TU/e Technische Universiteit Eindhoven University of Technology

Combining lattice sieving algorithms with (quantum) nearest neighbor searching

Thijs Laarhoven

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

Dagstuhl seminar, Wadern, Germany
(September 11, 2015)

TU/e

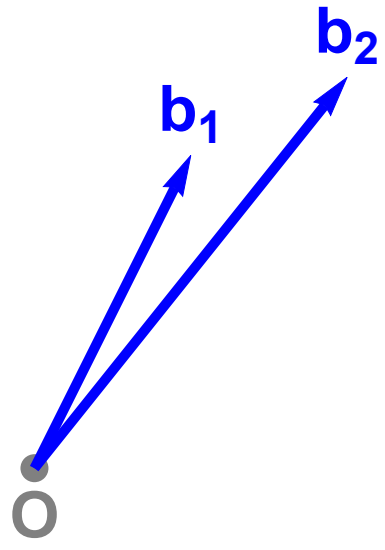
Lattices

What is a lattice?

TU/e

Lattices

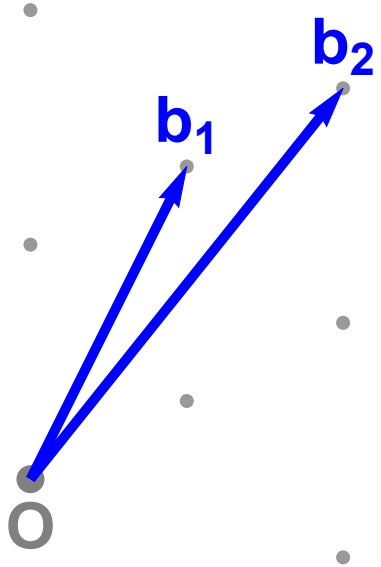
What is a lattice?



TU/e

Lattices

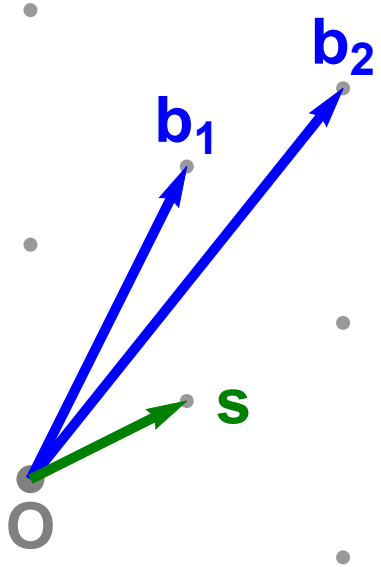
What is a lattice?



TU/e

Lattices

Shortest Vector Problem (SVP)



TU/e

Lattices

Exact SVP algorithms

	Algorithm	$\log_2$(Time)	$\log_2$(Space)
Provable SVP	Enumeration [Poh81, Kan83, ..., GNR10]	$\Omega(n \log n)$	$O(\log n)$
	AKS-sieve [AKS01, NV08, MV10, HPS11]	$3.398n$	$1.985n$
	ListSieve [MV10, MDB14]	$3.199n$	$1.327n$
	AKS-sieve-birthday [PS09, HPS11]	$2.648n$	$1.324n$
	ListSieve-birthday [PS09]	$2.465n$	$1.233n$
	Voronoi cell algorithm [MV10b]	$2.000n$	$1.000n$
Heuristic SVP	Nguyen-Vidick sieve [NV08]	$0.415n$	$0.208n$
	GaussSieve [MV10, ..., IKMT14, BNvdP14]	$0.415n$?	$0.208n$
	Two-level sieve [WLTB11]	$0.384n$	$0.256n$
	Three-level sieve [ZPH13]	$0.3778n$	$0.283n$

TU/e

Lattices

Exact SVP algorithms

	Algorithm	$\log_2$(Time)	$\log_2$(Space)
Provable SVP	Enumeration [Poh81, Kan83, ..., GNR10]	$\Omega(n \log n)$	$O(\log n)$
	AKS-sieve [AKS01, NV08, MV10, HPS11]	$3.398n$	$1.985n$
	ListSieve [MV10, MDB14]	$3.199n$	$1.327n$
	AKS-sieve-birthday [PS09, HPS11]	$2.648n$	$1.324n$
	ListSieve-birthday [PS09]	$2.465n$	$1.233n$
	Voronoi cell algorithm [MV10b]	$2.000n$	$1.000n$
	Discrete Gaussian sampling [ADRS15]	$1.000n$	$1.000n$
Heuristic SVP	Nguyen-Vidick sieve [NV08]	$0.415n$	$0.208n$
	GaussSieve [MV10, ..., IKMT14, BNvdP14]	$0.415n$?	$0.208n$
	Two-level sieve [WLTB11]	$0.384n$	$0.256n$
	Three-level sieve [ZPH13]	$0.3778n$	$0.283n$
	Overlattice sieving [BGJ14]	$0.3774n$	$0.293n$
	Hyperplane LSH [Laa15, MLB15]	$0.337n$	$0.208n$
	May and Ozerov's NNS method [BGJ15]	$0.311n$	$0.208n$
	Spherical LSH [LdW15]	$0.298n$	$0.208n$
	Cross-polytope LSH [BL15]	$0.298n$	$0.208n$
	Spherical filtering [BDGL15]	$0.293n$	$0.208n$

TU/e

Nguyen-Vidick sieve

TU/e

Nguyen-Vidick sieve

1. Sample a list L of random lattice vectors

TU/e

Nguyen-Vidick sieve

1. Sample a list L of random lattice vectors

TU/e

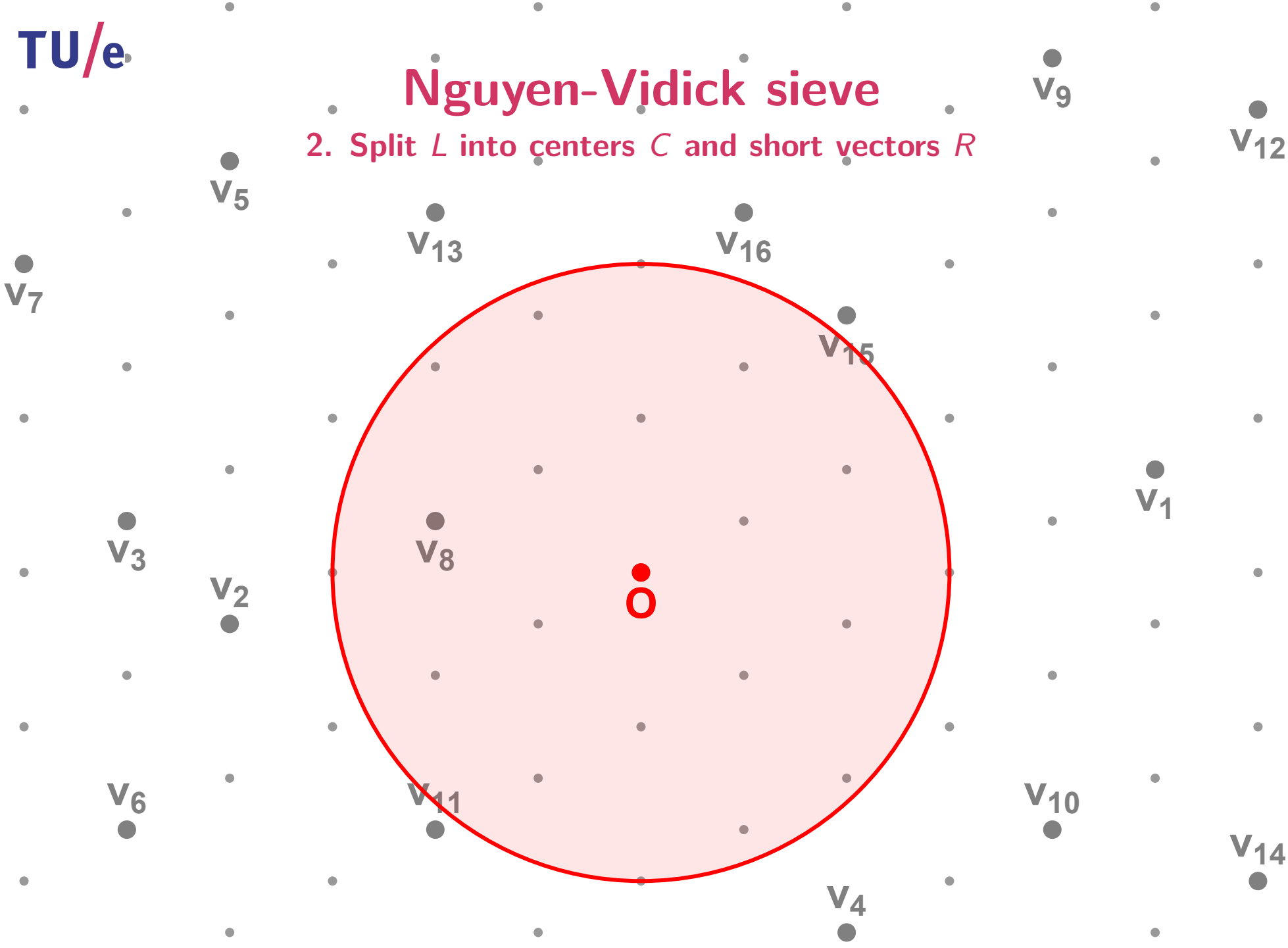
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R

TU/e

Nguyen-Vidick sieve

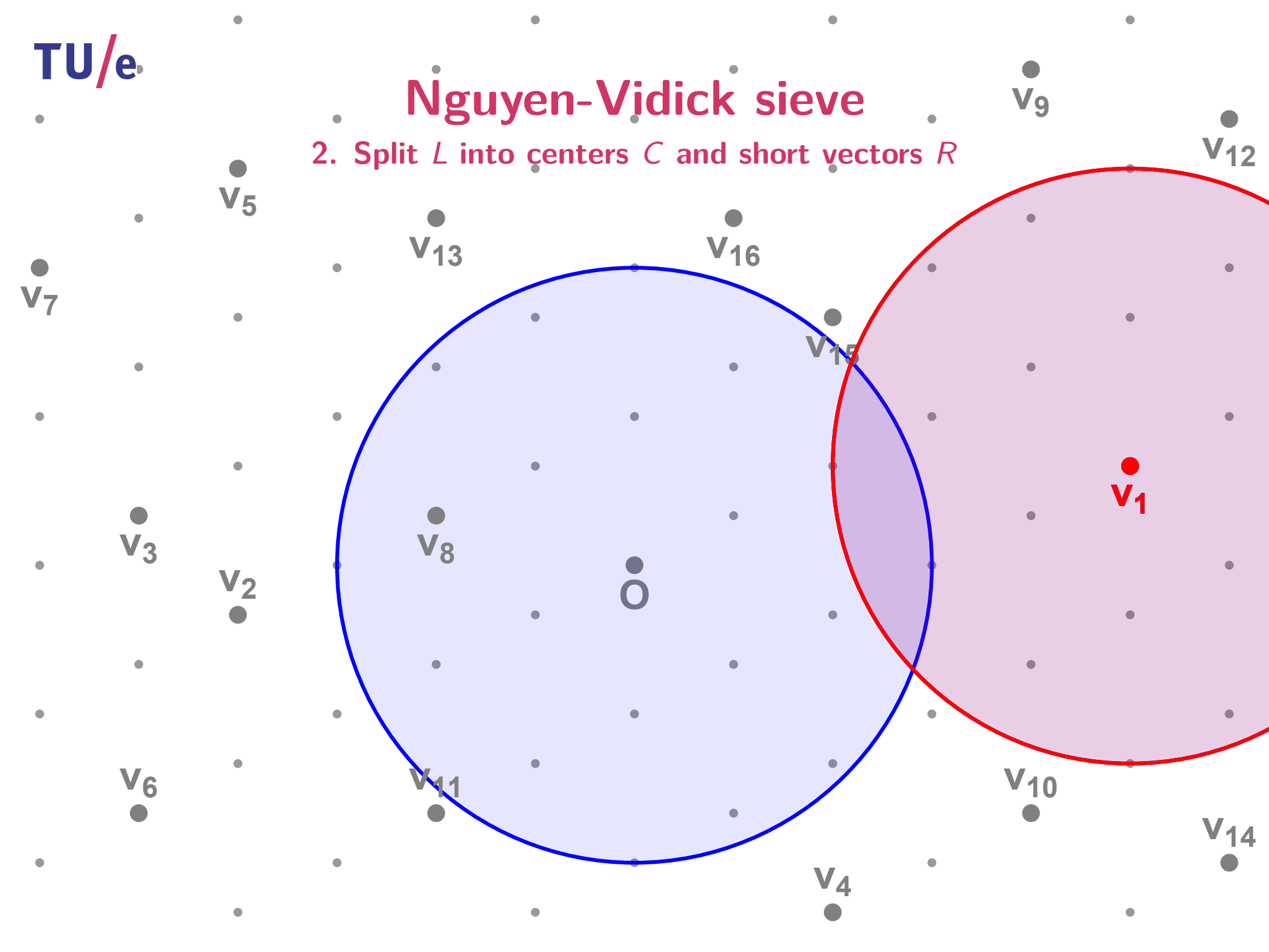
2. Split L into centers C and short vectors R



TU/e

Nguyen-Vidick sieve

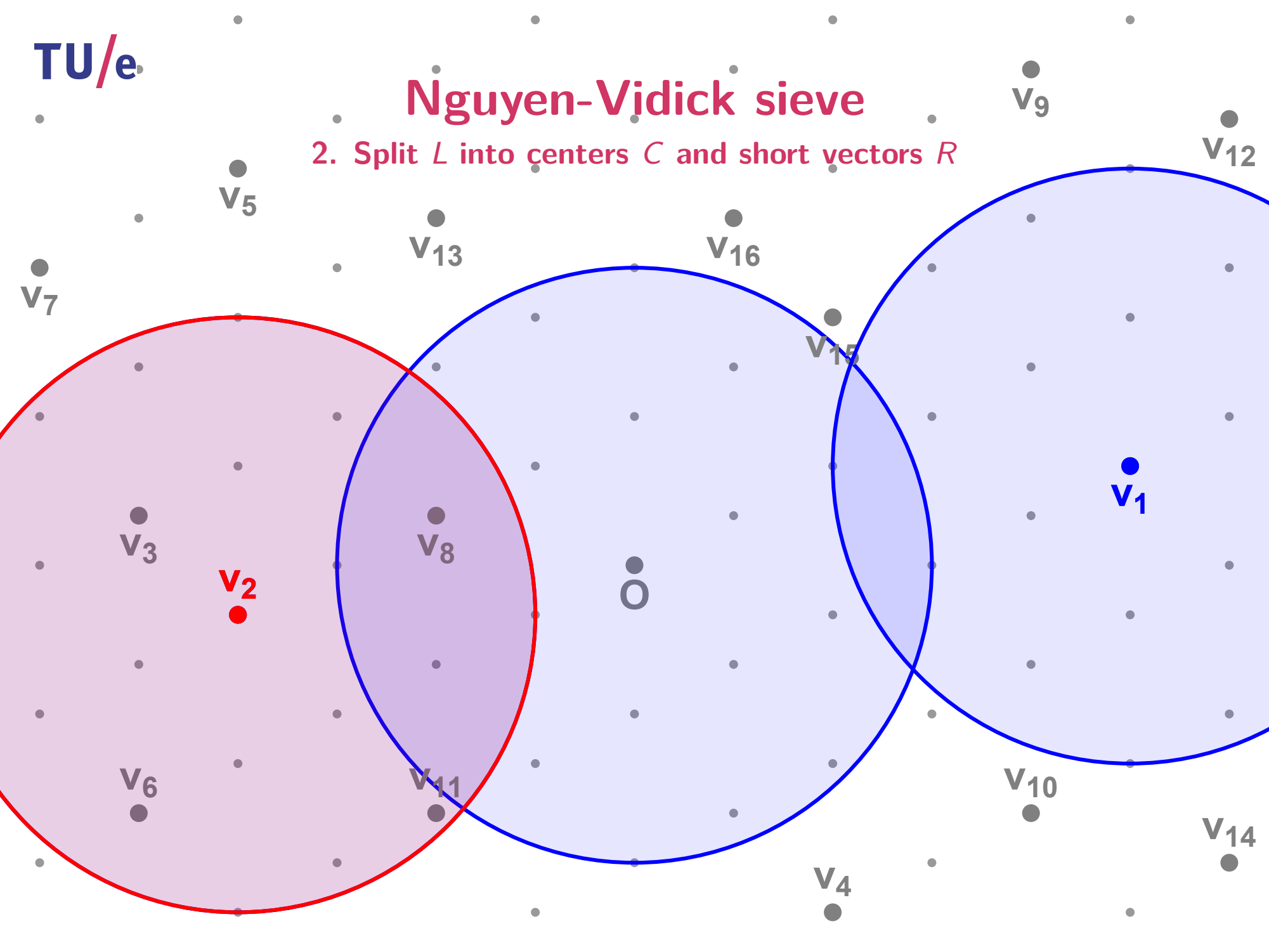
2. Split L into centers C and short vectors R



TU/e

Nguyen-Vidick sieve

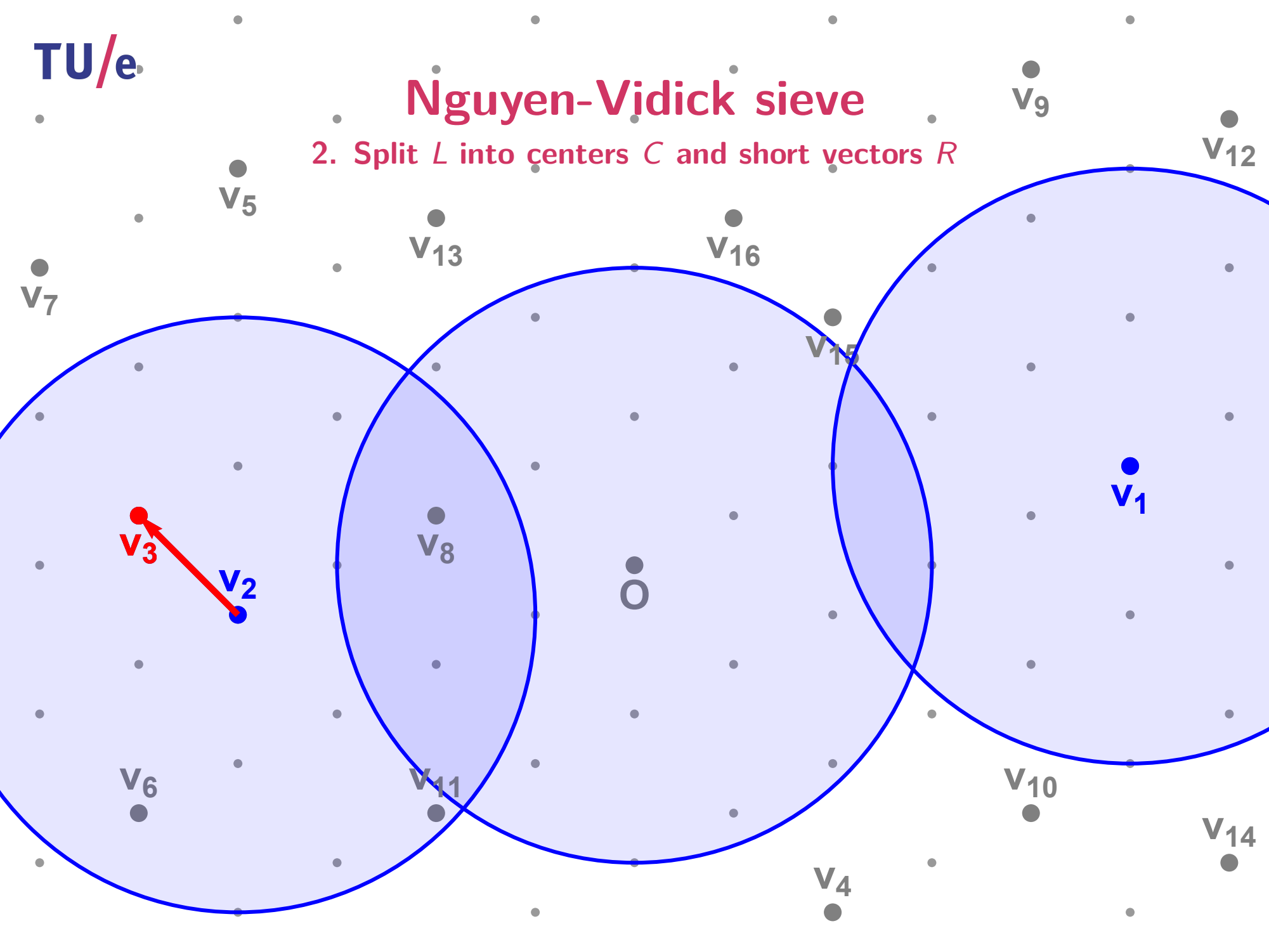
2. Split L into centers C and short vectors R



TU/e

Nguyen-Vidick sieve

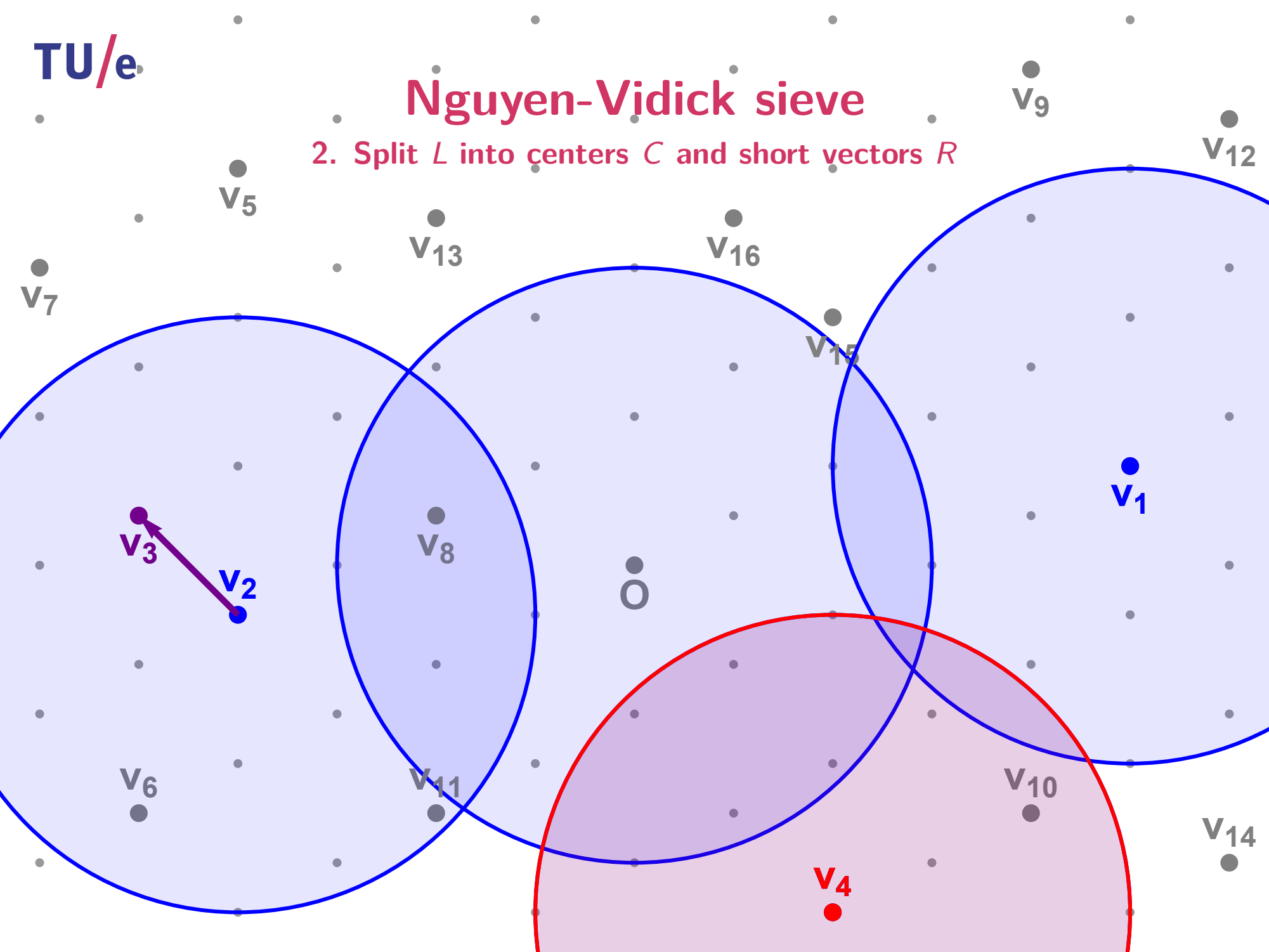
2. Split L into centers C and short vectors R



TU/e

Nguyen-Vidick sieve

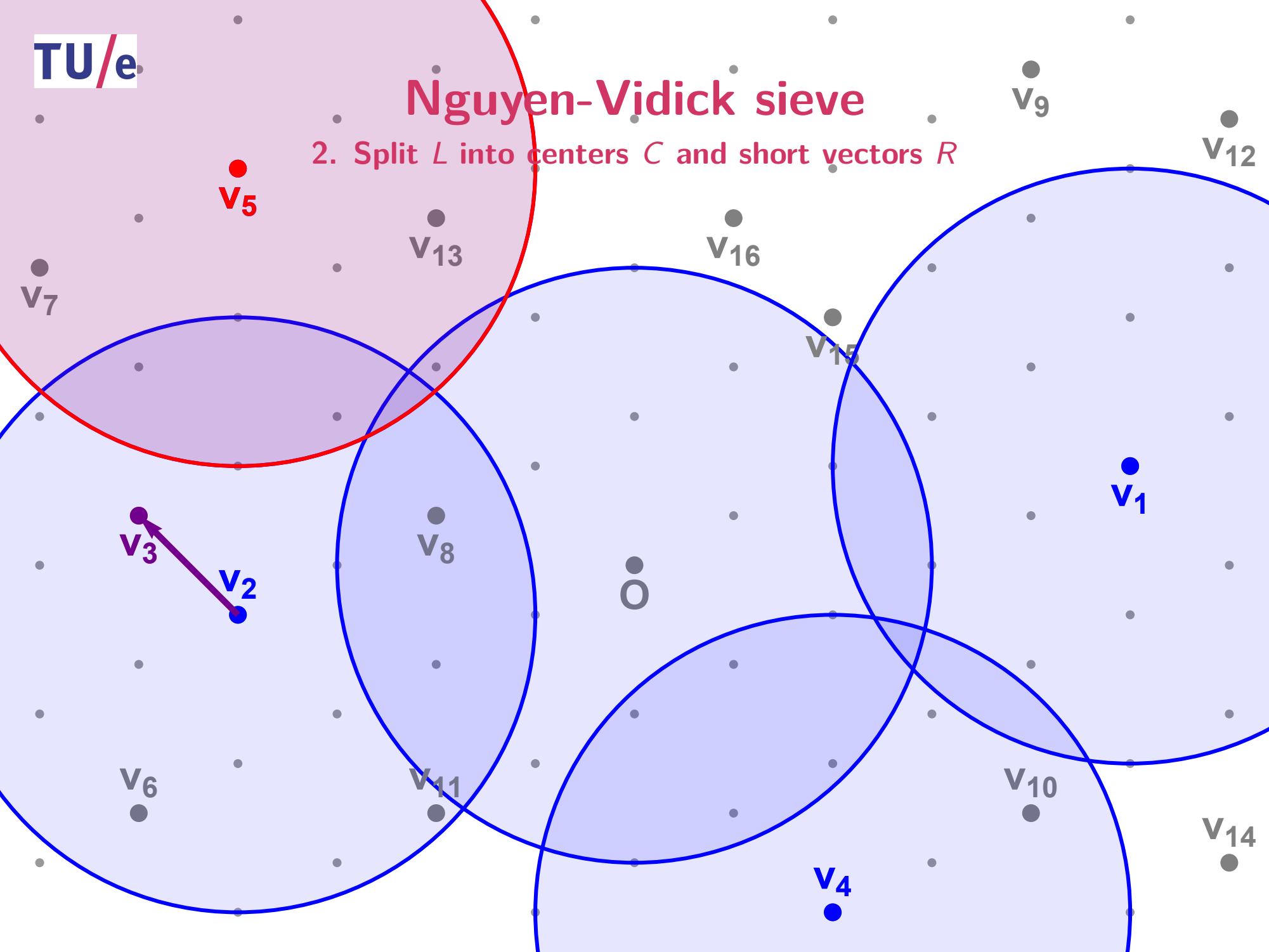
2. Split L into centers C and short vectors R



TU/e

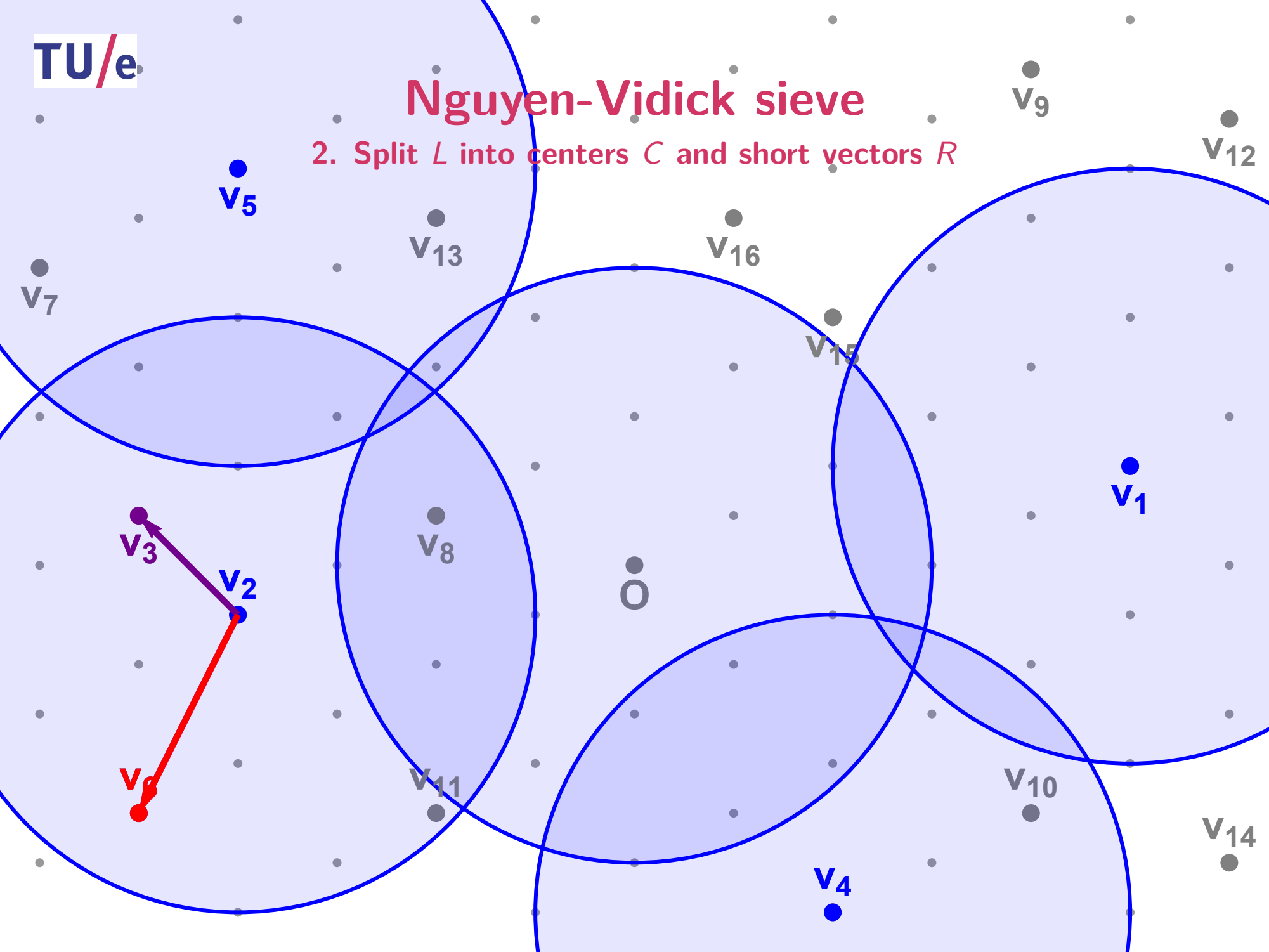
Nguyen-Vidick sieve

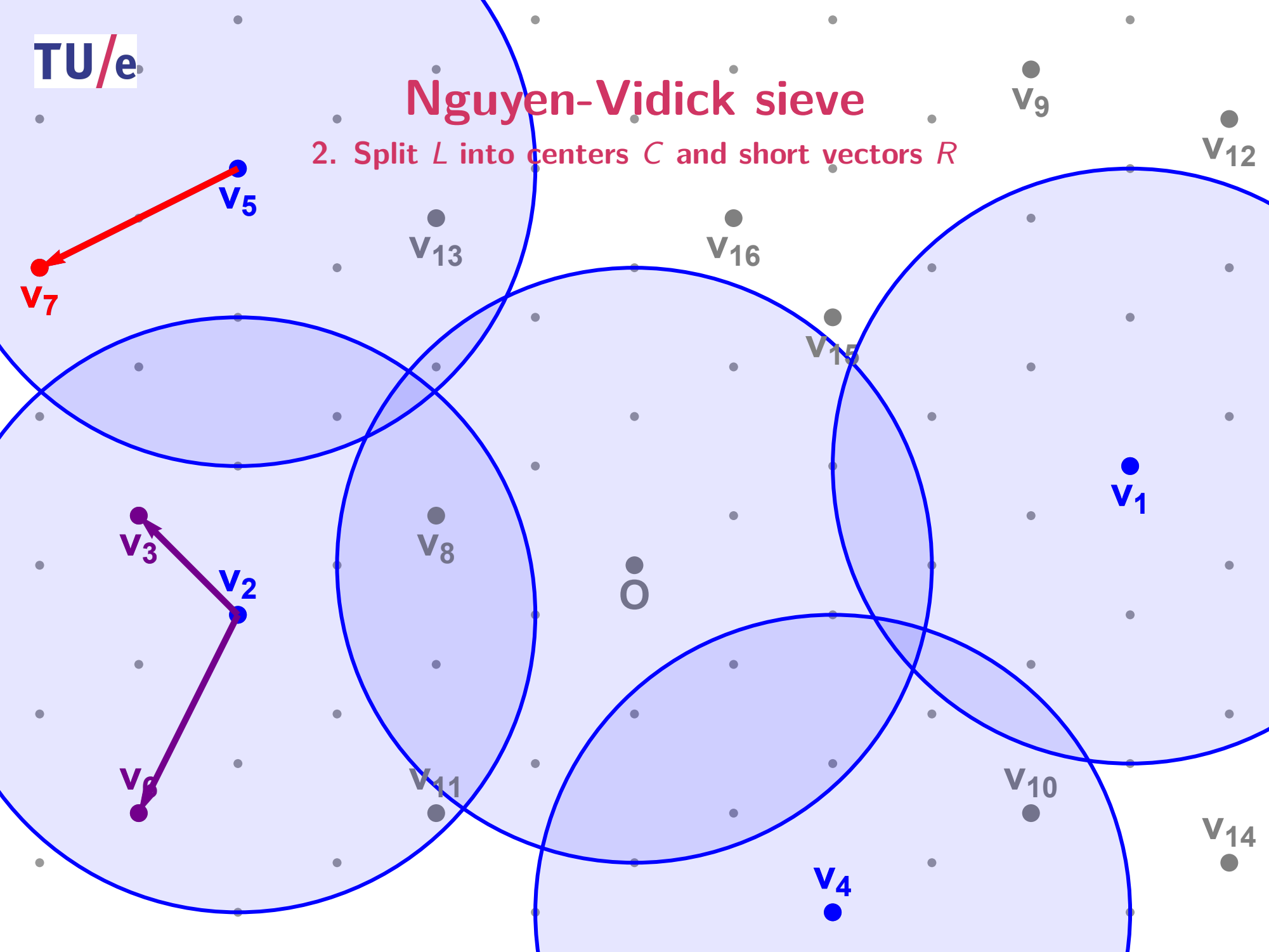
2. Split L into centers C and short vectors R



Nguyen-Vidick sieve

2. Split L into centers C and short vectors R

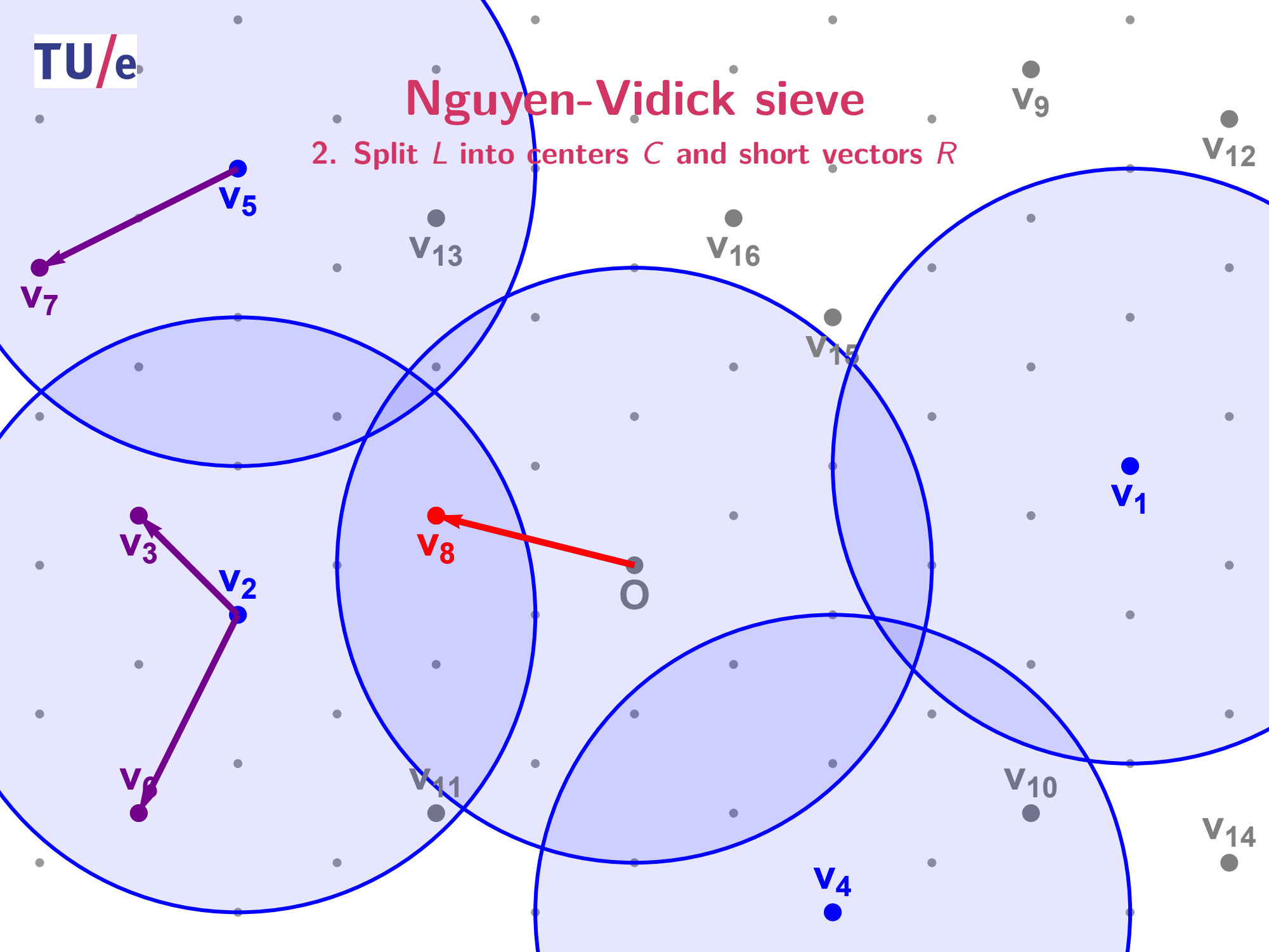


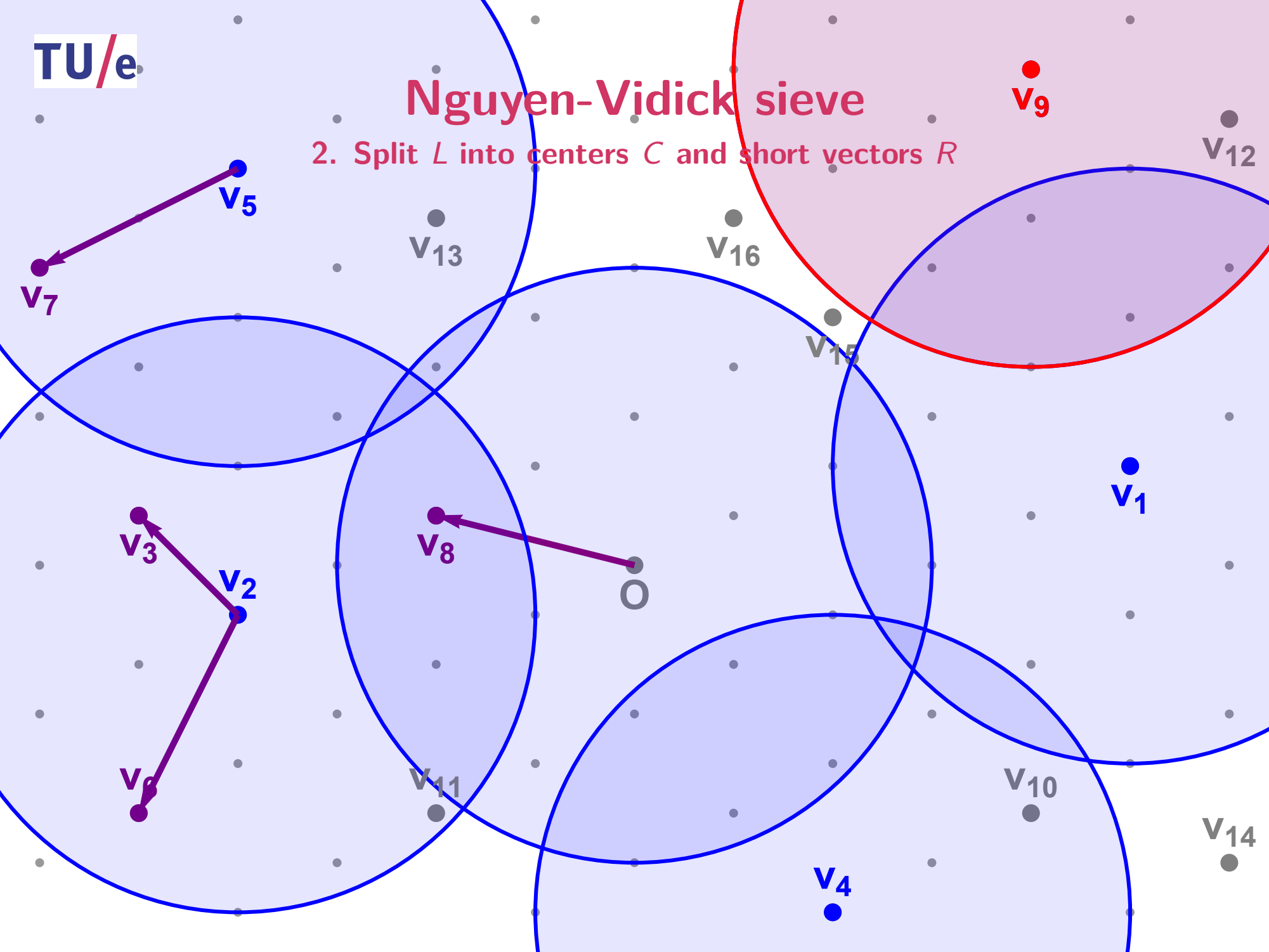

TU/e
Nguyen-Vidick sieve
2. Split L into centers C and short vectors R
v_5
v_7
v_{13}
v_{16}
v_9
v_{12}
v_{15}
v_1
v_3
v_2
v_8
O
v_6
v_{11}
v_{10}
v_{14}
v_4

