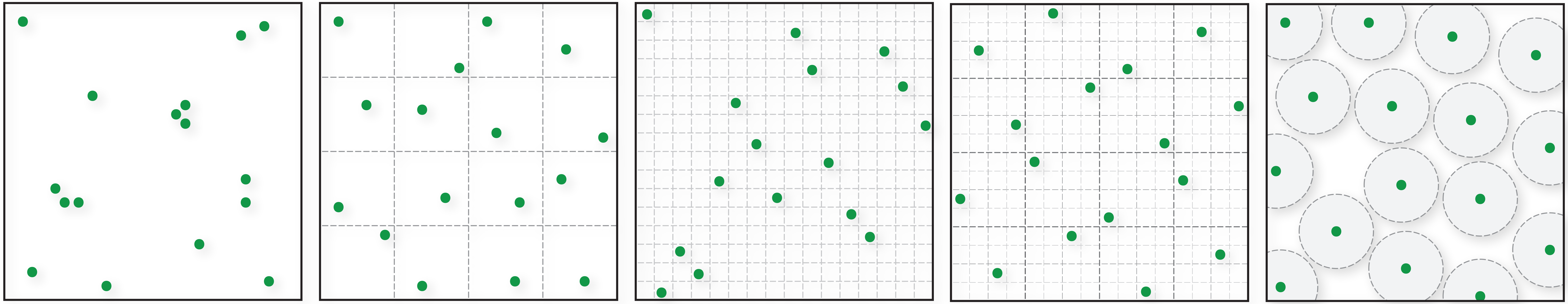
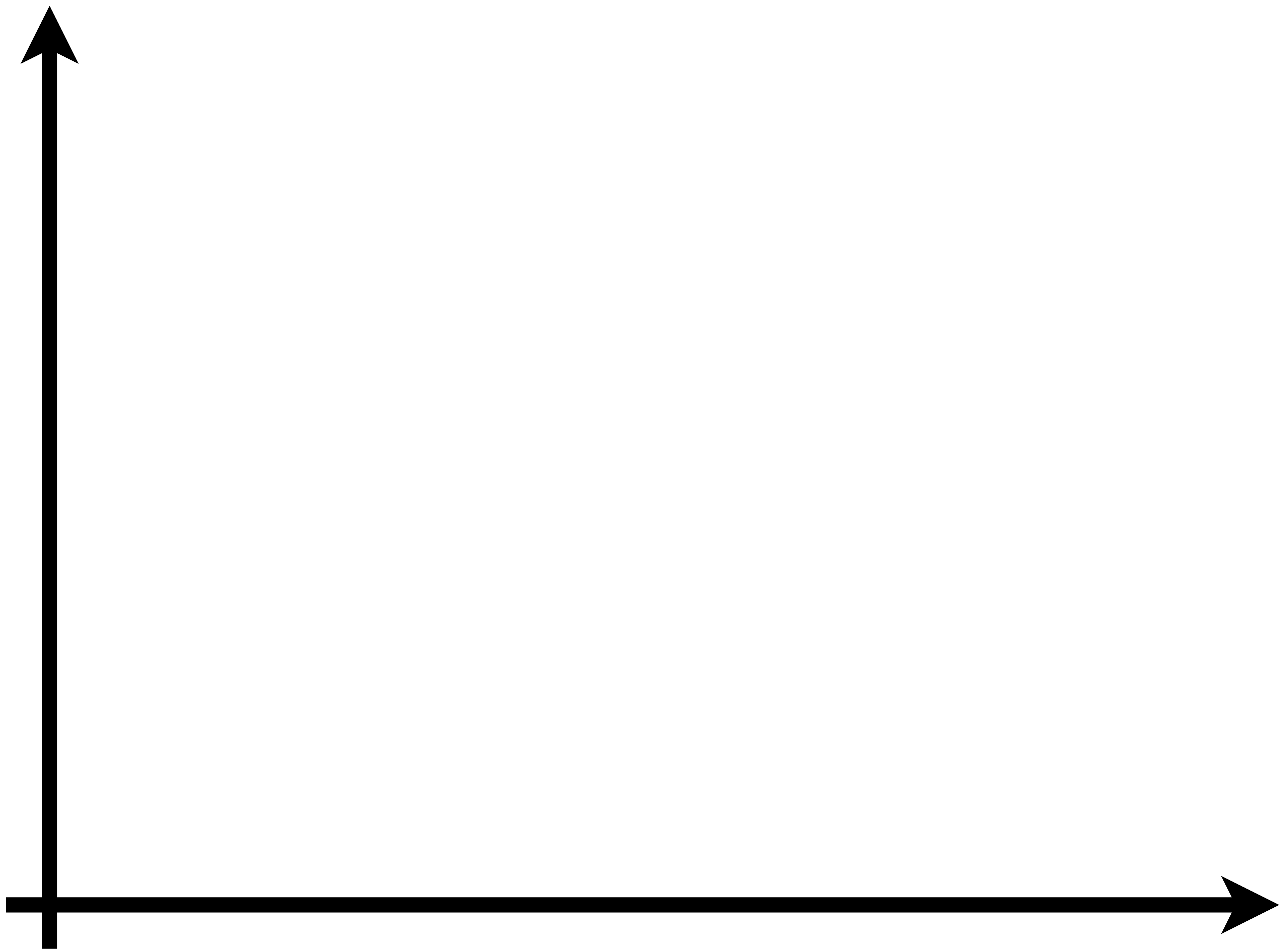
ADVANCED SAMPLING



Philipp Slusallek   *Karol Myszkowski*
Gurprit Singh

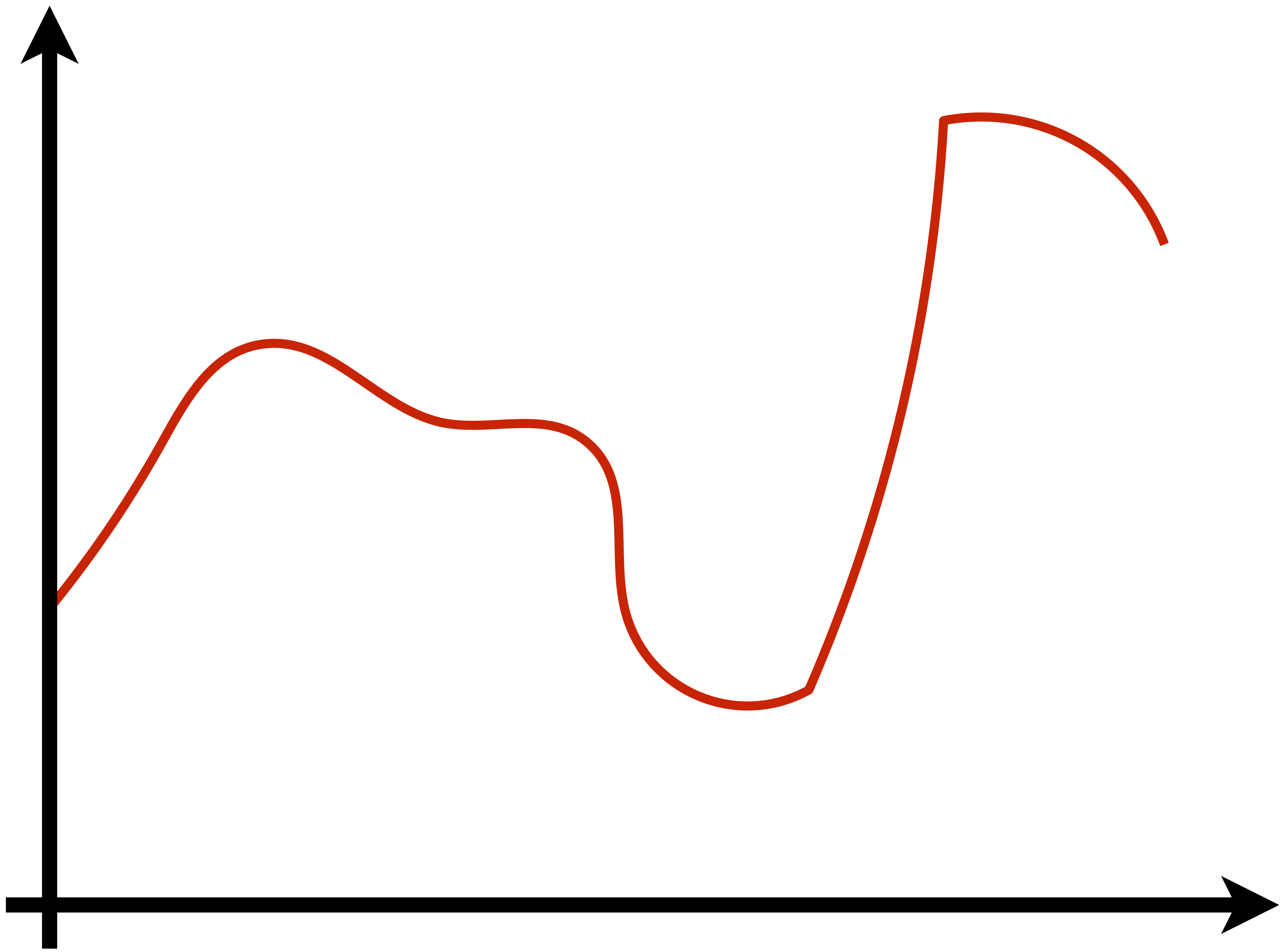
Recall: Monte Carlo Integration

$$I = \int_D f(x)\, \mathrm{d}x$$



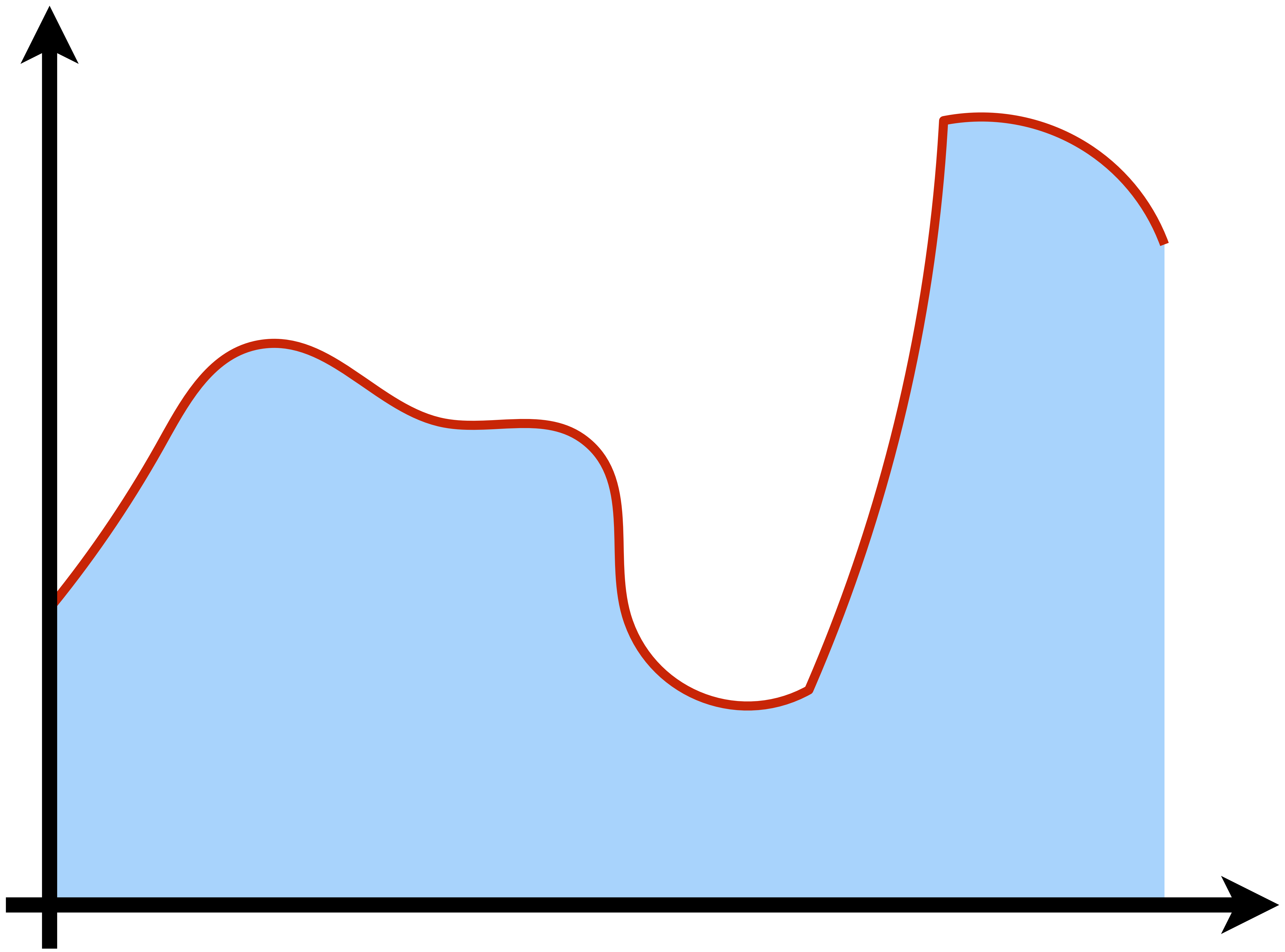
Recall: Monte Carlo Integration

$$I = \int_D f(x)\, \mathrm{d}x$$



Recall: Monte Carlo Integration

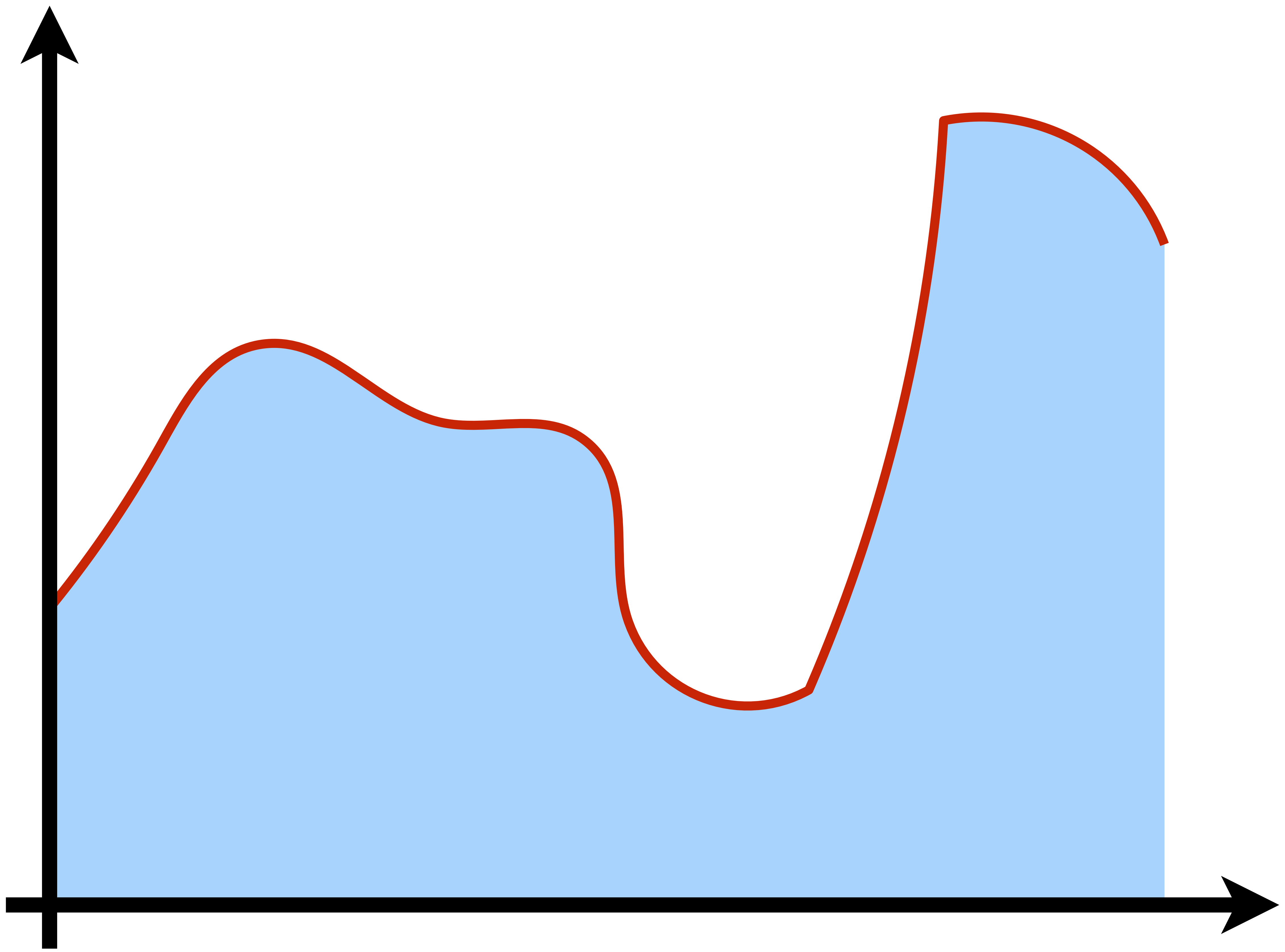
$$I = \int_D f(x)\,\mathrm{d}x$$



Recall: Monte Carlo Integration

$$I = \int_D f(x)\,\mathrm{d}x$$

$$\approx \int_D f(x)\,\mathbf{S}(x)\,\mathrm{d}x$$

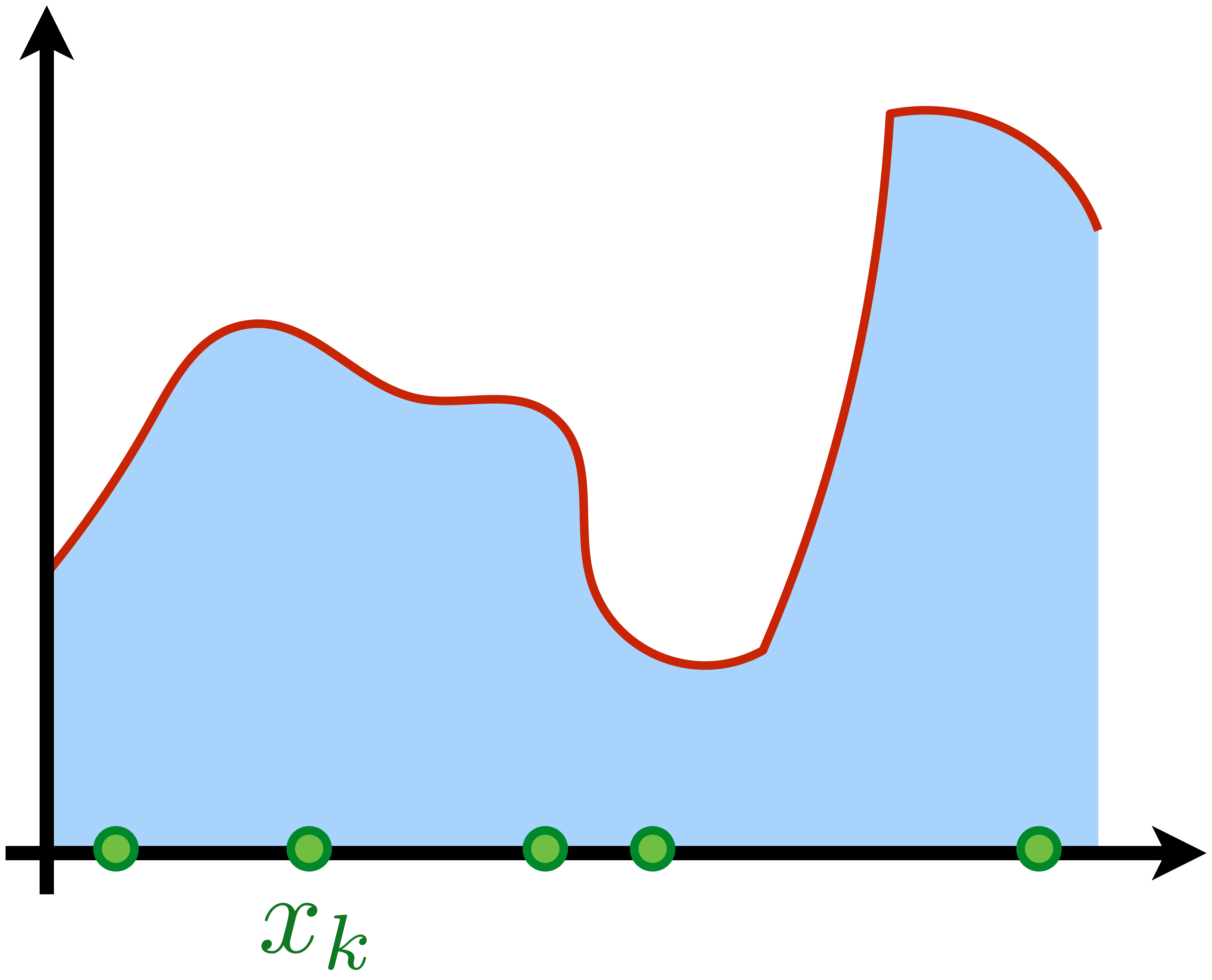


Recall: Monte Carlo Integration

$$I = \int_D f(x)\, \mathrm{d}x$$

$$\approx \int_D f(x)\, \mathbf{S}(x)\, \mathrm{d}x$$

$$\mathbf{S}(x) = \frac{1}{N} \sum_{k=1}^{N} \delta(x - x_k)$$

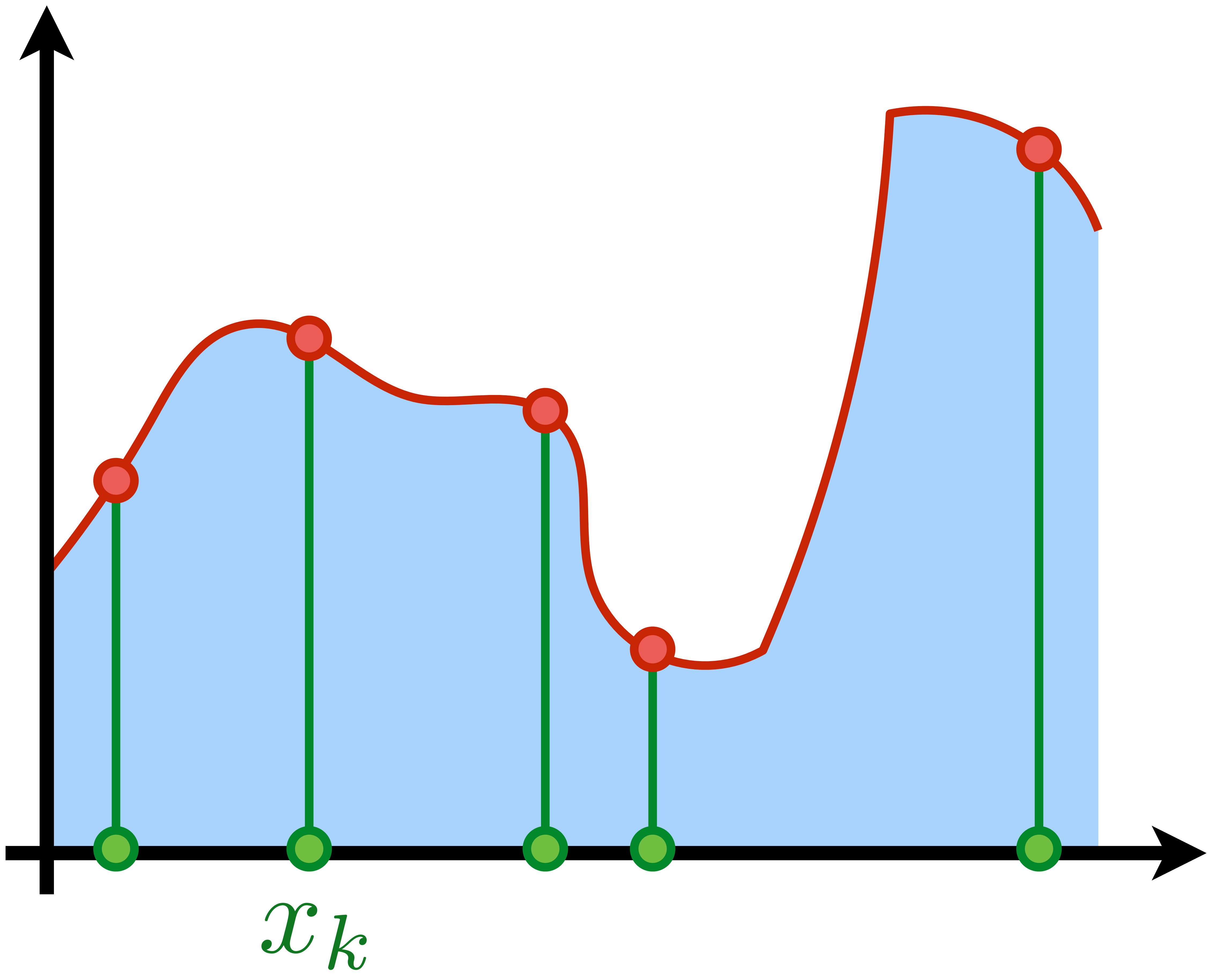


Recall: Monte Carlo Integration

$$I = \int_D f(x)\, \mathrm{d}x$$

$$\approx \int_D f(x)\, \mathbf{S}(x)\, \mathrm{d}x$$

$$\mathbf{S}(x) = \frac{1}{N} \sum_{k=1}^{N} \delta(x - x_k)$$



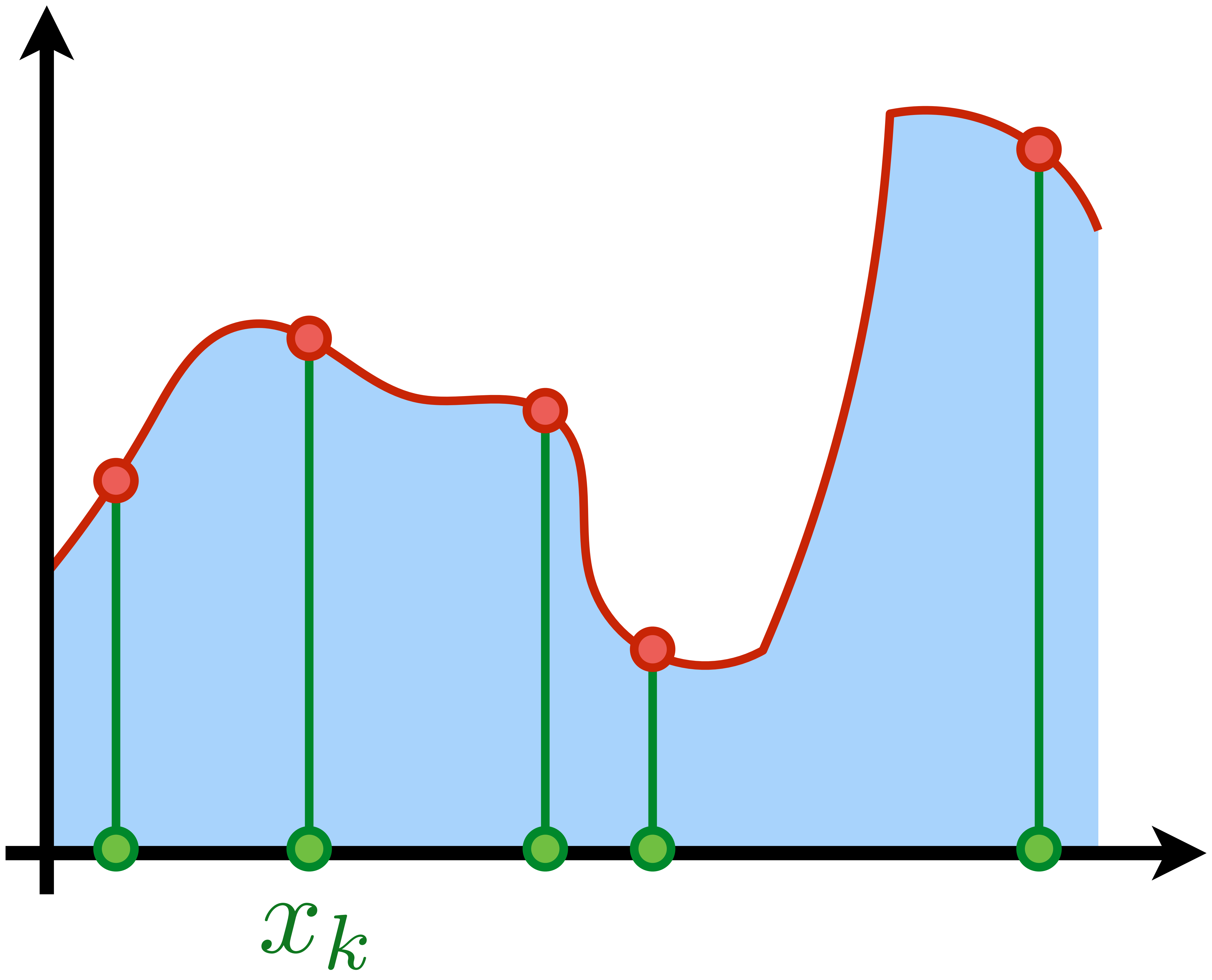
Recall: Monte Carlo Integration

$$I = \int_D f(x)\,\mathrm{d}x$$

$$\approx \int_D f(x)\,\mathbf{S}(x)\,\mathrm{d}x$$

$$\mathbf{S}(x) = \frac{1}{N}\sum_{k=1}^{N}\delta(x - x_k)$$

How to generate the locations x_k?

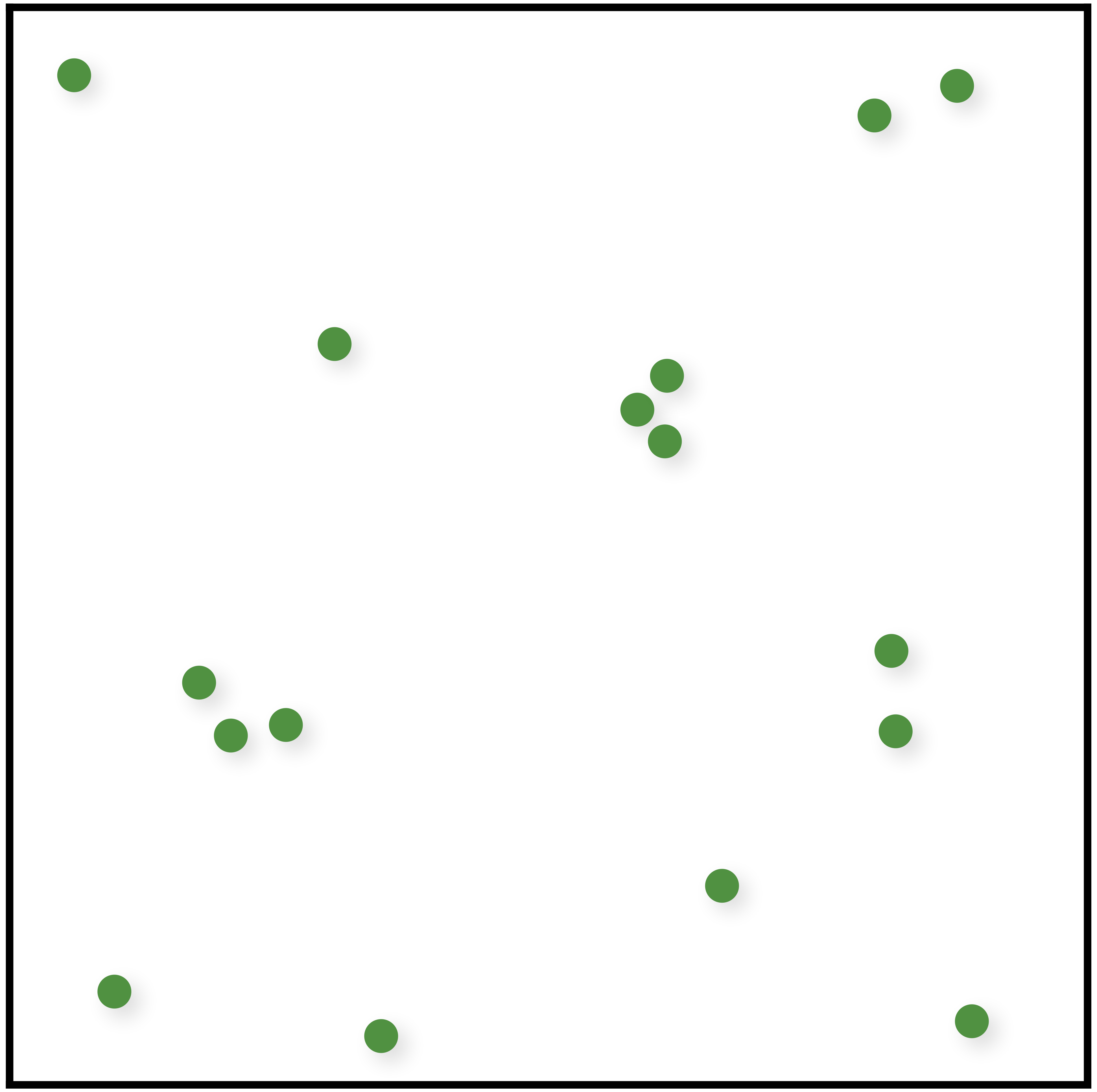


Independent Random Sampling

```
for (int k = 0; k < num; k++)
{
    samples(k).x = randf();
    samples(k).y = randf();
}
```

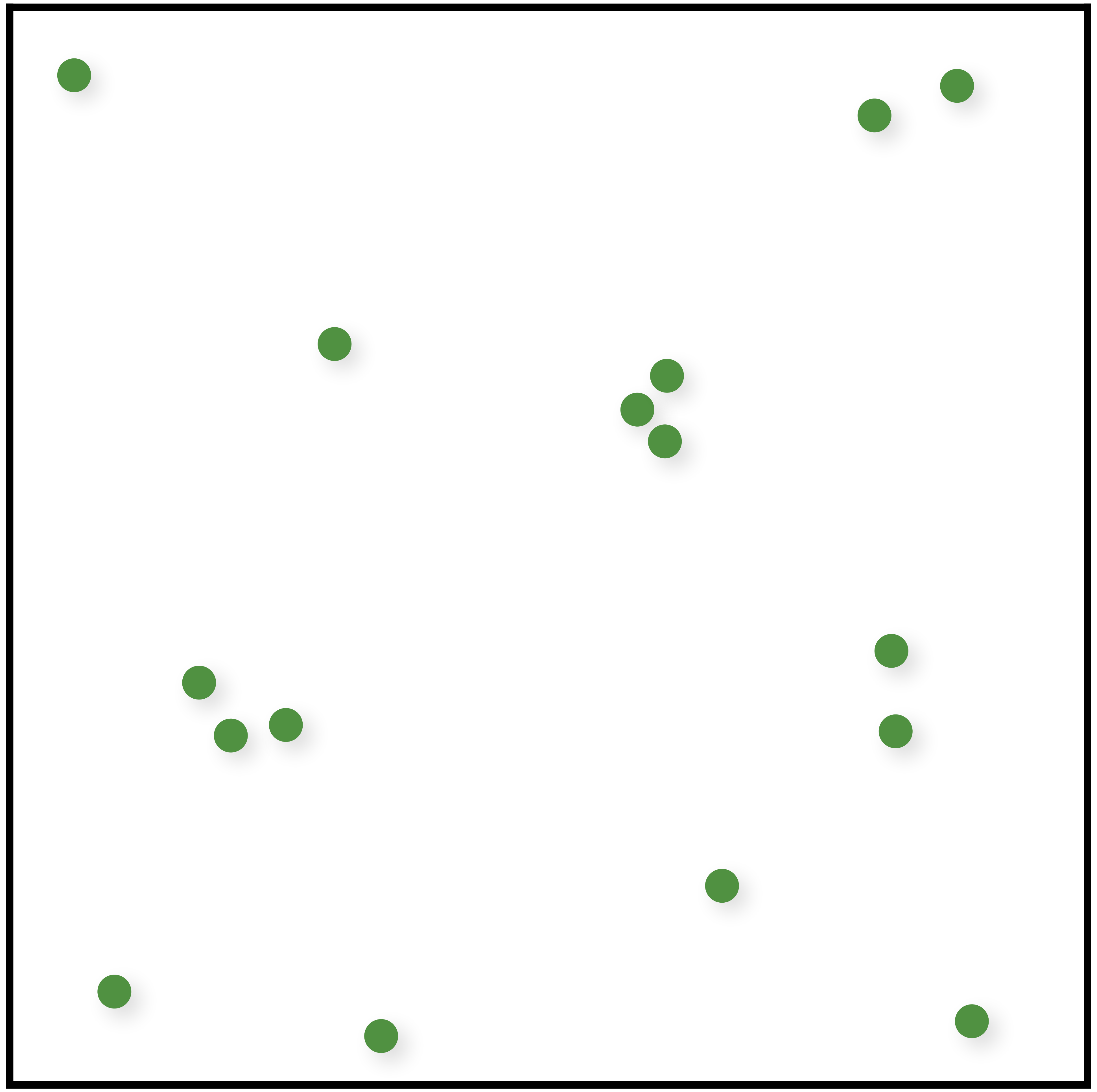
Independent Random Sampling

```
for (int k = 0; k < num; k++)
{
      samples(k).x = randf();
      samples(k).y = randf();
}
```



Independent Random Sampling

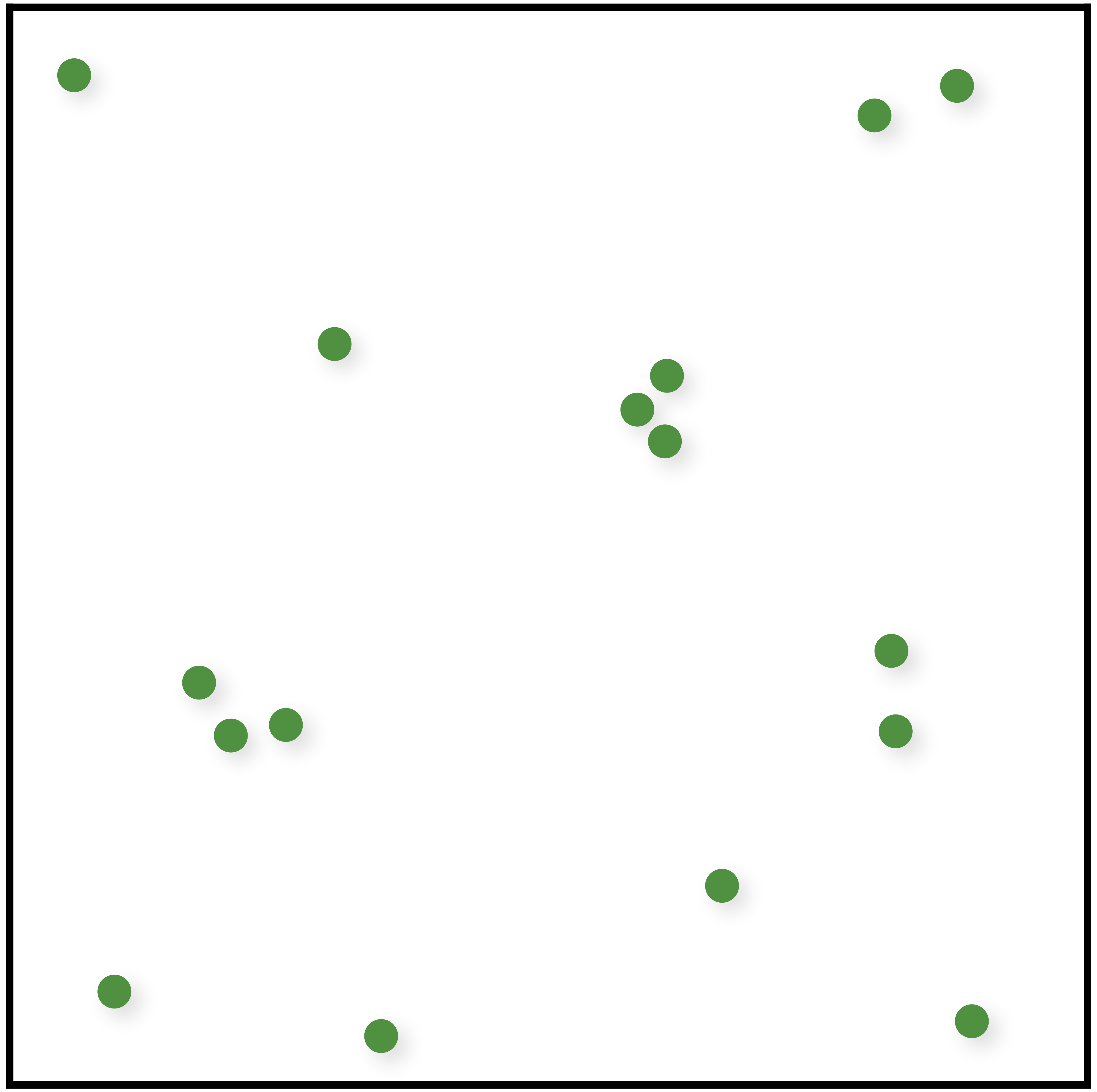
```
for (int k = 0; k < num; k++)
{
      samples(k).x = randf();
      samples(k).y = randf();
}
```



✔ Trivially extends to higher dimensions

Independent Random Sampling

```
for (int k = 0; k < num; k++)
{
    samples(k).x = randf();
    samples(k).y = randf();
}
```



✔ Trivially extends to higher dimensions

✔ Trivially progressive and memory-less

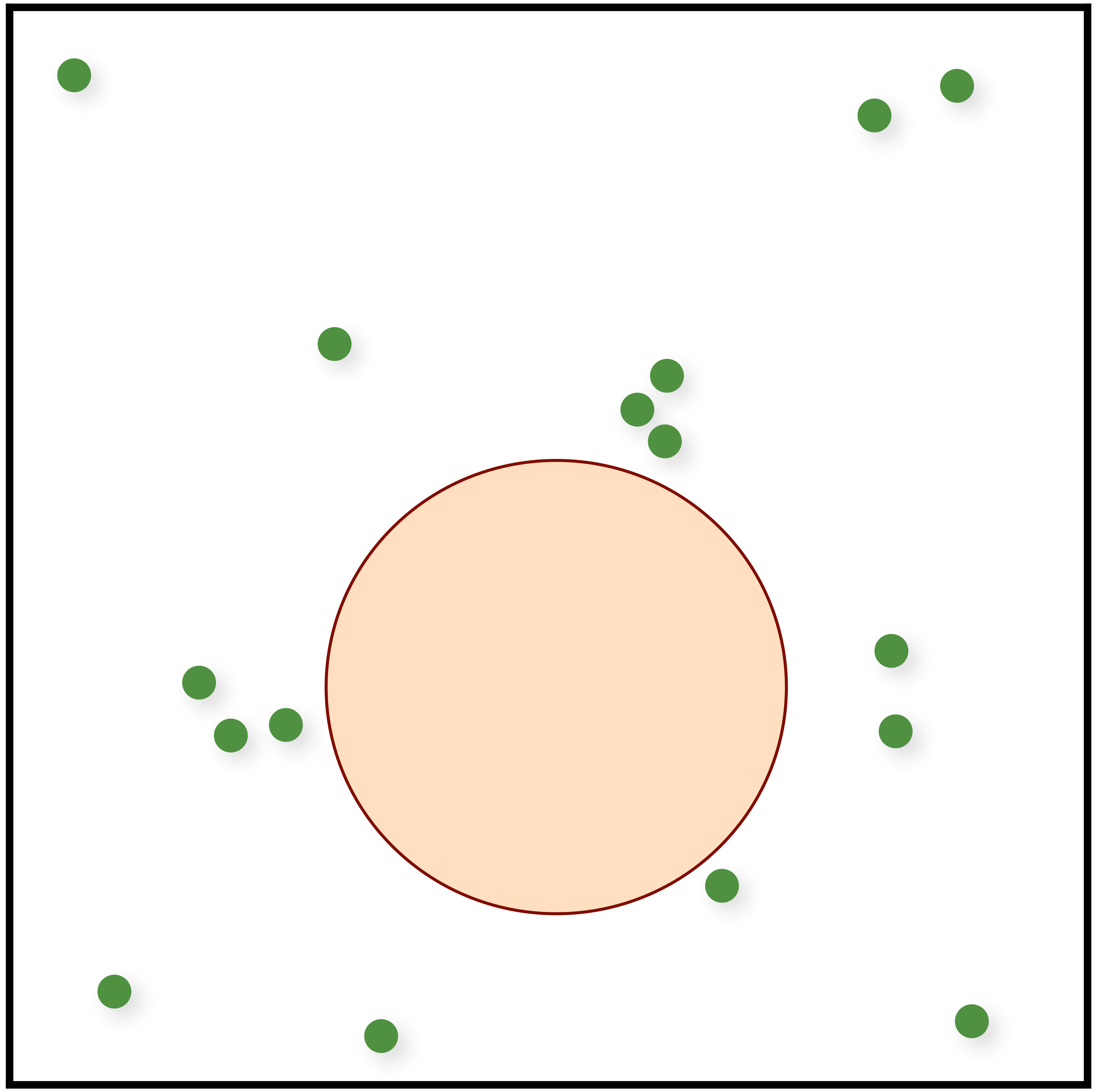
Independent Random Sampling

```
for (int k = 0; k < num; k++)
{
        samples(k).x = randf();
        samples(k).y = randf();
}
```

✔ Trivially extends to higher dimensions

✔ Trivially progressive and memory-less

✘ Big gaps



Independent Random Sampling

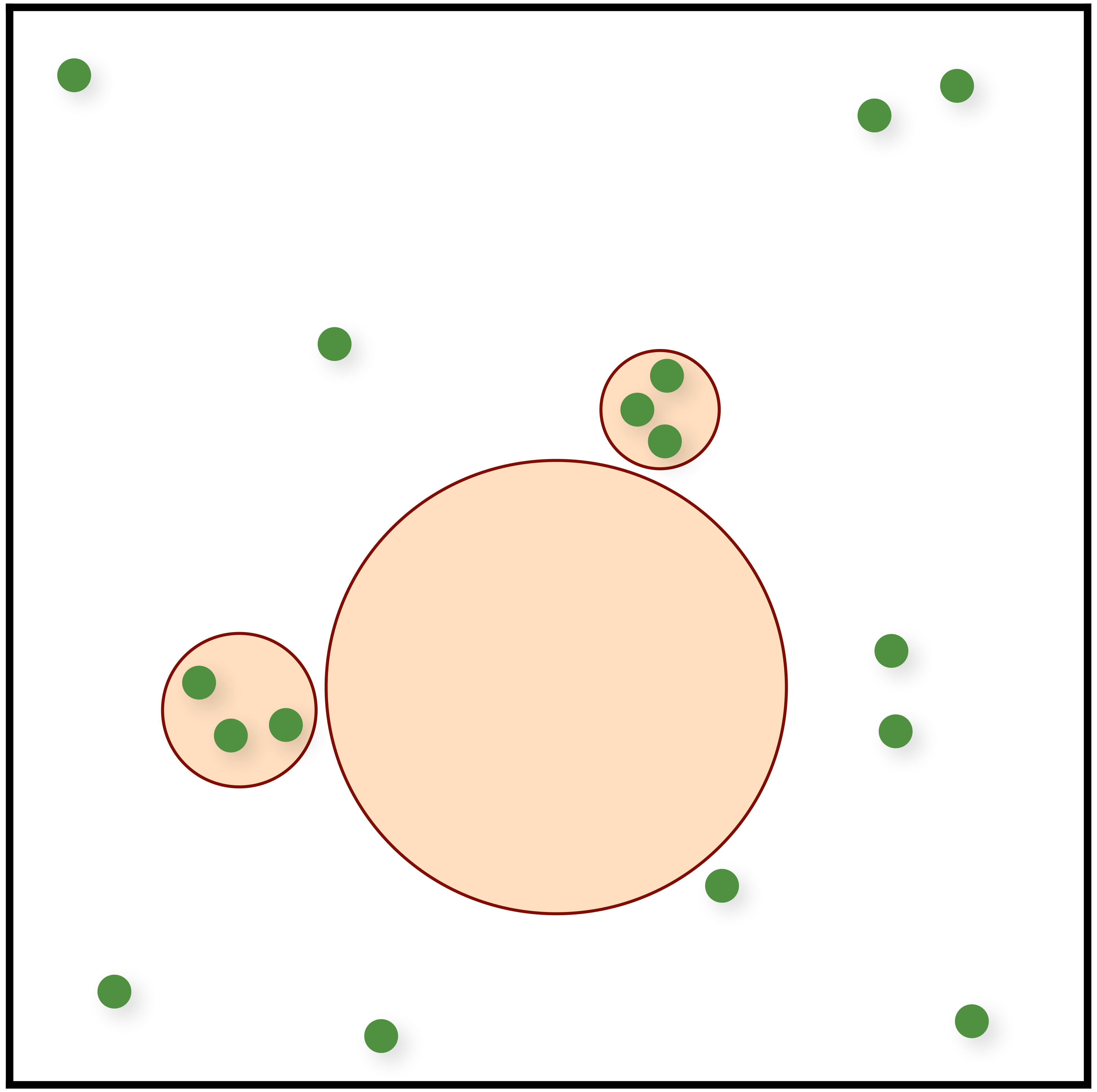
```
for (int k = 0; k < num; k++)
{
    samples(k).x = randf();
    samples(k).y = randf();
}
```

✔ Trivially extends to higher dimensions

✔ Trivially progressive and memory-less

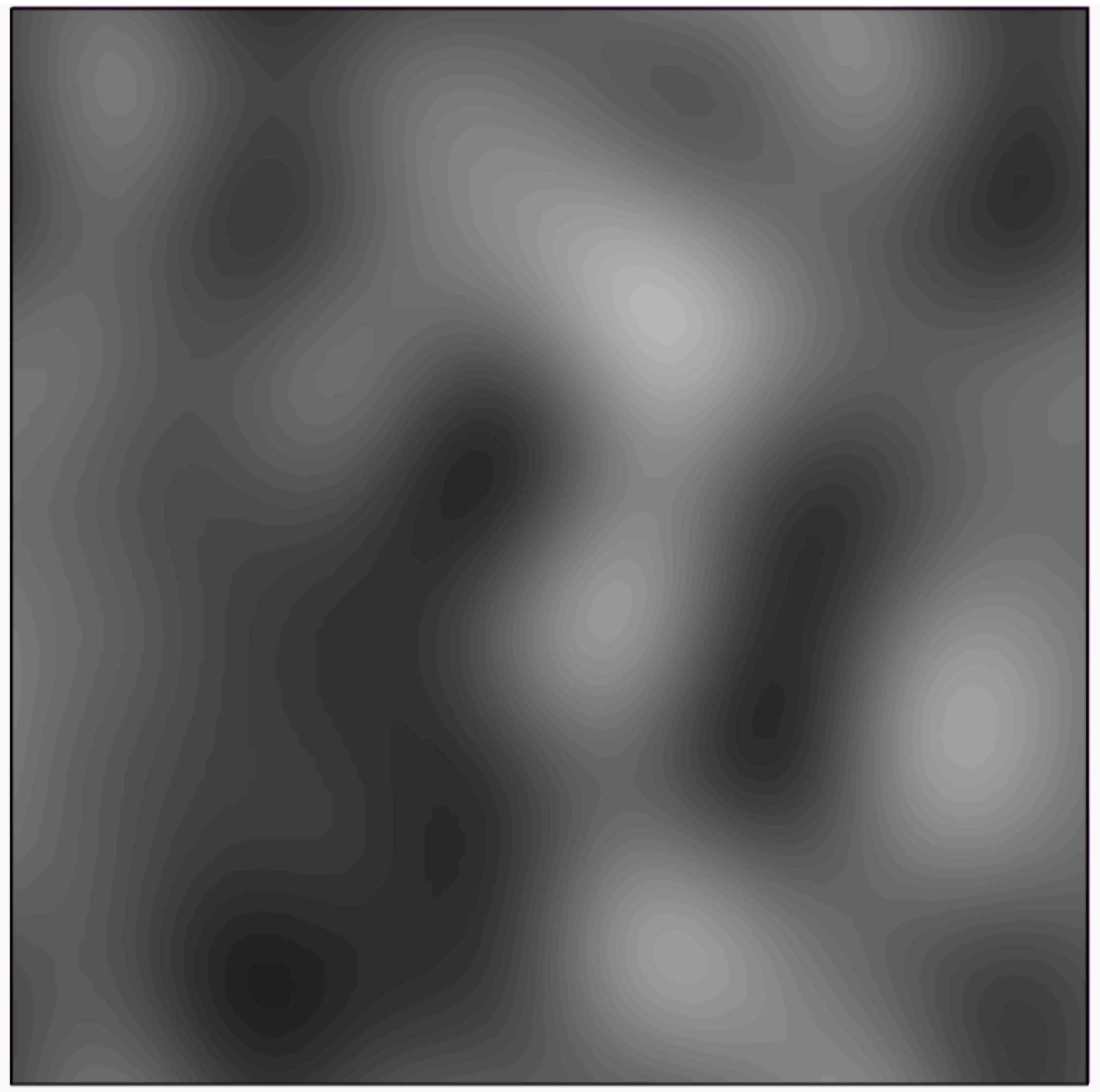
✘ Big gaps

✘ Clumping



Recall: Fourier Theory

Input Image



Power Spectrum

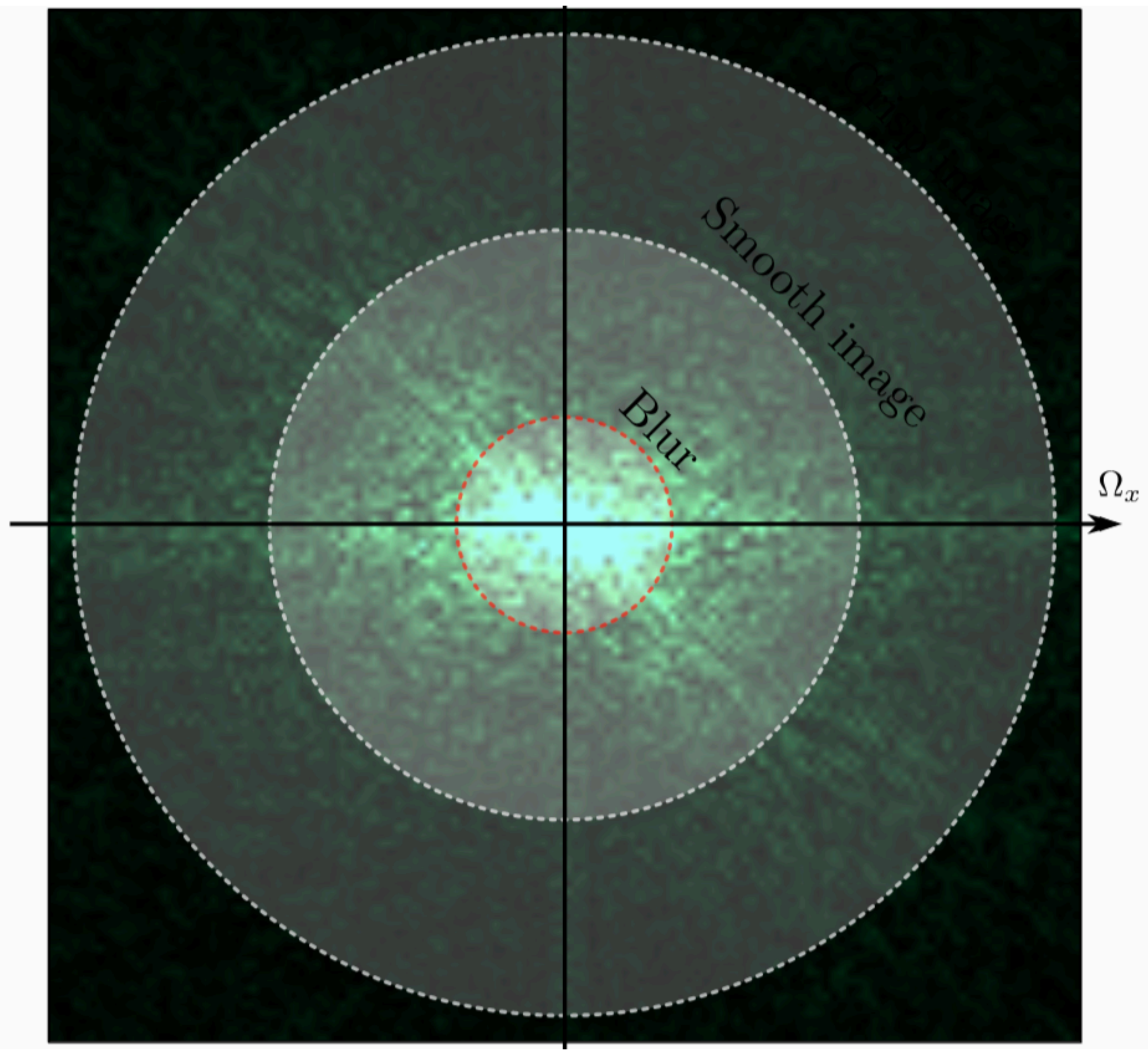


Image courtesy: Laurent Belcour

Recall: Fourier Theory

Input Image | Power Spectrum

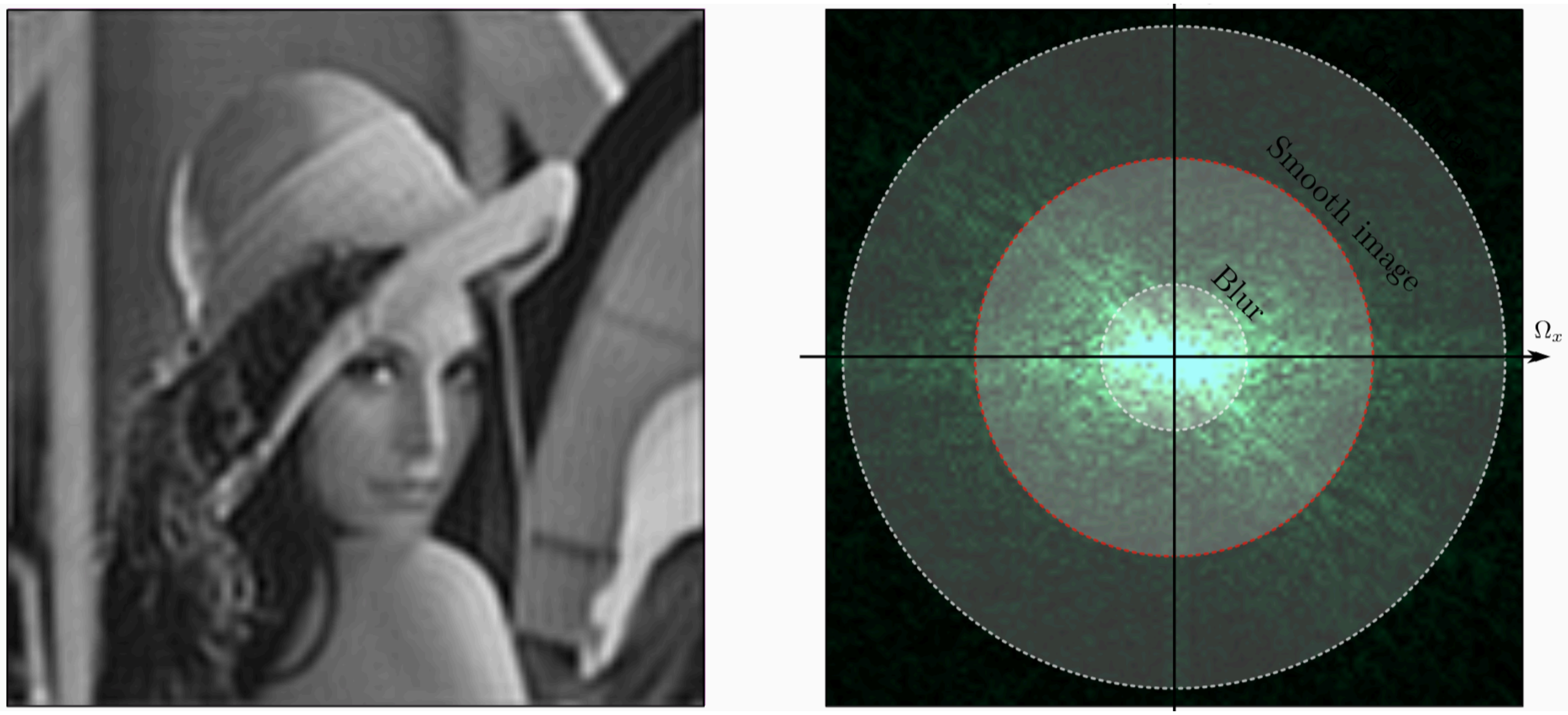


Image courtesy: Laurent Belcour

Recall: Fourier theory

Fourier transform: $\hat{f}(\omega) = \int_D f(x)\, \mathrm{e}^{-2\pi \imath \omega x}\, \mathrm{d}x$

Recall: Fourier theory

Fourier transform: $$\hat{f}(\vec{\omega}) = \int_D f(\vec{x})\, \mathrm{e}^{-2\pi\imath(\vec{\omega}\cdot\vec{x})}\, \mathrm{d}\vec{x}$$

Recall: Fourier theory

Fourier transform: $$\hat{f}(\vec{\omega}) = \int_D f(\vec{x})\, \mathrm{e}^{-2\,\pi\,\imath\,(\vec{\omega}\cdot\vec{x})}\; \mathrm{d}\vec{x}$$

Sampling function: $$\hat{\mathbf{S}}(\vec{\omega}) = \int_D \mathbf{S}(\vec{x})\, \mathrm{e}^{-2\,\pi\,\imath\,(\vec{\omega}\cdot\vec{x})}\; \mathrm{d}\vec{x}$$

Recall: Fourier theory

Fourier transform: $\hat{f}(\vec{\omega}) = \int_D f(\vec{x})\, \mathrm{e}^{-2\,\pi\,\imath\,(\vec{\omega}\cdot\vec{x})}\, \mathrm{d}\vec{x}$

Sampling function: $\hat{\mathbf{S}}(\vec{\omega}) = \int_D \frac{1}{N} \sum_{k=1}^{N} \delta(|\vec{x} - \vec{x}_k|)\, \mathrm{e}^{-2\,\pi\,\imath\,(\vec{\omega}\cdot\vec{x})}\, \mathrm{d}\vec{x}$

Recall: Fourier theory

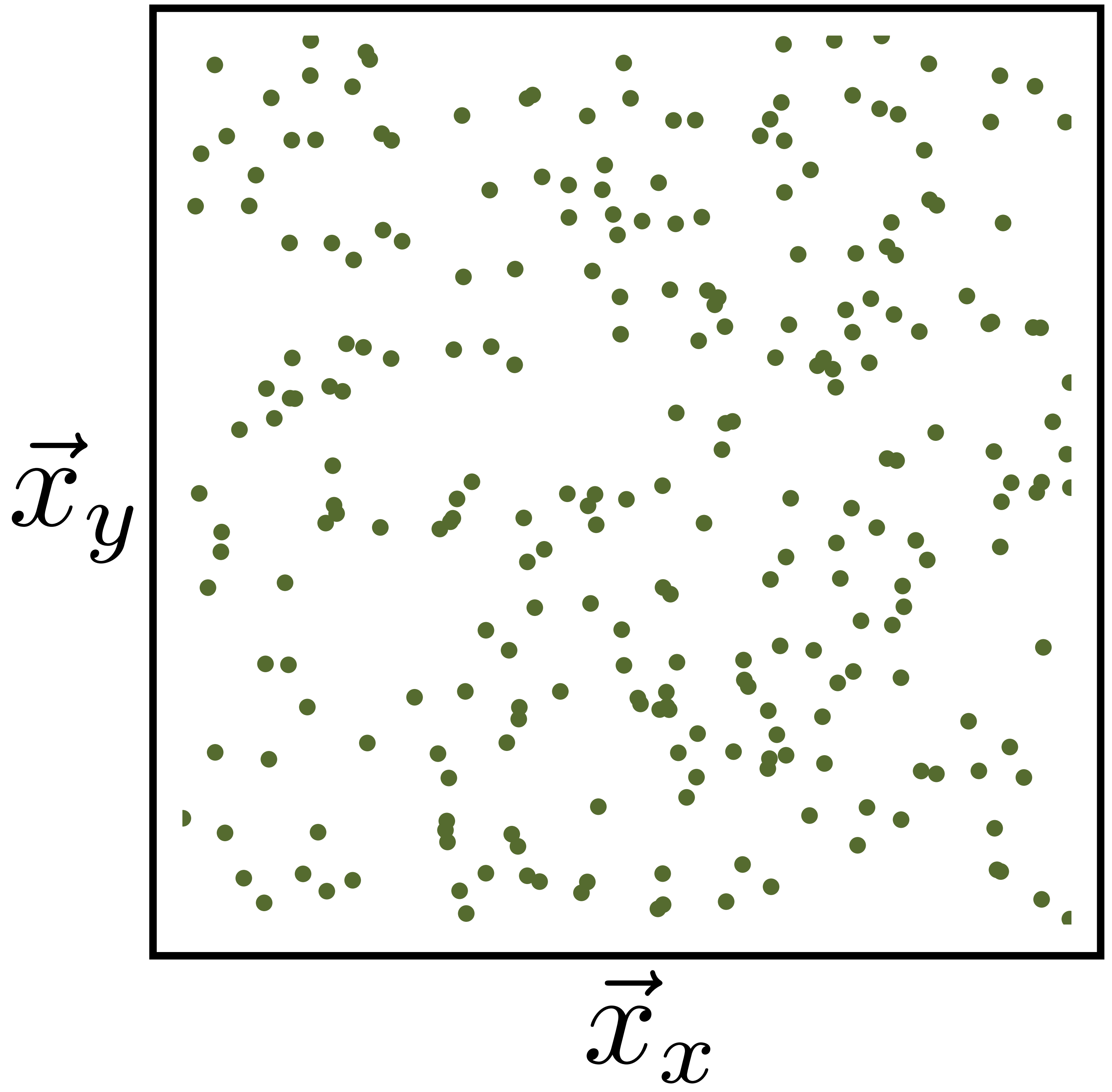
Fourier transform: $$\hat{f}(\vec{\omega}) = \int_D f(\vec{x})\, \mathrm{e}^{-2\,\pi\,\imath\,(\vec{\omega}\cdot\vec{x})}\, \mathrm{d}\vec{x}$$

Sampling function: $$\hat{\mathbf{S}}(\vec{\omega}) = \int_D \frac{1}{N} \sum_{k=1}^{N} \delta(|\vec{x} - \vec{x}_k|)\, \mathrm{e}^{-2\,\pi\,\imath\,(\vec{\omega}\cdot\vec{x})}\, \mathrm{d}\vec{x}$$

$$= \frac{1}{N} \sum_{k=1}^{N} \mathrm{e}^{-2\,\pi\,\imath\,(\vec{\omega}\cdot\vec{x}_k)}$$

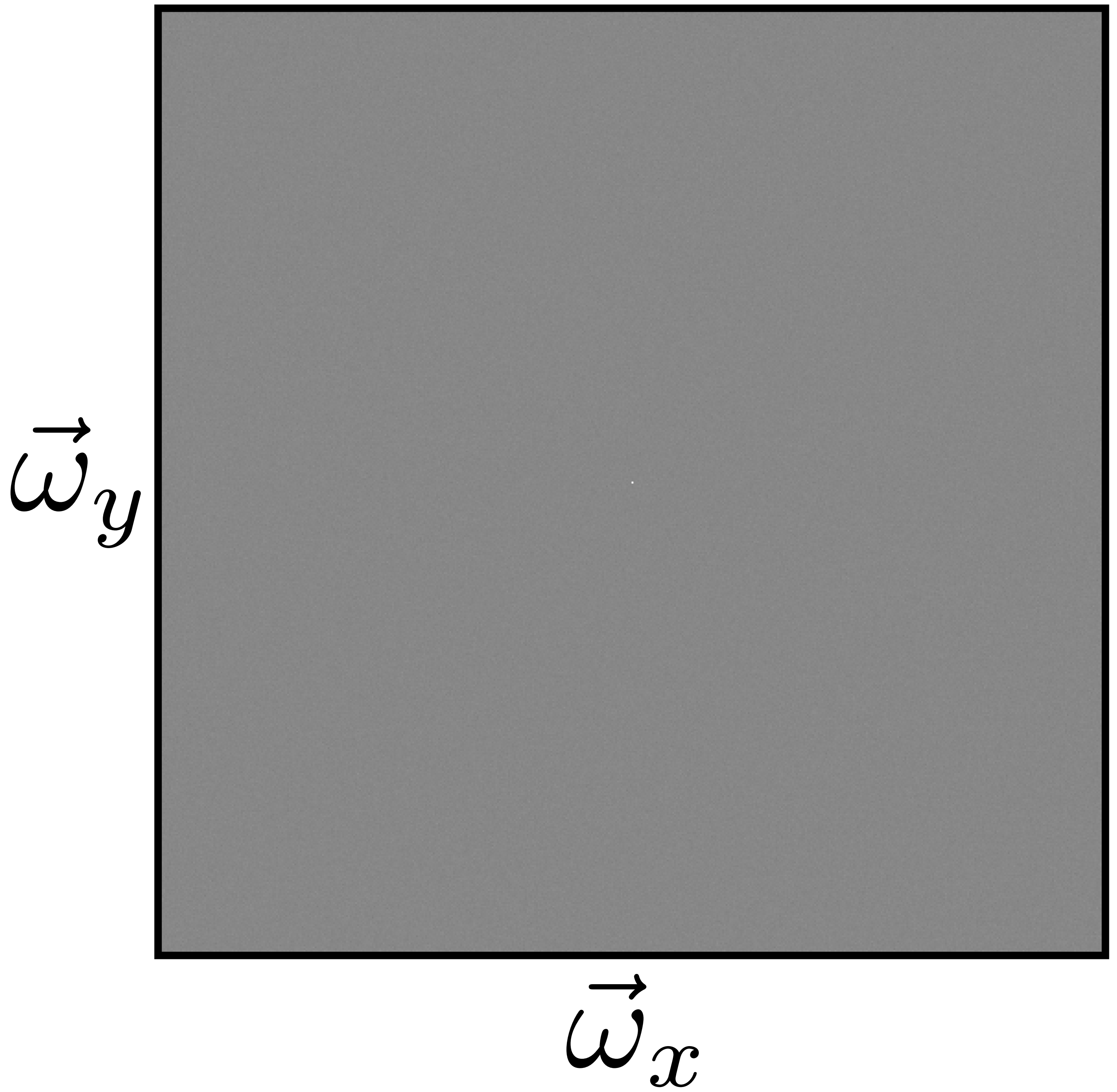
Independent Random Sampling

Samples



$$\frac{1}{N}\sum_{k=1}^{N}\delta(|\vec{x}-\vec{x}_k|)$$

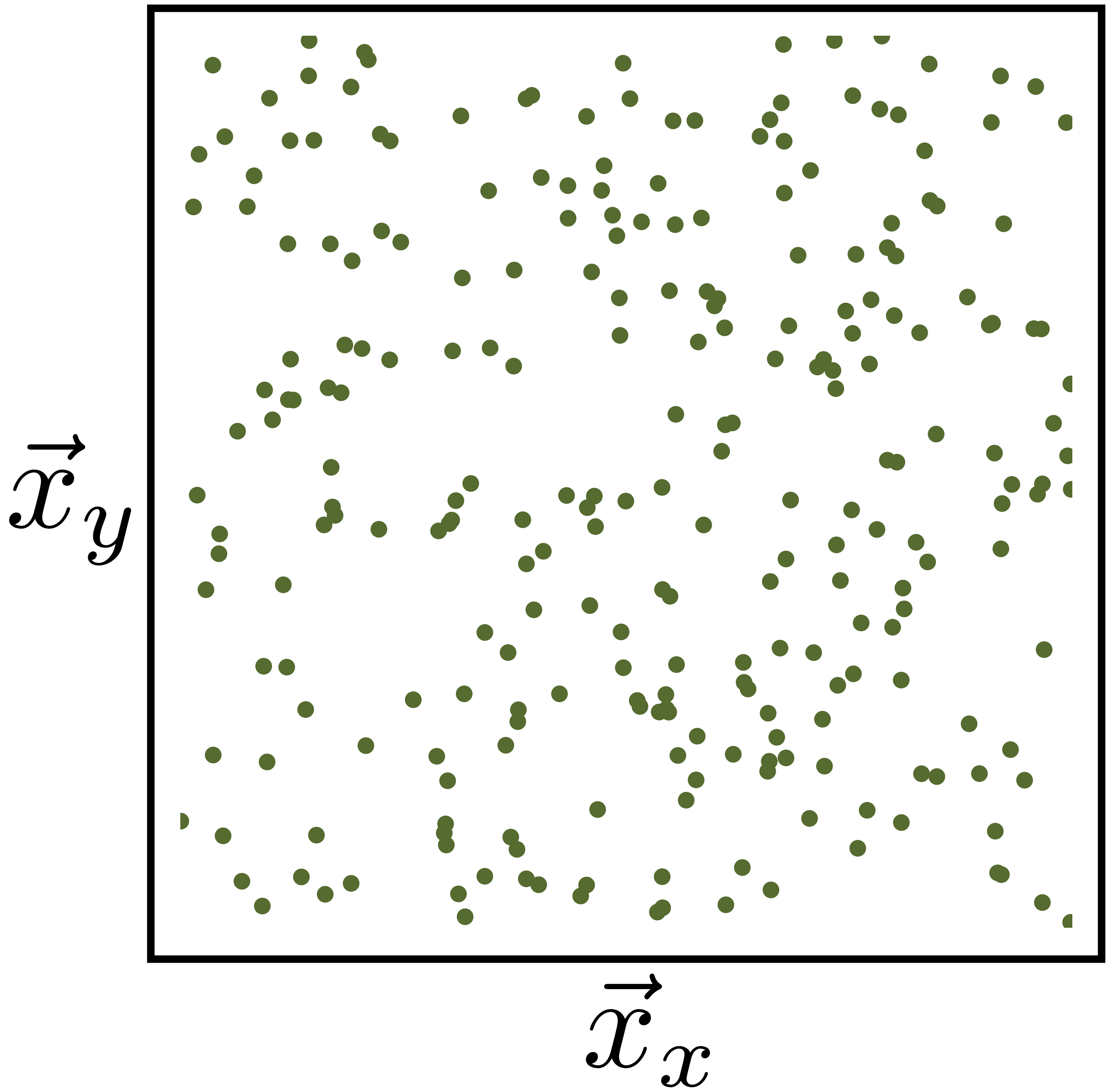
Power spectrum



$$\left|\frac{1}{N}\sum_{k=1}^{N}\mathrm{e}^{-2\,\pi\,\imath\,(\vec{\omega}\cdot\vec{x}_k)}\right|^2$$

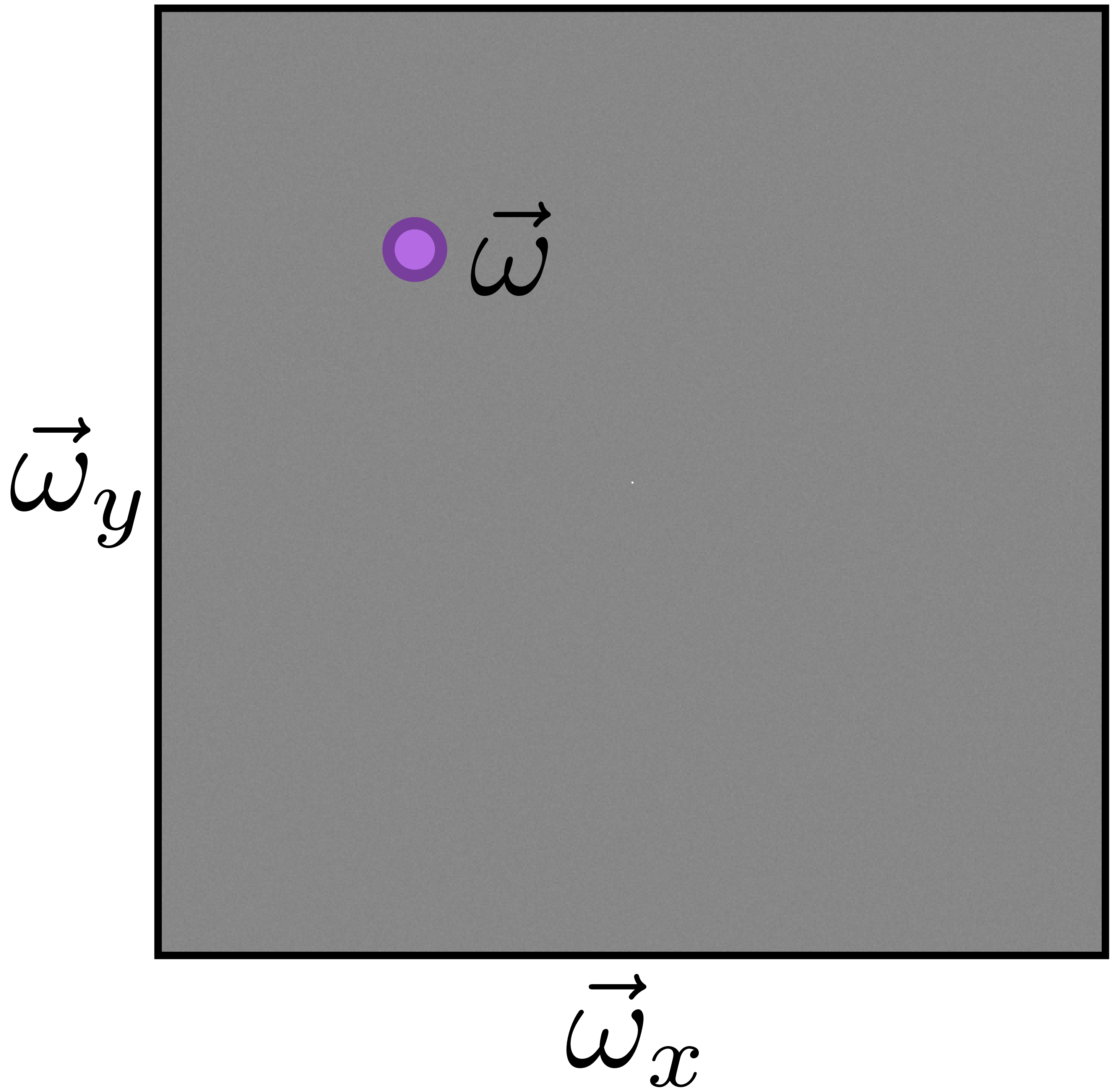
Independent Random Sampling

Samples



$$\frac{1}{N}\sum_{k=1}^{N}\delta(|\vec{x}-\vec{x}_k|)$$

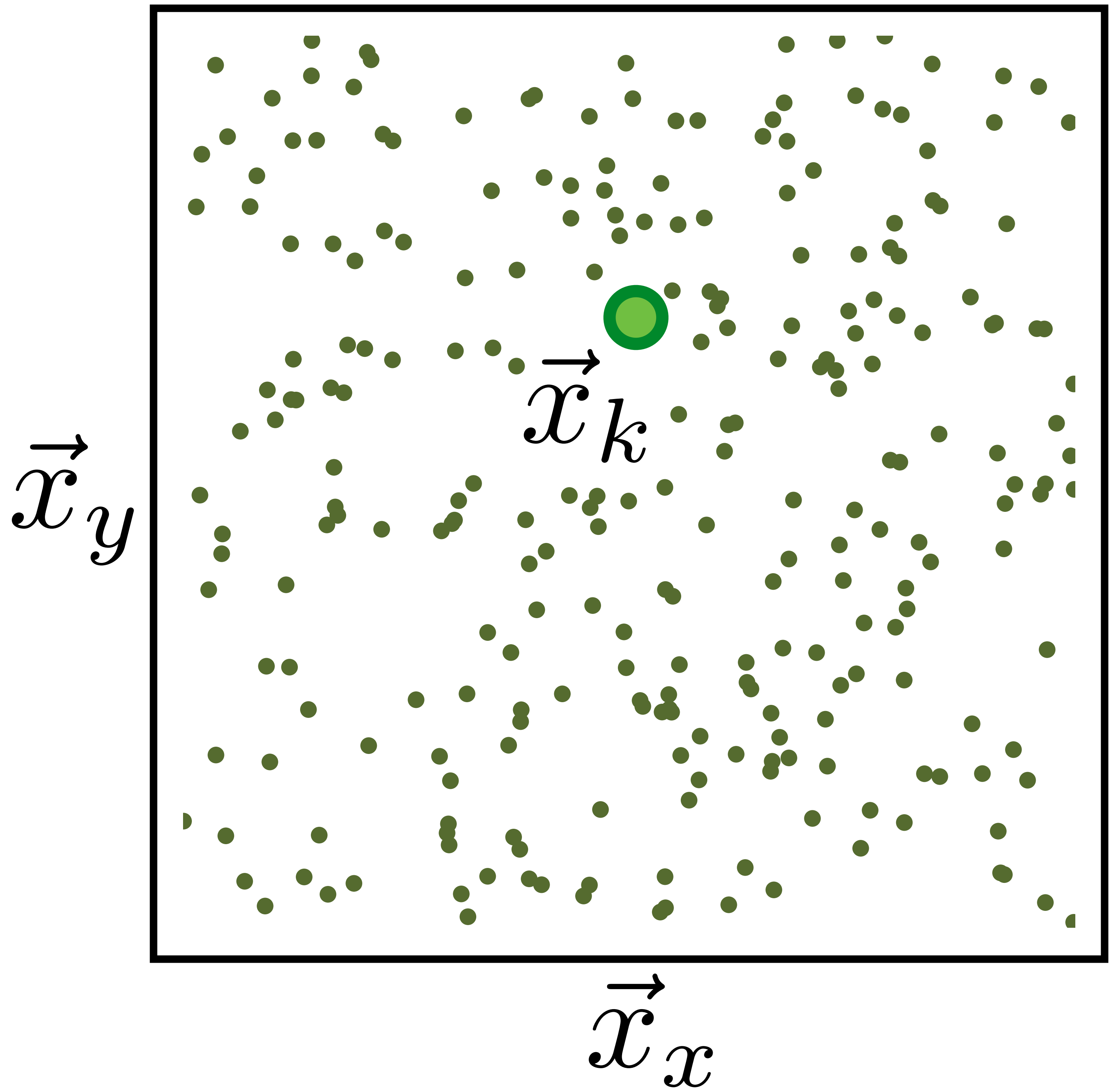
Power spectrum



$$\left|\frac{1}{N}\sum_{k=1}^{N}\mathrm{e}^{-2\pi\imath(\vec{\omega}\cdot\vec{x}_k)}\right|^2$$

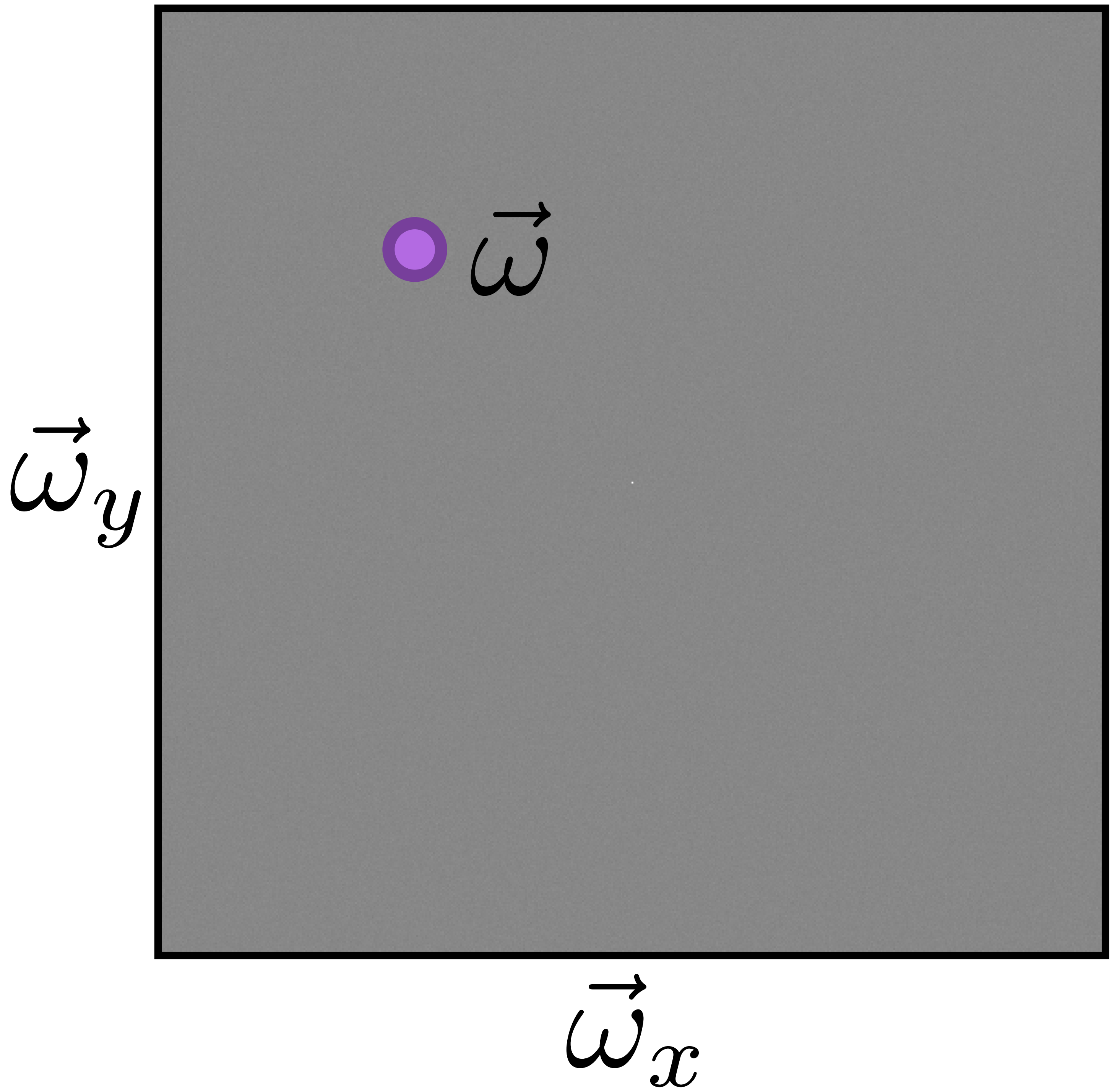
Independent Random Sampling

Samples



$$\frac{1}{N}\sum_{k=1}^{N}\delta(|\vec{x}-\vec{x}_k|)$$

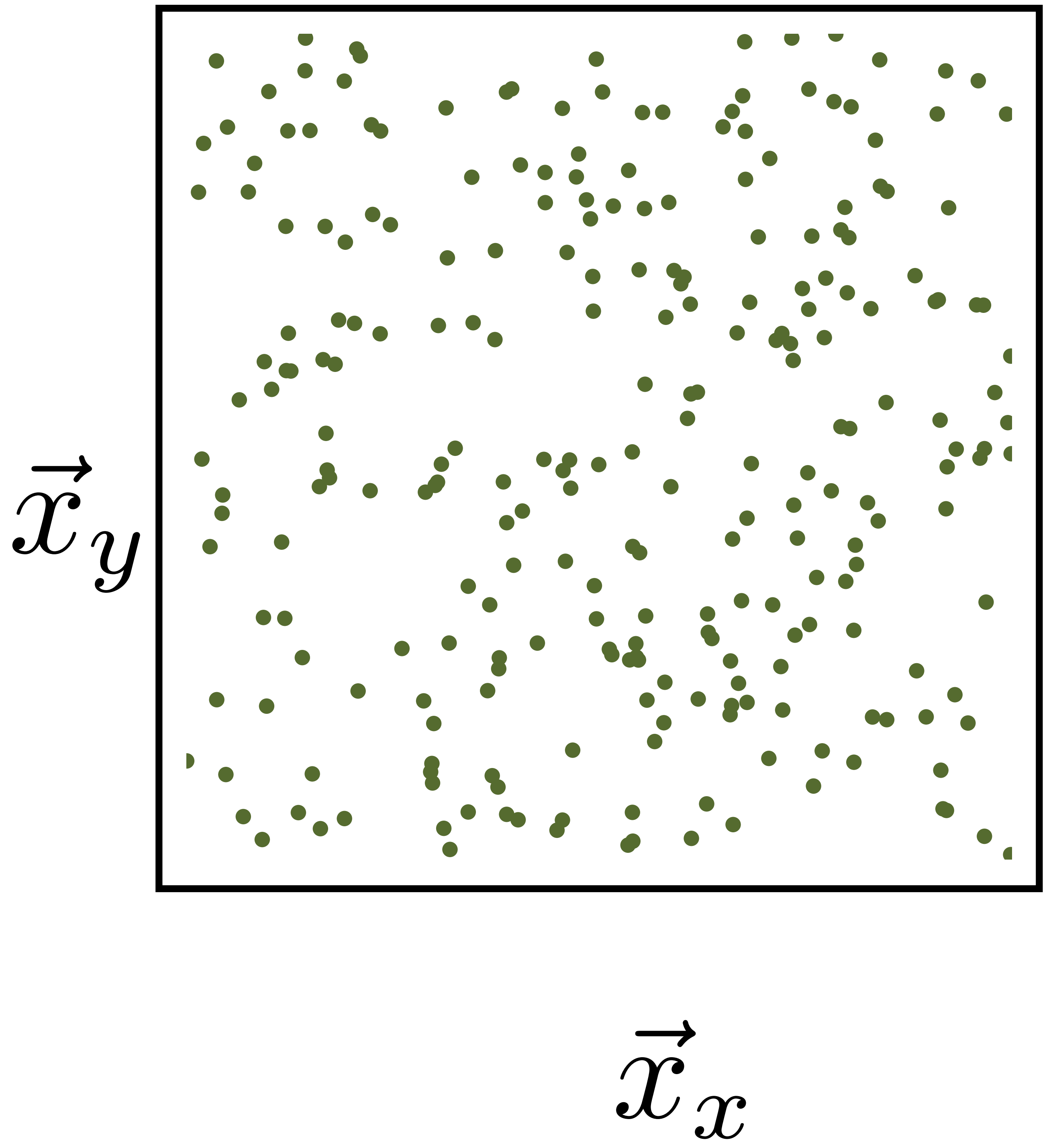
Power spectrum



$$\left|\frac{1}{N}\sum_{k=1}^{N}\mathrm{e}^{-2\pi\imath(\vec{\omega}\cdot\vec{x}_k)}\right|^2$$

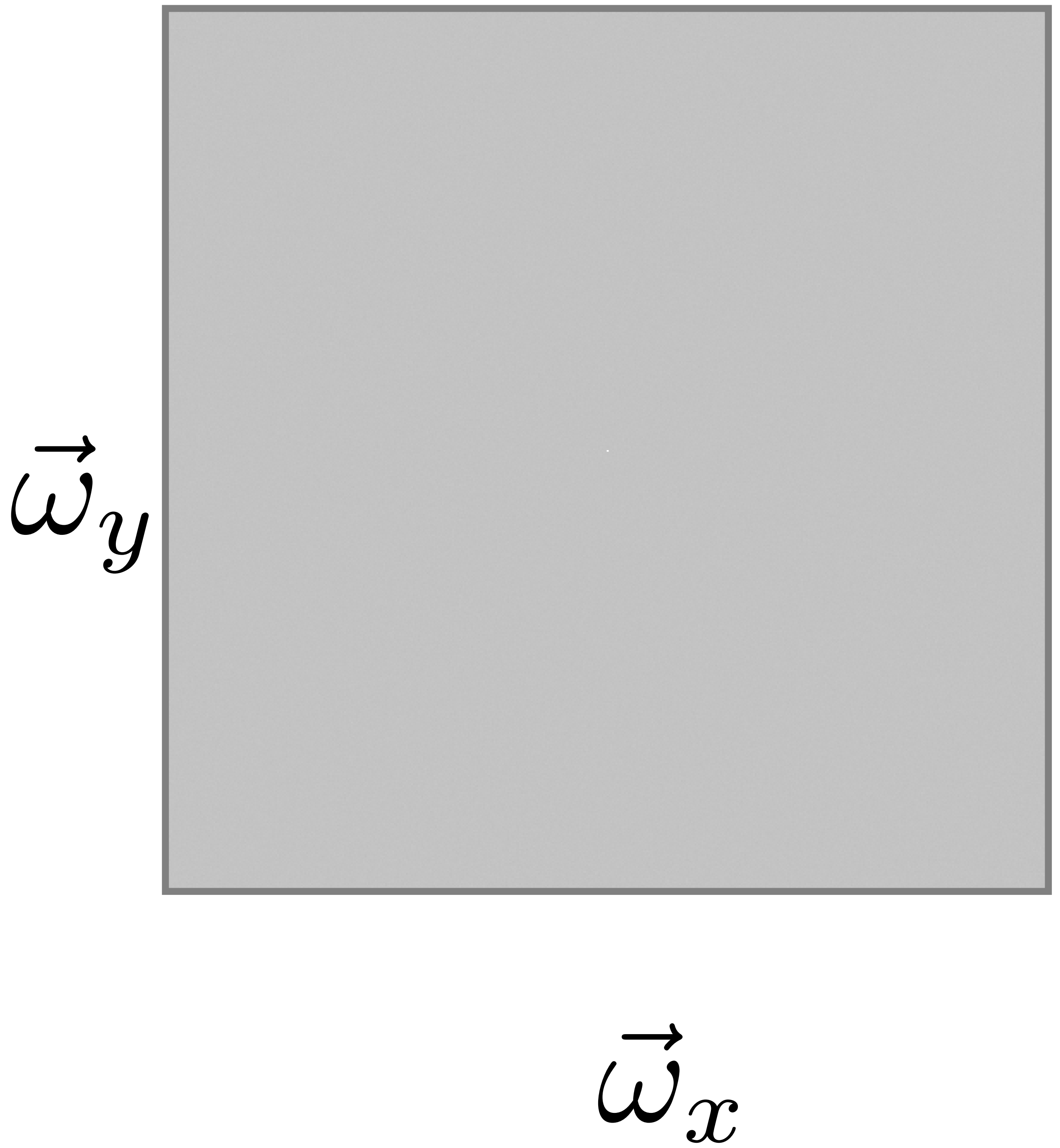
Independent Random Sampling

Many sample set realizations



$$\frac{1}{N}\sum_{k=1}^{N}\delta(|\vec{x}-\vec{x}_k|)$$

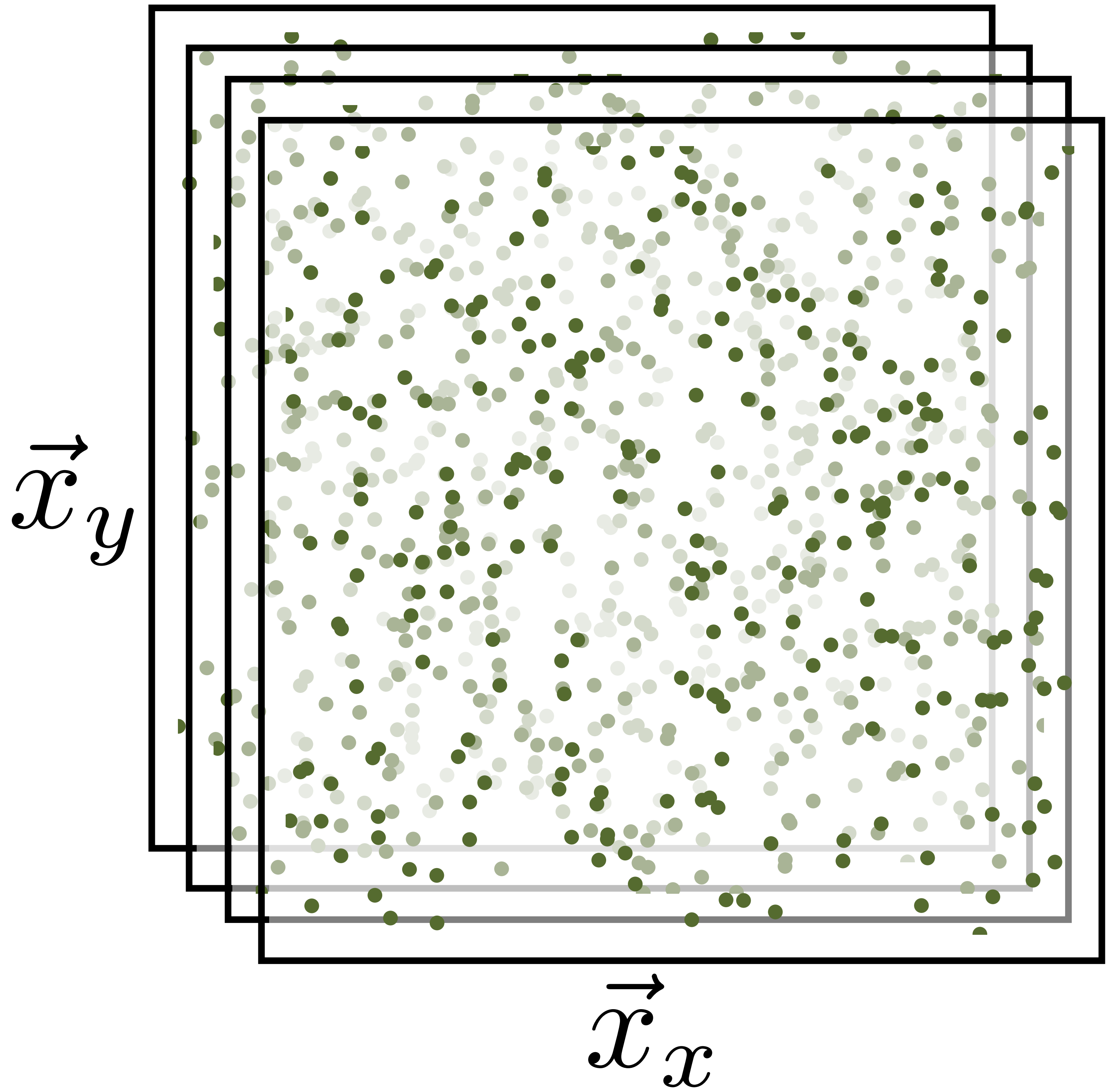
Expected power spectrum



$$\left|\frac{1}{N}\sum_{k=1}^{N}\mathrm{e}^{-2\pi\imath(\vec{\omega}\cdot\vec{x}_k)}\right|^2$$

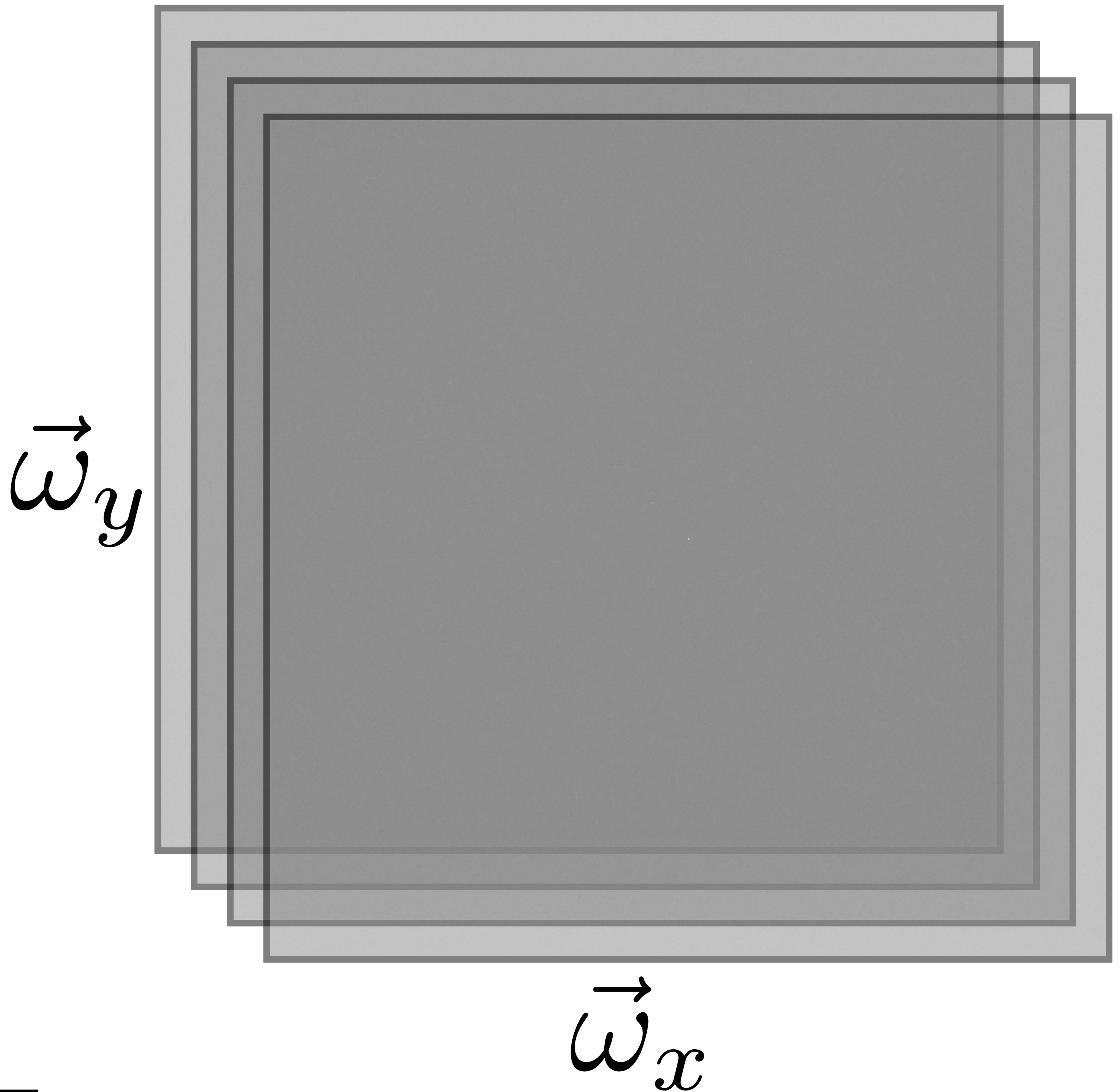
Independent Random Sampling

Many sample set realizations



$$\frac{1}{N}\sum_{k=1}^{N}\delta(|\vec{x}-\vec{x}_k|)$$

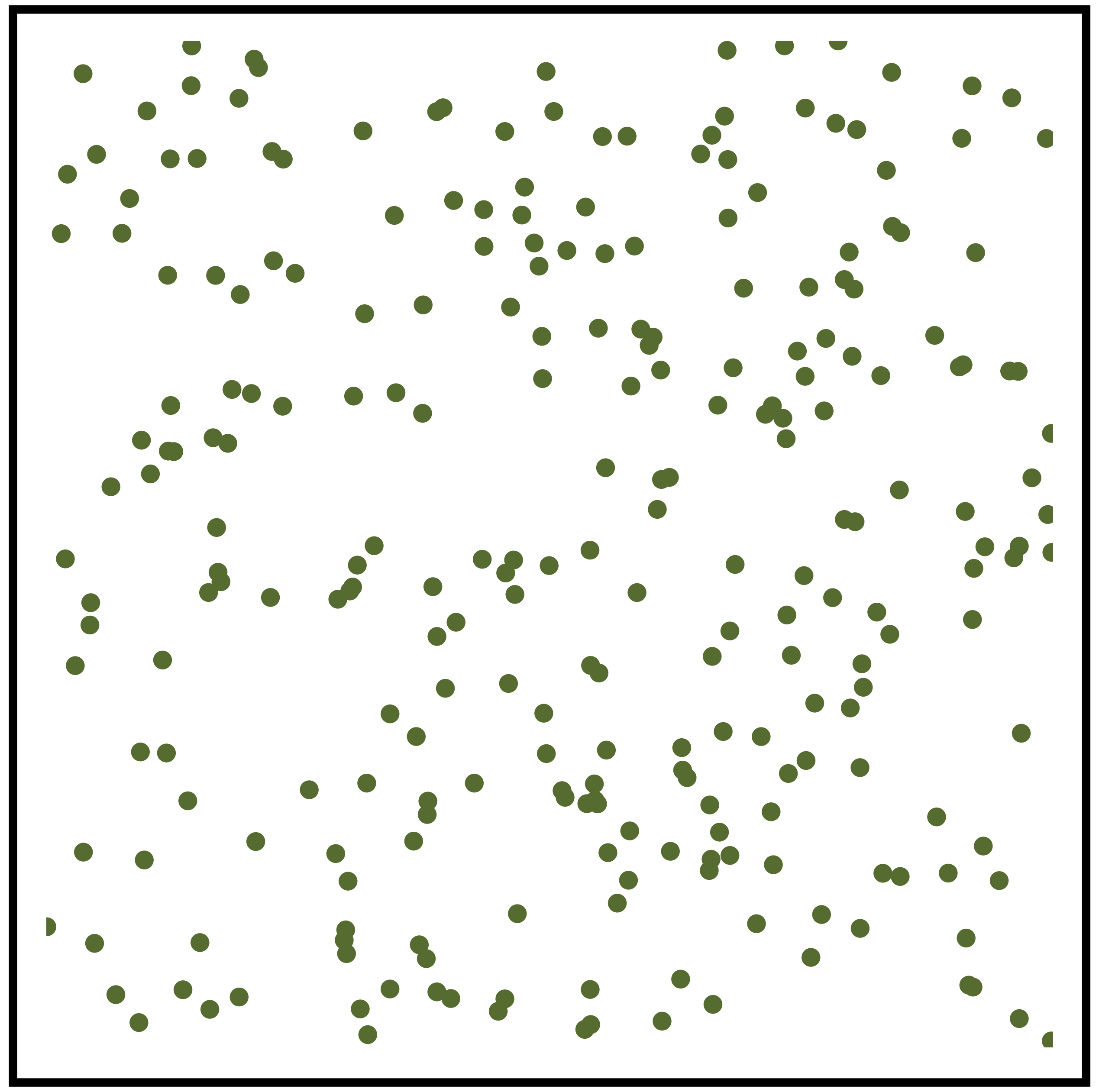
Expected power spectrum



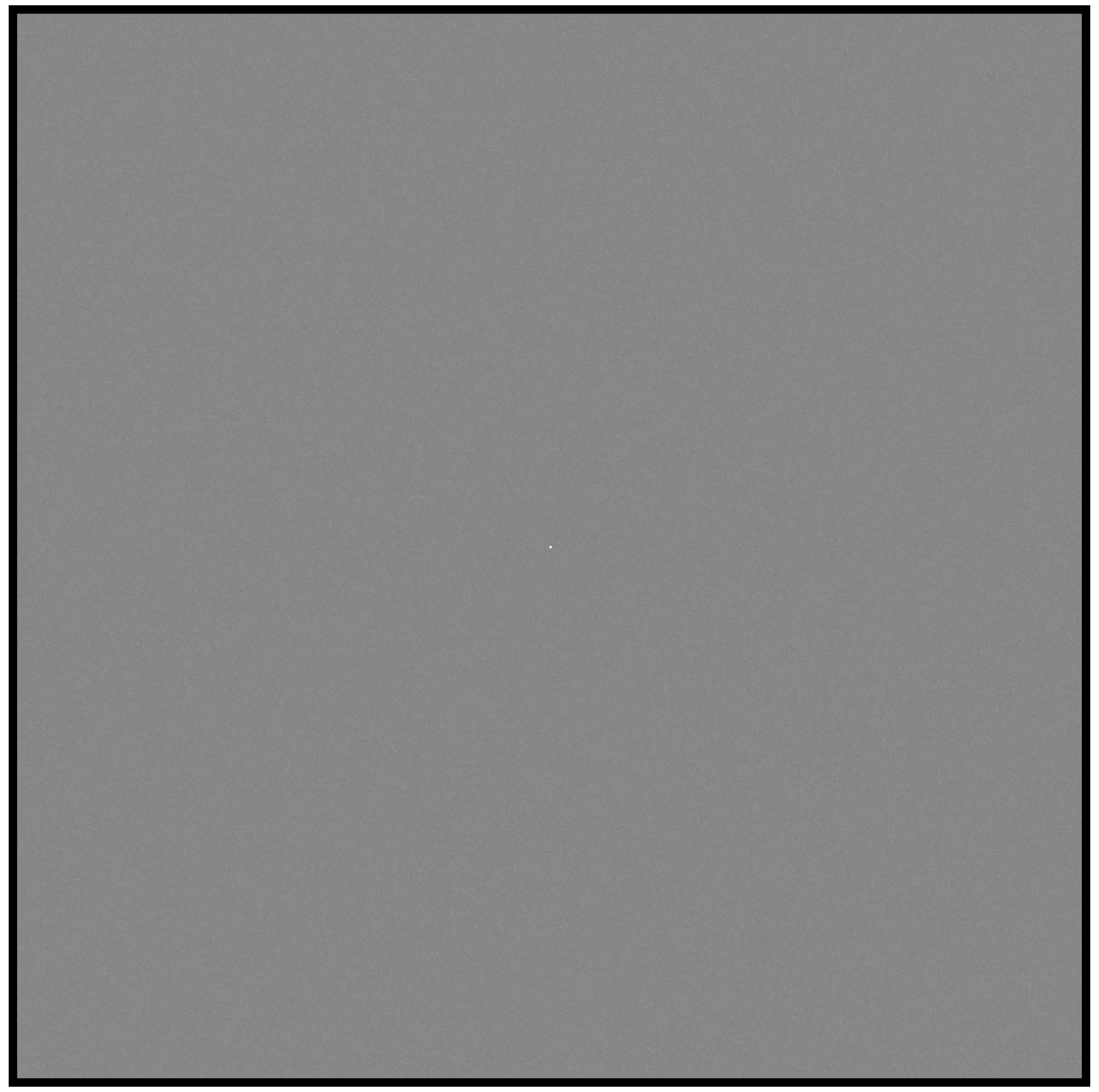
$$\mathrm{E}\left[\left|\frac{1}{N}\sum_{k=1}^{N}\mathrm{e}^{-2\,\pi\,\imath\,(\vec{\omega}\cdot\vec{x}_k)}\right|^2\right]$$

Independent Random Sampling

Samples



Expected power spectrum

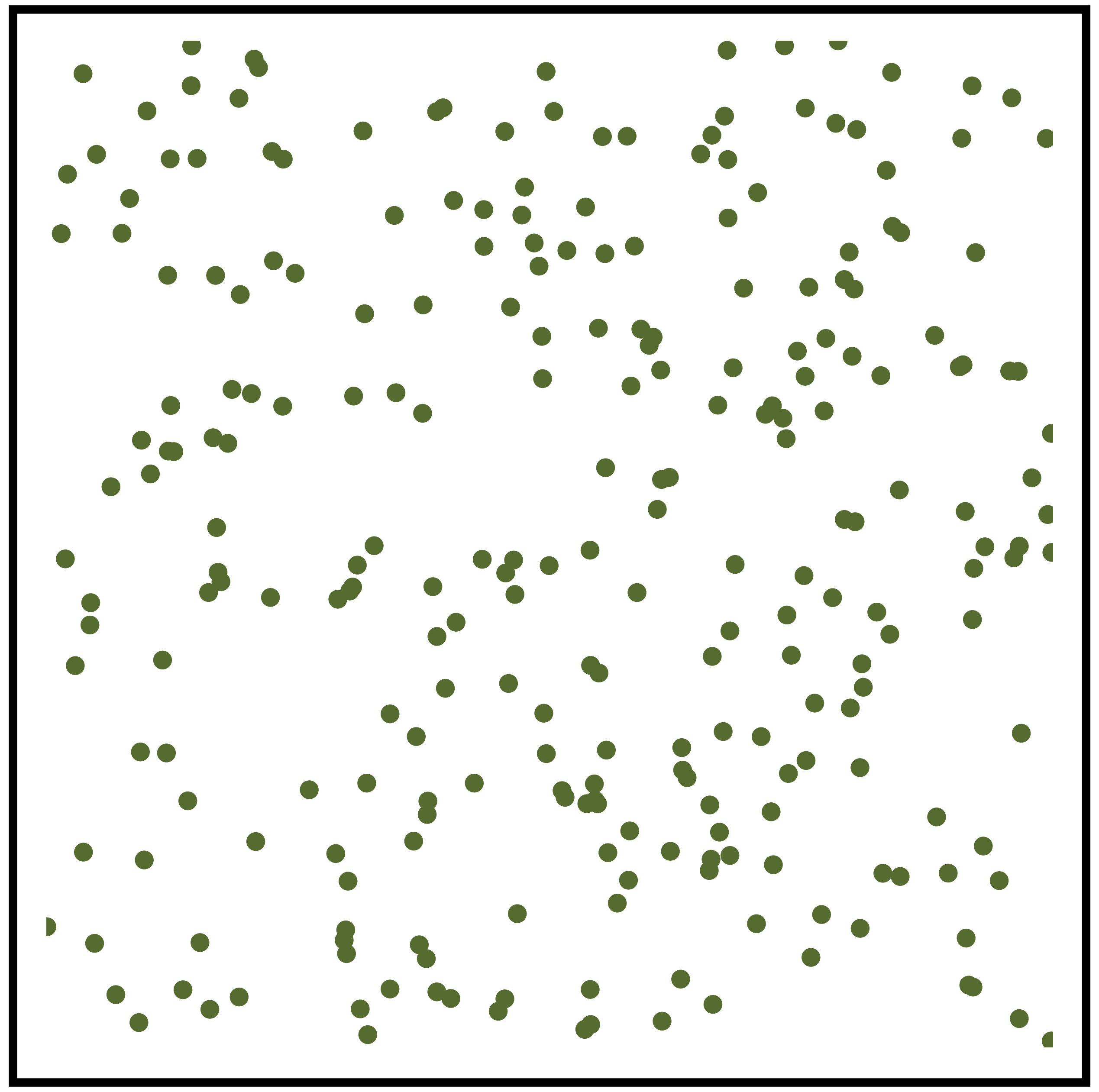


$$\frac{1}{N}\sum_{k=1}^{N}\delta(|\vec{x}-\vec{x}_k|)$$

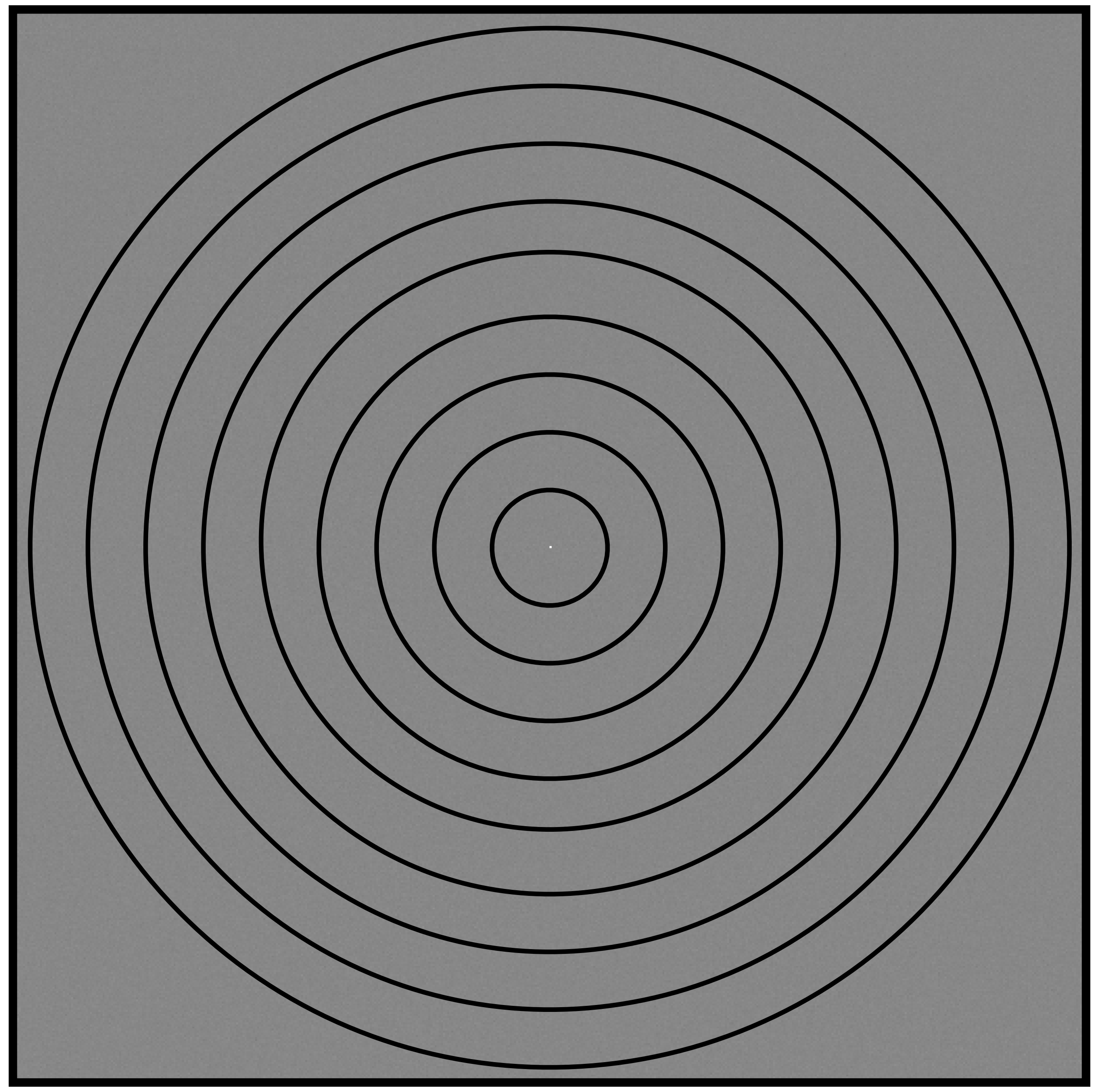
$$\mathrm{E}\left[\left|\frac{1}{N}\sum_{k=1}^{N}\mathrm{e}^{-2\,\pi\,\imath\,(\vec{\omega}\cdot\vec{x}_k)}\right|^2\right]$$

Independent Random Sampling

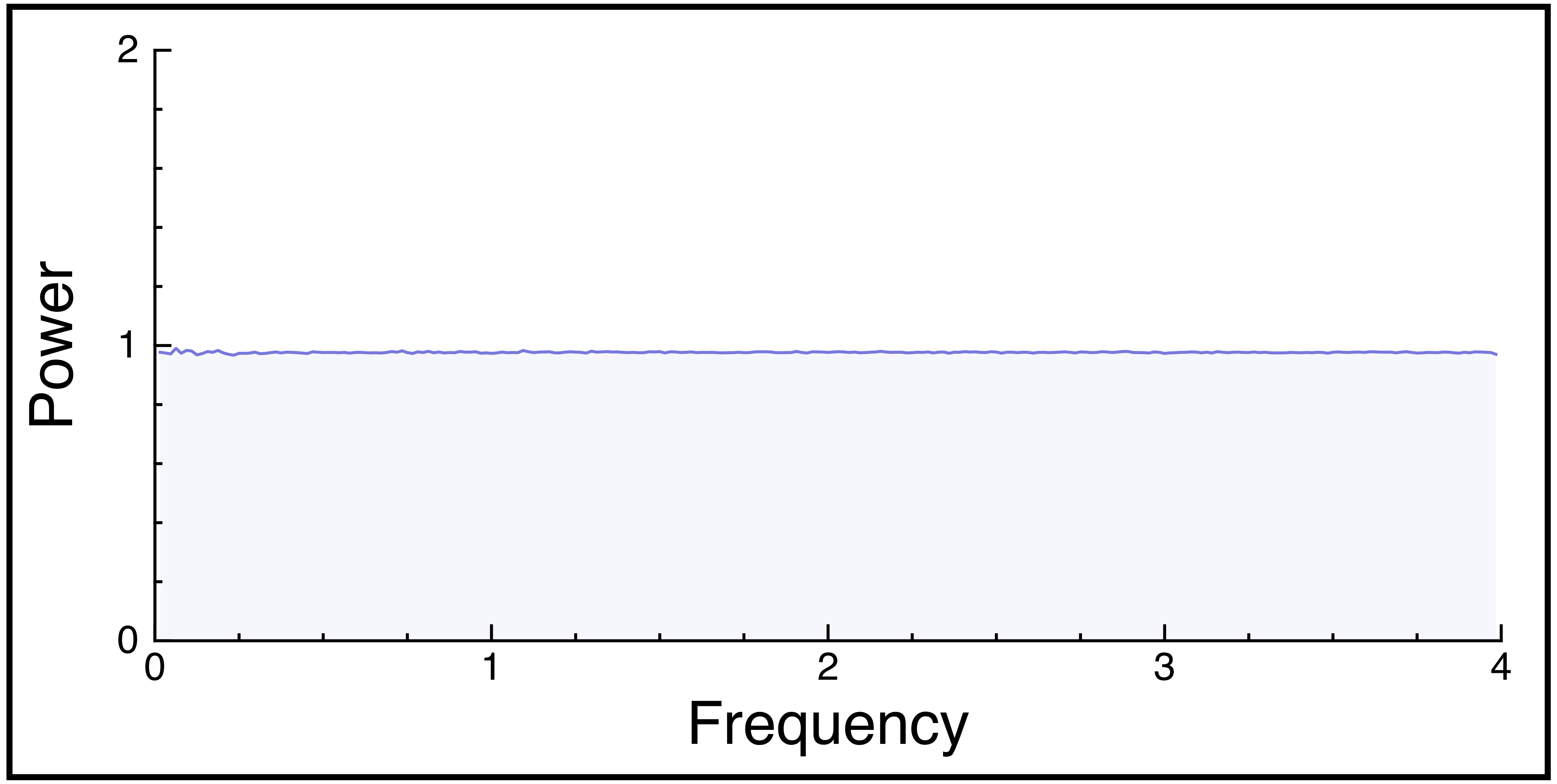
Samples



Expected power spectrum



Radial mean



$$\frac{1}{N}\sum_{k=1}^{N}\delta(|\vec{x}-\vec{x}_k|) \quad \mathrm{E}\left[\left|\frac{1}{N}\sum_{k=1}^{N} \mathrm{e}^{-2\,\pi\,\imath\,(\vec{\omega}\cdot\vec{x}_k)}\right|^2\right]$$

Independent Random Sampling

$$\frac{1}{N}\sum_{k=1}^{N}\delta(|\vec{x}-\vec{x}_k|) \quad \mathrm{E}\left[\left|\frac{1}{N}\sum_{k=1}^{N}\mathrm{e}^{-2\pi\imath(\vec{\omega}\cdot\vec{x}_k)}\right|^2\right]$$

Source code: Power spectrum

```
procedure powerSpectrum(samples, spectrumWidth, spectrumHeight)
  int N = samples.size()
  for u: 0 → spectrumWidth{
    for v: 0 → spectrumHeight{
    double real = 0, imag = 0;



    }
  }
  return power;
}
```

Source code: Power spectrum

```
procedure powerSpectrum(samples, spectrumWidth, spectrumHeight)
  int N = samples.size()
  for u: 0 → spectrumWidth{
    for v: 0 → spectrumHeight{
    double real = 0, imag = 0;

    //compute the real and imaginary fourier coefficients
    for(int k=0;k<N;k++){




    }
  }
  return power;
}
```

