Technische Universiteit Eindhoven University of Technology

Algorithms for hard lattice problems and lattice-based cryptography

Thijs Laarhoven

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

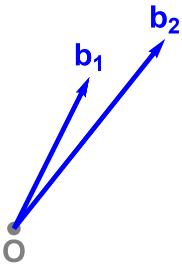
IBM Research, Rüschlikon, Switzerland
(November 26, 2015)

Lattices

What is a lattice?

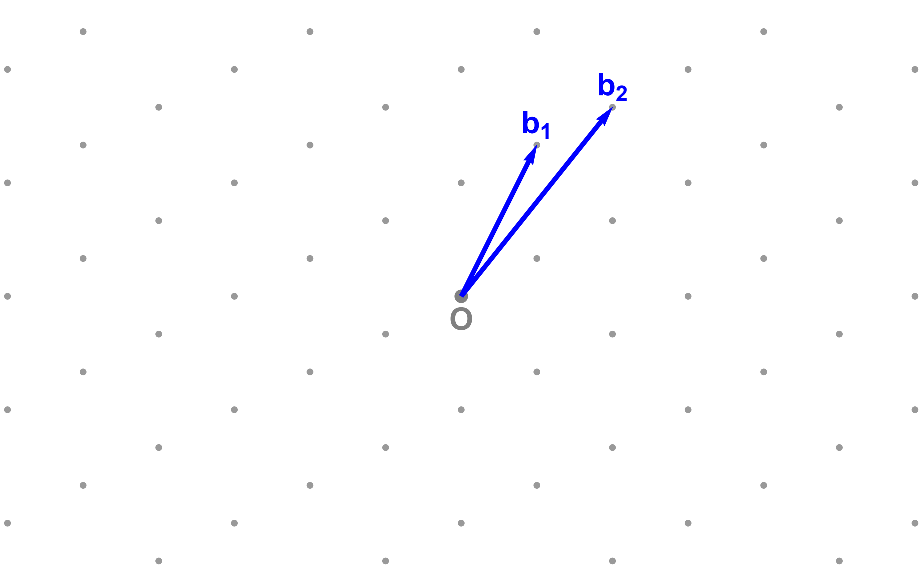
Lattices

What is a lattice?



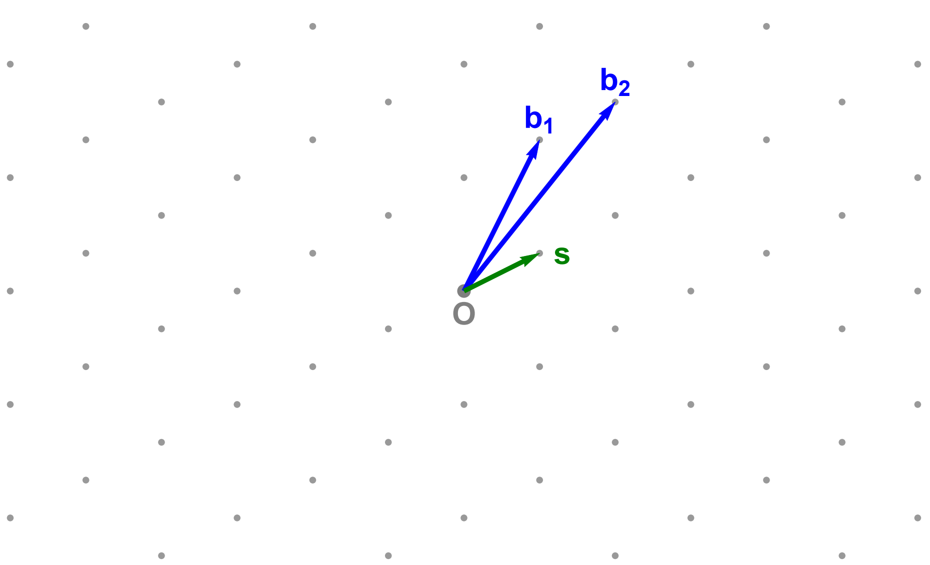
Lattices

What is a lattice?



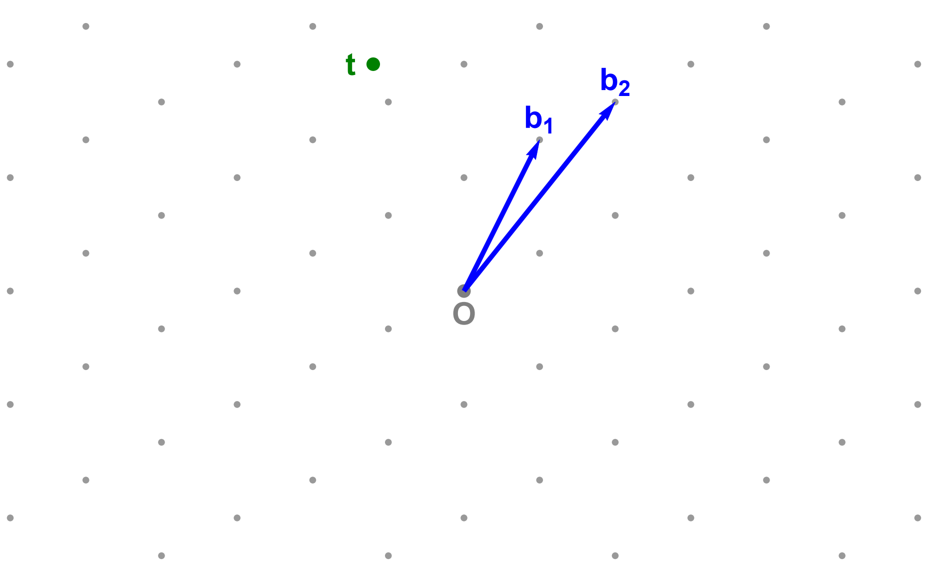
Lattices

Shortest Vector Problem (SVP)



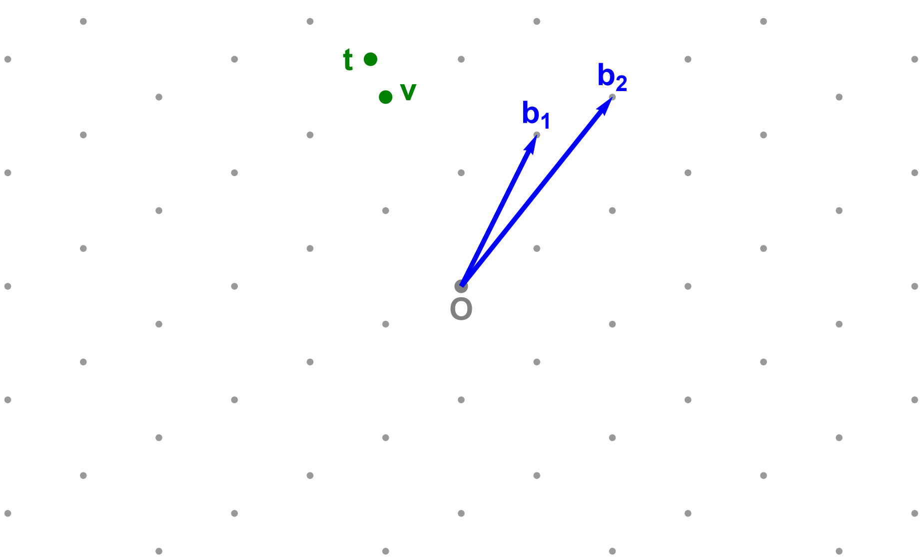
Lattices

Closest Vector Problem (CVP)



