Introduction to Geant4

African School of Fundamental Physics and Applications
Windhoek (Namibia) 3rd July 2018

African School of Fundamental
Physics and Applications

Carlo Mancini Terracciano
carlo.mancini.terracciano@roma1.infn.it

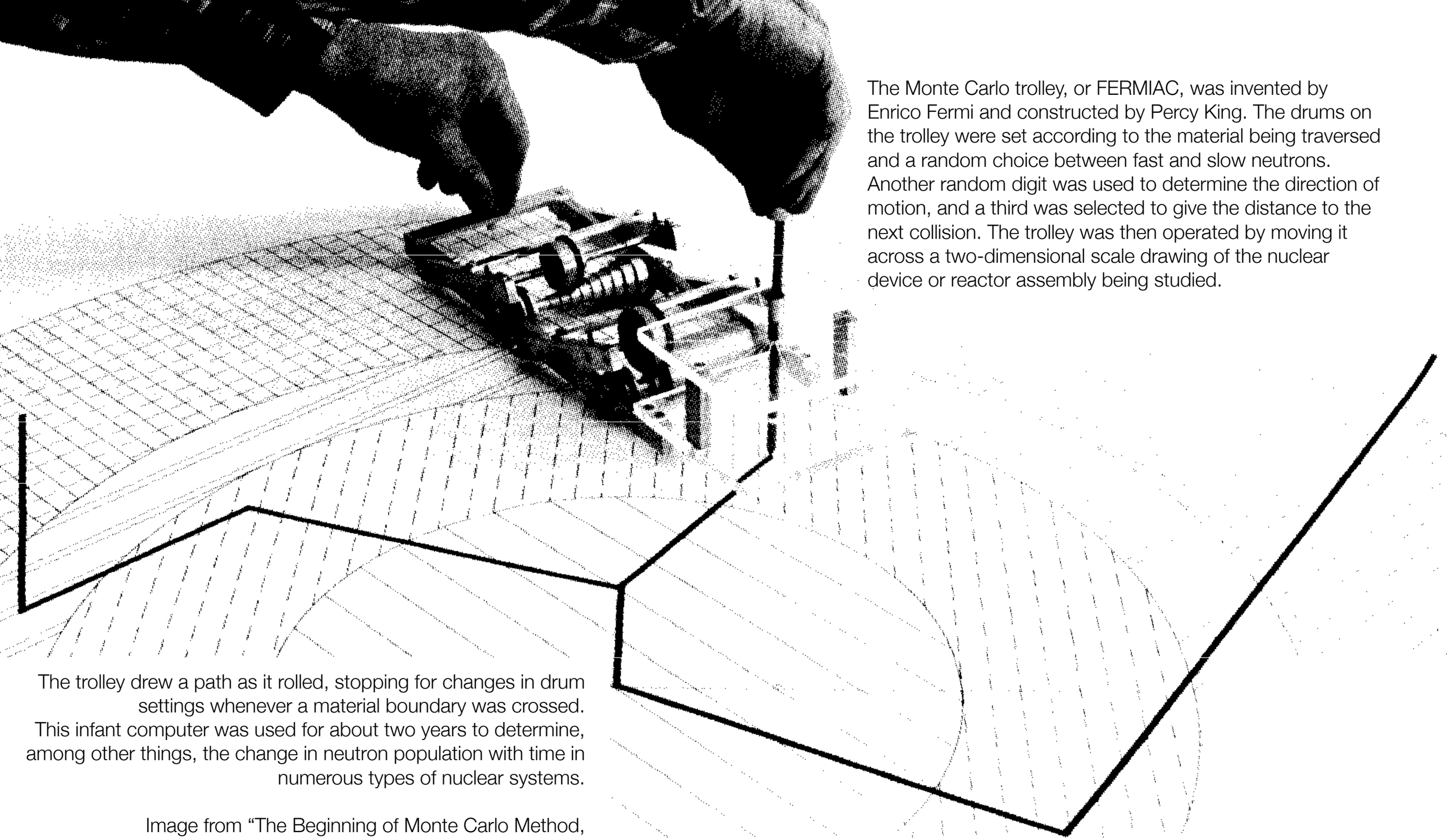
INFN
Istituto Nazionale di Fisica Nucleare

Overview

- The Monte Carlo method
- A short introduction to Geant4
  - The transport of particle
  - Geometry and scoring
  - Running an example
- Tips for the Geant4 installation
- You can download examples and slides from: http://www.roma1.infn.it/~mancinit/Teaching/ASP2018/

For these slides I took inspiration from:

- M. Asai (SLAC, Stanford)
- A. Dotti (SLAC, Stanford)
- S. Incerti (CNRS, Bordeaux)
- L. Pandola (INFN-LNS, Catania)
- C. Pistillo (LHEP, Bern)
- S. Rahatlou (Sapienza, Roma)
- www.cplusplus.com



The Monte Carlo trolley, or FERMIAC, was invented by Enrico Fermi and constructed by Percy King. The drums on the trolley were set according to the material being traversed and a random choice between fast and slow neutrons. Another random digit was used to determine the direction of motion, and a third was selected to give the distance to the next collision. The trolley was then operated by moving it across a two-dimensional scale drawing of the nuclear device or reactor assembly being studied.

The trolley drew a path as it rolled, stopping for changes in drum settings whenever a material boundary was crossed. This infant computer was used for about two years to determine, among other things, the change in neutron population with time in numerous types of nuclear systems.

Image from “The Beginning of Monte Carlo Method, N. Metropolis 1987

Introduction to the Monte Carlo method

The “FERMIAC”

How to calculate an integral

- Randomly choose couple of numbers (x_i, y_i) from the range and the domain, respectively, of the function f

- The fraction of points where $y_i \leq f(x_i)$ is equal to the fraction of the area below the function

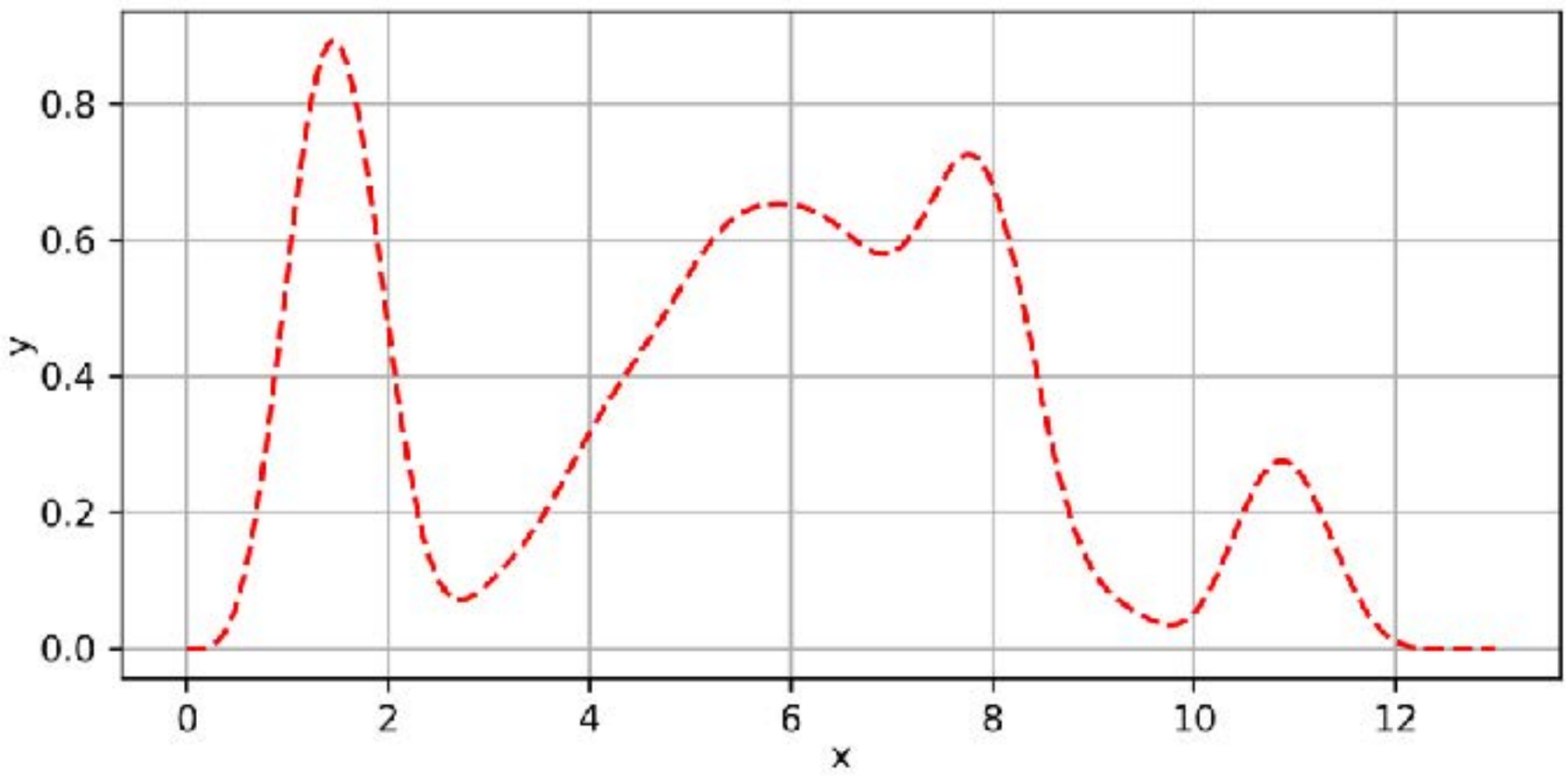
- Technique proposed by Von Neumann, known as the “acceptance-rejection method”

- It is used to generate random numbers for an arbitrary Probability Density Function (PDF)

Example of an integral

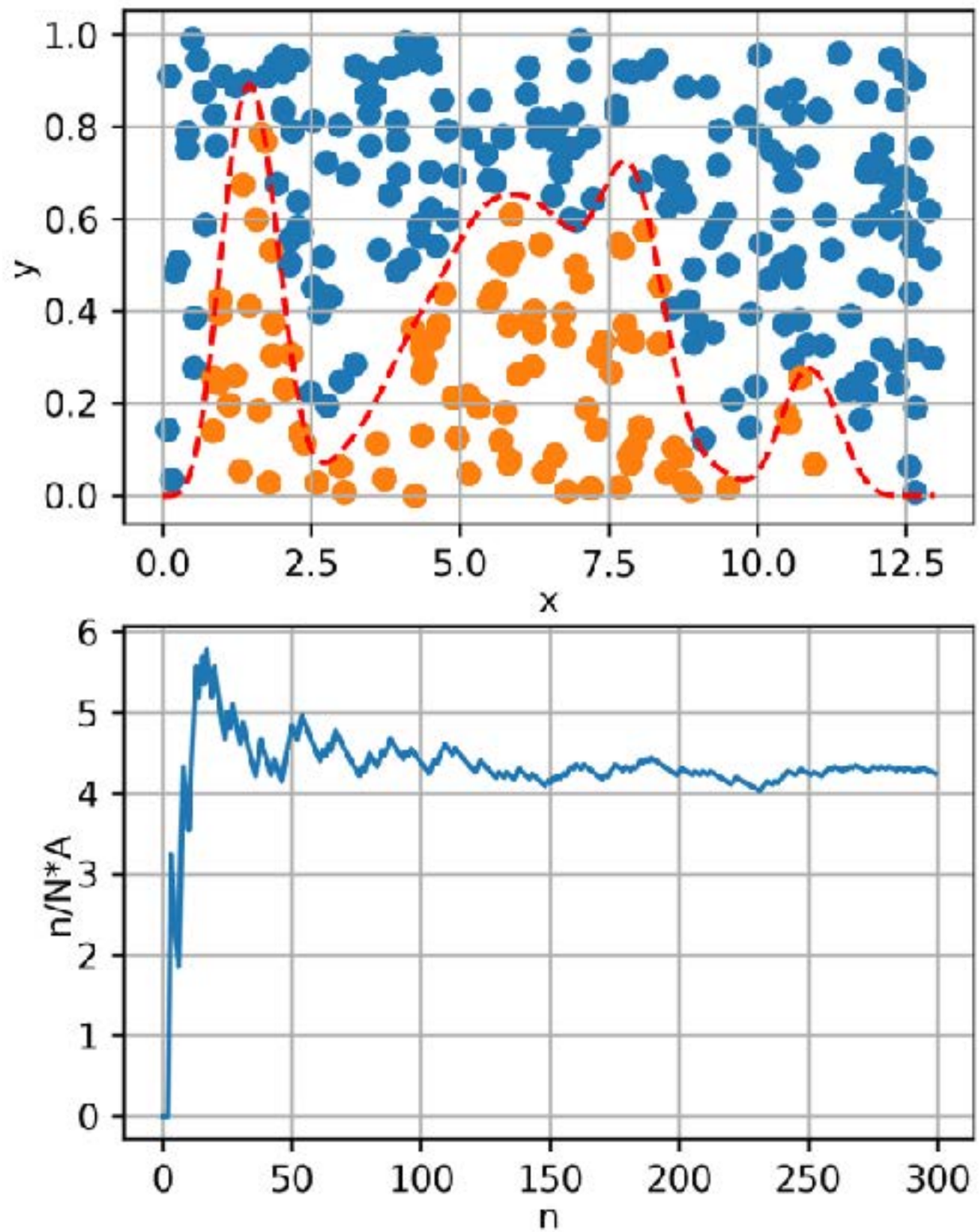
- What if you have to calculate the integral of a function as:

$$f(x) = 1.4 \left[\sin^4(x) \cos^2\left(\frac{x}{3}\right) + \sin^6\left(\frac{x}{4}\right) \right] \exp\left(-\frac{x}{8}\right)$$