Source code: Power spectrum

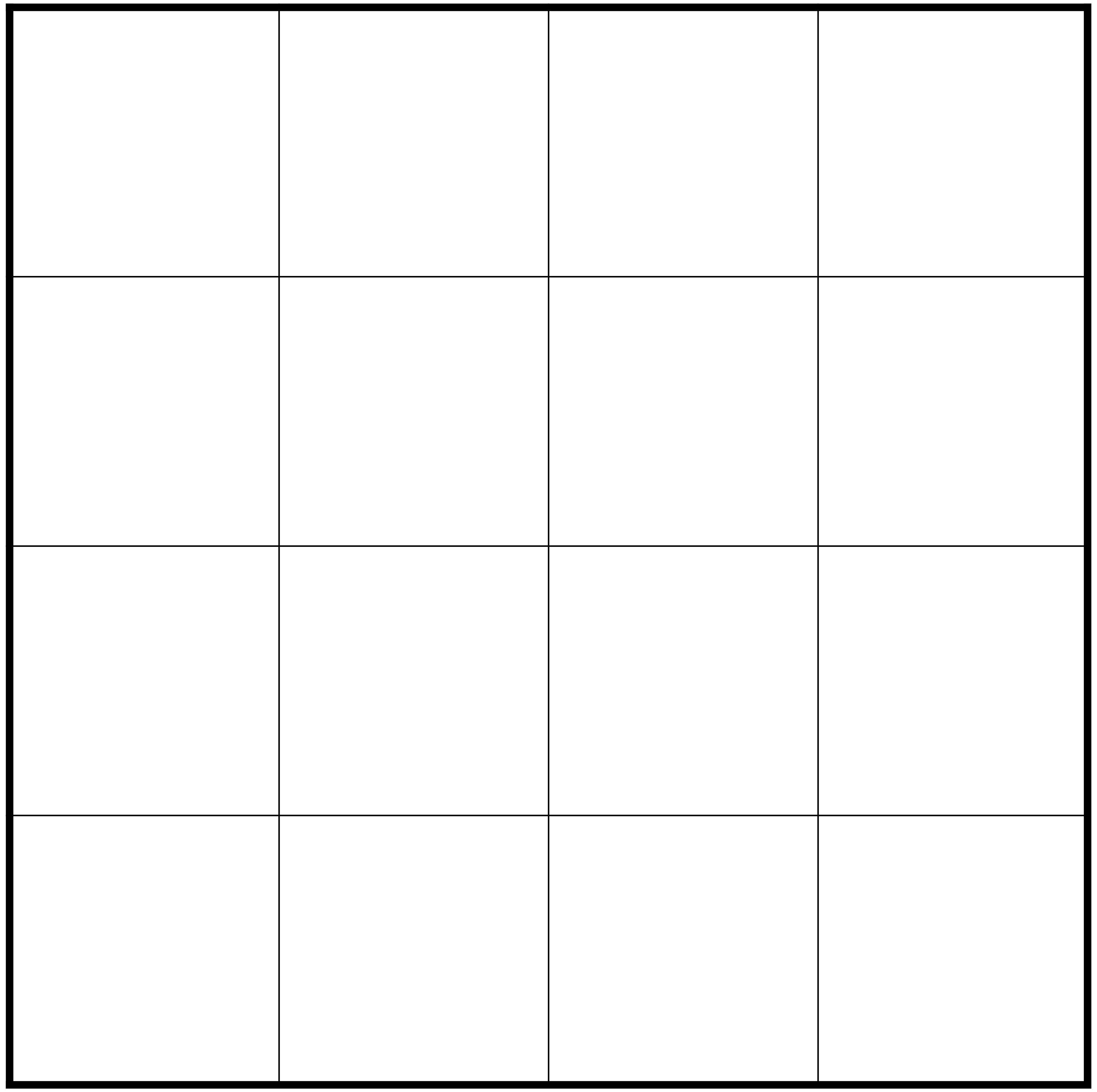
```
procedure powerSpectrum(samples, spectrumWidth, spectrumHeight)
  int N = samples.size()
  for u: 0 → spectrumWidth{
    for v: 0 → spectrumHeight{
    double real = 0, imag = 0;

    //compute the real and imaginary fourier coefficients
    for(int k=0;k<N;k++){
      real += cos(2 * π * (u * samples[k].x + v * samples[k].y));
      imag += sin(2 * π * (u * samples[k].x + v * samples[k].y));
    }



    }
  }
  return power;
}
```

Source code: Power spectrum

```
procedure powerSpectrum(samples, spectrumWidth, spectrumHeight)
  int N = samples.size()
  for u: 0 → spectrumWidth{
    for v: 0 → spectrumHeight{
    double real = 0, imag = 0;

    //compute the real and imaginary fourier coefficients
    for(int k=0;k<N;k++){
      real += cos(2 * π * (u * samples[k].x + v * samples[k].y));
      imag += sin(2 * π * (u * samples[k].x + v * samples[k].y));
    }

    //power spectrum is the magnitude square value of the coefficients
    power[u * spectrumWidth + v] = (real*real + imag * imag) / N;
    }
  }
  return power;
}
```

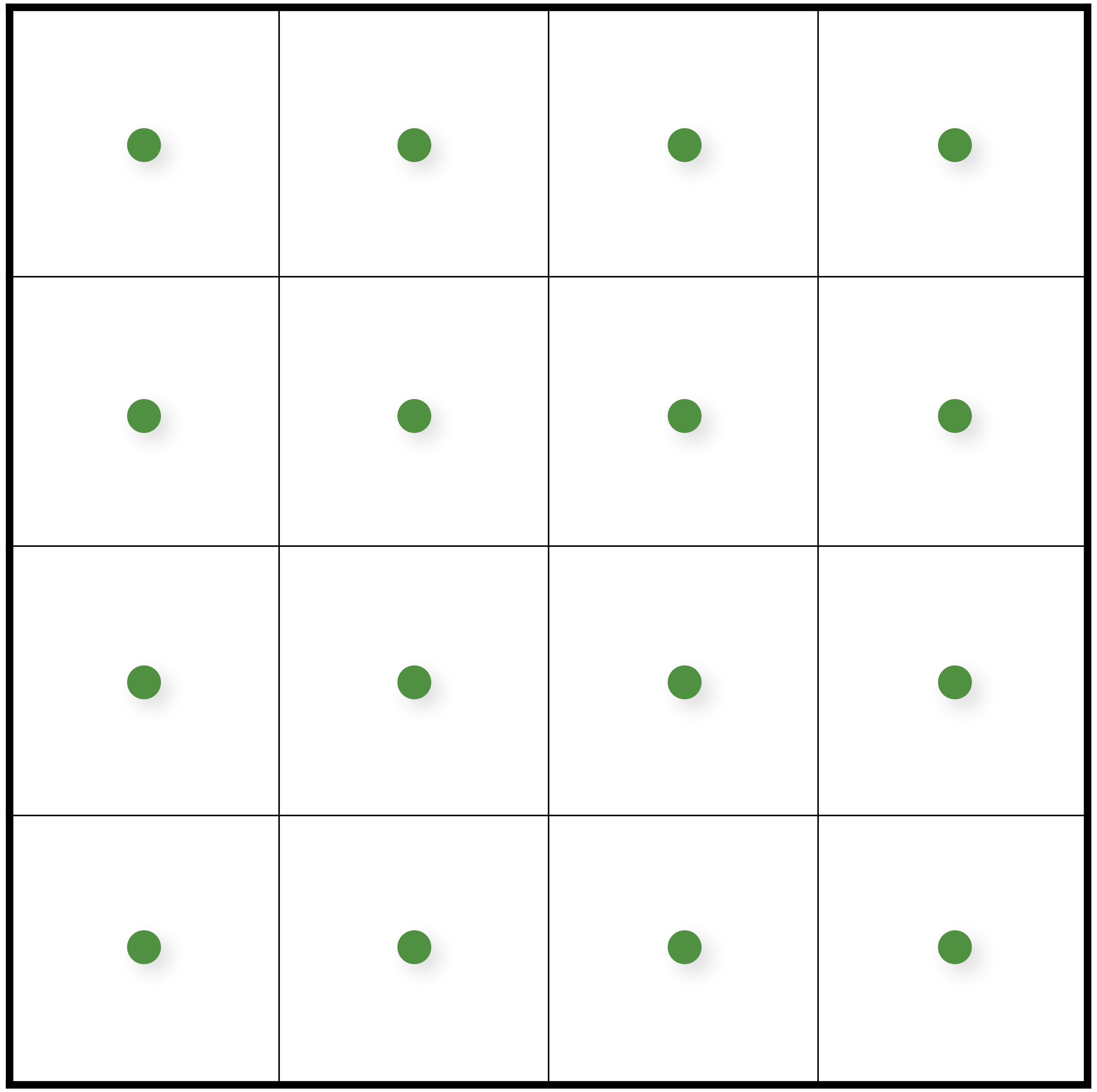
Regular Sampling

```
for (uint i = 0; i < numX; i++)
    for (uint j = 0; j < numY; j++)
    {
        samples(i,j).x = (i + 0.5)/numX;
        samples(i,j).y = (j + 0.5)/numY;
    }
```



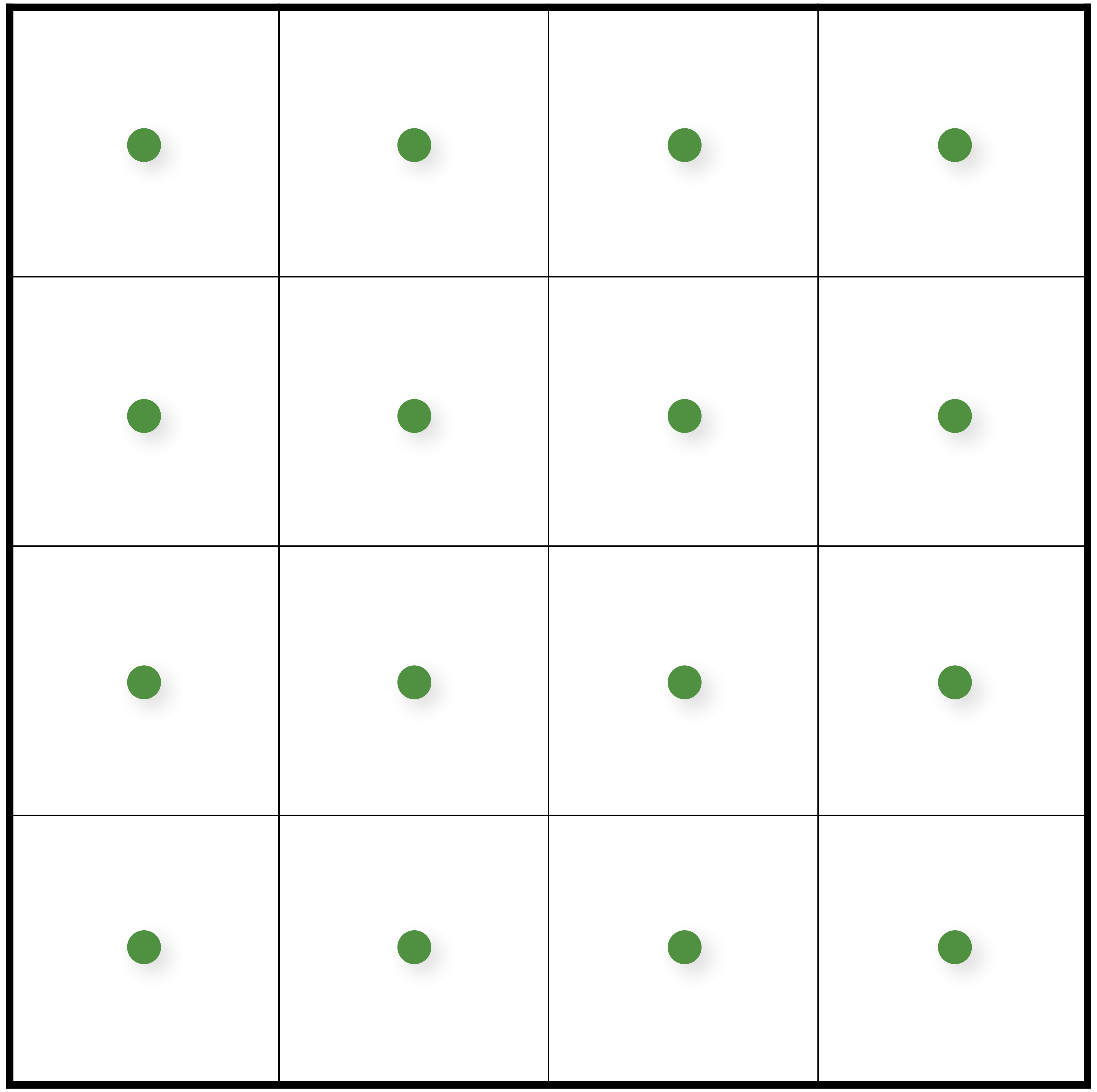
Regular Sampling

```
for (uint i = 0; i < numX; i++)
	for (uint j = 0; j < numY; j++)
	{
		samples(i,j).x = (i + 0.5)/numX;
		samples(i,j).y = (j + 0.5)/numY;
	}
```



Regular Sampling

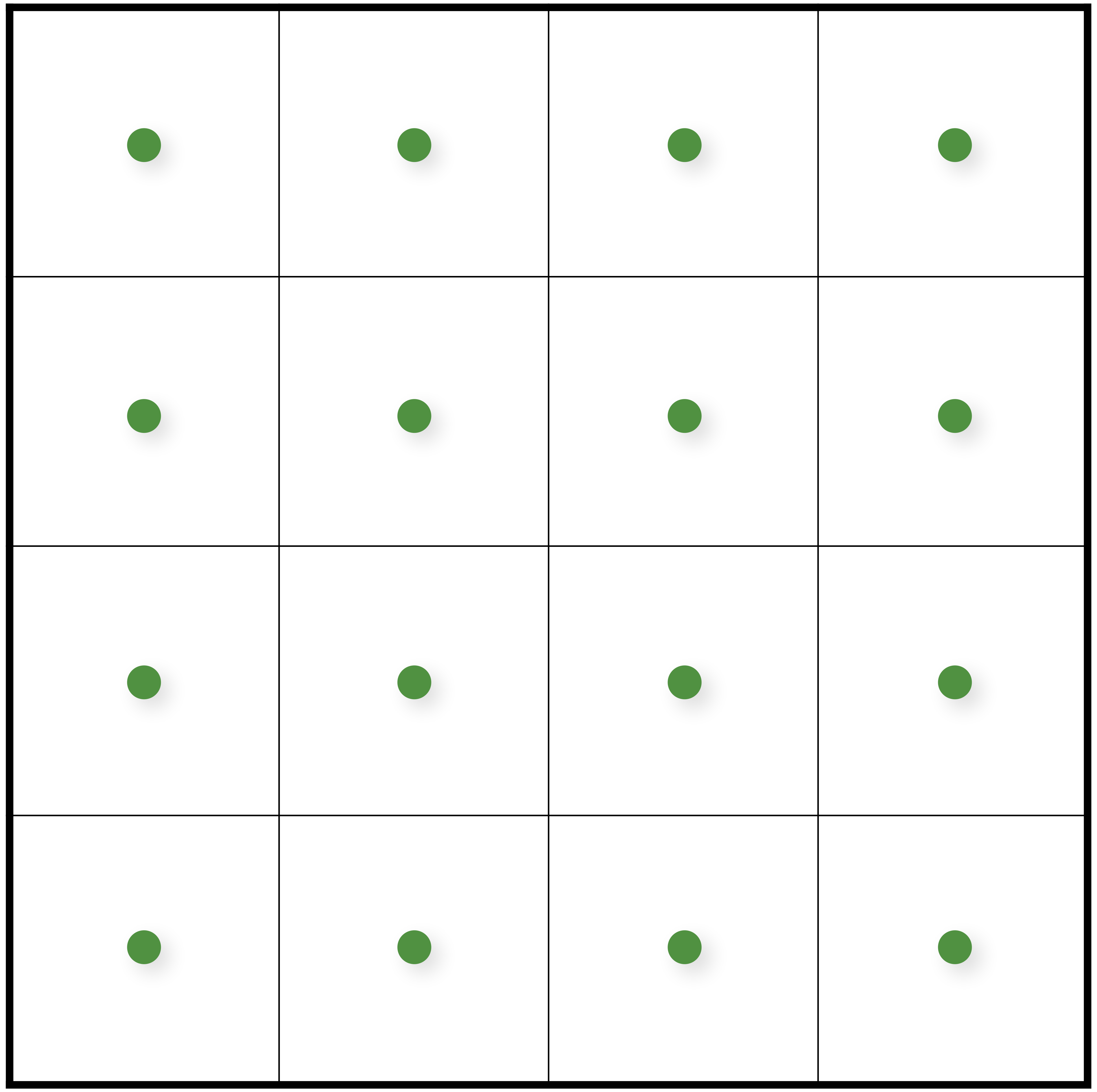
```
for (uint i = 0; i < numX; i++)
    for (uint j = 0; j < numY; j++)
    {
        samples(i,j).x = (i + 0.5)/numX;
        samples(i,j).y = (j + 0.5)/numY;
    }
```



✔ Extends to higher dimensions, but…

Regular Sampling

```
for (uint i = 0; i < numX; i++)
	for (uint j = 0; j < numY; j++)
	{
		samples(i,j).x = (i + 0.5)/numX;
		samples(i,j).y = (j + 0.5)/numY;
	}
```

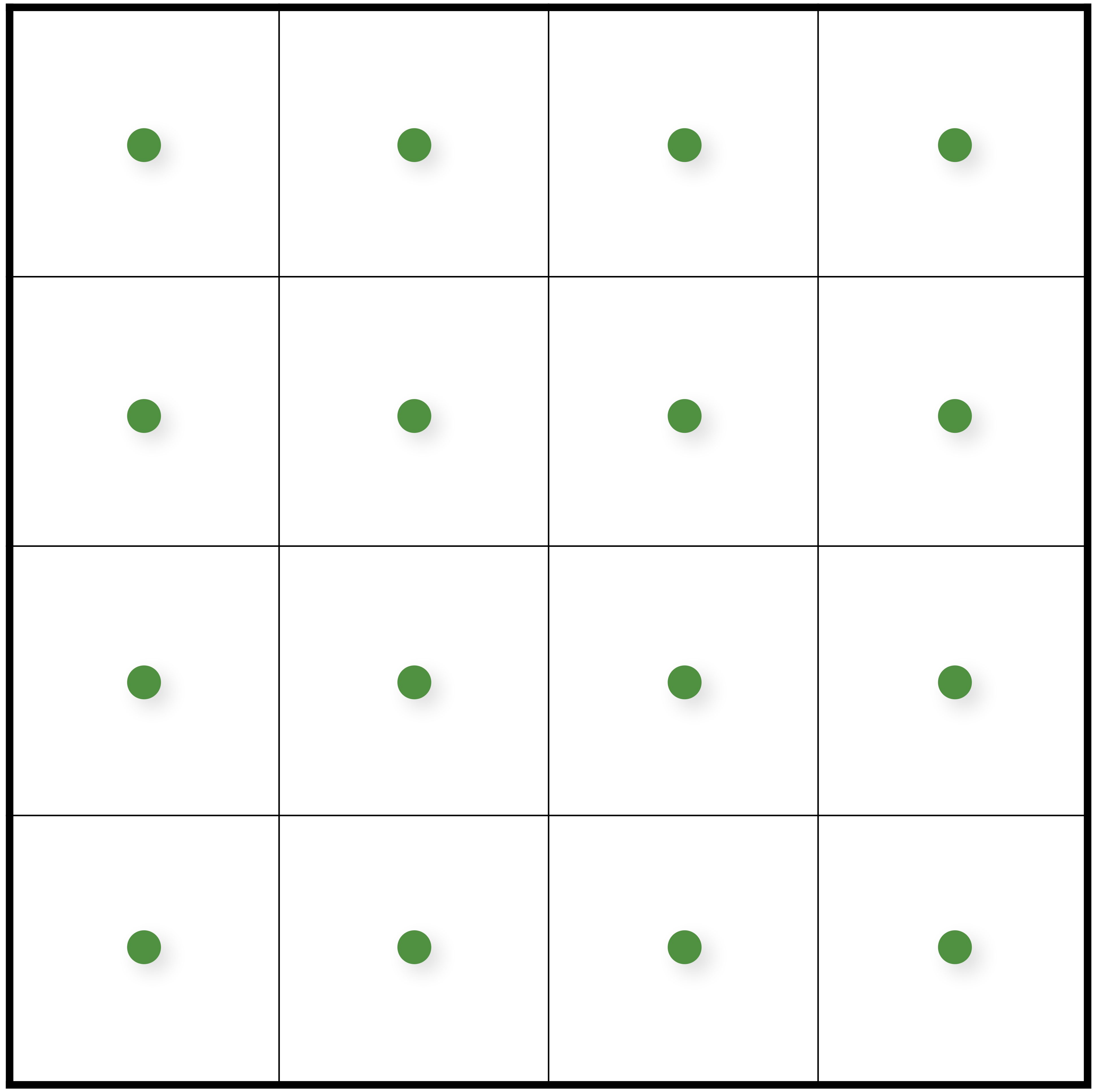


✔ Extends to higher dimensions, but…

✘ Curse of dimensionality

Regular Sampling

```
for (uint i = 0; i < numX; i++)
    for (uint j = 0; j < numY; j++)
    {
        samples(i,j).x = (i + 0.5)/numX;
        samples(i,j).y = (j + 0.5)/numY;
    }
```



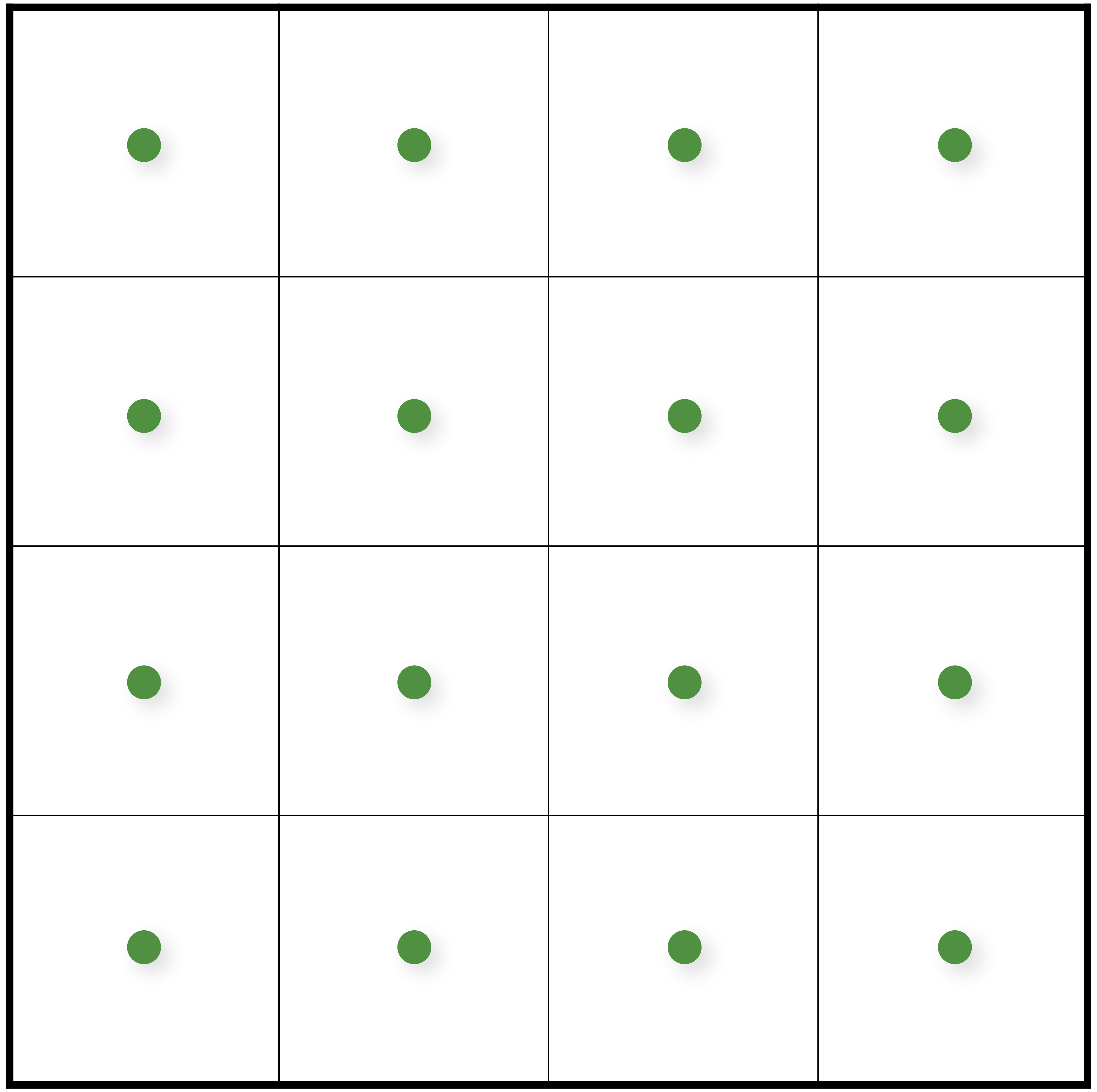
✔ Extends to higher dimensions, but…

✘ Curse of dimensionality

✘ Aliasing

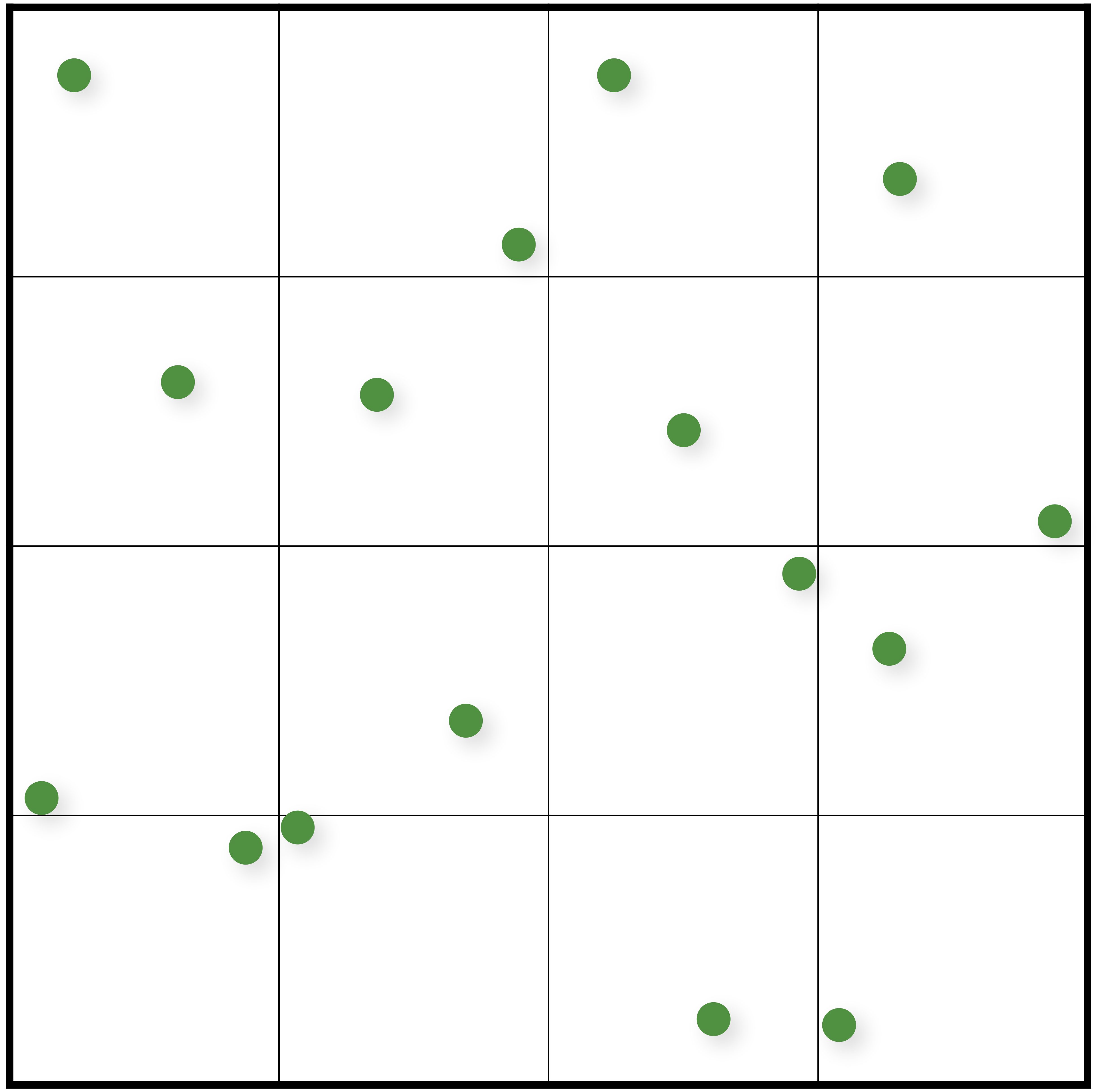
Regular Sampling

```
for (uint i = 0; i < numX; i++)
    for (uint j = 0; j < numY; j++)
    {
        samples(i,j).x = (i + 0.5)/numX;
        samples(i,j).y = (j + 0.5)/numY;
    }
```



Jittered/Stratified Sampling

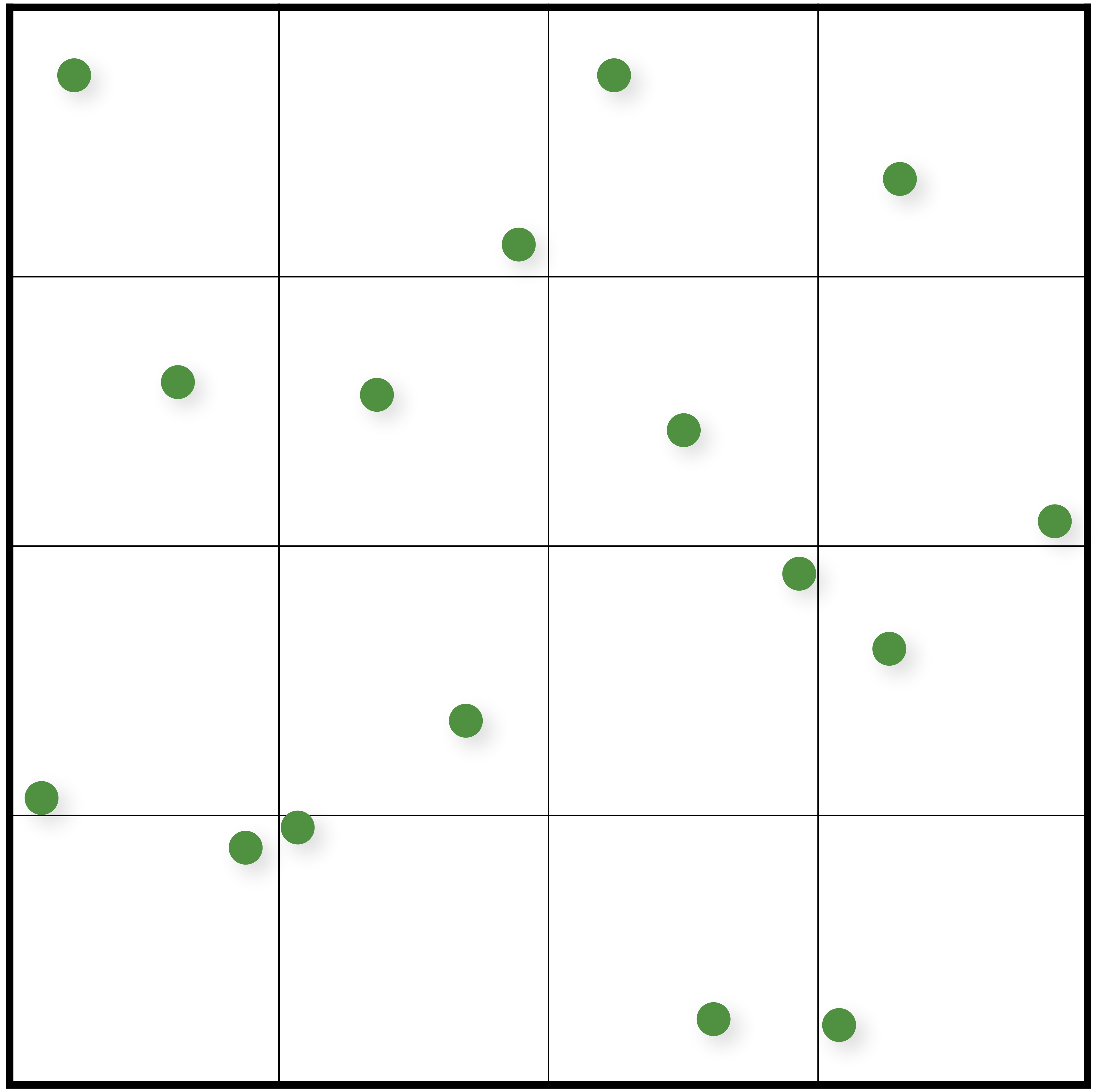
```
for (uint i = 0; i < numX; i++)
    for (uint j = 0; j < numY; j++)
    {
        samples(i,j).x = (i + randf())/numX;
        samples(i,j).y = (j + randf())/numY;
    }
```



Jittered/Stratified Sampling

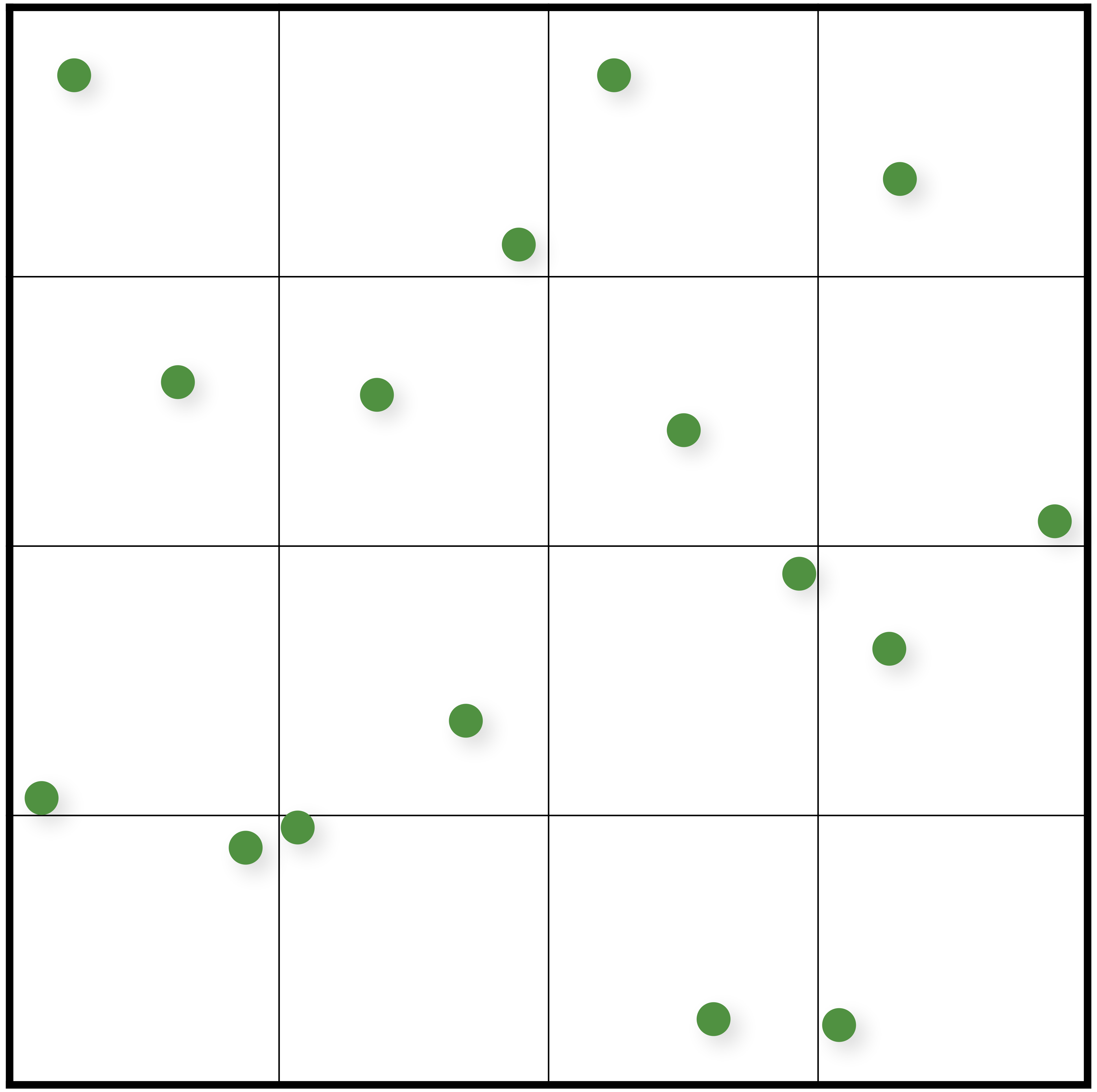
```
for (uint i = 0; i < numX; i++)
    for (uint j = 0; j < numY; j++)
    {
        samples(i,j).x = (i + randf())/numX;
        samples(i,j).y = (j + randf())/numY;
    }
```

✔ Provably cannot increase variance



Jittered/Stratified Sampling

```
for (uint i = 0; i < numX; i++)
    for (uint j = 0; j < numY; j++)
    {
        samples(i,j).x = (i + randf())/numX;
        samples(i,j).y = (j + randf())/numY;
    }
```

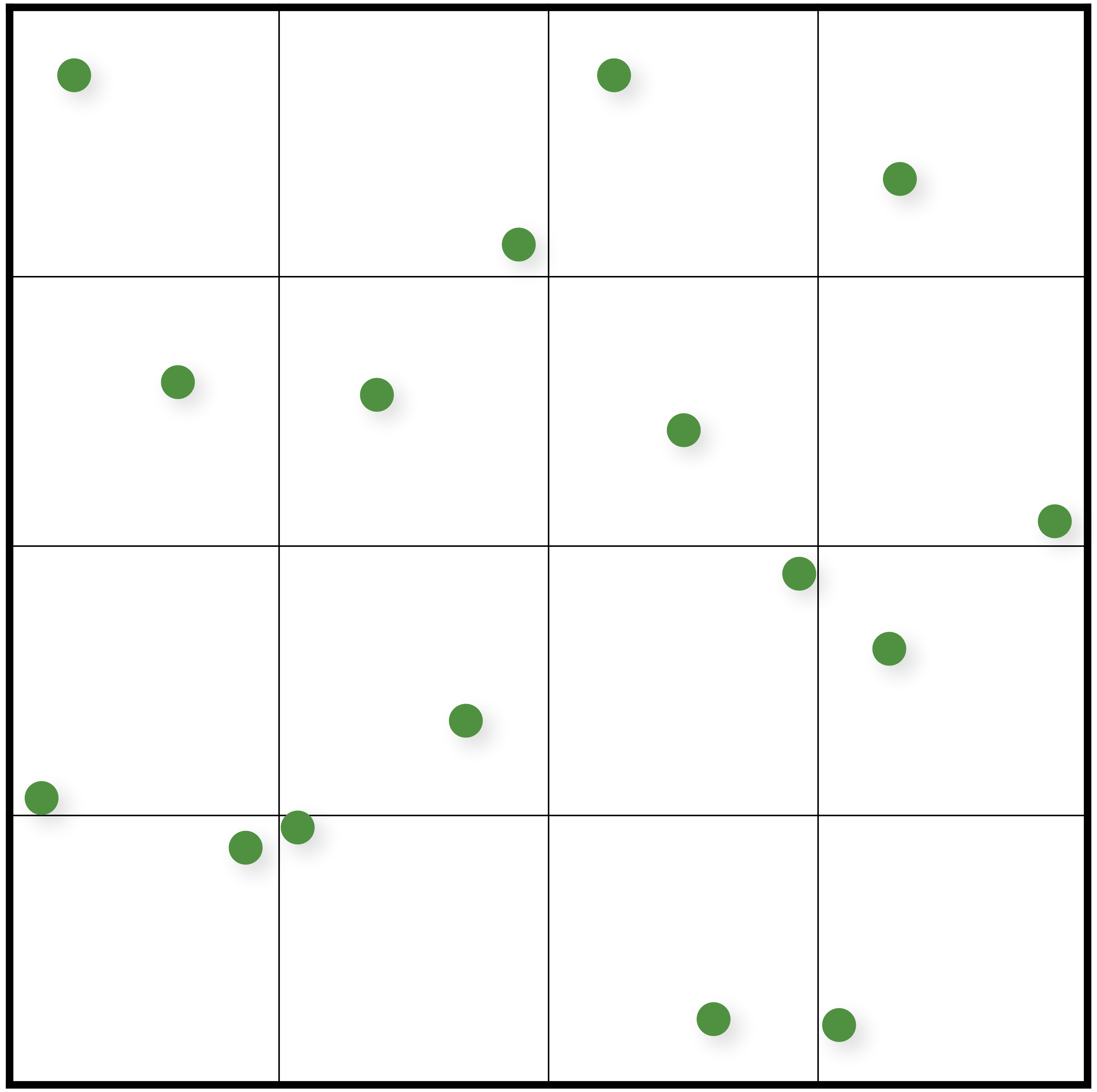


✔ Provably cannot increase variance

✔ Extends to higher dimensions, but…

Jittered/Stratified Sampling

```
for (uint i = 0; i < numX; i++)
    for (uint j = 0; j < numY; j++)
    {
        samples(i,j).x = (i + randf())/numX;
        samples(i,j).y = (j + randf())/numY;
    }
```



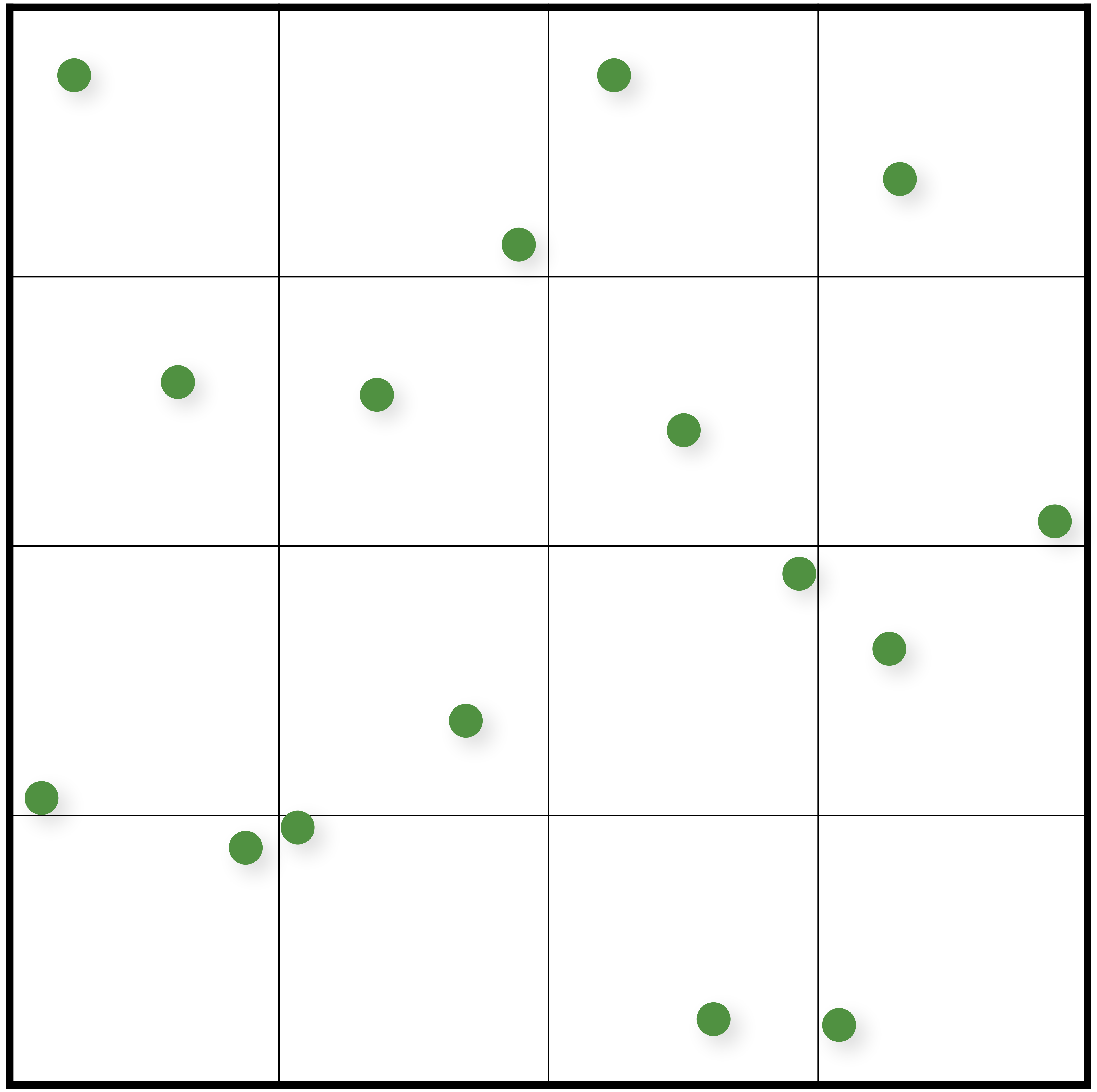
✔ Provably cannot increase variance

✔ Extends to higher dimensions, but…

✘ Curse of dimensionality

Jittered/Stratified Sampling

```
for (uint i = 0; i < numX; i++)
        for (uint j = 0; j < numY; j++)
        {
                samples(i,j).x = (i + randf())/numX;
                samples(i,j).y = (j + randf())/numY;
        }
```



✔ Provably cannot increase variance

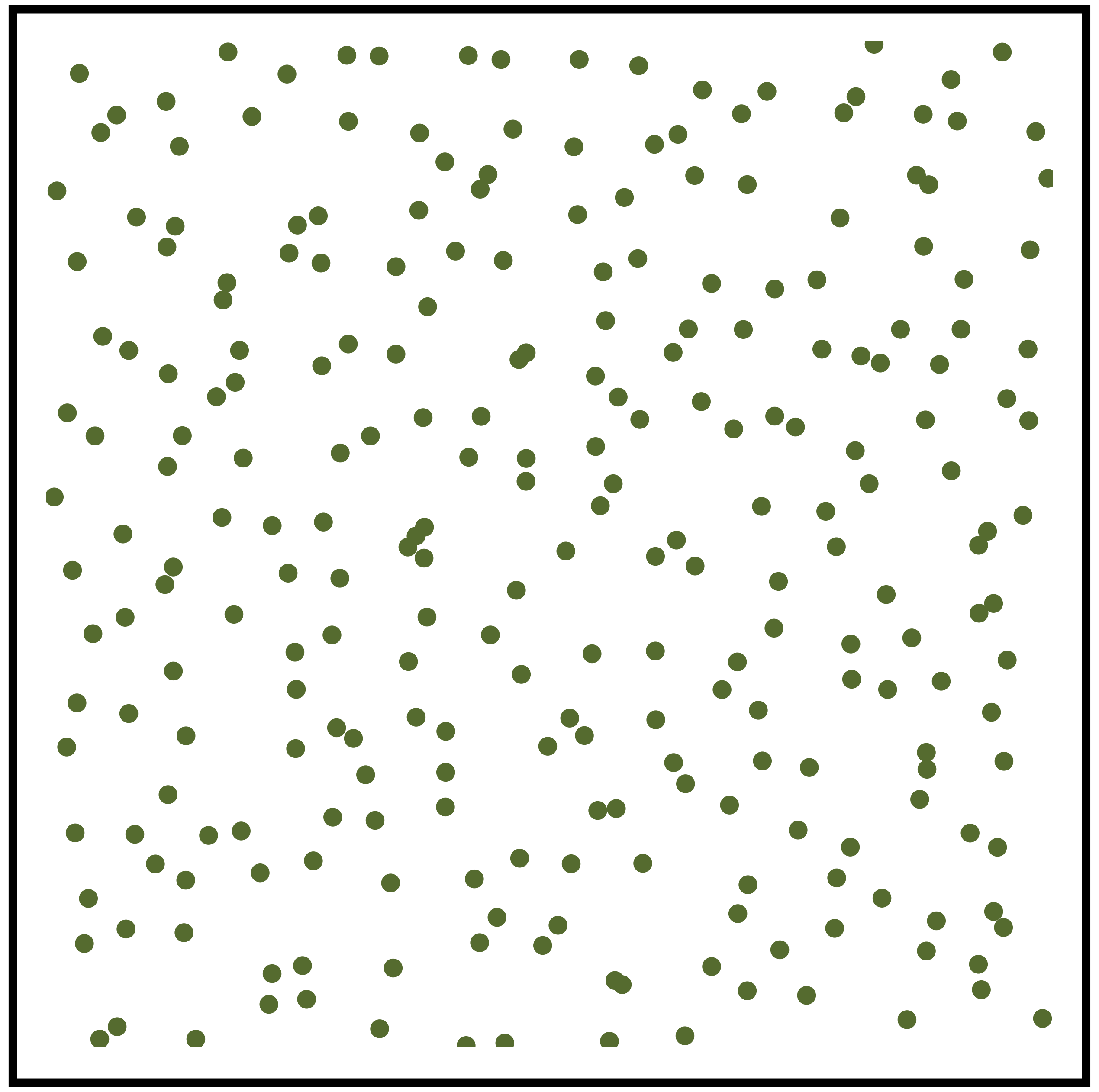
✔ Extends to higher dimensions, but…

✘ Curse of dimensionality

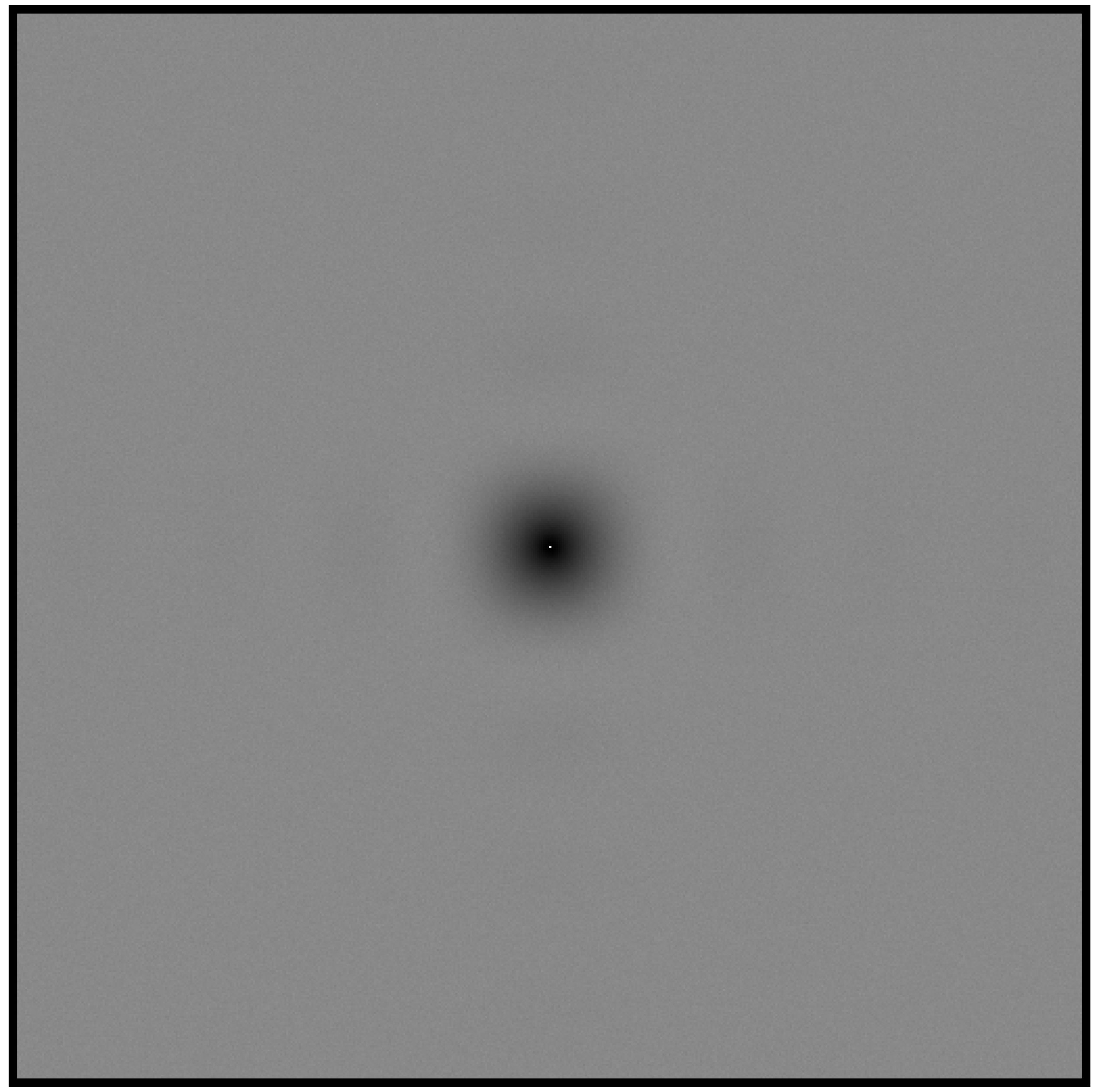
✘ Not progressive

Jittered Sampling

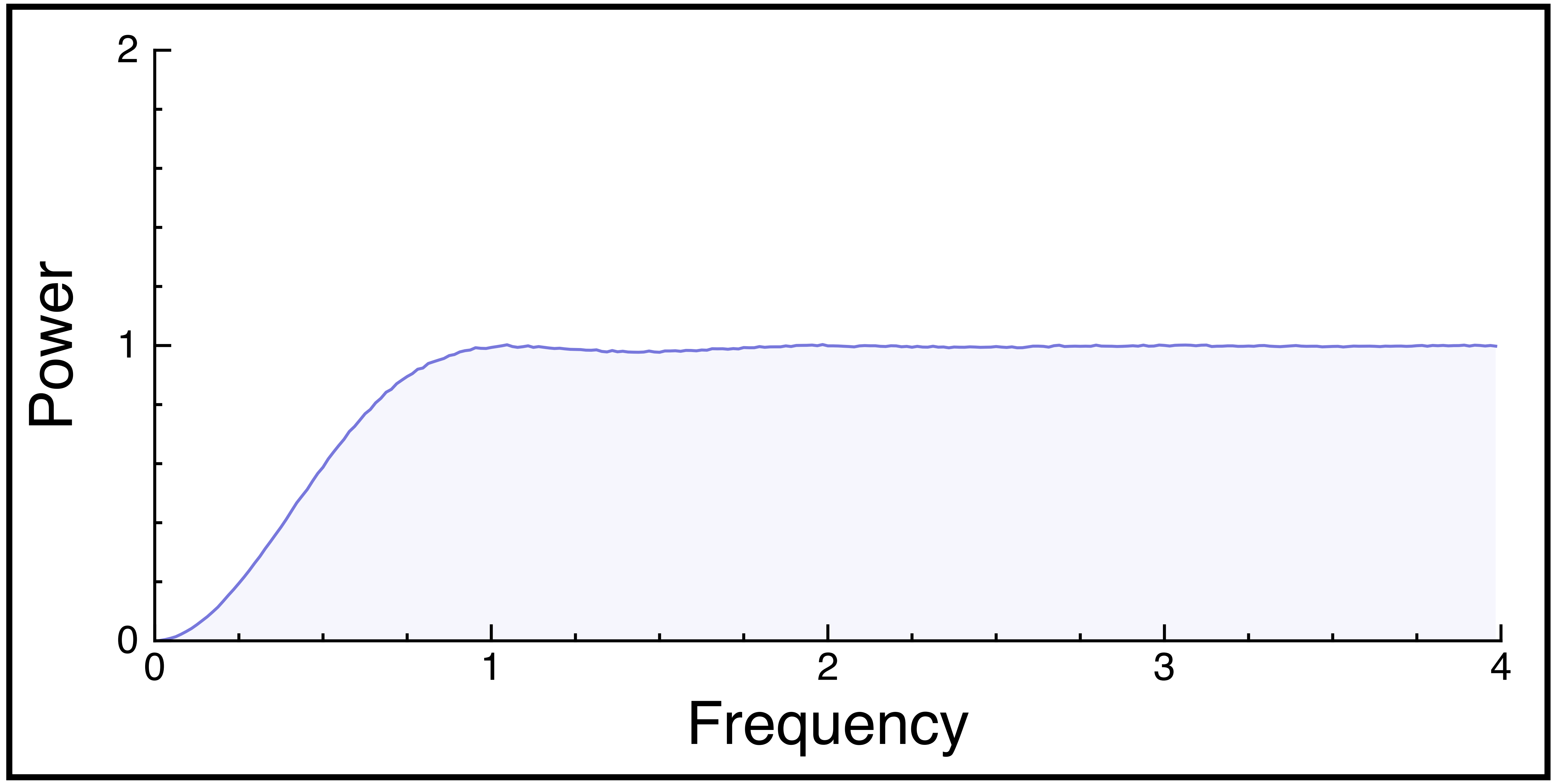
Samples



Expected power spectrum



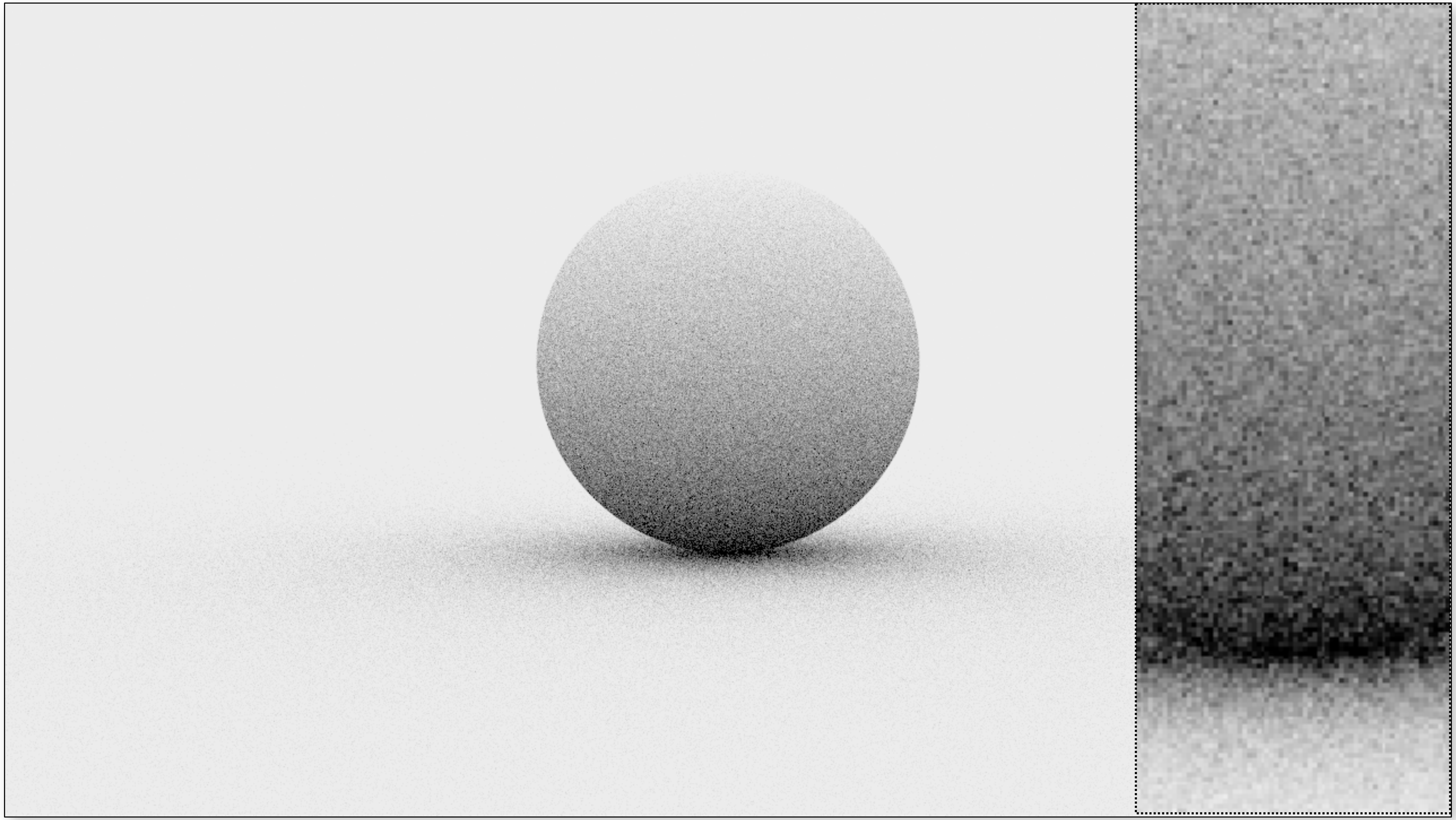
Radial mean



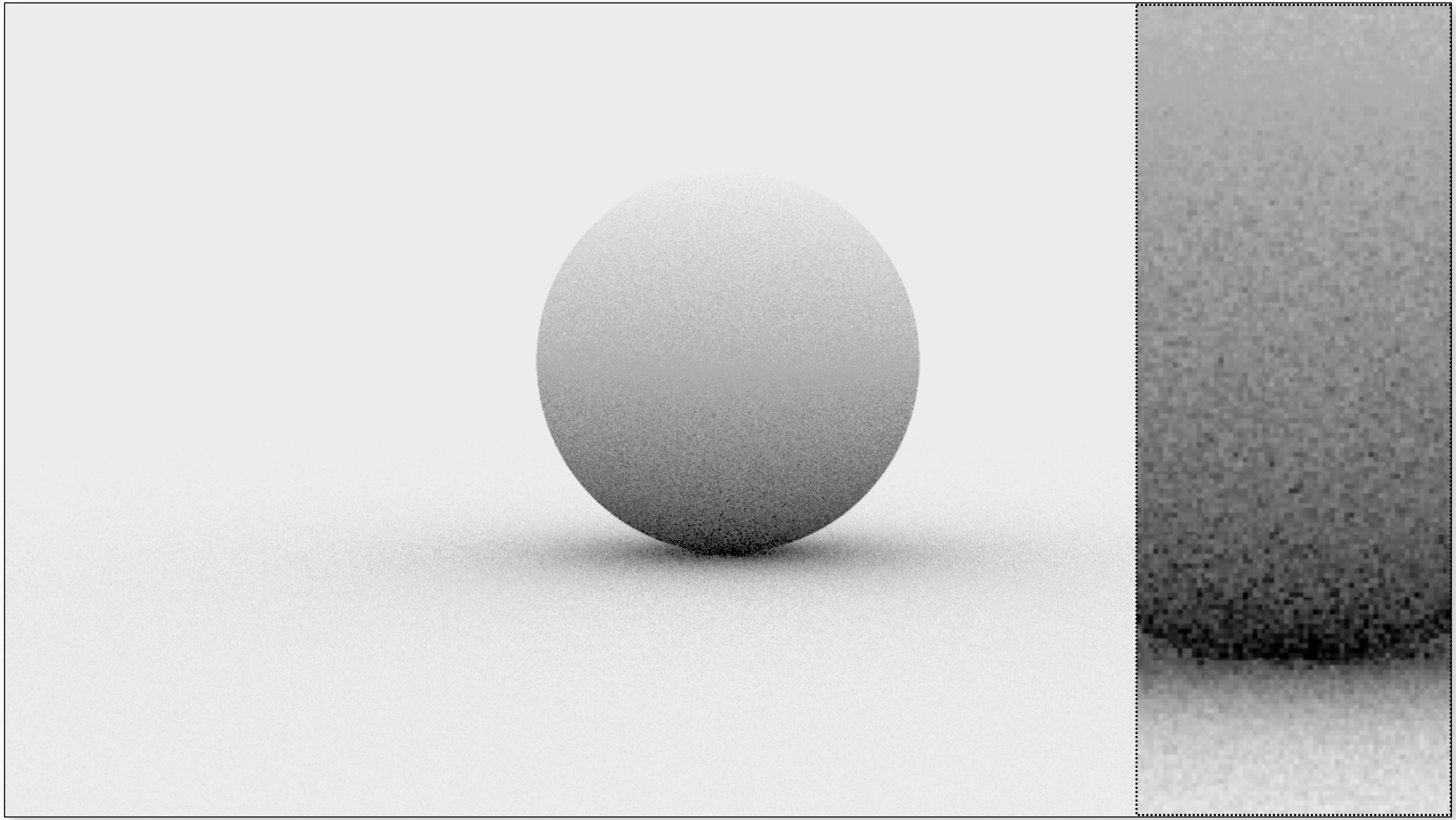
Independent Random Sampling

Samples | Expected power spectrum | Radial mean

Monte Carlo (16 random samples)



Monte Carlo (16 jittered samples)



Stratifying in Higher Dimensions

Stratification requires O(N^d) samples

- e.g. pixel (2D) + lens (2D) + time (1D) = 5D

Stratifying in Higher Dimensions

Stratification requires O(N^d) samples

- e.g. pixel (2D) + lens (2D) + time (1D) = 5D
  - splitting 2 times in 5D = 2^5 = 32 samples
  - splitting 3 times in 5D = 3^5 = 243 samples!

Stratifying in Higher Dimensions

Stratification requires $O(N^d)$ samples

- e.g. pixel (2D) + lens (2D) + time (1D) = 5D
  - splitting 2 times in 5D = 2^5 = 32 samples
  - splitting 3 times in 5D = 3^5 = 243 samples!

Inconvenient for large d

- cannot select sample count with fine granularity

Uncorrelated Jitter [Cook et al. 84]

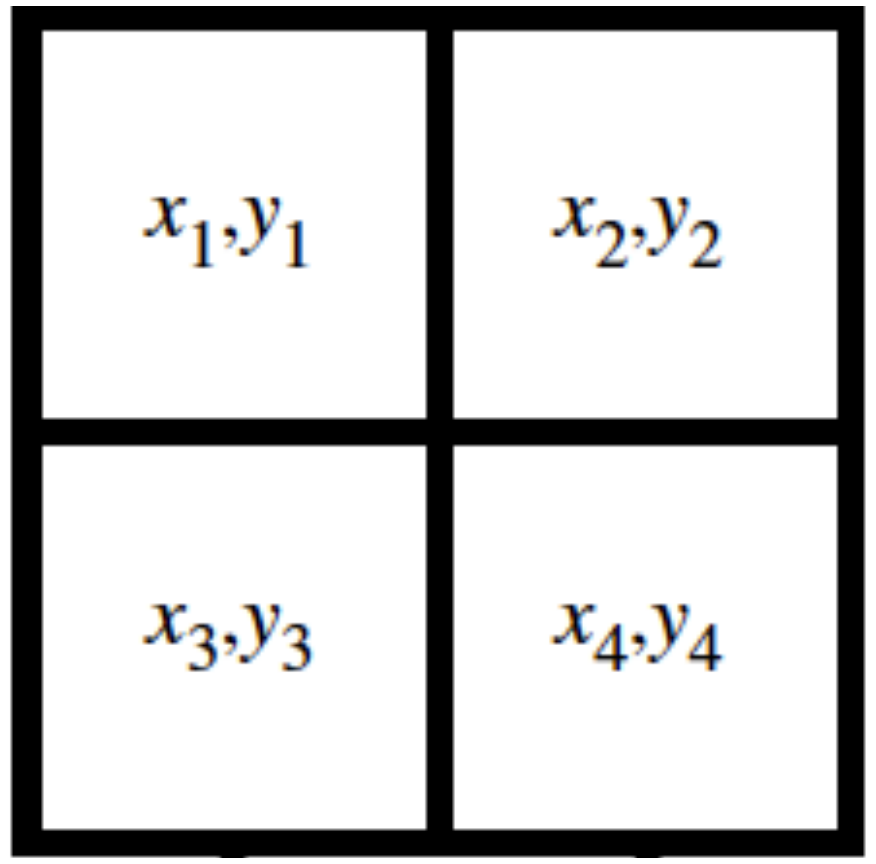
Uncorrelated Jitter [Cook et al. 84]

Compute stratified samples in sub-dimensions

Uncorrelated Jitter [Cook et al. 84]

Compute stratified samples in sub-dimensions

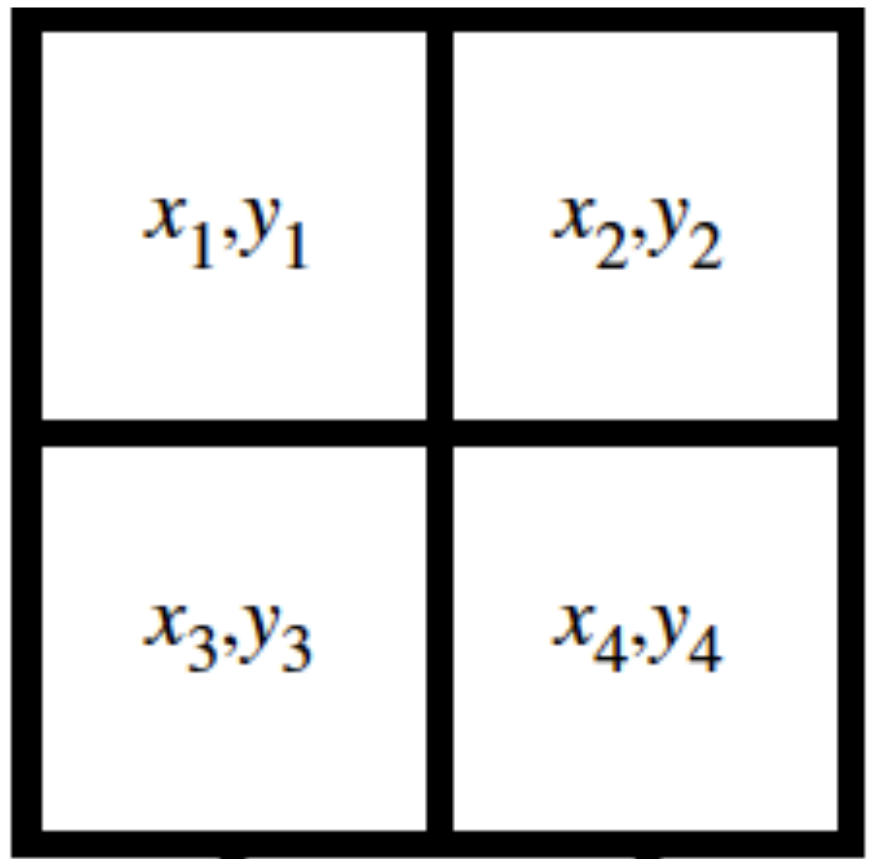
- 2D jittered (x,y) for pixel

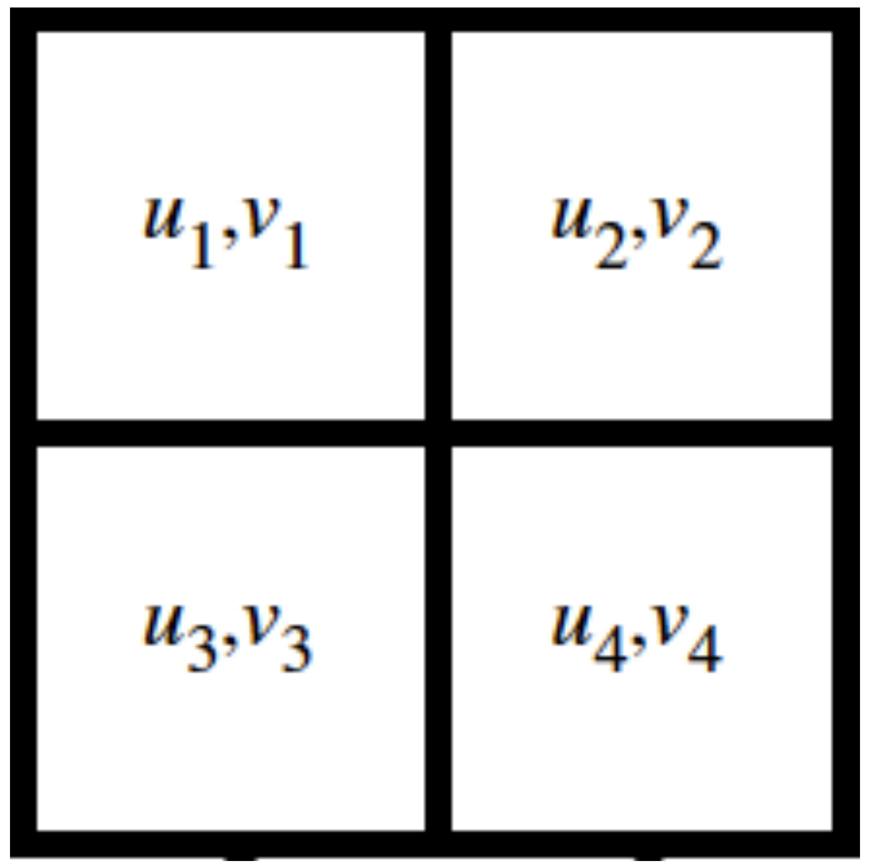


Uncorrelated Jitter [Cook et al. 84]

Compute stratified samples in sub-dimensions

- 2D jittered (x,y) for pixel
- 2D jittered (u,v) for lens

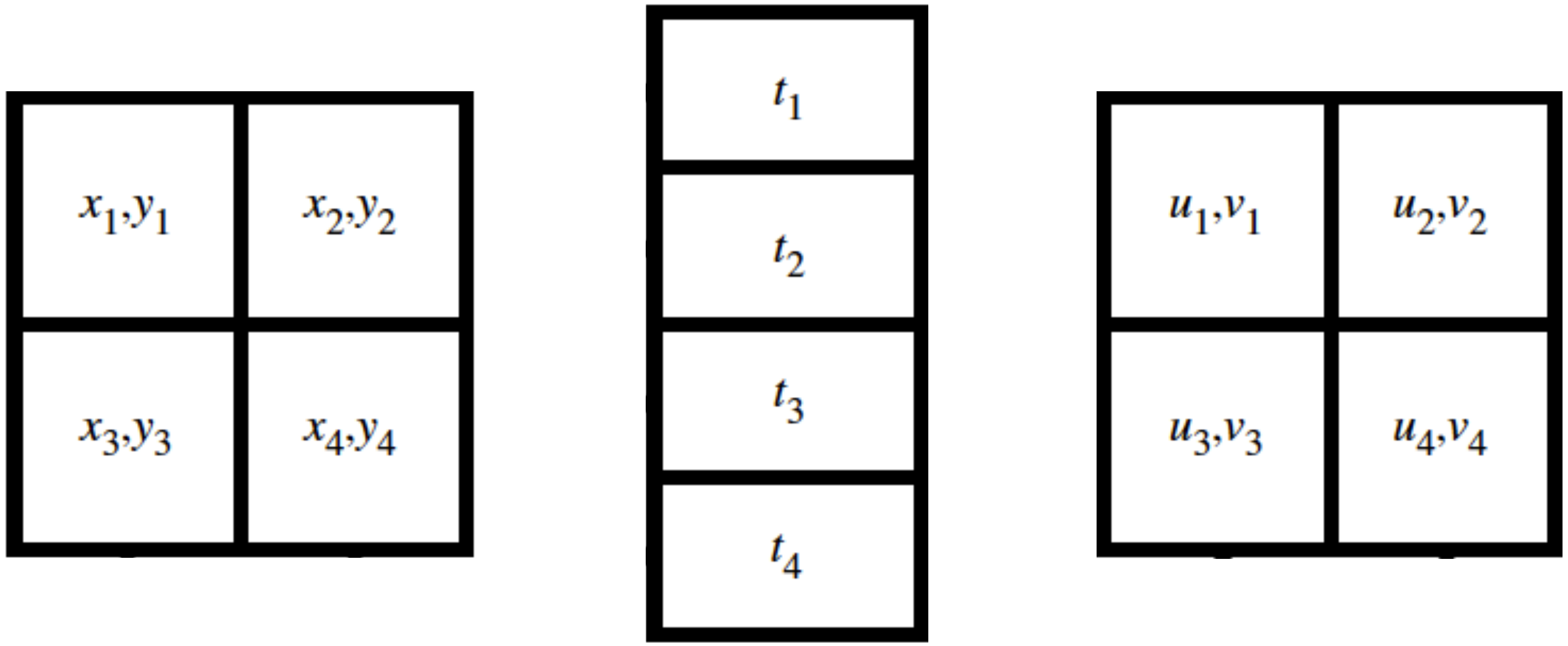




Uncorrelated Jitter [Cook et al. 84]

Compute stratified samples in sub-dimensions

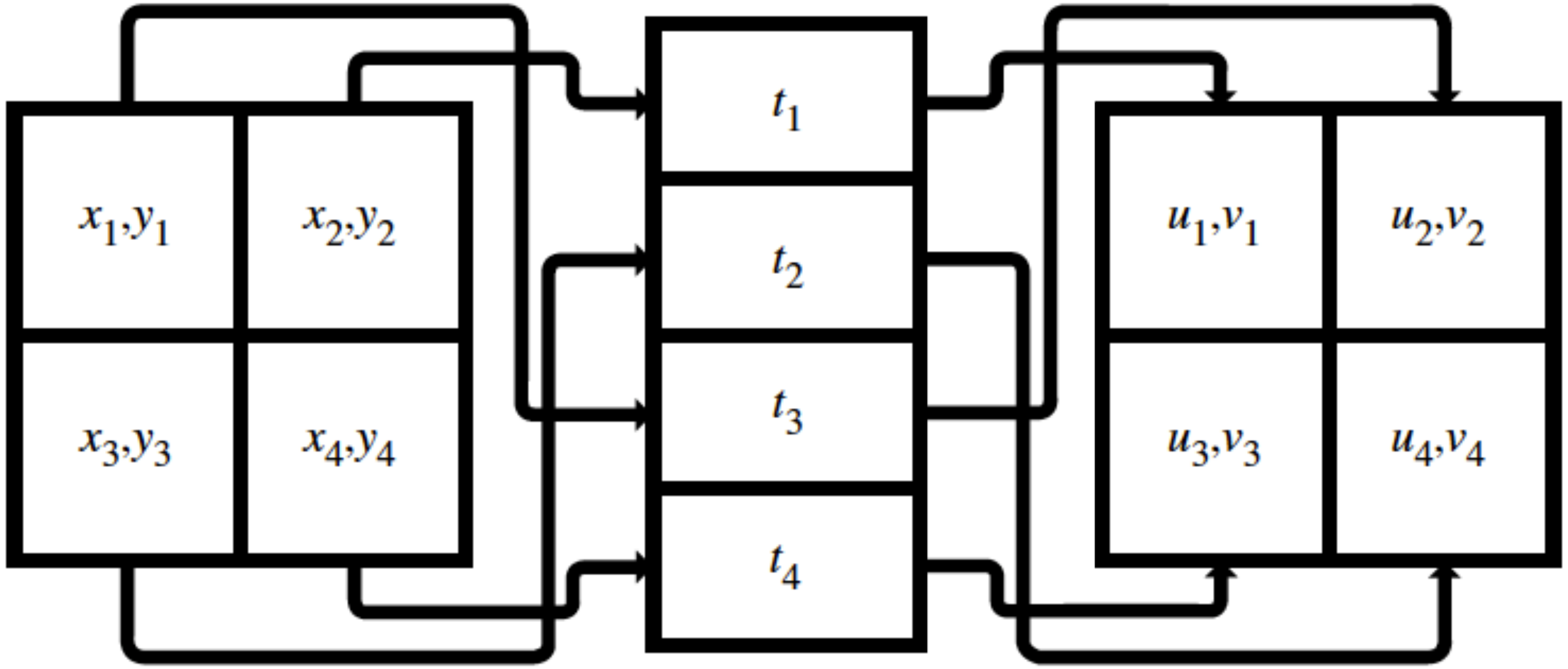
- 2D jittered (x,y) for pixel
- 2D jittered (u,v) for lens
- 1D jittered (t) for time



Uncorrelated Jitter [Cook et al. 84]

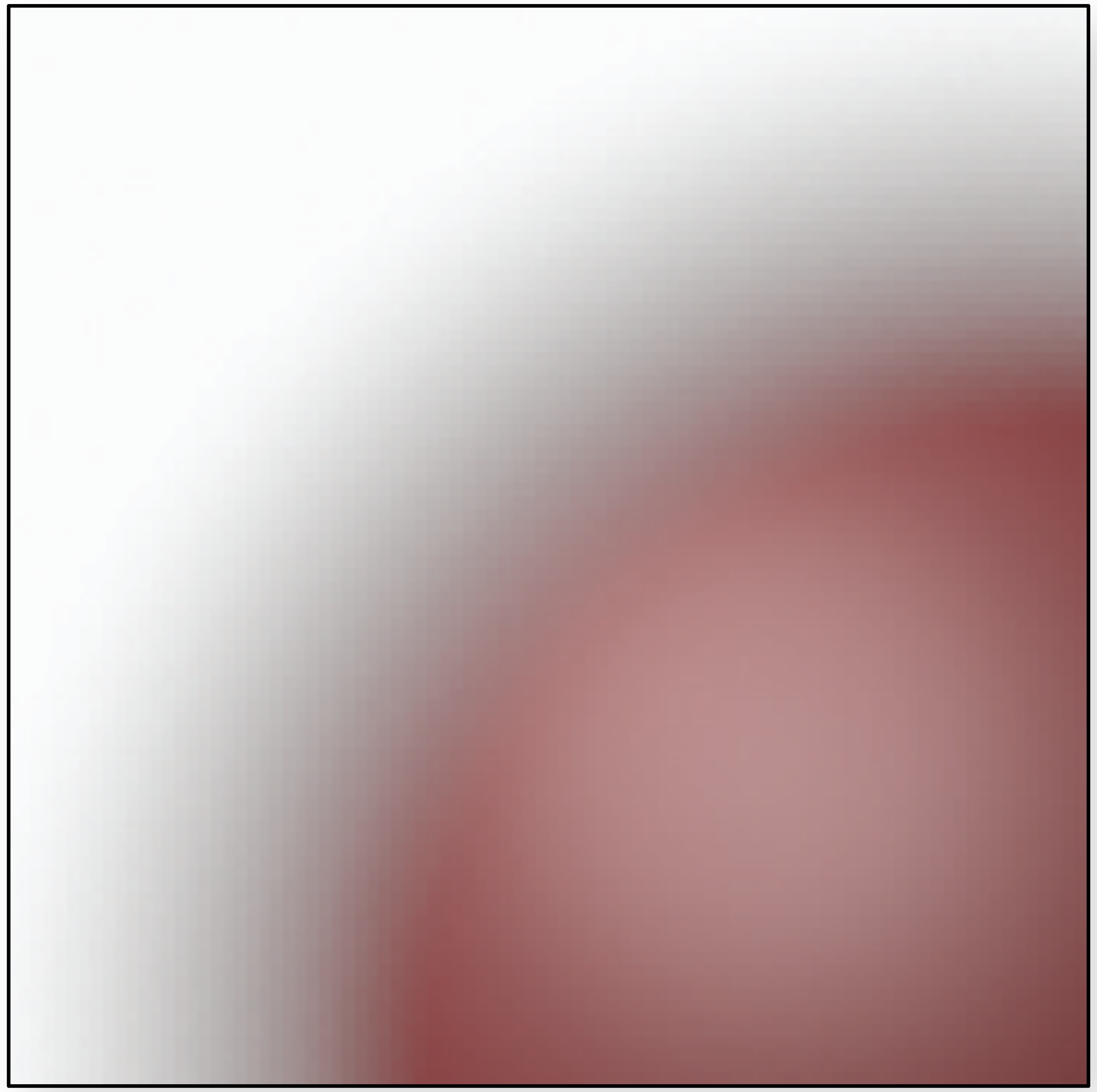
Compute stratified samples in sub-dimensions

- 2D jittered (x,y) for pixel
- 2D jittered (u,v) for lens
- 1D jittered (t) for time
- combine dimensions in random order

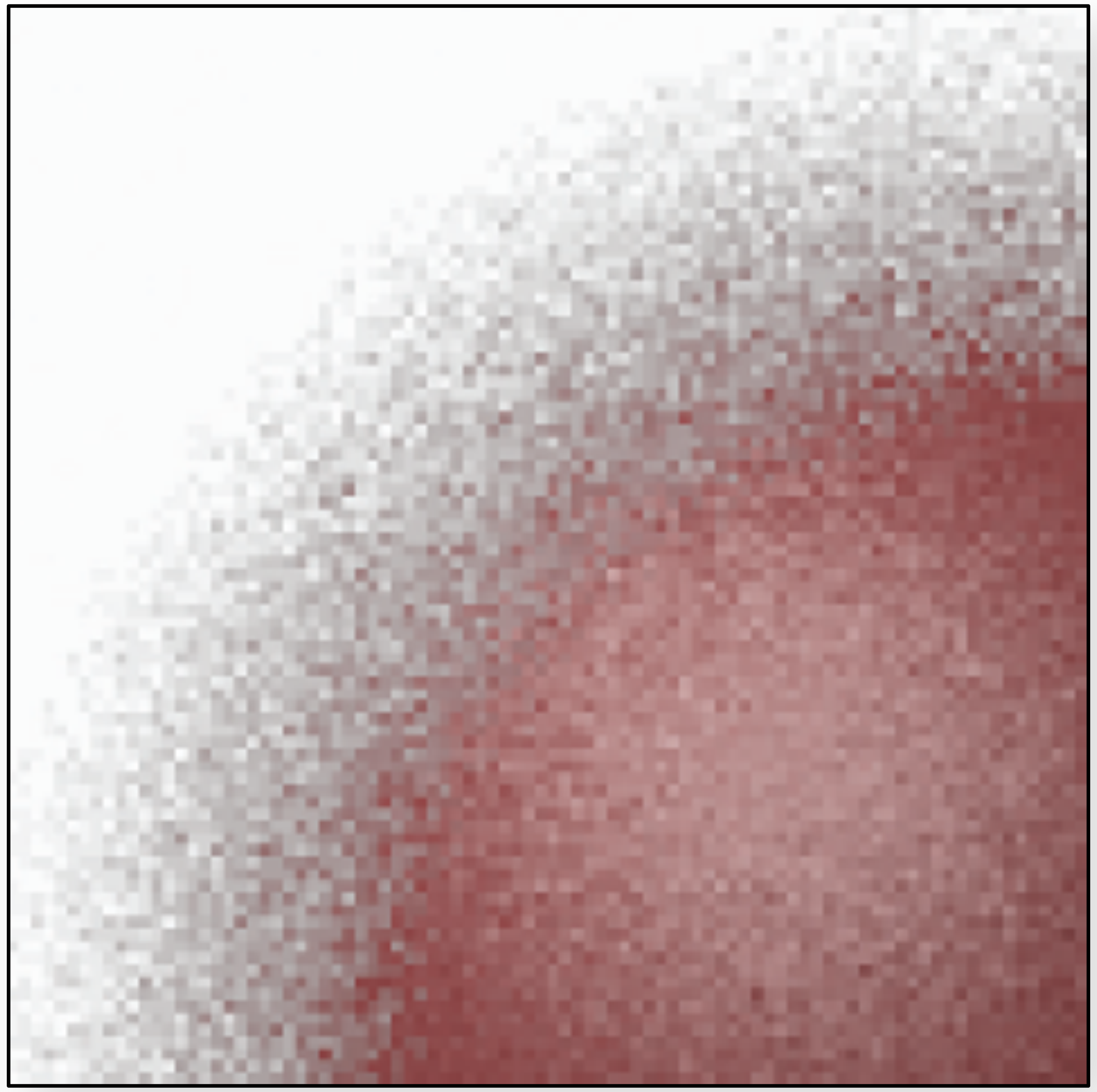


Depth of Field (4D)

Reference



Random Sampling



Uncorrelated Jitter

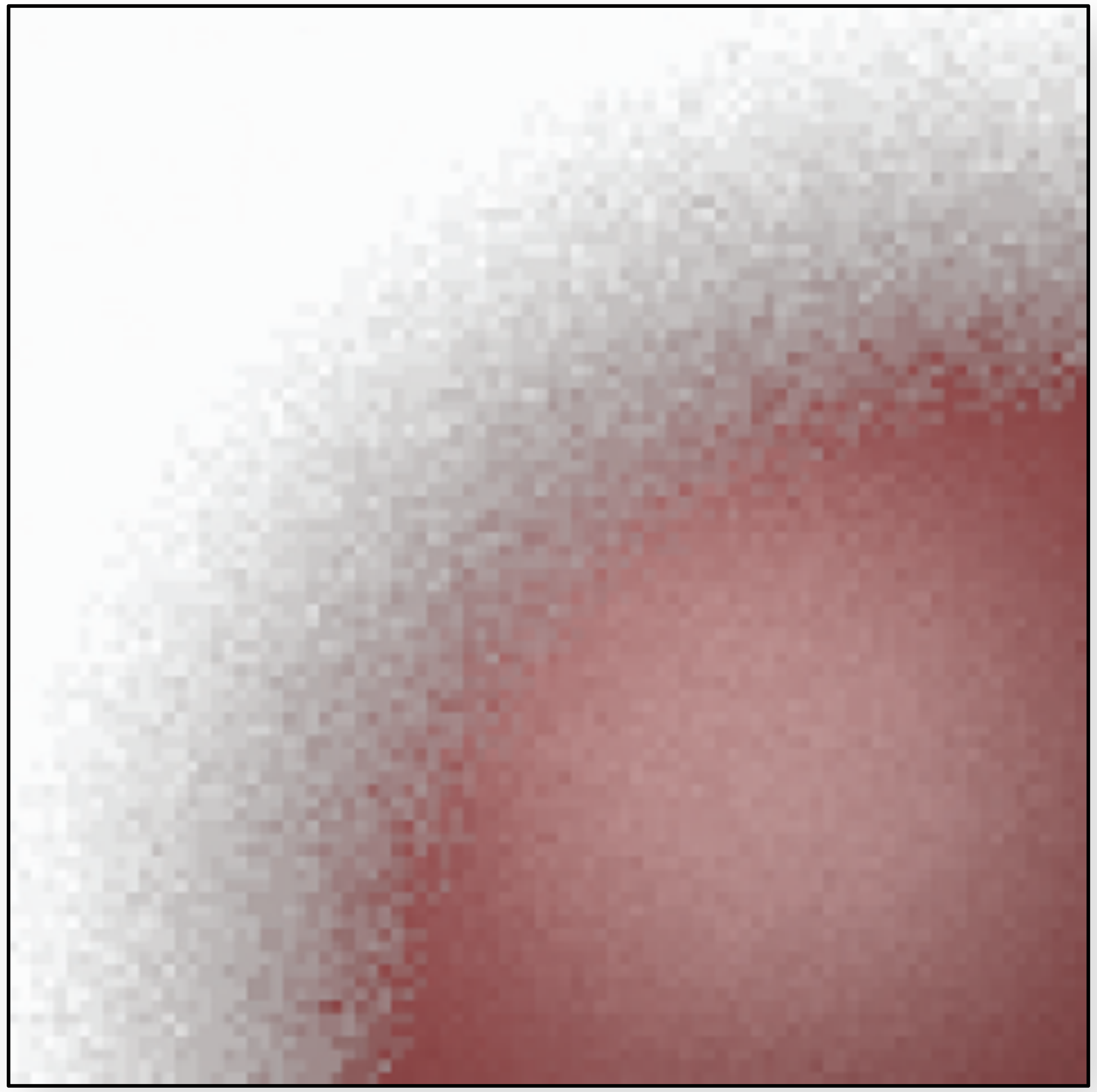


Image source: PBRTe2 [Pharr & Humphreys 2010]

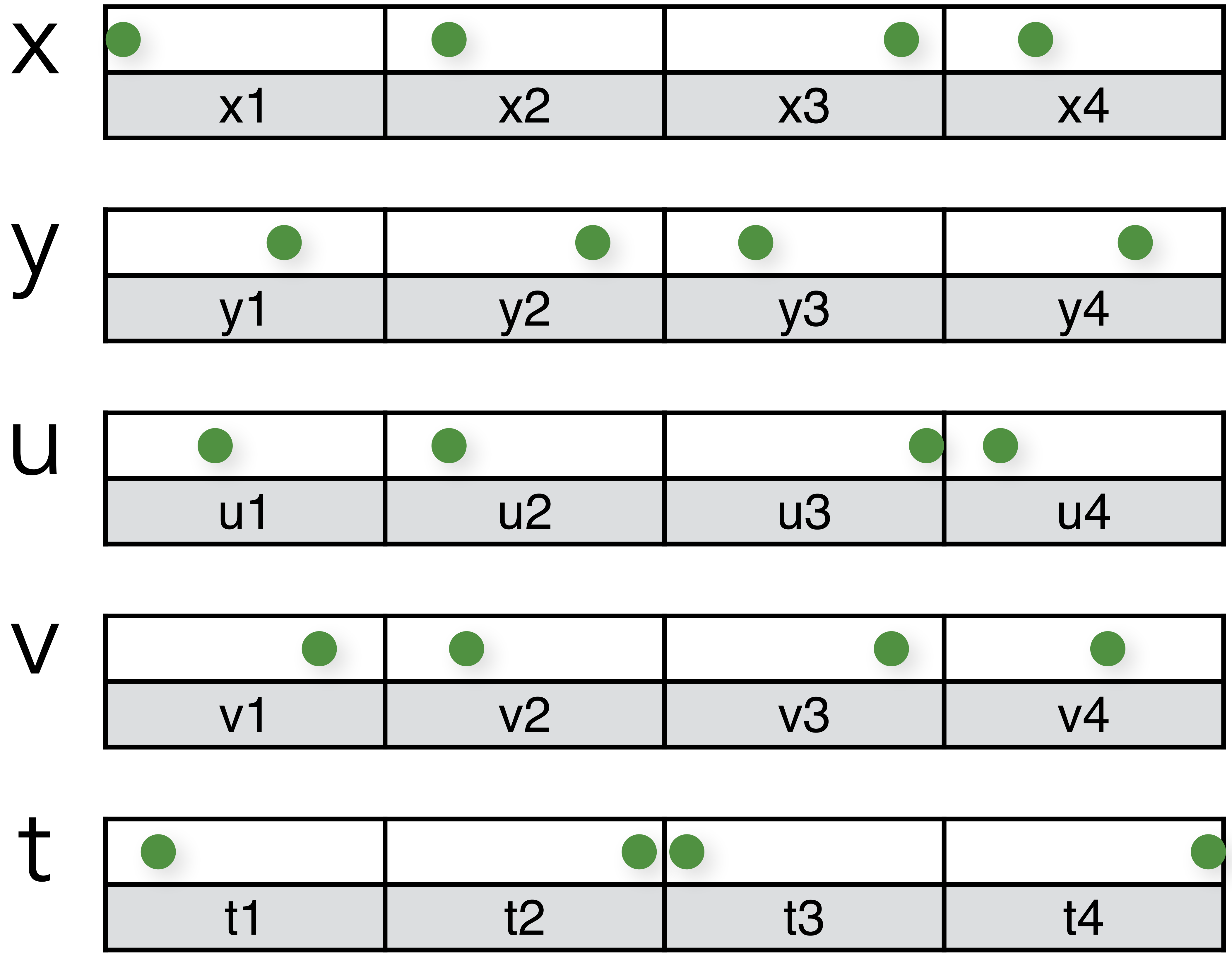
Uncorrelated Jitter ➔ Latin Hypercube

Stratify samples in each dimension separately

Uncorrelated Jitter ➔ Latin Hypercube

Stratify samples in each dimension separately

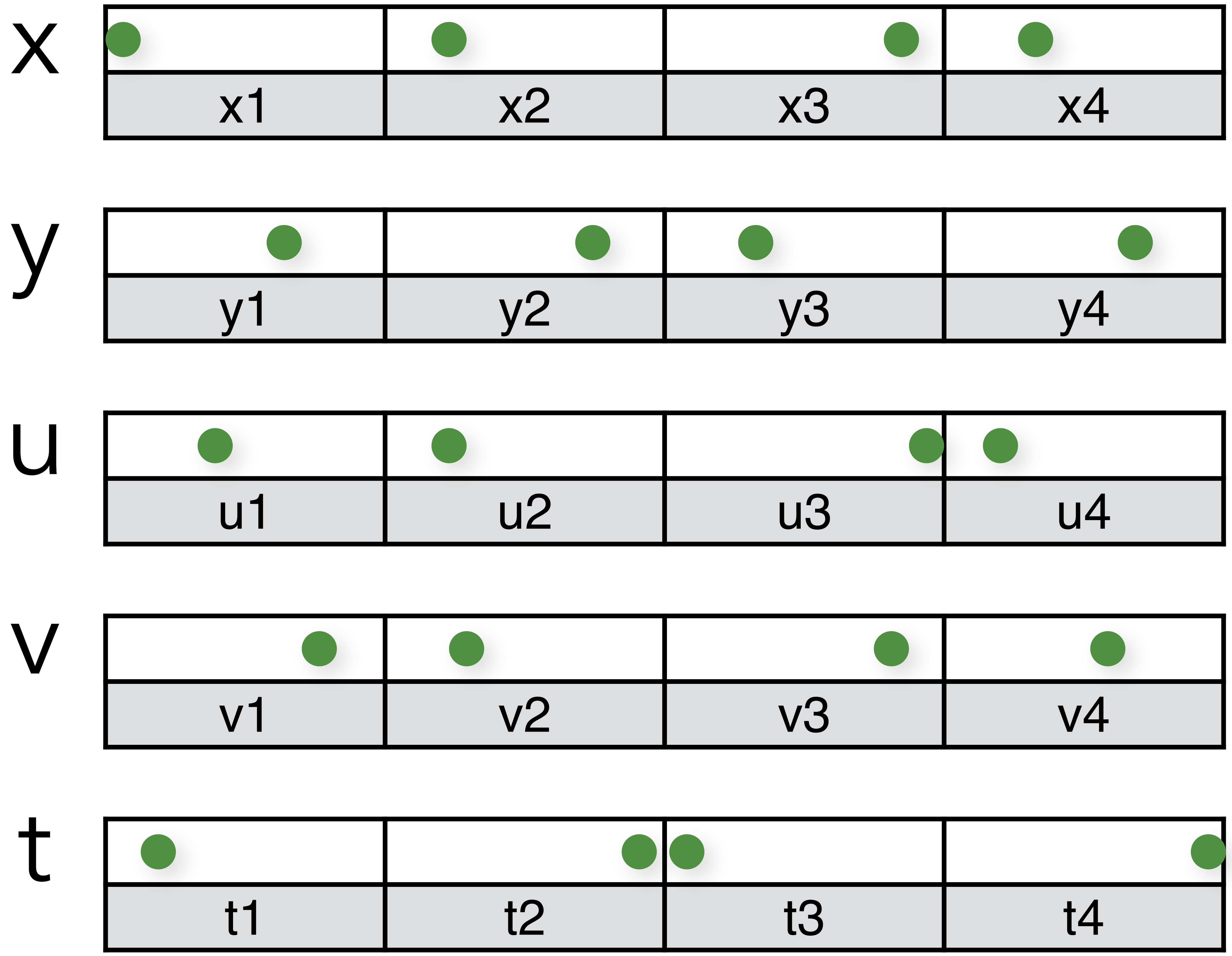
- for 5D: 5 separate 1D jittered point sets



Uncorrelated Jitter ➔ Latin Hypercube

Stratify samples in each dimension separately

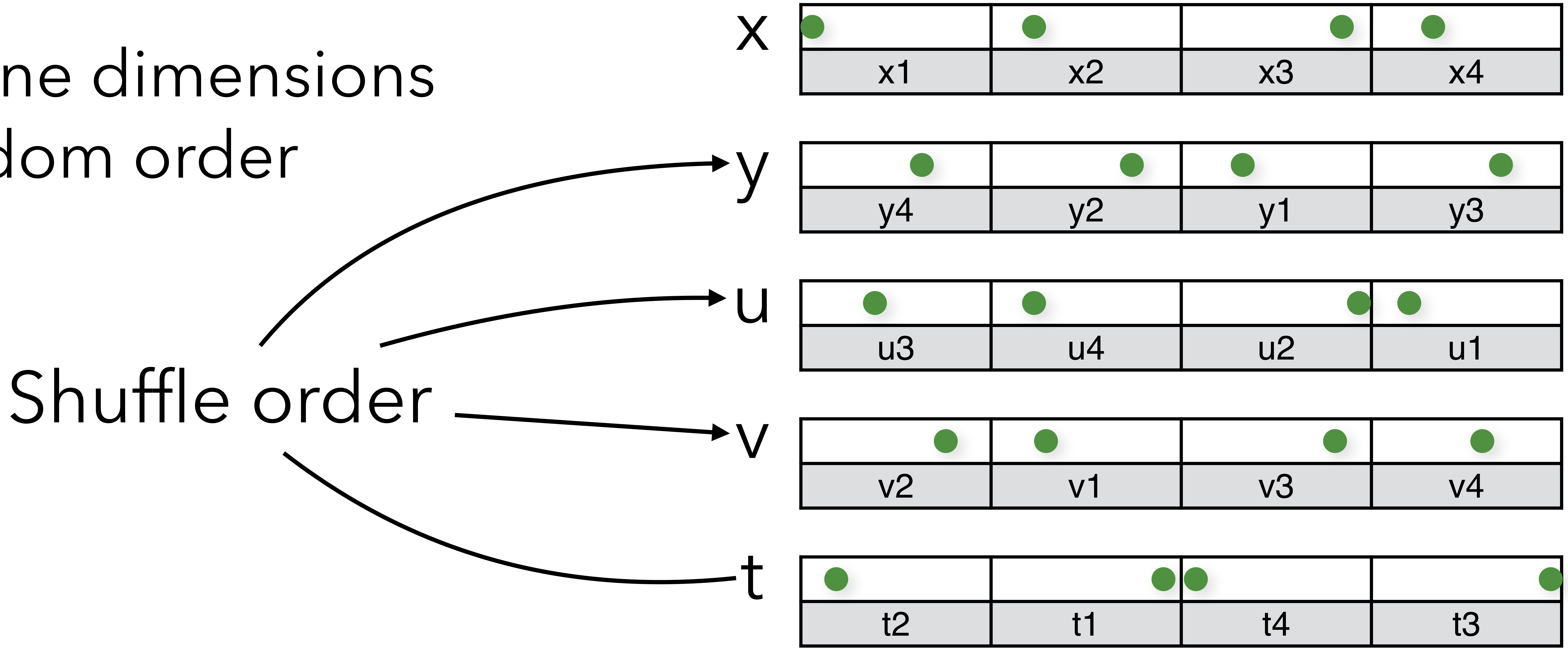
- for 5D: 5 separate 1D jittered point sets
- combine dimensions in random order



Uncorrelated Jitter ➔ Latin Hypercube

Stratify samples in each dimension separately

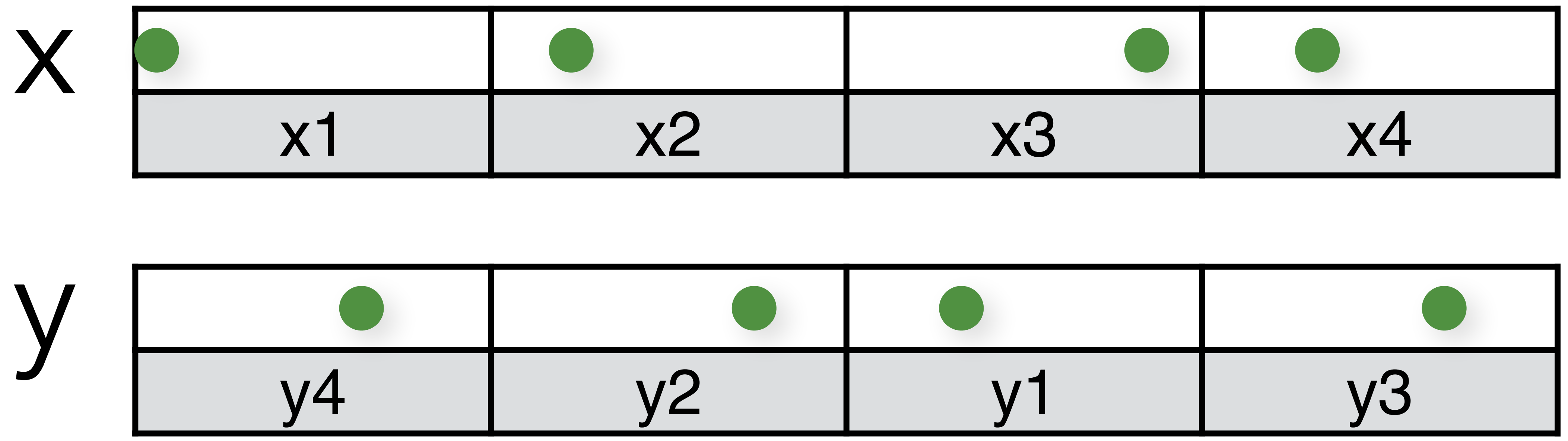
- for 5D: 5 separate 1D jittered point sets
- combine dimensions in random order



N-Rooks = 2D Latin Hypercube [Shirley 91]

Stratify samples in each dimension separately

- for **2D**: **2** separate 1D jittered point sets
- combine dimensions in random order



Latin Hypercube (N-Rooks) Sampling

[Shirley 91]

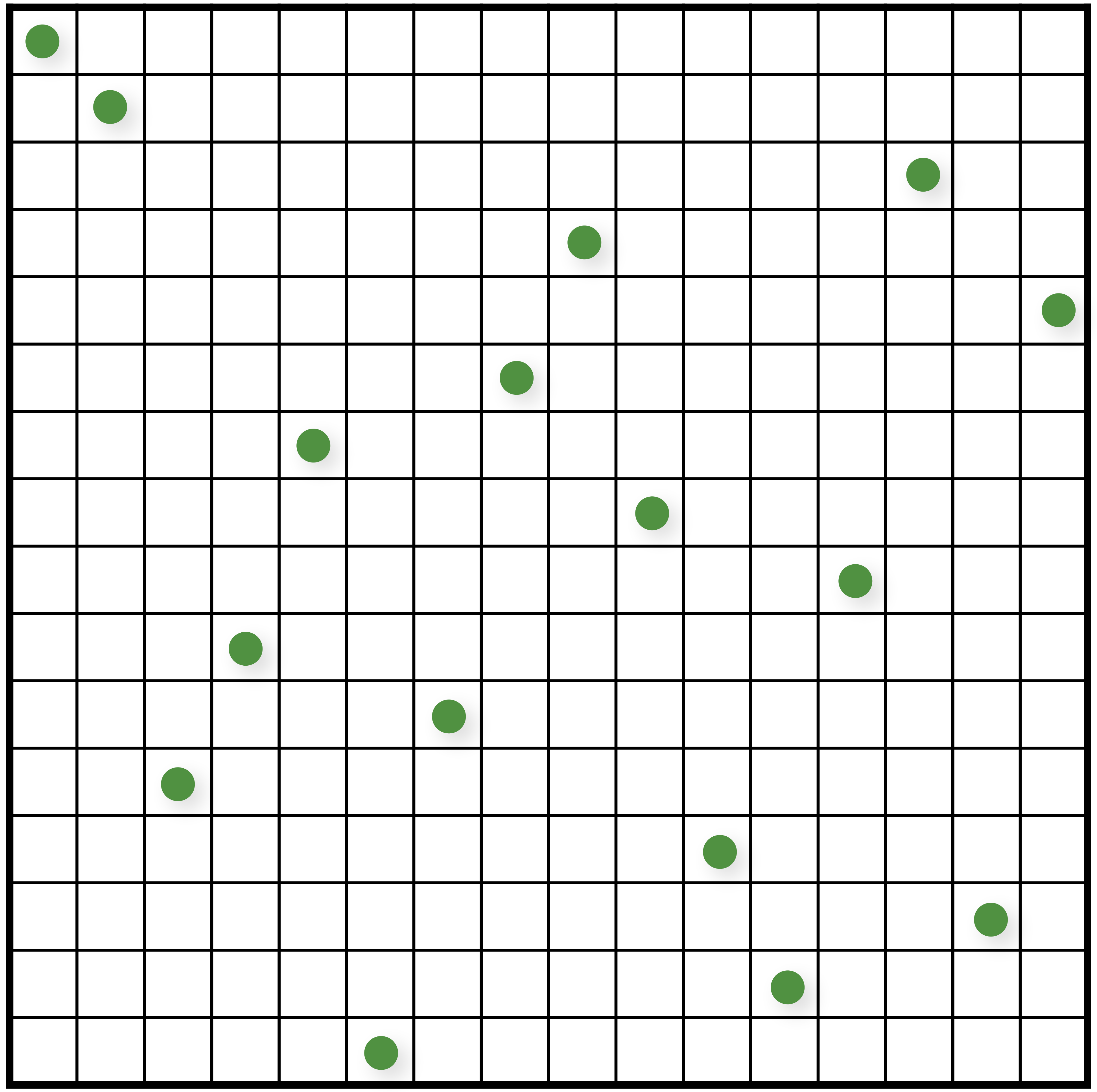
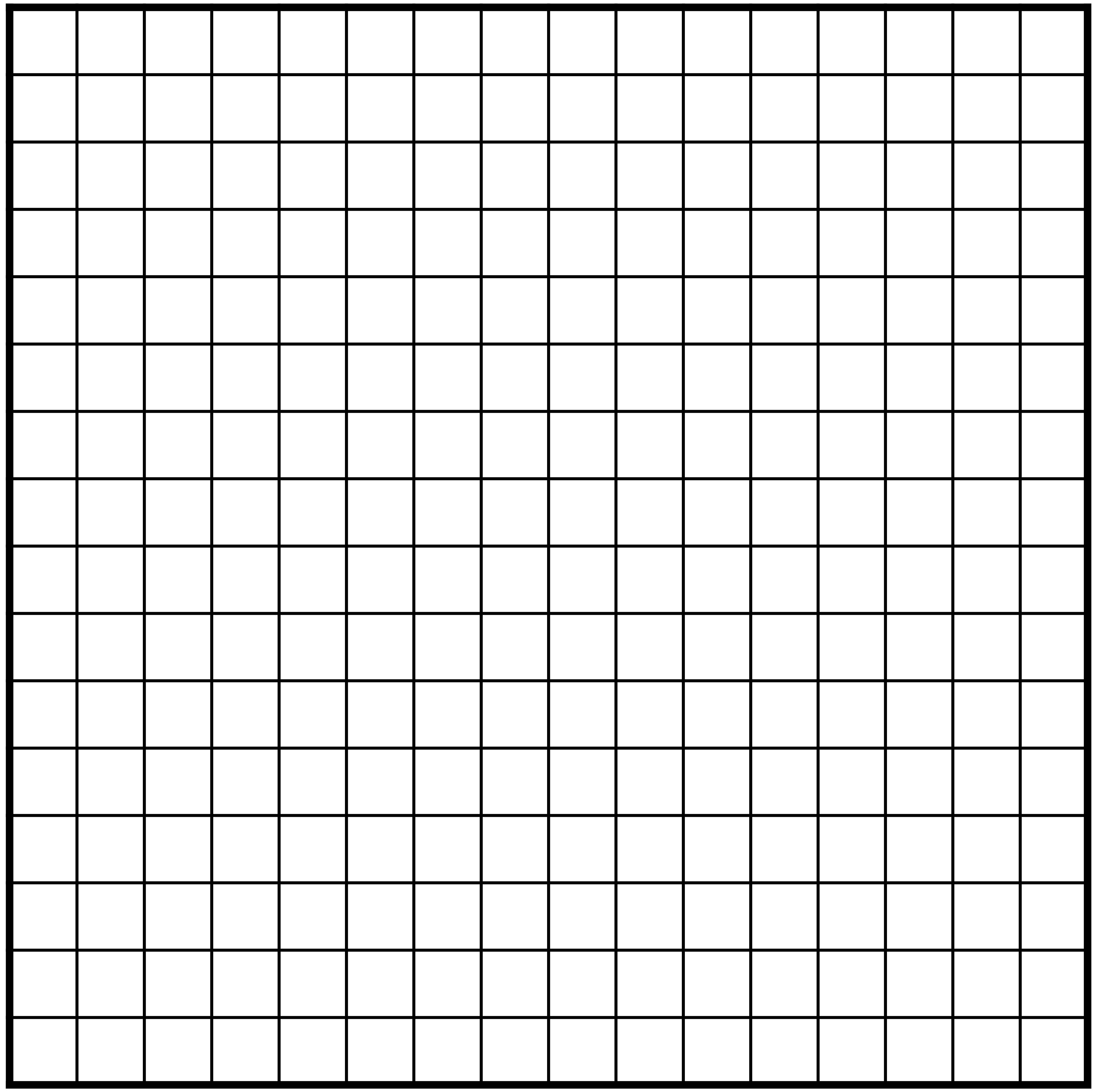


Image source: Michael Maggs, CC BY-SA 2.5

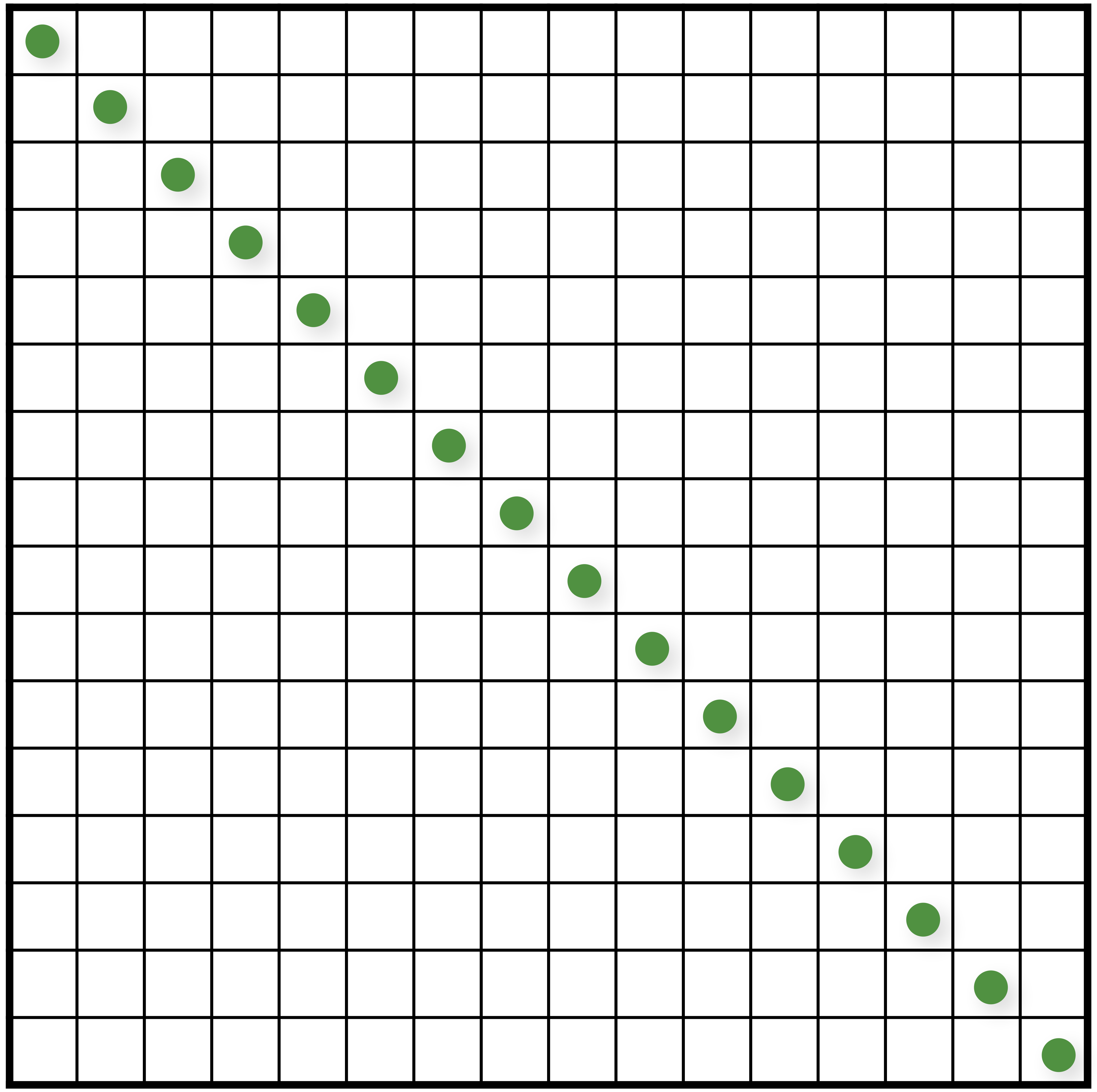
Latin Hypercube (N-Rooks) Sampling

```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
        for (uint i = 0; i < numS; i++)
                samples(d,i) = (i + randf())/numS;

// shuffle each dimension independently
for (uint d = 0; d < numDimensions; d++)
        shuffle(samples(d,:));
```



Latin Hypercube (N-Rooks) Sampling

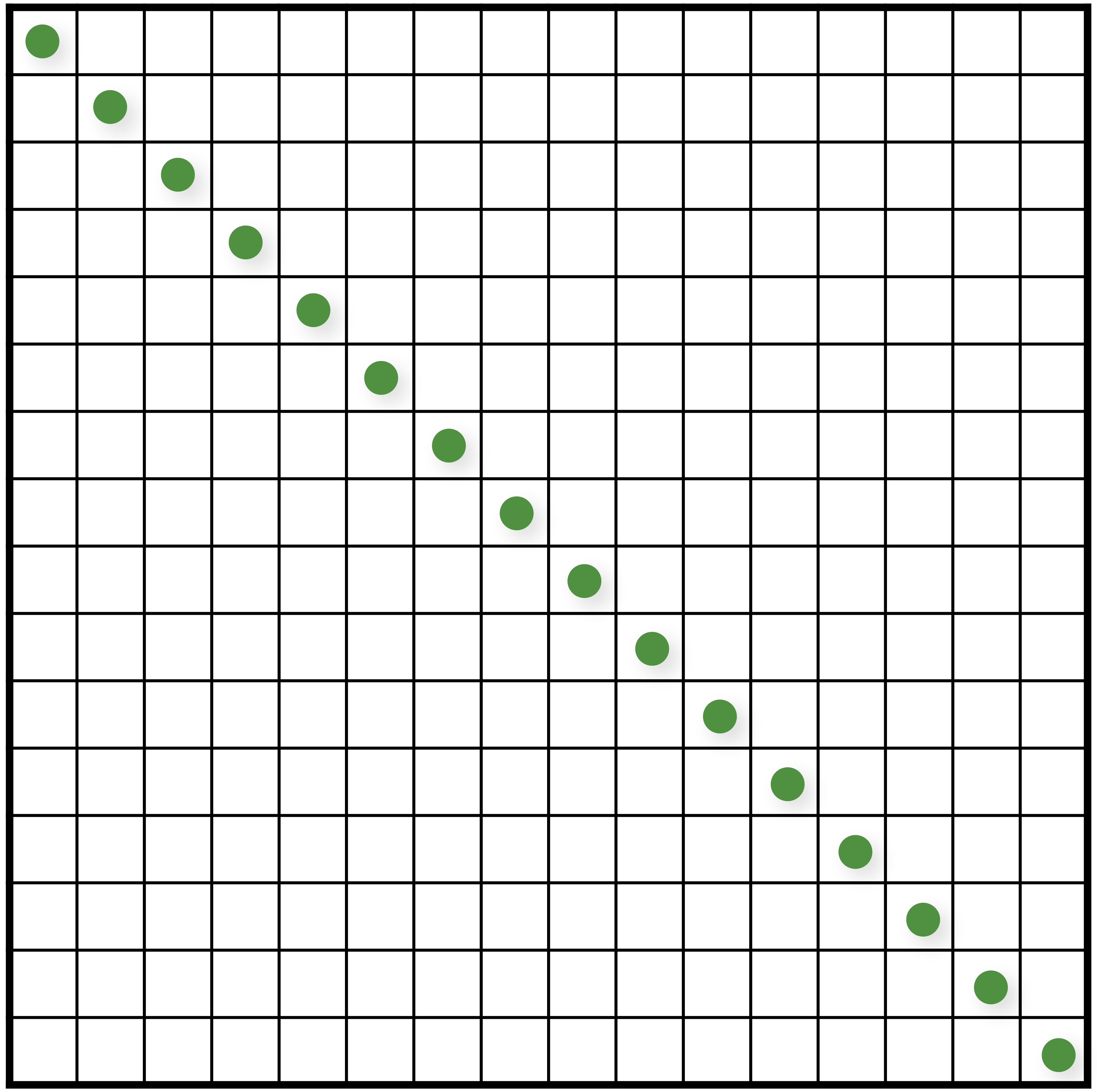
```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
    for (uint i = 0; i < numS; i++)
        samples(d,i) = (i + randf())/numS;

// shuffle each dimension independently
for (uint d = 0; d < numDimensions; d++)
    shuffle(samples(d,:));
```



Initialize

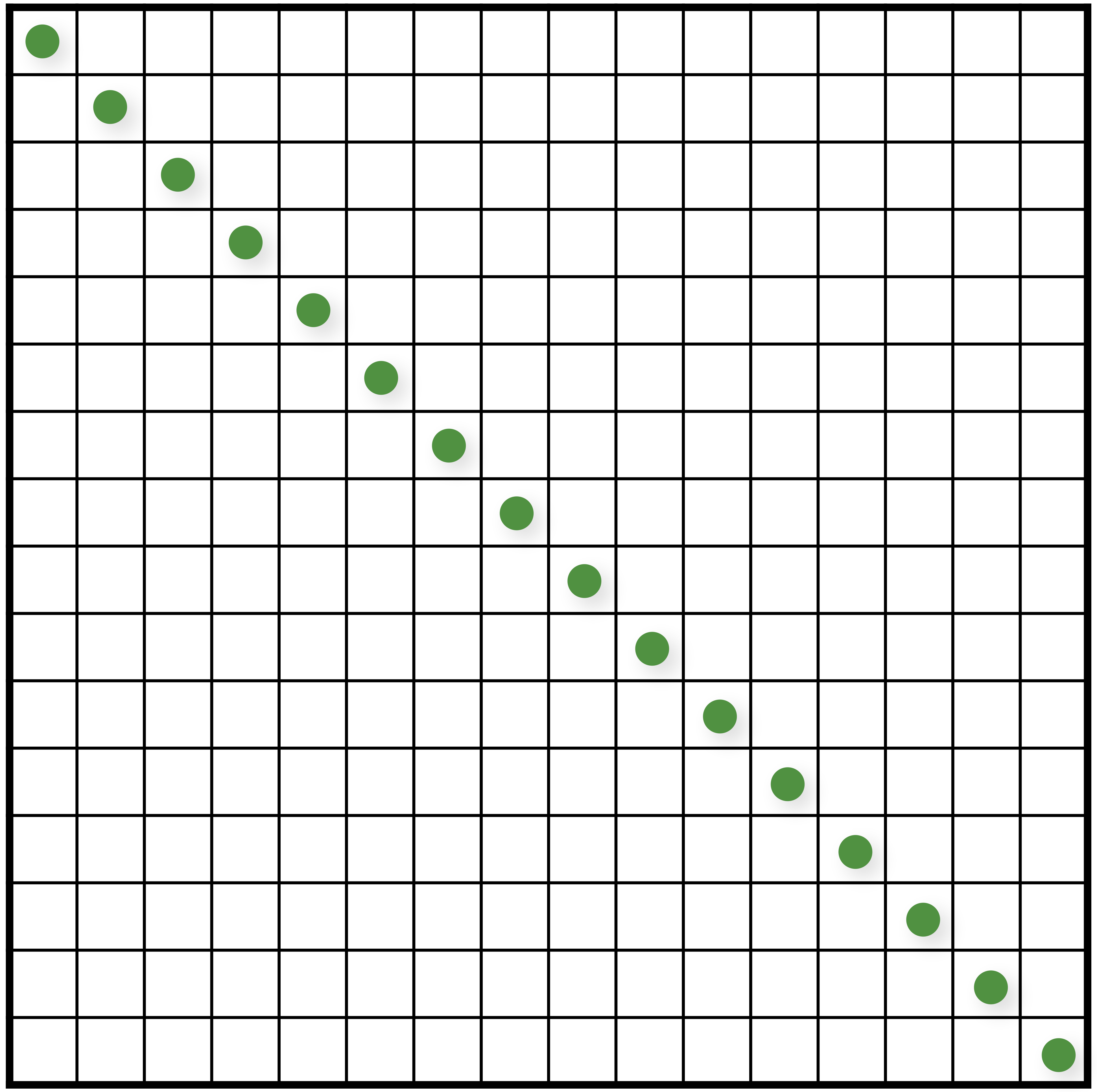
Latin Hypercube (N-Rooks) Sampling

```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
    for (uint i = 0; i < numS; i++)
        samples(d,i) = (i + randf())/numS;

// shuffle each dimension independently
for (uint d = 0; d < numDimensions; d++)
    shuffle(samples(d,:));
```



