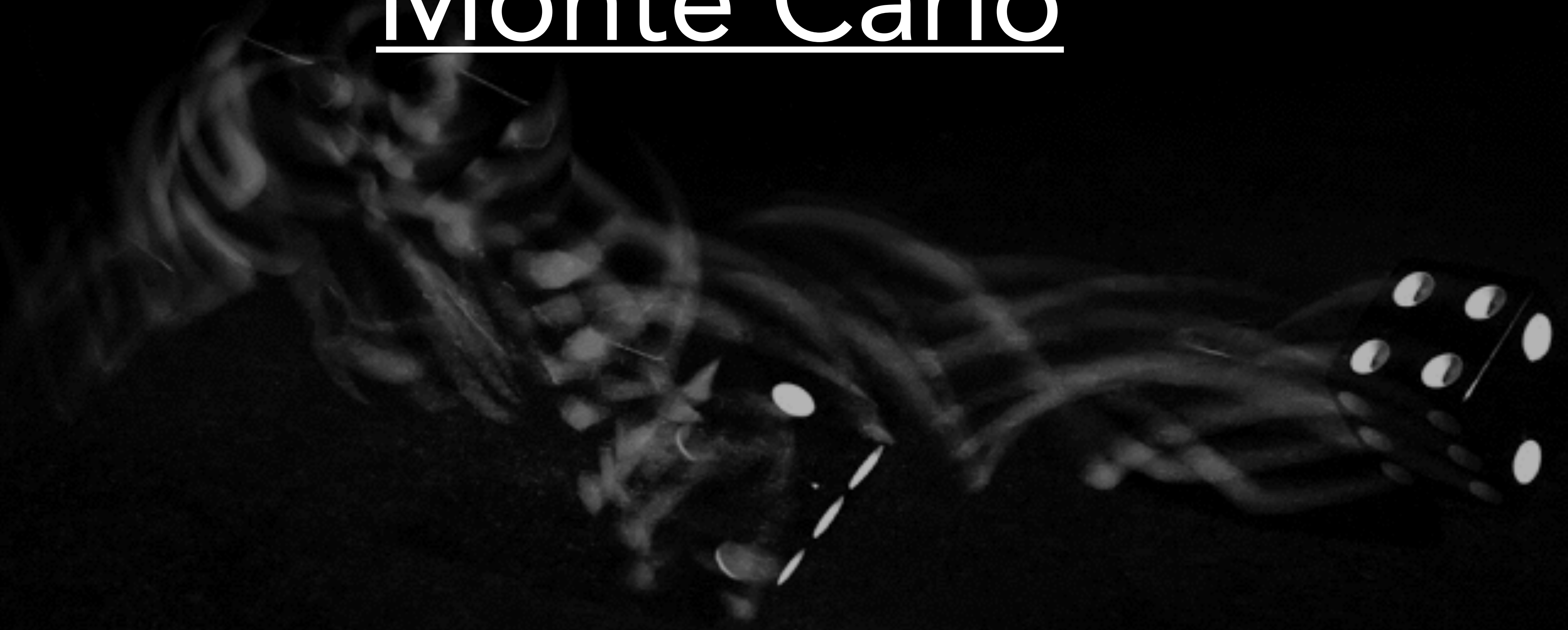


# Lecture 2: Monte Carlo

D. Jason Koskinen
koskinen@nbi.ku.dk

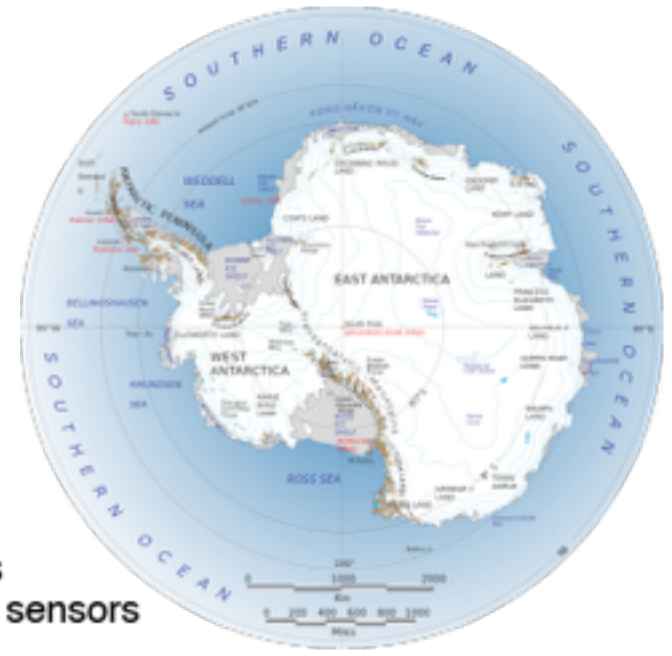
*Advanced Methods in Applied Statistics*
*Feb - Apr 2016*

