PART V

Basics of Physical Interaction & Monte Carlo in Action

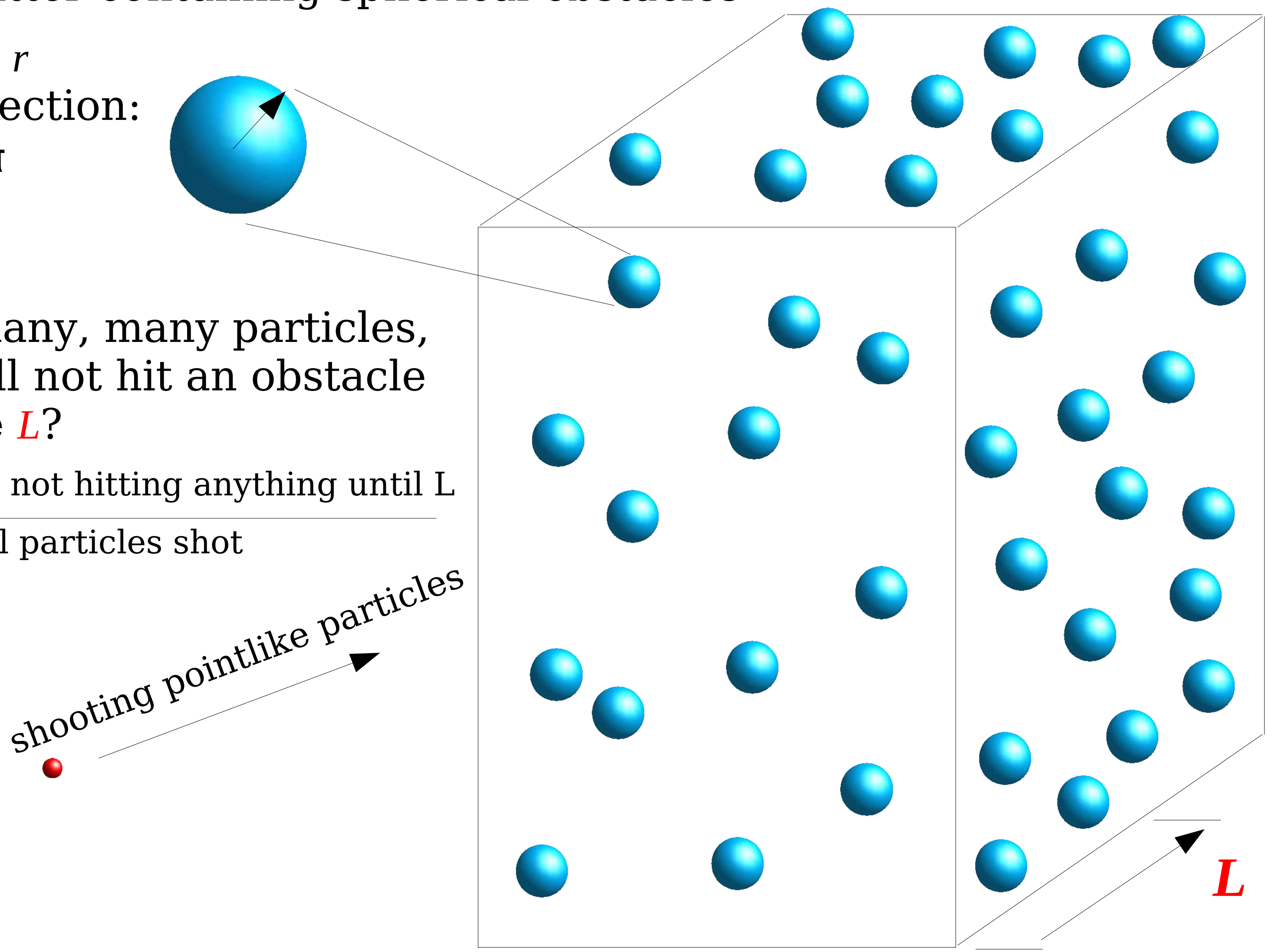
Tracking with Physics in Geant4

The Geant4 Physics Model

Monte Carlo in Geant4 – for Pedestrians

Piece of matter containing spherical obstacles

radius: r
cross section:
$\sigma = r^2\pi$



If we shoot many, many particles, how many will not hit an obstacle until distance L?

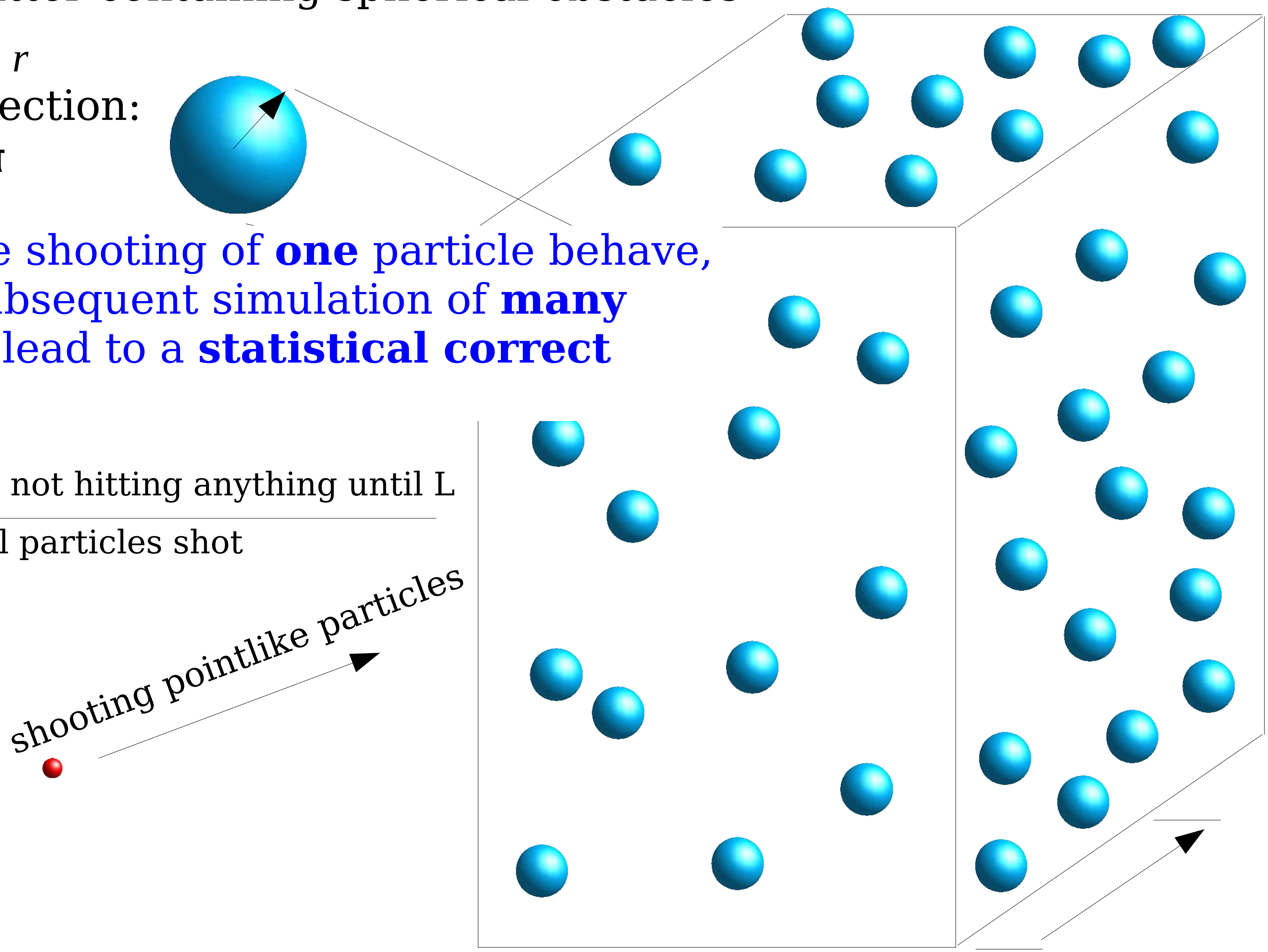
$$P(L) := \frac{\text{particles not hitting anything until L}}{\text{all particles shot}}$$

$P(L)$ ... probablility that one particle will go undisturbed until L

Monte Carlo in Geant4 – for Pedestrians

Piece of matter containing spherical obstacles

radius: r
cross section:
$\sigma = r^2\pi$



How must the shooting of **one** particle behave, so that the subsequent simulation of **many** particles will lead to a **statistical correct** behaviour?

$$P(L) := \frac{\text{particles not hitting anything until L}}{\text{all particles shot}}$$

$P(L)$ … probablility that one particle will go undisturbed until L

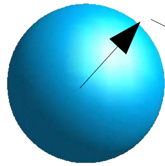
Monte Carlo in Geant4 – for Pedestrians

Piece of matter containing spherical obstacles

radius: r

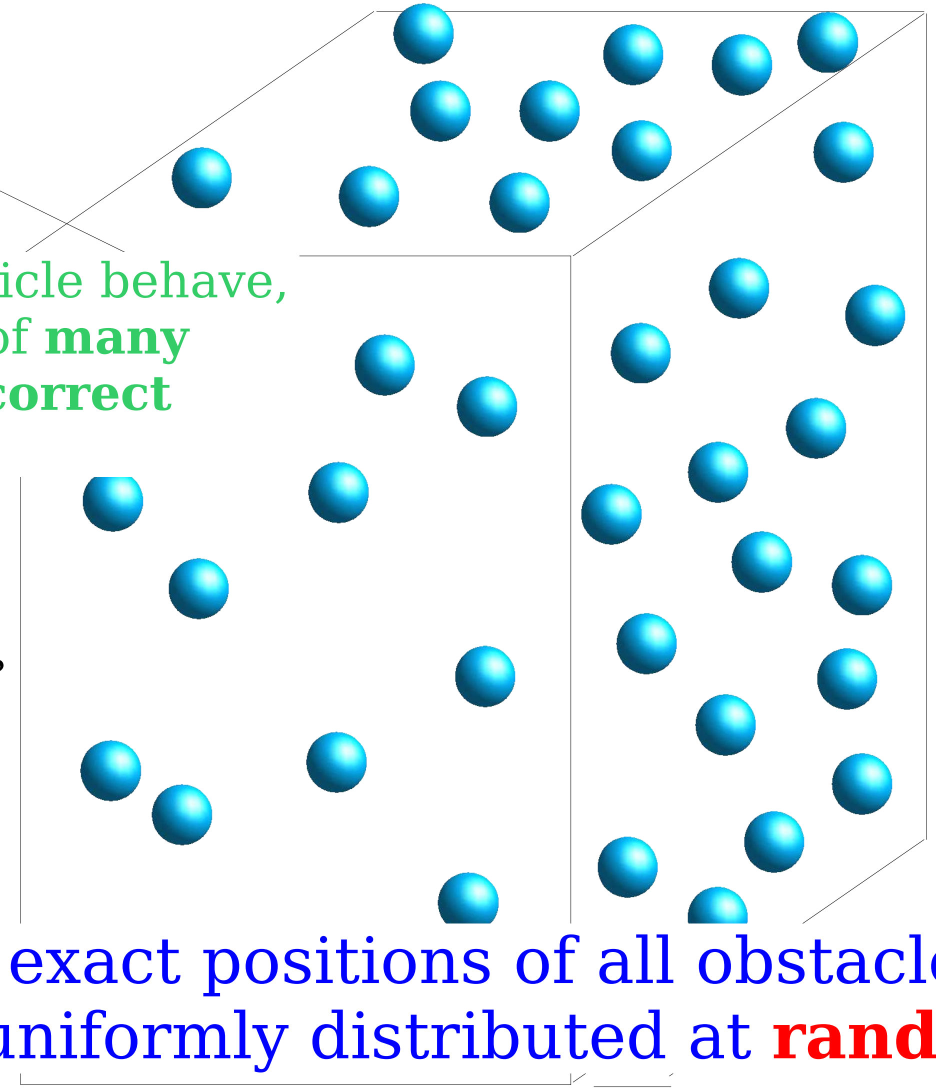
cross section:

$\sigma = r^2\pi$



How must the shooting of **one** particle behave, so that the subsequent simulation of **many** particles will lead to a **statistical correct** behaviour?

$$P(L) := \frac{\text{particles not hitting anything until L}}{\text{all particles shot}}$$



But we don't know the exact positions of all obstacles!
So we assume they are uniformly distributed at **random**!

(Alternatively, you can also think of shooting the particles uniformly distributed over an area of incidence.)

Monte Carlo in Geant4 – for Pedestrians

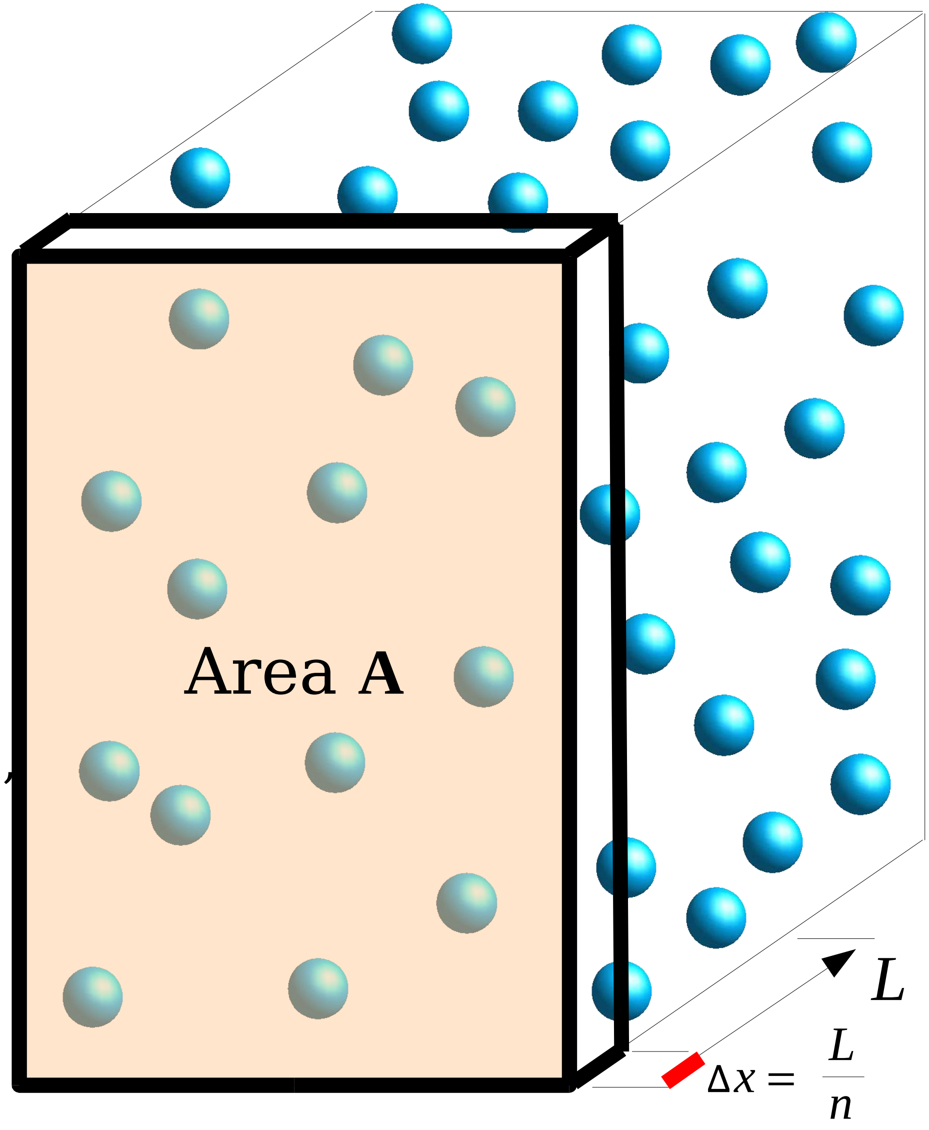
Piece of matter containing spherical obstacles

radius: r
cross section:
$\sigma = r^2\pi$

Subdivide the piece of material in thin slabs, each of thickness

$$\Delta x = \frac{L}{n}$$

If ρ denotes the density of obstacles, i.e. the no of obstacles per volume, then we have $\rho \cdot \Delta$ obstacles per area A (or $\rho \cdot A \cdot \Delta x$ obtacles in one thin slab)



Monte Carlo in Geant4 – for Pedestrians

Piece of matter containing spherical obstacles

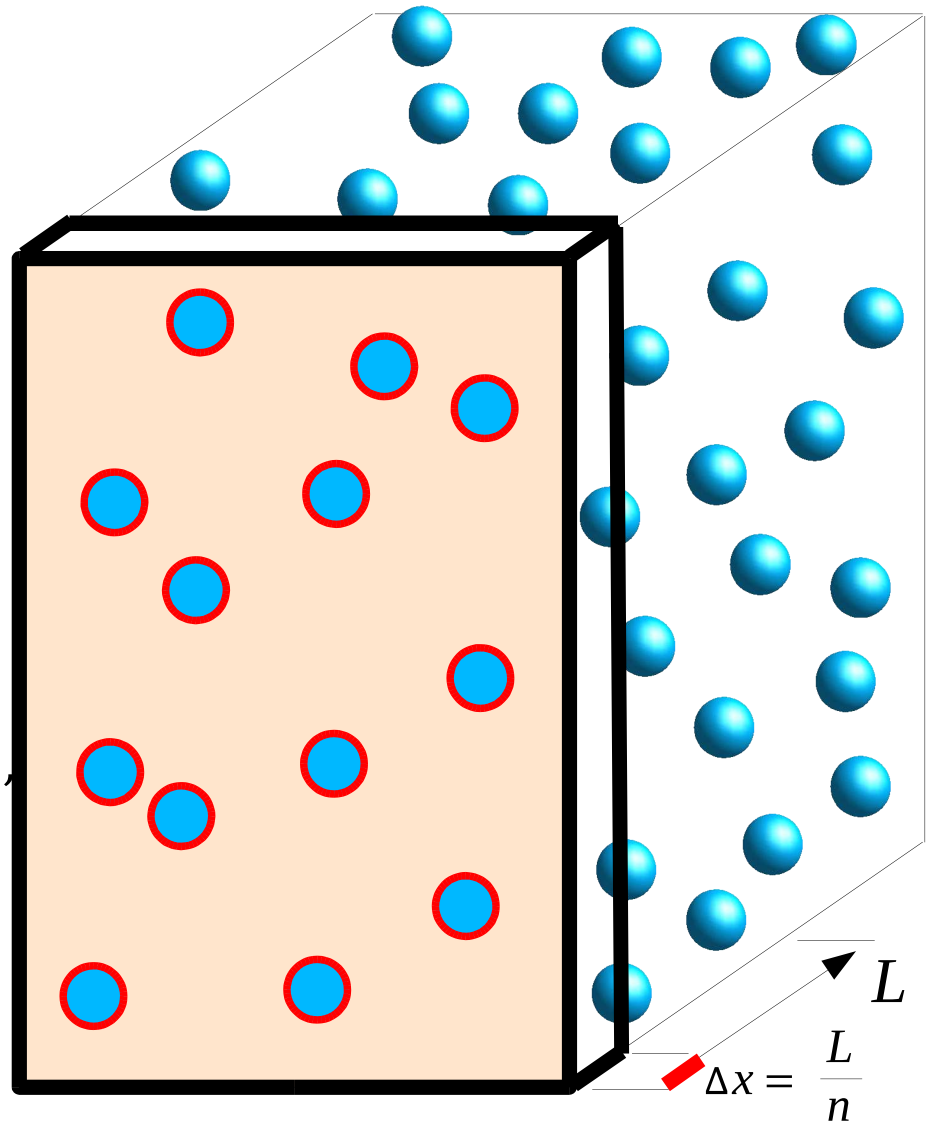
radius: r

cross section:

$$\sigma = r^2\pi$$

Subdivide the piece of material in thin slabs, each of thickness

$$\Delta x = \frac{n}{L}$$



If ρ denotes the density of obstacles, i.e. the no of obstacles per volume, then we have $\rho \cdot \Delta$ obstacles per area A (or $\rho \cdot A \cdot \Delta x$ obtacles in one thin slab)

The area covered by all obstacles in this slab is: $\rho \cdot A \cdot \sigma \cdot \Delta x$

Monte Carlo in Geant4 – for Pedestrians

The probability that one particle will hit an obstacle in the thin slab is then:

$$w' \cdot \Delta x = \frac{\text{Area covered by obstacles}}{\text{Area of incidence}}$$

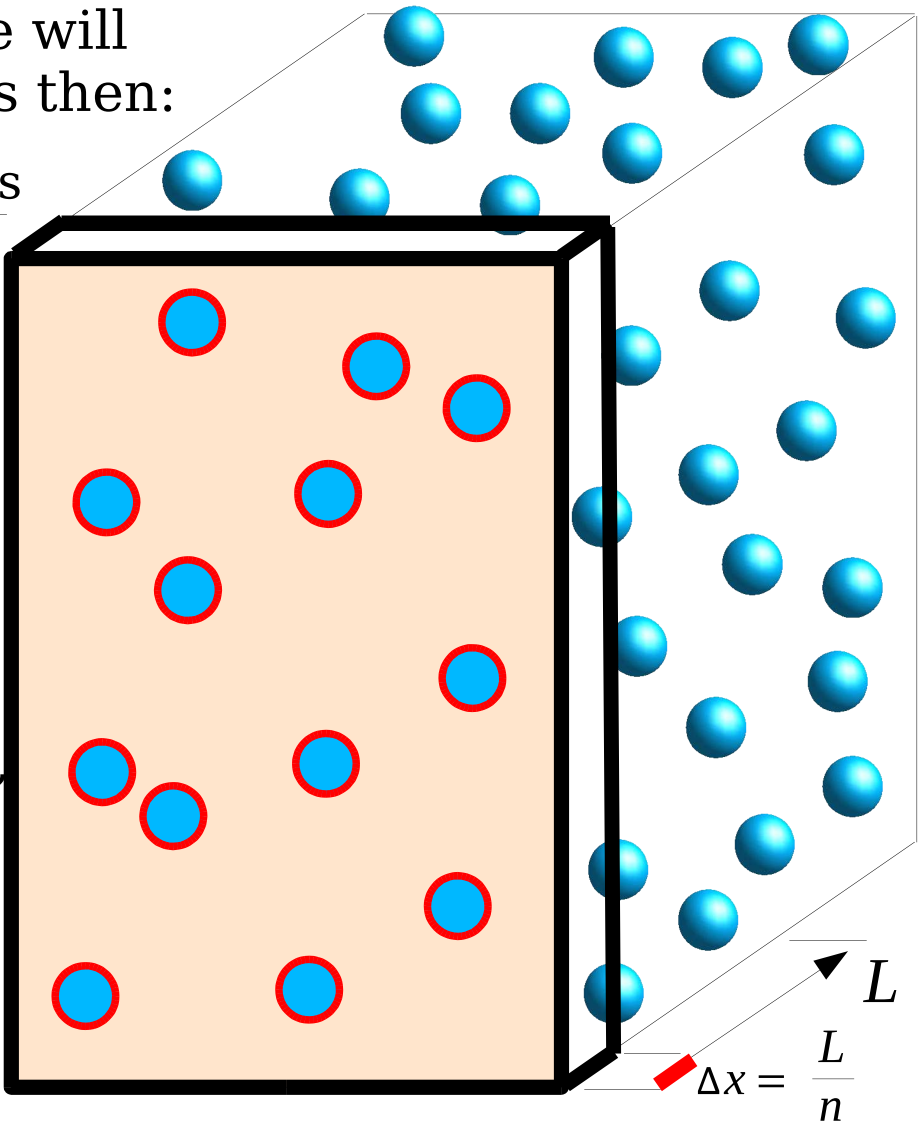
$$= \rho \cdot \sigma \cdot \Delta x$$

Thus, the prob. that the particle won't hit any obstacle:

$$w \cdot \Delta x = 1 - w'\Delta x = 1 - \rho \cdot \sigma \cdot \Delta x$$

If ρ denotes the density of obstacles, i.e. the no of obstacles per volume, then we have $\rho \cdot \Delta$ obstacles per area A (or $\rho \cdot A \cdot \Delta x$ obtacles in one thin slab)