Latin Hypercube (N-Rooks) Sampling

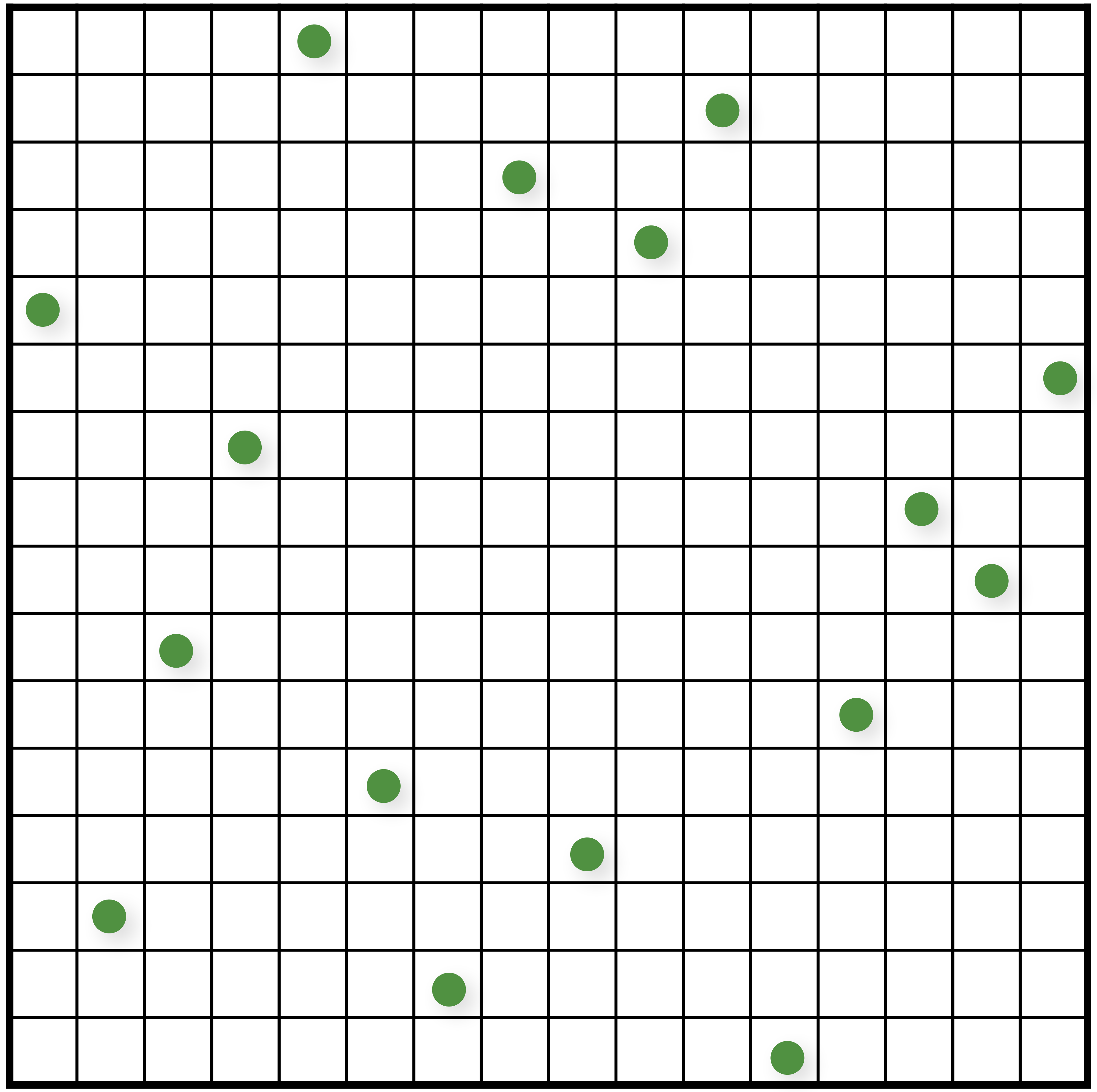
```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
        for (uint i = 0; i < numS; i++)
                samples(d,i) = (i + randf())/numS;

// shuffle each dimension independently
for (uint d = 0; d < numDimensions; d++)
        shuffle(samples(d,:));
```



Shuffle rows

Latin Hypercube (N-Rooks) Sampling

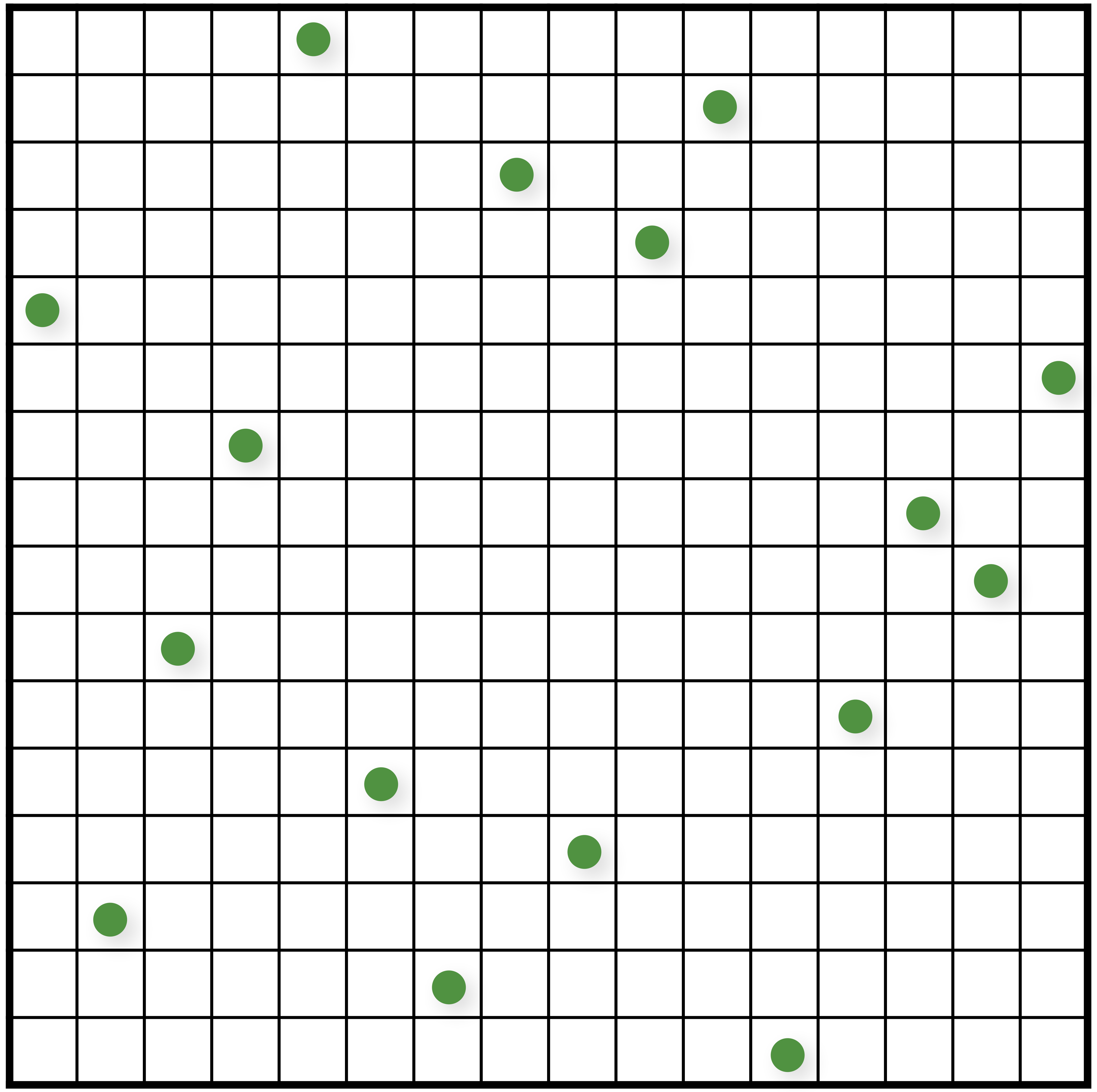
```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
        for (uint i = 0; i < numS; i++)
                samples(d,i) = (i + randf())/numS;

// shuffle each dimension independently
for (uint d = 0; d < numDimensions; d++)
        shuffle(samples(d,:));
```



Shuffle rows

Latin Hypercube (N-Rooks) Sampling

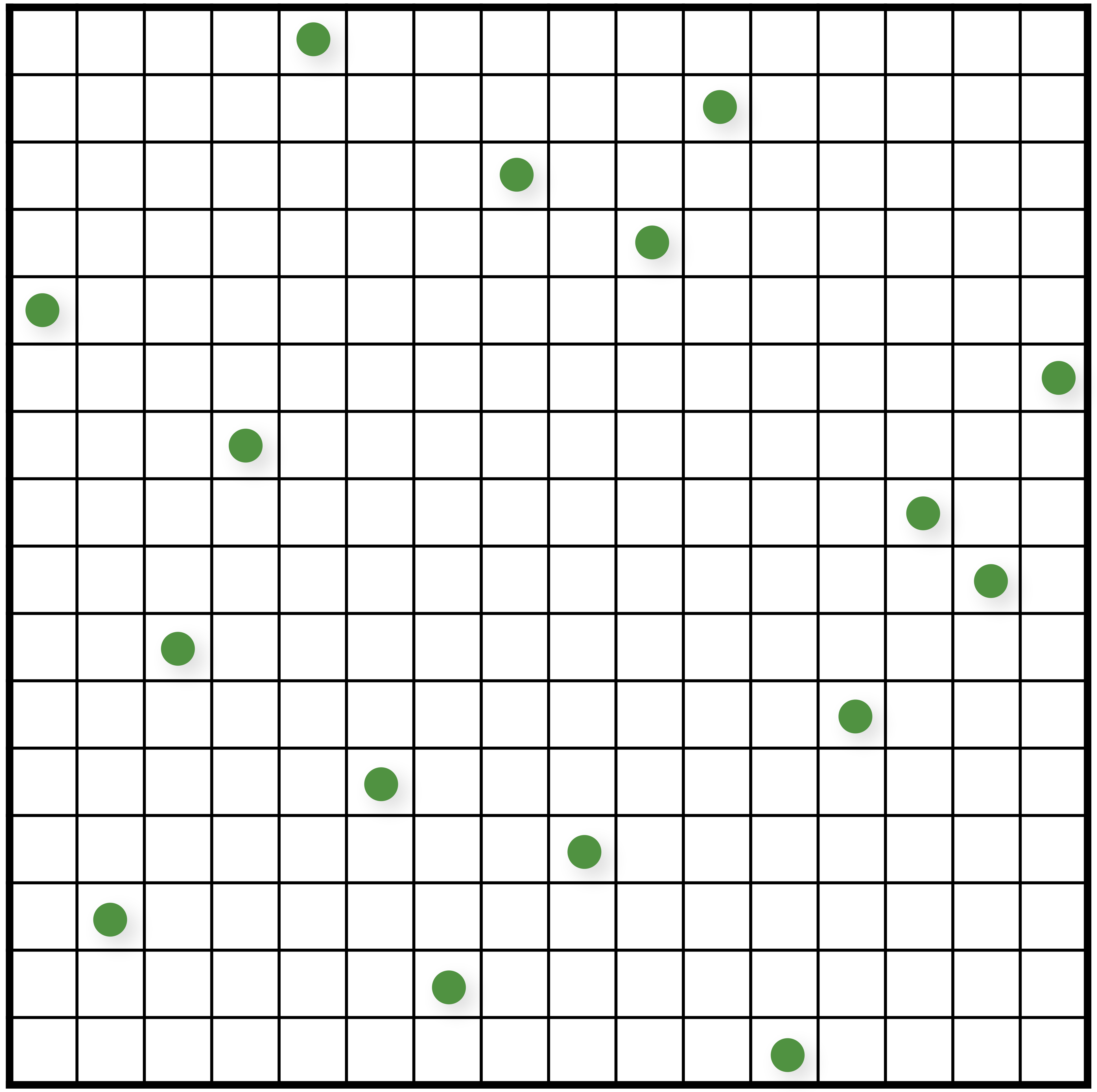
```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
    for (uint i = 0; i < numS; i++)
        samples(d,i) = (i + randf())/numS;

// shuffle each dimension independently
for (uint d = 0; d < numDimensions; d++)
    shuffle(samples(d,:));
```



Shuffle rows

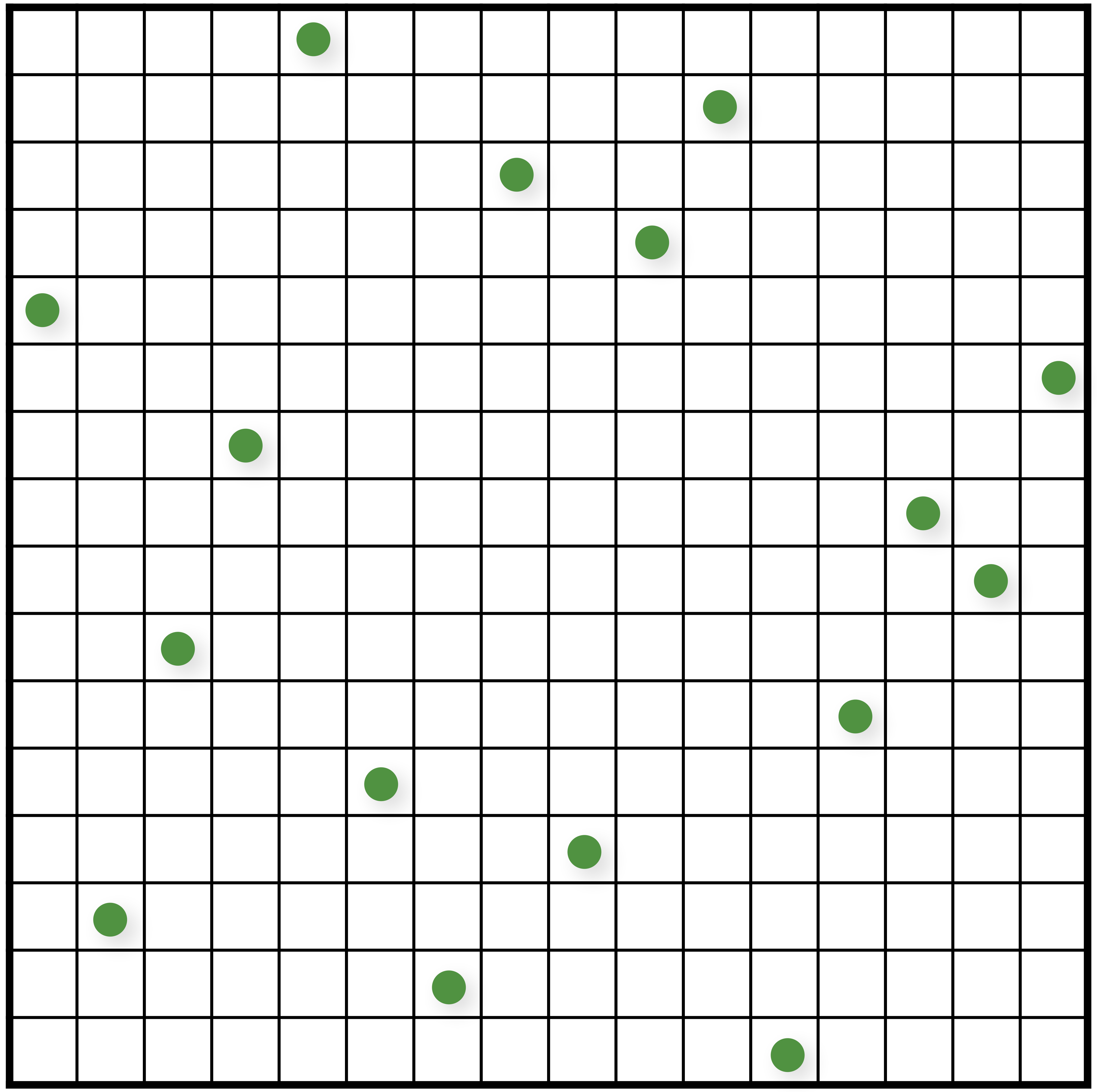
Latin Hypercube (N-Rooks) Sampling

```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
    for (uint i = 0; i < numS; i++)
        samples(d,i) = (i + randf())/numS;

// shuffle each dimension independently
for (uint d = 0; d < numDimensions; d++)
    shuffle(samples(d,:));
```



Latin Hypercube (N-Rooks) Sampling

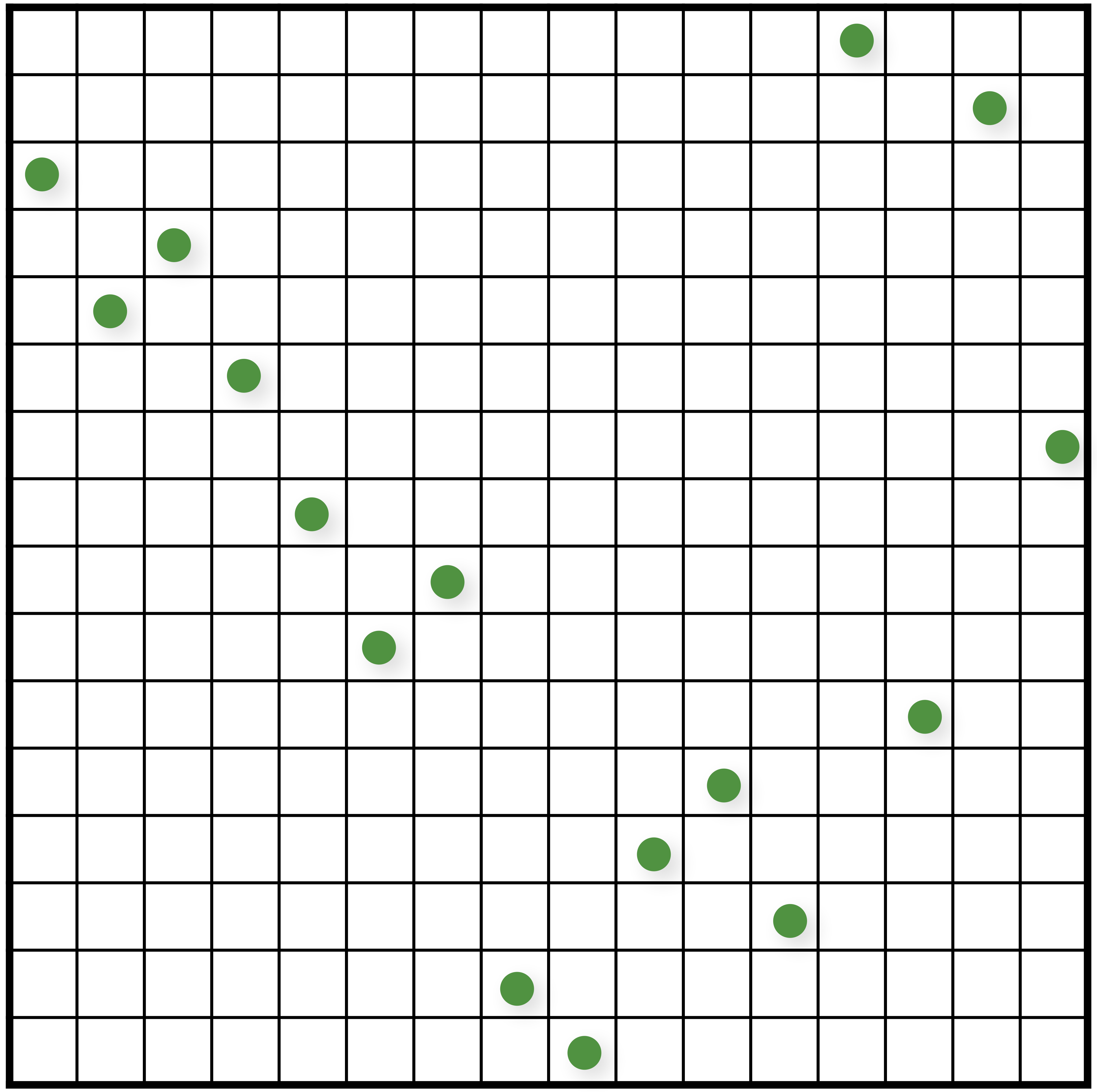
```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
    for (uint i = 0; i < numS; i++)
        samples(d,i) = (i + randf())/numS;

// shuffle each dimension independently
for (uint d = 0; d < numDimensions; d++)
    shuffle(samples(d,:));
```



Shuffle columns

Latin Hypercube (N-Rooks) Sampling

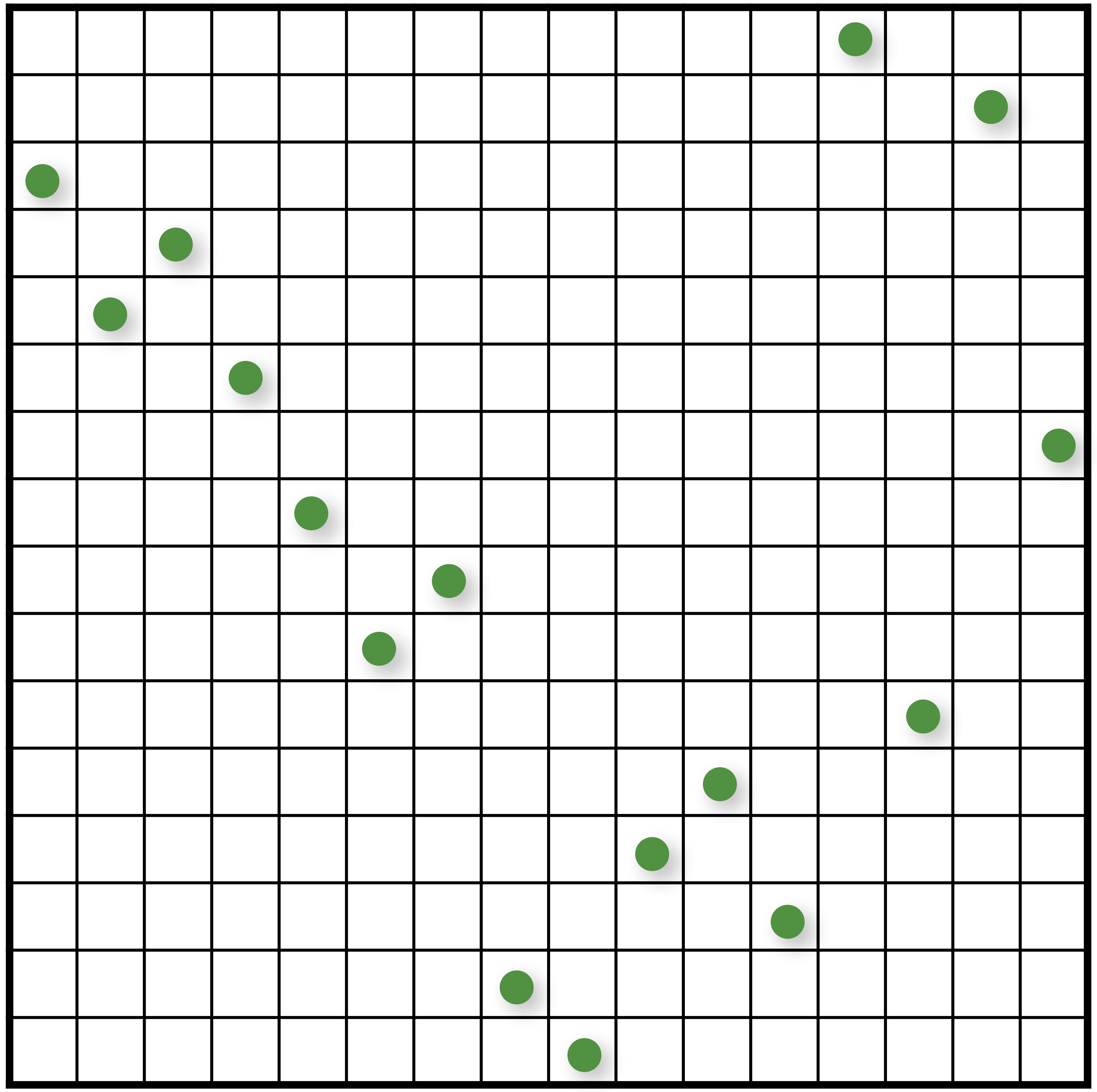
```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
        for (uint i = 0; i < numS; i++)
                samples(d,i) = (i + randf())/numS;

// shuffle each dimension independently
for (uint d = 0; d < numDimensions; d++)
        shuffle(samples(d,:));
```



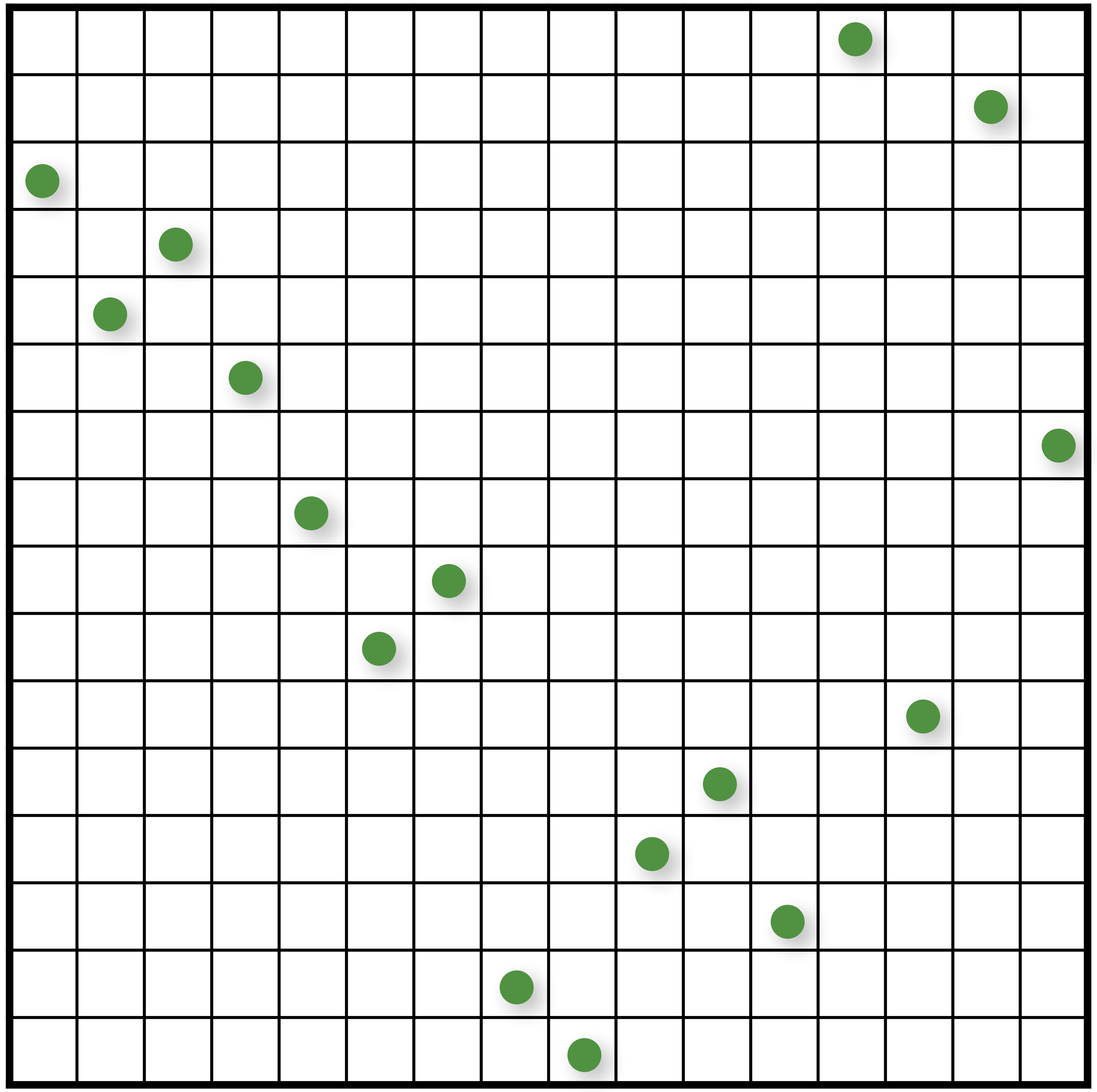
Shuffle columns

Latin Hypercube (N-Rooks) Sampling

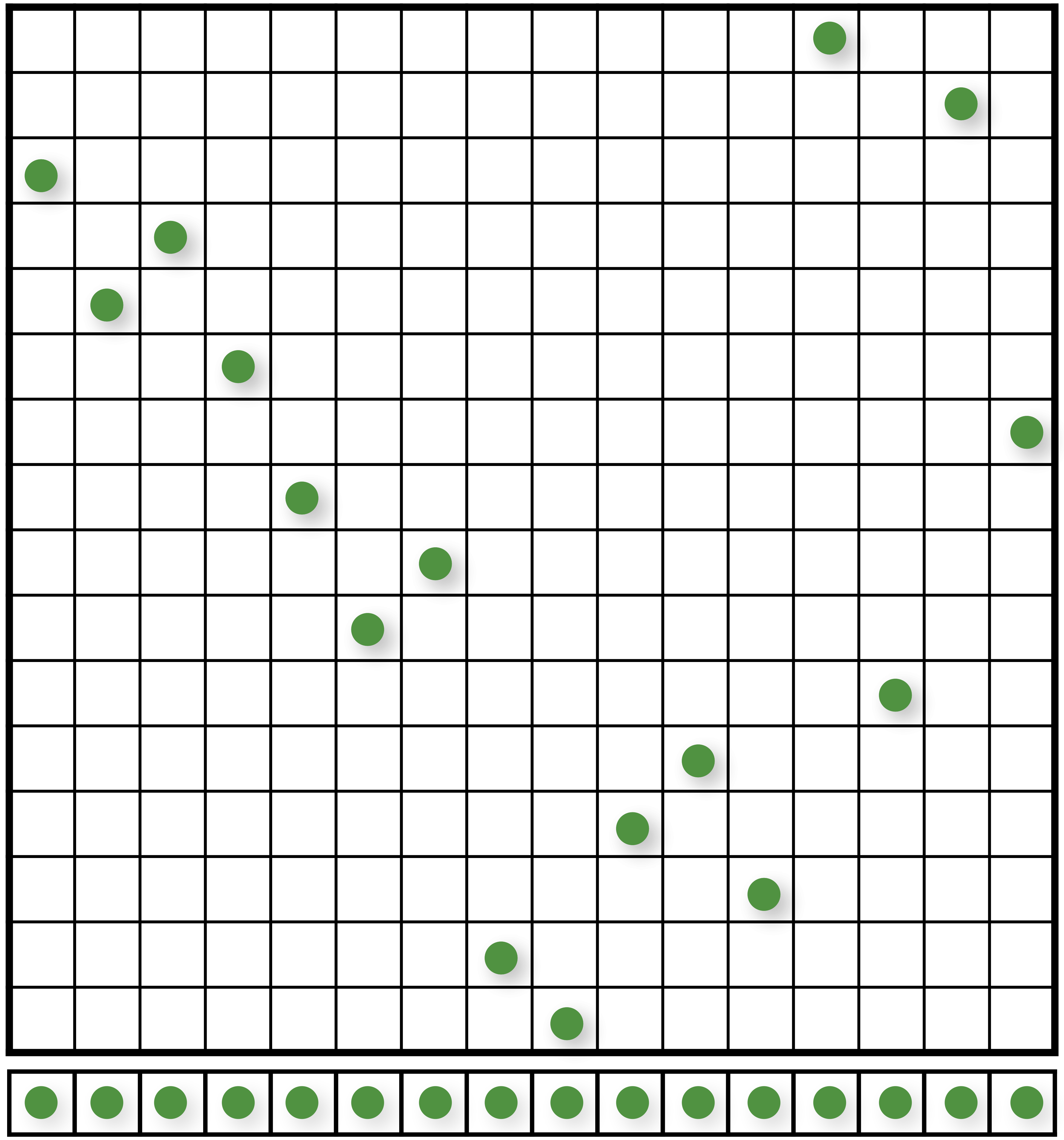
```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
	for (uint i = 0; i < numS; i++)
		samples(d,i) = (i + randf())/numS;

// shuffle each dimension independently
for (uint d = 0; d < numDimensions; d++)
	shuffle(samples(d,:));
```



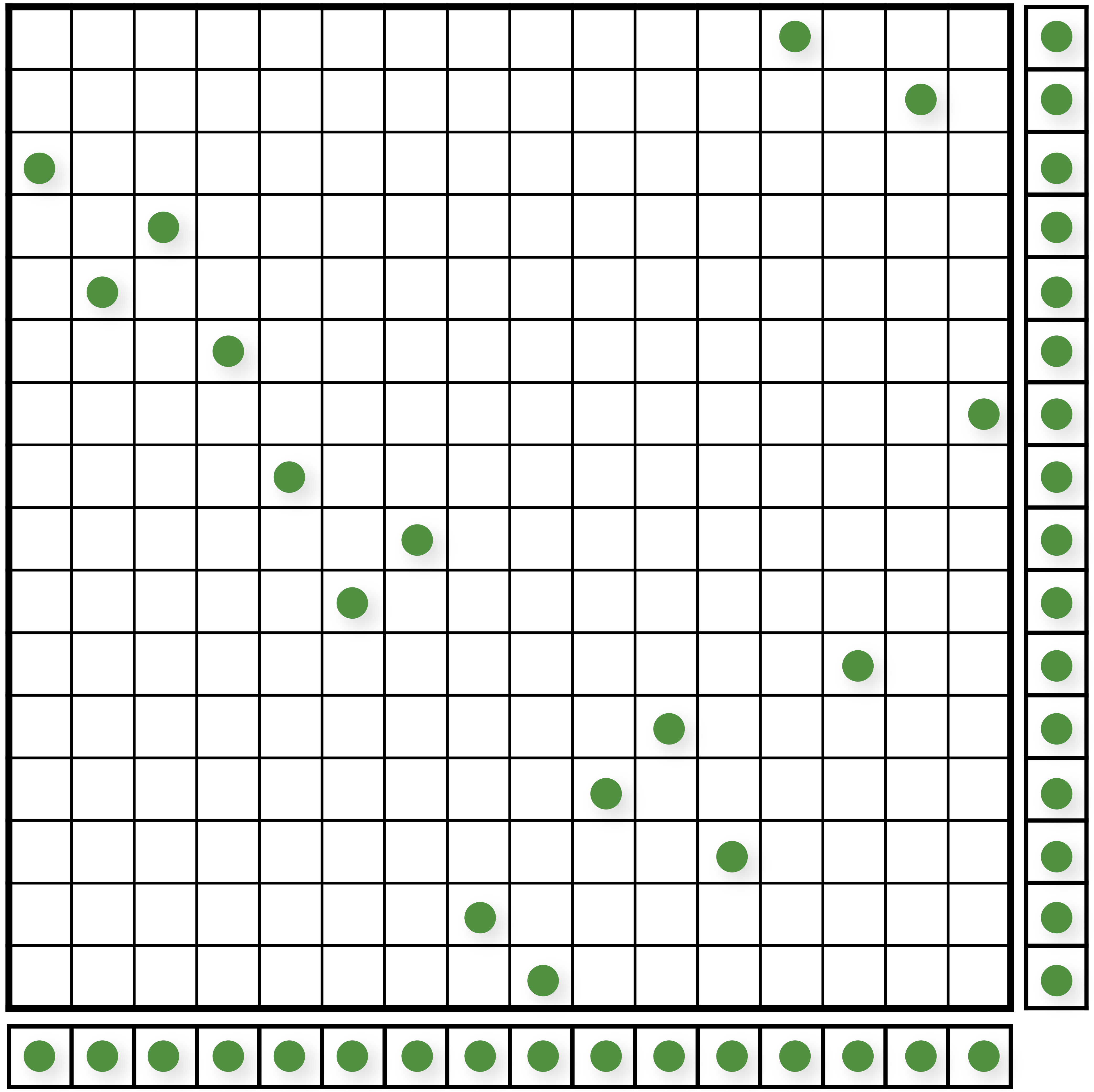
Latin Hypercube (N-Rooks) Sampling



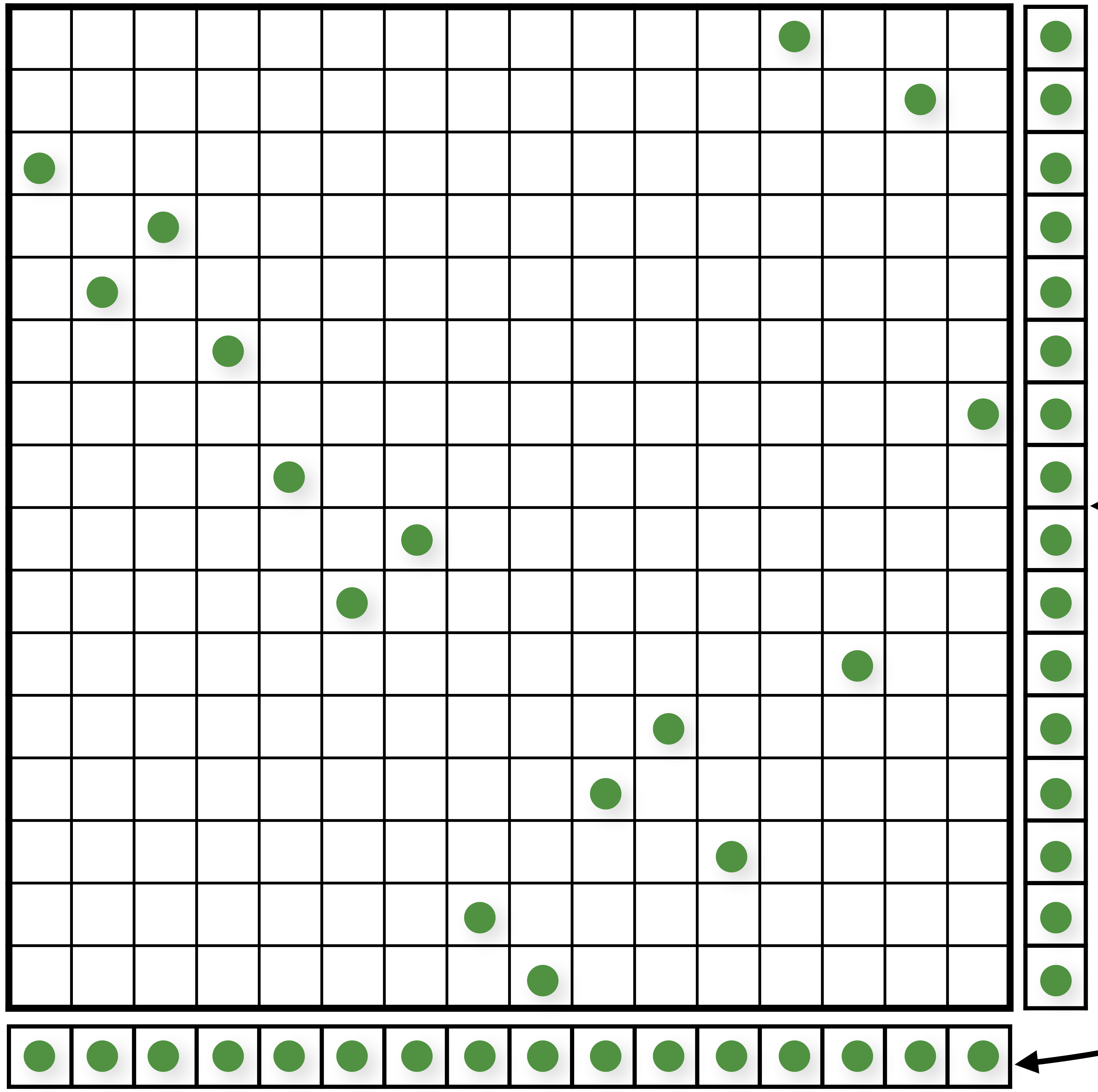
Latin Hypercube (N-Rooks) Sampling



Latin Hypercube (N-Rooks) Sampling

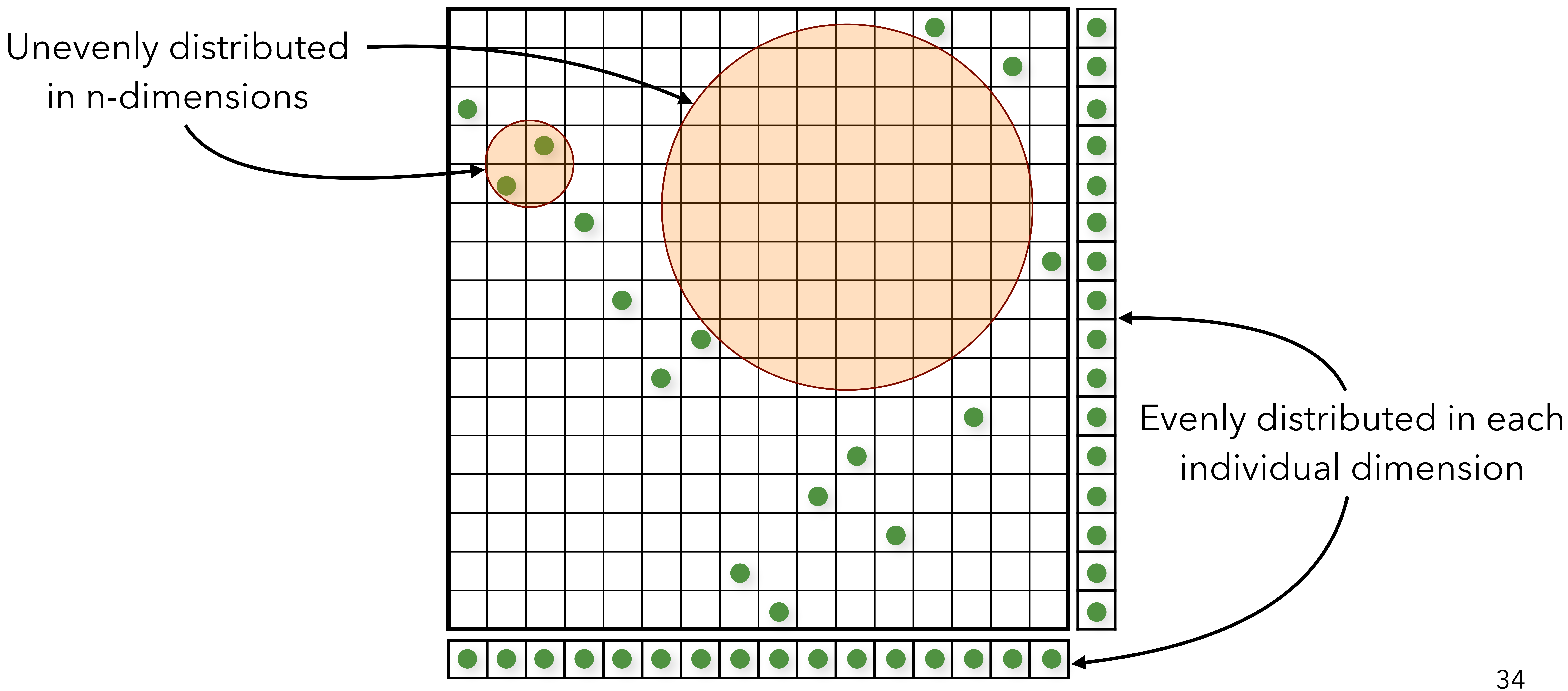


Latin Hypercube (N-Rooks) Sampling



Evenly distributed in each individual dimension

Latin Hypercube (N-Rooks) Sampling



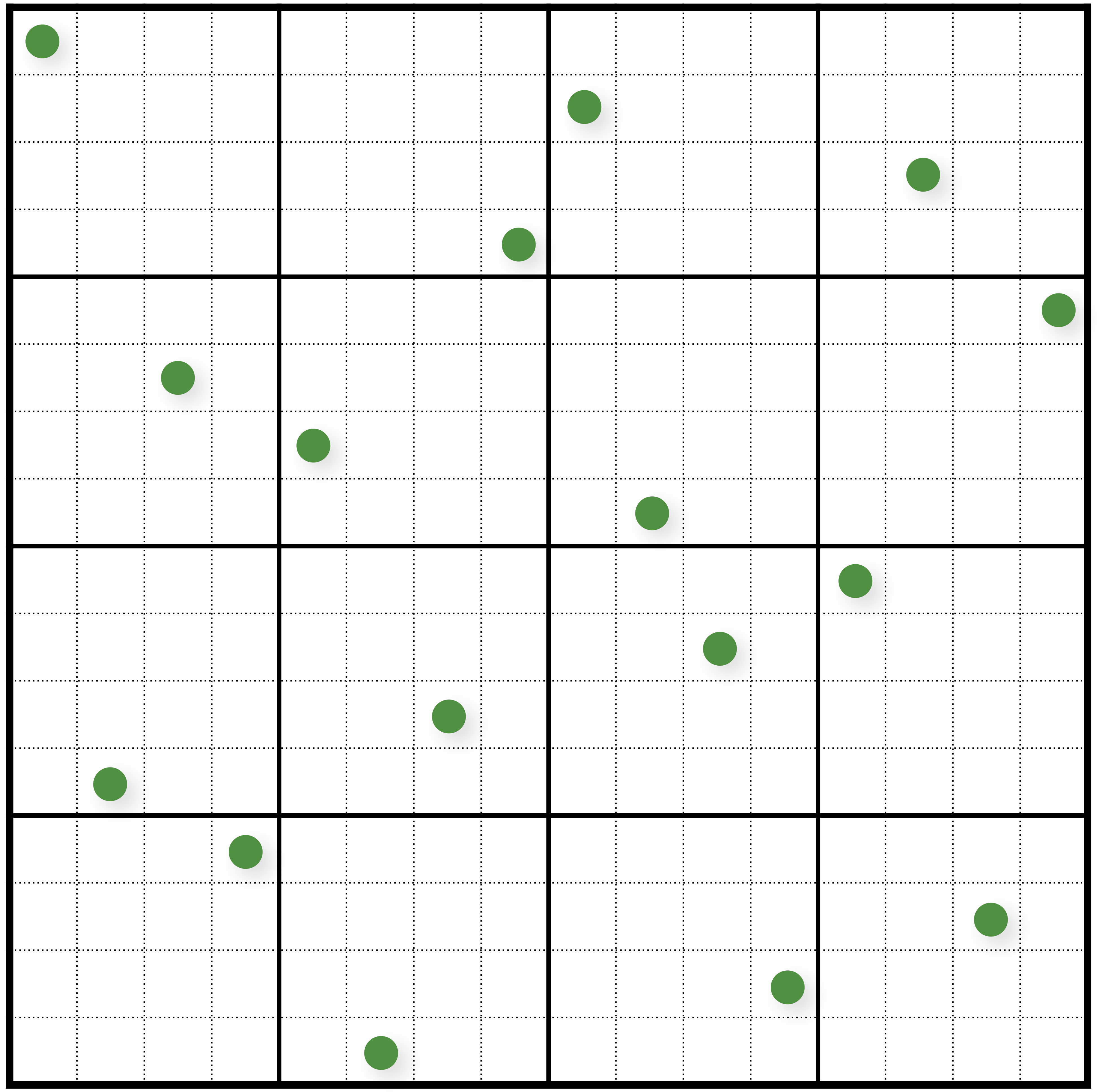
N-Rooks Sampling

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

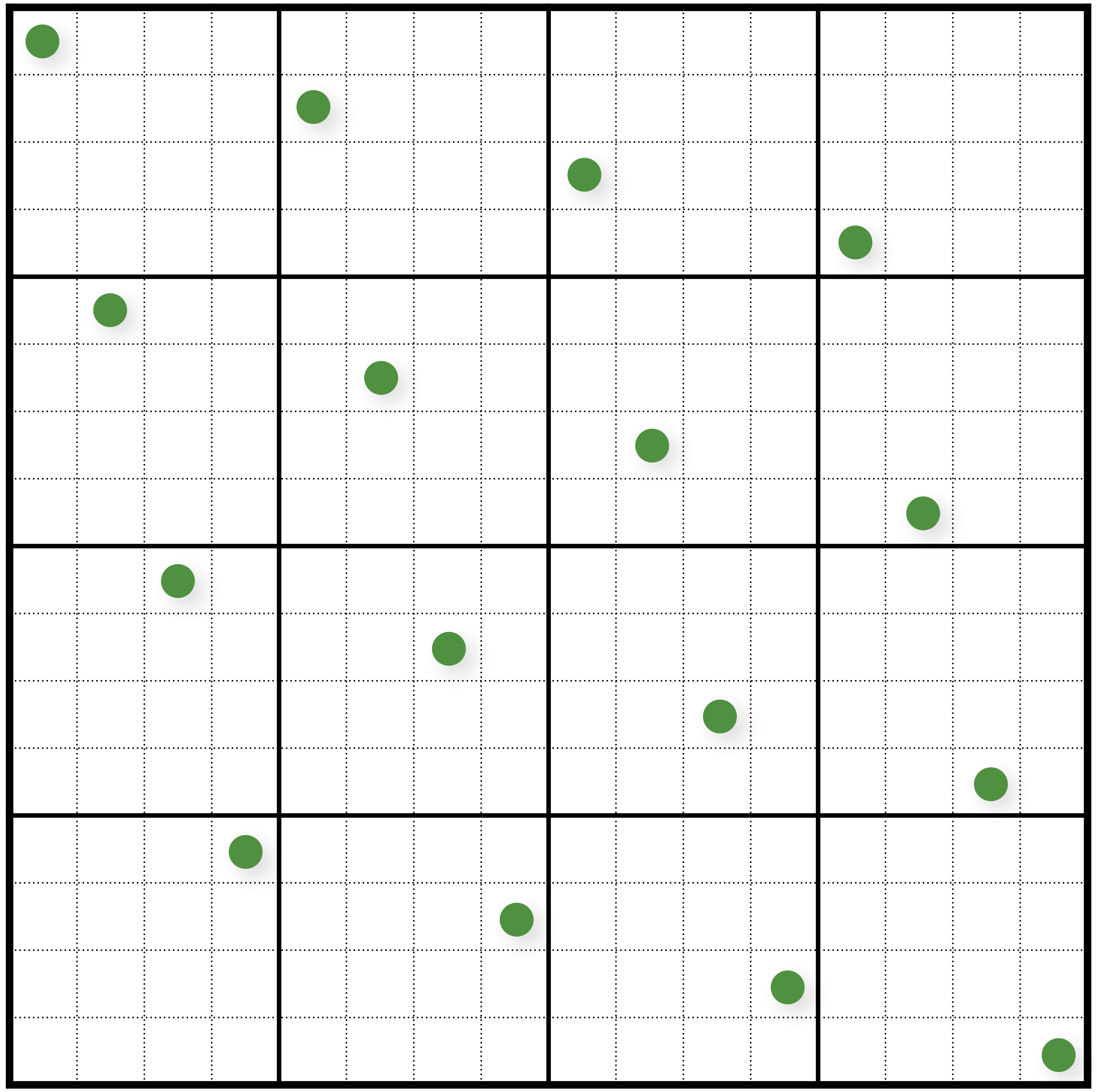
Multi-Jittered Sampling



Multi-Jittered Sampling

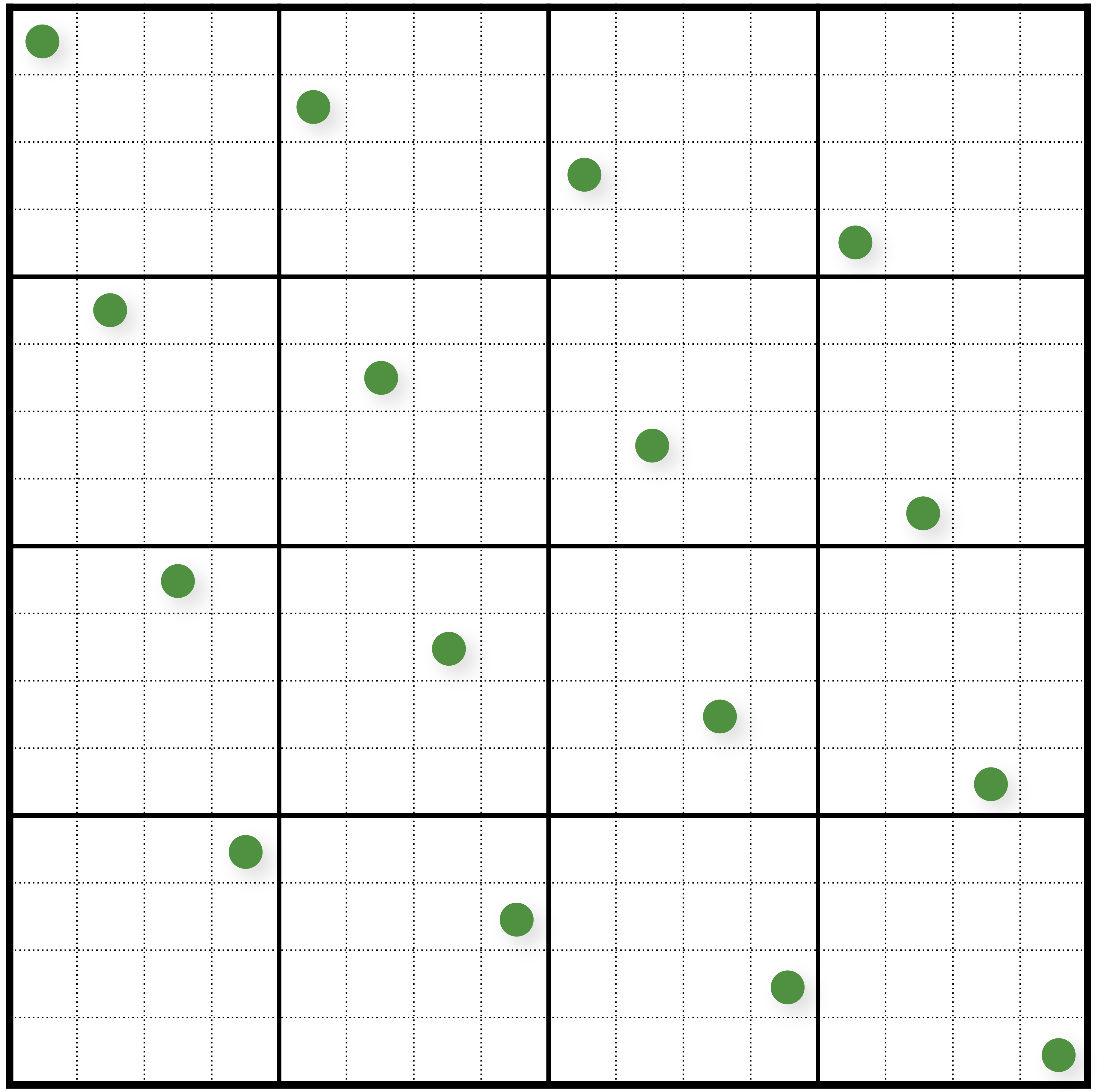
```
// initialize
float cellSize = 1.0 / (resX*resY);
for (uint i = 0; i < resX; i++)
	for (uint j = 0; j < resY; j++)
	{
		samples(i,j).x = i/resX + (j+randf()) / (resX*resY);
		samples(i,j).y = j/resY + (i+randf()) / (resX*resY);
	}

// 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);
```

Multi-Jittered Sampling



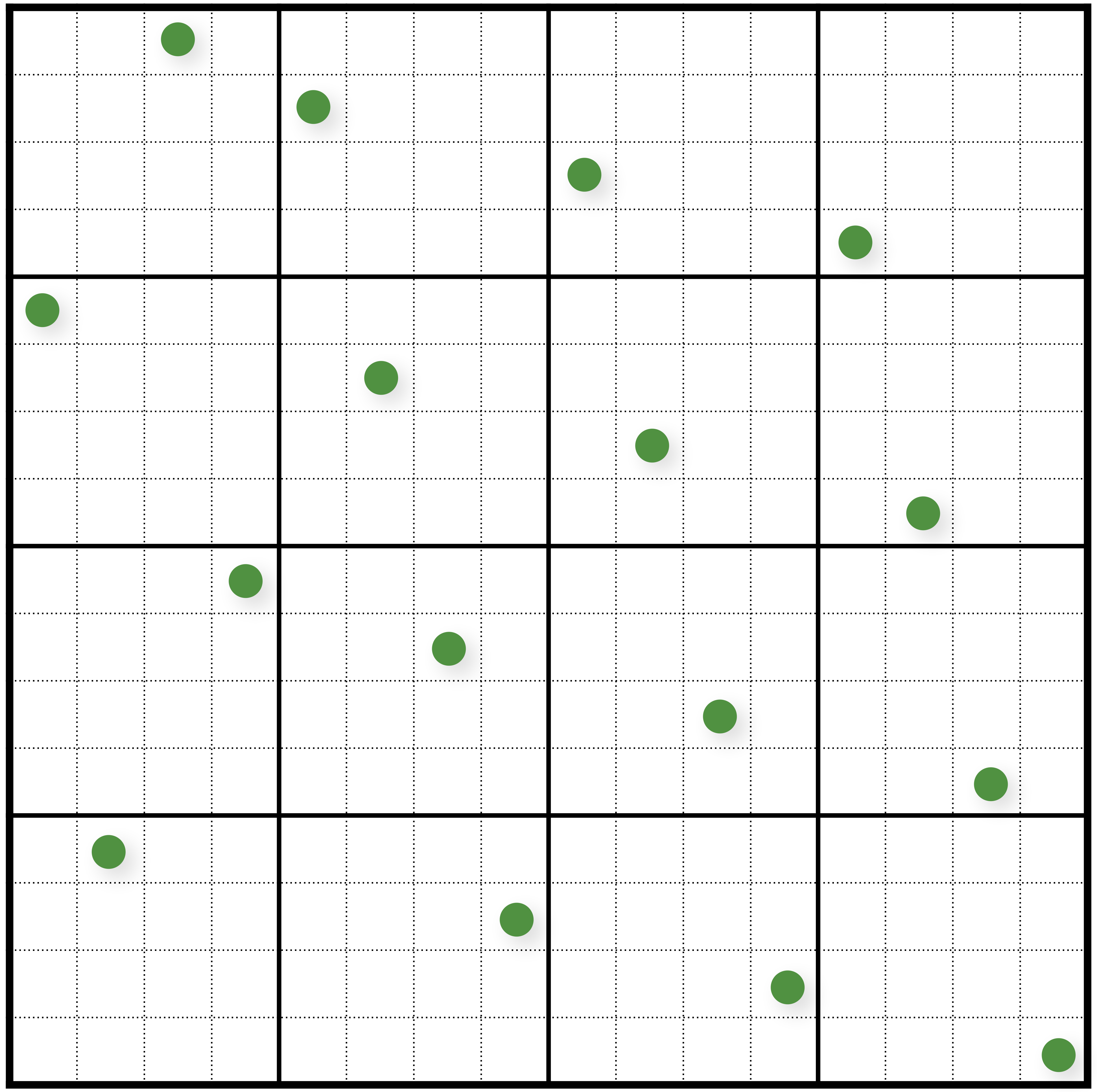
Initialize

Multi-Jittered Sampling



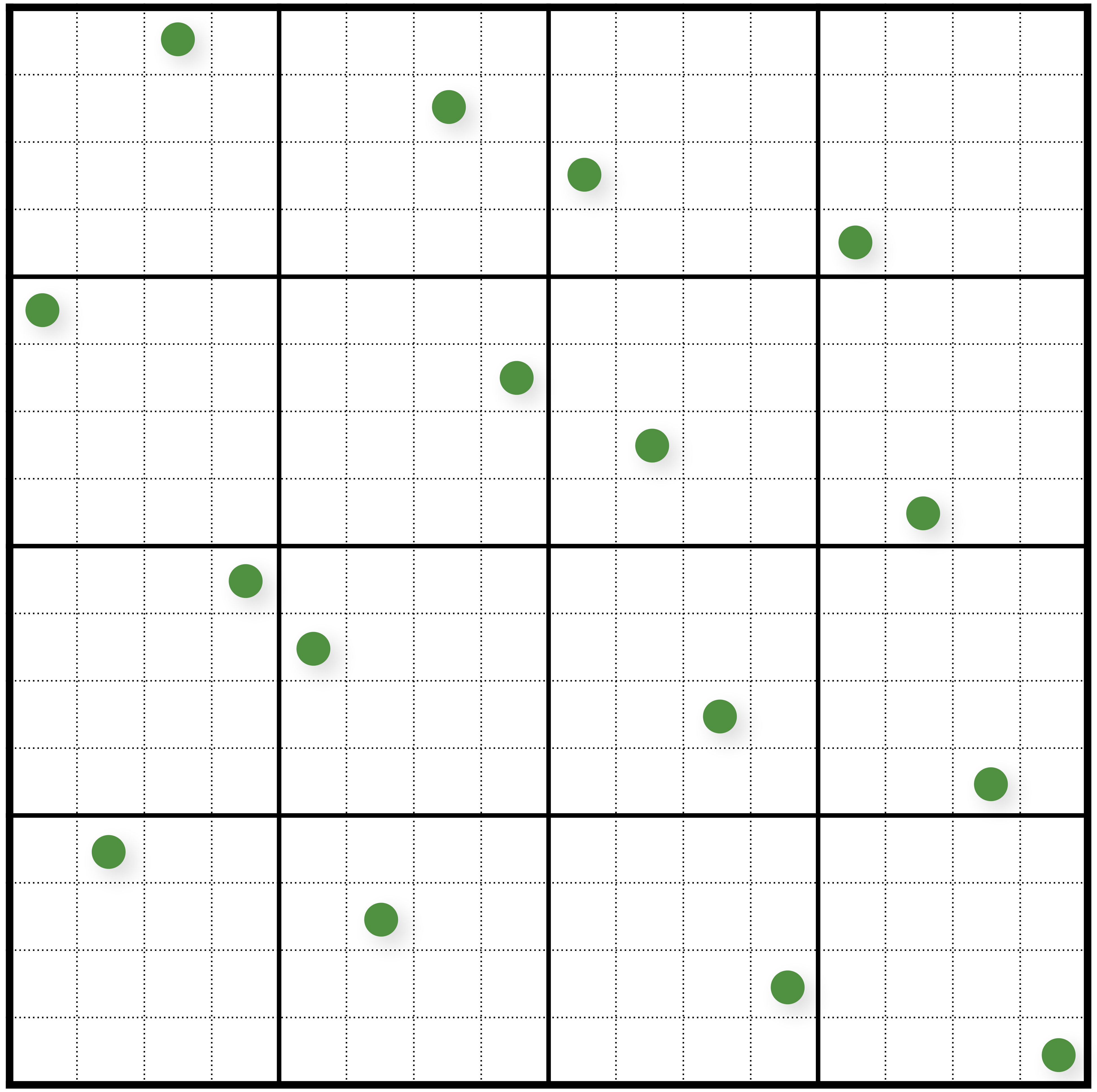
Shuffle x-coords

Multi-Jittered Sampling



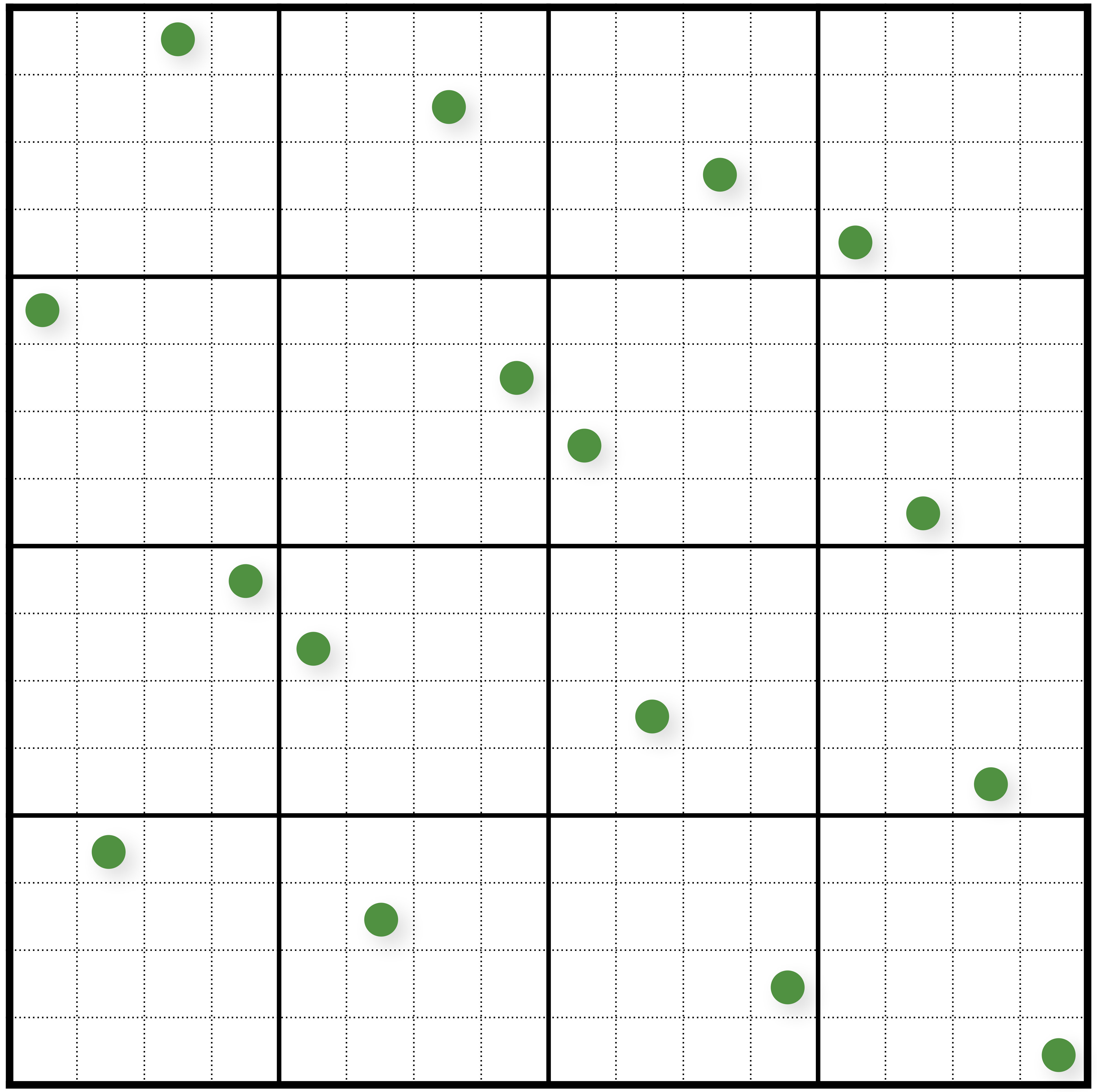
Shuffle x-coords

Multi-Jittered Sampling



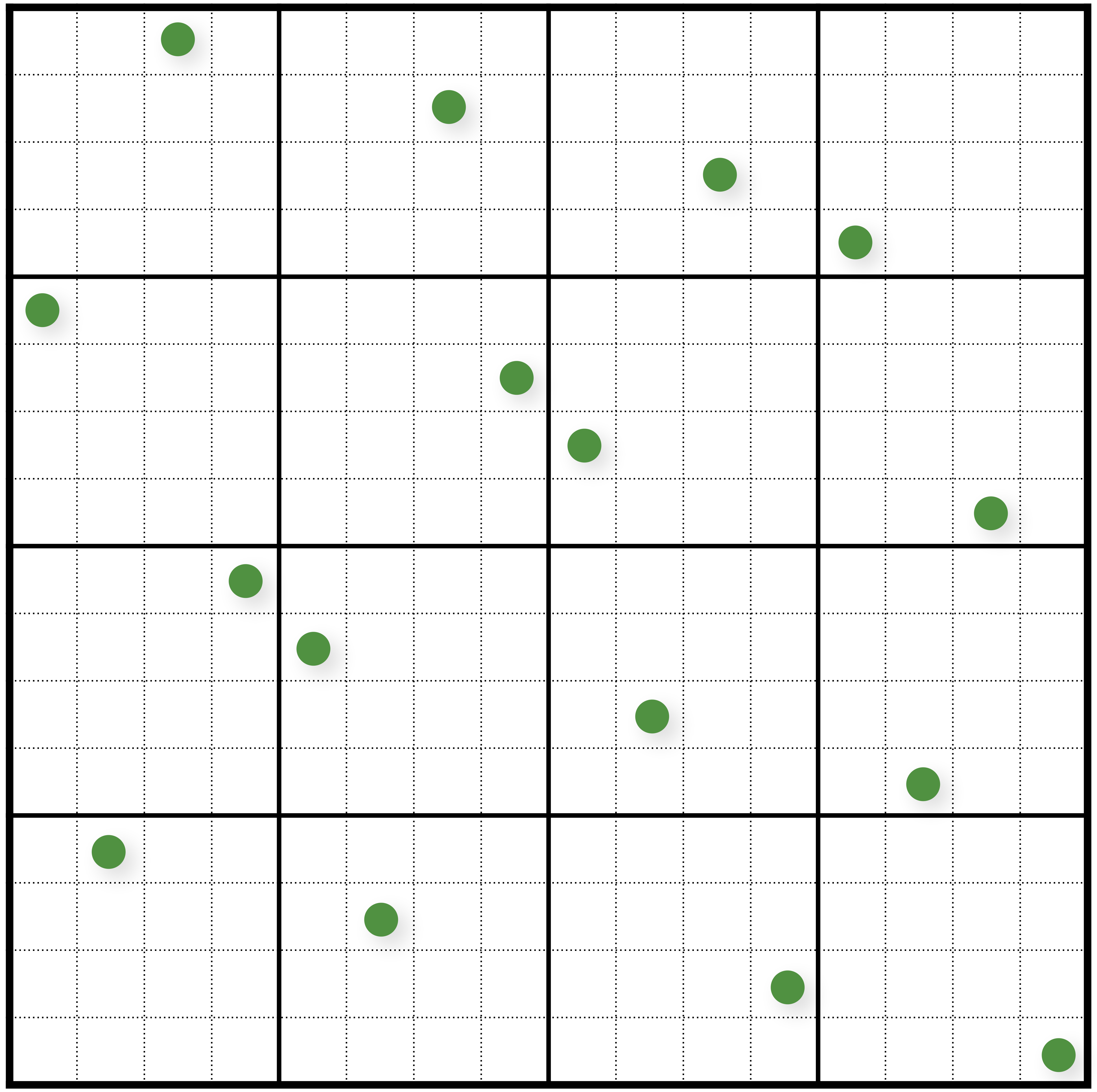
Shuffle x-coords

Multi-Jittered Sampling



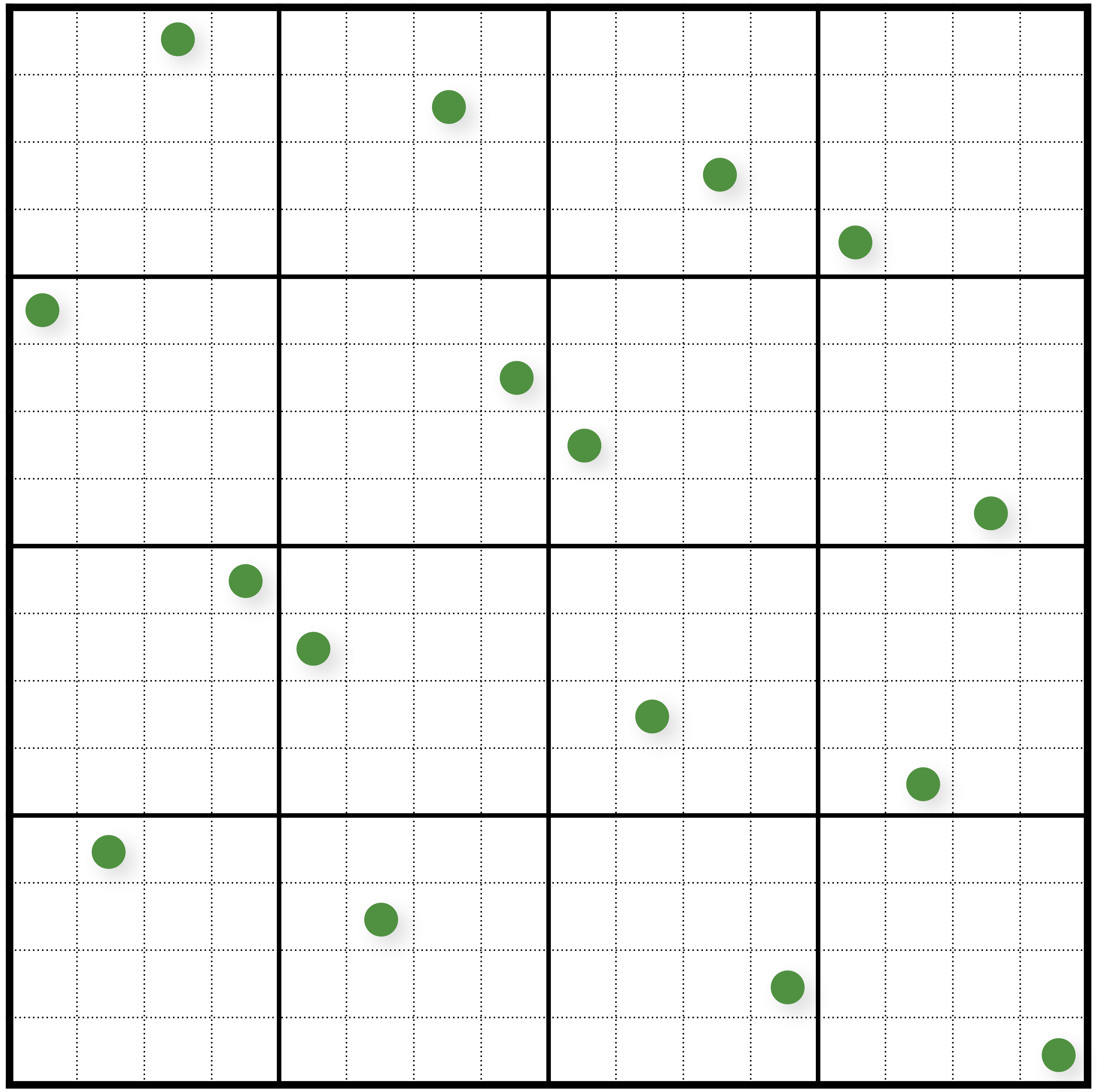
Shuffle x-coords

Multi-Jittered Sampling

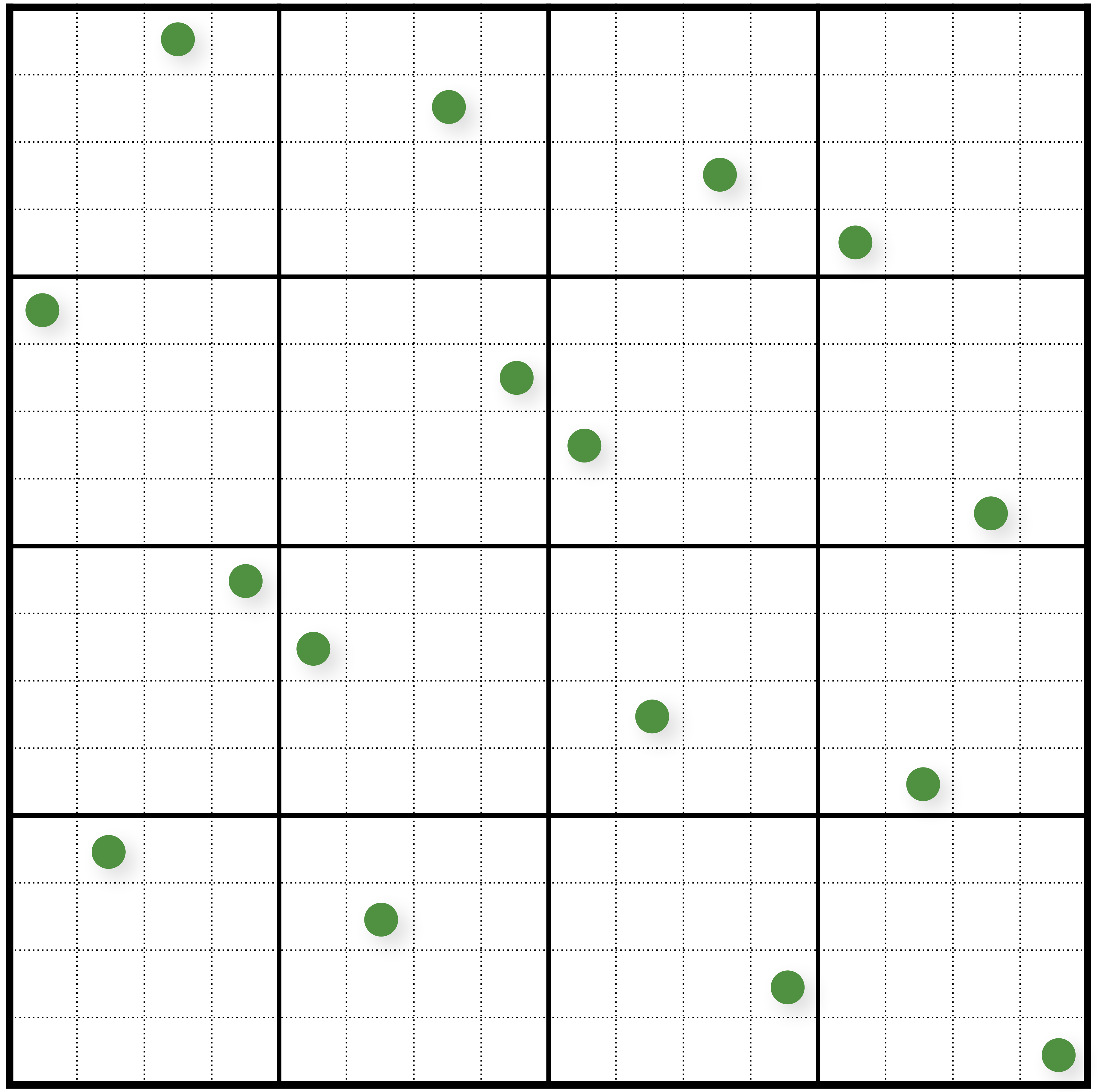


Shuffle x-coords

Multi-Jittered Sampling

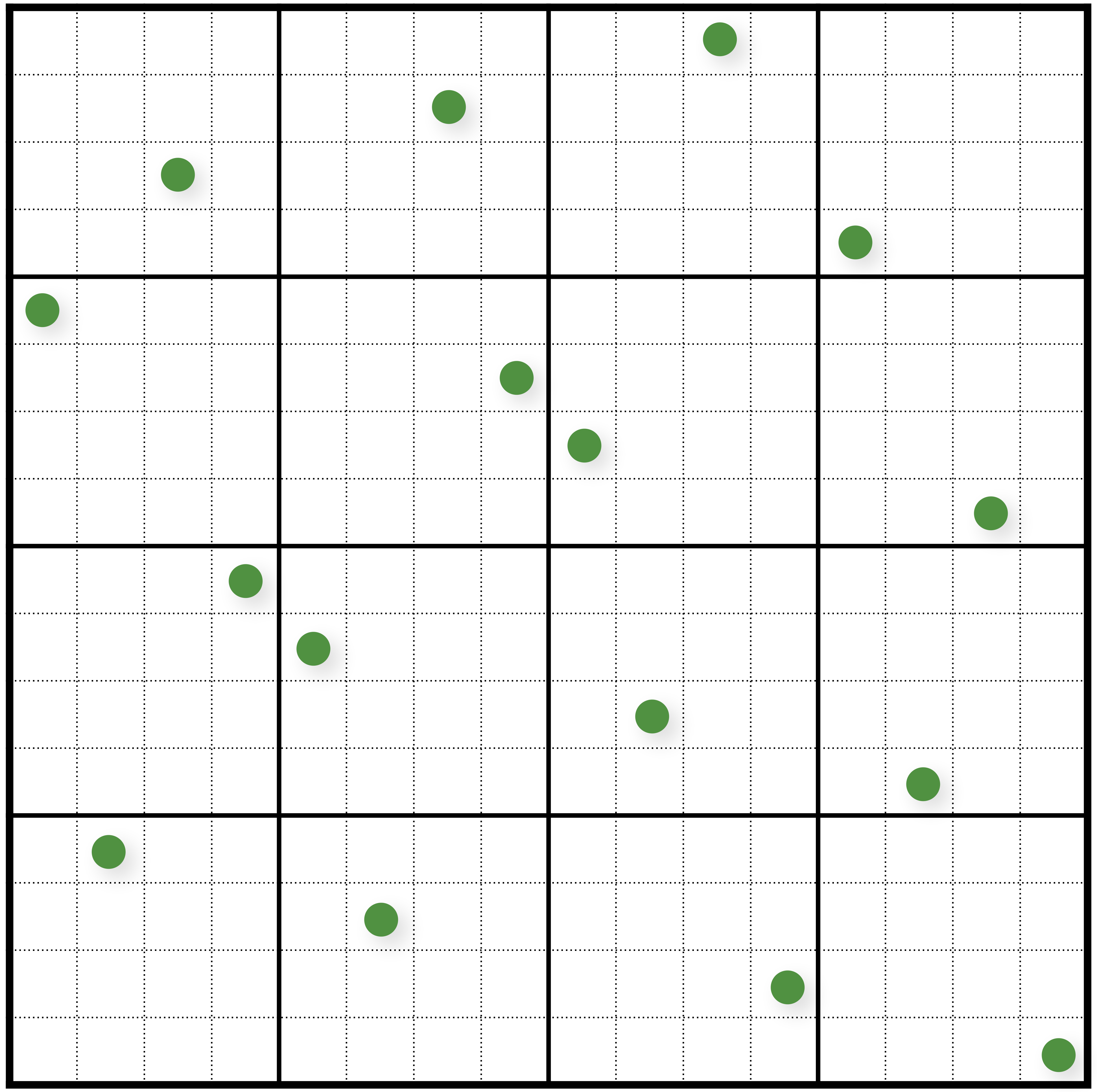


Multi-Jittered Sampling



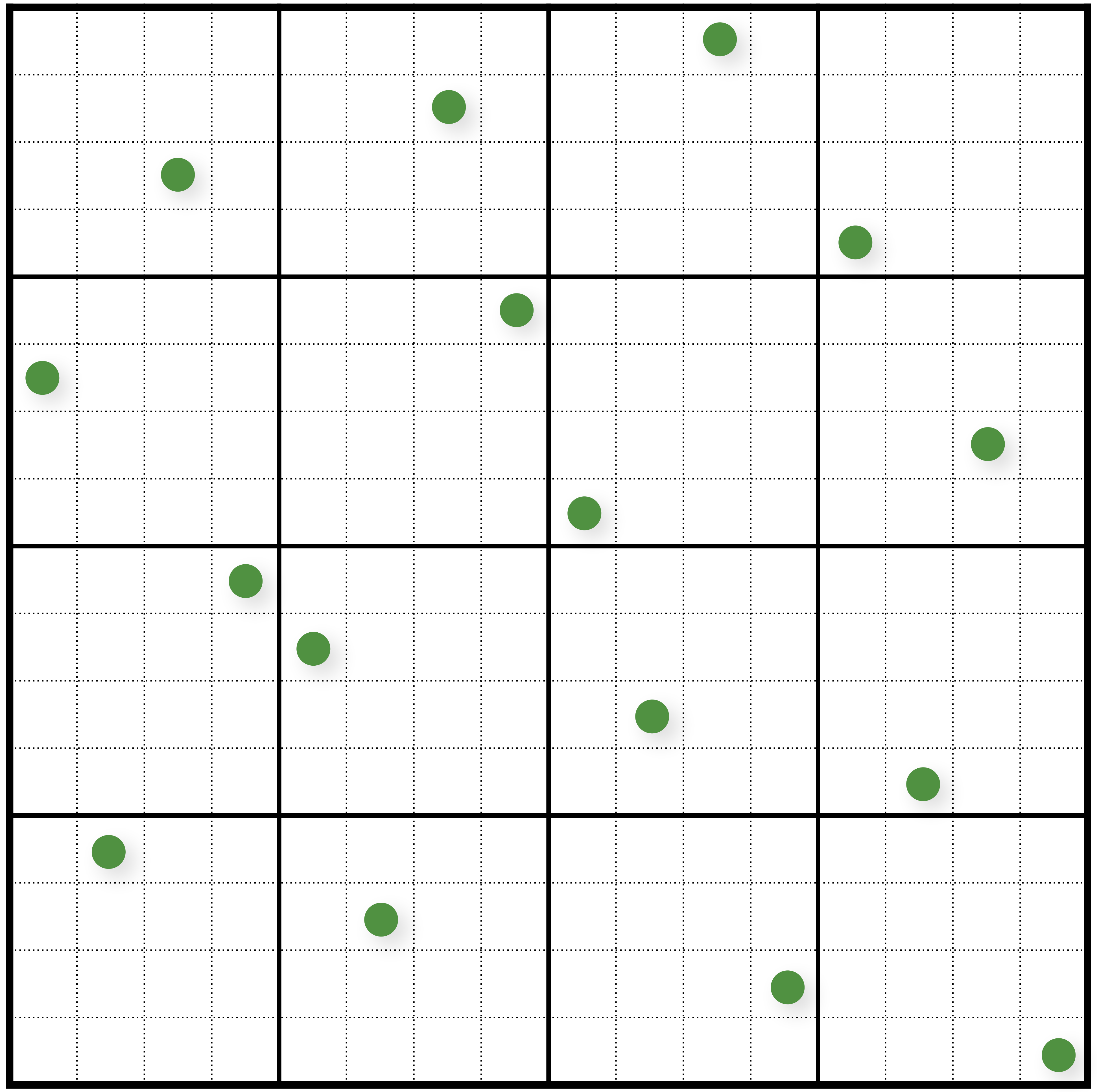
Shuffle y-coords

Multi-Jittered Sampling



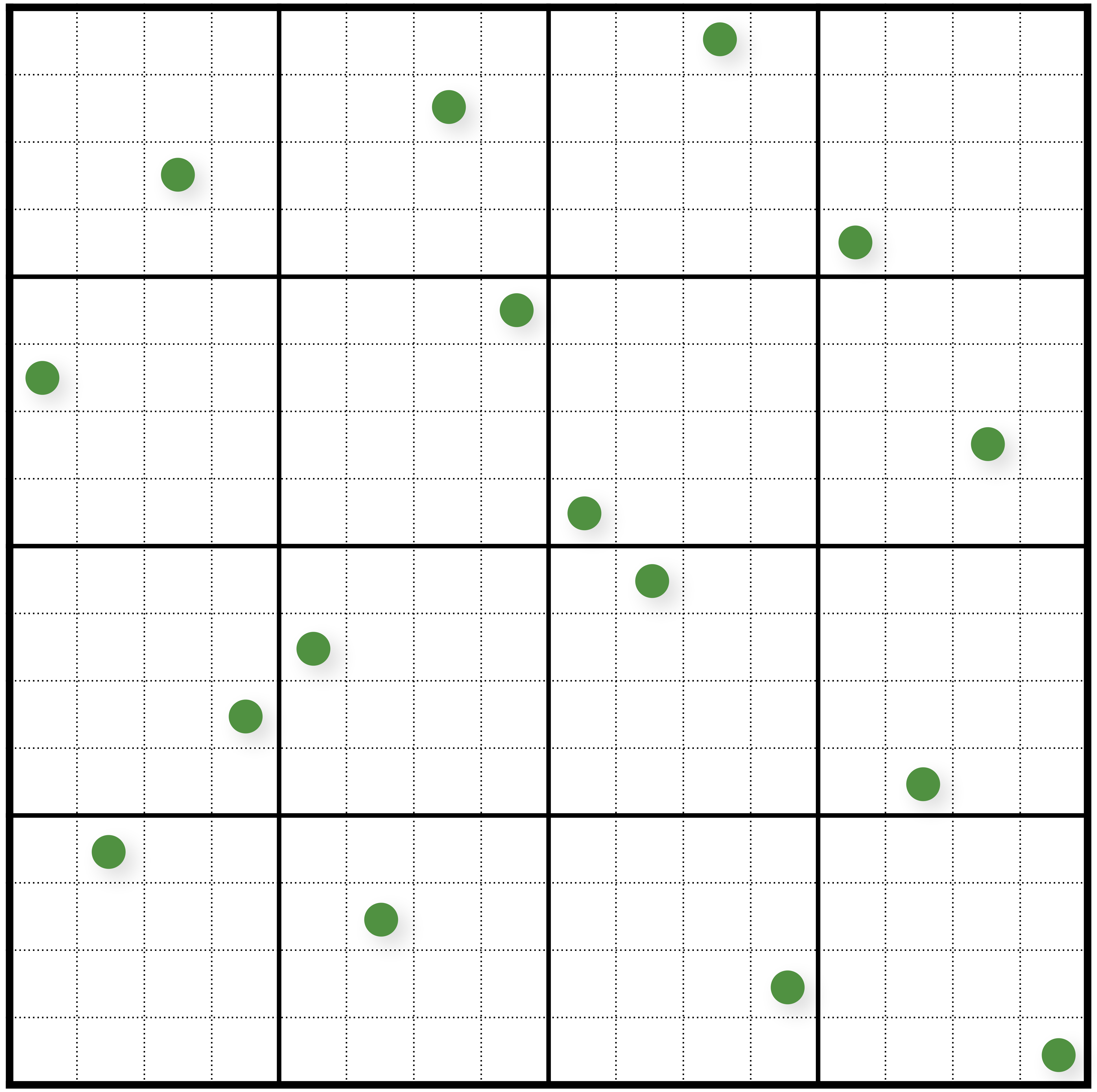
Shuffle y-coords

Multi-Jittered Sampling



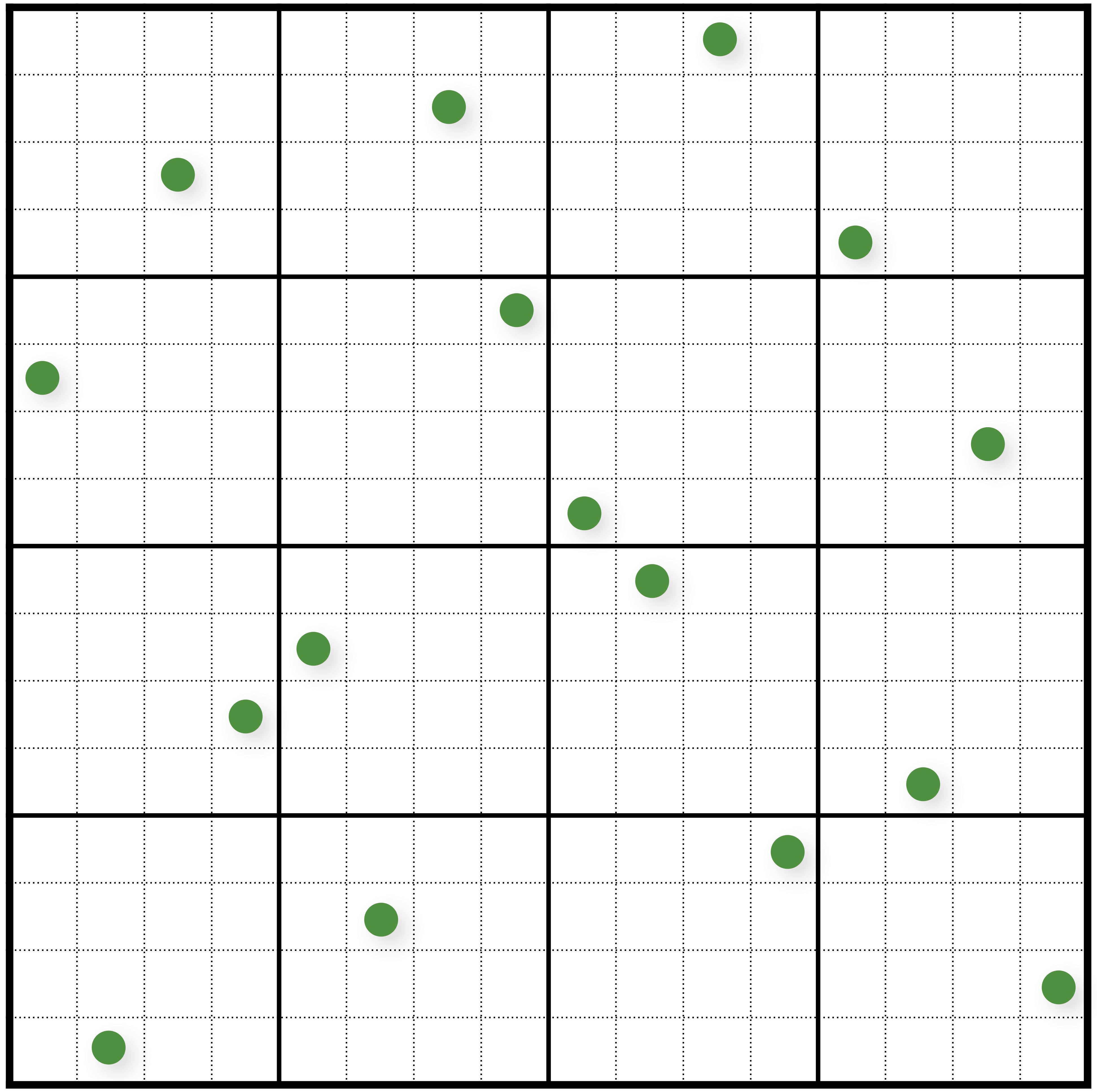
Shuffle y-coords

Multi-Jittered Sampling



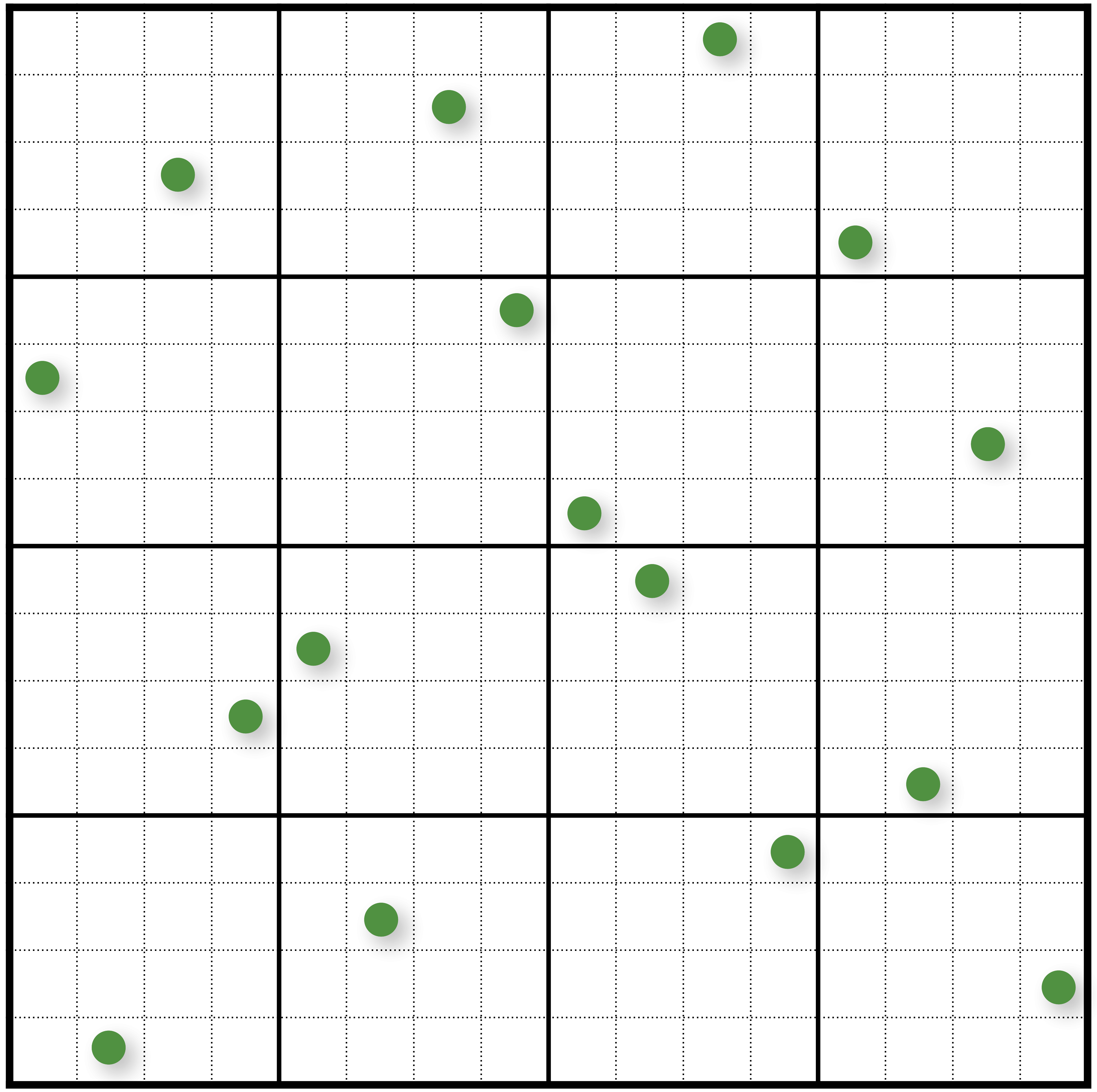
Shuffle y-coords

Multi-Jittered Sampling

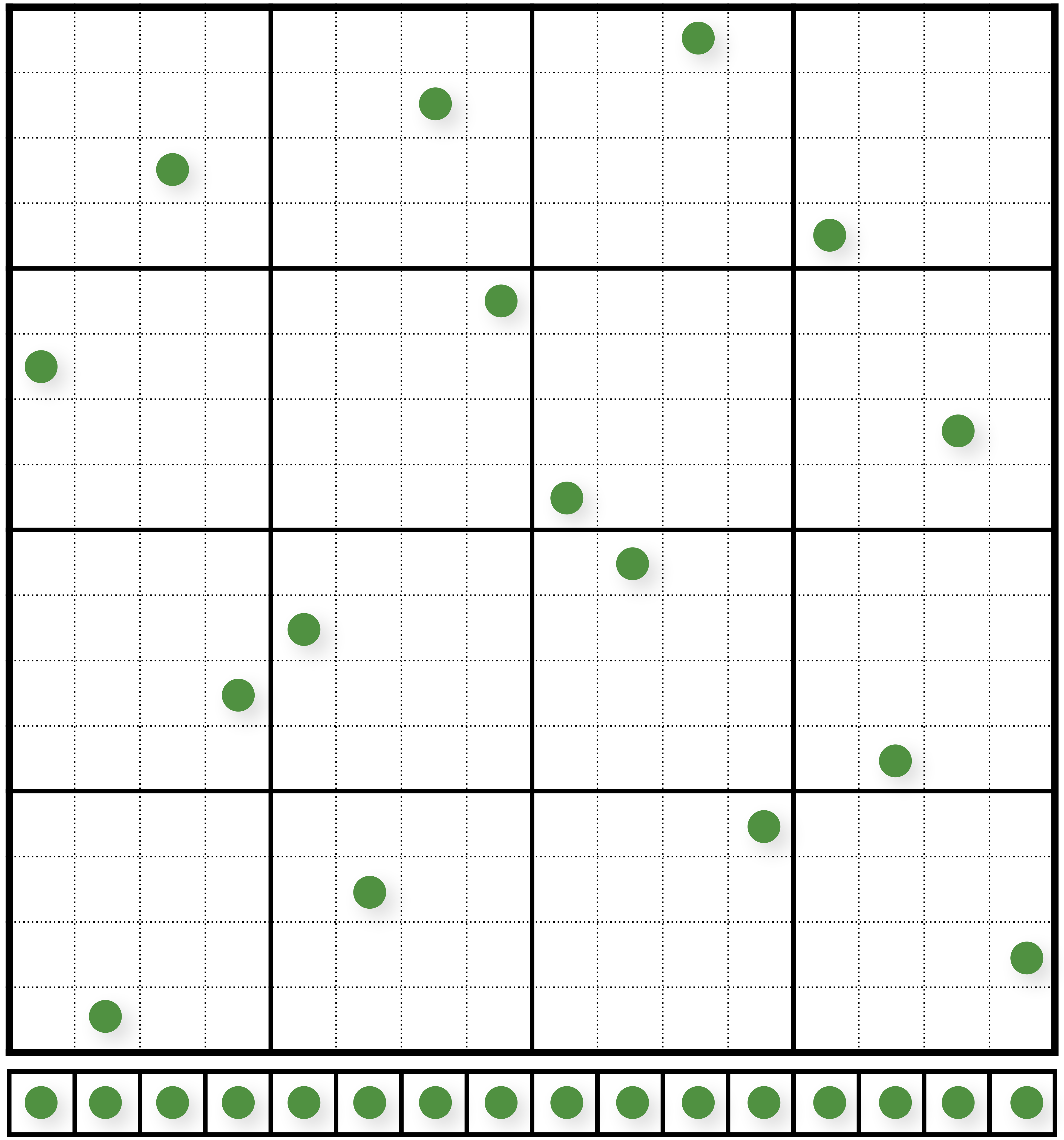


Shuffle y-coords

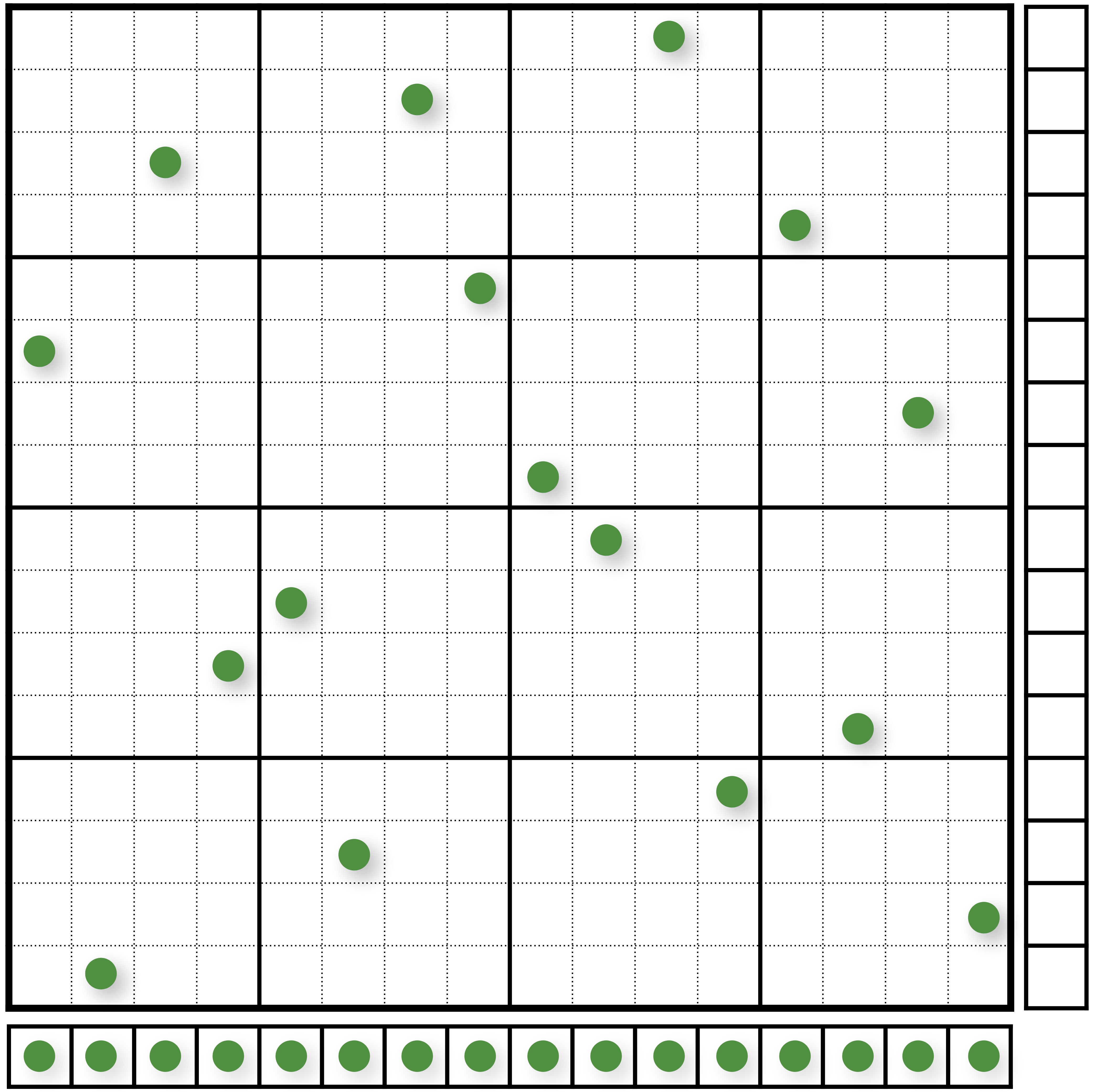
Multi-Jittered Sampling (Projections)



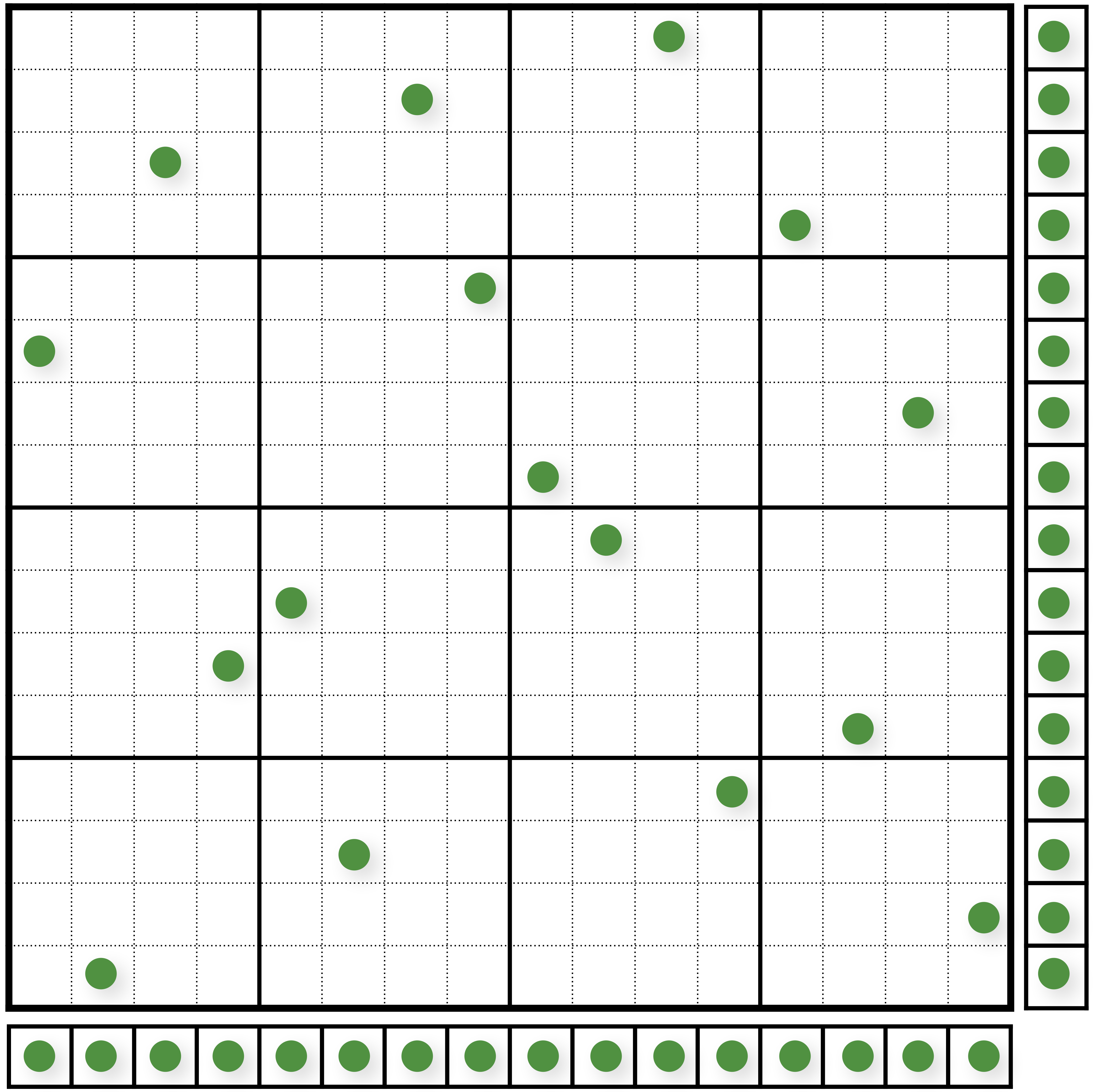
Multi-Jittered Sampling (Projections)



Multi-Jittered Sampling (Projections)

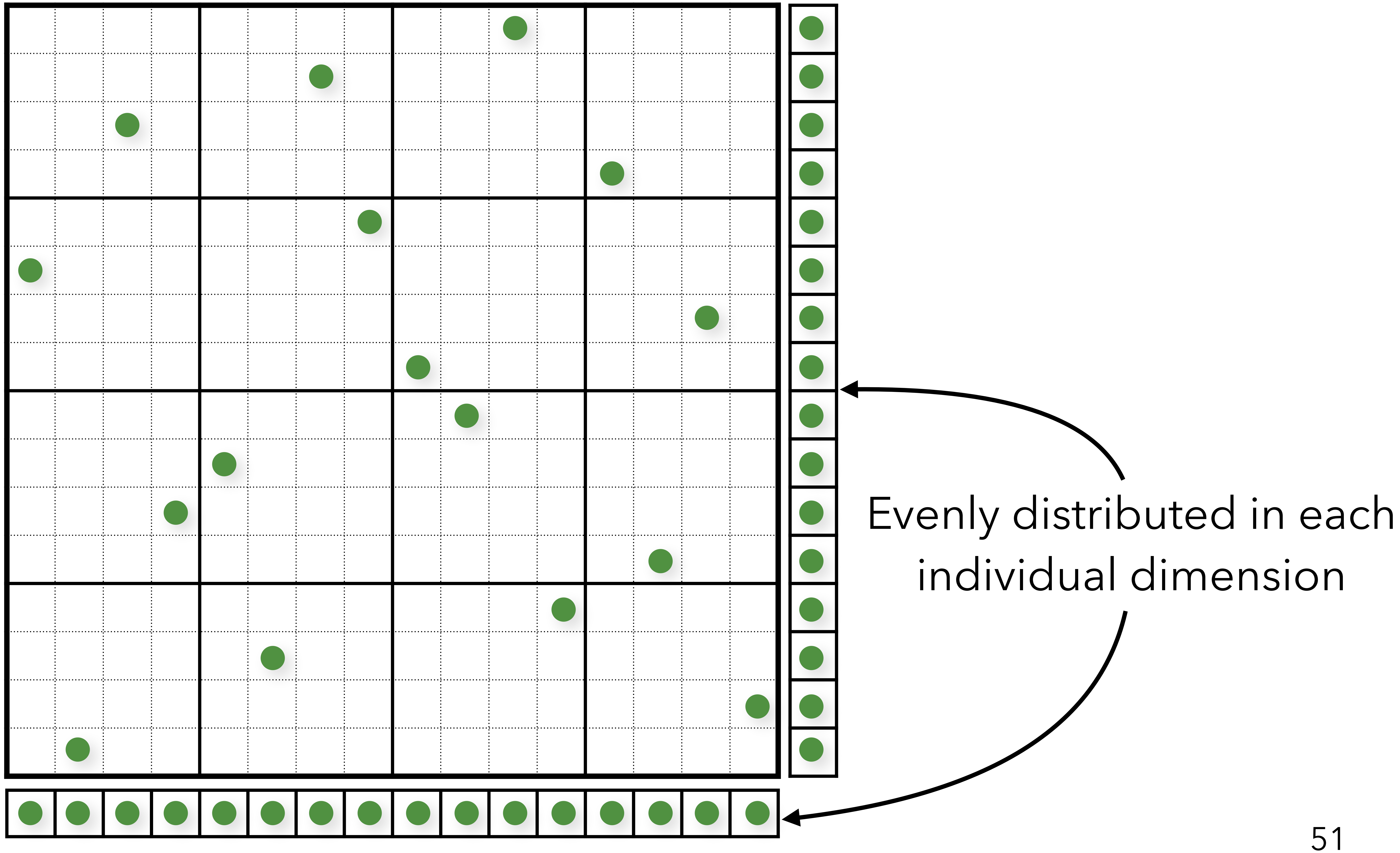


Multi-Jittered Sampling (Projections)

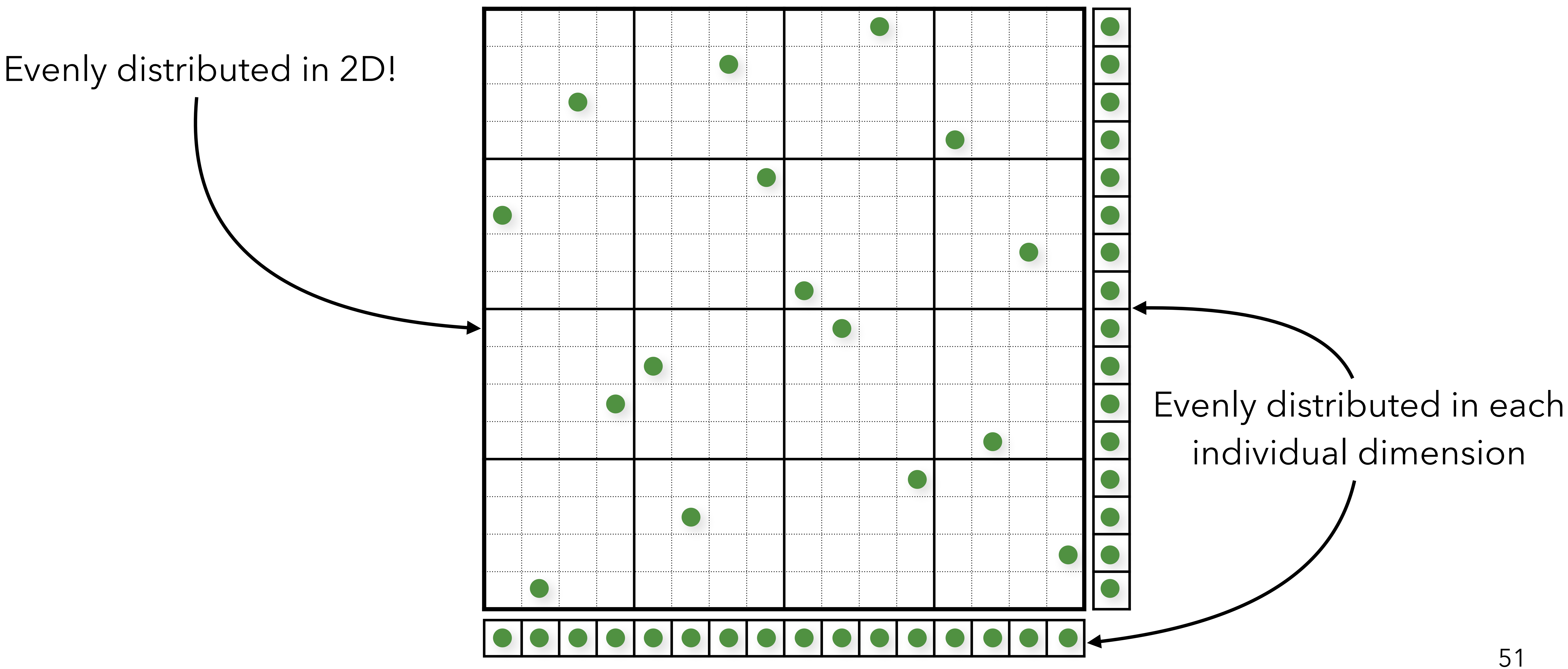


UNIVERSITÄT DES SAARLANDES

Multi-Jittered Sampling (Projections)

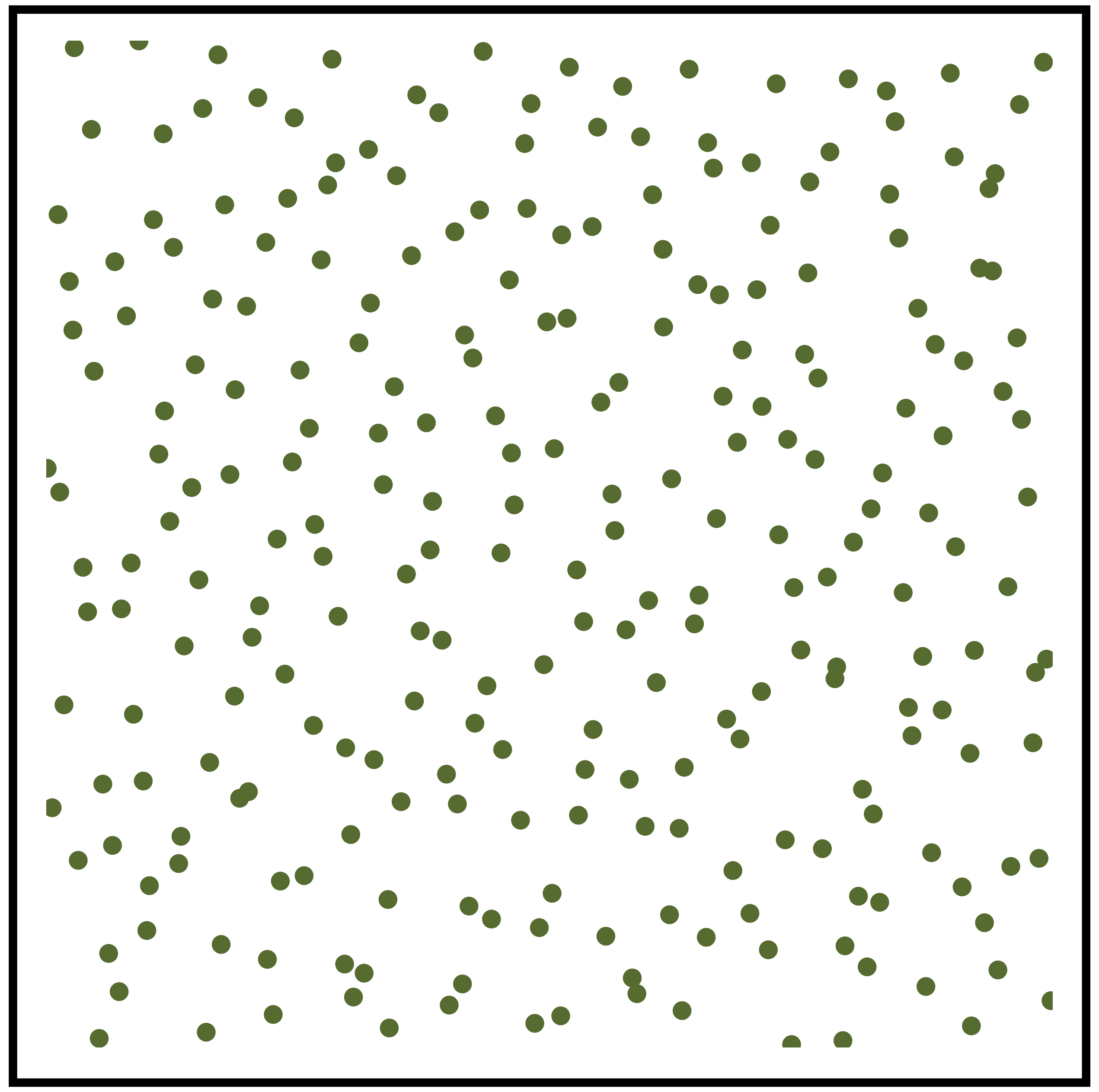


Multi-Jittered Sampling (Projections)

