IBM Research

Lattice-based cryptography (II)

Thijs Laarhoven

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

PQCrypto Summer School 2017
(June 20, 2017)

IBM Research

Part 2: Lattice algorithms for solving the shortest vector problem

Thijs Laarhoven

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

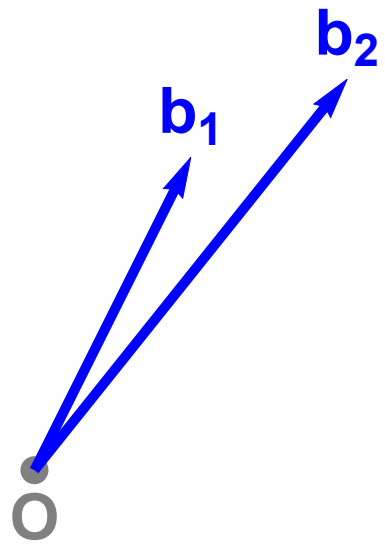
PQCrypto Summer School 2017
(June 20, 2017)

Lattices

Lattices

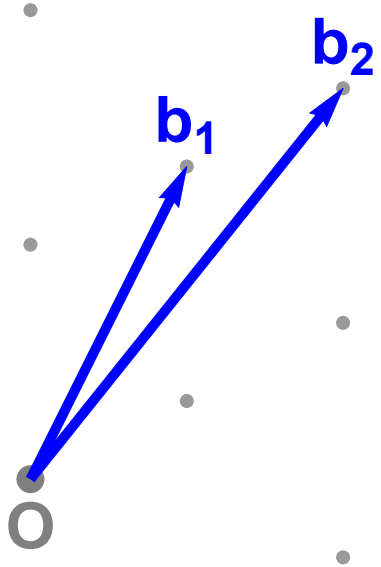
Lattices

Lattices



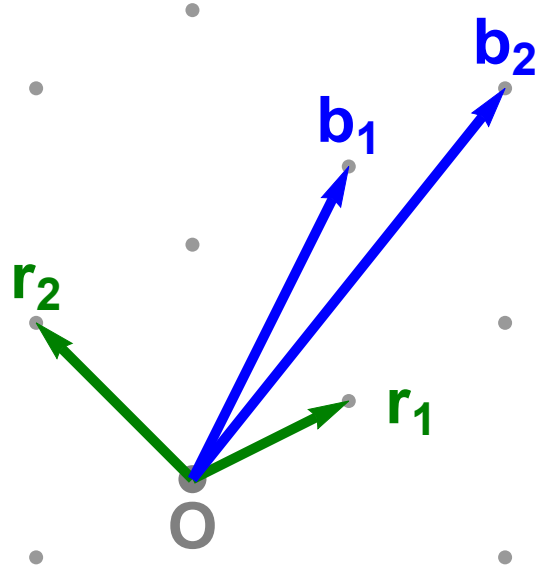
Lattices

Lattices



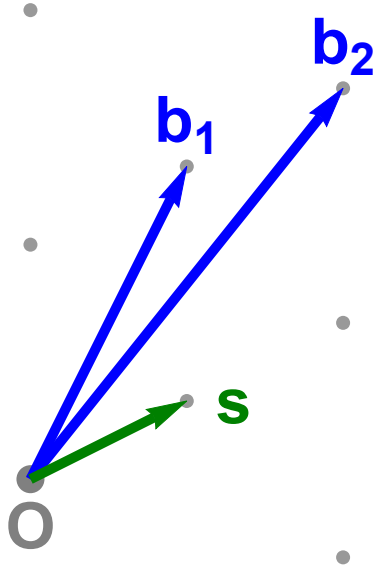
Lattices

Lattice basis reduction



Lattices

Shortest Vector Problem (SVP)



Outline

Enumeration algorithms

- Fincke-Pohst enumeration
- Kannan enumeration
- Pruning the enumeration tree

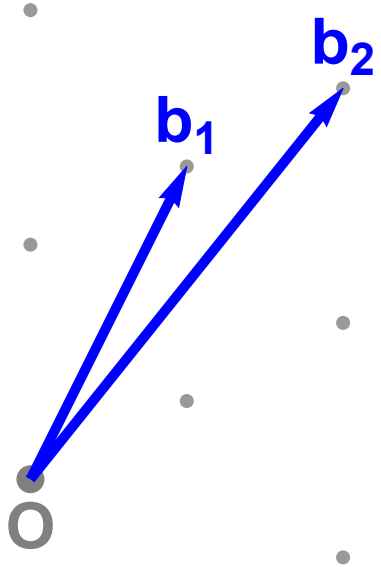
Sieving algorithms

- Nguyen-Vidick sieve
- Multiple levels
- Near neighbor techniques

Practical comparison

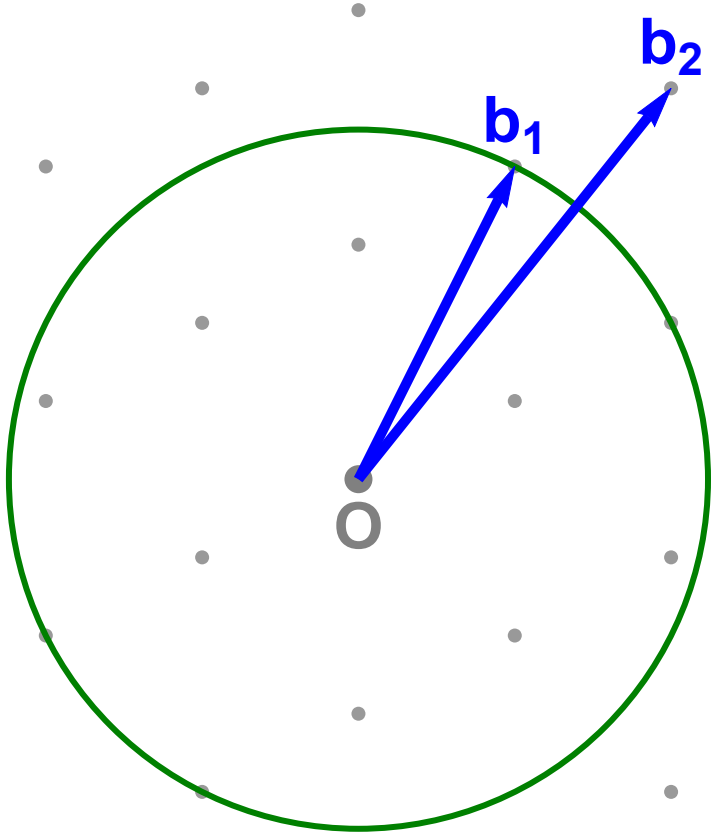
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



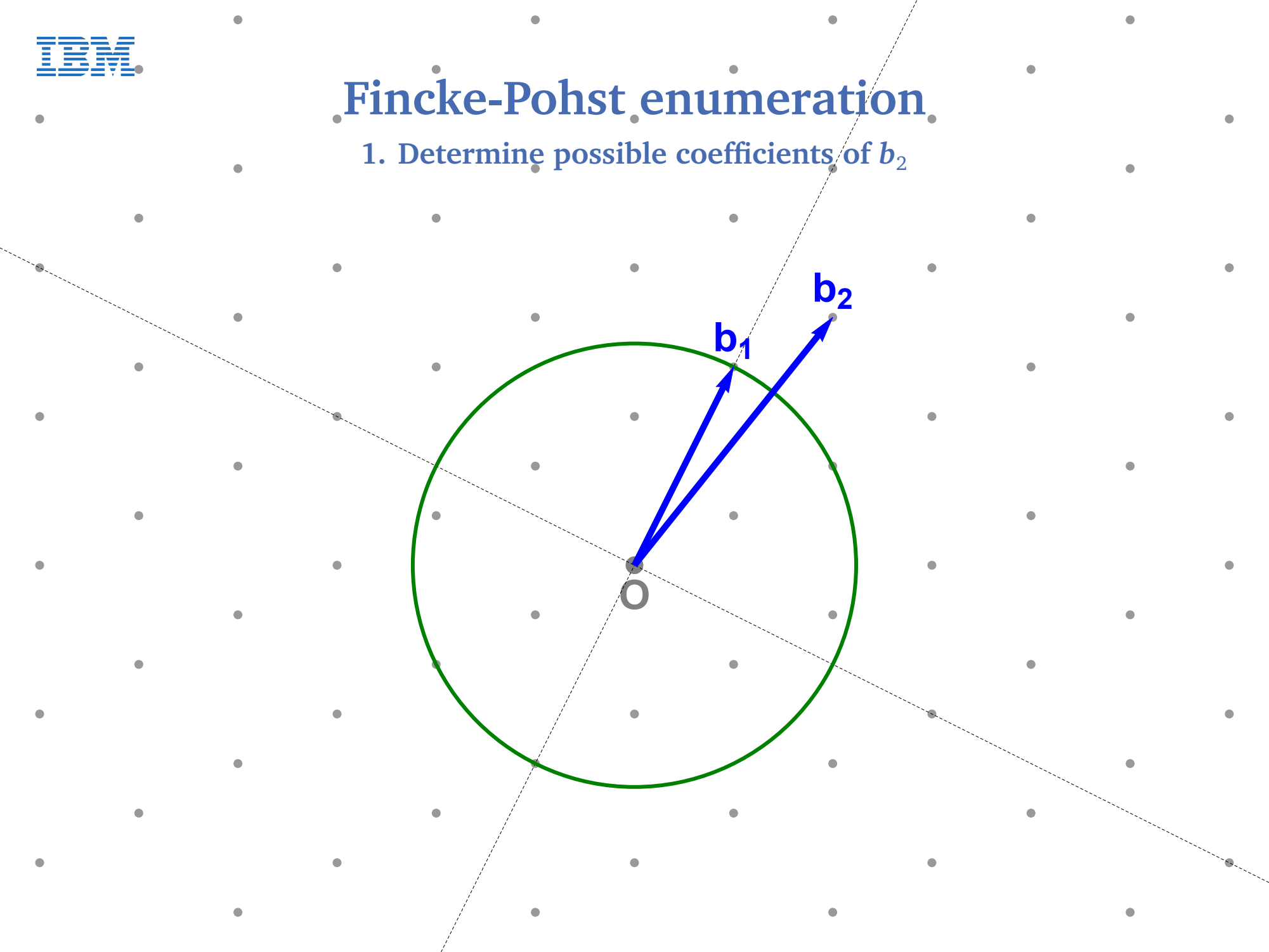
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



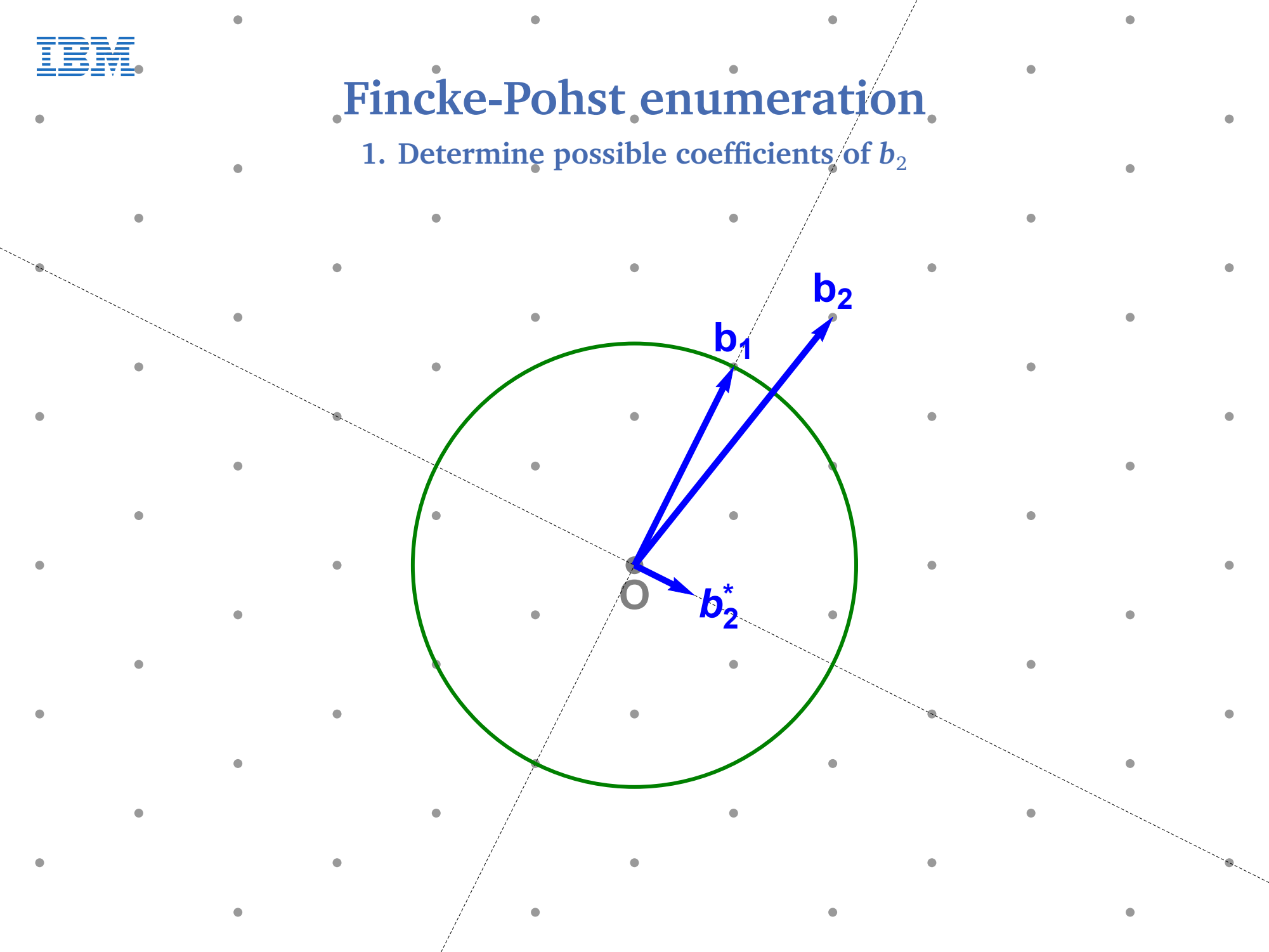
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



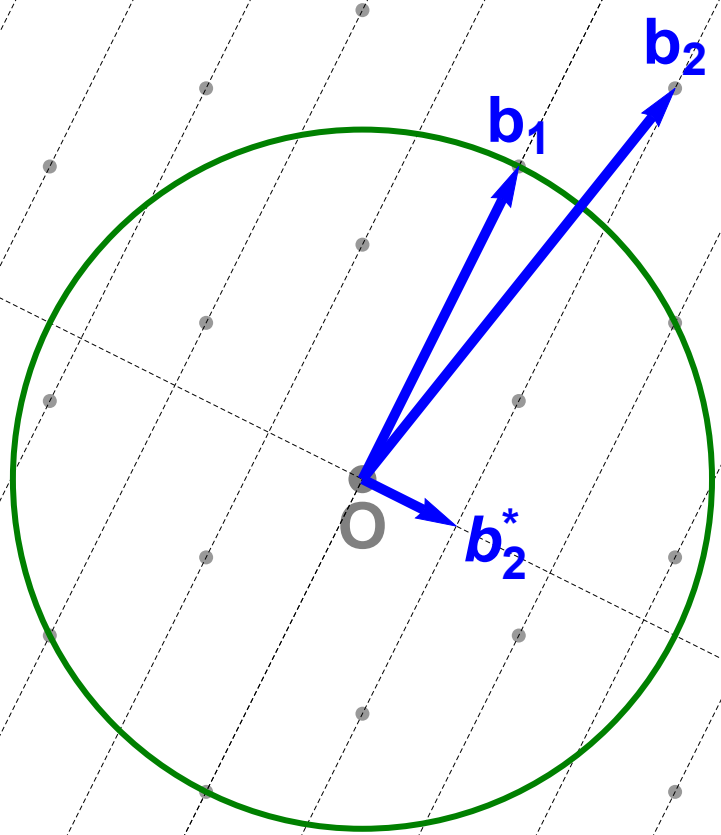
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



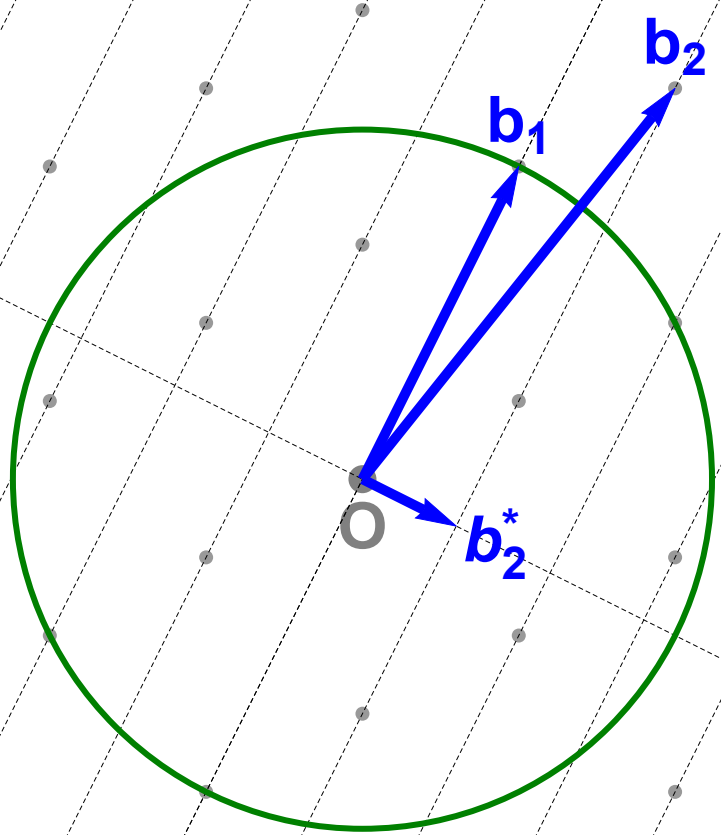
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



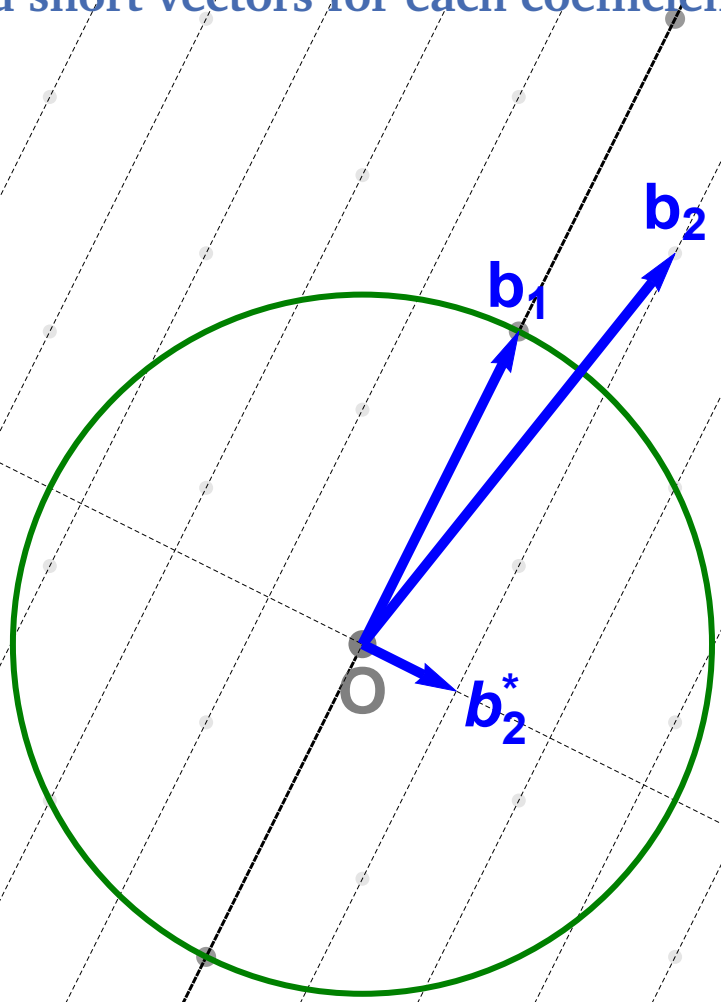
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



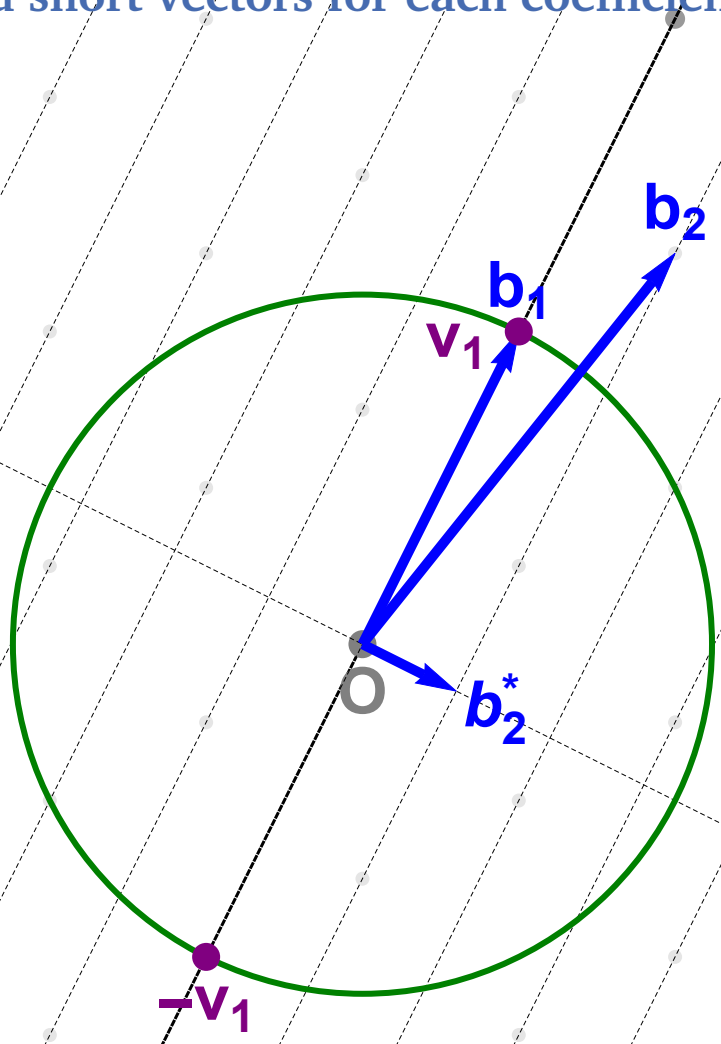
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



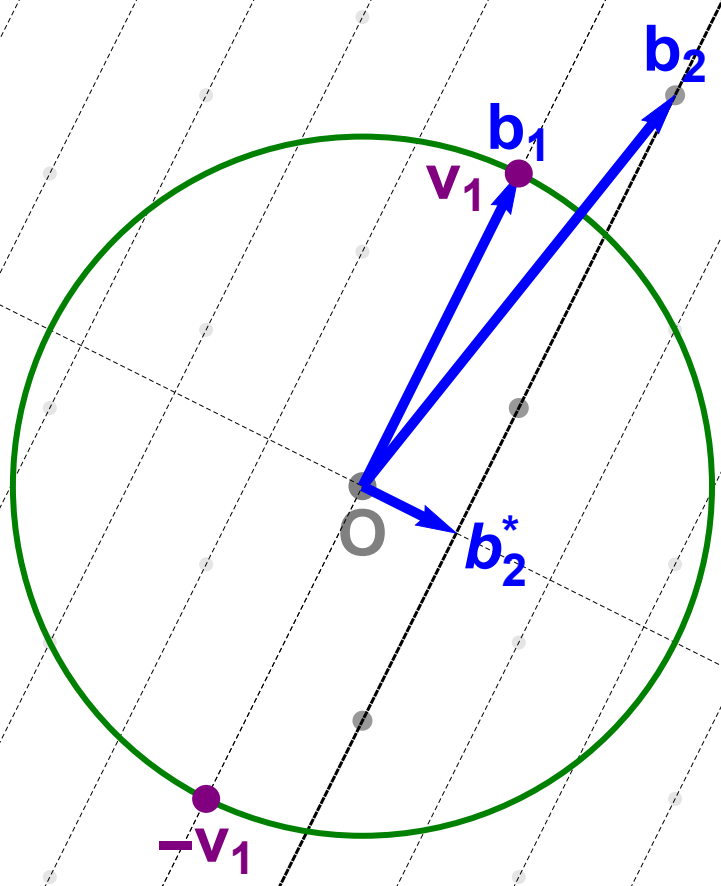
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



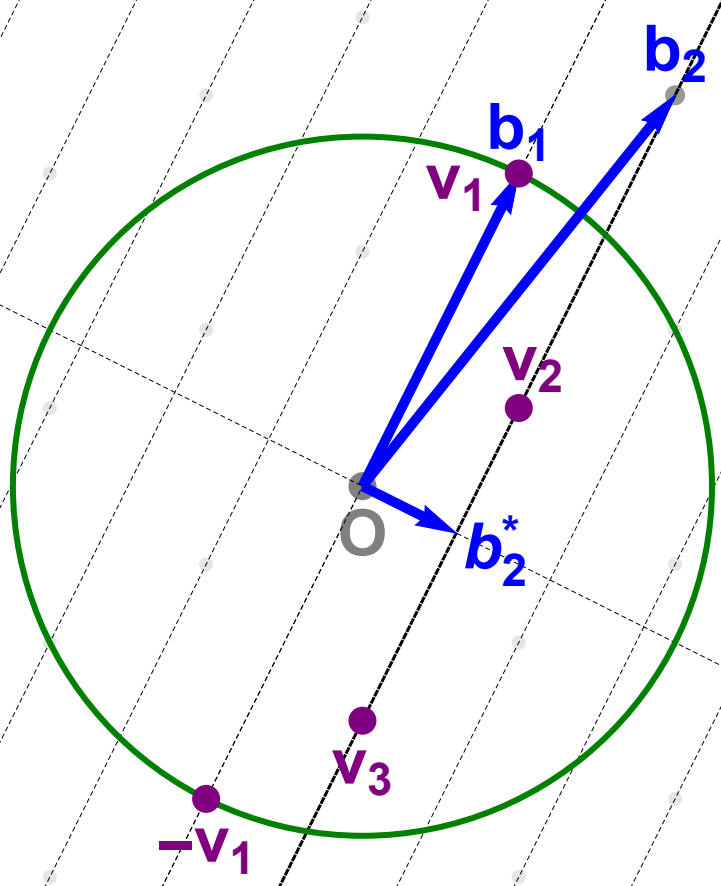
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



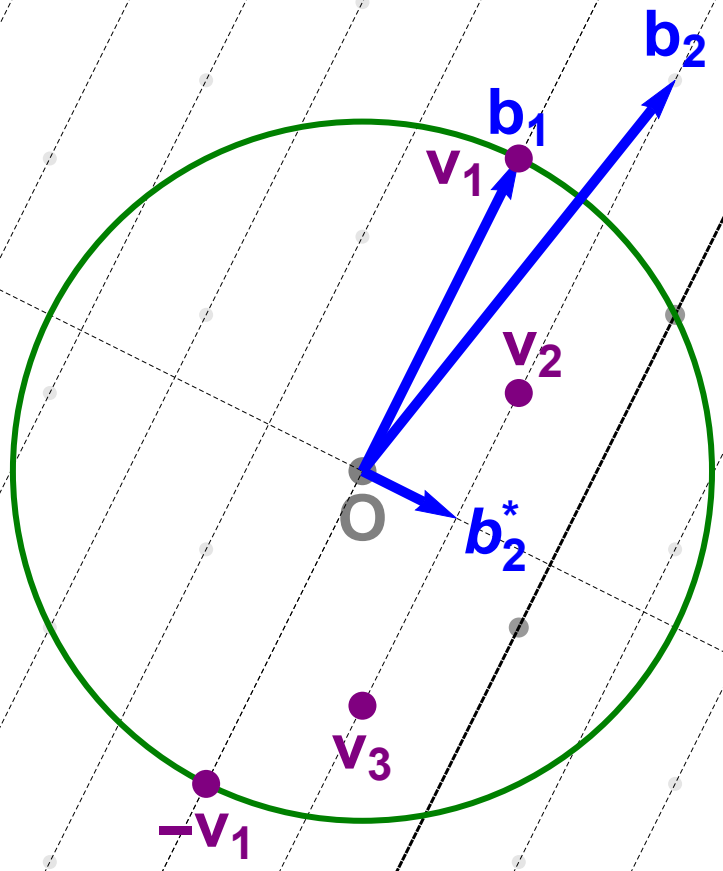
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



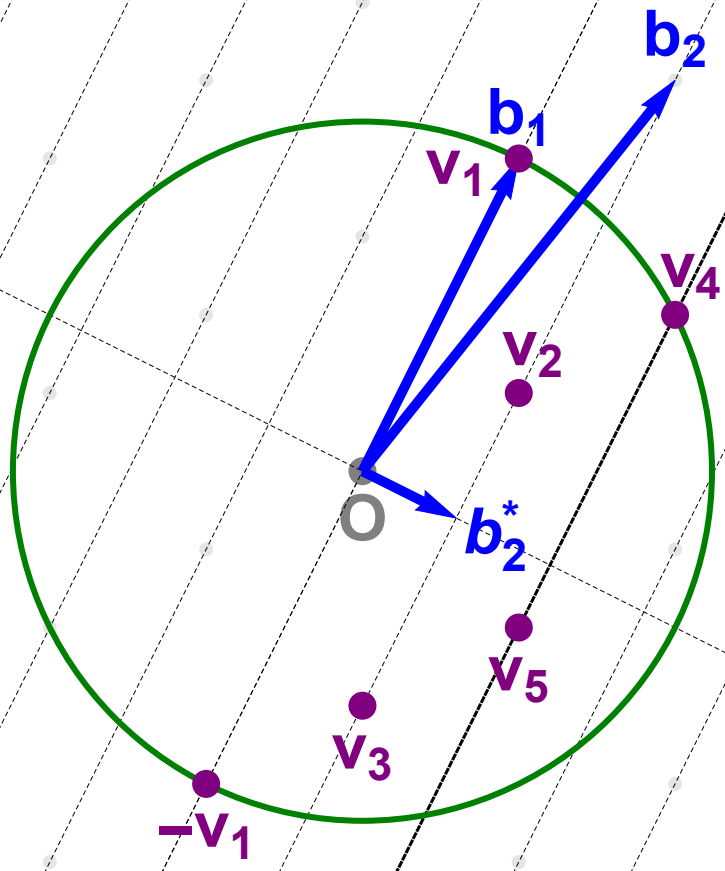
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



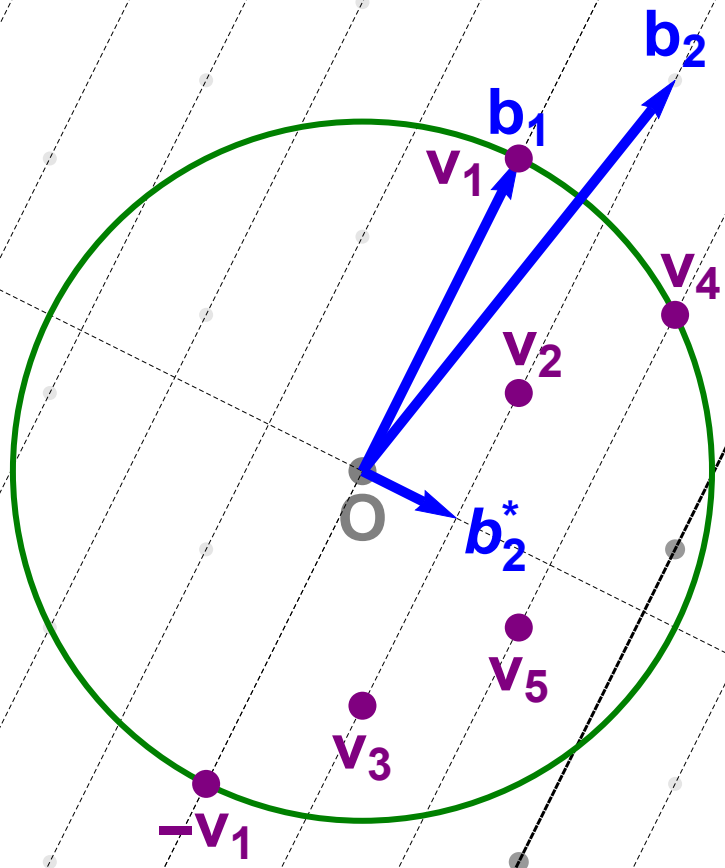
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



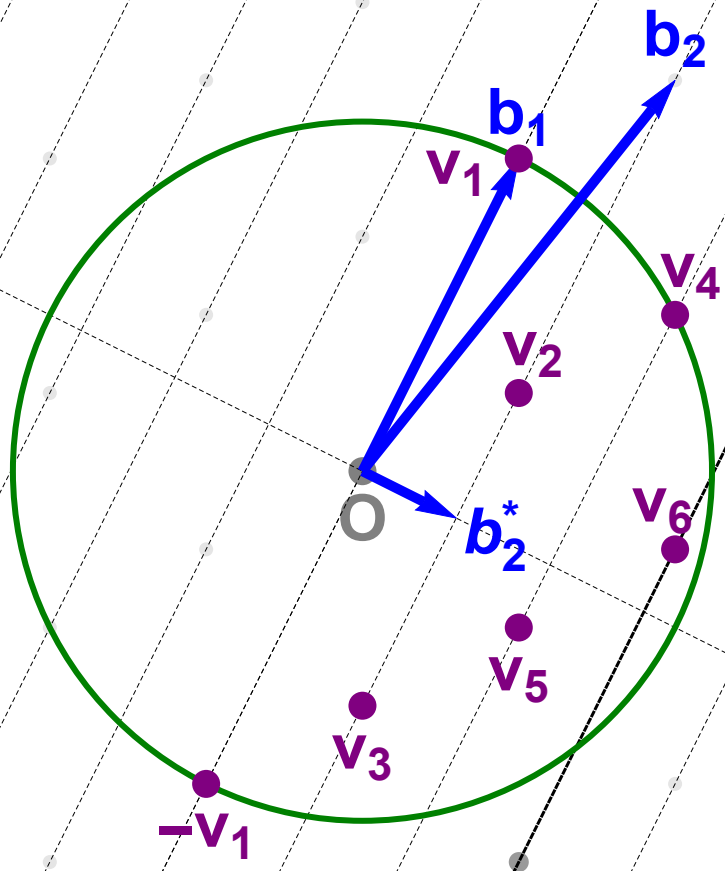
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



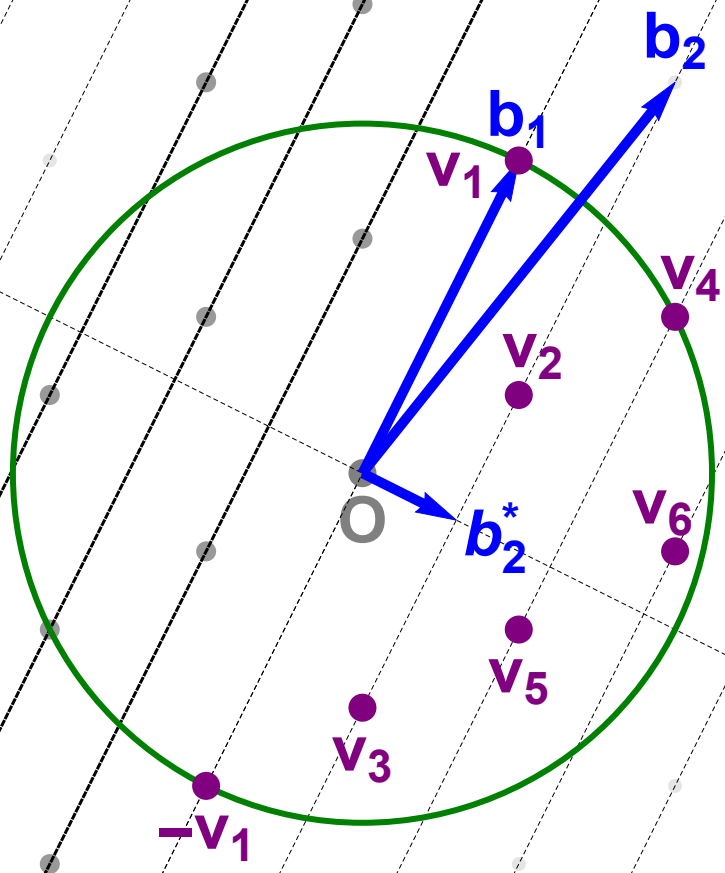
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



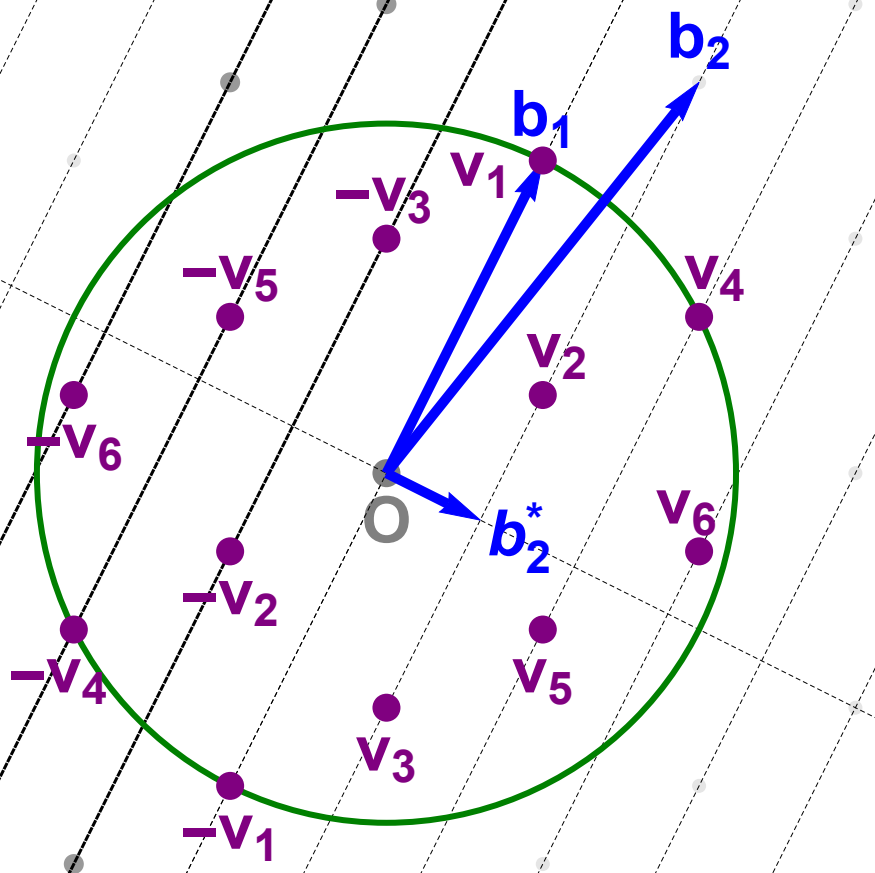
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



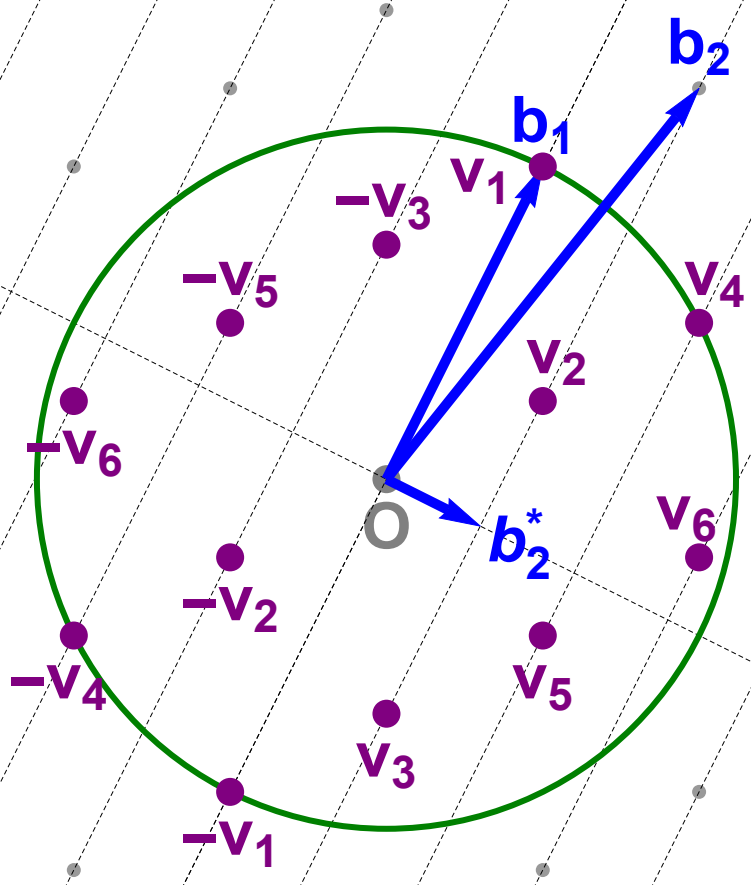
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



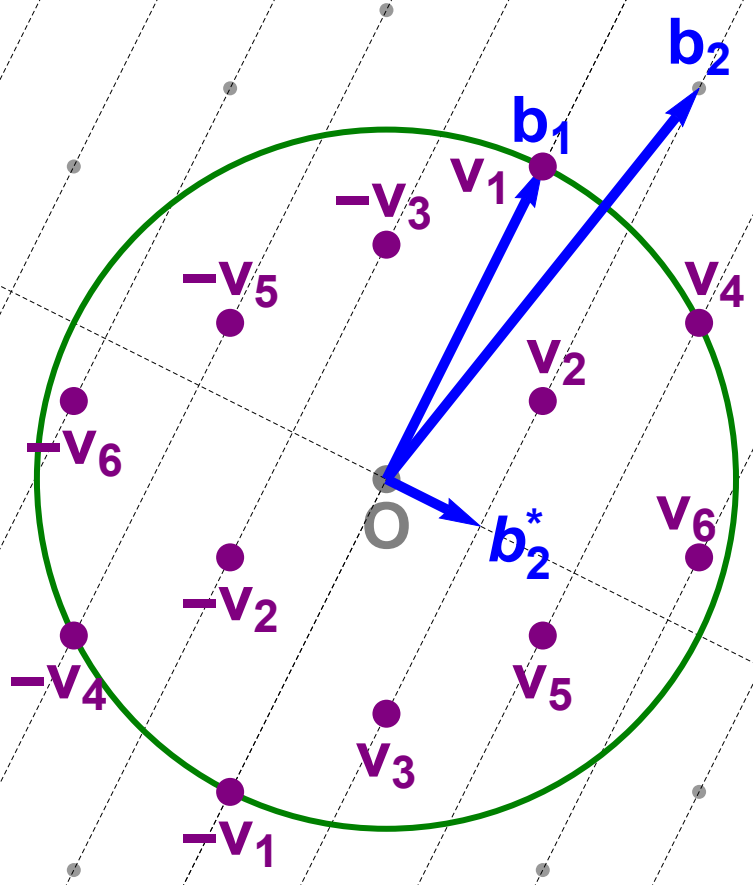
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



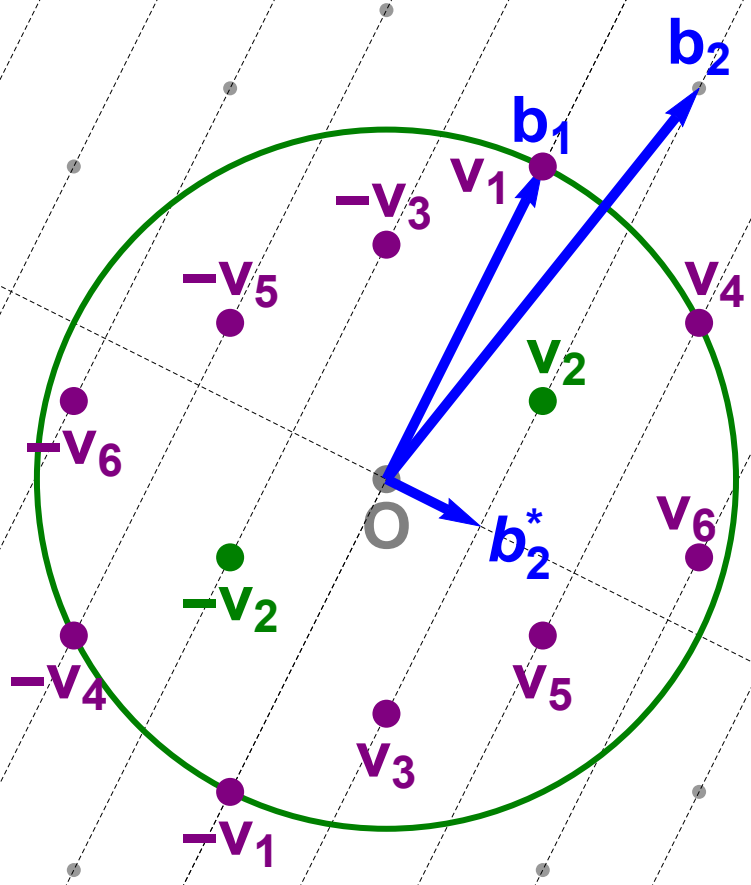
Fincke-Pohst enumeration

3. Find a shortest vector among all found vectors



Fincke-Pohst enumeration

3. Find a shortest vector among all found vectors

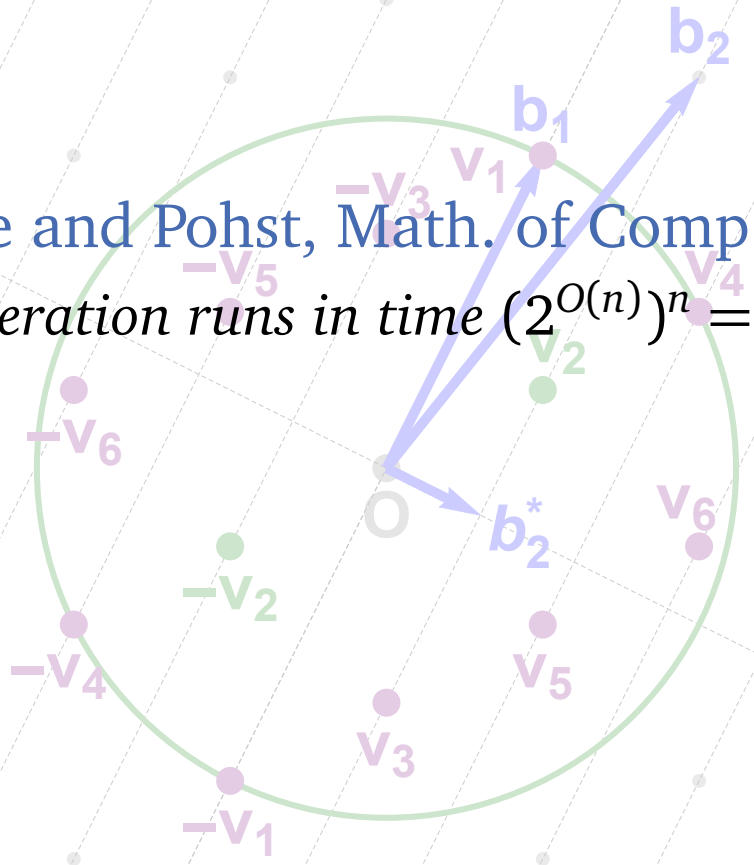


Fincke-Pohst enumeration

Overview

Fincke-Pohst enumeration

Overview

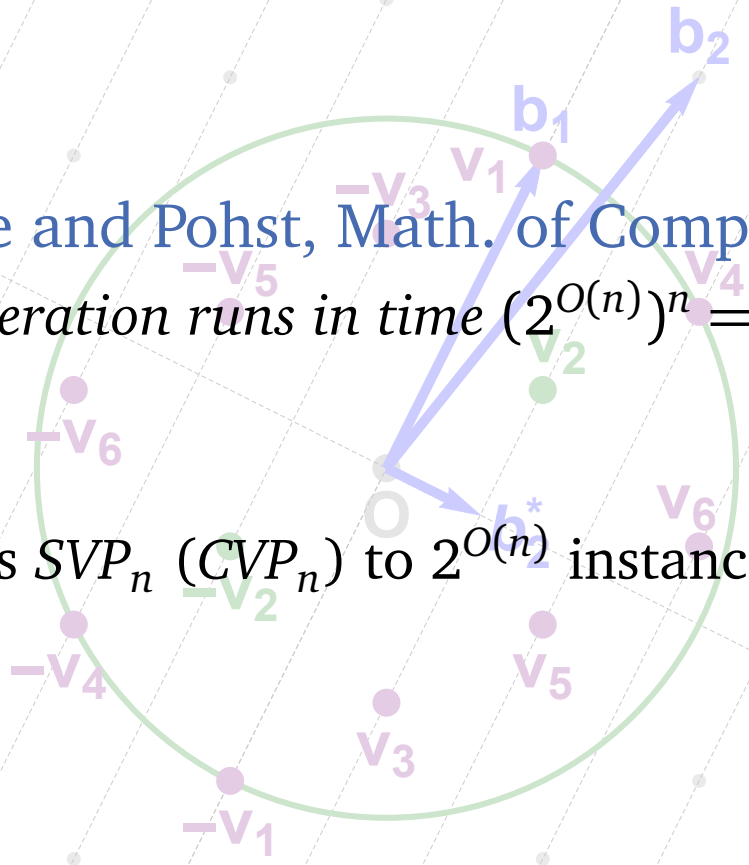


Theorem (Fincke and Pohst, Math. of Comp. '85)

Fincke-Pohst enumeration runs in time $(2^{O(n)})^n = 2^{O(n^2)}$ *and space* poly(n).

Fincke-Pohst enumeration

Overview



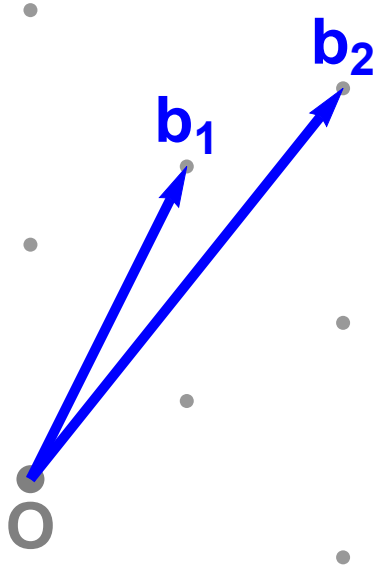
Theorem (Fincke and Pohst, Math. of Comp. '85)

Fincke-Pohst enumeration runs in time $(2^{O(n)})^n = 2^{O(n^2)}$ *and space* poly(n).

