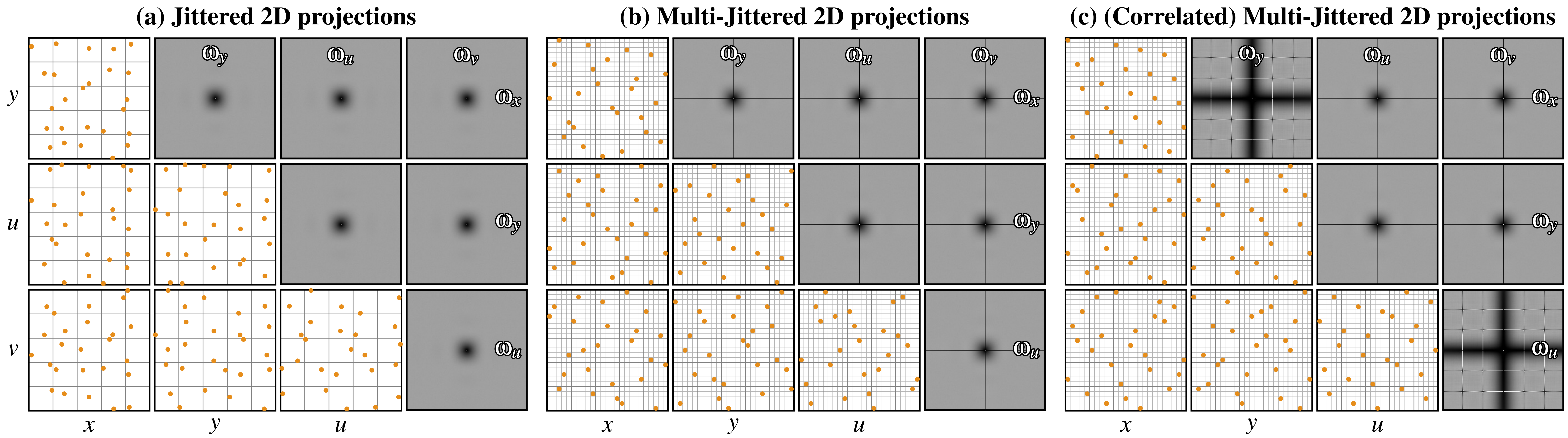
ORTHOGONAL ARRAY SAMPLING FOR MONTE CARLO RENDERING



Wojciech Jarosz Afnan Enayet Andrew Kensler Charlie Kilpatrick Per Christensen

DARTMOUTH VISUAL COMPUTING LAB

PIXAR ANIMATION STUDIOS

EGSR 2019 EUROGRAPHICS SYMPOSIUM ON RENDERING

$$F = \int_A \int_A \int_T \int_{\Omega_n \cdots \Omega_1} f(x, y, u, v, t, \vec{\omega}_1, \ldots, \vec{\omega}_n)\, \mathrm{d}\vec{\omega}_1 \cdots \mathrm{d}\vec{\omega}_n\, \mathrm{d}u\, \mathrm{d}v\, \mathrm{d}x\, \mathrm{d}y$$

Rendering = high-dimensional integration

$$F = \int_{[0,1]^d} f(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

Rendering = high-dimensional integration

$$F = \int_{[0,1]^d} f(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

Rendering = high-dimensional integration

PBRT Gallery

Monte Carlo

$$F = \int_{[0,1]^d} f(\mathbf{x})\, \mathrm{d}\mathbf{x} \qquad \approx \qquad F_N = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i)$$

Rendering = high-dimensional integration

PBRT Gallery

Monte Carlo

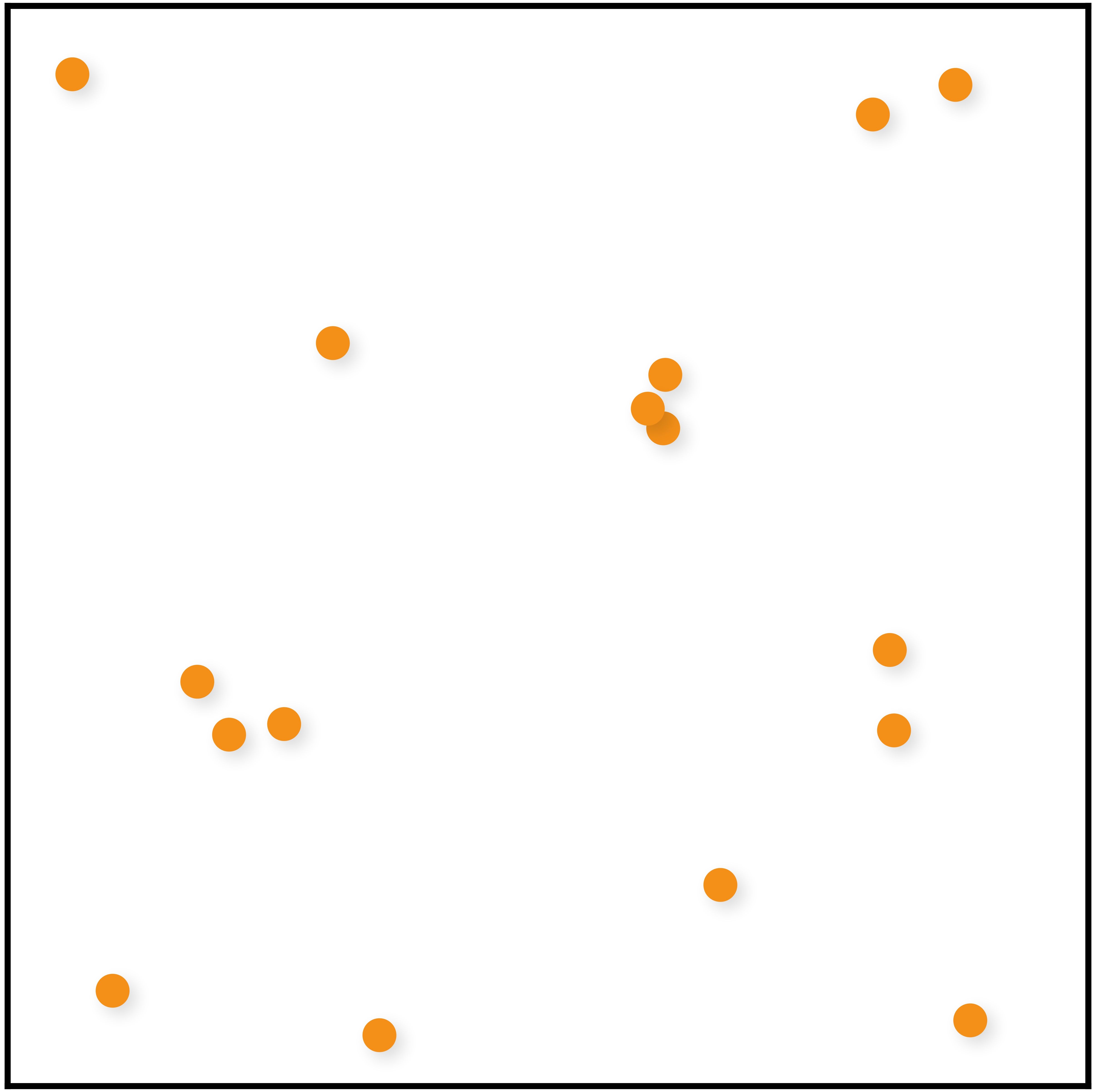
$$F = \int_{[0,1]^d} f(\mathbf{x})\, \mathrm{d}\mathbf{x} \qquad \approx \qquad F_N = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i)$$

Rendering = high-dimensional integration

Independent random sampling

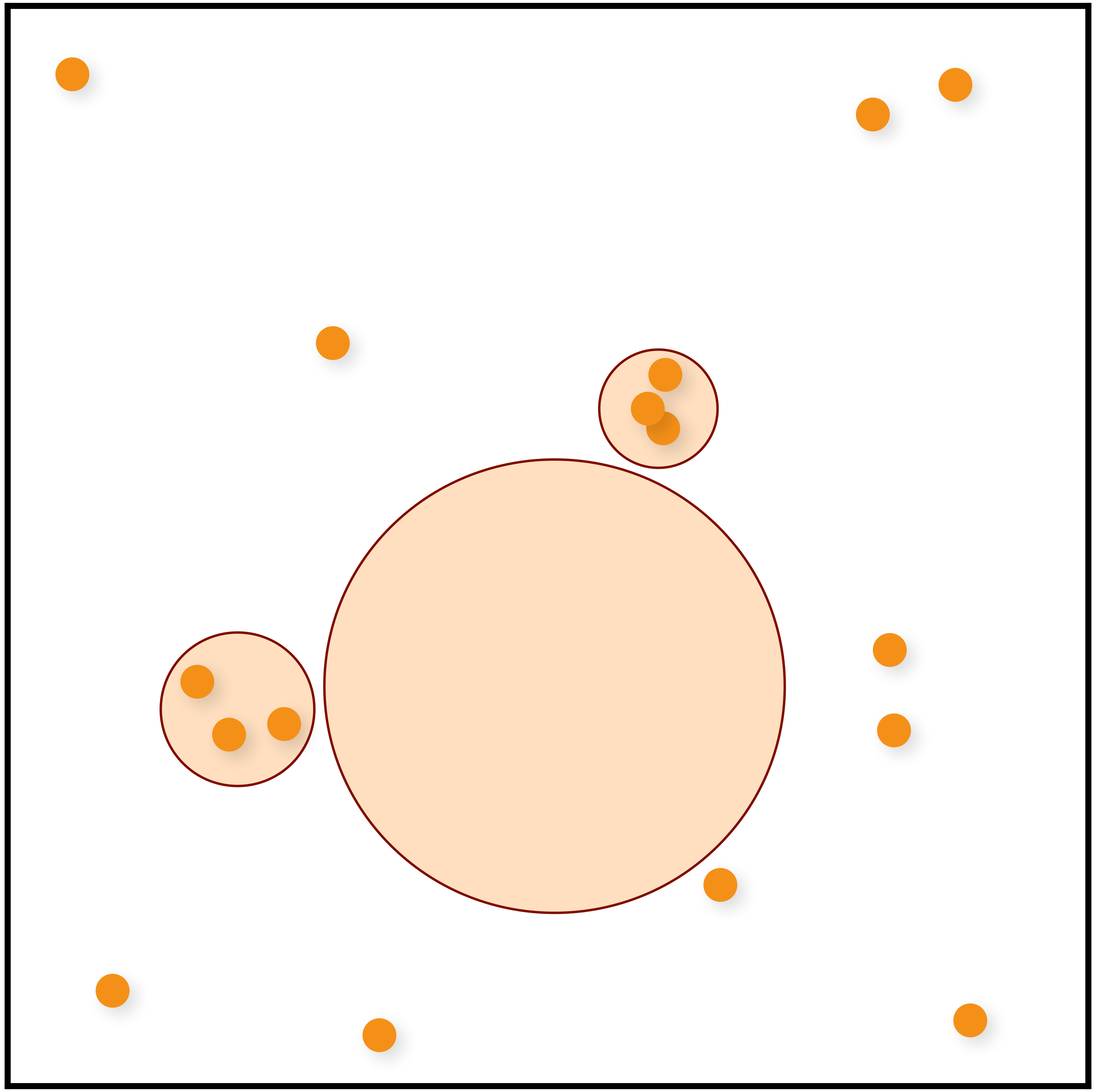
$$F_N = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i)$$

Independent random sampling



$$F_N = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i)$$

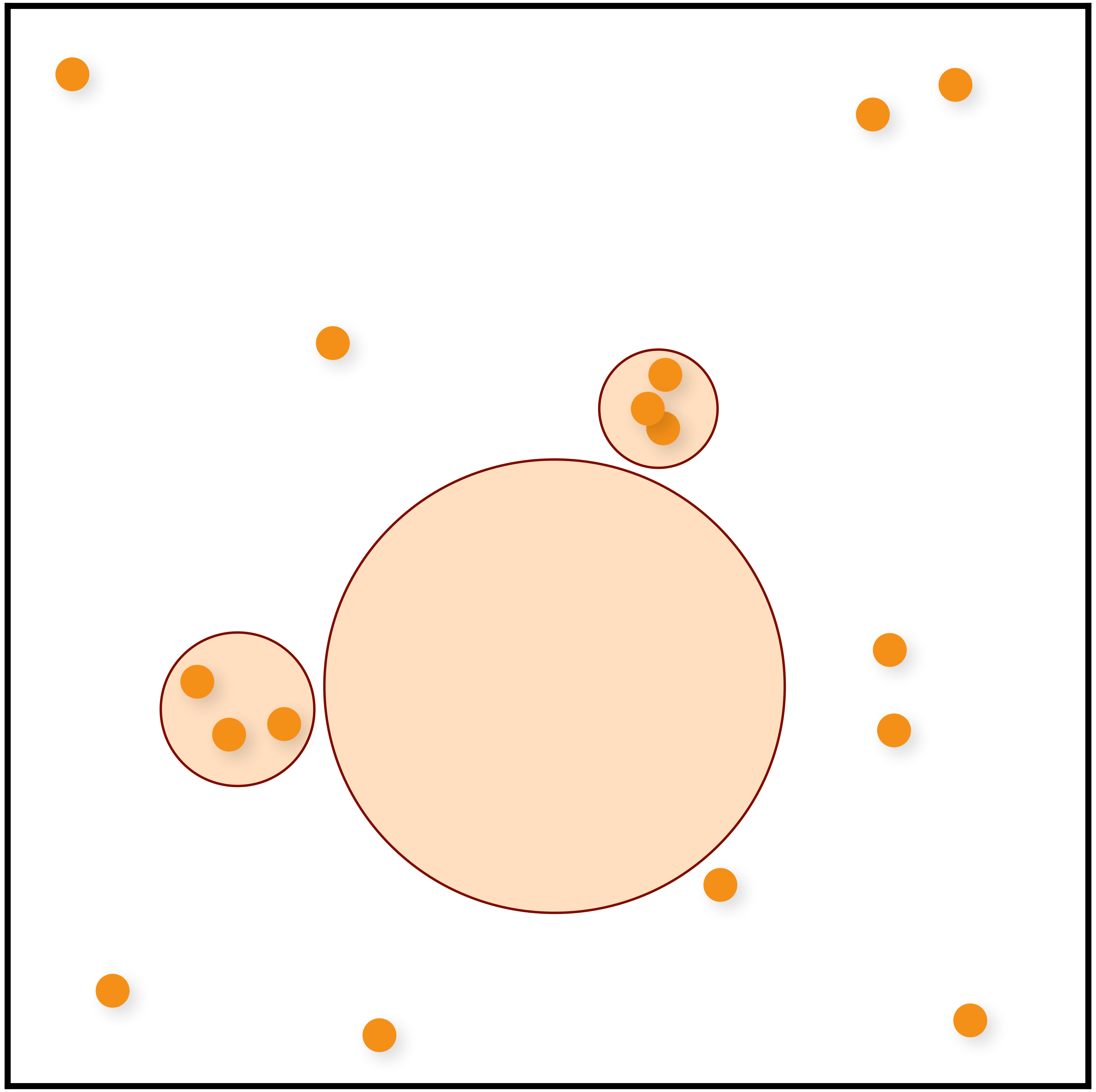
Independent random sampling



$$F_N = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i)$$

✘ Big gaps & clumps

Independent random sampling



$$F_N = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i)$$

✘ Big gaps & clumps

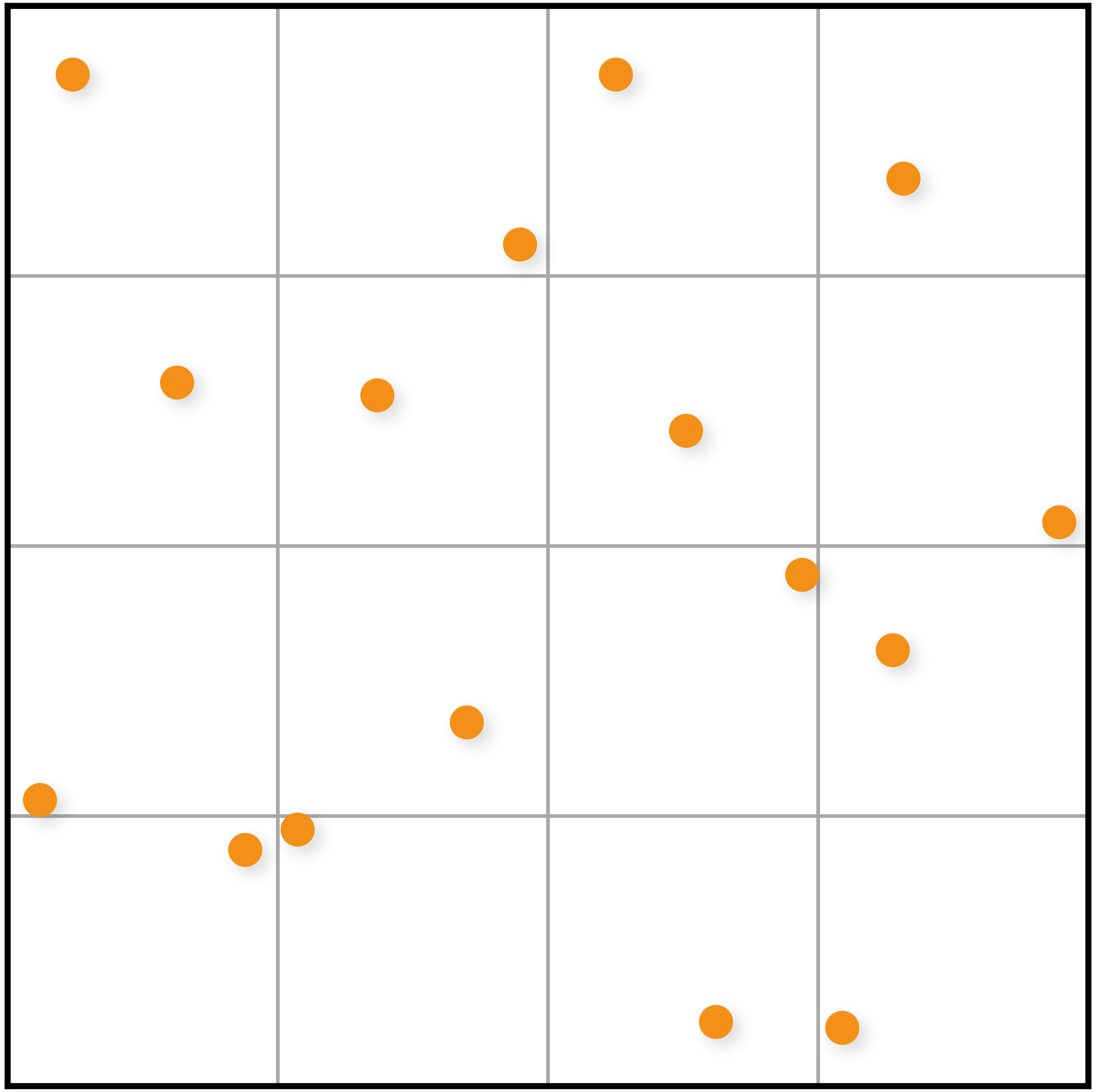
✘ Slow convergence:
Variance = $O(N^{-1})$

Jittered sampling

[Cook 86]

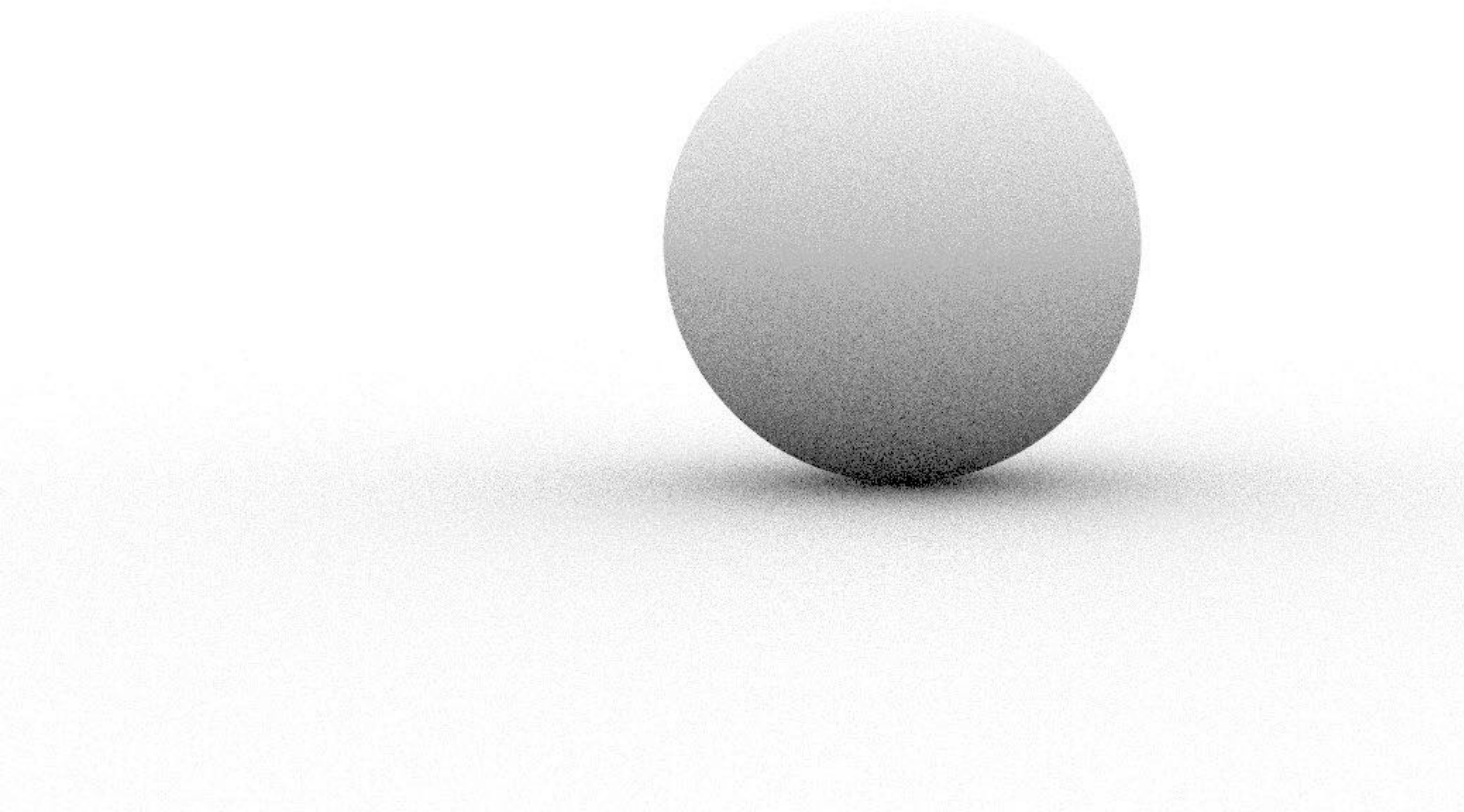
Jittered sampling

[Cook 86]

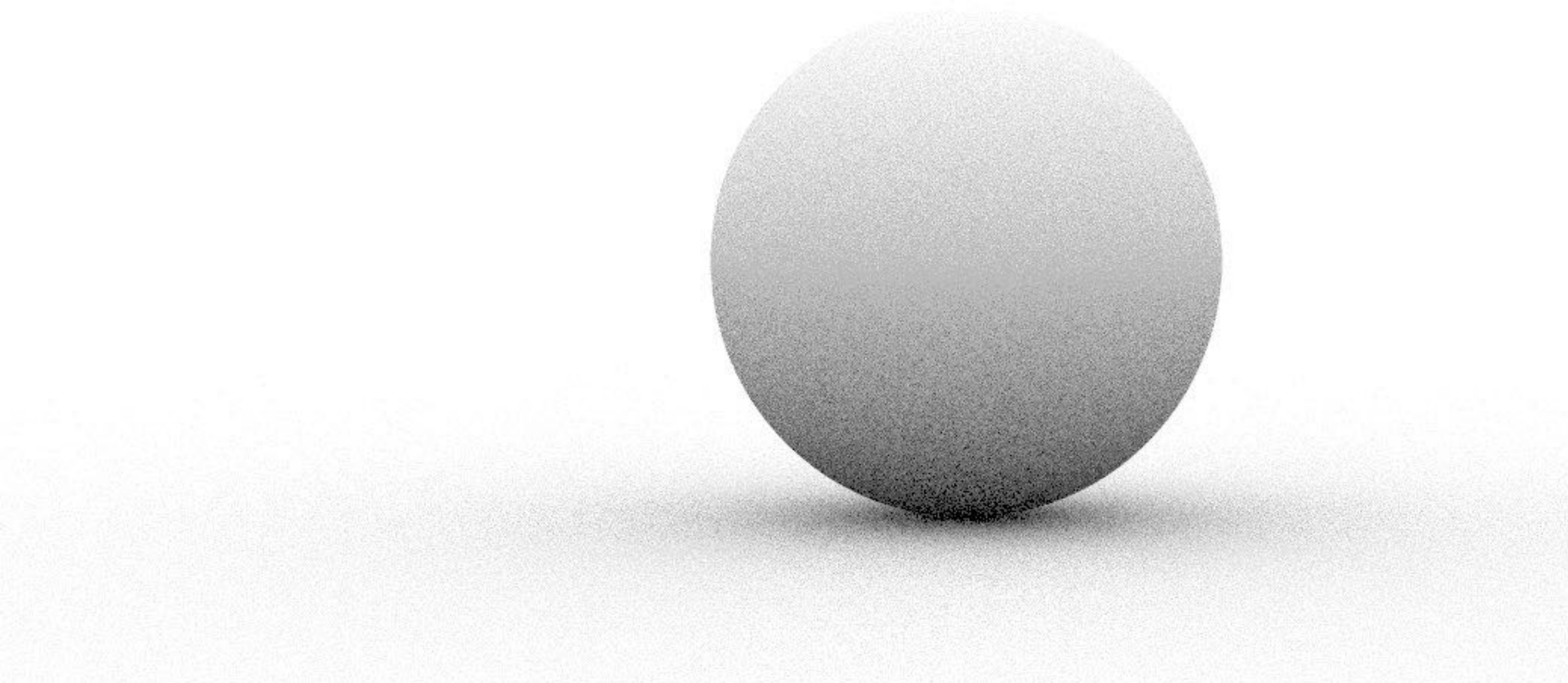


Monte Carlo (16 random samples)

Monte Carlo (16 stratified samples)



Monte Carlo (16 stratified samples)

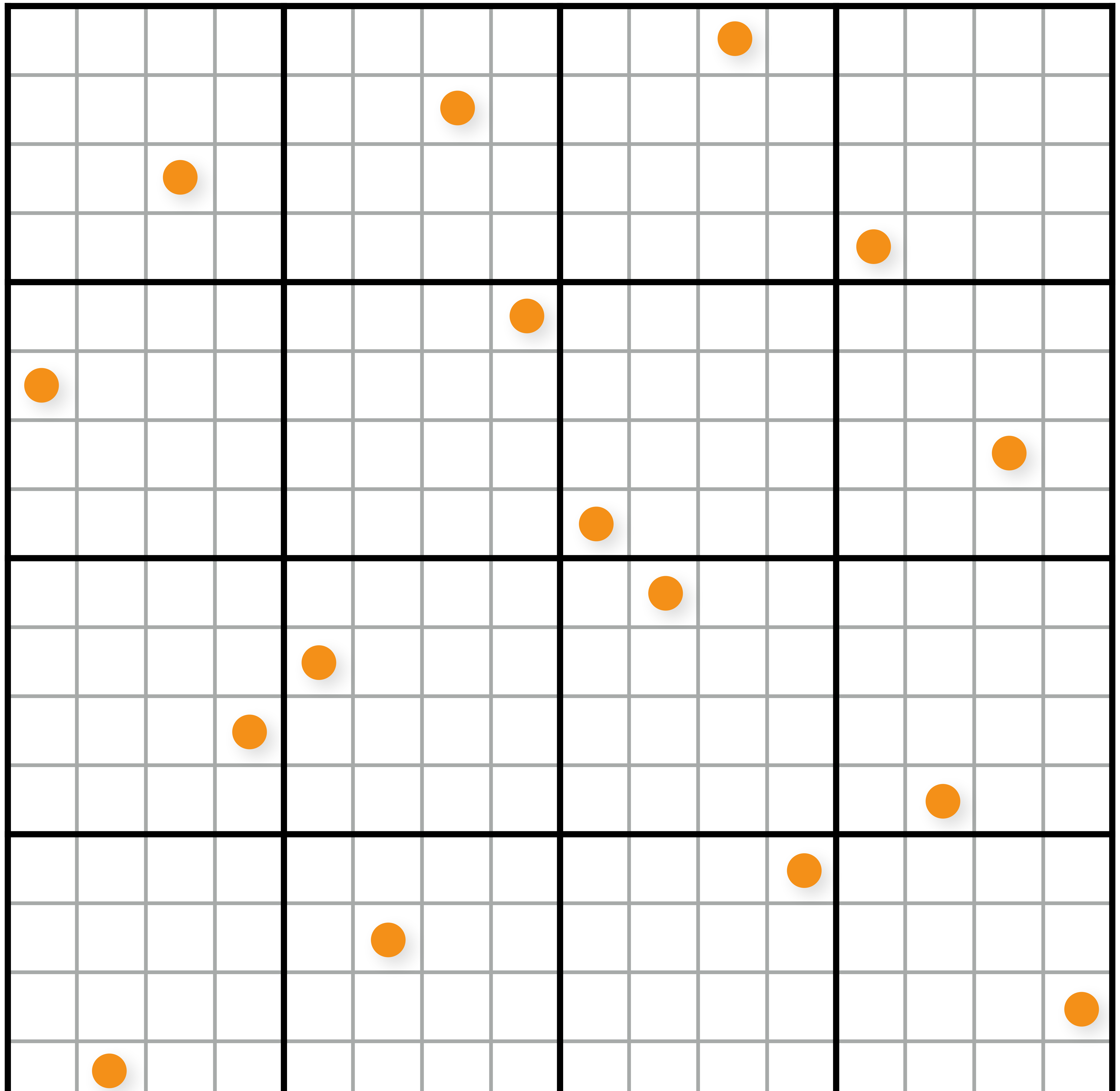


✔ Provably reduces variance

✘ But only practical in low dimensions (1-2D)

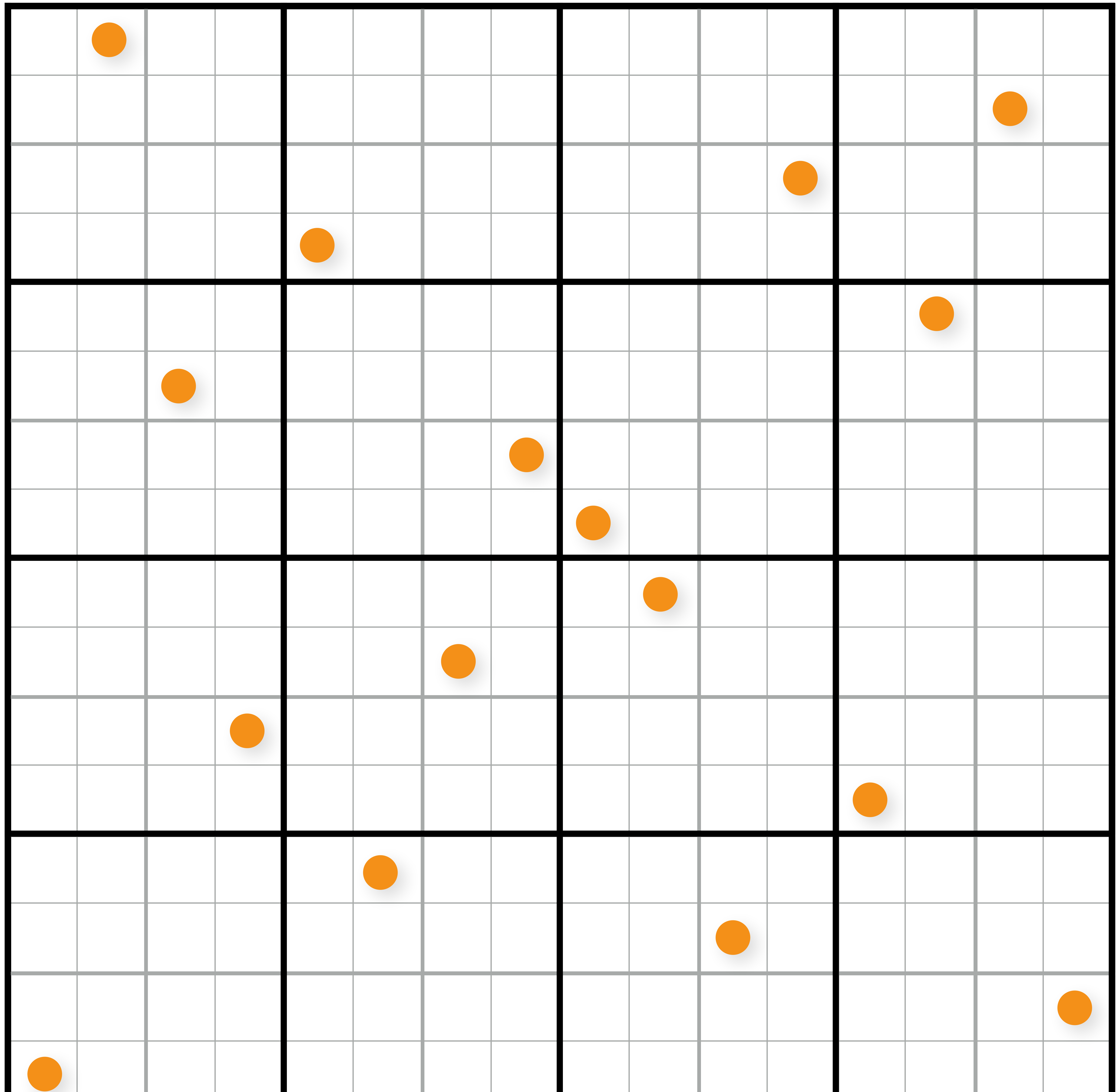
[Chiu et al. 94; Kensler 13]

Multi-Jittered



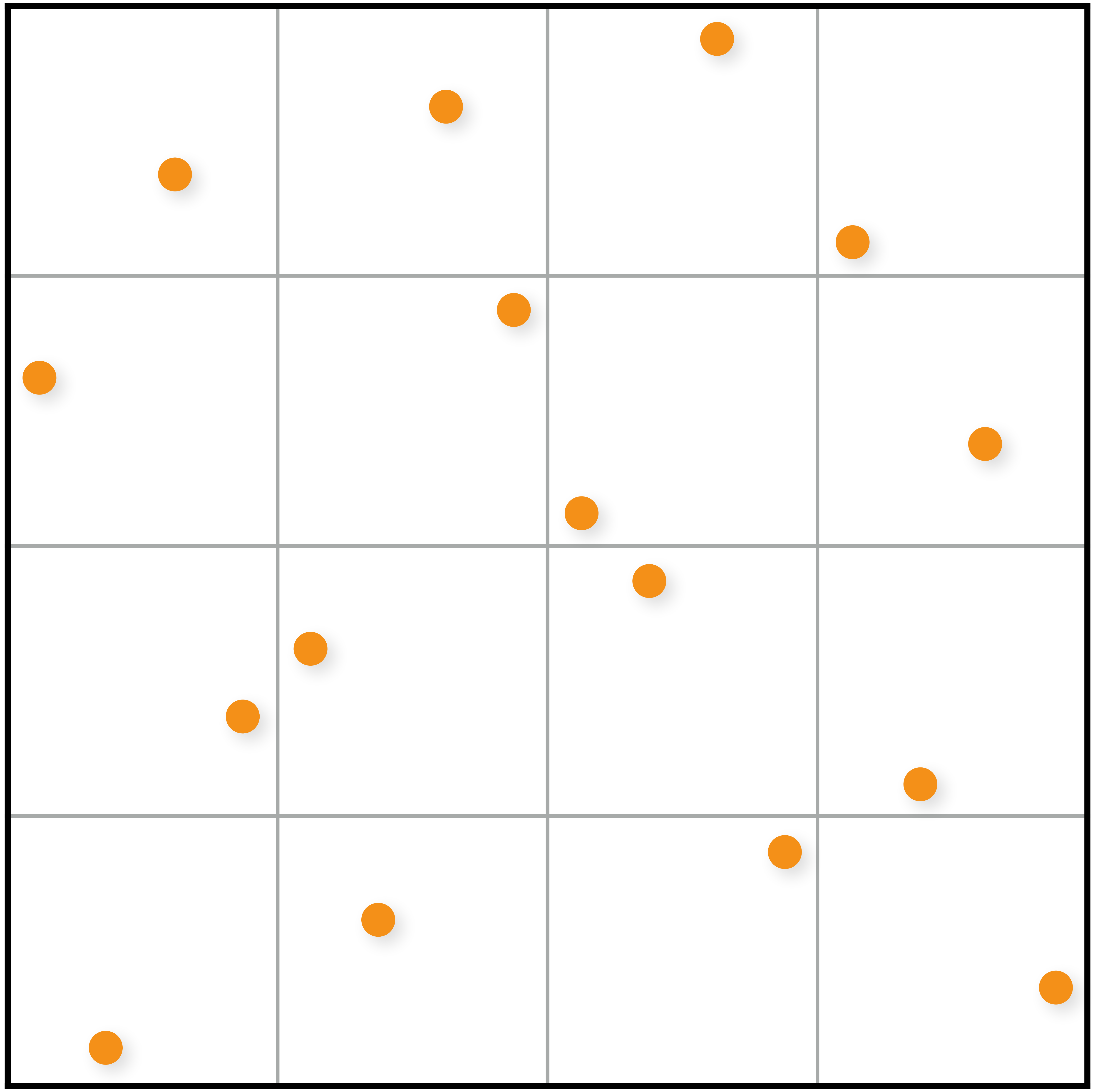
[Sobol 67; Kollig & Keller 02]

(0,2) sequence



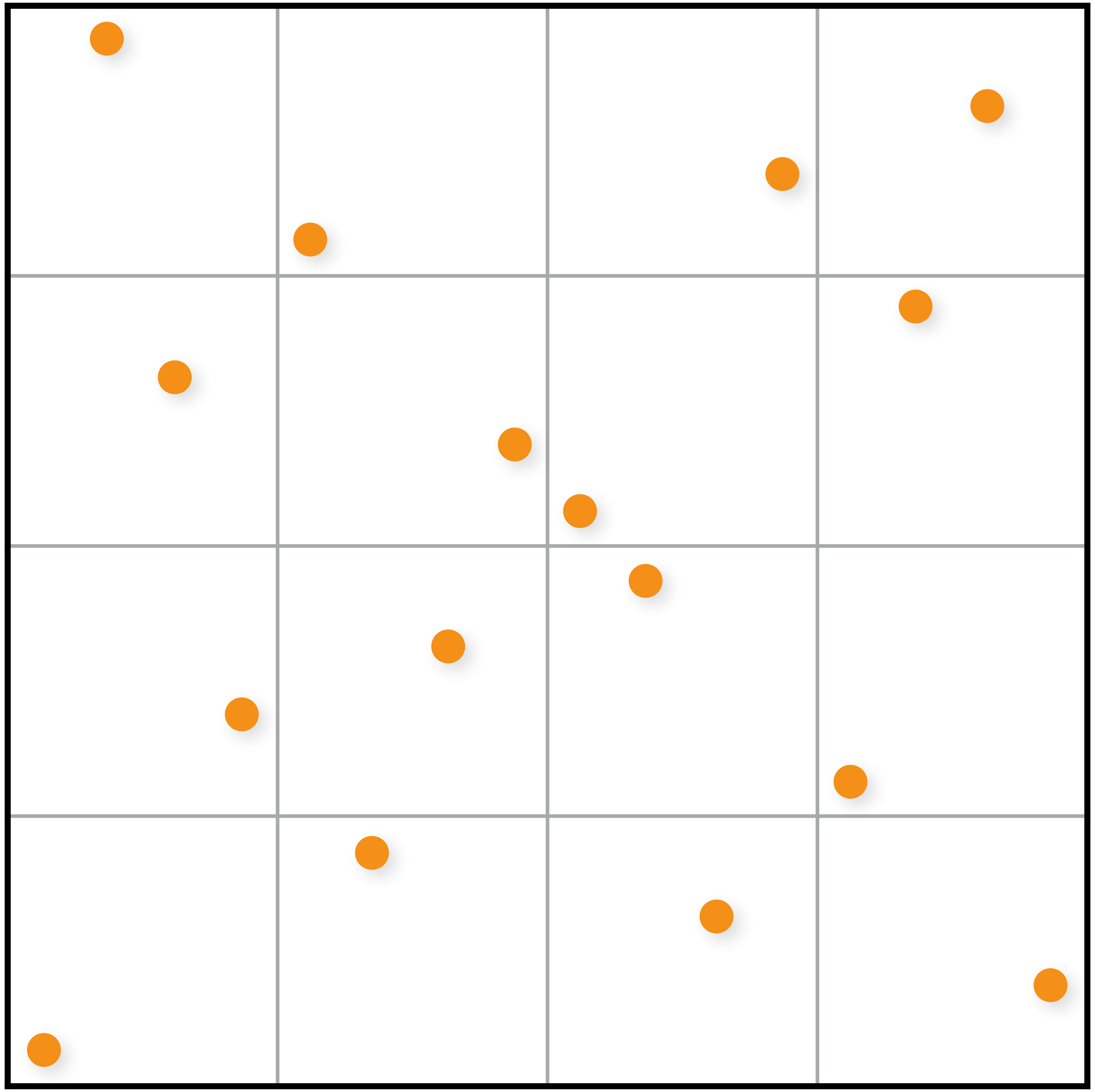
[Chiu et al. 94; Kensler 13]

Multi-Jittered



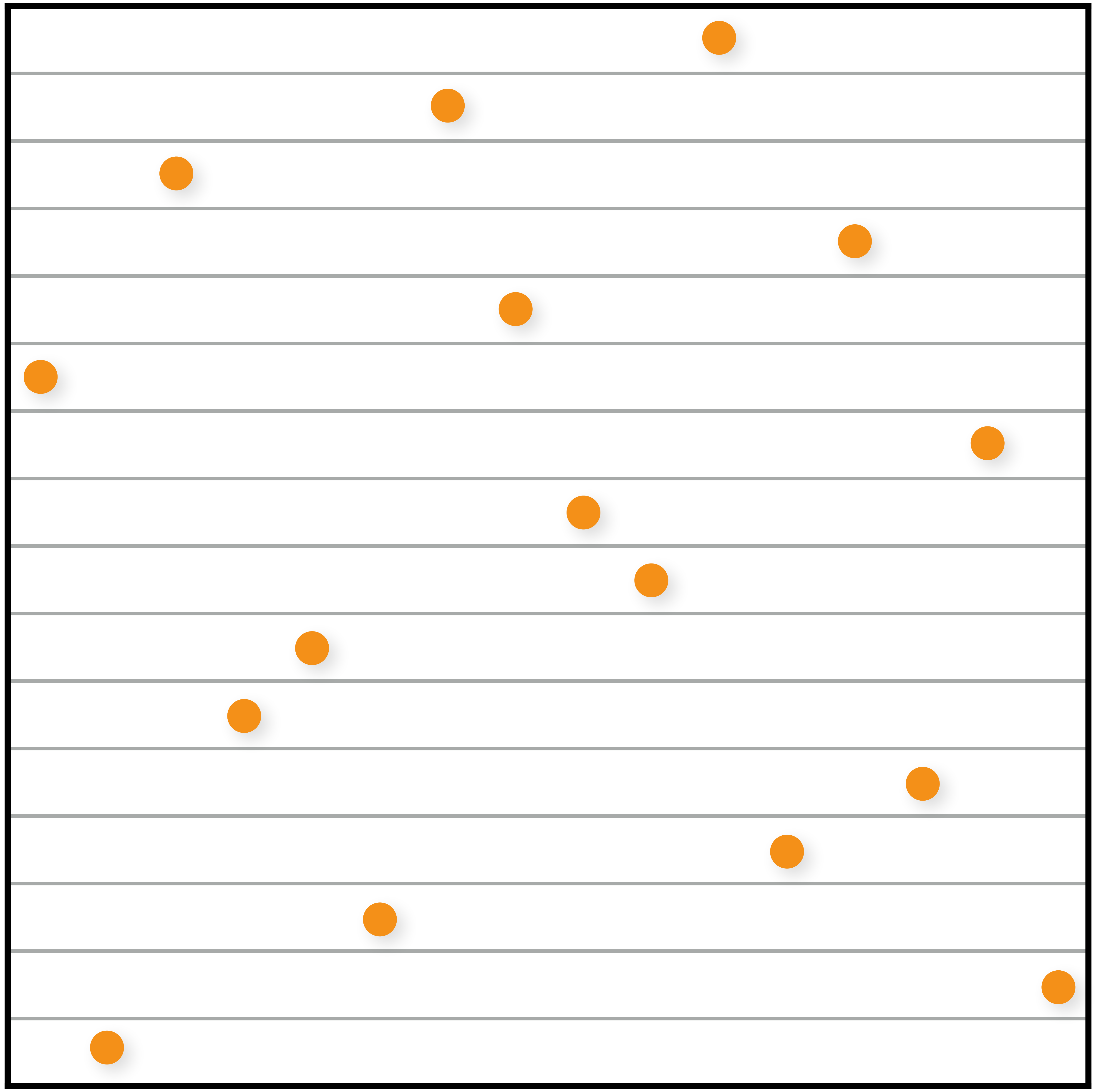
[Sobol 67; Kollig & Keller 02]

(0,2) sequence



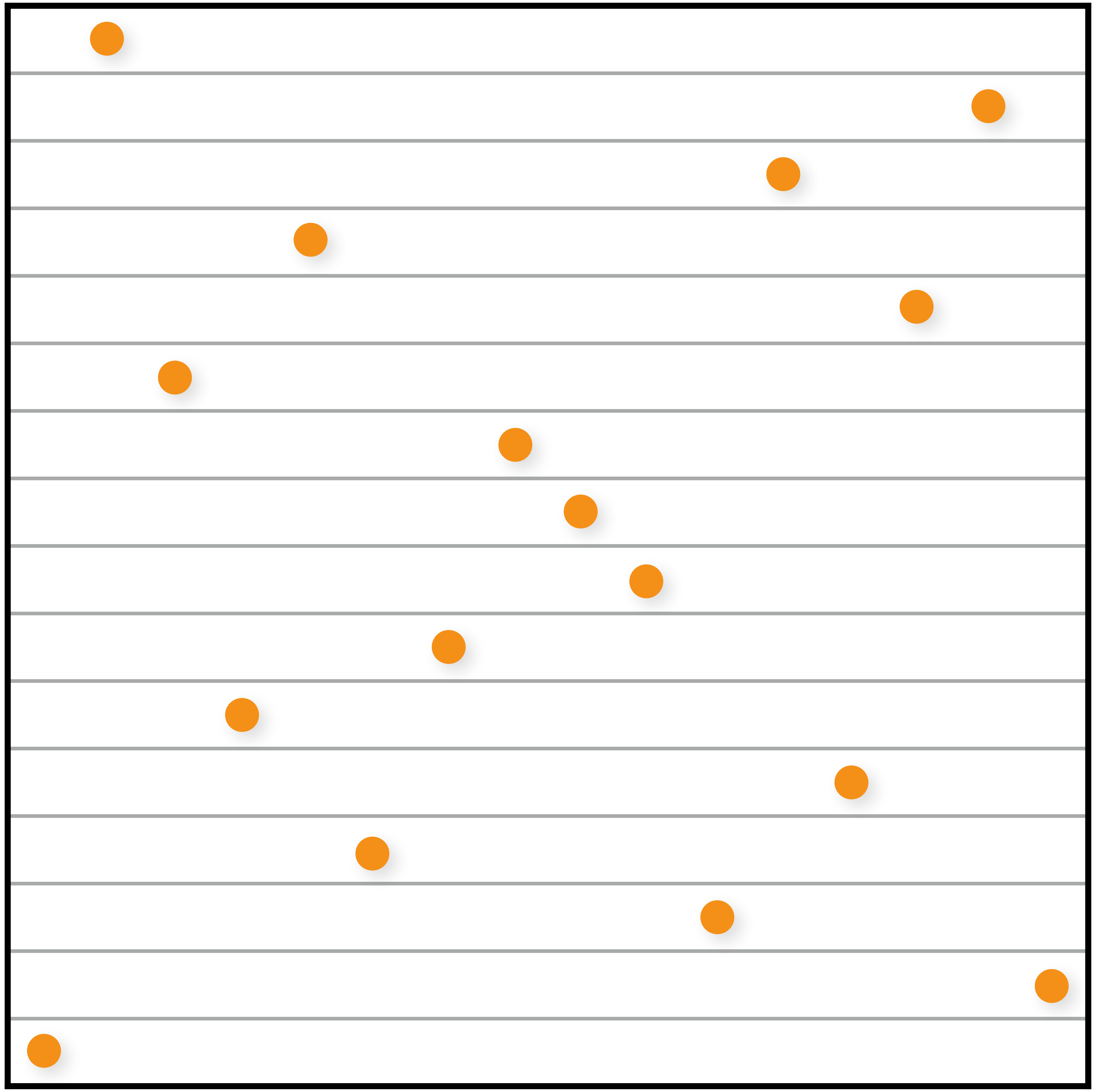
[Chiu et al. 94; Kensler 13]

Multi-Jittered



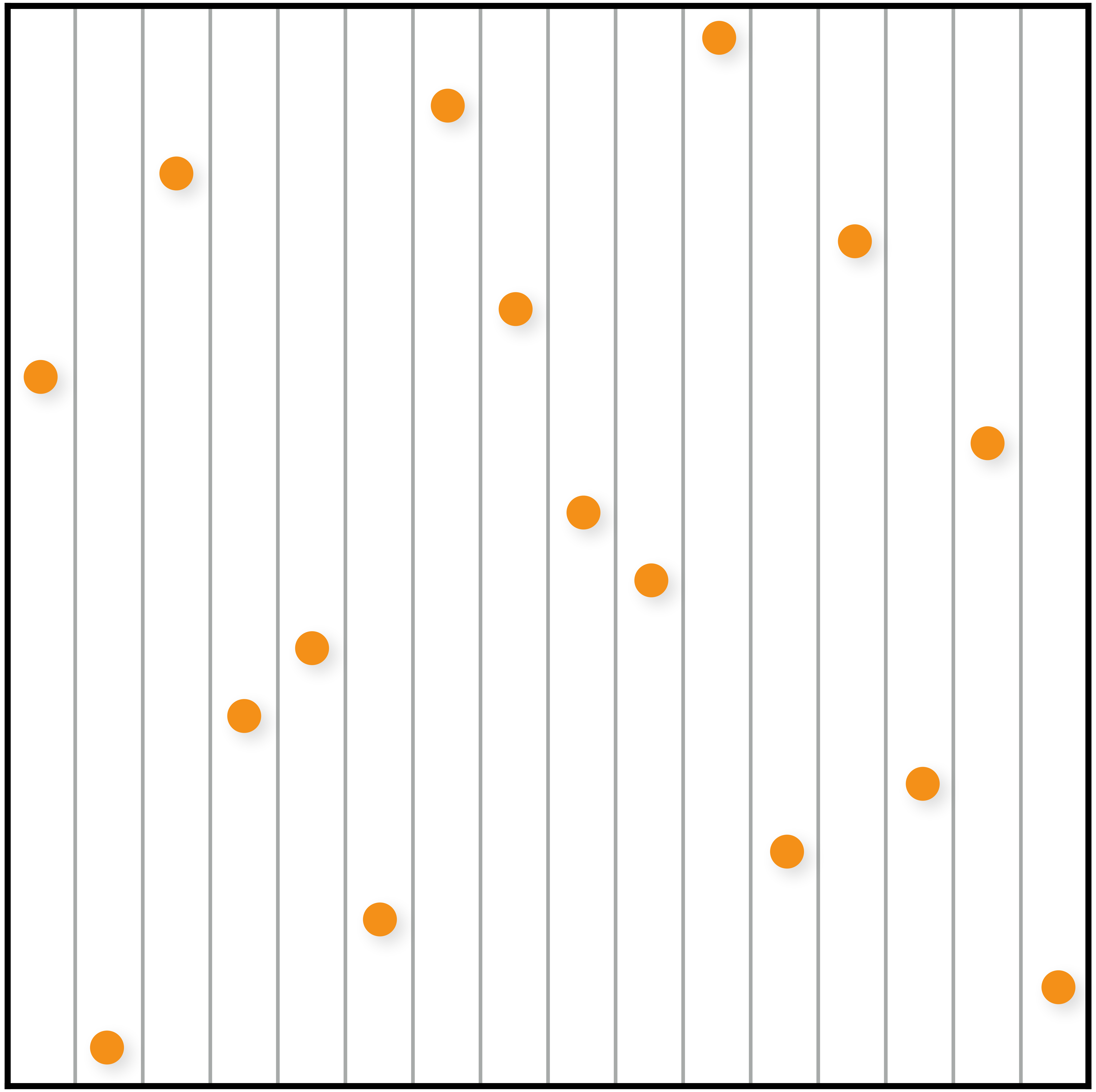
[Sobol 67; Kollig & Keller 02]

(0,2) sequence



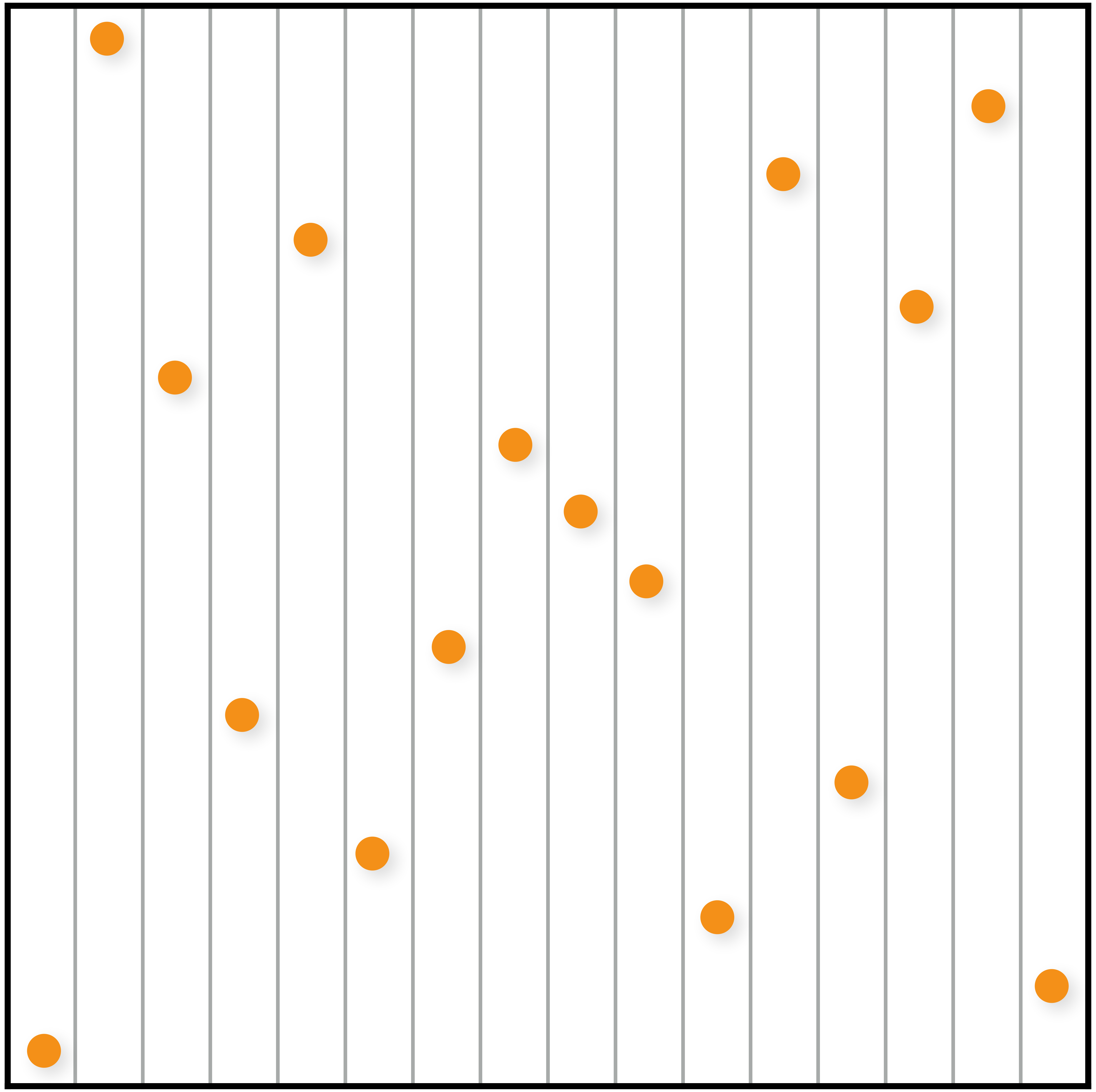
[Chiu et al. 94; Kensler 13]

Multi-Jittered



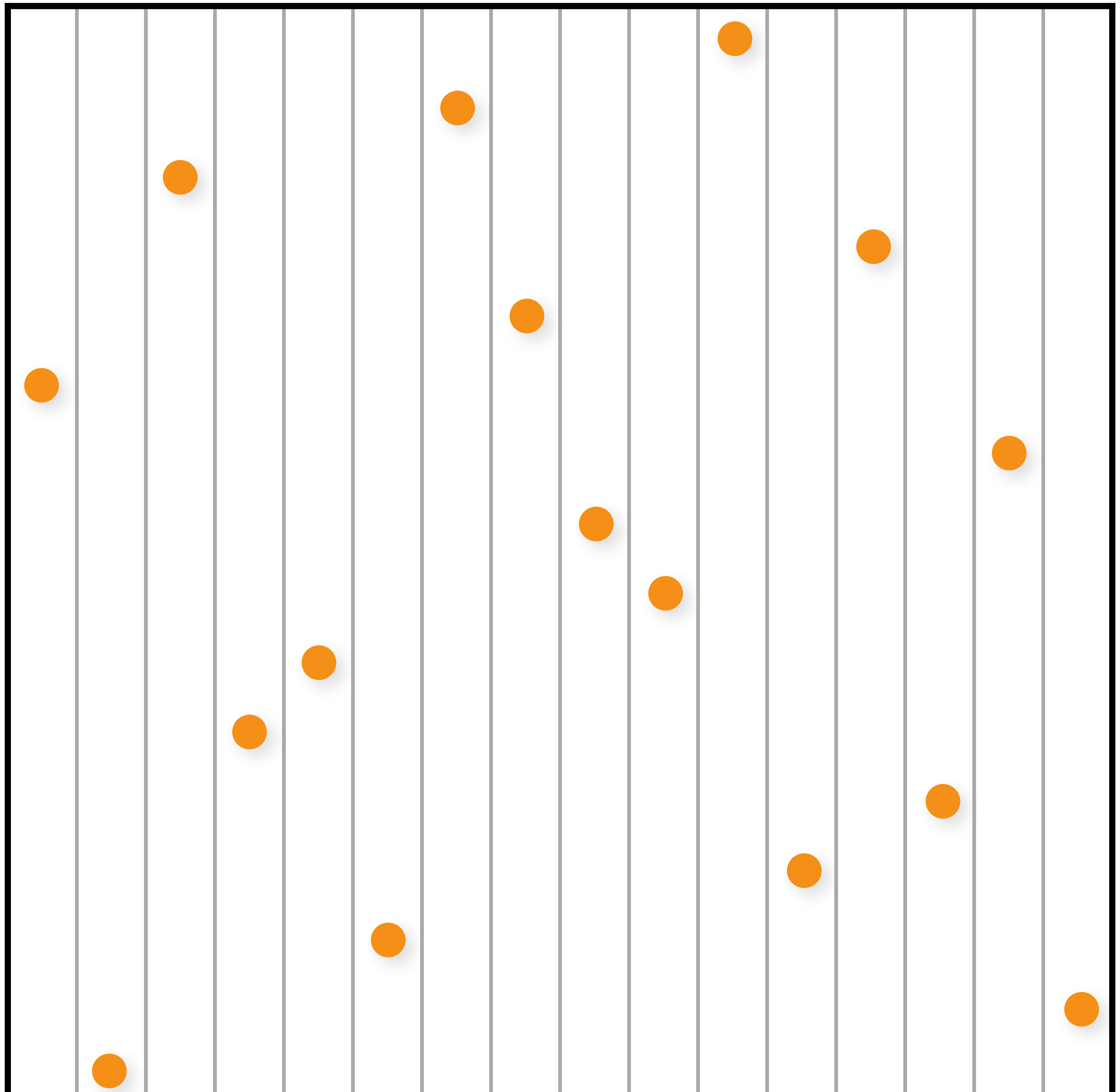
[Sobol 67; Kollig & Keller 02]

(0,2) sequence



[Chiu et al. 94; Kensler 13]

Multi-Jittered

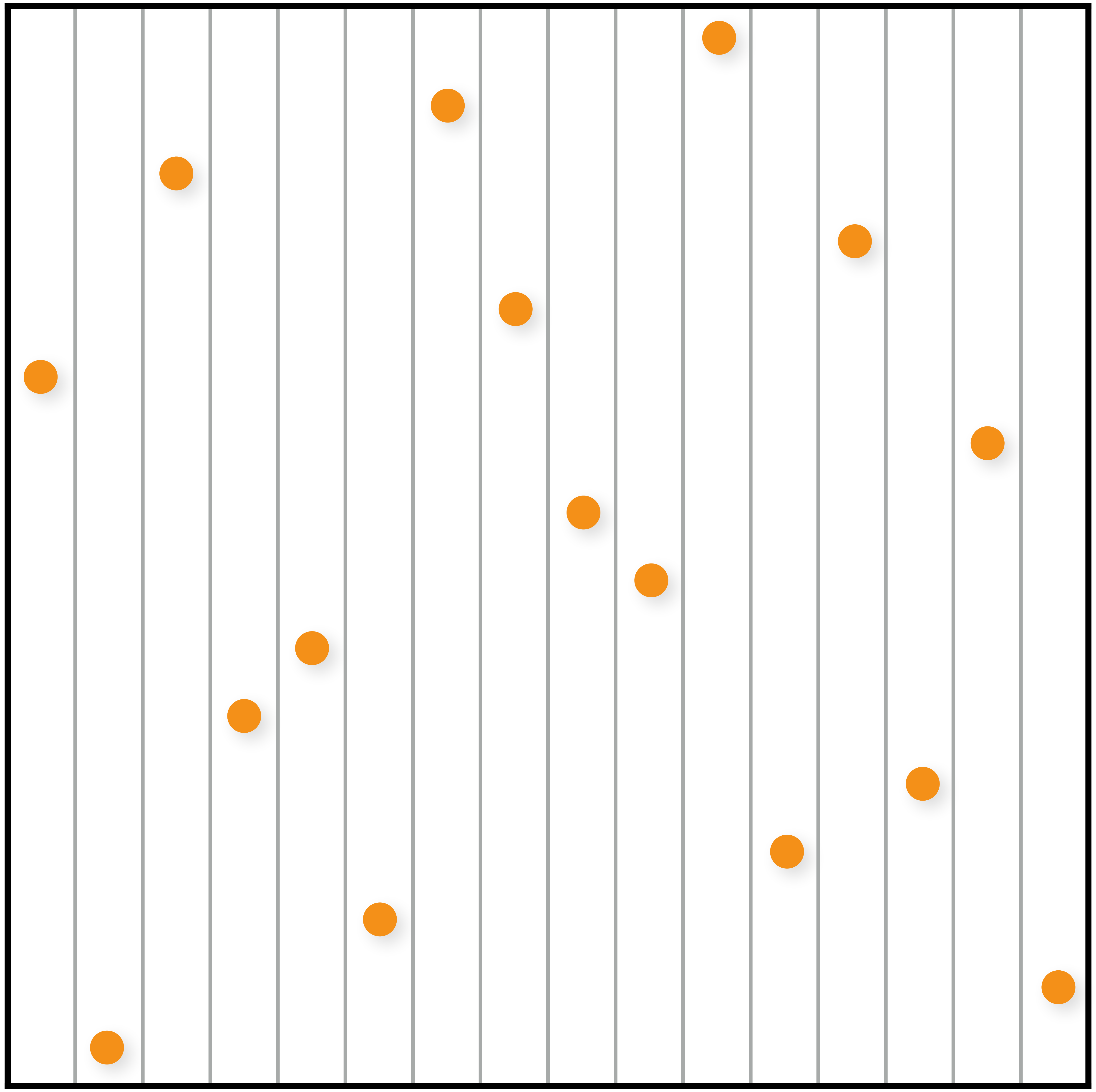


[Sobol 67; Kollig & Keller 02]

(0,2) sequence

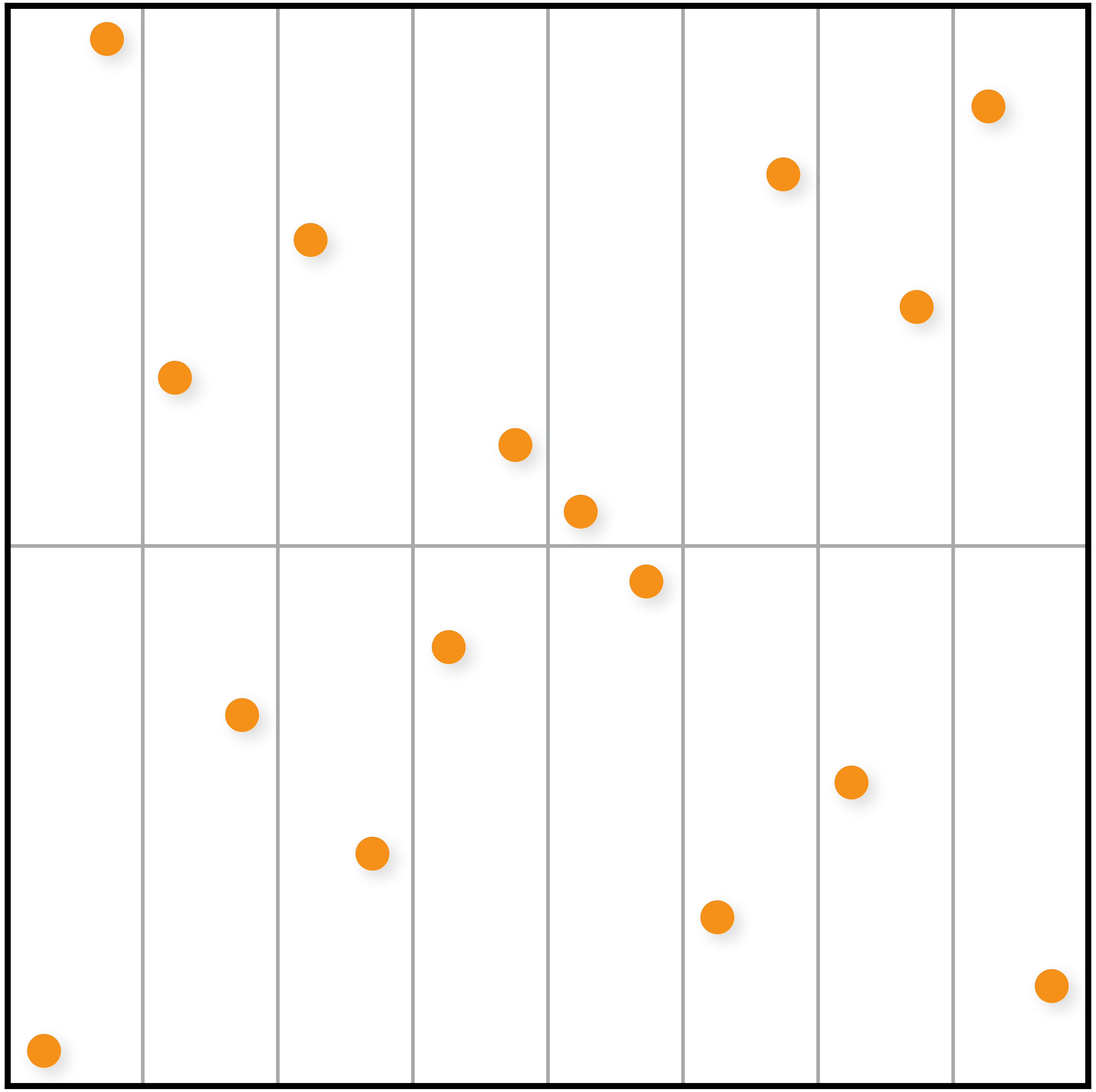
[Chiu et al. 94; Kensler 13]

Multi-Jittered



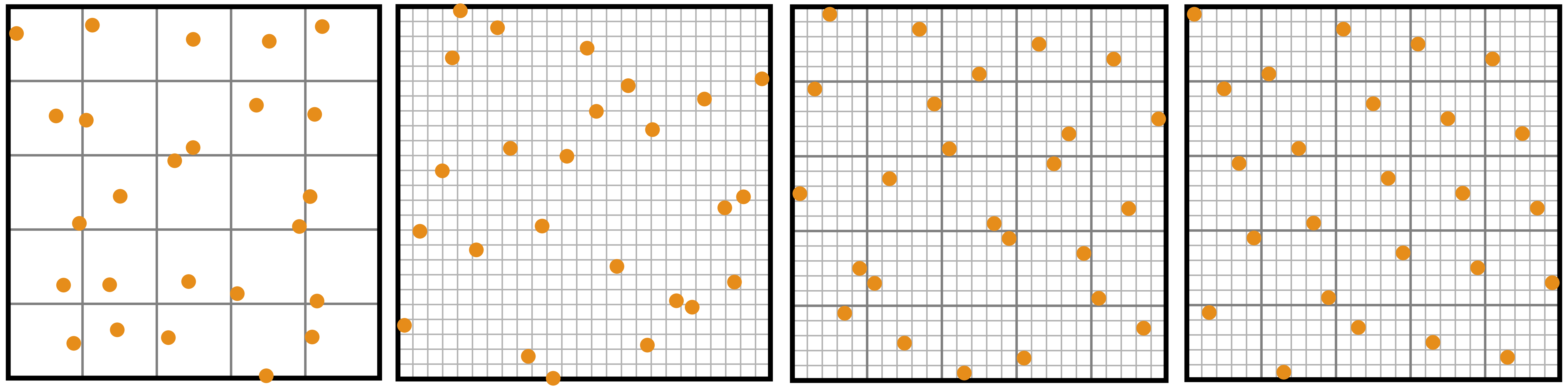
[Sobol 67; Kollig & Keller 02]

(0,2) sequence

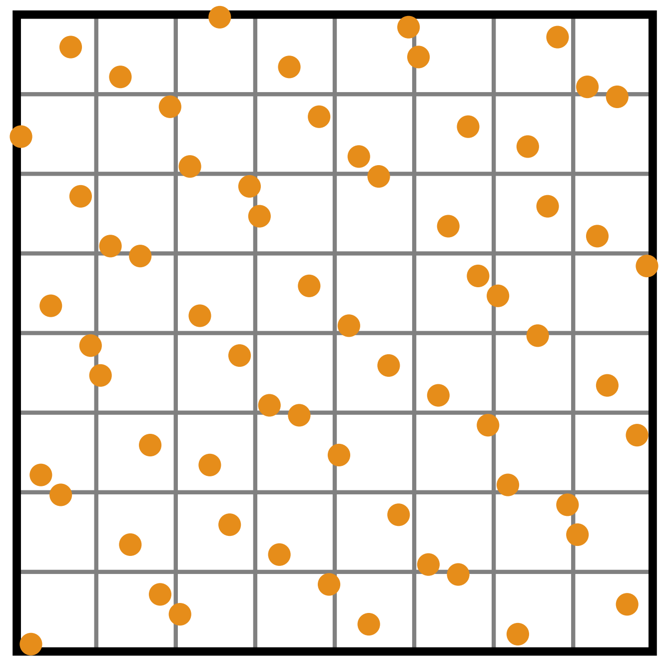


Correlated sampling zoo

Correlated sampling zoo

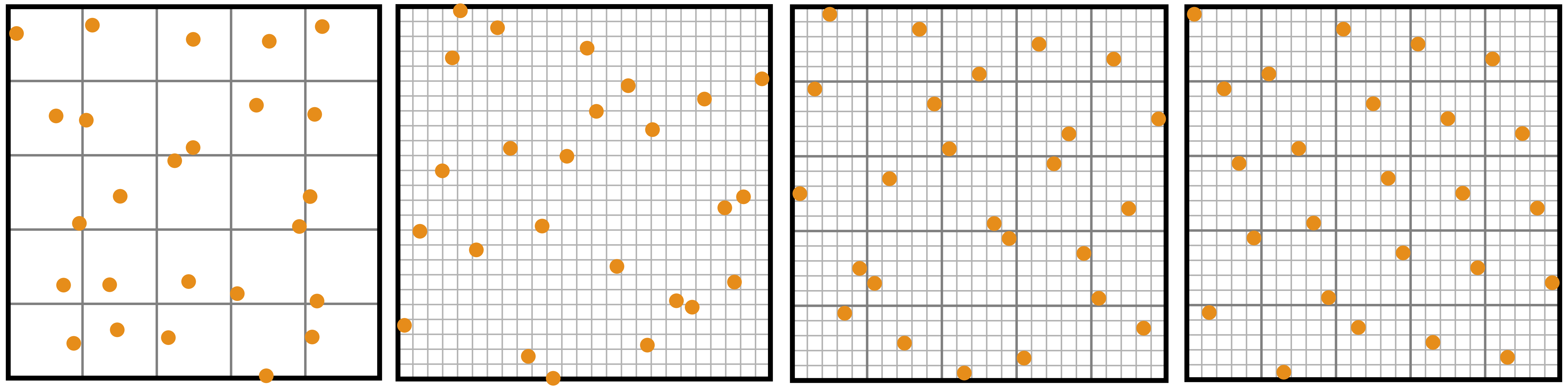


Stratification-based

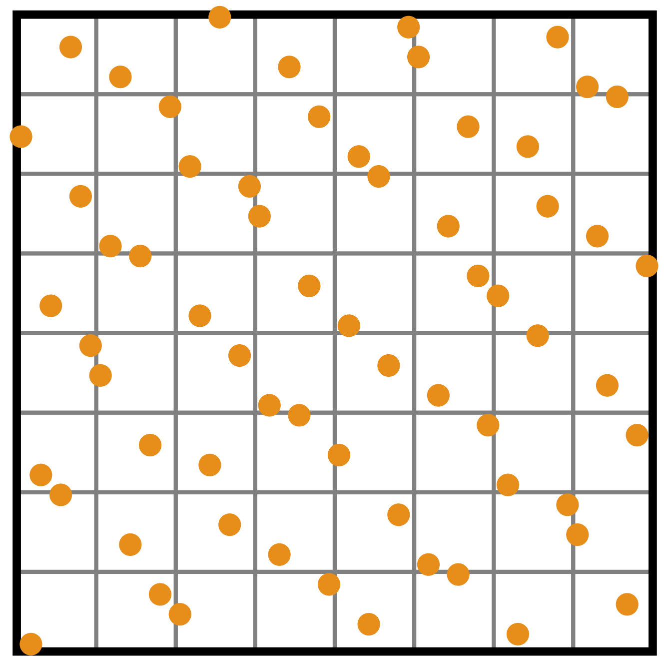


Quasi-MC/
low-discrepancy

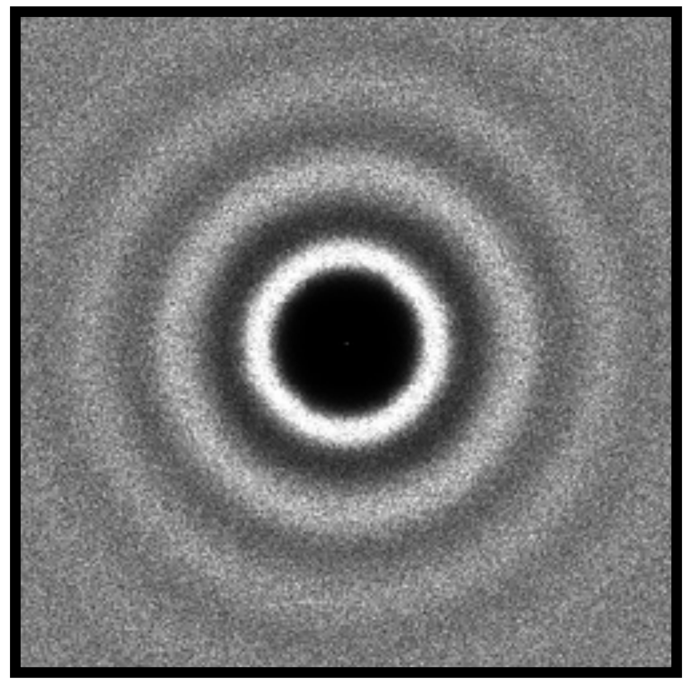
Correlated sampling zoo



Stratification-based

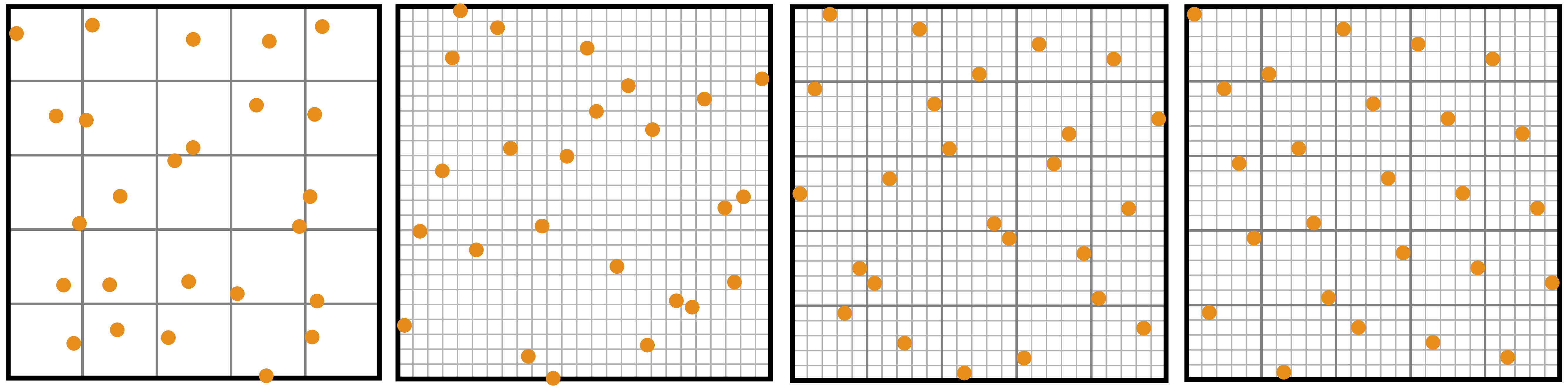


Quasi-MC/
low-discrepancy

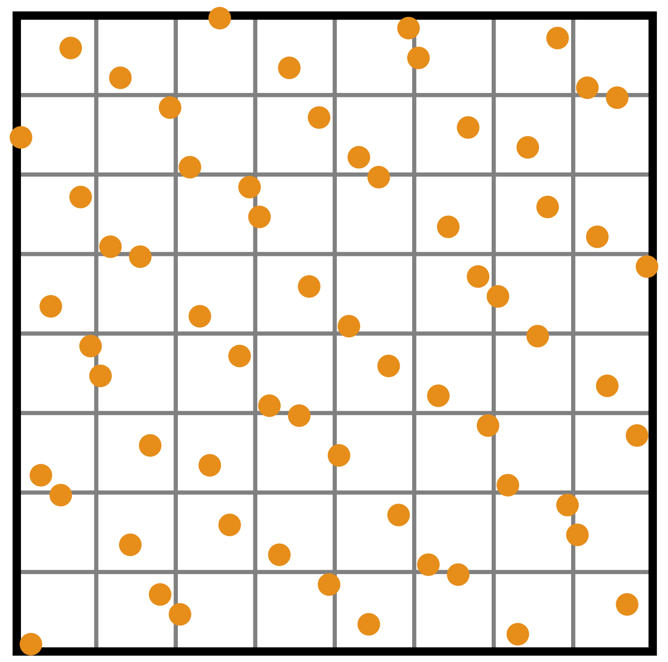


Frequency-based/
“blue-noise”

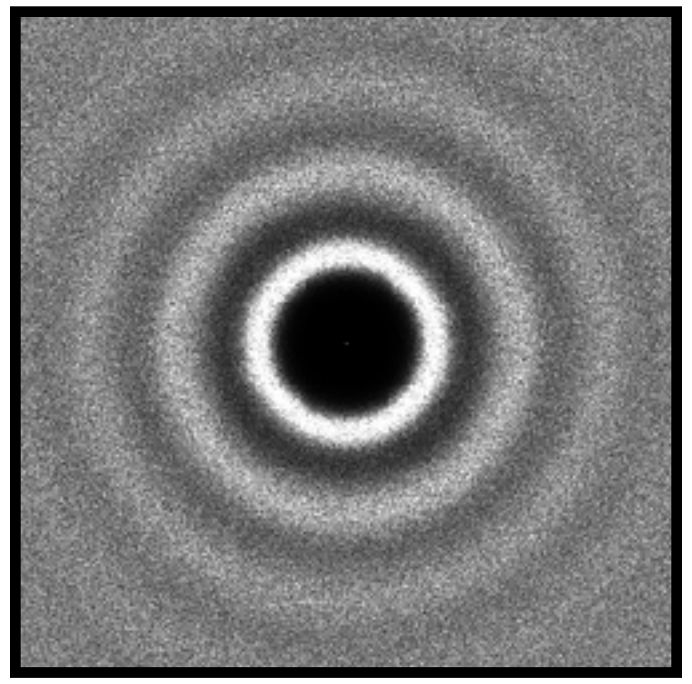
Correlated sampling zoo



Stratification-based



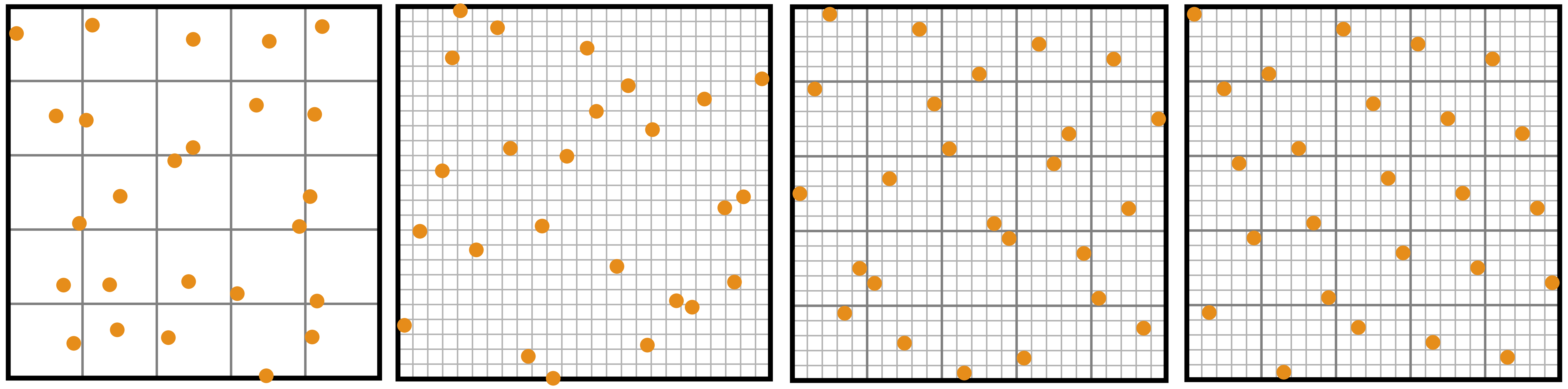
Quasi-MC/
low-discrepancy



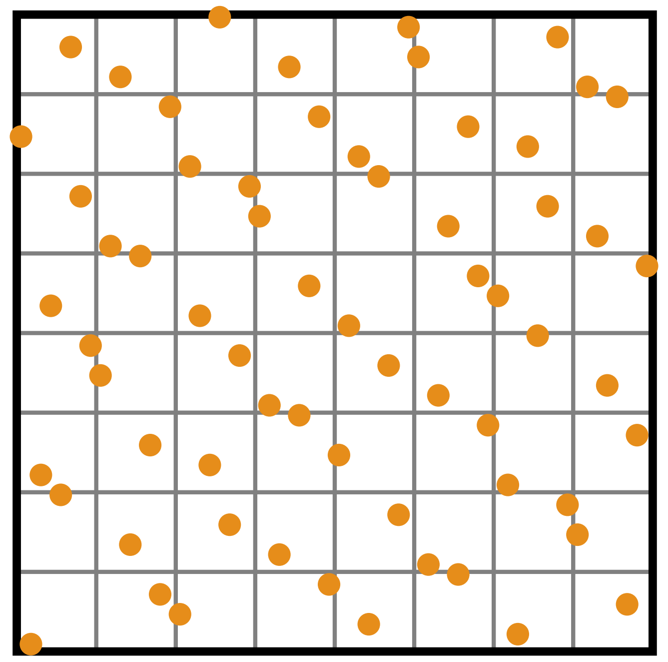
Frequency-based/
"blue-noise"

See recent STAR [SÖA*19]

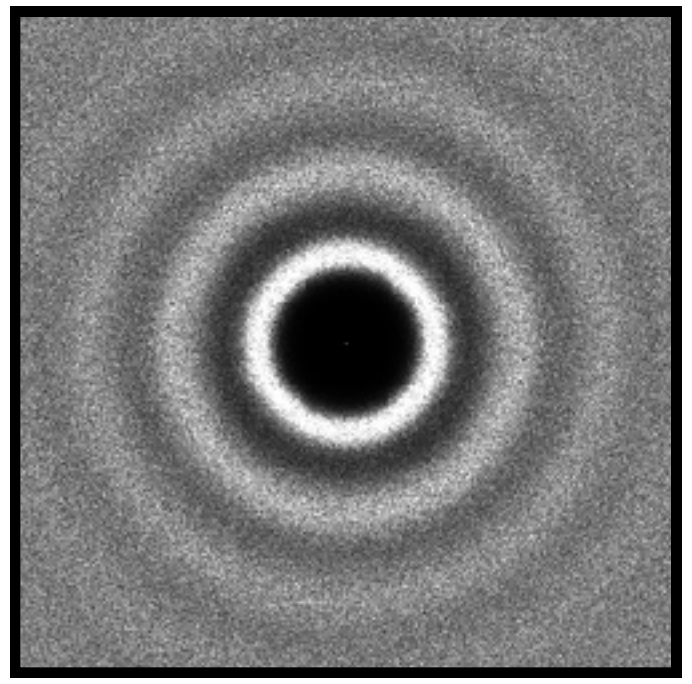
Correlated sampling zoo



Stratification-based



Quasi-MC/ low-discrepancy

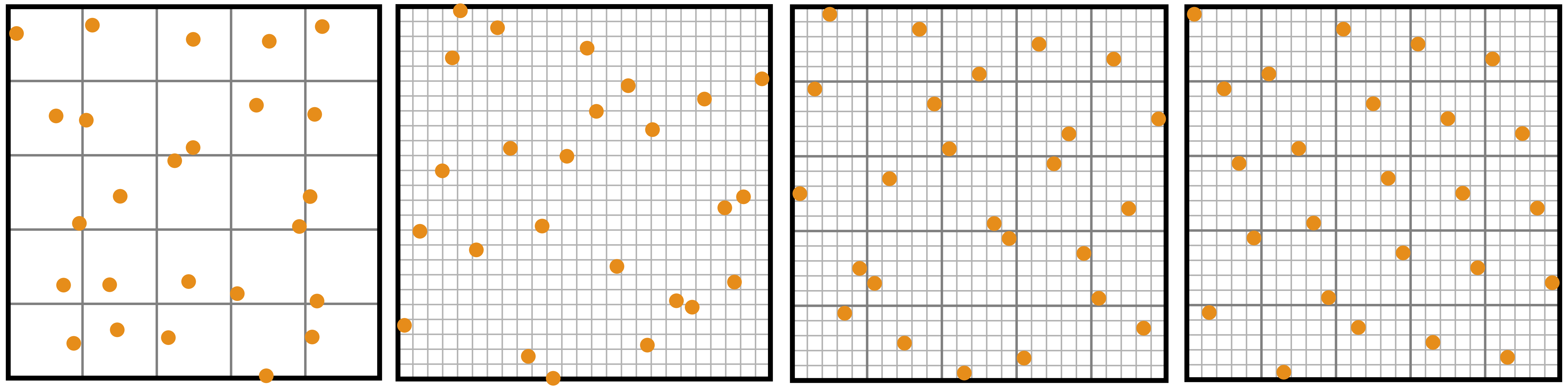


Frequency-based/ “blue-noise”

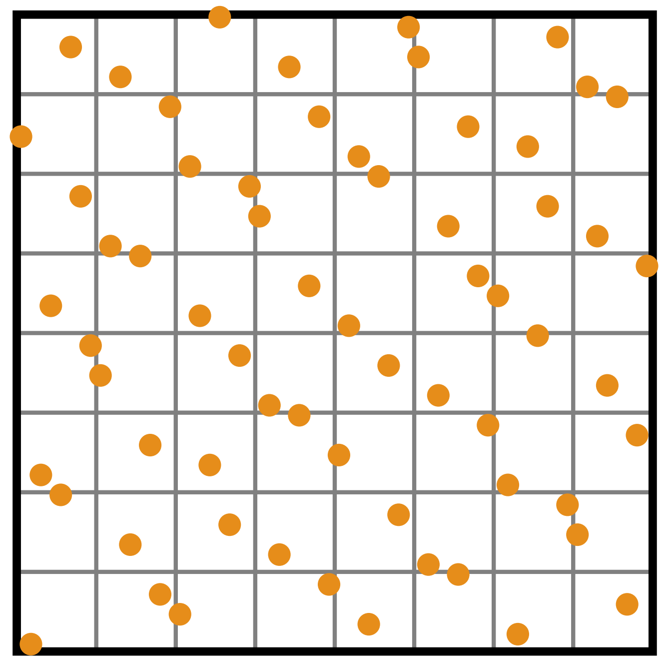
See recent STAR [SÖA*19]

✘ **Don’t generalize efficiently beyond 2D**

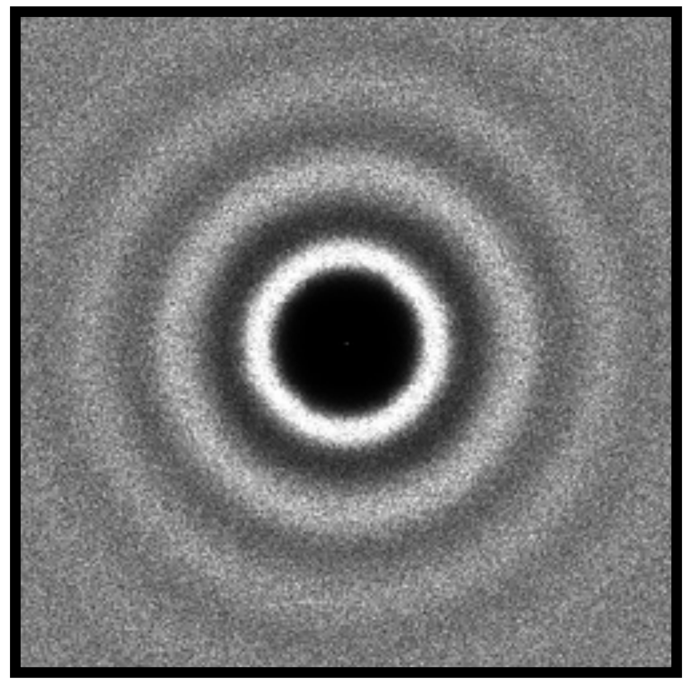
Correlated sampling zoo



Stratification-based



Quasi-MC/
low-discrepancy



Frequency-based/
“blue-noise”

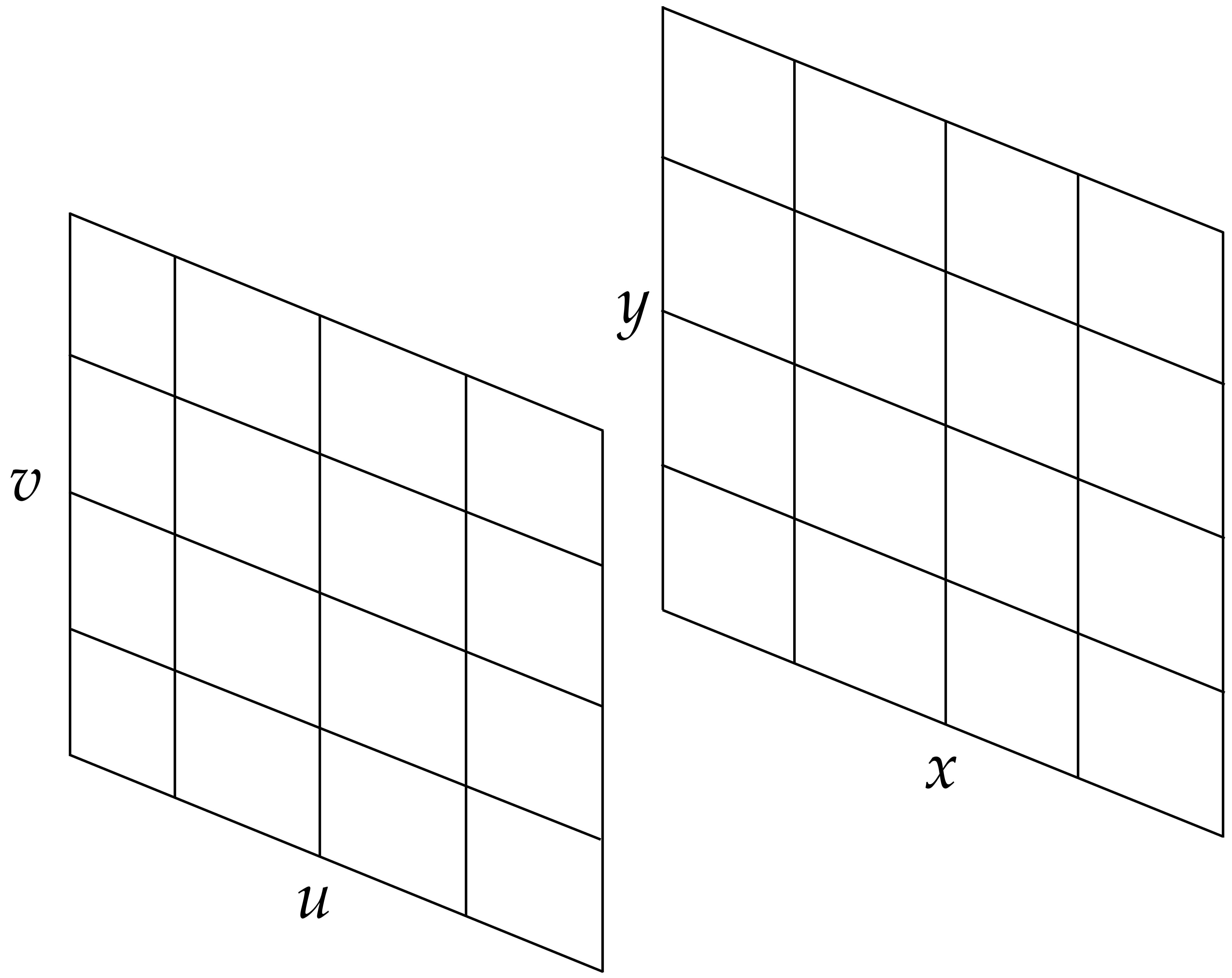
See recent STAR [SÖA*19]

✗ Don’t generalize efficiently beyond 2D

High dimensional samples?