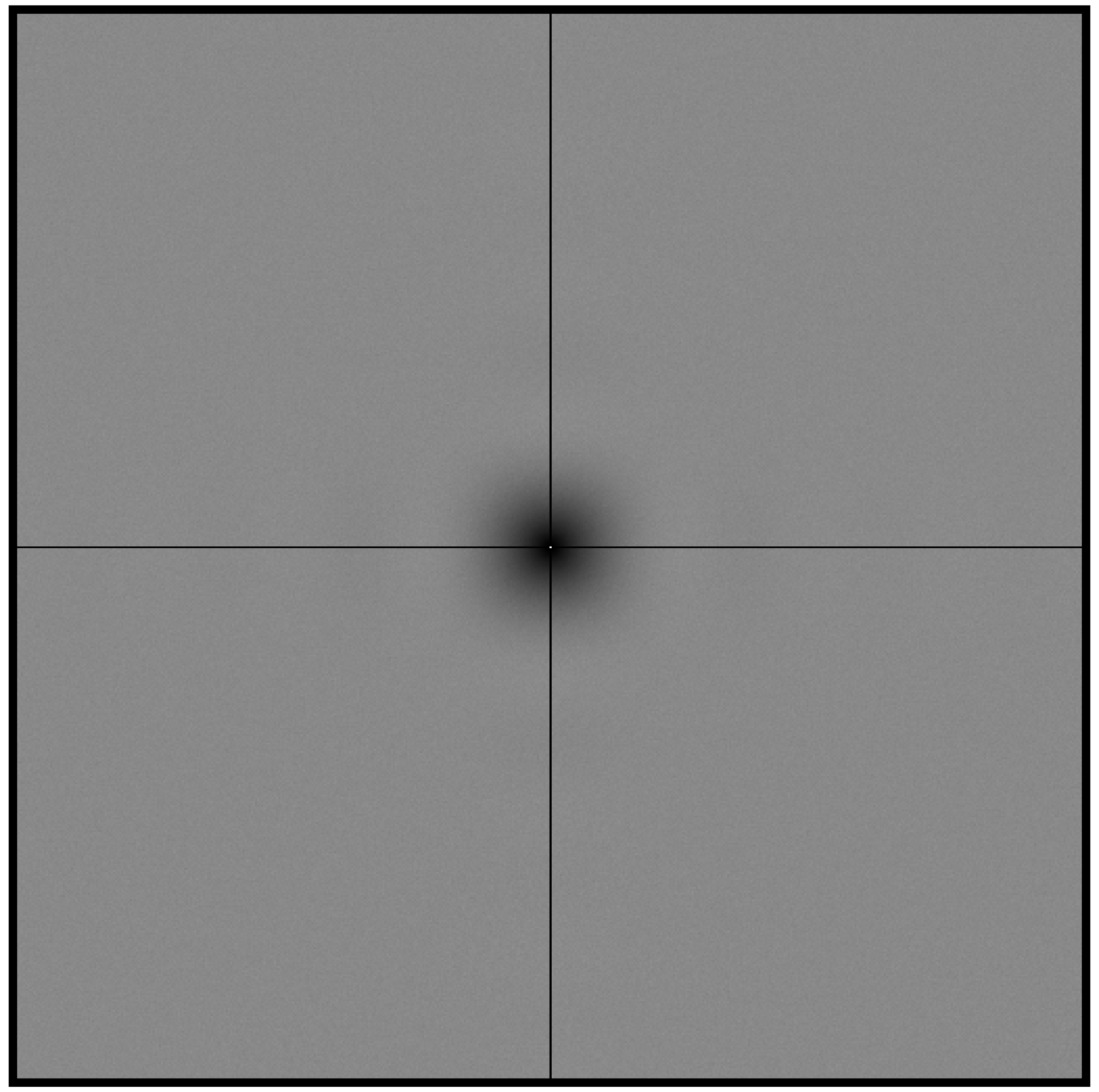


Multi-Jittered Sampling

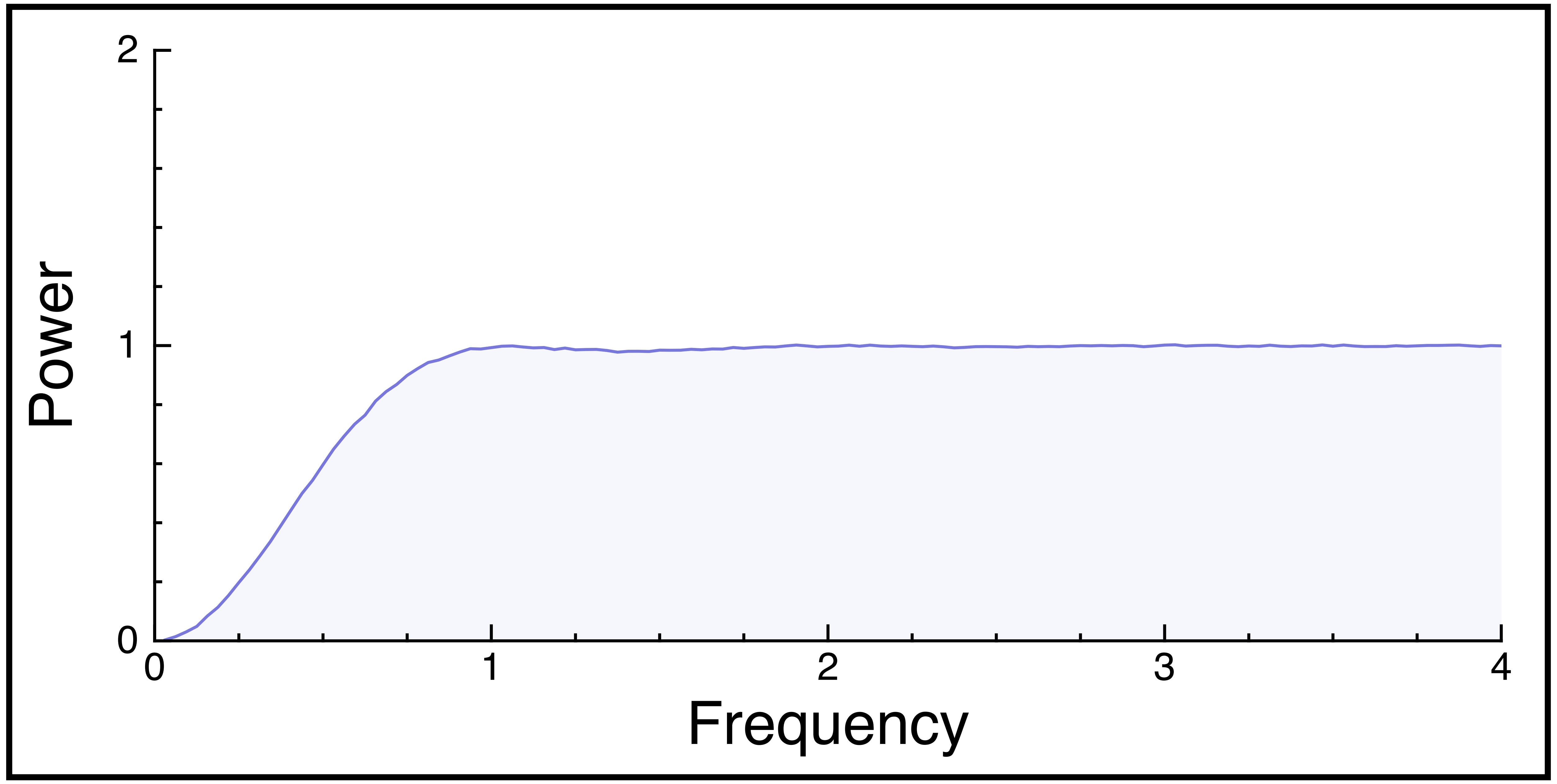
Samples



Expected power spectrum

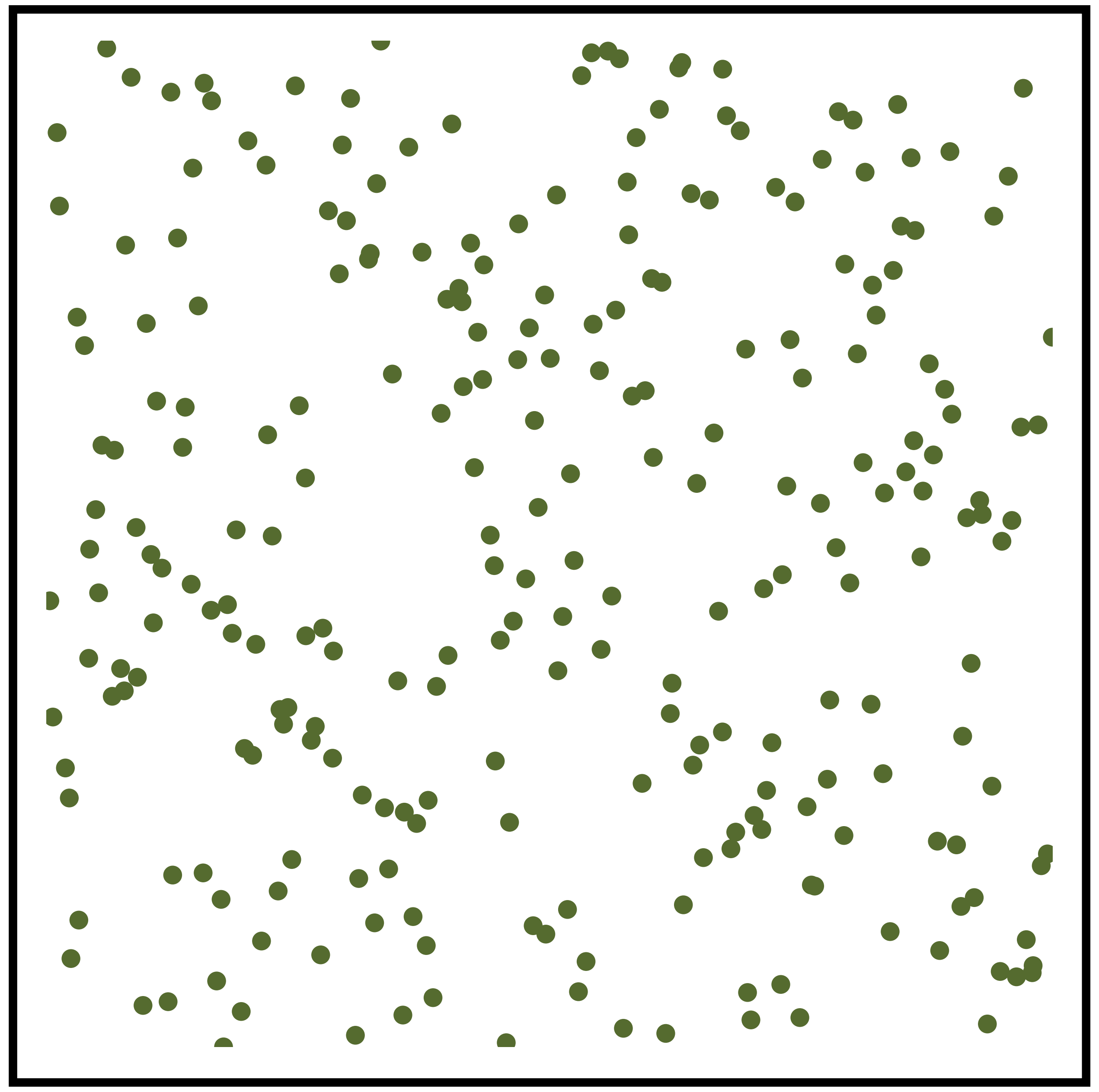


Radial mean

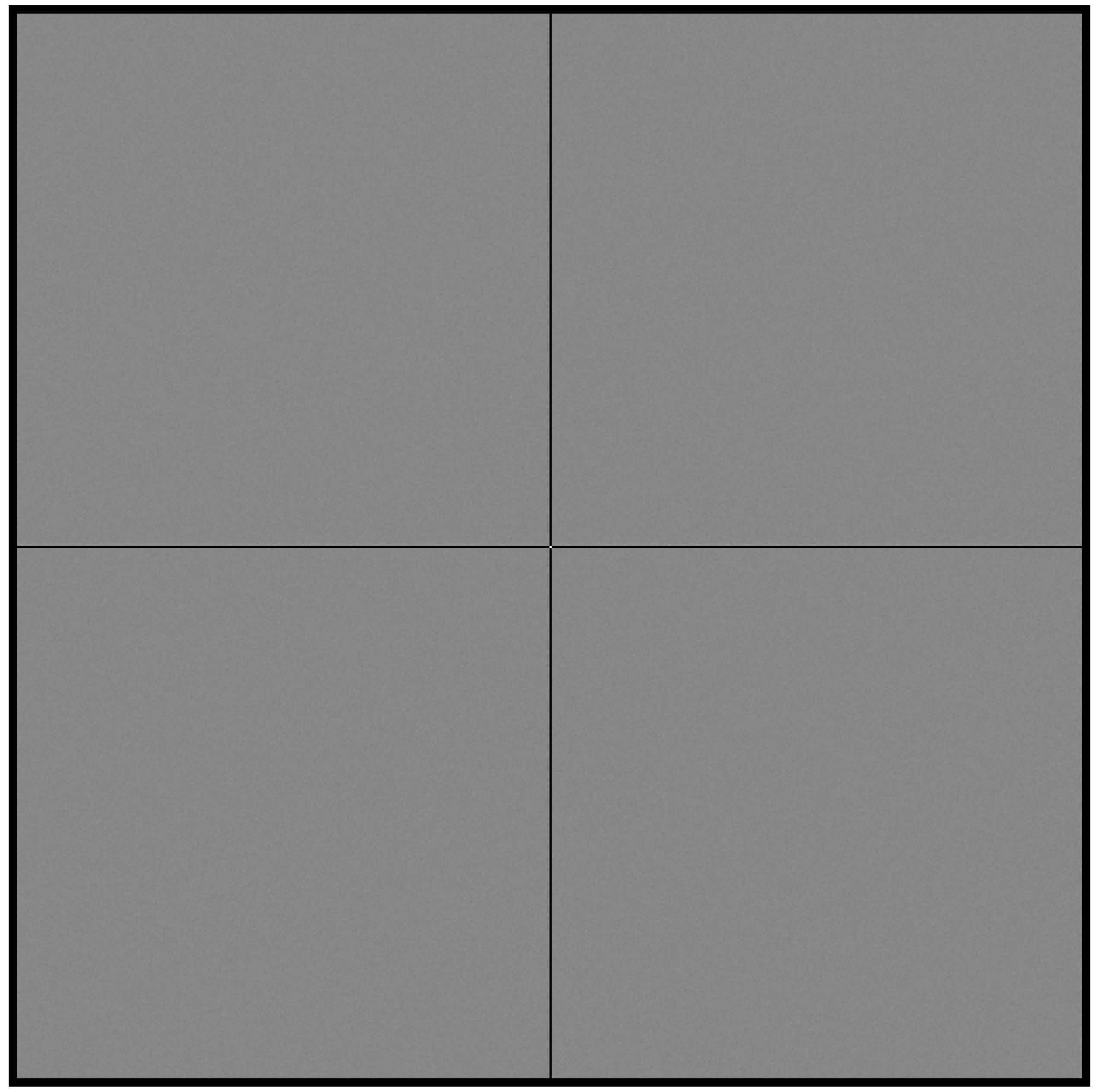


N-Rooks Sampling

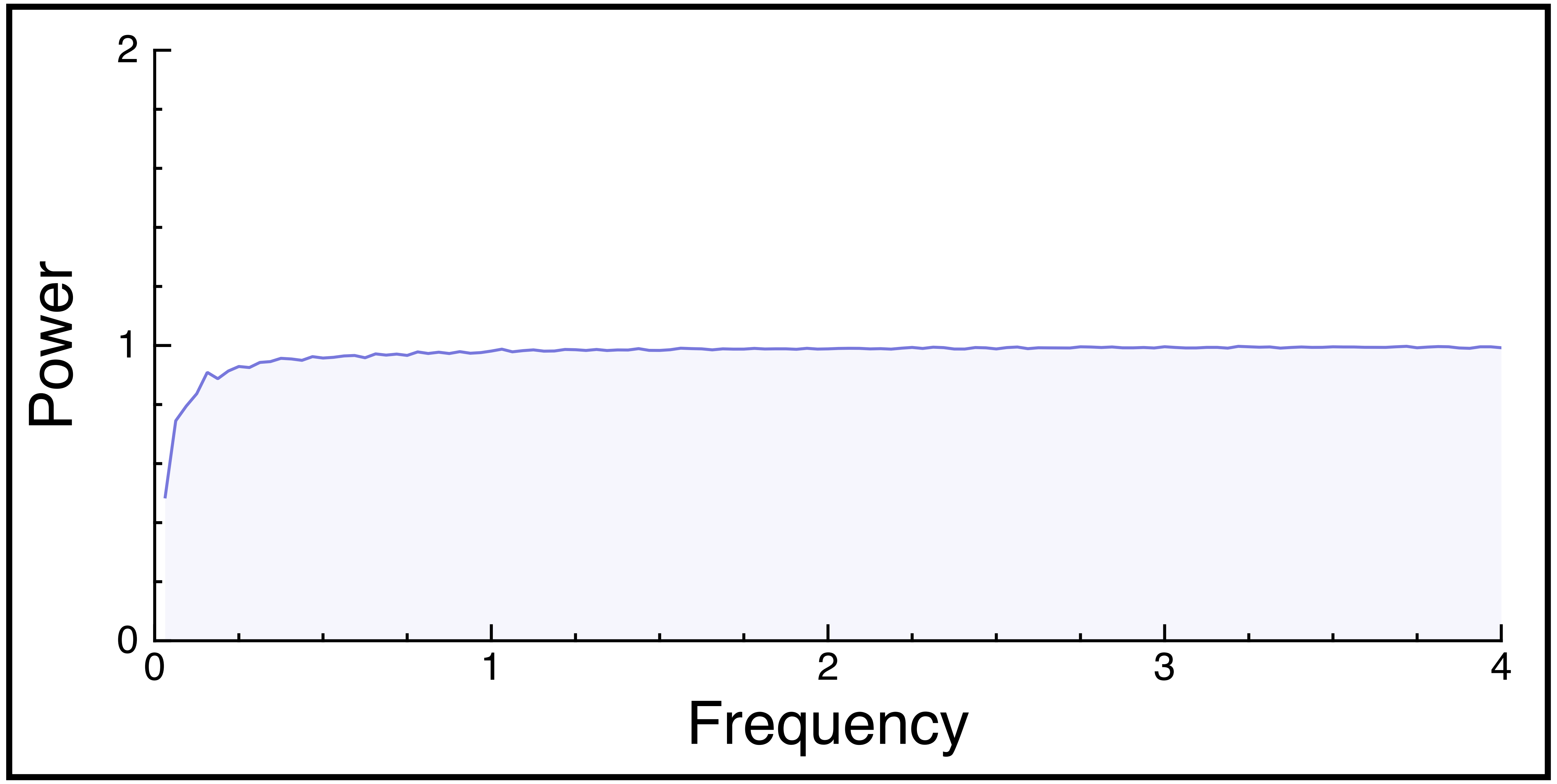
Samples



Expected power spectrum

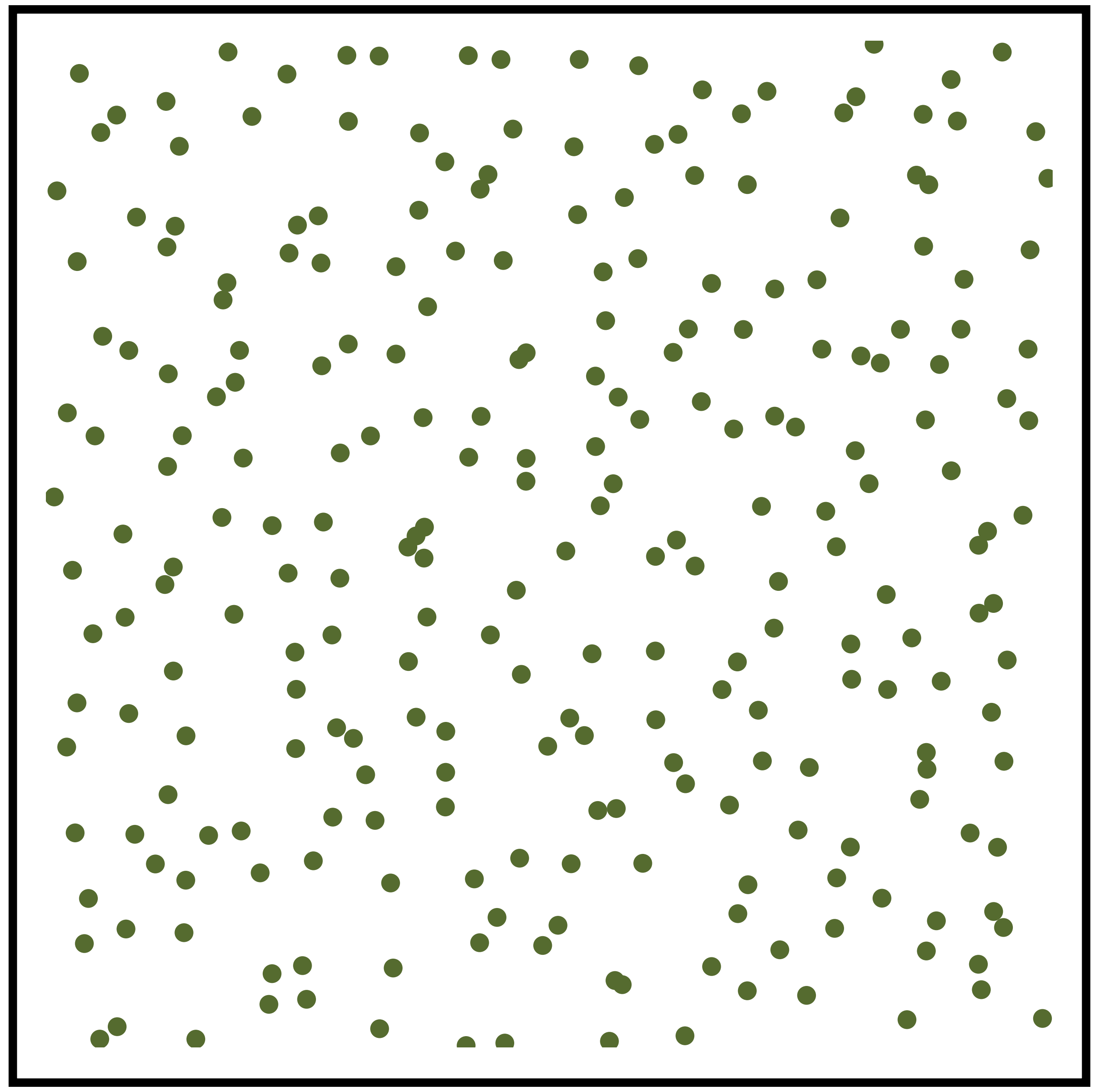


Radial mean

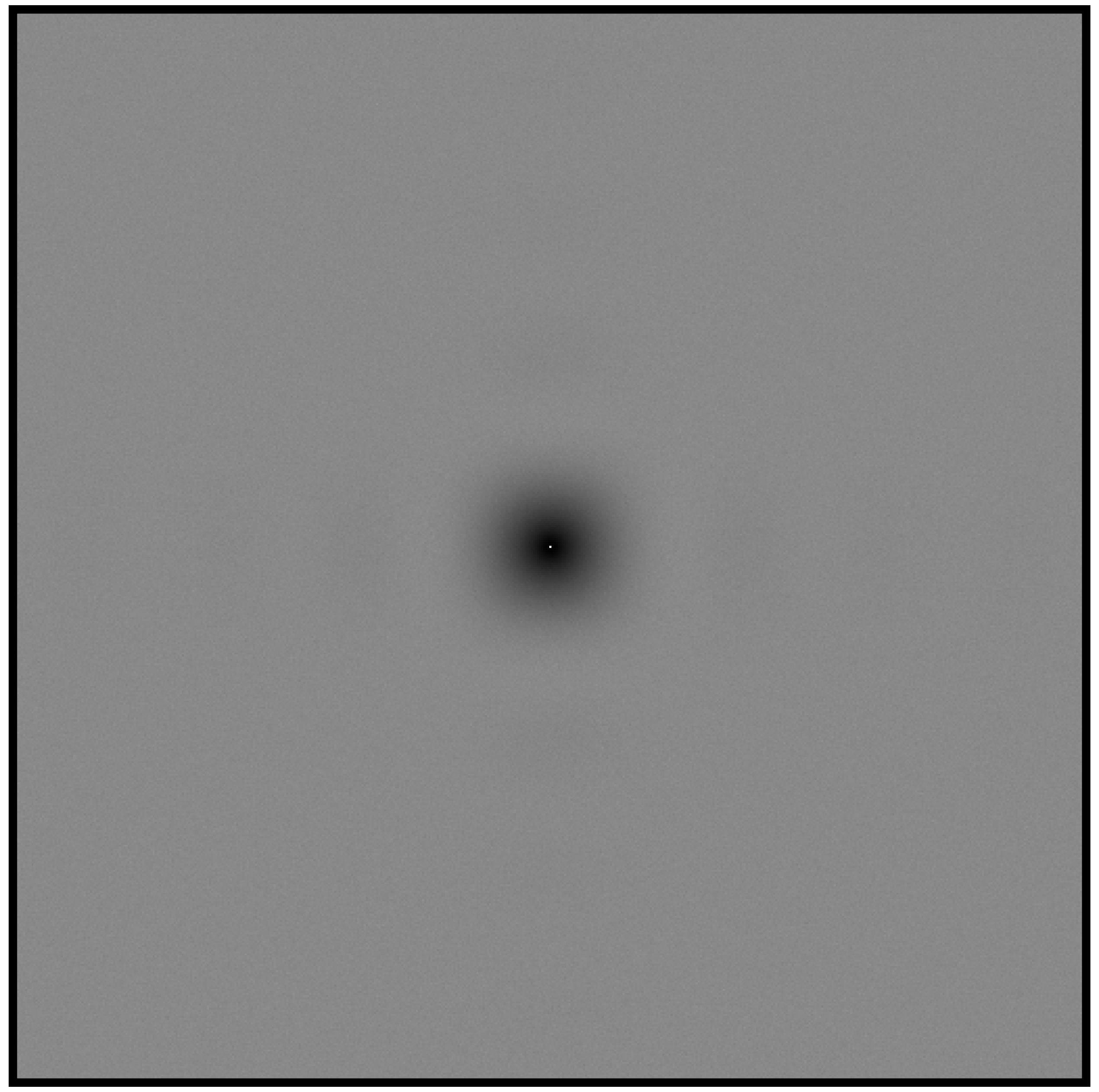


Jittered Sampling

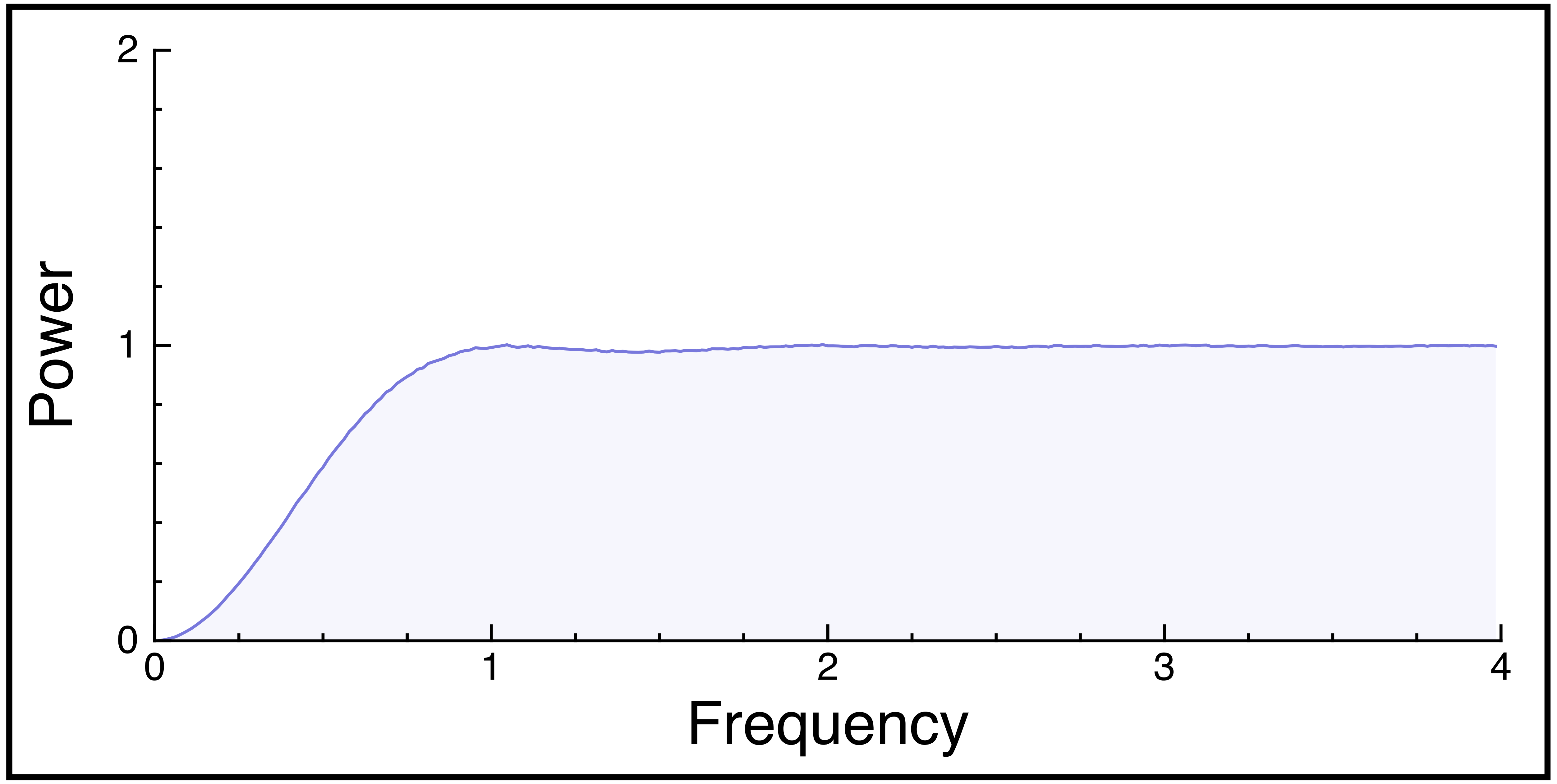
Samples



Expected power spectrum



Radial mean



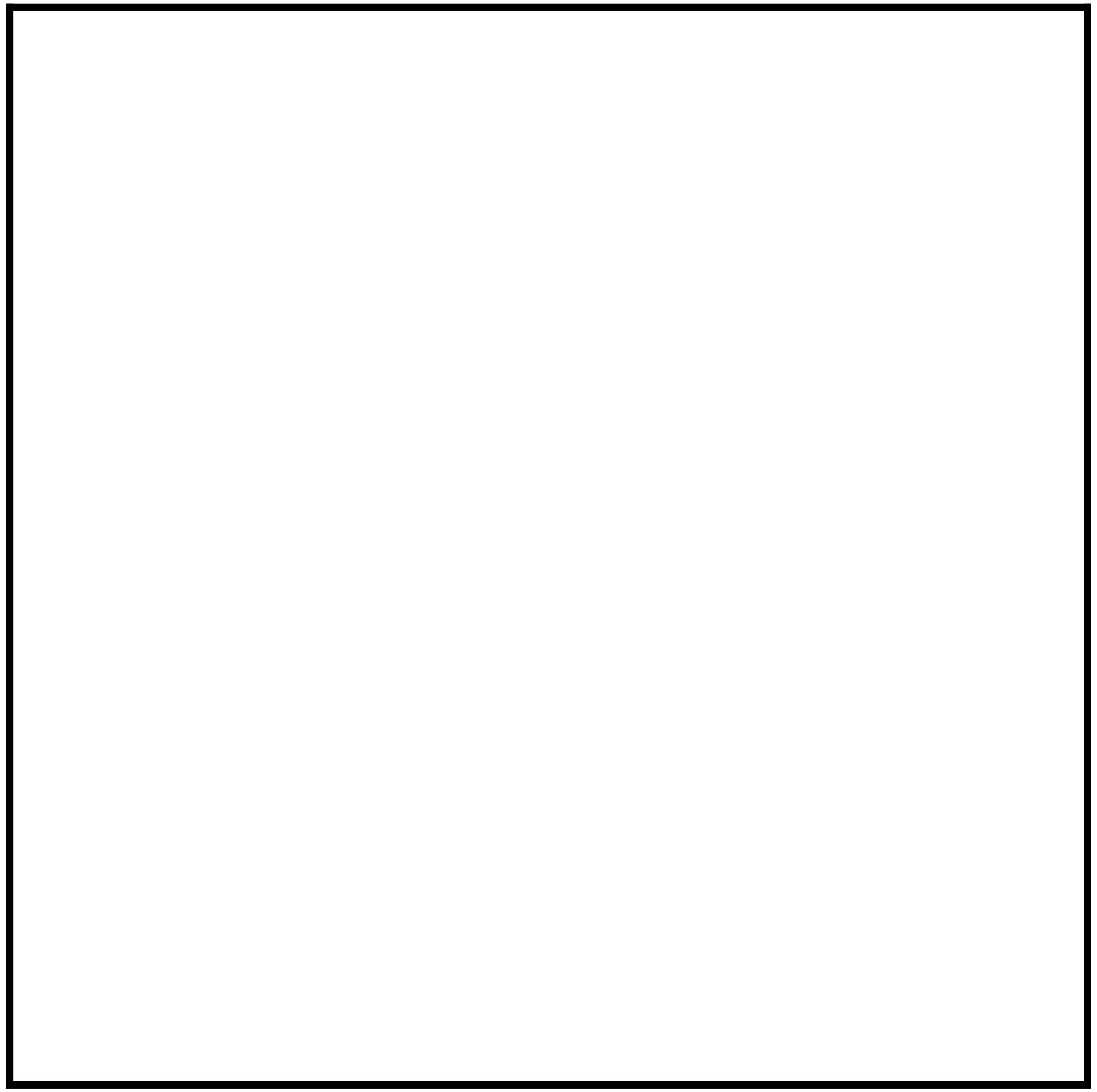
Poisson-Disk/Blue-Noise Sampling

Enforce a minimum distance between points

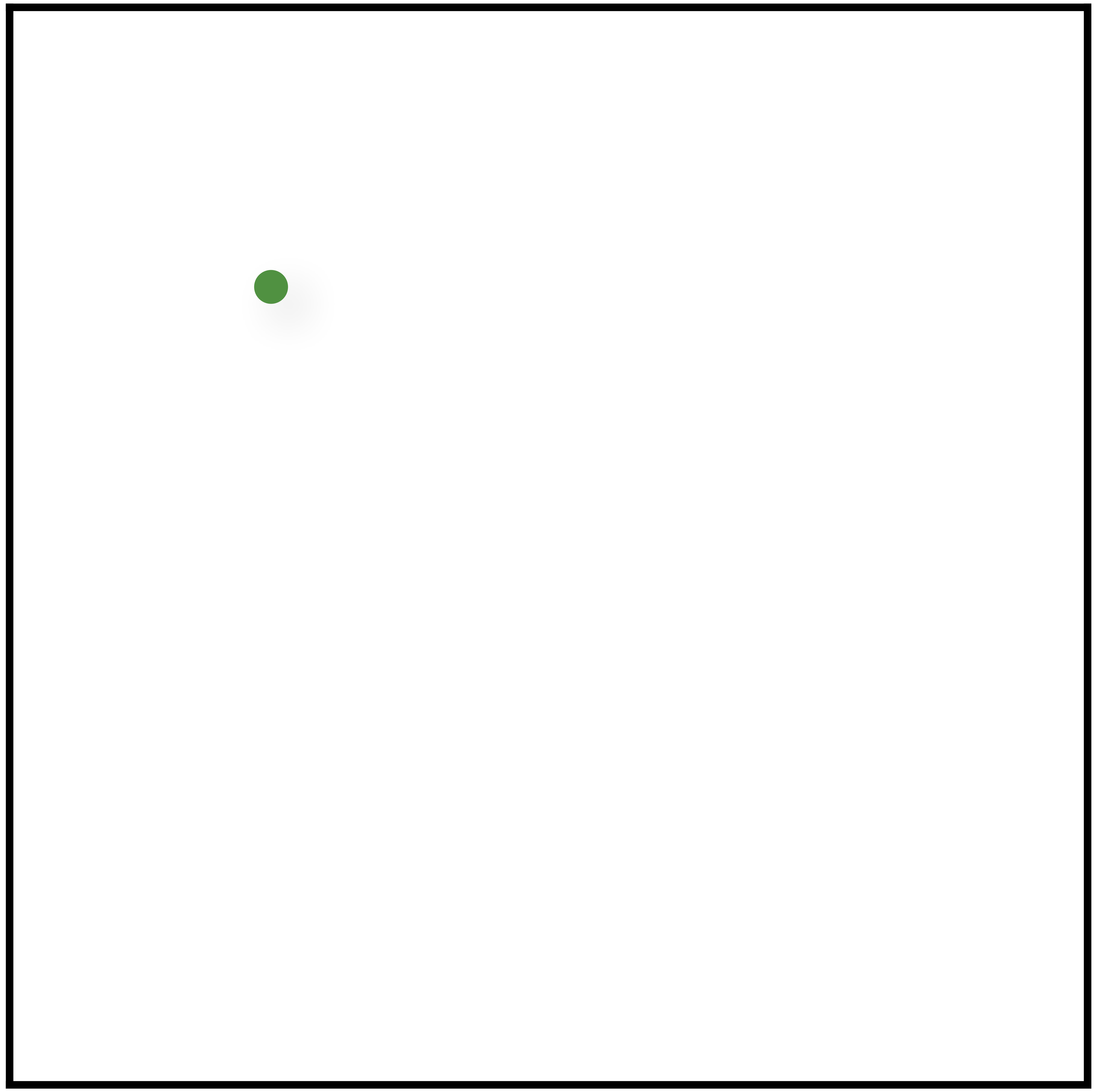
Poisson-Disk Sampling:

- Mark A. Z. Dippé and Erling Henry Wold. "Antialiasing through stochastic sampling." *ACM SIGGRAPH*, 1985.
- Robert L. Cook. "Stochastic sampling in computer graphics." *ACM Transactions on Graphics*, 1986.
- Ares Lagae and Philip Dutré. "A comparison of methods for generating Poisson disk distributions." *Computer Graphics Forum*, 2008.

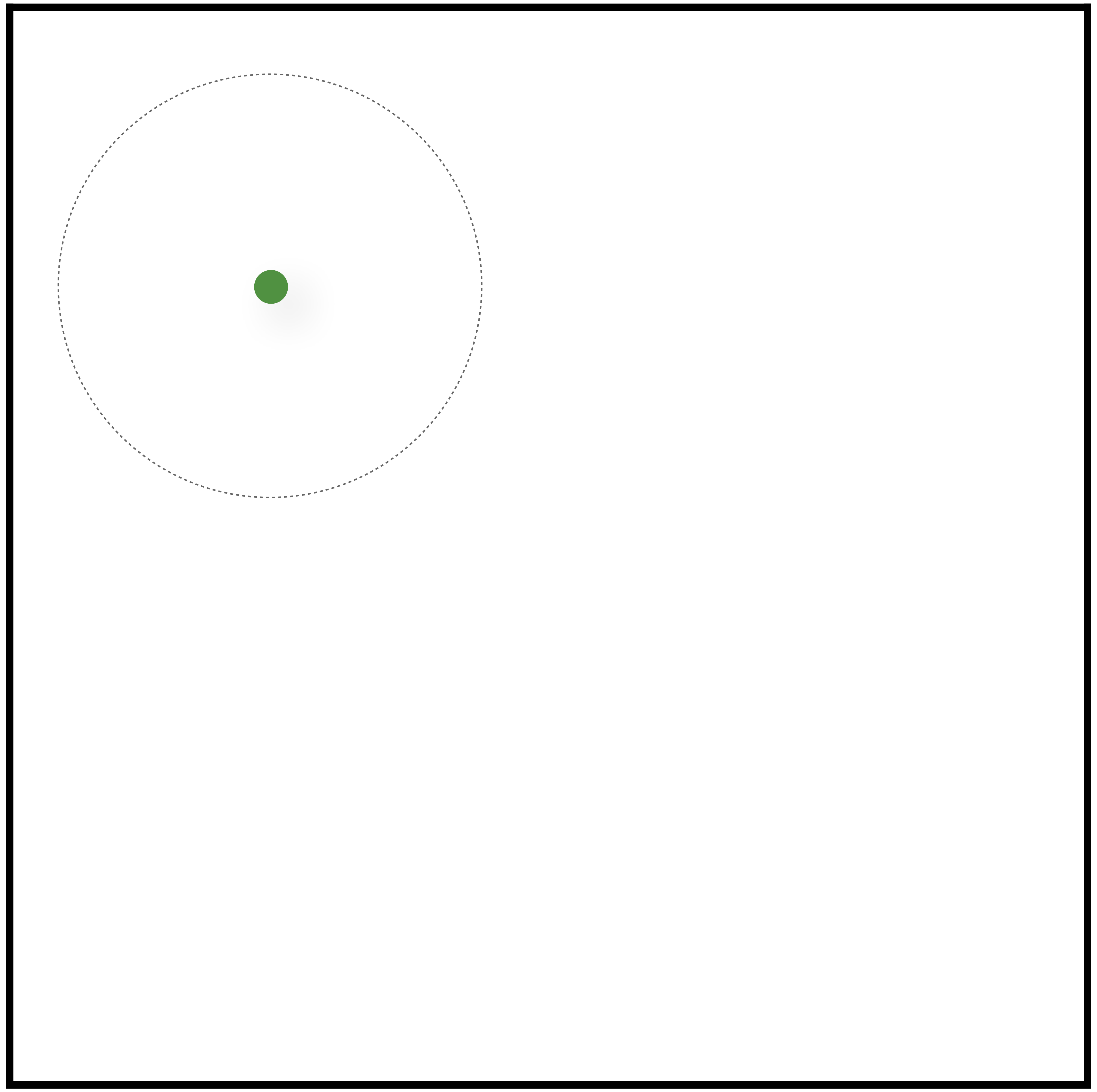
Random Dart Throwing



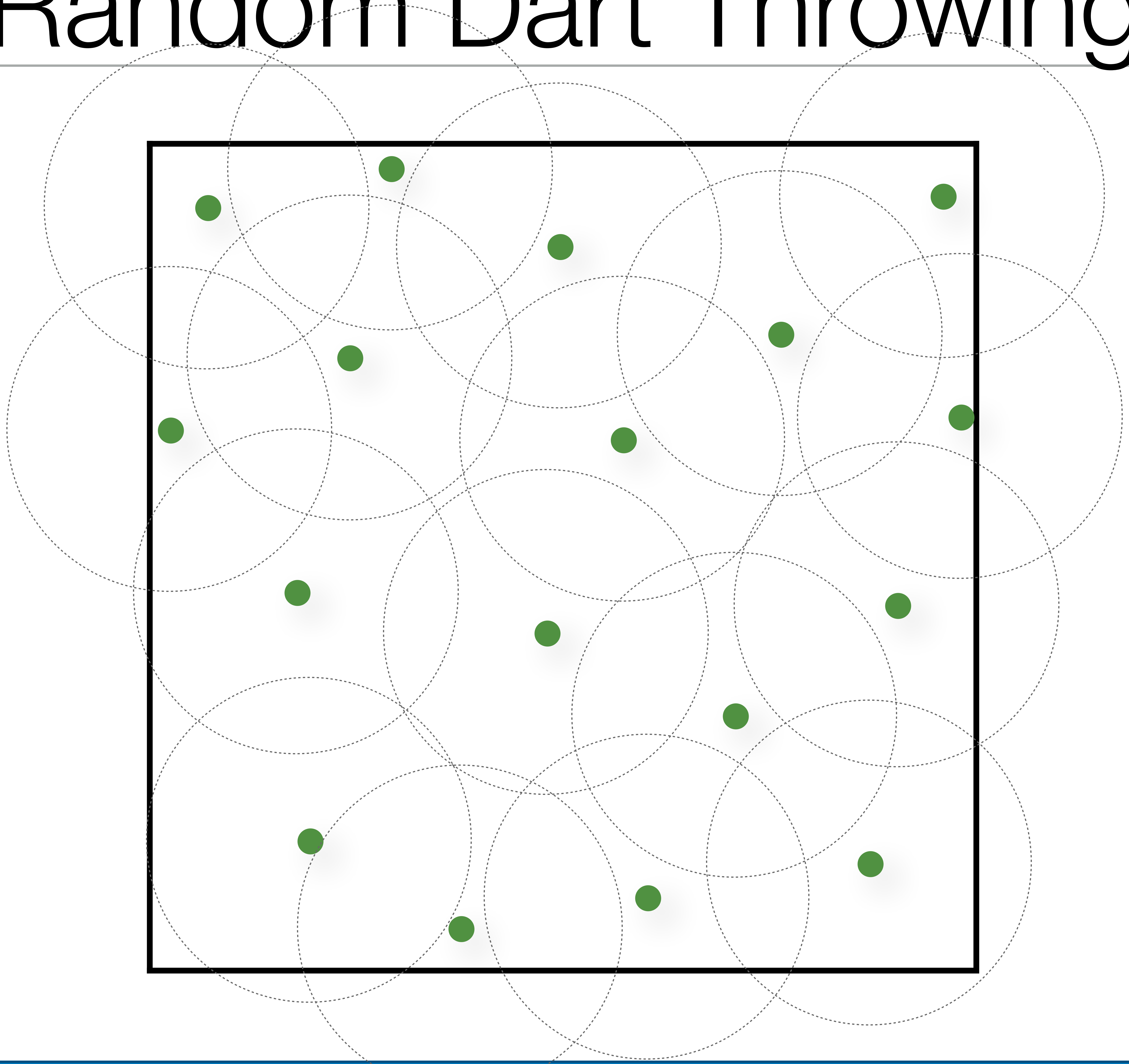
Random Dart Throwing



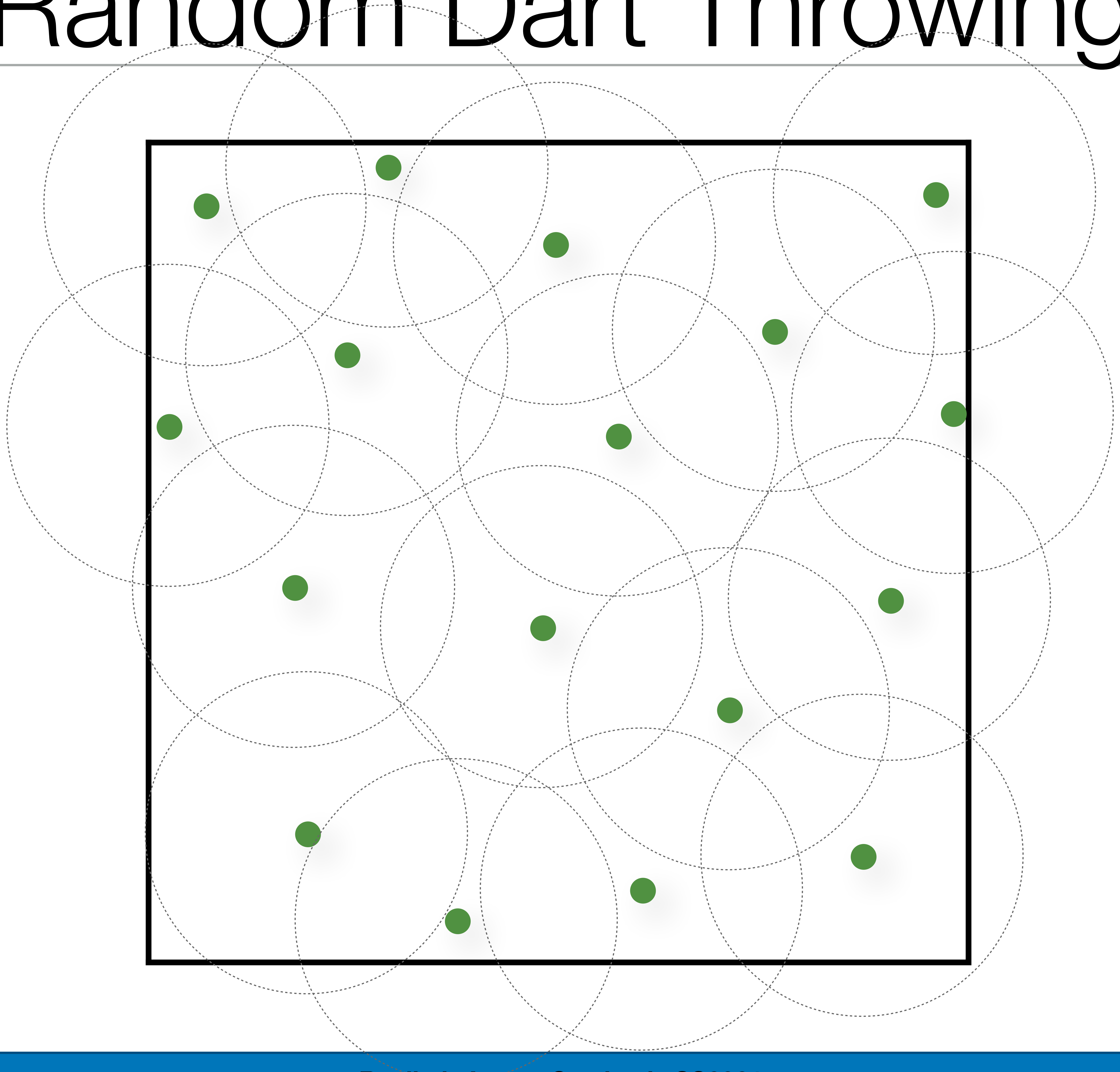
Random Dart Throwing



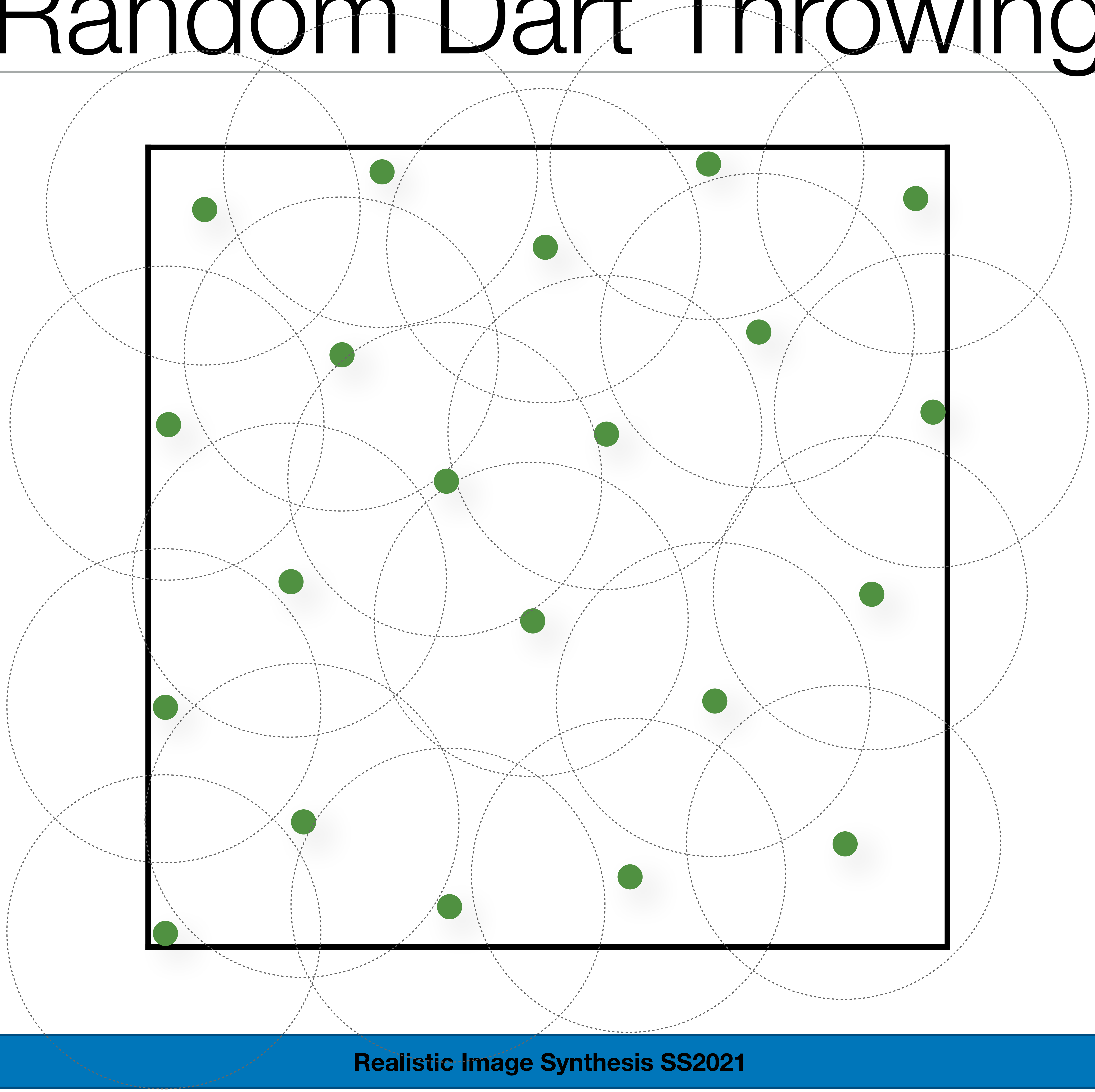
Random Dart Throwing



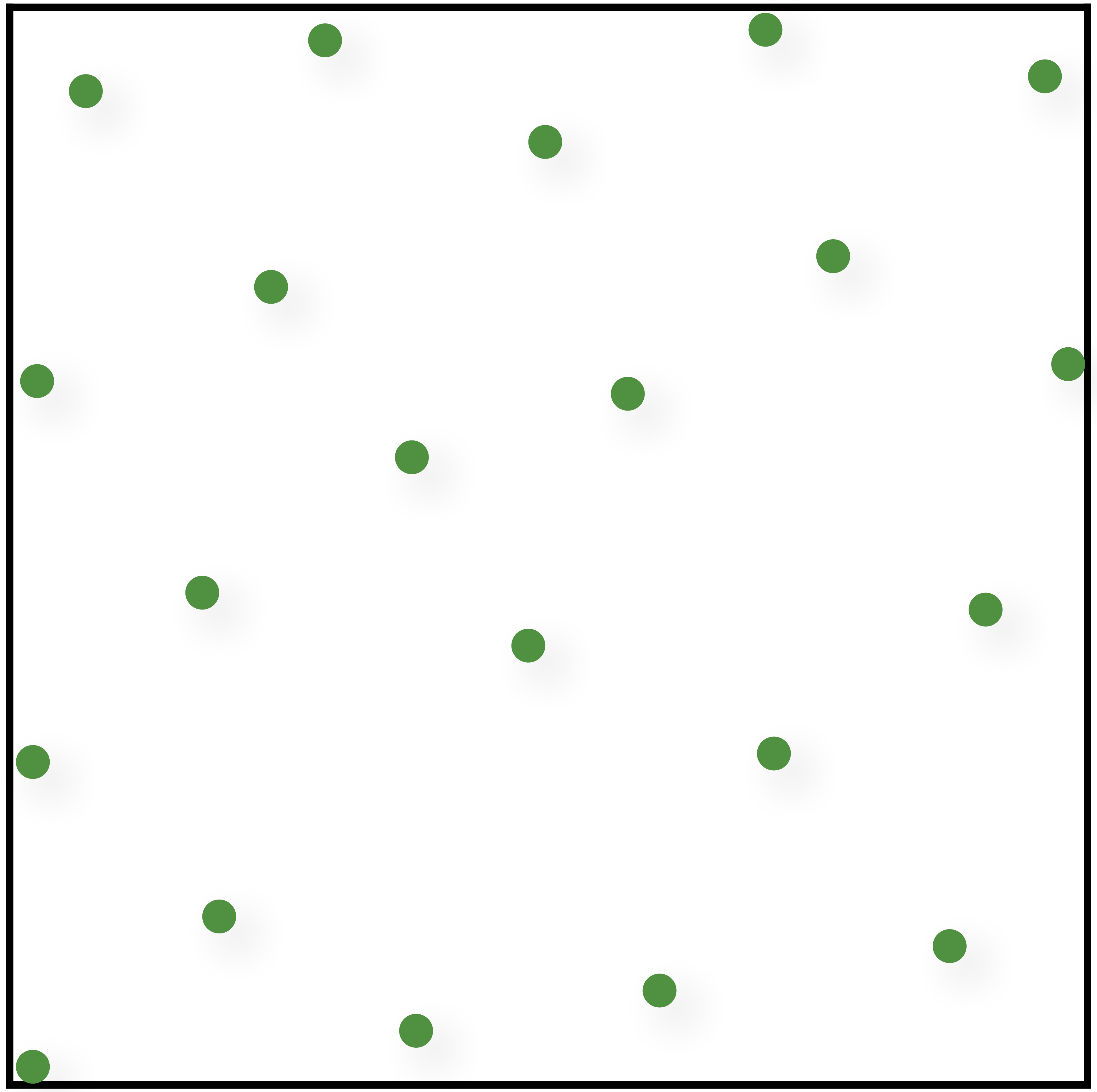
Random Dart Throwing



Random Dart Throwing

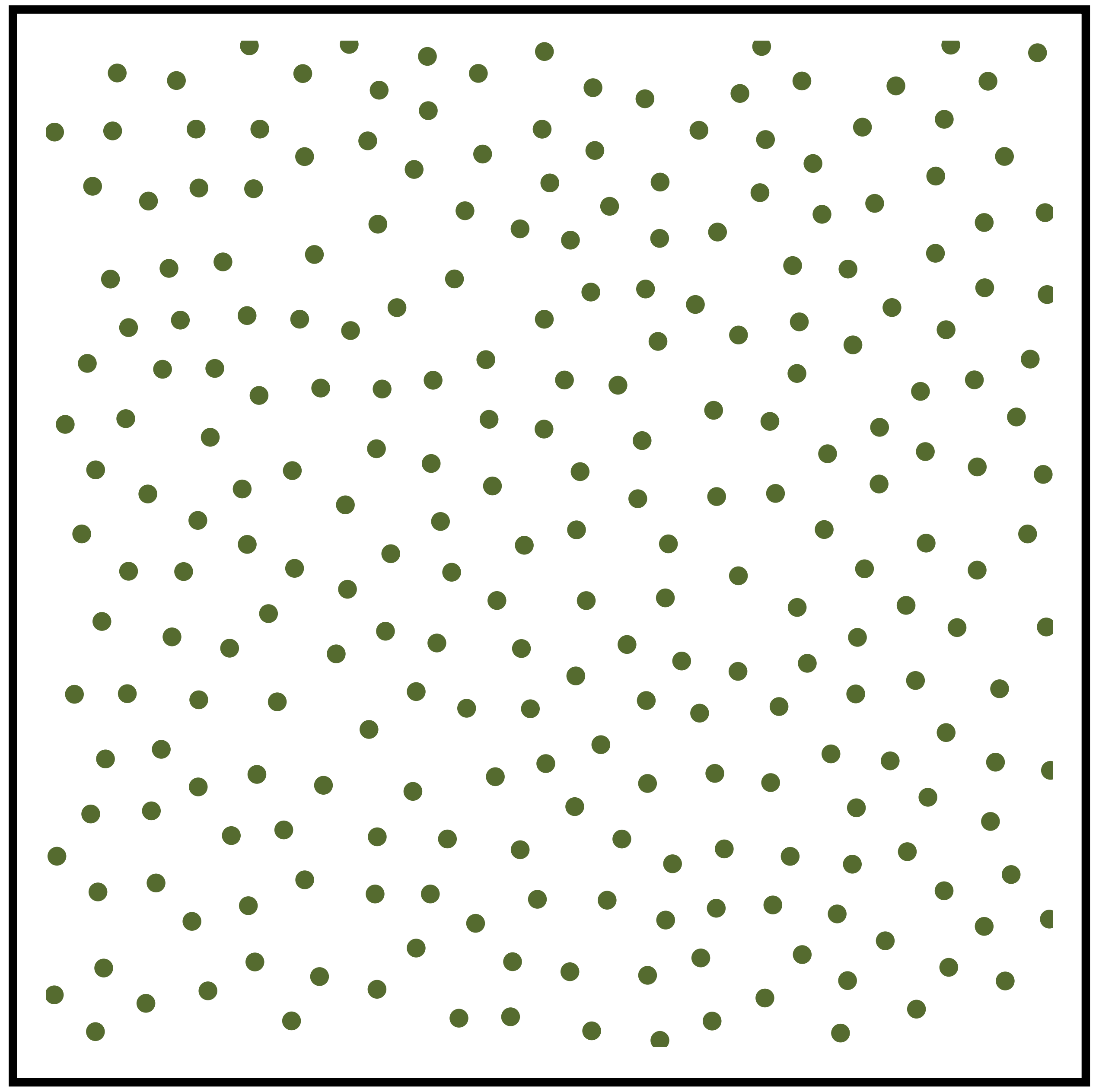


Random Dart Throwing

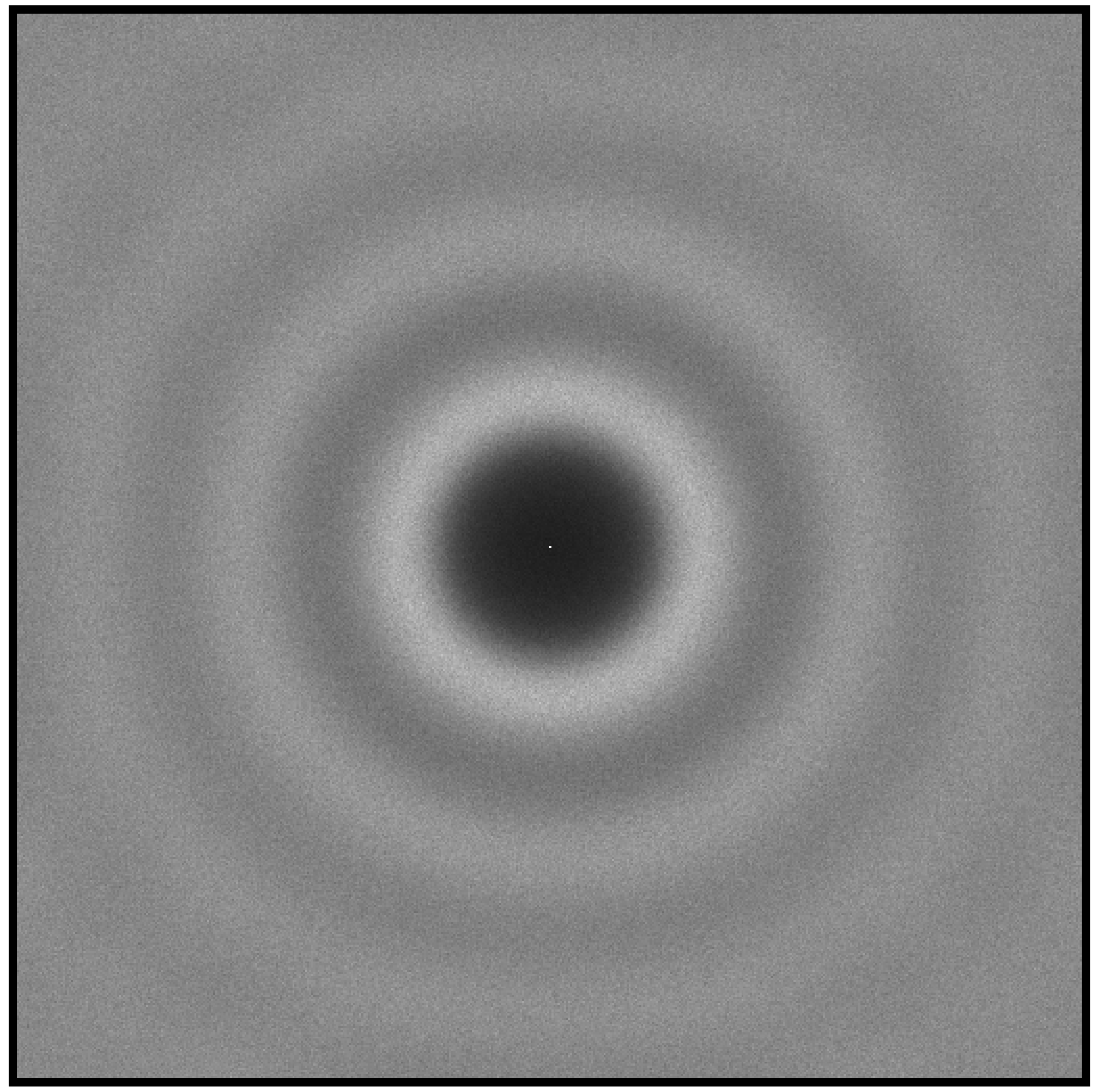


Poisson Disk Sampling

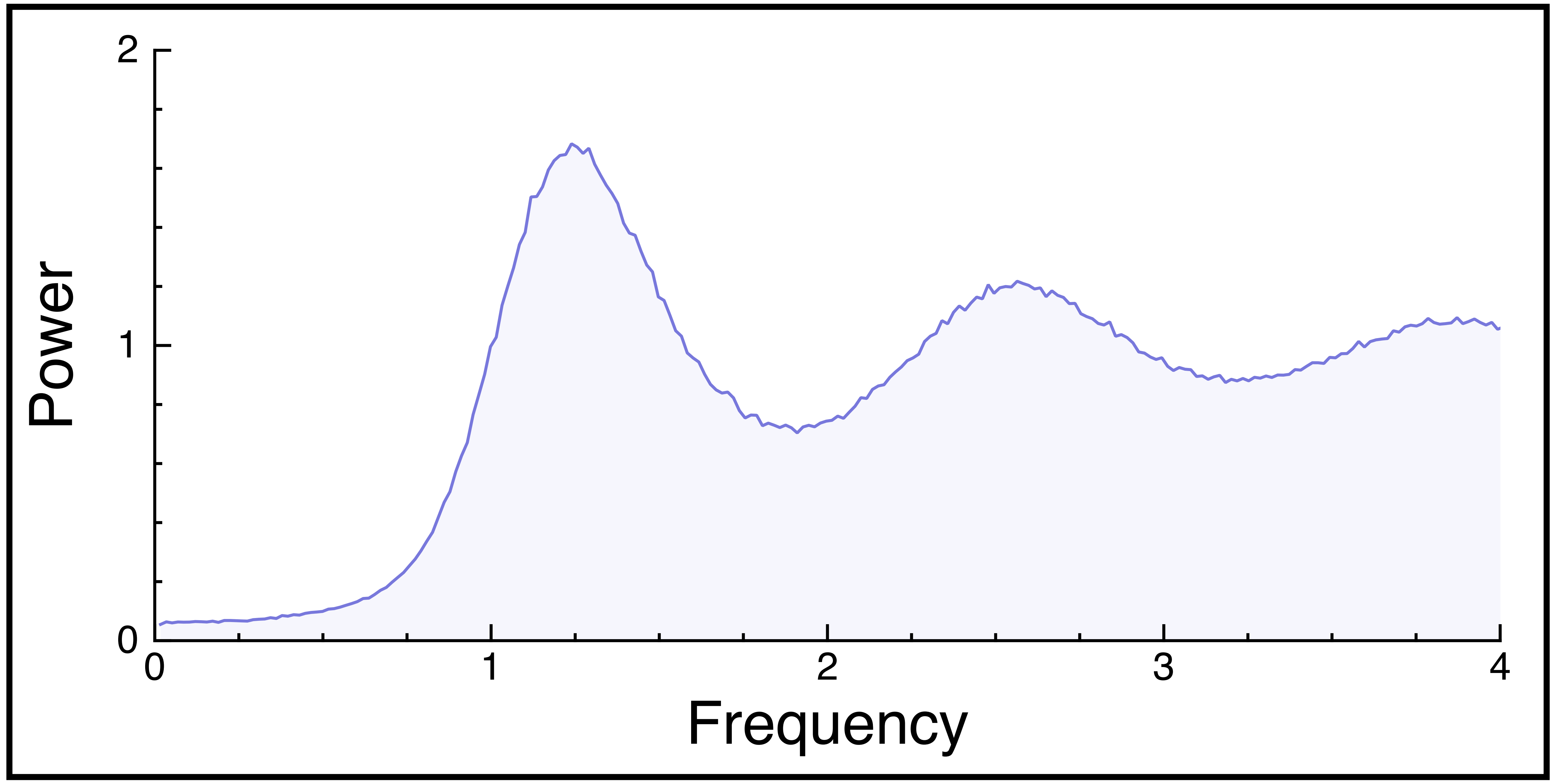
Samples



Expected power spectrum



Radial mean



Blue-Noise Sampling (Relaxation-based)

Blue-Noise Sampling (Relaxation-based)

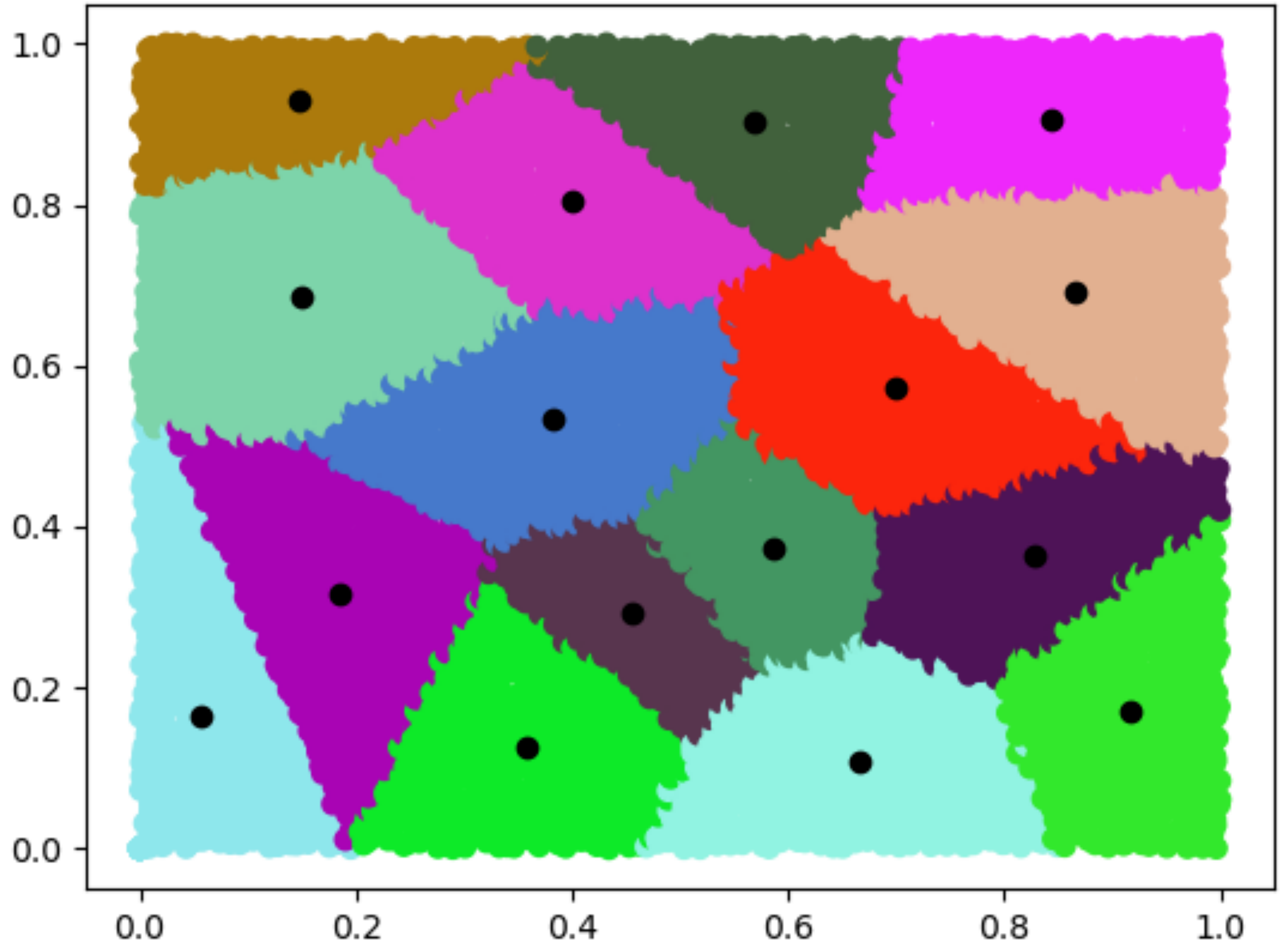
1. Initialize sample positions (e.g. random)

Blue-Noise Sampling (Relaxation-based)

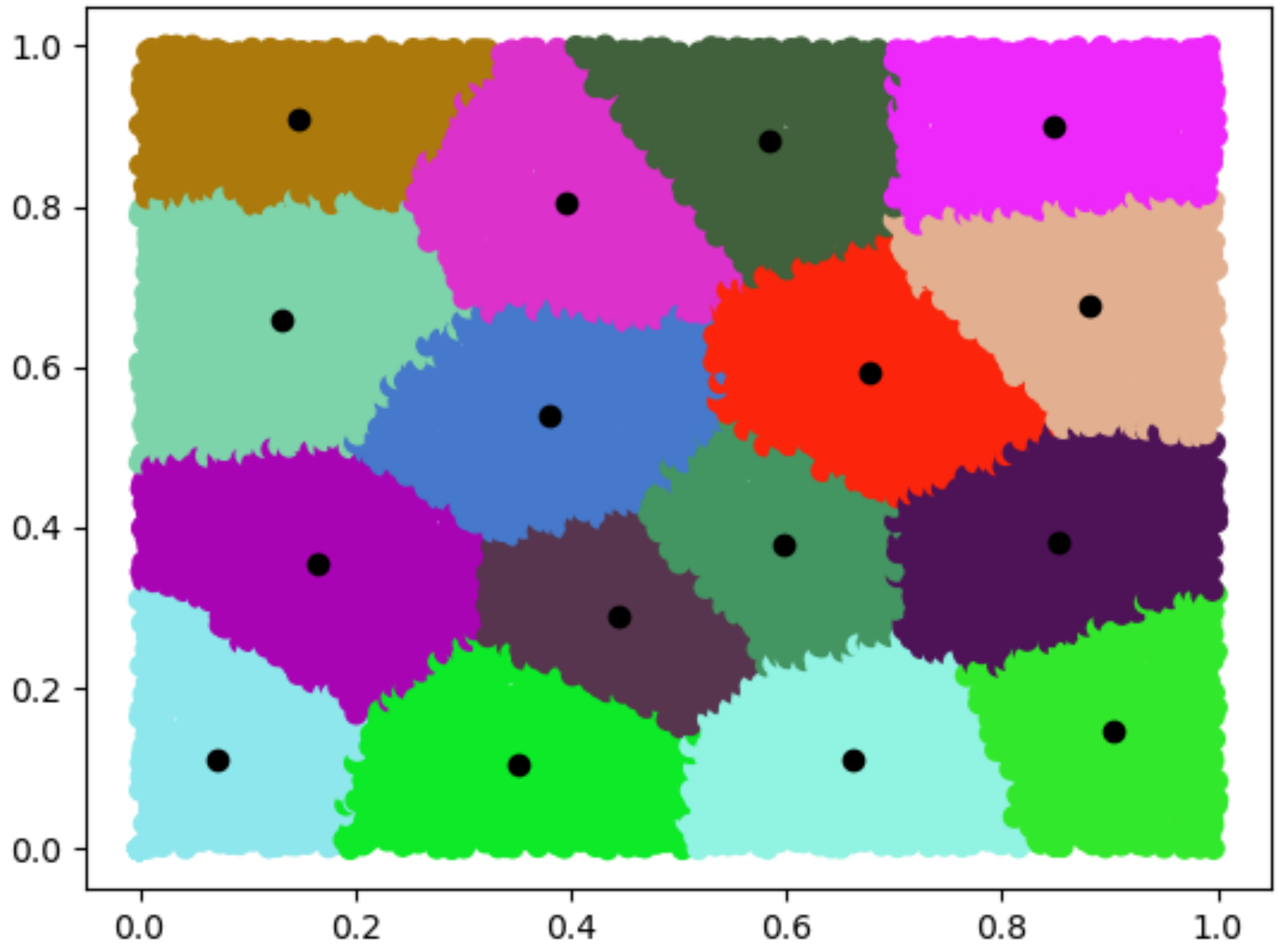
1. Initialize sample positions (e.g. random)
2. Use an iterative relaxation to move samples away from each other.

Lloyd-Relaxation Method

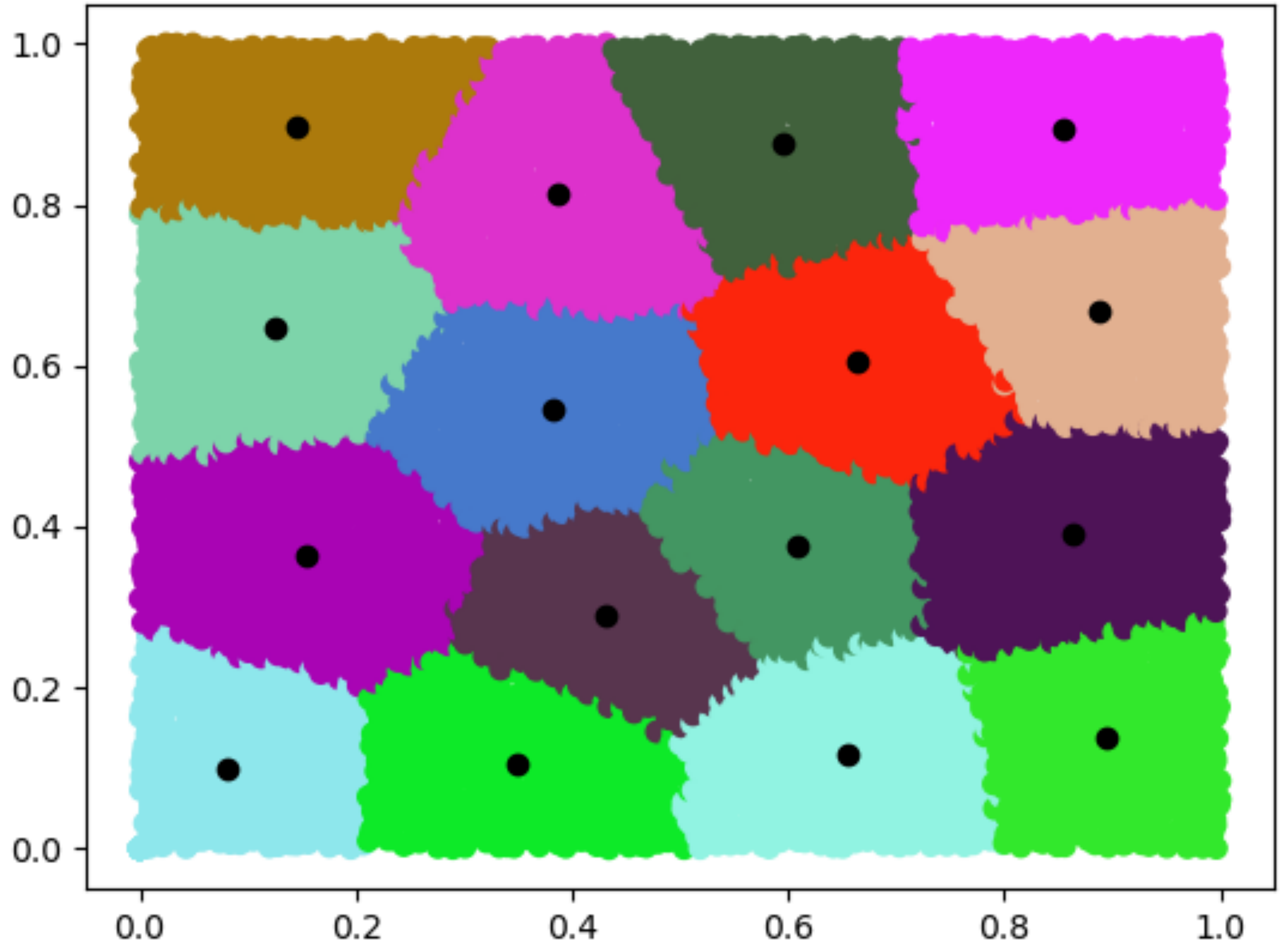
Lloyd-Relaxation Method



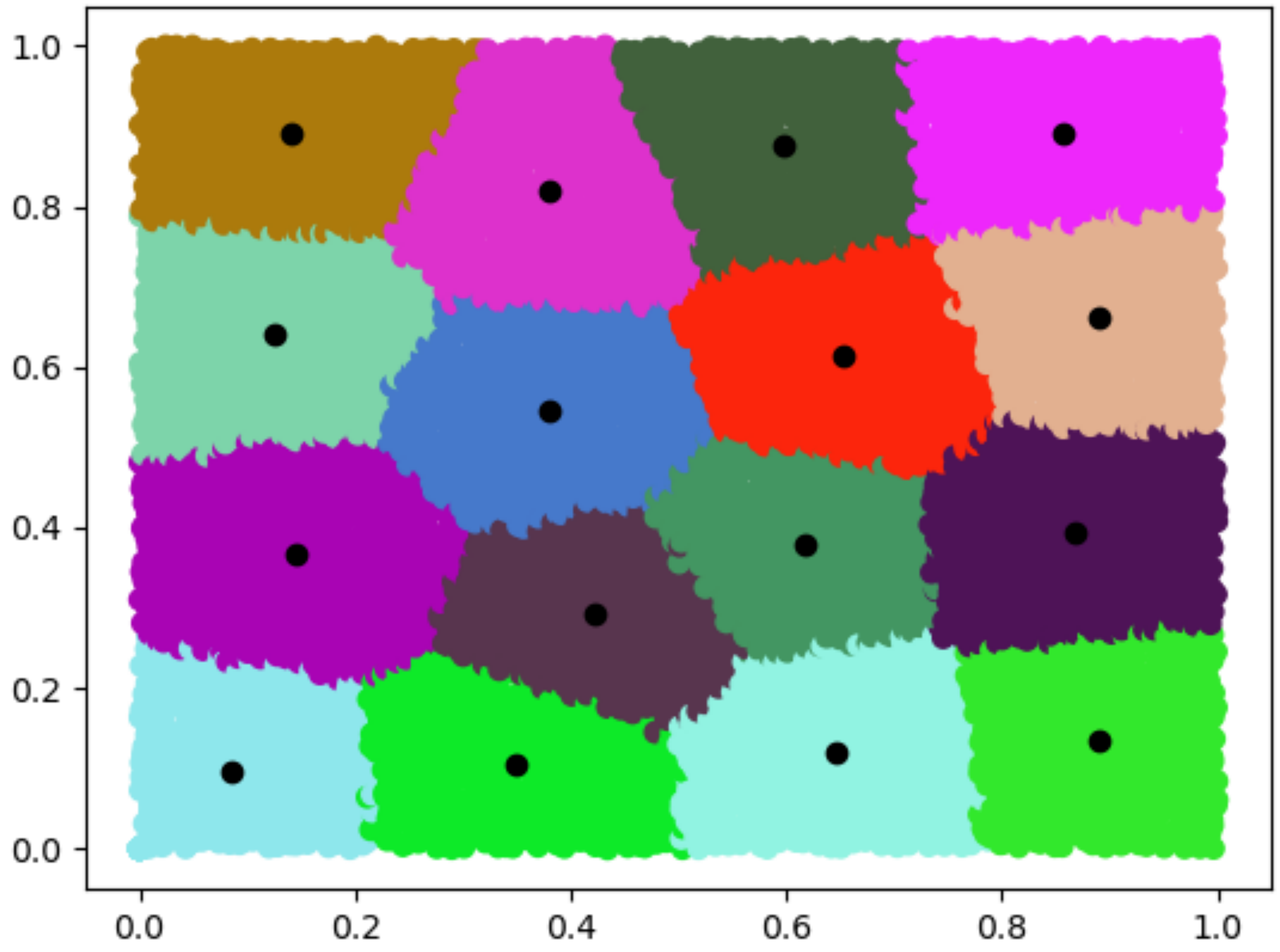
Lloyd-Relaxation Method



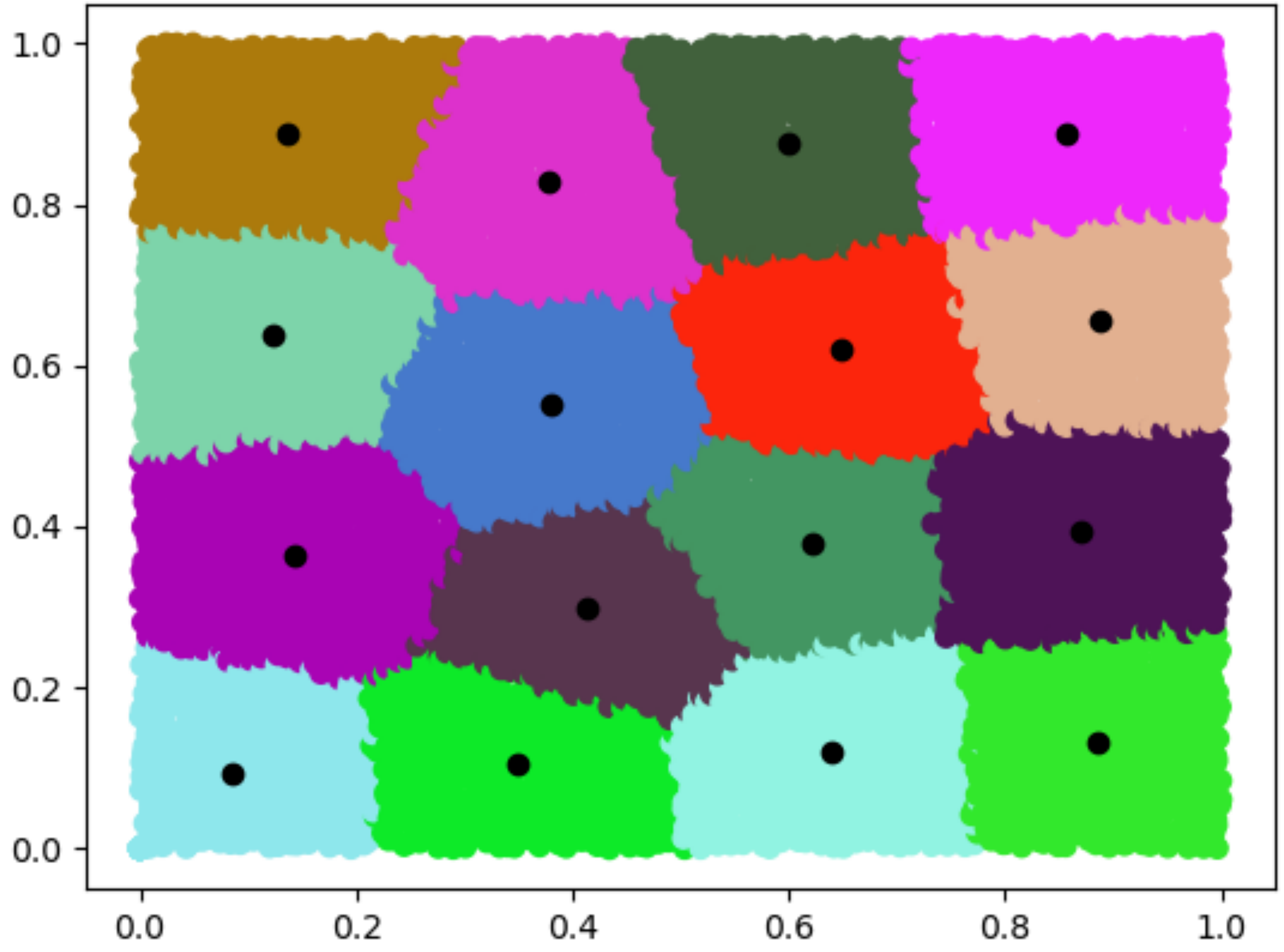
Lloyd-Relaxation Method



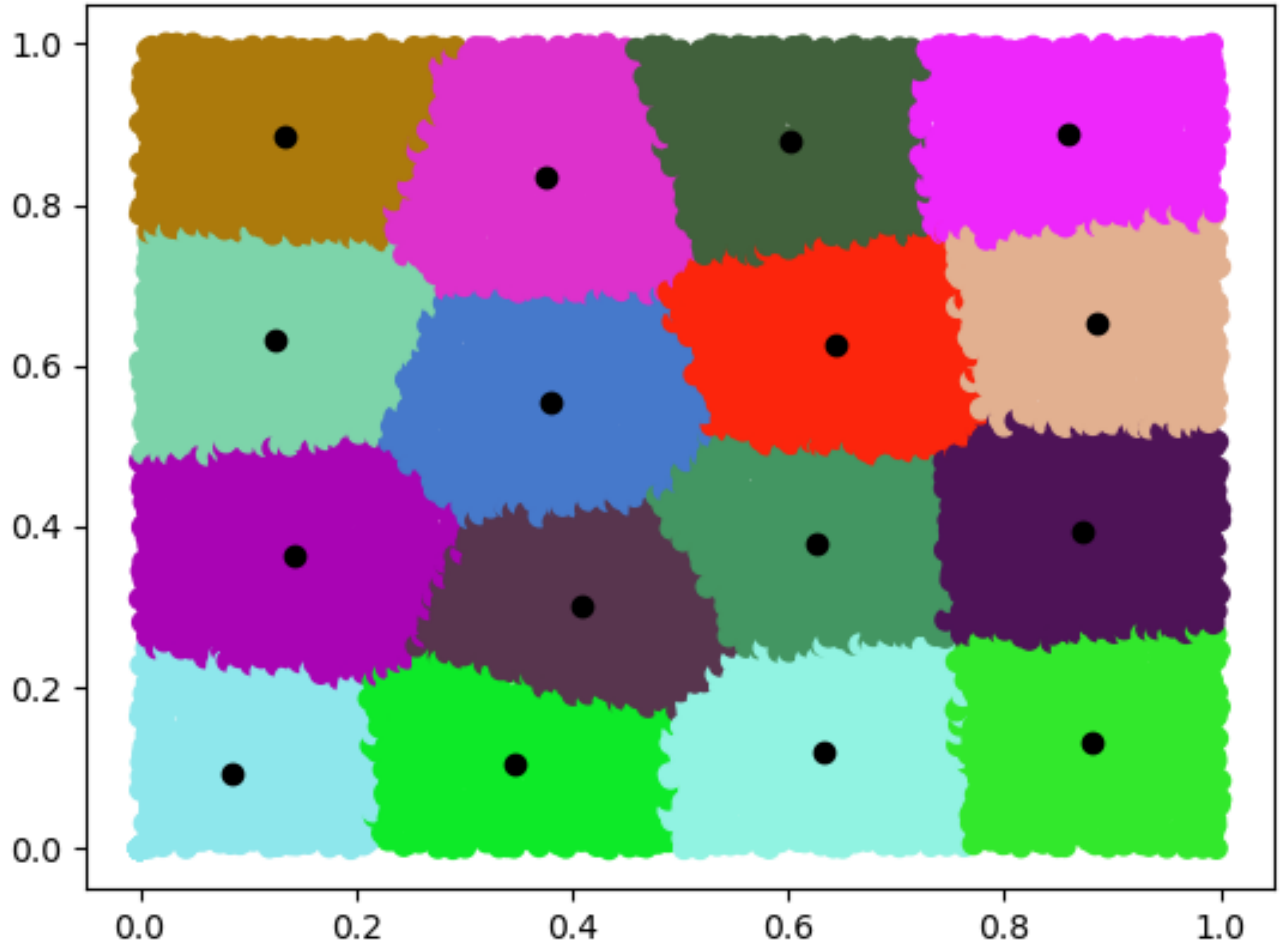
Lloyd-Relaxation Method



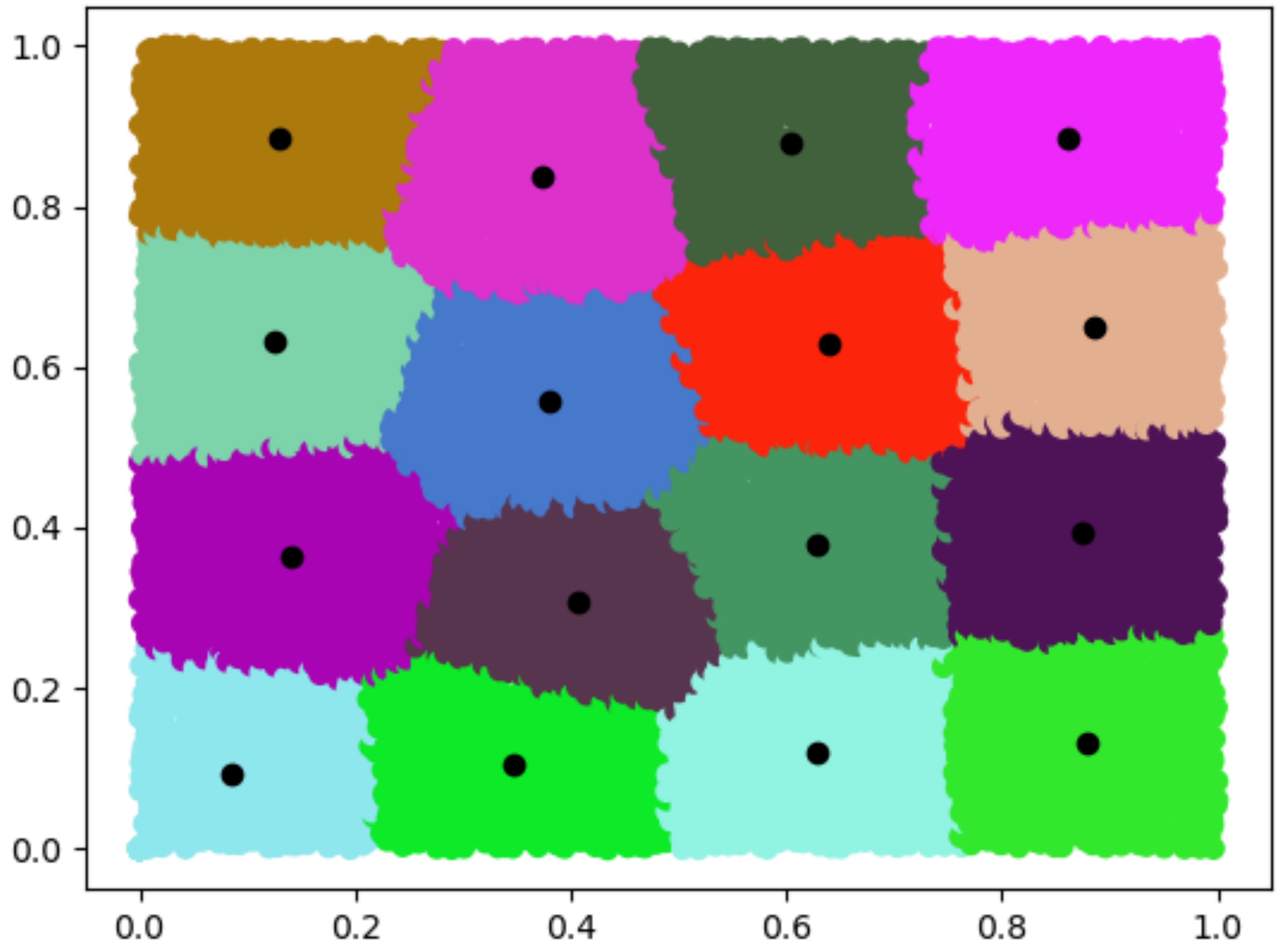
Lloyd-Relaxation Method



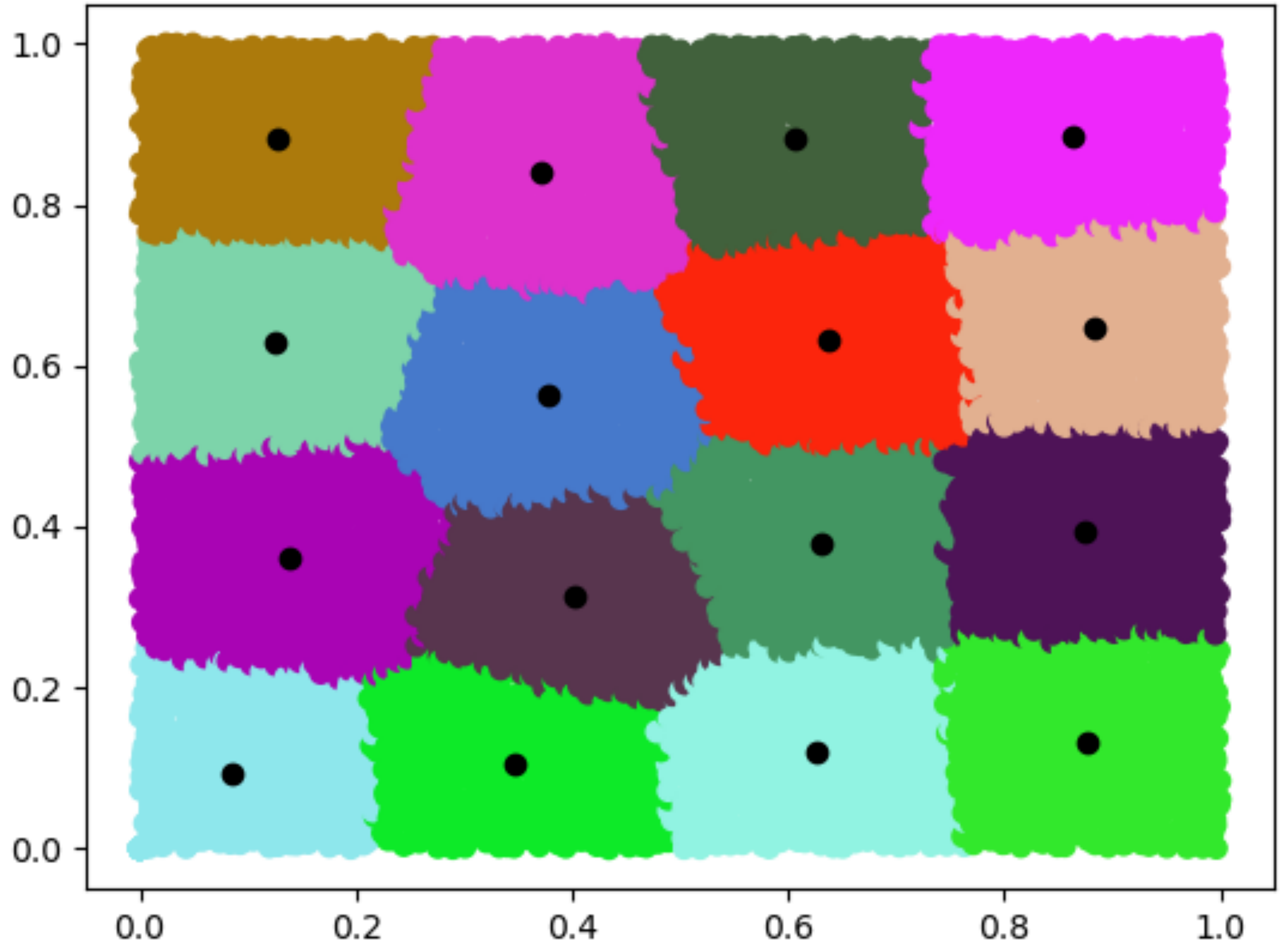
Lloyd-Relaxation Method



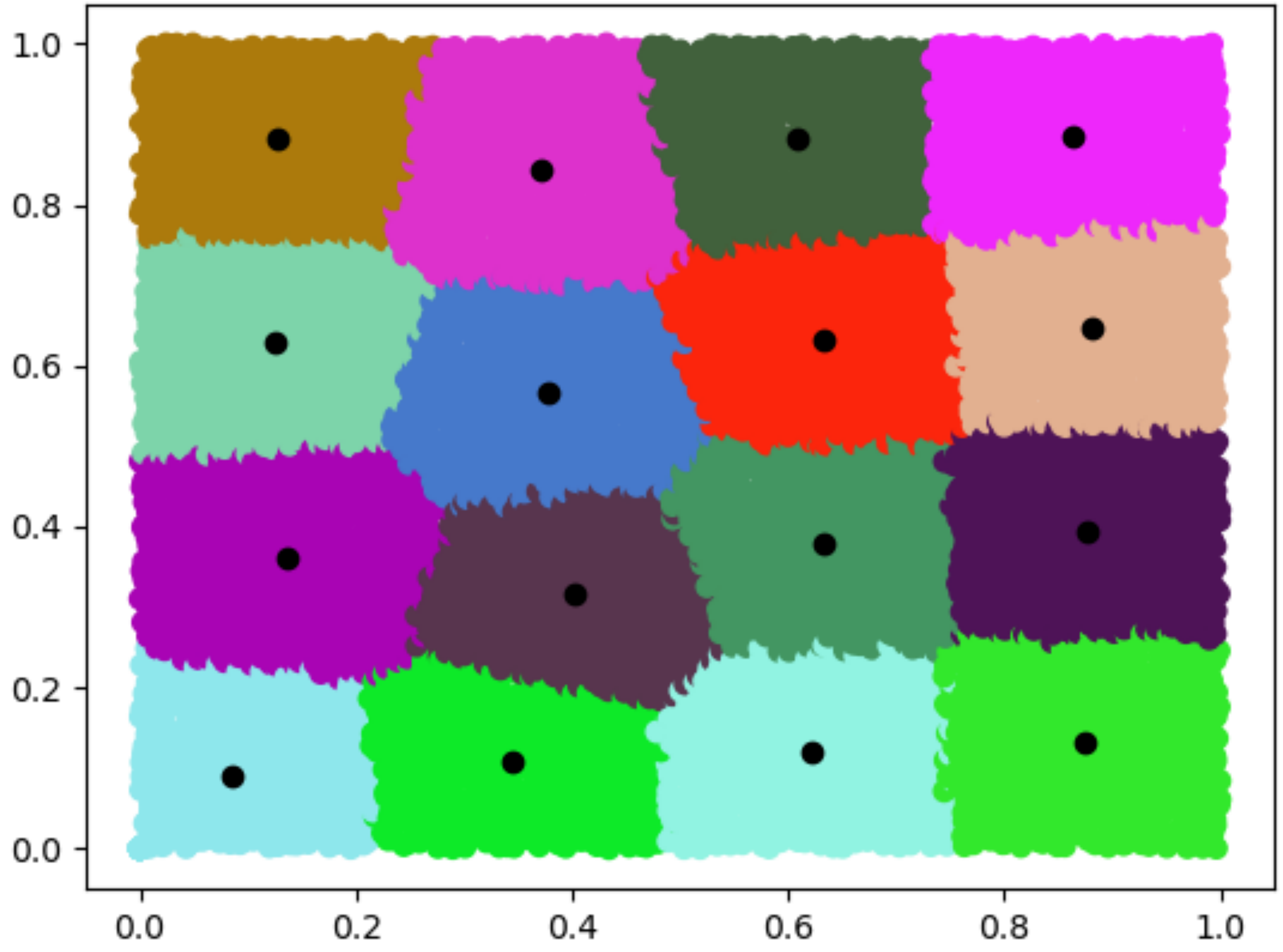
Lloyd-Relaxation Method



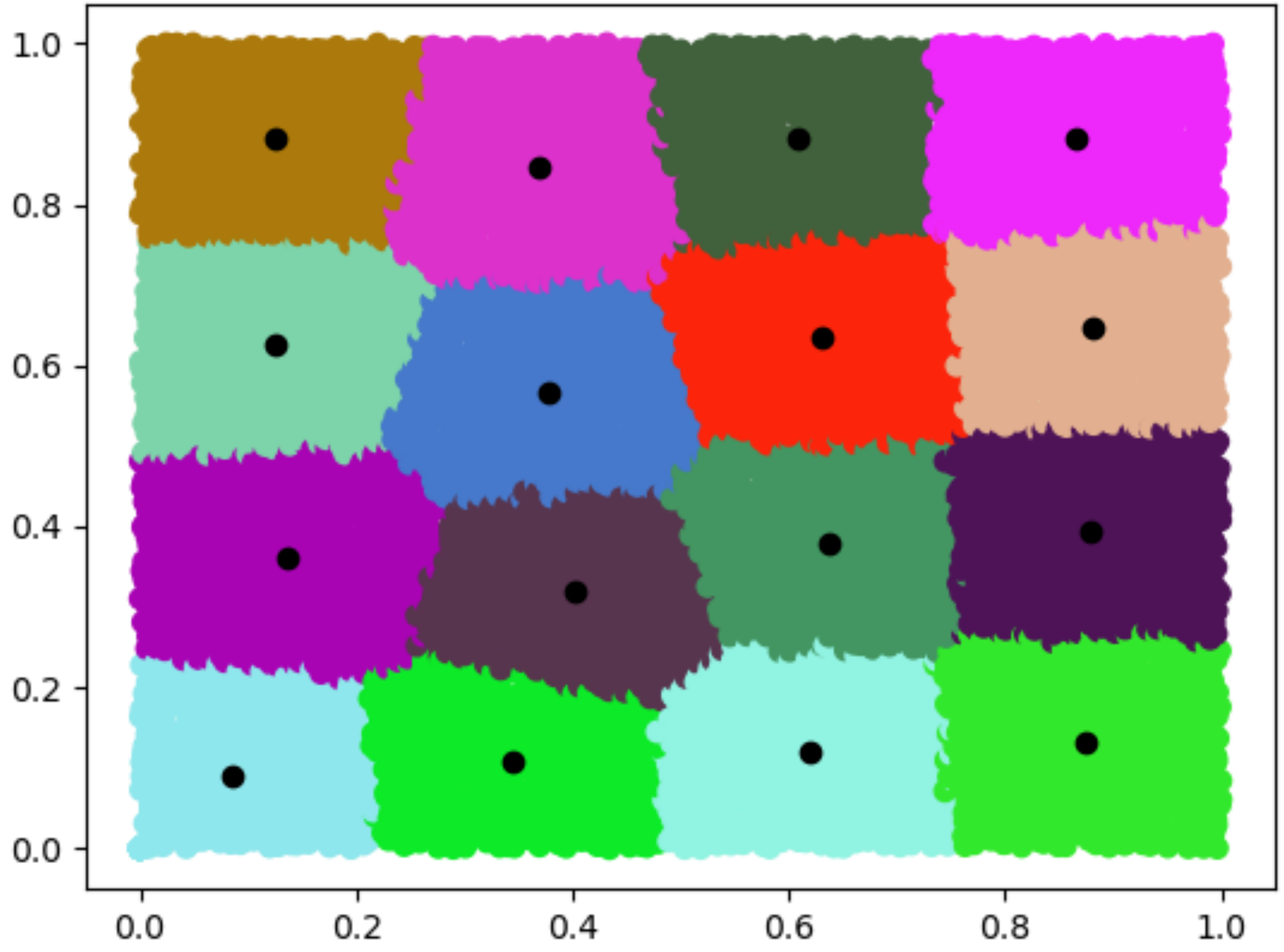
Lloyd-Relaxation Method



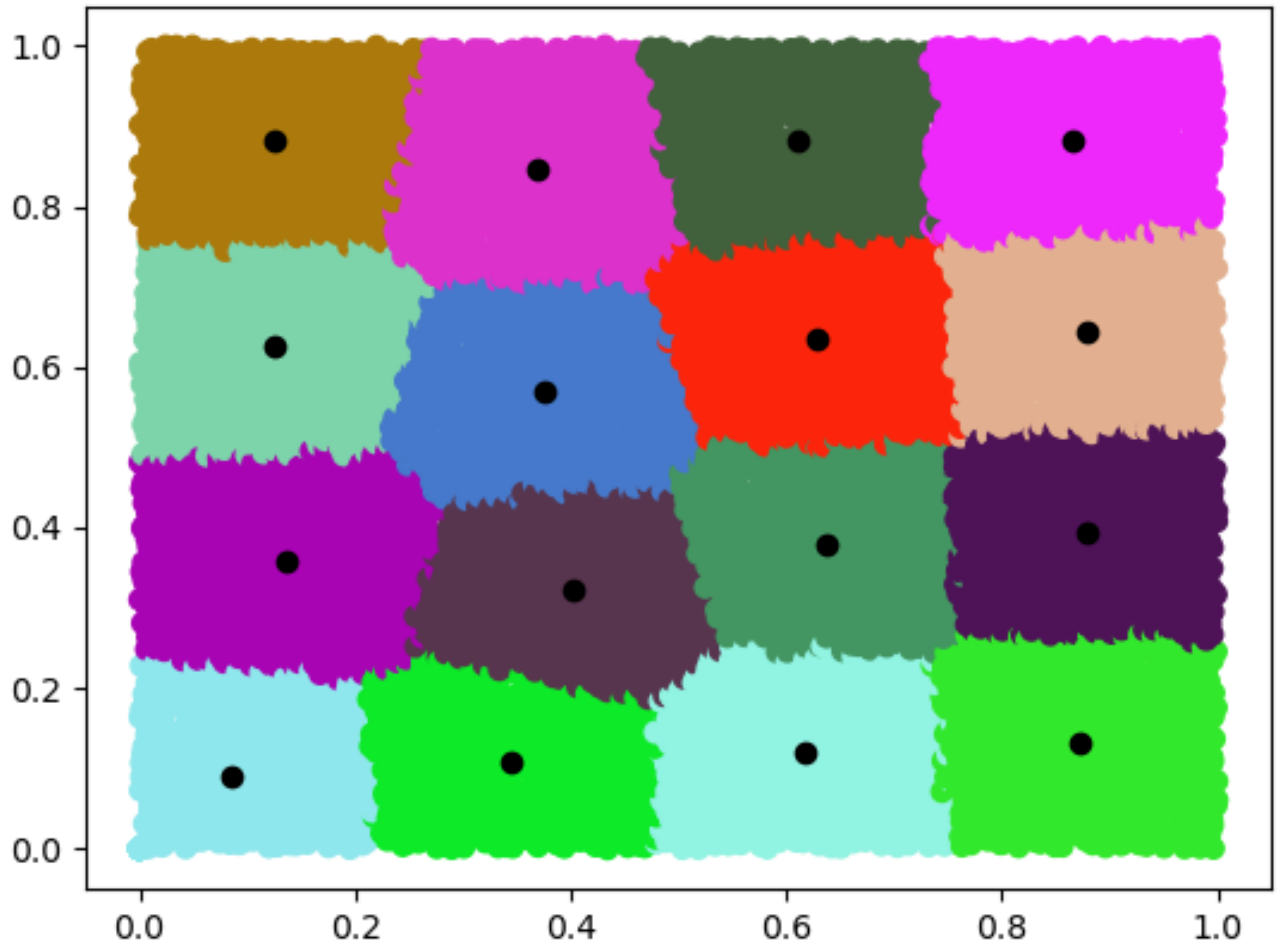
Lloyd-Relaxation Method



Lloyd-Relaxation Method



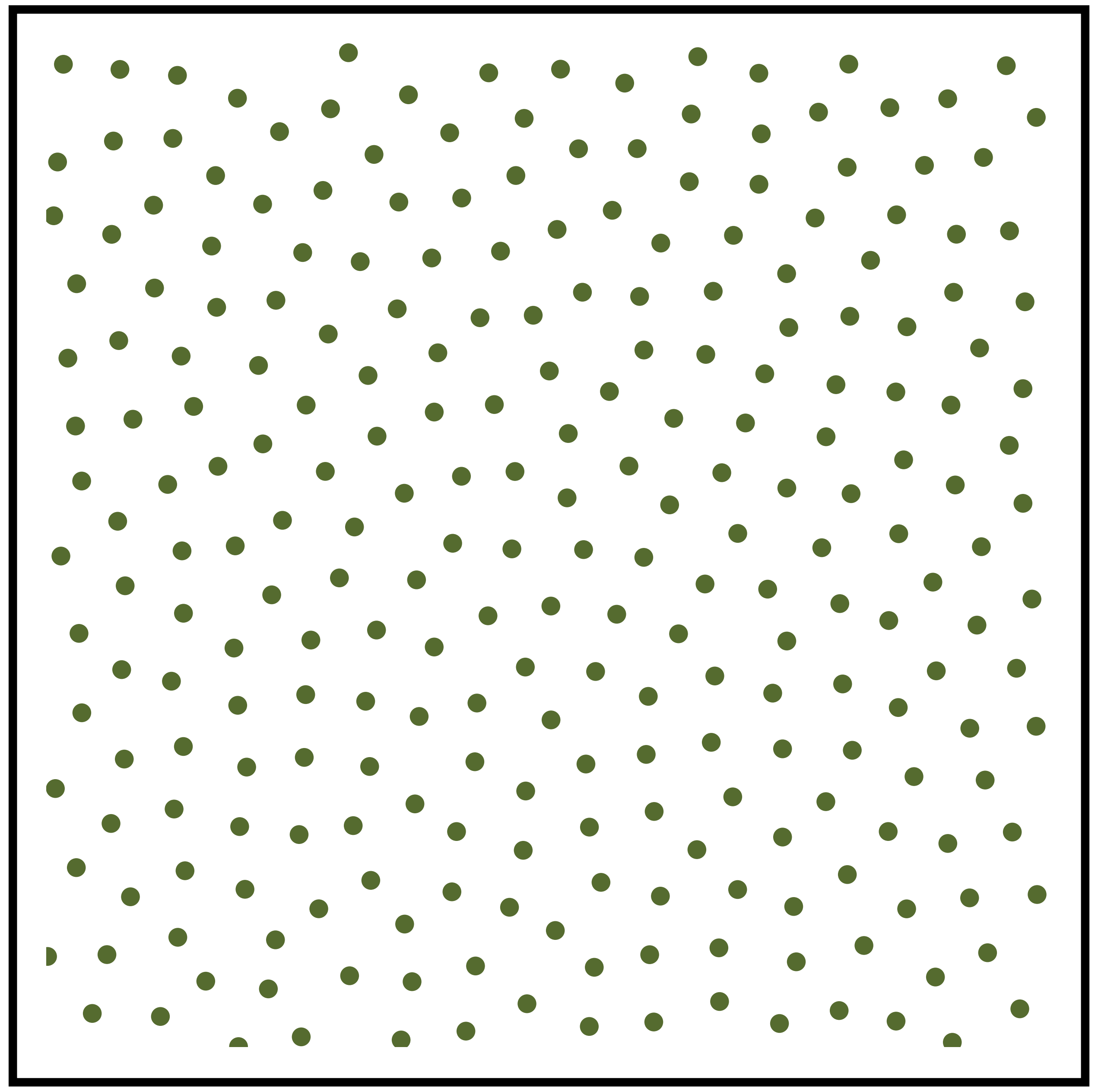
Lloyd-Relaxation Method



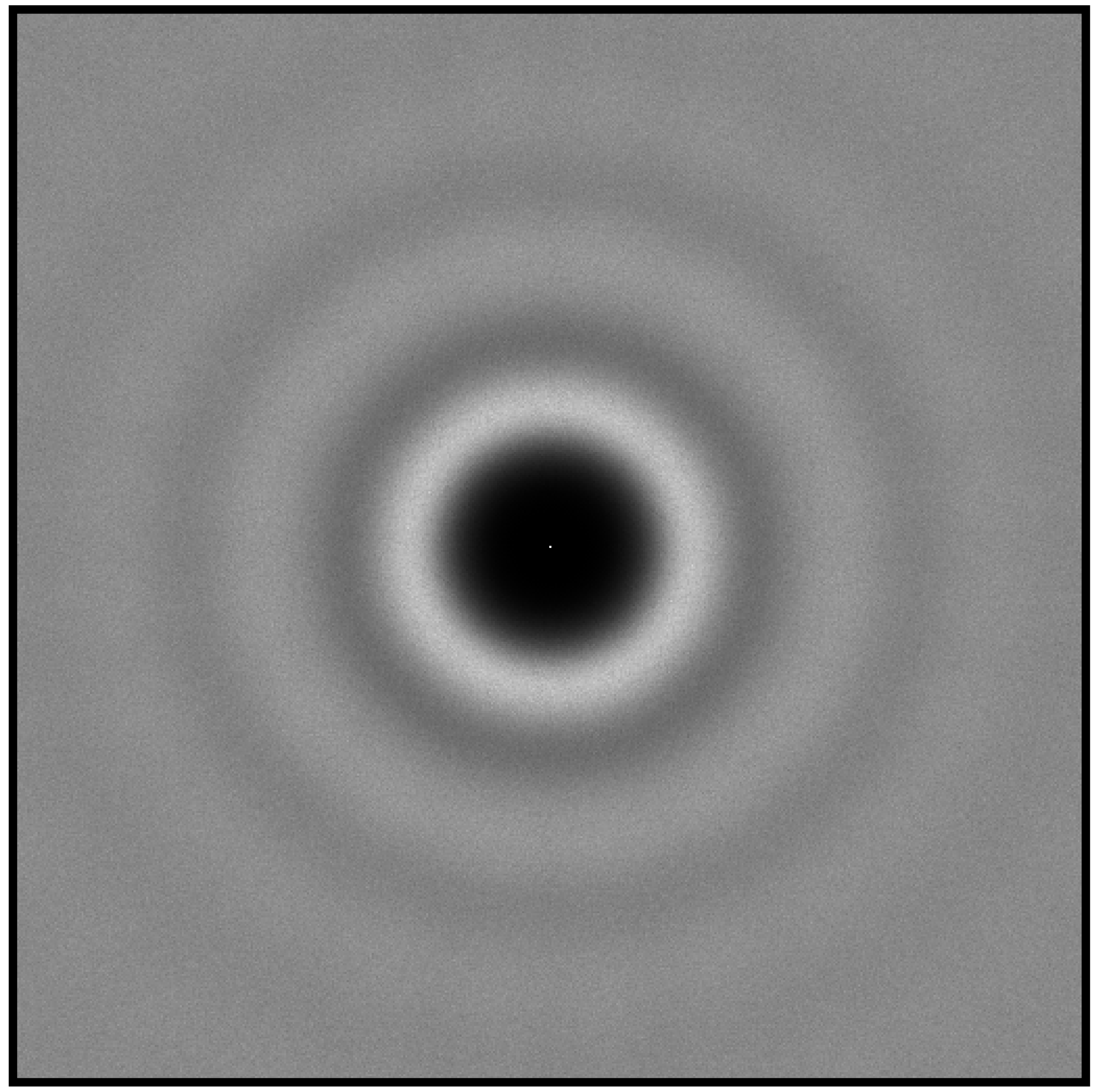
CCVT Sampling [Balzer et al. 2009]

CCVT Sampling [Balzer et al. 2009]

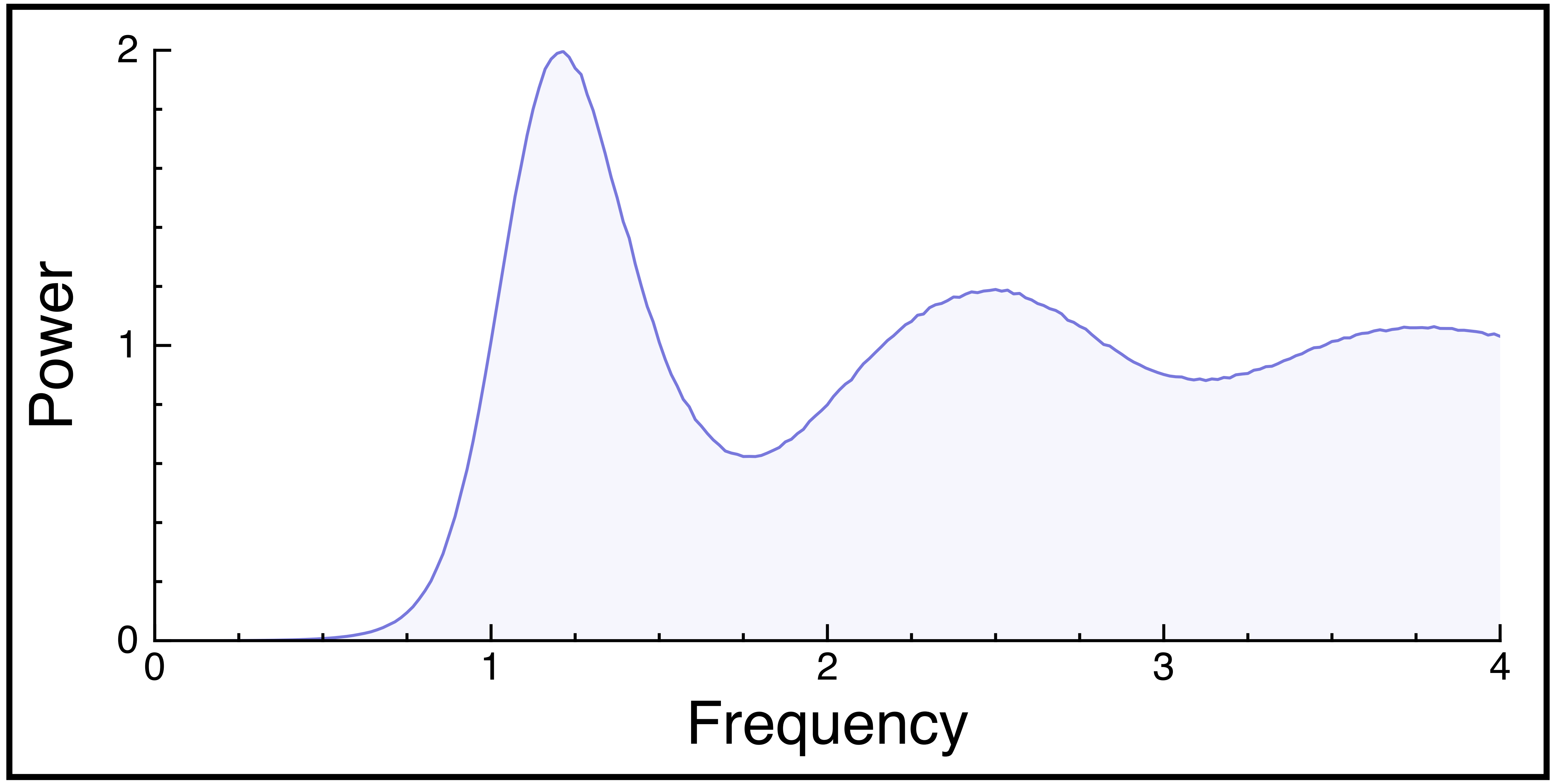
Samples



Expected power spectrum

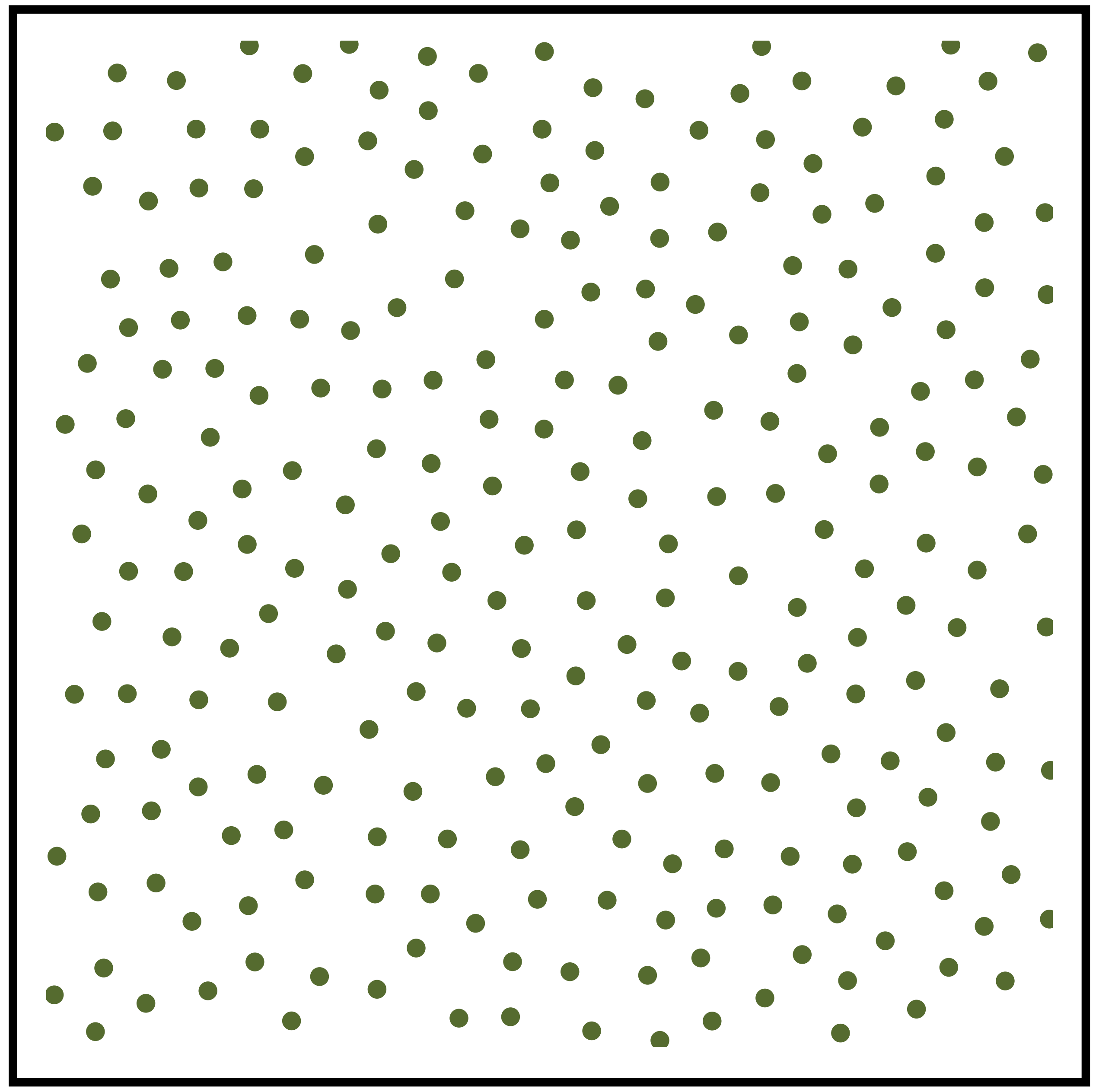


Radial mean

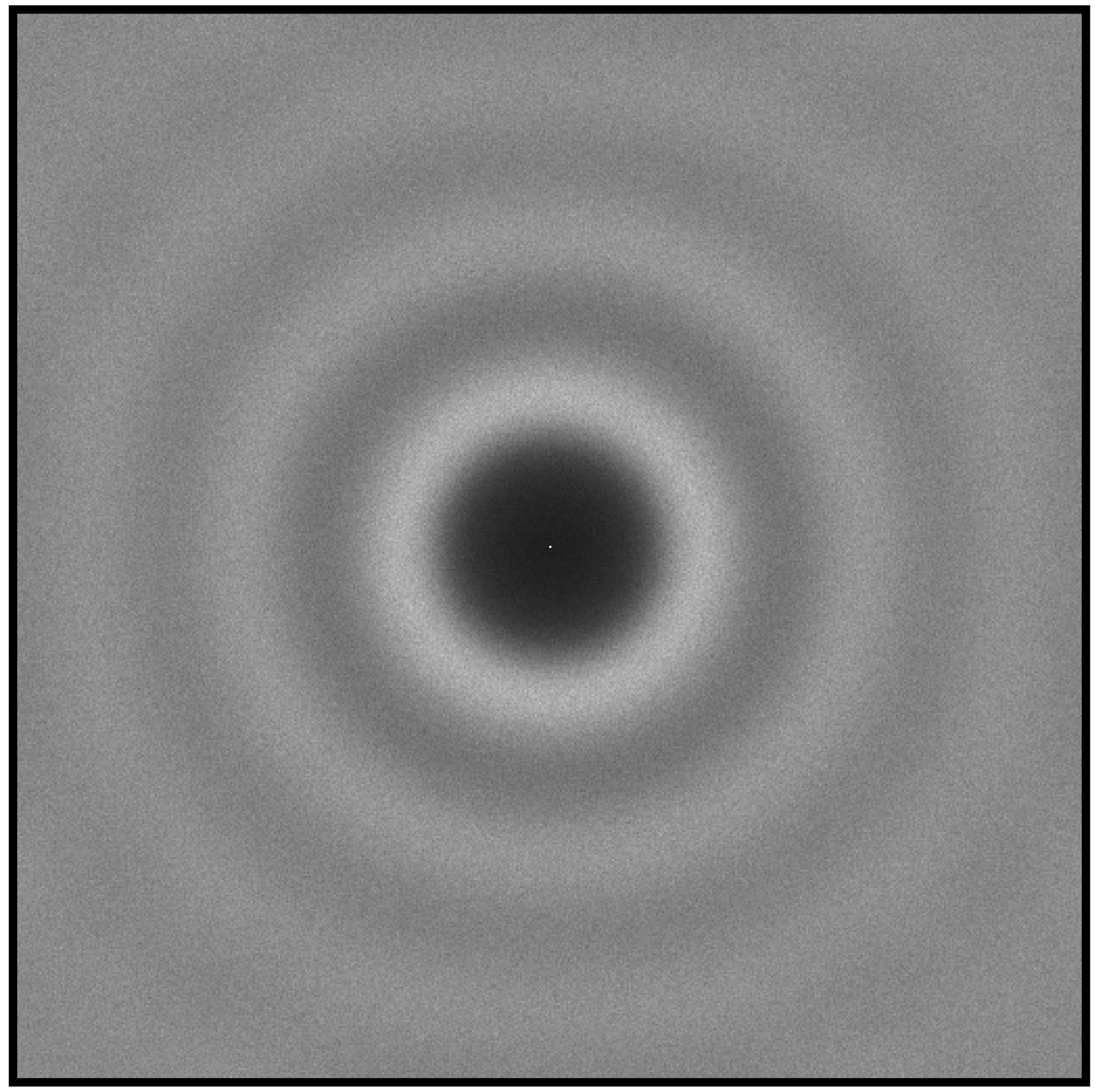


Poisson Disk Sampling

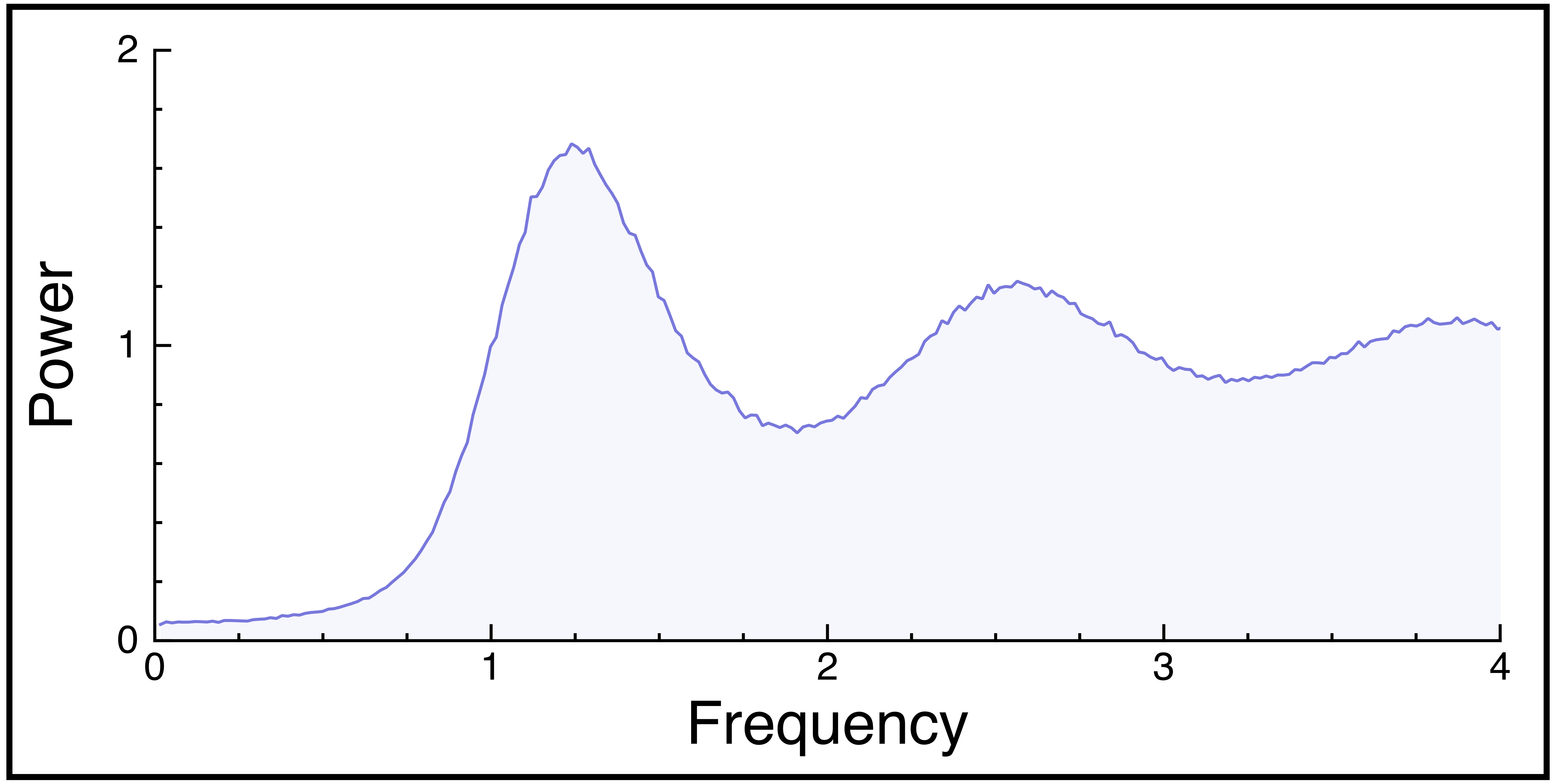
Samples



Expected power spectrum



Radial mean



Low-Discrepancy Sampling

Deterministic sets of points specially crafted to be evenly distributed (have low discrepancy).