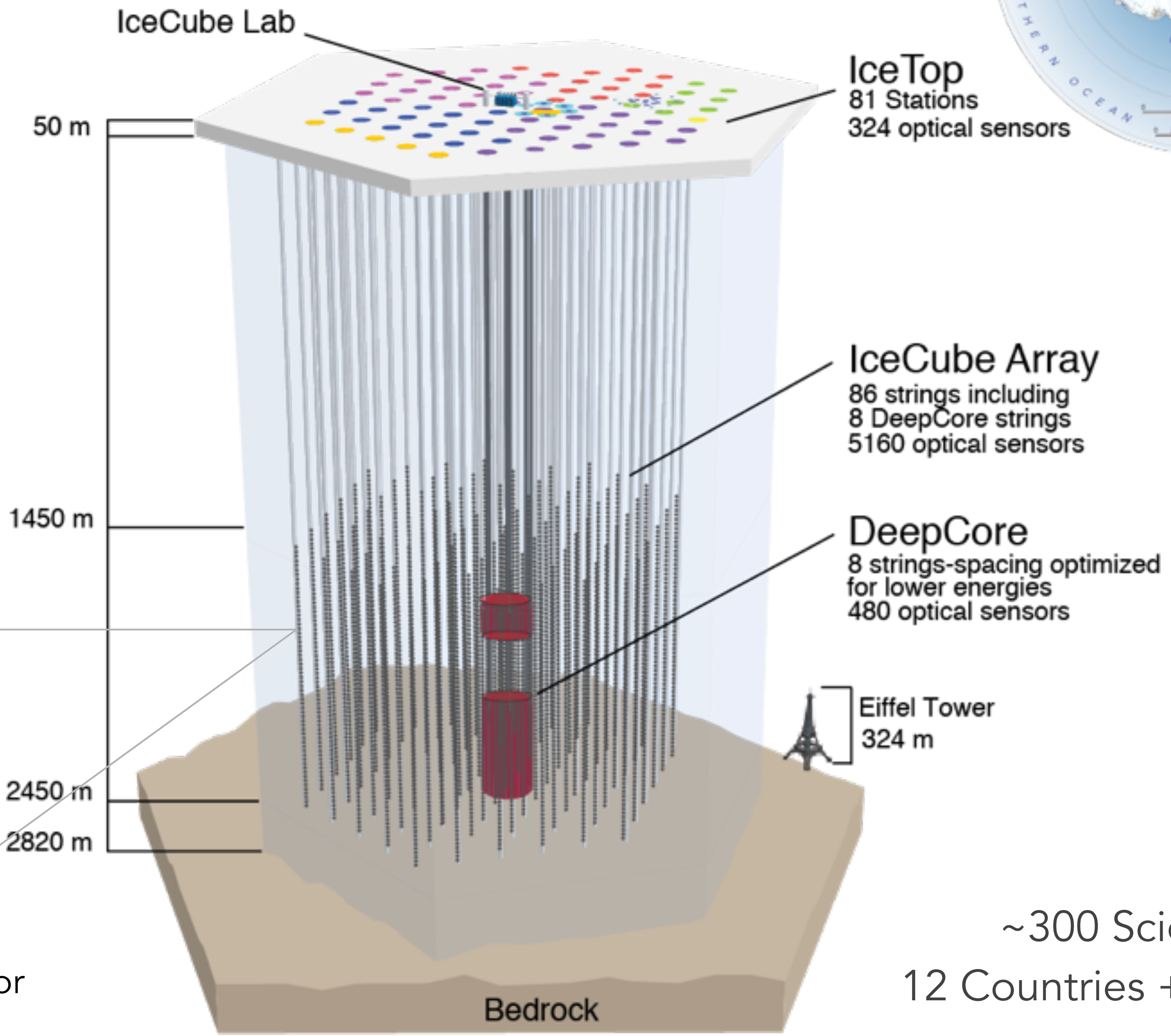
Photo by Howard Jackman

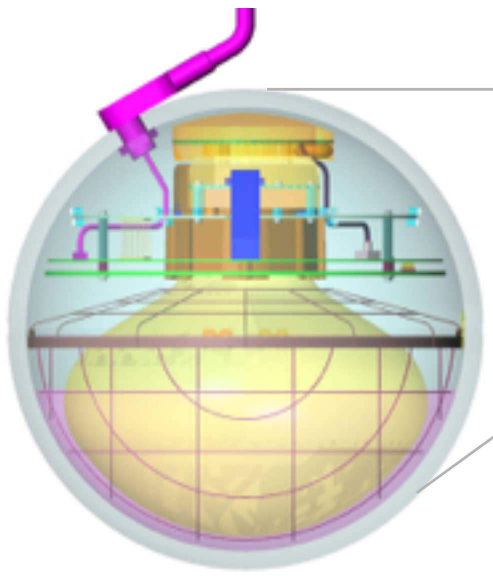
# Monte Carlo (Simulation)

- Some things are complicated, too complicated for simple analytic tools. Both for physics processes as well as instrumental responses ( noise, efficiency, thresholds, jitter, etc.)