Example of an integral

- Using the acceptance-rejection method
- The orange points are the accepted (*n*)
- The blue are the rejected (N = accepted + rejected)



Let’s run the example

- mkdir example
- cd example
- wget http://www.roma1.infn.it/~mancinit/Teaching/ASP2018/integral.py
- python integral.py

Let's calculate π

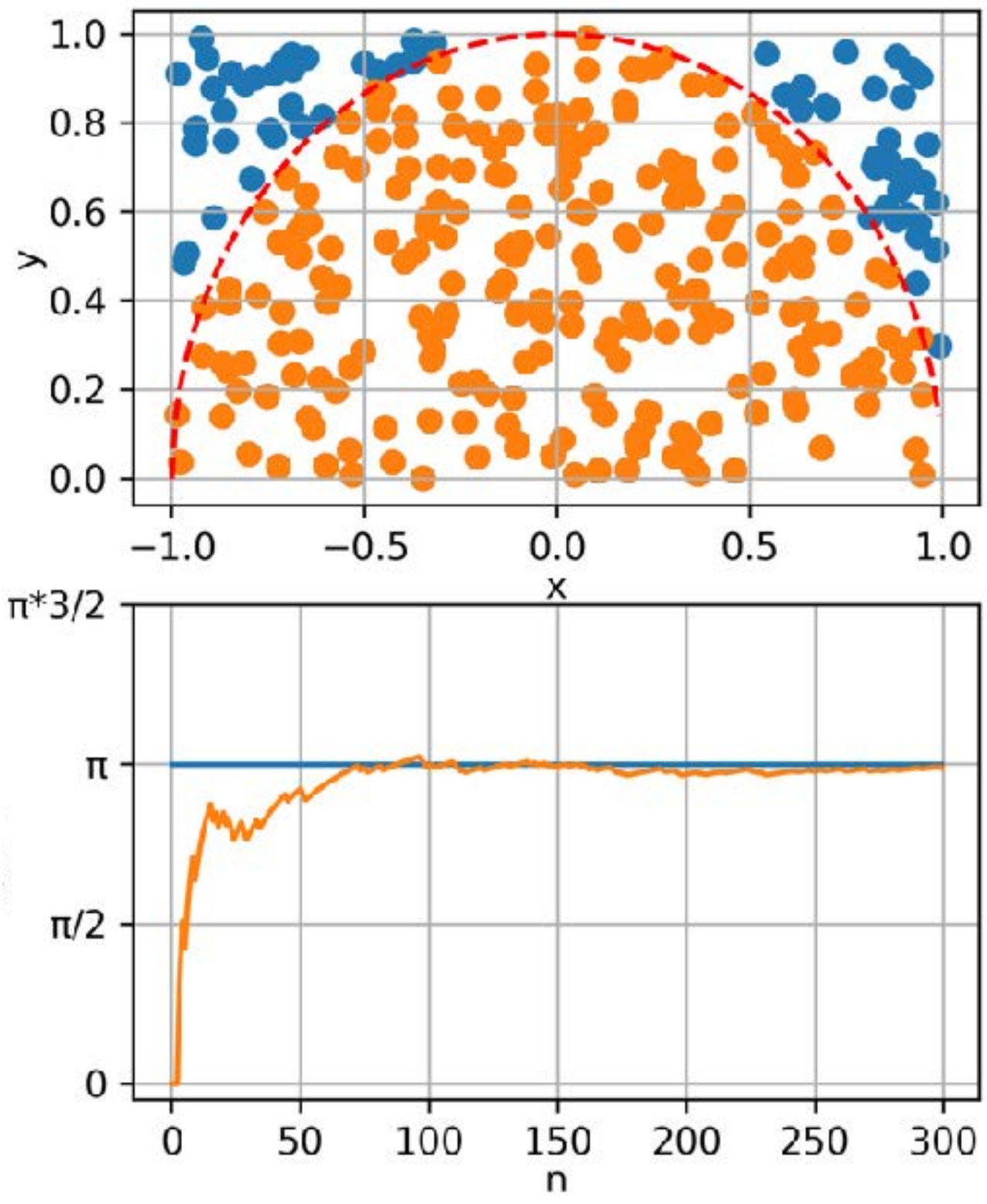
$$f(x) = \sqrt{(1 - x^2)}$$

$$A_{circ} = \int_{-1}^{1} f(x)dx = \pi\frac{r^2}{2}$$

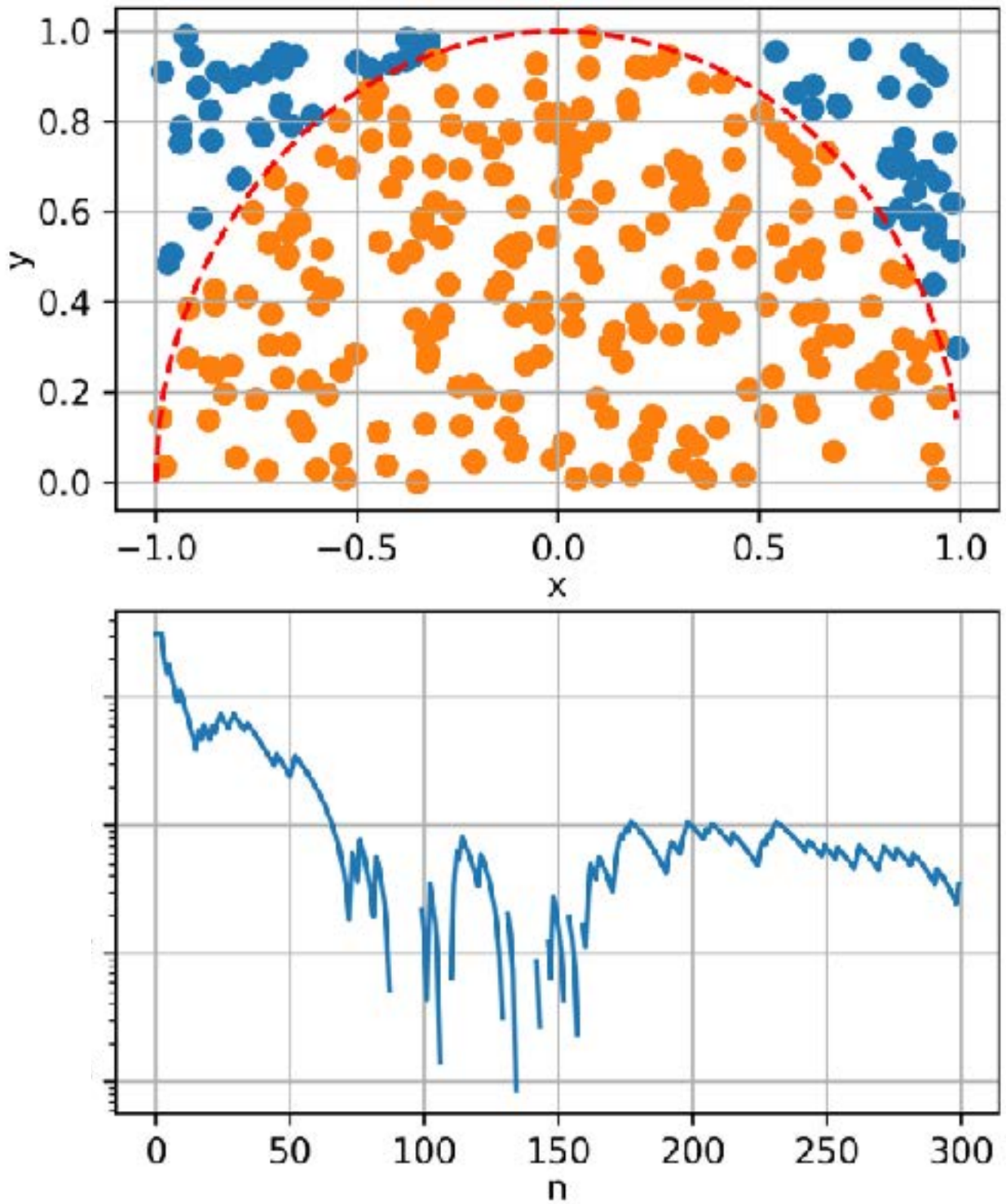
$$A_{rect} = \Delta y \cdot \Delta x$$

$$\frac{n}{N} \propto \frac{A_{circ}}{A_{rect}} = \frac{\pi r^2/2}{\Delta x \cdot \Delta y}$$