Essentially reduces SVP_n (CVP_n) to $2^{O(n)}$ instances of CVP_{n-1}

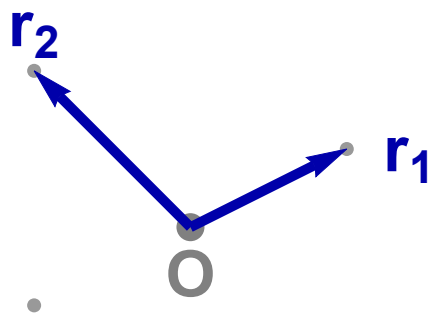
Kannan enumeration

Better bases



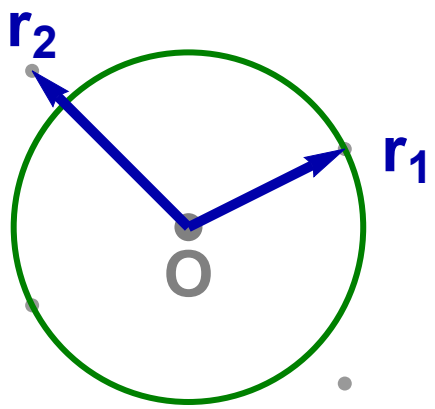
Kannan enumeration

Better bases



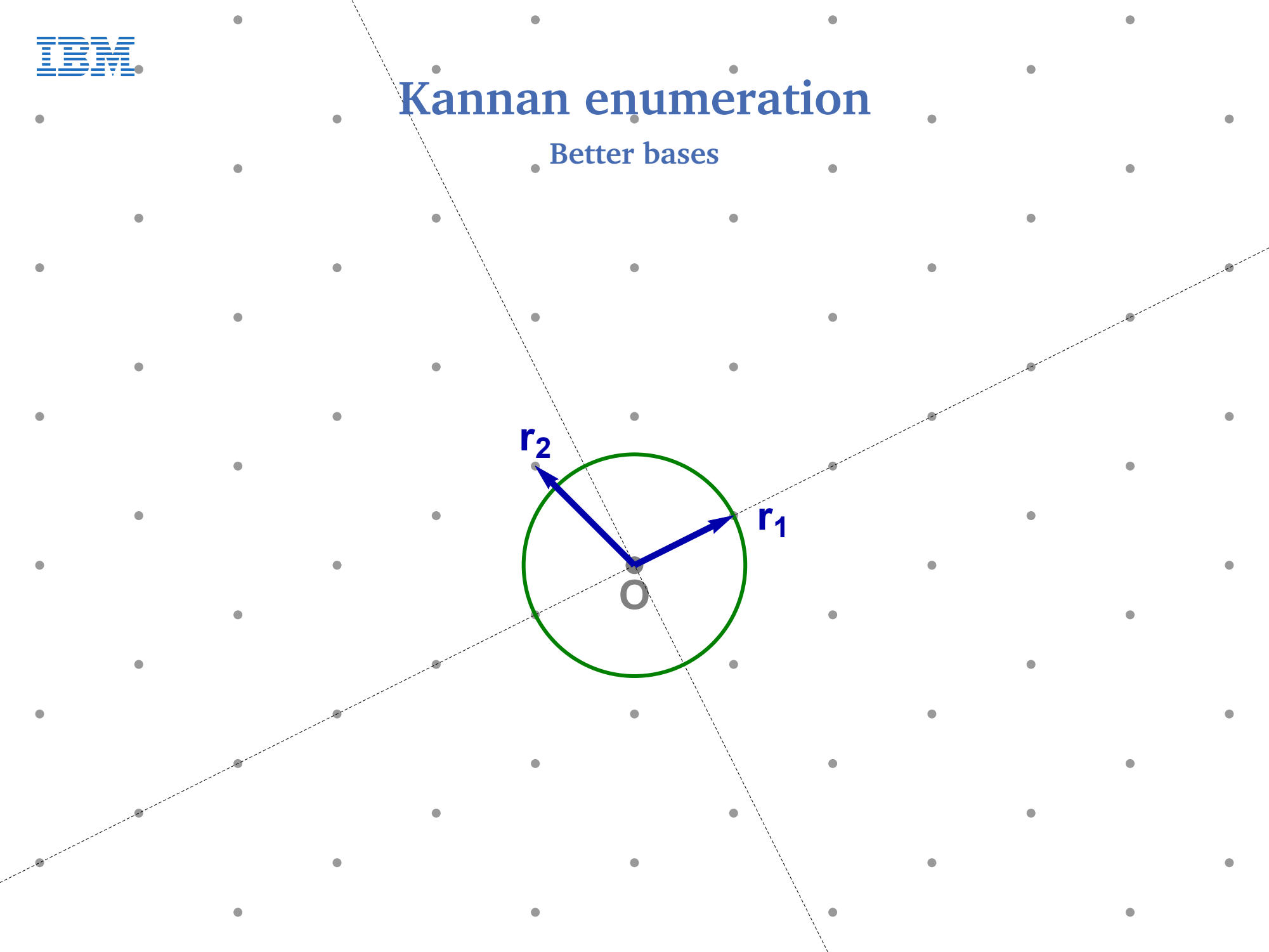
Kannan enumeration

Better bases



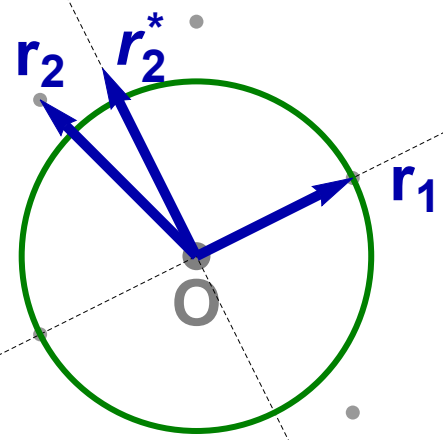
Kannan enumeration

Better bases



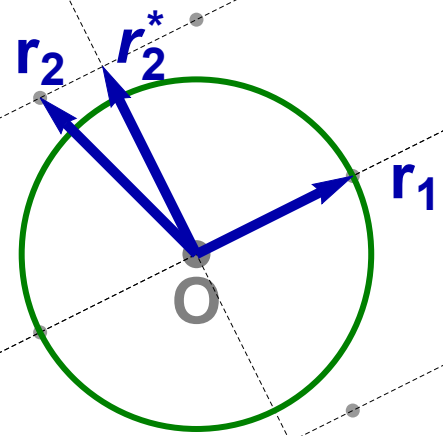
Kannan enumeration

Better bases



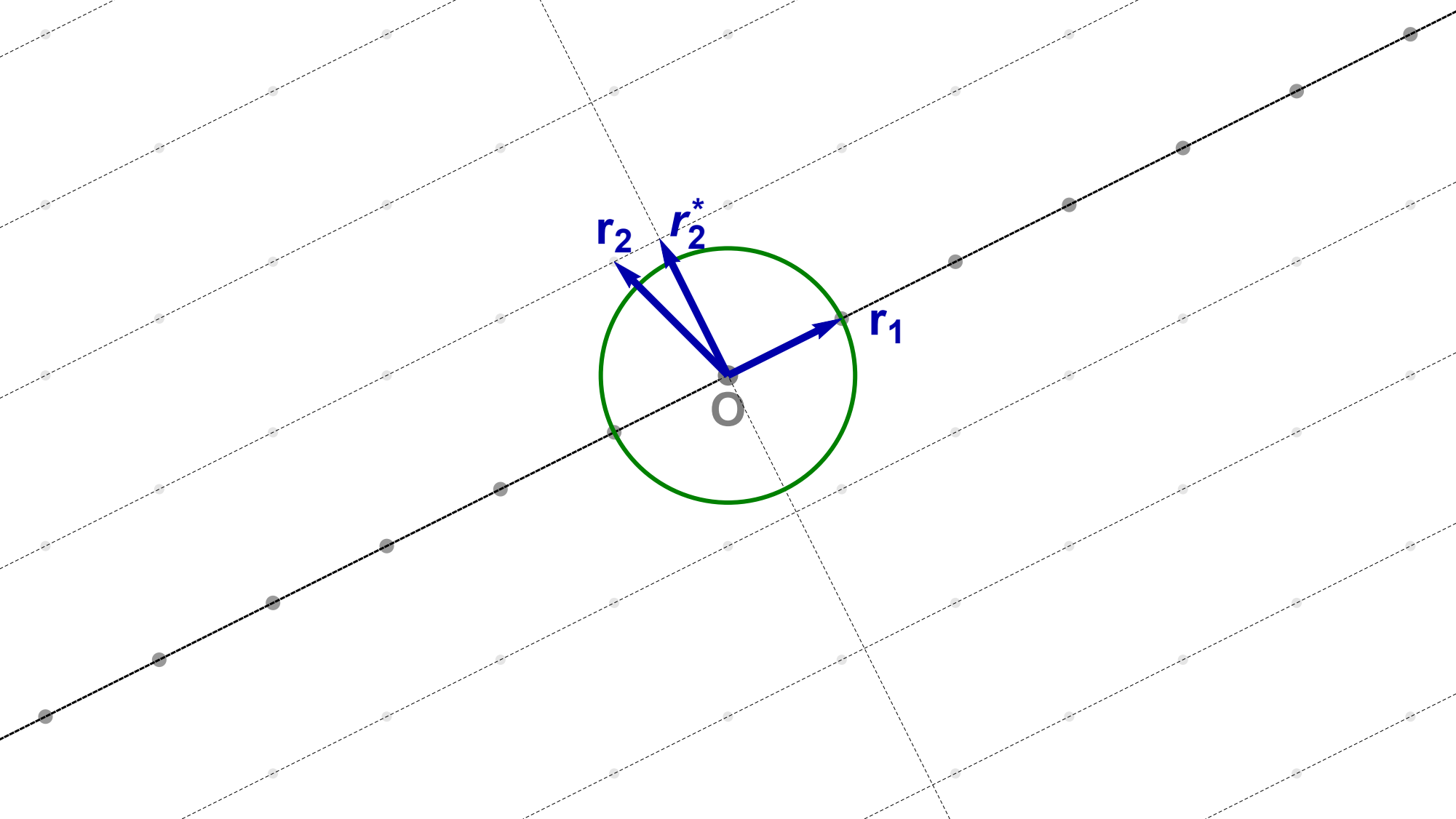
Kannan enumeration

Better bases



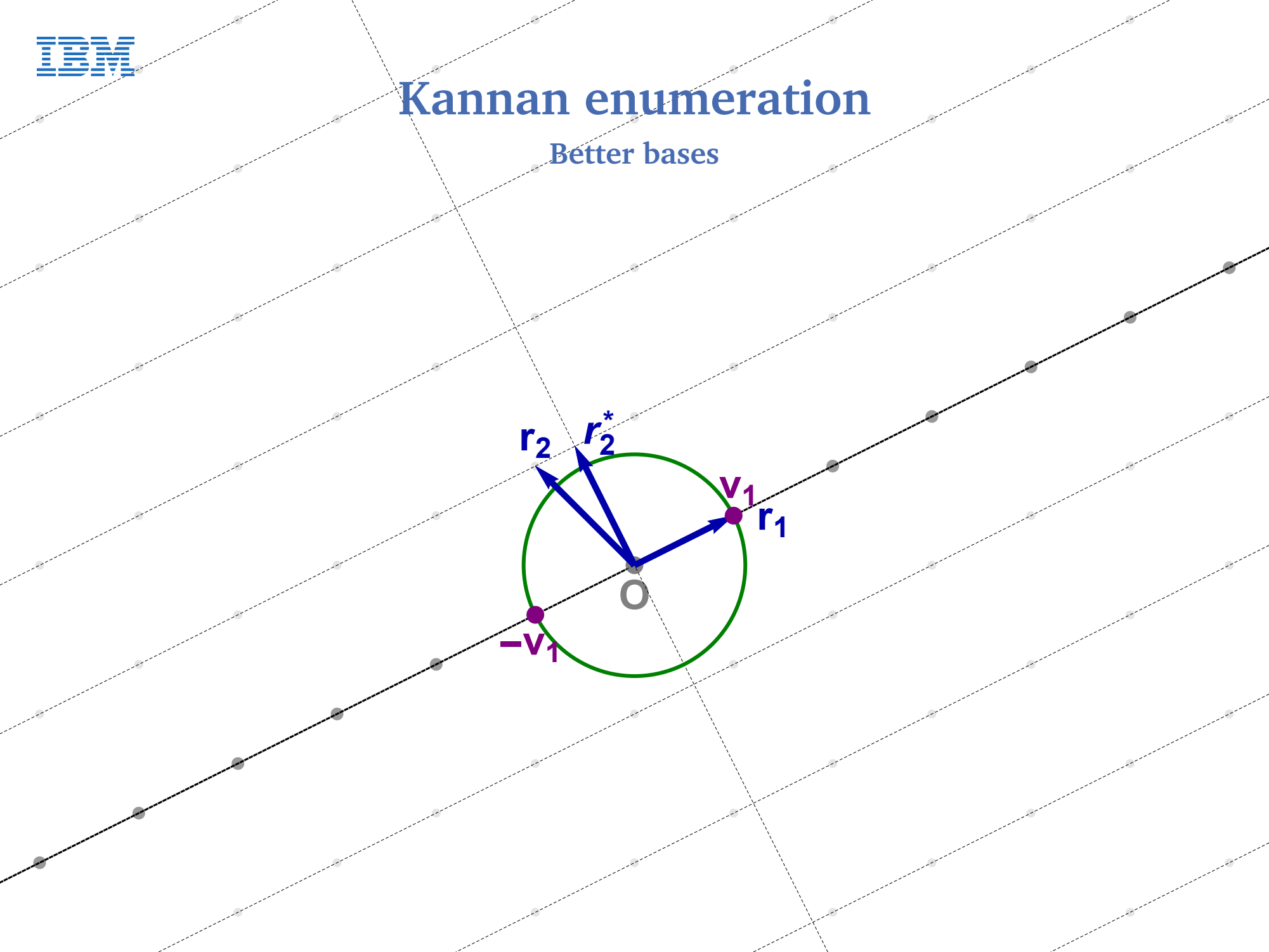
Kannan enumeration

Better bases



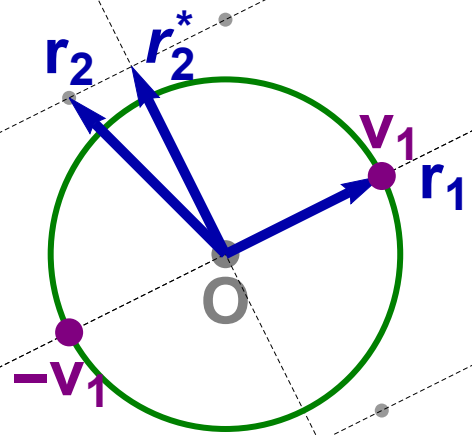
Kannan enumeration

Better bases



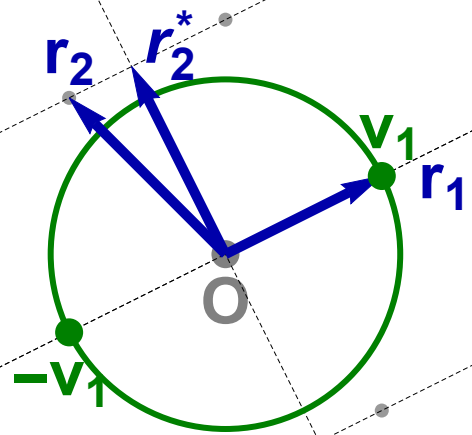
Kannan enumeration

Better bases



Kannan enumeration

Better bases



Kannan enumeration

Overview

Kannan enumeration

Overview

Theorem (Kannan, STOC'83)

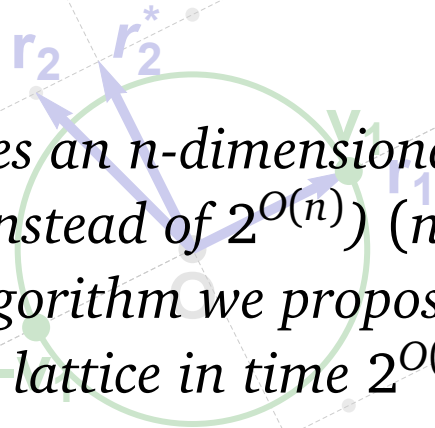
Kannan enumeration runs in time $2^{O(n \log n)}$ *and space* $\mathrm{poly}(n)$.

Kannan enumeration

Overview

Theorem (Kannan, STOC'83)

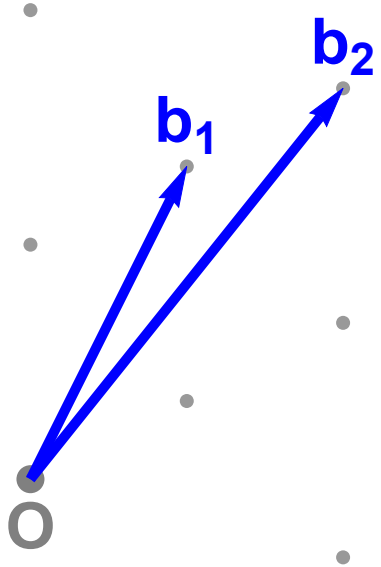
Kannan enumeration runs in time $2^{O(n \log n)}$ *and space* $\mathrm{poly}(n)$.



> *"Our algorithm reduces an n-dimensional problem to polynomially many (instead of* $2^{O(n)}$*)* $(n-1)$*-dimensional problems. [...] The algorithm we propose, first finds a more orthogonal basis for a lattice in time* $2^{O(n \log n)}$*."*
>
> – Kannan, *STOC'83*

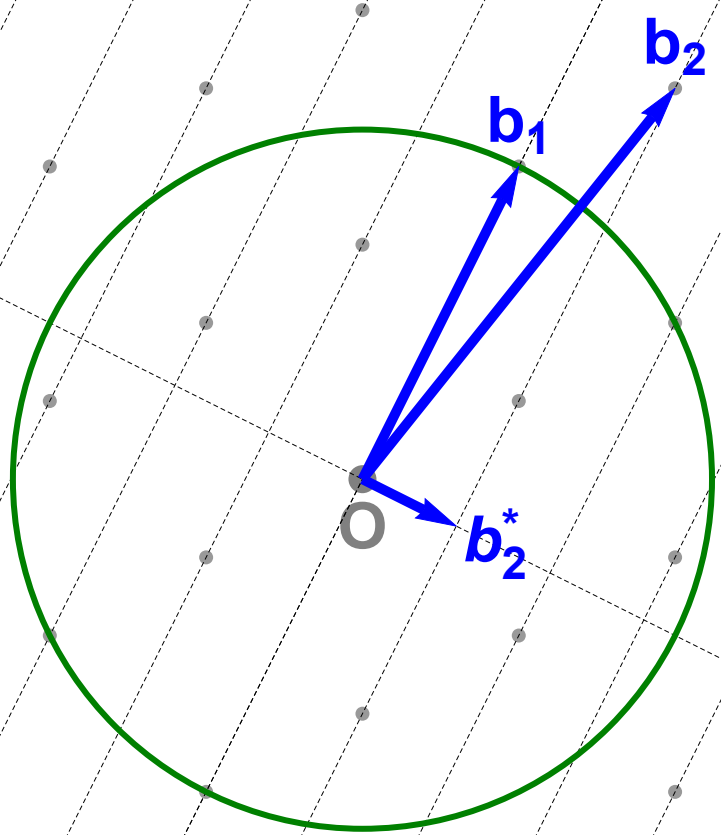
Pruned enumeration

Reducing the search space



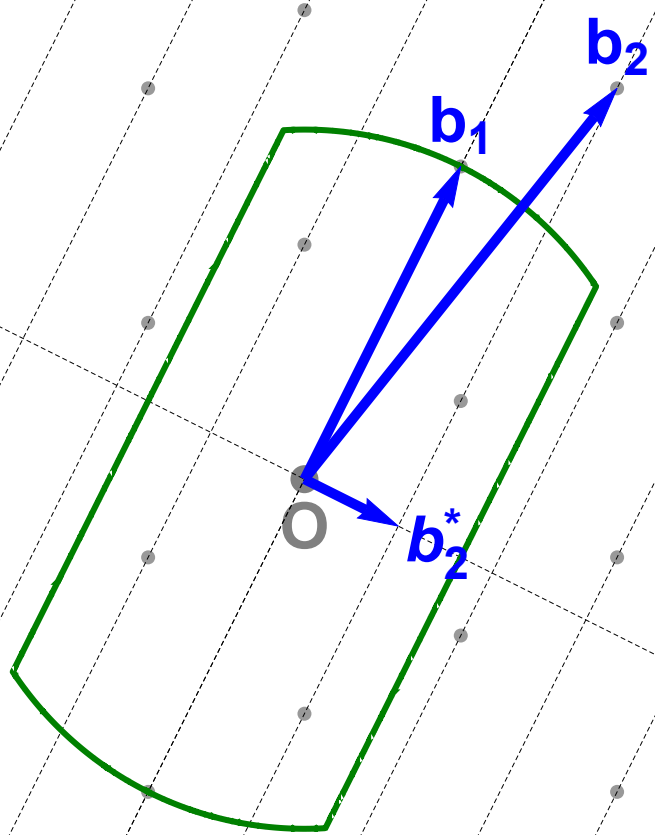
Pruned enumeration

Reducing the search space



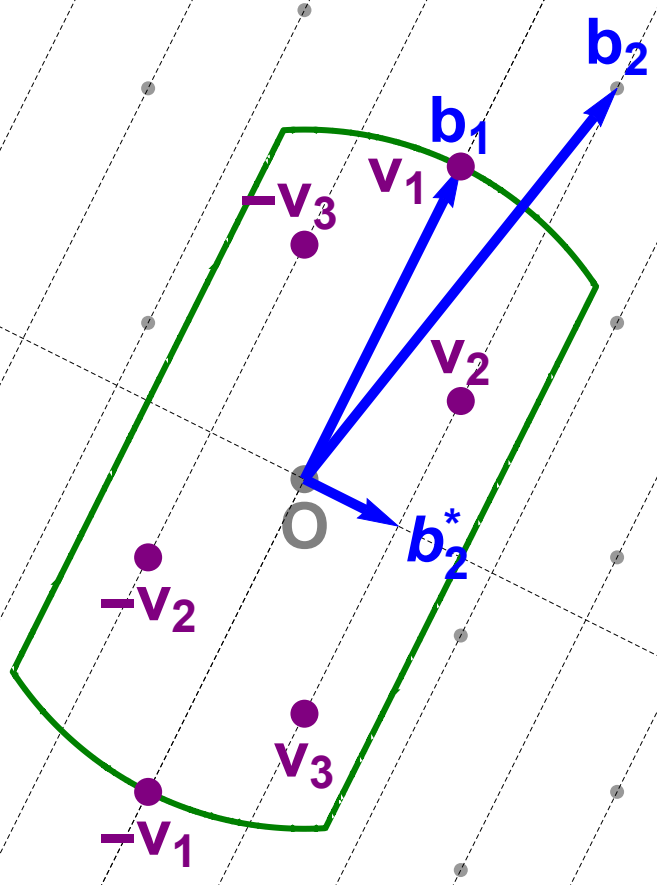
Pruned enumeration

Reducing the search space



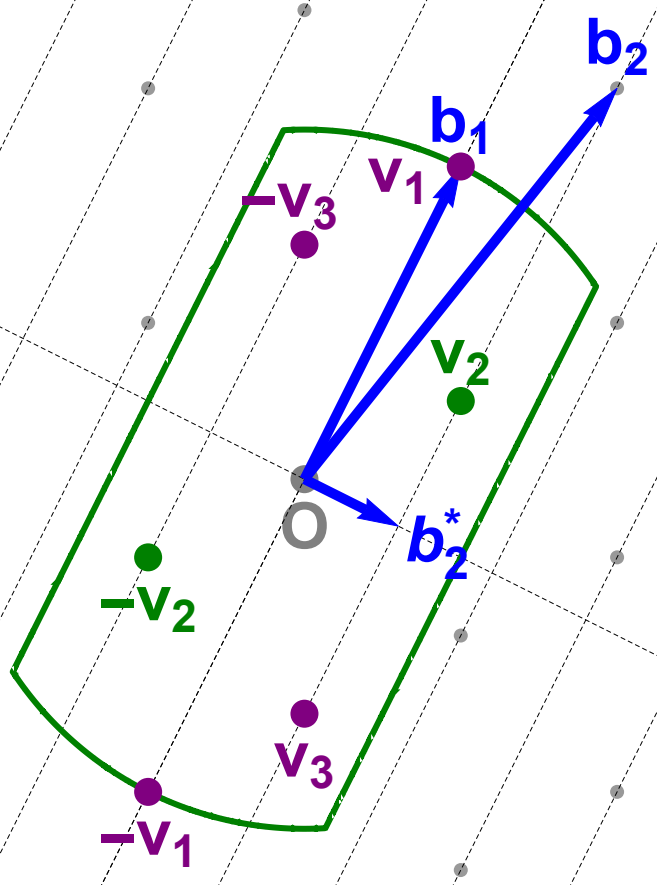
Pruned enumeration

Reducing the search space



Pruned enumeration

Reducing the search space



Pruned enumeration

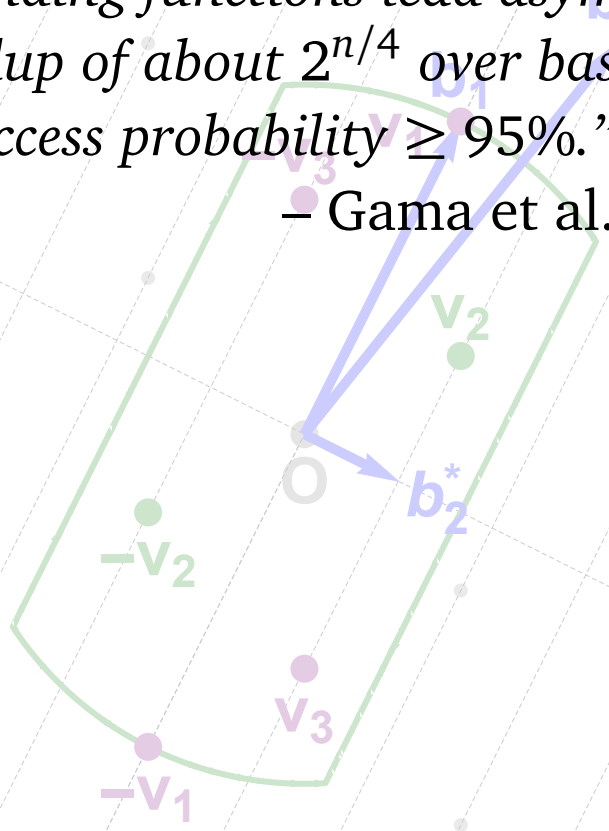
Overview

Pruned enumeration

Overview

"Well-chosen bounding functions lead asymptotically to an exponential speedup of about $2^{n/4}$ over basic enumeration, maintaining a success probability $\geq 95\%$."

– Gama et al., *EUROCRYPT'10*



Pruned enumeration

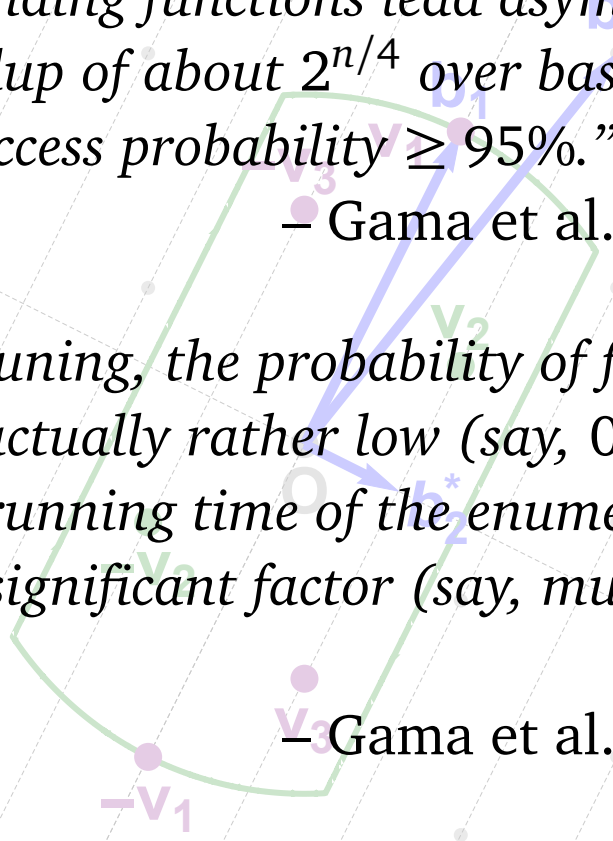
Overview

"Well-chosen bounding functions lead asymptotically to an exponential speedup of about $2^{n/4}$ over basic enumeration, maintaining a success probability $\geq 95\%$."

– Gama et al., *EUROCRYPT'10*

"With extreme pruning, the probability of finding the desired vector is actually rather low (say, 0.1%), but surprisingly, the running time of the enumeration is reduced by a much more significant factor (say, much more than 1000)."

– Gama et al., *EUROCRYPT'10*



Outline

Enumeration algorithms
- Fincke-Pohst enumeration
- Kannan enumeration
- Pruning the enumeration tree

Sieving algorithms
- Nguyen-Vidick sieve
- Multiple levels
- Near neighbor techniques

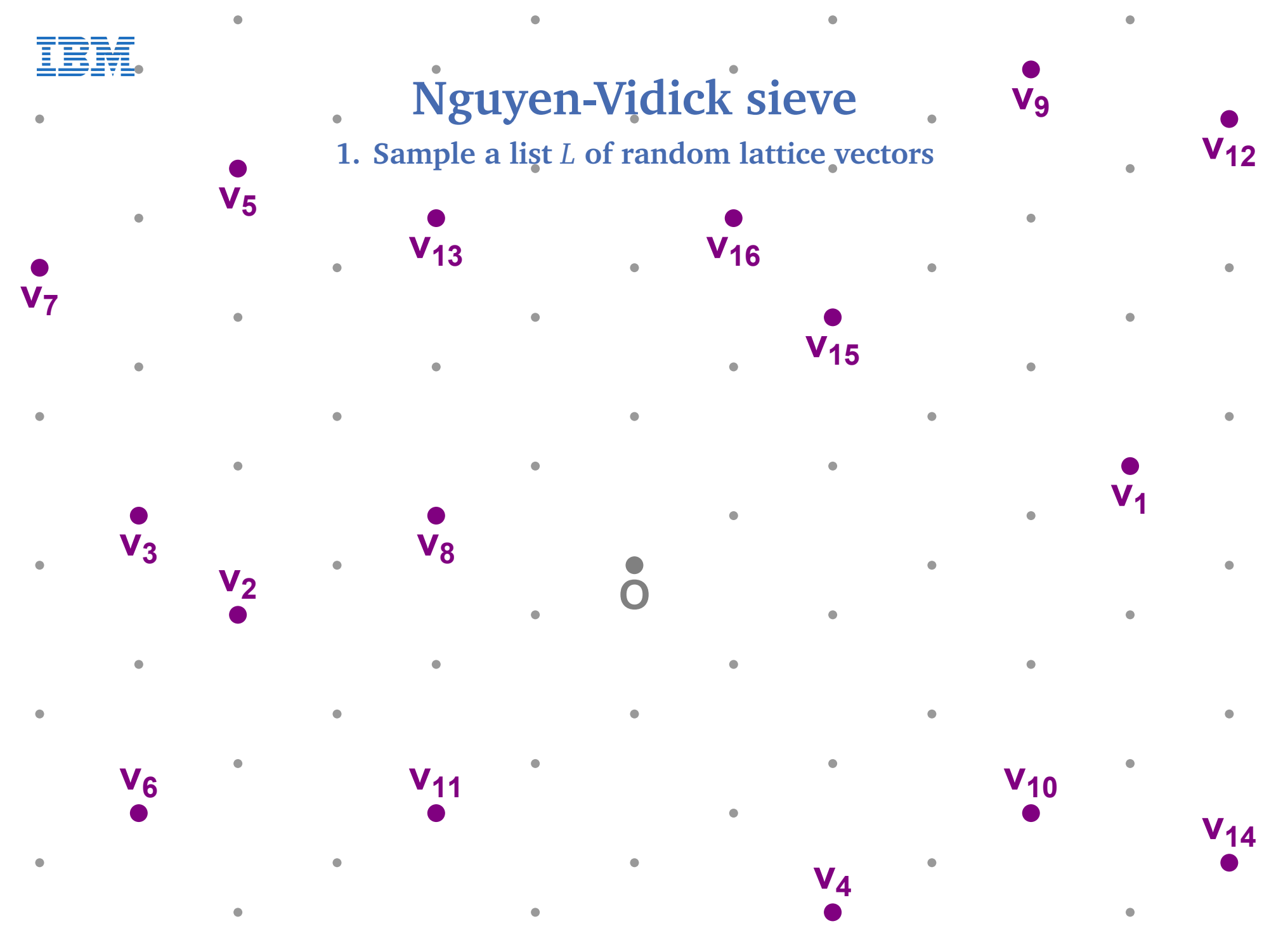
Practical comparison

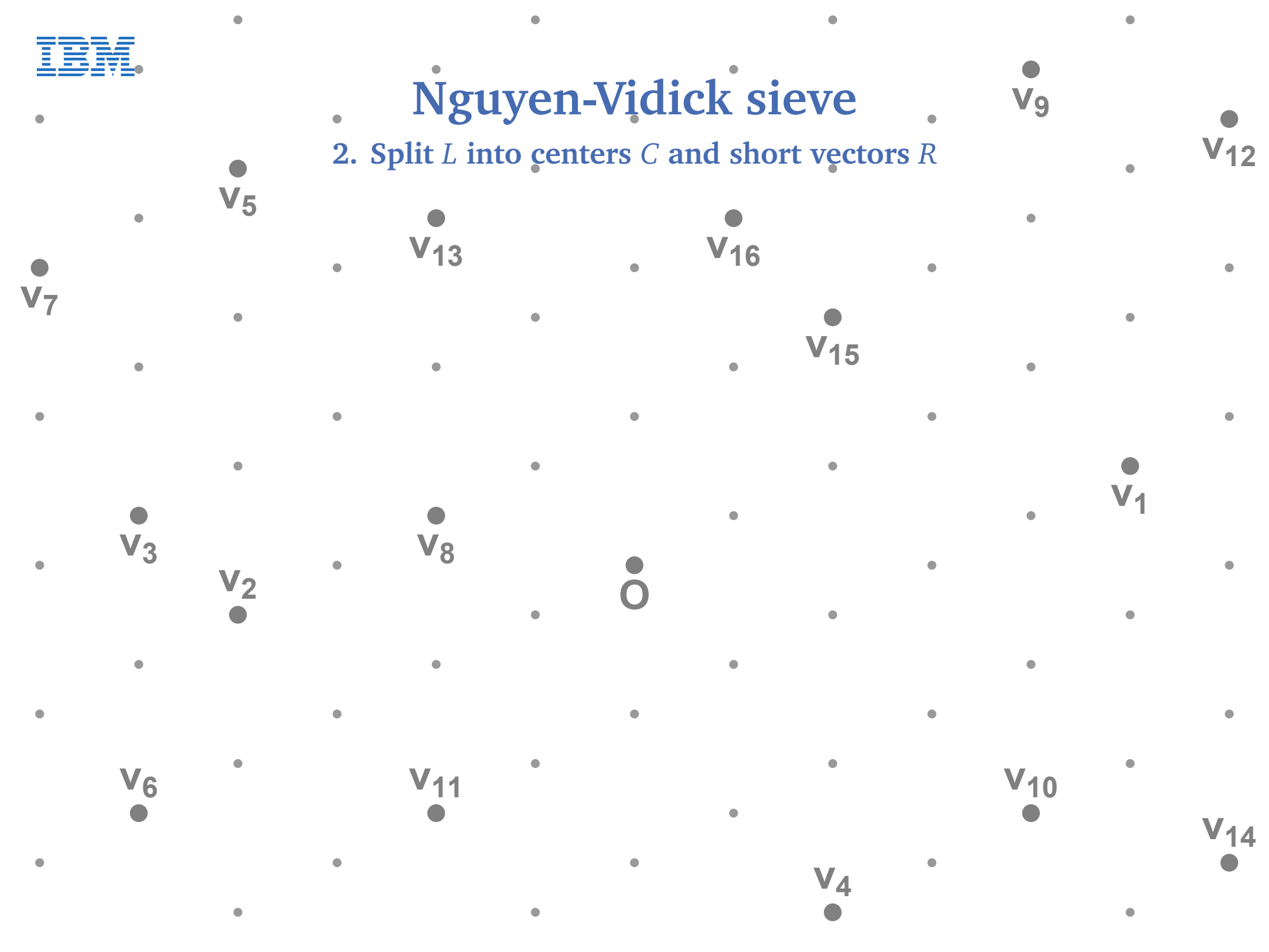
Nguyen-Vidick sieve

1. Sample a list L of random lattice vectors

Nguyen-Vidick sieve

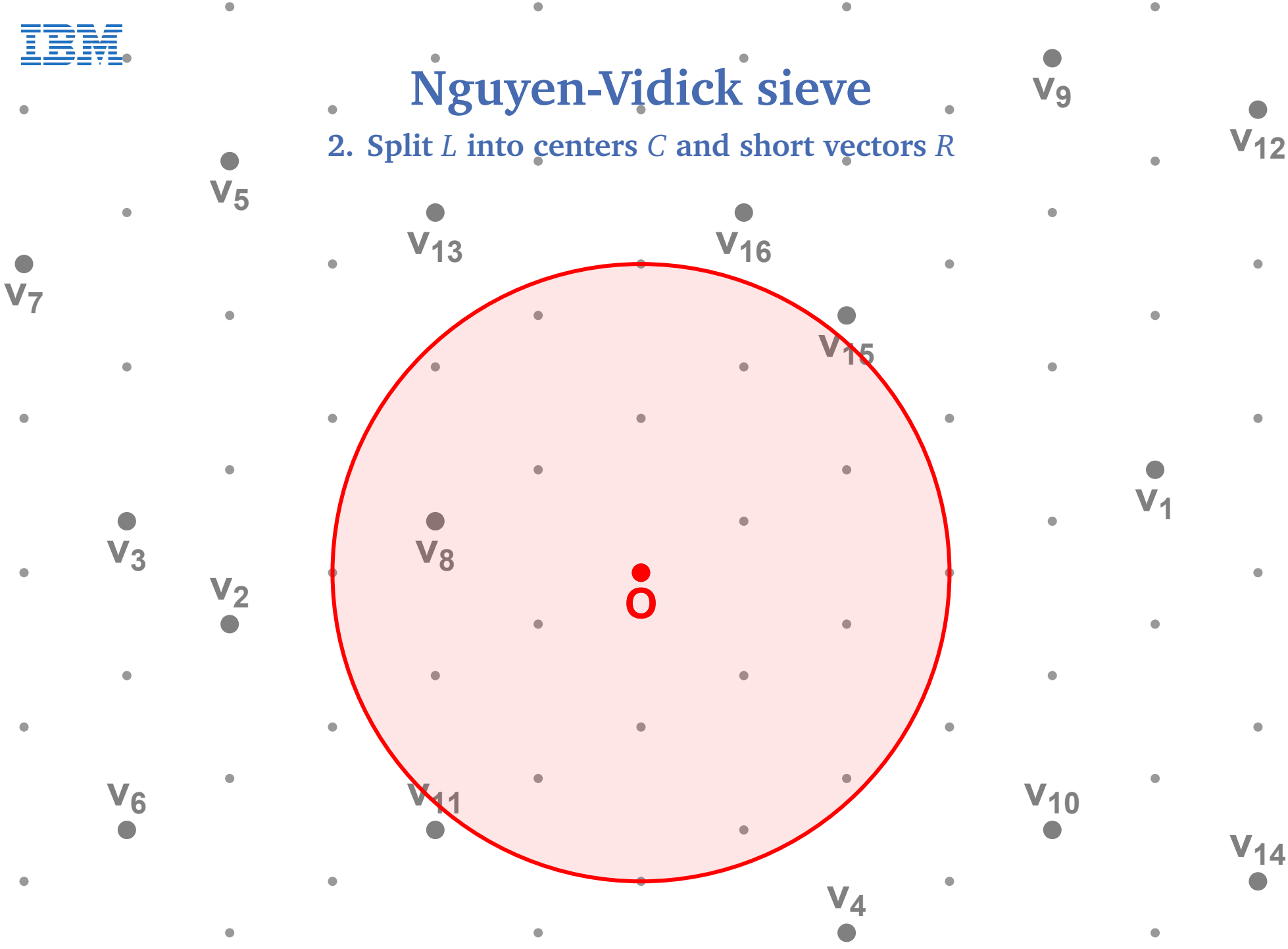
1. Sample a list L of random lattice vectors




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

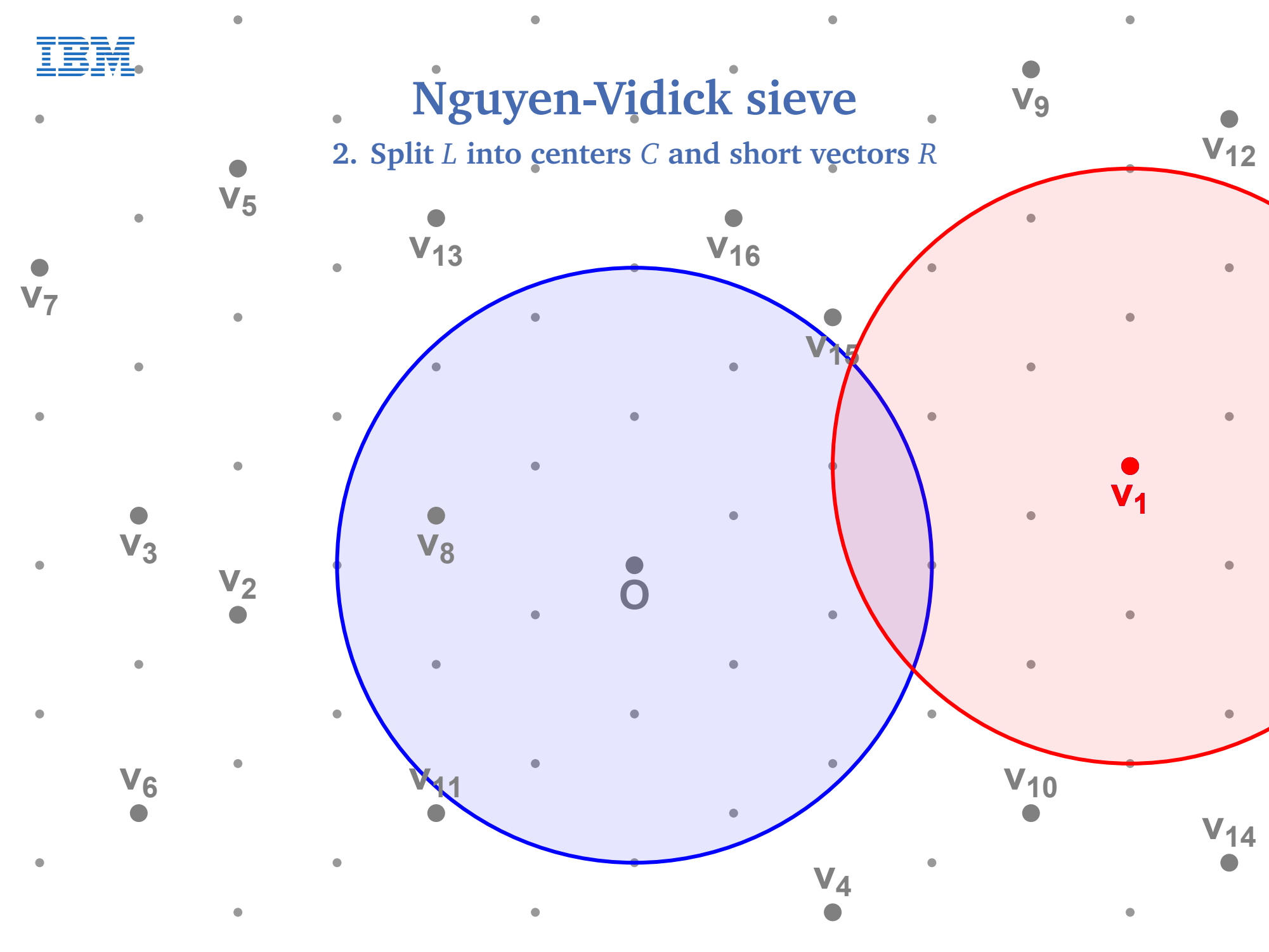
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



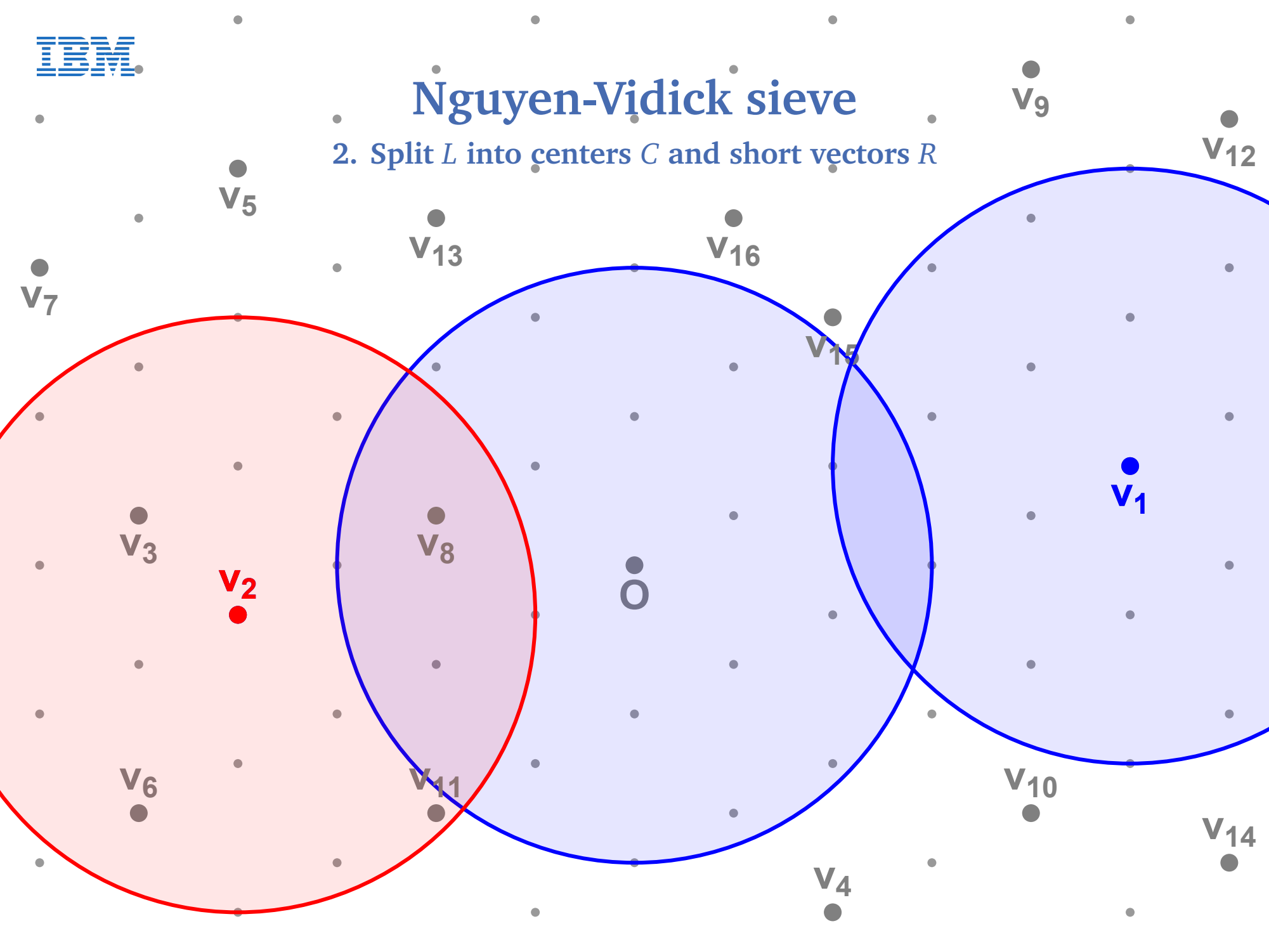
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



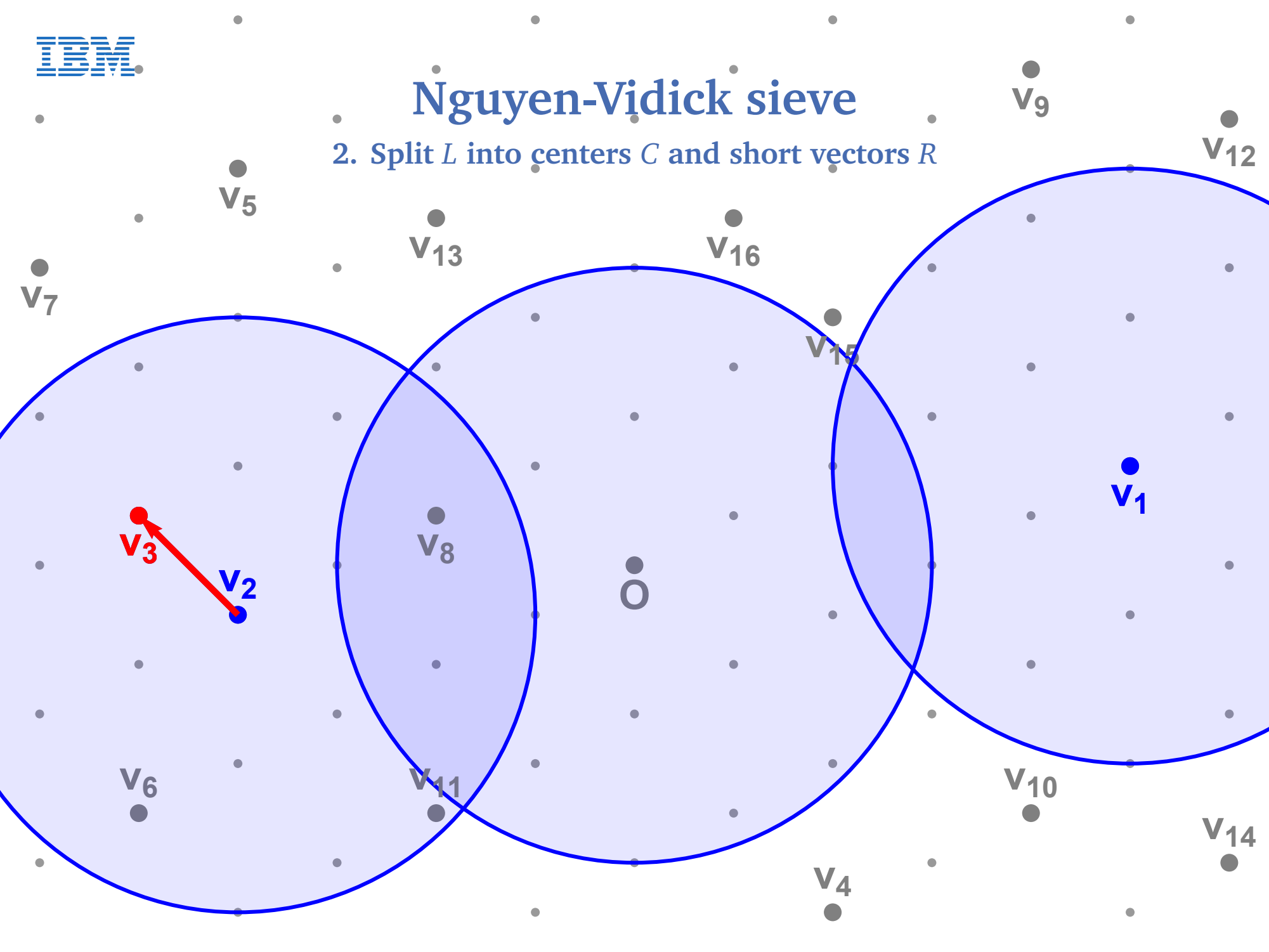
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



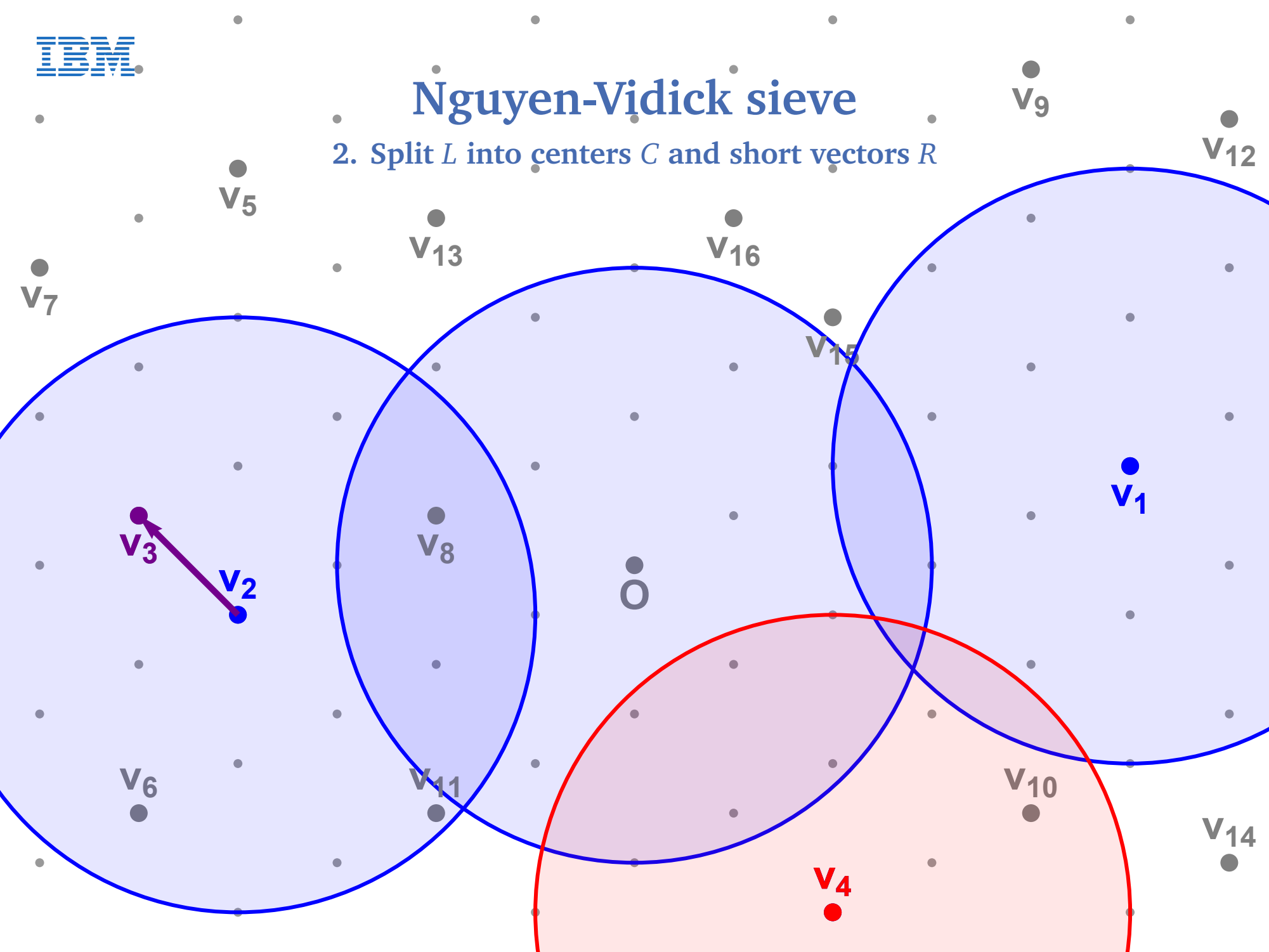
Nguyen-Vidick sieve

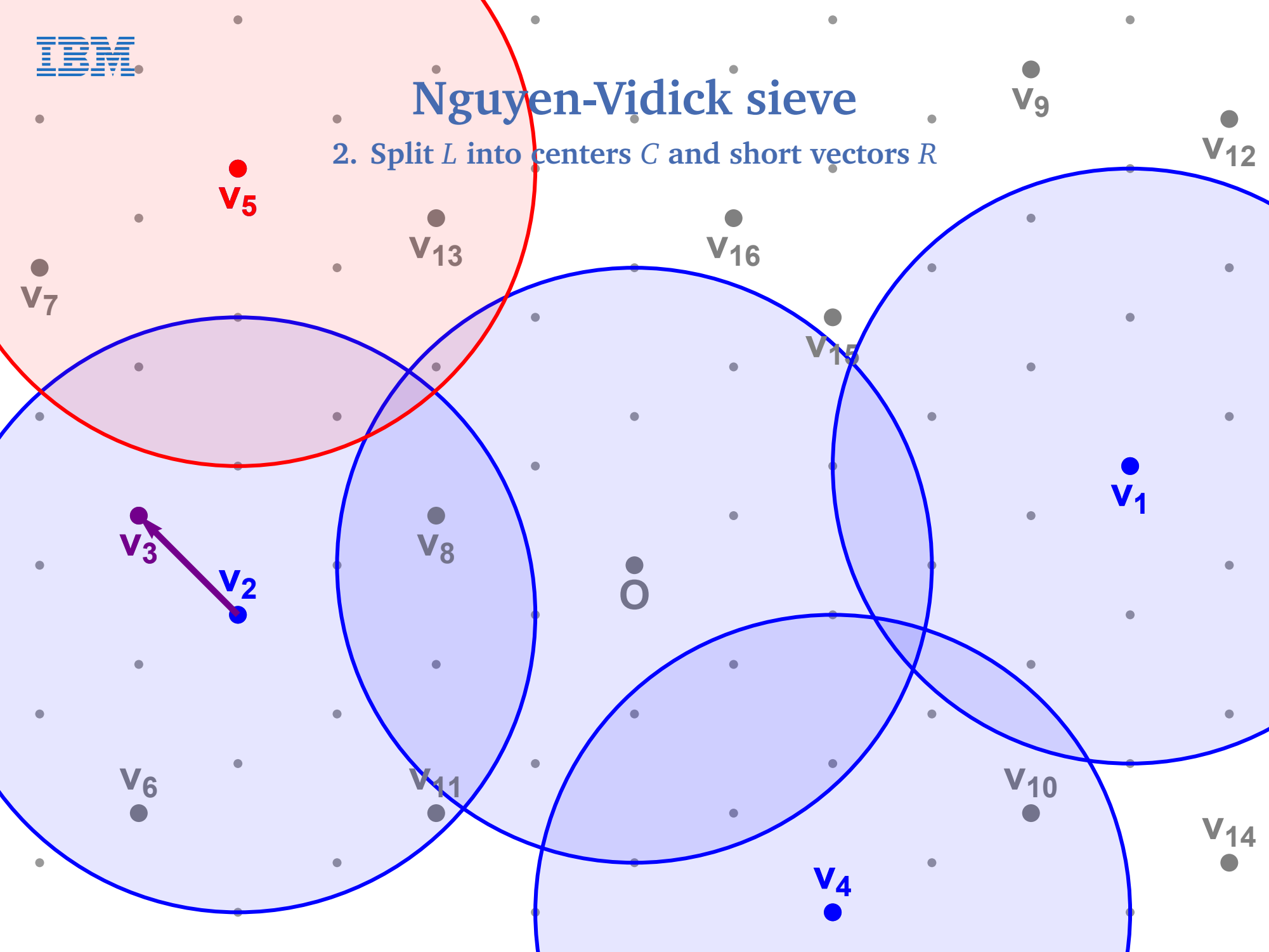
2. Split L into centers C and short vectors R



Nguyen-Vidick sieve

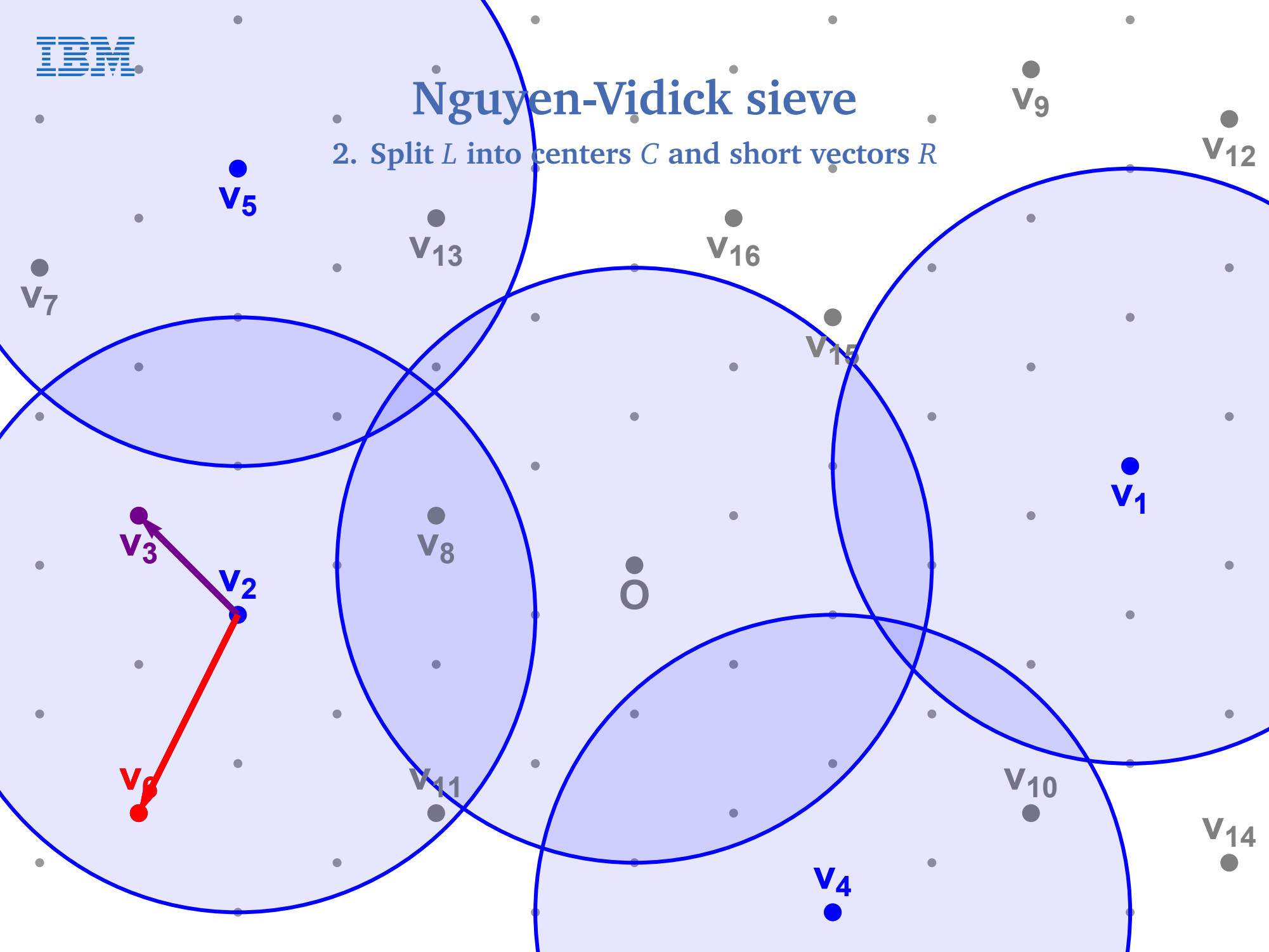
2. Split L into centers C and short vectors R




