IBM Research

Lattice-based cryptography (I)

Thijs Laarhoven

mail@thijs.com
http://www.thijs.com/

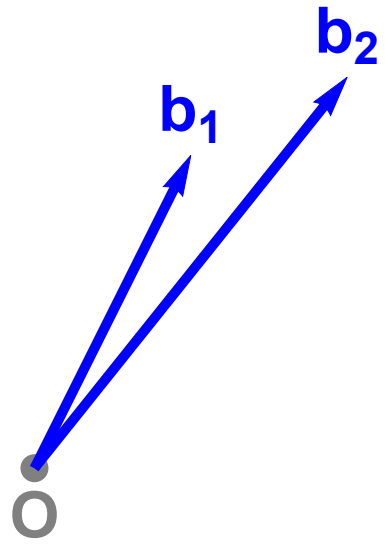
PQCrypto Executive School 2017
(June 22, 2017)

Lattices

What is a lattice?

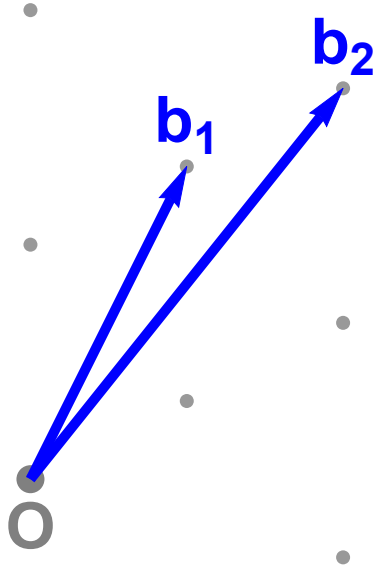
Lattices

What is a lattice?



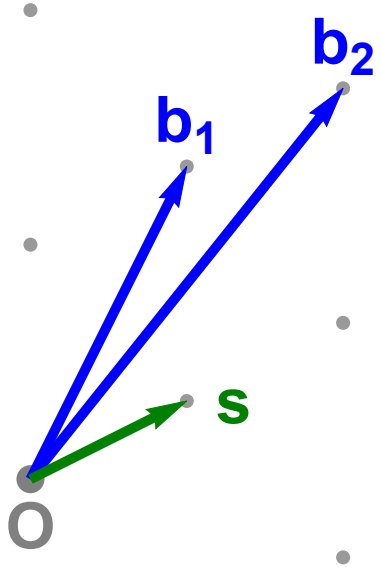
Lattices

What is a lattice?



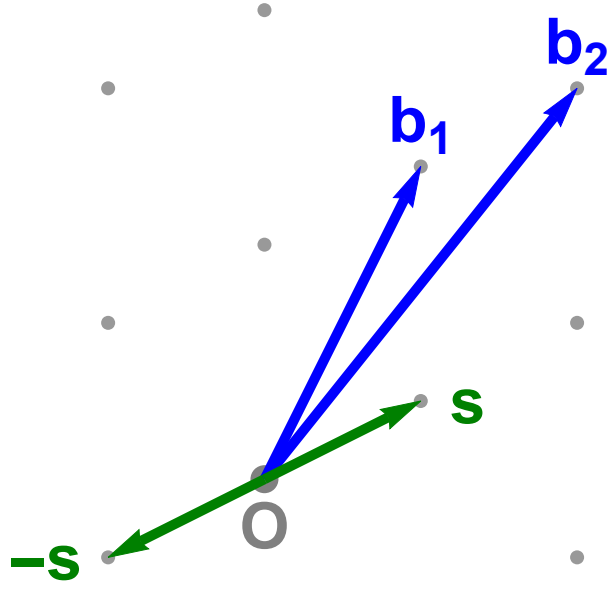
Lattices

Shortest Vector Problem (SVP)



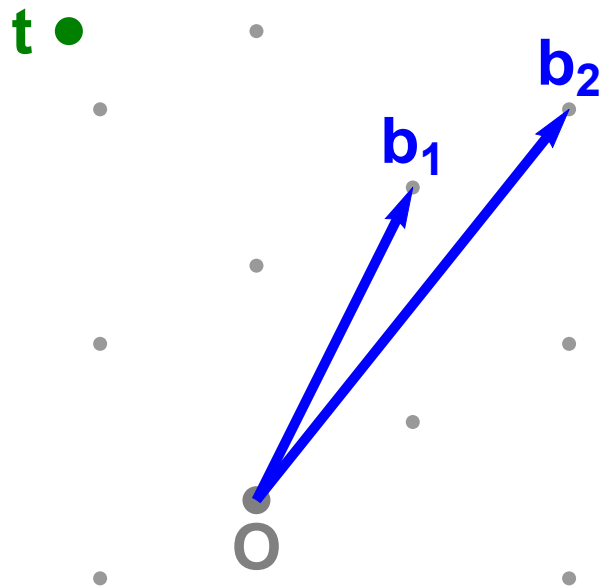
Lattices

Shortest Vector Problem (SVP)



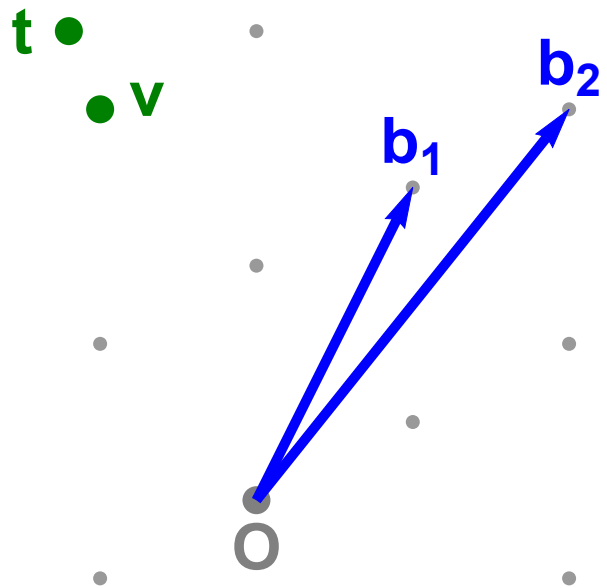
Lattices

Closest Vector Problem (CVP)



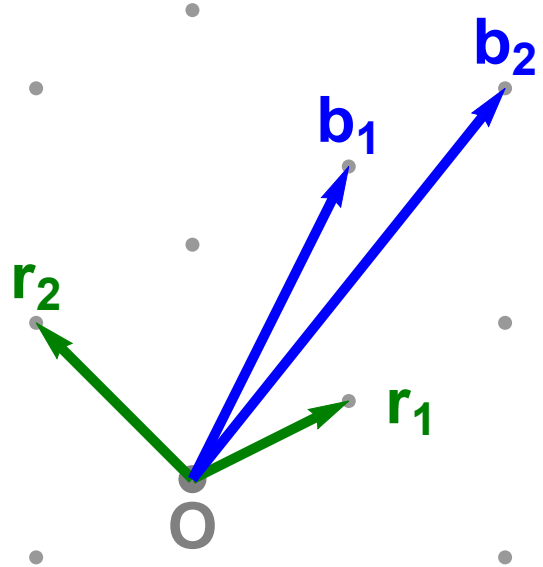
Lattices

Closest Vector Problem (CVP)



Lattices

Lattice basis reduction



Outline

Motivation: GGH encryption

Lattice basis reduction

- Gauss reduction
- LLL reduction
- BKZ reduction

Exact SVP algorithms

- Enumeration algorithms
- Sieving algorithms
- Practical comparison

Outline

- Motivation: GGH encryption
- Lattice basis reduction
  - Gauss reduction
  - LLL reduction
  - BKZ reduction
- Exact SVP algorithms
  - Enumeration algorithms
  - Sieving algorithms
  - Practical comparison

GGH cryptosystem

Overview

Private key: $R = \begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

Encrypt $\boldsymbol{m}$:

$$\boldsymbol{v} = \boldsymbol{m}B$$
$$\boldsymbol{c} = \boldsymbol{v} + \boldsymbol{e}$$

Decrypt $\boldsymbol{c}$:

$$\boldsymbol{v}' = \lfloor \boldsymbol{c}R^{-1} \rceil R$$
$$\boldsymbol{m}' = \boldsymbol{v}'B^{-1}$$

GGH cryptosystem

Private key

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

$v = mB$

$c = v + e$

Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v'B^{-1}$

GGH cryptosystem

Private key

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

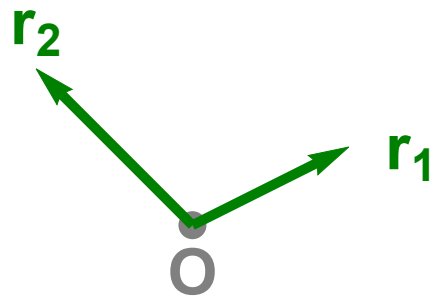
$v = mB$

$c = v + e$

Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v'B^{-1}$

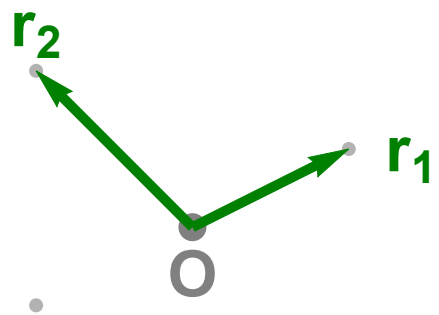


GGH cryptosystem

Public key

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$



Encrypt m:

$v = mB$

$c = v + e$

Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

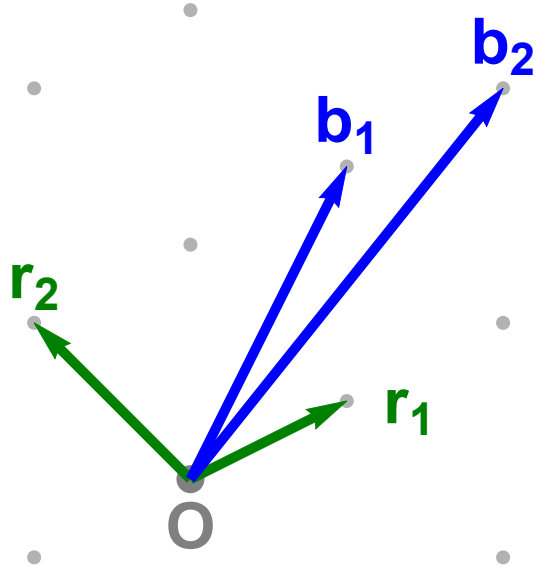
$m' = v'B^{-1}$

GGH cryptosystem

Public key

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$



Encrypt m:

$v = mB$

$c = v + e$

Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v'B^{-1}$

GGH cryptosystem

Encryption

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

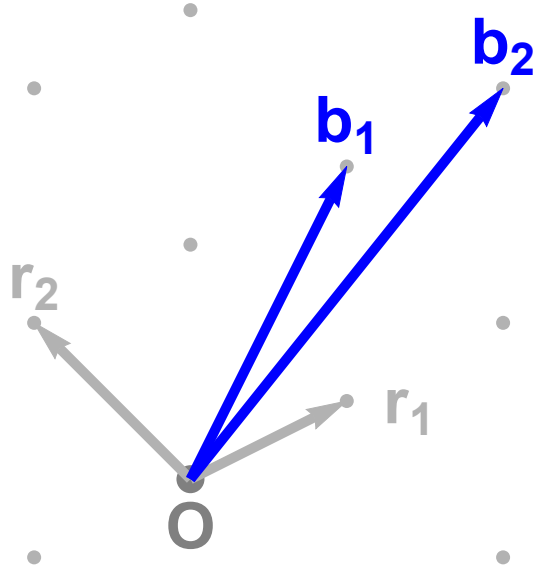
$$v = mB$$

$$c = v + e$$

Decrypt c:

$$v' = \lfloor cR^{-1} \rceil R$$

$$m' = v'B^{-1}$$



GGH cryptosystem

Encryption

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

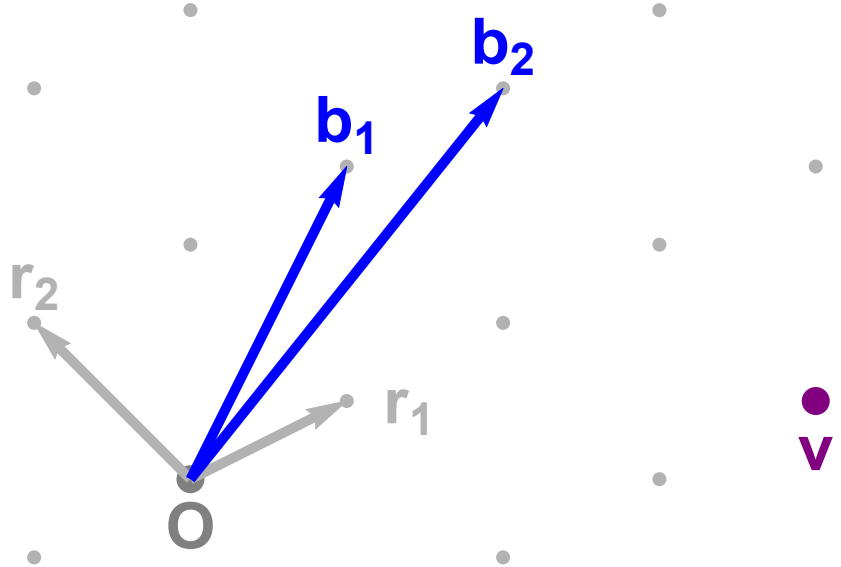
$v = mB$

$c = v + e$

Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v'B^{-1}$



GGH cryptosystem

Encryption

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

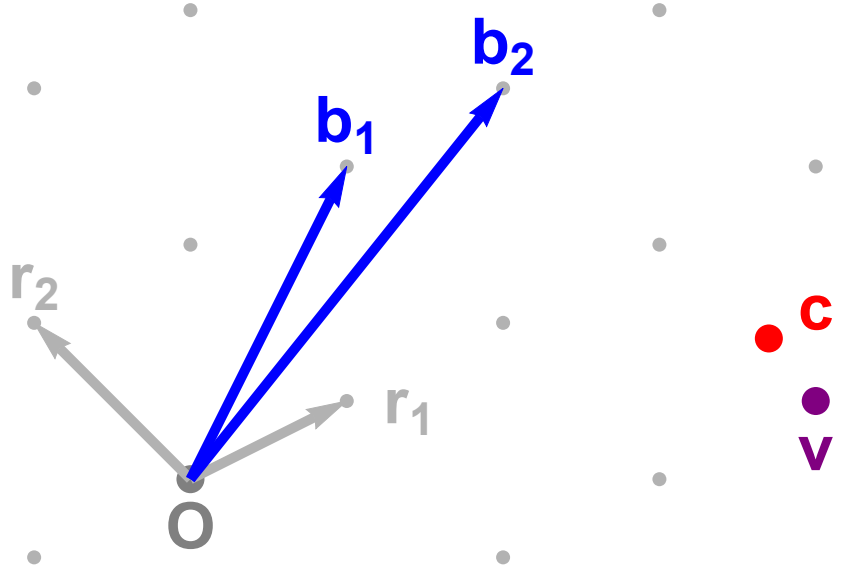
$v = mB$

$c = v + e$

Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v'B^{-1}$



GGH cryptosystem

Decryption with good basis

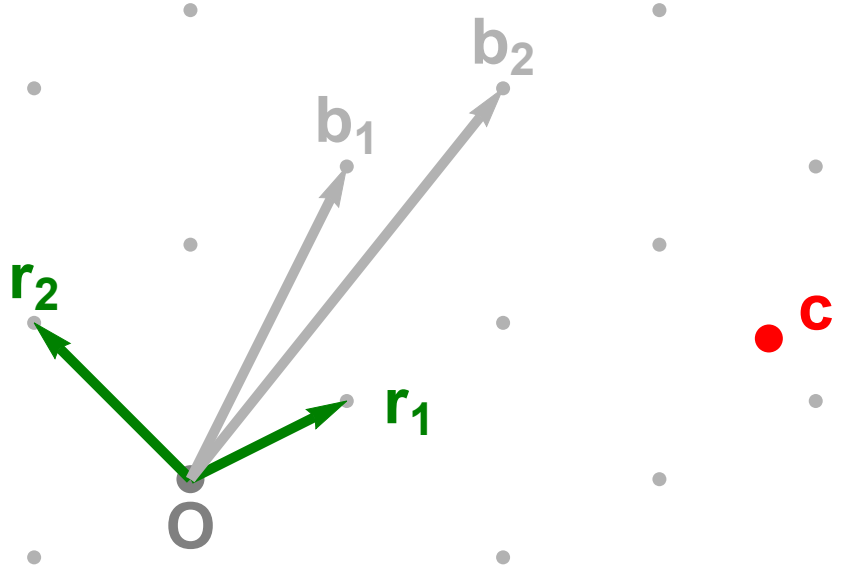
Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

$v = mB$

$c = v + e$



Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v'B^{-1}$

GGH cryptosystem

Decryption with good basis

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

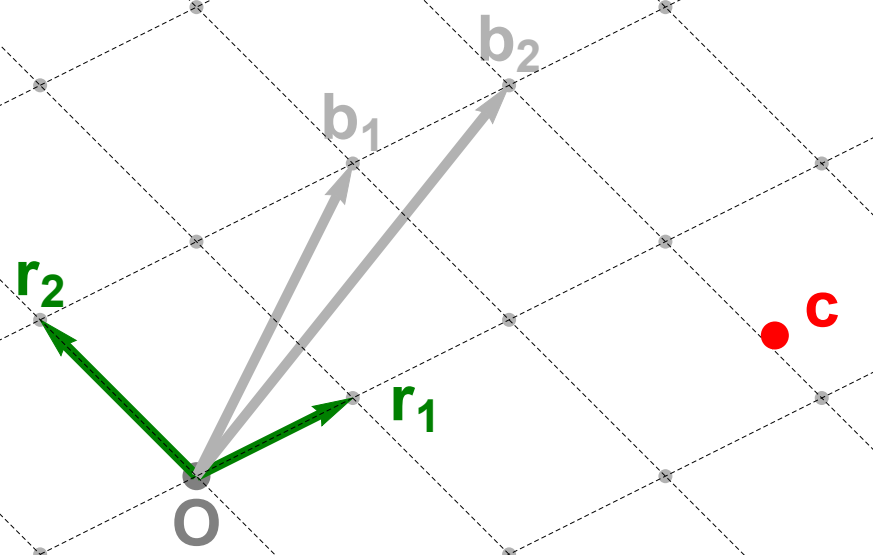
$v = mB$

$c = v + e$

Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v'B^{-1}$



GGH cryptosystem

Decryption with good basis

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

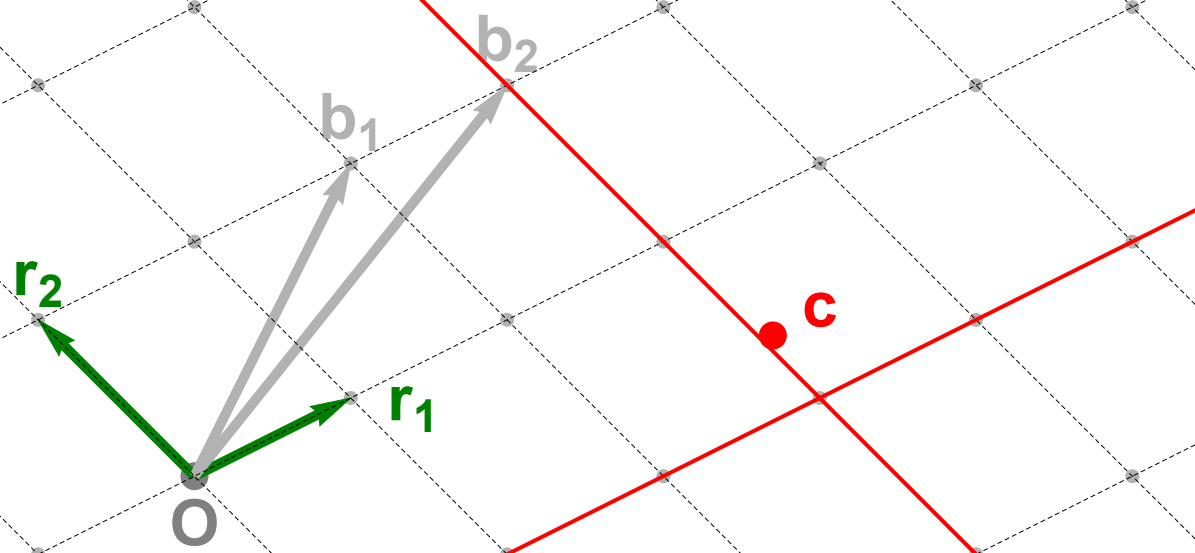
$v = mB$

$c = v + e$

Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v'B^{-1}$



GGH cryptosystem

Decryption with good basis

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

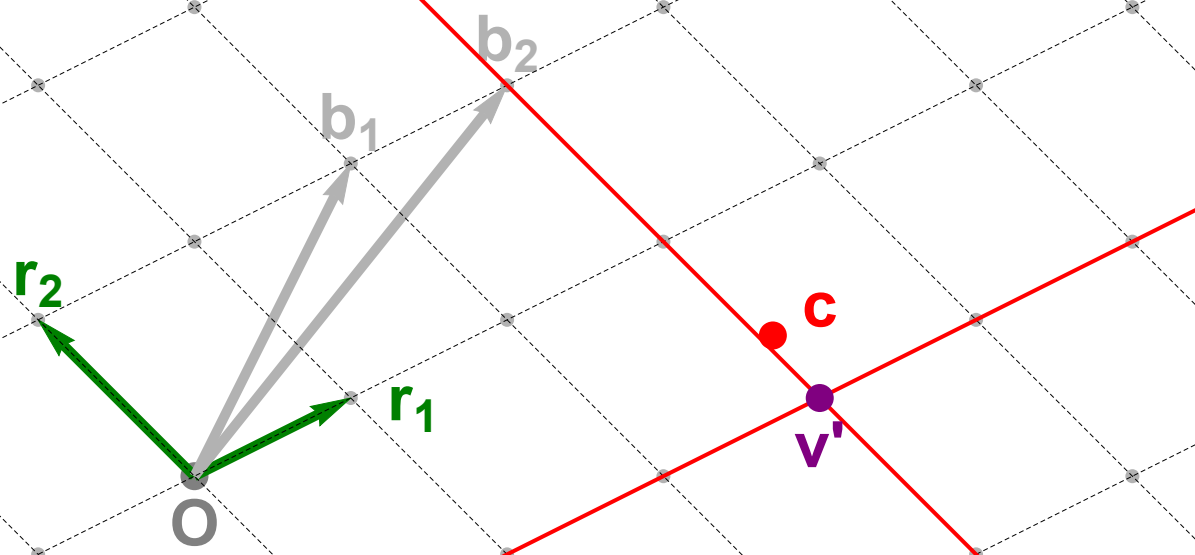
$v = mB$

$c = v + e$

Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v'B^{-1}$



GGH cryptosystem

Decryption with bad basis

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

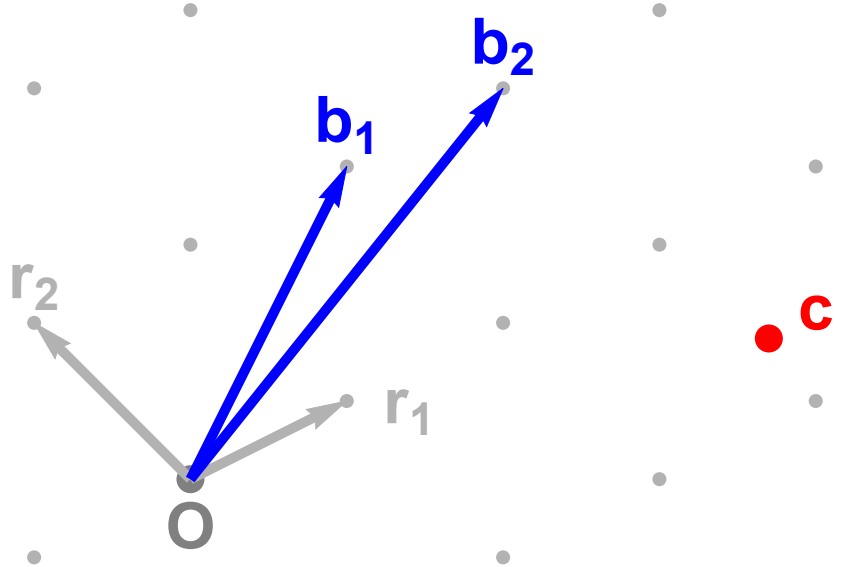
$v = mB$

$c = v + e$

Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v'B^{-1}$



GGH cryptosystem

Decryption with bad basis

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

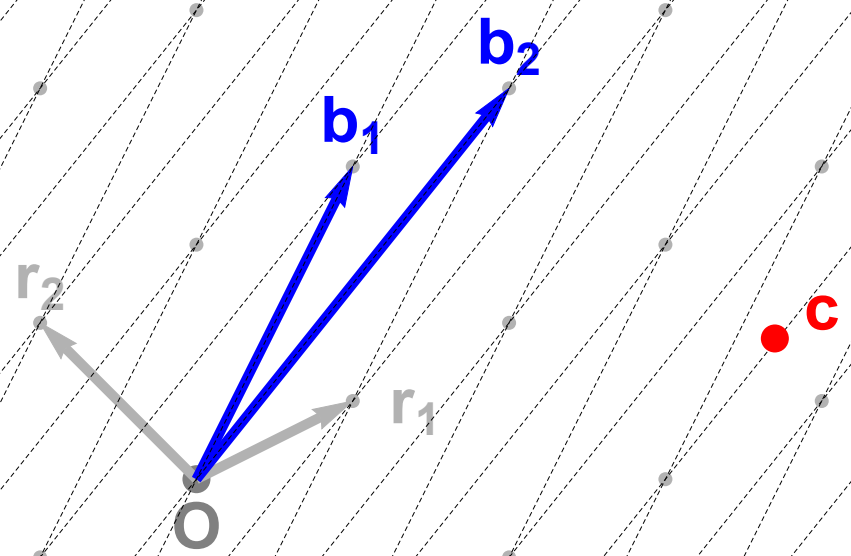
$$v = mB$$

$$c = v + e$$

Decrypt c:

$$v' = \lfloor cR^{-1} \rceil R$$

$$m' = v'B^{-1}$$



GGH cryptosystem

Decryption with bad basis

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

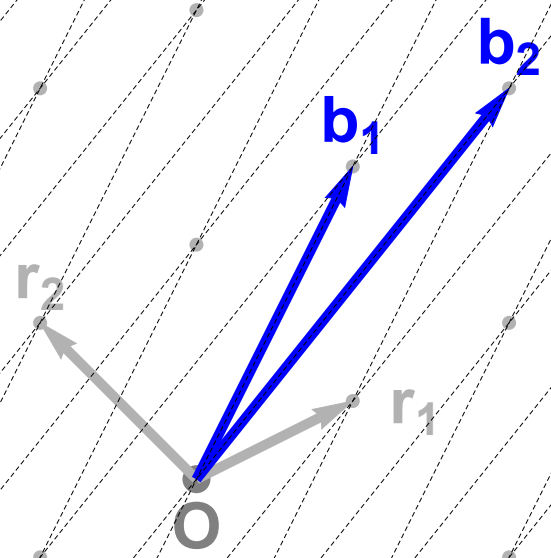
$$v = mB$$

$$c = v + e$$

Decrypt c:

$$v' = \lfloor cR^{-1} \rceil R$$

$$m' = v'B^{-1}$$



GGH cryptosystem

Decryption with bad basis

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

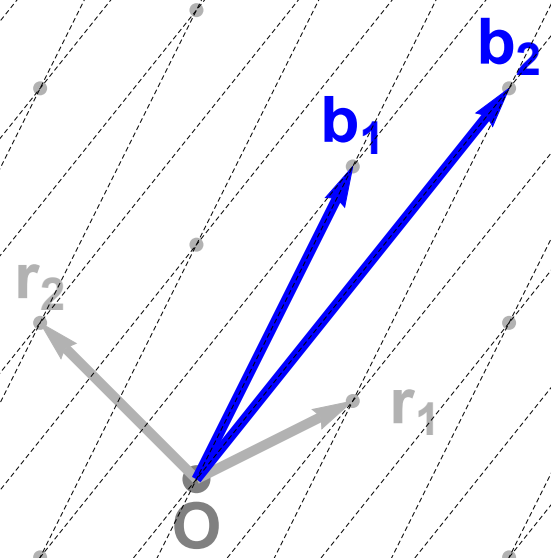
$v = mB$

$c = v + e$

Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v'B^{-1}$



GGH cryptosystem

Overview

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

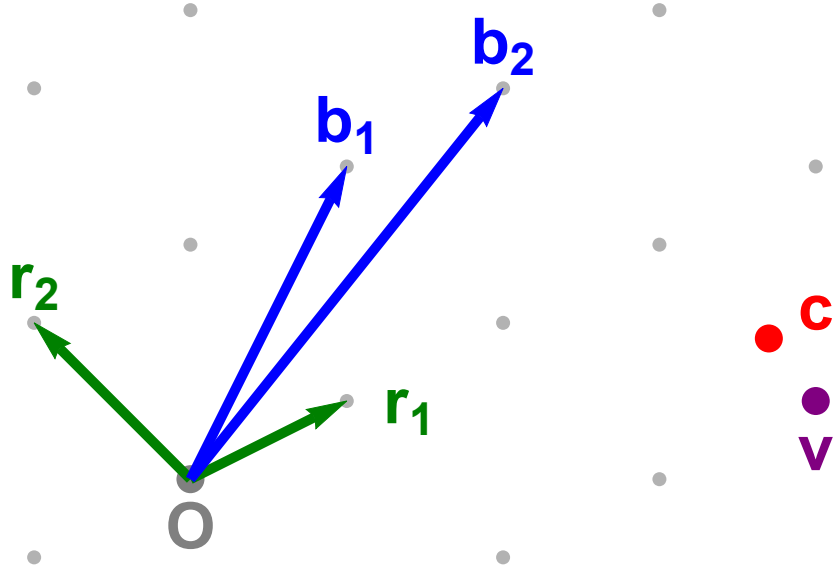
$v = mB$

$c = v + e$

Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v'B^{-1}$



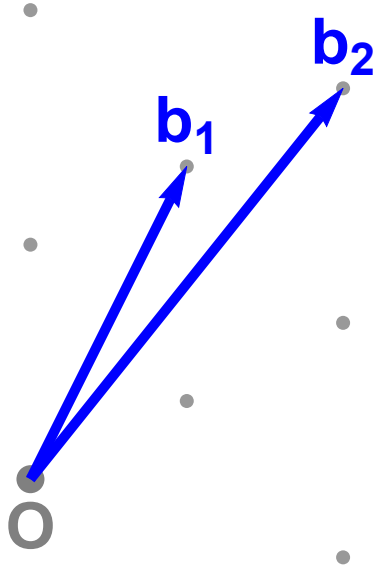
Outline

Motivation: GGH encryption

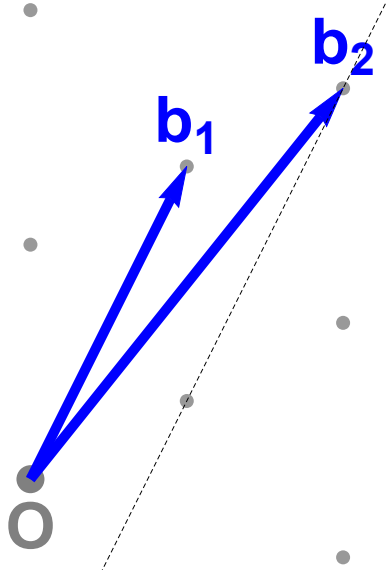
Lattice basis reduction
- Gauss reduction
- LLL reduction
- BKZ reduction

Exact SVP algorithms
- Enumeration algorithms
- Sieving algorithms
- Practical comparison

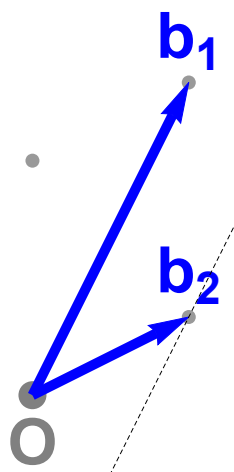
Gauss reduction



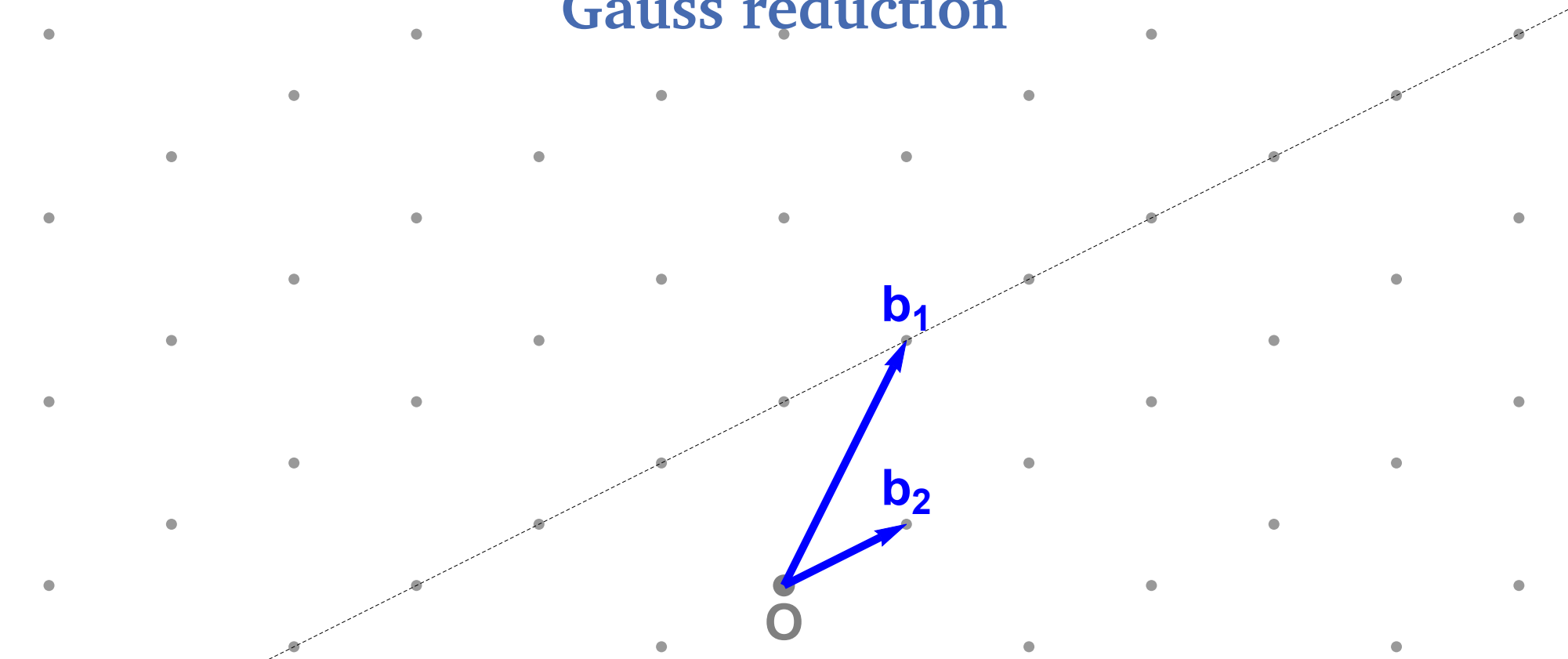
Gauss reduction



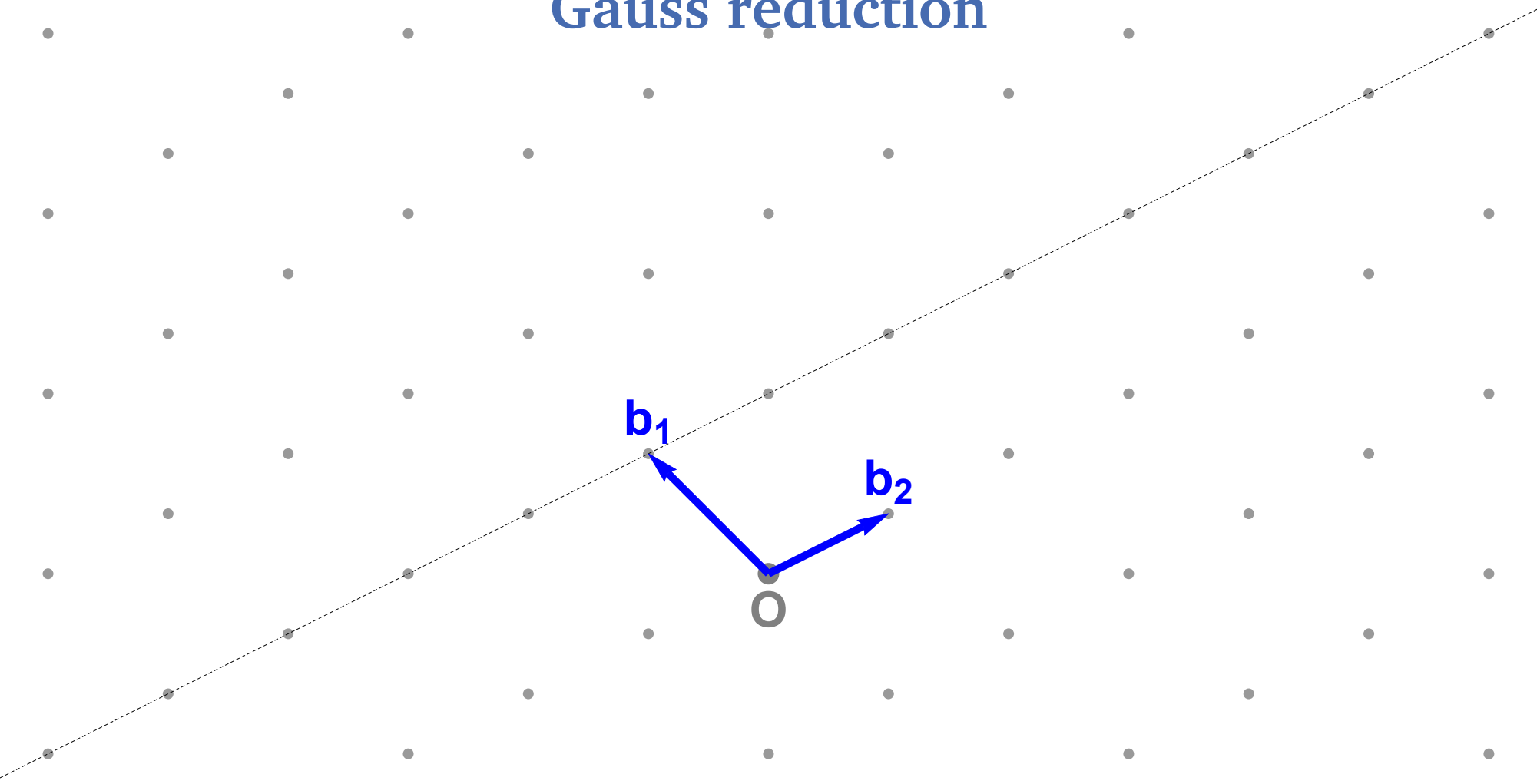
Gauss reduction



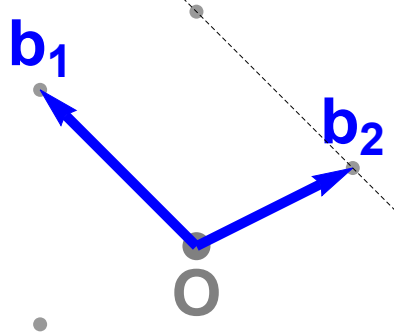
Gauss reduction



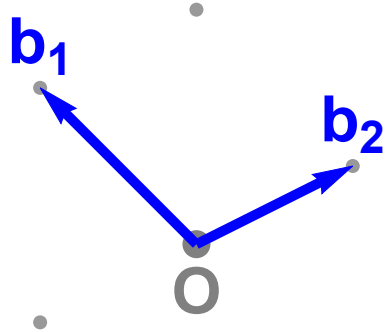
Gauss reduction



Gauss reduction



Gauss reduction



Gauss reduction

Given $\mathbf{B} = \{\boldsymbol{b}_1, \boldsymbol{b}_2\}$, repeat two steps:

- **Swap**: If $\|\boldsymbol{b}_1\| > \|\boldsymbol{b}_2\|$, then swap $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$.
- **Reduce**: While $\|\boldsymbol{b}_2 \pm \boldsymbol{b}_1\| < \|\boldsymbol{b}_2\|$, replace $\boldsymbol{b}_2 \leftarrow \boldsymbol{b}_2 \pm \boldsymbol{b}_1$.