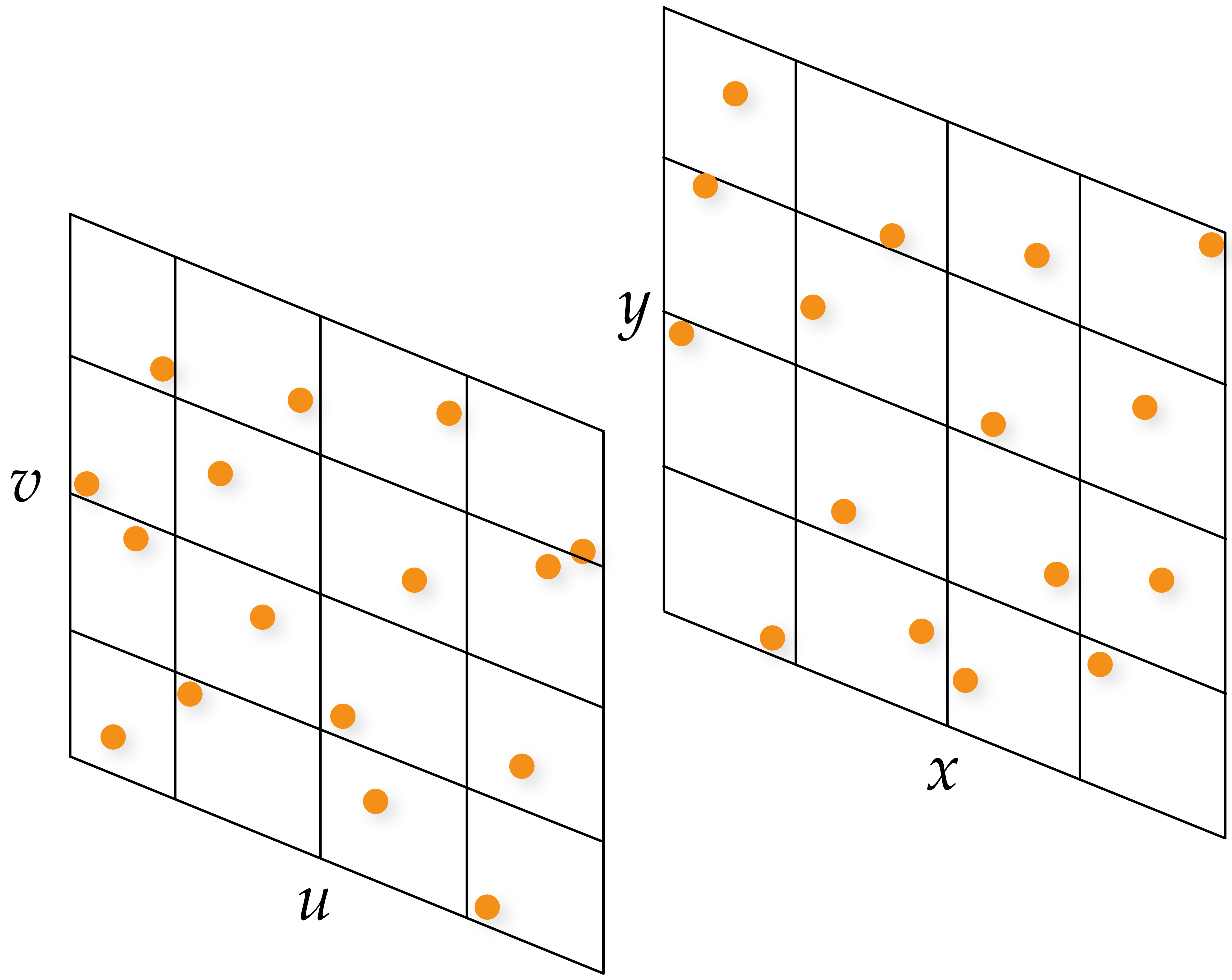
[Cook 86]

"Padding" 2D point sets



"Padding" 2D point sets

[Cook 86]

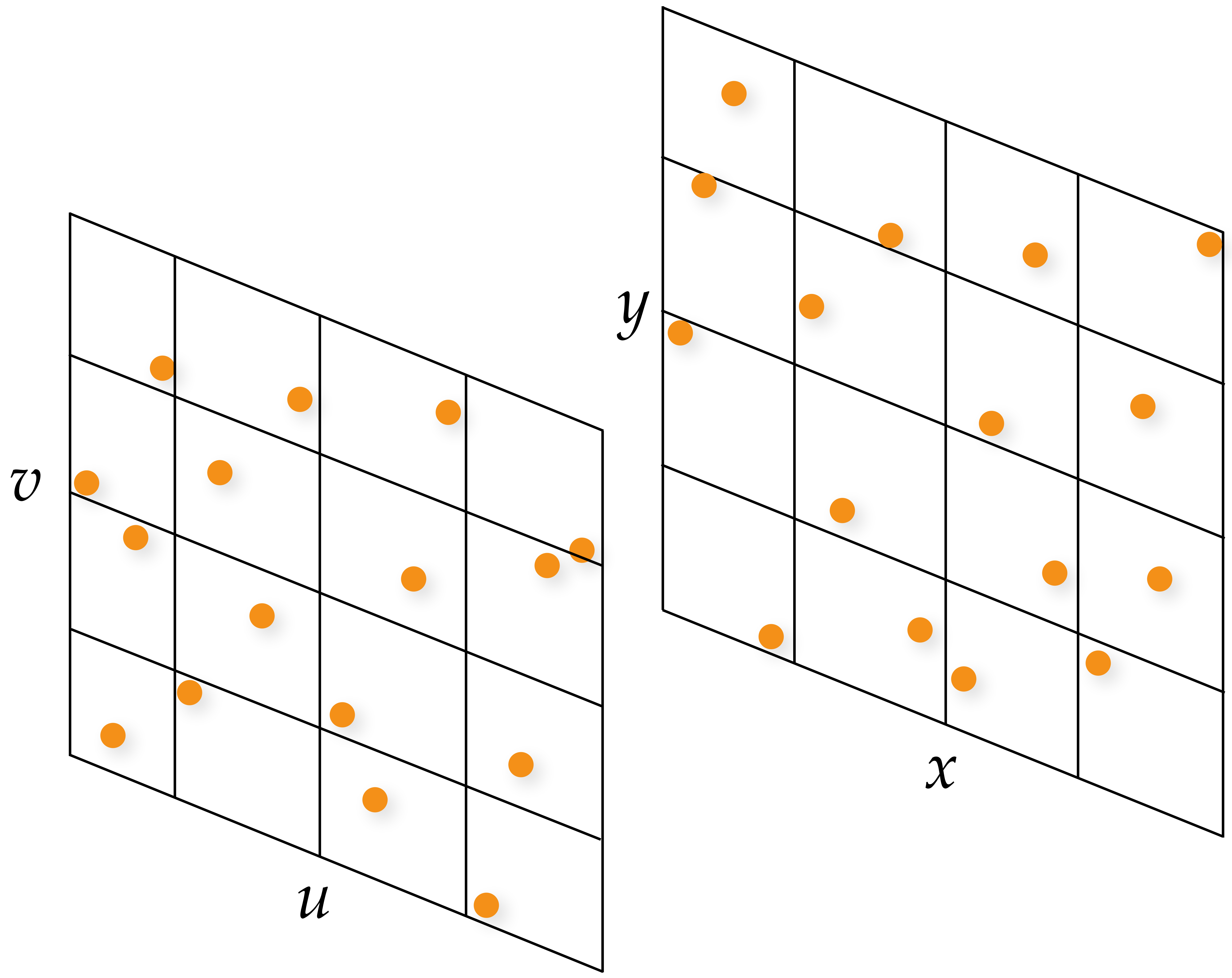


2D	2D
(x_1, y_1)	(u_1, v_1)
(x_2, y_2)	(u_2, v_2)
(x_3, y_3)	(u_3, v_3)
(x_4, y_4)	(u_4, v_4)
$\vdots$	$\vdots$

Slide after Gurprit Singh

"Padding" 2D point sets

[Cook 86]



2D → 2D

$$(x_1, y_1) \to (u_3, v_3)$$
$$(x_2, y_2) \to (u_1, v_1)$$
$$(x_3, y_3) \to (u_4, v_4)$$
$$(x_4, y_4) \to (u_2, v_2)$$
$$\vdots \qquad \vdots$$

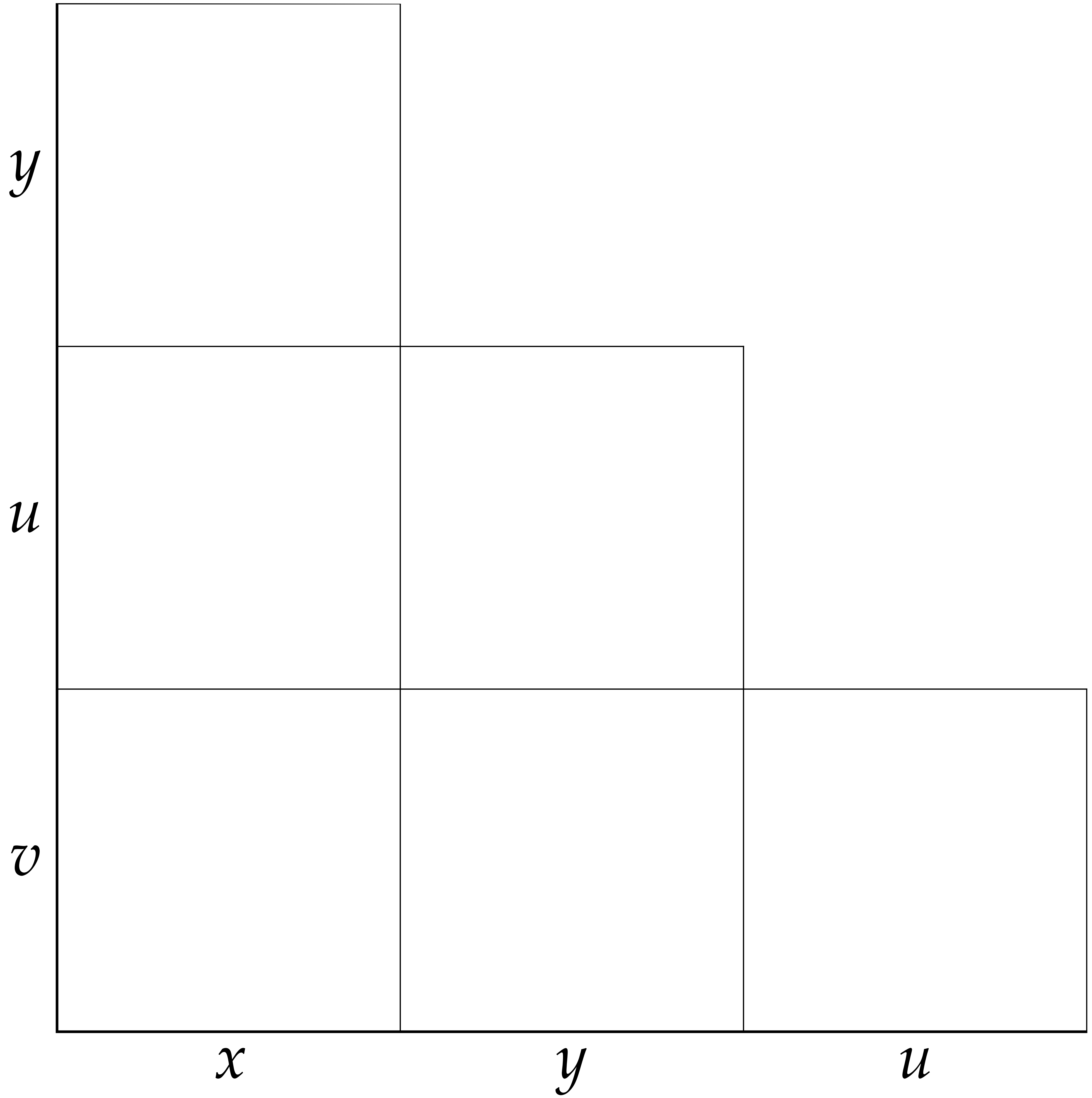
4D

$$(x_1, y_1, u_3, v_3)$$
$$(x_2, y_2, u_1, v_1)$$
$$(x_3, y_3, u_4, v_4)$$
$$(x_4, y_4, u_2, v_2)$$
$$\vdots$$

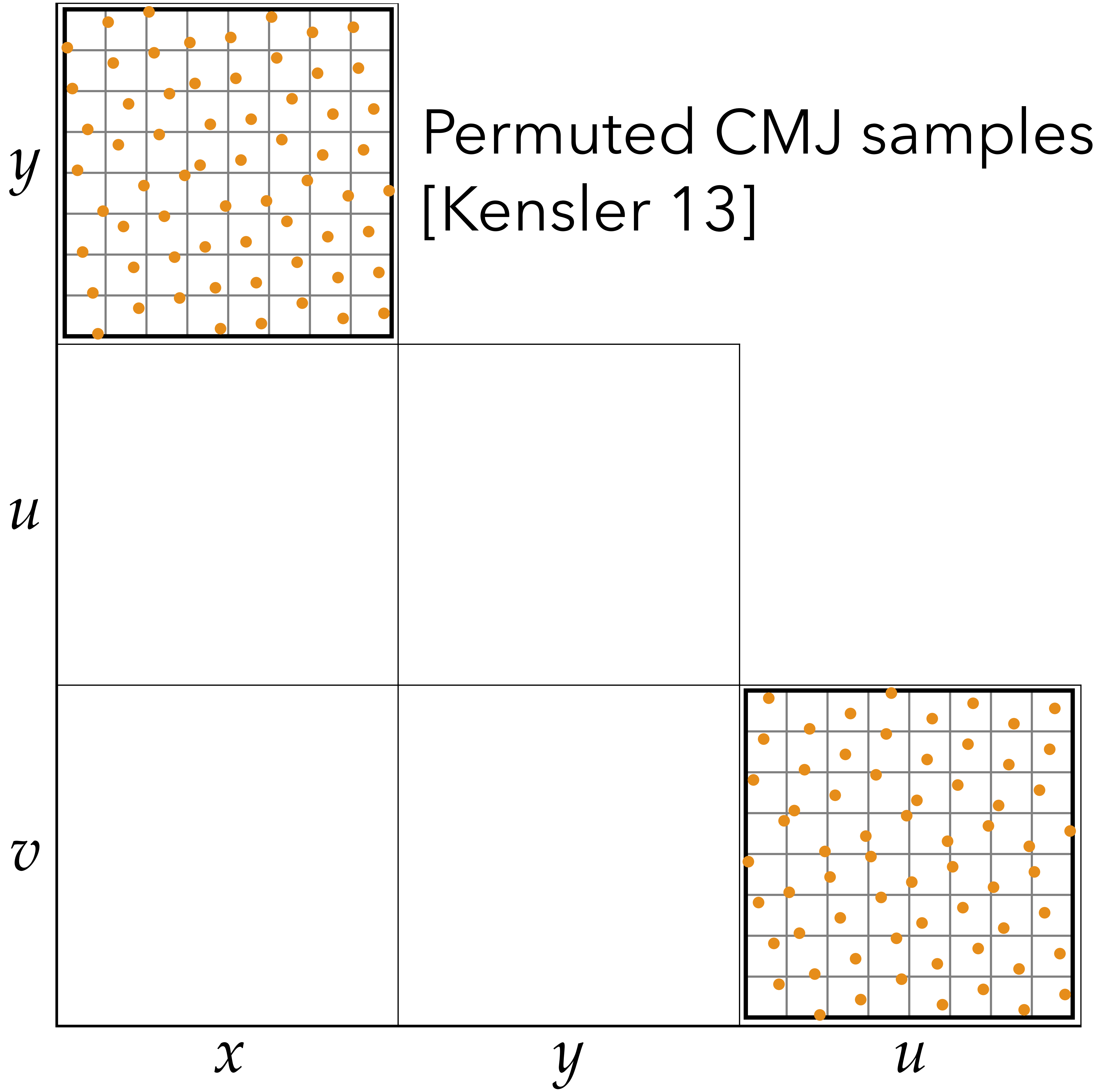
Slide after Gurprit Singh

“Padding” 2D point sets

"Padding" 2D point sets

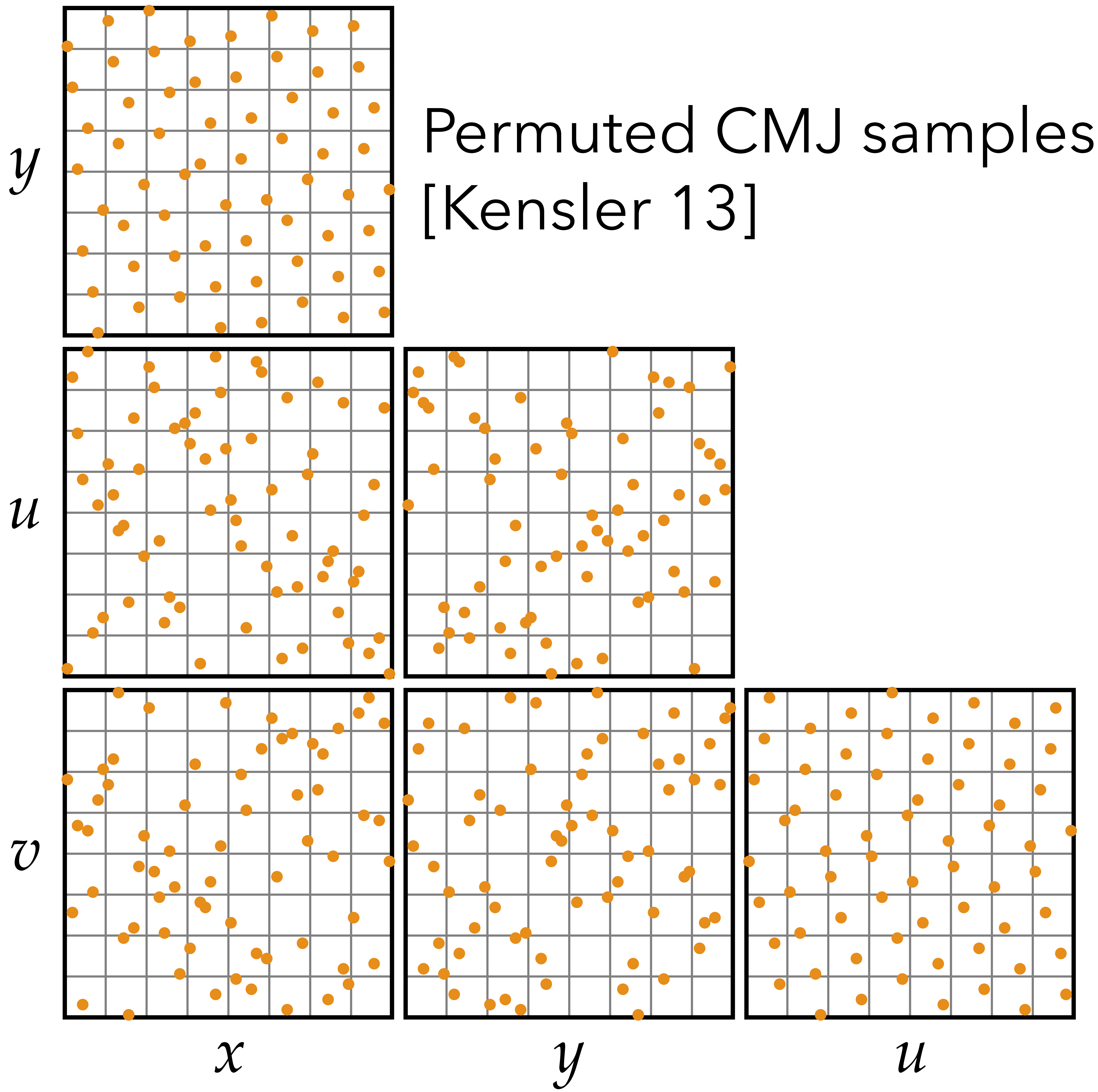


“Padding” 2D point sets



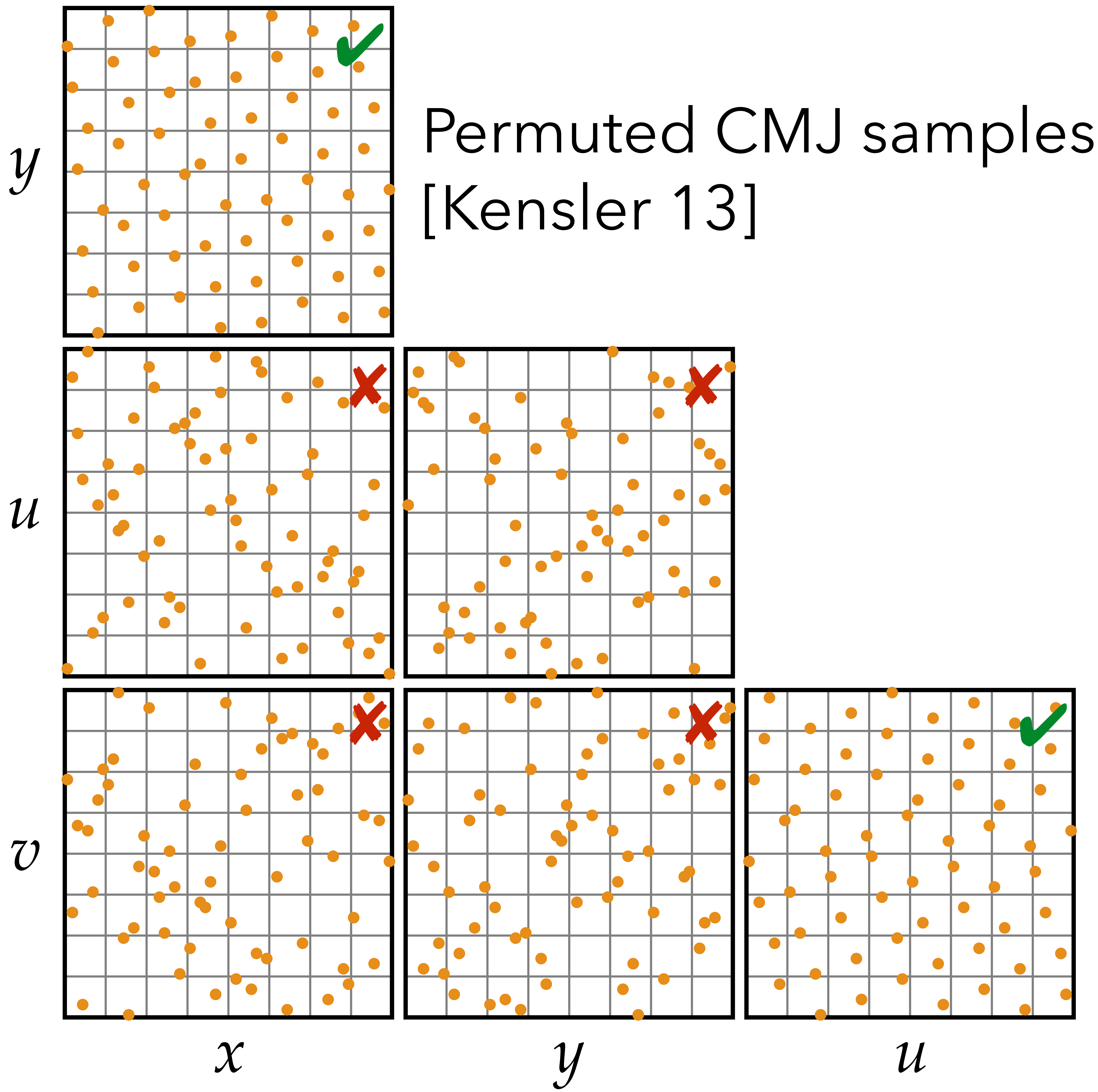
Permuted CMJ samples [Kensler 13]

"Padding" 2D point sets



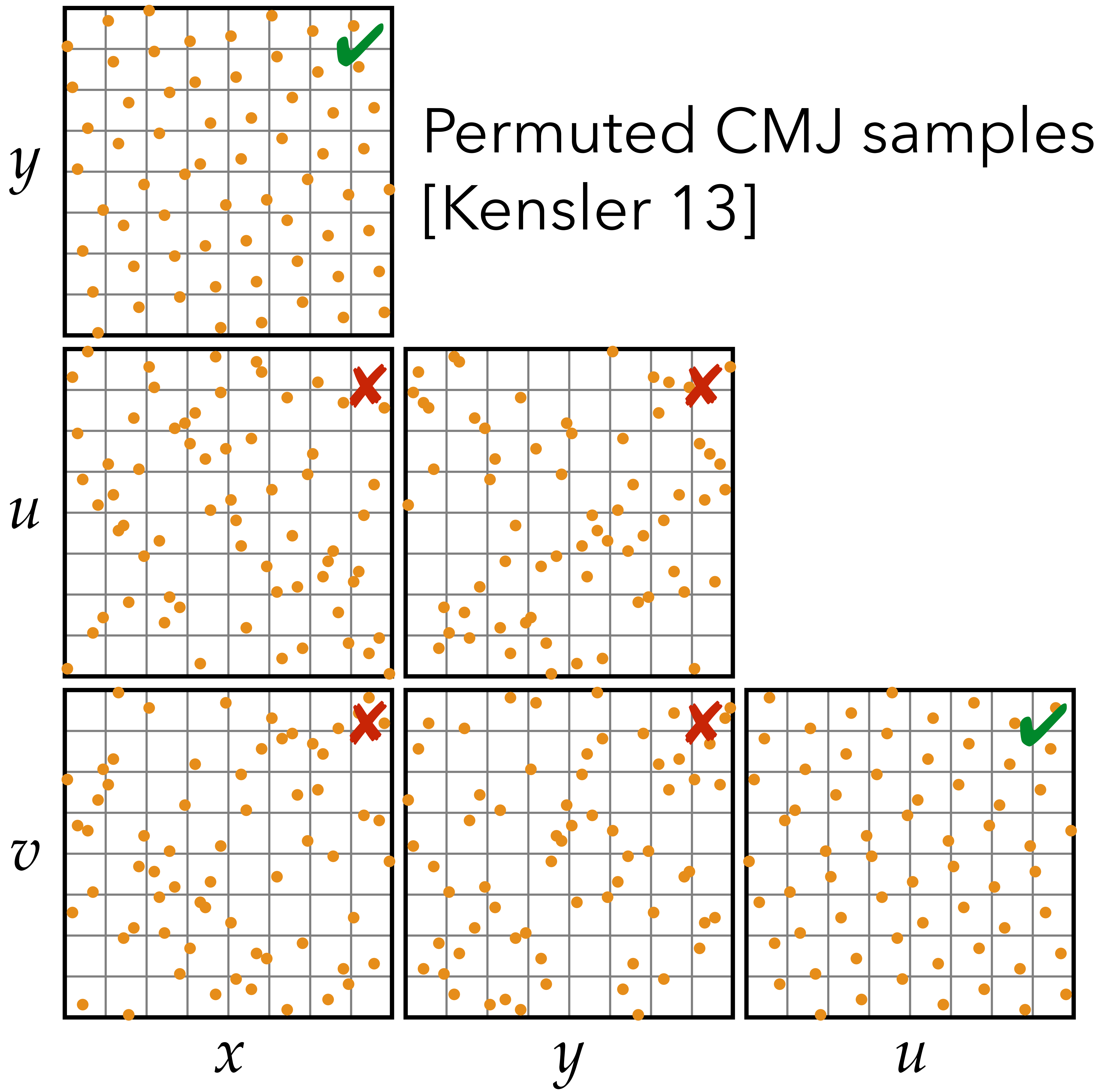
Permuted CMJ samples
[Kensler 13]

"Padding" 2D point sets



Permuted CMJ samples
[Kensler 13]

"Padding" 2D point sets



Permuted CMJ samples
[Kensler 13]

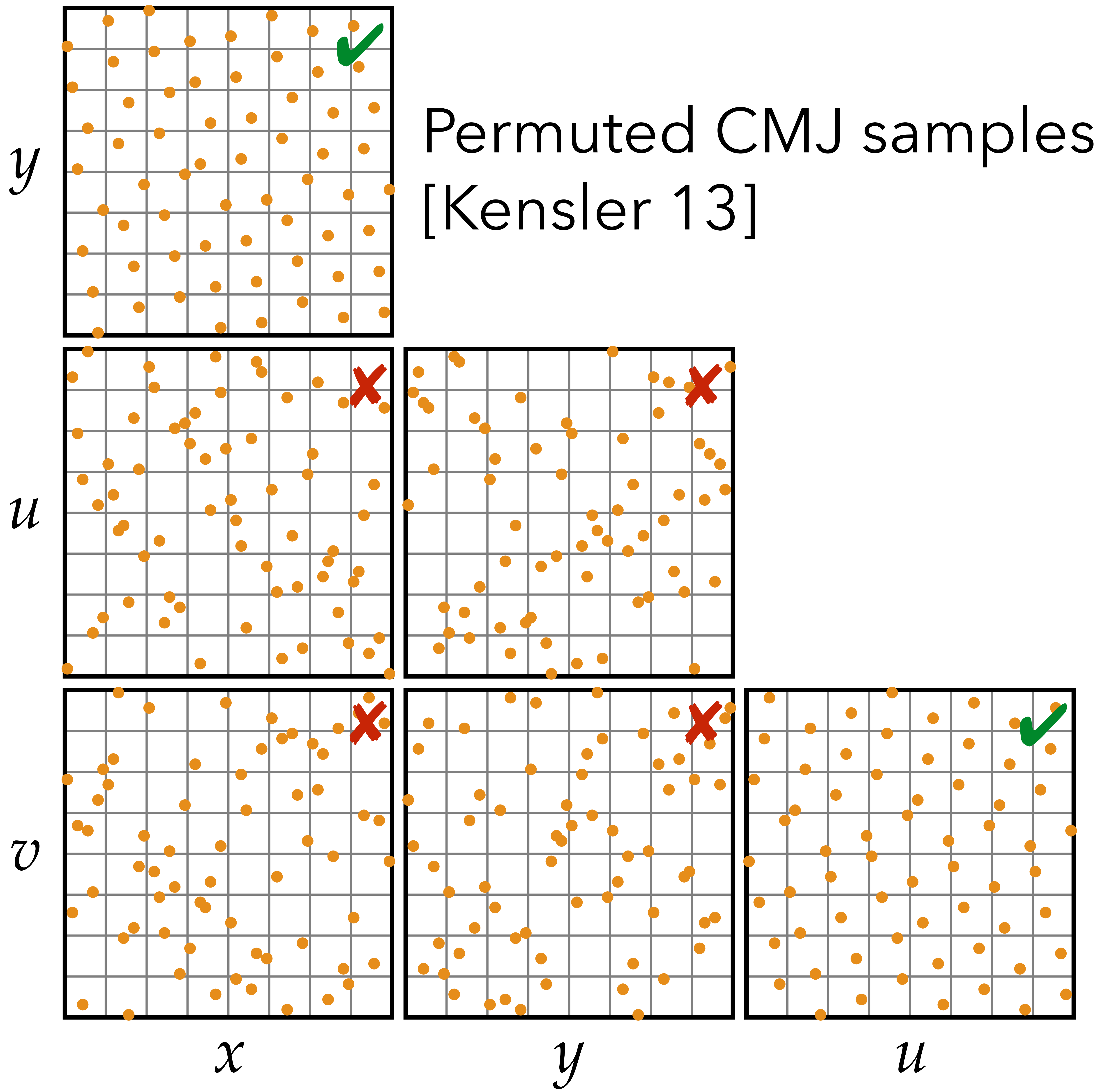
y

u

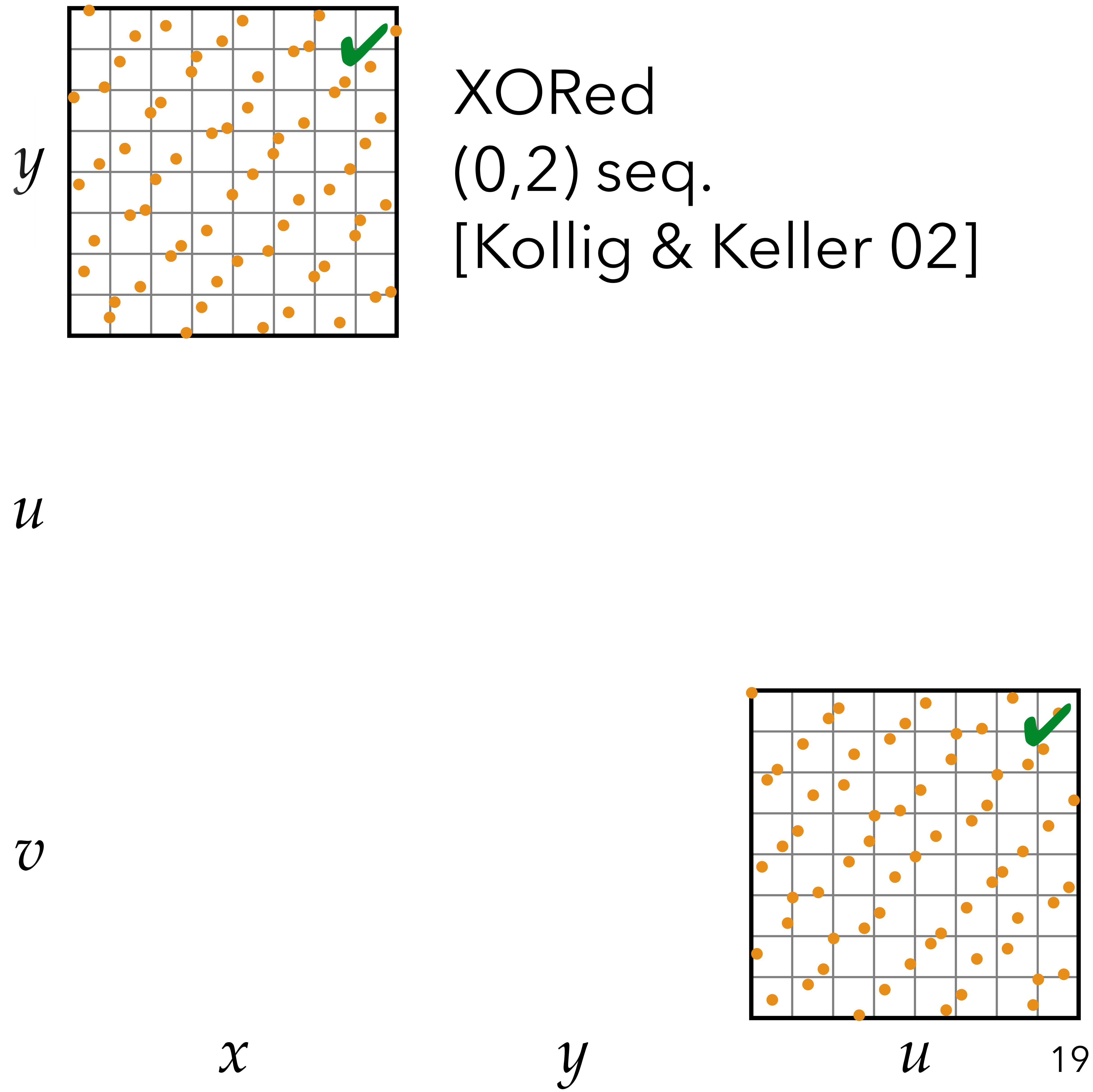
v

x y u

"Padding" 2D point sets

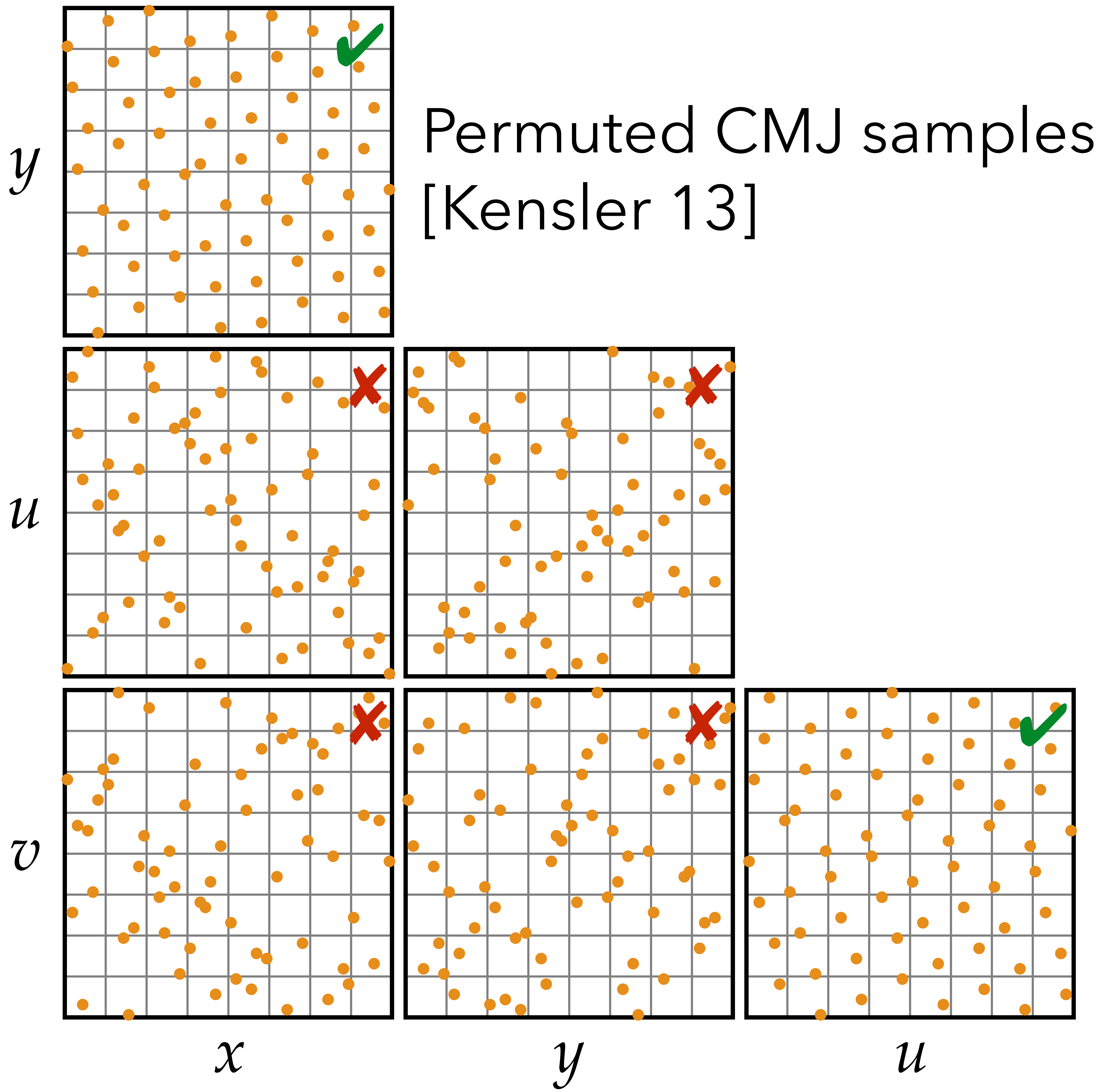


Permuted CMJ samples
[Kensler 13]

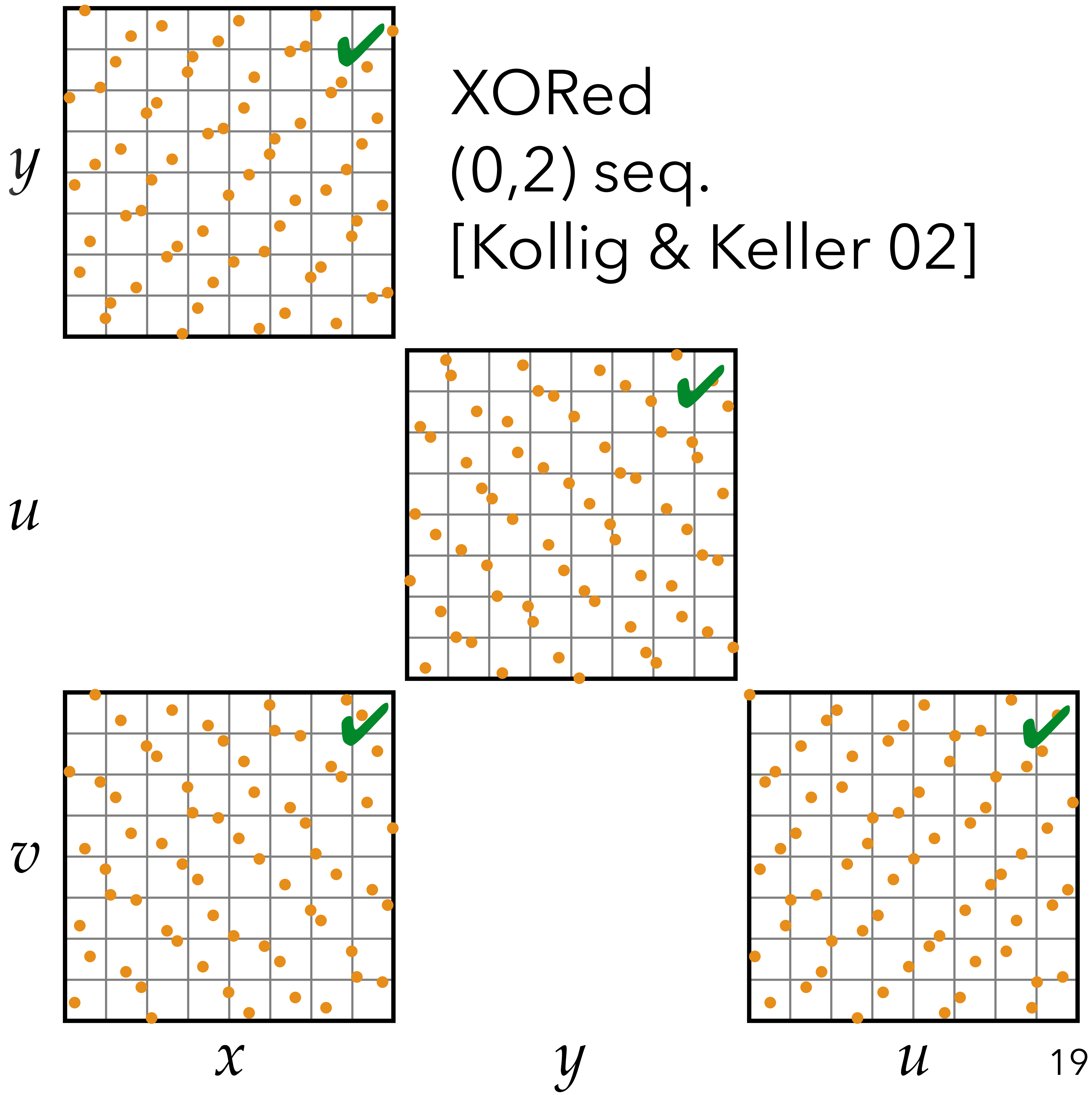


XORed
(0,2) seq.
[Kollig & Keller 02]

"Padding" 2D point sets

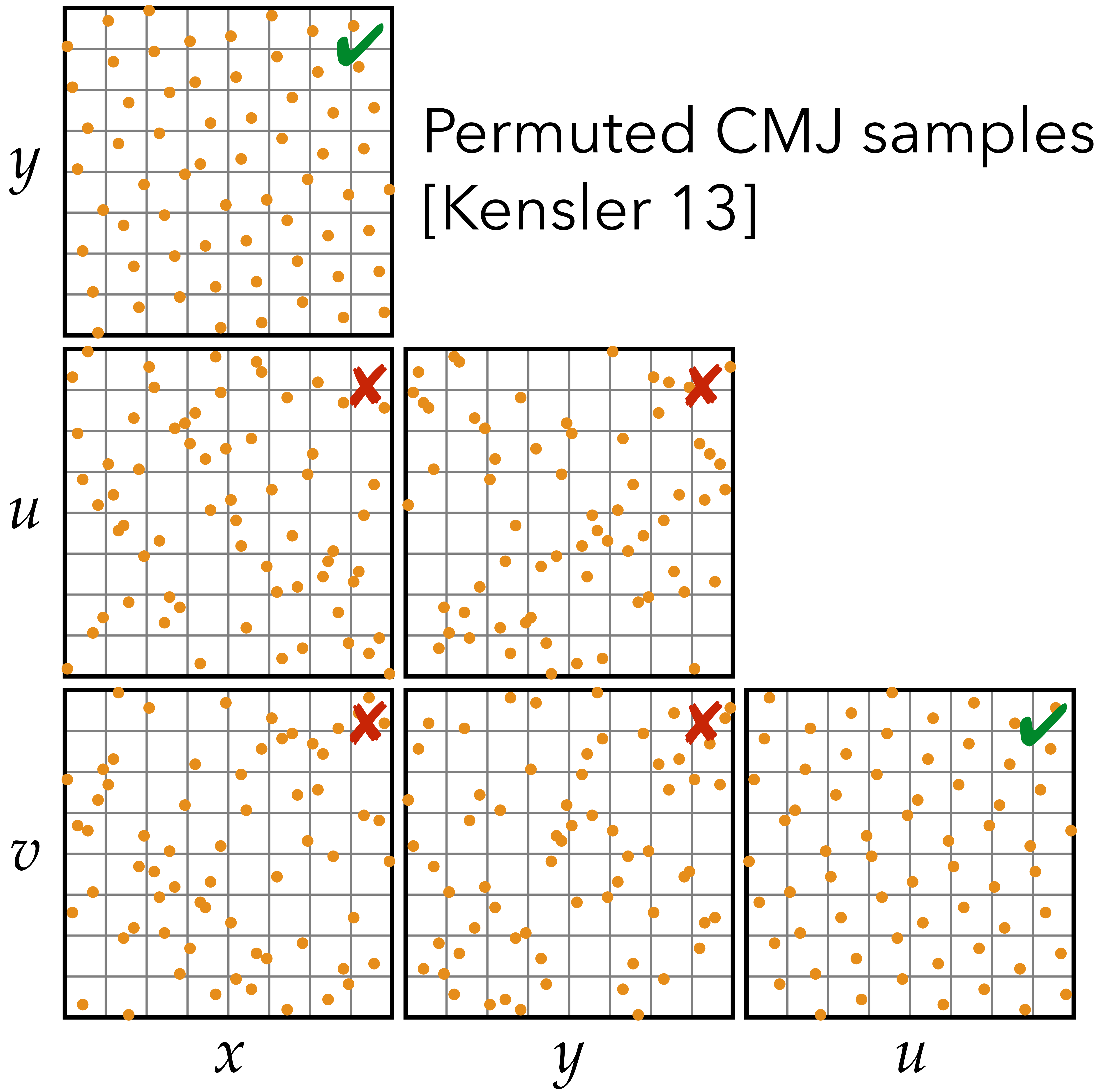


Permuted CMJ samples [Kensler 13]

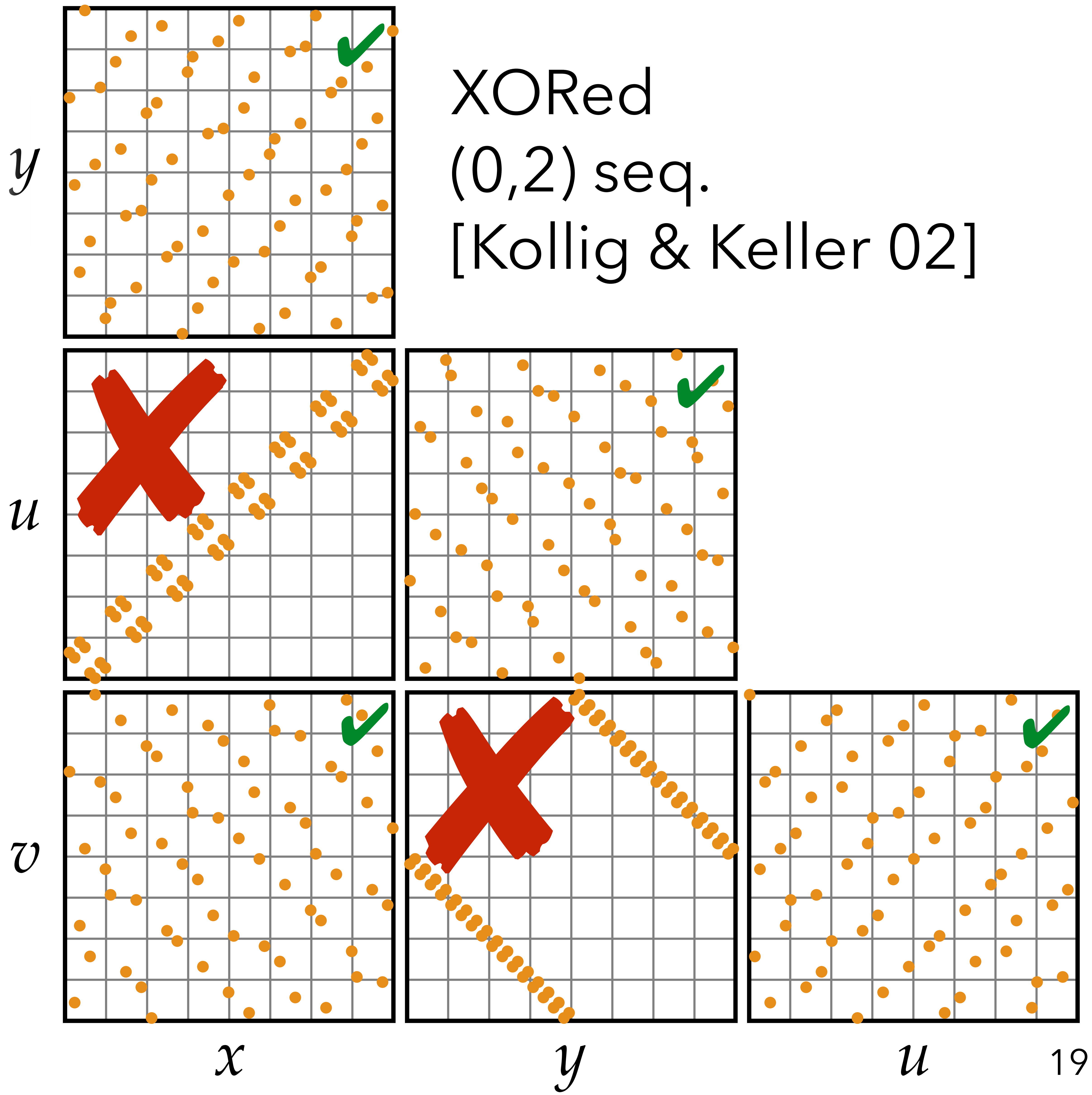


XORed (0,2) seq. [Kollig & Keller 02]

"Padding" 2D point sets

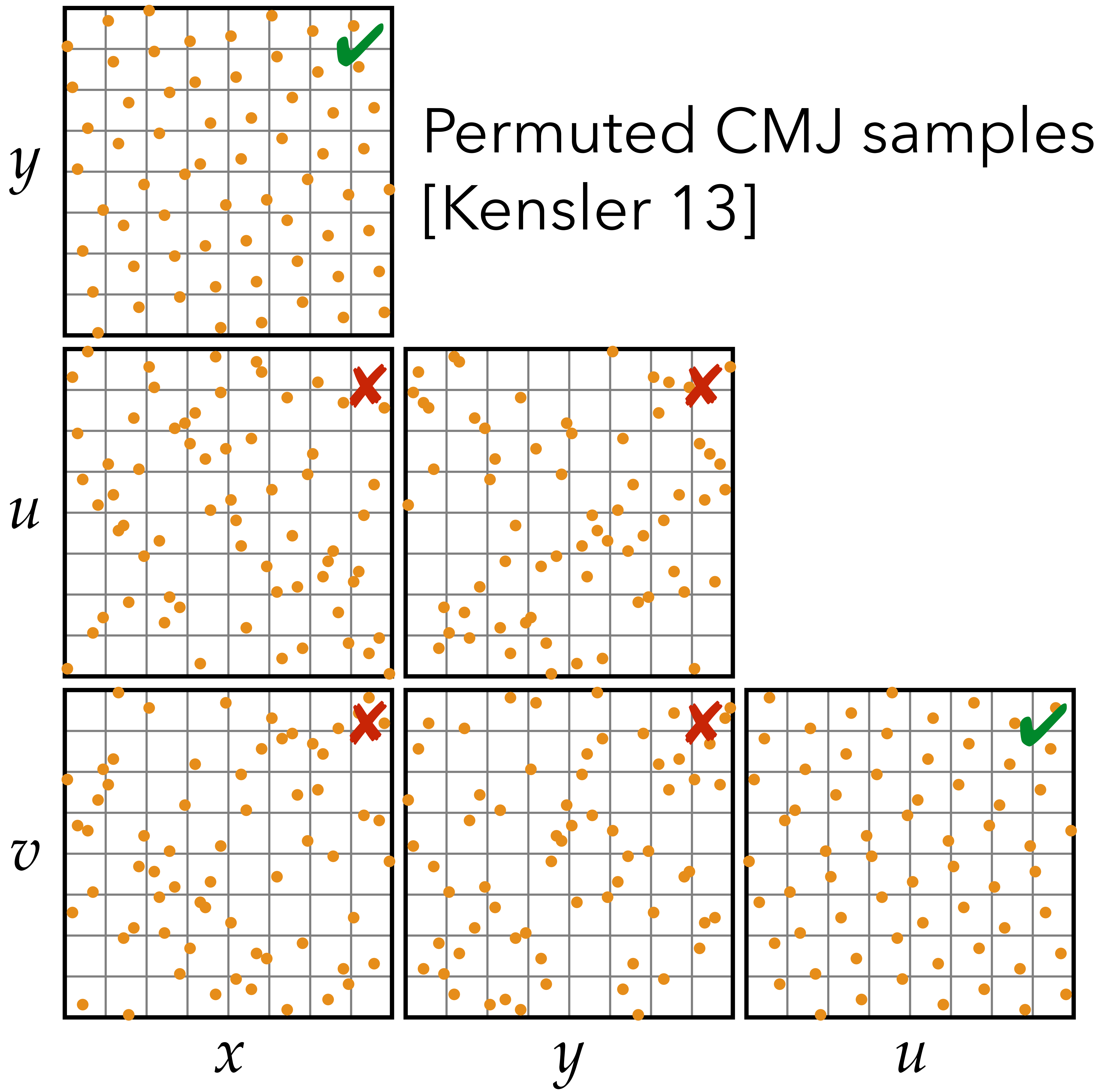


Permuted CMJ samples
[Kensler 13]

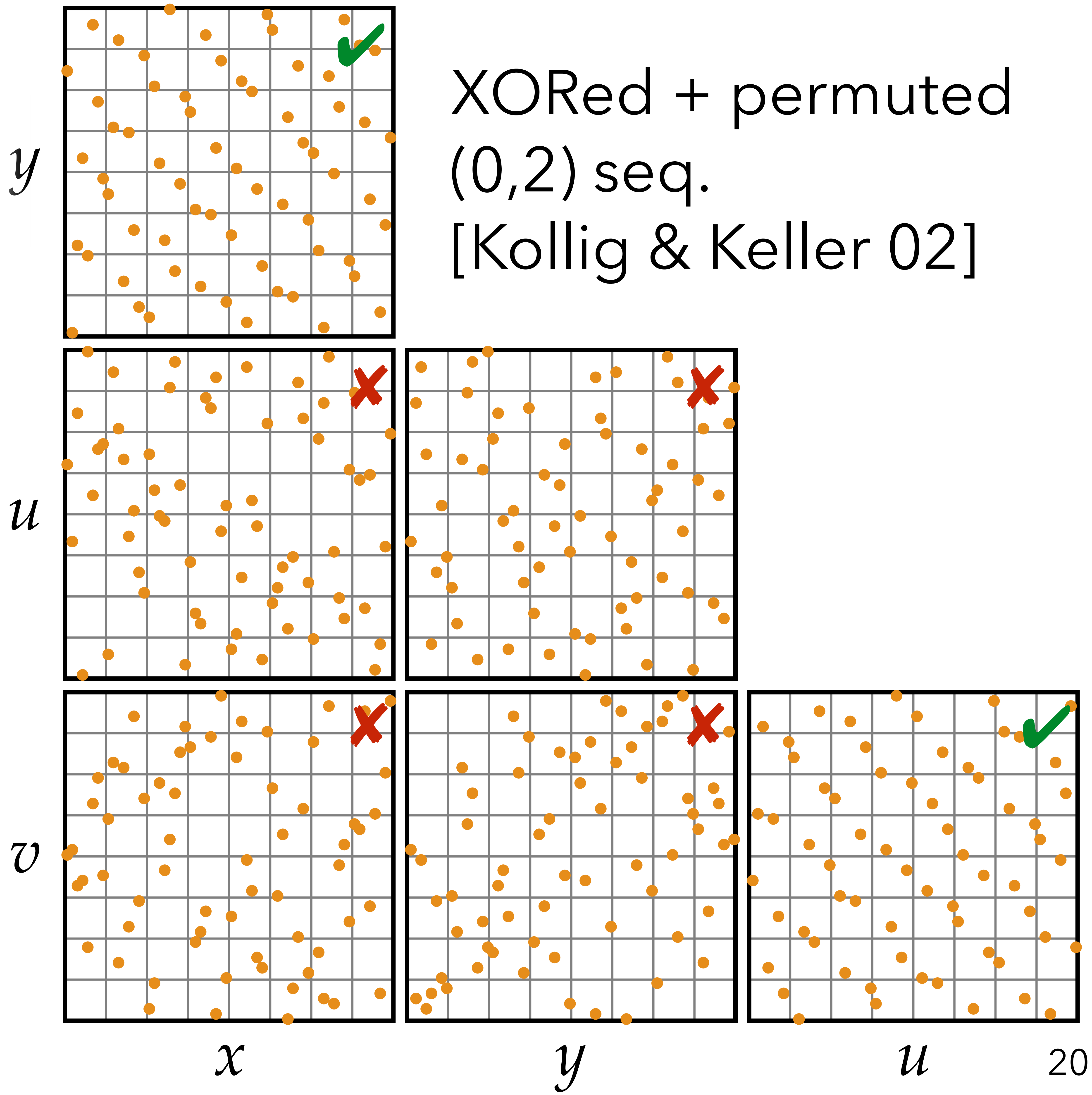


XORed
(0,2) seq.
[Kollig & Keller 02]

"Padding" 2D point sets



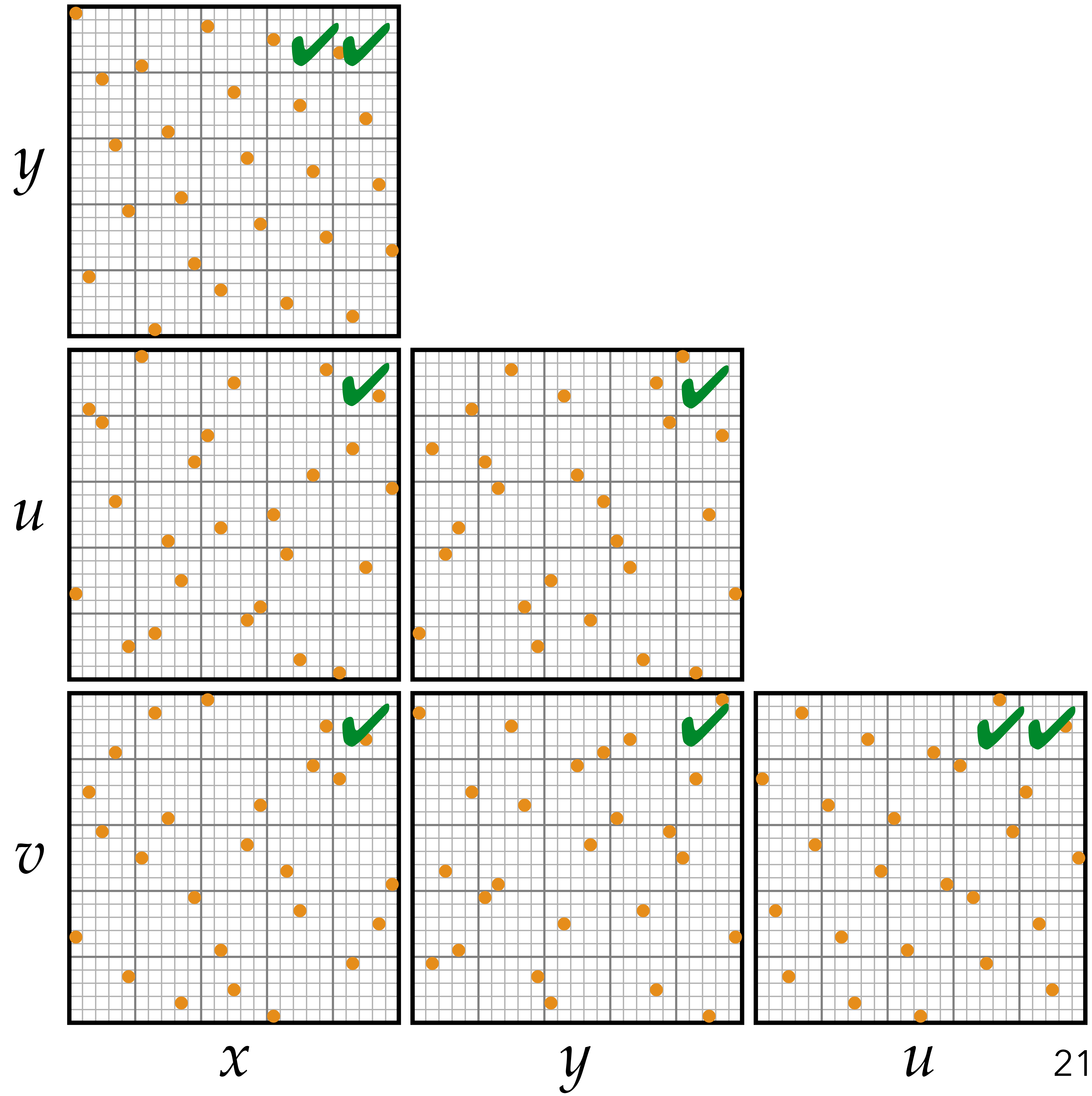
Permuted CMJ samples [Kensler 13]



XORed + permuted (0,2) seq. [Kollig & Keller 02]

Ours: stratifies all 1D and 2D projections

All 2D projections are (correlated) multi-jittered



Contributions

Contributions

Import/apply Orthogonal Arrays to rendering

Contributions

Import/apply Orthogonal Arrays to rendering

- Classic technique (1930s) from statistics/experimental design

Contributions

Import/apply Orthogonal Arrays to rendering

- Classic technique (1930s) from statistics/experimental design
- A precursor to quasi-Monte Carlo

Contributions

Import/apply Orthogonal Arrays to rendering

- Classic technique (1930s) from statistics/experimental design
- A precursor to quasi-Monte Carlo

✔ Natively creates stratified, higher-dimensional points

Contributions

Import/apply Orthogonal Arrays to rendering

- Classic technique (1930s) from statistics/experimental design
- A precursor to quasi-Monte Carlo

✔ Natively creates stratified, higher-dimensional points

Show how to make these fast and practical for rendering

Contributions

Import/apply Orthogonal Arrays to rendering

- Classic technique (1930s) from statistics/experimental design
- A precursor to quasi-Monte Carlo

✔ Natively creates stratified, higher-dimensional points

Show how to make these fast and practical for rendering

Provide a sort of *Rosetta Stone* to this literature

Background on orthogonal arrays

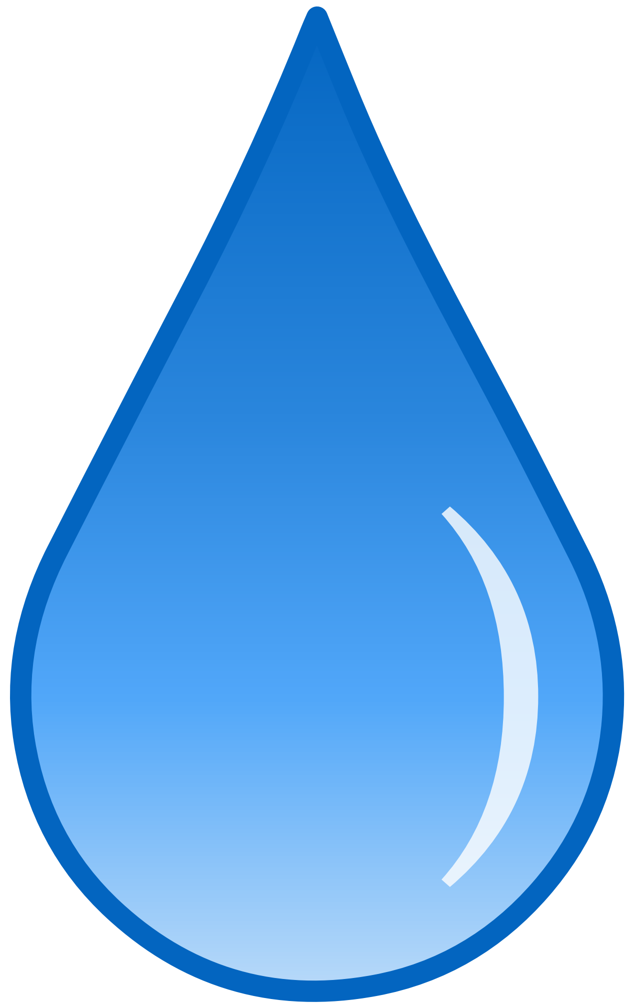
Experimental design

Experimental design

Factors:

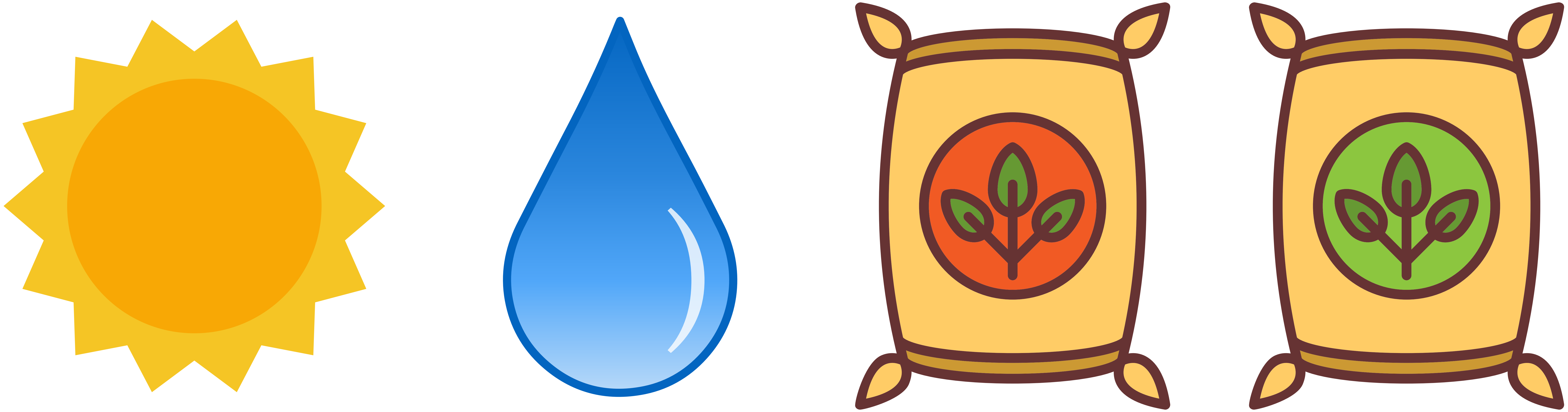
Experimental design

Factors:



Experimental design

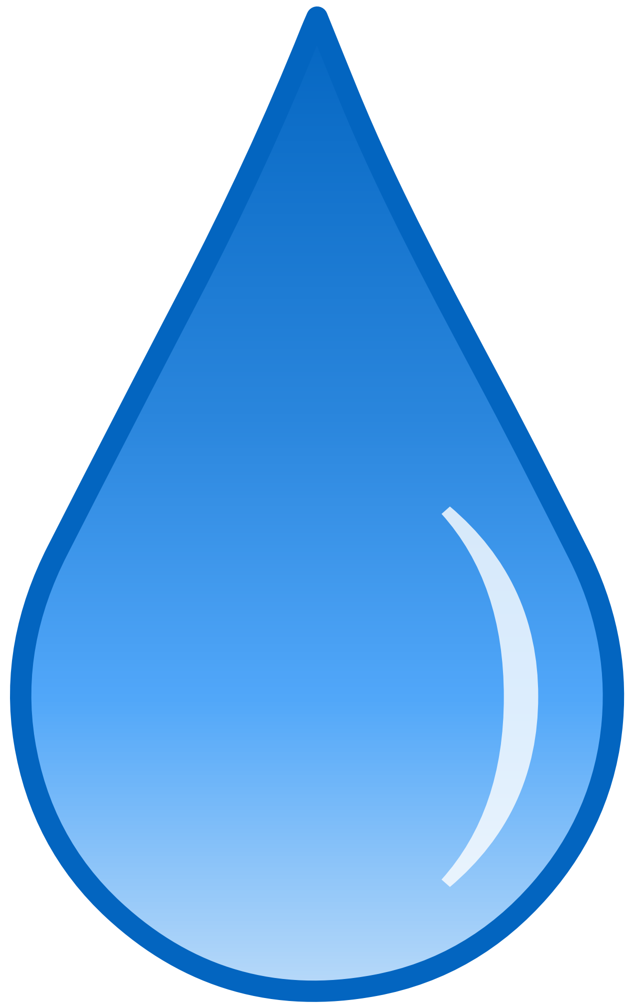
Factors:

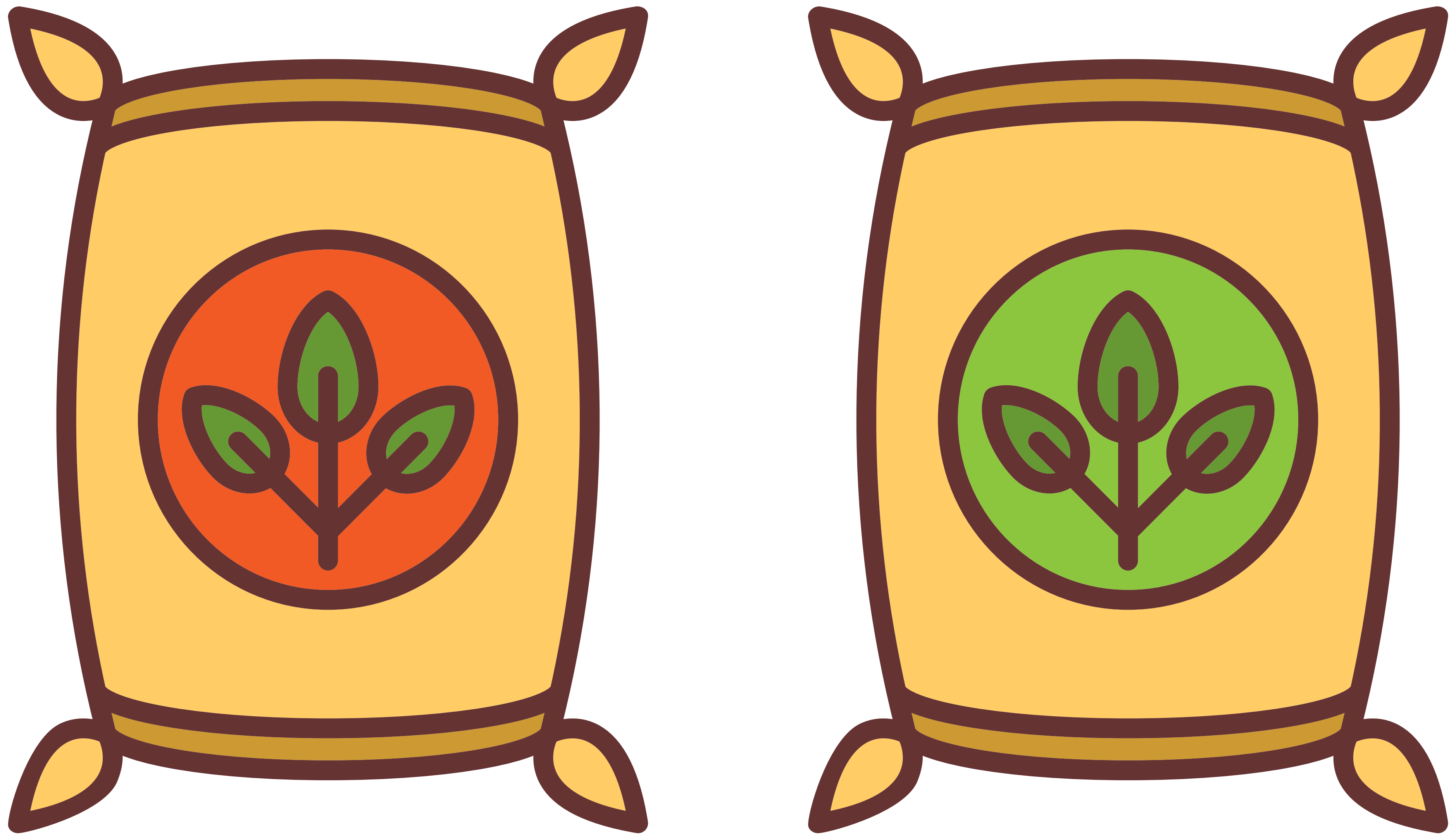


Experimental design

Factors:

$d = 4$

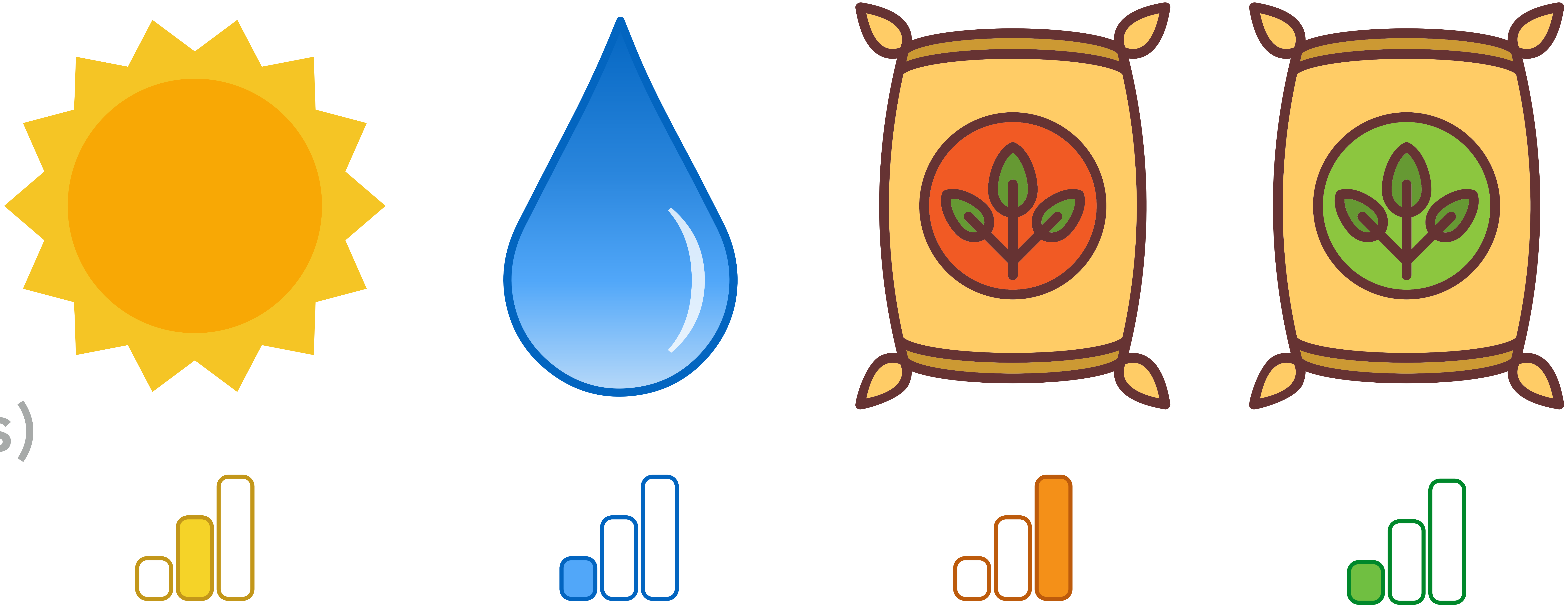




Experimental design

Factors:

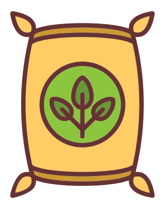
$d = 4$



(amounts)

Levels:

Experimental design



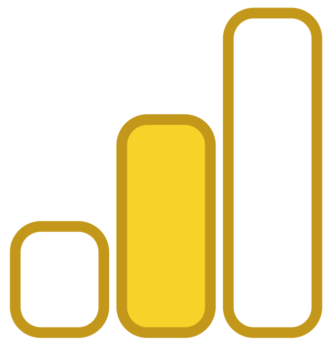
Factors:

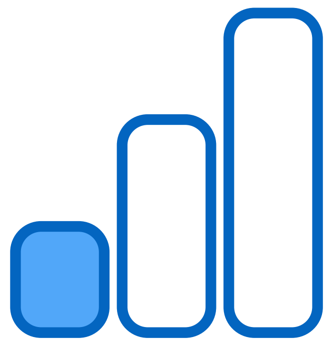
$d = 4$

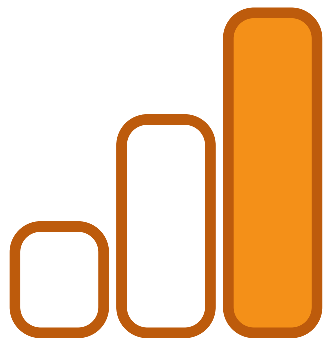
(amounts)

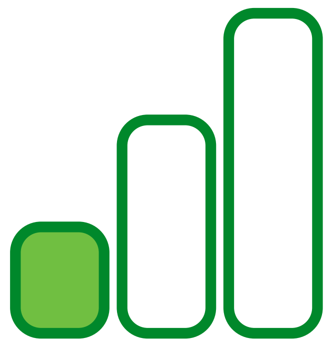
Levels:

$s = 3$

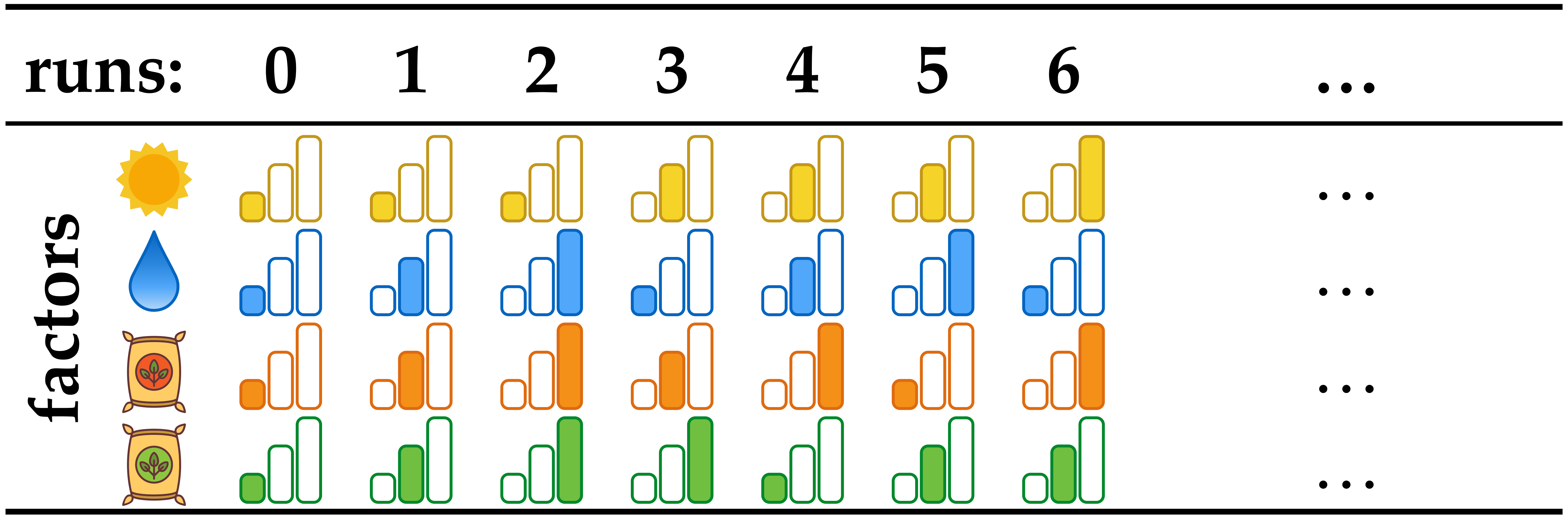








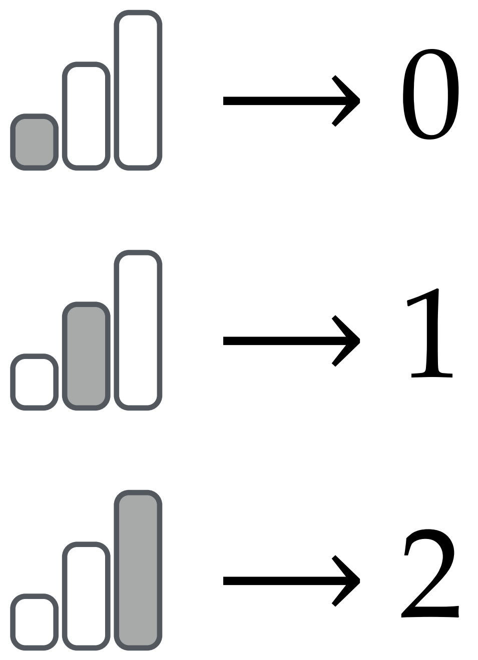
An experiment plan

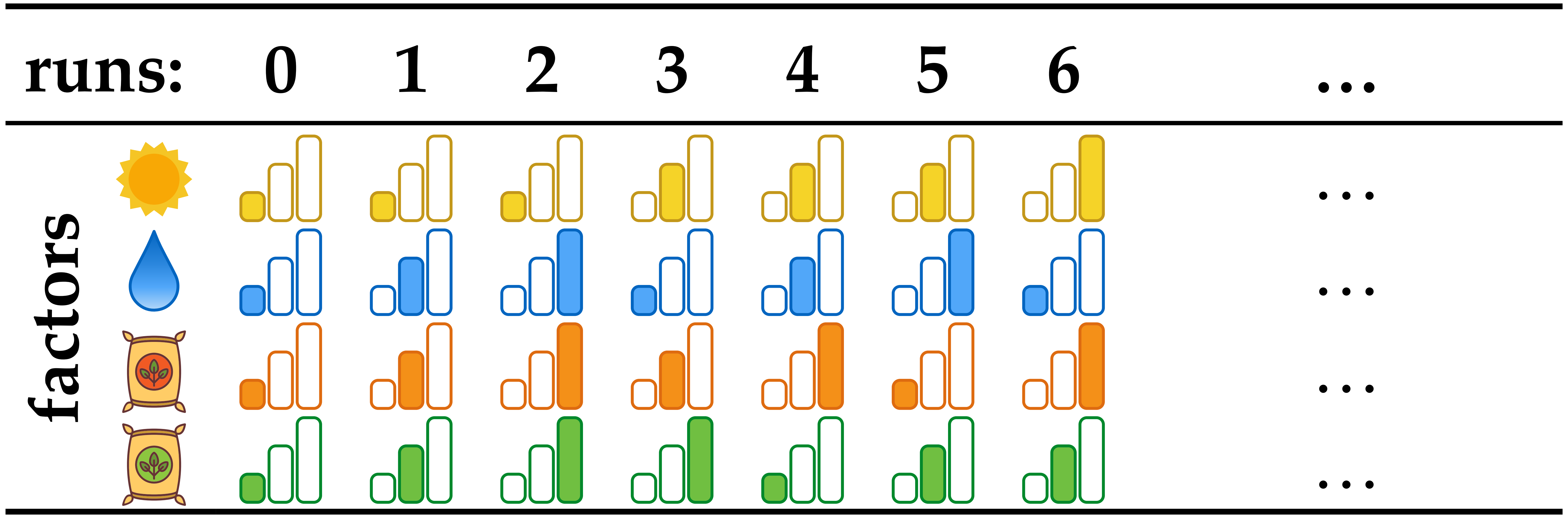


An experiment plan

s discrete levels

$\{0, \ldots, s\text{-}1\}$

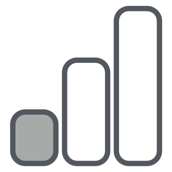


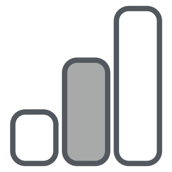


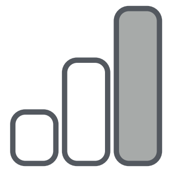
An experiment plan

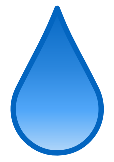
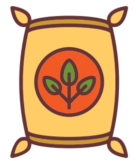
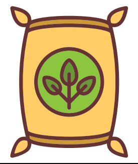
s discrete levels

$\{0, \ldots, s\text{-}1\}$

 $\longrightarrow 0$

 $\longrightarrow 1$

 $\longrightarrow 2$

	runs:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**…**
factors		0	0	0	1	1	1	2	…
		0	1	2	0	1	2	0	…
		0	1	2	1	2	0	2	…
		0	1	2	2	0	1	1	…

An experiment plan

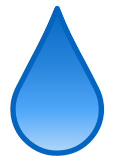
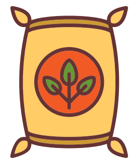
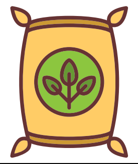
✘ Testing all combinations of factors is expensive: $N = s^d = 81$

runs:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**…**	**80**
factors	0	0	0	1	1	1	2	…	2
	0	1	2	0	1	2	0	…	2
	0	1	2	1	2	0	2	…	1
	0	1	2	2	0	1	1	…	0

An experiment plan

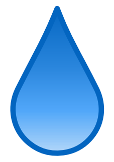
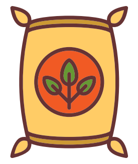
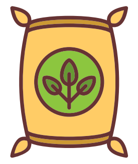
✘ Testing all combinations of factors is expensive: $N = s^d = 81$

- What if we consider at most 2-way interactions?

runs:		0	1	2	3	4	5	6	…	80
factors		0	0	0	1	1	1	2	…	2
		0	1	2	0	1	2	0	…	2
		0	1	2	1	2	0	2	…	1
		0	1	2	2	0	1	1	…	0

Orthogonal arrays (OAs)

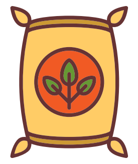
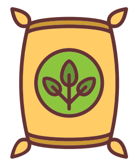
A **strength** $t = 2$ OA considers all 2-way interactions

runs:		0	1	2	3	4	5	6	7	8
factors		0	0	0	1	1	1	2	2	2
		0	1	2	0	1	2	0	1	2
		0	1	2	1	2	0	2	0	1
		0	1	2	2	0	1	1	2	0

Orthogonal arrays (OAs)

A **strength** $t = 2$ OA considers all 2-way interactions

Every combination of levels in these $t = 2$ factors is tested.

		runs:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
factors			0	0	0	1	1	1	2	2	2
			0	1	2	0	1	2	0	1	2
			0	1	2	1	2	0	2	0	1
			0	1	2	2	0	1	1	2	0

Orthogonal arrays (OAs)

A **strength** $t = 2$ OA considers all 2-way interactions

Every combination of levels in these $t = 2$ factors is tested.

runs:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
factors	0	0	0	1	1	1	2	2	2
	0	1	2	0	1	2	0	1	2
	0	1	2	1	2	0	2	0	1
	0	1	2	2	0	1	1	2	0

$s^2 = 9$ possible combinations

{0,0},
{0,1},
{0,2},
{1,0},
{1,1},
{1,2},
{2,0},
{2,1},
{2,2}

Orthogonal arrays (OAs)

A **strength** $t = 2$ OA considers all 2-way interactions

Every combination of levels in these $t = 2$ factors is tested.