# IceCube





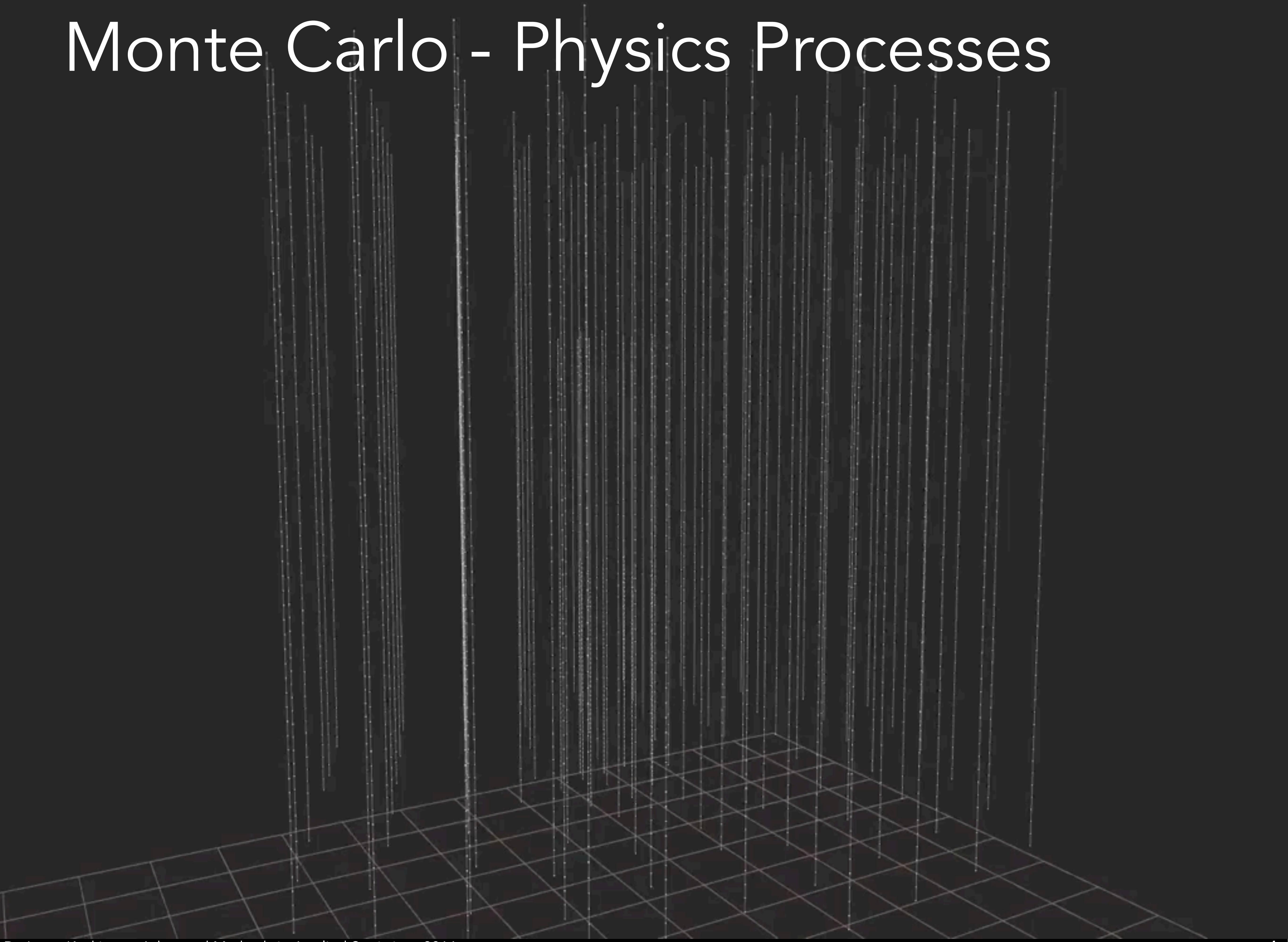


IceCube Optical Sensor

~300 Scientists

12 Countries + Antarctica

## Monte Carlo - Physics Processes



# Monte Carlo - Physics Processes

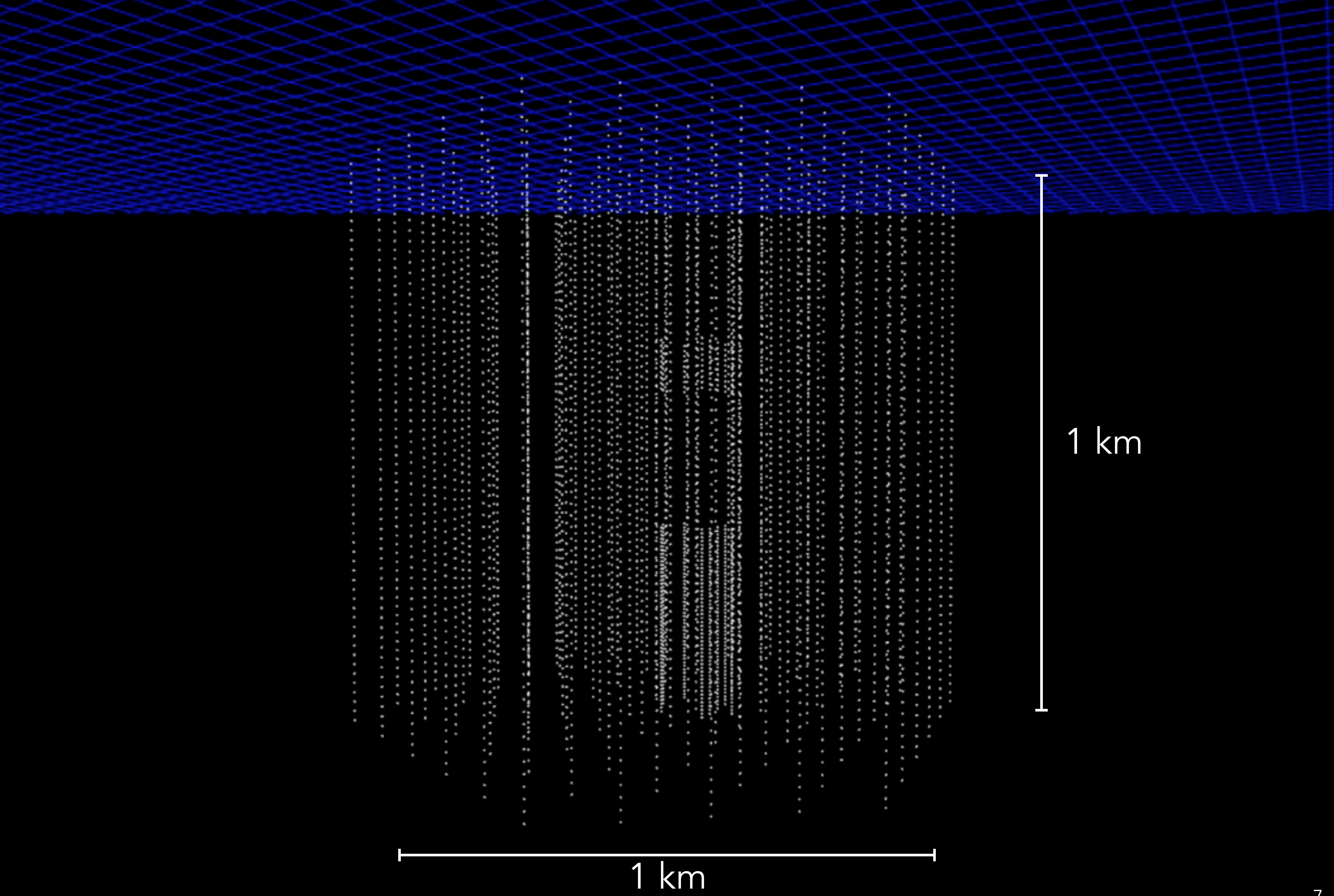
- The previous movie is the simulation of a high energy muon moving through the IceCube detector at the South Pole.
- As the muon moves through ice photons are emitted. Lots and lots of photons that are individually simulated as the thin strands (color denotes time since being emitted) as they scatter and then are ultimately absorbed.
- While the behavior that describes photon scatter and absorption may be simple, or complicated, it can be broken down at each photon 'step'
    - Does the photon get absorbed yes/no?
    - If no absorption, then does it scatter yes/no?
    - If it does scatter, how much between 0-360°?
    - Move one step and repeat
- This is impossible without computers

If we can give a probability of each of these outcomes, then we can break the complex process into manageable pieces

## Monte Carlo - Instrument/Detector

- Even with a purely analytic treatment of the physics, detectors/telescopes etc. are often complicated and extremely sensitive.