Gauss reduction

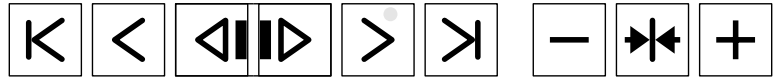
Given $\mathbf{B} = \{\boldsymbol{b}_1, \boldsymbol{b}_2\}$, repeat two steps:

- **Swap**: If $\|\boldsymbol{b}_1\| > \|\boldsymbol{b}_2\|$, then swap $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$.
- **Reduce**: While $\|\boldsymbol{b}_2 \pm \boldsymbol{b}_1\| < \|\boldsymbol{b}_2\|$, replace $\boldsymbol{b}_2 \leftarrow \boldsymbol{b}_2 \pm \boldsymbol{b}_1$.

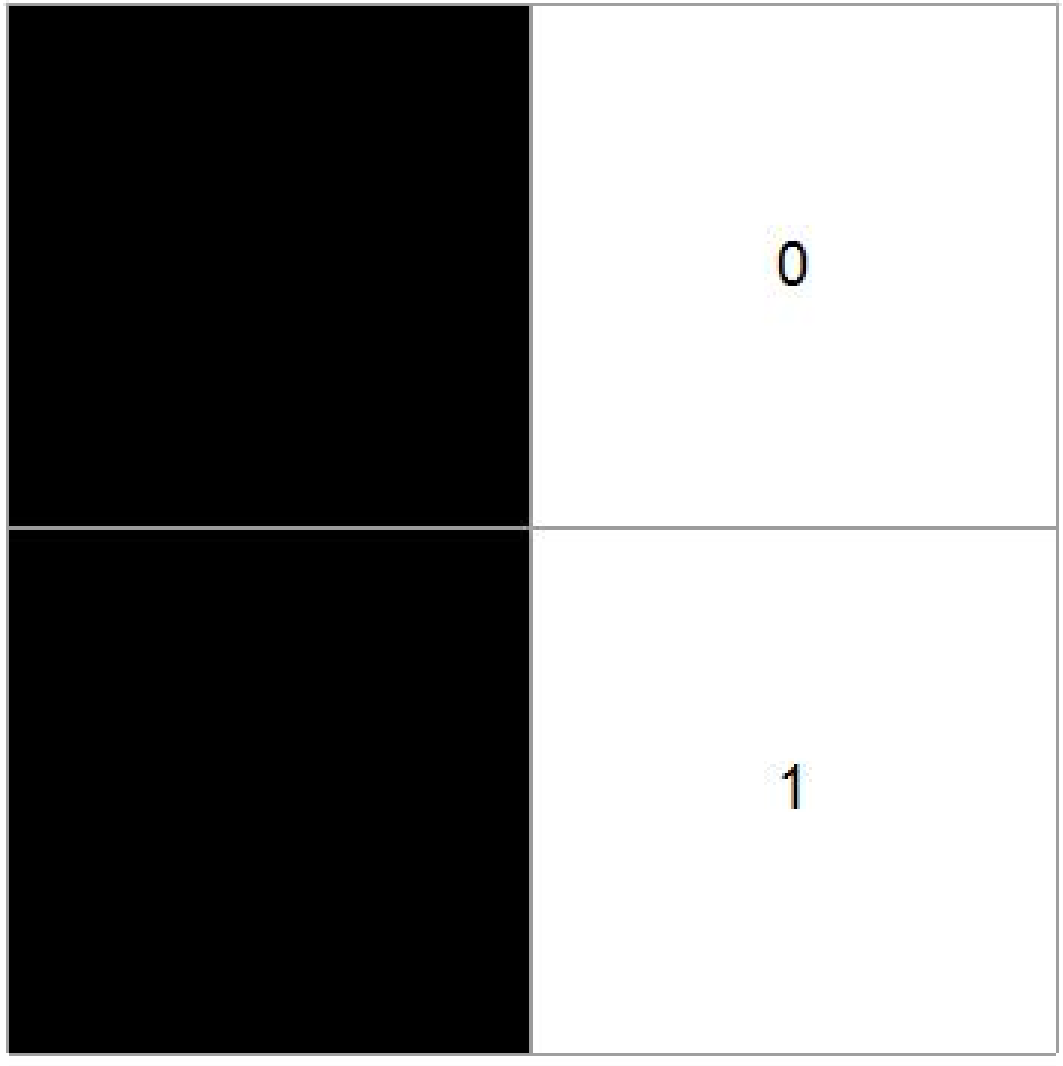
At the end, $\boldsymbol{b}_1$ is a shortest (non-zero) lattice vector and $\boldsymbol{b}_2$ a "second shortest" (non-zero) lattice vector.

Gauss reduction

3141592653	0
5897932384	1



Gauss reduction



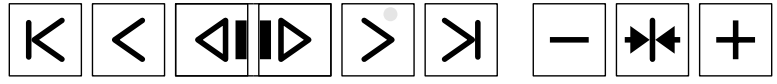
LLL algorithm

Lenstra-Lenstra-Lovasz (LLL) algorithm [LLL82]

- Blockwise generalization of Gauss reduction
- Do reductions/swaps on $(\boldsymbol{b}_i, \boldsymbol{b}_{i+1})$ for $i = 1, \ldots, n-1$

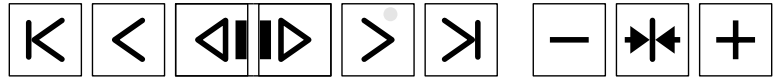
LLL algorithm

	0	0	0	0	0	0	0
	1	0	0	0	0	0	0
	0	1	0	0	0	0	0
	0	0	1	0	0	0	0
	0	0	0	1	0	0	0
	0	0	0	0	1	0	0
	0	0	0	0	0	1	0
	0	0	0	0	0	0	1

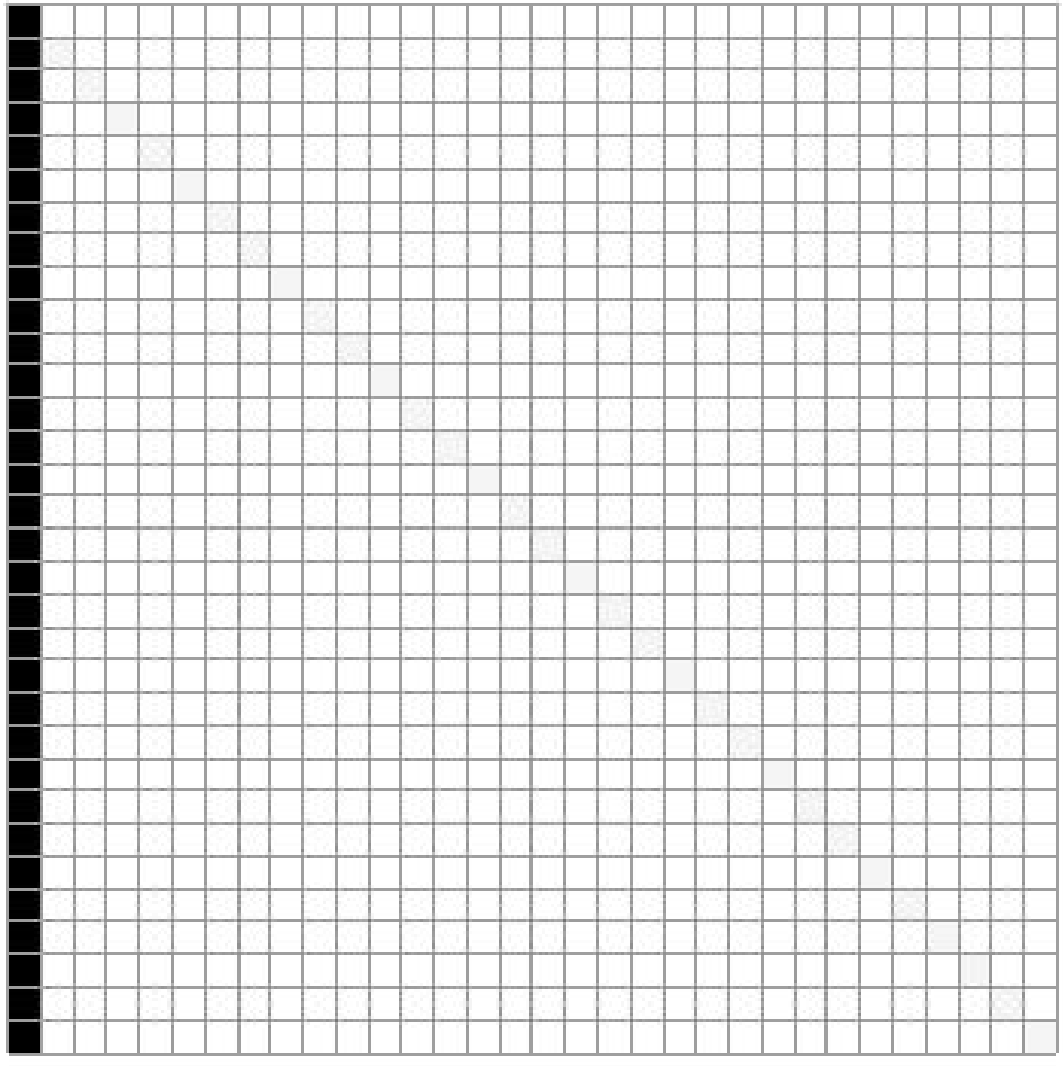


LLL algorithm

	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1



LLL algorithm



BKZ algorithm

Lenstra-Lenstra-Lovasz (LLL) algorithm [LLL82]

- Blockwise generalization of Gauss reduction
- Do reductions/swaps on $(\boldsymbol{b}_i, \boldsymbol{b}_{i+1})$ for $i = 1, \ldots, n-1$

BKZ algorithm

Lenstra-Lenstra-Lovasz (LLL) algorithm [LLL82]

- Blockwise generalization of Gauss reduction
- Do reductions/swaps on $(\boldsymbol{b}_i, \boldsymbol{b}_{i+1})$ for $i = 1, \ldots, n-1$
- Basis quality deteriorates with the dimension n
  - Theoretically: $\|\boldsymbol{b}_1\| \leq 1.075^n \cdot \det(\mathscr{L})$
  - Experimentally: $\|\boldsymbol{b}_1\| \approx 1.022^n \cdot \det(\mathscr{L})$

BKZ algorithm

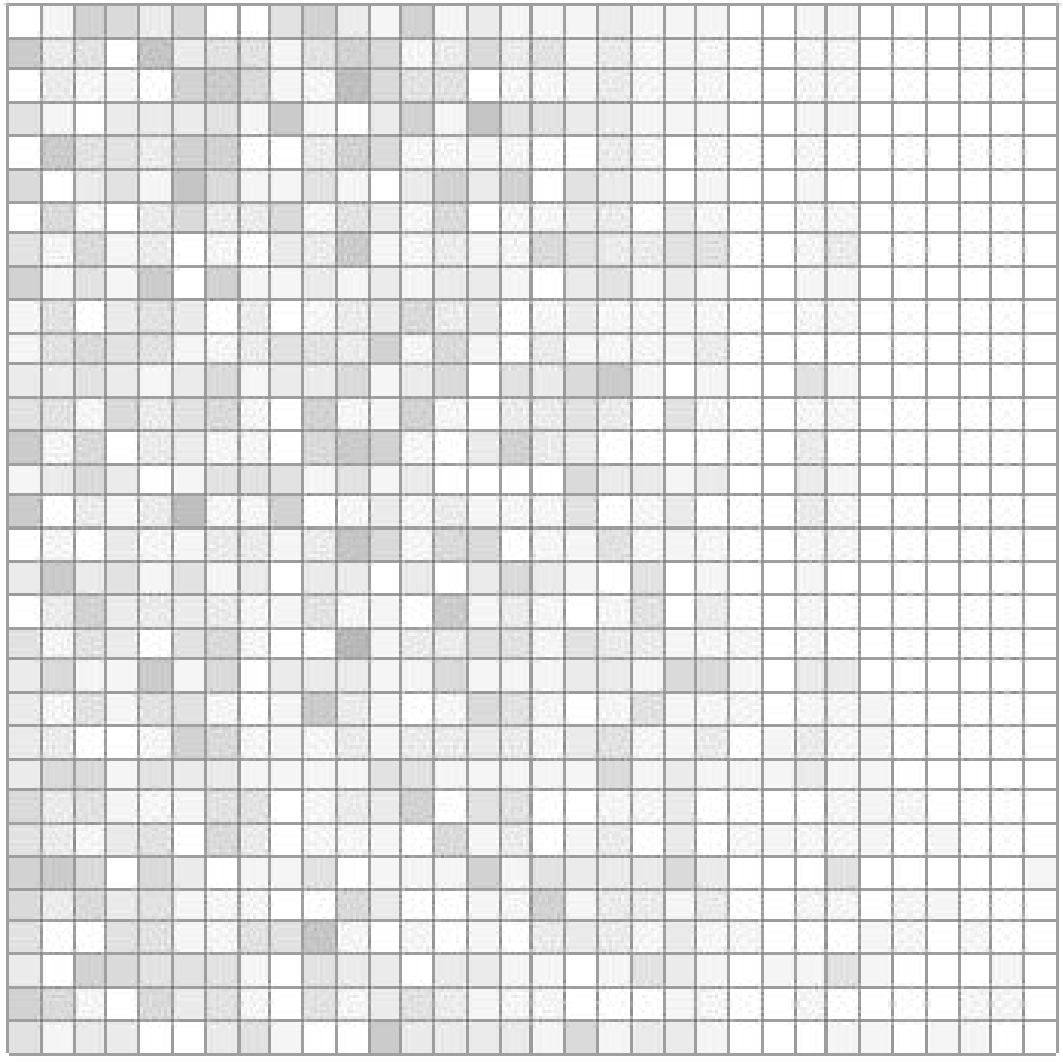
Lenstra-Lenstra-Lovasz (LLL) algorithm [LLL82]

- Blockwise generalization of Gauss reduction
- Do reductions/swaps on $(\boldsymbol{b}_i, \boldsymbol{b}_{i+1})$ for $i = 1, \ldots, n-1$
- Basis quality deteriorates with the dimension n
  - Theoretically: $\|\boldsymbol{b}_1\| \leq 1.075^n \cdot \det(\mathscr{L})$
  - Experimentally: $\|\boldsymbol{b}_1\| \approx 1.022^n \cdot \det(\mathscr{L})$

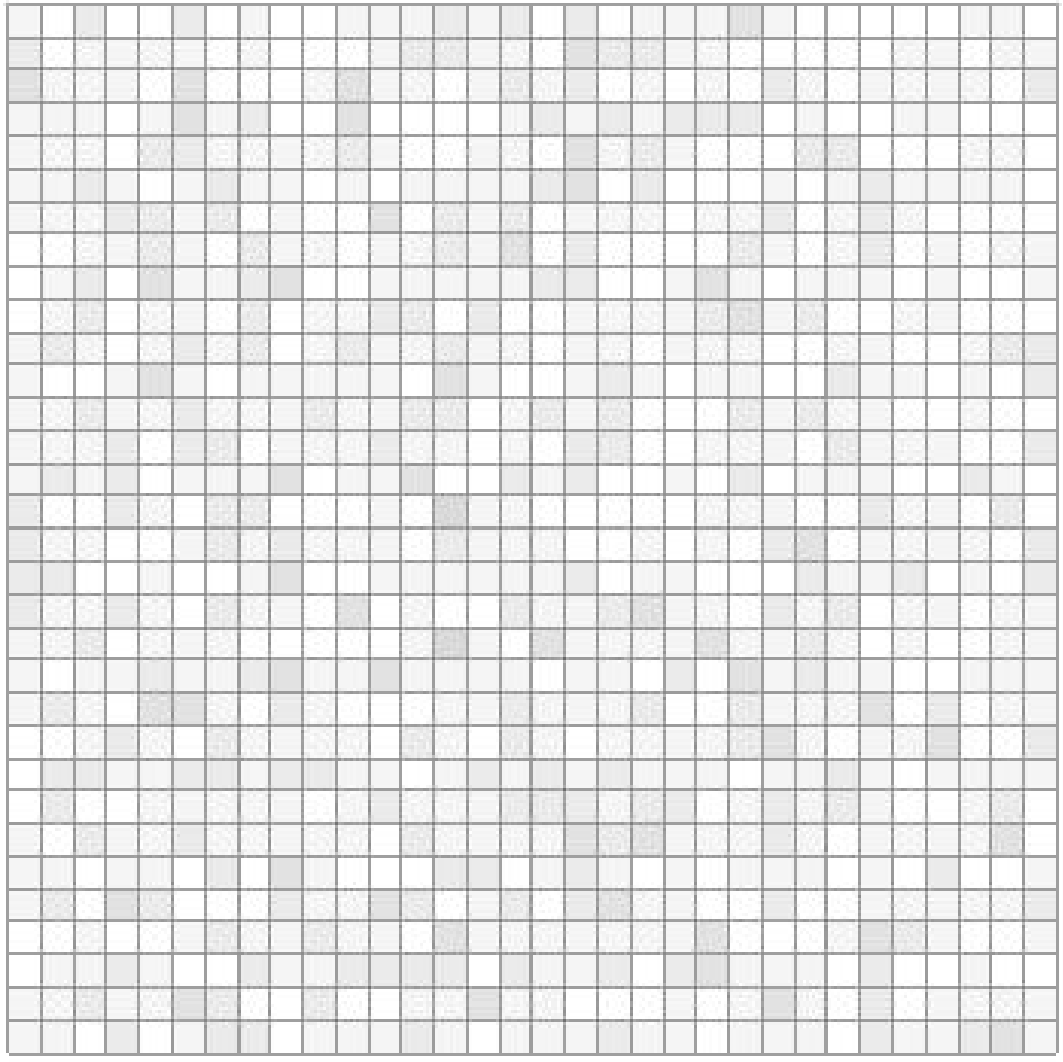
Blockwise Korkine-Zolotarev (BKZ) reduction [Sch87, SE94]

- Blockwise generalization of Korkine-Zolotarev reduction
- Do reductions/swaps on $(\boldsymbol{b}_i, \ldots, \boldsymbol{b}_{i+k-1})$ for $i = 1, \ldots, n-k+1$
- Blocksize k offers time-quality tradeoff

LLL algorithm



BKZ algorithm



BKZ algorithm

Lenstra-Lenstra-Lovasz (LLL) algorithm [LLL82]

- Blockwise generalization of Gauss reduction
- Do reductions/swaps on $(\boldsymbol{b}_i, \boldsymbol{b}_{i+1})$ for $i = 1, \ldots, n-1$
- Basis quality deteriorates with the dimension n
  - Theoretically: $\|\boldsymbol{b}_1\| \leq 1.075^n \cdot \det(\mathscr{L})$
  - Experimentally: $\|\boldsymbol{b}_1\| \approx 1.022^n \cdot \det(\mathscr{L})$

Blockwise Korkine-Zolotarev (BKZ) reduction [Sch87, SE94]

- Blockwise generalization of Korkine-Zolotarev reduction
- Do reductions/swaps on $(\boldsymbol{b}_i, \ldots, \boldsymbol{b}_{i+k-1})$ for $i = 1, \ldots, n-k+1$
- Blocksize k offers time-quality tradeoff

BKZ algorithm

Lenstra-Lenstra-Lovasz (LLL) algorithm [LLL82]

- Blockwise generalization of Gauss reduction
- Do reductions/swaps on $(\boldsymbol{b}_i, \boldsymbol{b}_{i+1})$ for $i = 1, \ldots, n-1$
- Basis quality deteriorates with the dimension n
  - Theoretically: $\|\boldsymbol{b}_1\| \leq 1.075^n \cdot \det(\mathscr{L})$
  - Experimentally: $\|\boldsymbol{b}_1\| \approx 1.022^n \cdot \det(\mathscr{L})$

Blockwise Korkine-Zolotarev (BKZ) reduction [Sch87, SE94]

- Blockwise generalization of Korkine-Zolotarev reduction
- Do reductions/swaps on $(\boldsymbol{b}_i, \ldots, \boldsymbol{b}_{i+k-1})$ for $i = 1, \ldots, n-k+1$
- Blocksize k offers time-quality tradeoff

BKZ uses **exact SVP** algorithm in dimension k as subroutine

IBM

BKZ algorithm

Lenstra-Lenstra-Lovasz (LLL) algorithm [LLL82]

- Blockwise generalization of Gauss reduction
- Do reductions/swaps on $(\boldsymbol{b}_i, \boldsymbol{b}_{i+1})$ for $i = 1, \ldots, n-1$
- Basis quality deteriorates with the dimension n
  - Theoretically: $\|\boldsymbol{b}_1\| \leq 1.075^n \cdot \det(\mathscr{L})$
  - Experimentally: $\|\boldsymbol{b}_1\| \approx 1.022^n \cdot \det(\mathscr{L})$

Blockwise Korkine-Zolotarev (BKZ) reduction [Sch87, SE94]

- Blockwise generalization of Korkine-Zolotarev reduction
- Do reductions/swaps on $(\boldsymbol{b}_i, \ldots, \boldsymbol{b}_{i+k-1})$ for $i = 1, \ldots, n-k+1$
- Blocksize k offers time-quality tradeoff

BKZ uses **exact SVP** algorithm in dimension k as subroutine
Question: How to solve exact SVP in high dimensions?

IBM

Outline

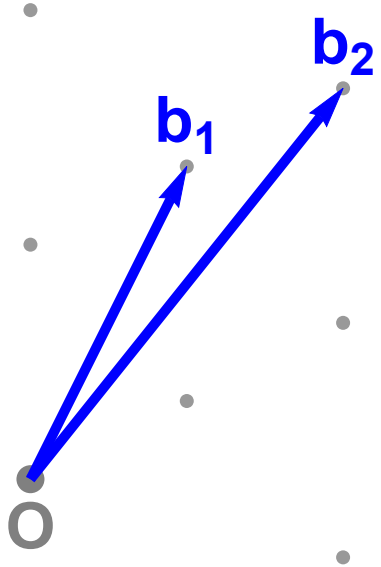
Motivation: GGH encryption

Lattice basis reduction
- Gauss reduction
- LLL reduction
- BKZ reduction

Exact SVP algorithms
- Enumeration algorithms
- Sieving algorithms
- Practical comparison

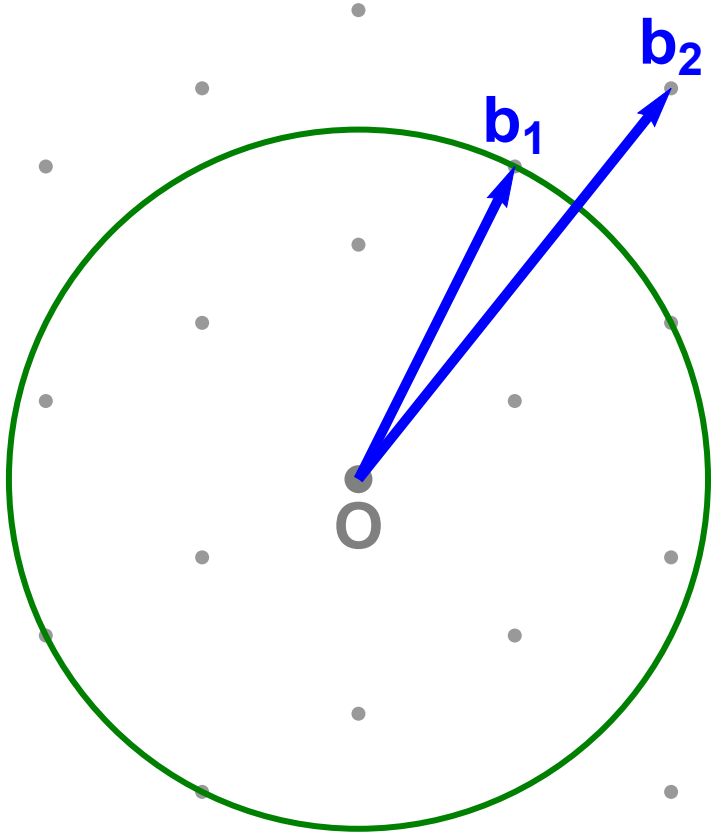
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



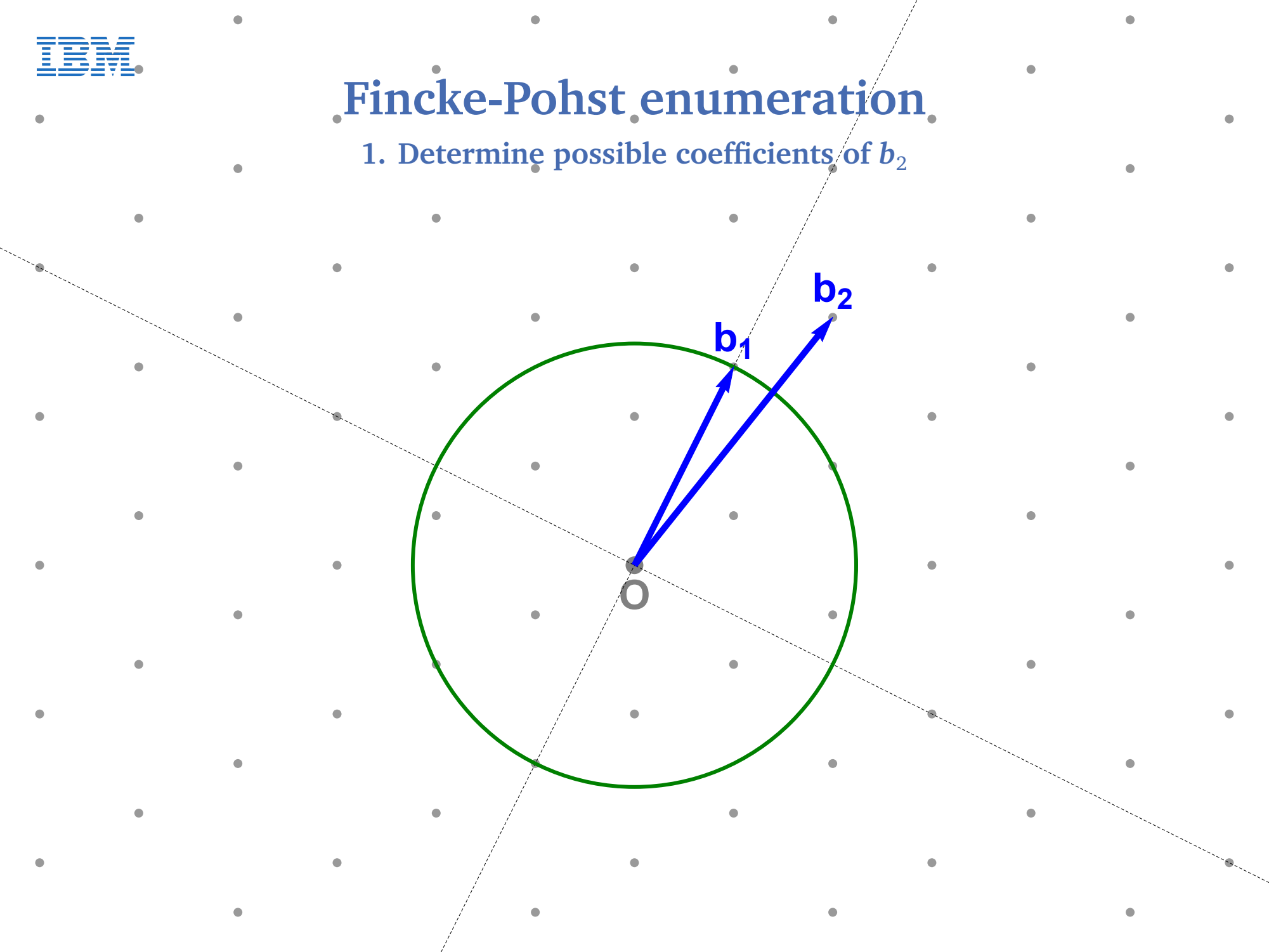
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



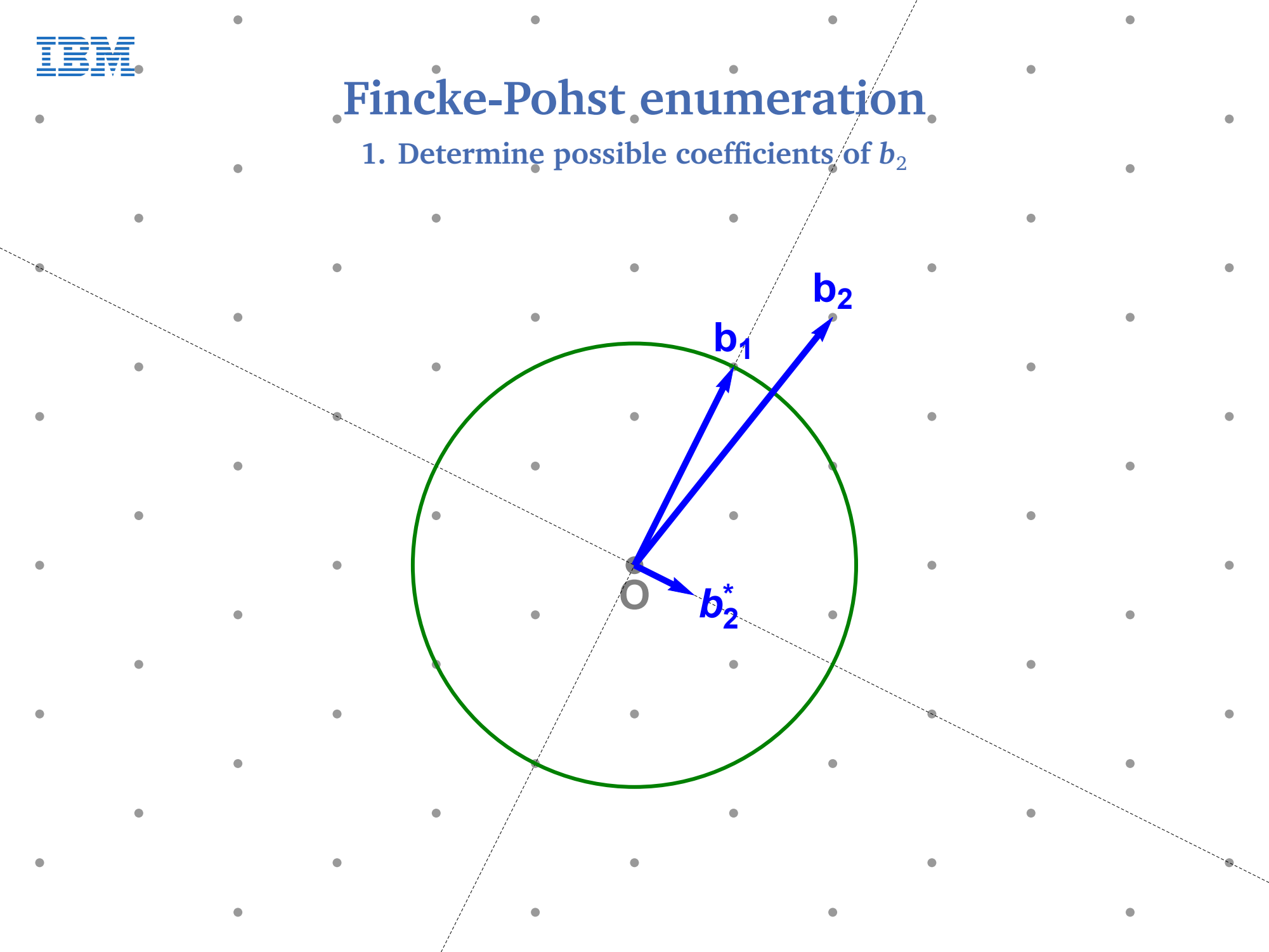
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



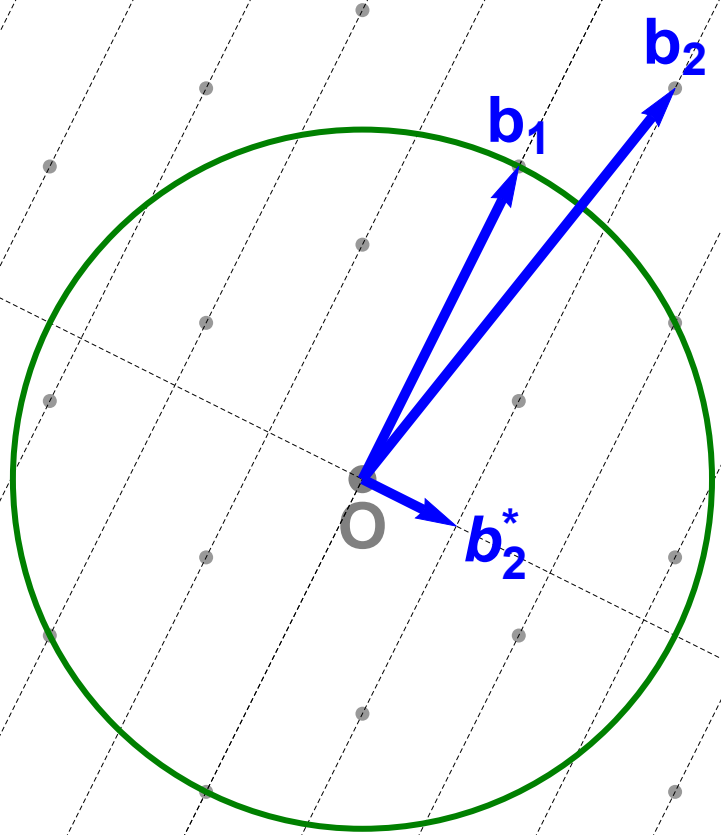
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



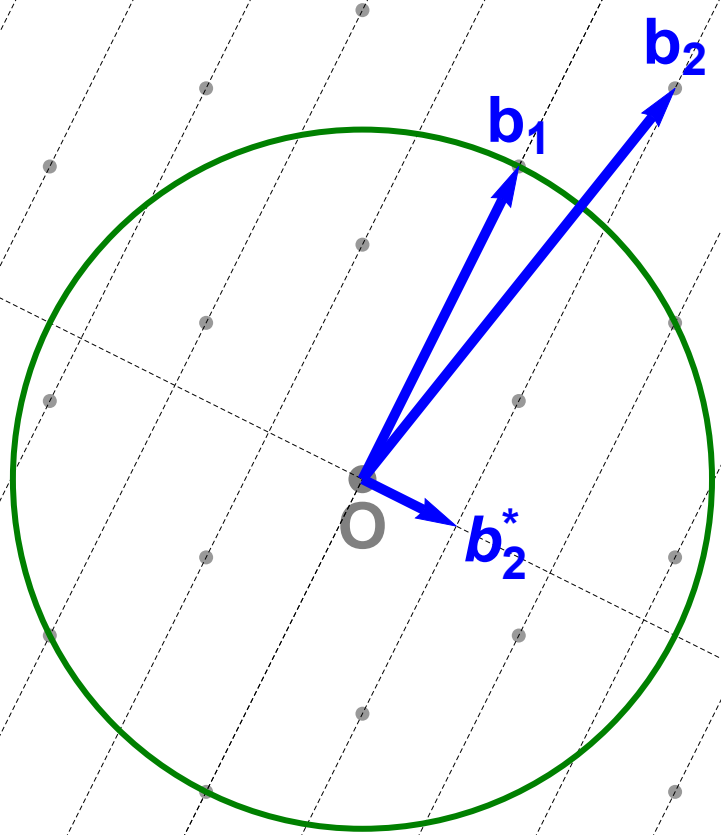
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



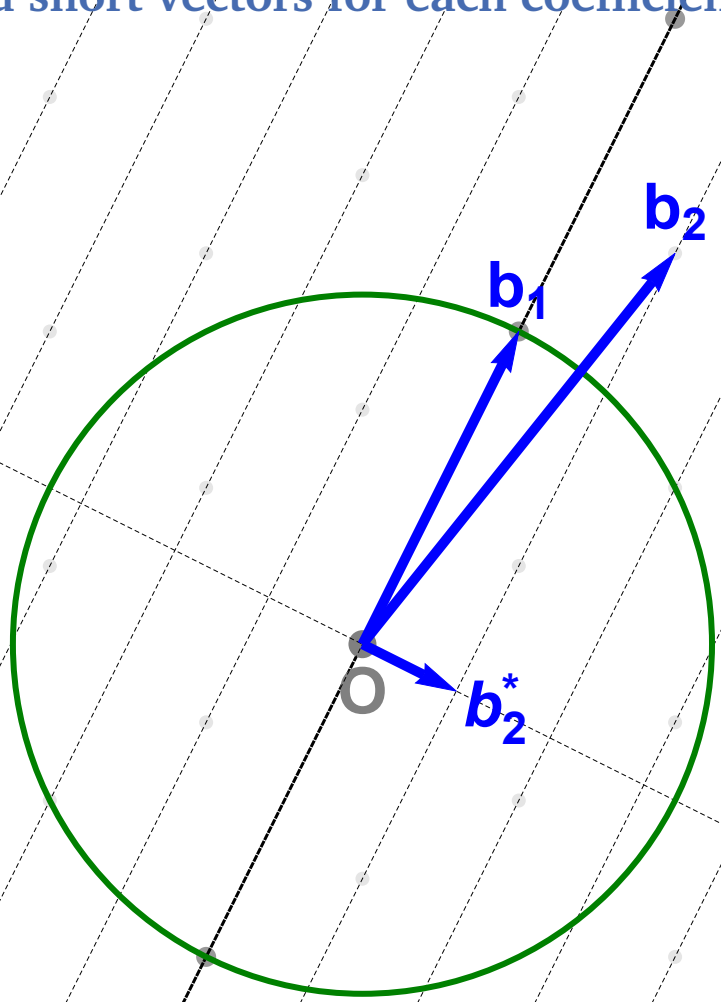
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



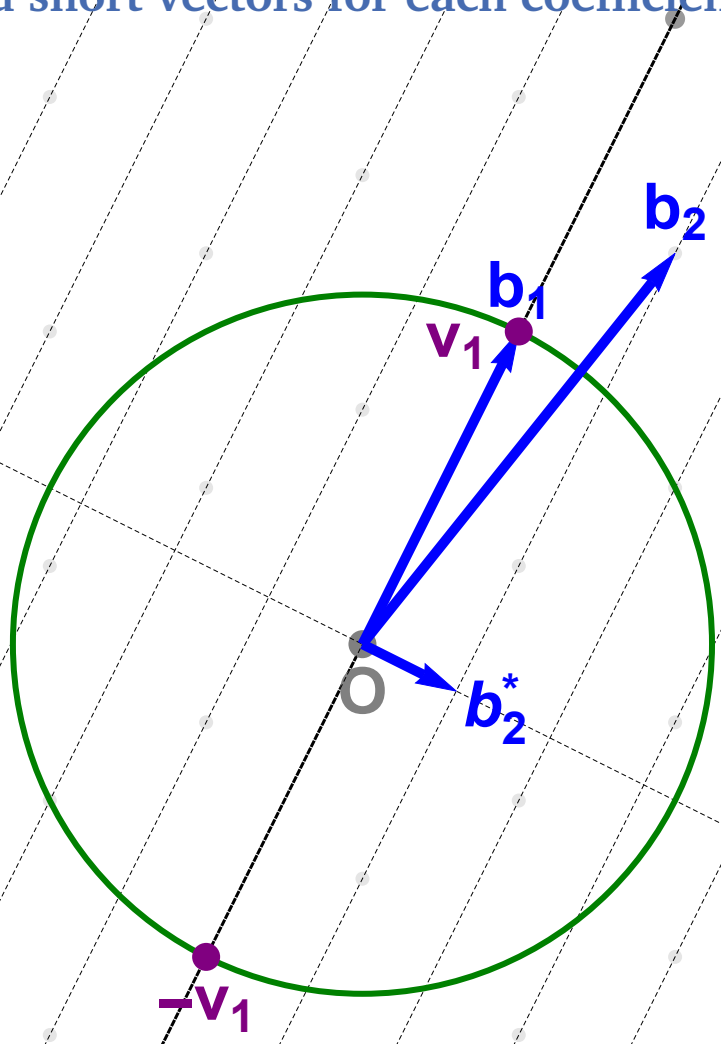
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