Entire field of study called Quasi-Monte Carlo (QMC)

The Van der Corput Sequence

Radical Inverse Φ_b in base 2

Subsequent points "fall into biggest holes"

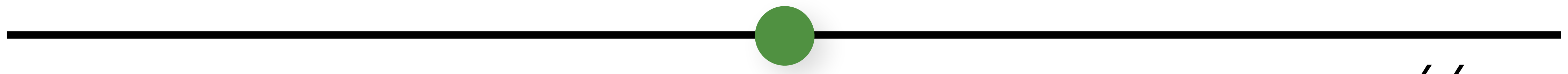
k	Base 2	Φ_b

The Van der Corput Sequence

Radical Inverse Φ_b in base 2

Subsequent points “fall into biggest holes”

k	Base 2	Φ_b
1	1	.1 = 1/2

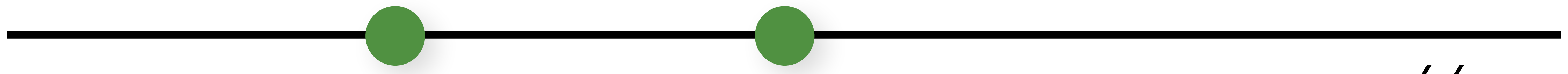


The Van der Corput Sequence

Radical Inverse Φ_b in base 2

Subsequent points “fall into biggest holes”

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

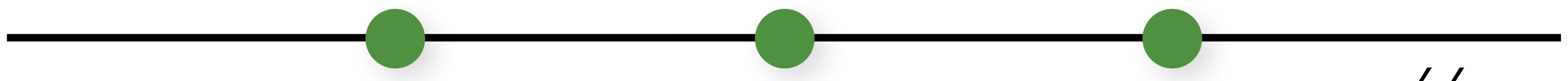


The Van der Corput Sequence

Radical Inverse Φ_b in base 2

Subsequent points “fall into biggest holes”

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

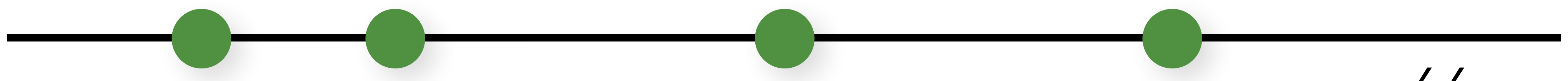


The Van der Corput Sequence

Radical Inverse Φ_b in base 2

Subsequent points “fall into biggest holes”

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

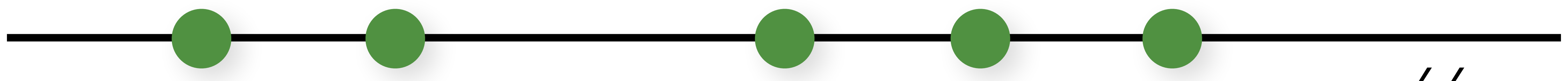


The Van der Corput Sequence

Radical Inverse Φ_b in base 2

Subsequent points “fall into biggest holes”

k	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

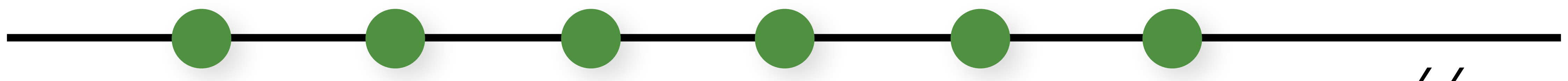


The Van der Corput Sequence

Radical Inverse Φ_b in base 2

Subsequent points “fall into biggest holes”

k	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

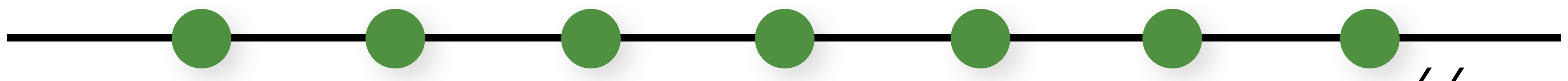


The Van der Corput Sequence

Radical Inverse Φ_b in base 2

Subsequent points "fall into biggest holes"

k	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

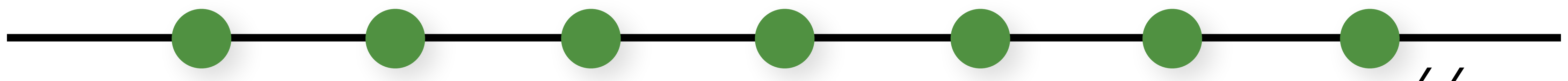


The Van der Corput Sequence

Radical Inverse Φ_b in base 2

Subsequent points “fall into biggest holes”

k	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
...		



Halton and Hammersley Points

Halton: Radical inverse with different base for each dimension:

$$\vec{x}_k = (\Phi_2(k), \Phi_3(k), \Phi_5(k), \ldots, \Phi_{p_n}(k))$$

Halton and Hammersley Points

Halton: Radical inverse with different base for each dimension:

$$\vec{x}_k = (\Phi_2(k), \Phi_3(k), \Phi_5(k), \ \ldots \ , \Phi_{p_n}(k))$$

- The bases should all be relatively prime.

Halton and Hammersley Points

Halton: Radical inverse with different base for each dimension:

$$\vec{x}_k = (\Phi_2(k), \Phi_3(k), \Phi_5(k), \ldots, \Phi_{p_n}(k))$$

- The bases should all be relatively prime.
- Incremental/progressive generation of samples

Halton and Hammersley Points

Halton: Radical inverse with different base for each dimension:

$$\vec{x}_k = (\Phi_2(k), \Phi_3(k), \Phi_5(k), \dots, \Phi_{p_n}(k))$$

- The bases should all be relatively prime.
- Incremental/progressive generation of samples

Hammersley: Same as Halton, but first dimension is k/N:

$$\vec{x}_k = ({}^k/_N, \Phi_2(k), \Phi_3(k), \Phi_5(k), \dots, \Phi_{p_n}(k))$$

Halton and Hammersley Points

Halton: Radical inverse with different base for each dimension:

$$\vec{x}_k = (\Phi_2(k), \Phi_3(k), \Phi_5(k), \ldots, \Phi_{p_n}(k))$$

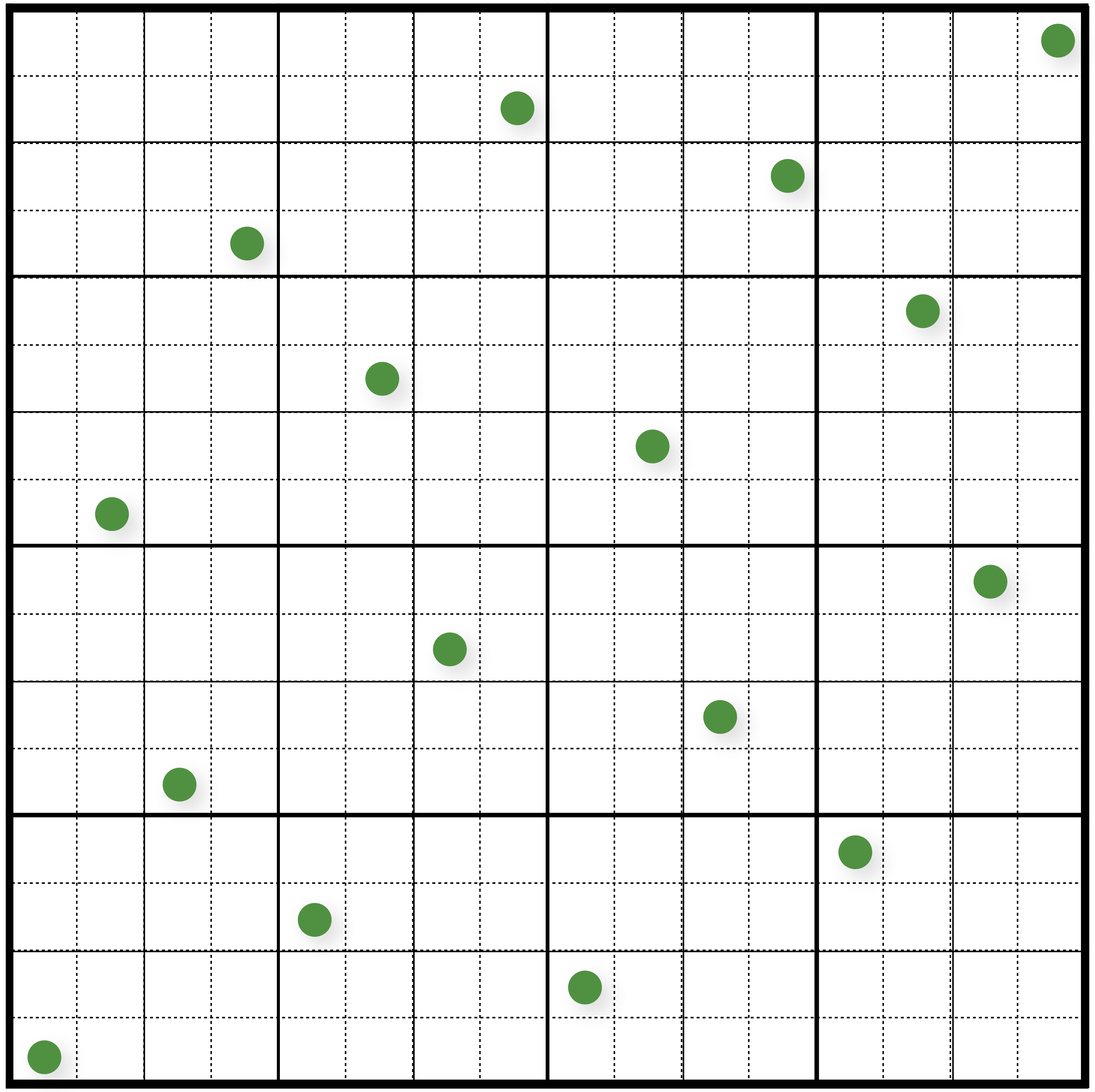
- The bases should all be relatively prime.
- Incremental/progressive generation of samples

Hammersley: Same as Halton, but first dimension is k/N:

$$\vec{x}_k = ({}^k/_N, \Phi_2(k), \Phi_3(k), \Phi_5(k), \ldots, \Phi_{p_n}(k))$$

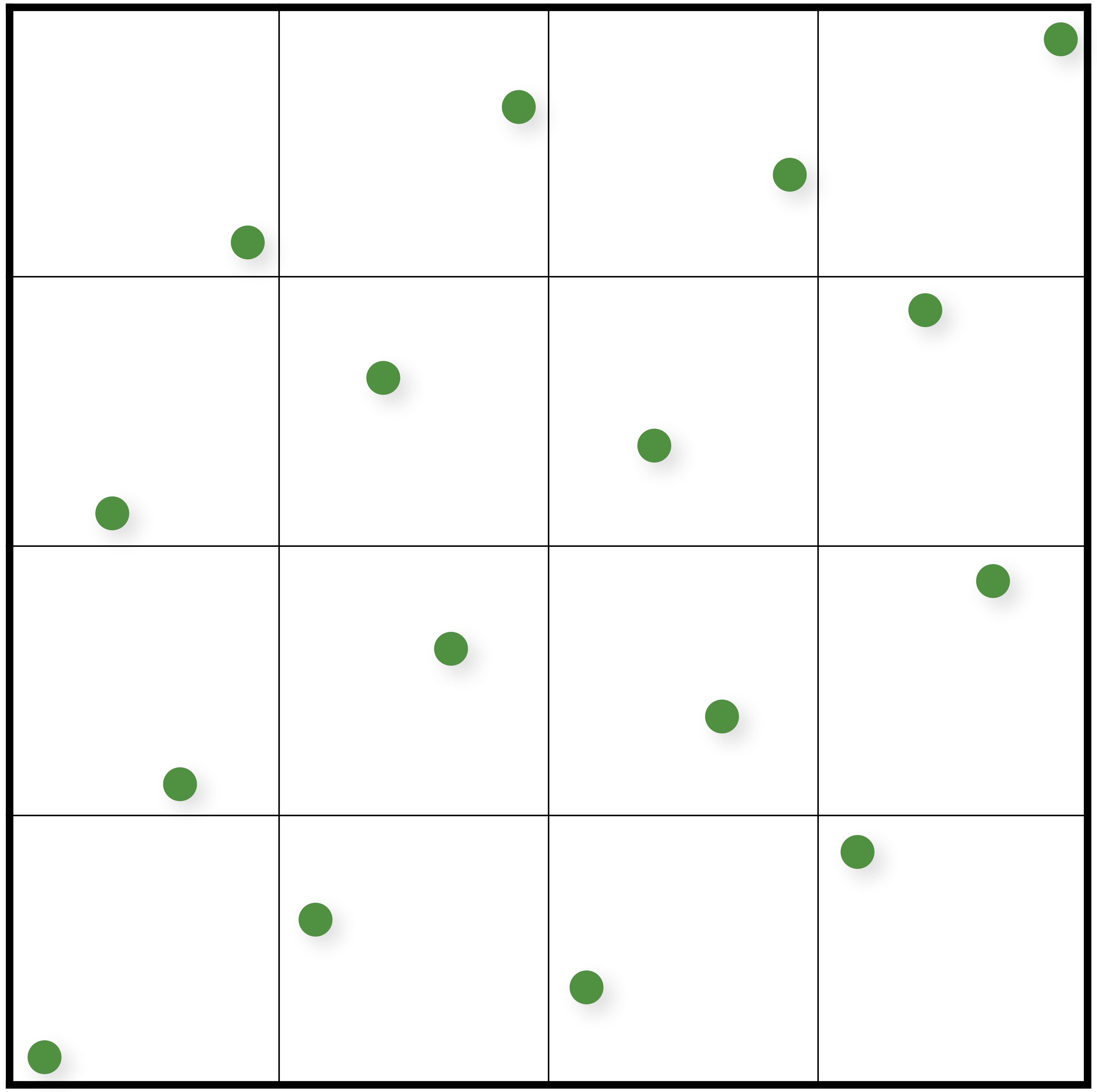
- Not incremental, need to know sample count, N, in advance

The Hammersley Sequence



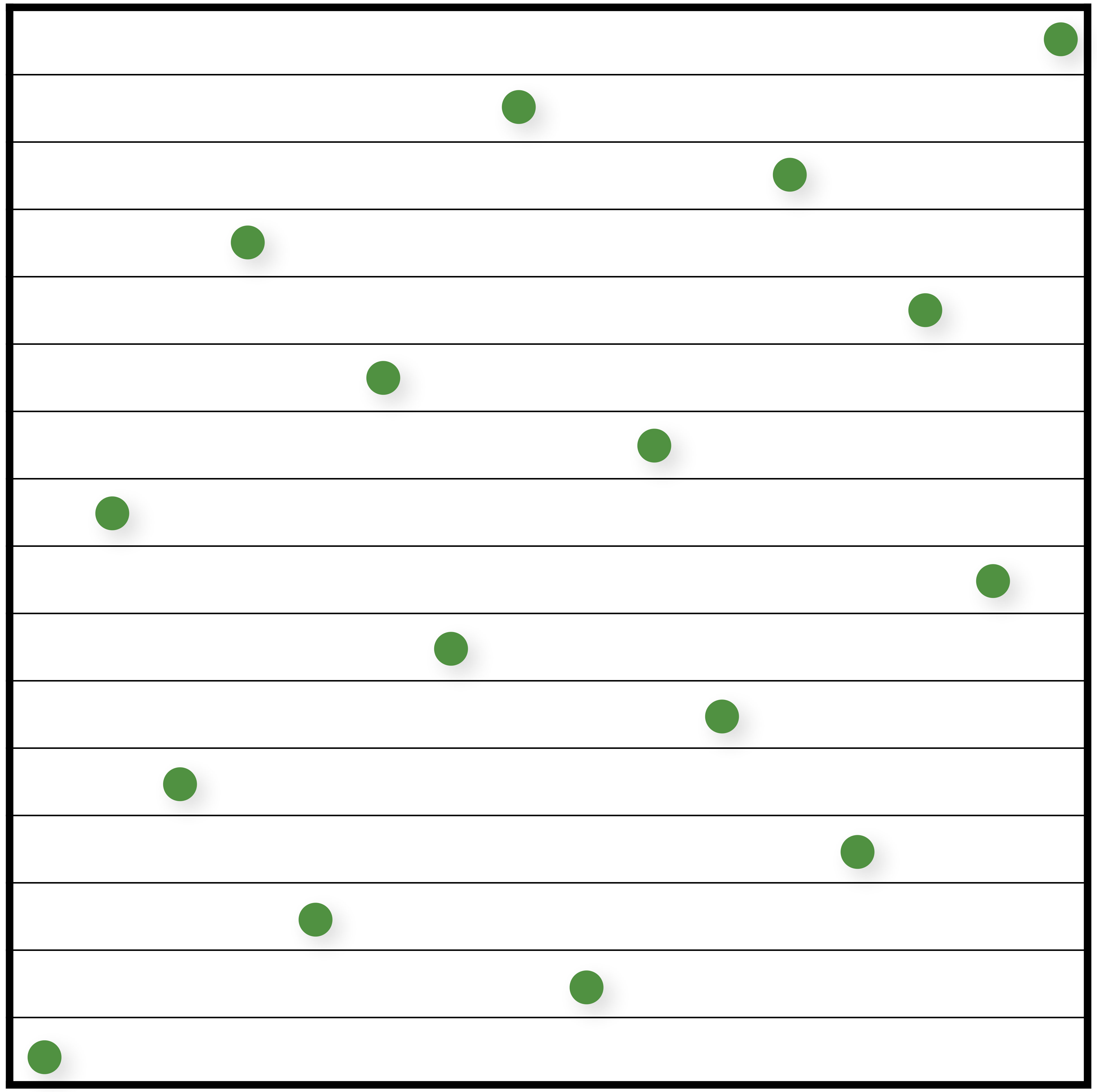
1 sample in each “elementary interval”

The Hammersley Sequence



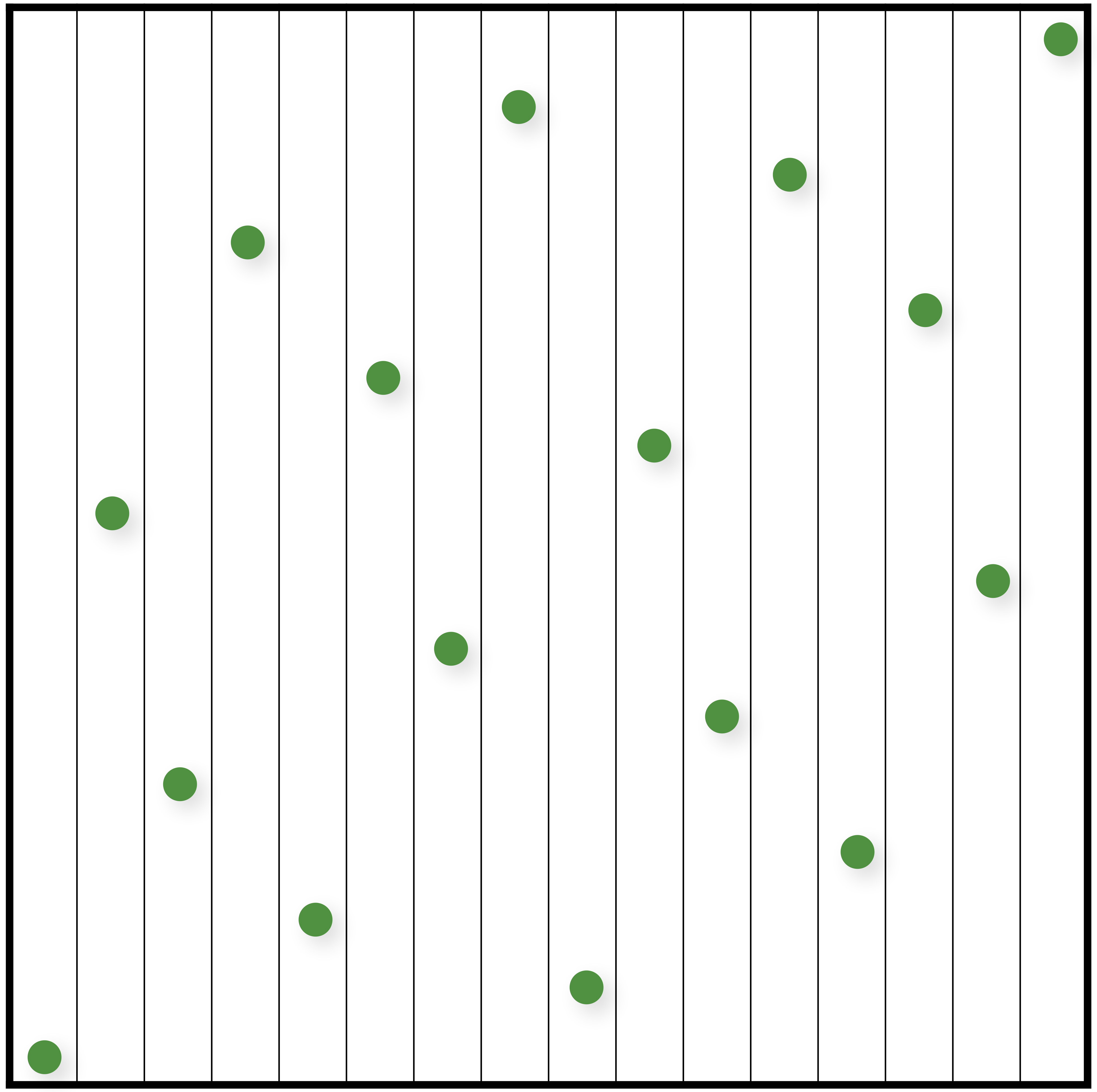
1 sample in each "elementary interval"

The Hammersley Sequence



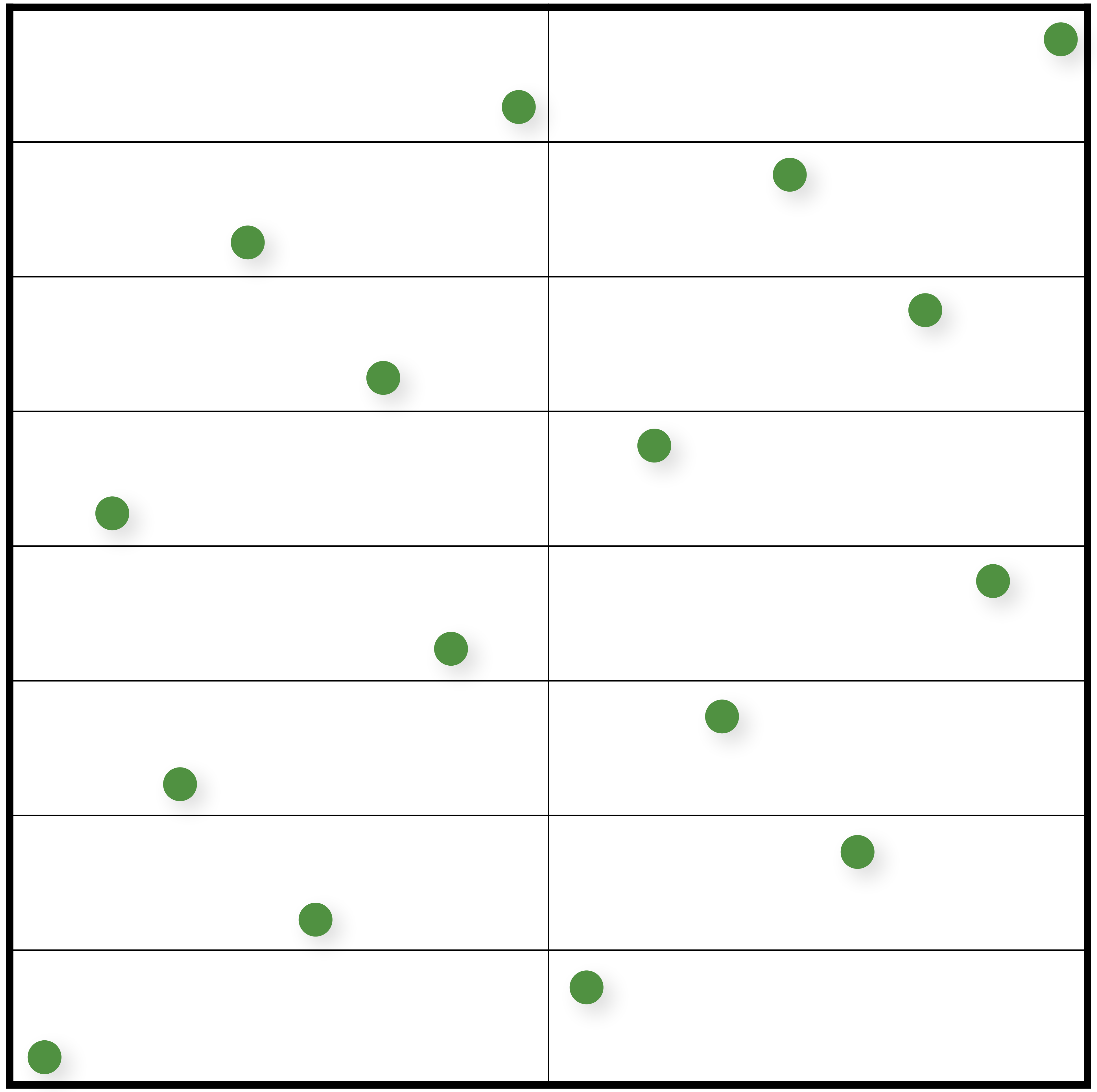
1 sample in each “elementary interval”

The Hammersley Sequence



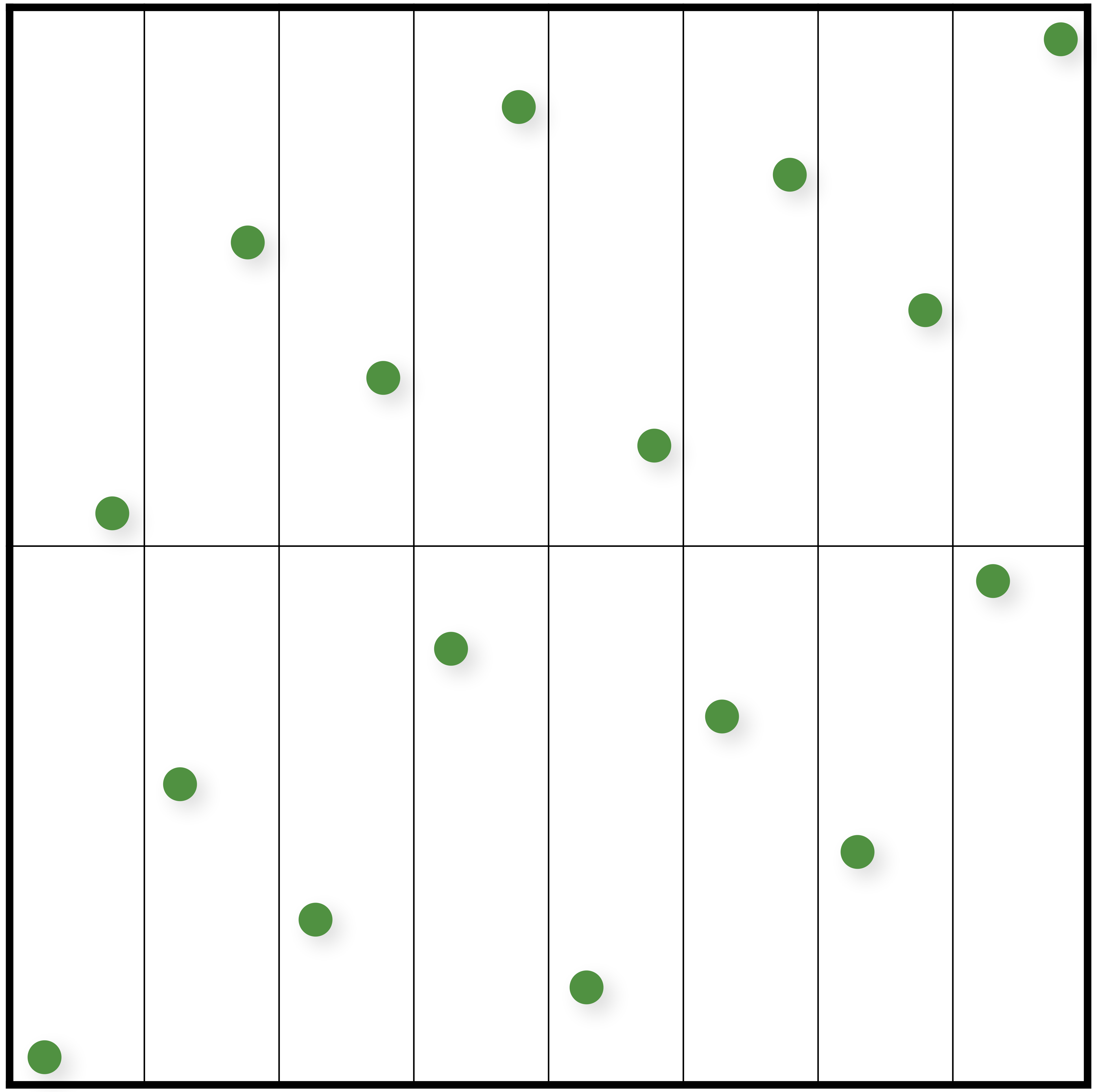
1 sample in each “elementary interval”

The Hammersley Sequence



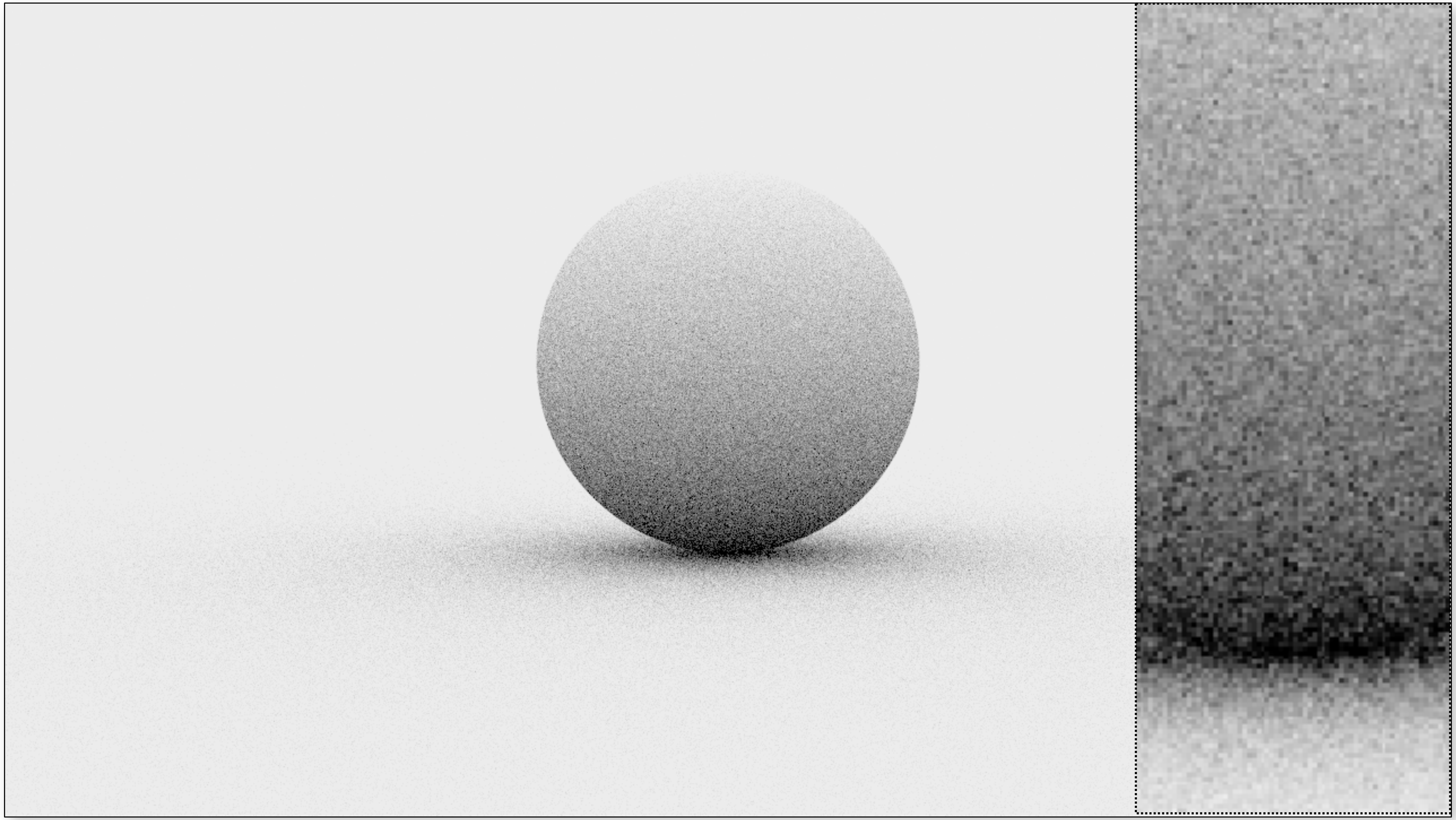
1 sample in each “elementary interval”

The Hammersley Sequence

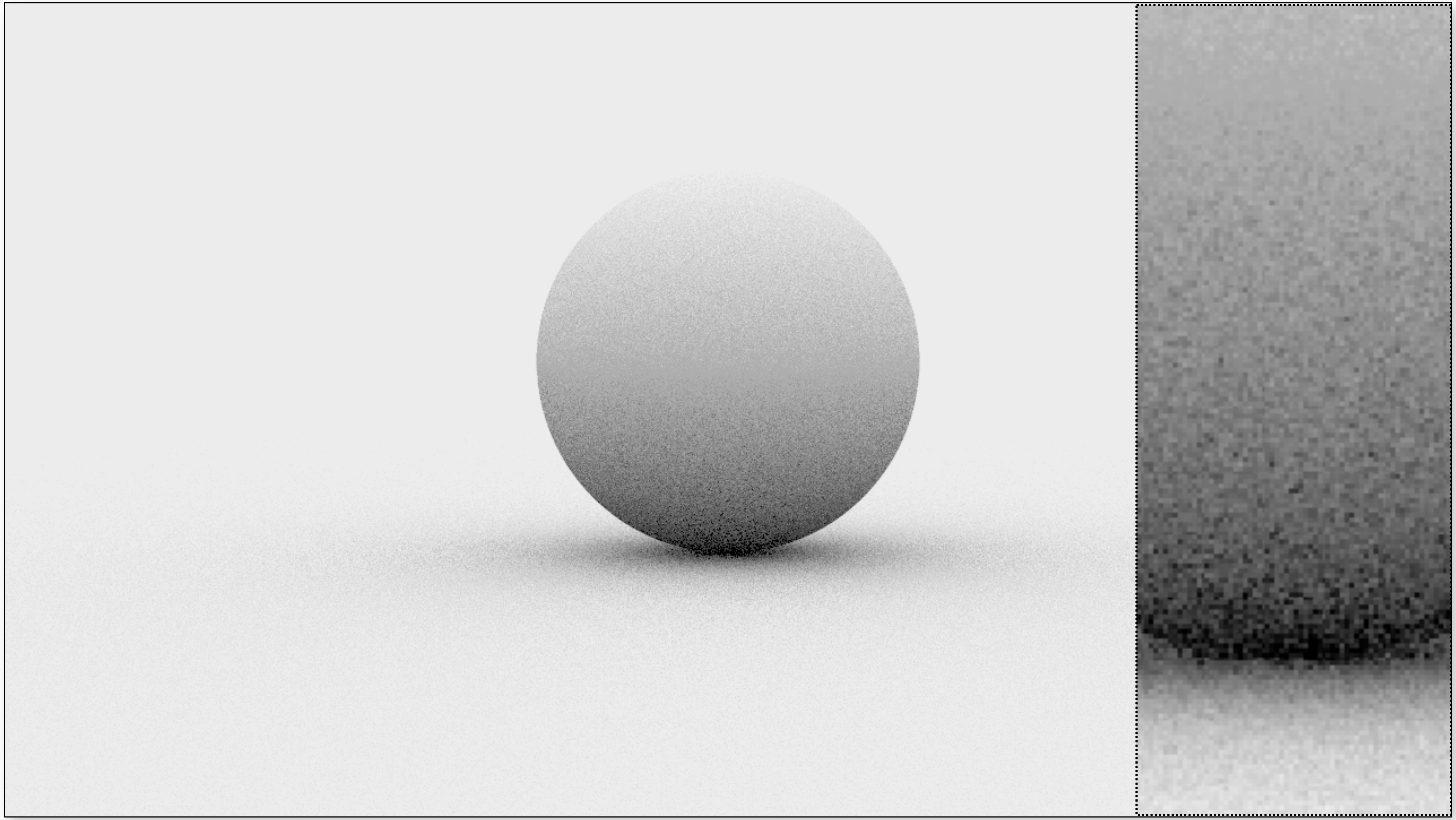


1 sample in each "elementary interval"

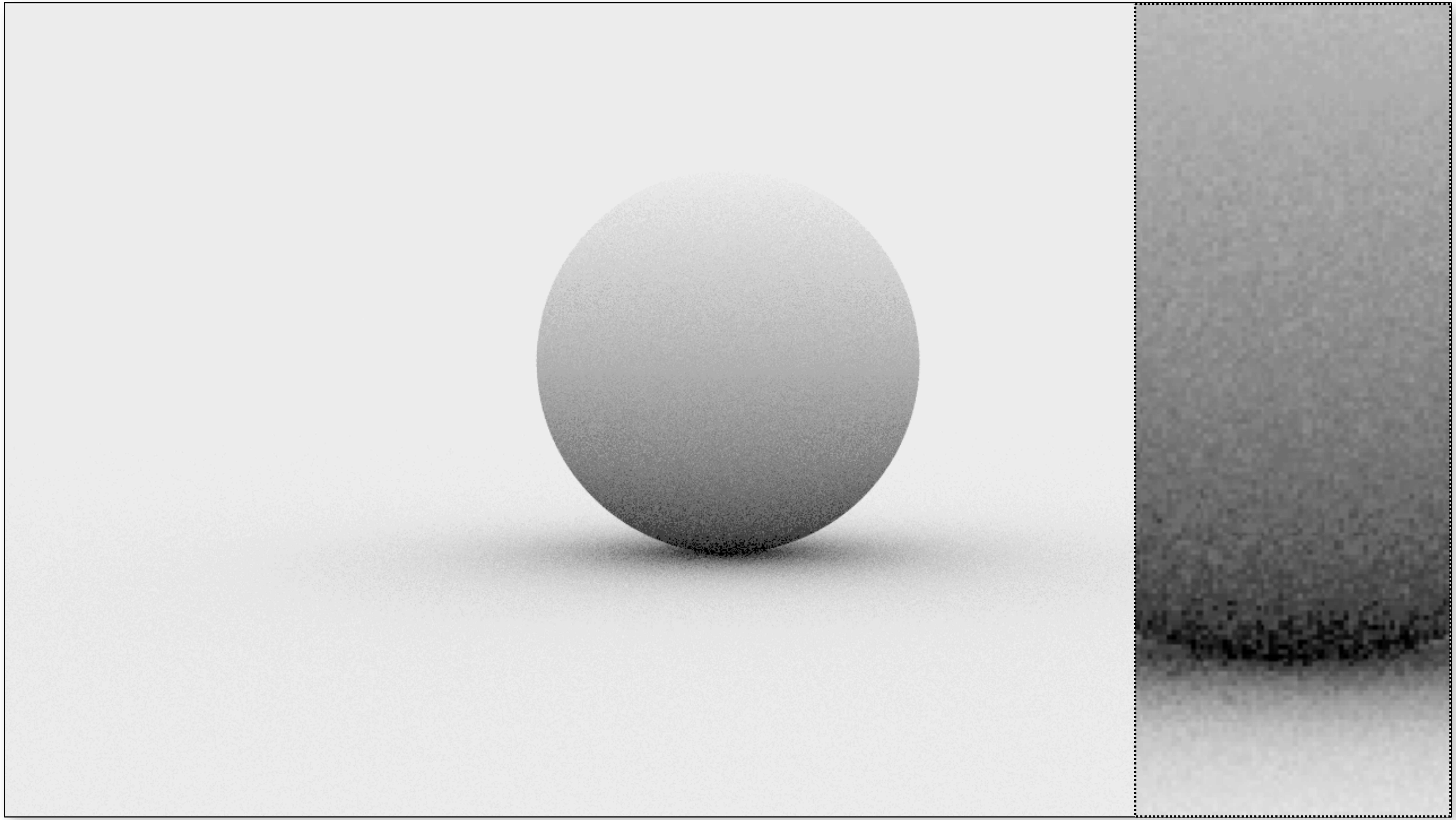
Monte Carlo (16 random samples)



Monte Carlo (16 jittered samples)



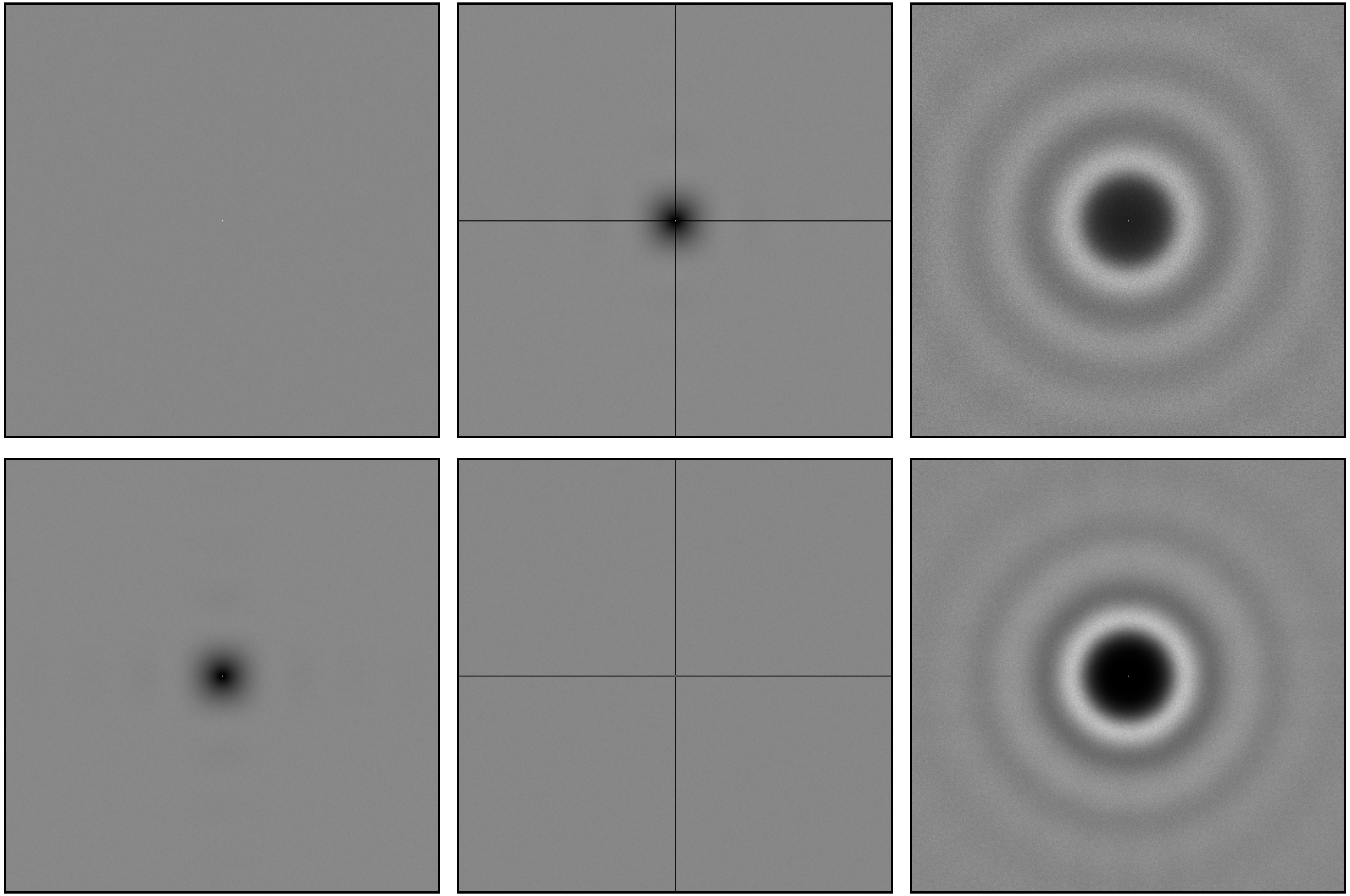
Scrambled Low-Discrepancy Sampling



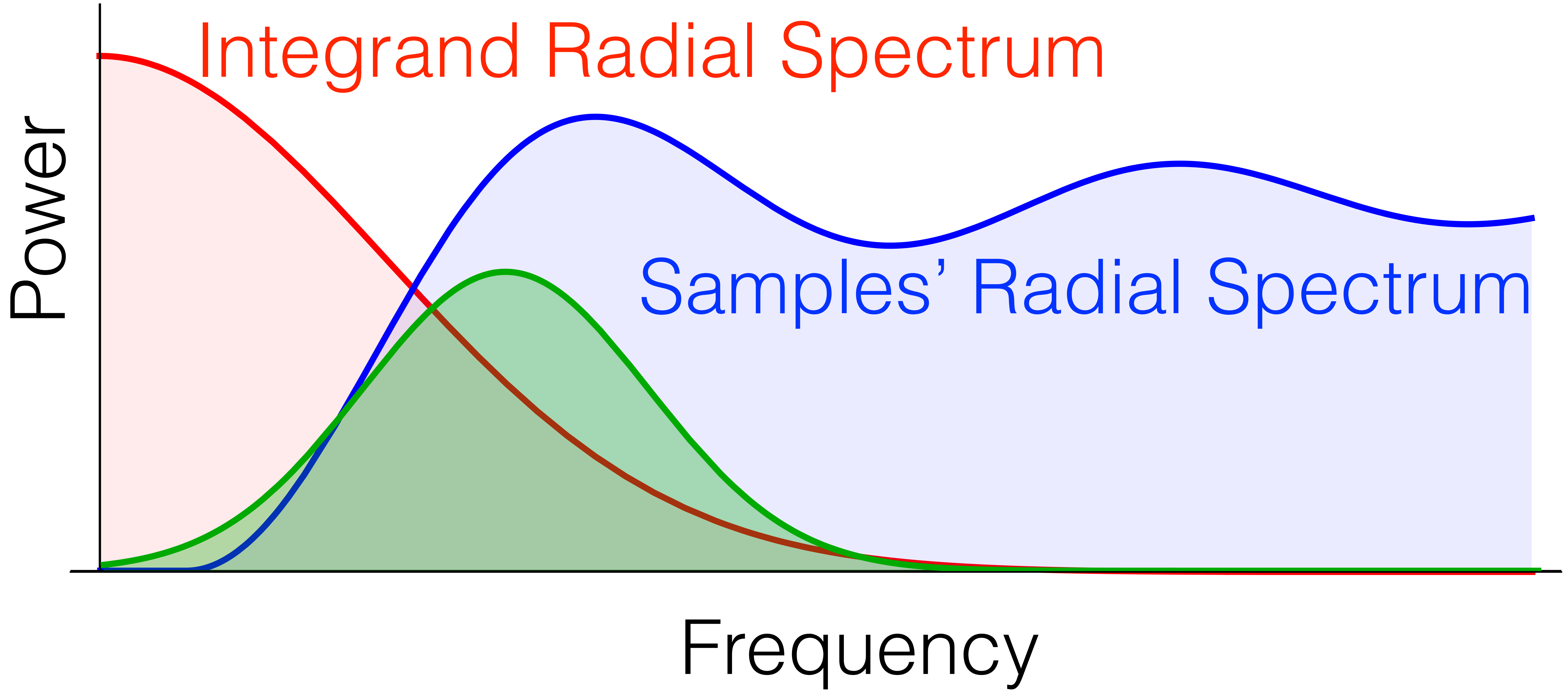
More info on QMC in Rendering

S. Premoze, A. Keller, and M. Raab.
Advanced (Quasi-) Monte Carlo Methods for Image Synthesis.
In SIGGRAPH 2012 courses.

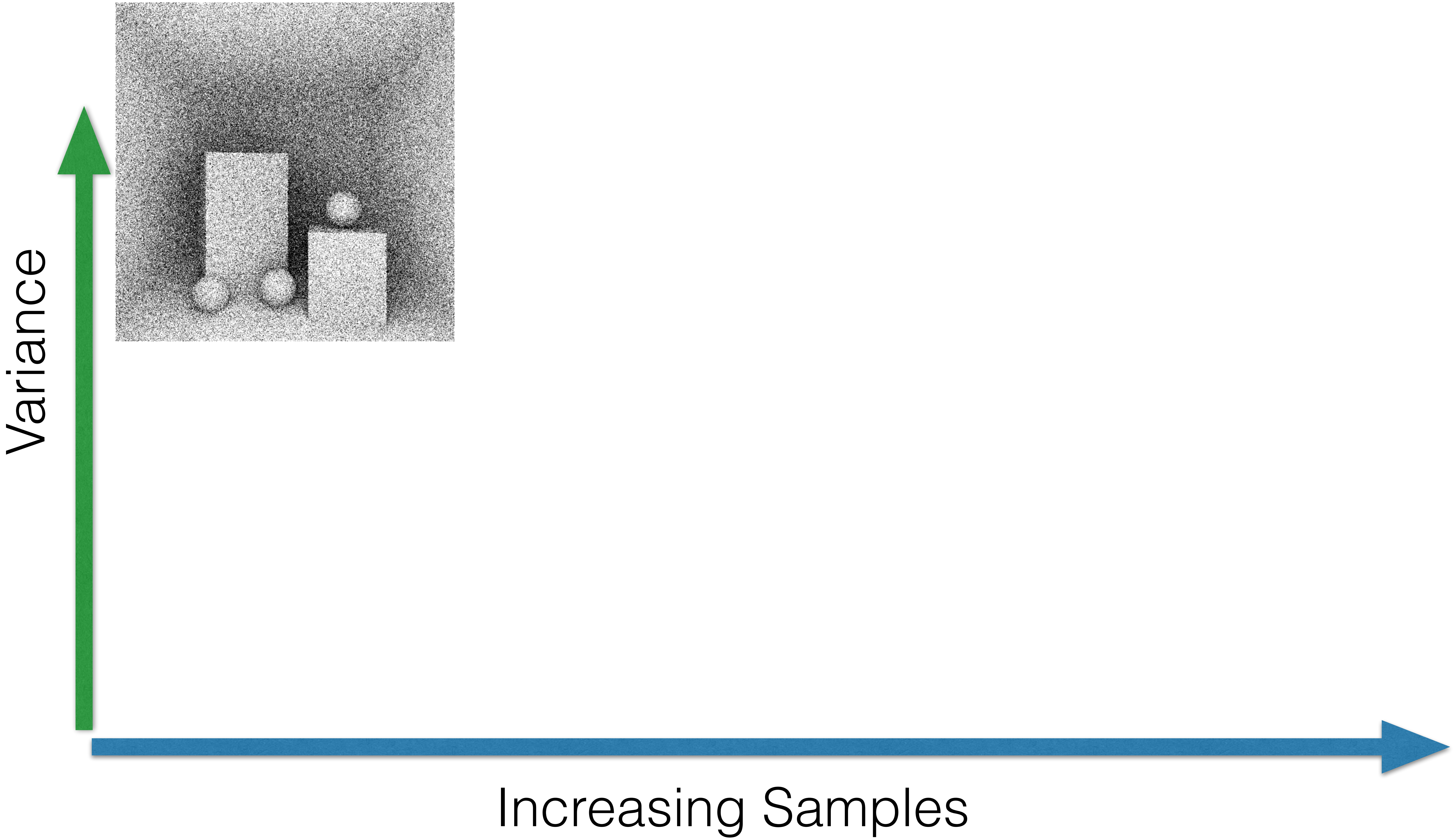
How can we predict error from these?



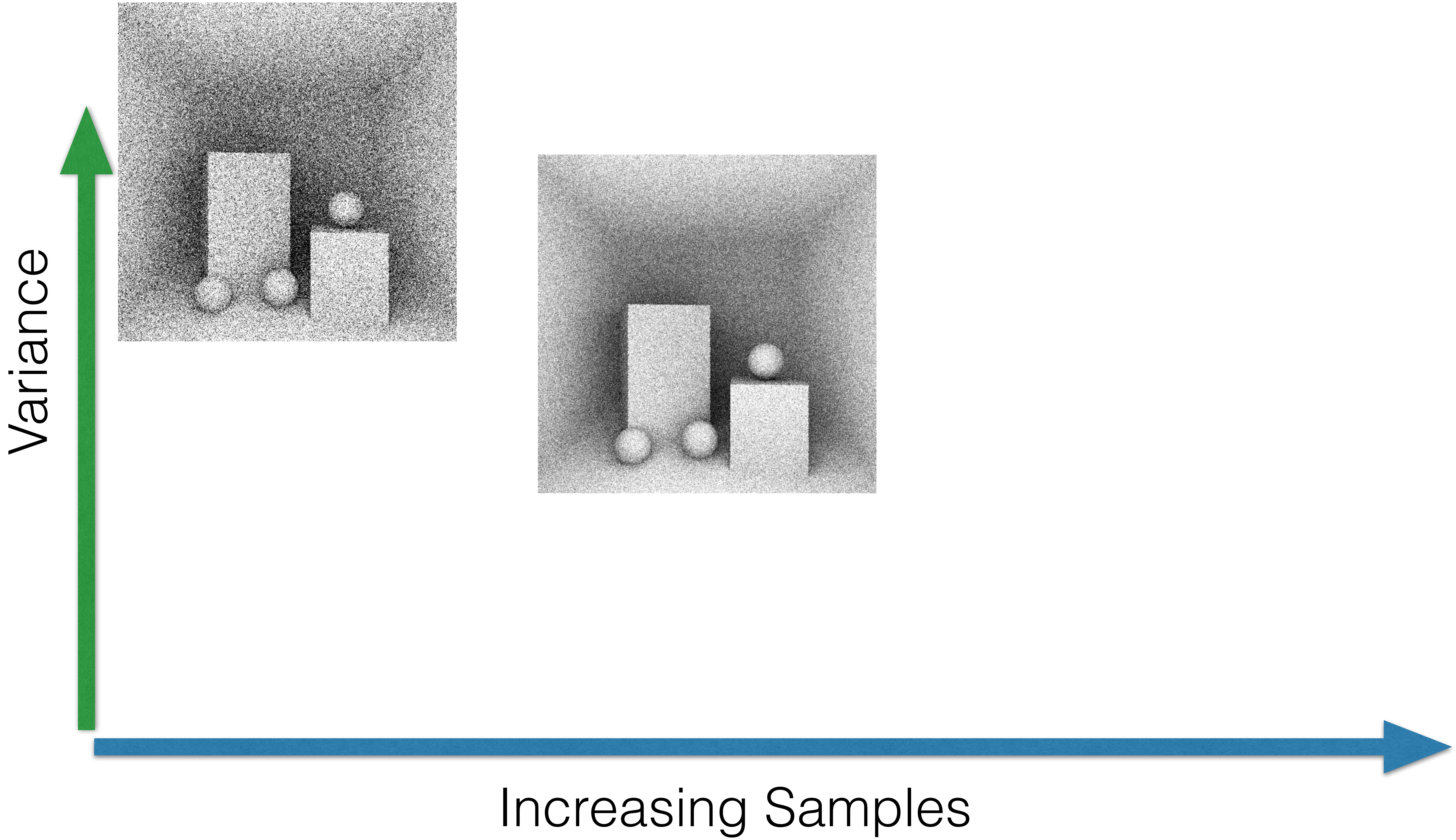
Part 2: Formal Treatment of MSE, Bias and Variance



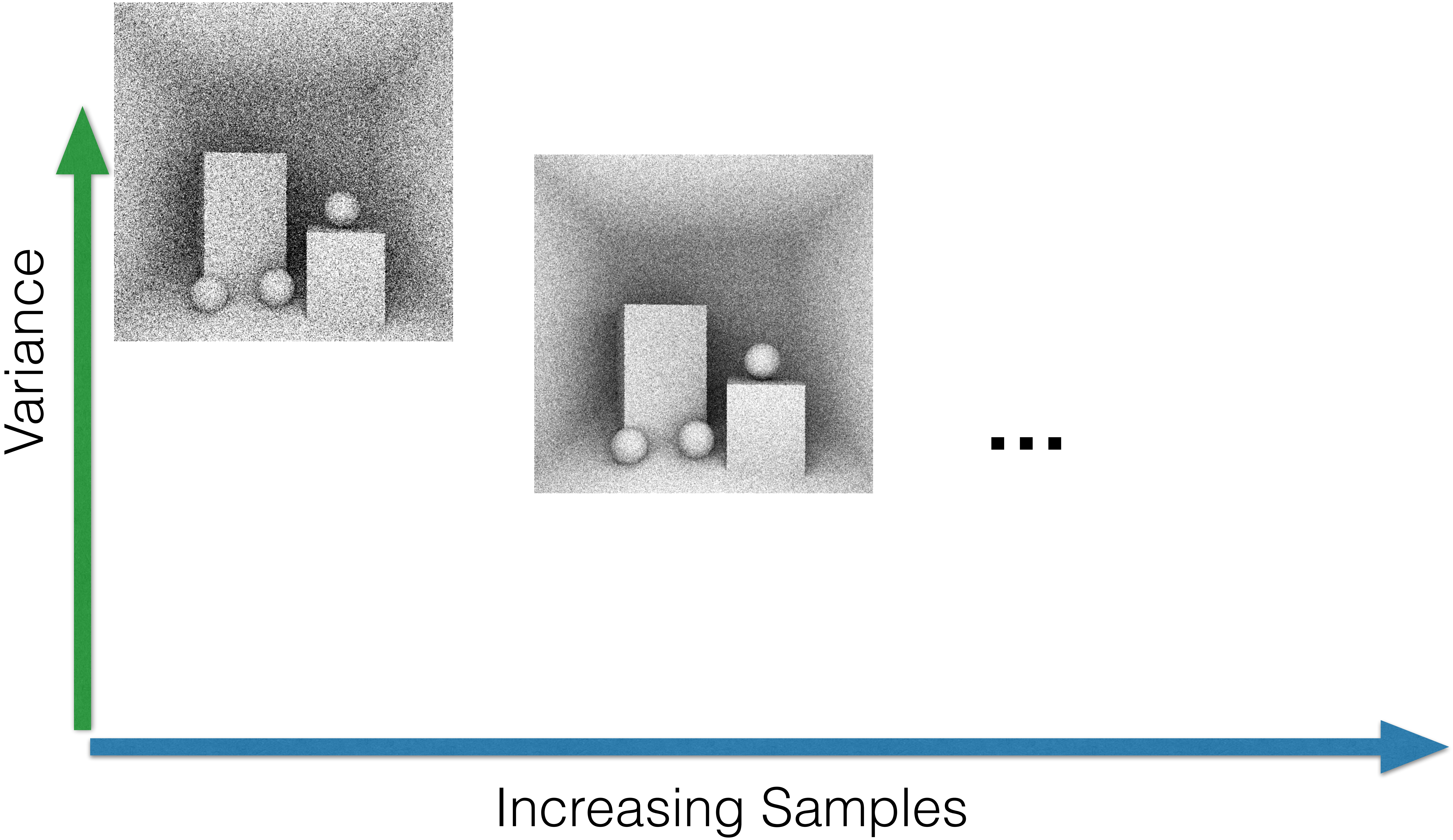
Convergence rate for Random Samples



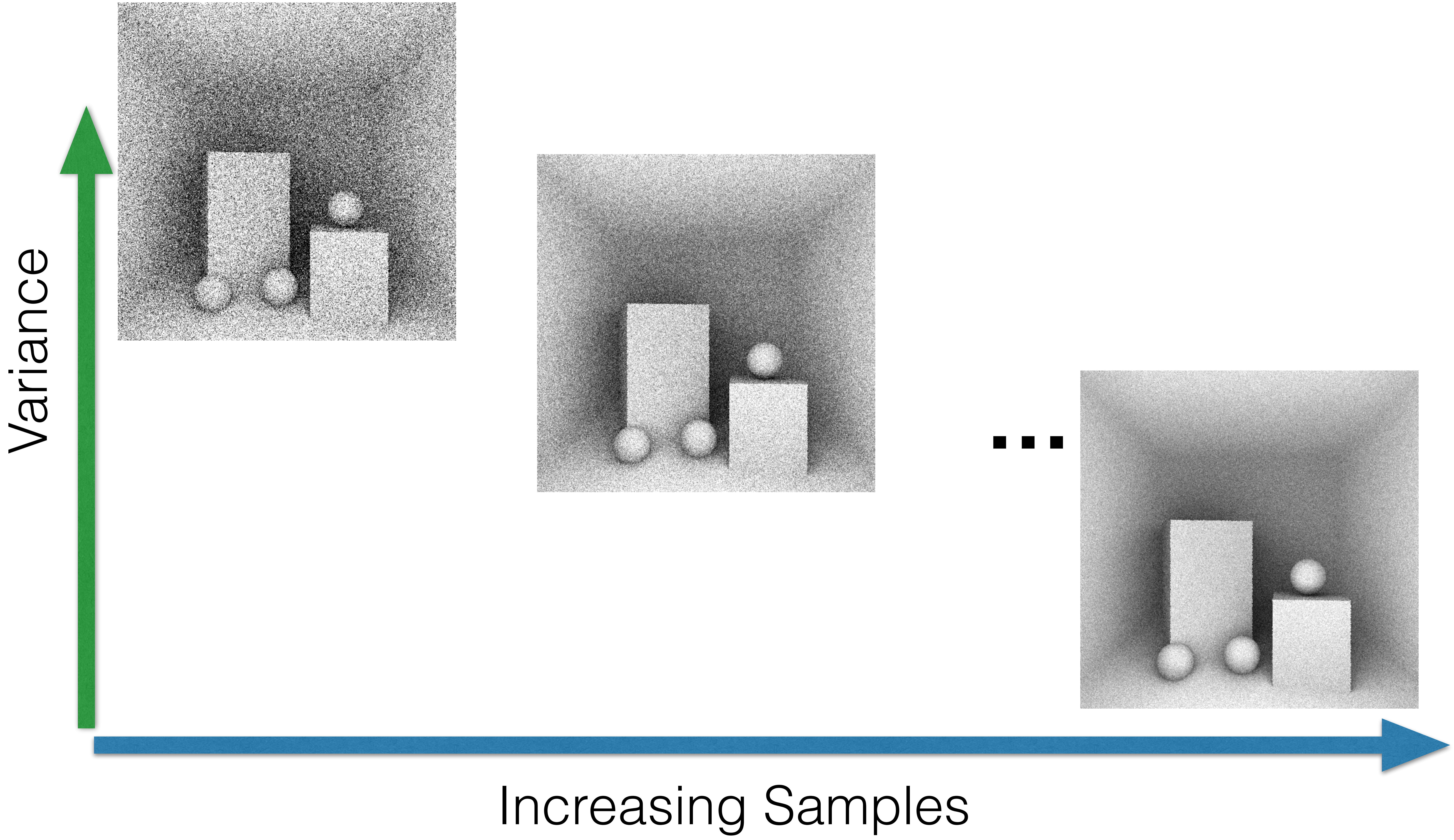
Convergence rate for Random Samples



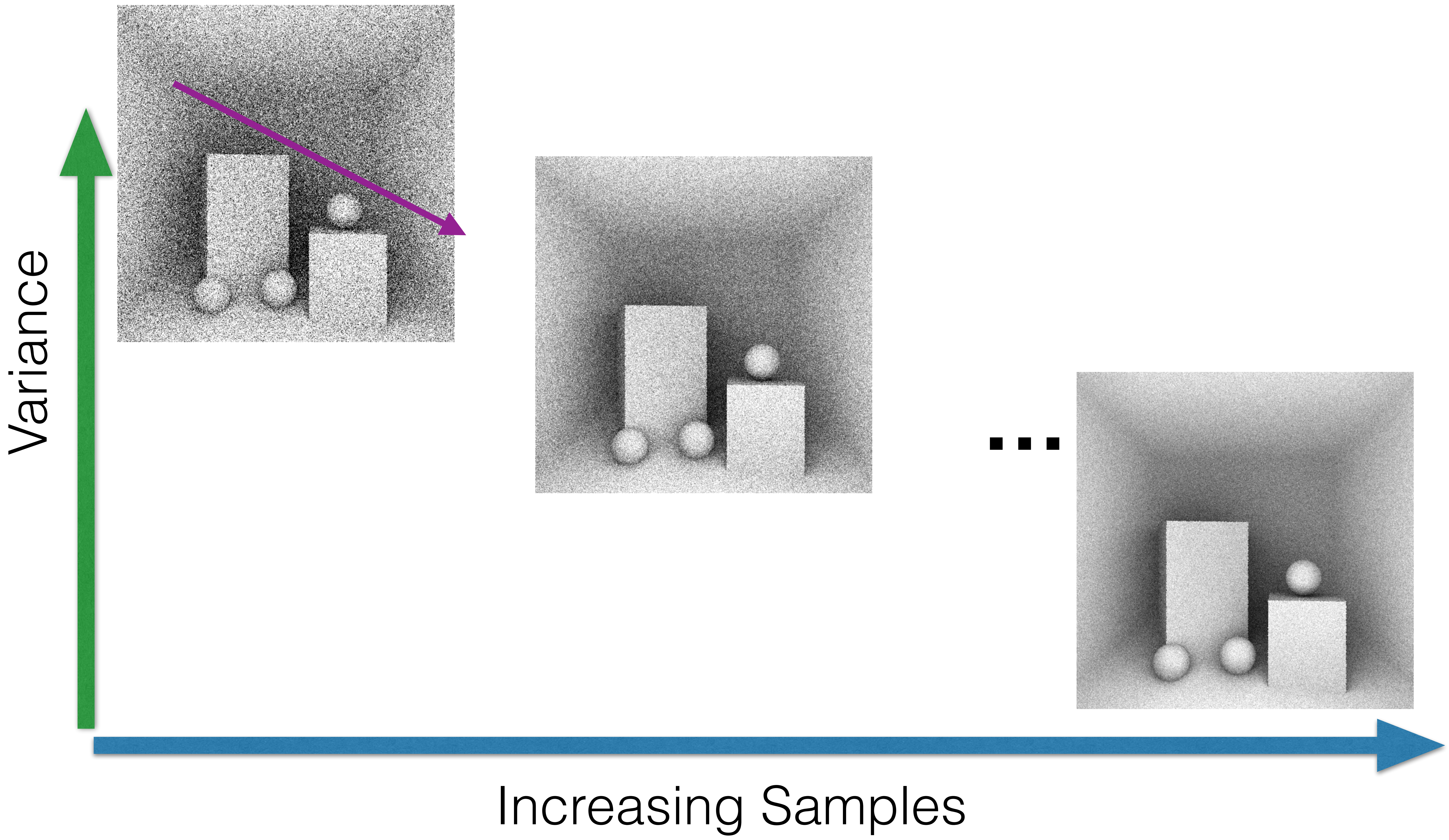
Convergence rate for Random Samples



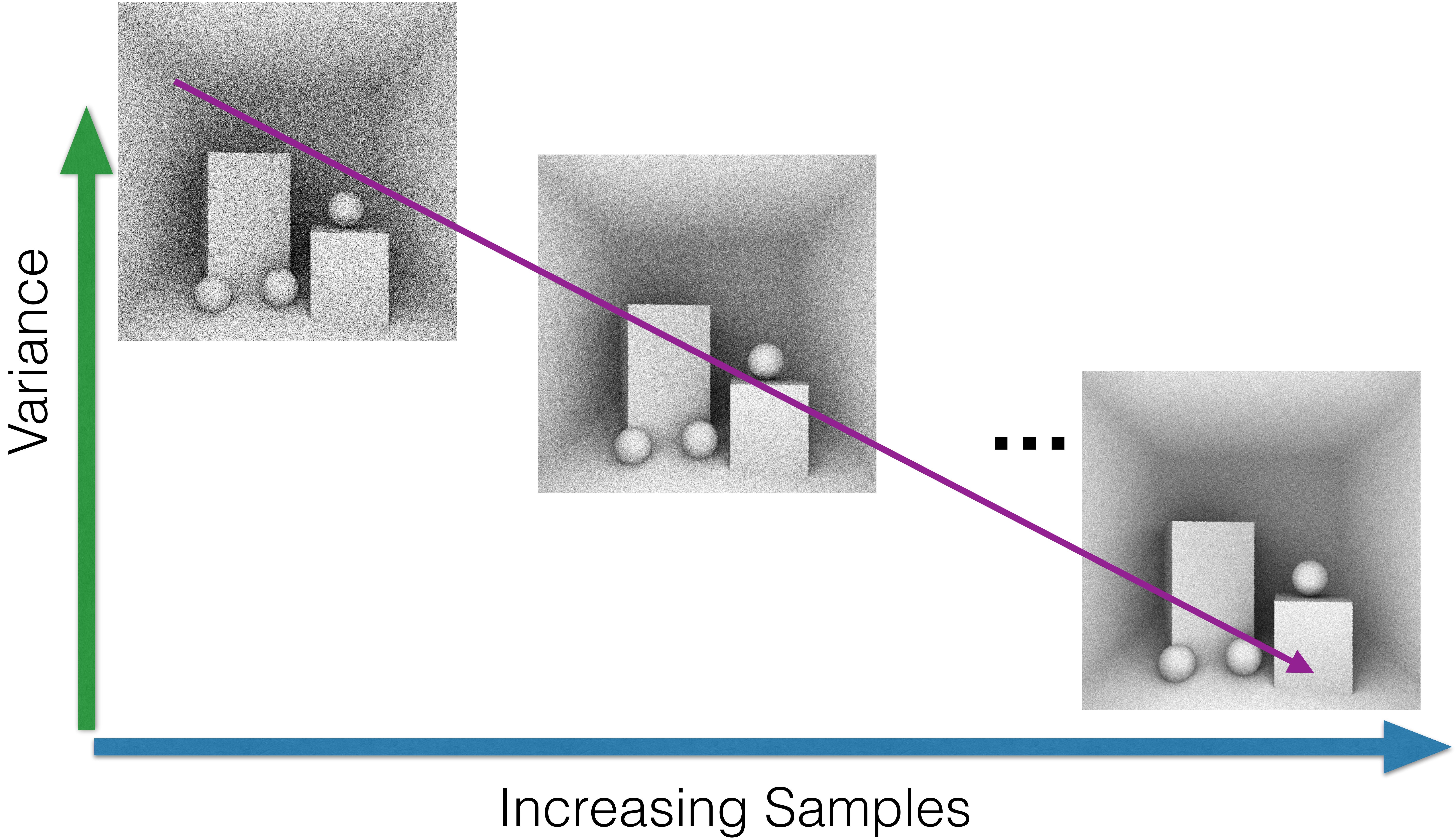
Convergence rate for Random Samples



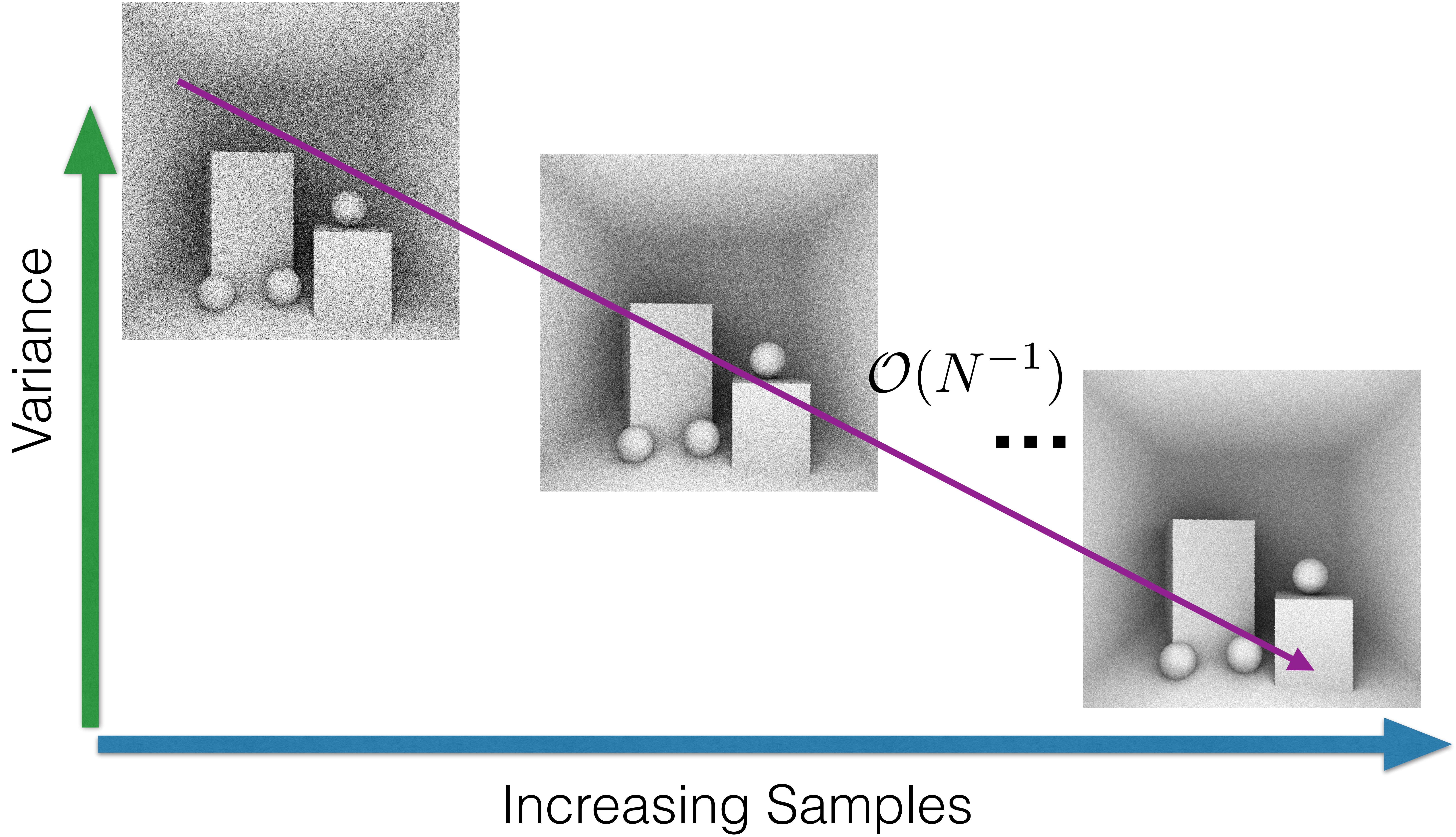
Convergence rate for Random Samples



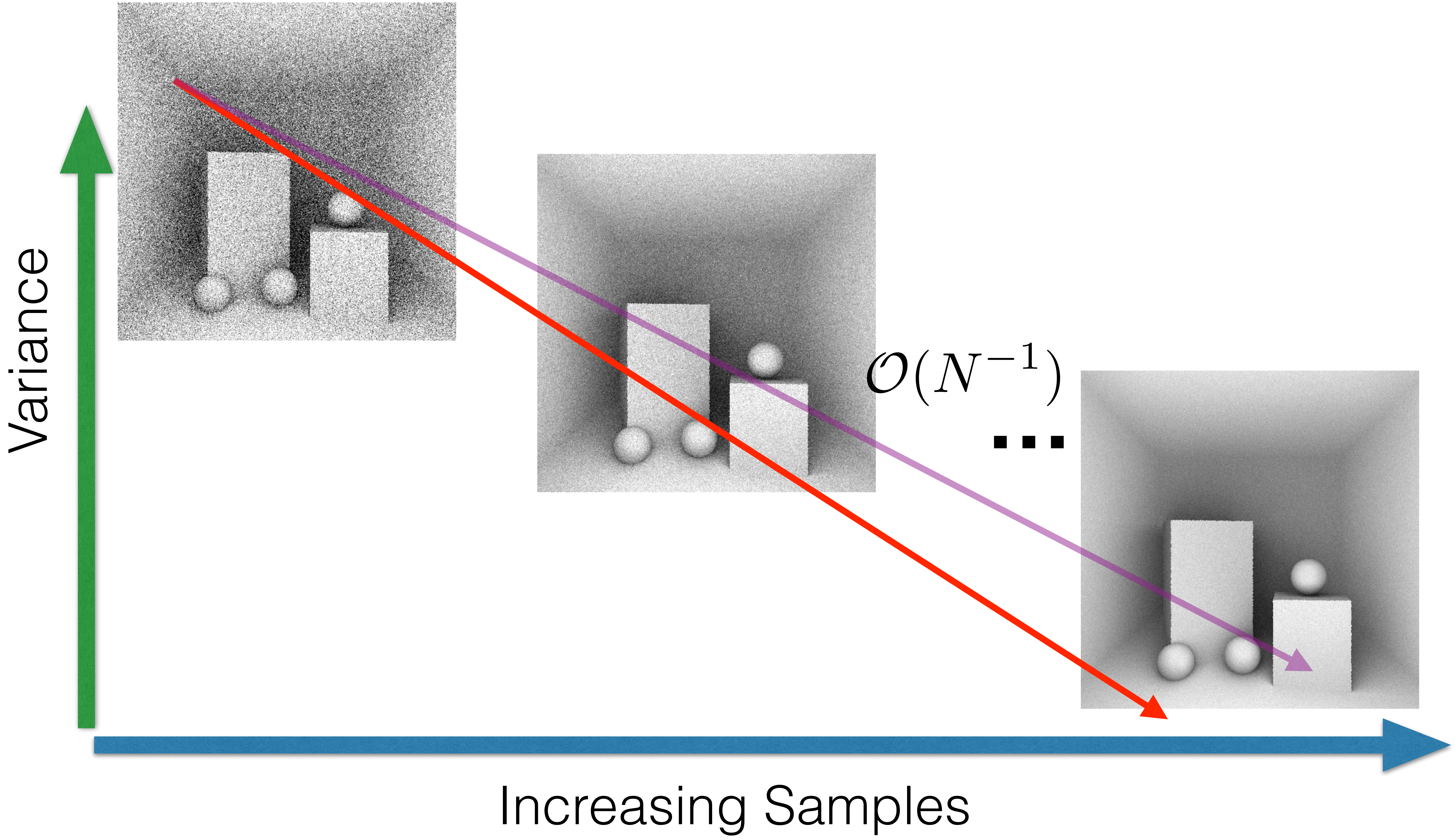
Convergence rate for Random Samples



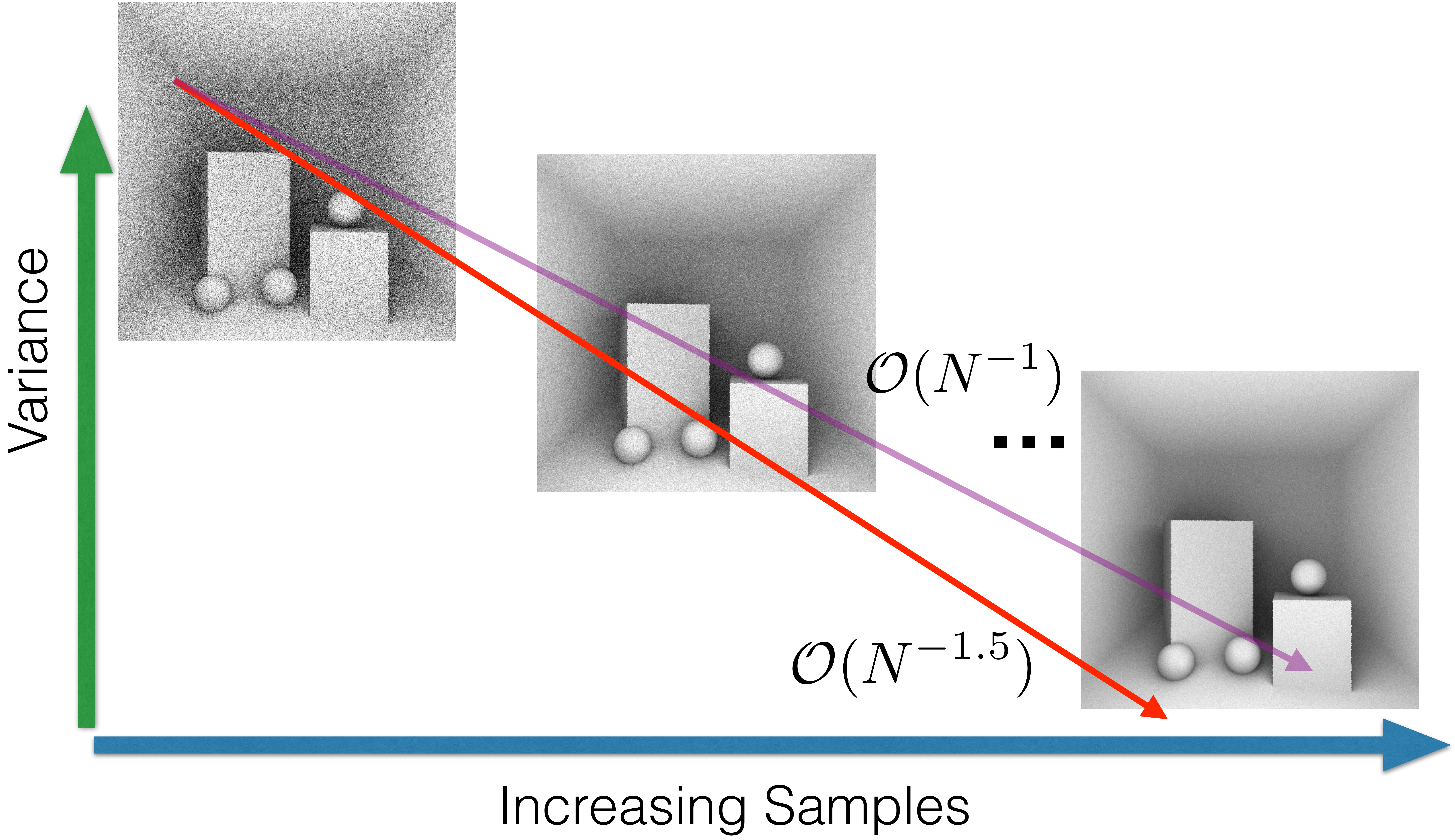
Convergence rate for Random Samples



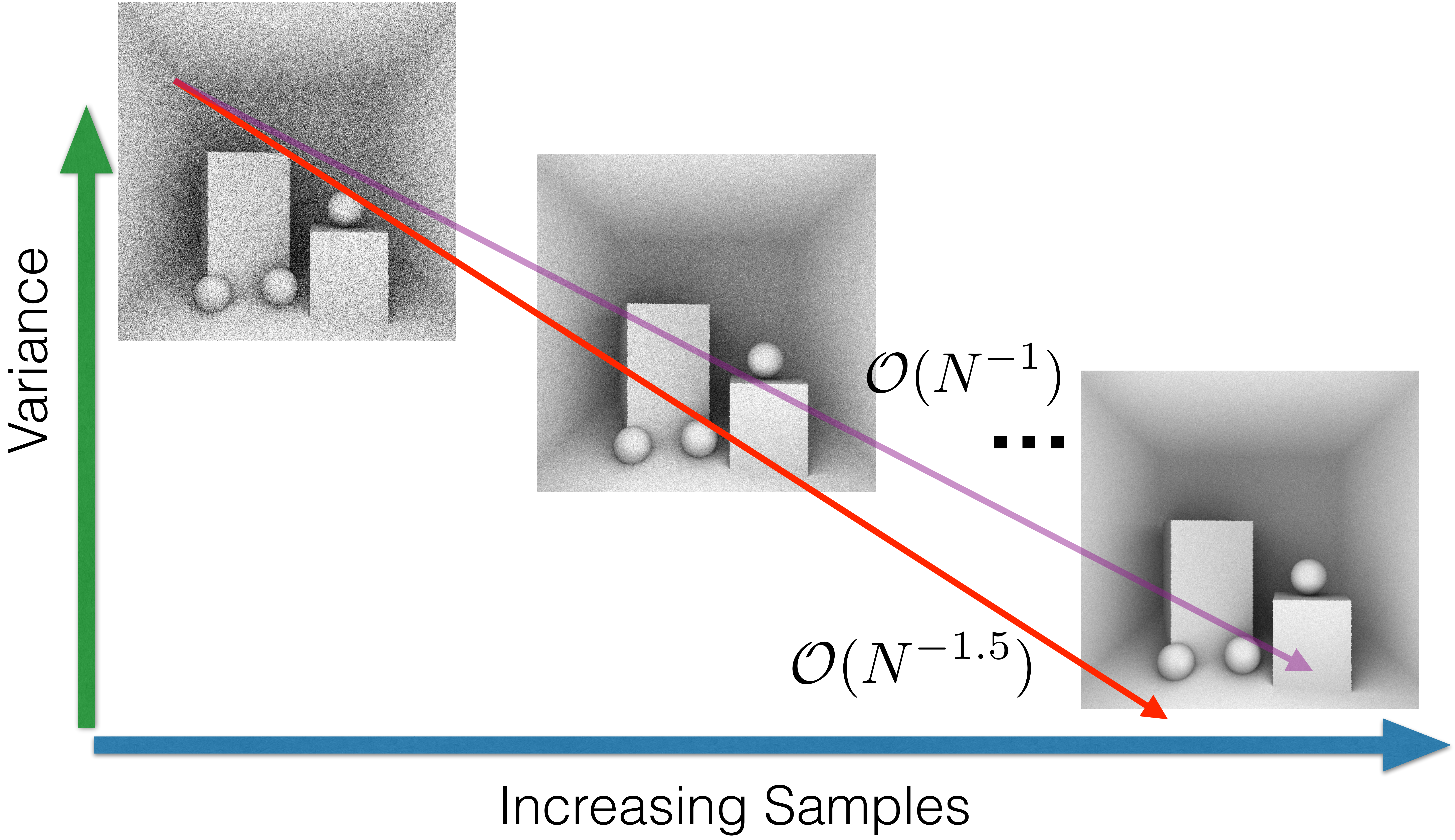
Convergence rate for Jittered Samples



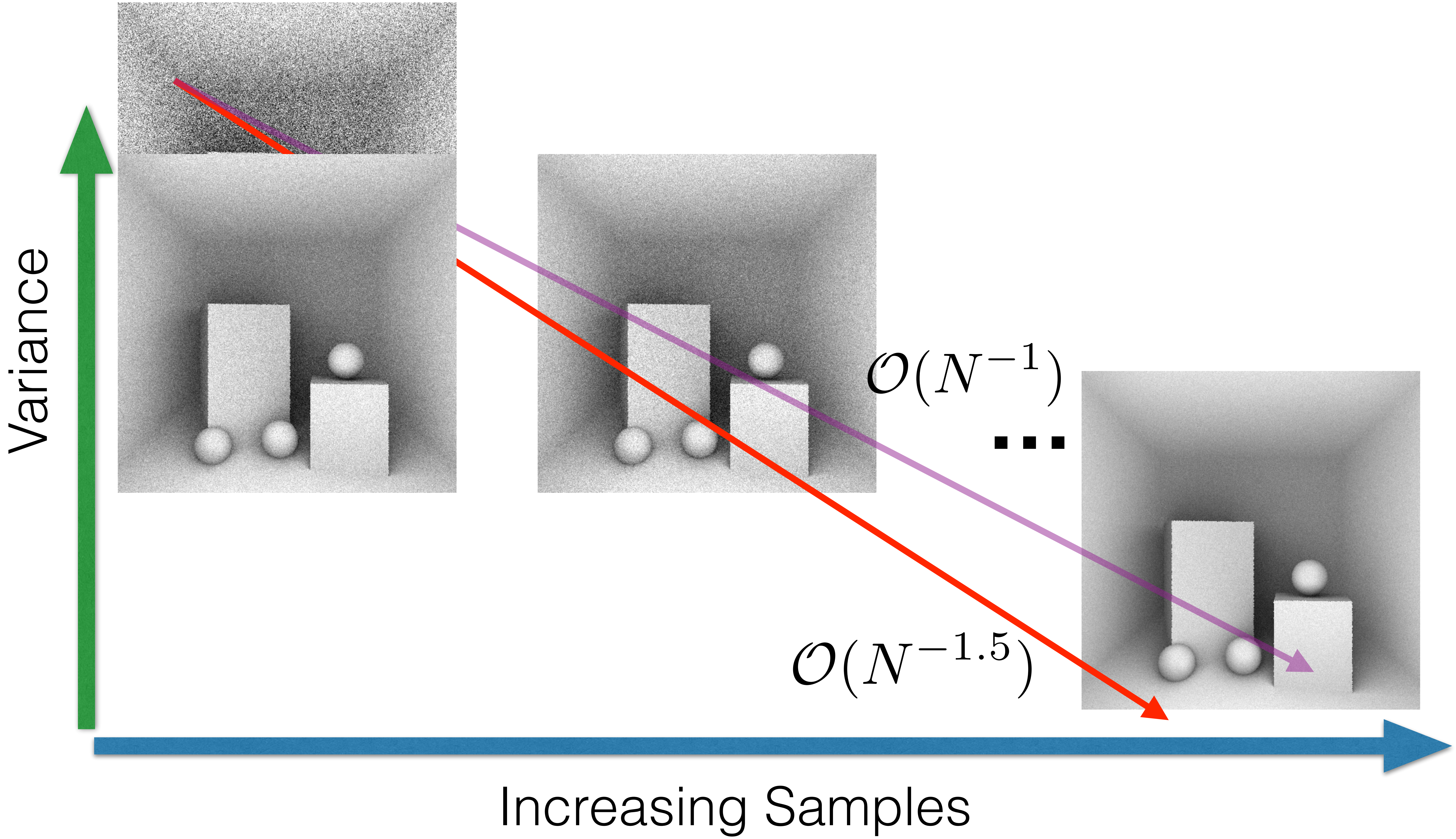
Convergence rate for Jittered Samples



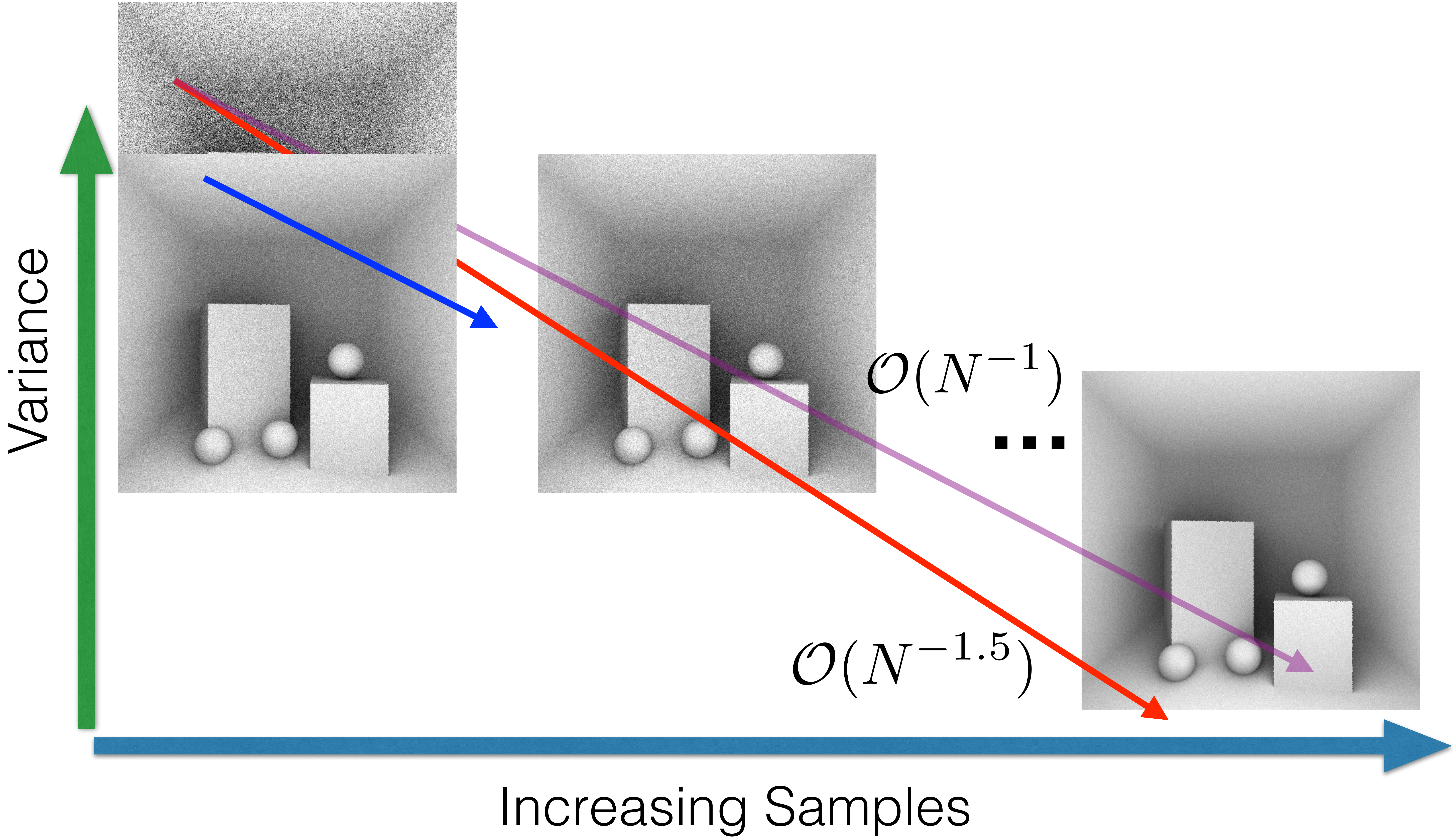
Convergence rate
Jittered vs Poisson Disk



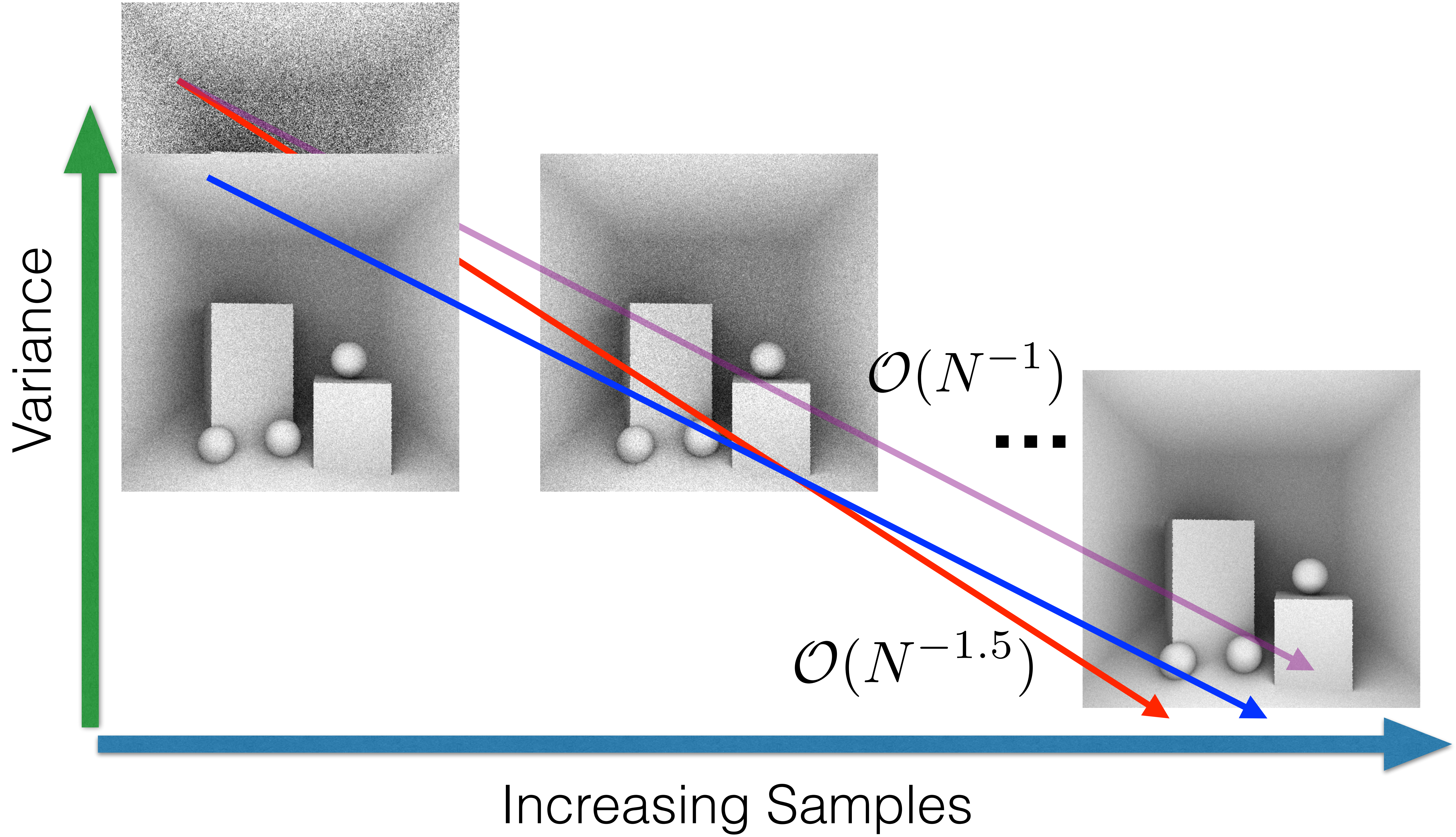
Convergence rate
Jittered vs Poisson Disk