$$\pi \approx 2\frac{n}{N}\frac{\Delta x \cdot \Delta y}{r^2} = 4\frac{n}{N}$$

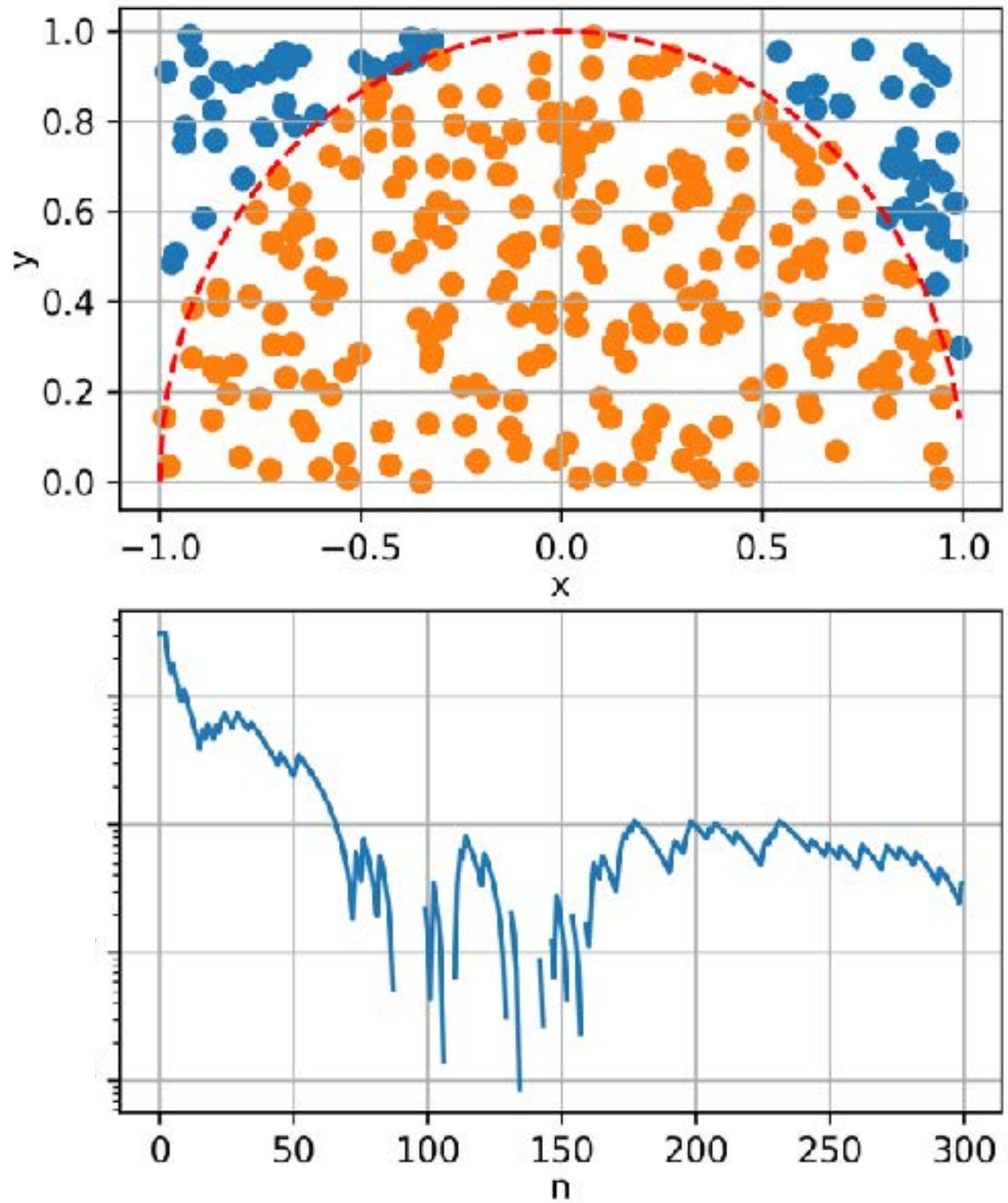


Let’s calculate π



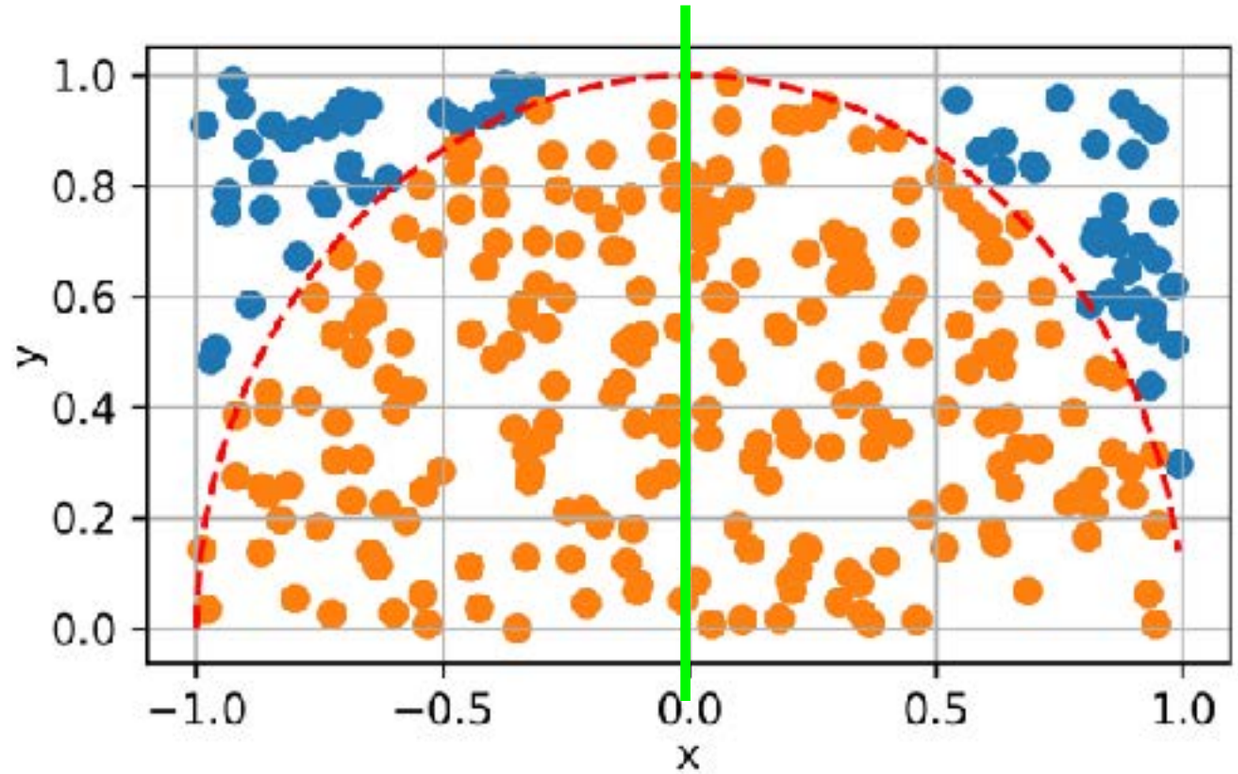
Let’s calculate π

- Is there a way to speed up the convergence of the computation?



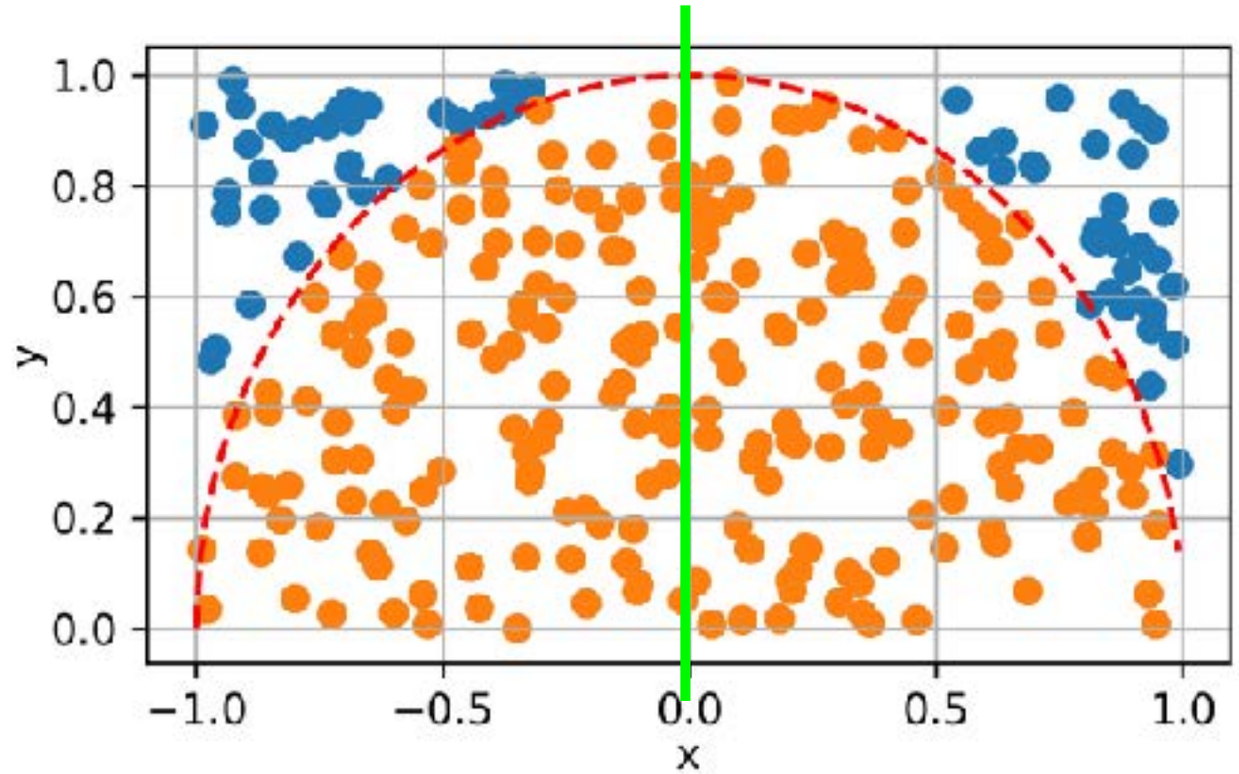
Let’s calculate π

- Is there a way to speed up the convergence of the computation?
- Use the symmetry!



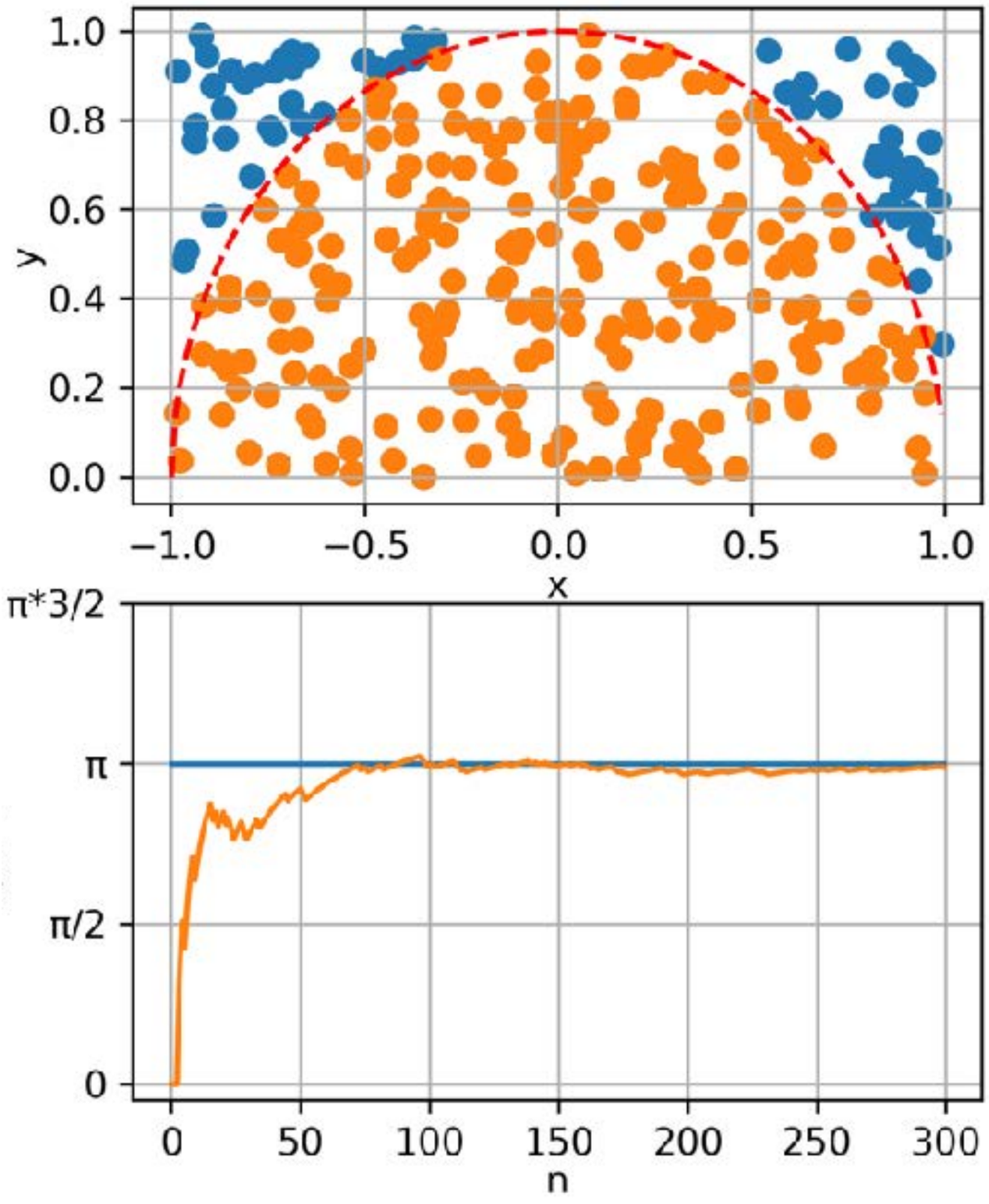
Let’s calculate π

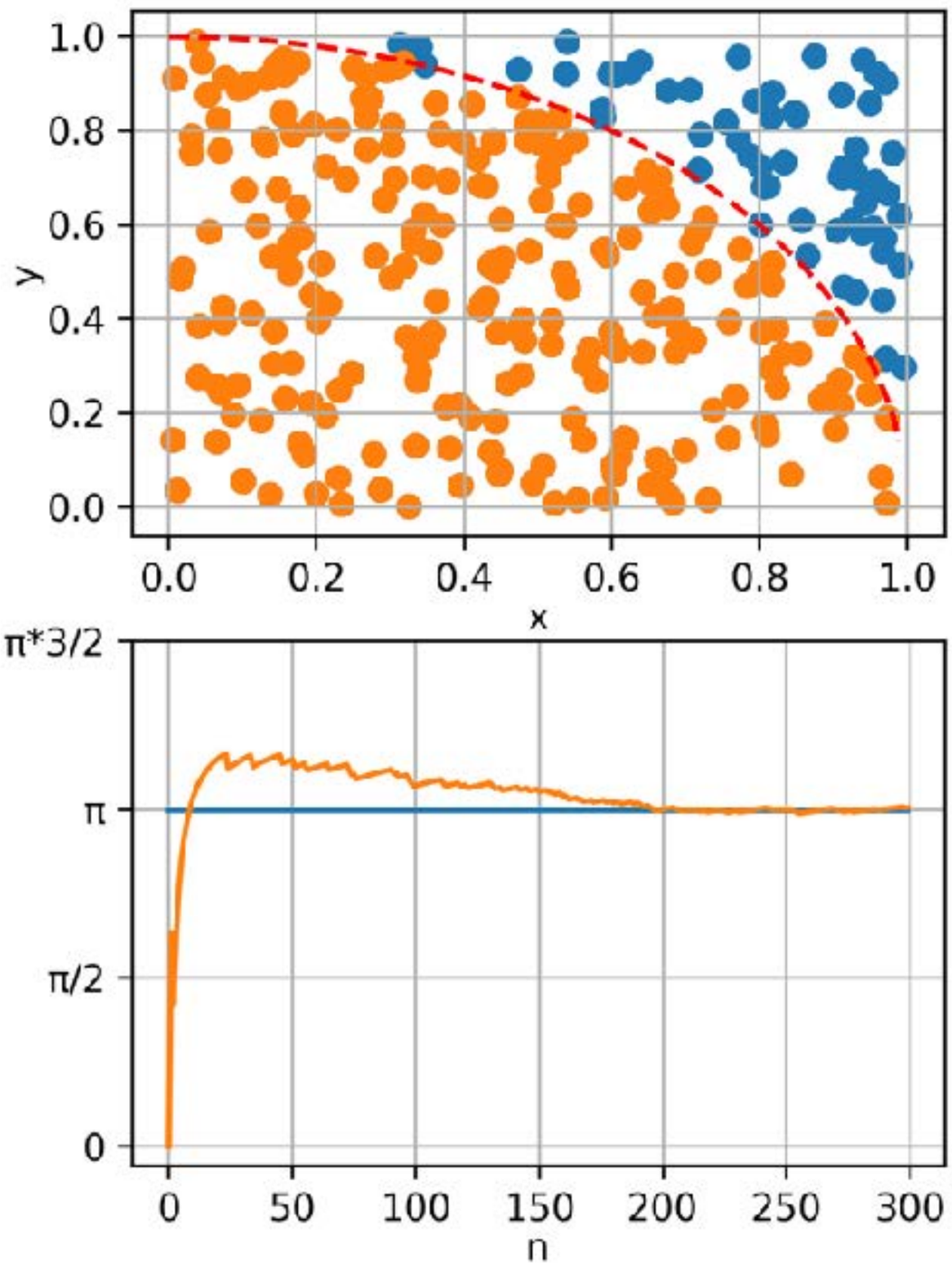
- Is there a way to speed up the convergence of the computation?
- Use the symmetry!



- This is the method for calculating π was proposed by Laplace in “Théorie Analytique des Probabilitiés” (1825)!

Use the simmetry!