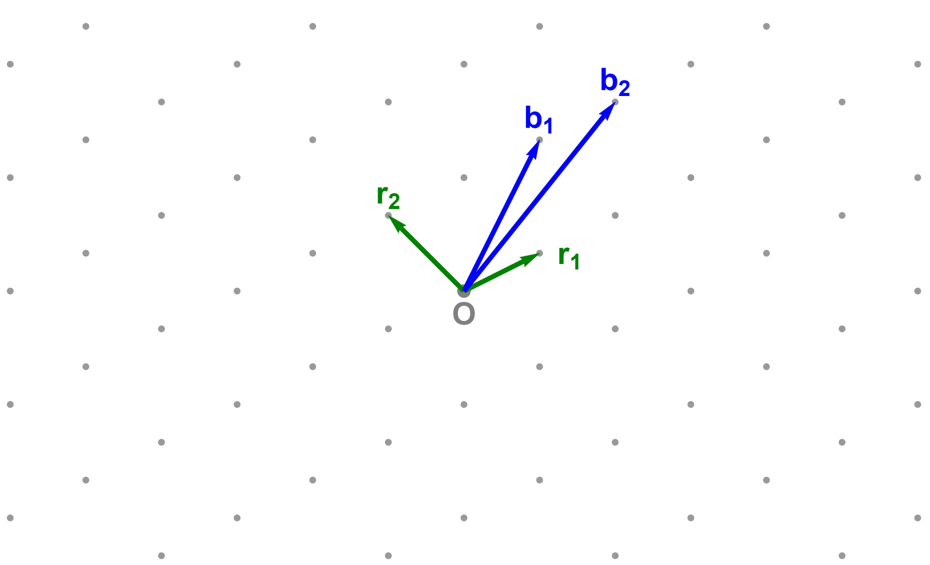
Lattices

Closest Vector Problem (CVP)



Lattices

Lattice basis reduction (e.g. LLL, BKZ)



Lattices
Applications

- "Constructive cryptography": Lattice-based cryptosystems
  - Based on hard lattice problems (SVP, CVP, LWE, SIS)
  - Worst-case to average-case reductions [Ajt96]
  - Candidate for "post-quantum cryptography"
  - NTRU cryptosystem [HPS98, ..., HPSSWZ15]
  - Fully homomorphic encryption [Gen09, ..., CM15]
  - Multilinear maps(?) [GGH13, ..., Cor15]

Lattices

Applications

- "Constructive cryptography": Lattice-based cryptosystems
  - Based on hard lattice problems (SVP, CVP, LWE, SIS)
  - Worst-case to average-case reductions [Ajt96]
  - Candidate for "post-quantum cryptography"
  - NTRU cryptosystem [HPS98, . . . , HPSSWZ15]
  - Fully homomorphic encryption [Gen09, . . . , CM15]
  - Multilinear maps(?) [GGH13, . . . , Cor15]
- "Destructive cryptography": Lattice cryptanalysis
  - Attack knapsack-based cryptosystems [Sha82, LO85, . . .]
  - Attack RSA with Coppersmith's method [Cop97, . . .]
  - Attack lattice-based cryptosystems [Ngu99, JJ00, . . .]

Lattices

Applications

- "Constructive cryptography": Lattice-based cryptosystems
  - Based on hard lattice problems (SVP, CVP, LWE, SIS)
  - Worst-case to average-case reductions [Ajt96]
  - Candidate for "post-quantum cryptography"
  - NTRU cryptosystem [HPS98, . . . , HPSSWZ15]
  - Fully homomorphic encryption [Gen09, . . . , CM15]
  - Multilinear maps(?) [GGH13, . . . , Cor15]
- "Destructive cryptography": Lattice cryptanalysis
  - Attack knapsack-based cryptosystems [Sha82, LO85, . . .]
  - Attack RSA with Coppersmith's method [Cop97, . . .]
  - Attack lattice-based cryptosystems [Ngu99, JJ00, . . .]

How does lattice-based cryptography work?

GGH cryptosystem

Overview [GGH97]

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

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

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

Encrypt $\boldsymbol{m}$:

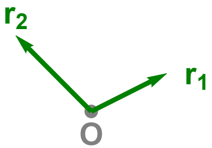
$\boldsymbol{v} = \boldsymbol{m}B$

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

Decrypt $\boldsymbol{c}$:

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

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

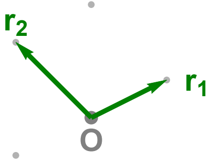


GGH cryptosystem

Public key

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

Public key

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 $\boldsymbol{m}$:

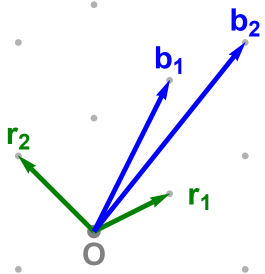
$\boldsymbol{v} = \boldsymbol{m}B$

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

Decrypt $\boldsymbol{c}$:

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

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



GGH cryptosystem

Encryption

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

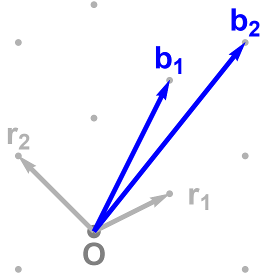
Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

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

Encryption

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 $\boldsymbol{m}$:

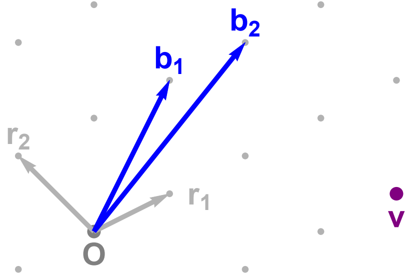
$\boldsymbol{v} = \boldsymbol{m}B$

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

Decrypt $\boldsymbol{c}$:

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

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



GGH cryptosystem

Encryption

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

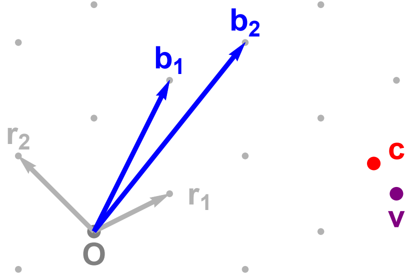
Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

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

Decryption with good basis

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

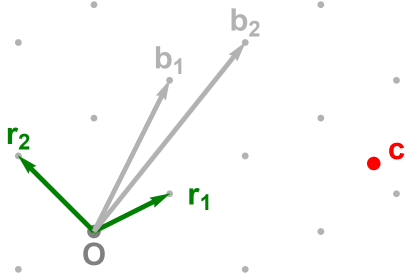
Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

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

Decryption with good basis

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

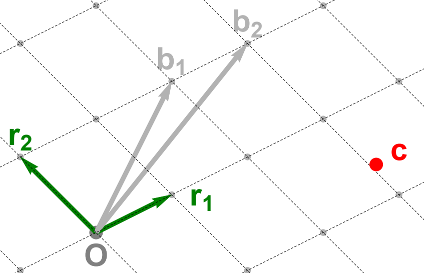
Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

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

Decryption with good basis

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

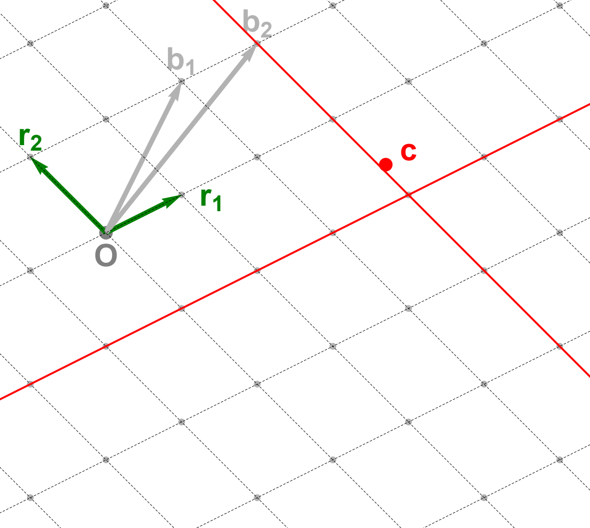
Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

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

Decryption with good basis

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 $\boldsymbol{m}$:

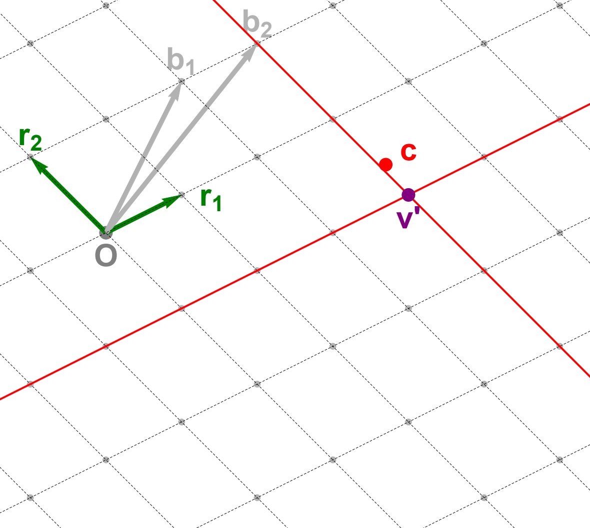
$\boldsymbol{v} = \boldsymbol{m}B$

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

Decrypt $\boldsymbol{c}$:

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

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



TU/e

GGH cryptosystem

Decryption with bad basis

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 $\boldsymbol{m}$:

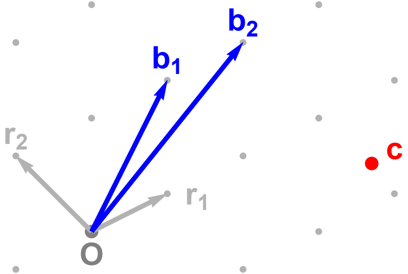
$\boldsymbol{v} = \boldsymbol{m}B$

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

Decrypt $\boldsymbol{c}$:

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

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



GGH cryptosystem

Decryption with bad basis

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 $\boldsymbol{m}$:

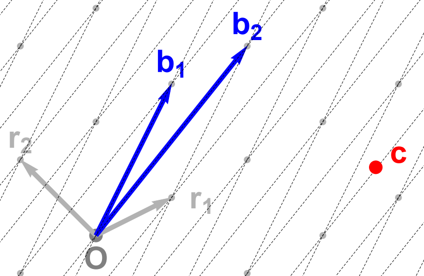
$\boldsymbol{v} = \boldsymbol{m}B$

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

Decrypt $\boldsymbol{c}$:

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

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



GGH cryptosystem

Decryption with bad basis

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

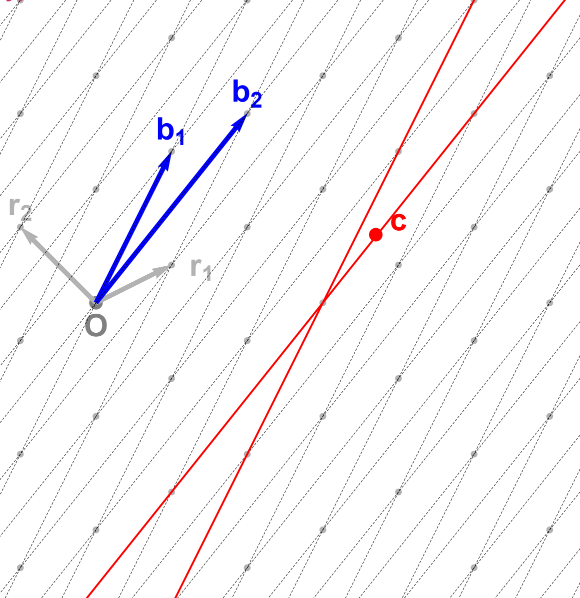
Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

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

Decryption with bad basis

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 $\boldsymbol{m}$:

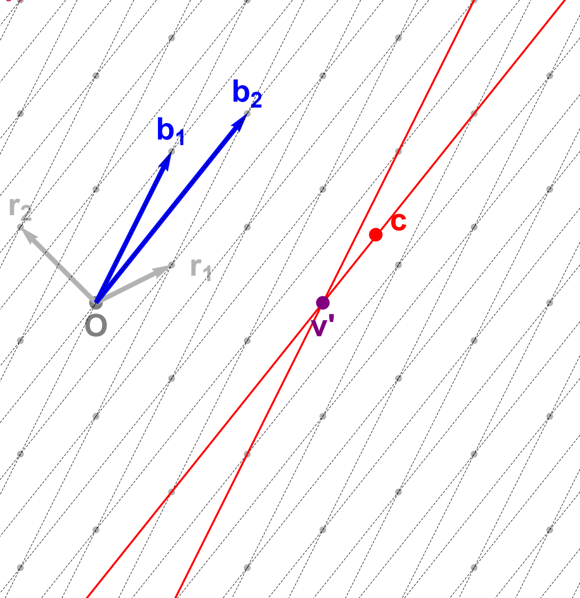
$\boldsymbol{v} = \boldsymbol{m}B$

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

Decrypt $\boldsymbol{c}$:

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

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



GGH cryptosystem

Overview

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

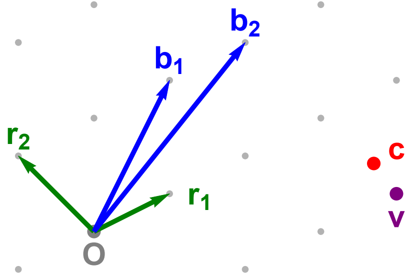
Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

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 signatures
Overview

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

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

Sign $\boldsymbol{m}$:

$\boldsymbol{c} = H(\boldsymbol{m})$

$\boldsymbol{s} = \lfloor \boldsymbol{c} R^{-1} \rceil R$

Verify $(\boldsymbol{m}, \boldsymbol{s})$:

$\boldsymbol{s}$ lies on the lattice

$\|\boldsymbol{s} - H(\boldsymbol{m})\|$ is small

GGH signatures

Private and public keys

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

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

Sign $\boldsymbol{m}$:

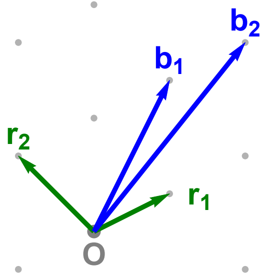
$\boldsymbol{c} = H(\boldsymbol{m})$

$\boldsymbol{s} = \lfloor \boldsymbol{c} R^{-1} \rceil R$

Verify $(\boldsymbol{m}, \boldsymbol{s})$:

$\boldsymbol{s}$ lies on the lattice

$\|\boldsymbol{s} - H(\boldsymbol{m})\|$ is small



GGH signatures

Signing messages

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

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

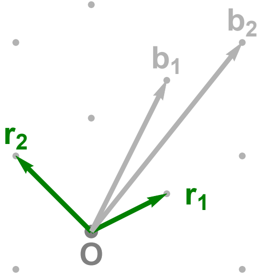
Sign $\boldsymbol{m}$:

$$\boldsymbol{c} = H(\boldsymbol{m})$$
$$\boldsymbol{s} = \lfloor \boldsymbol{c} R^{-1} \rceil R$$

Verify $(\boldsymbol{m}, \boldsymbol{s})$:

$\boldsymbol{s}$ lies on the lattice

$\|\boldsymbol{s} - H(\boldsymbol{m})\|$ is small



GGH signatures

Signing messages

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

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

Sign $\boldsymbol{m}$:

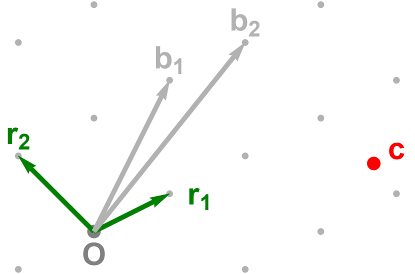
$\boldsymbol{c} = H(\boldsymbol{m})$

$\boldsymbol{s} = \lfloor \boldsymbol{c} R^{-1} \rceil R$

Verify $(\boldsymbol{m}, \boldsymbol{s})$:

$\boldsymbol{s}$ lies on the lattice

$\|\boldsymbol{s} - H(\boldsymbol{m})\|$ is small



GGH signatures

Signing messages

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

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

Sign $\boldsymbol{m}$:

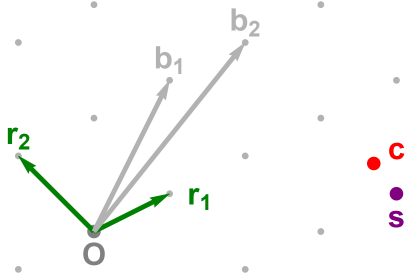
$\boldsymbol{c} = H(\boldsymbol{m})$

$\boldsymbol{s} = \lfloor \boldsymbol{c} R^{-1} \rceil R$

Verify $(\boldsymbol{m}, \boldsymbol{s})$:

$\boldsymbol{s}$ lies on the lattice

$\|\boldsymbol{s} - H(\boldsymbol{m})\|$ is small



GGH signatures

Verifying signatures

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

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

Sign $\boldsymbol{m}$:

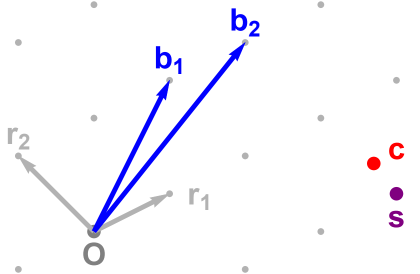
$\boldsymbol{c} = H(\boldsymbol{m})$

$\boldsymbol{s} = \lfloor \boldsymbol{c} R^{-1} \rceil R$

Verify $(\boldsymbol{m}, \boldsymbol{s})$:

$\boldsymbol{s}$ lies on the lattice

$\|\boldsymbol{s} - H(\boldsymbol{m})\|$ is small



GGH signatures

Overview

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

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

Sign $\boldsymbol{m}$:

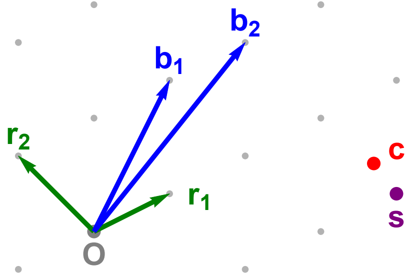
$\boldsymbol{c} = H(\boldsymbol{m})$

$\boldsymbol{s} = \lfloor \boldsymbol{c} R^{-1} \rceil R$

Verify $(\boldsymbol{m}, \boldsymbol{s})$:

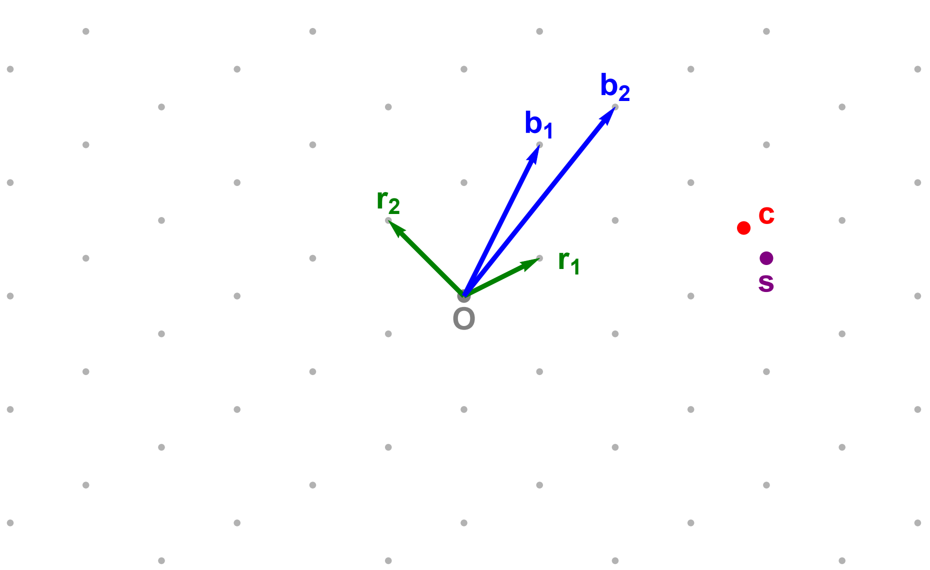
$\boldsymbol{s}$ lies on the lattice

$\|\boldsymbol{s} - H(\boldsymbol{m})\|$ is small



GGH signatures

Breaking the scheme [NR06]



GGH signatures

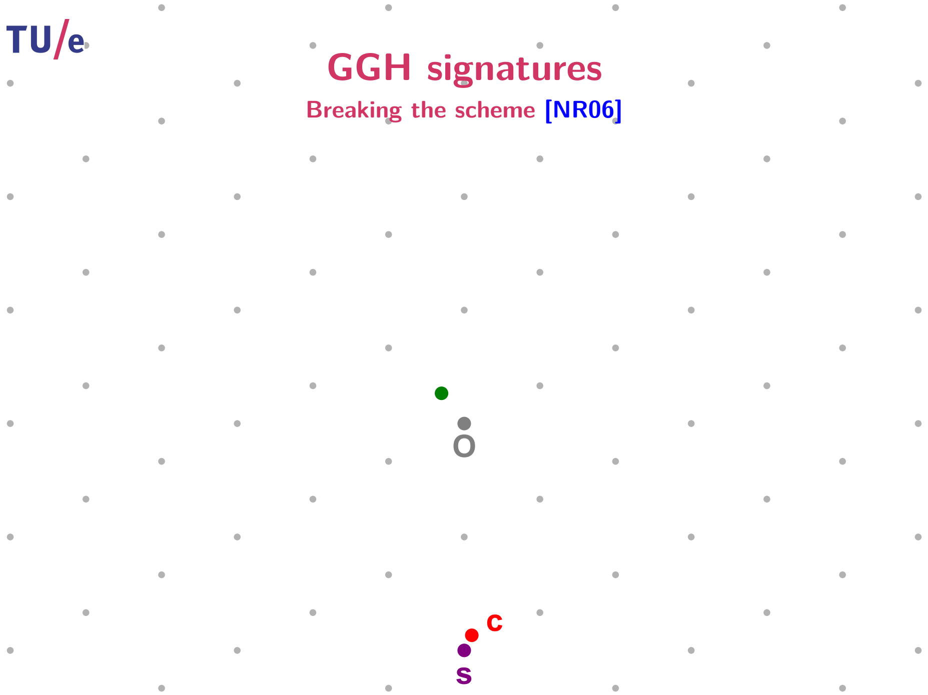
Breaking the scheme [NR06]

GGH signatures

Breaking the scheme [NR06]

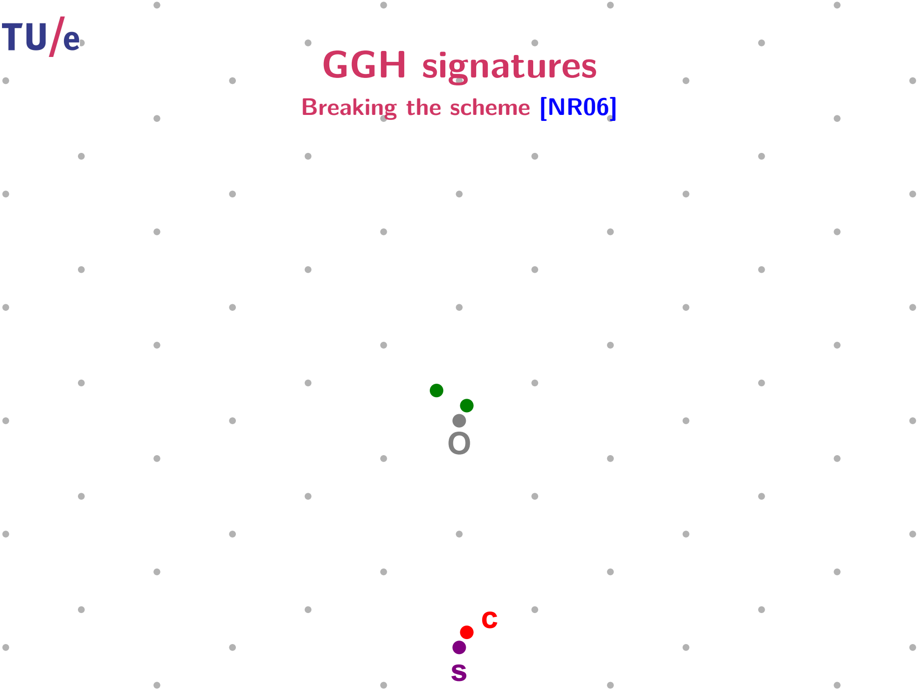
GGH signatures

Breaking the scheme [NR06]