$s^2 = 9$ possible combinations

	runs:	0	1	2	3	4	5	6	7	8
factors		0	0	0	1	1	1	2	2	2
		0	1	2	0	1	2	0	1	2
		0	1	2	1	2	0	2	0	1
		0	1	2	2	0	1	1	2	0

{0,0},
{0,1},
{0,2},
{1,0},
{1,1},
{1,2},
{2,0},
{2,1},
{2,2}

And these too.

Orthogonal arrays (OAs)

A **strength** $t = 2$ OA considers all 2-way interactions

Every combination of levels in these $t = 2$ factors is tested.

	runs:	0	1	2	3	4	5	6	7	8
factors	sun	0	0	0	1	1	1	2	2	2
	water	0	1	2	0	1	2	0	1	2
	seed bag (red)	0	1	2	1	2	0	2	0	1
	seed bag (green)	0	1	2	2	0	1	1	2	0

$s^2 = 9$ possible combinations

{0,0},
{0,1},
{0,2},
{1,0},
{1,1},
{1,2},
{2,0},
{2,1},
{2,2}

Yes, these too.

Orthogonal arrays (OAs)

A **strength** $t = 2$ OA considers all 2-way interactions

Every combination of levels in **any** $t = 2$ factors is tested.

runs:	0	1	2	3	4	5	6	7	8
factors (sun)	0	0	0	1	1	1	2	2	2
factors (water)	0	1	2	0	1	2	0	1	2
factors (fertilizer)	0	1	2	1	2	0	2	0	1
factors (seeds)	0	1	2	2	0	1	1	2	0

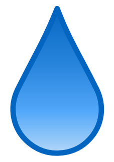
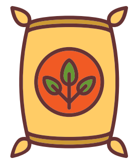
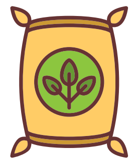
$s^2 = 9$ possible combinations

{0,0},
{0,1},
{0,2},
{1,0},
{1,1},
{1,2},
{2,0},
{2,1},
{2,2}

Orthogonal arrays (OAs)

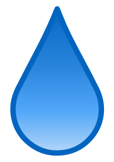
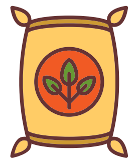
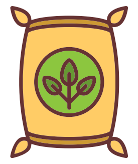
A **strength** $t = 2$ OA considers all 2-way interactions

Every combination of levels in **any** $t = 2$ factors is tested.

	runs:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
factors		0	0	0	1	1	1	2	2	2
		0	1	2	0	1	2	0	1	2
		0	1	2	1	2	0	2	0	1
		0	1	2	2	0	1	1	2	0

Now we only need $s^t = 3^2 = 9$ runs (for $s = 3$ levels at strength $t = 2$)!

Orthogonal arrays (OAs)

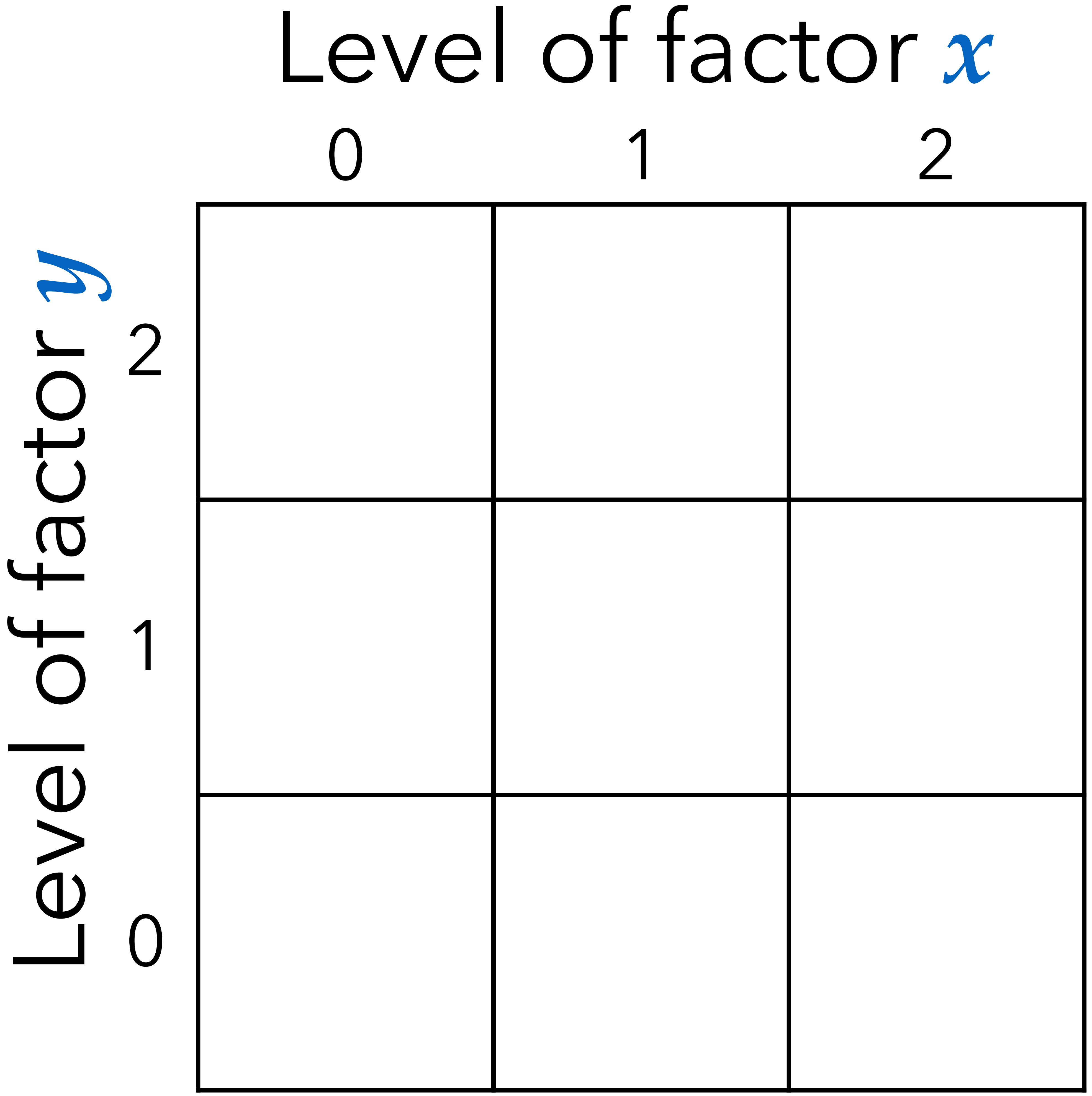
runs:		0	1	2	3	4	5	6	7	8
factors		0	0	0	1	1	1	2	2	2
		0	1	2	0	1	2	0	1	2
		0	1	2	1	2	0	2	0	1
		0	1	2	2	0	1	1	2	0

Orthogonal arrays (OAs)

	runs:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
factors	x:	0	0	0	1	1	1	2	2	2
	y:	0	1	2	0	1	2	0	1	2
	u:	0	1	2	1	2	0	2	0	1
	v:	0	1	2	2	0	1	1	2	0

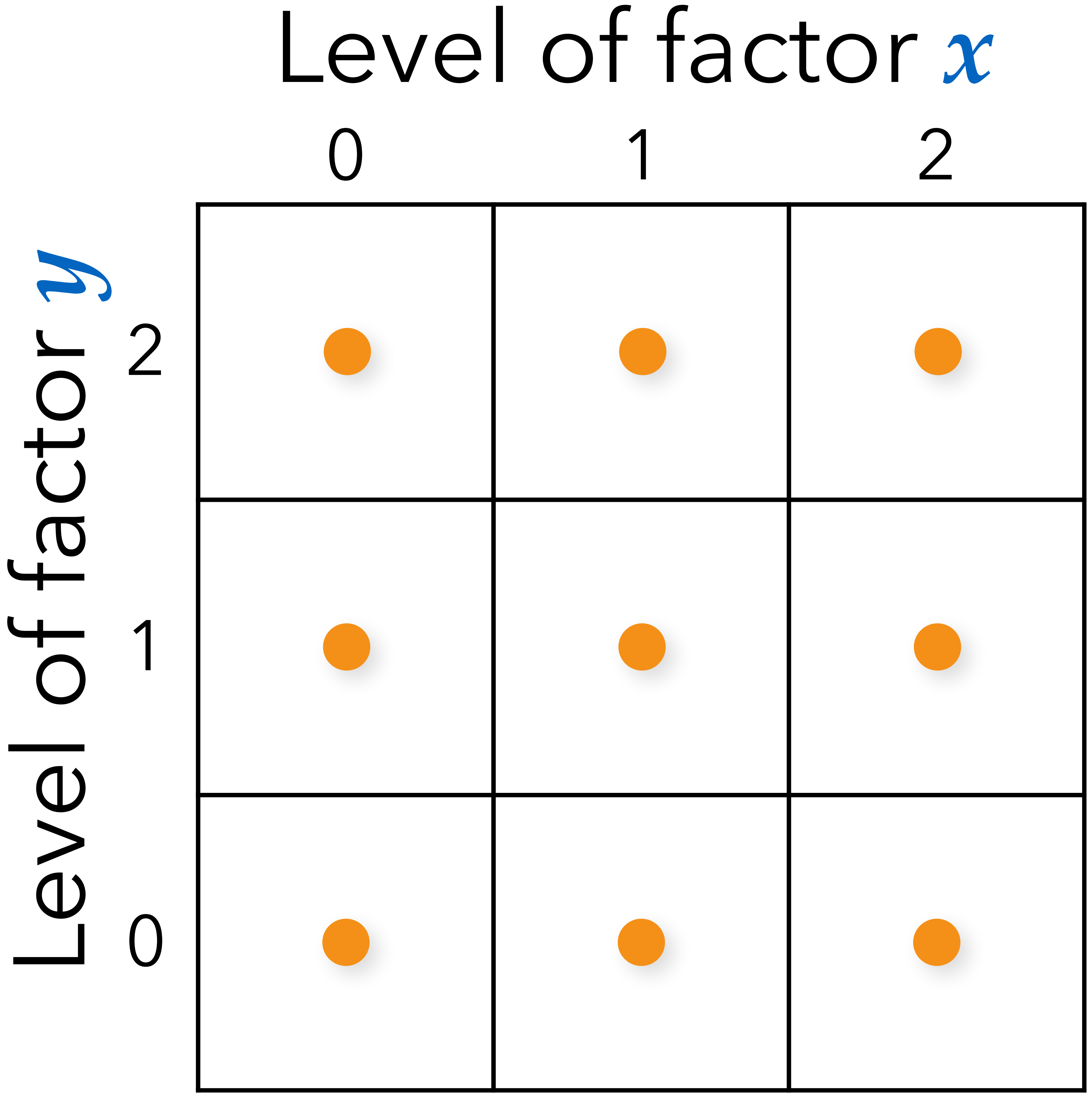
Orthogonal arrays (graphically)

	runs:	0	1	2	3	4	5	6	7	8
factors	x:	0	0	0	1	1	1	2	2	2
	y:	0	1	2	0	1	2	0	1	2
	u:	0	1	2	1	2	0	2	0	1
	v:	0	1	2	2	0	1	1	2	0



Orthogonal arrays (graphically)

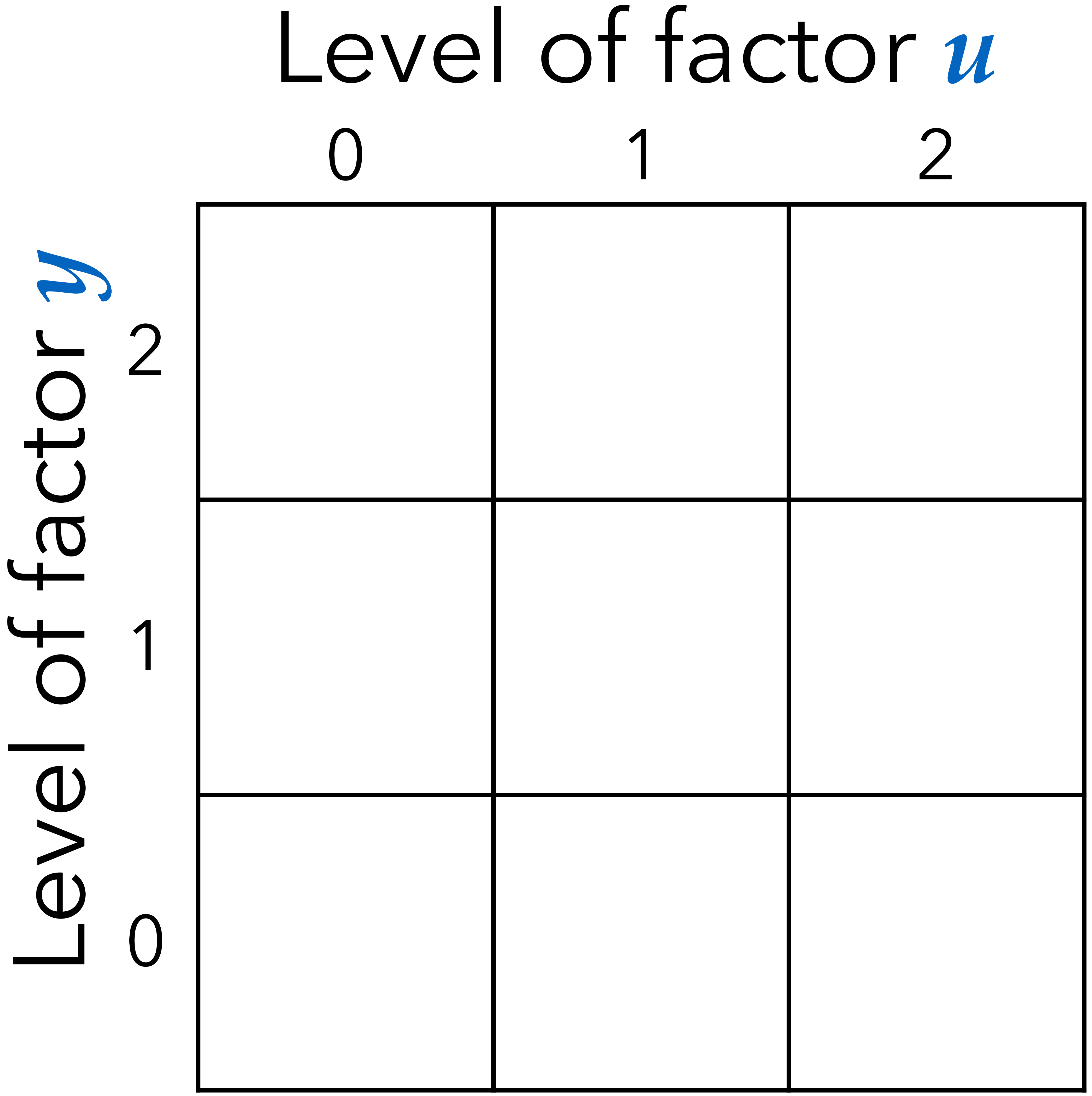
runs:		0	1	2	3	4	5	6	7	8
factors	x:	0	0	0	1	1	1	2	2	2
	y:	0	1	2	0	1	2	0	1	2
	u:	0	1	2	1	2	0	2	0	1
	v:	0	1	2	2	0	1	1	2	0



Orthogonal arrays (graphically)

runs:	0	1	2	3	4	5	6	7	8
x:	0	0	0	1	1	1	2	2	2
y:	0	1	2	0	1	2	0	1	2
u:	0	1	2	1	2	0	2	0	1
v:	0	1	2	2	0	1	1	2	0

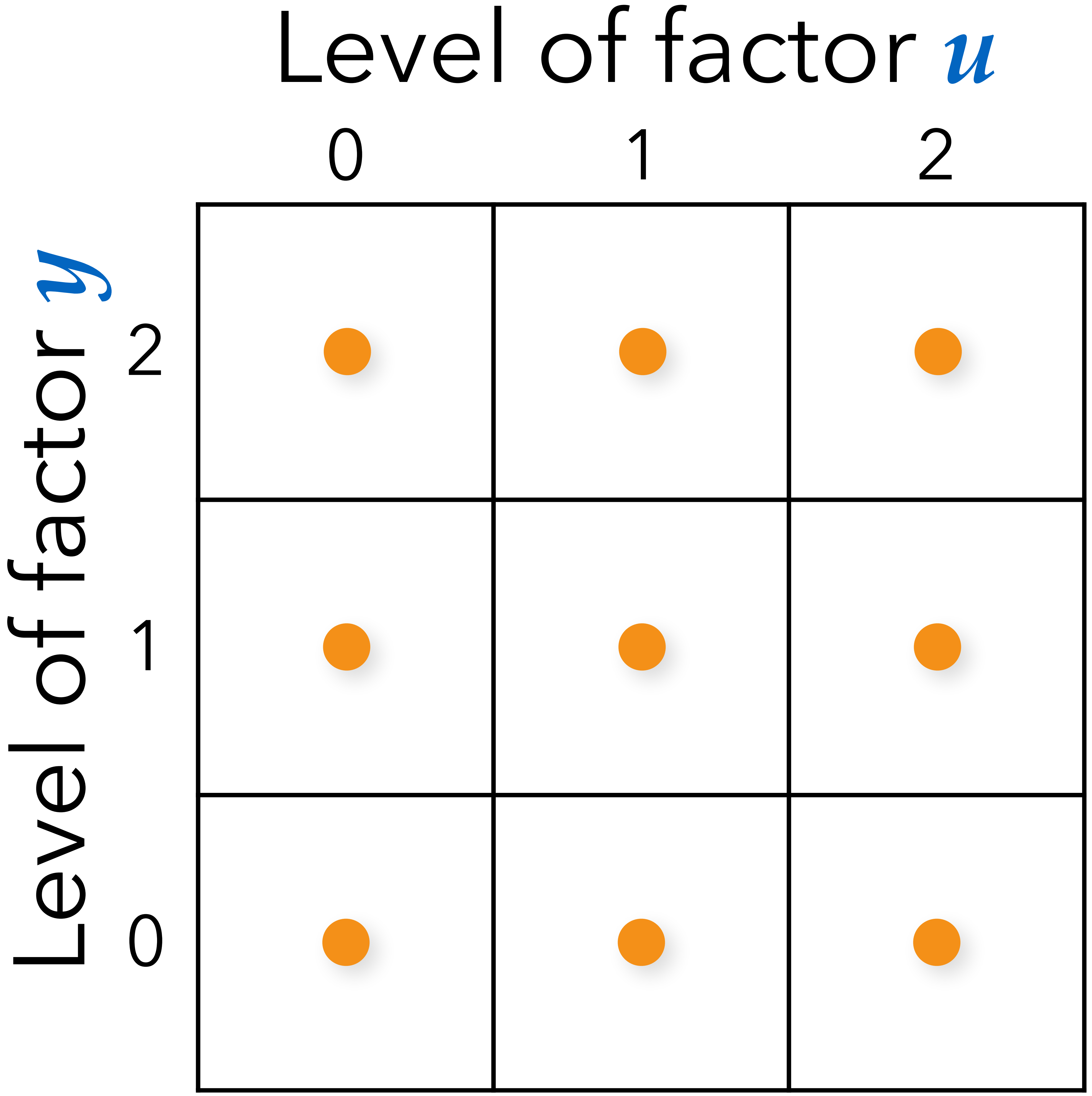
factors



Orthogonal arrays (graphically)

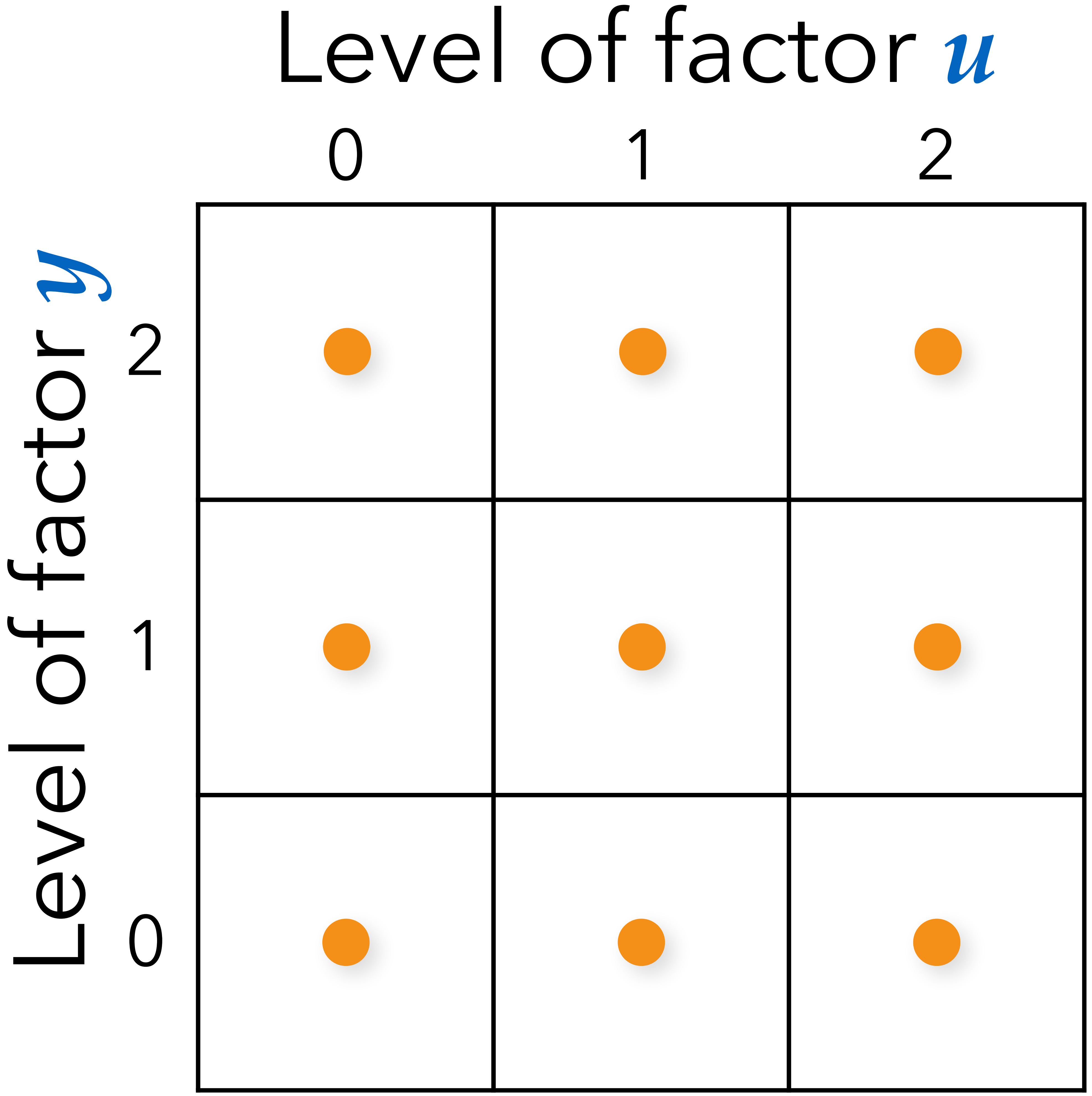
runs:	0	1	2	3	4	5	6	7	8
x:	0	0	0	1	1	1	2	2	2
y:	0	1	2	0	1	2	0	1	2
u:	0	1	2	1	2	0	2	0	1
v:	0	1	2	2	0	1	1	2	0

factors



Orthogonal arrays (graphically)

runs:	0	1	2	3	4	5	6	7	8
x:	0	0	0	1	1	1	2	2	2
factors y:	0	1	2	0	1	2	0	1	2
factors u:	0	1	2	1	2	0	2	0	1
v:	0	1	2	2	0	1	1	2	0



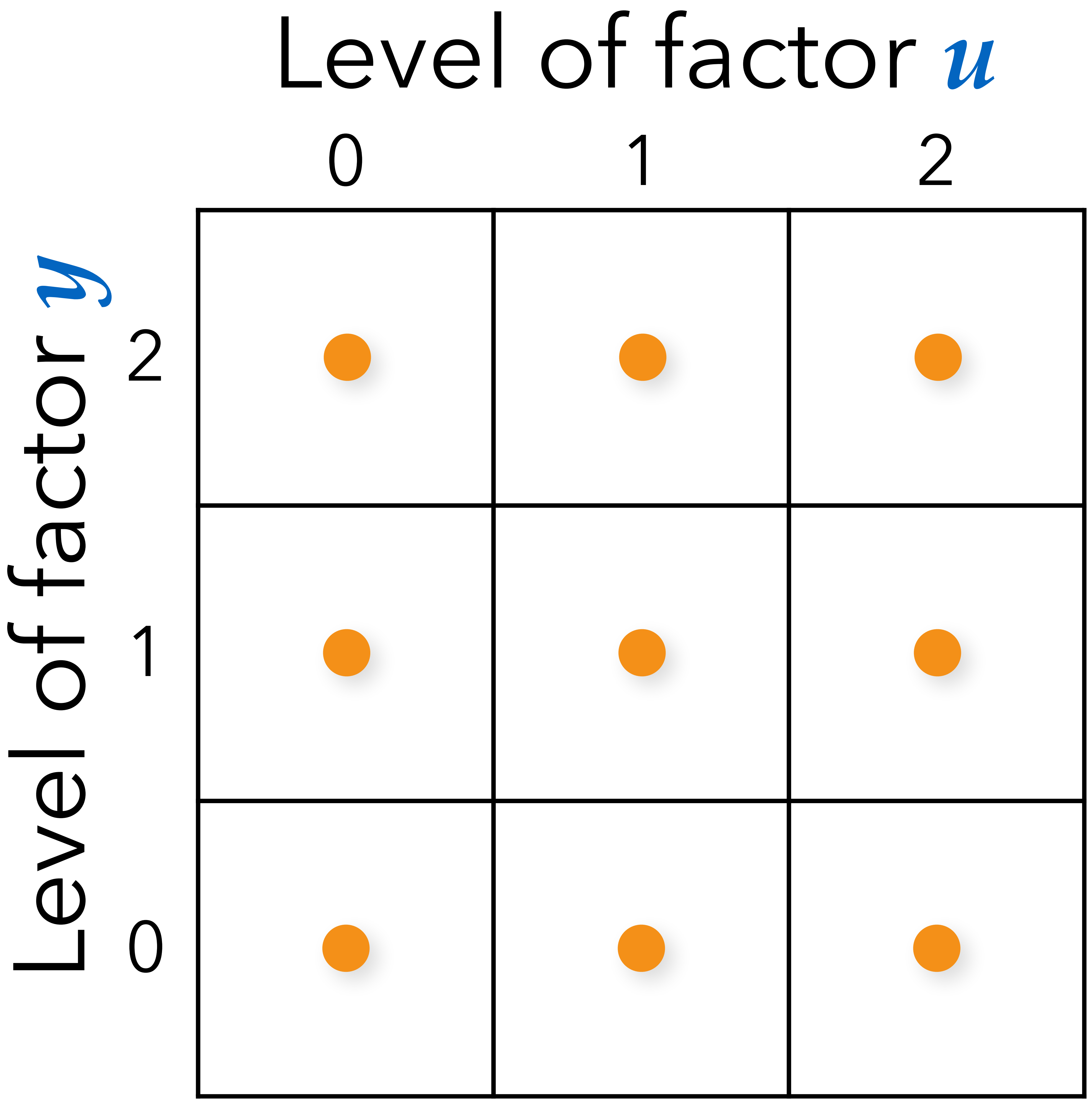
This OA encodes **nine 4D points,**

Orthogonal arrays (graphically)

runs:	0	1	2	3	4	5	6	7	8
factors x:	0	0	0	1	1	1	2	2	2
y:	0	1	2	0	1	2	0	1	2
u:	0	1	2	1	2	0	2	0	1
v:	0	1	2	2	0	1	1	2	0

This OA encodes **nine 4D points**,

which project to a regular **3** × **3** grid when plotting any pair of dimensions.



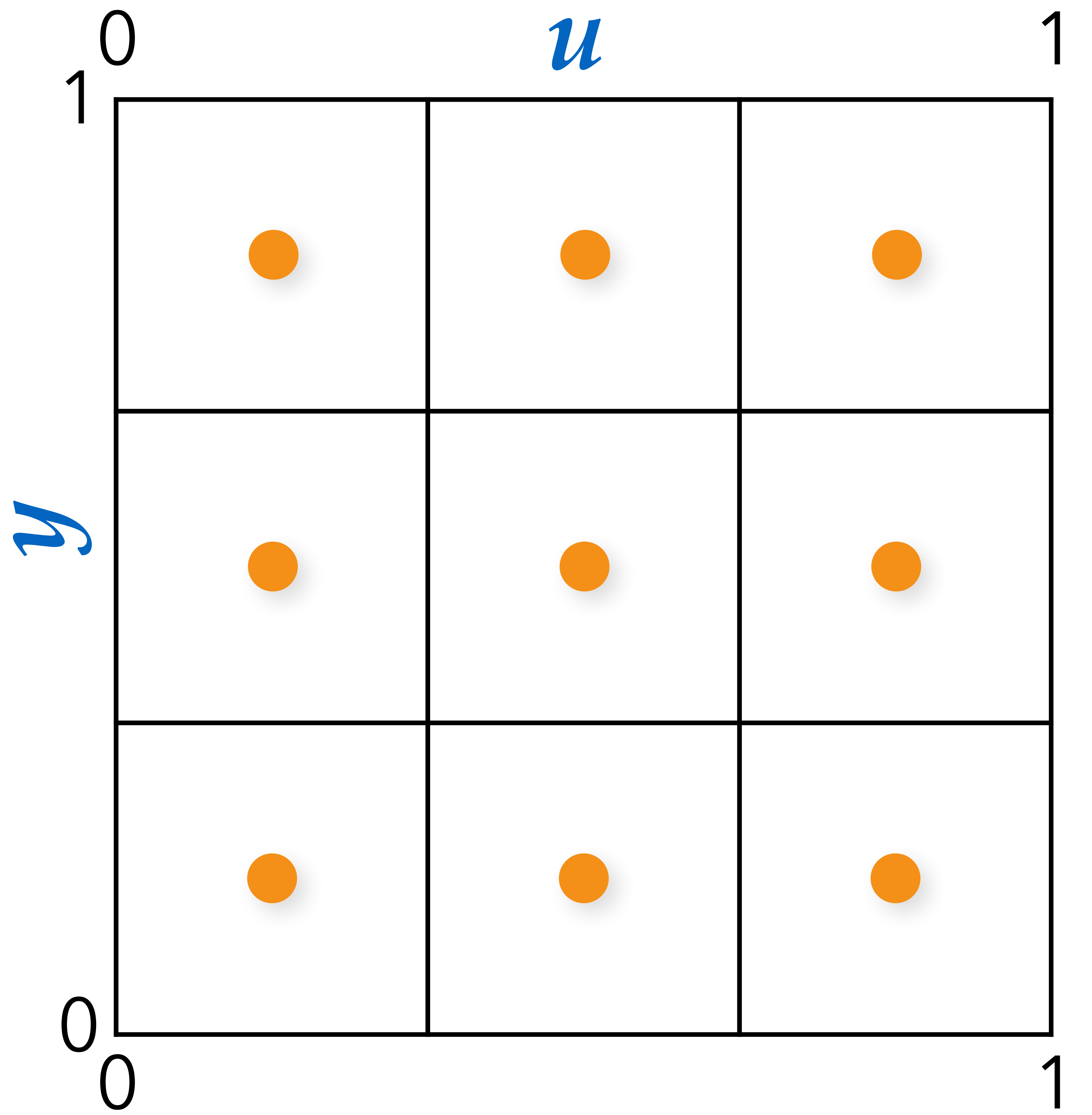
Monte Carlo using OAs

[Owen 92]

runs:	0	1	2	3	4	5	6	7	8
factors x:	0	0	0	1	1	1	2	2	2
y:	0	1	2	0	1	2	0	1	2
u:	0	1	2	1	2	0	2	0	1
v:	0	1	2	2	0	1	1	2	0

This OA encodes **nine 4D points**, which project to a regular **3** × **3** grid when plotting any pair of dimensions.

Rescale to [0,1) by dividing by s



Monte Carlo using OAs

[Owen 92]
OA-based jittered

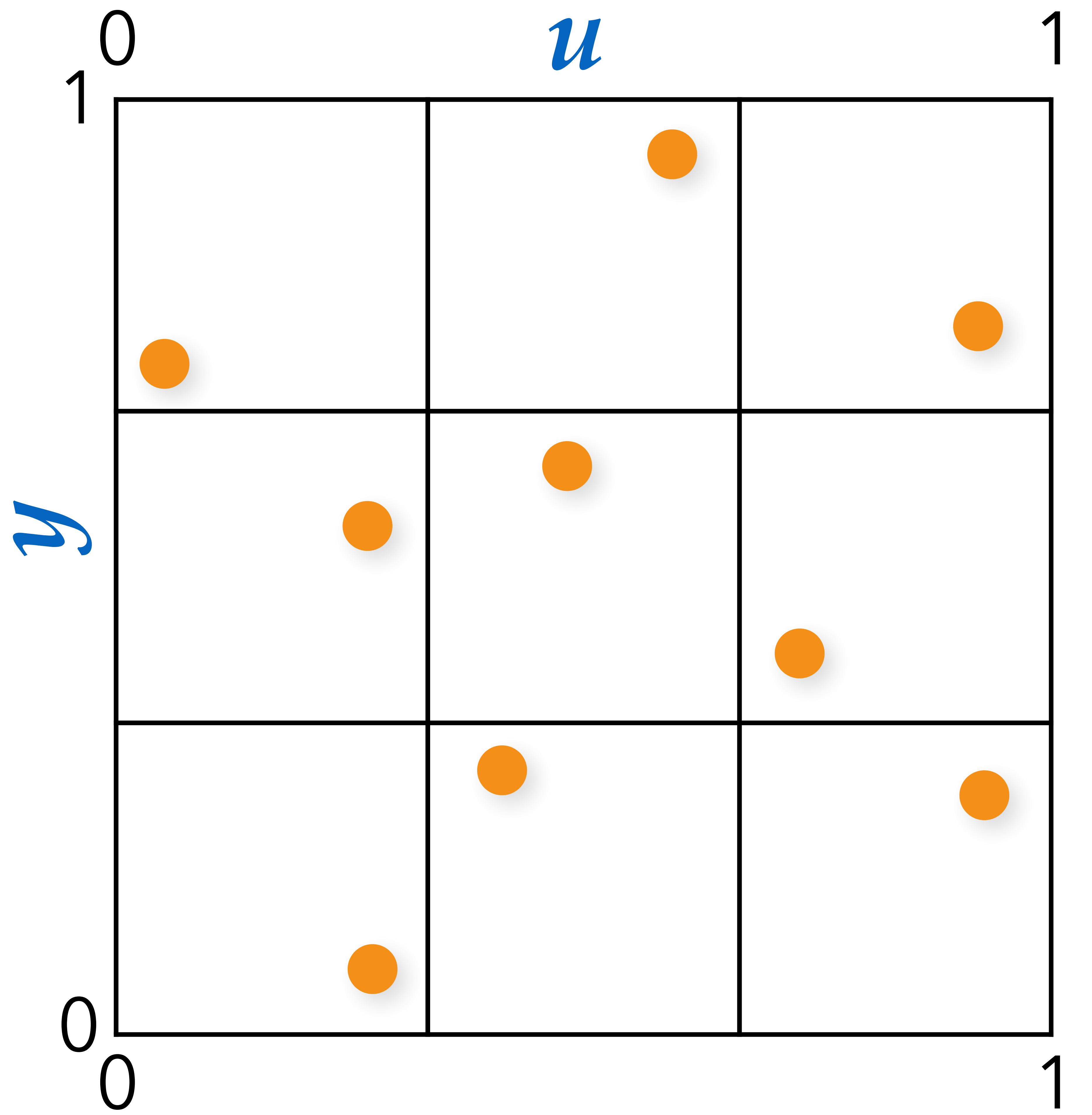
runs:	0	1	2	3	4	5	6	7	8
x:	0	0	0	1	1	1	2	2	2
y:	0	1	2	0	1	2	0	1	2
u:	0	1	2	1	2	0	2	0	1
v:	0	1	2	2	0	1	1	2	0

factors

This OA encodes **nine 4D points**, which project to a **jittered 3 × 3** grid when plotting any pair of dimensions.

✔ Add random offset per stratum

Jittered projections



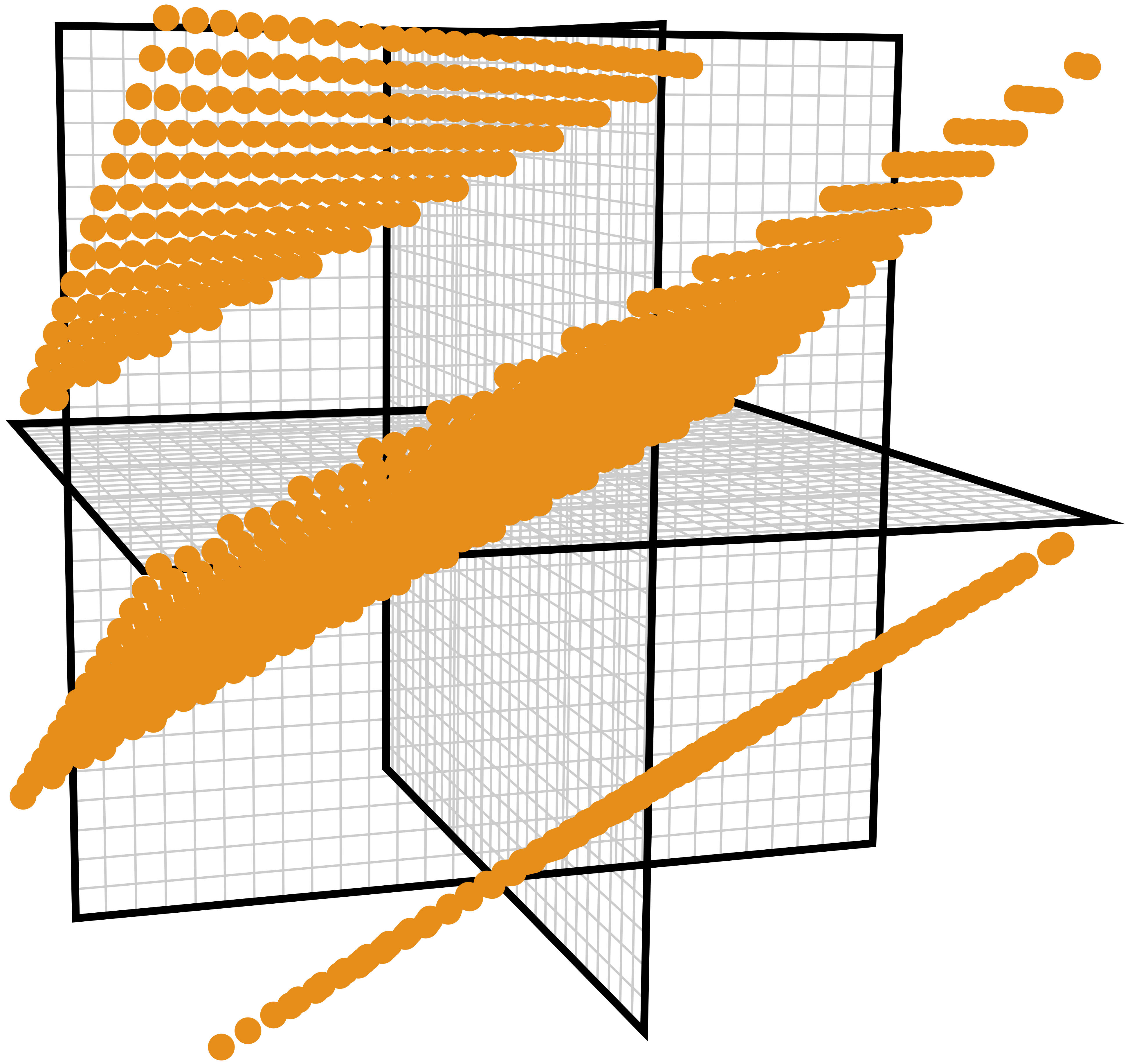
Monte Carlo using OAs

[Owen 92]
OA-based jittered

runs:	0	1	2	3	4	5	6	7	8
x:	0	0	0	1	1	1	2	2	2
y:	0	1	2	0	1	2	0	1	2
u:	0	1	2	1	2	0	2	0	1
v:	0	1	2	2	0	1	1	2	0

factors

Jittered projections



This OA encodes **nine 4D points**,

which project to a **jittered 3 × 3** grid when plotting any pair of dimensions.

✗ But not uniform in ***nD***

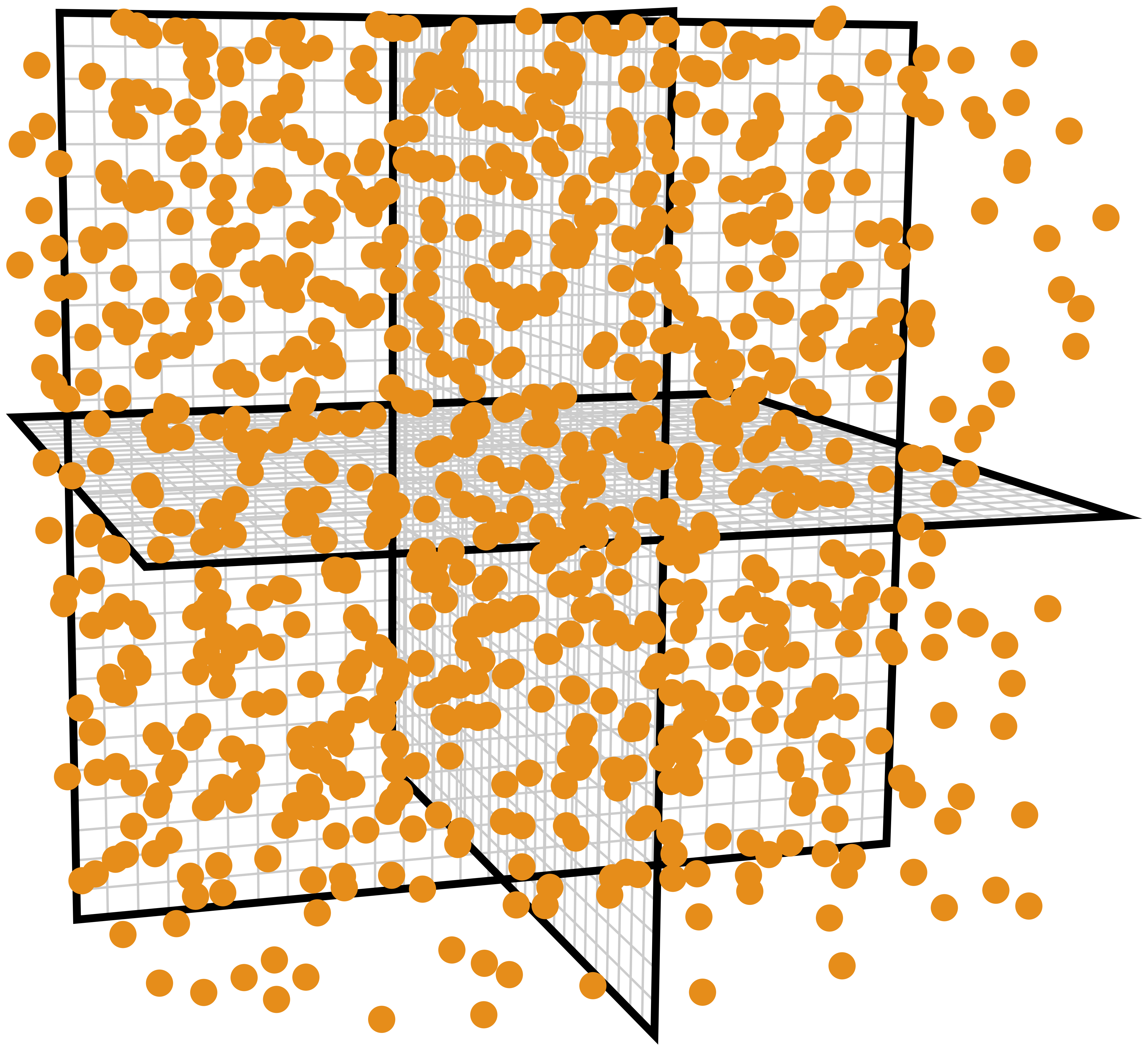
Monte Carlo using OAs

[Owen 92]
OA-based jittered

runs:	0	1	2	3	4	5	6	7	8
x:	1	2	0	2	1	2	1	0	1
y:	2	0	0	1	2	1	2	1	0
u:	1	2	1	0	0	1	2	2	0
v:	2	1	0	1	0	2	0	1	2

(factors)

Jittered projections



This OA encodes **nine 4D points**,

which project to a **jittered 3 × 3** grid when plotting any pair of dimensions.

✔ Permute levels in each dimension

Monte Carlo using OAs

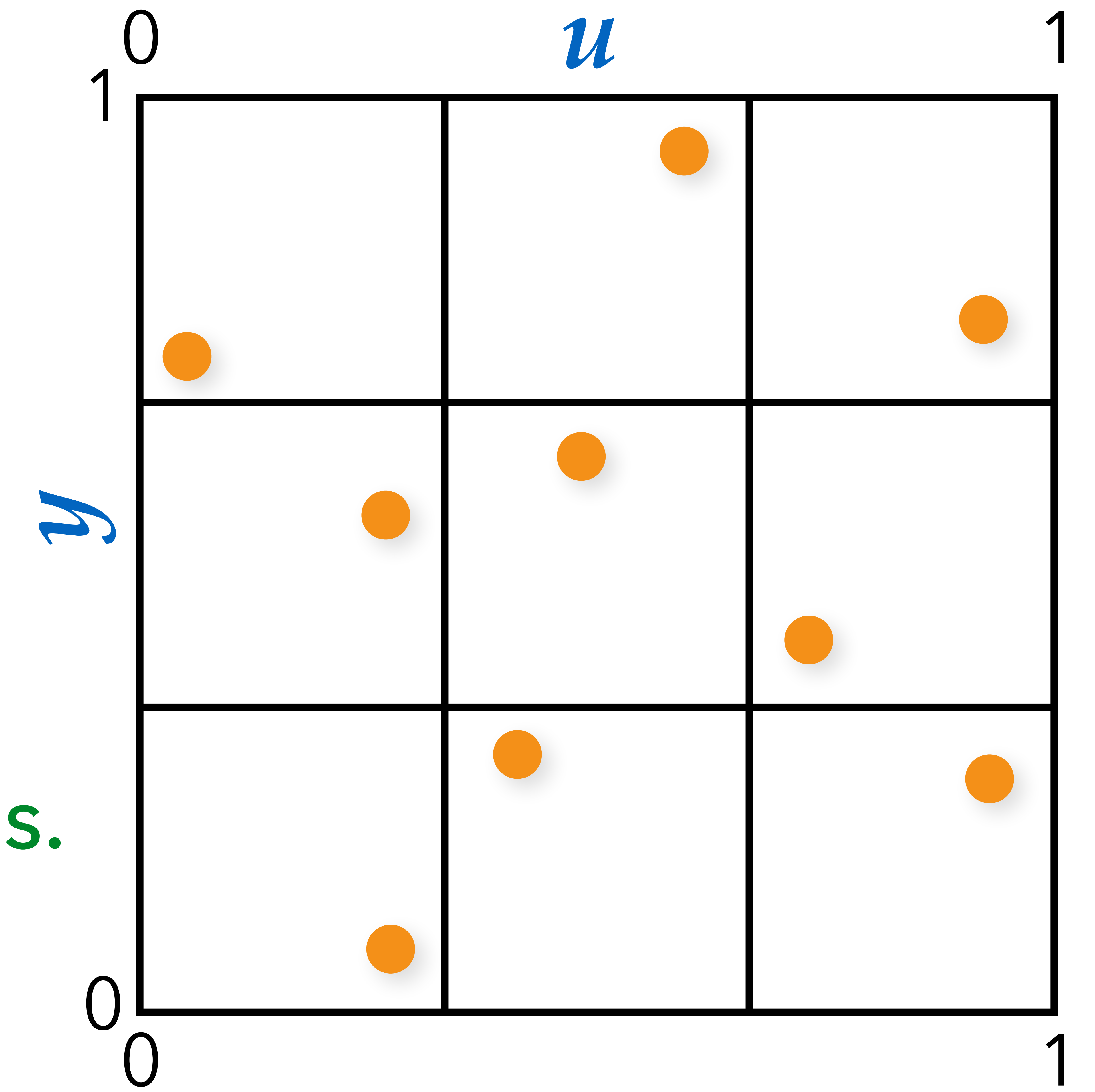
[Tang 93]
OA-based Latin hypercubes

runs:	0	1	2	3	4	5	6	7	8
factors x:	1	2	0	2	1	2	1	0	1
y:	2	0	0	1	2	1	2	1	0
u:	1	2	1	0	0	1	2	2	0
v:	2	1	0	1	0	2	0	1	2

This OA encodes **nine 4D points**,

which project to a **jittered 3 × 3**
grid when plotting any pair of dimensions.

jittered projections

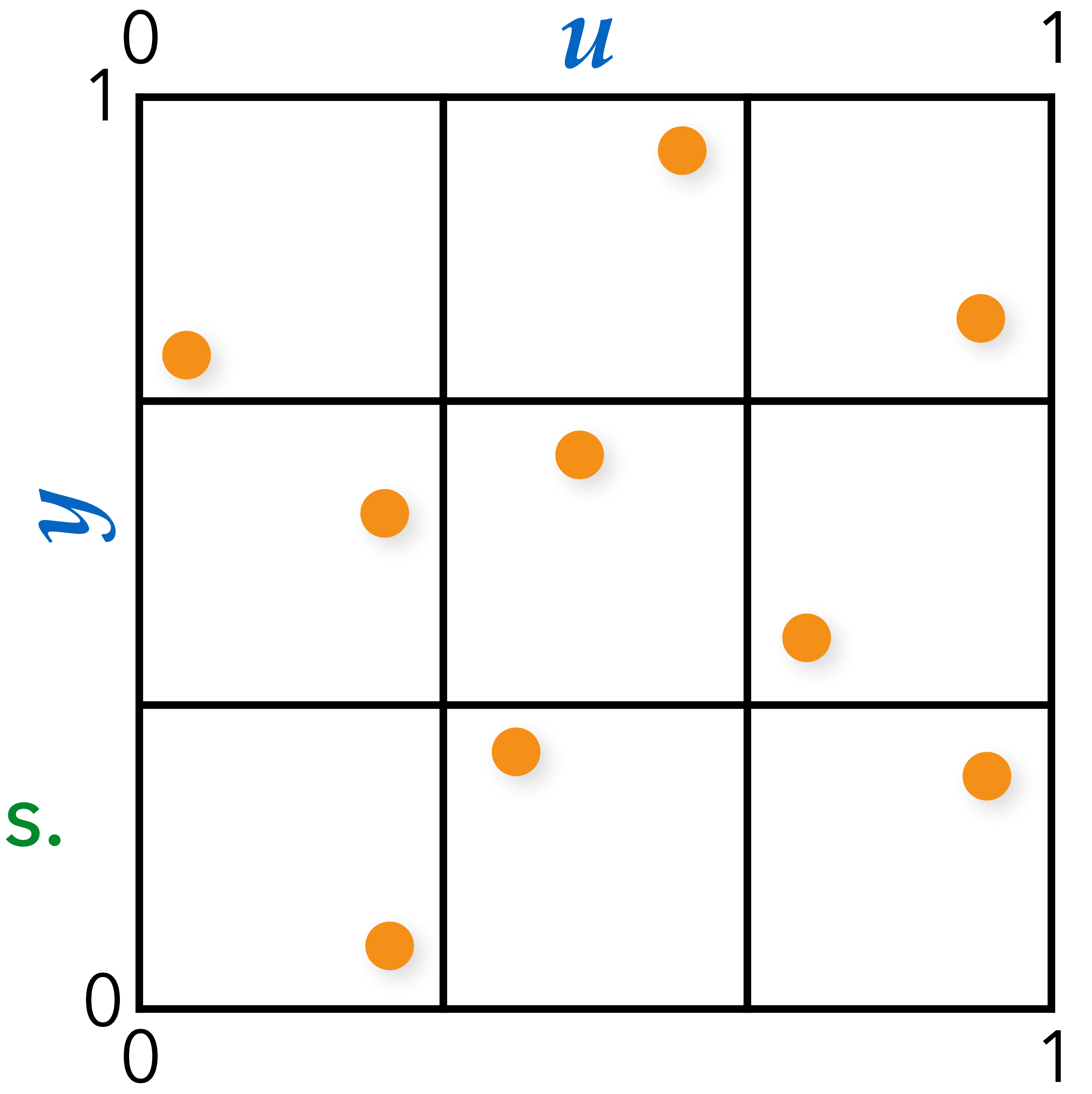


Monte Carlo using OAs

[Tang 93]
OA-based Latin hypercubes

	runs:	0	1	2	3	4	5	6	7	8
factors	x:	1	2	0	2	1	2	1	0	1
	y:	2	0	0	1	2	1	2	1	0
	u:	1	2	1	0	0	1	2	2	0
	v:	2	1	0	1	0	2	0	1	2

jittered projections



This OA encodes **nine 4D points**,

which project to a **jittered 3 × 3**
grid when plotting any pair of dimensions.

✔ Arrange points to fall in sub-strata

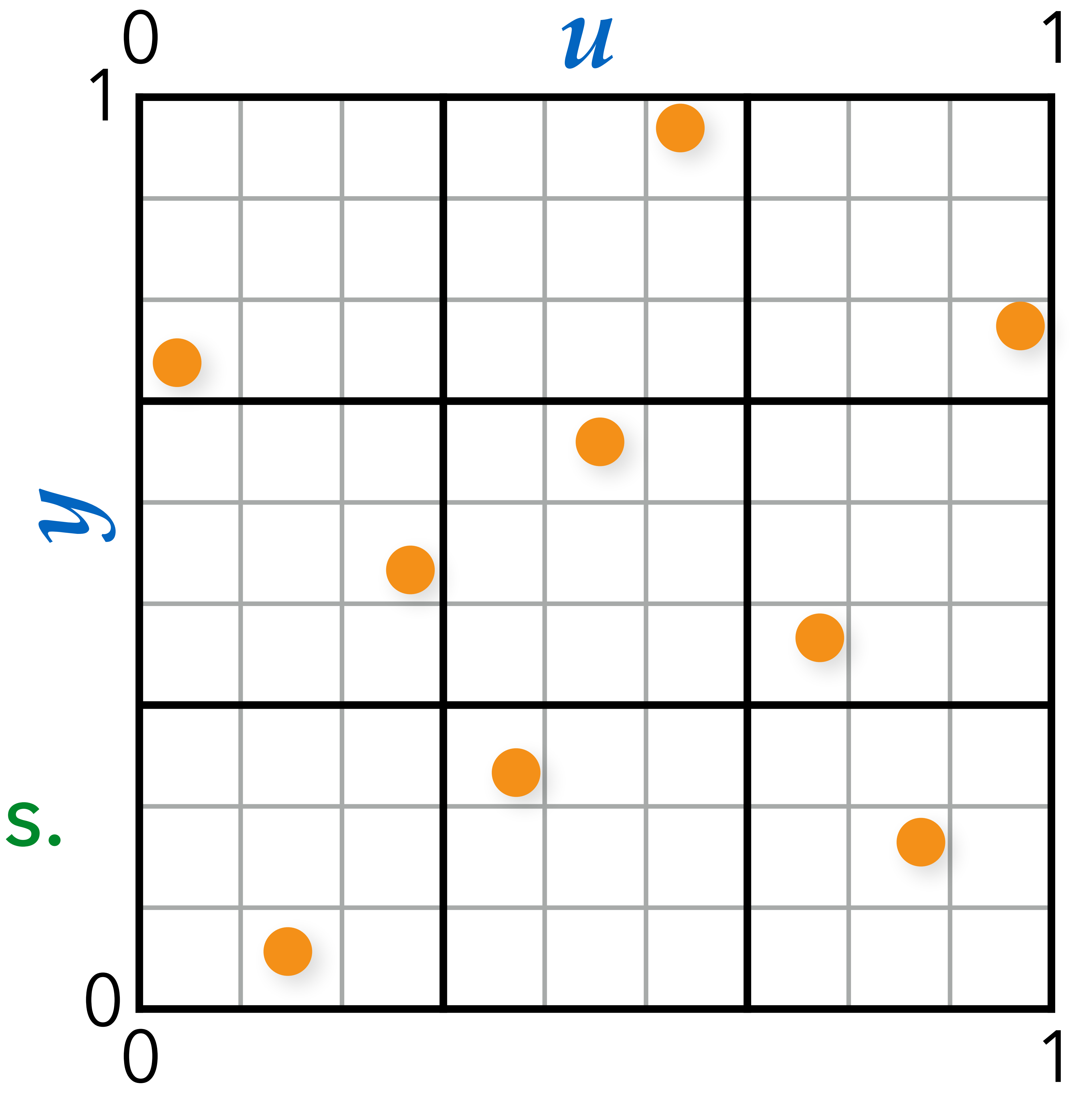
Monte Carlo using OAs

[Tang 93]
OA-based Latin hypercubes

runs:	0	1	2	3	4	5	6	7	8
x:	1	2	0	2	1	2	1	0	1
y:	2	0	0	1	2	1	2	1	0
u:	1	2	1	0	0	1	2	2	0
v:	2	1	0	1	0	2	0	1	2

factors

Multi-jittered projections



This OA encodes **nine 4D points**, which project to a **multi-jittered 3 × 3** grid when plotting any pair of dimensions.

✔ Arrange points to fall in sub-strata

OK, but how do we construct these?

OK, but how do we construct these?

Strength $t = 1$ OAs:

- Trivial to construct:

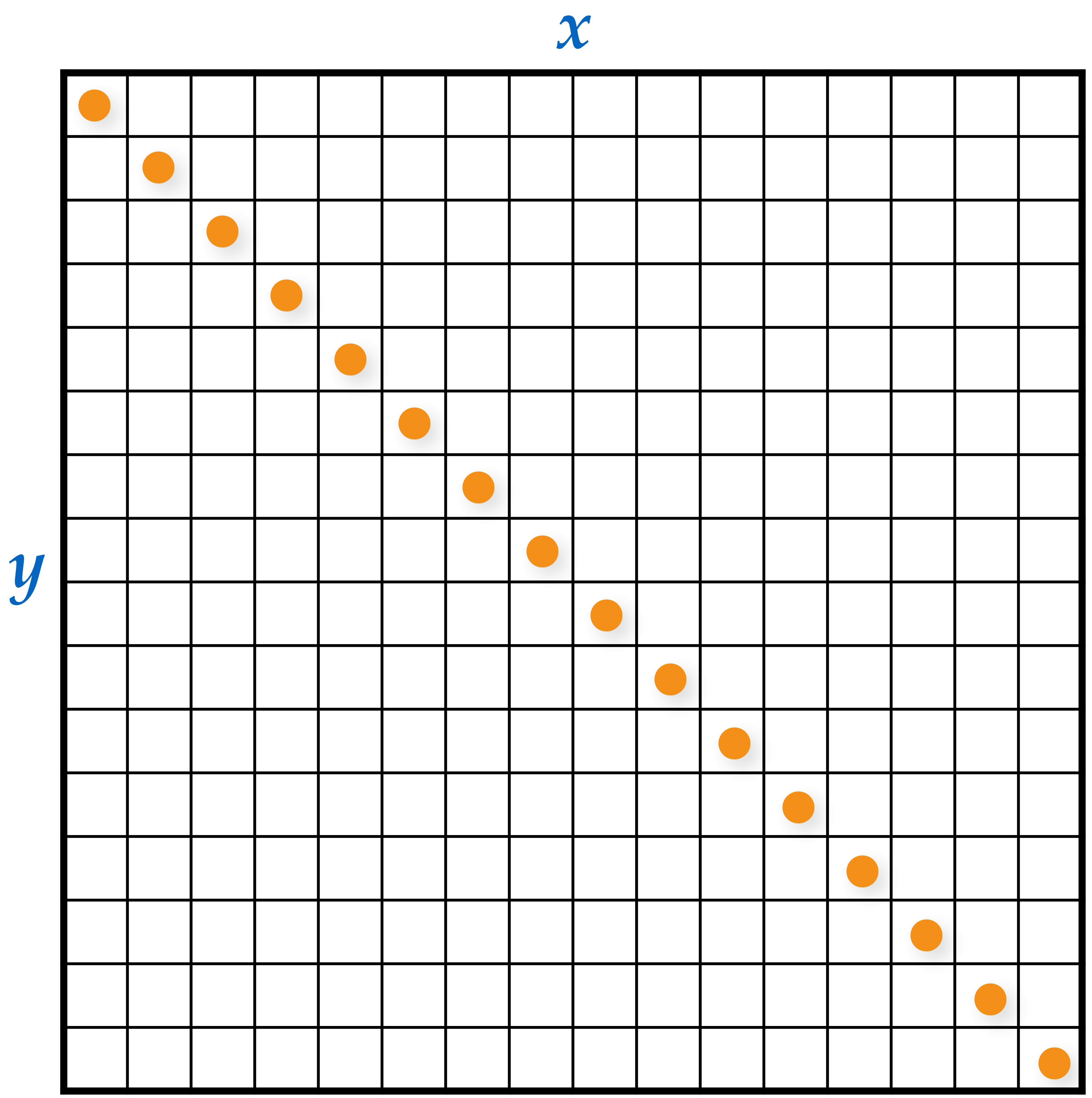
	runs:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
factors	x:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	y:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	u:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	v:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

OK, but how do we construct these?

Strength $t = 1$ OAs:

- Trivial to construct:

runs:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
factors x:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
y:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
u:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
v:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



initial OA

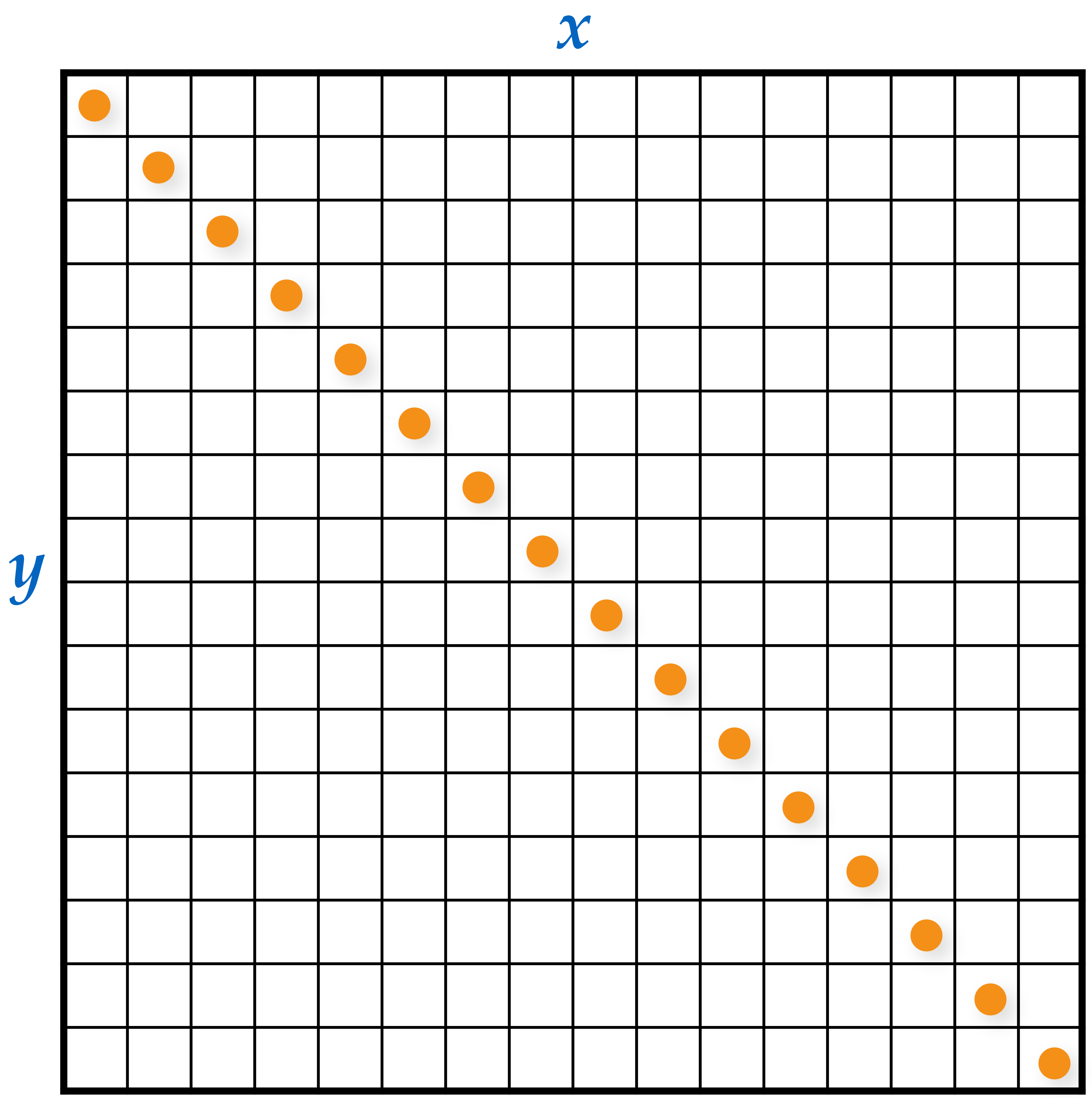
OK, but how do we construct these?

Strength $t = 1$ OAs:

- Trivial to construct:

	runs:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
factors	*x:*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	y:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	u:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	v:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- Stratify all 1D projections



initial OA

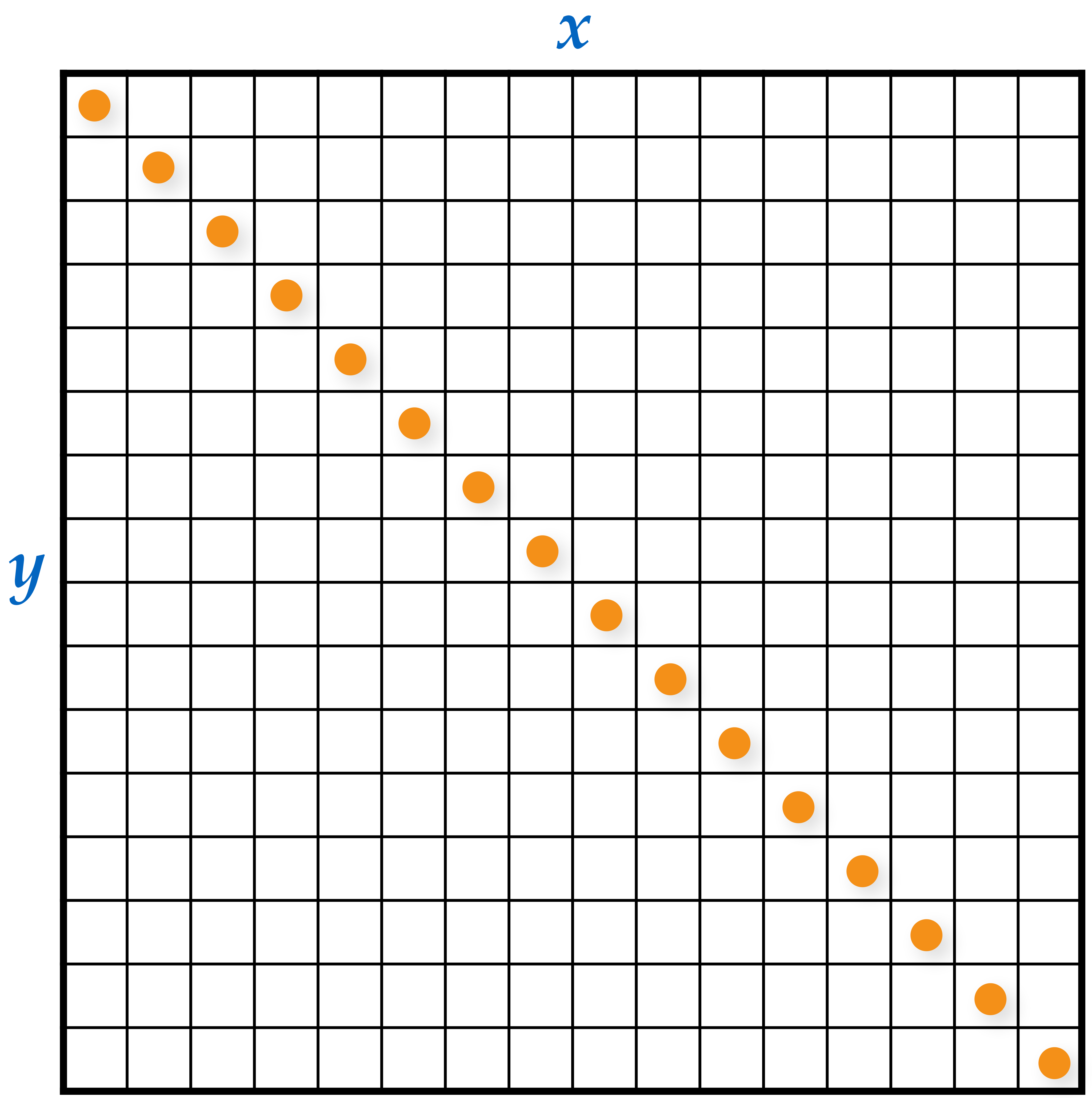
OK, but how do we construct these?

Strength $t = 1$ OAs:

- Trivial to construct:

	runs:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
factors	x:	12	8	1	14	2	10	0	5	4	11	3	13	2	15	7	9
	y:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	u:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	v:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- Stratify all 1D projections
- Permute levels



permute x-levels

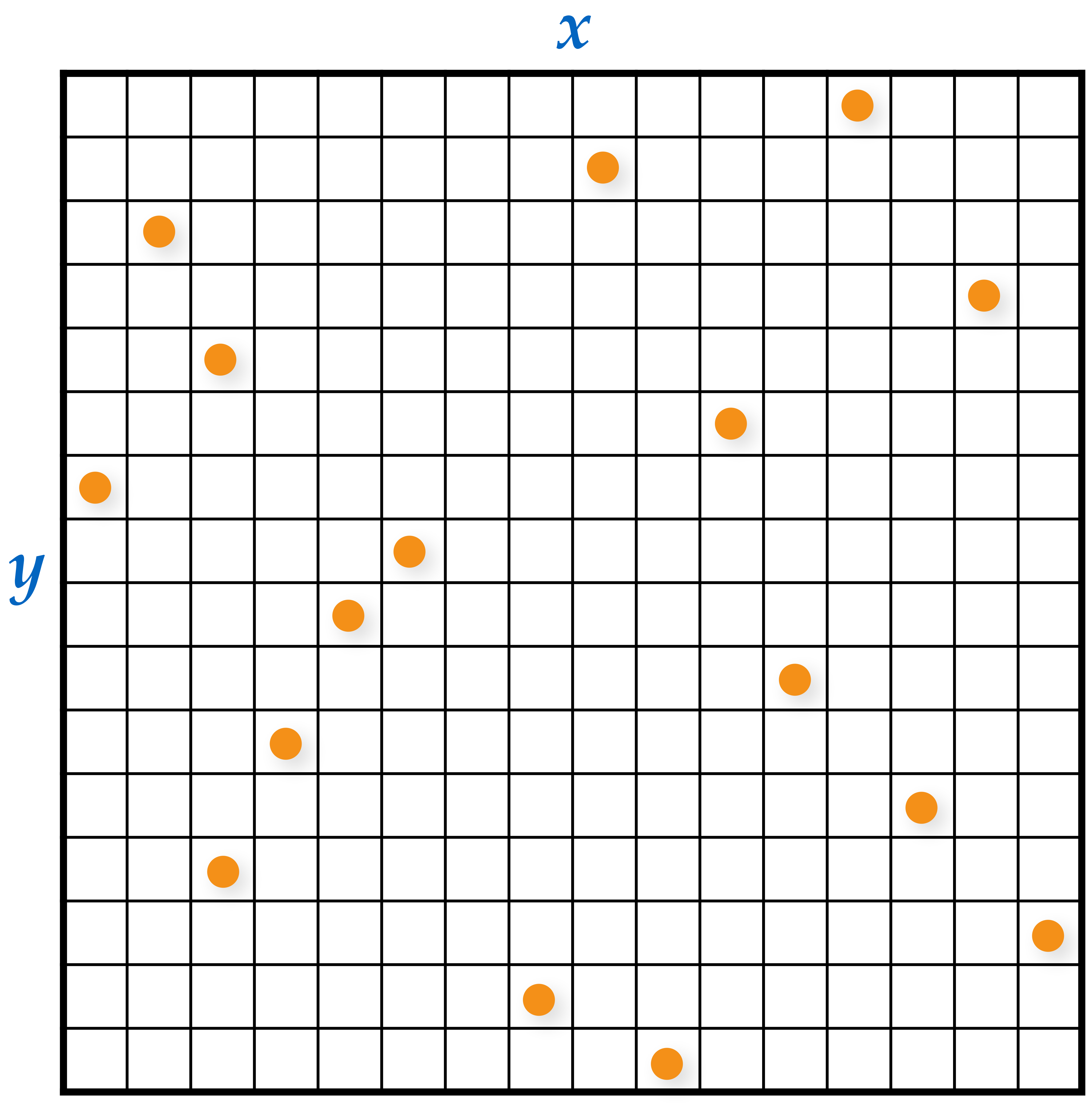
OK, but how do we construct these?

Strength $t = 1$ OAs:

- Trivial to construct:

	runs:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
factors	x:	12	8	1	14	2	10	0	5	4	11	3	13	2	15	7	9
	y:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	u:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	v:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- Stratify all 1D projections
- Permute levels



permute x-levels

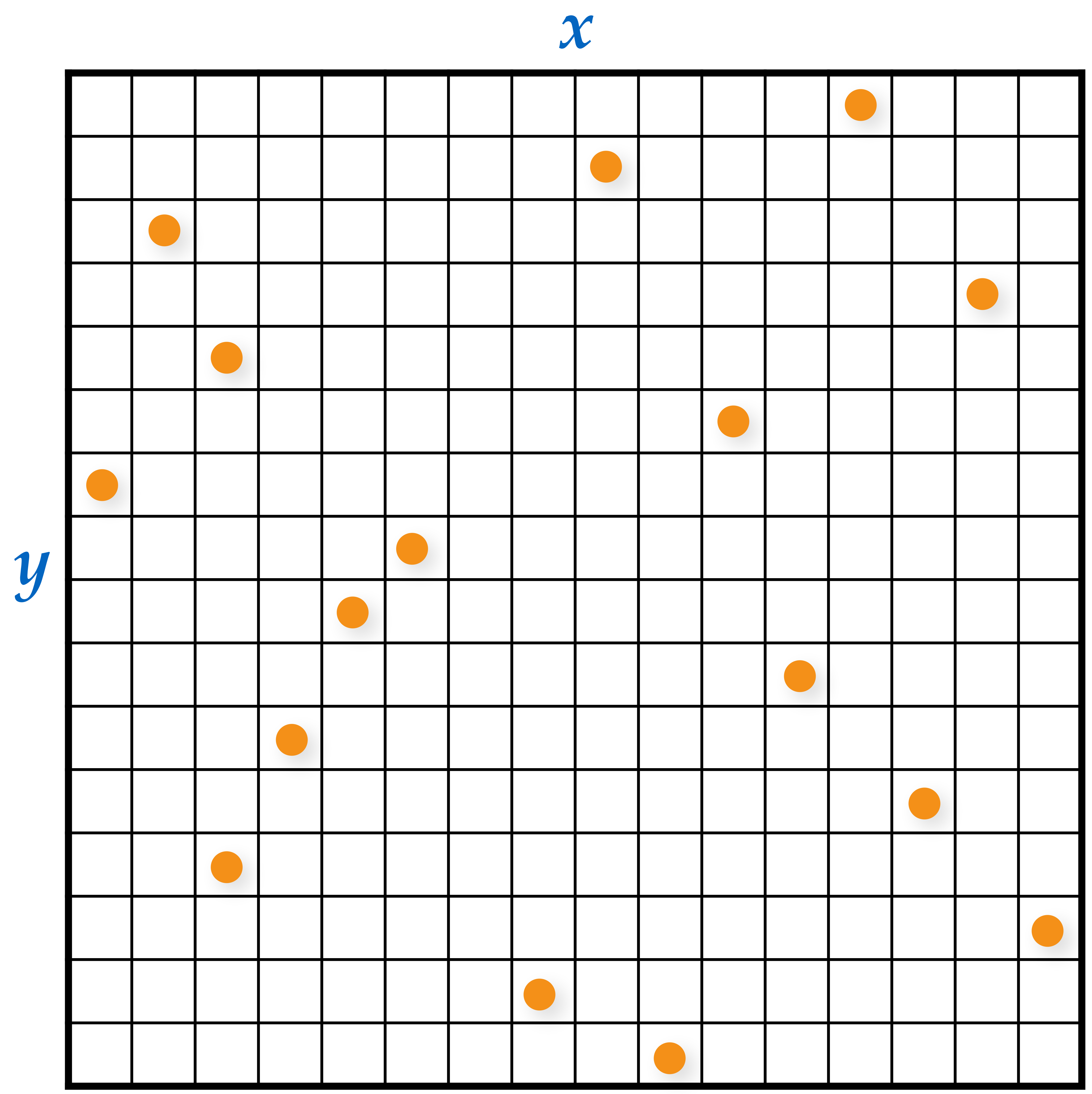
OK, but how do we construct these?

Strength $t = 1$ OAs:

- Trivial to construct:

	runs:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
factors	x:	12	8	1	14	2	10	0	5	4	11	3	13	2	15	7	9
	y:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	u:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	v:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

- Stratify all 1D projections
- Permute levels
  - Latin hypercube sampling (LHS)



Uniformly distributed in xy

OK, but how do we construct these?

What about $t \geq 2$?

- Generalization of LHS

OK, but how do we construct these?

What about $t \geq 2$?

- Generalization of LHS
- *Proofs*: for what values of N, s, d, t does an OA exist?

OK, but how do we construct these?

What about $t \geq 2$?

