Monte Carlo methods and Geant4

VI International Geant4 School
26-30 November 2018, Trento (Italy)

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

Istituto Nazionale di Fisica Nucleare

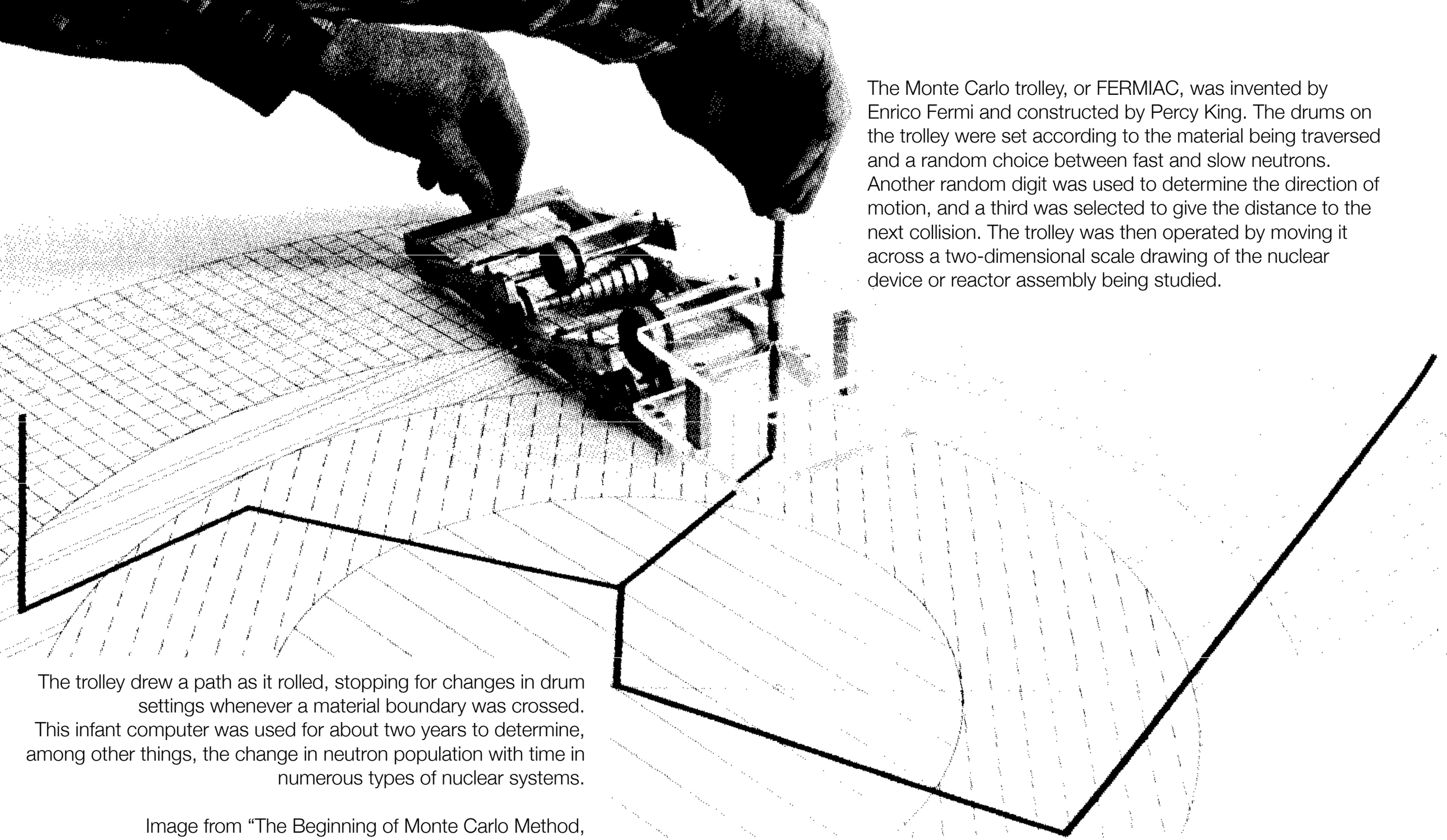
Overview

- The Monte Carlo method
- A short introduction to Geant4
  - The transport of particle
  - Geometry and scoring
  - Running an example

For these slides I took inspiration from:

- M. Asai (SLAC, Stanford)
- A. Dotti (SLAC, Stanford)
- S. Incerti (CNRS, Bordeaux)
- L. Pandola (INFN-LNS, Catania)

- You can download examples and slides from: http://www.roma1.infn.it/~mancinit/Teaching/Trento/



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”

Monte Carlo methods

- It is a mathematical approach using a sequence of random numbers to solve a problem

- Stochastic quantities (e.g.: average value of *mm* of rain)

- Deterministic problems (definite integral)

- Generate *N* random "points" $\vec{x_i}$ in the problem space

- Calculate $\langle f \rangle = \frac{1}{N} \sum_{i=1}^{N} f(\vec{x_i})$ and $\langle f^2 \rangle = \frac{1}{N} \sum_{i=1}^{N} f^2(\vec{x_i})$

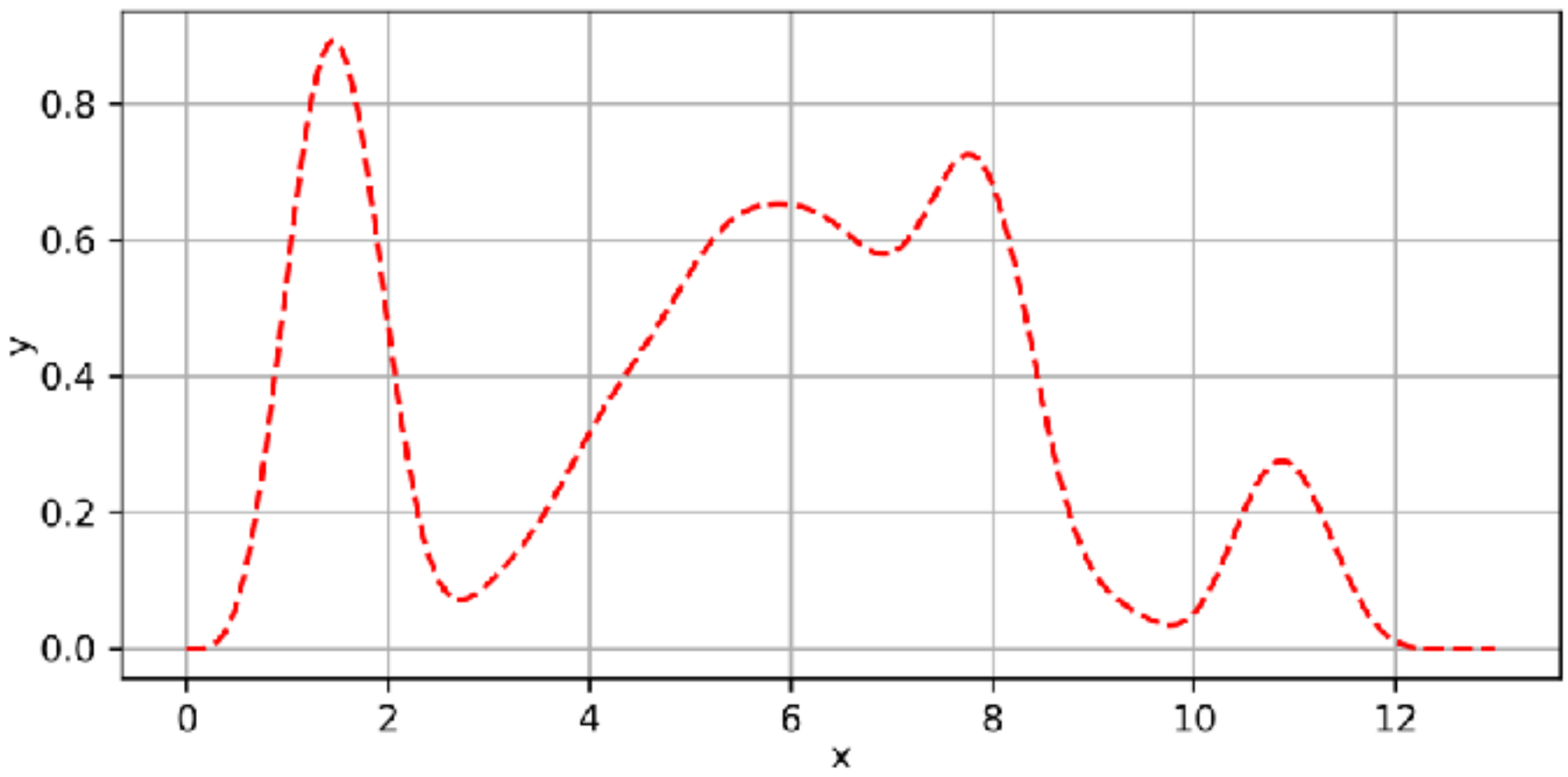
Monte Carlo methods

- Comte de Buffon (1777): needle tossing experiment to calculate the π;
- Laplace (1886): random points in a rectangle to calculate π;
- Fermi (1930): random approach to calculate the properties of the newly discovered neutron;
- Manhattan project (40's): simulations during the initial developments of thermonuclear weapons;
- Von Neumann and Ulam coined the term 'Monte Carlo' (1949);
- Exponential growth of the electronic computers (40's-60's);
- Berger (1963): first complete coupled electron-photon transportation code 'ETRAN'.

