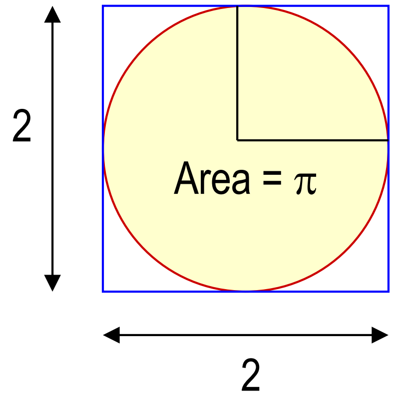
Example: Computing the value of π



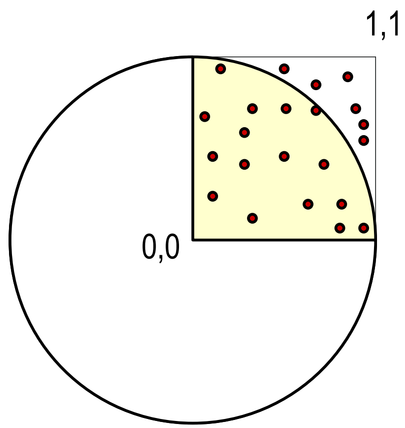
- We have a circle with radius 1 inside a square of size 2 x 2 with its centre in origo
  – the area of the circle is $\pi r^2 = \pi$, since $r = 1$
  – the area of the square is 4
  – the ratio between the area of the circle and the area of the square is $\pi/4$

- Algorithm to compute the value of π:
  – draw a large number of random points inside the square formed by the quadrant with positive coordinates
  – count how many of the points are within the circle
  – the fraction of points within the circle will be $\pi/4$
- Uses a Monte Carlo method

Monte Carlo methods

- Monte Carlo methods are randomized algorithms used to solve numerical or scientific problems
  - approximative methods, do not guarantee an exact solution
  - the accuracy of the solution can be improved by increasing the number of steps of the algorithm
- Based on random selections in the program execution
  - need a random number generator to generate pseudo-random numbers
- All calculations are independent of each other
  - very easy to do in parallel, all processes can work independently of each other
  - very efficient parallel solution, we only need communication to collect and combine the results from all processes

Decomposition



- All processes generate *N* random points *(x,y)* in the upper right quadrant delimited by (0,0) and (1,1)
  - draw two random numbers *x* and *y* between 0.0 and 1.0
  - count how many of the sampled points *(x, y)* are inside the circle:

```
if (sqrt(x²+y²) <= 1.0) in_circle++;
```

- The total number of sample points is *sum_n* = *p***N* when using *p* processes
- The total number of points within the circle is *sum_c* = $\sum_{i=0}^{p-1} in\_circle_i$
  - can use a MPI reduction operation to sum the values
- The value of Pi can then be computed as (*sum_c* / *sum_n*) * 4

Generating random numbers

- Pseudo random numbers are often generated using a linear congruential generator $x_{i+1} = (ax_i + c) \bmod m$
  - a, c and m are constants
  - the sequence is started from some initial seed value, x_0
- All processes should generate independent random numbers
- One solution is to initialize each process to start from different points in the pseudo random number sequence
  - we can use the process identifier as a seed for the random number generator
  - two processes can generate partially identical sequences
- A better solution would be to use a parallel random number generator
  - each process generates independent streams of random numbers
  - parallel random number generators are available in many numerical libraries

Parallel random number generation

- All processes generate pseudo random numbers from the same sequence
  - each process jumps *p* steps forward in the sequence for each new random number
  - ensures that all processes use unique sequences of random numbers
- The first *p* numbers must be generated sequentially
  - after that, each process can generate its own random numbers
- There are several numerical libraries which contain parallel random number generators
  - ACML (AMD Core Math Library)
  - SPRNG (Scalable Parallel Random Number Generator)