The area covered by all obstacles in this slab is: $\rho \cdot A \cdot \sigma \cdot \Delta x$



Monte Carlo in Geant4 – for Pedestrians

$0 \leq$ $w \cdot \Delta x = 1 - w' \Delta x = 1 - \rho \cdot \sigma \cdot \Delta x$ ≤ 1 w...probability density fuction for $\Delta x \rightarrow dx$

Probability that a particle will pass a distance Δx without hitting anything

In principle, we could start a Monte Carlo simulation right now:

(a) determine $w \cdot \Delta x$

(b) draw a **random** number z from a **flat distribution** over (0,1)

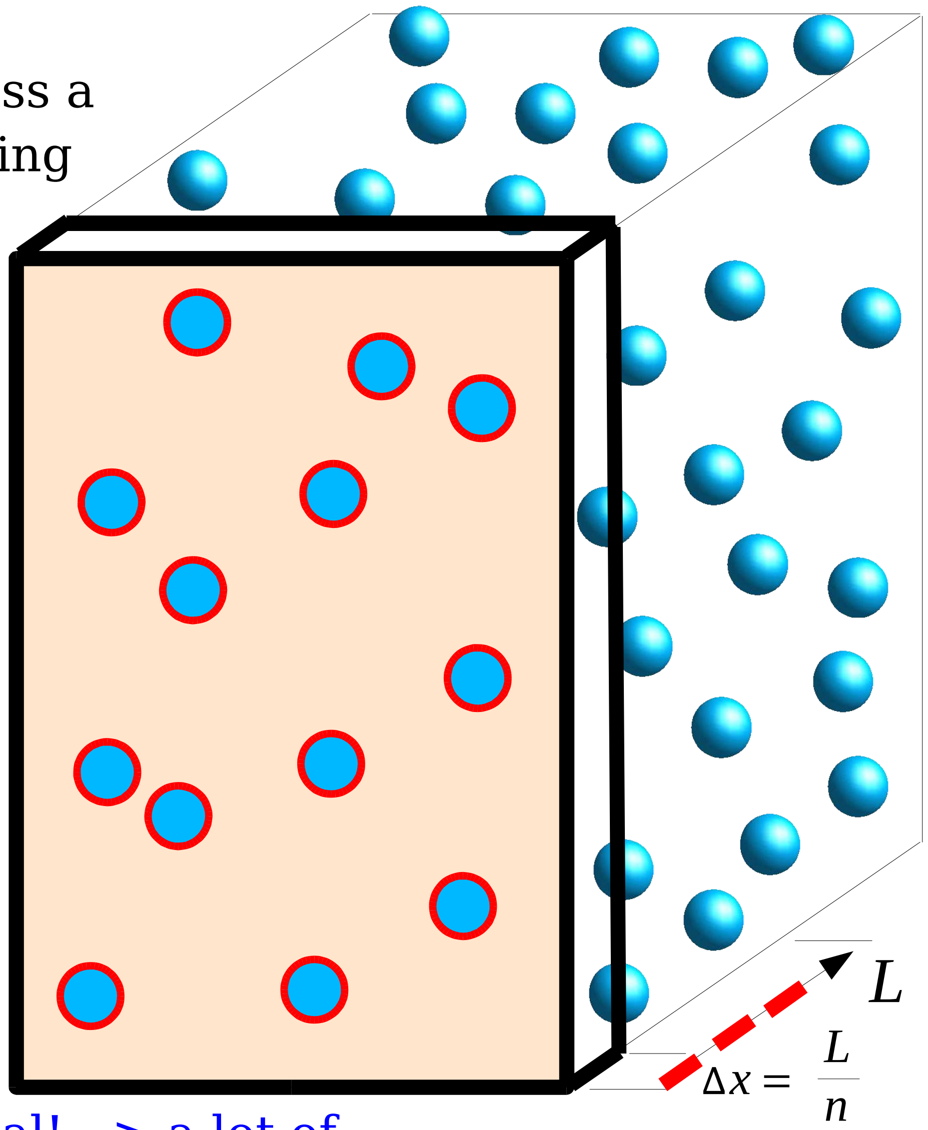
(c) if $z < w \cdot \Delta x$:

propagate particle by Δx

goto (b)

else:

simulate the interaction



Not very feasible:

For our assumptions to be valid, Δ must be "small", i.e. in the scale of inter-atomic distances in a material! -> a lot of computing time, only slow propagation of the particle!!

Monte Carlo in Geant4 – for Pedestrians

$$w \cdot \Delta x = 1 - \rho \cdot \sigma \cdot \Delta x$$

Probability that a particle will pass a distance Δ without hitting anything

$P(L)$ … probablility that one particle will go undisturbed until L

$$L = n \cdot \Delta x$$

$$P(0) = 1$$

$$P(\Delta x) = w \cdot \Delta x$$

$$P(2\Delta x) = (w \cdot \Delta x) \cdot (w \cdot \Delta x)$$

…

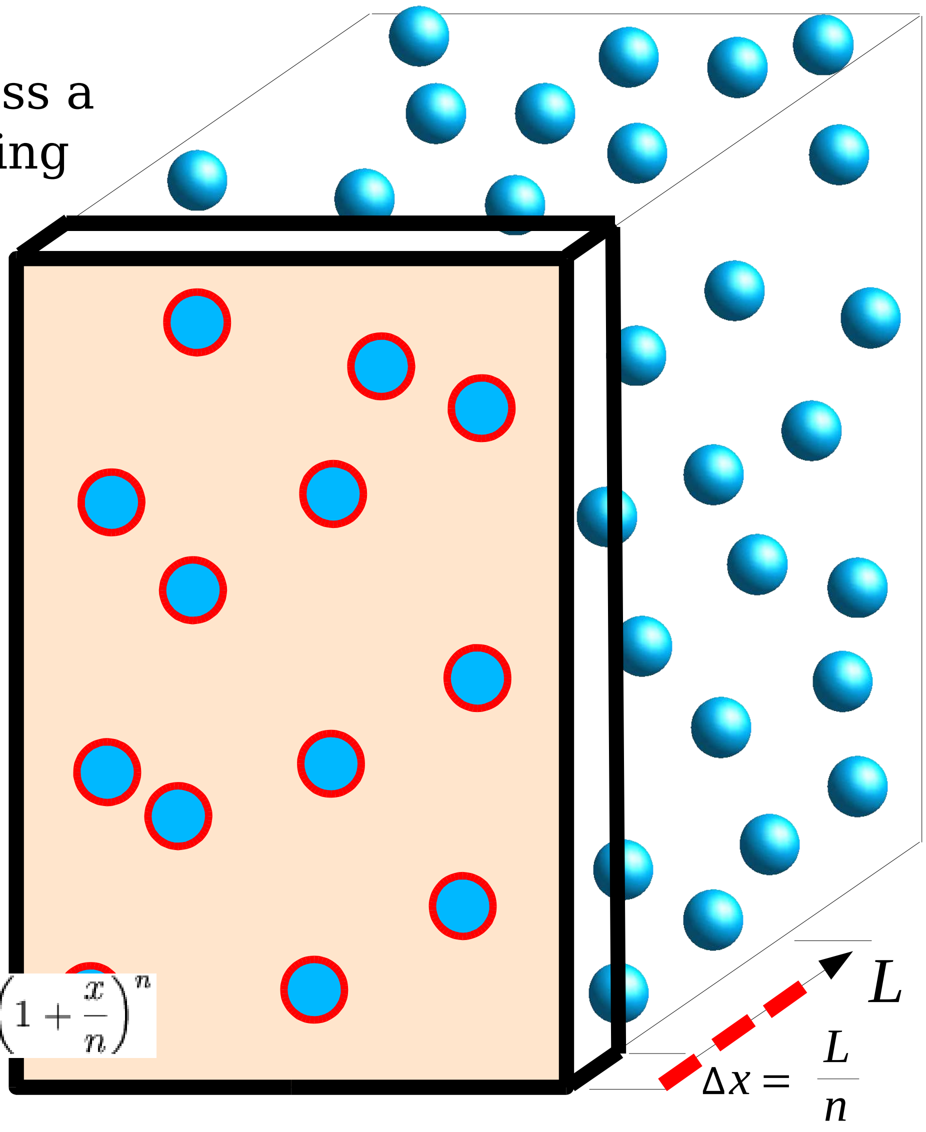
$$P(L{=}n \cdot \Delta x) = (w \cdot \Delta x)^n$$

$$P(L) = (1 - \rho \cdot \sigma \cdot \Delta x)^n$$

$$= \left(1 - \rho \cdot \sigma \frac{L}{n}\right)^n$$

$$\sim \exp(-\rho \cdot \sigma \cdot L)$$

$$\exp(x) = \lim_{n\to\infty}\left(1 + \frac{x}{n}\right)^n$$



Monte Carlo in Geant4 – for Pedestrians

$$w \cdot \Delta x = 1 - \rho \cdot \sigma \cdot \Delta x$$

Probability that a particle will pass a distance Δ without hitting anything

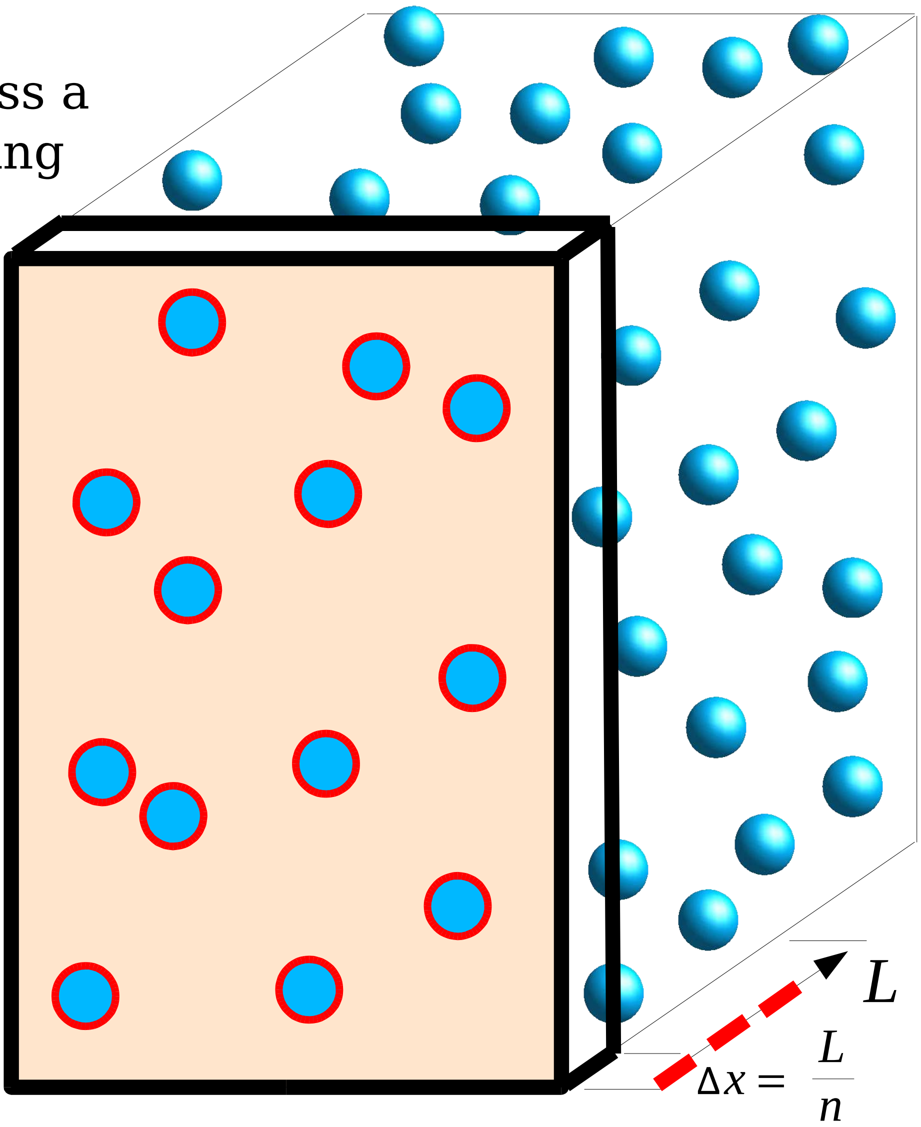
$$P(L) = \exp(-L/\lambda)$$

Probablility that one particle will go undisturbed until L

Between L and $L + \Delta x$:

$$w_{int}(L) \cdot \Delta x = P(L) \cdot w' \cdot \Delta x$$

$$P_{int}(L) = 1 - \exp(-L/\lambda)$$



Probability that one particle will have an interaction after traveling undisturbed the distance L

Monte Carlo in Geant4 – for Pedestrians

ρ density of obstacles (no of obstacles per volume)
σ cross section of an obstacle (interaction cross section)
$\lambda := 1/(\sigma \cdot \rho)$ mean free path length – average distance of undisturbed motion

$$w \cdot \Delta x = 1 - \rho \cdot \sigma \cdot \Delta x$$

Probability that a particle will pass a distance Δx without hitting anything

$$P_{int}(L) = 1 - \exp(- L/\lambda)$$

Probablility that a particle will have an interaction after traveling undisturbed the distance L

Instead of doing the Monte Carlo based on $w \cdot \Delta x$ to find out how far a particle will travel undisturbed, we can do the Monte Carlo based on $P_{int}(L)$ directly!

(a) draw a **random** number x from
$P_{int}(L) = 1 \quad \exp(\quad L/\lambda), L \in (0,\infty)$
(b) propagate the particle by x
(c) simulate the interaction

Instead of propagating the particle in (undisturbed) little steps of Δx, we can “boost” it directly by x

Monte C

ρ density of obsta
σ cross section of
$\lambda := 1/(\sigma \cdot \rho)$ mea ption

$$w \cdot \Delta x = 1 - \rho \cdot \sigma \cdot \Delta x$$

$$P_{int}(L) = 1 - \exp(- L/\lambda)$$

Instead of doing t
particle will trave
on $P_{int}(L)$ directly!

Note, that the Monte Carlo method for determining the undisturbed path length of a particle is justified because of the **randomness of the distribution of the obstacles!**

The functional form of the distribution stems from this randomness and not from the quantum mechanical character of the interaction itself!

(a) draw a **random** number x from
$P_{int}(L) = 1 \quad \exp(\quad L/\lambda), L \in (0,\infty)$
(b) propagate the particle by x
(c) simulate the interaction

Instead of propagating the particle in (undisturbed) little steps of Δx, we can "boost" it directly by x

Uff!

Monte Carlo in Geant4

The simple model shown is in fact what is happening for a given type of physical process – just that the cross section has not a geometrical meaning in the literal sense:

probability that a particle will hit an obstacle in the thin slab: $w' \cdot \Delta x = w_{interact} = \rho \cdot \sigma \cdot \Delta x$

$$\sigma = \frac{w_{interact}}{\rho \cdot \Delta x} = \frac{w_{interact}}{\text{areal density}} =$$ measure of the interaction strength of a single obstacle shot at by a particle of a specific type

Total cross section of the physics interaction:

$\sigma = \sigma$(particle-type, obstacle-type, energy, quark content, ..)

Here physics comes in, because σ is calculated based on quantum theoretical models! For us it's now only a number given by the "deus ex machina".

-> see Alberto's special lectures for many more details!!

Monte Carlo in Geant4

To determine how far a particle can travel undisturbed, Geant4 uses the same algorithm for each physical process which you want to participate in the simulation. Let's see:

Process 1: $P_{int,1}(L) = 1 - \exp(-L/\lambda_1)$

Process 2: $P_{int,2}(L) = 1 - \exp(-L/\lambda_2)$

$\lambda_n := 1/(\sigma_n \cdot \rho)$ (or appropriate weighted mean in case of non pure materials)

The density of "obstacles" is material dependent, and so is therefore the mean free path length – implying that the probability of undisturbed flight changes when the material changes: $\lambda_n = \lambda_{n,\,mat}$

Better use the probabilities of "number of undisturbed mean free path lengths" -> independent of material: $\boldsymbol{Y := L / \lambda}$, $L = \lambda\, Y$

Process 1: $Y_1 = -\ln(1 - \xi_1)$

Process 2: $Y_2 = -\ln(1 - \xi_2)$

ξ_n uniformly distributed in [0,1]

Monte Carlo in Geant4

To determine how far a particle can travel undisturbed, Geant4 uses the same algorithm for each physical process that you want to participate in the simulation. Let's see:

Process 1: $P_{int,1}(L) = 1 - \exp(-L/\lambda_1)$ $\lambda_n := 1/(\sigma_n \cdot \rho)$ or appropriate weighted mean in case of non pure materials

Process 2: $P_{int,2}(L) = 1 - \exp(-L/\lambda_2)$

The density of "obstacles" is material dependent, and so is therefore the mean free path length – implying that the probability of undisturbed flight changes when the material changes: $\lambda_n = \lambda_{n,\,mat}$

Better use the probabilities of "number of undisturbed mean free path lengths" -> independent of material: $\boldsymbol{Y := L / \lambda}$, $L = \lambda Y$

Process 1: $Y_1 = -\ln(1 - \xi_1)$

Process 2: $Y_2 = -\ln(1 - \xi_2)$

normalized exponential distributions, the same for every process!!! Sampling from inverse

ξ_n uniformly distributed in [0,1]

the **"dirty trick"**

Monte Carlo in Geant4

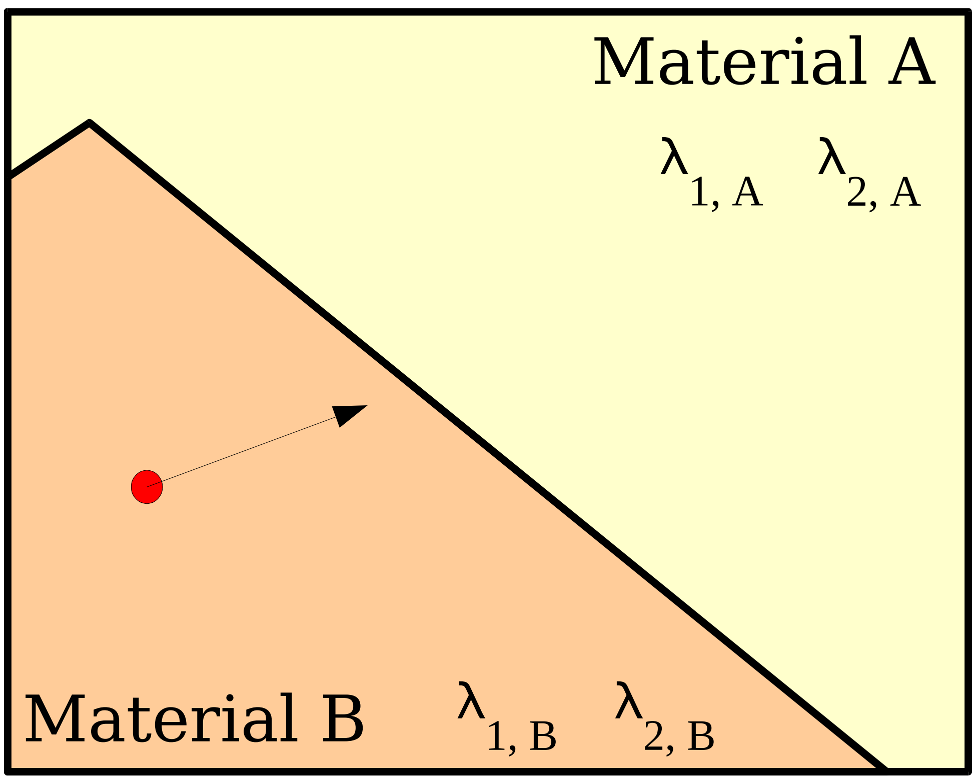
Process 1: $Y_1 = -\ln(1 - \xi_1)$

Process 2: $Y_2 = -\ln(1 - \xi_2)$

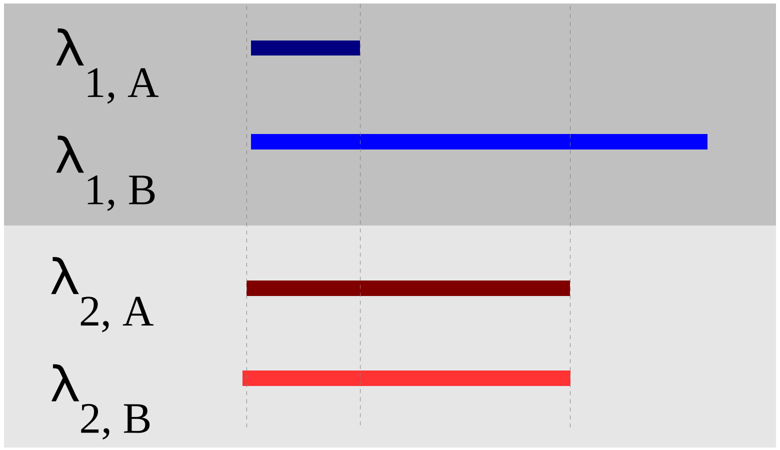
ξ_n uniformly distributed in [0,1]

Y	number of free path length – material, process, and energy independent!
$\lambda_n^n = \lambda_{n,\,mat}$	mean free path length of process n in material mat for a given energy
$L_{n,\,mat} = Y_n \lambda_{n,\,mat}$	undisturbed distance of particle subjected to process n in material mat

Sampling of the number of free path lengths



Geant4 “knows” all mean free path lengths. At initialization time, the physics processes calculate tables within their energy range for each material to determine $\lambda_{n,\,mat}$