Example of an integral

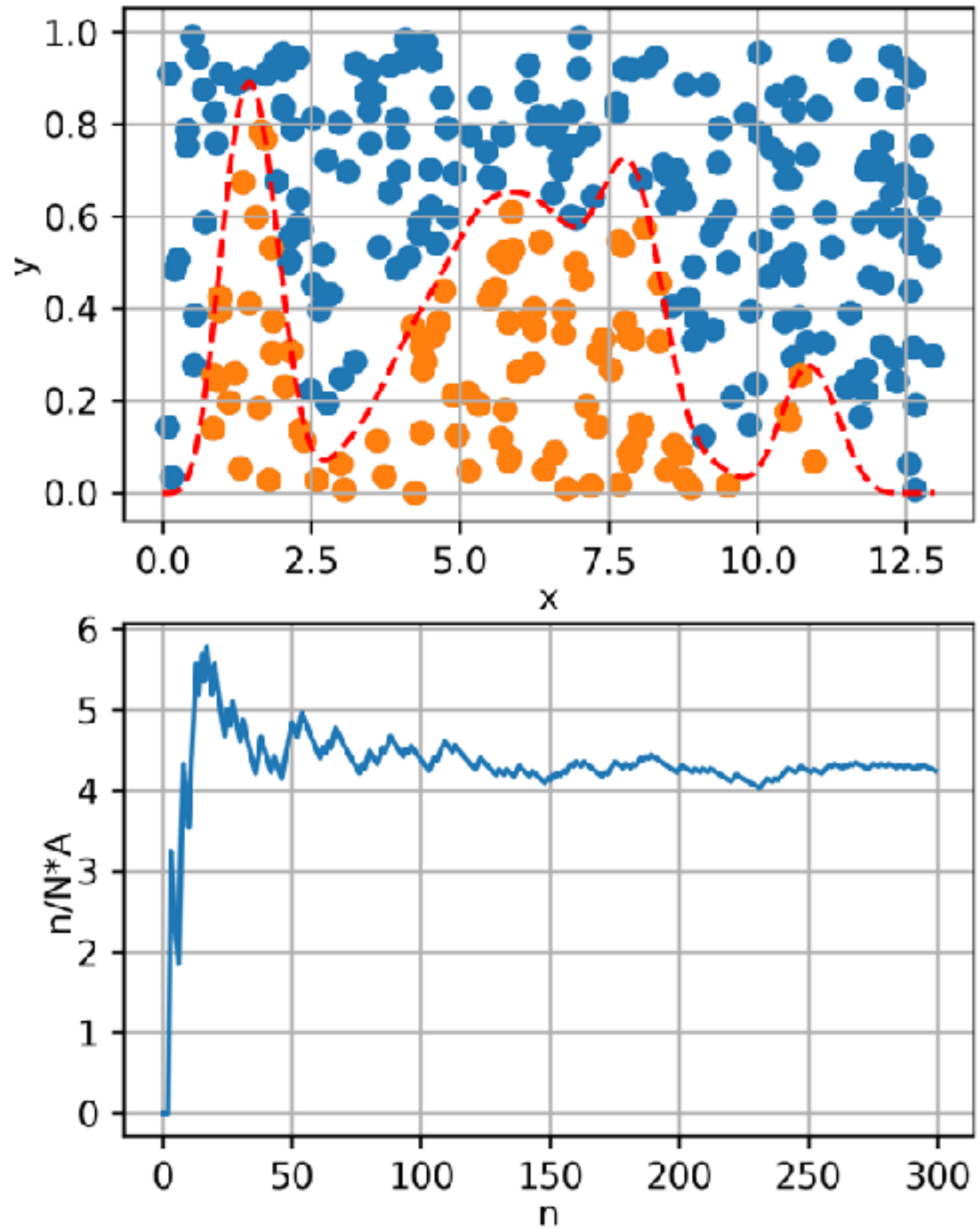
- What if you have to calculate the integral of a function as:

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



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 calculate π

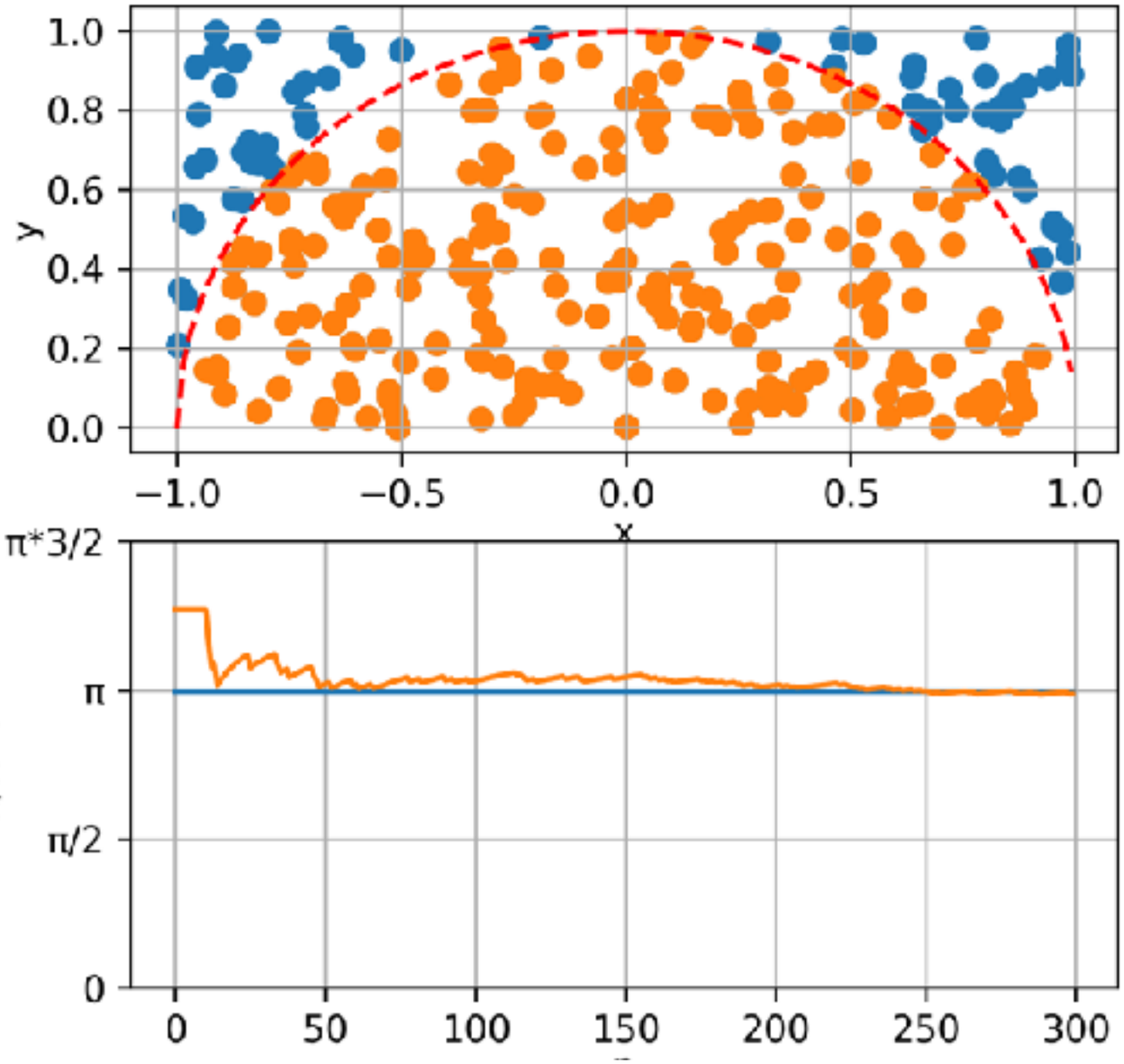
$$f(x) = \sqrt{(1 - x^2)}$$

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

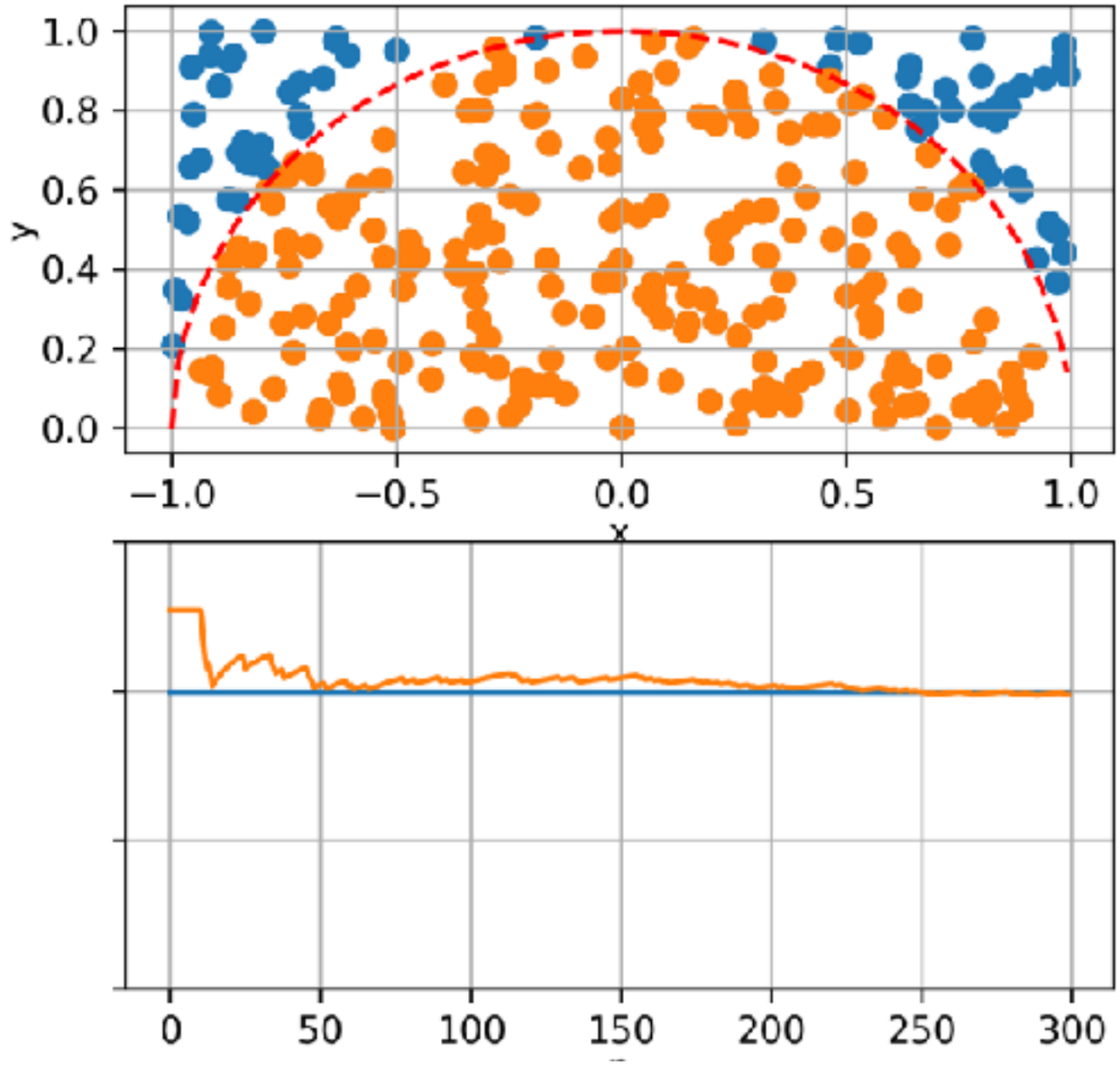
$$A_{rect} = \Delta y \cdot \Delta x$$

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

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

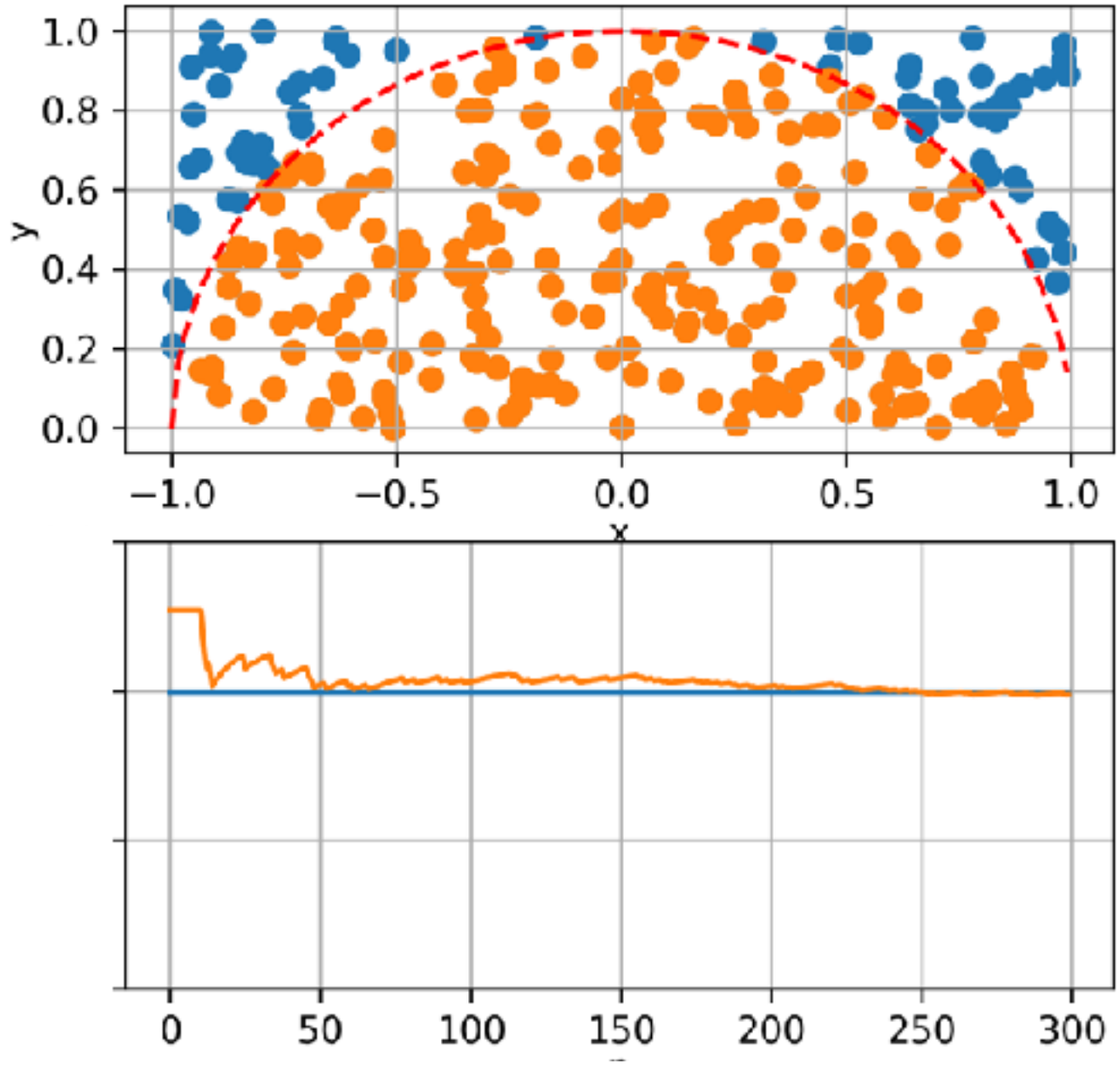


Let’s calculate π



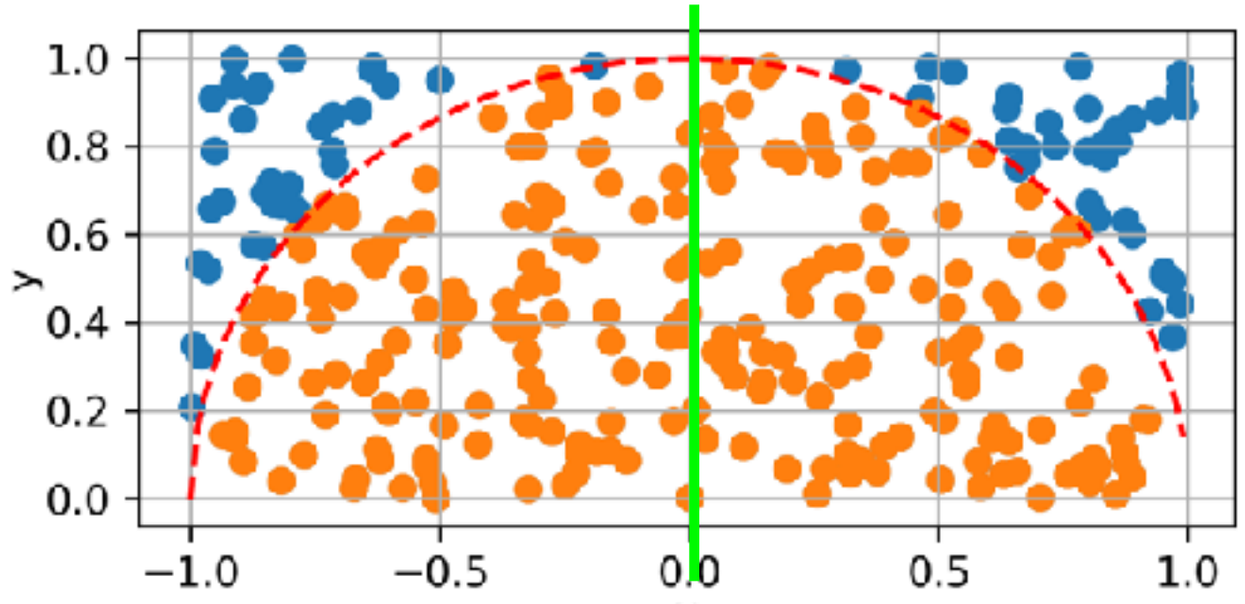
Let’s calculate π

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



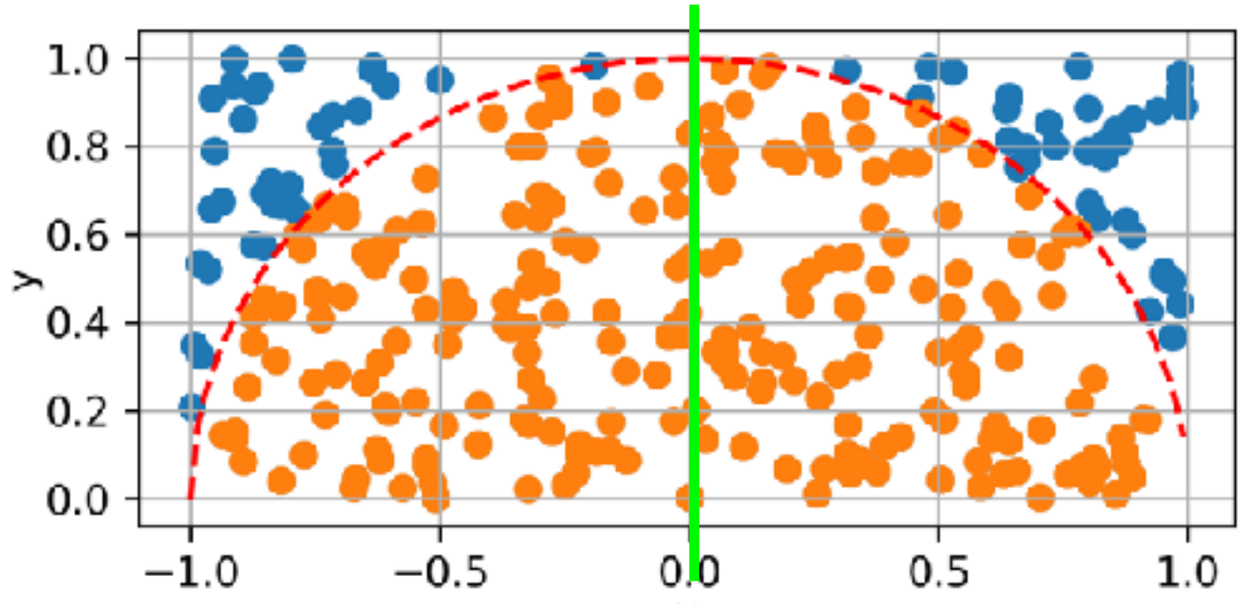
Let’s calculate π

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



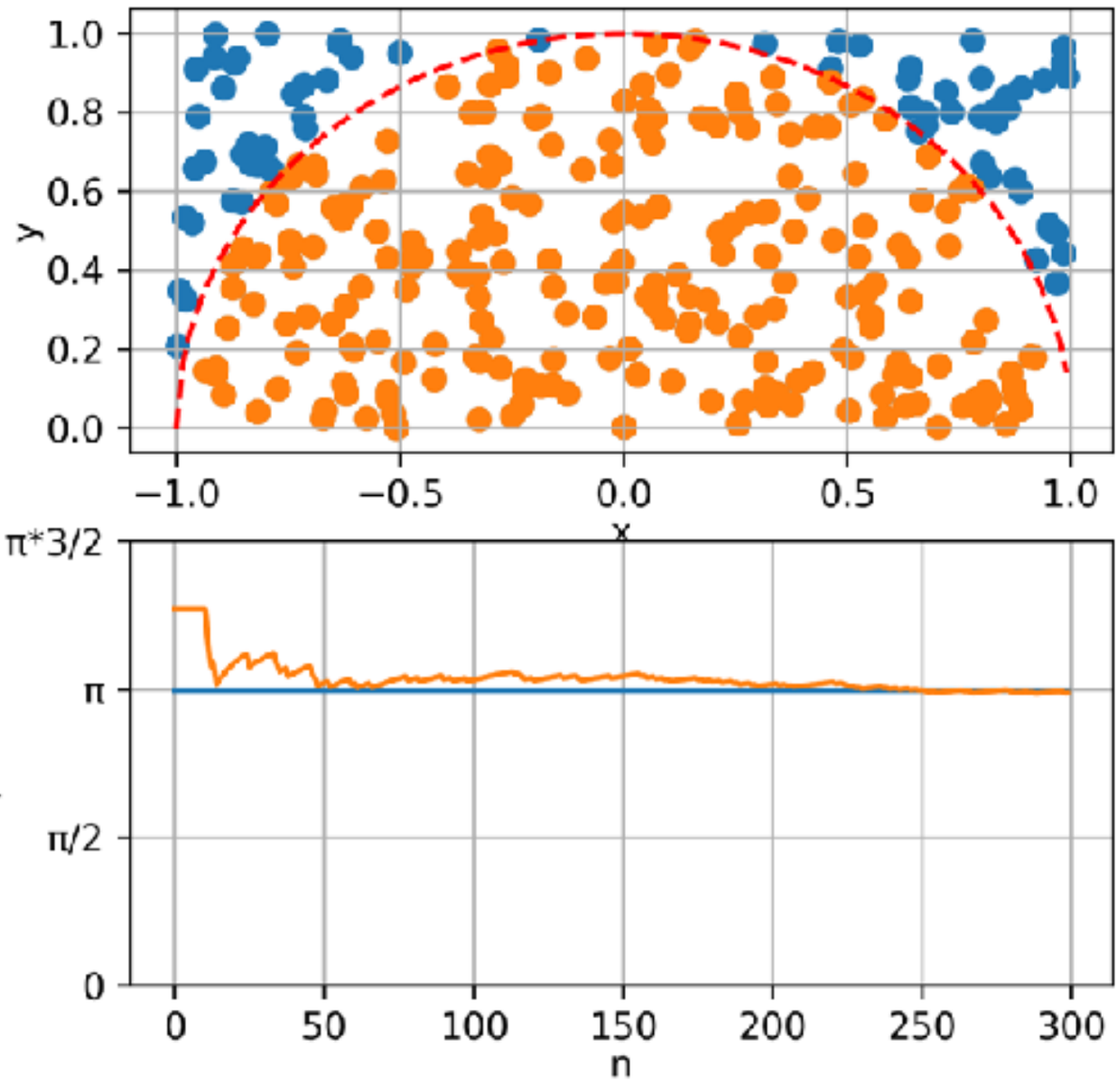
Let’s calculate π

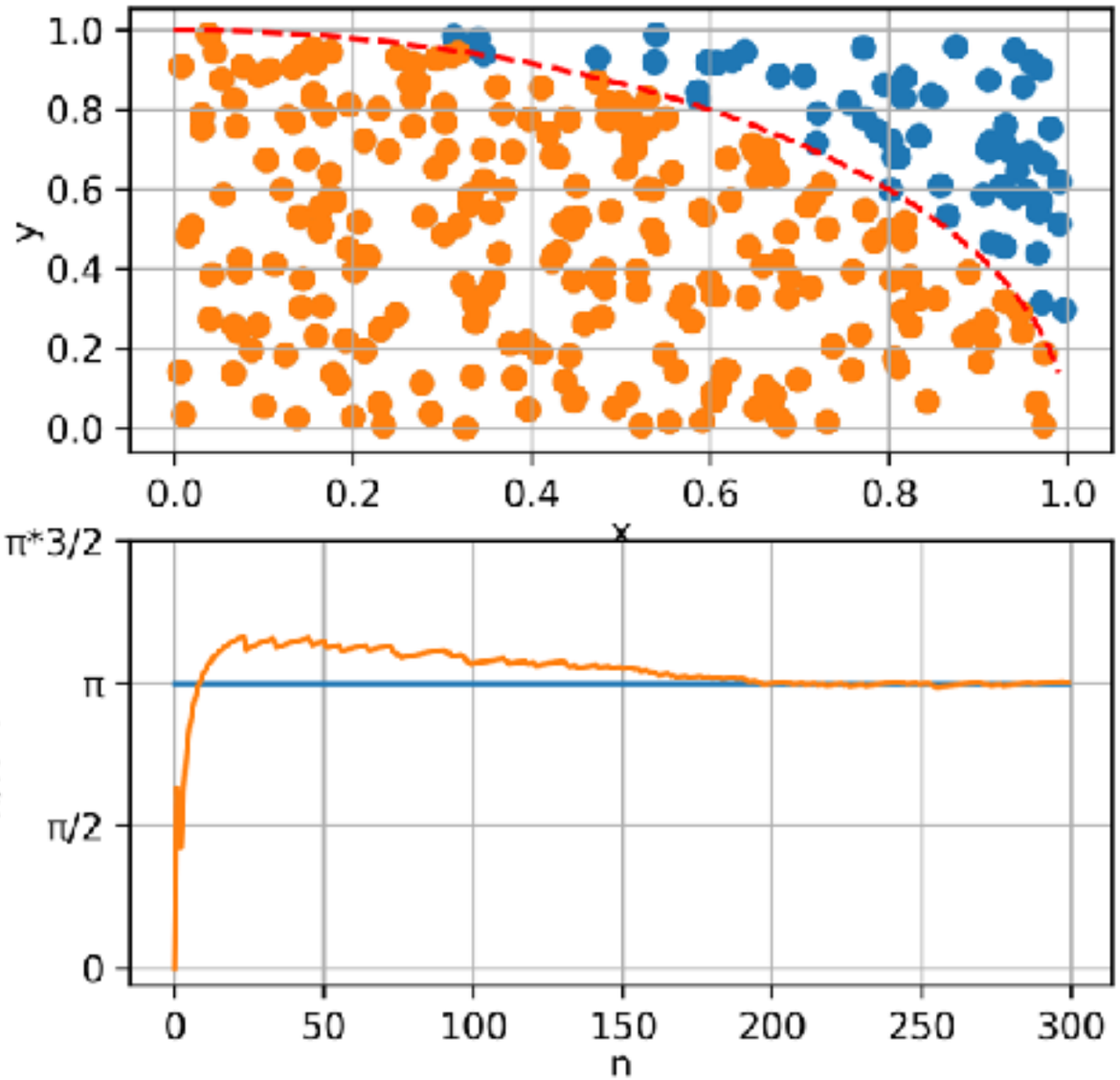
