Monte Carlo History

- Use random numbers to solve numerical problems
- Early use during development of atomic bomb
- Von Neumann, Ulam, Metropolis
- Named after the casino in Monte Carlo

- Las Vegas algorithms?




Monte Carlo vs Las Vegas

Monte Carlo Methods

- Pros
  - Flexible
  - Easy to implement
  - Easily handles complex integrands
  - Efficient for high dimensional integrands

Monte Carlo Methods

- Pros
  - Flexible
  - Easy to implement
  - Easily handles complex integrands
  - Efficient for high dimensional integrands
- Cons
  - Variance (noise)
  - Slow convergence* $O(1/\sqrt{N})$

Random Variables

- Random variable X

Random Variables

- Random variable X
- Cumulative distribution function (CDF)

$$P(x) = \text{Prob}\{X \leq x\} \quad P(x) \in [0, 1]$$

Random Variables

- Random variable X
- Cumulative distribution function (CDF)

$$P(x) = \text{Prob}\{X \leq x\} \quad P(x) \in [0, 1]$$

- Probability density function (PDF)

$$p(x) = \frac{dP(x)}{dx} \qquad p(x) \geq 0 \qquad \int p(x) = 1$$

Random Variables

- Random variable X
- Cumulative distribution function (CDF)

$$P(x) = \text{Prob}\{X \leq x\} \quad P(x) \in [0, 1]$$

- Probability density function (PDF)

$$p(x) = \frac{dP(x)}{dx} \qquad p(x) \geq 0 \qquad \int p(x) = 1$$

- Therefore

$$\text{Prob}\{a \leq X \leq b\} = \int_a^b p(x)\, dx = P(b) - P(a)$$

Random Variables

- Uniform random variables:

$$p(x) = \frac{dP}{dx}(x) = \text{const}$$

- Canonical uniform random variable ξ

$$p(x) = \begin{cases} 1 & x \in [0,1], \\ 0 & \text{otherwise.} \end{cases}$$

Expected Value & Variance

- Random variable $Y = f(X)$

Expected Value & Variance

- Random variable $Y = f(X)$
- Expected Value: $E[Y] = \int_D f(x)p(x)\ dx$

Expected Value & Variance

- Random variable $Y = f(X)$
- Expected Value: $E[Y] = \int_D f(x)p(x)\ dx$
- Variance: $V[Y] = E[(Y - E[Y])^2]$

Expected Value & Variance

- Random variable $Y = f(X)$
- Expected Value: $E[Y] = \int_D f(x)p(x)\ dx$
- Variance: $V[Y] = E[(Y - E[Y])^2]$
- Properties:

$$E[aY] = aE[Y]$$
$$E[Y_1 + Y_2] = E[Y_1] + E[Y_2]$$
$$V[aY] = a^2V[Y]$$

Expected Value & Variance

- Random variable $Y = f(X)$
- Expected Value: $E[Y] = \int_D f(x)p(x)\ dx$
- Variance: $V[Y] = E[(Y - E[Y])^2]$
- Properties:

$$E[aY] = aE[Y]$$
$$E[Y_1 + Y_2] = E[Y_1] + E[Y_2]$$
$$V[aY] = a^2V[Y]$$

- Therefore:

$$V[Y] = E[Y^2] - E[Y]^2$$

Expected Value

- Expected value:

$$E[Y] = \int_D f(x)p(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

where x_i are distributed according to $p(x_i)$

Expected Value

- Expected value:

$$E[Y] = \int_D f(x)p(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

where x_i are distributed according to $p(x_i)$

- Strong law of large numbers:

$$\text{Prob}\left\{E[Y] = \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} f(x_i)\right\} = 1$$

Monte Carlo Integration

- We want to compute the value F of an integral

$$E\left[Y\right] = \int_D f(x)p(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

$$F = \int_0^1 f(x)dx$$

Monte Carlo Integration

- We want to compute the value F of an integral

$$E\left[Y\right] = \int_D f(x)p(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

$$F = \int_0^1 f(x)dx = \int_D \left[\frac{f(x)}{p(x)}\right] p(x)dx$$

Monte Carlo Integration

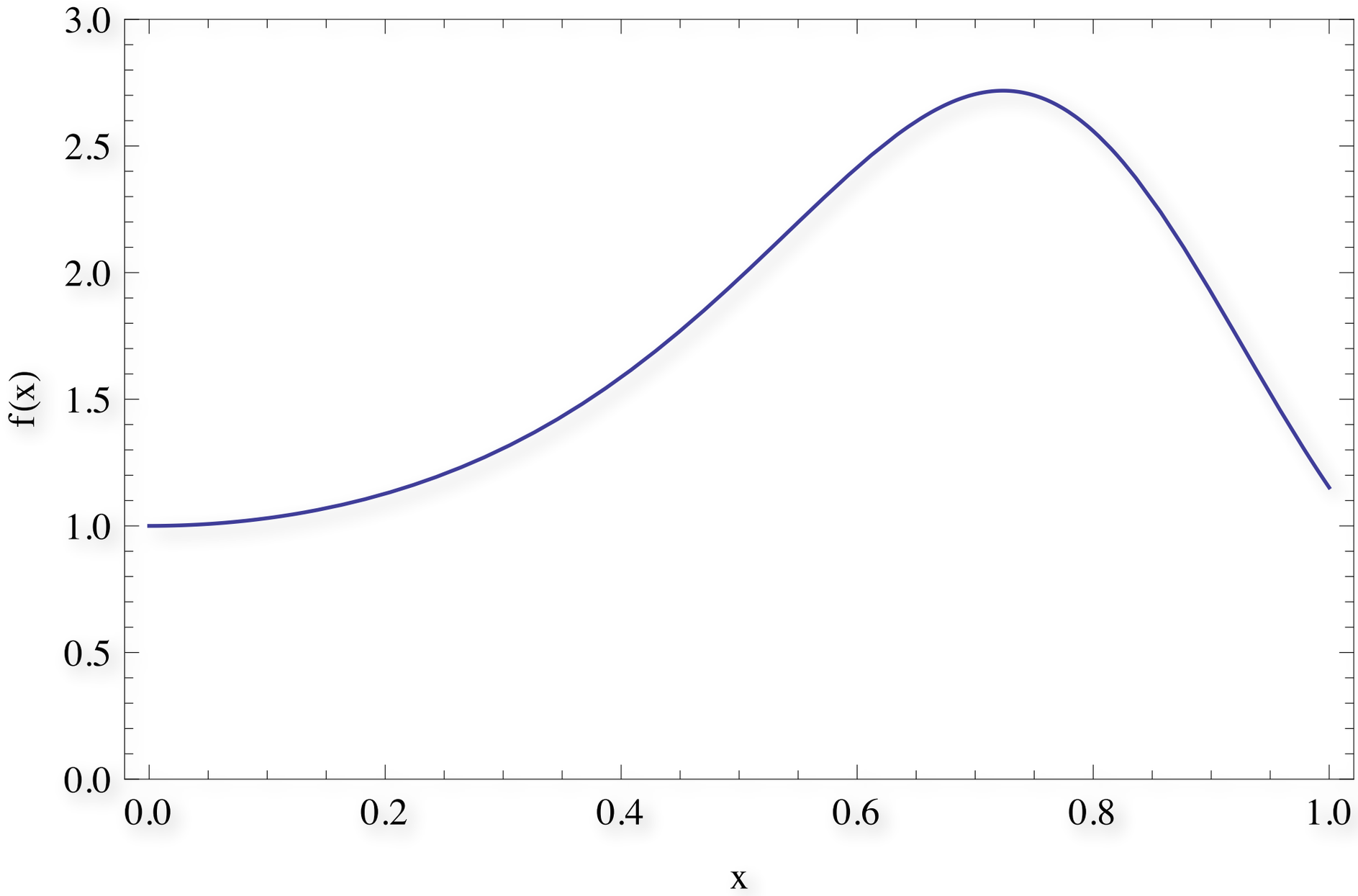
- We want to compute the value F of an integral

$$E\left[Y\right] = \int_D f(x)p(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

$$F = \int_0^1 f(x)dx = \int_D \left[\frac{f(x)}{p(x)}\right] p(x)dx \boxed{\approx \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)} = F_N}$$

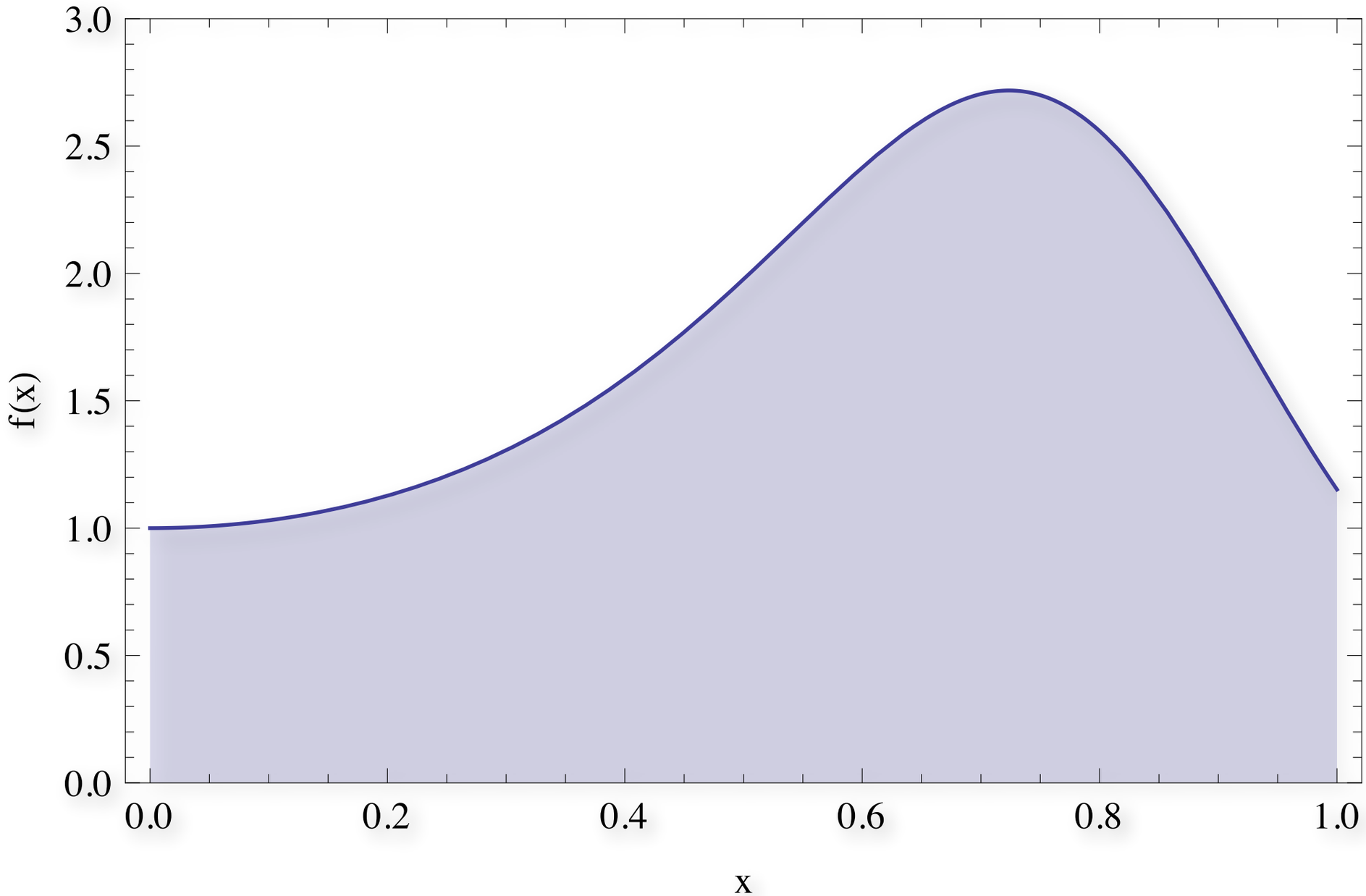
Monte Carlo Integration

$$f(x) = e^{\sin(3x^2)}$$



Monte Carlo Integration

$$F = \int_0^1 e^{\sin(3x^2)}\, dx$$



Monte Carlo Integration

$$F = \int_0^1 e^{\sin(3x^2)}\,dx \approx F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}$$

```
double integrate(int N)
{
   double x, sum=0.0;
   for (int i = 0; i < N; ++i) {
      x = drand48();
      sum += exp(sin(3*x*x));
   }
   return sum / double(N);
}
```

Monte Carlo Integration

$$F = \int_0^1 e^{\sin(3x^2)}\,dx \approx F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}$$

```
double integrate(int N)
{
   double x, sum=0.0;
   for (int i = 0; i < N; ++i) {
      x = drand48();
      sum += exp(sin(3*x*x));
   }
   return sum / double(N);
}
```

$$p(x_i) = 1$$

Monte Carlo Integration

$$F = \int_0^1 e^{\sin(3x^2)}\,dx \approx F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)} \Rightarrow \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

```
double integrate(int N)
{
   double x, sum=0.0;
   for (int i = 0; i < N; ++i) {
      x = drand48();
      sum += exp(sin(3*x*x));
   }
   return sum / double(N);
}
```

$$p(x_i) = 1$$

Monte Carlo Integration

$$F = \int_a^b e^{\sin(3x^2)}\,dx \approx F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}$$

```
double integrate(int N, double a, double b)
{
   double x, sum=0.0;
   for (int i = 0; i < N; ++i) {
      x = drand48();
      sum += exp(sin(3*x*x));
   }
   return sum / double(N);
}
```

Monte Carlo Integration

$$F = \int_a^b e^{\sin(3x^2)}\,dx \approx F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}$$

```
double integrate(int N, double a, double b)
{
   double x, sum=0.0;
   for (int i = 0; i < N; ++i) {
      x = a + drand48()*(b-a);
      sum += exp(sin(3*x*x));
   }
   return sum / double(N);
}
```

Monte Carlo Integration

$$F = \int_a^b e^{\sin(3x^2)}\,dx \approx F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}$$

```
double integrate(int N, double a, double b)
{
   double x, sum=0.0;
   for (int i = 0; i < N; ++i) {
      x = a + drand48()*(b-a);
      sum += exp(sin(3*x*x));
   }
   return sum / double(N);
}
```

$$p(x_i) = \frac{1}{b-a}$$

Monte Carlo Integration

$$F = \int_a^b e^{\sin(3x^2)}\,dx \approx F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)} \Rightarrow \frac{b-a}{N}\sum_{i=1}^{N} f(x_i)$$

```
double integrate(int N, double a, double b)
{
   double x, sum=0.0;
   for (int i = 0; i < N; ++i) {
      x = a + drand48()*(b-a);
      sum += exp(sin(3*x*x));
   }
   return sum * (b-a) / double(N);
}
```

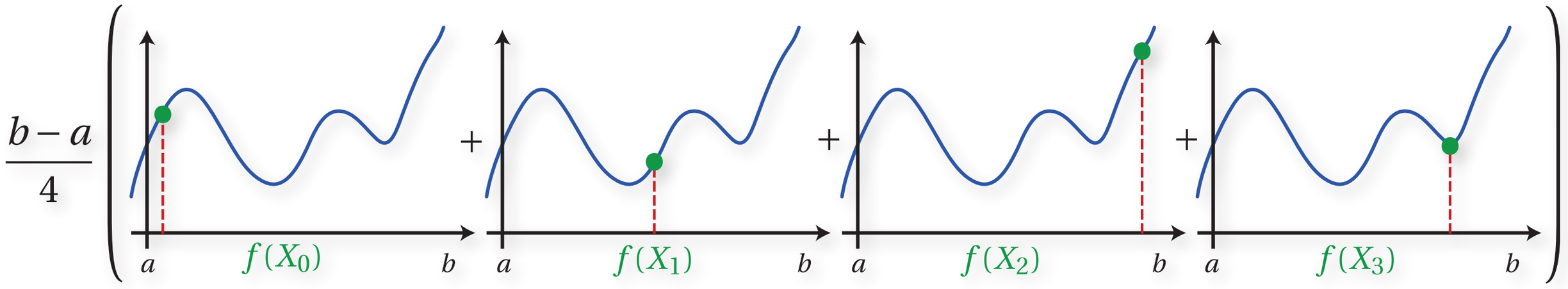
$$p(x_i) = \frac{1}{b-a}$$

Monte Carlo Integration

$$f(x) = e^{\sin(3x^2)}$$

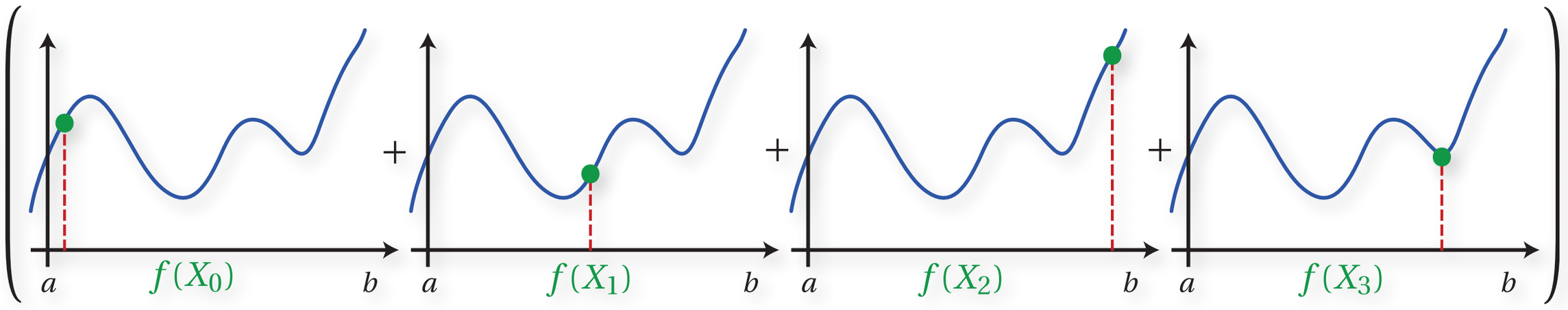
N	F
1	2.75039
10	1.9893
100	1.79139
1000	1.75146
10000	1.77313
100000	1.77862

Monte Carlo Visual Interpretation

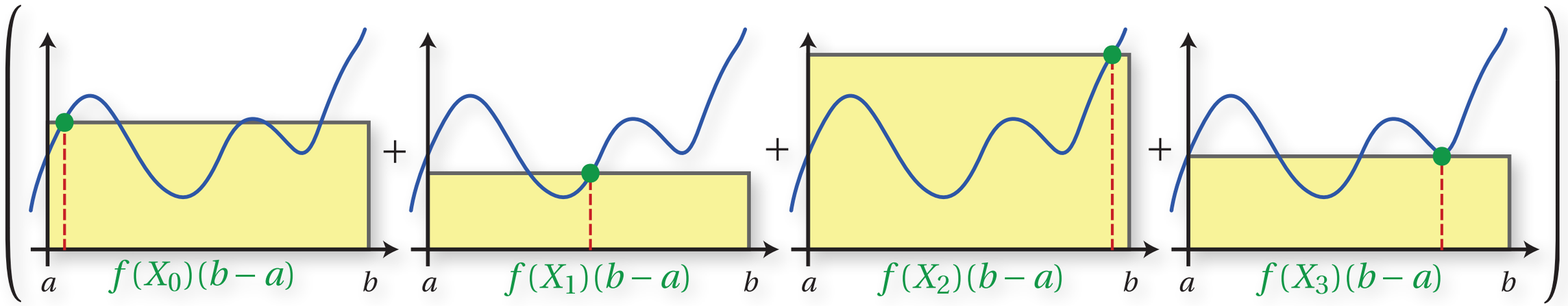


Monte Carlo Visual Interpretation

$$\frac{b-a}{4}\left(f(X_0) + f(X_1) + f(X_2) + f(X_3) \right)$$



$$\frac{1}{4}\left(f(X_0)(b-a) + f(X_1)(b-a) + f(X_2)(b-a) + f(X_3)(b-a) \right)$$



Monte Carlo Integration Summary

- Goal: evaluate integral $\int_a^b f(x)dx$

Monte Carlo Integration Summary

- Goal: evaluate integral $\int_a^b f(x)dx$

- Random variable $X_i \sim p(x)$

Monte Carlo Integration Summary

- Goal: evaluate integral $\int_a^b f(x)dx$

- Random variable $X_i \sim p(x)$

- Monte Carlo Estimator $F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}$

Monte Carlo Integration Summary

- Goal: evaluate integral $\int_a^b f(x)dx$

- Random variable $X_i \sim p(x)$

- Monte Carlo Estimator $F_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}$

- Expectation $E[F_N] = \int_a^b f(x)dx$

Monte Carlo Estimator

- *Unbiased* estimator
    - expected value equals integral

Monte Carlo Estimator

- *Unbiased* estimator
    - expected value equals integral
- Extension to higher dimensions straightforward
    - number of samples independent of dimension (cf. standard quadrature)

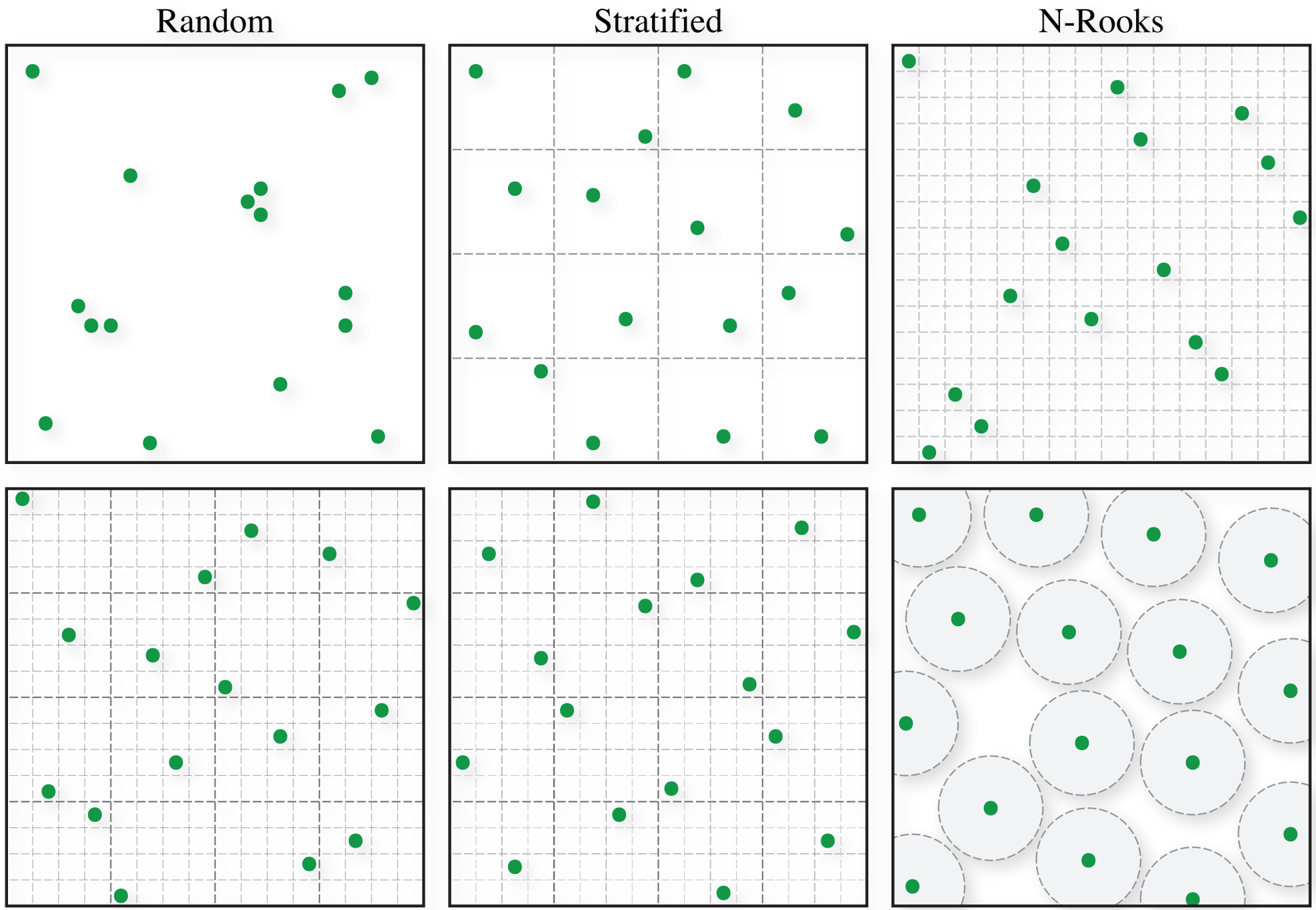
Monte Carlo Estimator

- *Unbiased* estimator
    - expected value equals integral
- Extension to higher dimensions straightforward
    - number of samples independent of dimension (cf. standard quadrature)
- Convergence* $O(1/\sqrt{N})$

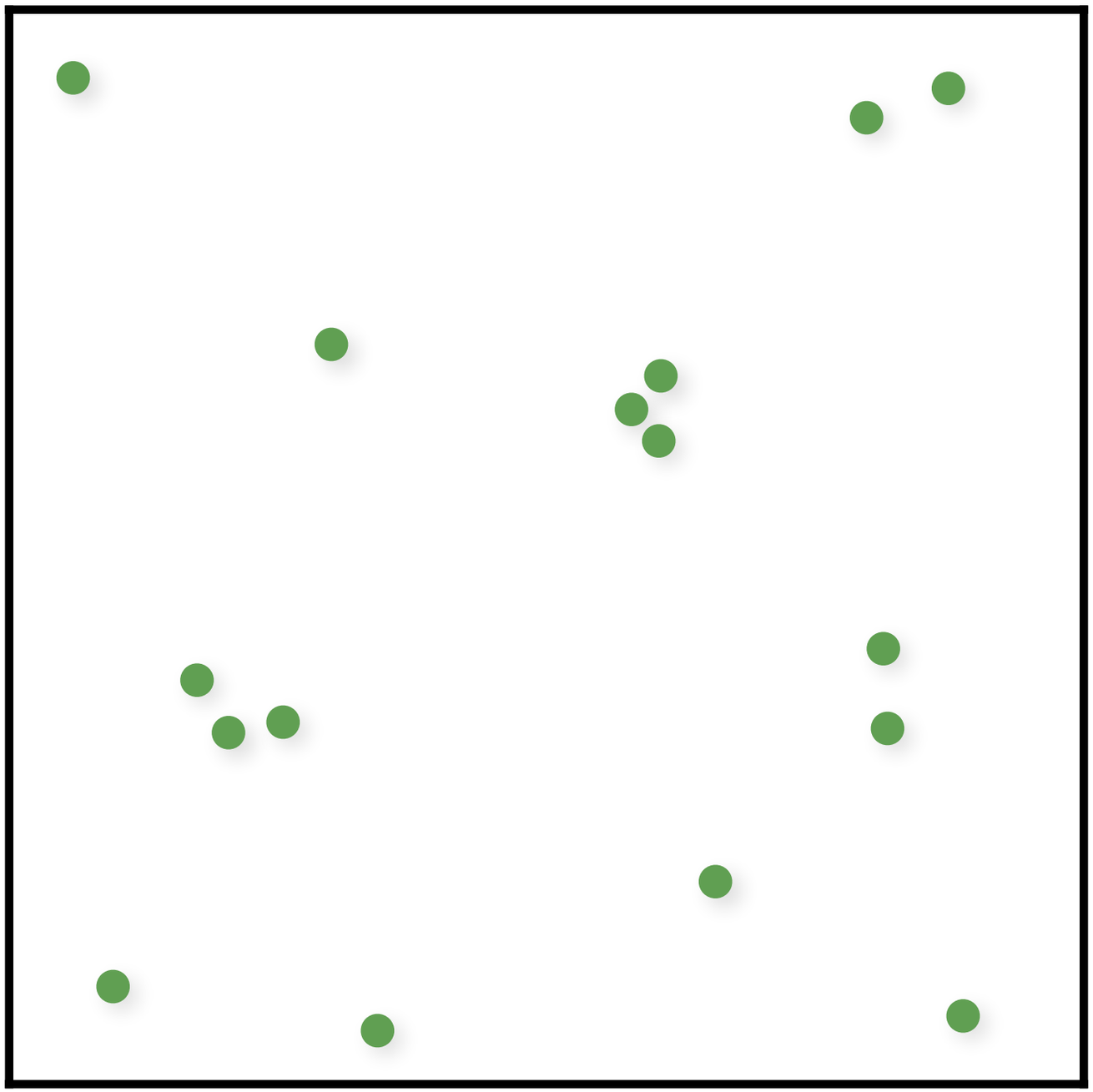
Monte Carlo Estimator

- *Unbiased* estimator
    - expected value equals integral
- Extension to higher dimensions straightforward
    - number of samples independent of dimension (cf. standard quadrature)
- Convergence* $O(1/\sqrt{N})$
    - reducing the error by a factor of 2 requires 4 times as many samples!