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

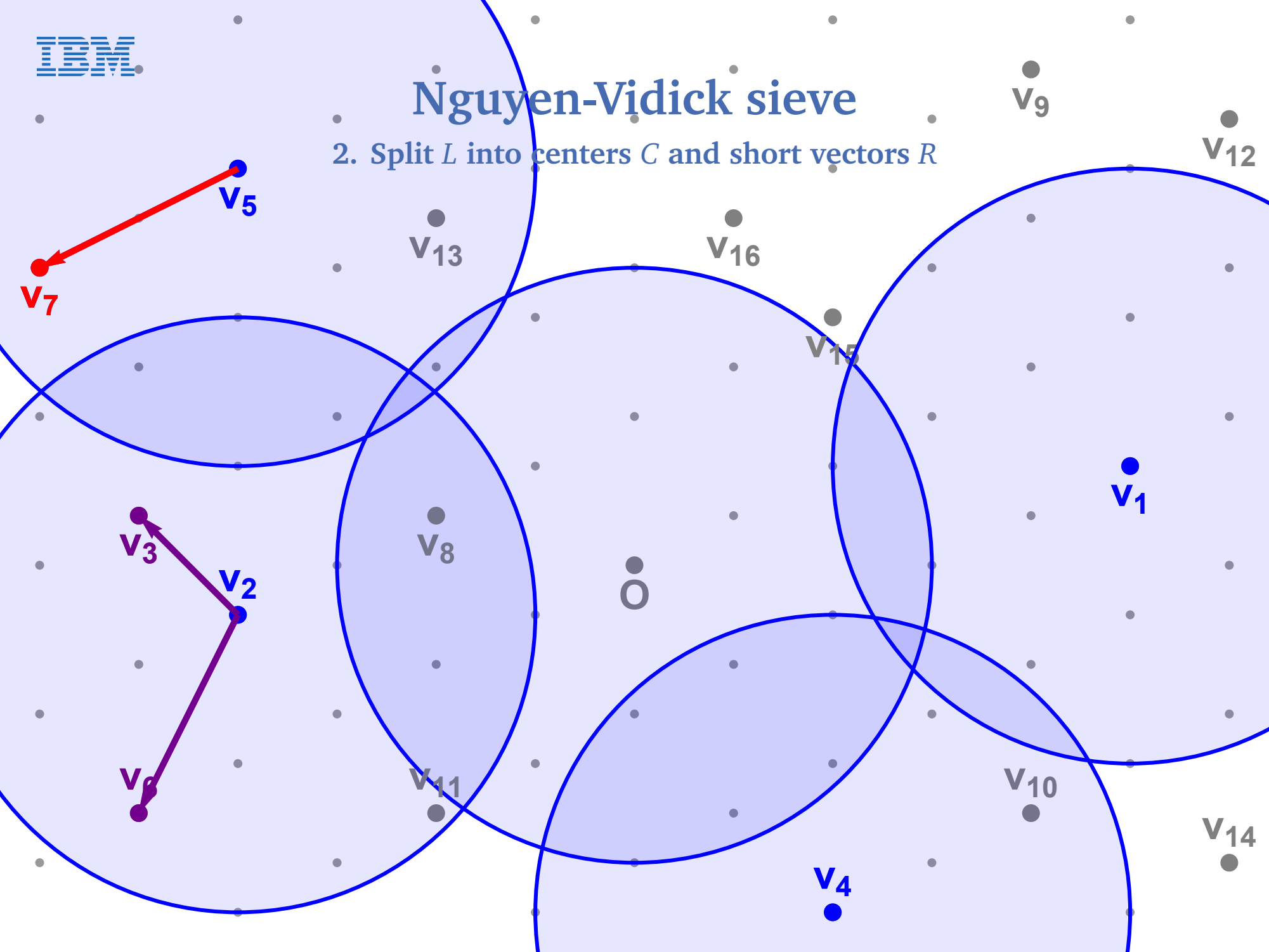
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



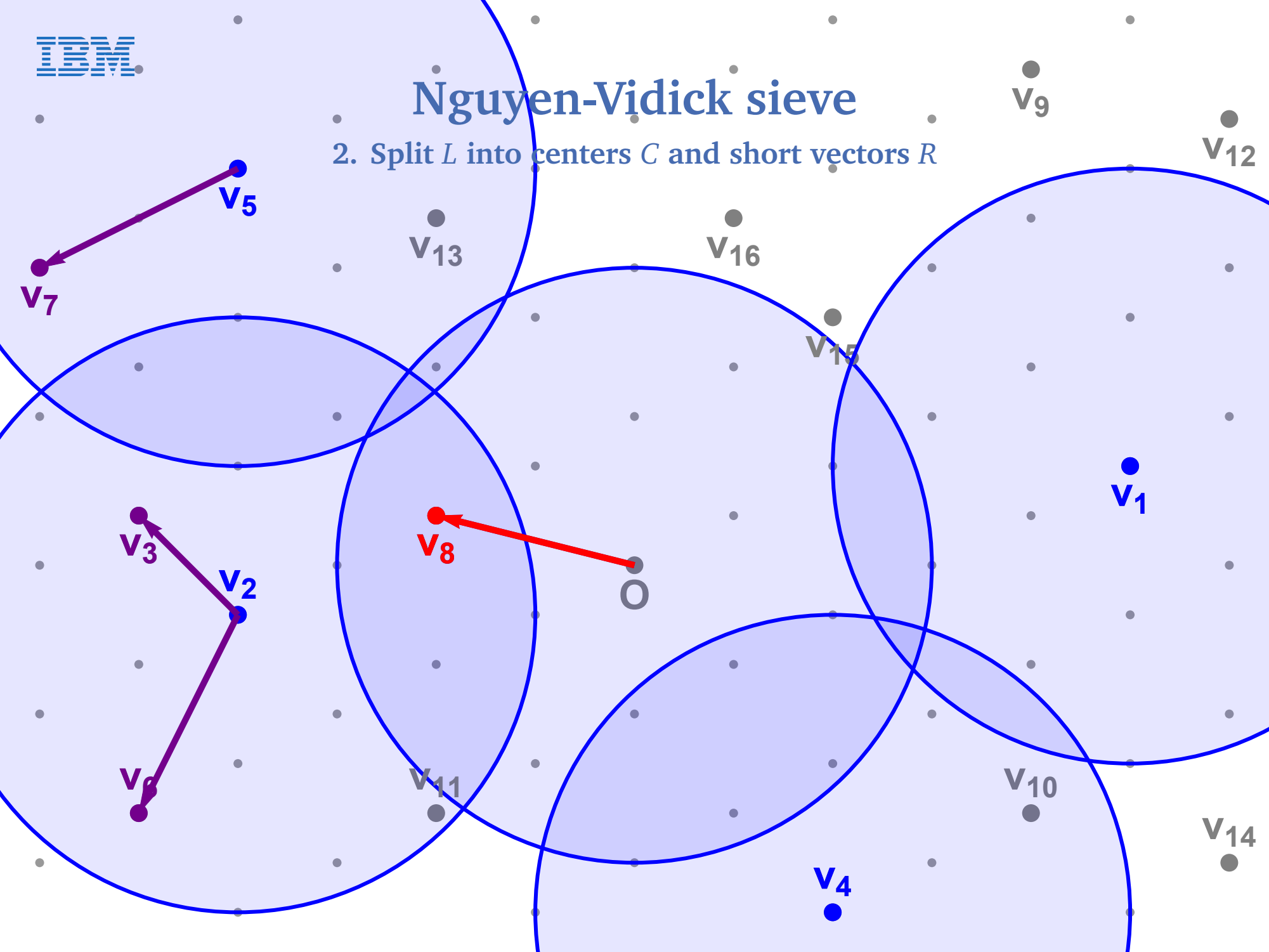
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



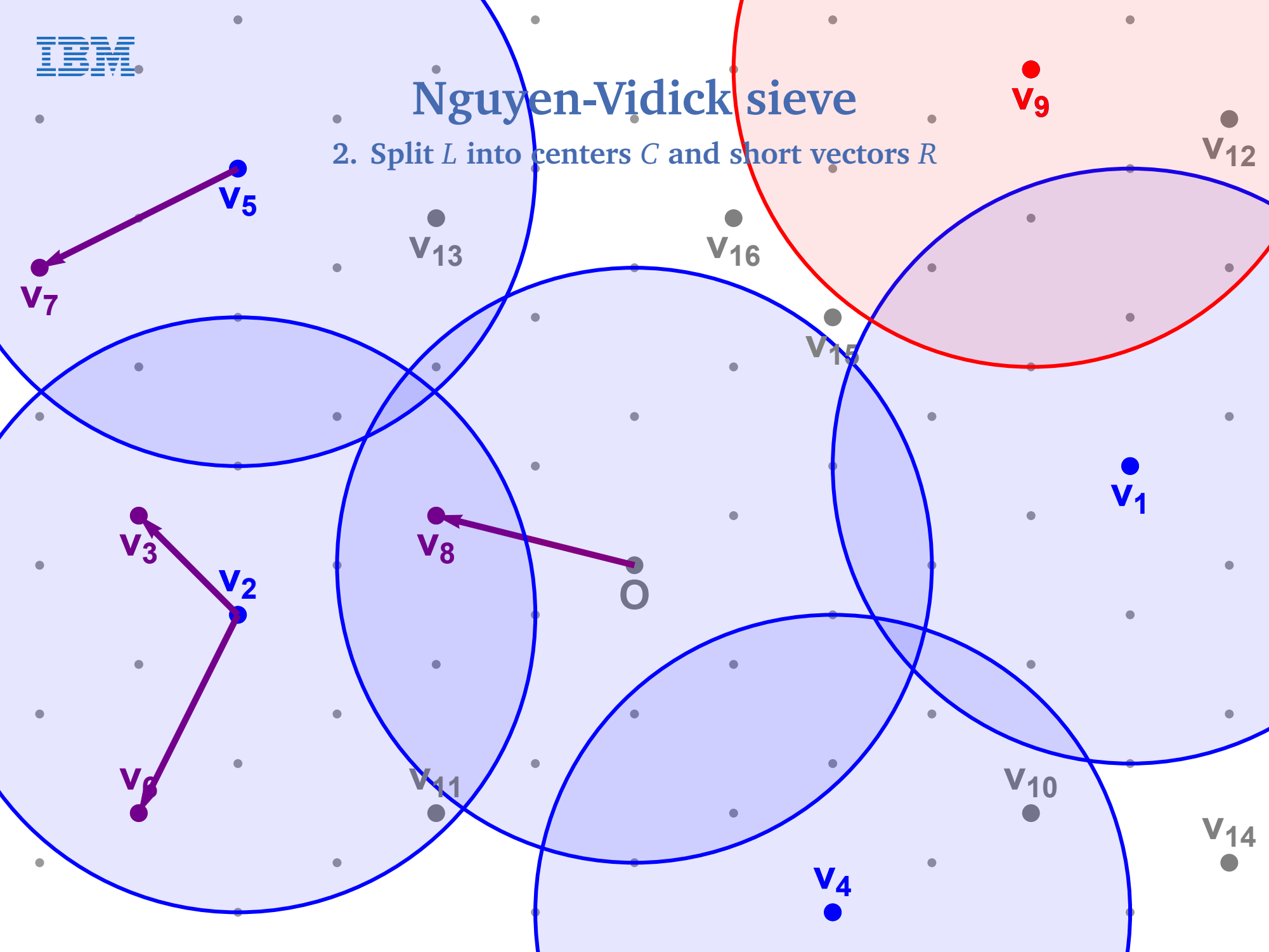
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



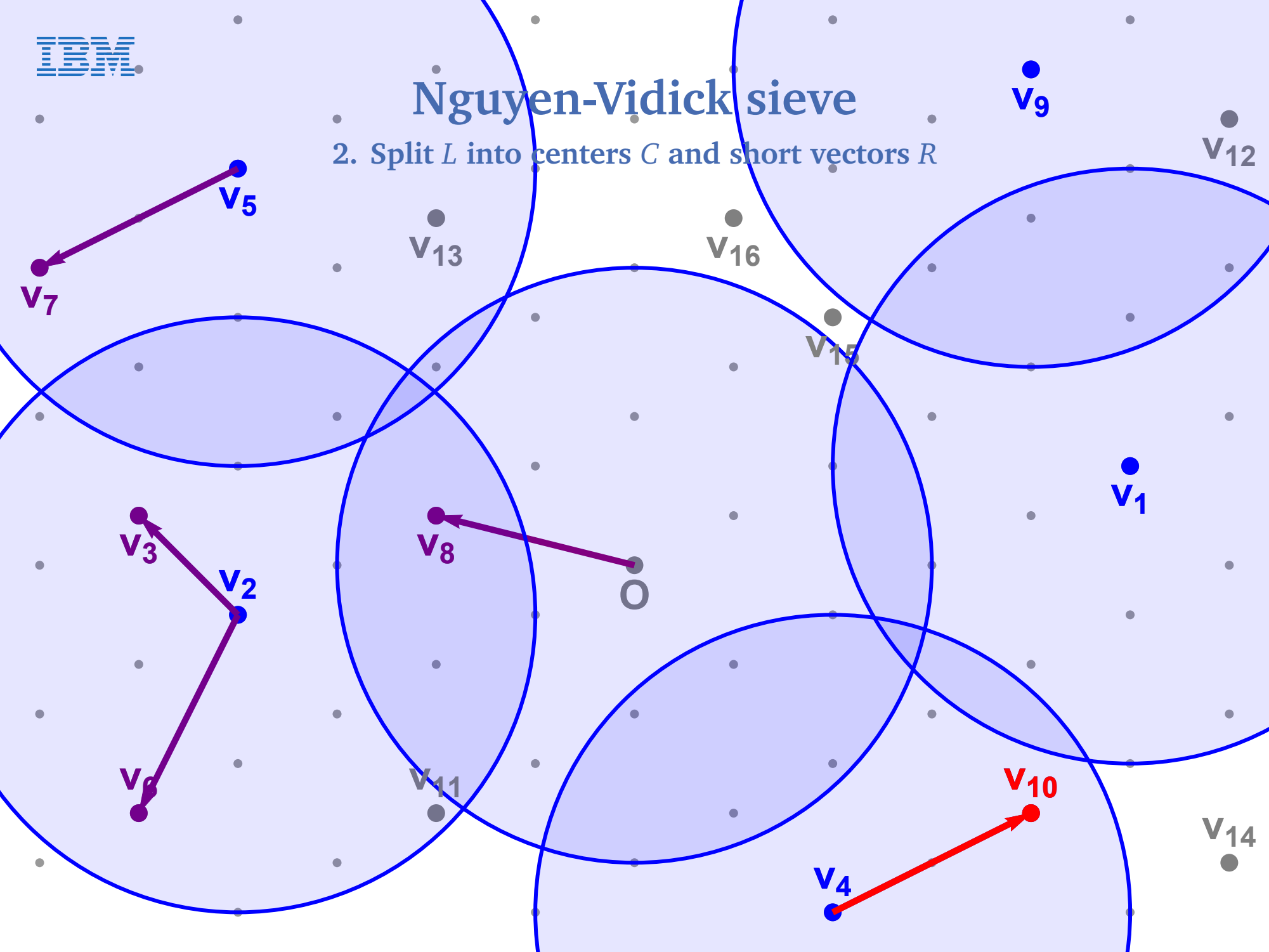
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



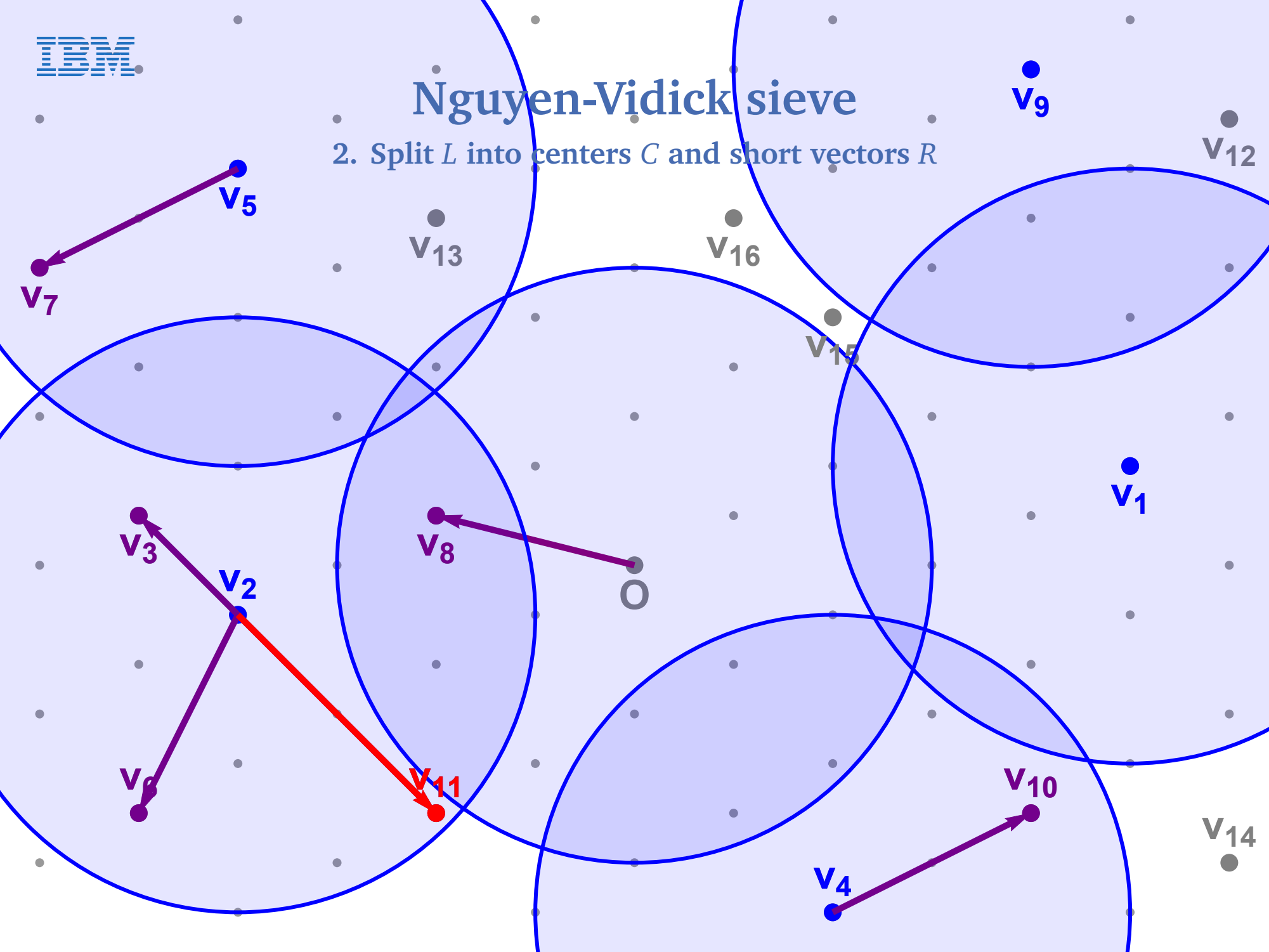
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



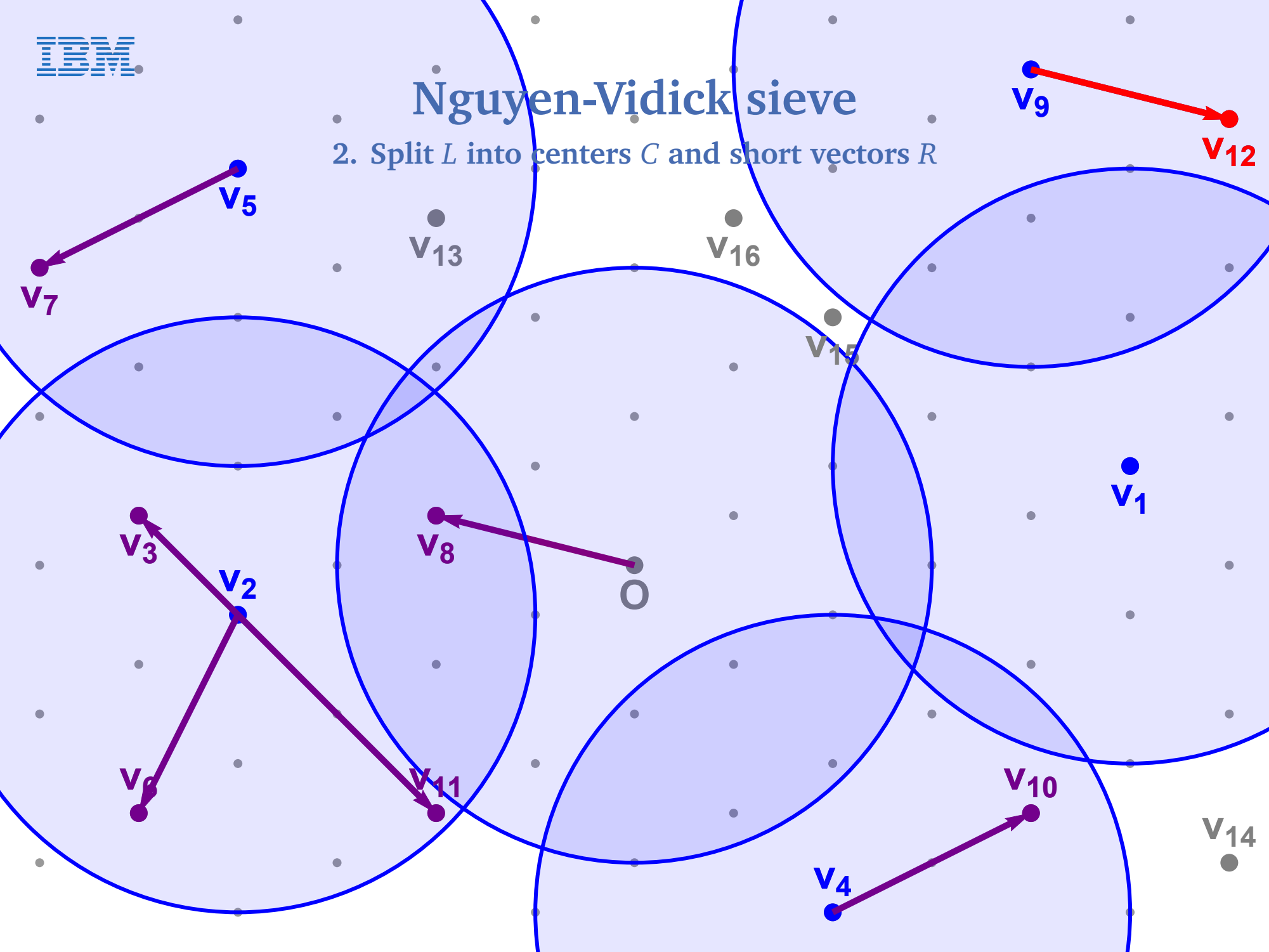
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



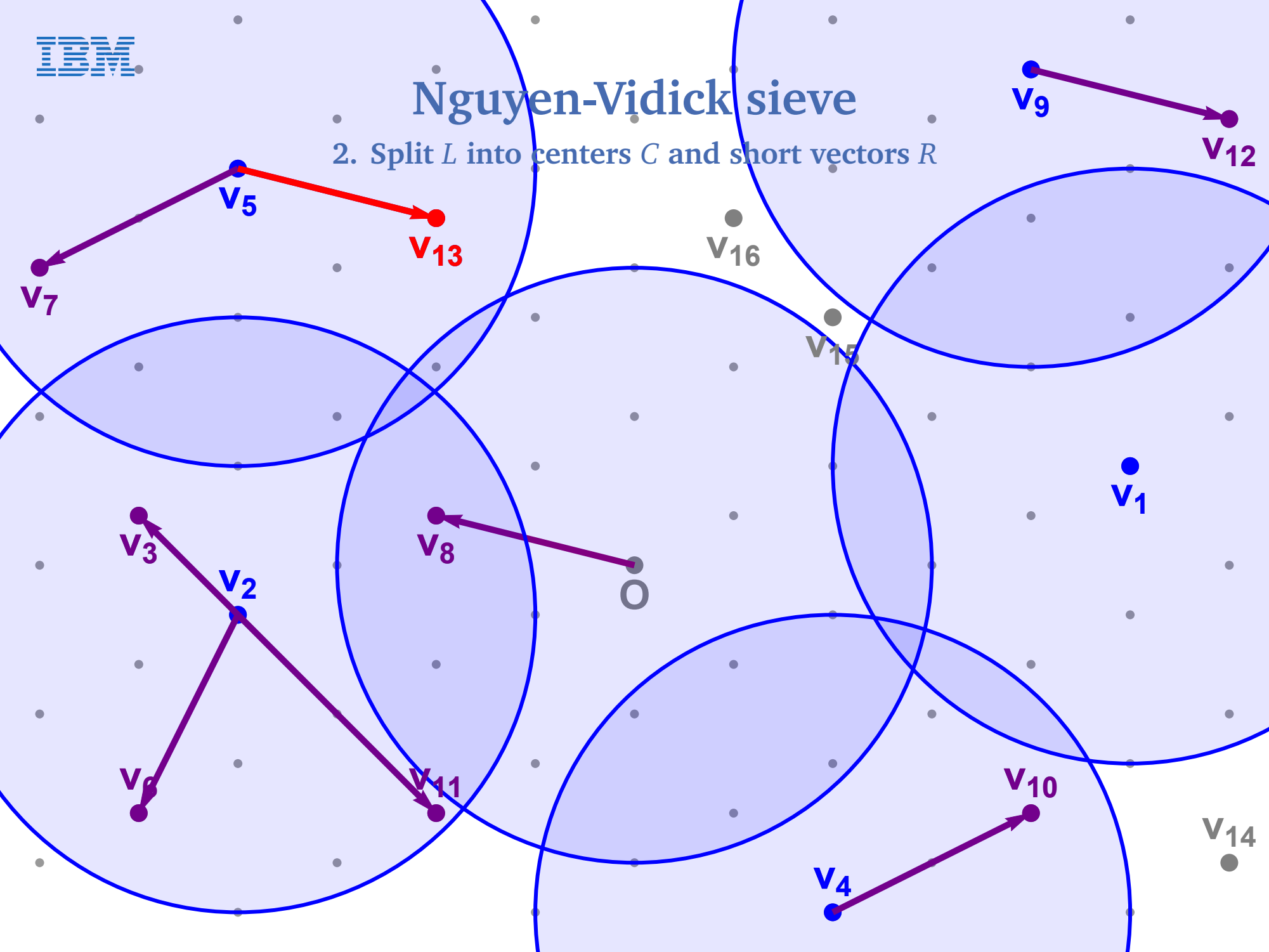
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



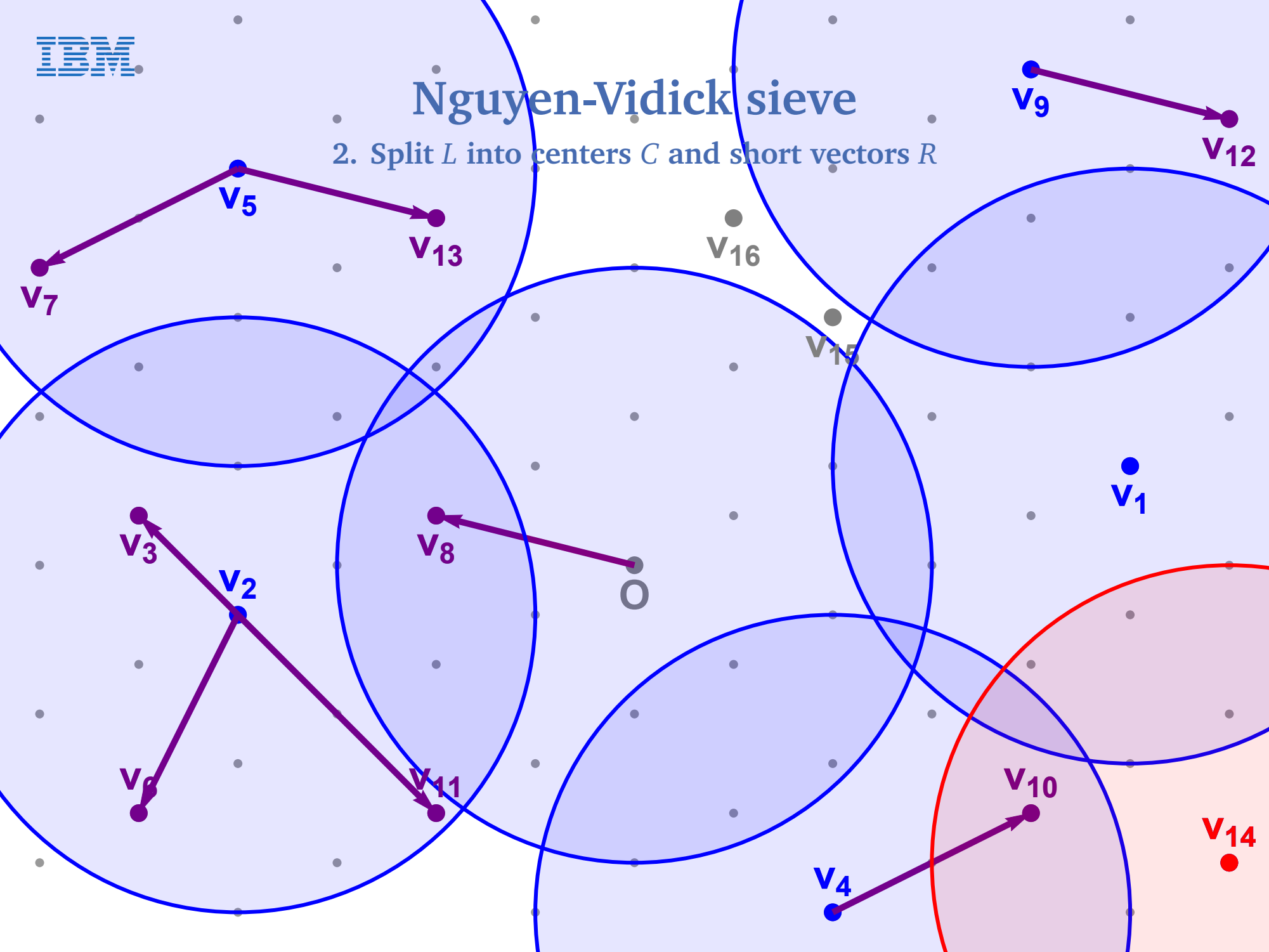
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



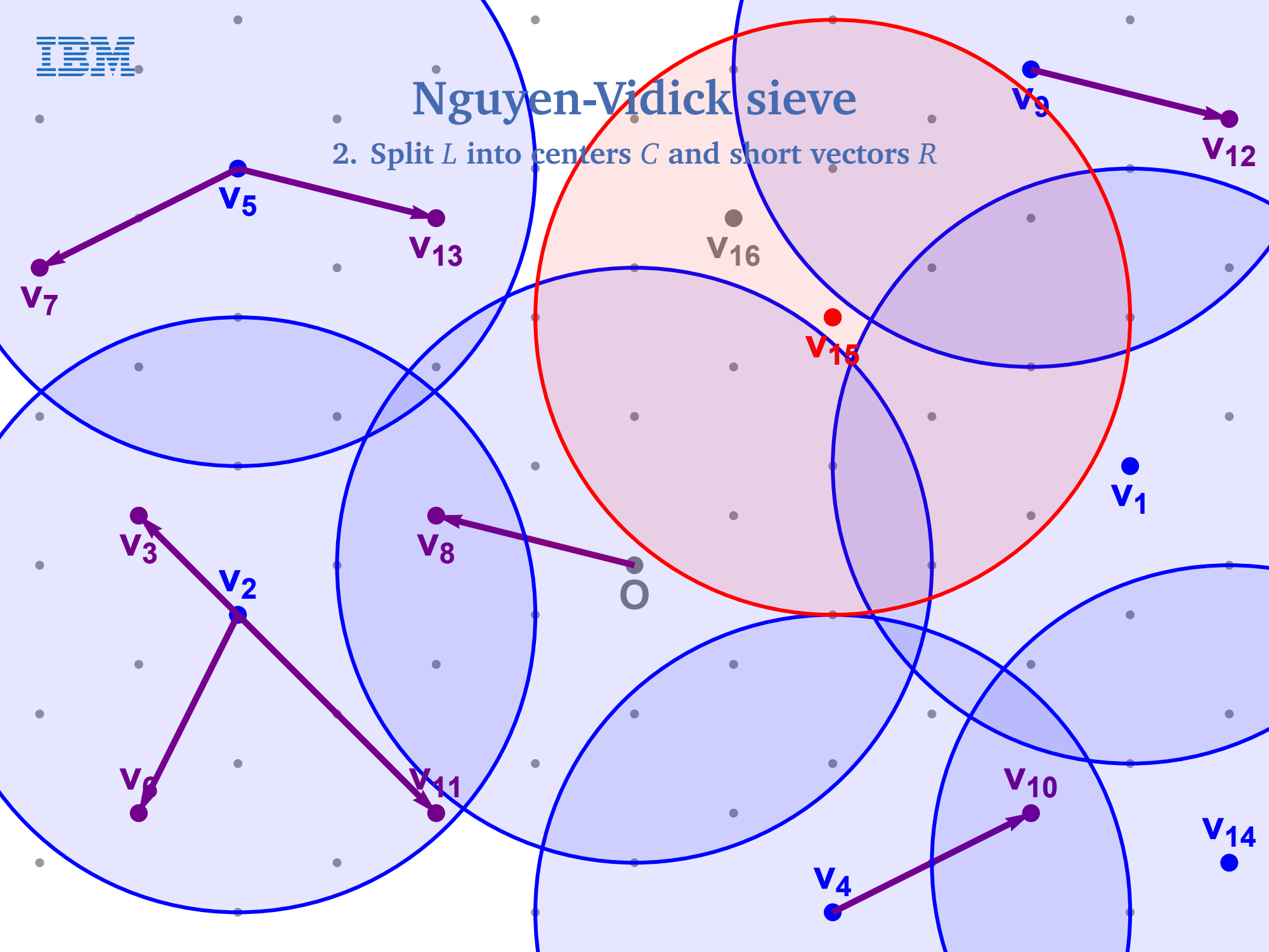
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



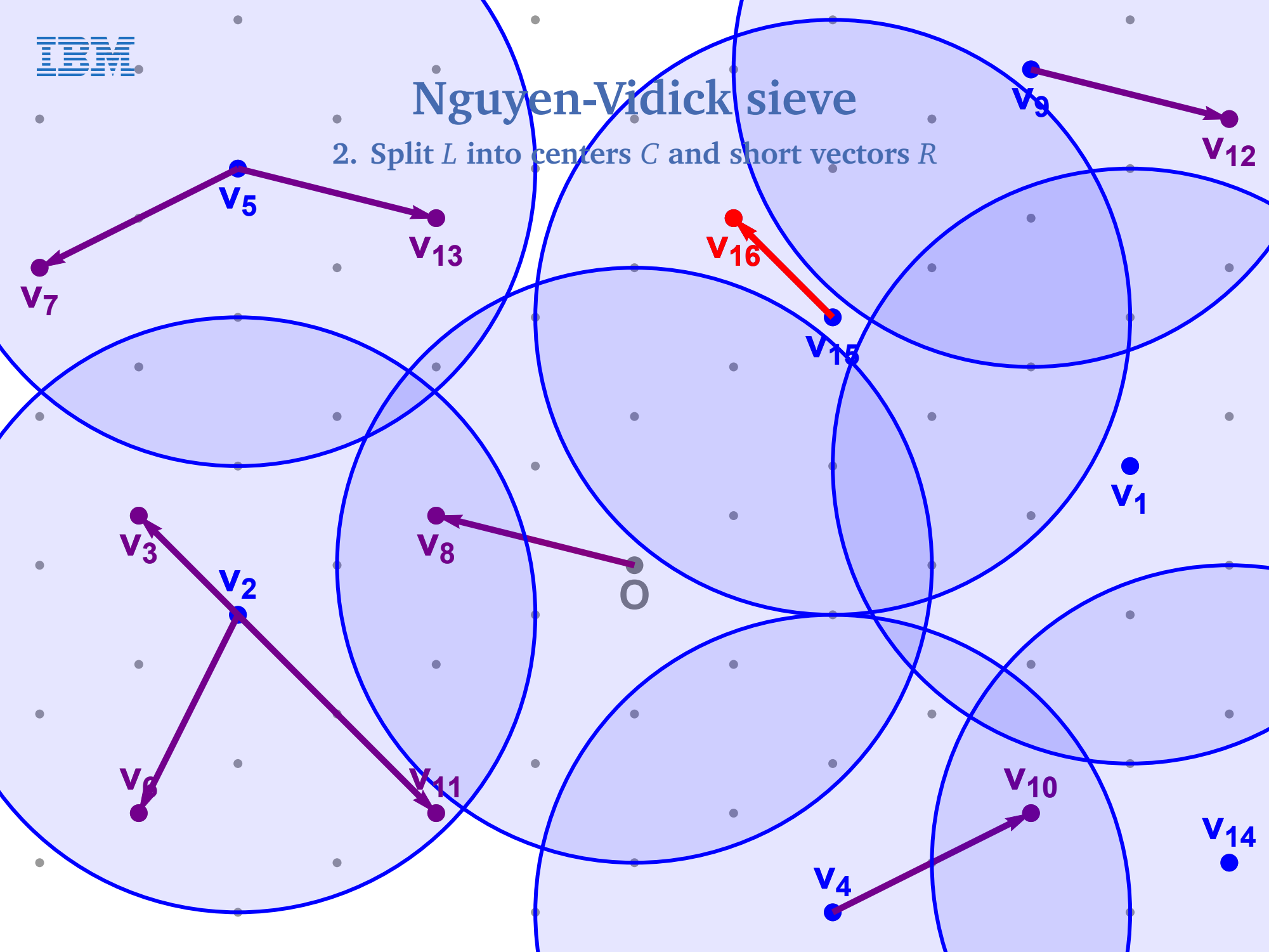
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



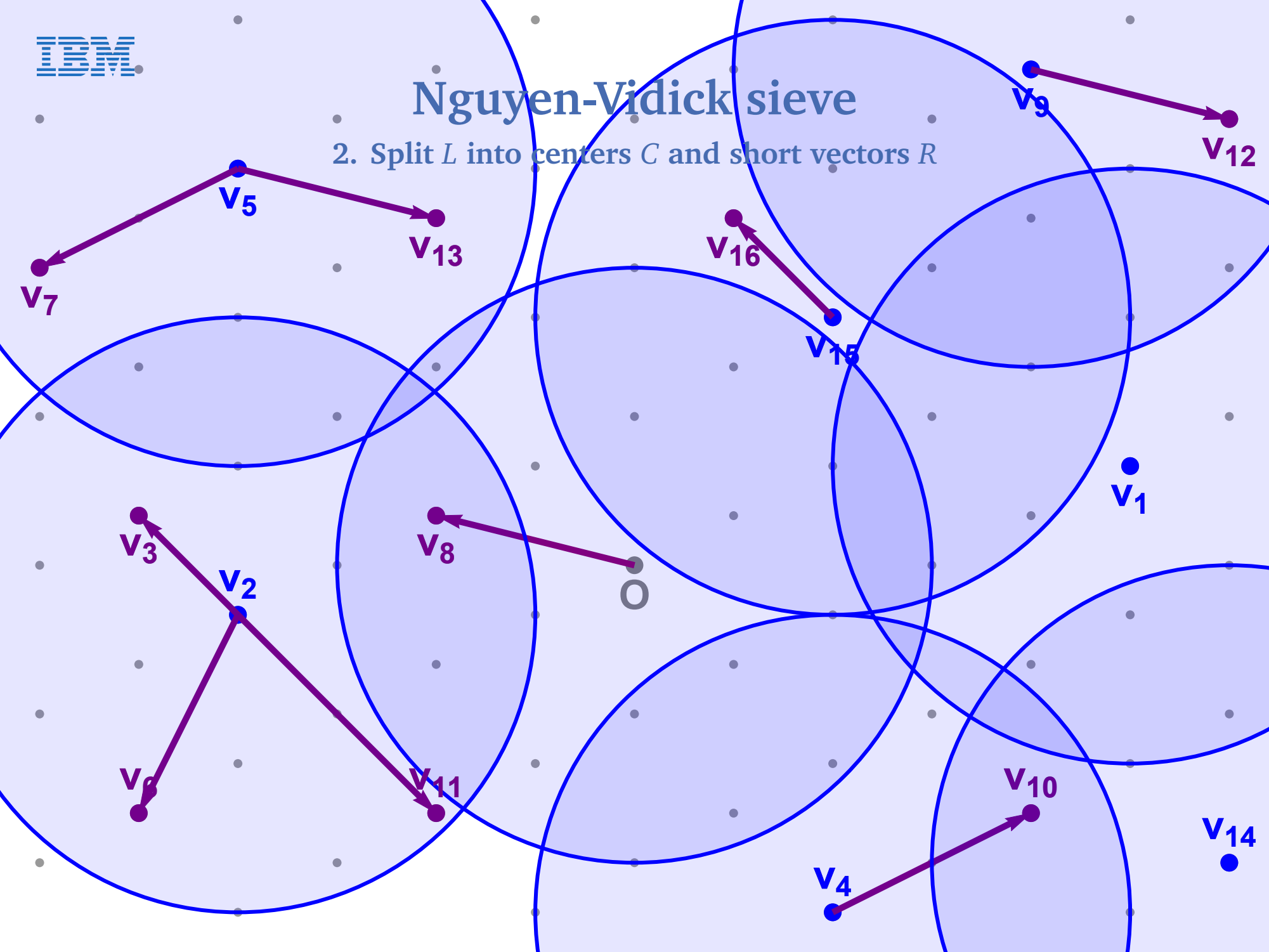
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



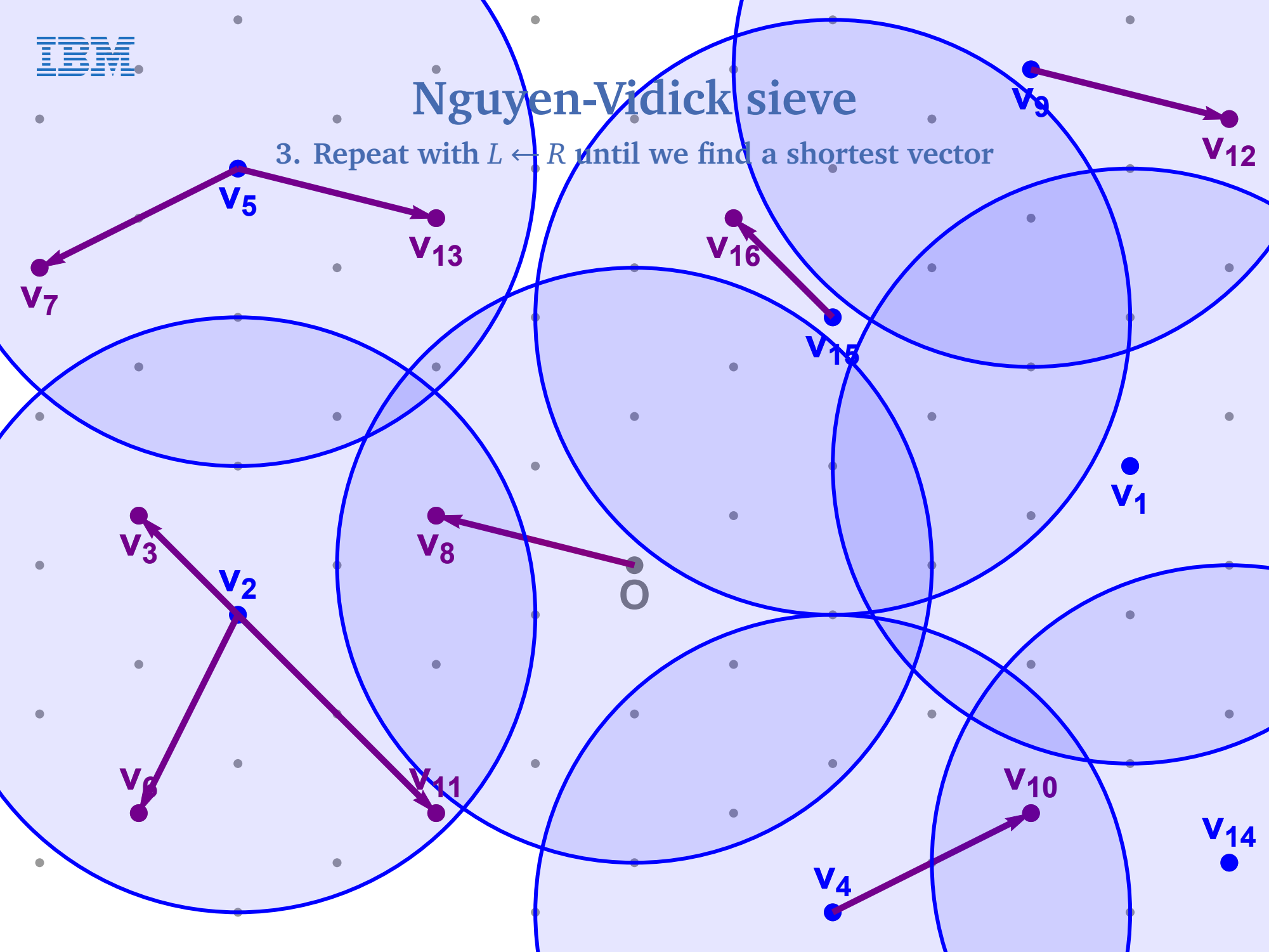
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



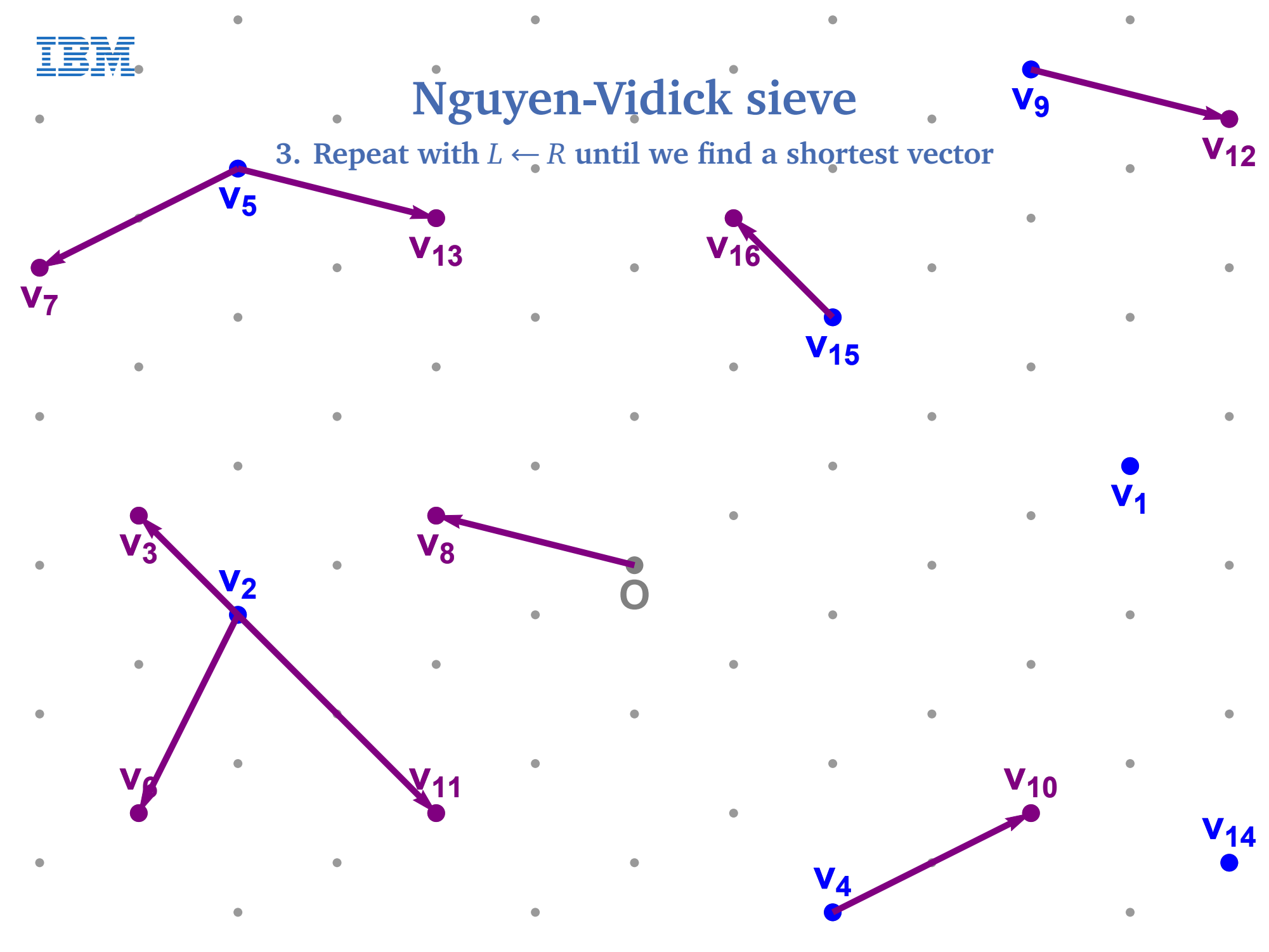
Nguyen-Vidick sieve

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



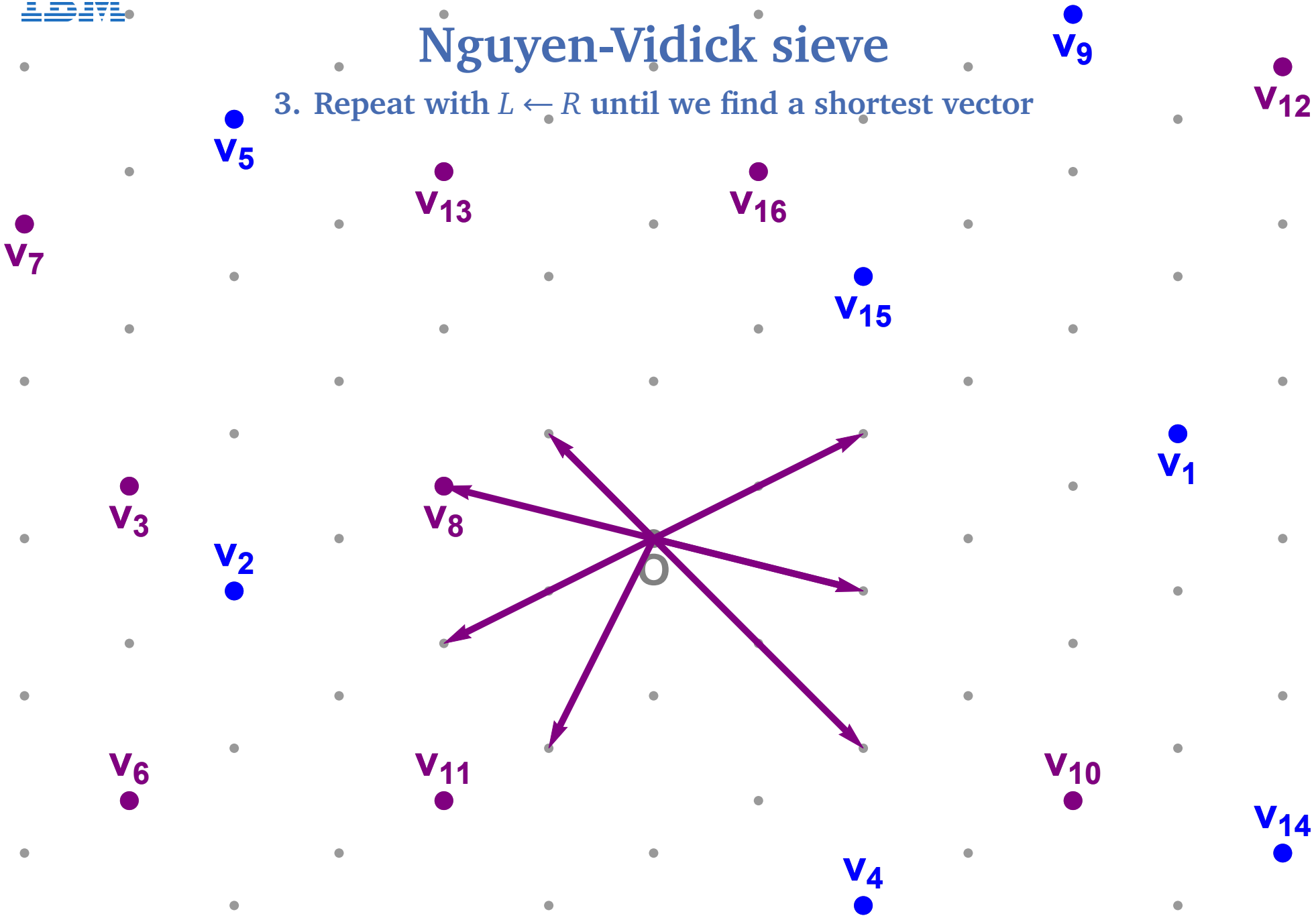
Nguyen-Vidick sieve

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



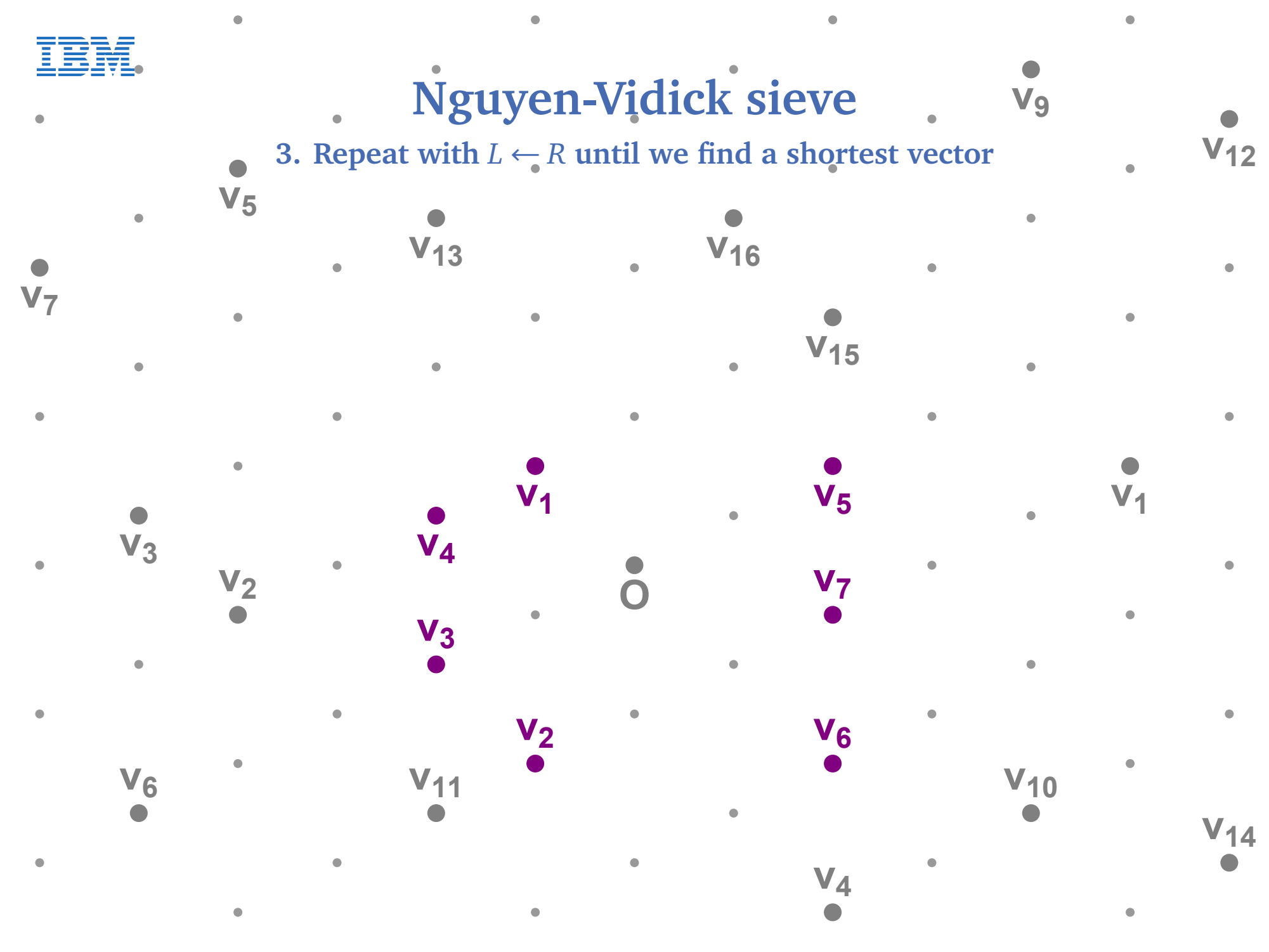
Nguyen-Vidick sieve

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

Nguyen-Vidick sieve

Overview

Heuristic (Nguyen and Vidick, J. Math. Crypt. '08)

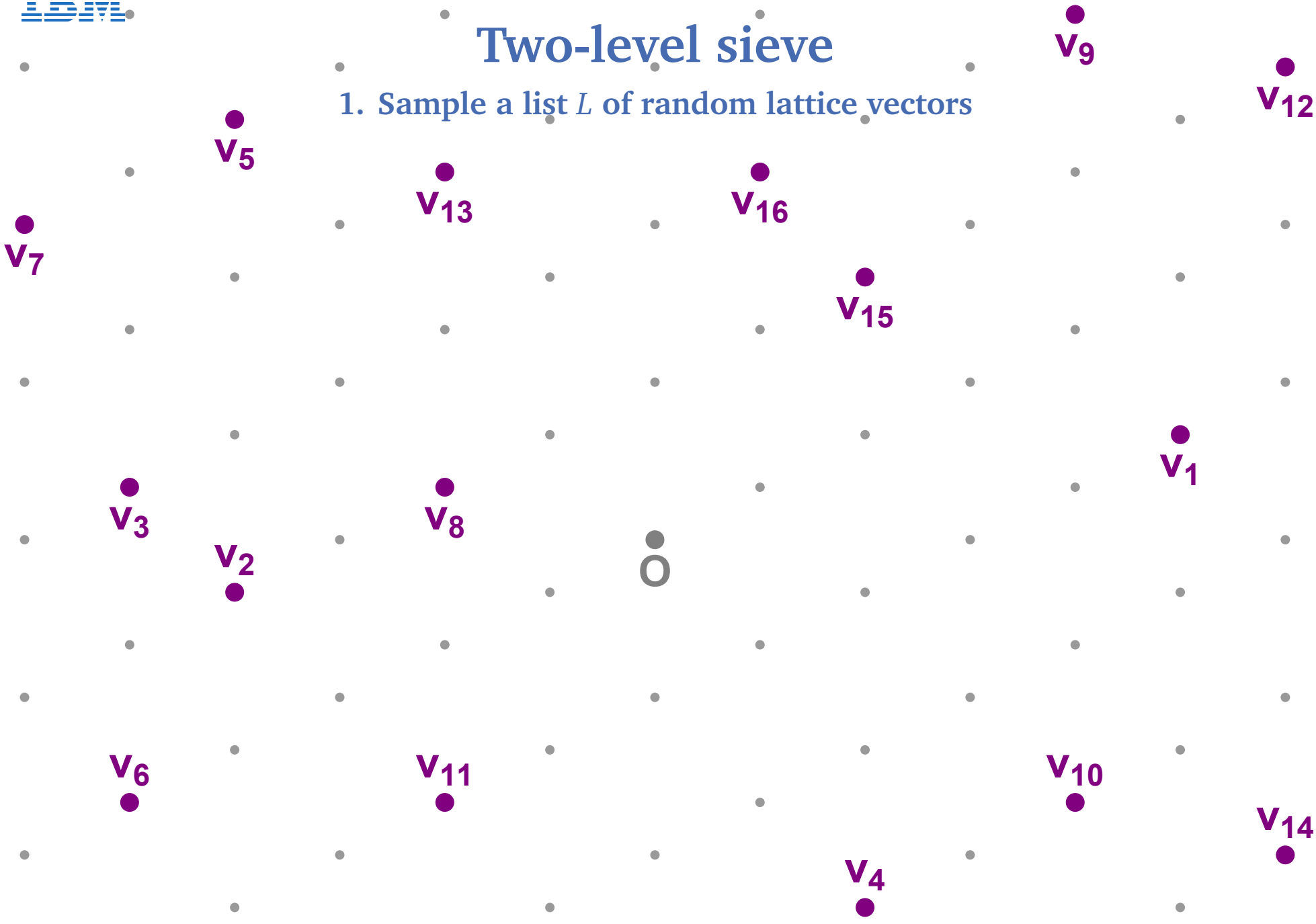
The Nguyen-Vidick sieve runs in time $(4/3)^n$ *and space* $\sqrt{4/3}^n$.