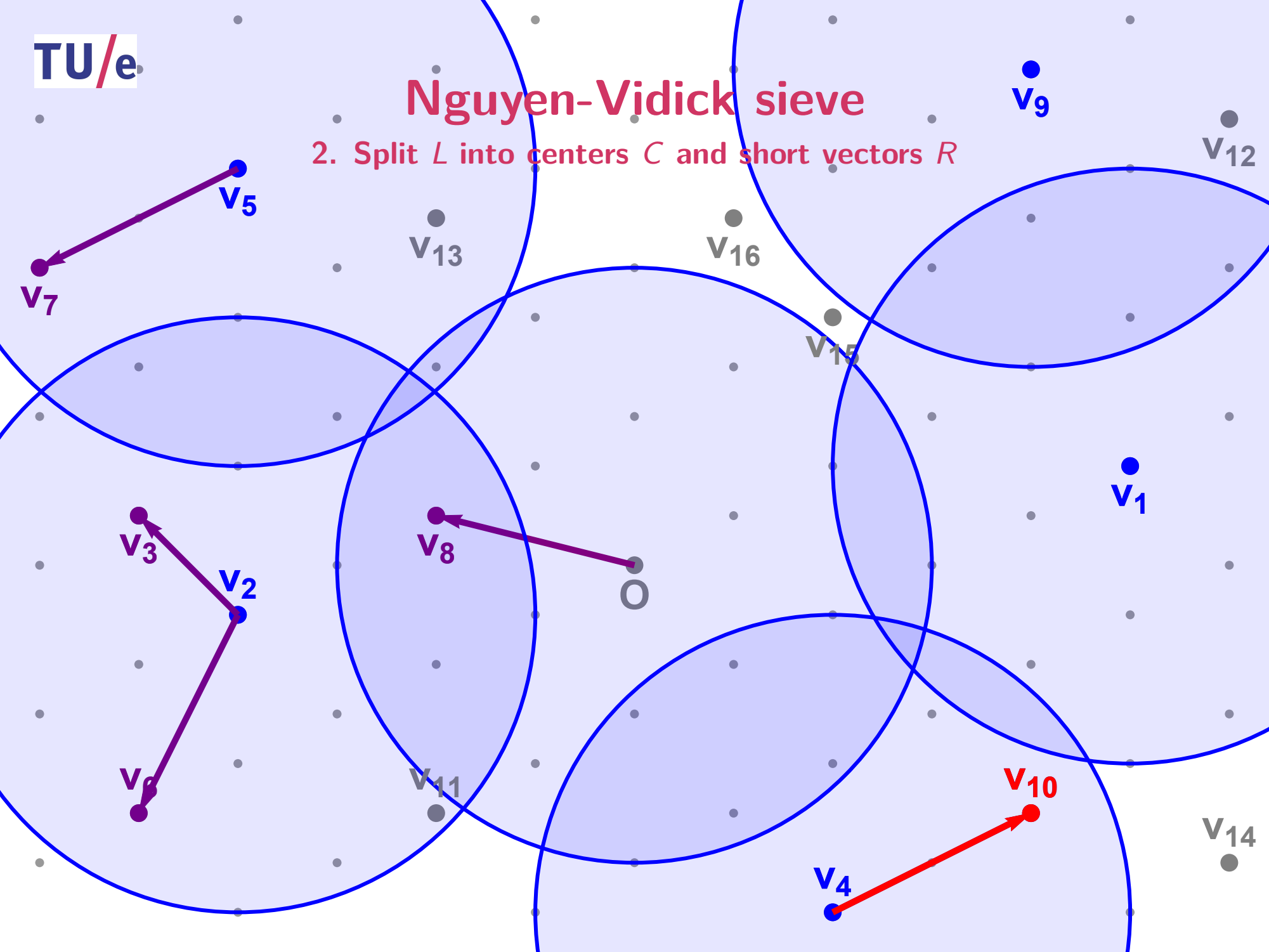
TU/e

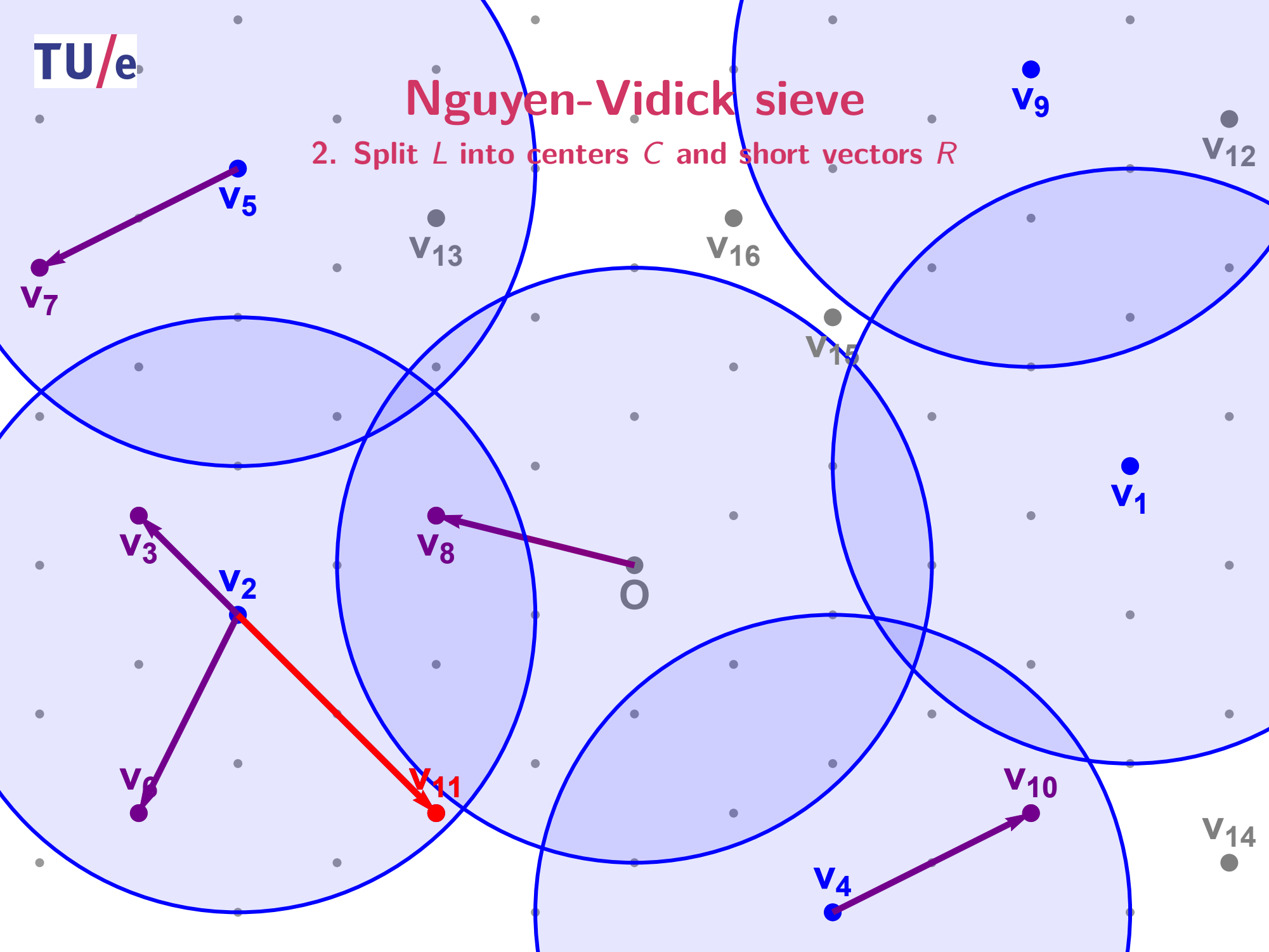
Nguyen-Vidick sieve

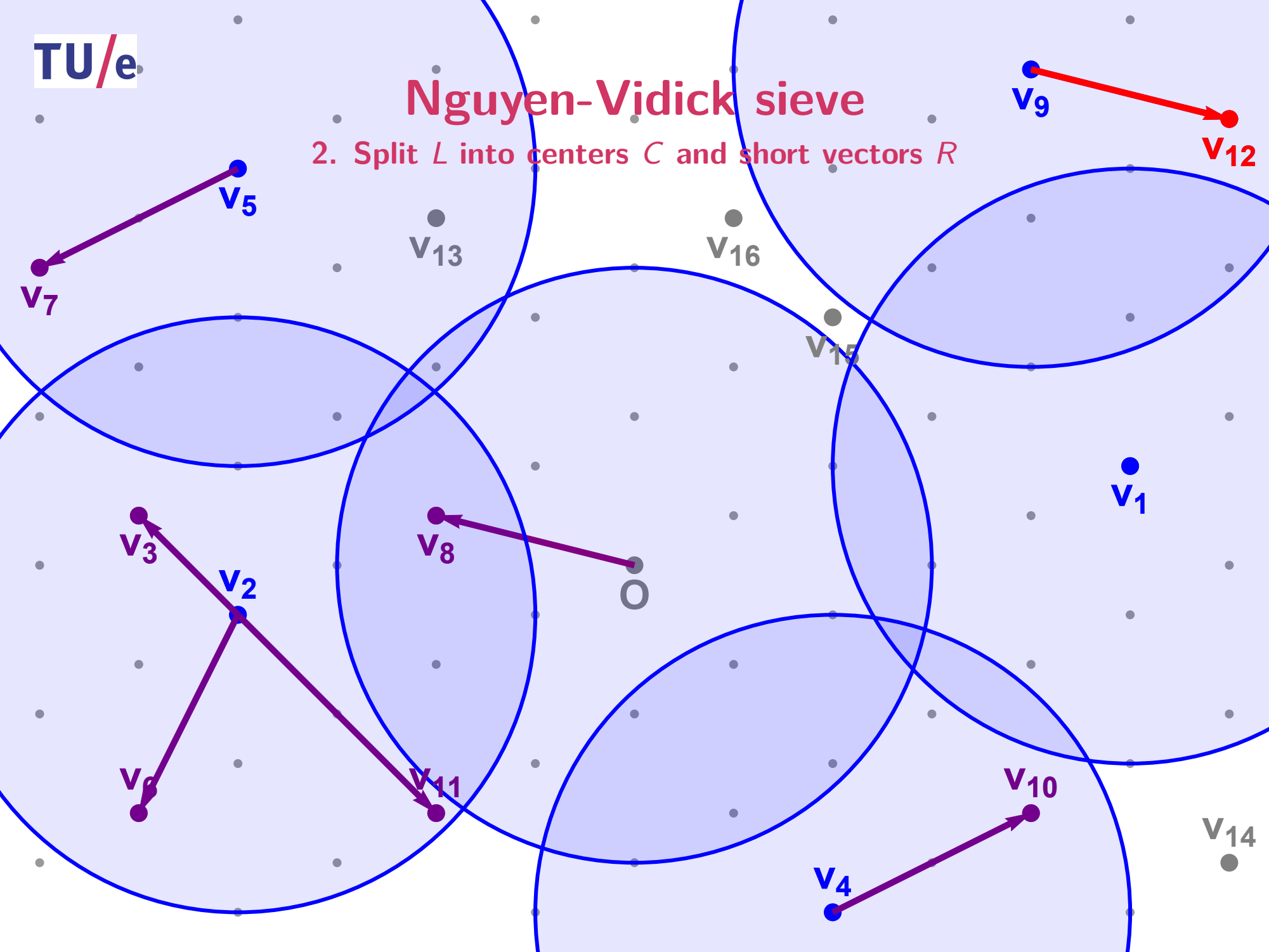
2. Split L into centers C and short vectors R

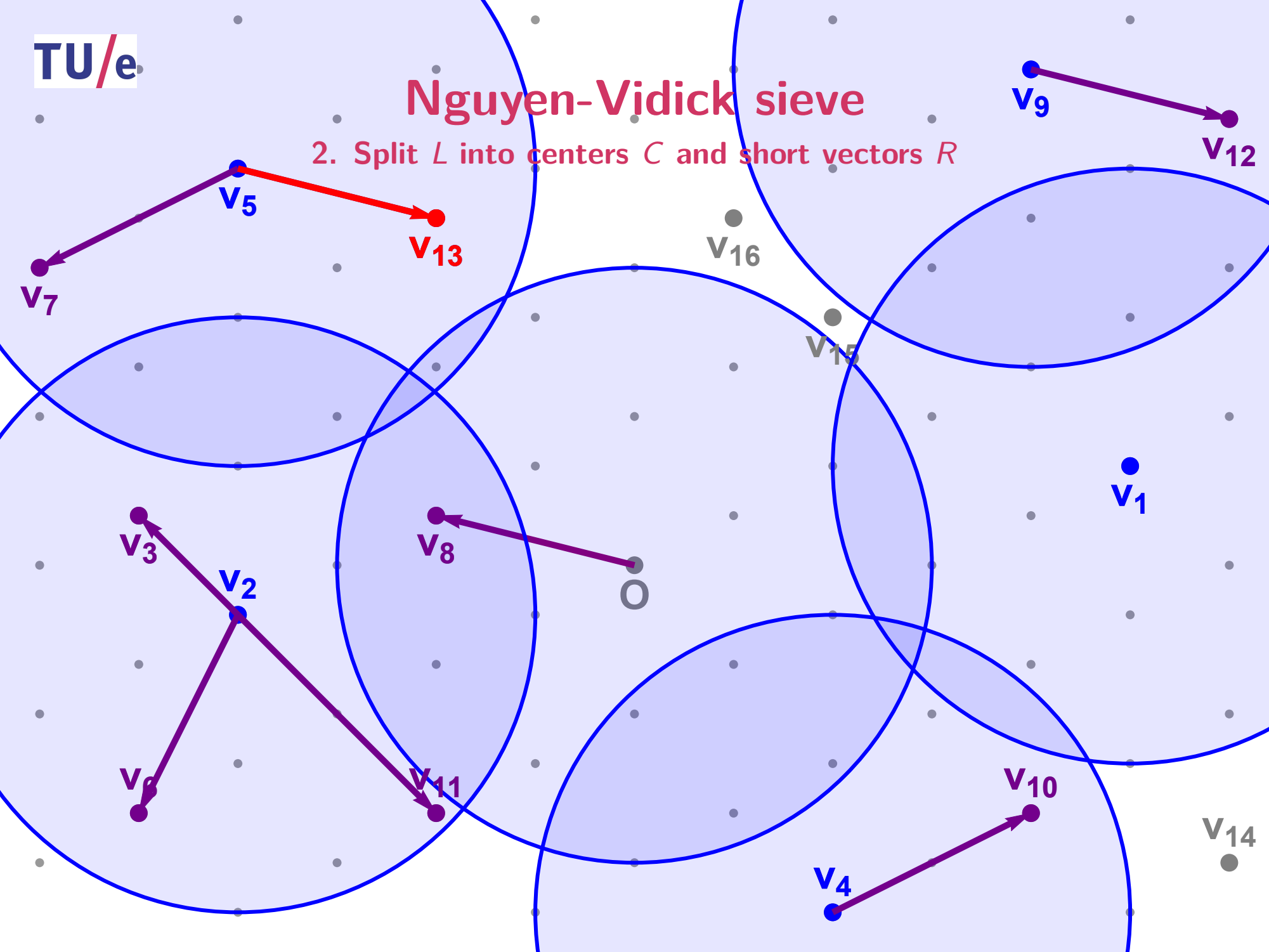



TU/e
Nguyen-Vidick sieve
2. Split L into centers C and short vectors R
v_9
v_{12}
v_5
v_{13}
v_{16}
v_7
v_{15}
v_1
v_3
v_8
v_2
O
v_6
v_{11}
v_{10}
v_{14}
v_4


TU/e
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_{11}
v_{10}
v_{14}
v_4


TU/e
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_{11}
v_{10}
v_{14}
v_4

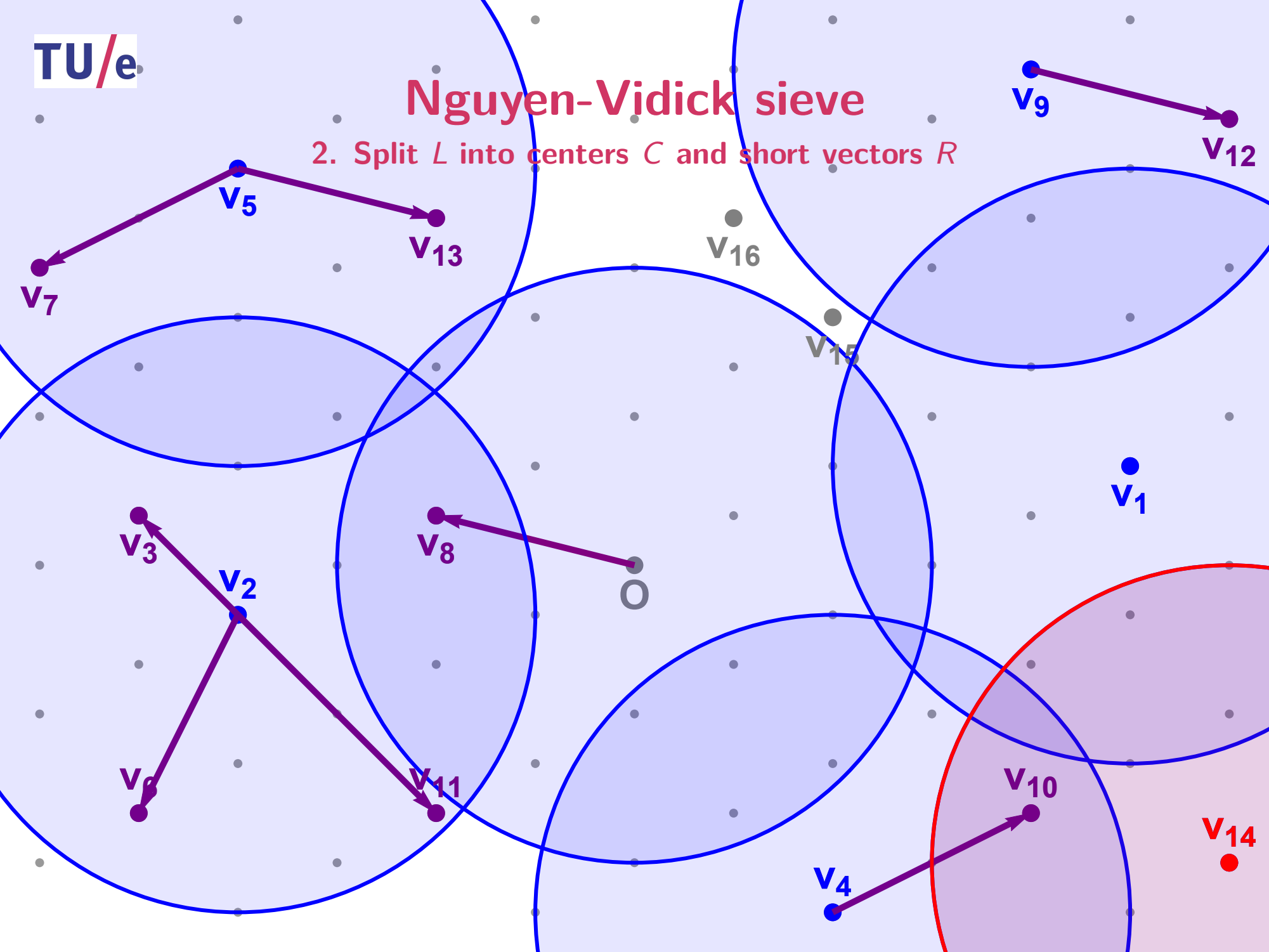

TU/e
Nguyen-Vidick sieve
2. Split L into centers C and short vectors R
v_9
v_{12}
v_5
v_{13}
v_{16}
v_7
v_{15}
v_1
v_3
v_8
v_2
O
v_6
v_{11}
v_{10}
v_{14}
v_4

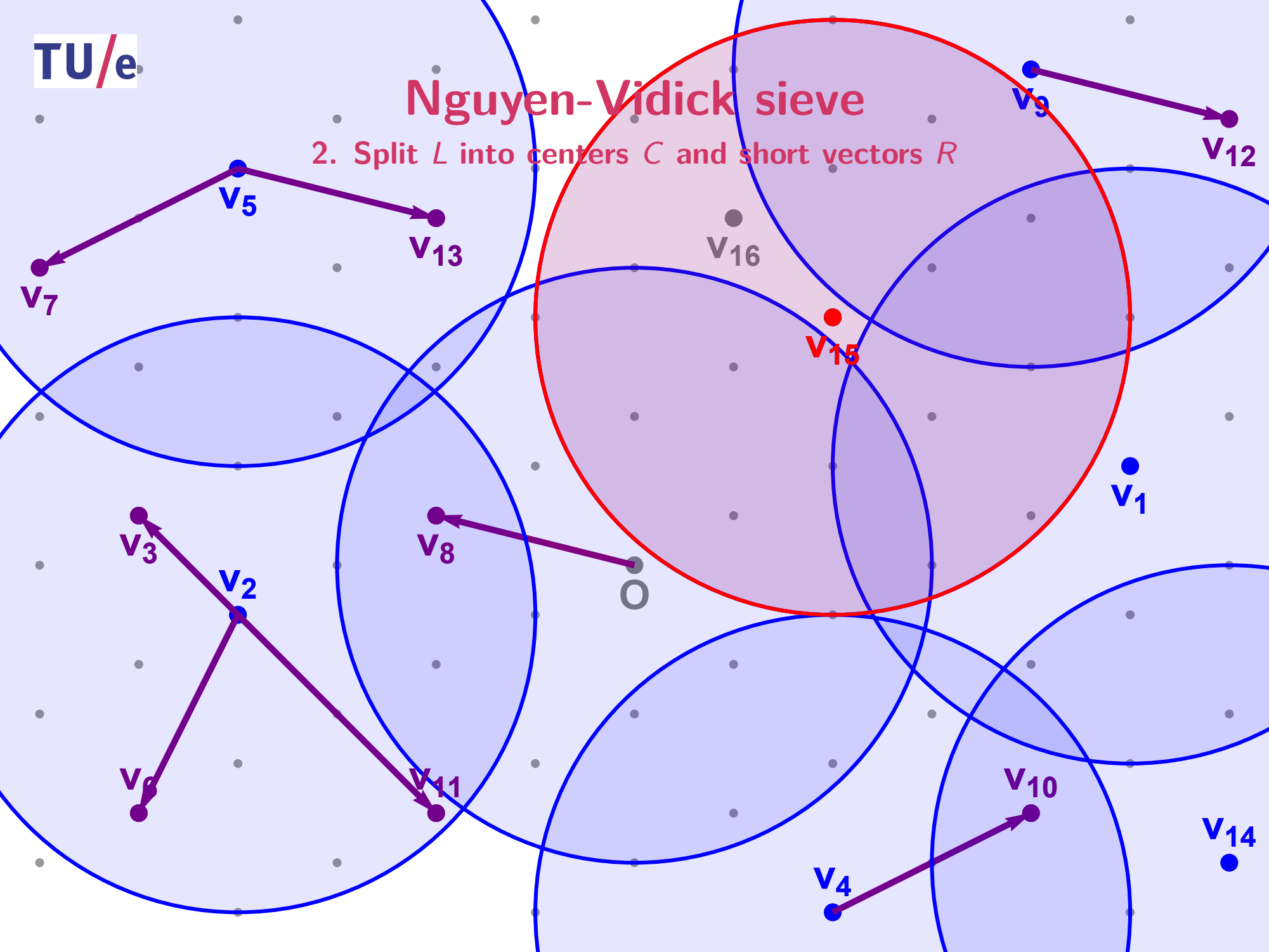

TU/e
Nguyen-Vidick sieve
2. Split L into centers C and short vectors R
v_9
v_{12}
v_5
v_{13}
v_{16}
v_7
v_{15}
v_1
v_3
v_8
v_2
O
v_6
v_{11}
v_{10}
v_{14}
v_4

TU/e

Nguyen-Vidick sieve

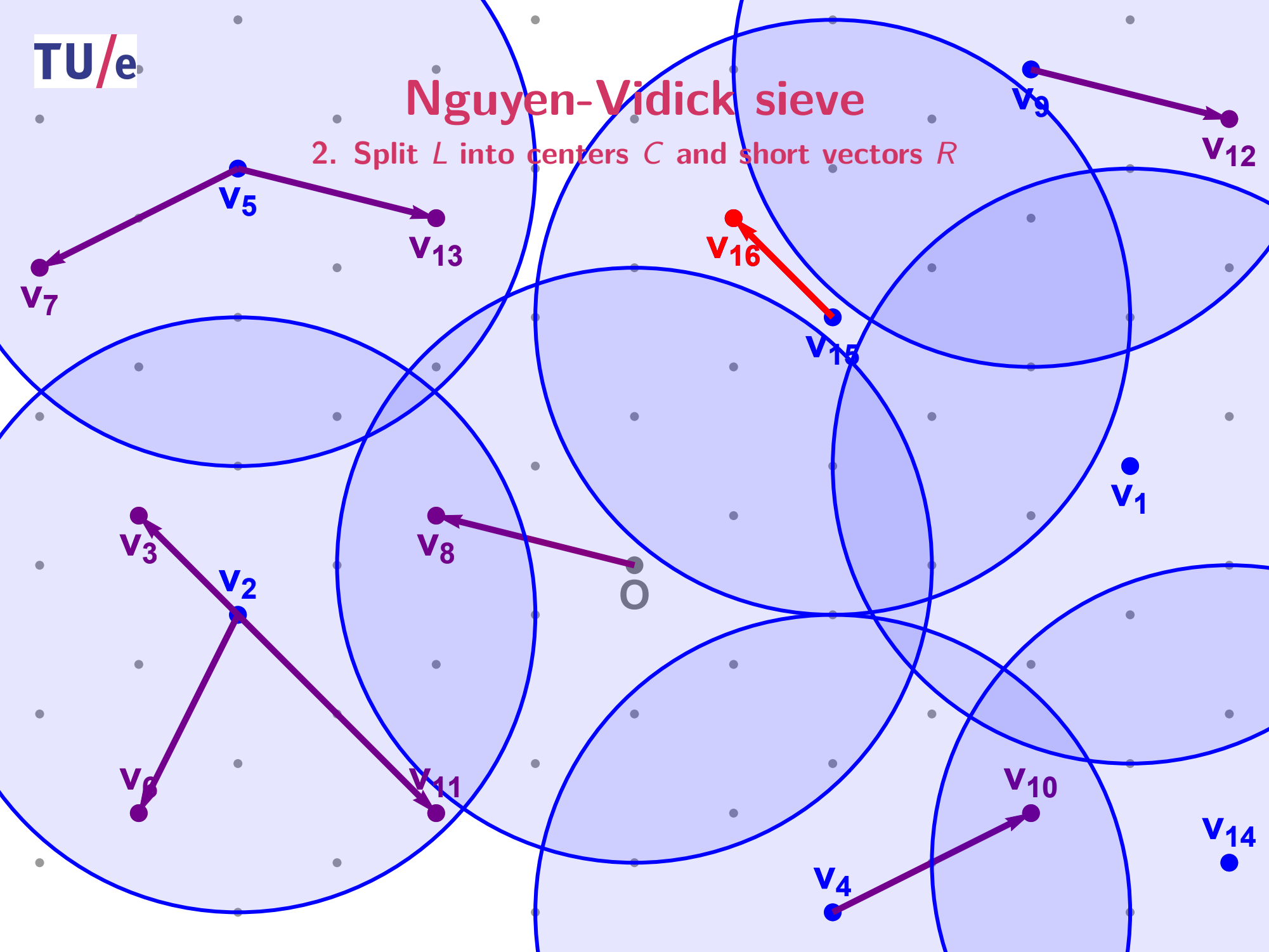
2. Split L into centers C and short vectors R




TU/e
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_8
v_2
O
v_6
v_{11}
v_{10}
v_{14}
v_4

Nguyen-Vidick sieve

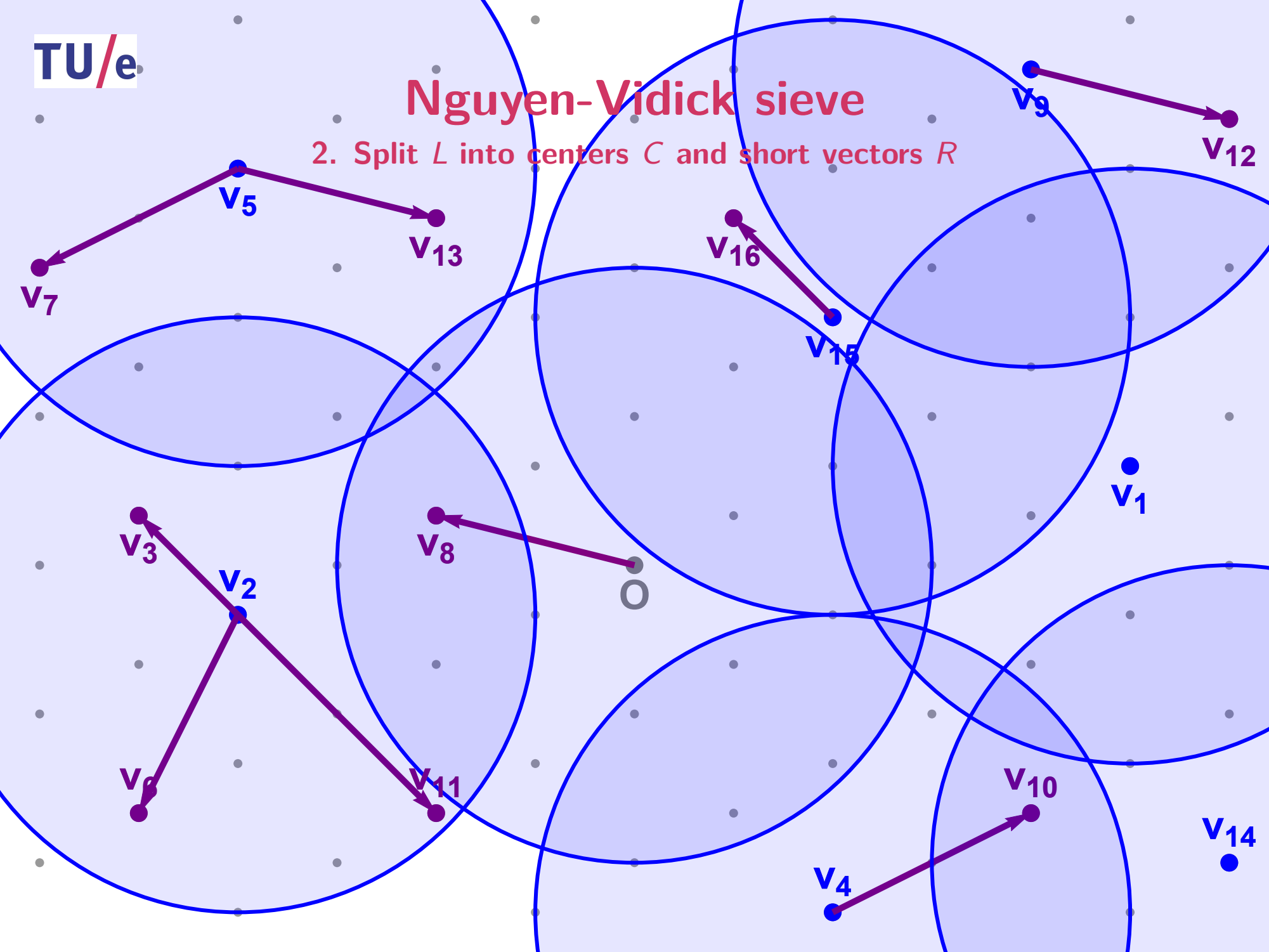
2. Split L into centers C and short vectors R



TU/e

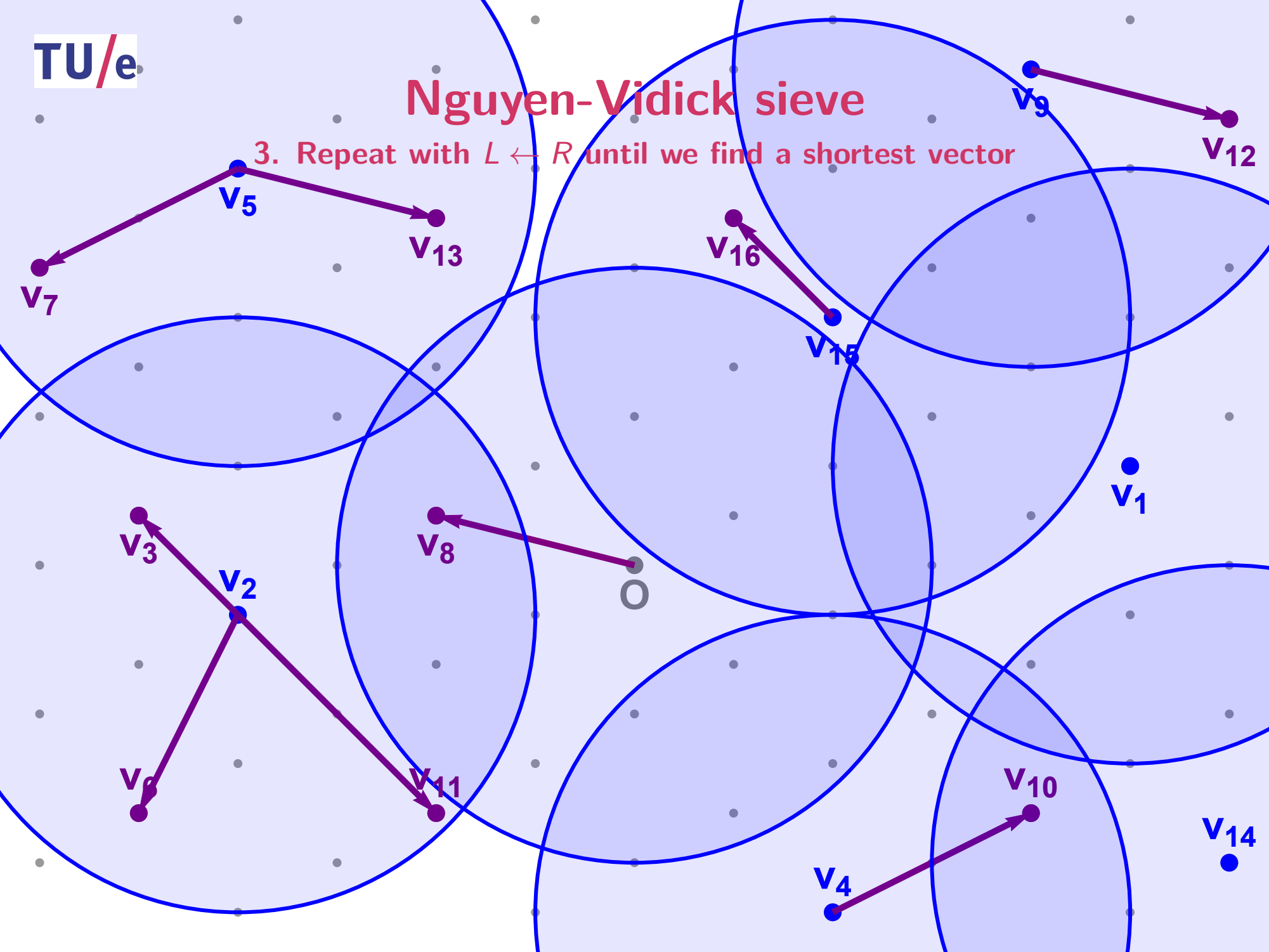
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



Nguyen-Vidick sieve

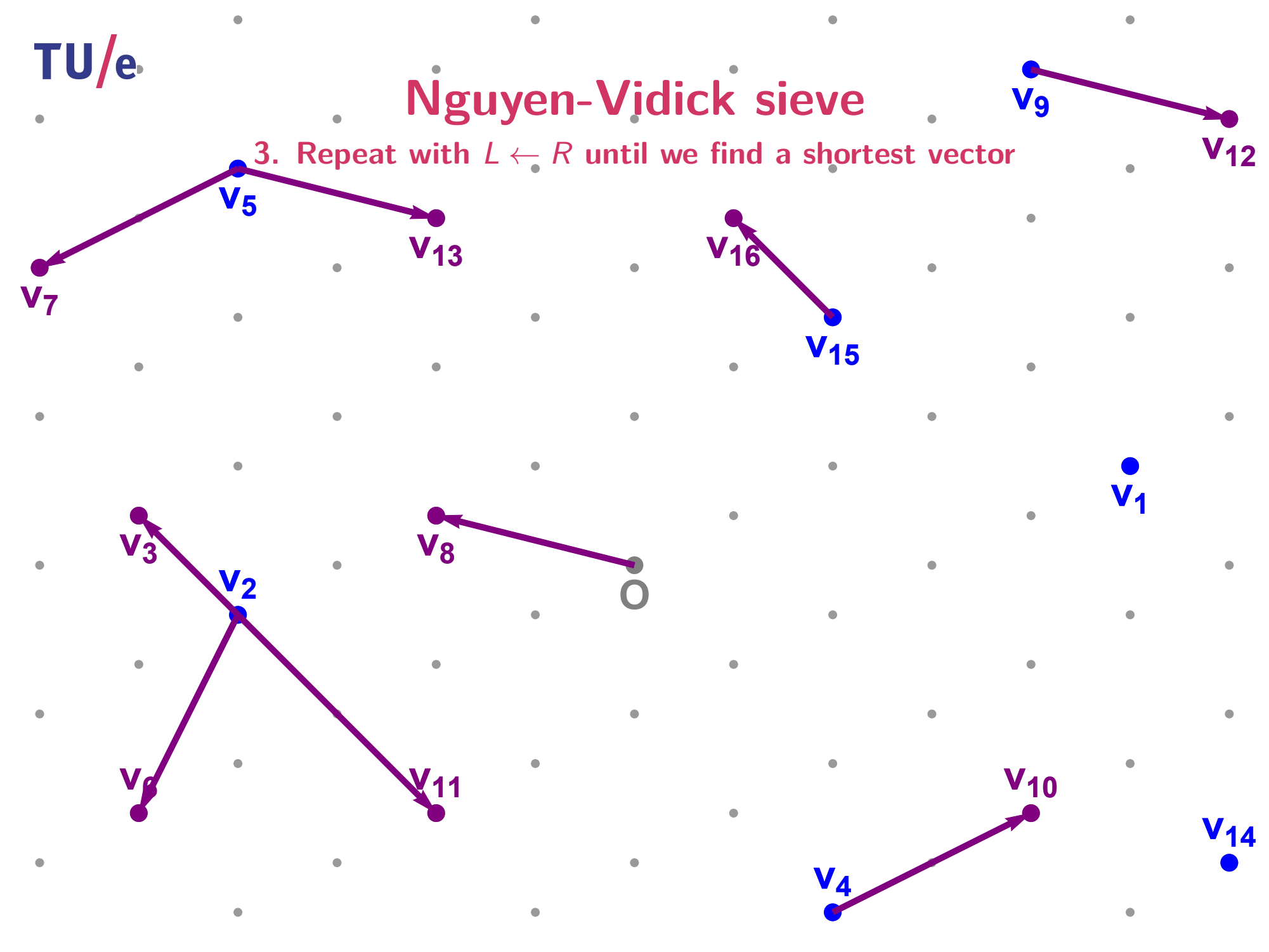
3. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

Nguyen-Vidick sieve

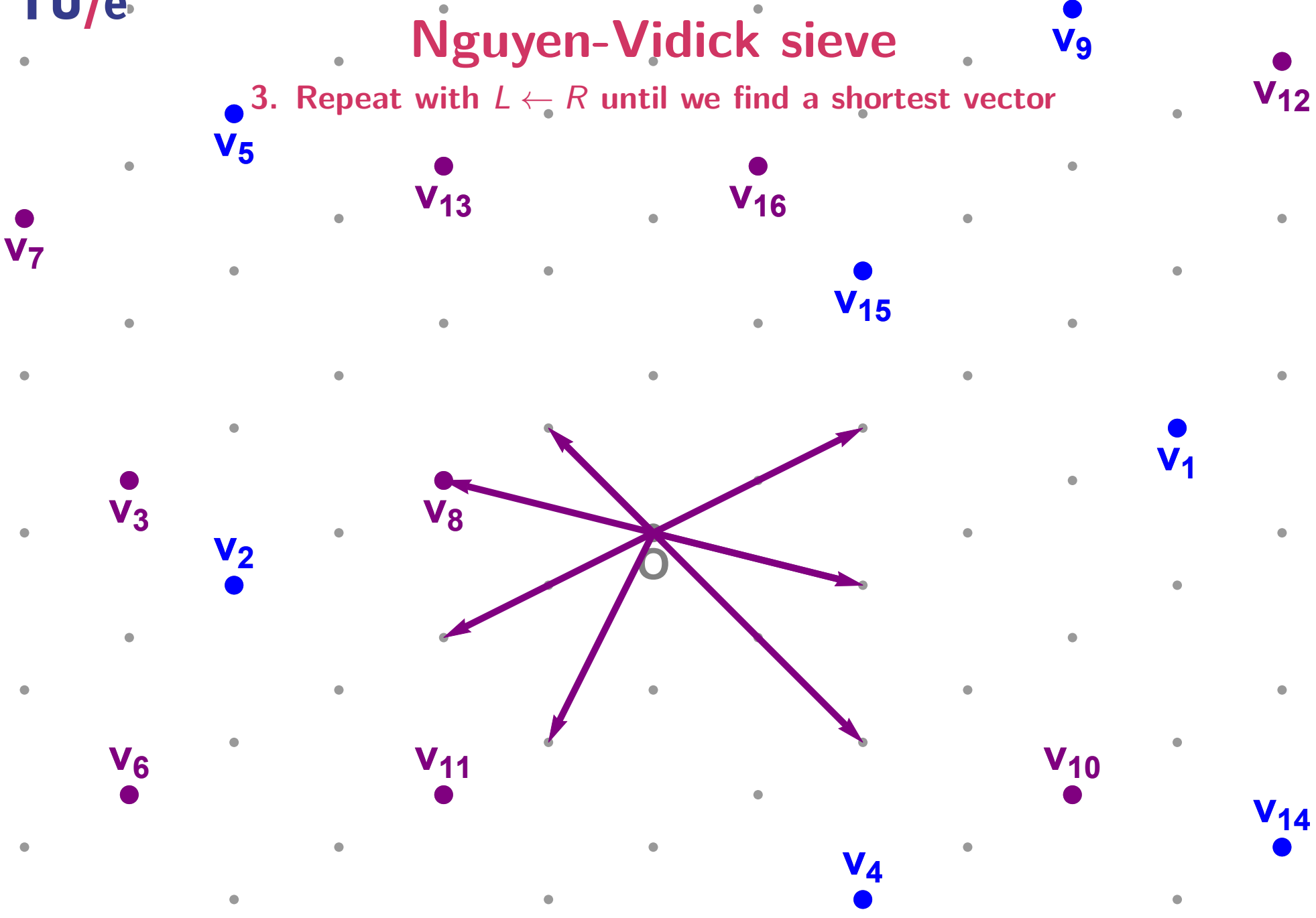
3. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