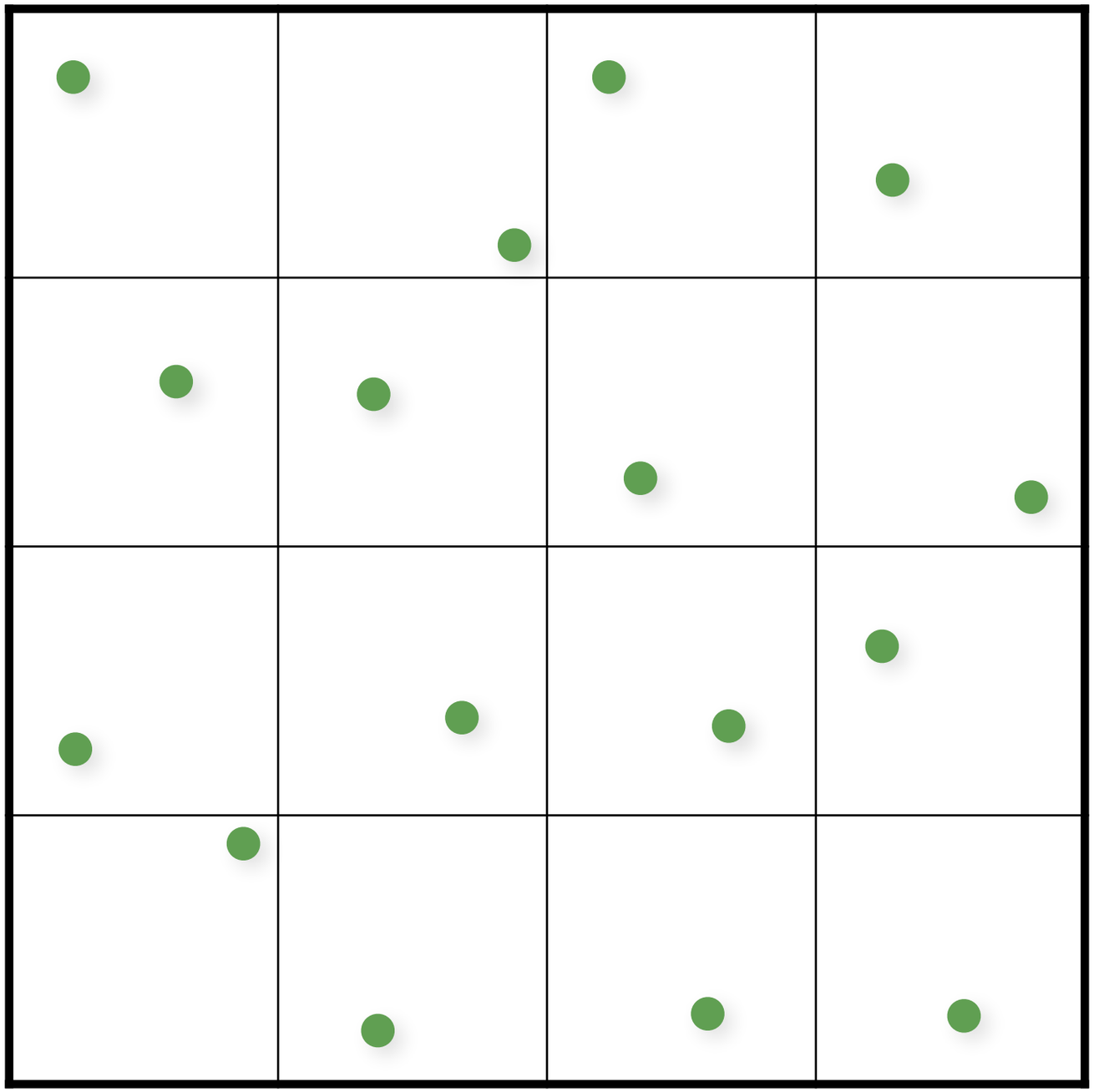
Careful Sample Placement



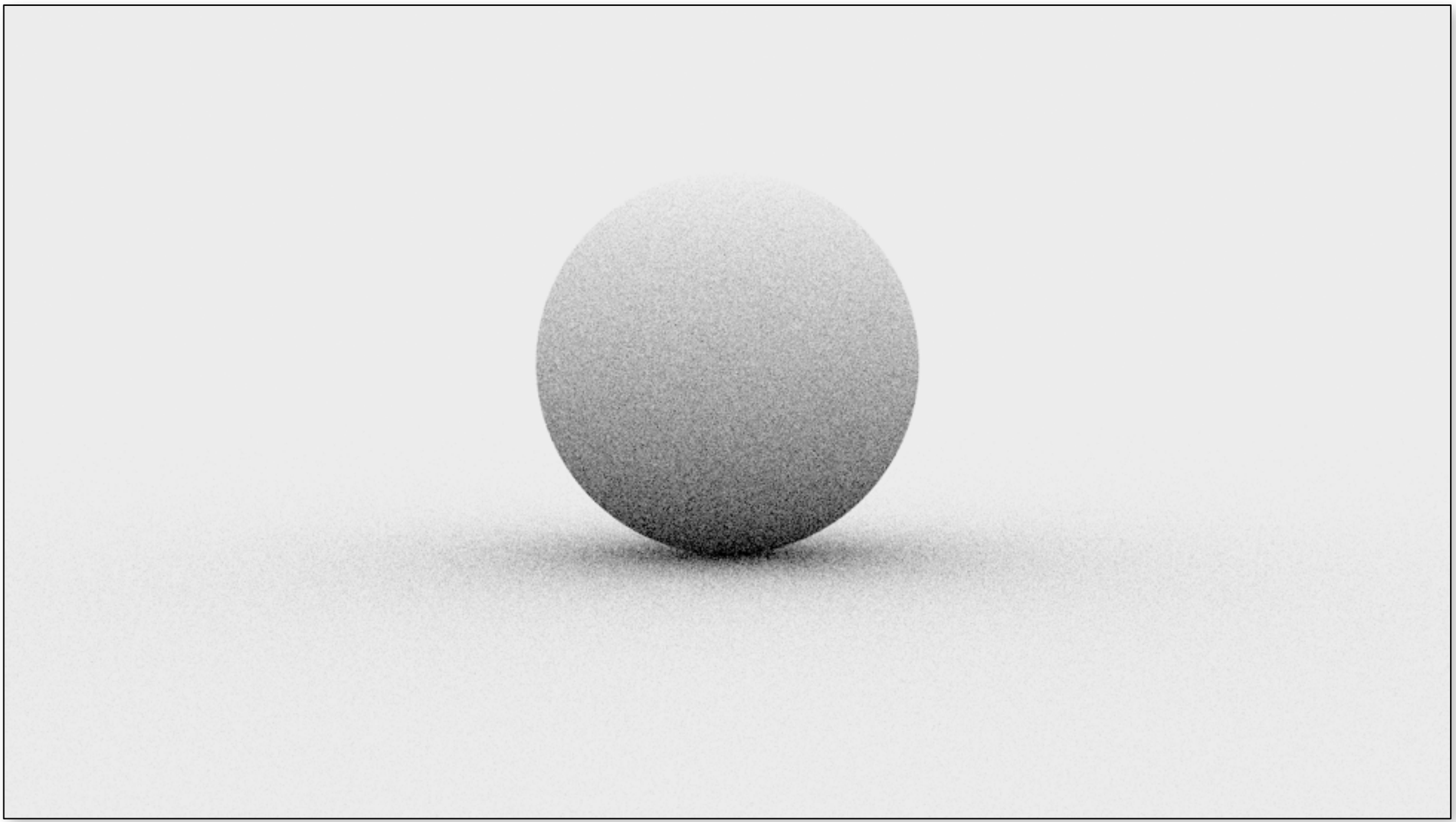
Random Sampling



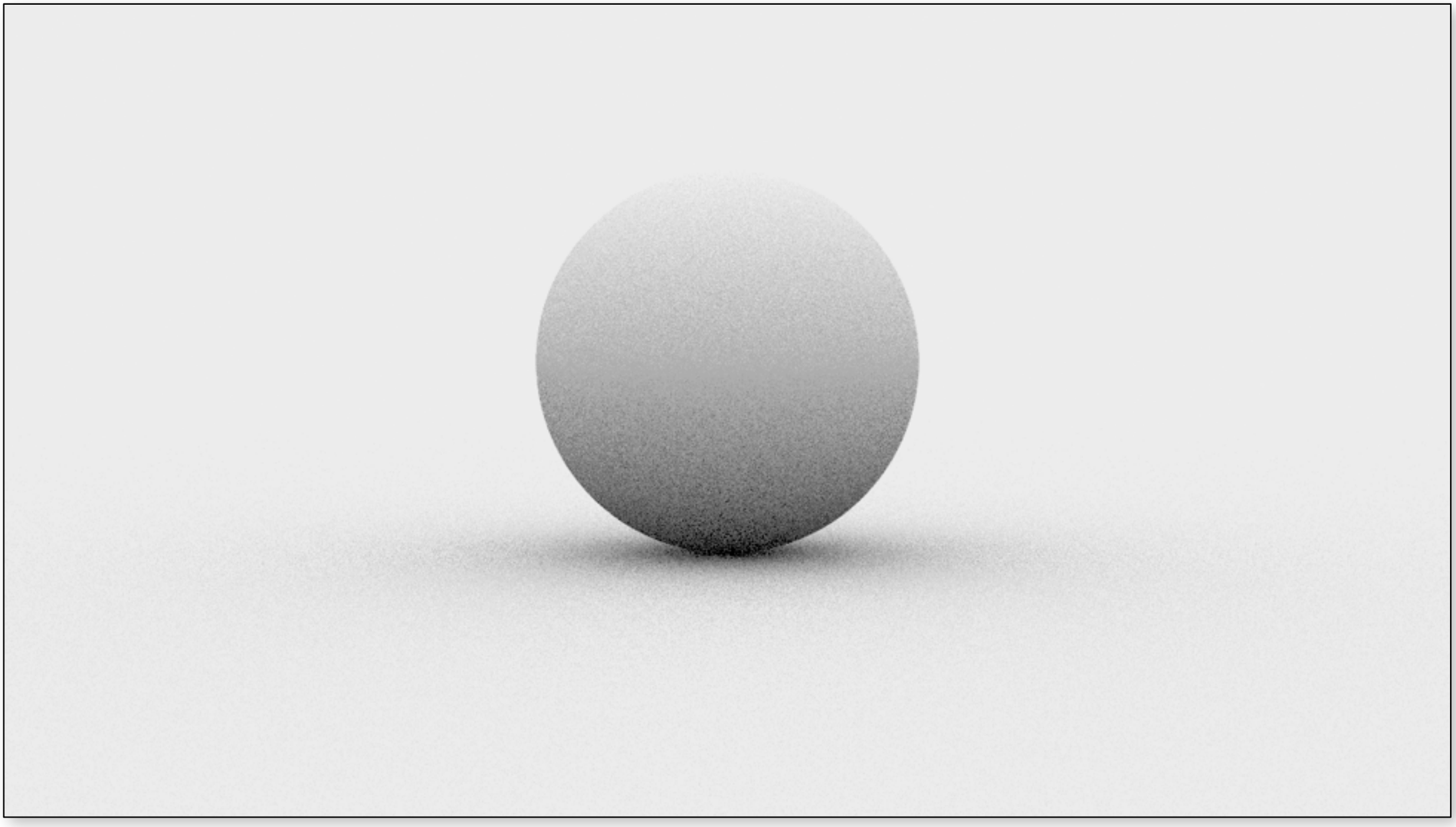
Stratified (Jittered) Sampling



Monte Carlo (16 random samples)



Monte Carlo (16 stratified samples)

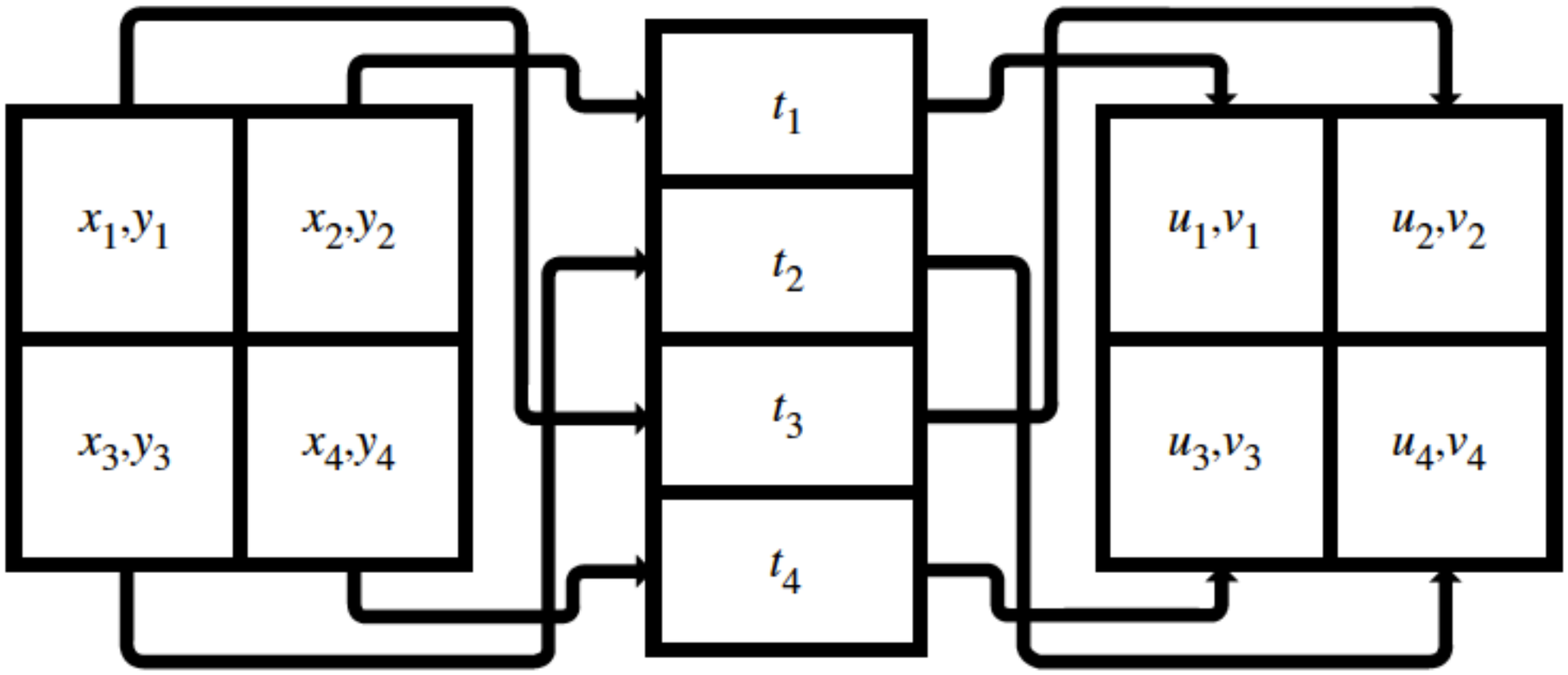


Stratifying in Higher Dimensions

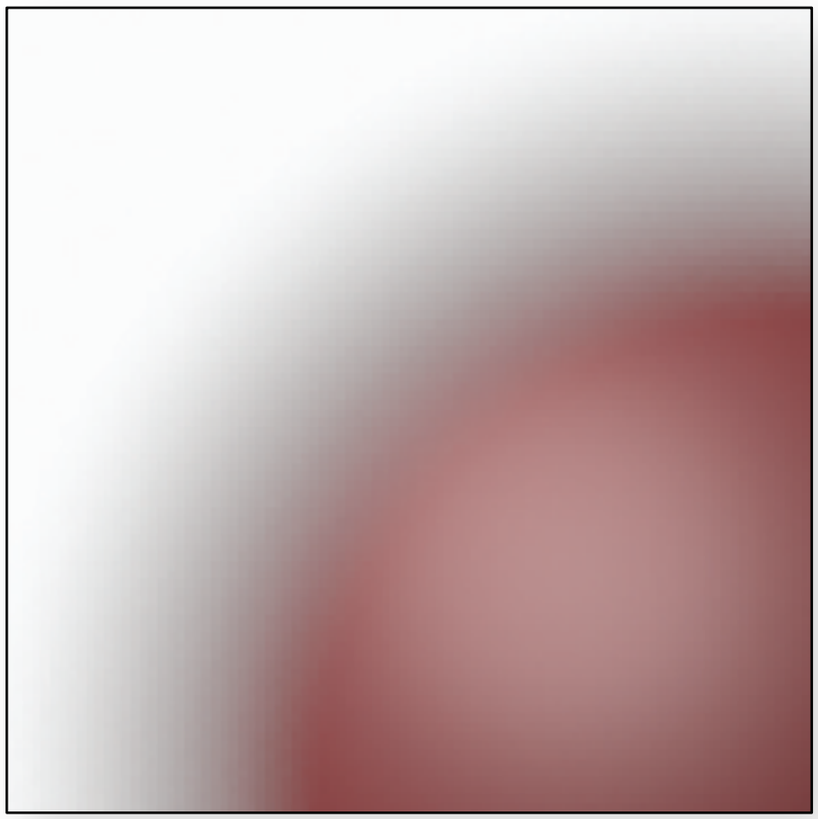
- Stratification requires N^d samples.
- Inconvenient for large d

Stratifying in Higher Dimensions

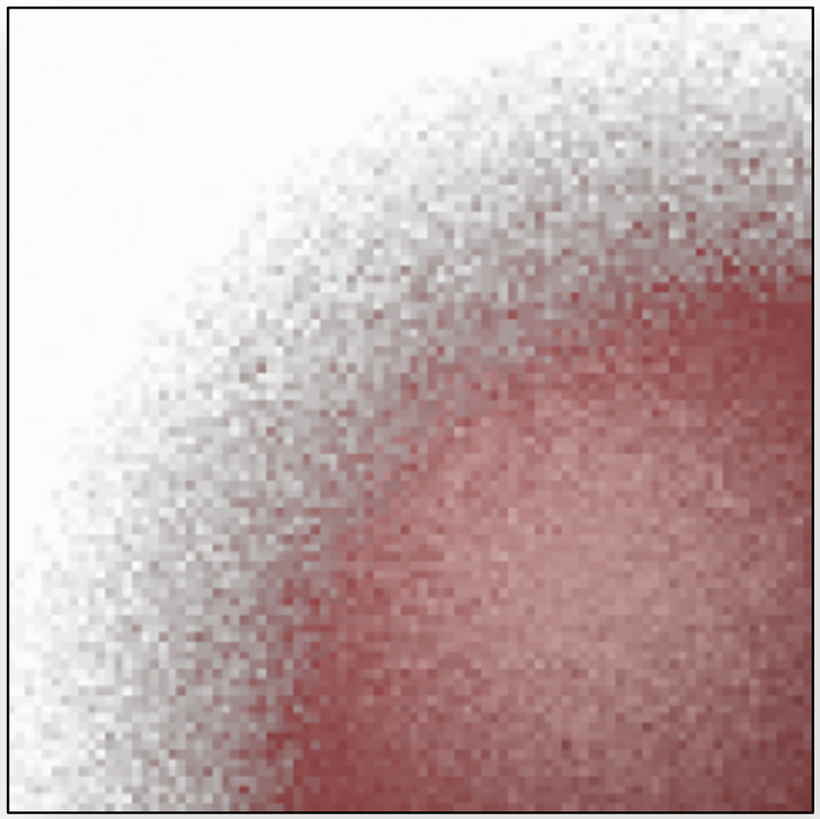
- Stratification requires N^d samples.
- Inconvenient for large d
- Compute stratified samples in lower-dimensions (e.g. (x,y), t, (u,v)), and randomly combine



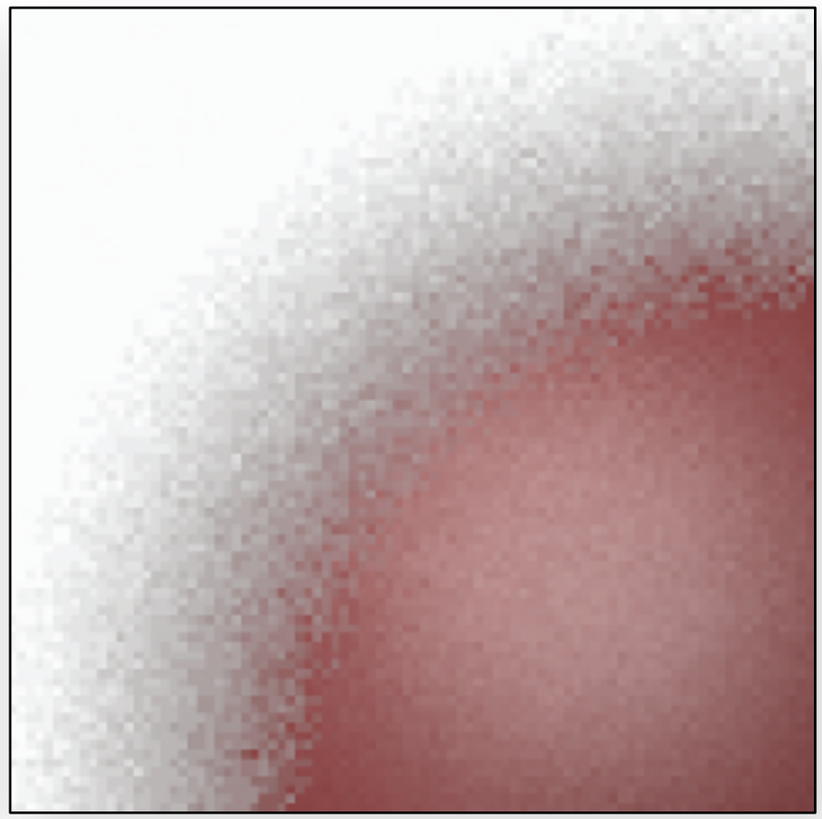
Depth-of-Field (4D)



Reference



Random Sampling

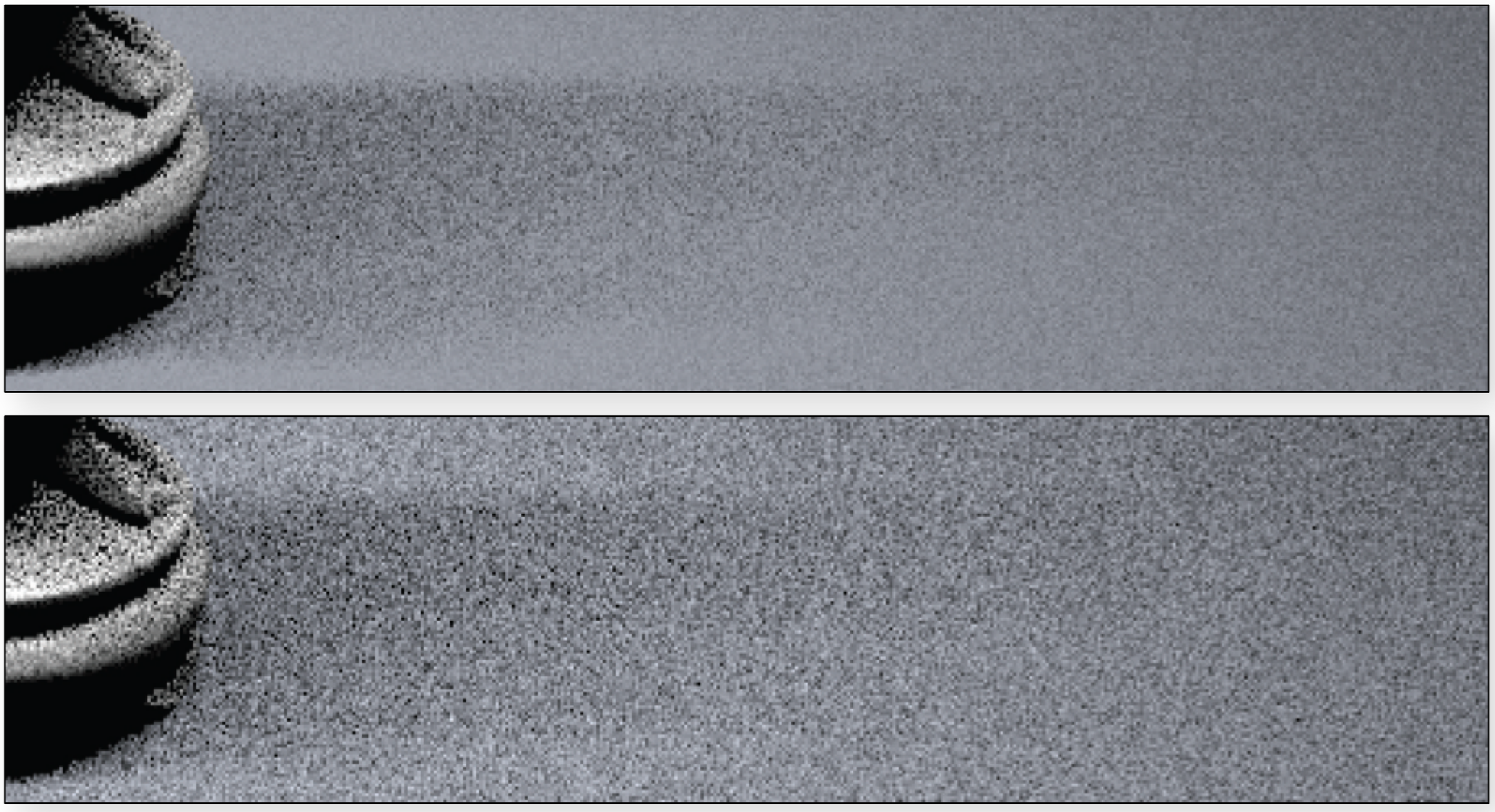


Stratified Sampling

Soft Shadows

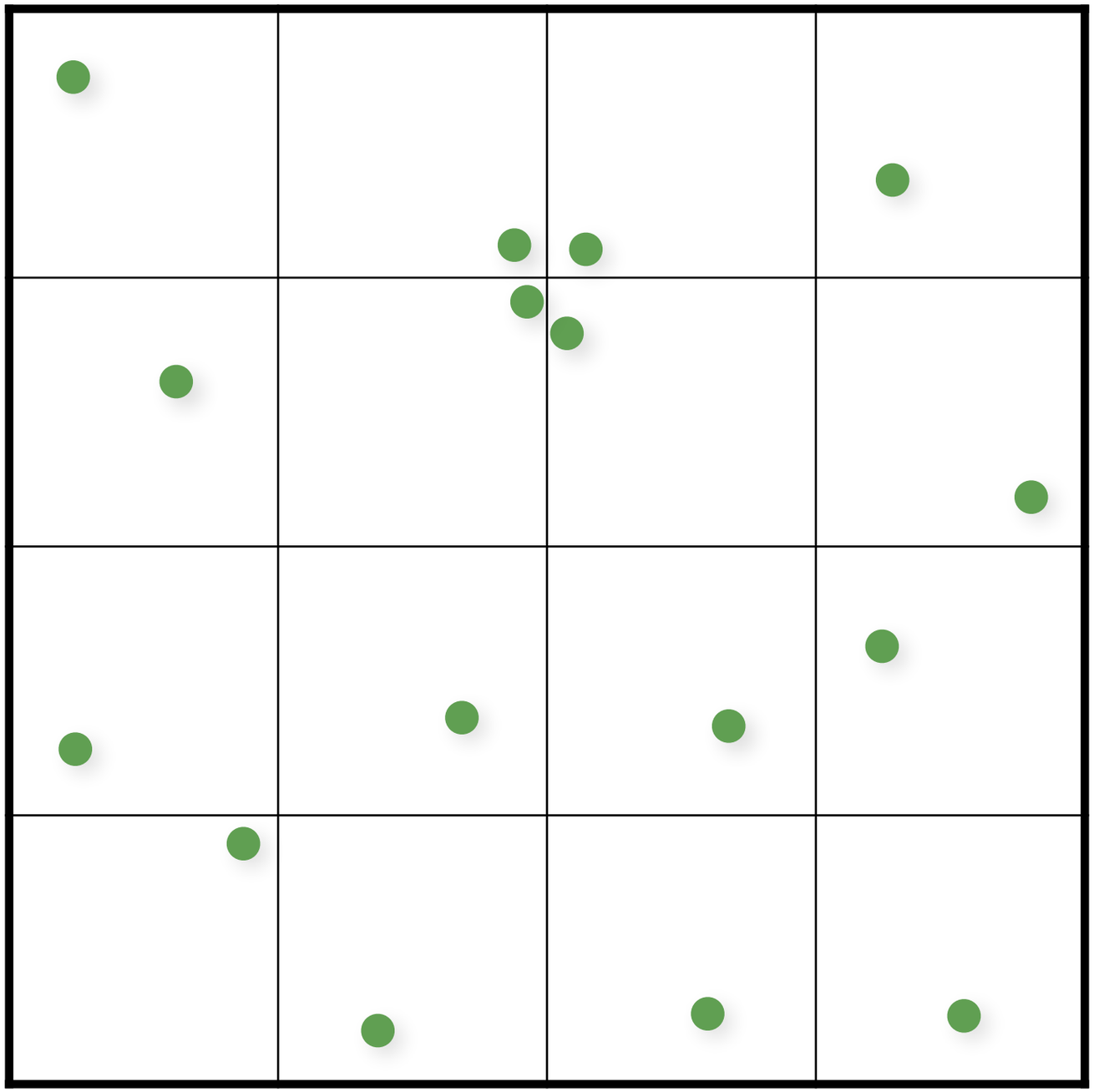
Soft Shadows (Stratified Sampling)

1 image sample x 16 light samples

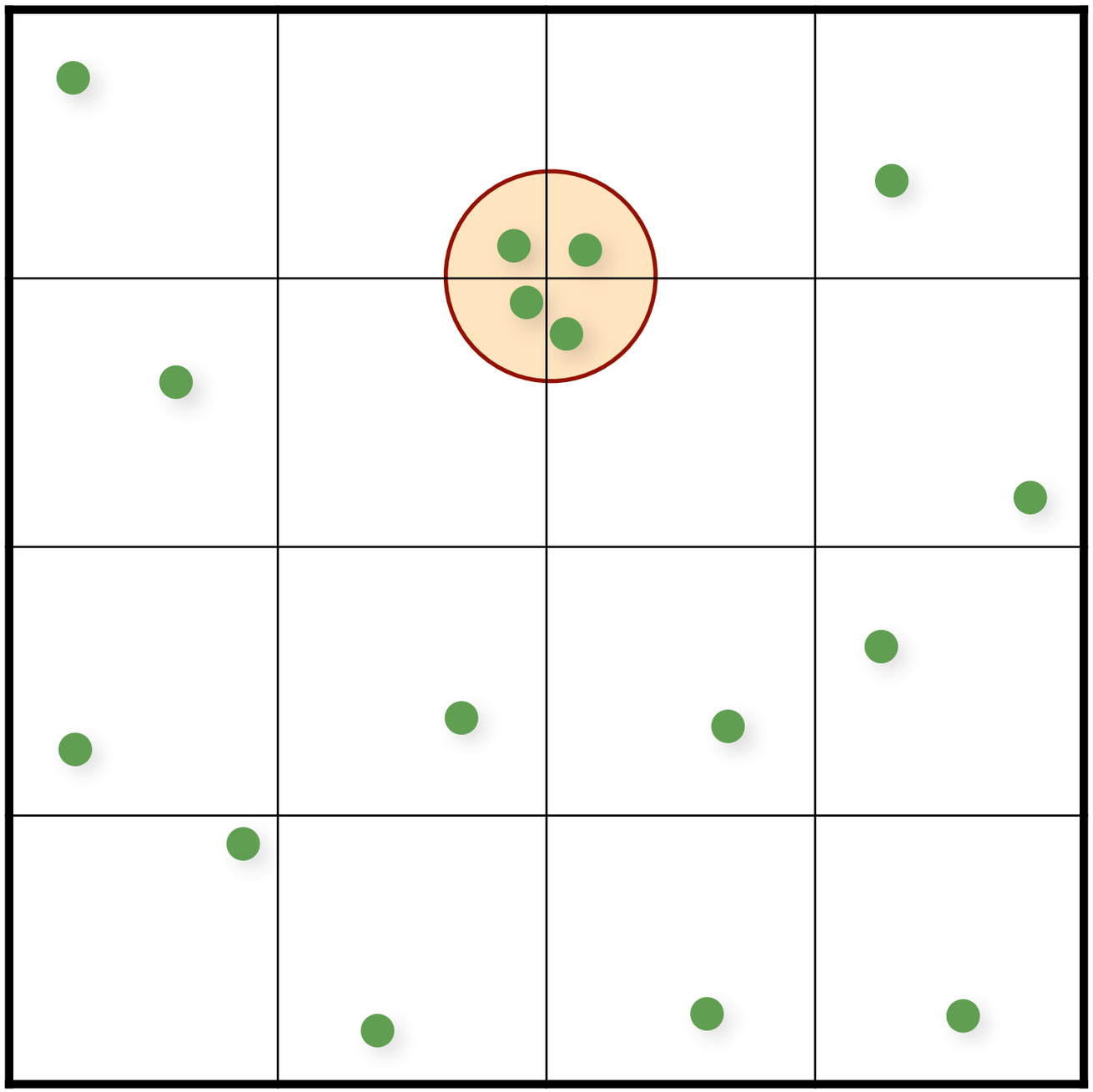


16 image samples x 1 light sample

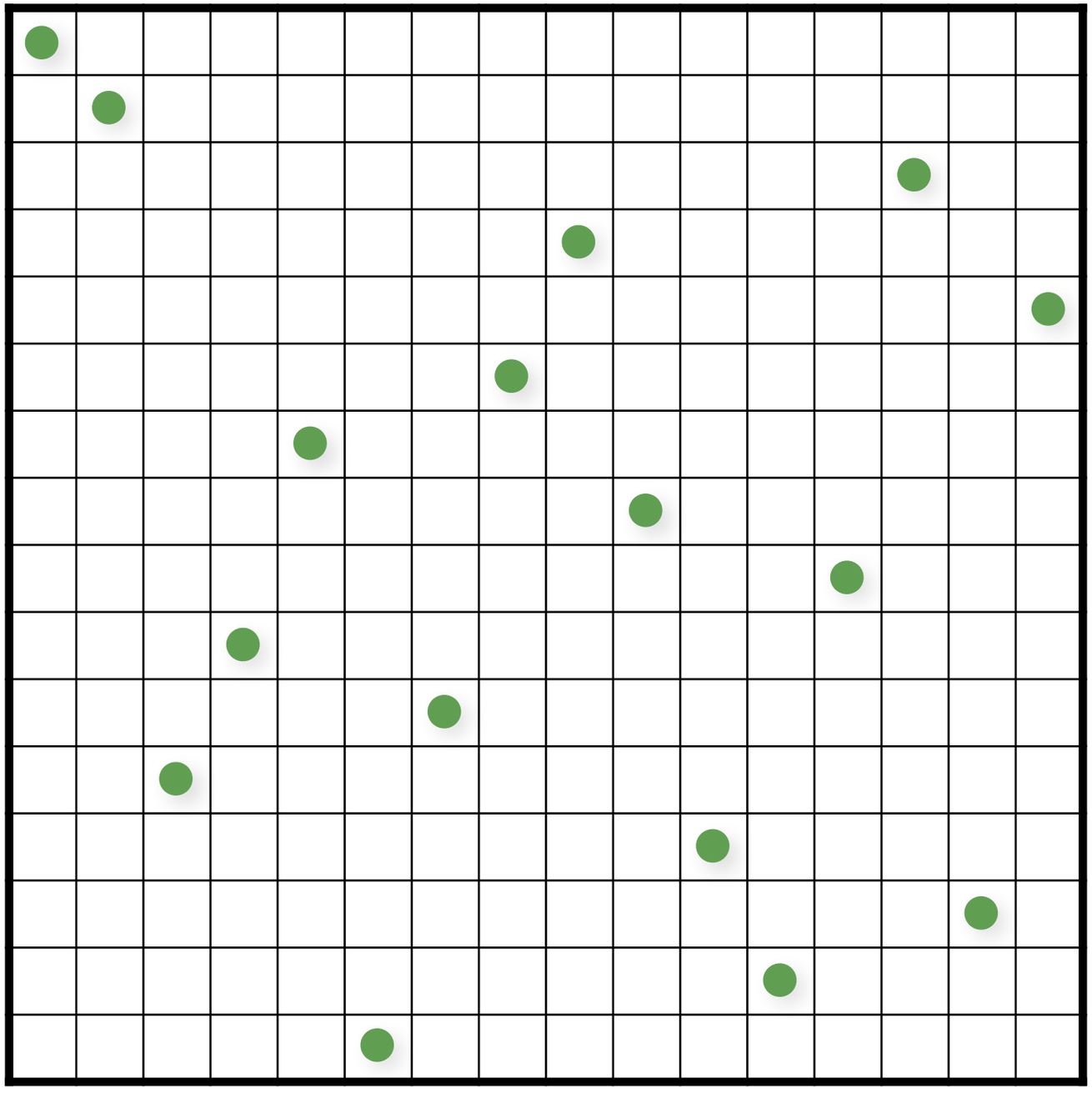
Stratified (Jittered) Sampling



Stratified (Jittered) Sampling



Latin Hypercube (N-Rooks) Sampling

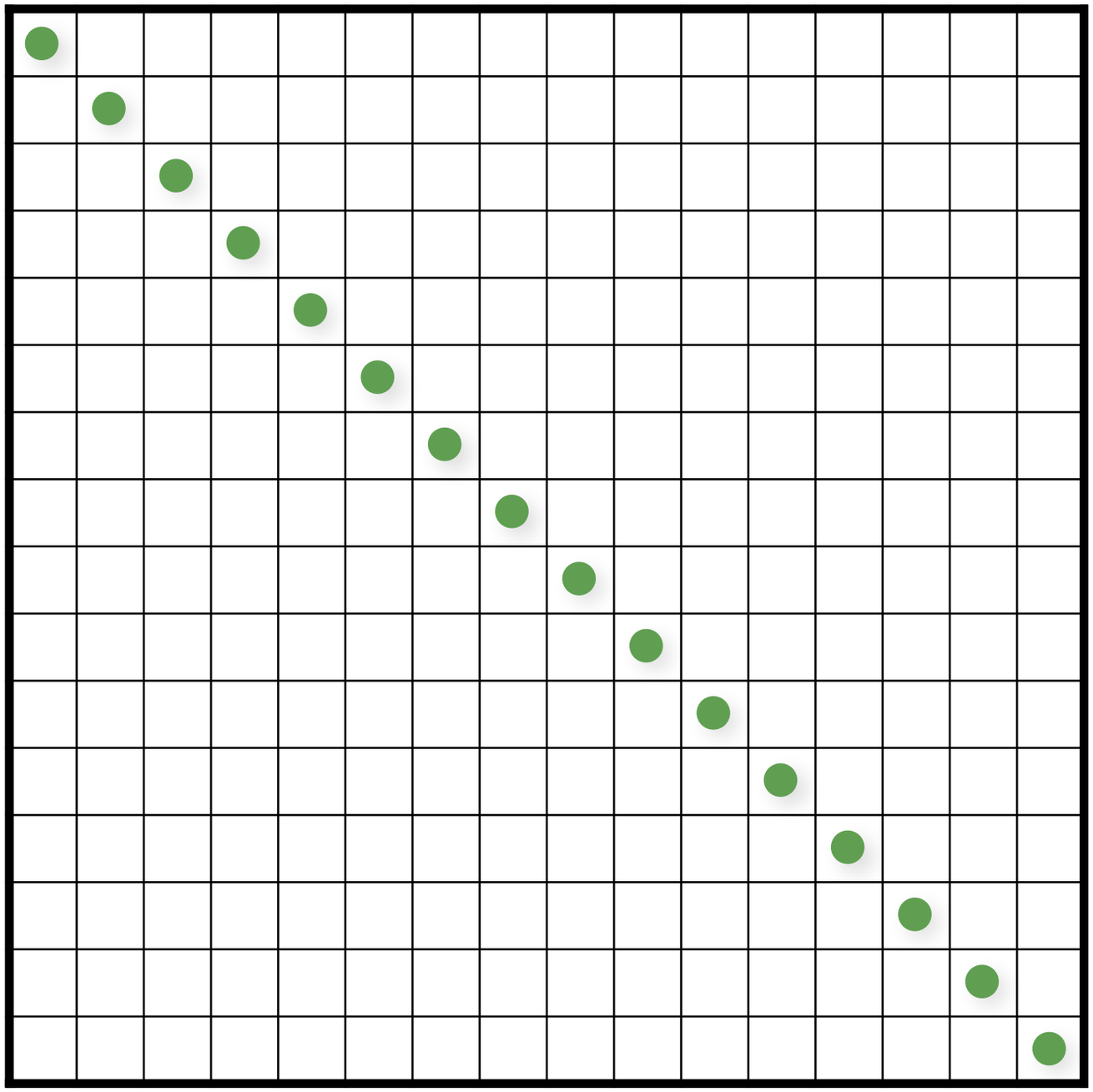


Latin-Hypercube Sampling

```
// initialize the diagonal
for (uint j = 0; j < numDimensions; j++)
   for (uint i = 0; i < numSamples; i++)
      samples[j][i] = (i + randf())/numSamples;
```