Two-level sieve

1. Sample a list L of random lattice vectors

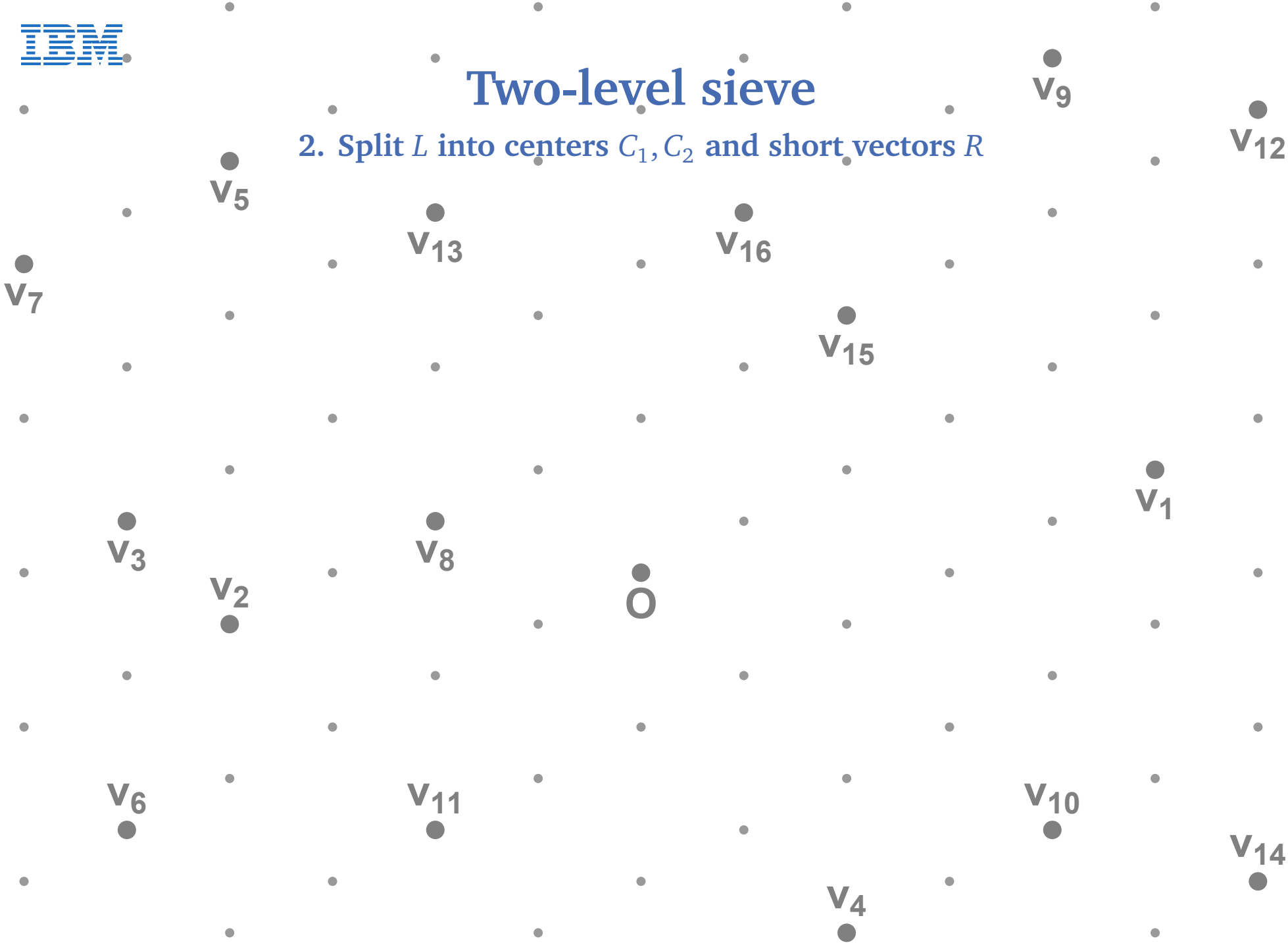
Two-level sieve

1. Sample a list L of random lattice vectors



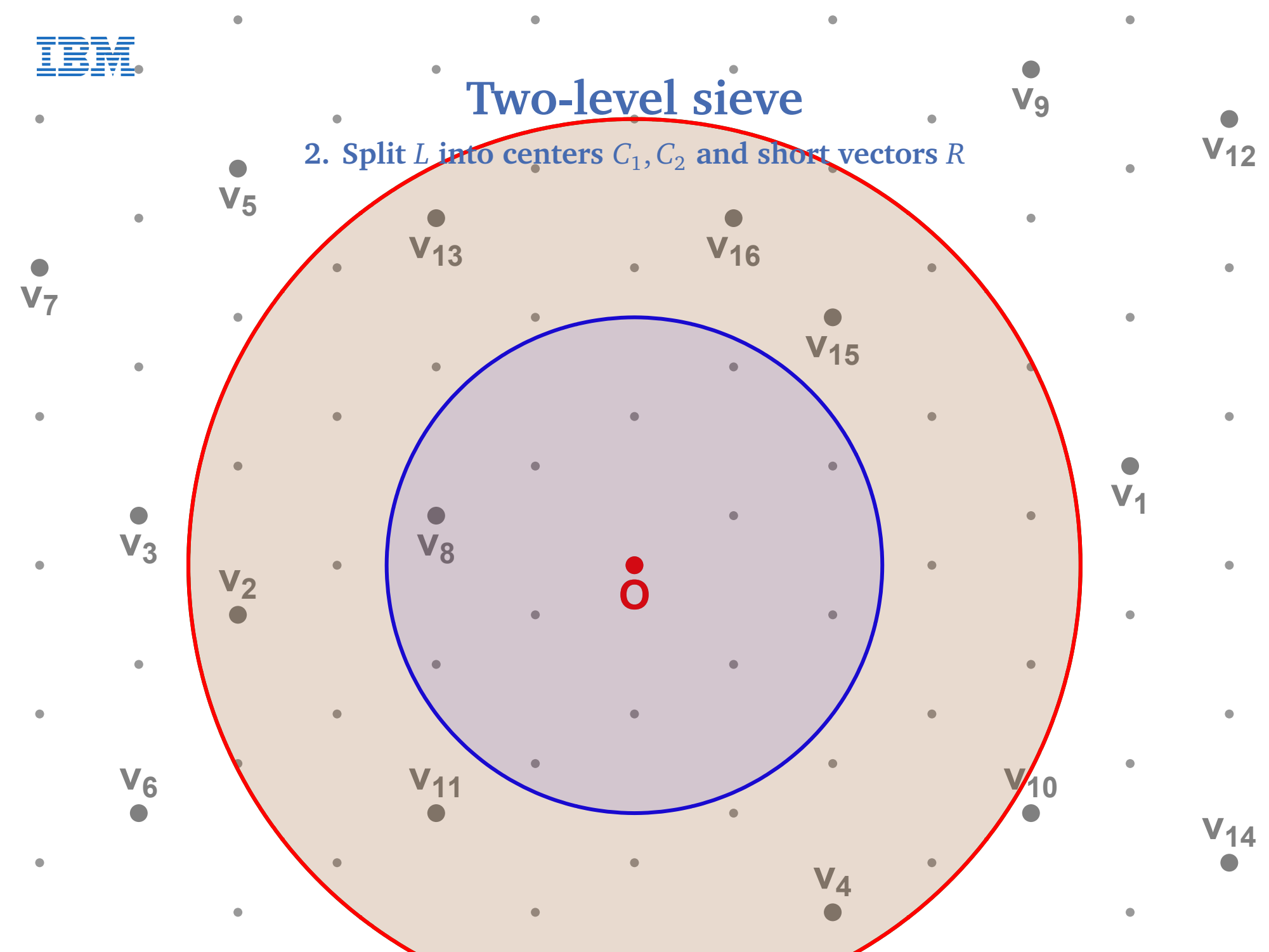
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



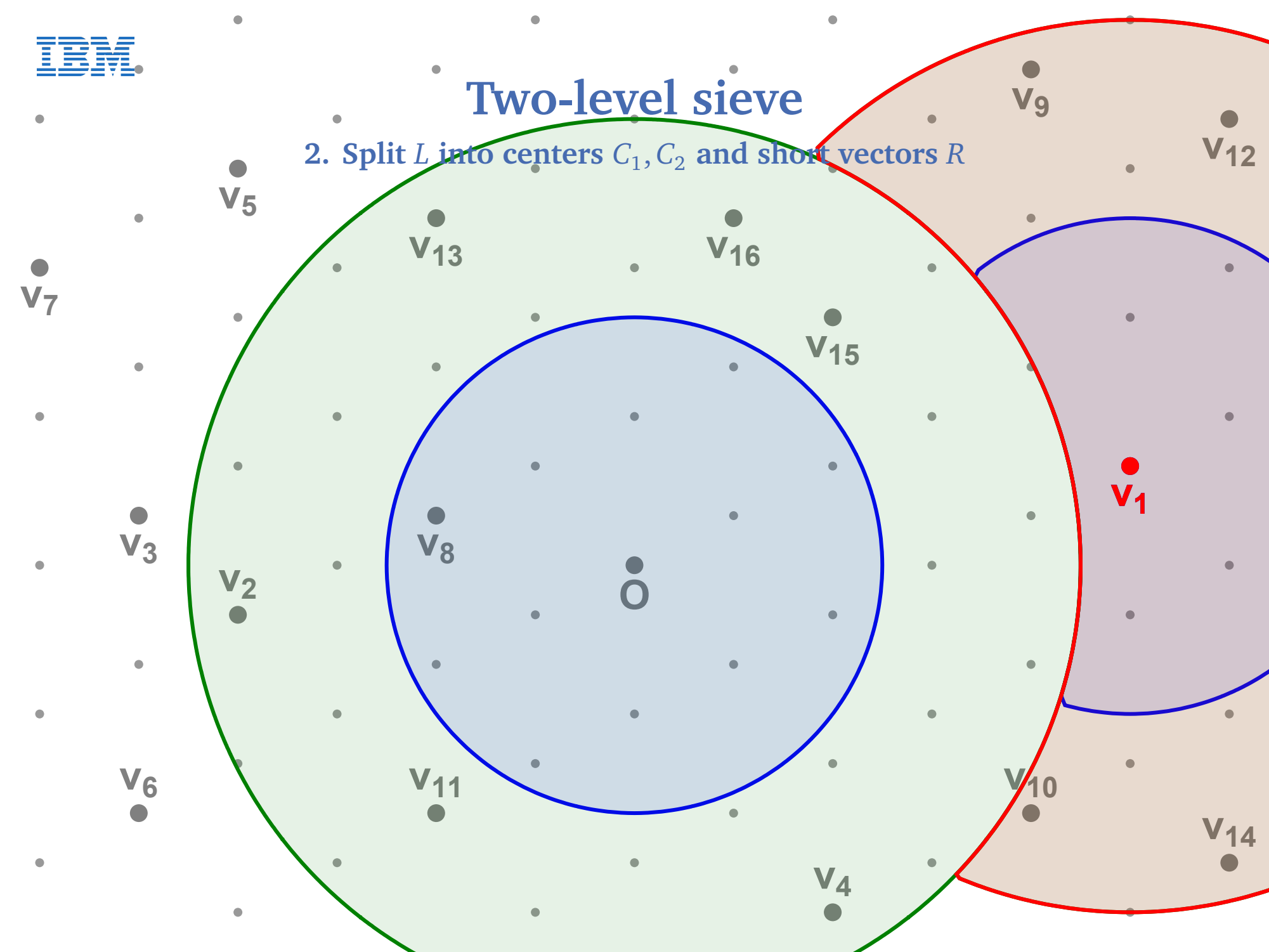
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



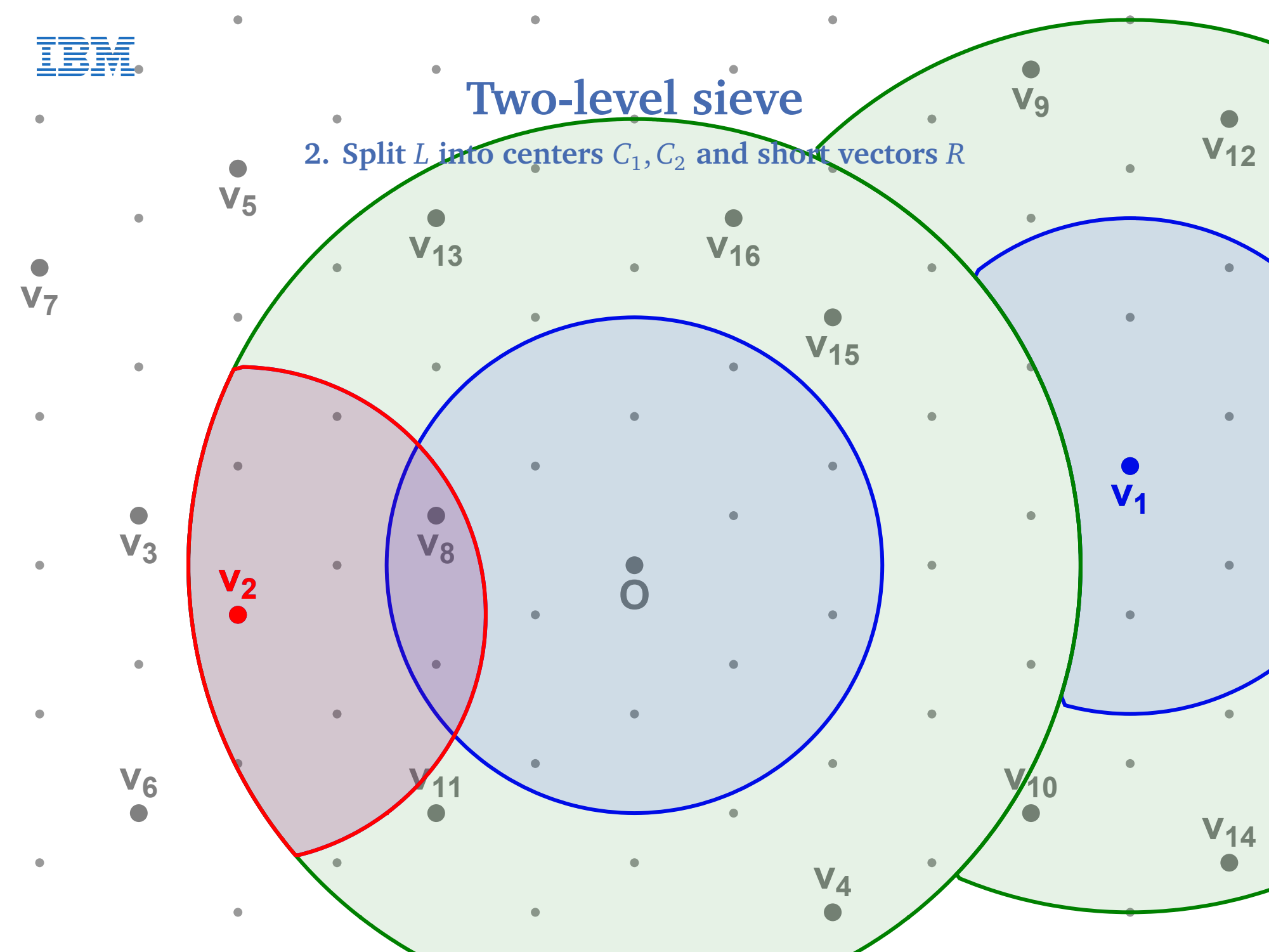
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



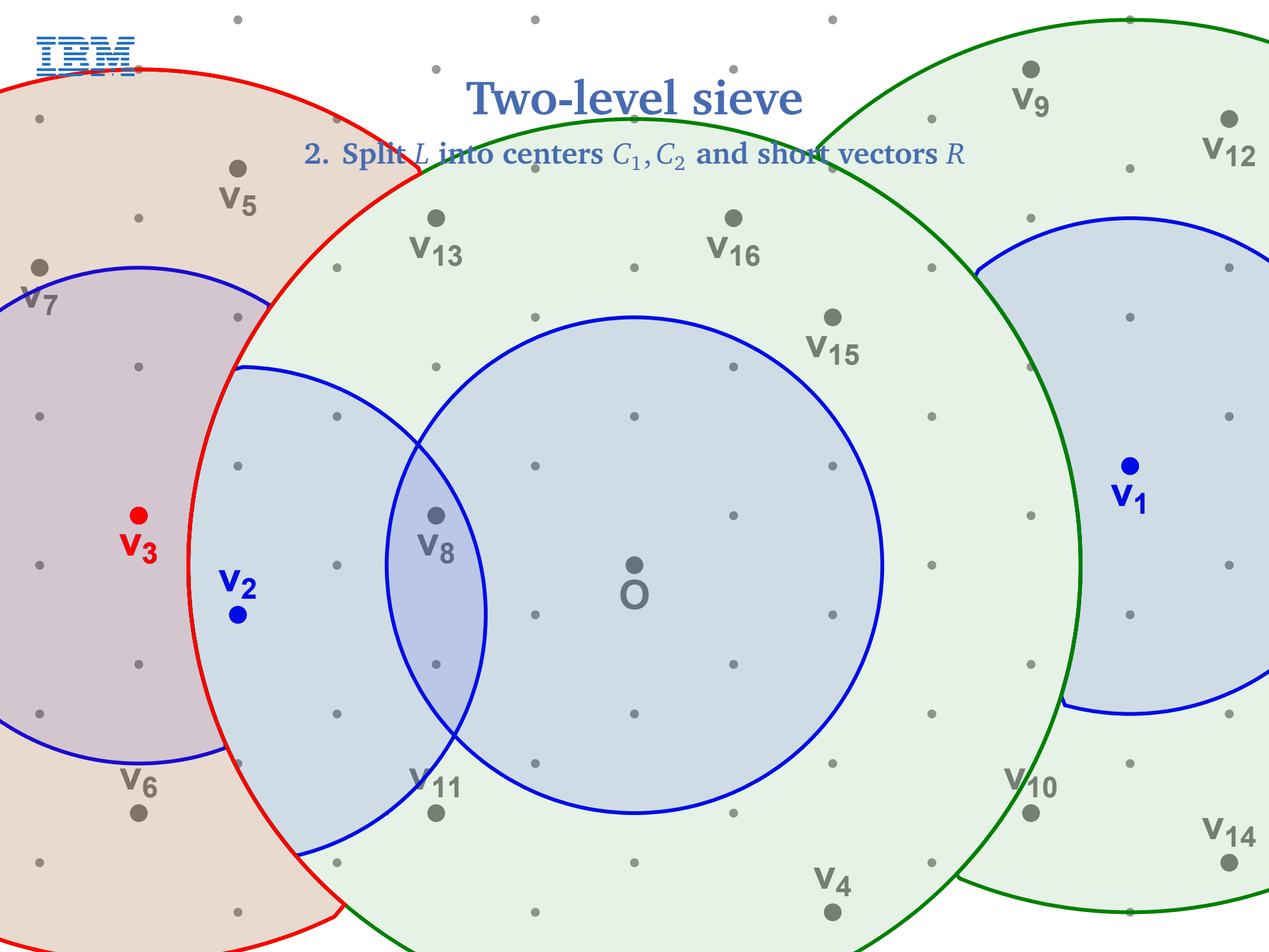
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



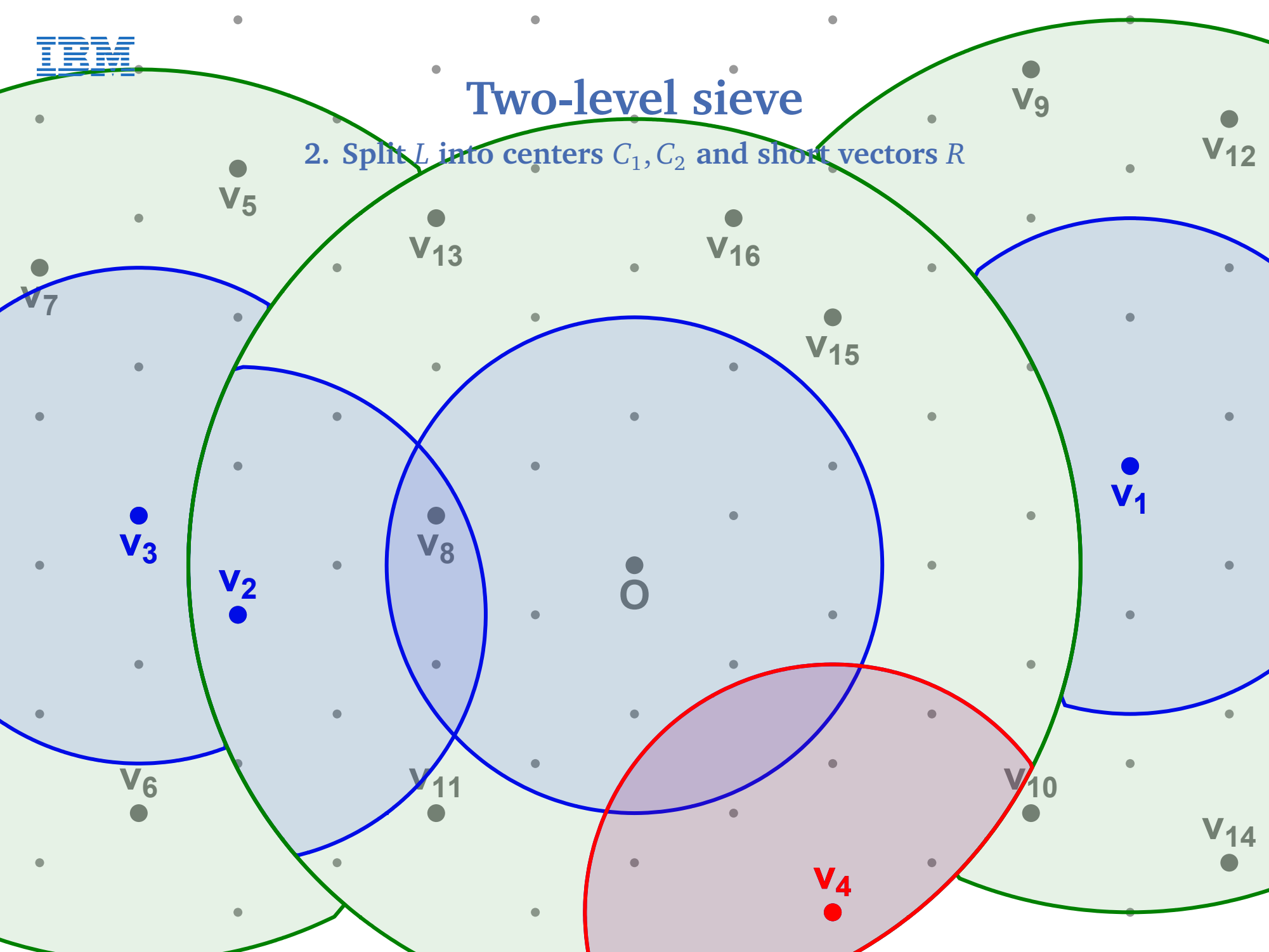
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



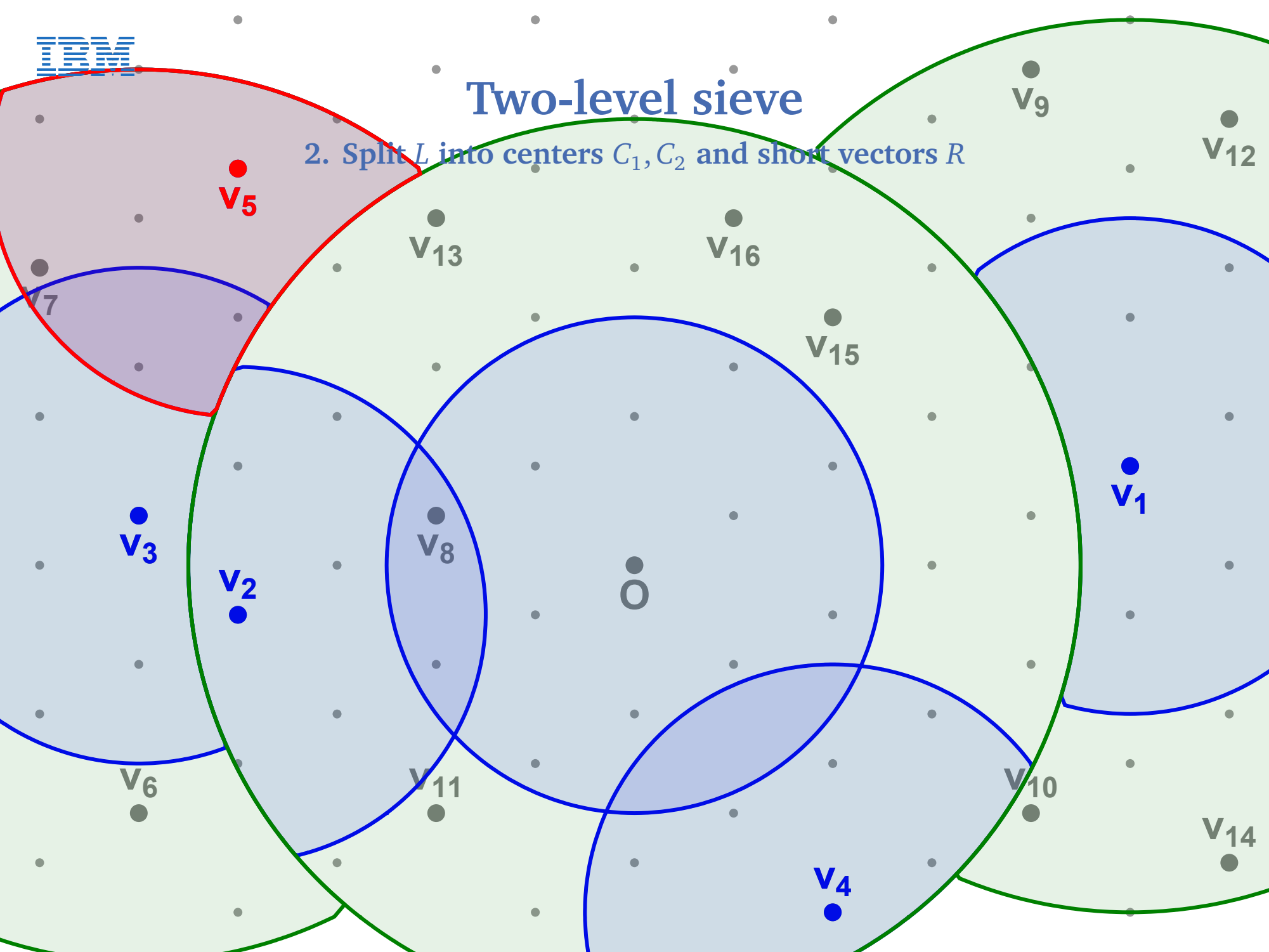
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



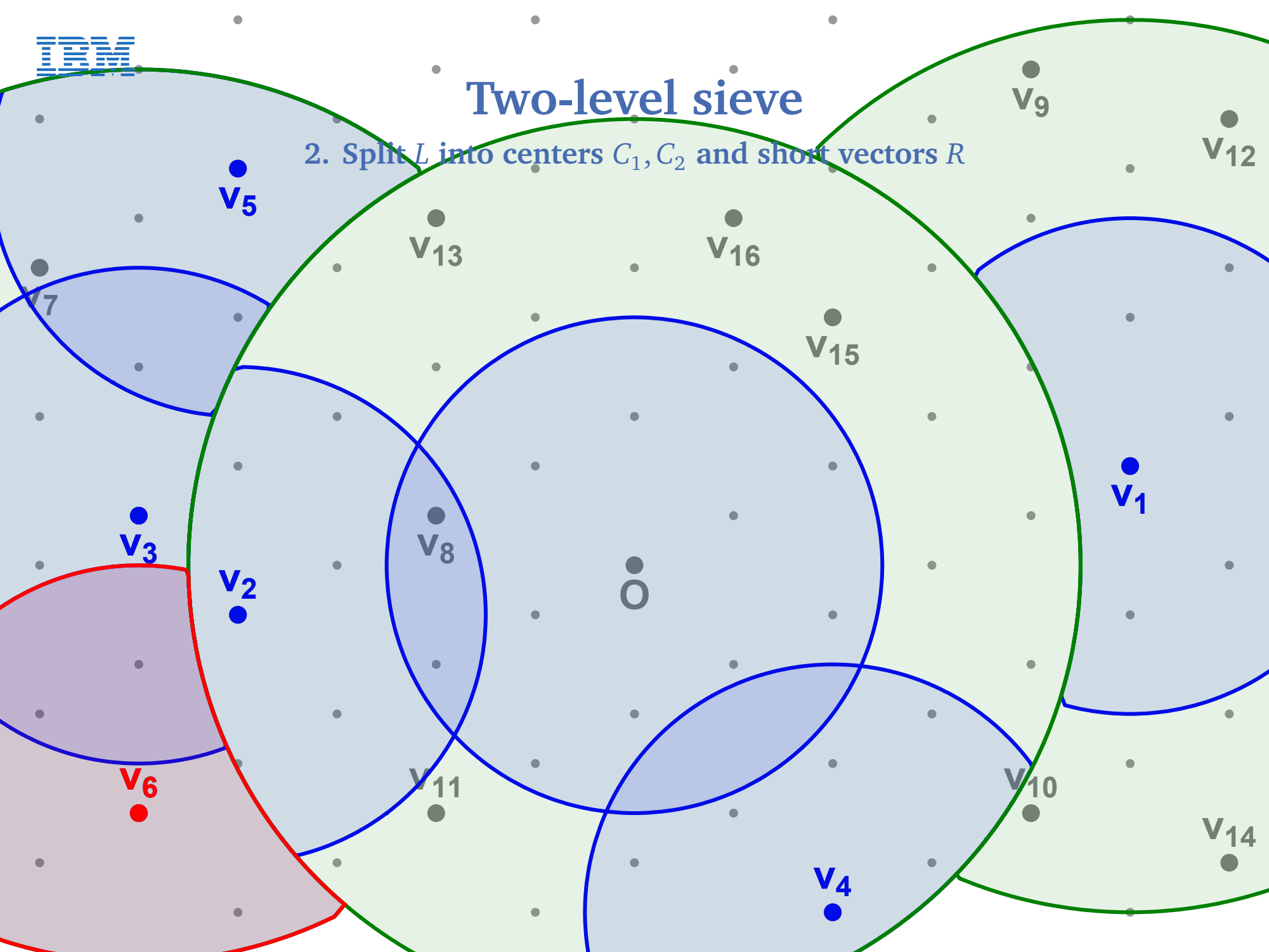
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



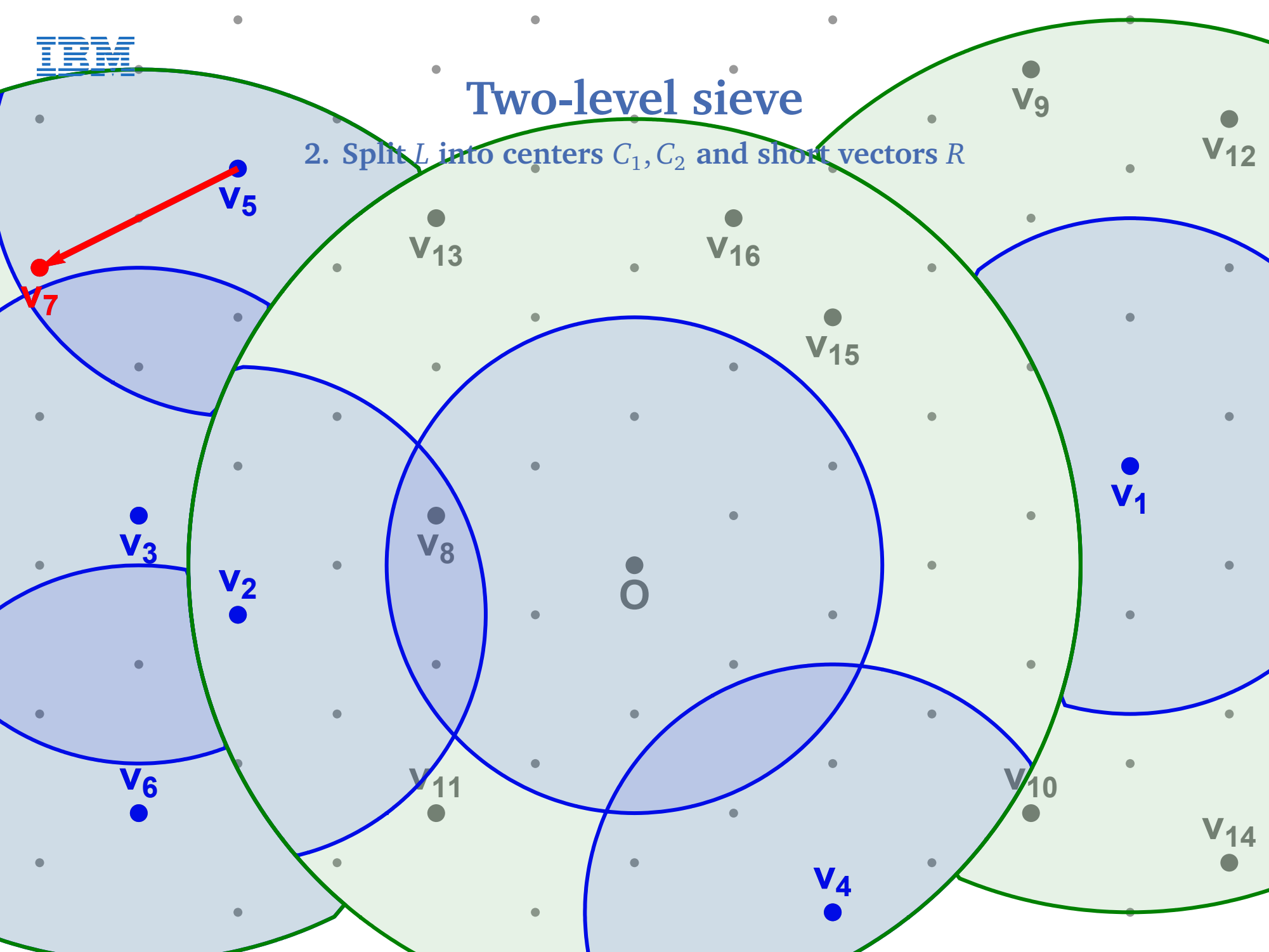
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



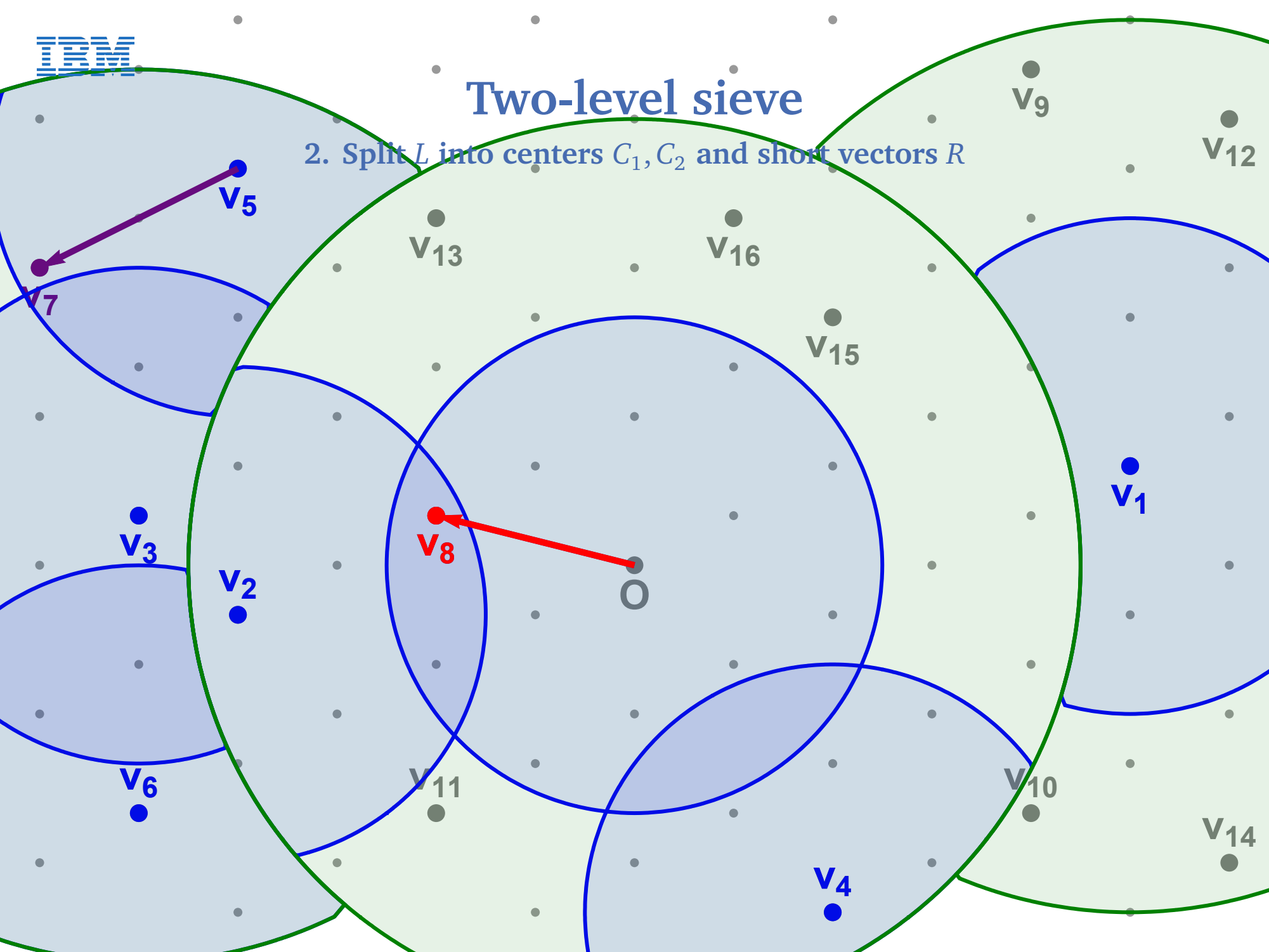
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



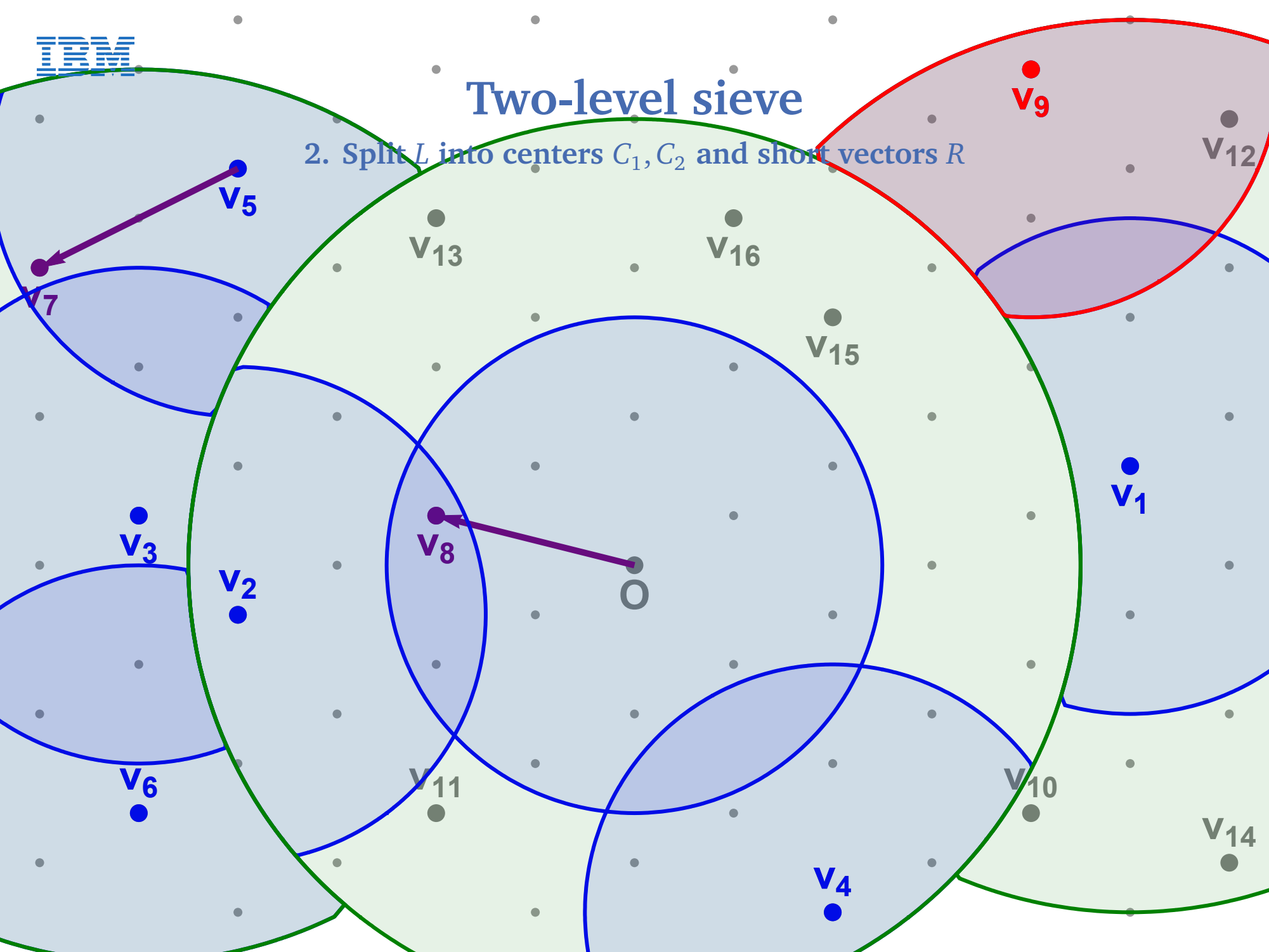
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



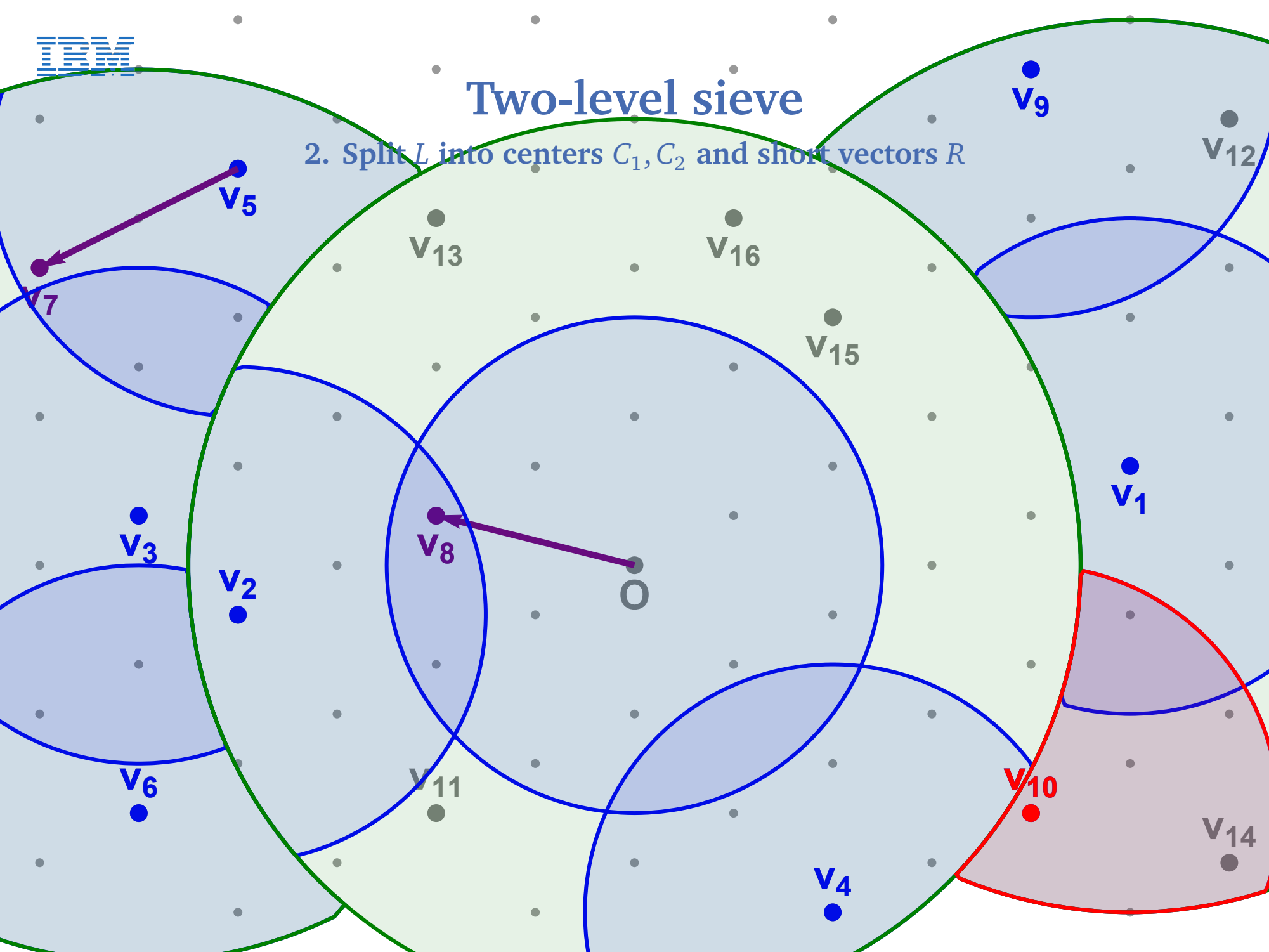
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



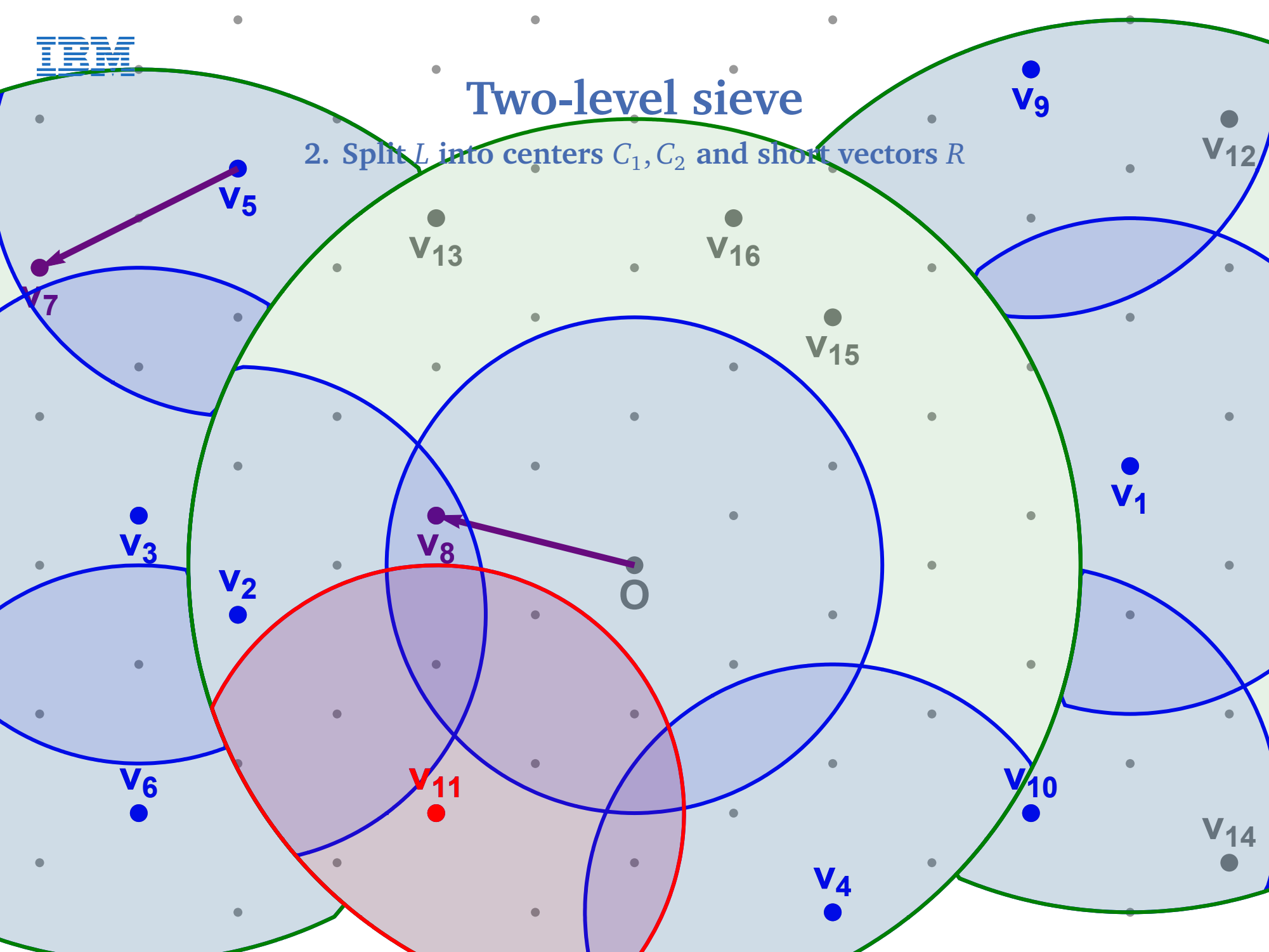
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



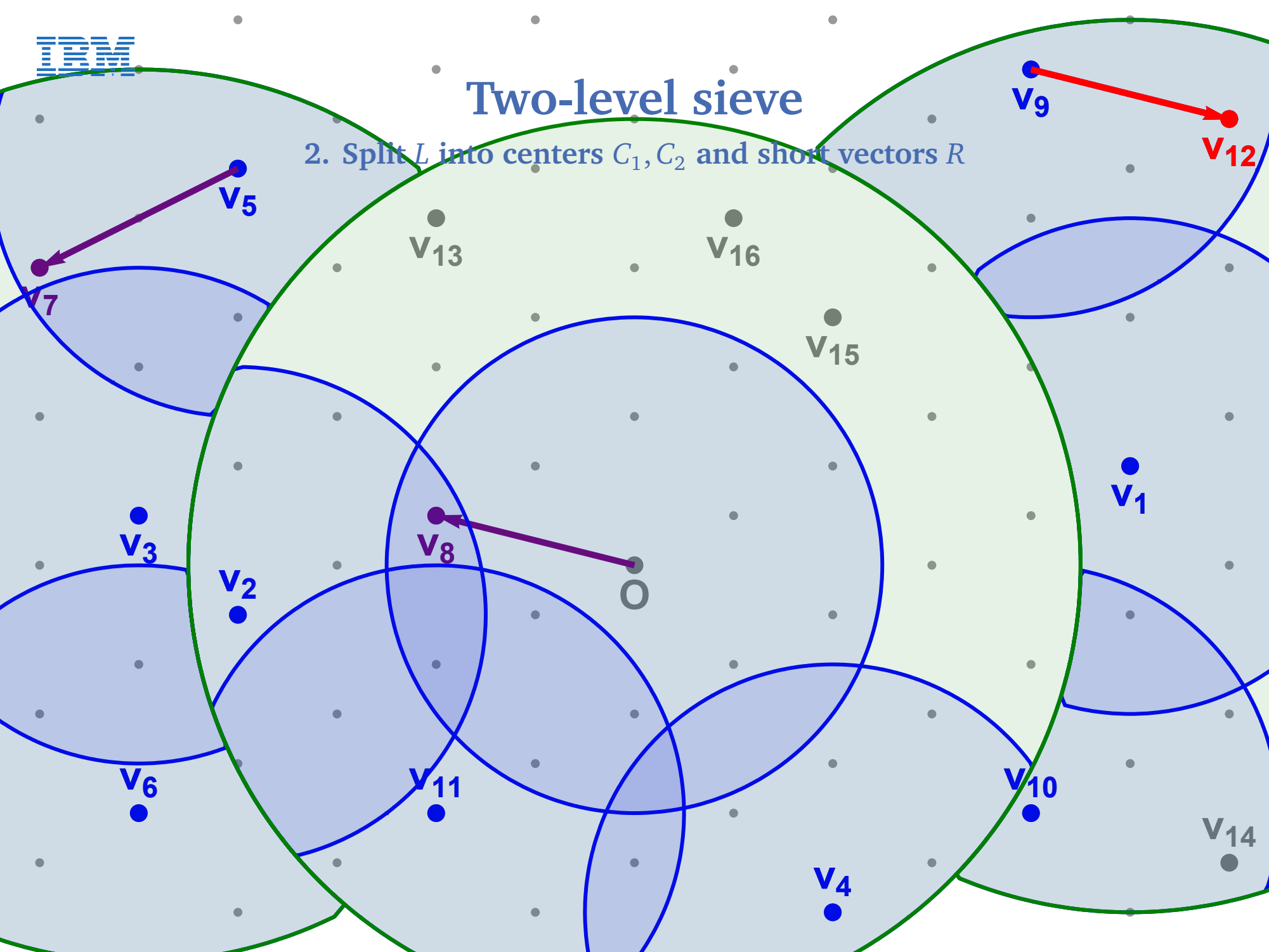
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