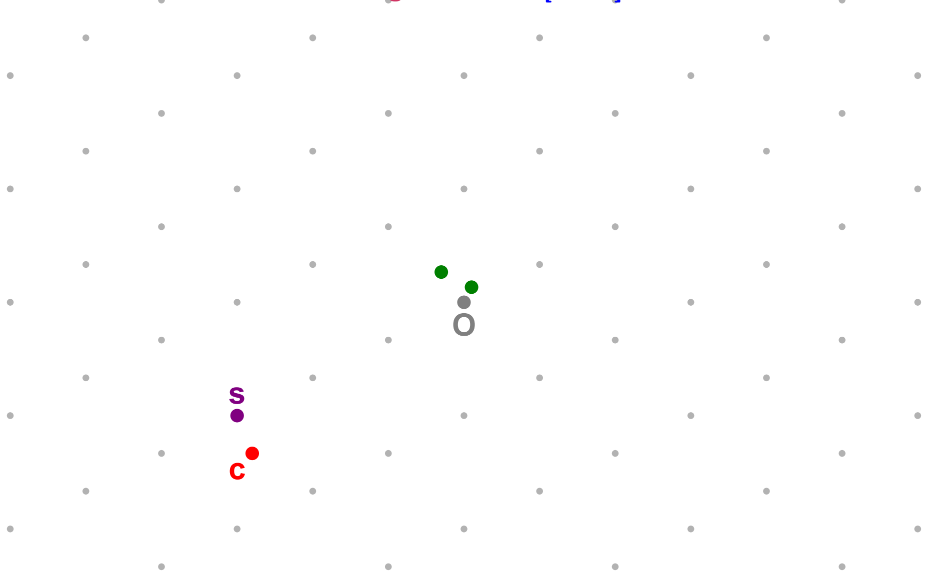
GGH signatures

Breaking the scheme [NR06]



GGH signatures

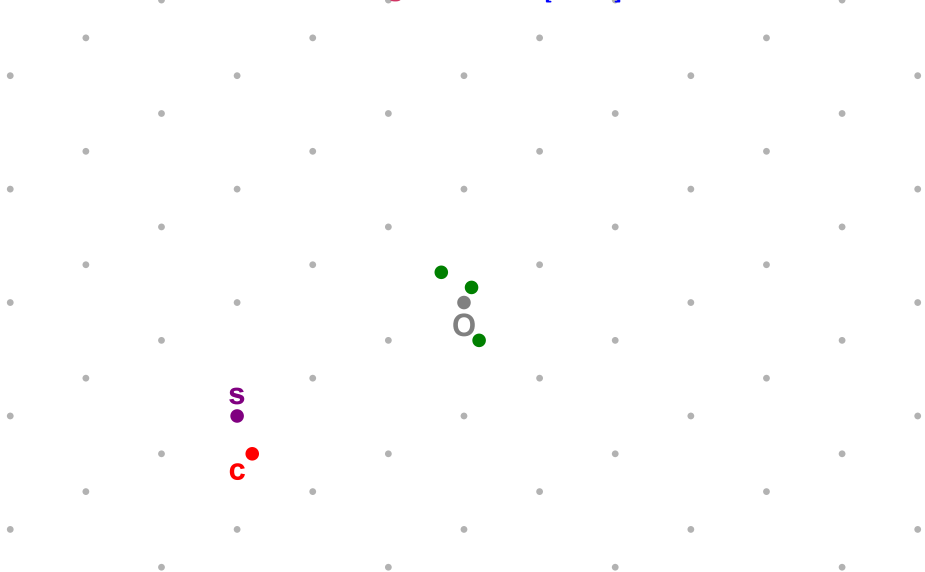
Breaking the scheme [NR06]



TU/e

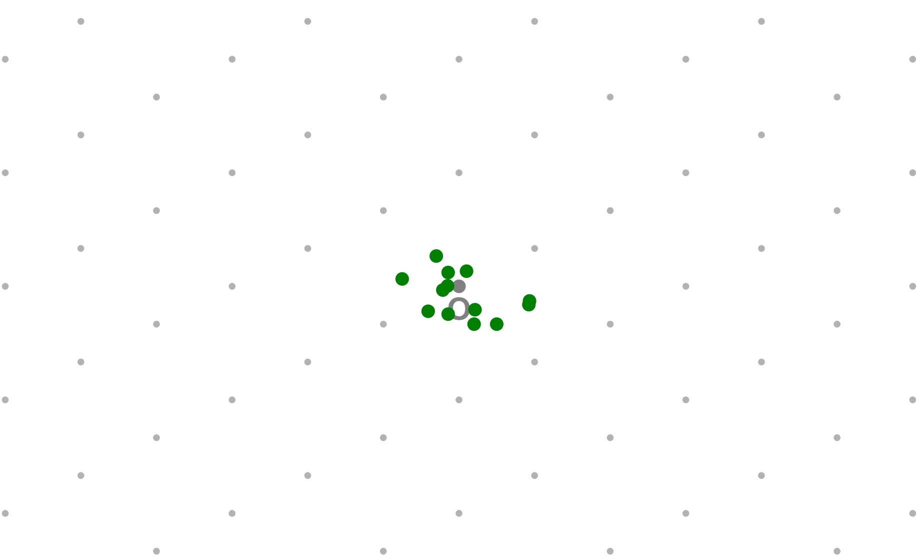
GGH signatures

Breaking the scheme [NR06]



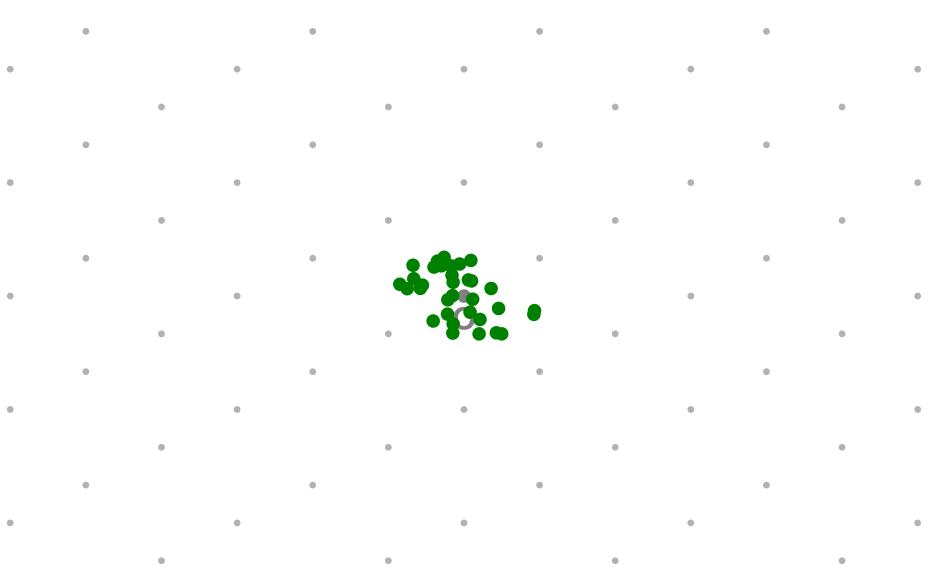
GGH signatures

Breaking the scheme [NR06]



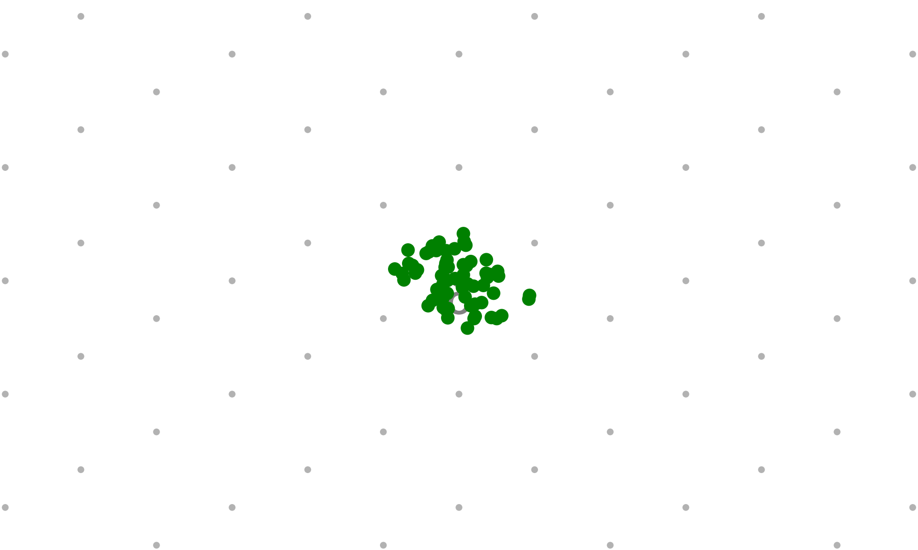
GGH signatures

Breaking the scheme [NR06]



