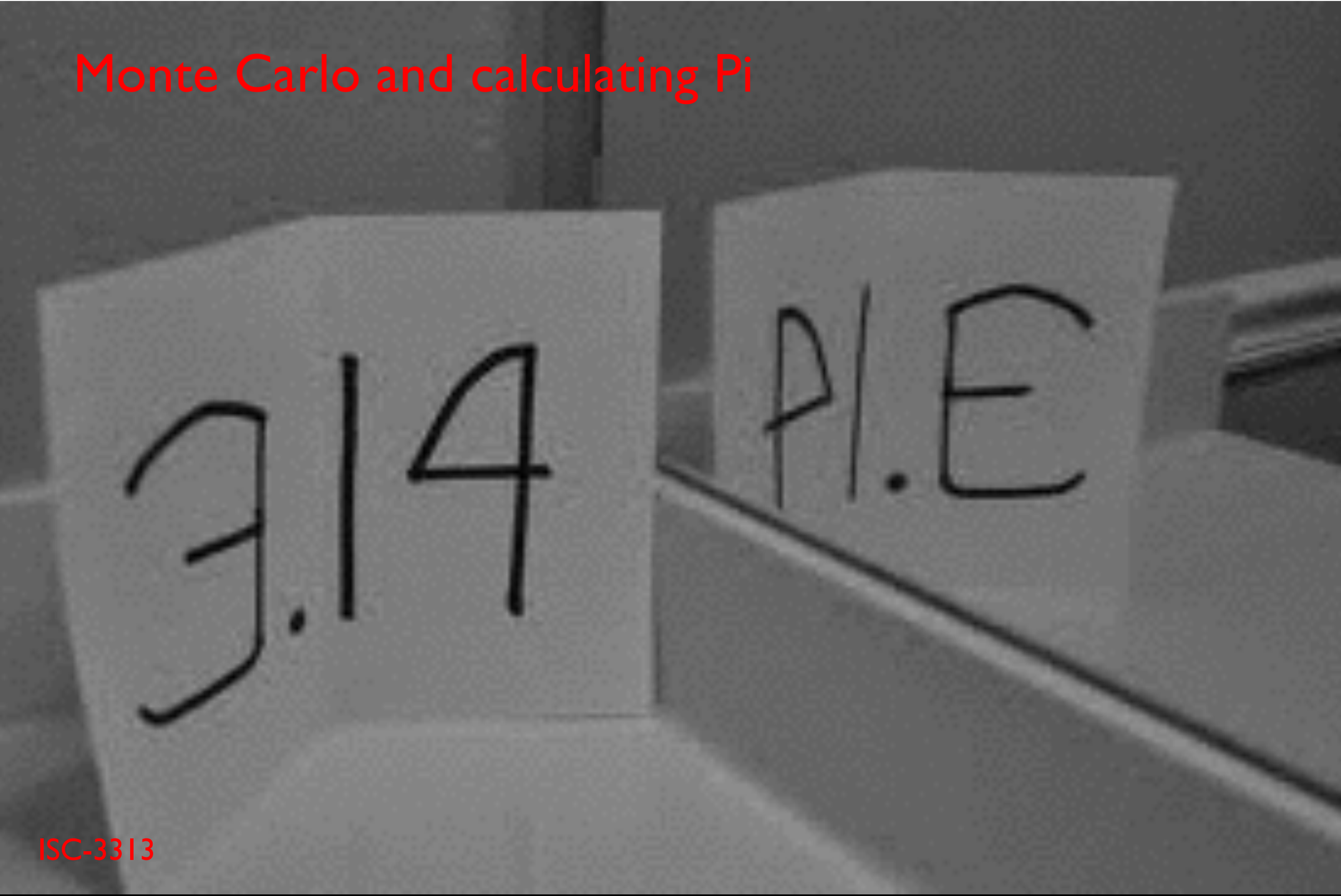