- Generalization of LHS
- *Proofs*: for what values of N, s, d, t does an OA exist?
- but little emphasis on constructing them ***quickly***

Contributions

Import/enhance 2 existing, and introduce 1 novel method

Contributions

Import/enhance 2 existing, and introduce 1 novel method

- make them ***fast***

Contributions

Import/enhance 2 existing, and introduce 1 novel method

- make them ***fast***
- generate samples and dimensions ***on-demand***

Contributions

Import/enhance 2 existing, and introduce 1 novel method

- make them ***fast***
- generate samples and dimensions ***on-demand***
- no need to compute entire array; ***no precomputation***

Constructions

Bose [1938] ($t = 2$):

Constructions

Bose [1938] ($t = 2$):

- stratifies all 2D projections [Owe92],

Constructions

Bose [1938] ($t = 2$):

- stratifies all 2D projections [Owe92],
- 1D projections [Tan93], and

Constructions

Bose [1938] ($t = 2$):

- stratifies all 2D projections [Owe92],
- 1D projections [Tan93], and
- correlated offsets [Ken13]

Constructions

Bose [1938] ($t = 2$):

- stratifies all 2D projections [Owe92],
- 1D projections [Tan93], and
- correlated offsets [Ken13]
- $N = s^2$ and $d = s+1$ where s is any prime

Constructions

Bose [1938] ($t = 2$):

- stratifies all 2D projections [Owe92],
- 1D projections [Tan93], and
- correlated offsets [Ken13]
- $N = s^2$ and $d = s+1$ where s is any prime

Bush [1952] ($t \geq 2$):

Constructions

Bose [1938] ($t = 2$):

- stratifies all 2D projections [Owe92],
- 1D projections [Tan93], and
- correlated offsets [Ken13]
- $N = s^2$ and $d = s+1$ where s is any prime

Bush [1952] ($t \geq 2$):

- stratifies all **tD** [Owe92] + 1D projections [Tan93]

Constructions

Bose [1938] ($t = 2$):

- stratifies all 2D projections [Owe92],
- 1D projections [Tan93], and
- correlated offsets [Ken13]
- $N = s^2$ and $d = s{+}1$ where s is any prime

Bush [1952] ($\boldsymbol{t \geq 2}$):

- stratifies all **tD** [Owe92] + 1D projections [Tan93]
- $N = \boldsymbol{s^t}$ and $\boldsymbol{d = s}$ where s is any prime

Constructions

Bose [1938] ($t = 2$):

- stratifies all 2D projections [Owe92],
- 1D projections [Tan93], and
- correlated offsets [Ken13]
- $N = s^2$ and $d = s{+}1$ where s is any prime

Bush [1952] ($\boldsymbol{t \geq 2}$):

- stratifies all **tD** [Owe92] + 1D projections [Tan93]
- $N = \boldsymbol{s^t}$ and $\boldsymbol{d = s}$ where s is any prime

High dimensional CMJ ($t \geq 2$):

Constructions

Bose [1938] ($t = 2$):

- stratifies all 2D projections [Owe92],
- 1D projections [Tan93], and
- correlated offsets [Ken13]
- $N = s^2$ and $d = s{+}1$ where s is any prime

Bush [1952] ($\boldsymbol{t \geq 2}$):

- stratifies all **tD** [Owe92] + 1D projections [Tan93]
- $N = \boldsymbol{s^t}$ and $\boldsymbol{d = s}$ where s is any prime

High dimensional CMJ ($t \geq 2$):

- stratifies all tD + 1D projections + correlated [Ken13]

Constructions

Bose [1938] ($t = 2$):

- stratifies all 2D projections [Owe92],
- 1D projections [Tan93], and
- correlated offsets [Ken13]
- $N = s^2$ and $d = s+1$ where s is any prime

Bush [1952] ($\boldsymbol{t \geq 2}$):

- stratifies all **tD** [Owe92] + 1D projections [Tan93]
- $N = \boldsymbol{s^t}$ and $\boldsymbol{d = s}$ where s is any prime

High dimensional CMJ ($t \geq 2$):

- stratifies all tD + 1D projections + correlated [Ken13]
- s is **any positive integer**; num samples $N = s^t$ and max dimension $\boldsymbol{d = t}$

Bose [1938] construction:

$$A_{i0} = \lfloor i/s \rfloor$$

$$A_{i1} = i \bmod s$$

$$A_{ij} = A_{i0} + (j-1)A_{i1} \bmod s$$

```c
float boseOA(int i,           // sample index
             int j,           // dimension (< s+1)
             int s,           // number of levels/strata
             int p) {         // pseudo-random permutation seed
    int Aij, Aik;
    int Ai0       = i / s;
    int Ai1       = i % s;
    if (j == 0) {
        Aij       = Ai0;
        Aik       = Ai1;
    } else if (j == 1) {
        Aij       = Ai1;
        Aik       = Ai0;
    } else {
        int k     = (j % 2) ? j-1 : j+1;
        Aij       = (Ai0 + (j-1) * Ai1) % s;
        Aik       = (Ai0 + (k-1) * Ai1) % s;
    }
    int stratum    = permute(Aij, s, p * j * 0x51633e2d);
    int subStratum = permute(Aik, s, p * j * 0x68bc21eb);
    float jitter   = randfloat(i,    p * j * 0x02e5be93);
    return (stratum + (subStratum + jitter) / s) / s;
}
```

Bose [1938] construction:

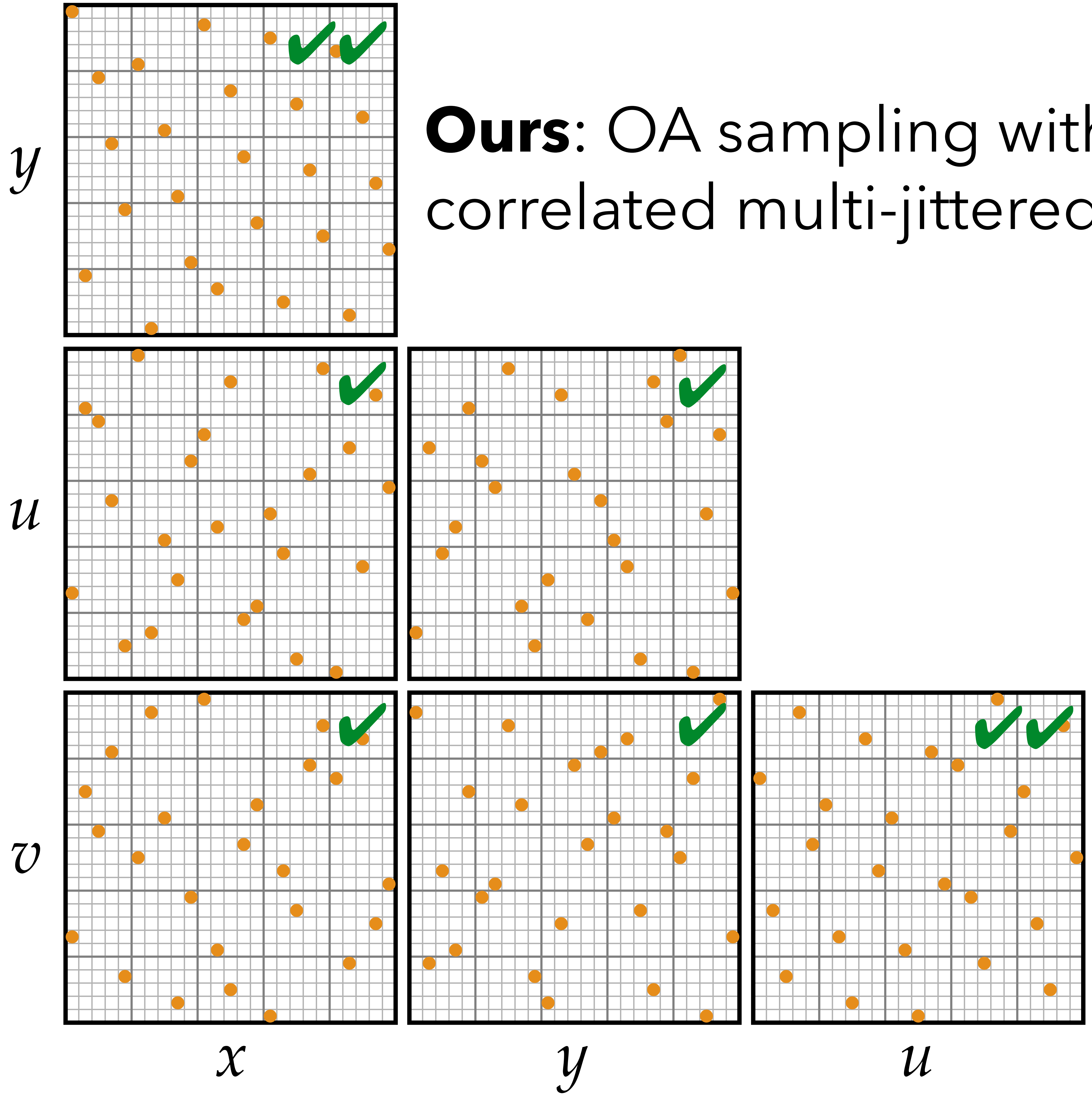
$$A_{i0} = \lfloor i/s \rfloor$$

$$A_{i1} = i \bmod s$$

$$A_{ij} = A_{i0} + (j-1)A_{i1} \bmod s$$

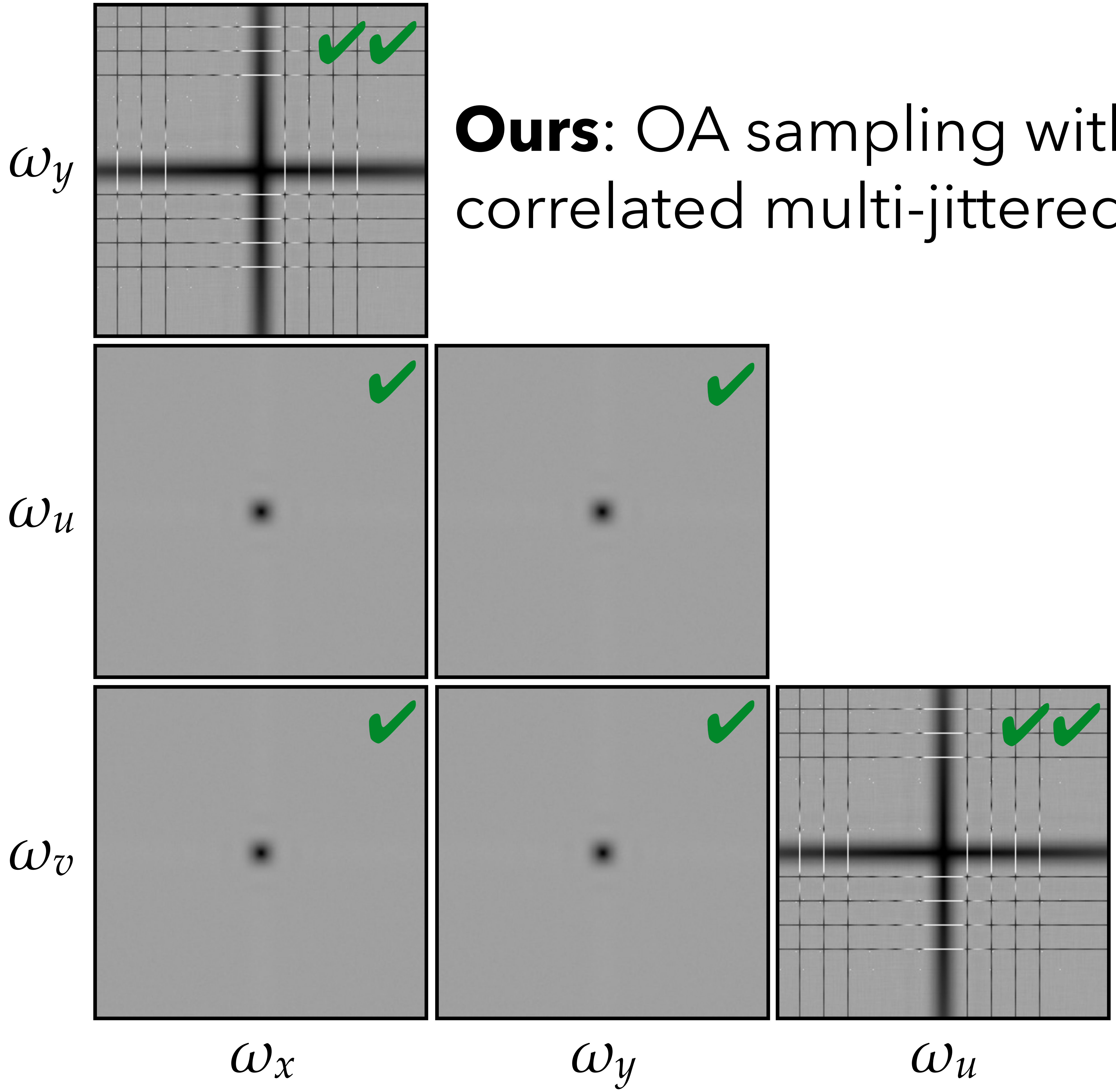
Analysis & Results

Power spectra validation



Ours: OA sampling with correlated multi-jittered offsets

Power spectra validation



Ours: OA sampling with correlated multi-jittered offsets

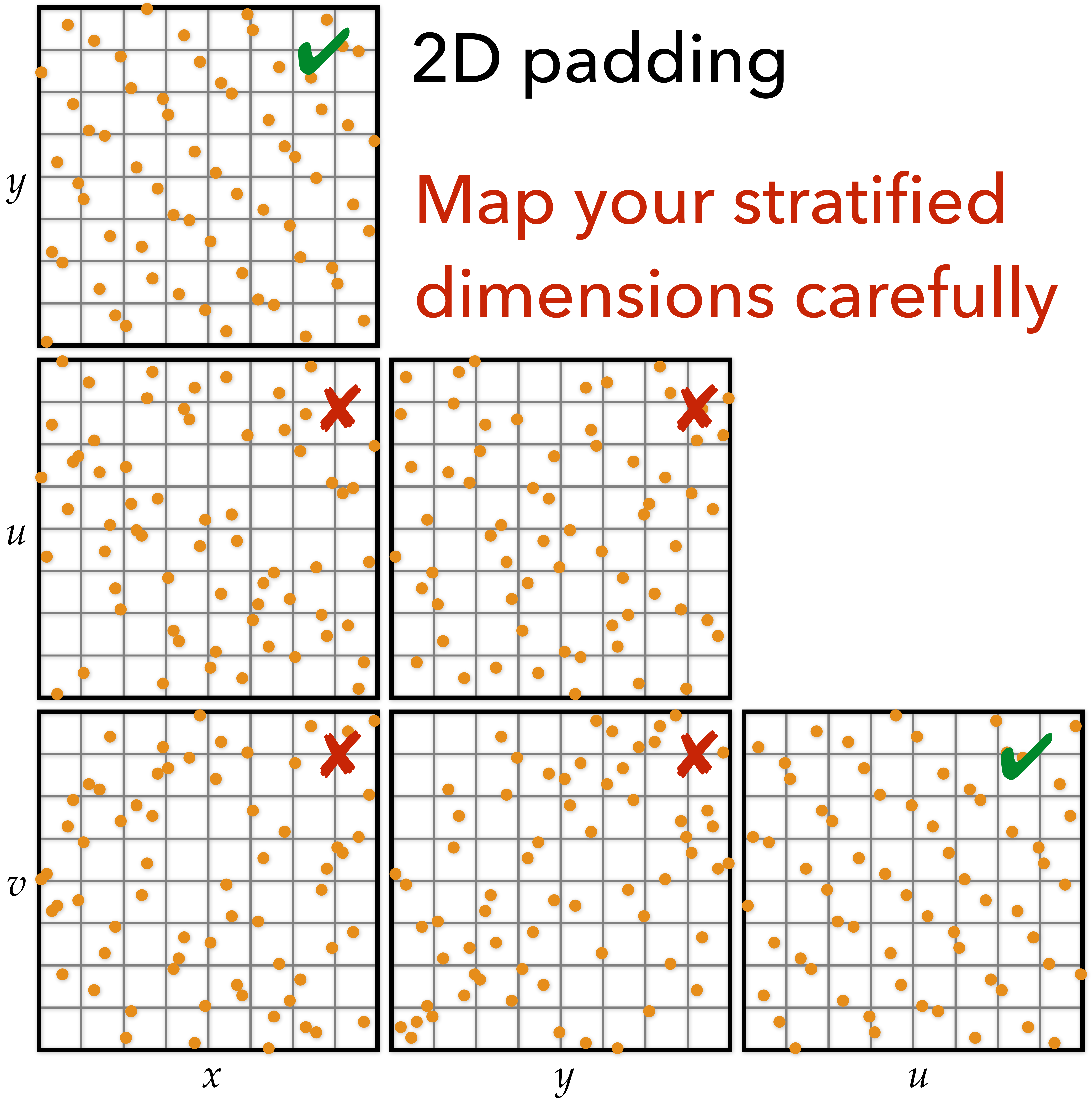
Power spectra validation

Ours: OA sampling with correlated multi-jittered offsets

Details in the paper

Details in the paper
1-additive $(f^D_{4,1+})$
2-additive $(f^D_{4,2+})$
2-multiplicative $(f^D_{4,2\times})$
4D-radial (f^D_4)
Gaussian $(D = \infty)$
Linear step $(D = 1)$
Binary step $(D = 0)$
Variance
Samples

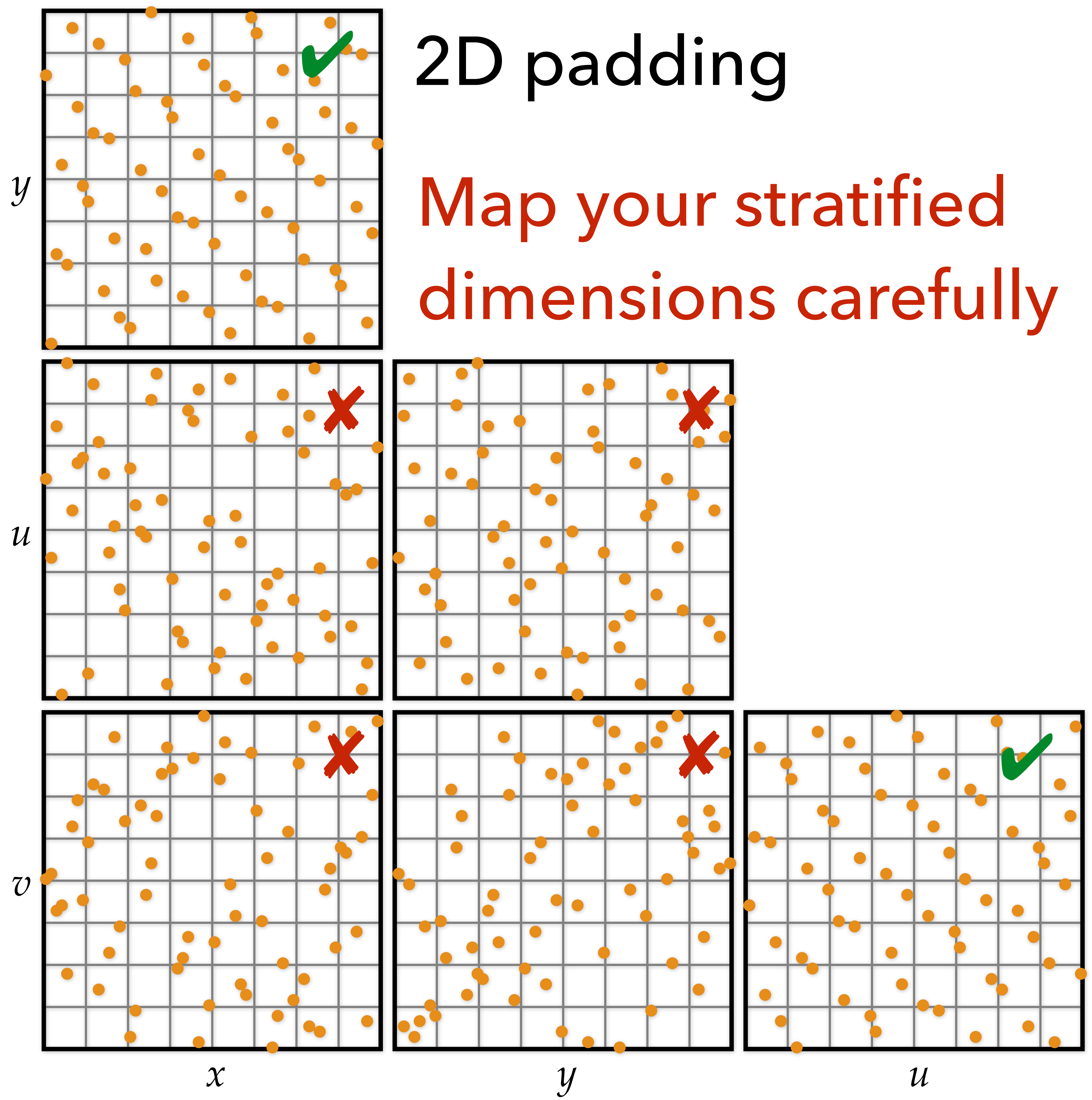
Which dimensions matter?

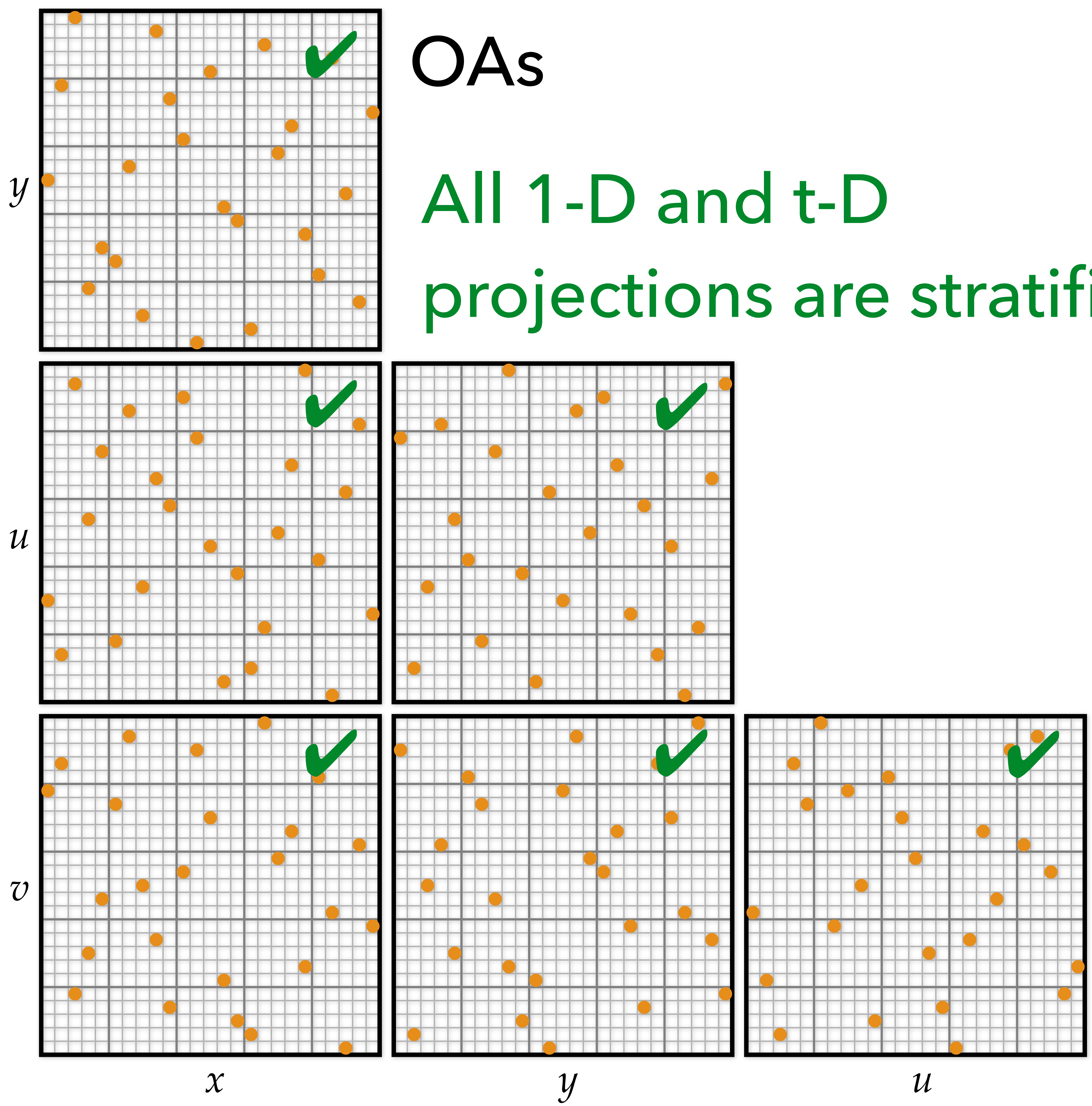


2D padding

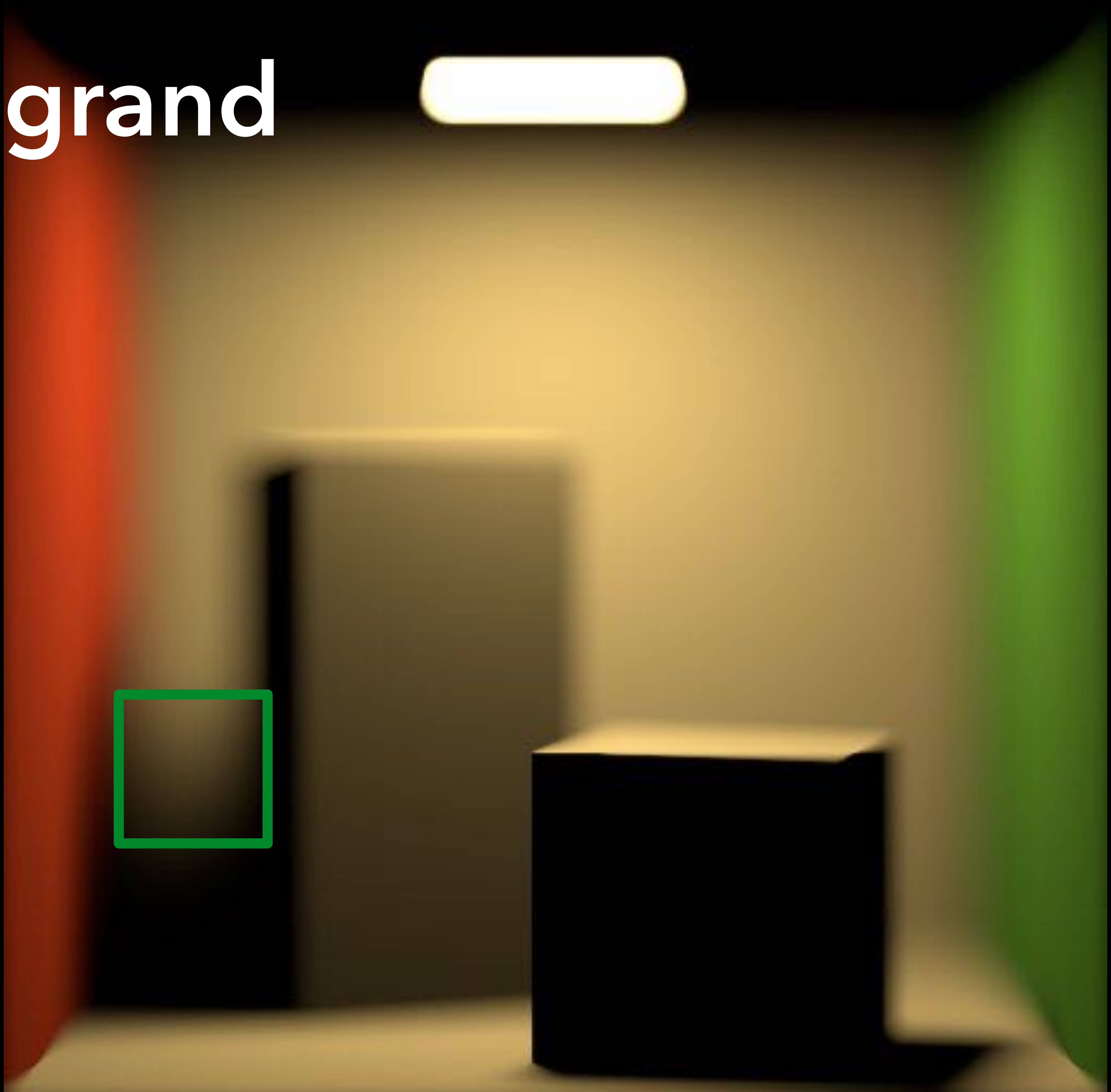
Map your stratified dimensions carefully

Which dimensions matter?





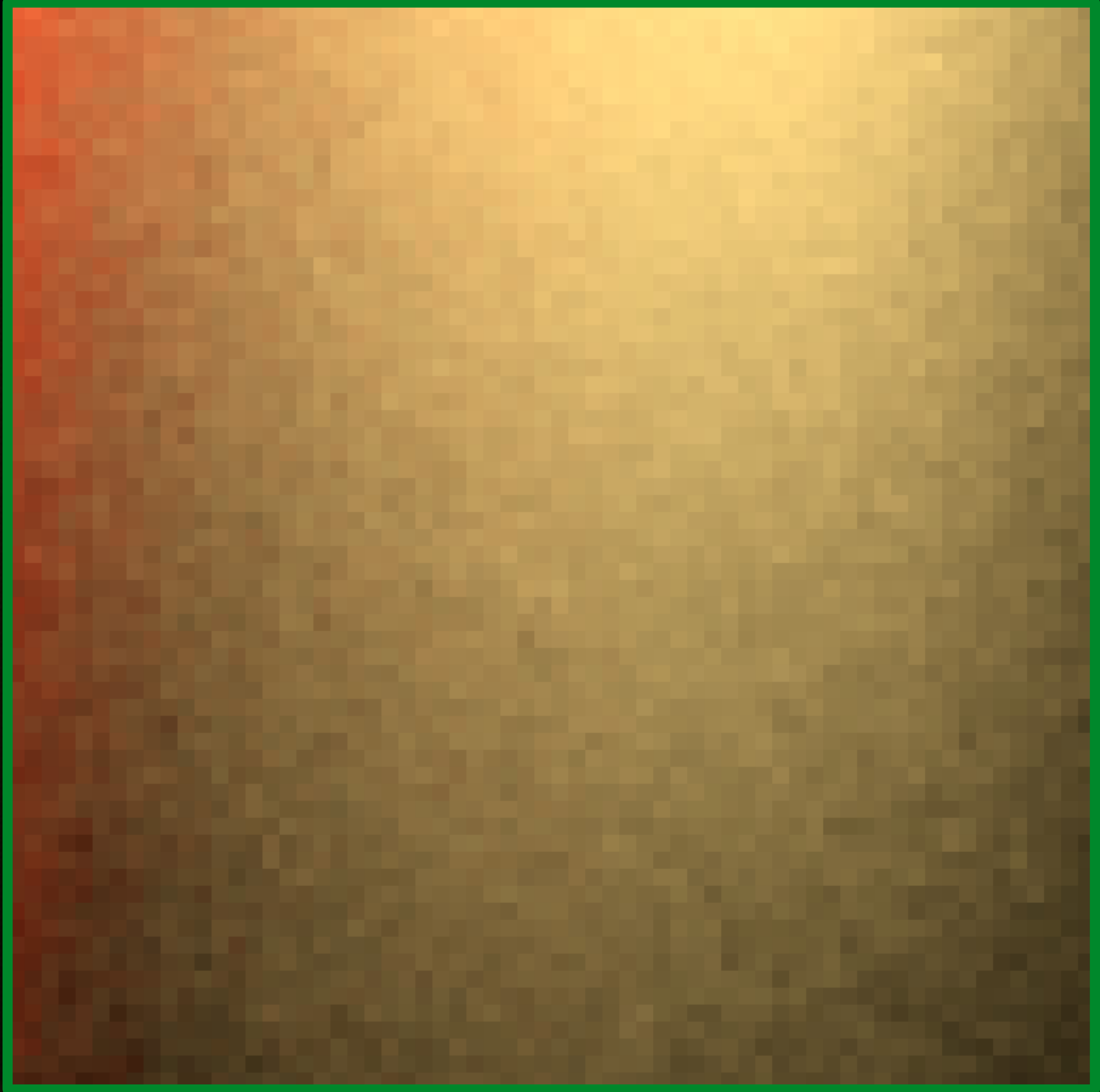
7D integrand



Random

121 spp

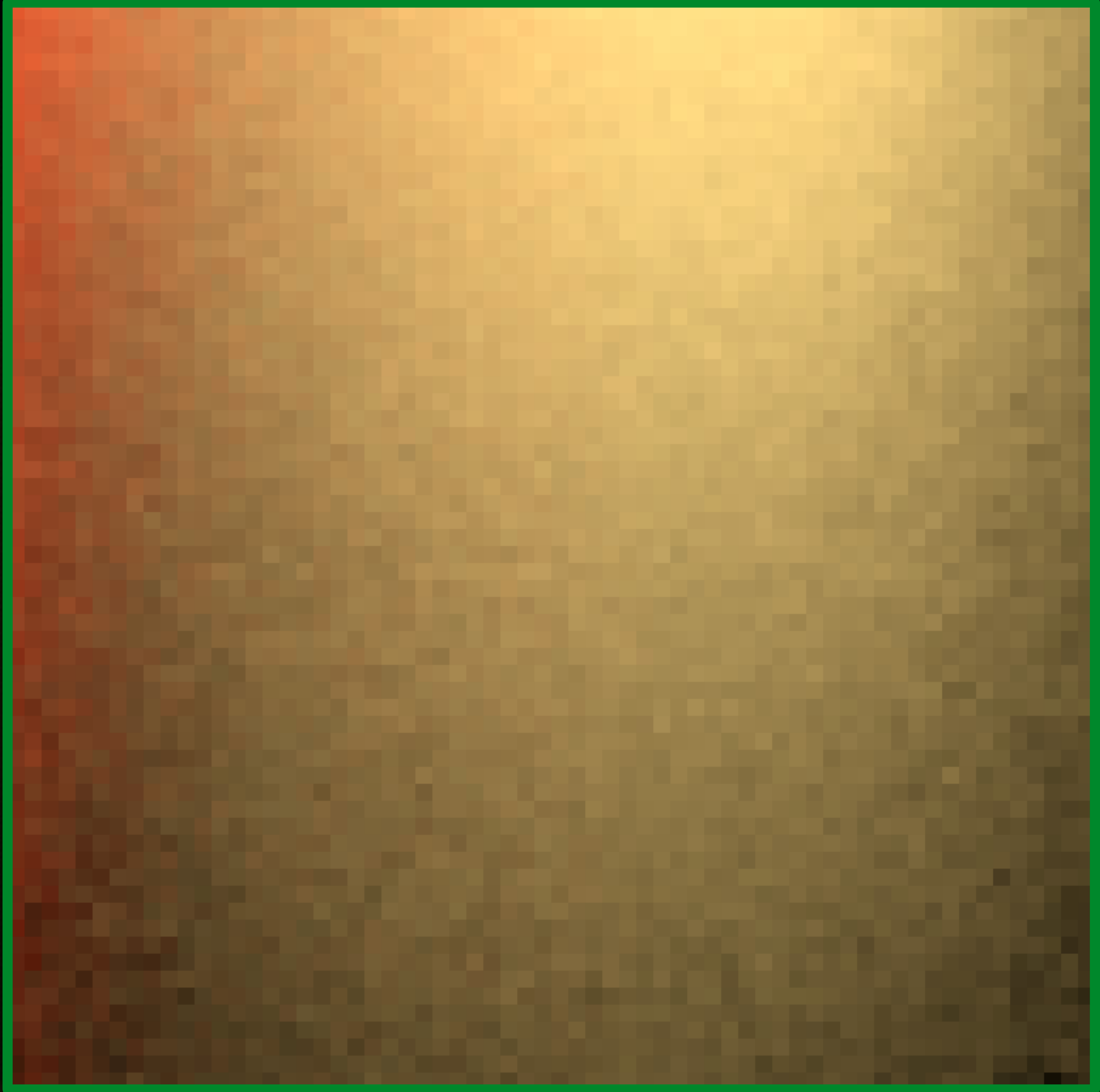
Sampler	Relative MSE	
	Full image	Crop
Random	1.481e-3	6.755e-4



Jittered2D (pad)

121 spp

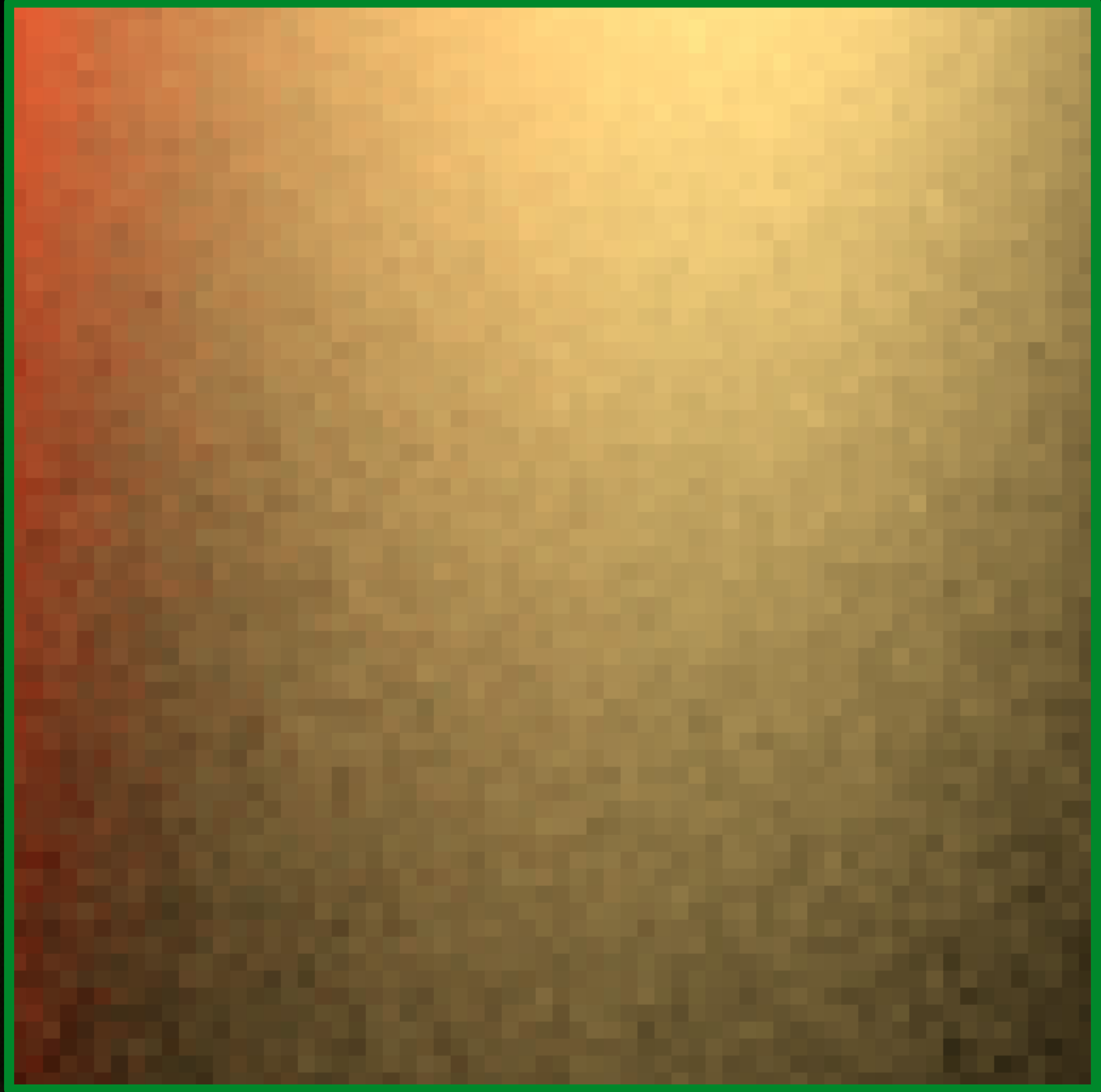
Sampler	Relative MSE	
	Full image	Crop
Random	1.481e-3	6.755e-4
Jittered2D (pad)	**1.036e-3**	**6.123e-4**



CMJ2D (pad)

121 spp

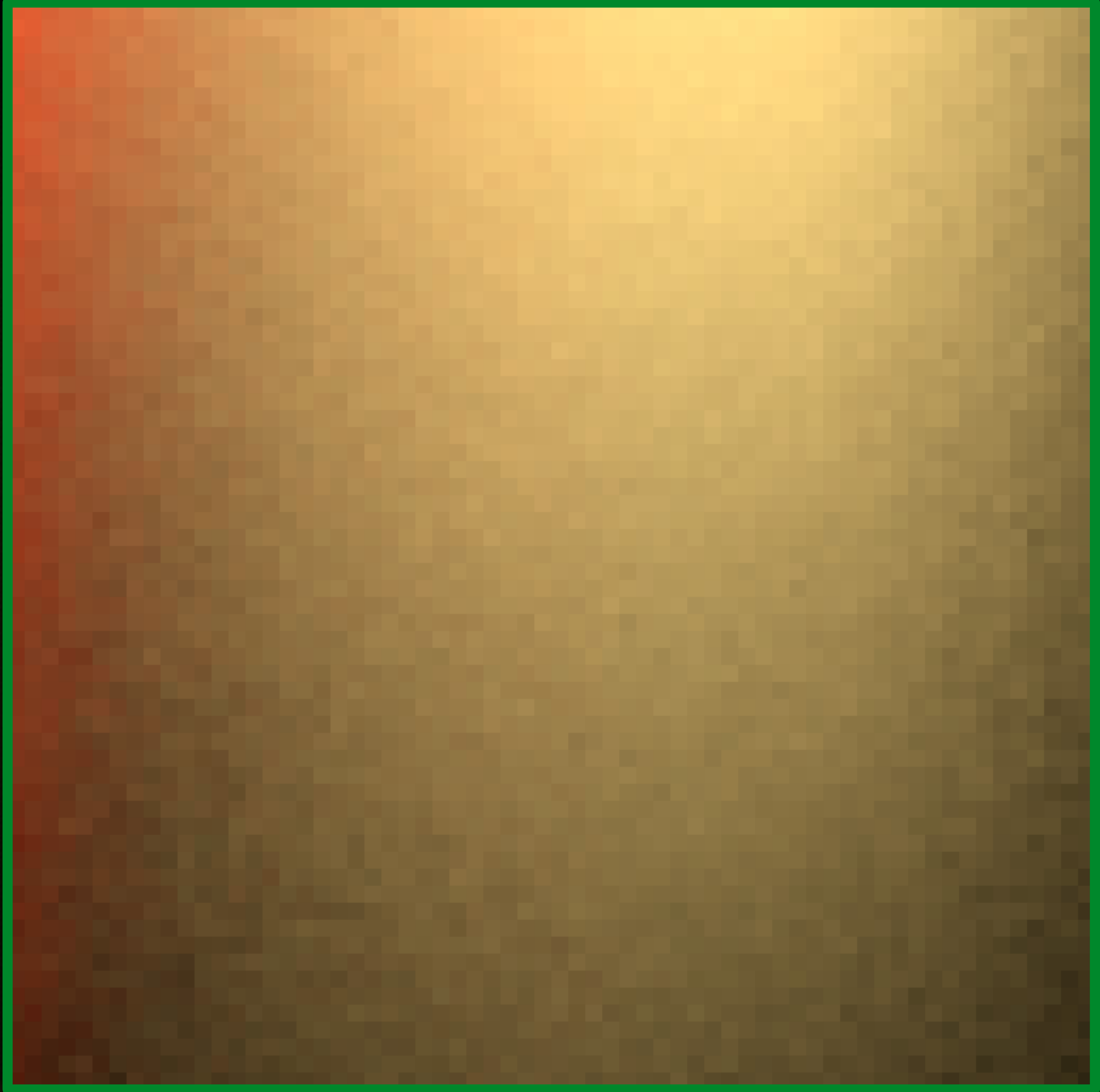
	Relative MSE	
Sampler	Full image	Crop
Random	1.481e-3	6.755e-4
Jittered2D (pad)	1.036e-3	**6.123e-4**
CMJ2D (pad)	**8.721e-4**	6.142e-4



(0,2)-seq (pad)

128 spp

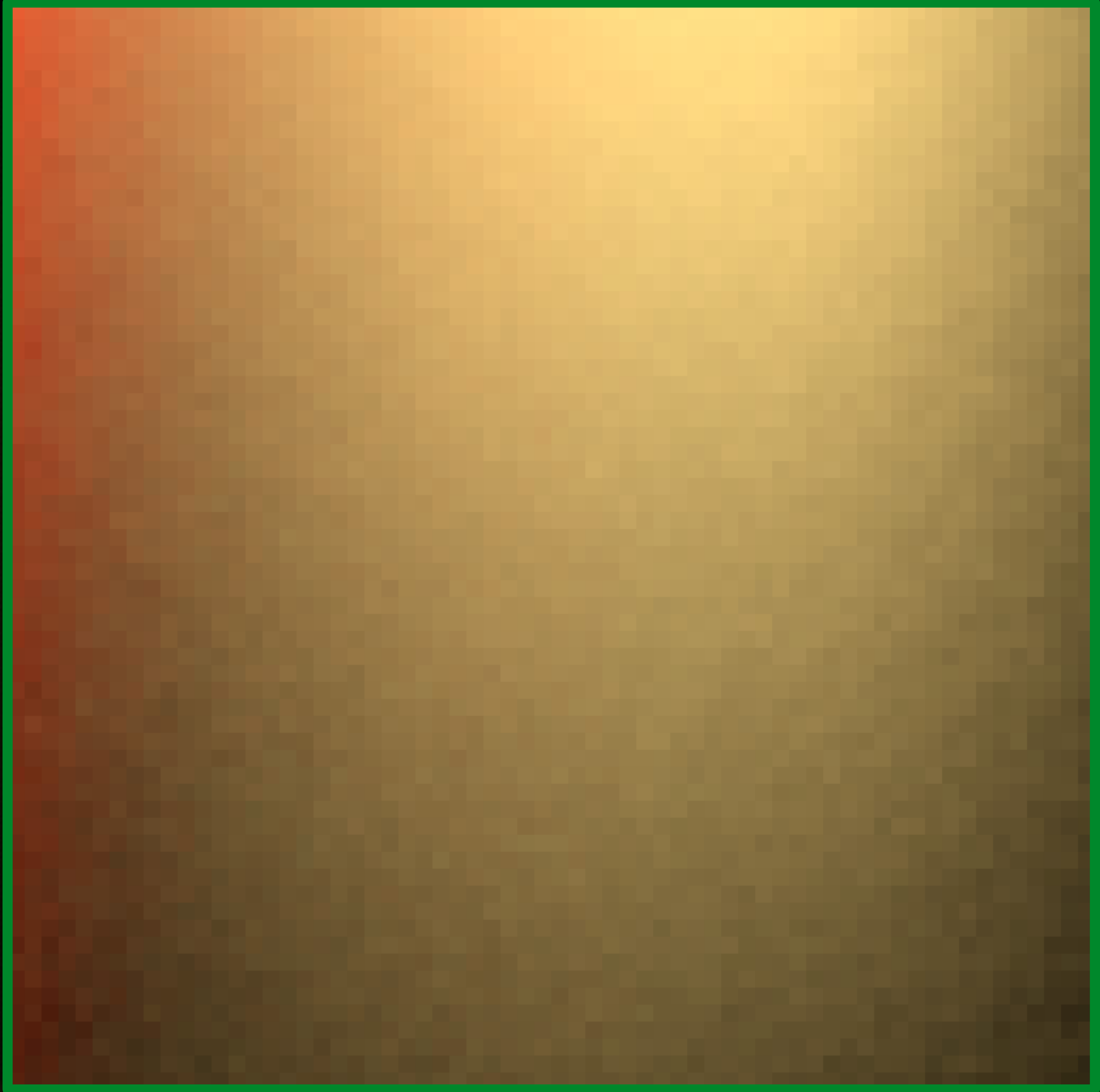
Sampler	Relative MSE	
	Full image	Crop
Random	1.481e-3	6.755e-4
Jittered2D (pad)	1.036e-3	6.123e-4
CMJ2D (pad)	8.721e-4	6.142e-4
(0,2)-seq. (pad)	**8.299e-4**	**2.825e-4**



Ours

121 spp

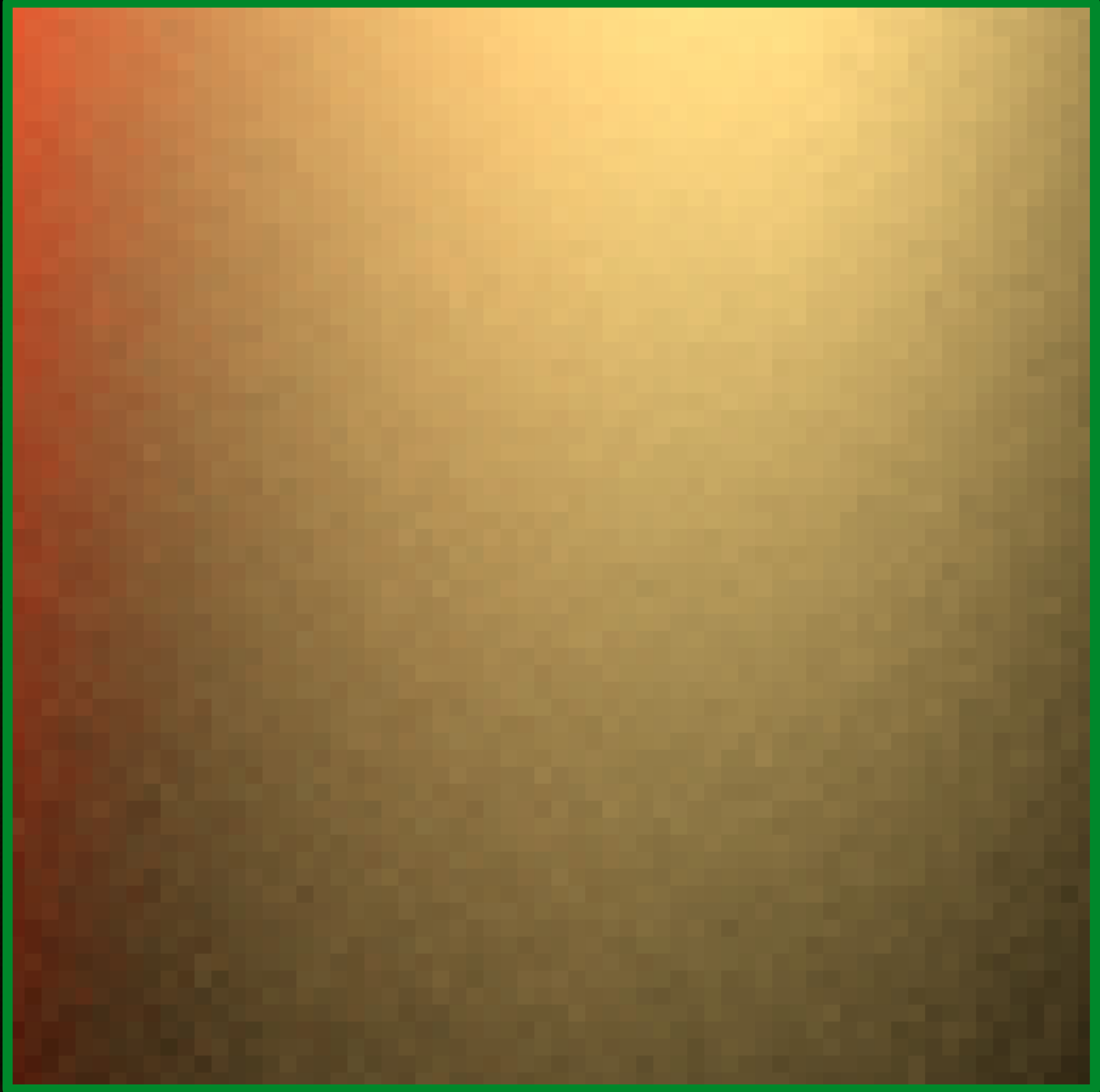
Sampler	Relative MSE	
	Full image	Crop
Random	1.481e-3	6.755e-4
Jittered2D (pad)	1.036e-3	6.123e-4
CMJ2D (pad)	8.721e-4	6.142e-4
(0,2)-seq. (pad)	8.299e-4	2.825e-4
Ours	**7.864e-4**	**1.587e-4**



Halton

121 spp

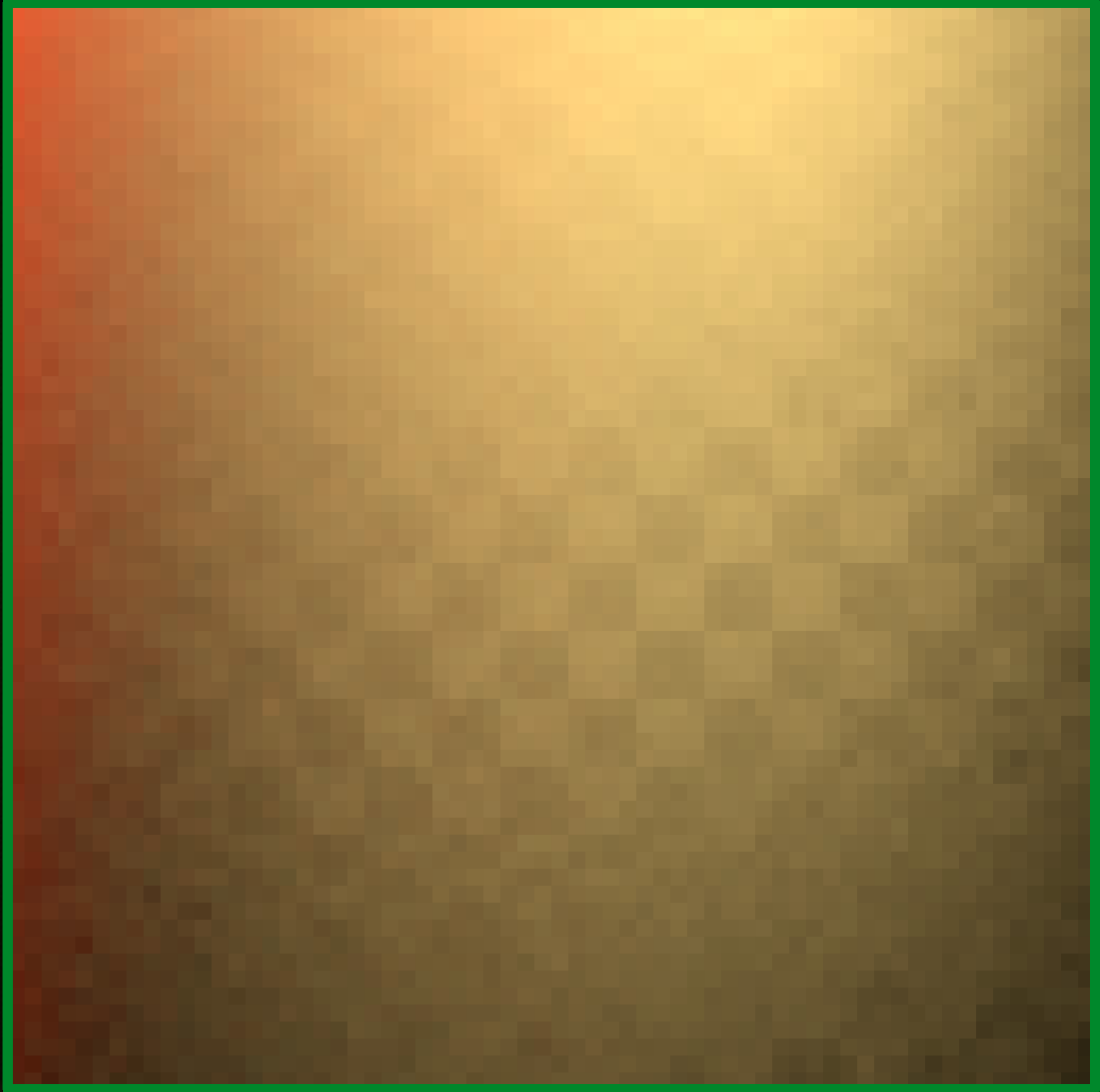
Sampler	Relative MSE	
	Full image	Crop
Random	1.481e-3	6.755e-4
Jittered2D (pad)	1.036e-3	6.123e-4
CMJ2D (pad)	8.721e-4	6.142e-4
(0,2)-seq. (pad)	8.299e-4	2.825e-4
Ours	7.864e-4	**1.587e-4**
Halton	**7.819e-4**	1.683e-4



Sobol

128 spp

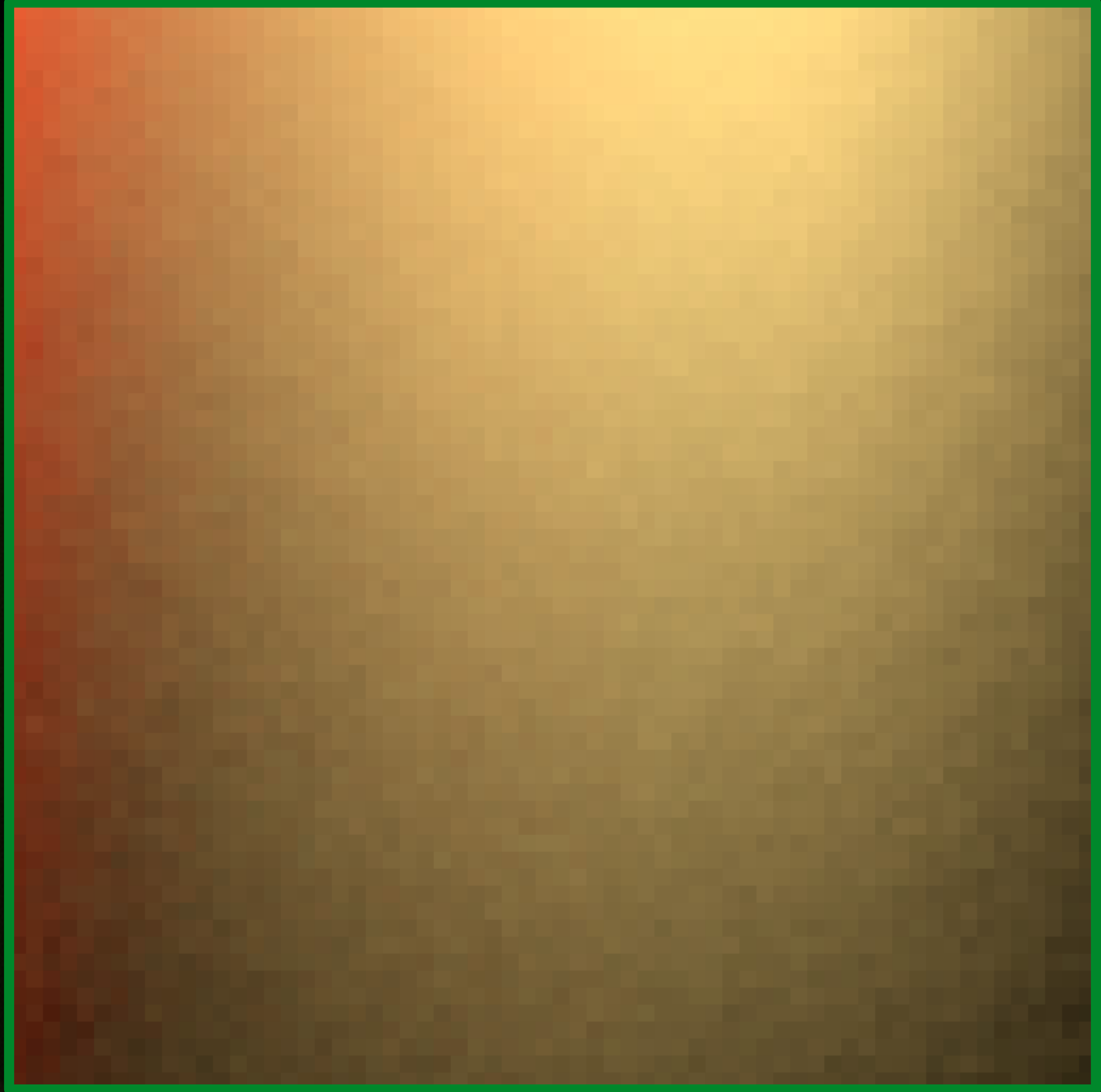
Sampler	Relative MSE	
	Full image	Crop
Random	1.481e-3	6.755e-4
Jittered2D (pad)	1.036e-3	6.123e-4
CMJ2D (pad)	8.721e-4	6.142e-4
(0,2)-seq. (pad)	8.299e-4	2.825e-4
Ours	7.864e-4	**1.587e-4**
Halton	7.819e-4	1.683e-4
Sobol	**6.510e-4**	3.493e-4



Ours

121 spp

Sampler	Relative MSE	
	Full image	Crop
Random	1.481e-3	6.755e-4
Jittered2D (pad)	1.036e-3	6.123e-4
CMJ2D (pad)	8.721e-4	6.142e-4
(0,2)-seq. (pad)	8.299e-4	2.825e-4
Ours	7.864e-4	**1.587e-4**
Halton	7.819e-4	1.683e-4
Sobol	**6.510e-4**	3.493e-4



9D integrand

Ours

121 spp

Sampler	Relative MSE	
	Full image	Crop
Random	3.959e-3	2.857e-2
Jittered2D (pad)	1.669e-3	1.112e-2
CMJ2D (pad)	1.557e-3	1.136e-2
(0,2)-seq. (pad)	1.477e-3	1.075e-2
Ours	**1.215e-3**	**6.099e-3**

Halton

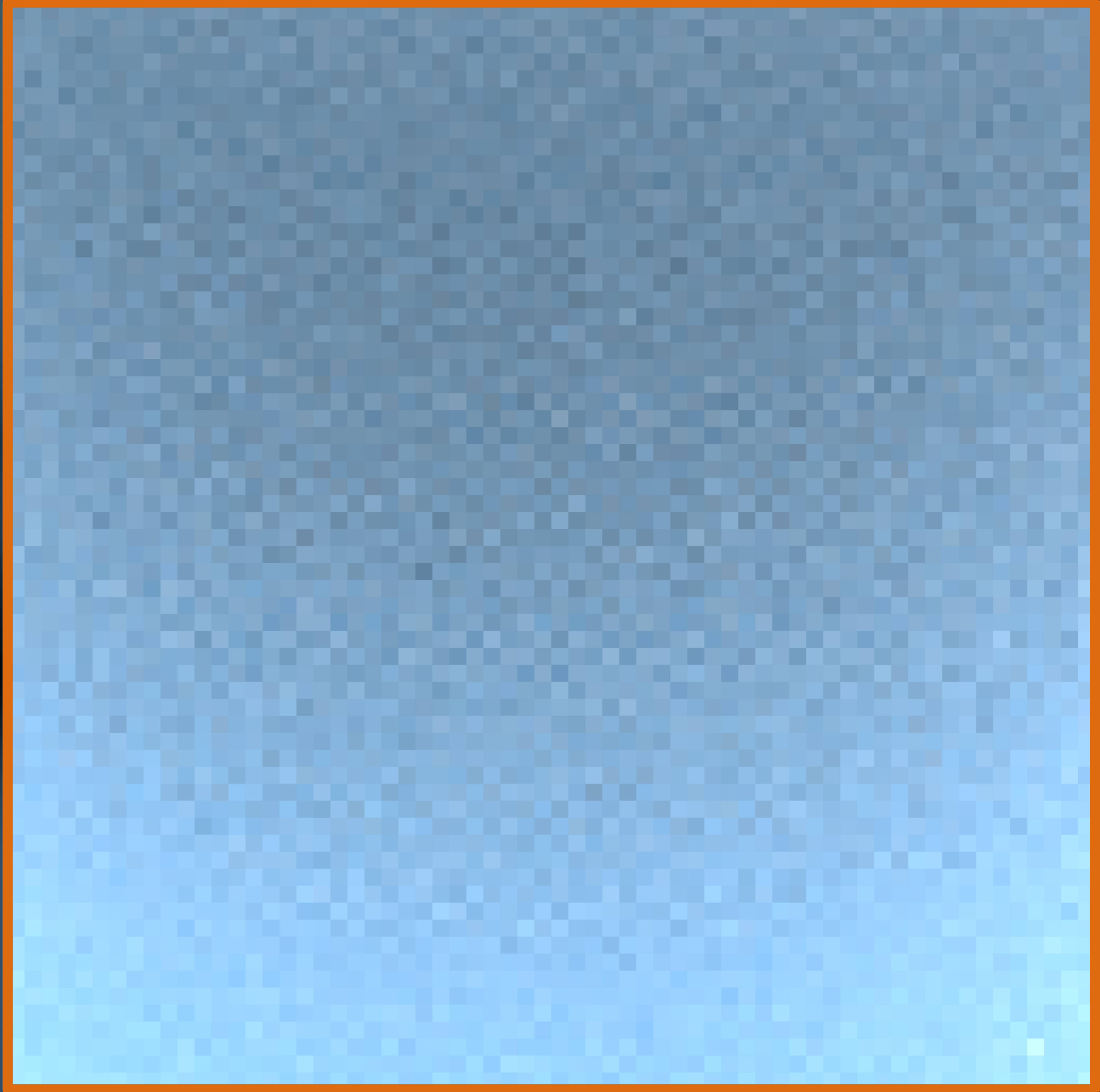
128 spp

Sampler	Relative MSE	
	Full image	Crop
Random	3.959e-3	2.857e-2
Jittered2D (pad)	1.669e-3	1.112e-2
CMJ2D (pad)	1.557e-3	1.136e-2
(0,2)-seq. (pad)	1.477e-3	1.075e-2
Ours	**1.215e-3**	**6.099e-3**
Halton	1.408e-3	6.912e-3

Sobol

128 spp

Sampler	Relative MSE	
	Full image	Crop
Random	3.959e-3	2.857e-2
Jittered2D (pad)	1.669e-3	1.112e-2
CMJ2D (pad)	1.557e-3	1.136e-2
(0,2)-seq. (pad)	1.477e-3	1.075e-2
Ours	1.215e-3	**6.099e-3**
Halton	1.408e-3	6.912e-3
Sobol	**1.117e-3**	6.185e-3



Ours

121 spp

Sampler	Relative MSE	
	Full image	Crop
Random	3.959e-3	2.857e-2
Jittered2D (pad)	1.669e-3	1.112e-2
CMJ2D (pad)	1.557e-3	1.136e-2
(0,2)-seq. (pad)	1.477e-3	1.075e-2
Ours	1.215e-3	**6.099e-3**
Halton	1.408e-3	6.912e-3
Sobol	**1.117e-3**	6.185e-3

43D integrand

Ours

3969 spp

	Relative MSE	
Sampler	Full image	Crop
Random	8.701e-4	1.503e-3
Jittered2D (pad)	7.385e-4	1.529e-3
CMJ2D (pad)	6.524e-4	9.821e-4
(0,2)-seq. (pad)	7.152e-4	1.457e-3
Ours	**6.024e-4**	**9.123e-4**

Sobol

4096 spp

Sampler	Relative MSE	
	Full image	Crop
Random	8.701e-4	1.503e-3
Jittered2D (pad)	7.385e-4	1.529e-3
CMJ2D (pad)	6.524e-4	9.821e-4
(0,2)-seq. (pad)	7.152e-4	1.457e-3
Ours	6.024e-4	9.123e-4
Halton	**5.773e-4**	9.845e-4
Sobol	5.994e-4	**8.753e-4**

Summary

Summary

OAs with $t = 2$ consistently outperform 2D padding

- drop-in replacement for 2D padded point sets!

Summary

OAs with $t = 2$ consistently outperform 2D padding

- drop-in replacement for 2D padded point sets!

High-dimensional QMC is sometimes better…

- but structured artifacts

Limitations/Future work

Limitations/Future work

✘ Only finite point sets; not progressive

Limitations/Future work

- ✘ Only finite point sets; not progressive
- ✘ Strength t OAs provide no stratification beyond tD

Limitations/Future work

- ✘ Only finite point sets; not progressive
- ✘ Strength t OAs provide no stratification beyond tD
  - Asymptotically no better than random when integrand $d > t$

Limitations/Future work

- ✘ Only finite point sets; not progressive
- ✘ Strength t OAs provide no stratification beyond tD
  - Asymptotically no better than random when integrand $d > t$

💡 Nested orthogonal arrays [He and Qian 2011, …]

Limitations/Future work

✘ Only finite point sets; not progressive

✘ Strength t OAs provide no stratification beyond tD

- Asymptotically no better than random when integrand $d > t$

💡 Nested orthogonal arrays [He and Qian 2011, …]

💡 Strong orthogonal arrays (SOA) [He and Tang 2013, …]

Limitations/Future work

✘ Only finite point sets; not progressive

✘ Strength t OAs provide no stratification beyond tD

- Asymptotically no better than random when integrand $d > t$

💡 Nested orthogonal arrays [He and Qian 2011, …]

💡 Strong orthogonal arrays (SOA) [He and Tang 2013, …]

- Instead of stratifying t-dimensions, stratify all dimensions $\leq t$

Limitations/Future work

✘ Only finite point sets; not progressive

✘ Strength t OAs provide no stratification beyond tD

- Asymptotically no better than random when integrand $d > t$

💡 Nested orthogonal arrays [He and Qian 2011, …]

💡 Strong orthogonal arrays (SOA) [He and Tang 2013, …]

- Instead of stratifying t-dimensions, stratify all dimensions $\leq t$
- (t, m, s)-nets and SOA equivalency

Limitations/Future work

✘ Only finite point sets; not progressive

✘ Strength t OAs provide no stratification beyond tD

- Asymptotically no better than random when integrand $d > t$

💡 Nested orthogonal arrays [He and Qian 2011, ...]

💡 Strong orthogonal arrays (SOA) [He and Tang 2013, ...]

- Instead of stratifying t-dimensions, stratify all dimensions $\leq t$
- (t, m, s)-nets and SOA equivalency
- (t, s)-sequences for progressive OA generation?

Thank you!