Let’s run the example

- mkdir example
- cd example
- wget http://www.roma1.infn.it/~mancinit/Teaching/ASP2018/calculatepi.py
- python calculatepi.py

Random Numbers Generators

Random Numbers Generators

- At the core of all Monte Carlo calculations is some mechanism to produce a long sequence of random numbers r_i that are uniformly distributed over the open interval *[0,1)*

Random Numbers Generators

- At the core of all Monte Carlo calculations is some mechanism to produce a long sequence of random numbers r_i that are uniformly distributed over the open interval *[0,1)*

- Digital computers, by design, are incapable of producing random results

Random Numbers Generators

- At the core of all Monte Carlo calculations is some mechanism to produce a long sequence of random numbers r_i that are uniformly distributed over the open interval *[0,1)*

- Digital computers, by design, are incapable of producing random results

- A true random sequence could, in principle, be obtained by coupling to our computer some external device that would produce a truly random signal

Random Numbers Generators

- At the core of all Monte Carlo calculations is some mechanism to produce a long sequence of random numbers r_i that are uniformly distributed over the open interval *[0,1)*

- Digital computers, by design, are incapable of producing random results

- A true random sequence could, in principle, be obtained by coupling to our computer some external device that would produce a truly random signal

- However, use of such a random number generator would not be practical!

Random Numbers Generators

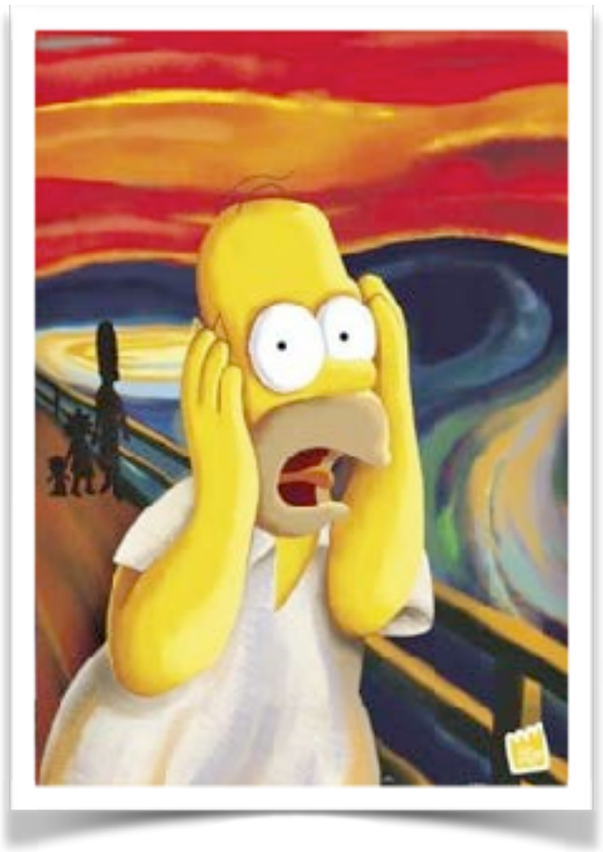
- At the core of all Monte Carlo calculations is some mechanism to produce a long sequence of random numbers r_i that are uniformly distributed over the open interval *[0,1)*

- Digital computers, by design, are incapable of producing random results

- A true random sequence could, in principle, be obtained by coupling to our computer some external device that would produce a truly random signal

- However, use of such a random number generator would not be practical!

- Imposible to debug!



Pseudo-random Number Generators

- Such a generator is a deterministic algorithm that, given the previous numbers (usually just the last number) in the sequence, the next number can be efficiently calculated

$$x_{n+1} = f(x_n, x_{n-1}, ..., x_0)$$

- x_0 is called "**seed**"

- A unique seed returns a unique random number sequence

- It is important to use a new seed every time that a random selection is initiated

- A typical error is the use of the same seed for multiple generation, which leads to the generation of the same sample of random numbers

Simple case: decay in flight

- Suppose a π^+ with momentum *p*

- The life time is a random value with a pdf

$$f(t) = \frac{1}{\tau} \exp\left(-\frac{1}{\tau}\right)$$

- Therefore, *t* can be sampled from the inverse of the cumulative:

$$t = F^{-1}(r) = -\tau \ln(1 - r)$$

$$r \in [0, 1)$$

Simple case: decay in flight

- Select the decay channel:

Mode			Fraction (Γ_i / Γ)
Γ_1	$\mu^+\nu_\mu$	[1]	$(99.98770 \pm 0.00004)\%$
Γ_2	$\mu^+\nu_\mu\gamma$	[2]	$(2.00 \pm 0.25) \times 10^{-4}$
Γ_3	$e^+\nu_e$	[1]	$(1.230 \pm 0.004) \times 10^{-4}$
Γ_4	$e^+\nu_e\gamma$	[2]	$(7.39 \pm 0.05) \times 10^{-7}$
Γ_5	$e^+\nu_e\pi^0$		$(1.036 \pm 0.006) \times 10^{-8}$
Γ_6	$e^+\nu_e e^+e^-$		$(3.2 \pm 0.5) \times 10^{-9}$
Γ_7	$e^+\nu_e\nu\overline{\nu}$		$< 5 \times 10^{-6}$

[table from PDG]

- In the CM frame the decay is isotropic

$$\theta \in [0, \pi); \quad \phi \in [0, 2\pi)$$

- Finally, Lorentz-boost in the Lab. frame

- 4 random numbers for one decay!

Problem

Problem

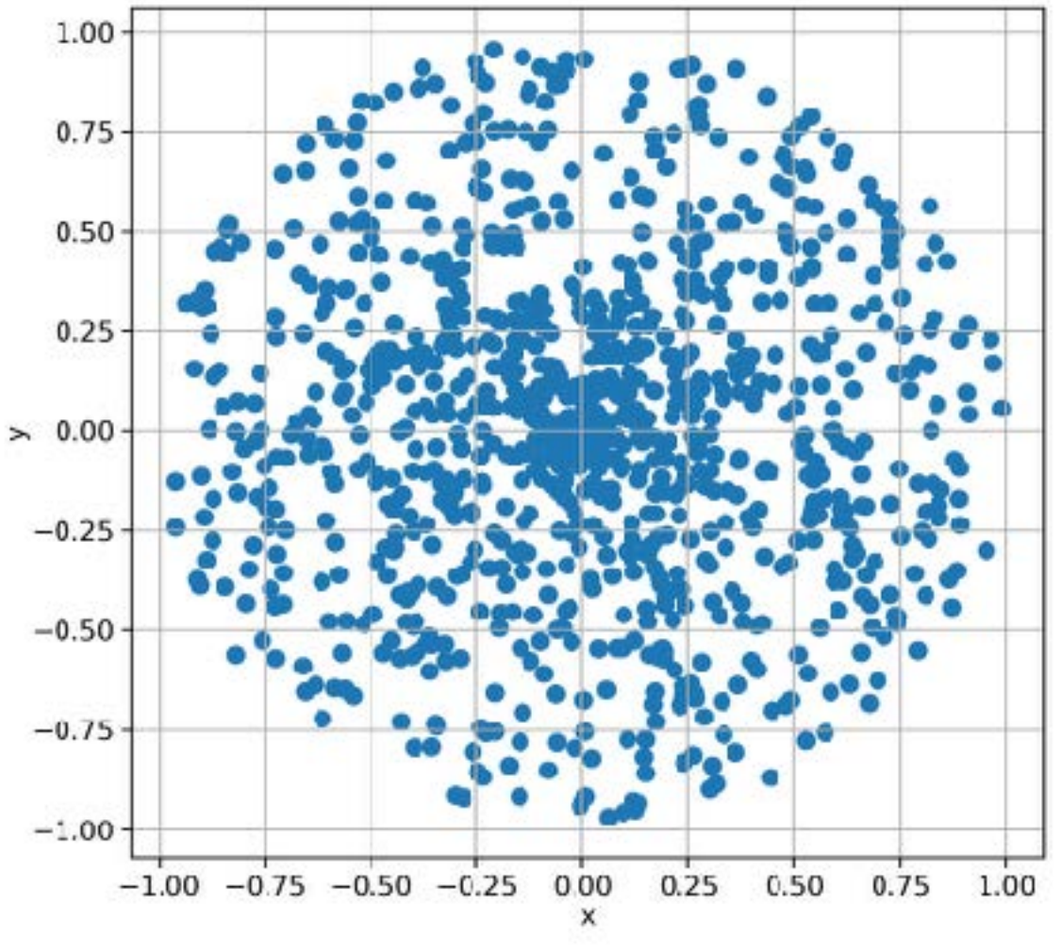
- Why did I sampled θ and ϕ in the CM frame?

Problem

- Why did I sampled θ and ϕ in the CM frame?
- What if I sample uniformly $\theta \in [0, 2\pi); \quad r \in [0, 1)$?

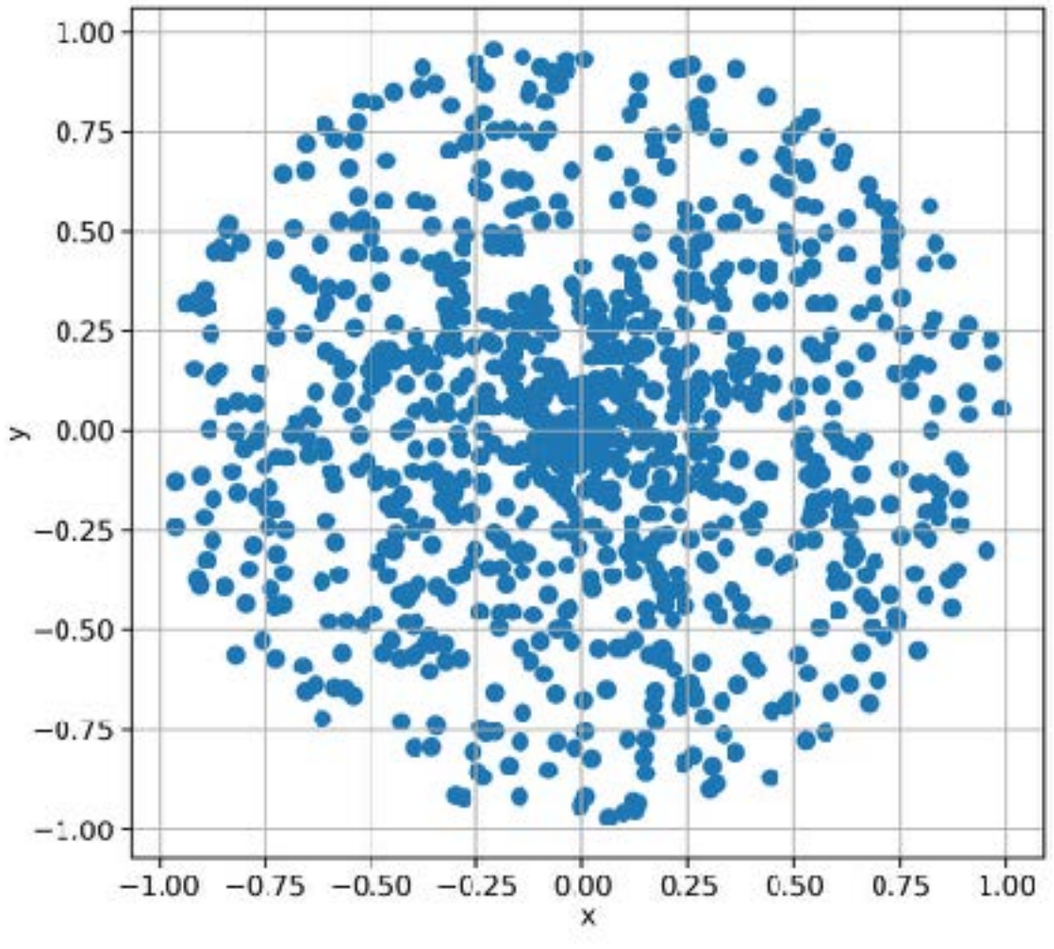
Problem

- Why did I sampled θ and ϕ in the CM frame?
- What if I sample uniformly $\theta \in [0, 2\pi); \quad r \in [0, 1)$?
- The extracted points gather in the centre



Problem

- Why did I sampled θ and ϕ in the CM frame?
- What if I sample uniformly $\theta \in [0, 2\pi); \quad r \in [0, 1)$?
- The extracted points gather in the centre
- A uniform distribution in polar coordinates is not uniform in orthogonal coordinate system



Inverse transform sampling

- If a PDF f is integrable (called **cumulative**, F)
- and the cumulative is invertible F^{-1}
- It is possible to sample x accordingly to f:

$$x = F^{-1}(u)$$

where *u* is uniformly distributed

Particle tracking

- It is the most common application of MC in Particle Physic

- Assume that all the possible interactions are known

- The distance *s* between two subsequent interactions is distributed as $$p(s) = \mu \exp(-\mu s)$$

- Being μ a property of the medium

Particle tracking

- μ is proportional to the probability of an interaction per unit length, therefore:

  - is proportional to the **total cross section**

$$\mu = N\sigma = N\sum_i \sigma_i = \sum_i \mu_i$$

- μ_i are the partial cross section of **all the competing processes**

- depends on the **density** of the material
  (N is the number of scattering centres in the medium)