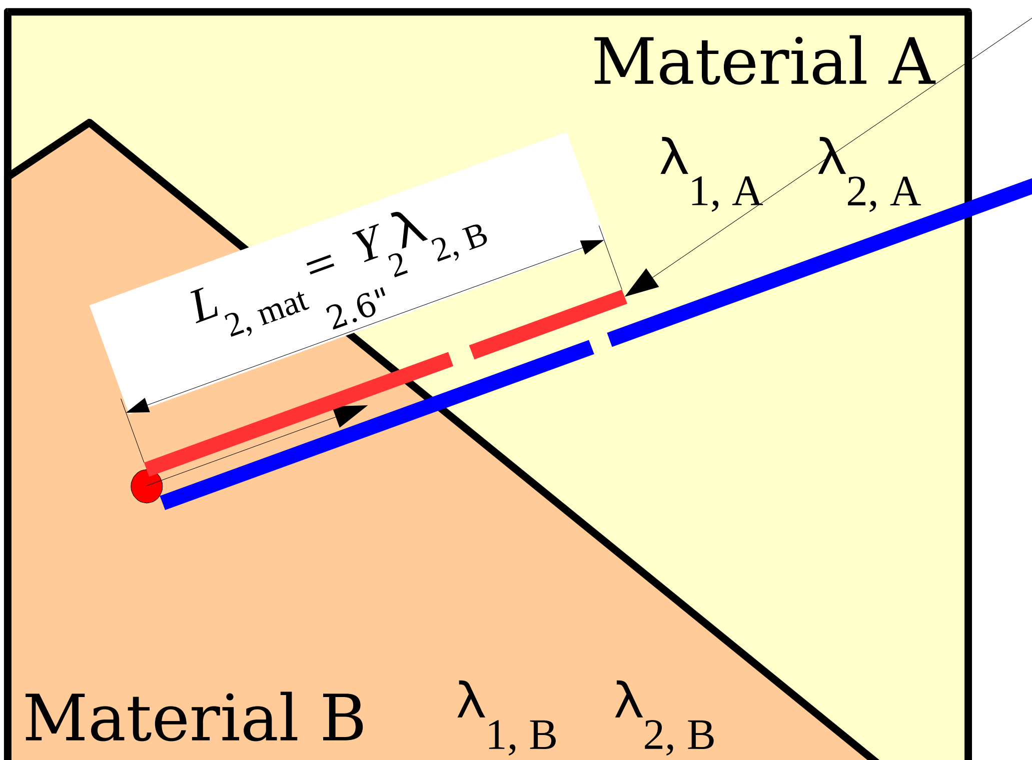
Monte Carlo in Geant4

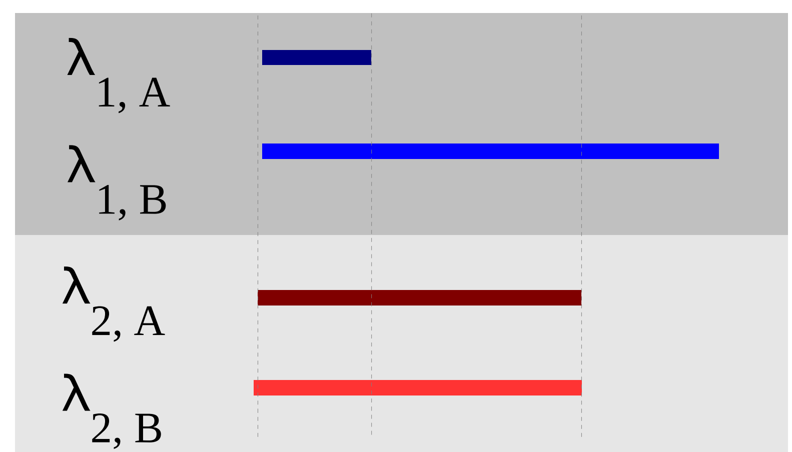
Process 1: $Y_1 = -\ln(1 - \xi_1)$ → **2** mean free path lengths

Process 2: $Y_2 = -\ln(1 - \xi_2)$ → **1.5** mean free path lengths

ξ_n uniformly distributed in [0,1]

looks like that process 2 has the shortest undisturbed path





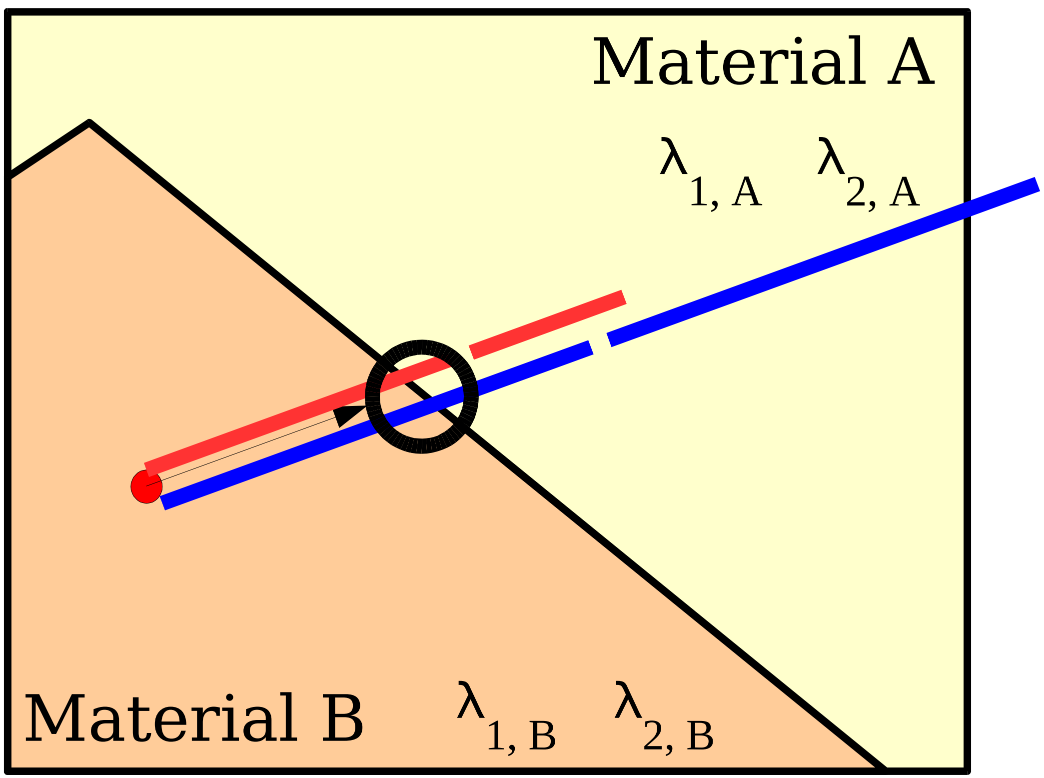
Monte Carlo in Geant4

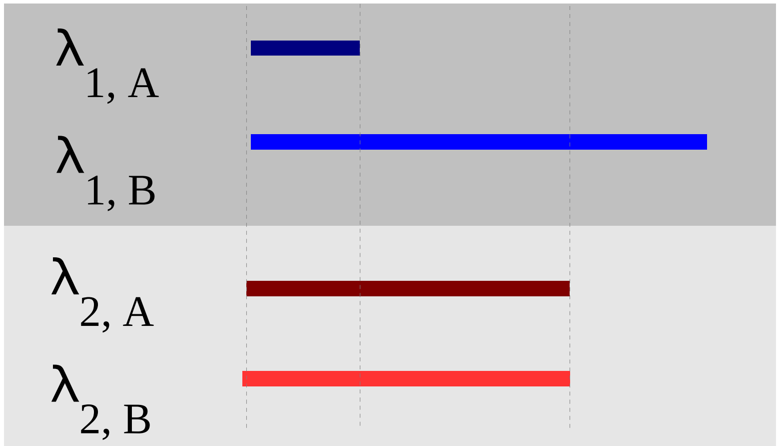
Process 1: $Y_1 = -\ln(1 - \xi_1)$ → **2**

Process 2: $Y_2 = -\ln(1 - \xi_2)$ → **1.5**

ξ_n uniformly distributed in [0,1]

but we have a boundary crossing and the material properties change,
thus also the mean free path lengths!!!
-> switch to the other set of mean free path lengths!



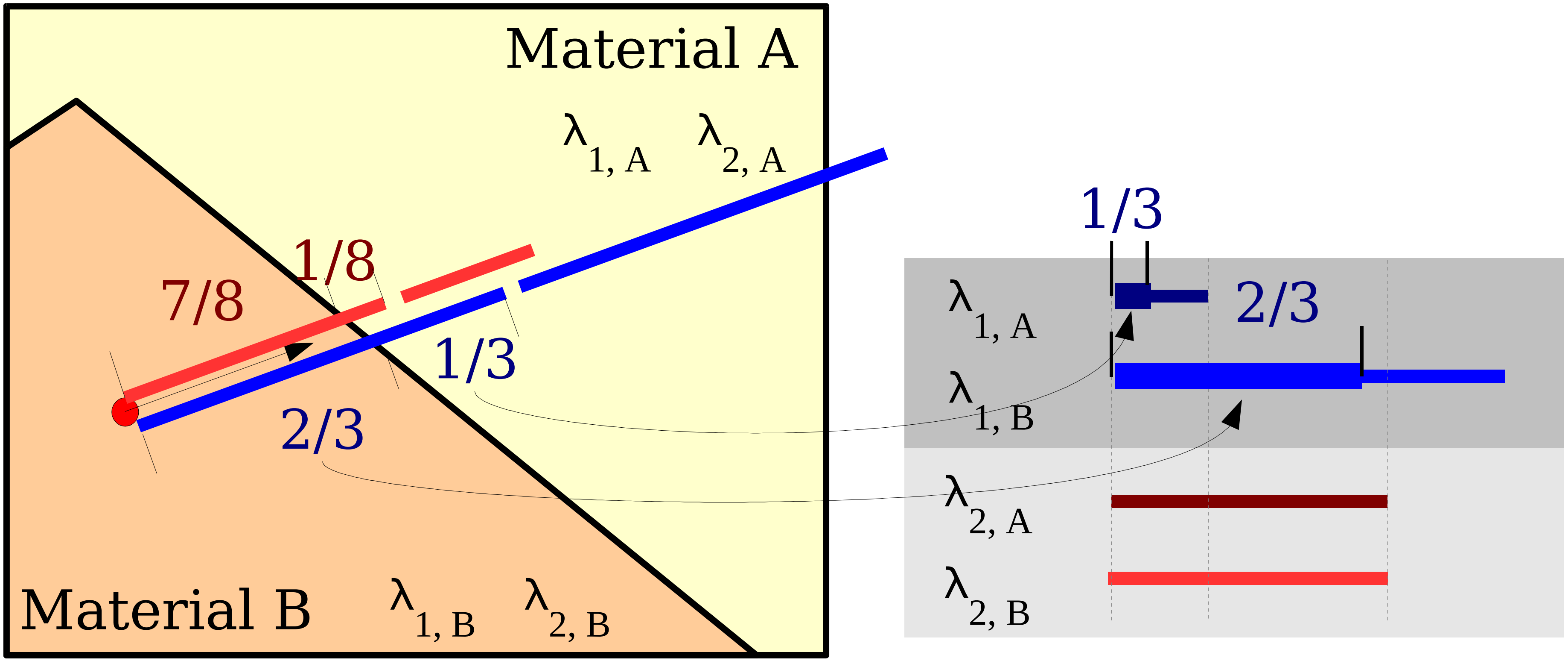


Monte Carlo in Geant4

Process 1: $Y_1 = -\ln(1 - \xi_1)$ → **2**

Process 2: $Y_2 = -\ln(1 - \xi_2)$ → **1.5**

ξ_n uniformly distributed in [0,1]

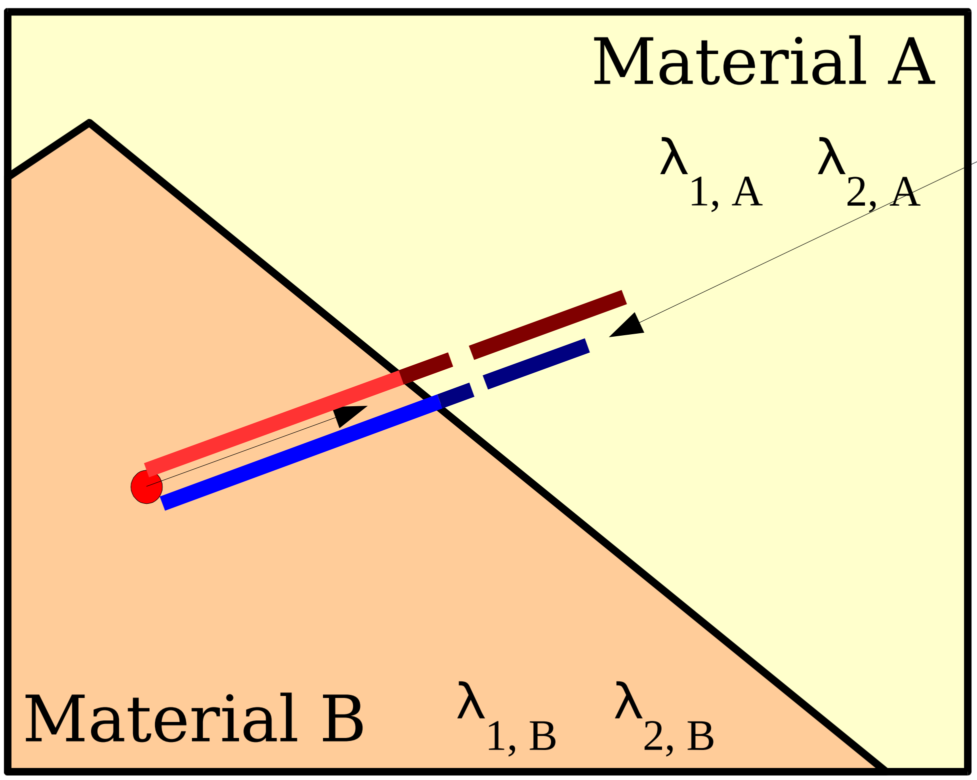


Monte Carlo in Geant4

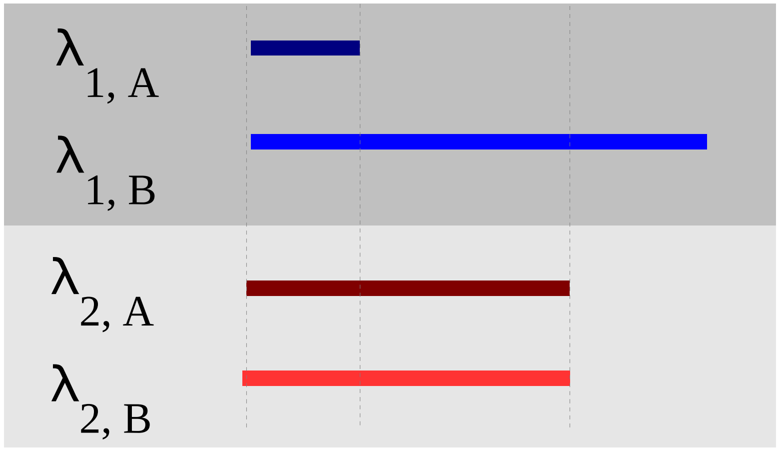
Process 1: $Y_1 = -\ln(1 - \xi_1)$ → **2**

Process 2: $Y_2 = -\ln(1 - \xi_2)$ → **1.5**

ξ_n uniformly distributed in [0,1]



Process 1 will disturb the particle first!

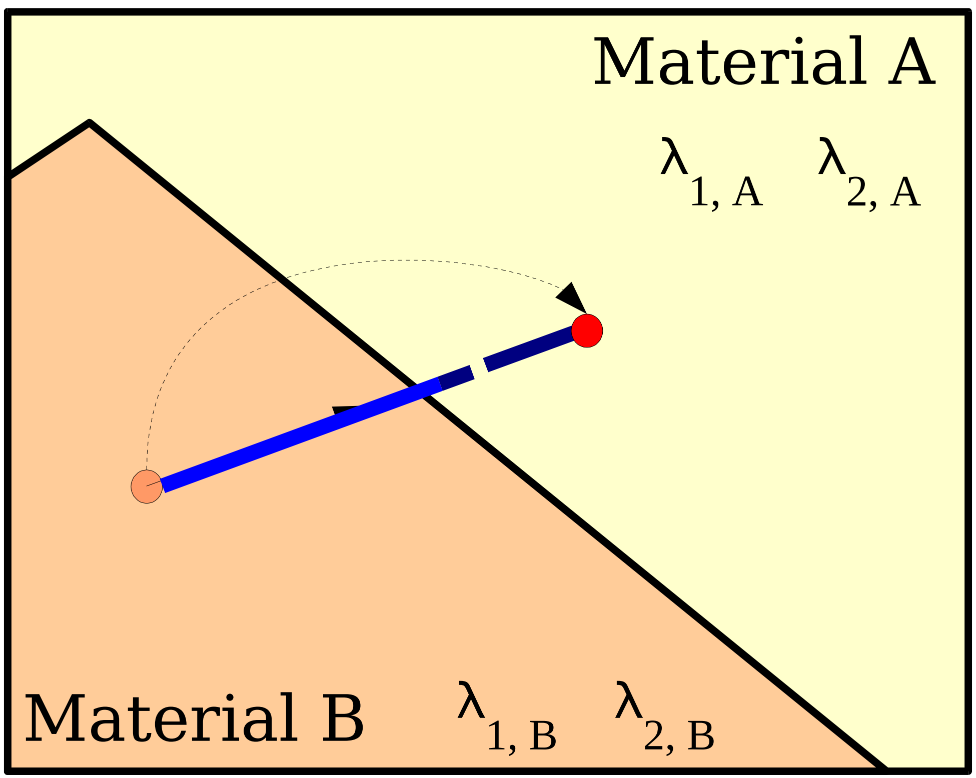


Monte Carlo in Geant4

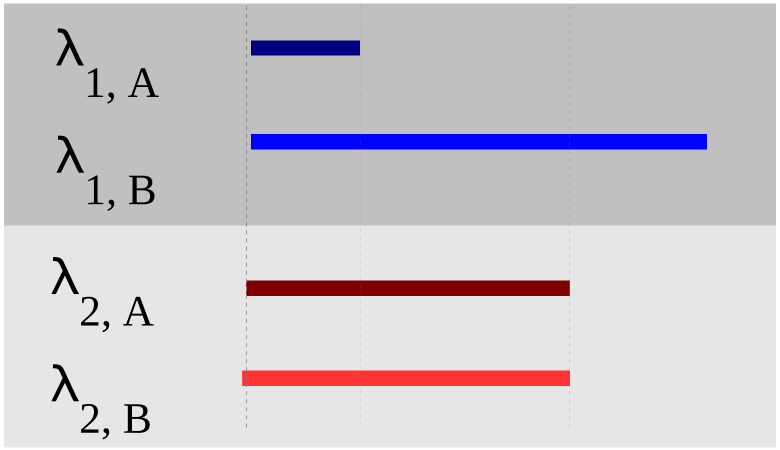
Process 1: $Y_1 = -\ln(1 - \xi_1)$ → **2**

Process 2: $Y_2 = -\ln(1 - \xi_2)$ → **1.5**

ξ_n uniformly distributed in [0,1]



propagate the particle undisturbed, then simulate the physics according to process 1

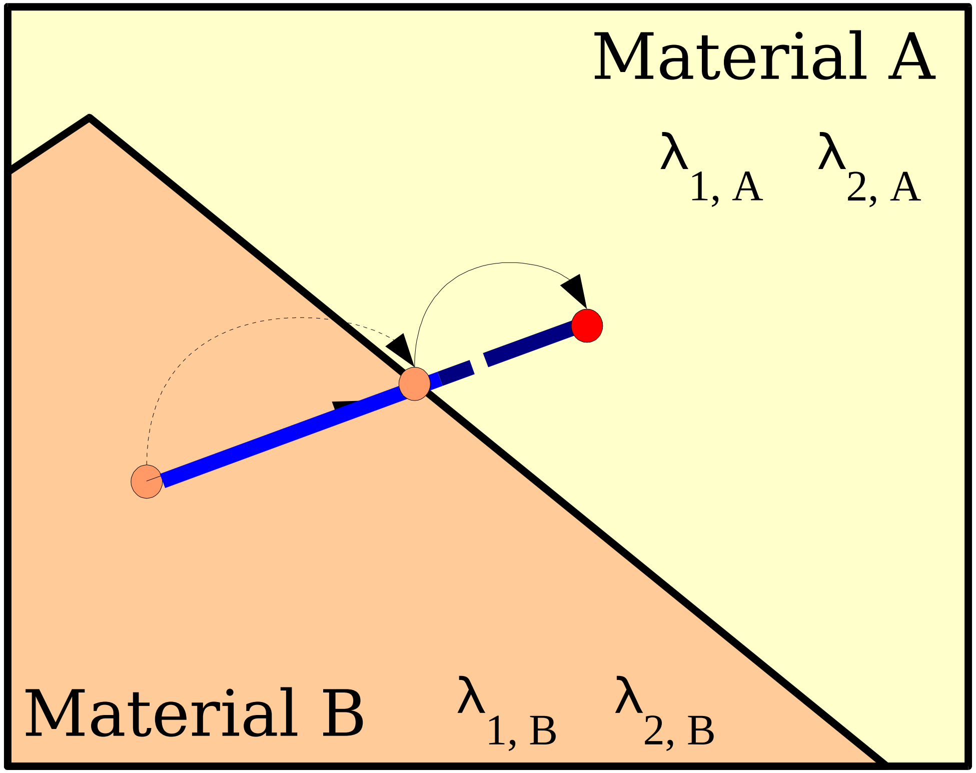


Monte Carlo in Geant4

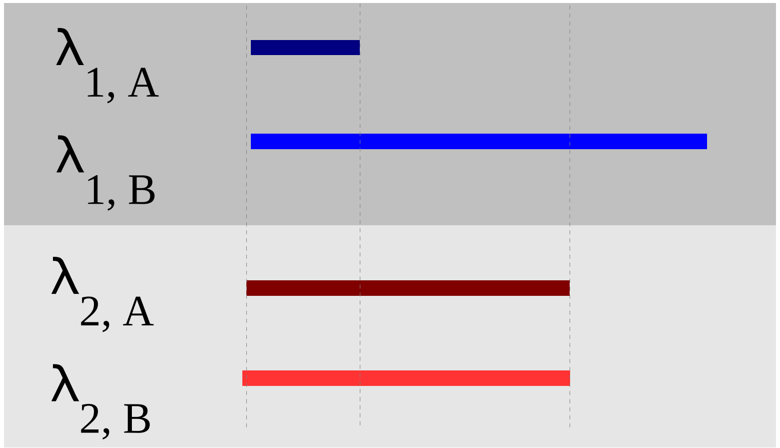
Process 1: $Y_1 = -\ln(1 - \xi_1)$ → **2**

Process 2: $Y_2 = -\ln(1 - \xi_2)$ → **1.5**

ξ_n uniformly distributed in [0,1]



Geant4 introduces an artificial intermediate step on the volume surface (user hook!)