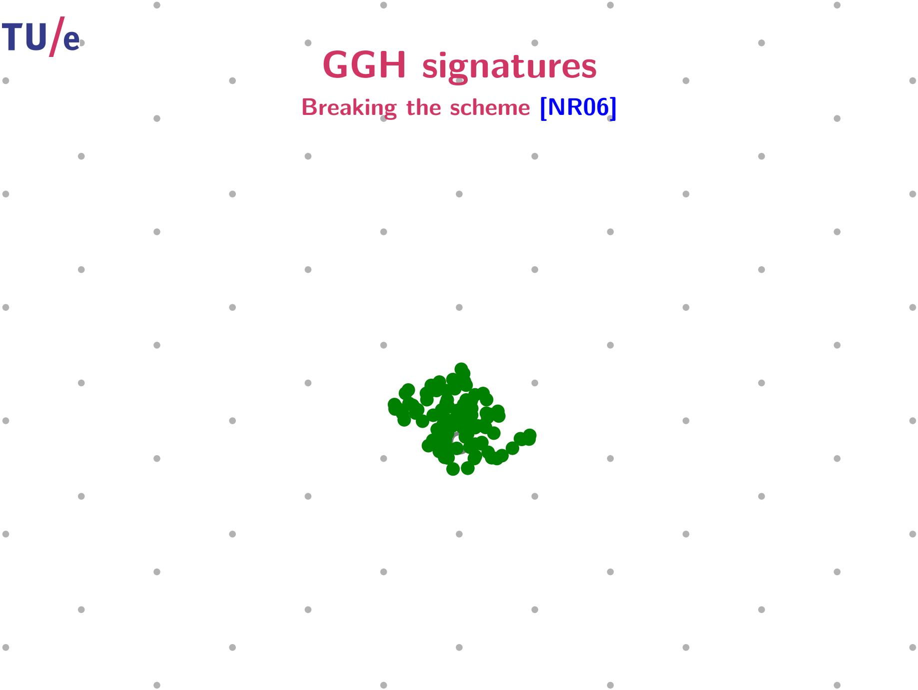
GGH signatures

Breaking the scheme [NR06]



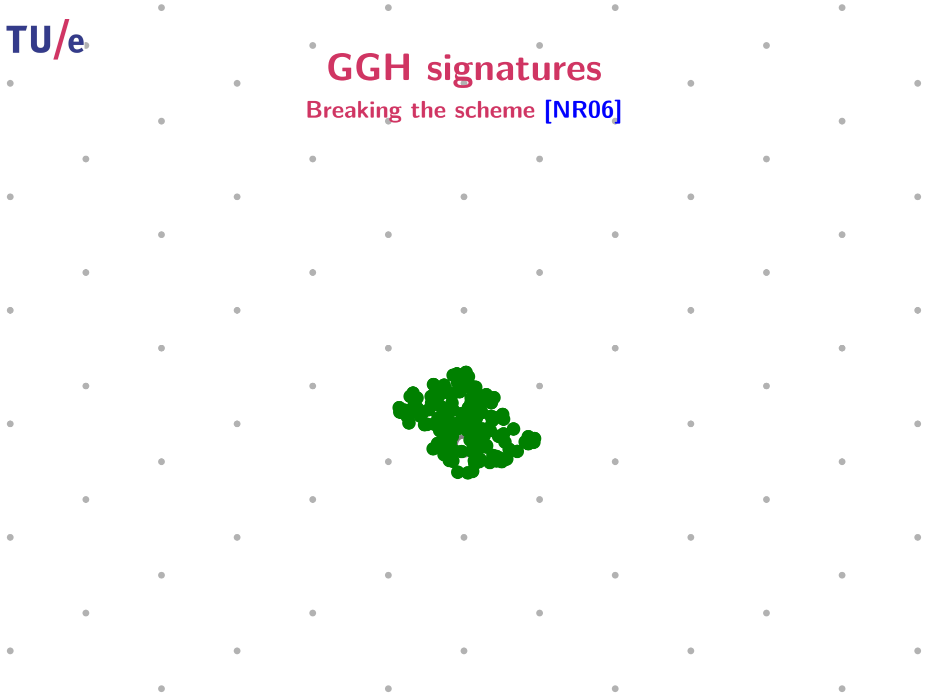
GGH signatures

Breaking the scheme [NR06]



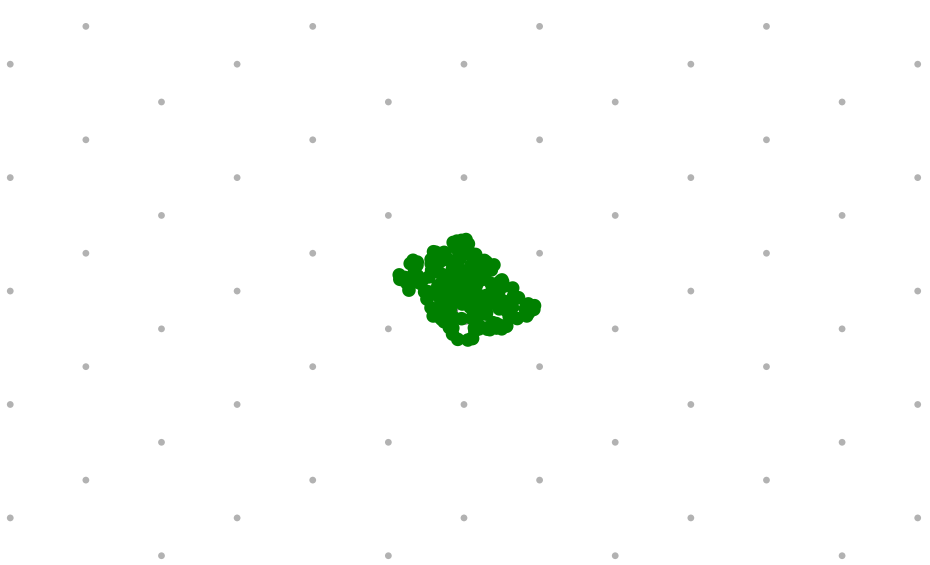
GGH signatures

Breaking the scheme [NR06]



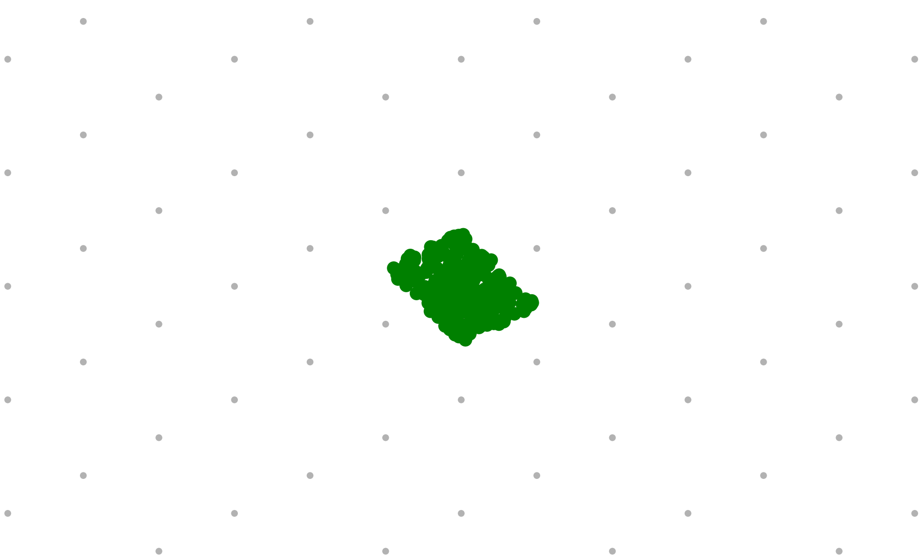
GGH signatures

Breaking the scheme [NR06]