Nguyen-Vidick sieve

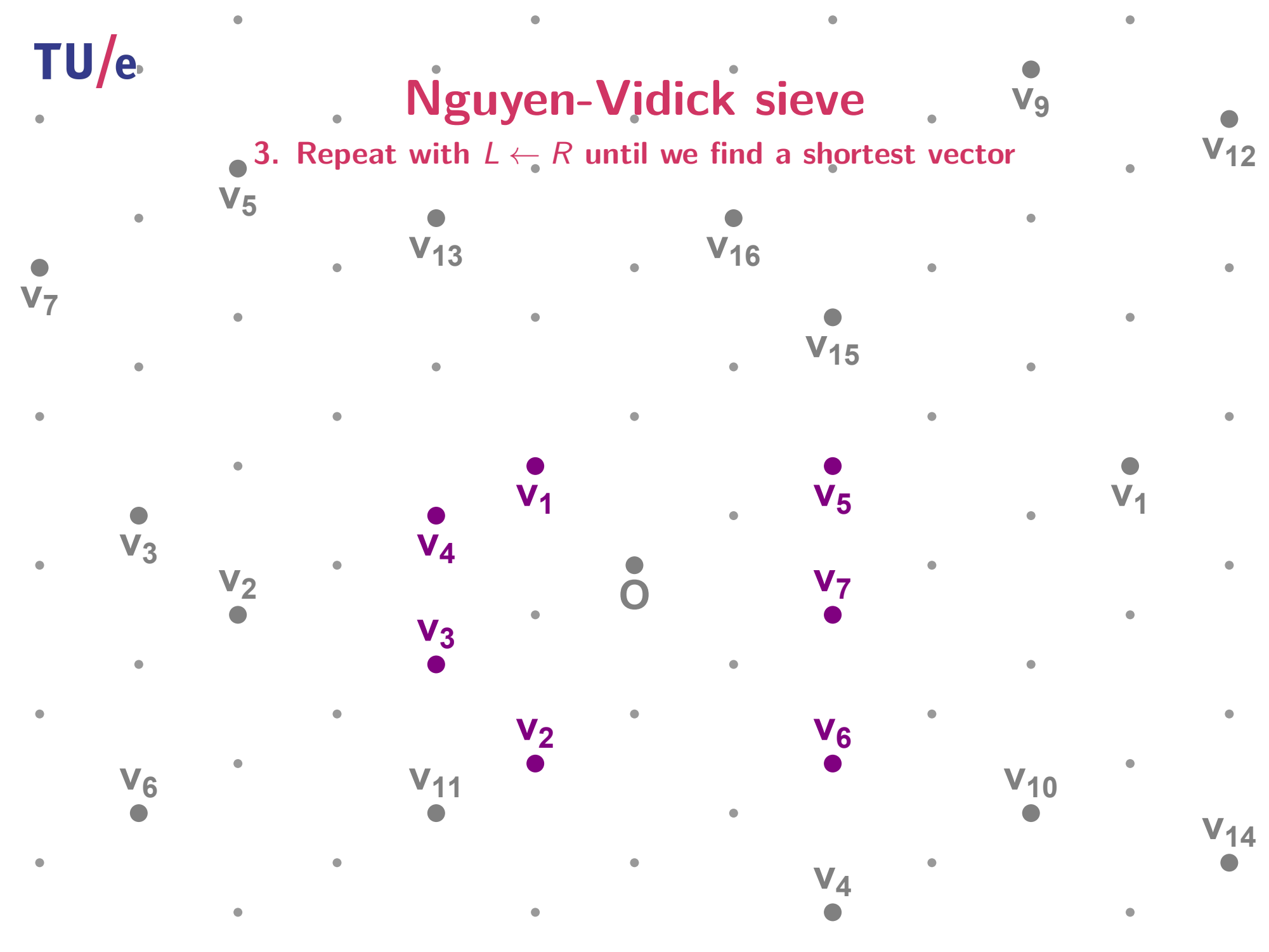
3. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

Nguyen-Vidick sieve

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



TU/e

Nguyen-Vidick sieve

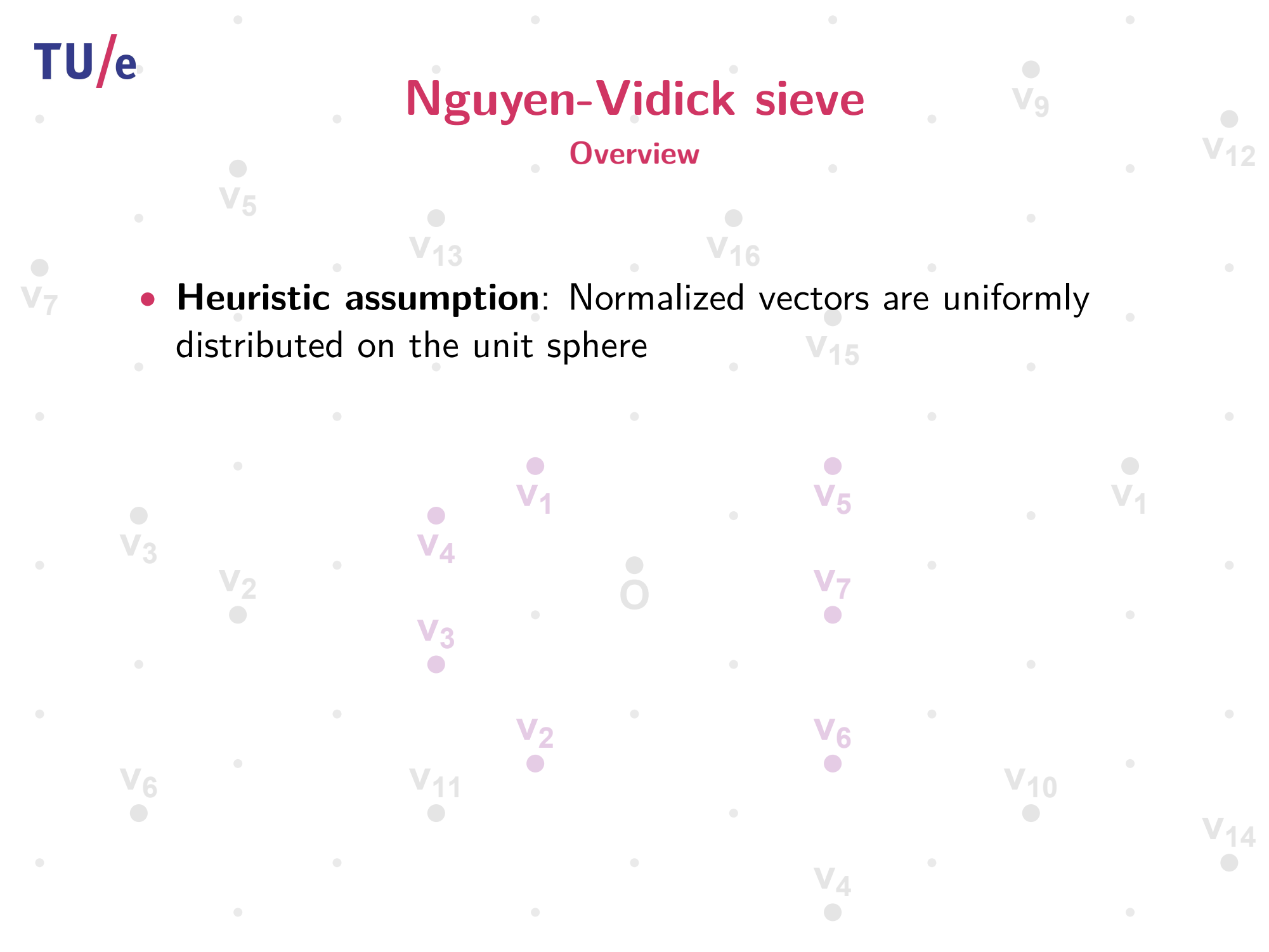
Overview

TU/e

Nguyen-Vidick sieve

Overview

- **Heuristic assumption**: Normalized vectors are uniformly distributed on the unit sphere



TU/e

Nguyen-Vidick sieve

Overview

- **Heuristic assumption**: Normalized vectors are uniformly distributed on the unit sphere
- Space complexity: $(\sqrt{4/3})^n \approx 2^{0.208n+o(n)}$ vectors
  - Each center covers $(\sin\frac{\pi}{3})^{-n} = (\sqrt{3/4})^n$ of the space
  - Need $(\sqrt{4/3})^{n+o(n)}$ vectors to cover all corners of $\mathbb{R}^n$

TU/e

Nguyen-Vidick sieve

Overview

- **Heuristic assumption**: Normalized vectors are uniformly distributed on the unit sphere
- Space complexity: $(\sqrt{4/3})^n \approx 2^{0.208n+o(n)}$ vectors
  - Each center covers $(\sin\frac{\pi}{3})^{-n} = (\sqrt{3/4})^n$ of the space
  - Need $(\sqrt{4/3})^{n+o(n)}$ vectors to cover all corners of $\mathbb{R}^n$
- Time complexity: $(4/3)^n \approx 2^{0.415n+o(n)}$

TU/e

Nguyen-Vidick sieve

Overview

- **Heuristic assumption**: Normalized vectors are uniformly distributed on the unit sphere
- Space complexity: $(\sqrt{4/3})^n \approx 2^{0.208n+o(n)}$ vectors
  - Each center covers $(\sin\frac{\pi}{3})^{-n} = (\sqrt{3/4})^n$ of the space
  - Need $(\sqrt{4/3})^{n+o(n)}$ vectors to cover all corners of $\mathbb{R}^n$
- Time complexity: $(4/3)^n \approx 2^{0.415n+o(n)}$

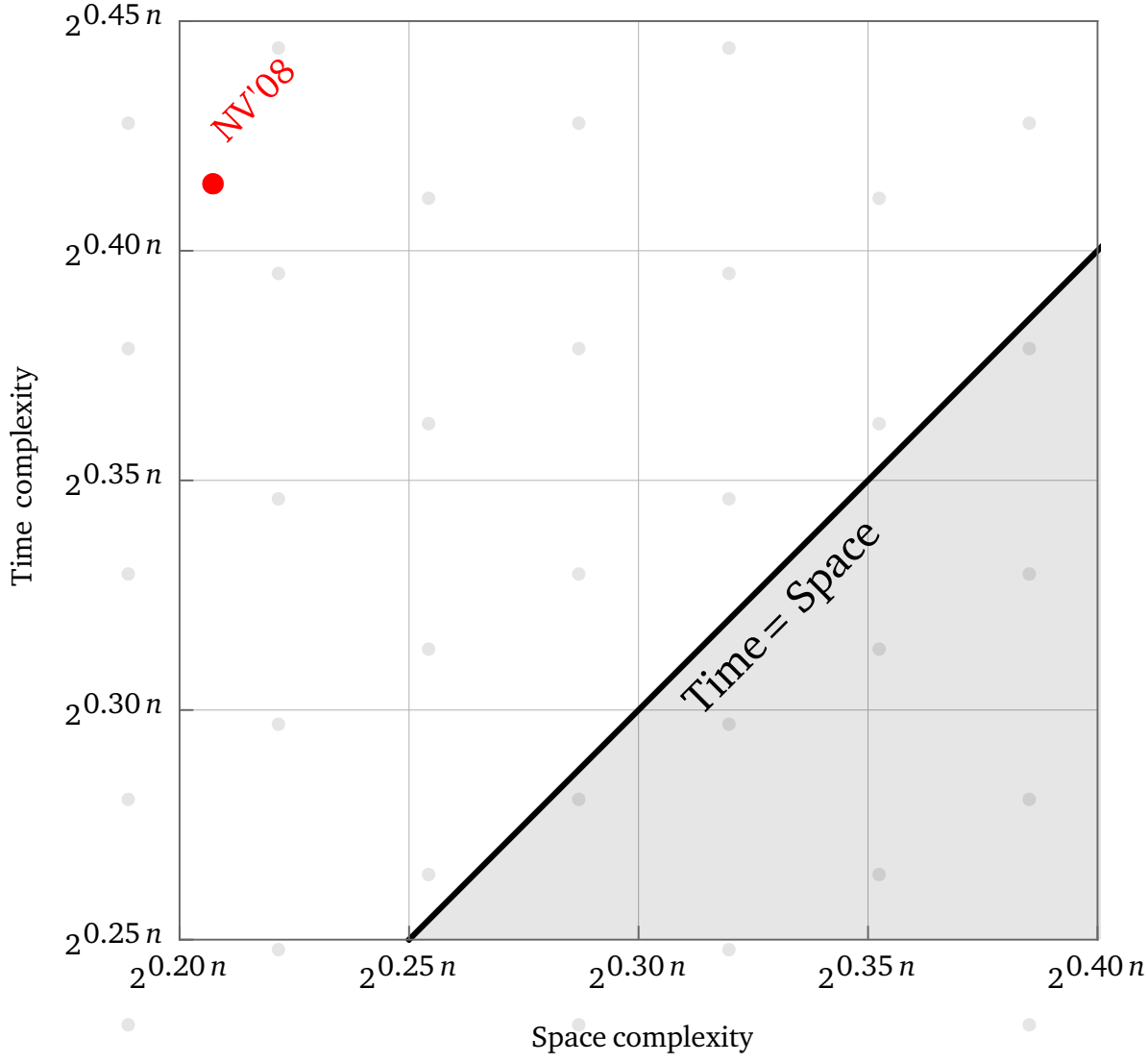
Theorem (Nguyen and Vidick, J. Math. Crypt. '08)

The Nguyen-Vidick sieve heuristically solves SVP in time $2^{0.415n+o(n)}$ and space $2^{0.208n+o(n)}$.

TU/e

Nguyen-Vidick sieve

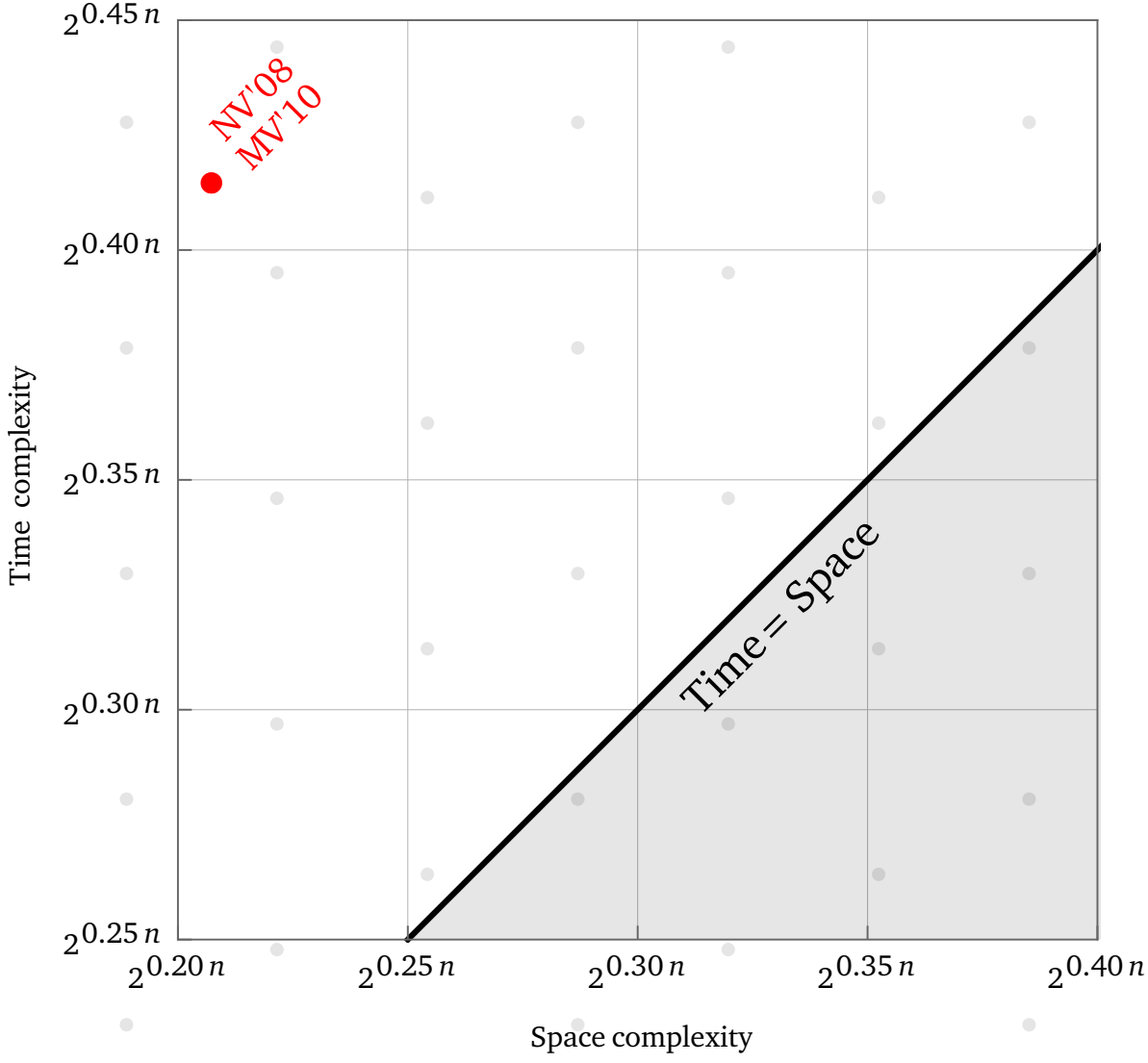
Space/time trade-off



TU/e

GaussSieve

Space/time trade-off



TU/e

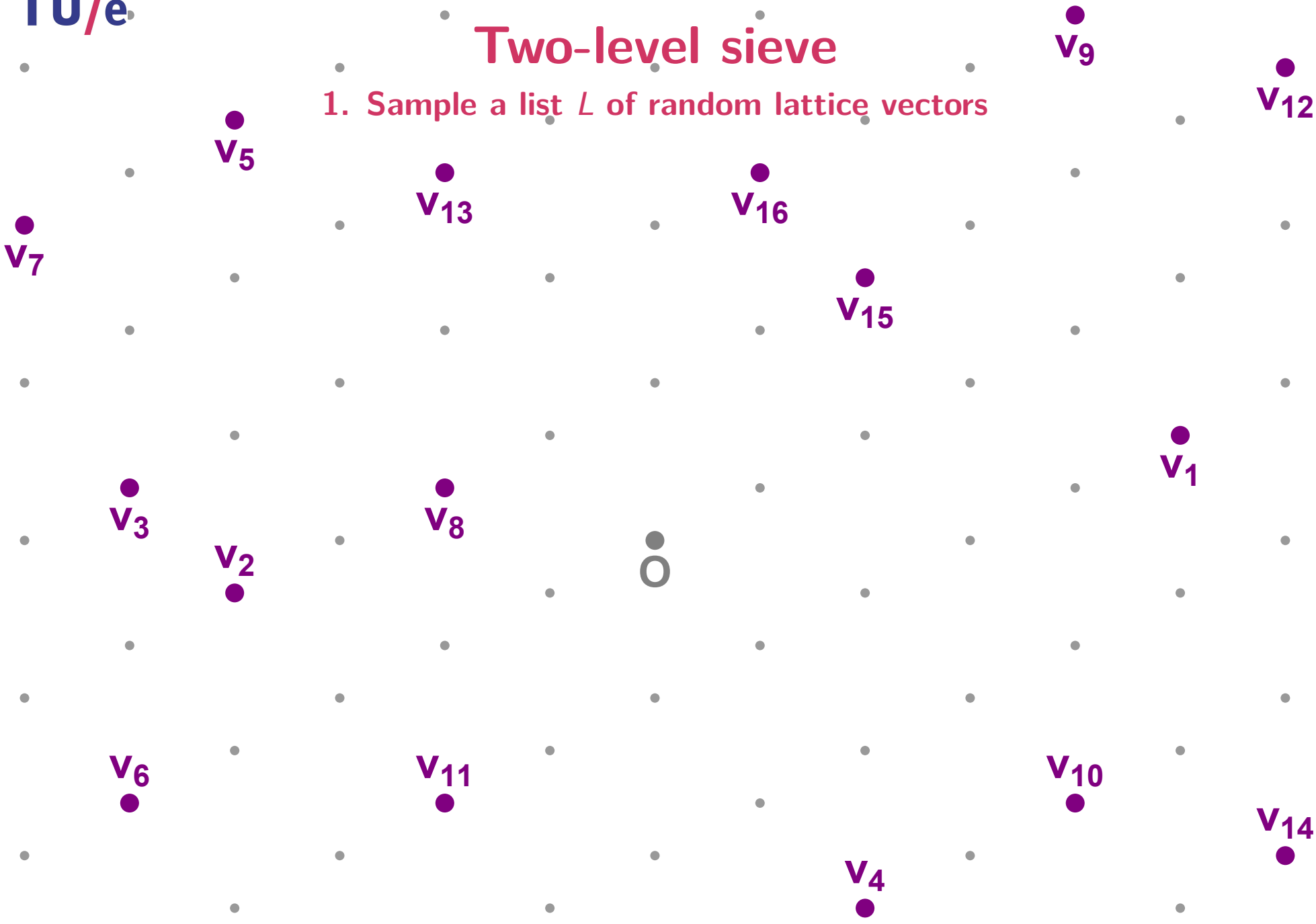
Two-level sieve

1. Sample a list L of random lattice vectors

TU/e

Two-level sieve

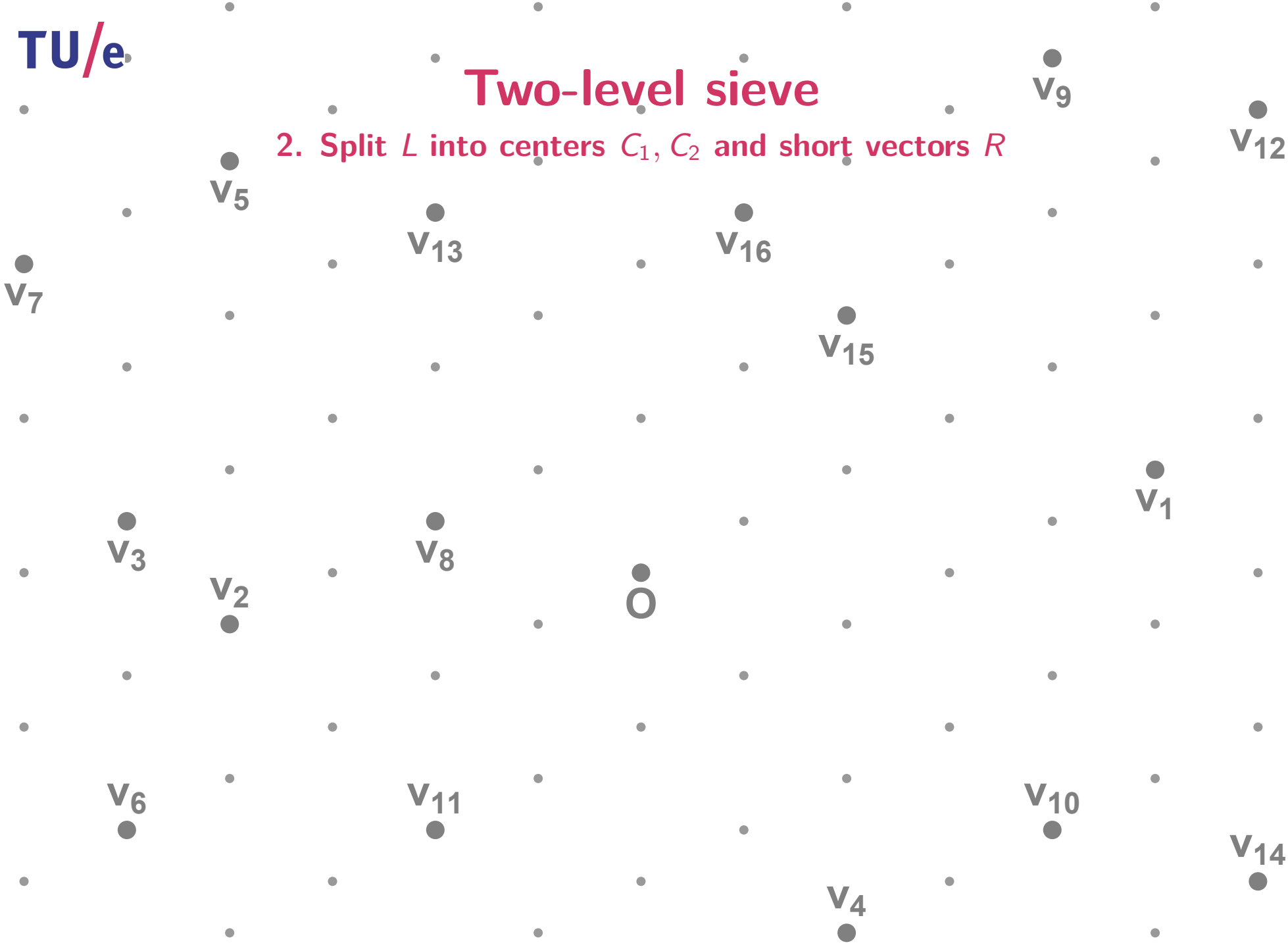
1. Sample a list L of random lattice vectors



TU/e

Two-level sieve

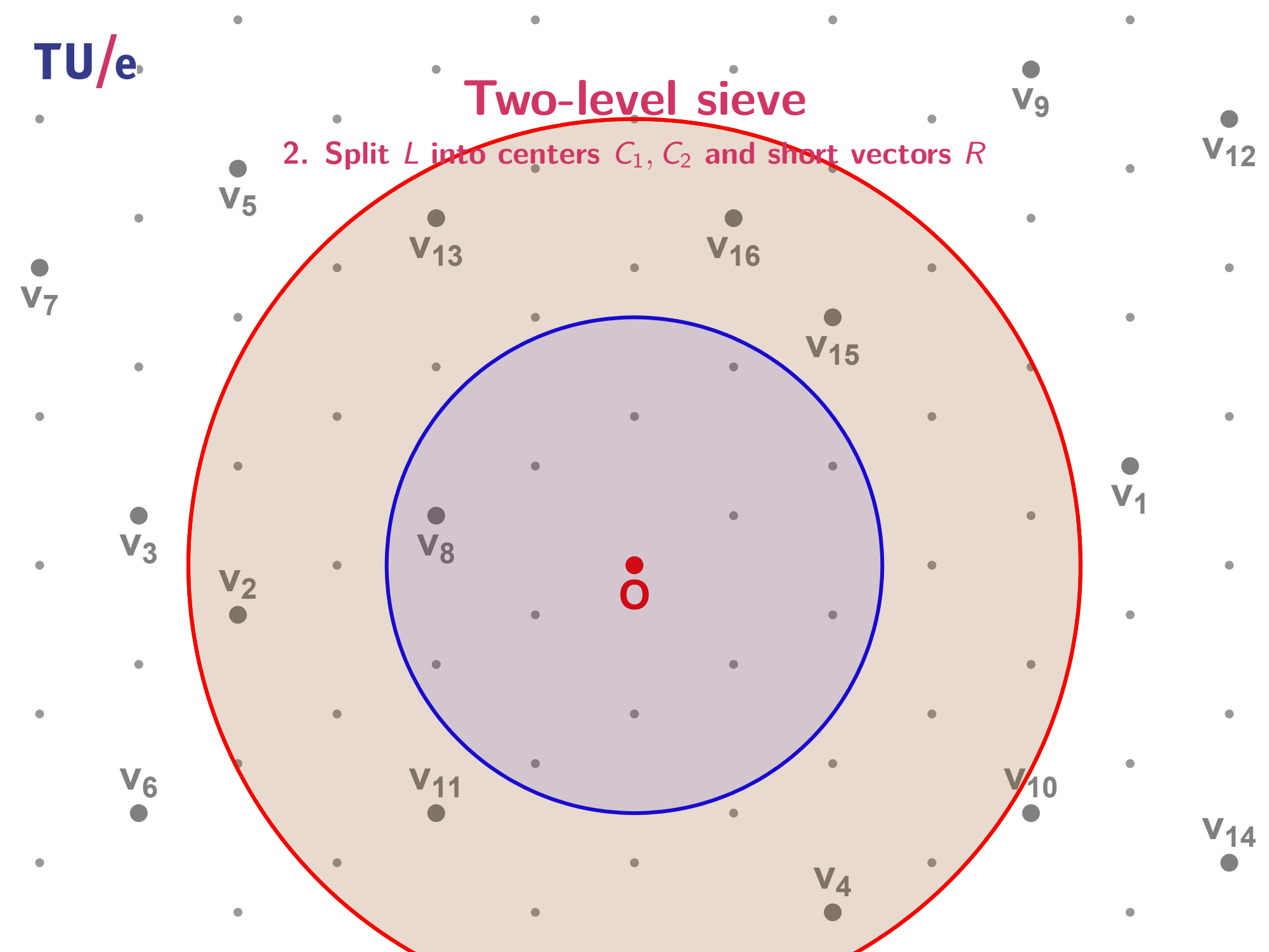
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

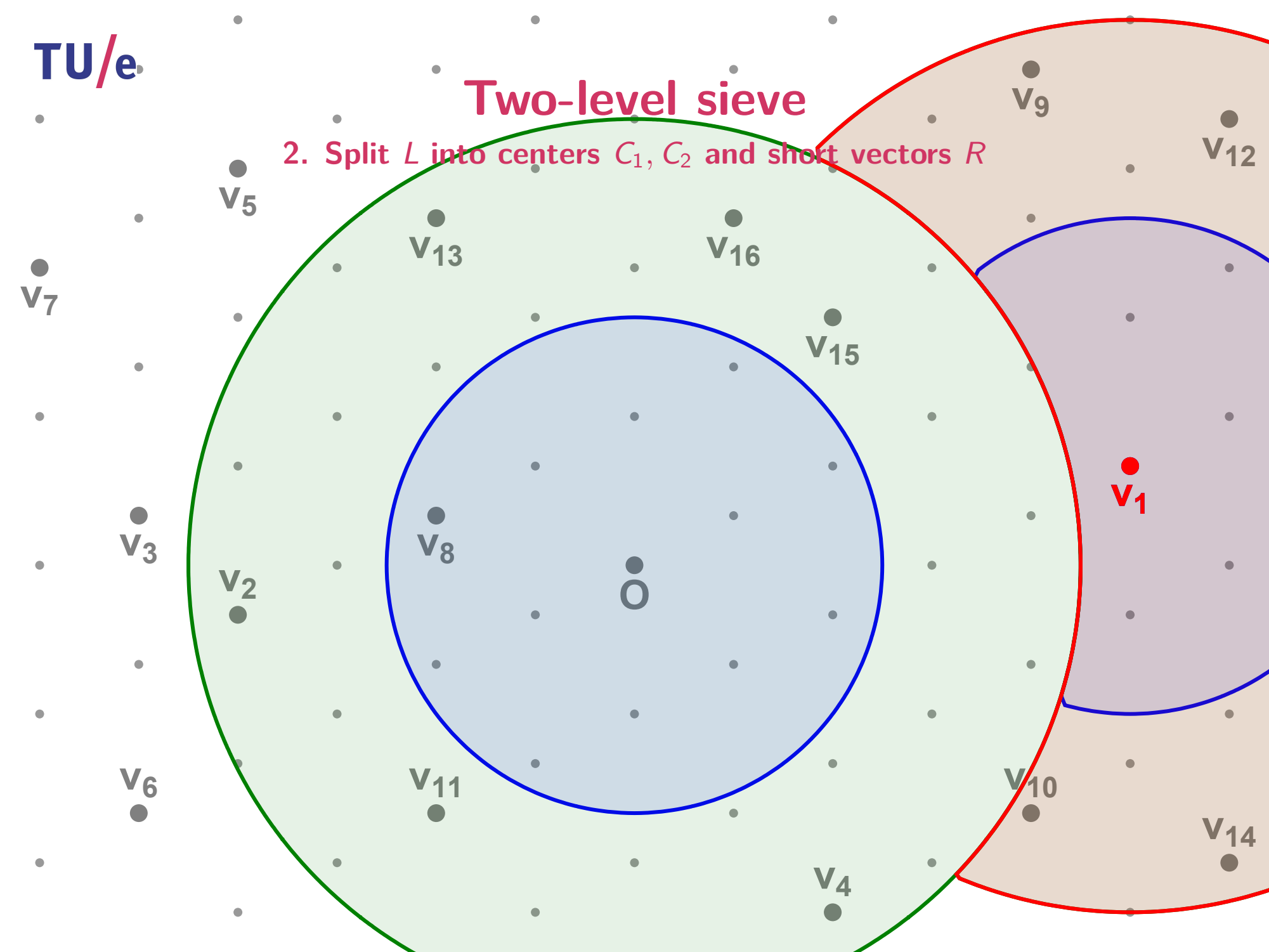
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

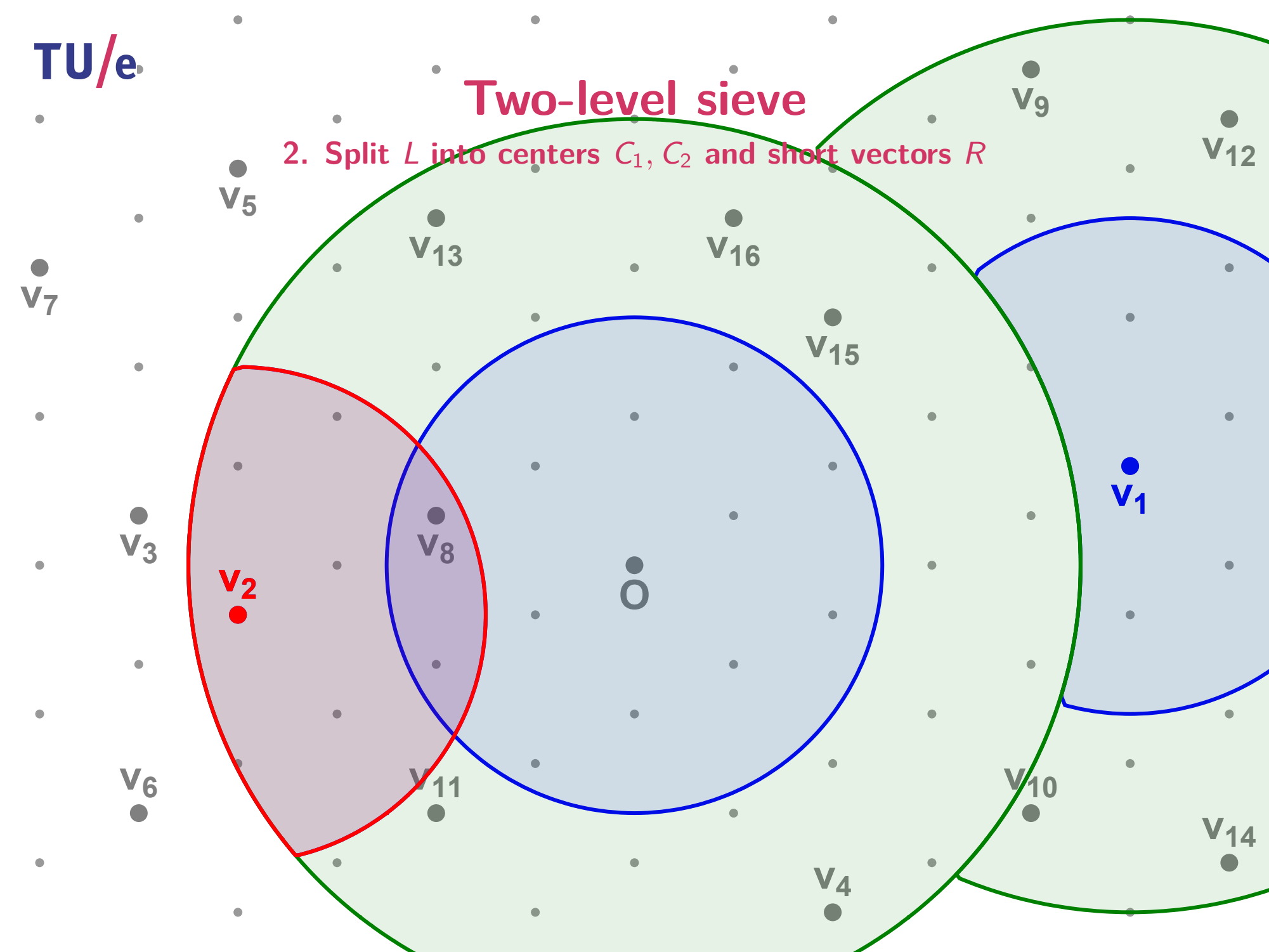
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

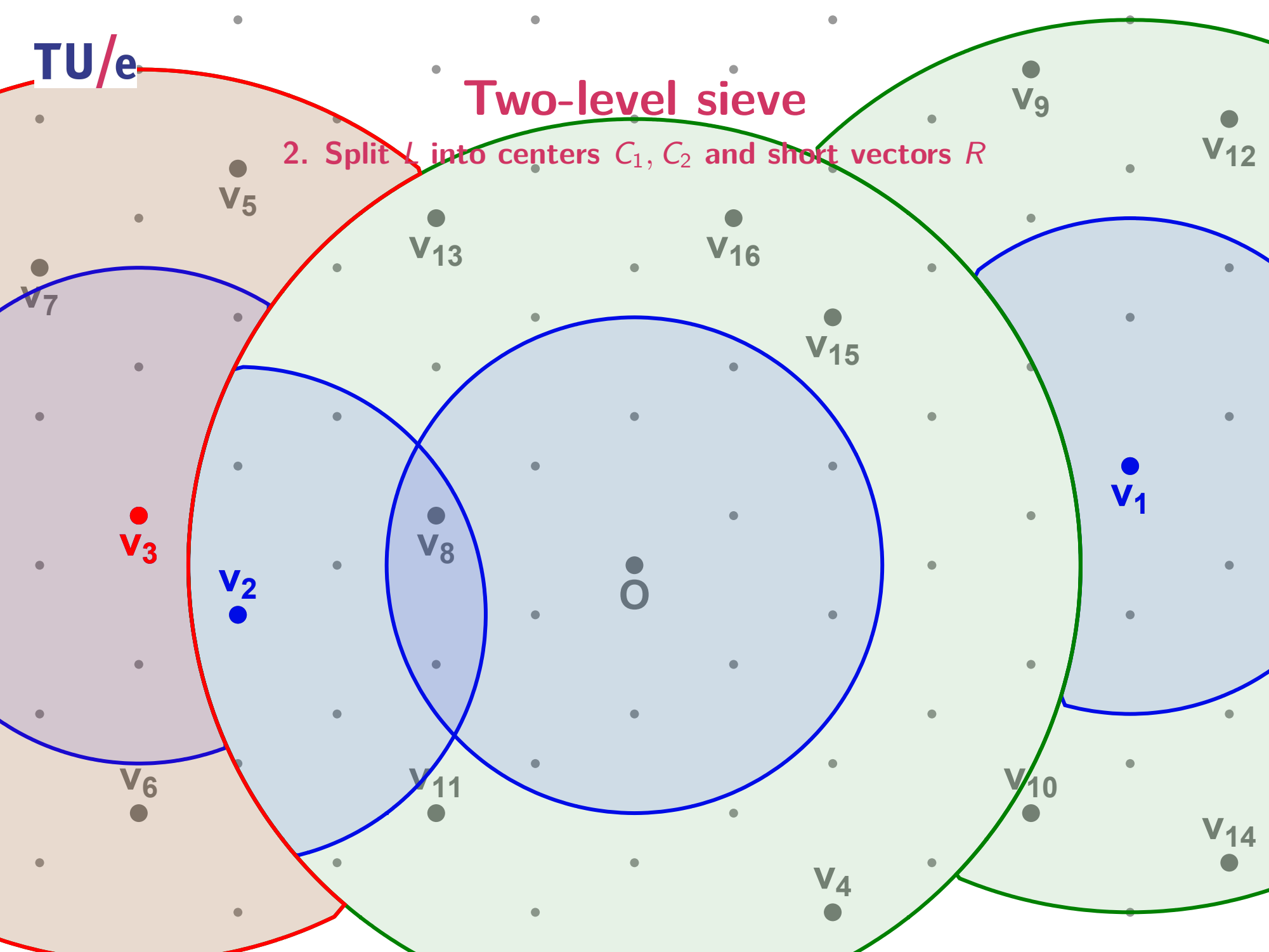
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

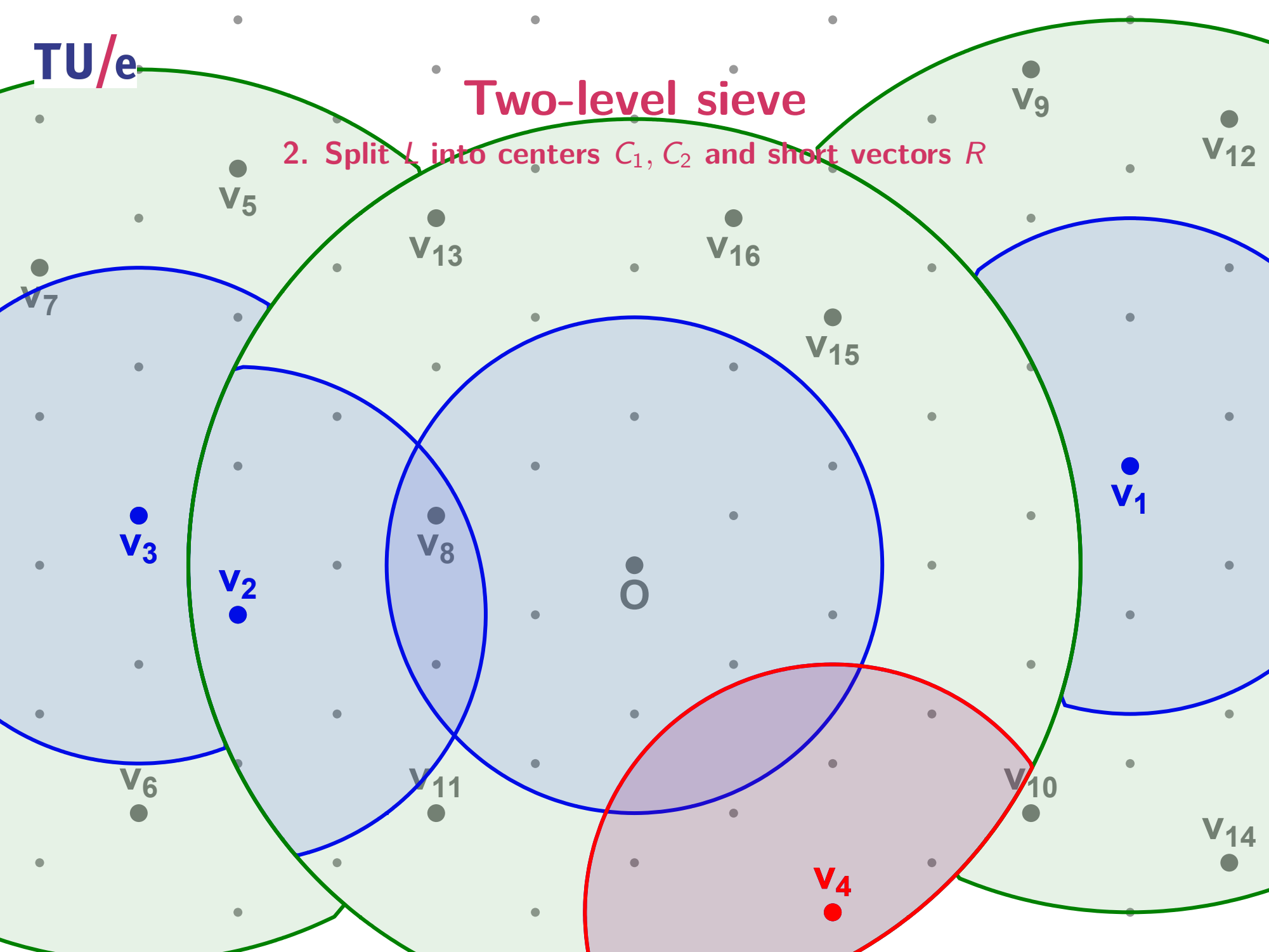
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