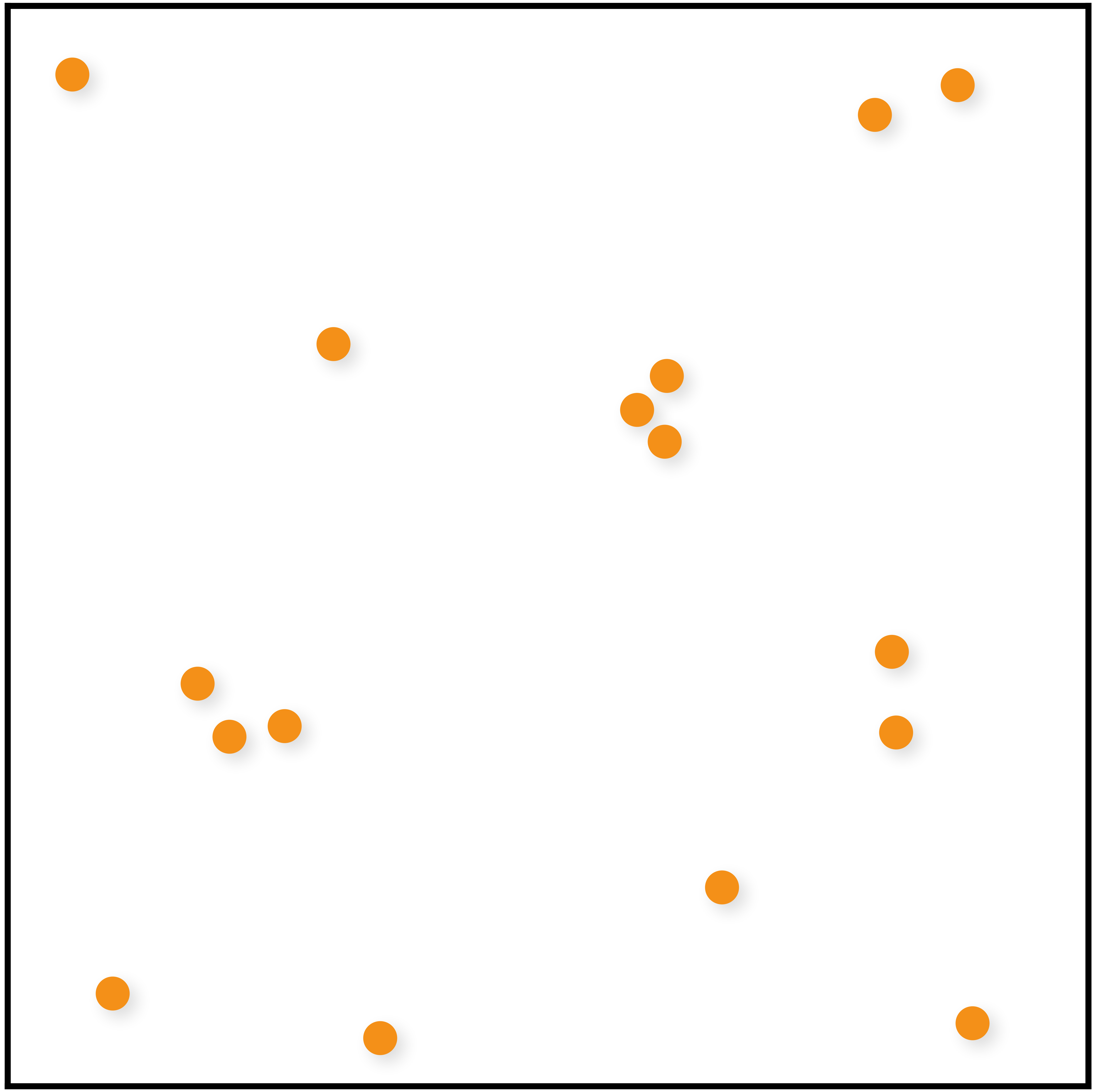
dartgo.org/OAS

additional results / code

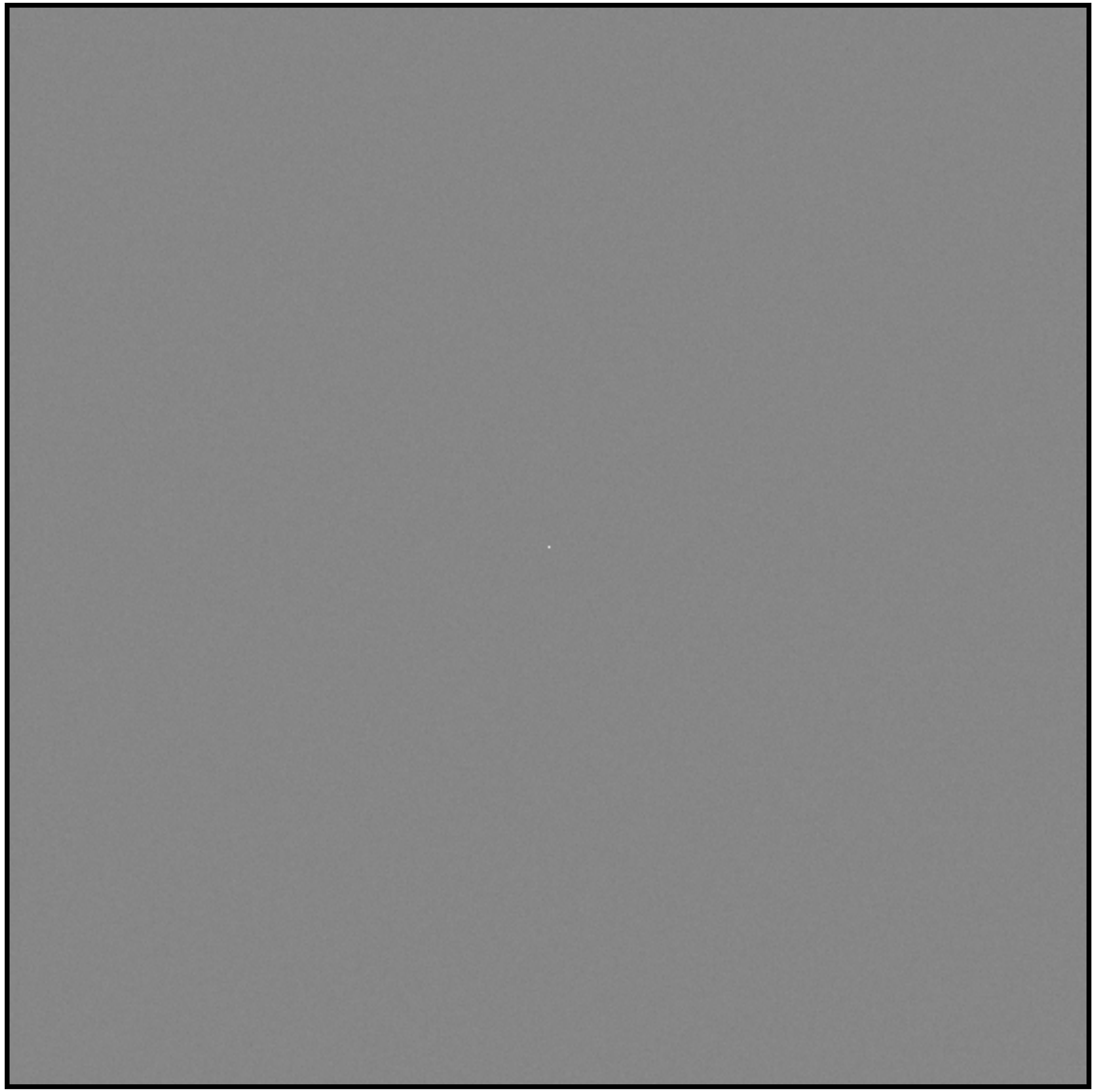
Backup slides

Independent Random Sampling

Spatial domain

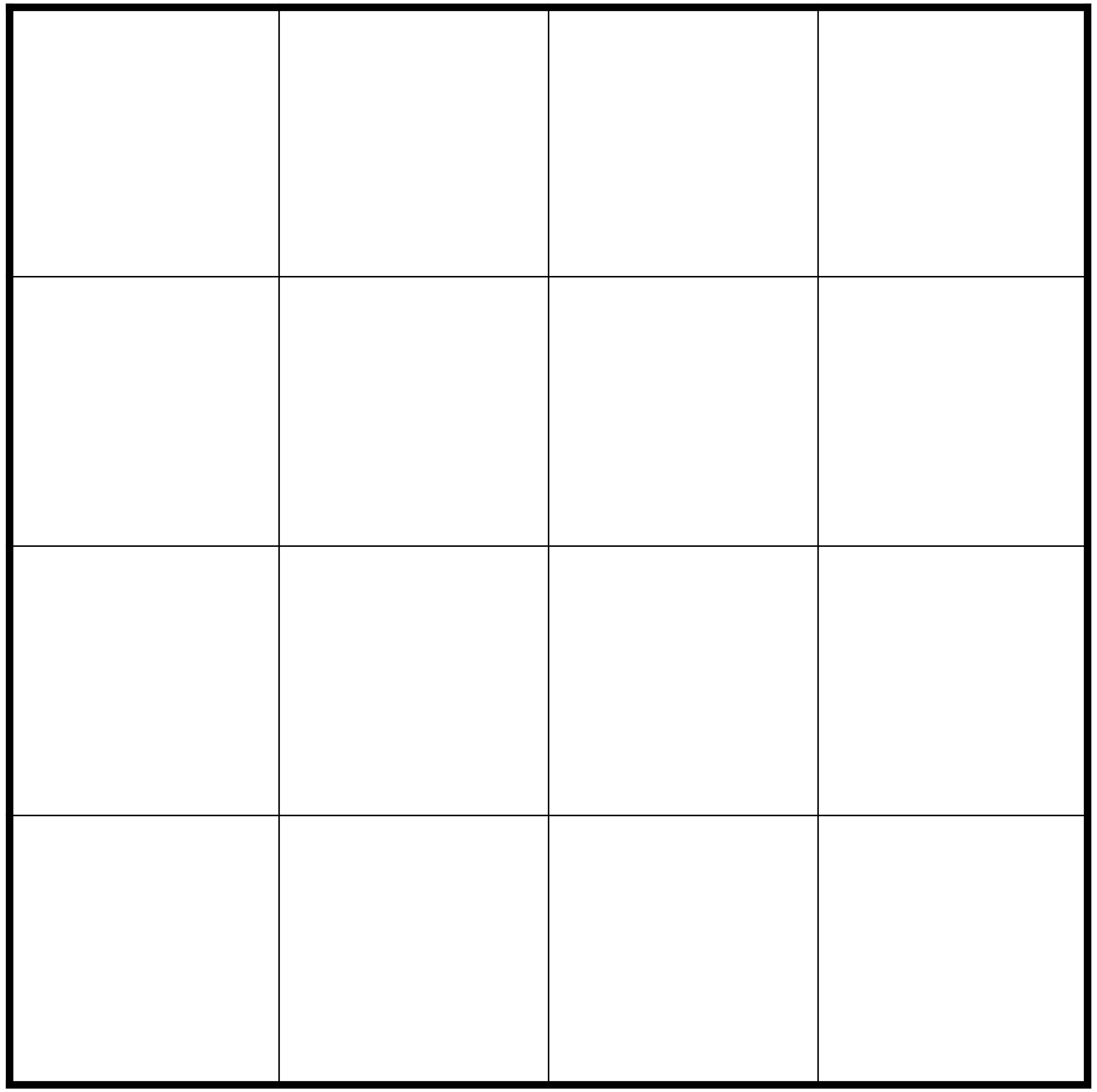


Fourier domain



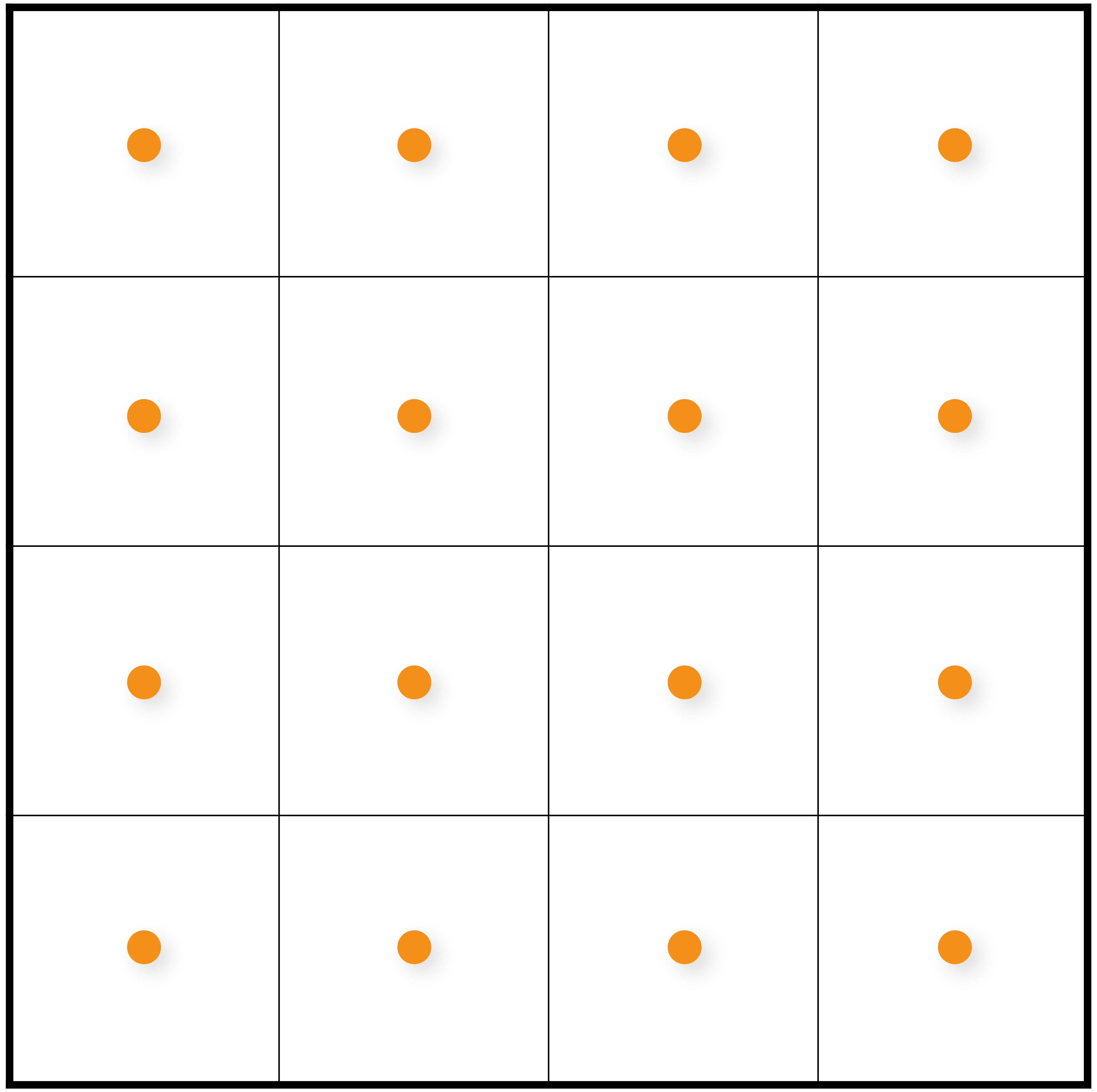
Regular Sampling

```
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

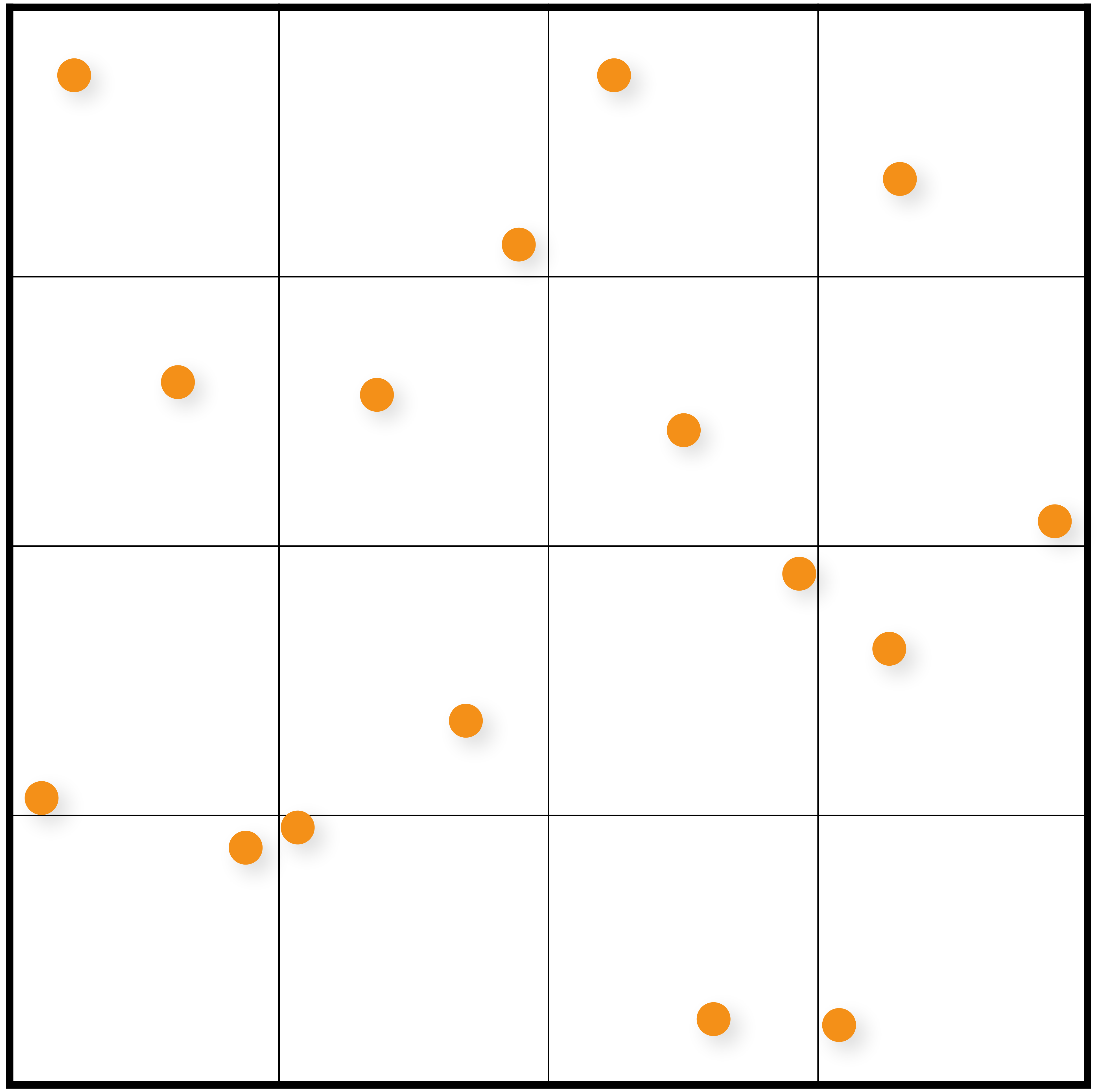
```
for (uint i = 0; i < numX; i++)
    for (uint j = 0; j < numY; j++)
    {
        samples(i,j).x = (i + 0.5)/numX;
        samples(i,j).y = (j + 0.5)/numY;
    }
```



Jittered Sampling

[Cook 86]

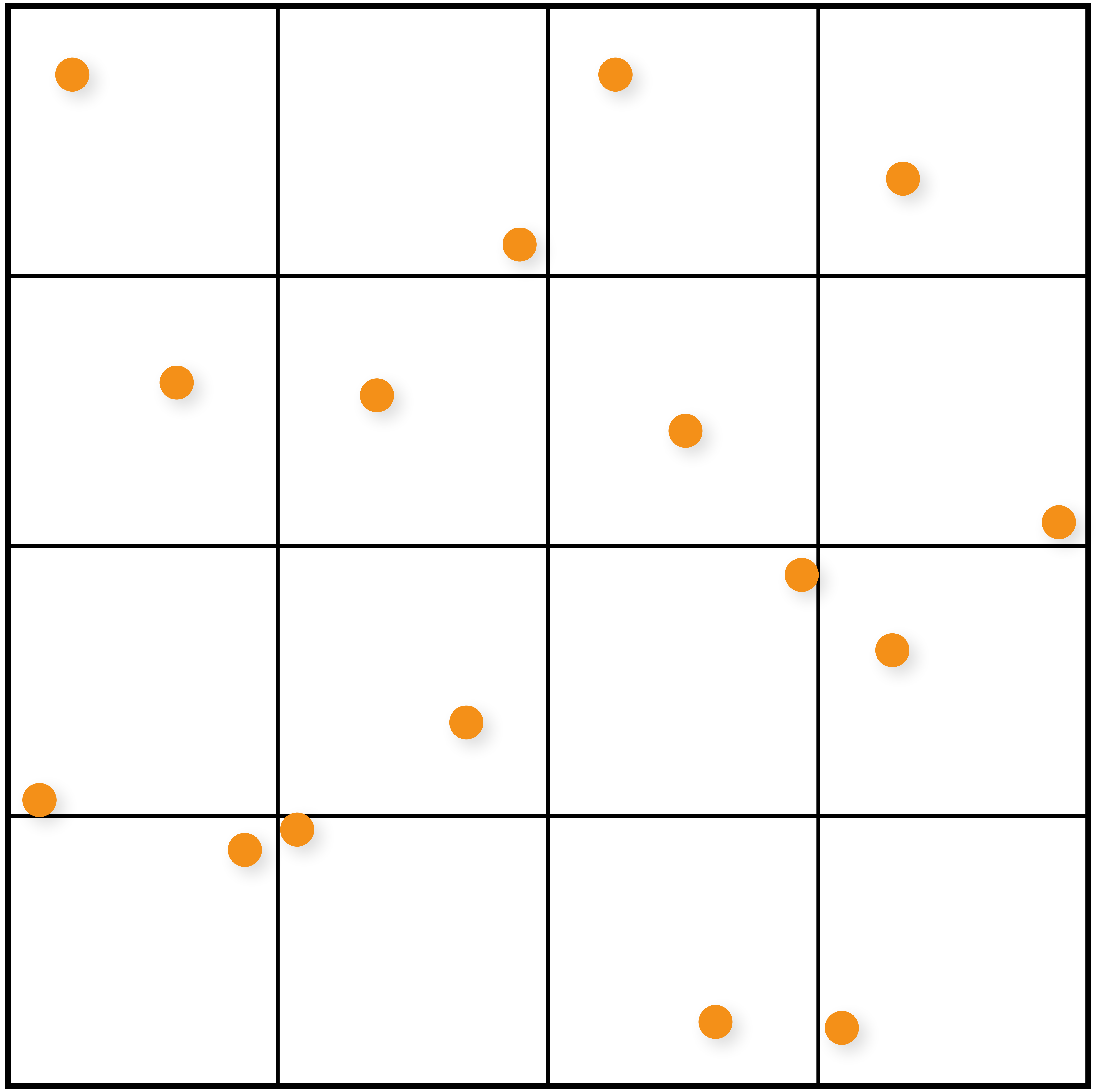
```
for (uint i = 0; i < numX; i++)
    for (uint j = 0; j < numY; j++)
    {
        samples(i,j).x = (i + randf())/numX;
        samples(i,j).y = (j + randf())/numY;
    }
```



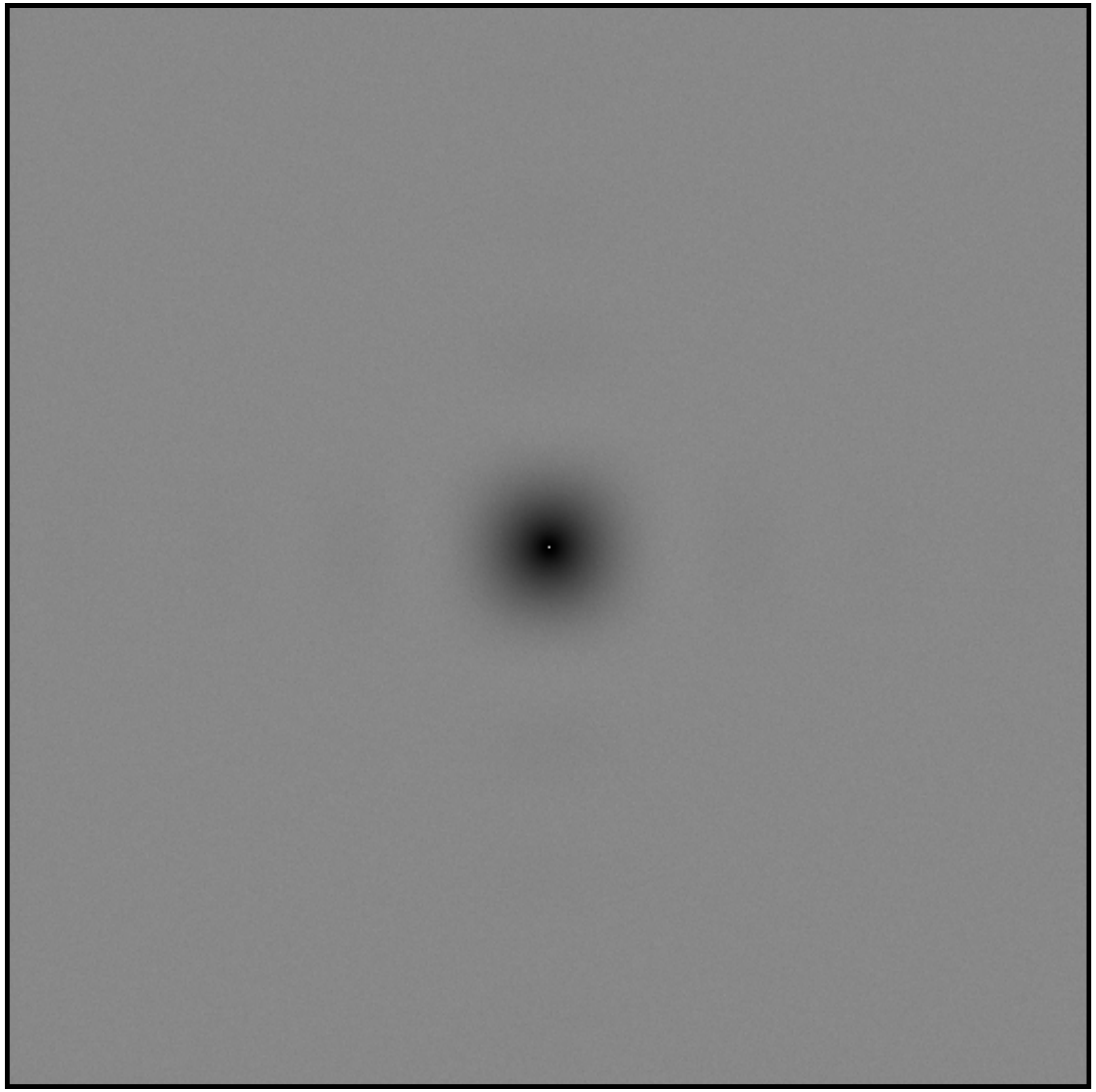
Jittered Sampling

[Cook 86]

Spatial domain

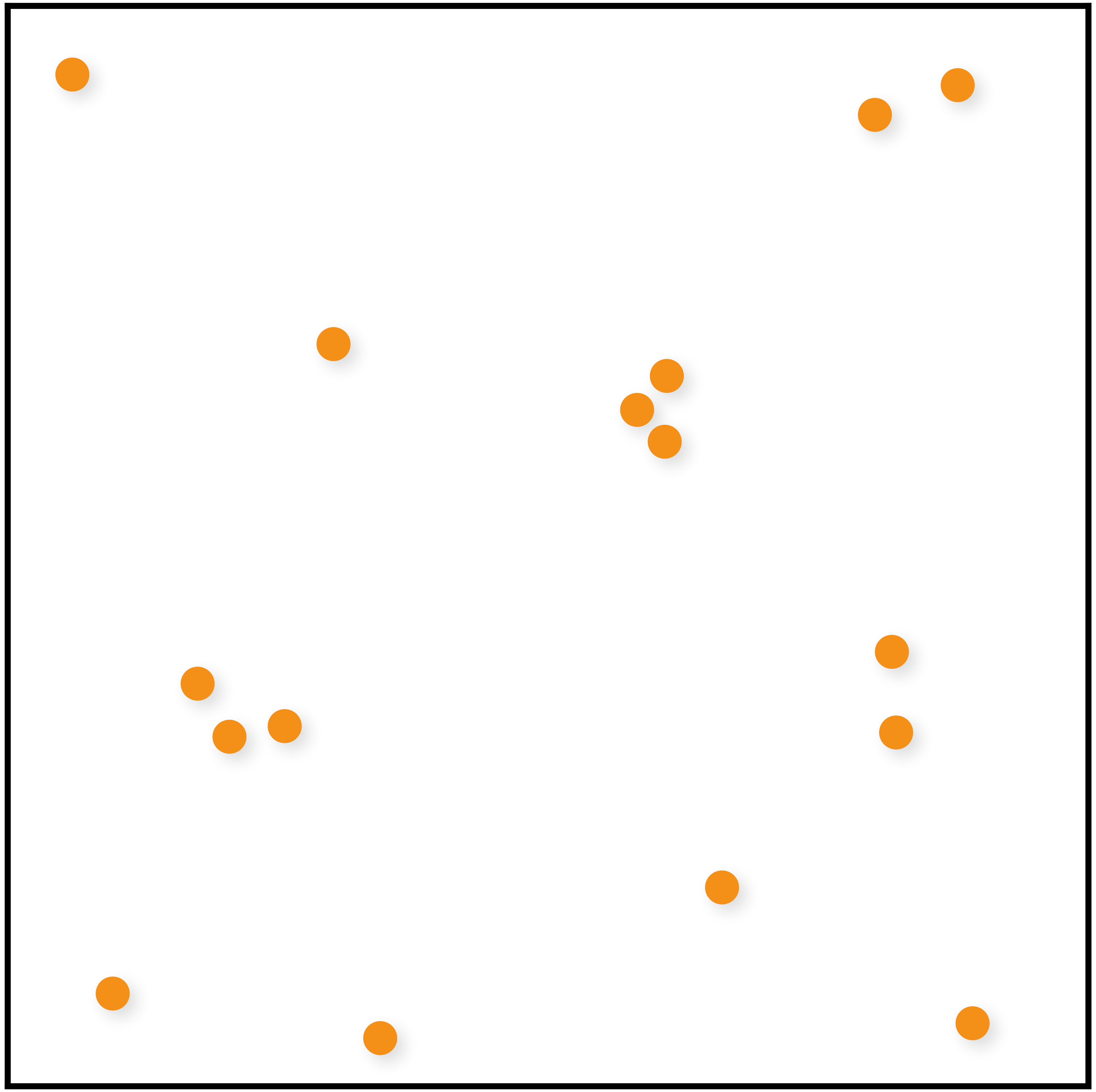


Fourier domain

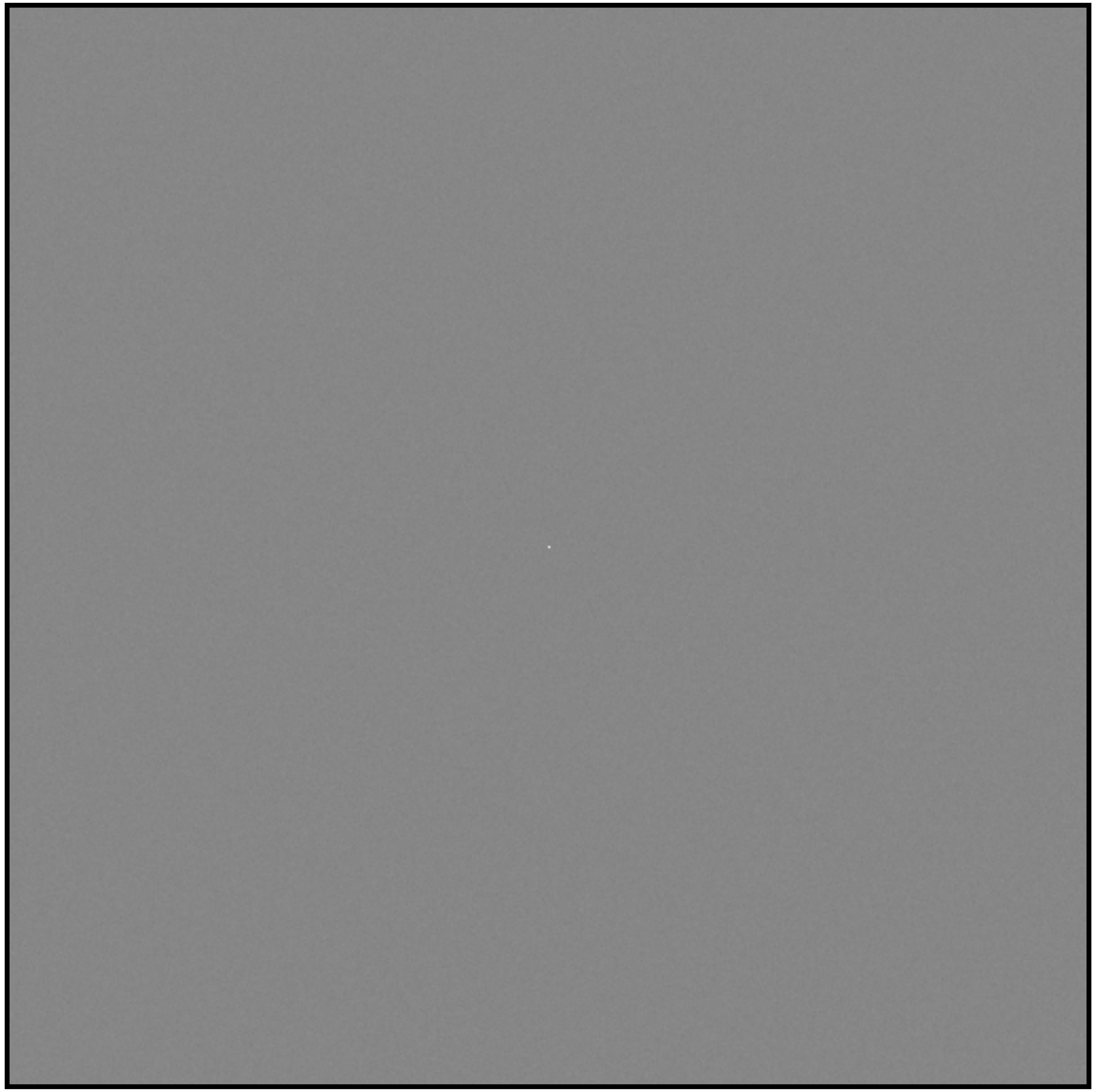


Independent Random Sampling

Spatial domain



Fourier domain



Latin Hypercube (N-Rooks) Sampling

[McKay et al. 79]

[Shirley 91]

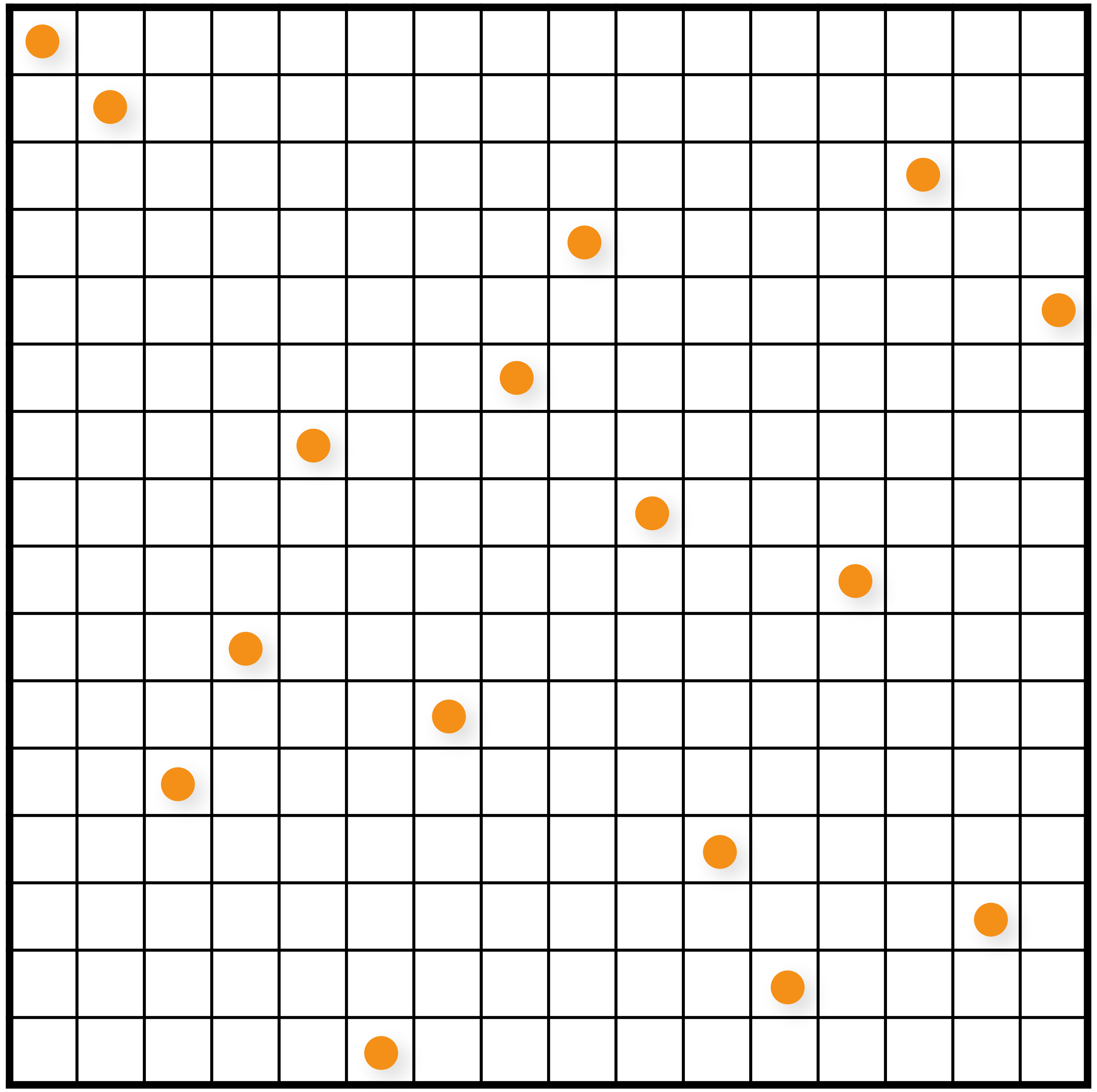
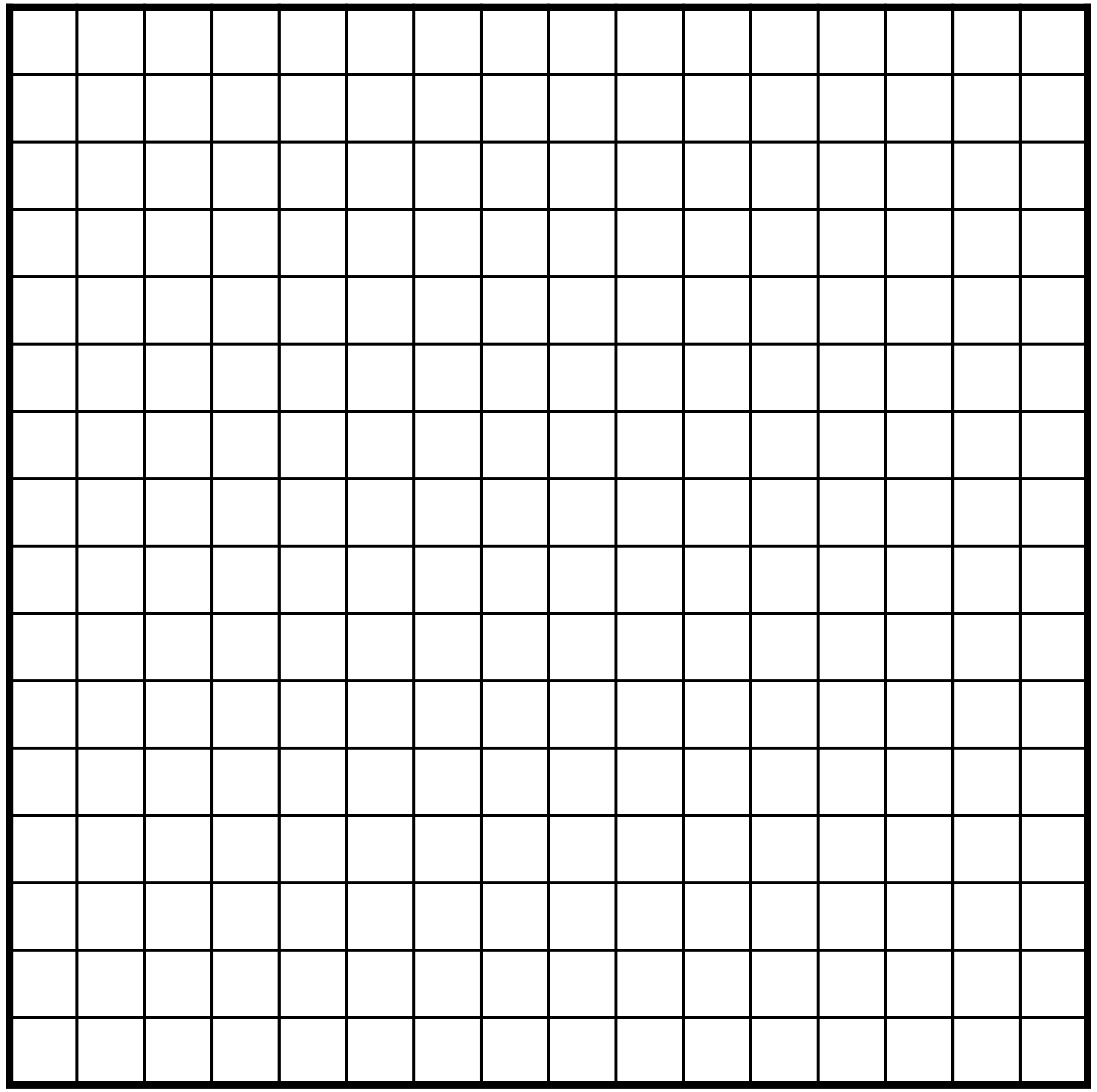
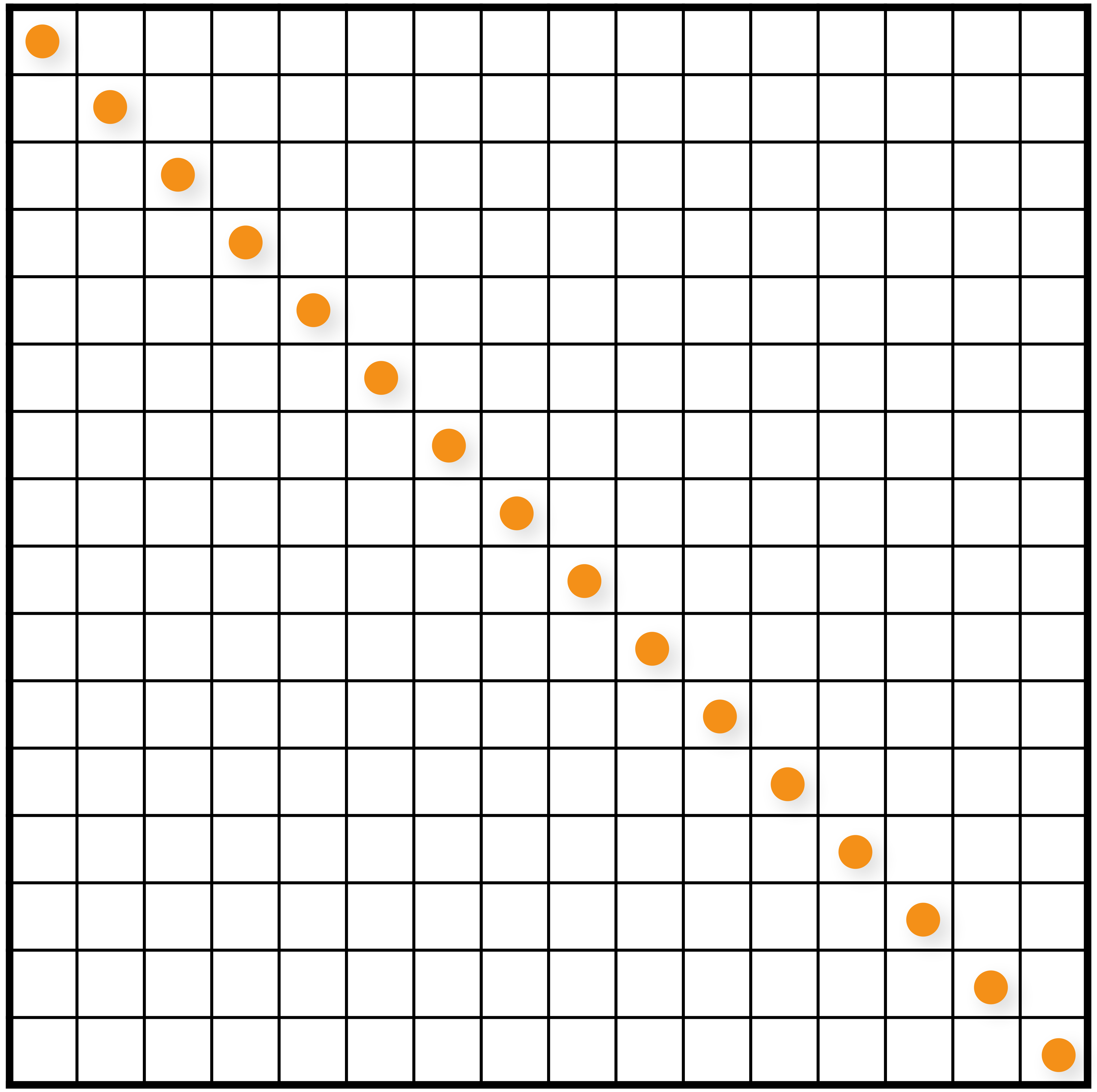


Image source: Michael Maggs, CC BY-SA 2.5

Latin Hypercube (N-Rooks) Sampling

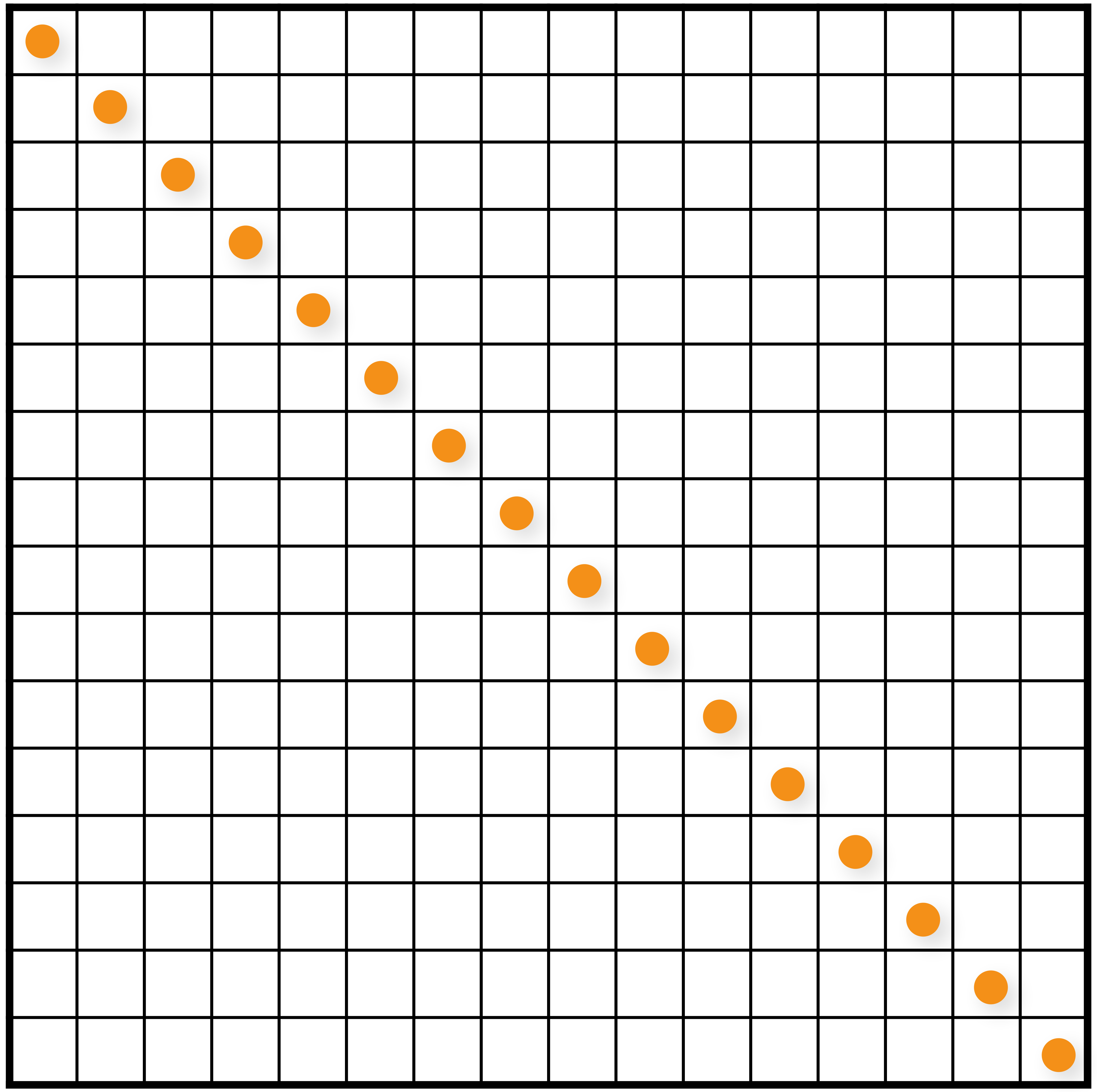


Latin Hypercube (N-Rooks) Sampling



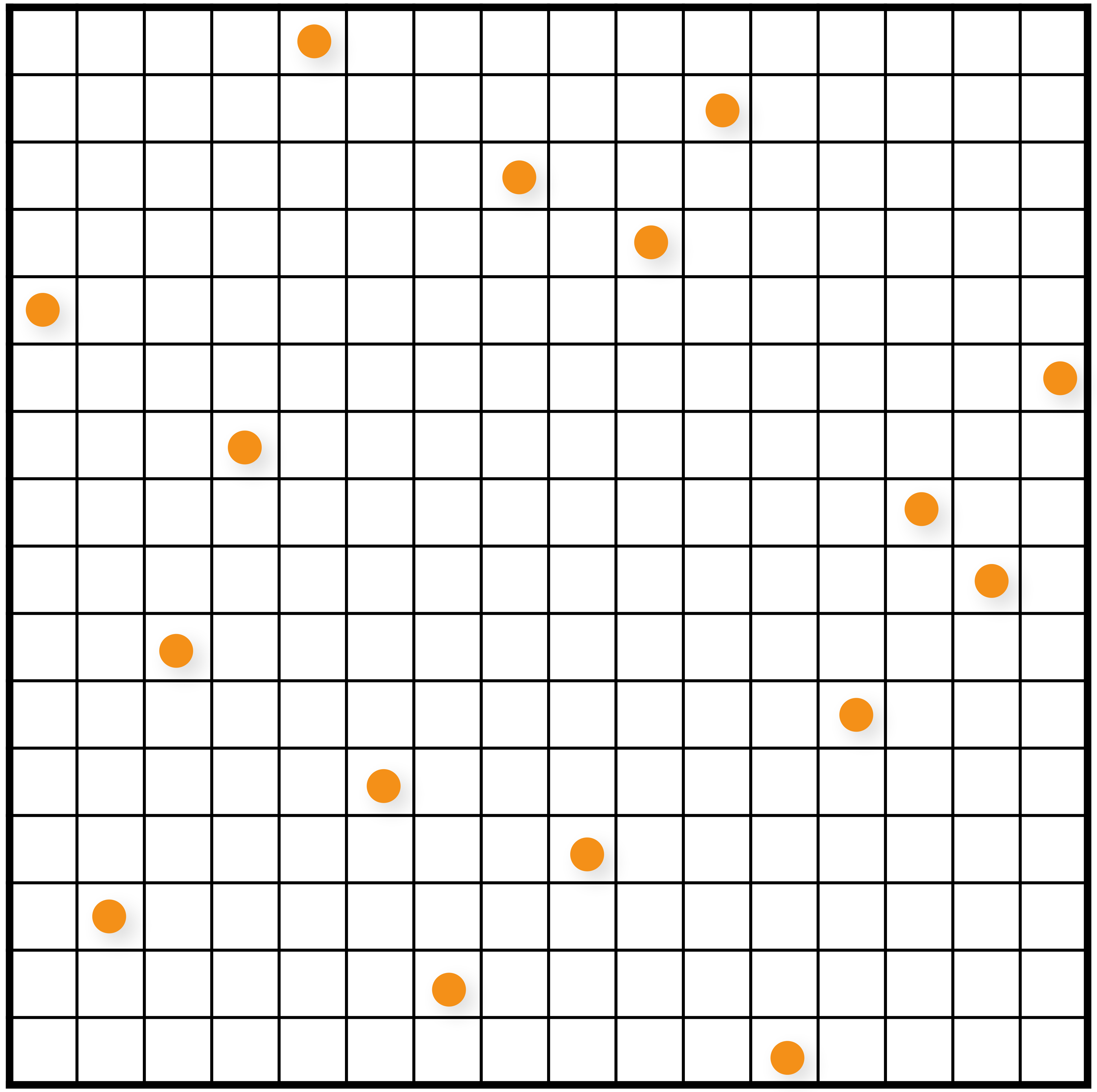
Initialize

Latin Hypercube (N-Rooks) Sampling



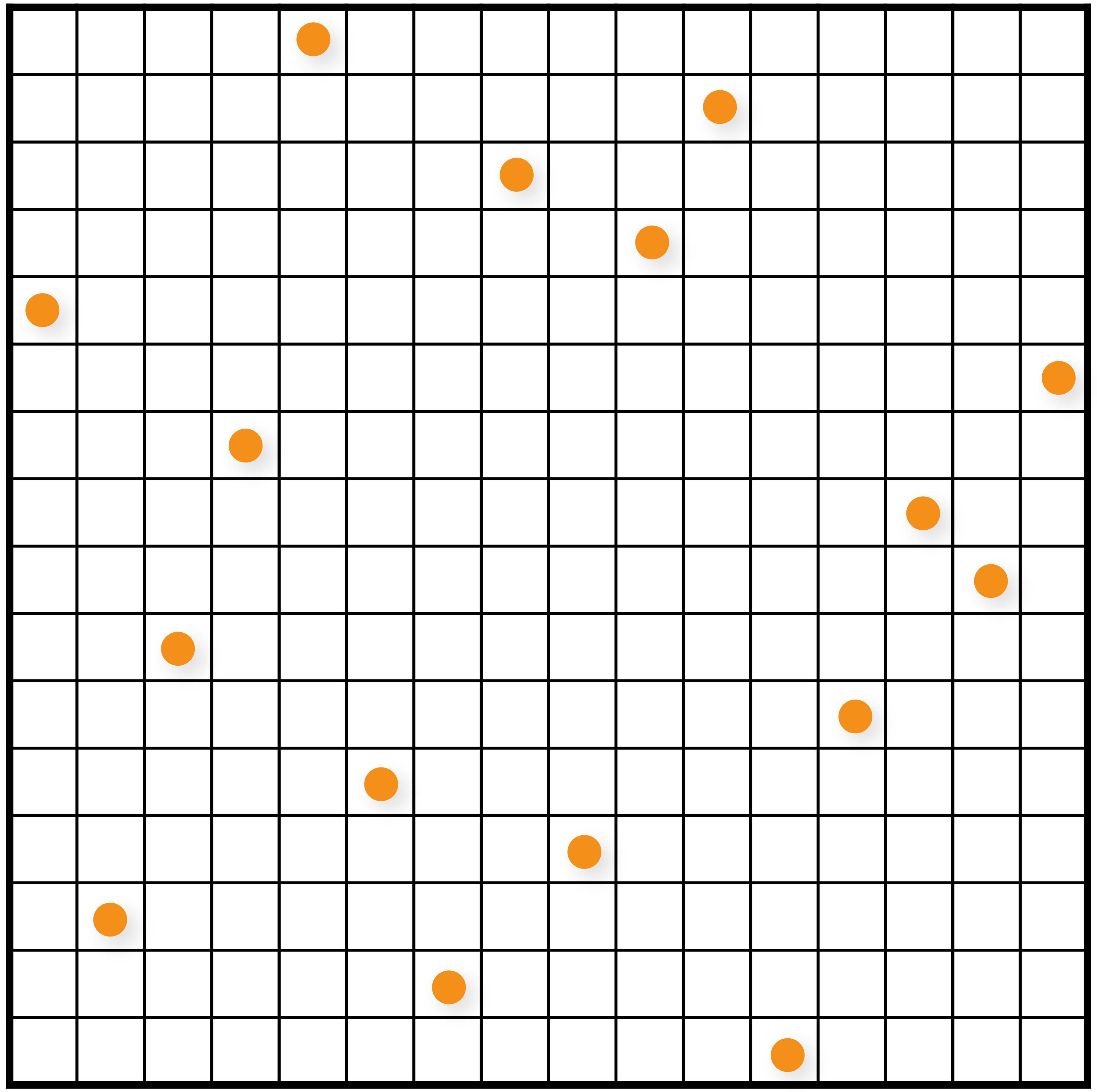
Shuffle rows

Latin Hypercube (N-Rooks) Sampling



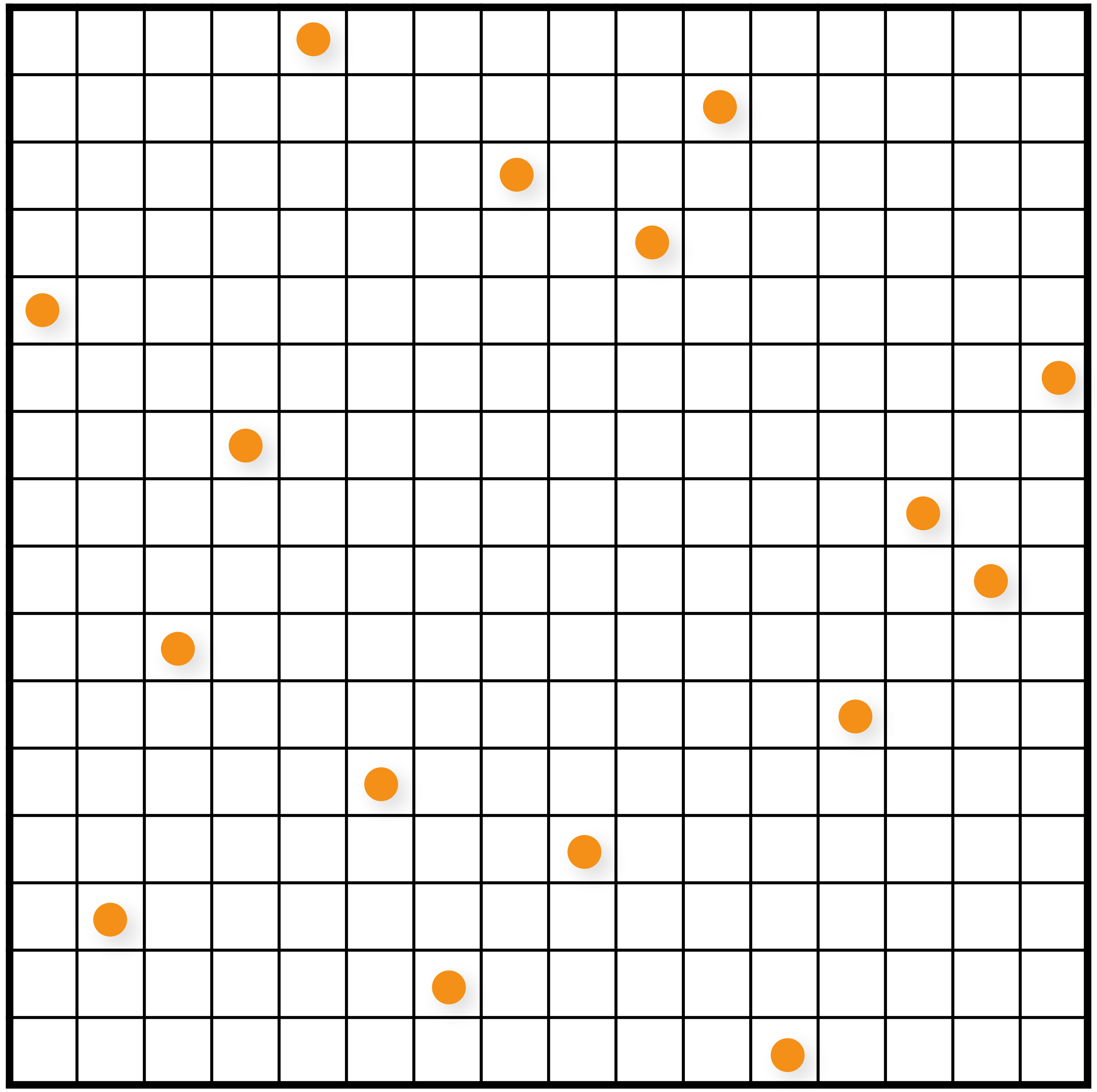
Shuffle rows

Latin Hypercube (N-Rooks) Sampling

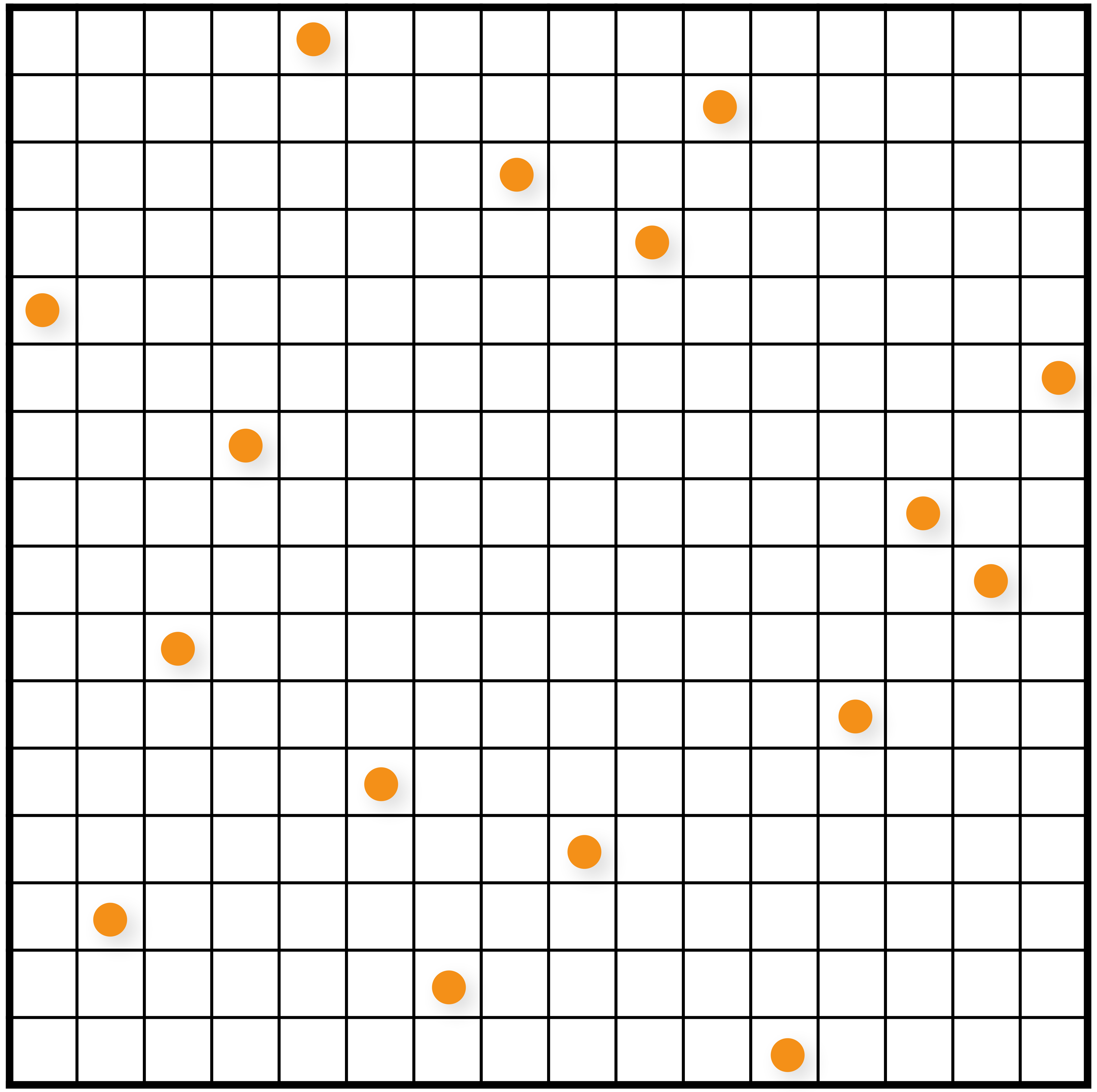


Shuffle rows

Latin Hypercube (N-Rooks) Sampling

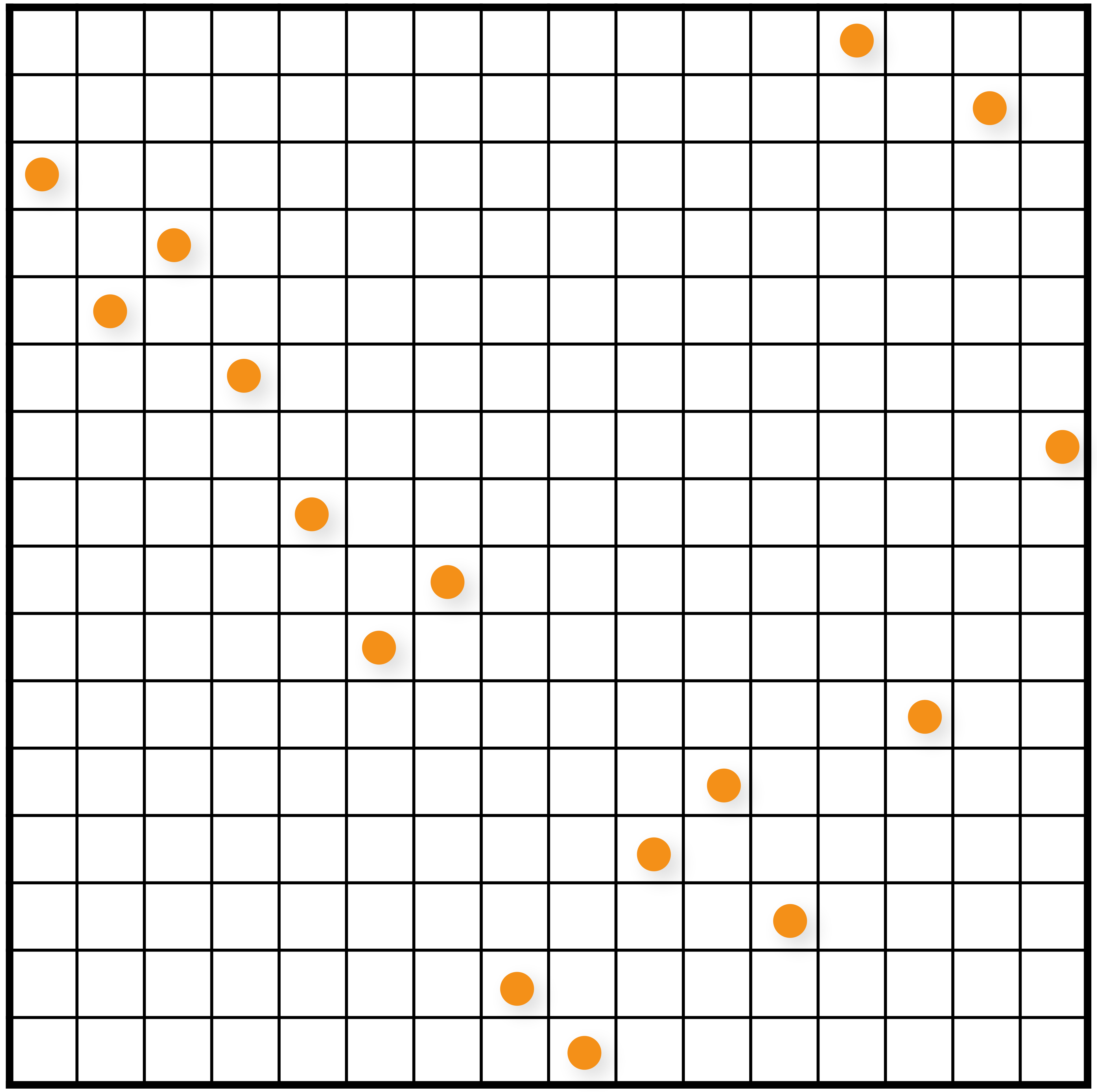


Latin Hypercube (N-Rooks) Sampling



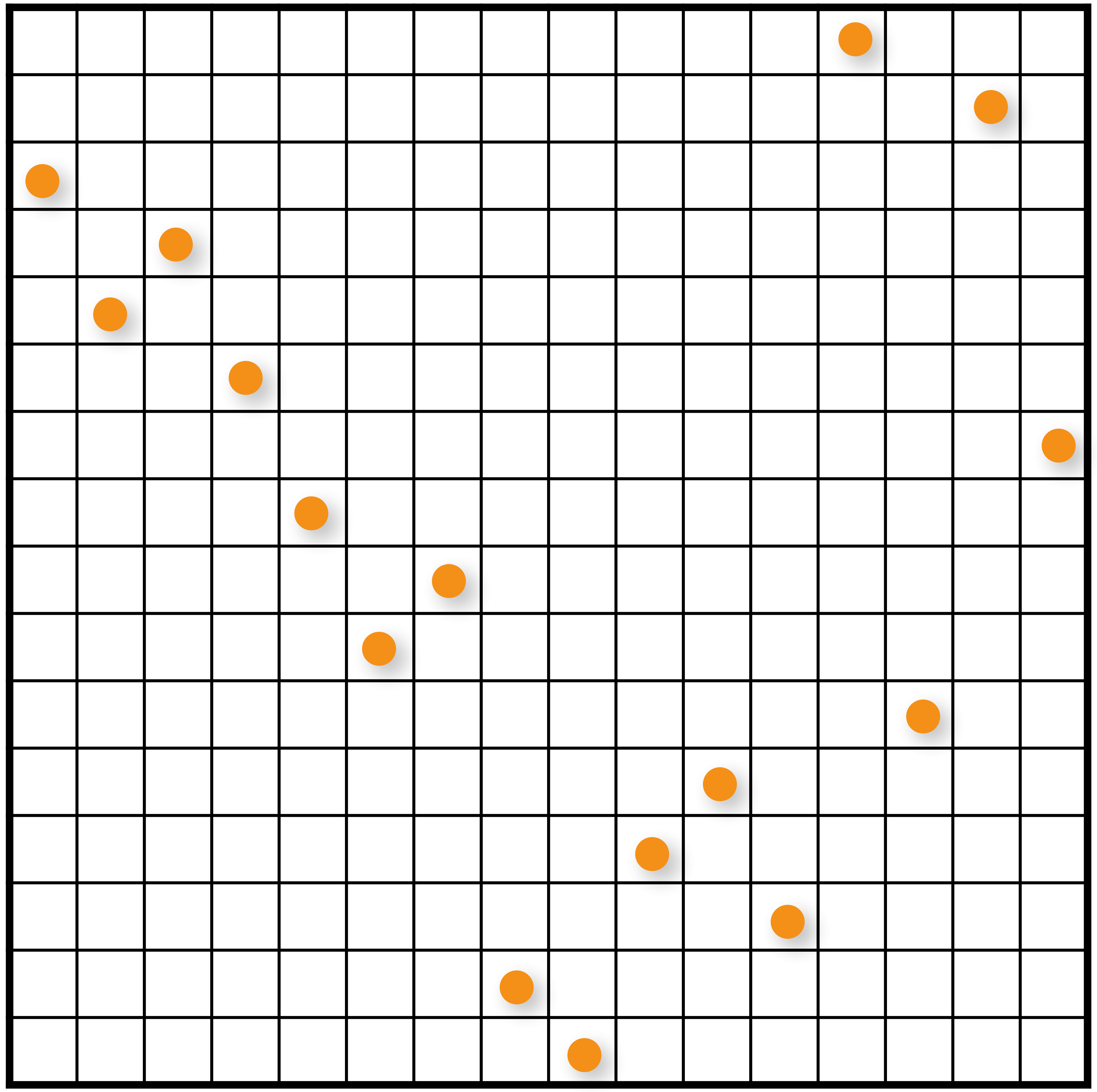
Shuffle columns

Latin Hypercube (N-Rooks) Sampling



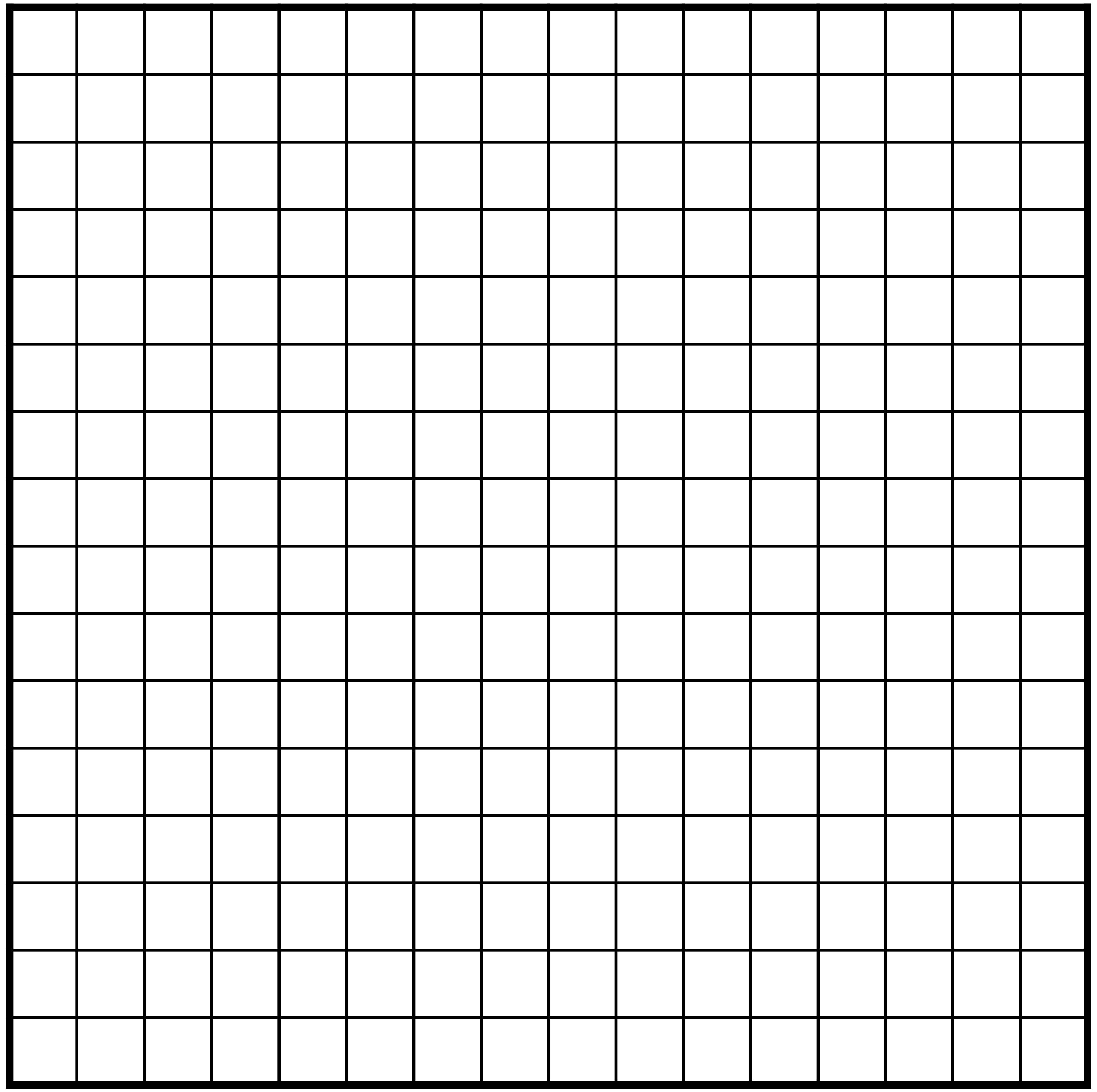
Shuffle columns

Latin Hypercube (N-Rooks) Sampling



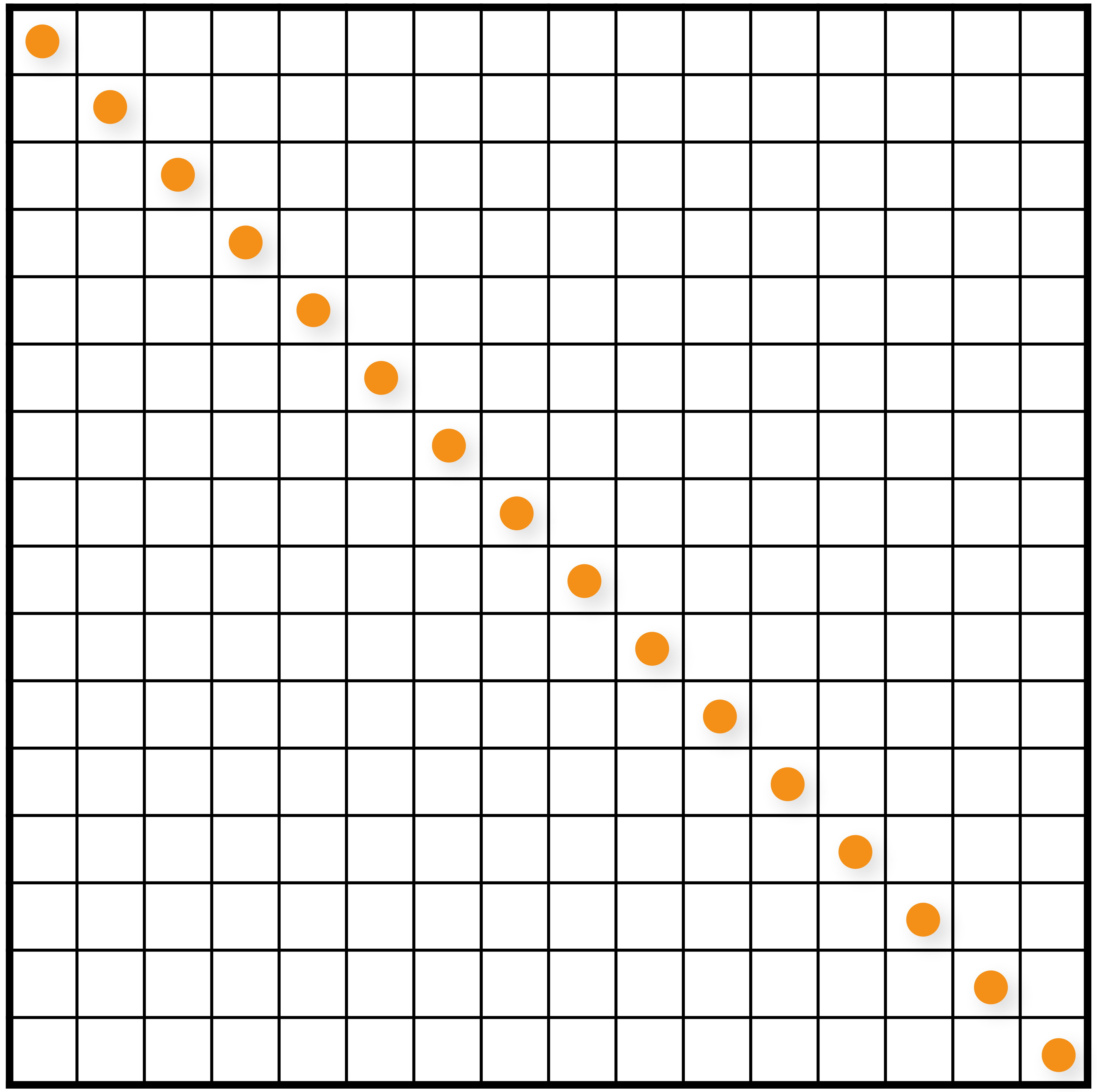
Latin Hypercube (N-Rooks) Sampling

```
// 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,:));
```

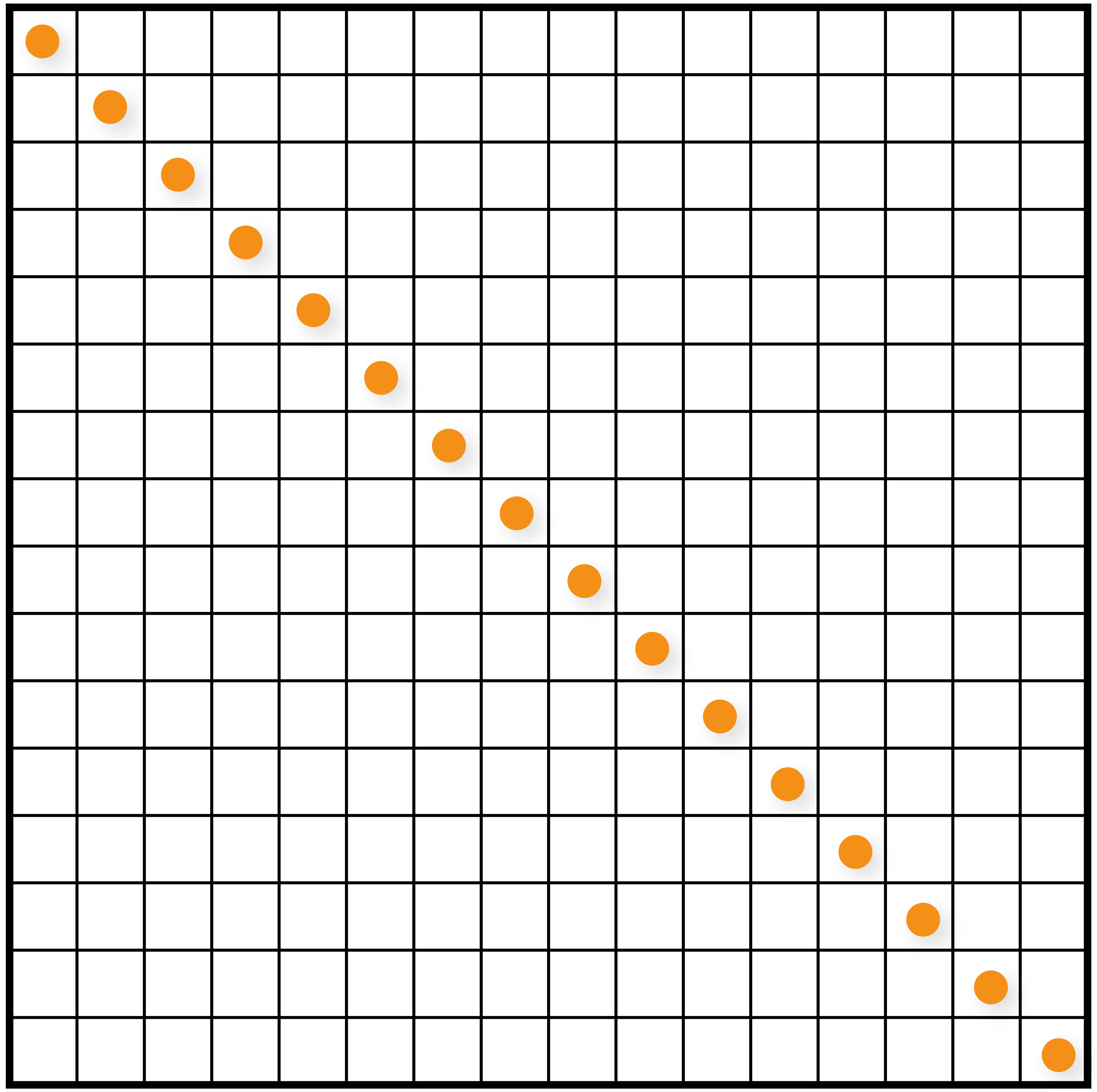


Initialize

Latin Hypercube (N-Rooks) Sampling

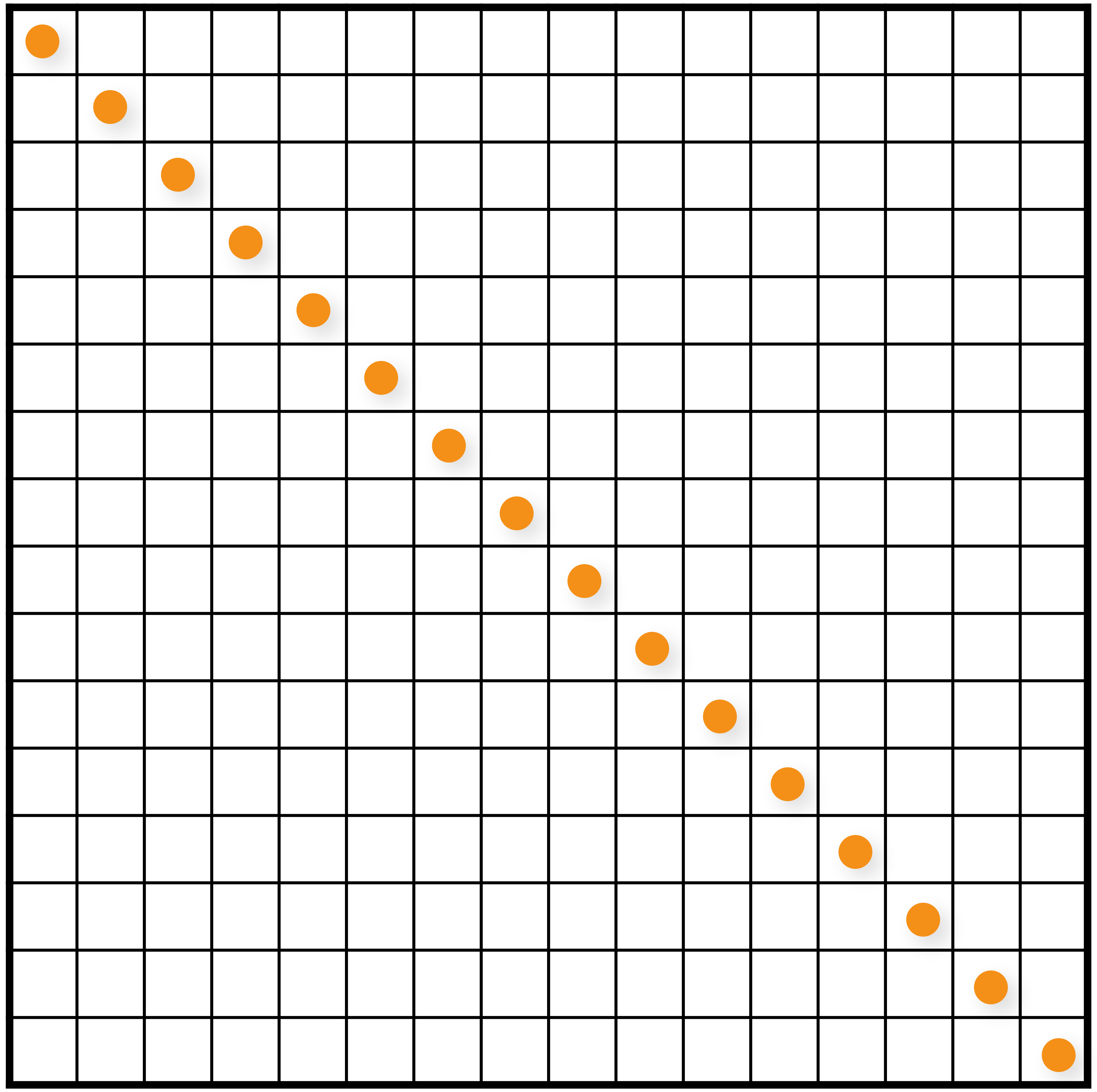
```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
    for (uint i = 0; i < numS; i++)
        samples(d,i) = (i + randf())/numS;
```

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



Latin Hypercube (N-Rooks) Sampling

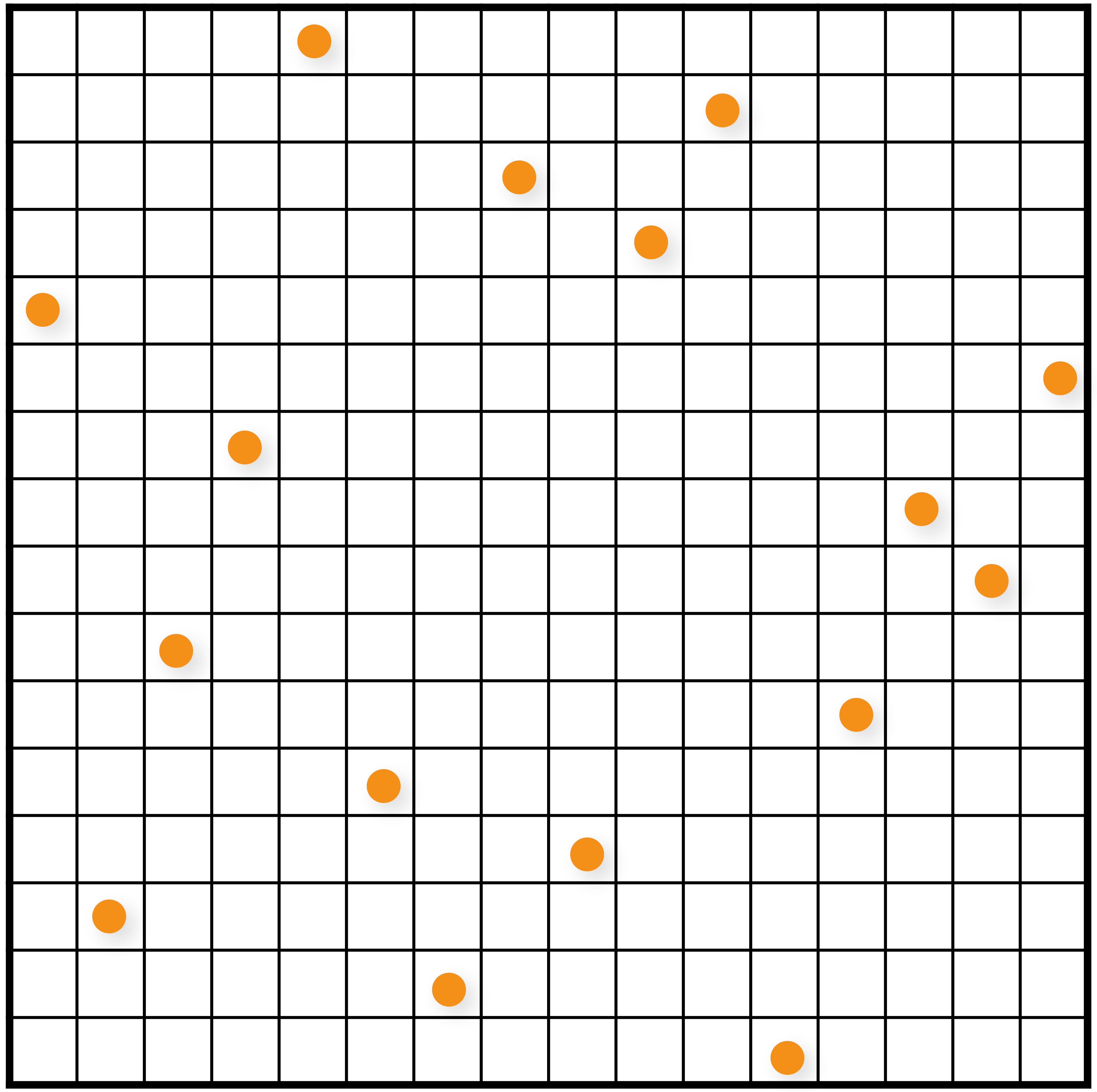
```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
    for (uint i = 0; i < numS; i++)
        samples(d,i) = (i + randf())/numS;

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



Shuffle rows

Latin Hypercube (N-Rooks) Sampling

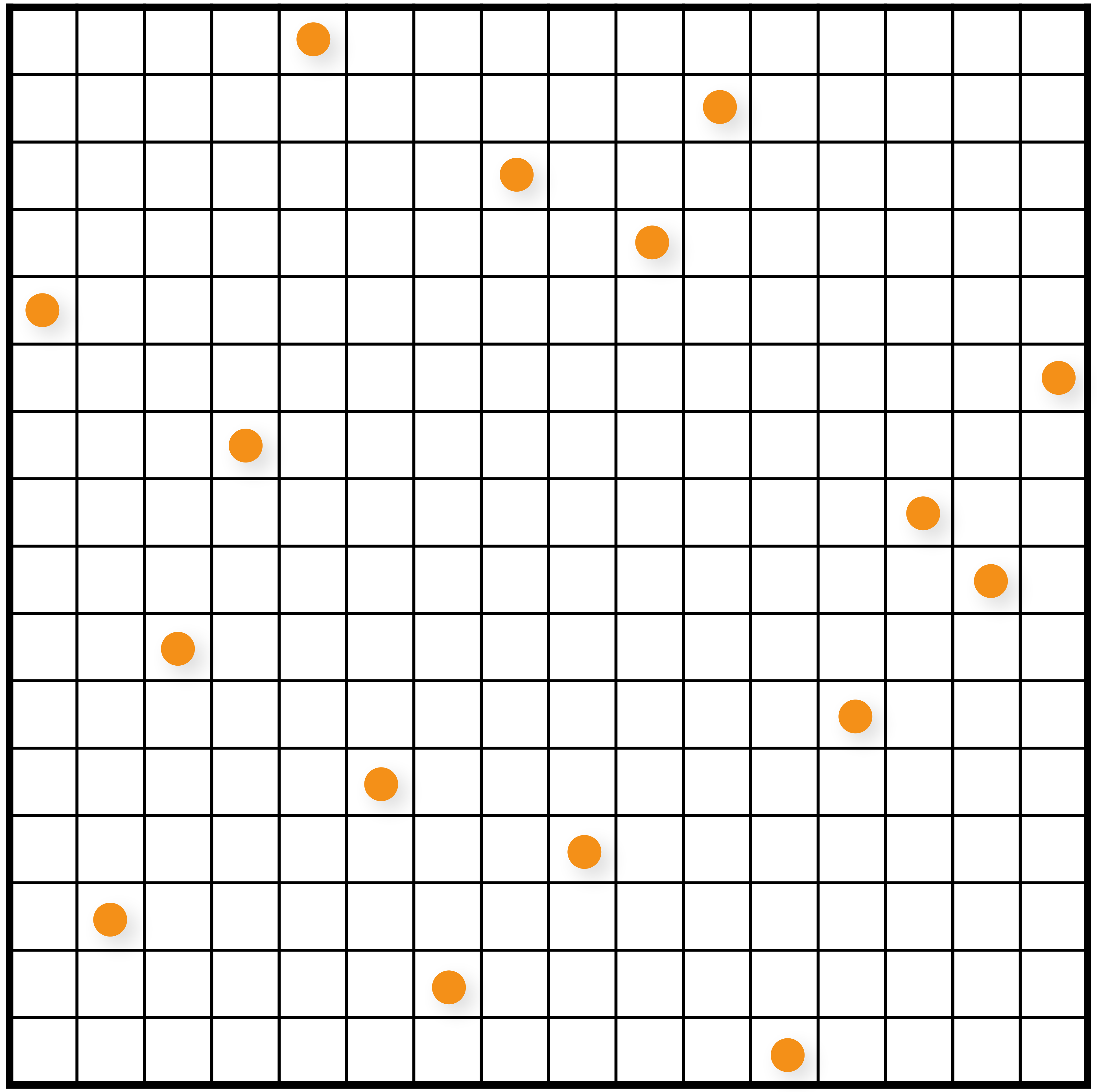
```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
    for (uint i = 0; i < numS; i++)
        samples(d,i) = (i + randf())/numS;

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



Shuffle rows

Latin Hypercube (N-Rooks) Sampling

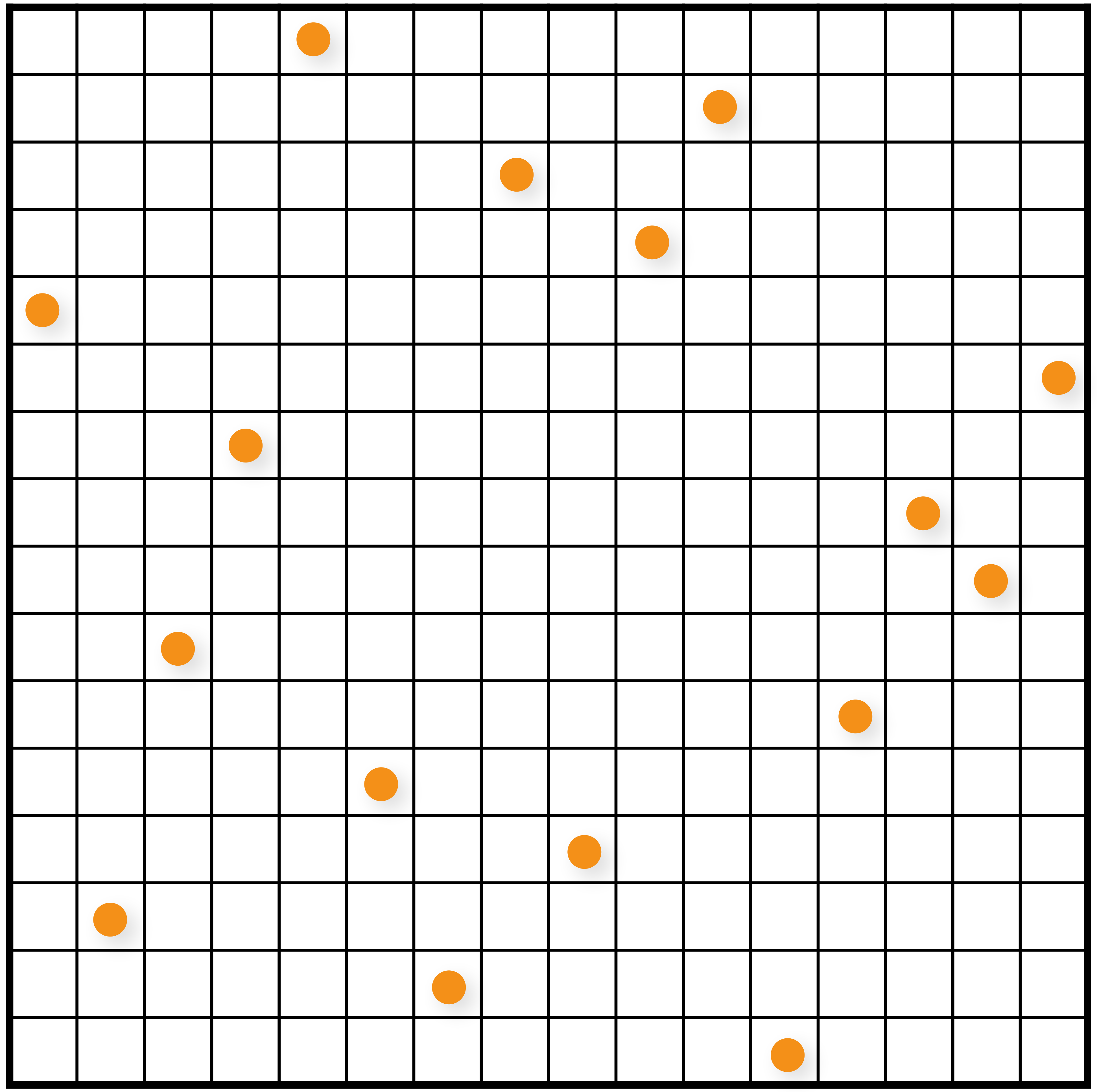
```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
    for (uint i = 0; i < numS; i++)
        samples(d,i) = (i + randf())/numS;

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



Shuffle rows

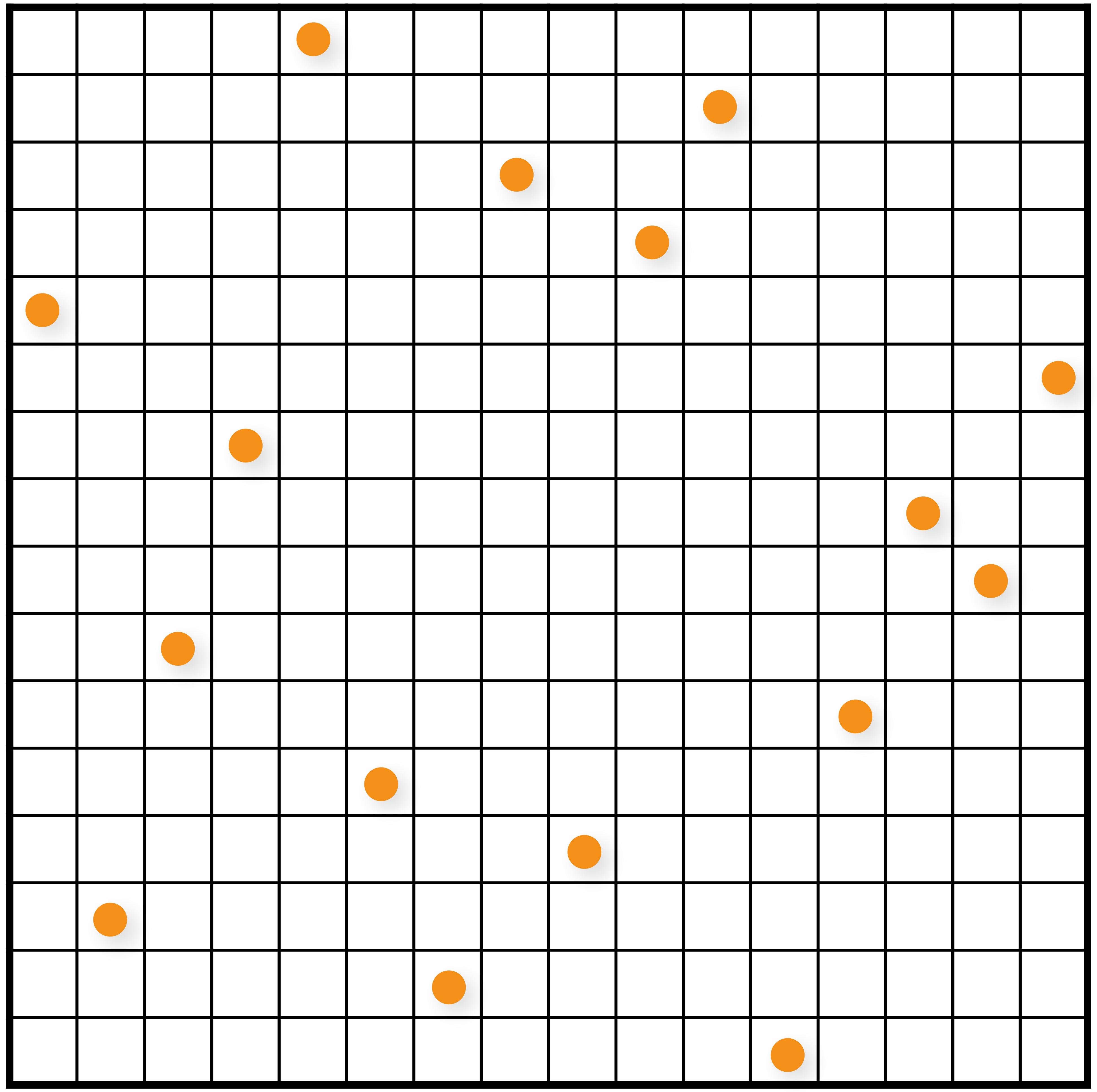
Latin Hypercube (N-Rooks) Sampling

```
// 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

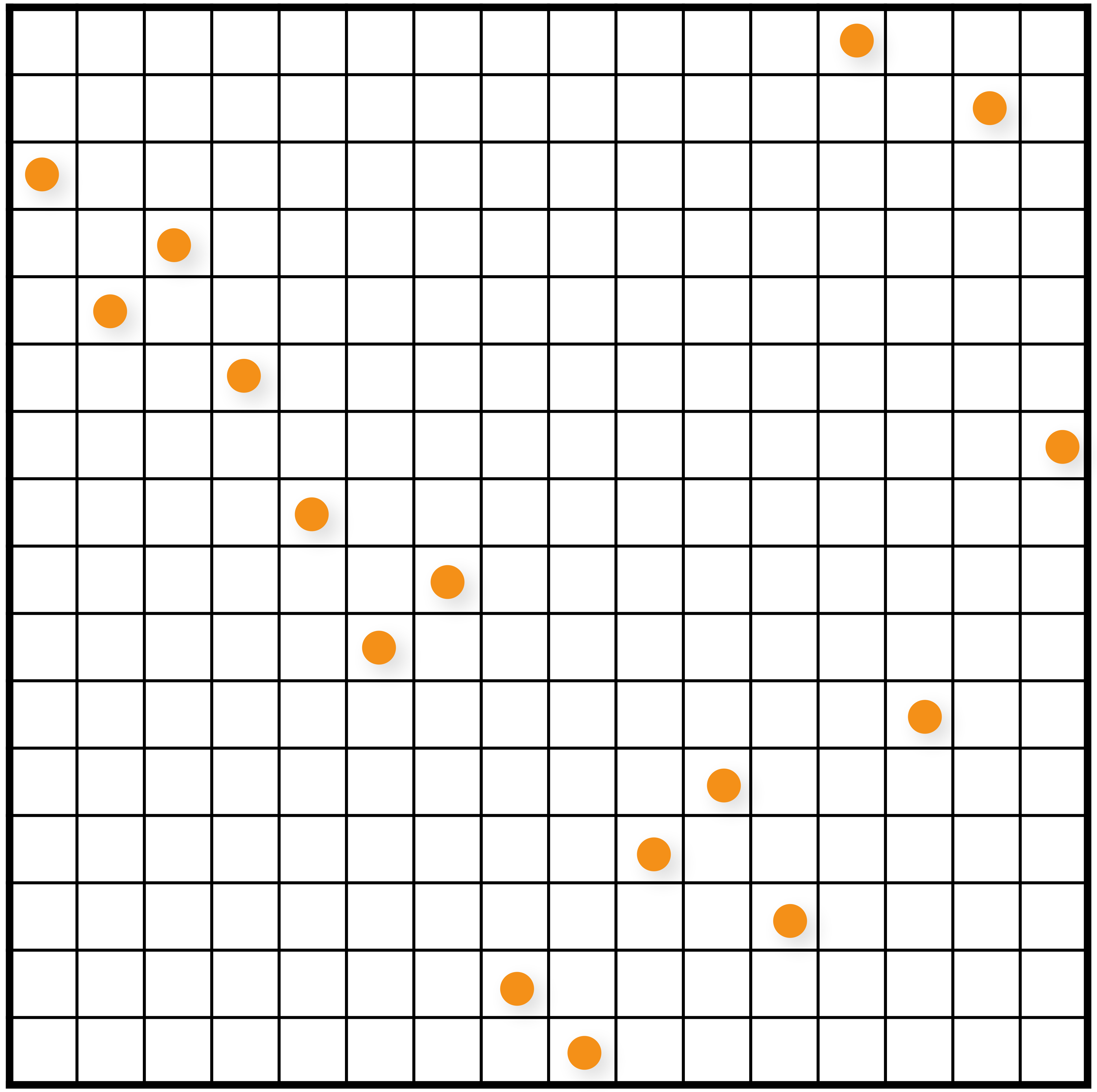
```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
    for (uint i = 0; i < numS; i++)
        samples(d,i) = (i + randf())/numS;