Convergence rate
Jittered vs Poisson Disk

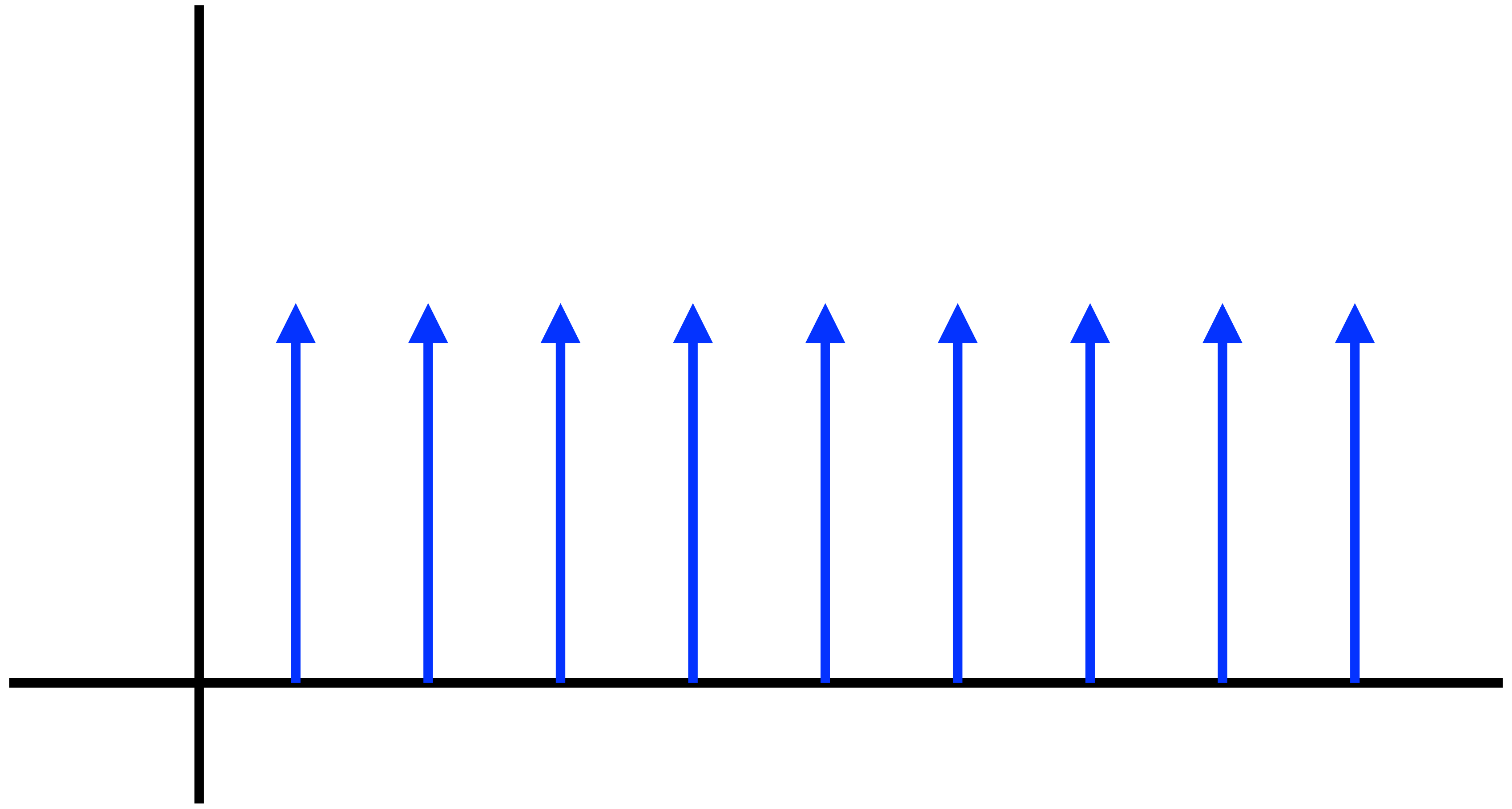


Convergence rate
Jittered vs Poisson Disk



Samples and function in Fourier Domain

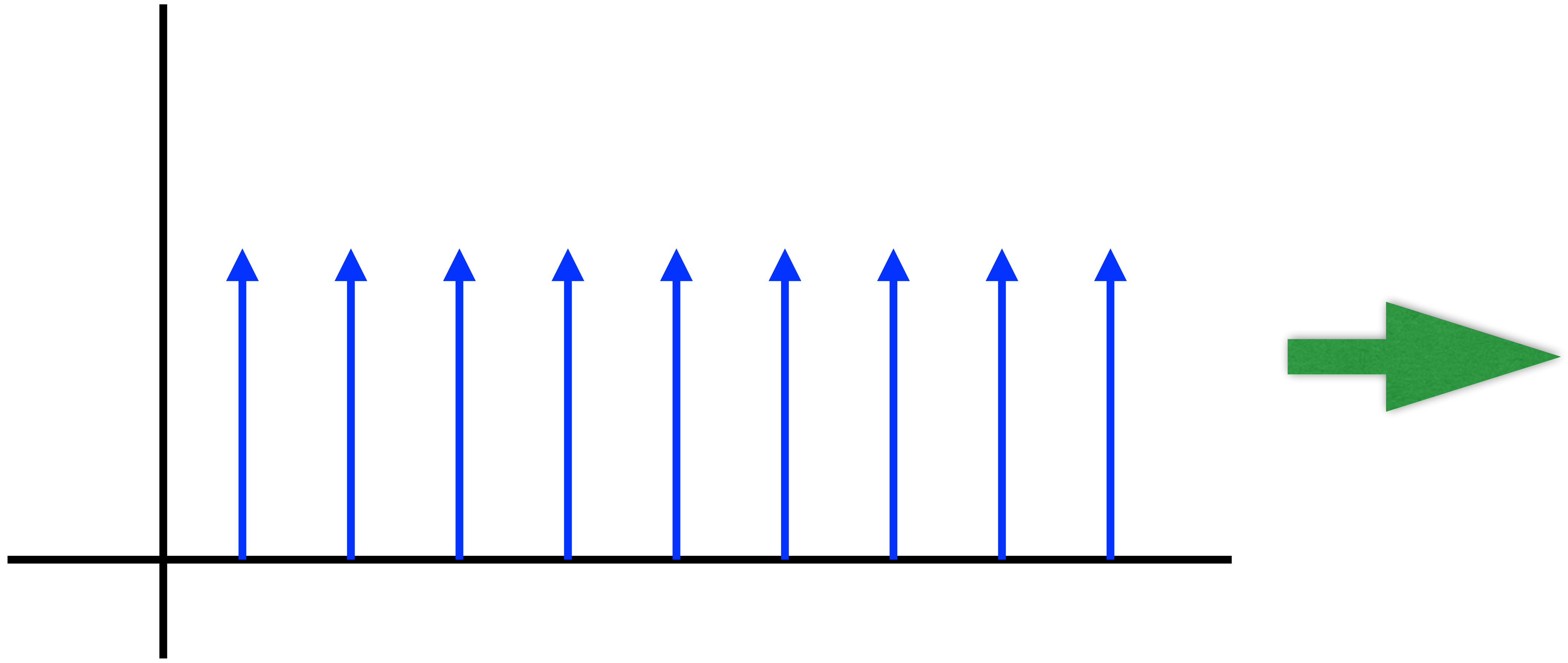
Spatial Domain



Fourier Domain

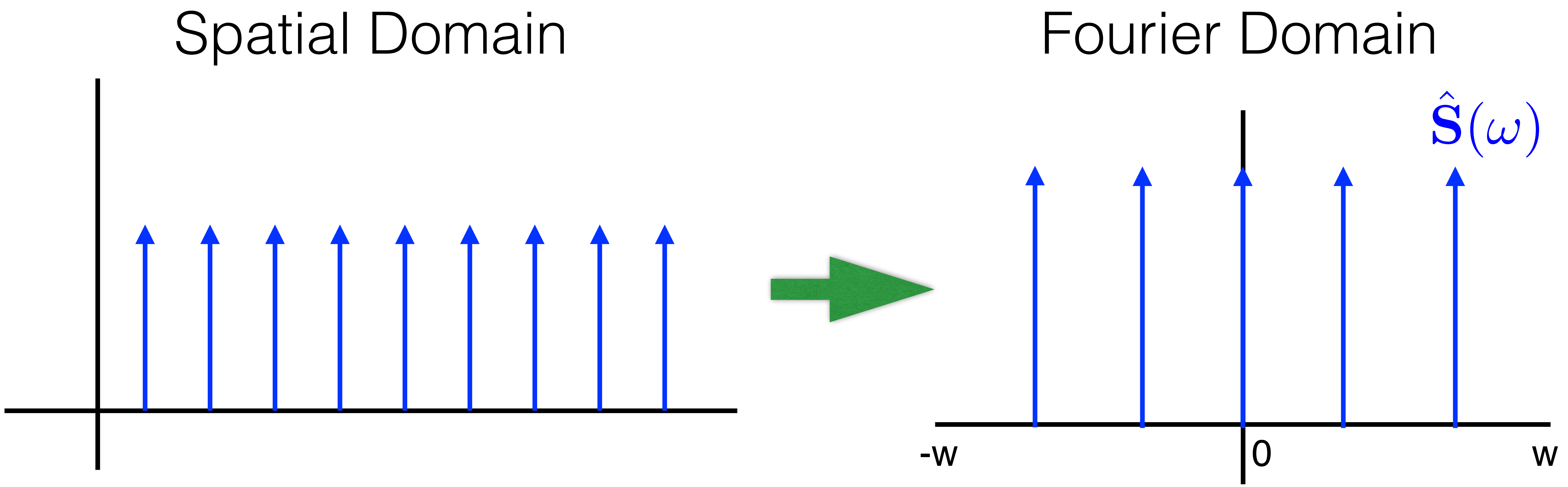
Samples and function in Fourier Domain

Spatial Domain

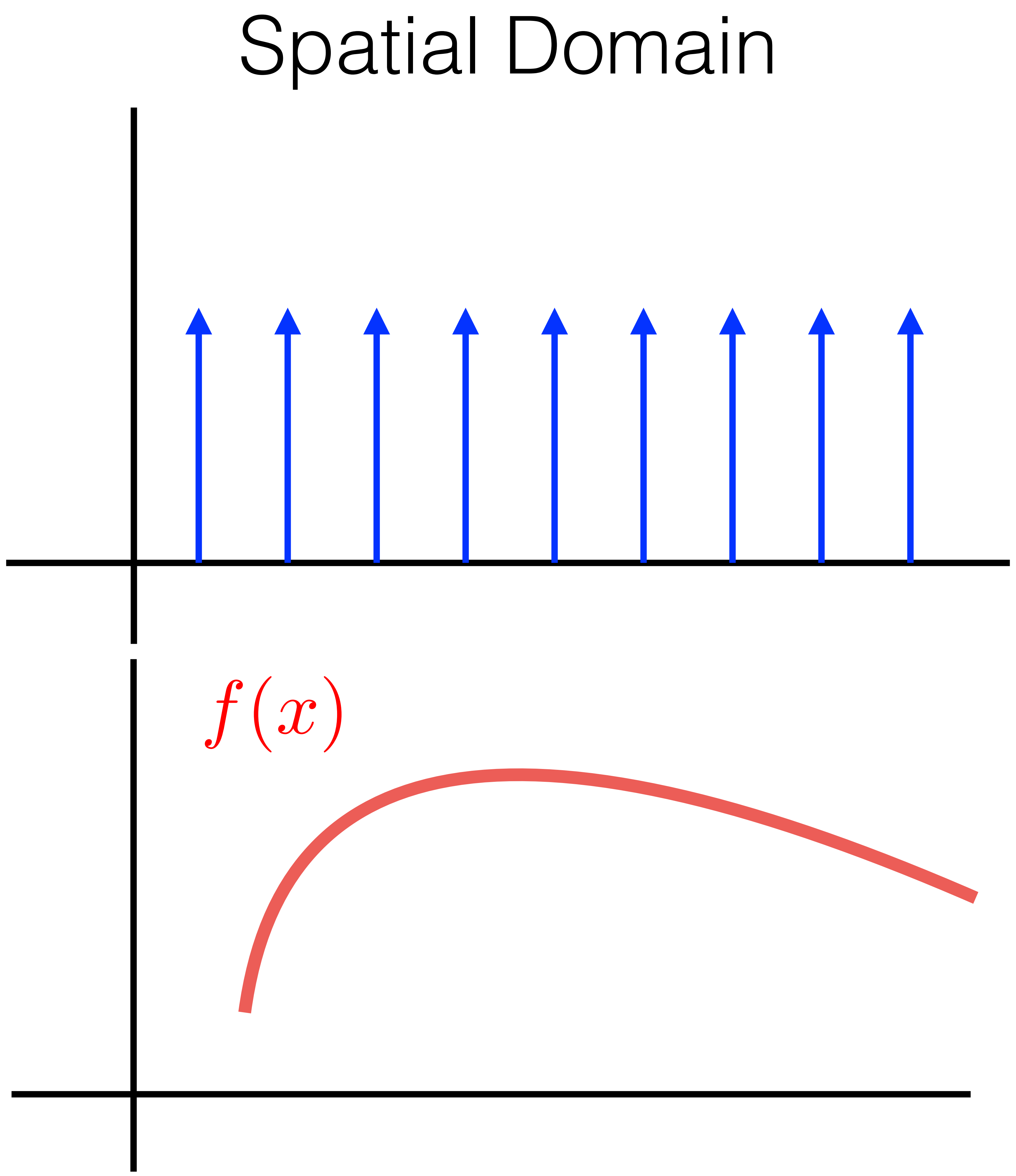


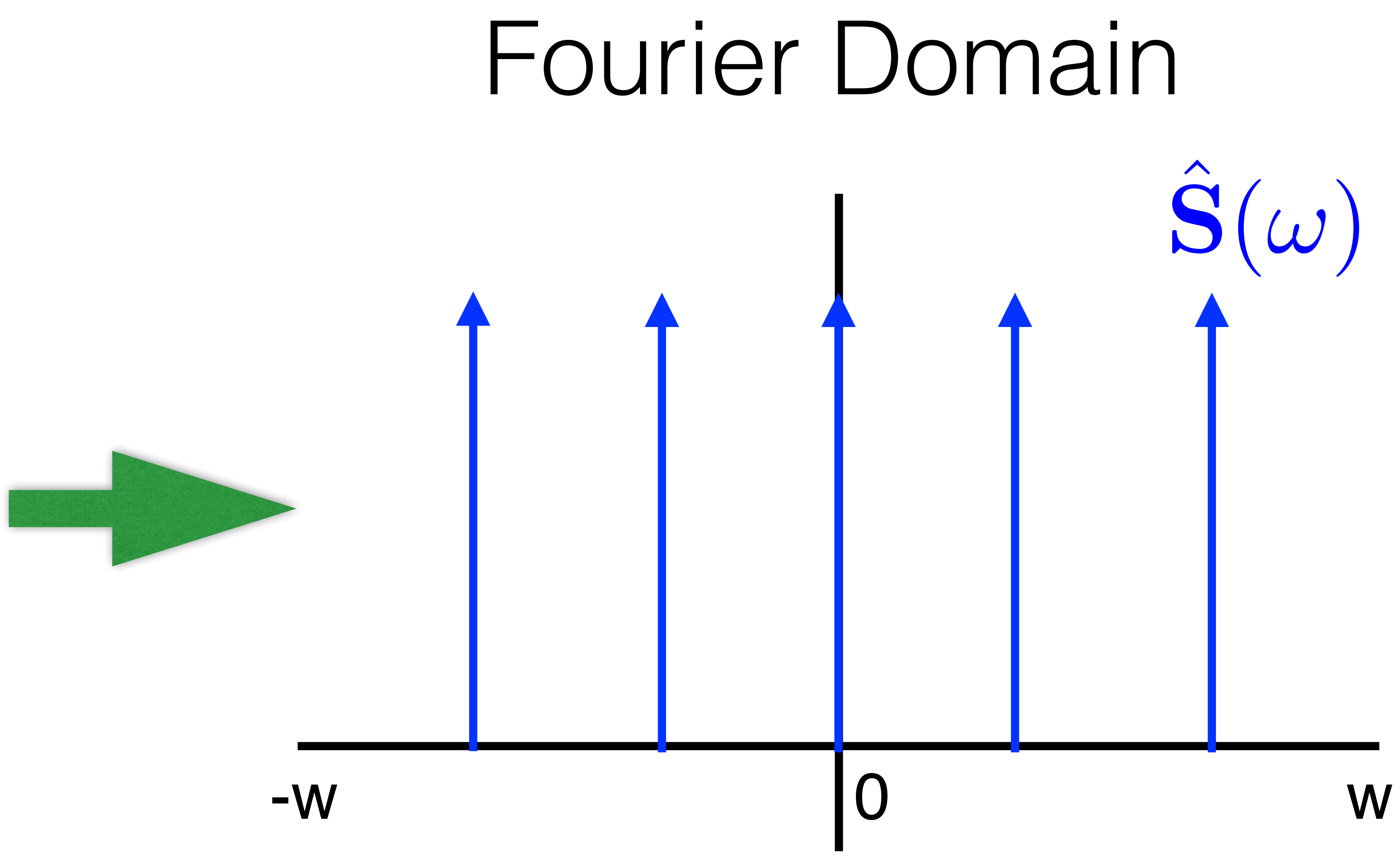
Fourier Domain

Samples and function in Fourier Domain

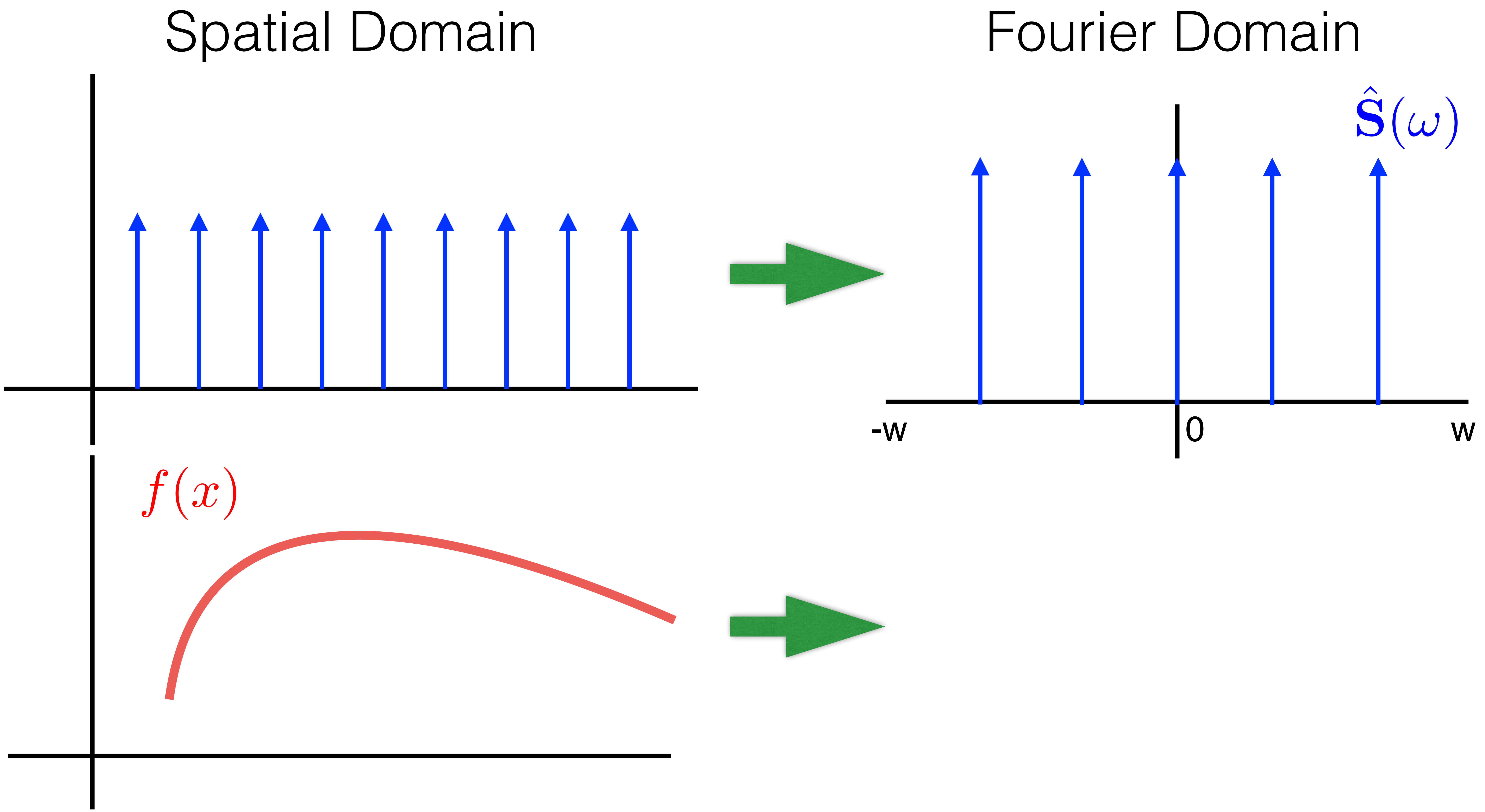


Samples and function in Fourier Domain

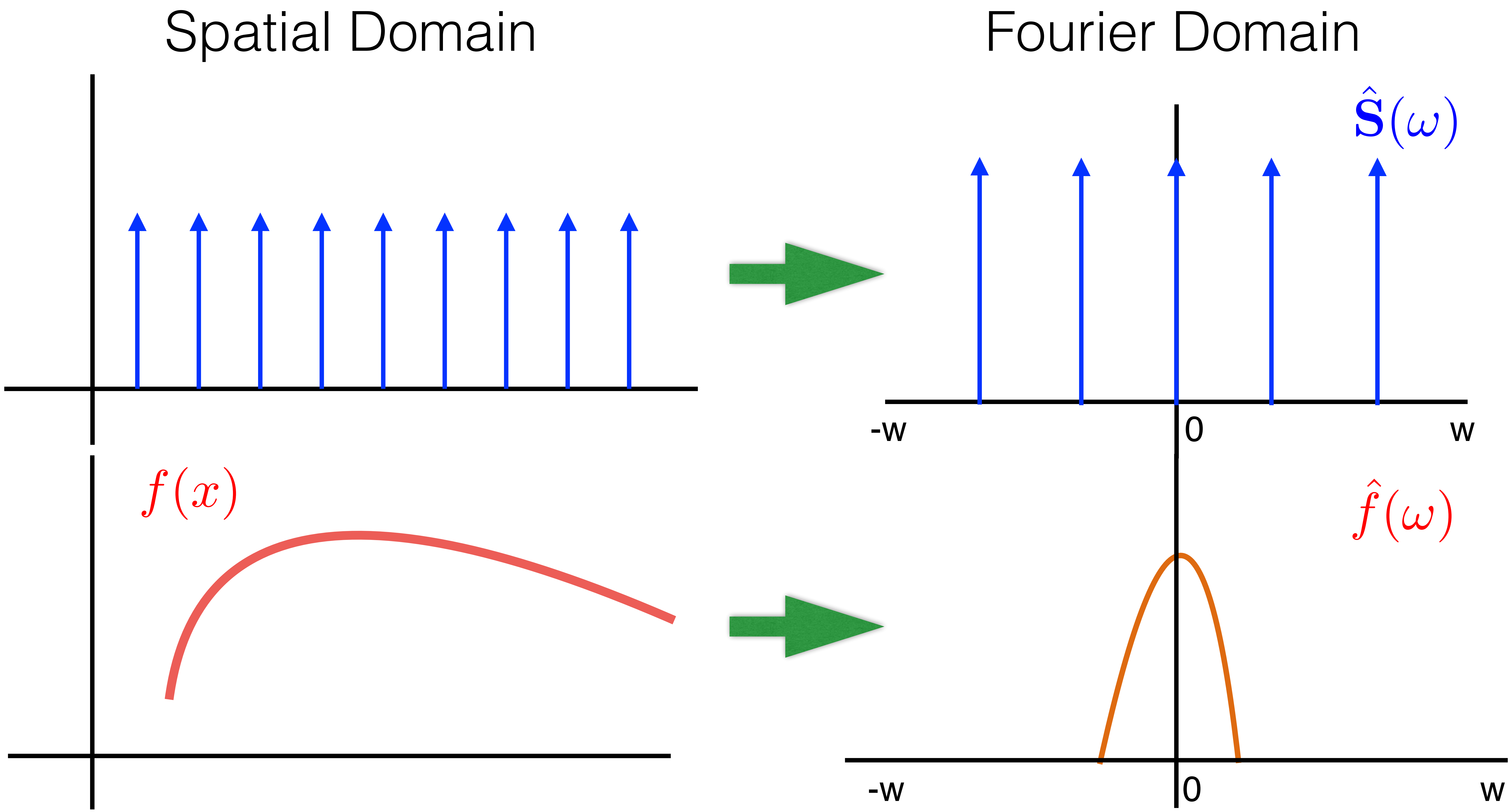




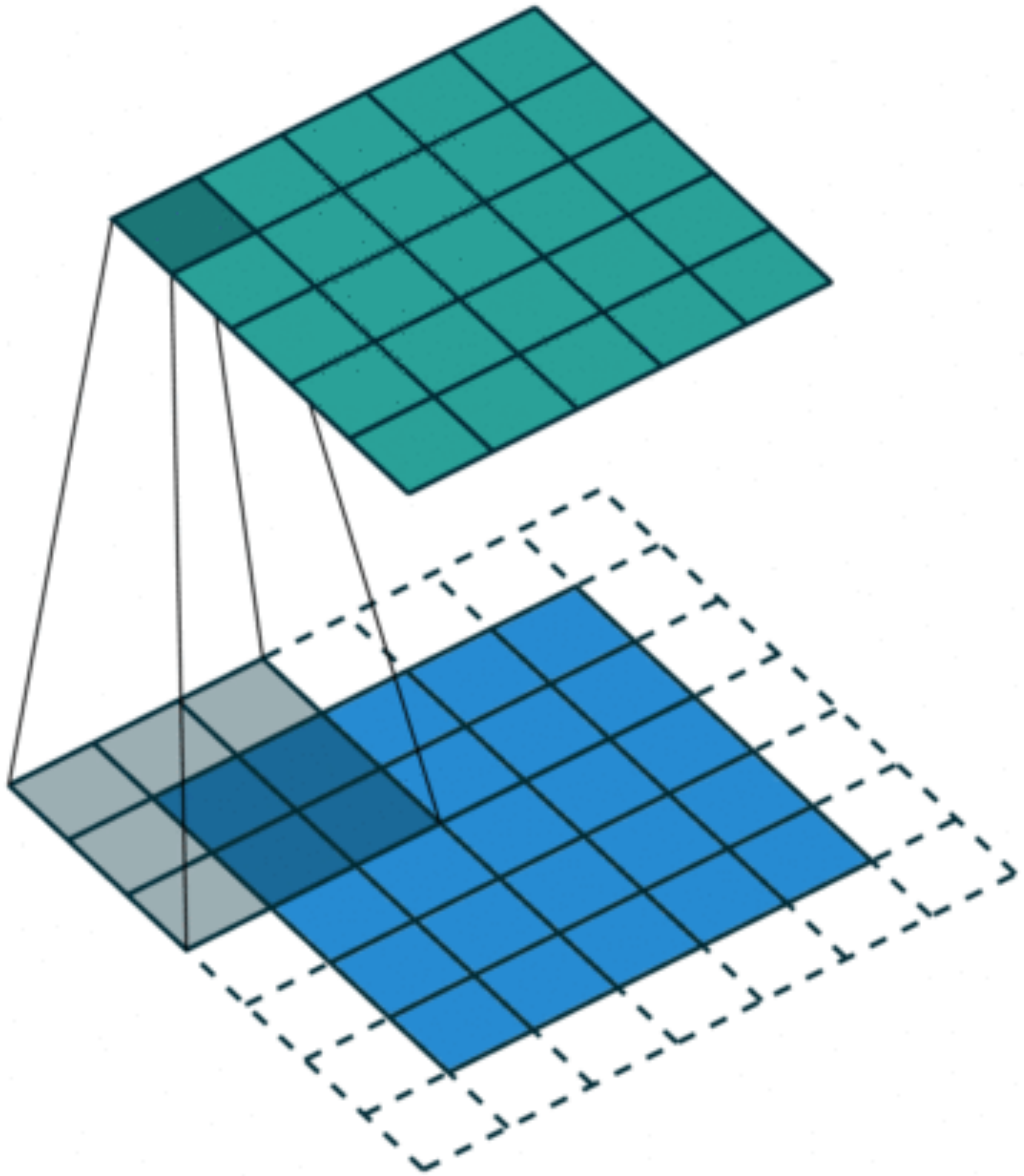
Samples and function in Fourier Domain



Samples and function in Fourier Domain

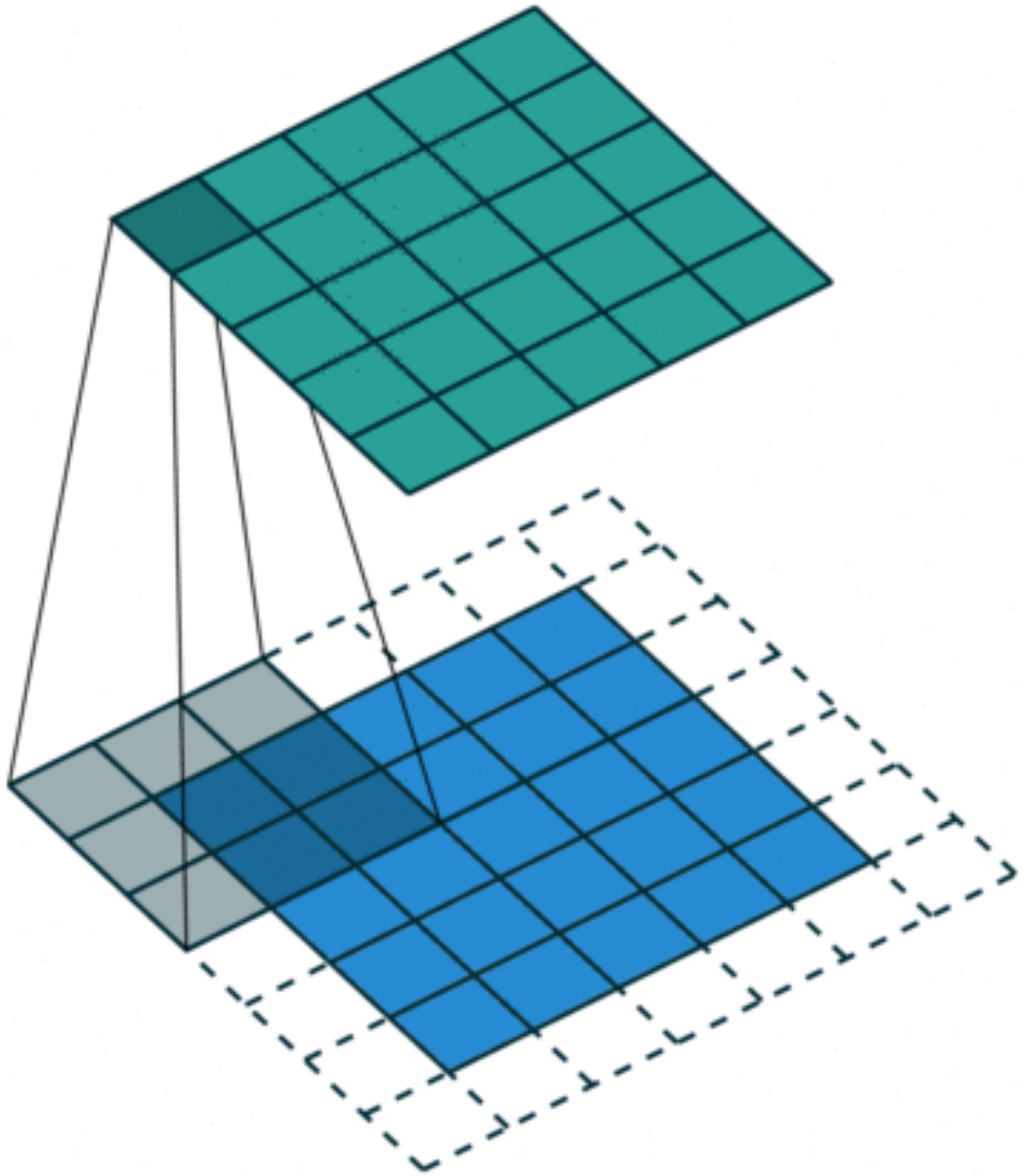


Convolution



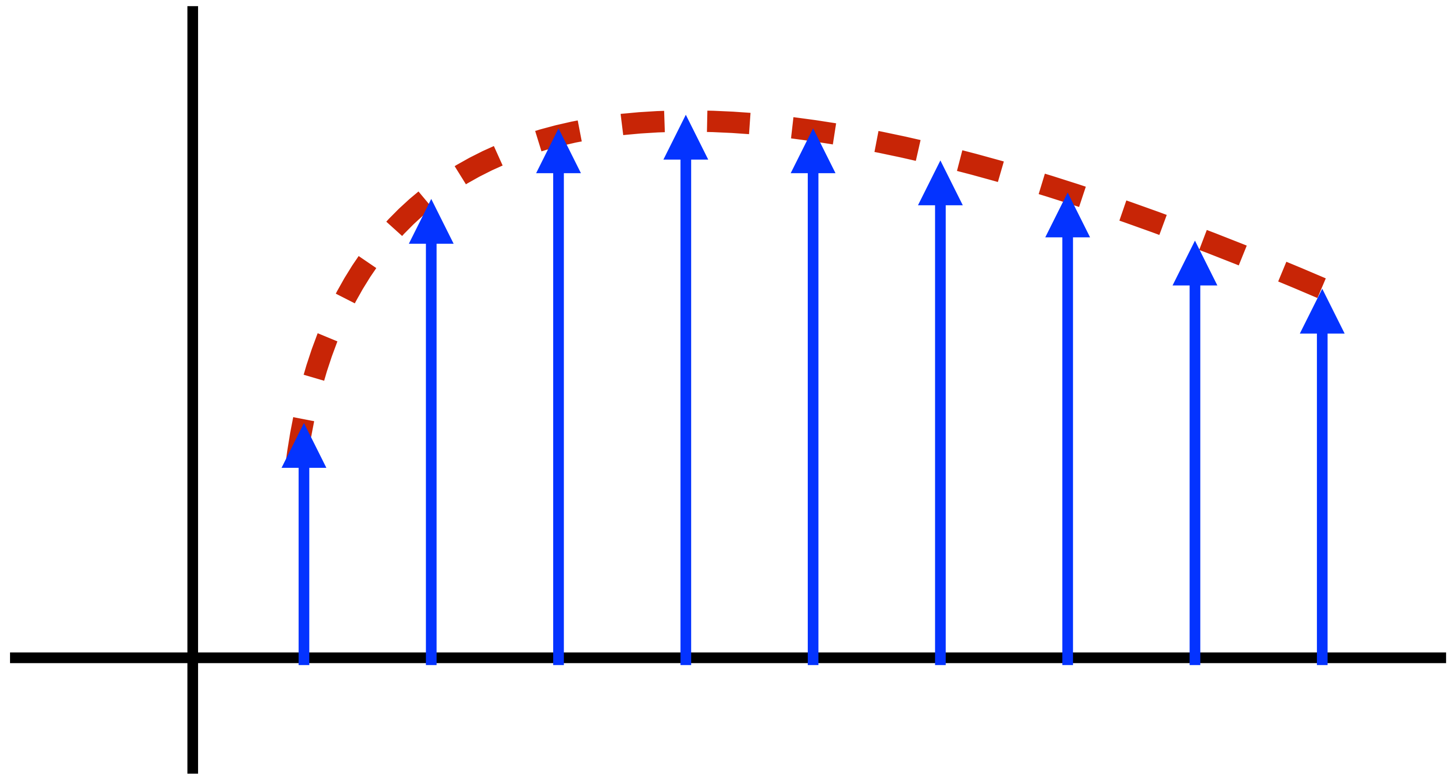
Source: vdumoulin-github

Convolution



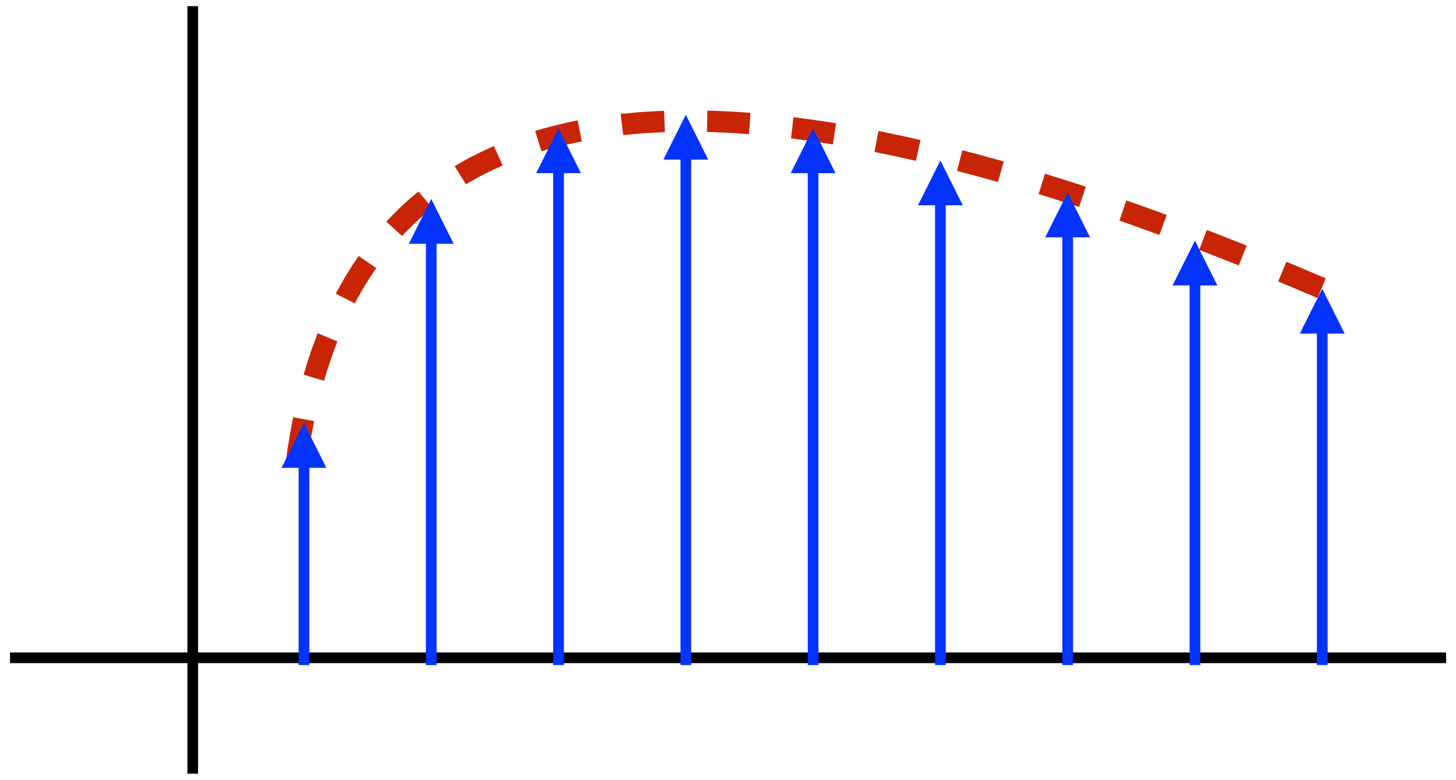
Source: vdumoulin-github

Sampling in Primal Domain is Convolution in Fourier Domain



$f(x)\,\mathbf{S}(x)$

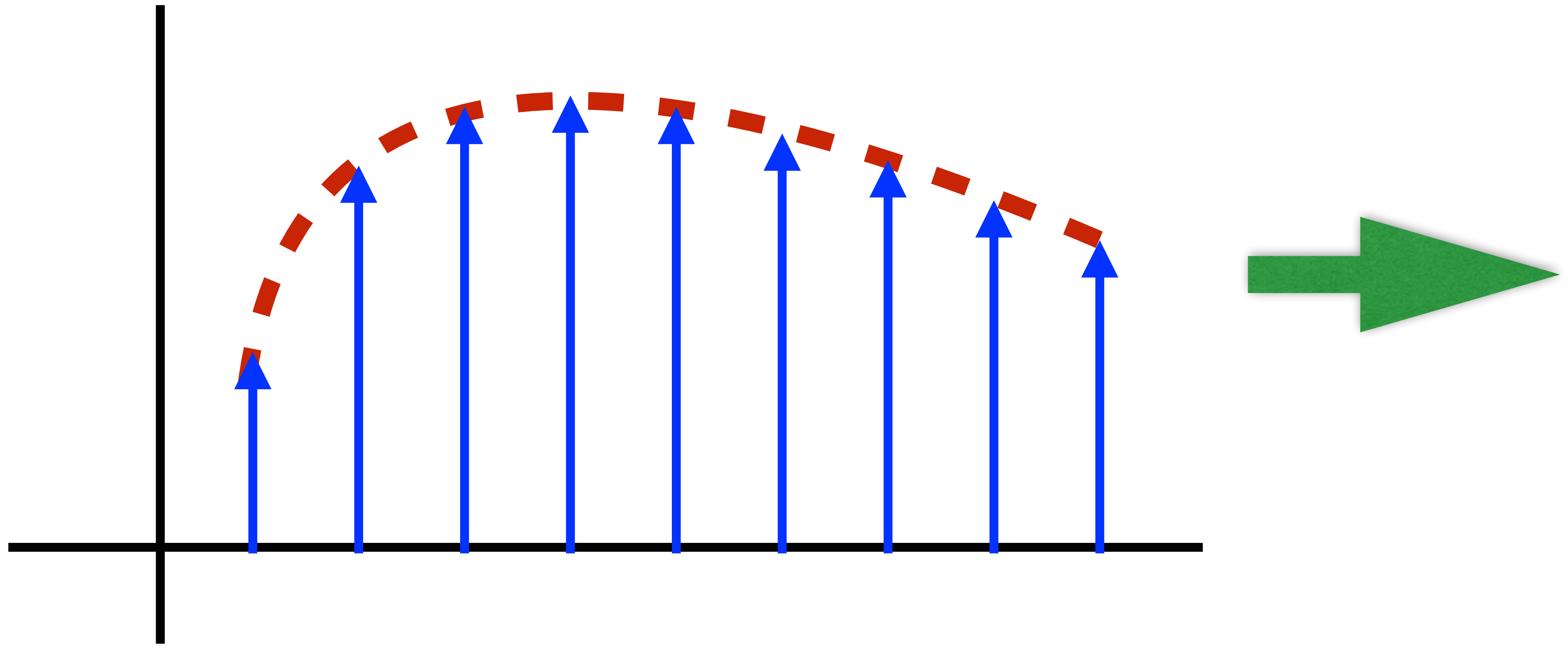
Sampling in Primal Domain is Convolution in Fourier Domain



$f(x)\,\mathbf{S}(x)$

Fredo Durand [2011]

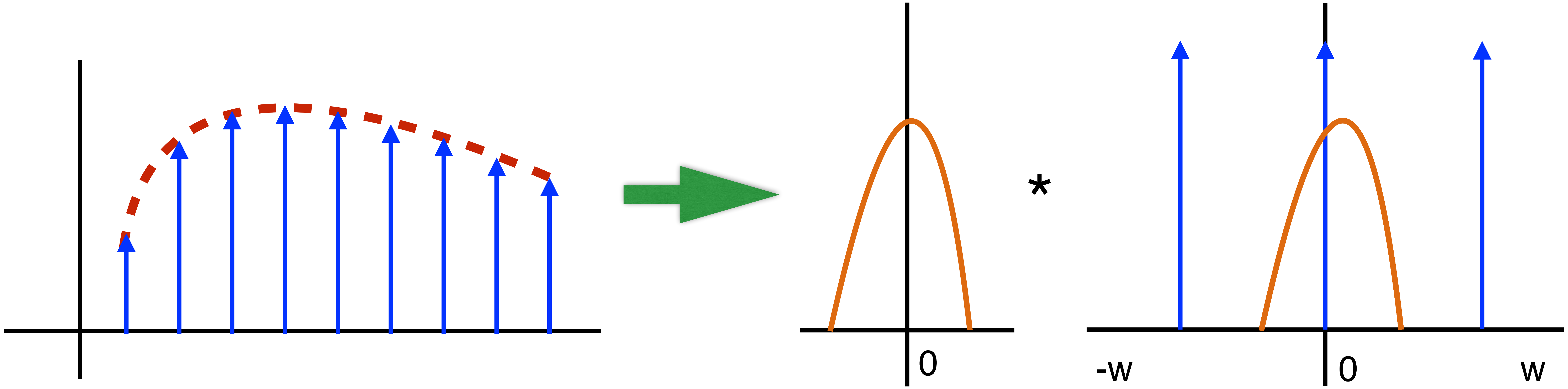
Sampling in Primal Domain is Convolution in Fourier Domain



$f(x)\,\mathbf{S}(x)$

Fredo Durand [2011]

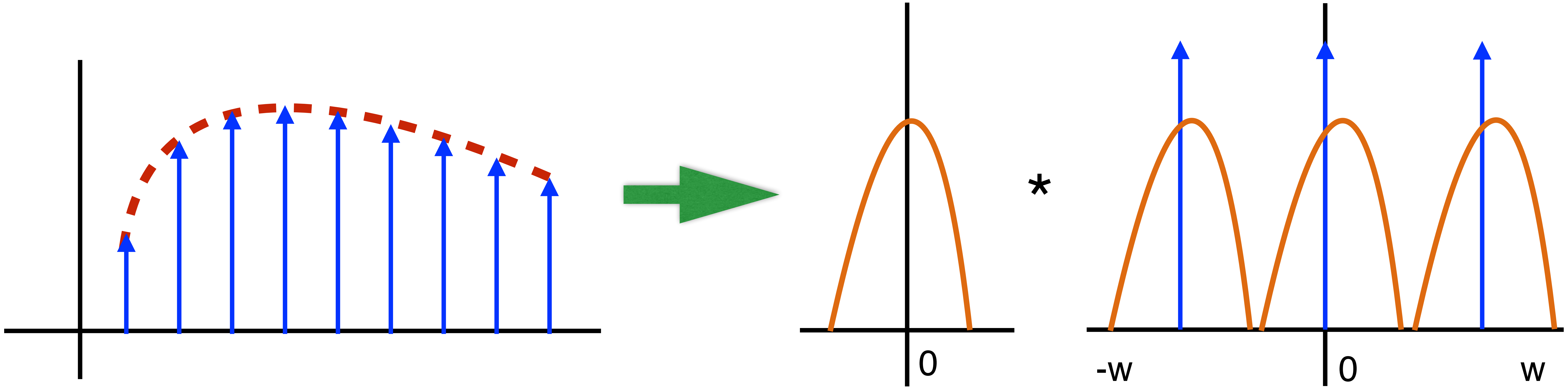
Sampling in Primal Domain is Convolution in Fourier Domain



$f(x)\,\mathbf{S}(x)$ $\quad\quad$ $\hat{f}(\omega) \otimes \hat{\mathbf{S}}(\omega)$

Fredo Durand [2011]

Sampling in Primal Domain is Convolution in Fourier Domain

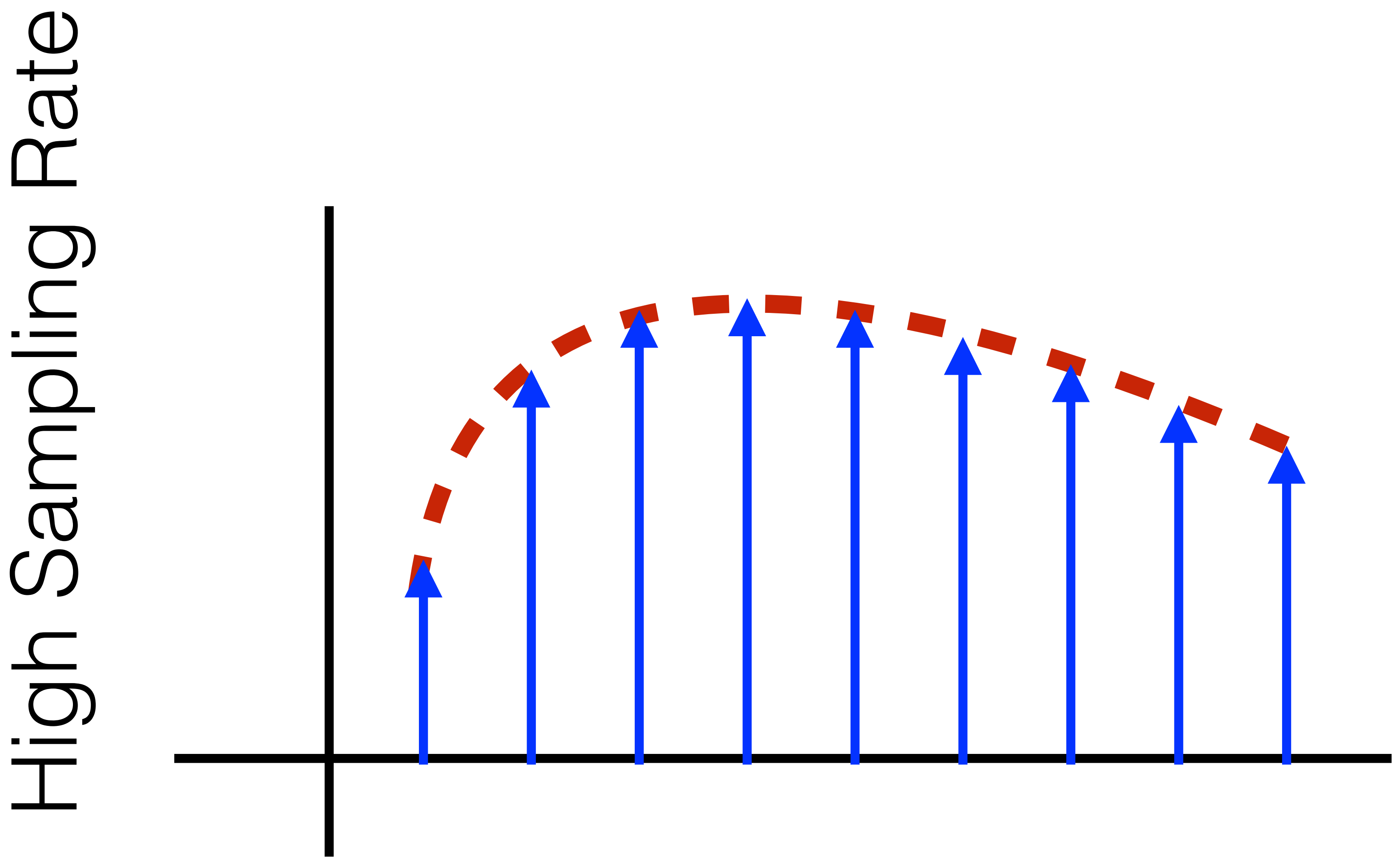


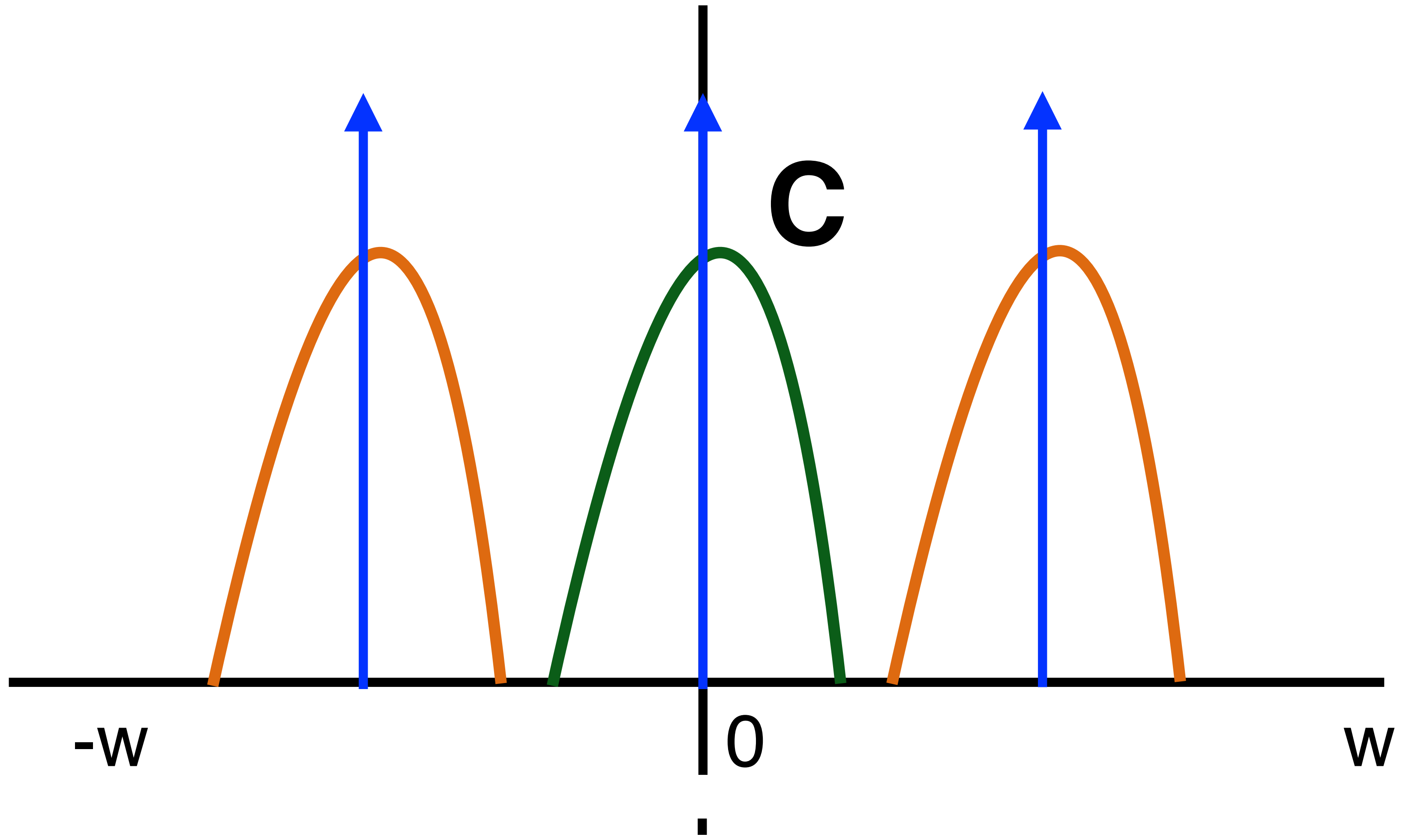
$$f(x)\,\mathbf{S}(x)$$

$$\hat{f}(\omega) \otimes \hat{\mathbf{S}}(\omega)$$

Fredo Durand [2011]

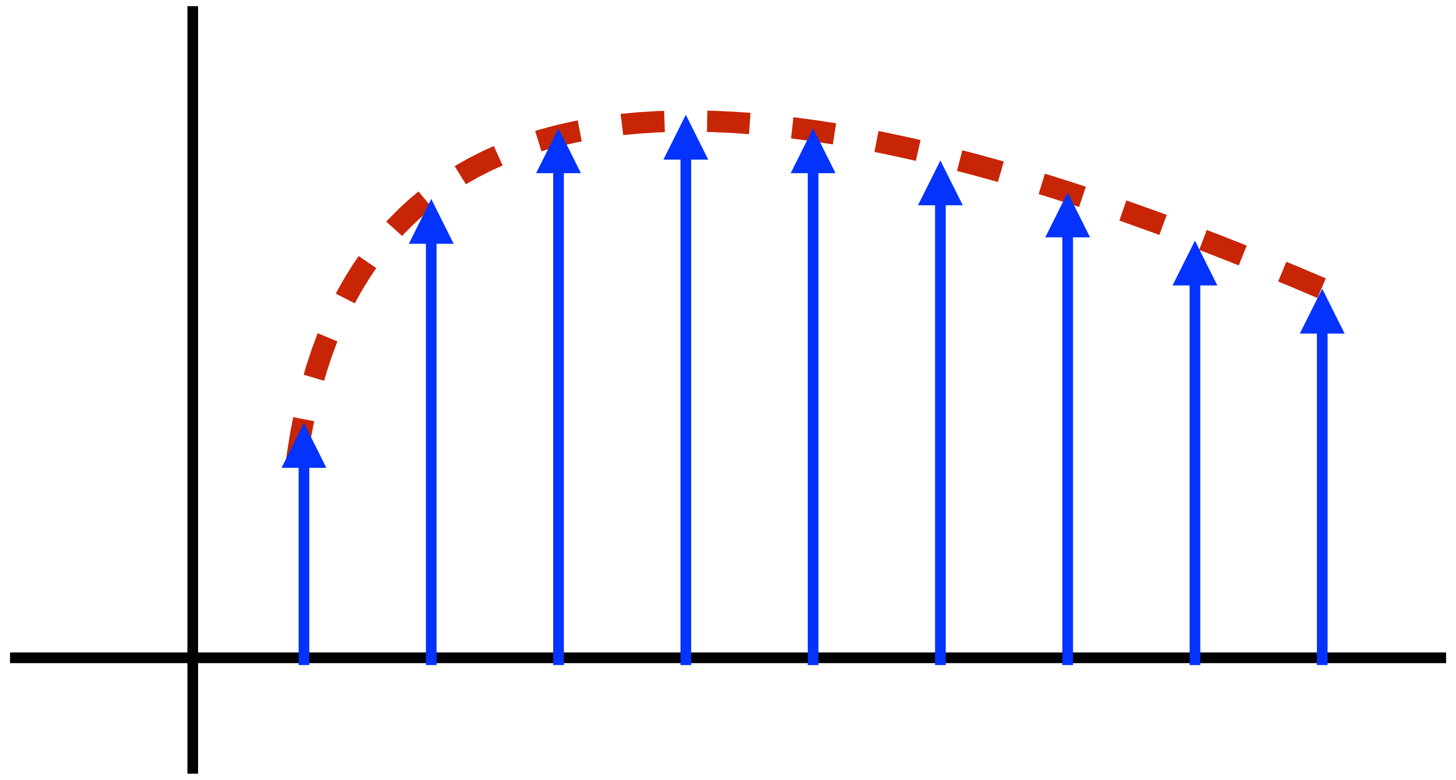
Aliasing in Reconstruction

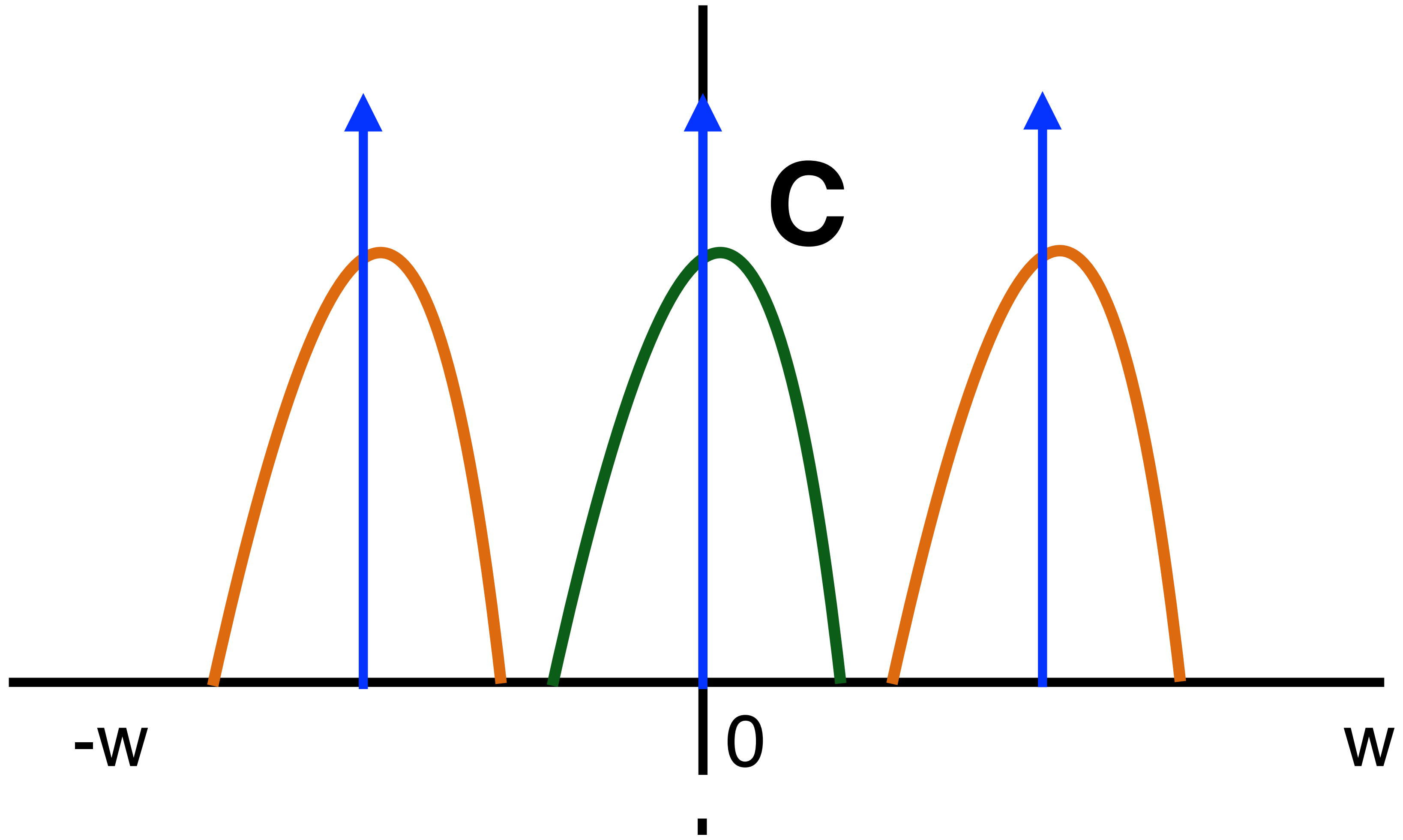




Aliasing in Reconstruction

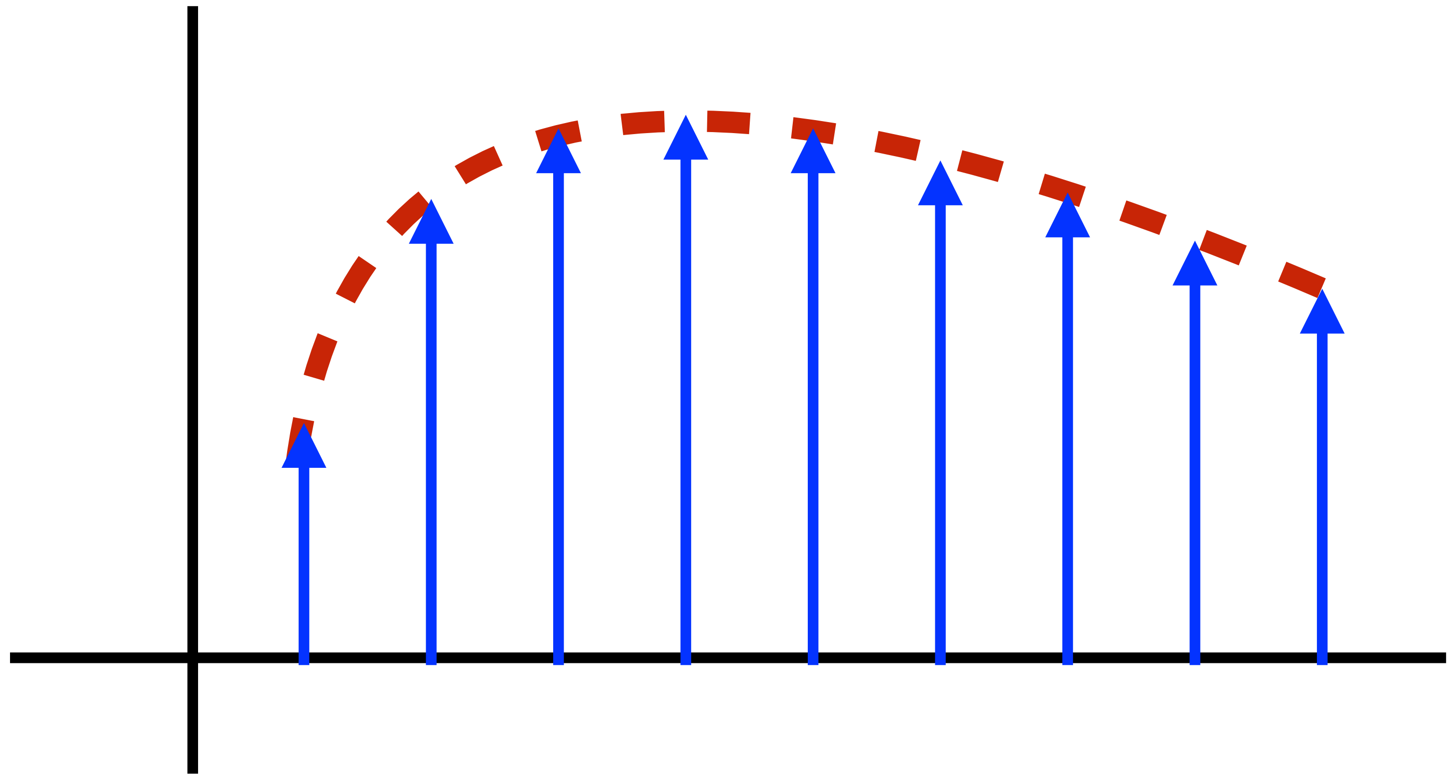
High Sampling Rate

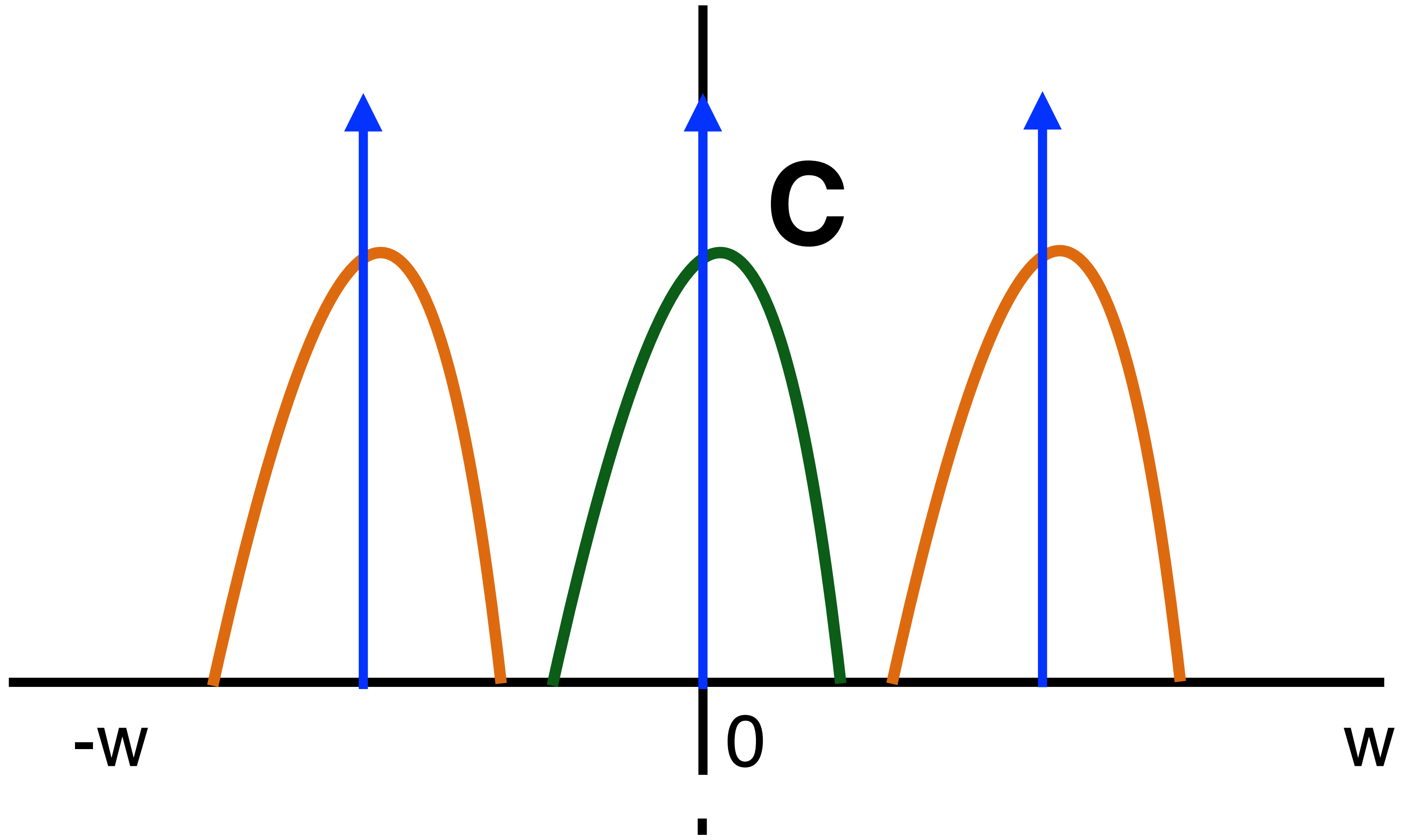




Aliasing in Reconstruction

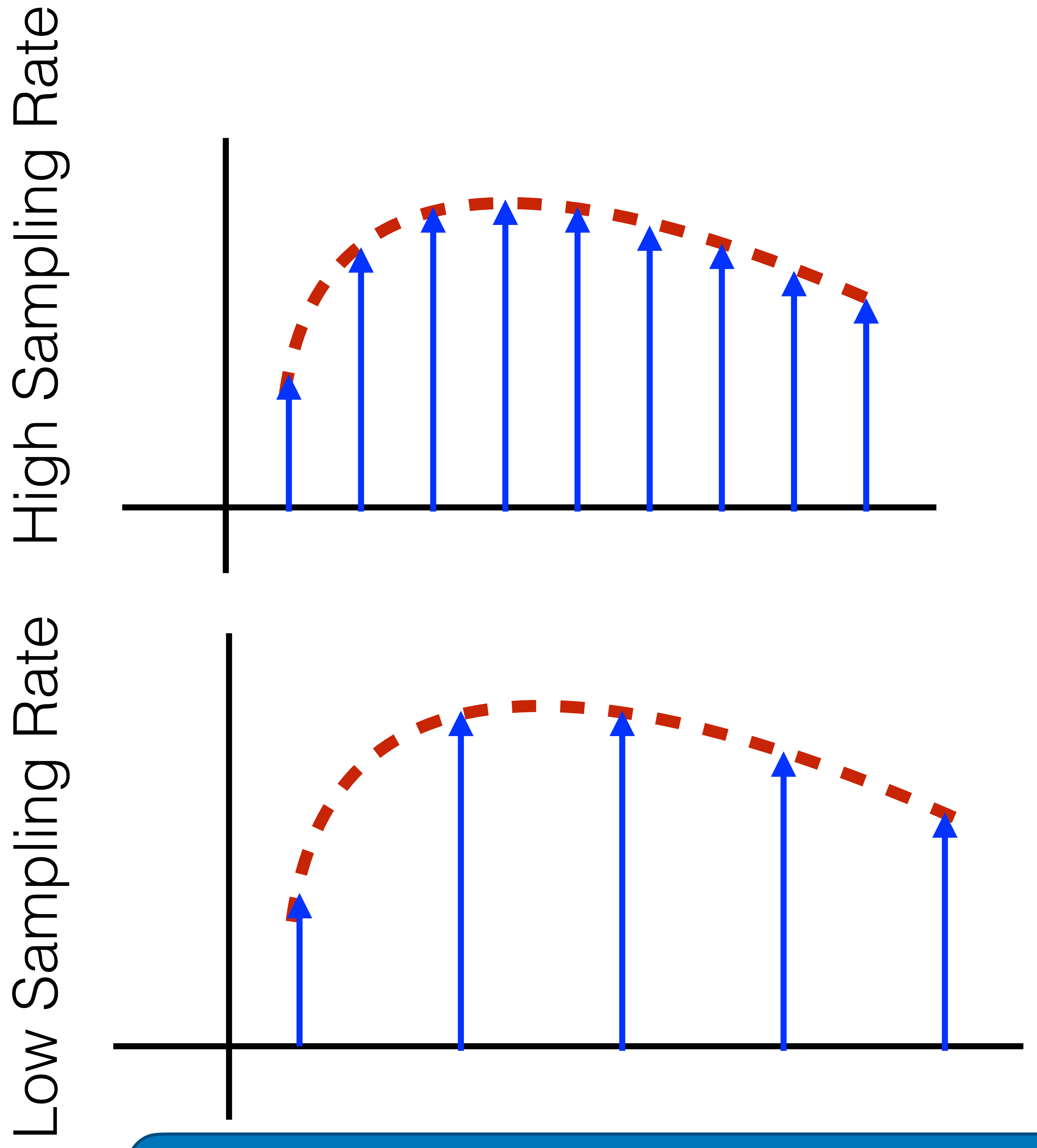
High Sampling Rate

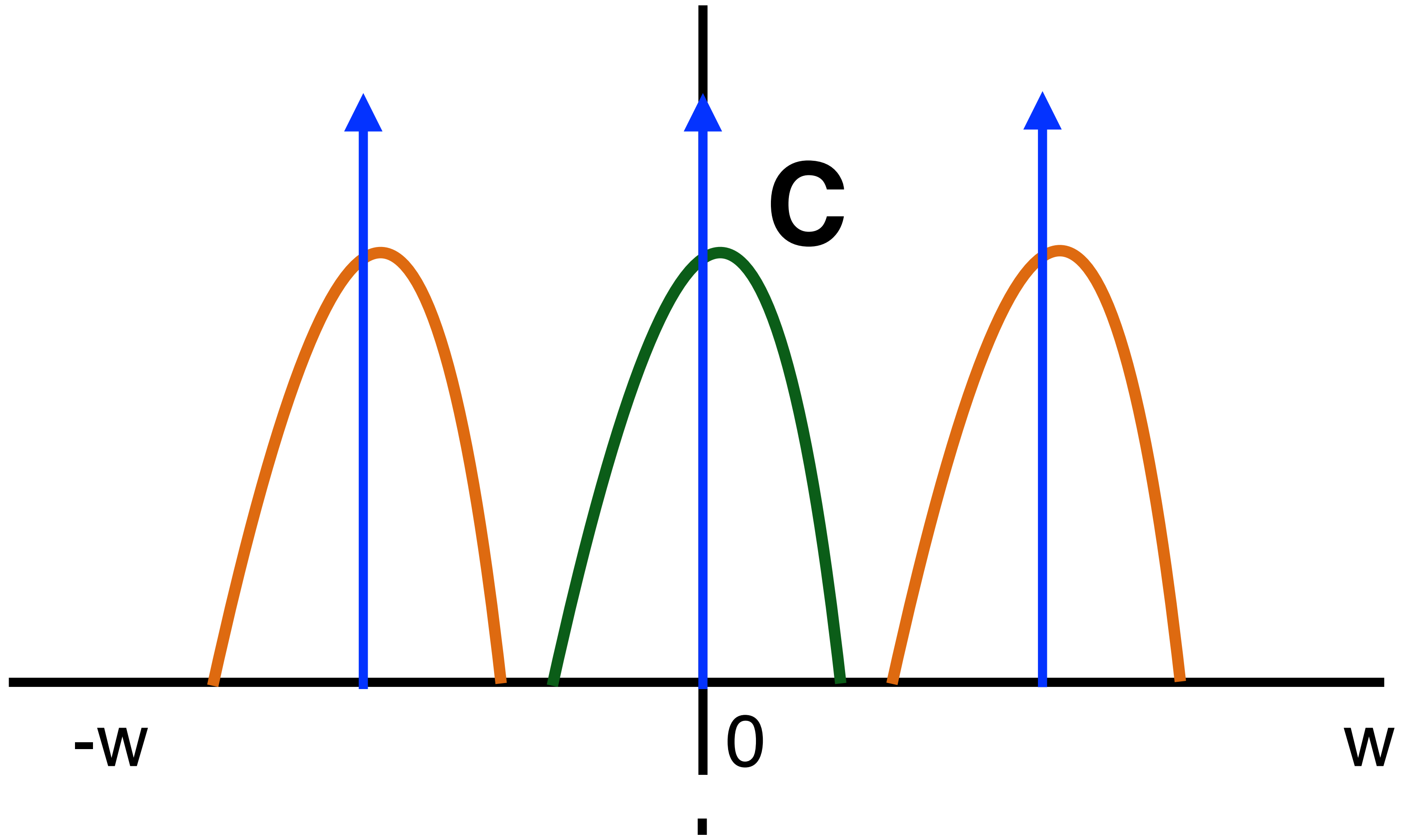




Low Sampling Rate

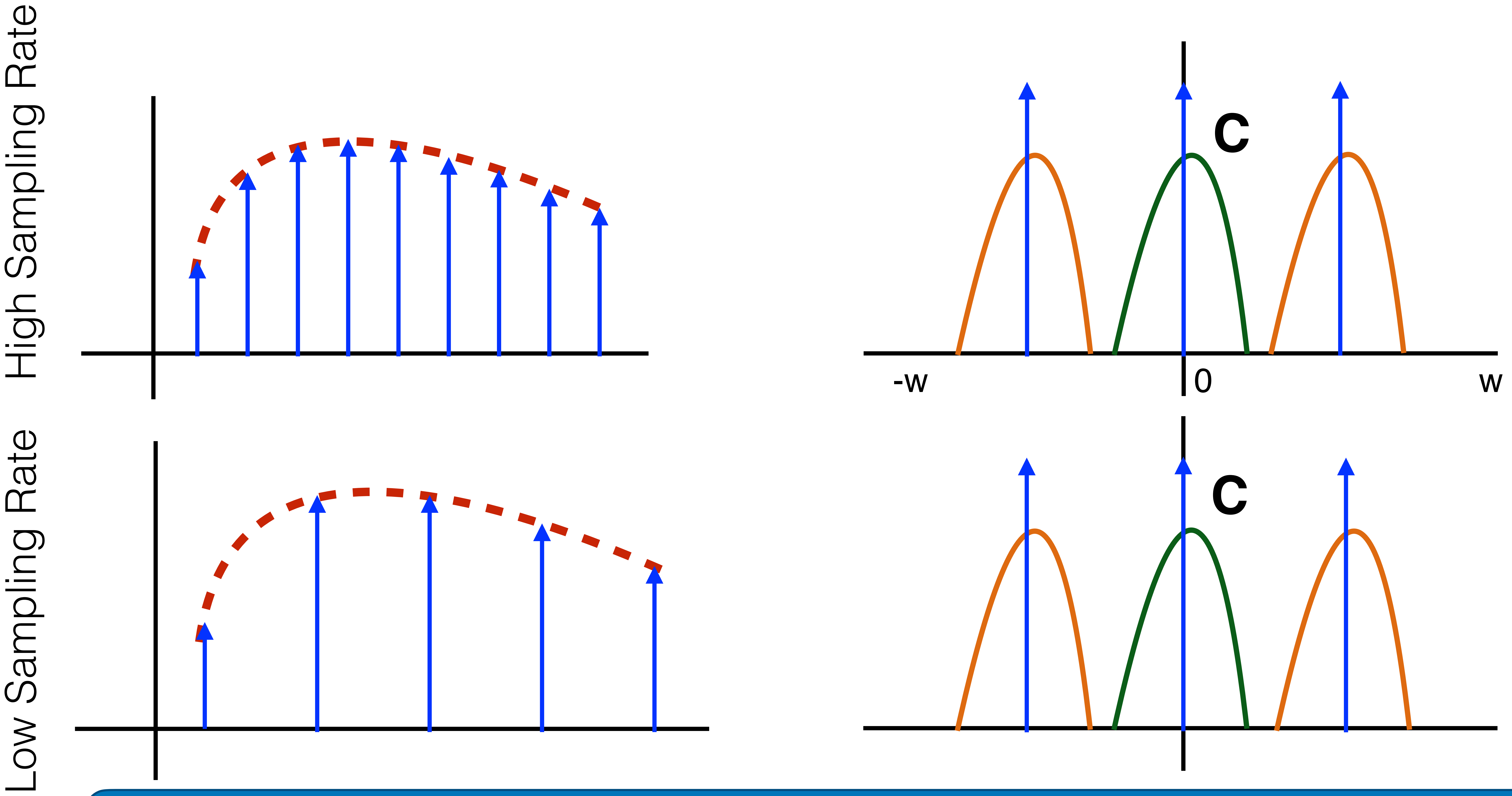
Aliasing in Reconstruction



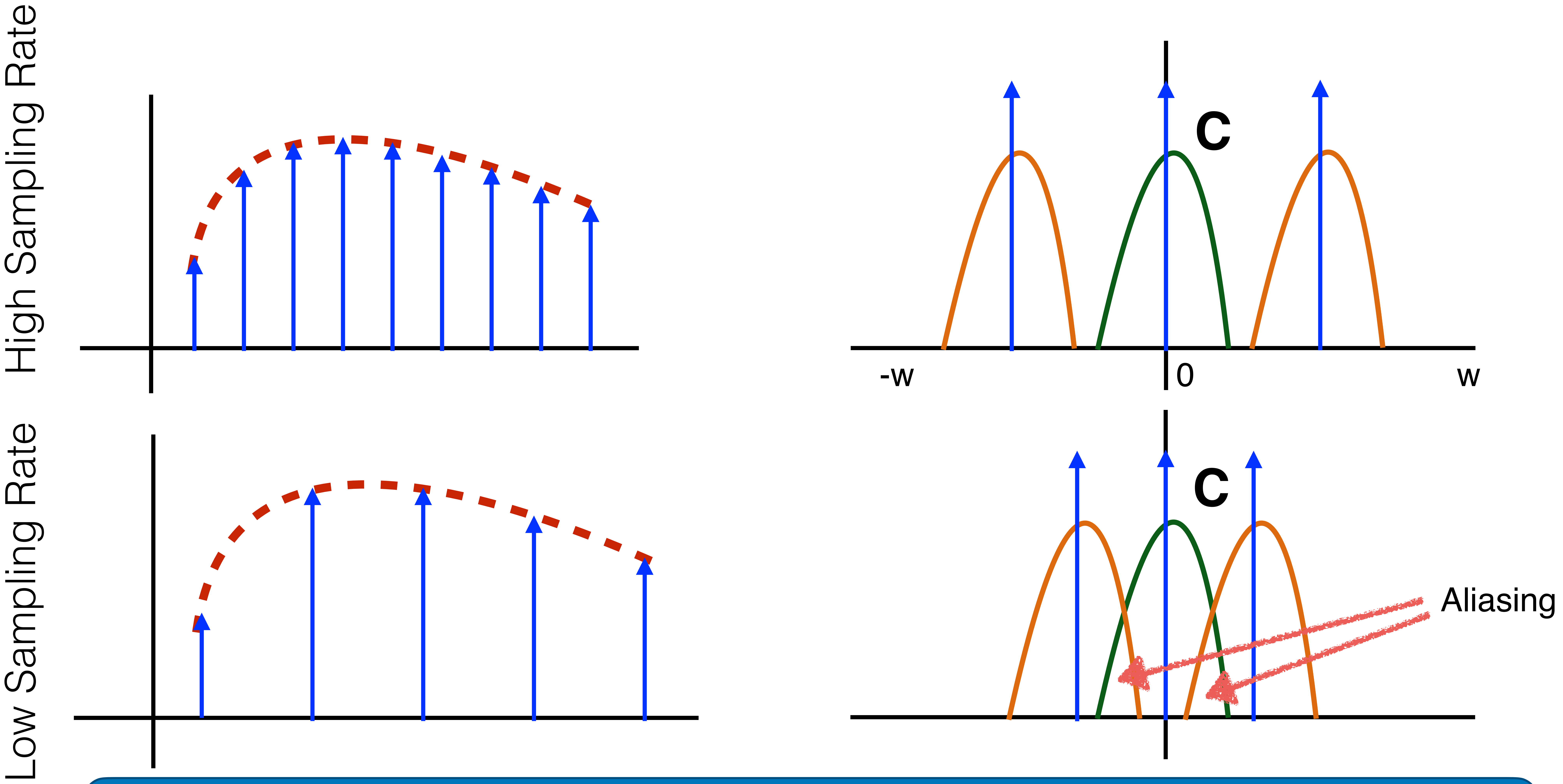


UNIVERSITÄT DES SAARLANDES

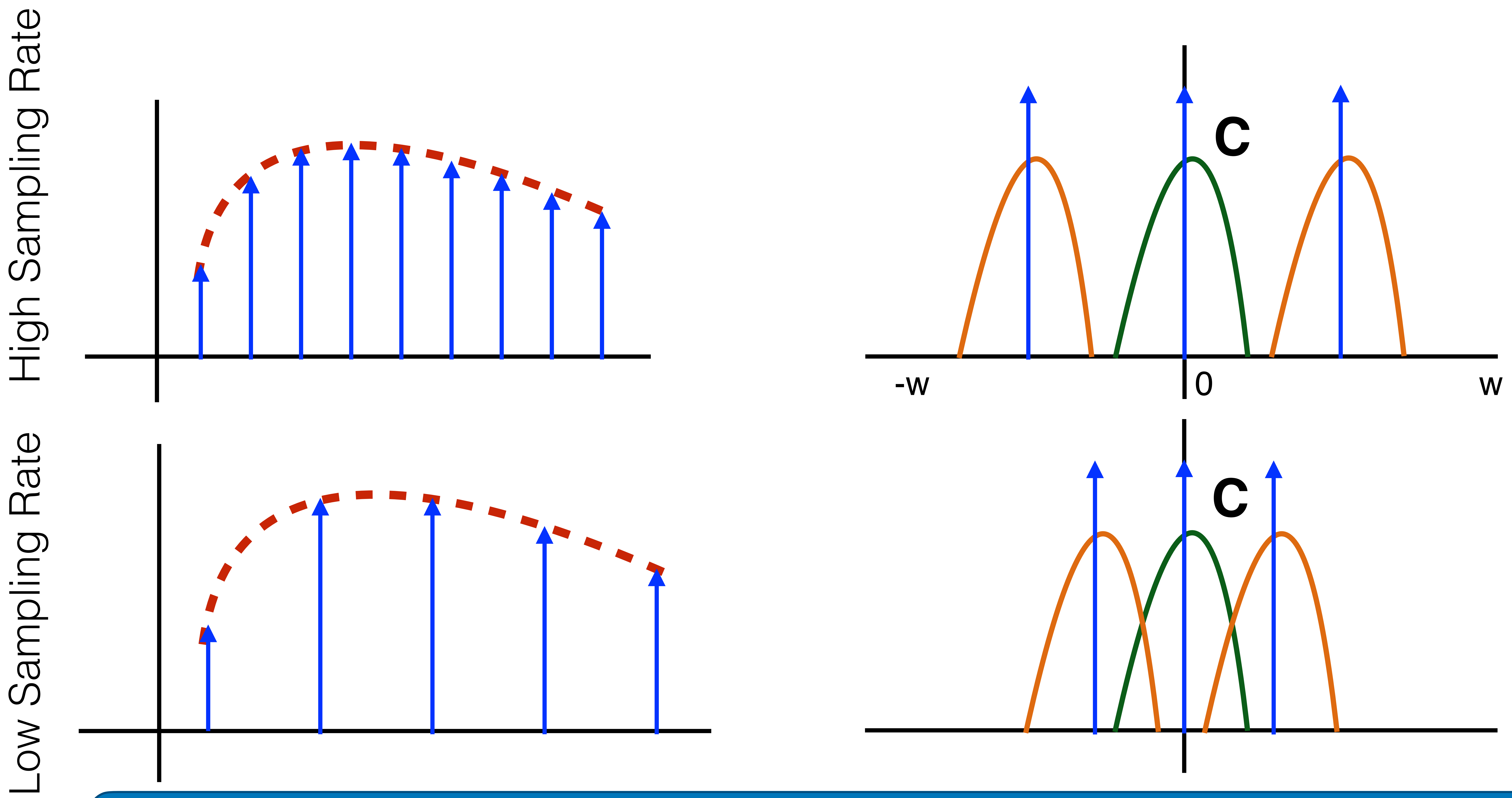
Aliasing in Reconstruction



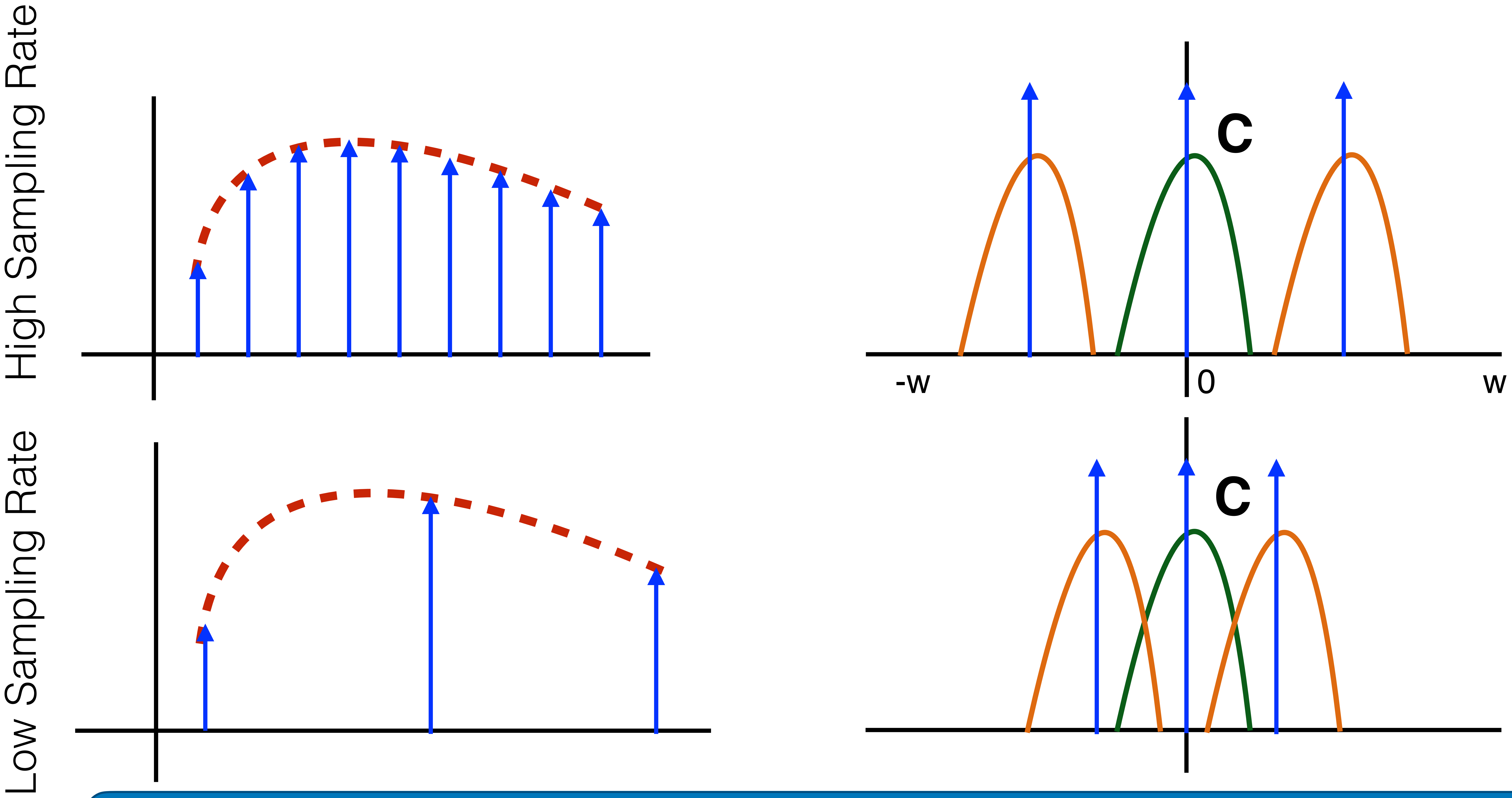
Aliasing in Reconstruction



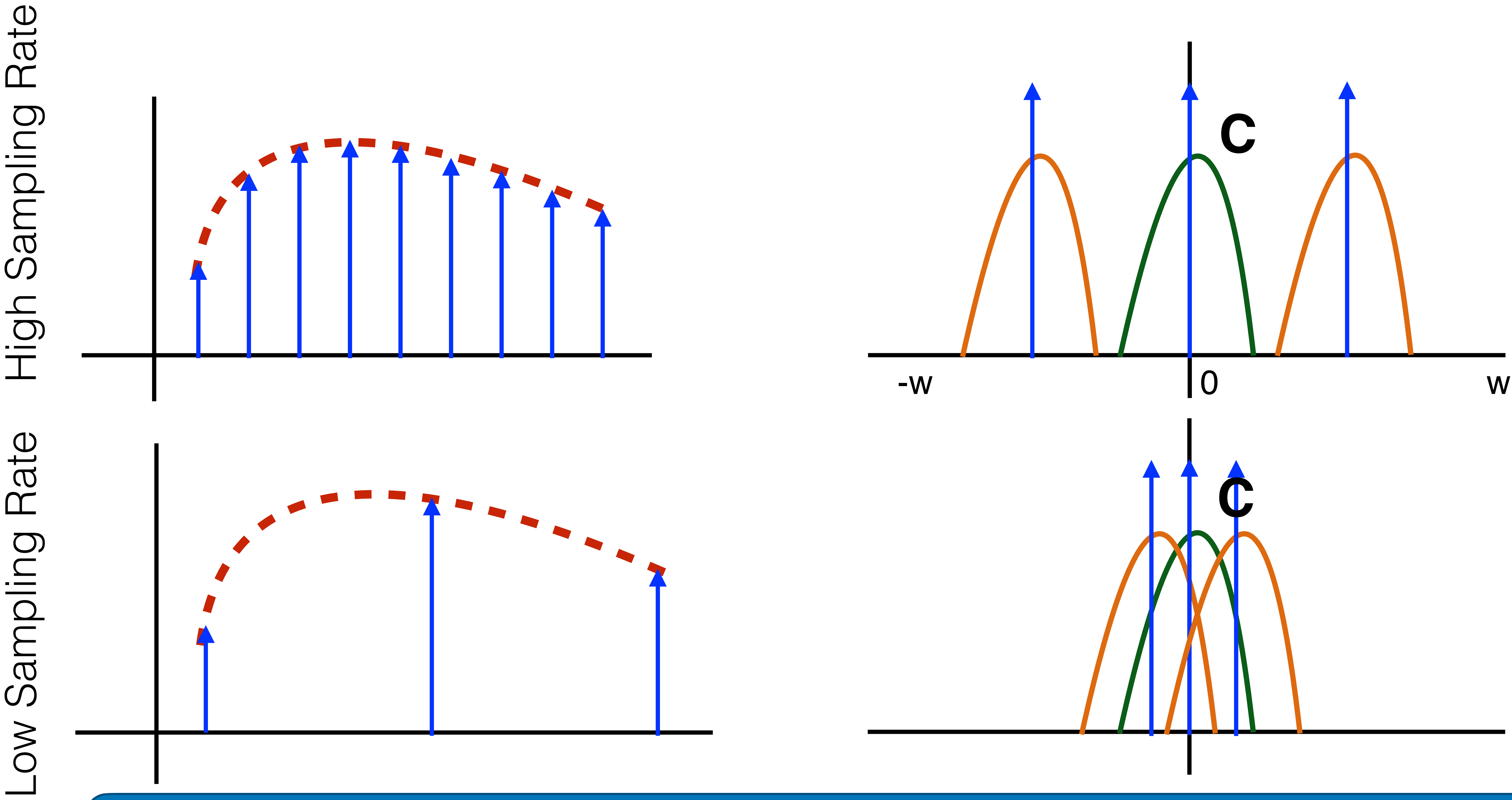
Aliasing in Reconstruction



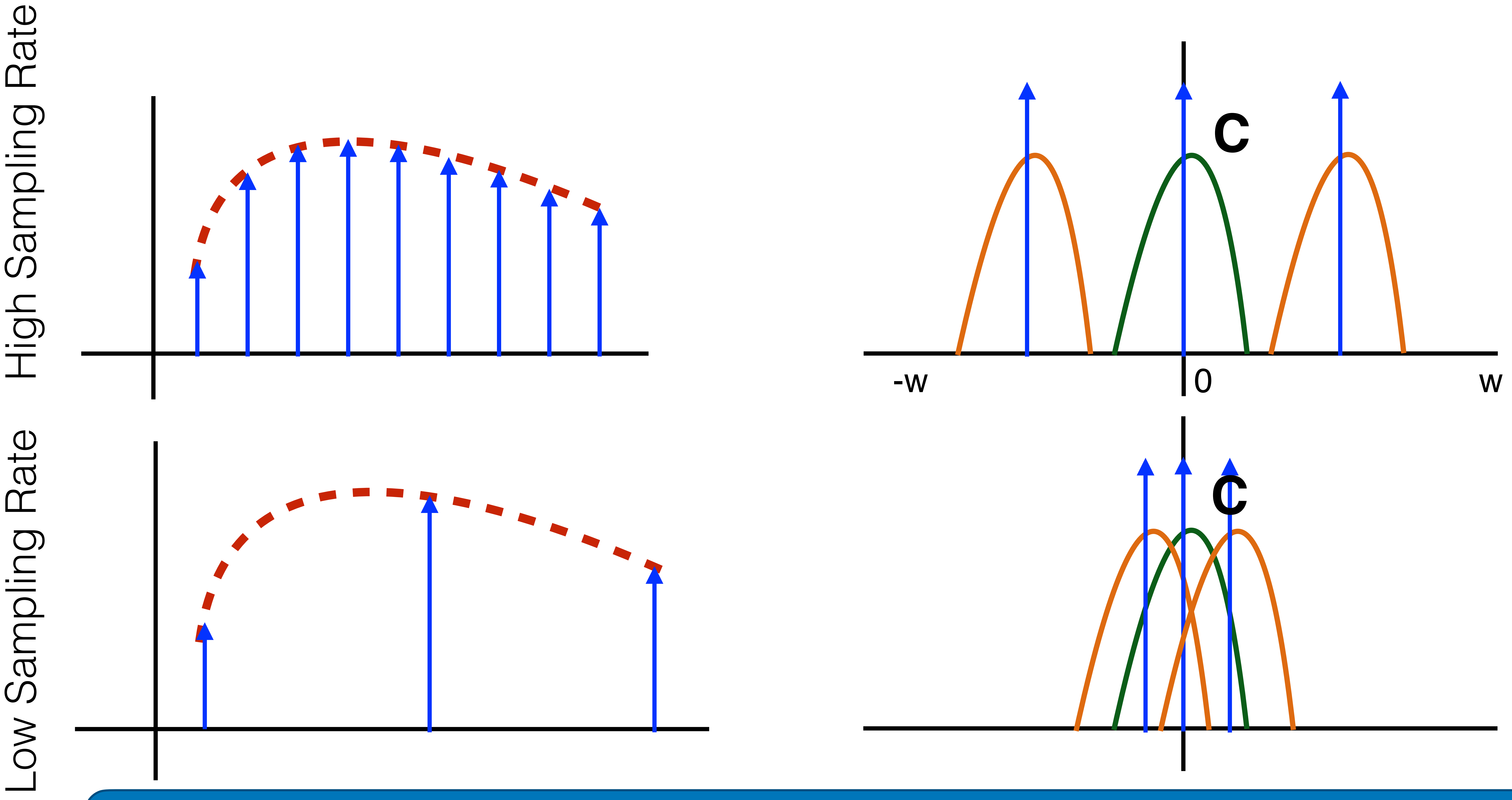
Error in Monte Carlo Integration



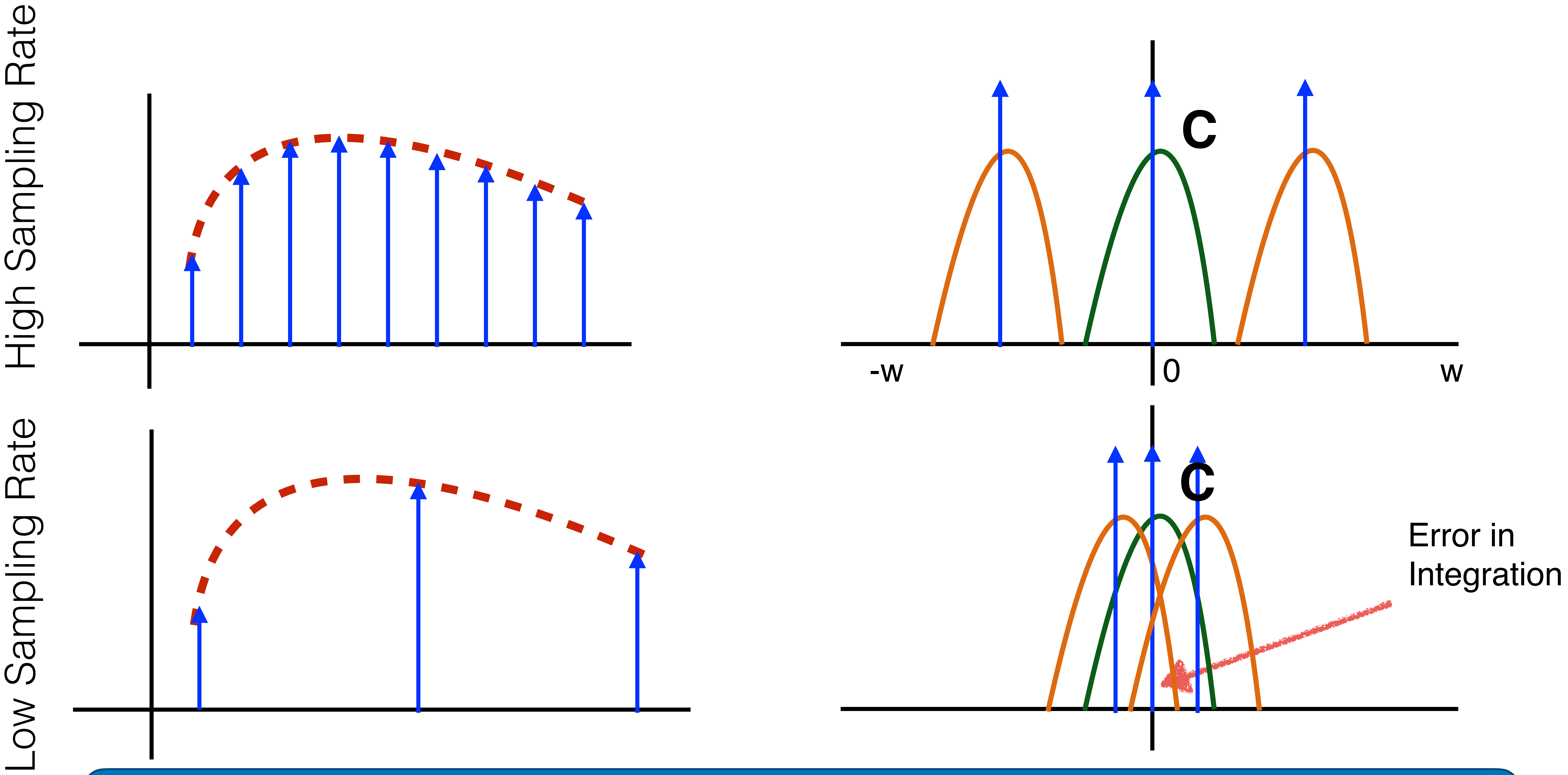
Error in Monte Carlo Integration



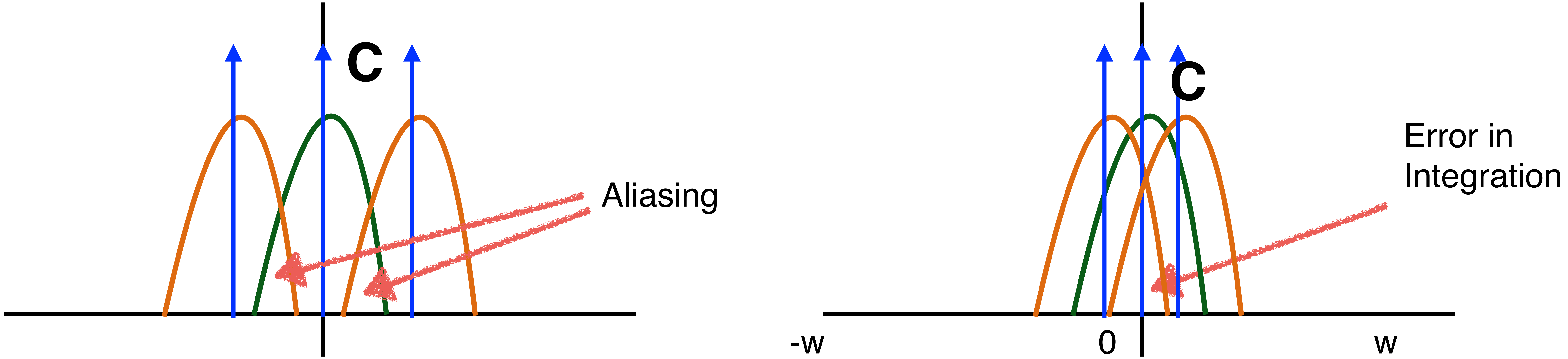
Error in Monte Carlo Integration



Error in Monte Carlo Integration

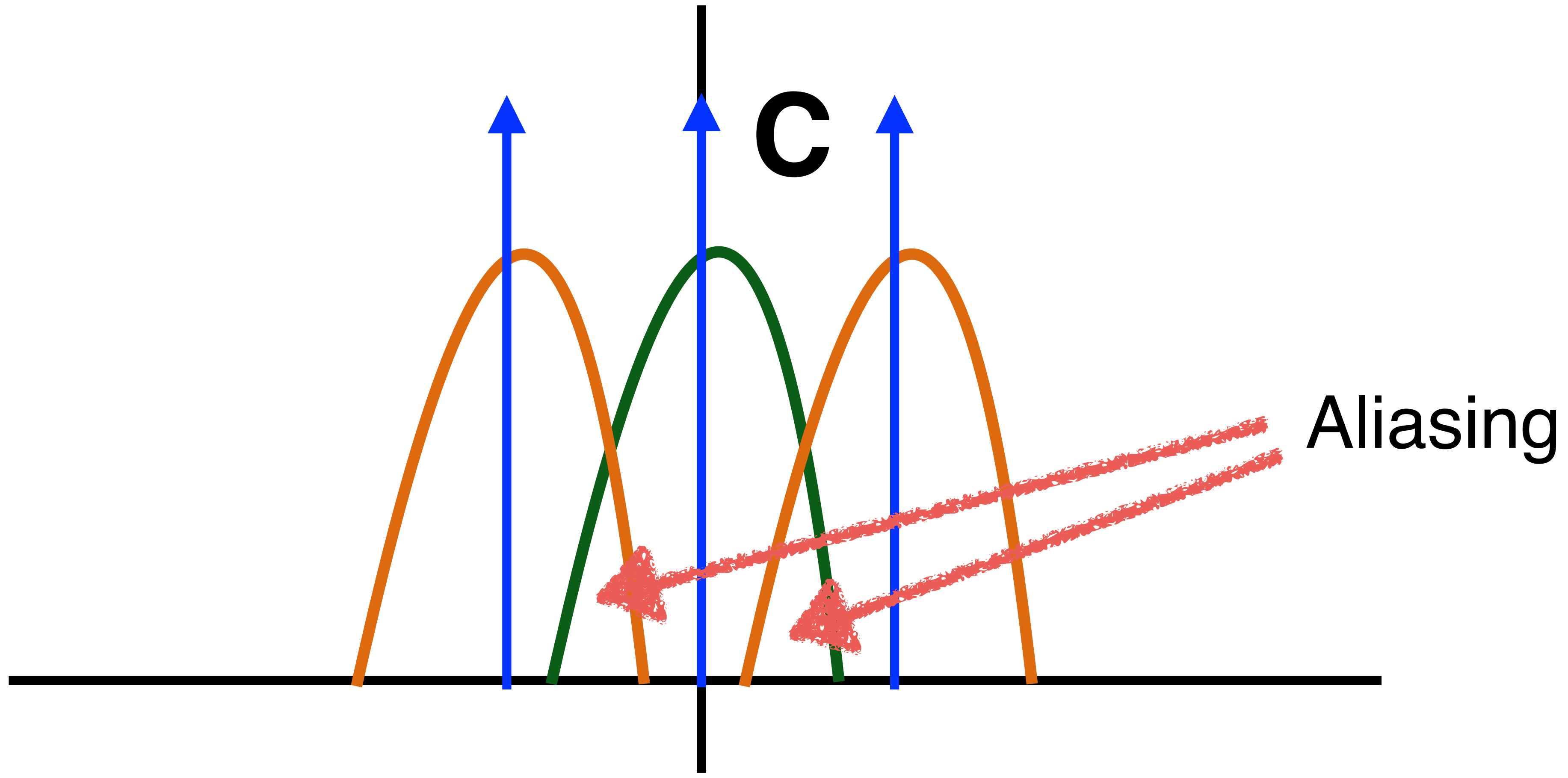


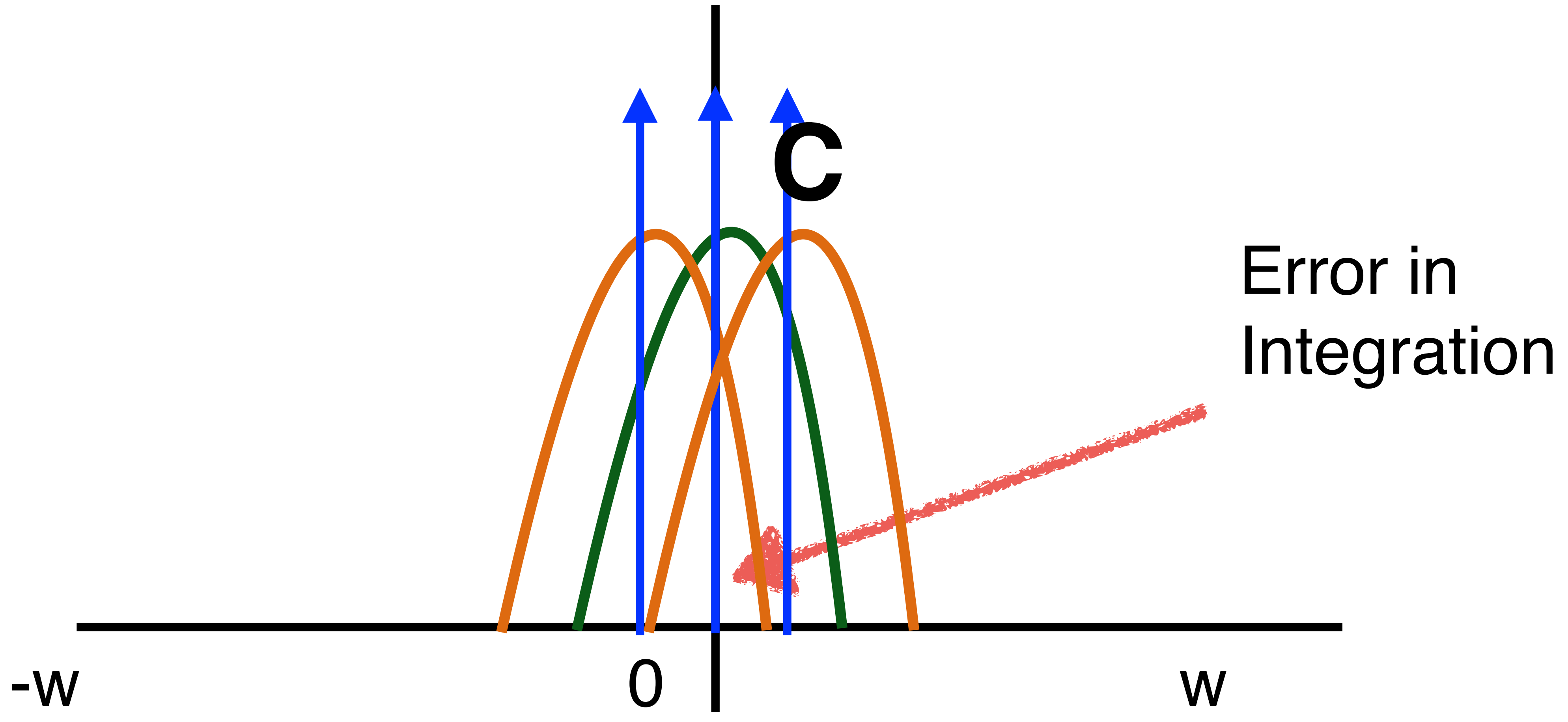
Aliasing (Reconstruction) vs. Error (Integration)



Aliasing (Reconstruction) vs. Error (Integration)

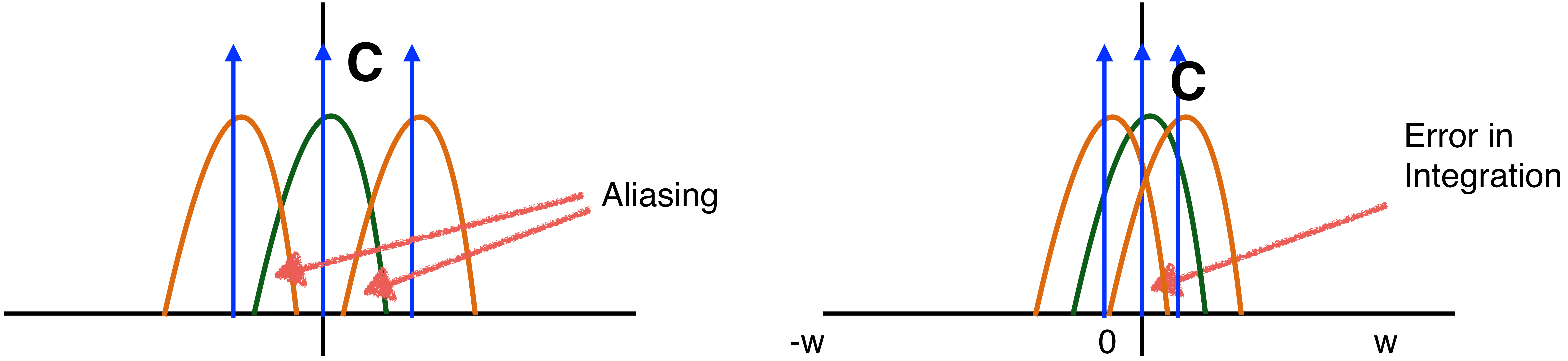
Fredo Durand [2011]
Belcour et al. [2013]





Aliasing (Reconstruction) vs. Error (Integration)

Fredo Durand [2011]
Belcour et al. [2013]



Integration in the Fourier Domain

Integration is the DC term in the Fourier Domain

Spatial Domain:

$$I = \int_D f(x) dx$$

Integration is the DC term in the Fourier Domain

Spatial Domain:

$$I = \int_D f(x)dx$$

Fourier Domain:

Integration is the DC term in the Fourier Domain

Spatial Domain:

$$I = \int_D f(x) dx$$

Fourier Domain:

$$\hat{f}(0)$$

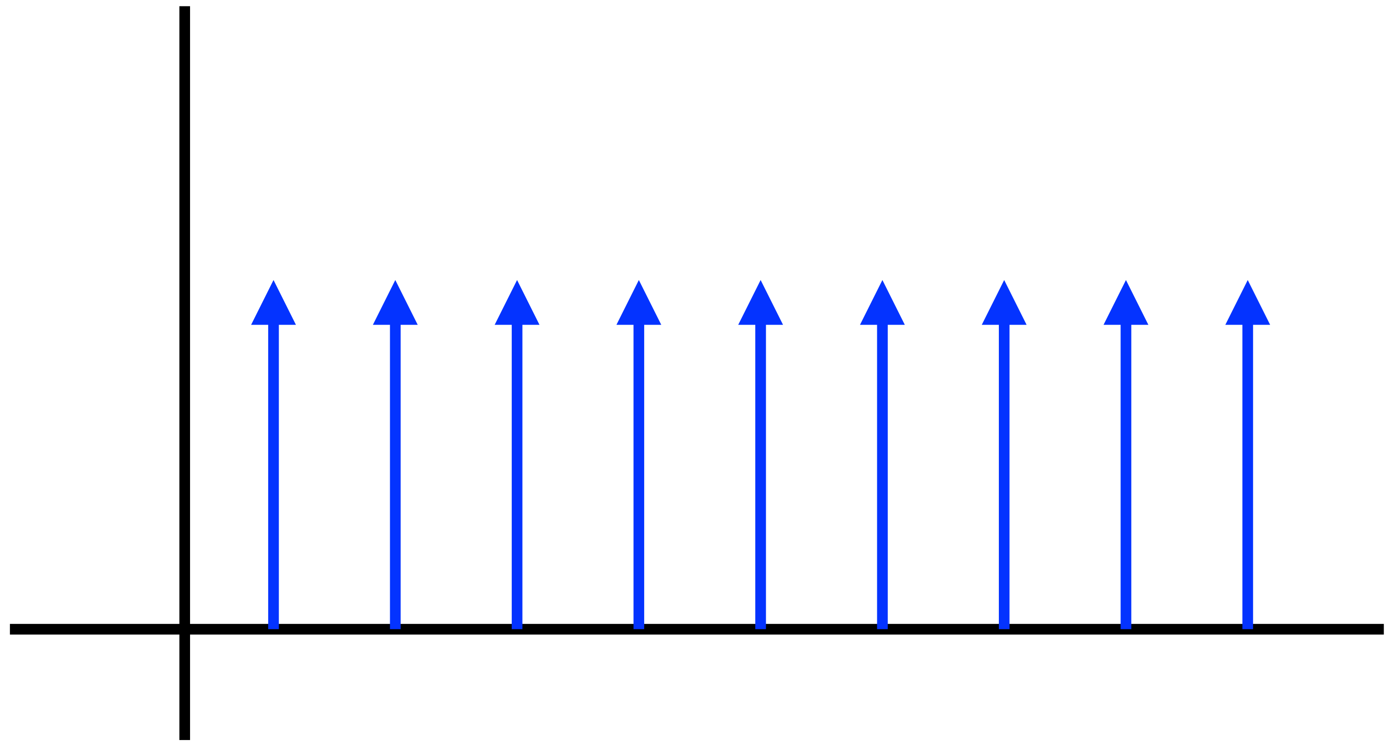
Monte Carlo Estimator in Spatial Domain

$$\tilde{\mu}_N = \int_D f(x)\mathbf{S}(x)dx$$

Monte Carlo Estimator in Spatial Domain

$$\tilde{\mu}_N = \int_D f(x)\mathbf{S}(x)dx$$

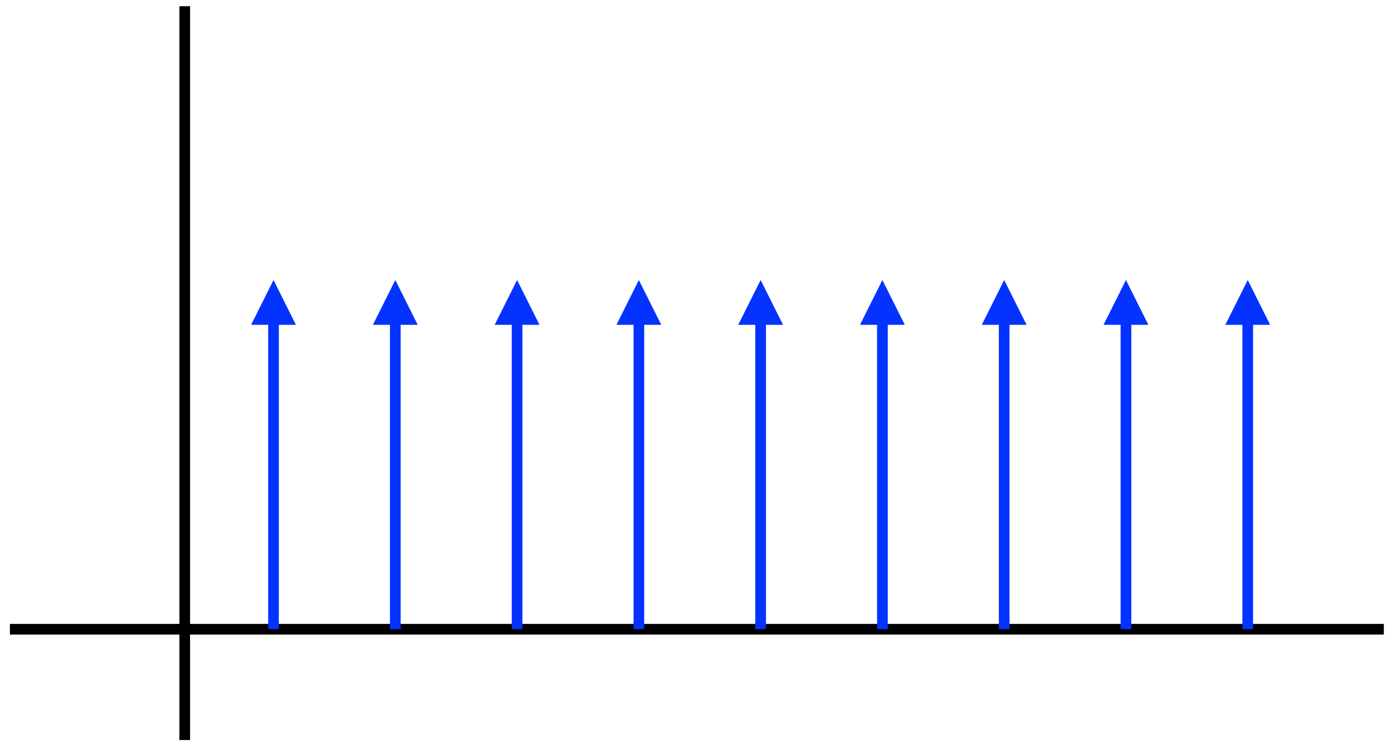
$$\mathbf{S}(x) = \frac{1}{N}\sum_{k=1}^{N}\delta(x - x_k)$$



Monte Carlo Estimator in Spatial Domain

$$\tilde{\mu}_N = \int_D f(x)\mathbf{S}(x)dx$$

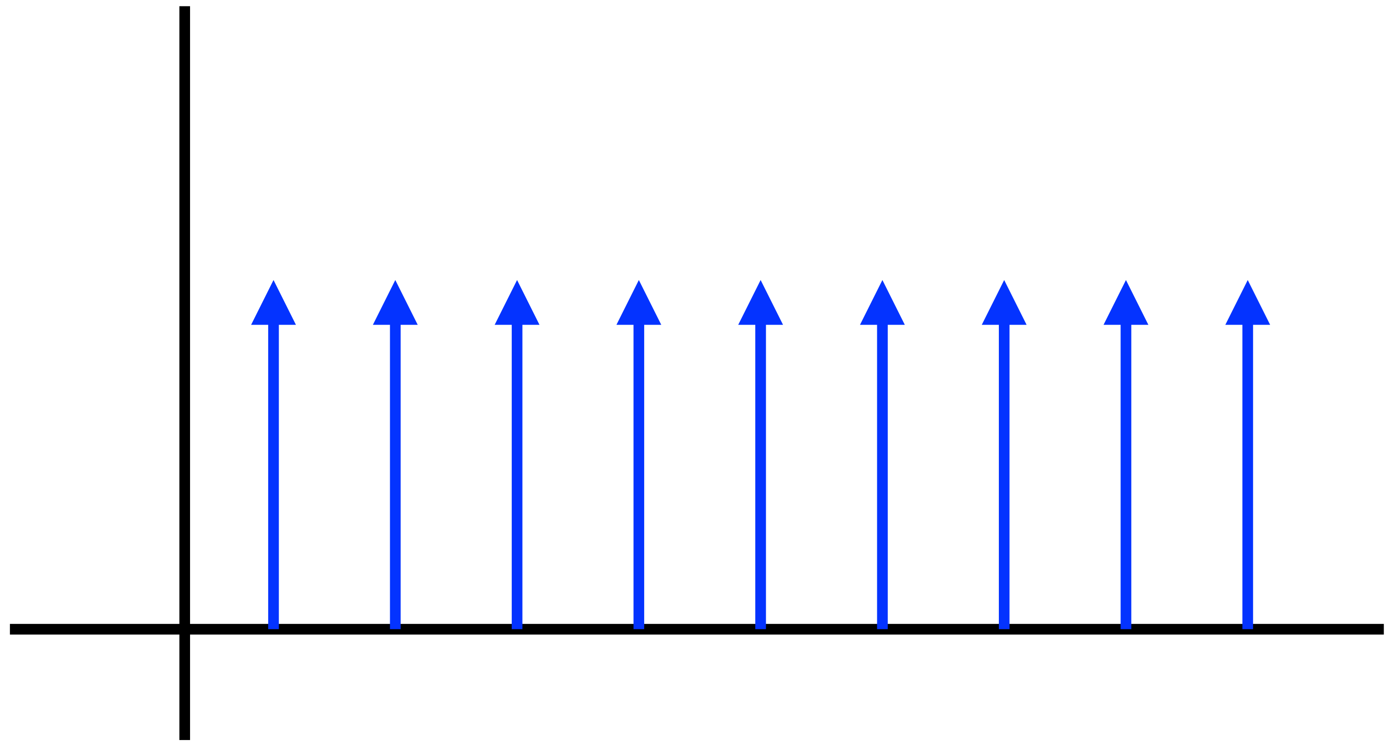
$$\mathbf{S}(x) = \frac{1}{N}\sum_{k=1}^{N}\delta(x - x_k)$$



Monte Carlo Estimator in Spatial Domain

$$\tilde{\mu}_N = \int_D f(x)\mathbf{S}(x)dx$$

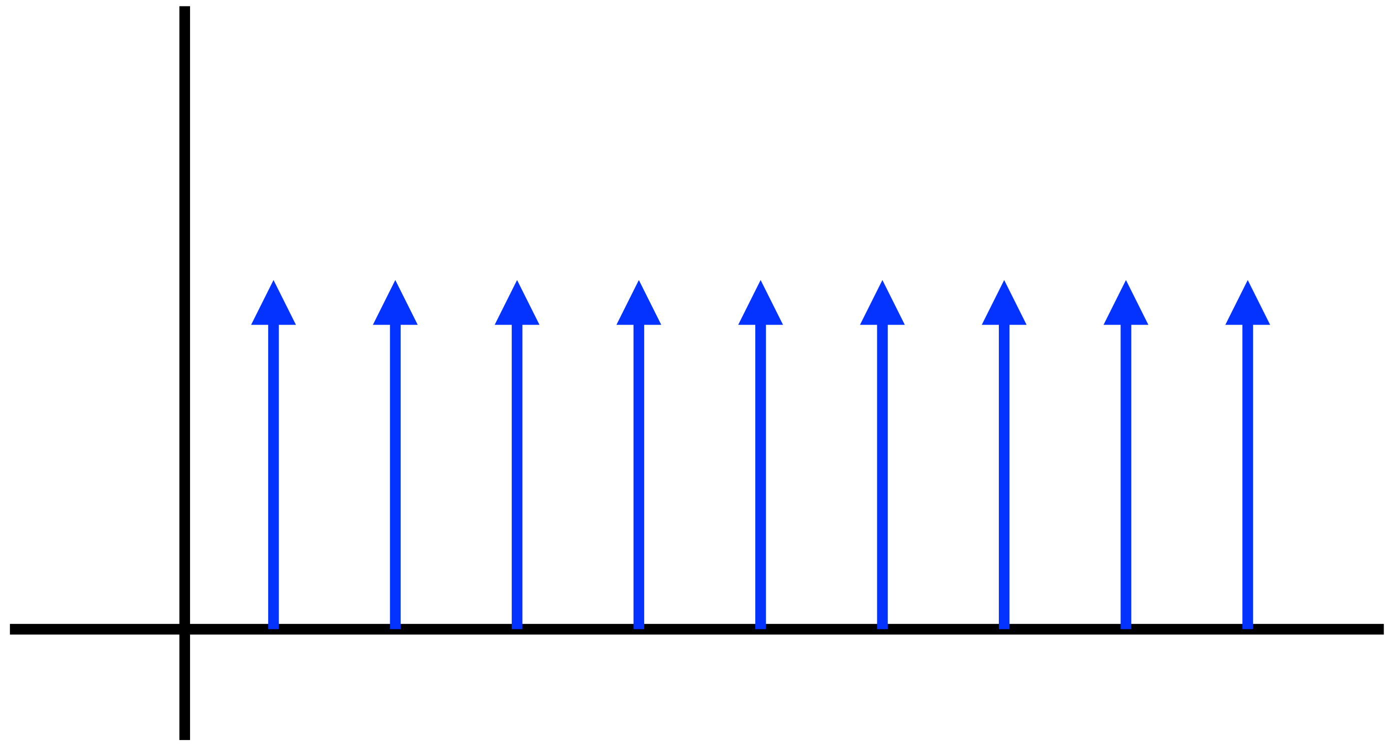
$$\mathbf{S}(x) = \frac{1}{N}\sum_{k=1}^{N}\delta(x - x_k)$$



Monte Carlo Estimator in Spatial Domain

$$\tilde{\mu}_N = \int_D f(x)\mathbf{S}(x)dx$$

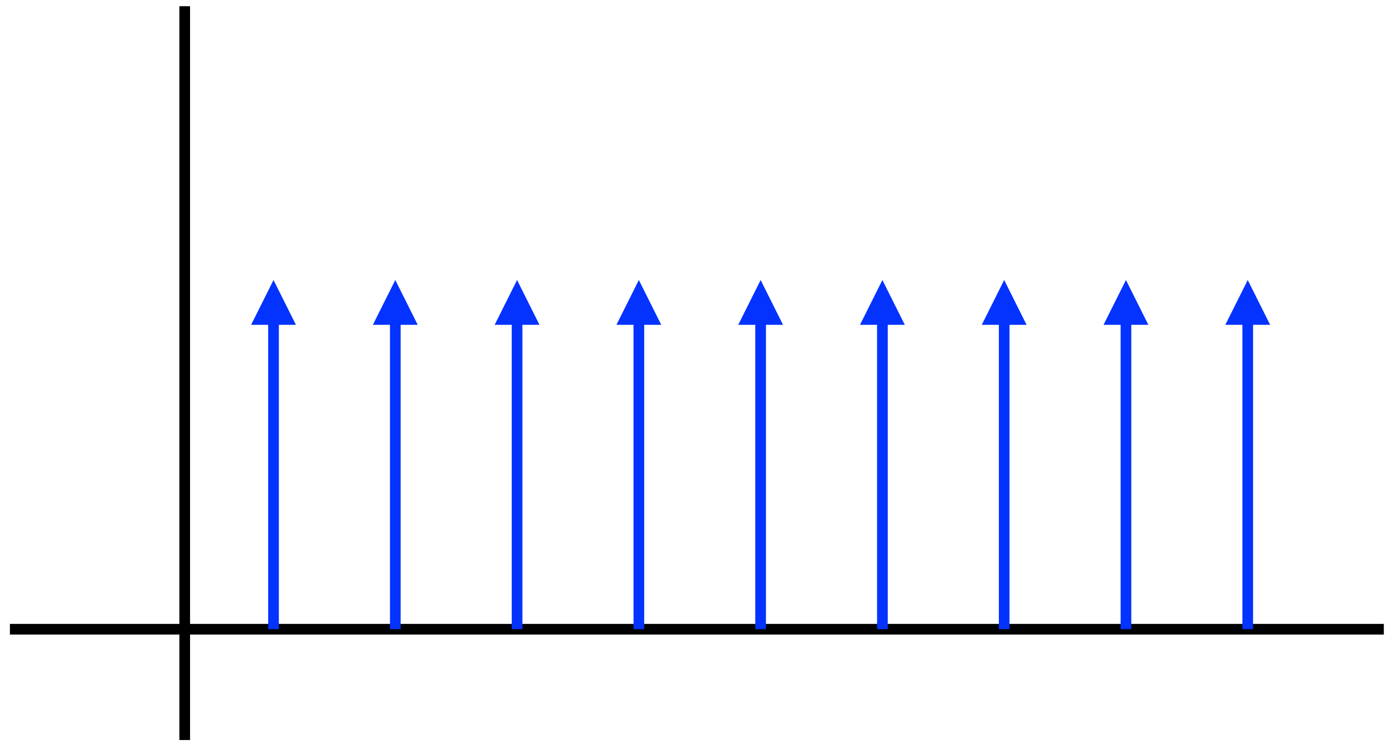
$$\mathbf{S}(x) = \frac{1}{N}\sum_{k=1}^{N}\delta(x - x_k)$$



Monte Carlo Estimator in Spatial Domain

$$\tilde{\mu}_N = \int_D f(x)\mathbf{S}(x)dx = \int_\Omega \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

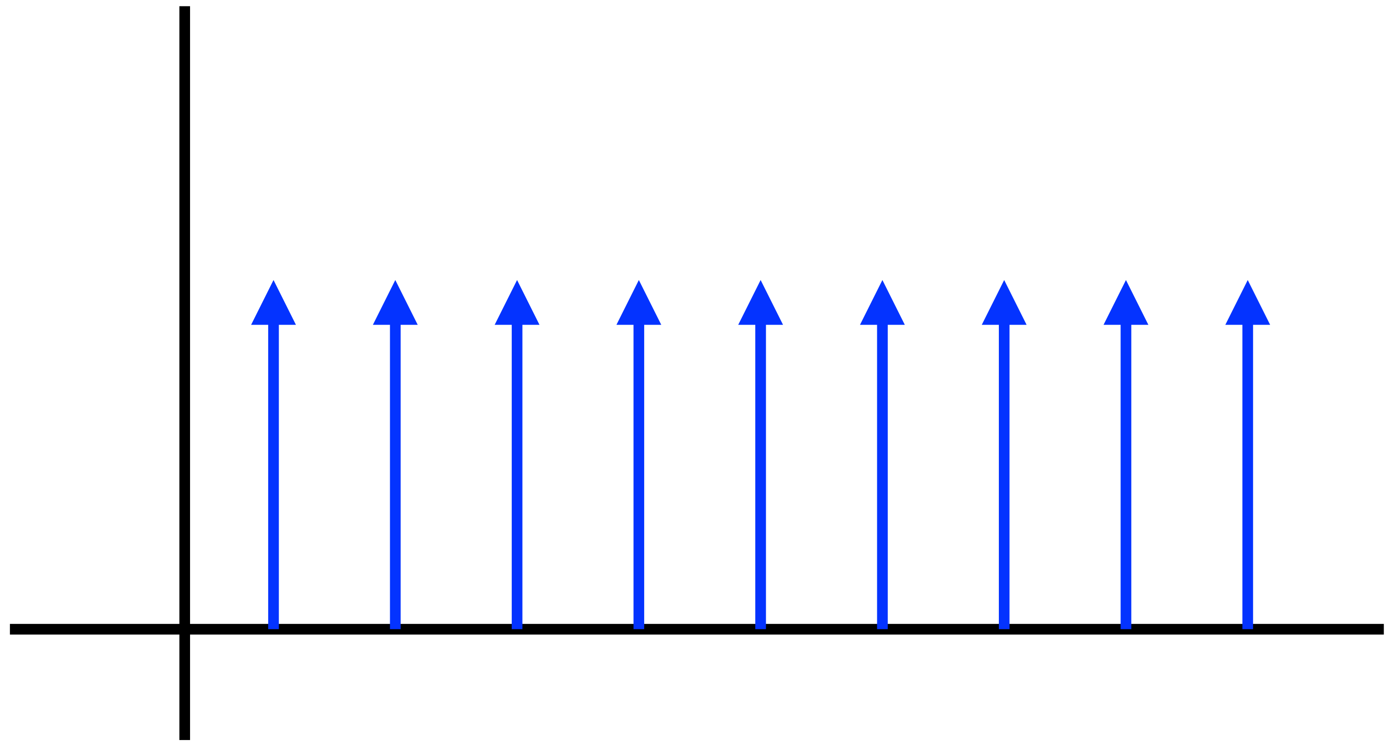
$$\mathbf{S}(x) = \frac{1}{N}\sum_{k=1}^{N}\delta(x - x_k)$$



Monte Carlo Estimator in Fourier Domain

$$\tilde{\mu}_N = \int_D f(x)\mathbf{S}(x)dx = \int_\Omega \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

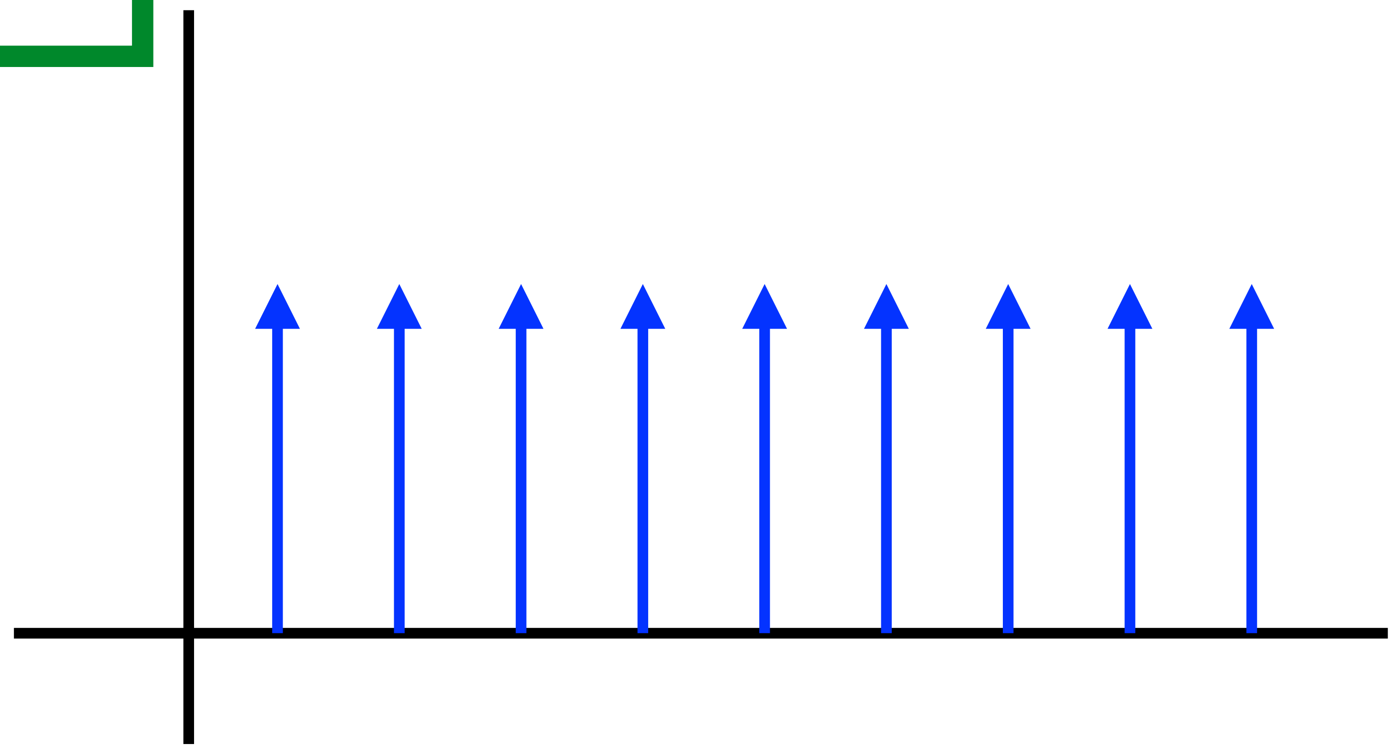
$$\mathbf{S}(x) = \frac{1}{N}\sum_{k=1}^{N}\delta(x - x_k)$$



Monte Carlo Estimator in Fourier Domain

$$\tilde{\mu}_N = \int_D f(x)\mathbf{S}(x)dx = \int_\Omega \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

$$\mathbf{S}(x) = \frac{1}{N}\sum_{k=1}^{N}\delta(x - x_k)$$

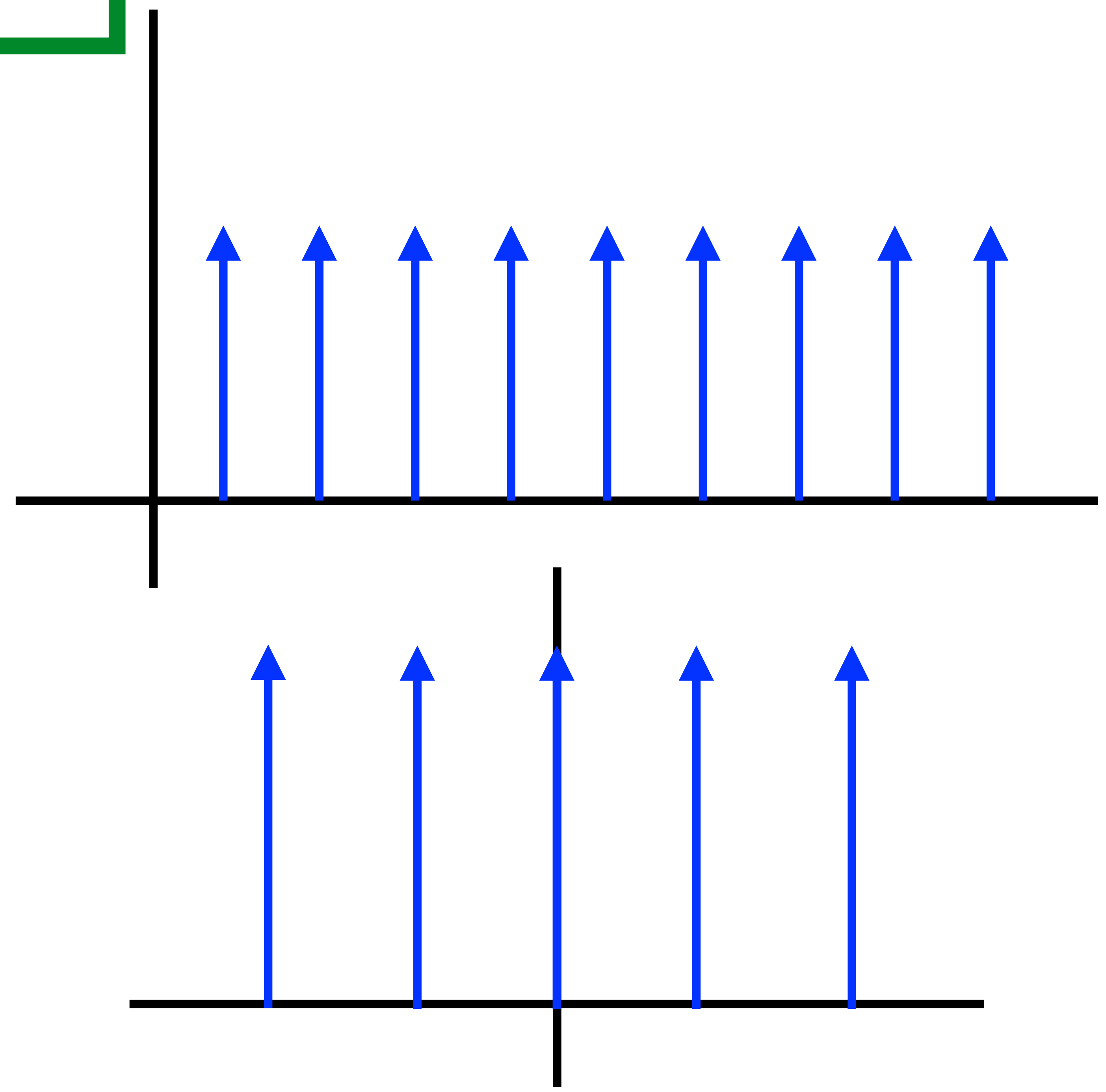


Monte Carlo Estimator in Fourier Domain

$$\tilde{\mu}_N = \int_D f(x)\mathbf{S}(x)dx = \int_\Omega \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

$$\mathbf{S}(x) = \frac{1}{N}\sum_{k=1}^{N}\delta(x - x_k)$$

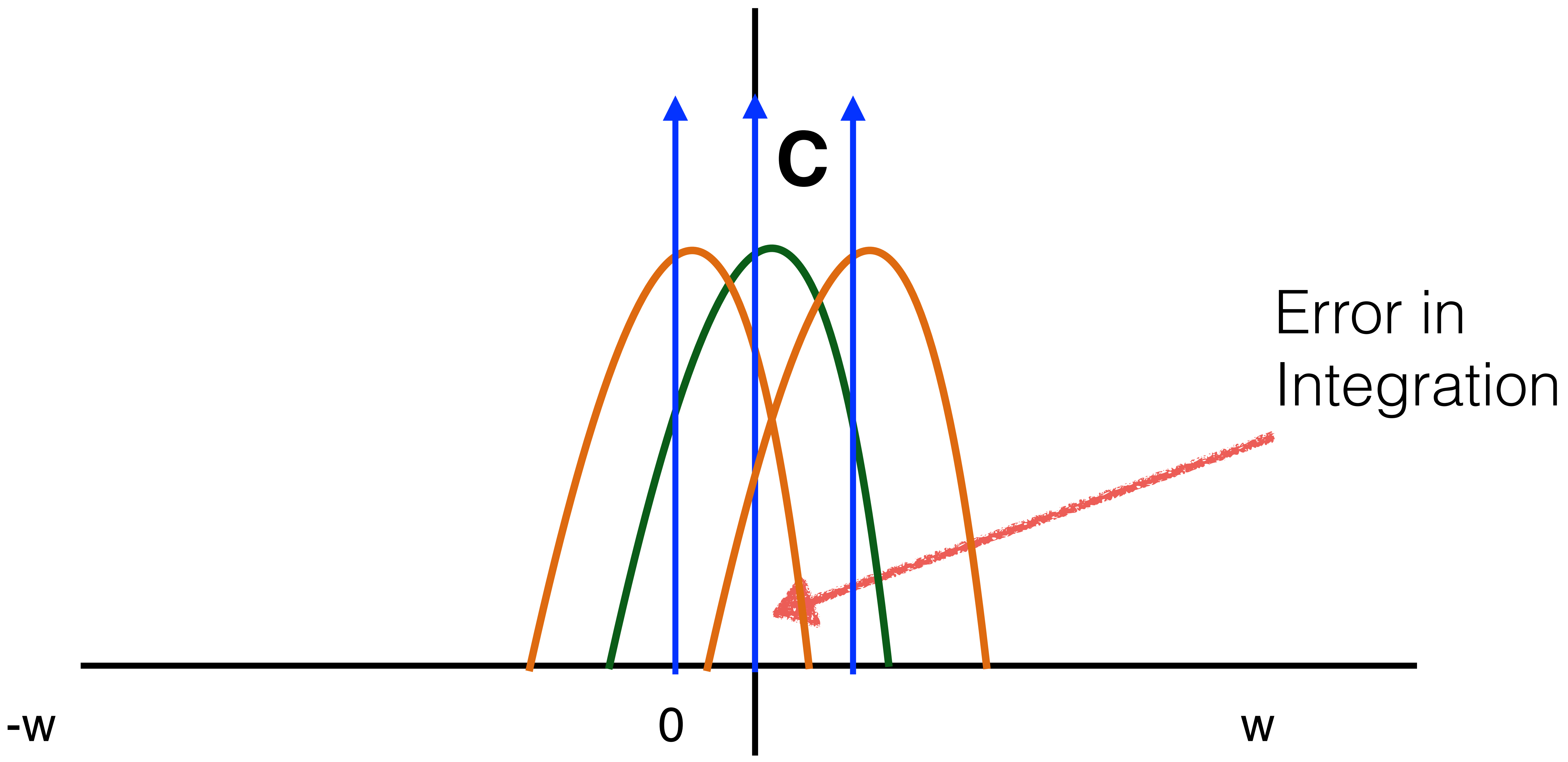
$$\hat{\mathbf{S}}(\omega) = \frac{1}{N}\sum_{k=1}^{N}e^{-i2\pi\omega x_k}$$



How to Formulate Error in Fourier Domain ?

$$I = \hat{f}(0)$$

$$\tilde{\mu}_N = \int_{\Omega} \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

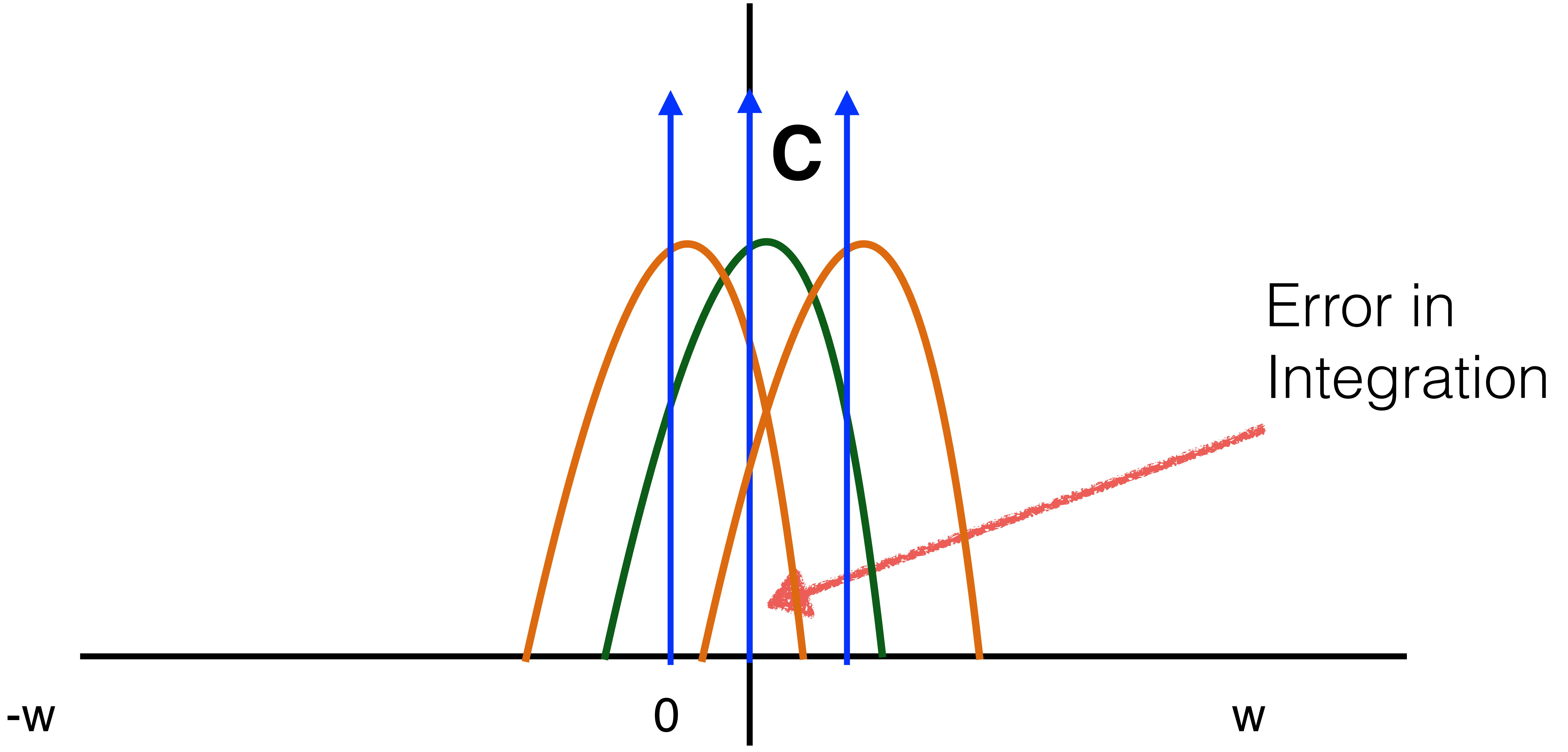


Fredo Durand [2011]

How to Formulate Error in Fourier Domain ?

$$I = \hat{f}(0)$$

$$\tilde{\mu}_N = \int_\Omega \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$



Fredo Durand [2011]

Error in Spatial Domain

$$I = \hat{f}(0)$$

$$\tilde{\mu}_N = \int_{\Omega} \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

$$I - \tilde{\mu}_N = \int_D f(x)dx - \int_D f(x)\mathbf{S}(x)dx$$

Error in Spatial Domain

$$I = \hat{f}(0)$$

$$\tilde{\mu}_N = \int_{\Omega} \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

True Integral

$$I - \tilde{\mu}_N = \boxed{\int_D f(x)dx} - \int_D f(x)\mathbf{S}(x)dx$$

Error in Spatial Domain

$$I = \hat{f}(0)$$

$$\tilde{\mu}_N = \int_{\Omega} \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

True Integral

$$I - \tilde{\mu}_N = \int_D f(x)dx - \int_D f(x)\mathbf{S}(x)dx$$

Monte Carlo Estimator

Error in Spatial Domain

$$I = \hat{f}(0)$$

$$\tilde{\mu}_N = \int_{\Omega} \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

$$I - \tilde{\mu}_N = \int_D f(x)dx - \int_D f(x)\mathbf{S}(x)dx$$

Error in Spatial Domain

$$I = \hat{f}(0)$$

$$\tilde{\mu}_N = \int_{\Omega} \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

$$I - \tilde{\mu}_N = \int_D f(x)dx - \int_D f(x)\mathbf{S}(x)dx$$

Error in Fourier Domain

$$I = \hat{f}(0)$$

$$\tilde{\mu}_N = \int_\Omega \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

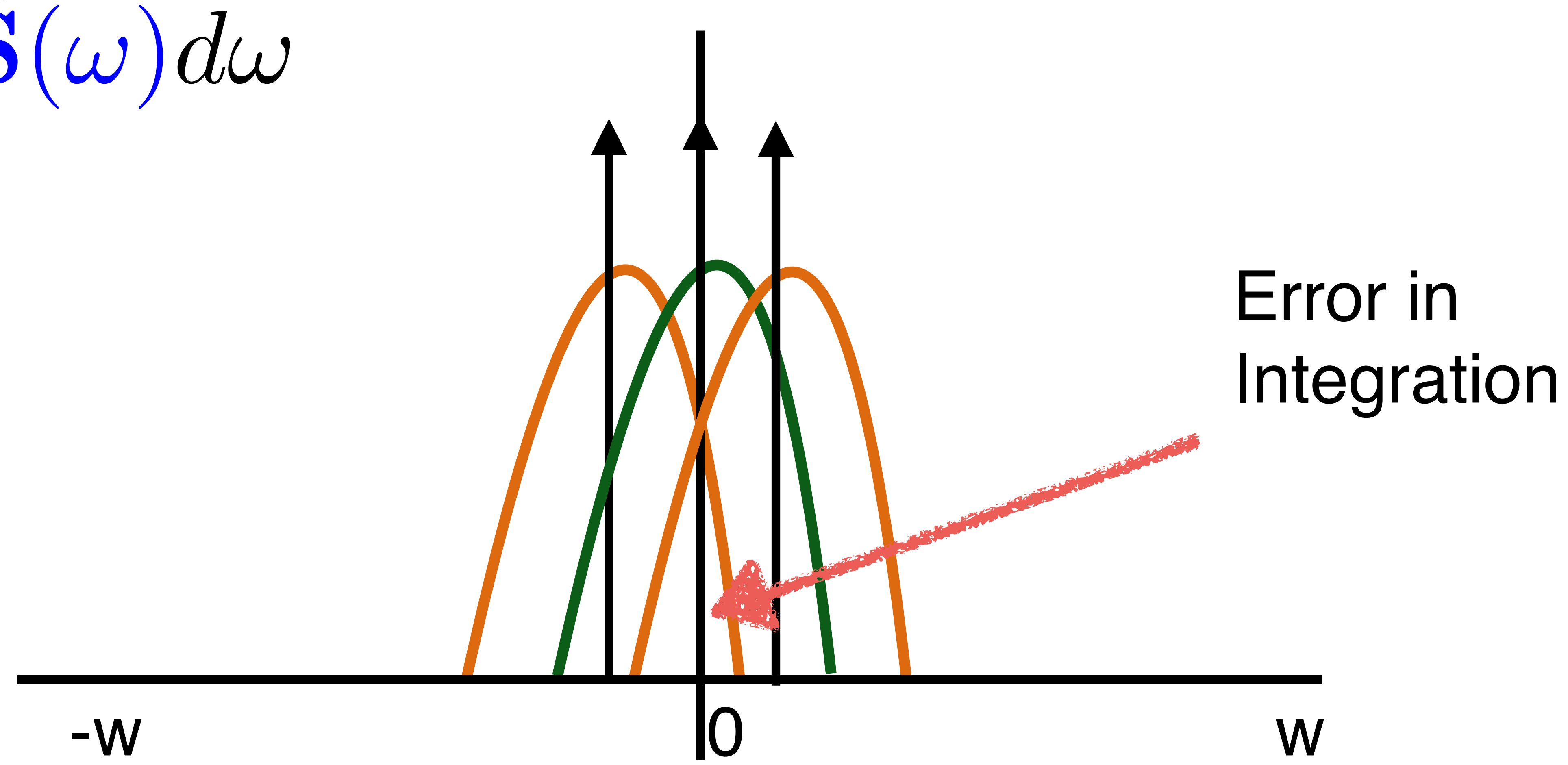
$$I - \tilde{\mu}_N = \int_D f(x)dx - \int_D f(x)\mathbf{S}(x)dx$$

$$I - \tilde{\mu}_N = \hat{f}(0) - \int_\Omega \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

Fredo Durand [2011]

Error in Fourier Domain

$$I - \tilde{\mu}_N = \hat{f}(0) - \int_{\Omega} \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$



Fredo Durand [2011]

$$\text{Error} = \text{Bias}^2 + \text{Variance}$$

Properties of Error

- Bias
- Variance

Properties of Error

- Bias: Expected value of the Error
- Variance

Properties of Error

- Bias: Expected value of the Error $\langle I - \tilde{\mu}_N \rangle$
- Variance

Properties of Error

- Bias: Expected value of the Error $\langle I - \tilde{\mu}_N \rangle$
- Variance: $\mathrm{Var}(I - \mu_N)$

Subr and Kautz [2013]

Bias in the Monte Carlo Estimator

Bias in Fourier Domain

Error:

$$I - \tilde{\mu}_N = \hat{f}(0) - \int_{\Omega} \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

Bias in Fourier Domain

Error: $$I - \tilde{\mu}_N = \hat{f}(0) - \int_{\Omega} \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

Bias in Fourier Domain

Error: $$I - \tilde{\mu}_N = \hat{f}(0) - \int_{\Omega} \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

Bias: $$\langle I - \tilde{\mu}_N \rangle$$

Bias in Fourier Domain

Bias: $$\langle I - \tilde{\mu}_N \rangle = \hat{f}(0) - \left\langle \int_{\Omega} \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega) d\omega \right\rangle$$

Bias in Fourier Domain

$$\langle I - \tilde{\mu}_N \rangle = \hat{f}(0) - \left\langle \int_\Omega \hat{f}^*(\omega) \hat{\mathbf{S}}(\omega) d\omega \right\rangle$$

Bias in Fourier Domain

$$\langle I - \tilde{\mu}_N \rangle = \hat{f}(0) - \left\langle \int_\Omega \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega \right\rangle$$

Bias in Fourier Domain

$$\langle I - \tilde{\mu}_N \rangle = \hat{f}(0) - \left\langle \int_\Omega \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega) d\omega \right\rangle$$

$$\langle I - \tilde{\mu}_N \rangle = \hat{f}(0) - \int_\Omega \hat{f}^*(\omega) \left\langle \hat{\mathbf{S}}(\omega) \right\rangle d\omega$$

Subr and Kautz [2013]

Bias in Fourier Domain

$$\langle I - \tilde{\mu}_N \rangle = \hat{f}(0) - \int_\Omega \hat{f}^*(\omega) \, \langle \hat{\mathbf{S}}(\omega) \rangle \, d\omega$$

Subr and Kautz [2013]

Bias in Fourier Domain

$$\langle I - \tilde{\mu}_N \rangle = \hat{f}(0) - \boxed{\int_\Omega \hat{f}^*(\omega) \, \langle \hat{\mathbf{S}}(\omega) \rangle \, d\omega}$$

Subr and Kautz [2013]

Bias in Fourier Domain

$$\langle I - \tilde{\mu}_N \rangle = \hat{f}(0) - \boxed{\int_{\Omega} \hat{f}^*(\omega) \, \langle \hat{\mathbf{S}}(\omega) \rangle \, d\omega}$$

To obtain an unbiased estimator:

Subr and Kautz [2013]

Bias in Fourier Domain

$$\langle I - \tilde{\mu}_N \rangle = \hat{f}(0) - \boxed{\int_{\Omega} \hat{f}^*(\omega) \, \langle \hat{\mathbf{S}}(\omega) \rangle \, d\omega}$$

To obtain an unbiased estimator:

Subr and Kautz [2013]

$$\langle \hat{\mathbf{S}}(\omega) \rangle = 0$$

for frequencies other than zero

How to obtain $\langle \hat{\mathbf{S}}(\omega) \rangle = 0$?