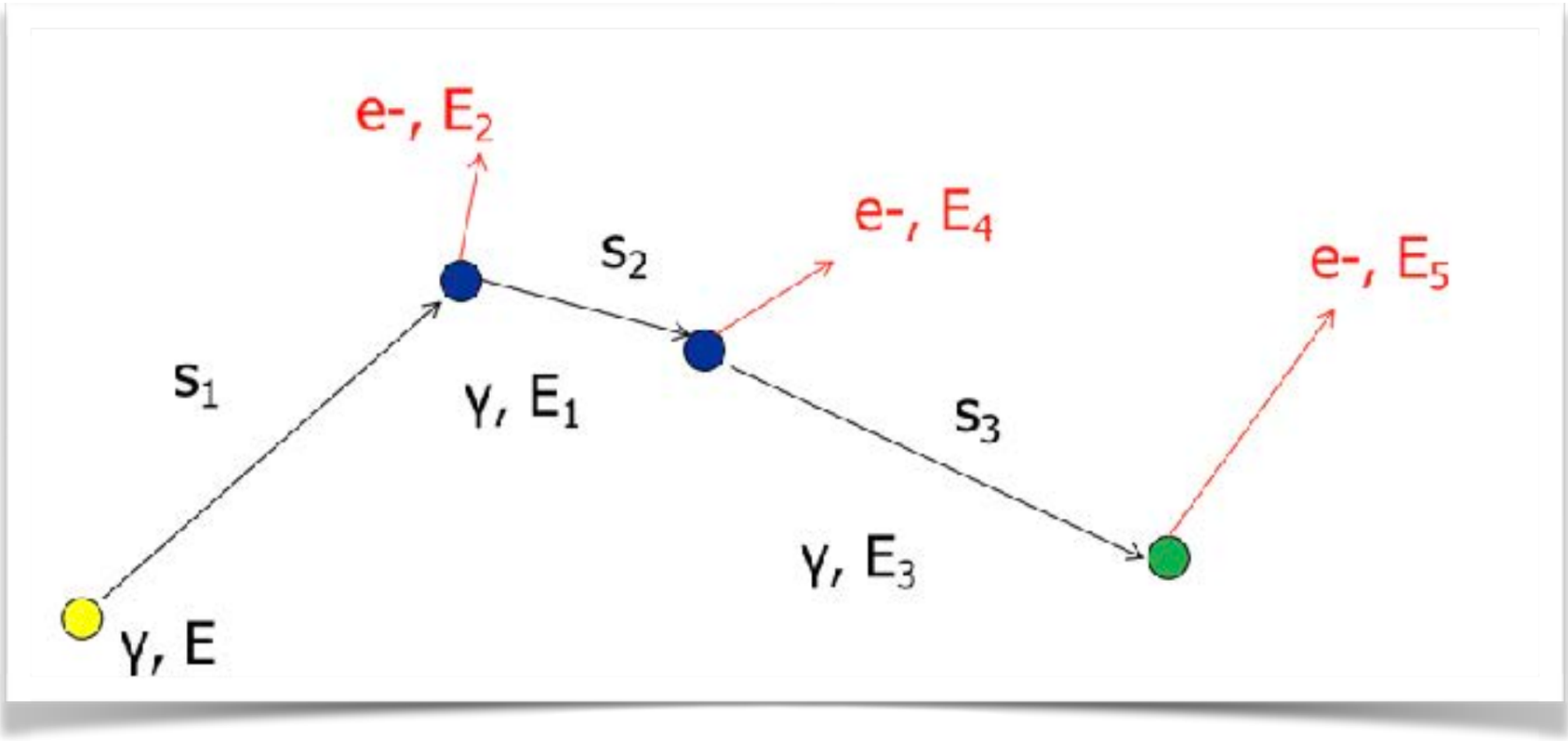
Particle tracking

- Divide the particle trajectory in **“steps”**
  - Straight free-flight tracks along the step
  - Could be limited by geometry boundaries
- Sampling the step length accordingly to $p(s)$
- Sampling the interaction at the end of the step
- Sampling the interaction accordingly to μ_i / μ
- Sampling the final state using the physics model of the interaction i
  - Update the properties of the primary particle
  - Add the possible secondaries produced (to be tracked later)

Particle tracking

- Follow all secondaries, until absorbed
  (or leave the geometry)

- μ depends on the energy (cross sections do!)



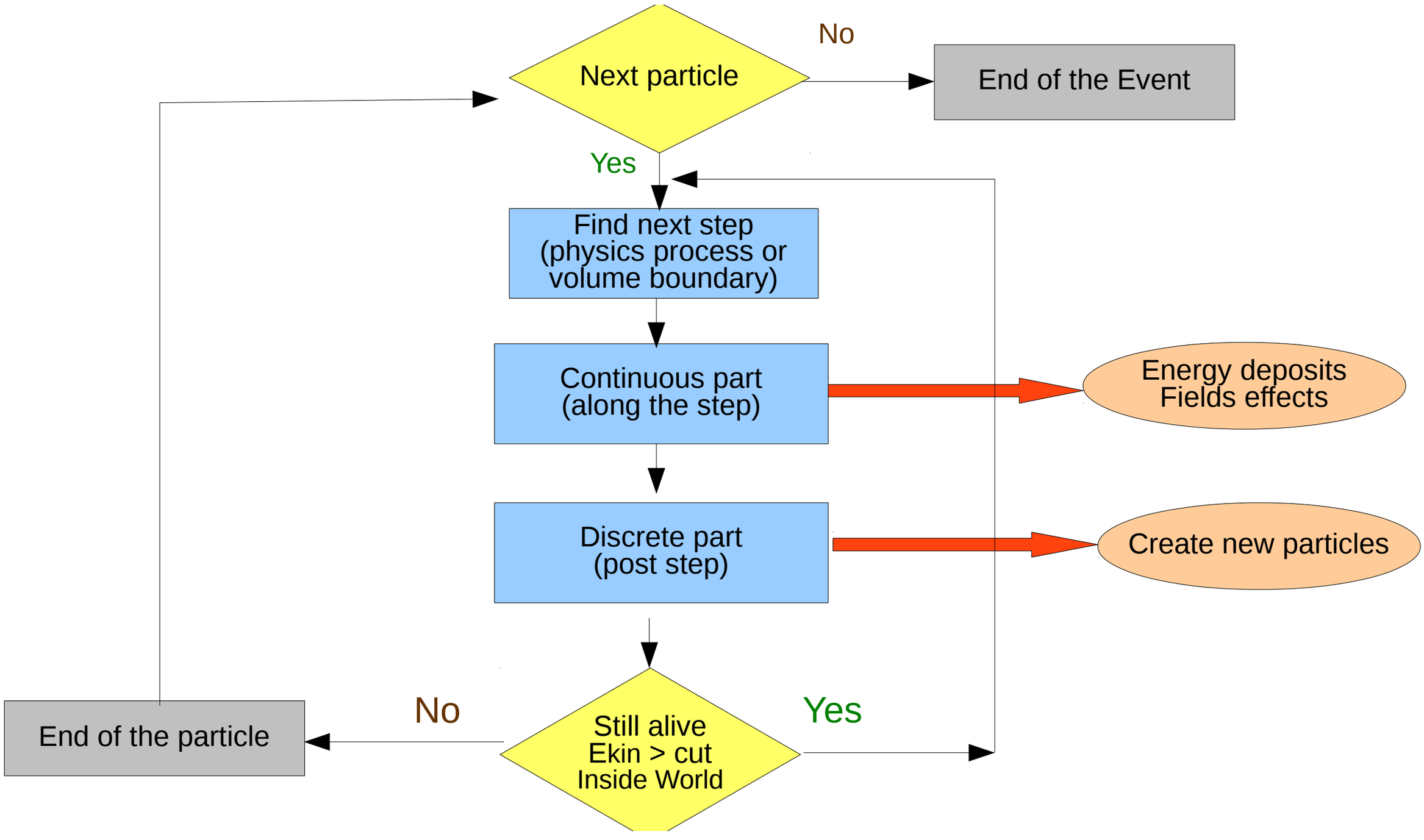
Tracking, not so easy…

- This basic recipe doesn't work well for charged particles
- The **cross sections** of some processes (ionisation and bremsstrahlung) **is very high**, so the **steps** would be very **small**
- In each interaction **only a small fraction of energy is lost** and the effect on the particle are small
- A lot of CPU time used to simulate many interactions having small effects

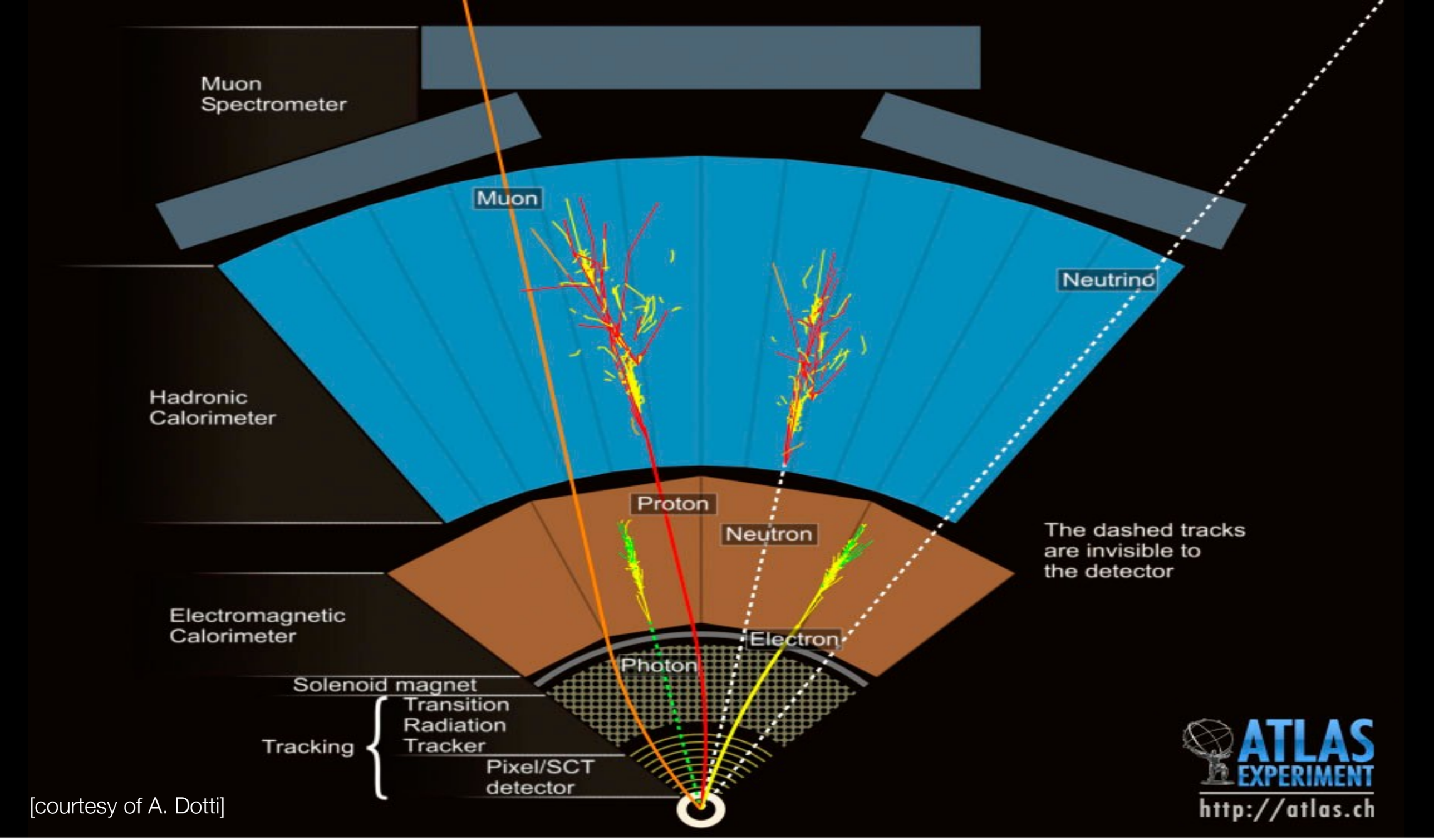
The solution: approximate

- Simulate explicitly interactions only if the energy loss is above a threshold E_0 (**hard** interactions)
  - Detailed simulation
- The effects of all sub-threshold interactions is described cumulatively (**soft** interactions)
- Hard interactions occur much less frequently than soft interactions

Flowchart of an event



…luckily enough, somebody else already implemented the tracking algorithms for us (and much more)



[courtesy of A. Dotti]

A short introduction to Geant4

A sketch of the ATLAS MC simulation

Geant4 (GEometry ANd Traking)

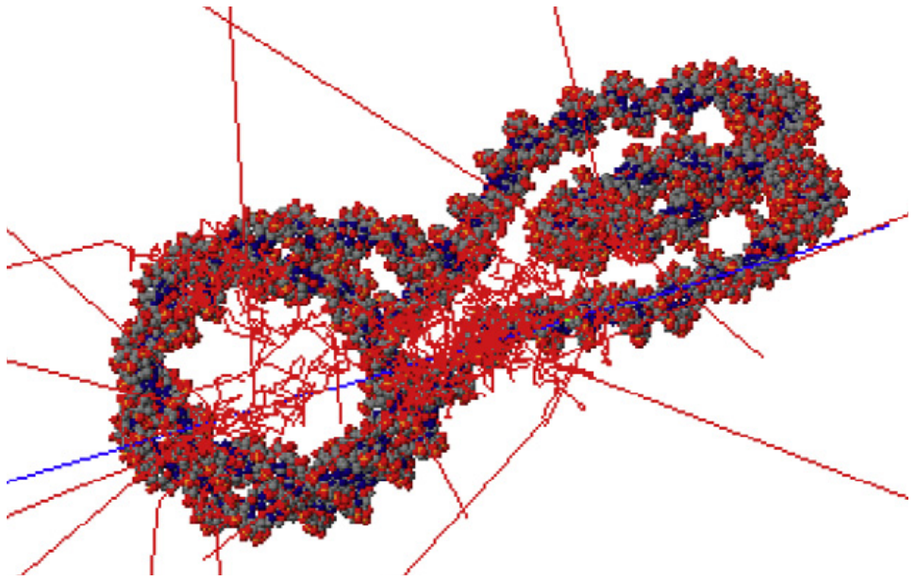
- Developed by an International Collaboration
  - Established in 1998
  - Approximately 100 members, from Europe, US and Japan
  - http://geant4.org
- Open source
- Written in C++ language
  - Takes advantage from the Object Oriented software technology

[Geant4, a simulation toolkit Nucl. Inst. and Methods Phys. Res. A, 506 250-303

Geant4 developments and applications Transaction on Nuclear Science 53, 270-278]

Geant4 applications

- Physics experiments
- but also:
  - Hadrontherapy
  - Radiobiology
- and many others…



atomistic view of a dinucleosome irradiated by a single 100 keV proton

Image from M. A. Bernal et al Physica Medica, vol. 31, no. 8, pp. 861–874, Dec. 2015.