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



Shuffle columns

Latin Hypercube (N-Rooks) Sampling

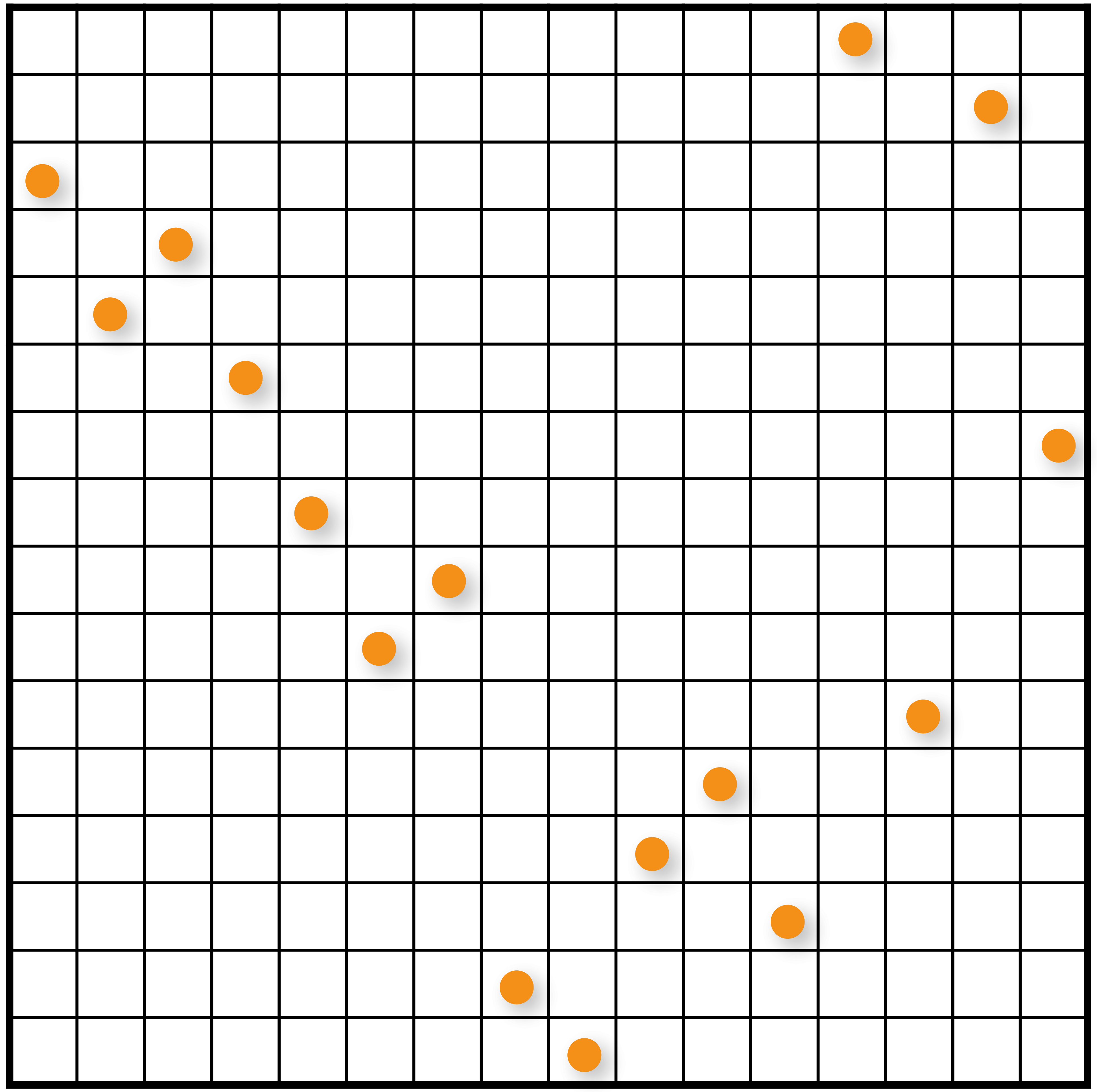
```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
    for (uint i = 0; i < numS; i++)
        samples(d,i) = (i + randf())/numS;

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



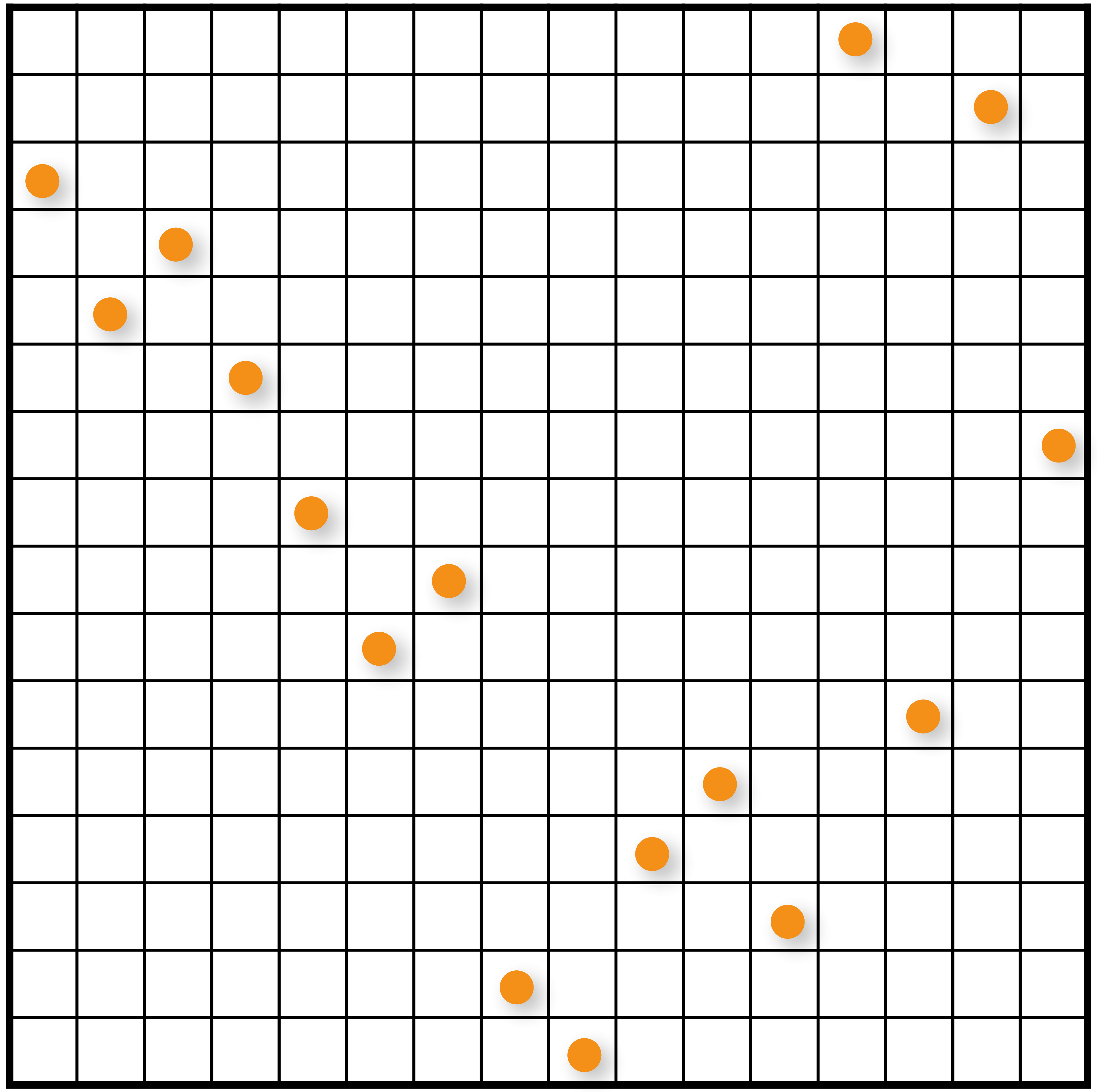
Shuffle columns

Latin Hypercube (N-Rooks) Sampling

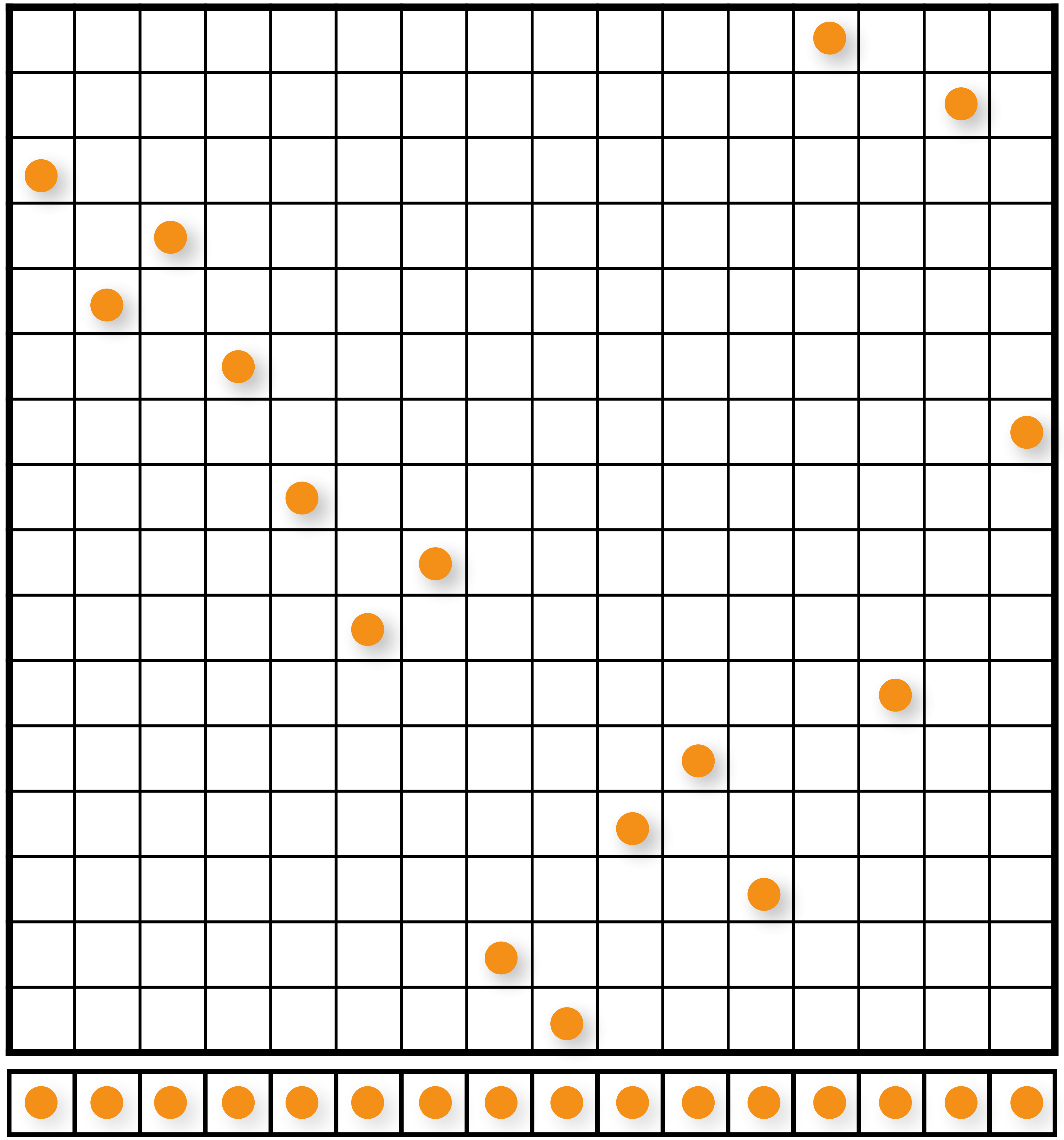
```
// initialize the diagonal
for (uint d = 0; d < numDimensions; d++)
    for (uint i = 0; i < numS; i++)
        samples(d,i) = (i + randf())/numS;

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



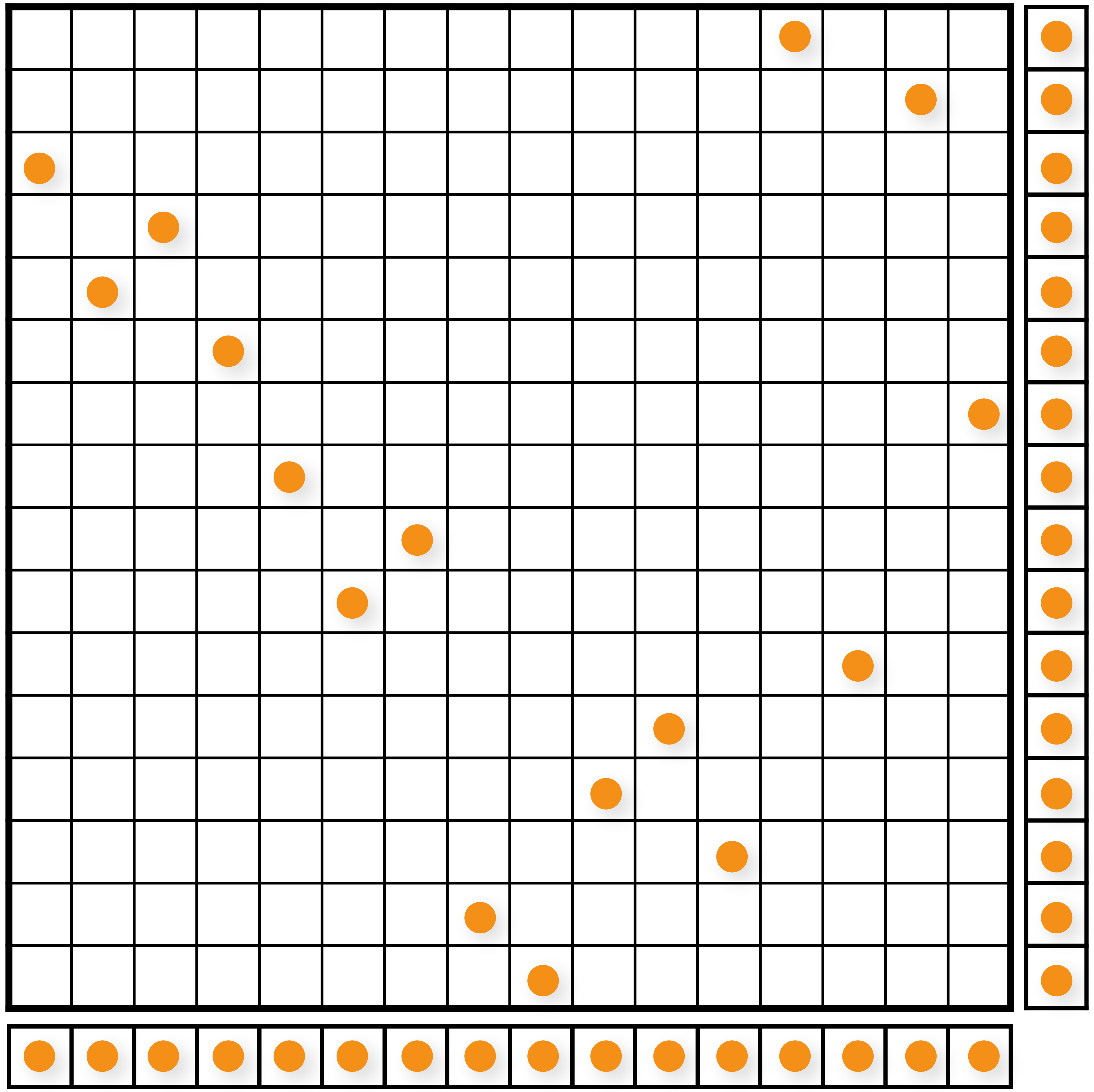
Latin Hypercube (N-Rooks) Sampling



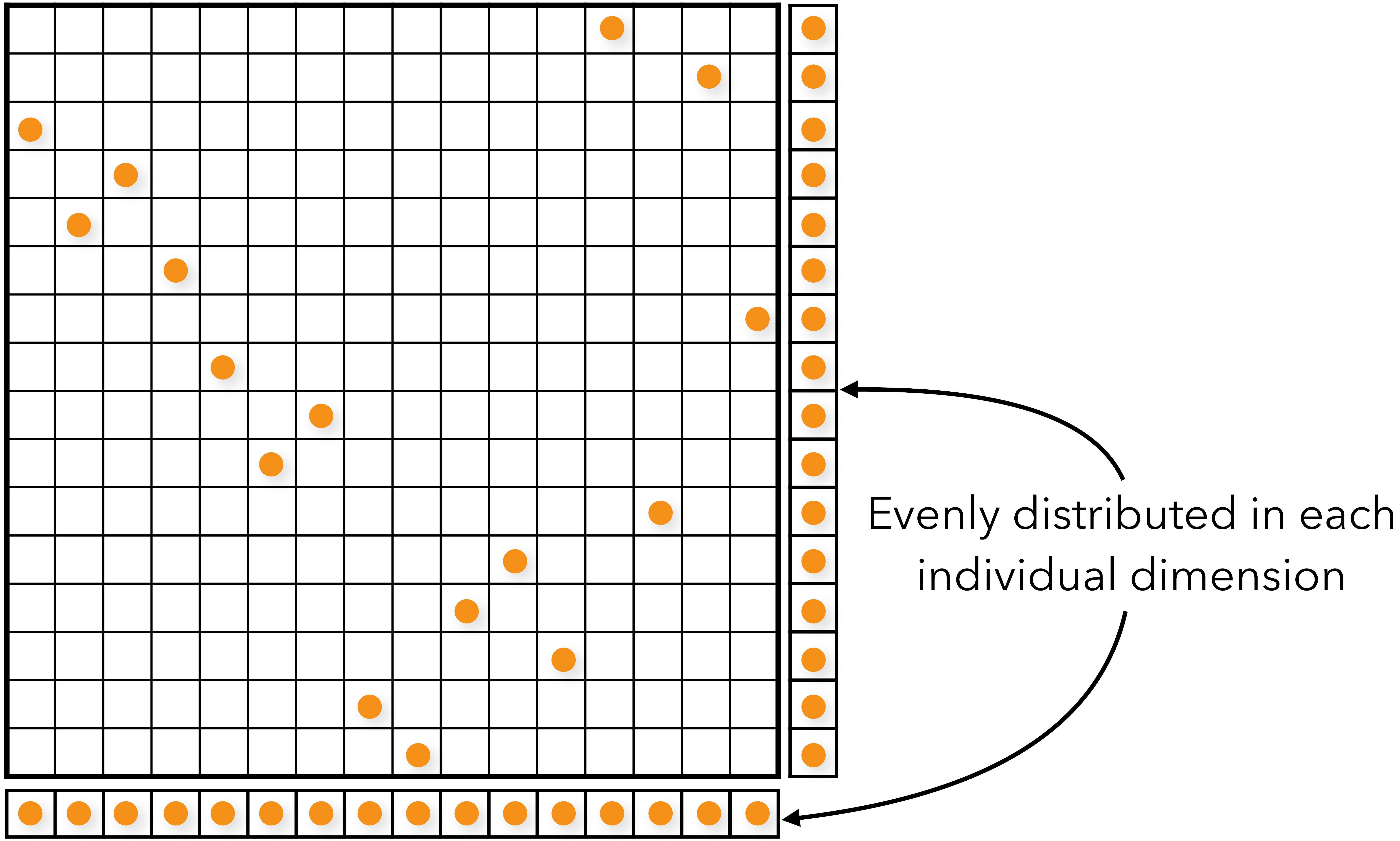
Latin Hypercube (N-Rooks) Sampling



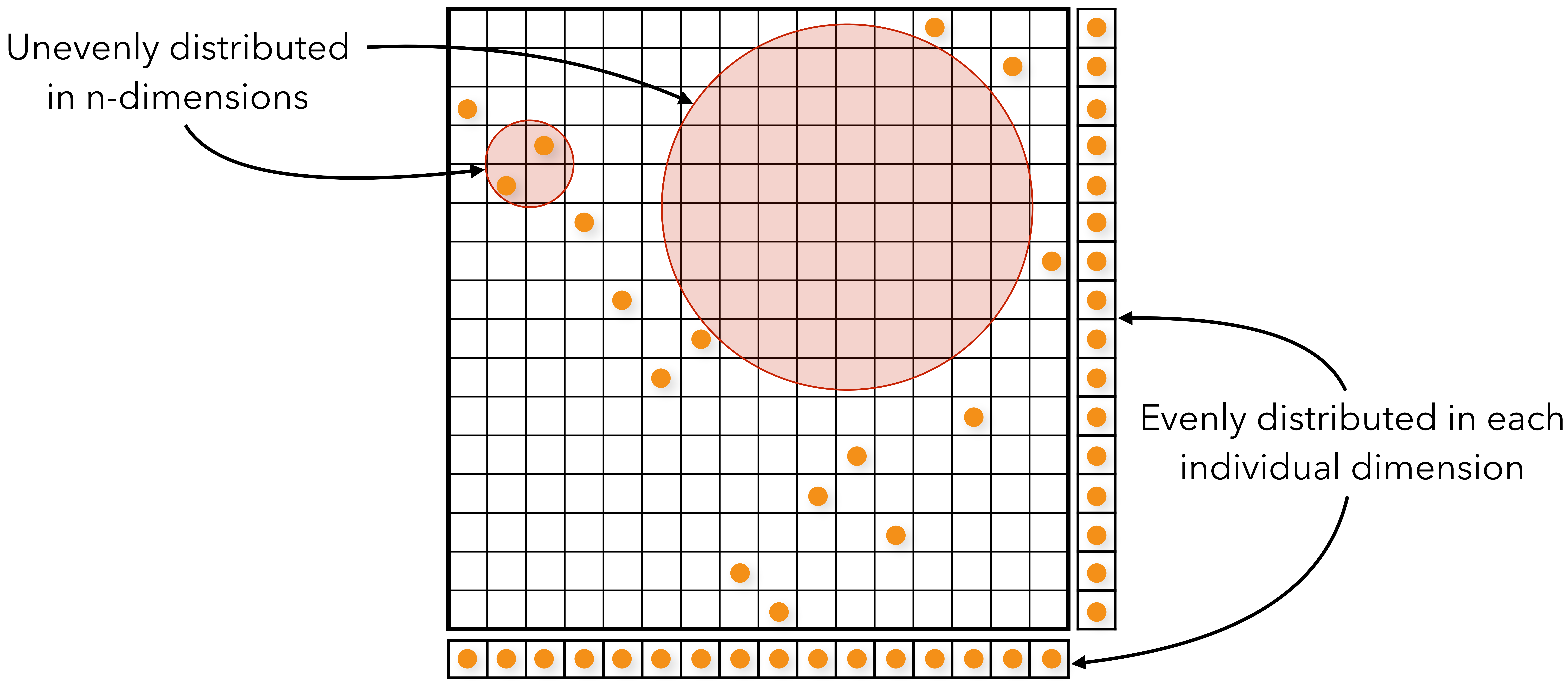
Latin Hypercube (N-Rooks) Sampling



Latin Hypercube (N-Rooks) Sampling

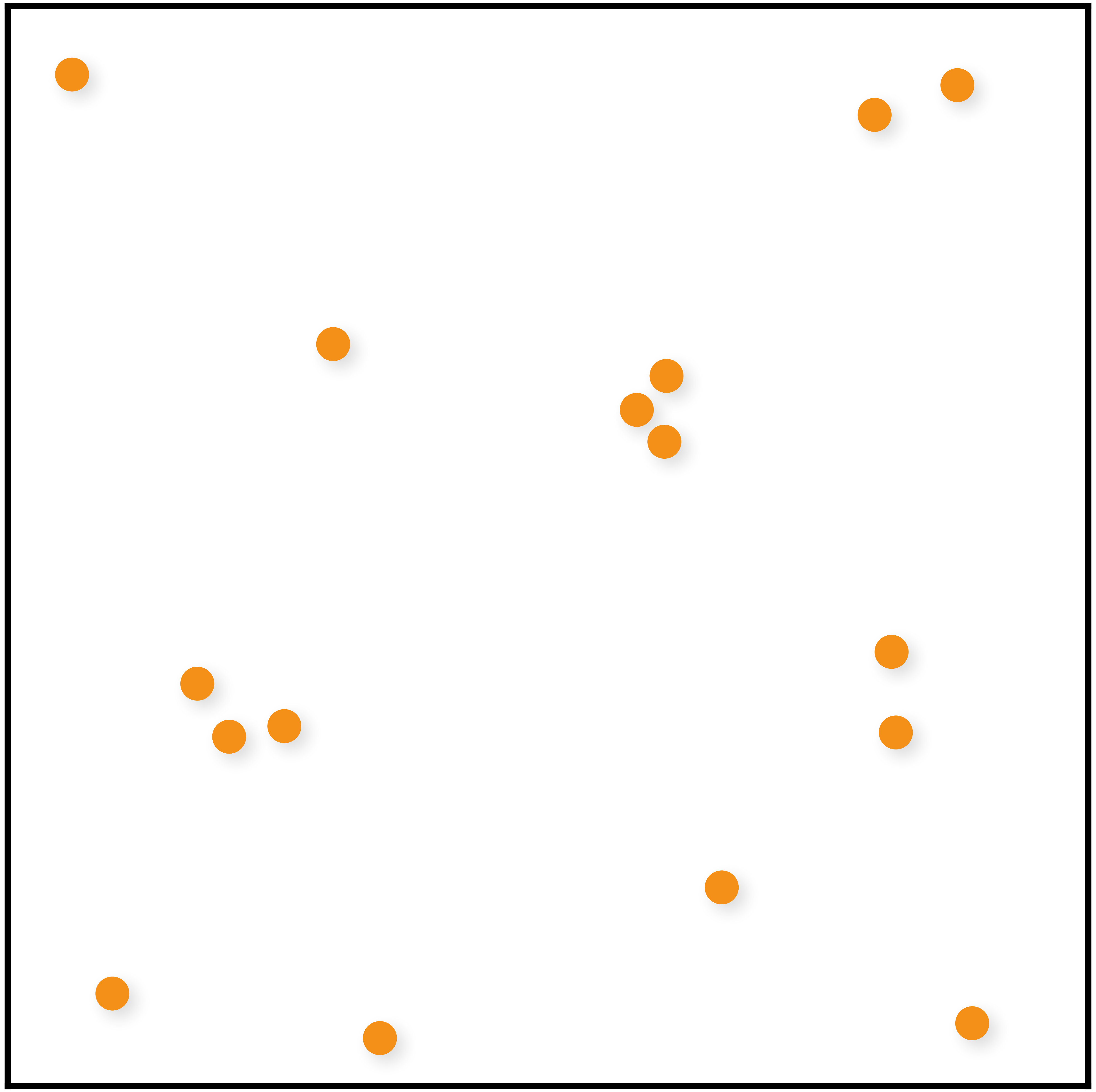


Latin Hypercube (N-Rooks) Sampling

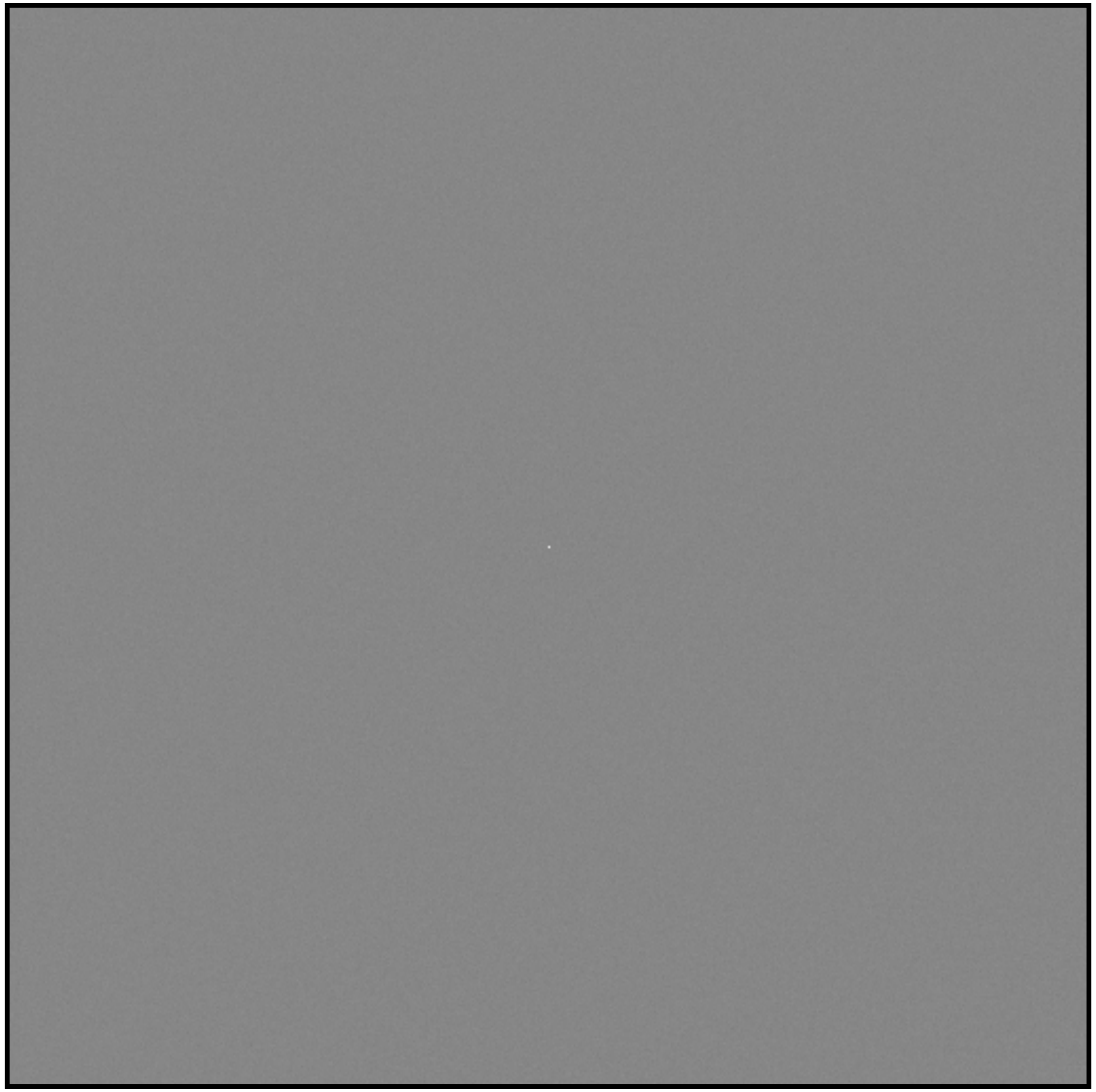


Independent Random Sampling

Spatial domain



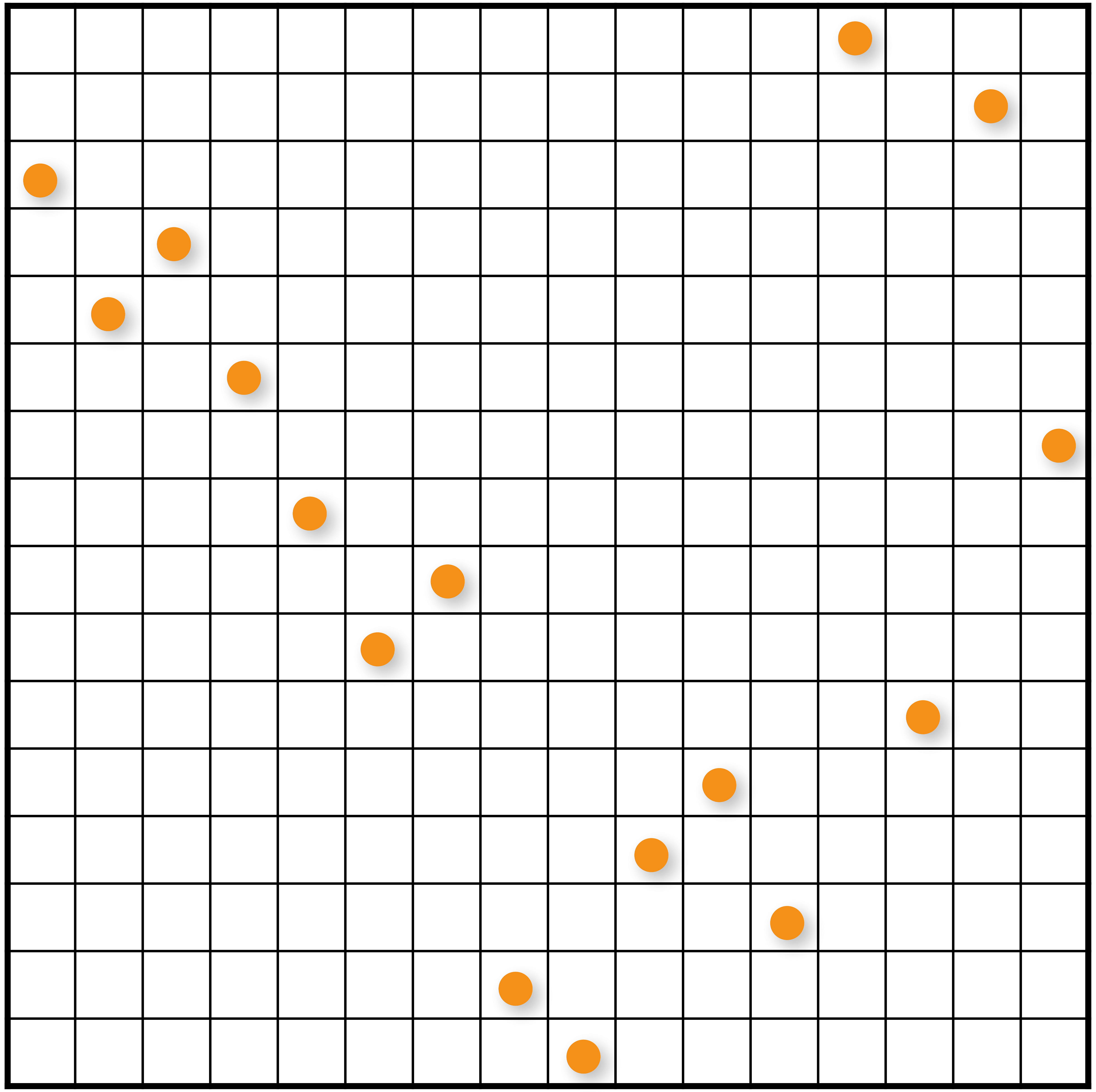
Fourier domain



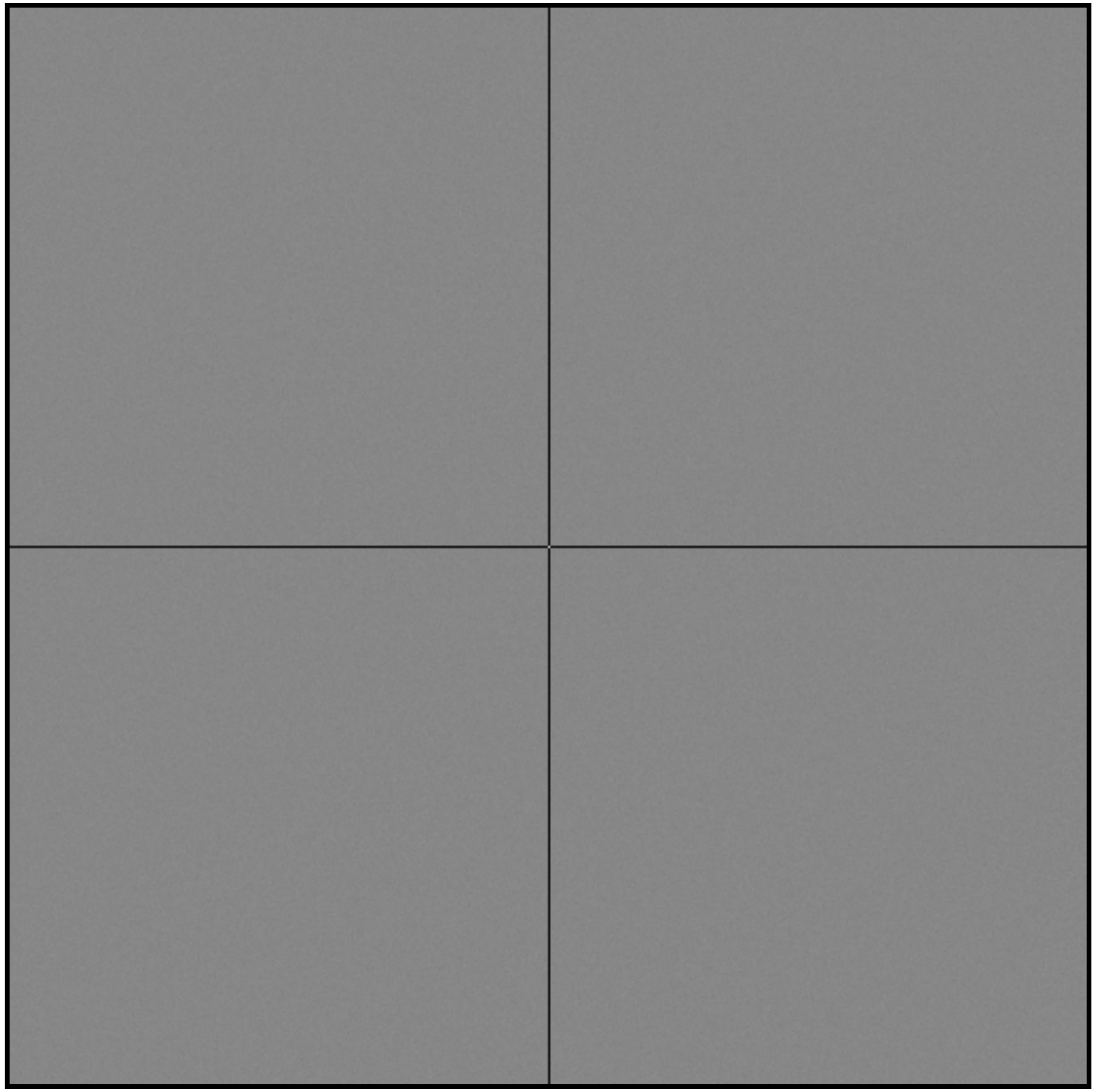
N-Rooks Sampling

[McKay et al. 79]
[Shirley 94]

Spatial domain



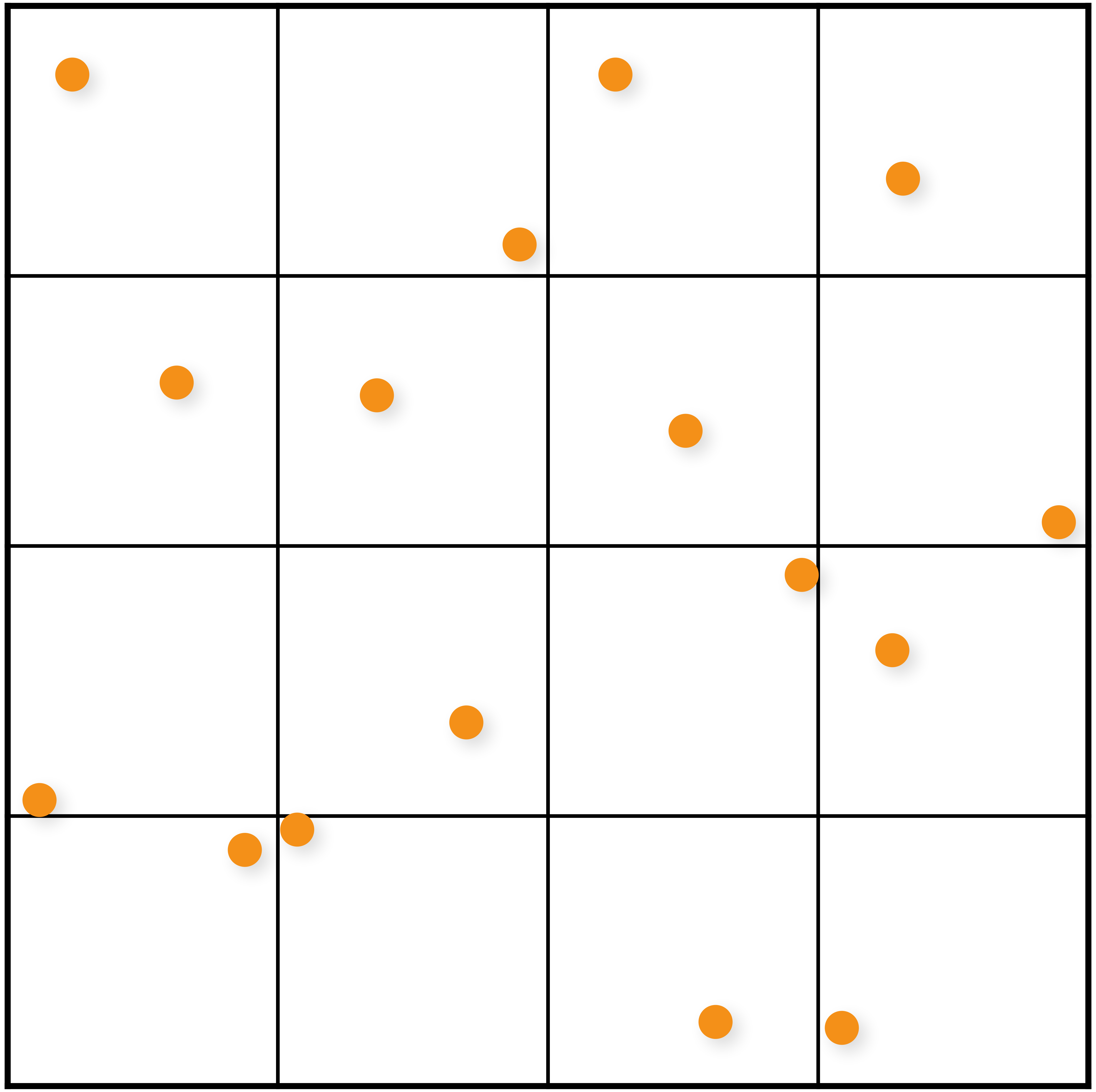
Fourier domain



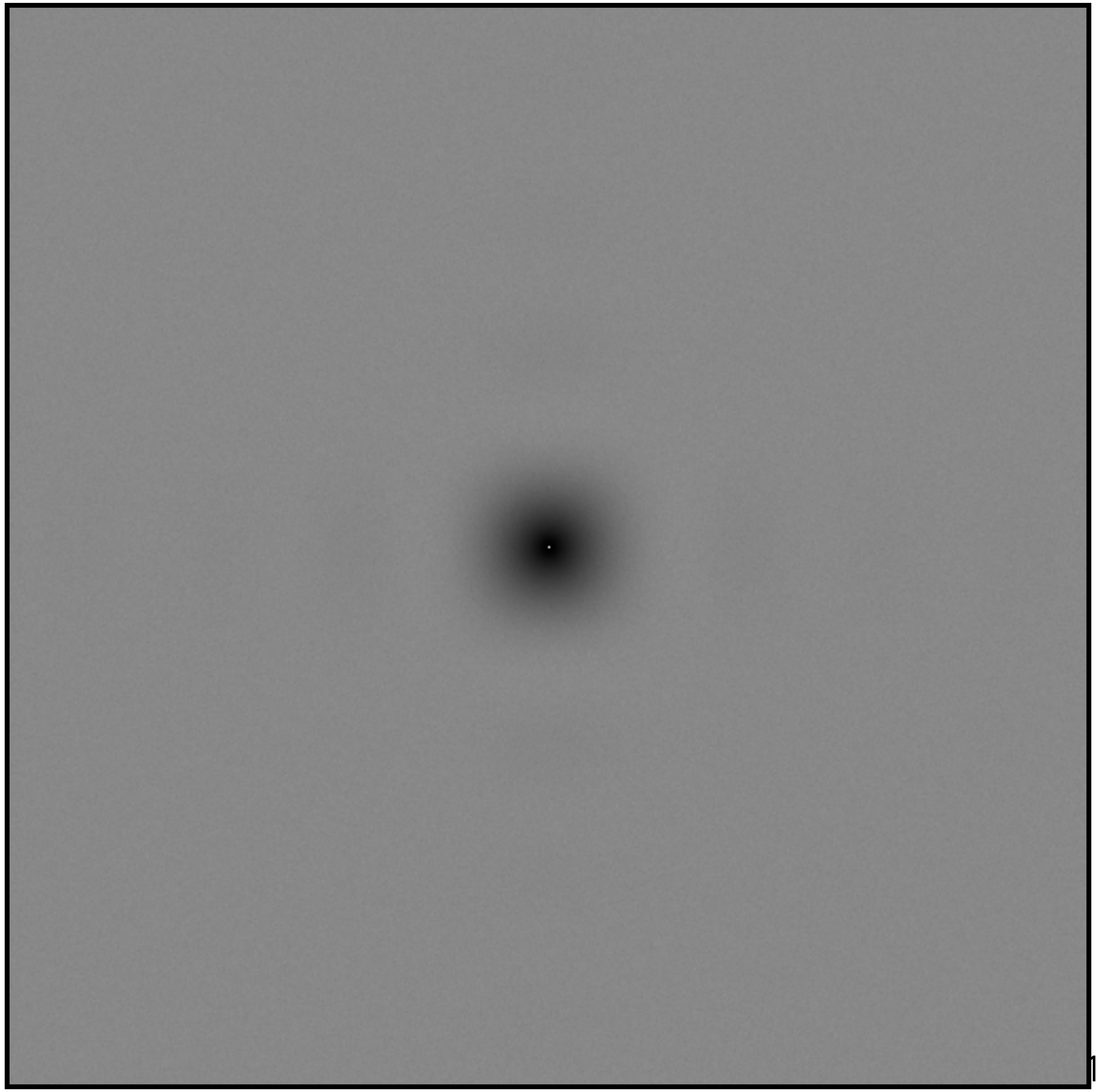
Jittered Sampling

[Cook 86]

Spatial domain

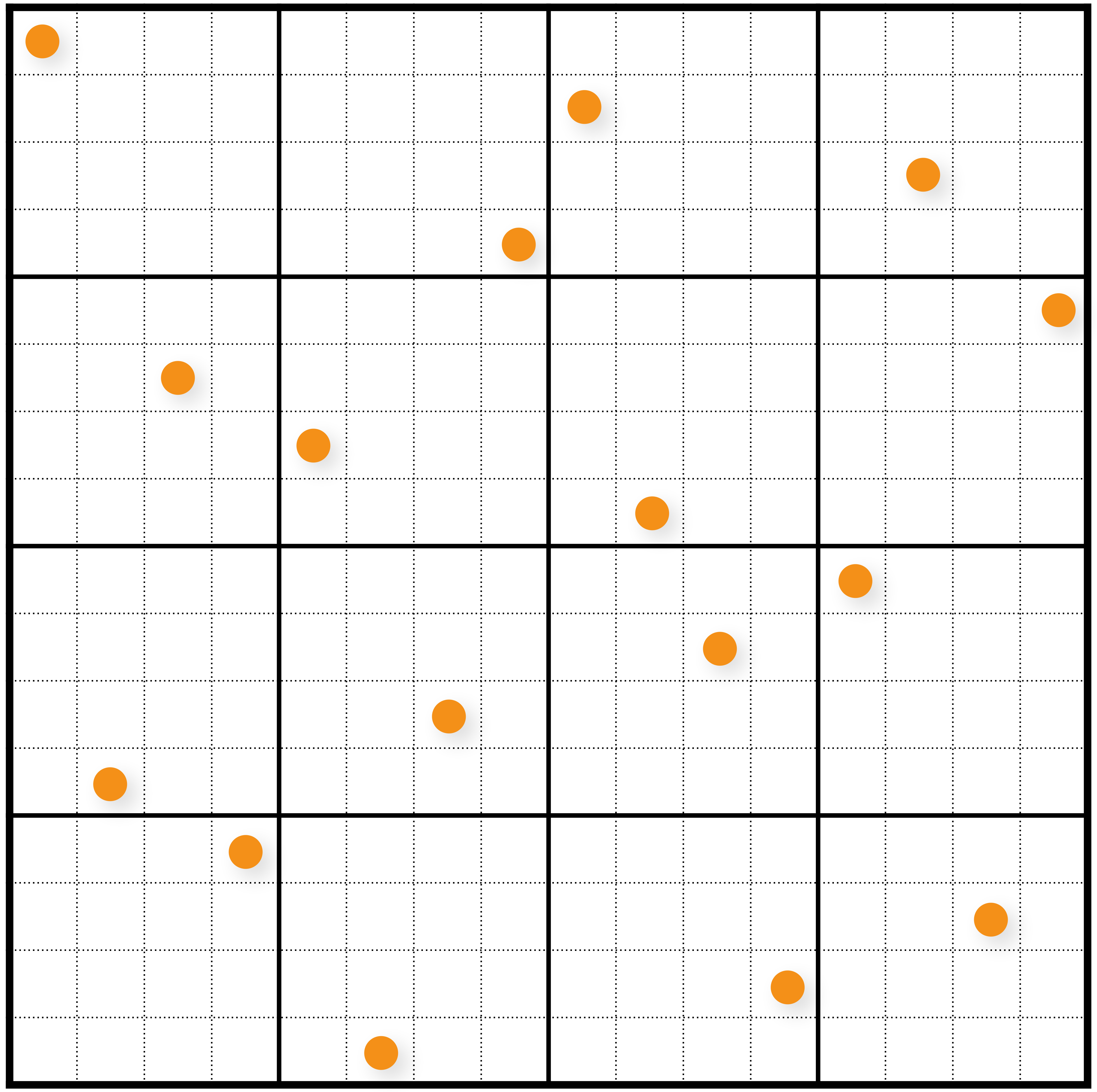


Fourier domain

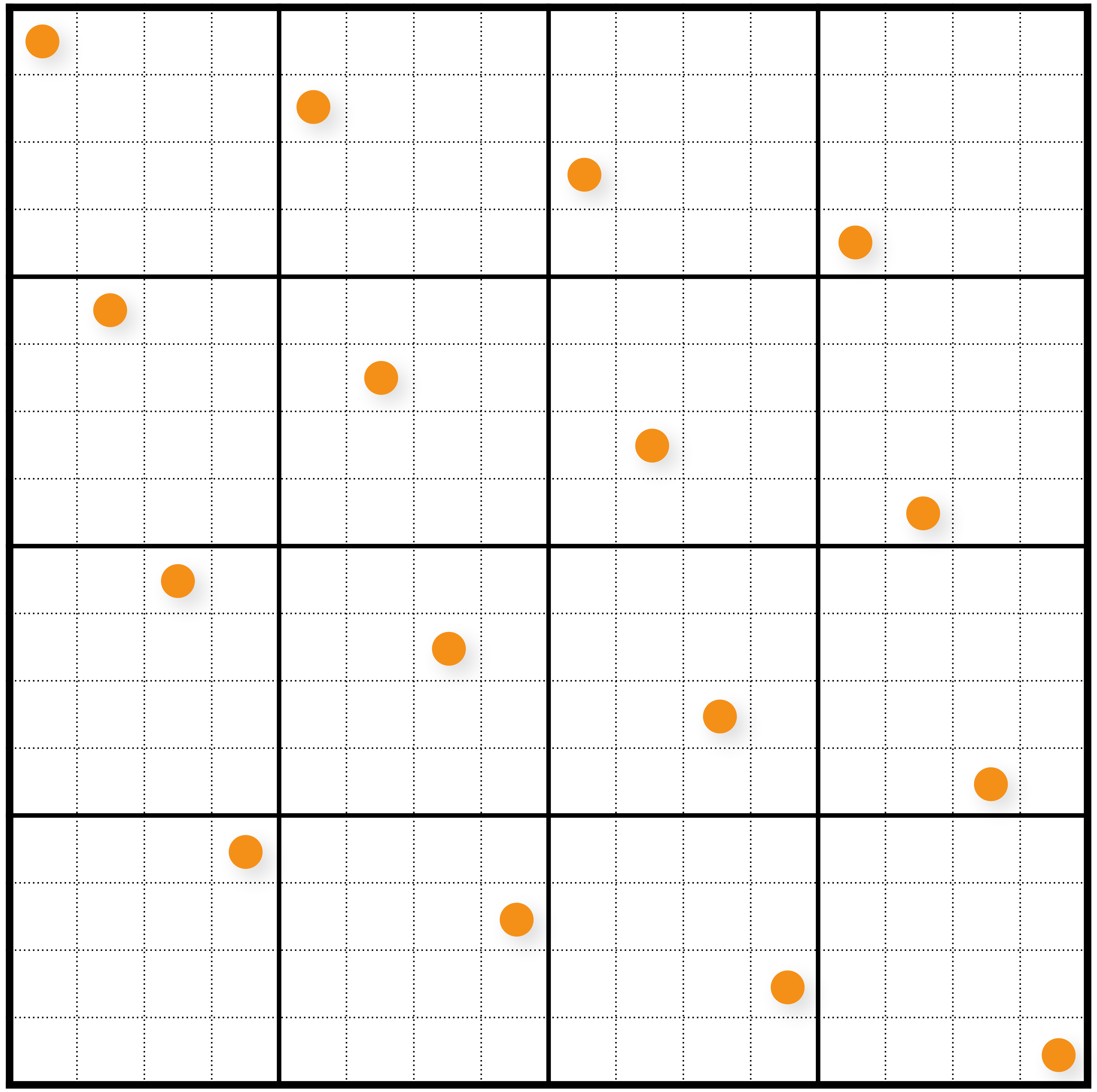


Multi-Jittered Sampling

[Chiu et al. 94]