# Monte Carlo - Instrument/Detector



## Monte Carlo - Instrument/Detector

- The previous movie is 10 ms of simulated data including noise and cosmic ray muons (green lines)
- The background rate (cosmic ray muons) is ~2200 Hz, whereas the neutrino signal for some analyses are 1 event every 1-3 months
- Lots of Monte Carlo data makes sure you can optimize your analysis to keep signal and remove background, and has many, many more benefits
- It's all possible because of (pseudo) random number generators and computers

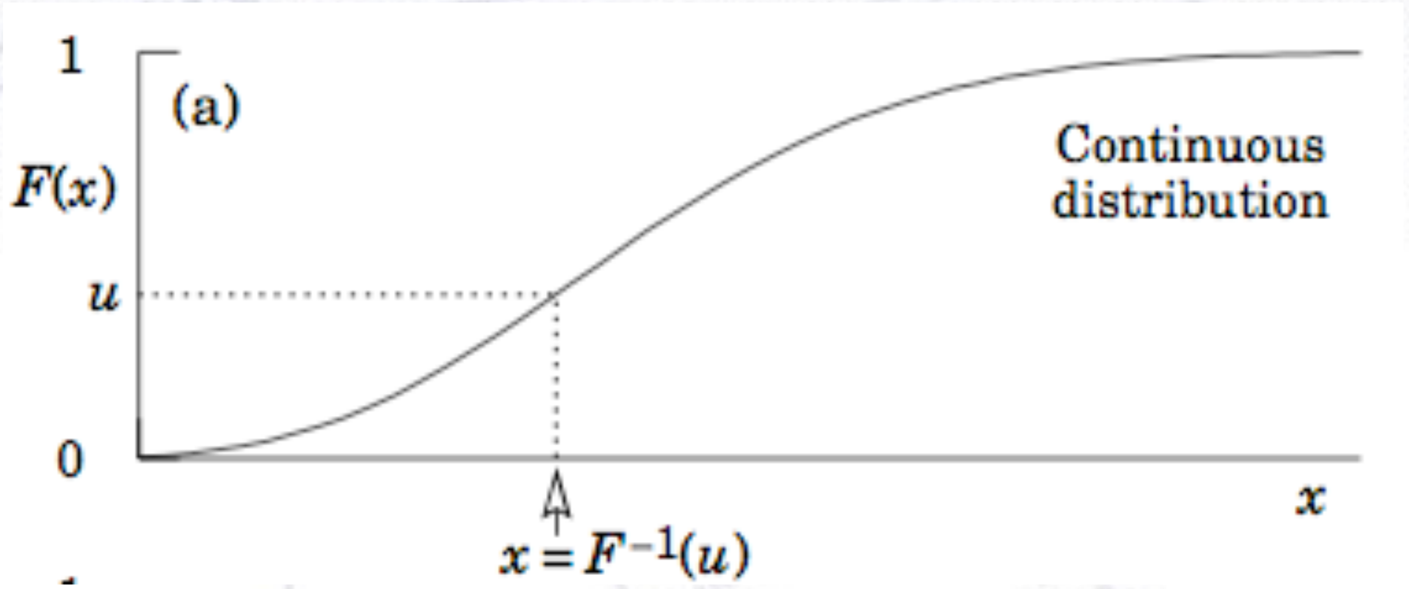
## Random Numbers

- A pillar of Monte Carlos and many analytic tools is the use of a generator that can produce ‘random’ values, most often 0-1
- Random in a linear fashion from 0-1 is nice because:
  - Probabilities go from 0-1
  - It is usually easy to map/transform linear in 0-1 to other ranges, e.g. 2-27, and functions. Note that zero can be problematic for some functions: 1/x, natural log, etc.
  - Many default random number generators produce values that are linear in 0-1, e.g. numpy.random.uniform()
- There are two main ways to apply random numbers for Monte Carlo simulation and probability distribution function sampling: Transformation method and Accept-Reject.

# Transformation method

We have uniformly distributed random numbers r. We want random numbers x according to some distribution.

We want to try to find a function x(r), such that g(r) (uniformly distributed numbers) will be transformed into the desired distribution f(x).



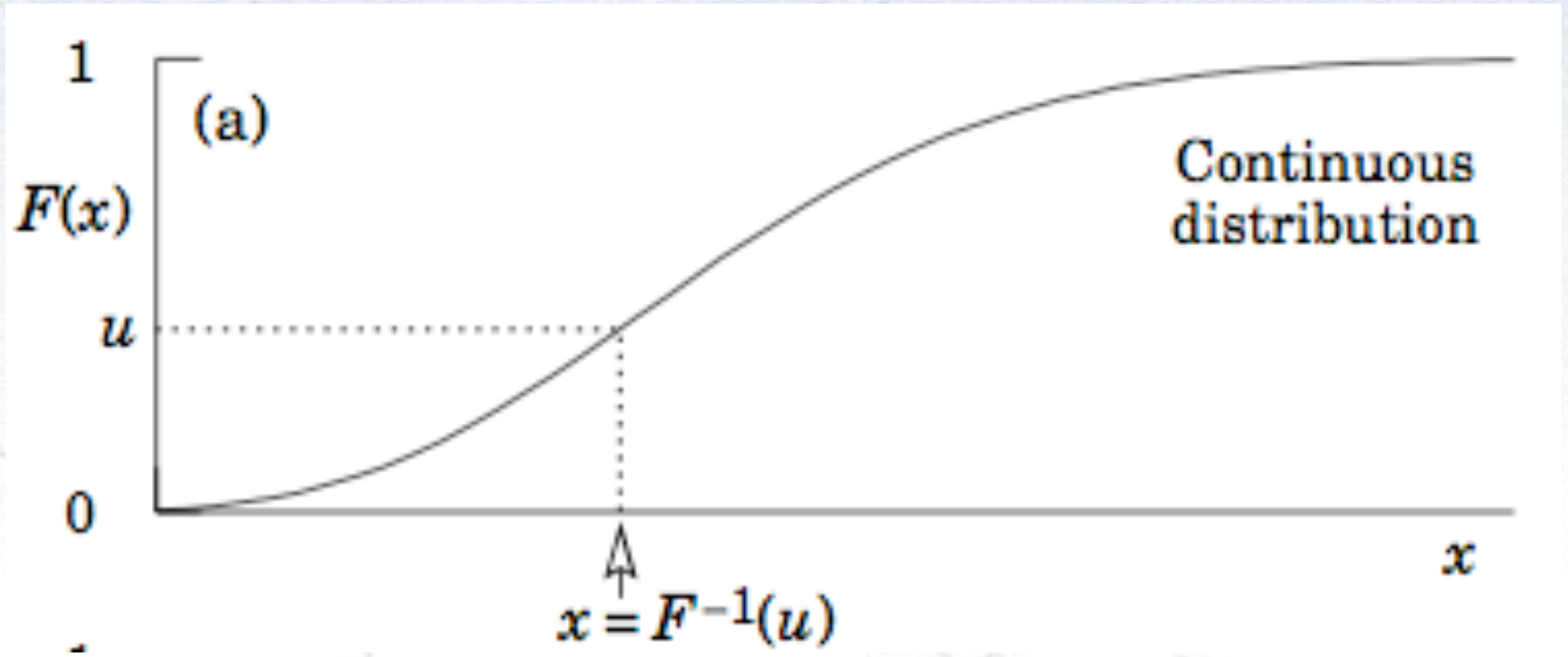
It turns out, that this is only possible, if one can (in this order):
- Integrate f(x)
- Invert F(x)

As this is rare, this method can not often be used by itself. However, in combination with the Hit-and-Miss method, it can pretty much solve all problems.