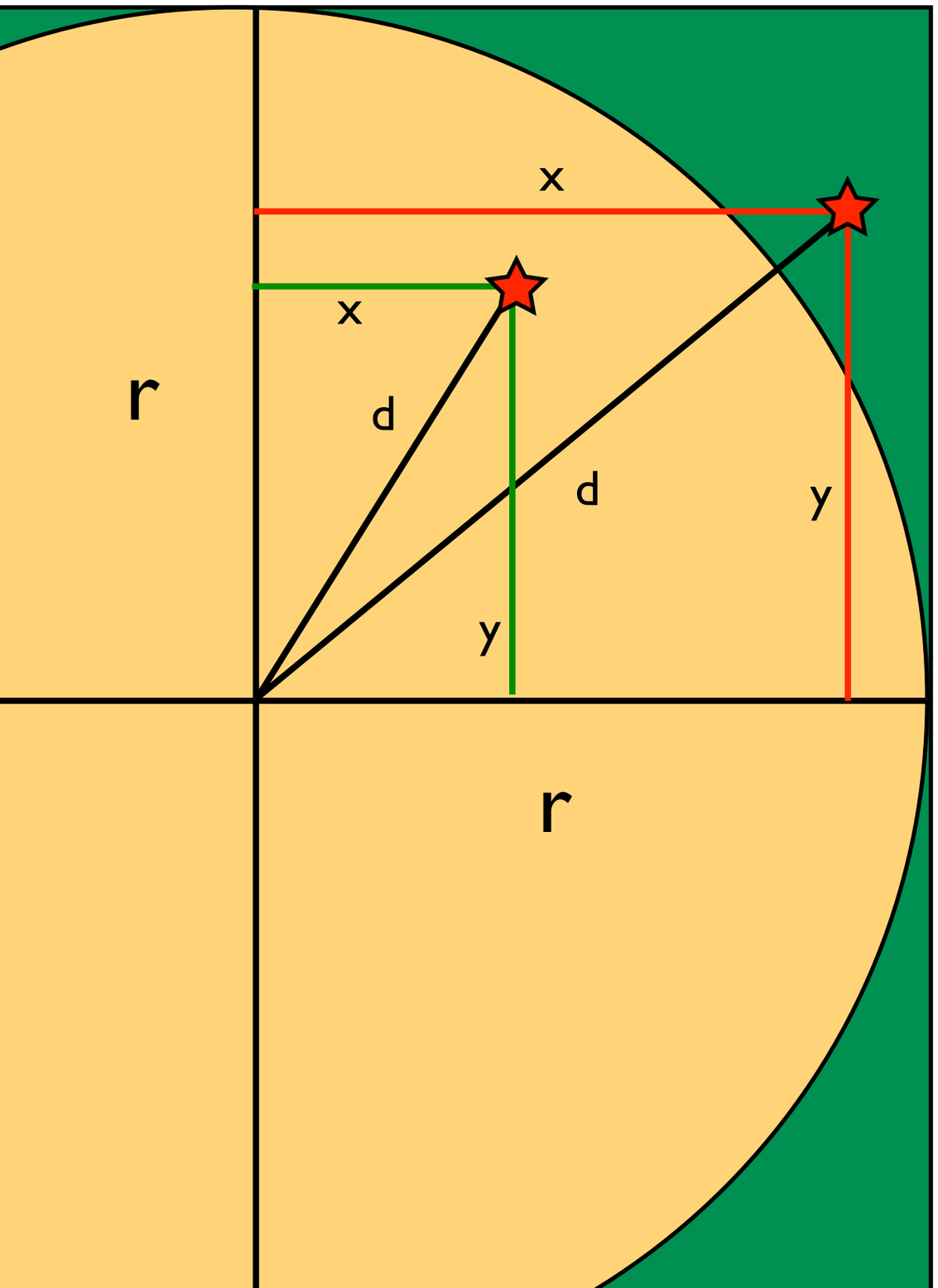
Monte Carlo and calculating Pi