Geant4, further applications

- Radio-protection in space mission
- Shielding for satellites
- Single event upset and radiation damages to electronics
- Simulations for nuclear spallation sources
- Radioactive waste

First slide of the talk “ESA Geant4 R&D Activities from the Geant4 Space User Workshop Hiroshima, 26 August 2015

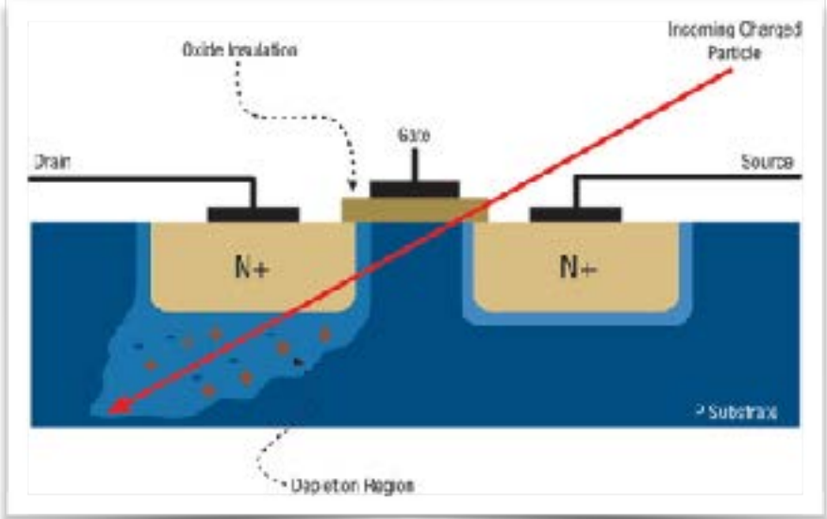


Figure from M. Sawant, COTS Journal Jan. 2012

Geant4 is a toolkit

- Geant4 is a **toolkit** (= a collection of tools)
  - i.e. you **cannot** run it out of the box
  - You **must write an application**, which uses Geant4
- Consequences:
  - There are **no** such concepts as “Geant4 **defaults**”
  - You must provide the necessary information to configure your simulation
  - You must deliberately choose which Geant4 tools to use

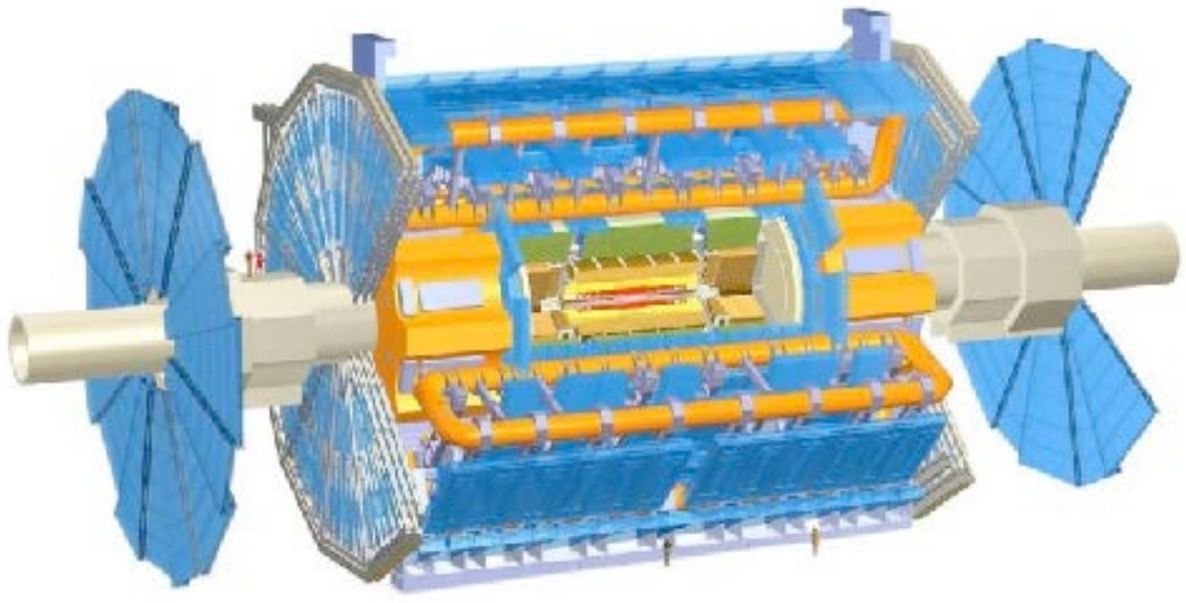
- Guidance: many examples are provided
  - **Basic Examples**: overview of Geant4 tools
  - Advanced Examples: Geant4 tools in real-life applications

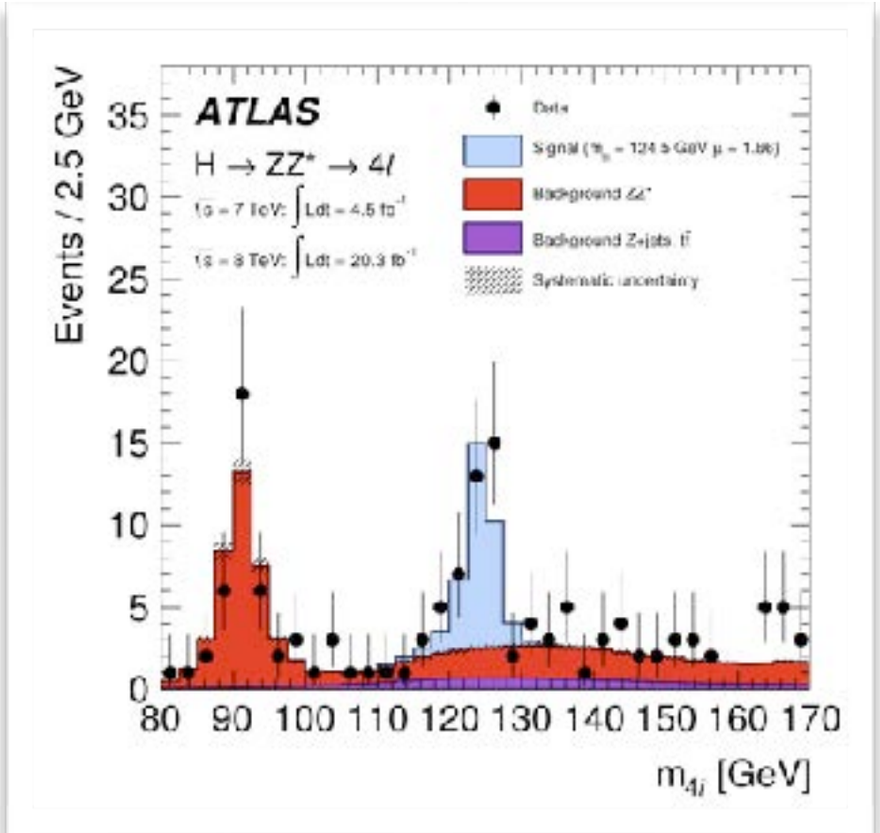
You MUST:

- Describe your experimental set-up
- Provide the primary particles input to your simulation
- Decide **which particles** and **physics models** you want to use out of those available in Geant4 and the precision of your simulation (cuts to produce and track secondary particles)

You may also want to:

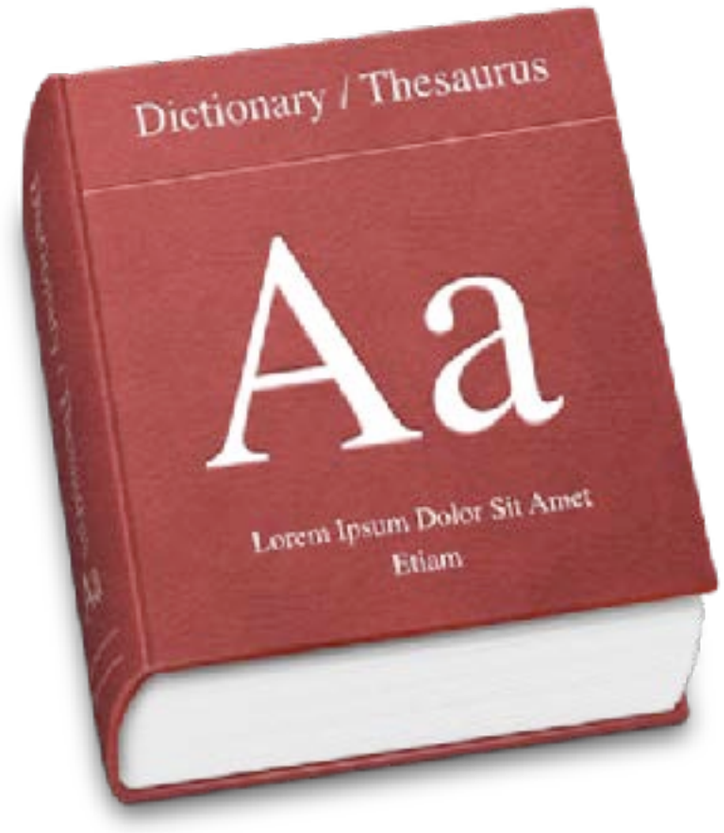
- Interact with Geant4 kernel to control your simulation
- Visualise your simulation configuration or results
- Produce histograms, tuples etc. to be further analysed





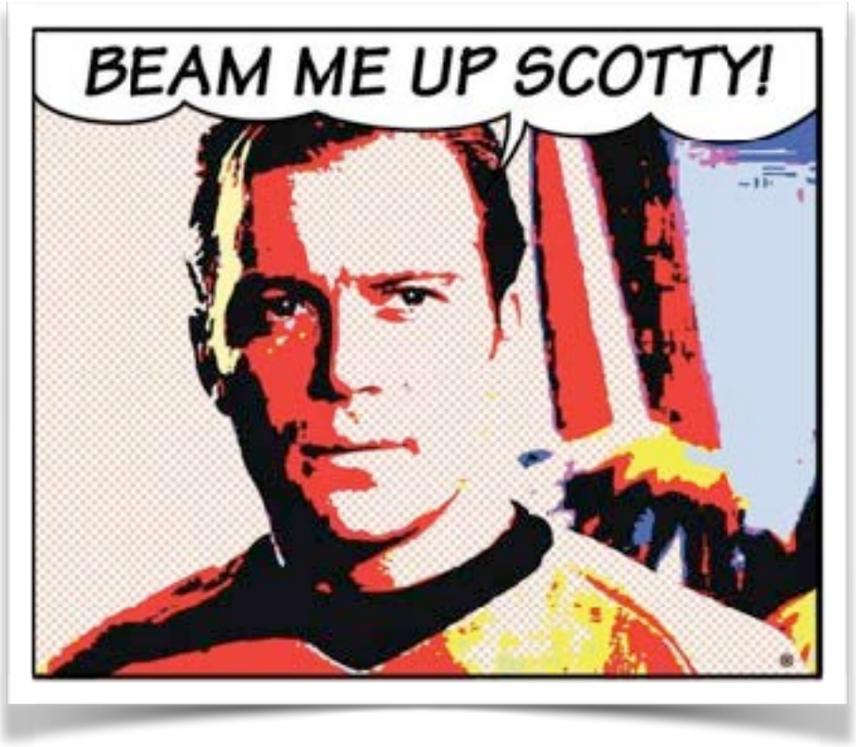
Jargons

- Run, event, track, step, step point
- Process
  - At rest, along step, post step
- Cut = production threshold
- Sensitive detector, score, hit, hits collection,



A run in Geant4

- As an analogy of the real experiment, a run of Geant4 starts with **“Beam On”**