*T. Petersen, Applied Statistics

# Transformation method

So the "recipe" can be summarised as follows:



So the "recipe" can be summarised as follows:

- Ensure that the PDF is normalised!
- Integrate f(x) to get F(x) with the definite integral: $$F(x) = \int_{-\infty}^{x} f(x')dx'$$
- Invert F(x)

**Now you can generate random numbers, x, according to f(x), by choosing x = $F^{-1}$(u), where u is a random uniform number.**

## In Practice

- Transformation method is efficient when possible, but rarely practical and will NOT be in any problem set or exam for this course.
- Instead we focus on the Accept-Reject method, which is straightforward and many of the inherent inefficiencies are rarely noticed because of modern computer power

# Acceptance-Rejection Method

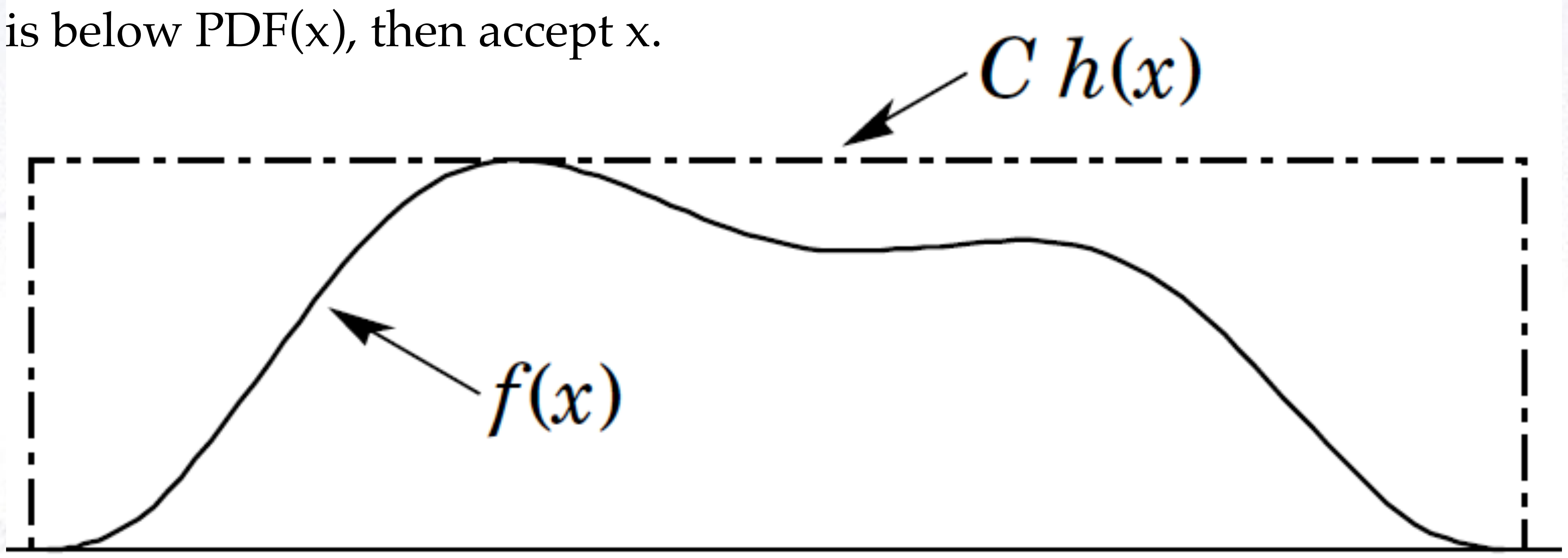
- For a probability distribution function that can nicely fit in a easily to integrate width/area/volume/etc. it is possible to generate a PDF-based random number generator

# Accept-Reject method

**(Von Neumann method)**

If the PDF we wish to sample from is bounded both in x and y, then we can use the "Accept-Reject" method to select random numbers from it, as follows:

- Pick x and y uniformly without in the range of the PDF.
- If y is below PDF(x), then accept x.



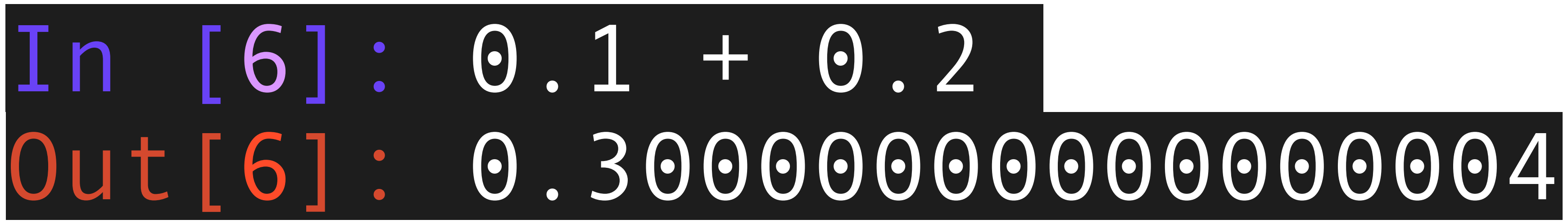
The main advantage of this method is its **simplicity**, and given modern computers, one does not care much about efficiency. However, it requires **boundaries**!

## Random Number Generators

- Get your favorite random number generator (and I know that you all have favorites) and start randomly sampling
- Make plots
  - See if there's a sequence to the generator
  - Is there a 'seed' option? If so does that change anything?
- Time the random number generator
  - Does it take 10 times longer run if it produces 10 times more random numbers?