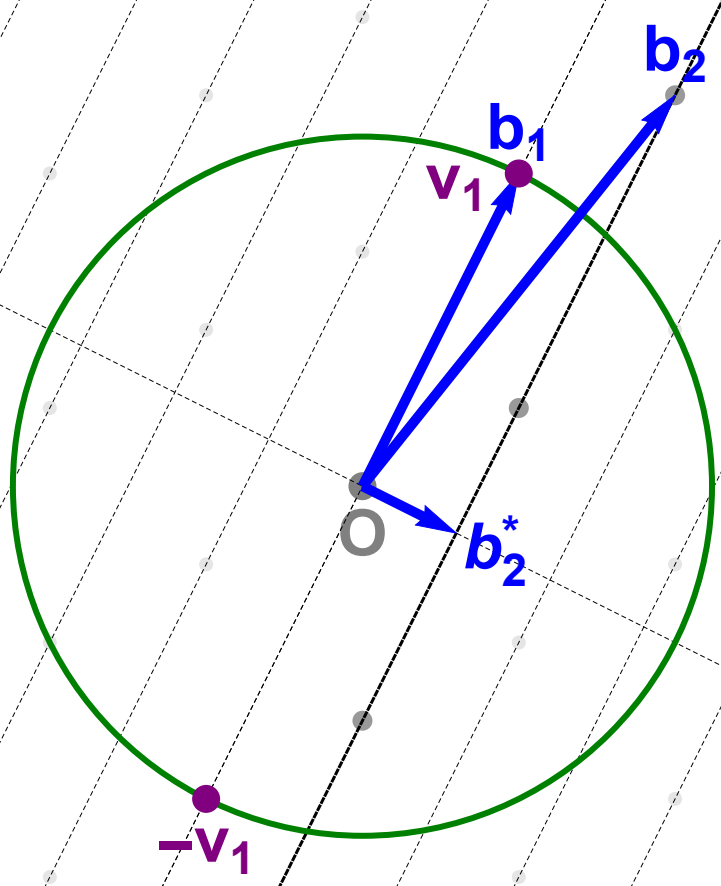
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



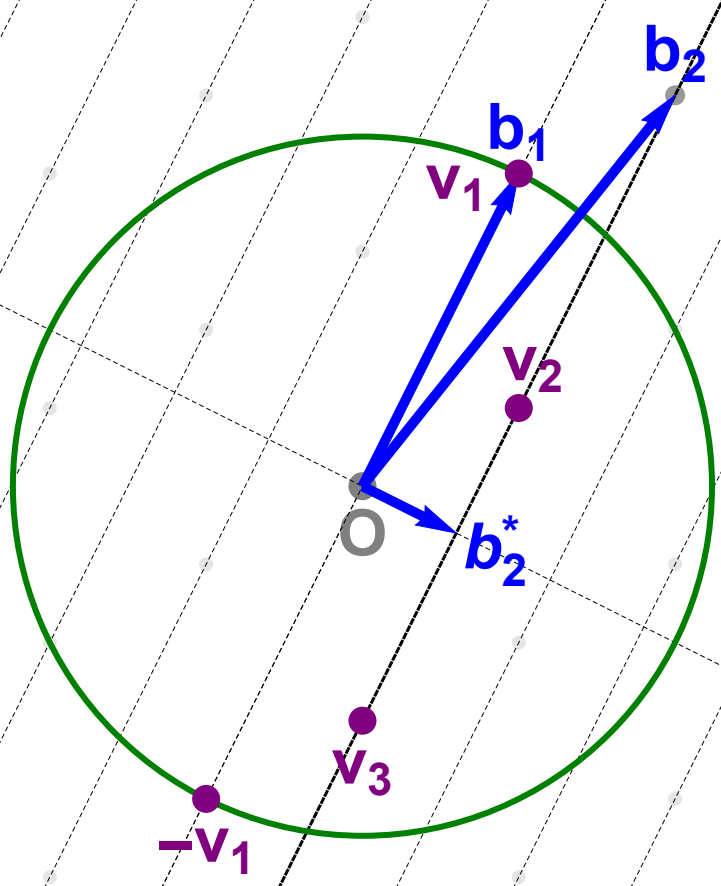
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



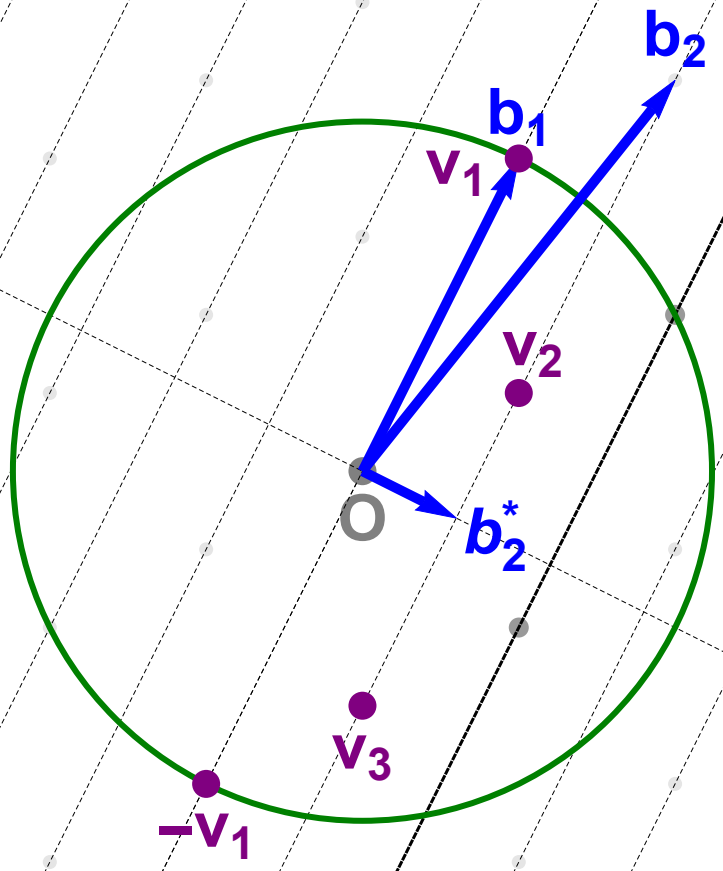
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



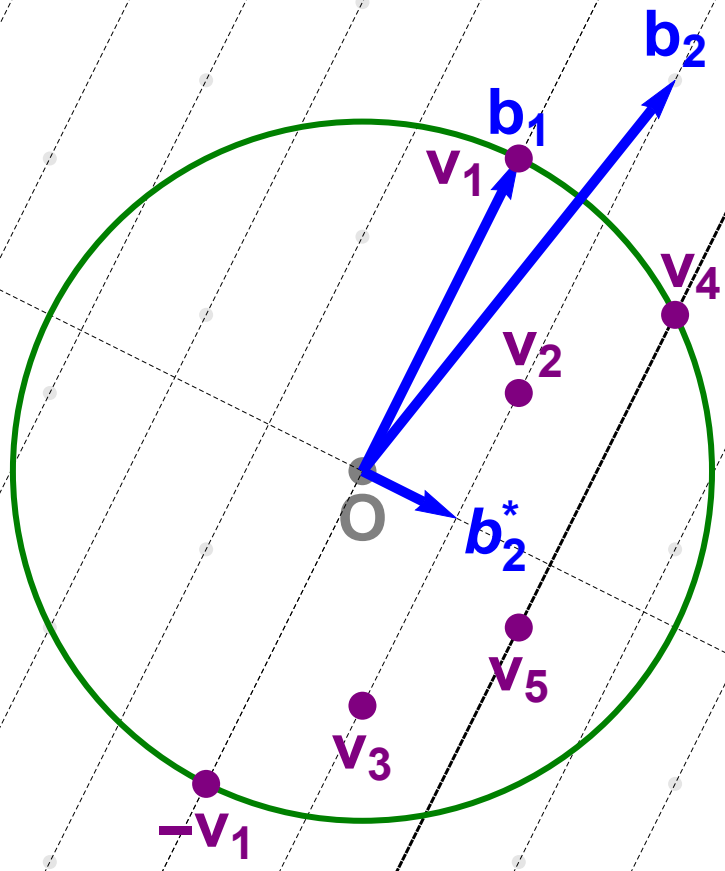
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



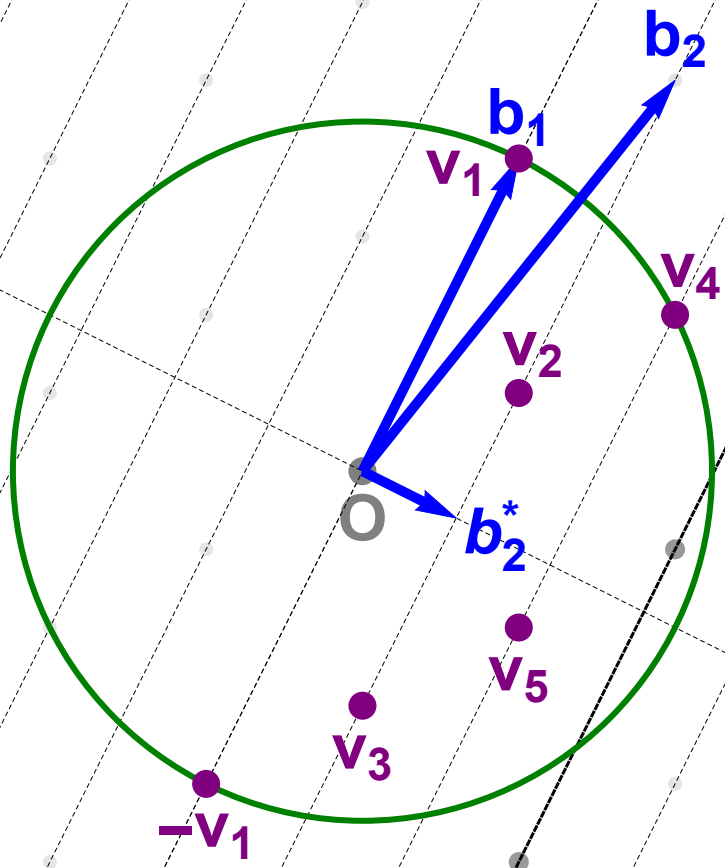
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



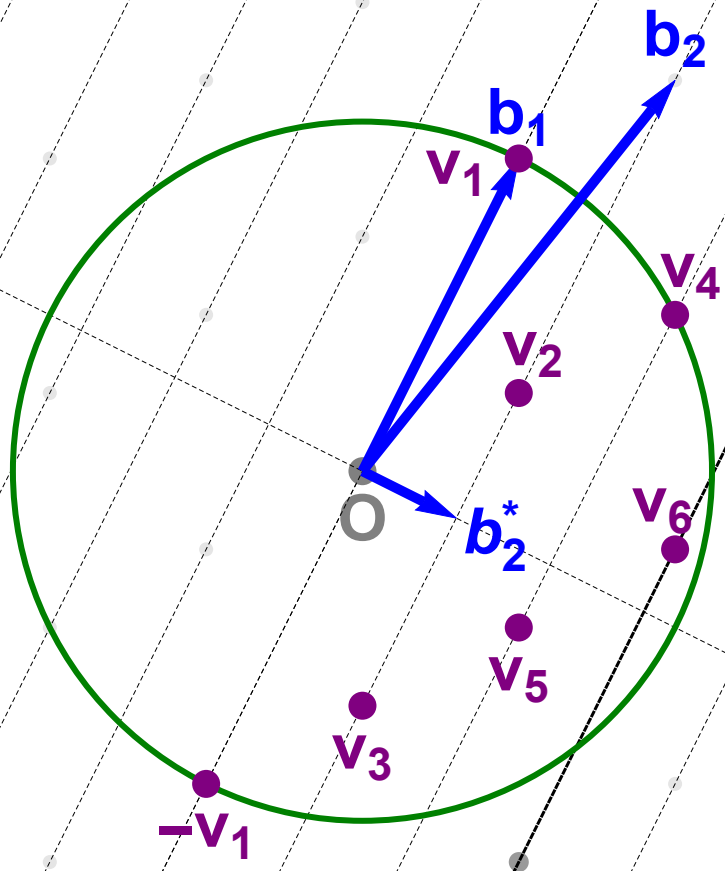
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



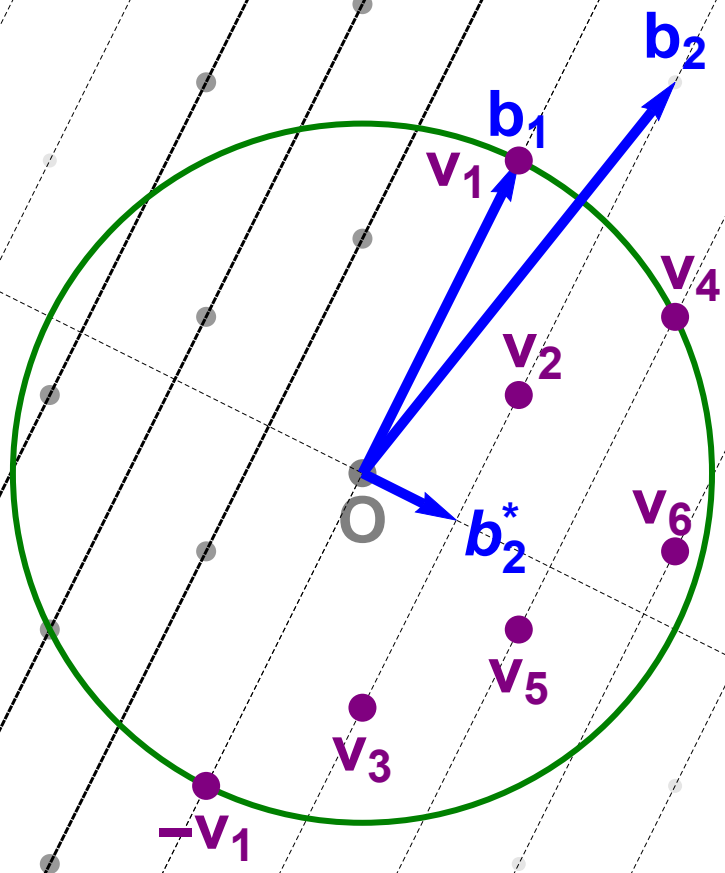
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



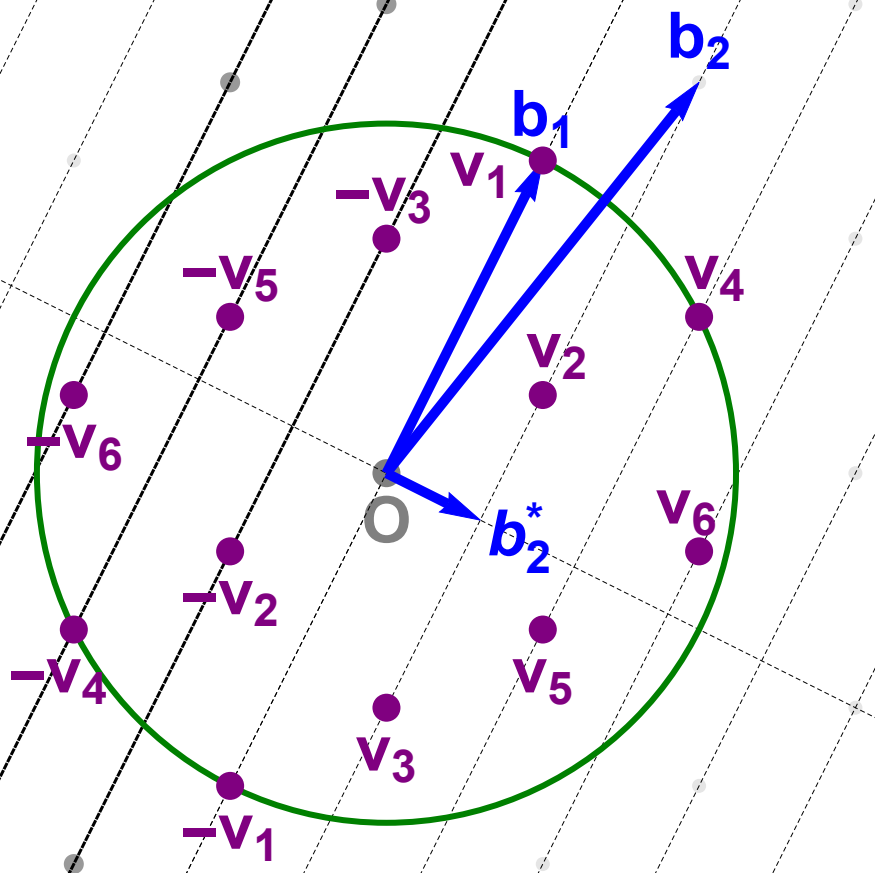
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



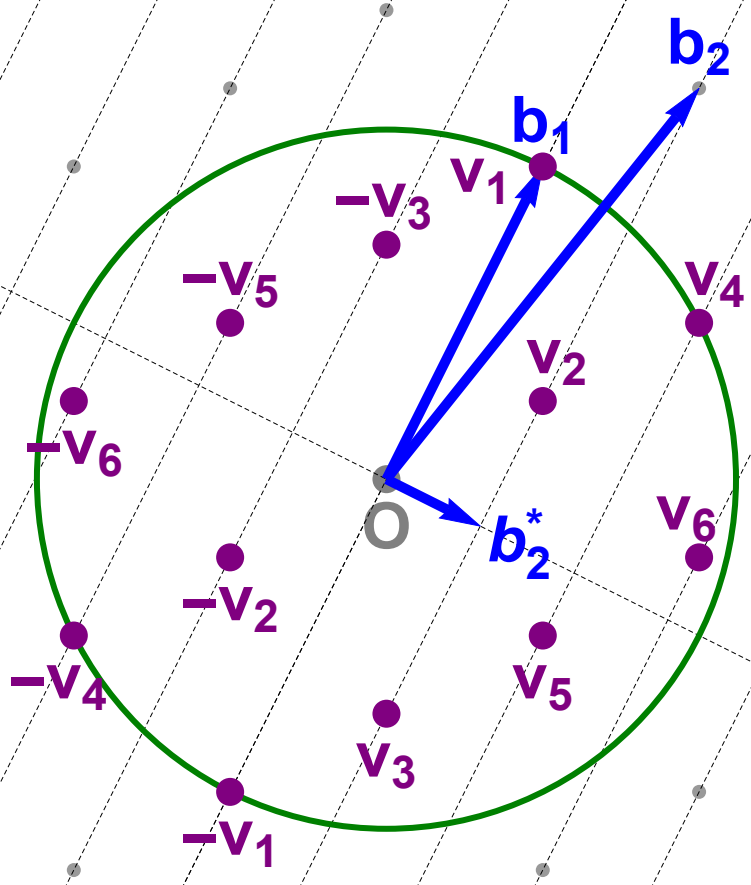
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



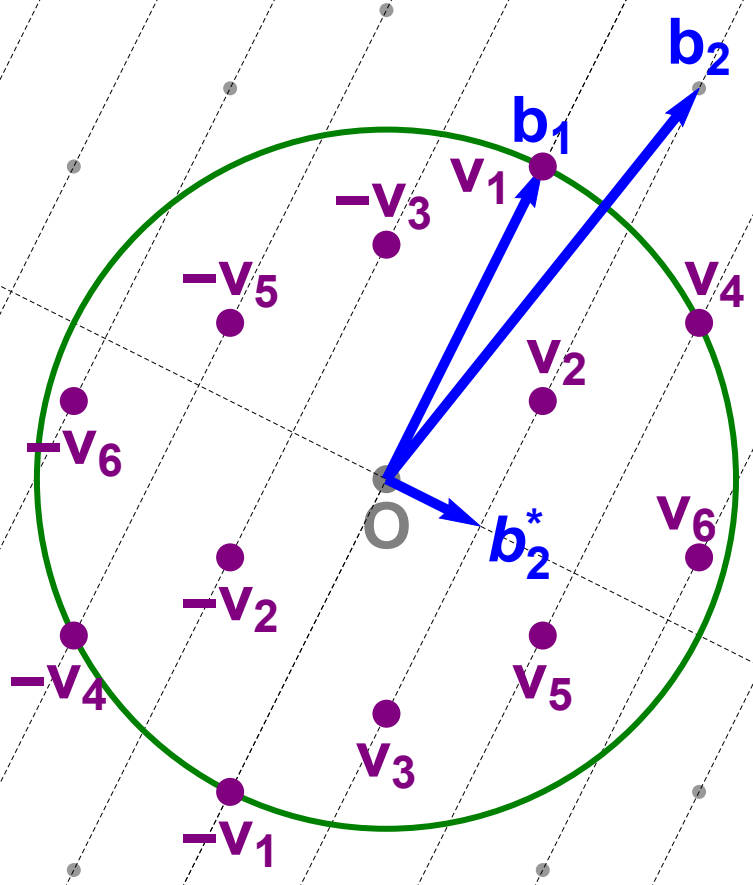
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