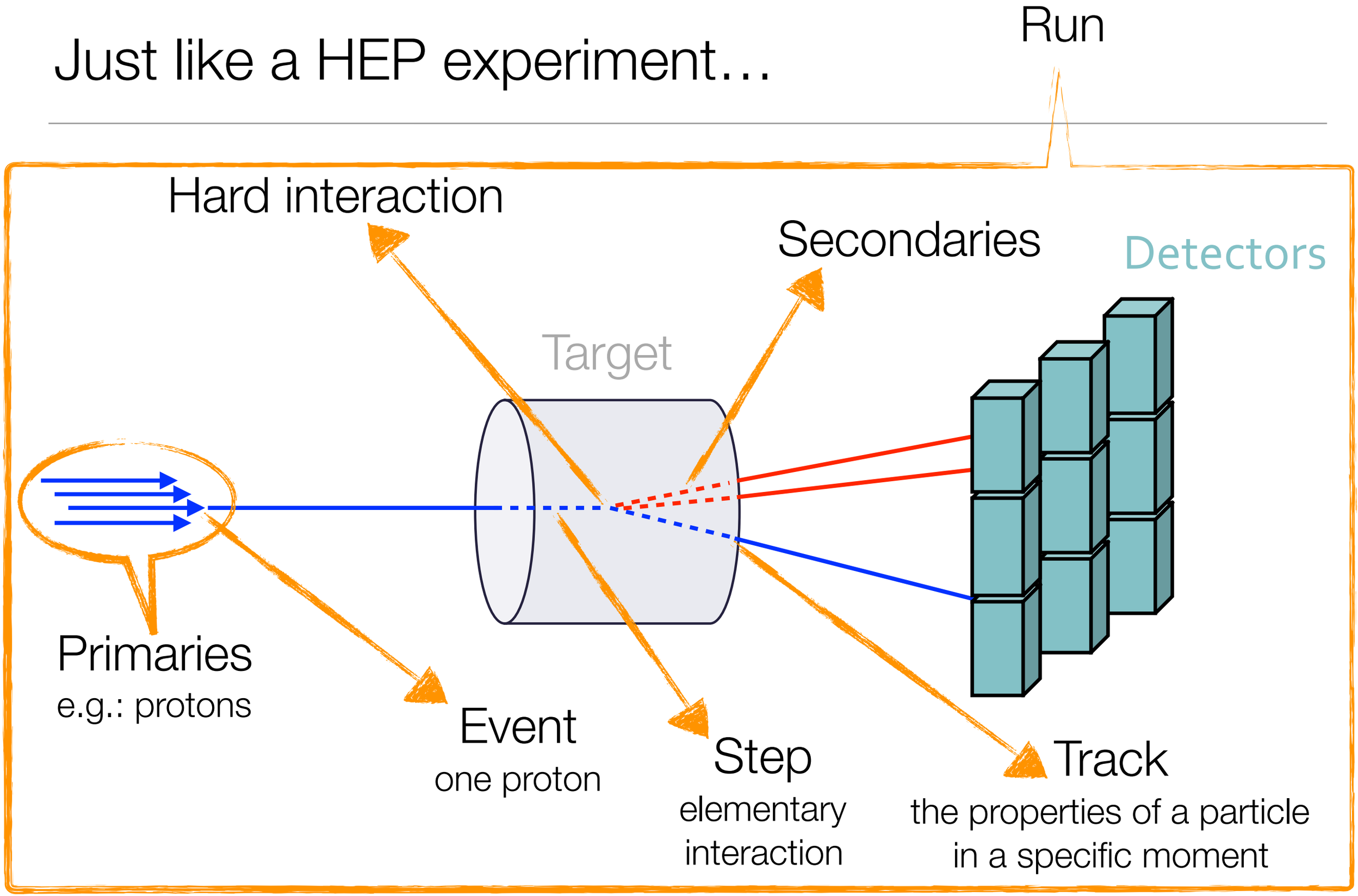


- A run is a collection of events which share the same detector and physics conditions

- **G4RunManager** class manages processing a run

- **G4UserRunAction** is the optional user hook

Just like a HEP experiment…



Let’s run an example

- git clone https://github.com/carlomt/AnaEx01.git
- mkdir anaEx01build
- cd anaEx01build
- cmake -DGeant4_DIR=G4INSTALLDIR ../AnaEx01
- make

Particle in Geant4

- A particle in Geant4 is represented by three layers of classes:
  - **G4Track**
    - Geometrical information (position)

  - **G4DynamicalParticle**
    - Dynamic physical properties (momentum, energy, spin…)
  - **G4ParticleDefinition**
    - Static properties (charge, mass, life time)

Sampling the step size

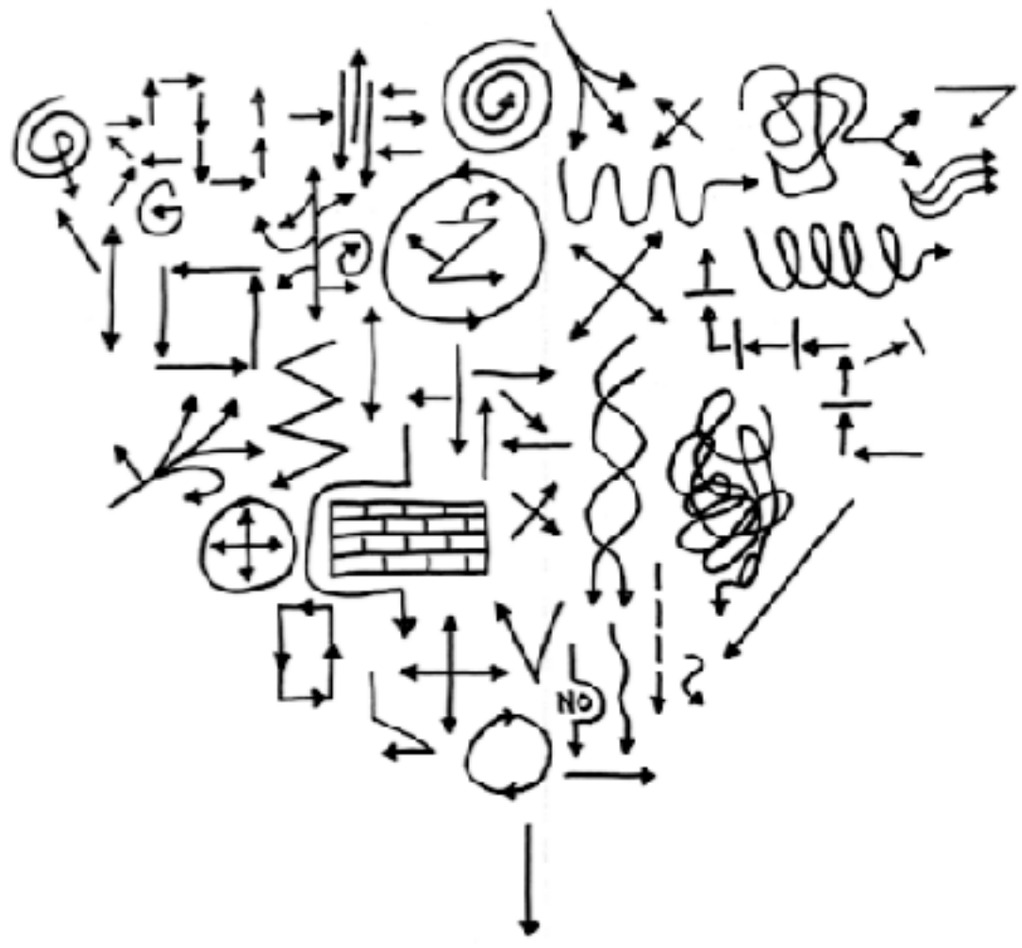
- In Geant4, particle transportation is a process as well, by which a particle interacts with geometrical volume boundaries and field of any kind

- Each particle has its own list of applicable processes. At each step, all processes listed are invoked to get proposed physical interaction lengths

- The process which requires the shortest interaction length limits the step

Process kinds

- At rest
  - for instance: muon decay
- Along the step
  - like Cherenkov
- Post step
  - decay on the fly, hard interactions



Let’s cut it out… (cuts in Geant4)

- A Cut in Geant4 is a production threshold
- It is applied only for physics processes that have infrared divergence

System of Units

- Internal unit system used in Geant4 is completely hidden not only from user's code but also from Geant4 source code implementation

- Each hard-coded number must be multiplied by its proper unit:
  - radius = 10.0 * cm; E = 1.*GeV;

- To get a number, it must be divided by a proper unit:
  - G4cout<< "E dep: "<< eDep/MeV <<" [MeV]"<<G4endl;

User classes

- You **have** to write the **main**()
- Initialisation classes:
  - **G4VUserDetectorConstruction**
  - **G4VUserPhysicsList**
  - **G4VUserActionInitialization**
- Action classes
  - **G4VUserPrimaryGeneratorAction**
  - **G4UserRunAction**
  - **G4UserEventAction**
  - **G4UserStackingAction**
  - **G4UserTrackinAction**
  - **G4UserSteppingAction**

classes written in red are mandatory!

Physics List

- A class which collects all the particles, physics processes and production thresholds needed for your application
- It tells the run manager how and when to invoke physics
- It is a very flexible way to build a physics environment
  - user can pick the particles he wants
  - user can pick the physics to assign to each particle
- But, **user must have a good understanding of the physics required**
- omission of particles or physics could cause errors or poor simulation

There is not default, but…

- Geant4 provides several “production physics lists” which are routinely validated and updated with each release these should be considered only as starting points which you may need to validate or modify for your application

- There are currently 19 packaged physics lists available

- 6 reference physics lists:

  - FTFP_BERT, FTFP_BERT_HP QGSP_BERT, QGSP_BERT_HP, QGSP_BIC QGSP_FTFP_BERT

Naming…

- The following acronyms refer to various hadronic options
  - QGS -> Quark Gluon String model (>~20 GeV) FTF -> Fritiof string model (>~5 GeV)
    BIC -> Binary Cascade (<~ 10 GeV)
    BERT -> Bertini-style cascade (<~ 10 GeV)
- P -> G4Precompund model used for de-excitation
- HP -> High Precision neutron model (< 20 MeV)
- EM options designated by
  - no suffix: standard EM physics
  - EMV suffix: older but faster EM processes
    – other suffixes for other EM options

Production Physics Lists

- FTFP_BERT
  - recommended by Geant4 for HEP
  - contains all standard EM processes
  - uses Bertini-style cascade for hadrons < 5 GeV
  - uses FTF (Fritiof) model for high energies (> 4 GeV)

- QGSP_BERT
  - all standard EM processes
  - Bertini-style cascade up to 9.9 GeV
  - QGS model for high energies (> ~18 GeV) FTF in between

Production Physics Lists

- QGSP_BIC
  - same as QGSP_BERT, but replaces Bertini cascade with Binary cascade and G4Precompound model
  - recommended for use at energies below 200 MeV (many medical applications)

- FTFP_BERT_HP
  - same as FTFP_BERT, but with high precision neutron model used for neutrons below 20 MeV
  - signifcantly slower than FTFP_BERT when full thermal cross sections used there’s an option to turn this of
  - for radiation protection and shielding applications

Other Physics Lists

- If primary particle energy in your application is < 5 GeV (for example, clinical proton beam of 150 MeV)
  - start with a physics list which includes BIC or BERT
  - e.g. QGSP_BIC, QGSP_BERT, FTFP_BERT, etc.

- If neutron transport is important
    - start with physics list containing “HP”
    - e.g. QGSP_BIC_HP, FTFP_BERT_HP, etc.

- If you’re interested in Bragg curve physics
    - use a physics list ending in “EMV” or “EMX” or “EMY”
    - e.g. QGSP_BIC_EMY

An event in Geant4

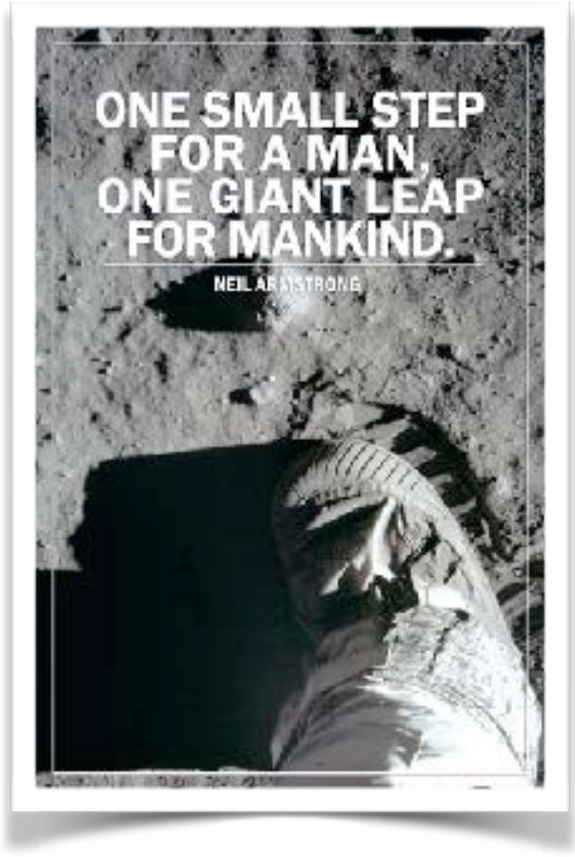
- An event is the basic unit of simulation in Geant4
- **G4Event** class represents an event. It has following objects at the end of its (successful) processing
  - List of primary vertices and particles (as input)
  - Hits and Trajectory collections (as output)
- **G4EventManager** class manages processing an event
- **G4UserEventAction** is the optional user hook

A track in Geant4

- Track is a snapshot of a particle
- It has physical quantities of current instance only. It does not record previous quantities
- It’s not a collection of steps. Instead, a track is being updated by steps
- G4TrackingManager manages processing a track, a track is represented by G4Track class
- G4UserTrackingAction is the optional user hook

A step in Geant4

- A step is a variation of a track
- Has two points (pre and post step points)
- In case a step is limited by a boundary, the end point stands on the boundary, and it logically belongs to the next volume
- Boundary processes such as transition radiation or refraction could be simulated