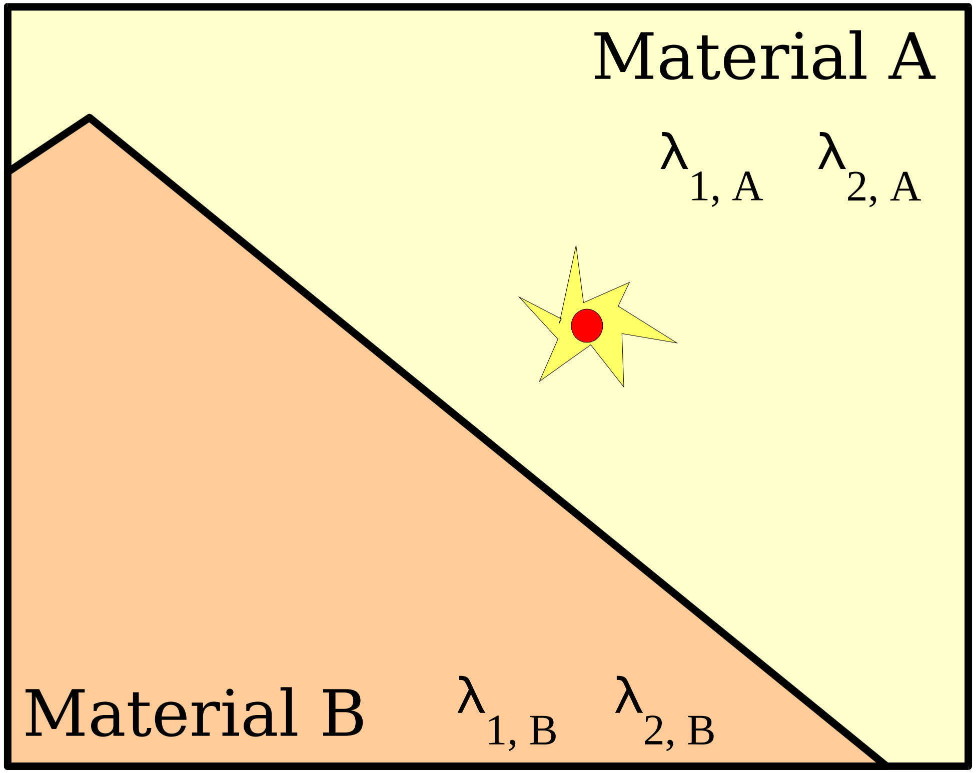


Monte Carlo in Geant4

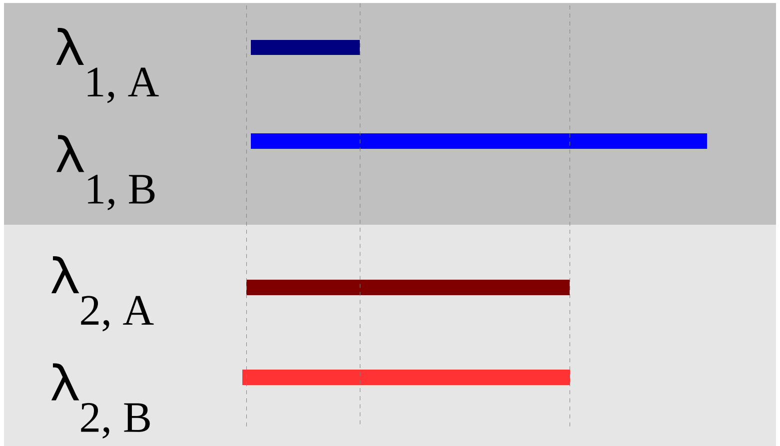
Process 1: $Y_1 = -\ln(1 - \xi_1)$ → **2**

Process 2: $Y_2 = -\ln(1 - \xi_2)$ → **1.5**

ξ_n uniformly distributed in [0,1]



Simulate the physics interaction of process 1 by applying the **Monte Carlo** method once more. Now the **randomness** stems from the **quantum character** of the interaction itself: **distribution of the final states** of the process



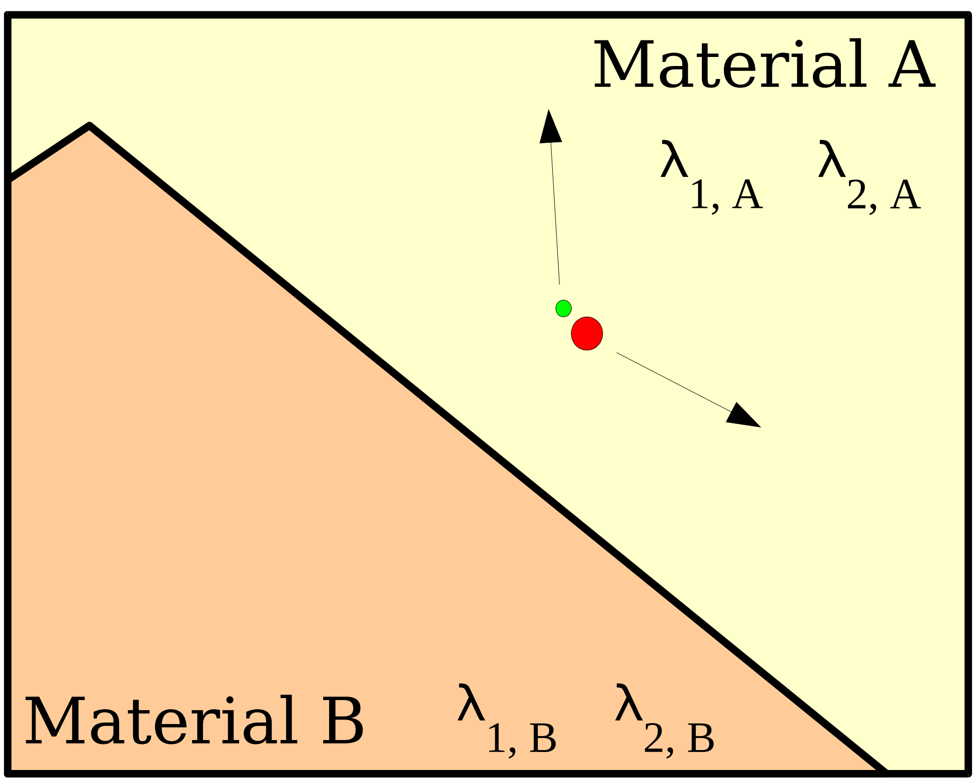
Monte Carlo in Geant4

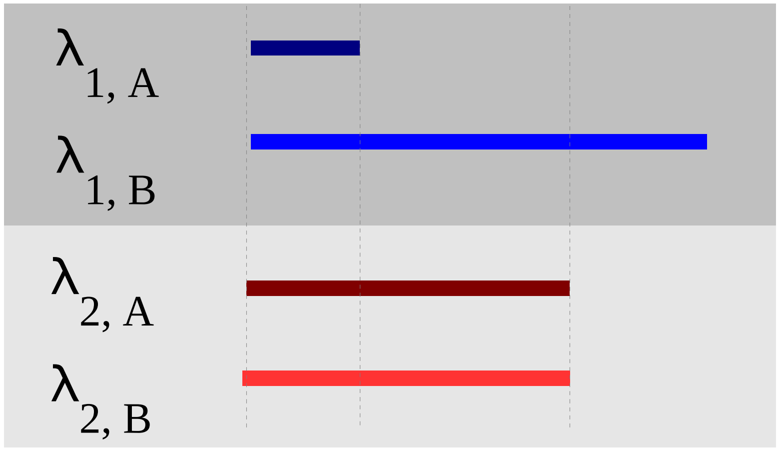
Process 1: $Y_1 = -\ln(1 - \xi_1)$ ⟶ **2**

Process 2: $Y_2 = -\ln(1 - \xi_2)$ ⟶ **1.5**

ξ_n uniformly distributed in [0,1]

depending on the physics process:
- sample energy loss
- sample momentum change
- sample production of new particles (put them on a stack for later simulation)
- ...



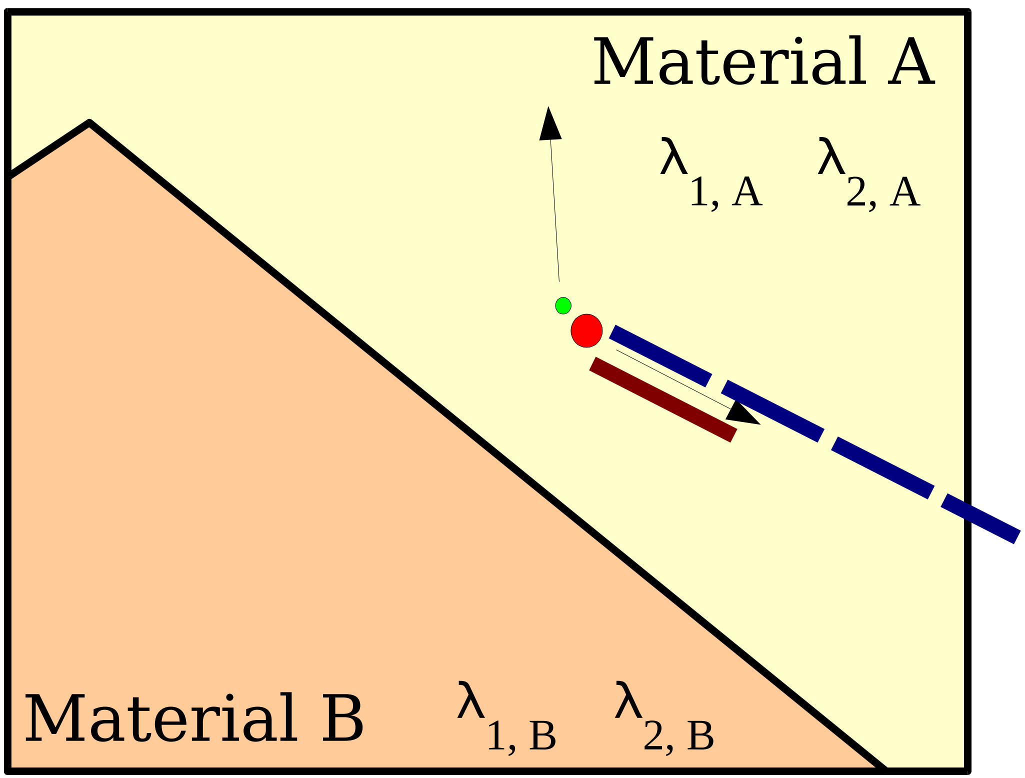


Monte Carlo in Geant4

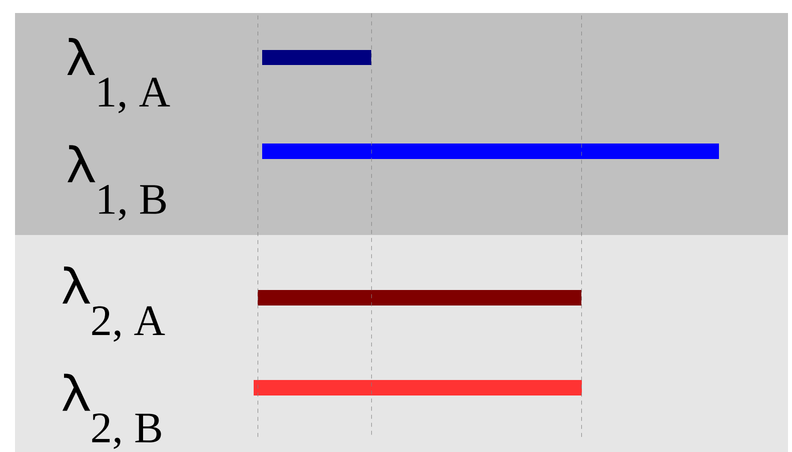
Process 1: $Y_1 = -\ln(1 - \xi_1)$ → **3.75**

Process 2: $Y_2 = -\ln(1 - \xi_2)$ → **0.5**

ξ_n uniformly distributed in [0,1]



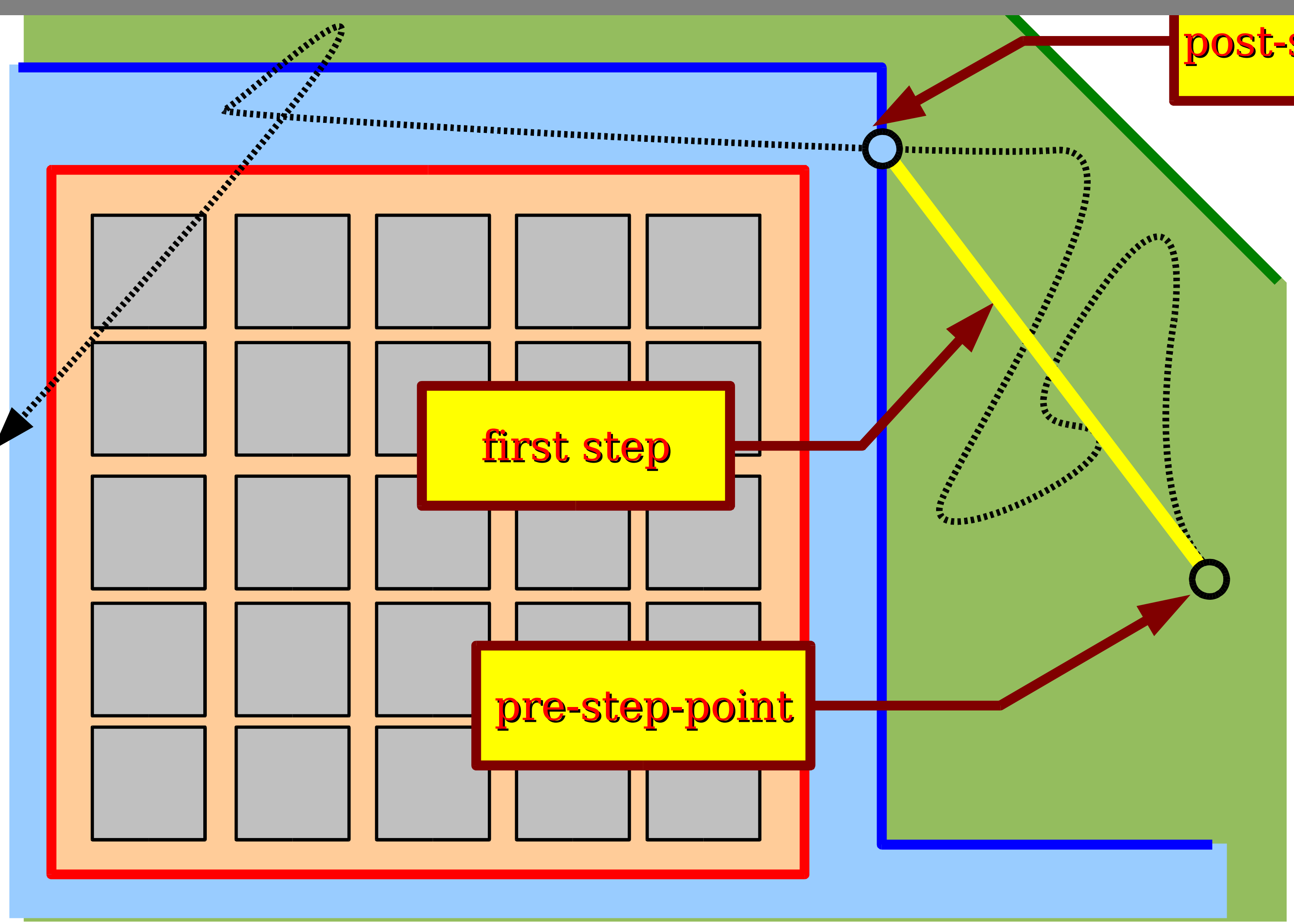
Start the game again!

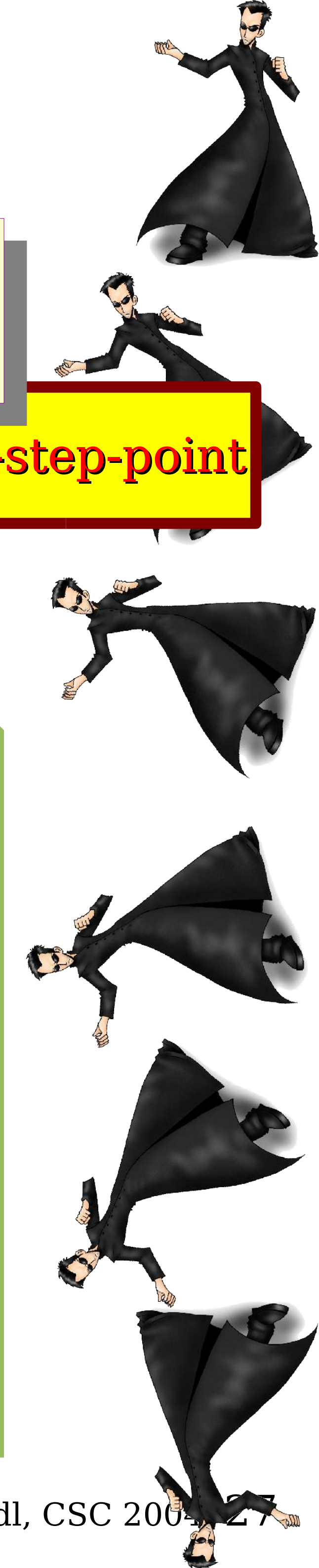


The Geant4 Stepdance

RELOADED

Reminder: without activated physics, geometry determines the free step length

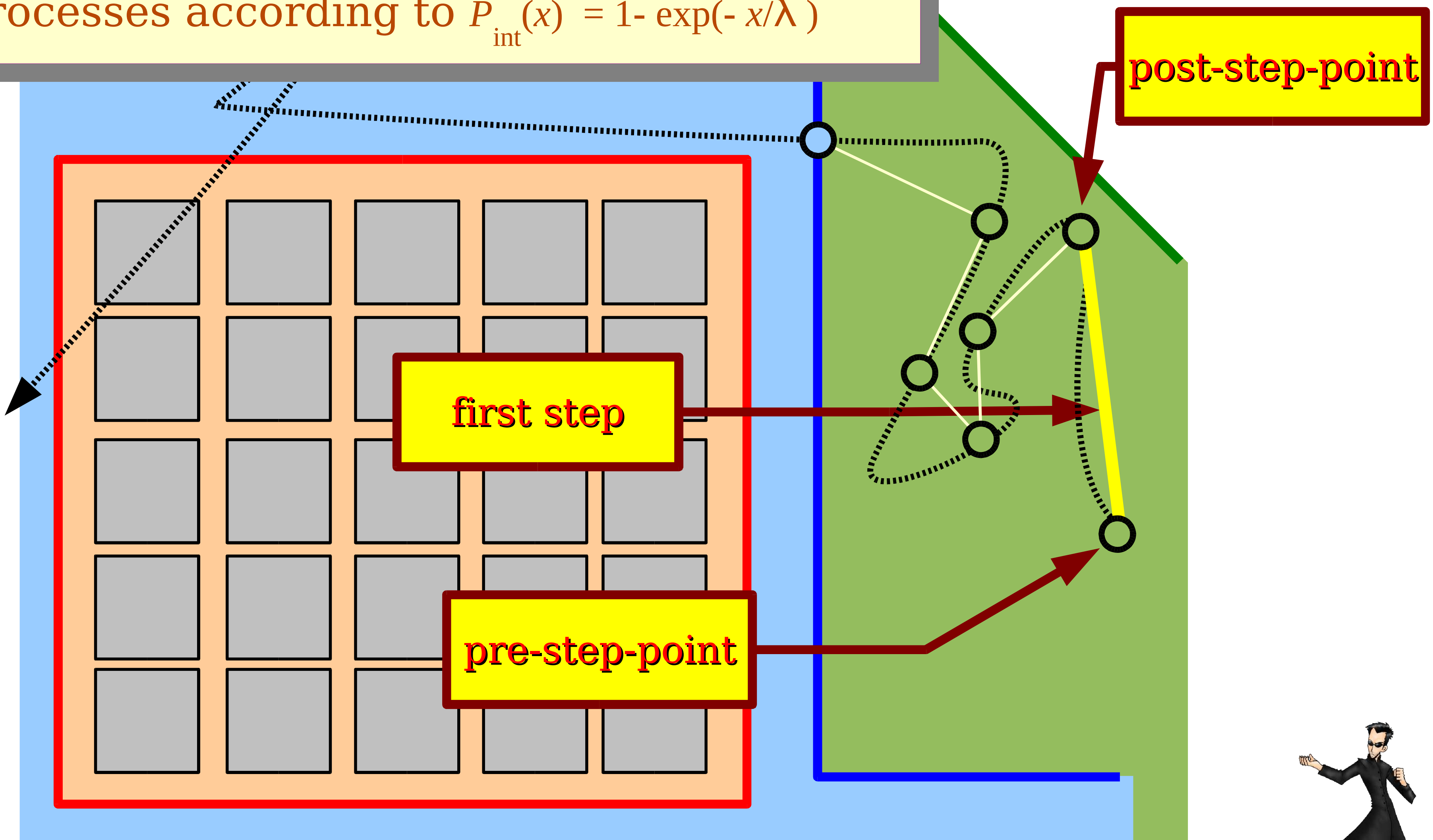




The Geant4 Stepdance

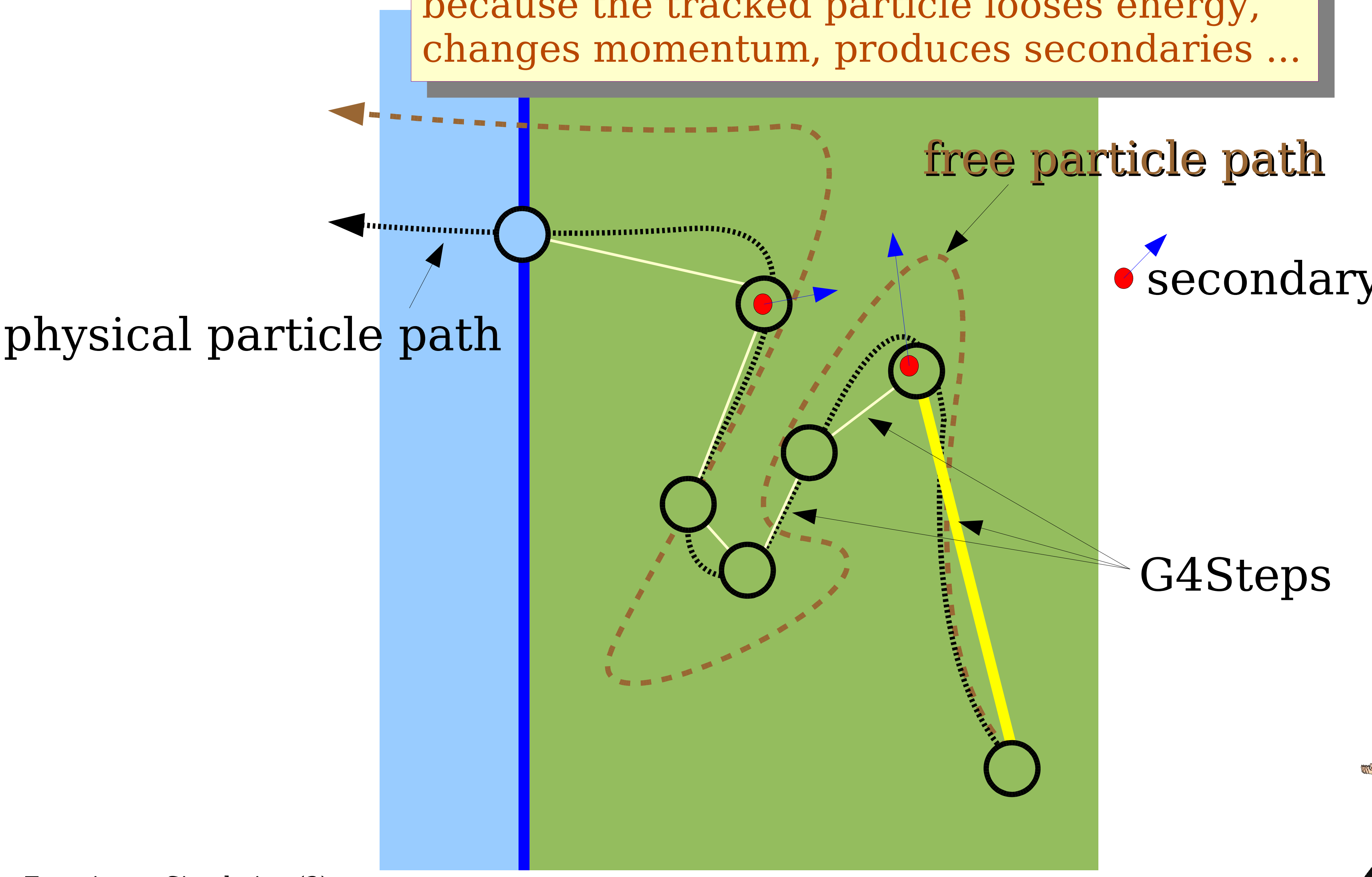
RELOADED

Physics: geometrical steps are sub-divided into "shortest free paths" of the activated processes according to $P_{int}(x) = 1 - \exp(-x/\lambda)$