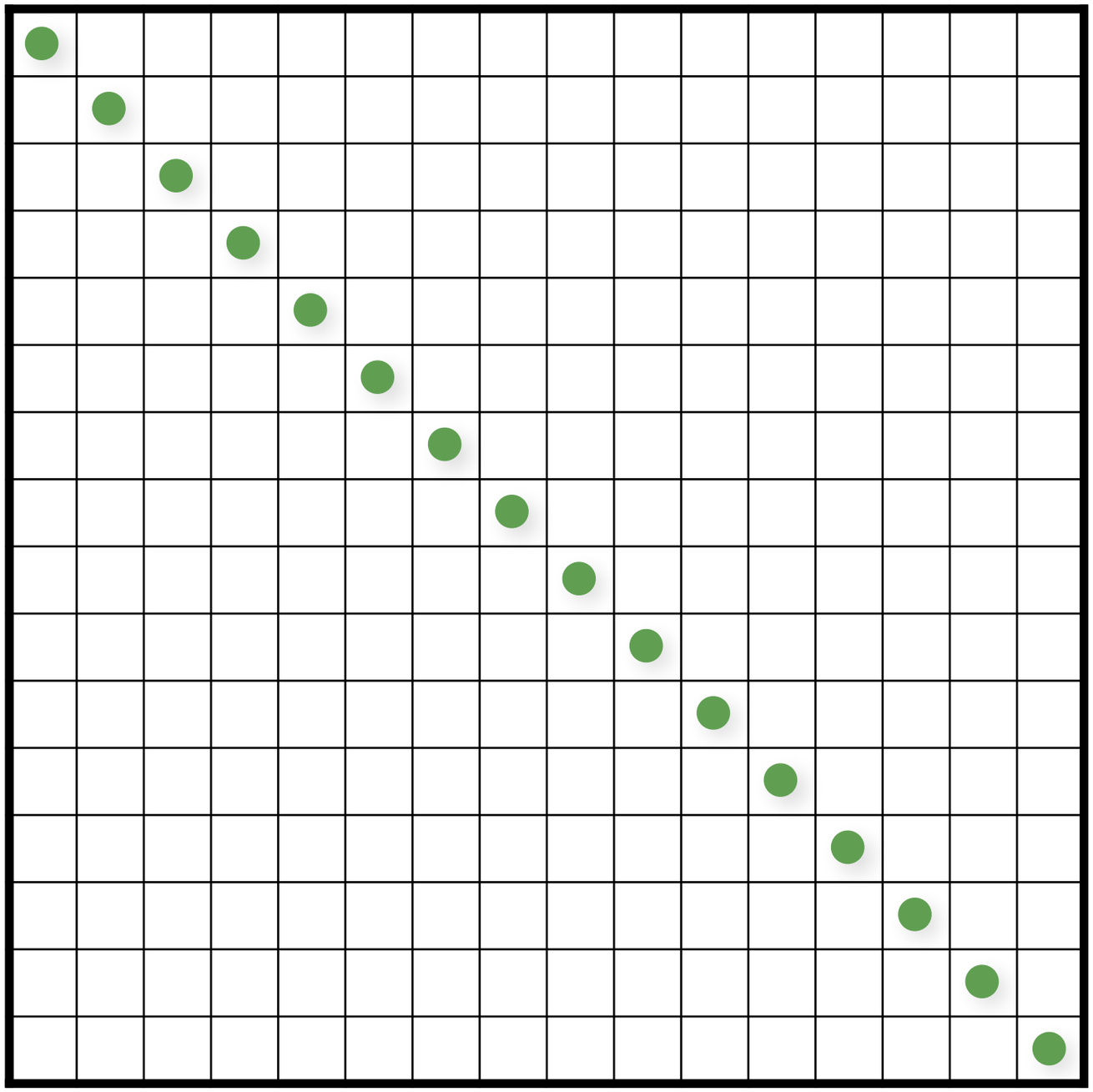
```
// shuffle each dimension independently
for (uint j = 0; j < numDimensions; j++)
   for (uint i = numSamples-1; i >= 1; i--)
      swap(samples[j][i], samples[j][randi(0,i)]);
```

Latin Hypercube (N-Rooks) Sampling



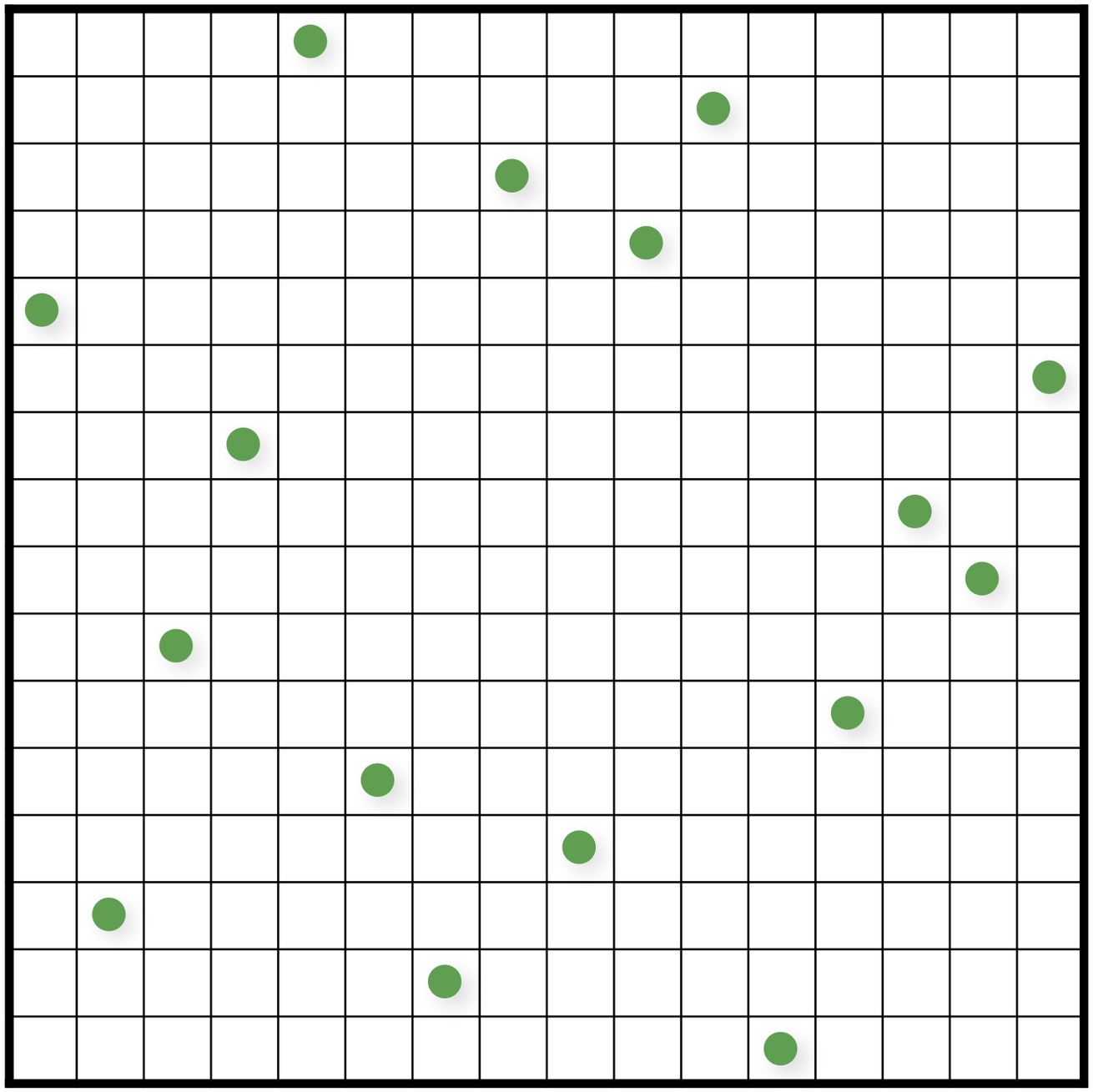
Initialize

Latin Hypercube (N-Rooks) Sampling



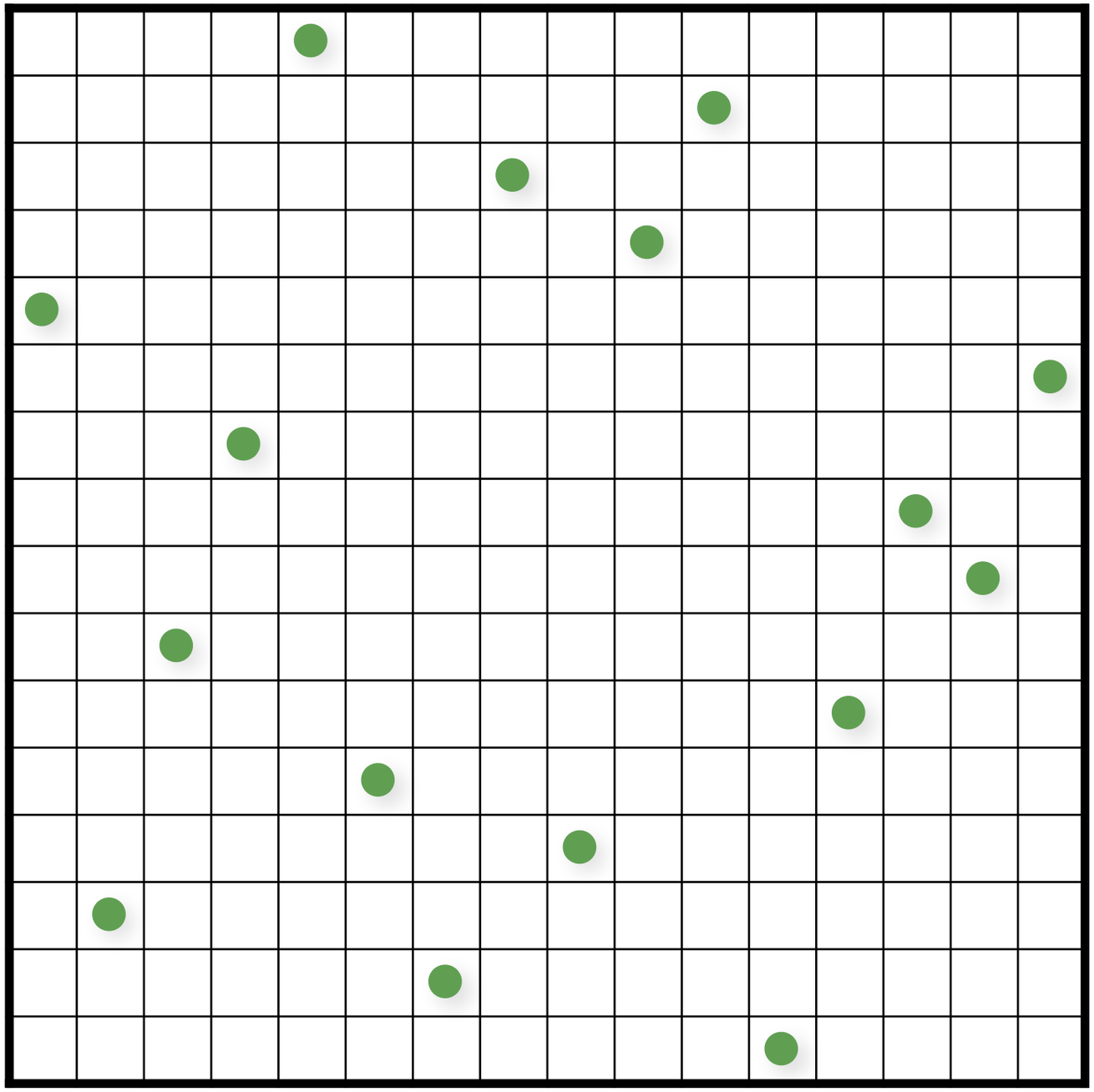
Shuffle rows

Latin Hypercube (N-Rooks) Sampling

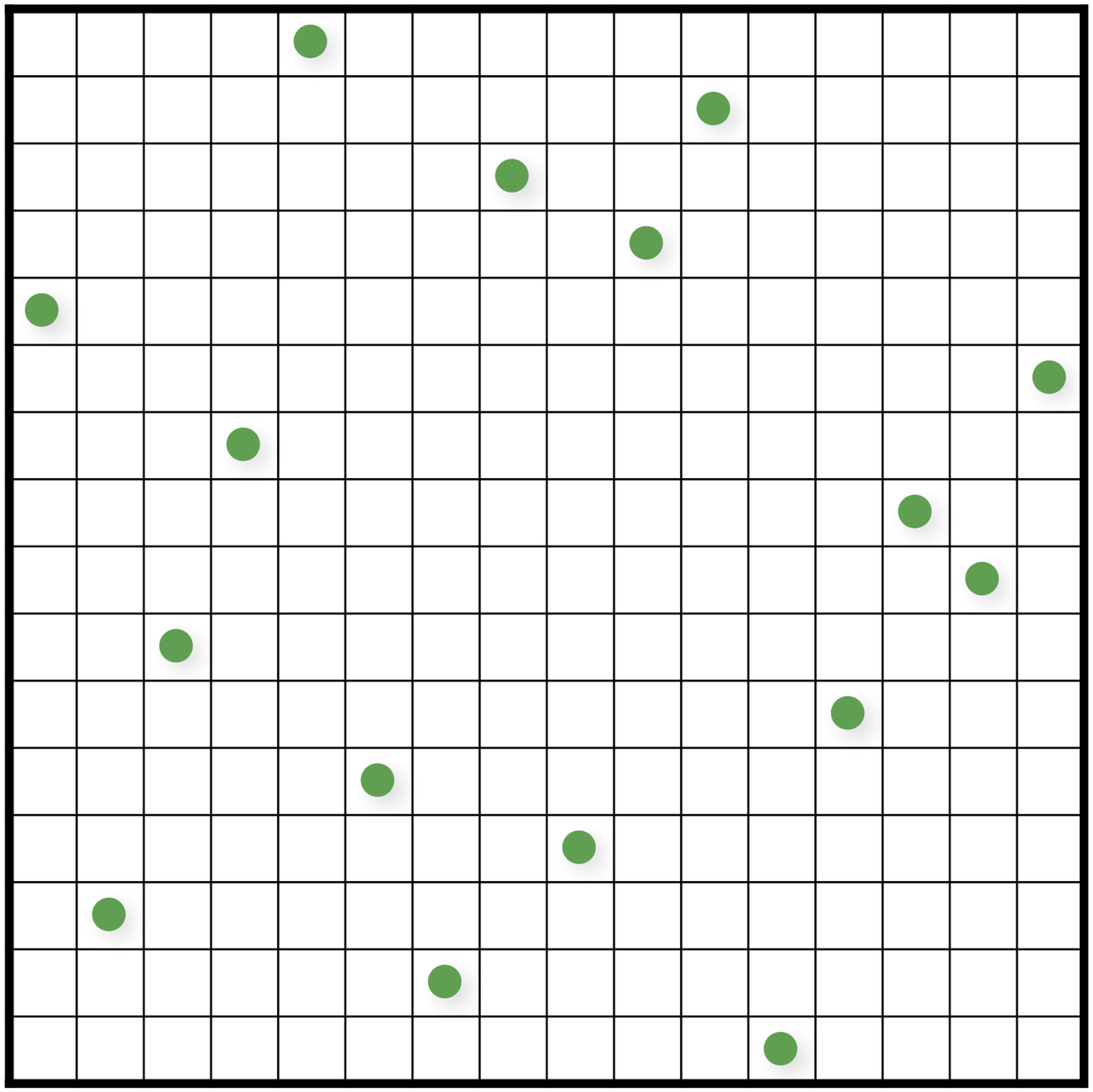


Shuffle rows

Latin Hypercube (N-Rooks) Sampling

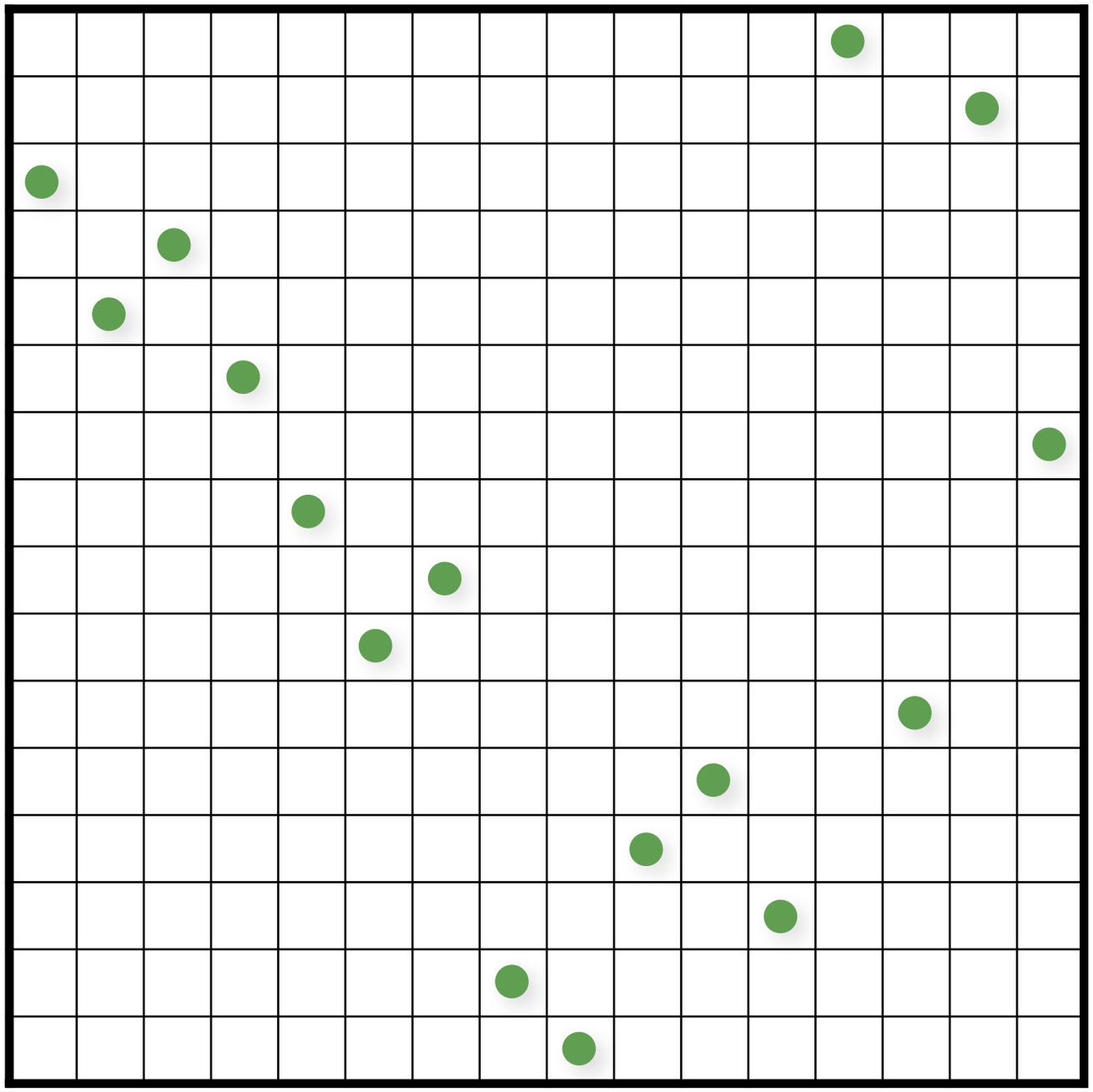


Latin Hypercube (N-Rooks) Sampling



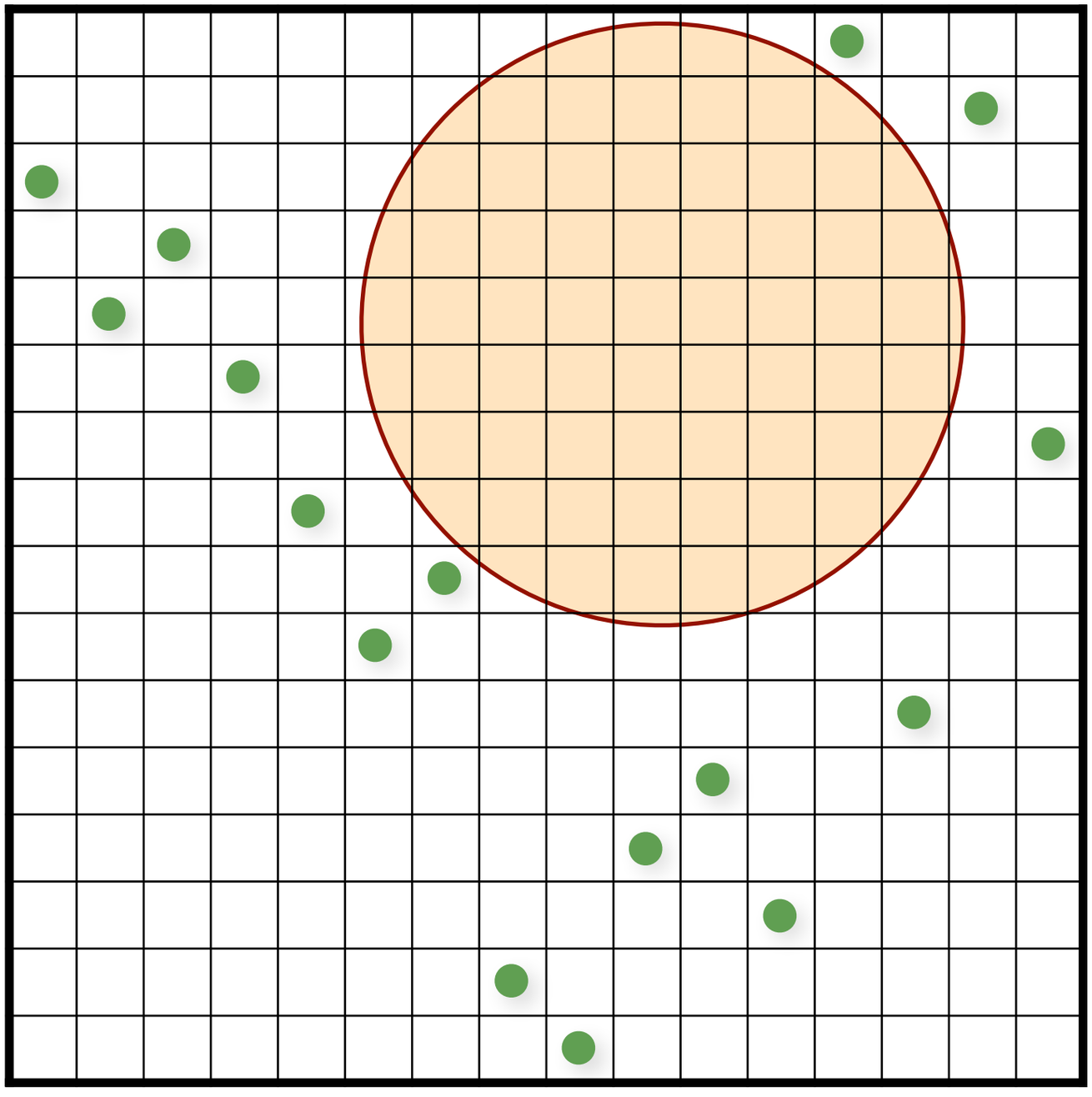
Shuffle columns

Latin Hypercube (N-Rooks) Sampling



Shuffle columns

Latin Hypercube (N-Rooks) Sampling

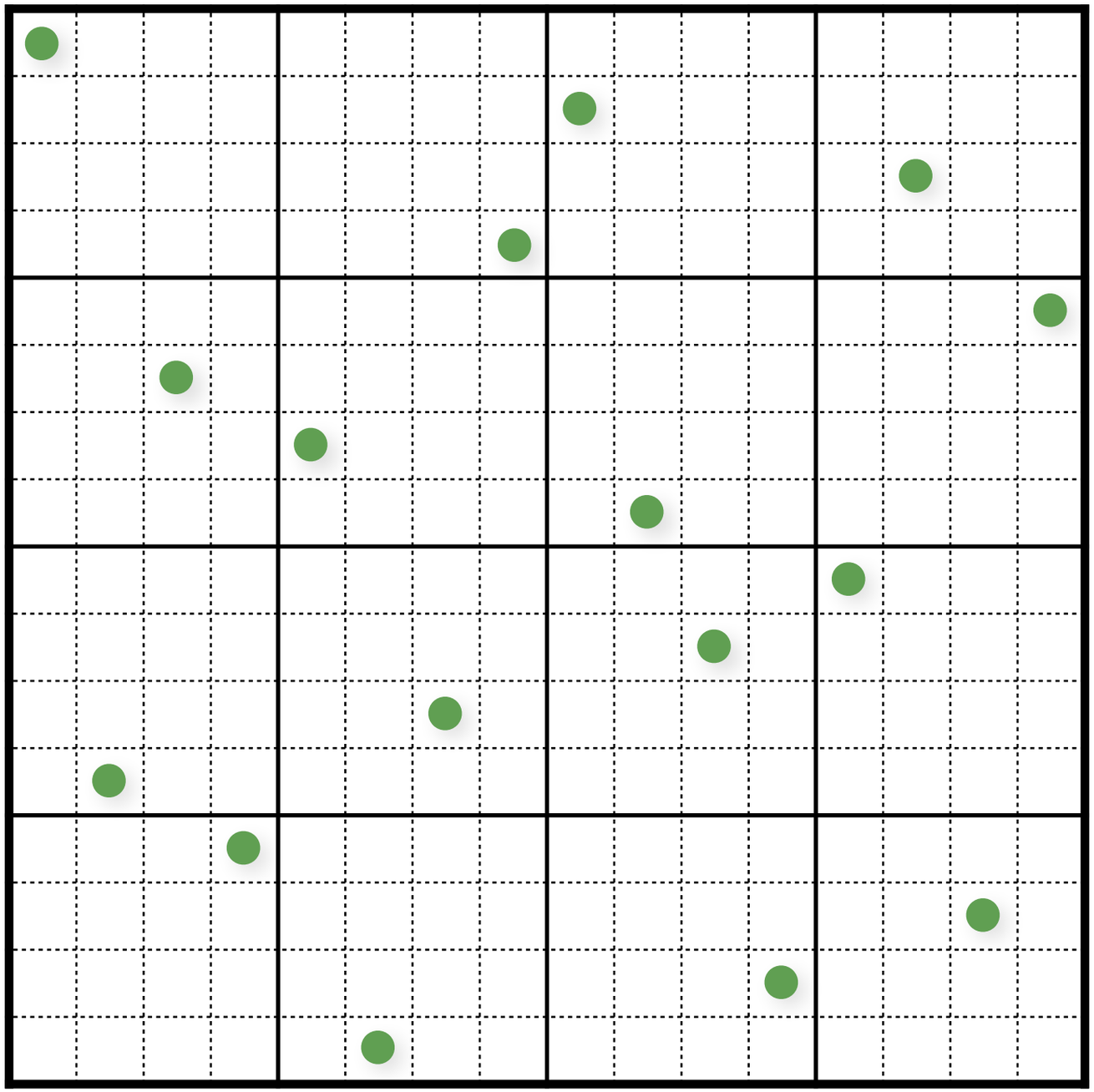


Shuffle columns

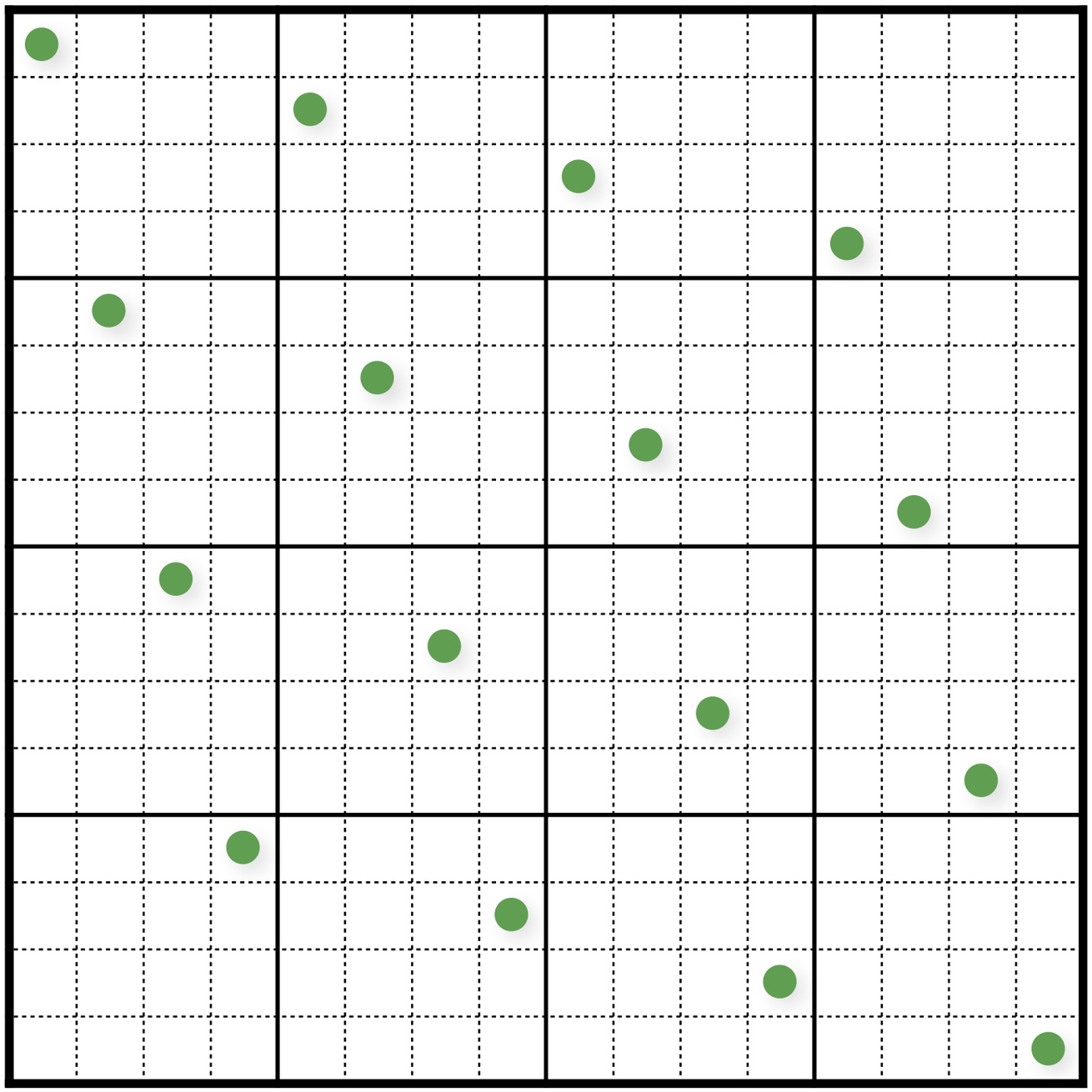
Multi-Jittered Sampling

- Kenneth Chiu, Peter Shirley, and Changyaw Wang. “Multi-jittered sampling.” In Graphics Gems IV, pp. 370–374. Academic Press, May 1994.
  - combine N-Rooks and Jittered stratification constraints