# Precision

- Random numbers from generators are as accurate as the method, e.g. Mersenne twister, as well as the computational precision of the variable(s)
- Variable types related to int, float, and double have a characteristic precision, 8-bit, 32-bit, etc.
- For example, the float precision in some python versions is 53-bits, and therefore there is intrinsic rounding for precision at the scale of $1/2^{53}$



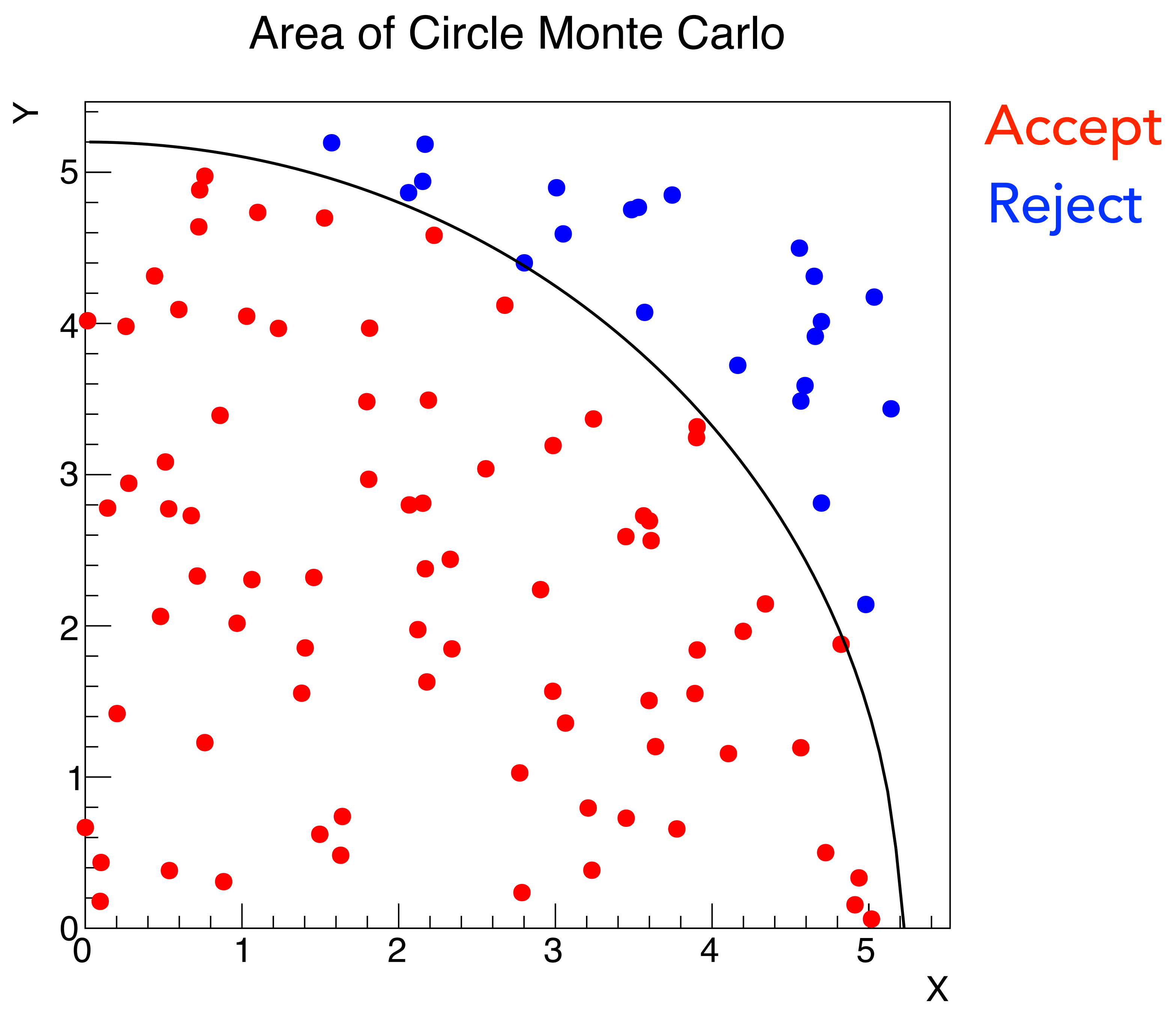
```
In [6]: 0.1 + 0.2
Out[6]: 0.30000000000000004
```

*ipython 4.0.1

## Classic & Simple Monte Carlo Usage

- Okay, so now you have a random number generator, let’s put it to use
- You have no clue about the value of π, but you want to calculate the area of a circle with only two things:
  - $x^2+y^2=r^2$
  - random number generator
- The core concept of a random number generator is that they are mostly random, but repeatable
- Show visualization, i.e. plot, of your method for calculating the area of a circle