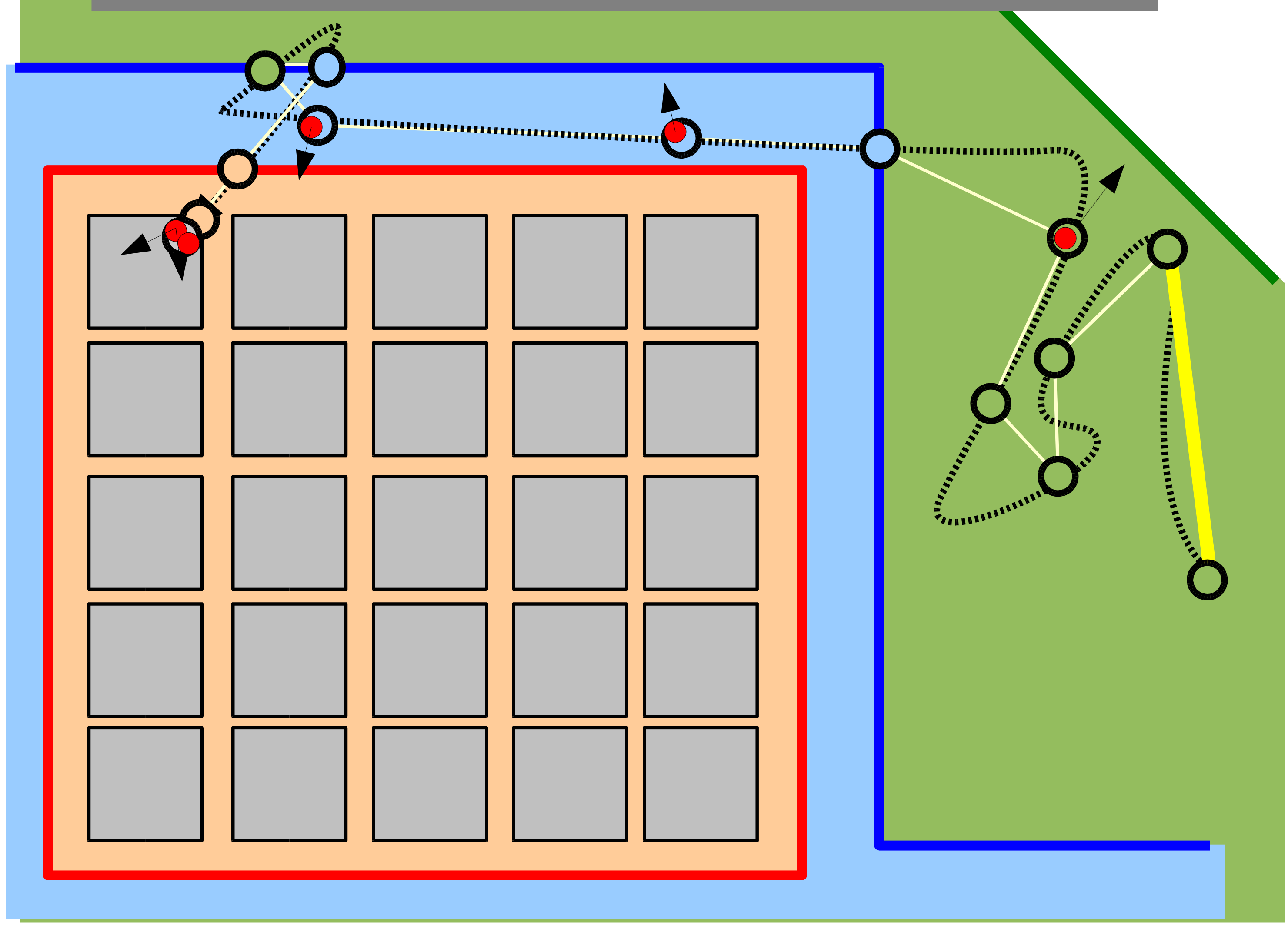
The Geant4 **Stepdance**

Physics: also the trajectory will be different because the tracked particle looses energy, changes momentum, produces secondaries ...



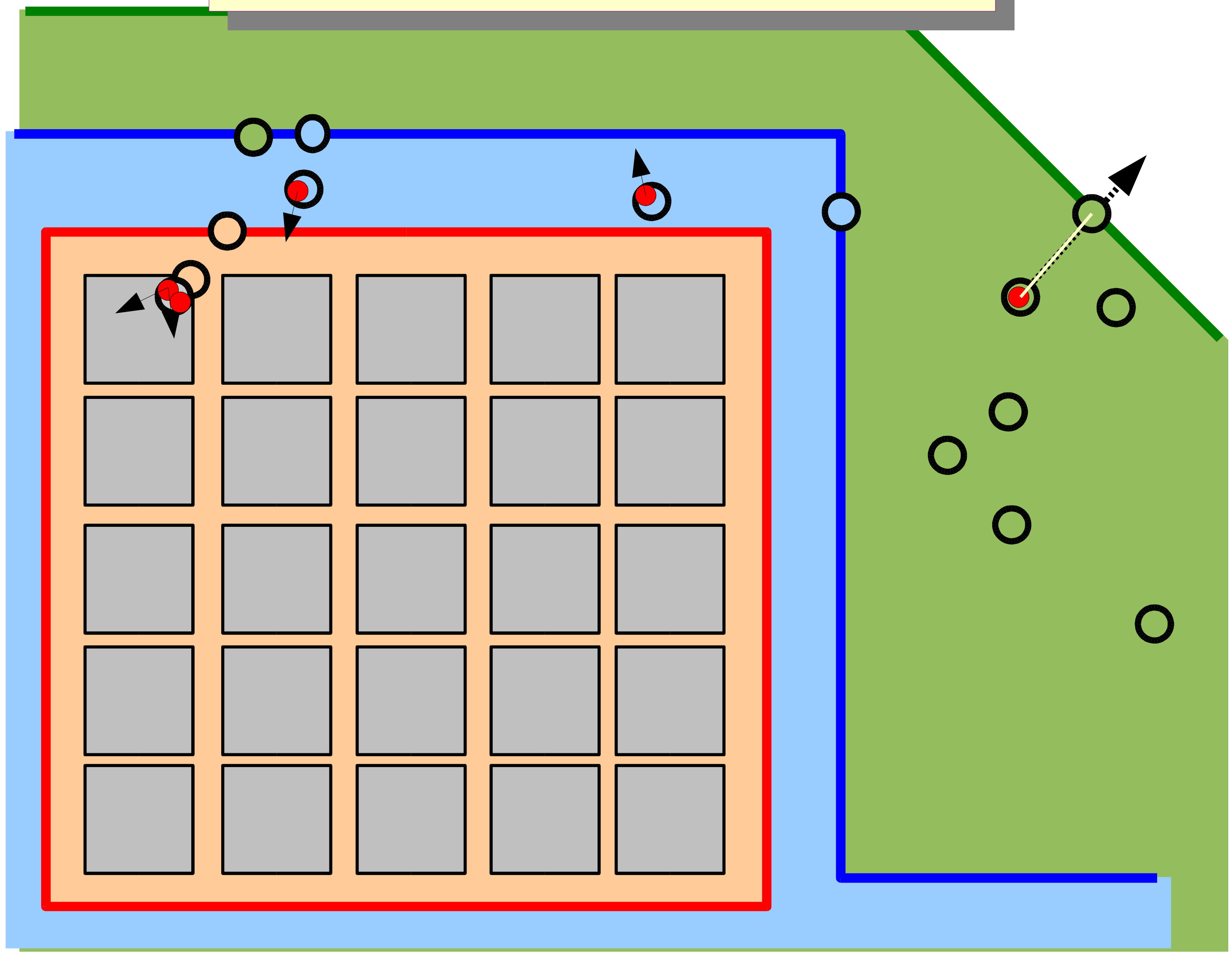
The Geant4 Stepdance

One particle is tracked until it either decays, annihilates, gets absorbed, or leaves the world volume.



The Geant4 Stepdance

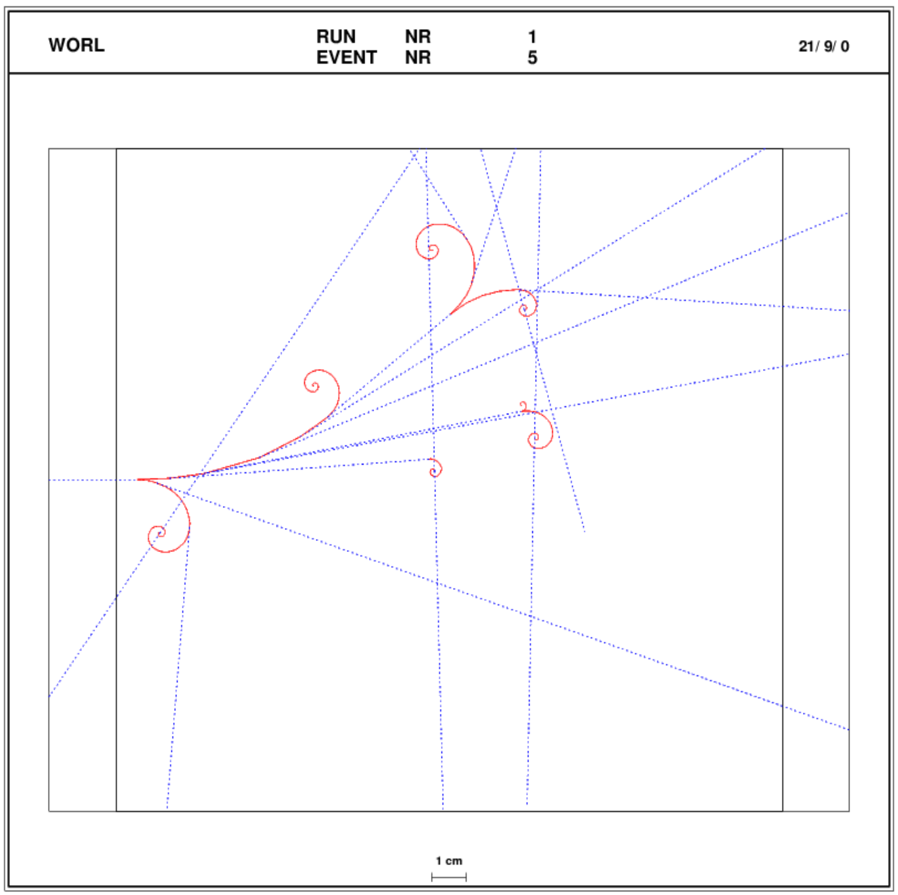
Then the secondaries are tracked.

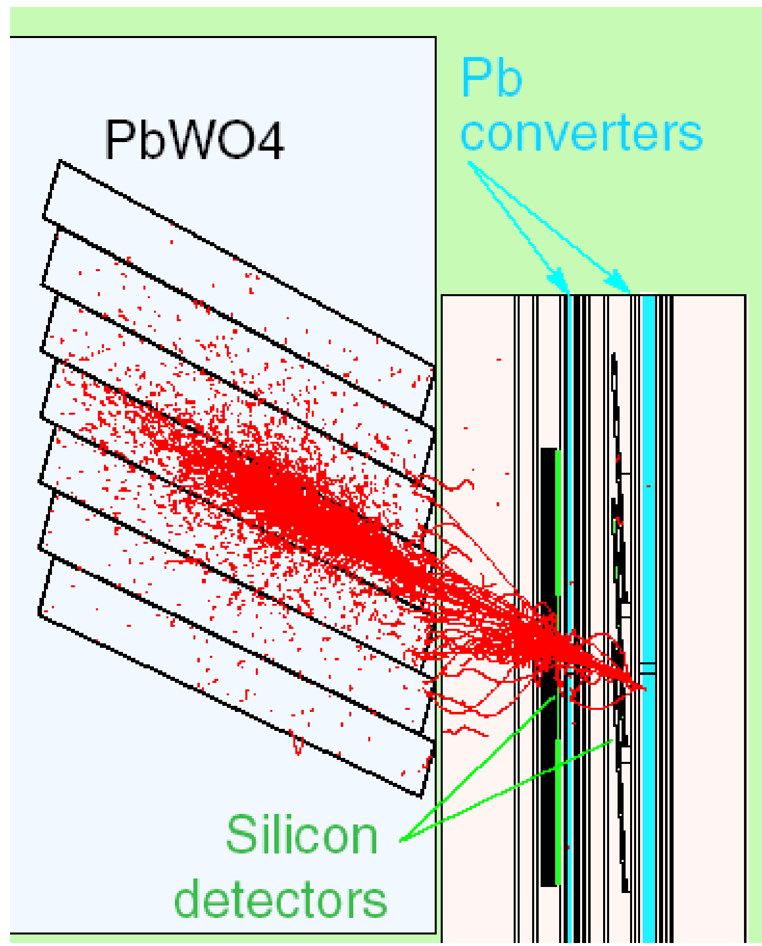


The Geant4 Stepdance

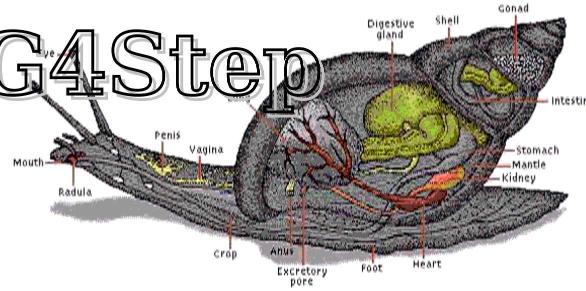
Example: electromagnetic shower
Especially when high energetic e-, e+, or photons hit some dense material, electromagnetic showers develop. Many, many secondaries. Demand lots of computing time!

γ 200 MeV in 2 X_0 Aluminium. Pair + brem





Anatomy of a G4Process/G4Step

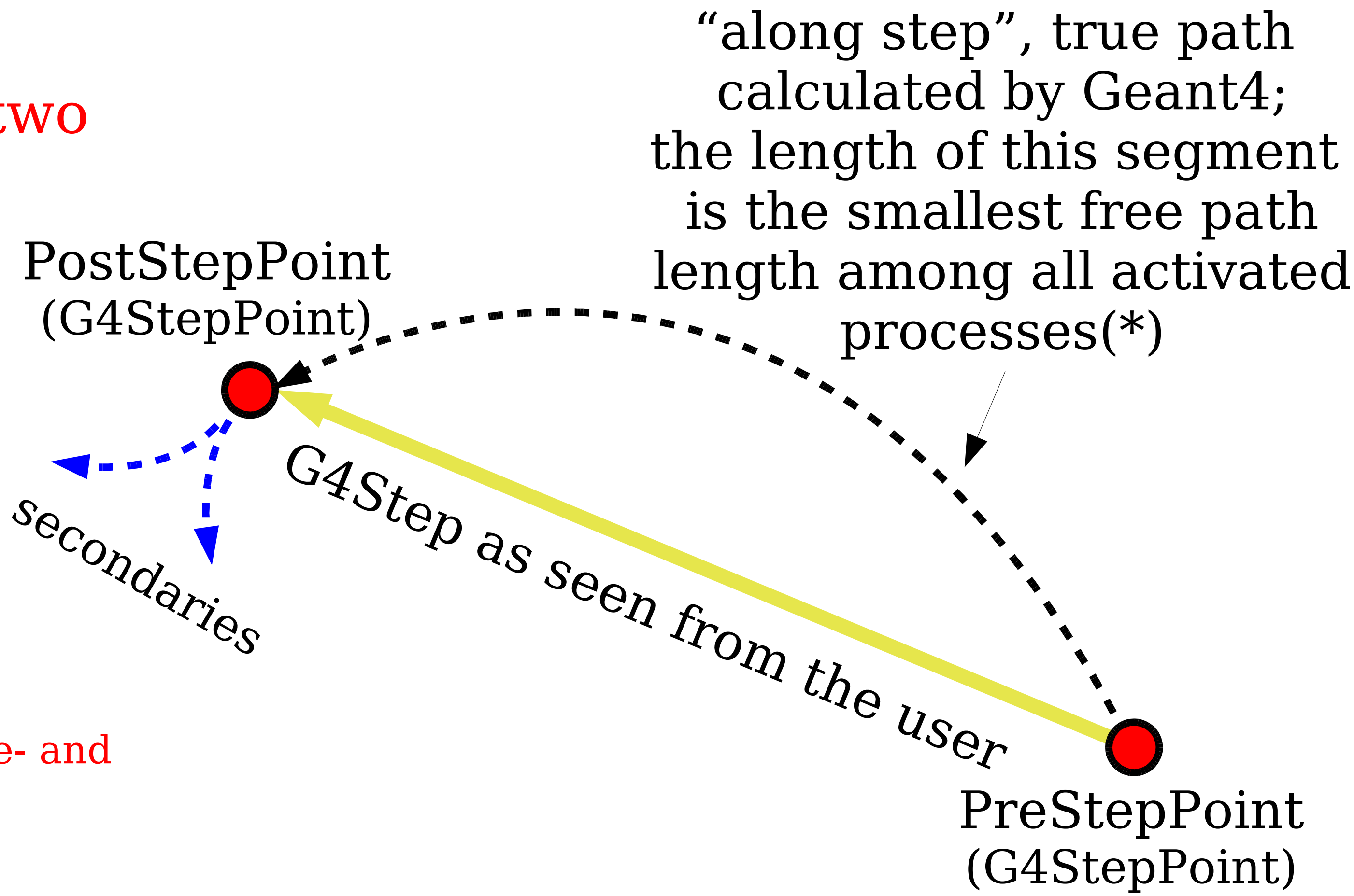


Recap:

- G4Step consists of two G4StepPoints
  - PreStepPoint, PostStepPoint
  - without physics, steps are delimited by geometry surfaces
- A set of activated physics processes delimits the step
  - based on sampling from the distribution of the mean free path length (process and material dependent)
- New particles, secondaries, can be created at the PostStepPoint or inbetween Pre- and PostStepPoint
  - depending on the kind of physics process which delimited this step
  - secondaries are created by sampling from the quantum mechanical distribution of the “final states” of the process
  - secondaries are put on a stack for later tracking

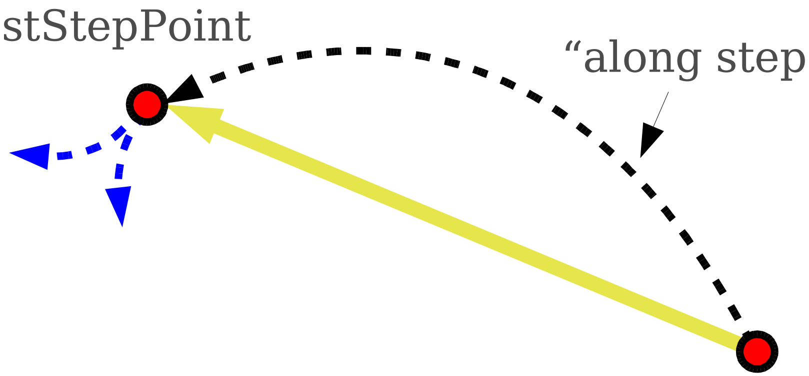
Anatomy of a G4Process/G4Step

- G4Step consists of two G4StepPoints
- A set of activated physics processes delimits the step
- New particles, secondaries, can be created at the PostStepPoint (or, exceptionally, in between Pre- and PostStepPoint)
  - along the step: secondary vertex is only an approximation in case of curved tracks!



(*) not completely true, if multiple coulomb scattering is also taken into consideration ... Let's ignore it for the moment! :-)

Anatomy of a G4Process/G4Step



class G4VProcess:

- mother of all processes in Geant4
- 3 sets of abstract methods

A particular implementation of G4VProcess has to implement all of them!