Multi-Jittered Sampling

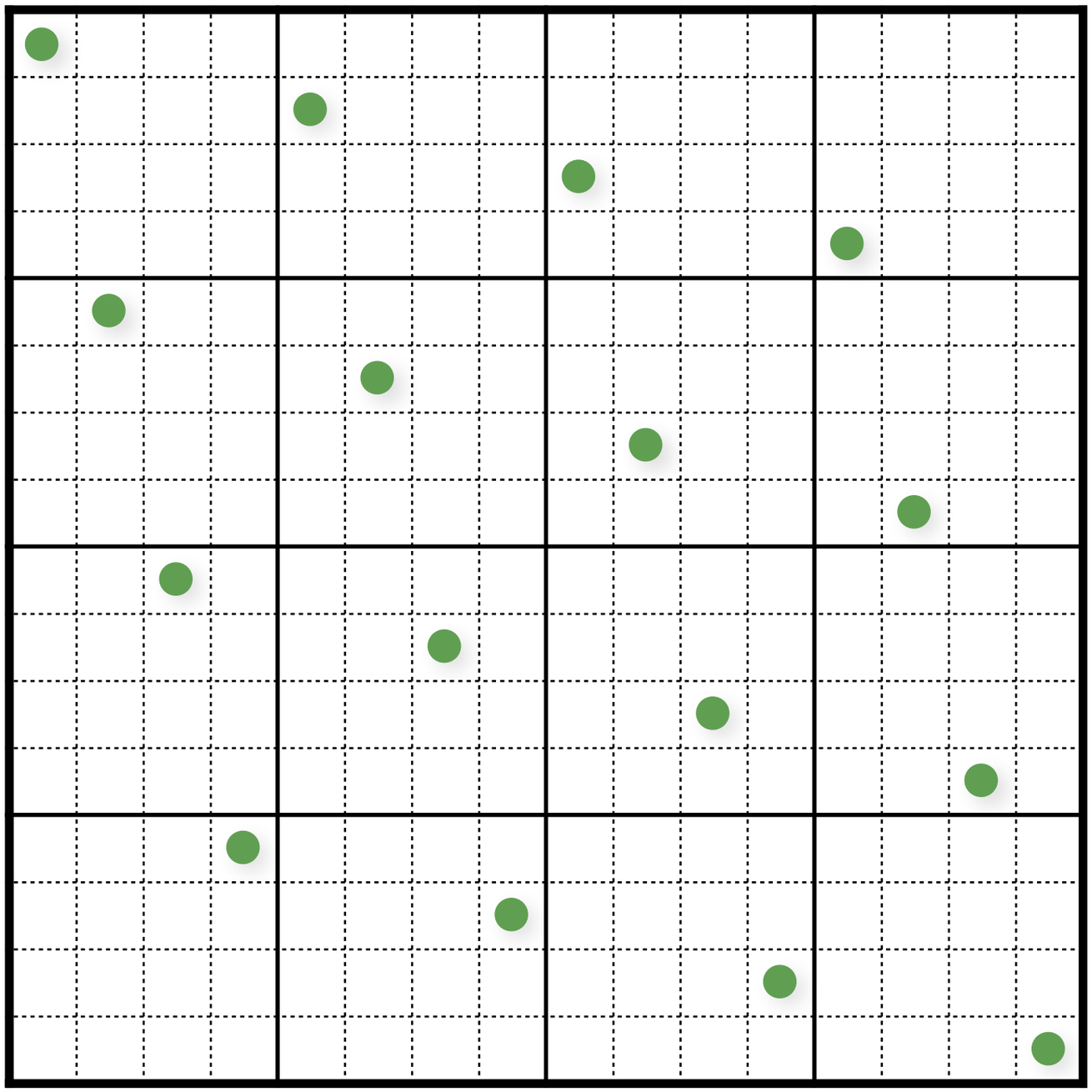


Multi-Jittered Sampling



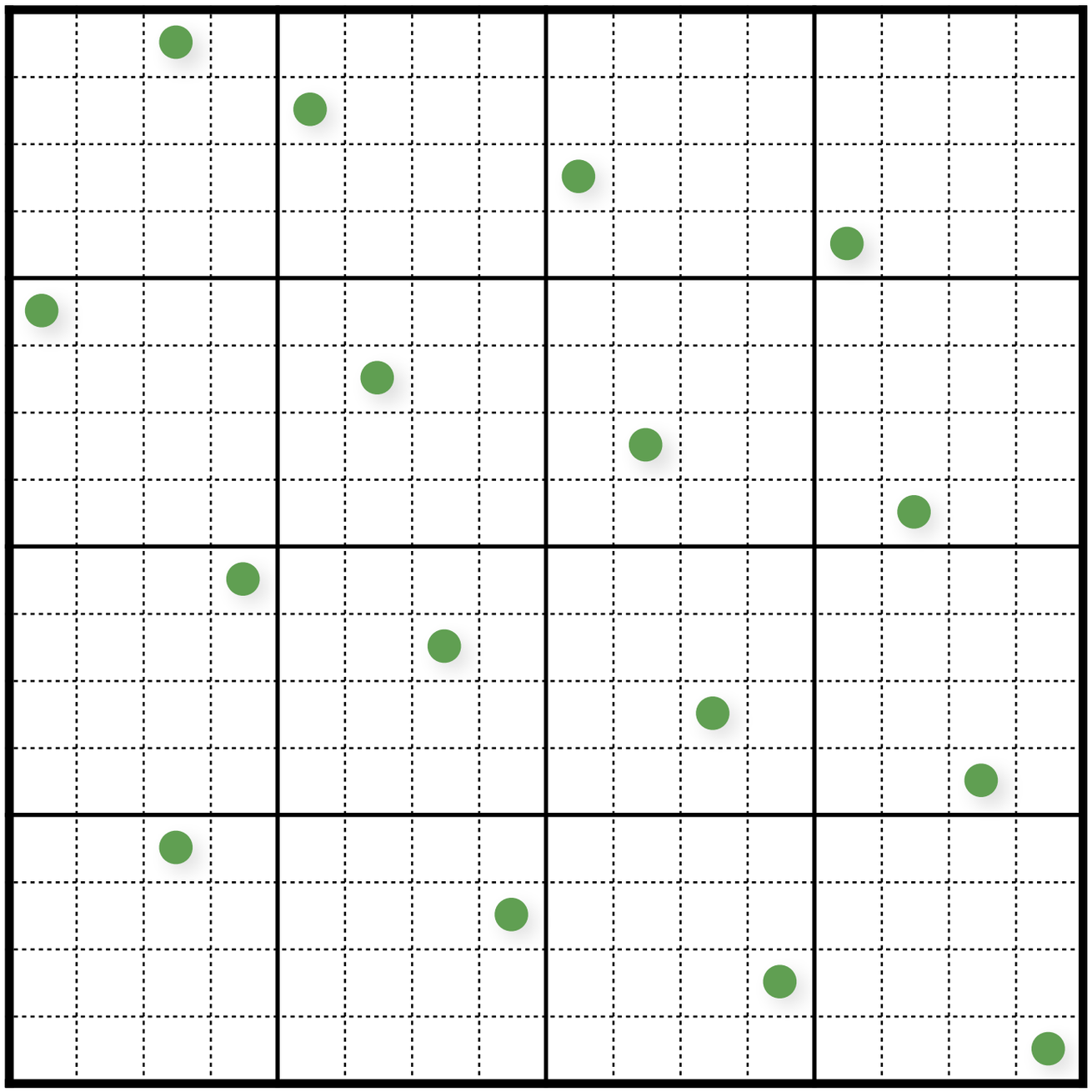
Initialize

Multi-Jittered Sampling



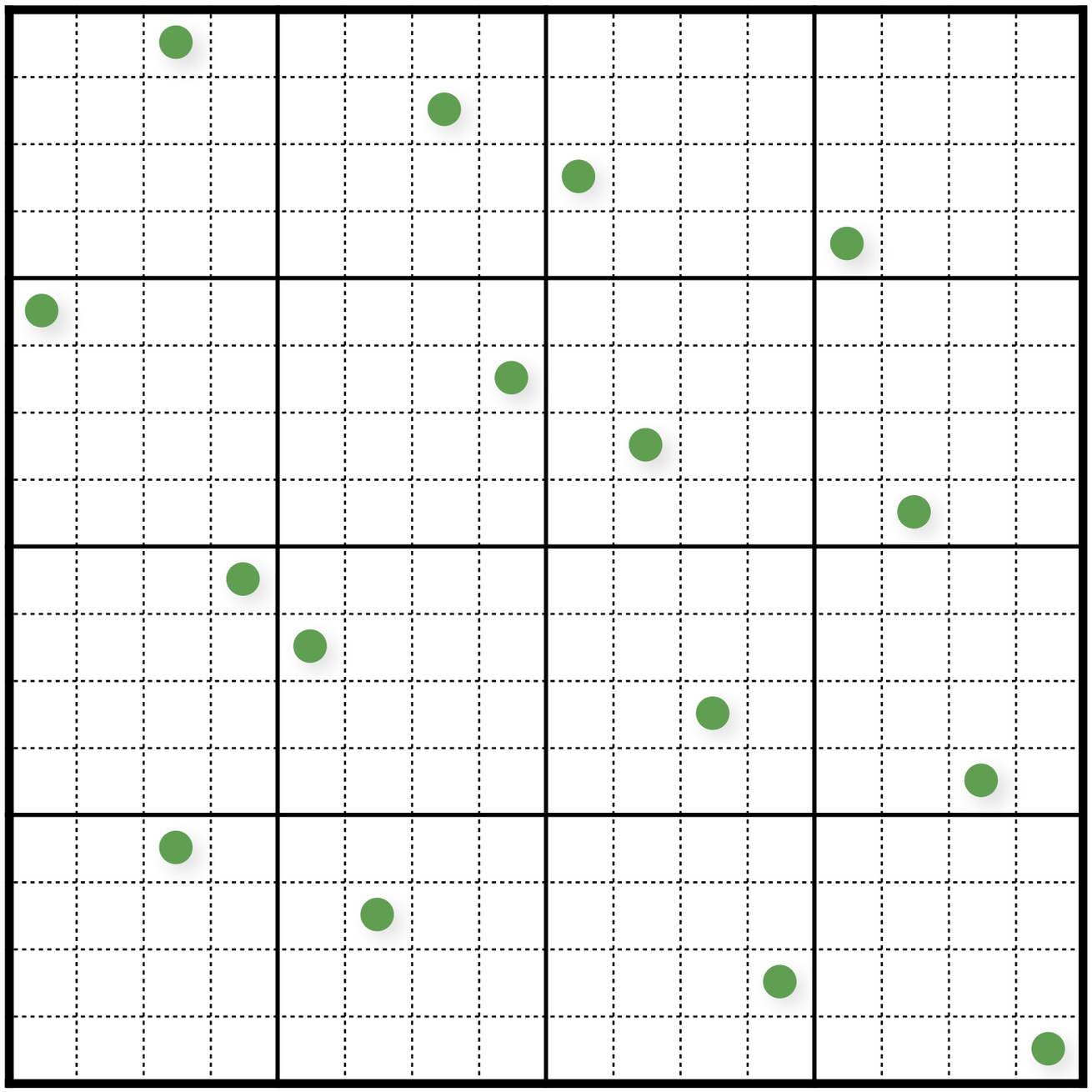
Shuffle x-coords

Multi-Jittered Sampling



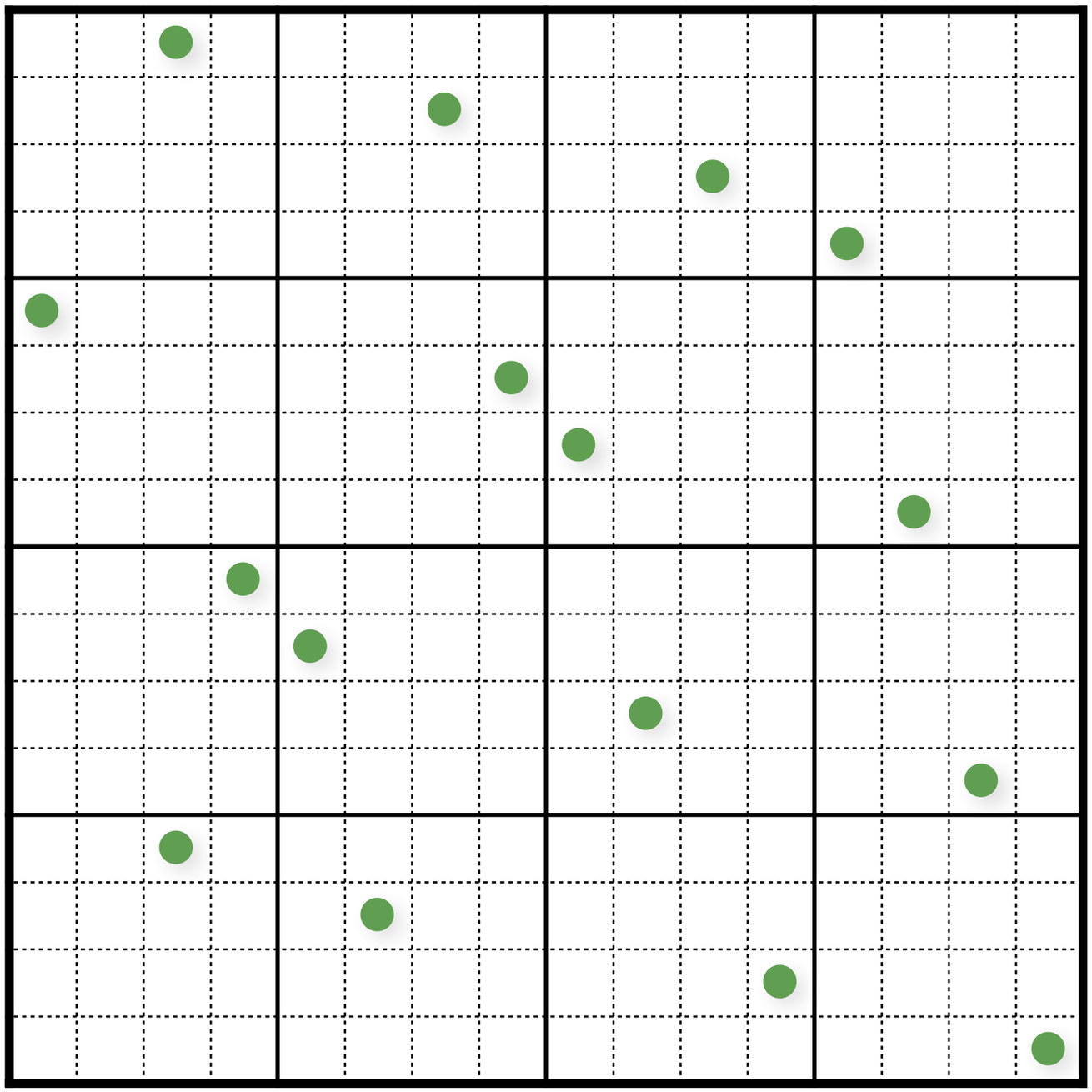
Shuffle x-coords

Multi-Jittered Sampling



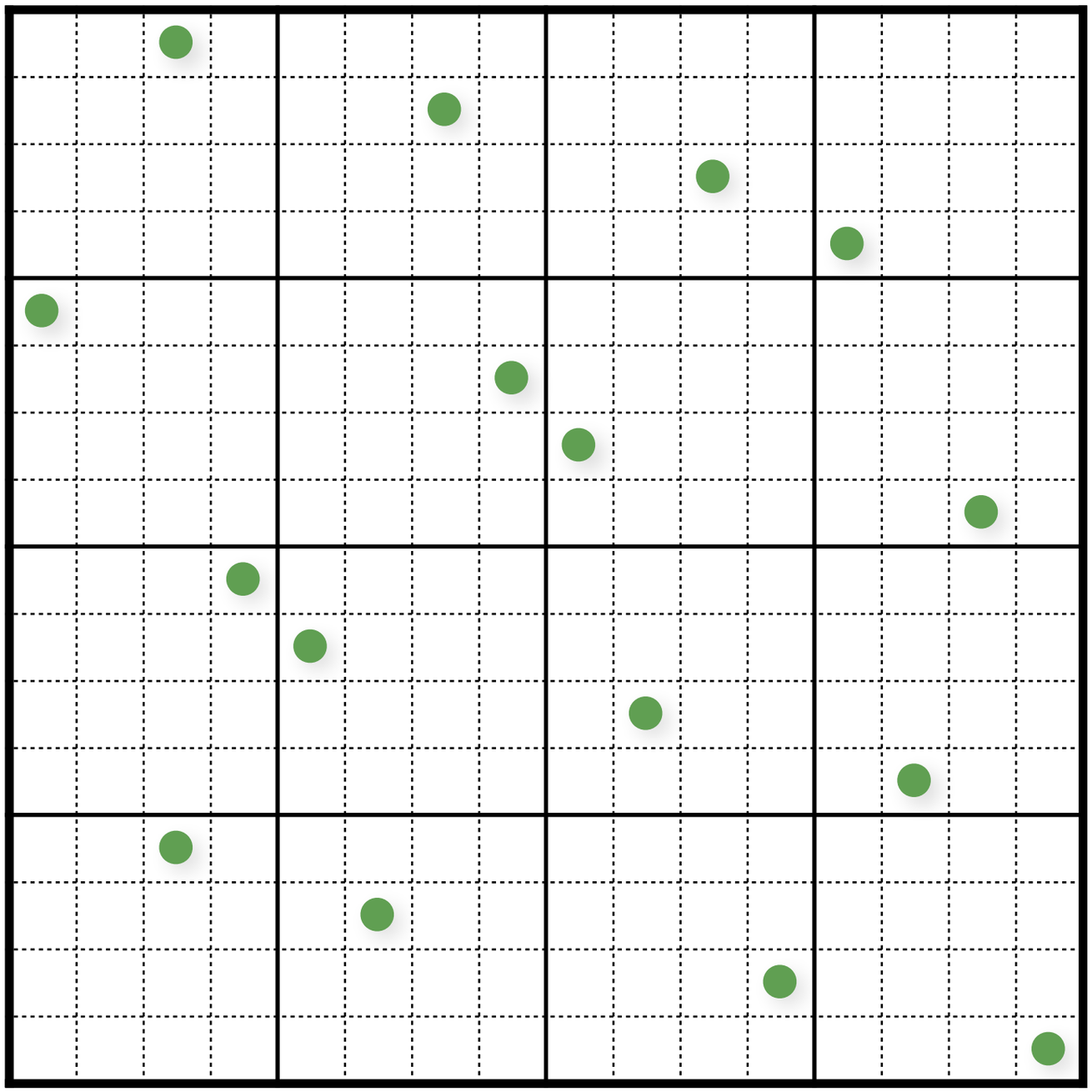
Shuffle x-coords

Multi-Jittered Sampling



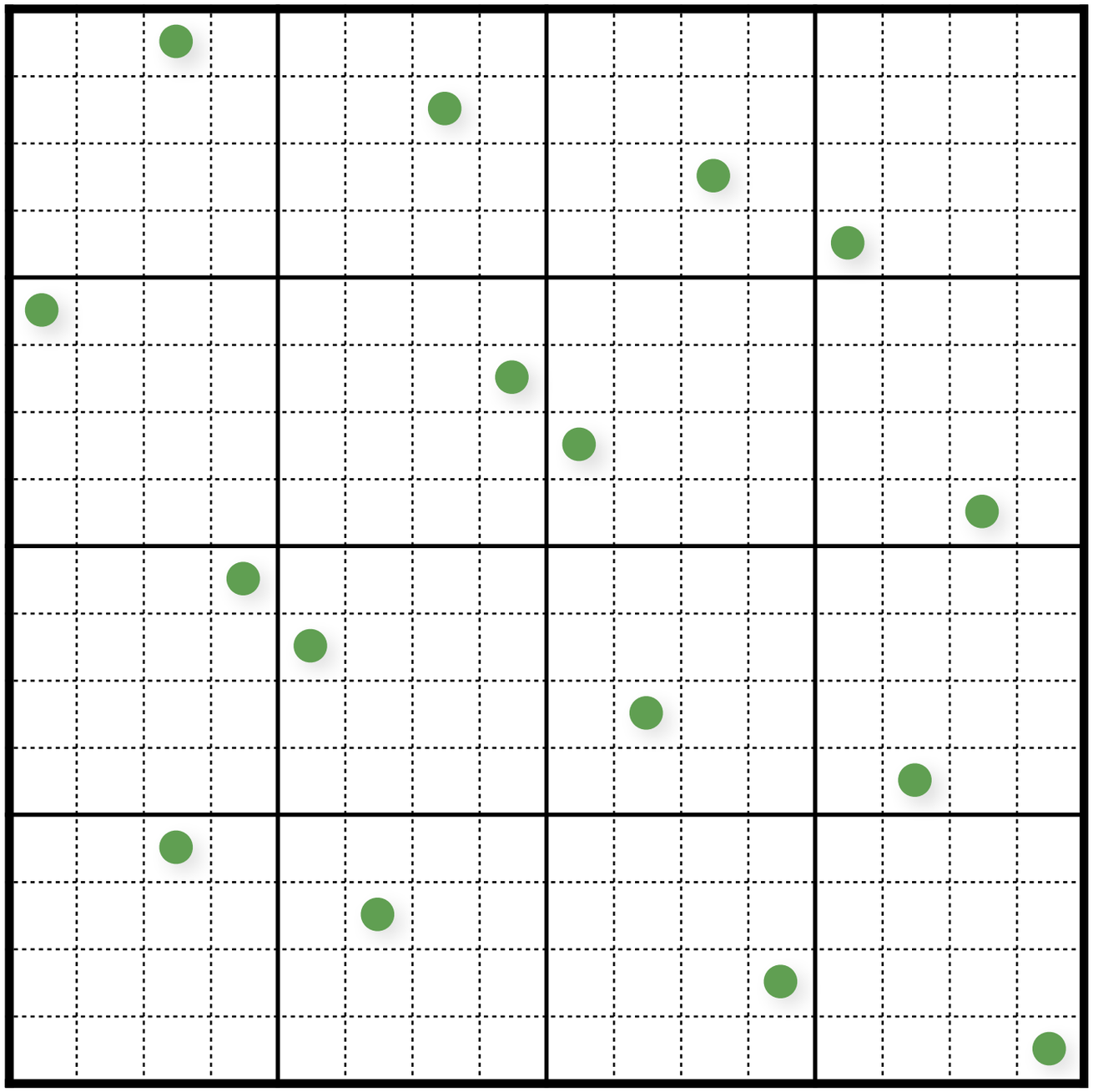
Shuffle x-coords

Multi-Jittered Sampling

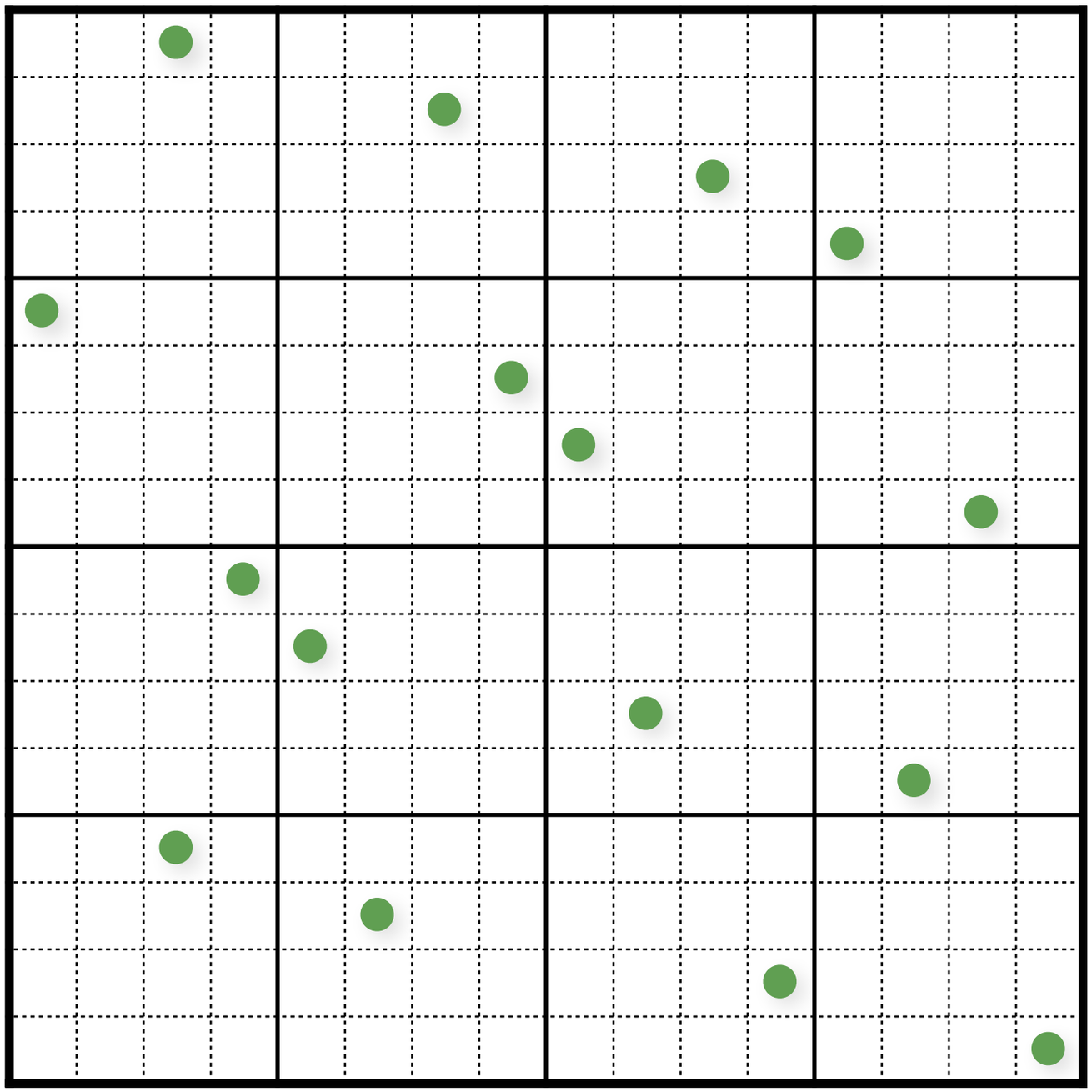


Shuffle x-coords

Multi-Jittered Sampling

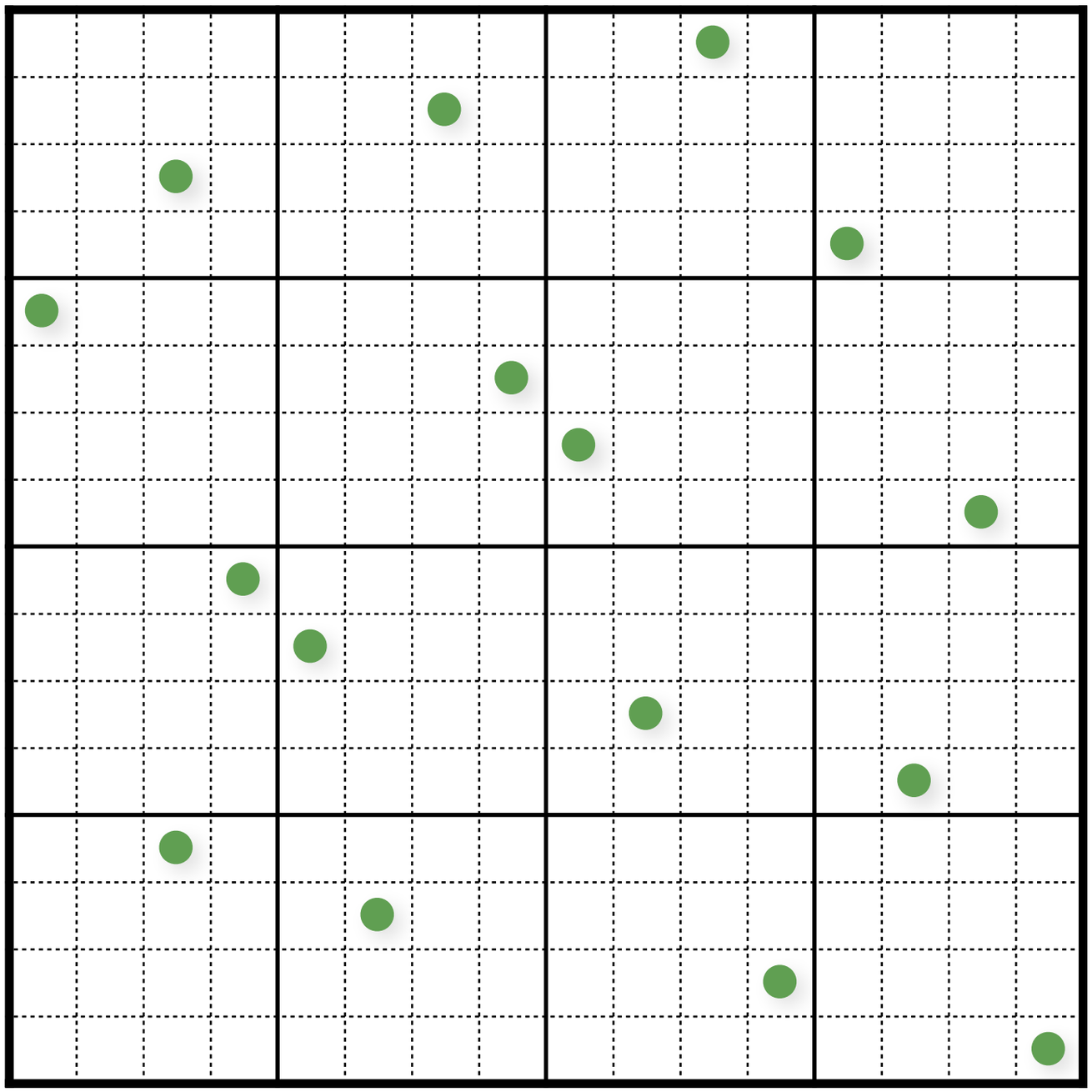


Multi-Jittered Sampling



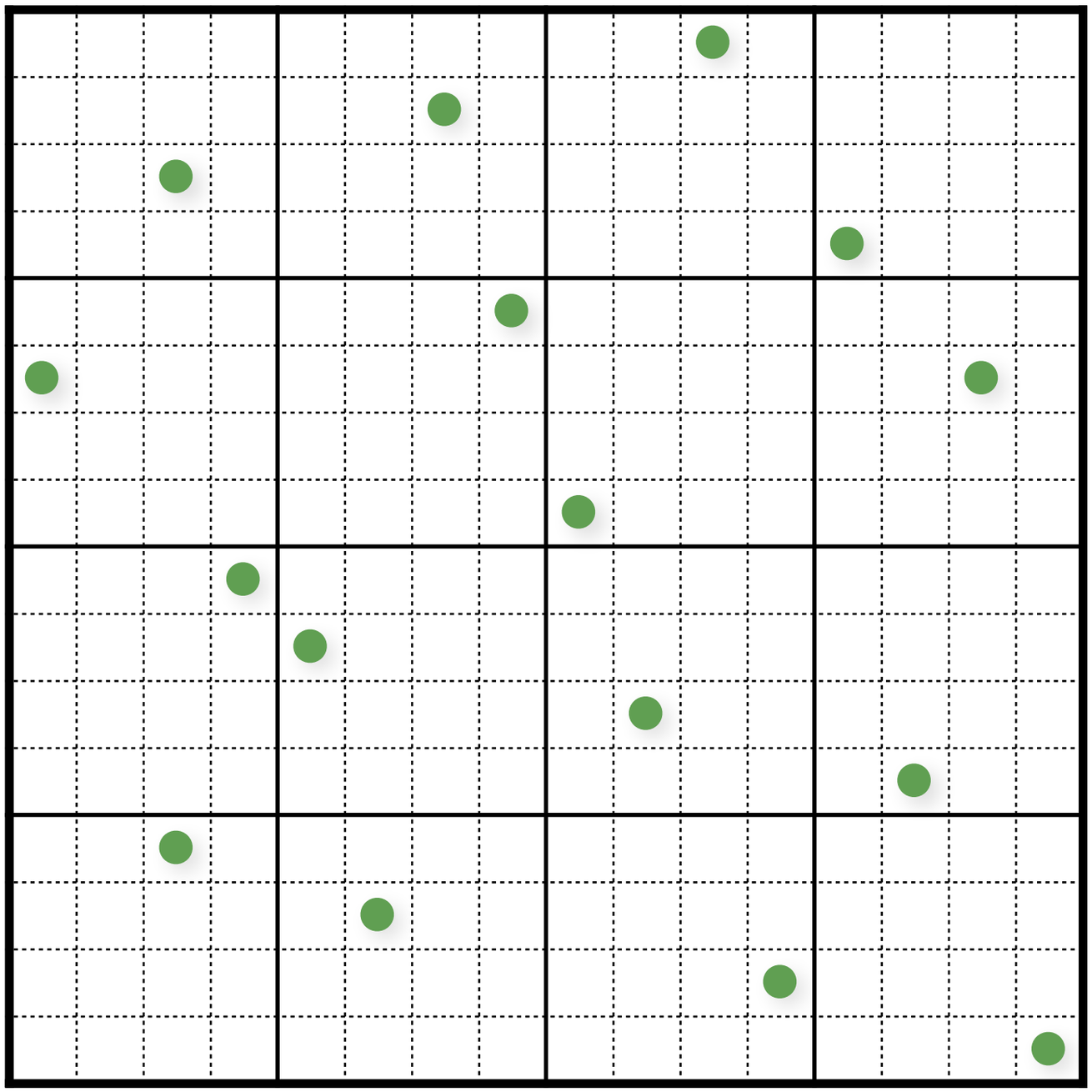
Shuffle y-coords

Multi-Jittered Sampling



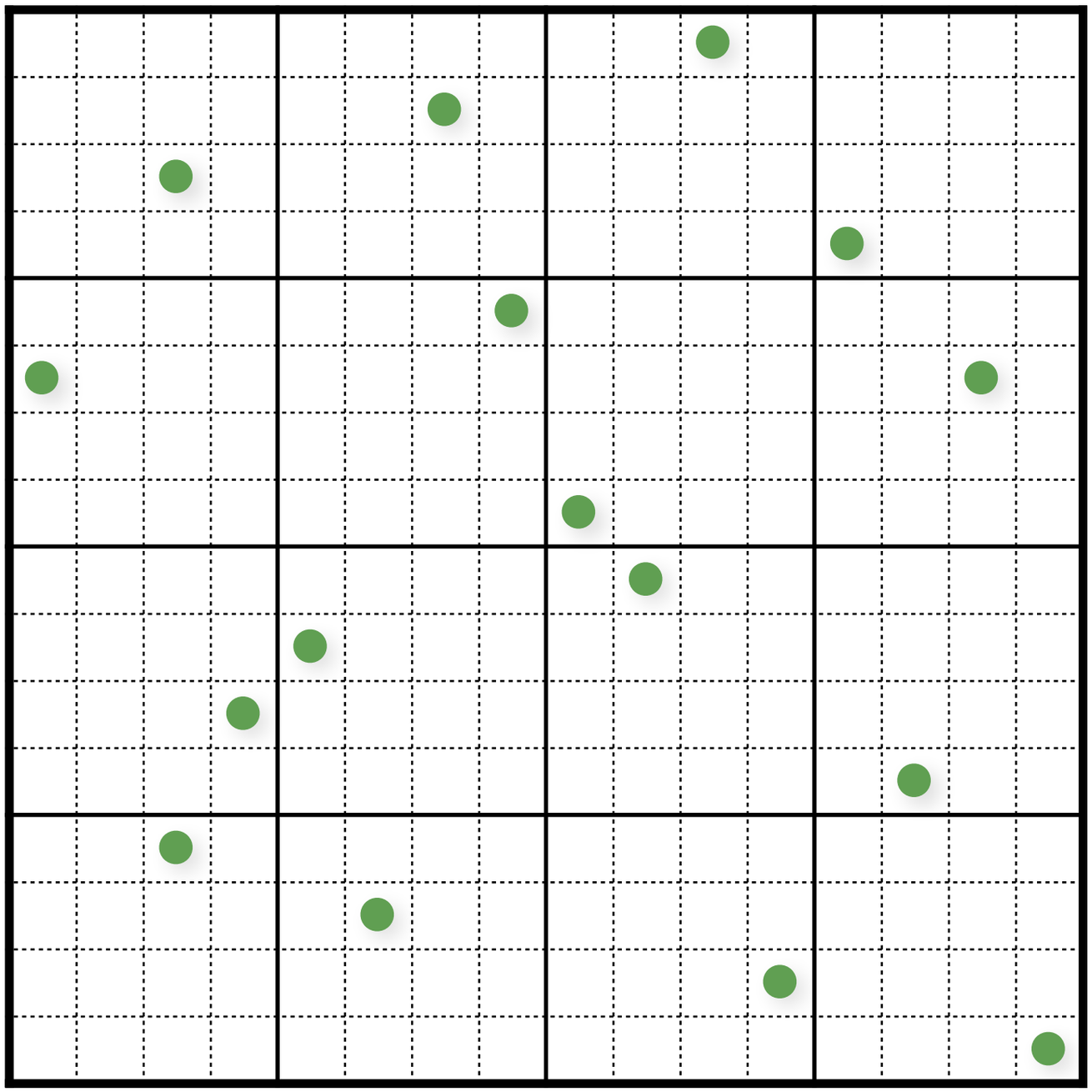
Shuffle y-coords

Multi-Jittered Sampling



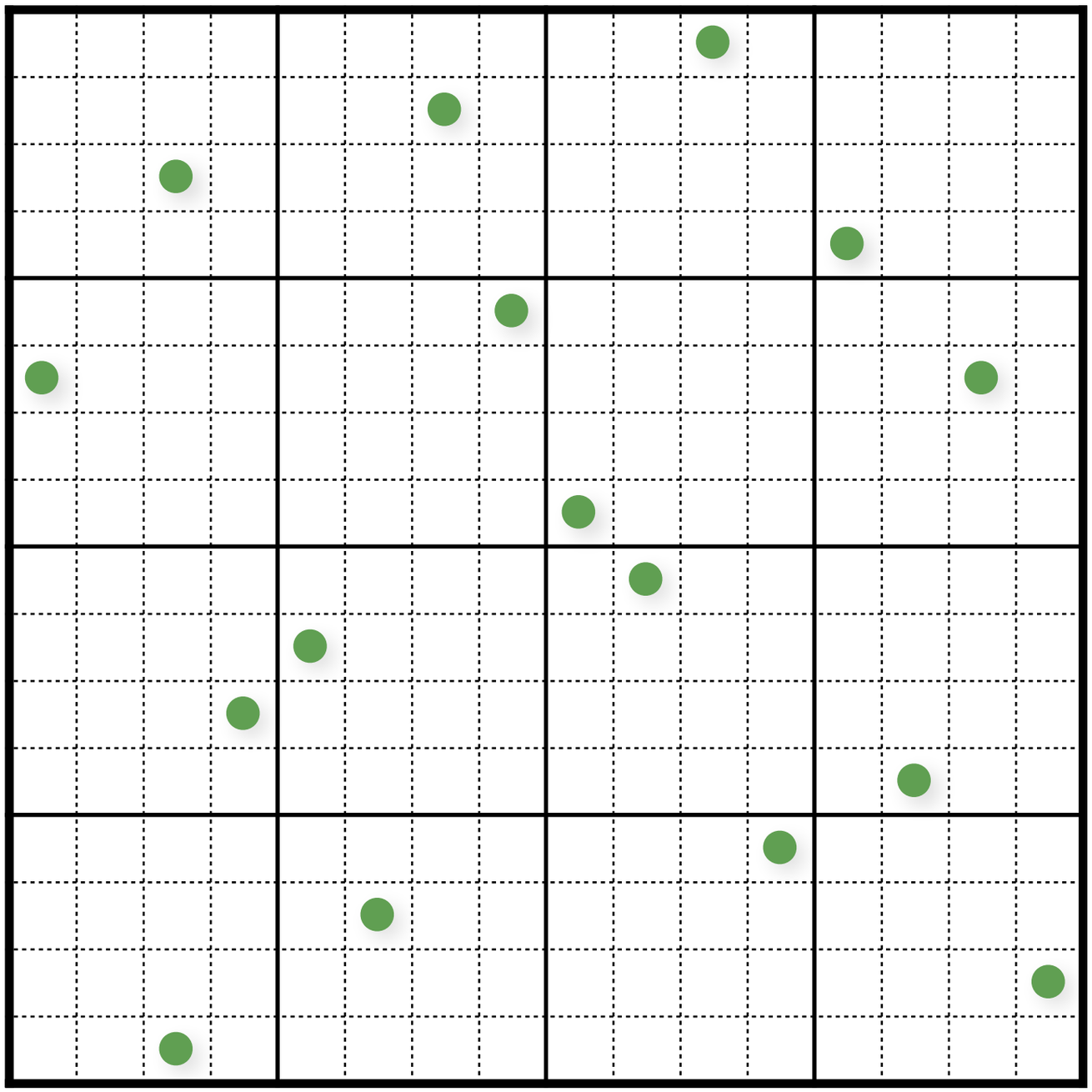
Shuffle y-coords

Multi-Jittered Sampling



Shuffle y-coords

Multi-Jittered Sampling



Shuffle y-coords

Multi-Jittered Sampling

```
// initialize