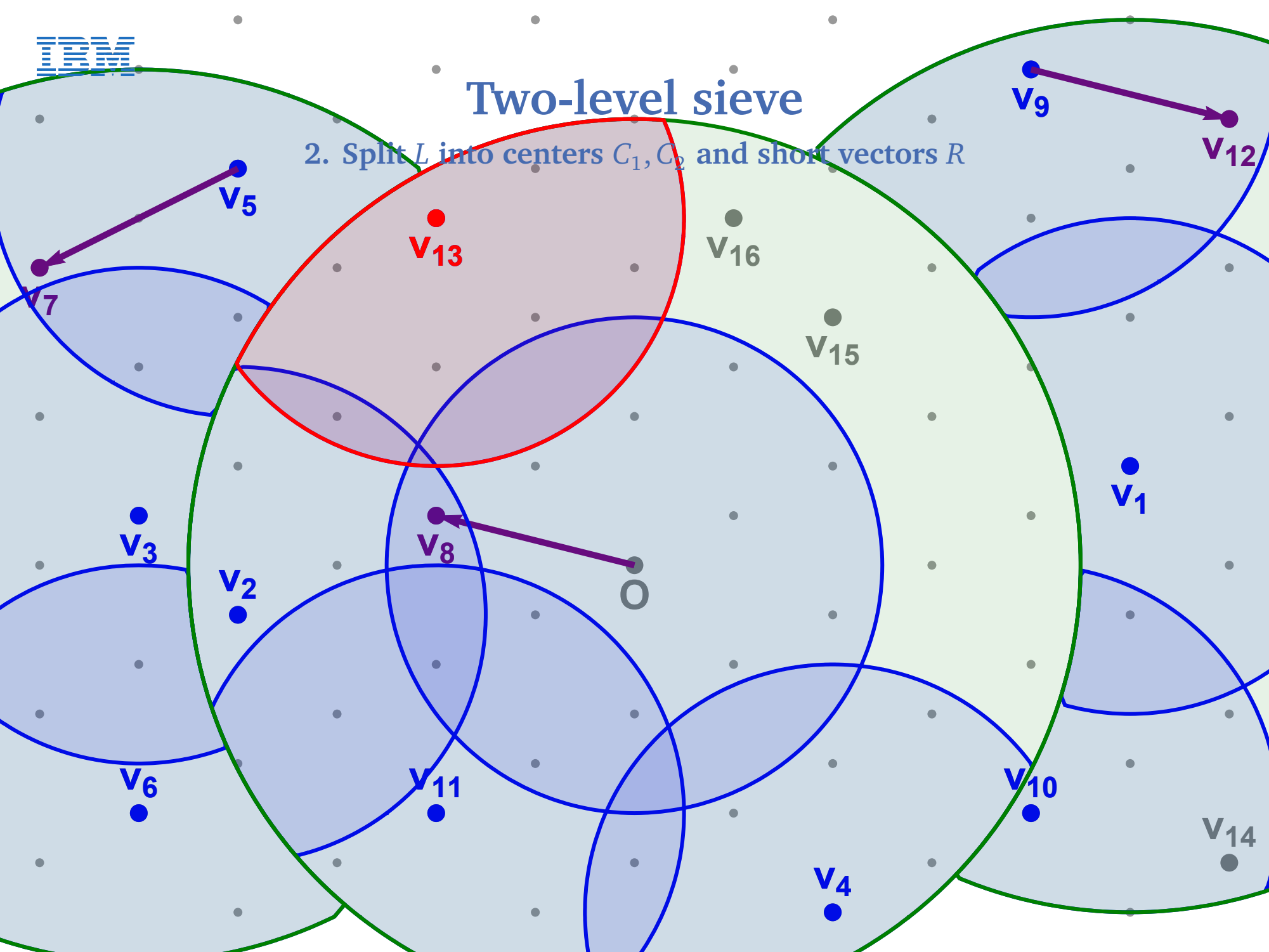
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



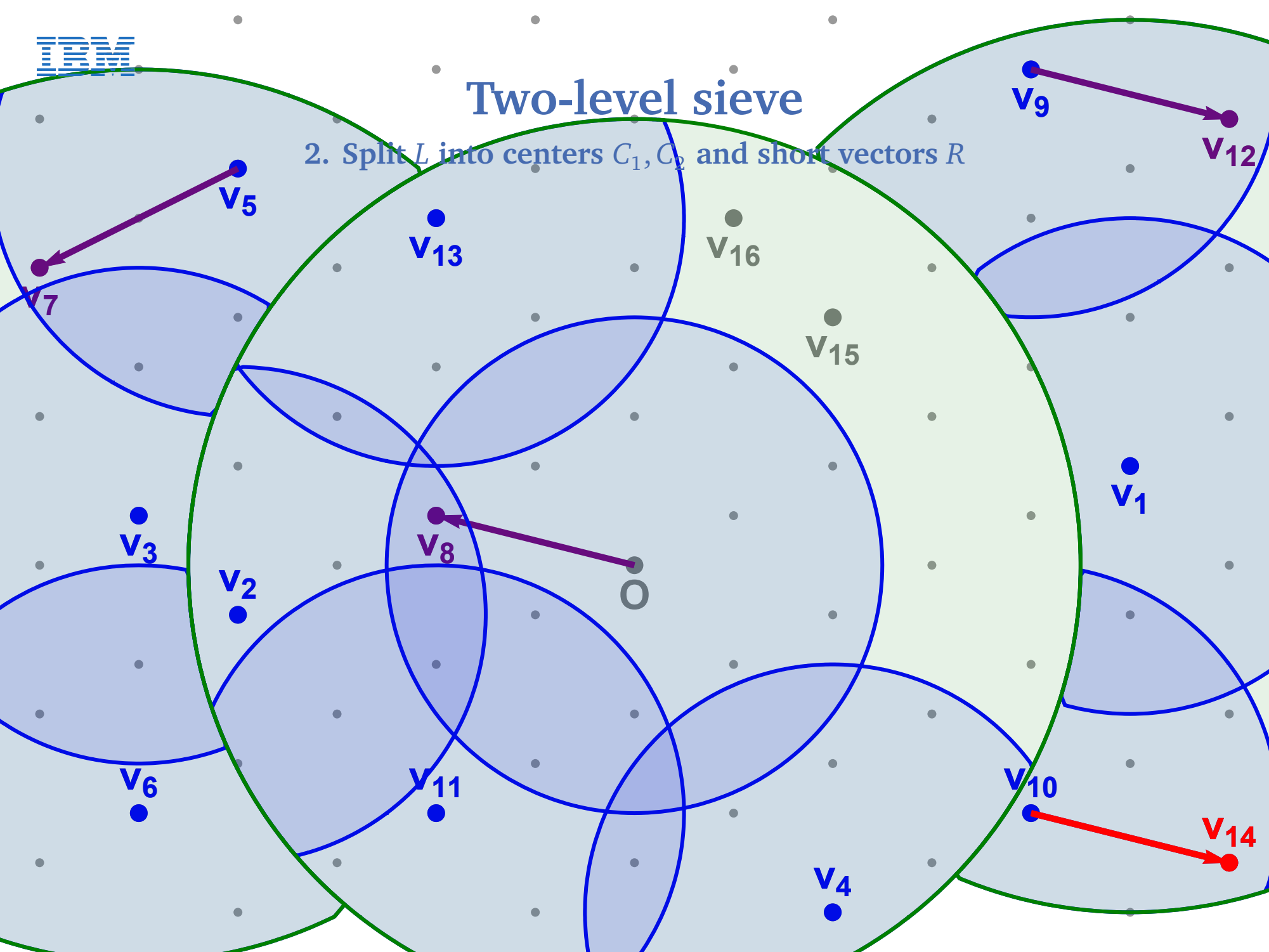
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



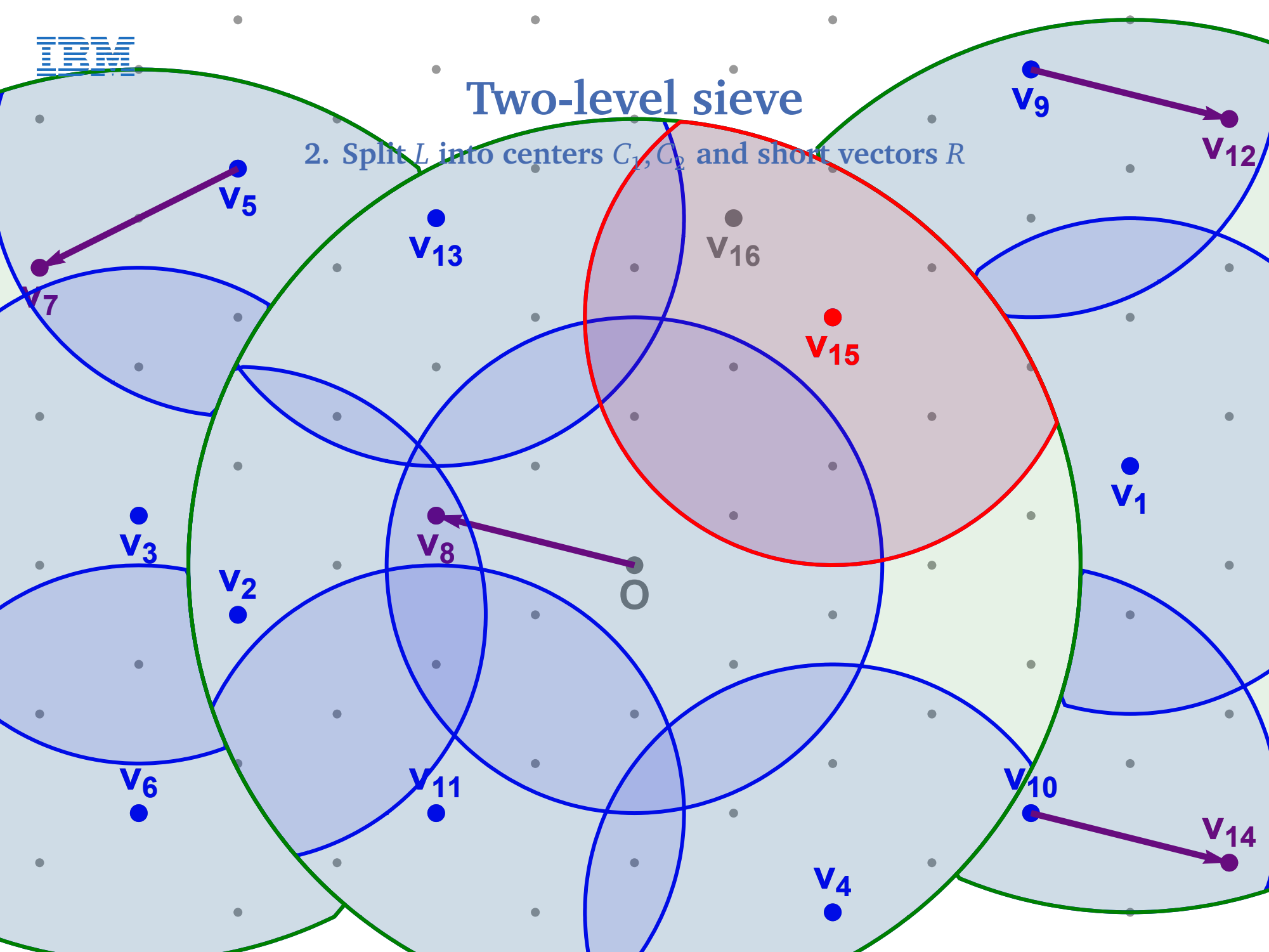
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



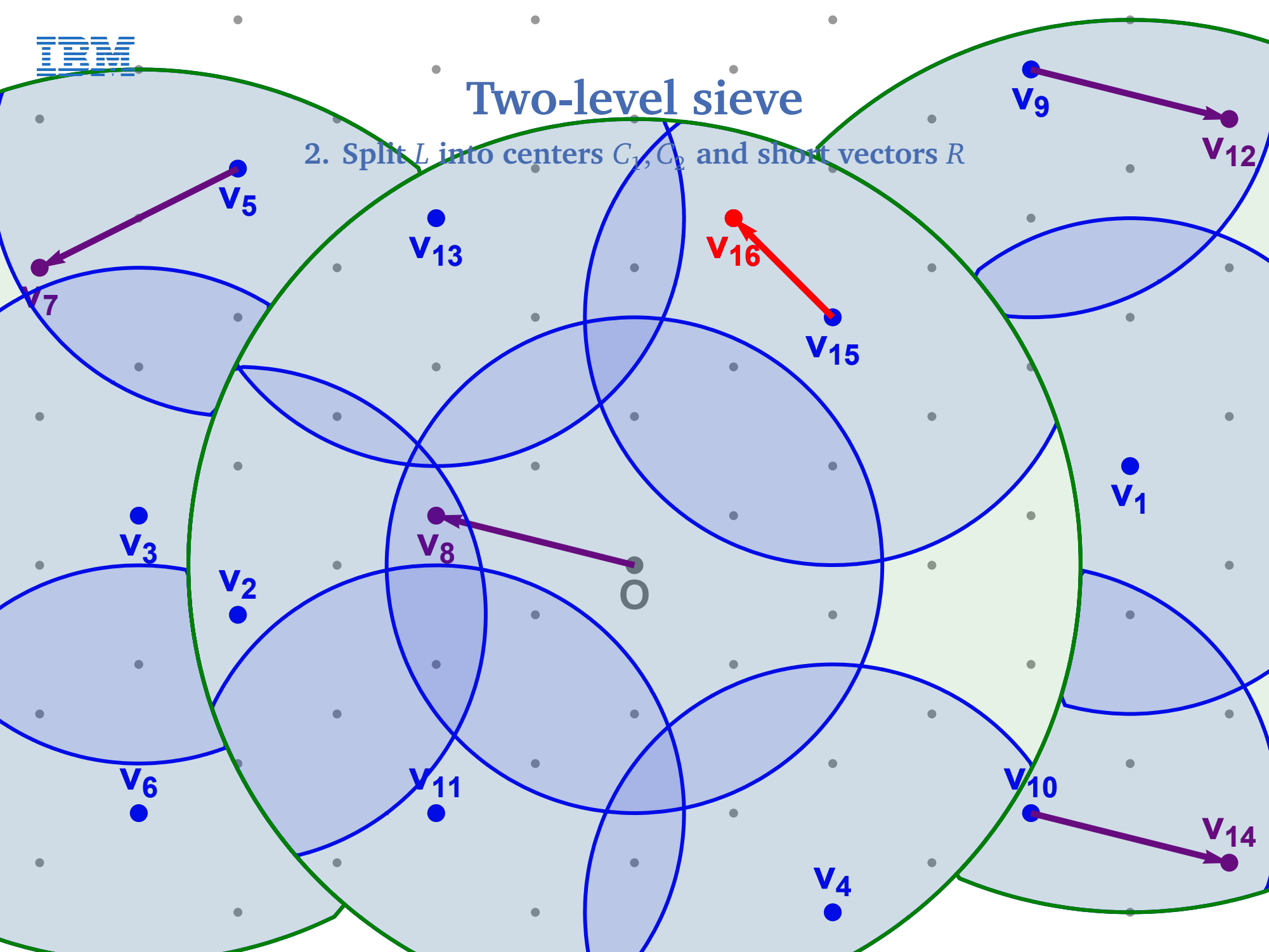
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



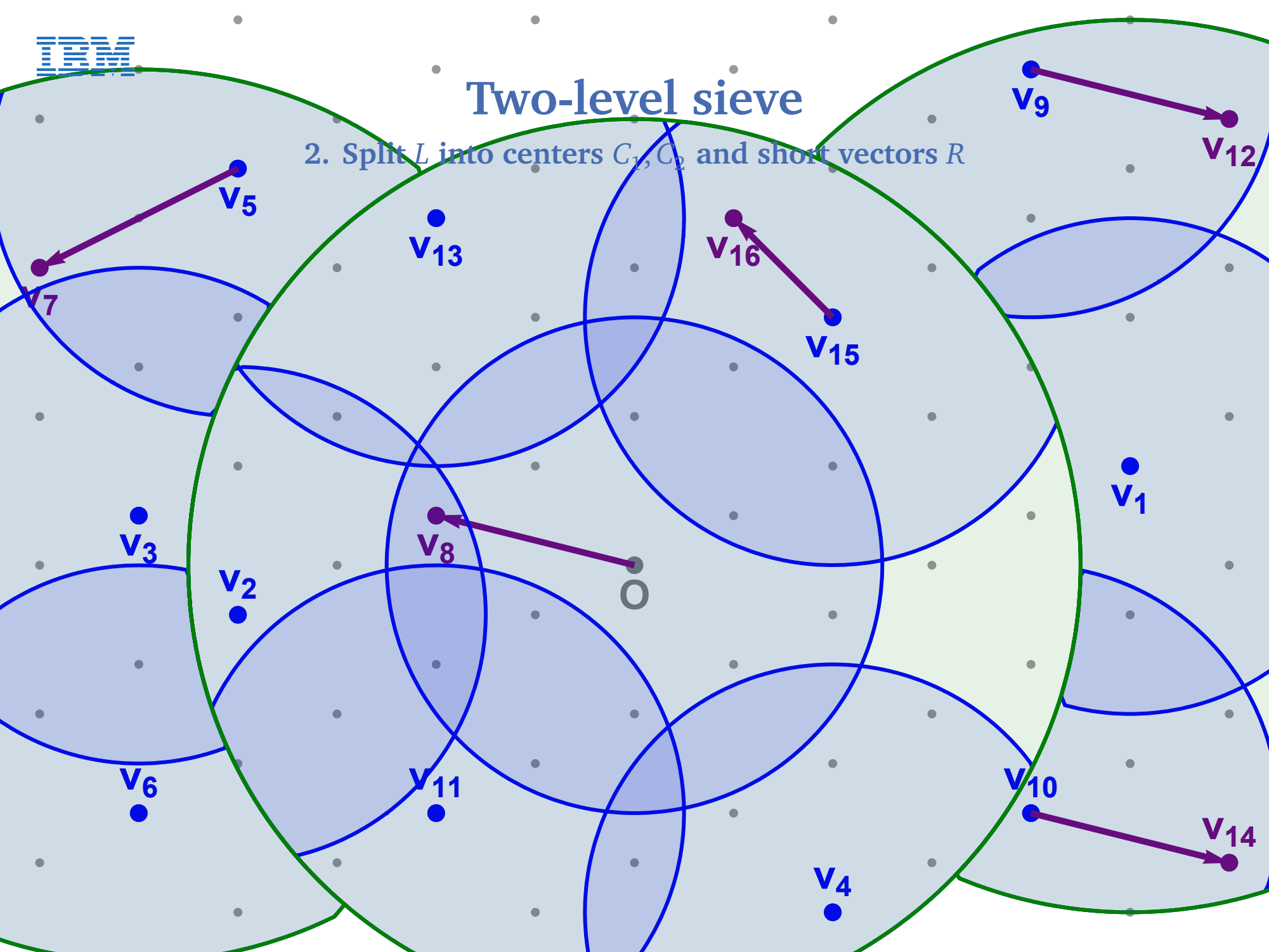
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



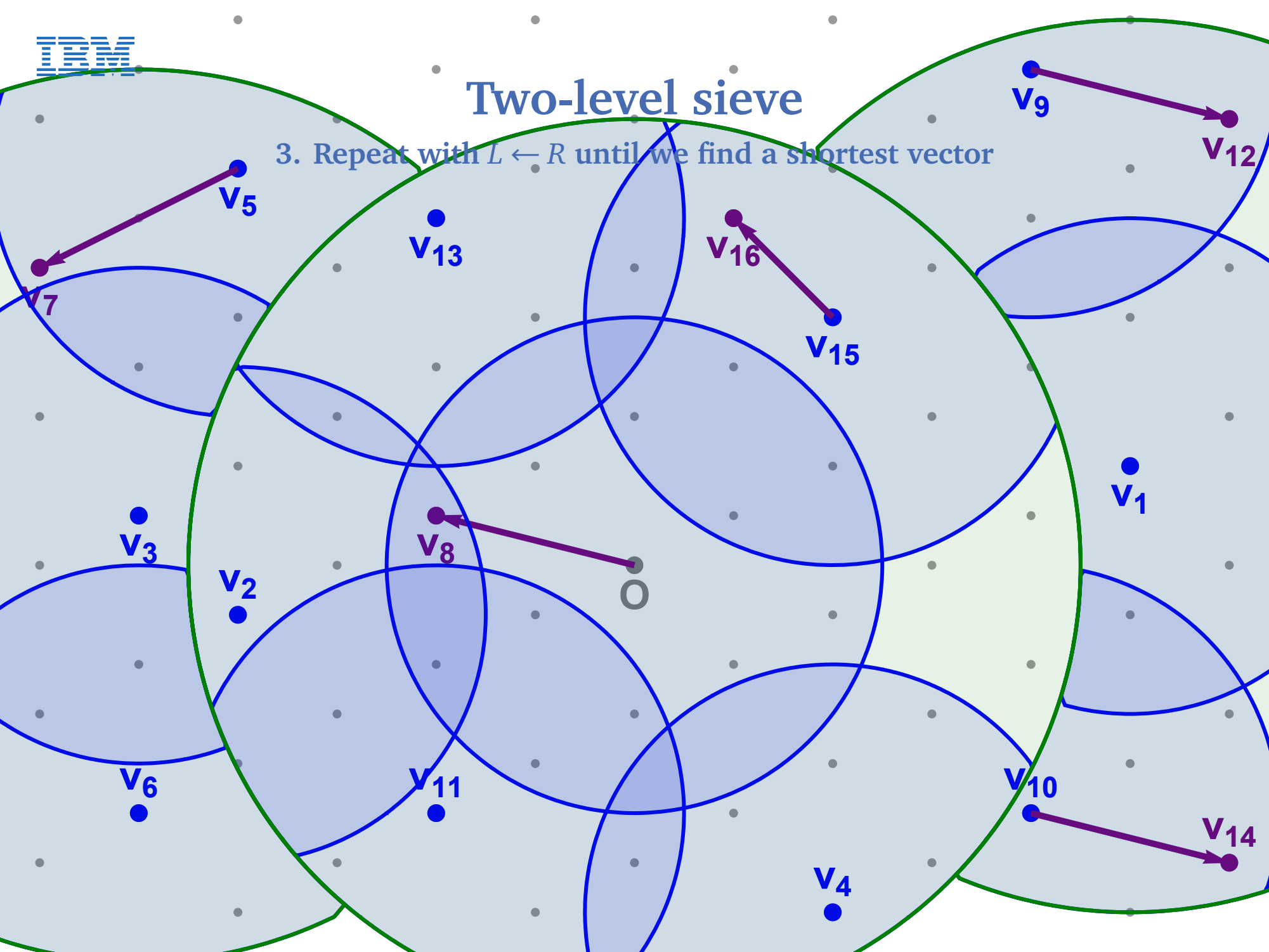
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



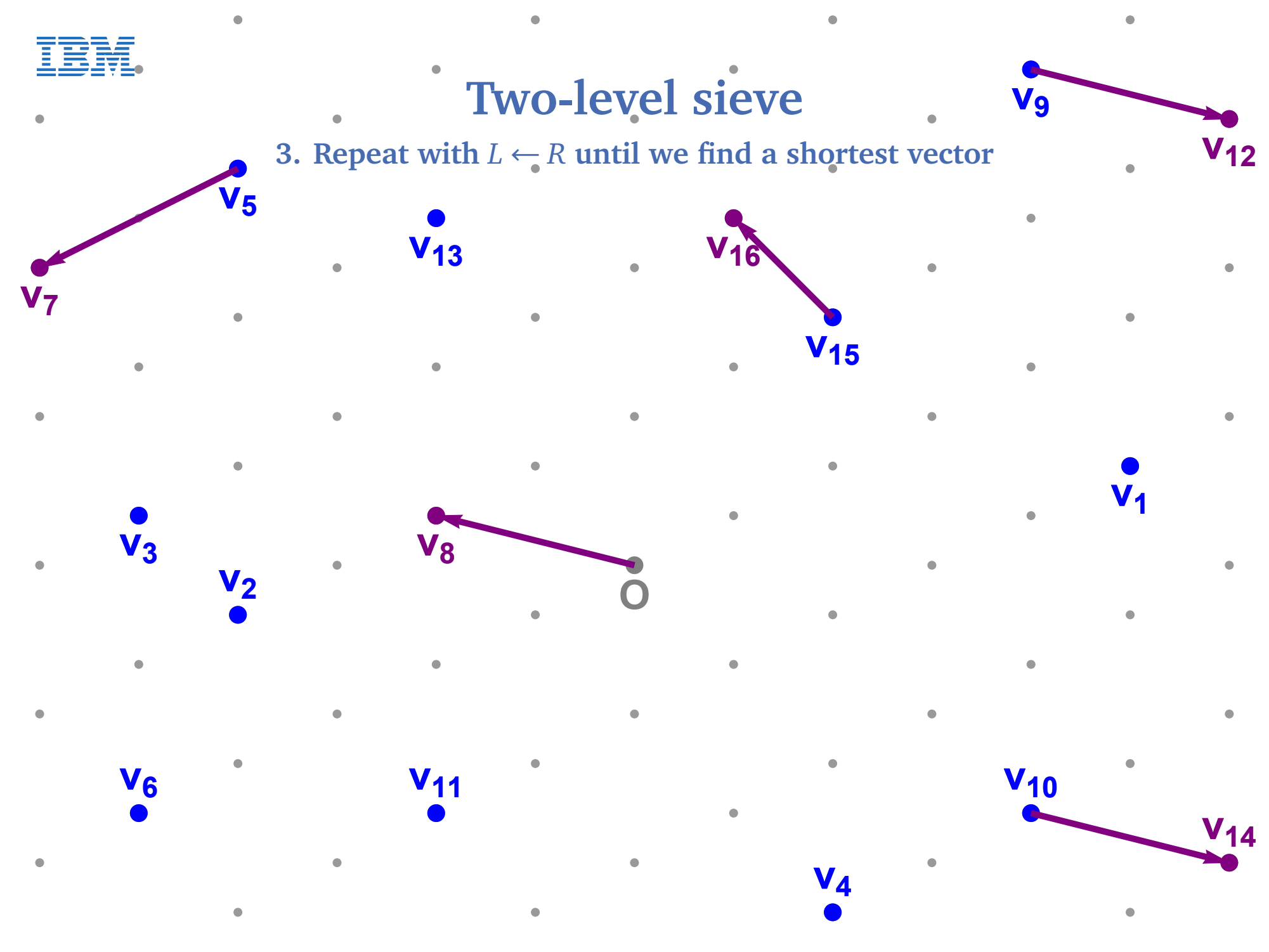
Two-level sieve

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



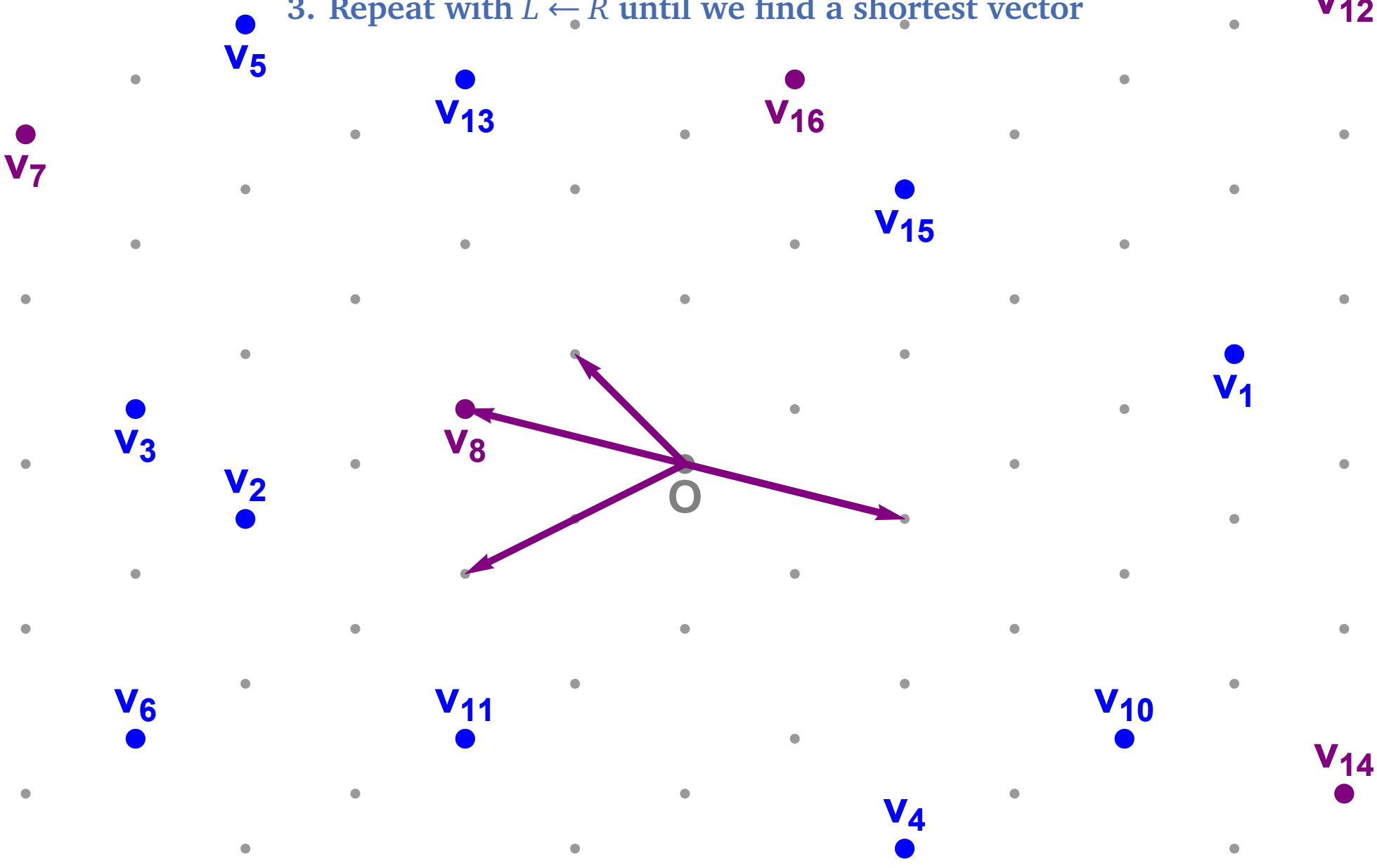
Two-level sieve

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



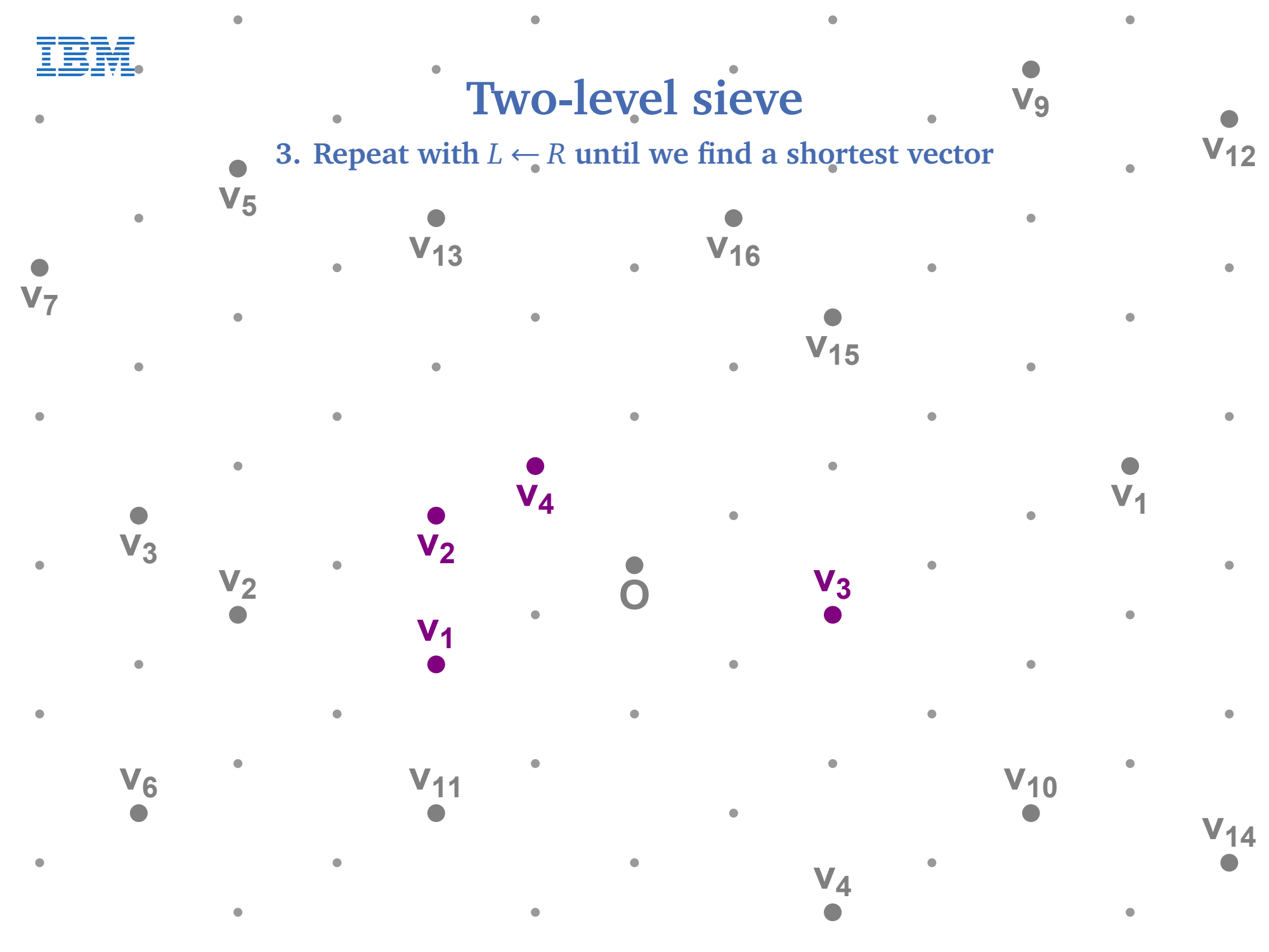
Two-level sieve

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



Two-level sieve

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



Multiple levels

Overview

Multiple levels

Overview

Heuristic (Nguyen and Vidick, J. Math. Crypt. '08)

The one-level sieve runs in time $2^{0.4150n}$ *and space* $2^{0.2075n}$.

Multiple levels

Overview

Heuristic (Nguyen and Vidick, J. Math. Crypt. '08)

The one-level sieve runs in time $2^{0.4150n}$ *and space* $2^{0.2075n}$.

Heuristic (Wang et al., ASIACCS'11)

The two-level sieve runs in time $2^{0.3836n}$ *and space* $2^{0.2557n}$.

Multiple levels

Overview

Heuristic (Nguyen and Vidick, J. Math. Crypt. '08)

The one-level sieve runs in time $2^{0.4150n}$ *and space* $2^{0.2075n}$.

Heuristic (Wang et al., ASIACCS'11)

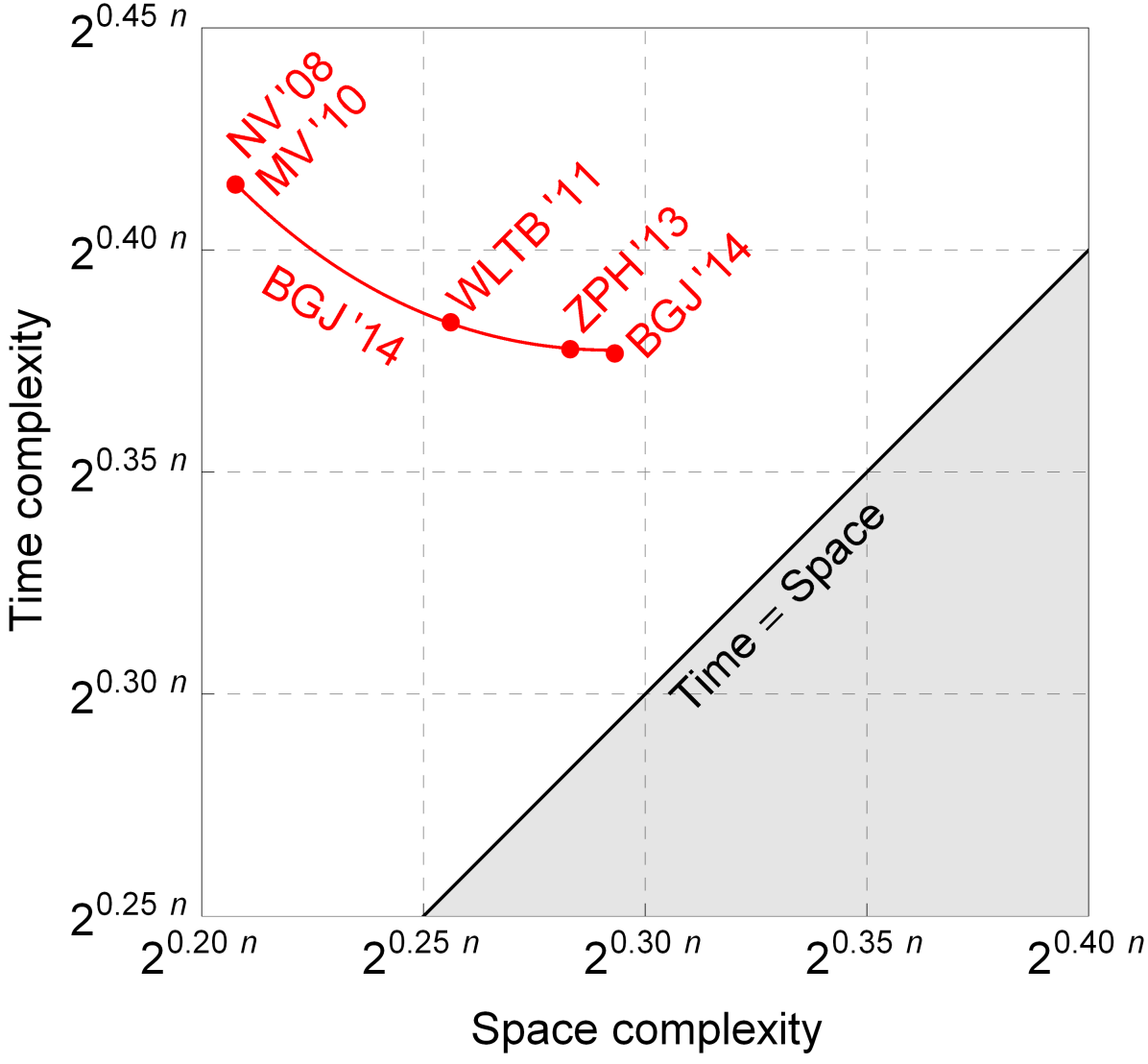
The two-level sieve runs in time $2^{0.3836n}$ *and space* $2^{0.2557n}$.

Heuristic (Zhang et al., SAC'13)

The three-level sieve runs in time $2^{0.3778n}$ *and space* $2^{0.2833n}$.