Geant4 provides some subclassed “shortcuts” with default implementations

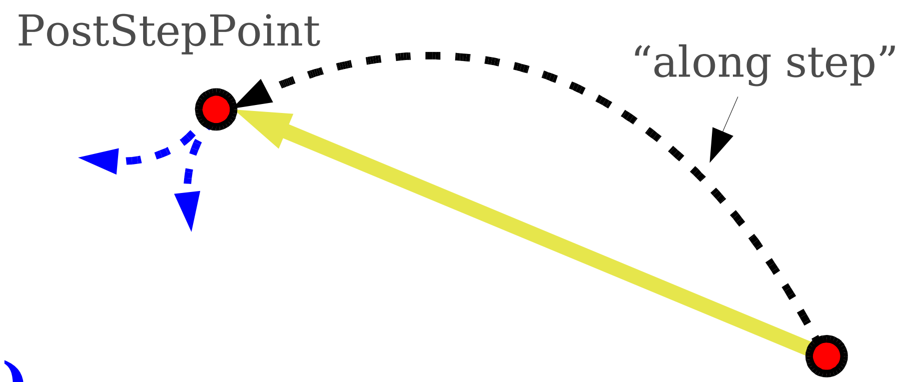
G4VProcess	PostStep	AlongStep	AtRest
GetPhysicalInteractionLength()			
DoIt()			

abstract default process

Anatomy of a G4Process/G4Step

class G4VDiscreteProcess :
public G4VProcess

- something happens at the PostStepPoint



GetPhysicalInteractionLength()
must return the free path length, includes sampling from $P(L) = \exp(L/\lambda)$

DoIt() implements the action of the process, when it was chosen because of returning the shortest interaction length of all activated processes for the particle being tracked

G4VDiscreteProcess	PostStep	AlongStep	AtRest
GetPhysicalInteractionLength()			
DoIt()			

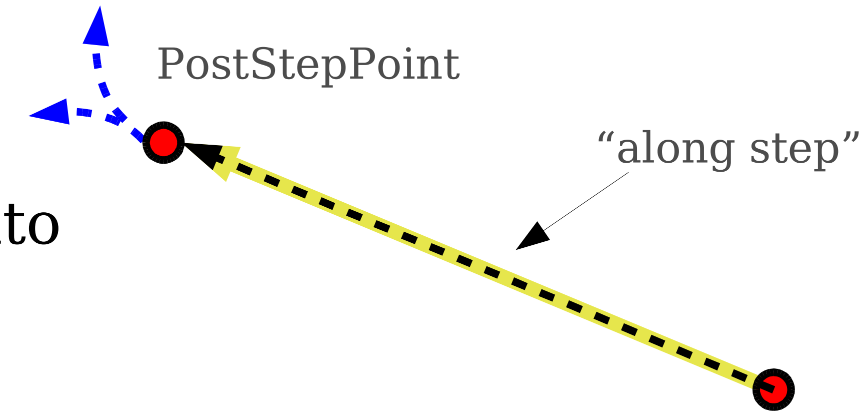
abstract default process

Anatomy of a G4Process/G4Step

Example:

class G4GammaConversion:
public G4VDiscreteProcess

- the tracked photon is converted into an electron – positron pair



GetPhysicalInteractionLength()
returns the free path length (x-section of gamma-conversion, material properties, sampling from $P(L) = \exp(L/\lambda)$)

DoIt() creates the (e-, e+) pair (sampling of energy, momentum, killing of photon)

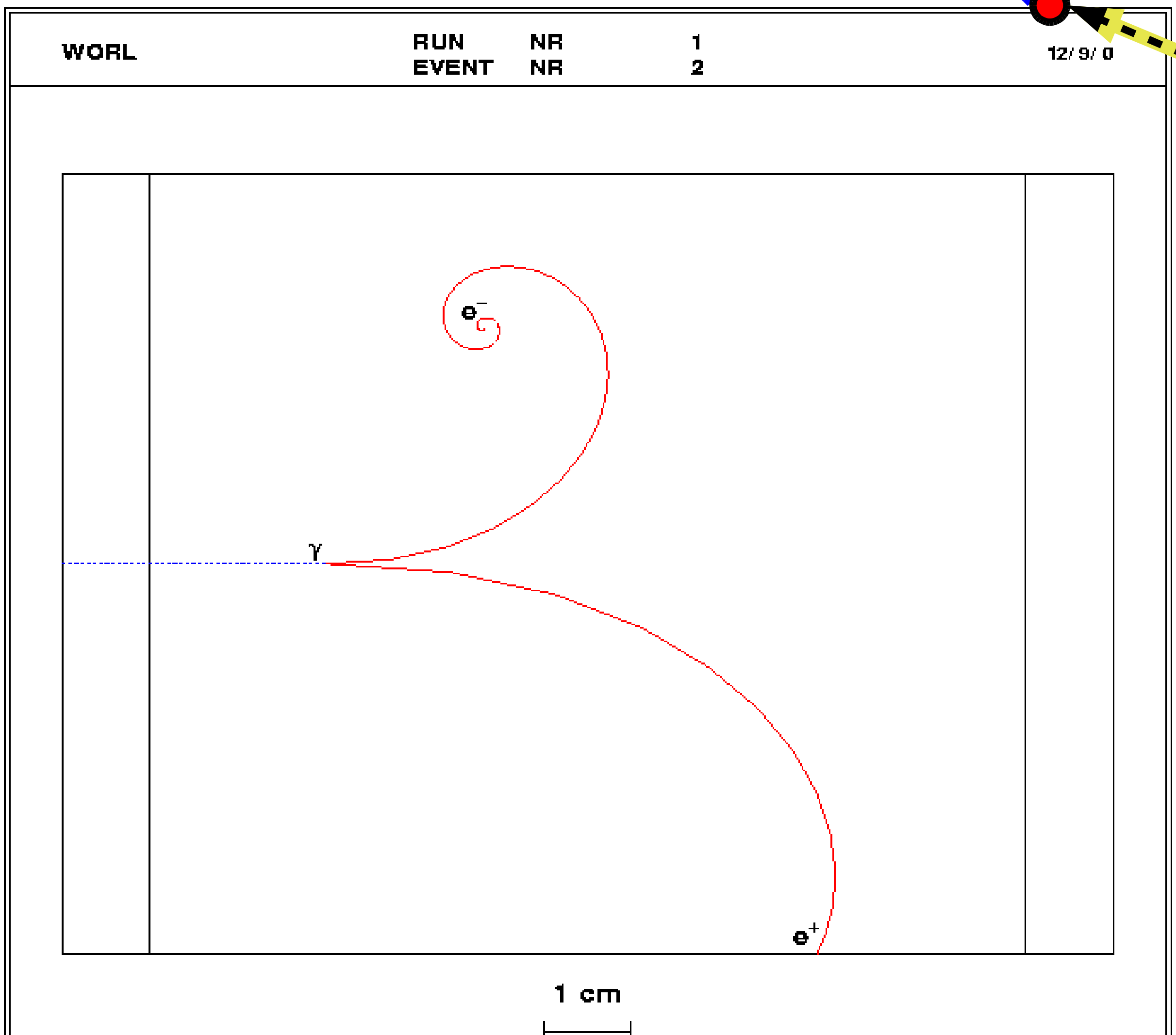
G4VDiscreteProcess	PostStep	AlongStep	AtRest
GetPhysicalInteractionLength()			
DoIt()			

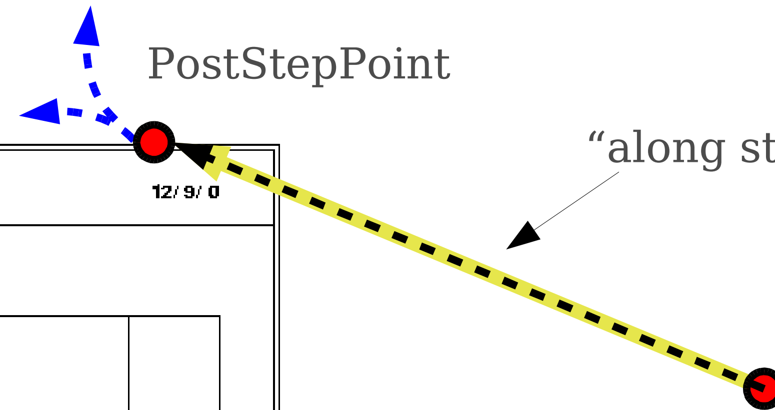
abstract default process

Anatomy of a G4Process/G4Step

Example:

γ 200 MeV in 10 cm Aluminium. Field 5 tesla





a-conversion,

rgy, momentum,

AtRest

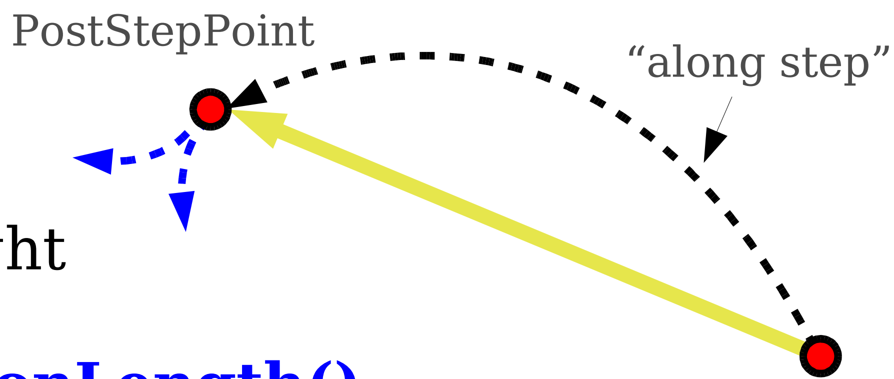
default

process

Anatomy of a G4Process/G4Step

class G4VContinousDiscreteProcess :
public G4VProcess

- something happens at the PostStepPoint
- something happens along the flight



AlongStepGetPhysicalInteractionLength()
returns a free path length – for example, limit the step, if the assumptions of the continous model would break down at a larger step-size ...

AlongStepDoIt() is ALLWAYS invoked BEFORE any PostStepDoIts to account for the continuous changes for the particle being tracked

G4VDiscreteProcess	PostStep	AlongStep	AtRest
GetPhysicalInteractionLength()			
DoIt()			

abstract default process

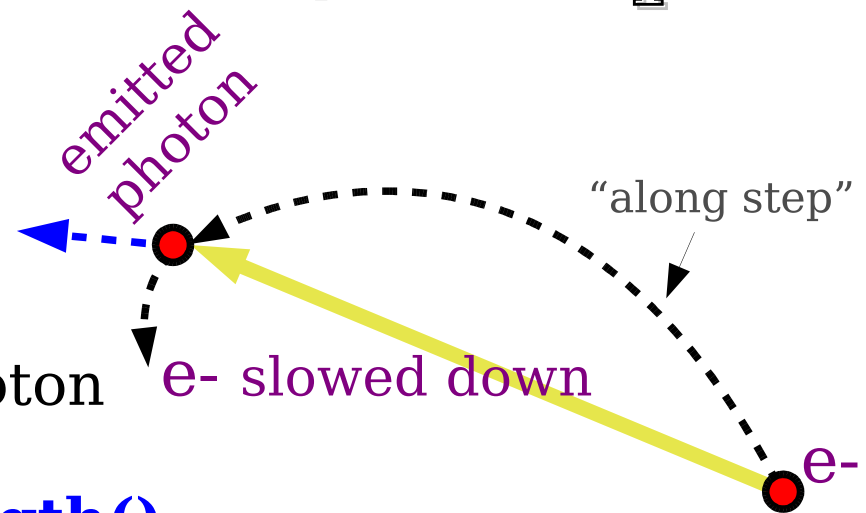
Anatomy of a G4Process/G4Step

Example:

class G4eBremsstrahlung :

public ….

- along step: avarage energy loss of the e-
- post step: emission of a brems-photon



PostStepPhysicalInteractionLength()
returns a free path length until the photon emission
PostStepDoit() emitts the photon
AlongStepPhysicalInteractionLength()
returns a step limit compatible with the employed dE/dx model
AlongStepDoIt() calculate dE/dx for the actual step taken
(note: the step taken is not necessarily due to Bremsstrahlung ...)

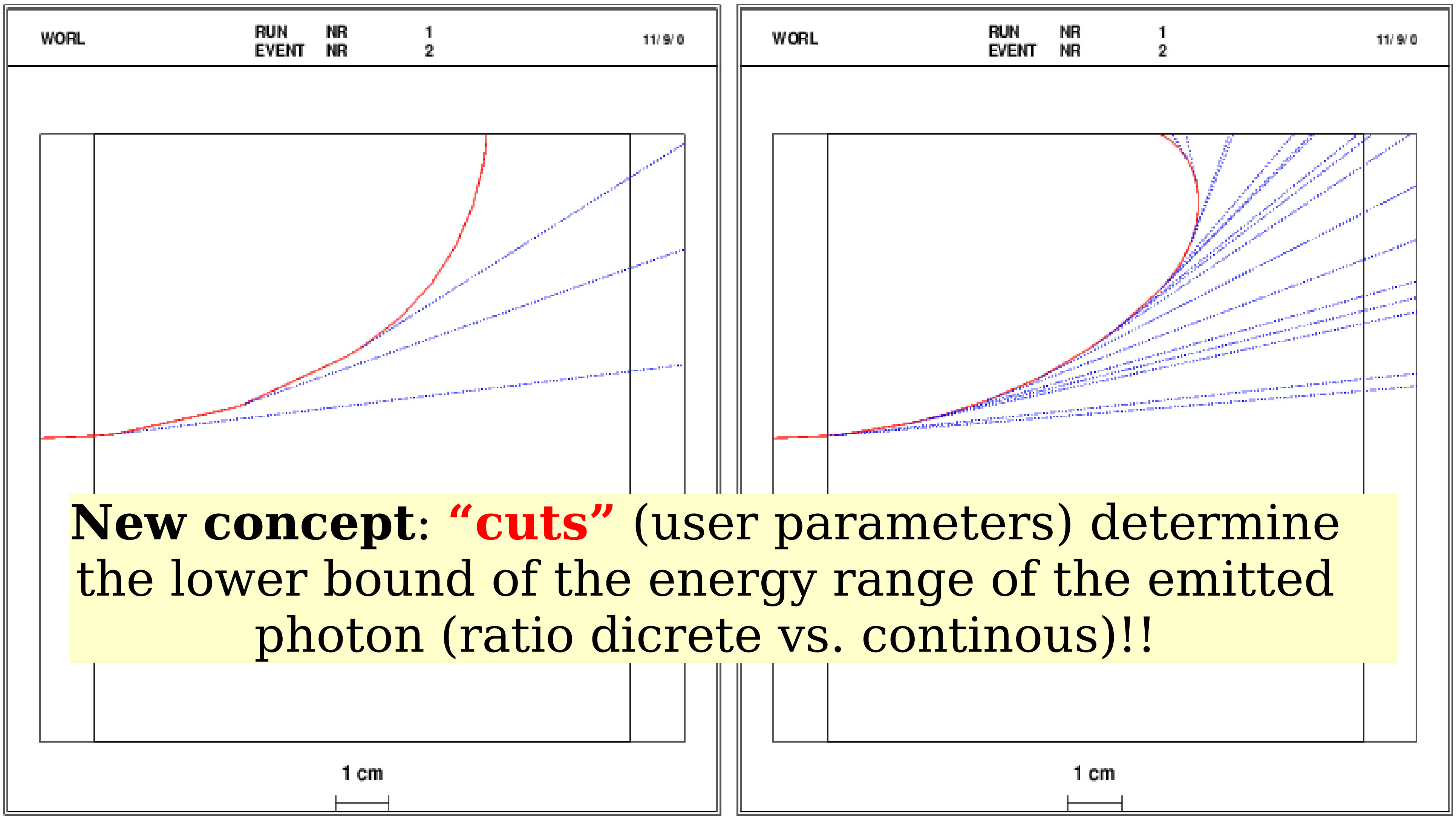
G4VDiscreteProcess	PostStep	AlongStep	AtRest
GetPhysicalInteractionLength()			
DoIt()			

abstract default process

Anatomy of a G4Process/G4Step

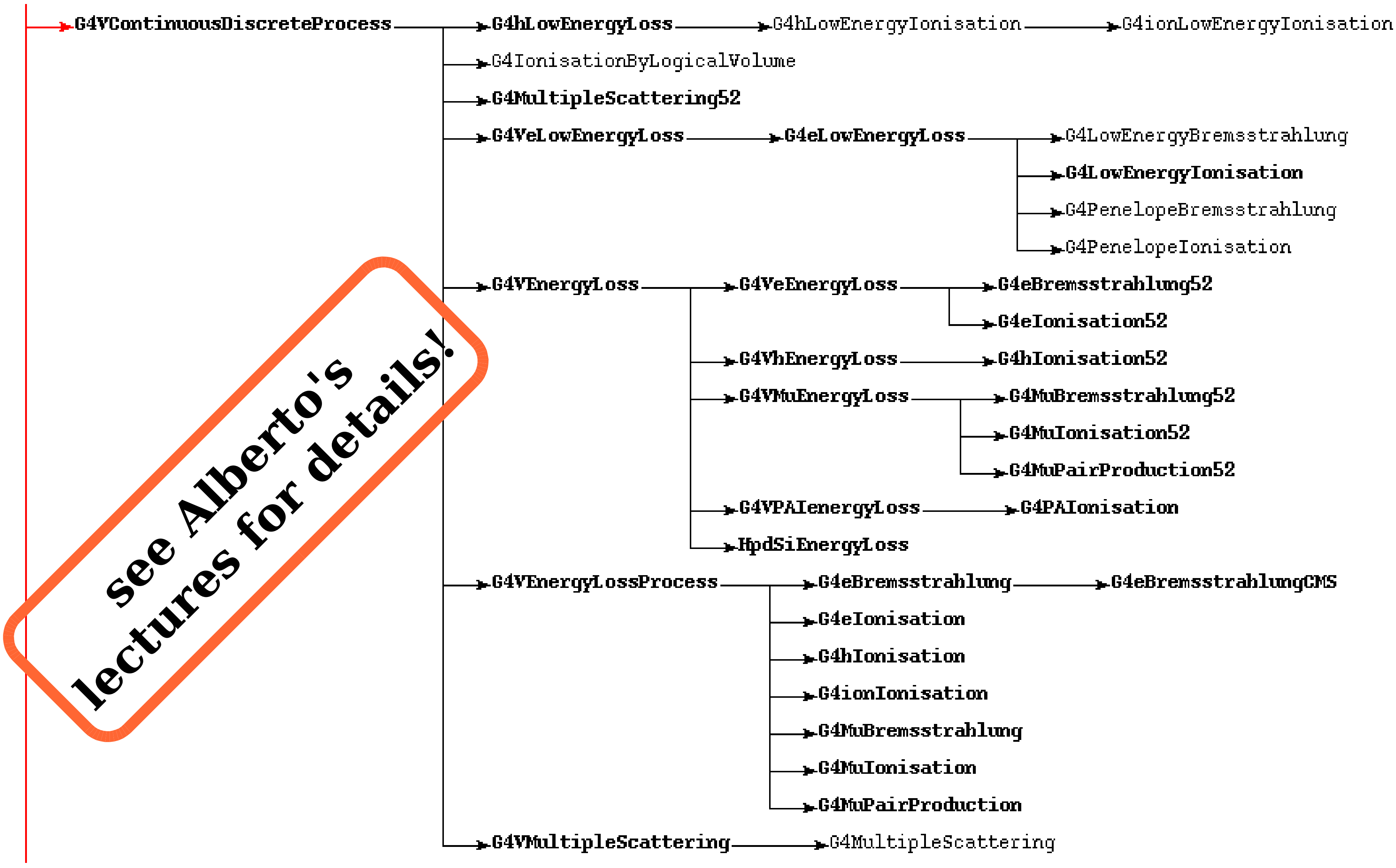
Example:

e^- 200 MeV in 10 cm Aluminium (cut: 1 MeV, 10 keV). Field 5 tesla



New concept: **"cuts"** (user parameters) determine the lower bound of the energy range of the emitted photon (ratio dicrete vs. continous)!!

G4VProcesses

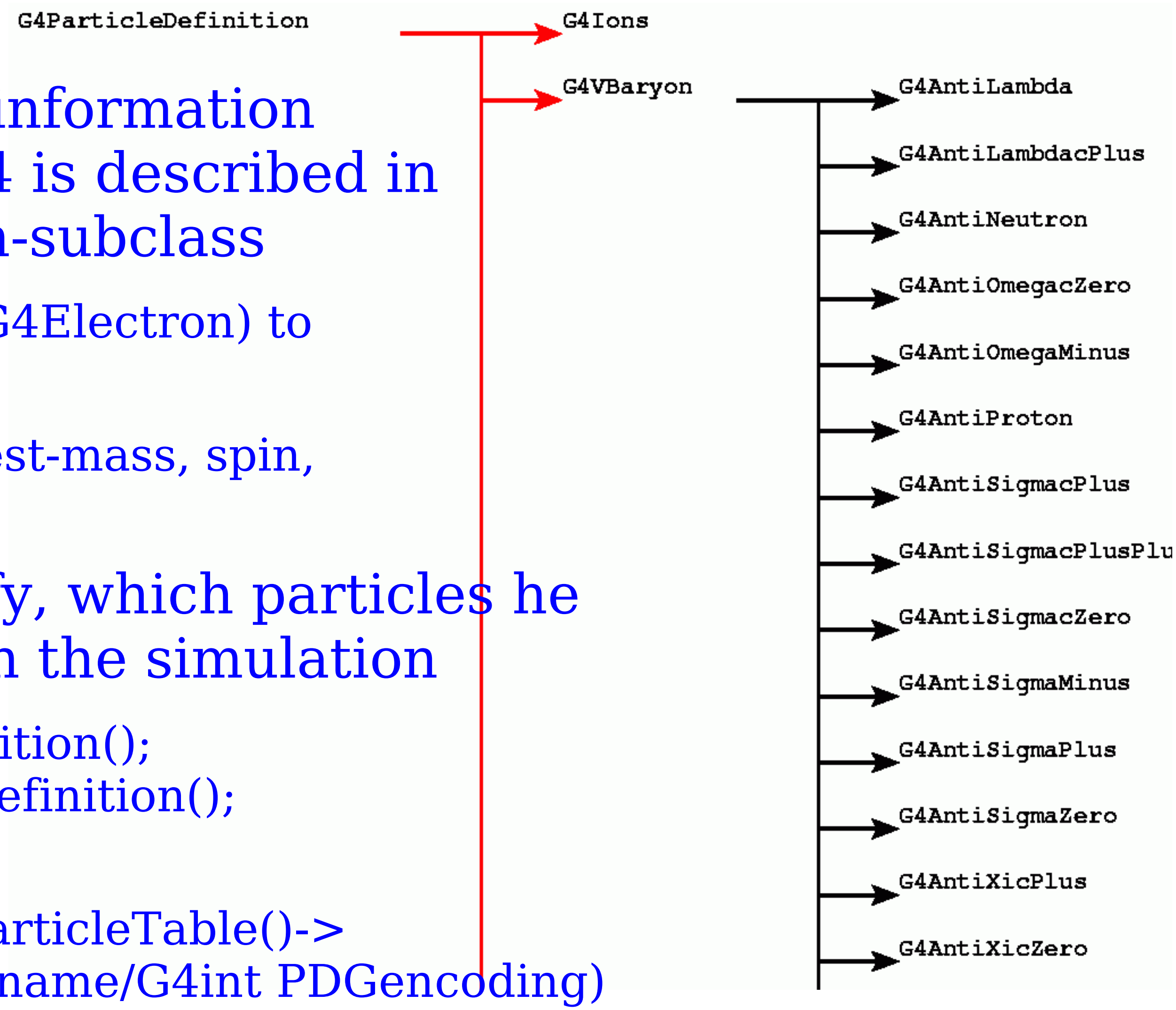


(only continuous-discrete electromagnetic processes shown)

Putting things together

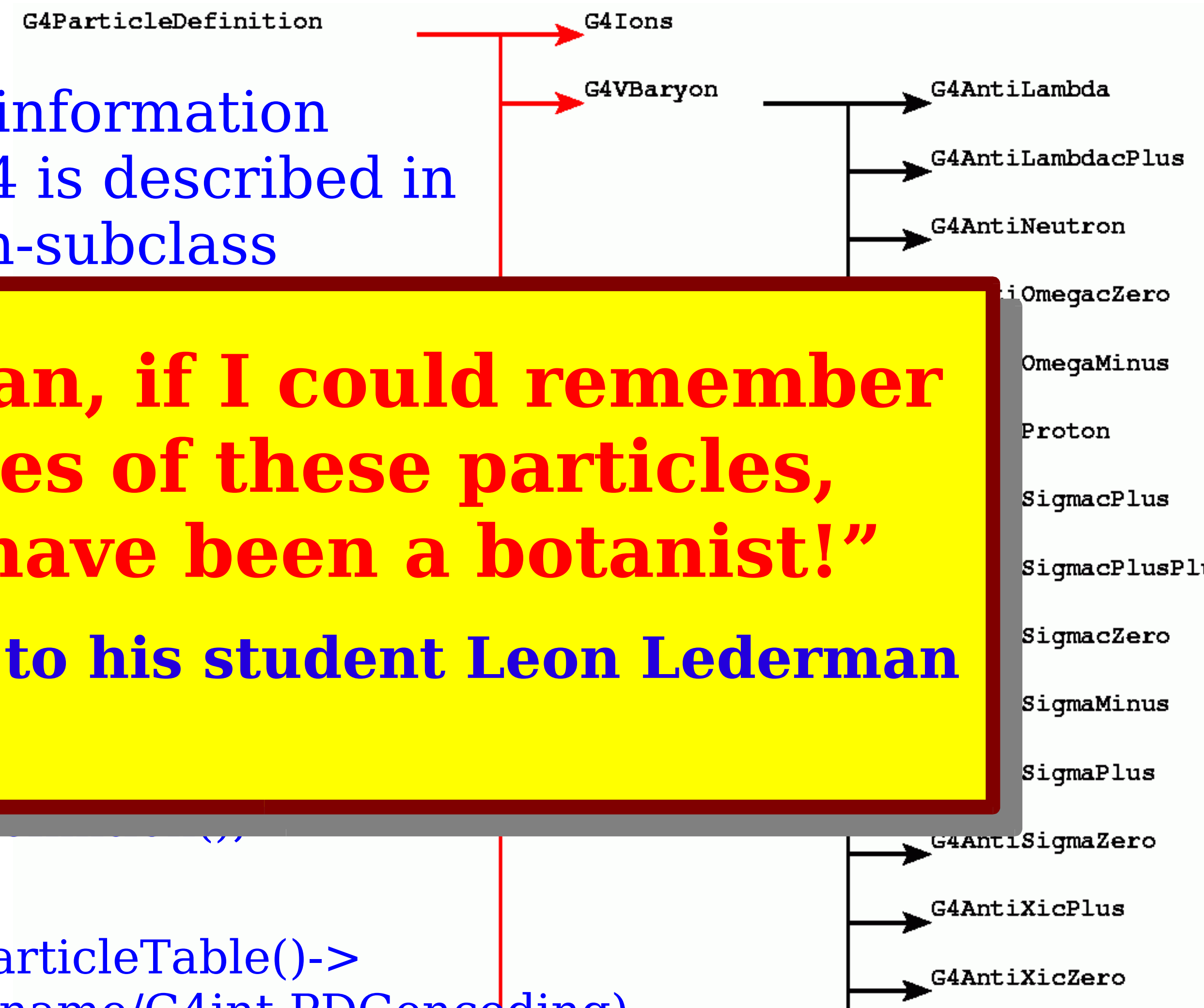
- **Tracking** in external fields without physics
  - is itself **modeled as a G4VProcess**
  - geometrical boundaries determine the “post-step-interaction length” (not Monte Carlo numbers)
- Each **particle** known to Geant4 has a **list of G4VProcesses** that influence the tracking of the particle
  - all lists together are called “Physics Lists” in Geant4
  - at program initialization time, each process calculates its cross-sections for each material for its energy range
- The Geant4 **Tracking Manager** invokes all processes via their polymorphic G4VProcess-interface
  - “rat race” for the shortest interaction length
  - doit() of all processes “along step”, doit() of the process with the shortest interaction length
  - book-keeping & tracking of secondaries

Particles & Processes



- In general, the static information of a particle in Geant4 is described in a G4ParticleDefinition-subclass
  - static member (static G4Electron) to represent the particle
  - name, particle-code, rest-mass, spin, iso-spin, ...
- The user has to specify, which particles he wants to participate in the simulation
  - G4Proton::ProtonDefinition();
    G4Electron::ElectronDefinition();
    ....
  - G4ParticleTable::GetParticleTable()->
    FindParticle(G4String name/G4int PDGencoding)
  - G4LeptonConstructor pConstructor; pConstructor.ConstructParticle();

Particles & Processes



- In general, the static information of a particle in Geant4 is described in a G4ParticleDefinition-subclass
  - sta
    rep
  - na
    iso
- The u
  wants
  - G4
    G4
    ....
  - G4ParticleTable::GetParticleTable()-> FindParticle(G4String name/G4int PDGencoding)
  - G4LeptonConstructor pConstructor; pConstructor.ConstructParticle();

"Young man, if I could remember the names of these particles, I would have been a botanist!"