float cellSize = 1.0 / (resX*resY);
for (uint i = 0; i < resX; i++)
   for (uint j = 0; j < resY; j++)
   {
      samples(i,j).x = i/resX + (j+randf()) / (resX*resY);
      samples(i,j).y = j/resY + (i+randf()) / (resX*resY);
   }
```

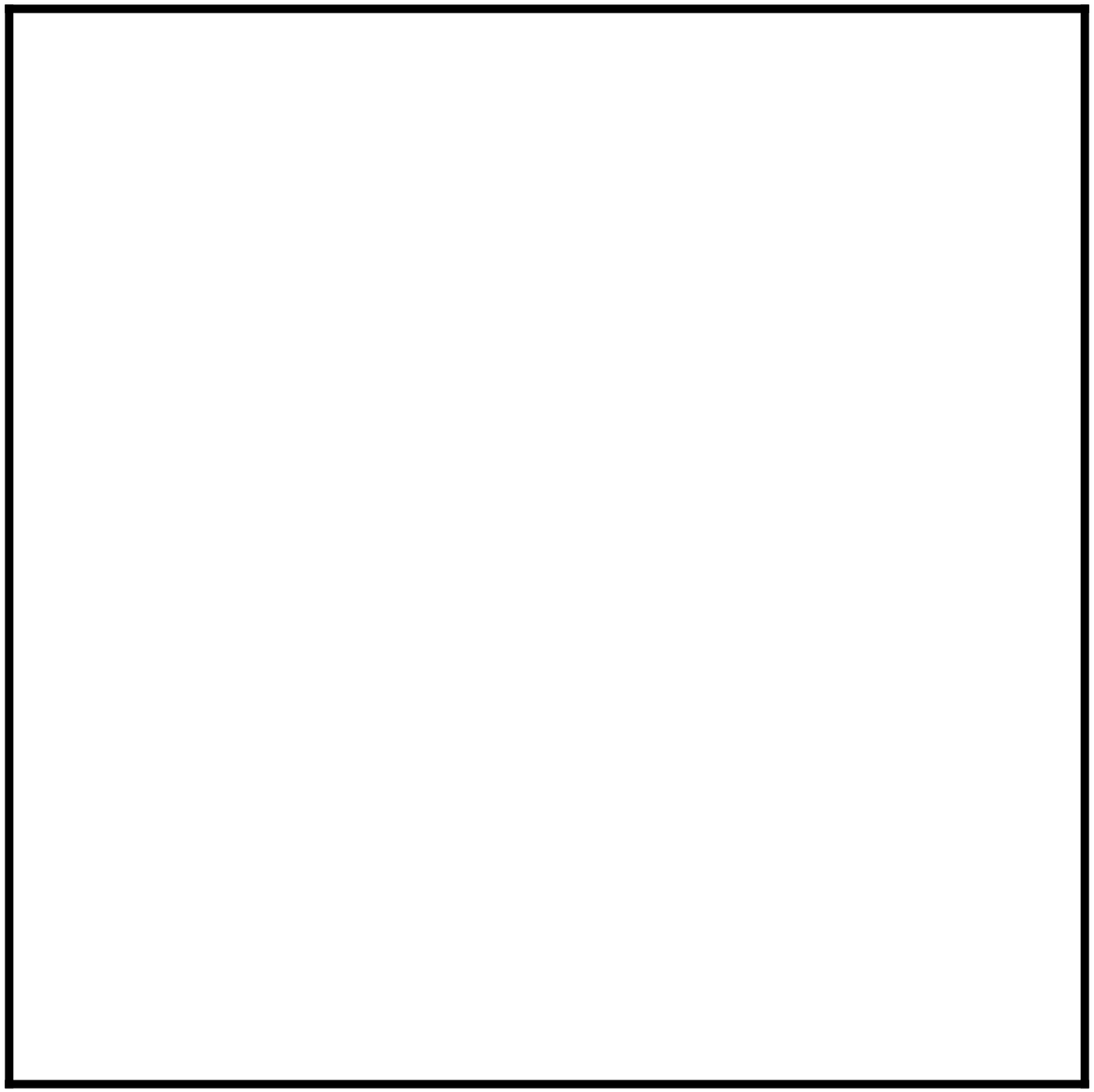
```
// shuffle x coordinates within each column of cells
for (uint i = 0; i < resX; i++)
   for (uint j = resY-1; j >= 1; j--)
      swap(samples(i, j).x, samples(i, randi(0, j)).x);

// shuffle y coordinates within each row of cells
for (unsigned j = 0; j < resY; j++)
   for (unsigned i = resX-1; i >= 1; i--)
      swap(samples(i, j).y, samples(randi(0, i), j).y);
```

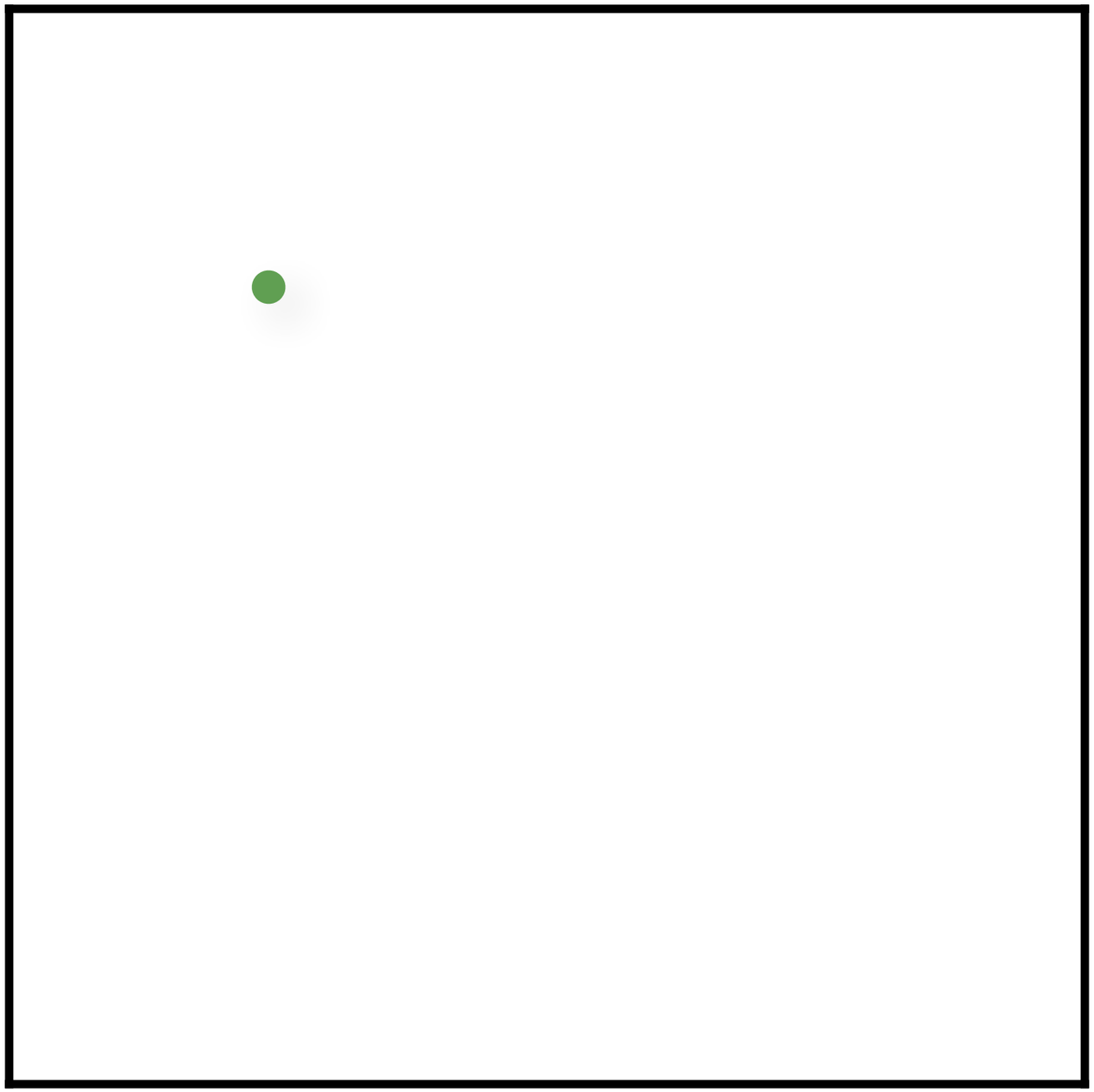
Poisson-Disk/Blue-Noise Sampling

- Enforce a minimum distance between points
- Poisson-Disk Sampling:
  - Robert L. Cook. Stochastic sampling in computer graphics. ACM Transactions on Graphics. 5(1): 51–72, 1986.
  - Ares Lagae and Philip Dutré. A comparison of methods for generating Poisson disk distributions. Computer Graphics Forum, 27(1):114–129, March 2008.