- Is there a way to speed up the convergence of the computation?
- Use the symmetry!



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

Use the simmetry!





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

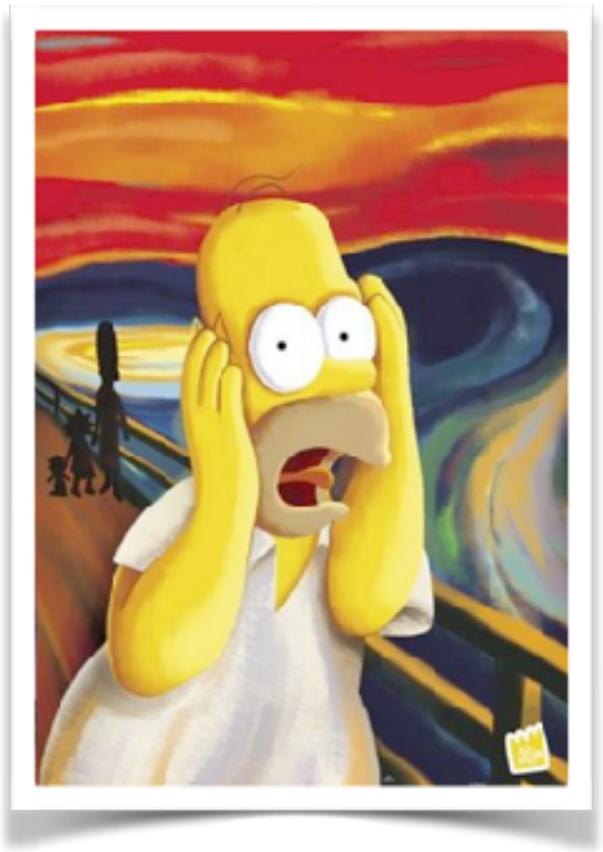
- At the core of all Monte Carlo calculations is some mechanism to produce a long sequence of random numbers r_i that are uniformly distributed over the open interval *[0,1)*

- 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

Pseudo-random Number Generators

- Because the set of numbers directly representable in the computer is finite, the sequence will necessarily repeat

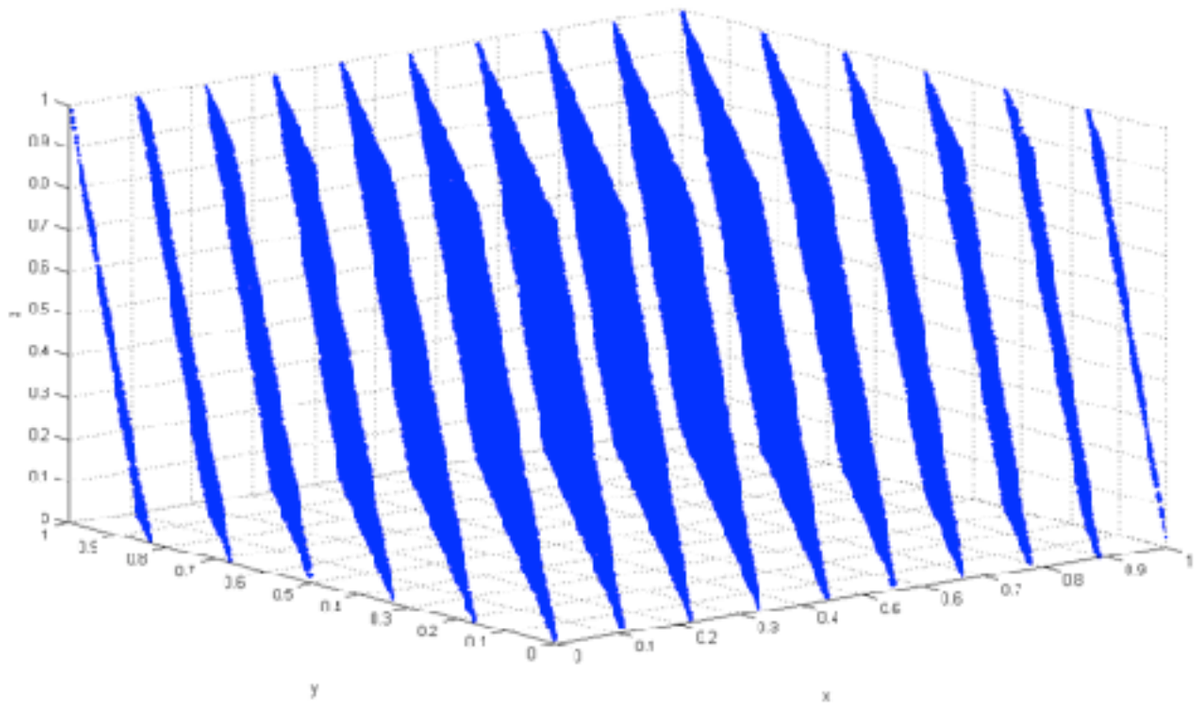
- The length of the sequence prior to beginning to repeat is called **period**

- Many pseudo-random number generators have been proposed and used over the years in a wide variety of Monte Carlo work.