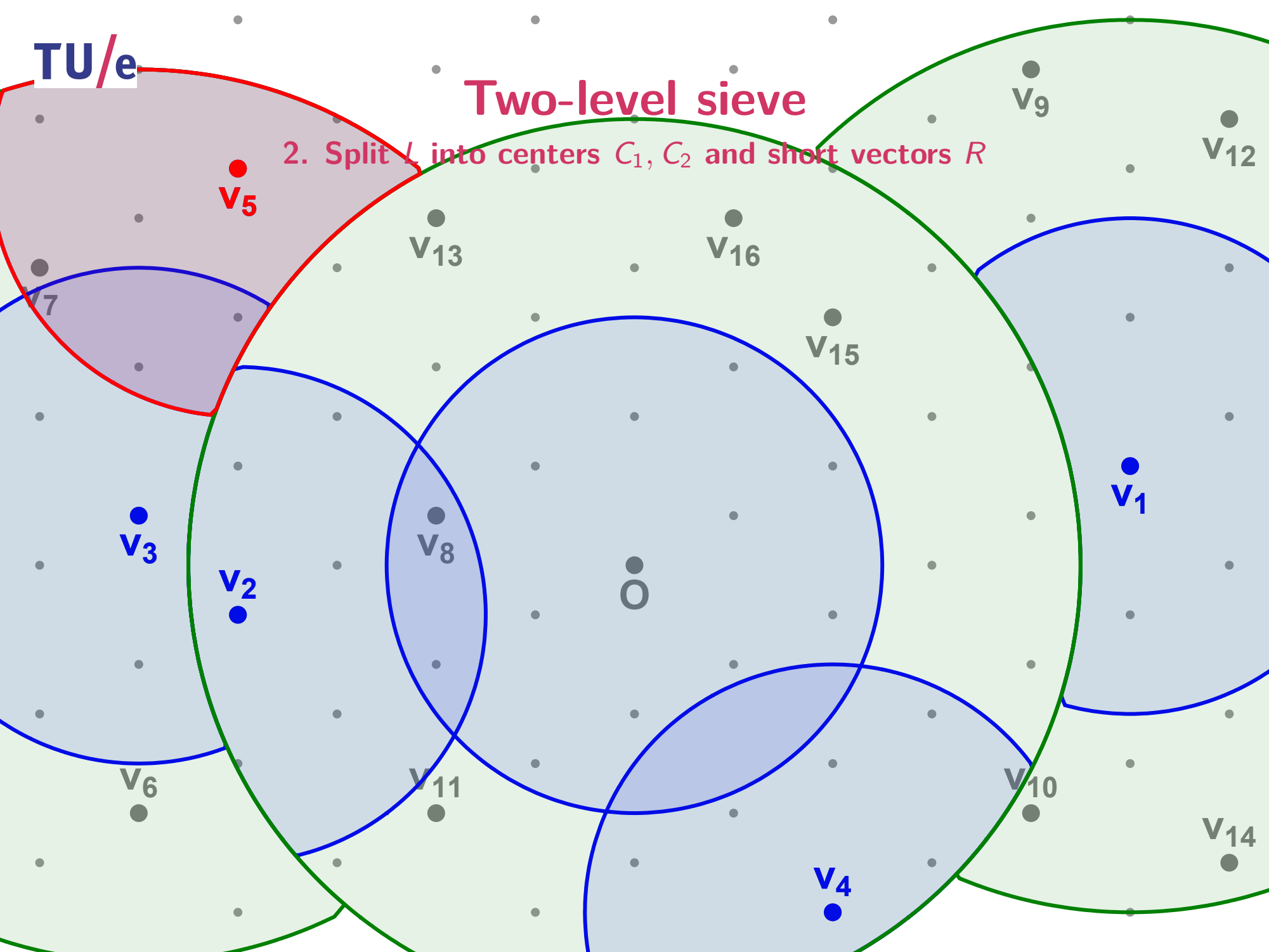
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

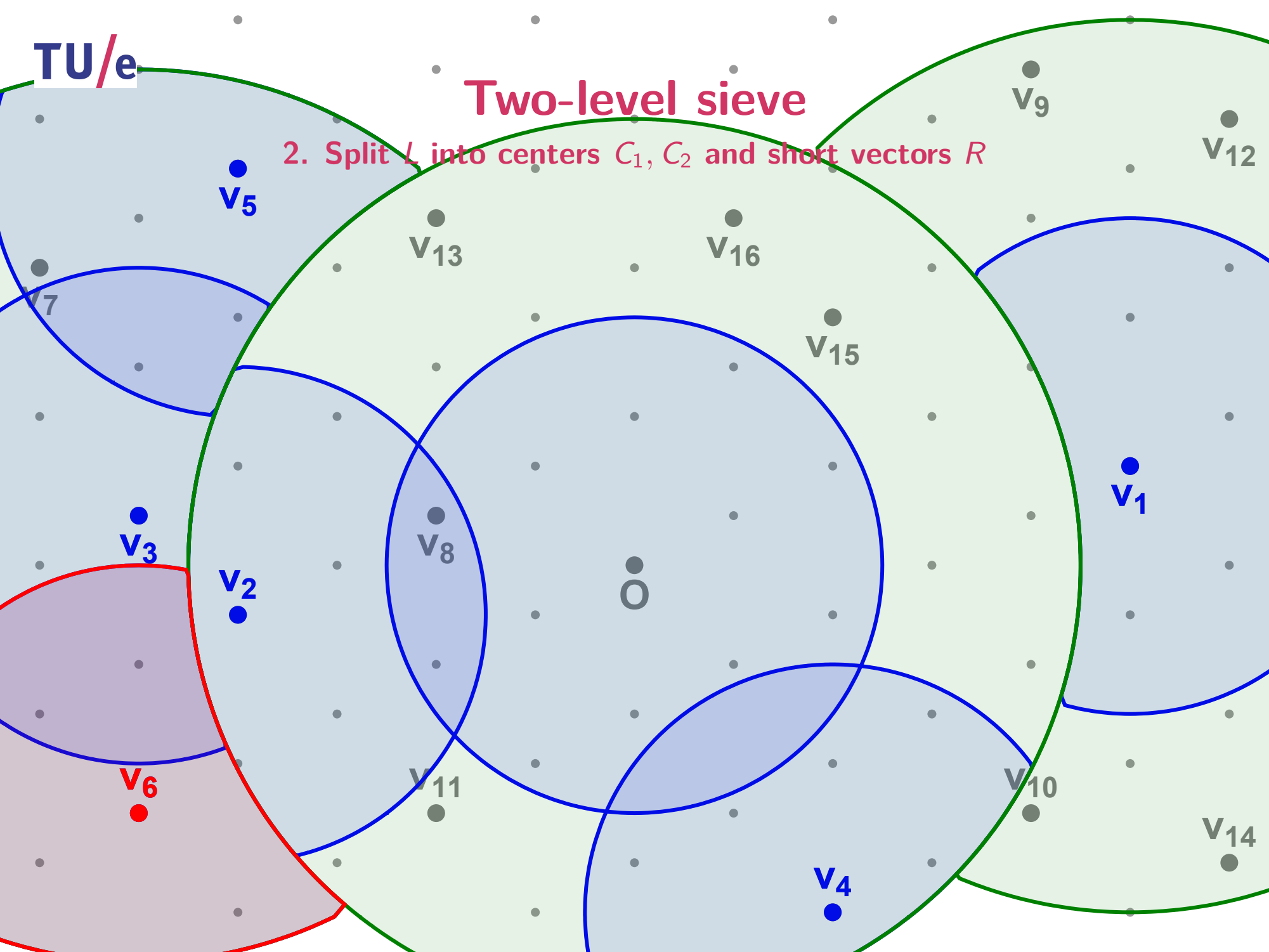
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

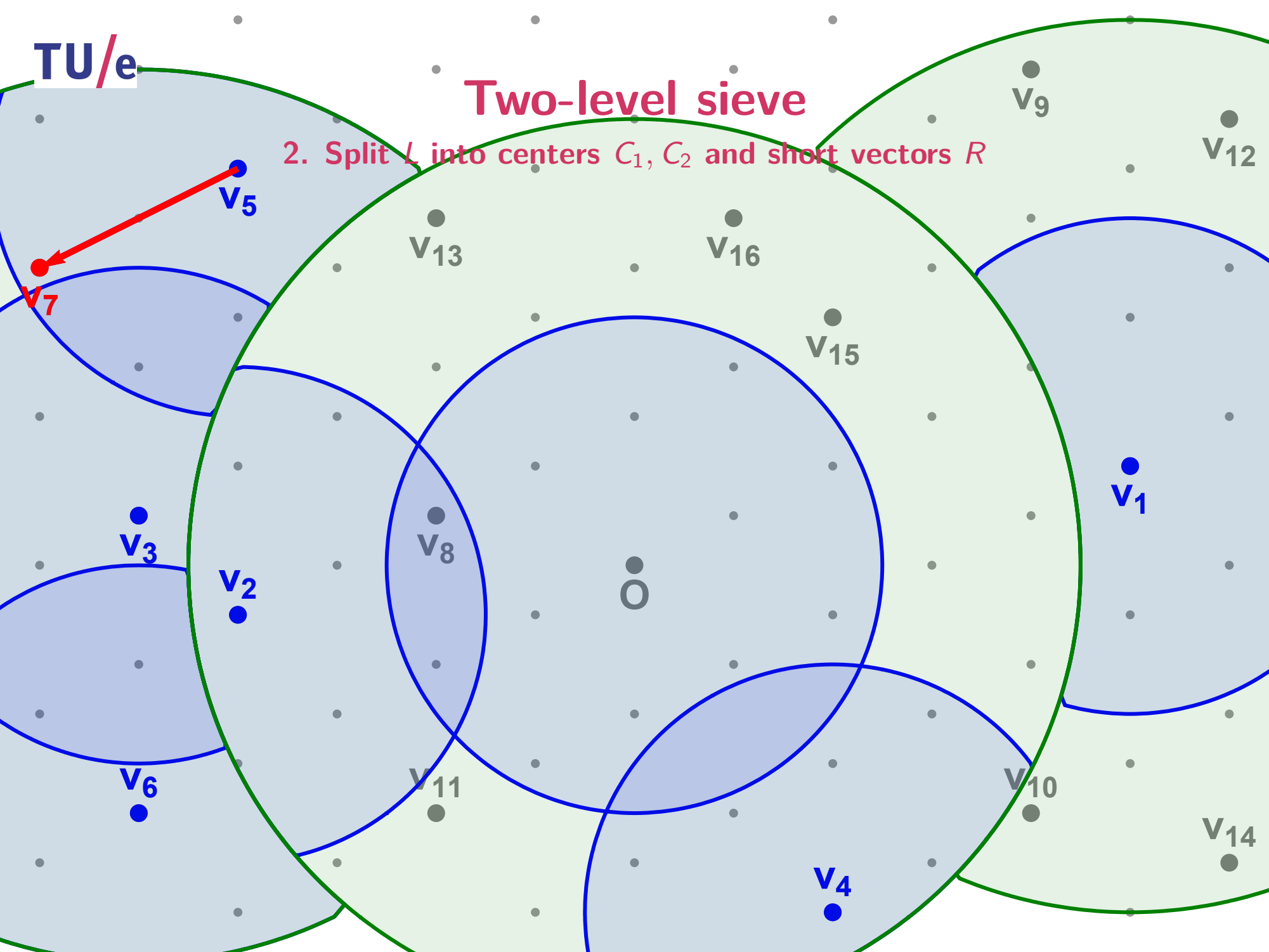
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

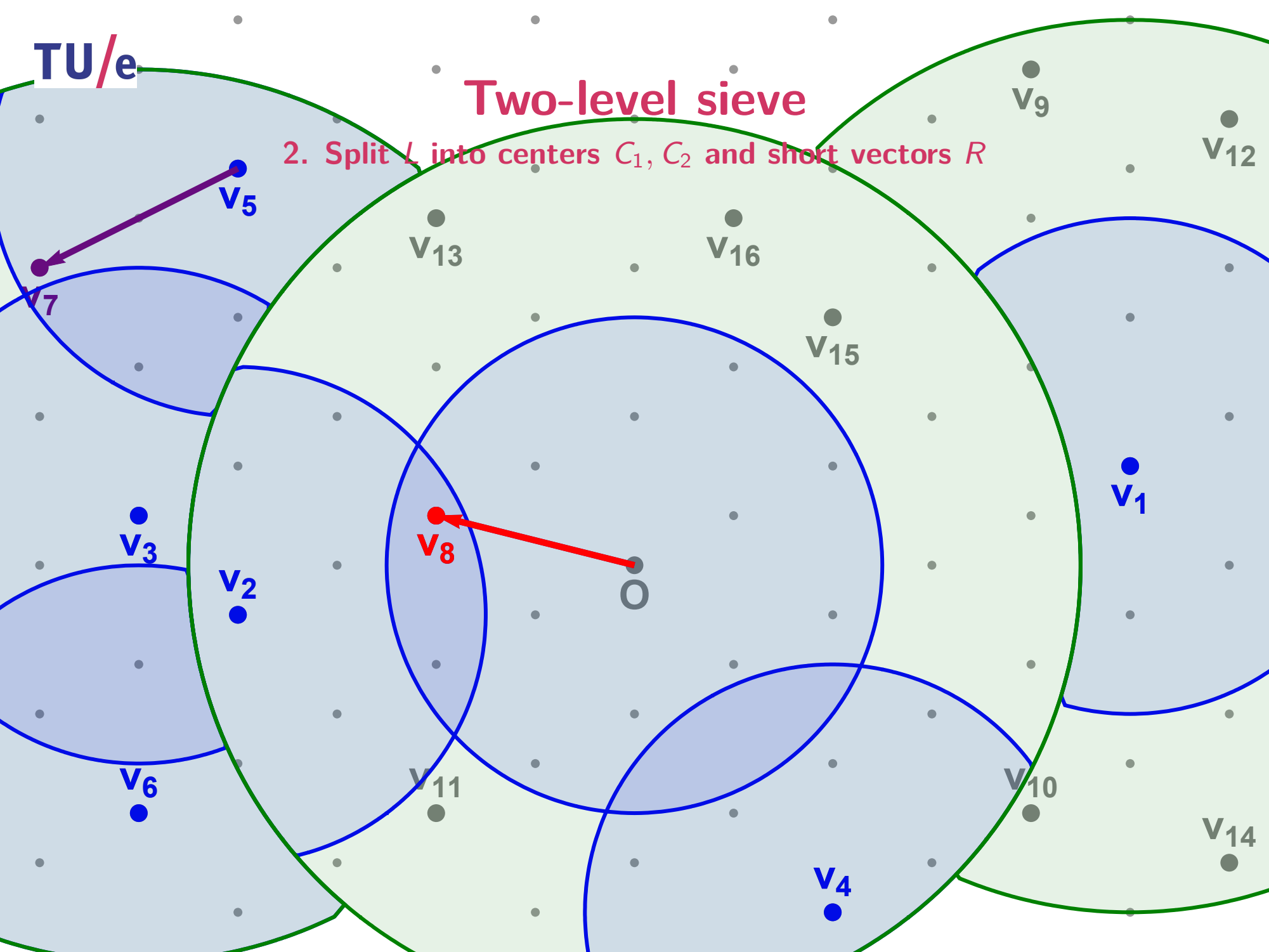
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

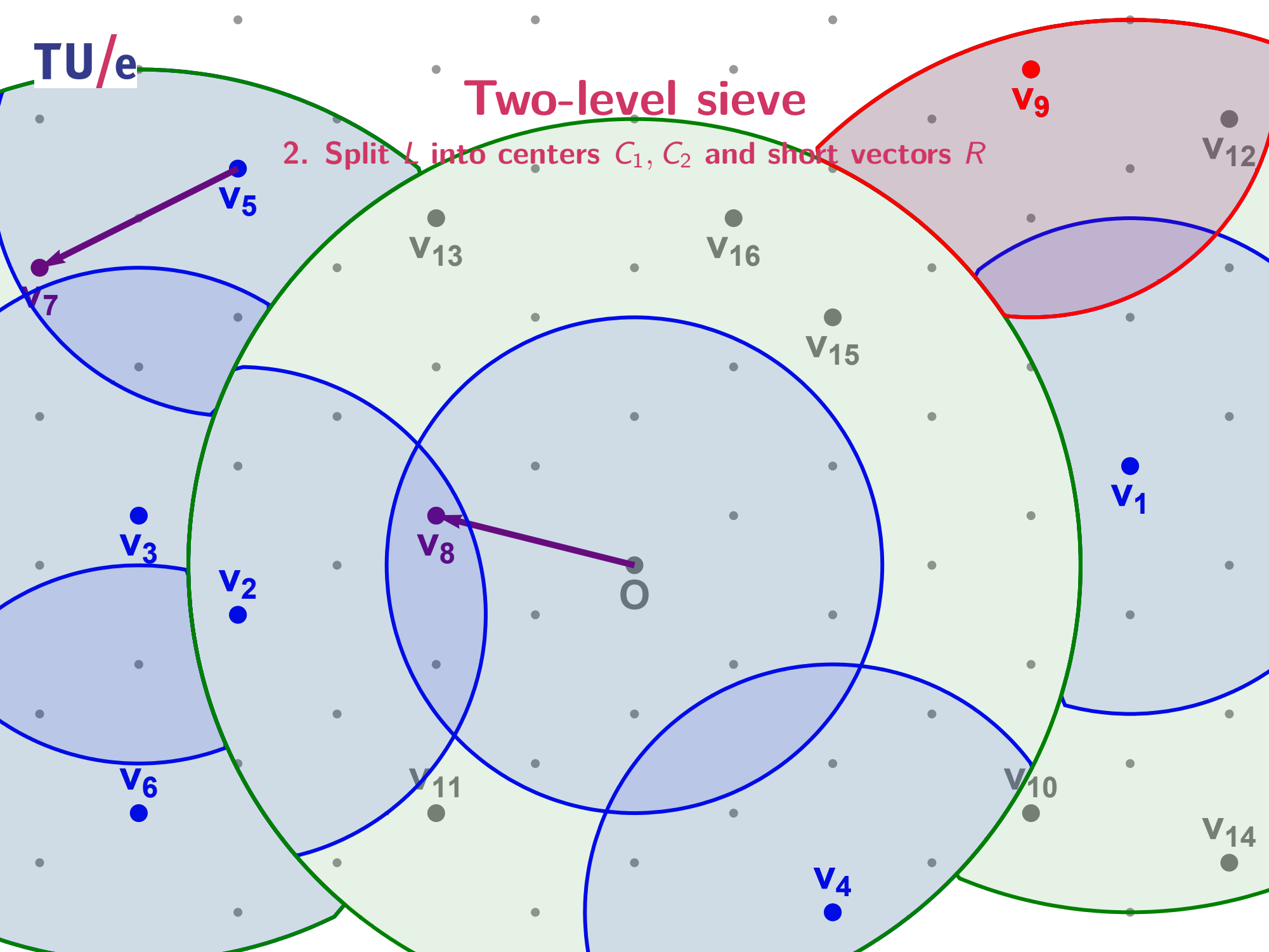
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

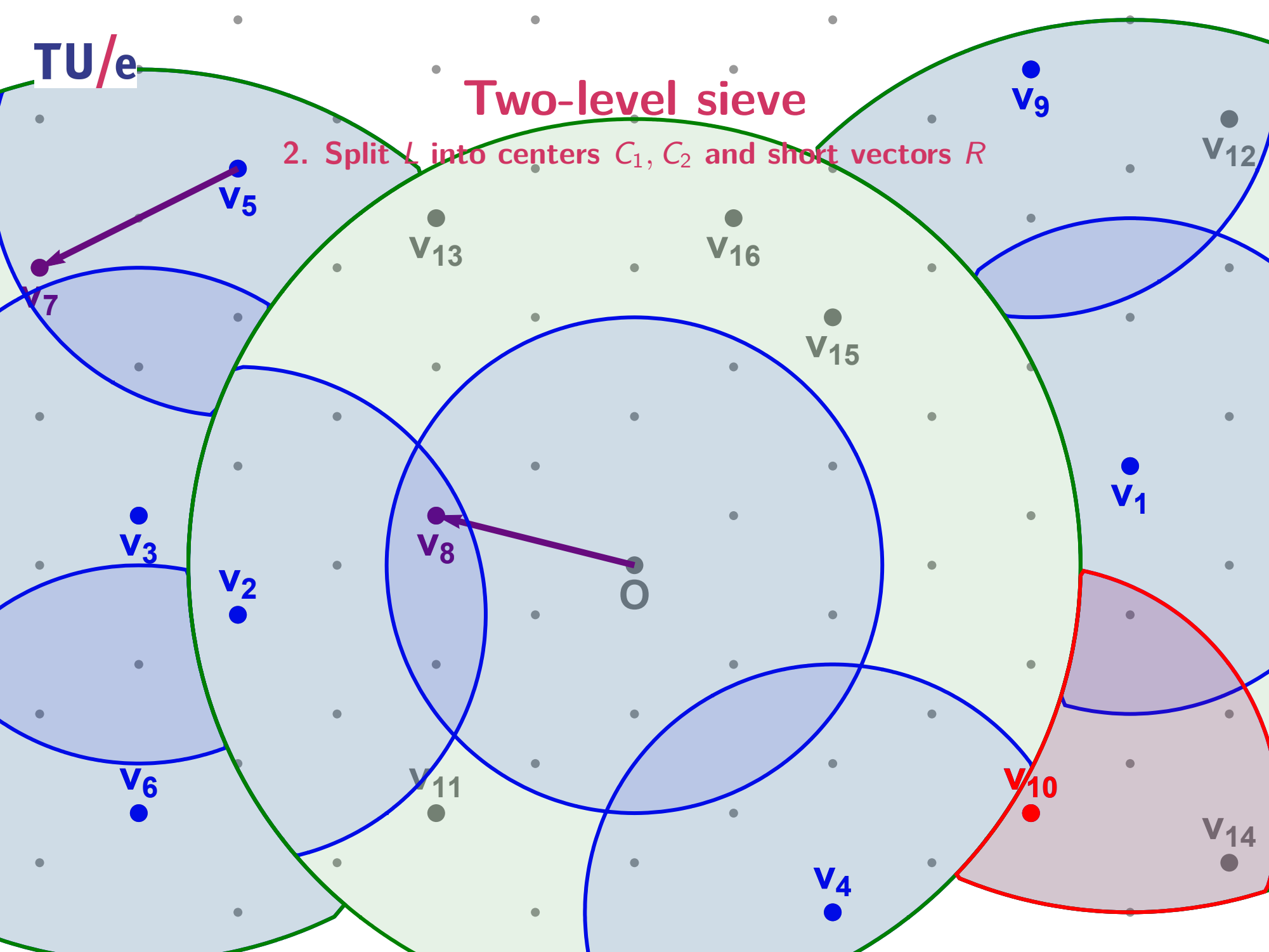
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

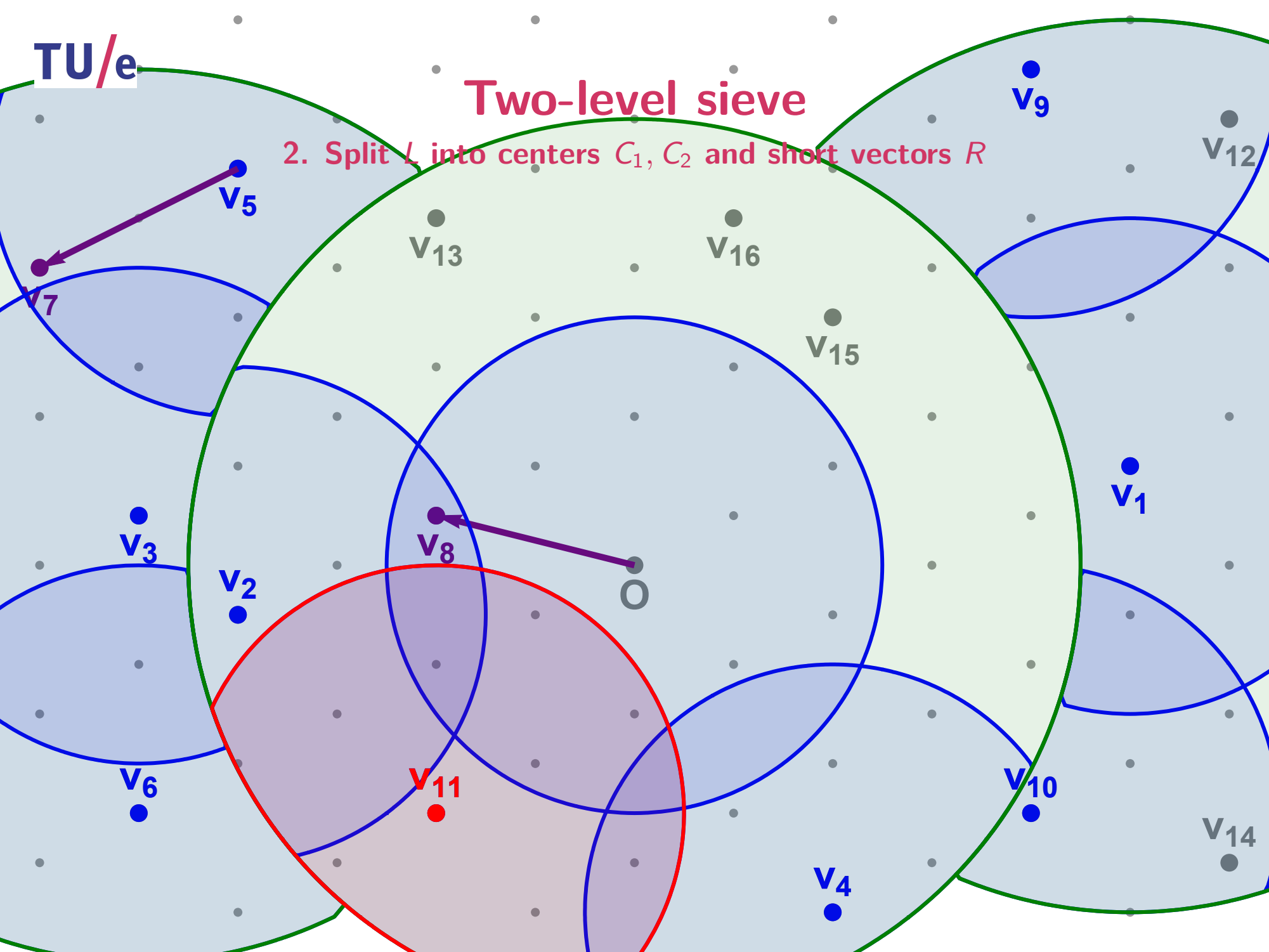
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

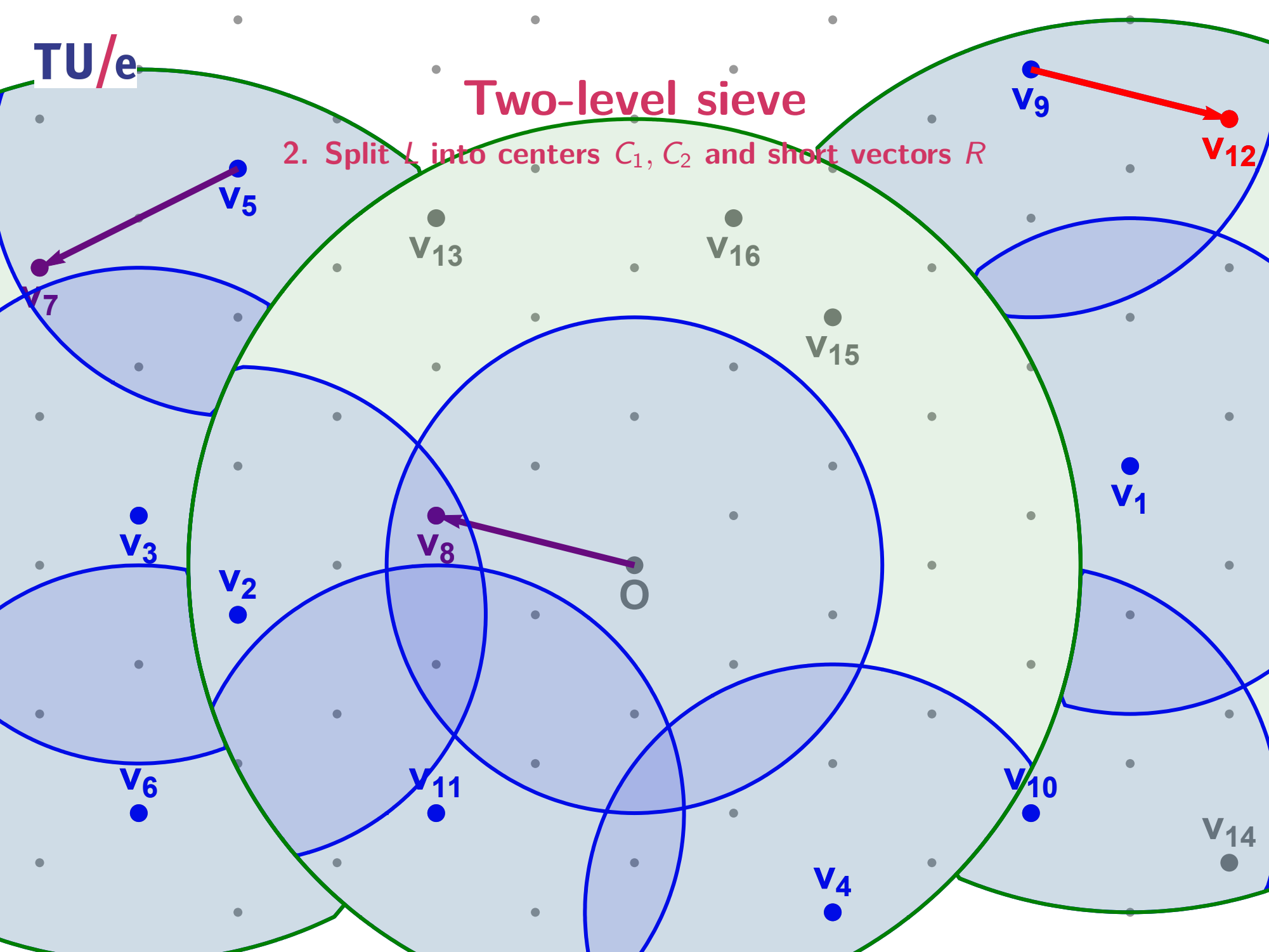
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

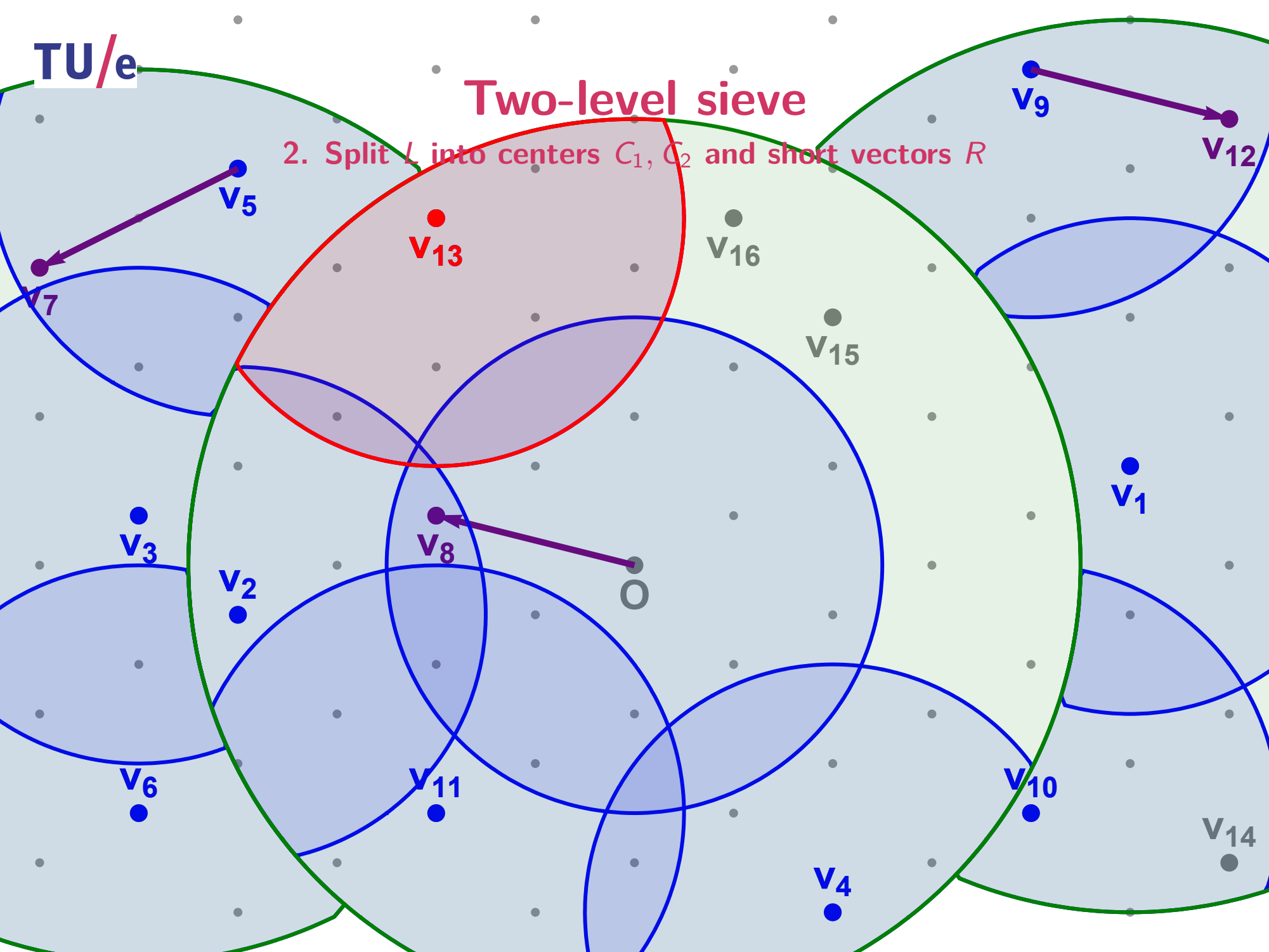
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

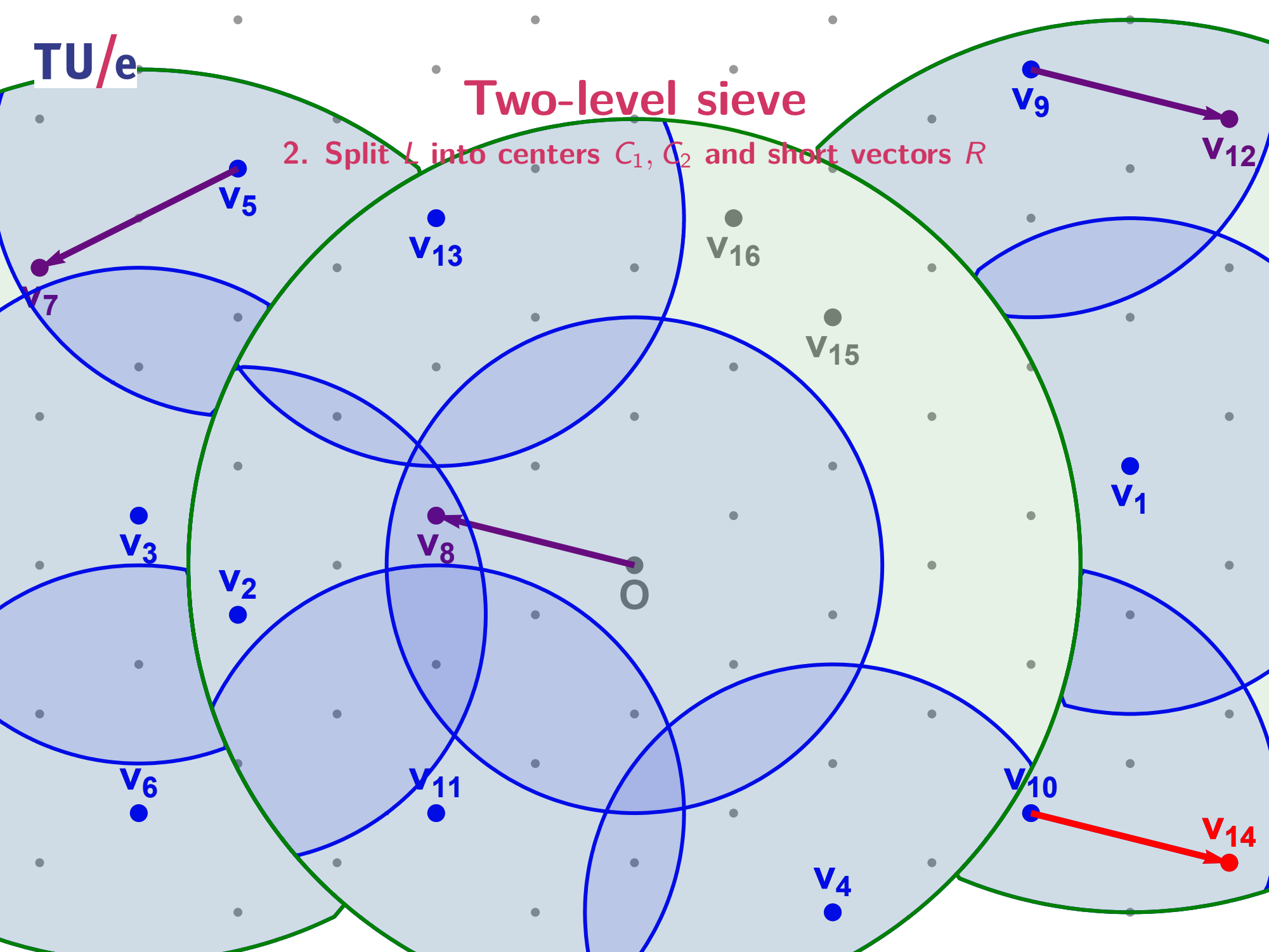
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

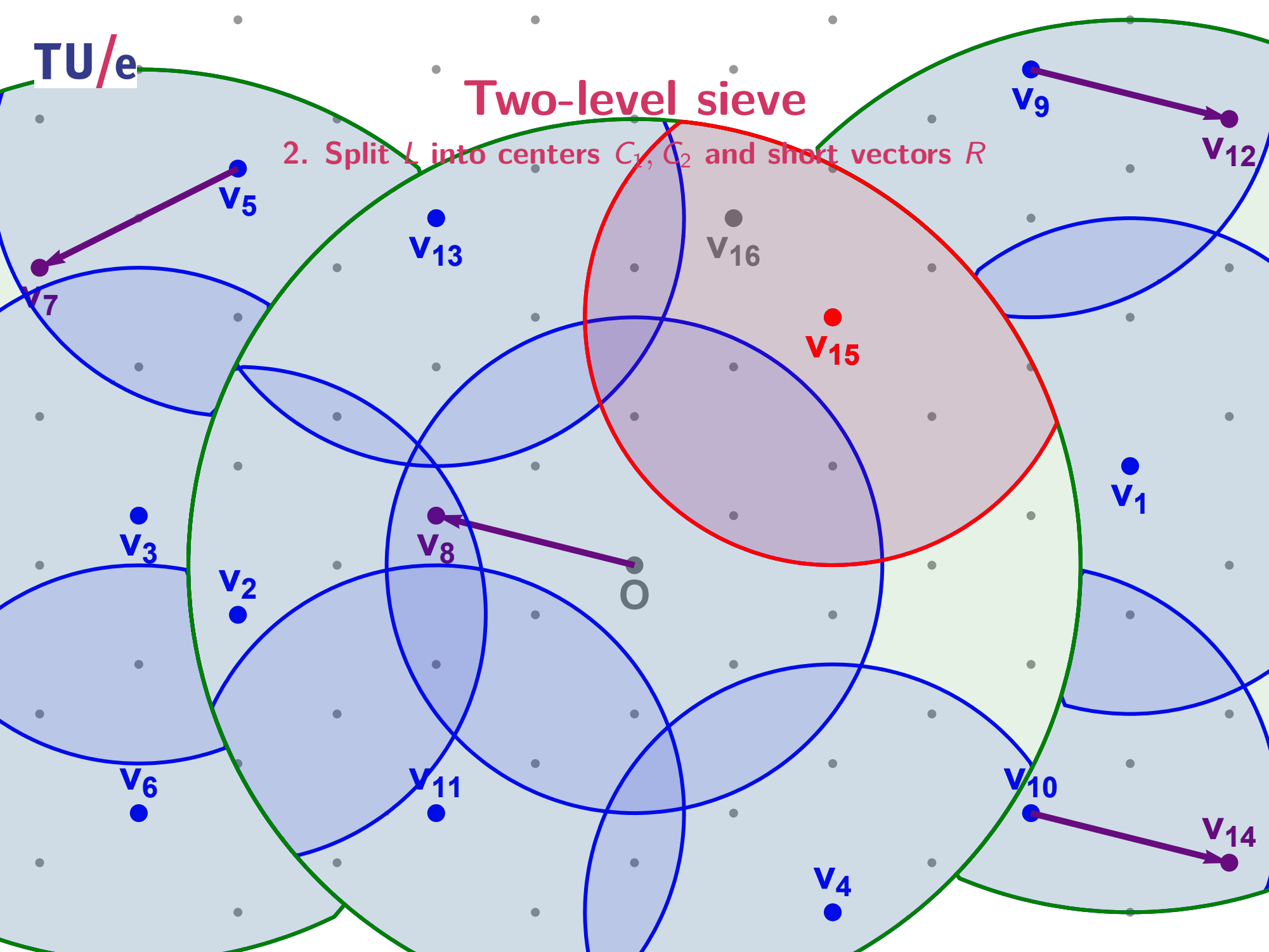
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