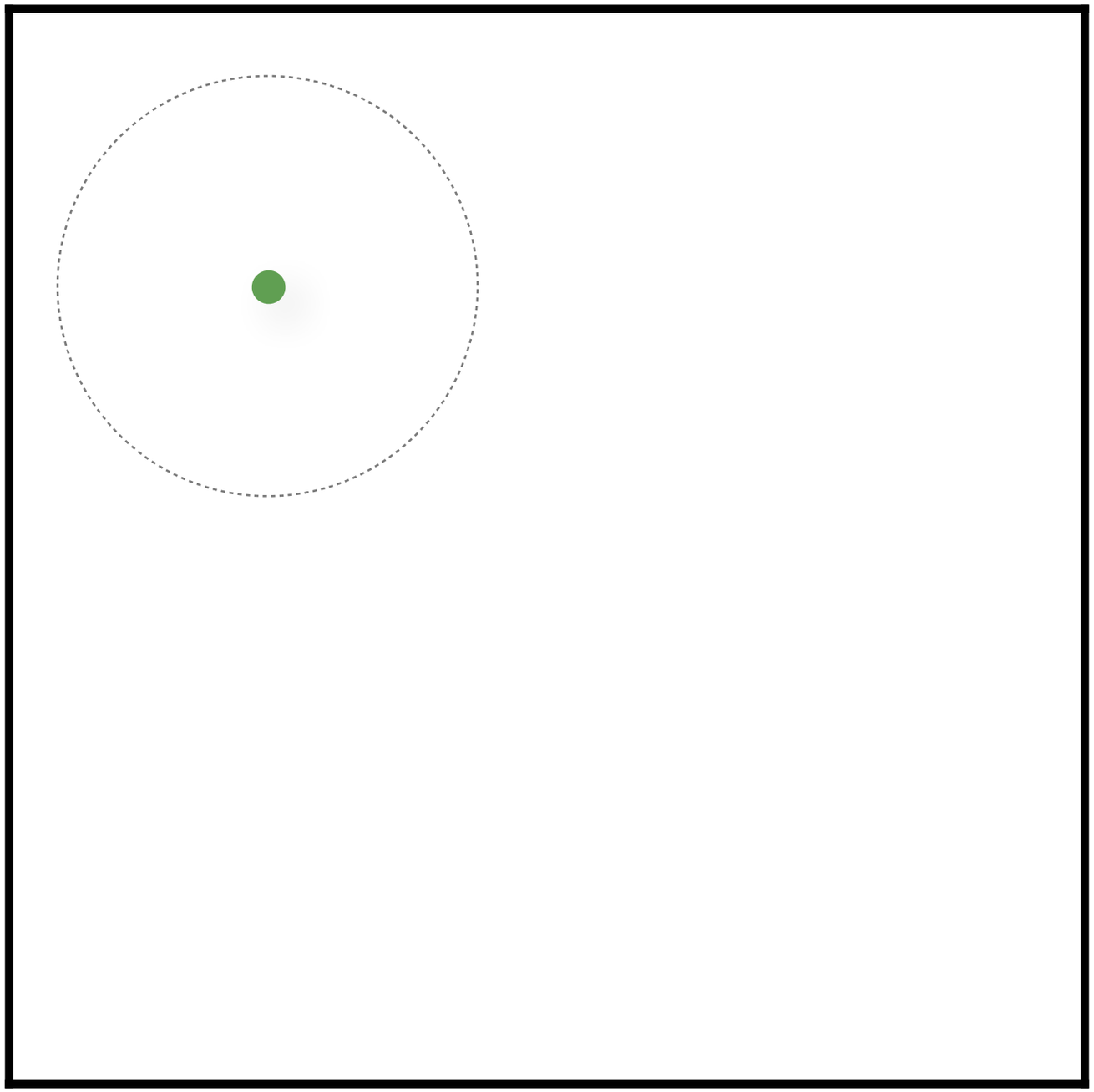
Random Dart Throwing



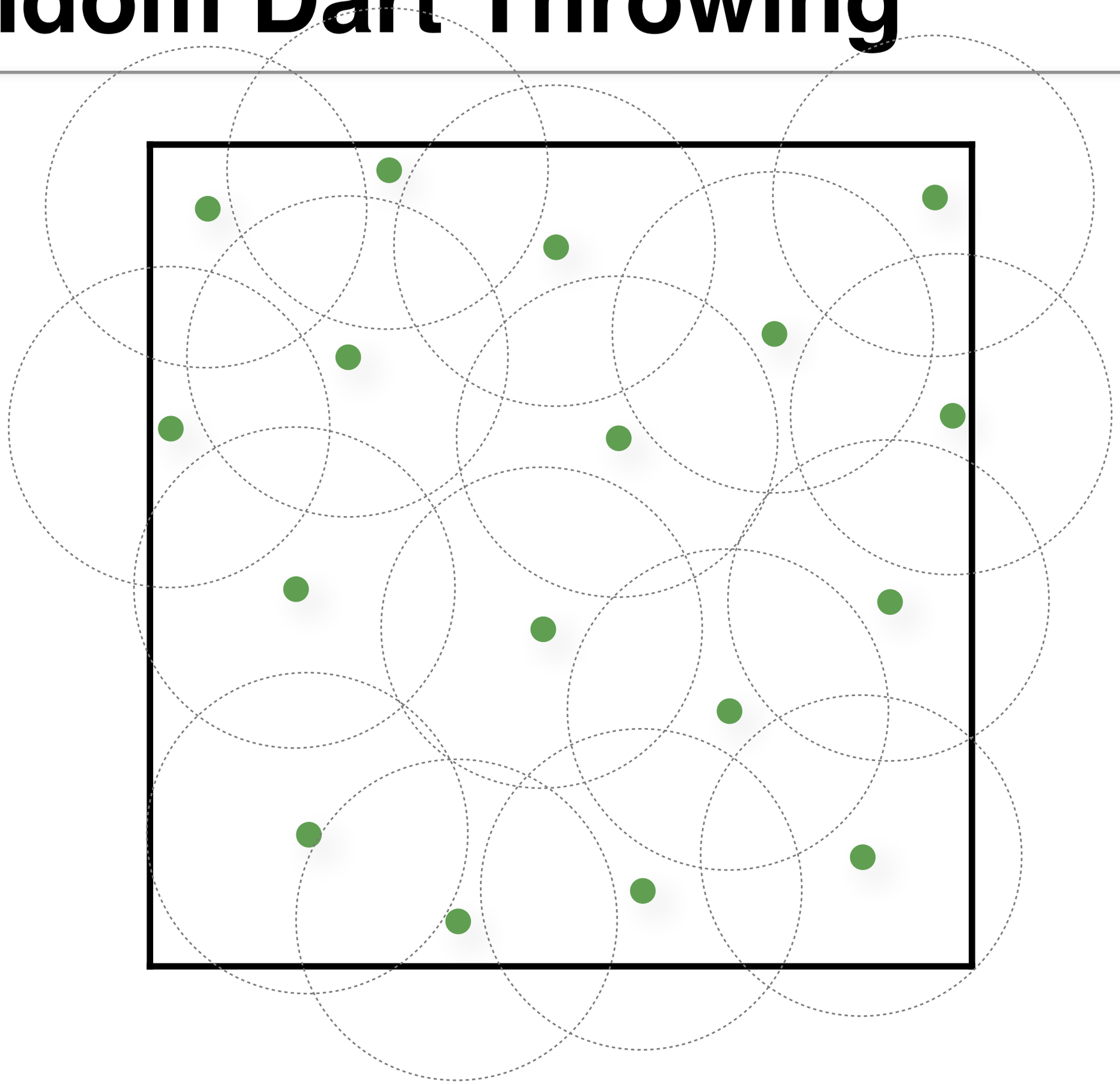
Random Dart Throwing



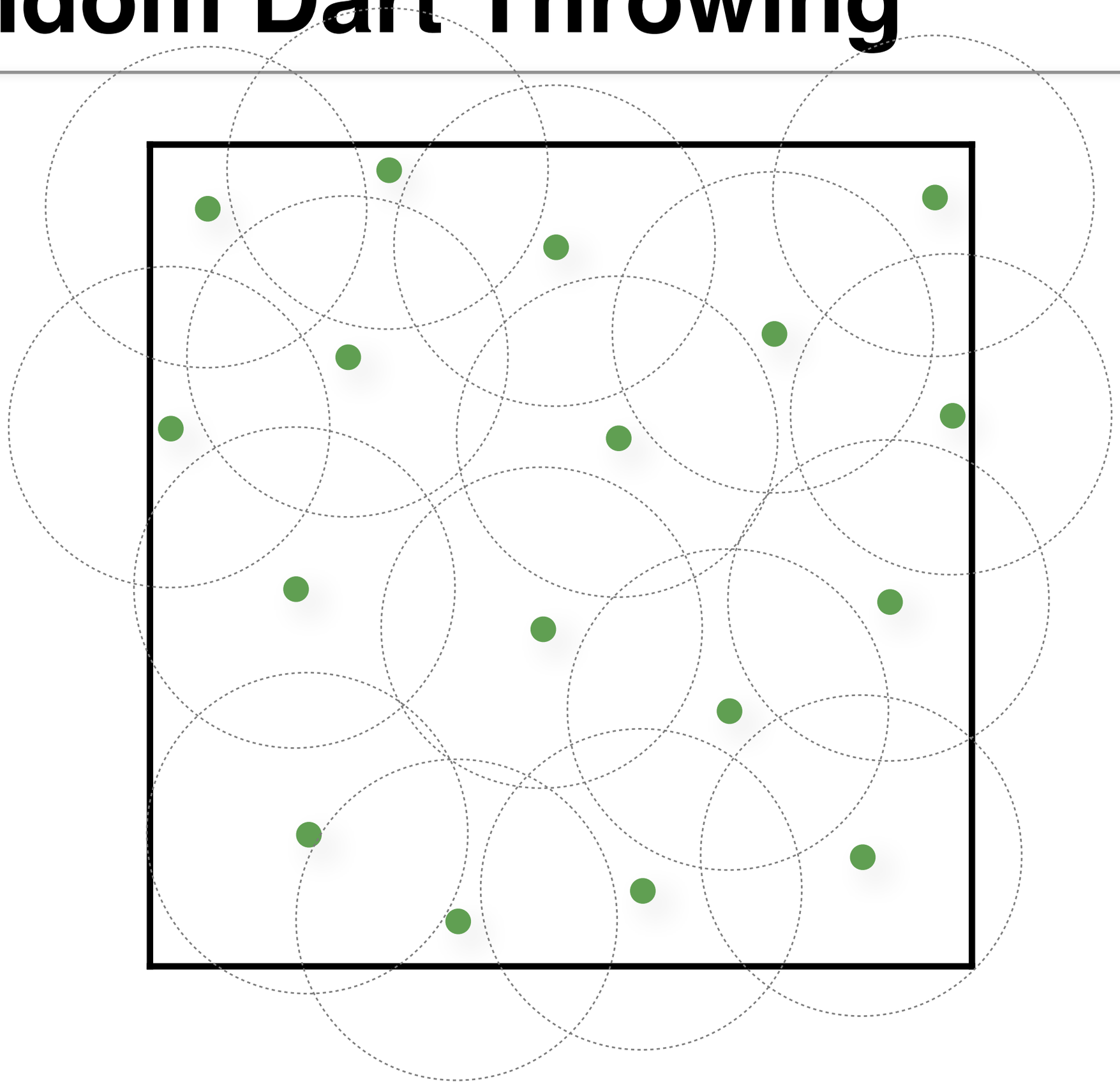
Random Dart Throwing



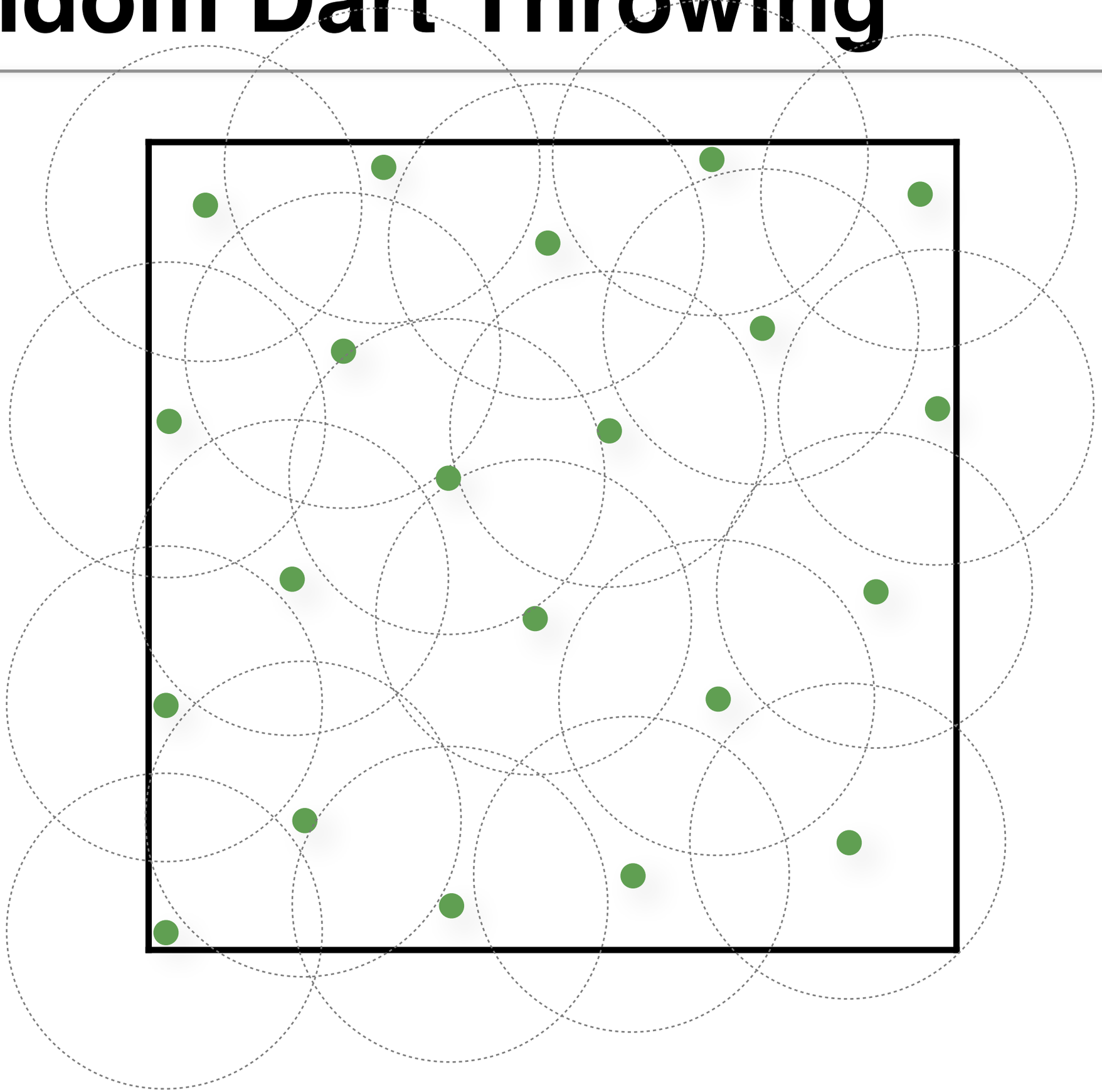
Random Dart Throwing



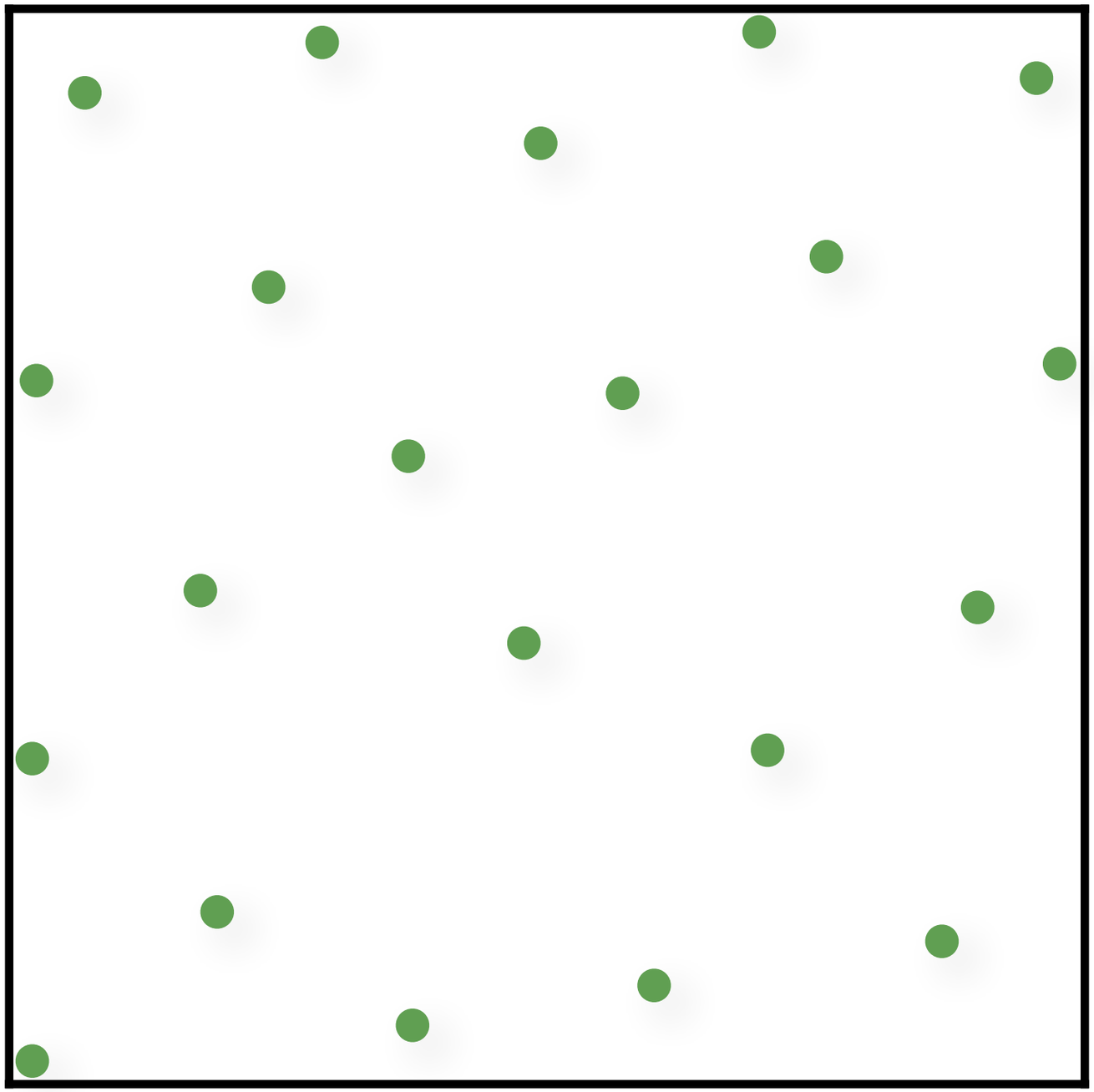
Random Dart Throwing



Random Dart Throwing



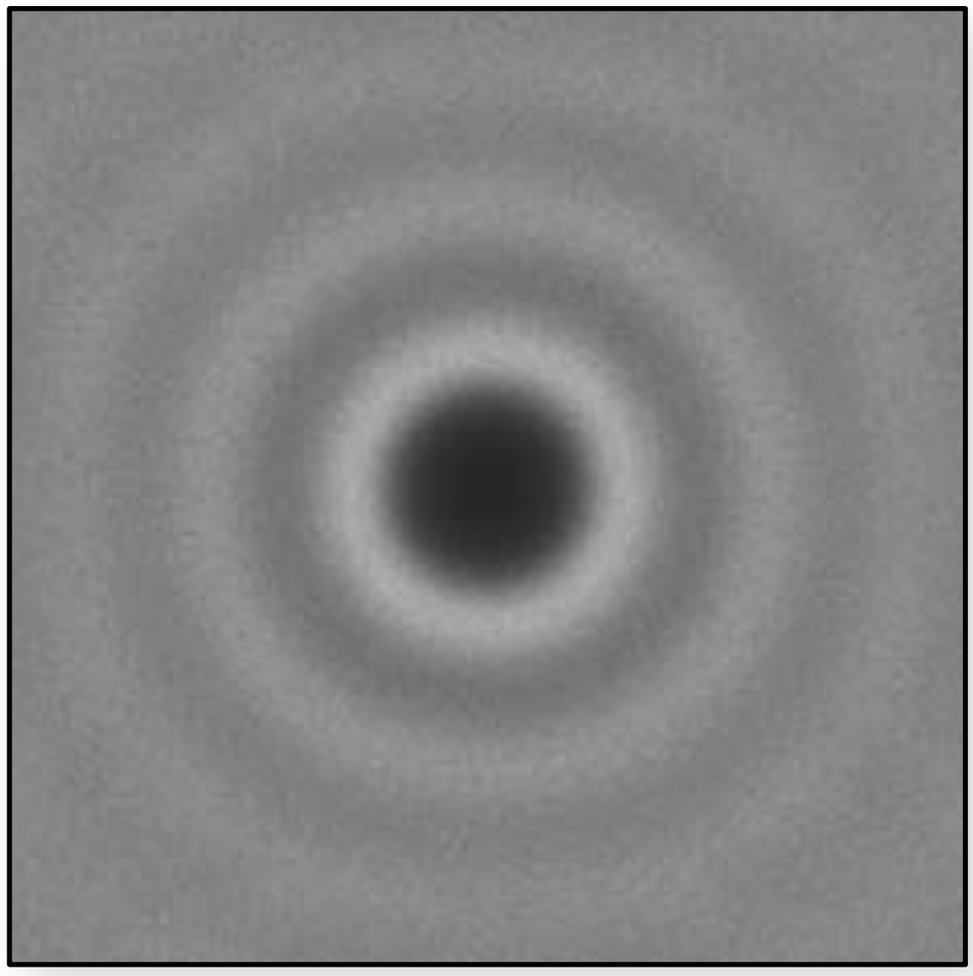
Random Dart Throwing

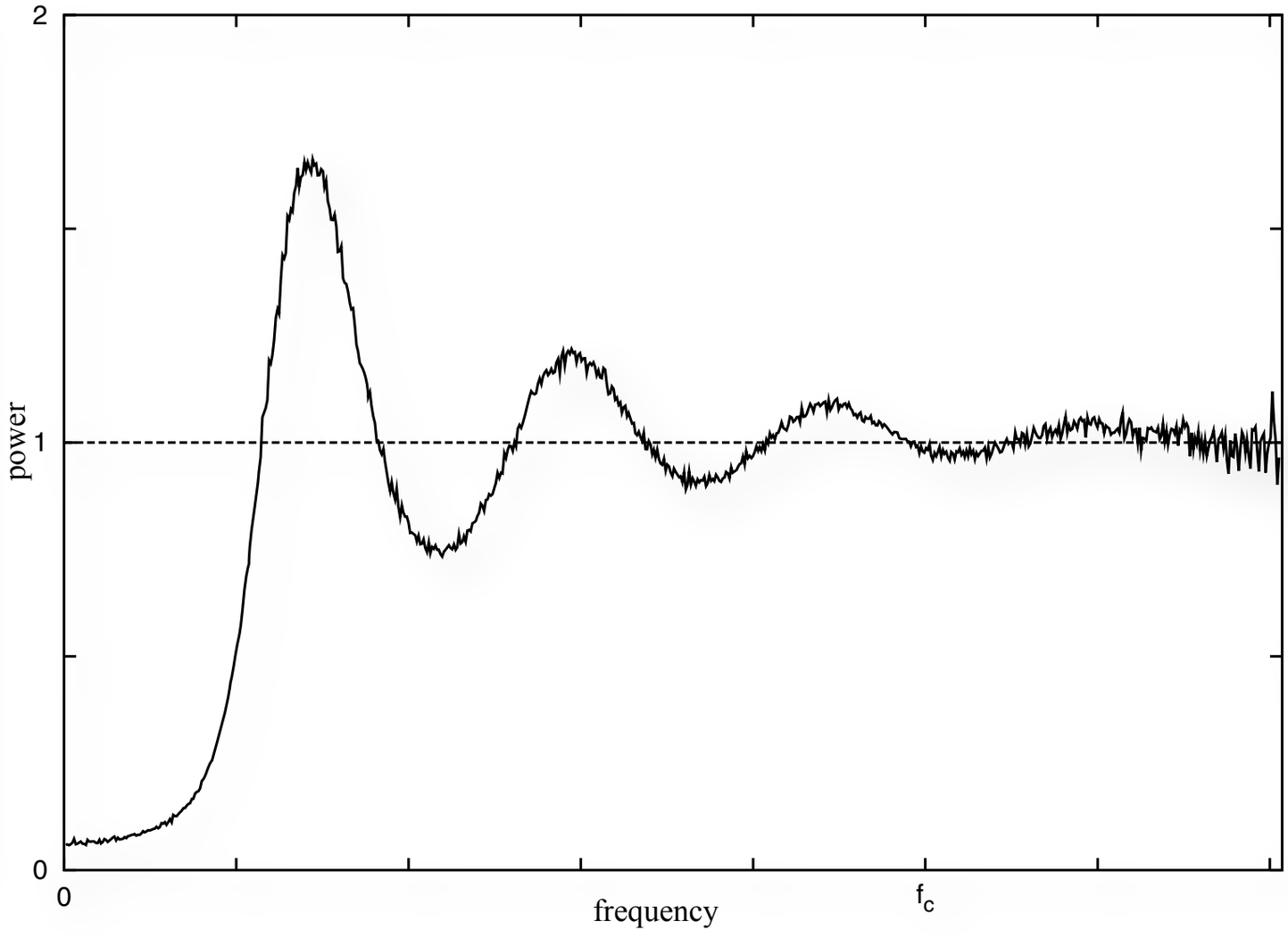


Stratified Sampling

Best-Candidate Sampling

Power Spectrum

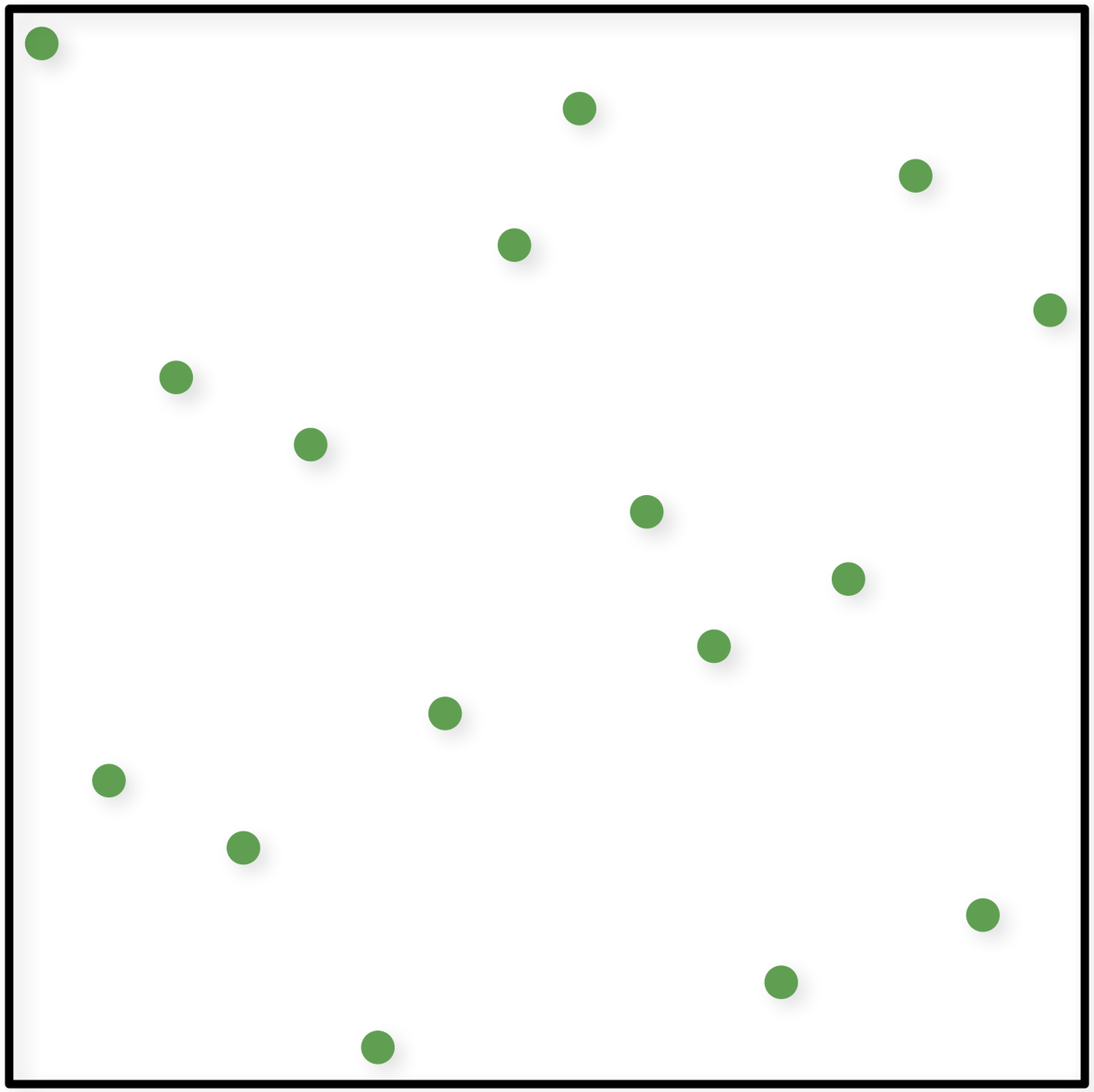




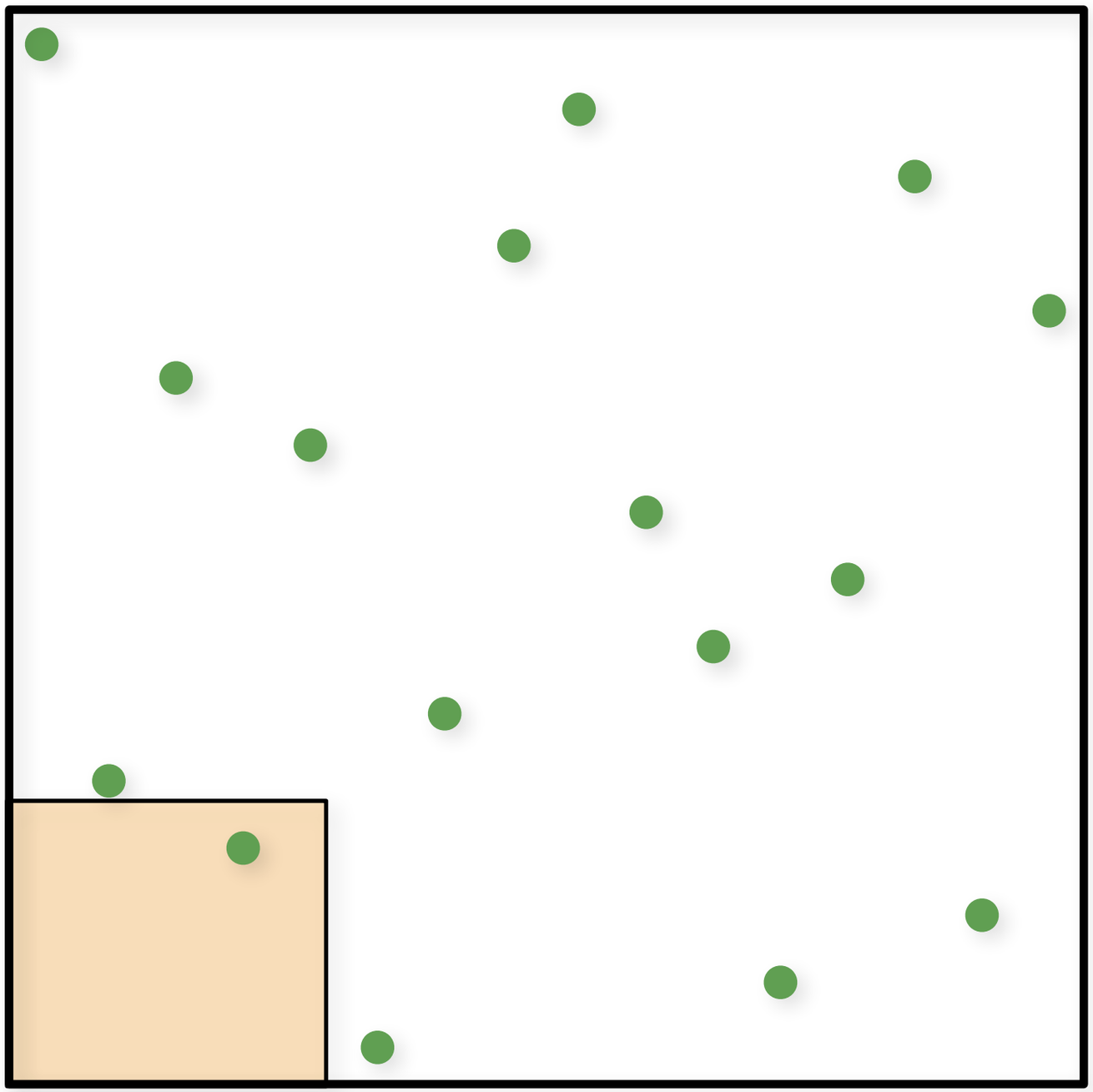
Discrepancy

- Previous stratified approaches try to minimizing "clumping"
- "Discrepancy" is another possible formal definition of clumping: $\mathbf{D^*(x_1, \ldots, x_n)}$
  - maximum difference between fraction of points and volume of containing subregion

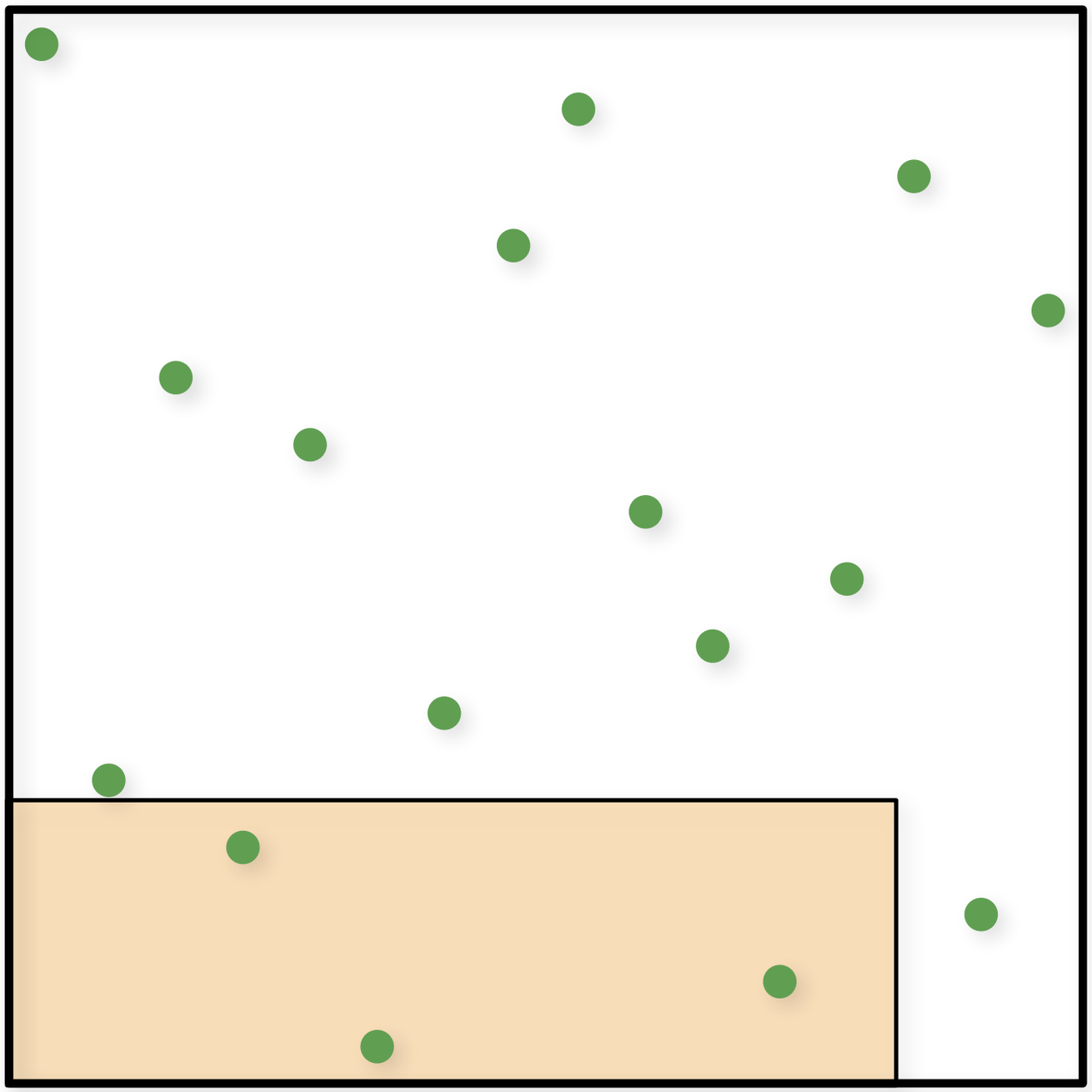
Discrepancy



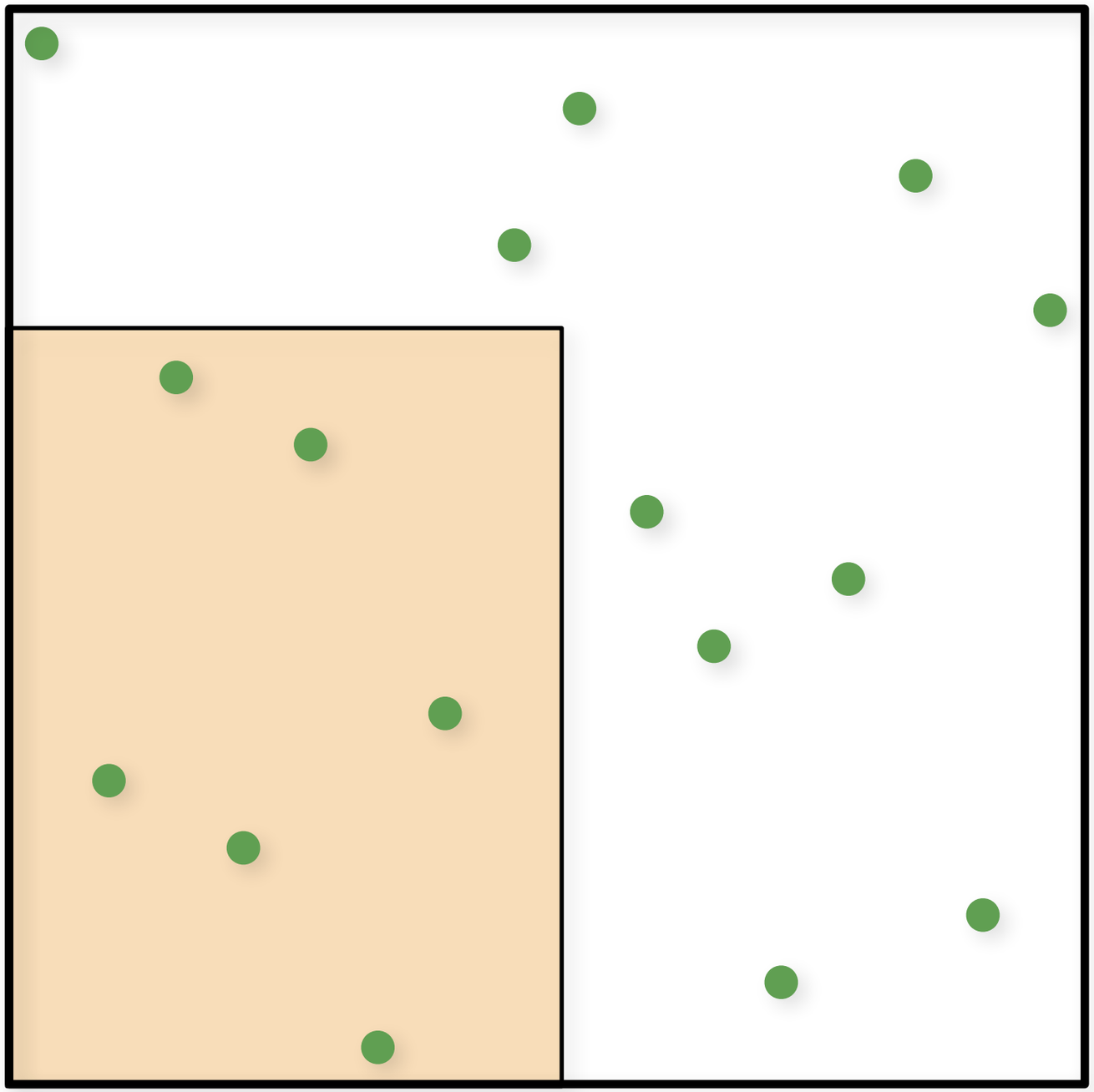
Discrepancy



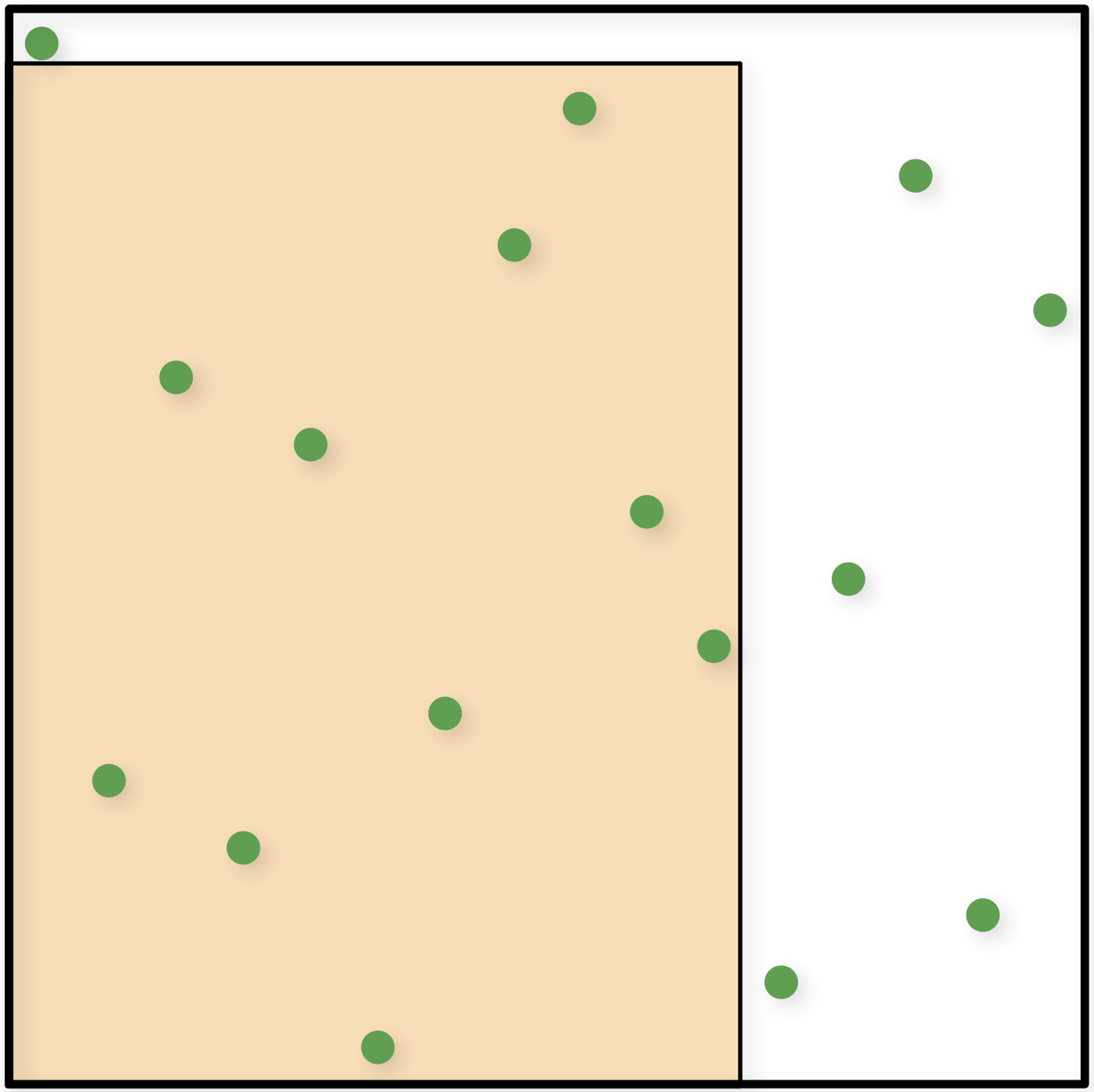
Discrepancy



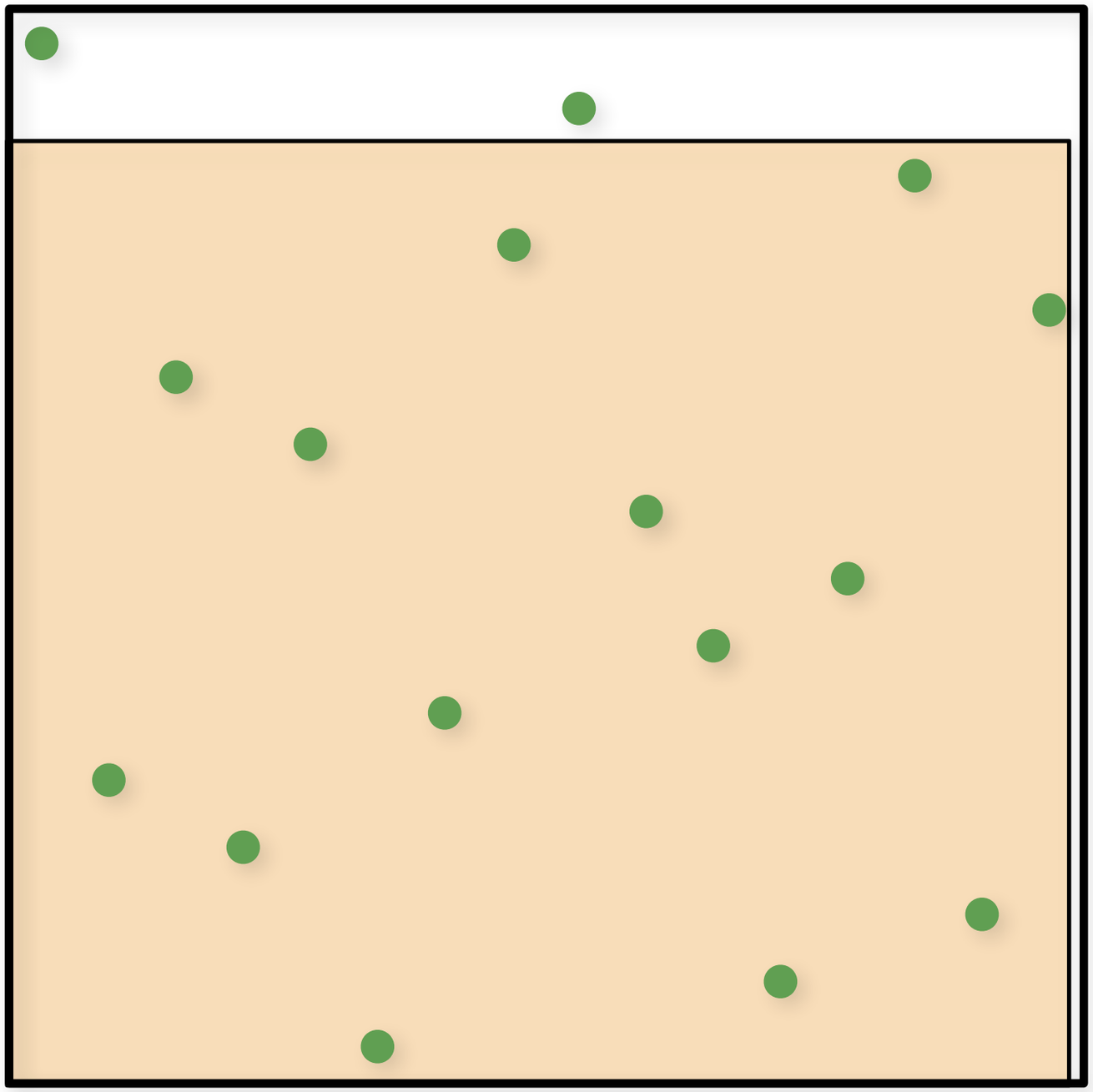
Discrepancy



Discrepancy



Discrepancy



Low-Discrepancy Sampling

- Low-Discrepancy sequences are specially crafted to have small discrepancy values.
- Entire field of study called Quasi-Monte Carlo (QMC)