Sieving

Space/time trade-off

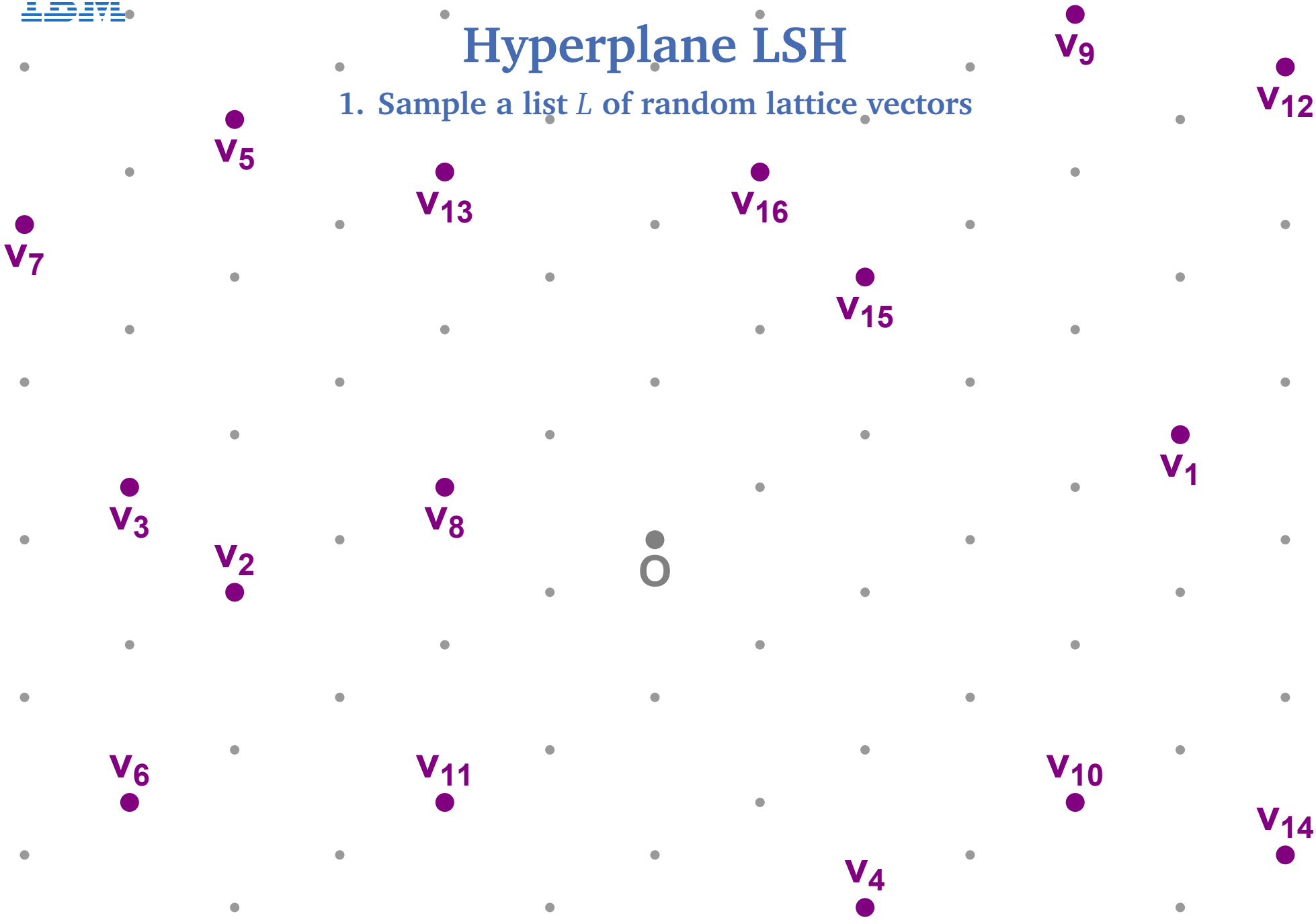


Hyperplane LSH

1. Sample a list L of random lattice vectors

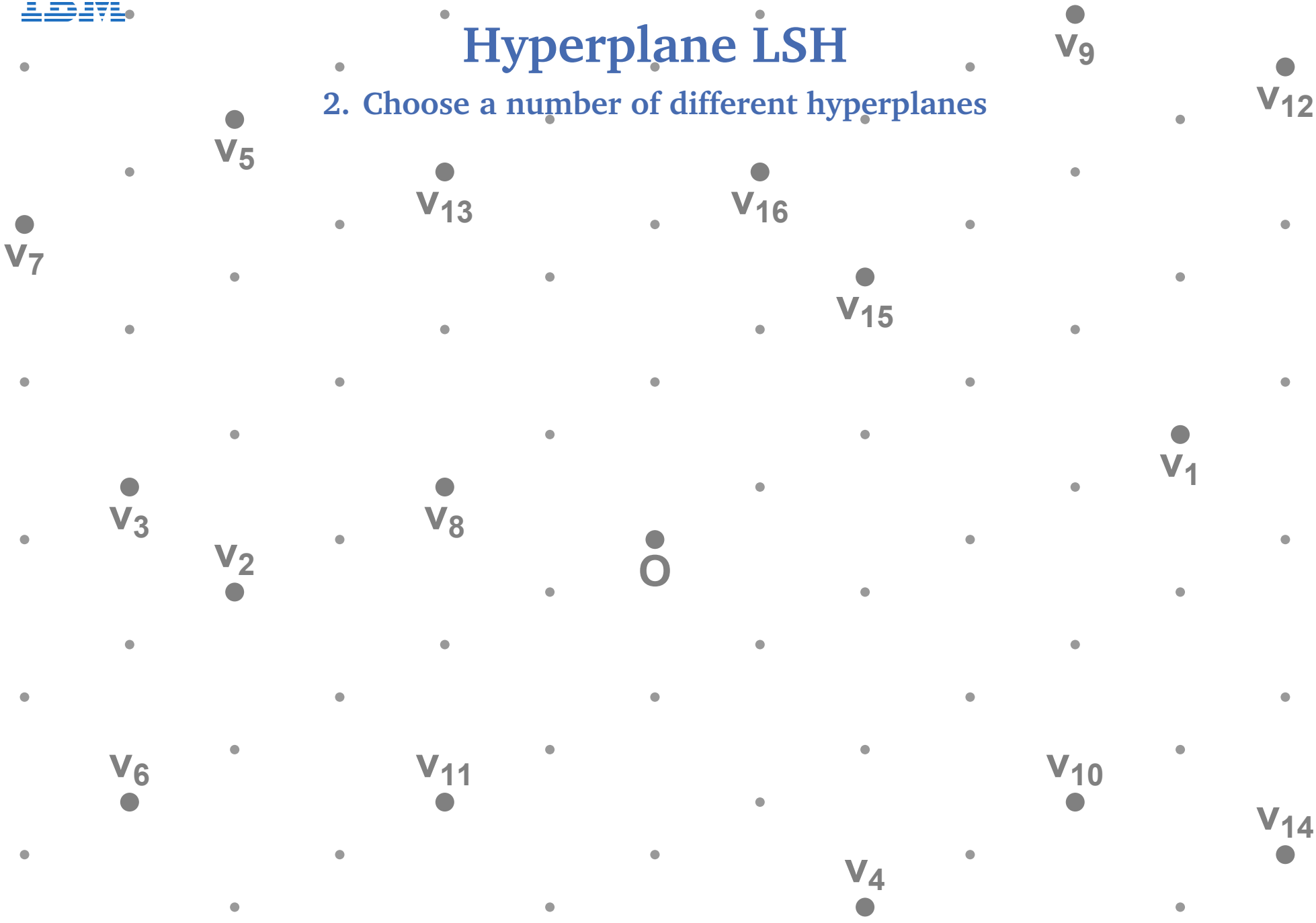
Hyperplane LSH

1. Sample a list L of random lattice vectors



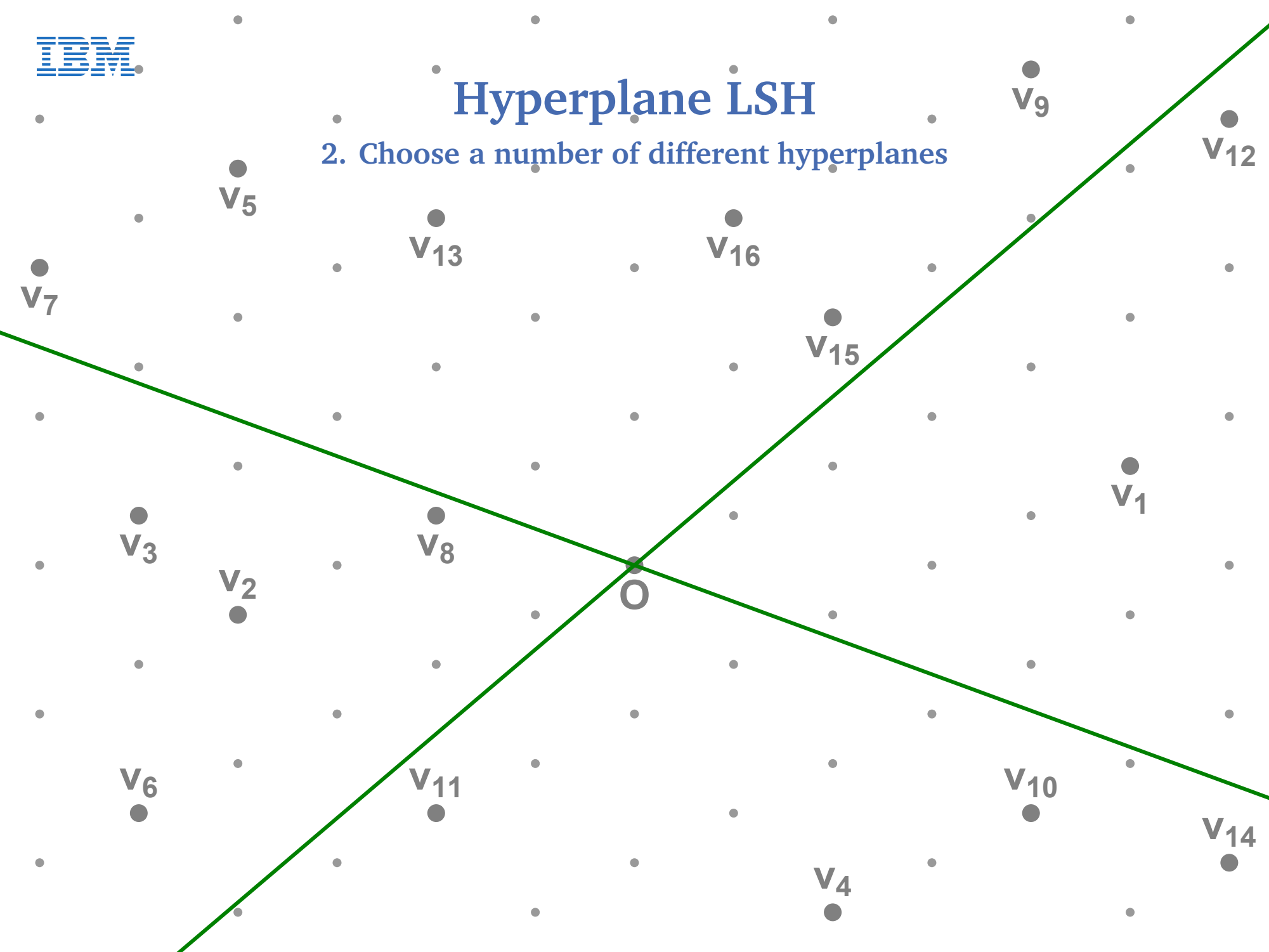
Hyperplane LSH

2. Choose a number of different hyperplanes



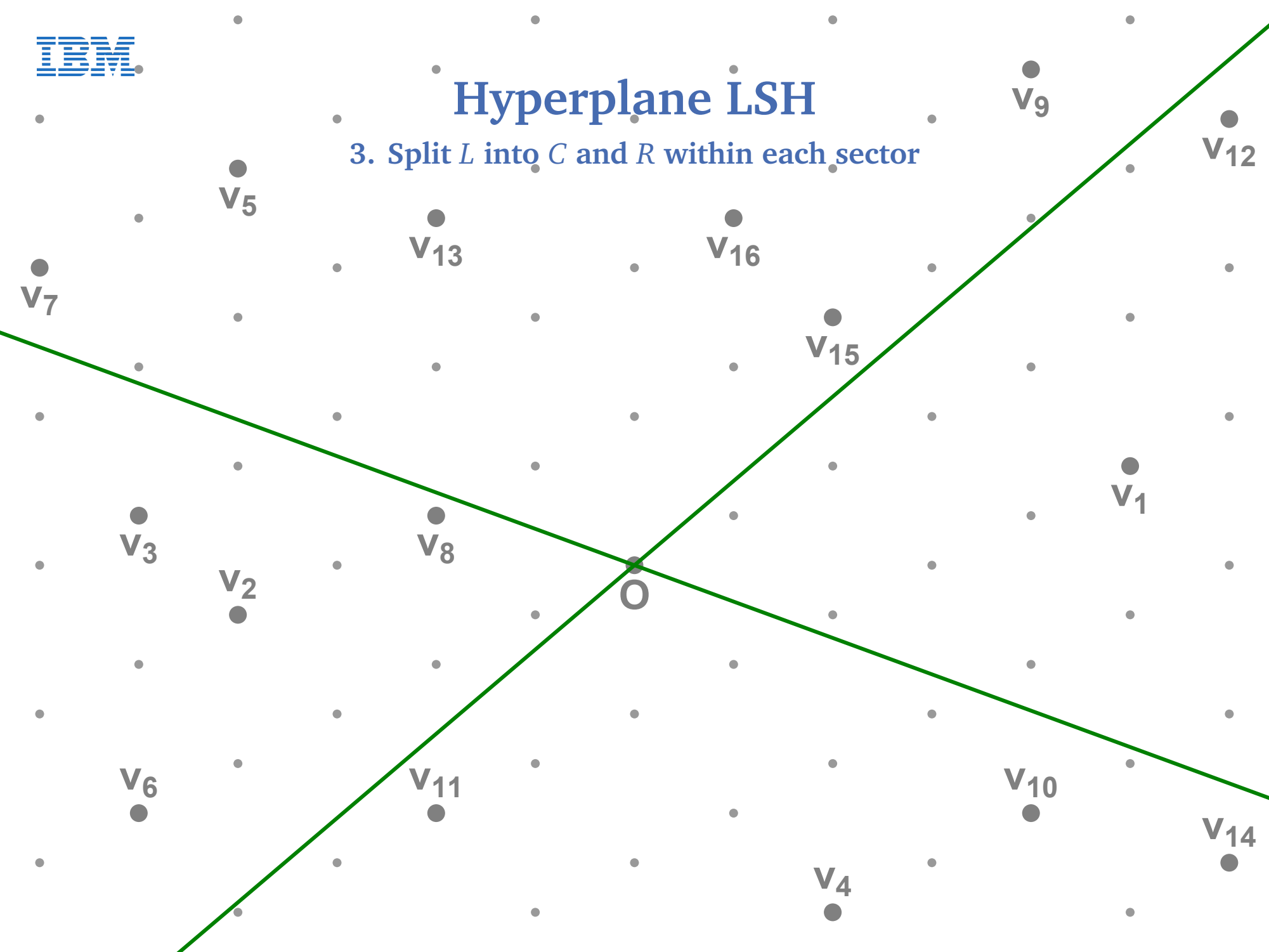
Hyperplane LSH

2. Choose a number of different hyperplanes



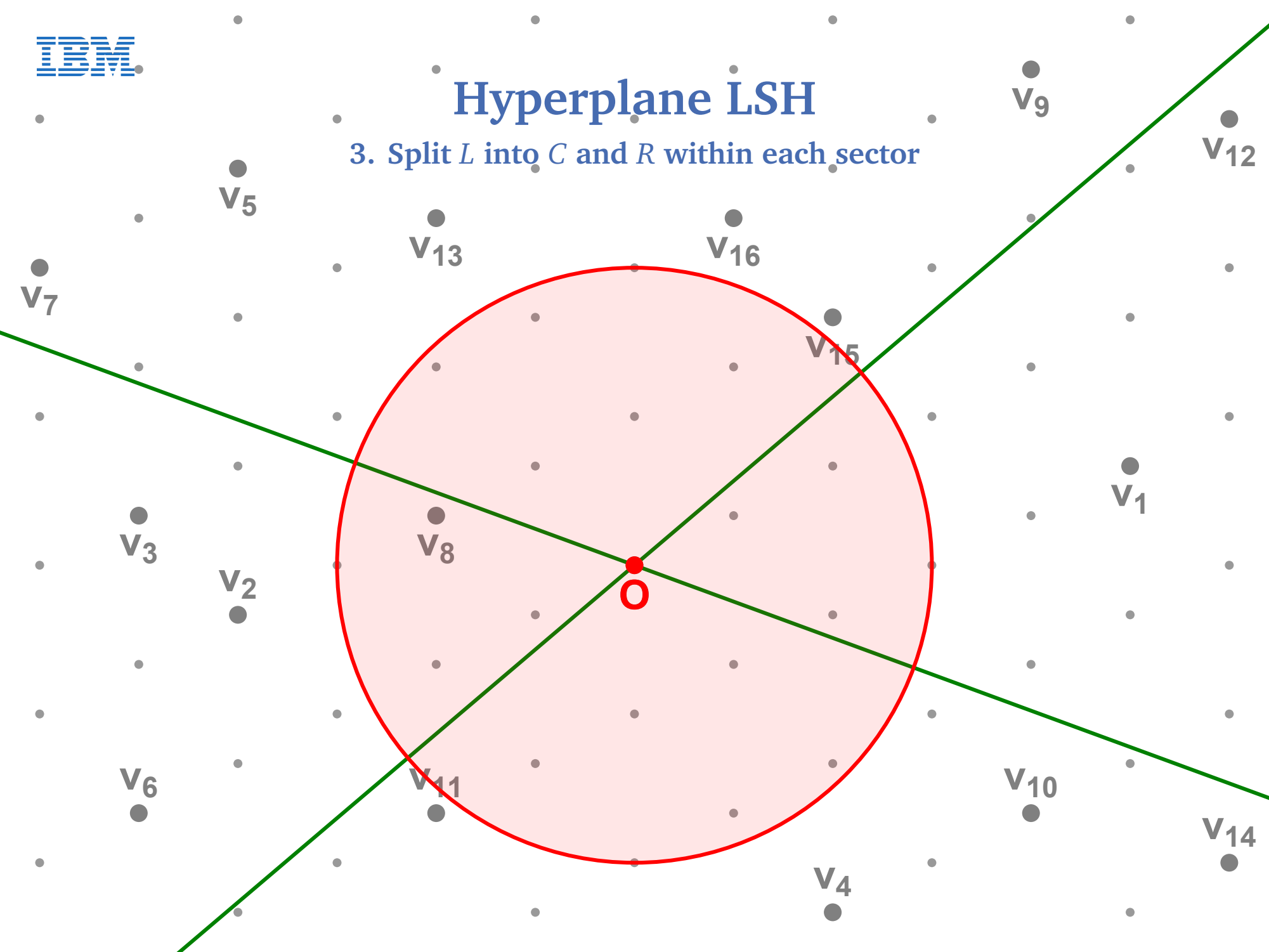
Hyperplane LSH

3. Split L into C and R within each sector



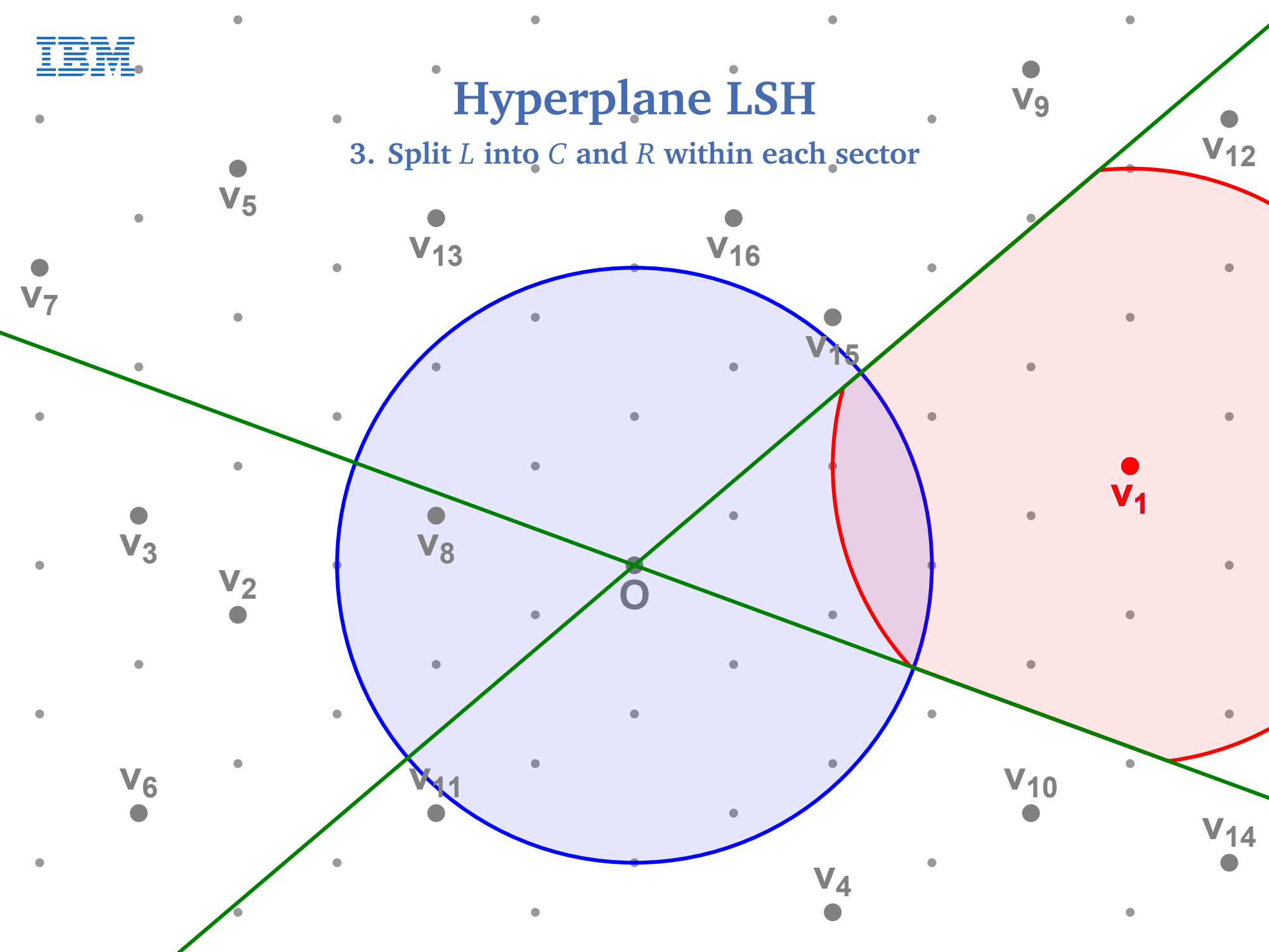
Hyperplane LSH

3. Split L into C and R within each sector



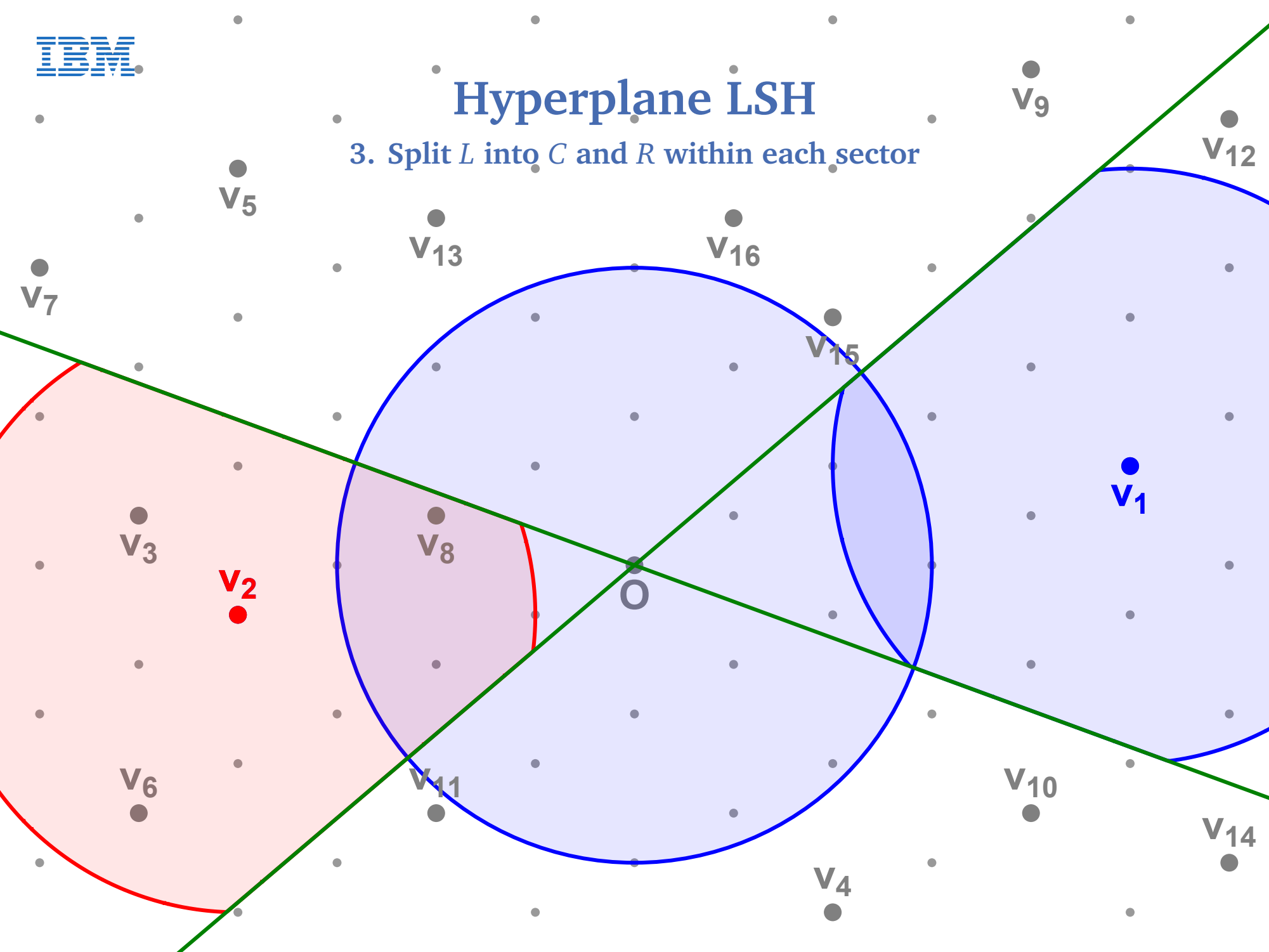
Hyperplane LSH

3. Split L into C and R within each sector



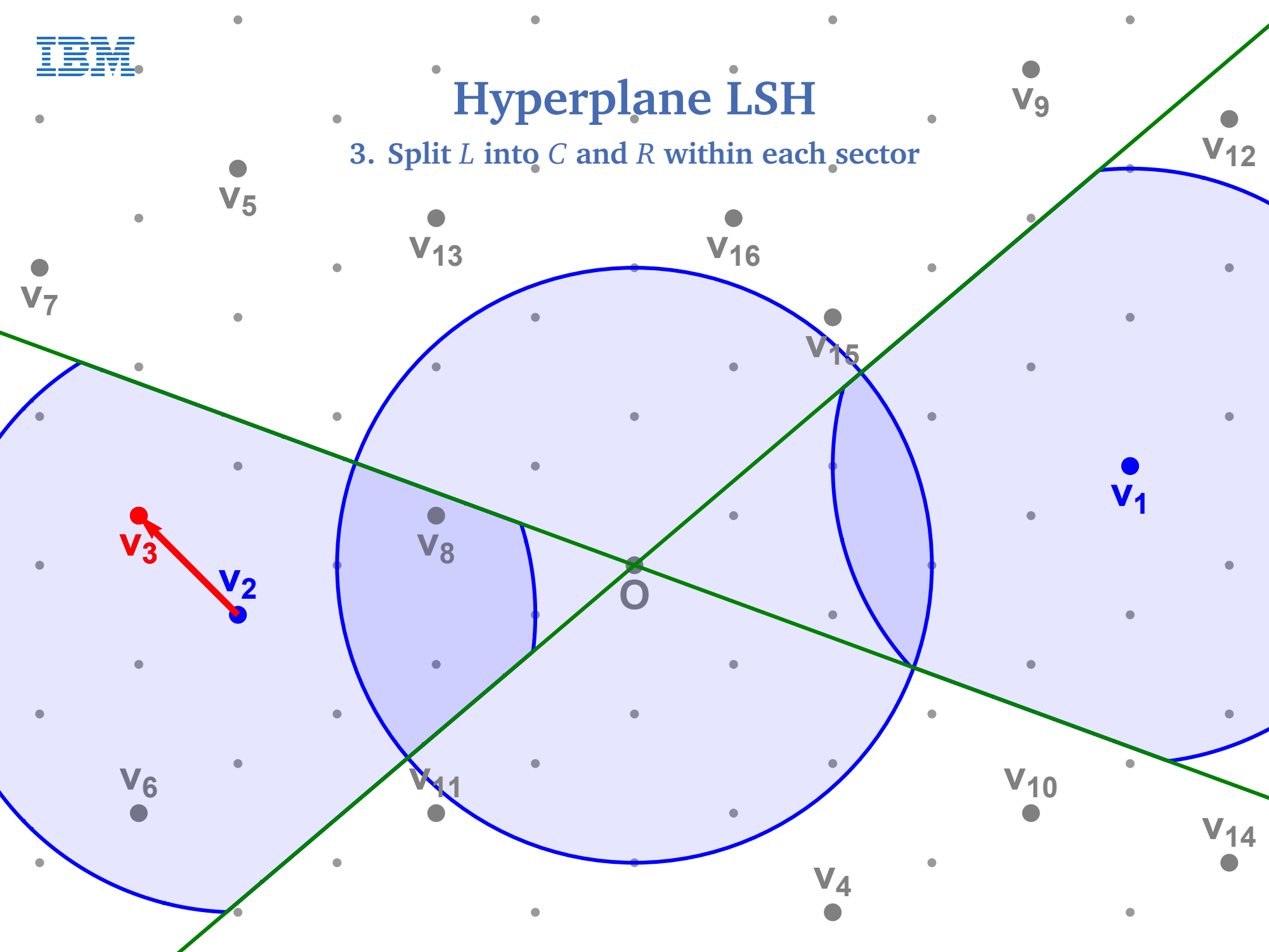
Hyperplane LSH

3. Split L into C and R within each sector



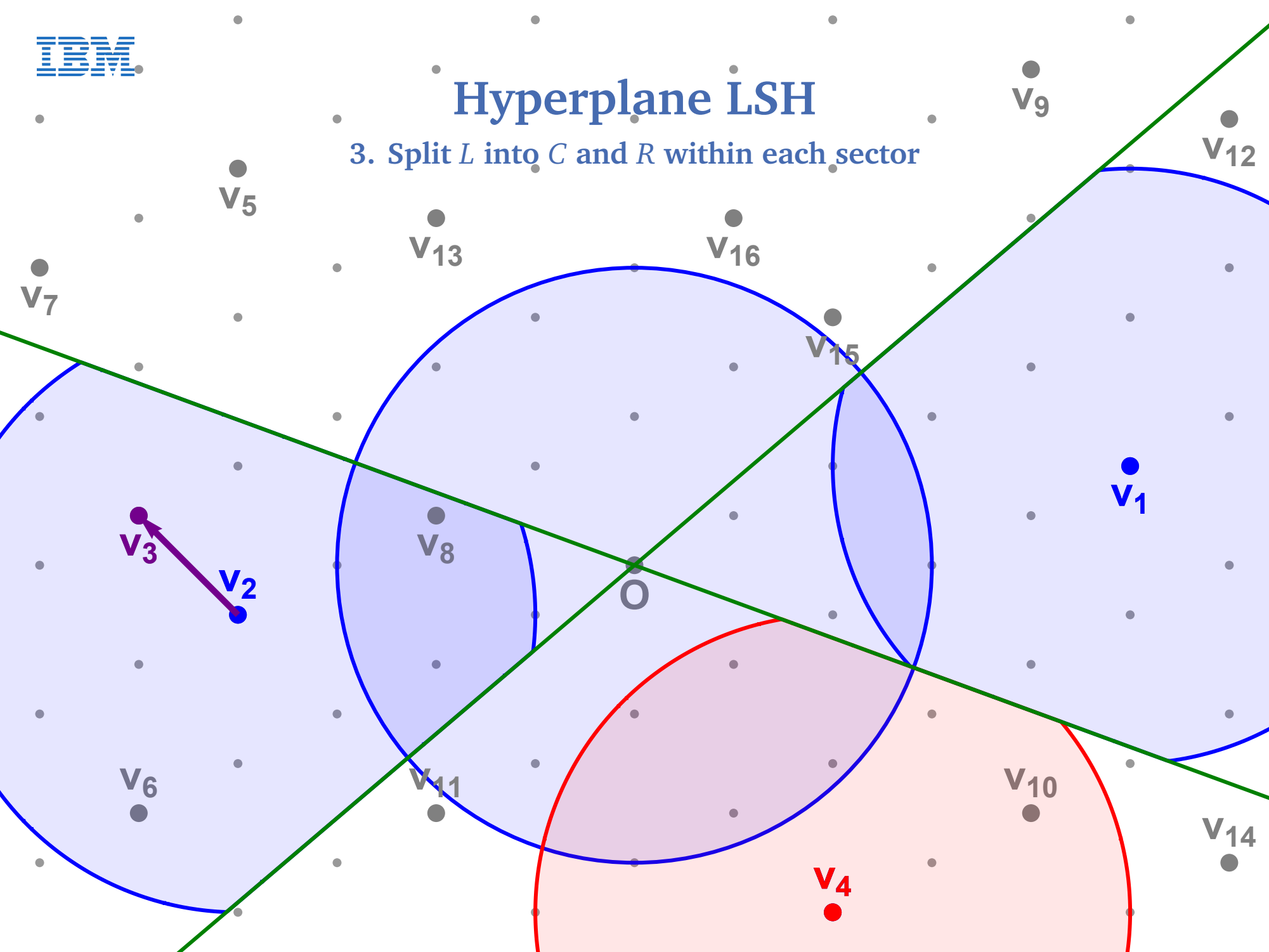
Hyperplane LSH

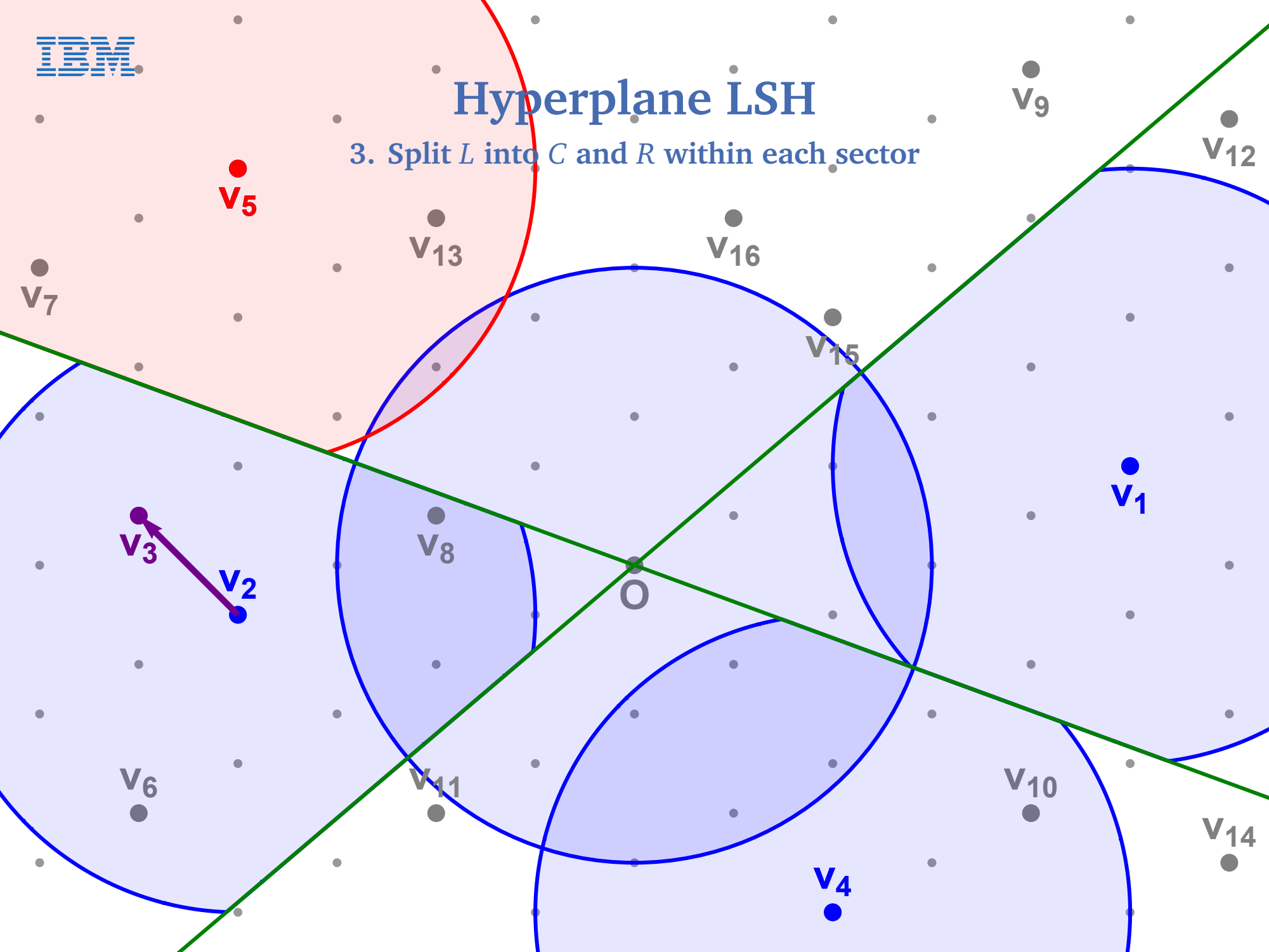
3. Split L into C and R within each sector



Hyperplane LSH

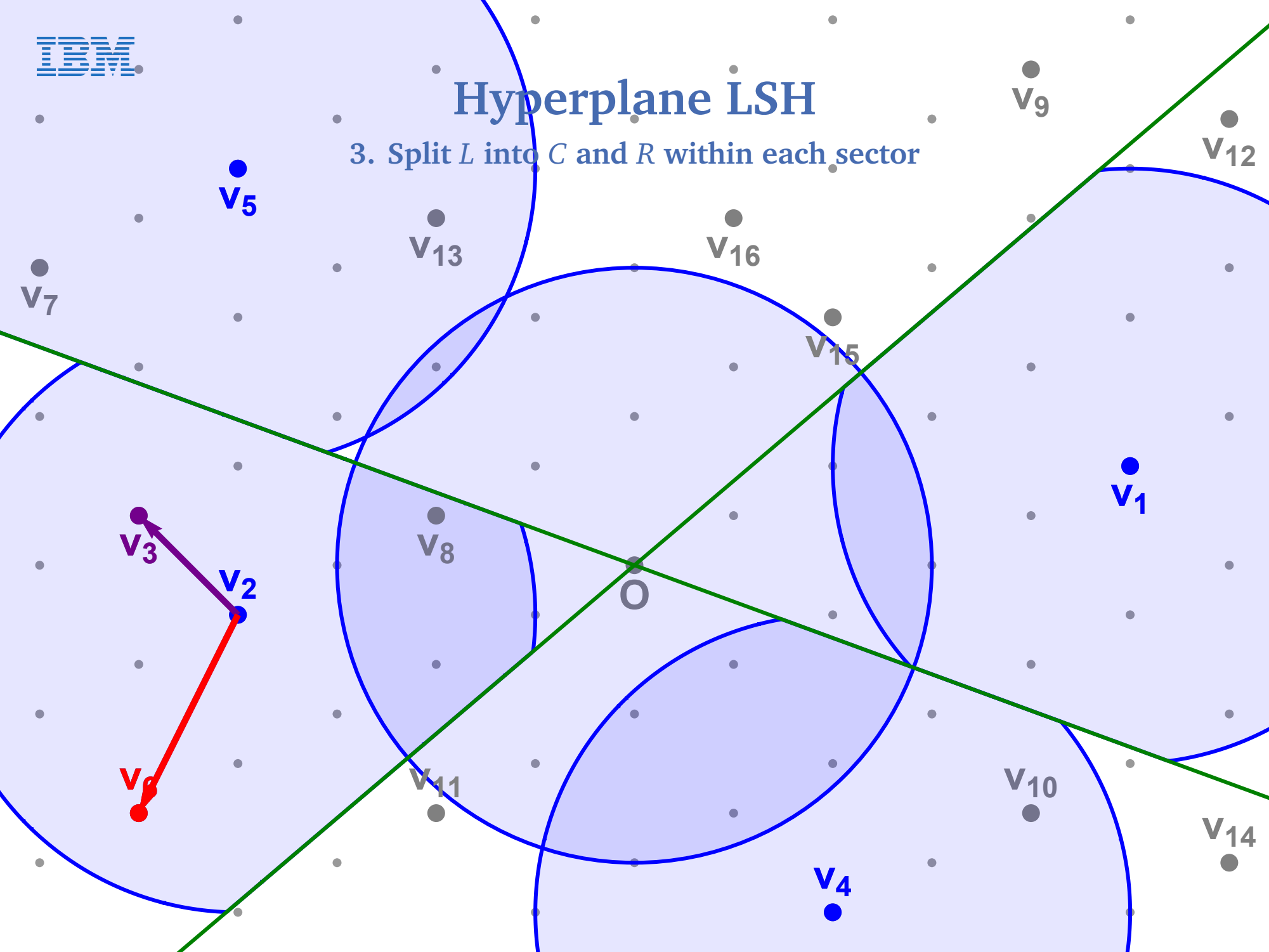
3. Split L into C and R within each sector




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

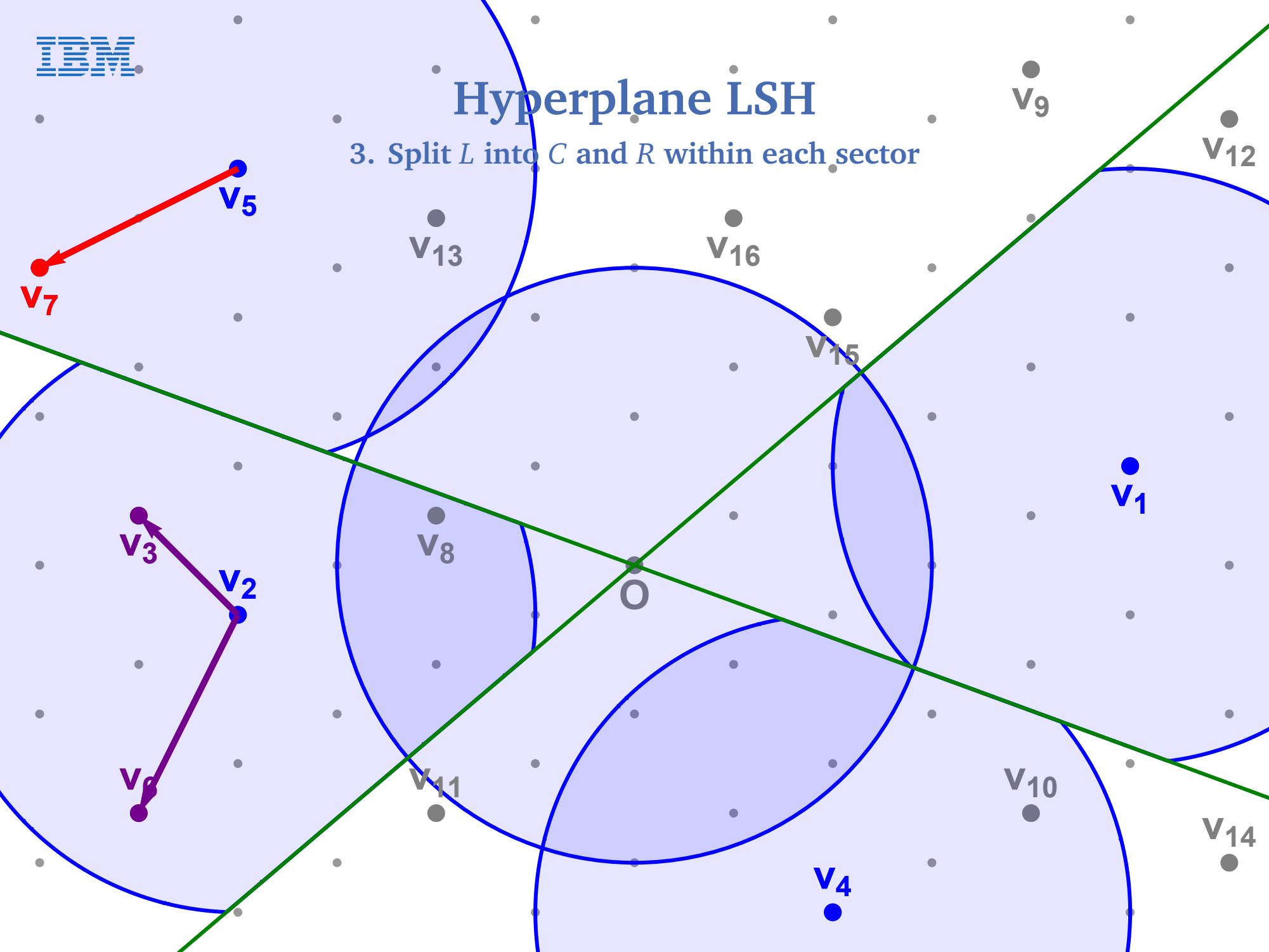
Hyperplane LSH

3. Split L into C and R within each sector



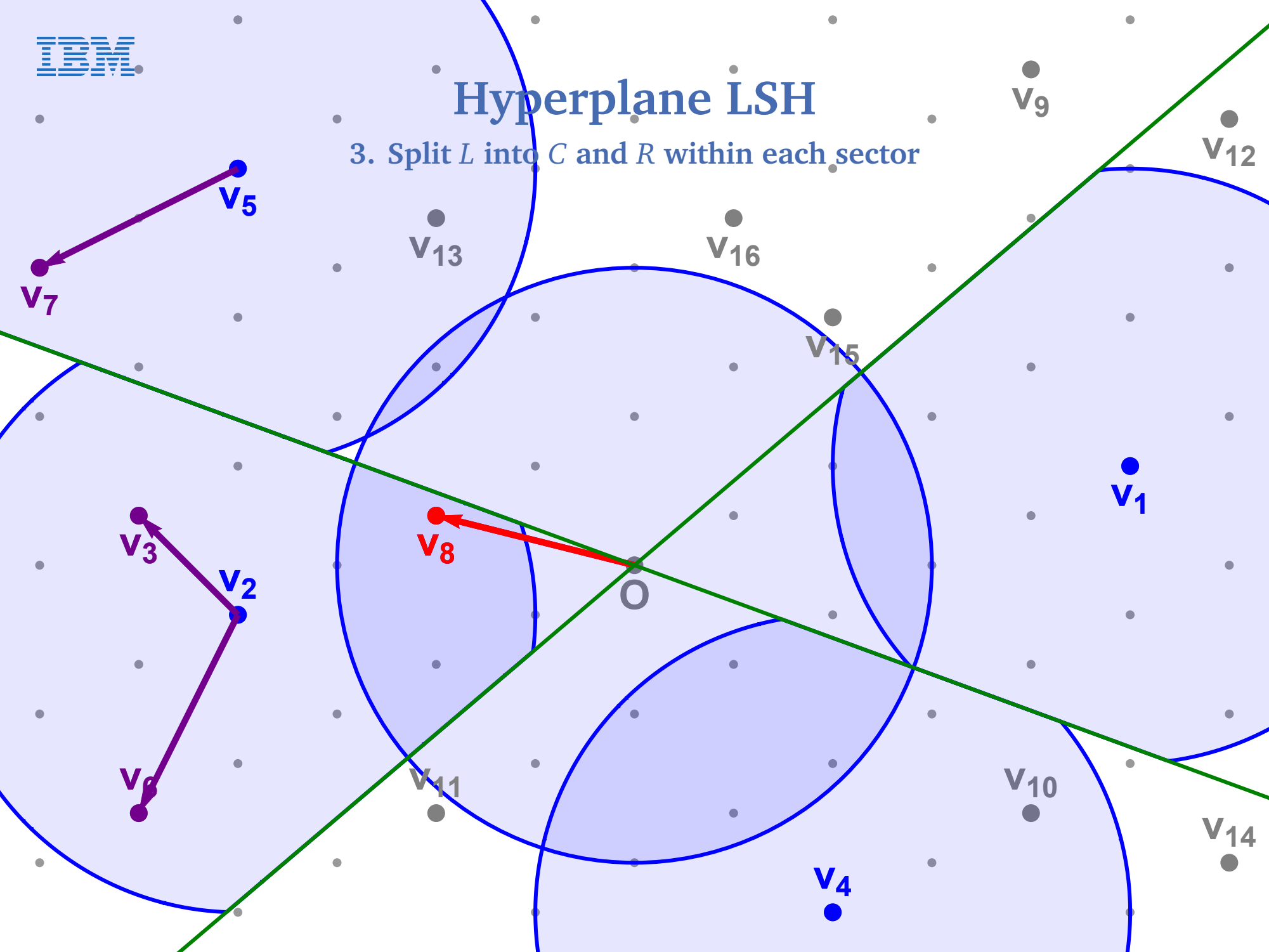
Hyperplane LSH

3. Split L into C and R within each sector



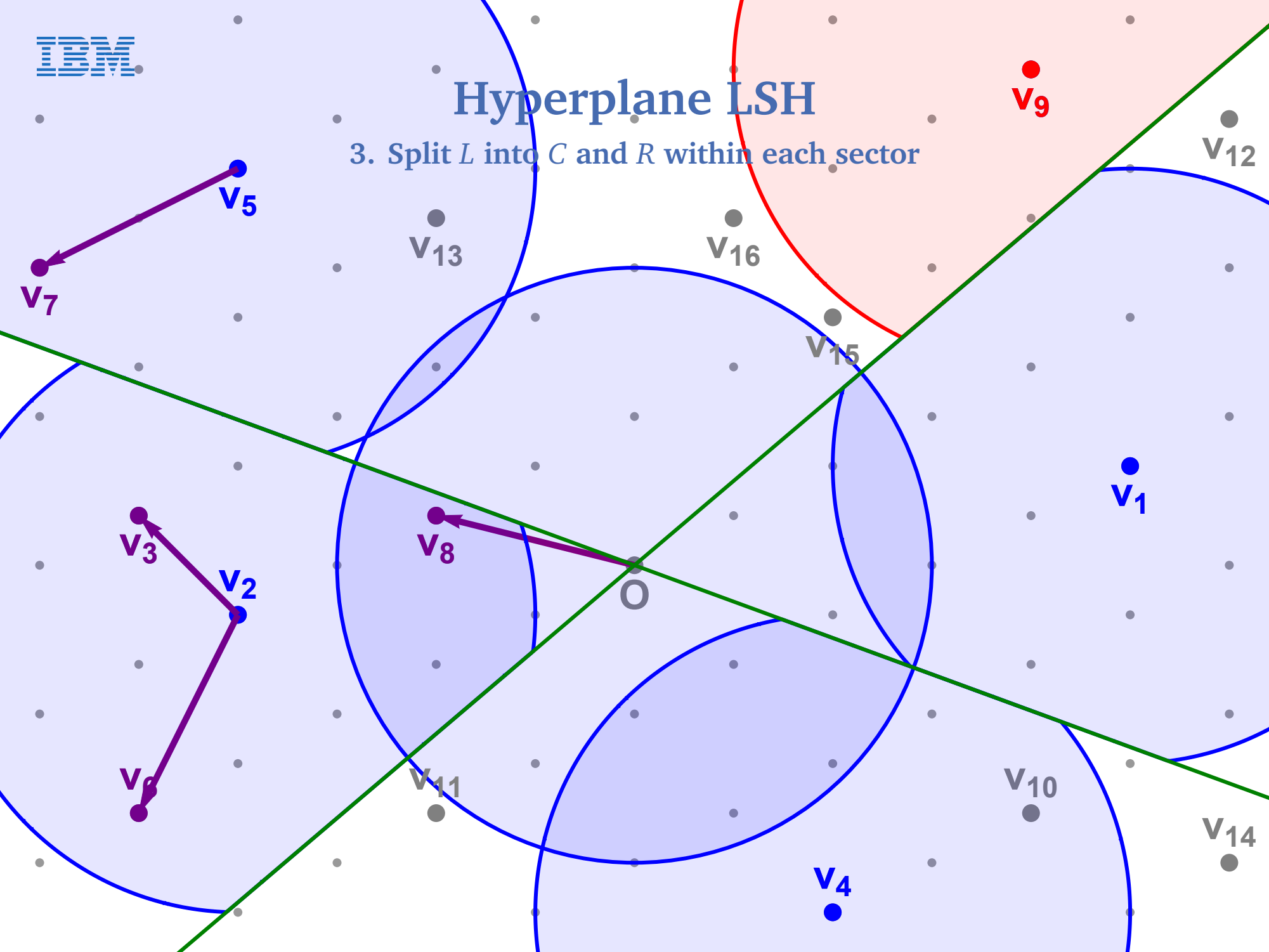
Hyperplane LSH

3. Split L into C and R within each sector



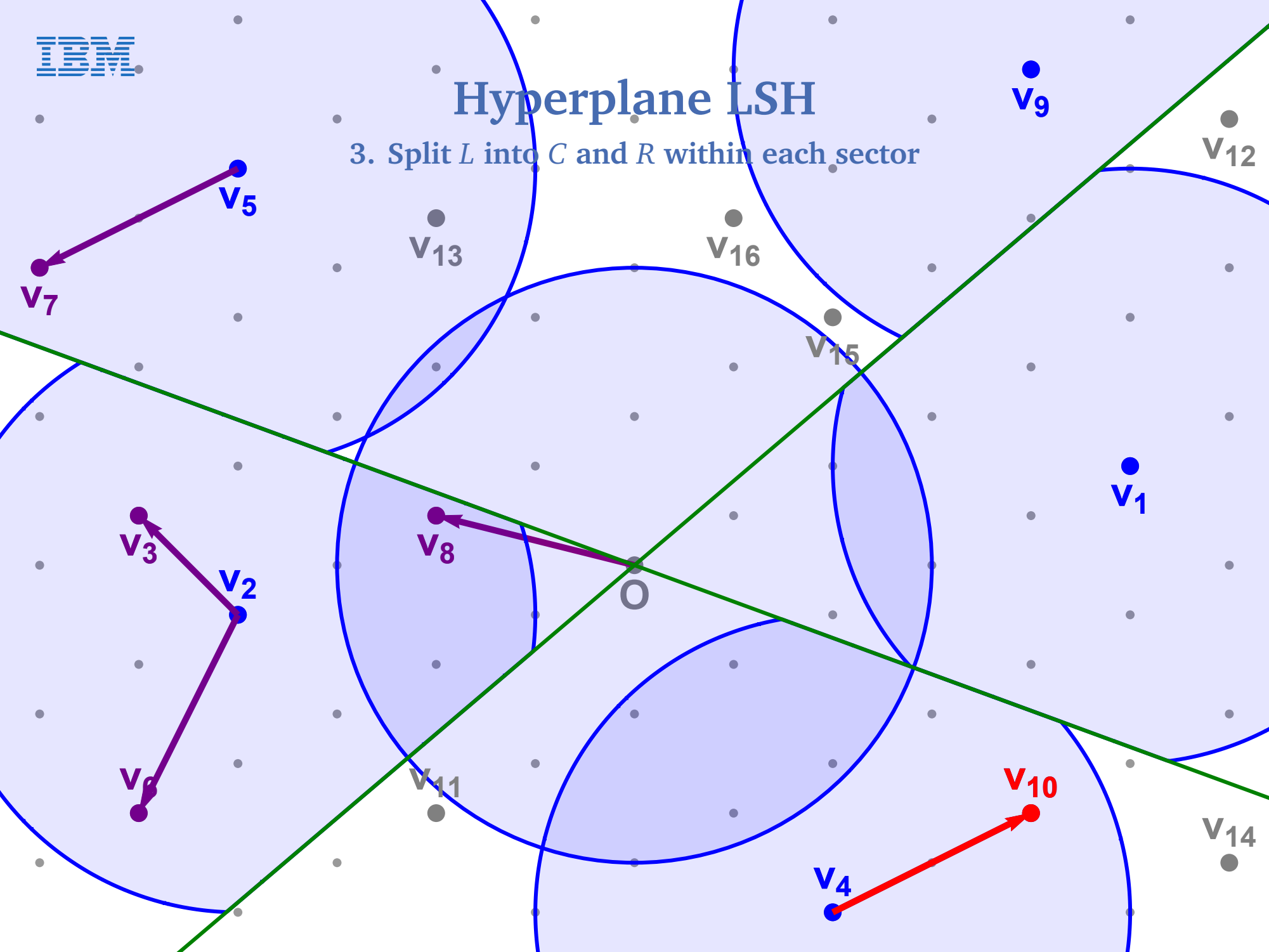
Hyperplane LSH

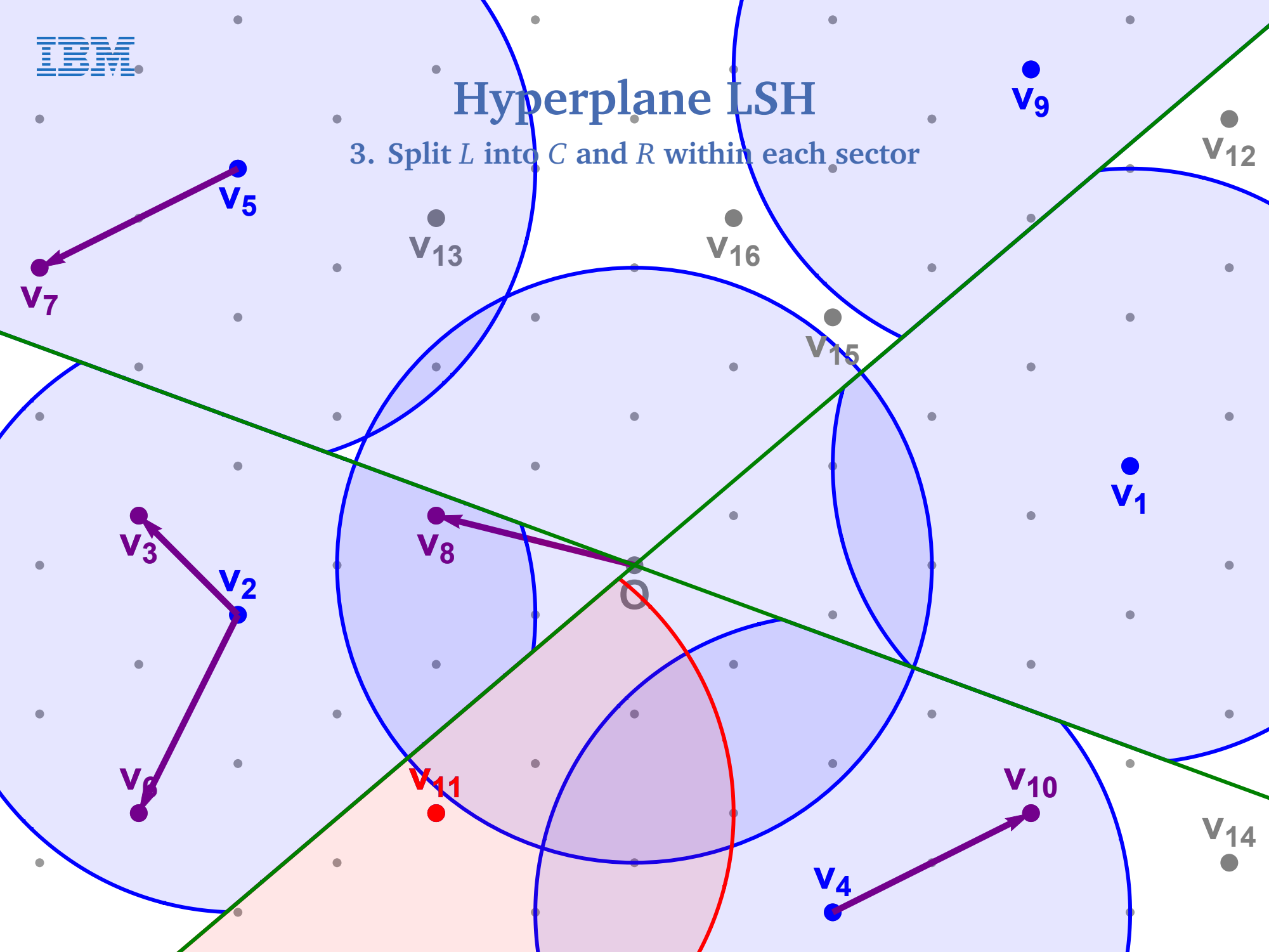
3. Split L into C and R within each sector



Hyperplane LSH

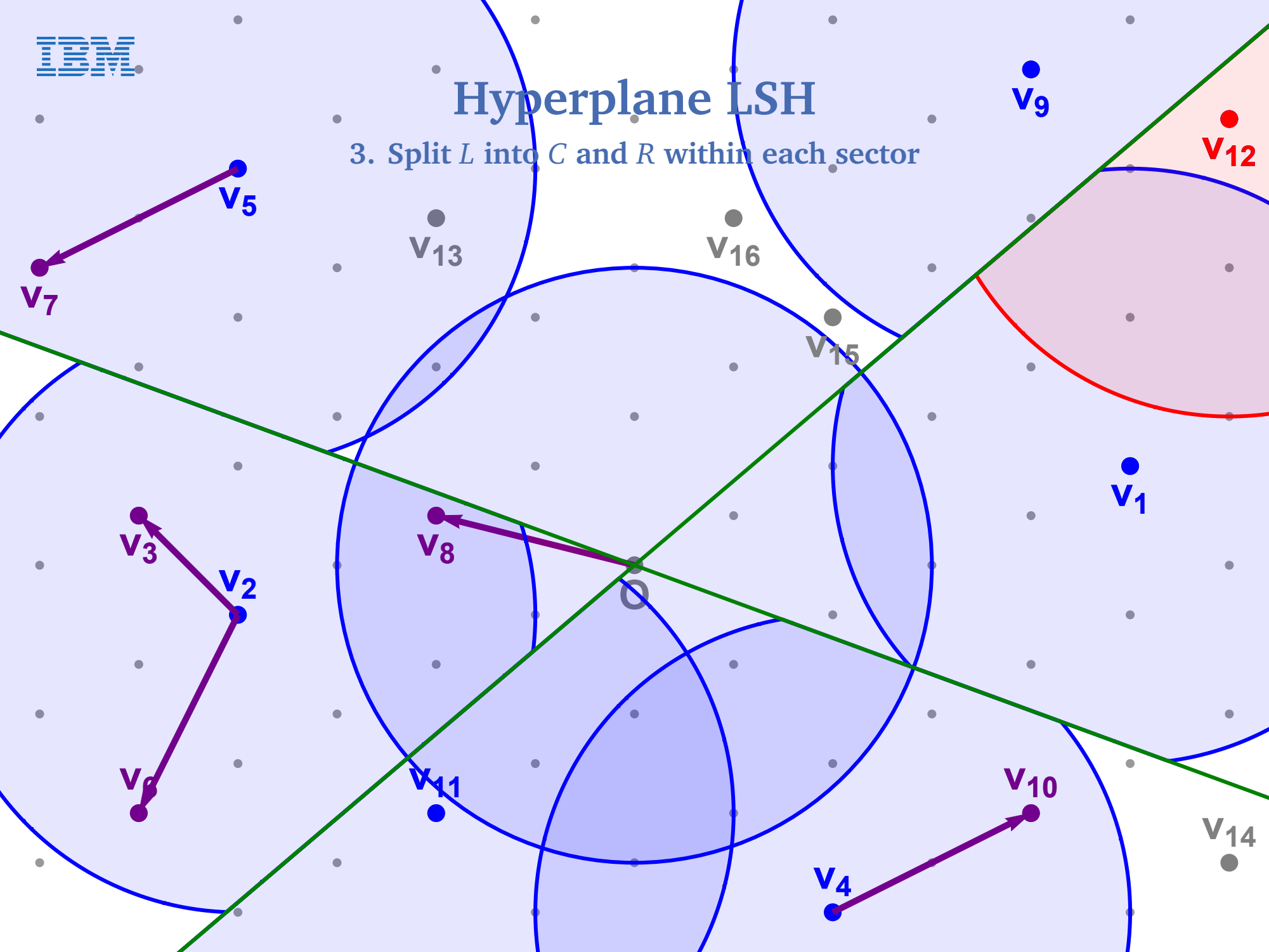
3. Split L into C and R within each sector