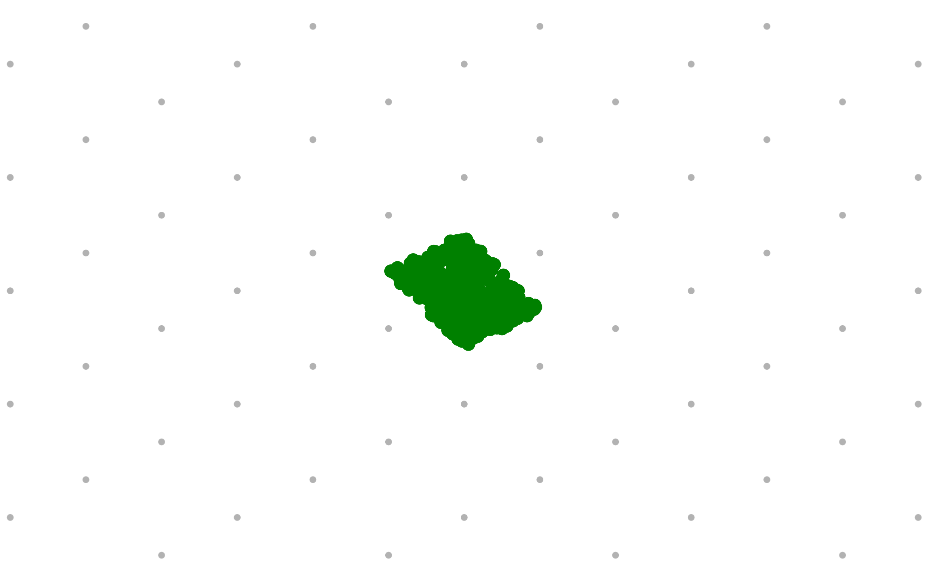
GGH signatures

Breaking the scheme [NR06]



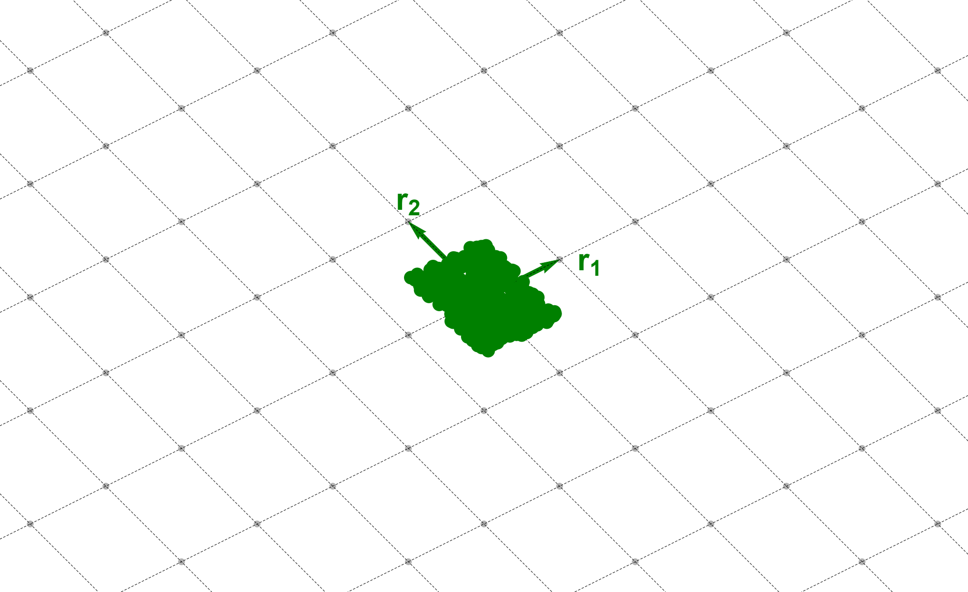
GGH signatures

Breaking the scheme [NR06]



GGH signatures

Breaking the scheme [NR06]



Lattices

Applications

- "Constructive cryptography": Lattice-based cryptosystems
  - Based on hard lattice problems (SVP, CVP, LWE, SIS)
  - Worst-case to average-case reductions [Ajt96]
  - Candidate for "post-quantum cryptography"
  - NTRU cryptosystem [HPS98, . . . , HPSSWZ15]
  - Fully homomorphic encryption [Gen09, . . . , CM15]
  - Multilinear maps(?) [GGH13, . . . , Cor15]
- "Destructive cryptography": Lattice cryptanalysis
  - Attack knapsack-based cryptosystems [Sha82, LO85, . . .]
  - Attack RSA with Coppersmith's method [Cop97, . . .]
  - Attack lattice-based cryptosystems [Ngu99, JJ00, . . .]

How does lattice-based cryptography work?

Lattices

Applications

- "Constructive cryptography": Lattice-based cryptosystems
  - Based on hard lattice problems (SVP, CVP, LWE, SIS)
  - Worst-case to average-case reductions [Ajt96]
  - Candidate for "post-quantum cryptography"
  - NTRU cryptosystem [HPS98, . . . , HPSSWZ15]
  - Fully homomorphic encryption [Gen09, . . . , CM15]
  - Multilinear maps(?) [GGH13, . . . , Cor15]
- "Destructive cryptography": Lattice cryptanalysis
  - Attack knapsack-based cryptosystems [Sha82, LO85, . . .]
  - Attack RSA with Coppersmith's method [Cop97, . . .]
  - Attack lattice-based cryptosystems [Ngu99, JJ00, . . .]

How does lattice-based cryptography work?
How hard are hard lattice problems such as SVP?

Lattices

Exact SVP algorithms

	Algorithm	$\log_2(\text{Time})$	$\log_2(\text{Space})$
Provable SVP	Enumeration [Poh81, Kan83, . . . , MW15]	$\Omega(n \log n)$	$O(\log n)$
	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 [AEVZ02, MV10b]	$2.000n$	$1.000n$
	Discrete Gaussians [ADRS15, ADS15, Ste16]	$1.000n$	$1.000n$
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 sieve [BGJ14]	$0.3774n$	$0.293n$
	Hyperplane LSH [Laa15, MLB15, Mar15]	$0.337n$	$0.208n$
	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 [BDGL16, Laa15, ML15]	$0.293n$	$0.208n$

Nguyen–Vidick sieve

Nguyen–Vidick sieve

1. Sample a list L of random lattice vectors

Nguyen–Vidick sieve

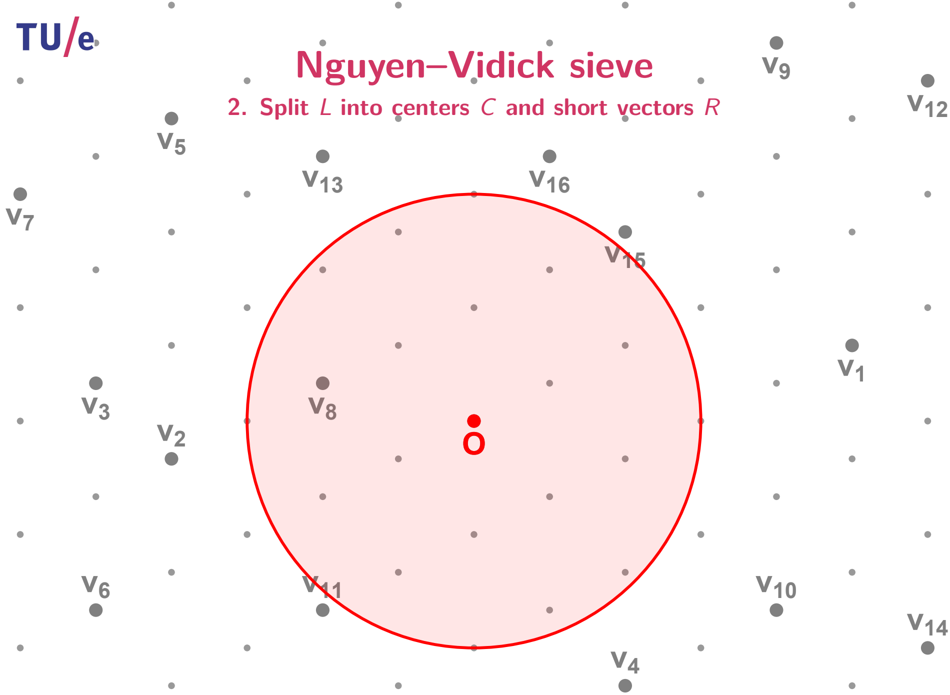
1. Sample a list L of random lattice vectors