Enrico Fermi to his student Leon Lederman

Particles & Processes

- Each process must (amongst others ...) implement
  - G4VProcess::IsApplicable(const G4ParticleDefinition &);
- Therefore: m:n relation between processes and particles established by a process-manager
- It's the users responsibility to define which processes and particles to be used in the simulation:

```
//  Get the process manager for gamma
G4ParticleDefinition* particle = G4Gamma::GammaDefinition();
G4ProcessManager* pmanager = particle->GetProcessManager();

// Construct processes for gamma
G4PhotoElectricEffect * thePhotoElectricEffect
                        = new G4PhotoElectricEffect();

G4ComptonScattering * theCompton = new G4ComptonScattering();
G4GammaConversion* theGammaConversion = new G4GammaConversion();

// Register processes to gamma's process manager
pmanager->AddDiscreteProcess(thePhotoElectricEffect);
pmanager->AddDiscreteProcess(theCompton);
pmanager->AddDiscreteProcess(theGammaConversion);
```

Particles & Processes

- Building a “good” physics-list is an art!
- Even then you have to tune your physics
  - trade-off accuracy/performance
  - tuning with testbeam data!!
- For example, especially for electromagnetic processes:
  - definition of **production cuts / thresholds**
  - tell the process when to stop producing secondaries (infrared divergences in the cross-sections, speed, ...)
  - a cut is given as a length: for a given process no secondary is created if this secondary would not survive the given distance
- see **Alberto's lectures** for further details on physics processes, tuning & validation of various physics-lists

PART VI

Setting up a Geant4
based Simulation:
User Initializers
User Actions
The G4 Event Loop

Task-List

(for a self-sustained Geant4 based simulation)

:-) **Detector description**

:-) **Tracking & Monte Carlo**

:-) **Particles & Processes**

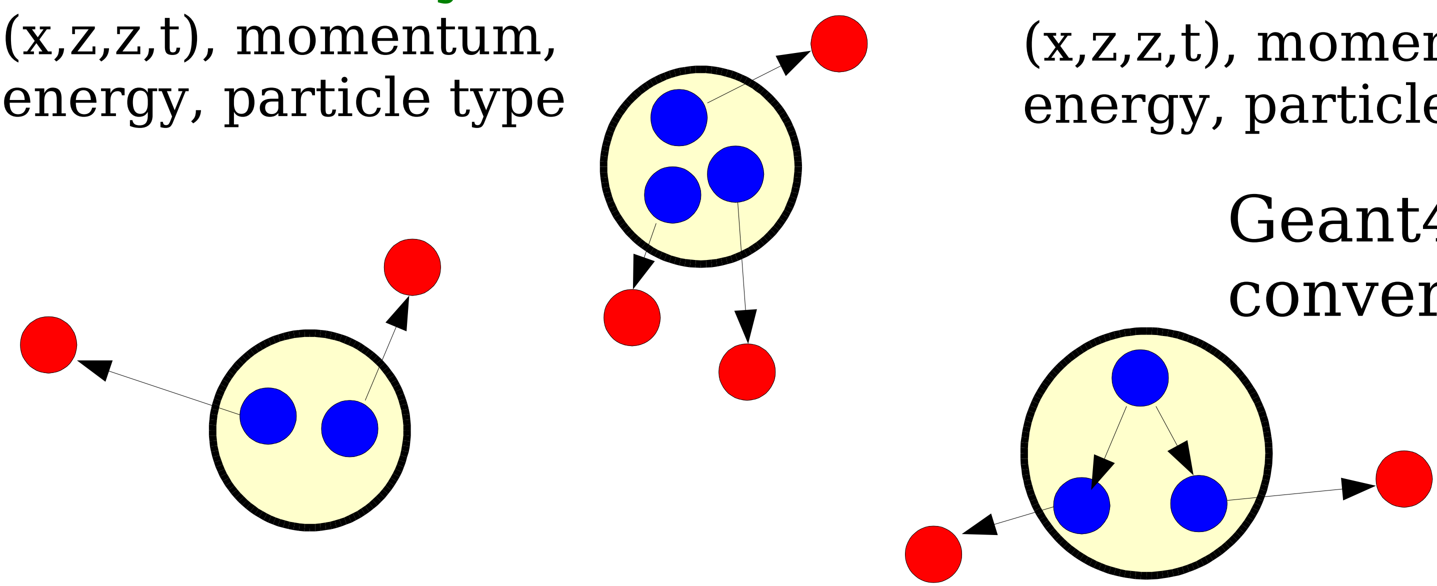
→ **Primaries** ←

Primary Particles

The user's responsibility to provide a list of particles and their kinematic properties to be tracked by Geant4:

G4DynamicParticle
track-able by G4
(x,z,z,t), momentum,
energy, particle type

G4PrimaryParticle
not track-able
(x,z,z,t), momentum,
energy, particle type, daughters



Geant4:
convert( ) →

G4PrimaryVertex
(x,y,z,t)

1 — n

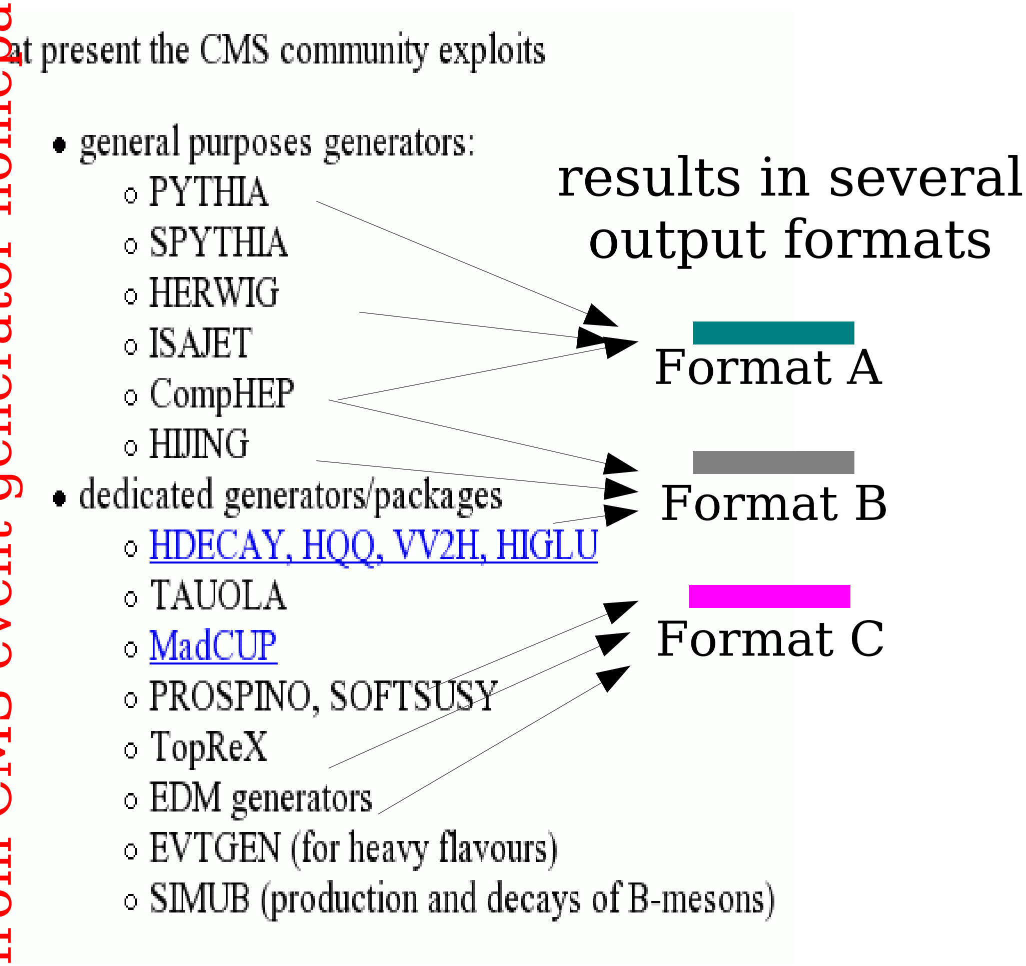
G4PrimaryParticle 1 — n

The Primary generating Action

class G4VUserPrimaryGeneratorAction

virtual void GeneratePrimaries (G4Event *)=0;

class MyGenerator : public G4VUserPrimaryGeneratorAction



from CMS event-generator homepage

G4Event::AddPrimaryVertex (G4PrimaryVertex*)

?

Format Geant4

The Primary generating Action

class G4VUserPrimaryGeneratorAction

virtual void GeneratePrimaries (G4Event *)=0;

class MyGenerator : public G4VUserPrimaryGeneratorAction

from CMS event-generator homepage

at present the CMS community exploits

- general purposes generators:
  - PYTHIA
  - SPYTHIA
  - HERWIG
  - ISAJET
  - CompHEP
  - HIJING
- dedicated generators/packages
  - HDECAY, HQQ, VV2H, HIGLU
  - TAUOLA
  - MadCUP
  - PROSPINO, SOFTSUSY
  - TopReX
  - EDM generators
  - EVTGEN (for heavy flavours)
  - SIMUB (production and decays of B-mesons)

results in several output formats

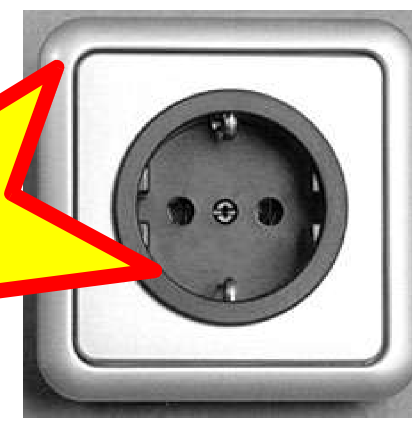
Format A

Format B

Format C

G4Event::AddPrimaryVertex (G4PrimaryVertex*)

magic!



Format Geant4

your implementation of G4VUserPrimaryGeneratorAction

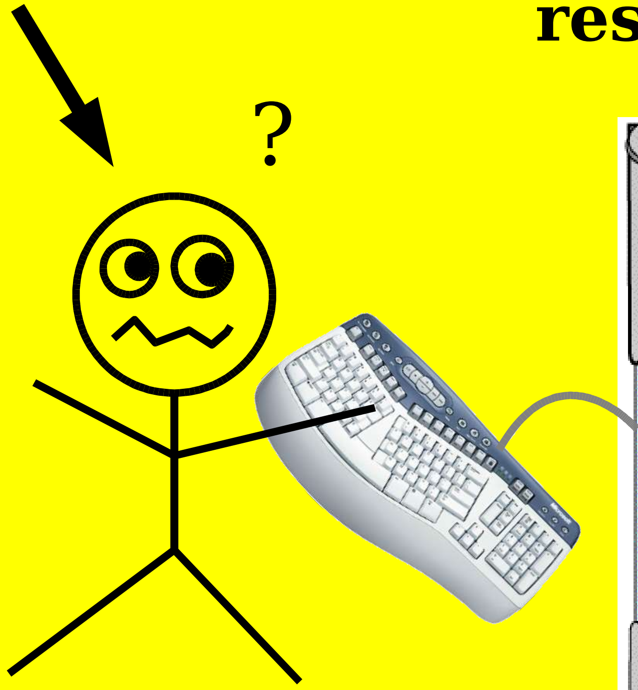
>> Mandatory Implementation <<

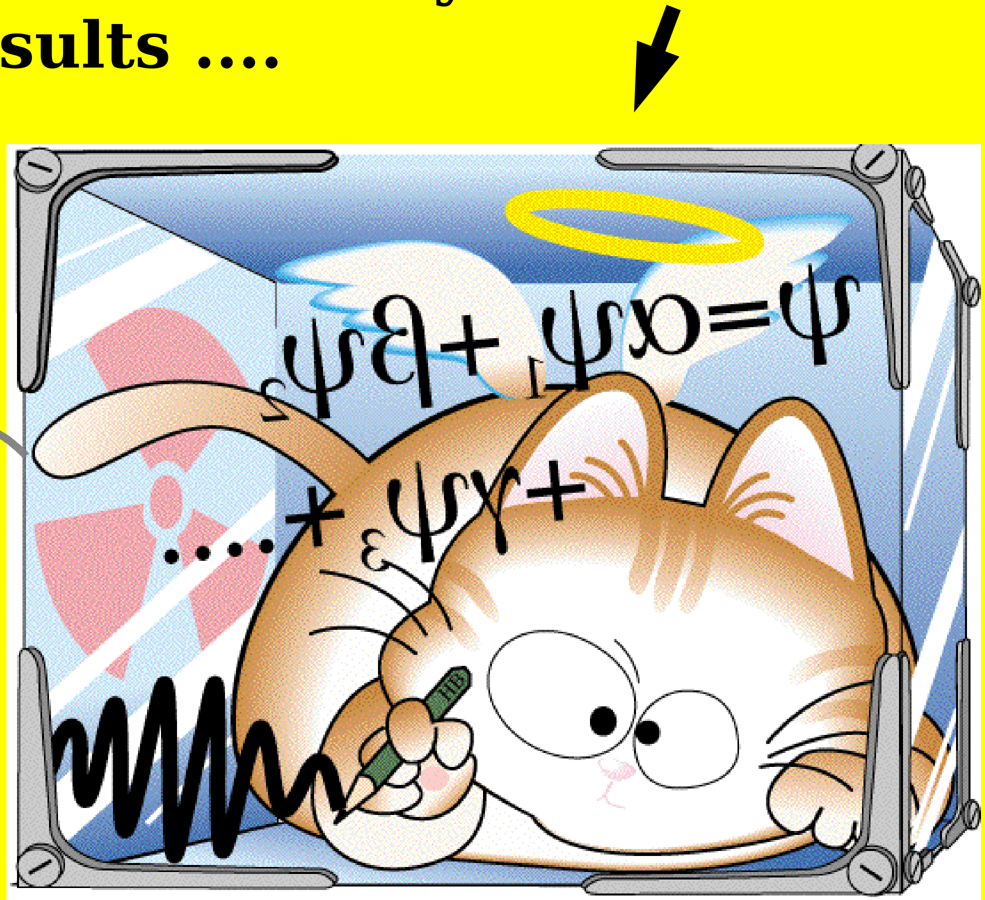
- class G4VUserDetectorConstruction
  - virtual G4VPhysicalVolume * Construct() = 0;
  - overload to implement the detector description
- class G4VUserPhysicsList
  - virtual void ConstructParticles() = 0;
    virtual void ConstructProcesses() = 0;
  - overload for particles & physics processes selection
- class G4VUserPrimaryGenerator
  - virtual void GeneratePrimaries(class G4Event *) = 0;
  - overload to generate primary particles

Once implementations of these three base classes are provided, a Geant4 based simulation can run!

>> Mandatory Implementation <<

But apart from CPU processing time,
you, the user, don't observe any simulation results ....

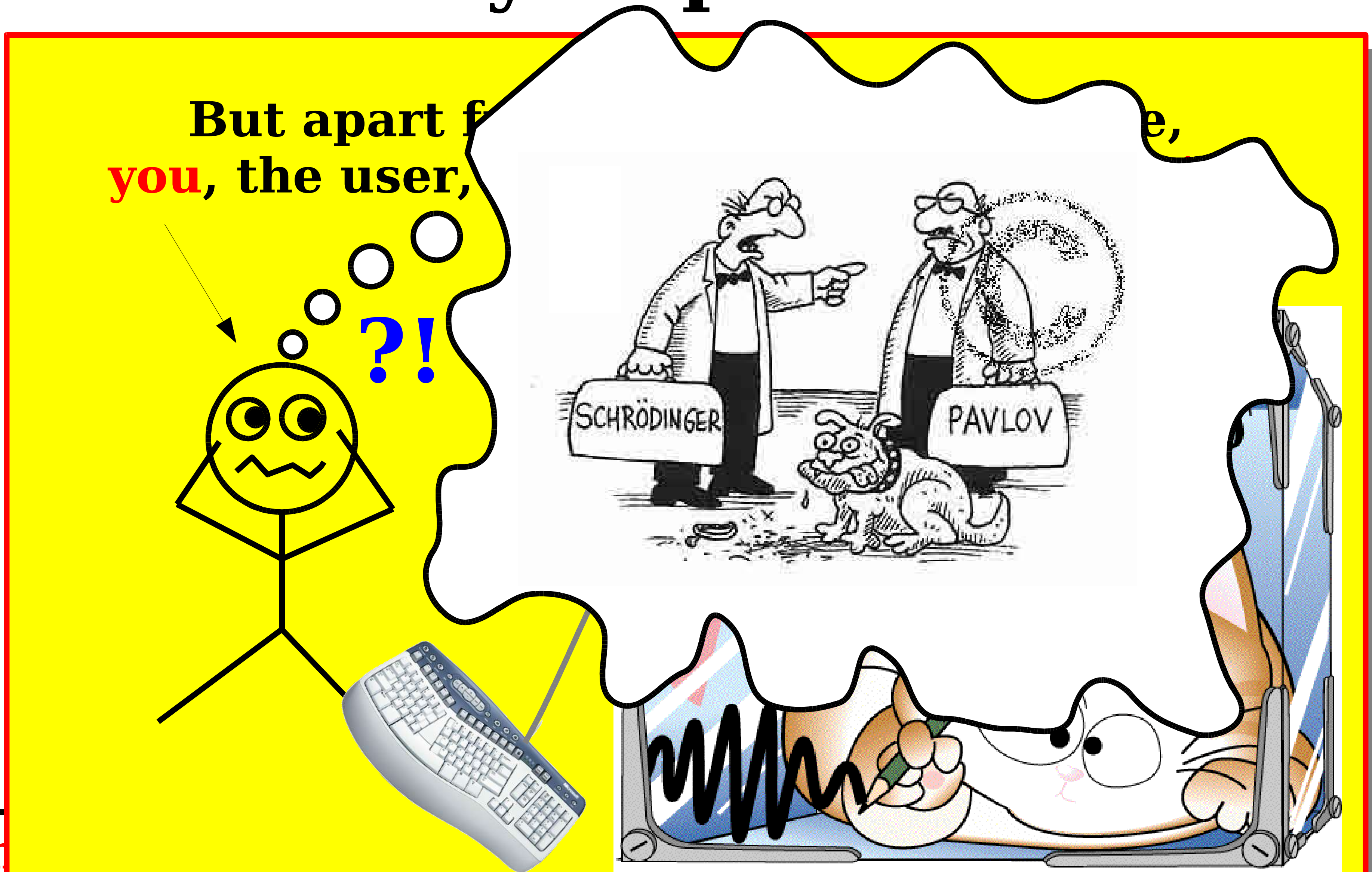




On

provided, a Geant4 based simulation can run!

>> Mandatory Implementation <<



On

provided, a Geant4 based simulation can run!

Squeezing it out!