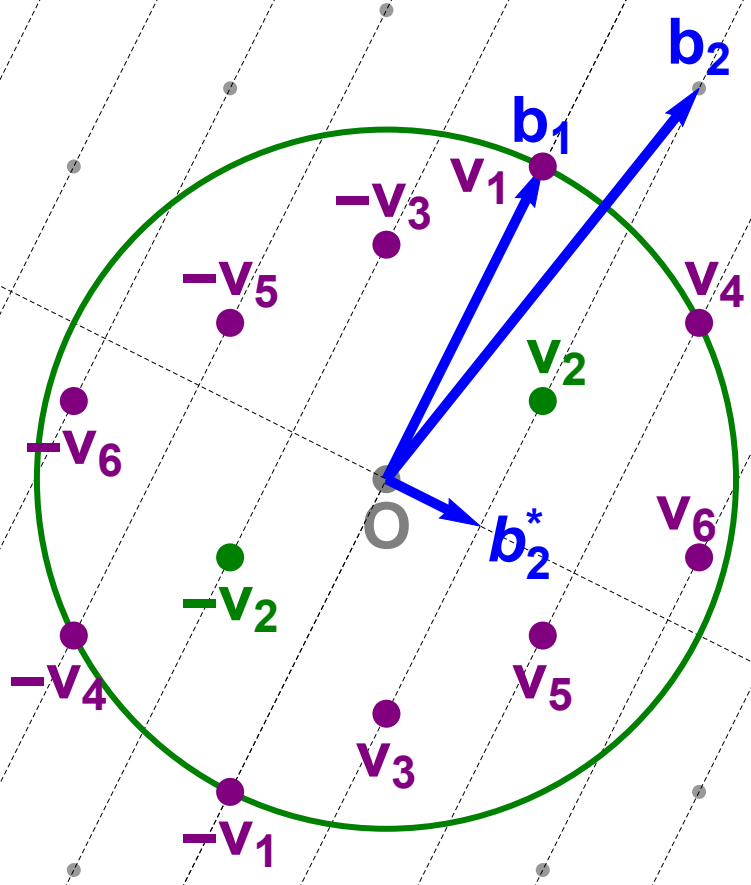
Fincke-Pohst enumeration

3. Find a shortest vector among all found vectors



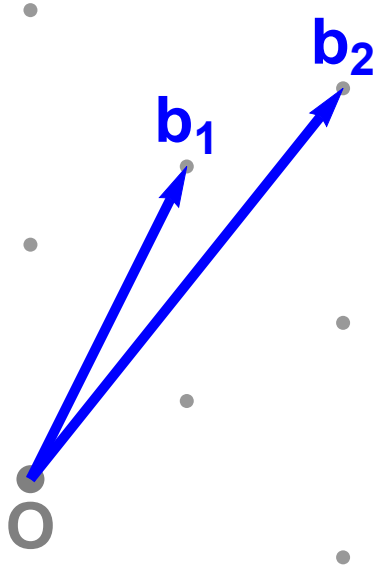
Fincke-Pohst enumeration

3. Find a shortest vector among all found vectors



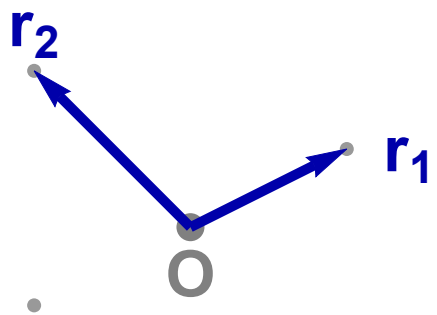
Kannan enumeration

Better bases



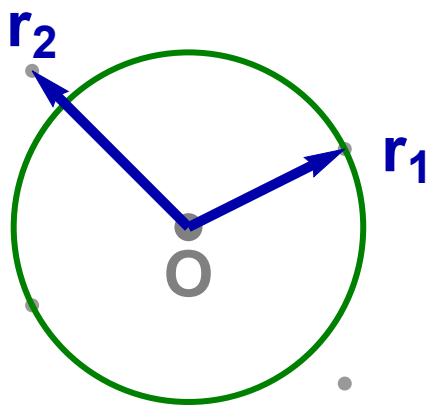
Kannan enumeration

Better bases



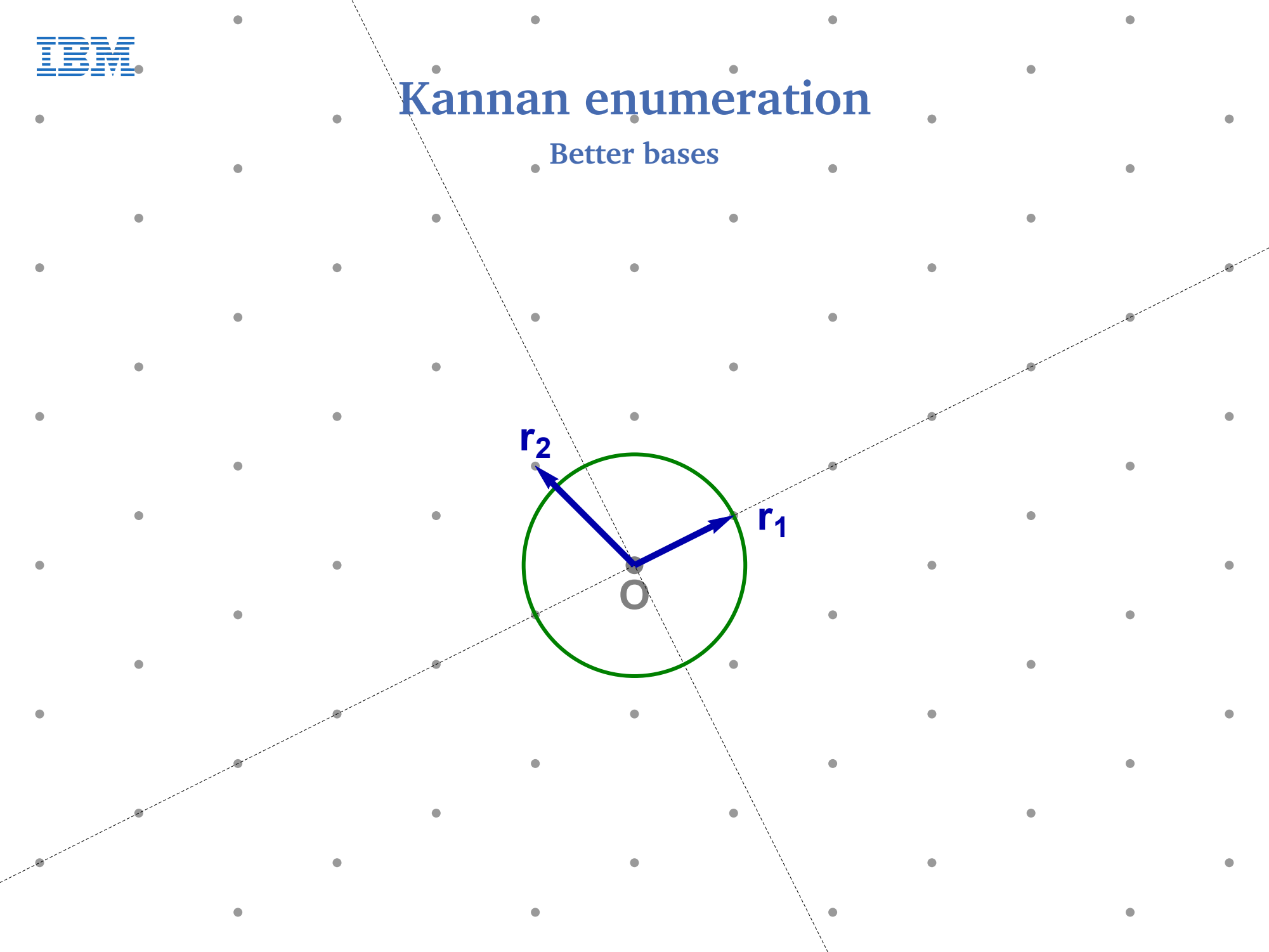
Kannan enumeration

Better bases



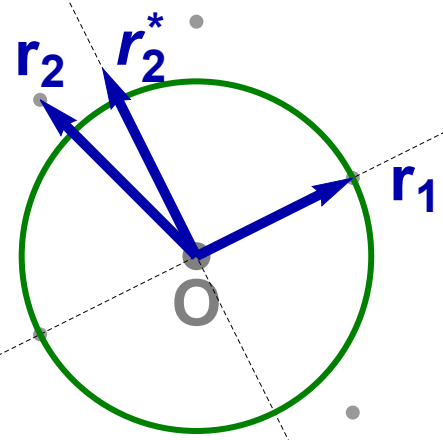
Kannan enumeration

Better bases



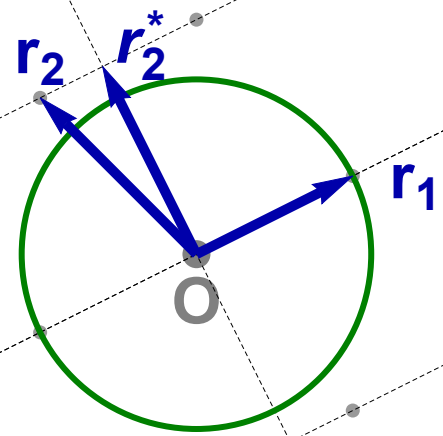
Kannan enumeration

Better bases



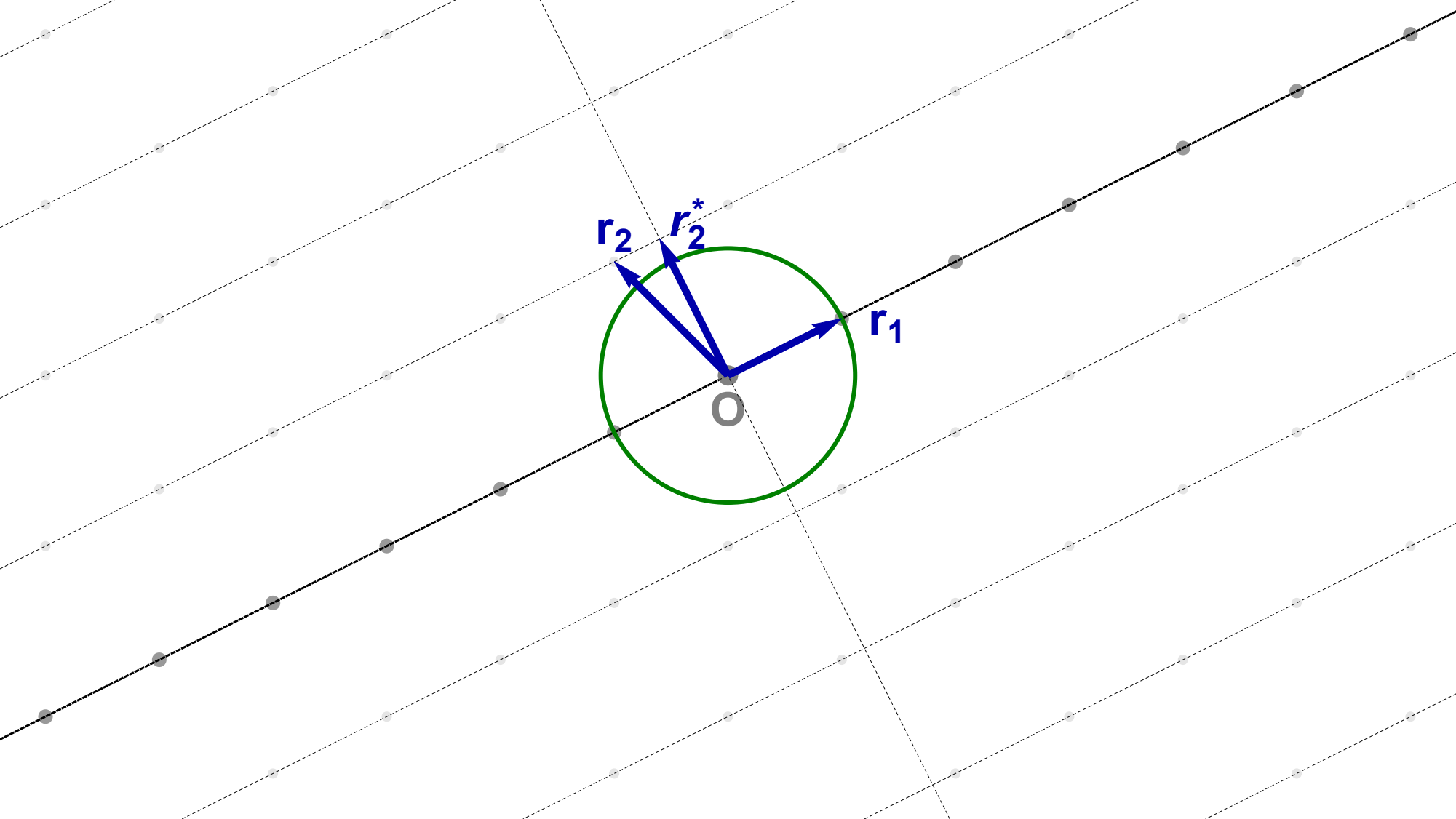
Kannan enumeration

Better bases



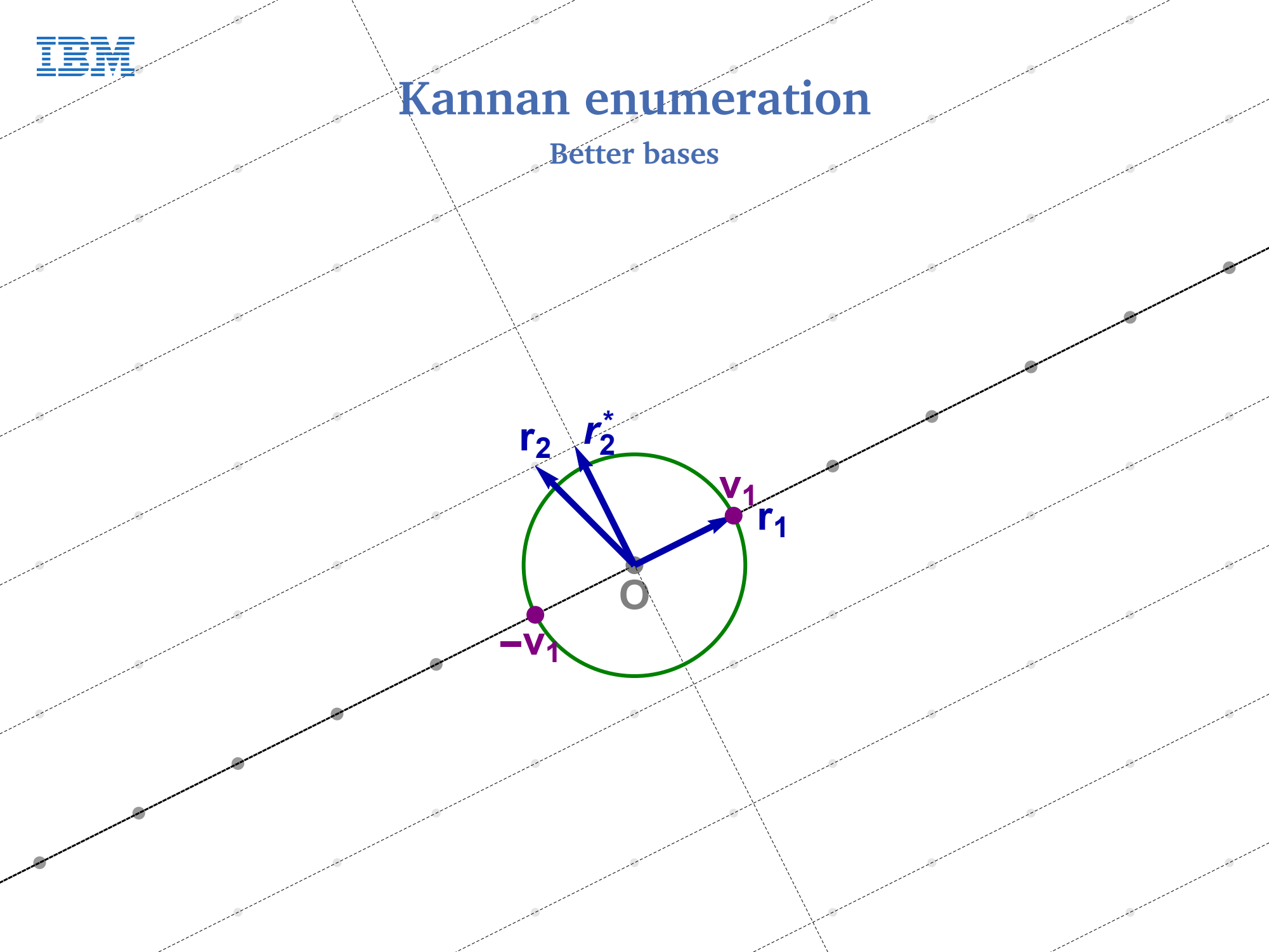
Kannan enumeration

Better bases



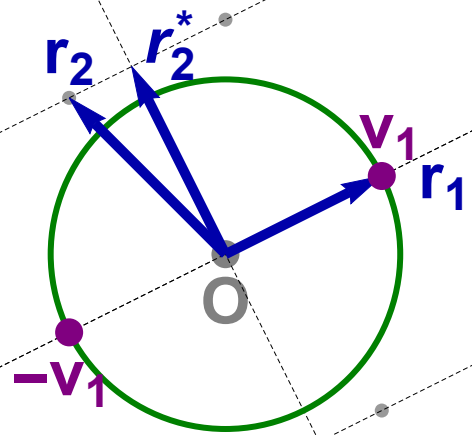
Kannan enumeration

Better bases



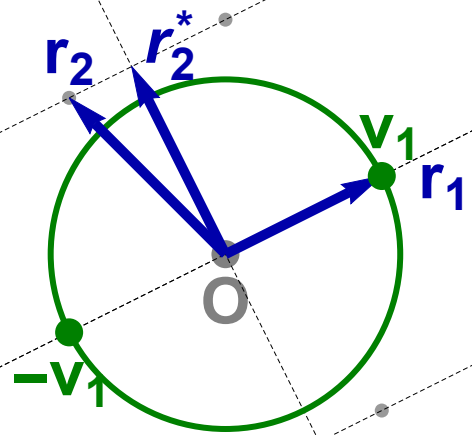
Kannan enumeration

Better bases



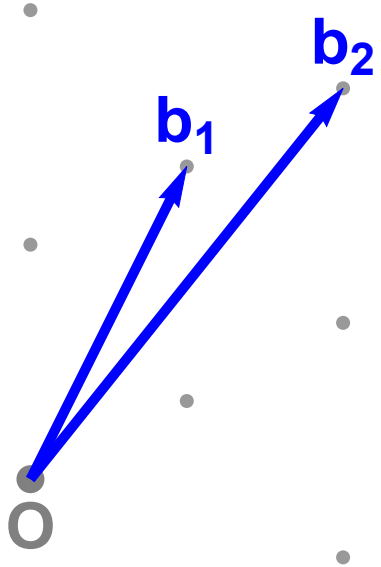
Kannan enumeration

Better bases



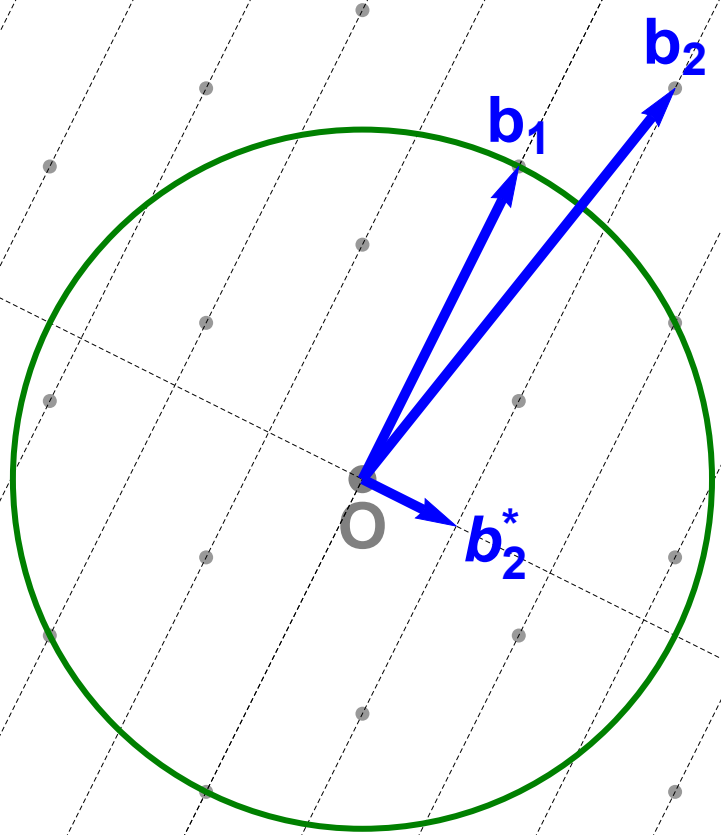
Pruned enumeration

Reducing the search space



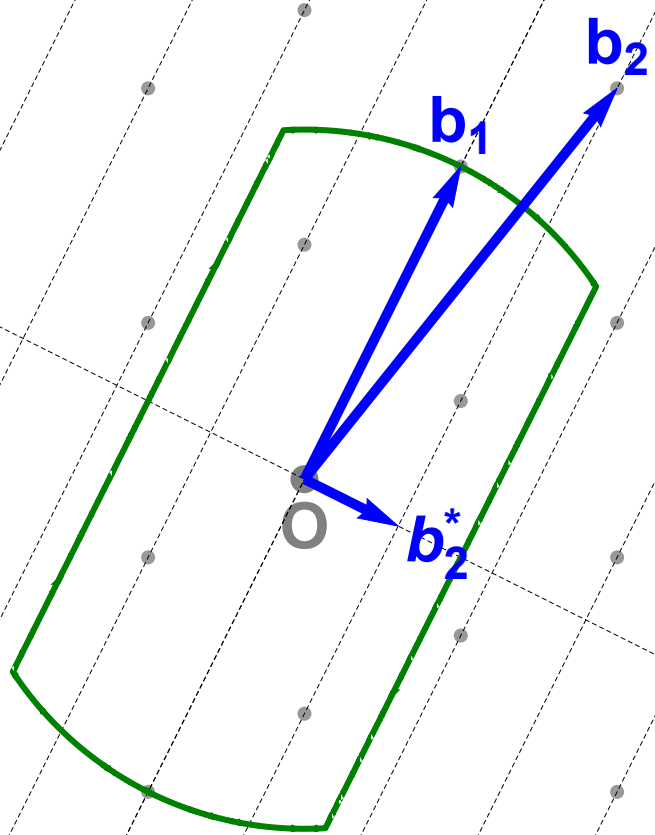
Pruned enumeration

Reducing the search space



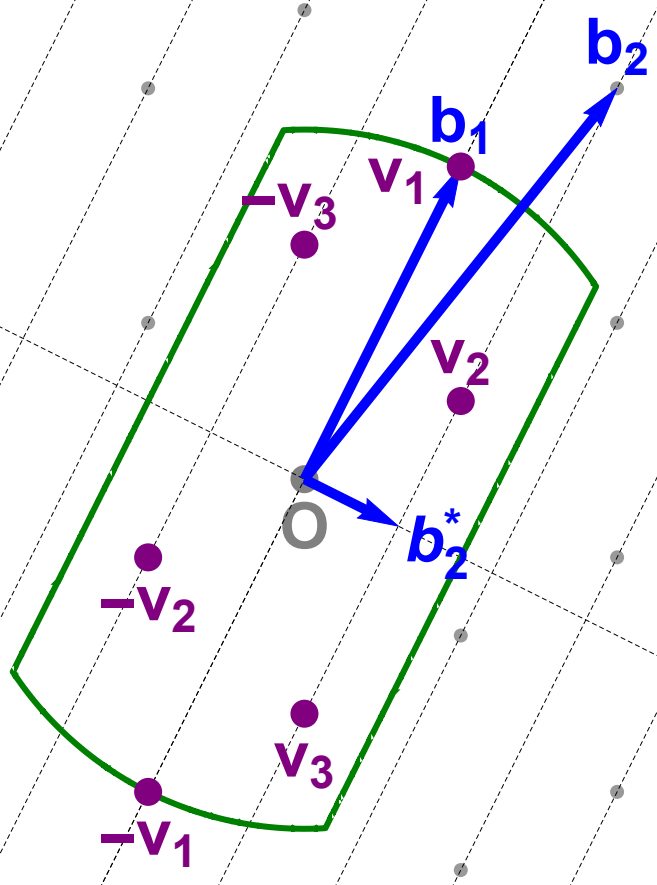
Pruned enumeration

Reducing the search space



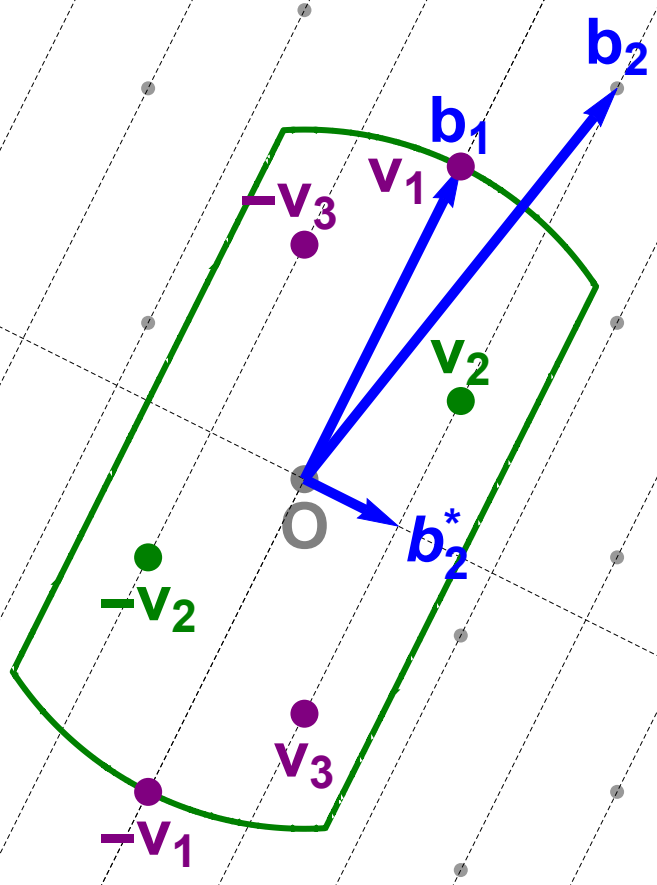
Pruned enumeration

Reducing the search space



Pruned enumeration

Reducing the search space



Exact SVP algorithms

Lattice enumeration

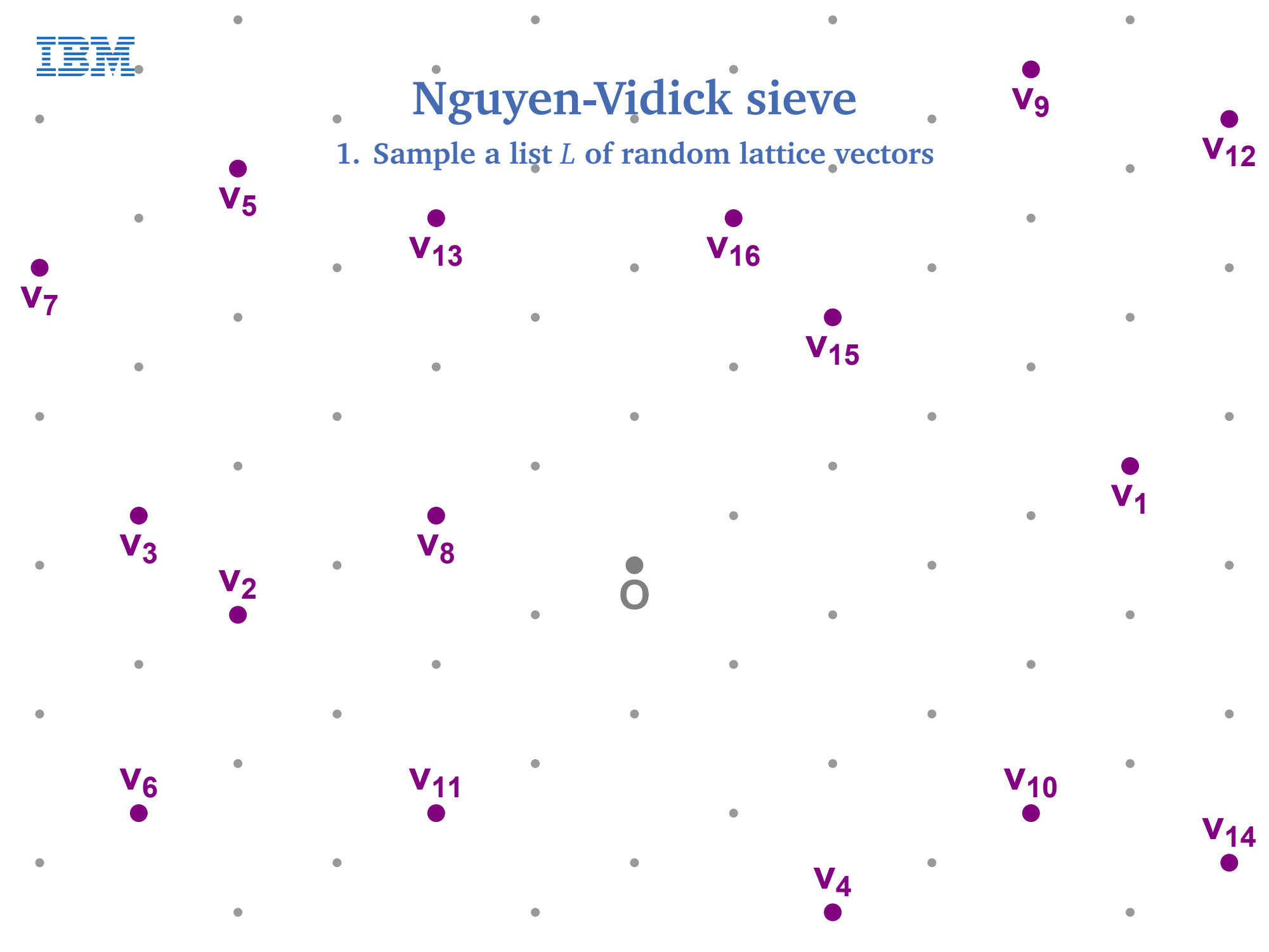
- Memory efficient
- Fincke-Pohst enumeration: $2^{O(n^2)}$ time
- Kannan enumeration: $2^{O(n \log n)}$ time
- Practical speedups with pruning heuristics

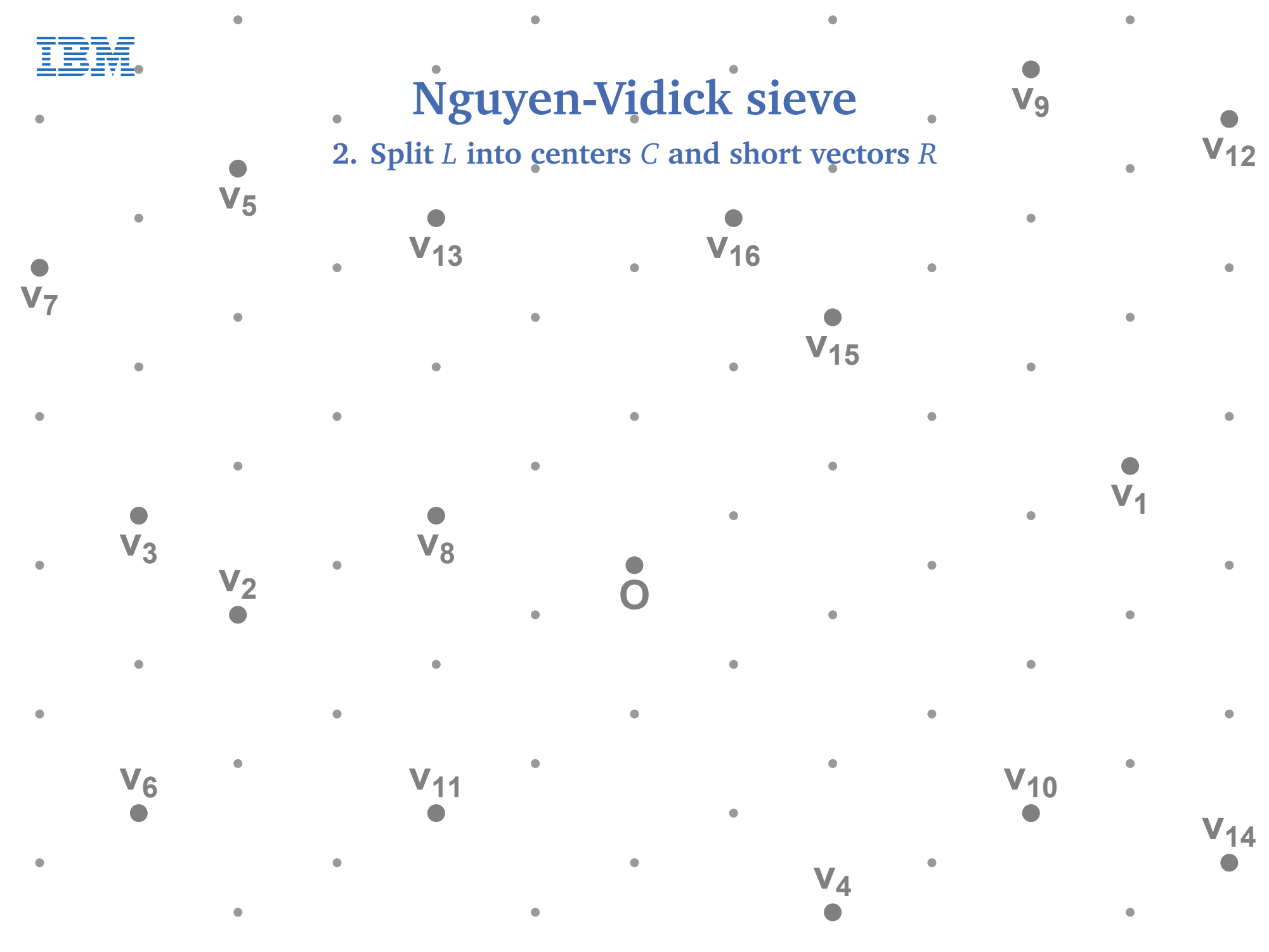
Nguyen-Vidick sieve

1. Sample a list L of random lattice vectors

Nguyen-Vidick sieve

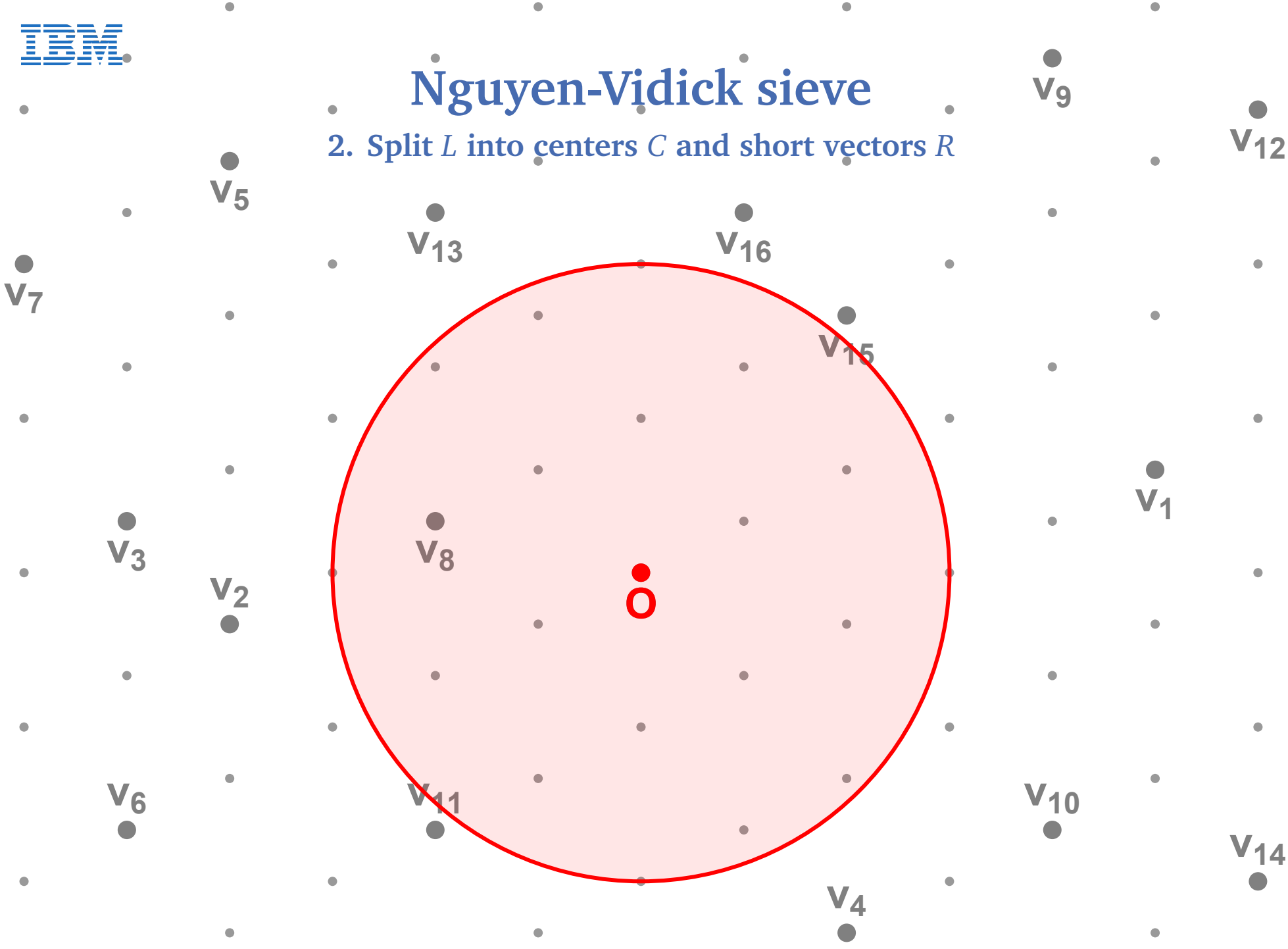
1. Sample a list L of random lattice vectors




IBM
Nguyen-Vidick sieve
2. Split L into centers C and short vectors R
v_9
v_{12}
v_5
v_{13}
v_{16}
v_7
v_{15}
v_1
v_3
v_8
v_2
O
v_6
v_{11}
v_{10}
v_{14}
v_4

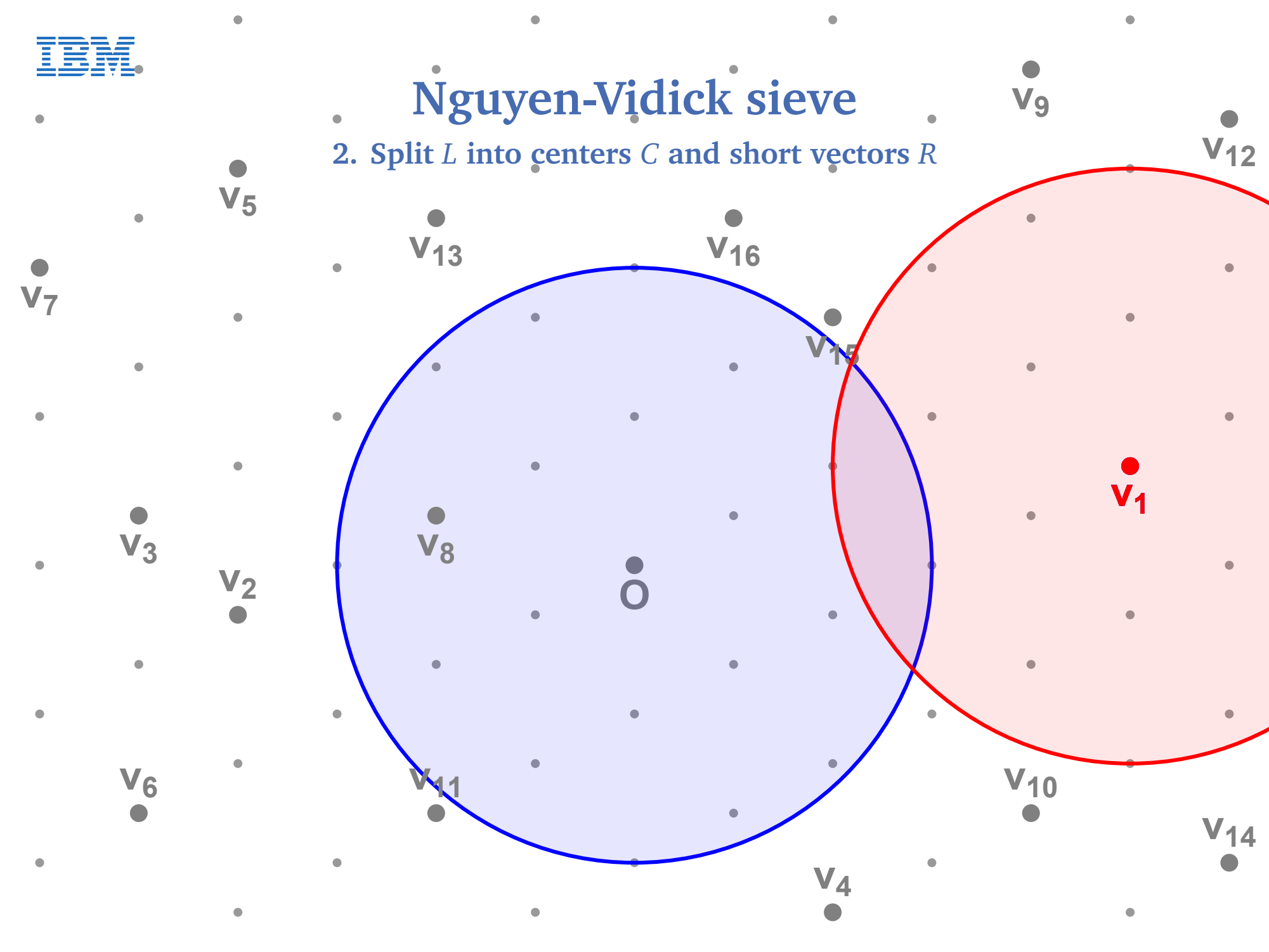
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



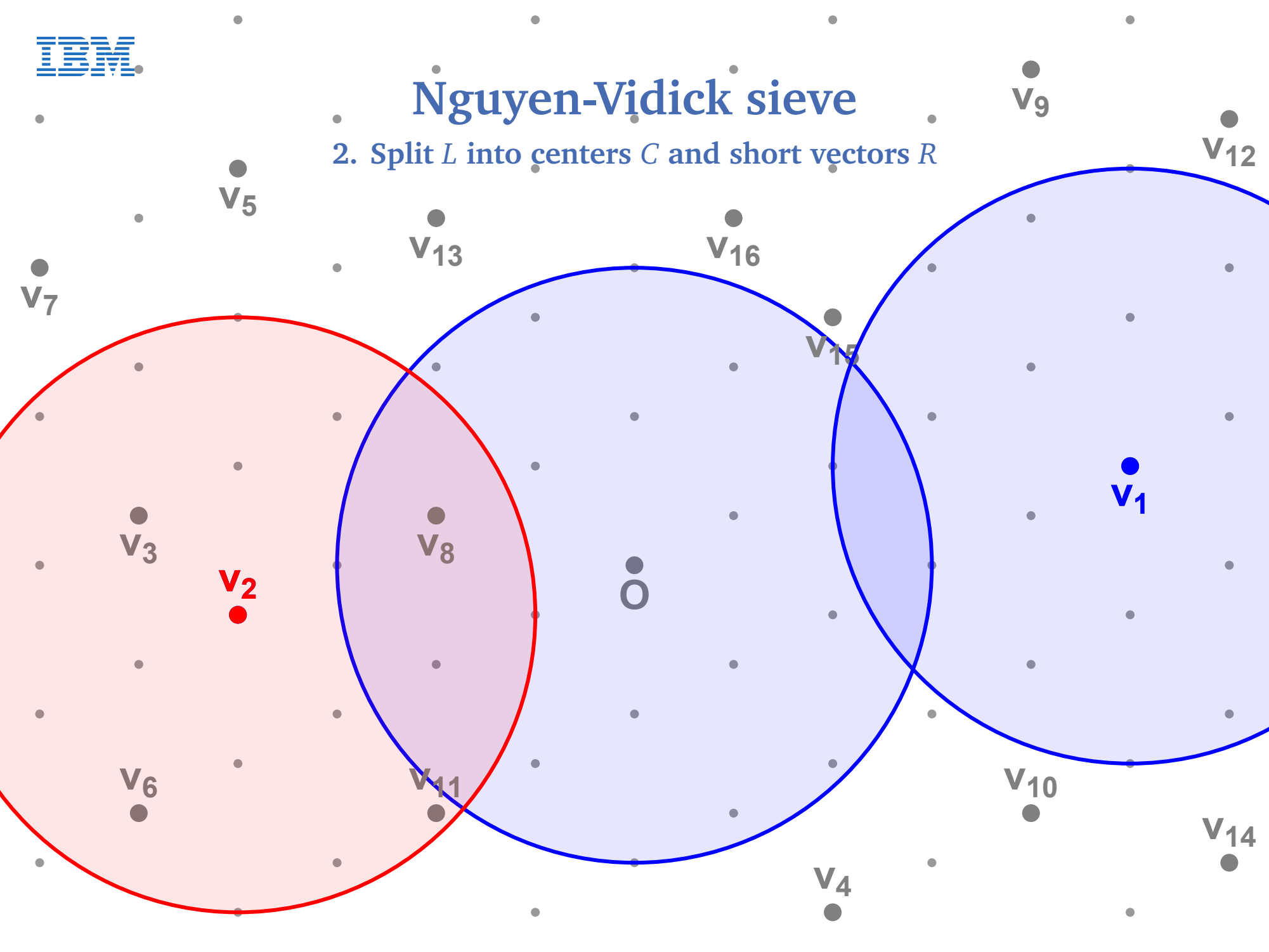
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



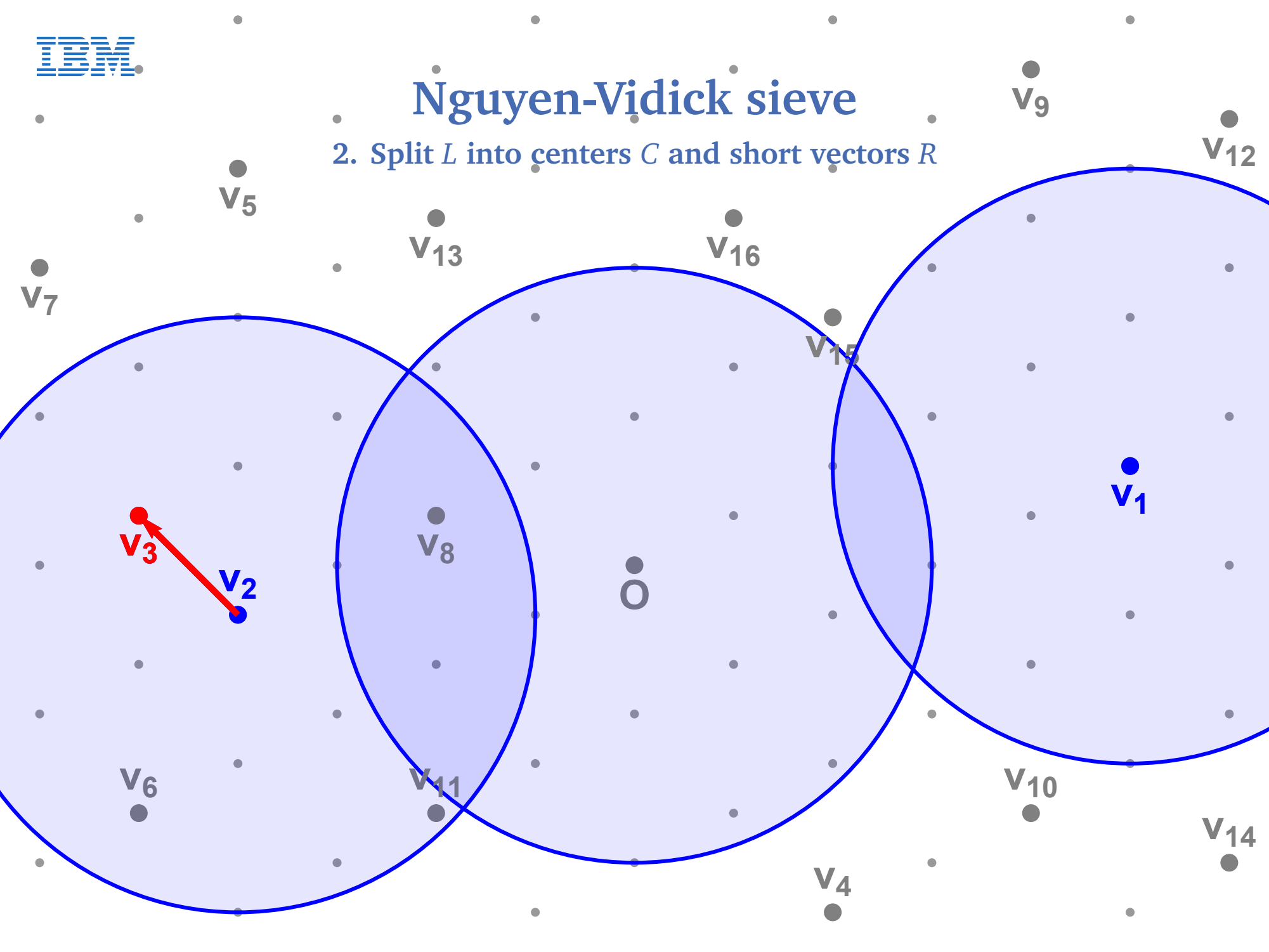
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



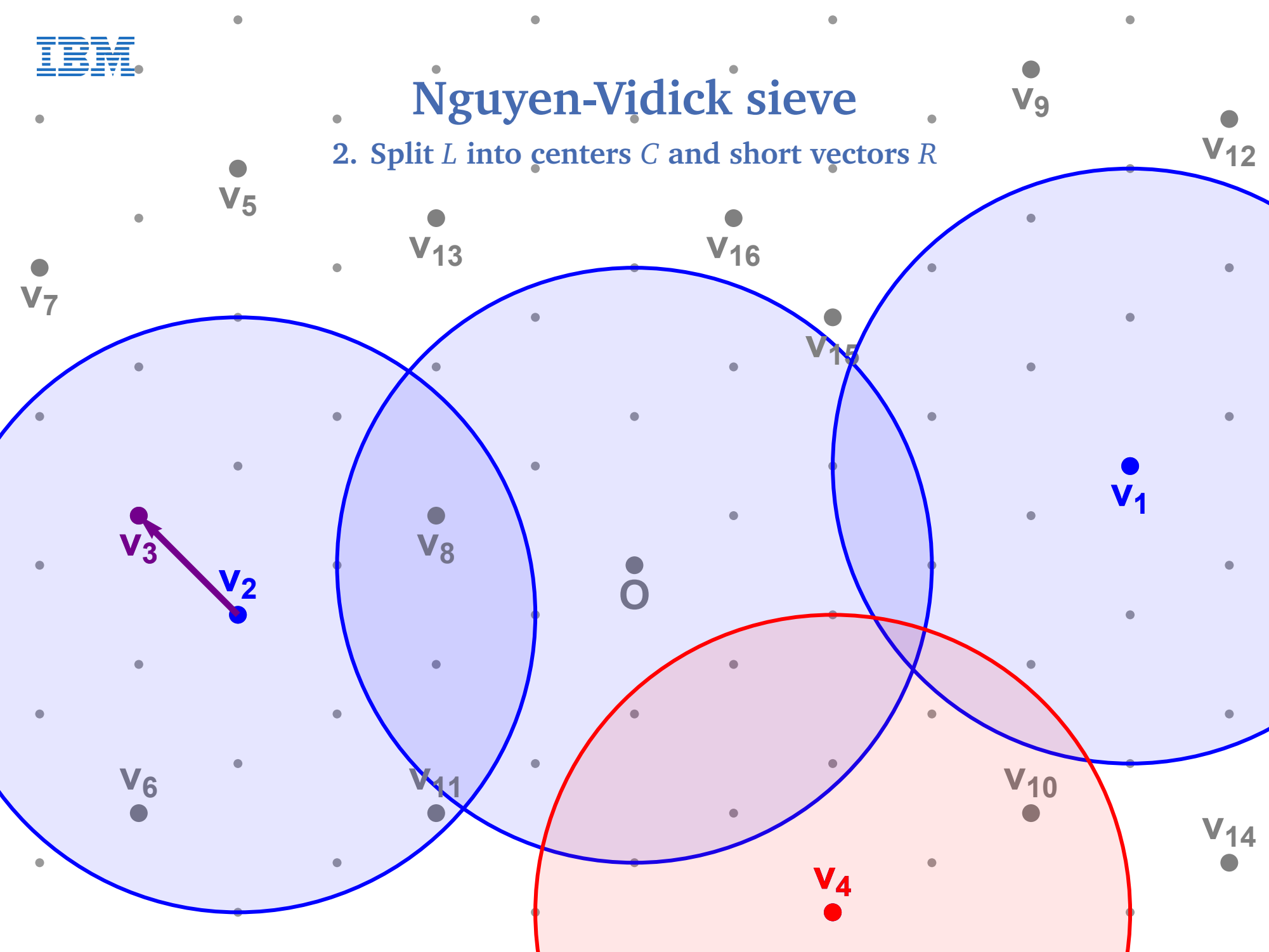
Nguyen-Vidick sieve

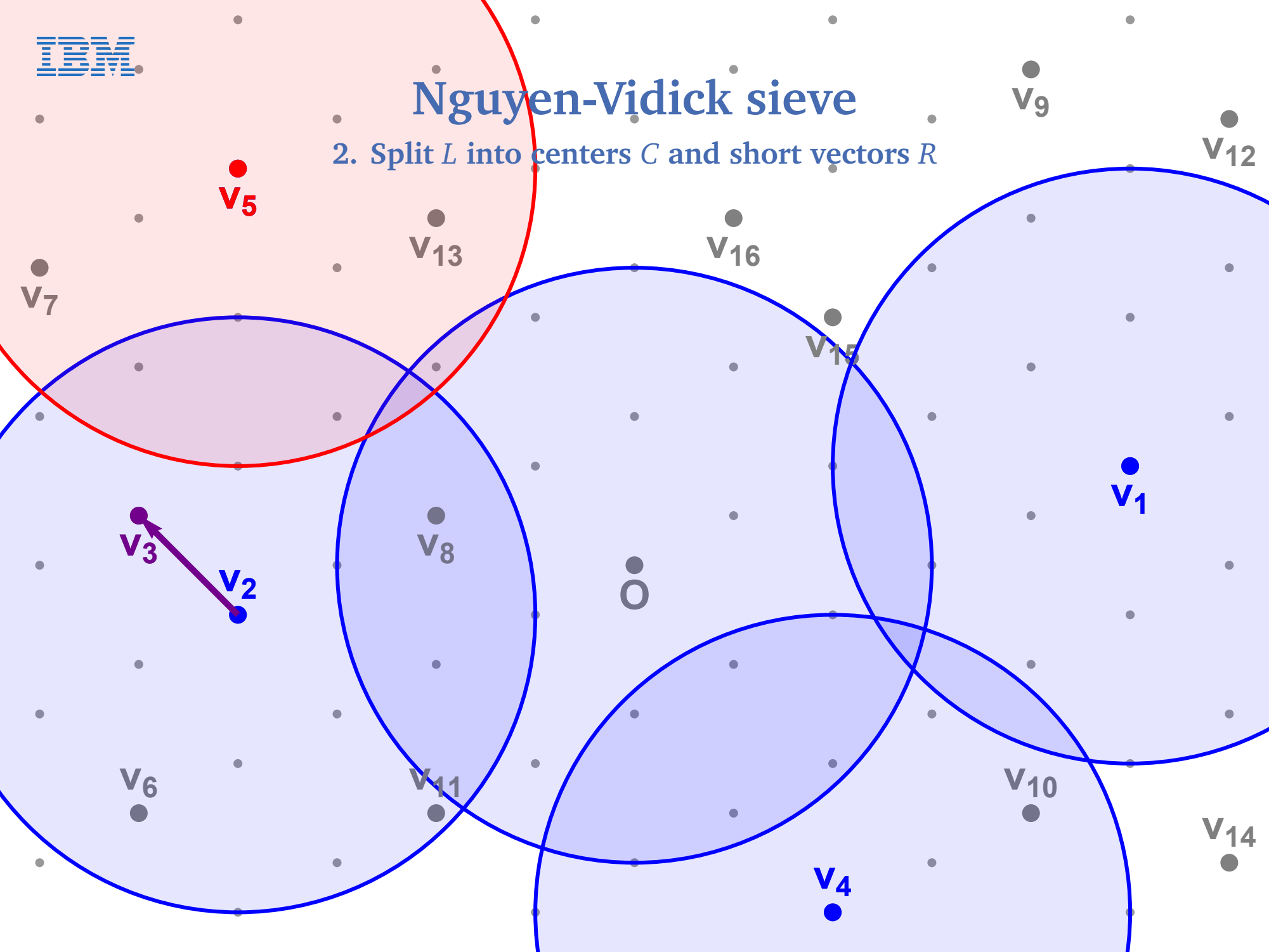
2. Split L into centers C and short vectors R



Nguyen-Vidick sieve

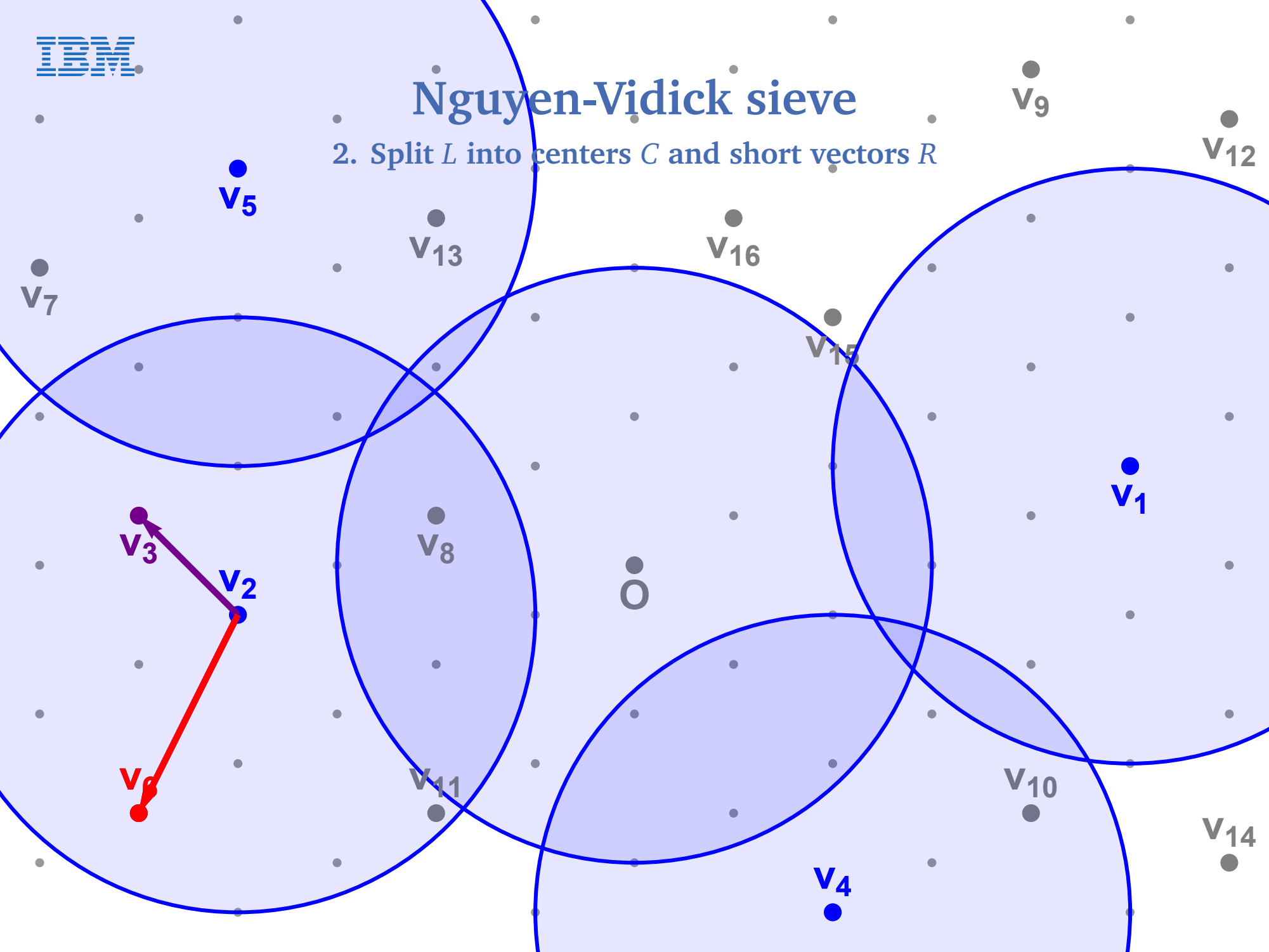
2. Split L into centers C and short vectors R