Complex form in Amplitude and Phase

$$\langle \hat{\mathbf{S}}(\omega) \rangle = |\langle \hat{\mathbf{S}}(\omega) \rangle| \, e^{-\Phi(\langle \hat{\mathbf{S}}(\omega) \rangle)}$$

Complex form in Amplitude and Phase

Amplitude

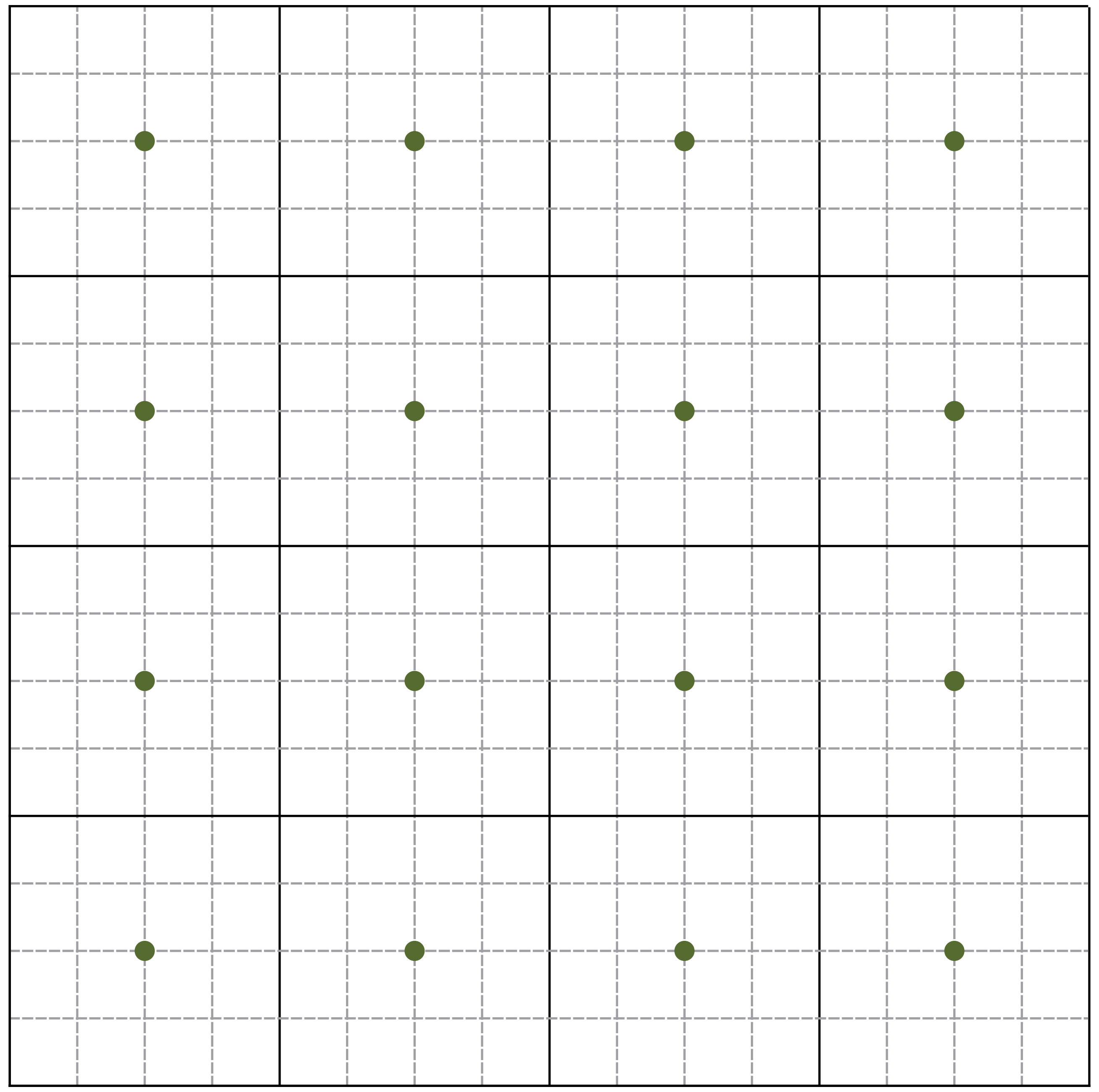
$$\langle \hat{\mathbf{S}}(\omega) \rangle = \boxed{|\langle \hat{\mathbf{S}}(\omega) \rangle|} e^{-\Phi(\langle \hat{\mathbf{S}}(\omega) \rangle)}$$

Complex form in Amplitude and Phase

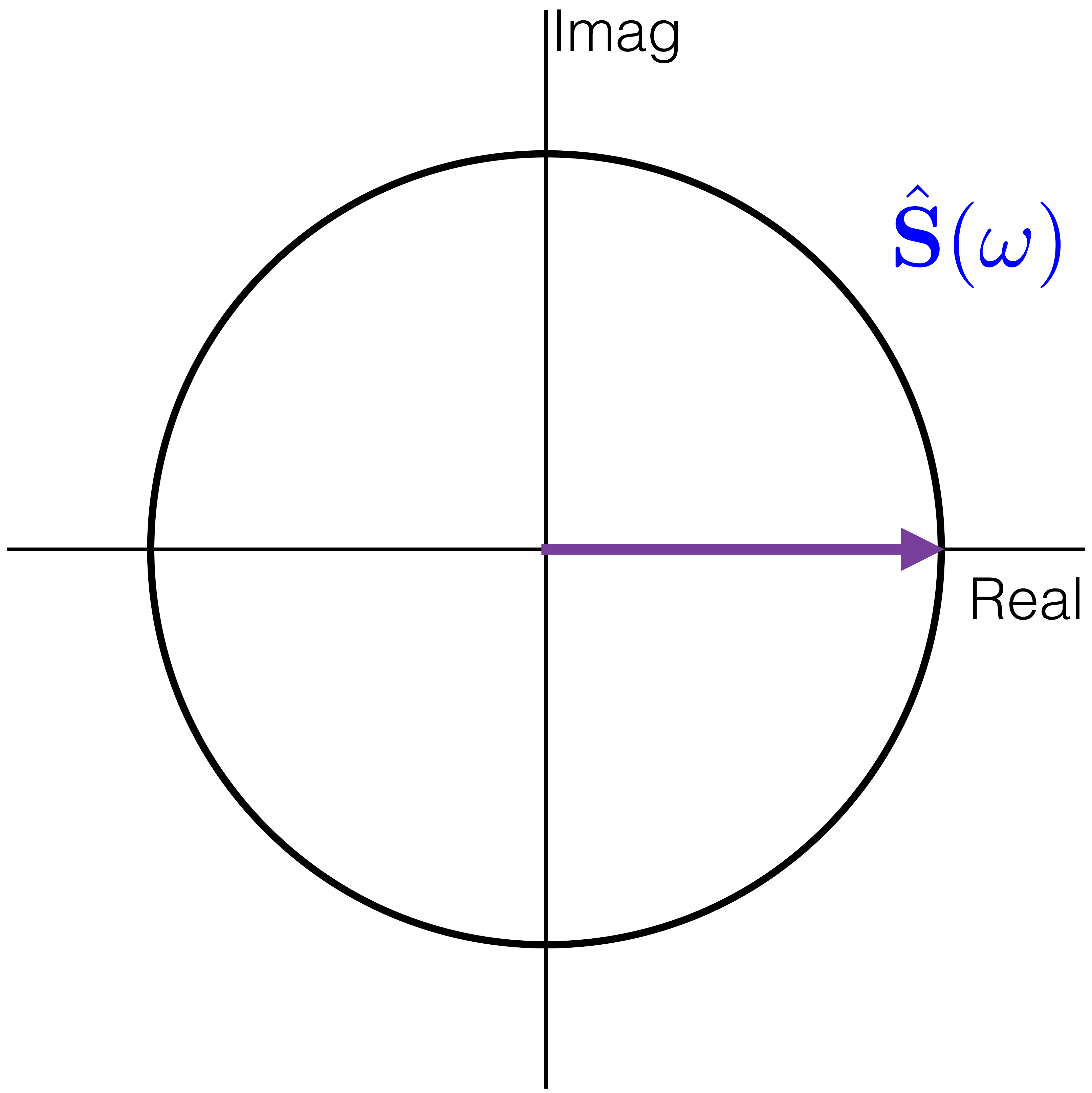
Amplitude Phase

$$\langle \hat{\mathbf{S}}(\omega) \rangle = |\langle \hat{\mathbf{S}}(\omega) \rangle| e^{-\Phi(\langle \hat{\mathbf{S}}(\omega) \rangle)}$$

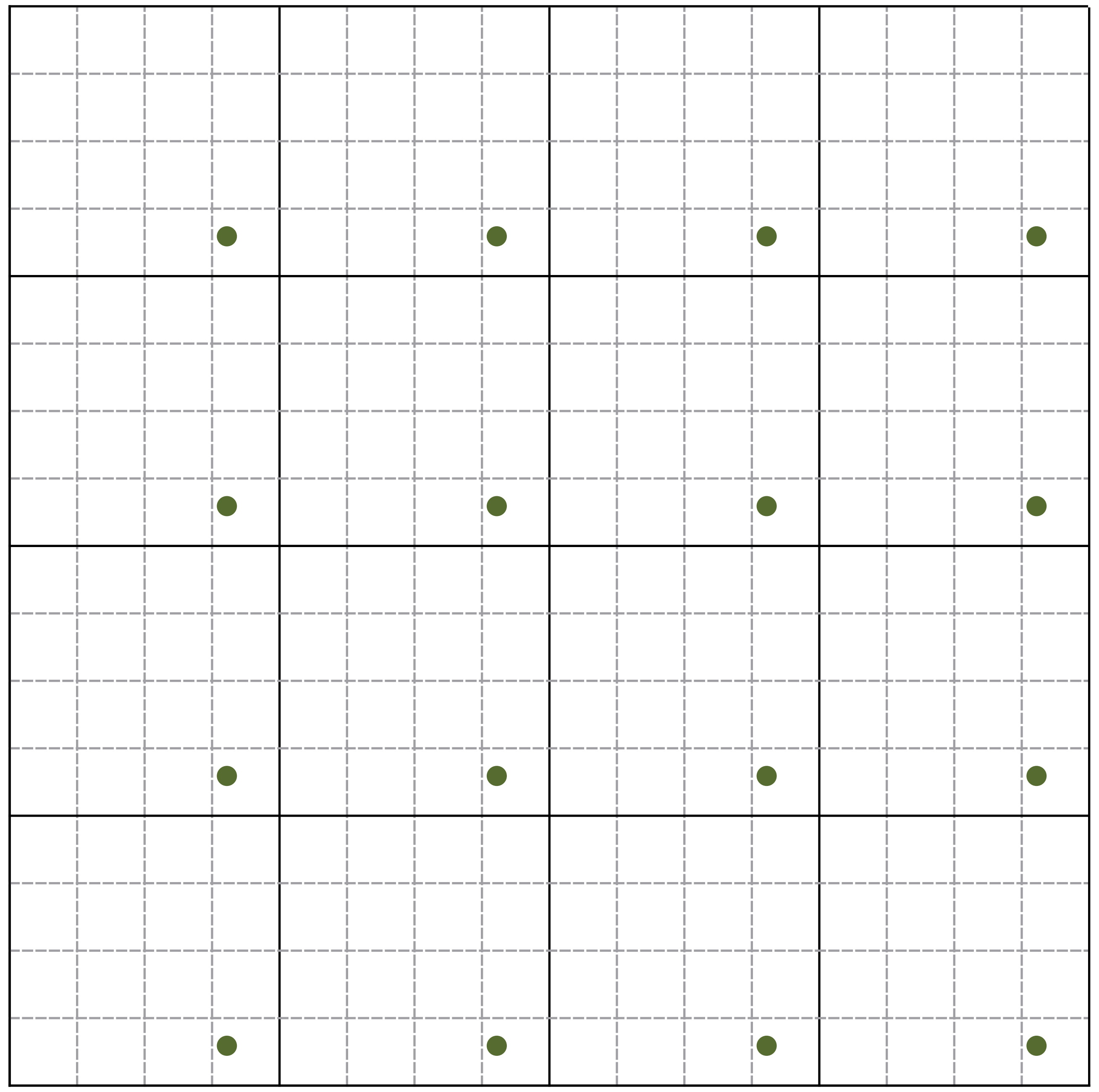
Phase change due to Random Shift



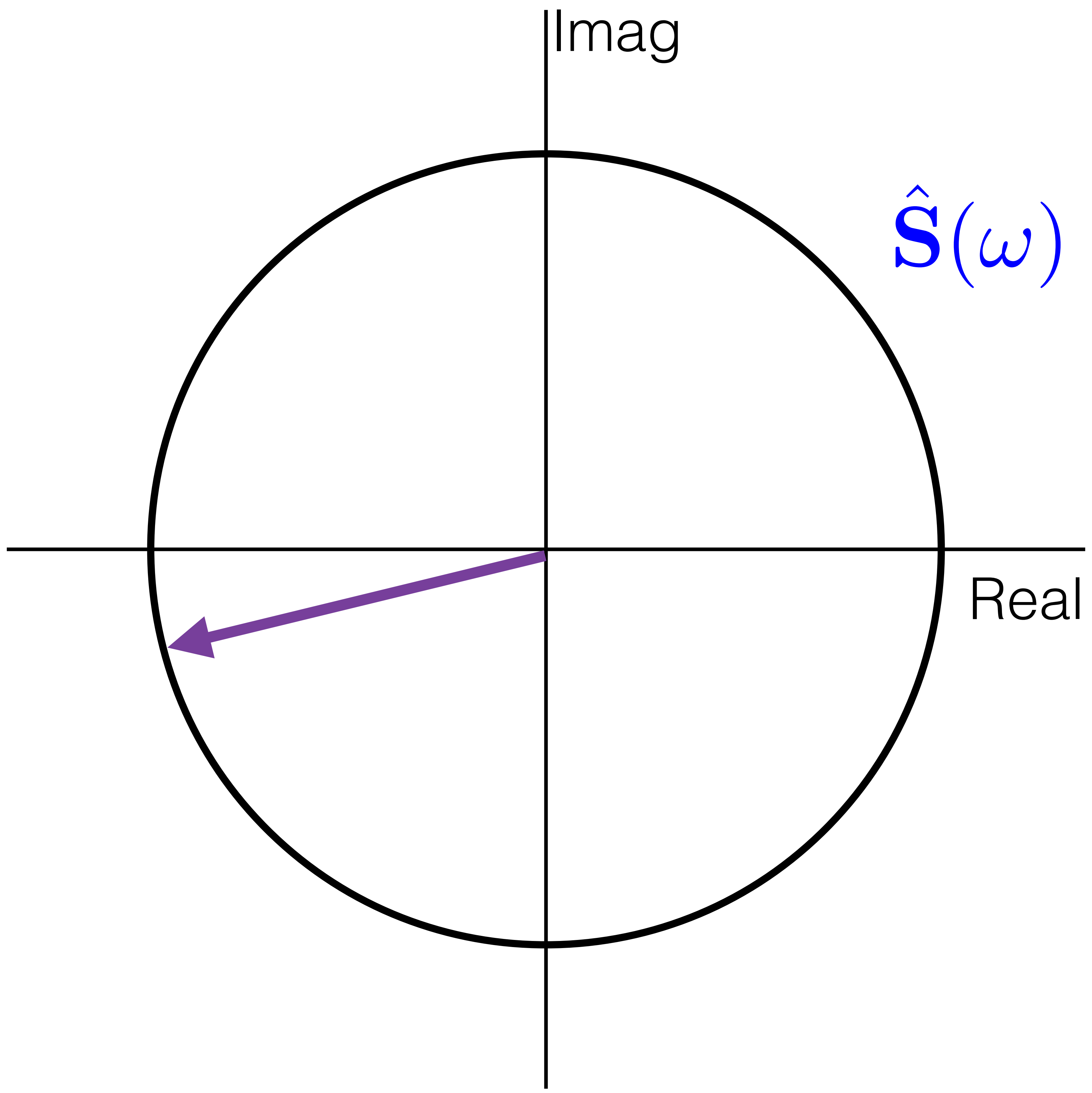
For a given frequency ω



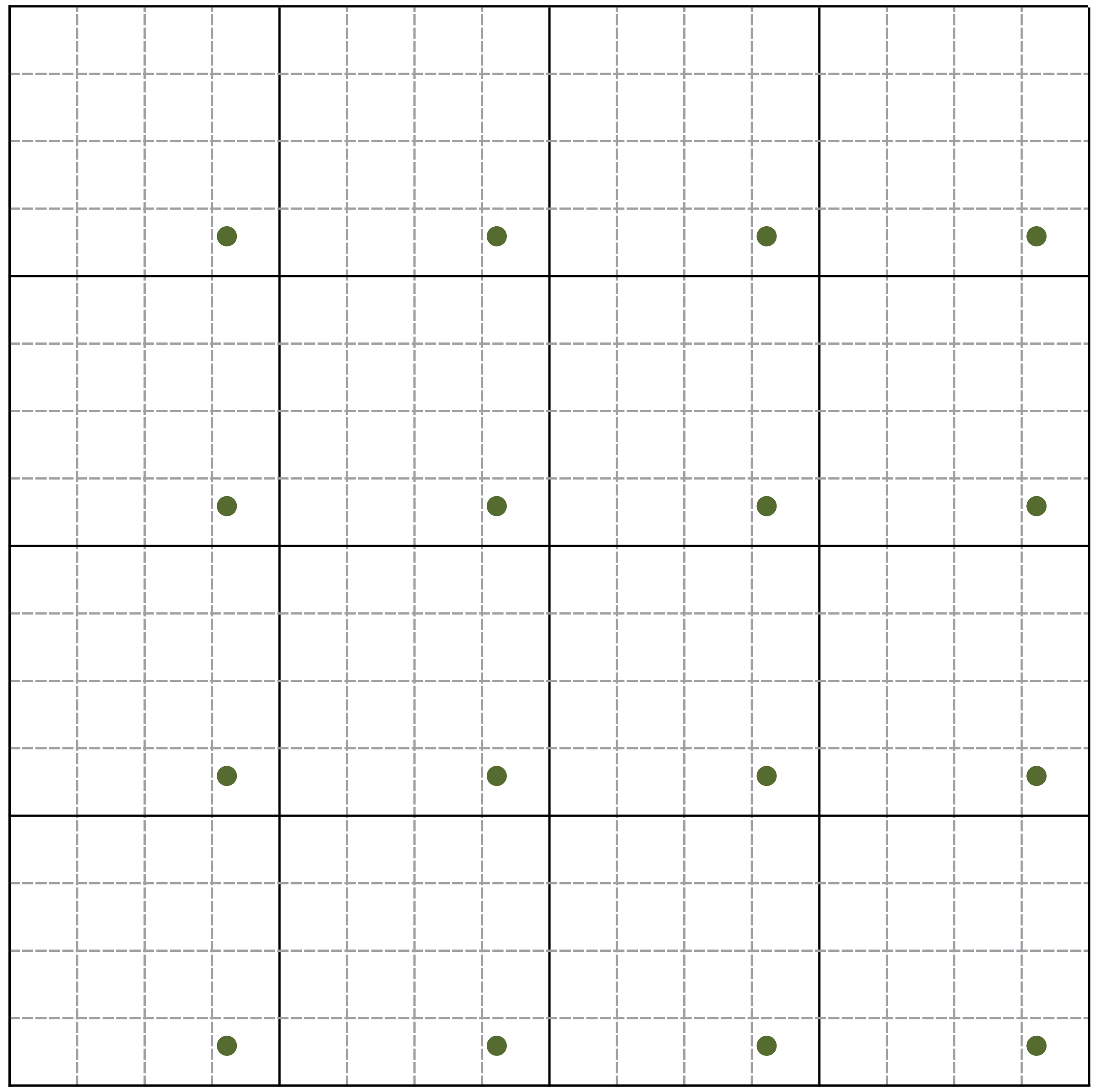
Phase change due to Random Shift



For a given frequency ω

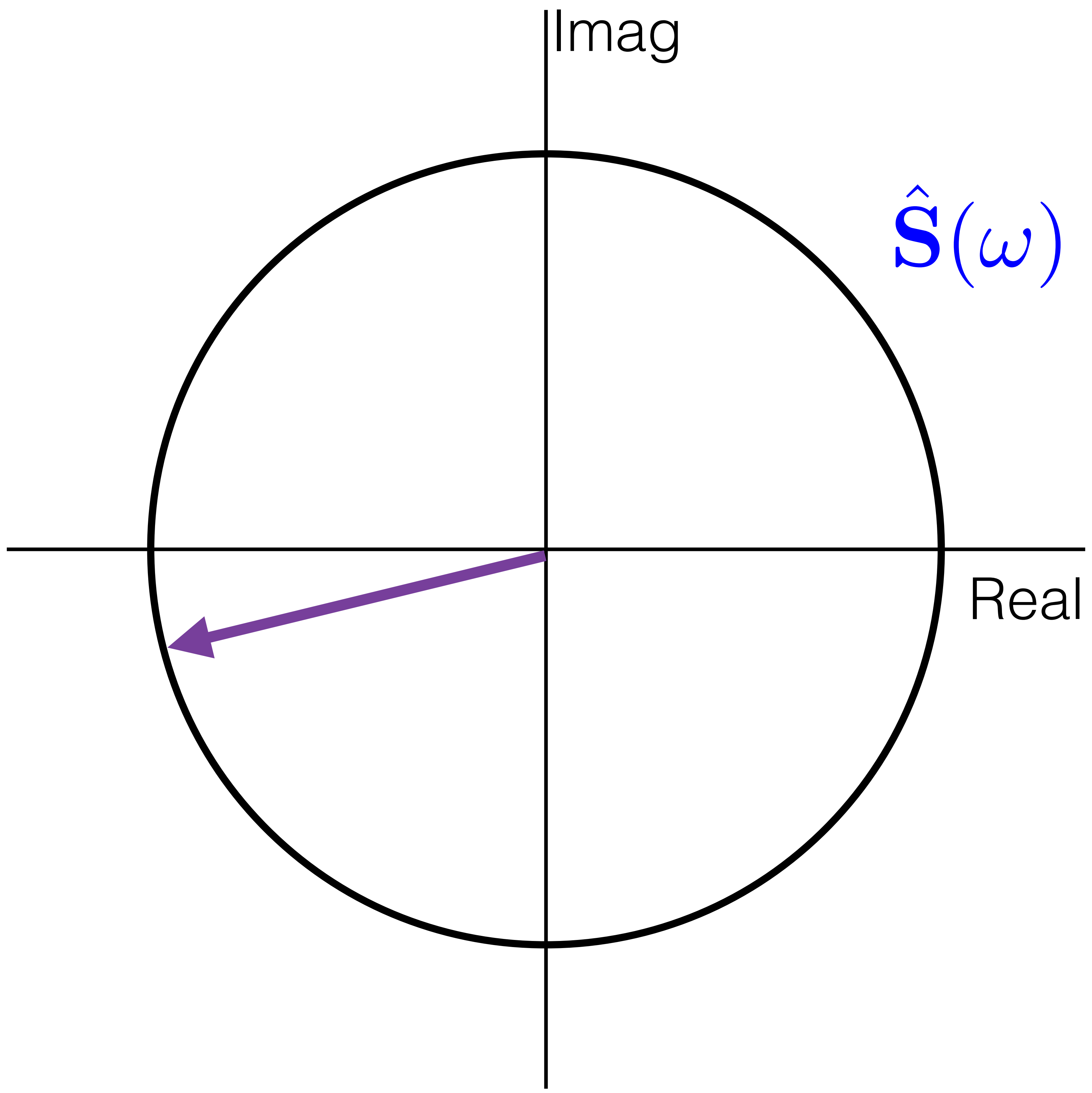


Phase change due to Random Shift

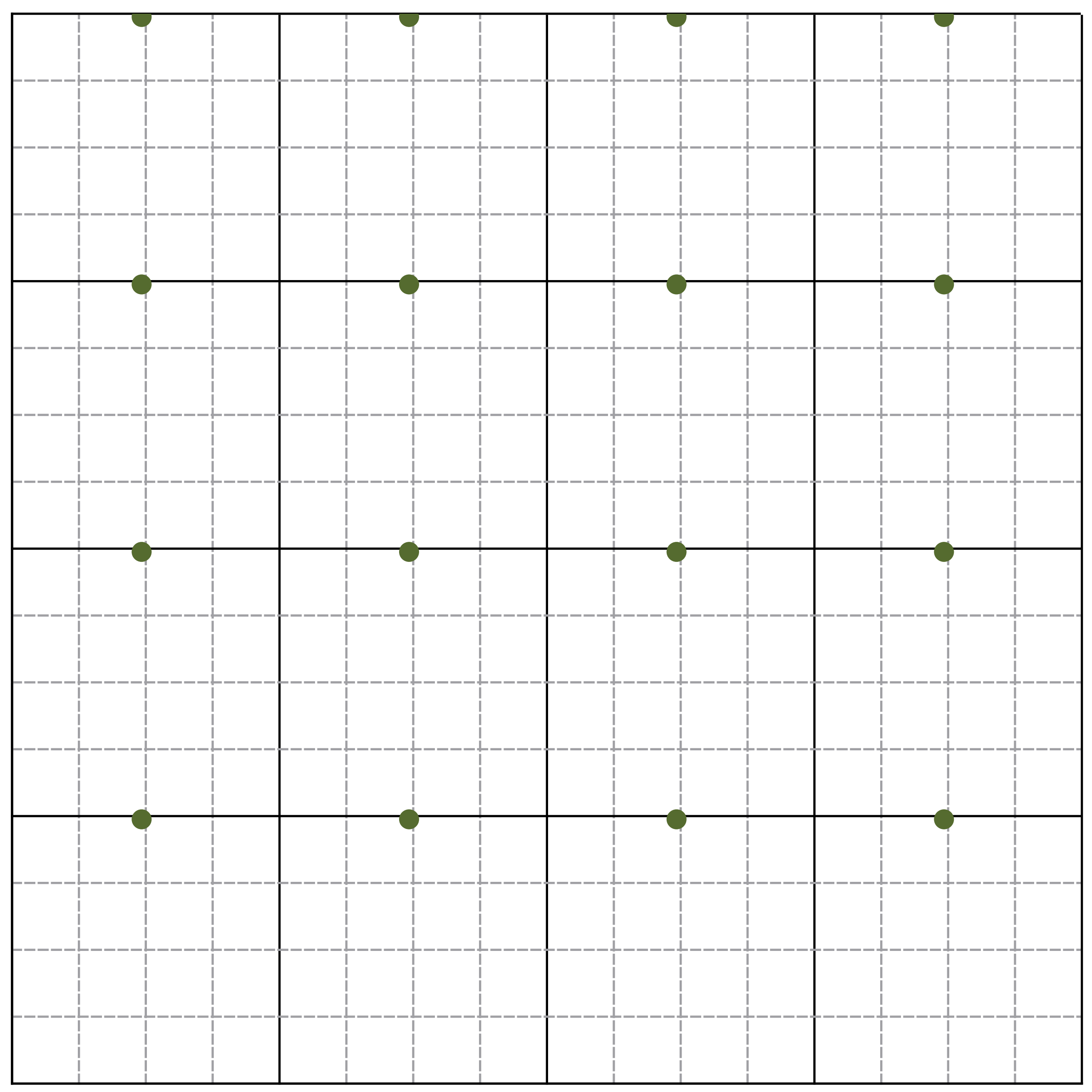


Pauly et al. [2000]
Ramamoorthi et al. [2012]

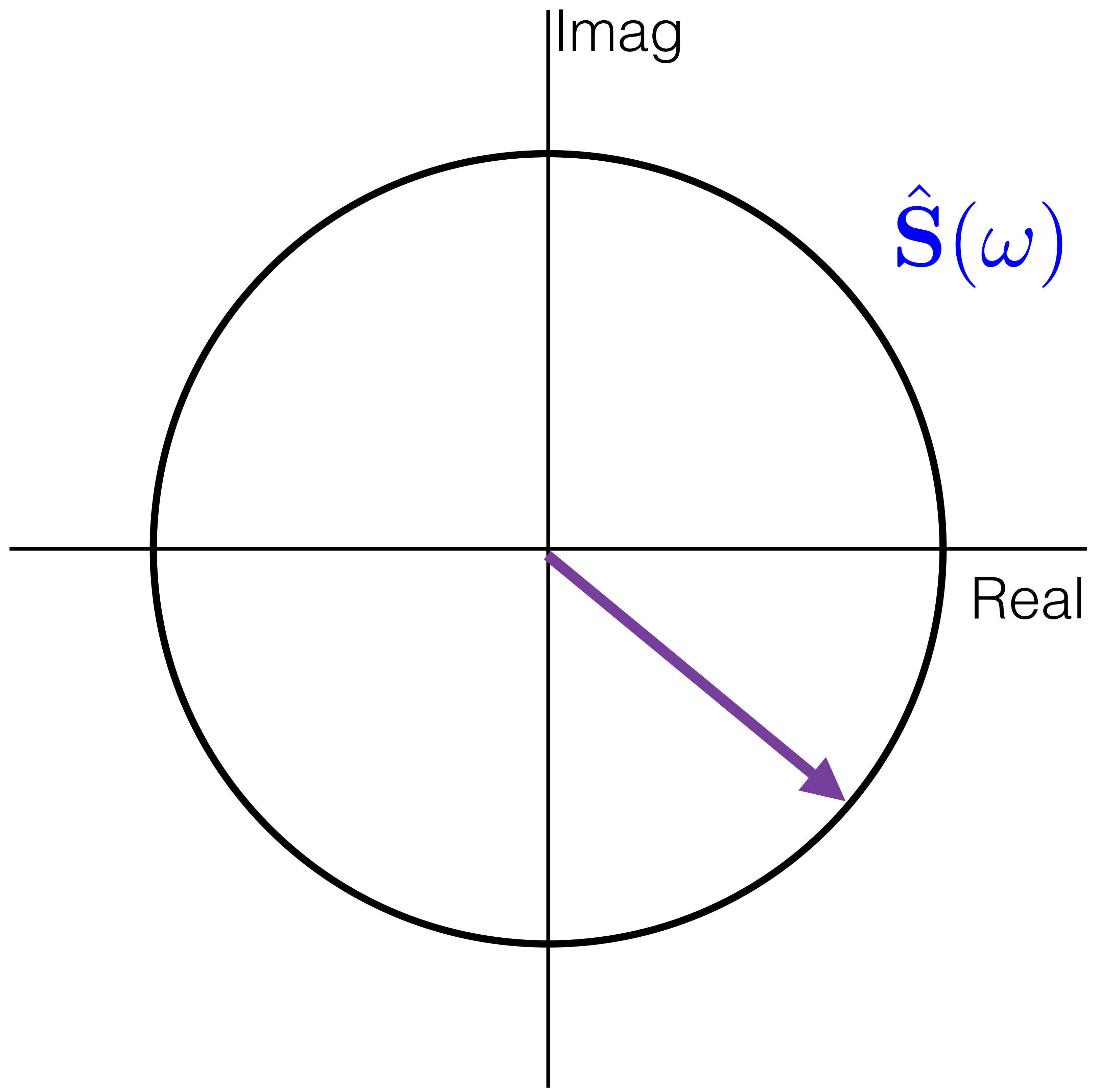
For a given frequency ω



Phase change due to Random Shift



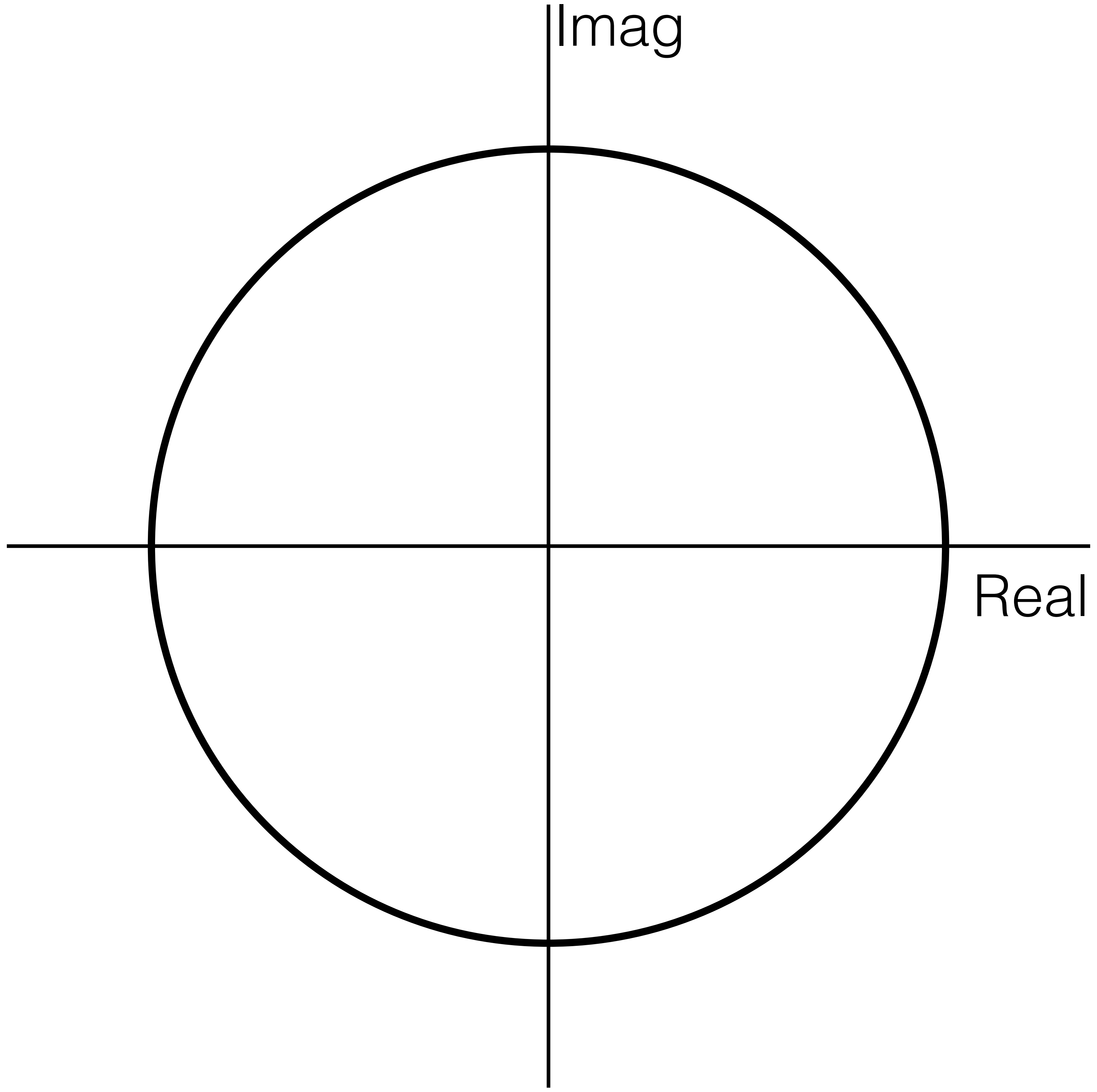
For a given frequency ω



Phase change due to Random Shift

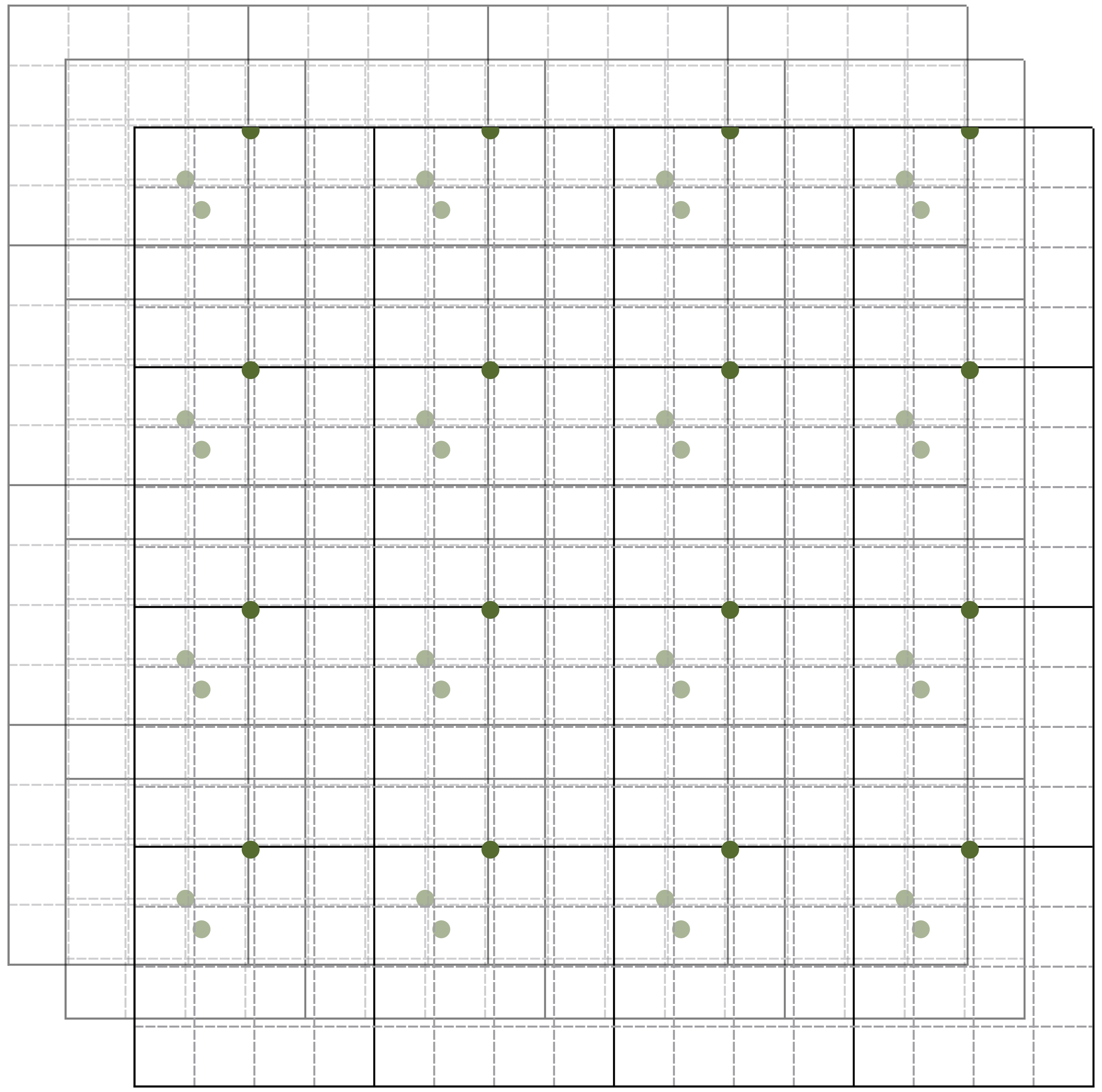
Multiple realizations

For a given frequency ω

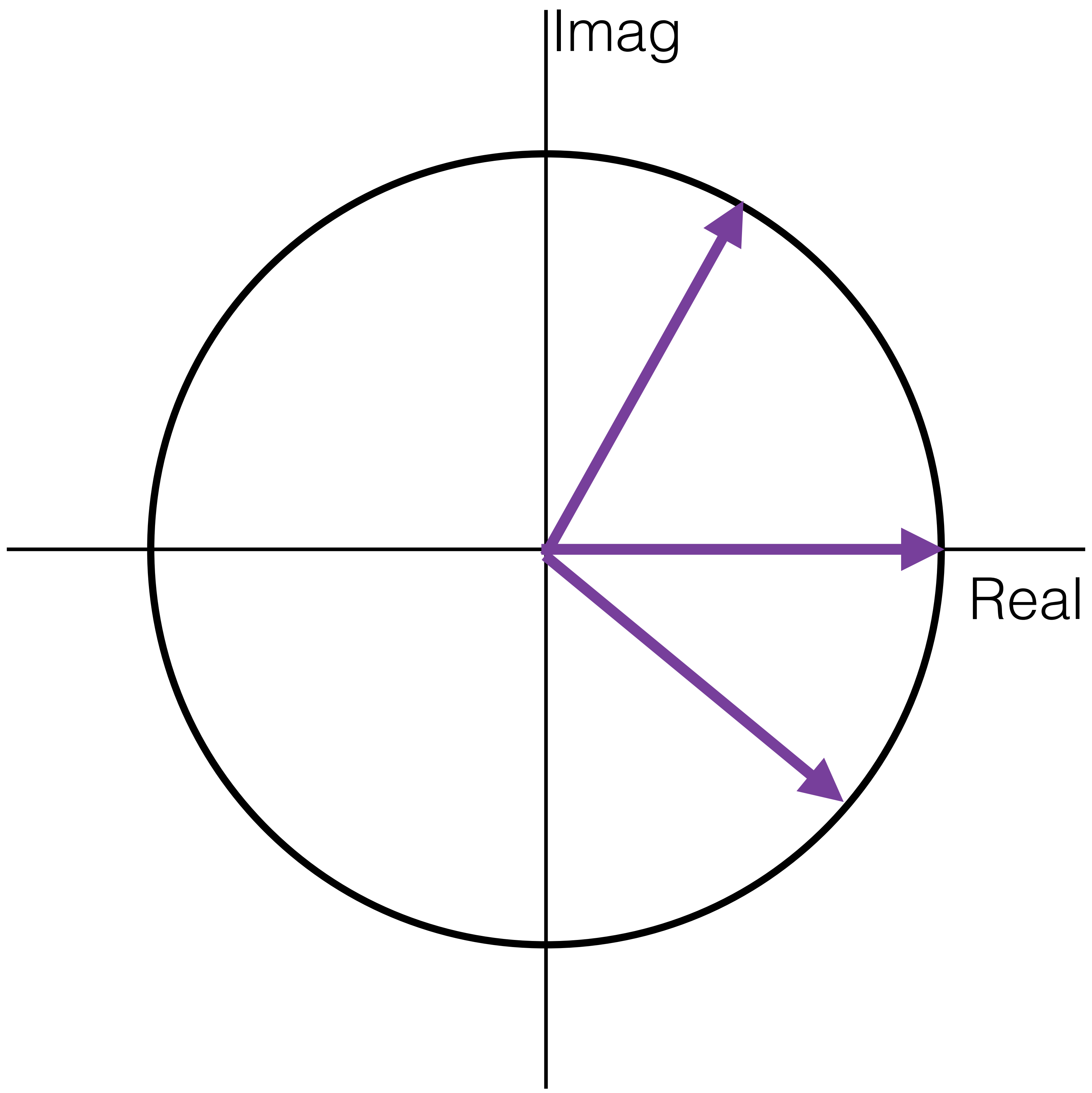


Phase change due to Random Shift

Multiple realizations

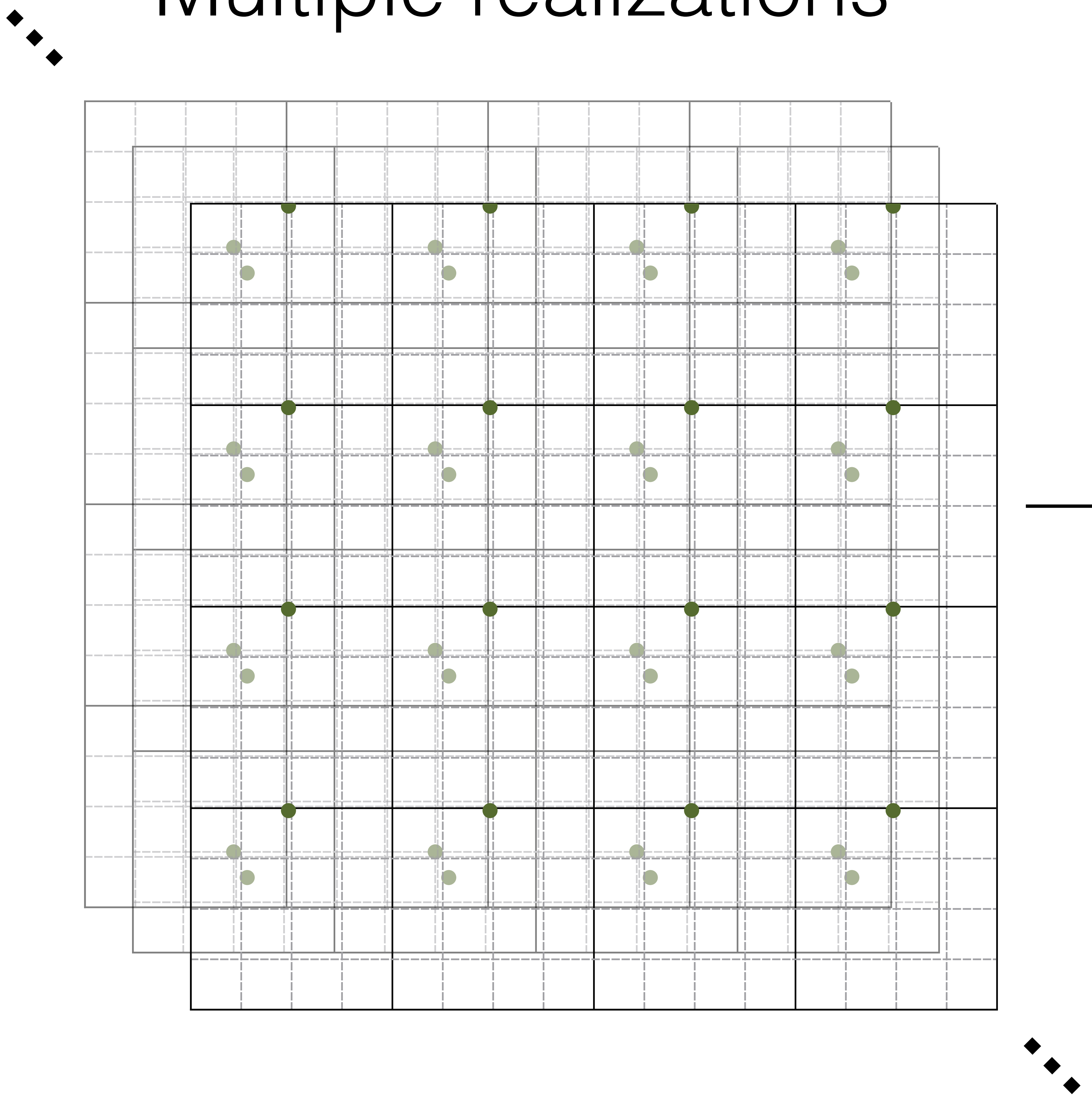


For a given frequency ω

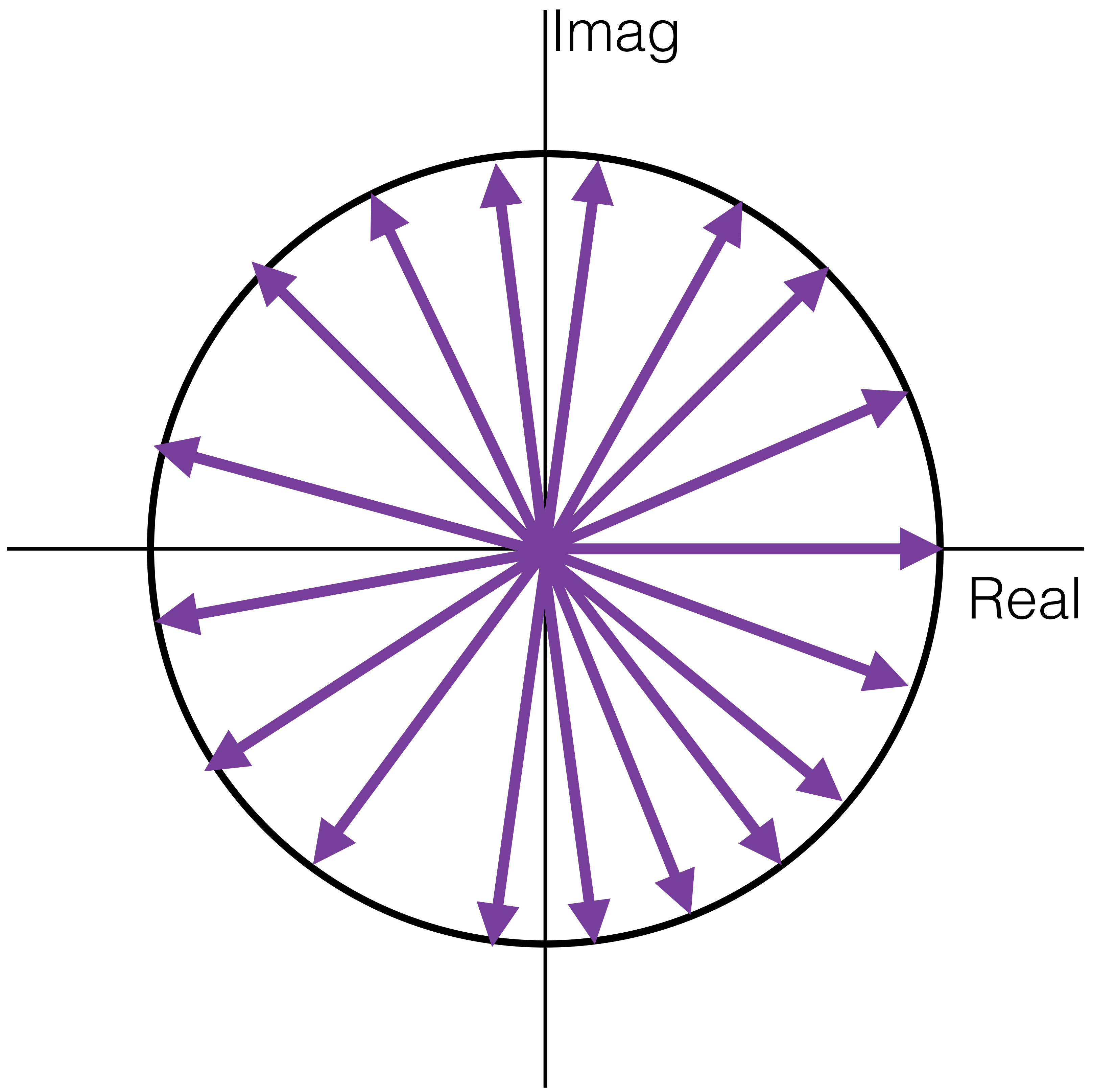


Phase change due to Random Shift

Multiple realizations

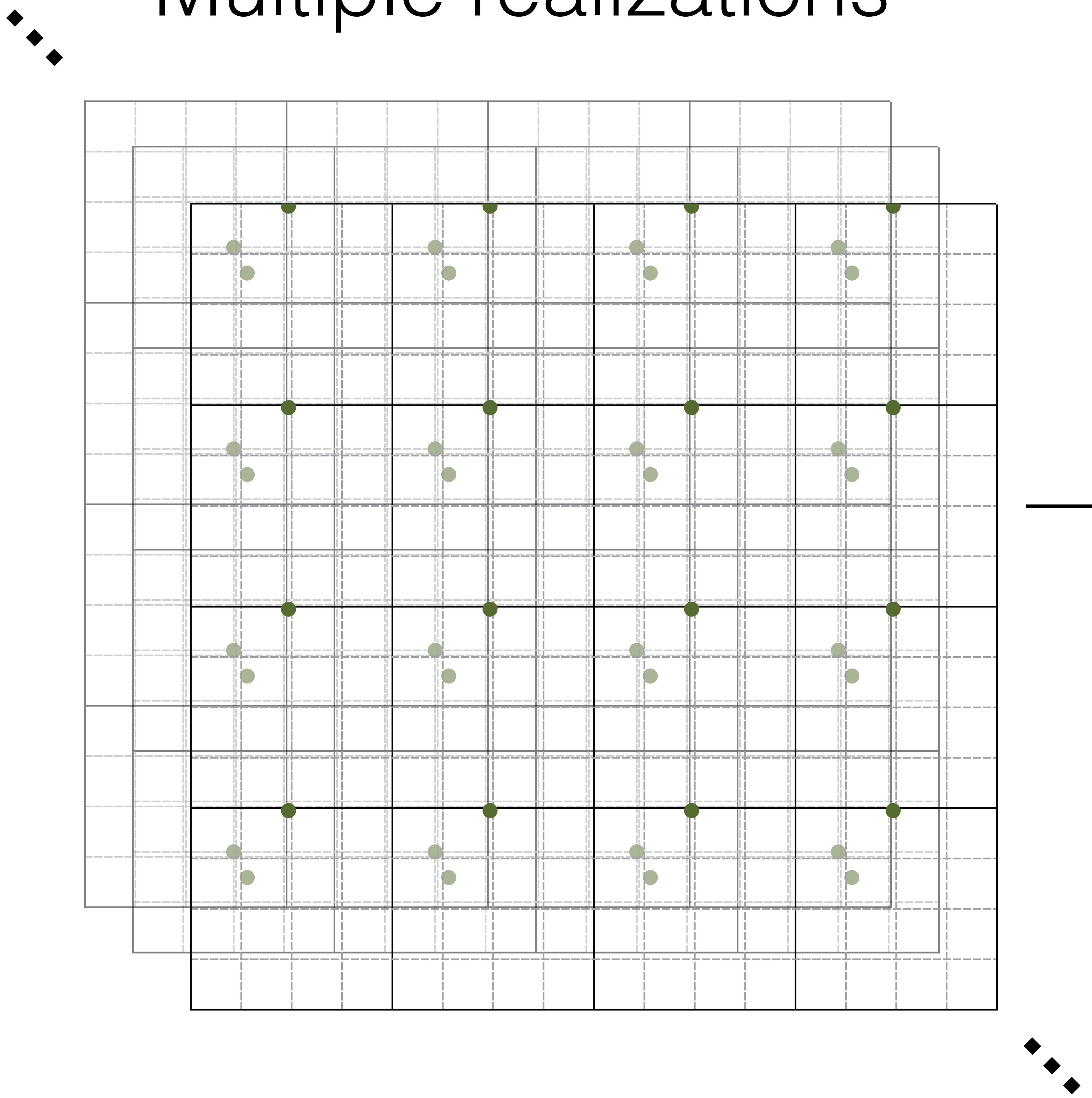


For a given frequency ω

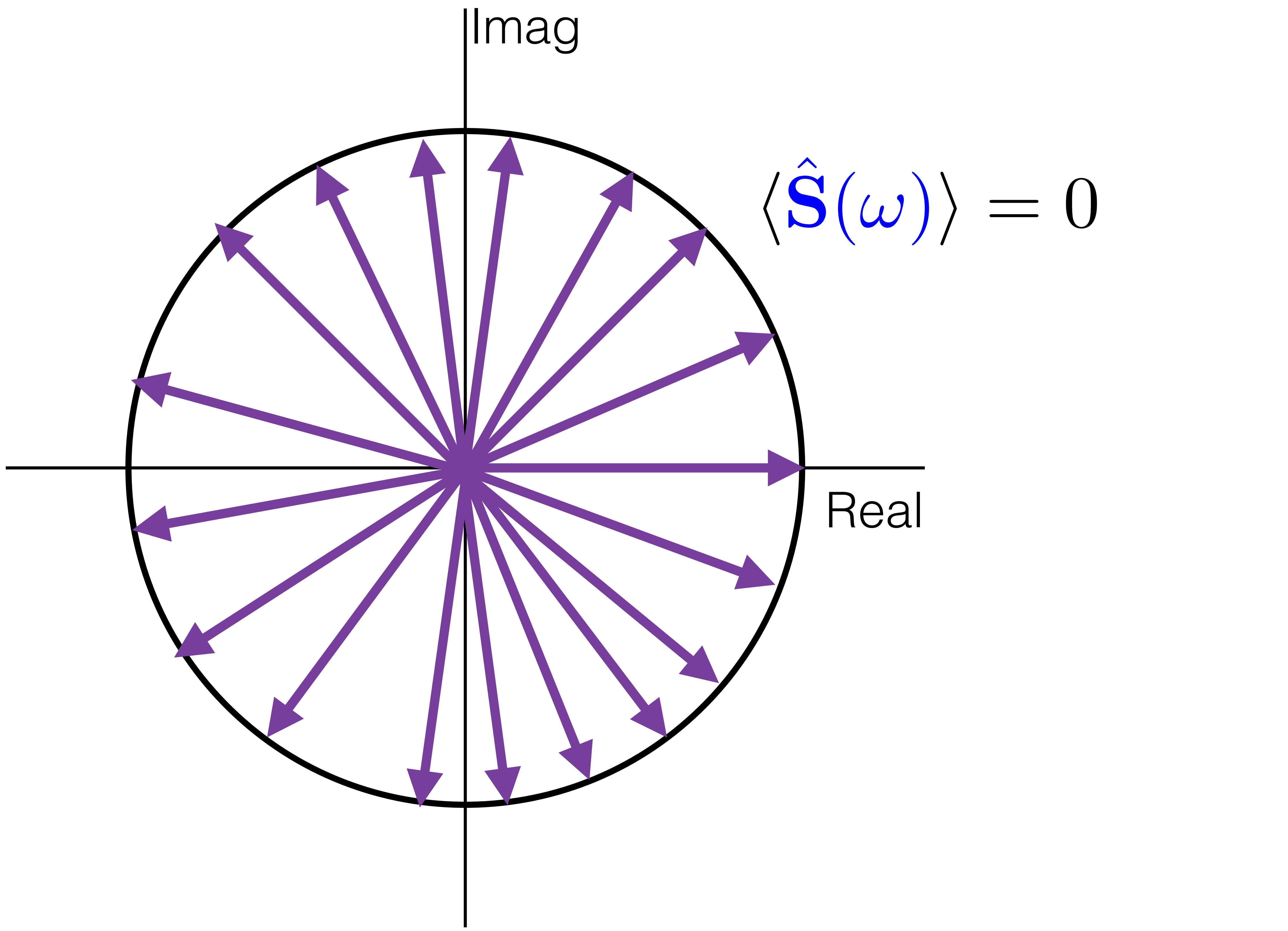


Phase change due to Random Shift

Multiple realizations

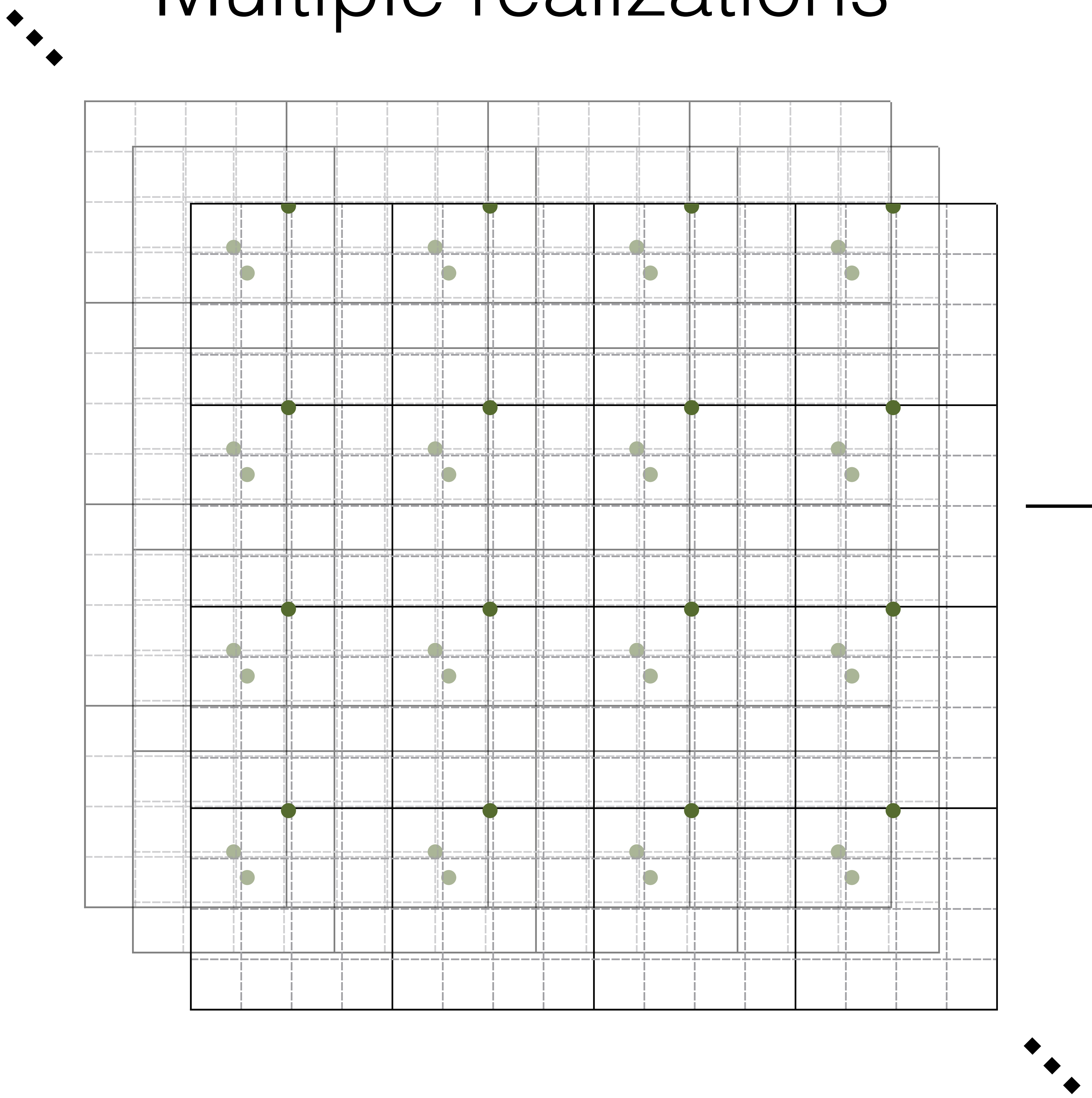


For a given frequency ω

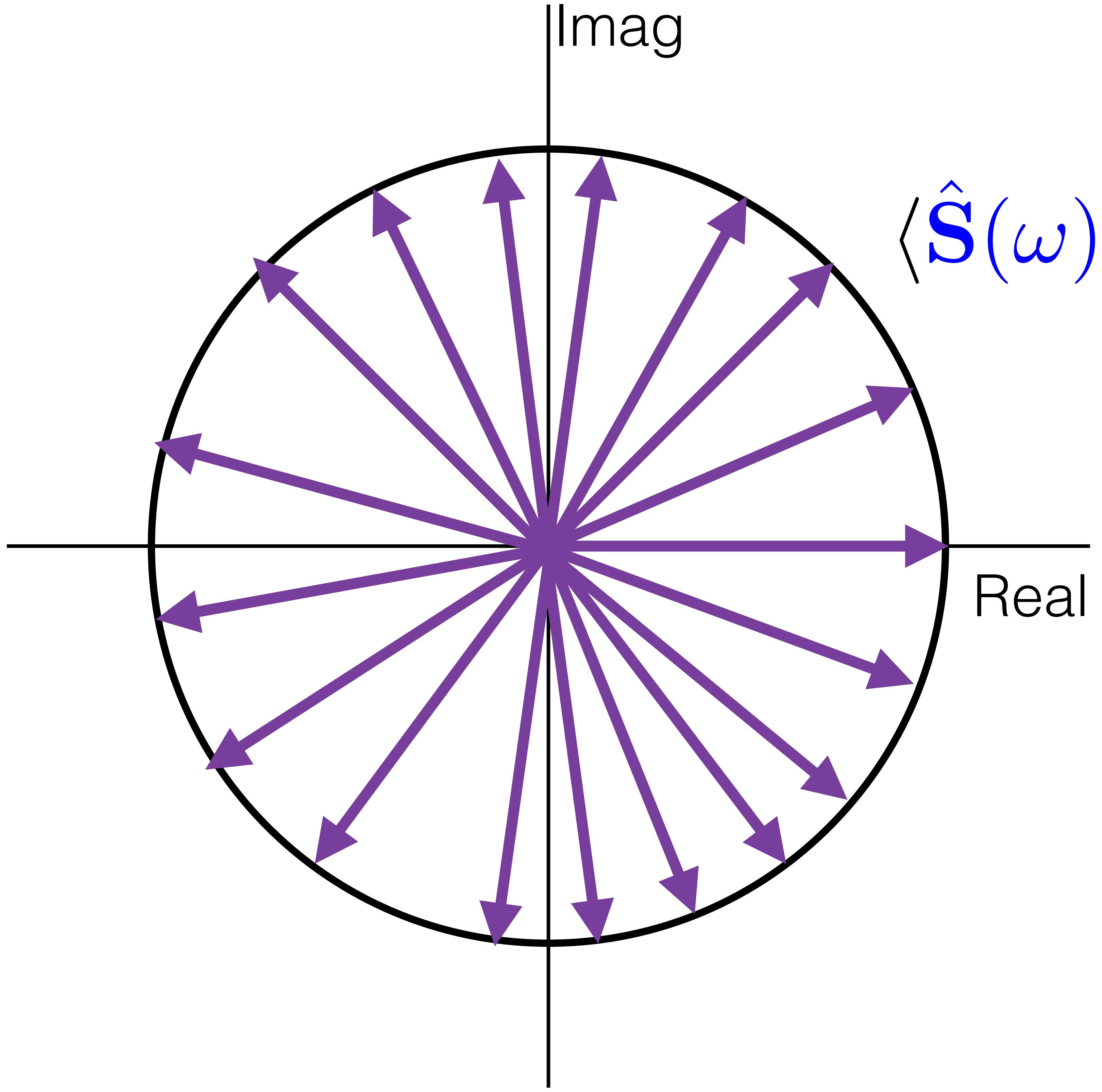


Phase change due to Random Shift

Multiple realizations



For a given frequency ω



$$\langle \hat{\mathbf{S}}(\omega) \rangle = 0 \;\; \forall \omega \neq 0$$

$$\text{Error} = \text{Bias}^2 + \text{Variance}$$

$$\text{Error} = \text{Bias}^2 + \text{Variance}$$

$$\text{Error} = \text{Bias}^2 + \text{Variance}$$

- Homogenization allows representation of error only in terms of variance

$$\text{Error} = \text{Bias}^2 + \text{Variance}$$

- Homogenization allows representation of error only in terms of variance
- We can take any sampling pattern and homogenize it to make the Monte Carlo estimator unbiased.

Variance in the Fourier domain

Variance in the Fourier domain

Error: $$I - \tilde{\mu}_N = \hat{f}(0) - \int_\Omega \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

Variance in the Fourier domain

Error: $$I - \tilde{\mu}_N = \hat{f}(0) - \int_\Omega \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

$$\mathrm{Var}(I - \tilde{\mu}_N)$$

Variance in the Fourier domain

Error: $$I - \tilde{\mu}_N = \hat{f}(0) - \int_{\Omega} \hat{f}^*(\omega)\hat{\mathbf{S}}(\omega)d\omega$$

$$\mathrm{Var}(I - \tilde{\mu}_N) = \mathrm{Var}\left(\hat{f}(0) - \int_{\Omega} \hat{f}^*(\omega)\,\hat{\mathbf{S}}(\omega)\,d\omega\right)$$

Variance in the Fourier domain

$$\mathrm{Var}(I - \tilde{\mu}_N) = \mathrm{Var}\left(\hat{f}(0) - \int_\Omega \hat{f}^*(\omega)\, \hat{\mathbf{S}}(\omega)\, d\omega \right)$$

Variance in the Fourier domain

$$\mathrm{Var}(I - \tilde{\mu}_N) = \mathrm{Var}\left(\hat{f}(0) - \int_\Omega \hat{f}^*(\omega)\, \hat{\mathbf{S}}(\omega)\, d\omega \right)$$

Variance in the Fourier domain

$$\mathrm{Var}(I - \tilde{\mu}_N) = \mathrm{Var}\left(\hat{f}(0) - \int_{\Omega} \hat{f}^*(\omega)\, \hat{\mathbf{S}}(\omega)\, d\omega \right)$$

Variance in the Fourier domain

$$\mathrm{Var}(I - \tilde{\mu}_N) = \mathrm{Var}\left(\hat{f}(0) - \int_{\Omega} \hat{f}^*(\omega)\, \hat{\mathbf{S}}(\omega)\, d\omega \right)$$

$$\mathrm{Var}(\tilde{\mu}_N) = \mathrm{Var}\left(\int_{\Omega} \hat{f}^*(\omega)\, \hat{\mathbf{S}}(\omega) d\omega \right)$$

Variance in the Fourier domain

$$\mathrm{Var}(\tilde{\mu}_N) = \mathrm{Var}\left(\int_\Omega \hat{f}^*(\omega)\,\hat{\mathbf{S}}(\omega)d\omega\right)$$

Variance in the Fourier domain

$$\mathrm{Var}(\tilde{\mu}_N) = \mathrm{Var}\left(\int_\Omega \hat{f}^*(\omega)\, \hat{\mathbf{S}}(\omega) d\omega \right)$$

Variance in the Fourier domain

$$\text{Var}(\tilde{\mu}_N) = \text{Var}\left(\int_\Omega \hat{f}^*(\omega)\,\hat{\mathbf{S}}(\omega) d\omega\right)$$

Variance in the Fourier domain

$$\mathrm{Var}(\tilde{\mu}_N) = \mathrm{Var}\left(\int_\Omega \hat{f}^*(\omega)\,\hat{\mathbf{S}}(\omega) d\omega\right)$$

$$\mathrm{Var}(\tilde{\mu}_N) = \int_\Omega P_f(\omega)\,\mathrm{Var}\left(\hat{\mathbf{S}}(\omega)\right) d\omega$$

where,

$$P_f(\omega) = |\hat{f}^*(\omega)|^2$$ Power Spectrum

Variance in the Fourier domain

$$\mathrm{Var}(\tilde{\mu}_N) = \int_\Omega P_f(\omega)\,\mathrm{Var}\left(\hat{\mathbf{S}}(\omega)\right) d\omega$$

Variance in the Fourier domain

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} {\color{red}P_f(\omega)}\, \mathrm{Var}\left({\color{blue}\hat{\mathbf{S}}(\omega)}\right) d\omega$$

Subr and Kautz [2013]

Variance in the Fourier domain

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} P_f(\omega)\,\mathrm{Var}\left(\hat{\mathbf{S}}(\omega)\right) d\omega$$

Subr and Kautz [2013]

This is a general form, both for homogenised as well as non-homogenised sampling patterns

Variance in the Fourier domain

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} P_f(\omega)\,\mathrm{Var}\left(\hat{\mathbf{S}}(\omega)\right) d\omega$$

Variance in the Fourier domain

$$\mathrm{Var}(\tilde{\mu}_N) = \int_\Omega P_f(\omega)\,\mathrm{Var}\left(\hat{\mathbf{S}}(\omega)\right) d\omega$$

Variance in the Fourier domain

$$\mathrm{Var}(\tilde{\mu}_N) = \int_\Omega P_f(\omega)\, \mathrm{Var}\left(\hat{\mathbf{S}}(\omega)\right) d\omega$$

Variance in the Fourier domain

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} P_f(\omega)\, \mathrm{Var}\left(\hat{\mathbf{S}}(\omega)\right) d\omega$$

For purely random samples:

Variance in the Fourier domain

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} P_f(\omega)\, \mathrm{Var}\left(\hat{\mathbf{S}}(\omega)\right) d\omega$$

For purely random samples:

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} P_f(\omega)\, \langle P_S(\omega) \rangle\, d\omega$$

Fredo Durand [2011]

where,

$$P_S(\omega) = |\hat{\mathbf{S}}(\omega)|^2$$

Variance in the Fourier domain

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} P_f(\omega)\, \mathrm{Var}\left(\hat{\mathbf{S}}(\omega)\right) d\omega$$

For purely random samples: $\langle \hat{\mathbf{S}}(\omega) \rangle = 0$

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} P_f(\omega)\, \langle P_S(\omega) \rangle\, d\omega$$

Fredo Durand [2011]

where,

$$P_S(\omega) = |\hat{\mathbf{S}}(\omega)|^2$$

Variance using Homogenized Samples

Homogenizing any sampling pattern makes $\langle \hat{\mathbf{S}}(\omega) \rangle = 0$

Variance using Homogenized Samples

Homogenizing any sampling pattern makes $\langle \hat{\mathbf{S}}(\omega) \rangle = 0$

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} P_f(\omega)\ \langle P_S(\omega) \rangle \, d\omega$$

Pilleboue et al. [2015]

where,

$$P_S(\omega) = |\hat{\mathbf{S}}(\omega)|^2$$

Variance using Homogenized Samples

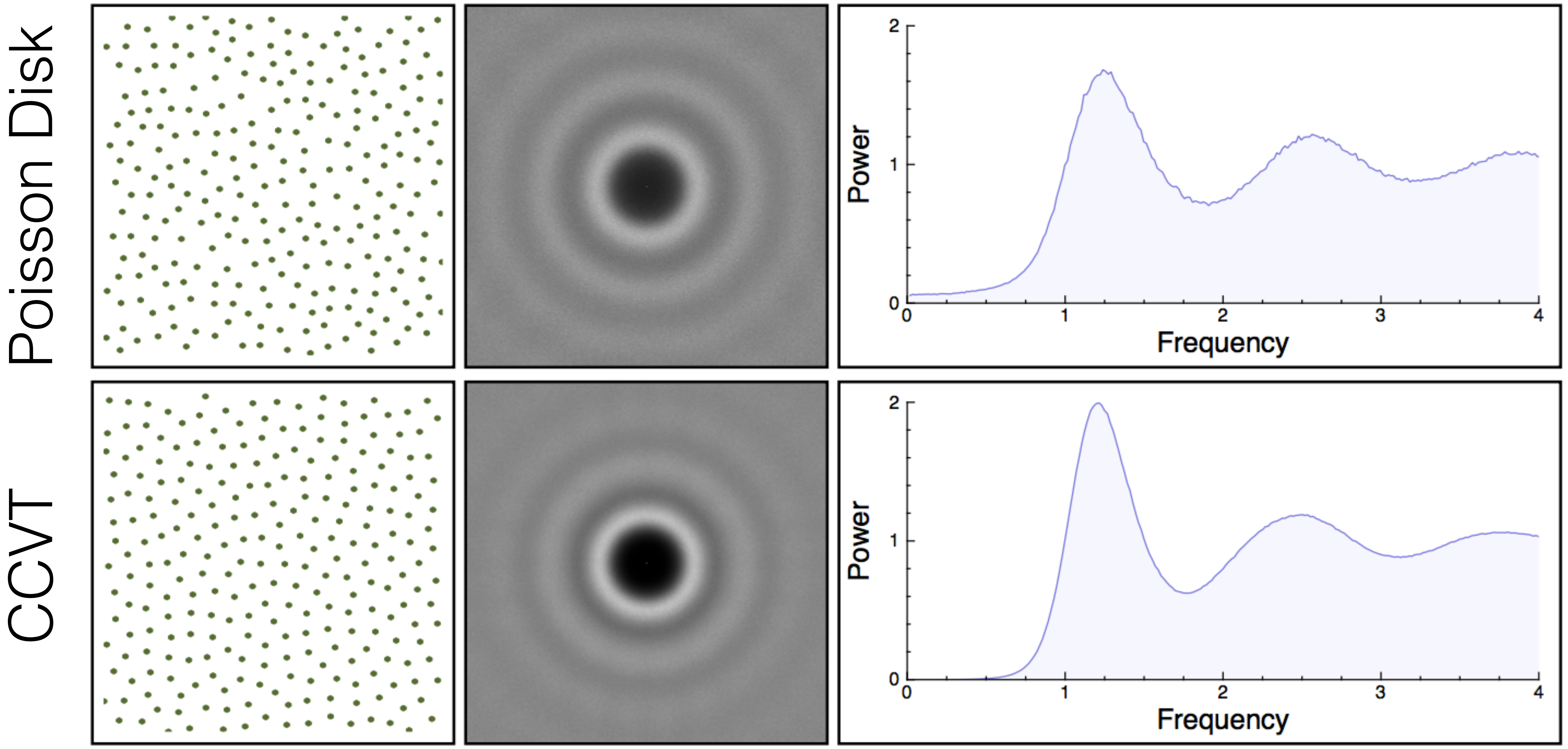
$$\mathrm{Var}(\tilde{\mu}_N) = \int_\Omega P_f(\omega)\ \langle P_S(\omega)\rangle\, d\omega$$

Variance using Homogenized Samples

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} P_f(\omega) \, \langle P_S(\omega) \rangle \, d\omega$$

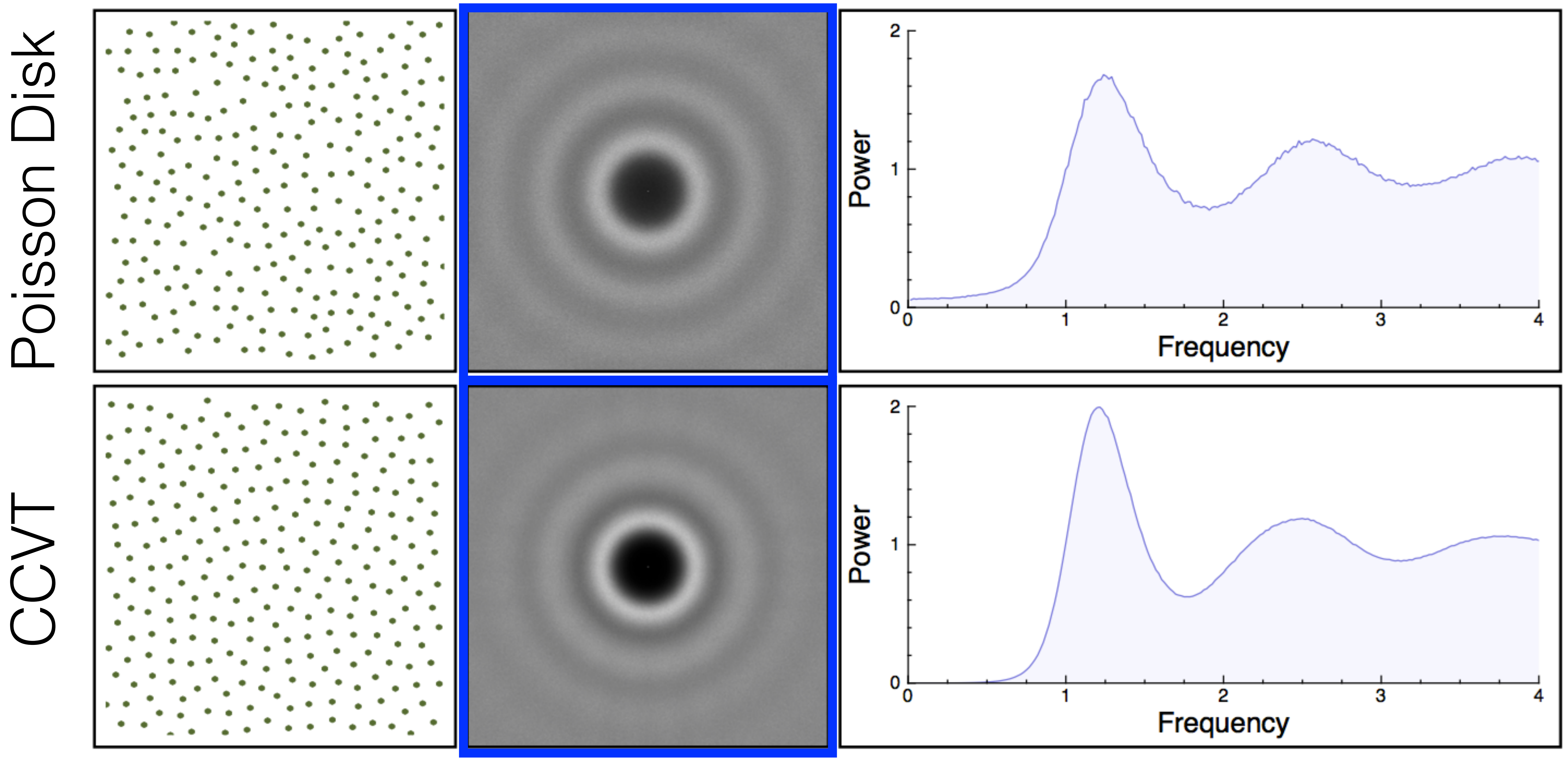
Variance in terms of n-dimensional Power Spectra

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} P_f(\omega) \, \langle P_S(\omega) \rangle \, d\omega$$



Variance in terms of n-dimensional Power Spectra

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} P_f(\omega)\ \langle P_S(\omega)\rangle\, d\omega$$



Variance in the Polar Coordinates

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} P_f(\omega) \, \langle P_S(\omega) \rangle \, d\omega$$

Variance in the Polar Coordinates

$$\mathrm{Var}(\tilde{\mu}_N) = \int_\Omega P_f(\omega)\ \langle P_S(\omega)\rangle\, d\omega$$

In polar coordinates:

Variance in the Polar Coordinates

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} P_f(\omega) \, \langle P_S(\omega) \rangle \, d\omega$$

In polar coordinates:

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^{\infty} \int_{\mathcal{S}^{d-1}} P_f(\rho\mathbf{n}) \, \langle P_{\mathbf{S}}(\rho\mathbf{n}) \rangle \, d\mathbf{n} \, d\rho$$

Variance in the Polar Coordinates

$$\mathrm{Var}(\tilde{\mu}_N) = \int_{\Omega} P_f(\omega) \, \langle P_S(\omega) \rangle \, d\omega$$

In polar coordinates:

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^{\infty} \int_{\mathcal{S}^{d-1}} P_f(\rho \mathbf{n}) \, \langle P_{\mathbf{S}}(\rho \mathbf{n}) \rangle \, d\mathbf{n} \, d\rho$$

Variance in the Polar Coordinates

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^\infty \int_{\mathcal{S}^{d-1}} P_f(\rho \mathbf{n}) \left\langle P_{\mathbf{S}}(\rho \mathbf{n}) \right\rangle d\mathbf{n}\, d\rho$$

Variance in the Polar Coordinates

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^{\infty} \int_{\mathcal{S}^{d-1}} P_f(\rho\mathbf{n}) \left\langle P_{\mathbf{S}}(\rho\mathbf{n}) \right\rangle d\mathbf{n}\, d\rho$$

Variance for Isotropic Power Spectra

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^{\infty} \int_{\mathcal{S}^{d-1}} P_f(\rho\mathbf{n}) \left\langle P_{\mathbf{S}}(\rho\mathbf{n}) \right\rangle d\mathbf{n}\, d\rho$$

Variance for Isotropic Power Spectra

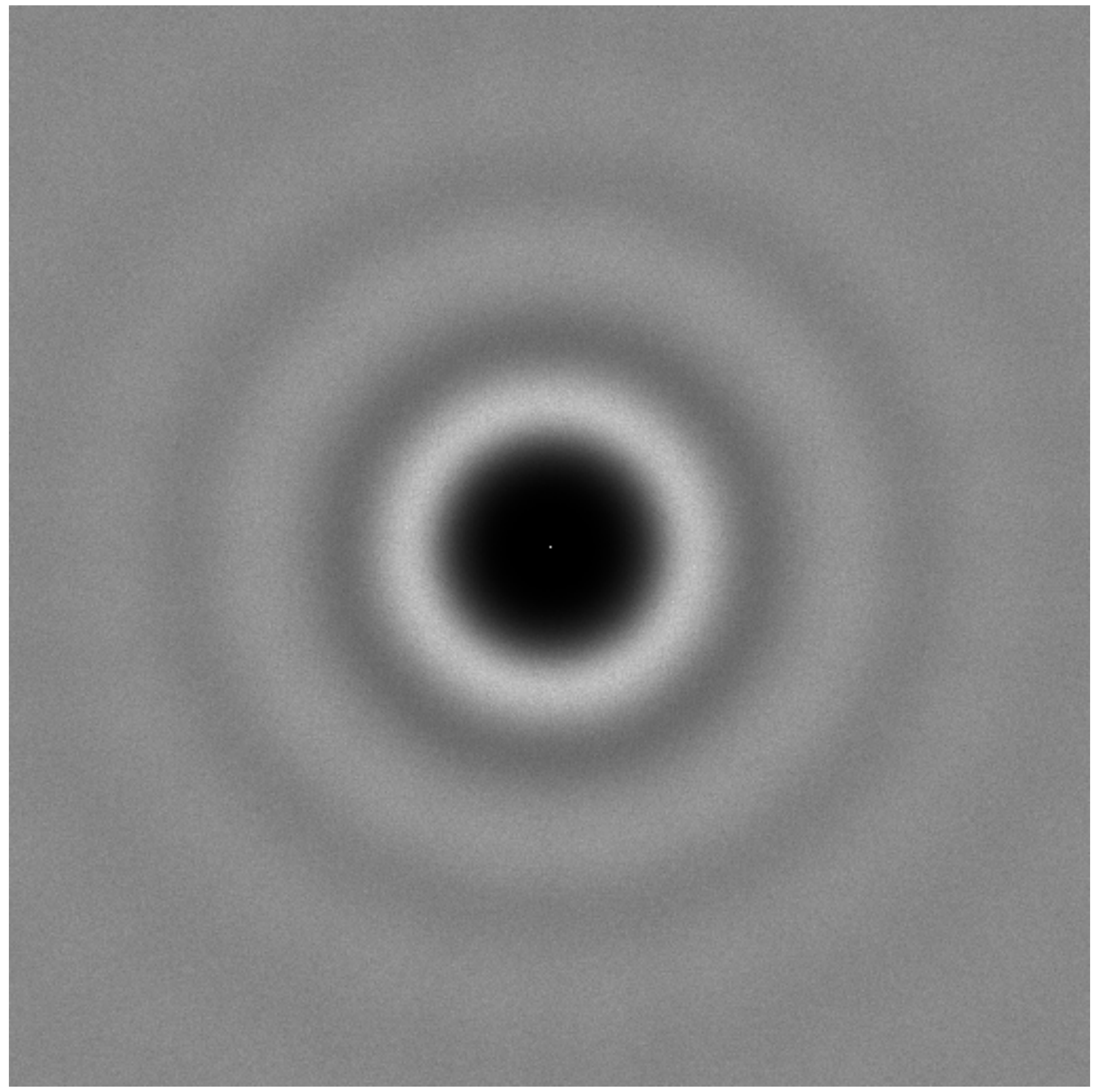
$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^\infty \int_{\mathcal{S}^{d-1}} P_f(\rho\mathbf{n}) \left\langle P_{\mathbf{S}}(\rho\mathbf{n}) \right\rangle d\mathbf{n}\, d\rho$$

For isotropic power spectra:

Variance for Isotropic Power Spectra

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^\infty \int_{\mathcal{S}^{d-1}} P_f(\rho\mathbf{n}) \langle P_{\mathbf{S}}(\rho\mathbf{n}) \rangle \, d\mathbf{n} \, d\rho$$

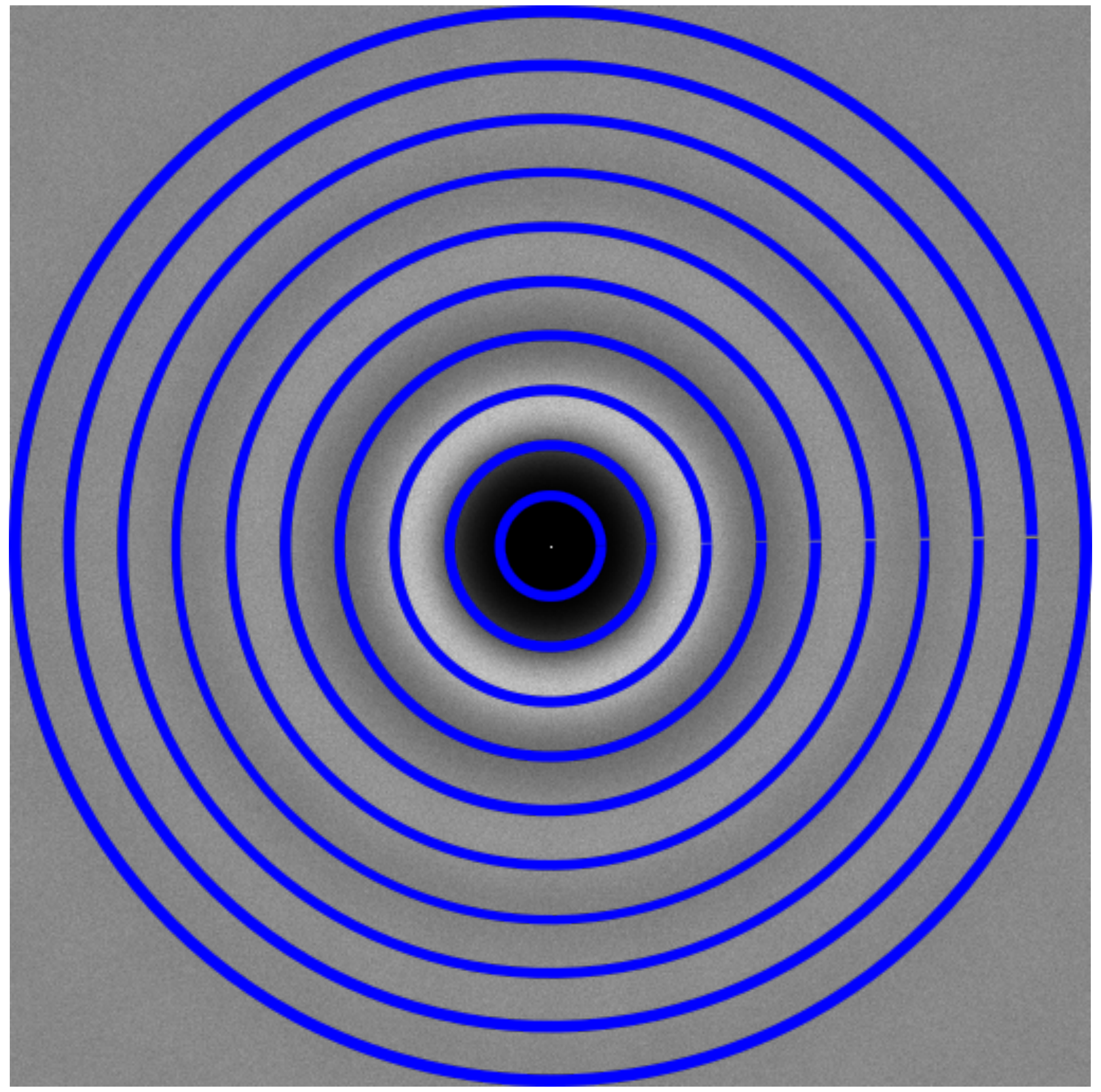
For isotropic power spectra:



Variance for Isotropic Power Spectra

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^\infty \int_{\mathcal{S}^{d-1}} P_f(\rho\mathbf{n}) \langle P_{\mathbf{S}}(\rho\mathbf{n}) \rangle \, d\mathbf{n} \, d\rho$$

For isotropic power spectra:

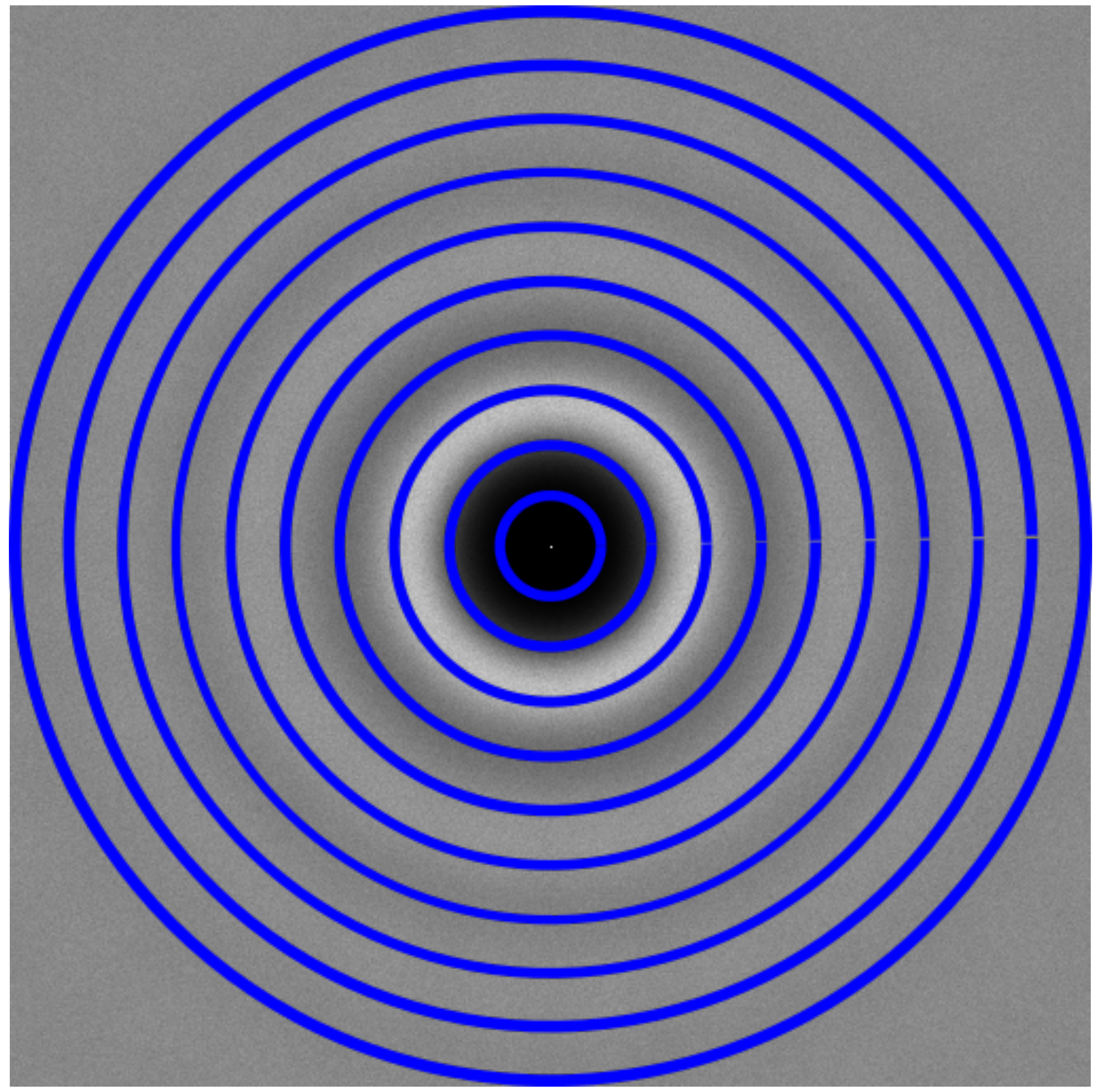


Variance for Isotropic Power Spectra

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^\infty \int_{\mathcal{S}^{d-1}} P_f(\rho \mathbf{n}) \langle P_{\mathbf{S}}(\rho \mathbf{n}) \rangle \, d\mathbf{n} \, d\rho$$

For isotropic power spectra:

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^\infty \tilde{P}_f(\rho) \langle \tilde{P}_{\mathbf{S}}(\rho) \rangle \, d\rho$$

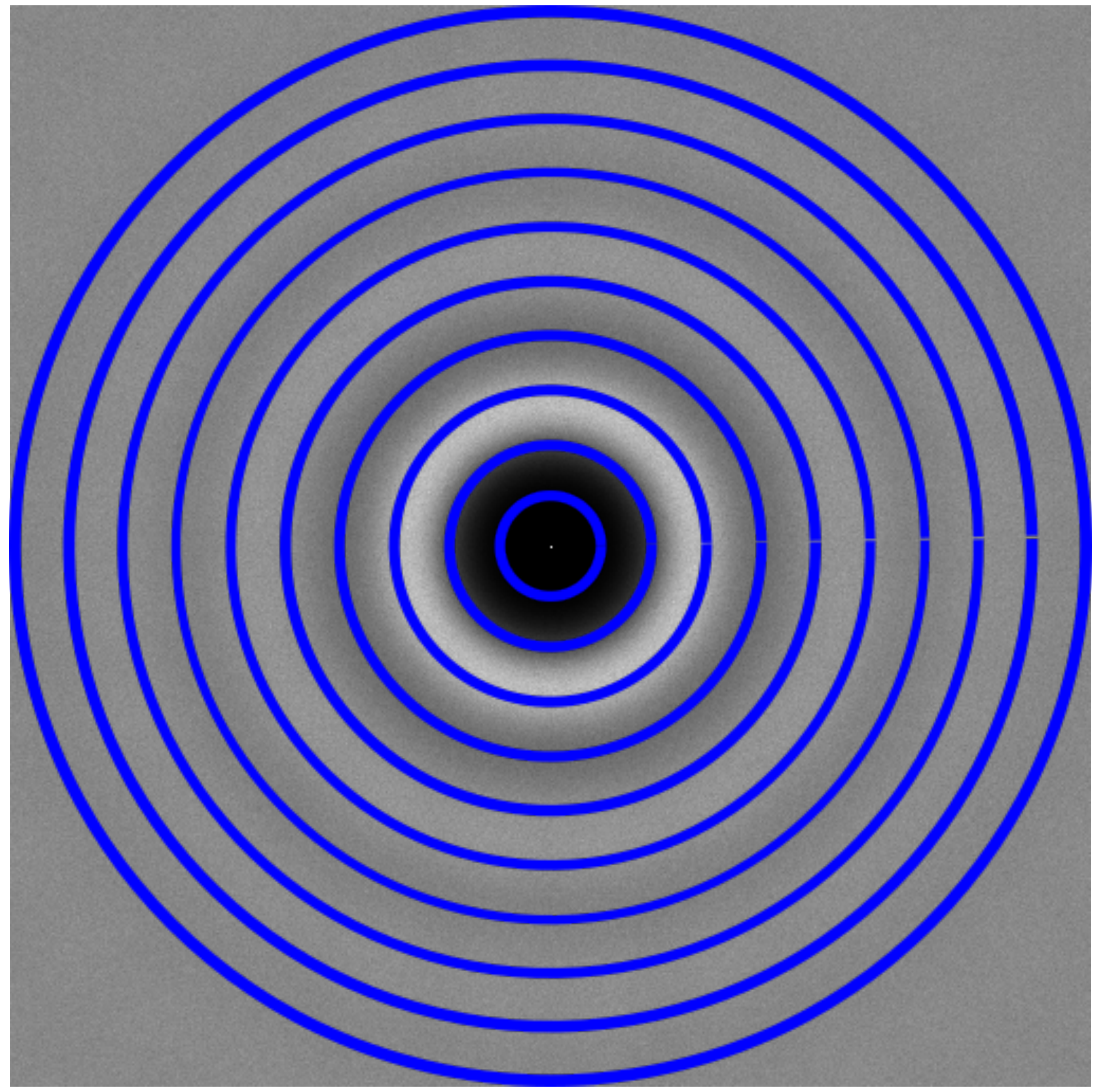


Variance for Isotropic Power Spectra

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^{\infty} \int_{\mathcal{S}^{d-1}} P_f(\rho \mathbf{n}) \left\langle P_{\mathbf{S}}(\rho \mathbf{n}) \right\rangle d\mathbf{n}\, d\rho$$

For isotropic power spectra:

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^{\infty} \tilde{P}_f(\rho) \left\langle \tilde{P}_{\mathbf{S}}(\rho) \right\rangle d\rho$$



Variance for Isotropic Power Spectra

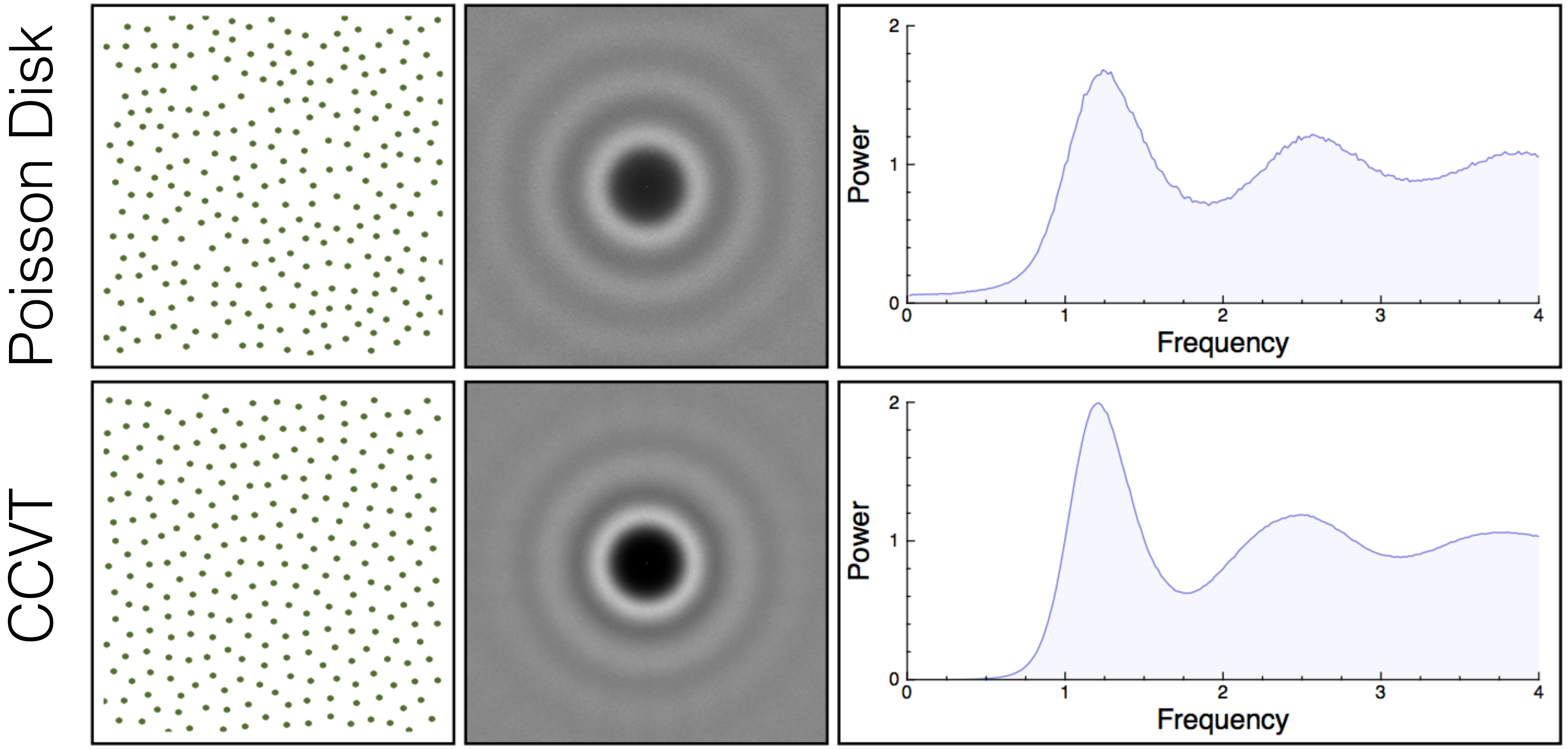
$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^\infty \int_{\mathcal{S}^{d-1}} P_f(\rho\mathbf{n}) \langle P_{\mathbf{S}}(\rho\mathbf{n})\rangle \, d\mathbf{n} \, d\rho$$

For isotropic power spectra:

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^\infty \tilde{P}_f(\rho) \langle \tilde{P}_{\mathbf{S}}(\rho)\rangle \, d\rho$$

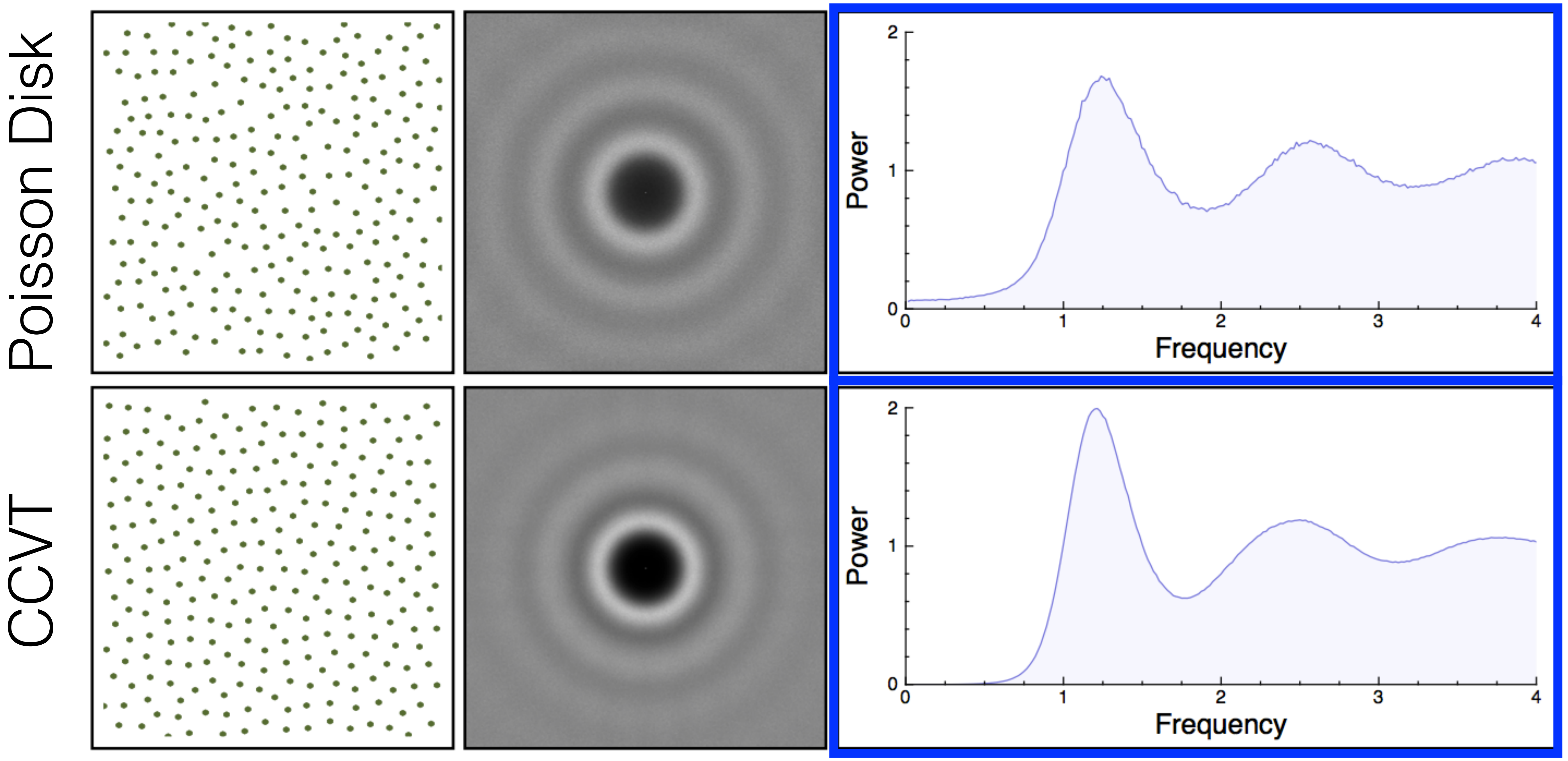
Variance in terms of 1-dimensional Power Spectra

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^{\infty} \tilde{P}_f(\rho) \, \langle \tilde{P}_{\mathbf{S}}(\rho) \rangle \, d\rho$$



Variance in terms of 1-dimensional Power Spectra

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^{\infty} \tilde{P}_f(\rho) \langle \tilde{P}_{\mathbf{S}}(\rho) \rangle \, d\rho$$

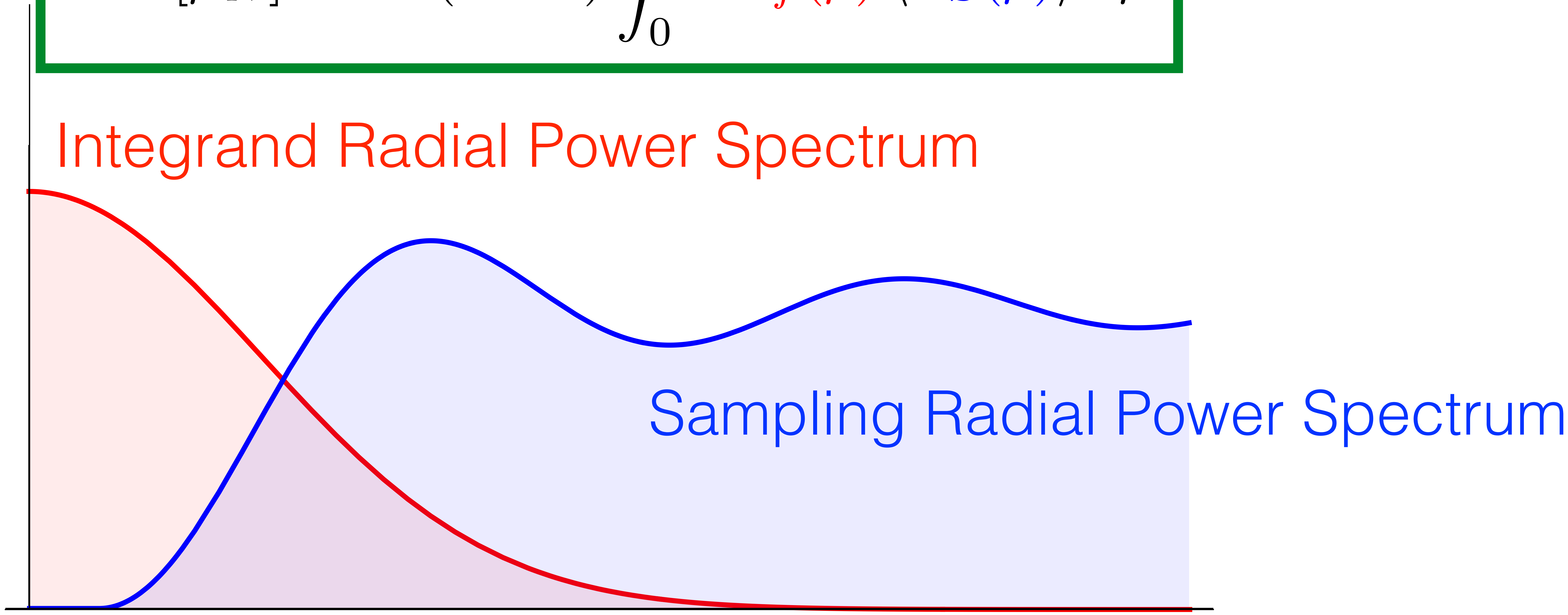


Variance: Integral over Product of Power Spectra

$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^{\infty} \tilde{P}_f(\rho) \, \langle \tilde{P}_{\mathbf{S}}(\rho) \rangle \, d\rho$$