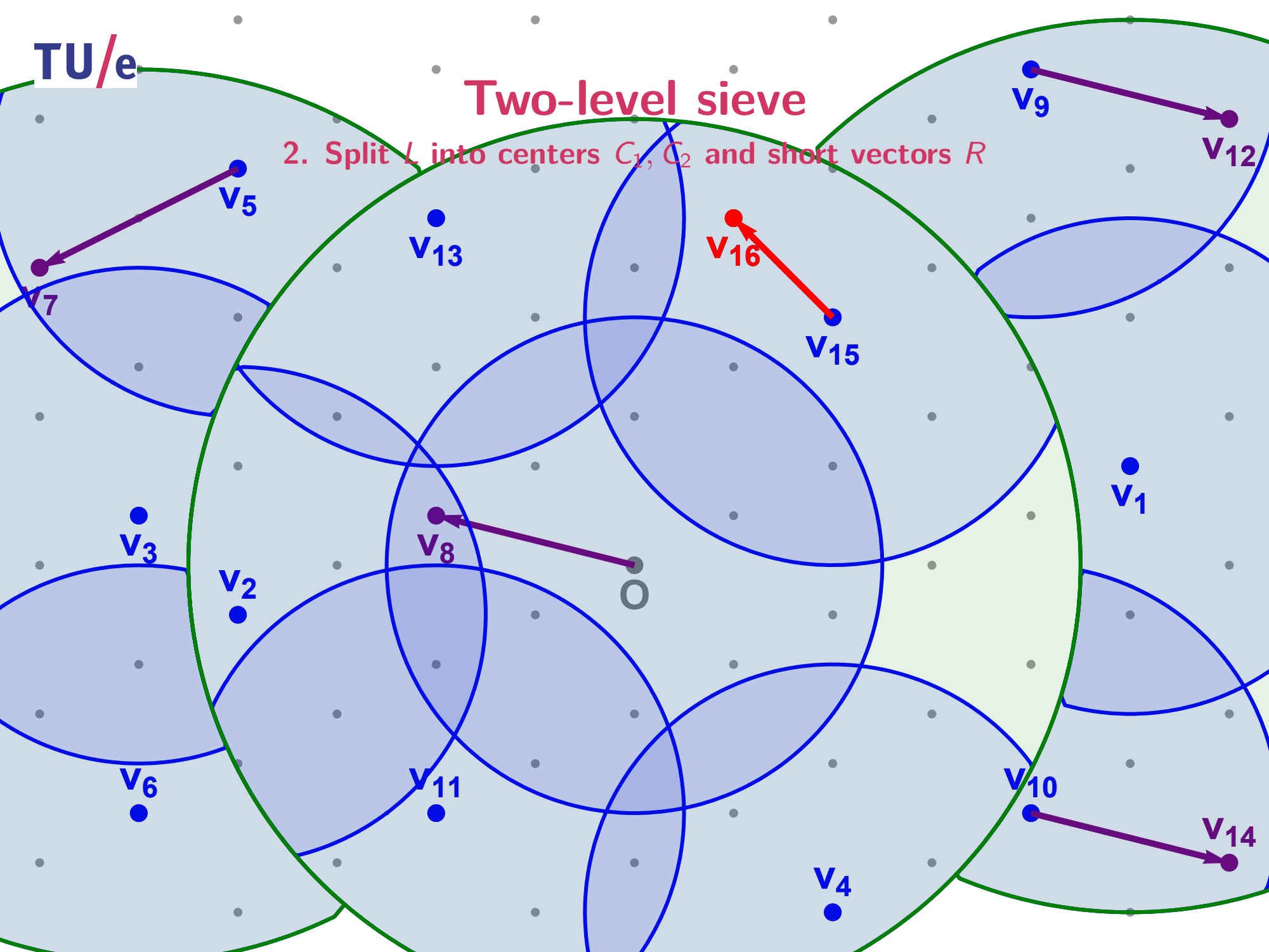
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

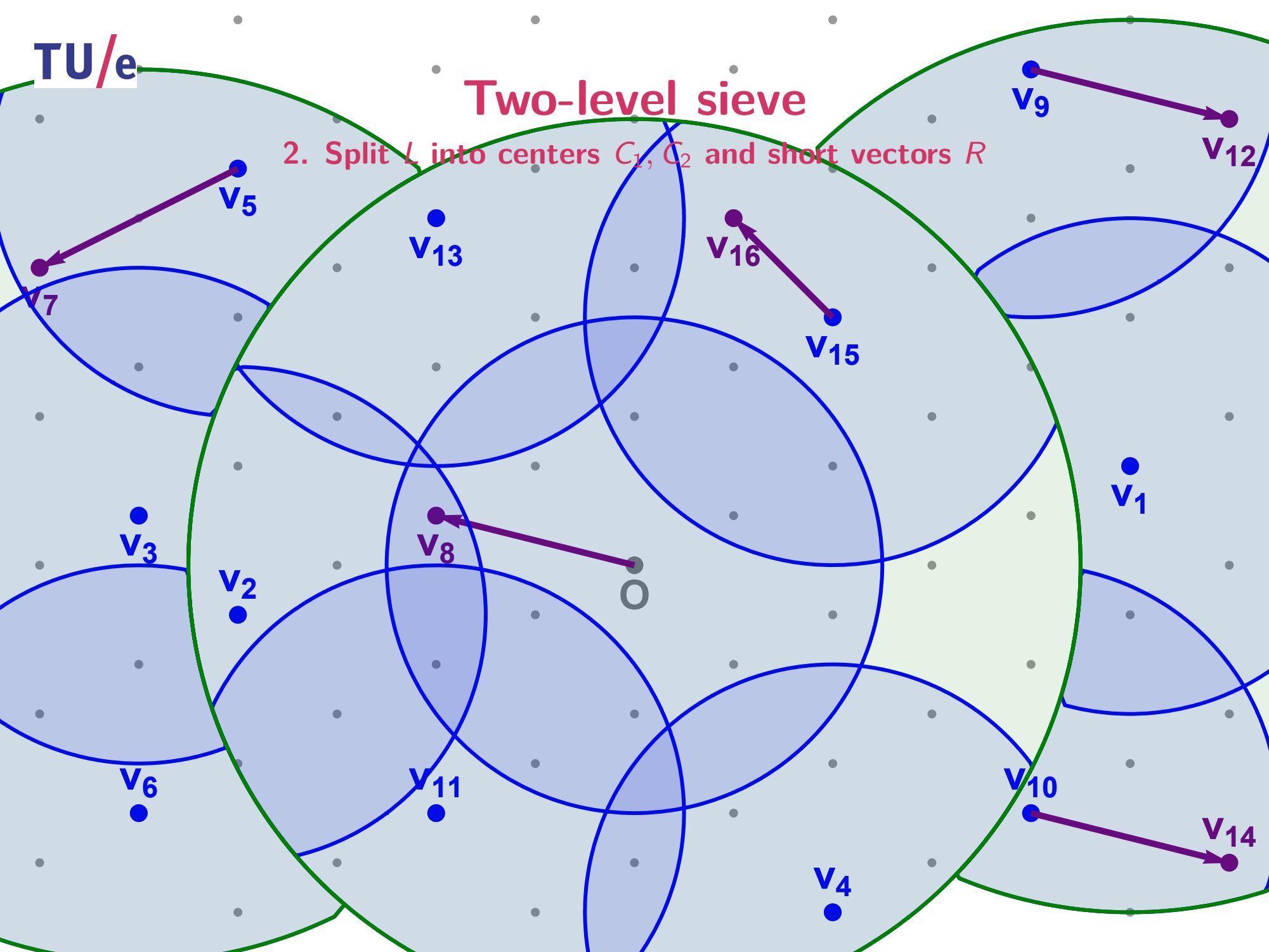
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

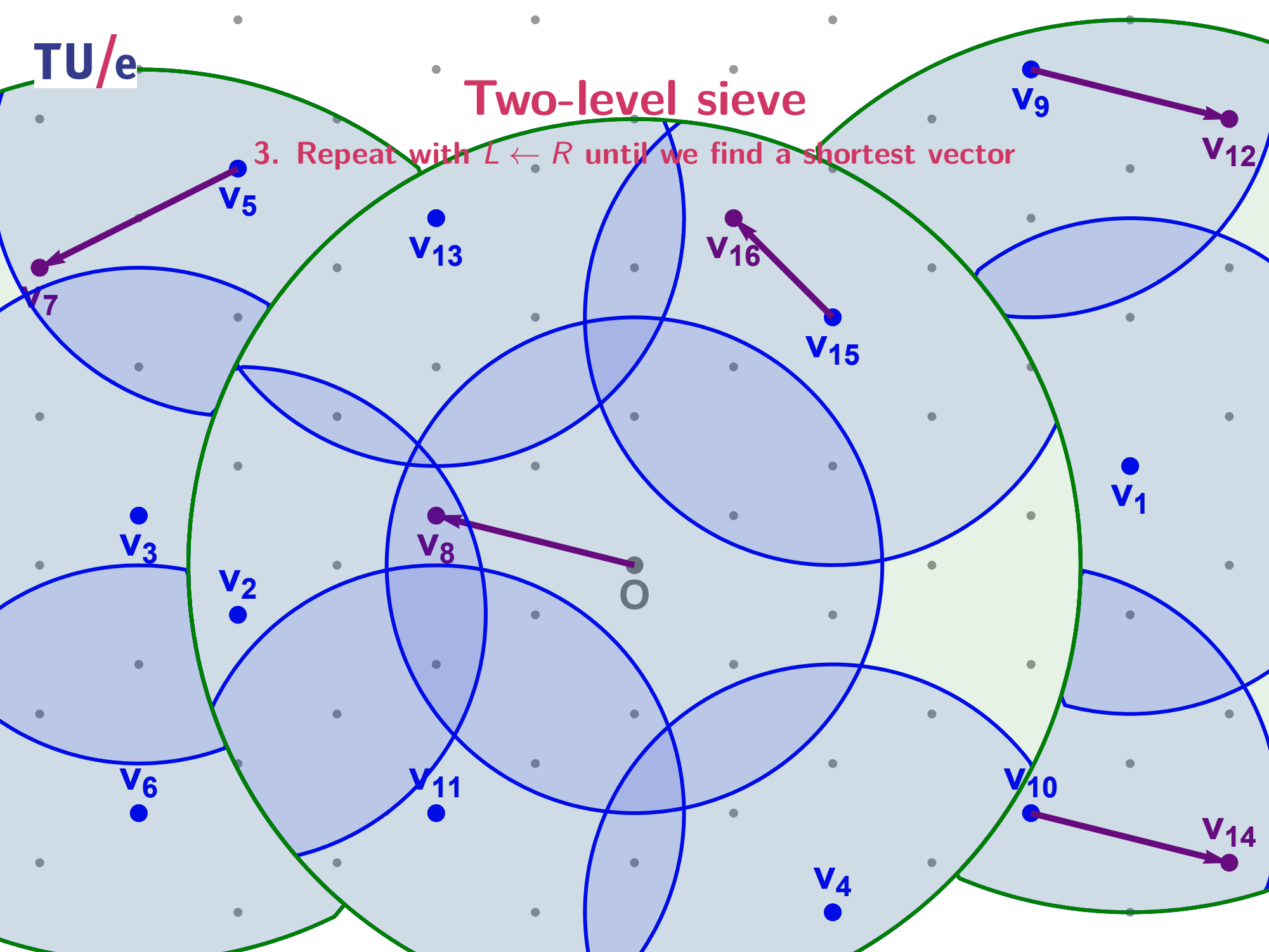
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

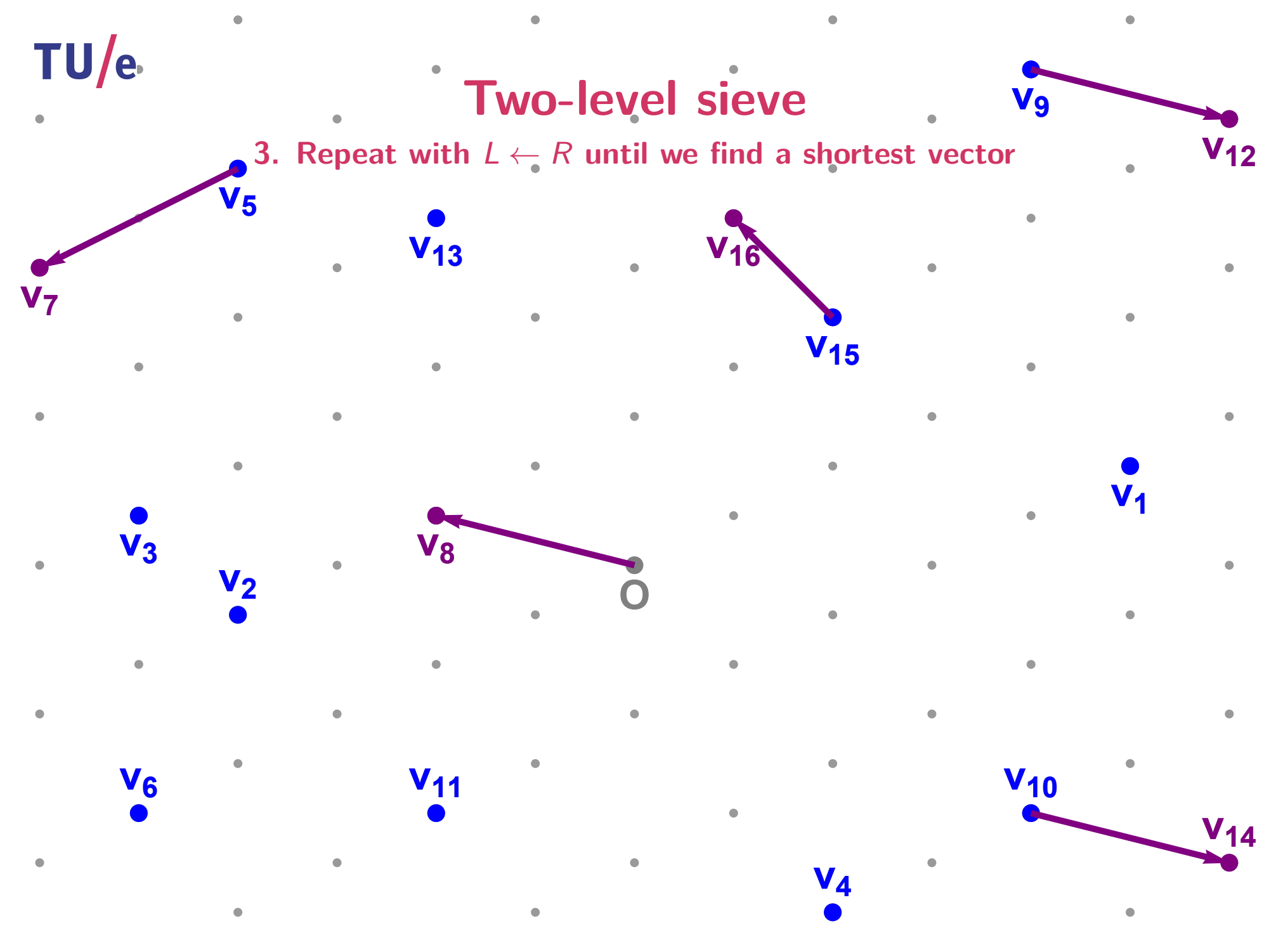
3. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

Two-level sieve

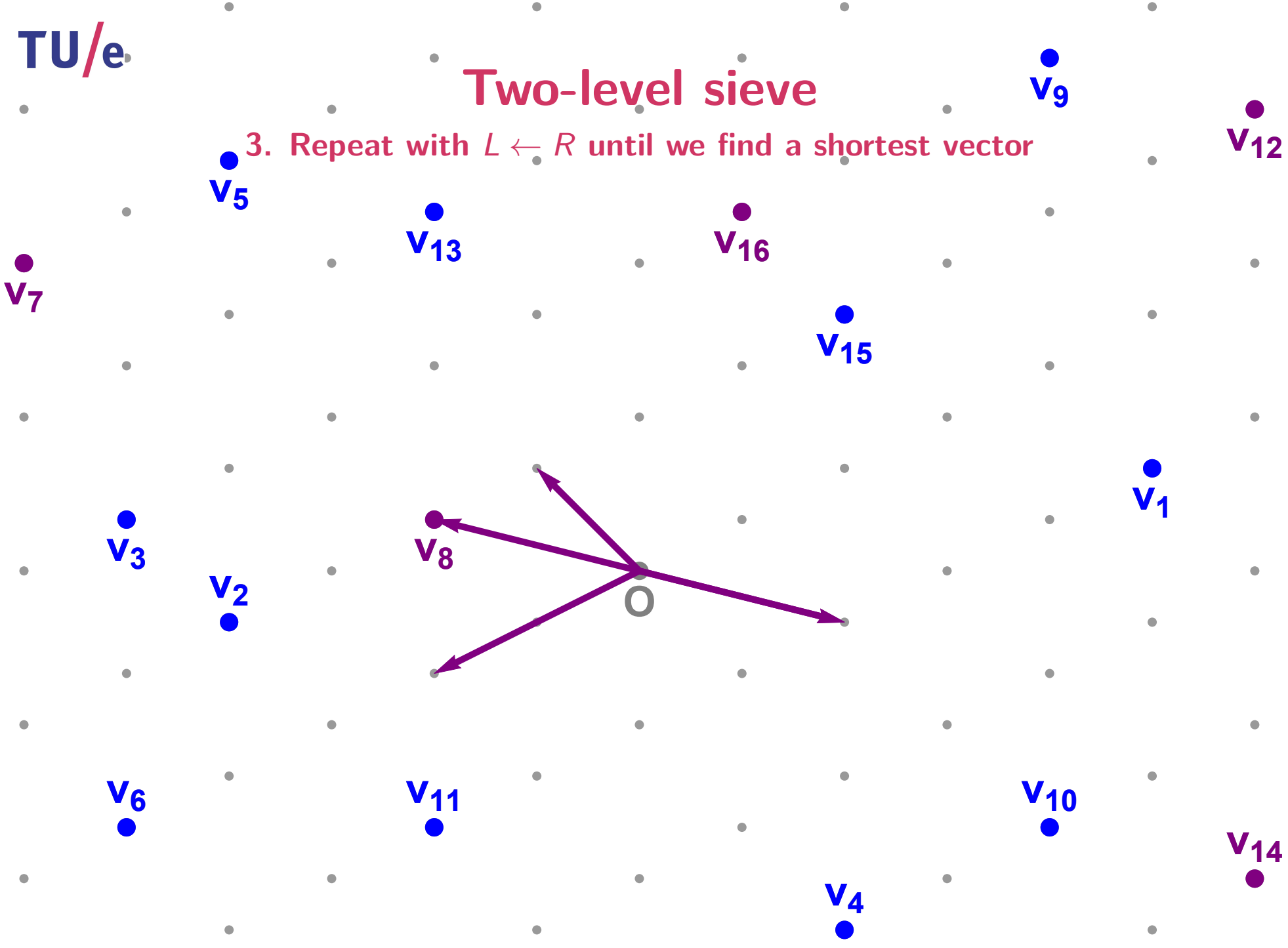
3. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

Two-level sieve

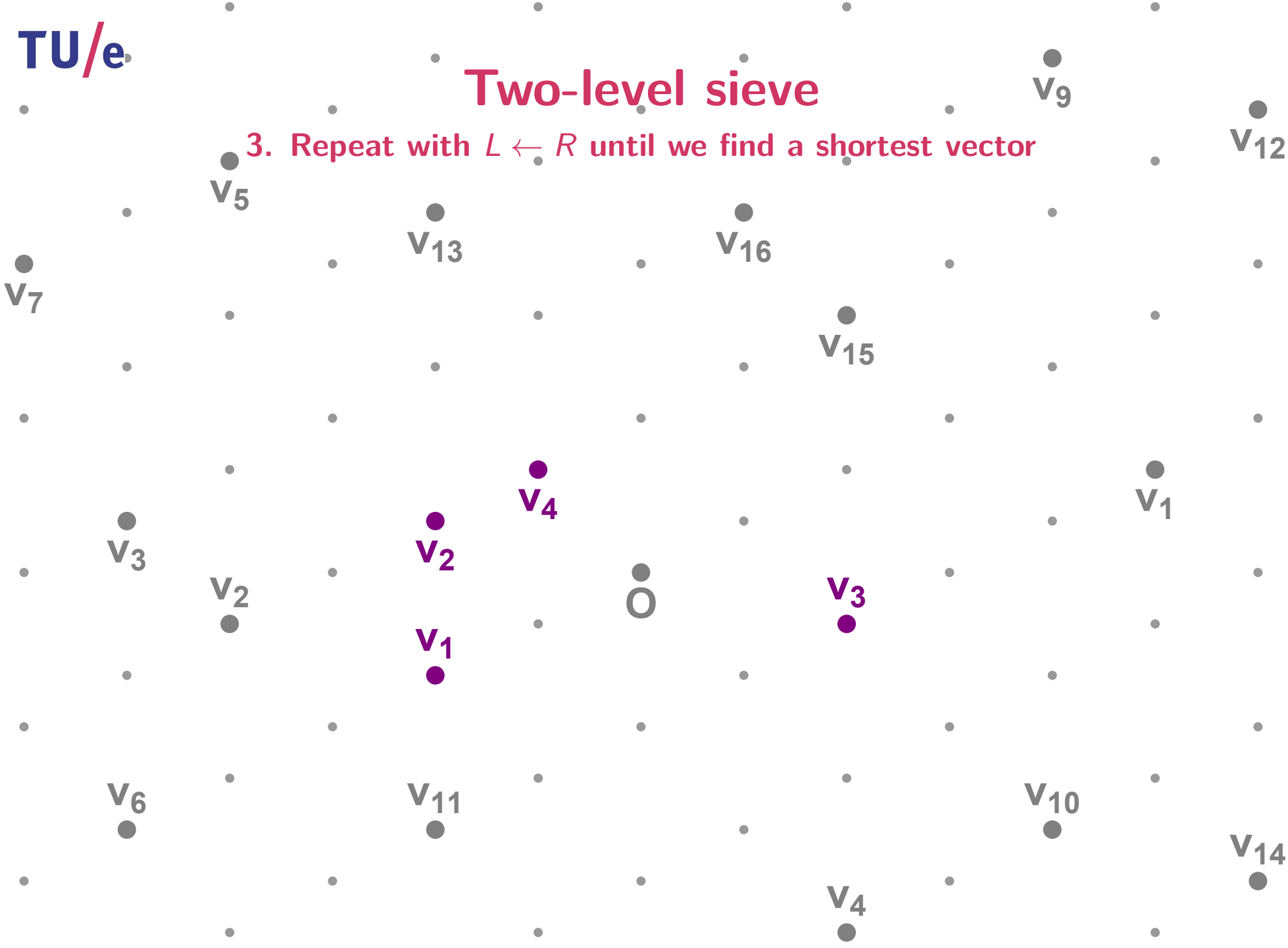
3. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

Two-level sieve

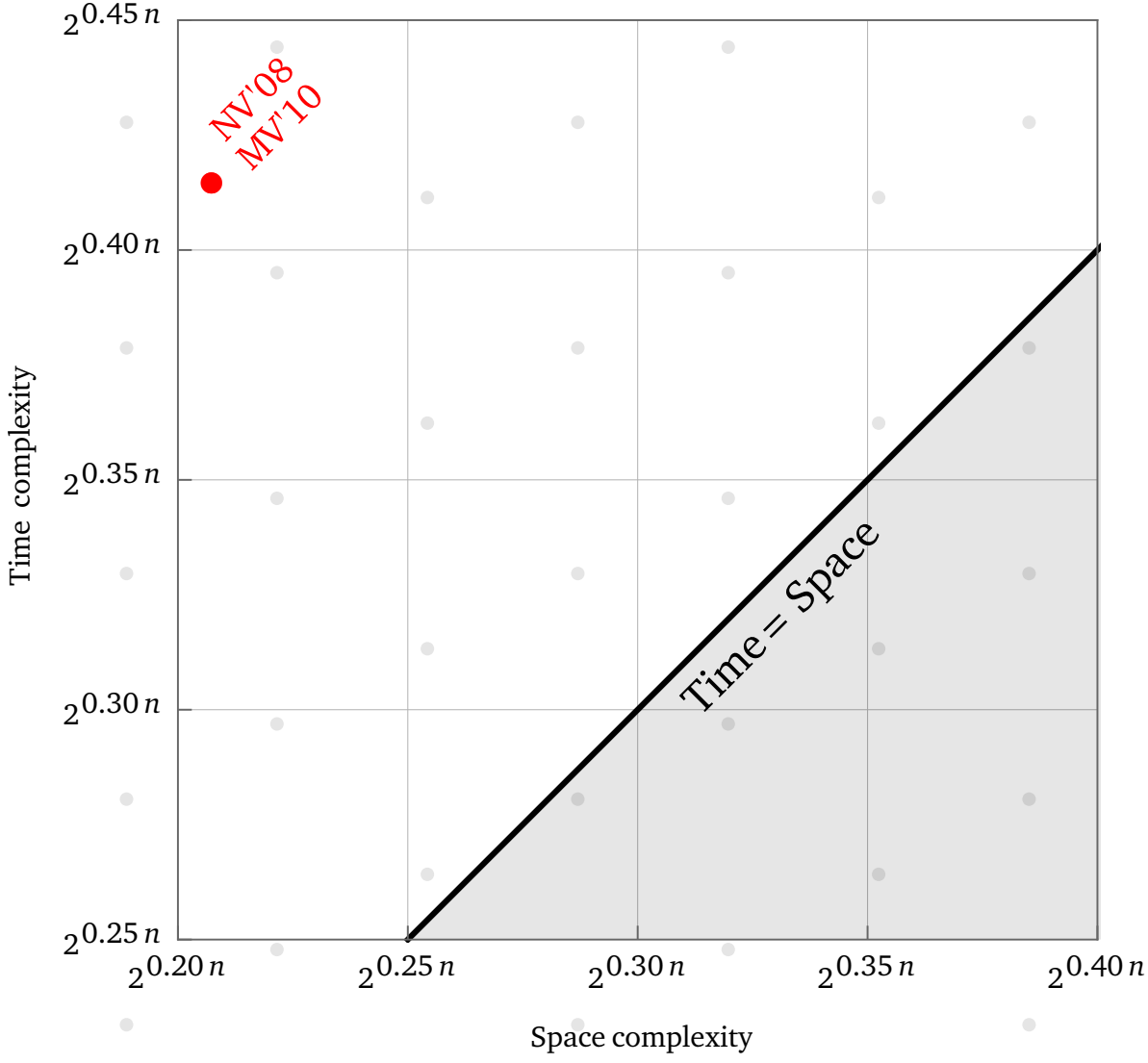
3. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

Two-level sieve

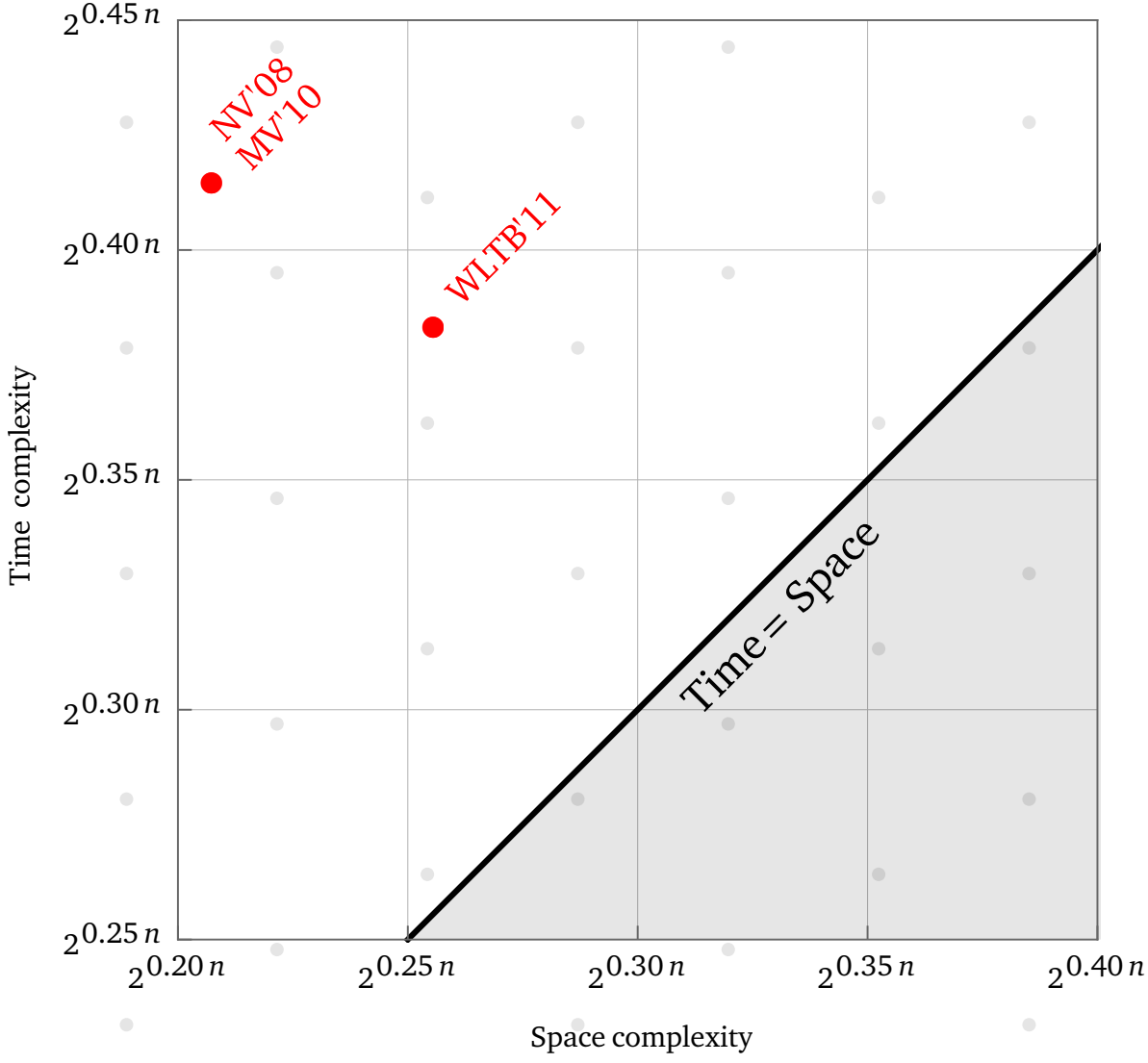
Space/time trade-off



TU/e

Two-level sieve

Space/time trade-off



TU/e

Three-level sieve

Overview

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

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

TU/e

Three-level sieve

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 $2^{0.2557n}$.

TU/e

Three-level sieve

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 $2^{0.2833n}$.

TU/e

Three-level sieve

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 $2^{0.2833n}$.

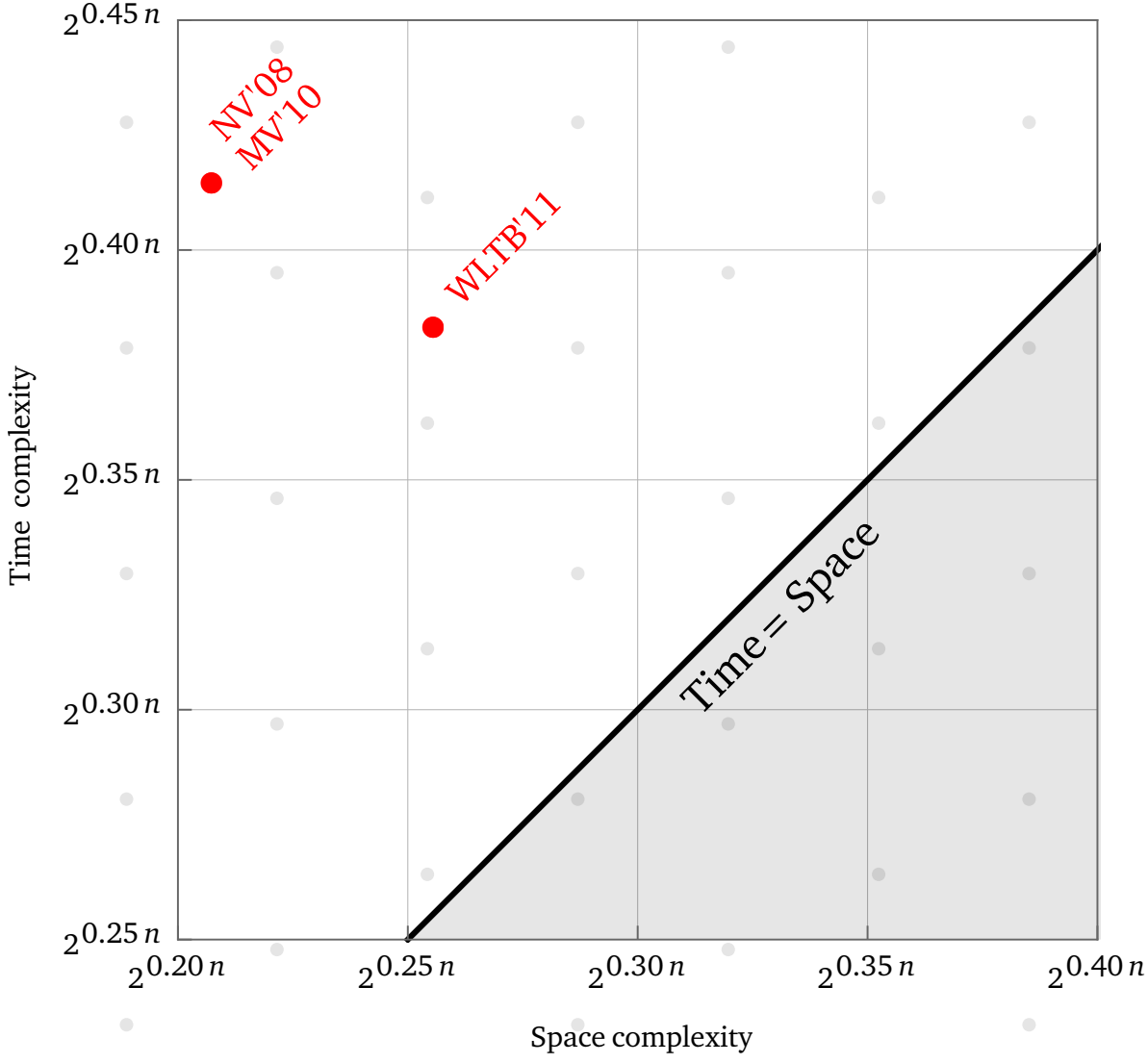
Conjecture

The four-level sieve runs in time $2^{0.3774n}$ and space $2^{0.2925n}$, and higher-level sieves are not faster than this.

TU/e

Three-level sieve

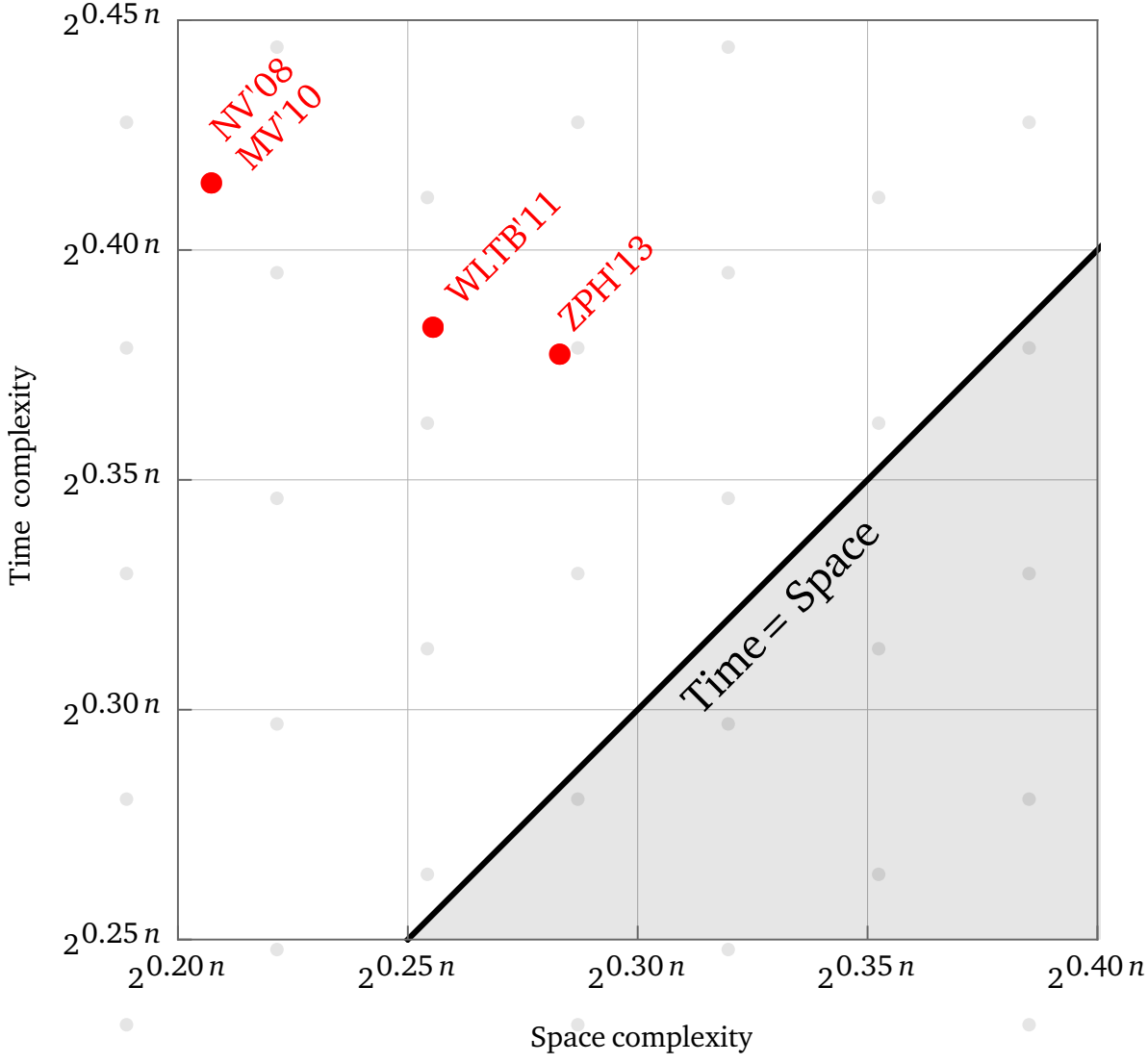
Space/time trade-off



TU/e

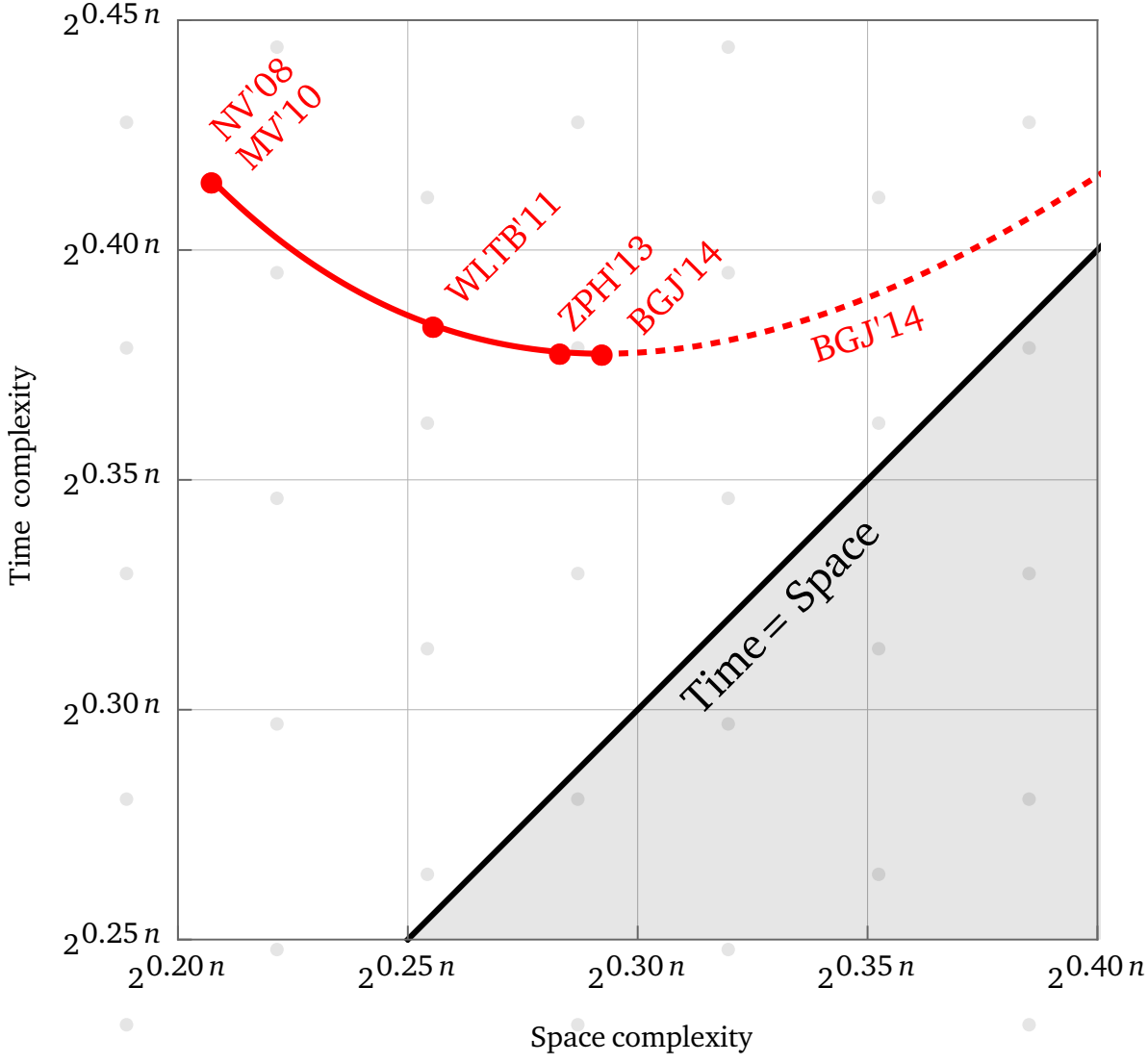
Three-level sieve

Space/time trade-off



Overlattice sieving

Space/time trade-off



TU/e

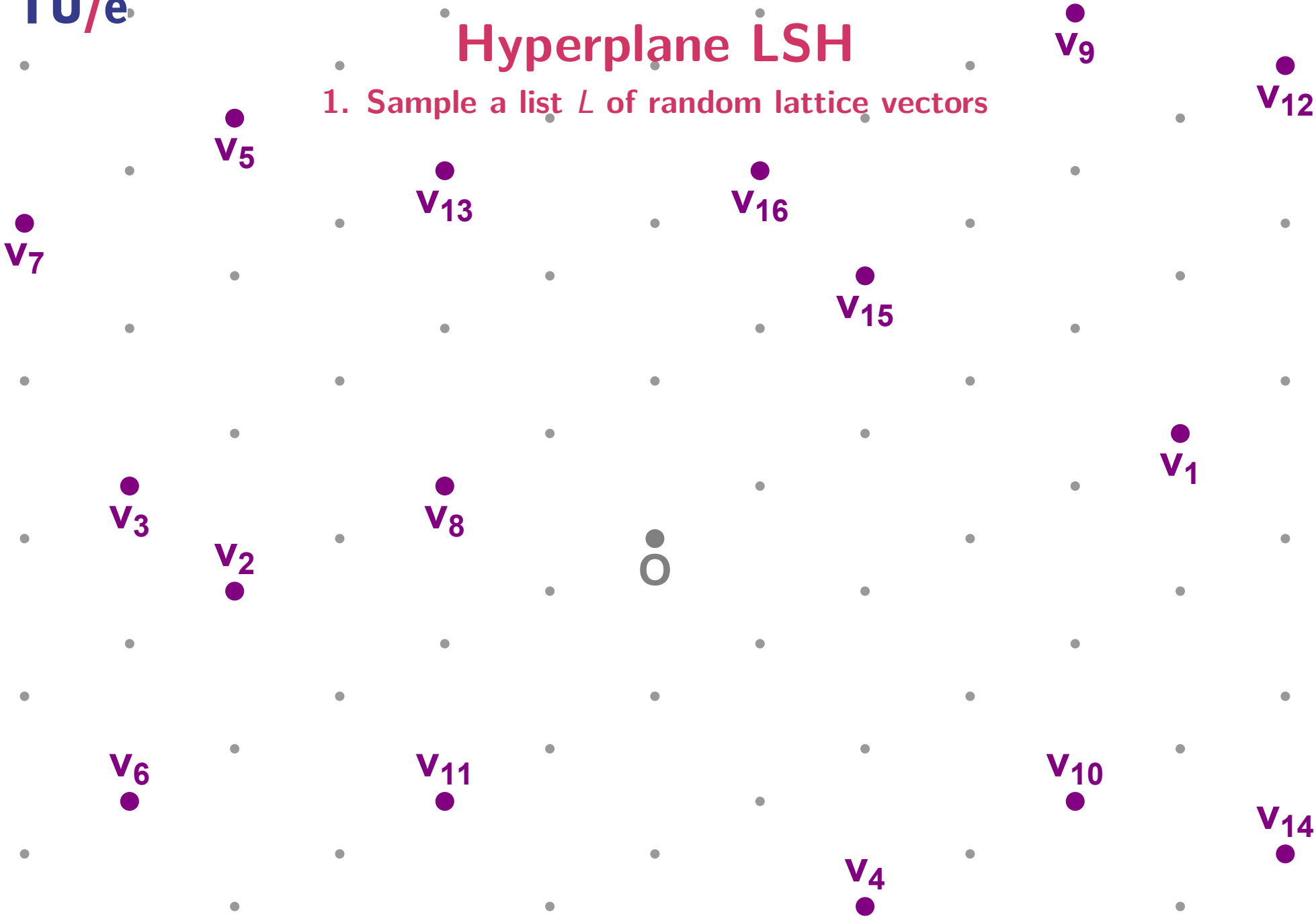
Hyperplane LSH

1. Sample a list L of random lattice vectors

TU/e

Hyperplane LSH

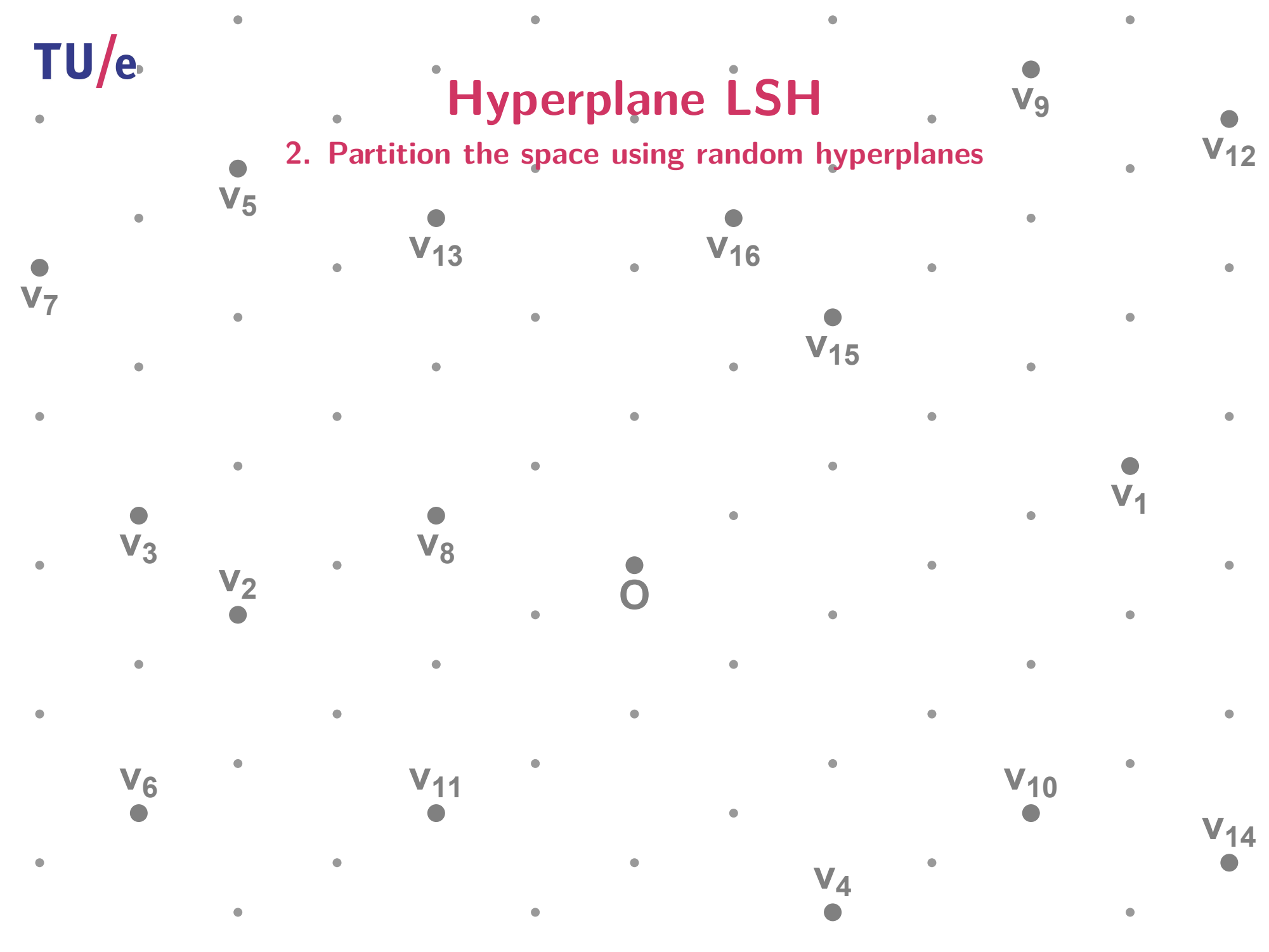
1. Sample a list L of random lattice vectors