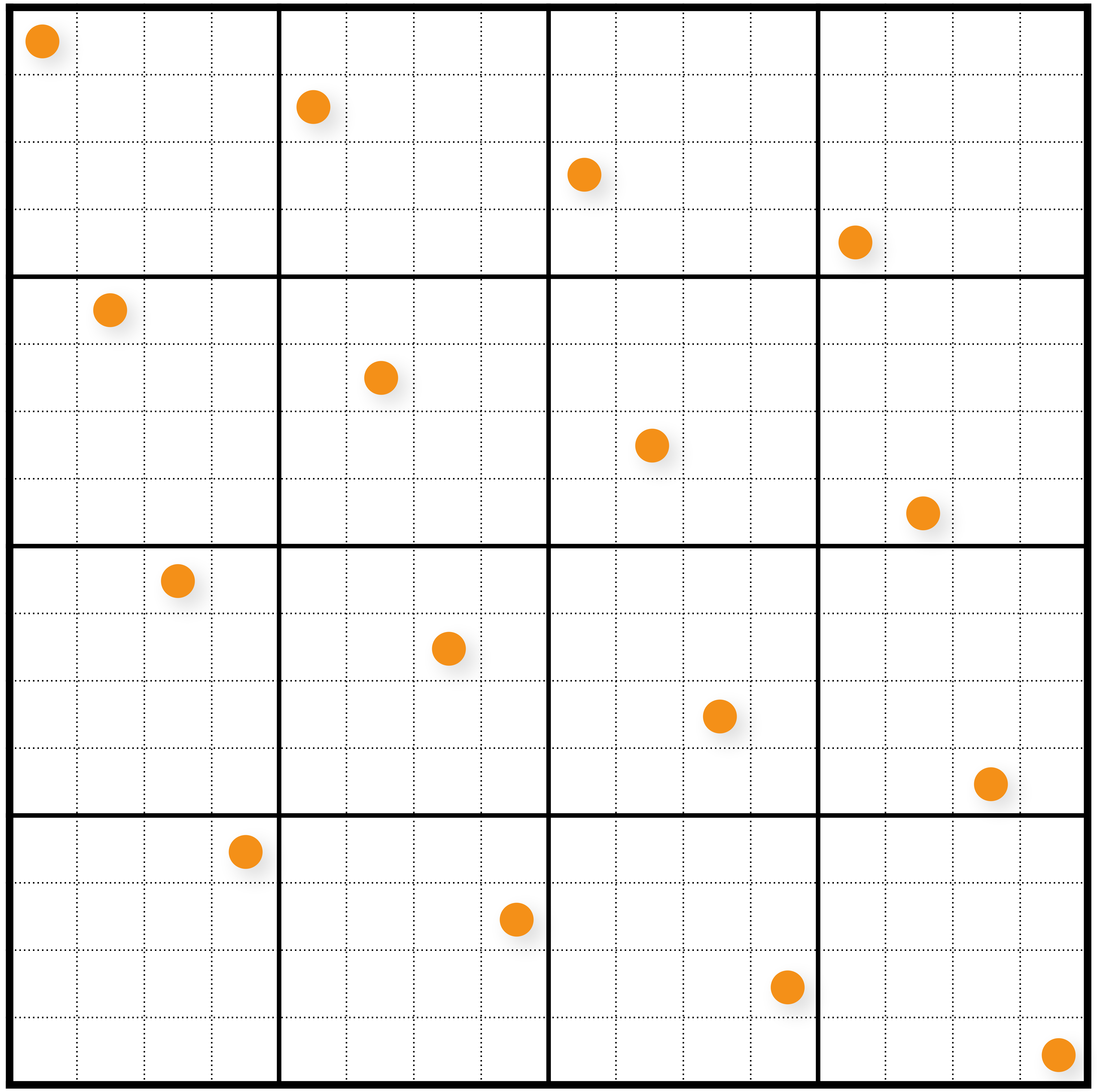


Multi-Jittered Sampling



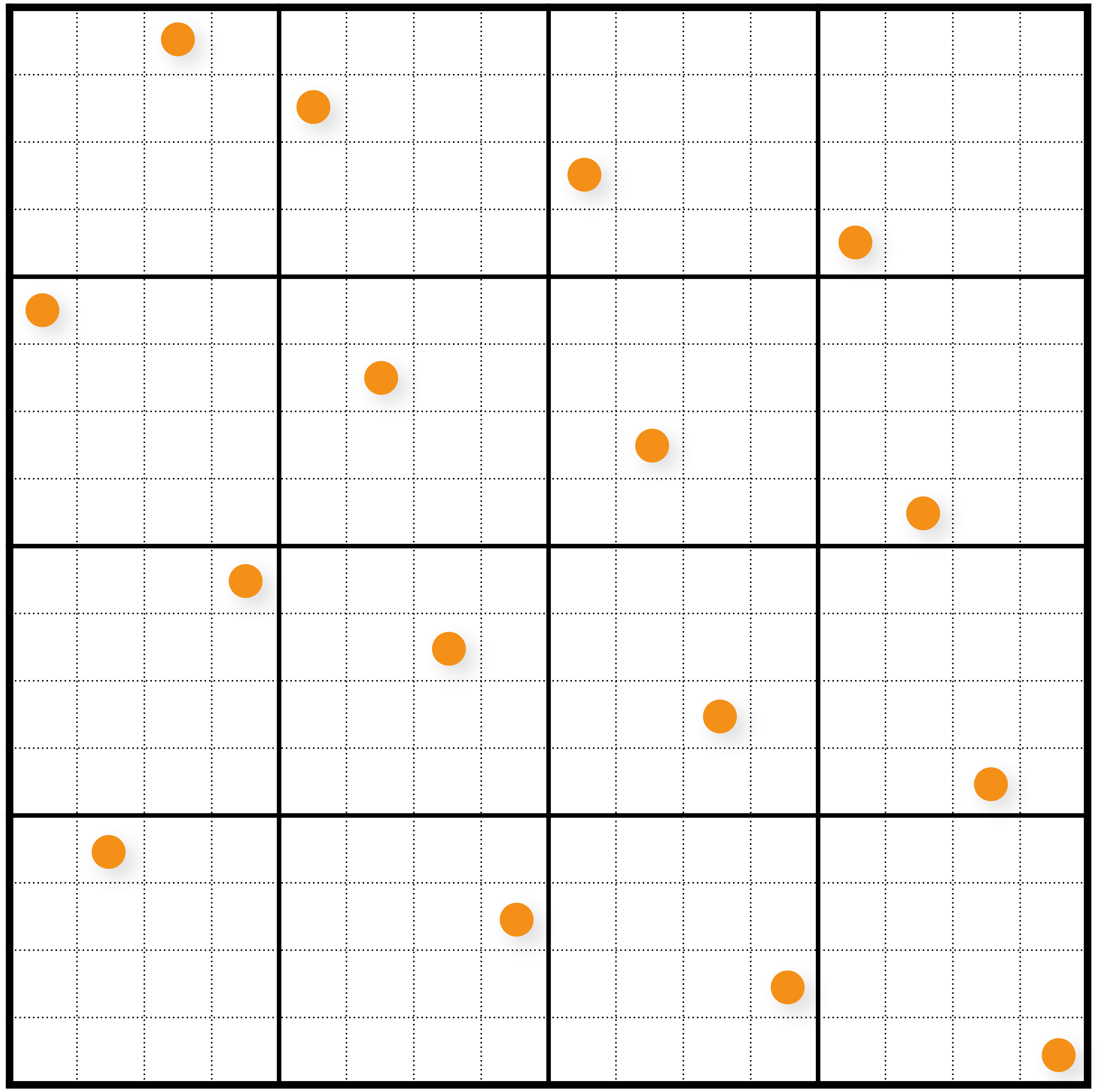
Initialize

Multi-Jittered Sampling



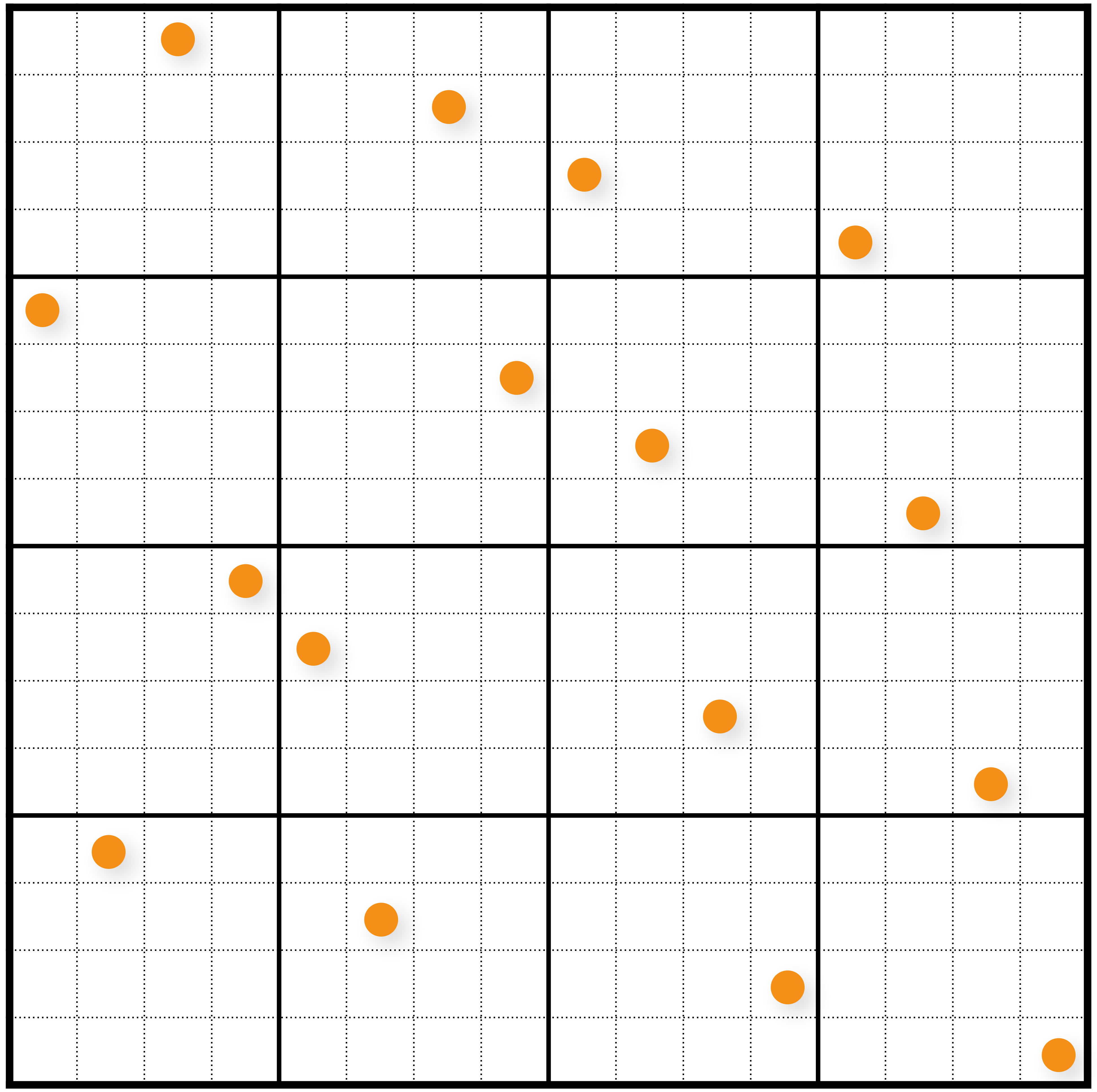
Shuffle x-coords

Multi-Jittered Sampling



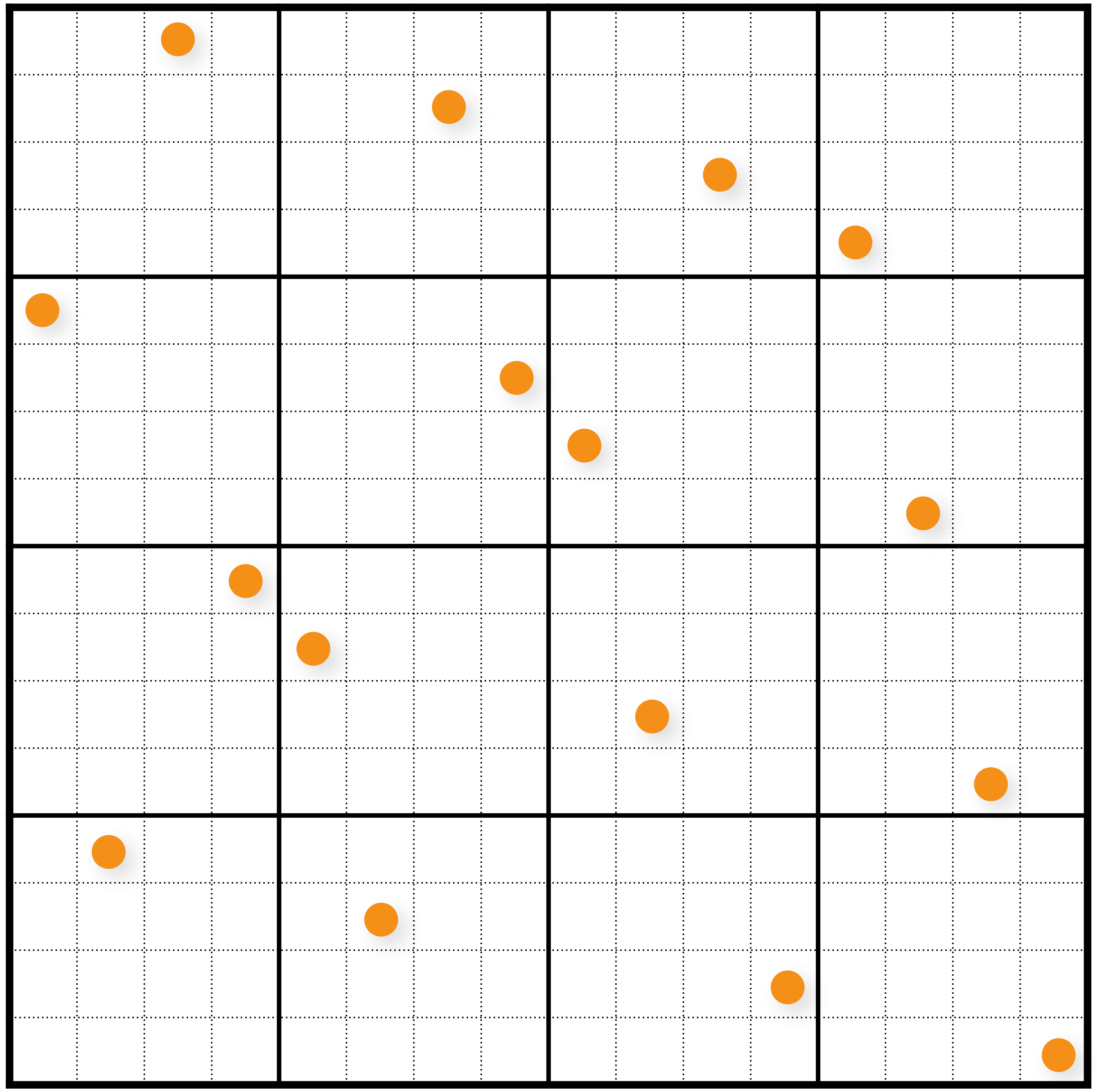
Shuffle x-coords

Multi-Jittered Sampling



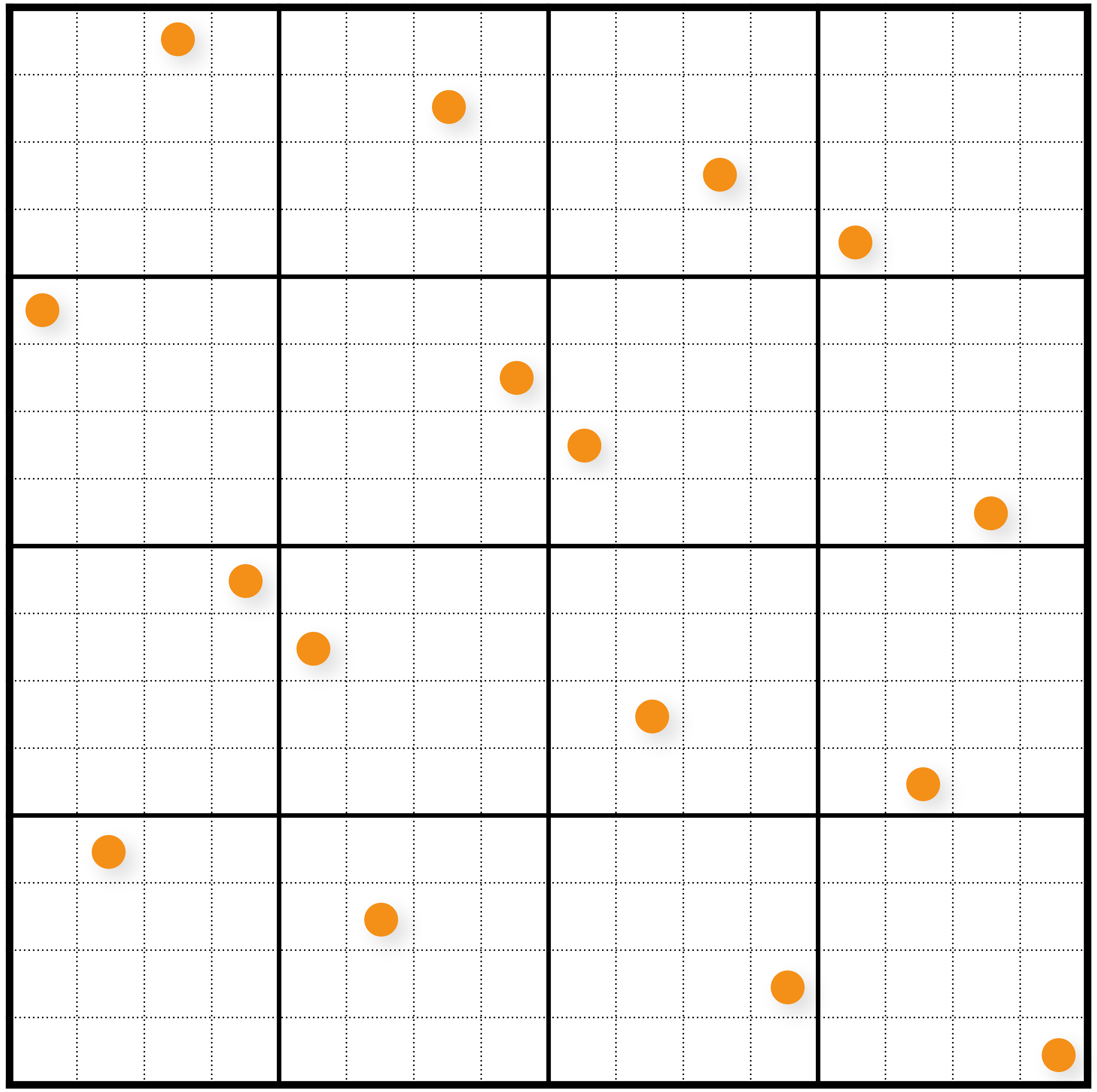
Shuffle x-coords

Multi-Jittered Sampling



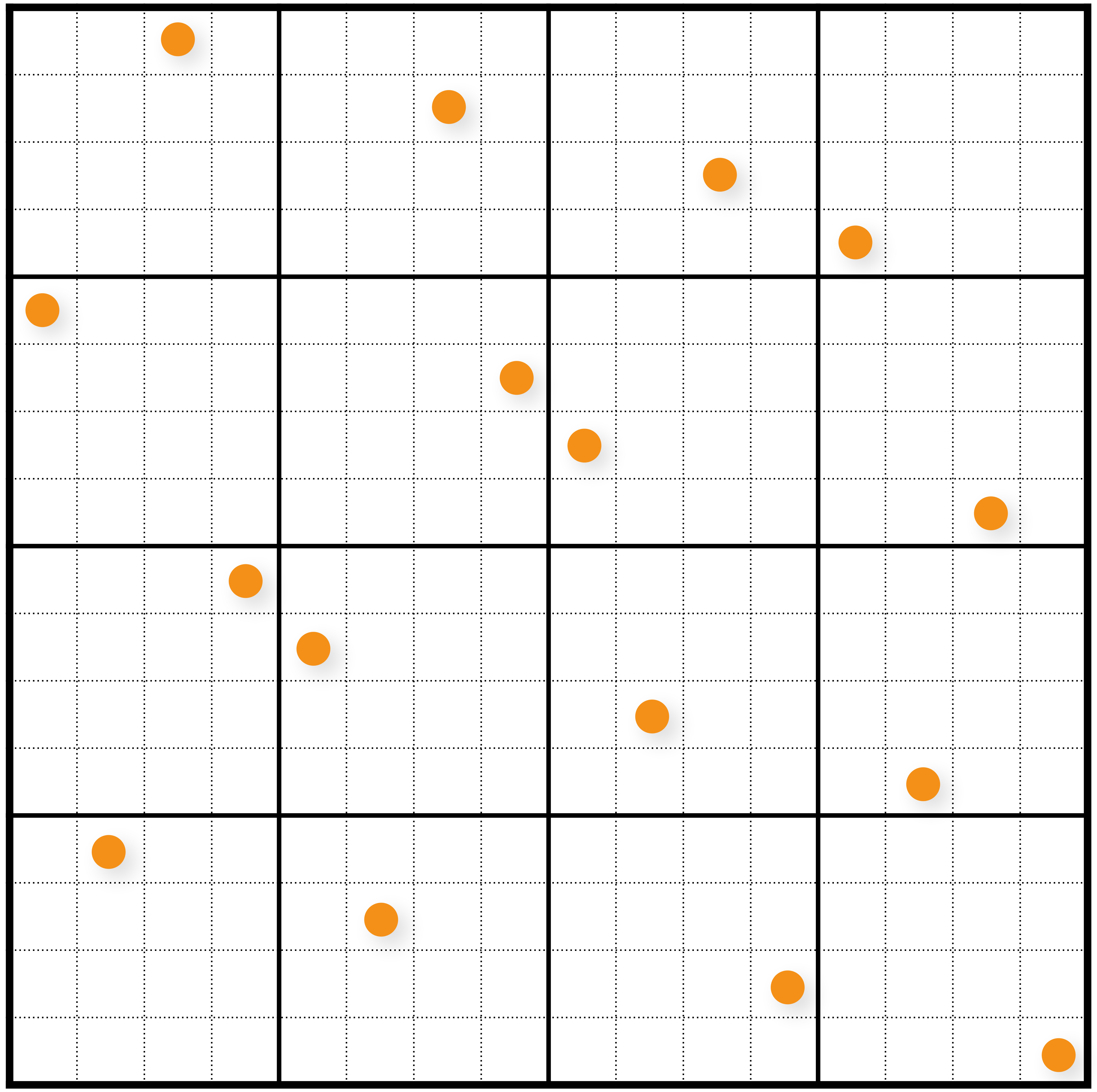
Shuffle x-coords

Multi-Jittered Sampling

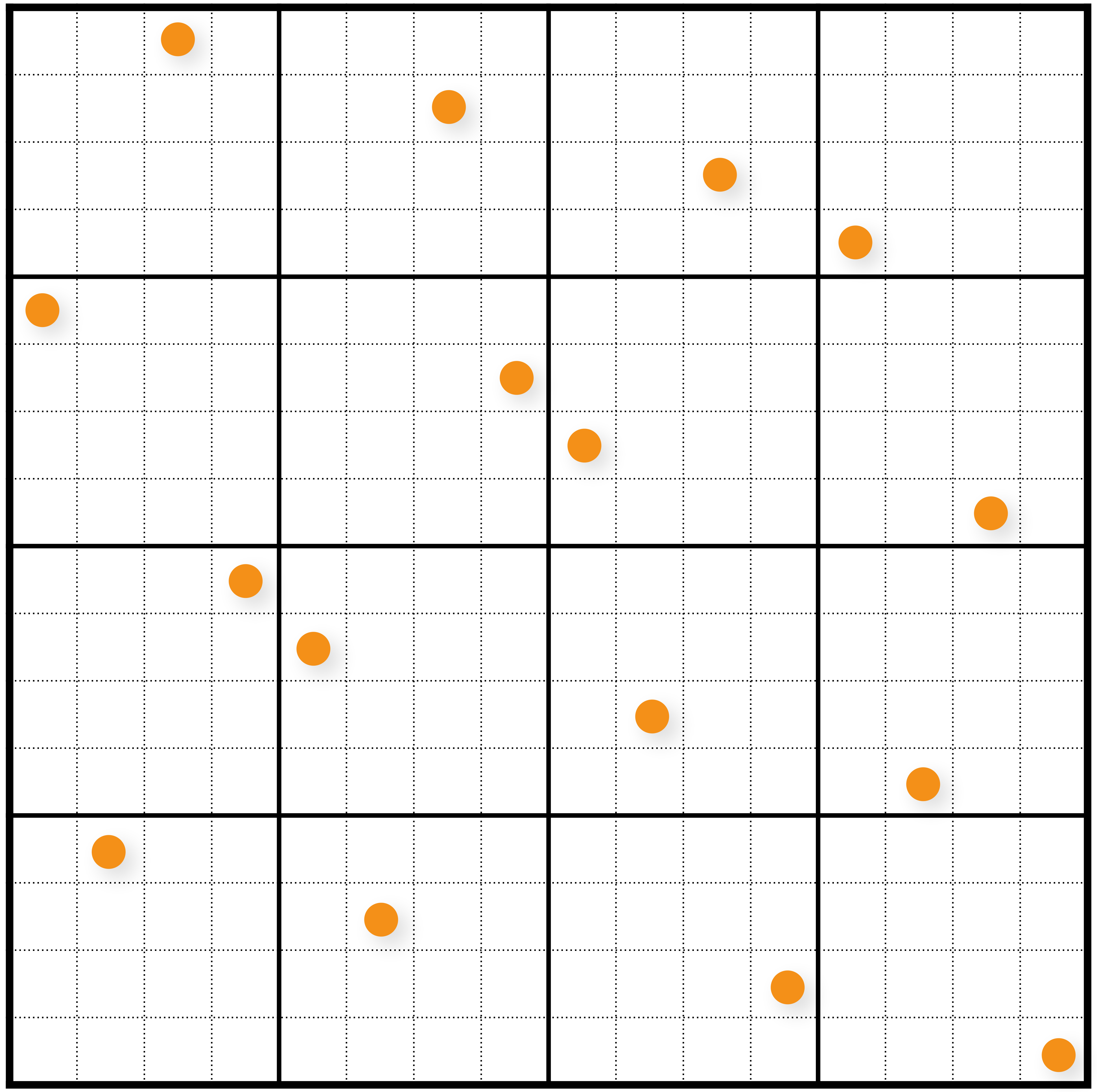


Shuffle x-coords

Multi-Jittered Sampling

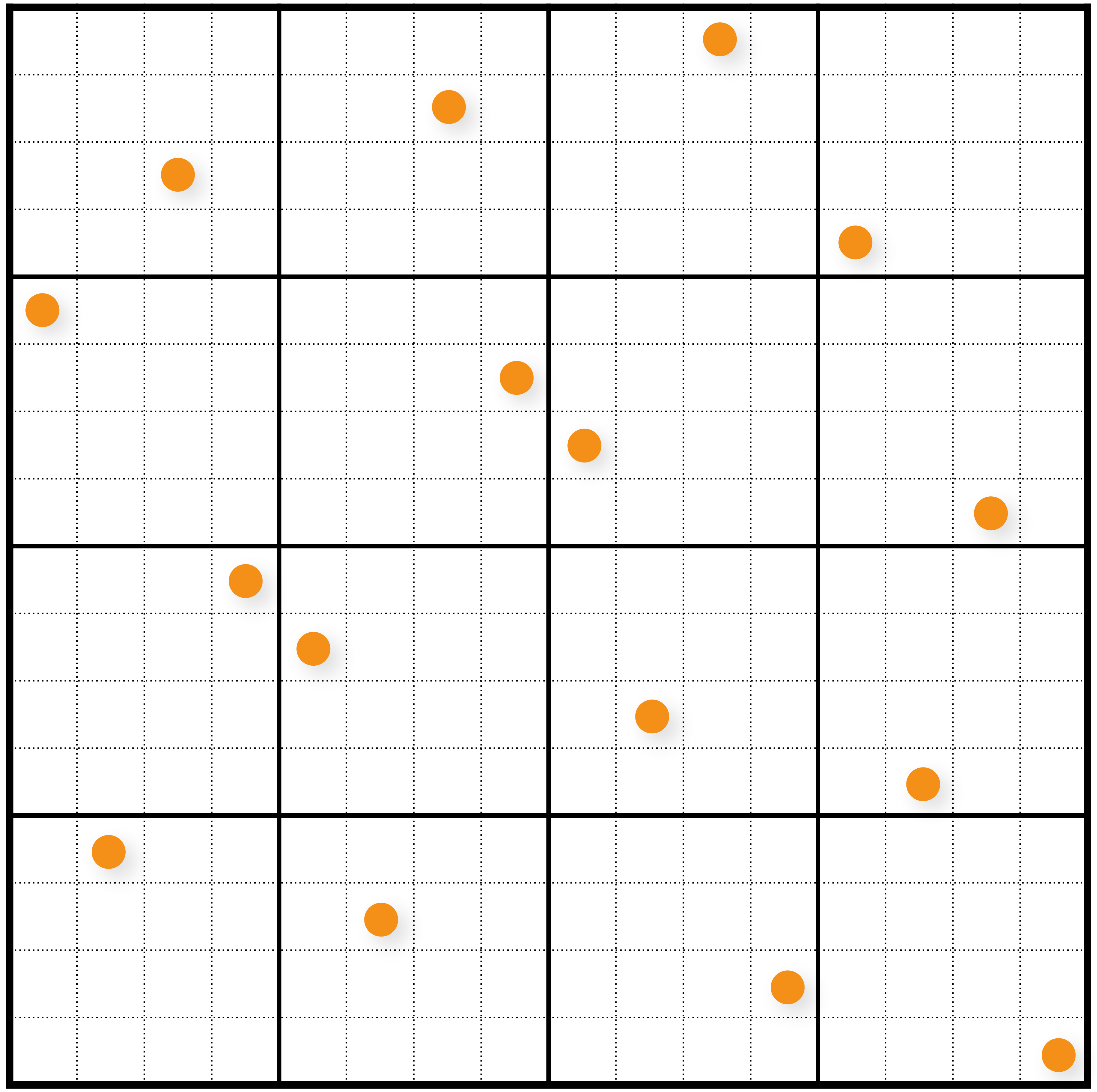


Multi-Jittered Sampling



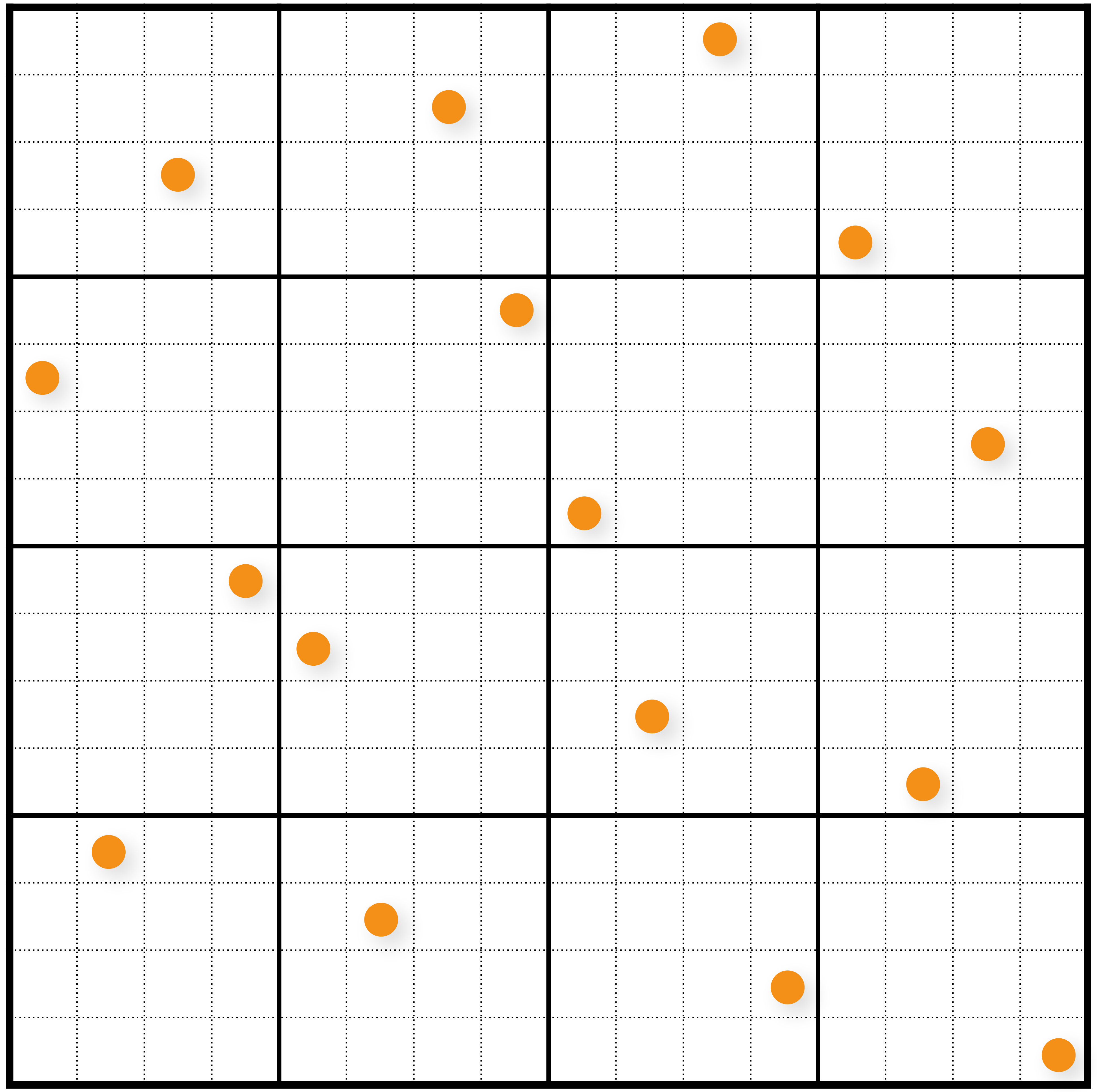
Shuffle y-coords

Multi-Jittered Sampling



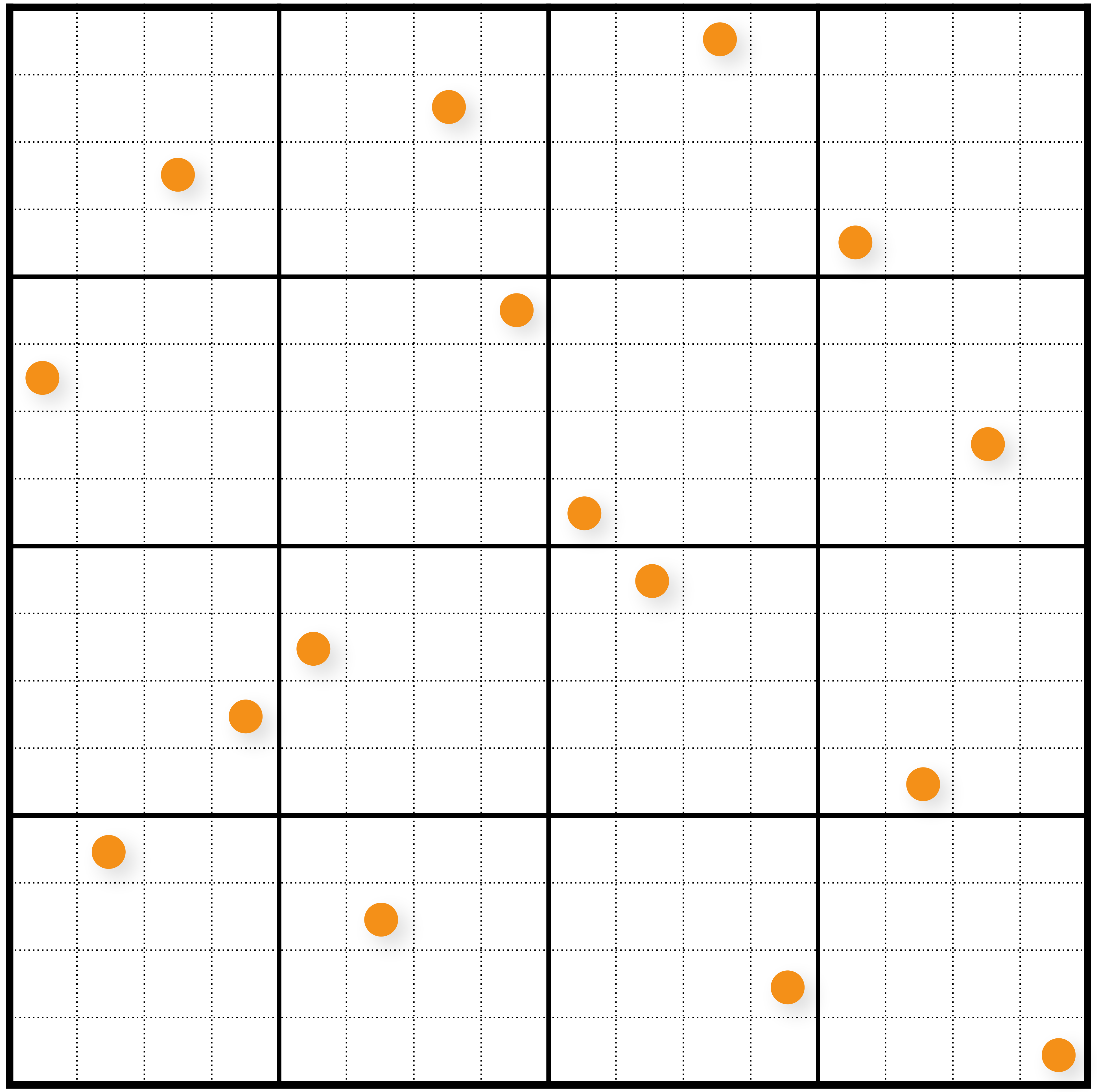
Shuffle y-coords

Multi-Jittered Sampling

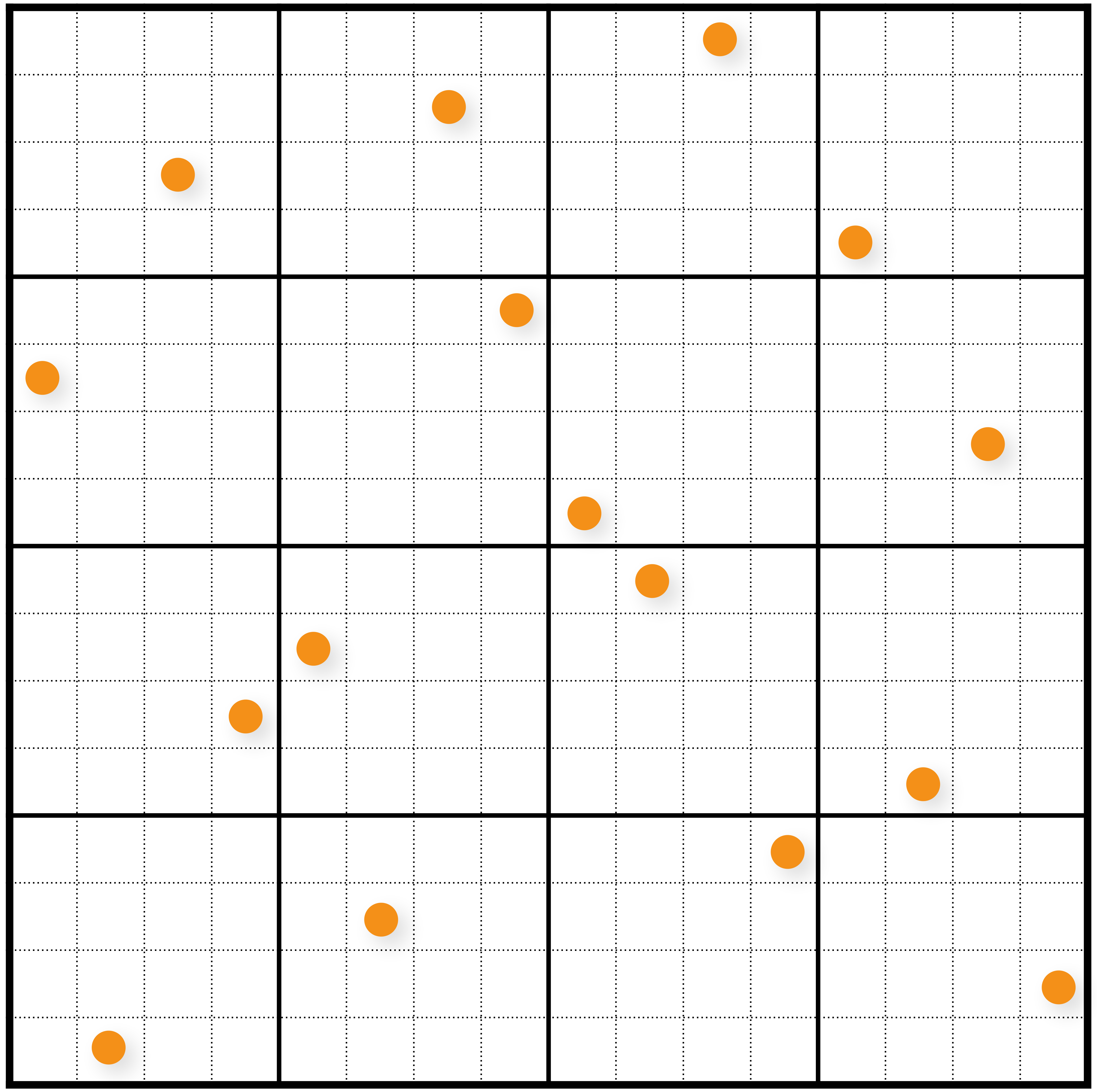


Shuffle y-coords

Multi-Jittered Sampling

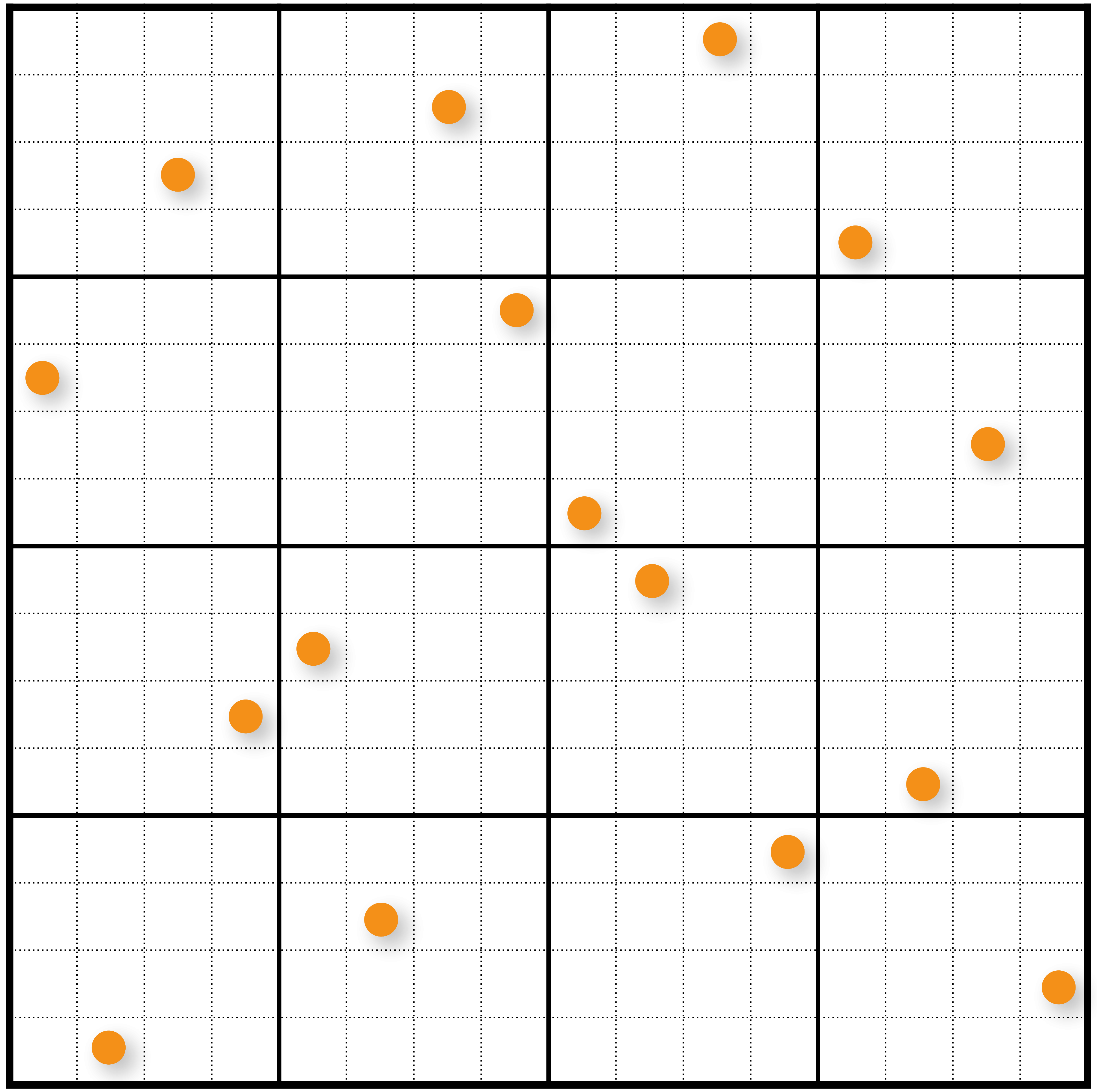

Shuffle y-coords

Multi-Jittered Sampling

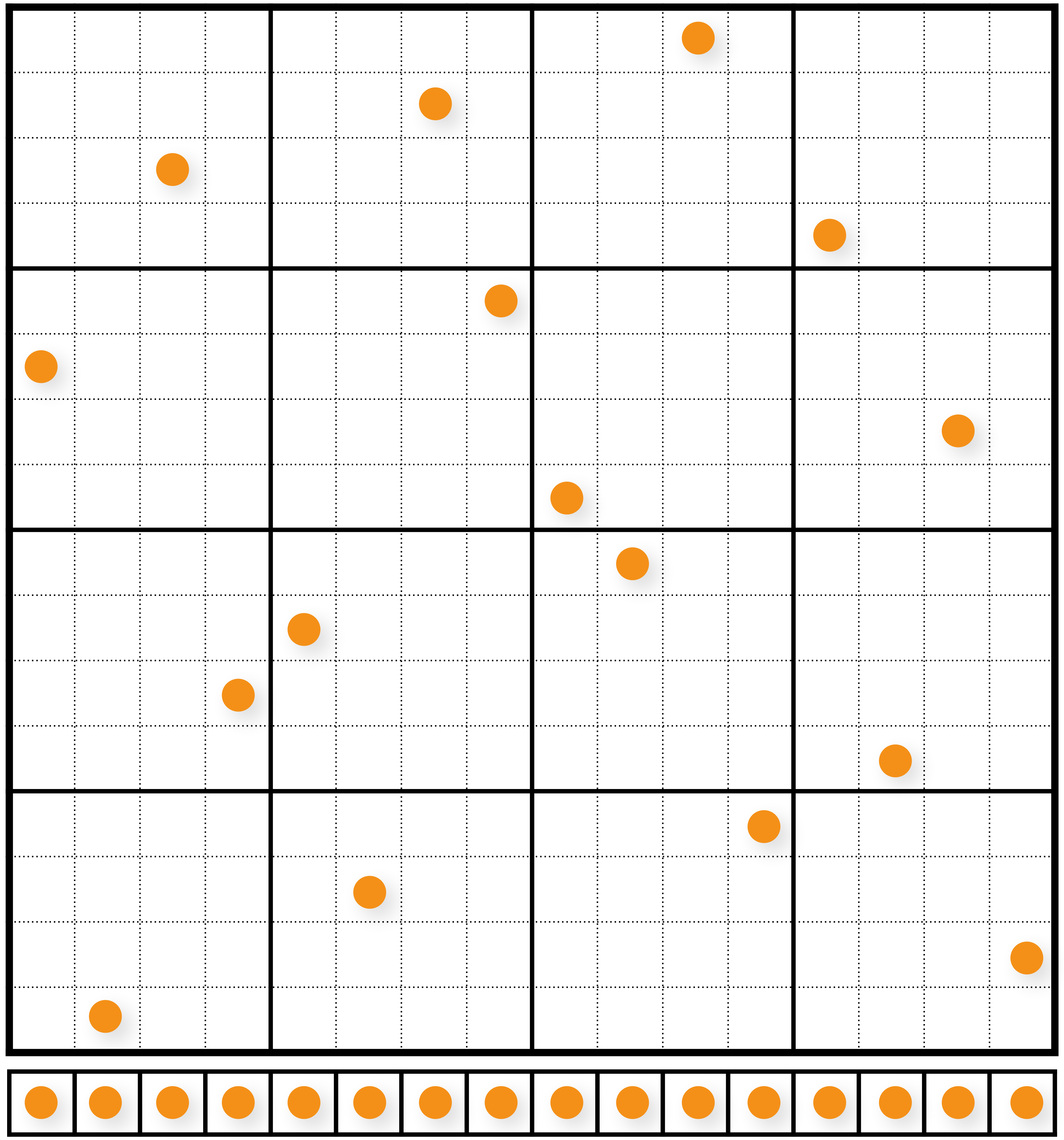


Shuffle y-coords

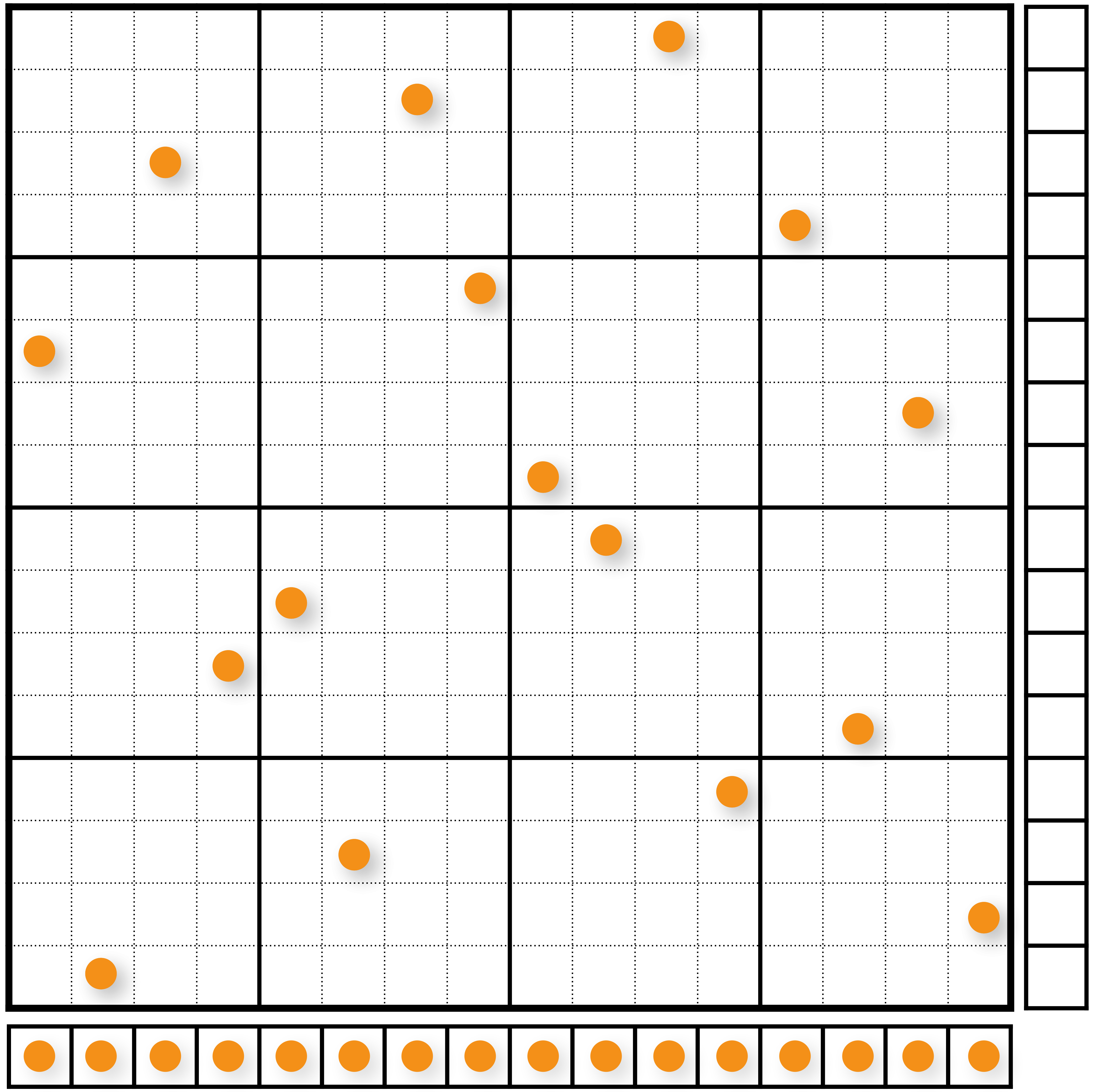
Multi-Jittered Sampling (Projections)



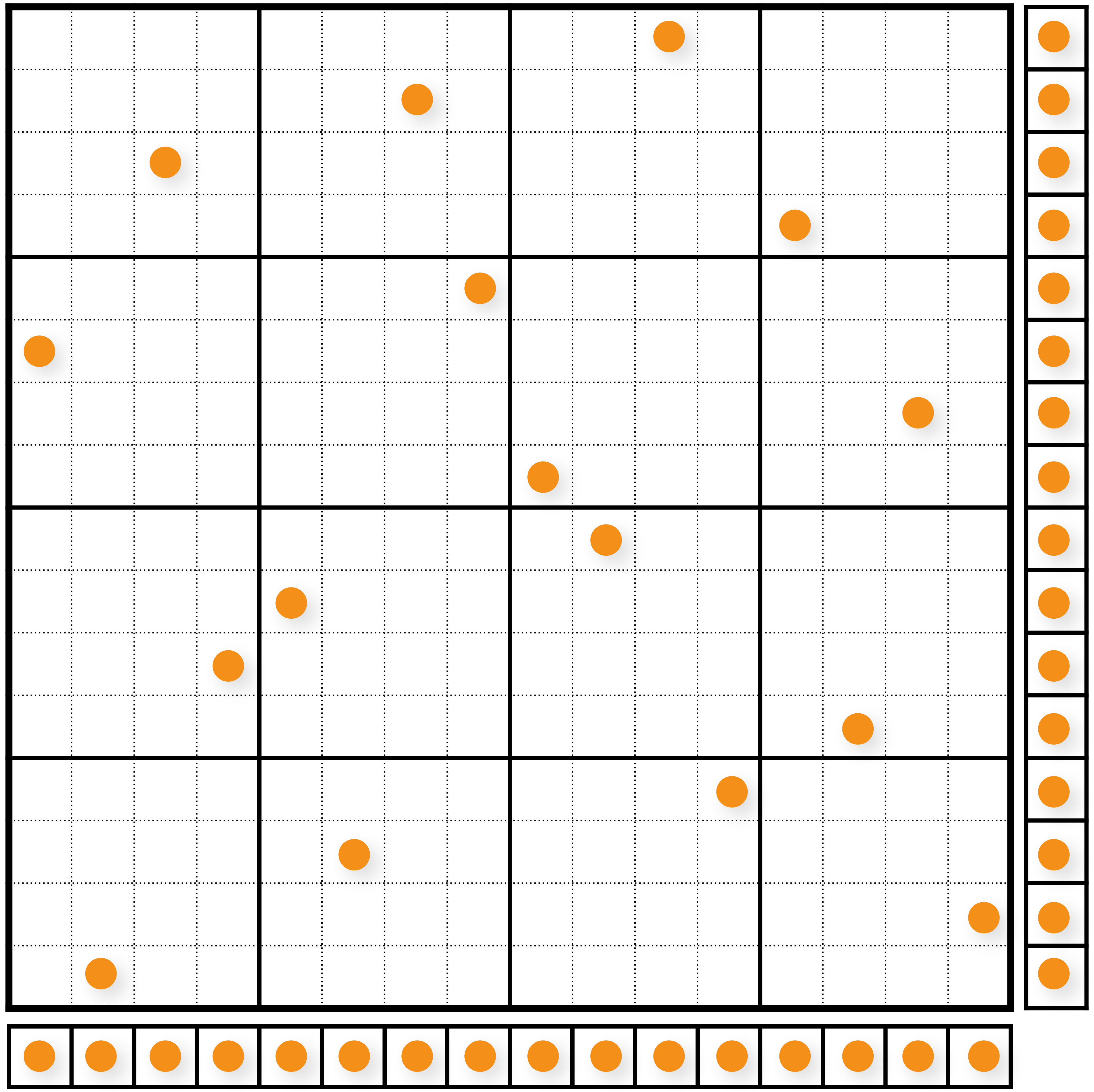
Multi-Jittered Sampling (Projections)



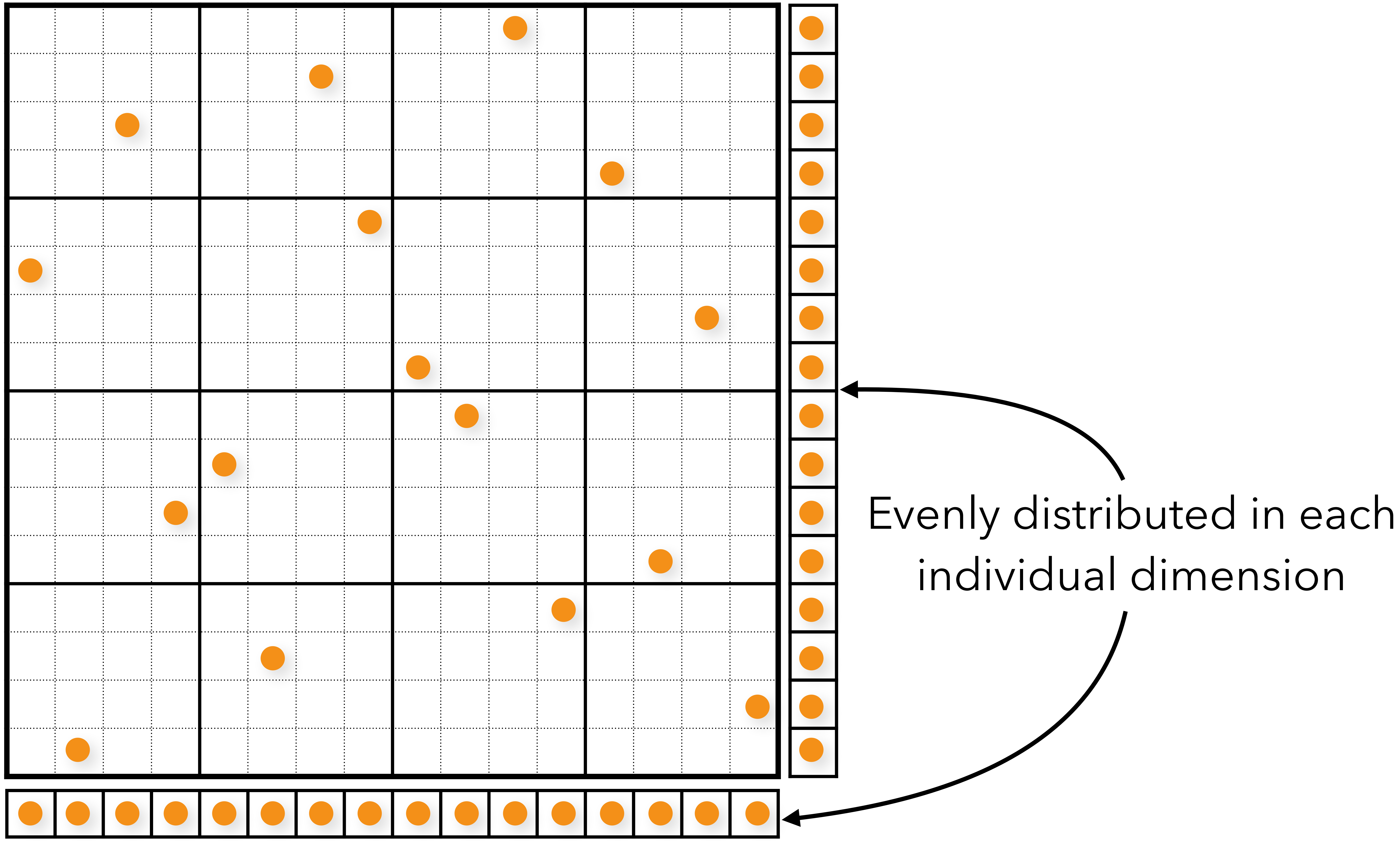
Multi-Jittered Sampling (Projections)



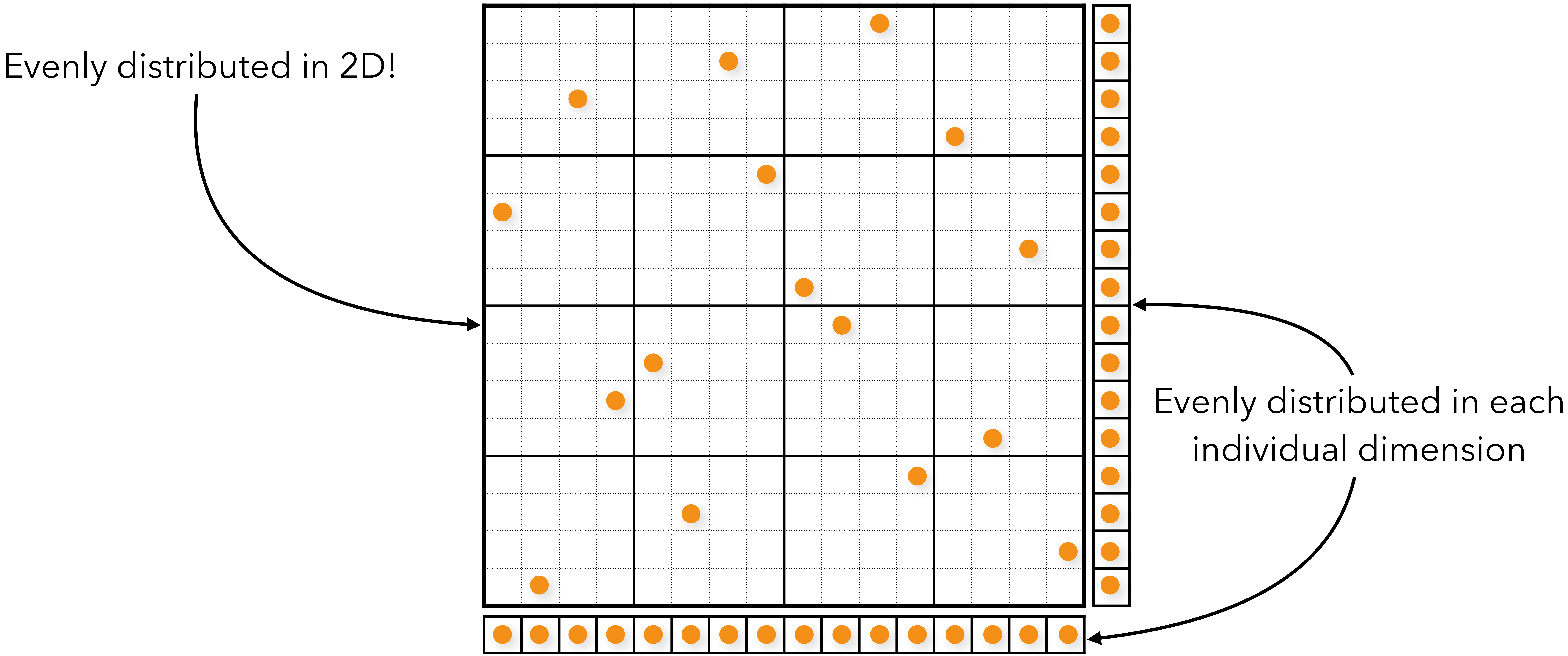
Multi-Jittered Sampling (Projections)



Multi-Jittered Sampling (Projections)



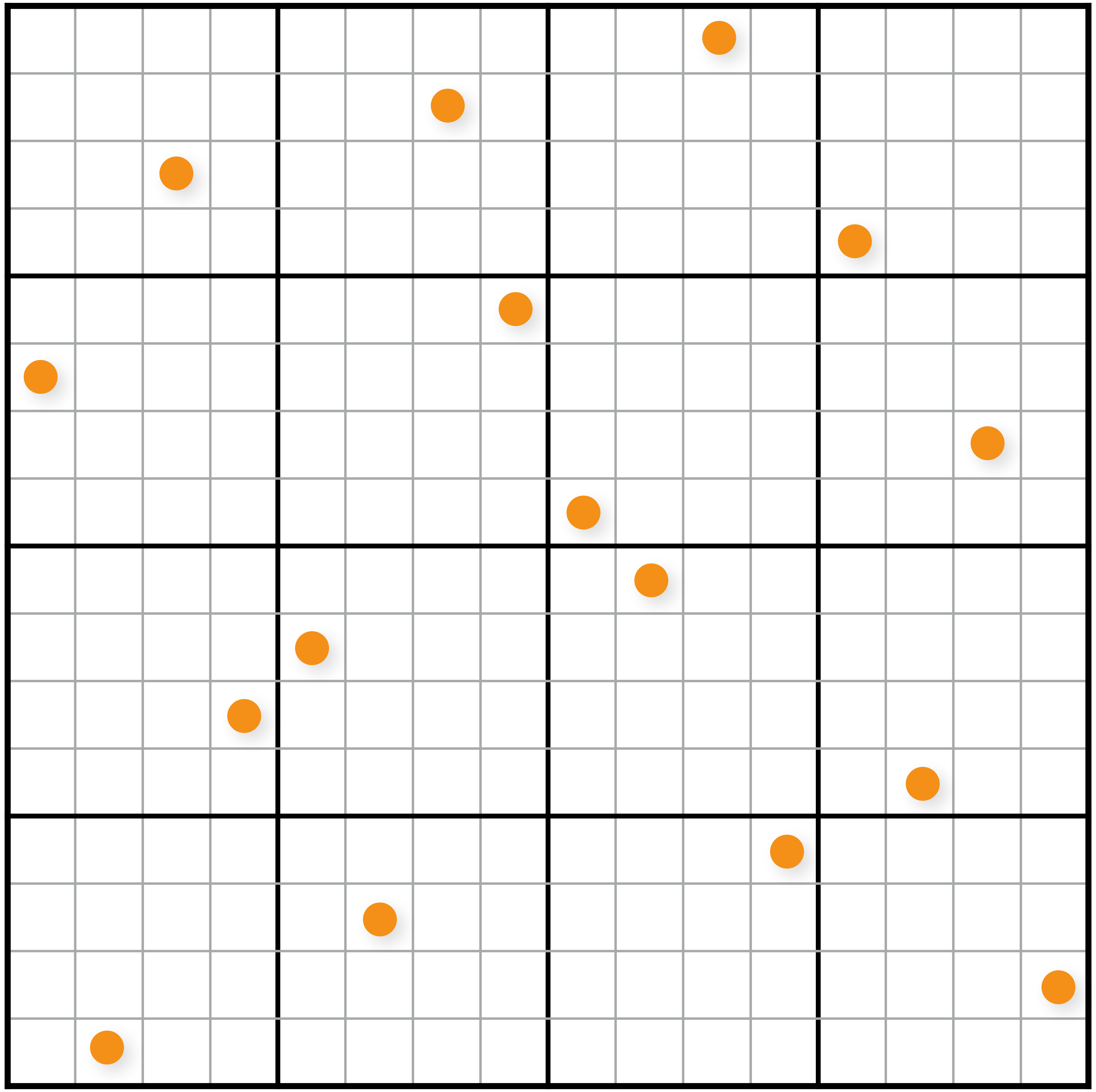
Multi-Jittered Sampling (Projections)



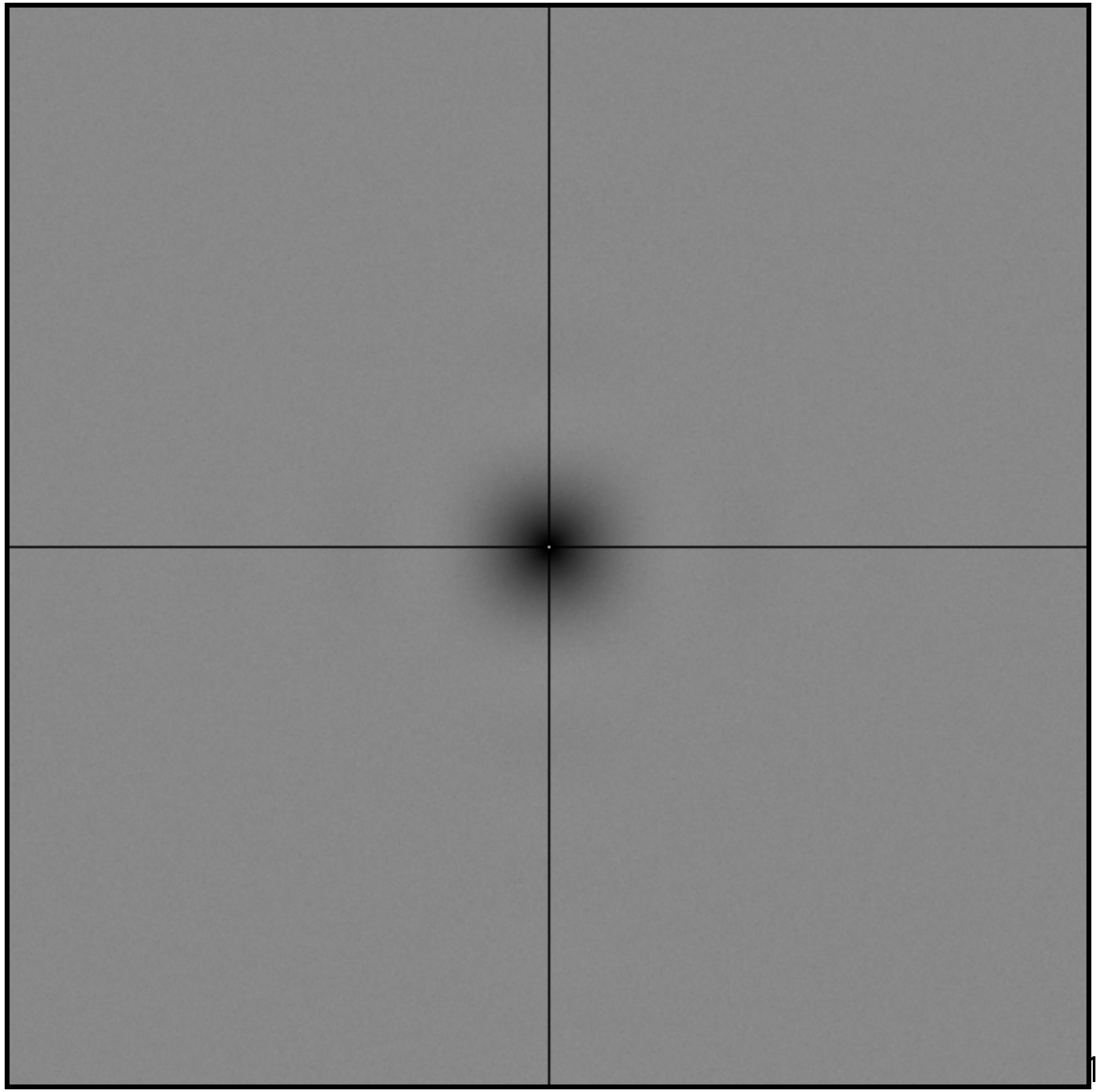
Multi-Jittered Sampling

[Chiu et al. 94]

Spatial domain

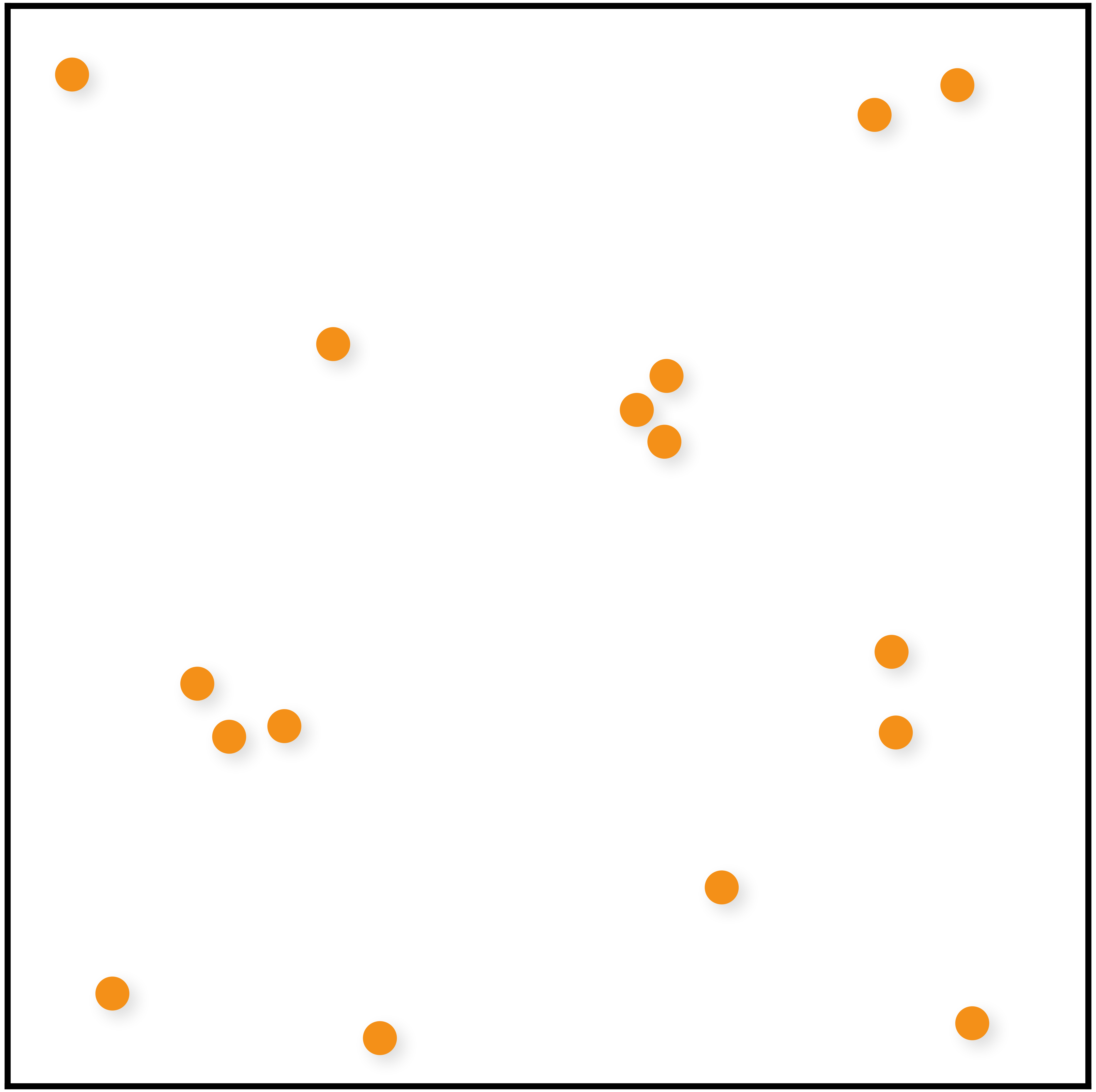


Fourier domain

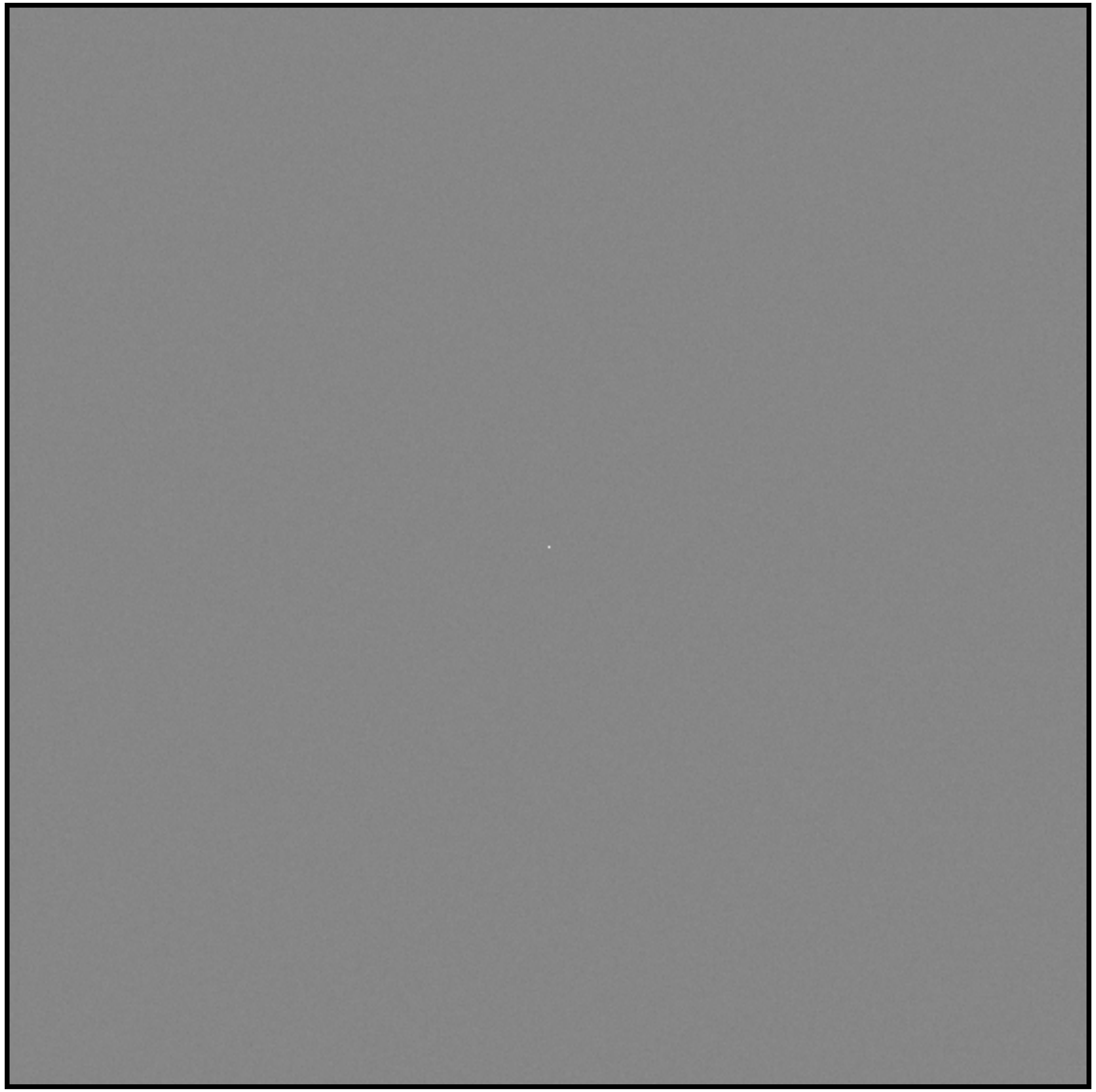


Independent Random Sampling

Spatial domain



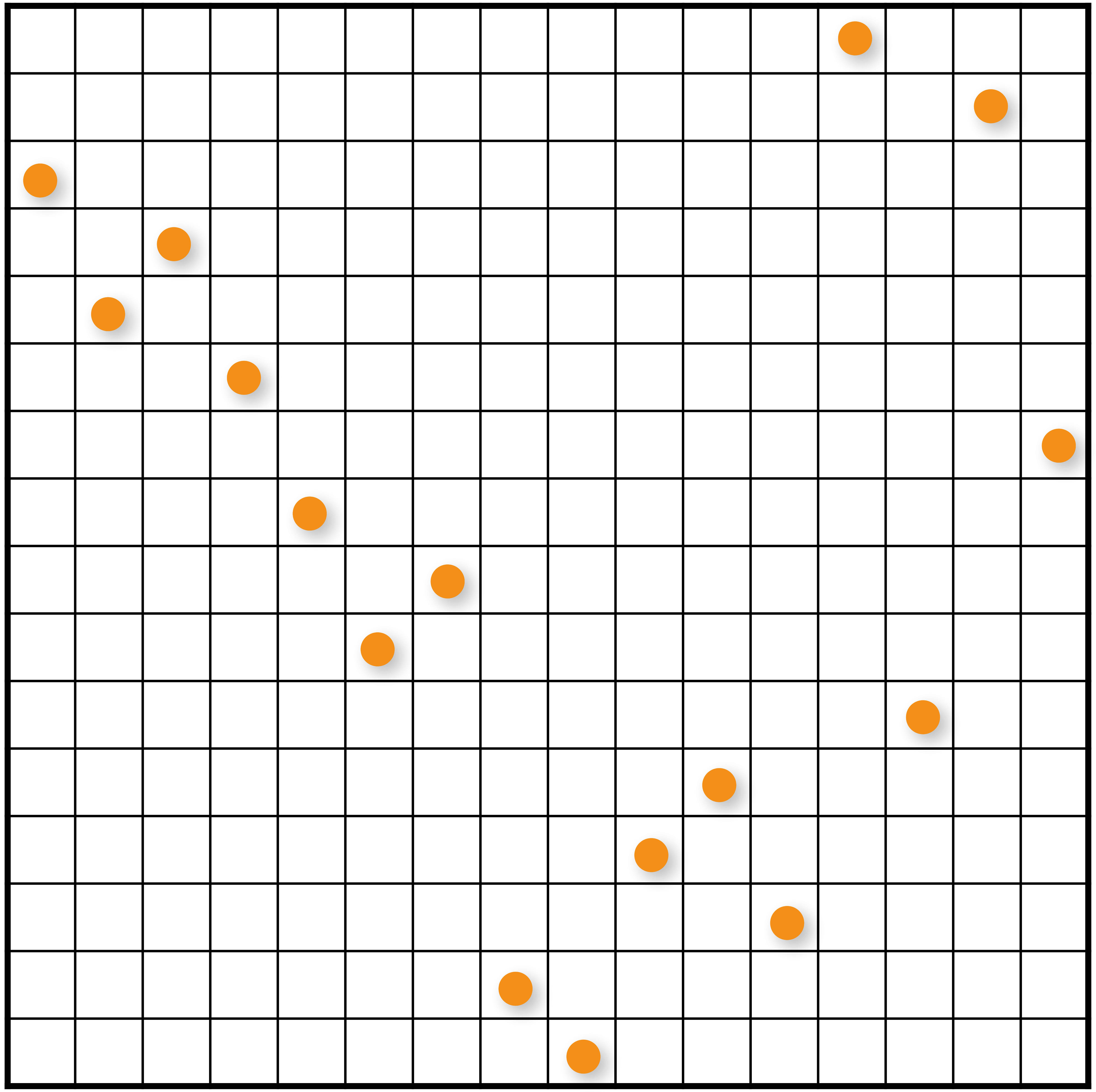
Fourier domain



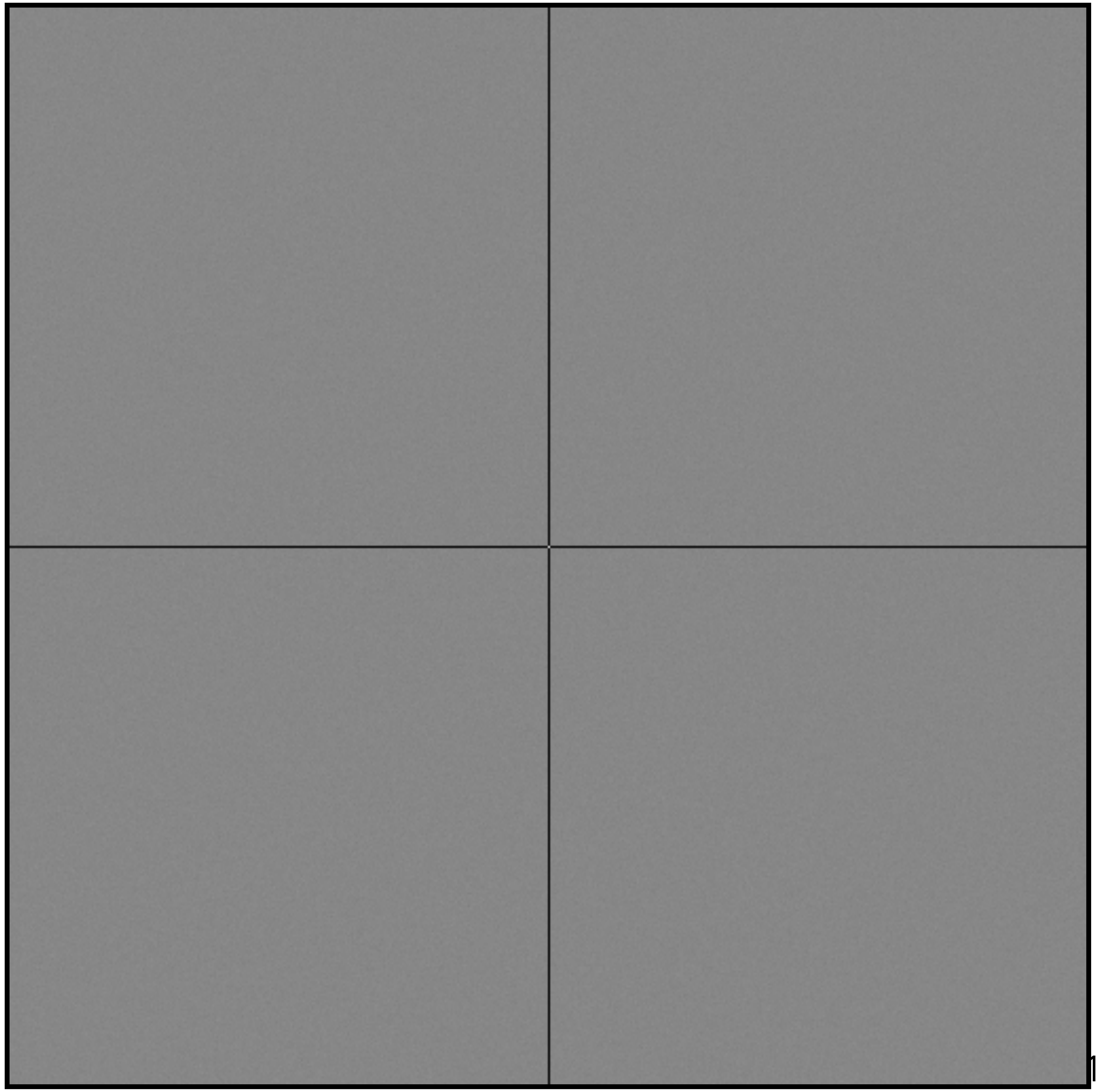
N-Rooks Sampling

[McKay et al. 79]
[Shirley 94]

Spatial domain

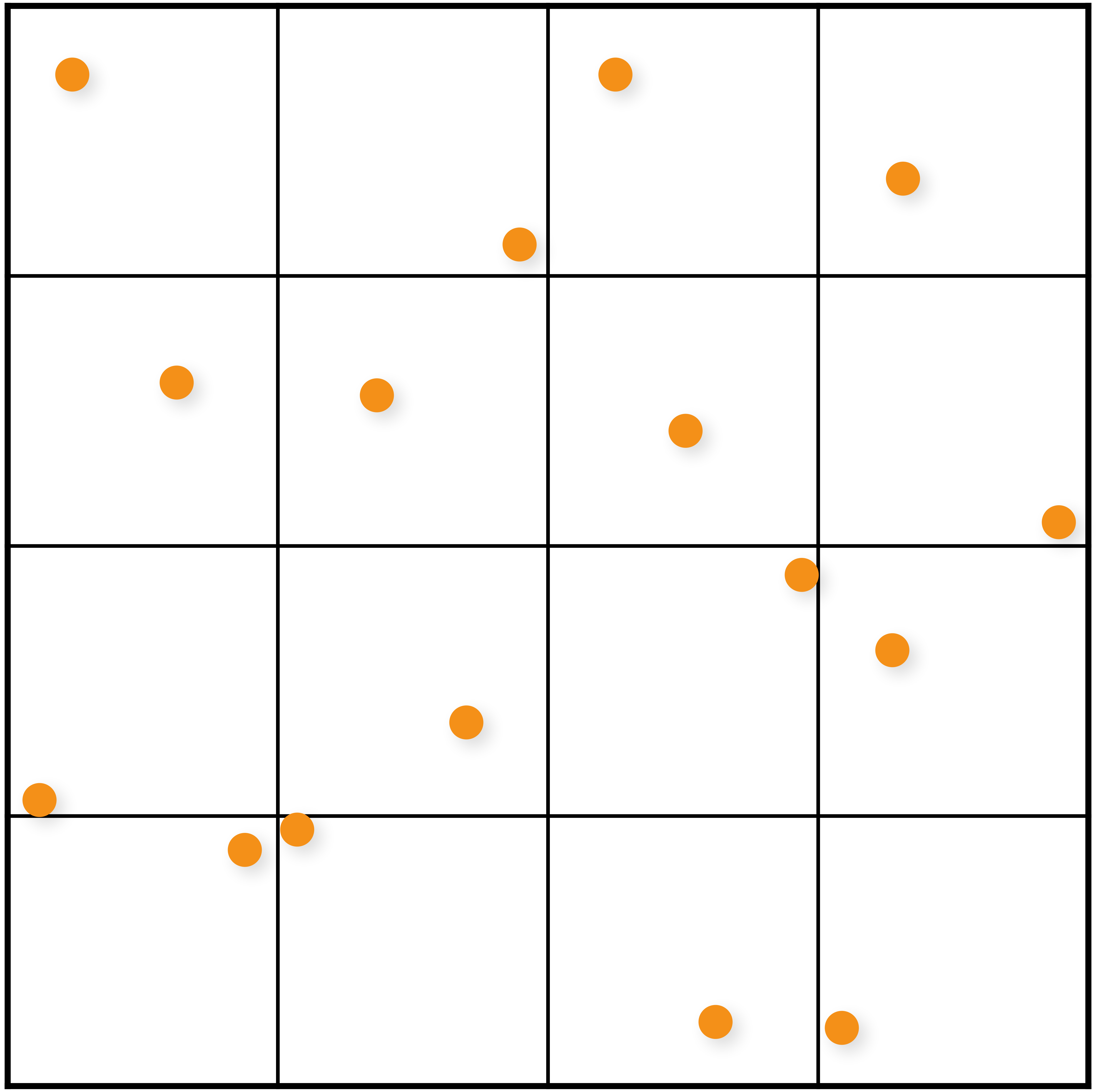


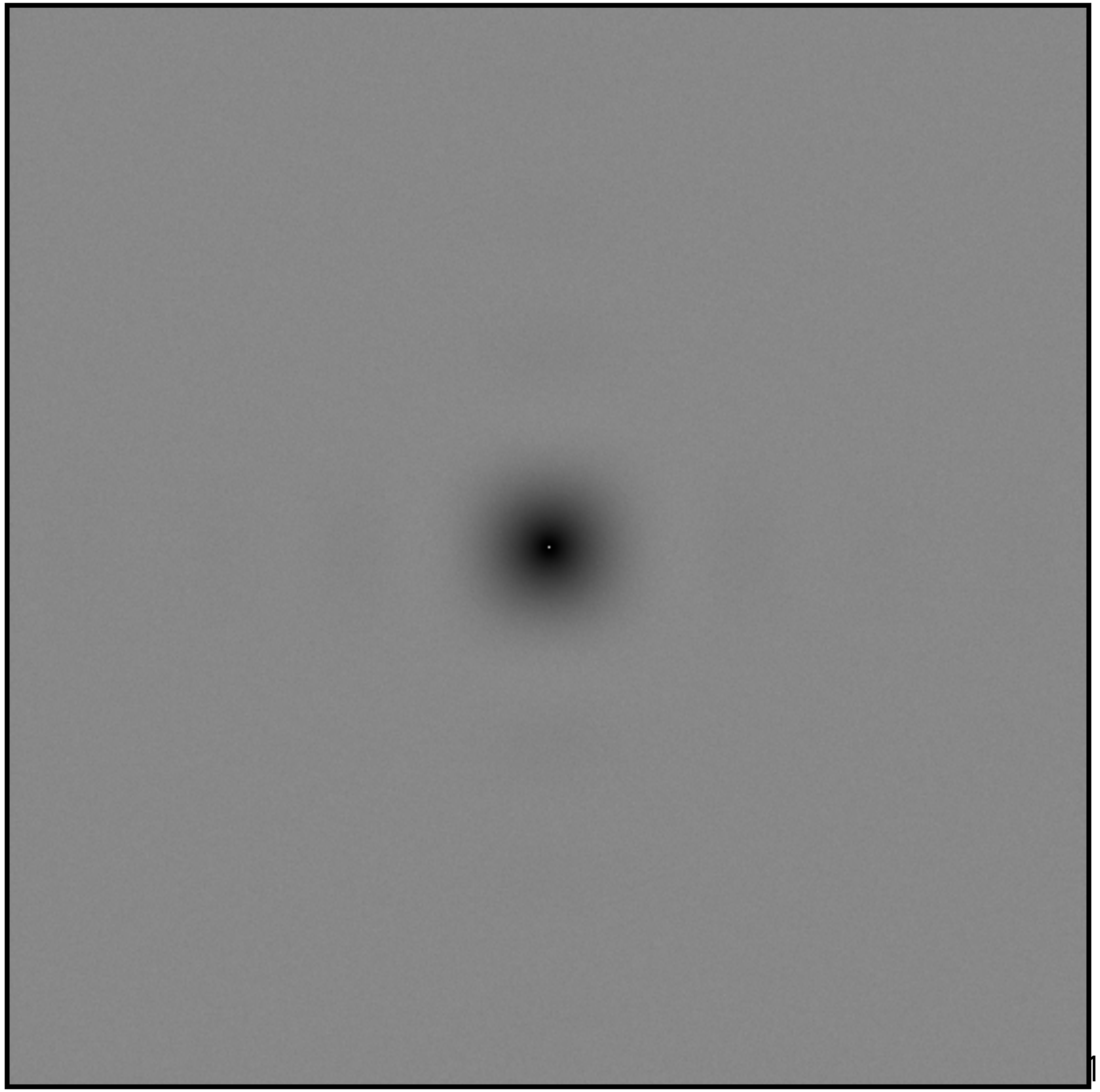
Fourier domain



Jittered Sampling

[Cook 86]

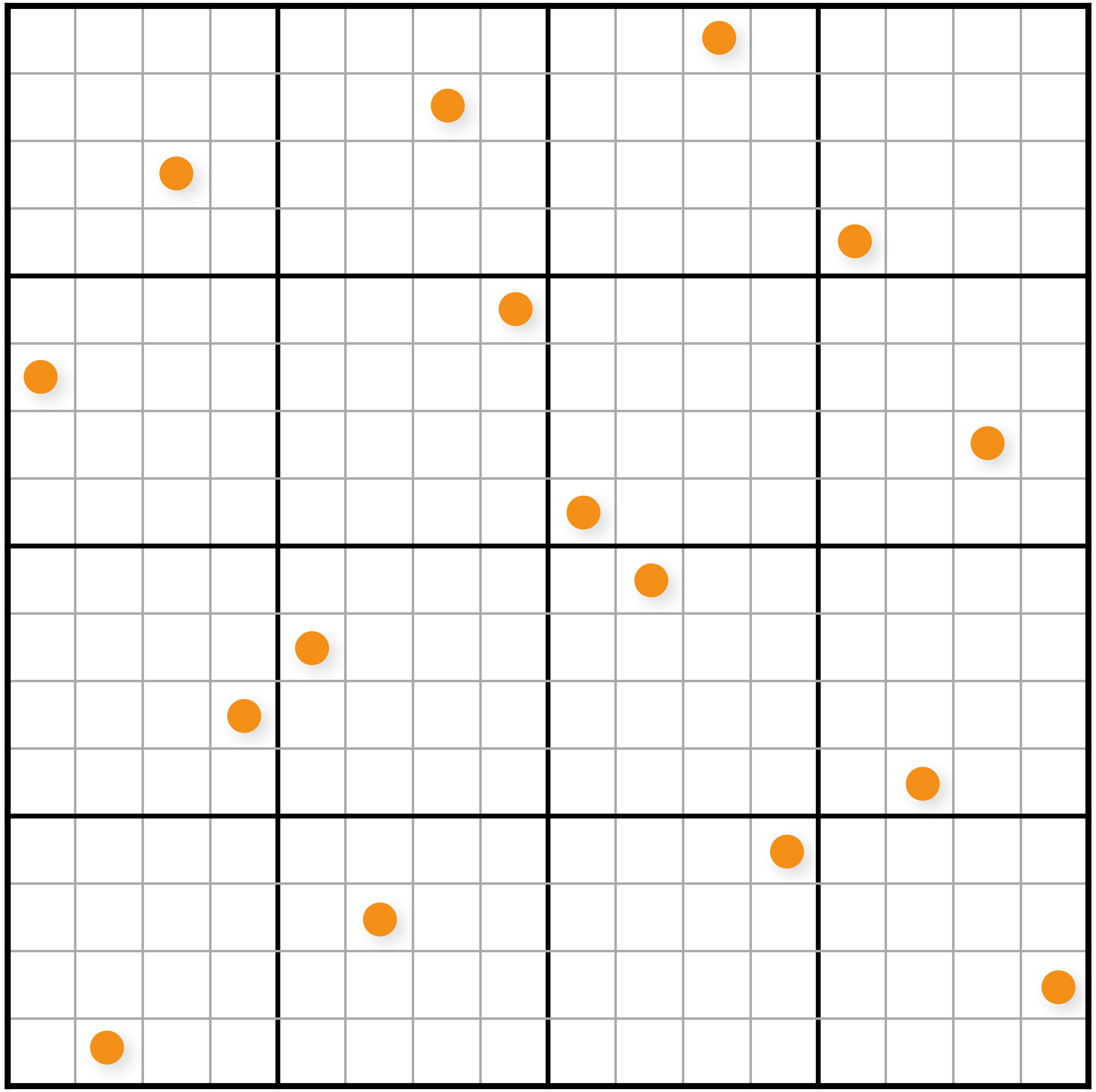




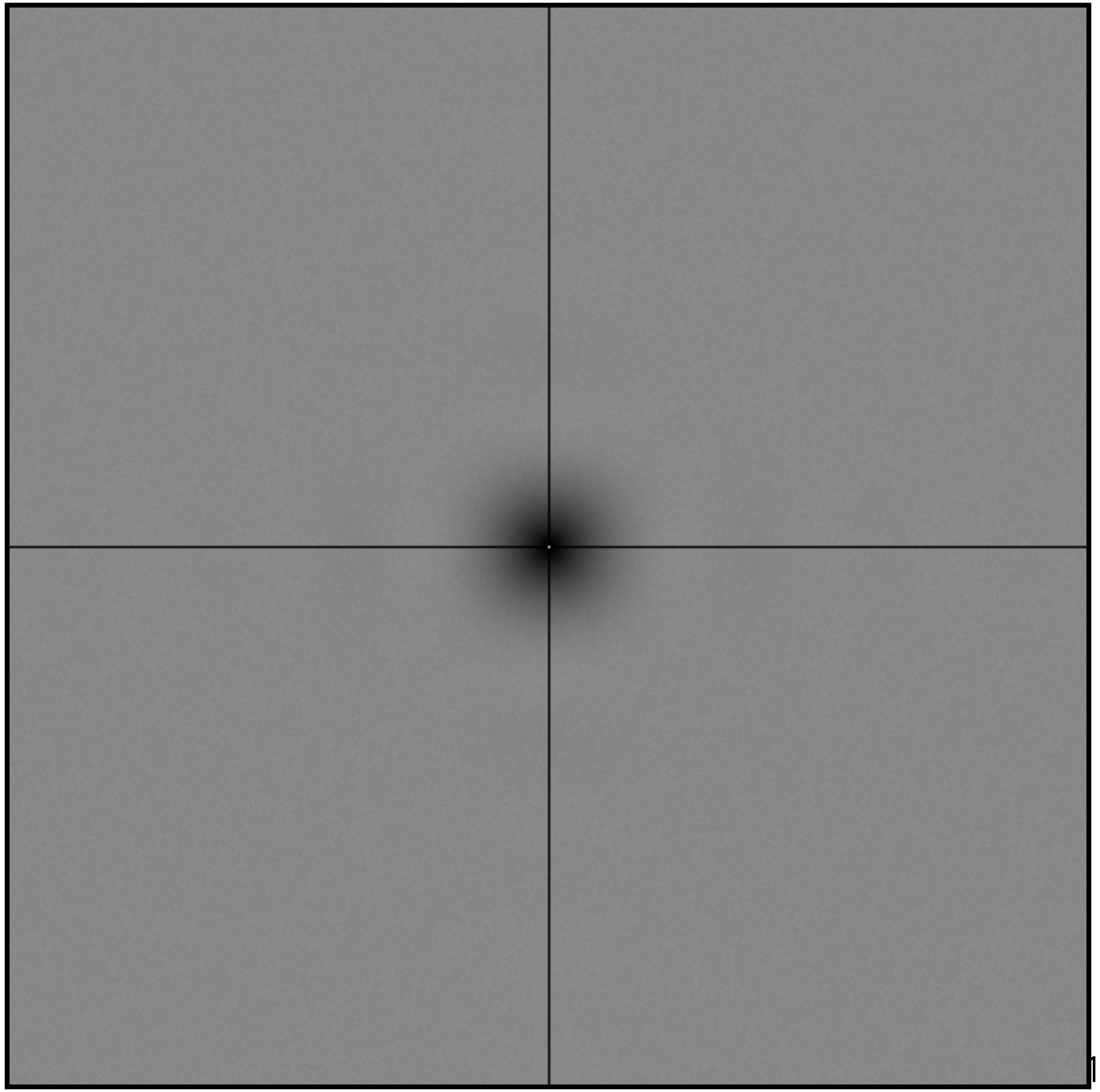
Multi-Jittered Sampling

[Chiu et al. 94]

Spatial domain



Fourier domain

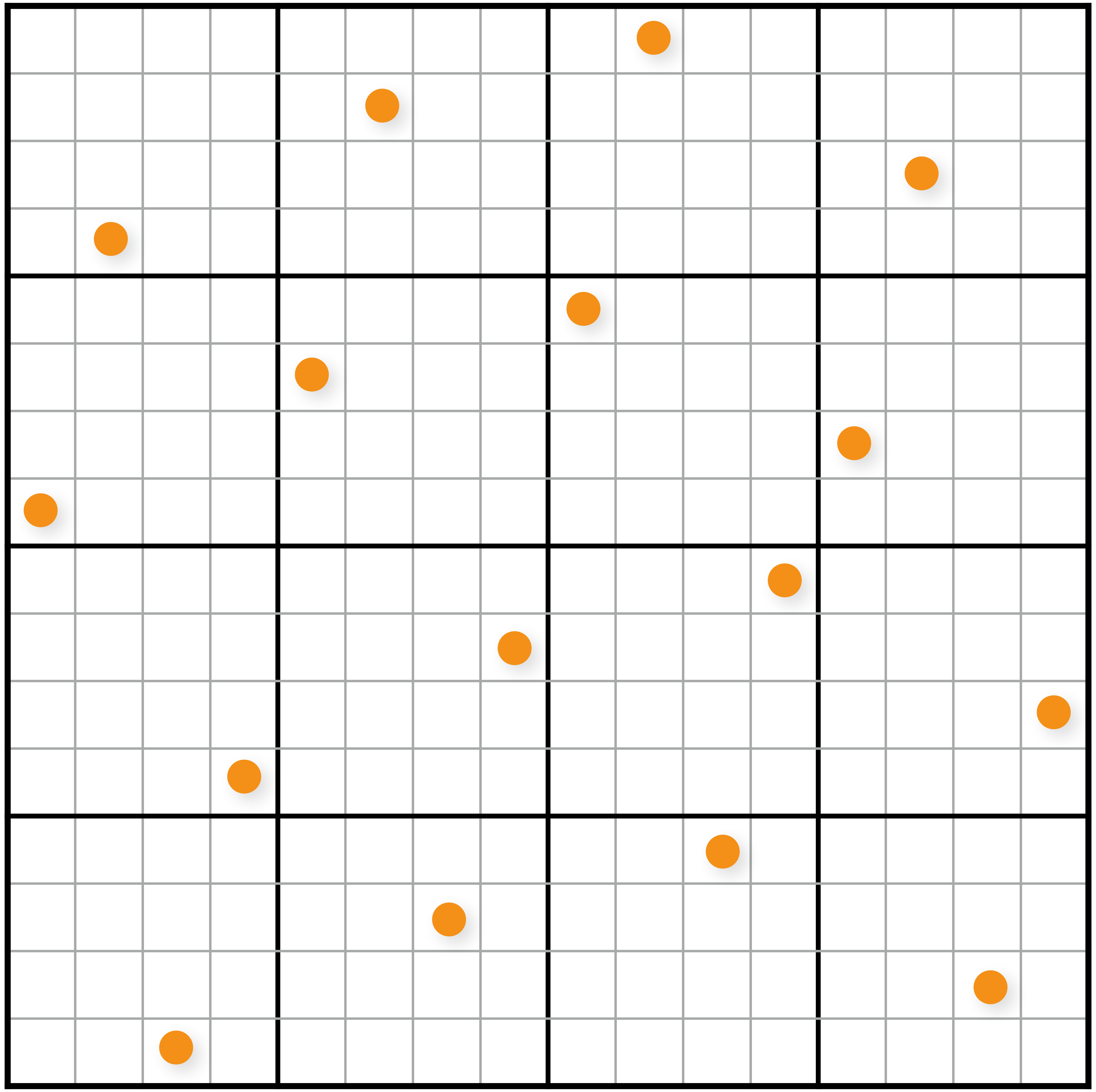


same shuffle for all rows/columns

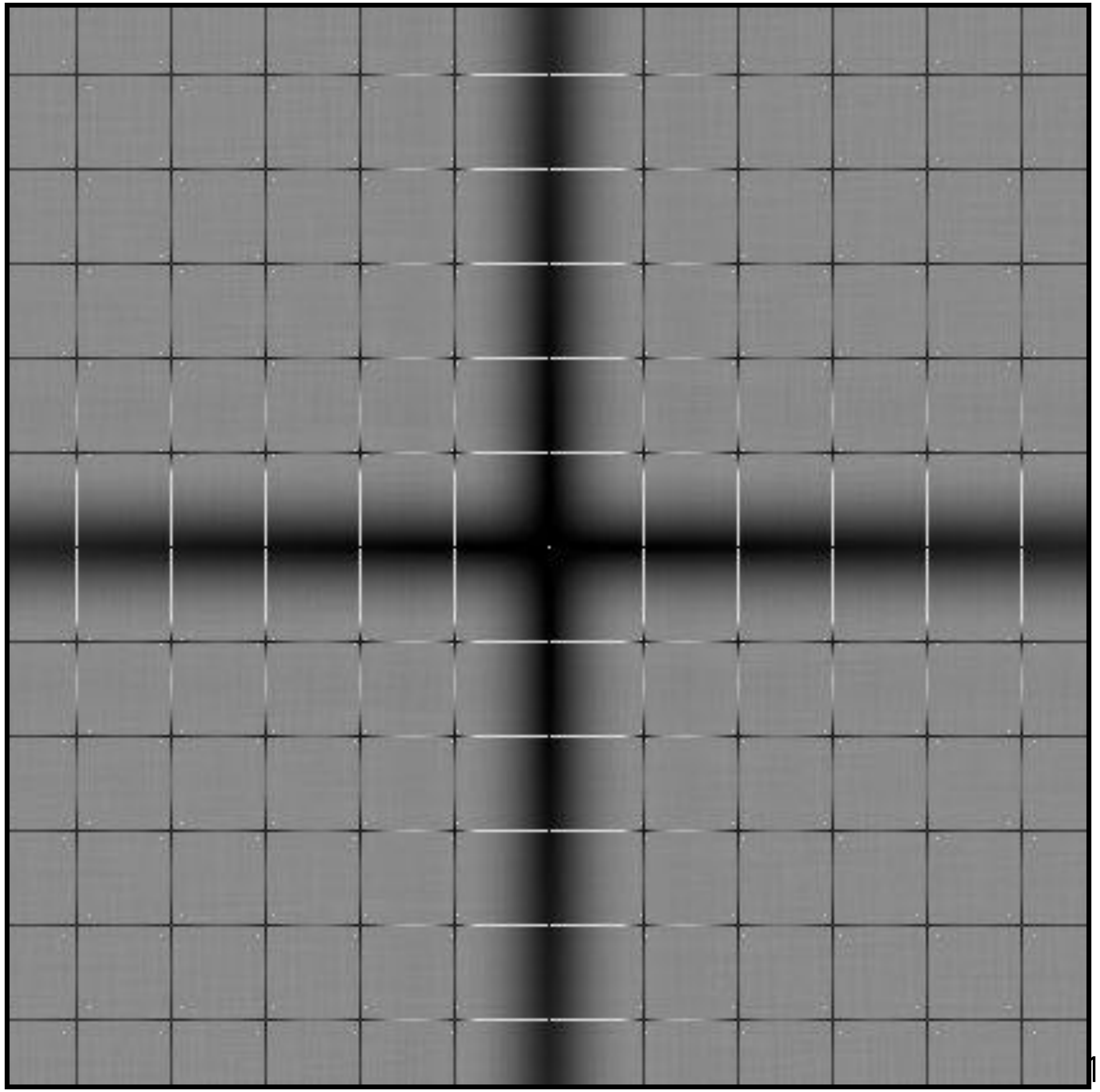
Correlated MJ Sampling

[Kensler 13]

Spatial domain



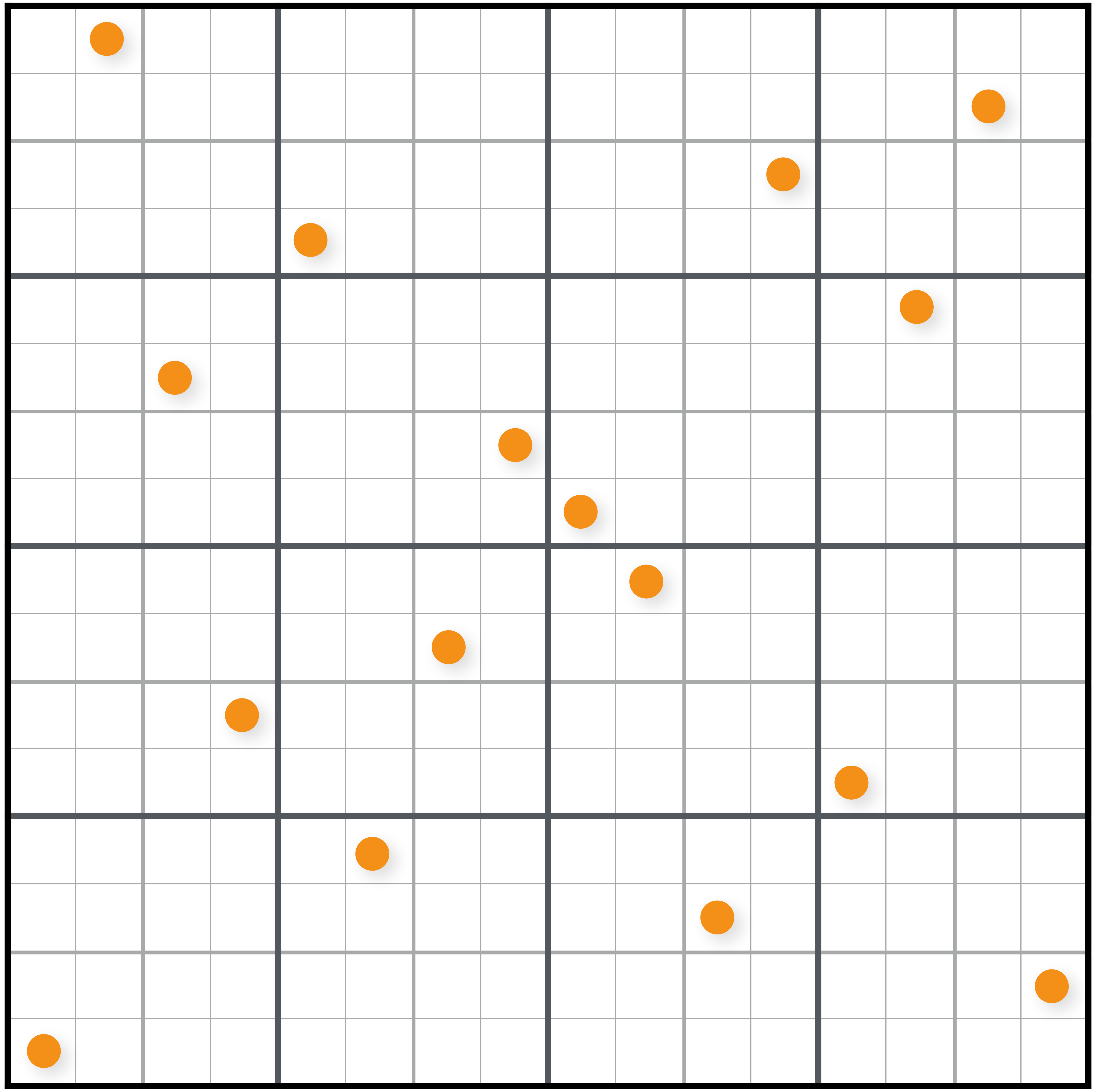
Fourier domain



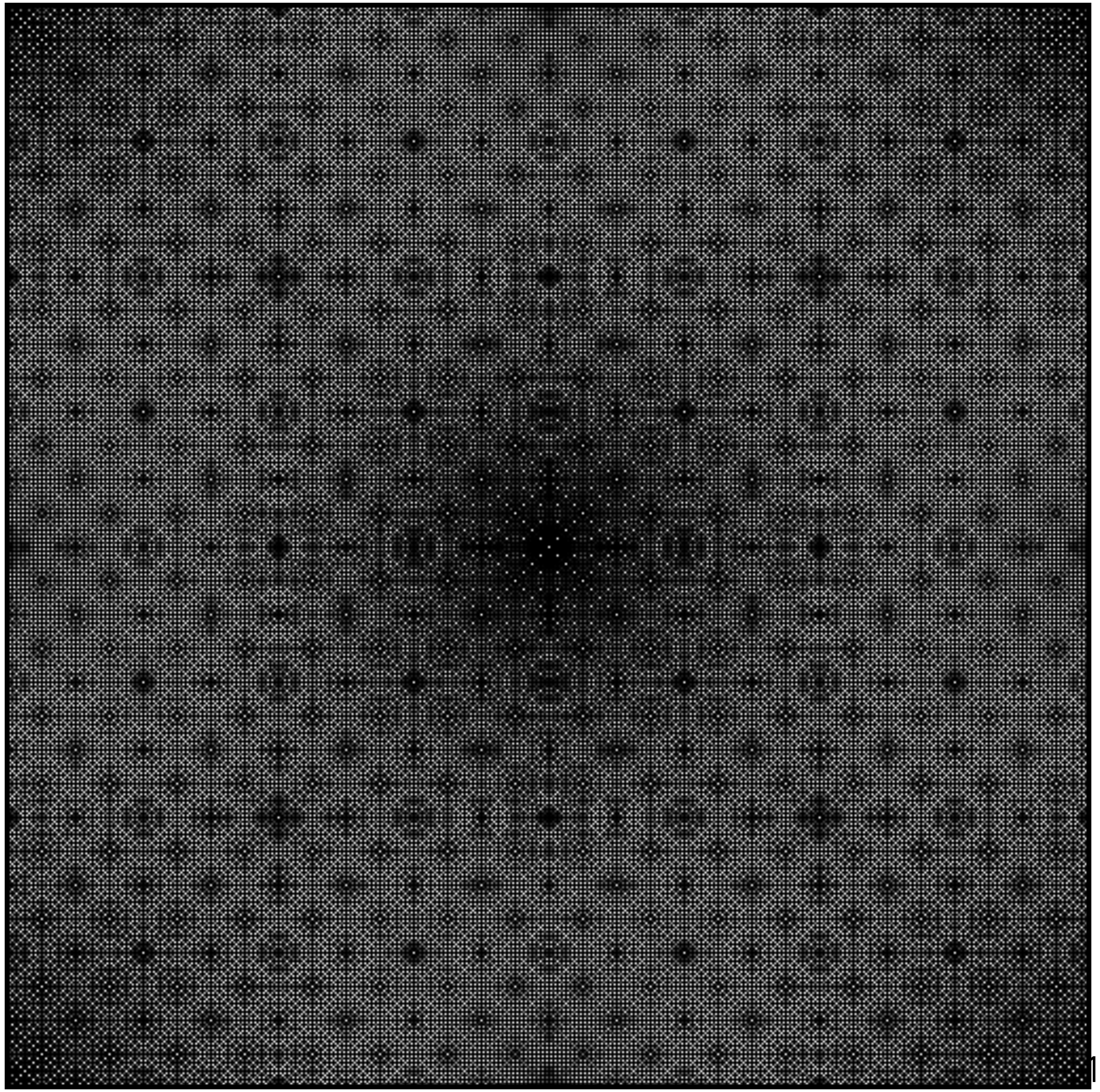
(0,2) sequence

[Sobol 67]
[Kollig & Keller 02]