ISC-3313

How to calculate π



$$\frac{A_c}{A_s} = \frac{r^2}{\frac{\pi}{4}r^2} = \frac{\pi}{4}$$

The goal is now to estimate the ratio of the areas. We can devise an algorithm that draws random coordinates from the square and marks whether the coordinate fell into the circle or not. We can calculate the distance from the circle center using Pythagoras:

$$d = \sqrt{(x^2 + y^2)}$$

If d is smaller than r than we know the coordinate is in the circle otherwise only in the square. We can now create an algorithm for our program.

THE FEYNMAN ALGORITHM

[1] WRITE DOWN THE PROBLEM

[2] THINK VERY HARD

[3] WRITE DOWN THE ANSWER

C++ coding

- think about the algorithm
- *write it down, I usually use comments in the code file*
- *work on parts and test often (do NOT write the whole program and never test: will fail with certainty.*

How are variables defined

Fundamental data types: The values of variables are stored somewhere in an unspecified location in the computer memory as zeros and ones. Our program does not need to know the exact location where a variable is stored; it can simply refer to it by its name. What the program needs to be aware of is the kind of data stored in the variable. It's not the same to store a simple integer as it is to store a letter or a large floating-point number; even though they are all represented using zeros and ones, they are not interpreted in the same way, and in many cases, they don't occupy the same amount of memory.

Fundamental data types are basic types implemented directly by the language that represent the basic storage units supported natively by most systems. They can mainly be classified into:

- **Character types:** They can represent a single character, such as `'A'` or `'$'`. The most basic type is `char`, which is a one-byte character. Other types are also provided for wider characters.
- **Numerical integer types:** They can store a whole number value, such as `7` or `1024`. They exist in a variety of sizes, and can either be *signed* or *unsigned*, depending on whether they support negative values or not.
- **Floating-point types:** They can represent real values, such as `3.14` or `0.01`, with different levels of precision, depending on which of the three floating-point types is used.
- **Boolean type:** The boolean type, known in C++ as `bool`, can only represent one of two states, `true` or `false`.

Here is the complete list of fundamental types in C++:

Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than `char`. At least 16 bits.
	char32_t	Not smaller than `char16_t`. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as `char`. At least 8 bits.
	signed **short** *int*	Not smaller than `char`. At least 16 bits.
	signed **int**	Not smaller than `short`. At least 16 bits.
	signed **long** *int*	Not smaller than `int`. At least 32 bits.
	signed **long long** *int*	Not smaller than `long`. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than `float`
	long double	Precision not less than `double`
Boolean type	**bool**	
Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	

Size	Unique representable values	Notes
8-bit	256	$= 2^8$
16-bit	65 536	$= 2^{16}$
32-bit	4 292 967 296	$= 2^{32}$ (~4 billion)
64-bit	18 446 744 073 309 551 616	$= 2^{64}$ (~18 billion billion)

`type identifier = initial_value;`
For example, to declare a variable of type `int` called `x` and initialize it to a value of zero from the same moment it is declared, we can write:

```
int a=5;
```

A second method, known as *constructor initialization* (introduced by the C++ language), encloses the initial value between parentheses (`()`):

`type identifier (initial_value);`
For example:

```
int a (5);
```

Finally, a third method, known as *uniform initialization*, similar to the above, but using curly braces (`{}`) instead of parentheses (this was introduced by the revision of the C++ standard, in 2011):

`type identifier {initial_value};`
For example:

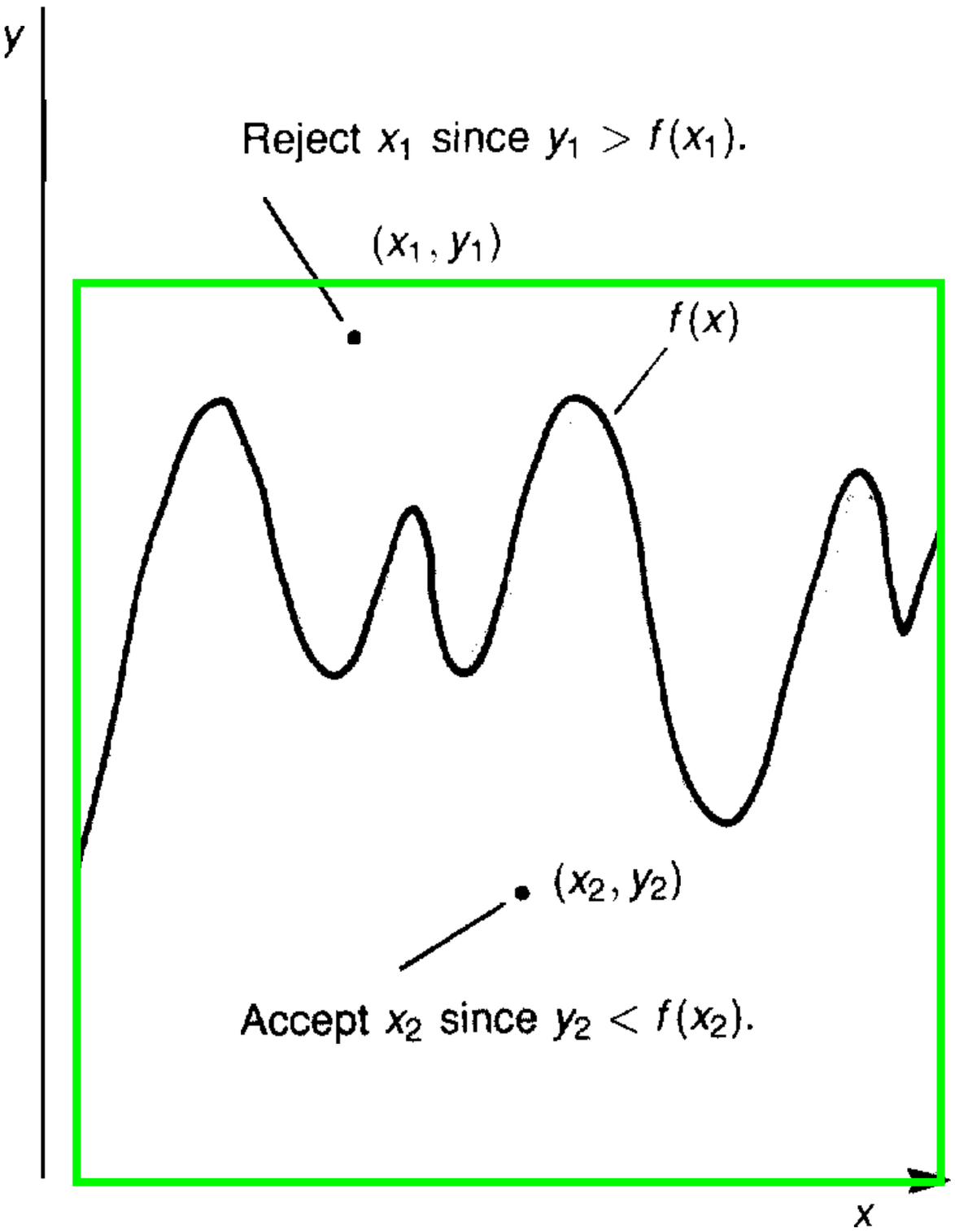
```
int a {5};
```

needs C++11

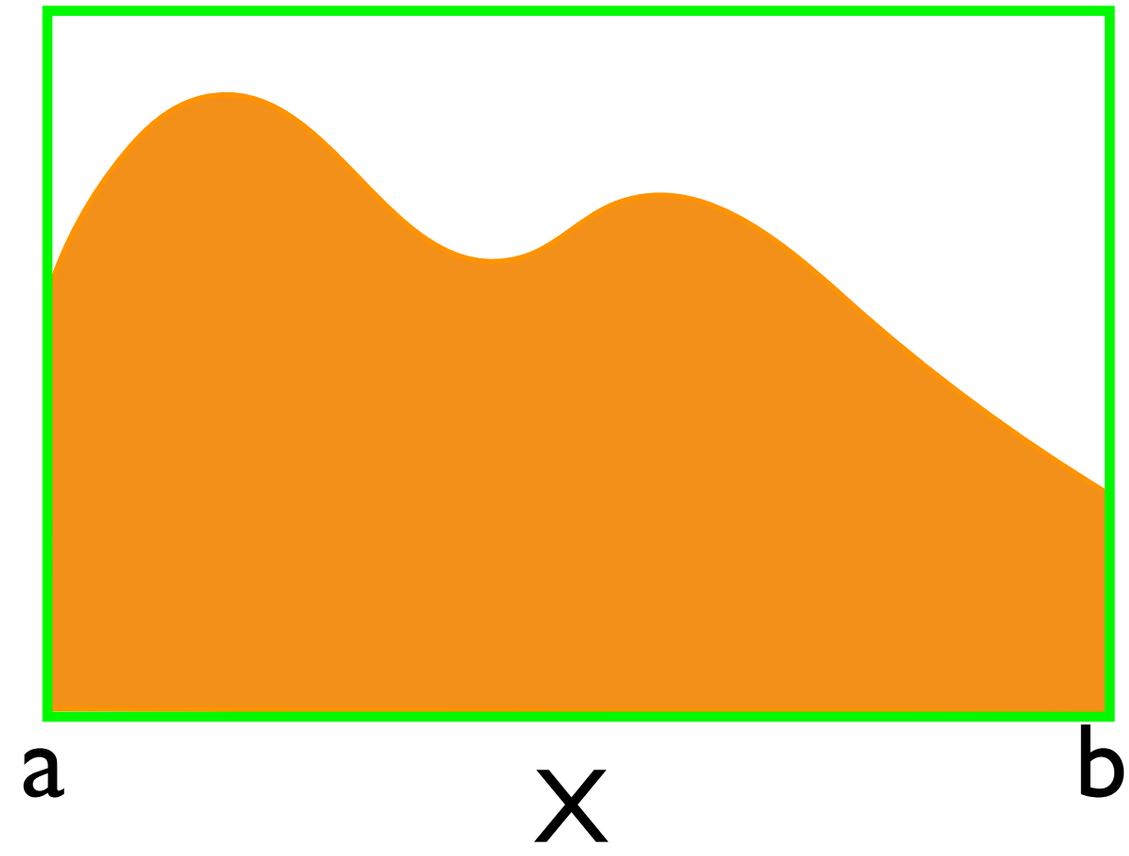
we work on the von Neumann random number generator
and we develop the Π estimator

THE ACCEPTANCE-REJECTION METHOD

Fig. 4. If two independent sets of random numbers are used, one of which (x^i) extends uniformly over the range of the distribution function f and the other (y^i) extends over the domain of f, then an acceptance-rejection technique based on whether or not $y^i \leq f(x^i)$ will generate a distribution for (x^i) whose density is $f(x^i)\,dx^i$.