Variance: Integral over Product of Power Spectra

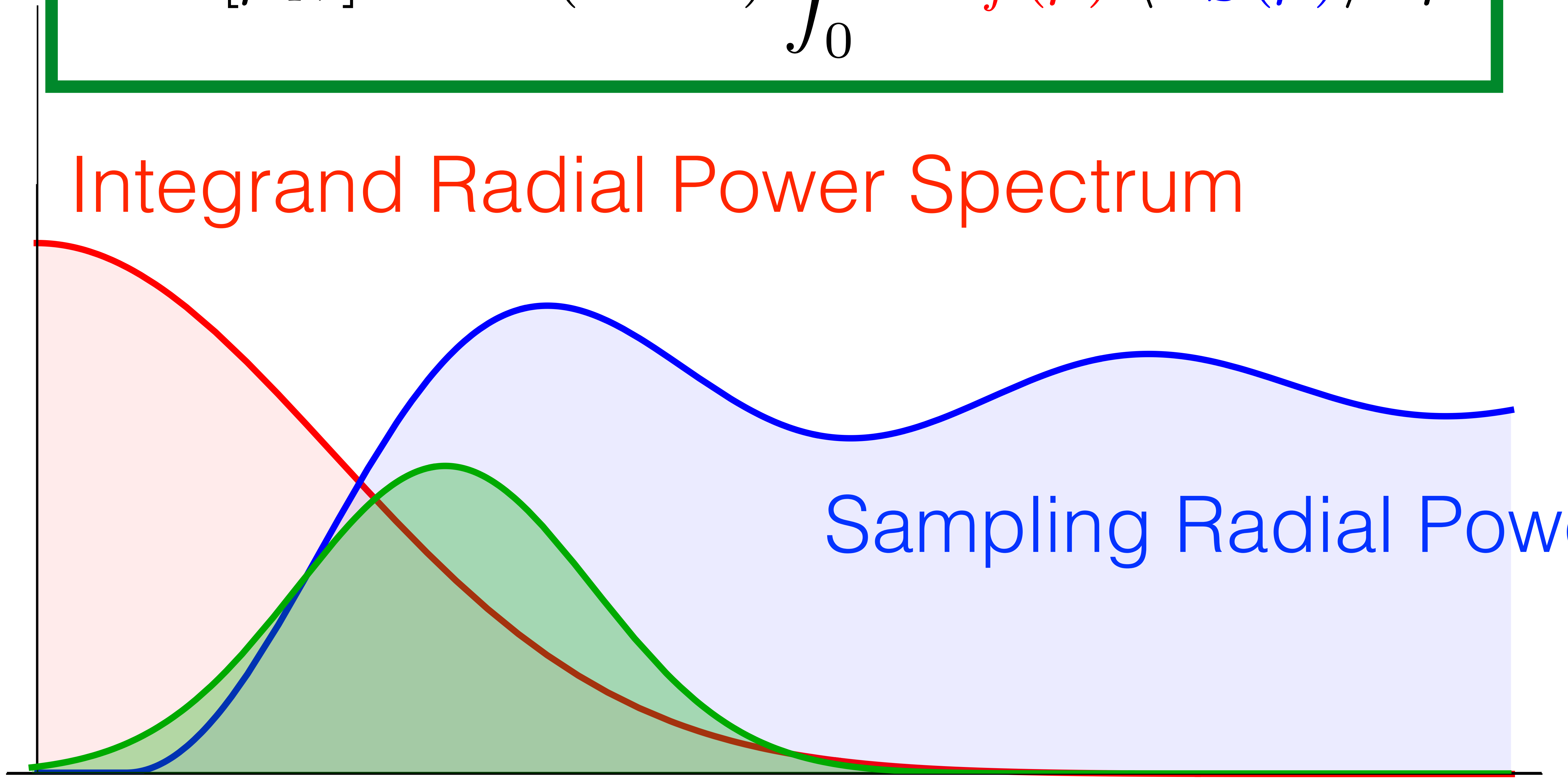
$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^\infty \tilde{P}_f(\rho) \, \langle \tilde{P}_{\mathbf{S}}(\rho) \rangle \, d\rho$$



For given number of Samples

Variance: Integral over Product of Power Spectra

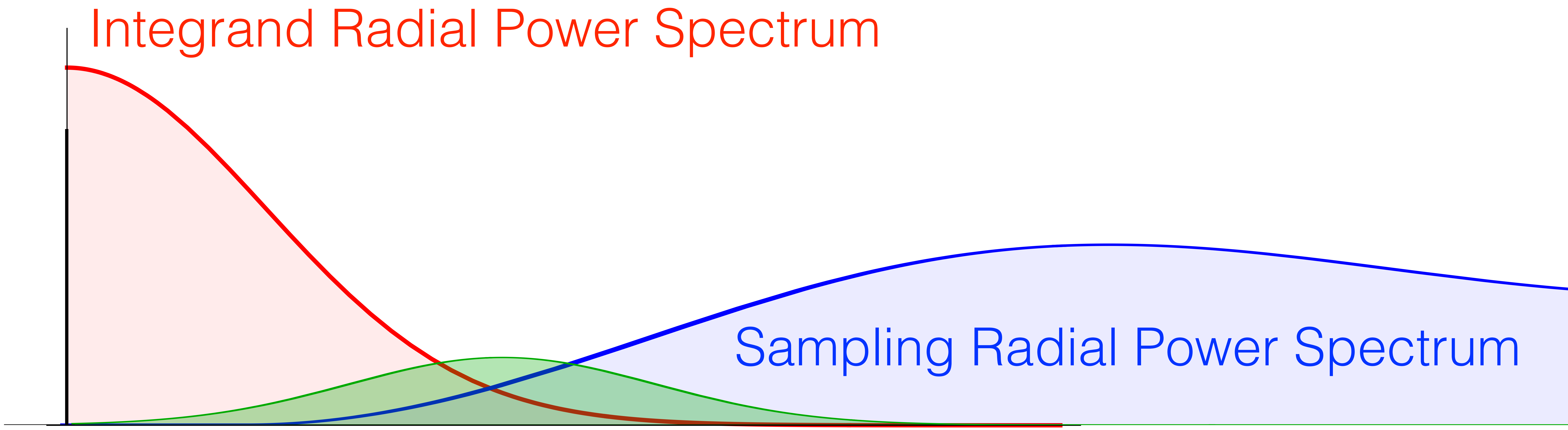
$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^\infty \tilde{P}_f(\rho) \langle \tilde{P}_{\mathbf{S}}(\rho) \rangle \, d\rho$$



For given number of Samples

Variance: Integral over Product of Power Spectra

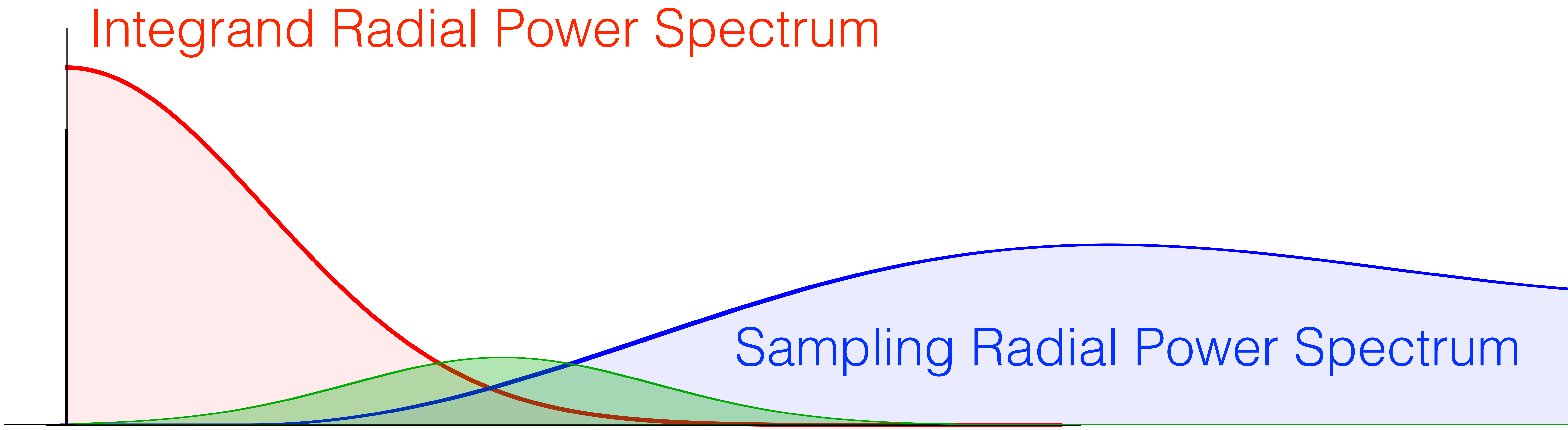
$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^\infty \tilde{P}_f(\rho) \langle \tilde{P}_{\mathbf{S}}(\rho) \rangle \, d\rho$$



For given number of Samples

Variance: Integral over Product of Power Spectra

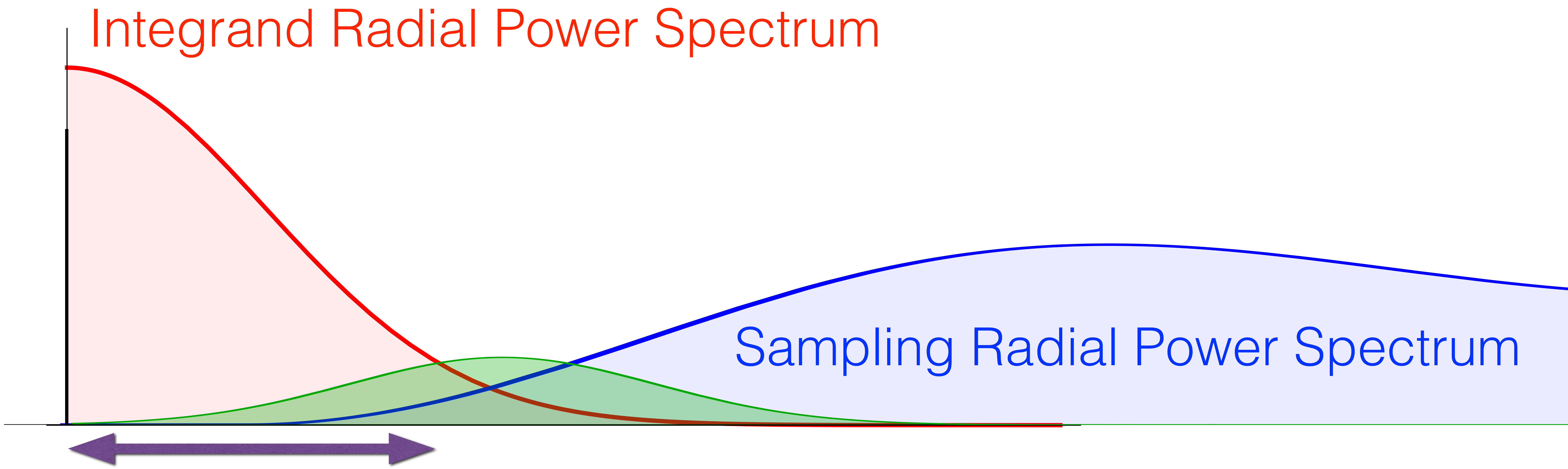
$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^\infty \tilde{P}_f(\rho) \, \langle \tilde{P}_{\mathbf{S}}(\rho) \rangle \, d\rho$$



For given number of Samples

Variance: Integral over Product of Power Spectra

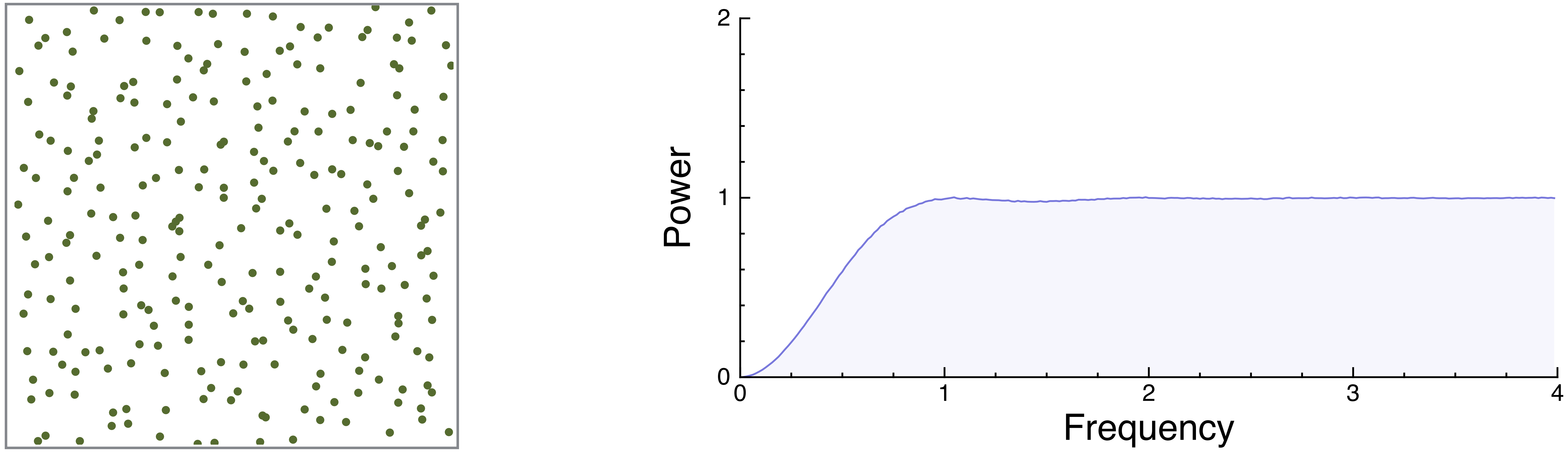
$$Var[\tilde{\mu}_N] = \mathcal{M}(\mathcal{S}^{d-1}) \int_0^{\infty} \tilde{P}_f(\rho) \, \langle \tilde{P}_{\mathbf{S}}(\rho) \rangle \, d\rho$$



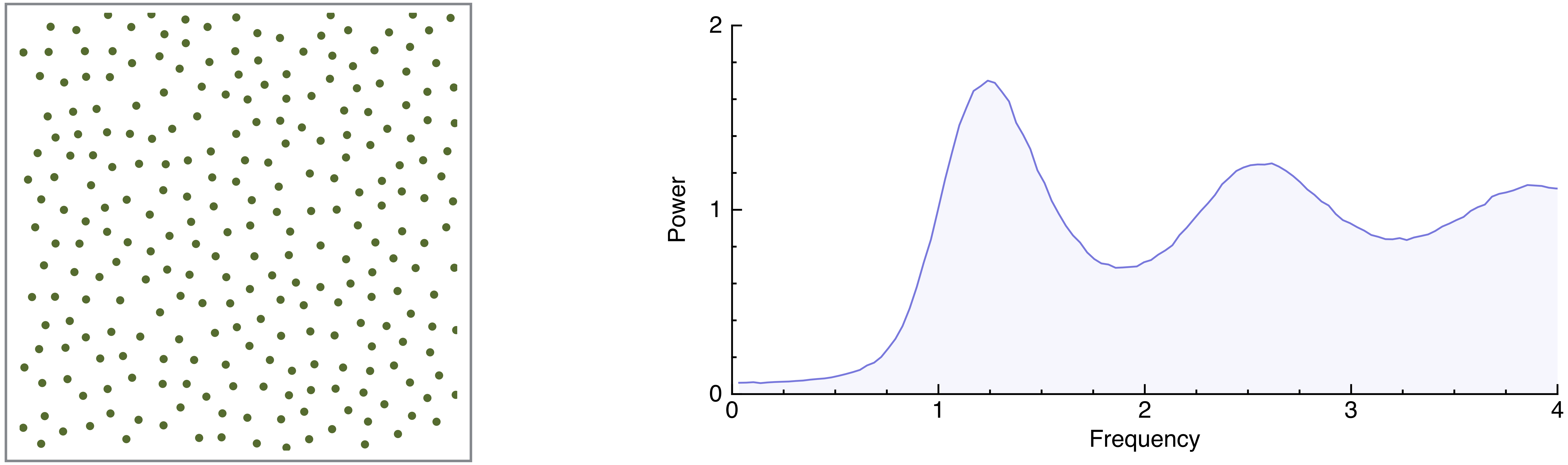
For given number of Samples

Spatial Distribution vs Radial Mean Power Spectra

Jitter



Poisson Disk



For 2-dimensions

Samplers	Worst Case	Best Case
Random		
Jitter		
Poisson Disk		
CCVT		

Pilleboue et al. [2015]

For 2-dimensions

Samplers	Worst Case	Best Case
Random	$\mathcal{O}(N^{-1})$	
Jitter		
Poisson Disk		
CCVT		

Pilleboue et al. [2015]

For 2-dimensions

Samplers	Worst Case	Best Case
Random	$\mathcal{O}(N^{-1})$	$\mathcal{O}(N^{-1})$
Jitter		
Poisson Disk		
CCVT		

Pilleboue et al. [2015]

For 2-dimensions

Samplers	Worst Case	Best Case
Random	$\mathcal{O}(N^{-1})$	$\mathcal{O}(N^{-1})$
Jitter	$\mathcal{O}(N^{-1.5})$	
Poisson Disk		
CCVT		

Pilleboue et al. [2015]

For 2-dimensions

Samplers	Worst Case	Best Case
Random	$\mathcal{O}(N^{-1})$	$\mathcal{O}(N^{-1})$
Jitter	$\mathcal{O}(N^{-1.5})$	$\mathcal{O}(N^{-2})$
Poisson Disk		
CCVT		

Pilleboue et al. [2015]

For 2-dimensions

Samplers	Worst Case	Best Case
Random	$\mathcal{O}(N^{-1})$	$\mathcal{O}(N^{-1})$
Jitter	$\mathcal{O}(N^{-1.5})$	$\mathcal{O}(N^{-2})$
Poisson Disk	$\mathcal{O}(N^{-1})$	
CCVT		

Pilleboue et al. [2015]

For 2-dimensions

Samplers	Worst Case	Best Case
Random	$\mathcal{O}(N^{-1})$	$\mathcal{O}(N^{-1})$
Jitter	$\mathcal{O}(N^{-1.5})$	$\mathcal{O}(N^{-2})$
Poisson Disk	$\mathcal{O}(N^{-1})$	$\mathcal{O}(N^{-1})$
CCVT		

Pilleboue et al. [2015]

For 2-dimensions

Samplers	Worst Case	Best Case
Random	$\mathcal{O}(N^{-1})$	$\mathcal{O}(N^{-1})$
Jitter	$\mathcal{O}(N^{-1.5})$	$\mathcal{O}(N^{-2})$
Poisson Disk	$\mathcal{O}(N^{-1})$	$\mathcal{O}(N^{-1})$
CCVT	$\mathcal{O}(N^{-1.5})$	

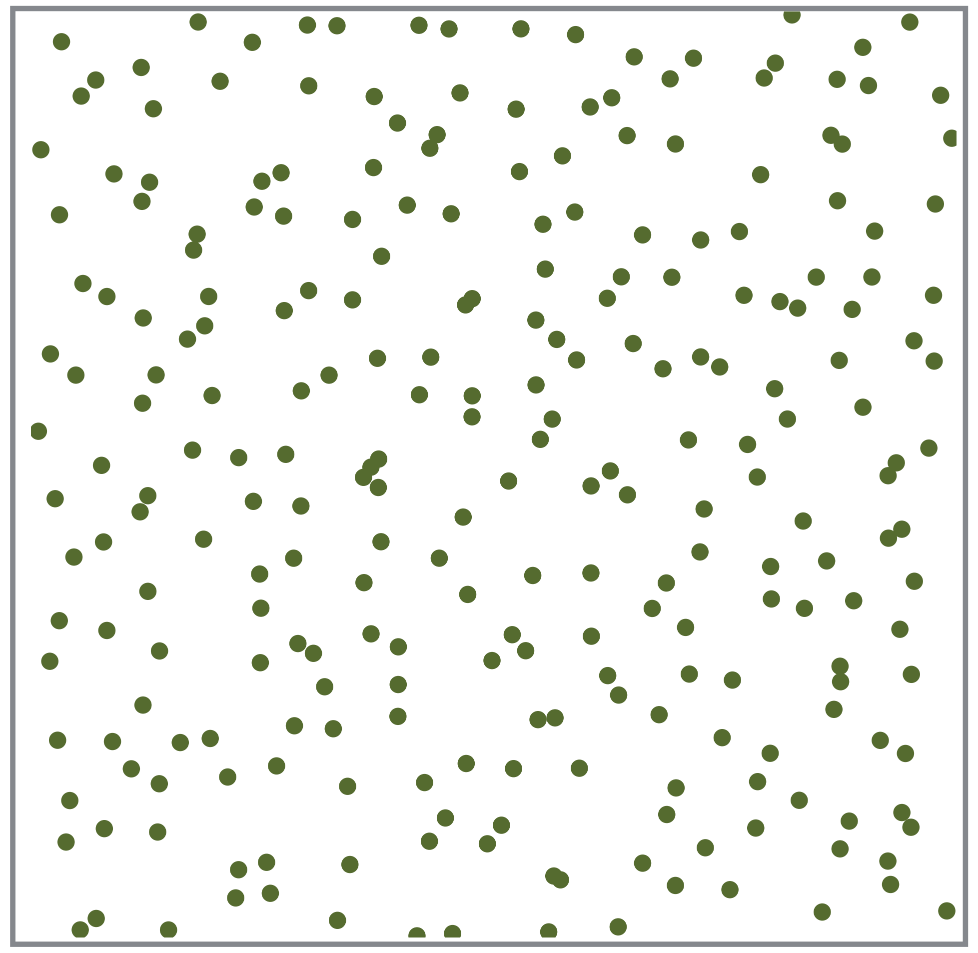
Pilleboue et al. [2015]

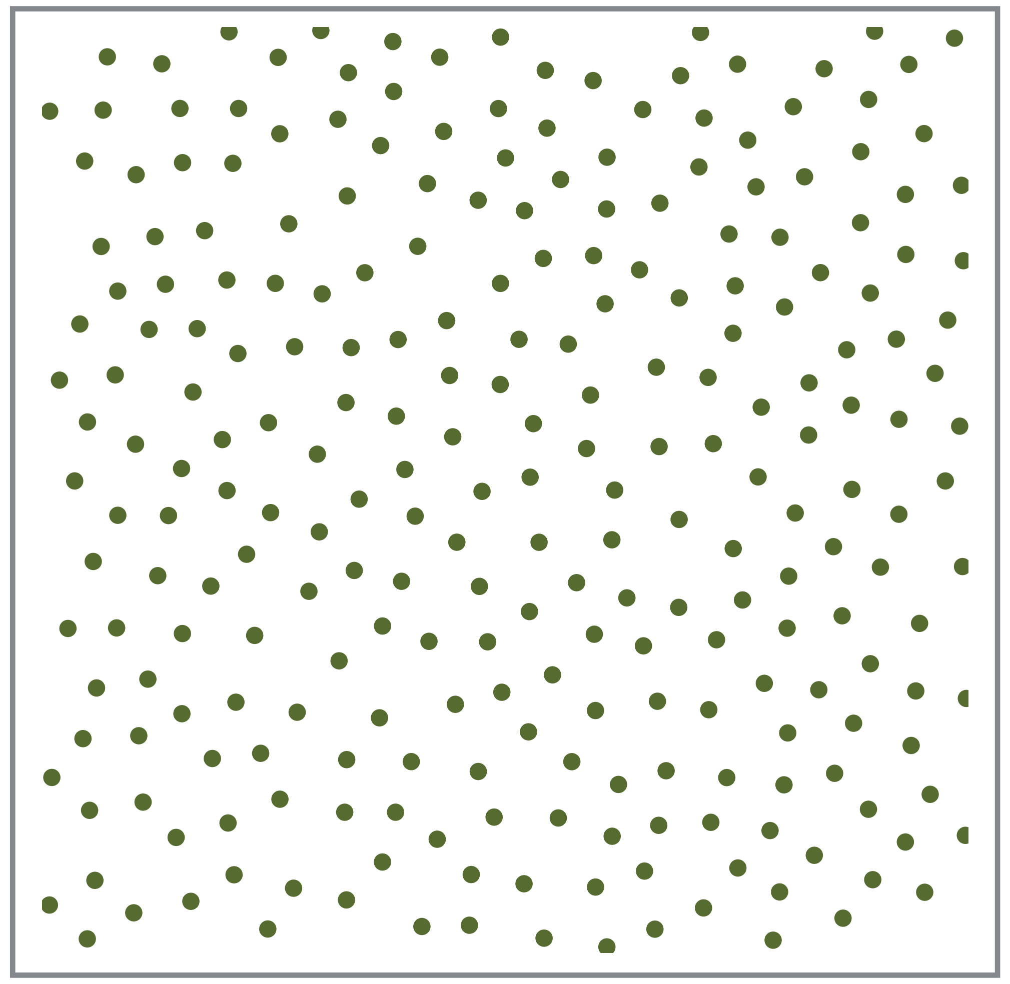
For 2-dimensions

Samplers	Worst Case	Best Case
Random	$\mathcal{O}(N^{-1})$	$\mathcal{O}(N^{-1})$
Jitter	$\mathcal{O}(N^{-1.5})$	$\mathcal{O}(N^{-2})$
Poisson Disk	$\mathcal{O}(N^{-1})$	$\mathcal{O}(N^{-1})$
CCVT	$\mathcal{O}(N^{-1.5})$	$\mathcal{O}(N^{-3})$

Pilleboue et al. [2015]

For 2-dimensions


Jitter

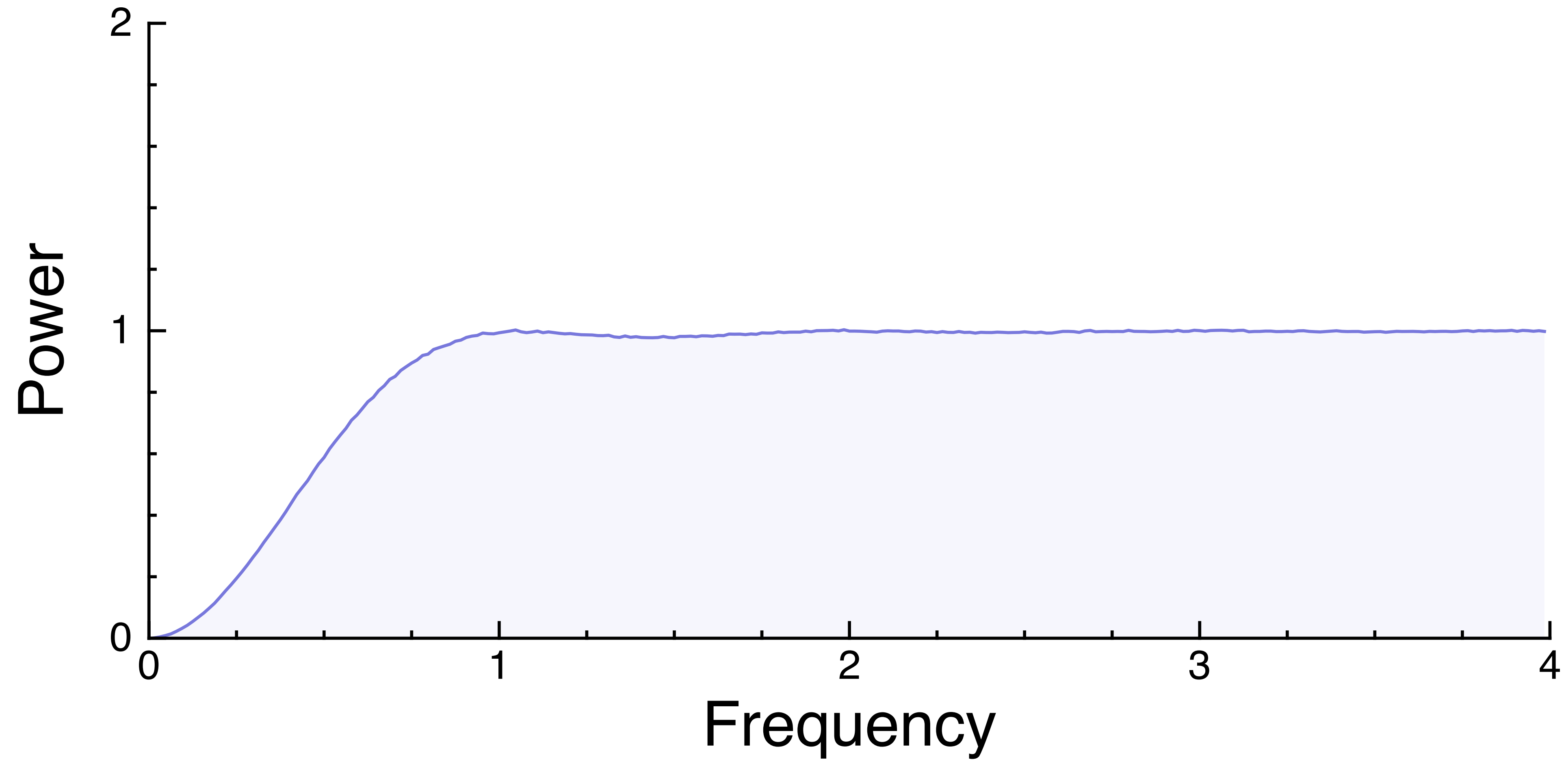

Poisson Disk

Samplers	Worst Case	Best Case
Random	$\mathcal{O}(N^{-1})$	$\mathcal{O}(N^{-1})$
Jitter	$\mathcal{O}(N^{-1.5})$	$\mathcal{O}(N^{-2})$
Poisson Disk	$\mathcal{O}(N^{-1})$	$\mathcal{O}(N^{-1})$
CCVT	$\mathcal{O}(N^{-1.5})$	$\mathcal{O}(N^{-3})$

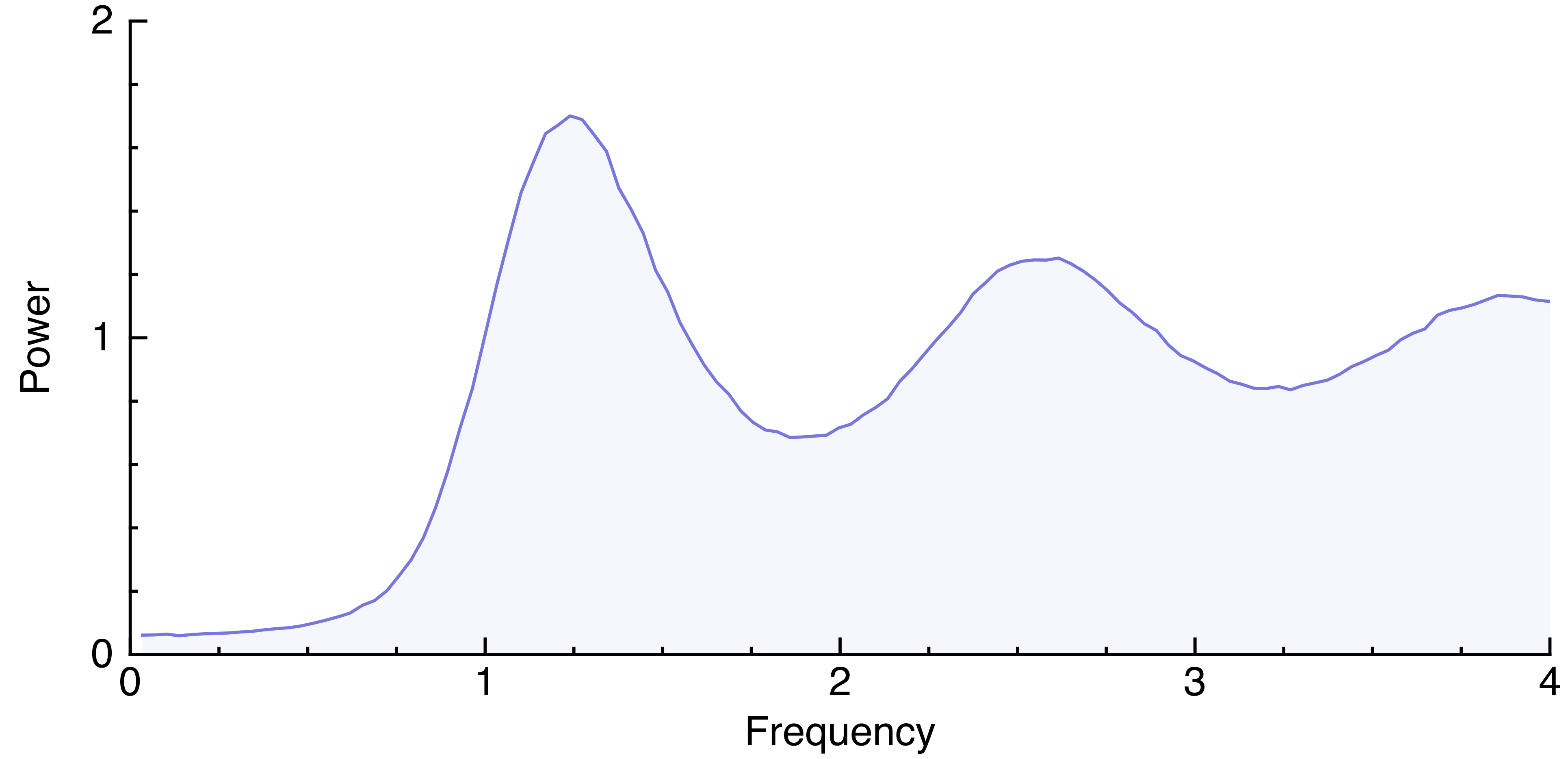
Pilleboue et al. [2015]

Low Frequency Region

Jitter

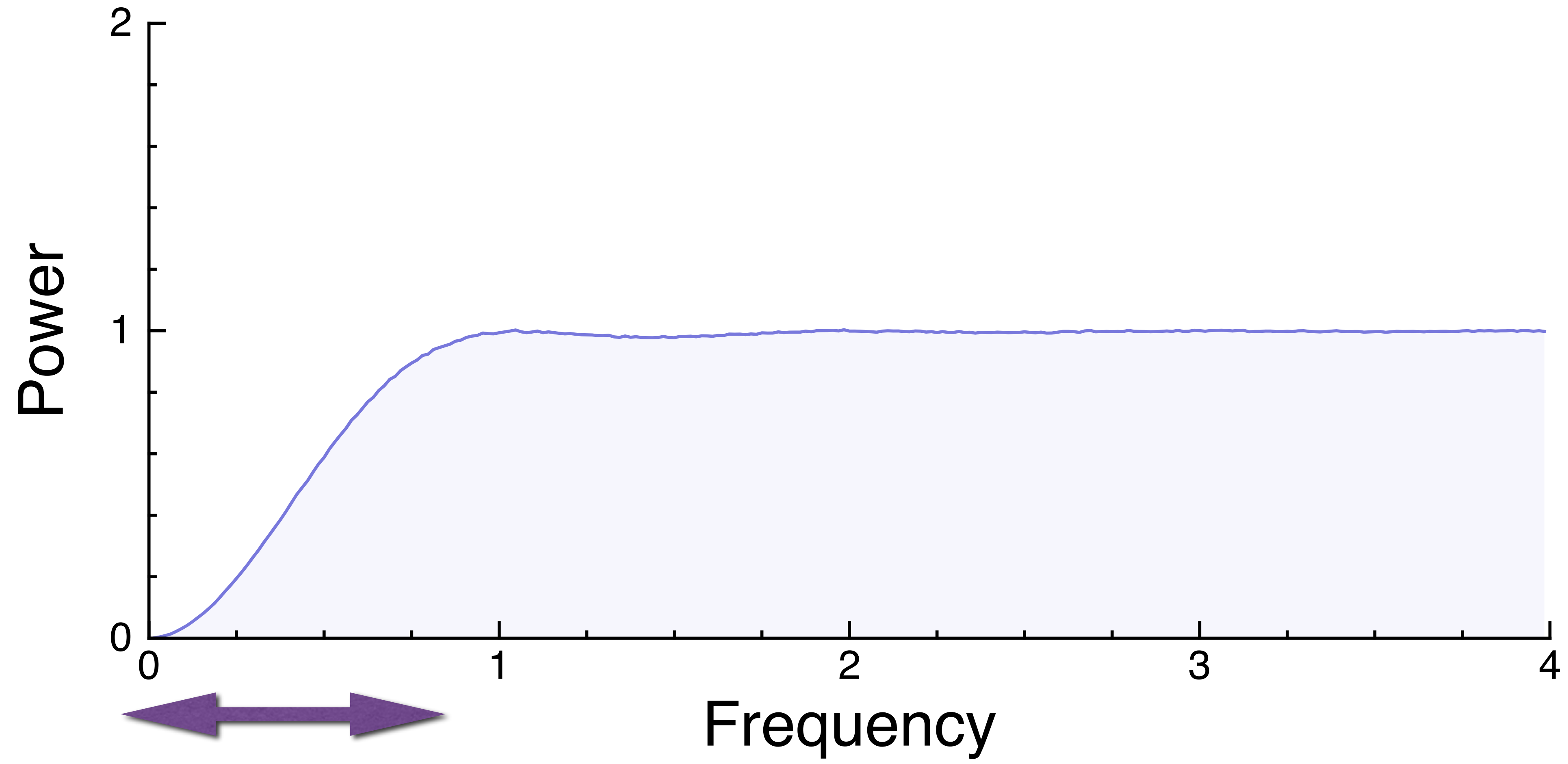


Poisson Disk

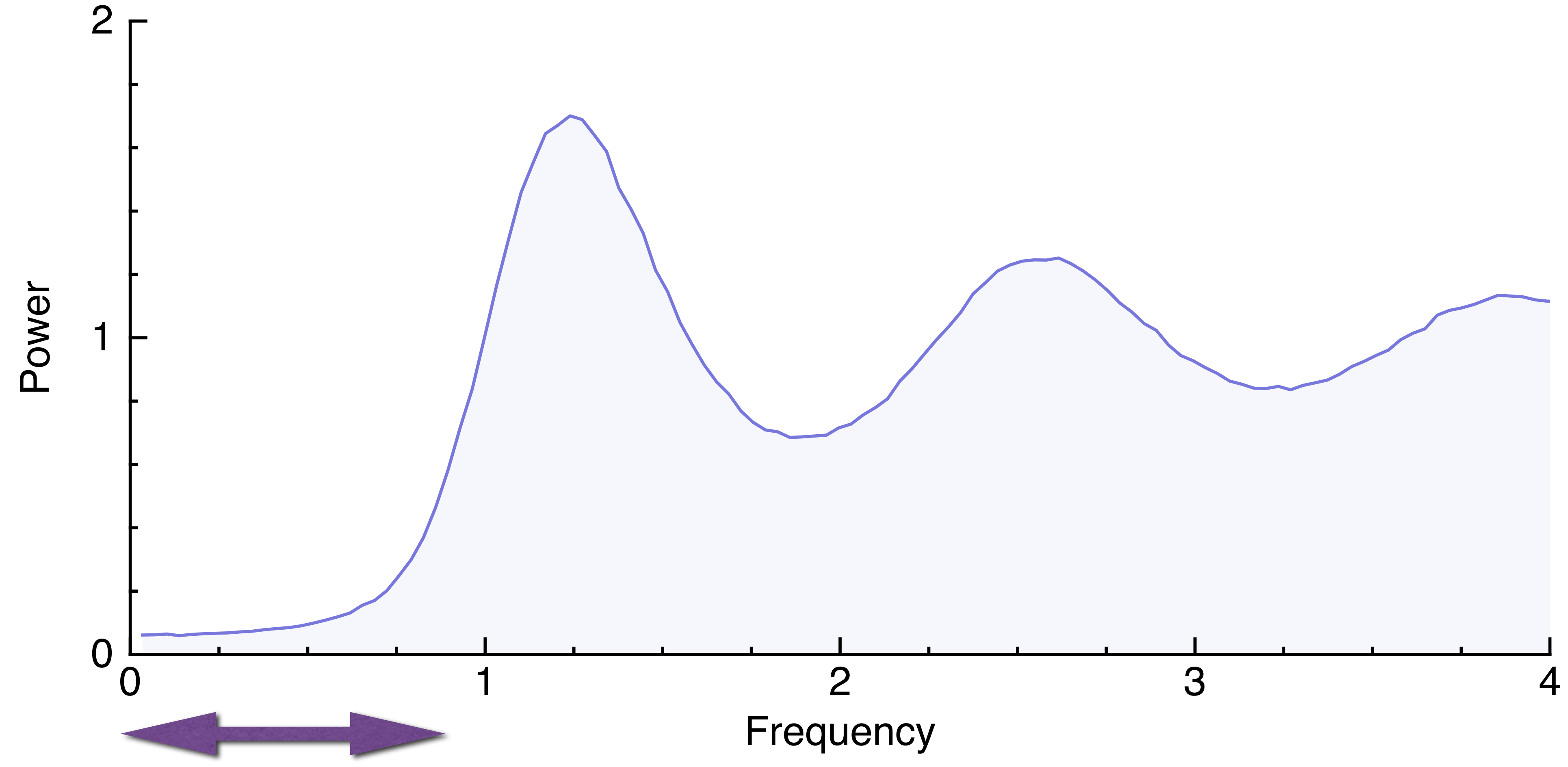


Low Frequency Region

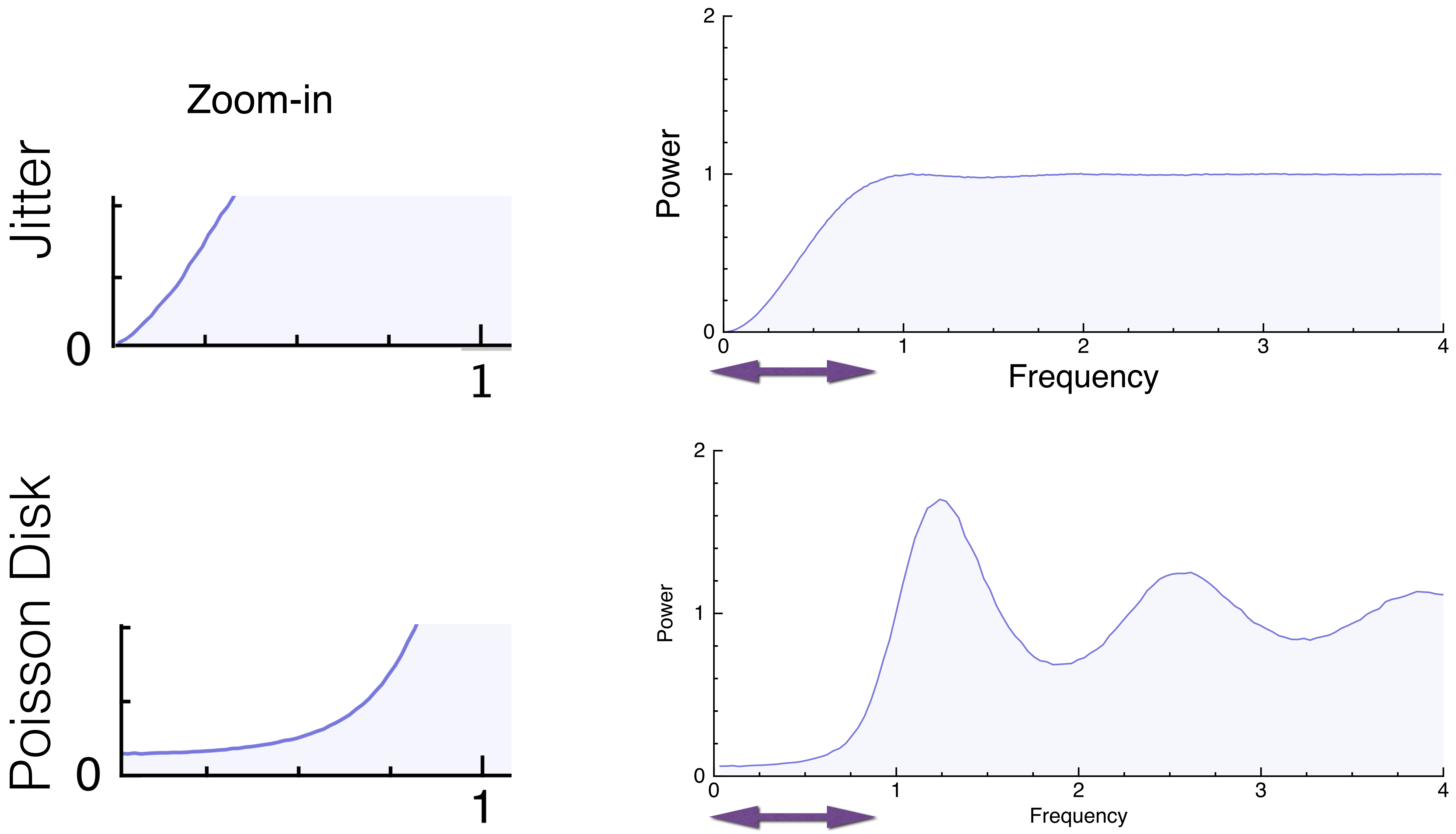
Jitter



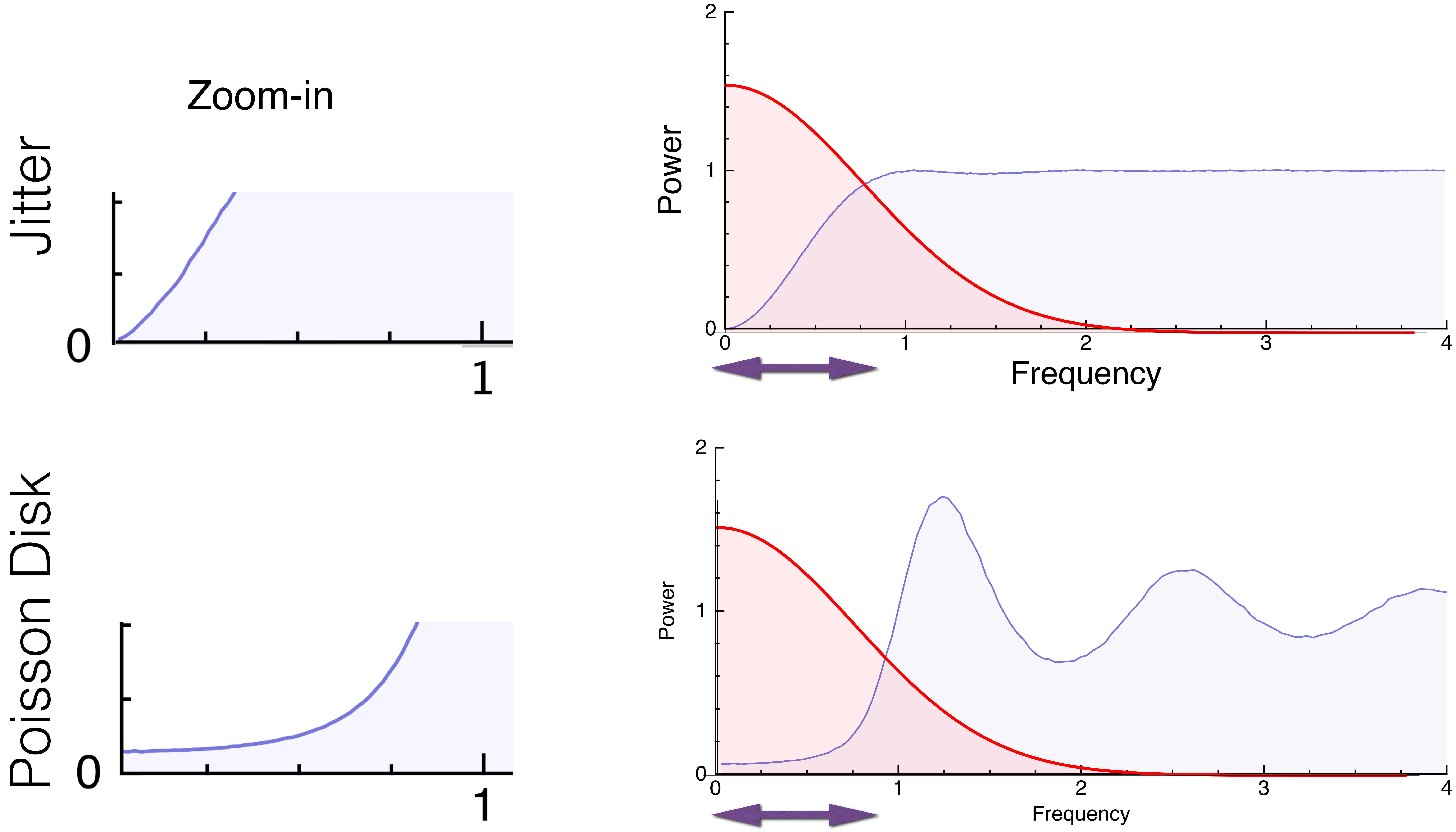
Poisson Disk



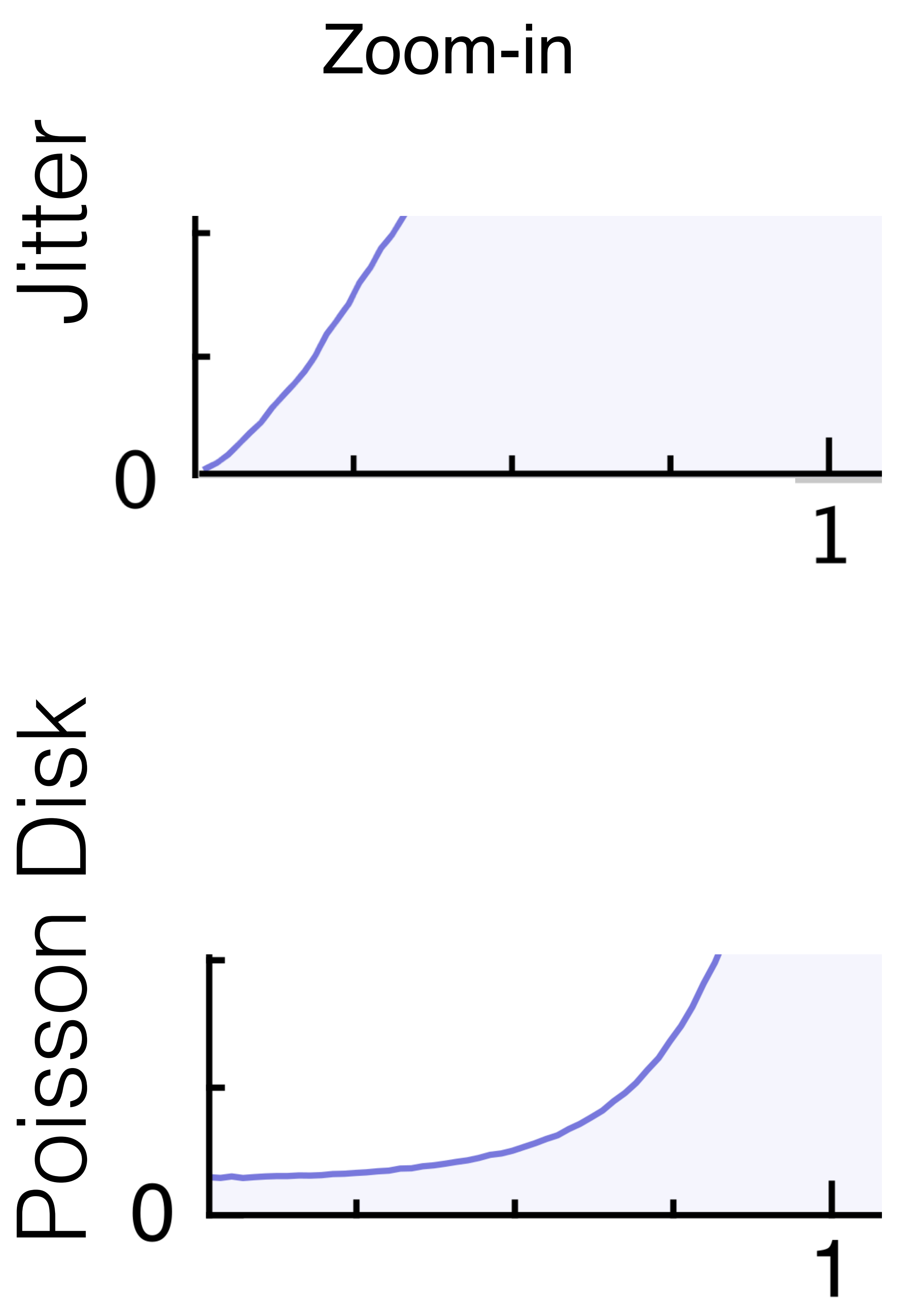
Low Frequency Region

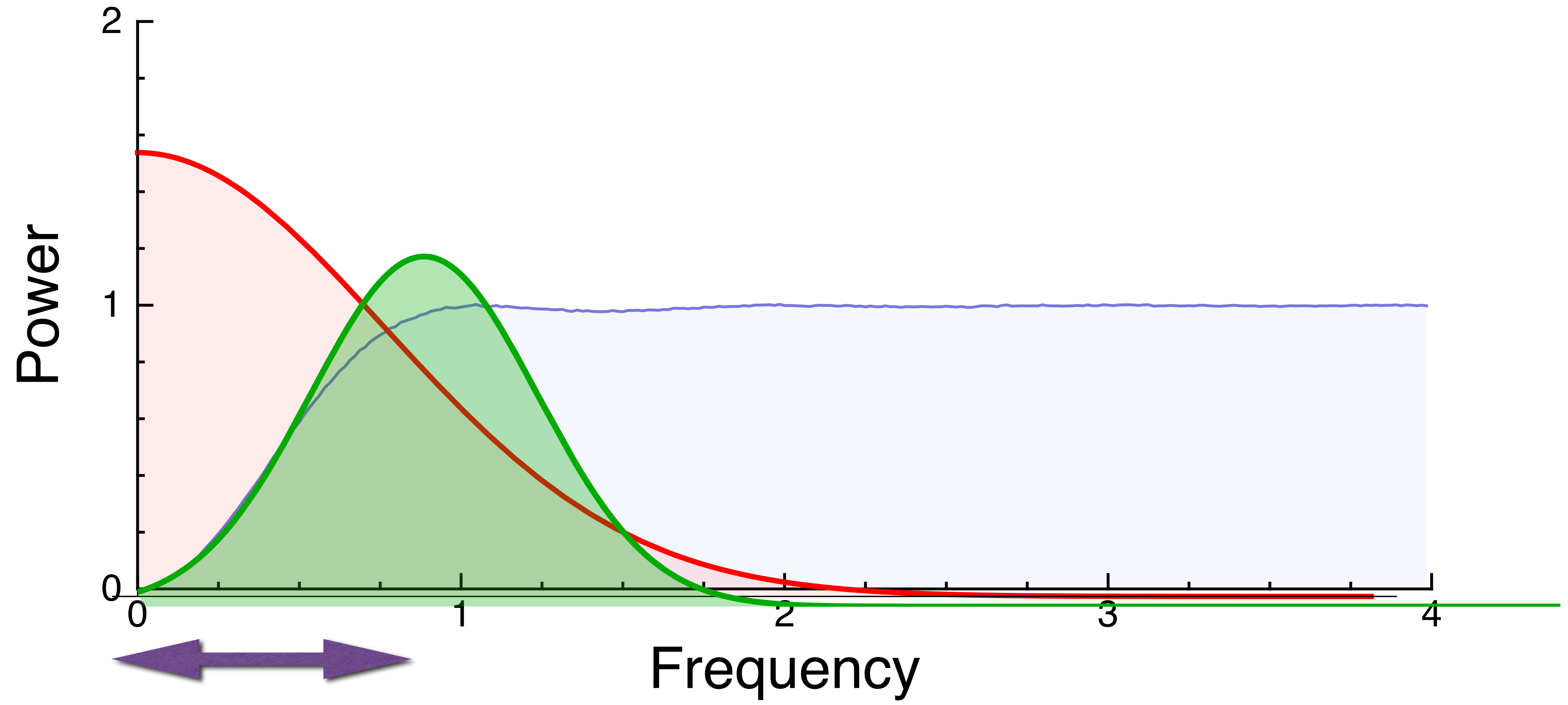


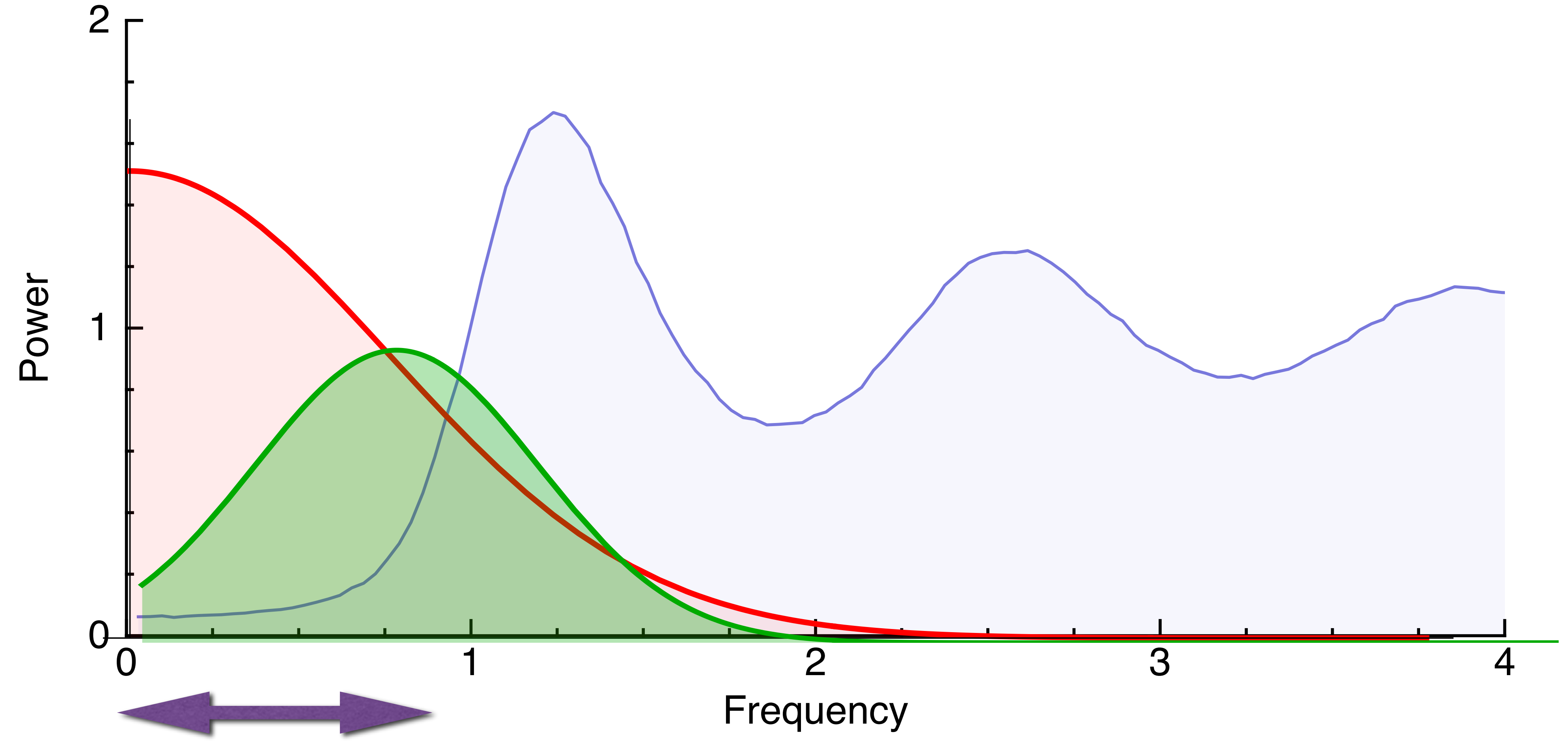
Variance for Low Sample Count



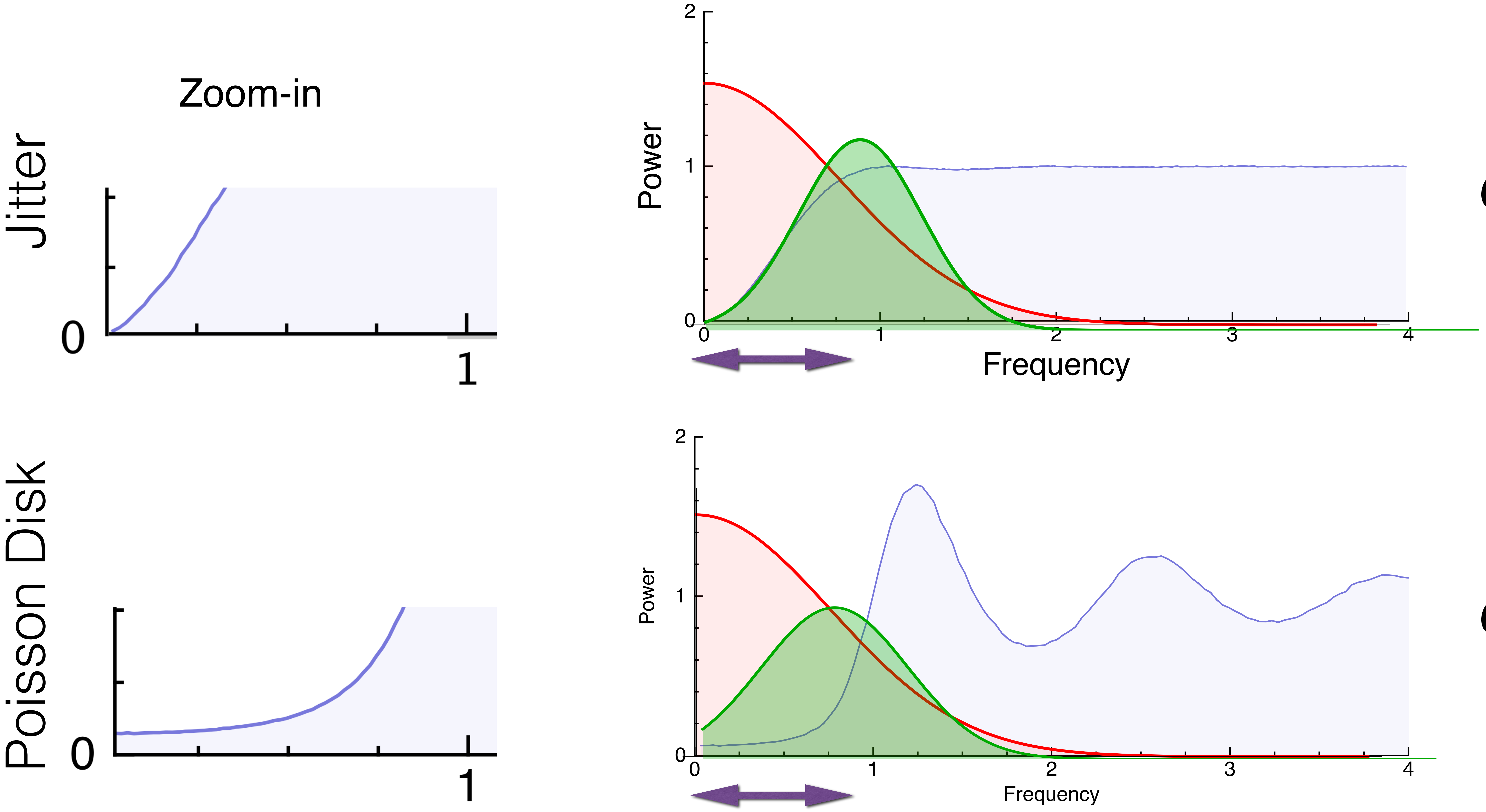
Variance for Low Sample Count







Variance for Increasing Sample Count

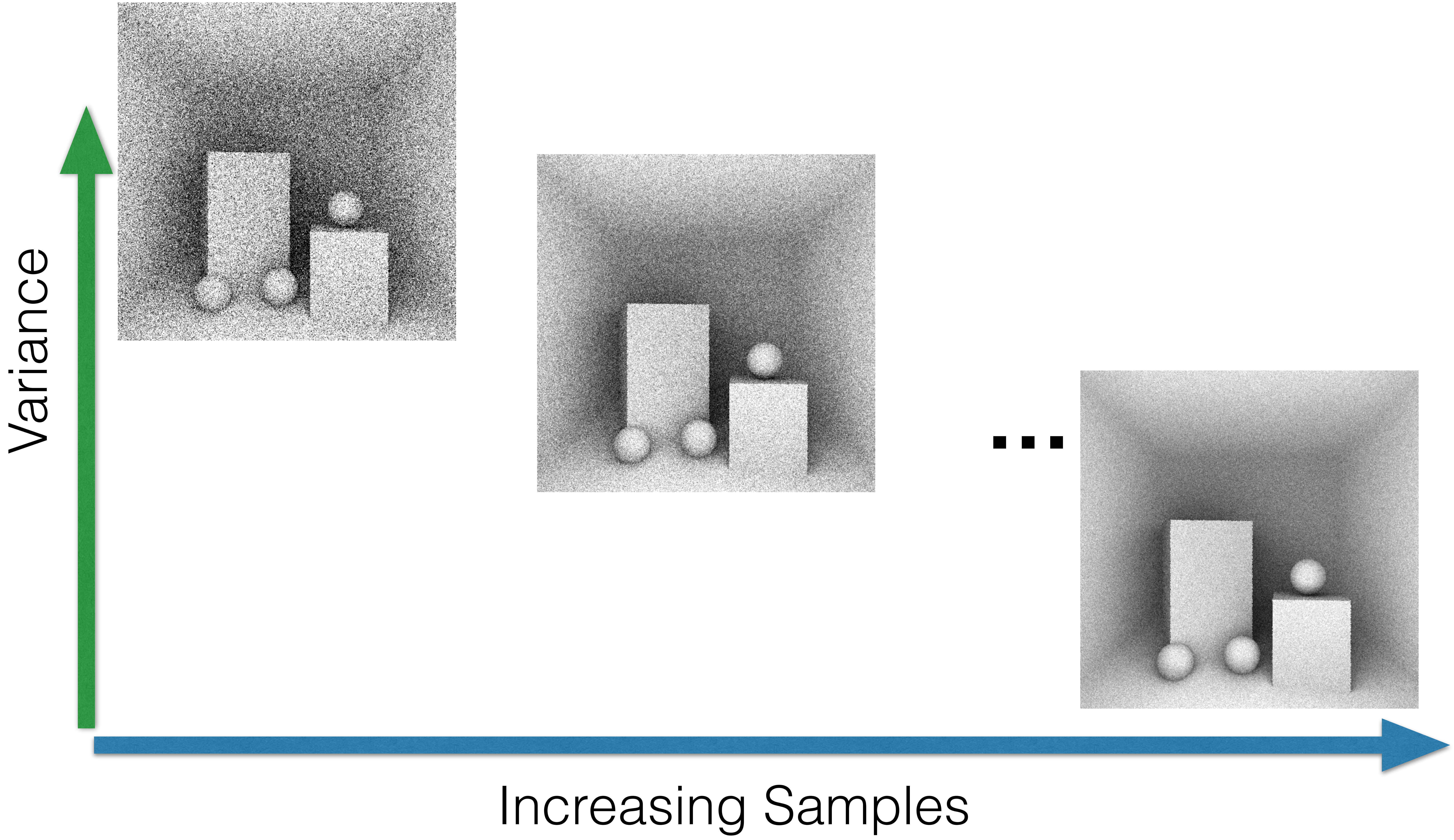


$\mathcal{O}(N^{-2})$

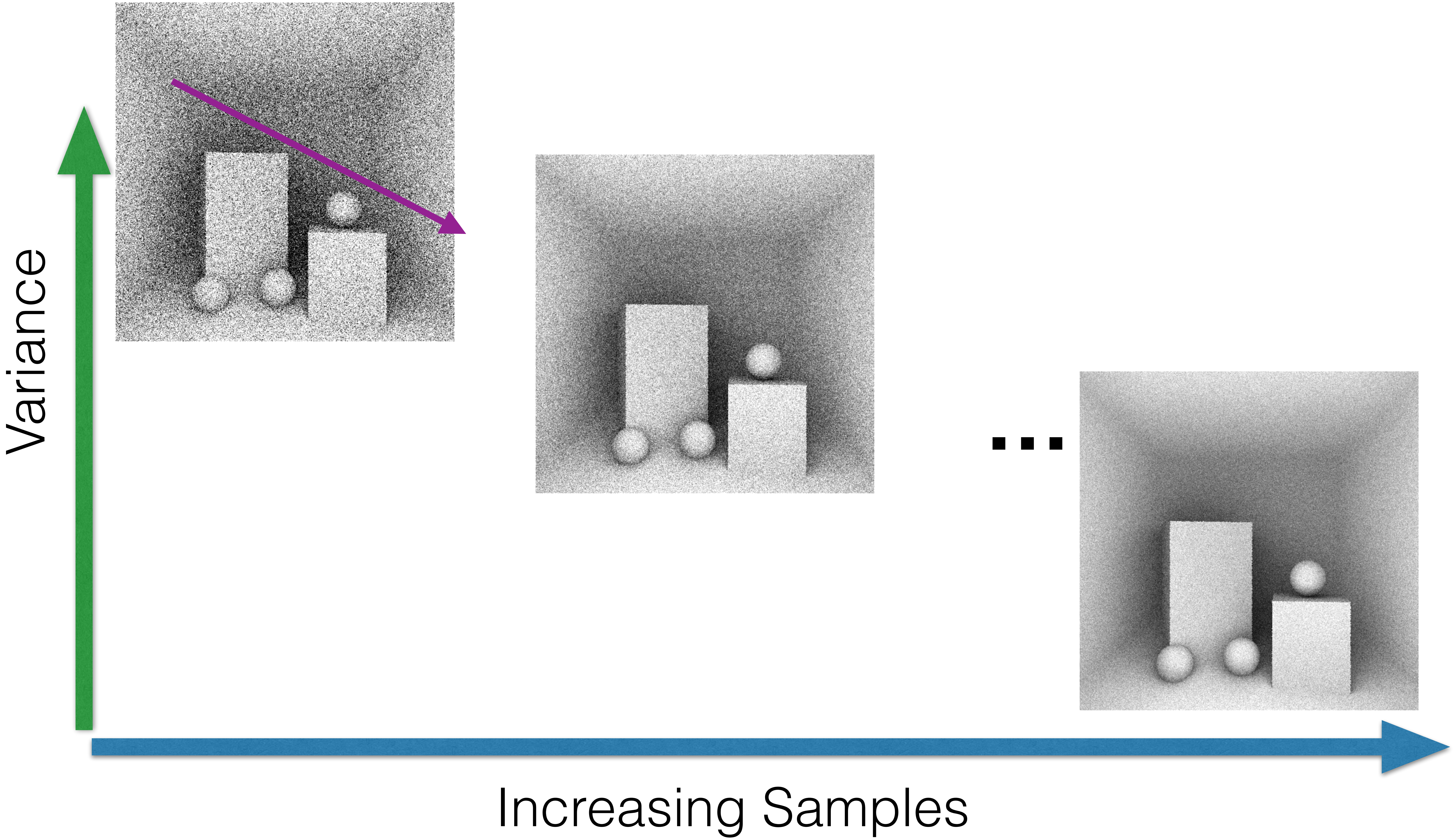
$\mathcal{O}(N^{-1})$

Experimental Verification

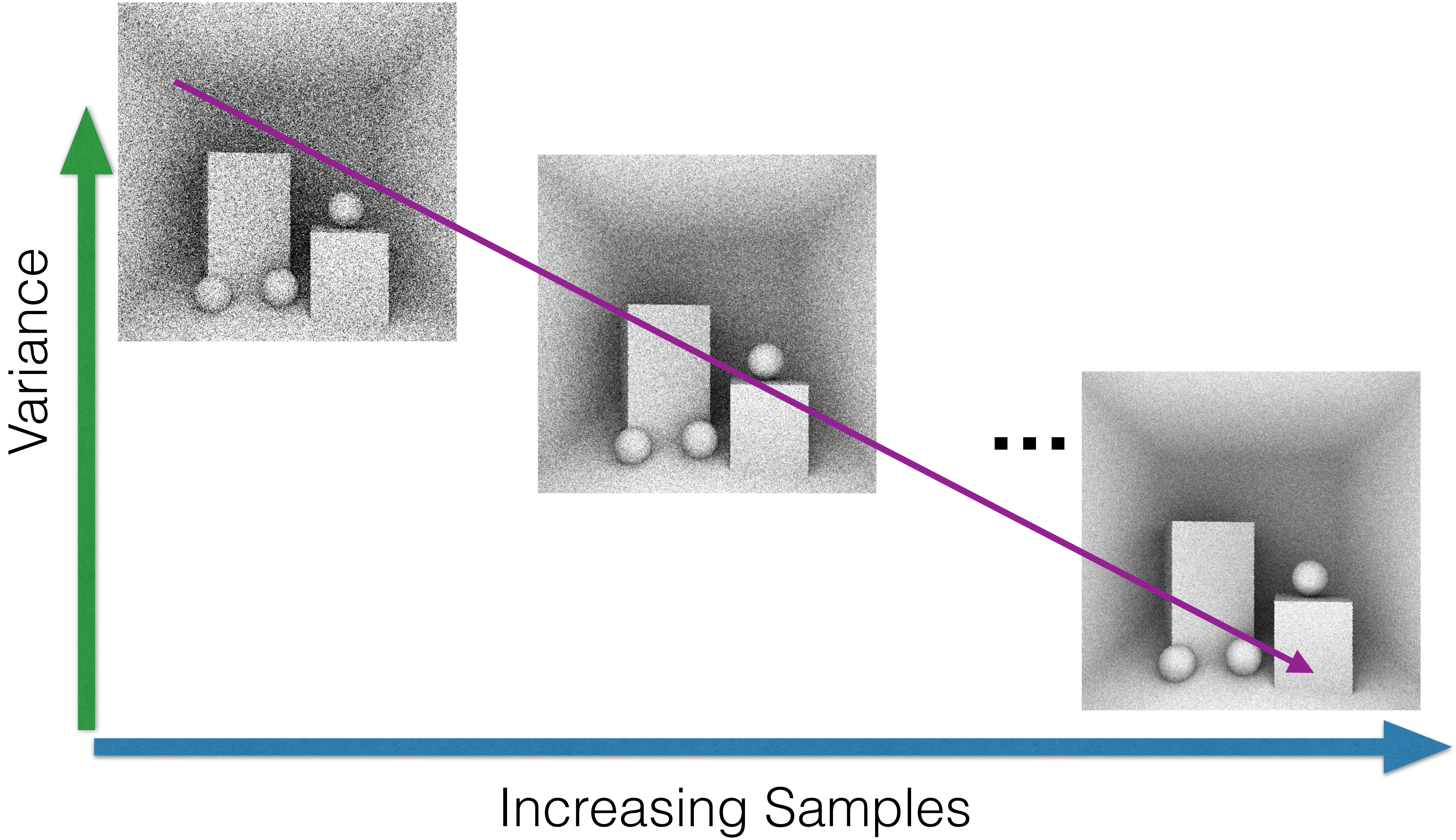
Convergence rate



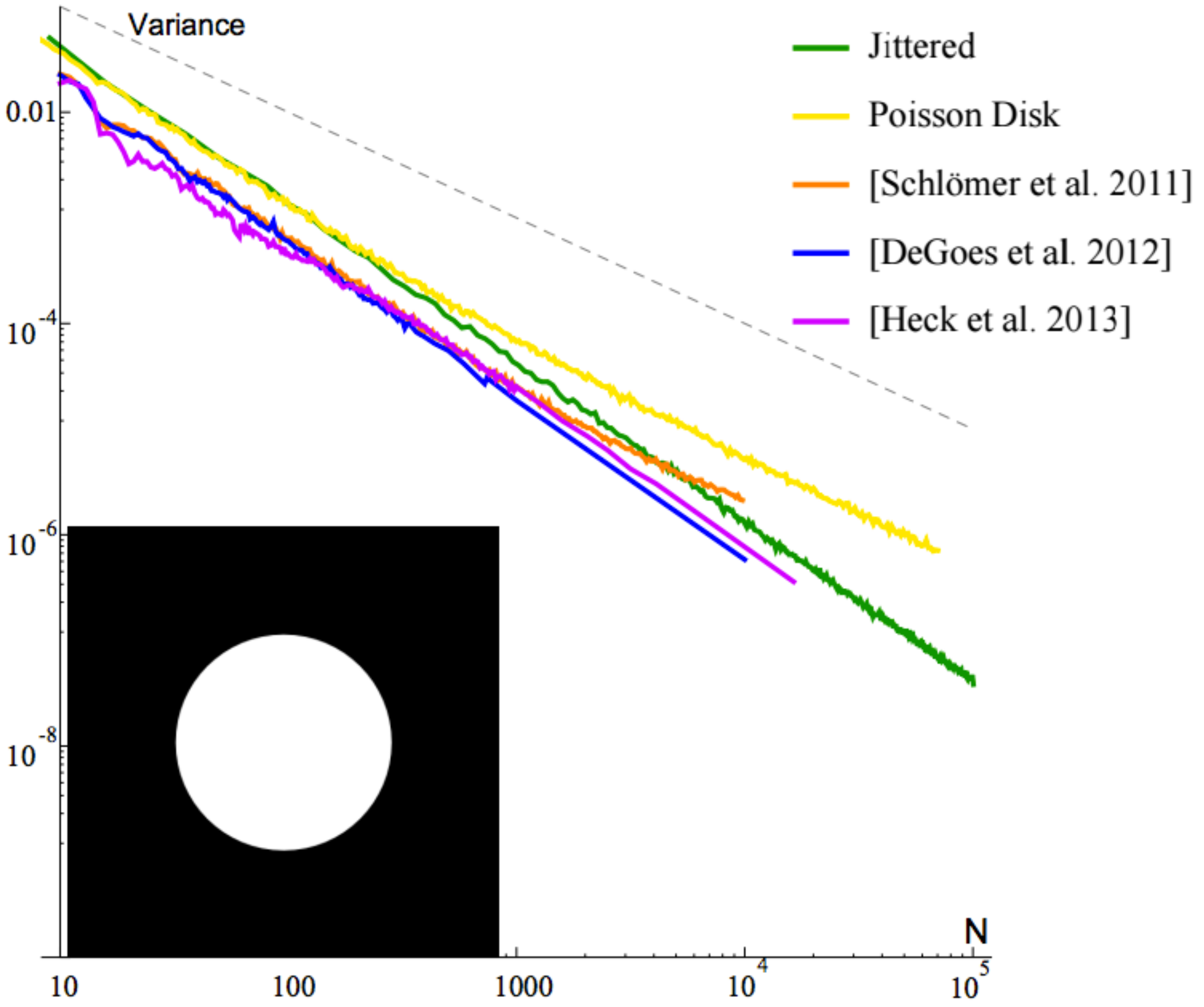
Convergence rate



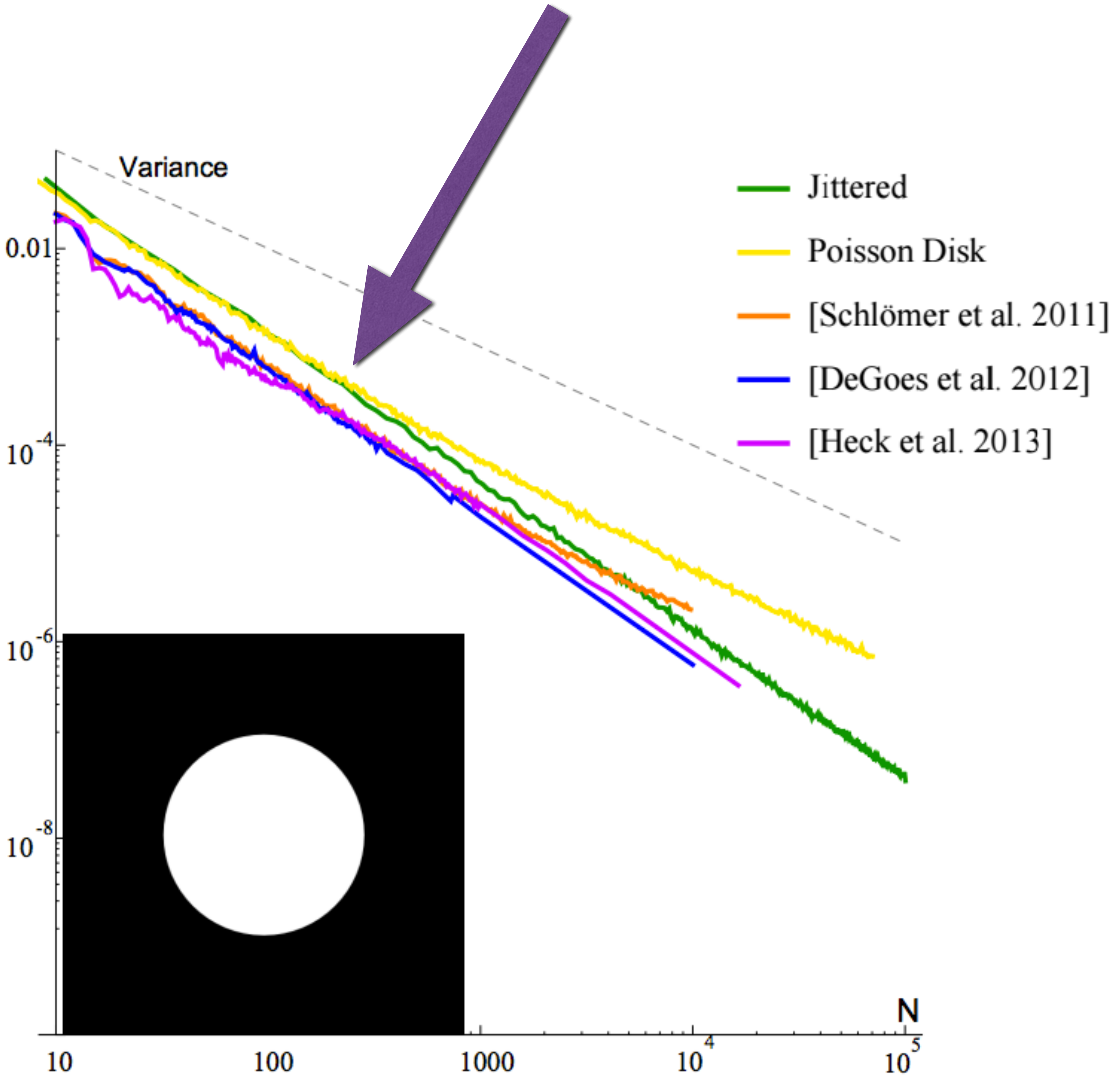
Convergence rate



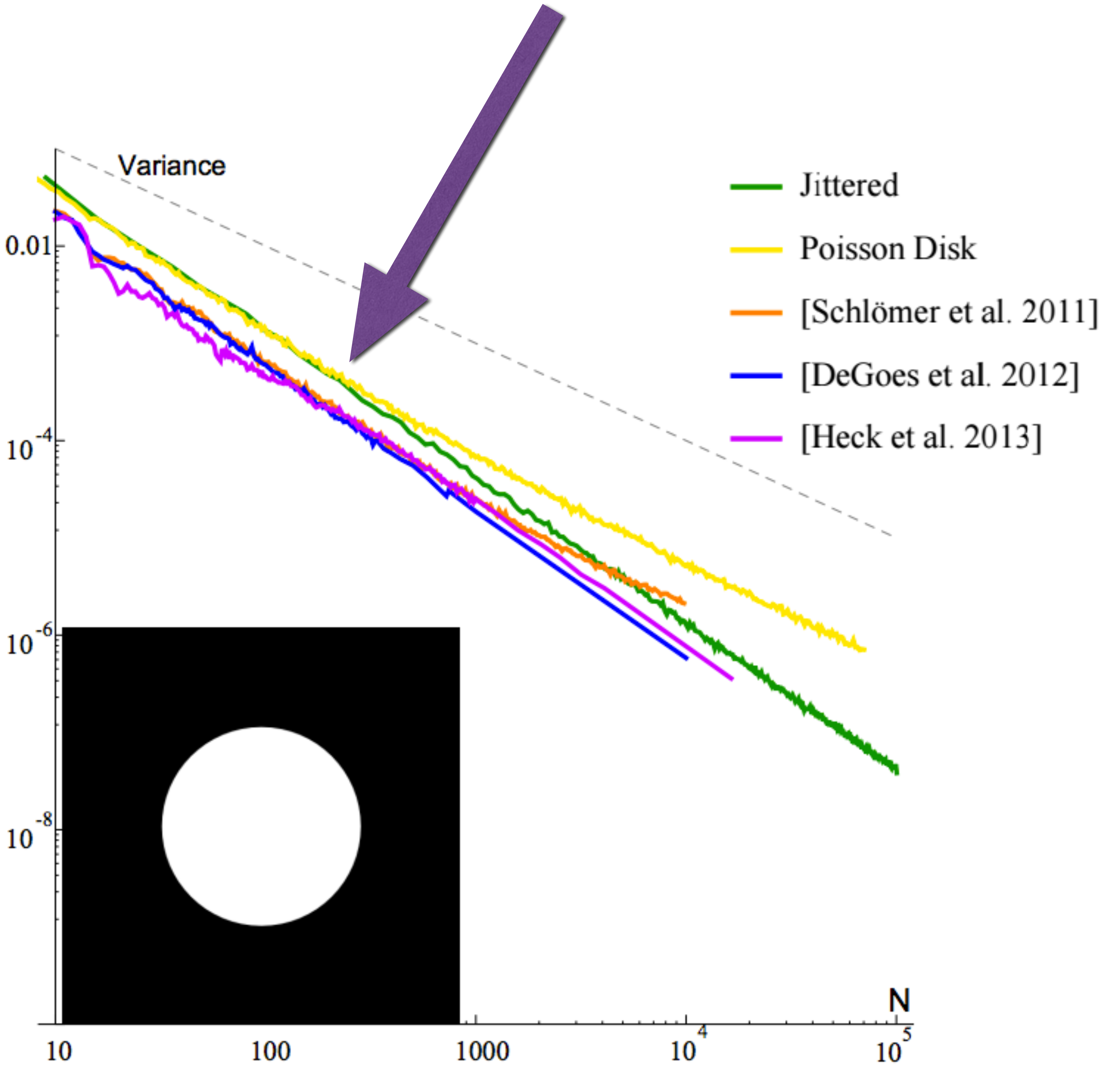
Disk Function as Worst Case



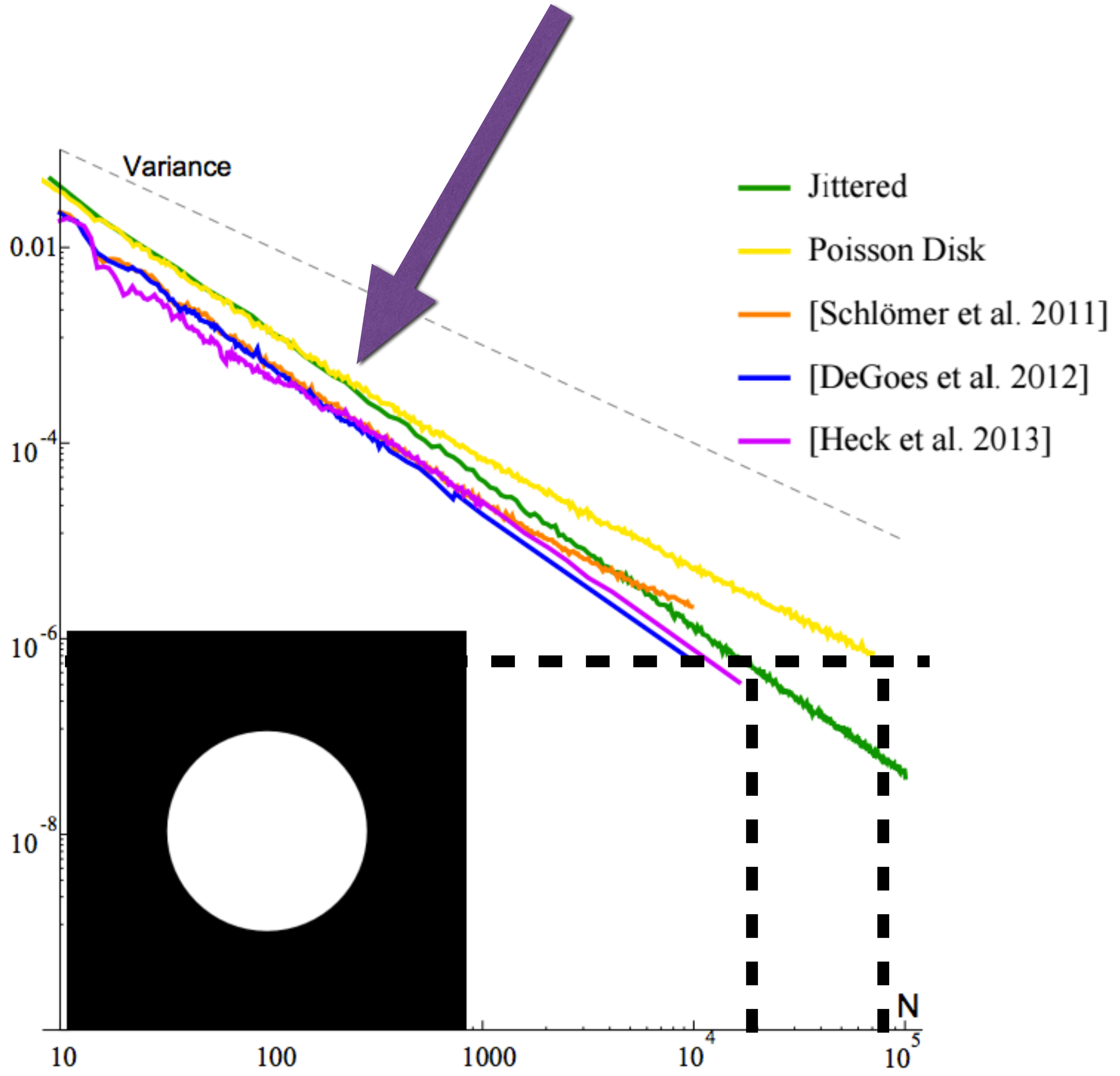
Disk Function as Worst Case



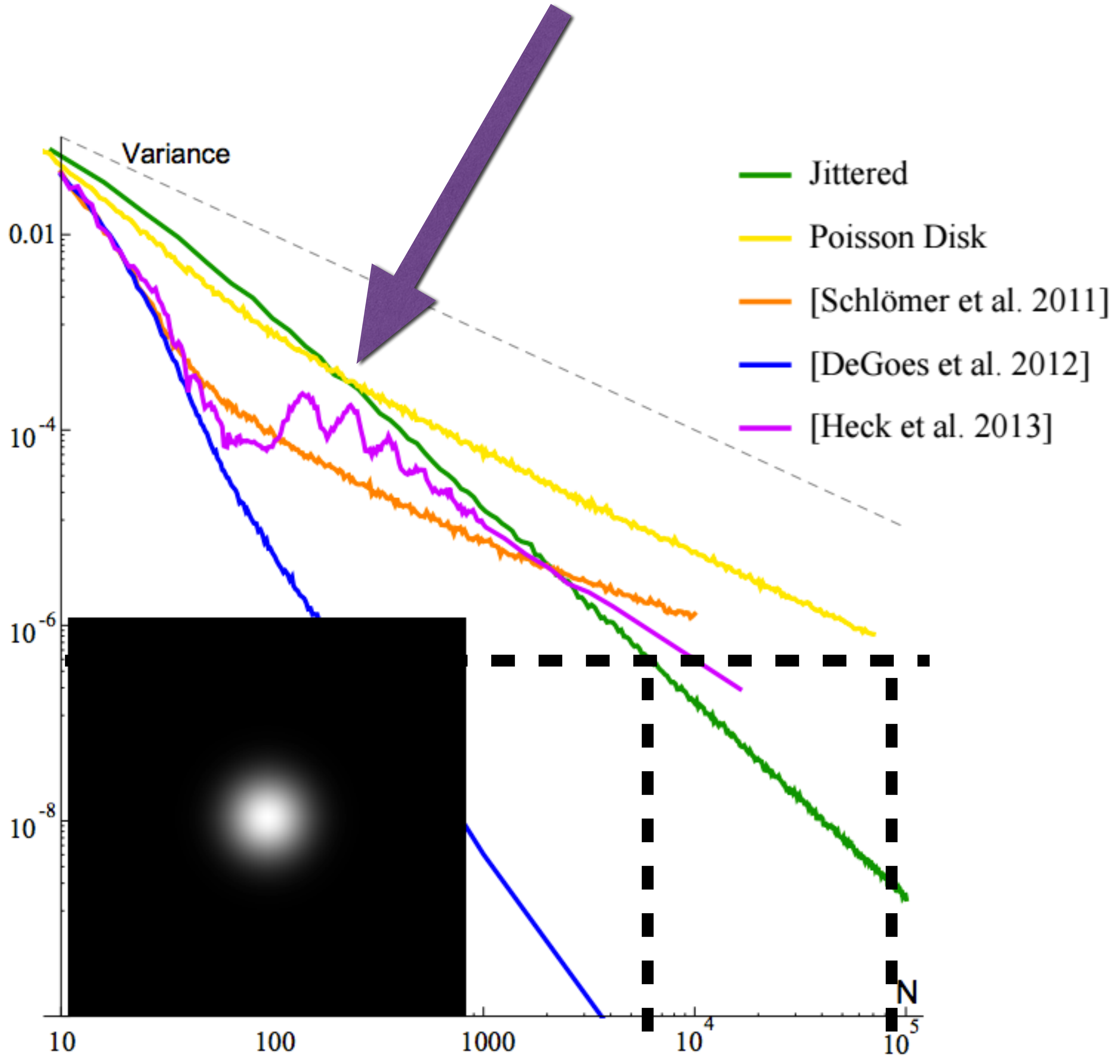
Disk Function as Worst Case



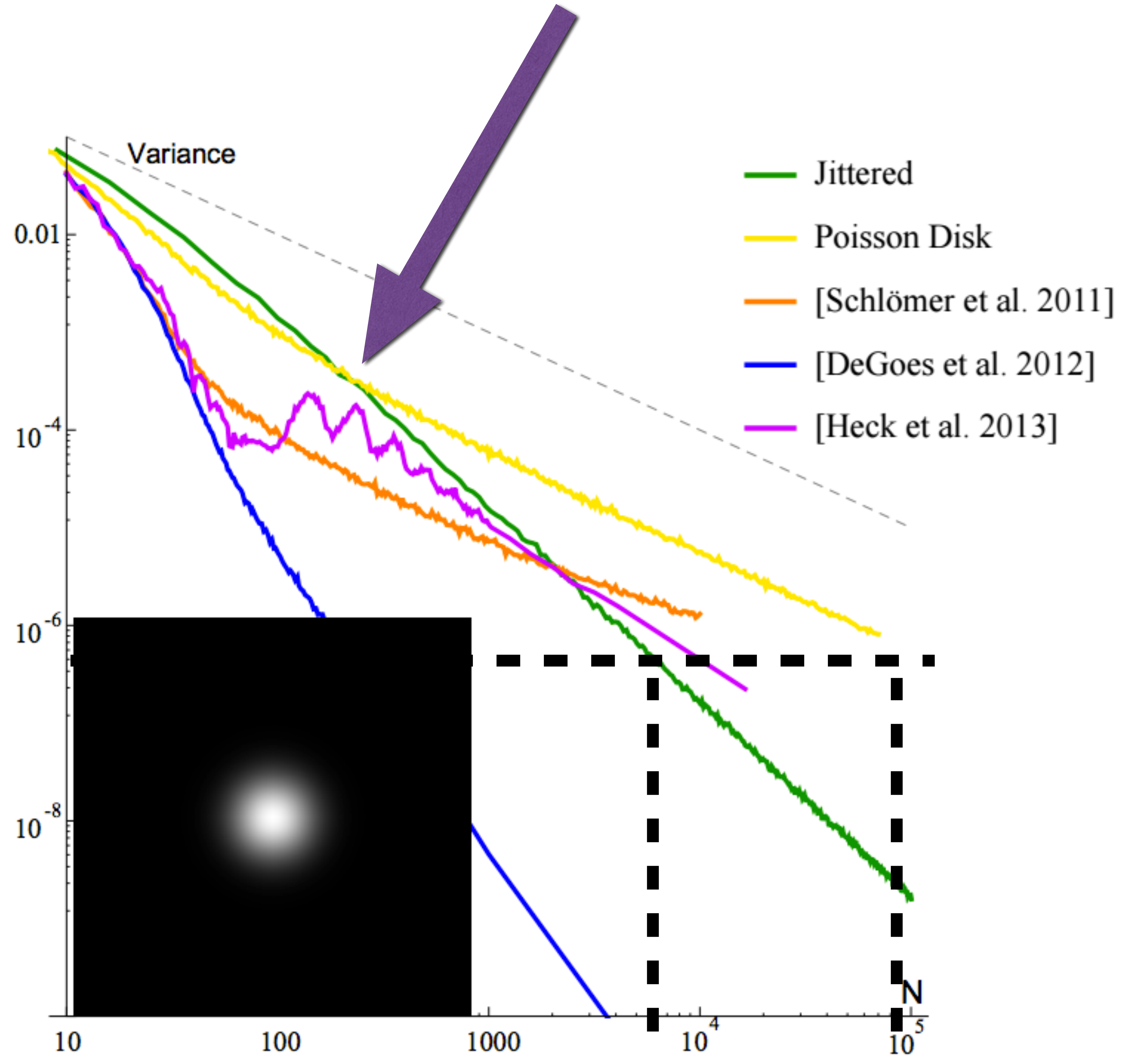
Disk Function as Worst Case



Gaussian as Best Case



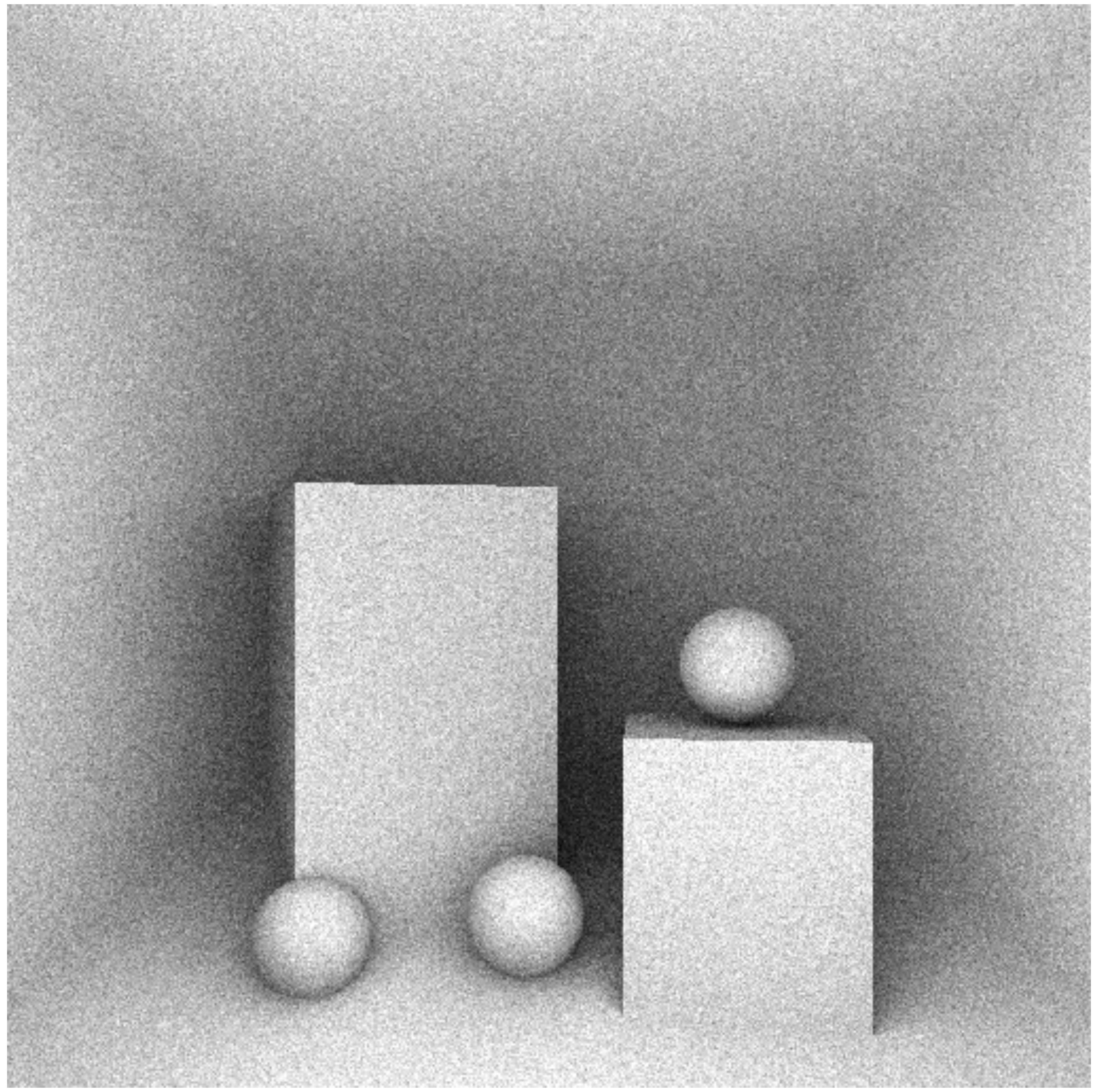
Gaussian as Best Case



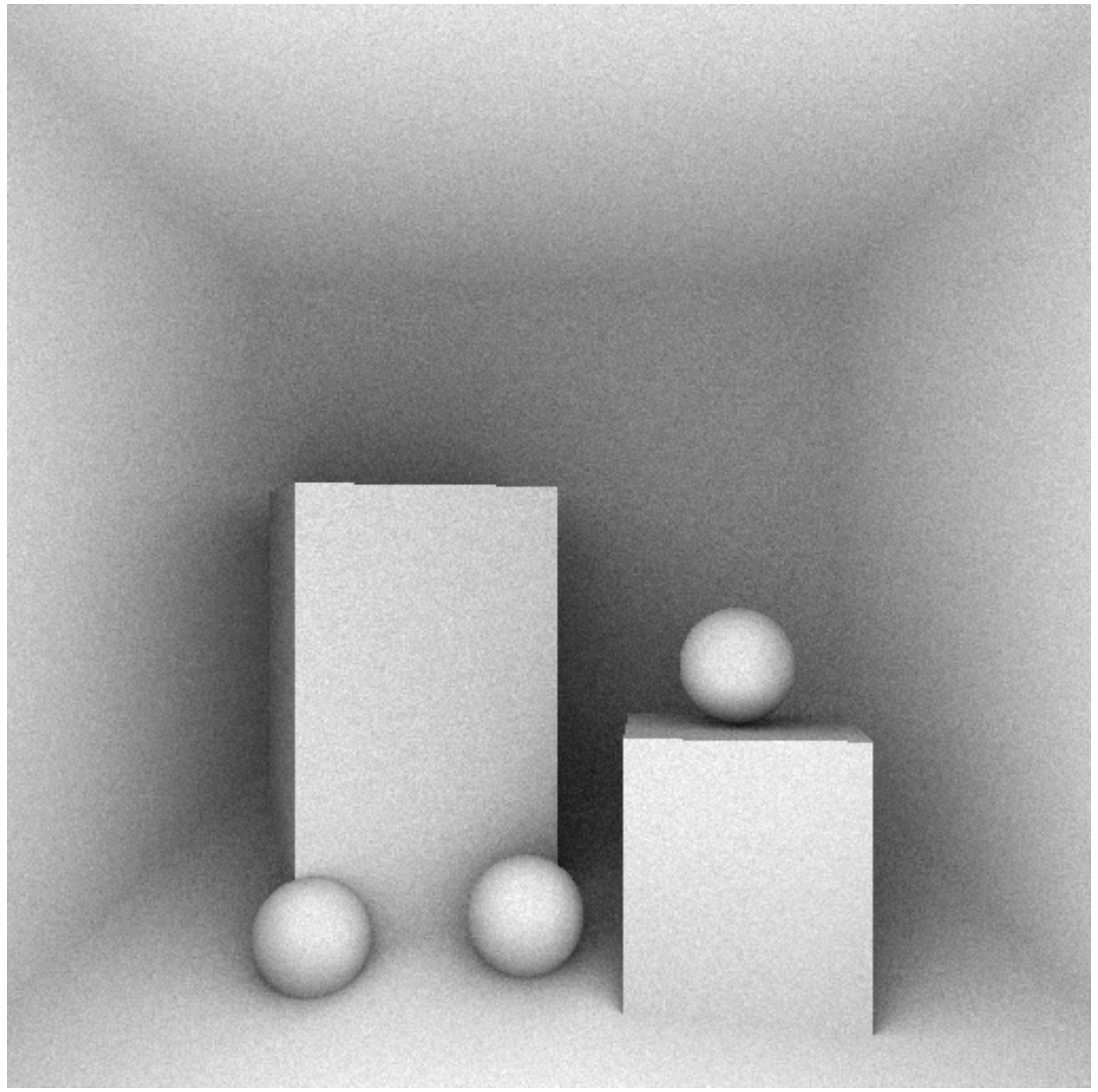
Ambient Occlusion Examples

Random vs Jittered

96 Secondary Rays



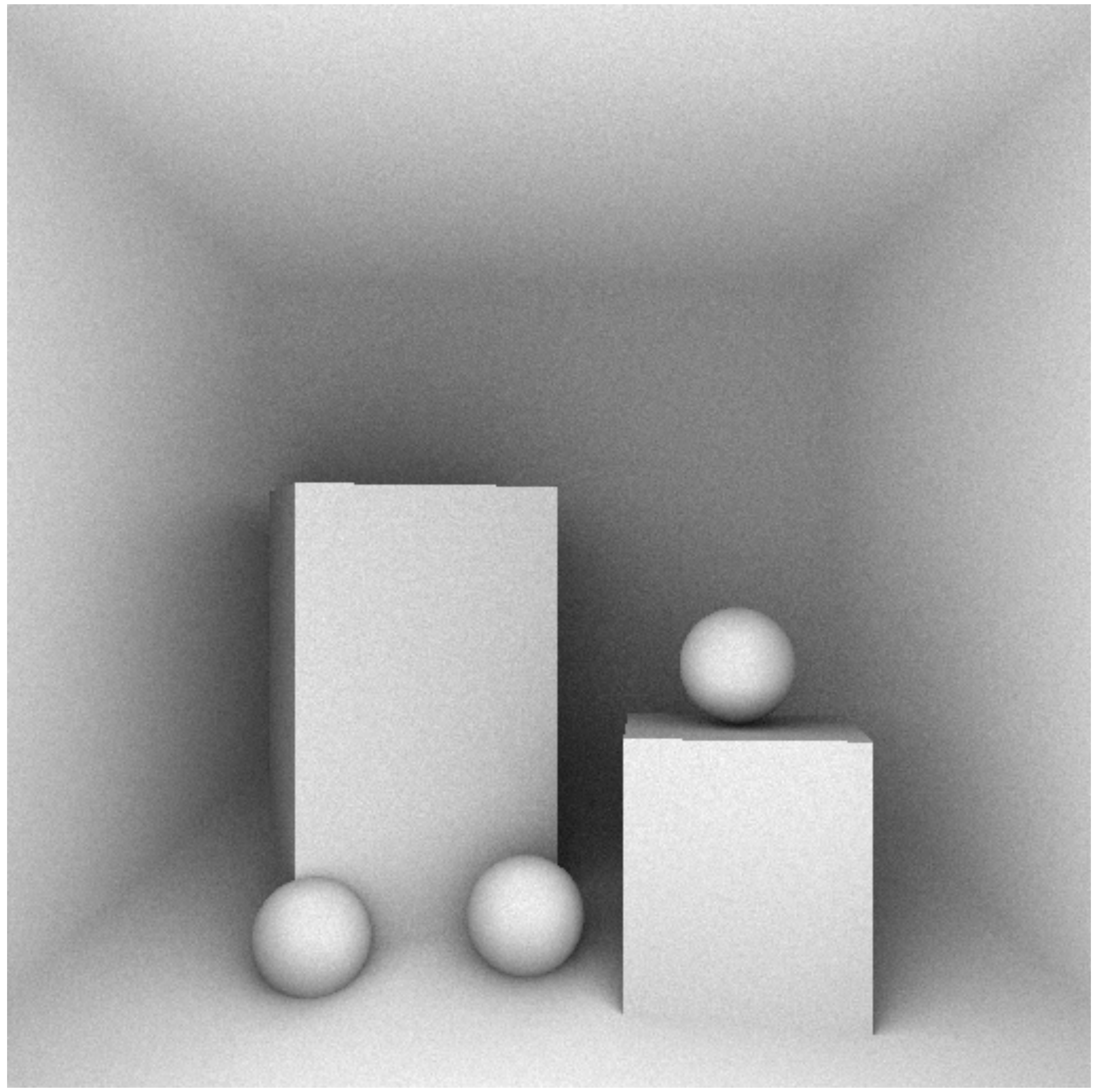
MSE: 4.74 x 10e-3



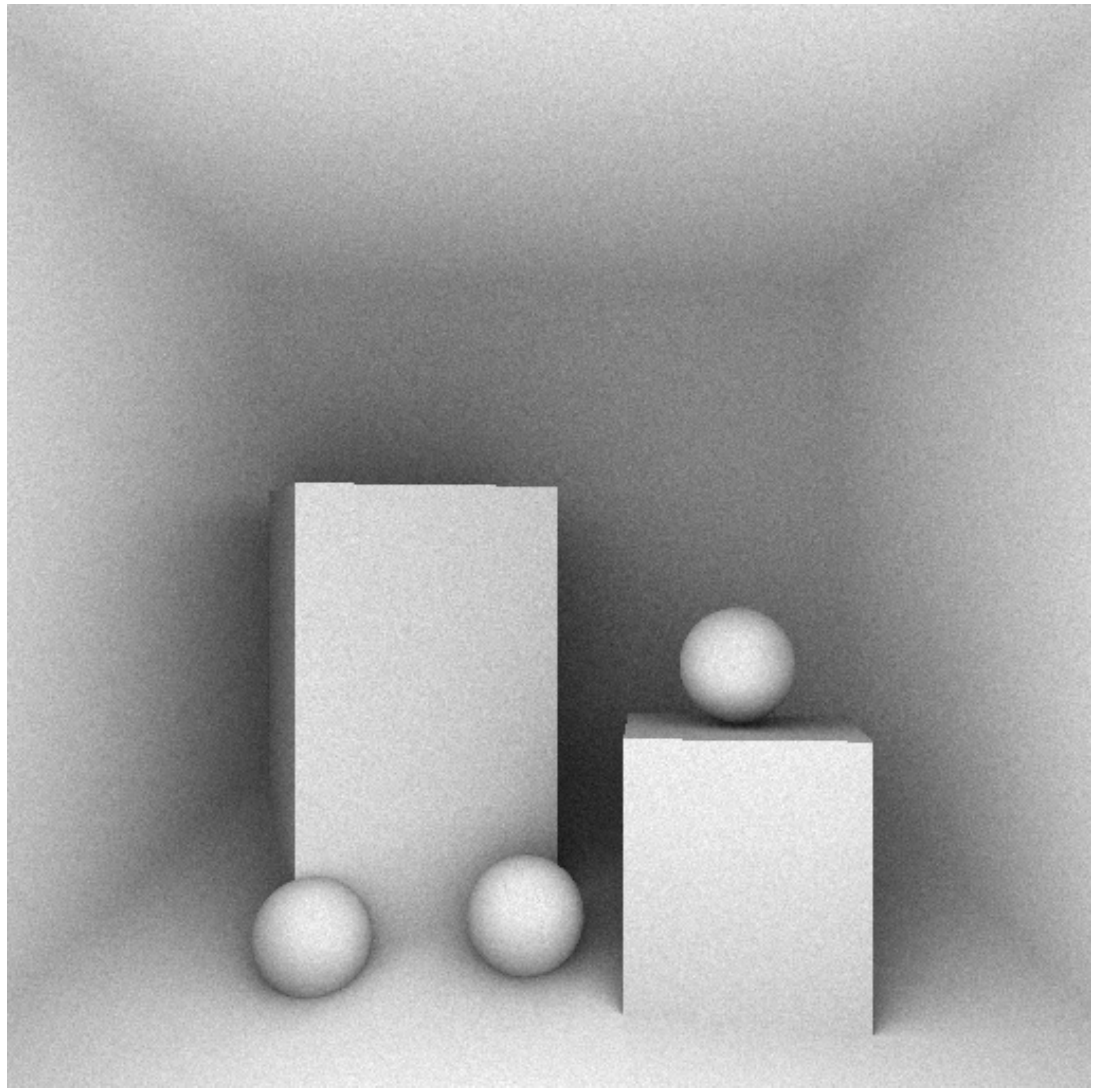
MSE: 8.56 x 10e-4

CCVT vs. Poisson Disk

96 Secondary Rays

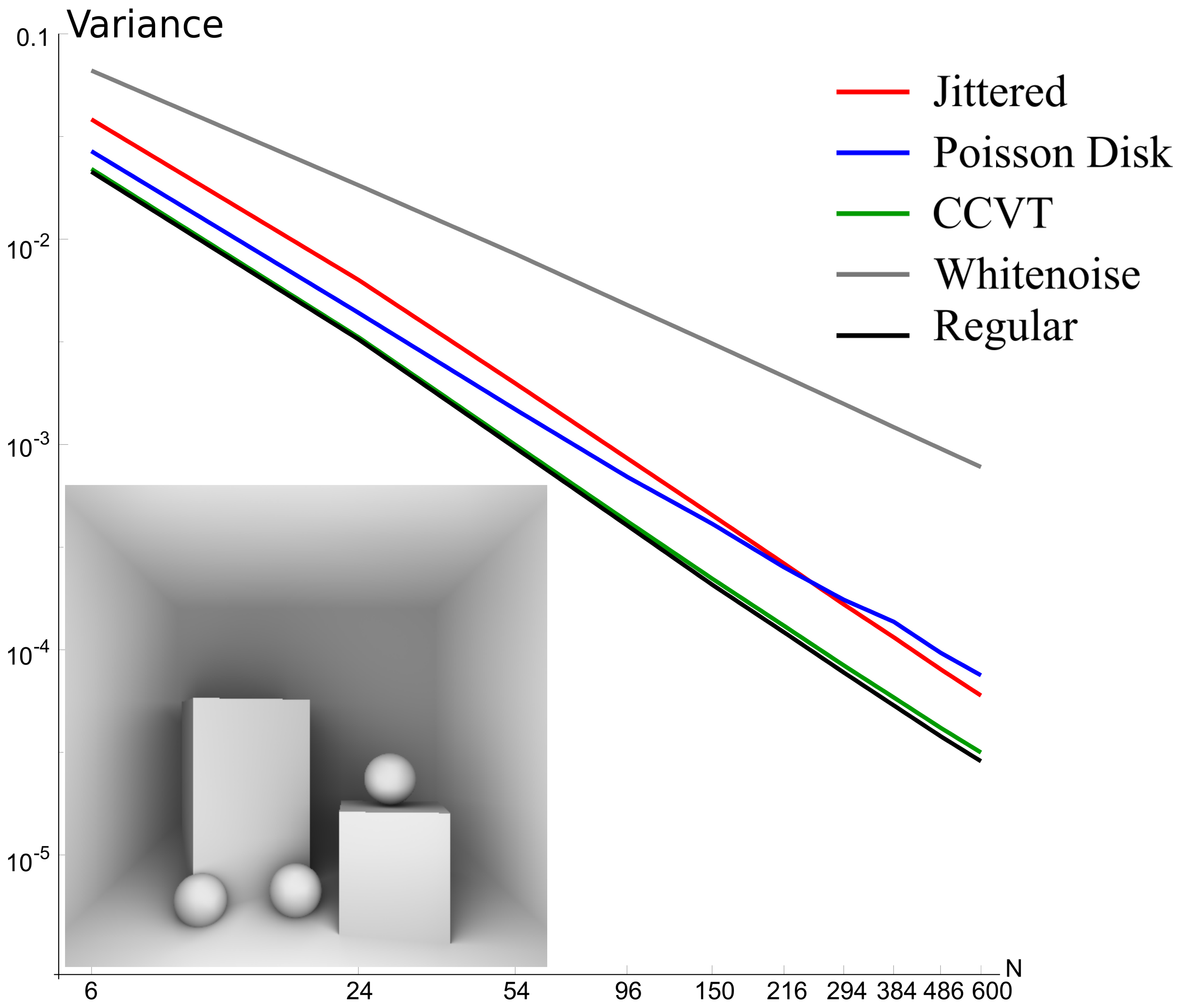


MSE: 4.24 x 10e-4

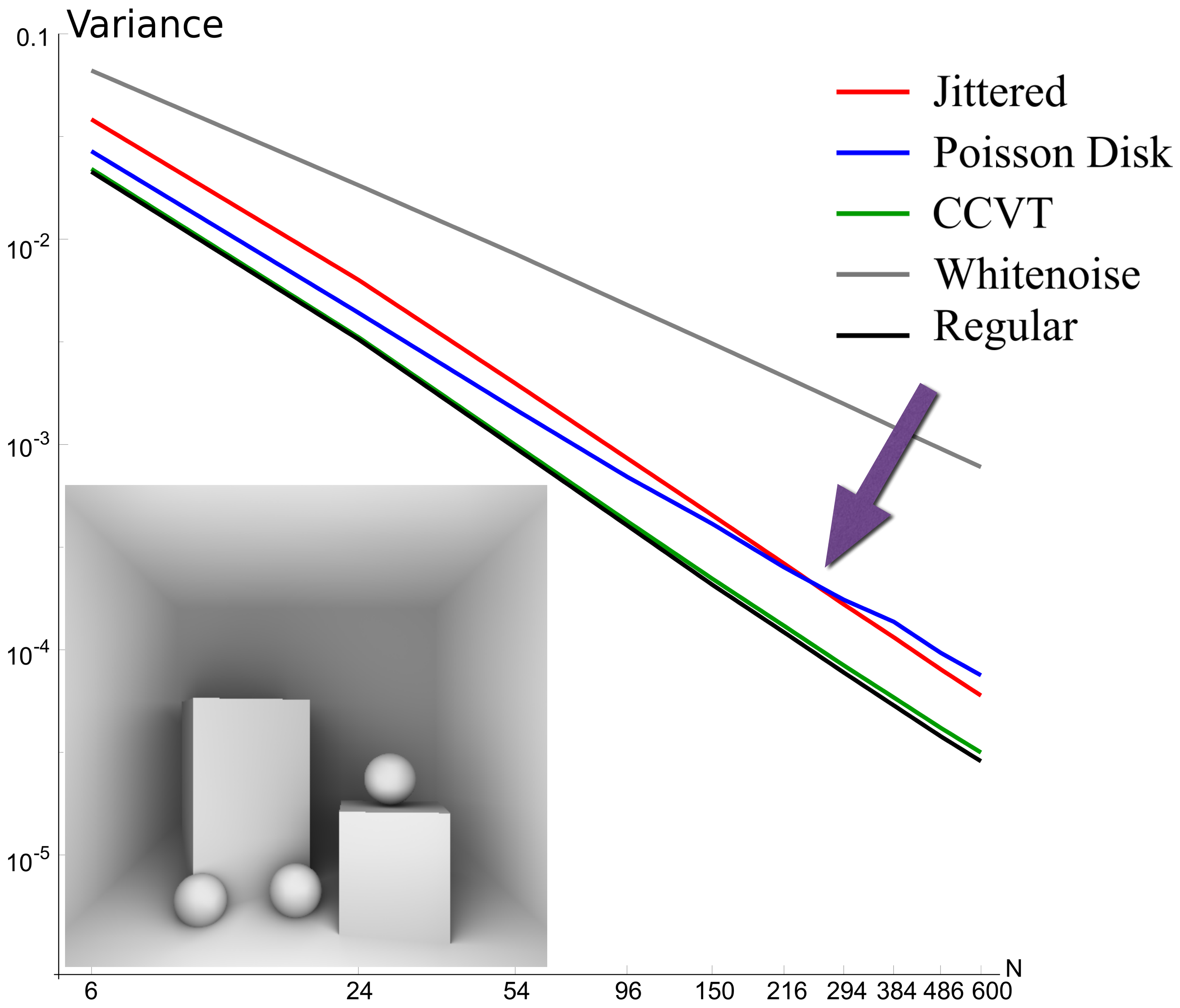


MSE: 6.95 x 10e-4

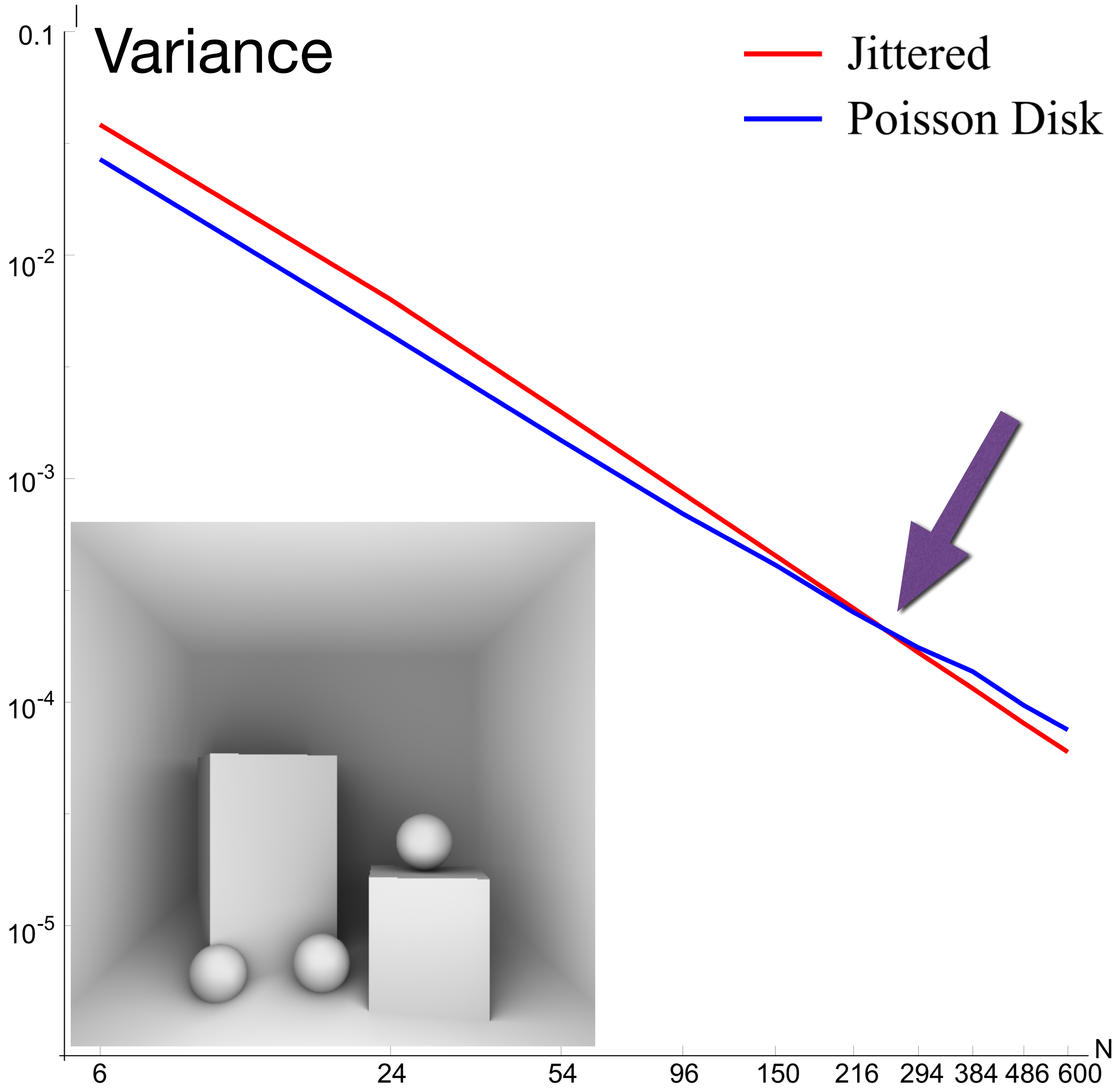
Convergence rates



Convergence rates



Jittered vs Poisson Disk



What are the benefits of this analysis ?

What are the benefits of this analysis ?

- For offline rendering, analysis tells which samplers would converge faster.

What are the benefits of this analysis ?

- For offline rendering, analysis tells which samplers would converge faster.
- For real time rendering, blue noise samples are more effective in reducing variance for a given number of samples

Acknowledgements

Fourier Analysis of Numerical Integration in Monte Carlo Rendering

Kartic Subr Gurprit Singh *Wojciech Jarosz

Render the Possibilities SIGGRAPH2016

*First part of slides are from Wojciech Jarosz