TU/e

Hyperplane LSH

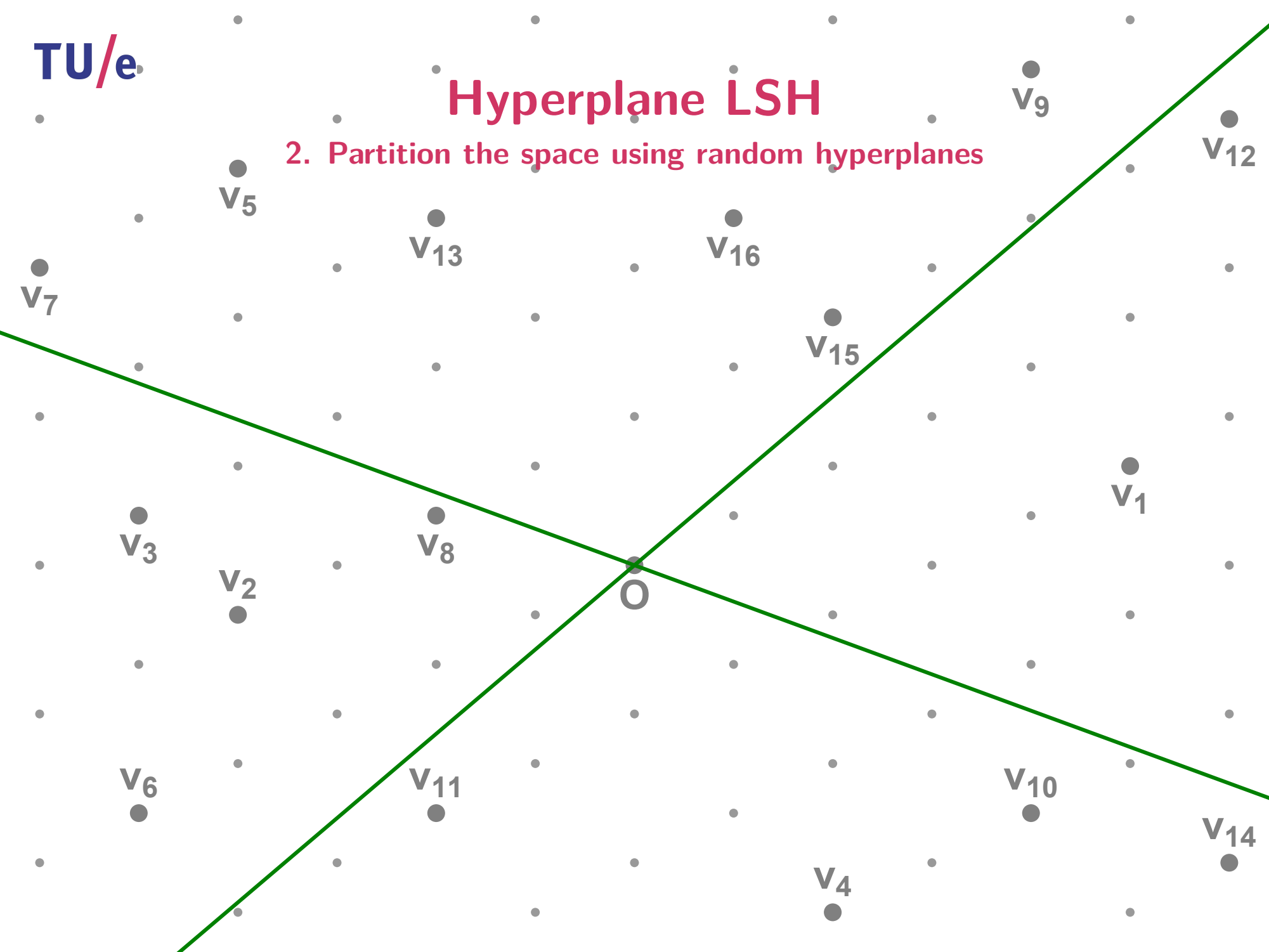
2. Partition the space using random hyperplanes



TU/e

Hyperplane LSH

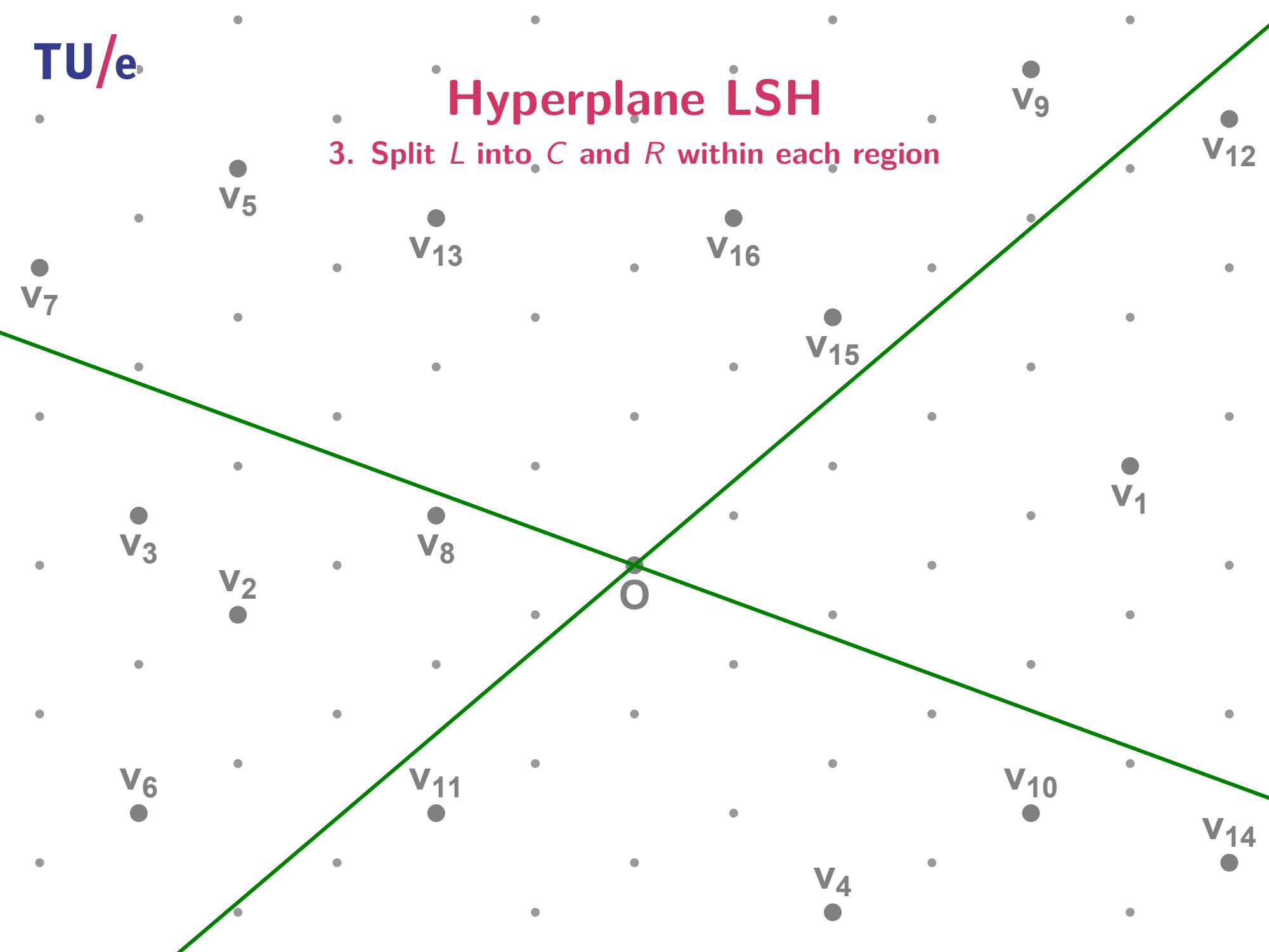
2. Partition the space using random hyperplanes



TU/e

Hyperplane LSH

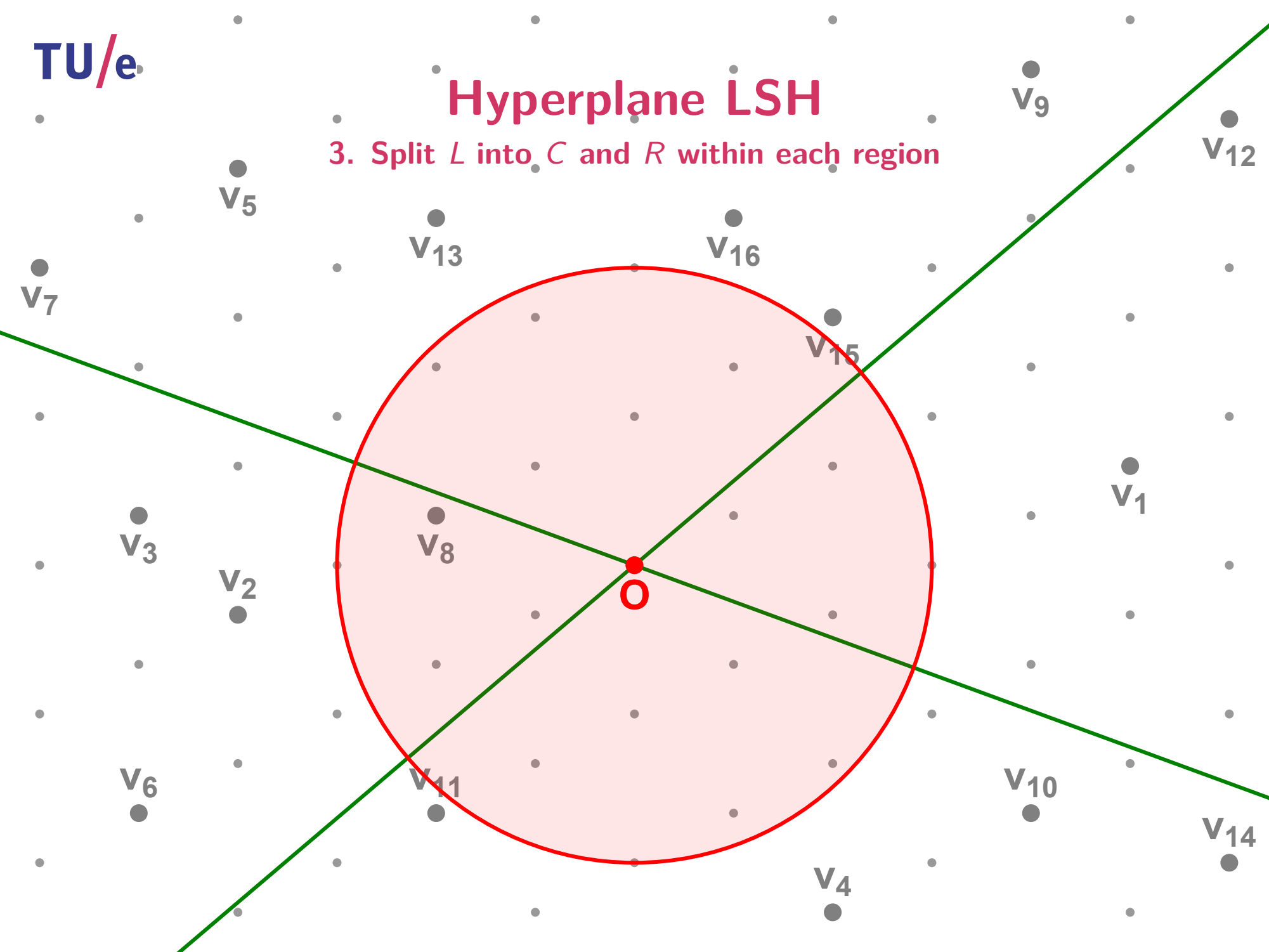
3. Split L into C and R within each region



TU/e

Hyperplane LSH

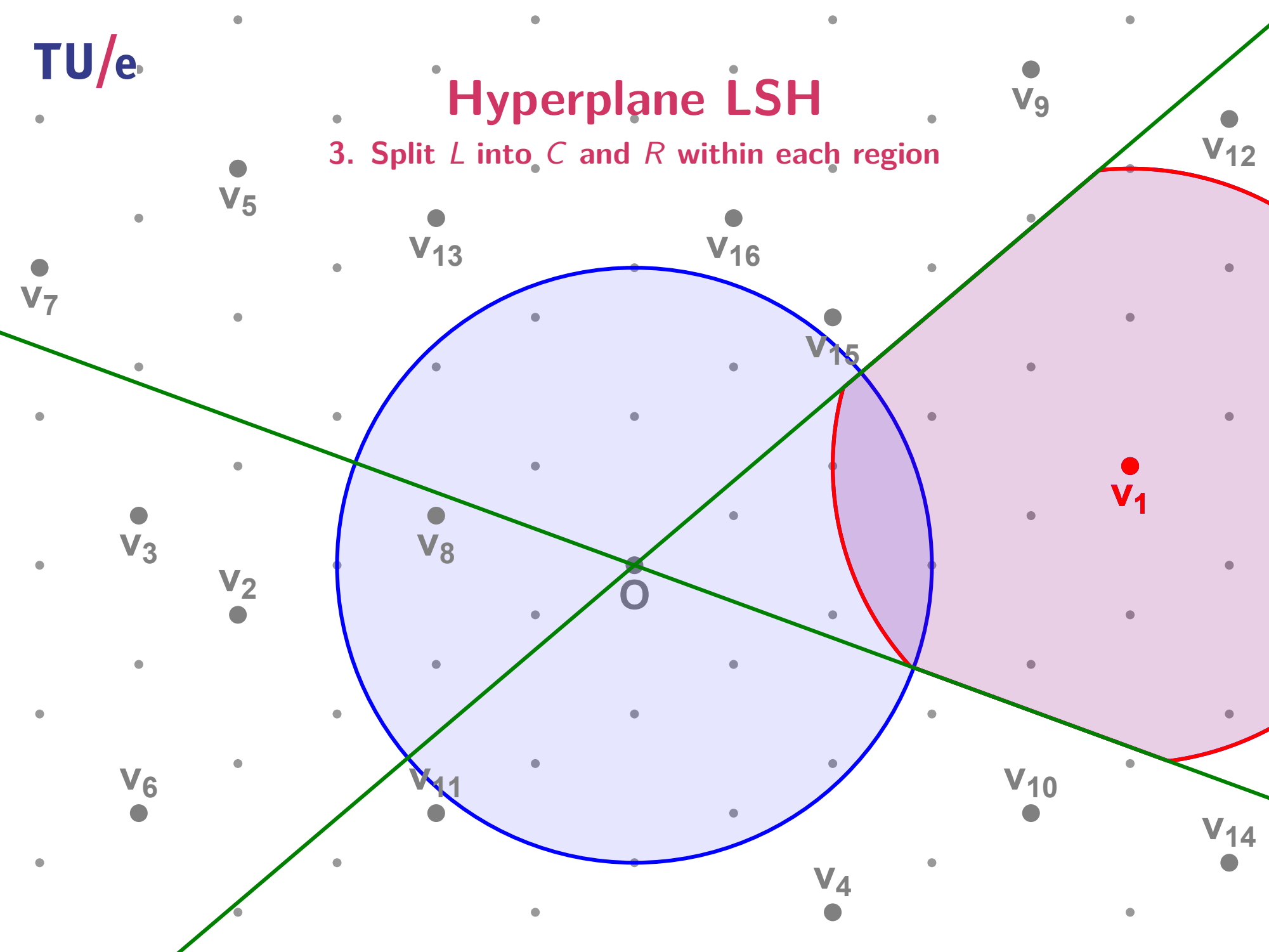
3. Split L into C and R within each region



TU/e

Hyperplane LSH

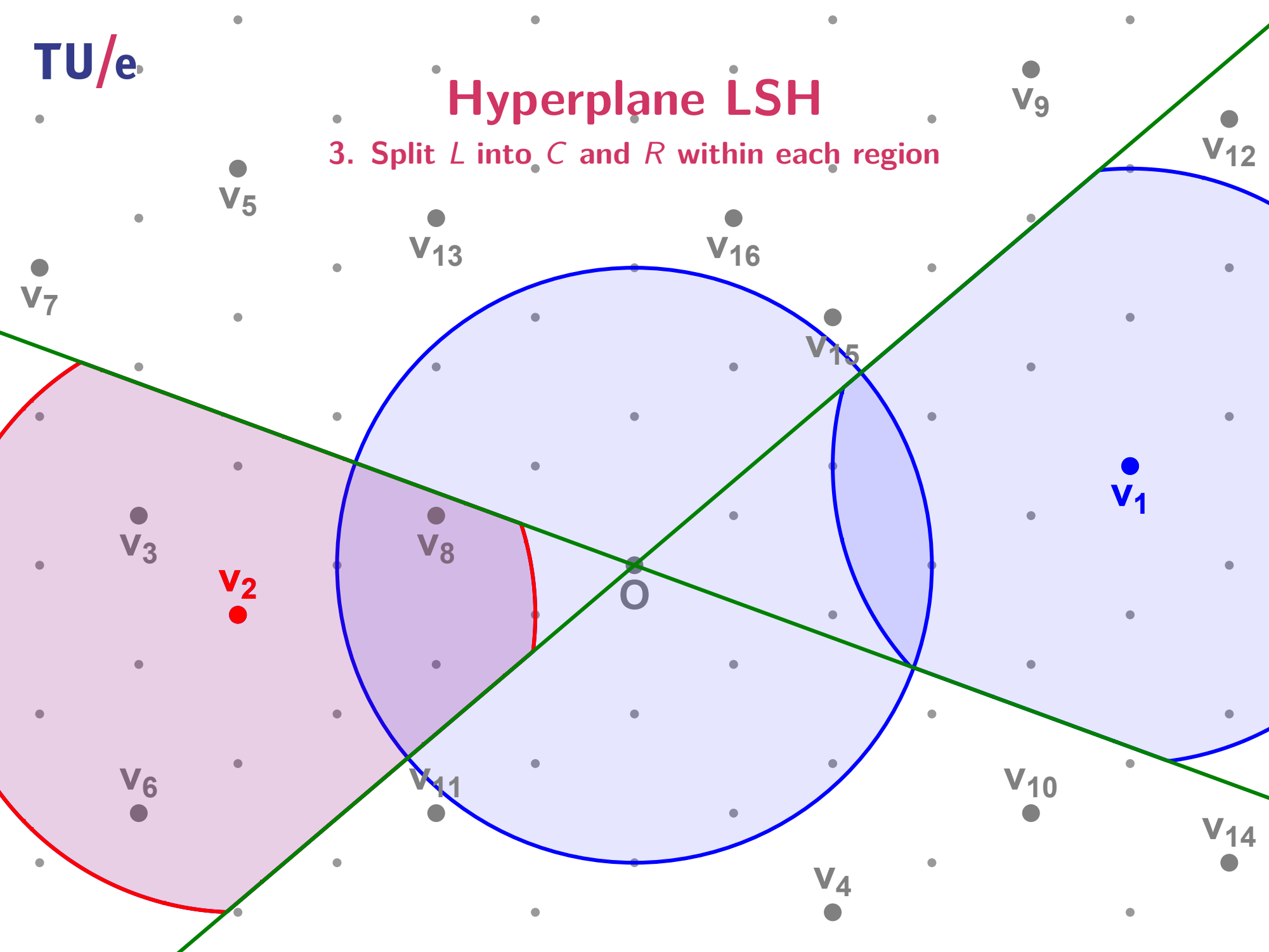
3. Split L into C and R within each region



TU/e

Hyperplane LSH

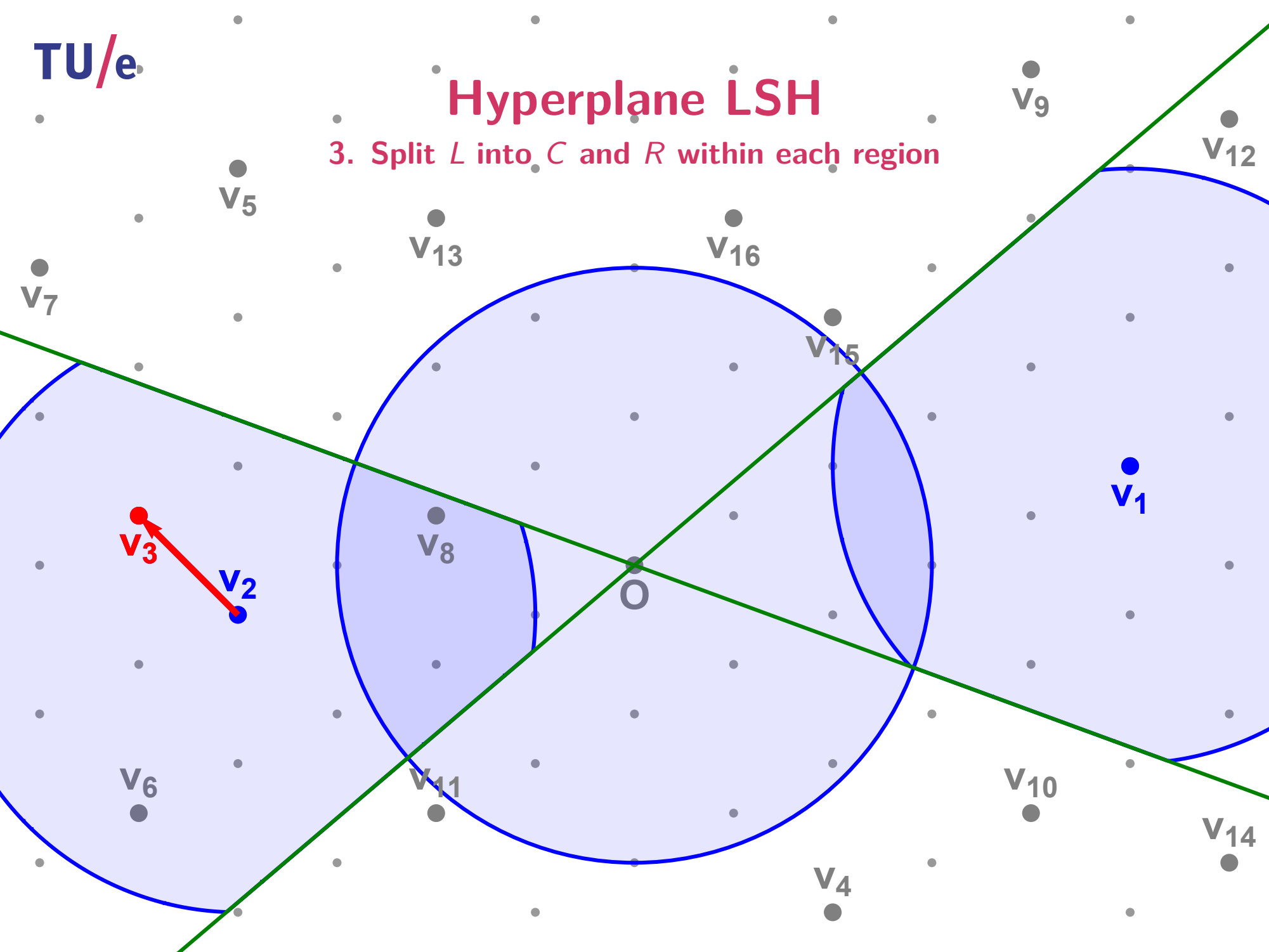
3. Split L into C and R within each region



TU/e

Hyperplane LSH

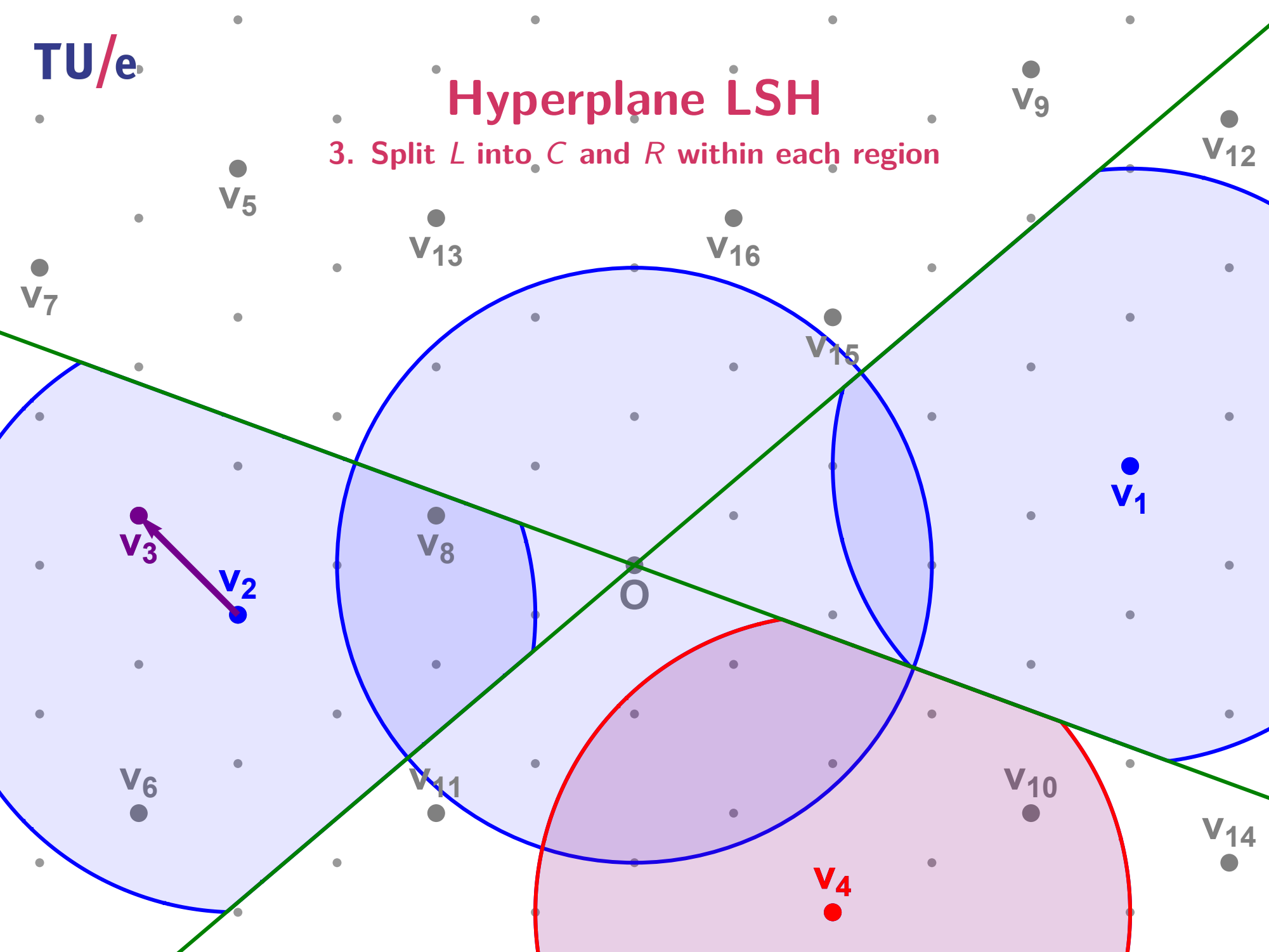
3. Split L into C and R within each region



TU/e

Hyperplane LSH

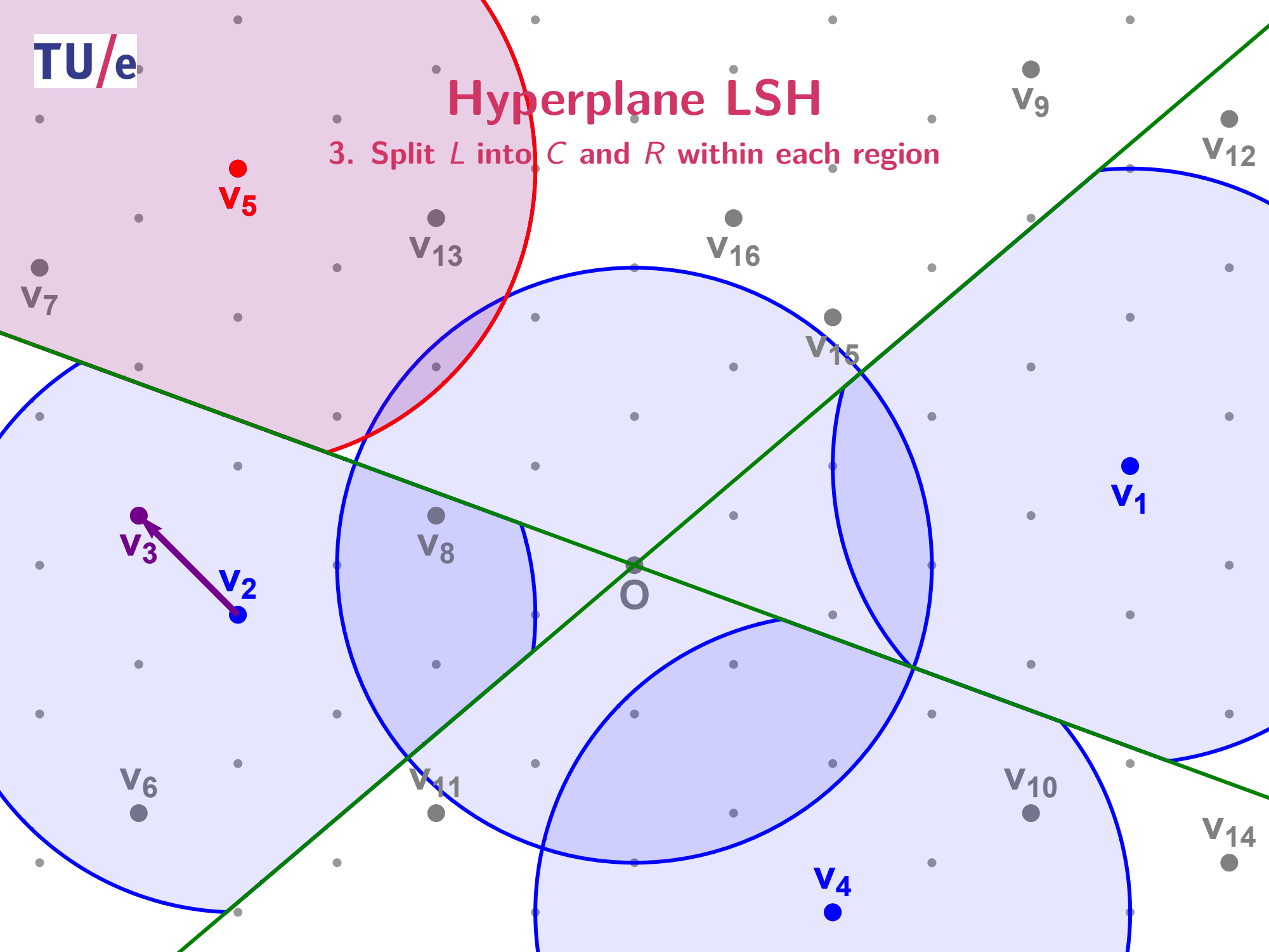
3. Split L into C and R within each region



TU/e

Hyperplane LSH

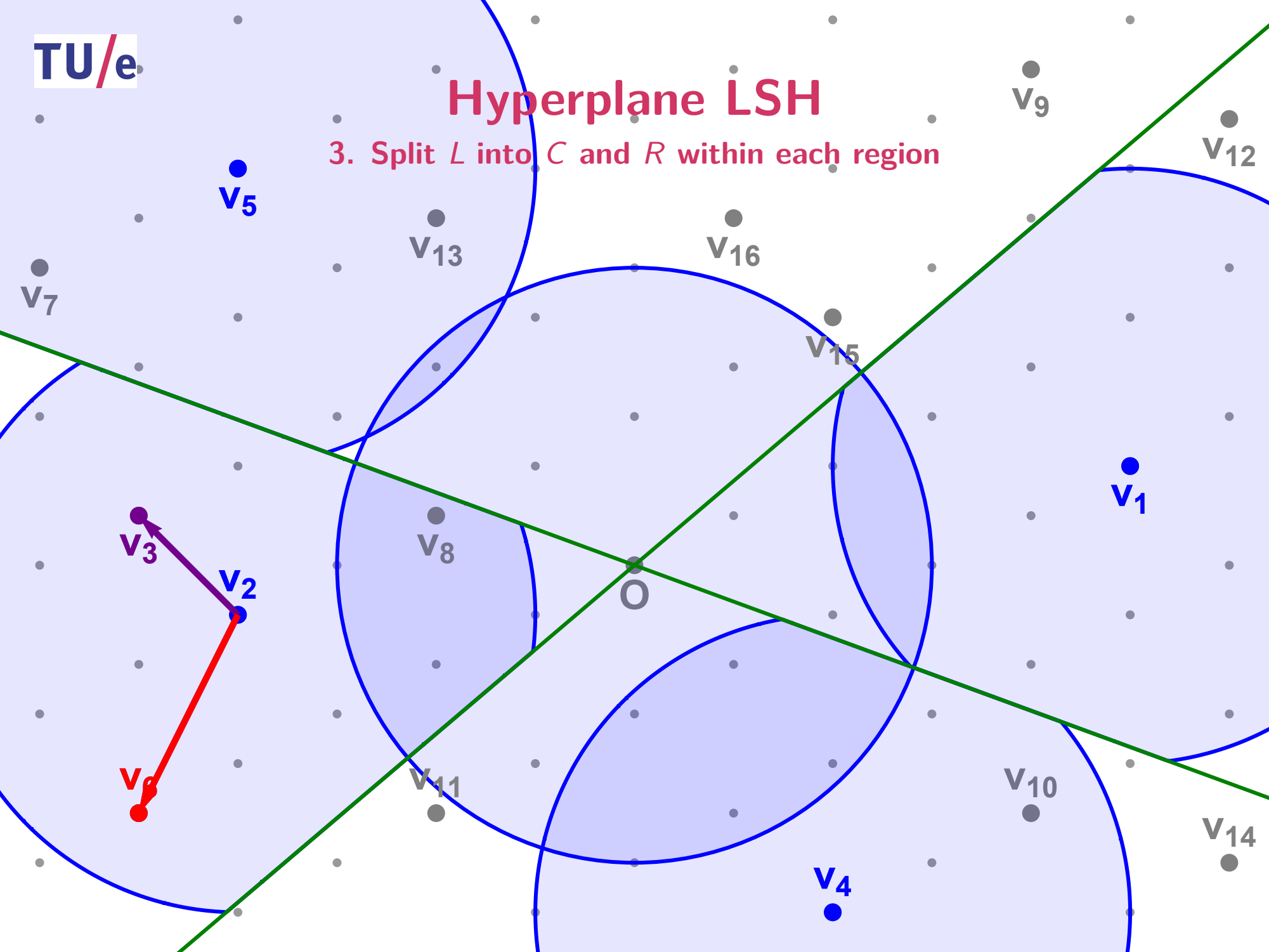
3. Split L into C and R within each region



TU/e

Hyperplane LSH

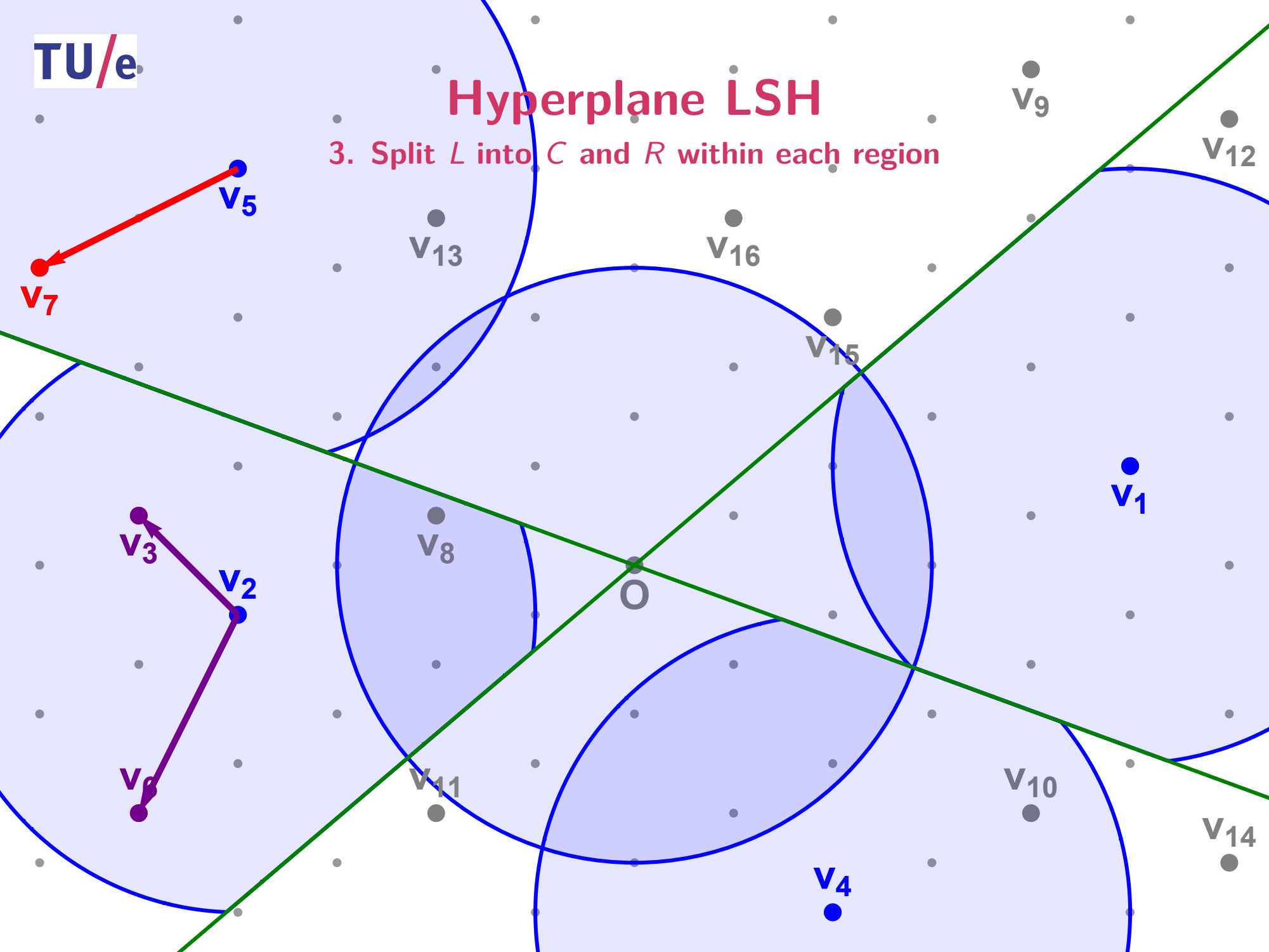
3. Split L into C and R within each region