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

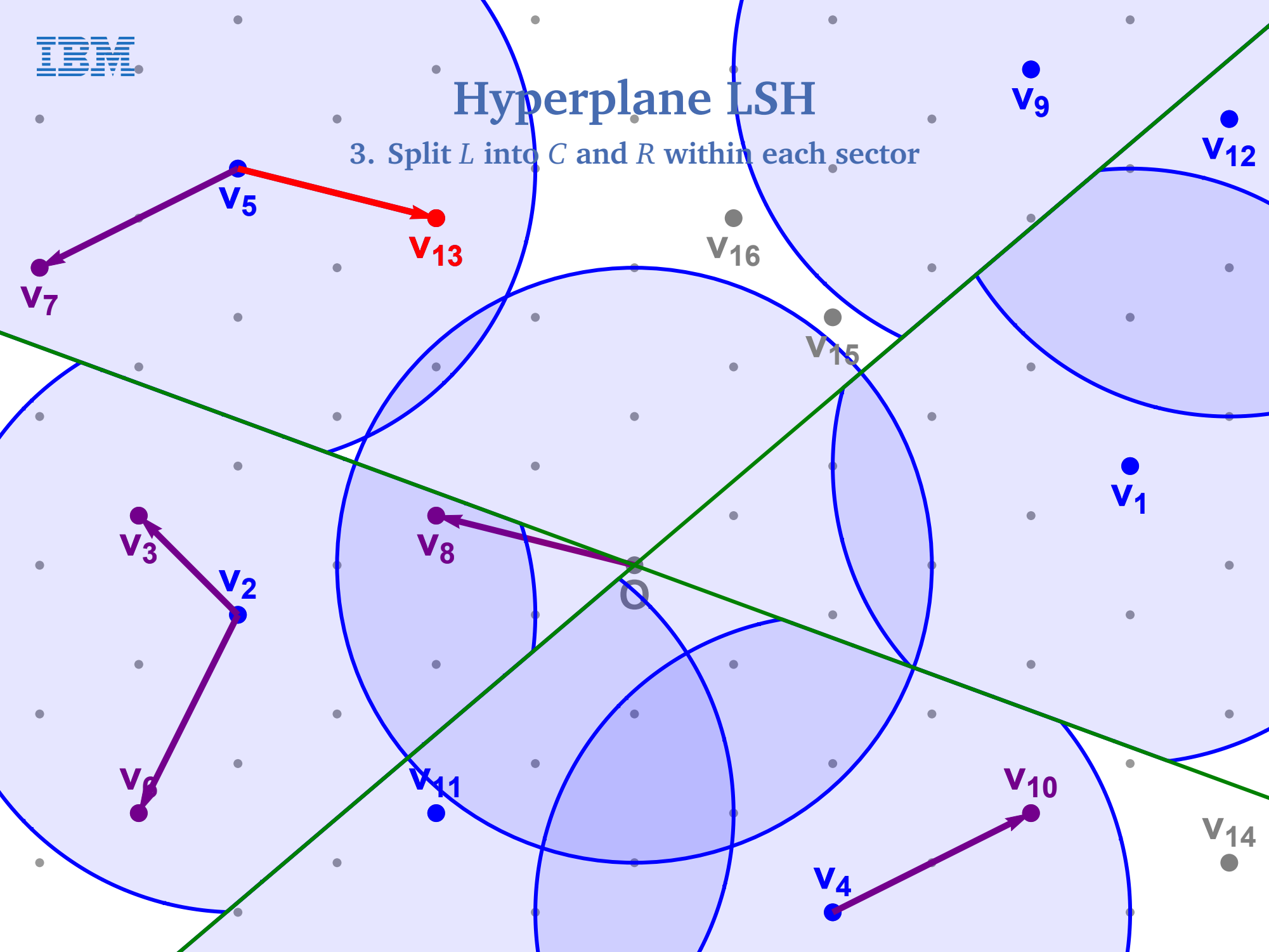
Hyperplane LSH

3. Split L into C and R within each sector



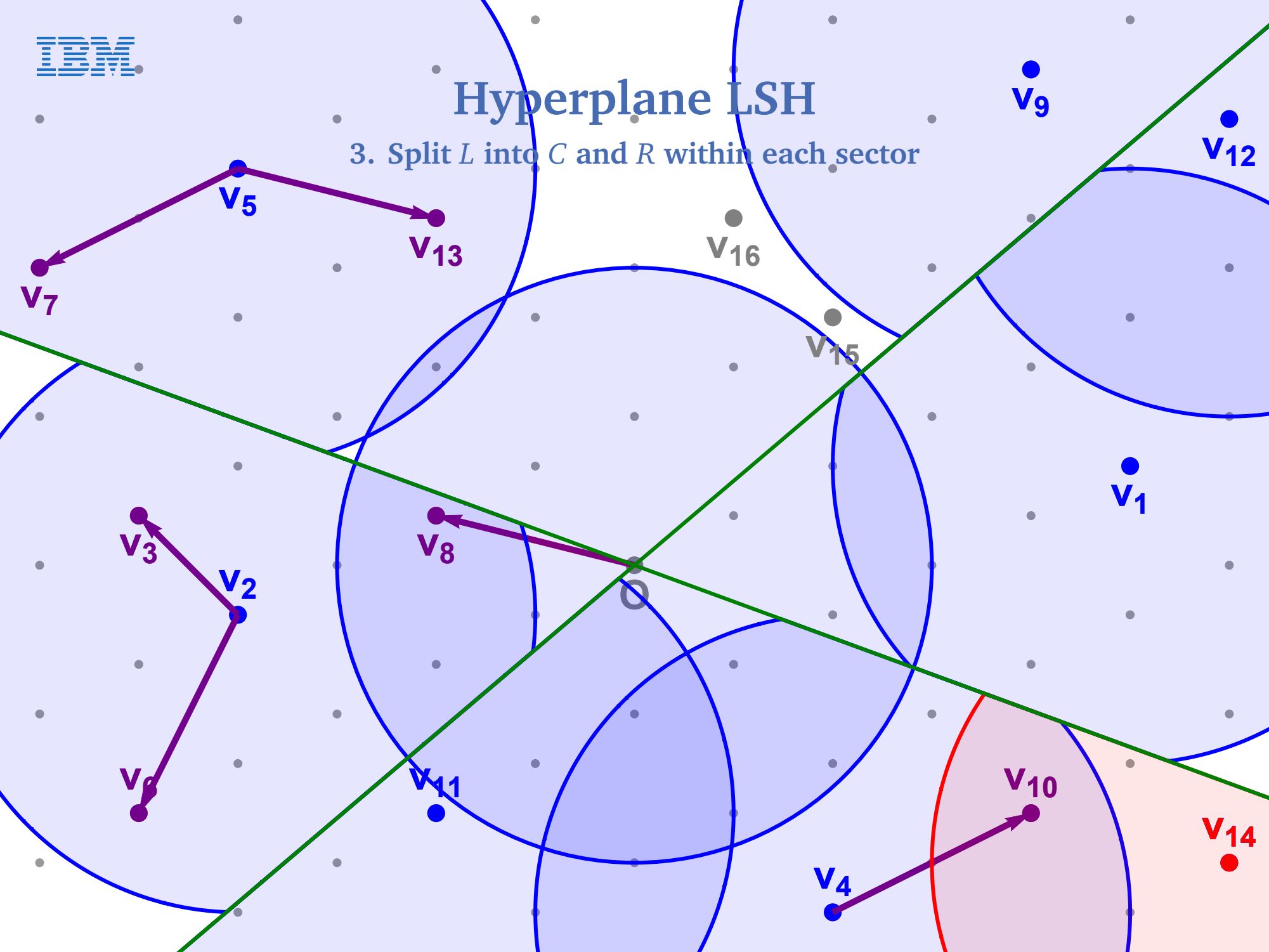
Hyperplane LSH

3. Split L into C and R within each sector



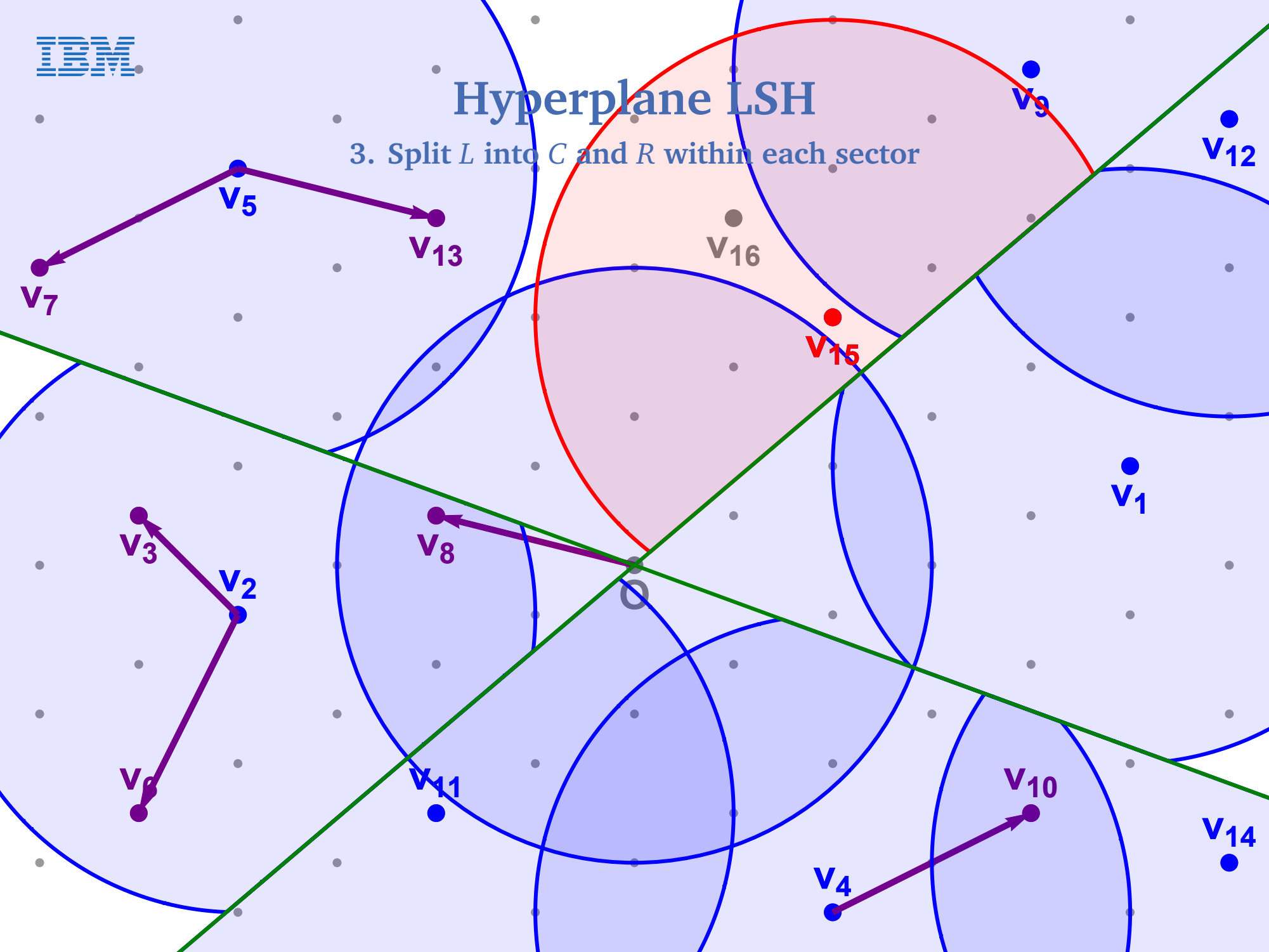
Hyperplane LSH

3. Split L into C and R within each sector



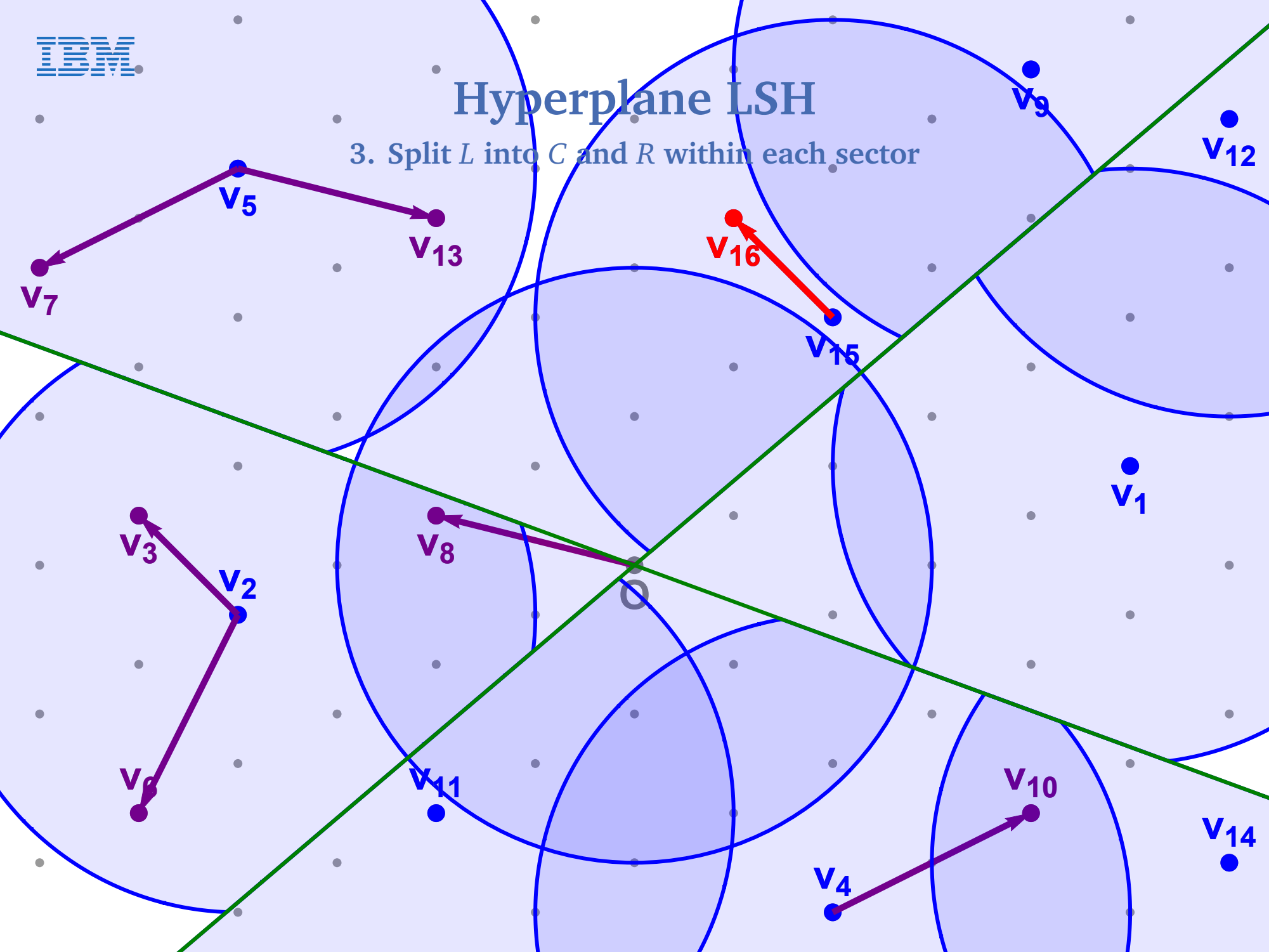
Hyperplane LSH

3. Split L into C and R within each sector



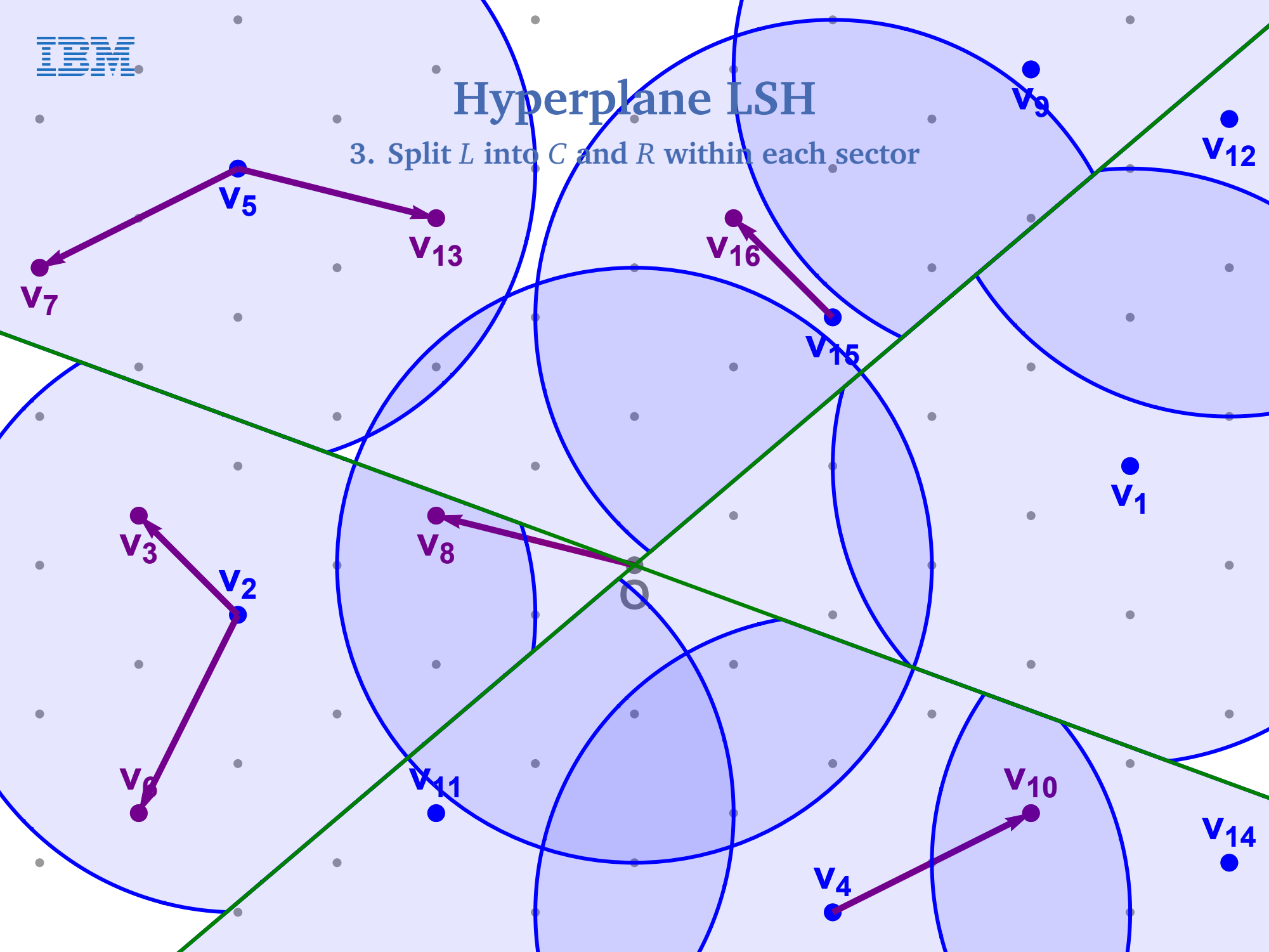
Hyperplane LSH

3. Split L into C and R within each sector



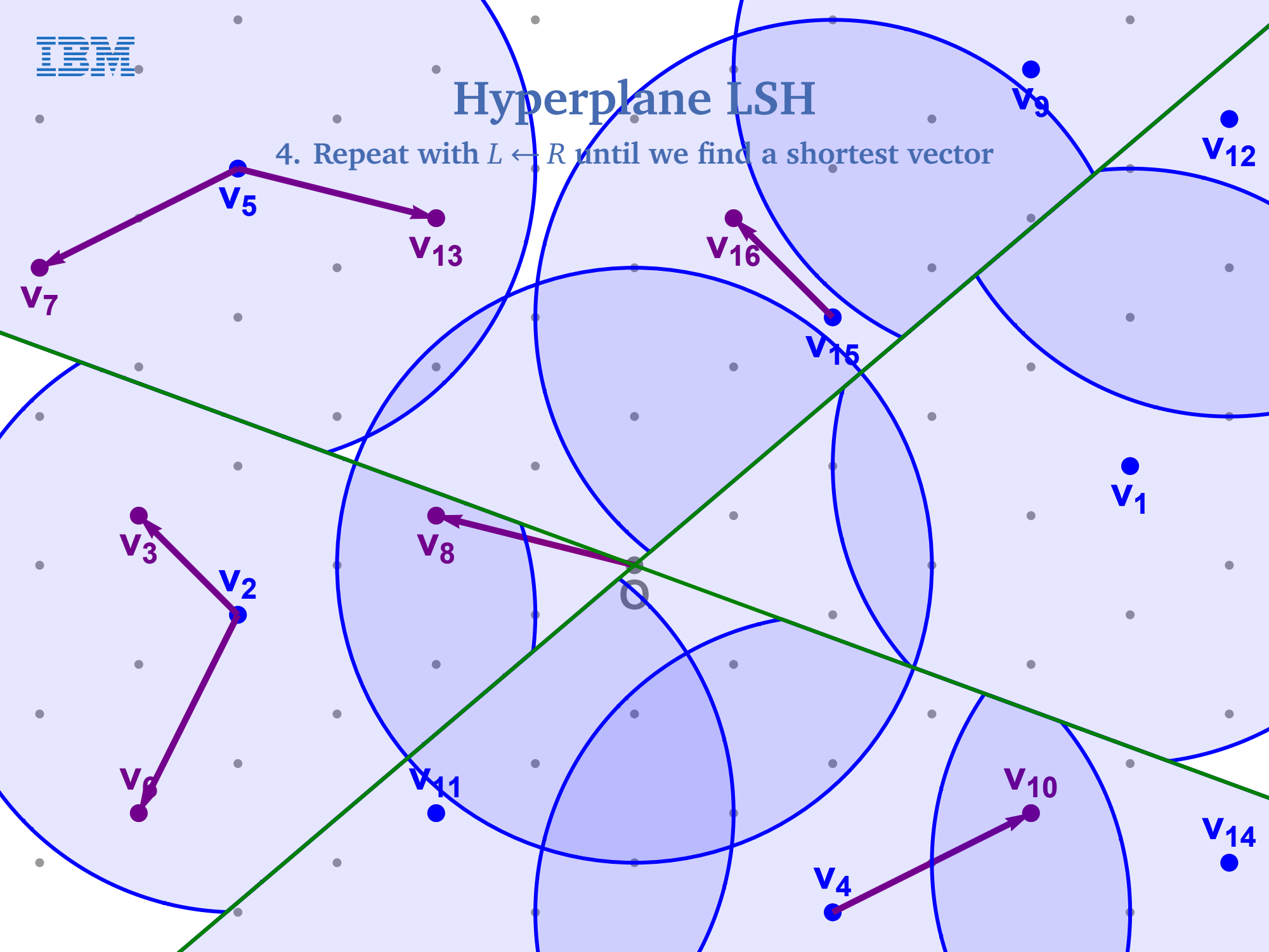
Hyperplane LSH

3. Split L into C and R within each sector



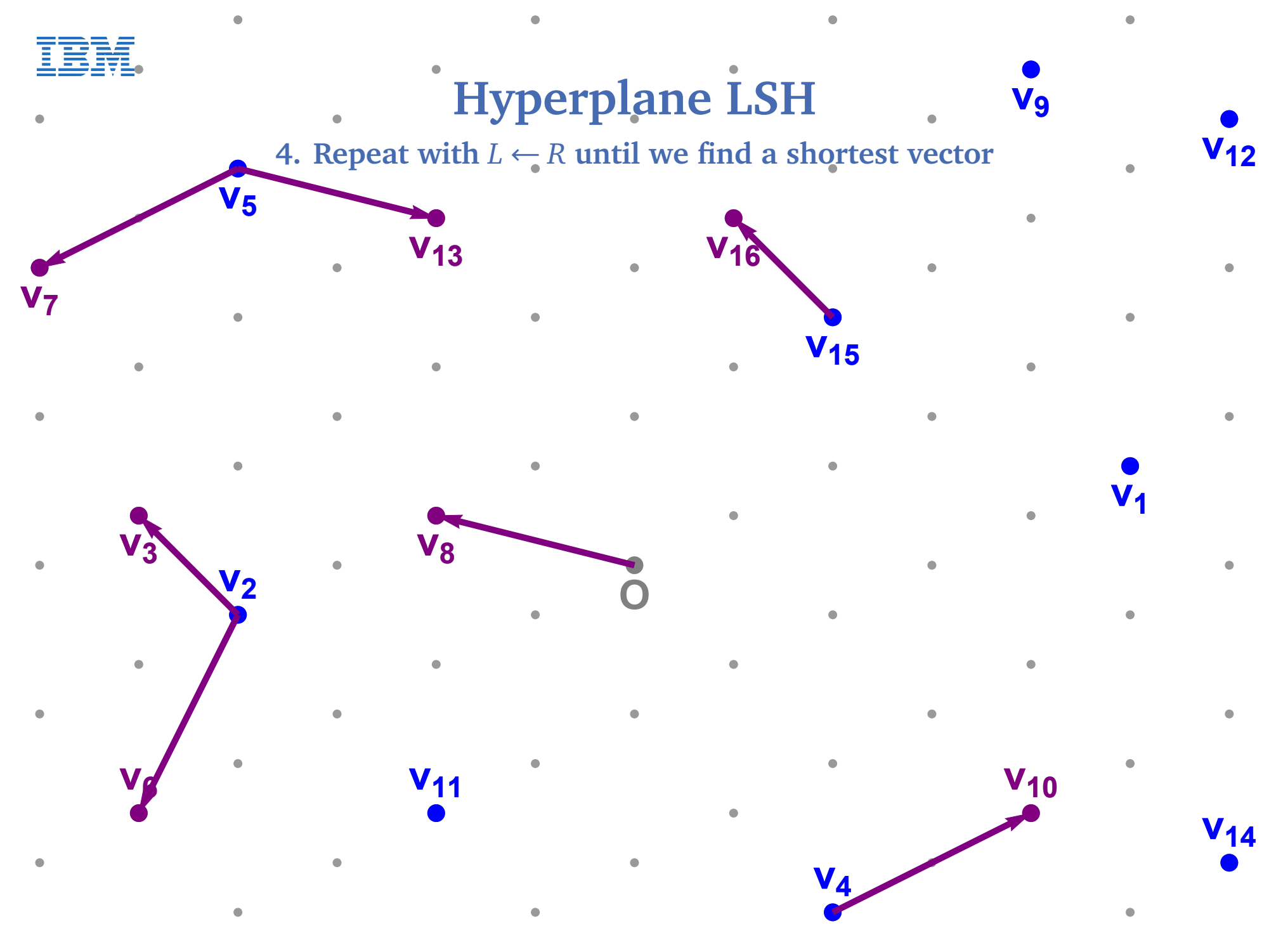
Hyperplane LSH

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



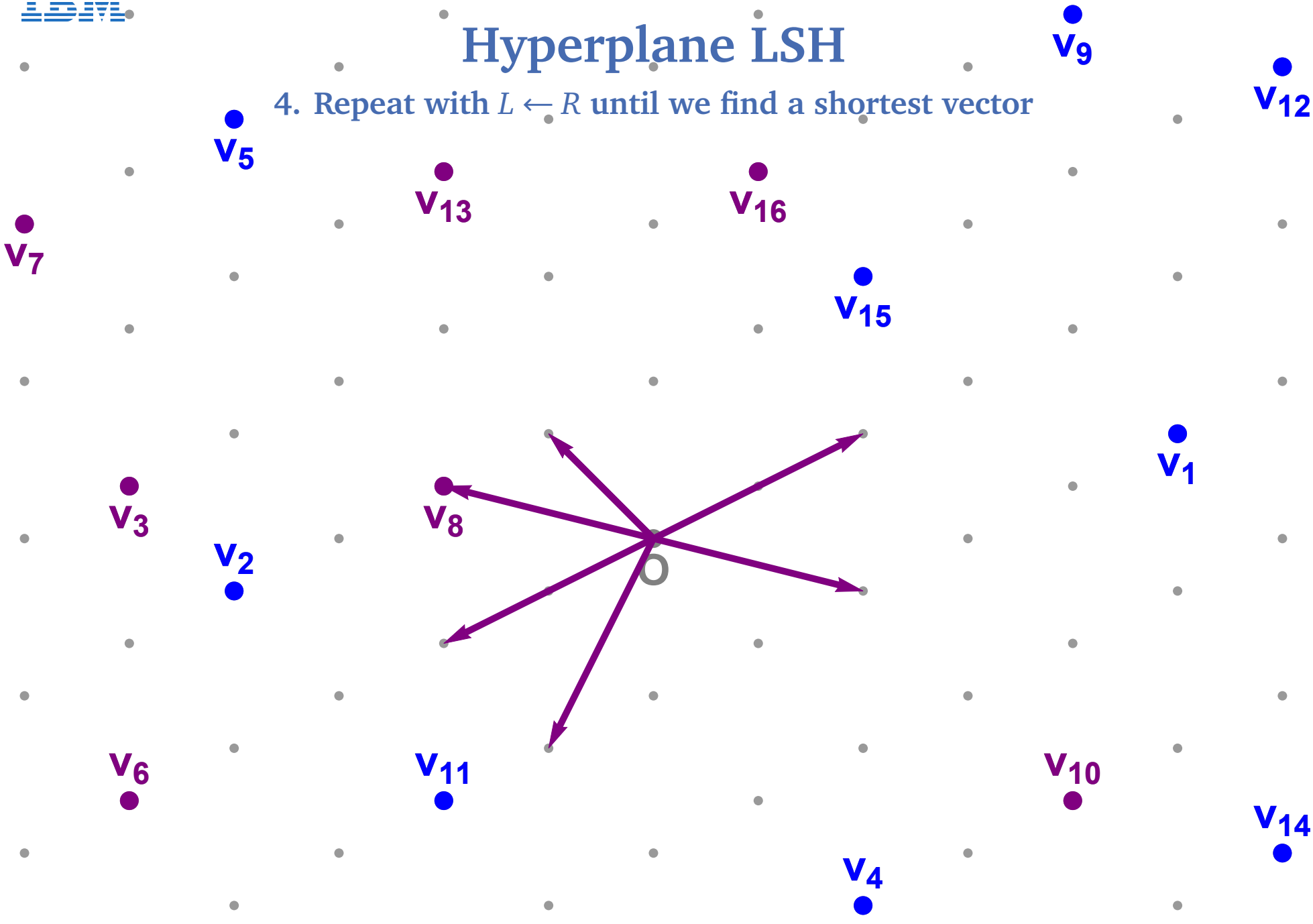
Hyperplane LSH

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



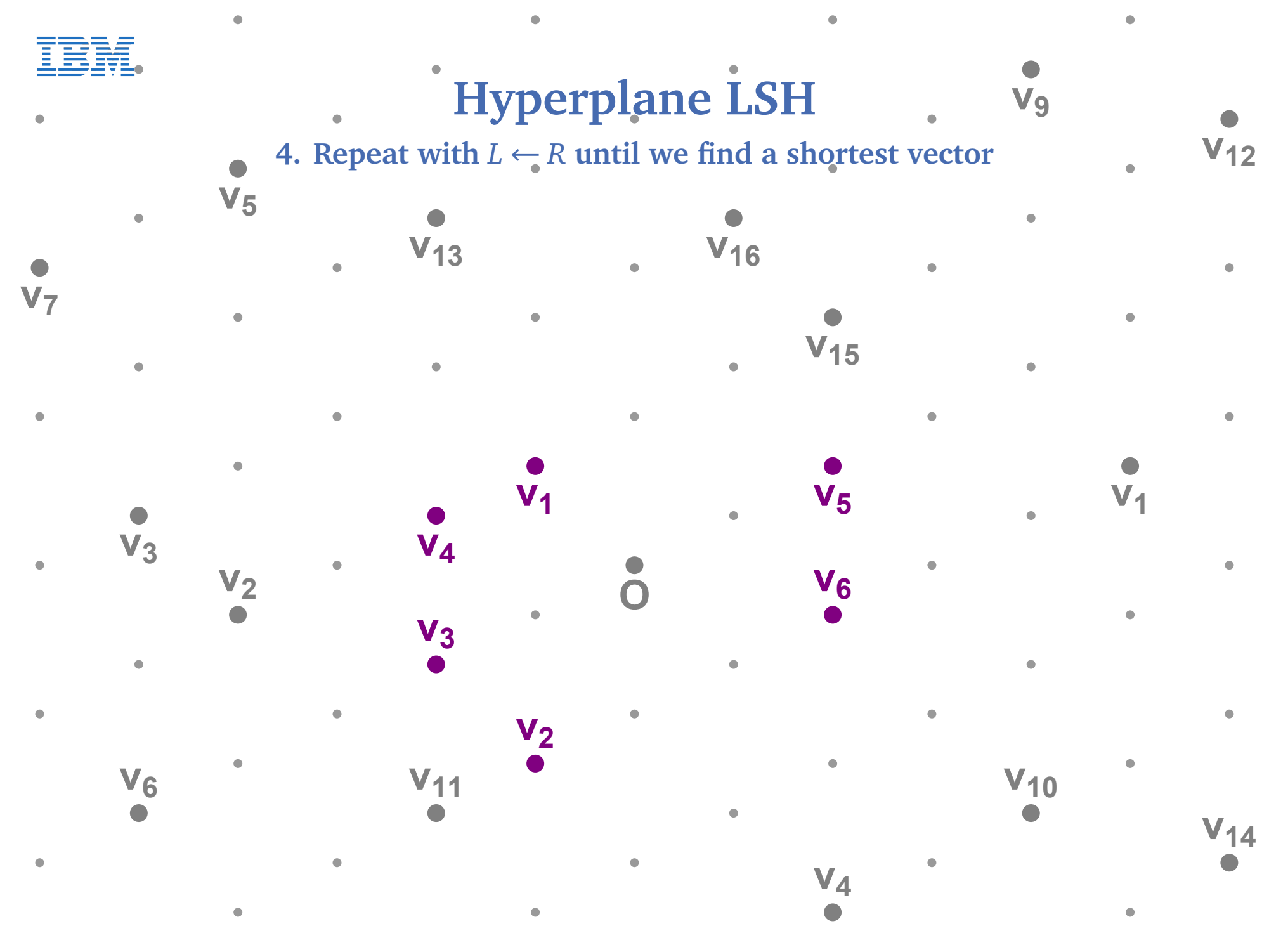
Hyperplane LSH

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



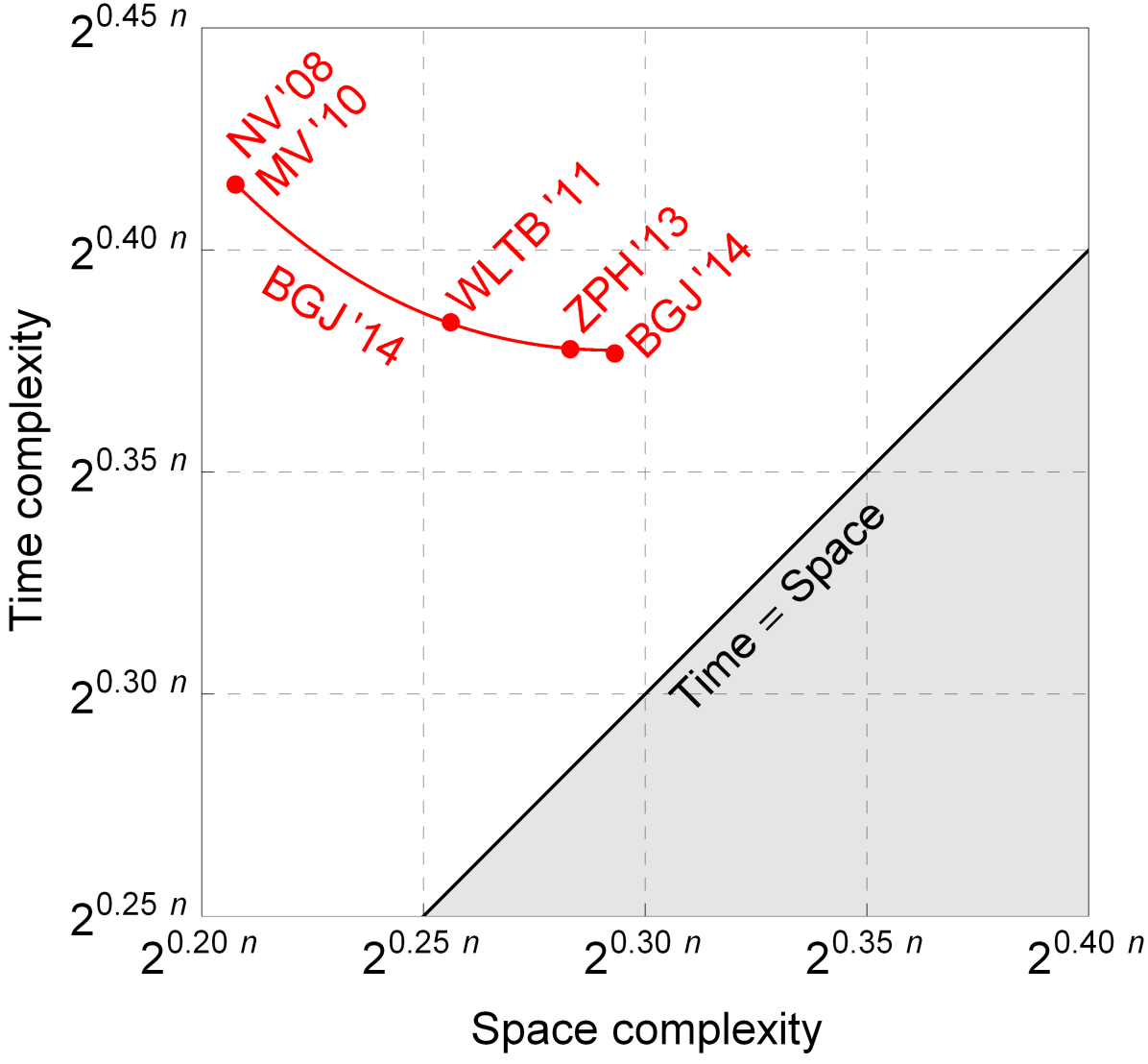
Hyperplane LSH

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



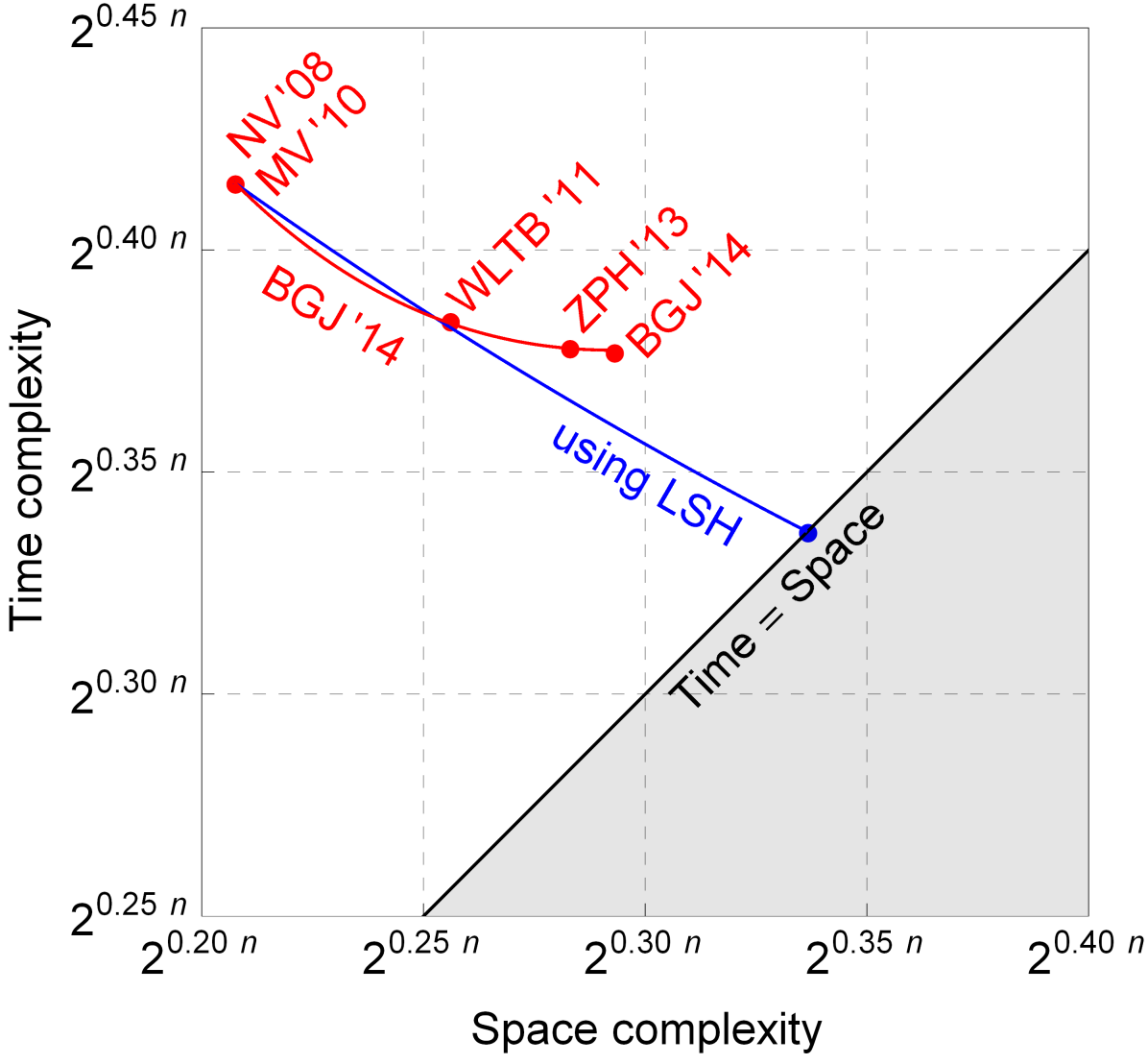
Hyperplane LSH

Space/time trade-off



Hyperplane LSH

Space/time trade-off

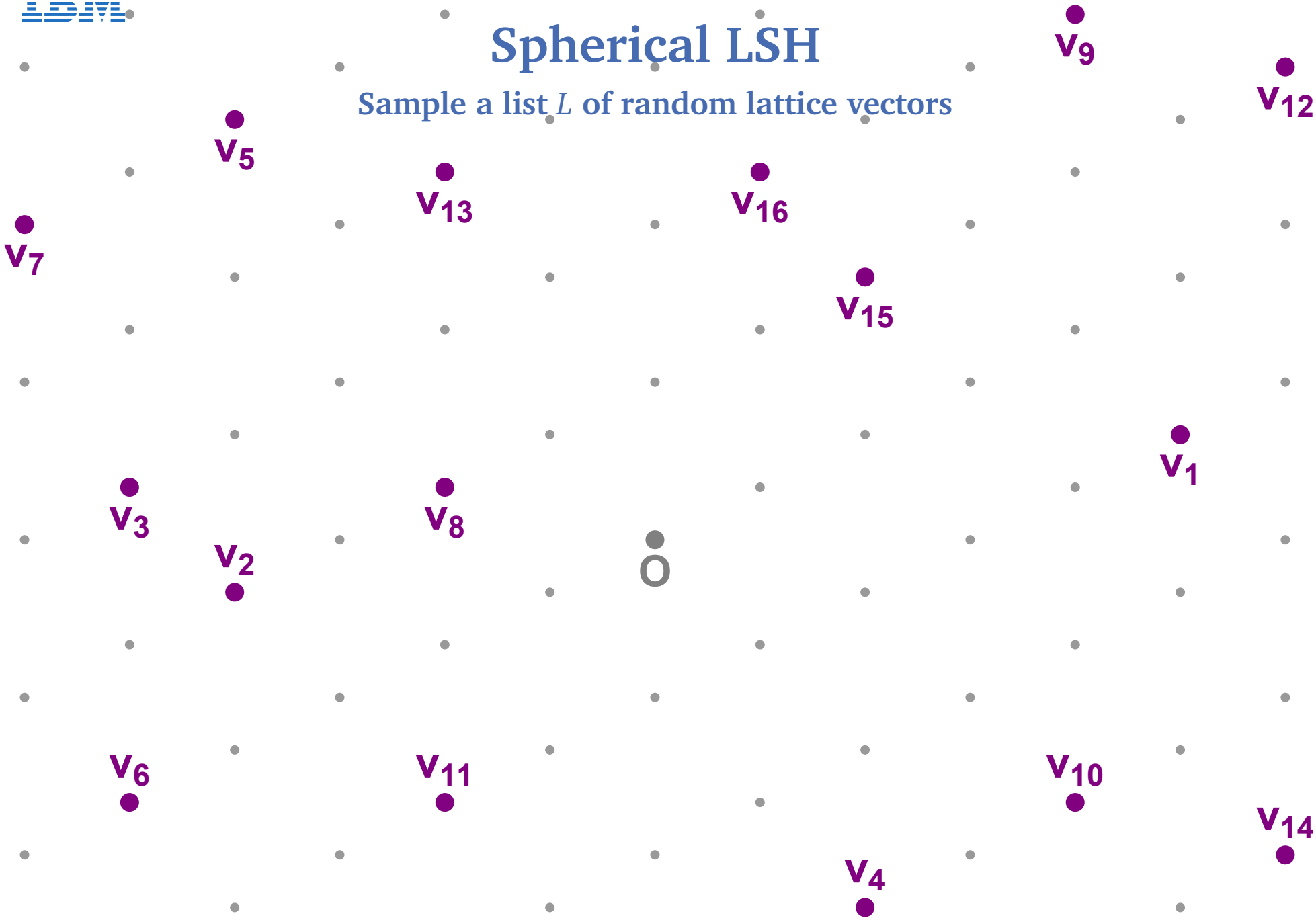


Spherical LSH

Sample a list L of random lattice vectors

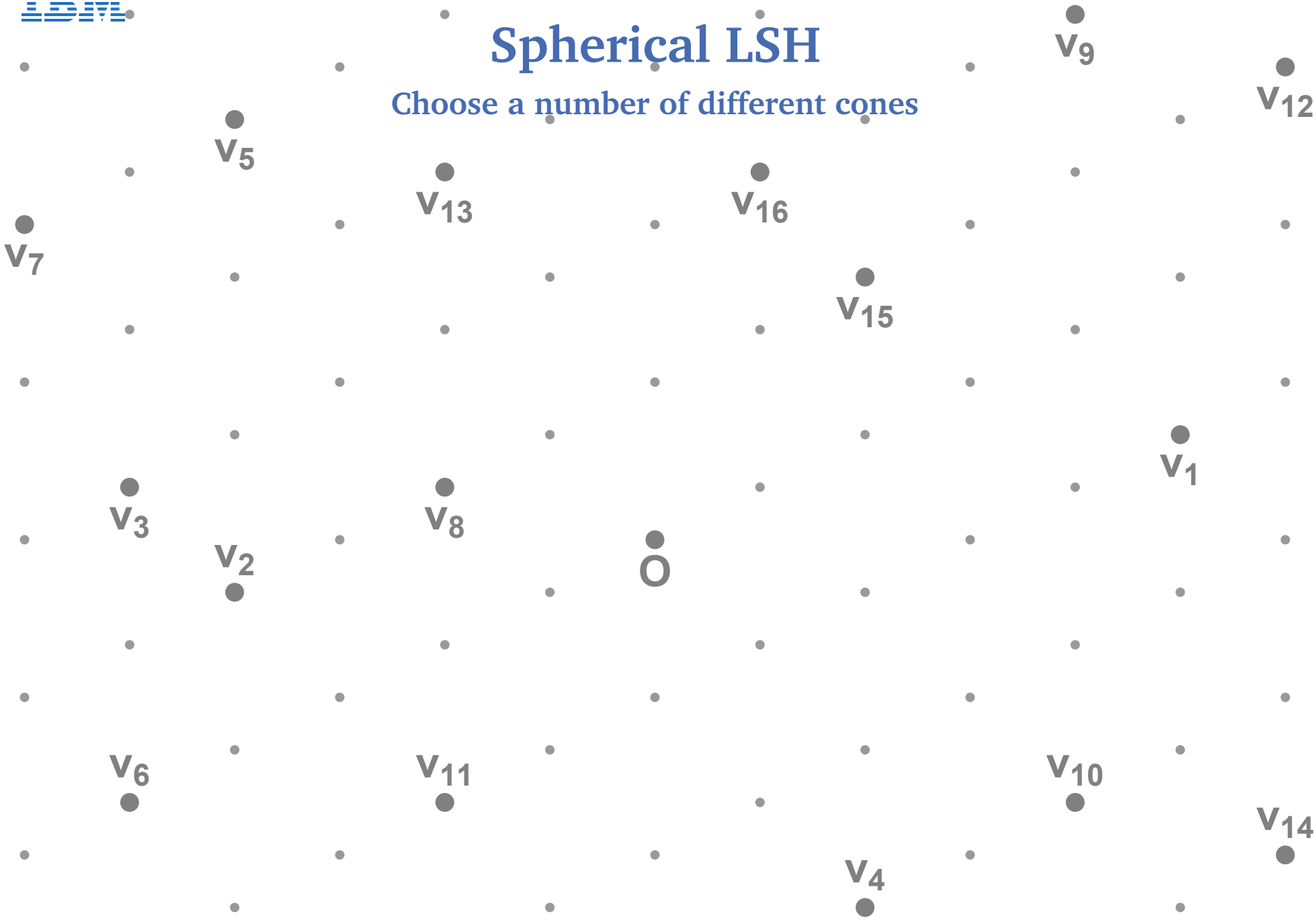
Spherical LSH

Sample a list L of random lattice vectors



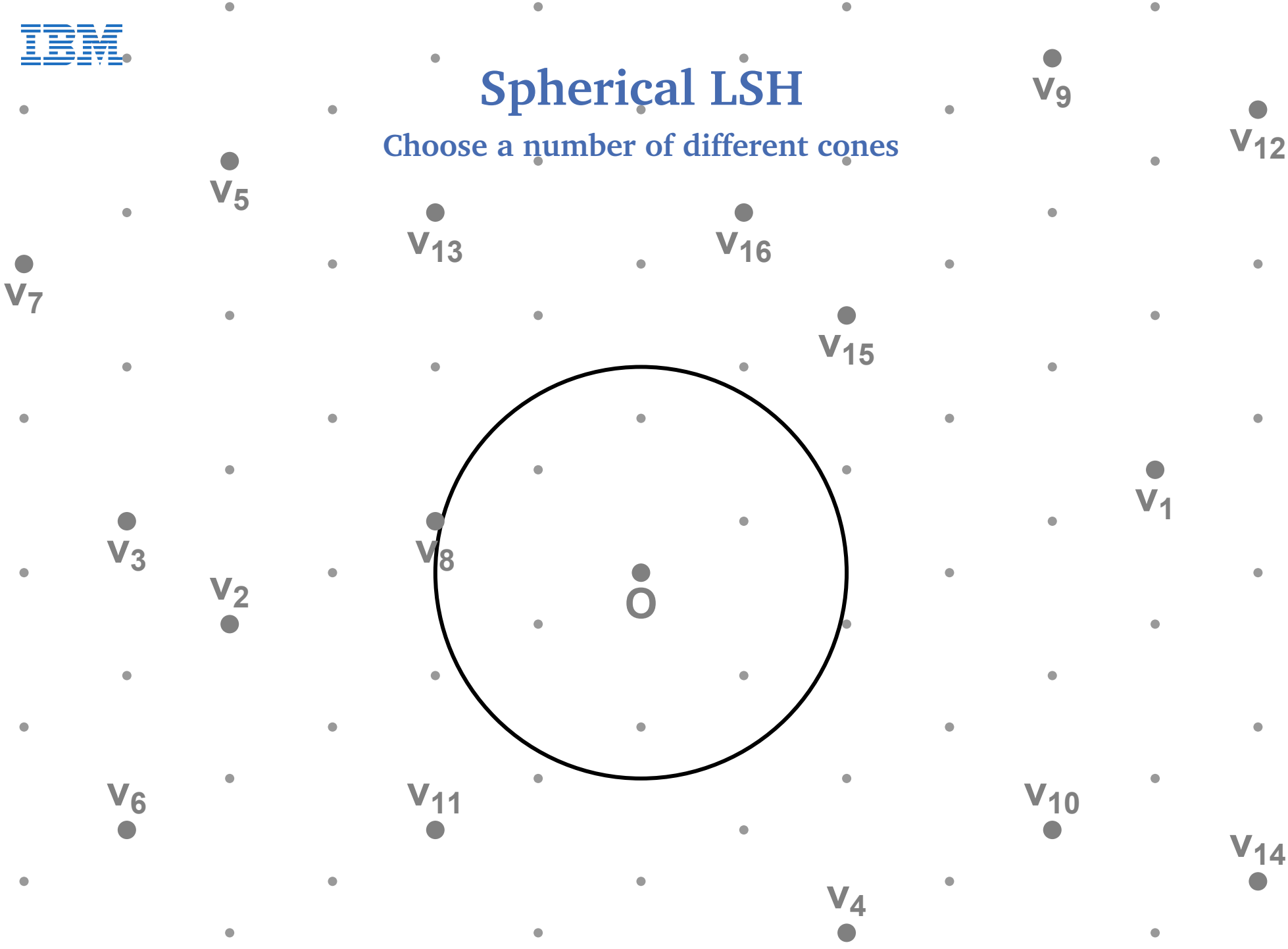
Spherical LSH

Choose a number of different cones



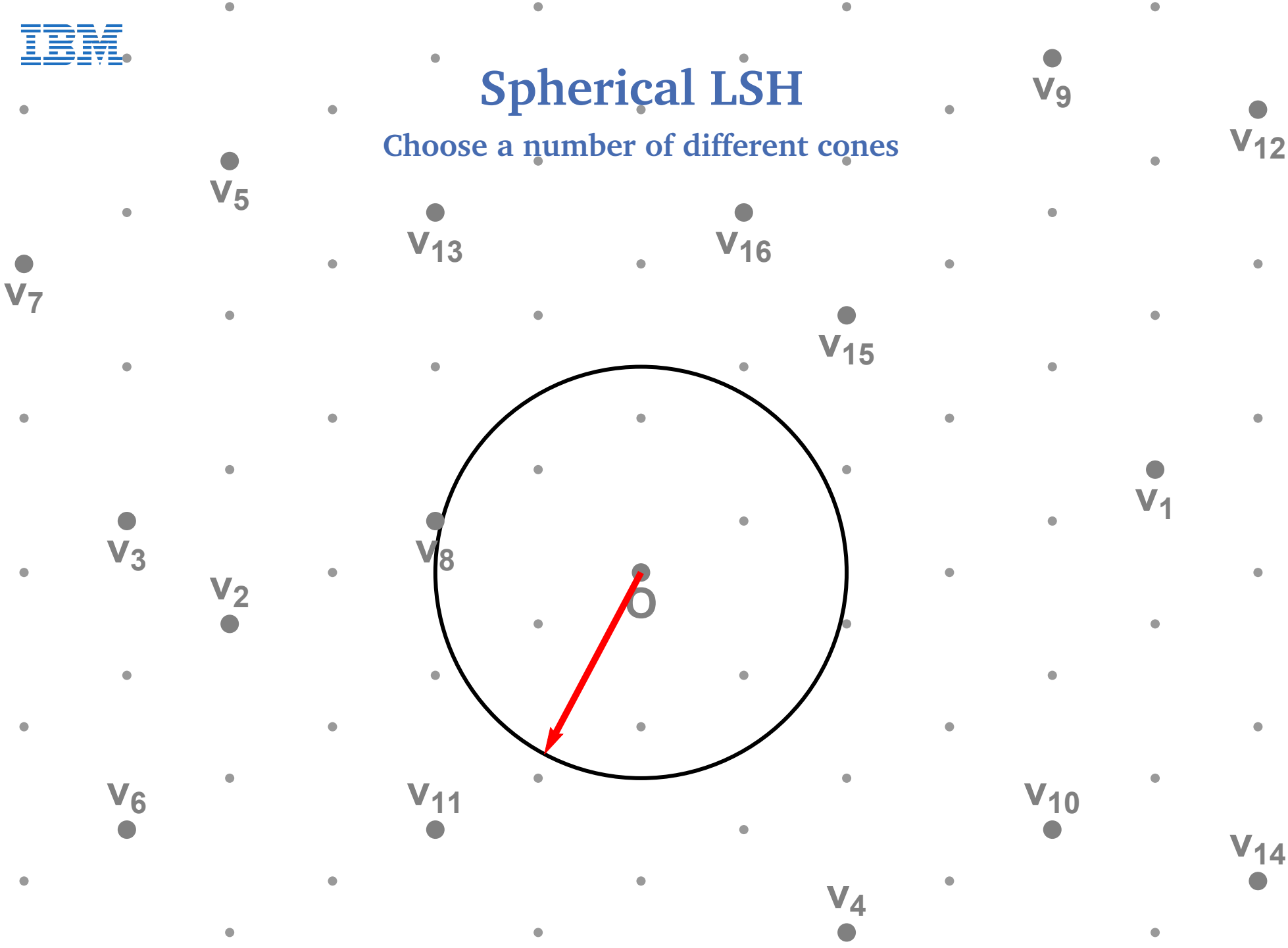
Spherical LSH

Choose a number of different cones



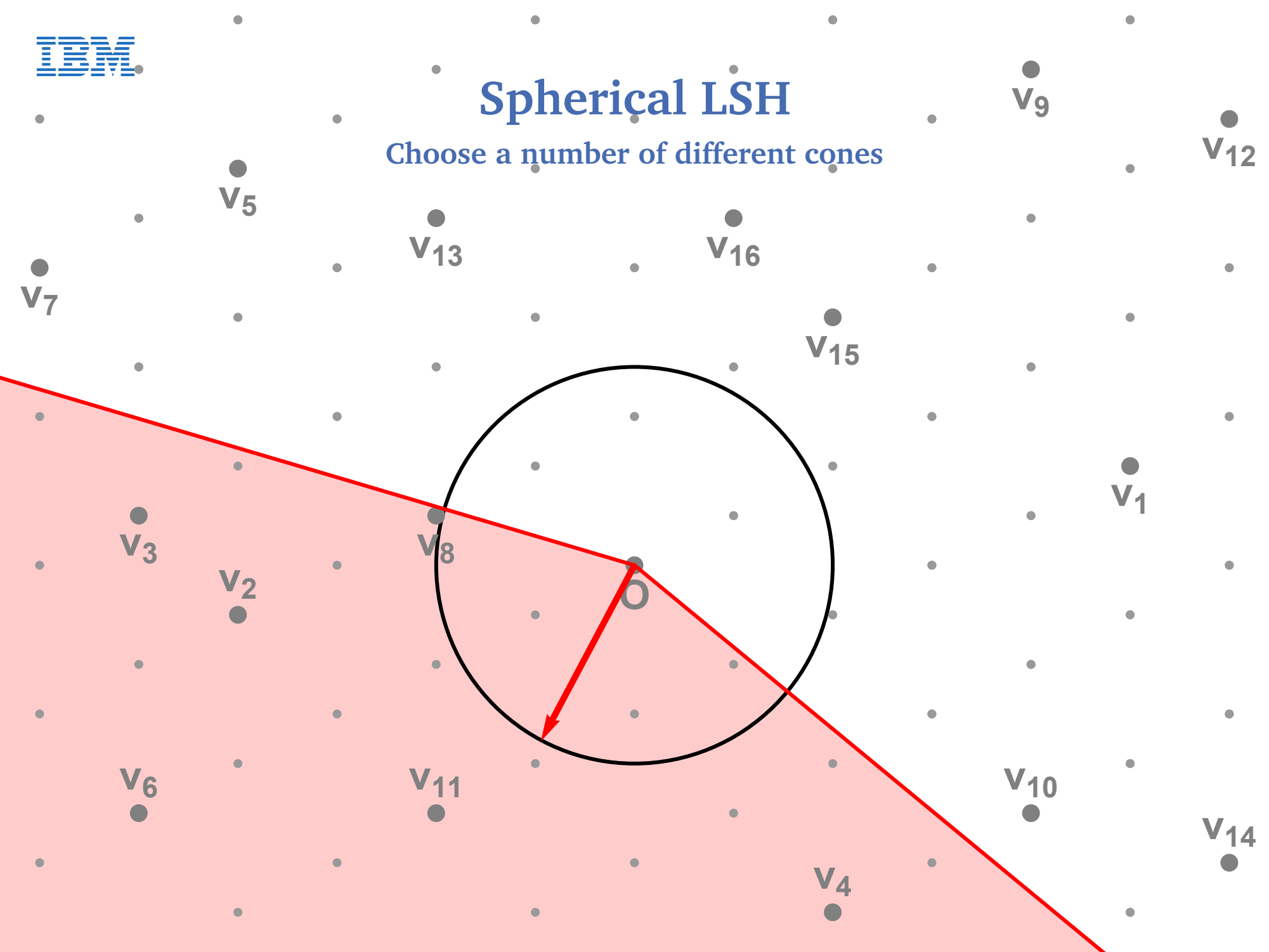
Spherical LSH

Choose a number of different cones



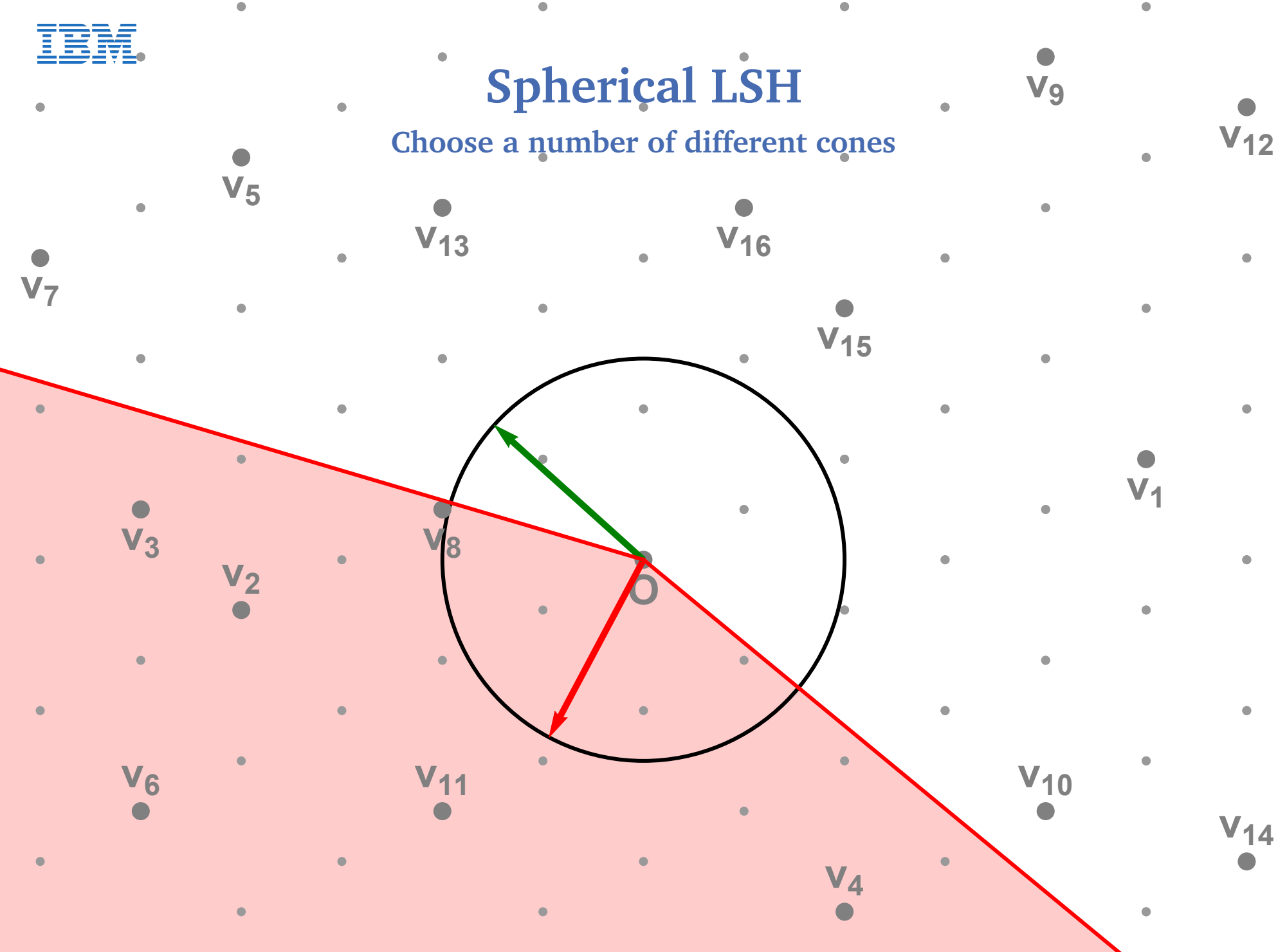
Spherical LSH

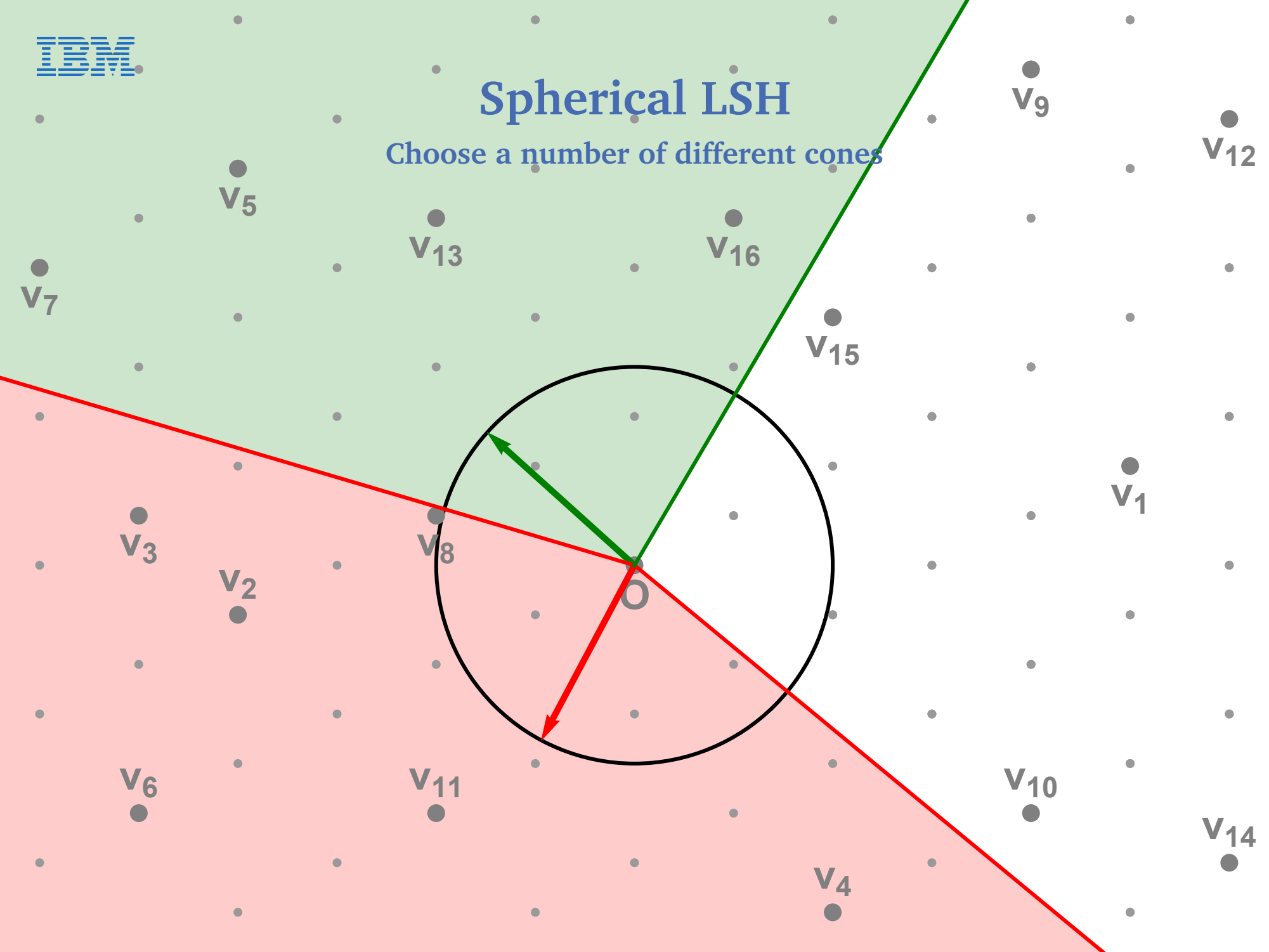
Choose a number of different cones

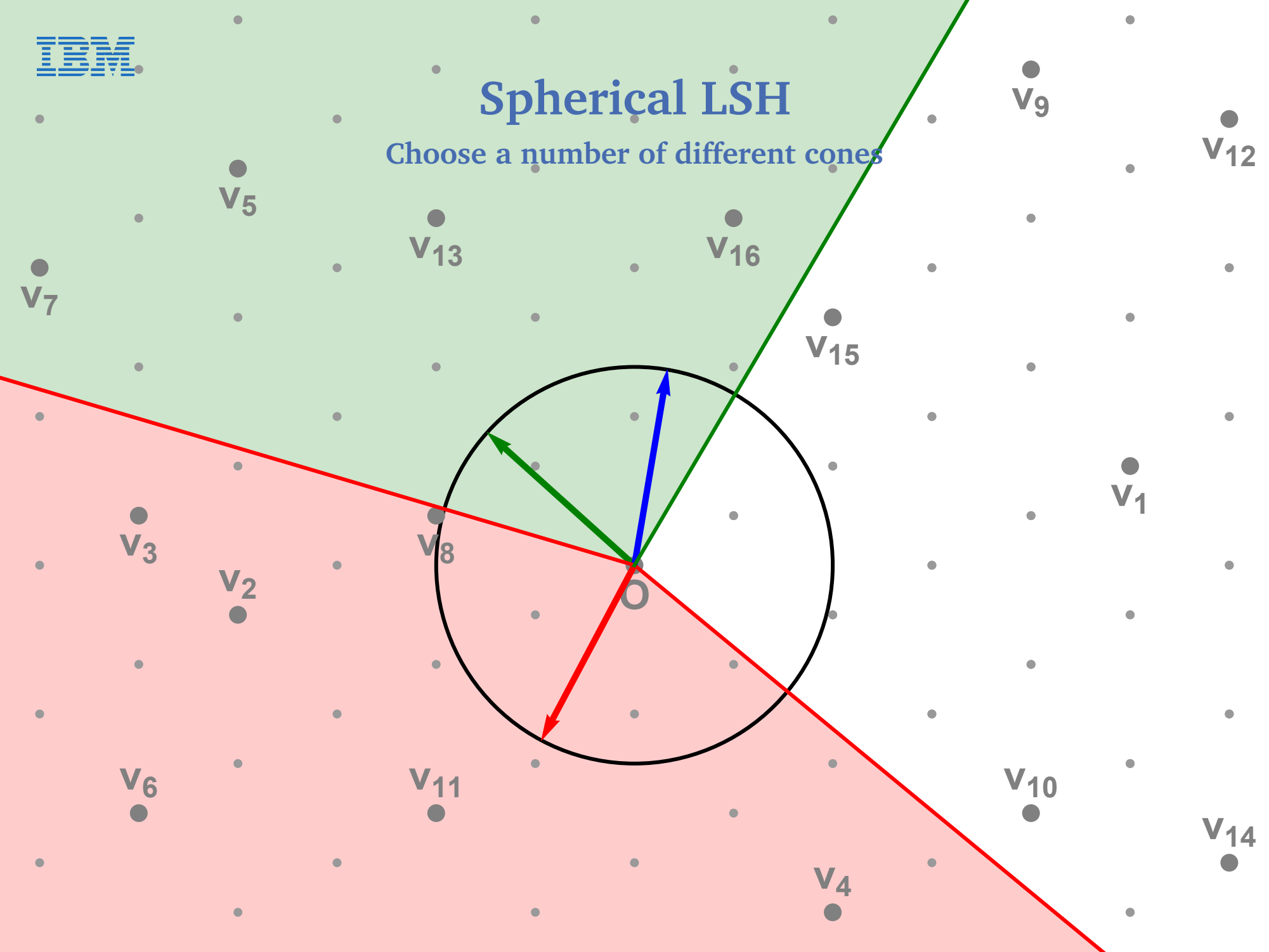


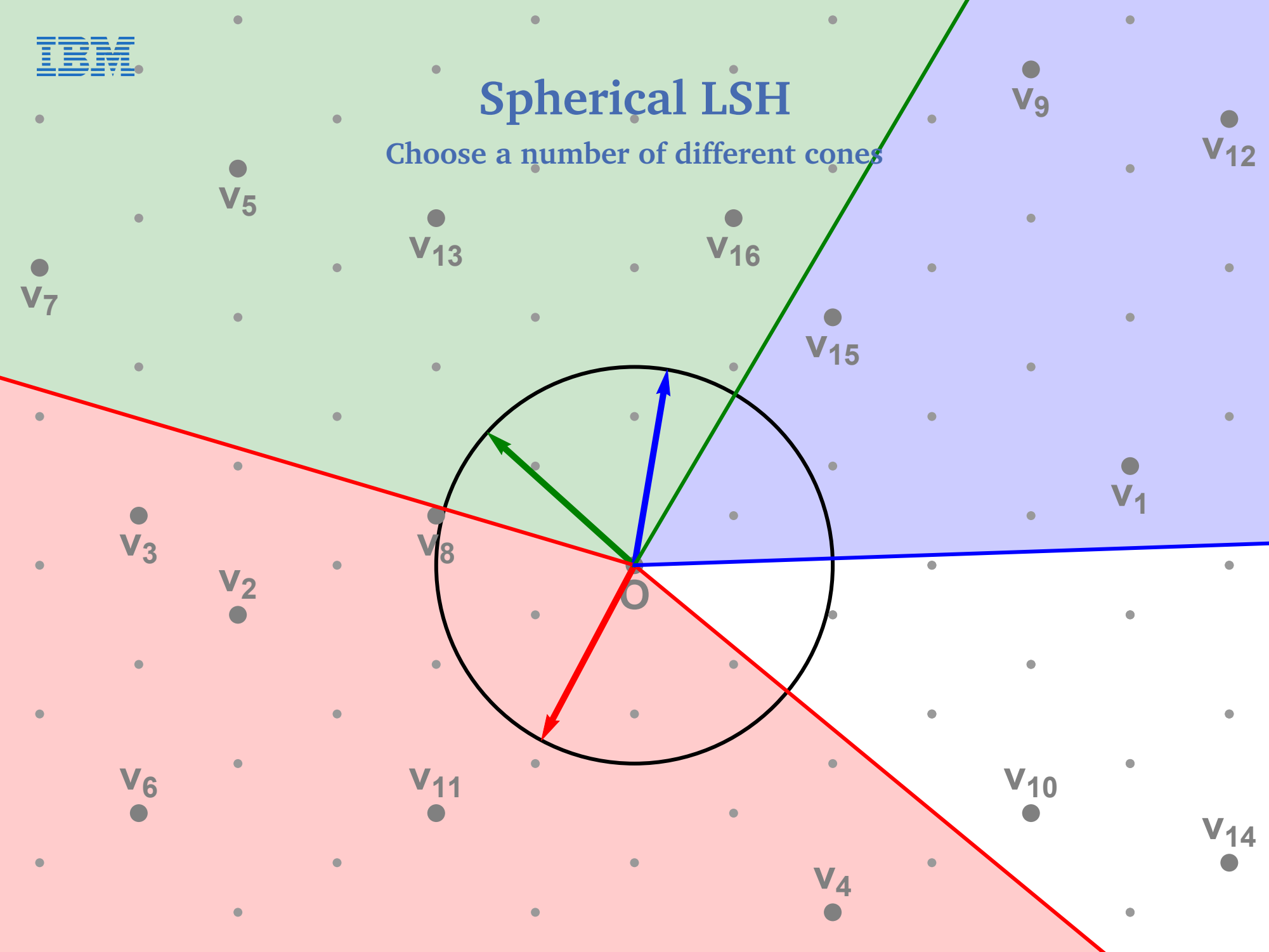
Spherical LSH

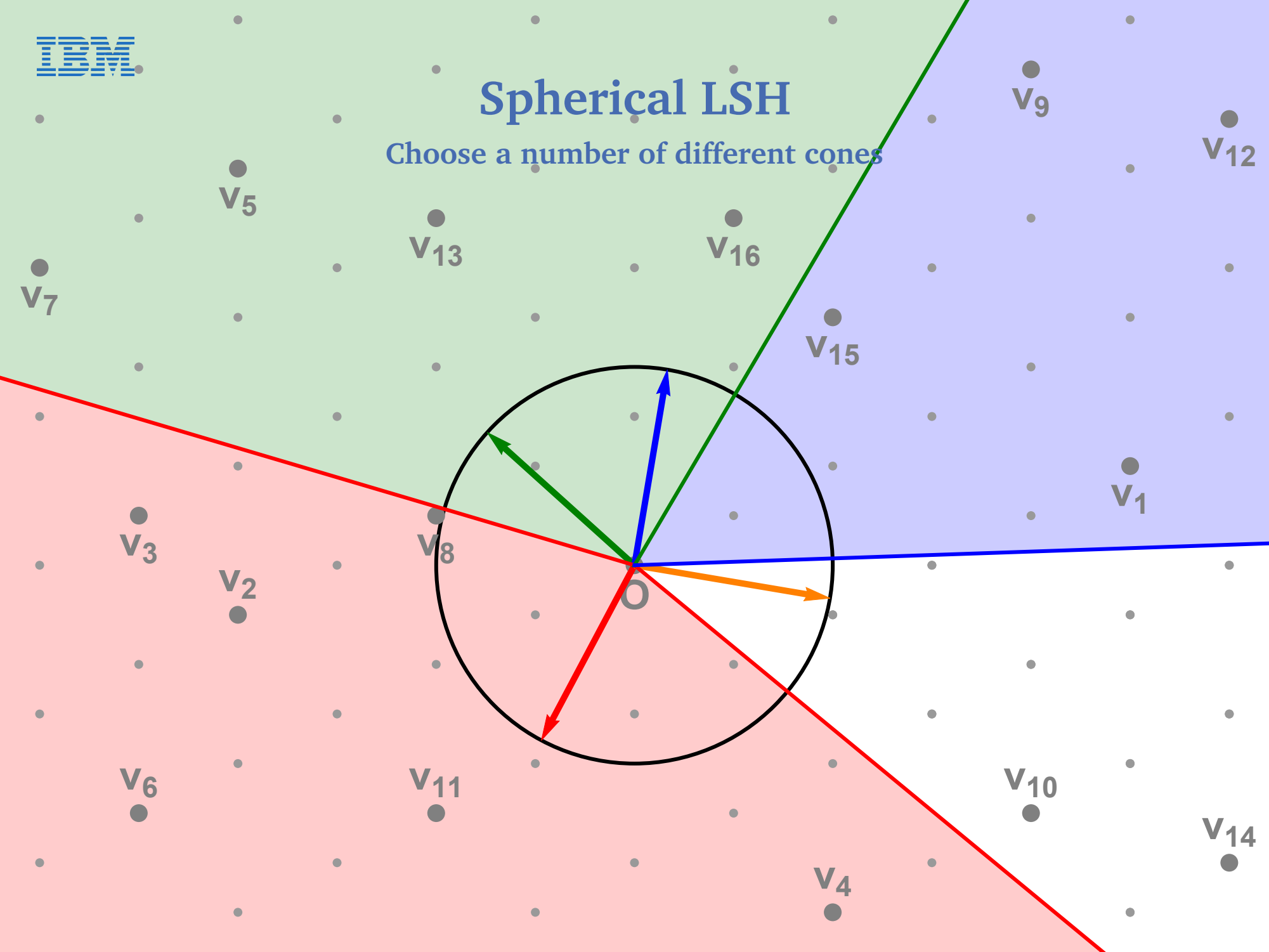
Choose a number of different cones

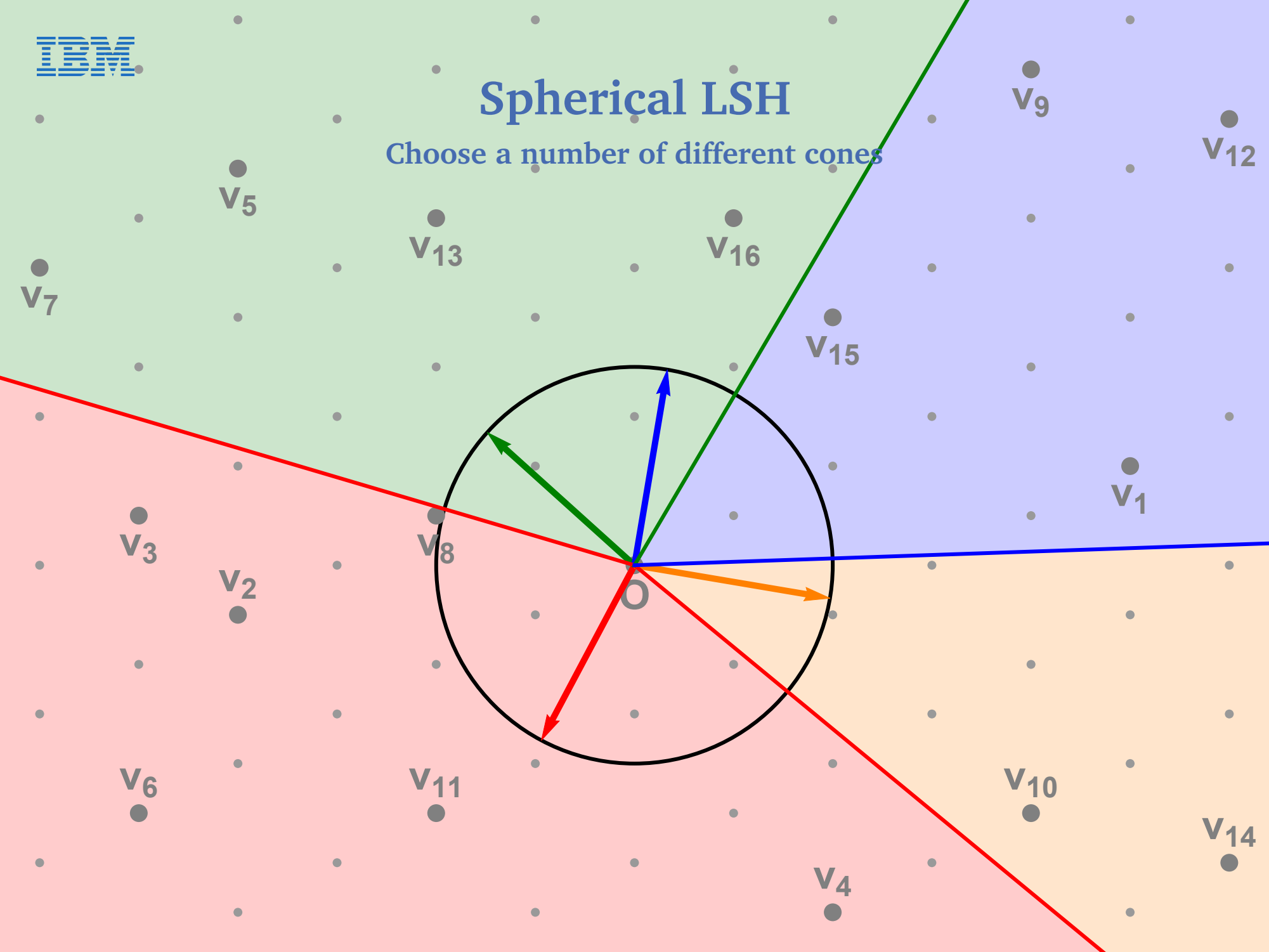


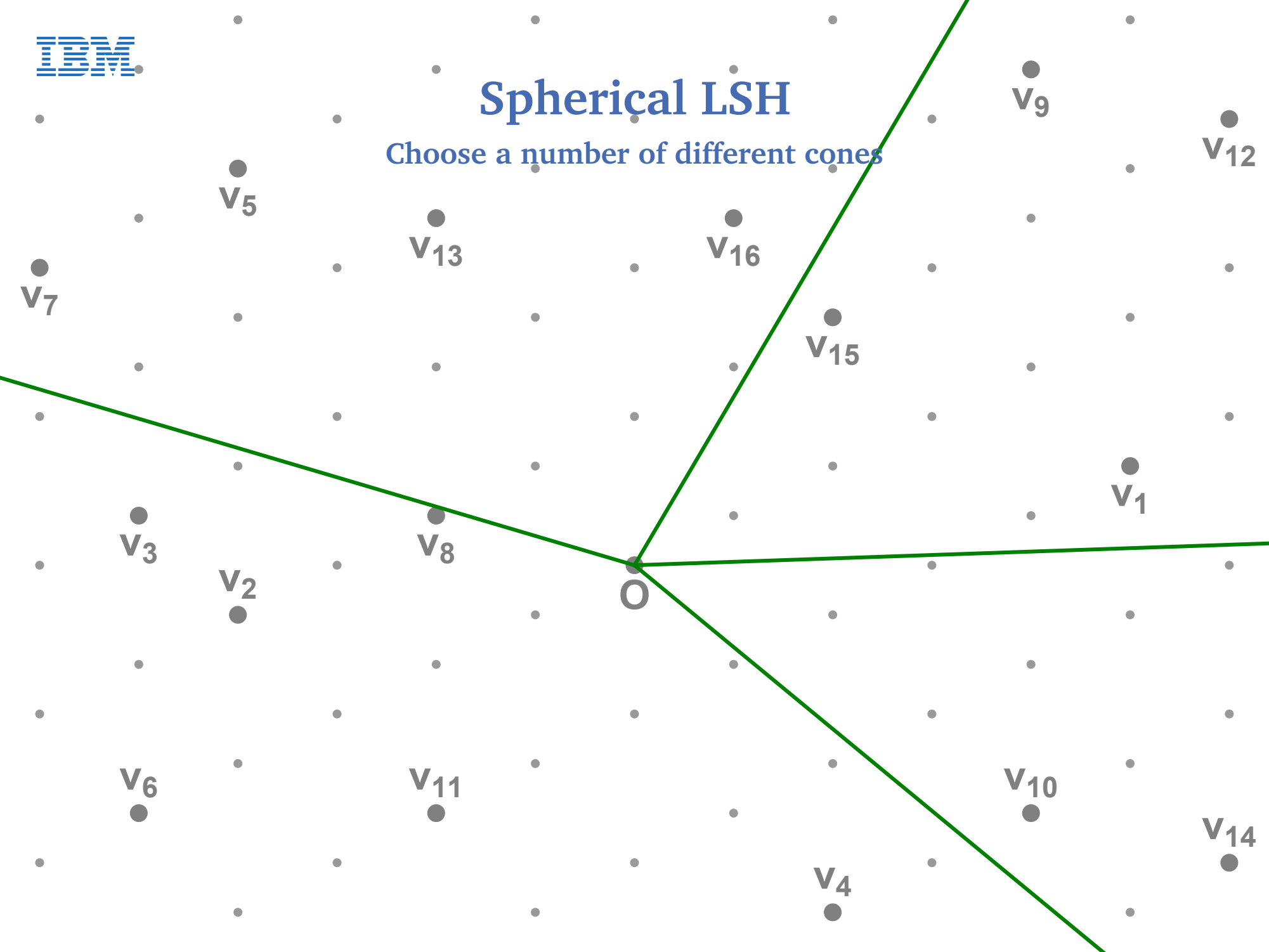

IBM
Spherical LSH
Choose a number of different cones
v5
v13
v16
v9
v12
v7
v15
v1
v3
v8
v2
O
v6
v11
v10
v14
v4


IBM
Spherical LSH
Choose a number of different cones
v5
v13
v16
v9
v12
v7
v15
v1
v3
v8
v2
O
v6
v11
v10
v14
v4


IBM
Spherical LSH
Choose a number of different cones
v5
v13
v16
v9
v12
v7
v15
v1
v3
v8
v2
O
v6
v11
v10
v14
v4


IBM
Spherical LSH
Choose a number of different cones
v5
v13
v16
v9
v12
v7
v15
v1
v3
v8
v2
O
v6
v11
v10
v14
v4


IBM
Spherical LSH
Choose a number of different cones
v5
v13
v16
v9
v12
v7
v15
v1
v3
v8
v2
O
v6
v11
v10
v14
v4

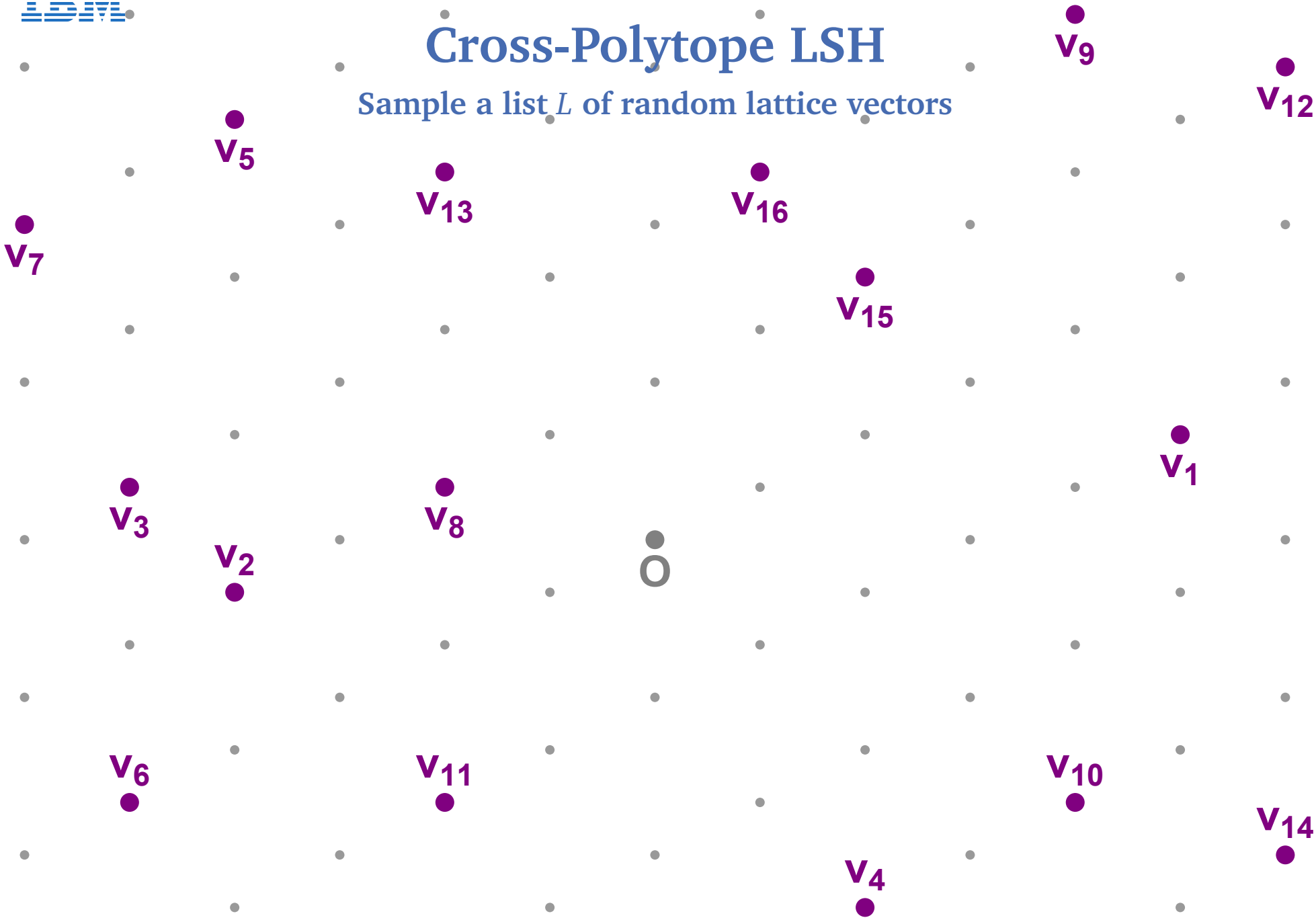

IBM
Spherical LSH