# My Circle Area Visualization



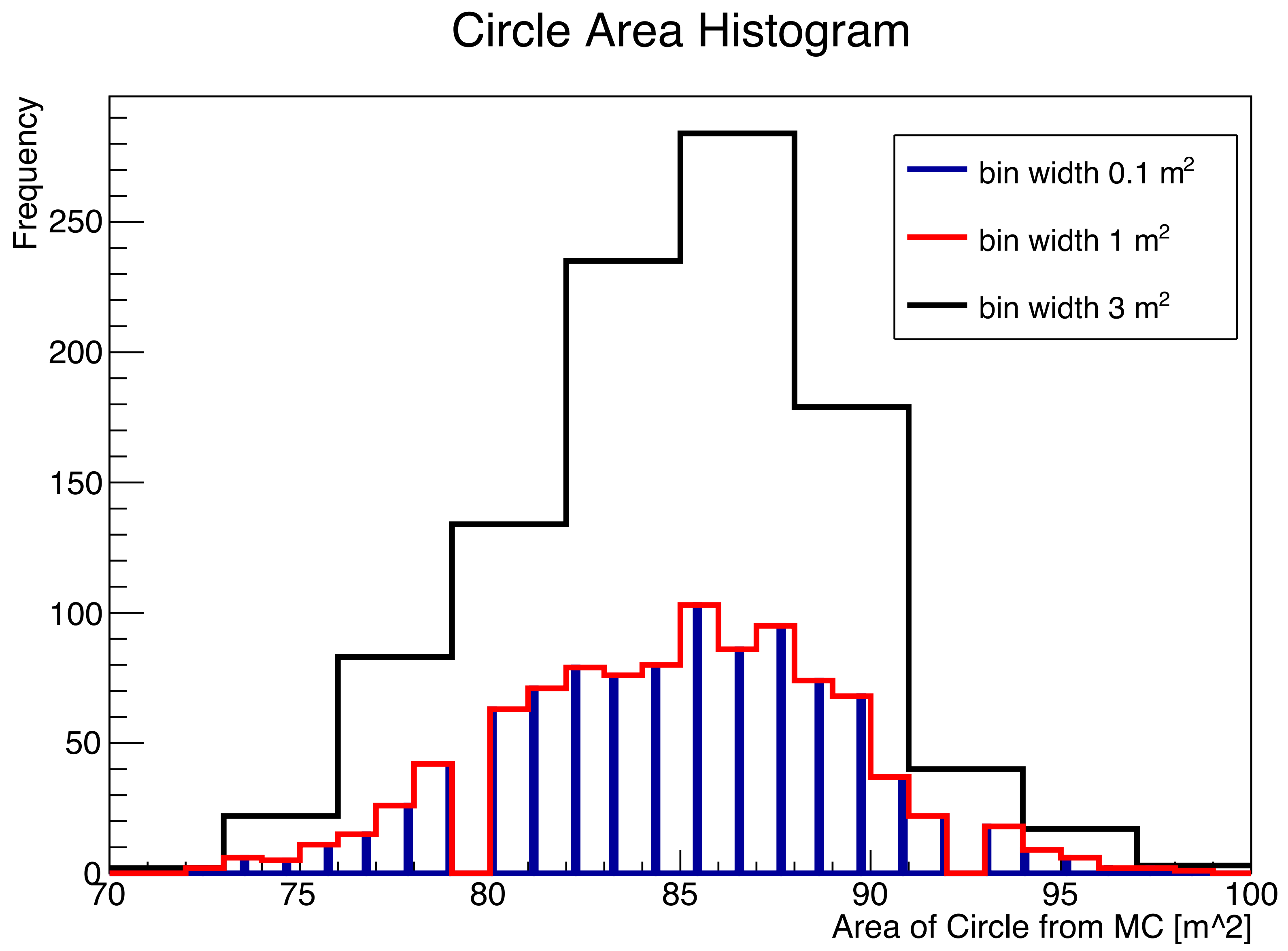
- This is the classic illustration. Can you show your method in a different format that is understandable?

# Random Number Generator

- Do we really want a ‘random’ generator to be random?
- Using the previous code for estimating the area of a circle, plot the resulting area value for 1000 separate tests using 100 throws per test for a radius of 5.2 meters
  - Does your previous code show actual randomness? How would you know?
  - Make histograms of the frequency of the area for the <u>same</u> 1000 separate trials, i.e. not 3 separate histograms of 3 different 1000 trials
    - Using bin widths of 3 $m^2$, 1 $m^2$ , and 0.1 $m^2$
    - Are there gaps w/ no entries for certain values of the area? Should there be?

# Dissecting a Plot

- I made 1,000 separate Monte Carlo trials, each having 100 random number generator throws to calculate the area of a circle with radius of 5.2 m