- Designing better random number generators (and test them) is still an active area of research

Bad PRNG: RANDU

- Linear congruential PRNG used since ’60
- Was the most widely used random number generator in the world
- Developed by IBM
- Three-dimensional plot of 100,000 values generated with RANDU
- Each point represents 3 consecutive pseudorandom values



[Image from Wikipedia]

$$x_i = 65539 \cdot x_{x-1} \mod (2^{31})$$

not only the period is important!

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

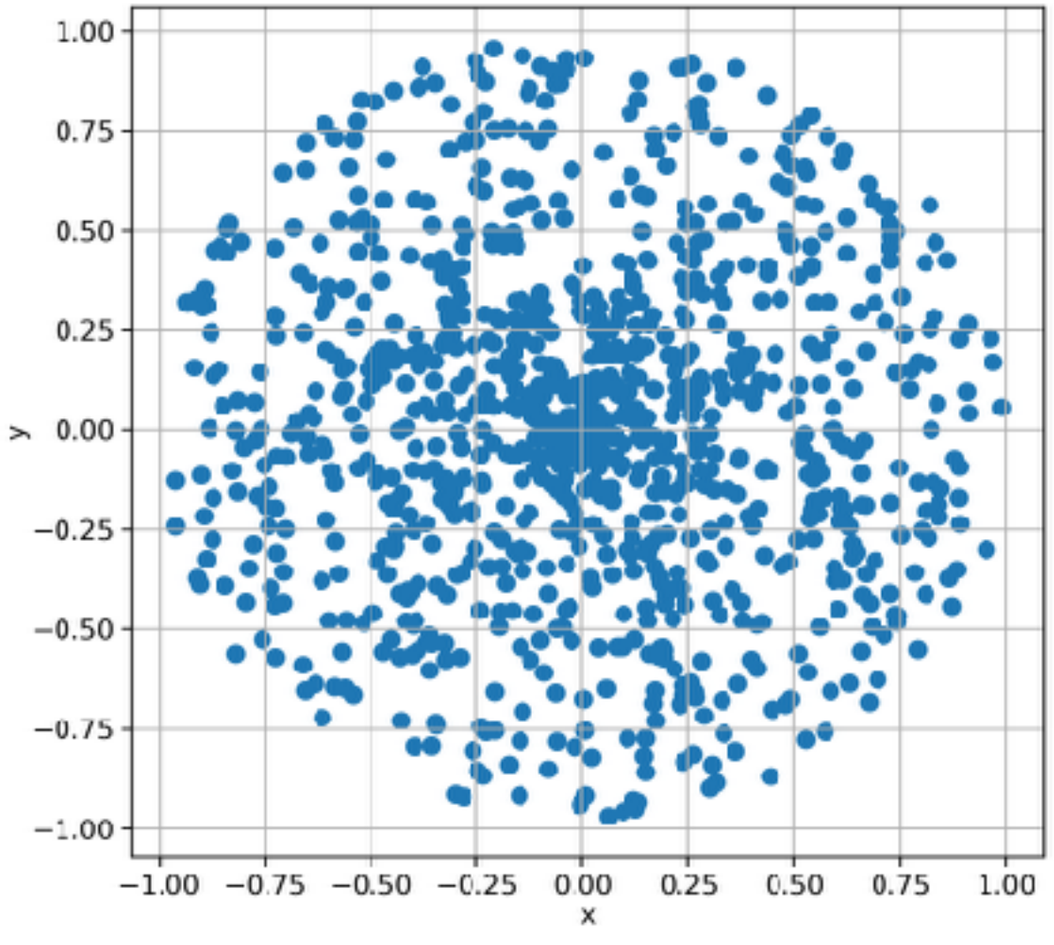
- Why did I sampled θ and ϕ in the CM frame?

Problem

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

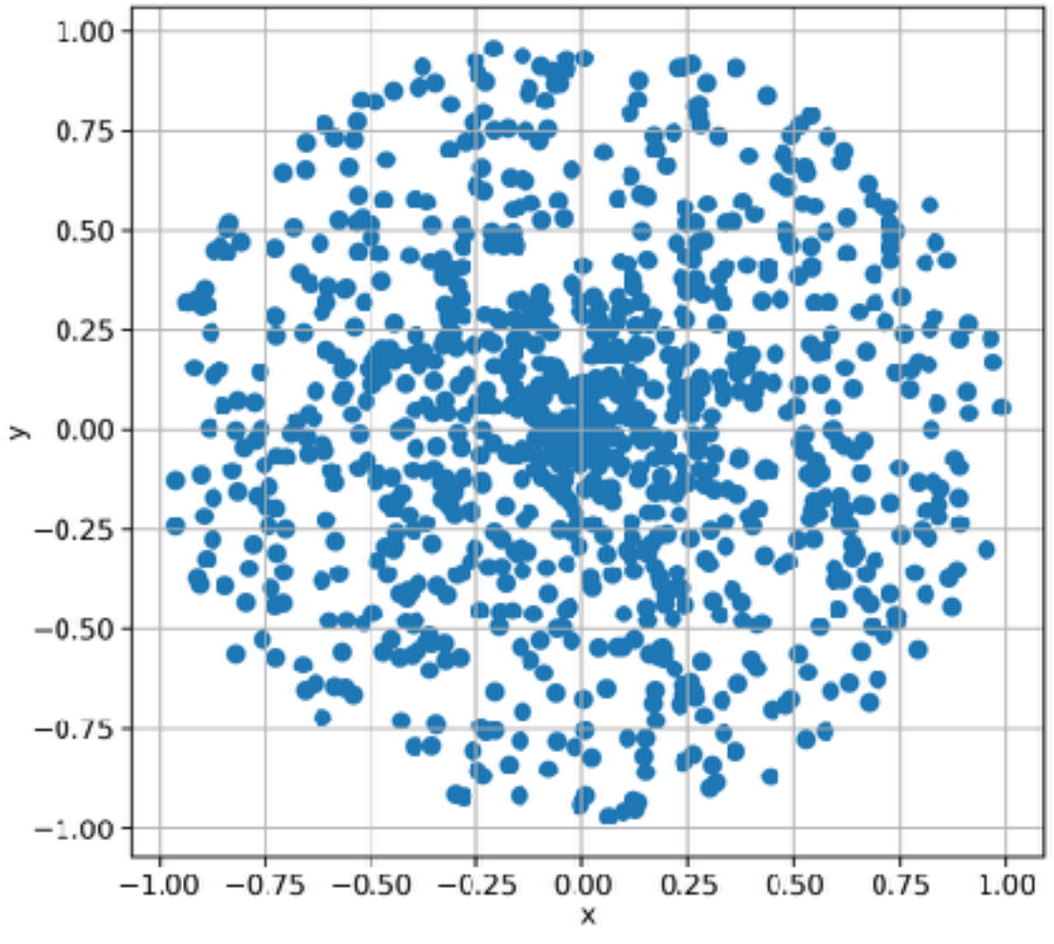
Problem

- 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



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)