Choose a number of different cones
v5
v13
v16
v9
v12
v7
v15
v1
v3
v8
v2
O
v6
v11
v10
v14
v4

Cross-Polytope LSH

Sample a list L of random lattice vectors

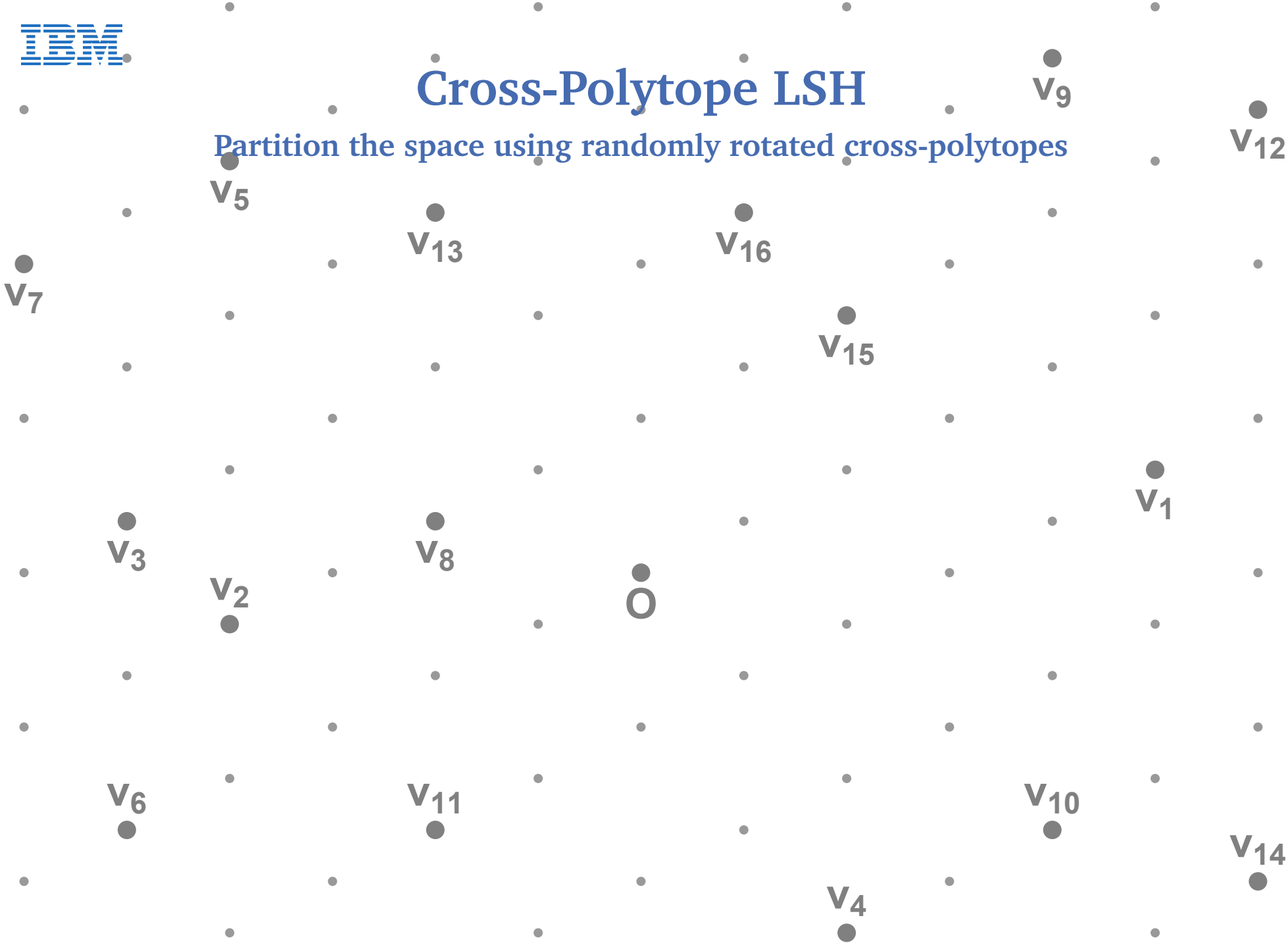
Cross-Polytope LSH

Sample a list L of random lattice vectors



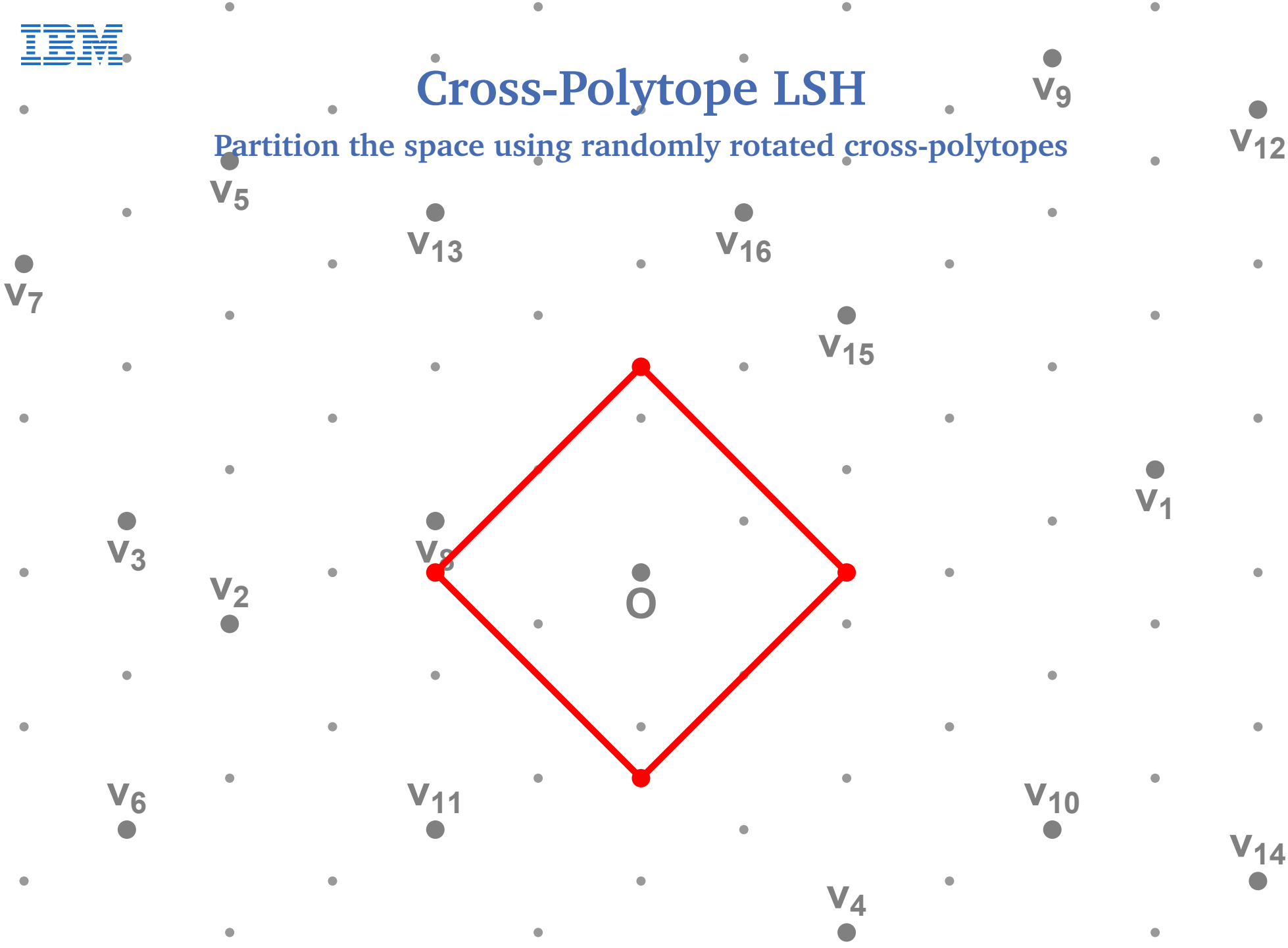
Cross-Polytope LSH

Partition the space using randomly rotated cross-polytopes



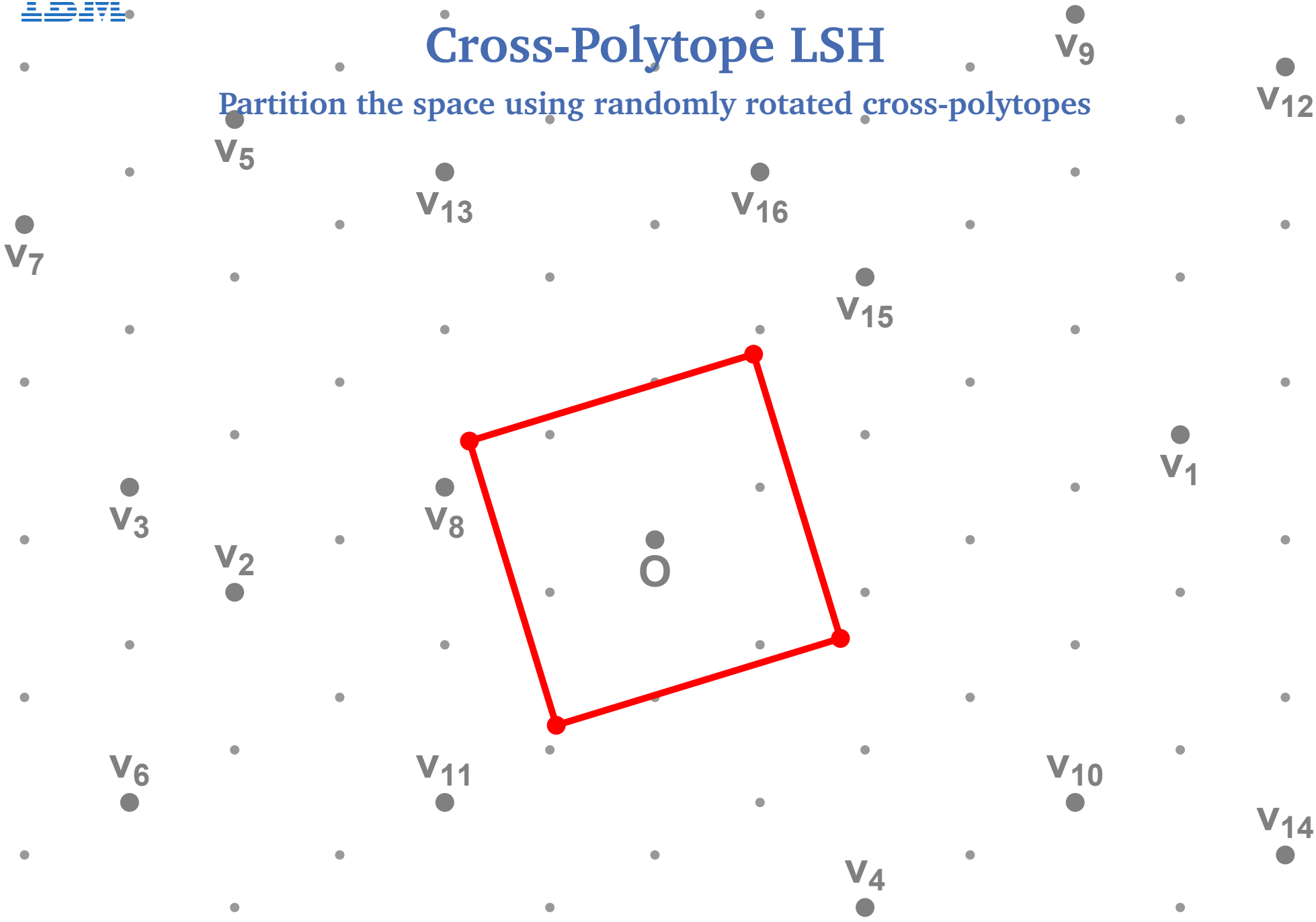
Cross-Polytope LSH

Partition the space using randomly rotated cross-polytopes



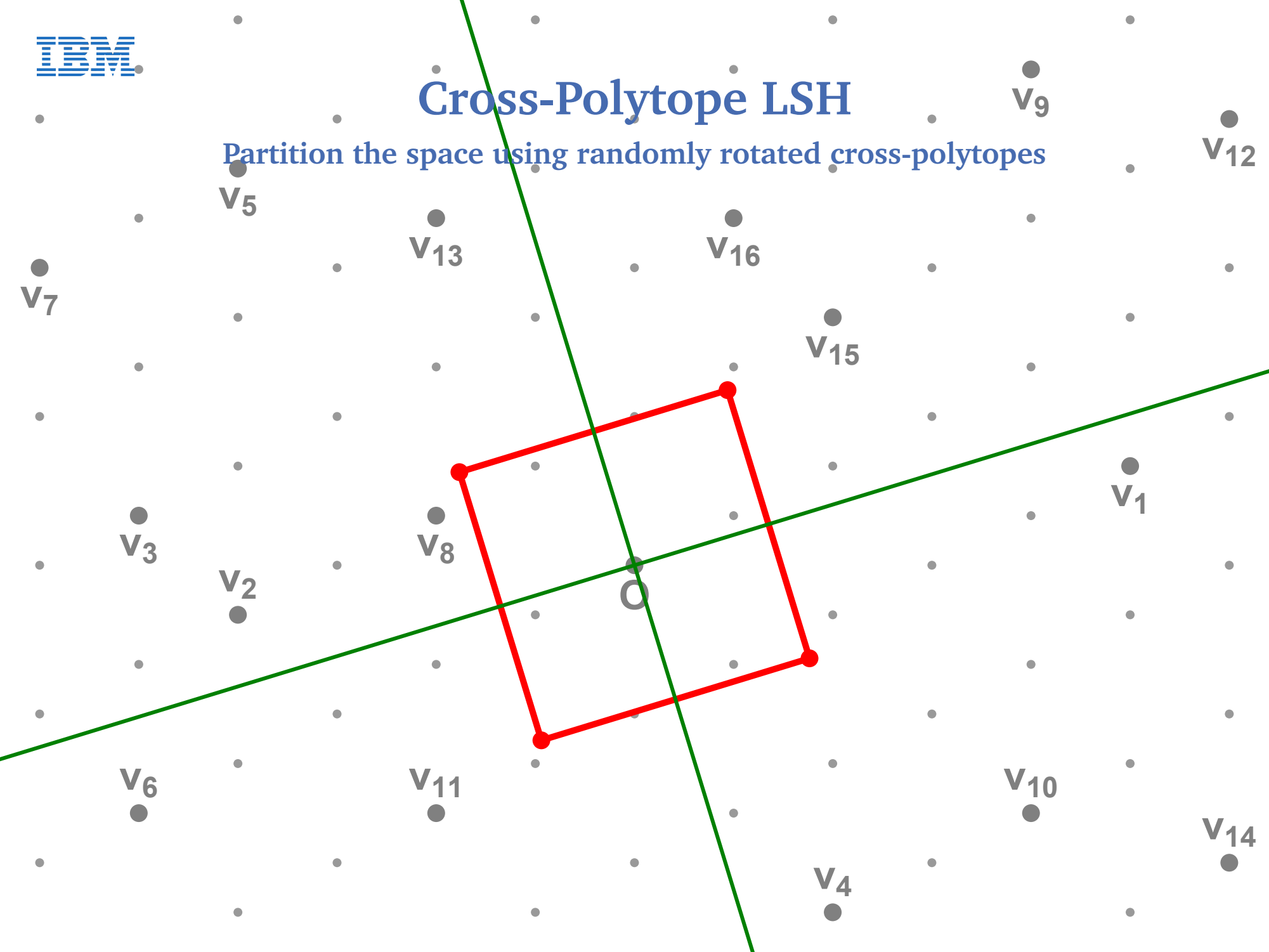
Cross-Polytope LSH

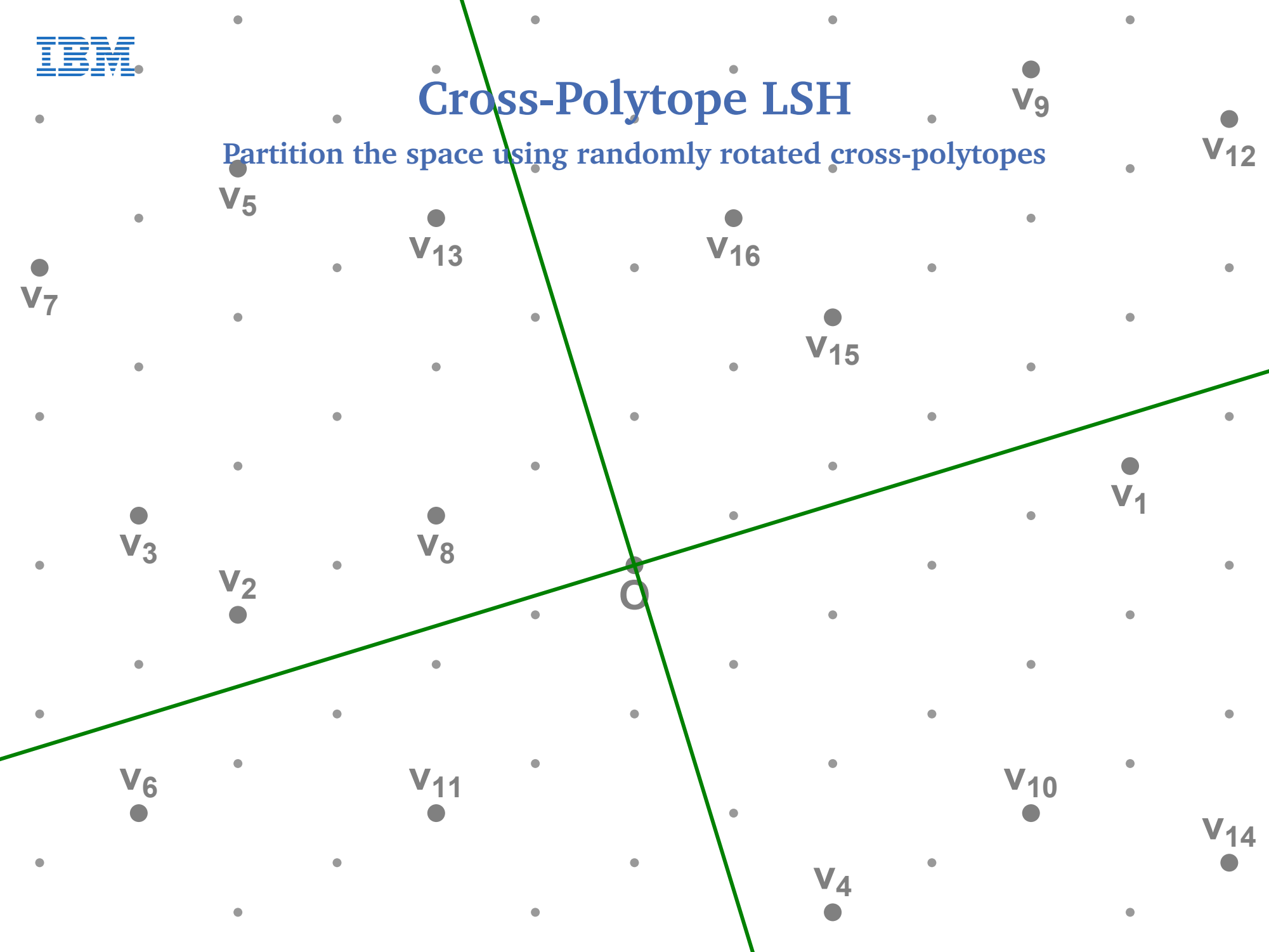
Partition the space using randomly rotated cross-polytopes



Cross-Polytope LSH

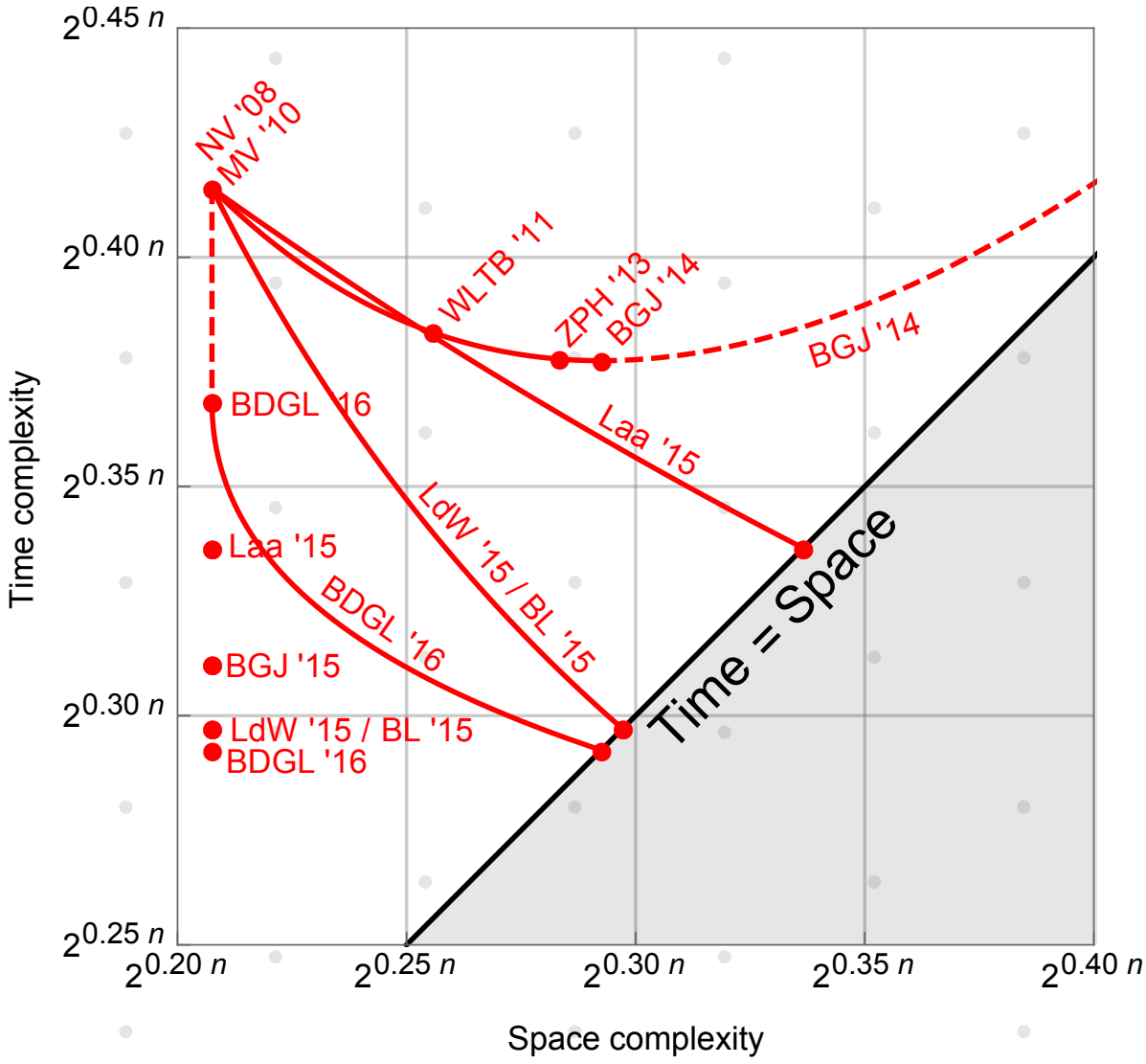
Partition the space using randomly rotated cross-polytopes




IBM
Cross-Polytope LSH
Partition the space using randomly rotated cross-polytopes
v5
v13
v16
v9
v12
v7
v15
v1
v3
v8
v2
O
v6
v11
v10
v14
v4

Sieving algorithms

Overview



Outline

Enumeration algorithms
- Fincke-Pohst enumeration
- Kannan enumeration
- Pruning the enumeration tree

Sieving algorithms
- Nguyen-Vidick sieve
- Multiple levels
- Near neighbor techniques

Practical comparison

IBM

SVP in practice

"We expect our [enumeration] algorithm to be more efficient than lattice sieving up to dimension $n = 1895$."

– Micciancio–Walter, *SODA'15*

SVP in practice

"We expect our [enumeration] algorithm to be more efficient than lattice sieving up to dimension $n = 1895$."

– Micciancio–Walter, *SODA'15*

"As far as I know, everyone who has tried sieving as a BKZ subroutine in place of enumeration has concluded that sieving is much too slow to be useful—the cutoff is beyond cryptographically relevant sizes."

– Bernstein, *Google groups '16*

SVP in practice

"We expect our [enumeration] algorithm to be more efficient than lattice sieving up to dimension $n = 1895$."

– Micciancio–Walter, *SODA'15*

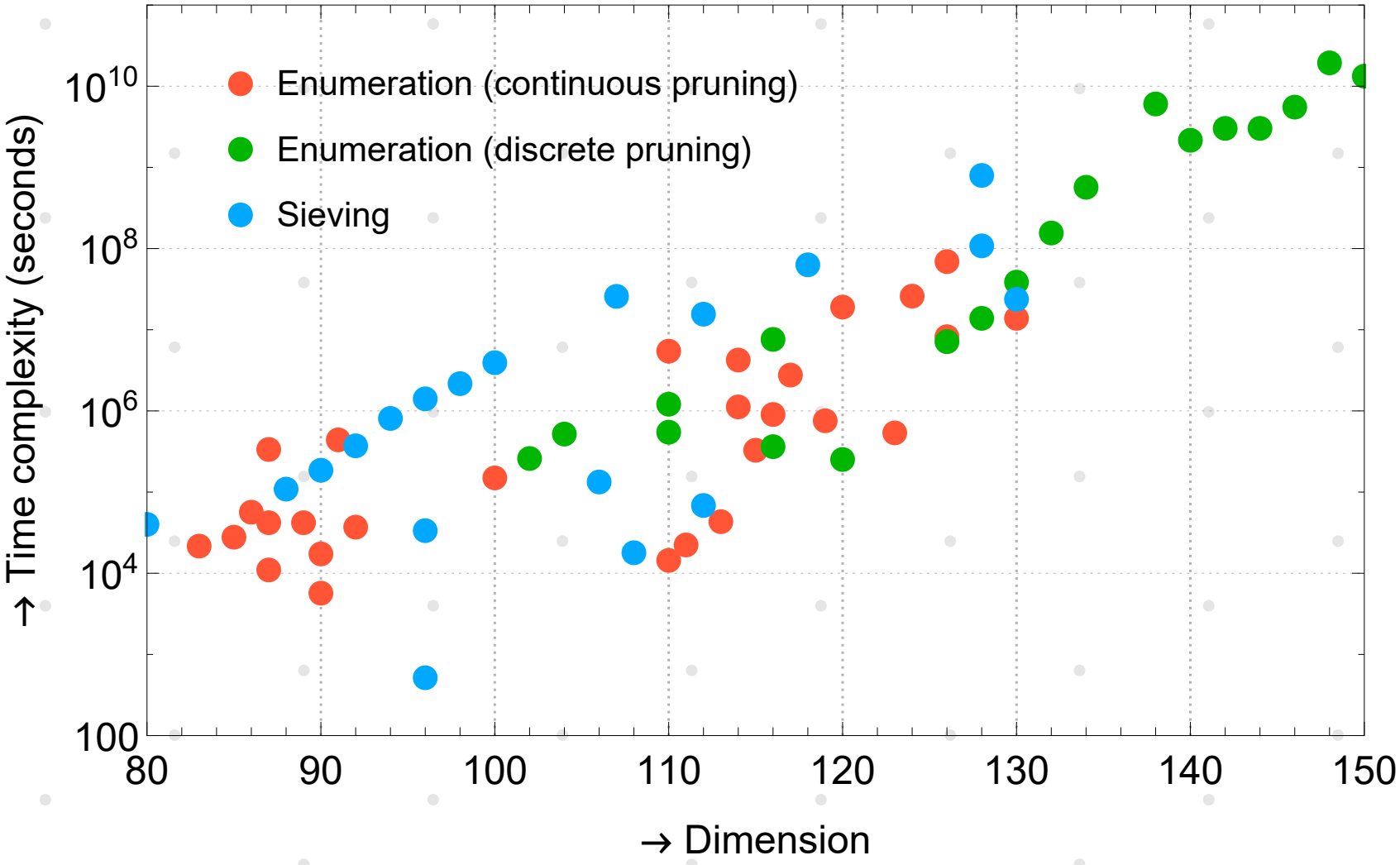
"As far as I know, everyone who has tried sieving as a BKZ subroutine in place of enumeration has concluded that sieving is much too slow to be useful—the cutoff is beyond cryptographically relevant sizes."

– Bernstein, *Google groups '16*

"I compute a cross-over point between enumeration and the HashSieve at dimension $b = 217$."

– Ducas, *Google groups '16*

SVP in practice



IBM

Summary

- Lattice-based crypto relies on hardness of finding short bases
- State-of-the-art basis reduction: BKZ with fast SVP subroutine
- Enumeration for SVP:
  - Memory-efficient
  - Best in low dimensions
  - Fast pruning heuristics
- Sieving for SVP:
  - Large memory requirement
  - Fastest in high dimensions
  - Practical near neighbor speedups
- Enumeration still leading, but sieving is catching up!

Questions?