Koksma-Hlawka inequality

$$\left| \frac{1}{n} \sum_{i=1}^{n} f(x_i) - \int f(u)\,\mathrm{d}u \right| \leq V(f) D^*(x_1, \ldots, x_n)$$

The Radical Inverse

- A positive integer value n can be expressed in a base b with a sequence of digits $d_m...d_2d_1$ uniquely determined by:

$$n = \sum_{i=1}^{\infty} d_i b^{i-1}$$

The Radical Inverse

- A positive integer value n can be expressed in a base b with a sequence of digits $d_m...d_2d_1$ uniquely determined by:

$$n = \sum_{i=1}^{\infty} d_i b^{i-1}$$

- The radical inverse function Φ_b in base b converts a non-negative integer n to a floating-point value in [0, 1) by reflecting these digits about the decimal point:

$$\Phi_b(n) = 0.d_1 d_2 \ldots d_m$$

The Radical Inverse

- A positive integer value n can be expressed in a base b with a sequence of digits $d_m...d_2d_1$ uniquely determined by:

$$n = \sum_{i=1}^{\infty} d_i b^{i-1}$$

- The radical inverse function Φ_b in base b converts a non-negative integer n to a floating-point value in [0, 1) by reflecting these digits about the decimal point:

$$\Phi_b(n) = 0.d_1 d_2 \ldots d_m$$

- Subsequent points “fall into biggest holes”

The Radical Inverse

```
float radicalInverse(int n, int base, float inv)
{
   float v = 0.0f;
   for (float p = inv; n != 0; p *= inv, n /= base)
      v += (n % base) * p;
   return v;
}

float radicalInverse(int n, int base)
{
   return radicalInverse(n, base, 1.0f / base);
}
```

The Radical Inverse

```
float radicalInverse(int n, int base, float inv)
{
   float v = 0.0f;
   for (float p = inv; n != 0; p *= inv, n /= base)
      v += (n % base) * p;
   return v;
}

float radicalInverse(int n, int base)
{
   return radicalInverse(n, base, 1.0f / base);
}
```

More efficient version available for base 2

The Van der Corput Sequence

- Radical Inverse in base two

n	Base 2	Φ_b

The Van der Corput Sequence

- Radical Inverse in base two

n	Base 2	Φ_b
1	1	.1 = 1/2

The Van der Corput Sequence

- Radical Inverse in base two

n	Base 2	Φ_b
1	1	.1 = 1/2
2	10	.01 = 1/4

The Van der Corput Sequence

- Radical Inverse in base two

n	Base 2	Φ_b
1	1	.1 = 1/2
2	10	.01 = 1/4
3	11	.11 = 3/4

The Van der Corput Sequence

- Radical Inverse in base two

n	Base 2	Φ_b
1	1	.1 = 1/2
2	10	.01 = 1/4
3	11	.11 = 3/4
4	100	.001 = 1/8

The Van der Corput Sequence

- Radical Inverse in base two

n	Base 2	Φ_b
1	1	.1 = 1/2
2	10	.01 = 1/4
3	11	.11 = 3/4
4	100	.001 = 1/8
5	101	.101 = 5/8

The Van der Corput Sequence

- Radical Inverse in base two

n	Base 2	Φ_b
1	1	.1 = 1/2
2	10	.01 = 1/4
3	11	.11 = 3/4
4	100	.001 = 1/8
5	101	.101 = 5/8
6	110	.011 = 3/8

The Van der Corput Sequence

- Radical Inverse in base two

n	Base 2	Φ_b
1	1	.1 = 1/2
2	10	.01 = 1/4
3	11	.11 = 3/4
4	100	.001 = 1/8
5	101	.101 = 5/8
6	110	.011 = 3/8
7	111	.111 = 7/8

The Van der Corput Sequence

- Radical Inverse in base two

n	Base 2	Φ_b
1	1	.1 = 1/2
2	10	.01 = 1/4
3	11	.11 = 3/4
4	100	.001 = 1/8
5	101	.101 = 5/8
6	110	.011 = 3/8
7	111	.111 = 7/8
...		

The Van der Corput Sequence

- Radical Inverse in base two

n	Base 2	Φ_b
1	1	.1 = 1/2
2	10	.01 = 1/4
3	11	.11 = 3/4
4	100	.001 = 1/8
5	101	.101 = 5/8
6	110	.011 = 3/8
7	111	.111 = 7/8
...		

Discrepancy: $D^*(x_1,\ldots,x_n) \in O\left(\frac{\log N}{N}\right)$

The Radical Inverse (Base 2)

```
float vanDerCorputRIU(uint n)
{
    n = (n << 16) | (n >> 16);
    n = ((n & 0x00ff00ff) << 8) | ((n & 0xff00ff00) >> 8);
    n = ((n & 0x0f0f0f0f) << 4) | ((n & 0xf0f0f0f0) >> 4);
    n = ((n & 0x33333333) << 2) | ((n & 0xcccccccc) >> 2);
    n = ((n & 0x55555555) << 1) | ((n & 0xaaaaaaaa) >> 1);
    return n / float (0x100000000LL);
}
```

The Halton Sequence

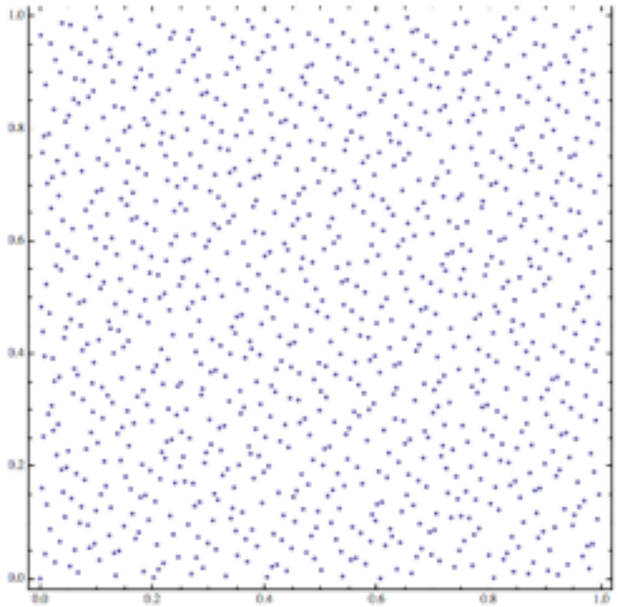
- An n-dimensional Halton sequence uses the radical inverse with a different base b for each dimension of the pattern.
- The bases must all be relatively prime.

$$x_i = (\Phi_2(i), \Phi_3(i), \Phi_5(i), \ldots, \Phi_{p_n}(i))$$

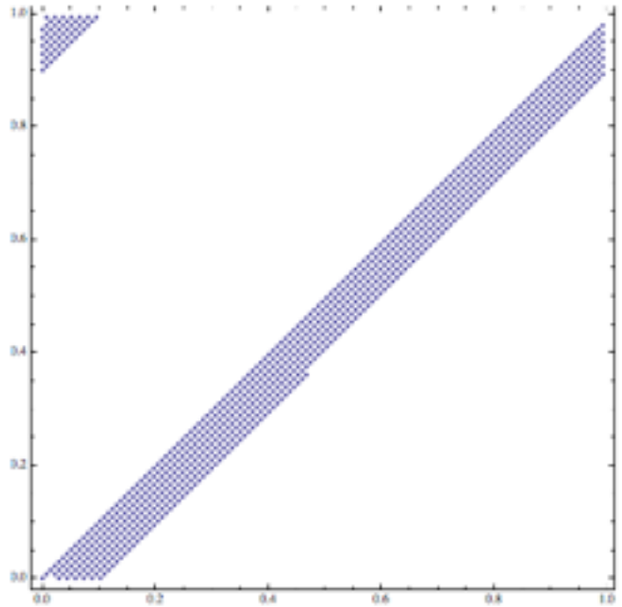
- Incremental

Scrambling

Without scrambling

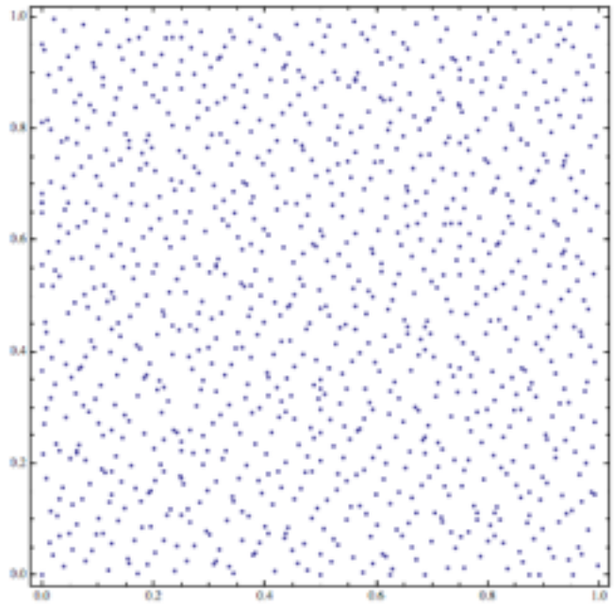


Dimensions 1 and 2

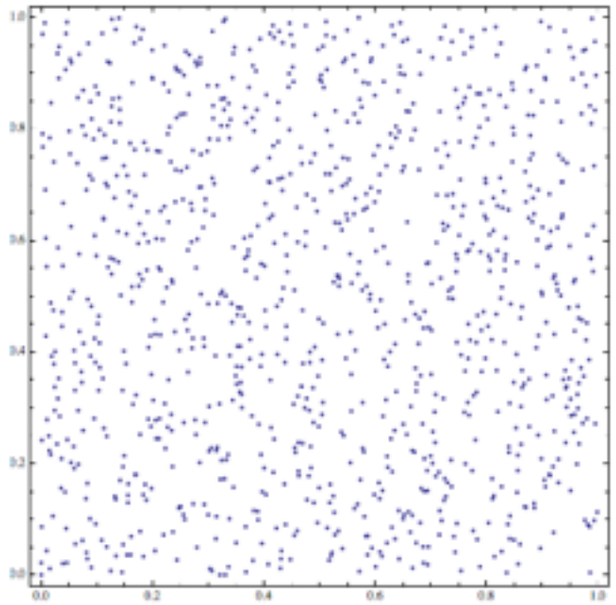


Dimensions 32 and 33

With scrambling



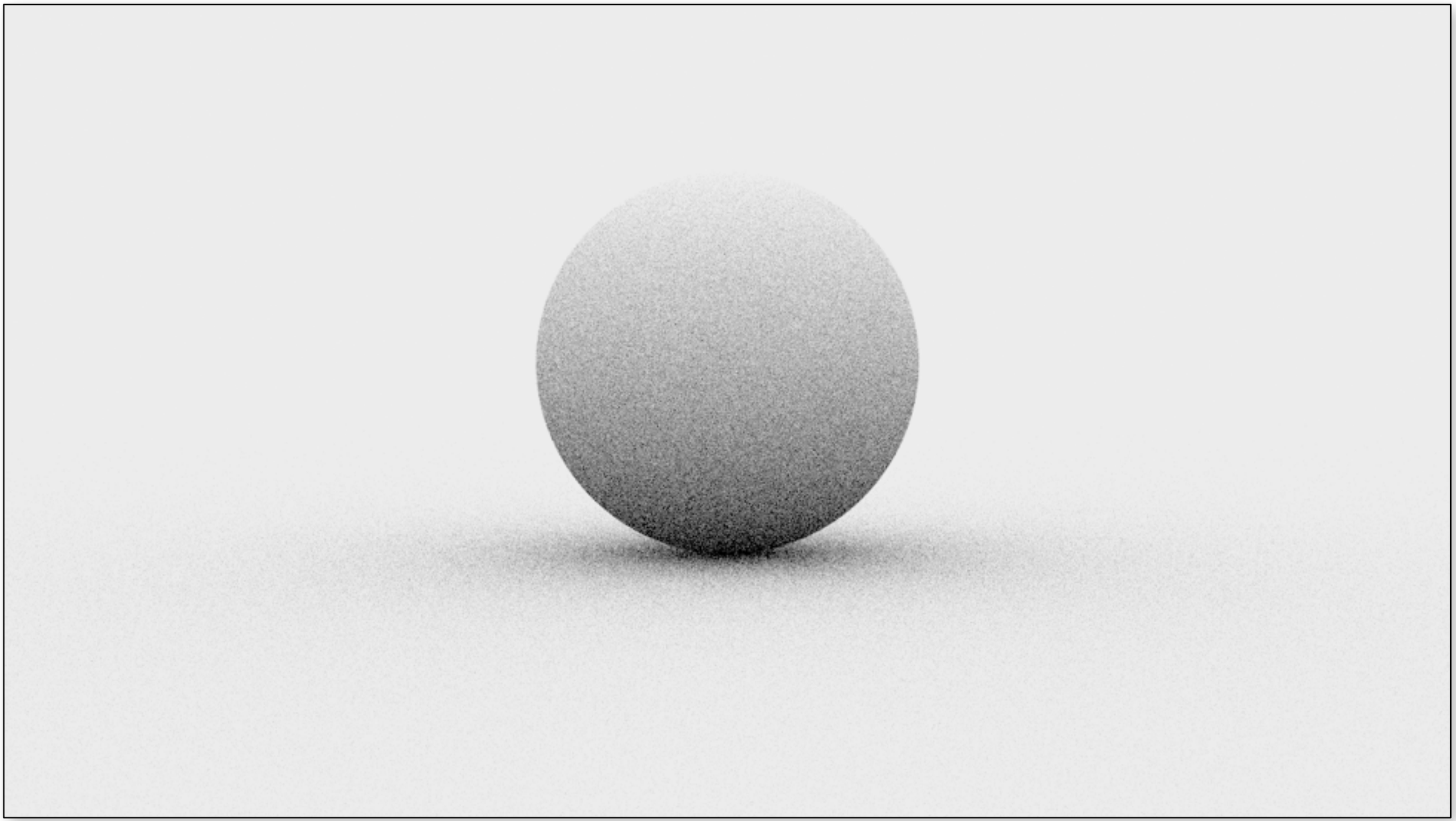
Dimensions 1 and 2



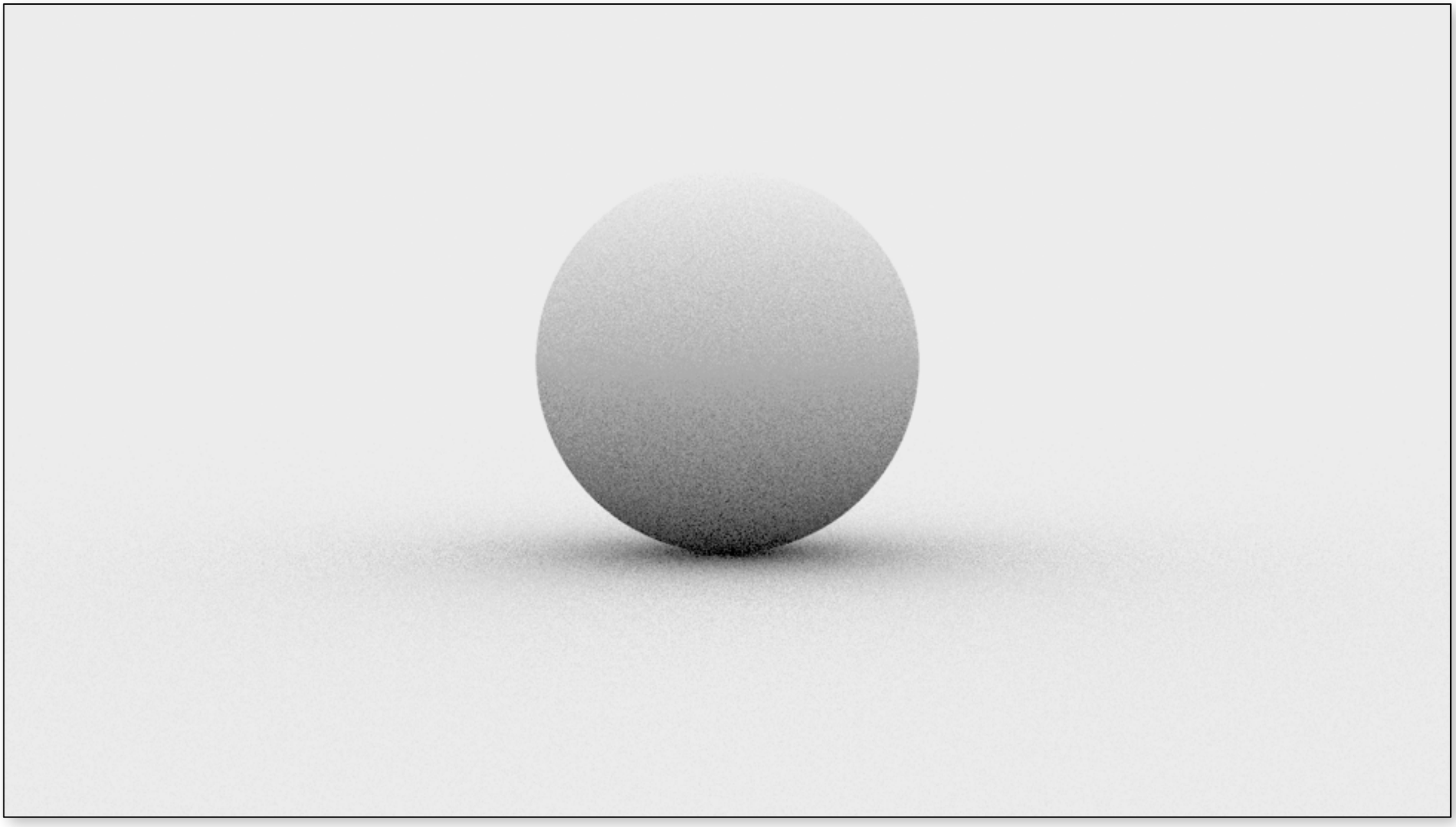
Dimensions 32 and 33

$$\Phi_b(n) = 0.\pi(d_1)\pi(d_2)...\pi(d_m)$$

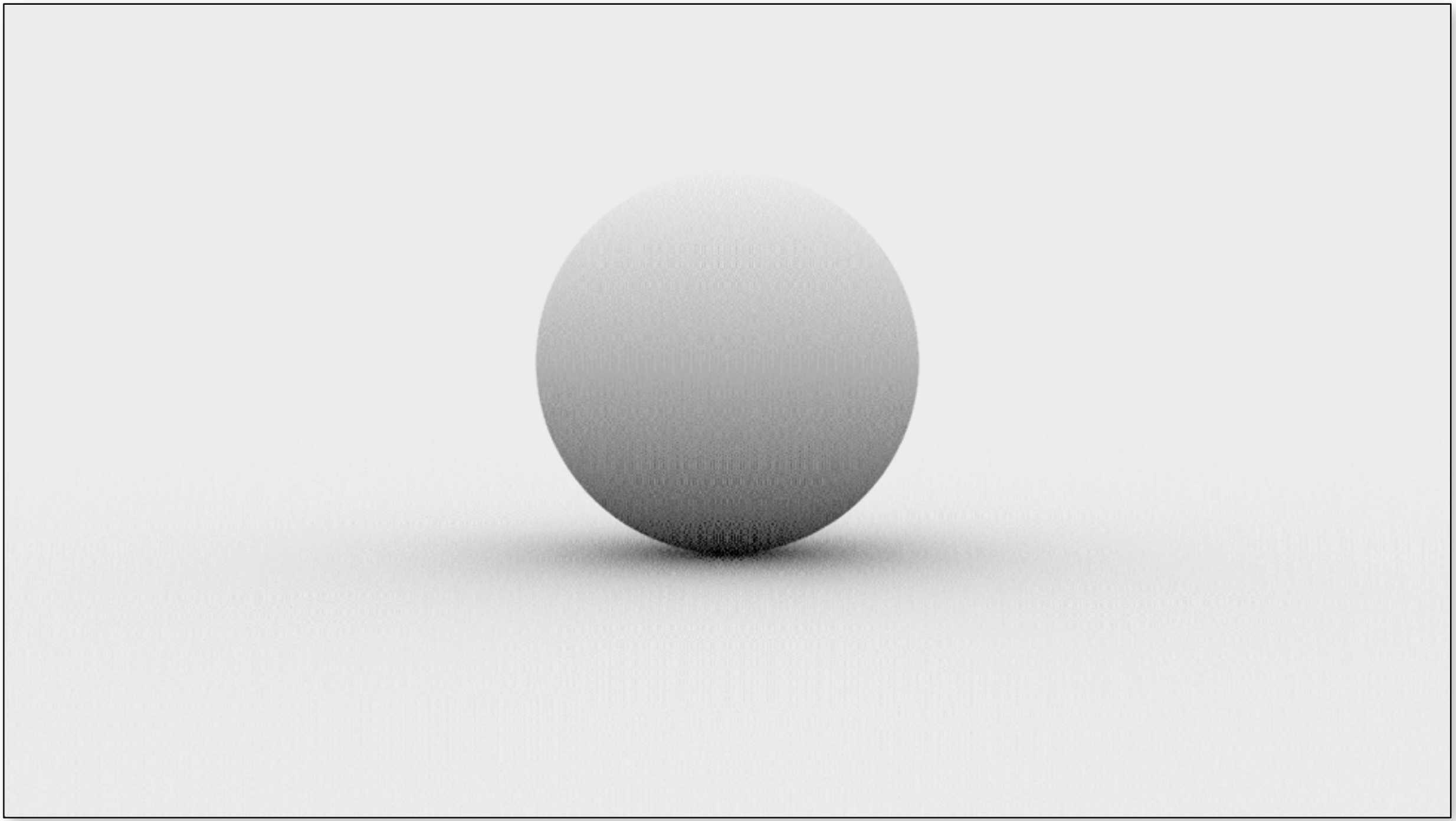
Monte Carlo (16 random samples)



Monte Carlo (16 stratified samples)



Quasi-Monte Carlo (16 Halton samples)

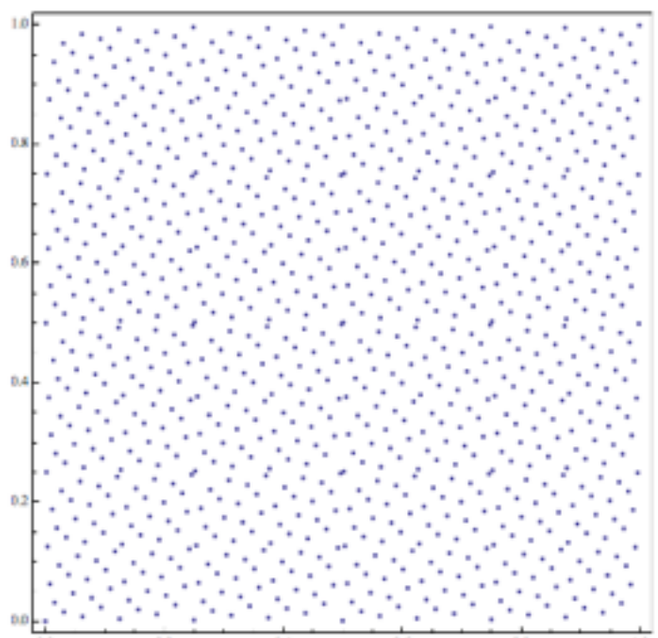


The Hammersley Sequence

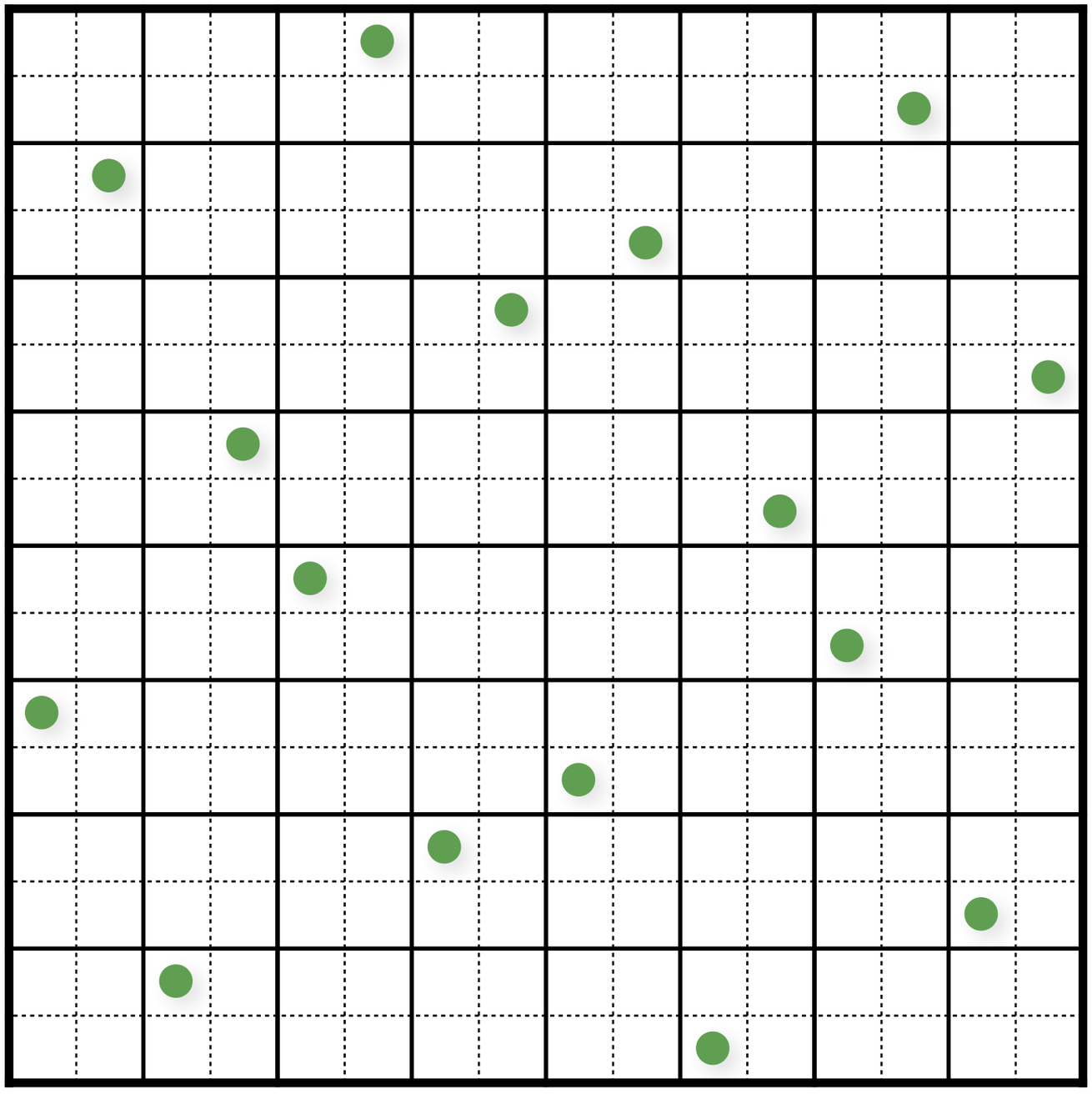
- Same as Halton, but uses i/N for first dimension:

$$x_i = \left(\frac{i}{N}, \Phi_2(i), \Phi_3(i), \ldots, \Phi_{p_n}(i) \right)$$

- Provides slightly lower discrepancy
- Not incremental, need to know total number of samples, N, in advance