Spatial domain



Fourier domain

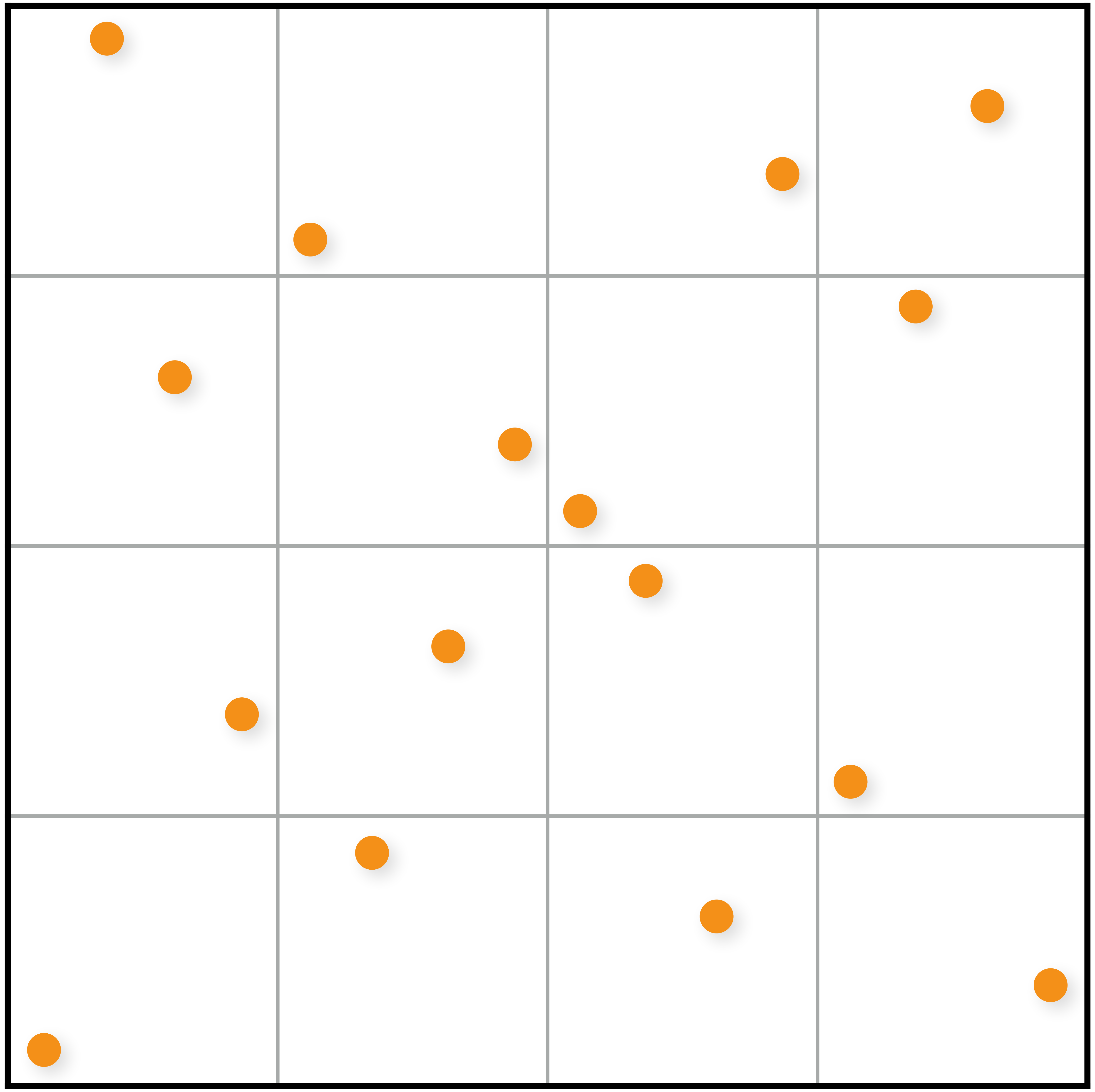


(0,2) sequence

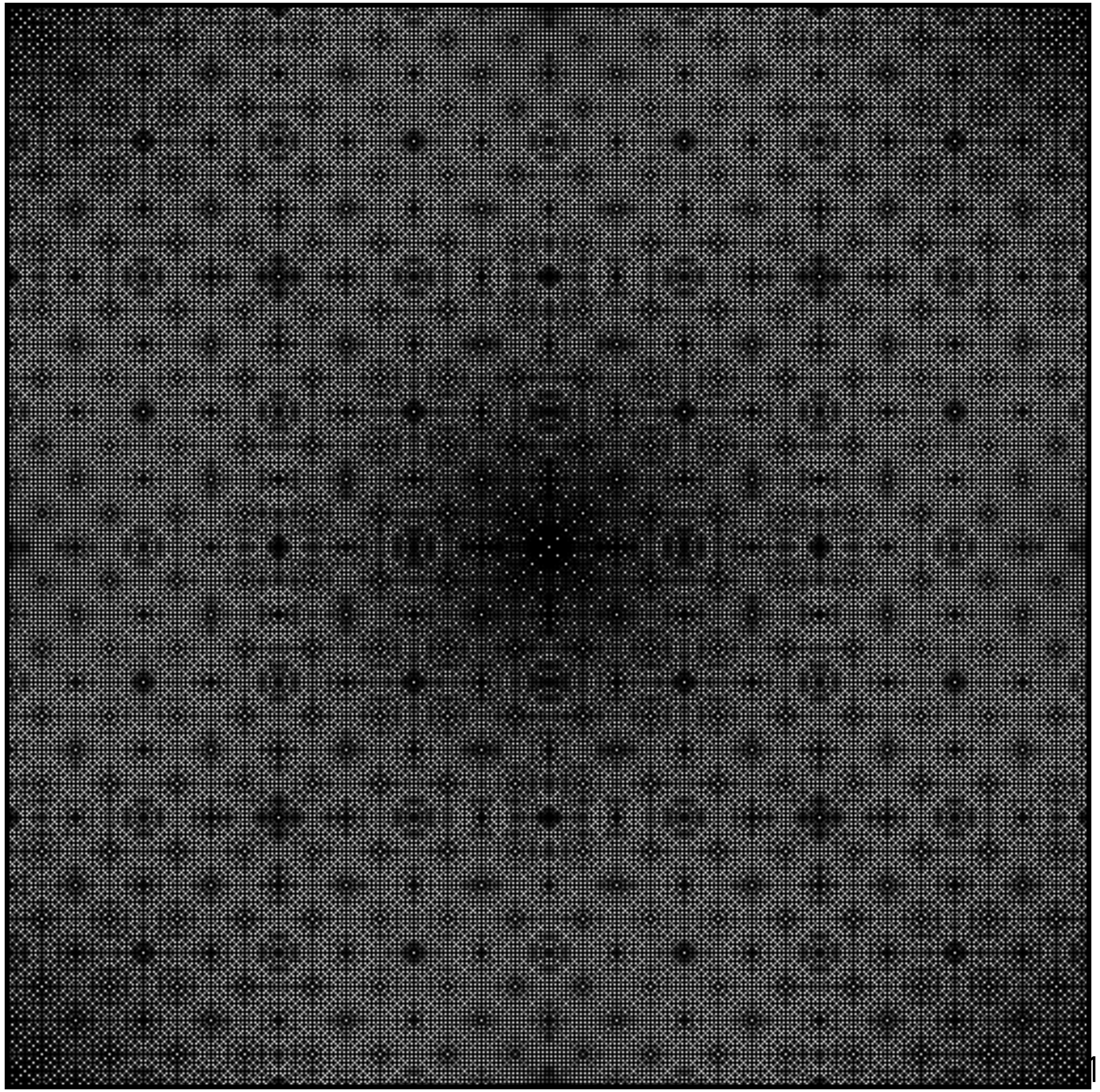
1 sample in each
“elementary interval”

[Sobol 67]
[Kollig & Keller 02]

Spatial domain



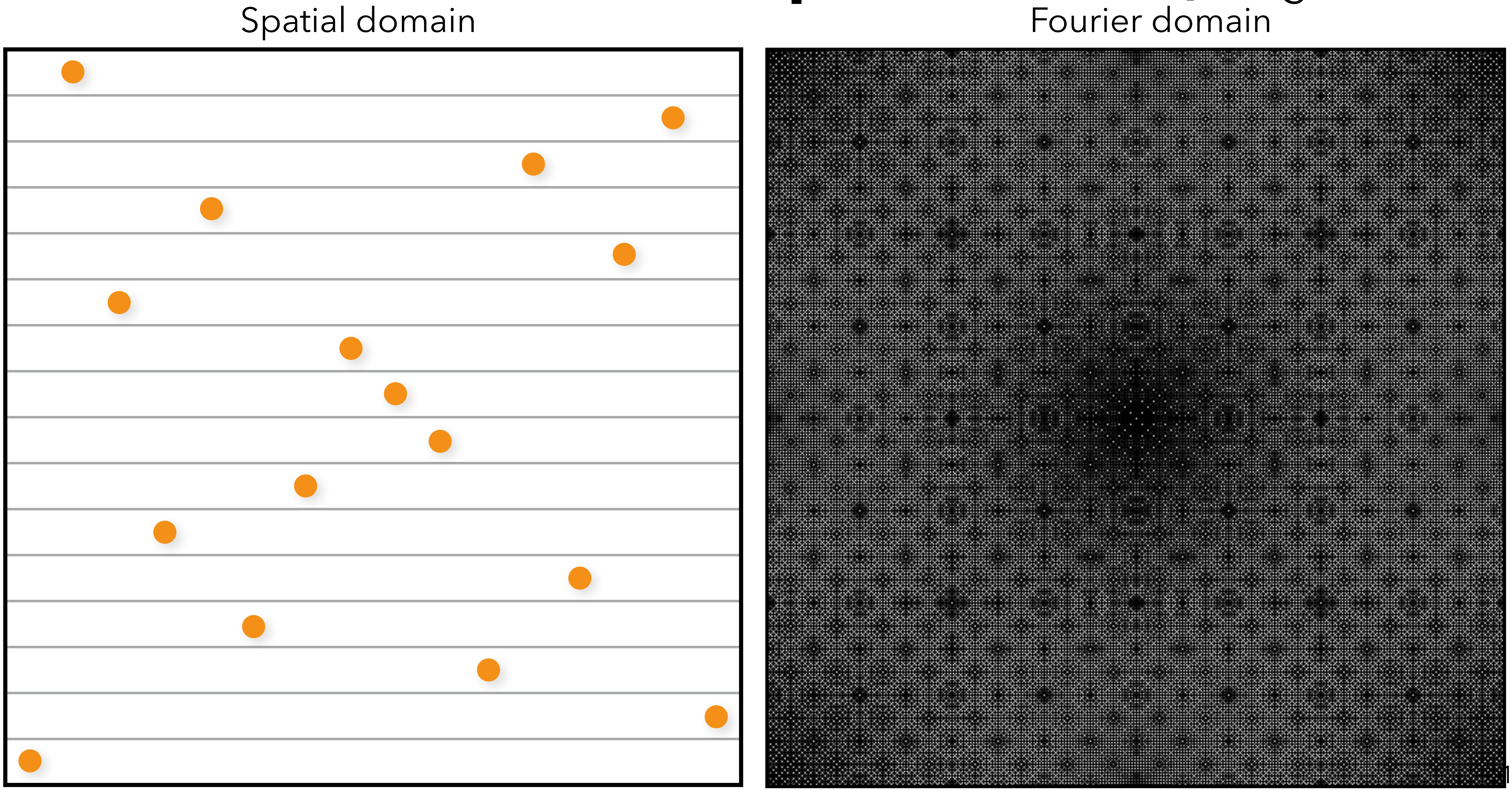
Fourier domain



(0,2) sequence

1 sample in each
"elementary interval"

[Sobol 67]
[Kollig & Keller 02]

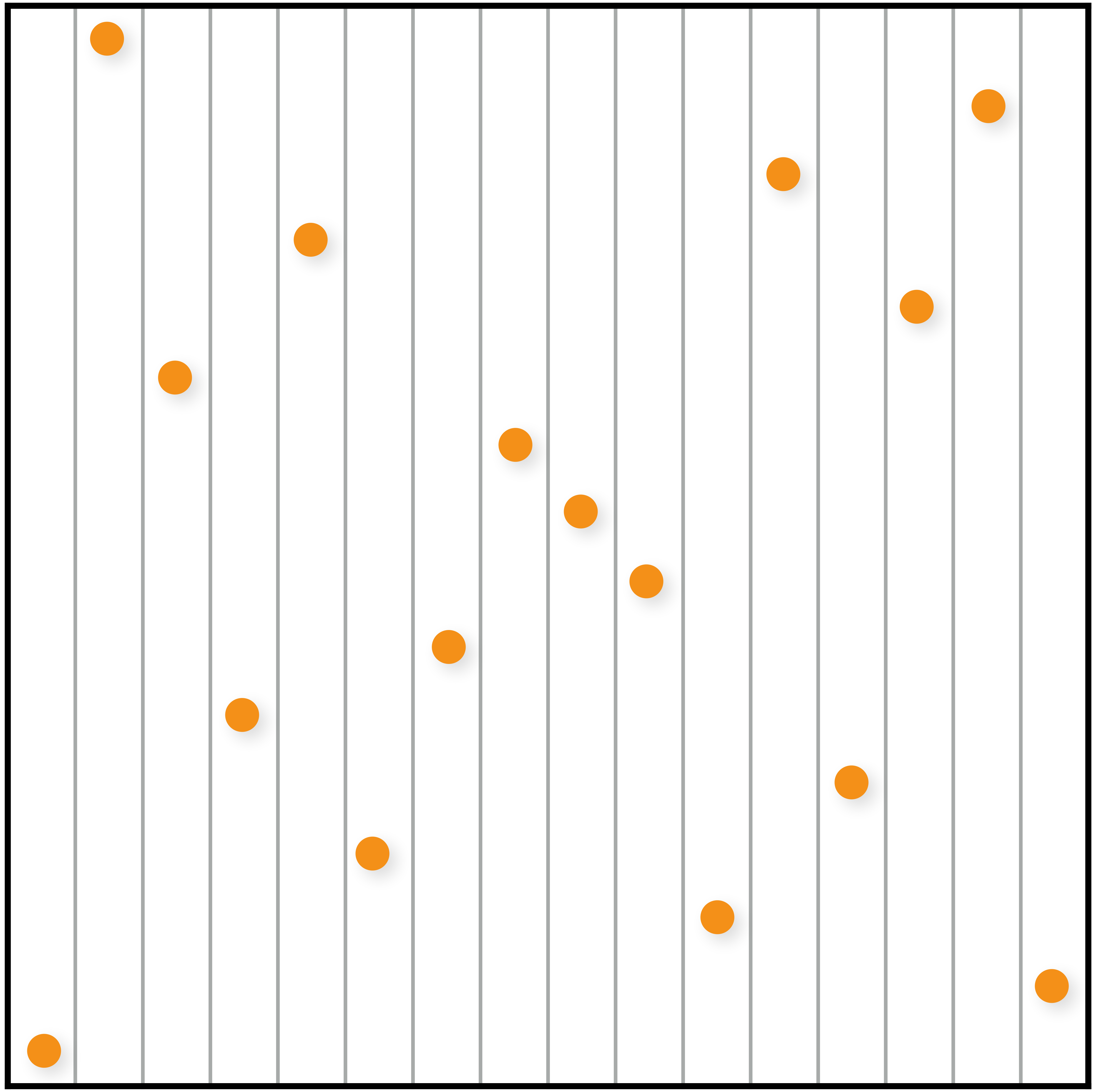


1 sample in each
"elementary interval"

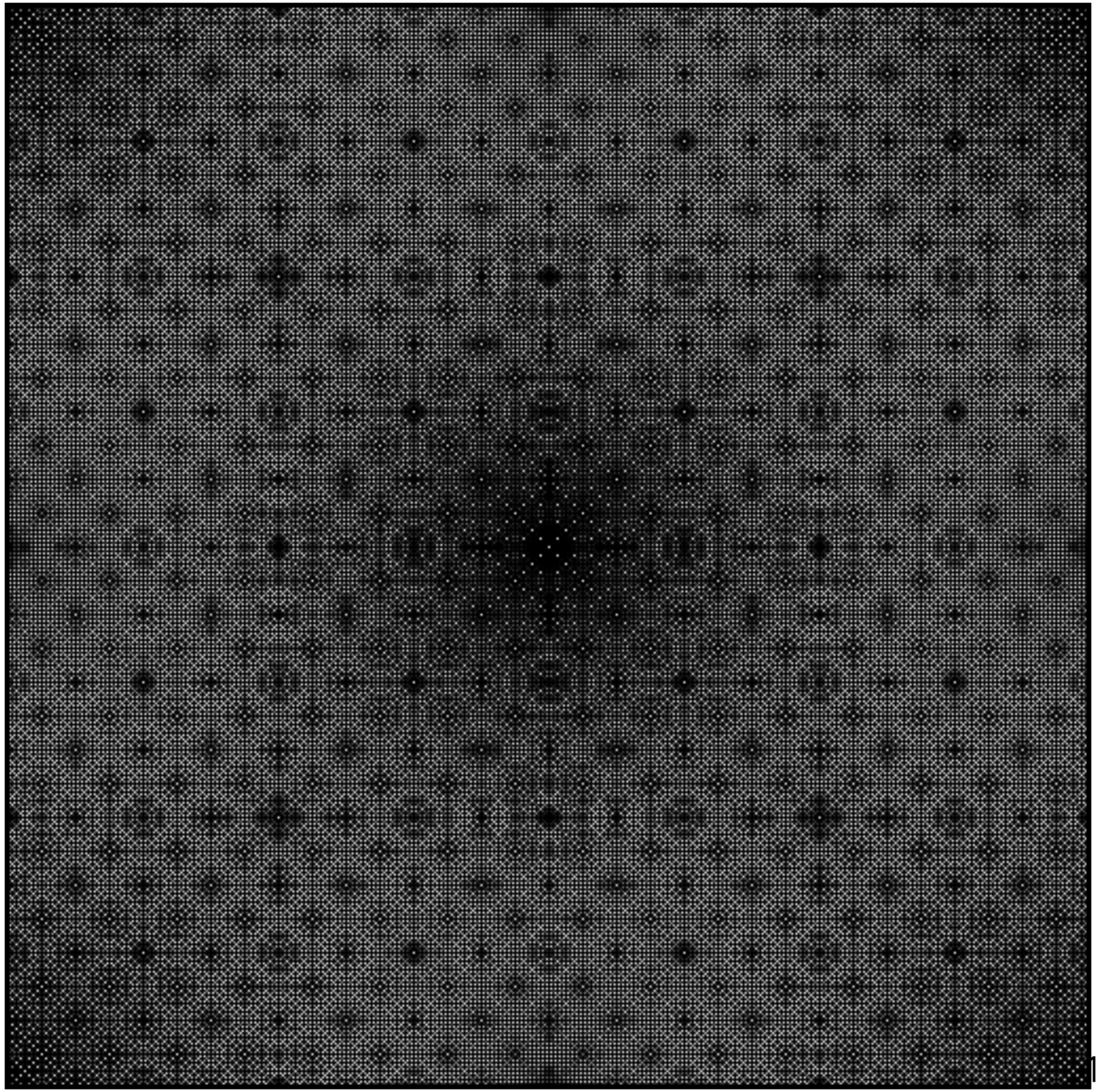
(0,2) sequence

[Sobol 67]
[Kollig & Keller 02]

Spatial domain



Fourier domain

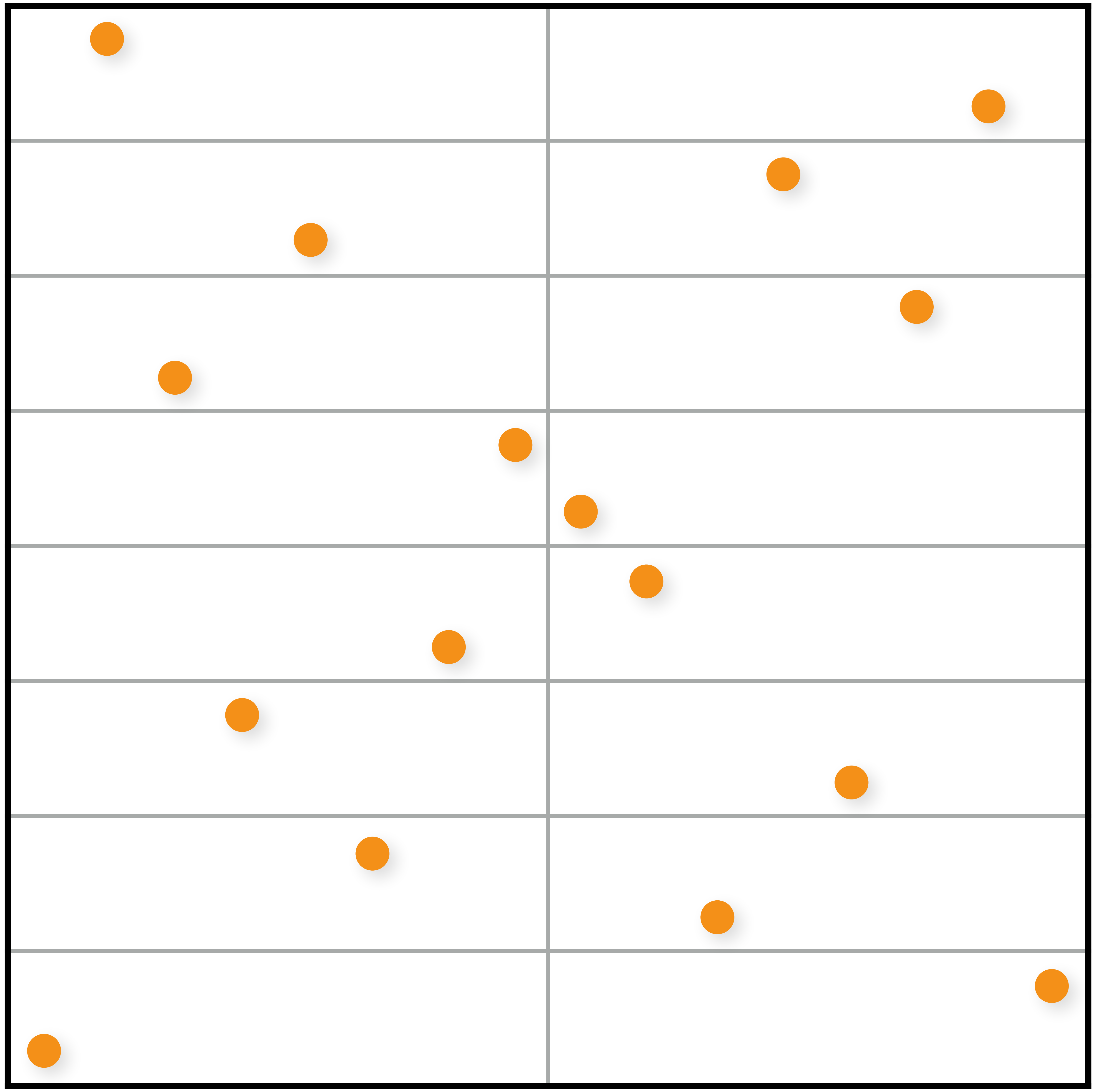


1 sample in each
“elementary interval”

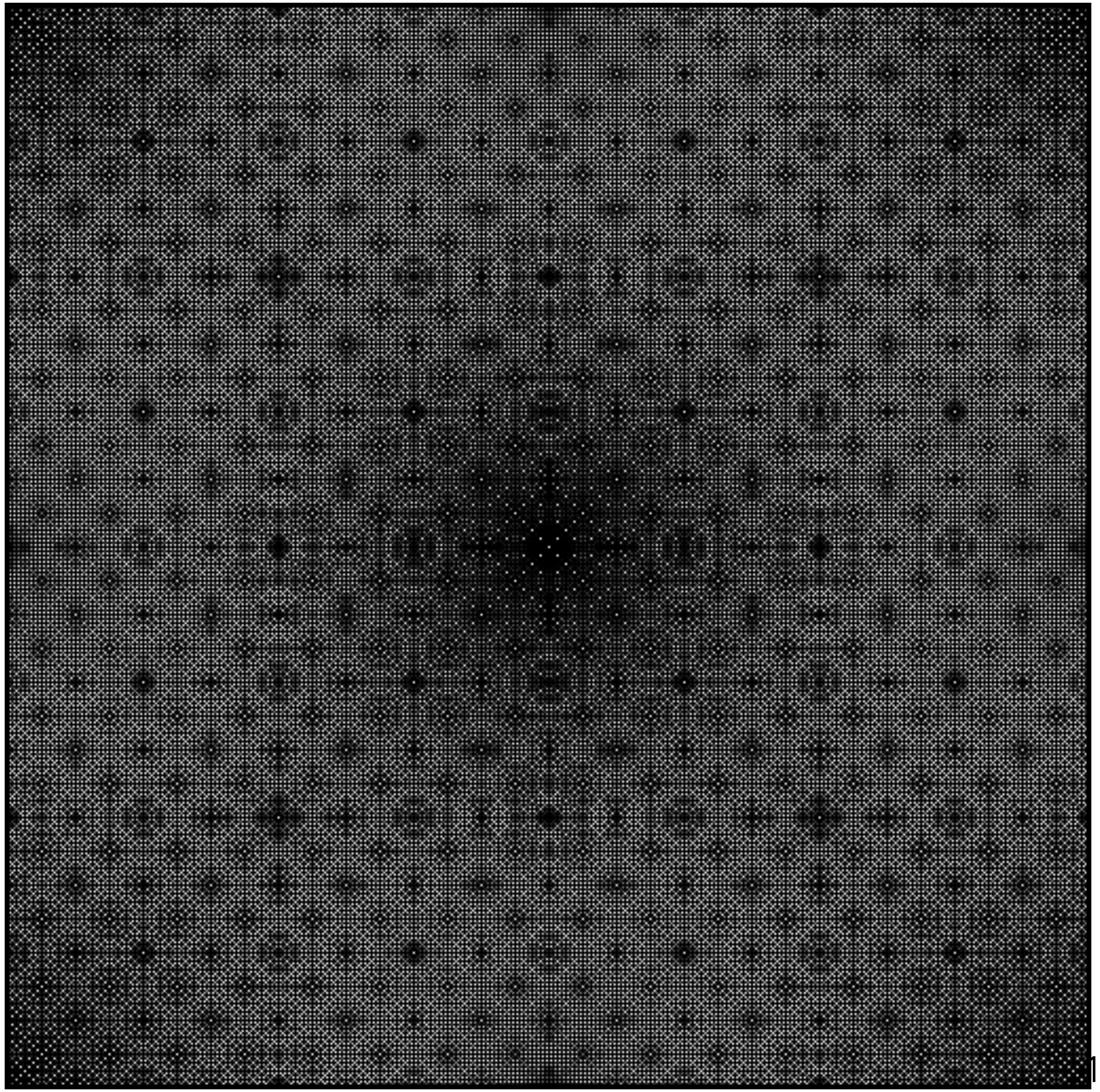
(0,2) sequence

[Sobol 67]
[Kollig & Keller 02]

Spatial domain



Fourier domain

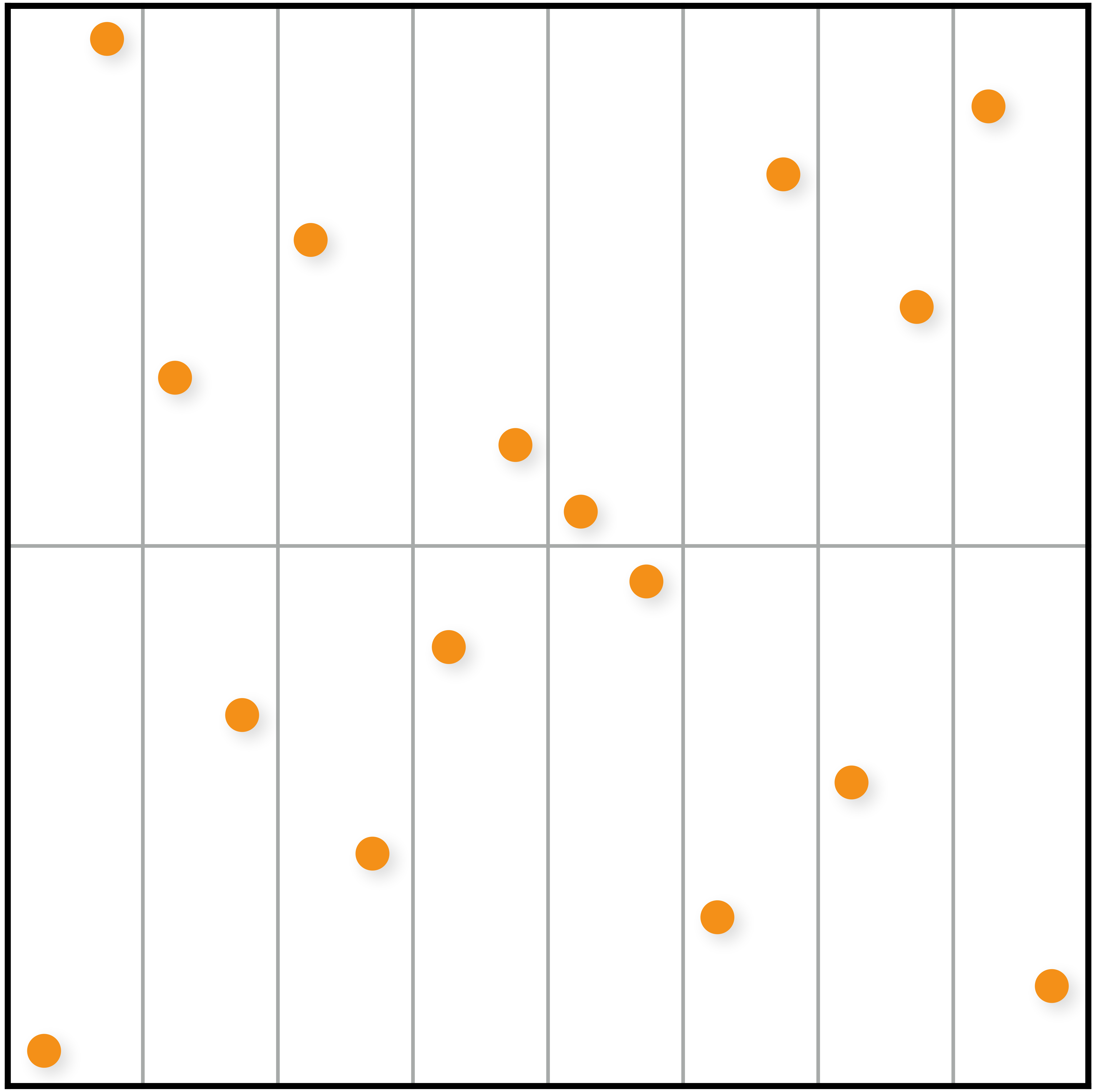


1 sample in each
"elementary interval"

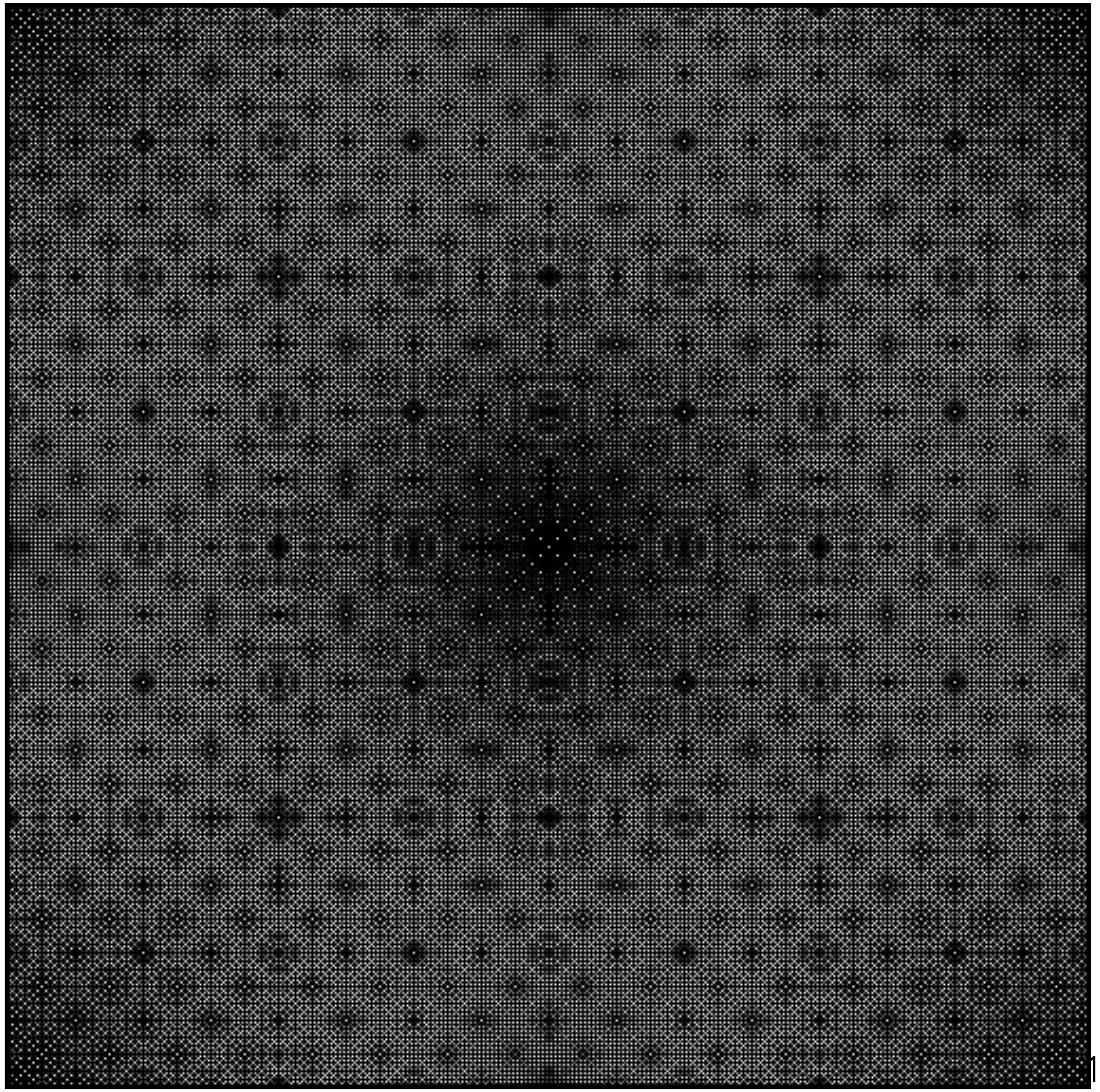
(0,2) sequence

[Sobol 67]
[Kollig & Keller 02]

Spatial domain



Fourier domain

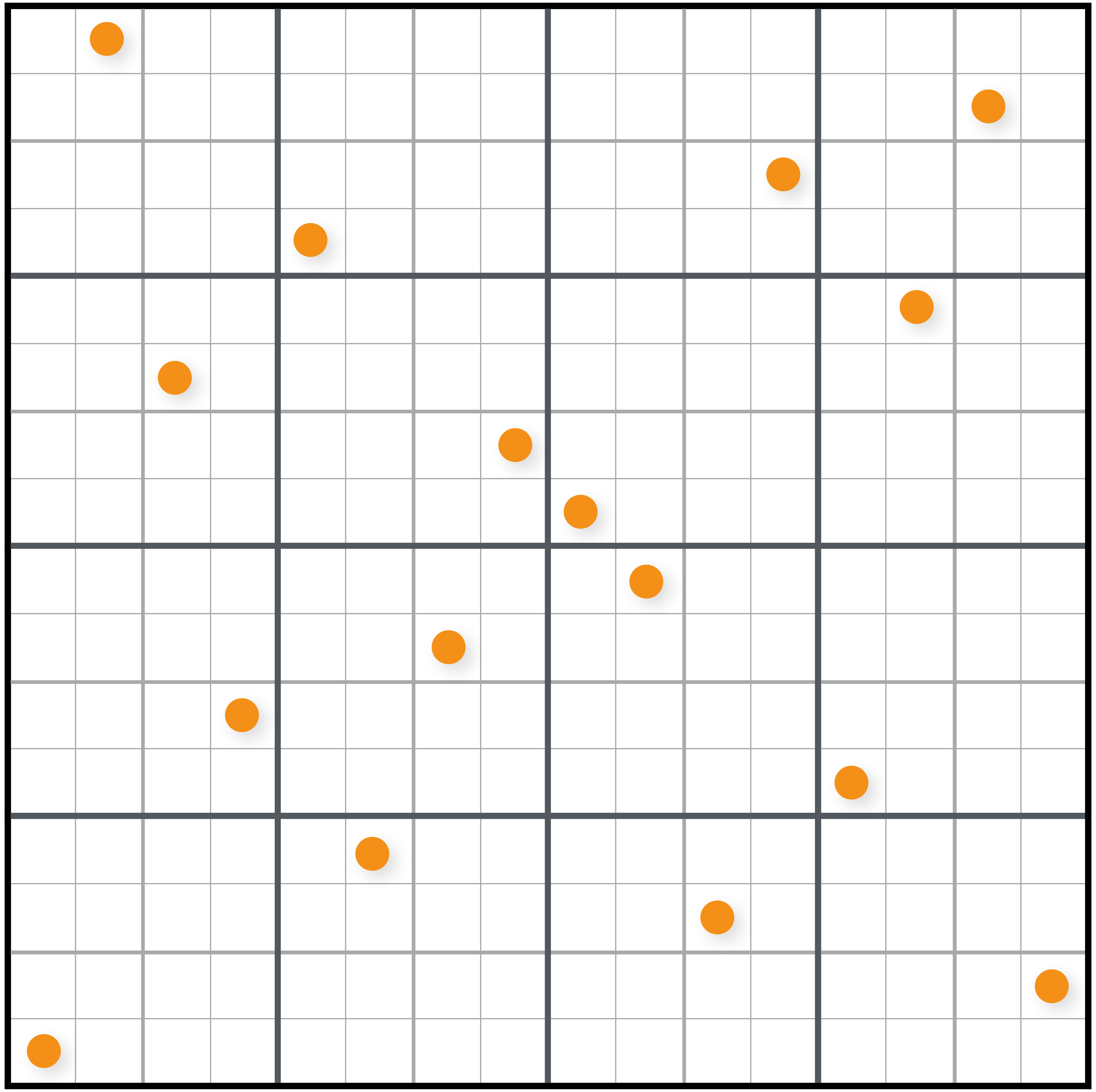


(0,2) sequence

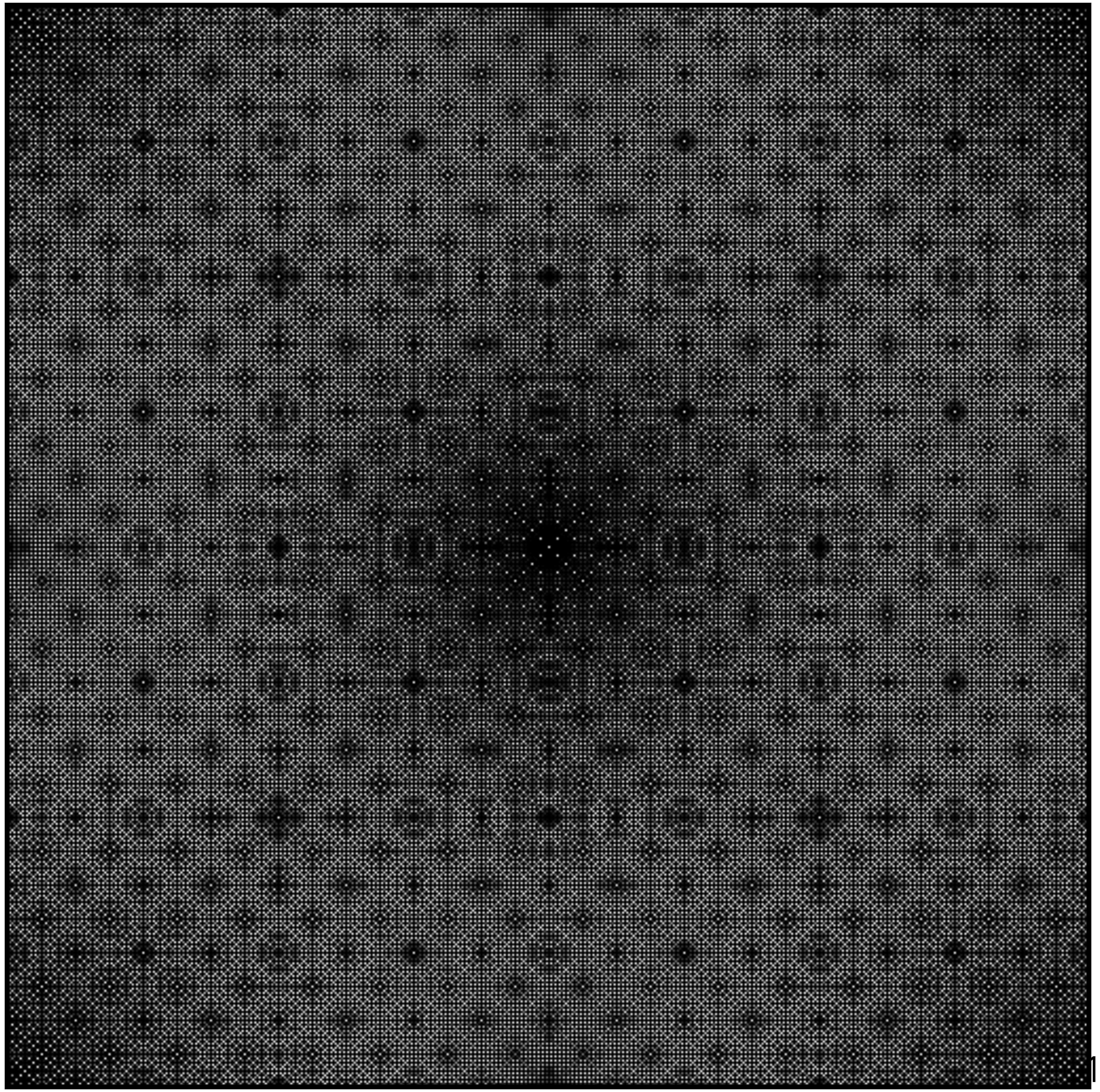
1 sample in each
"elementary interval"

[Sobol 67]
[Kollig & Keller 02]

Spatial domain



Fourier domain



Limitations/Future work

Limitations/Future work

Sample count: $N = p^t$ where t is strength, and p is prime

Limitations/Future work

Sample count: $N = p^t$ where t is strength, and p is prime

- Galois/finite fields

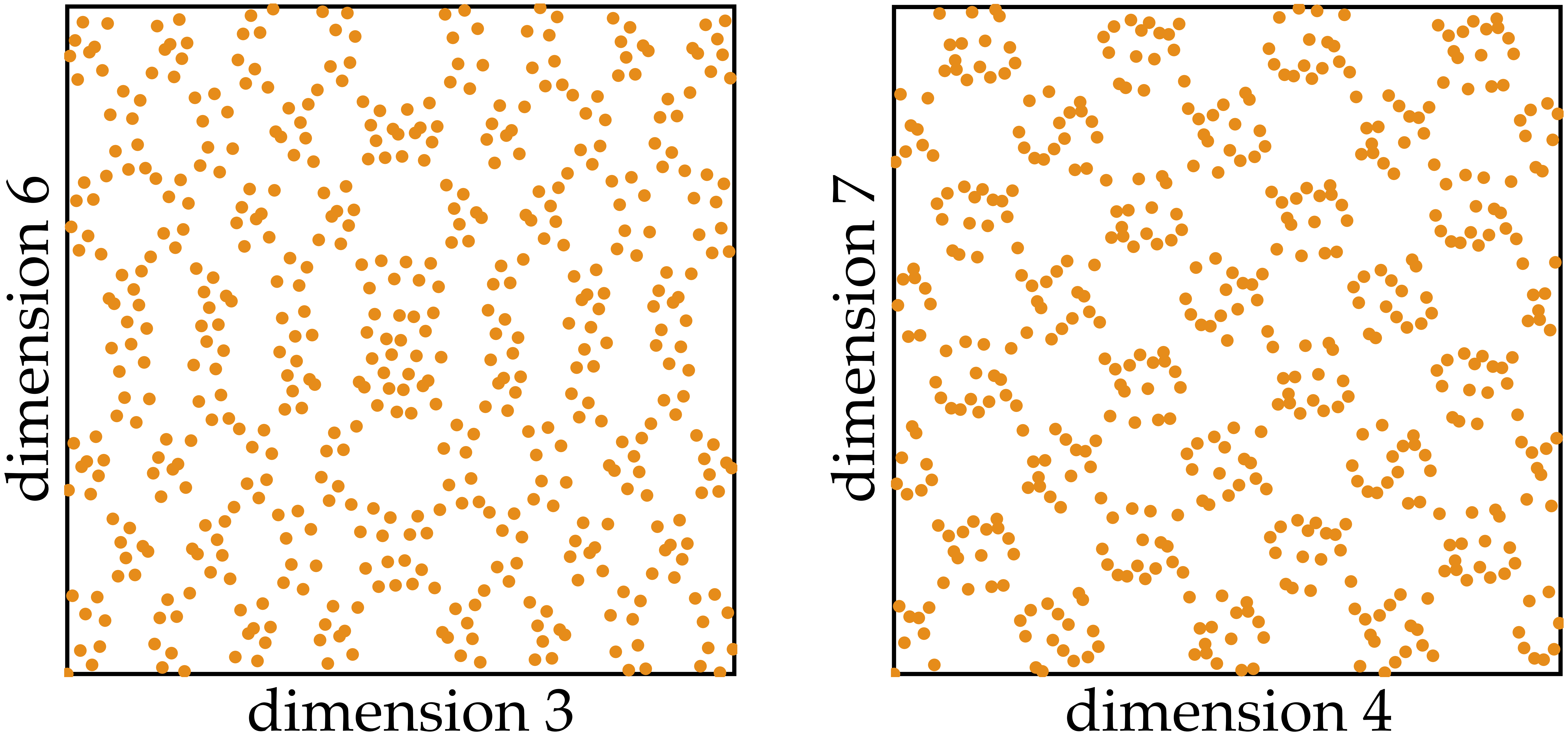
High-dimensional QMC (Sobol, Halton)

✘ Higher dimensions rarely as well-stratified as first two

High-dimensional QMC (Sobol, Halton)

✘ Higher dimensions rarely as well-stratified as first two

- Sobol:

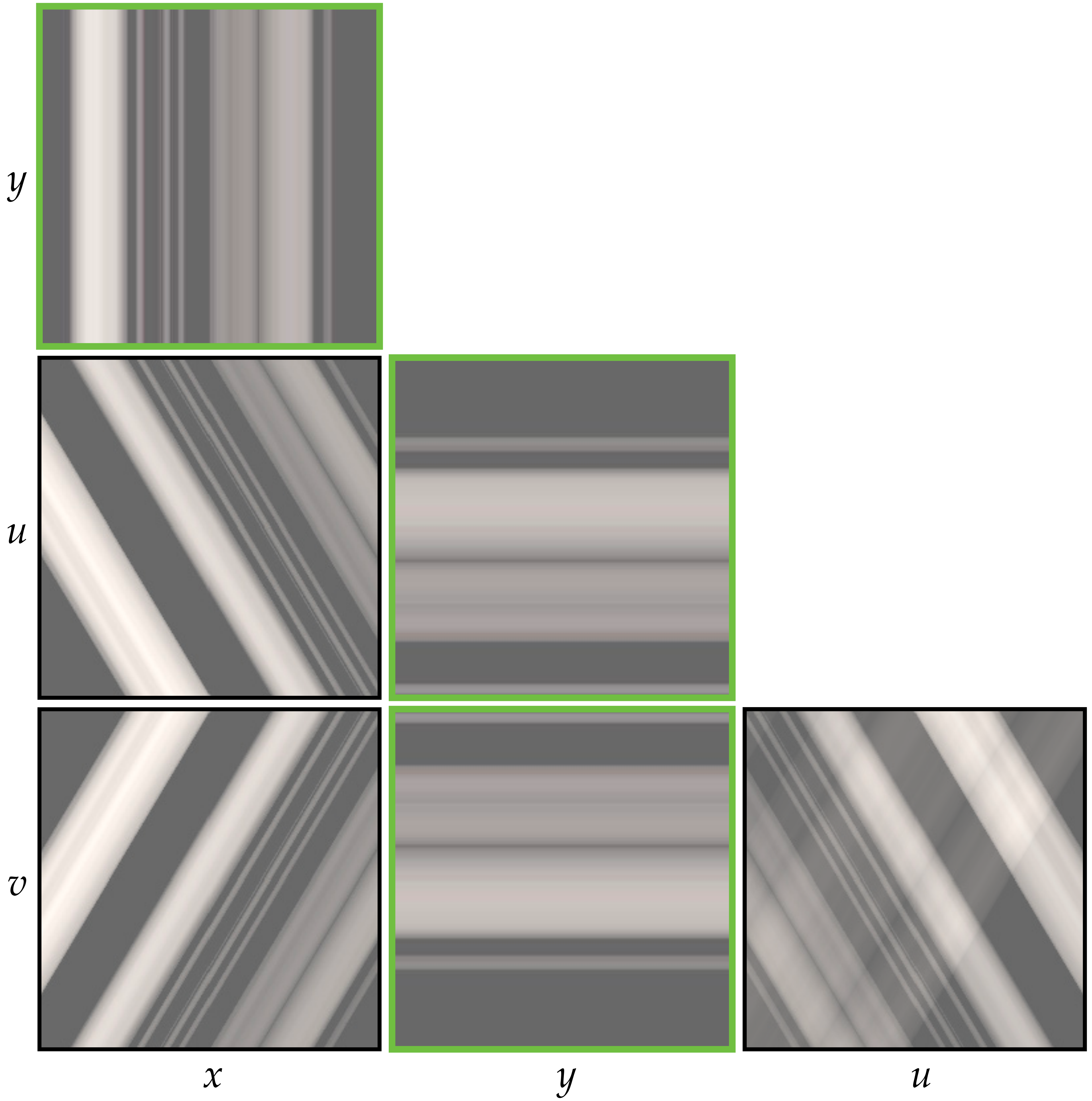


PBRT

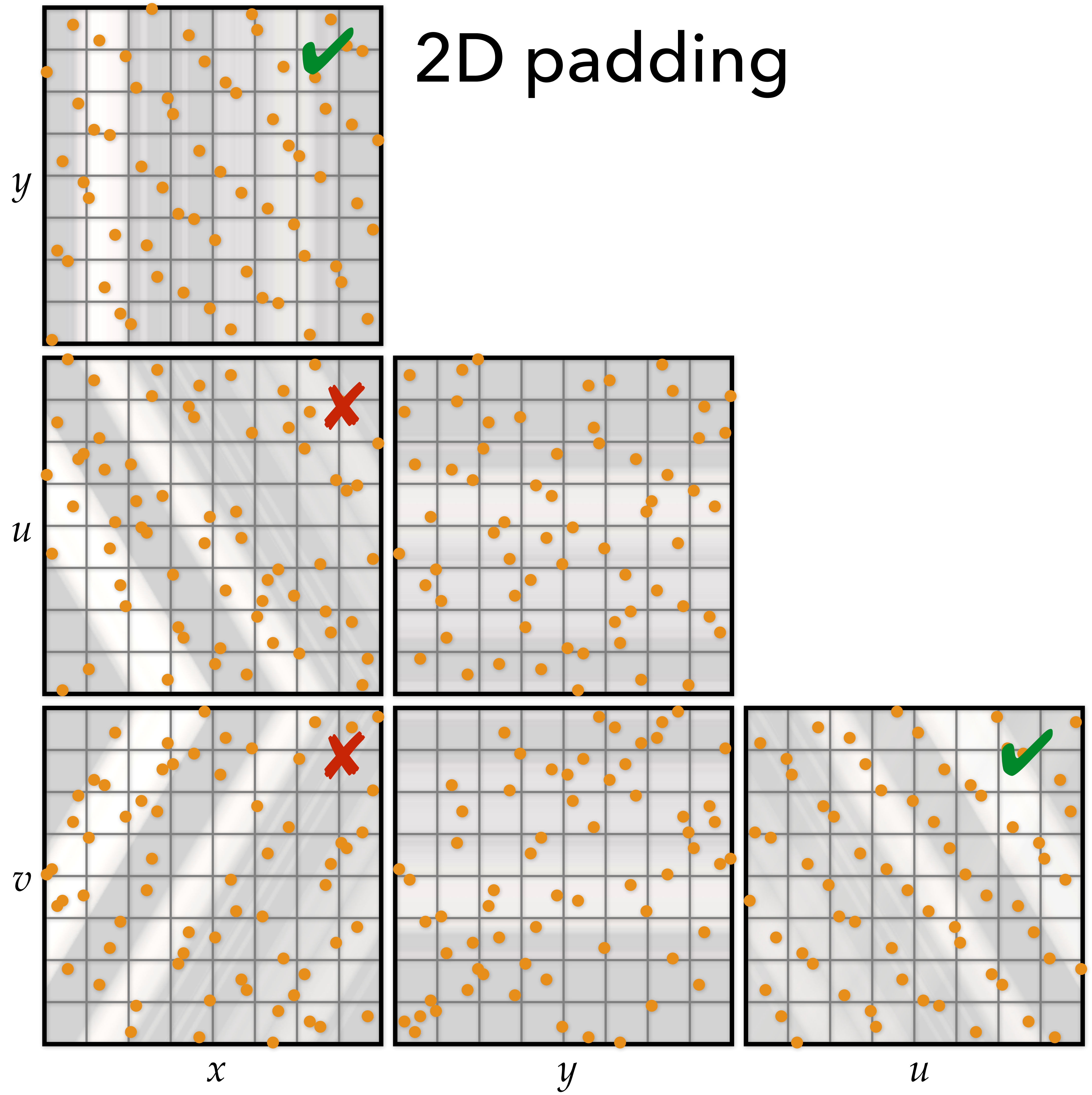
High-dimensional Sobol sampler

✘ Structured artifacts

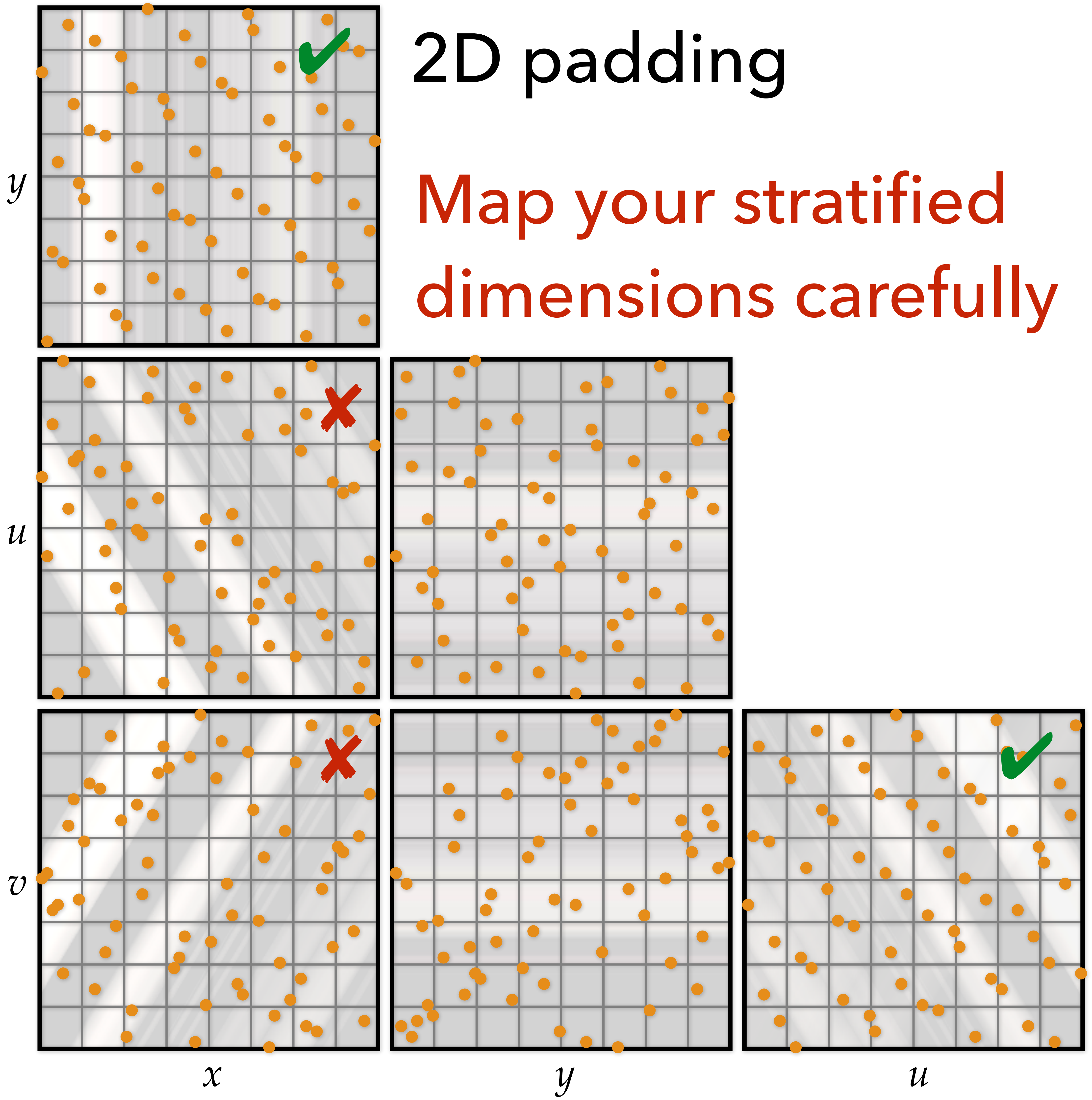
Which dimensions matter?



Which dimensions matter?



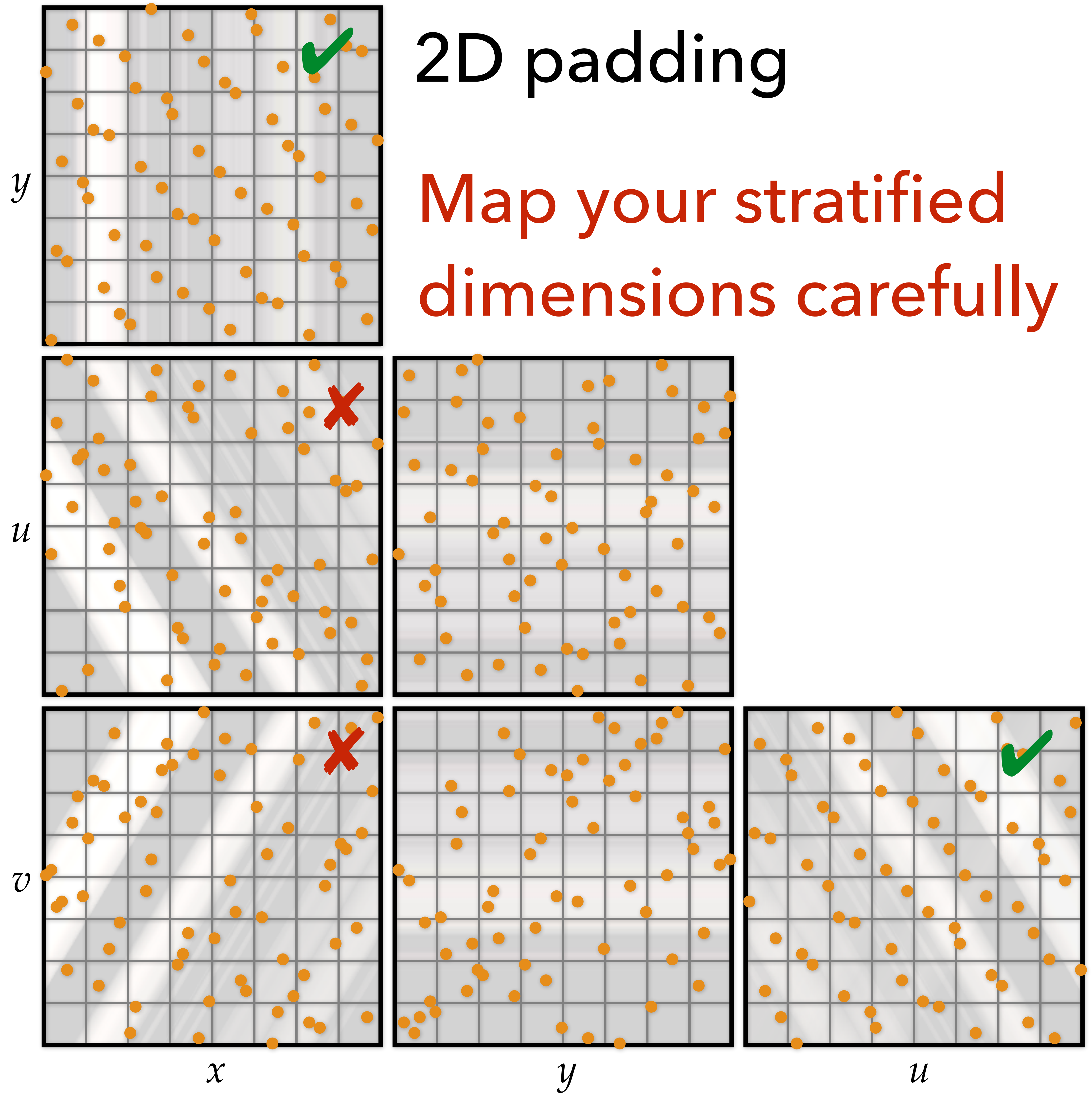
Which dimensions matter?

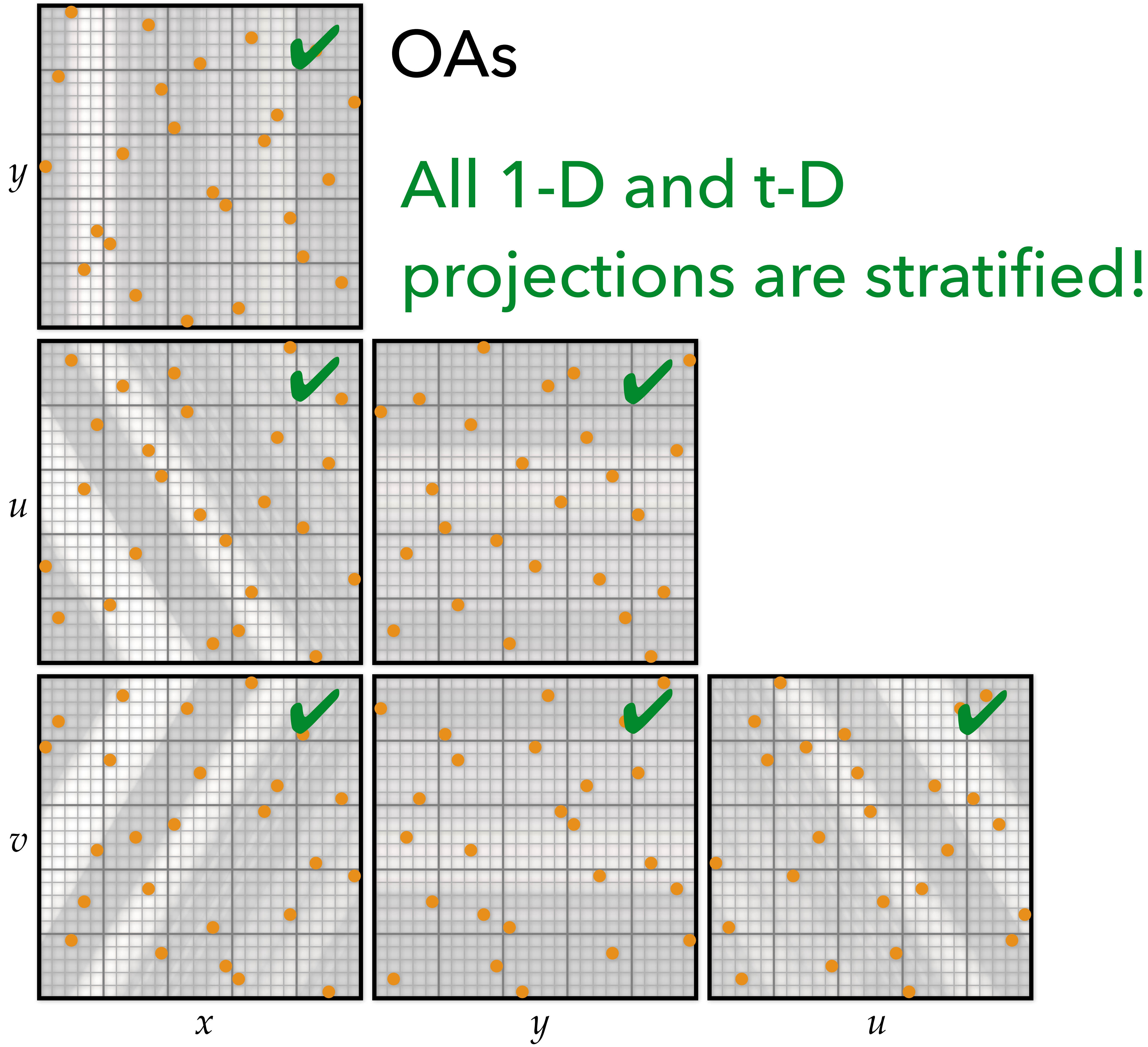


2D padding

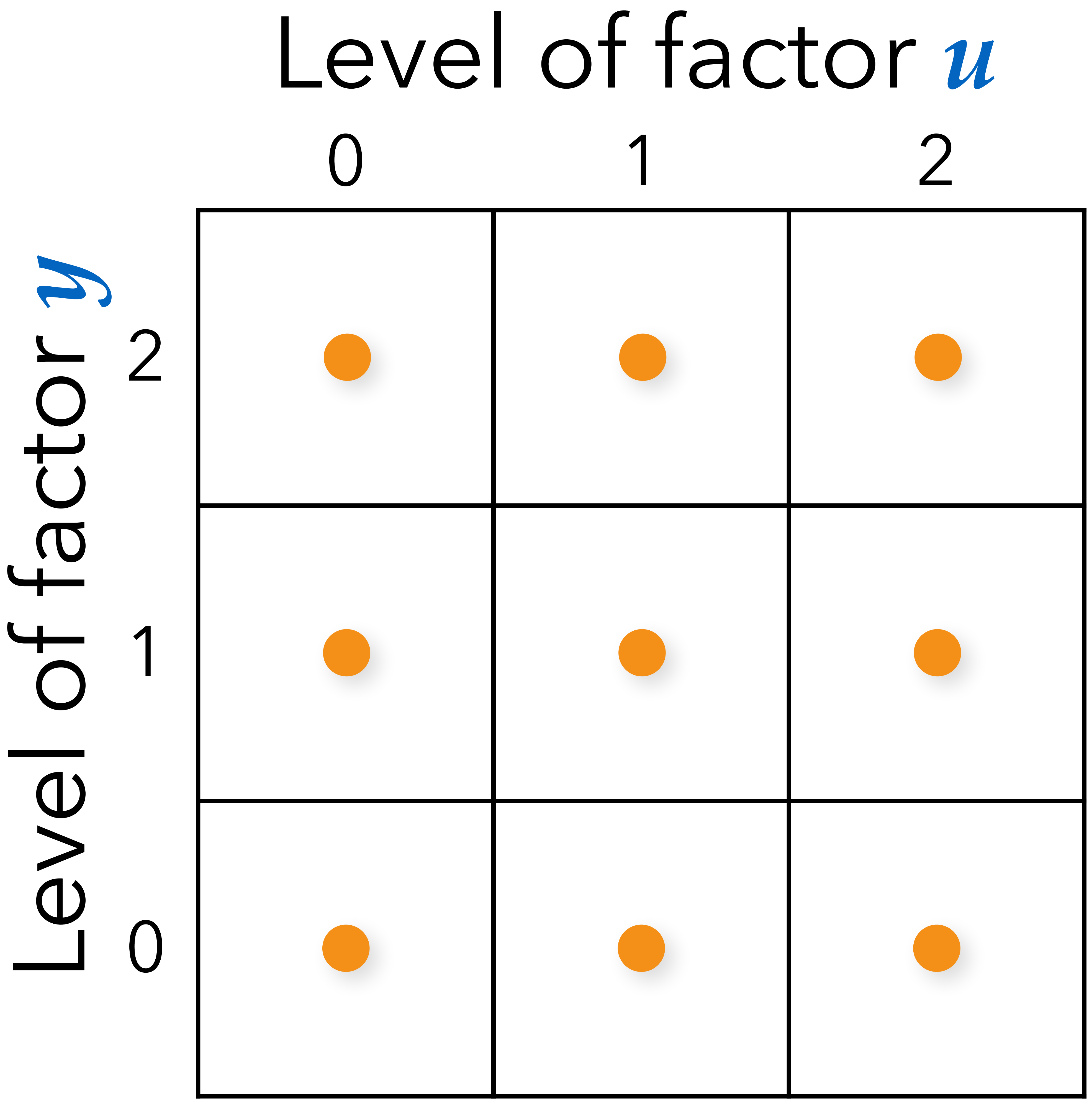
Map your stratified dimensions carefully

Which dimensions matter?



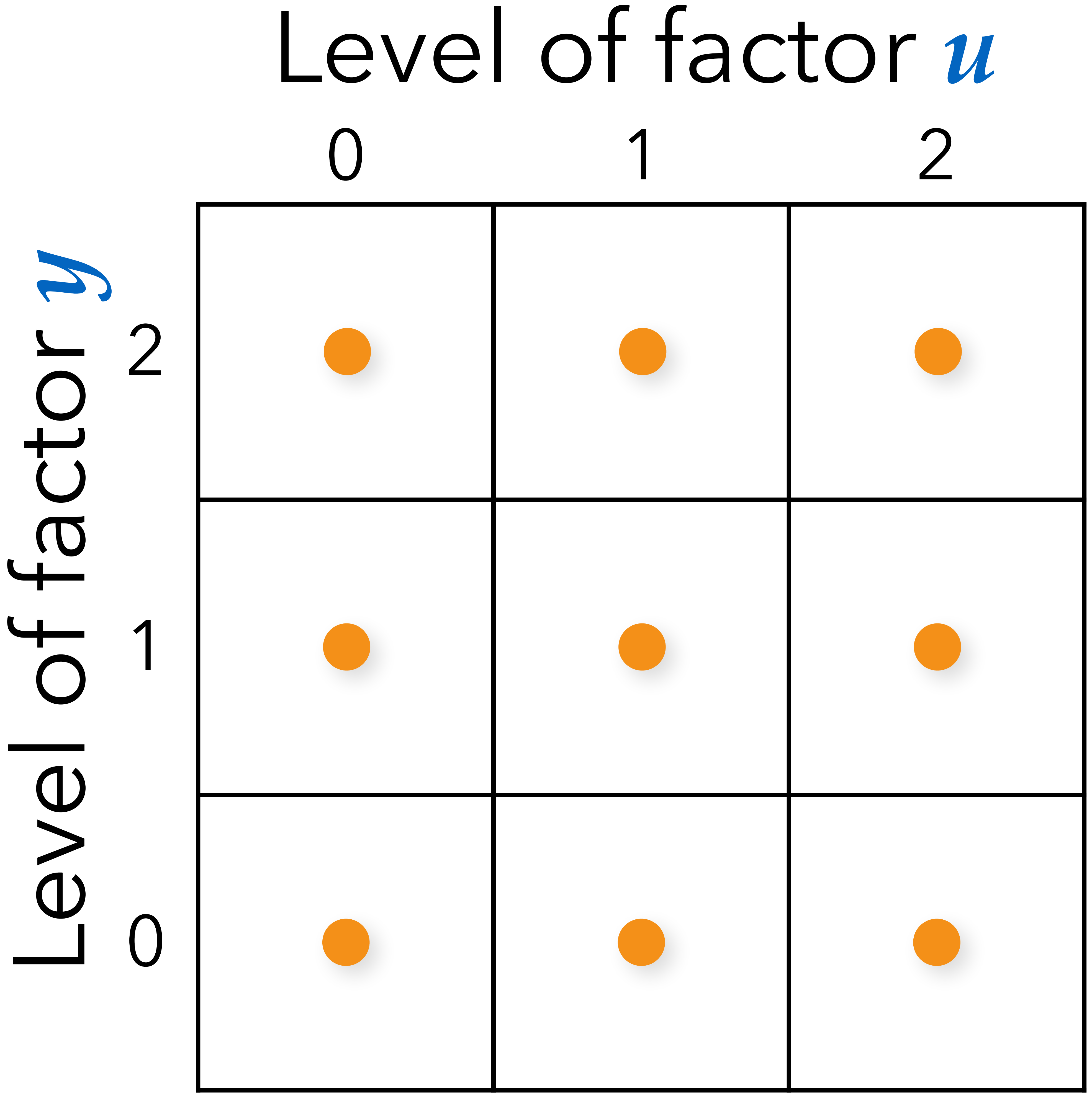


Monte Carlo using OAs



Monte Carlo using OAs

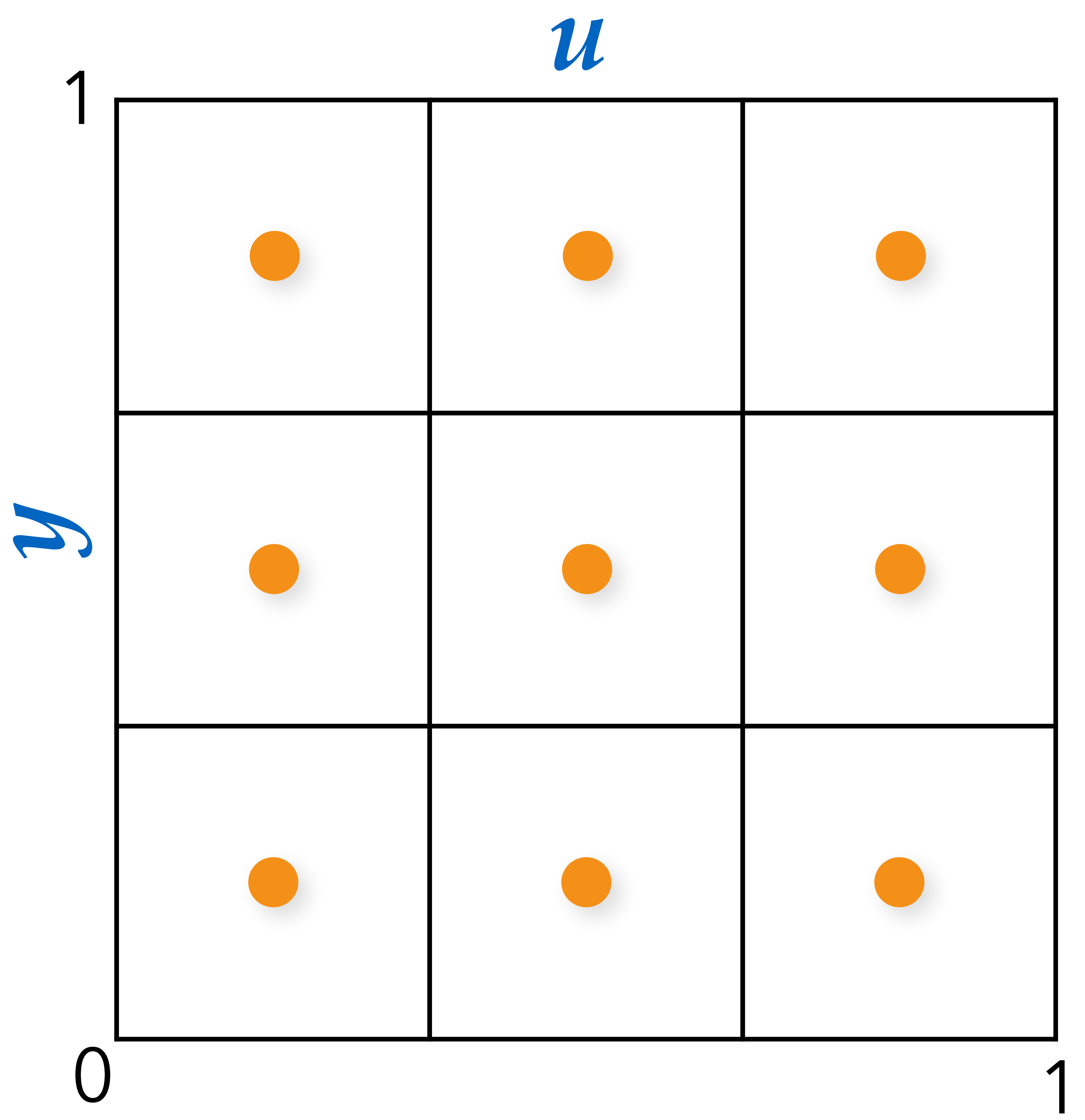
If A_{ij} denotes j^{th} factor in i^{th} run (j^{th} dimension of i^{th} point), then:



Monte Carlo using OAs

If A_{ij} denotes j^{th} factor in i^{th} run (j^{th} dimension of i^{th} point), then:

$$X_{ij} = \frac{A_{ij} + 0.5}{s} \in [0,1)^d$$



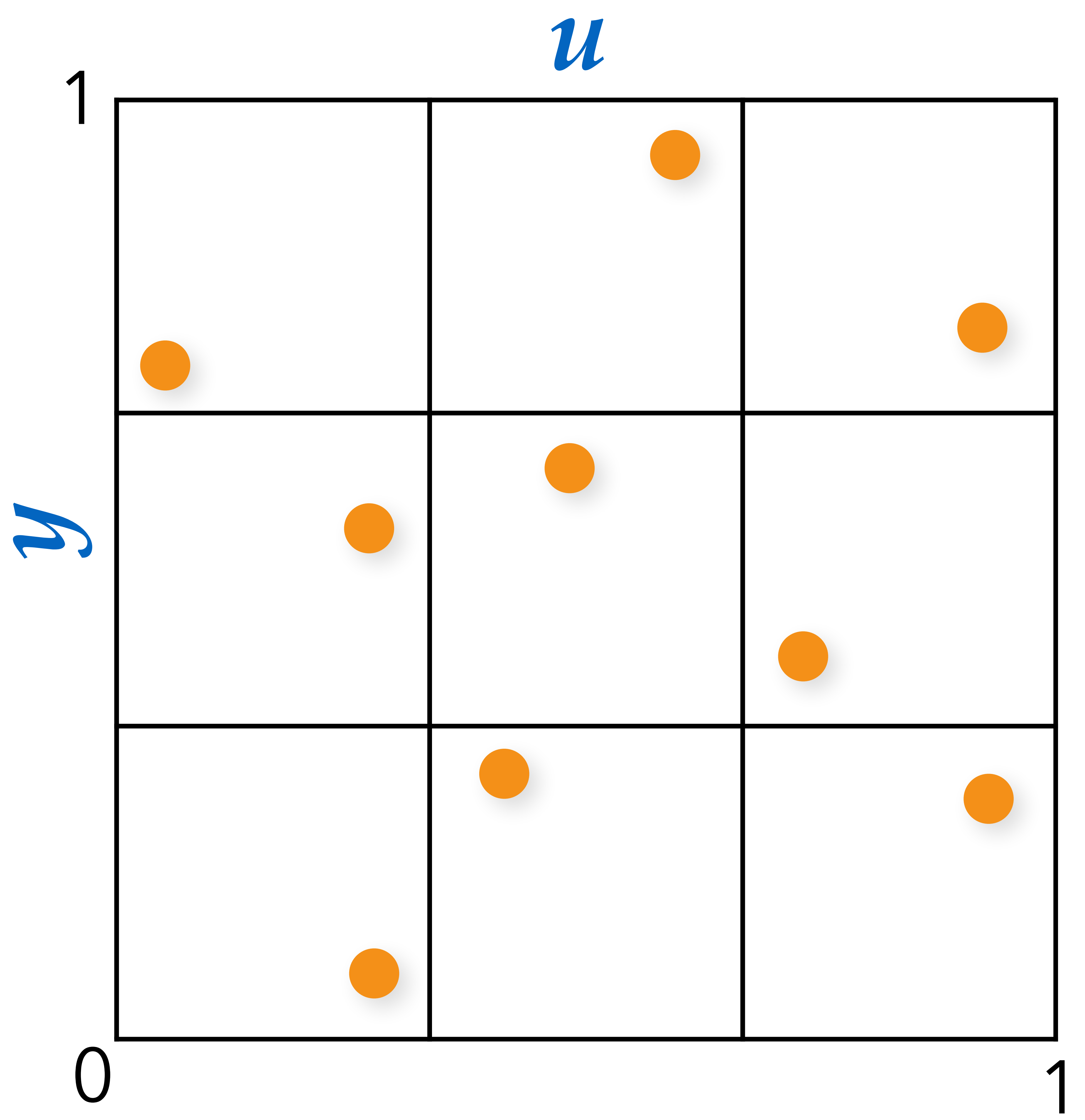
Monte Carlo using OAs

[Owen 92]
OA-based jittered

If A_{ij} denotes j^{th} factor in i^{th} run (j^{th} dimension of i^{th} point), then:

$$X_{ij} = \frac{A_{ij} + \xi_{ij}}{s} \in [0,1)^d$$

Jittered sampling



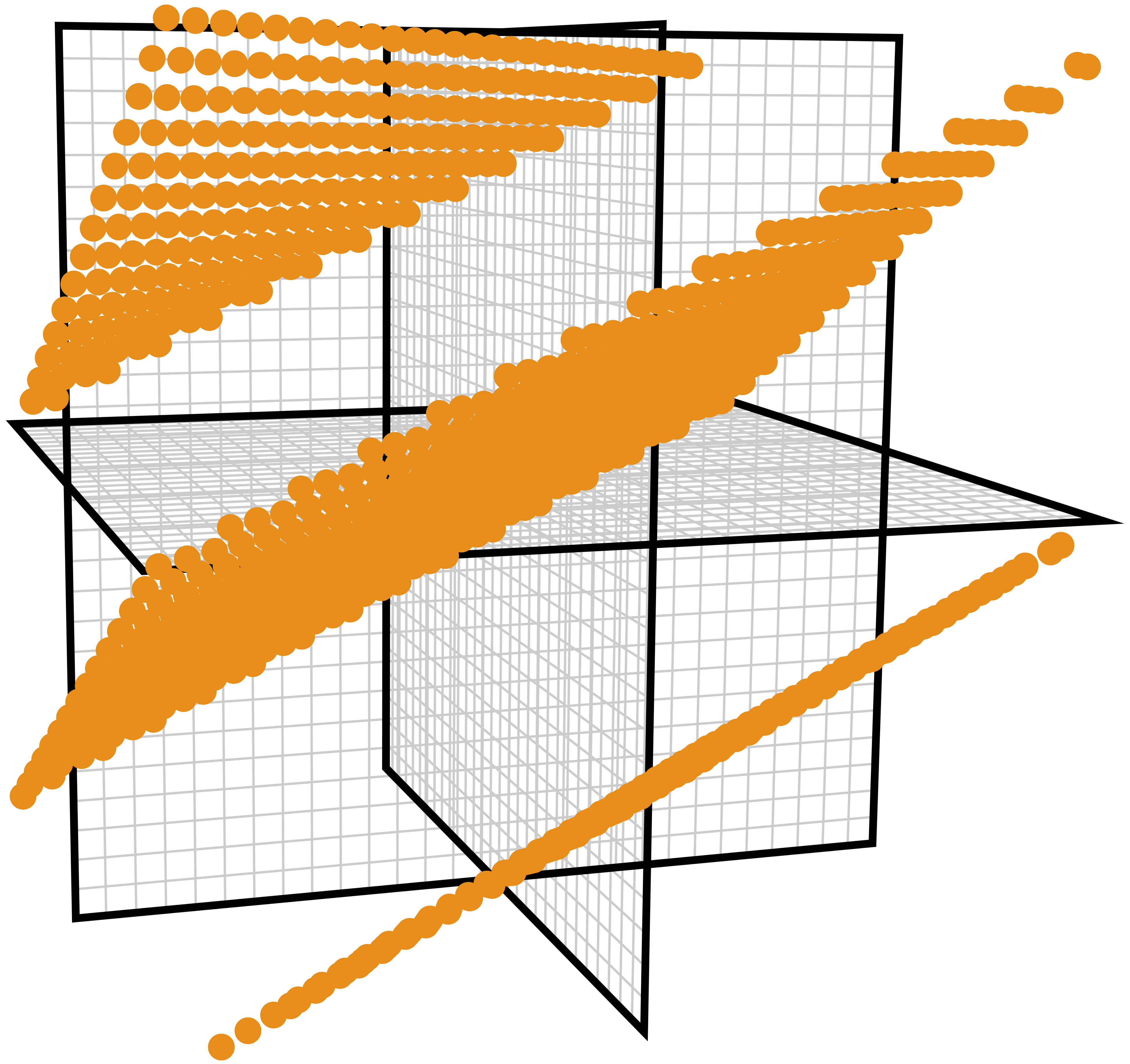
Monte Carlo using OAs

[Owen 92]
OA-based jittered

If A_{ij} denotes j^{th} factor in i^{th} run (j^{th} dimension of i^{th} point), then:

$$X_{ij} = \frac{A_{ij} + \xi_{ij}}{s} \in [0,1)^d$$

Jittered sampling



Monte Carlo using OAs

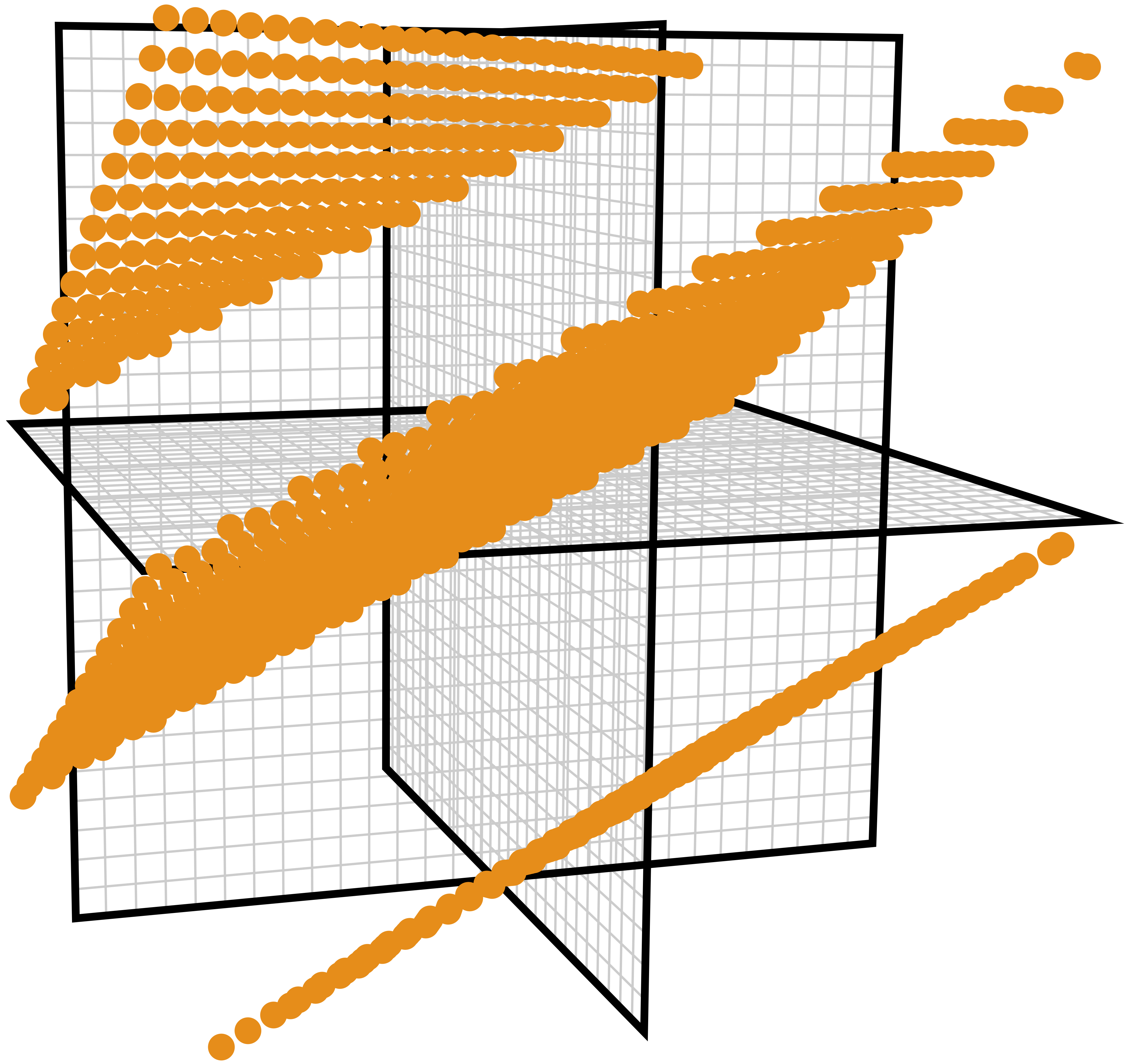
[Owen 92]
OA-based jittered

If A_{ij} denotes j^{th} factor in i^{th} run (j^{th} dimension of i^{th} point), then:

$$X_{ij} = \frac{\pi_j(A_{ij}) + \xi_{ij}}{s} \in [0,1)^d$$

different pseudo-random permutation of s levels for each dimension j

Jittered sampling



Monte Carlo using OAs

[Owen 92]
OA-based jittered

If A_{ij} denotes j^{th} factor in i^{th} run (j^{th} dimension of i^{th} point), then:

$$X_{ij} = \frac{\pi_j(A_{ij}) + \xi_{ij}}{s} \in [0,1)^d$$

Jittered sampling