TU/e

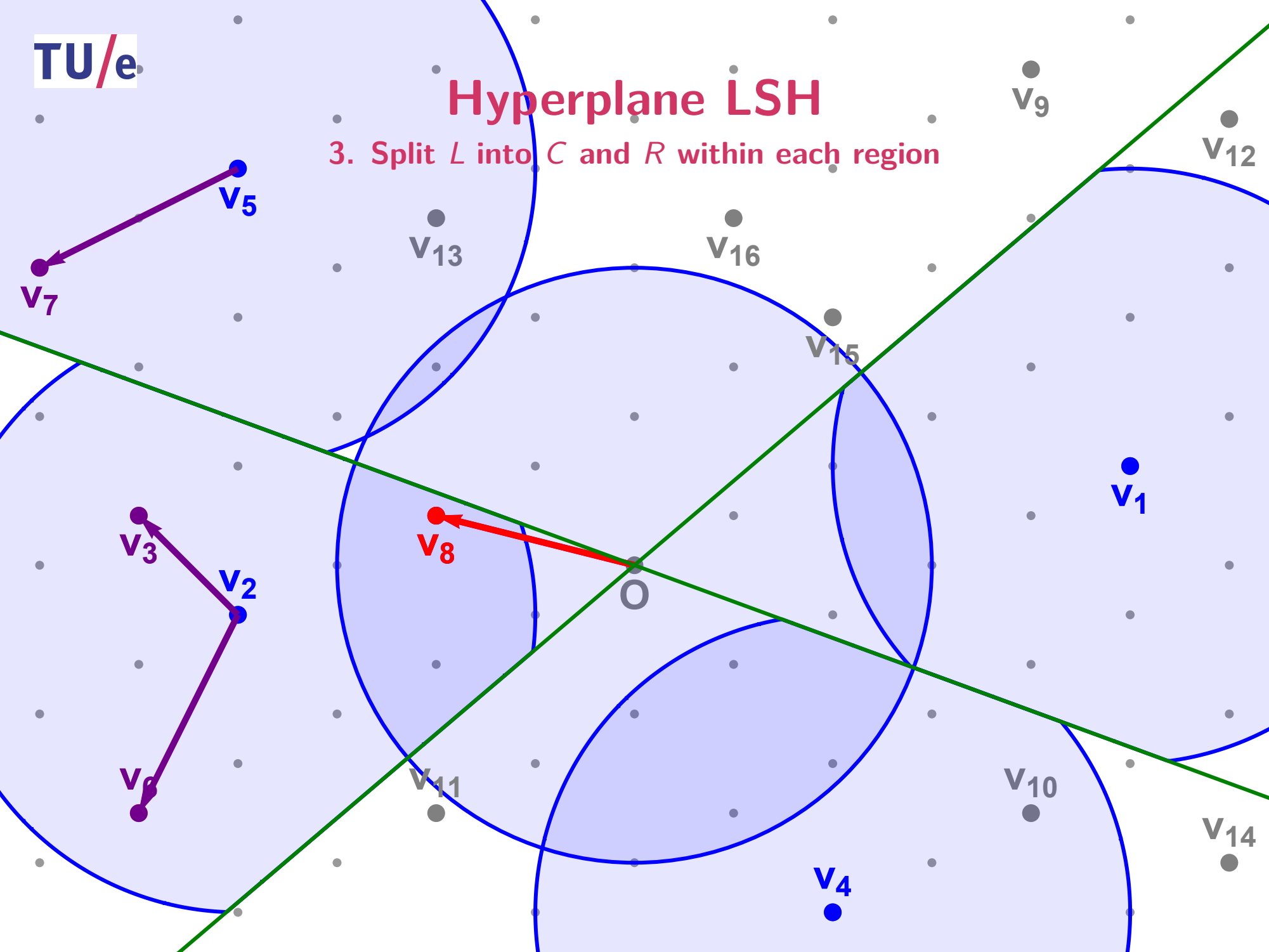
Hyperplane LSH

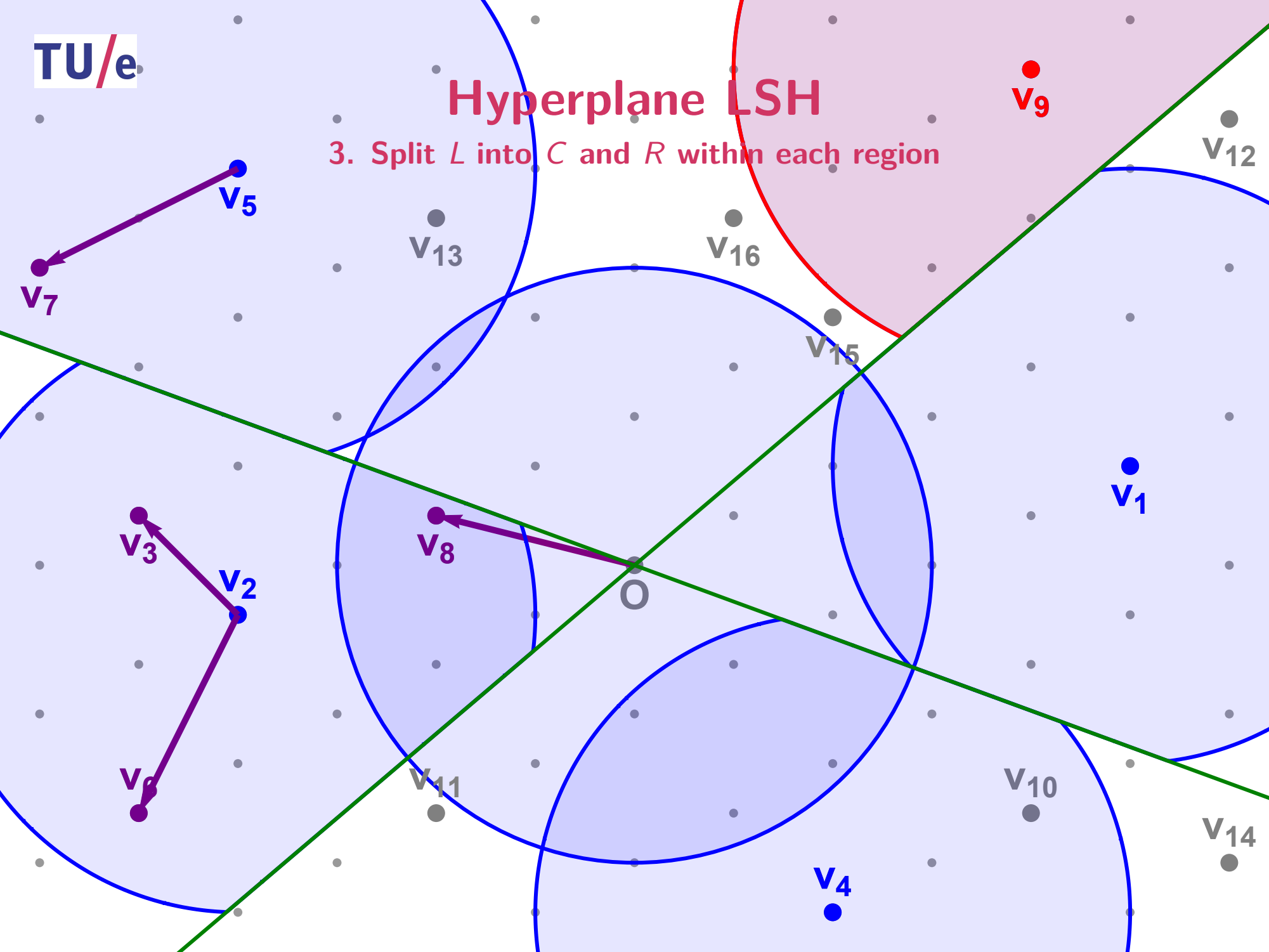
3. Split L into C and R within each region

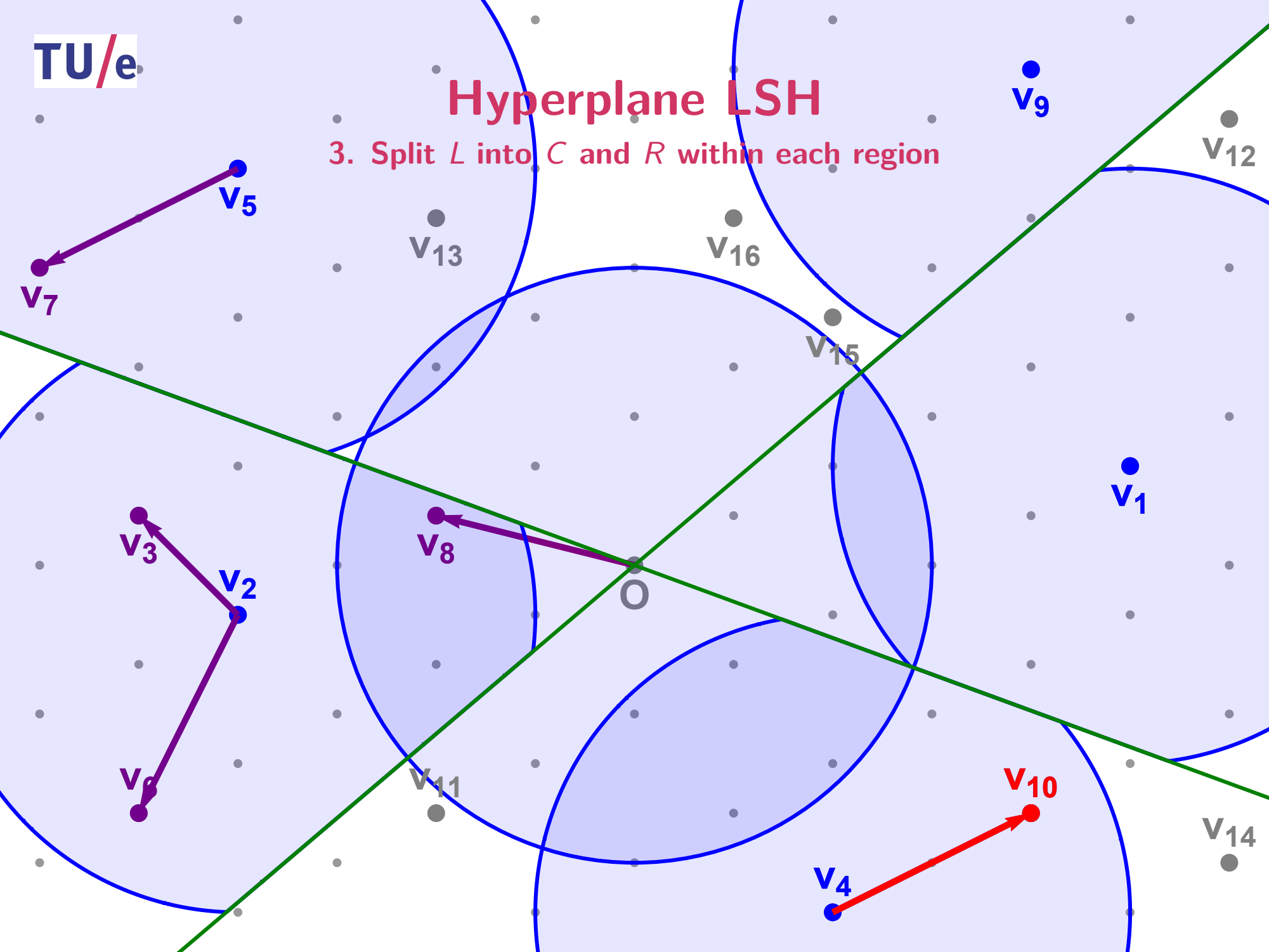


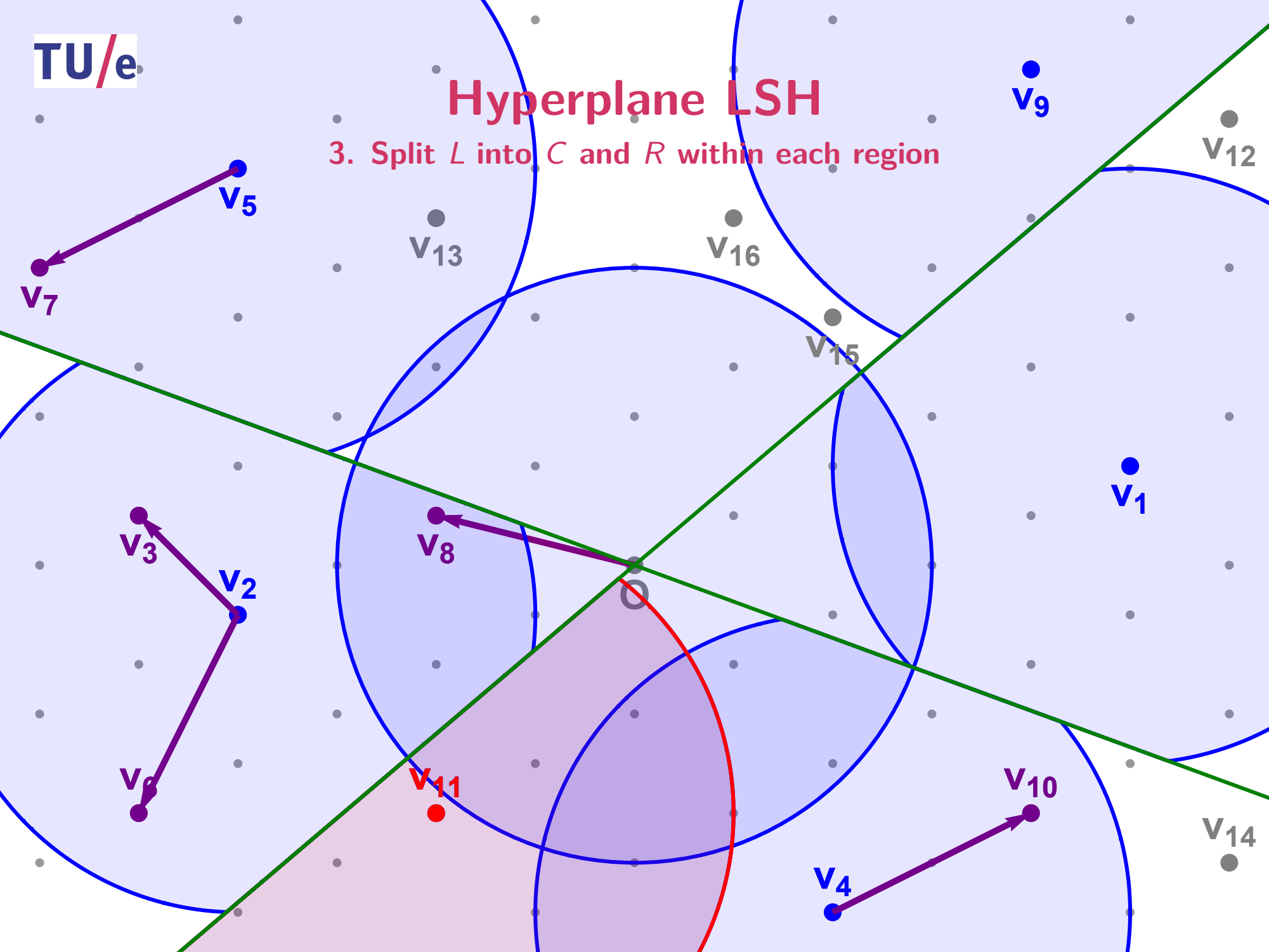
Hyperplane LSH

3. Split L into C and R within each region




TU/e
Hyperplane LSH
3. Split L into C and R within each region
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

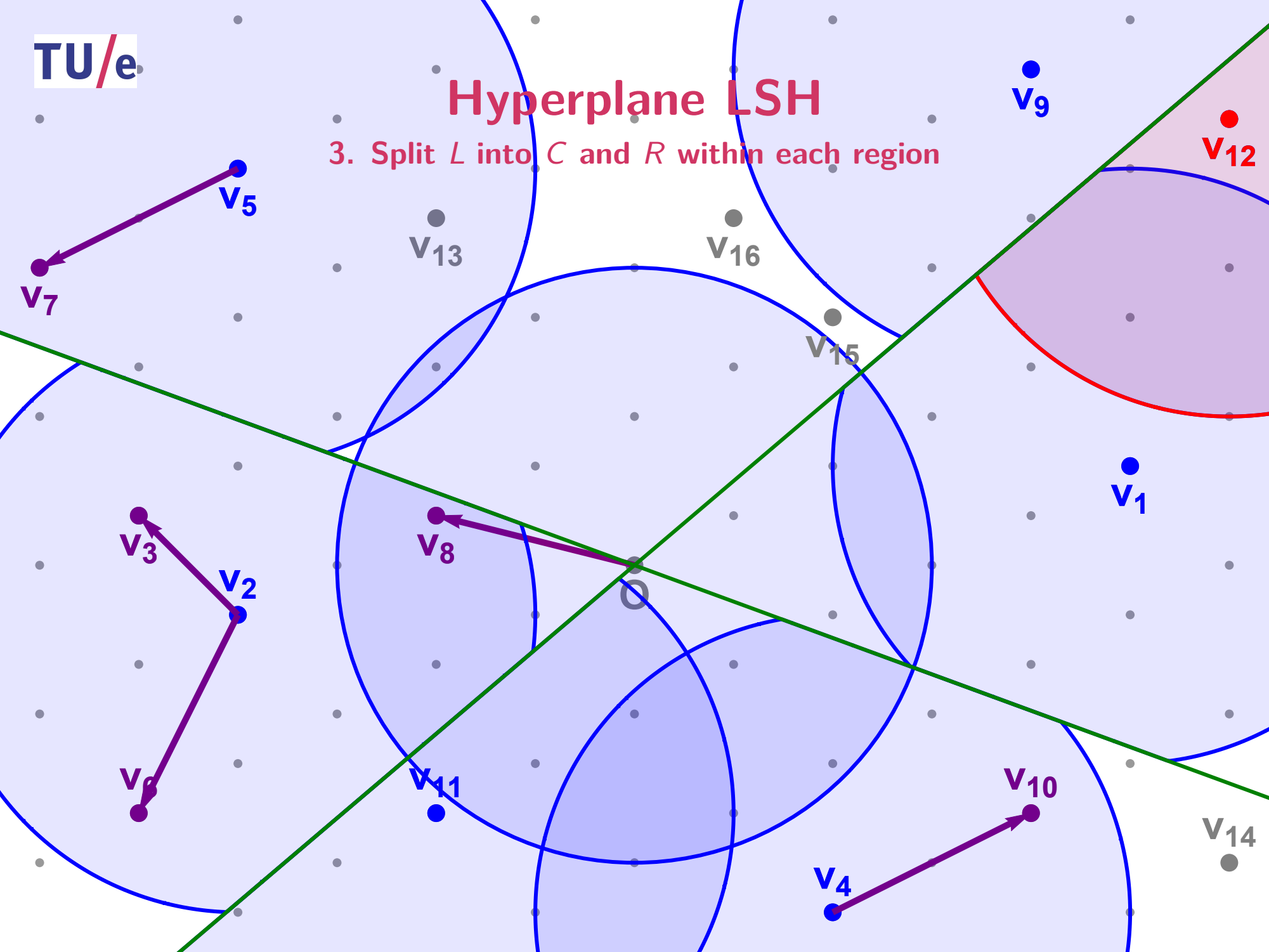

TU/e
Hyperplane LSH
3. Split L into C and R within each region
v_5
v_7
v_{13}
v_{16}
v_9
v_{12}
v_{15}
v_1
v_3
v_2
v_8
O
v_6
v_{11}
v_{10}
v_{14}
v_4

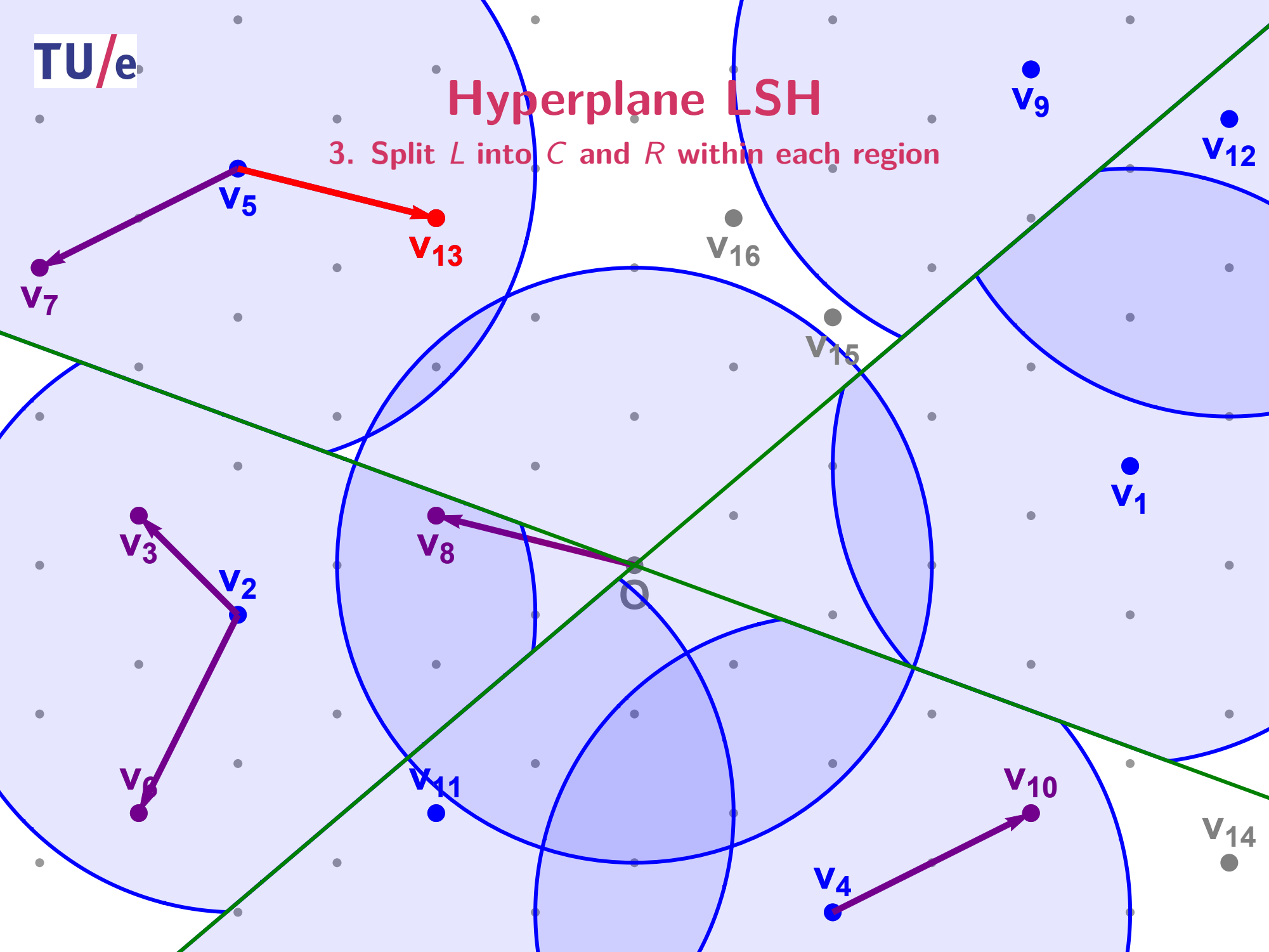

TU/e
Hyperplane LSH
3. Split L into C and R within each region
v_9
v_{12}
v_5
v_{13}
v_{16}
v_7
v_{15}
v_1
v_3
v_8
v_2
o
v_{11}
v_{10}
v_{14}
v_4

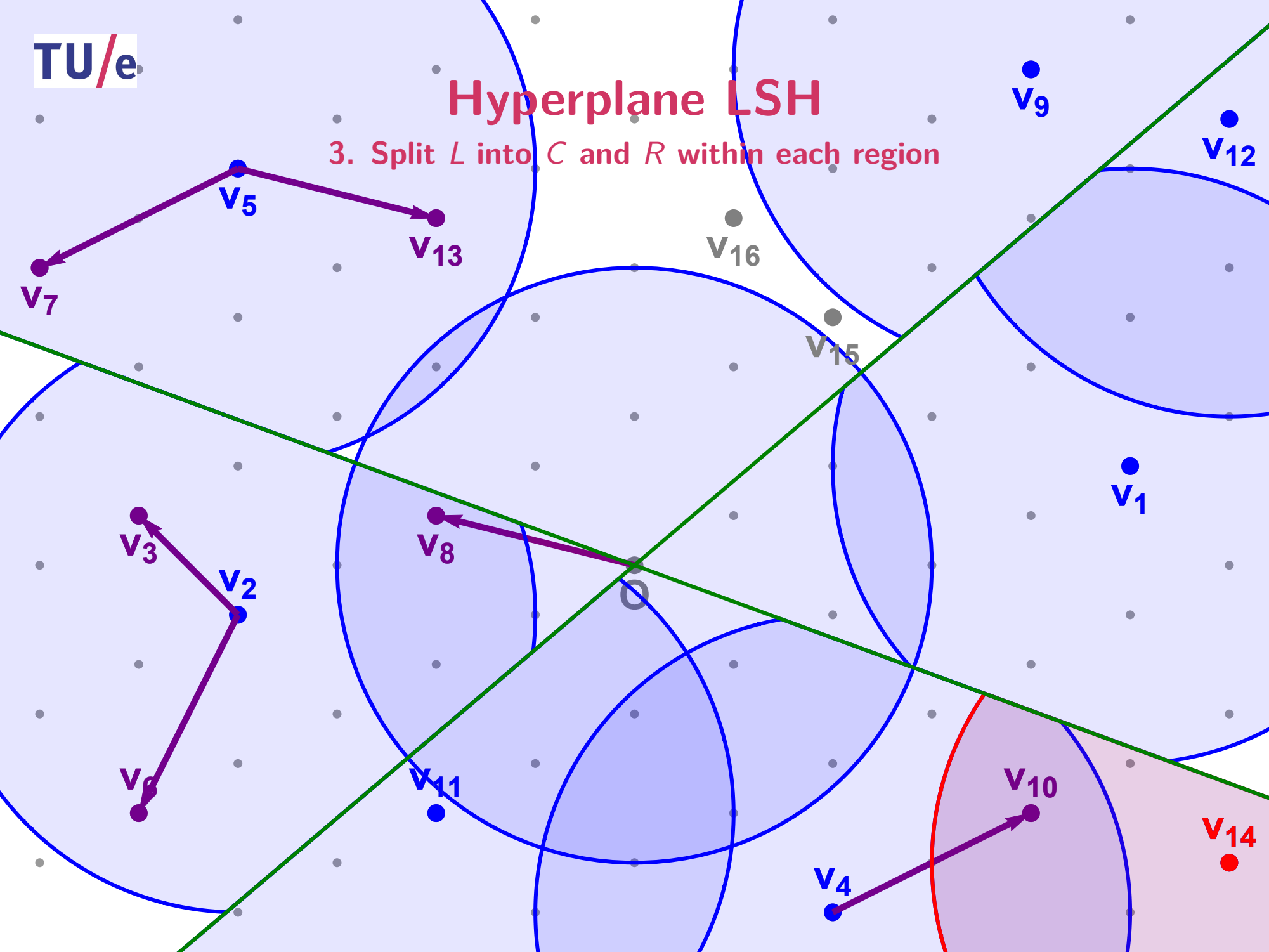
TU/e

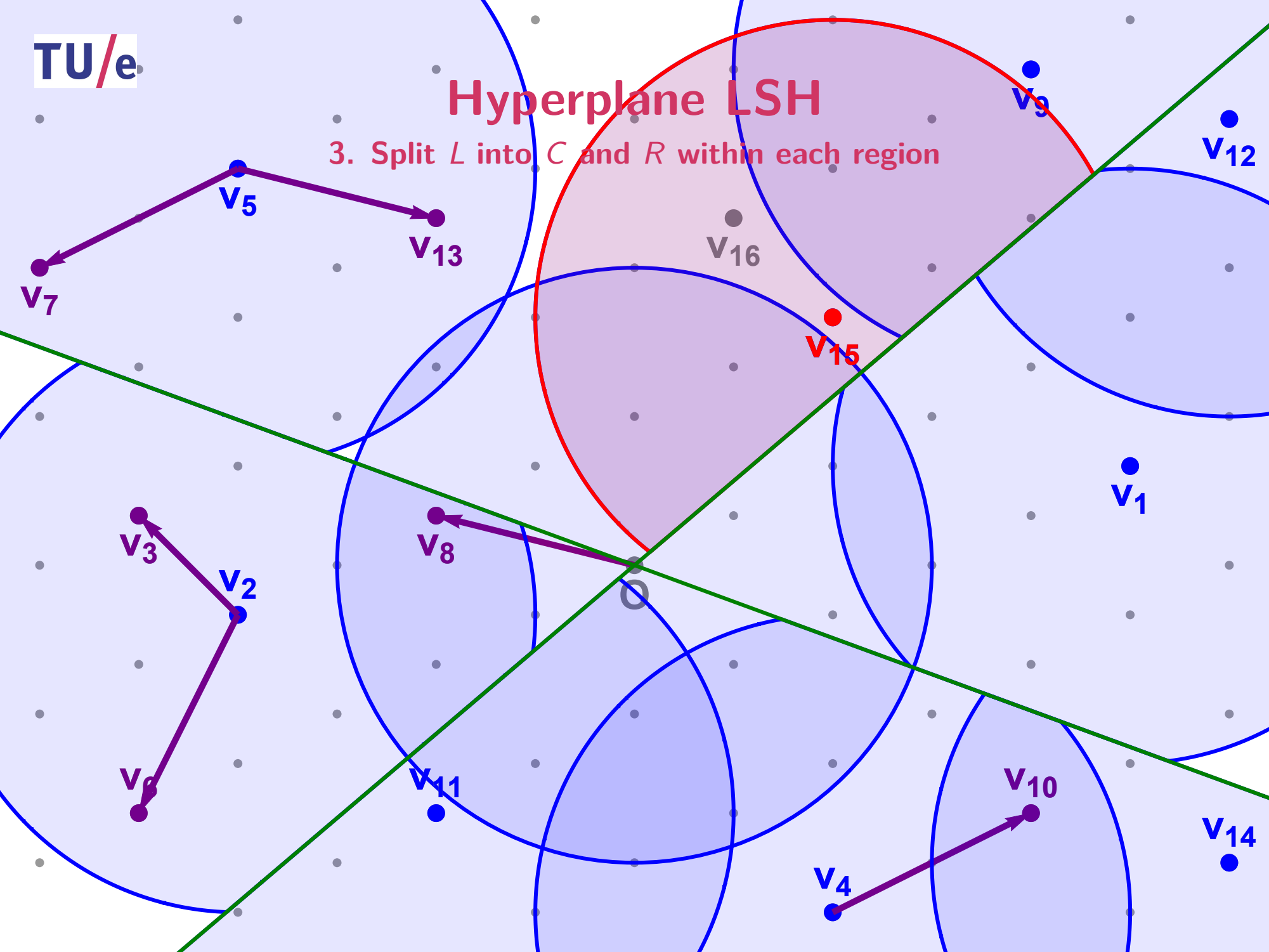
Hyperplane LSH

3. Split L into C and R within each region



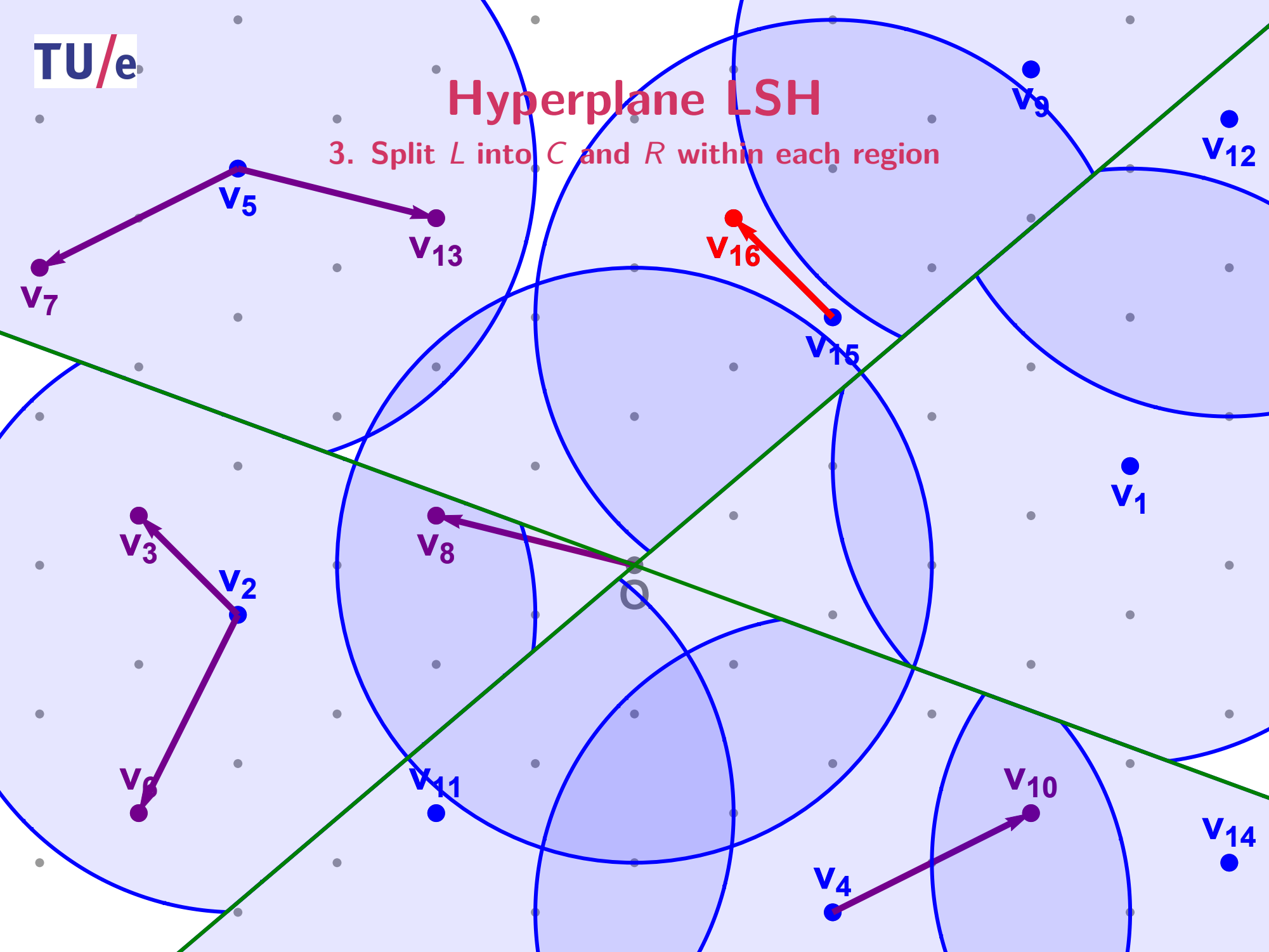

TU/e
Hyperplane LSH
3. Split L into C and R within each region
v_5
v_{13}
v_7
v_{16}
v_9
v_{12}
v_{15}
v_1
v_3
v_2
v_8
v_{11}
v_{10}
v_4
v_{14}

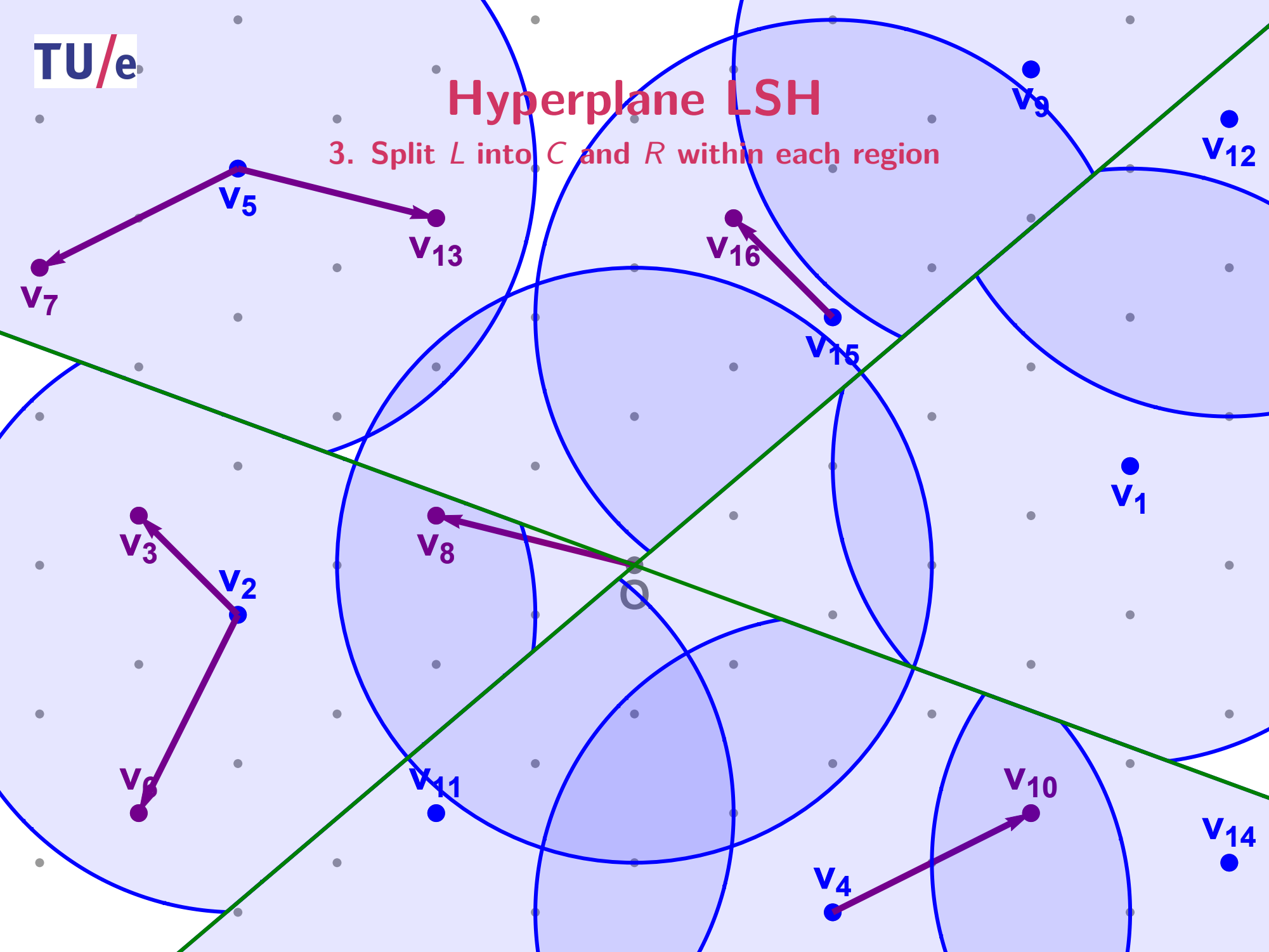

TU/e
Hyperplane LSH
3. Split L into C and R within each region
v_9
v_{12}
v_5
v_{13}
v_{16}
v_7
v_{15}
v_1
v_3
v_8
v_2
v_{11}
v_{10}
v_{14}
v_4


TU/e
Hyperplane LSH
3. Split L into C and R within each region
v_5
v_{13}
v_7
v_{16}
v_9
v_{12}
v_{15}
v_1
v_3
v_2
v_8
v_6
v_{11}
v_{10}
v_{14}
v_4

Hyperplane LSH

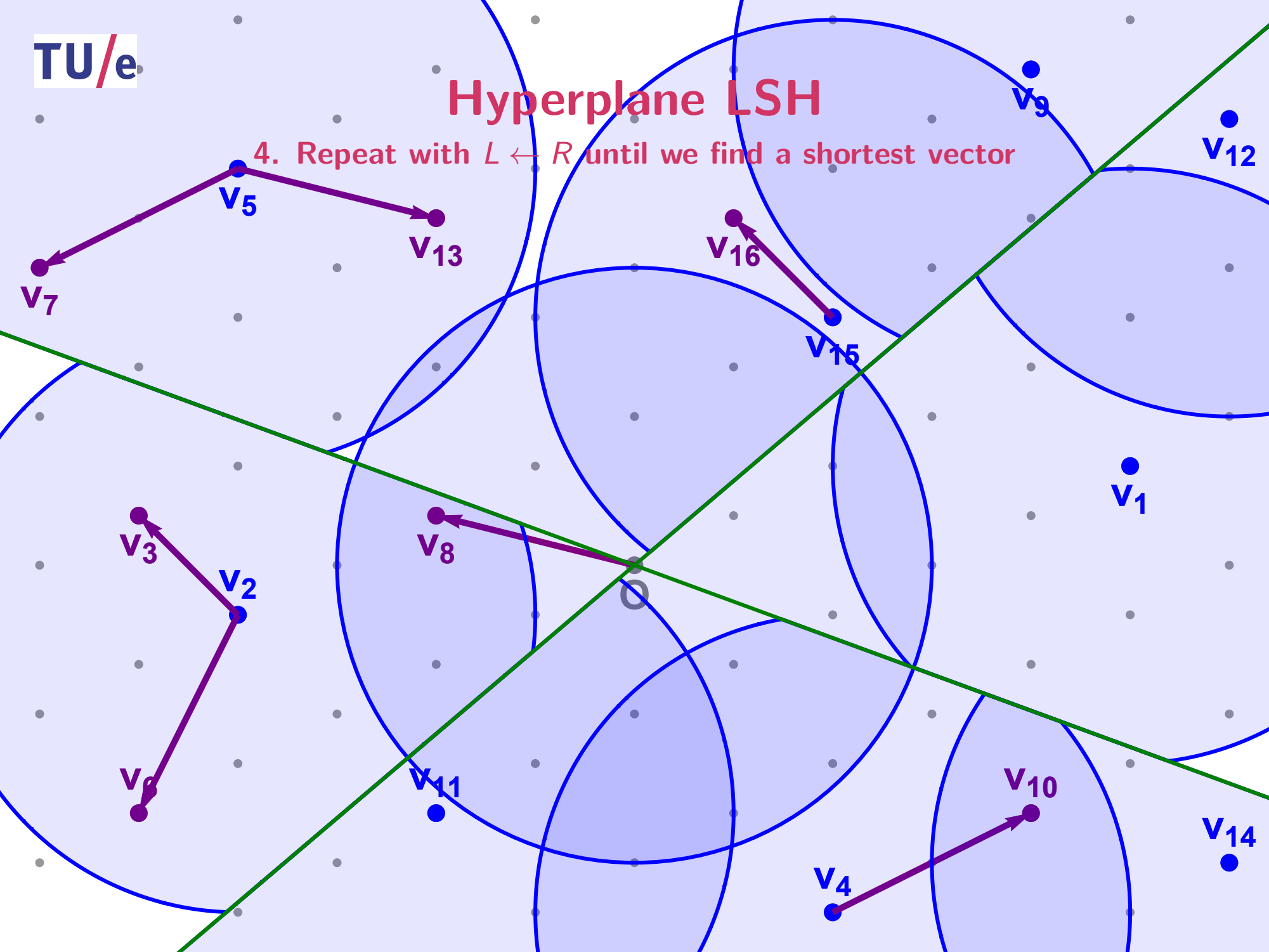
3. Split L into C and R within each region