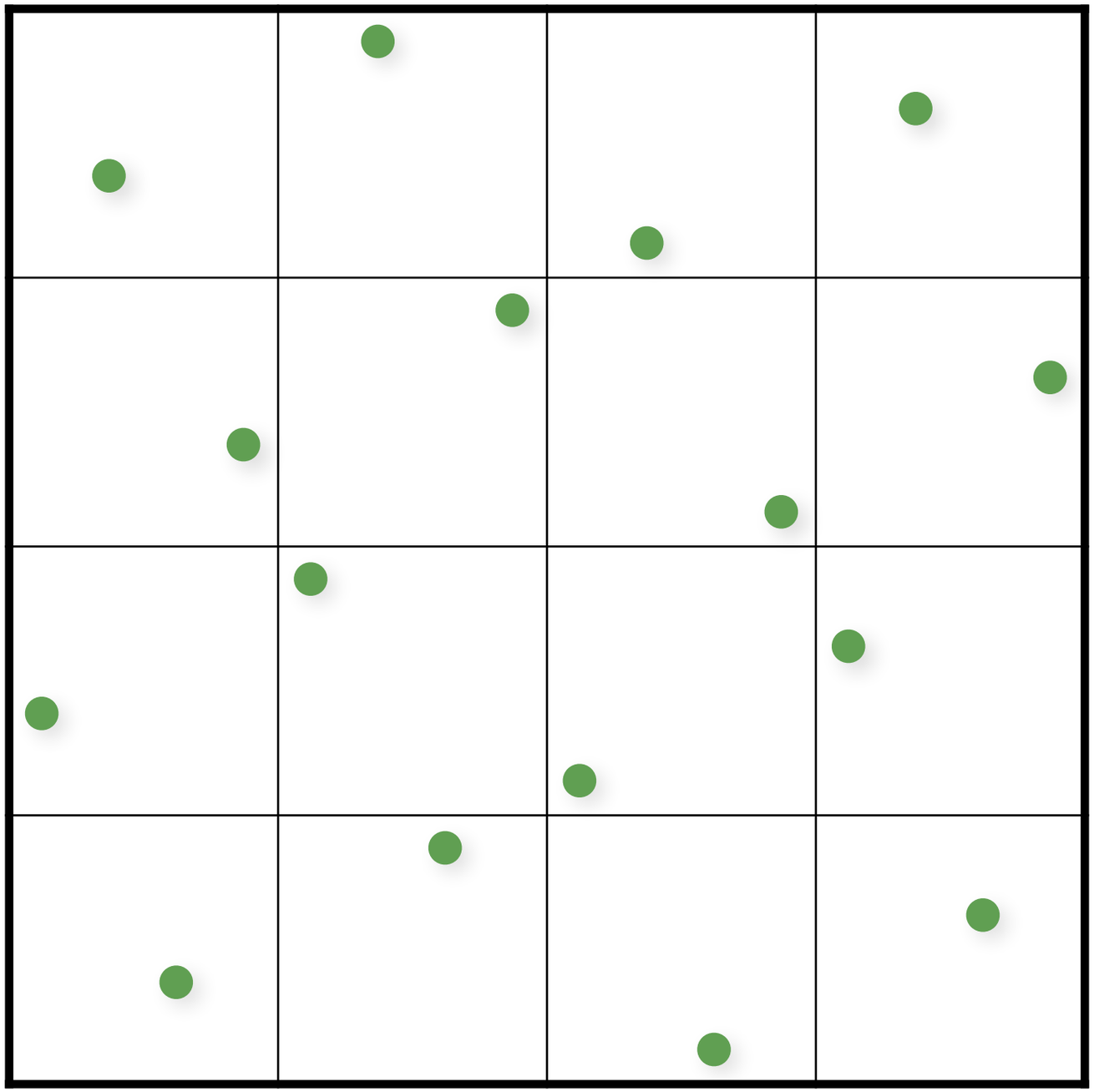


(0,2)-Sequences



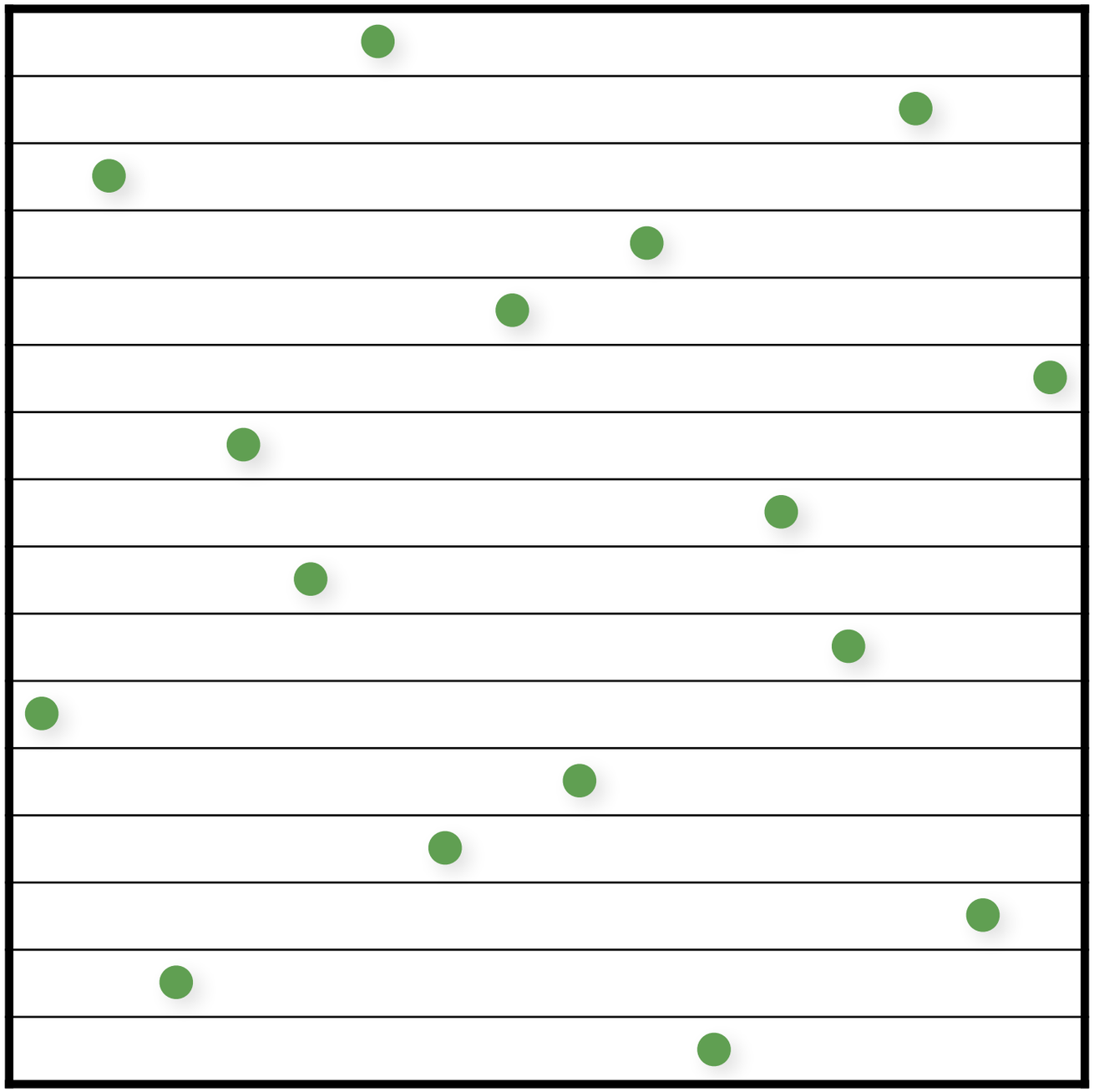
1 sample in each “elementary interval”

(0,2)-Sequences



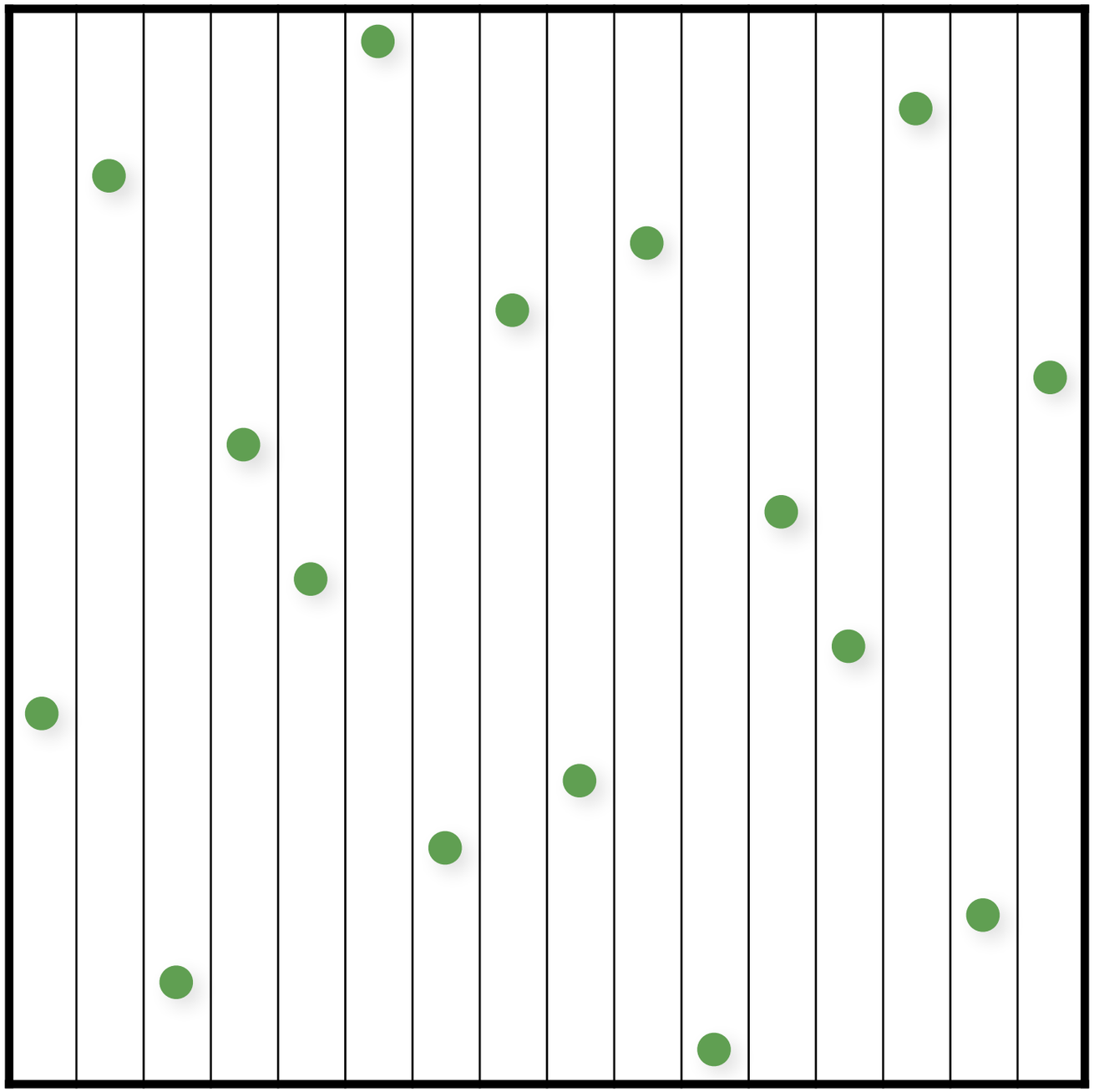
1 sample in each “elementary interval”

(0,2)-Sequences



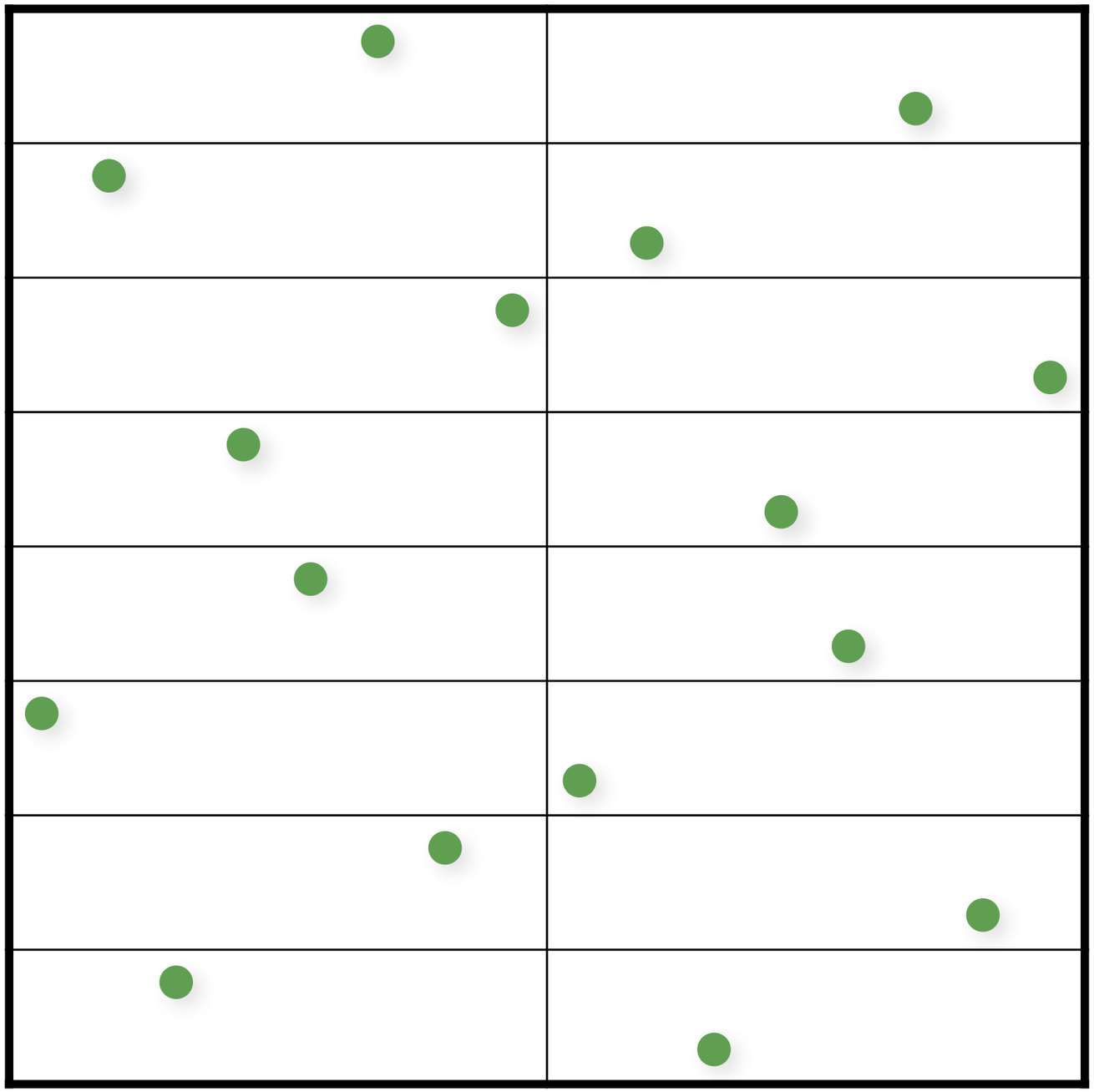
1 sample in each “elementary interval”

(0,2)-Sequences



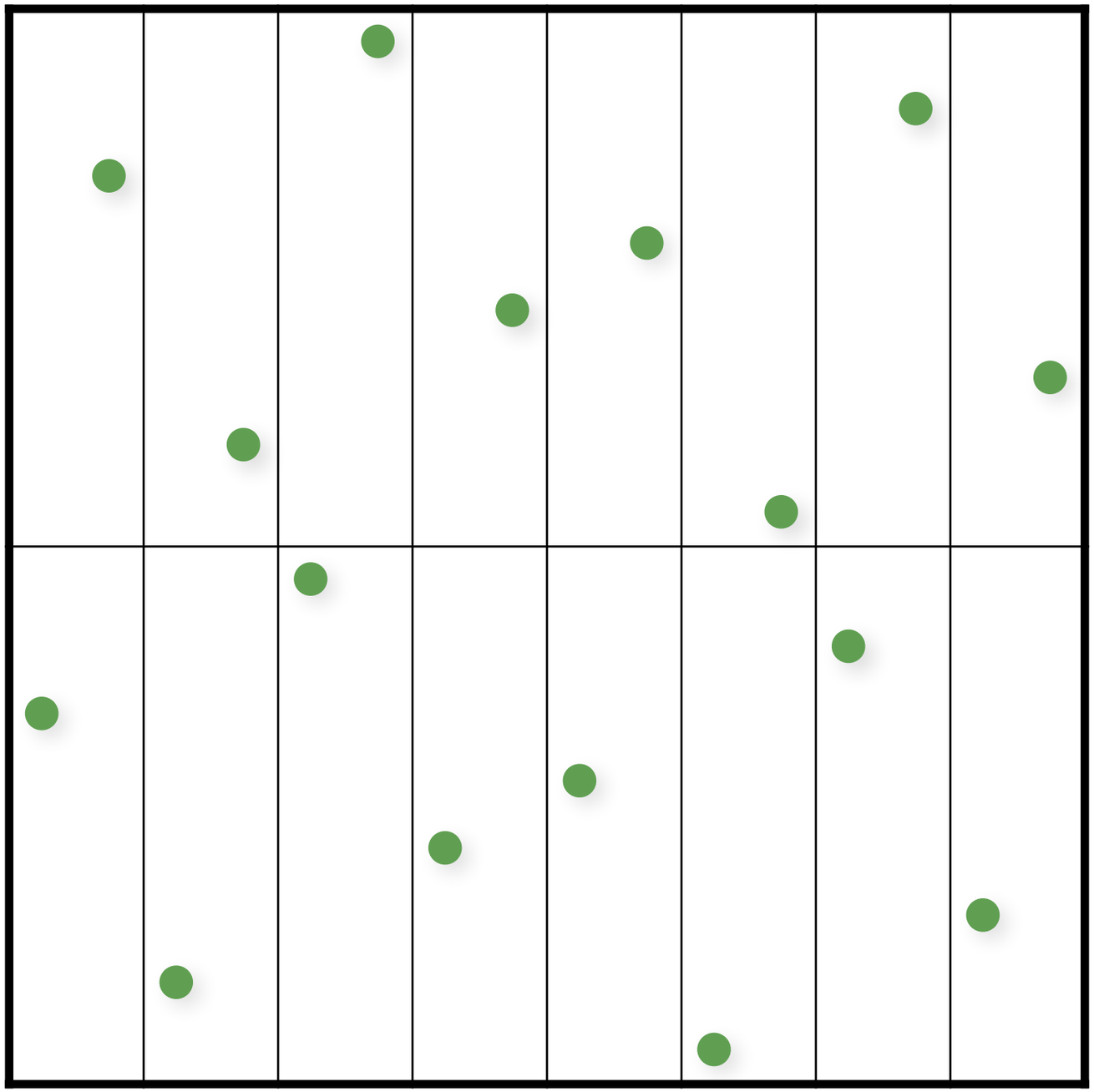
1 sample in each “elementary interval”

(0,2)-Sequences



1 sample in each “elementary interval”

(0,2)-Sequences



1 sample in each “elementary interval”

Additional resources

Enumerating Quasi-Monte Carlo Point Sequences in Elementary Intervals

L. Grünschloß, M. Raab and A. Keller

Monte Carlo and Quasi-Monte Carlo Methods 2010

http://gruenschloss.org/

Many more...

- Sobol
- Faure
- Larcher-Pillichshammer
- Folded Radical Inverse
- (t,s)-sequences & (t,m,s)-nets
- Scrambling/randomization
- much more...

Randomized/Scrambled Sequences

- LD sequence identical for multiple runs
  - cannot average independent images!
  - no “random” seed

Randomized/Scrambled Sequences

- LD sequence identical for multiple runs
  - cannot average independent images!
  - no “random” seed
- Cranley-Patterson rotation: add random number modulo 1
  - does not preserve elementary interval property

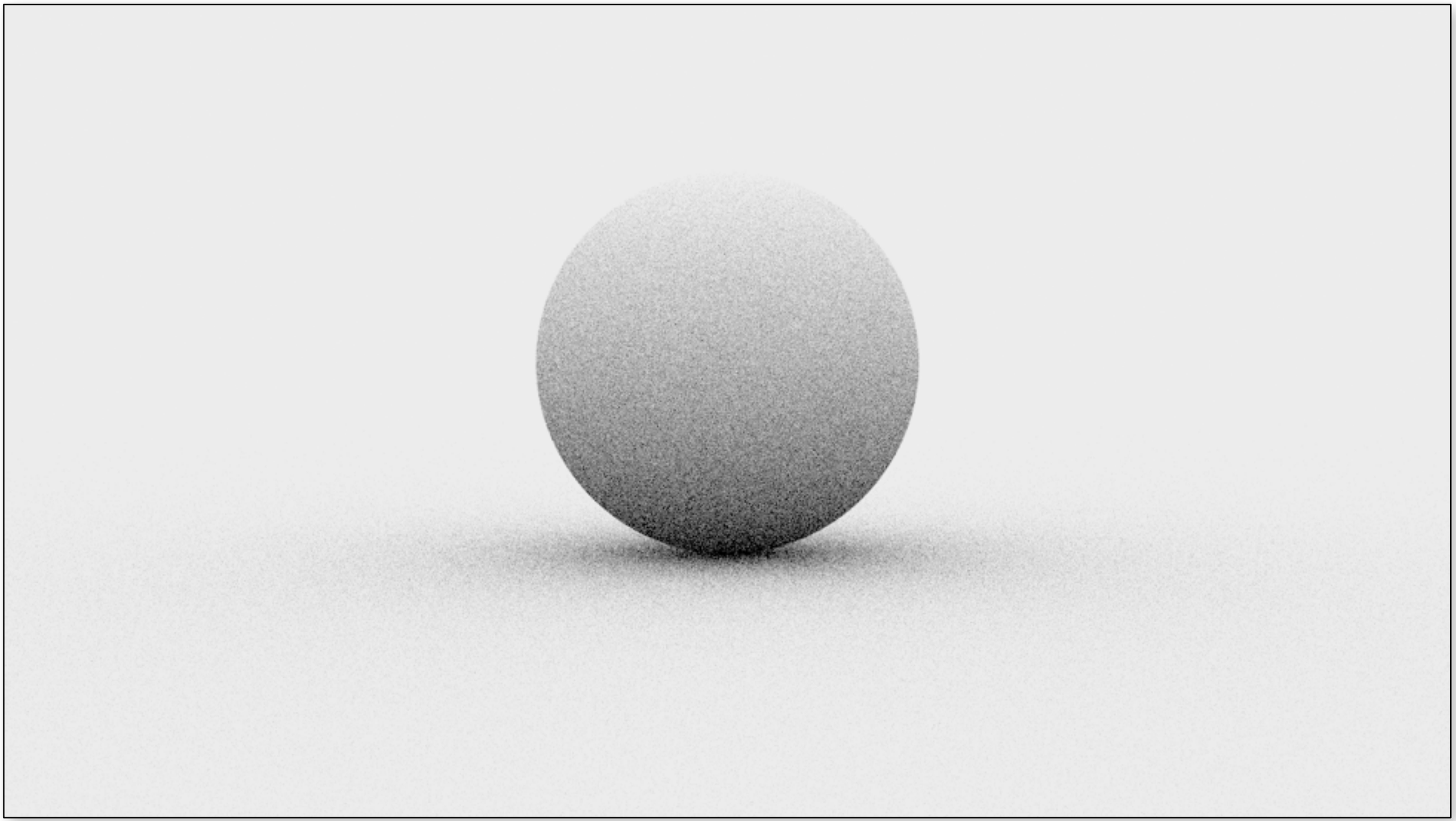
Randomized/Scrambled Sequences

- LD sequence identical for multiple runs
  - cannot average independent images!
  - no “random” seed
- Cranley-Patterson rotation: add random number modulo 1
  - does not preserve elementary interval property
- Random permutations: compute a permutation table for the order of the digits and use it when computing the radical inverse
  - Can be done very efficiently for base 2 with XOR operation

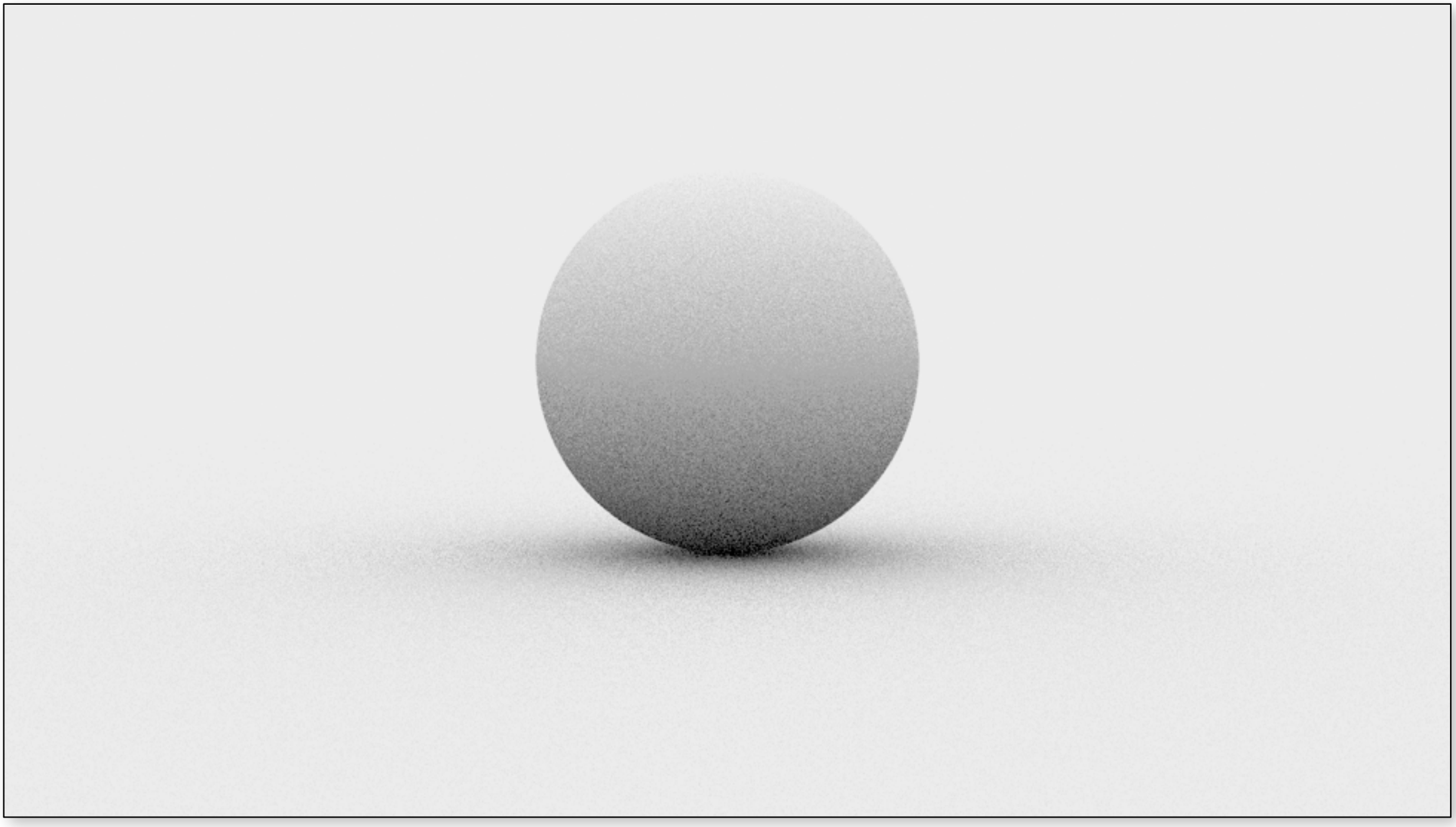
Radical Inverse (Base 2)

```
float vanDerCorputRIU(uint n, uint scramble = 0)
{
    n = (n << 16) | (n >> 16);
    n = ((n & 0x00ff00ff) << 8) | ((n & 0xff00ff00) >> 8);
    n = ((n & 0x0f0f0f0f) << 4) | ((n & 0xf0f0f0f0) >> 4);
    n = ((n & 0x33333333) << 2) | ((n & 0xcccccccc) >> 2);
    n = ((n & 0x55555555) << 1) | ((n & 0xaaaaaaaa) >> 1);
    n ^= scramble;
    return n / float (0x100000000LL);
}
```

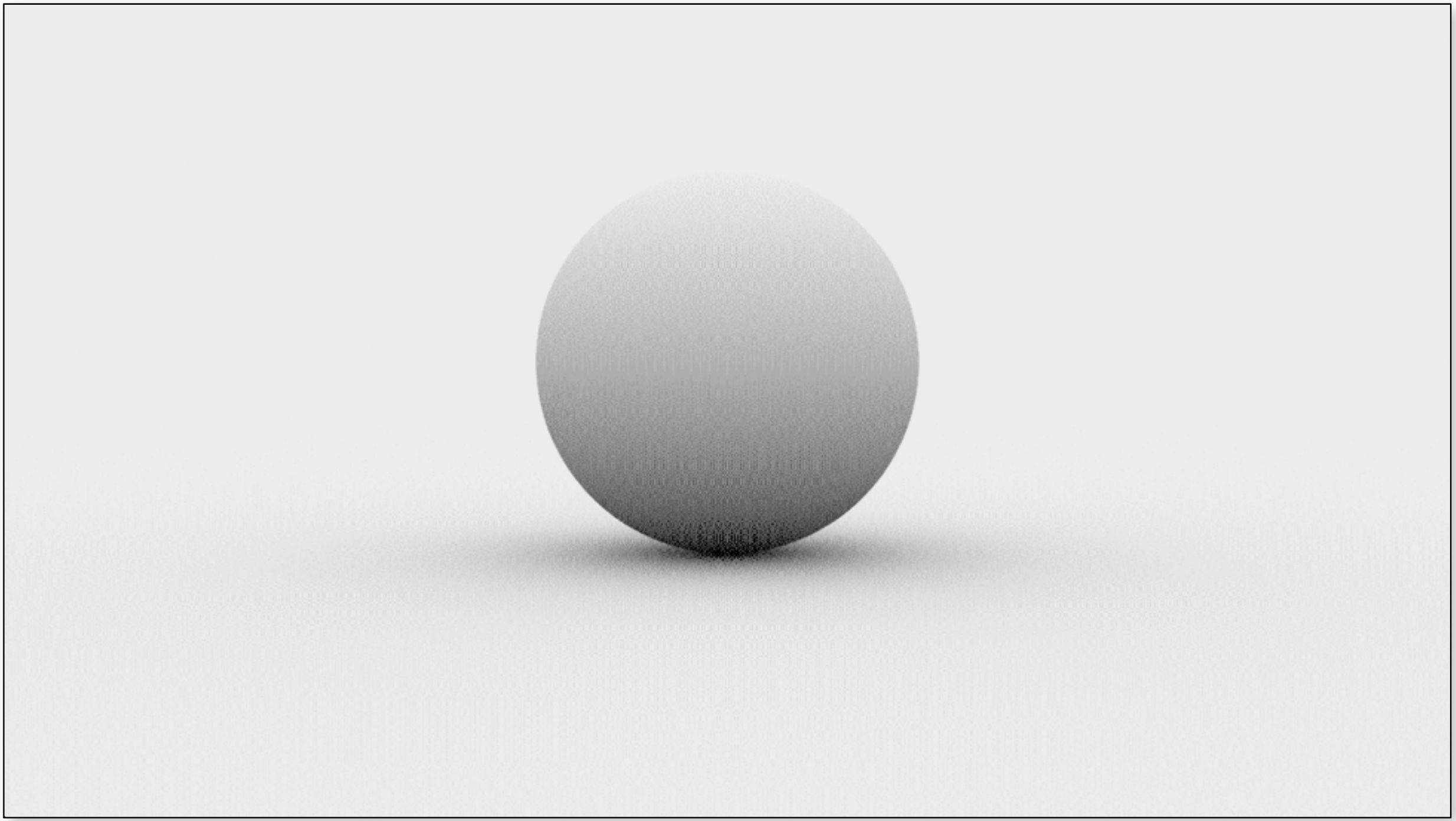
Monte Carlo (16 random samples)



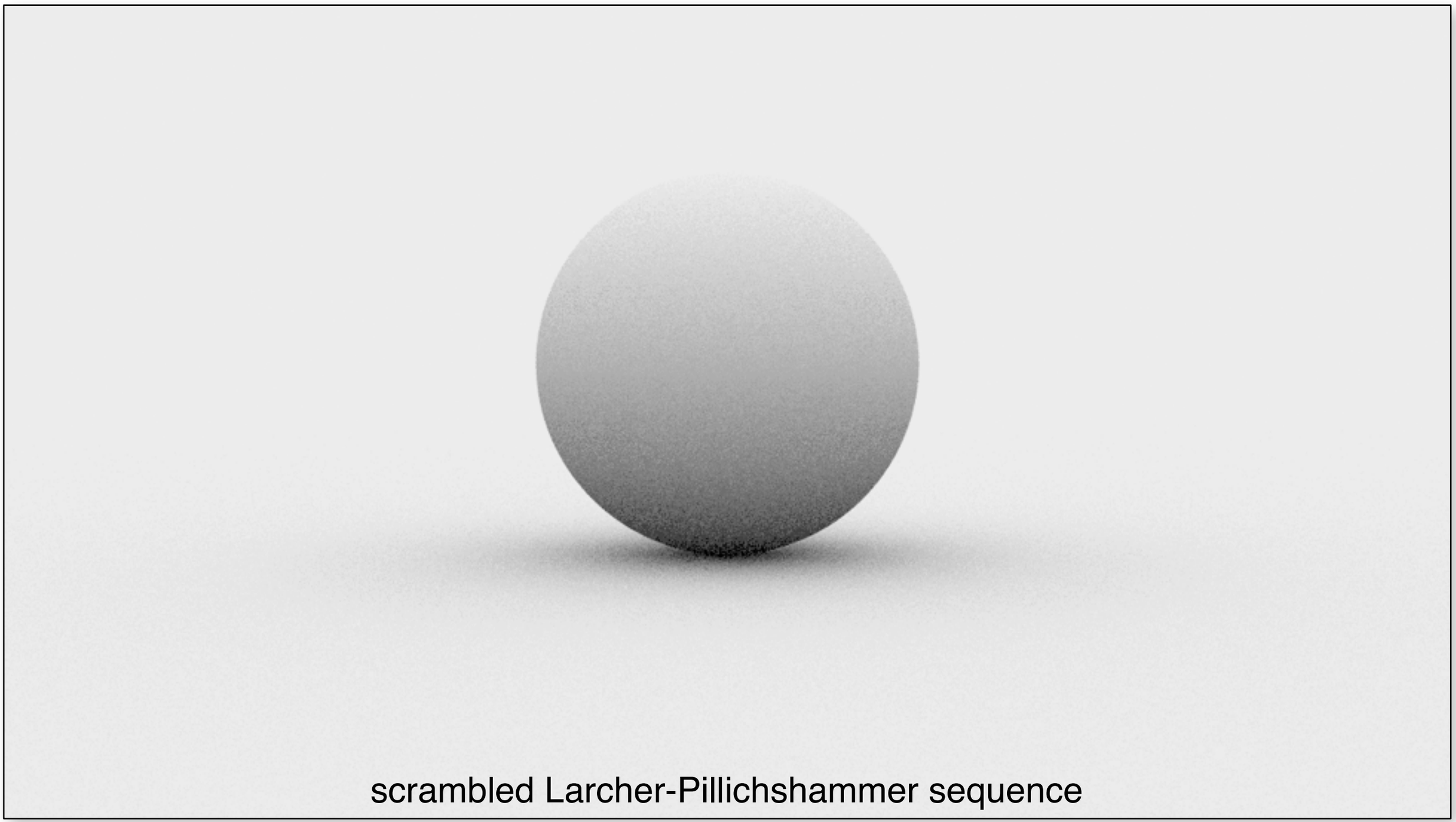
Monte Carlo (16 stratified samples)



Quasi-Monte Carlo (16 Halton samples)



Quasi-Monte Carlo



scrambled Larcher-Pillichshammer sequence

Implementation tips

- Using QMC can often lead to unintuitive, difficult to debug, problems.
  - Always code up MC algorithms first, using random numbers, to ensure correctness
  - Only after confirming correctness, slowly incorporate QMC into the mix

Questions?