Acceptance-Rejection method



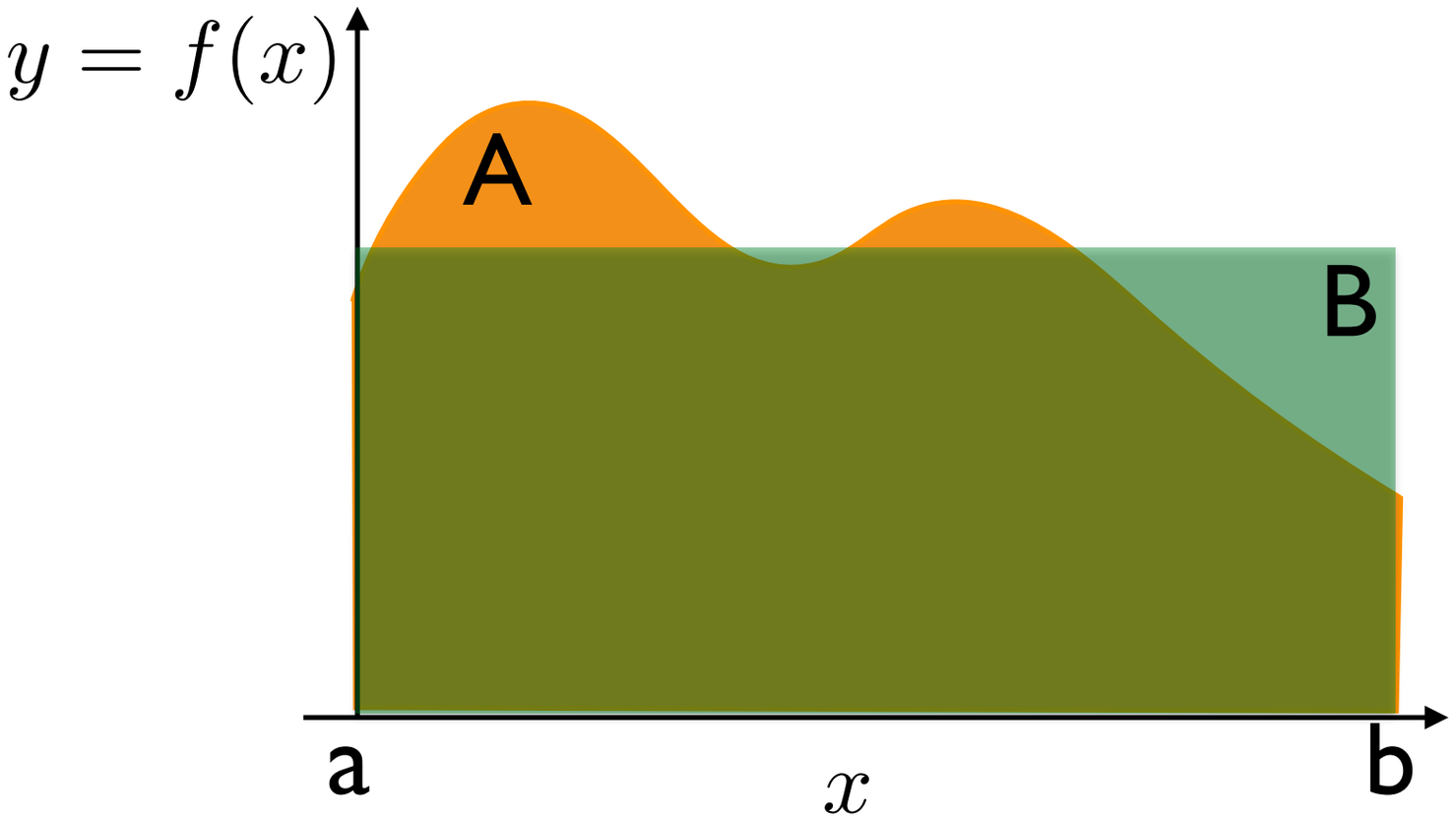
$$M = \frac{\text{Area under curve}}{\text{Bounding box}} \qquad \text{Area under curve} = A_b \frac{\#inside}{\#bounding\ box} \qquad \text{Bounding box} = A_b = M\,(b-a)$$

- How to choose the bounding box? It need to be big enough to contain the whole function. But if it is too big then we draw often random numbers above the function. If the bounding box is much larger than the area under the curve then we need many draws (or steps) to get a good accuracy.
- We need to draw two random numbers and 'discard' the draws that are above the function.

We need to be able to calculate f(x) for any possible x with the range a and b.

We draw 2 random values, one for x and one for y. then evaluate f(x), if y < f(x) then we count this as #inside. The #bounding box is the total number of draws.

Usual Monte Carlo Integration



$$A = \int_a^b f(x)dx$$

$$B = (b-a)f(c)$$

There must be an $f(c)$ that satisfies A = B.

Find $f(c)$ and we are done!

$$f(c) = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

where x_i are drawn uniformly between a and b

Monte Carlo Integration Algorithm

- Draw many x_i *between the boundaries a and b*
- *Calculate the average* $\overline{x}$ *of the collected* x_i
- *Calculate the area as (b-a)* $\overline{x}$