IBM
Nguyen-Vidick sieve
2. Split L into centers C and short vectors R
v_5
v_{13}
v_7
v_{16}
v_9
v_{12}
v_{15}
v_1
v_3
v_2
v_8
O
v_6
v_{11}
v_{10}
v_{14}
v_4

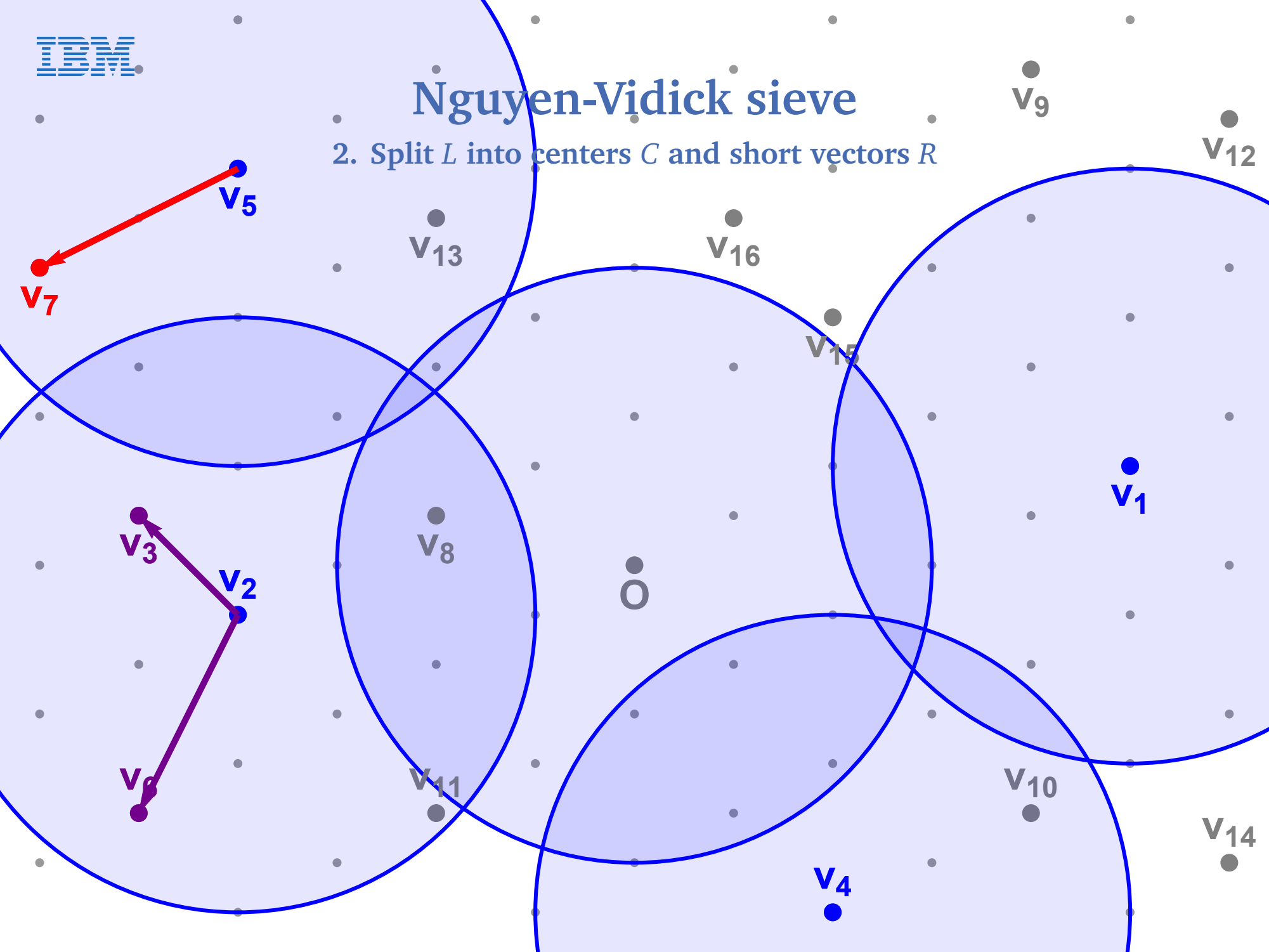
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



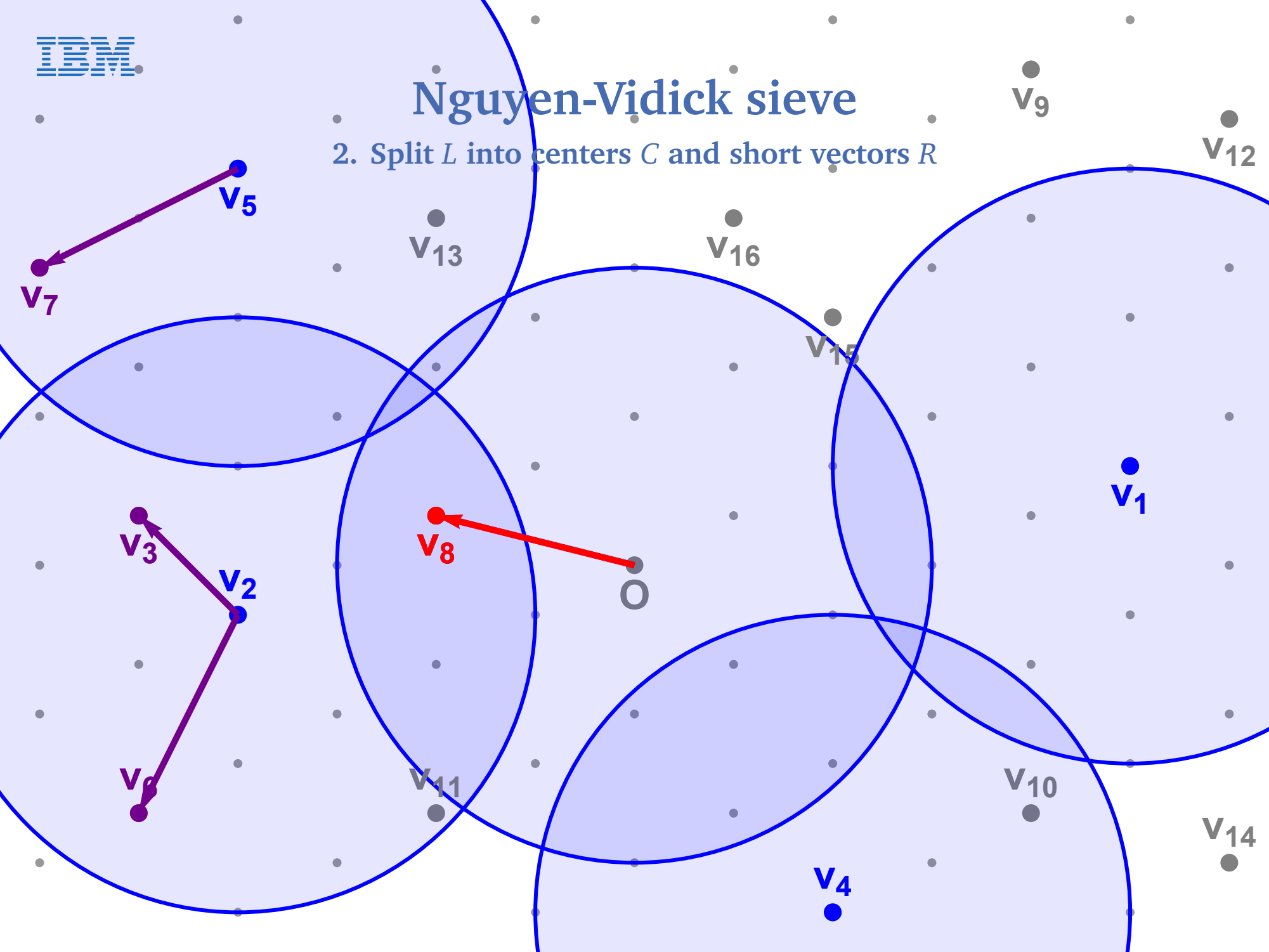
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



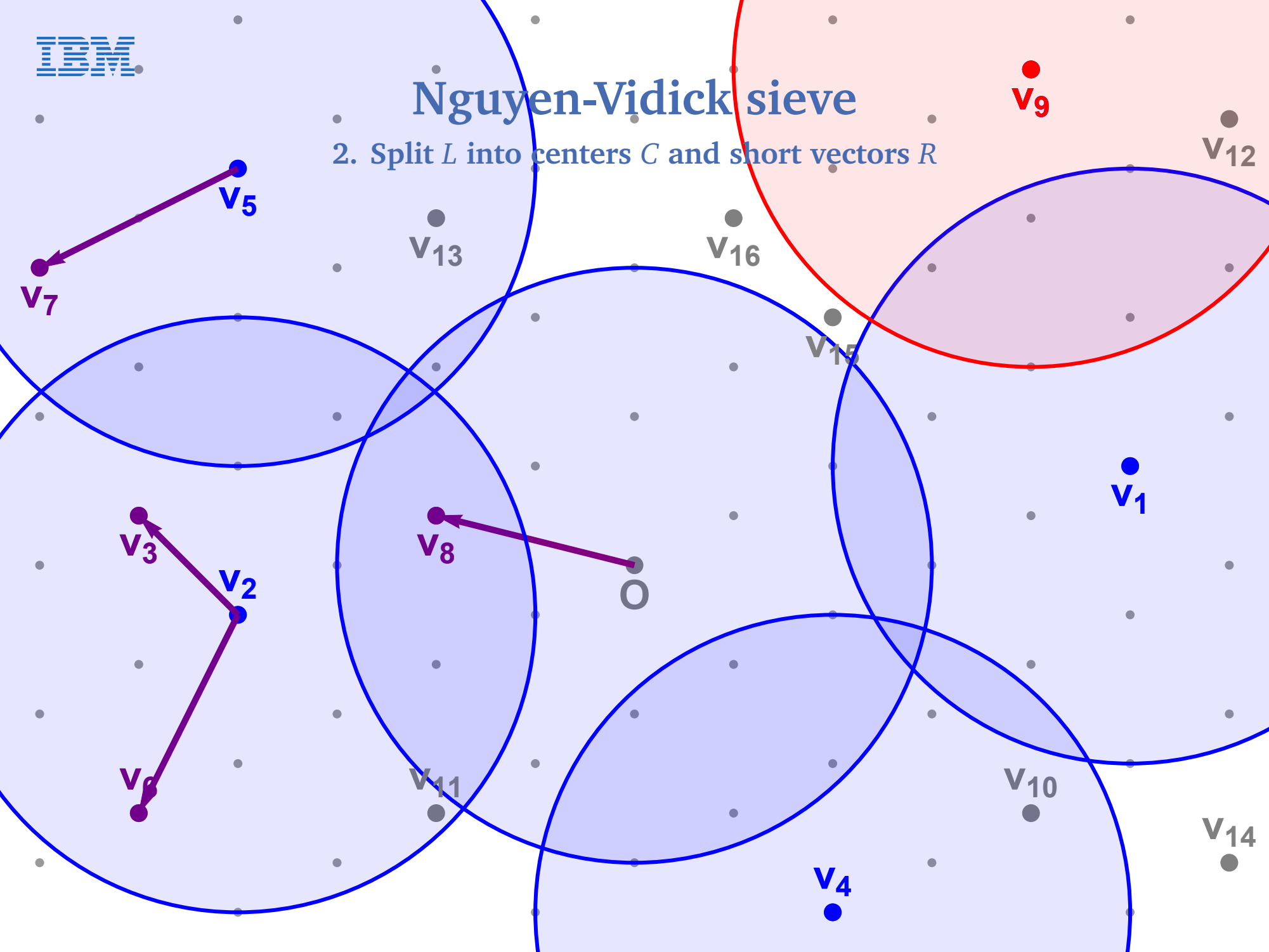
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



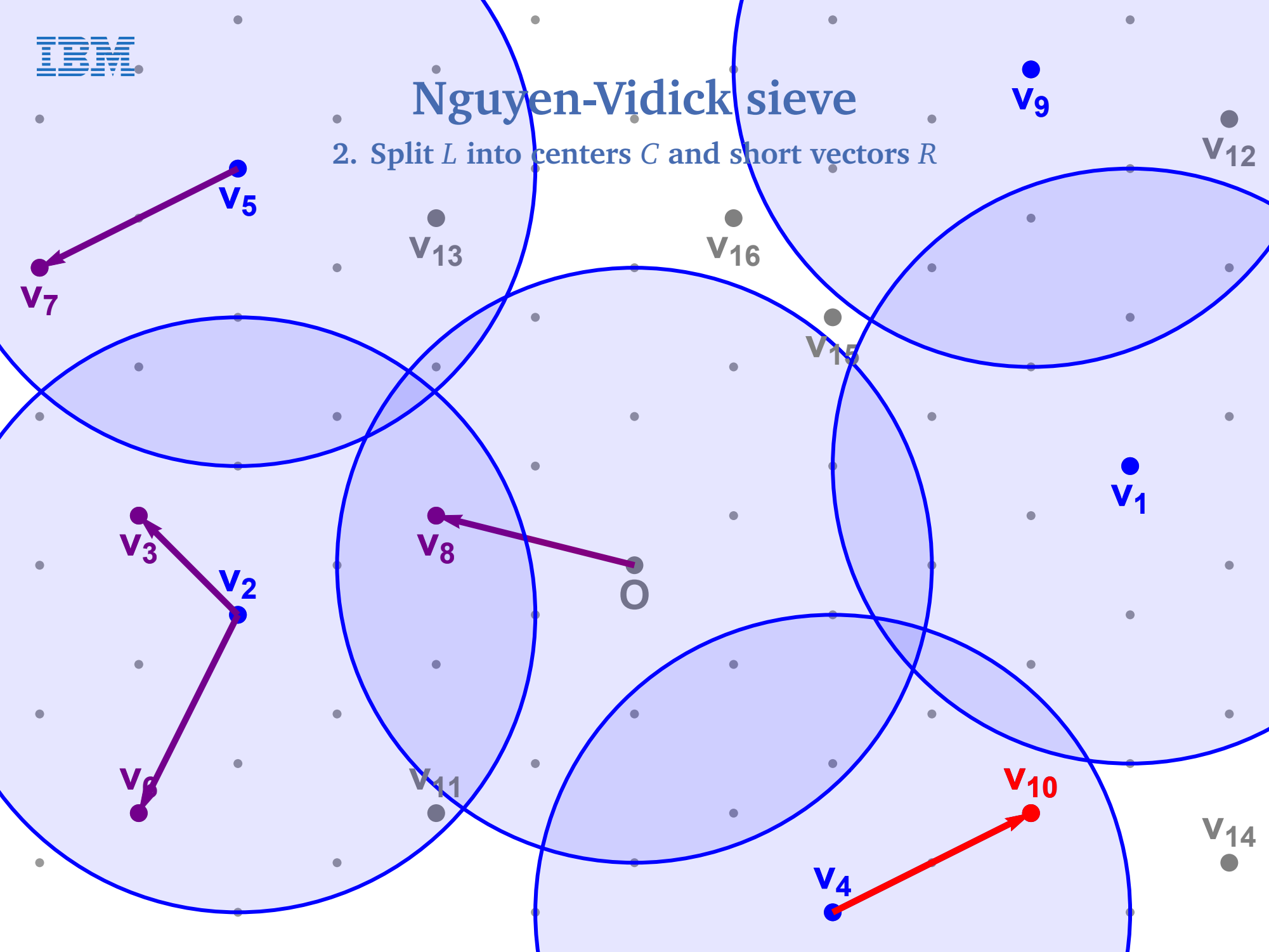
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



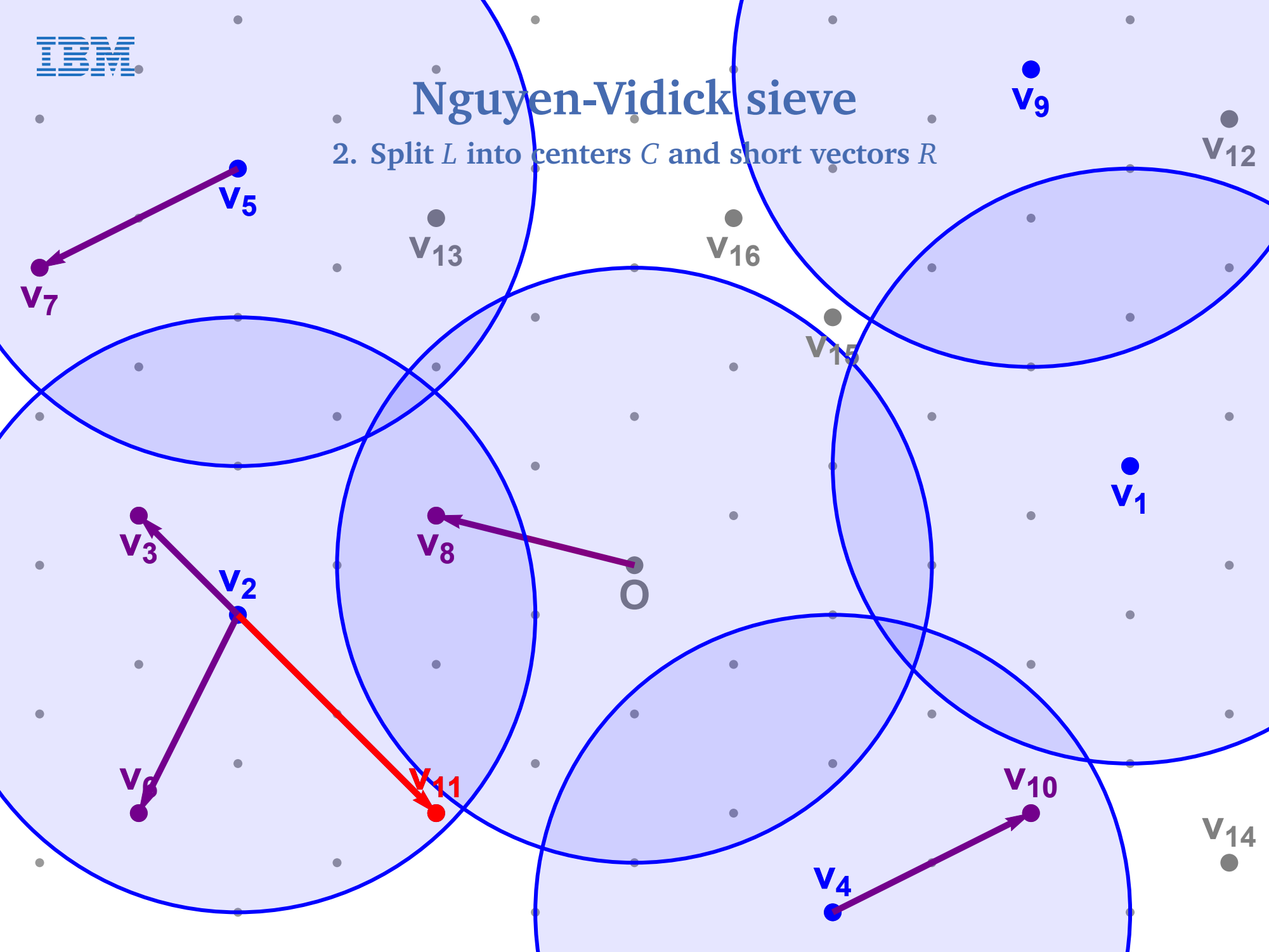
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



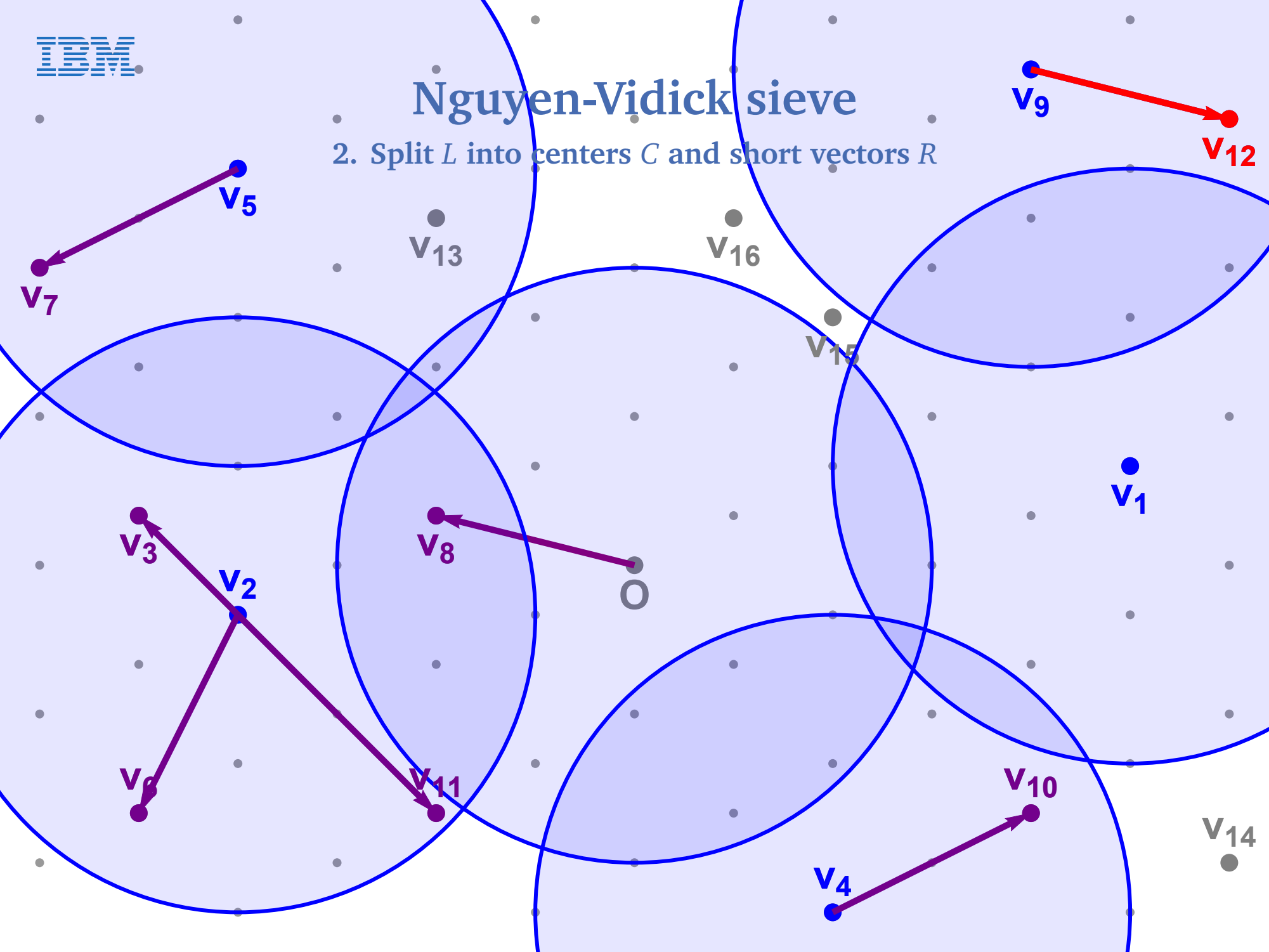
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



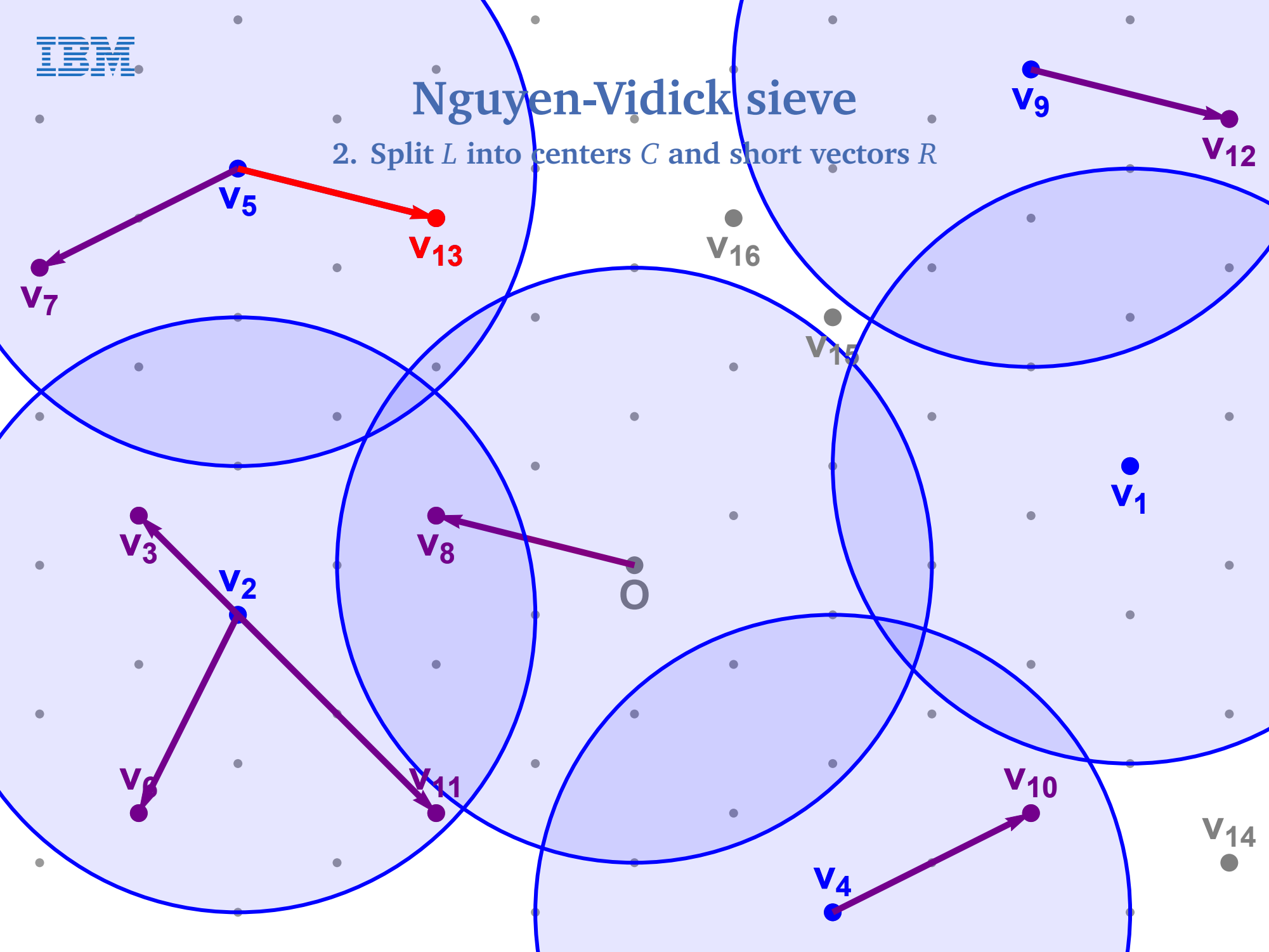
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



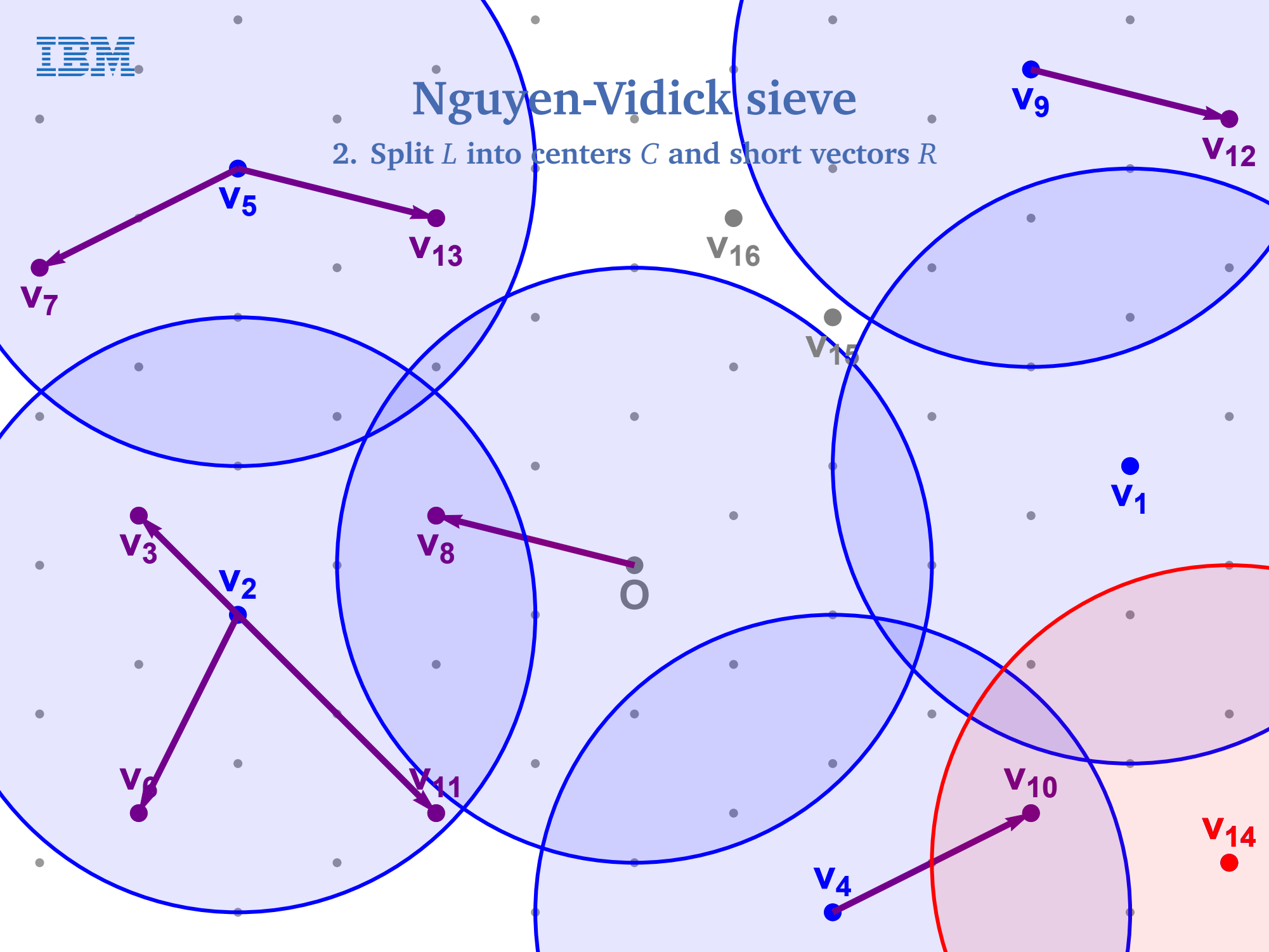
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



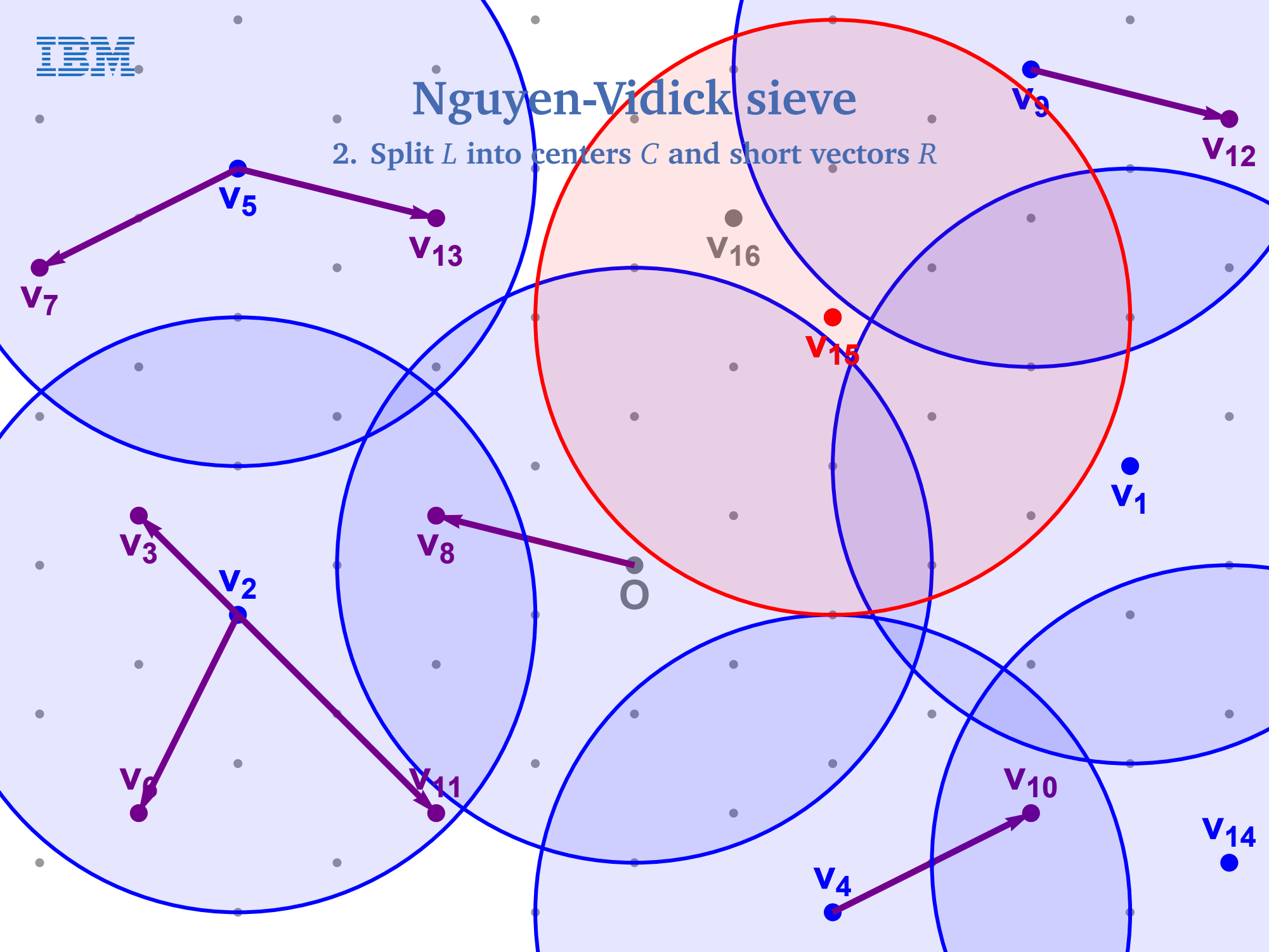
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



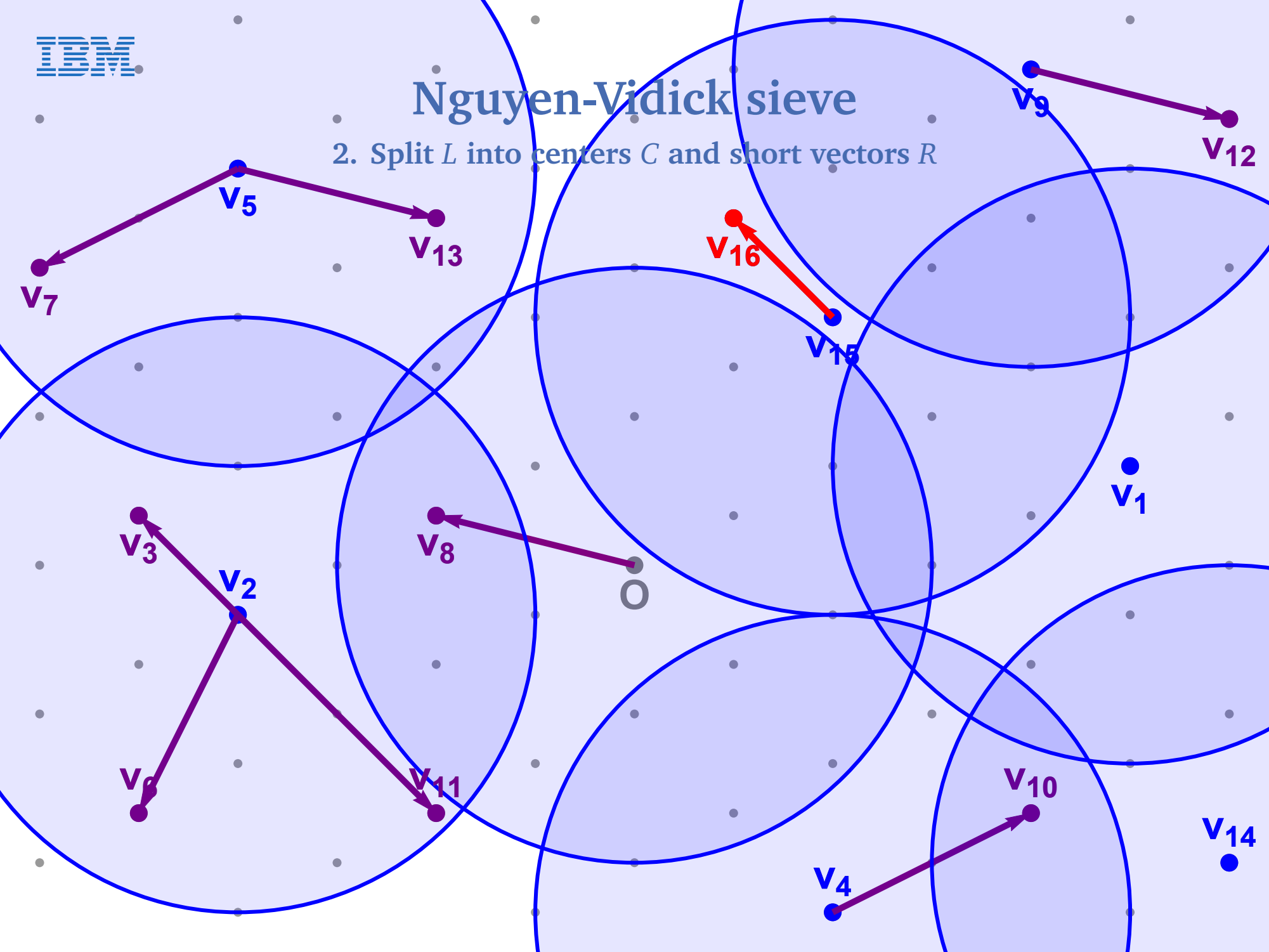
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



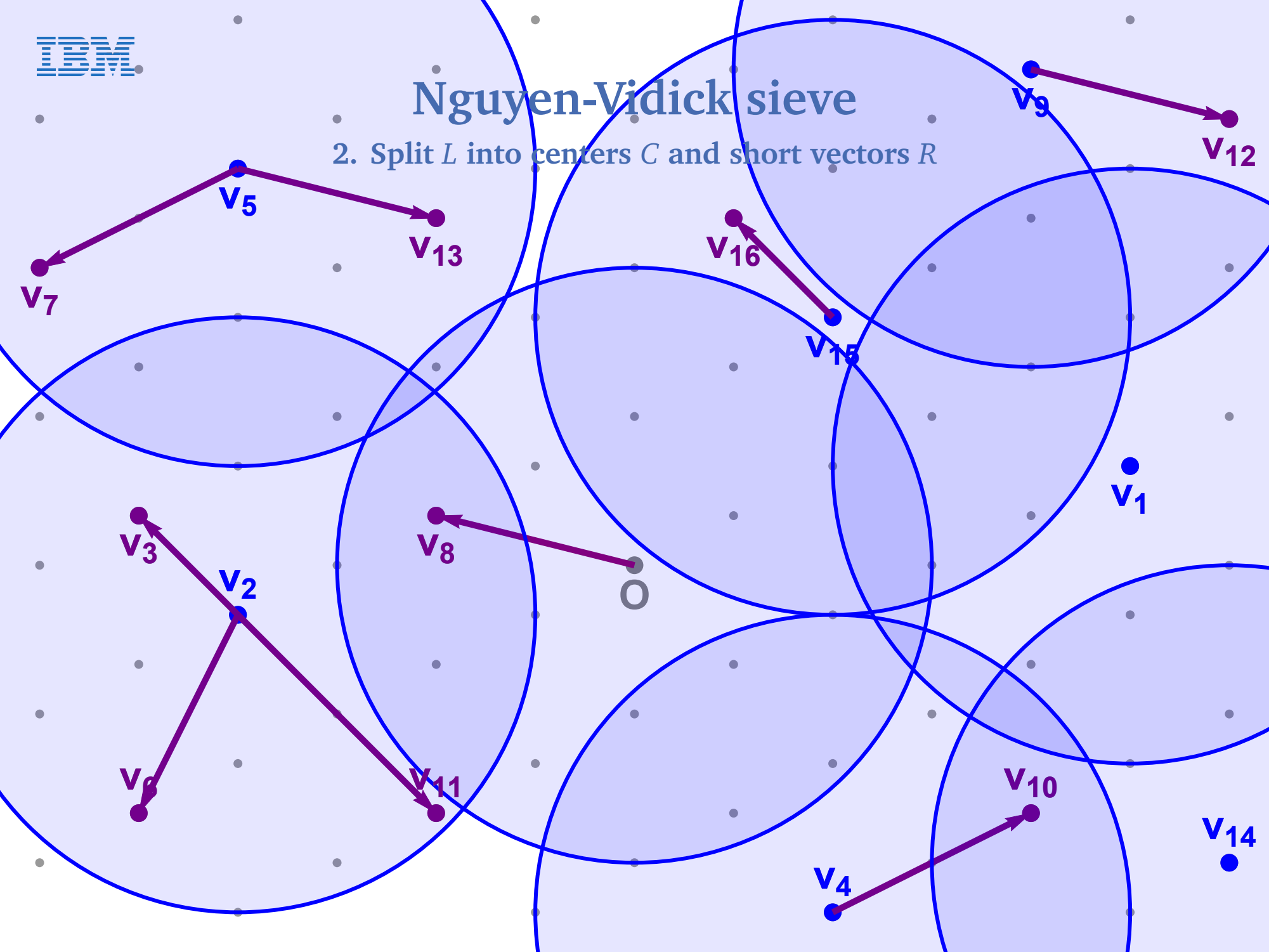
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