- What we don't know yet – most importantly for the user:
  - How do we squeeze out simulation information?
- Classification of a Geant4 simulation loop
  - Simulation, Run, Event, Track, Step
- Concept of a “Sensitive Detector”
  - Hits
  - Digitization
- Typical usage
  - where is what kind of information extracted

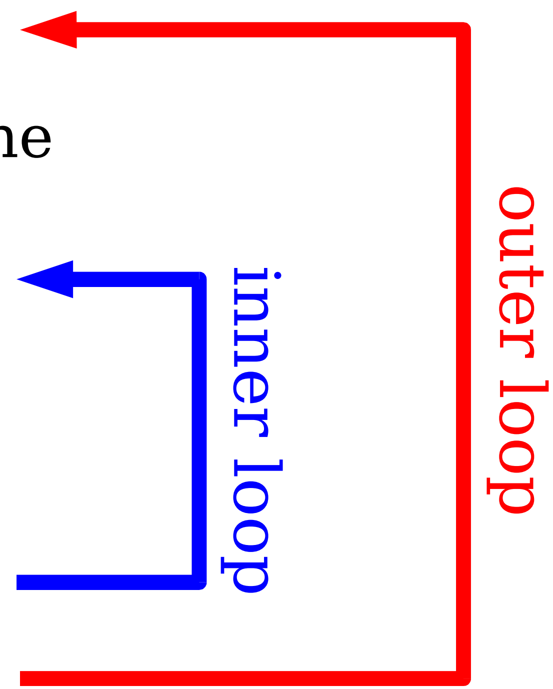
User Actions / Initializations

- Geant4 foresees user hooks to instrument the simulation:
  - User-Actions: sub-classes of G4UserXYZAction
  - polymorphic methods are called by the G4-kernel
- **Mandatory User Actions** (we have seen them already)
  - Detector description
  - Physics List
  - Primary event generation
- **Optional User Actions**
  - Run-Action: beginning and end of a Run
  - Event-Action: beginning and end of an Event
  - Track: beginning and end of the track of one particle
  - Step: after each G4Step
  - Hooks for hit and digitization processing

Task Breakdown

User-Action & Initialization-Code invoked during
various stages during a simulation run

- Initialization
  - detector description
  - physics processes selection and configuration
- Run
  - contains several Events under the same simulation conditions
- Event
  - generation of primary particles
  - tracking of all particles



Task Breakdown

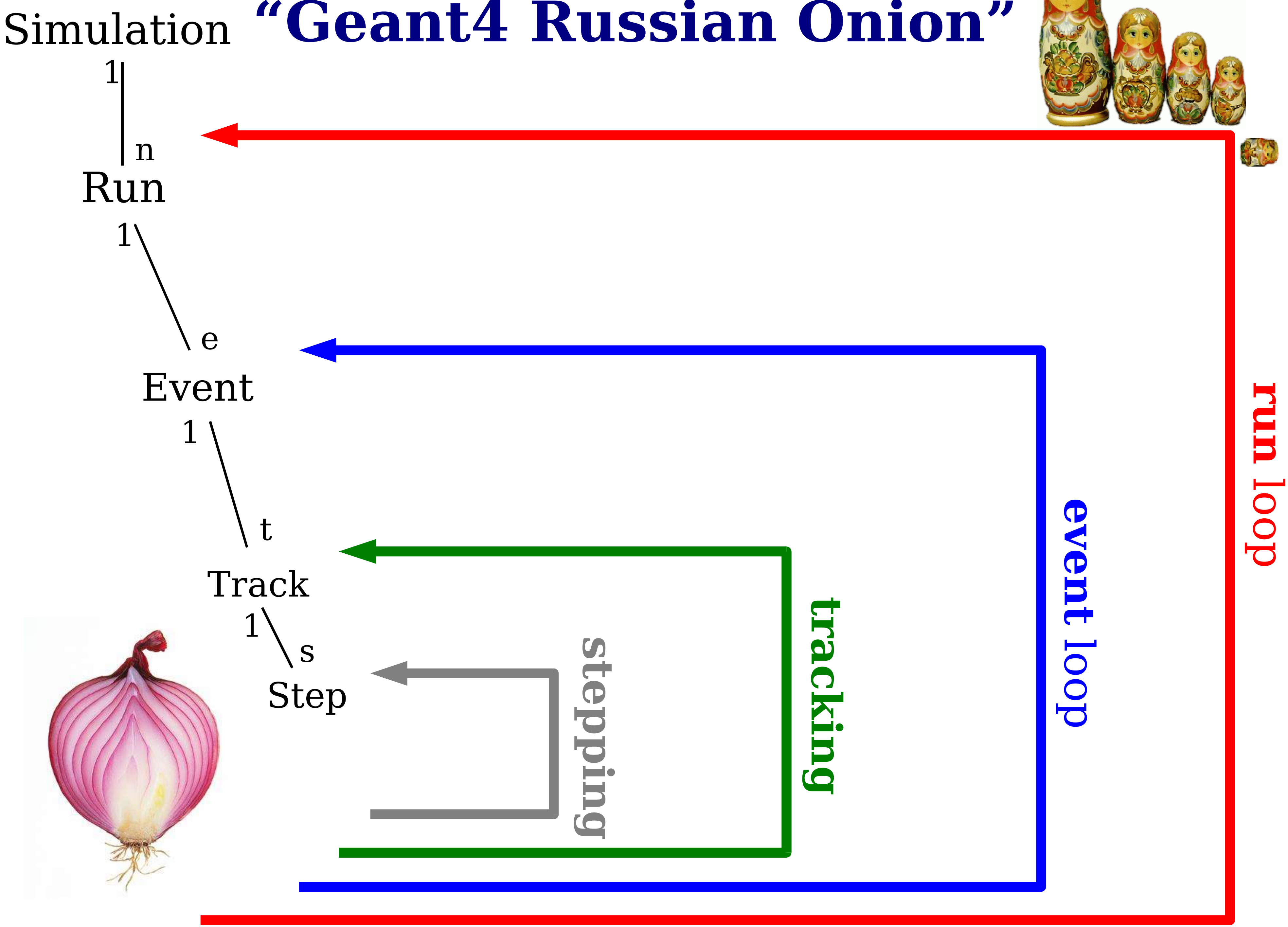
- In
  - –
  - –
- Ru
  - –
- Ev
  - –
  - –

Within these two main nested loops,
Run & Event,
Geant4 offers several user-hooks (call-backs)
that can be implemented by the
simulation user to extract simulation information
during tracking or to influence / steer
the simulation:

Run-, Event-, Track-, Step-Actions,
Primary Particle - Action

(not treated in these lectures:
Stack-Action)

“Geant4 Russian Onion”



User hooks via sub-classes

Simulation
1
n
G4Run
1
e
Event
1
t
Track
1
s
Step

G4VUserDetectorConstruction

G4VUserPhysicsList

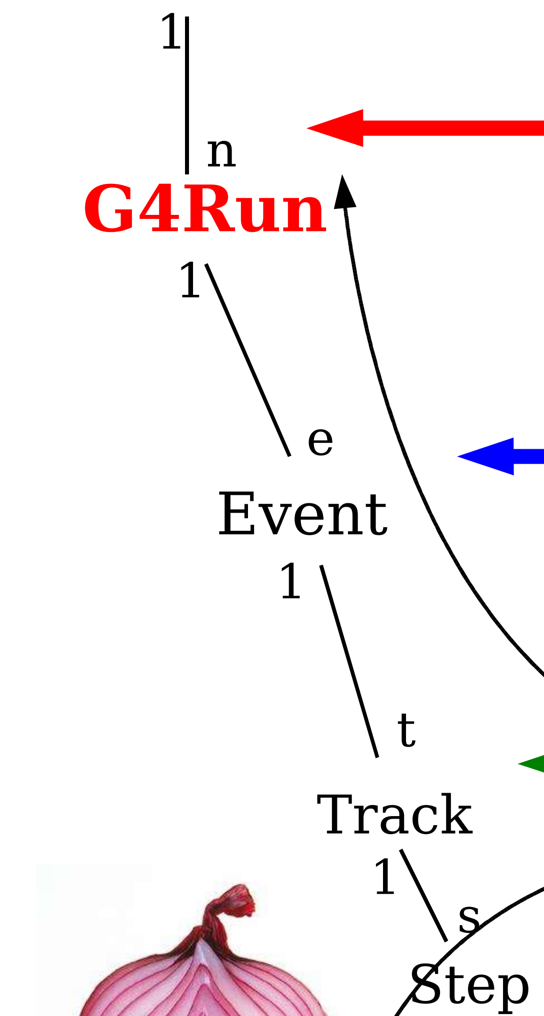
class MyDetector
: public G4VUserDetectorConstruction

class MyPhysics
: public G4VUserPhysicsList

before a Run starts:
- process user-parameters
  (not fully treated in these lectures)
- create materials, geometry, physics processes

run loop

User hooks via sub-classes



G4Run

RecordEvent(G4Event*)
int GetRunID()
.. administration of
hit collection tables ..

class MyRun
: public G4Run

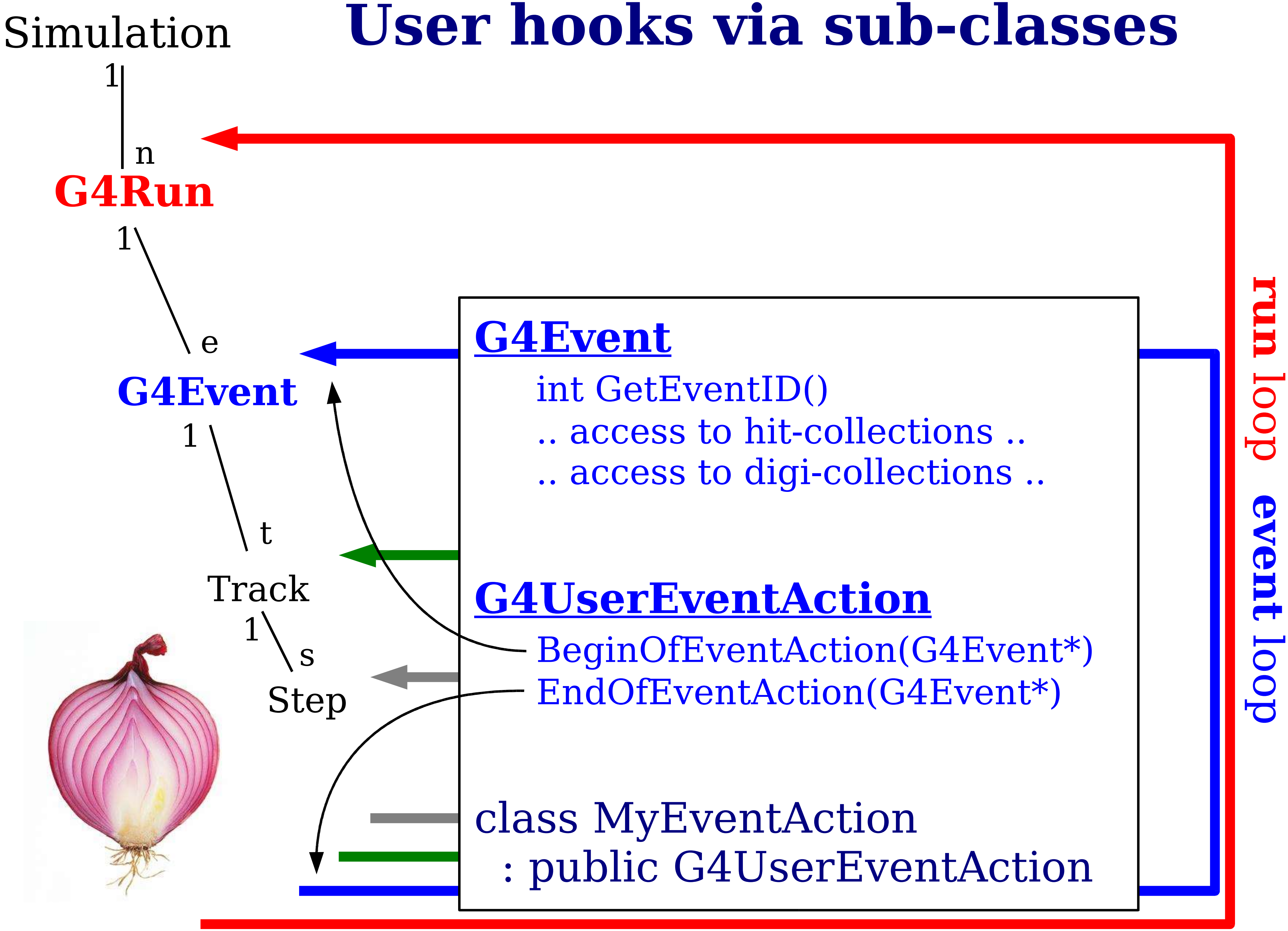
G4UserRunAction

BeginOfRunAction(G4Run*)
EndOfRunAction(G4Run*)

class MyRunAction
: public G4UserRunAction

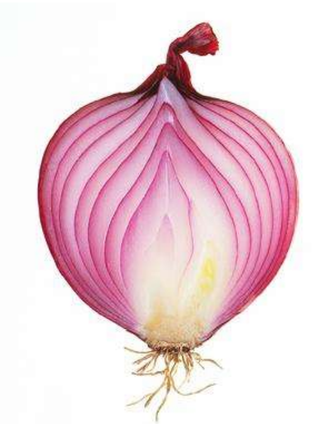
run loop

User hooks via sub-classes



User hooks via sub classes

Simulation
1
n
G4Run
1
e
G4Event
1
t
Track
1
s
Step



G4VUserPrimary-GeneratorAction

GeneratePrimaries
(G4Event *)

class MyGenerator : public
G4VUserPrimary....

G4Event

int GetEventID()
.. access to hit-collections
.. access to digi-collections ..

G4UserEventAction

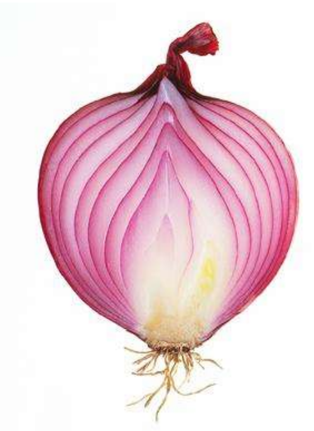
BeginOfEventAction(G4Event*)
EndOfEventAction(G4Event*)

class MyEventAction
: public G4UserEventAction

run loop
event loop

User hooks via sub-classes

Simulation
1
n
G4Run
1
e
G4Event
1
t
G4Track
1
s
Step



G4Track

int GetTrackID()
.. access track vertex ..
.. access current kinematics ..

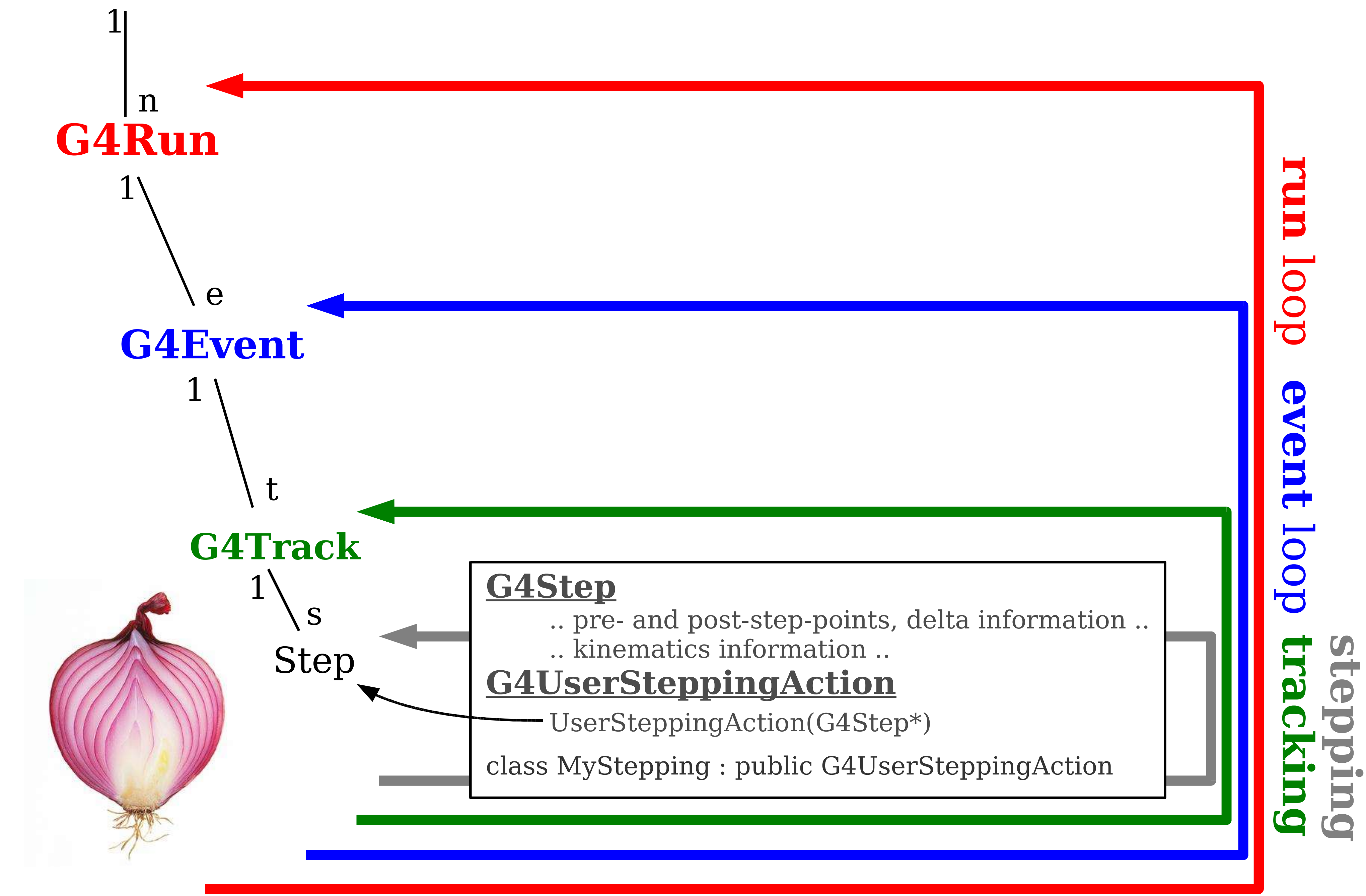
G4UserTrackingAction

PreUserTrackingAction(G4Track*)
PostUserTrackingAction(G4Track*)

class MyTrackingAction
: public G4UserTrackingAction

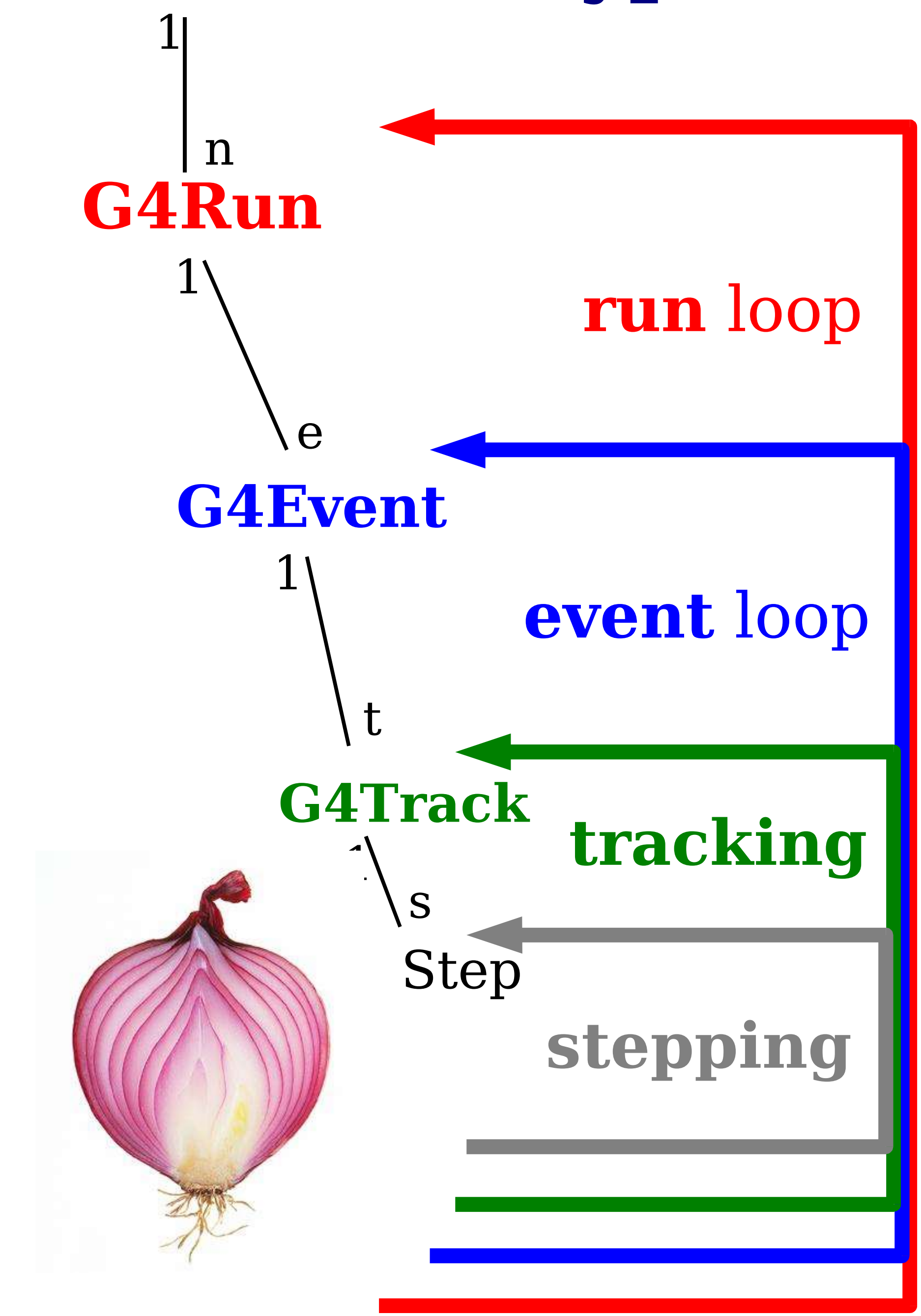
run loop **event** loop **tracking**

User hooks via sub-classes



Typical usage of User-Actions

Simulation



Persistency related tasks;
extraction of meta-data;
pile-up; digitization

Persistency related tasks;
hit analysis;
digitization

collecting "MC truth";
influencing order of simulation

collecting "MC truth";
debugging

Putting them together ...

- Singleton **G4RunManager** is the central registry for the mandatory and optional user extensions to Geant4:
  - SetUserInitialization(..*)
    - detector description: G4VUserDetectorConstruction **[m]**
    - physics selection: G4VUserPhysicsList **[m]**
  - SetUserAction(..*)
    - primary generator: G4VUserPrimaryGeneratorAction **[m]**
    - run: G4VUserRunAction **[o]**
    - event: G4VUserEventAction **[o]**
    - track: G4VUserTrackingAction **[o]**
    - step: G4VUserSteppingAction **[o]**
  - BeamOn(G4int **n**)
    - start the simulation of **n events.**

[m] mandatory
[o] optional

*(G4Run- & other Managers)

- G4RunManger registers actions to other **managers**
  - Event, Tracking, SteppingManagers, ..
  - many are singletons, i.e. kind of registries,
  - Managers are responsible for the steering of a specific simulation-layer
  - usually not visible to the user
- G4RunManager::beamOn(G4int n)
  - triggers a Run
  - n Events are simulated by invoking the G4EventManger
    - G4EventManager invokes G4TrackingManager
      - G4TrackingManager invokes G4SteppingManager
        - ...
- Also Managers for geometry, visualization, user-interface, ... (no time in this lectures ...)

Putting them together ...

- **BUT: you don't have to use G4RunManger**
  - you can re-write it completely to fit your simulation requirements
  - then you have to care about how your actions are registered to Geant4
  - then you have to control the Run- & Event-Loops!
    - i.e. implement the counterpart of BeamOn(G4int n)
    - invoke your PrimaryGenerator
  - CMS does this!
- **Many technical details**
  - all described in the Geant4 manuals
  - **but not time to show today ...**