TU/e
Hyperplane LSH
3. Split L into C and R within each region
v_5
v_{13}
v_7
v_{16}
v_{15}
v_9
v_{12}
v_1
v_3
v_2
v_8
v_6
v_{11}
v_{10}
v_4
v_{14}

Hyperplane LSH

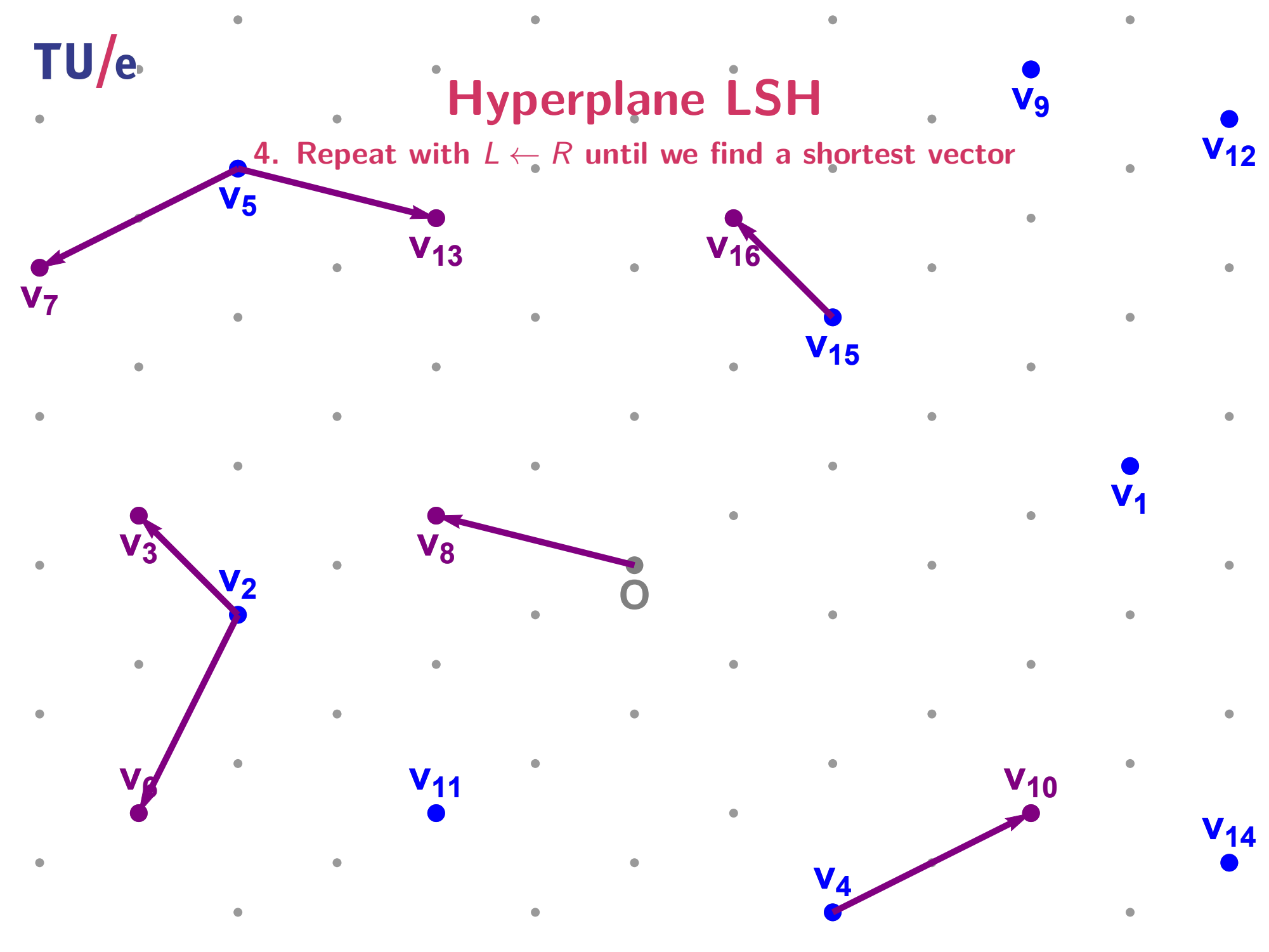
4. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

Hyperplane LSH

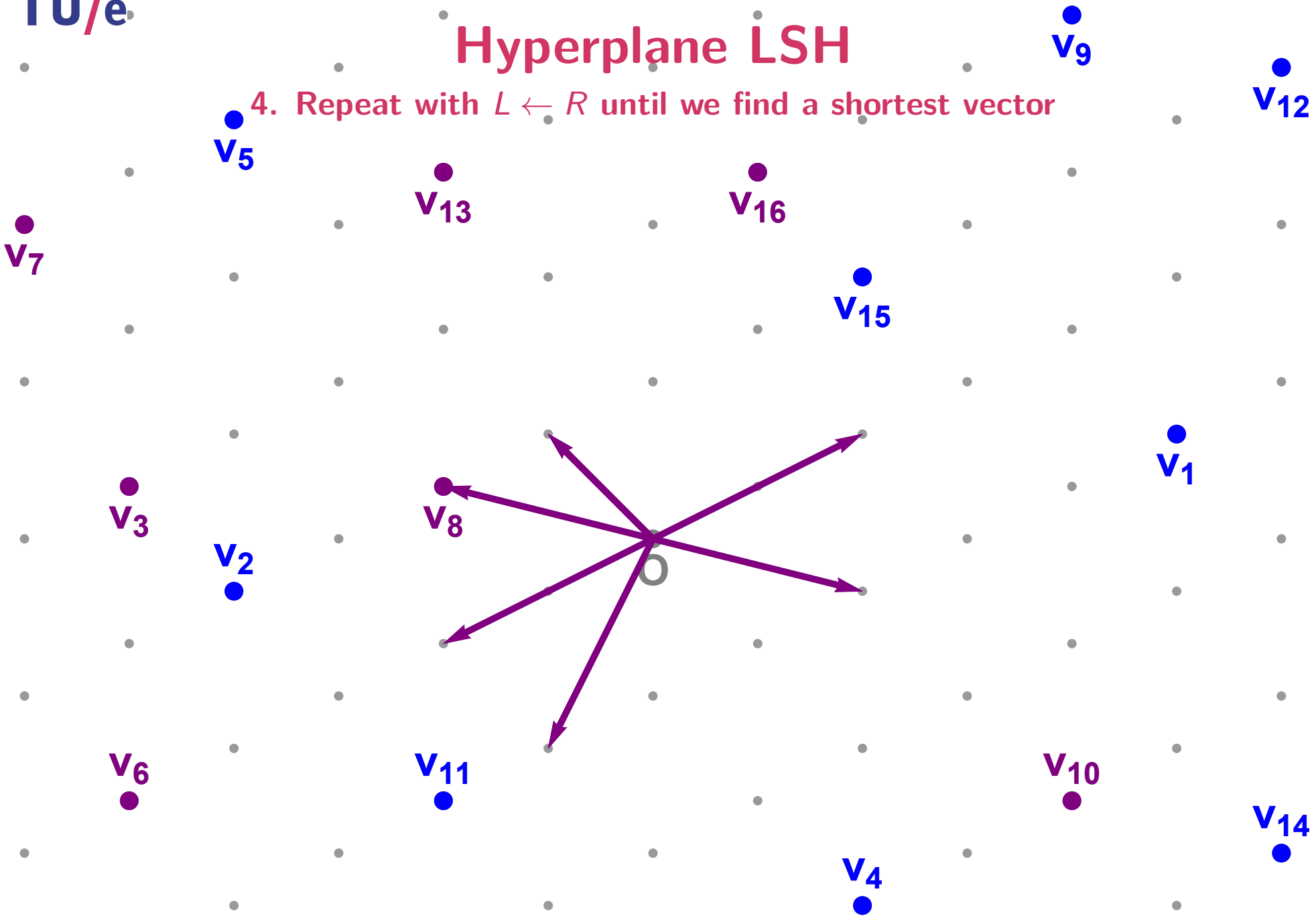
4. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

Hyperplane LSH

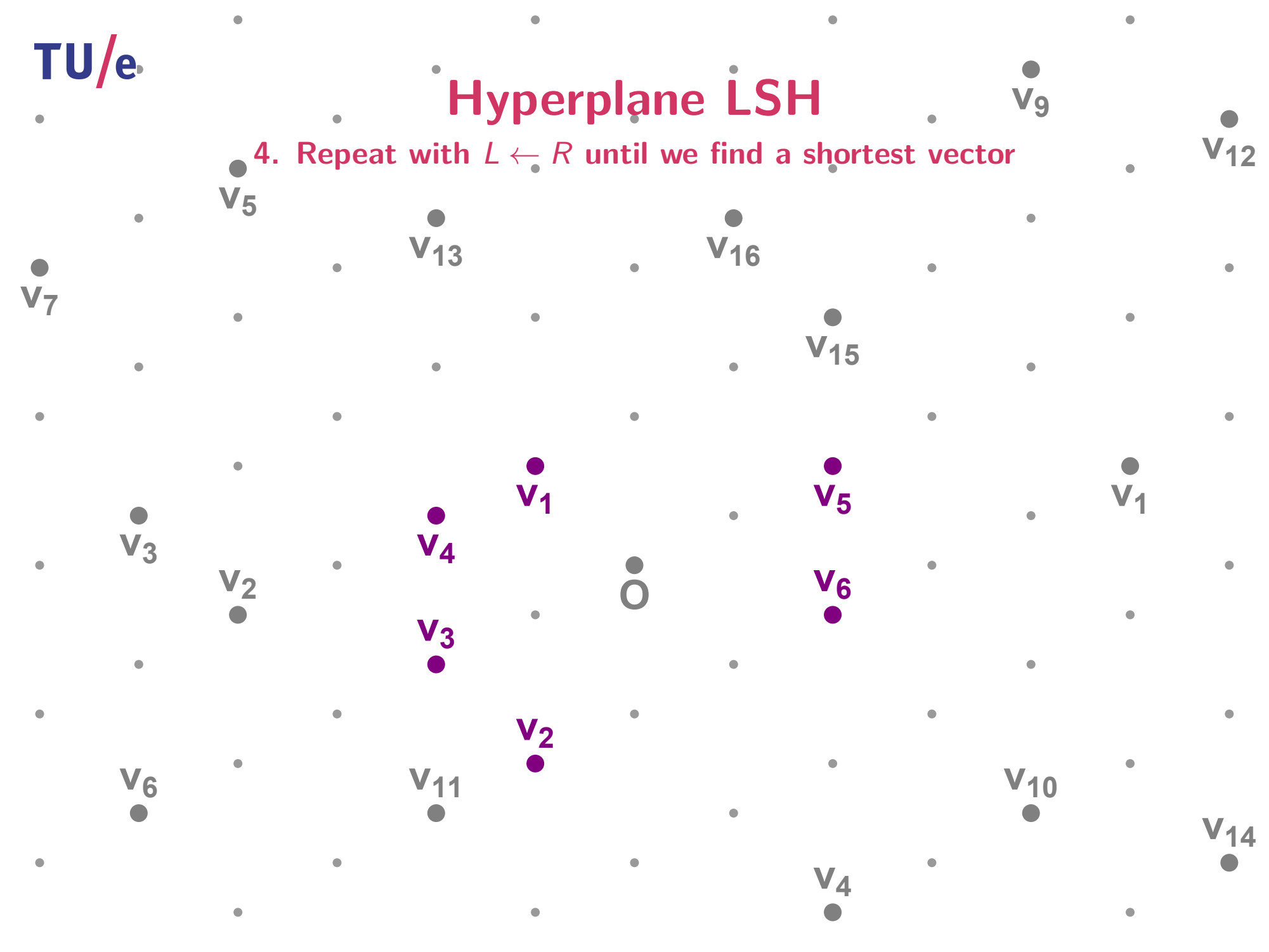
4. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

Hyperplane LSH

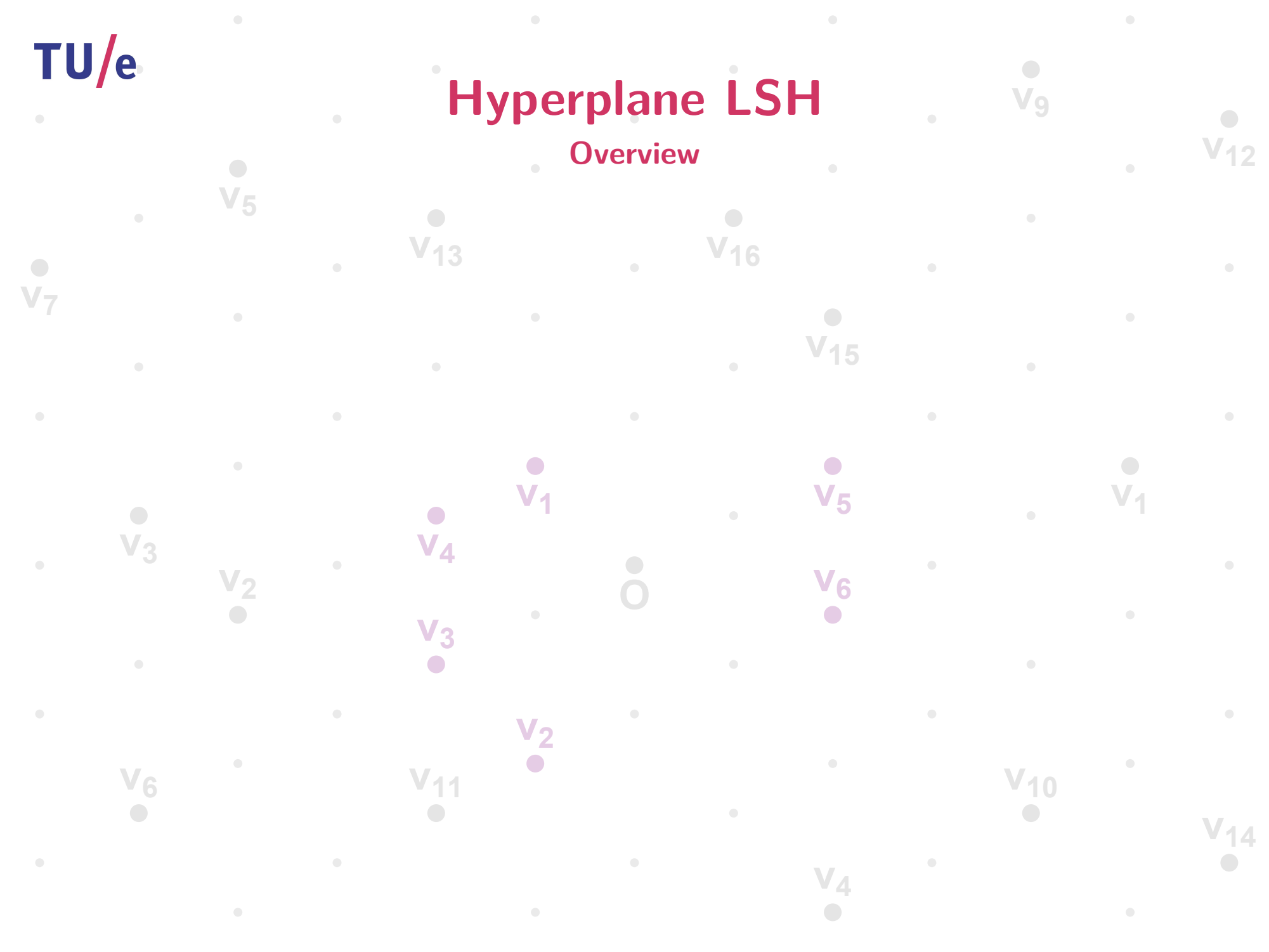
4. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

Hyperplane LSH

Overview



TU/e

Hyperplane LSH

Overview

- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent "hash tables"

TU/e

Hyperplane LSH

Overview

- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent "hash tables"
- Space complexity: $2^{0.337n+o(n)}$
  - Number of vectors: $2^{0.208n+o(n)}$
  - Number of hash tables: $2^{0.129n+o(n)}$
  - Each hash table contains all vectors

TU/e

Hyperplane LSH

Overview

- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent "hash tables"
- Space complexity: $2^{0.337n+o(n)}$
  - Number of vectors: $2^{0.208n+o(n)}$
  - Number of hash tables: $2^{0.129n+o(n)}$
  - Each hash table contains all vectors
- Time complexity: $2^{0.337n+o(n)}$
  - Cost of computing hashes: $2^{0.129n+o(n)}$
  - Candidate nearest vectors: $2^{0.129n+o(n)}$
  - Repeat this for each list vector: $2^{0.208n+o(n)}$

TU/e

Hyperplane LSH

Overview

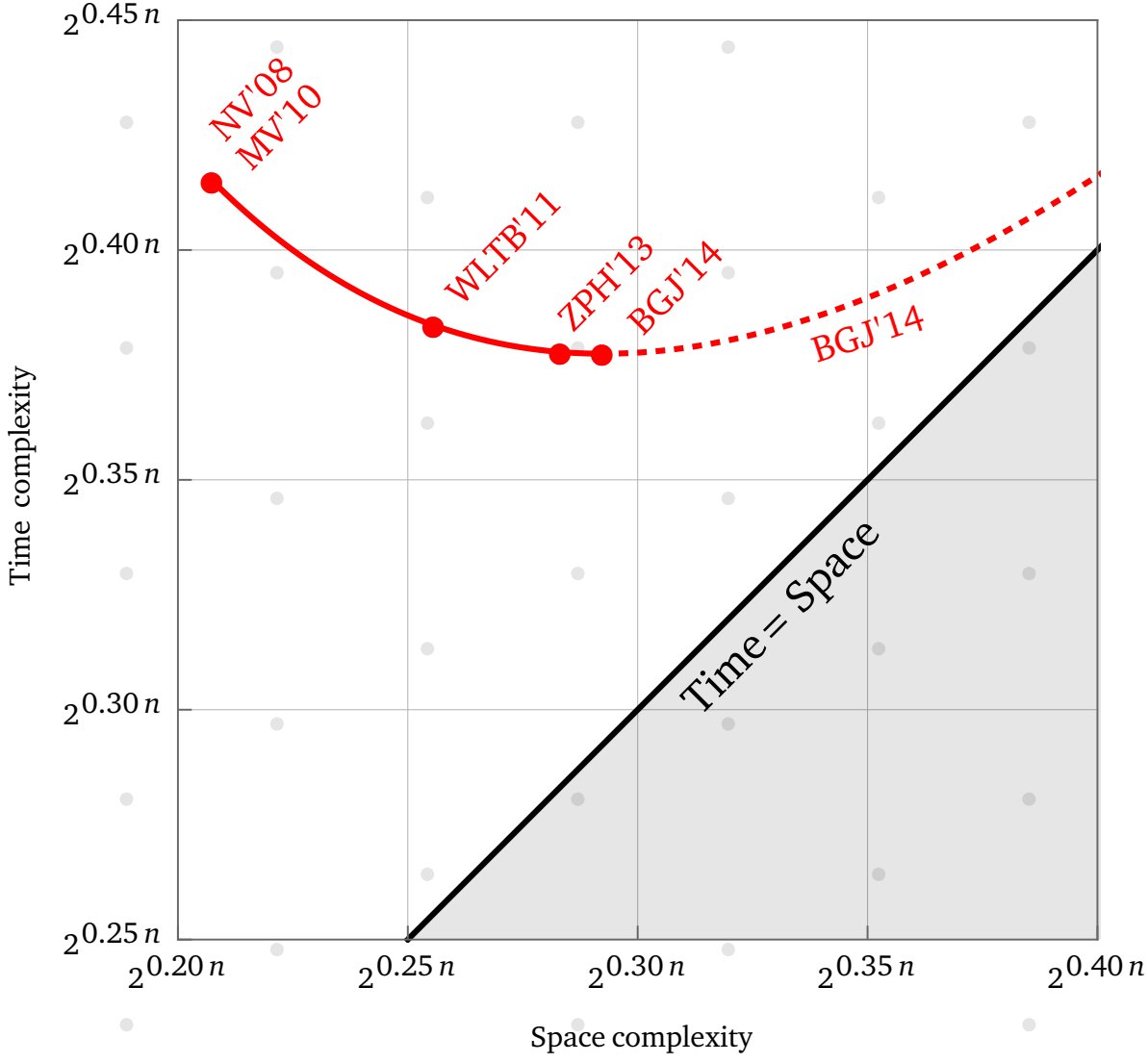
- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent "hash tables"
- Space complexity: $2^{0.337n+o(n)}$
  - Number of vectors: $2^{0.208n+o(n)}$
  - Number of hash tables: $2^{0.129n+o(n)}$
  - Each hash table contains all vectors
- Time complexity: $2^{0.337n+o(n)}$
  - Cost of computing hashes: $2^{0.129n+o(n)}$
  - Candidate nearest vectors: $2^{0.129n+o(n)}$
  - Repeat this for each list vector: $2^{0.208n+o(n)}$

Theorem

Sieving with hyperplane LSH heuristically solves SVP in time and space $2^{0.337n+o(n)}$.

Hyperplane LSH

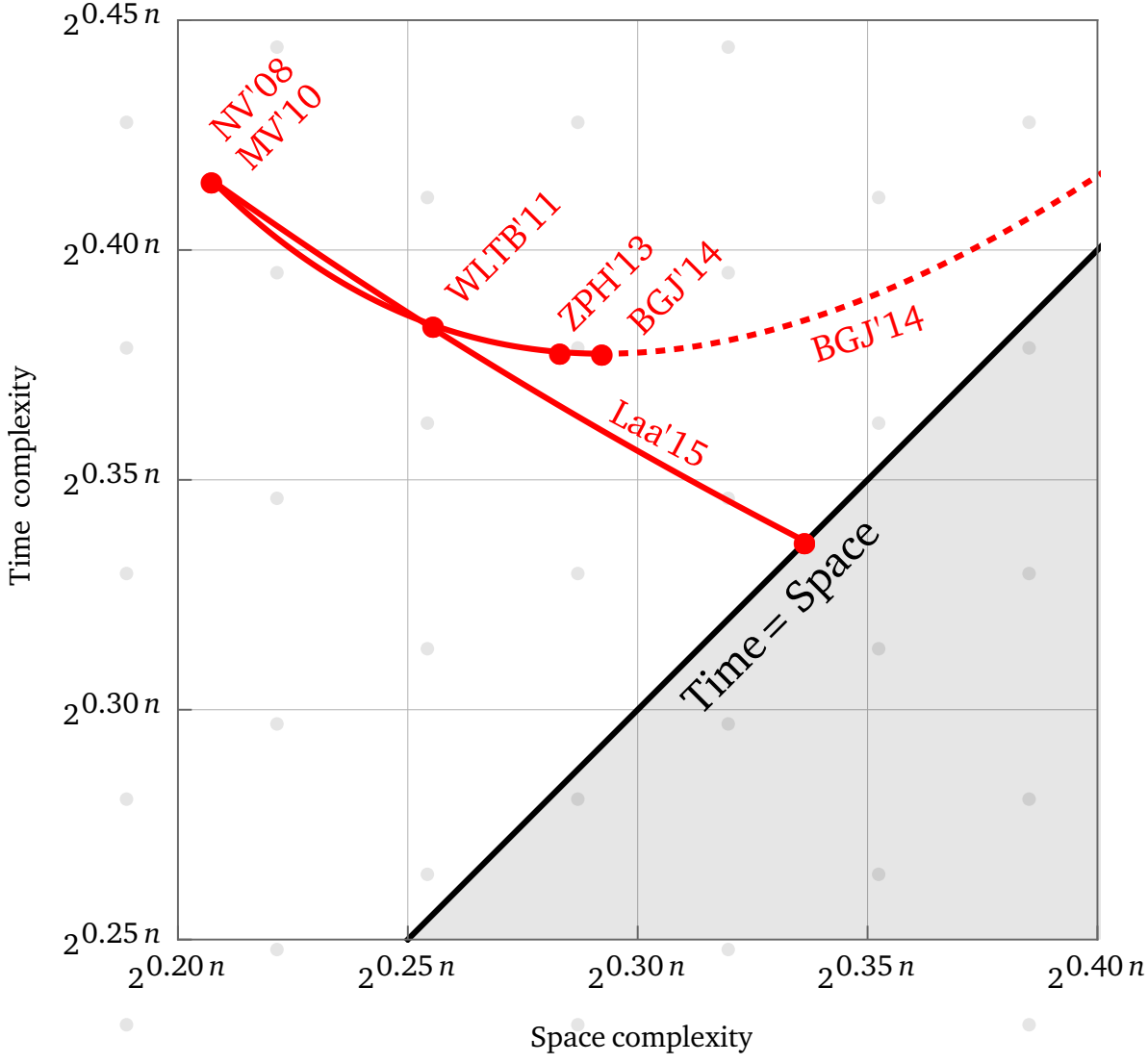
Space/time trade-off



TU/e

Hyperplane LSH

Space/time trade-off



TU/e

NNS methods

Overview

- [Laa15]: Hyperplane LSH

TU/e

NNS methods

Overview

- [Laa15]: Hyperplane LSH
  - Asymptotically suboptimal
  - Efficient in practice

TU/e

NNS methods

Overview

- [Laa15]: Hyperplane LSH
  - Asymptotically suboptimal
  - Efficient in practice
- [LdW15]: Spherical LSH

TU/e

NNS methods

Overview

- [Laa15]: Hyperplane LSH
  - Asymptotically suboptimal
  - Efficient in practice
- [LdW15]: Spherical LSH
  - Asymptotically optimal
  - Not very efficient in practice

TU/e

NNS methods

Overview

- [Laa15]: Hyperplane LSH
  - Asymptotically suboptimal
  - Efficient in practice
- [LdW15]: Spherical LSH
  - Asymptotically optimal
  - Not very efficient in practice
- [BL15]: Cross-polytope LSH

TU/e

NNS methods

Overview

- [Laa15]: Hyperplane LSH
  - Asymptotically suboptimal
  - Efficient in practice
- [LdW15]: Spherical LSH
  - Asymptotically optimal
  - Not very efficient in practice
- [BL15]: Cross-polytope LSH
  - Asymptotically optimal
  - Efficient in practice

TU/e

NNS methods

Overview

- [Laa15]: Hyperplane LSH
  - Asymptotically suboptimal
  - Efficient in practice
- [LdW15]: Spherical LSH
  - Asymptotically optimal
  - Not very efficient in practice
- [BL15]: Cross-polytope LSH
  - Asymptotically optimal
  - Efficient in practice
- [BDGL15]: Spherical filtering

TU/e

NNS methods

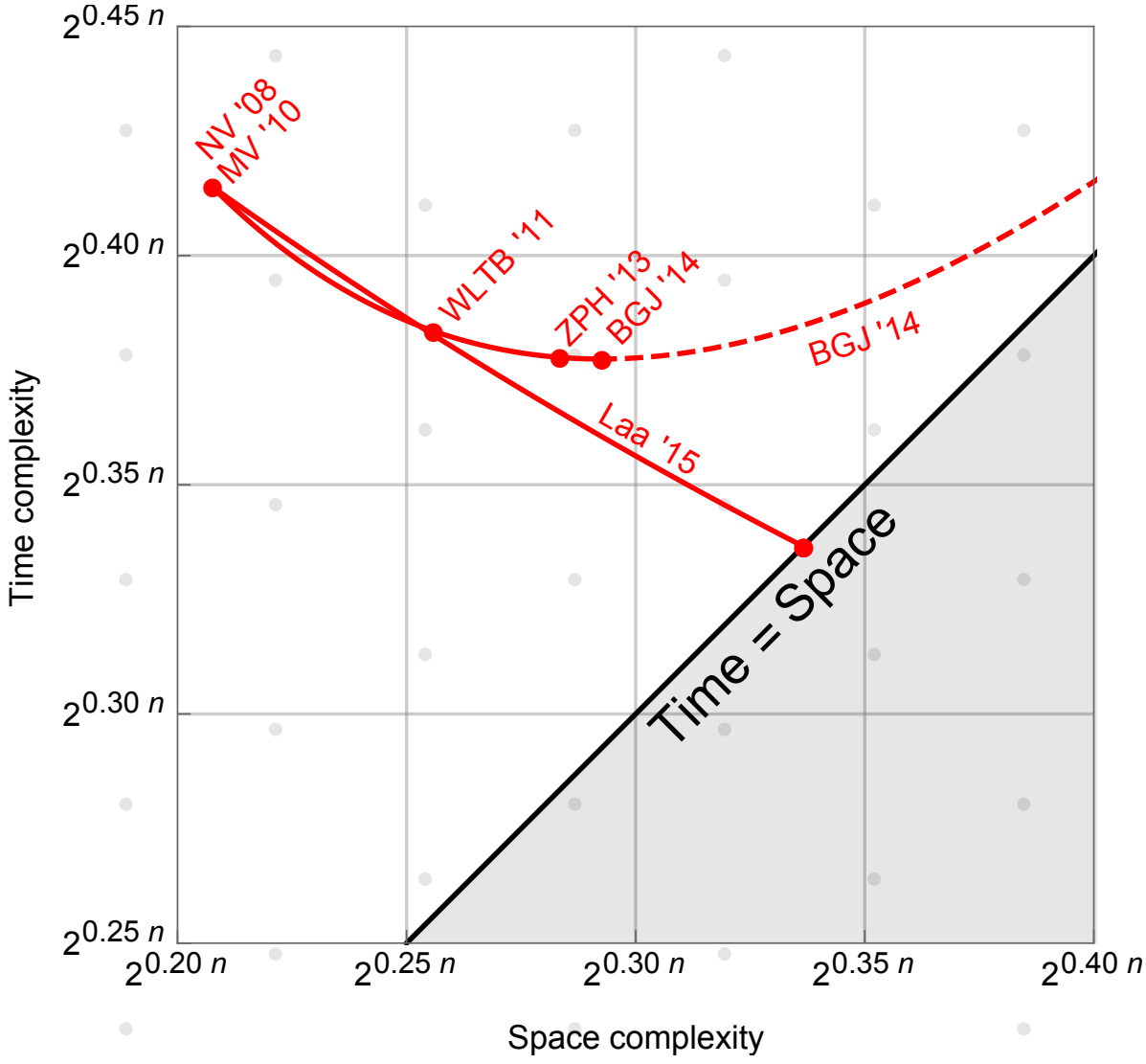
Overview

- [Laa15]: Hyperplane LSH
  - Asymptotically suboptimal
  - Efficient in practice
- [LdW15]: Spherical LSH
  - Asymptotically optimal
  - Not very efficient in practice
- [BL15]: Cross-polytope LSH
  - Asymptotically optimal
  - Efficient in practice
- [BDGL15]: Spherical filtering
  - Asymptotically super-optimal
  - Efficient in practice

TU/e

NNS methods

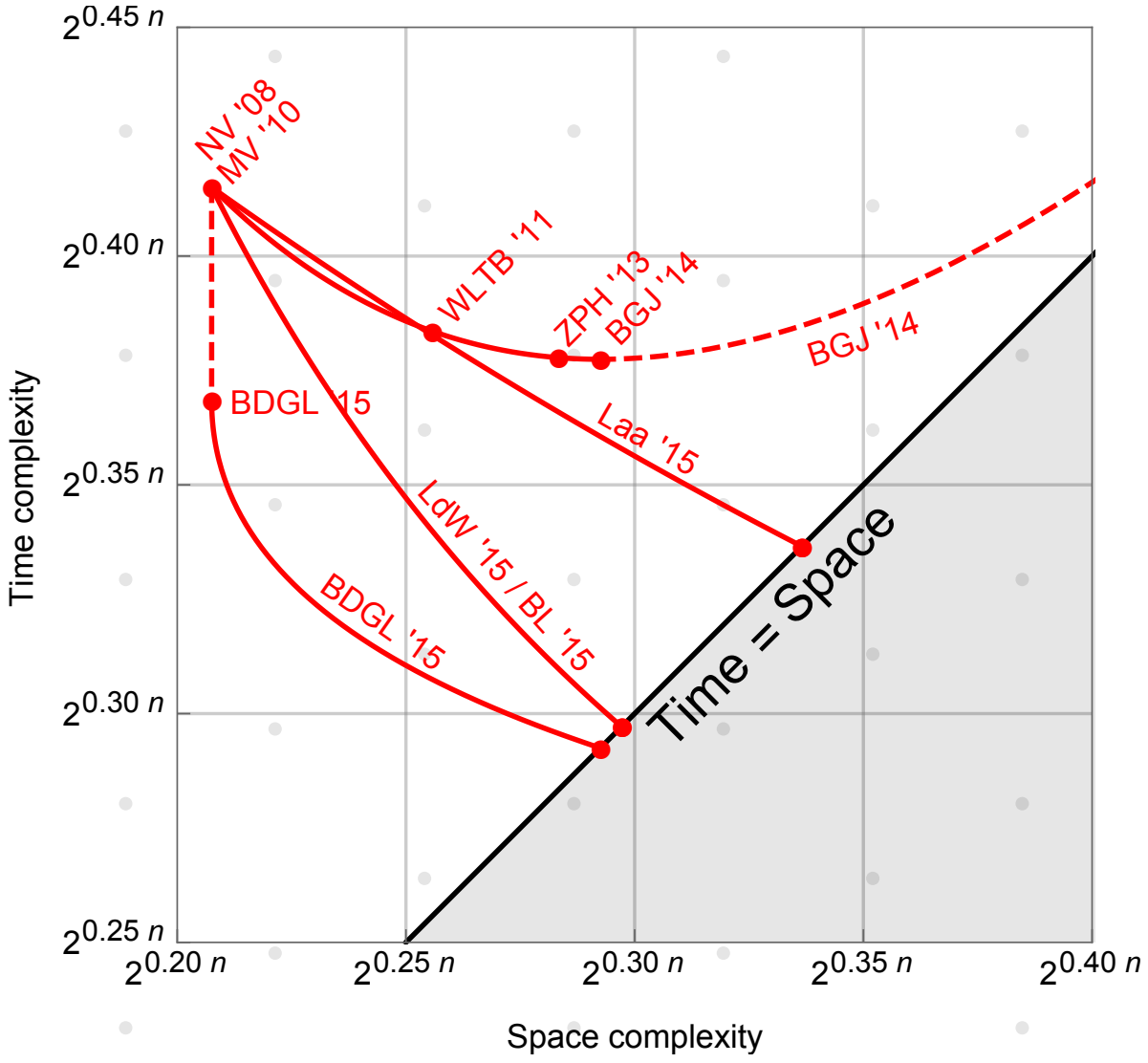
Space/time trade-off



TU/e

NNS methods

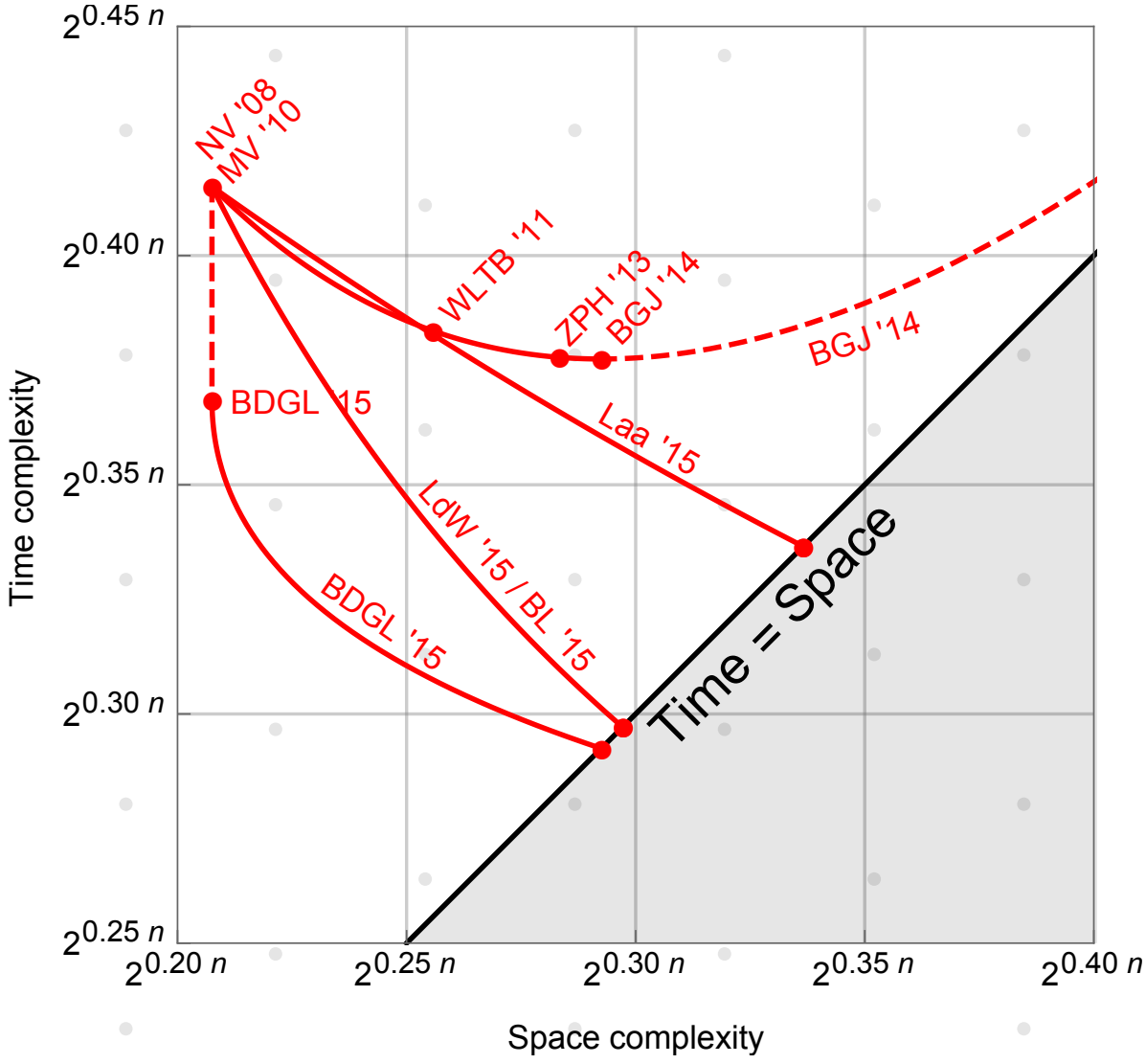
Space/time trade-off



TU/e

Quantum searching

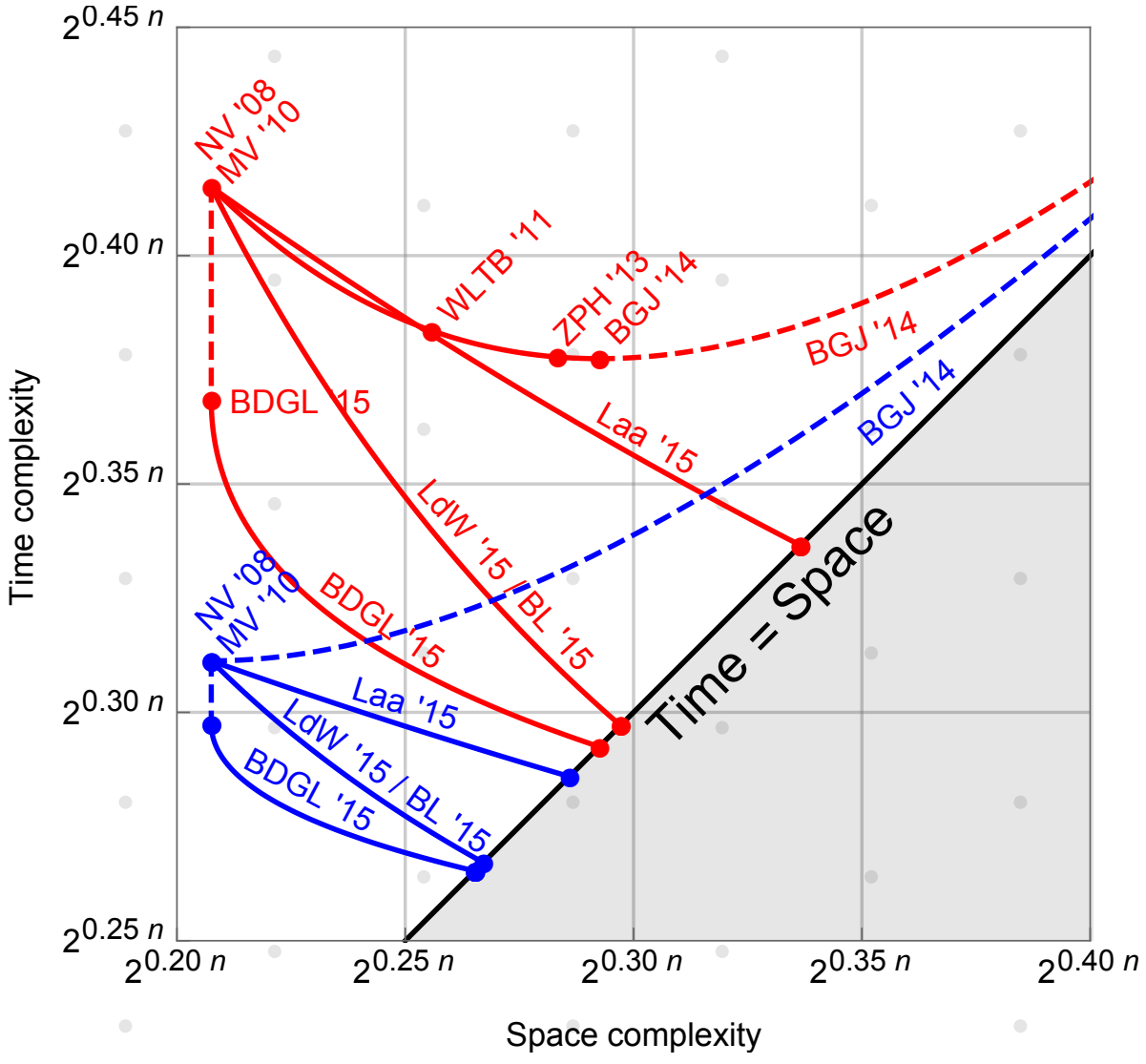
Space/time trade-off



TU/e

Quantum searching

Space/time trade-off



TU/e

Questions?

[vdP'12]