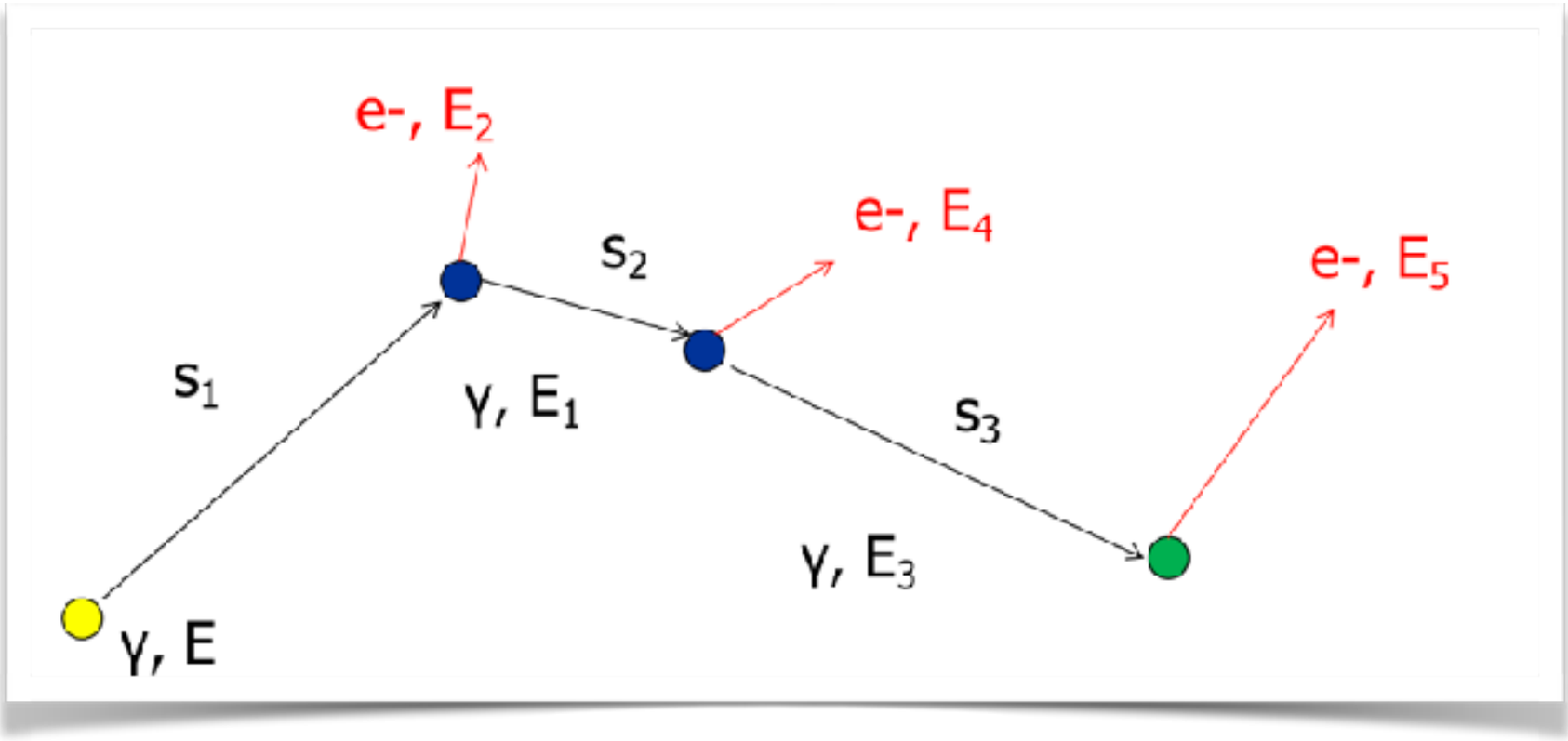
Particle tracking

- 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!)



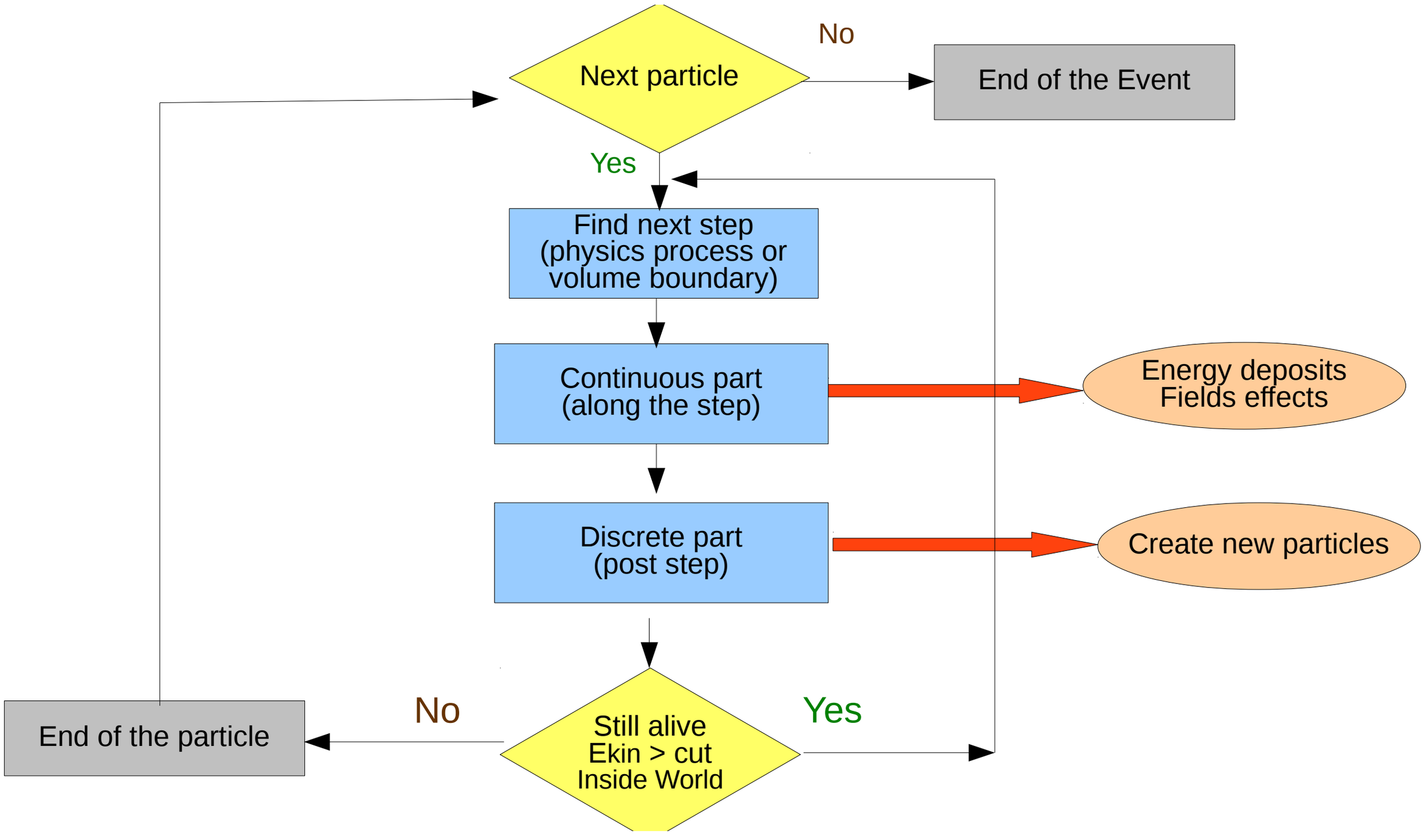
Tracking, not so easy…

- 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

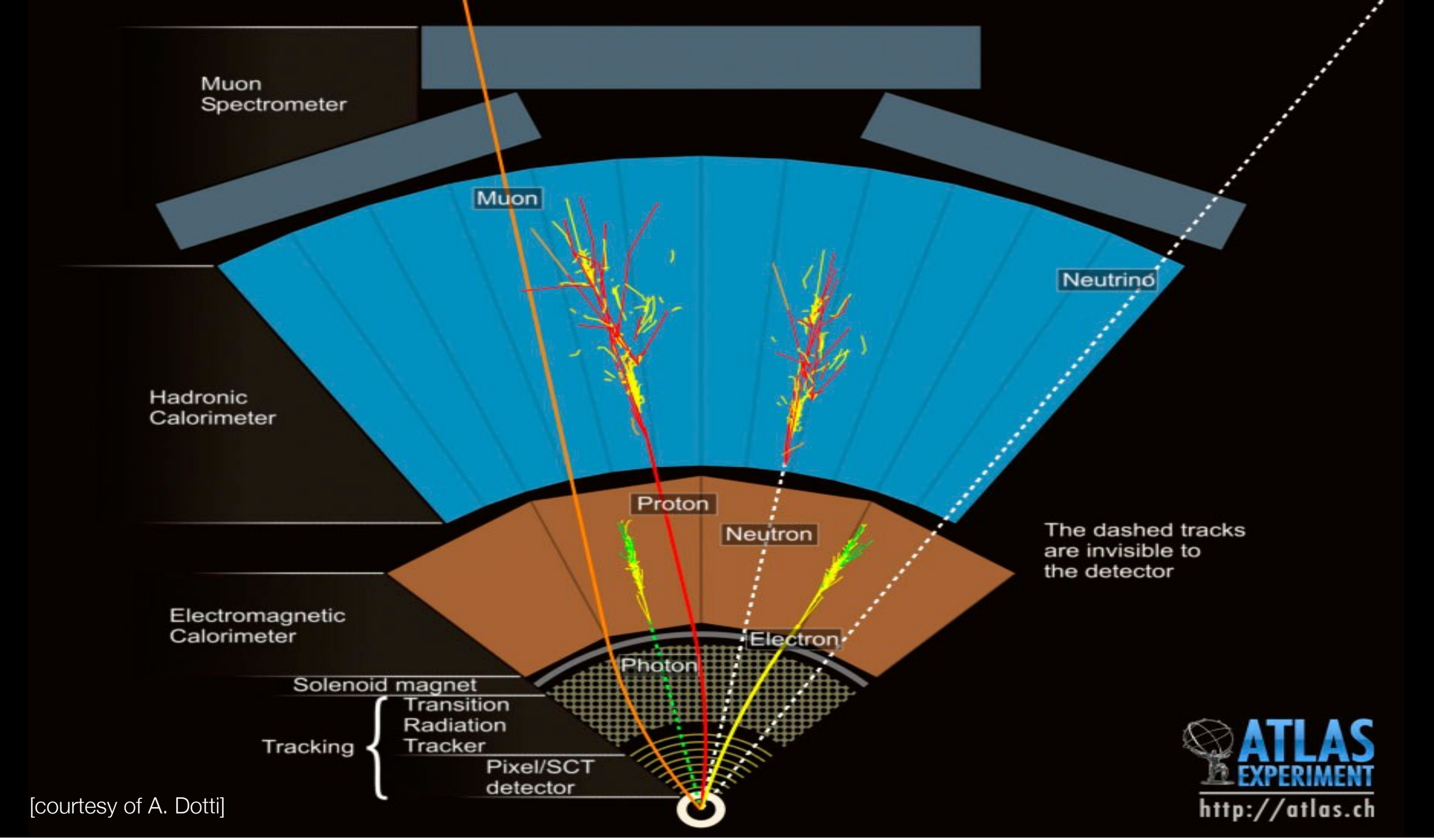
The solution: approximate

- 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)