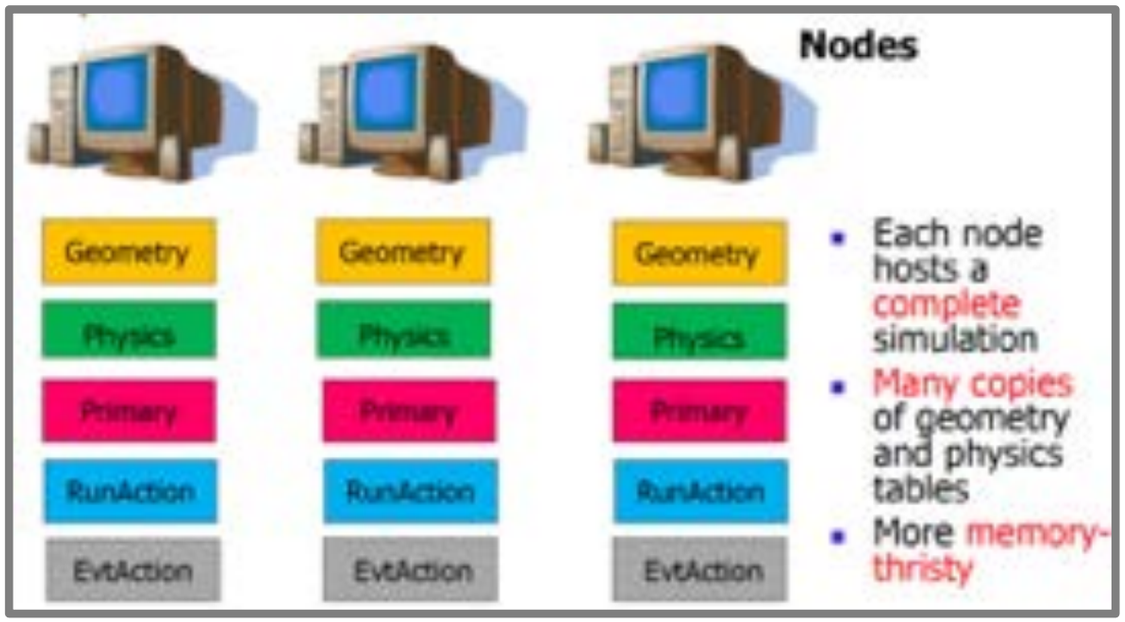
- **G4SteppingManager** class manages processing a step, a step is represented by **G4Step** class
- **G4UserSteppingAction** is the optional user hook

Your program

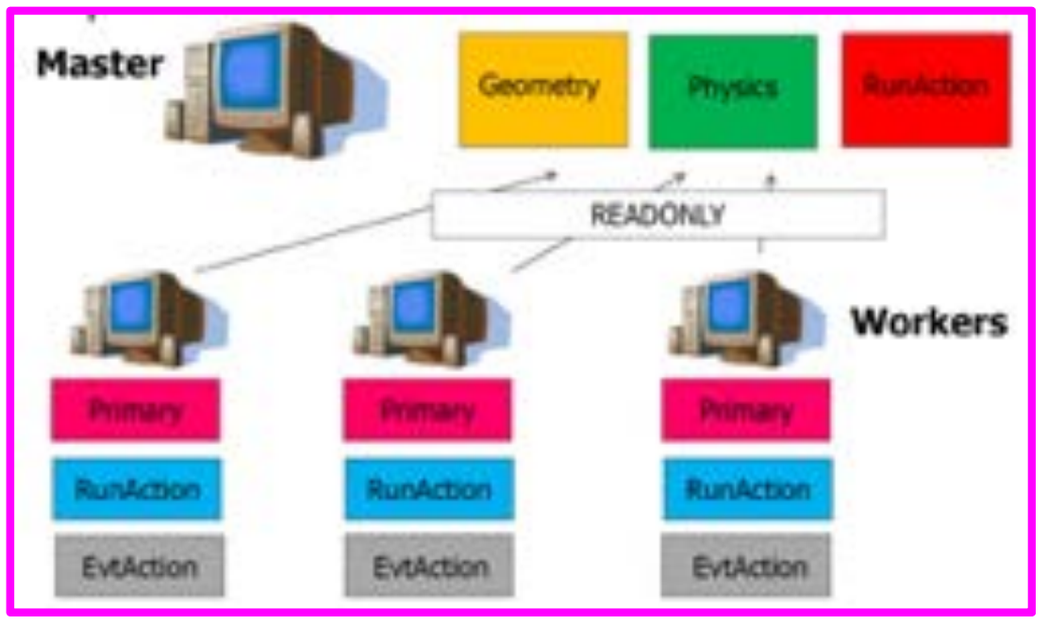
- Geant4 does not provide a main()
- In your main(), you have to
- Construct **G4RunManager** (sequential mode)
- Set user mandatory initialisation classes to RunManager
  - **G4VUserDetectorConstruction**
  - **G4VUserPhysicsList**
  - **G4VUserActionInitialization**
- You can define VisManager, (G)UI session,
- You can initialise optional user action classes

MultiThreading in Geant4

Parallel



Multithread



- Speed-up of simulations even on a laptop
- More efficient usage of memory: all cores only read geometry and physics of the Geant4 simulation (no duplication)

Prerequisites on CentOS7

- sudo yum update
- sudo yum groupinstall “Development Tools”
- sudo yum install xerces-c xerces-c-devel
- sudo yum install centos-release-scl-rh
- sudo yum install python27 python27-pyhton-pip python27-pyhton-tools python27-numpy
- sudo yum install qt qt-x11 qt-devel

CMake3 for CentOS7

- cmake is needed to compile and use Geant4
- the version available on the CentOS7 repository is too old, we have to compile it
- wget https://cmake.org/files/v3.11/cmake-3.11.3.tar.gz
- tar -xvzf cmake-3.11.3.tar.gz
- cd cmake-3.11.3
- ./bootstrap && make && make install
- Add to ~/.bashrc the line:
  export PATH=/home/soft/cmake-3.11.3/bin:$PATH

Prerequisites on Debian/Ubuntu

- sudo apt update
- sudo apt install build-essential git cmake qt4-default qt4-dev-tools libxerces-c-dev

Prerequisites on a Mac

- install XCode and type in a terminal:
  xcode-select —install

- install XQuartz (https://www.xquartz.org/)

- install brew:
  /usr/bin/ruby -e "$(curl -fsSL https://
  raw.githubusercontent.com/Homebrew/install/master/install)”

- brew doctor; brew update

- brew install cmake, qt, xerces-c

Preparing the installation

- wget http://cern.ch/geant4/support/source/geant4.10.04.p012tar.gz
- tar -xvzf geant4.10.04.p02.tar.gz
  this will create geant4.10.04.p02/
- mkdir geant4.10.04.p02-build
- mkdir geant4.10.04.p02-install
- mkdir geant4-data
- cd geant4.10.04.p02-build

Compile and install

- cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo -DGEANT4_INSTALL_DATA=ON -DGEANT4_INSTALL_DATADIR=../geant4-data -DGEANT4_BUILD_MULTITHREADED=OFF -DGEANT4_USE_GDML=ON -DGEANT4_USE_QT=ON -DGEANT4_BUILD_CXXSTD=c++11 -DCMAKE_INSTALL_PREFIX=../geant4.10.04.p02-install ../geant4.10.04.p02

- make -j `nproc`

- make install

- Add to your .bashrc (~/.bash_profile on a Mac):
  export G4DIR=~/geant4.10.04.p02-install
  . $G4DIR/bin/geant4.sh
  alias g4make='cmake -DGeant4_DIR=$G4DIR'

Just an introduction

- This is not a C++ course
- Just few information useful to understand the Geant4 examples
- For a complete course: http://www.roma1.infn.it/people/rahatlou/programmazione++/

Few things about C++

- A general-purpose programming language
- Has imperative, **object-oriented** and generic programming features
- Provides facilities for low-level memory manipulation
- In 1983, "C with Classes" was renamed to "C++"
  (++ being the increment operator in C)
- Initially standardised in 1998
  (current standard is C++17 but the most used is C++11)

Classes

- Classes are an expanded concept of data structures: like data structures, they can contain data members, but they can also contain functions as members

Like Plato's ideas (the idea of apple), classes have generic attributes (e.g. color).
Each instance (this Golden Delicious apple) of the class have a specific attribute (e.g. yellow)

```
class Apple {
public:
  void setColor(color);
  color getColor();

private:
  color fColor;
```

Example of class usage

```
#include <iostream>
using std::cout;

class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area() {return width*height;}
};

void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}

int main () {
  Rectangle rect;
  rect.set_values (3,4);
```

Idea of rectangle

An instance of rectangle

Example of class usage

```
#include <iostream>
using std::cout;

class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area() {return width*height;}
};

void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}

int main () {
  Rectangle rect;
  rect.set_values (3,4);
```

Declaration

Namespace

Implementation

Usage of the methods

Example of class usage

```
#include <iostream>
using std::cout;

class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area() {return width*height;}
};

void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}

int main () {
  Rectangle rect;
  rect.set_values (3,4);
```

Hyperuranion

(ὑπερουράνιος τόπος)

literally: "place beyond heaven”

“Real” world

What if I want to protect the rectangle properties (the dimensions), once instantiated?

Constructors

```
#include <iostream>
using std::cout;

class Rectangle {
    int width, height;
  public:
     Rectangle(int x, int y);
    int area() {return width*height;}
};

void Rectangle::Rectangle(int x, int y)
{
  width = x;
  height = y;
}

int main () {
  Rectangle rect(3,4);
```

Using the constructor and removing the setting method

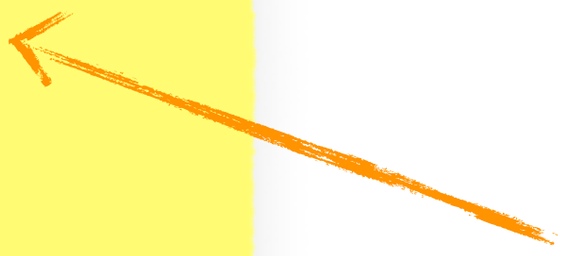
Constructors

```
#include <iostream>
using std::cout;

class Rectangle {
    int width, height;
  public:
     Rectangle(int x, int y);
    int area() {return width*height;}
};

Rectangle::Rectangle (int x, int y) : width(x),
height(y) { }


int main () {
  Rectangle rect(3,4);
  cout << "area: " << rect.area();
  return 0;
}
```



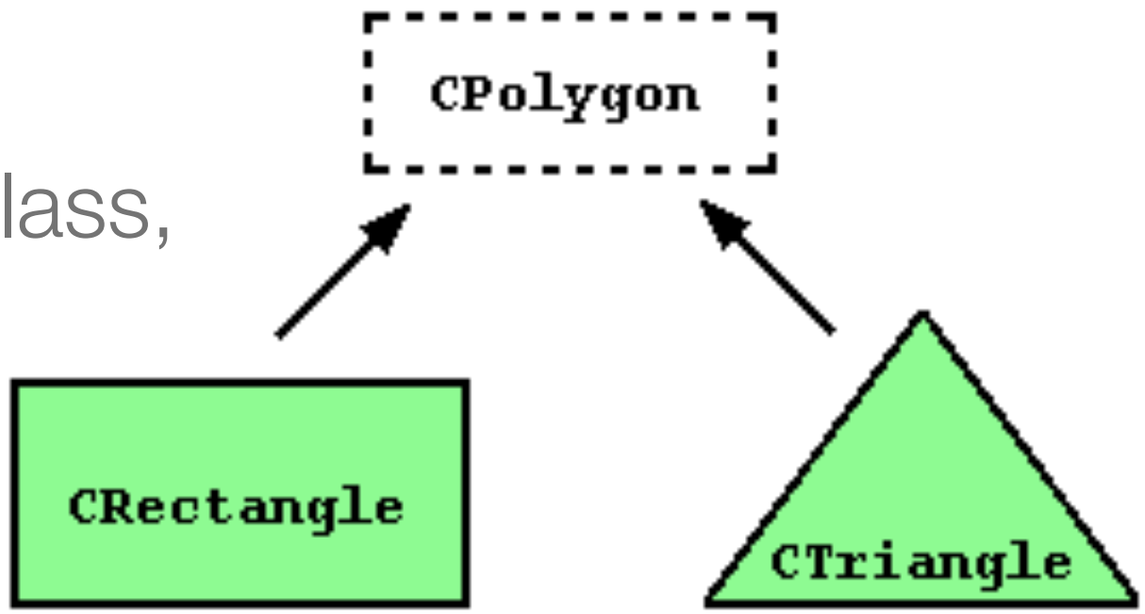
Better implementation!

Inheritance

- Classes in C++ can be extended, creating new classes which retain characteristics of the base class

- This process, known as inheritance, involves a base class and a derived class

- The derived class inherits the members of the base class, on top of which it can add its own members



Inheritance, an example

```
class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b;}
```

```
class Rectangle: public Polygon
{
 public:
  int area ()
  {
   return width*height;
  }
```

```
class Triangle: public Polygon
{
 public:
  int area()
  {
   return width*height/2;
  }
```

Protected and not private!

- The protected access specifier used in class Polygon is similar to private. Its only difference occurs in fact with inheritance:

- When a class inherits another one, the members of the derived class can access the protected members inherited from the base class, but not its private member

- By declaring width and height as protected instead of private, these members are also accessible from the derived classes Rectangle and Triangle, instead of just from members of Polygon

- If they were public, they could be accessed just from anywhere

Let’s use the classes…

```
#include <iostream>
using std::cout;
using std::endl;

int main () {
  Rectangle rect;
  Triangle trgl;
  rect.set_values (4,5);
  trgl.set_values (4,5);
  cout << rect.area() << endl;
  cout << trgl.area() << endl;
```

have a look at the example
https://github.com/carlomt/inheritance_example
for more details