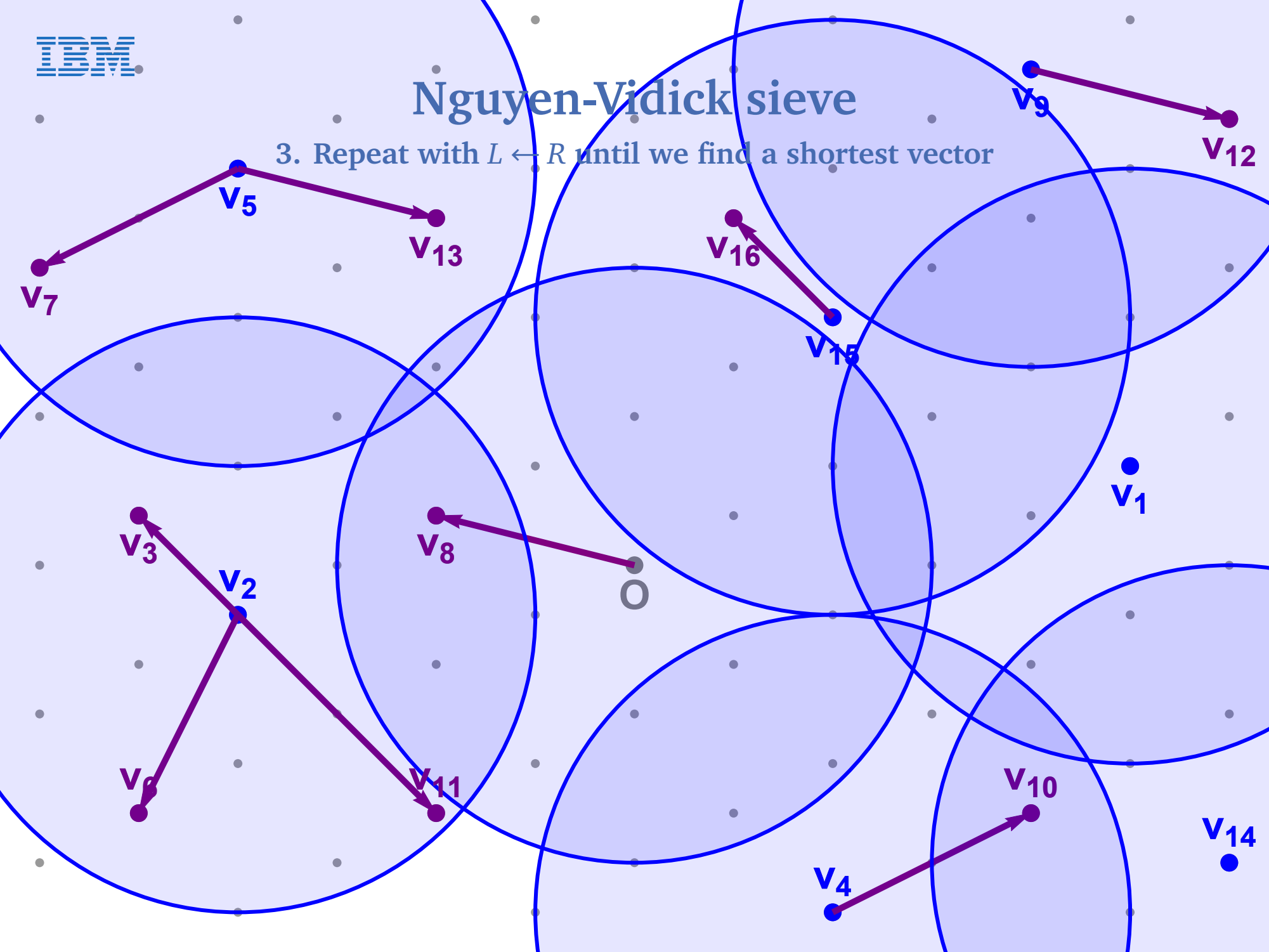
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