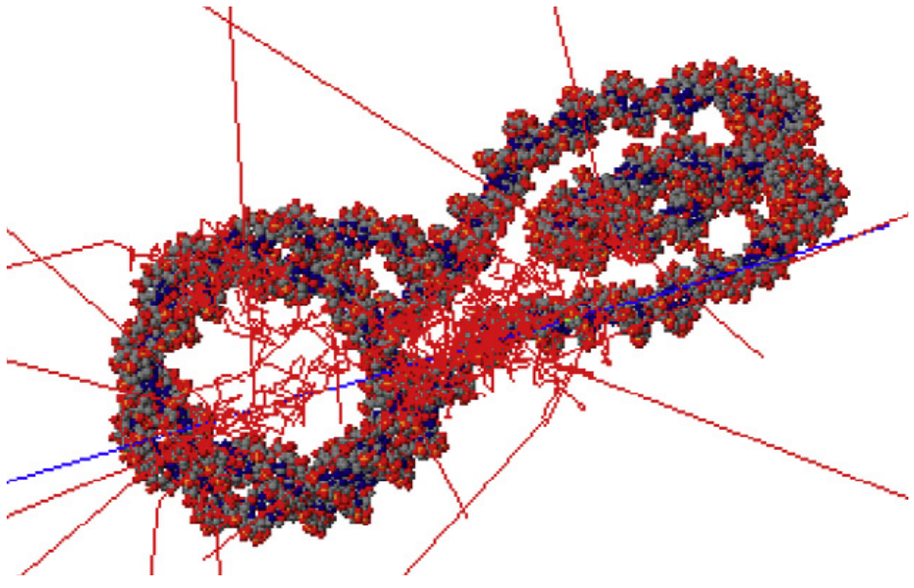
- Developed by an International Collaboration
  - Established in 1998
  - Approximately 100 members, from Europe, US and Japan
  - http://geant4.org
- Open source
- Written in C++ language
  - Takes advantage from the Object Oriented software technology

[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

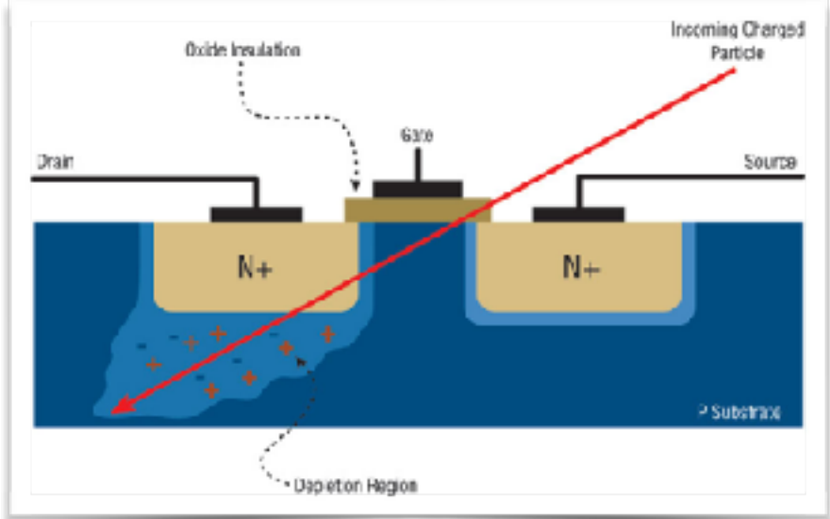


Figure from M. Sawant, COTS Journal Jan. 2012

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

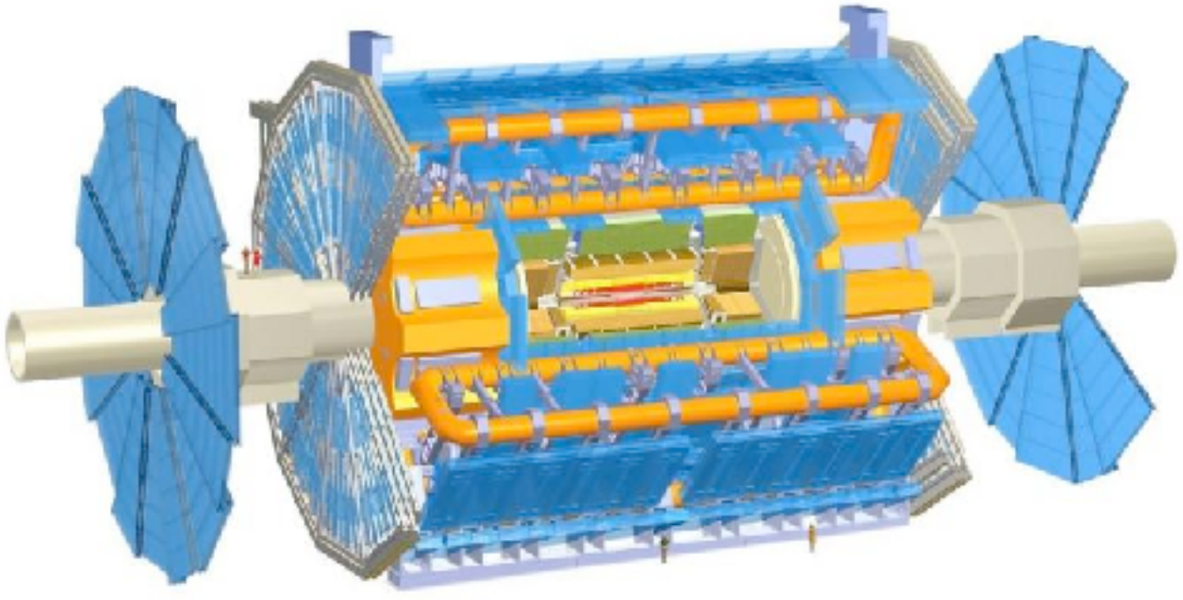
- Guidance: many examples are provided
  - **Basic Examples**: overview of Geant4 tools
  - Advanced Examples: Geant4 tools in real-life applications

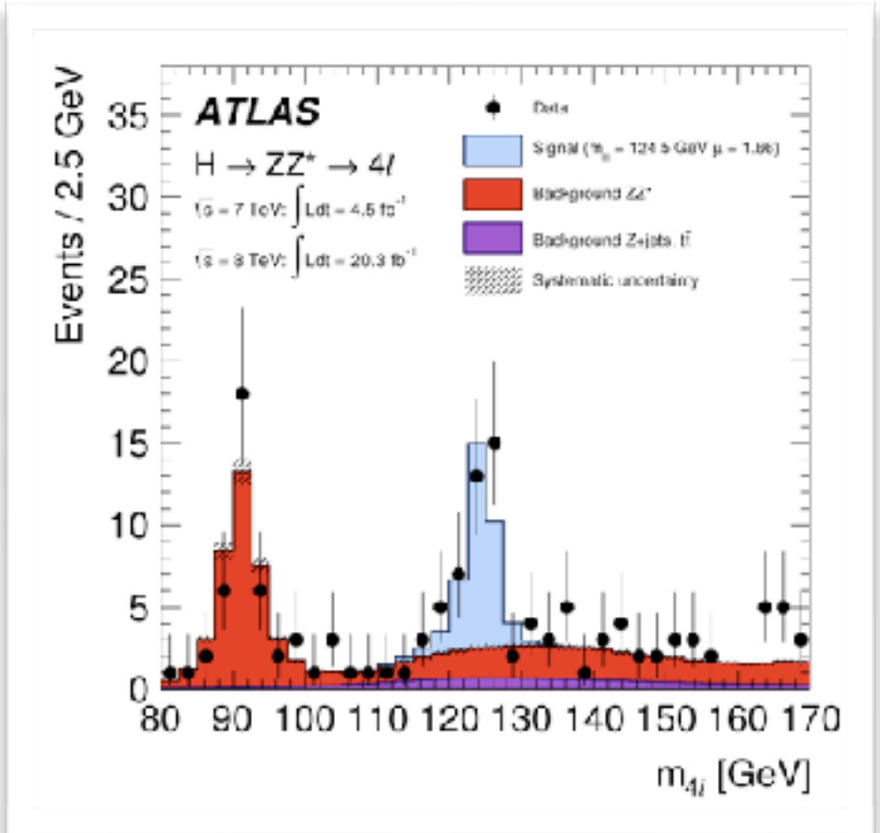
You MUST:

- 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:

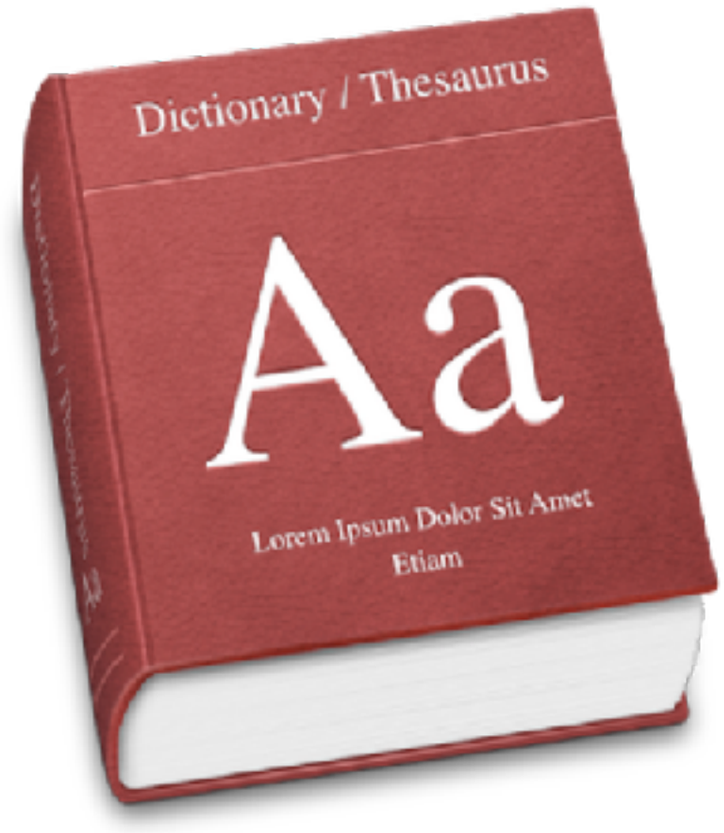
- Interact with Geant4 kernel to control your simulation
- Visualise your simulation configuration or results
- Produce histograms, tuples etc. to be further analysed