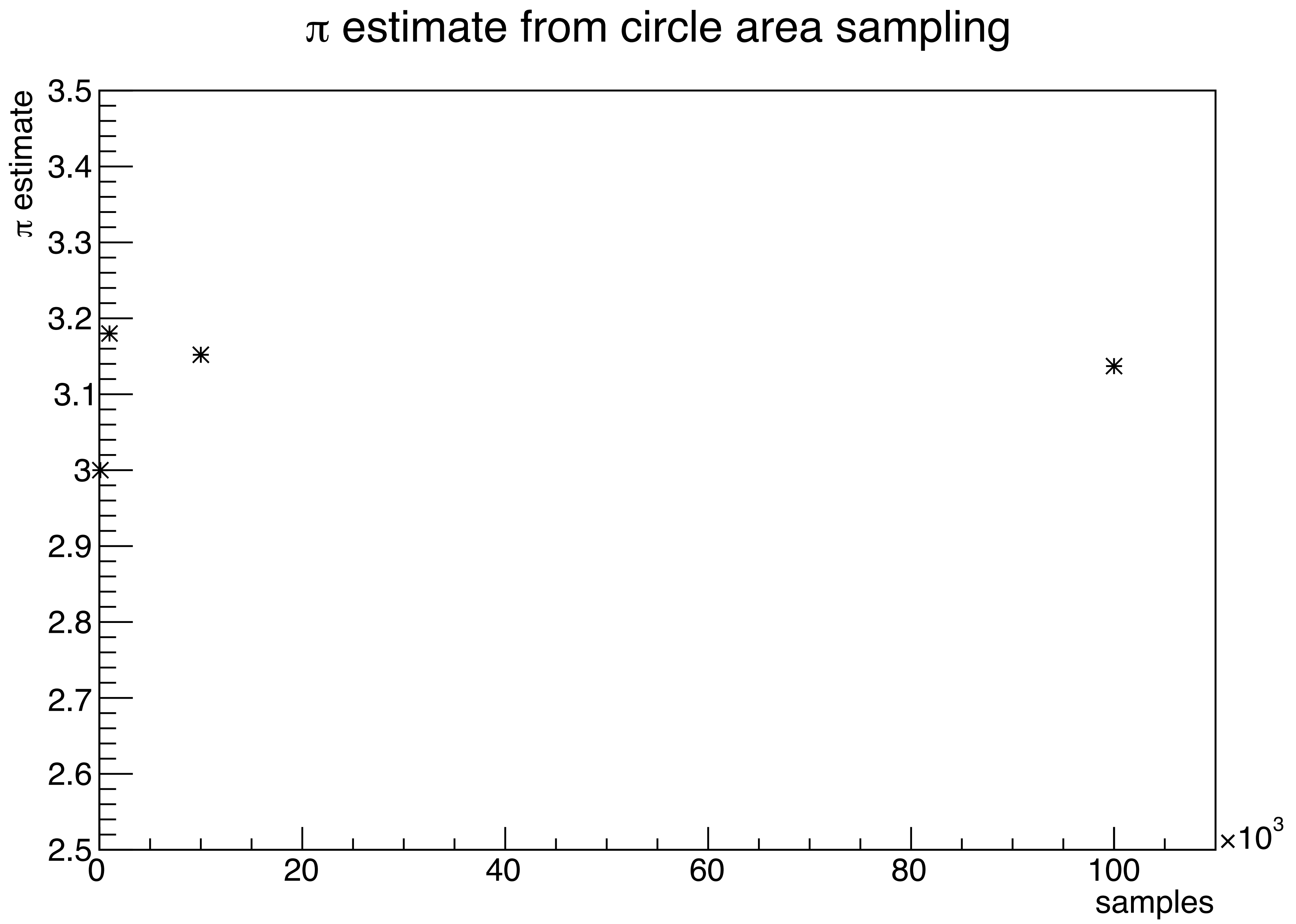


## Calculate the Precision of Pi

- Using your Monte Carlo and the circle equation $x^2+y^2=r^2$ find out the value of π knowing that the area is $\pi r^2$
  - After 10 successive 'throws' of your random number generator you have an estimate of the circle area. A bad estimate, but an estimate nonetheless. Using that estimate of the area, and knowing the radius ($r=5.2$), you can estimate π.
  - Repeat the estimation of π after 10 throws, 100 throws, 1000 throws, 10000 throws, and 100000 throws
  - Plot. On the y-axis have the estimate of π and on the x-axis have the number of throws.

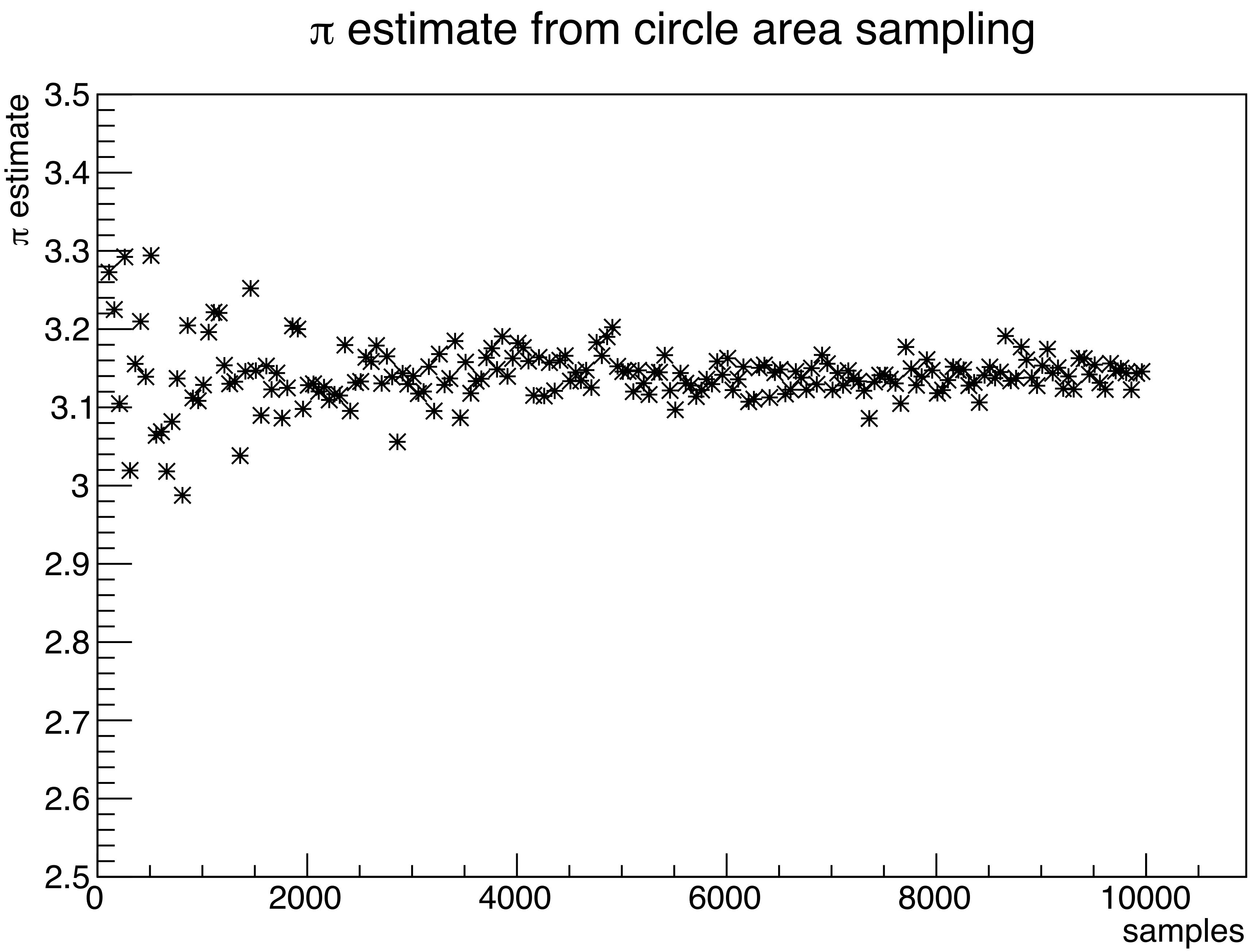
# Calculate the Precision of Pi (1)

- Repeat the estimation of π after 10 throws, 100 throws, 1000 throws, 10000 throws, and 100000 throws



# Calculate the Precision of Pi (2)

- Repeat the estimation of π for at least 100 different sampling points between 1-10000



# Pi and the Central Limit Theorem

- On Tuesday I talked briefly about the Central Limit Theorem, i.e. for a large number of measurements of continuous variables the outcome approaches a gaussian distribution.
- Use your new found Monte Carlo simulation and estimates of π, see if the CLT holds for your estimation method and random number generator choice.
    - If you use 100 throws and to estimate π repeat it tens, hundreds, thousands, etc. of times is the ensuing collect of estimates a gaussian distribution?

# Exercise

- Get your favorite random number generator (and I know that you all have favorites) and break it
    - You can write your own, or use some package (numpy, R, ROOT, javascript online, actually roll dice or repeatedly flip 53 coins and convert to binary, etc…)
    - Sample enough times and in a specific range to show that the random number generator is not actually random.
    - This can be either by method, i.e. values start to repeat after some amount of iterations, or because the computer variable precision you give it is bad, or because the method internally uses finite precision variables
- The goal is to find if you can get it to not be random at some point. Quite possibly your favorite generator is too good to break even by direct attempt.

## Slack Channel

- Join the slack-team AdvancedMethodsKU2016
- sign in with your ***@alumni.ku.dk -mail
- Choose password and username
- Profit