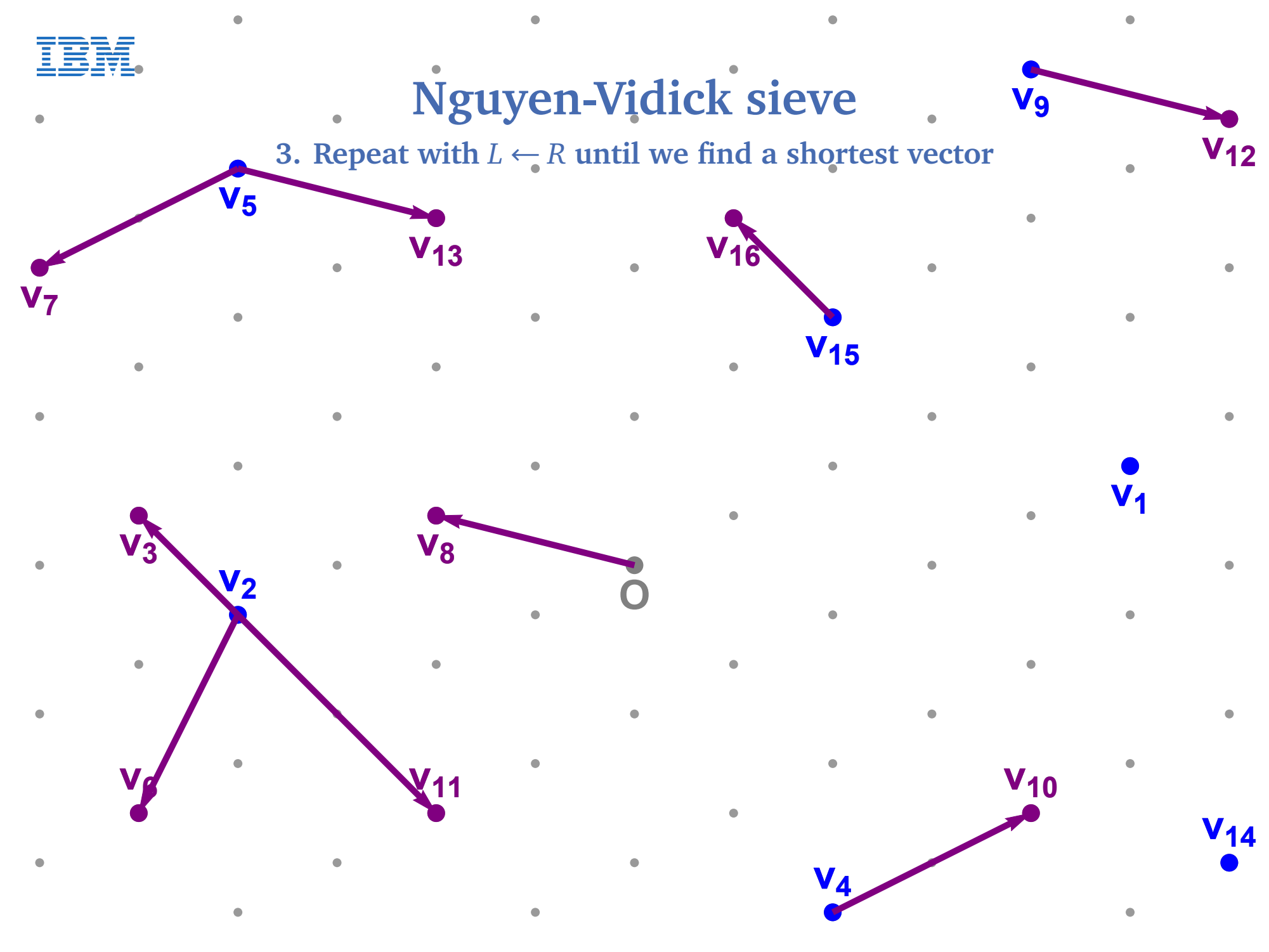
Nguyen-Vidick sieve

3. Repeat with $L \hookleftarrow R$ until we find a shortest vector



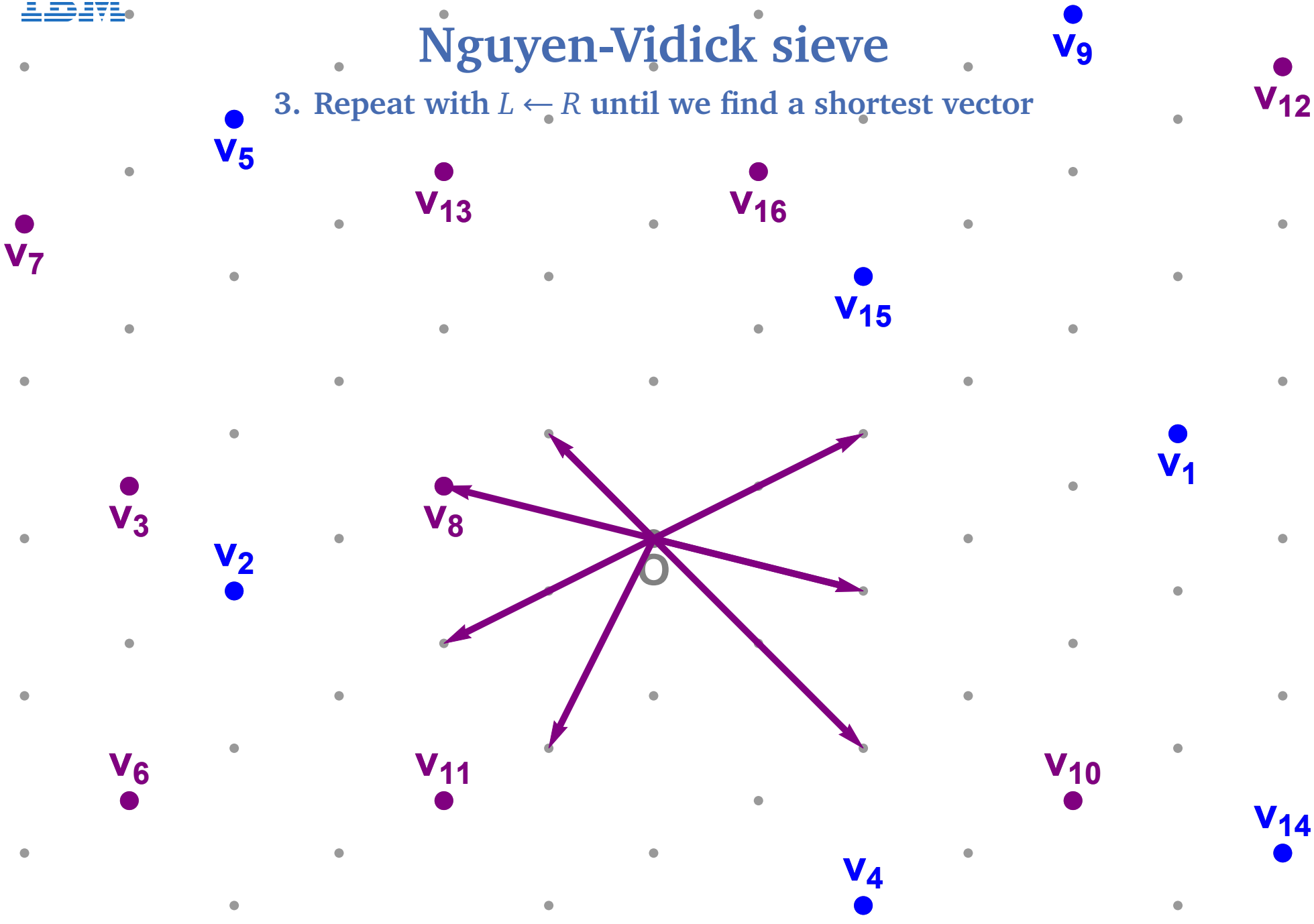
Nguyen-Vidick sieve

3. Repeat with $L \leftarrow R$ until we find a shortest vector



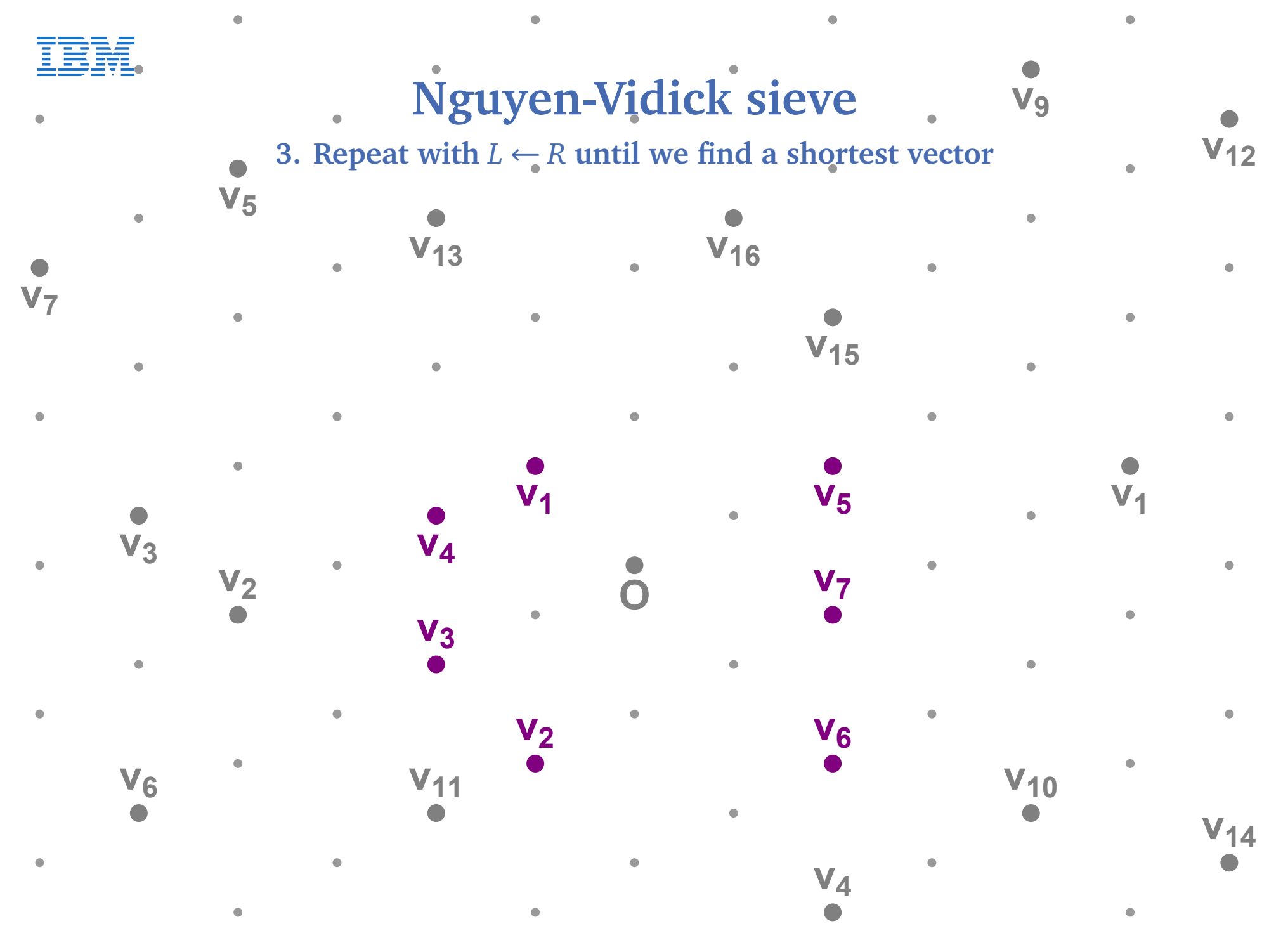
Nguyen-Vidick sieve

3. Repeat with $L \leftarrow R$ until we find a shortest vector



Nguyen-Vidick sieve

3. Repeat with $L \leftarrow R$ until we find a shortest vector



Nguyen-Vidick sieve

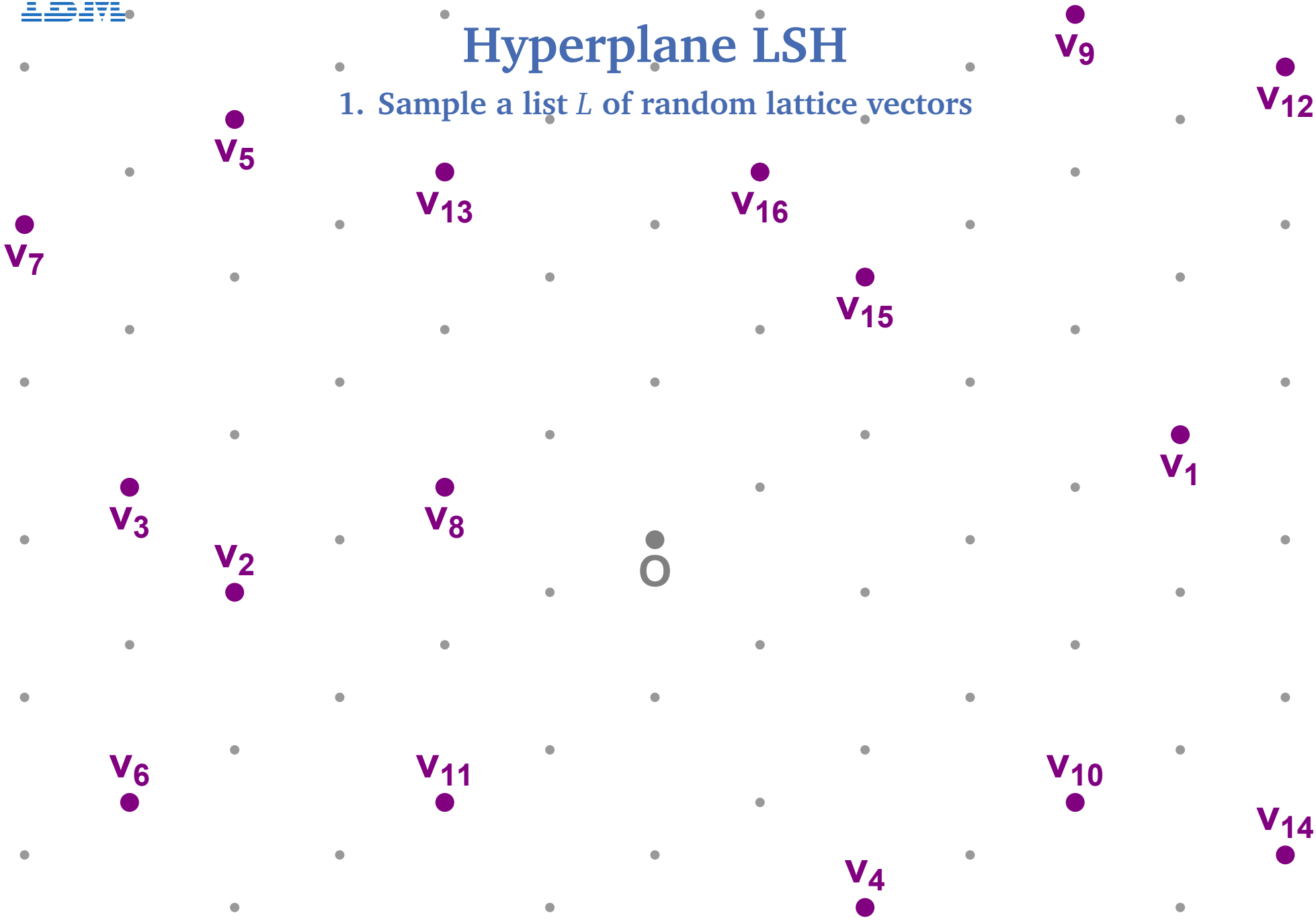
Overview

Hyperplane LSH

1. Sample a list L of random lattice vectors

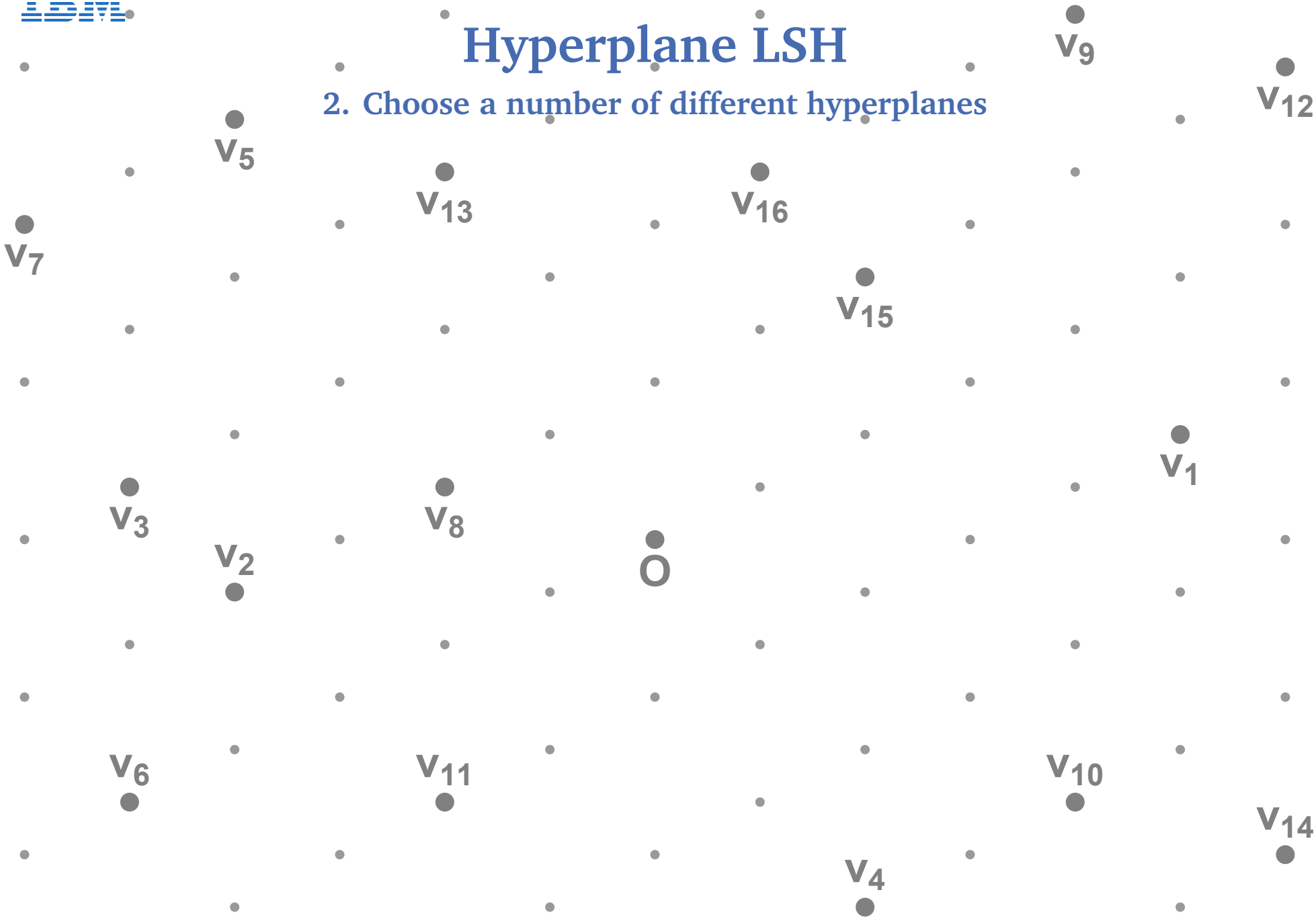
Hyperplane LSH

1. Sample a list L of random lattice vectors



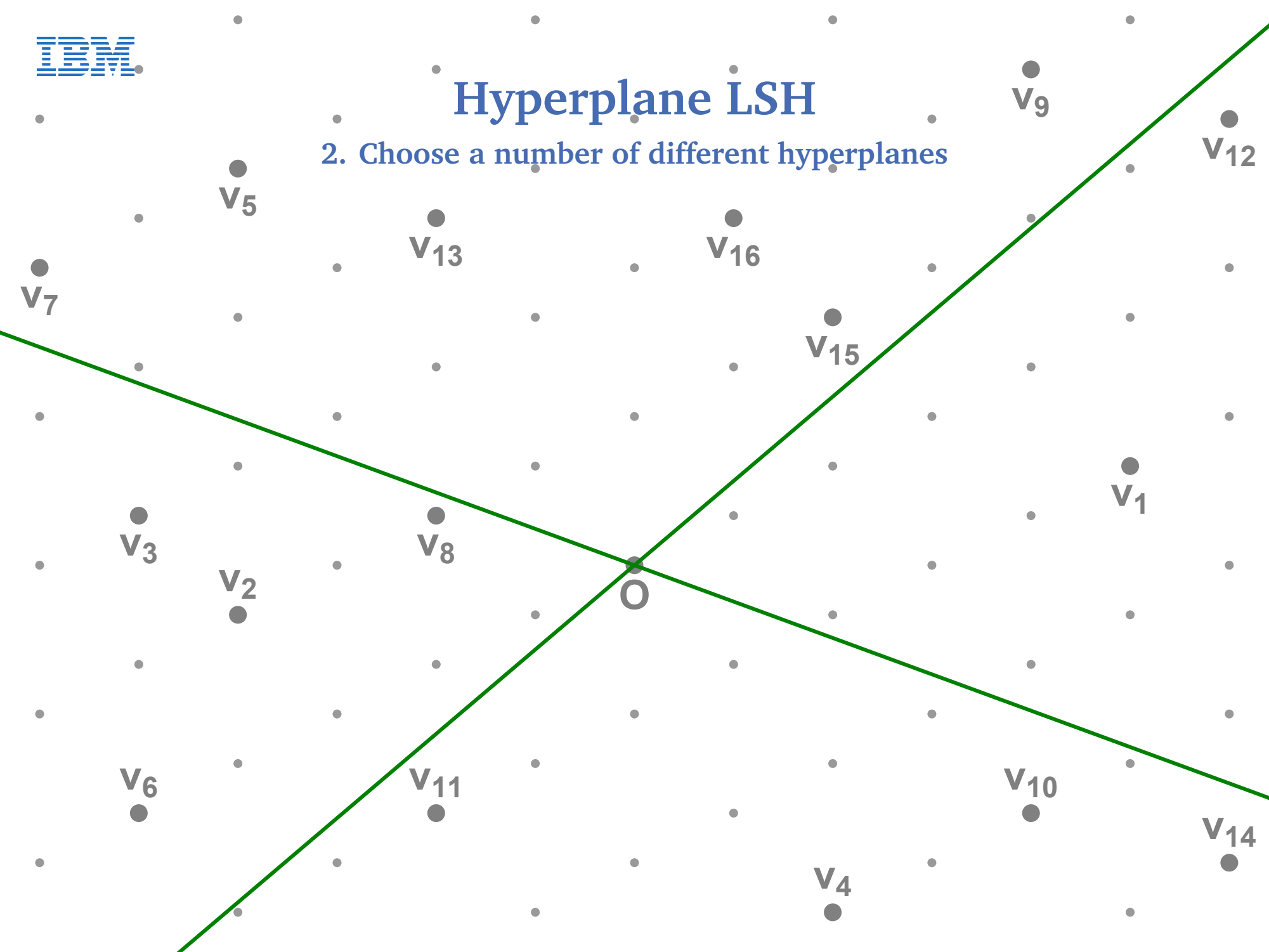
Hyperplane LSH

2. Choose a number of different hyperplanes



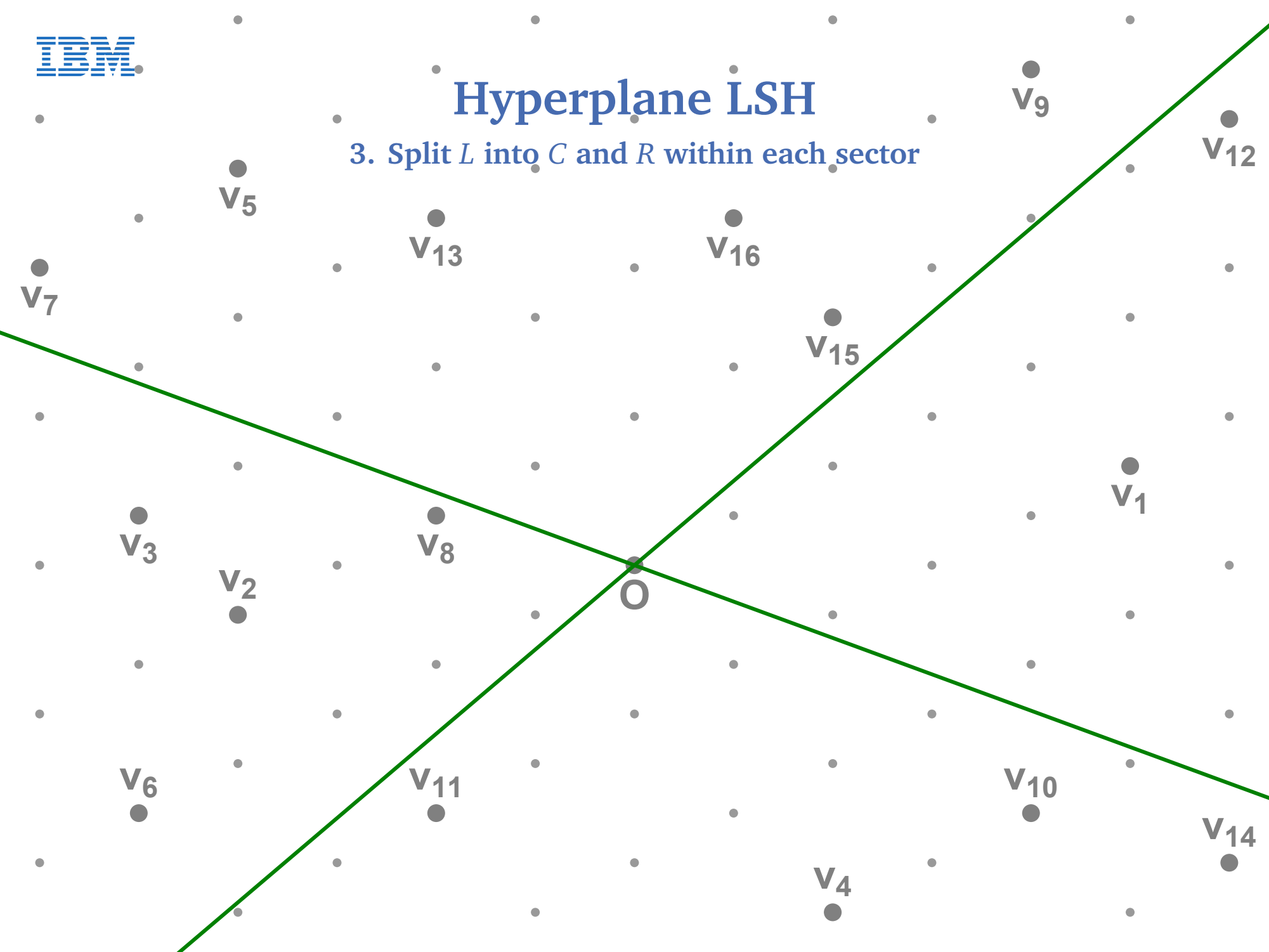
Hyperplane LSH

2. Choose a number of different hyperplanes



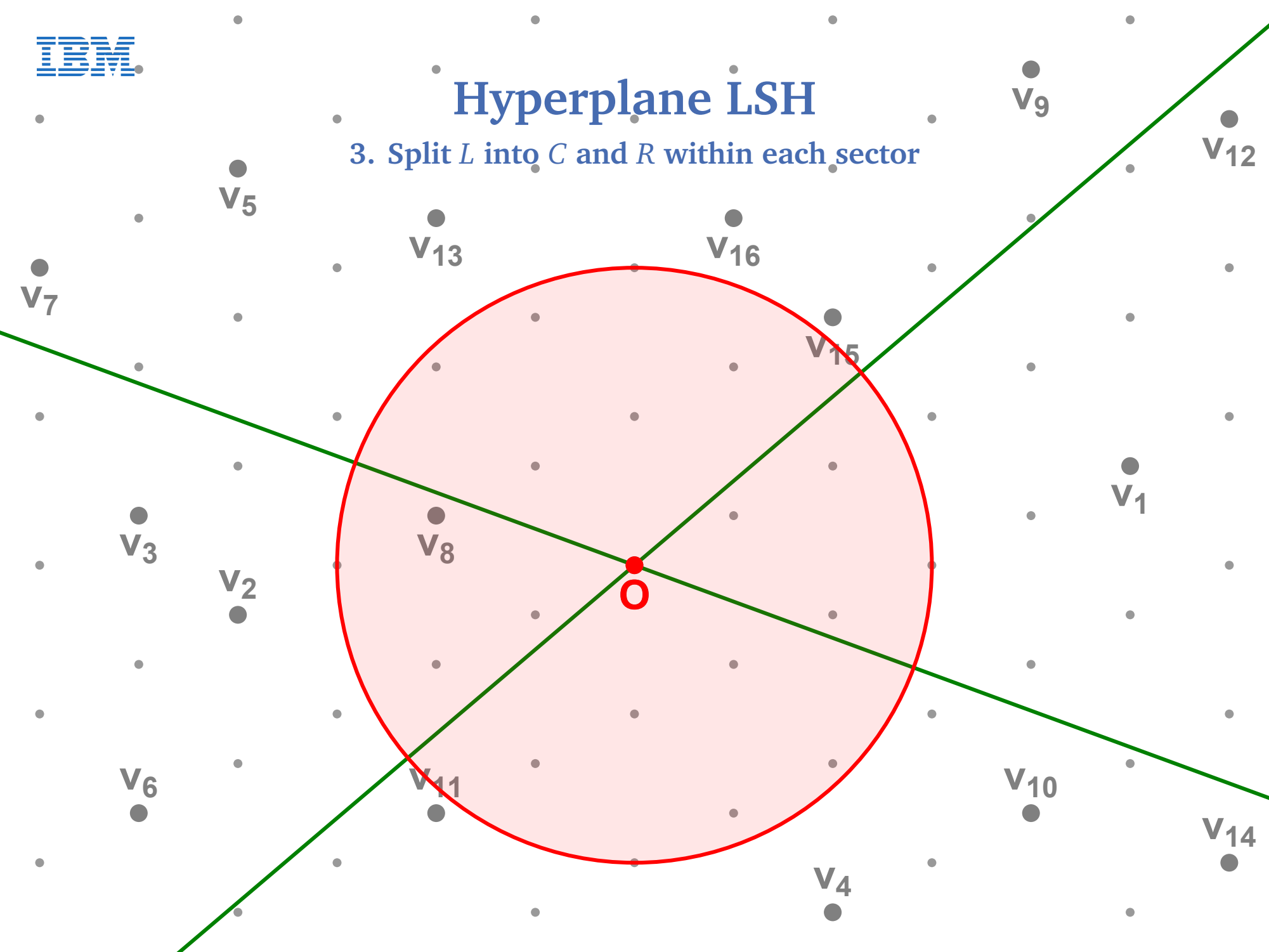
Hyperplane LSH

3. Split L into C and R within each sector



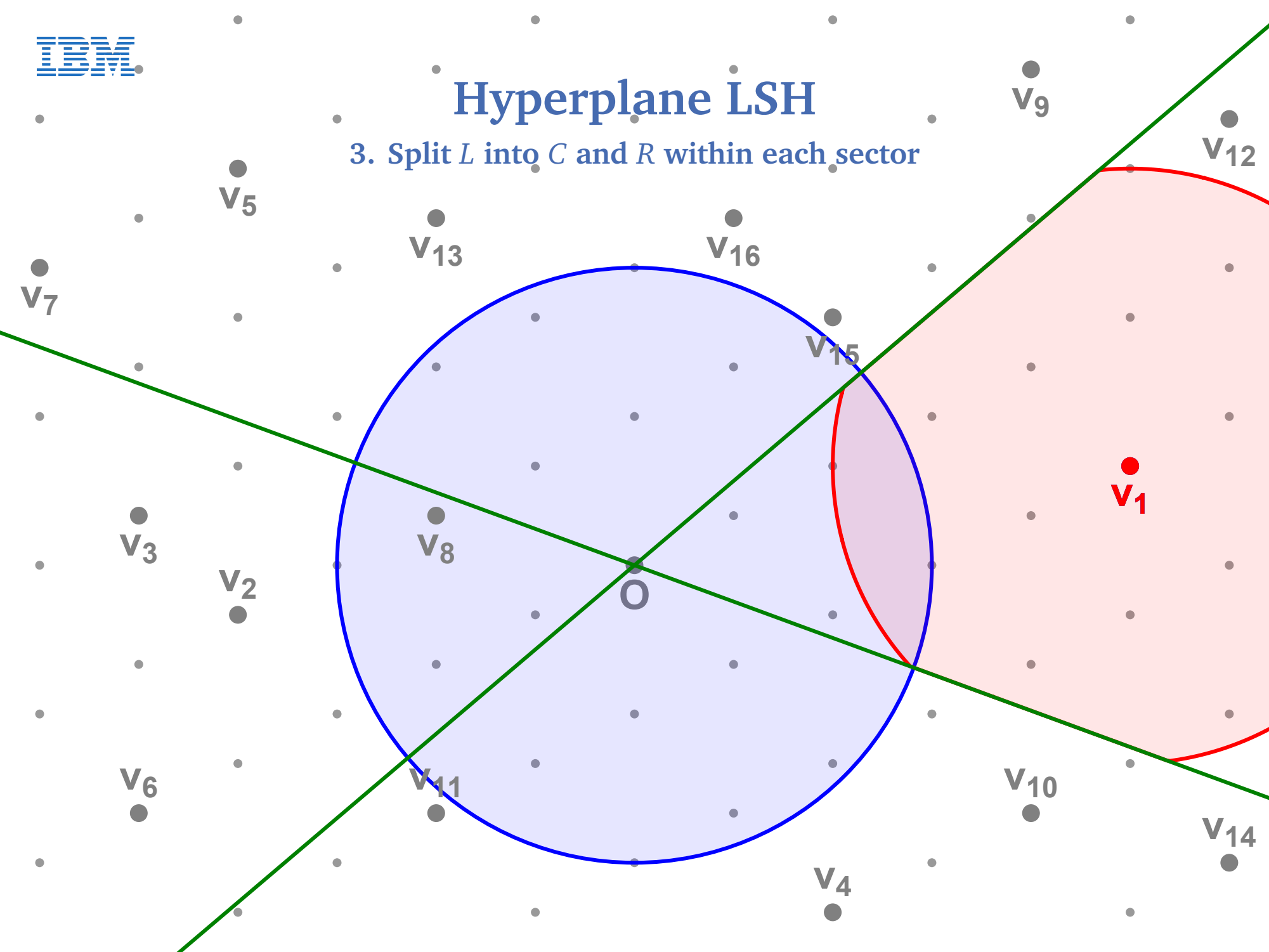
Hyperplane LSH

3. Split L into C and R within each sector



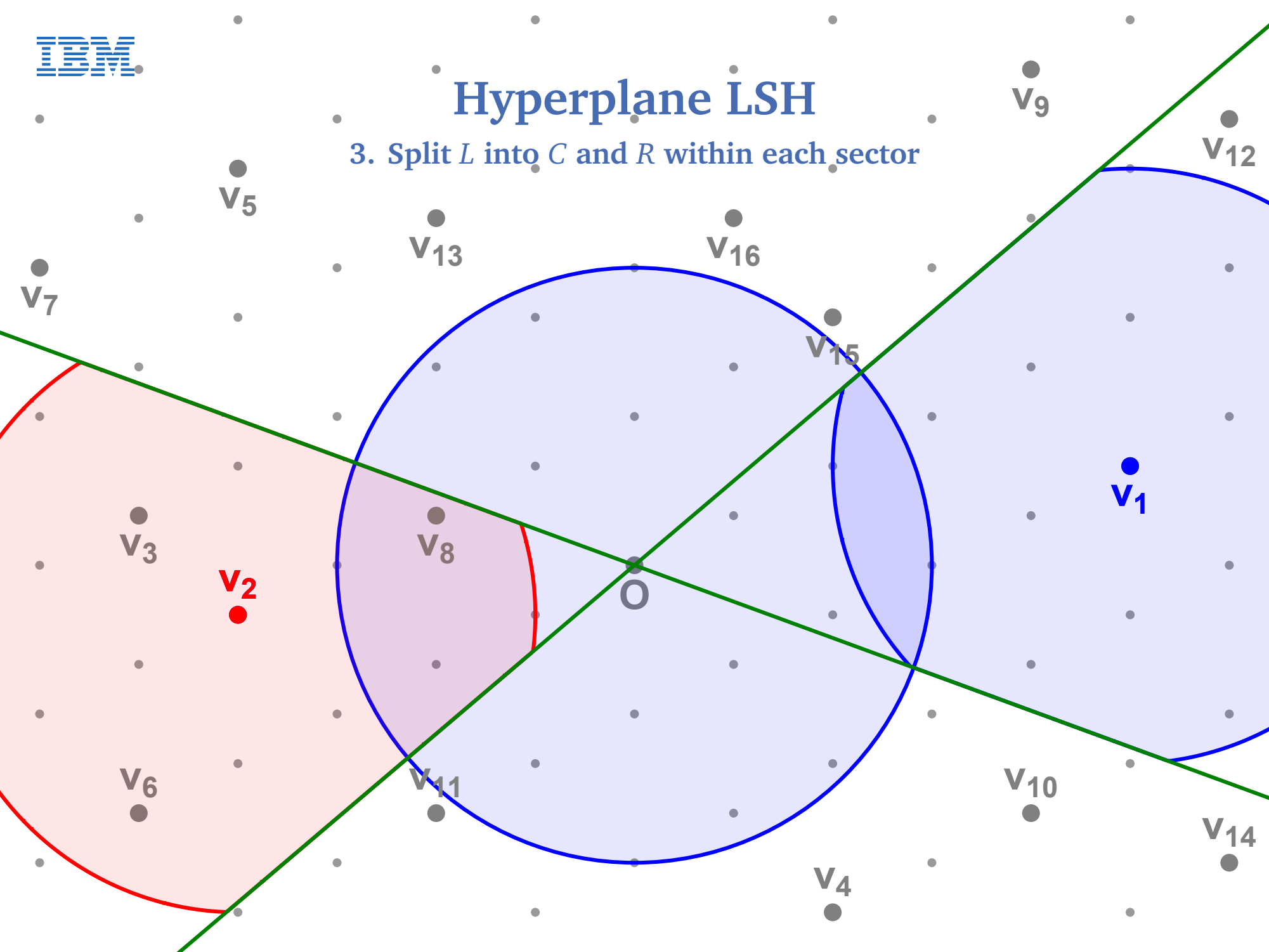
Hyperplane LSH

3. Split L into C and R within each sector



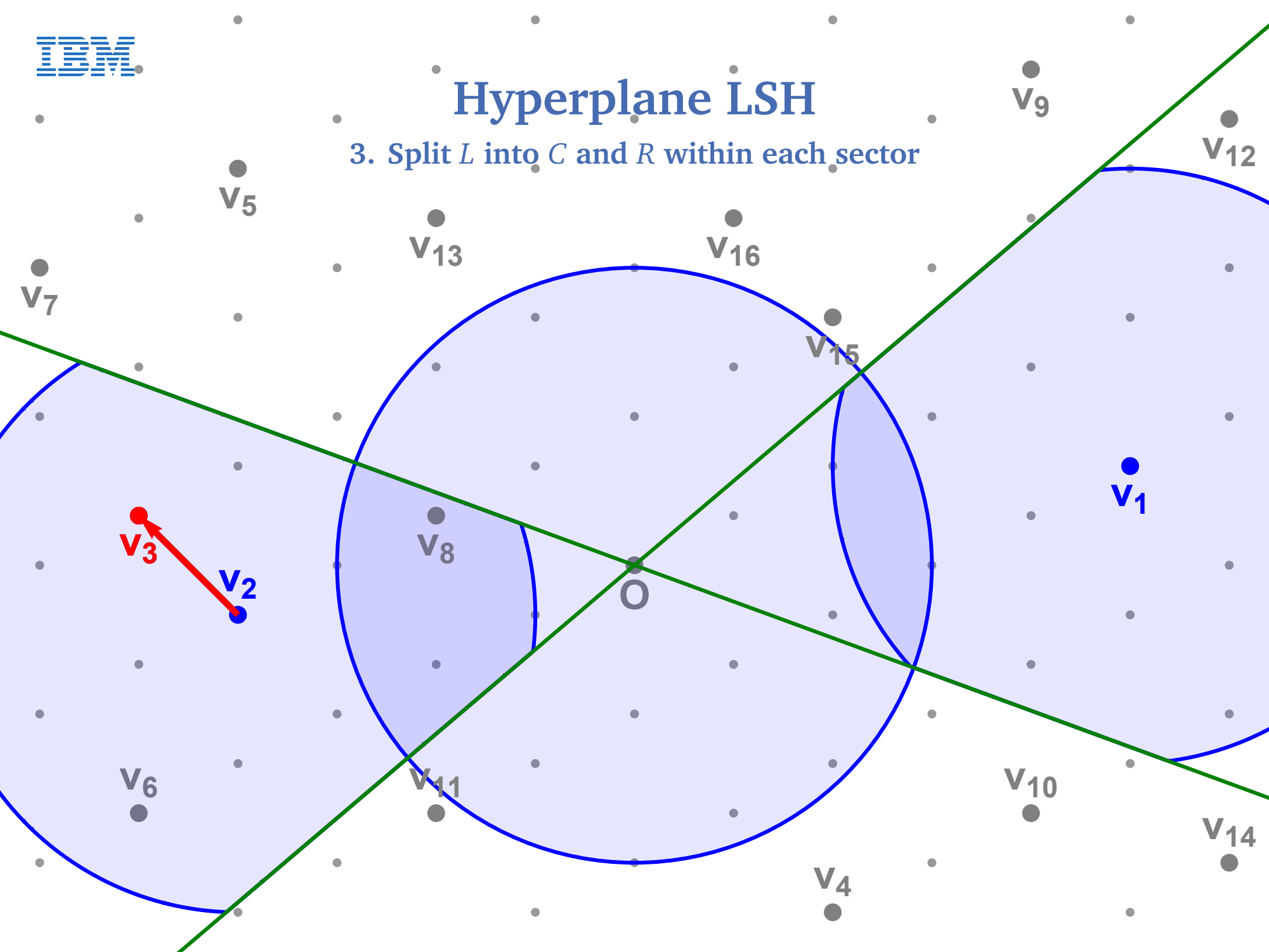
Hyperplane LSH

3. Split L into C and R within each sector



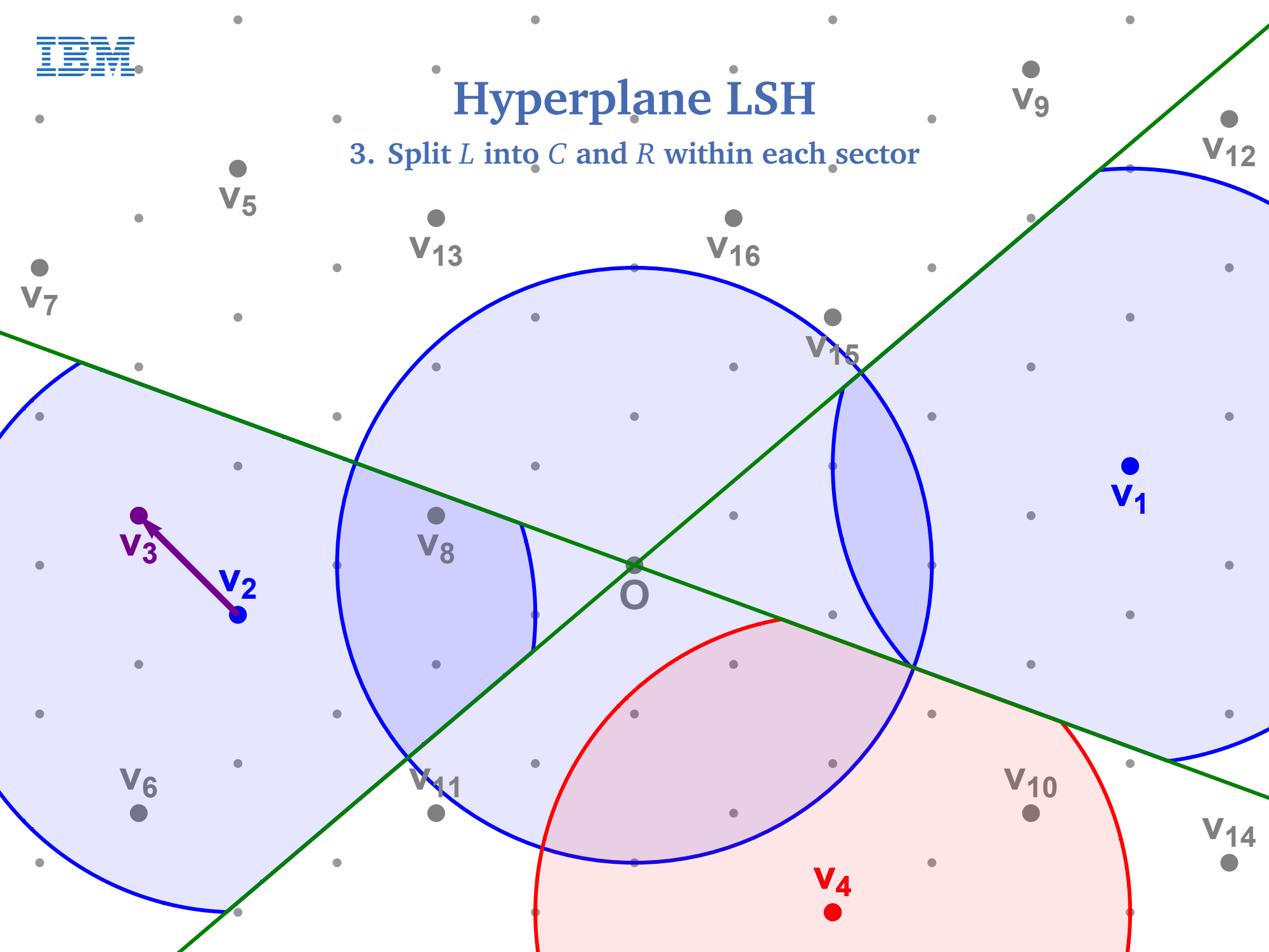
Hyperplane LSH

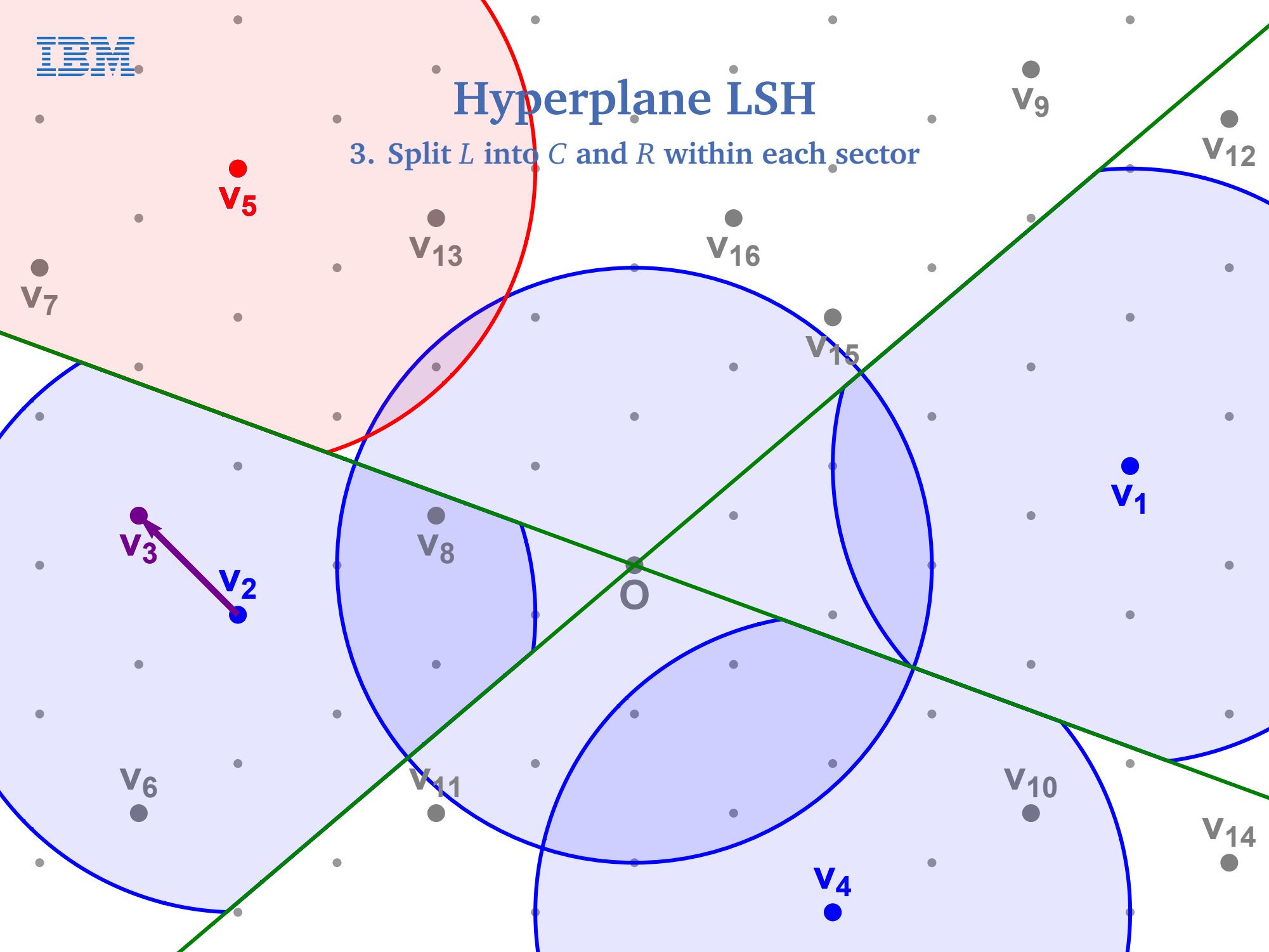
3. Split L into C and R within each sector



Hyperplane LSH

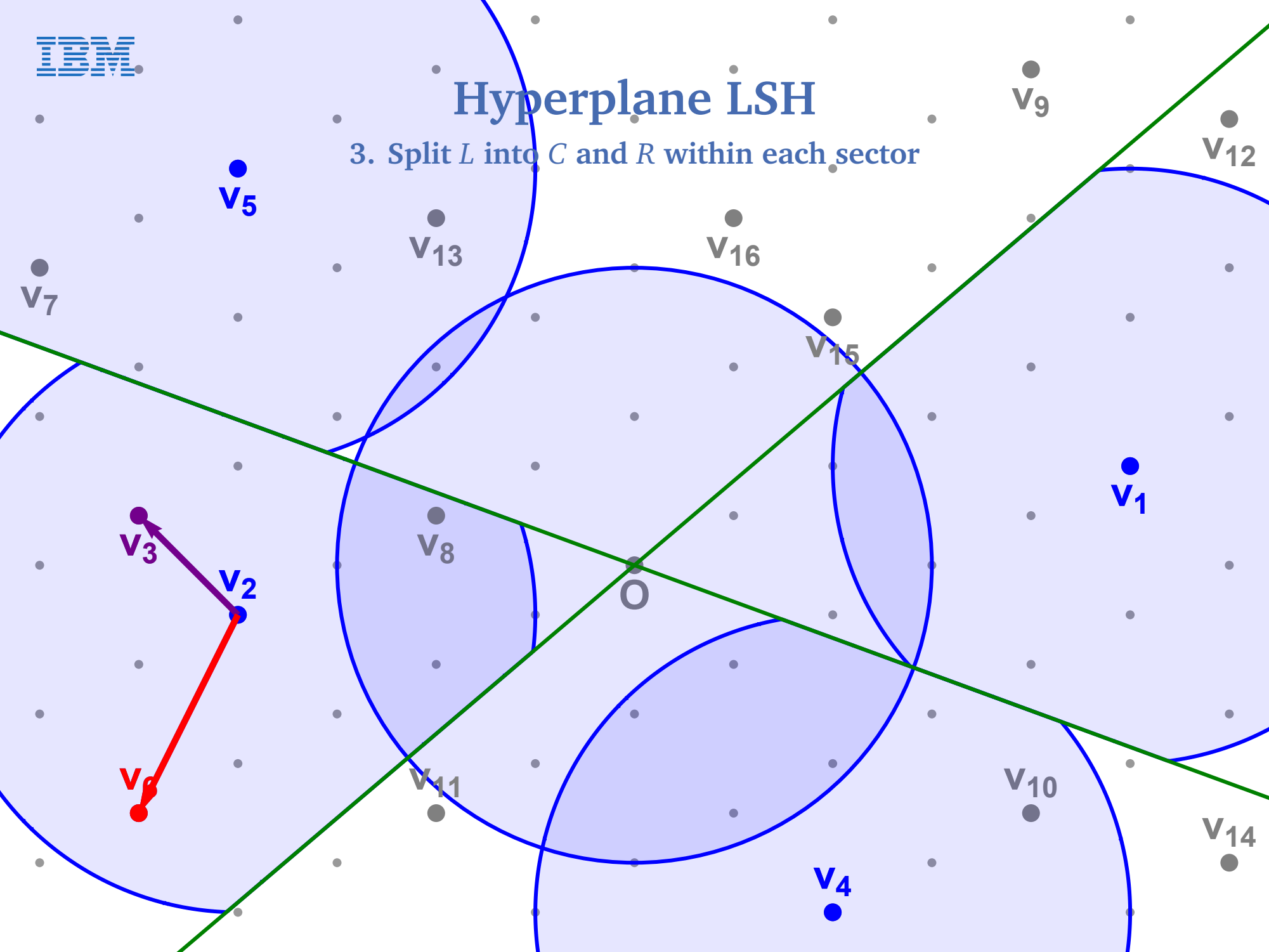
3. Split L into C and R within each sector




IBM
Hyperplane LSH
3. Split L into C and R within each sector
v5
v13
v16
v9
v12
v7
v15
v1
v3
v2
v8
O
v6
v11
v10
v14
v4

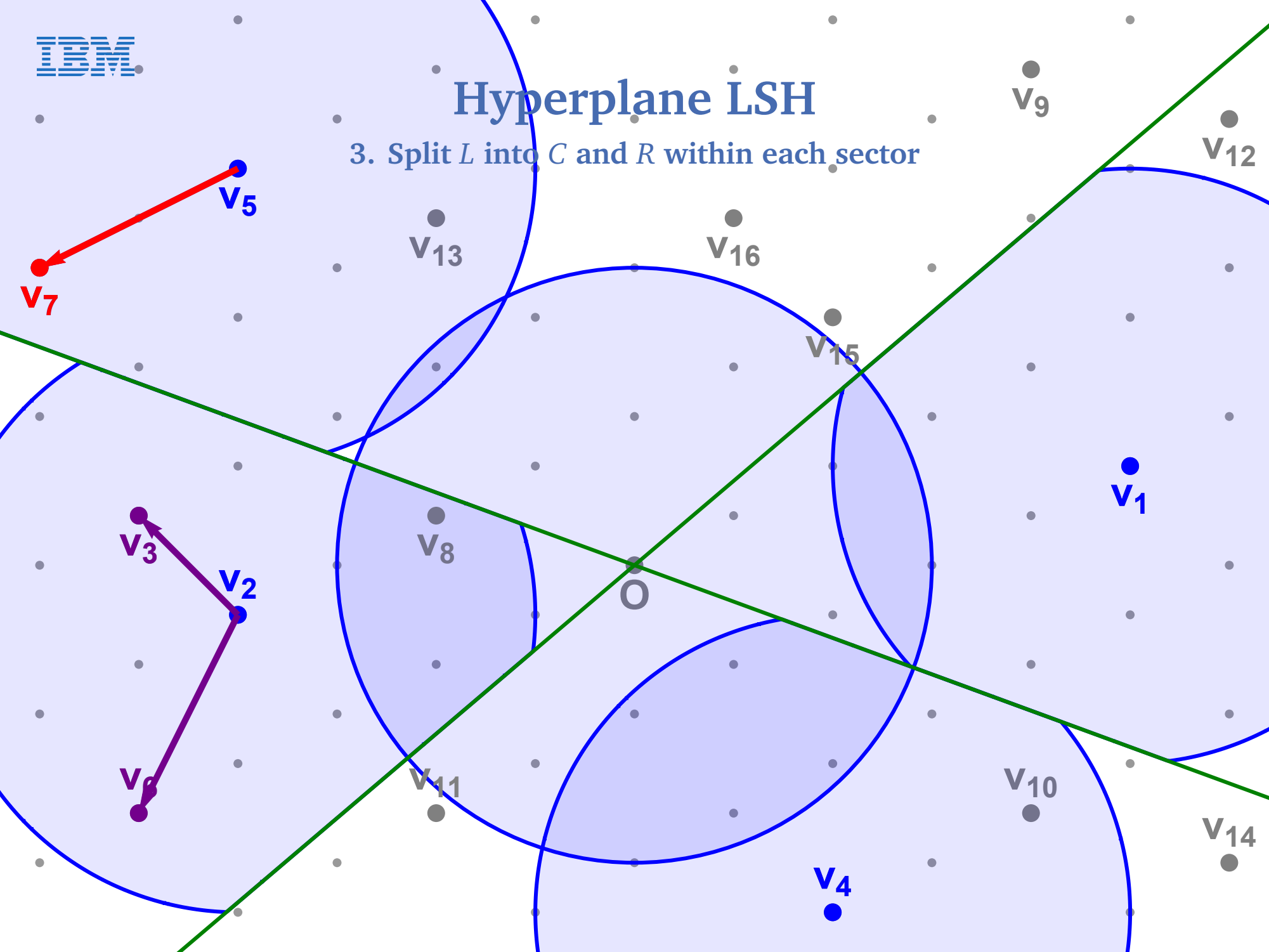
Hyperplane LSH

3. Split L into C and R within each sector



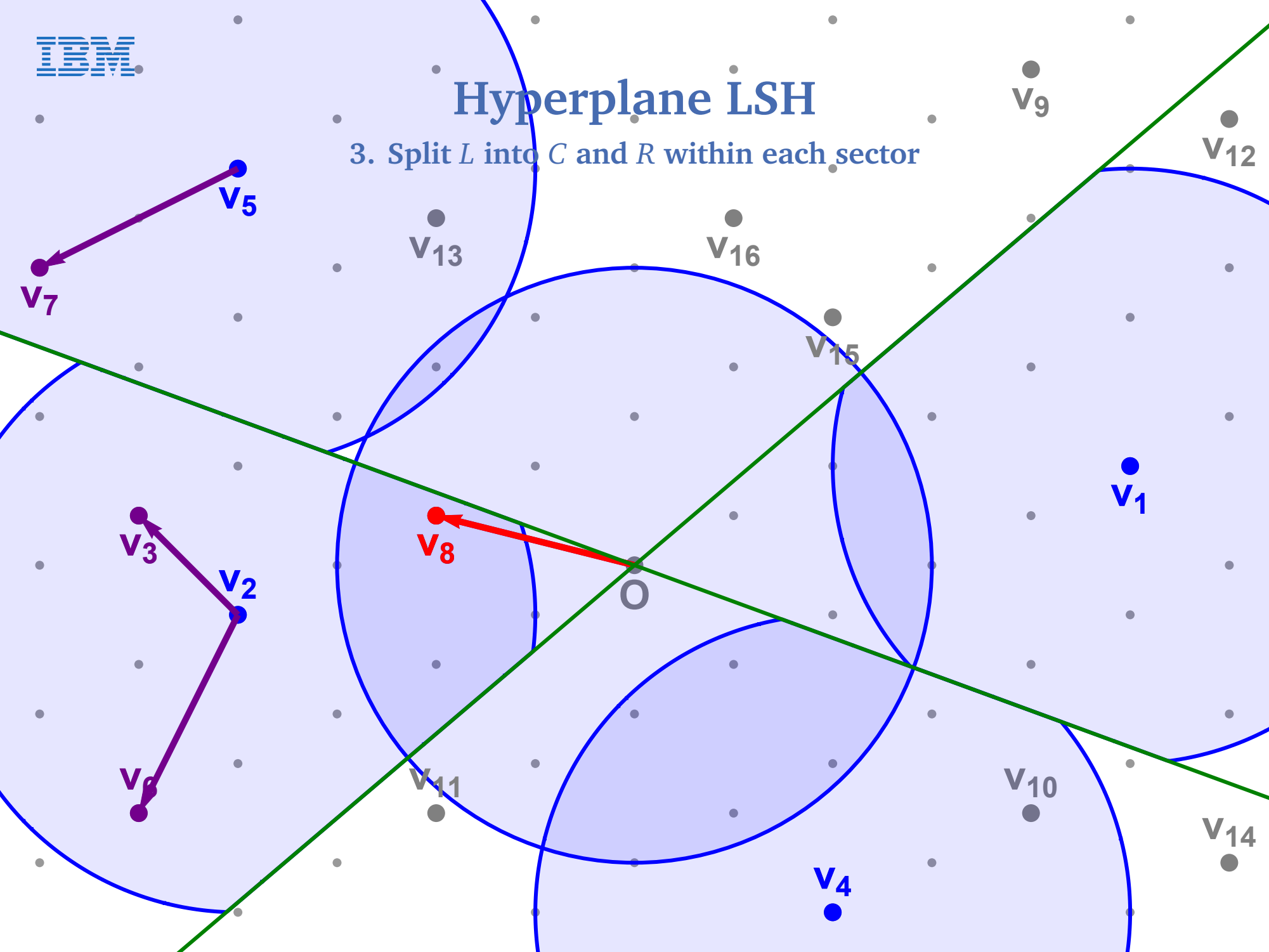
Hyperplane LSH

3. Split L into C and R within each sector



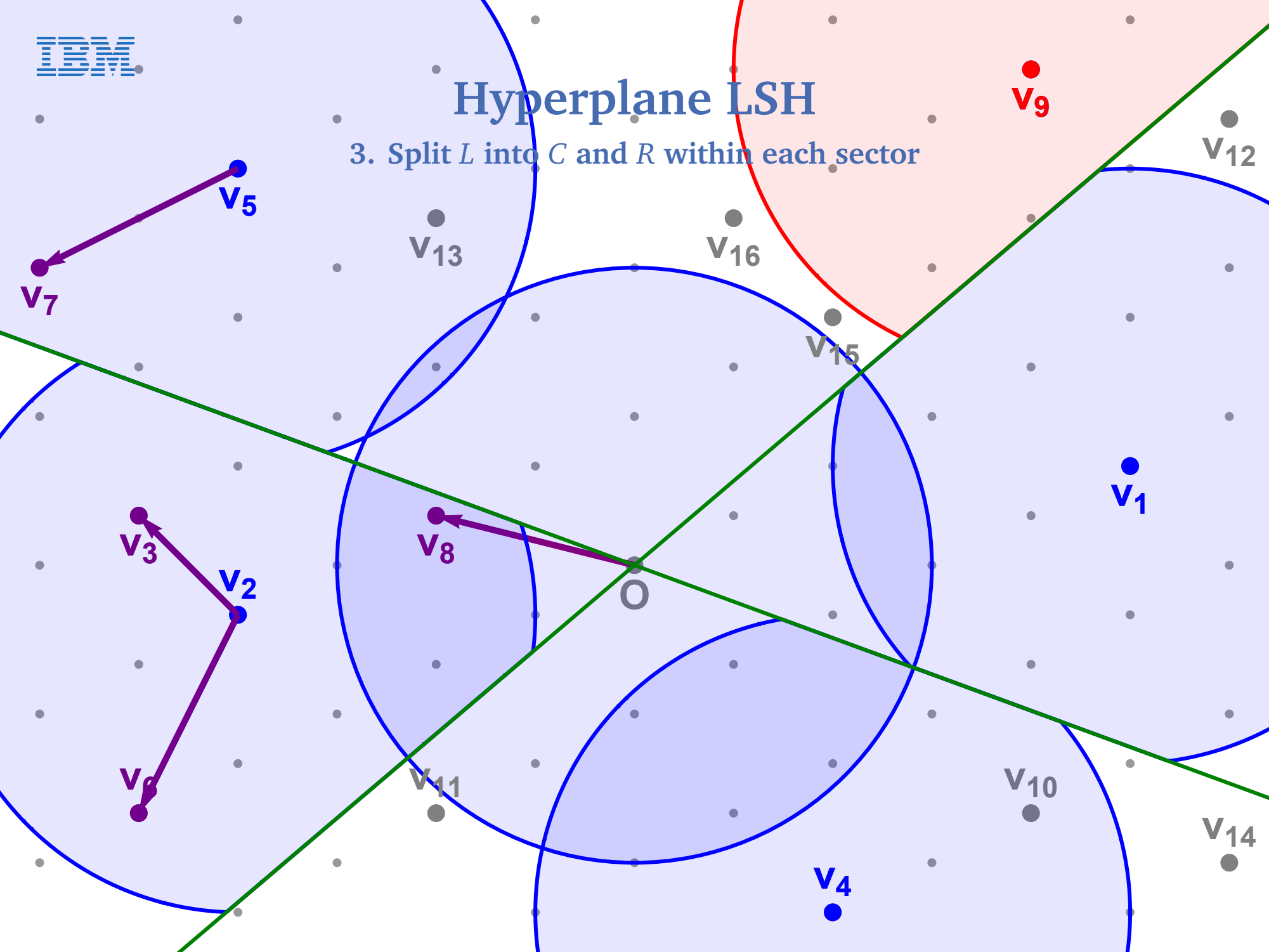
Hyperplane LSH

3. Split L into C and R within each sector



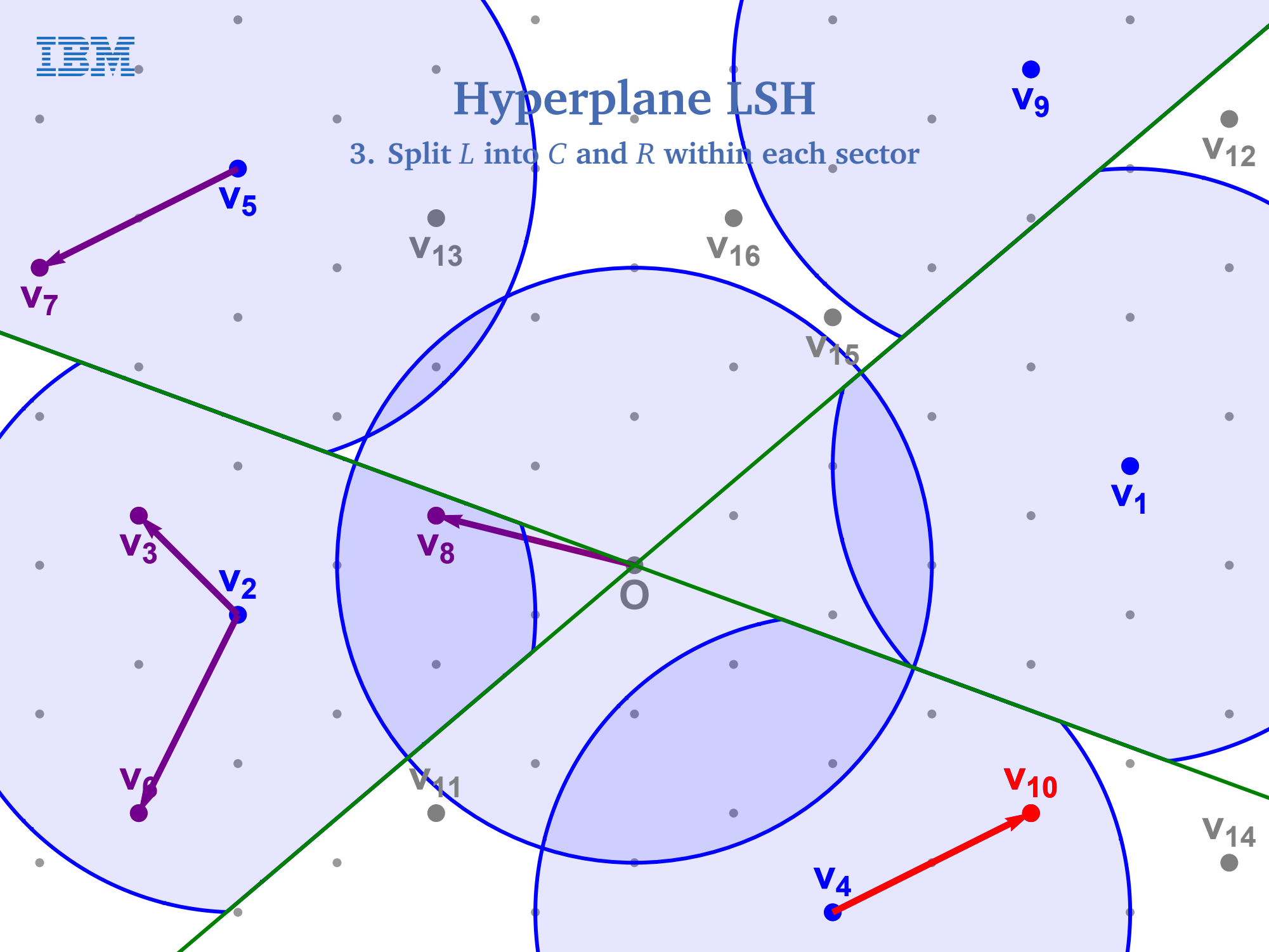
Hyperplane LSH

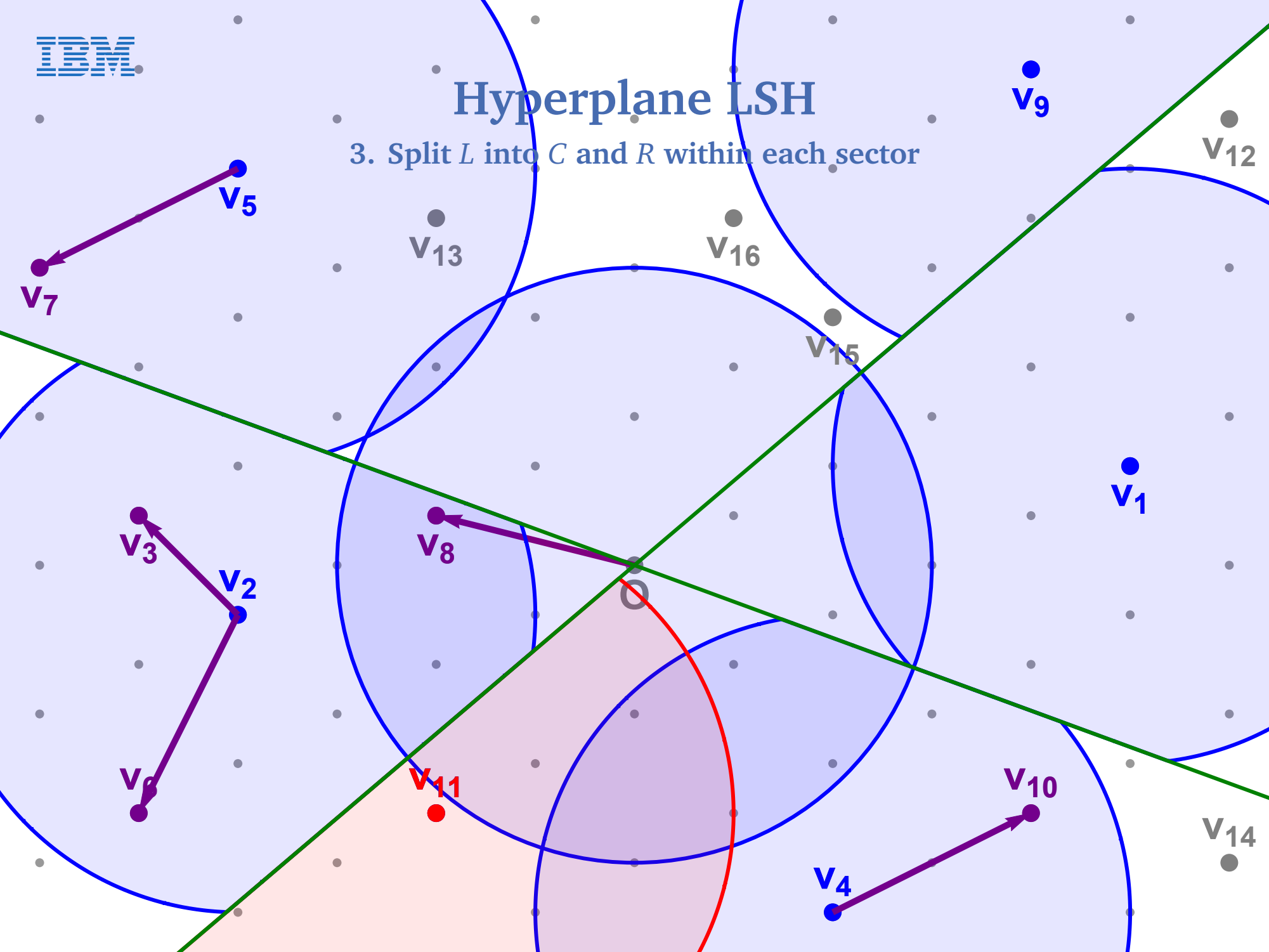
3. Split L into C and R within each sector



Hyperplane LSH

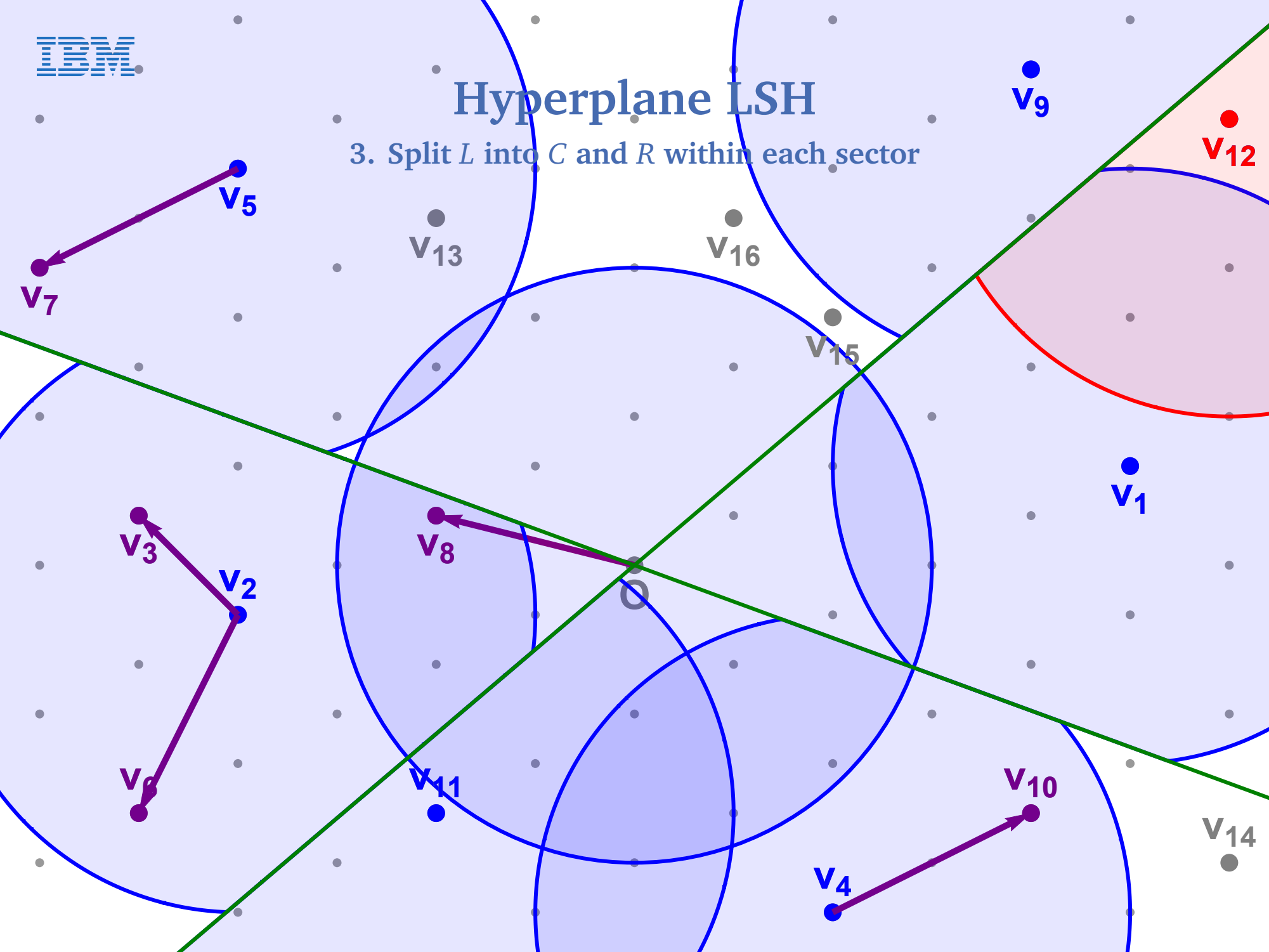
3. Split L into C and R within each sector