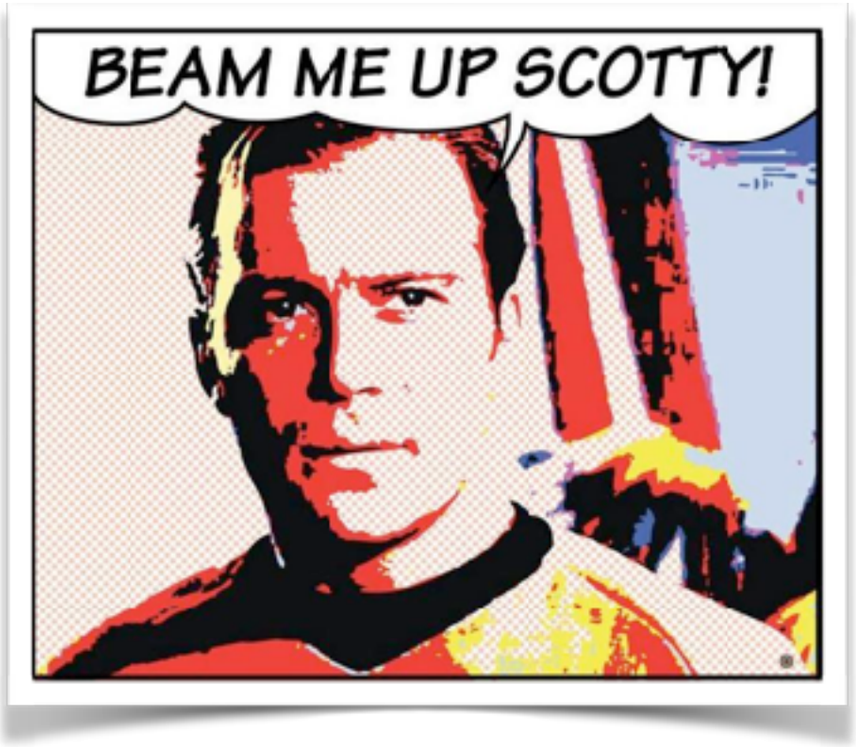
Jargons

- 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”**

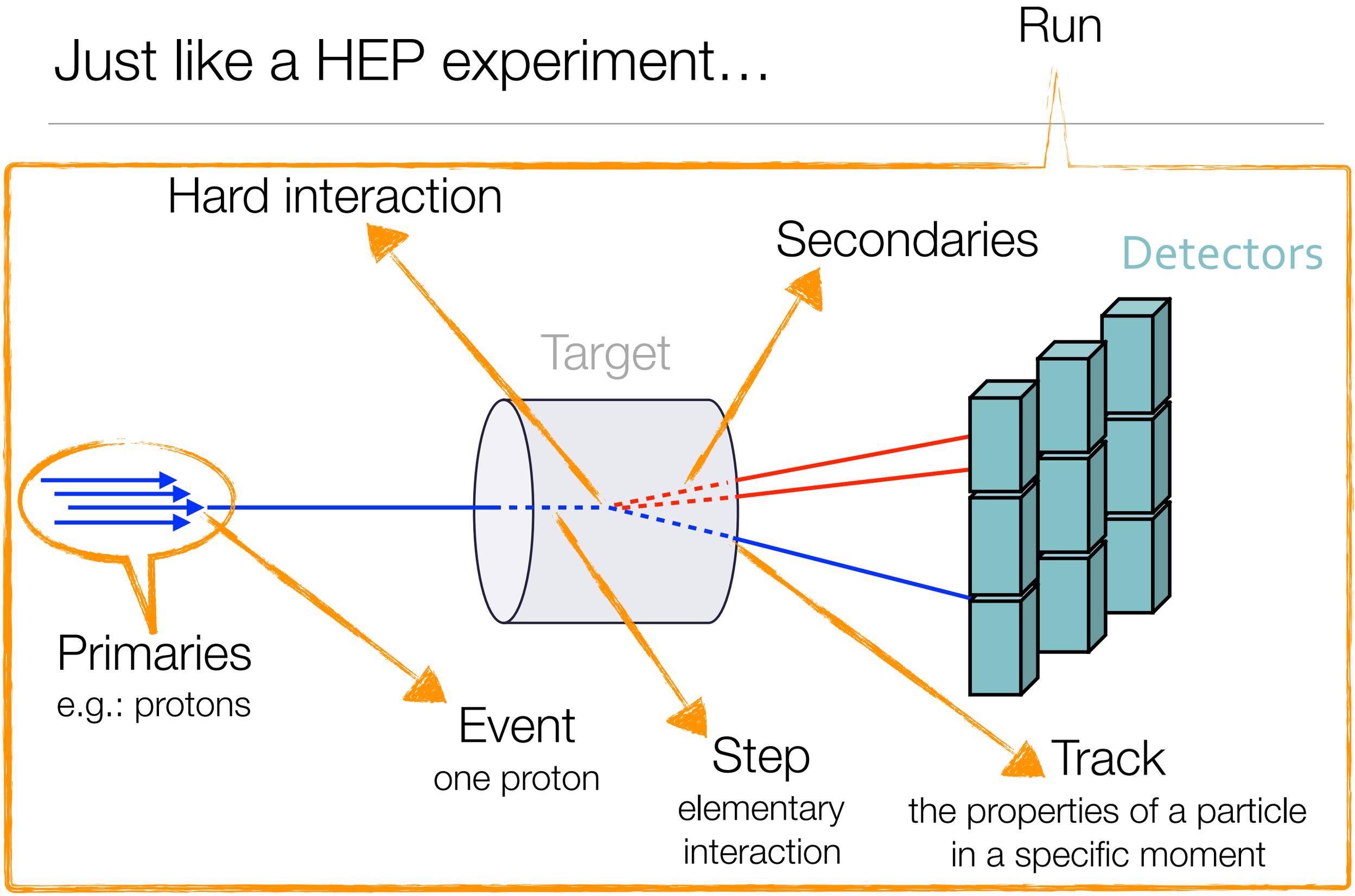


- A run is a collection of events which share the same detector and physics conditions

- **G4RunManager** class manages processing a run

- **G4UserRunAction** is the optional user hook

Just like a HEP experiment…



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**

User classes

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

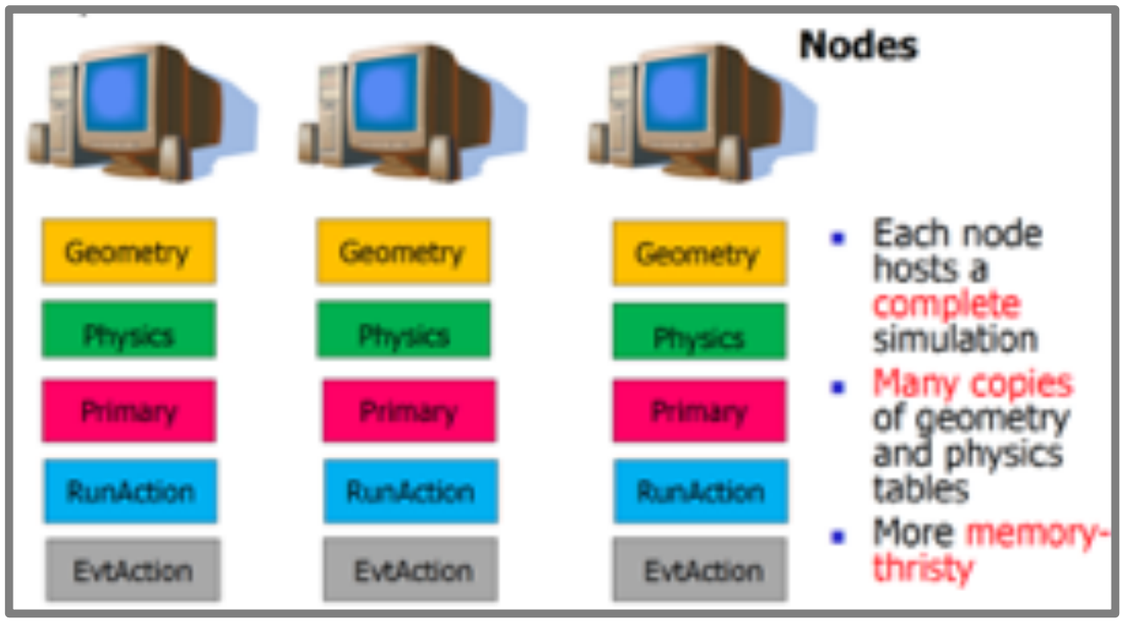
classes written in red are mandatory!

Your program

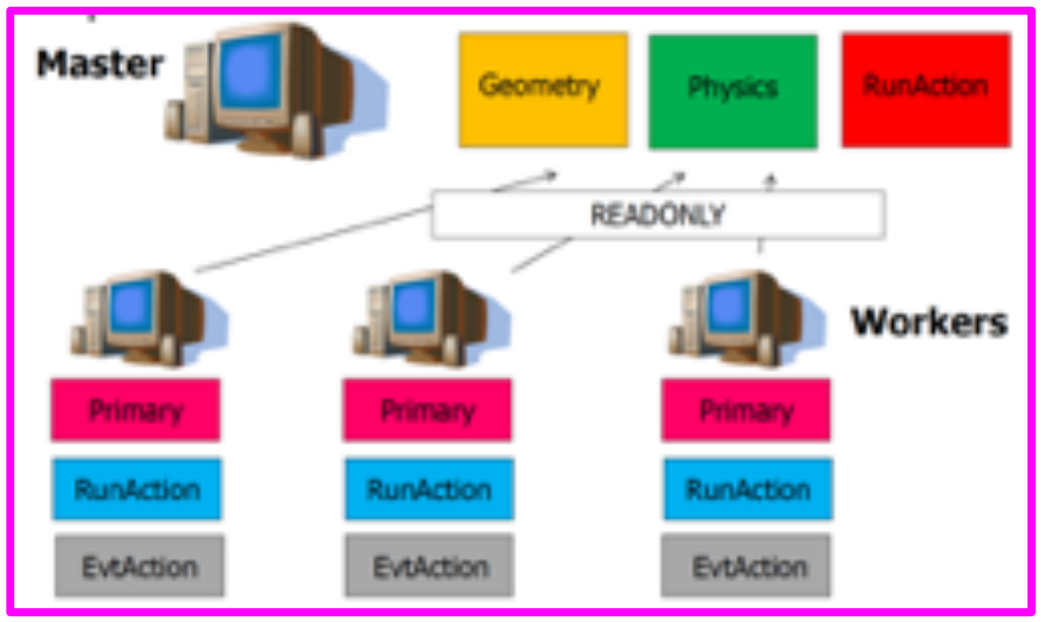
- Geant4 does not provide a main()
- In your main(), you have to
- Construct `G4RunManager` (sequential mode)
- Set user mandatory initialisation classes to RunManager
  - **`G4VUserDetectorConstruction`**
  - **`G4VUserPhysicsList`**
  - **`G4VUserActionInitialization`**
- You can define VisManager,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

Multi-threading in Giant

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)

let’s run an example…

thank you for your attention!