IBM
Hyperplane LSH
3. Split L into C and R within each sector
v9
v12
v5
v13
v16
v7
v15
v1
v3
v8
v2
O
v6
v11
v10
v14
v4

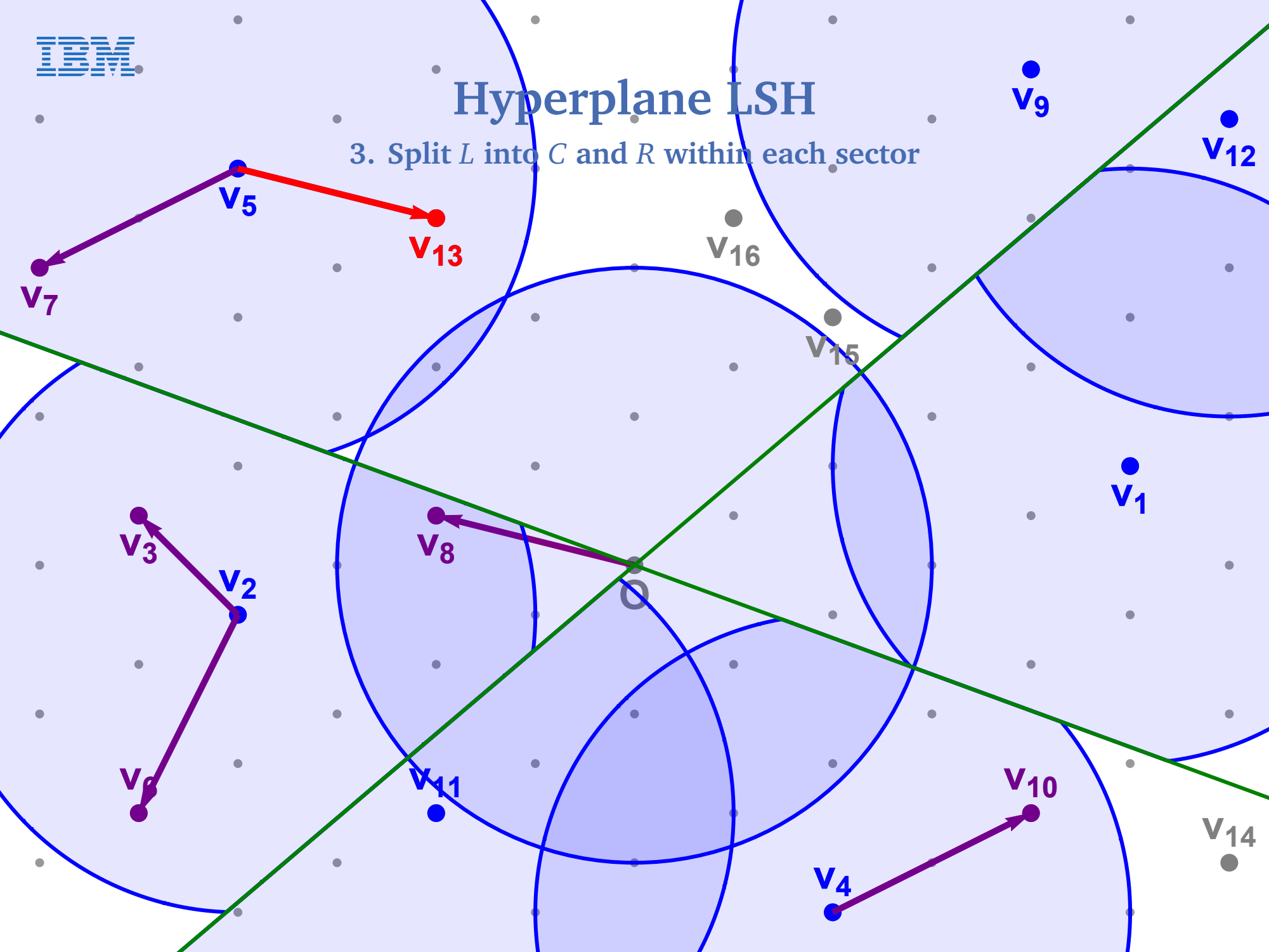
Hyperplane LSH

3. Split L into C and R within each sector



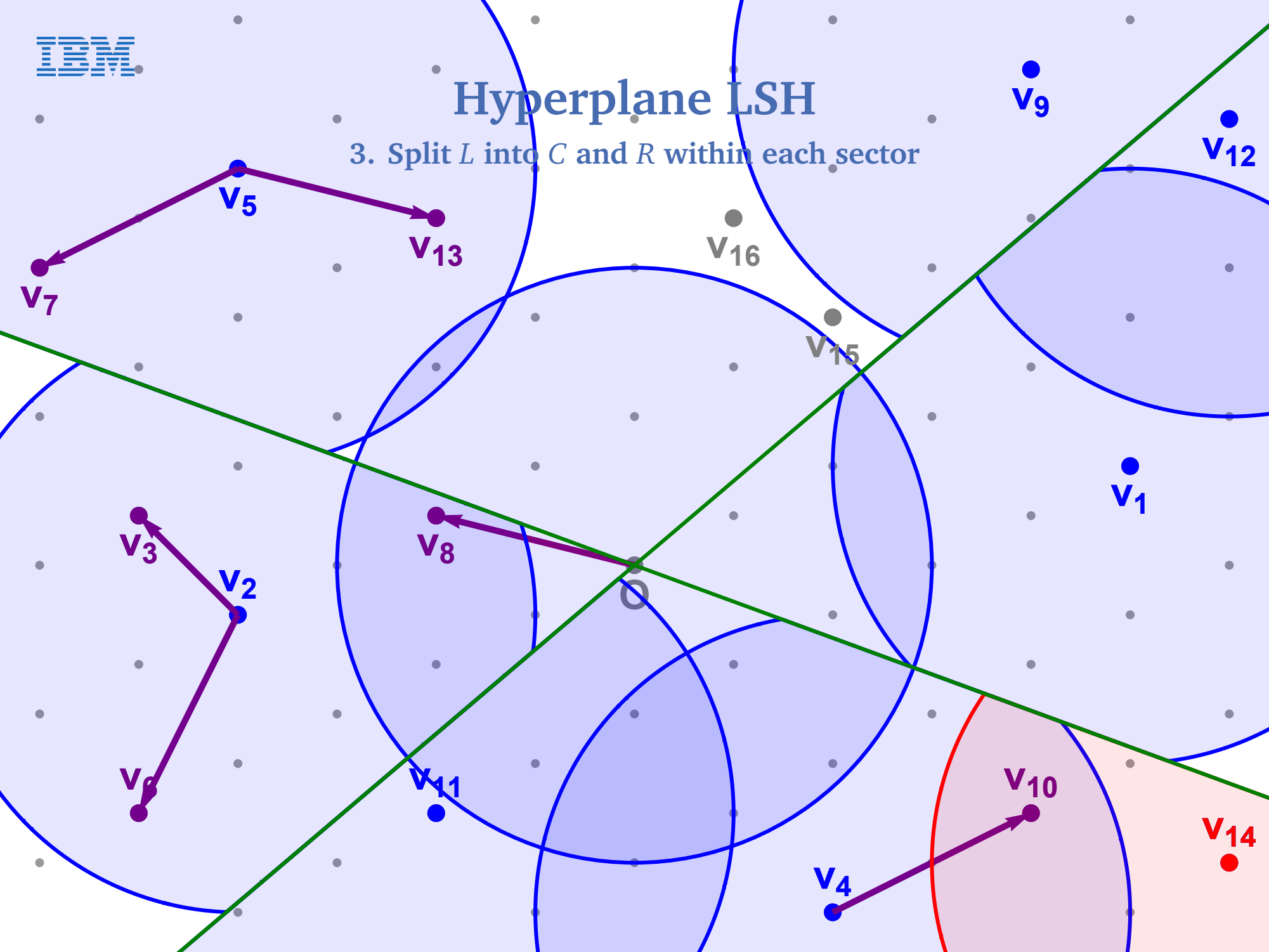
Hyperplane LSH

3. Split L into C and R within each sector



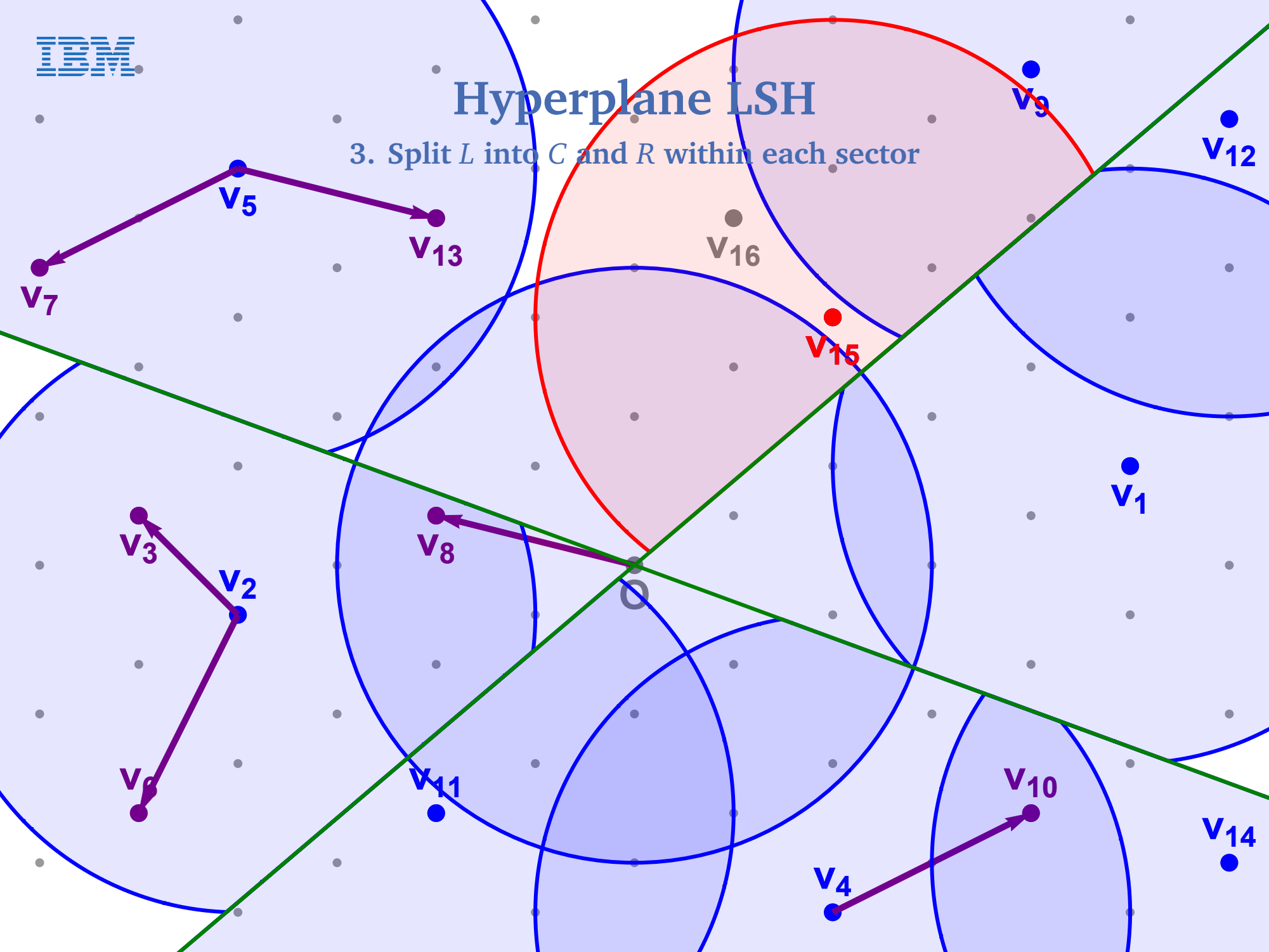
Hyperplane LSH

3. Split L into C and R within each sector



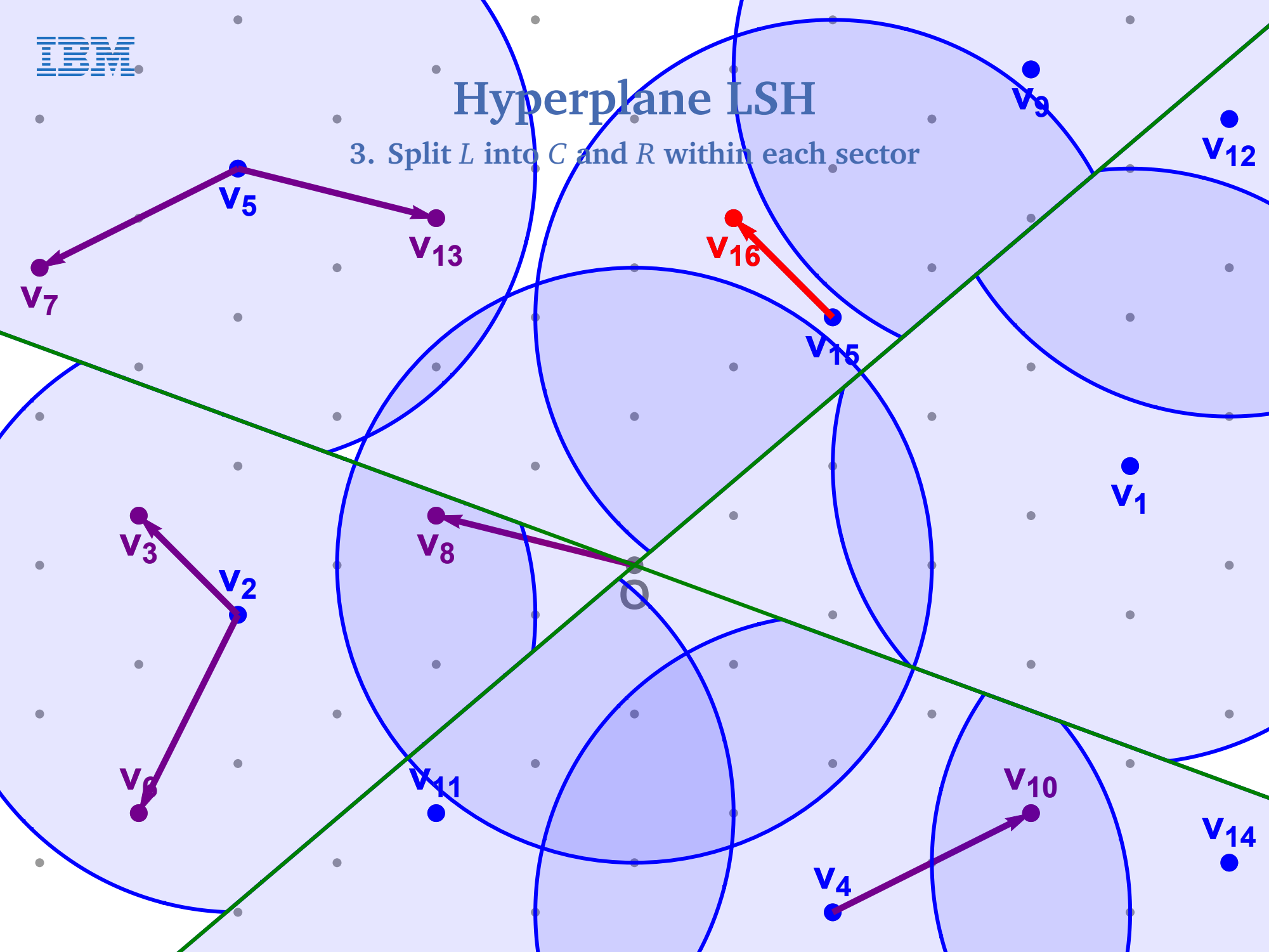
Hyperplane LSH

3. Split L into C and R within each sector



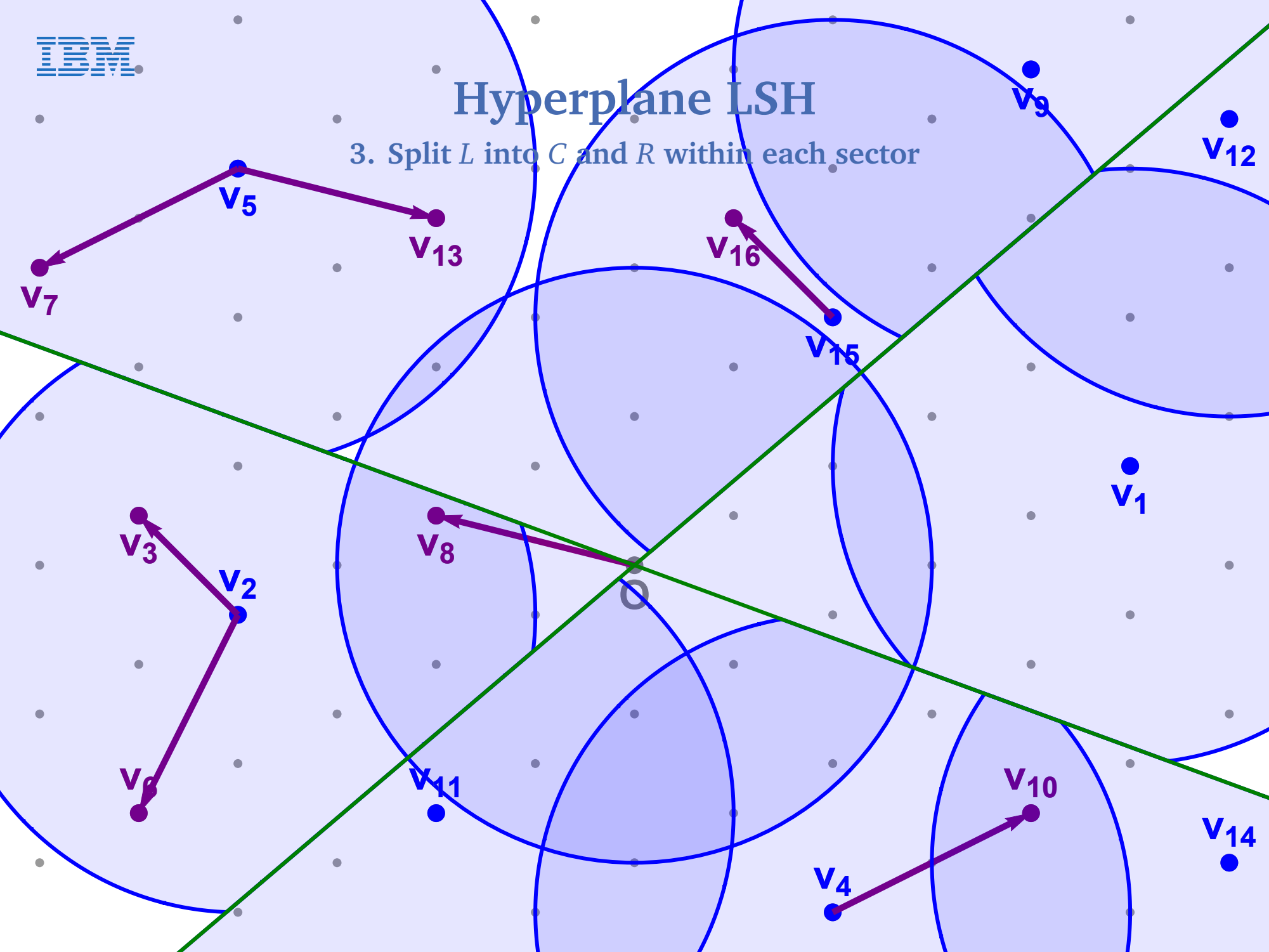
Hyperplane LSH

3. Split L into C and R within each sector



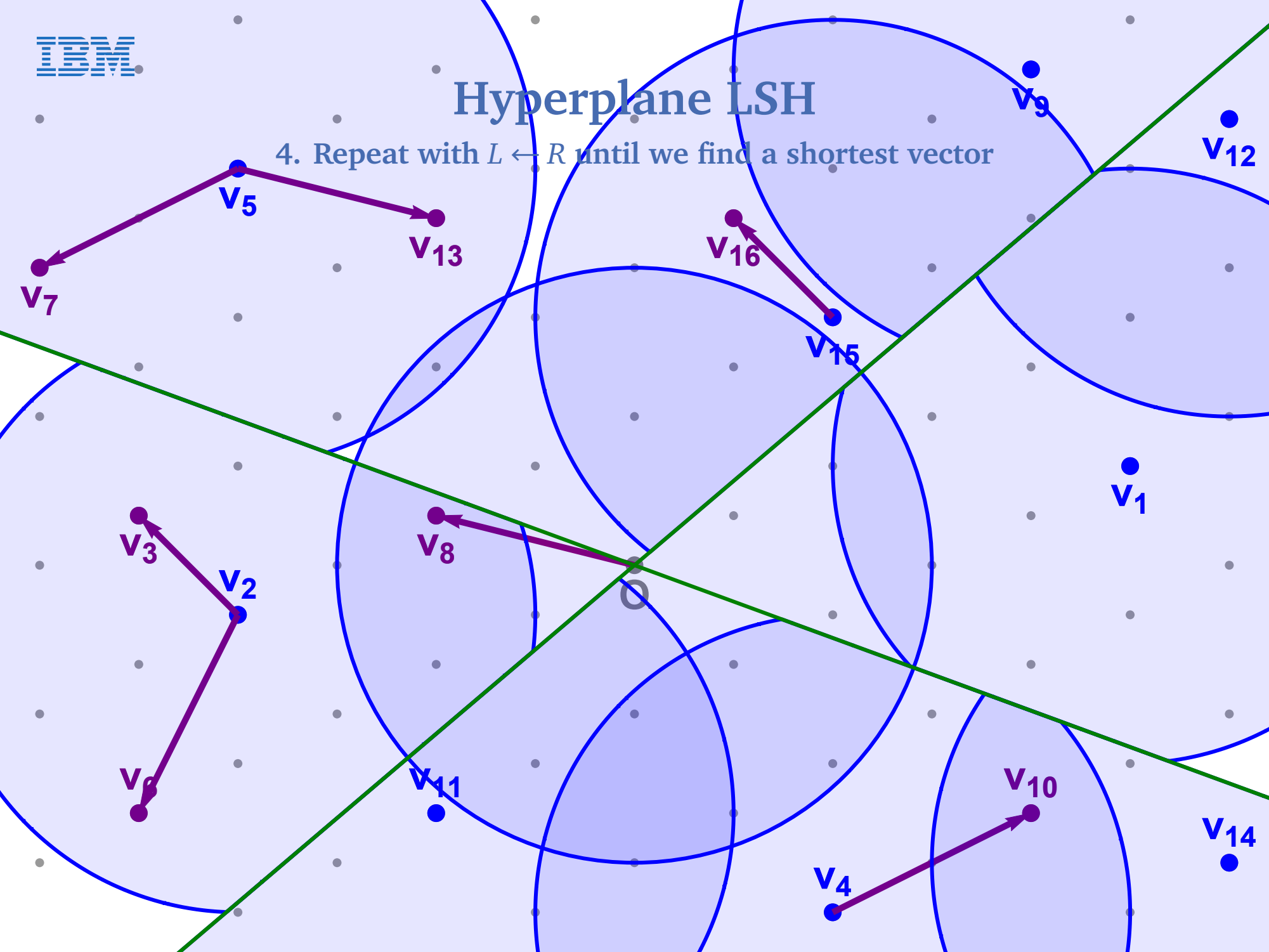
Hyperplane LSH

3. Split L into C and R within each sector



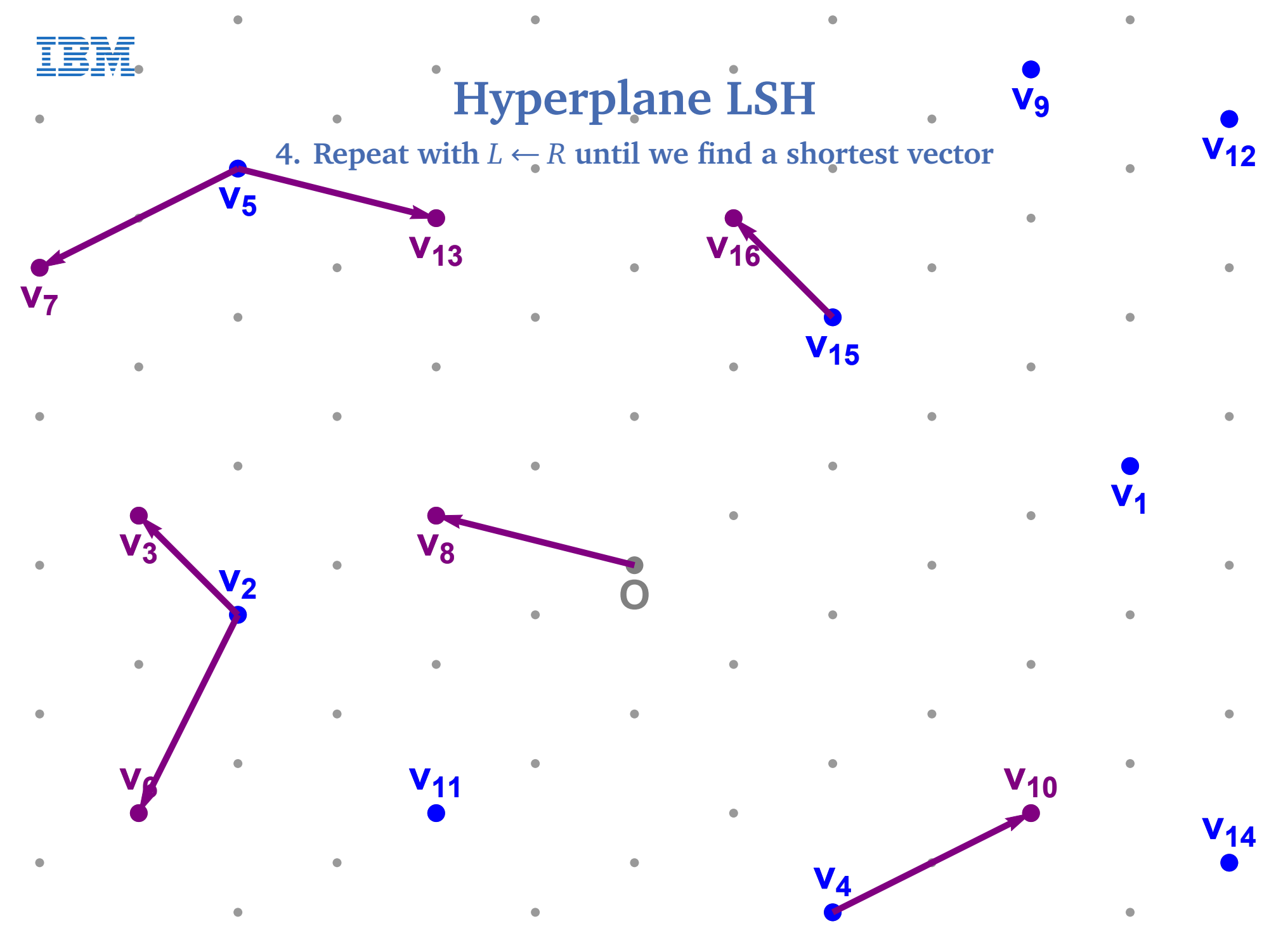
Hyperplane LSH

4. Repeat with $L \leftarrow R$ until we find a shortest vector



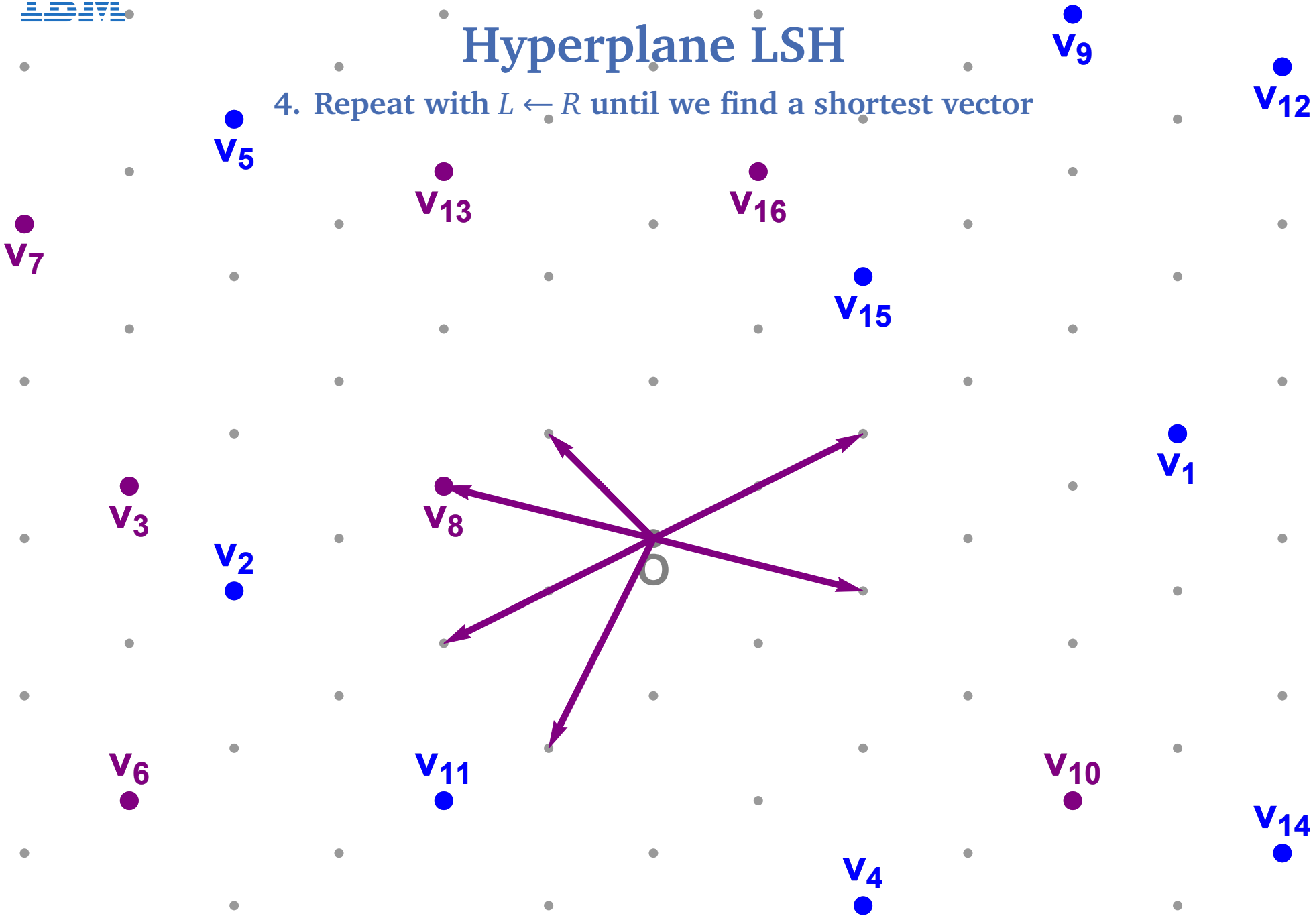
Hyperplane LSH

4. Repeat with $L \leftarrow R$ until we find a shortest vector



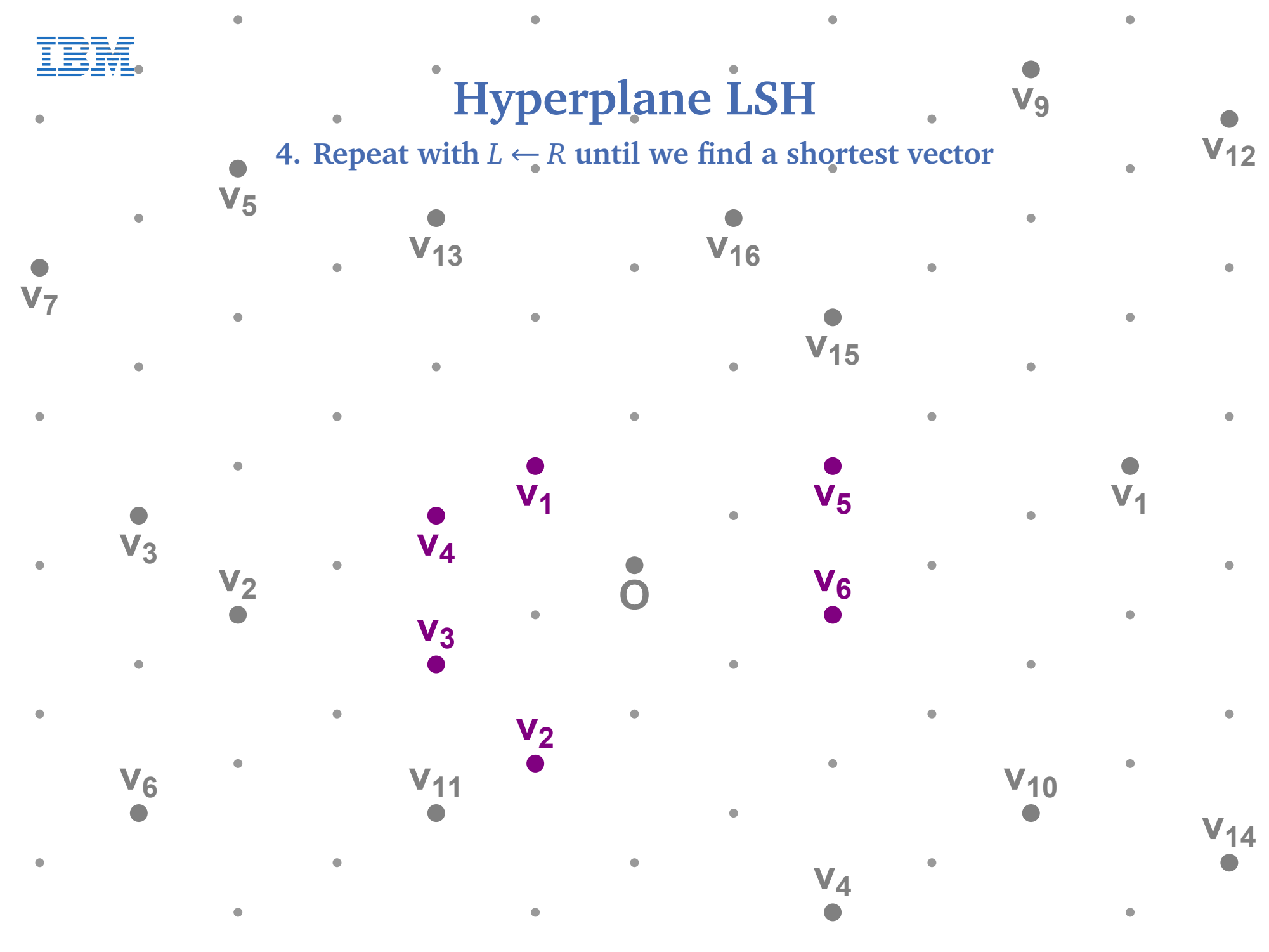
Hyperplane LSH

4. Repeat with $L \leftarrow R$ until we find a shortest vector



Hyperplane LSH

4. Repeat with $L \leftarrow R$ until we find a shortest vector



Exact SVP algorithms

Lattice enumeration

- Memory efficient
- Fincke-Pohst enumeration: $2^{O(n^2)}$ time
- Kannan enumeration: $2^{O(n \log n)}$ time
- Practical speedups with pruning heuristics

Exact SVP algorithms

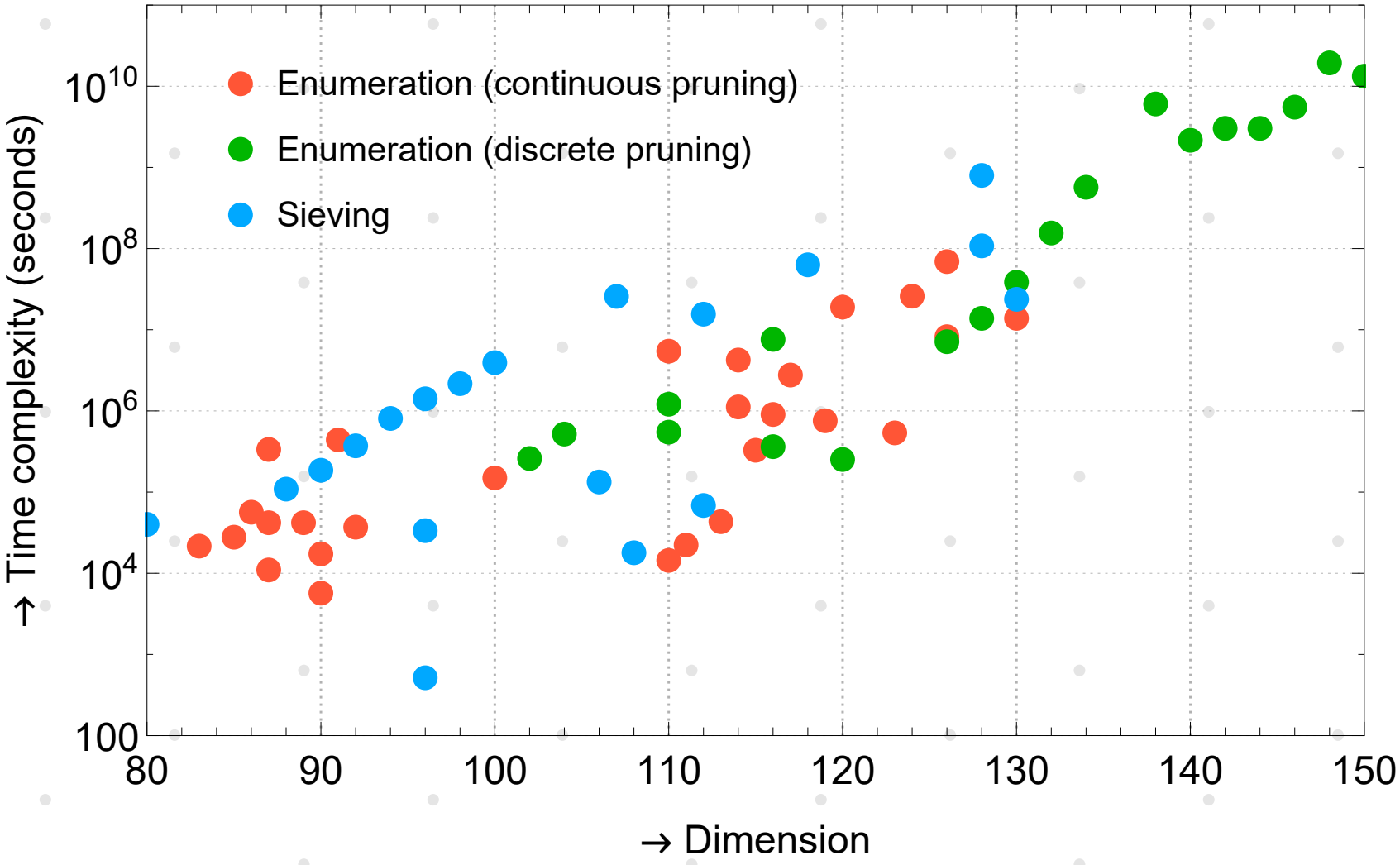
Lattice enumeration

- Memory efficient
- Fincke-Pohst enumeration: $2^{O(n^2)}$ time
- Kannan enumeration: $2^{O(n \log n)}$ time
- Practical speedups with pruning heuristics

Lattice sieving

- Exponential memory footprint
- Asymptotically faster: $2^{O(n)}$ time
- Practical near neighbor speedups

SVP in practice



Summary

- Lattice-based crypto relies on hardness of finding short bases
- State-of-the-art basis reduction: BKZ with fast SVP subroutine
- Competitive SVP methods: enumeration and sieving
- Practice: Enumeration leads, sieving catching up

$$\text{SVP costs} \implies \text{BKZ costs} \implies \text{Security estimates} \implies \text{Parameters}$$

Questions?