Nguyen–Vidick sieve

2. Split L into centers C and short vectors R

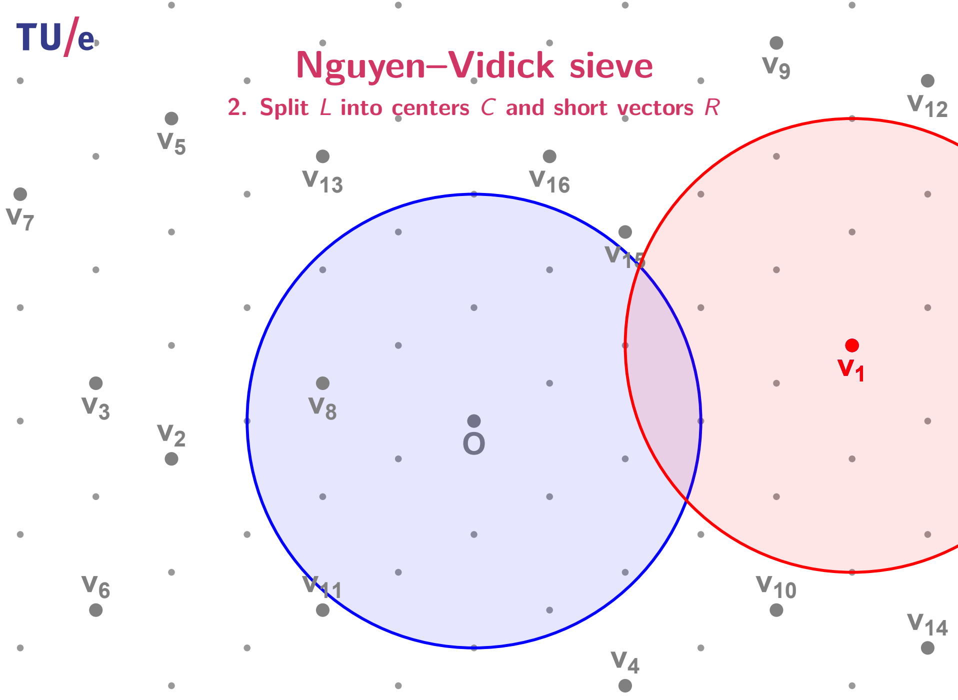
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



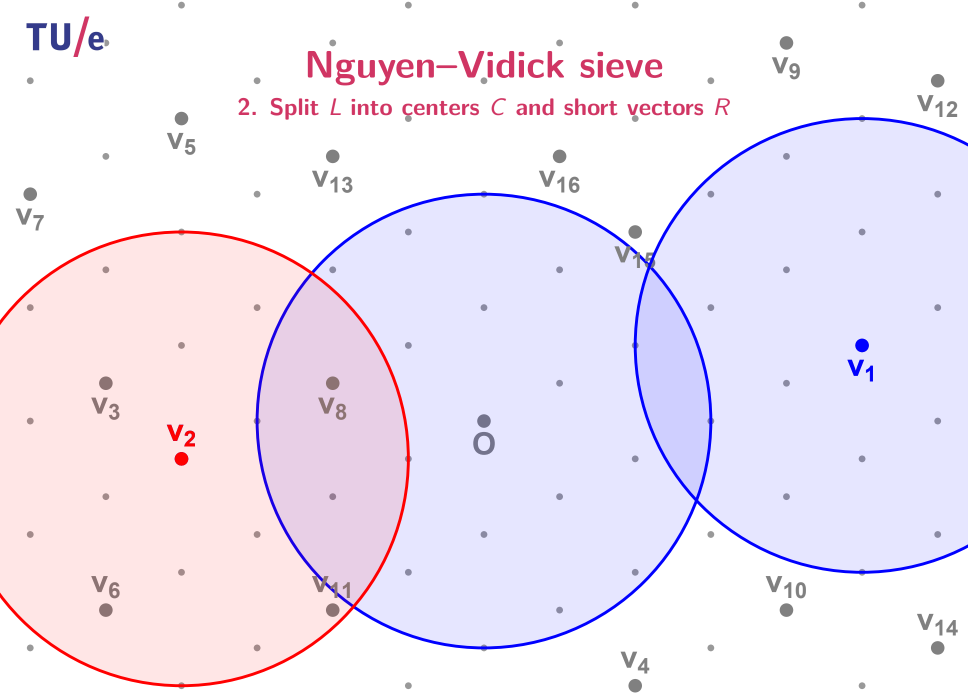
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



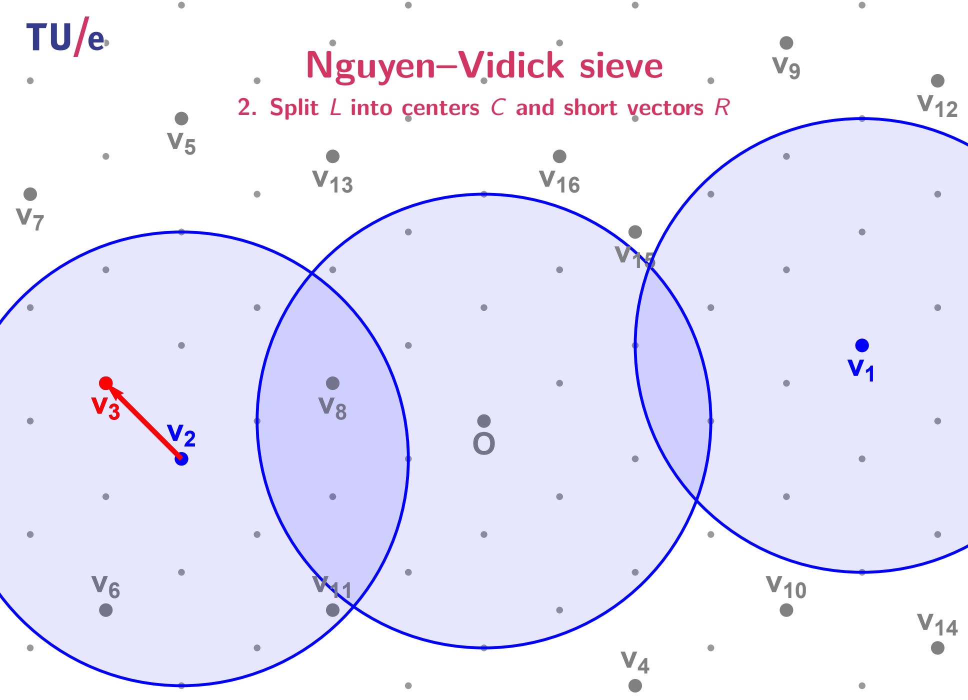
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



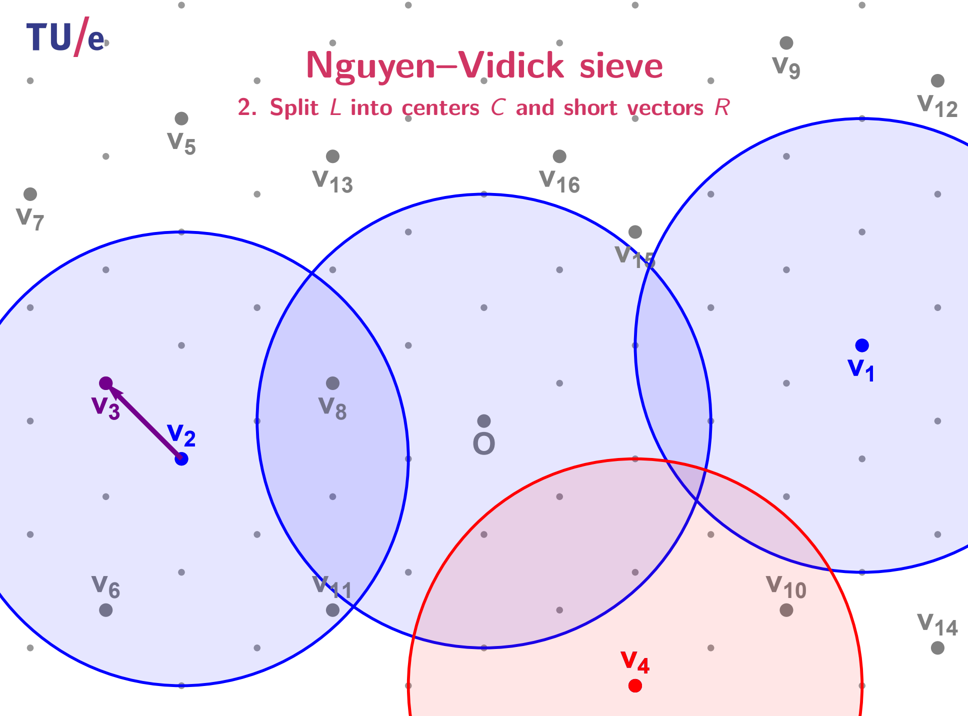
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



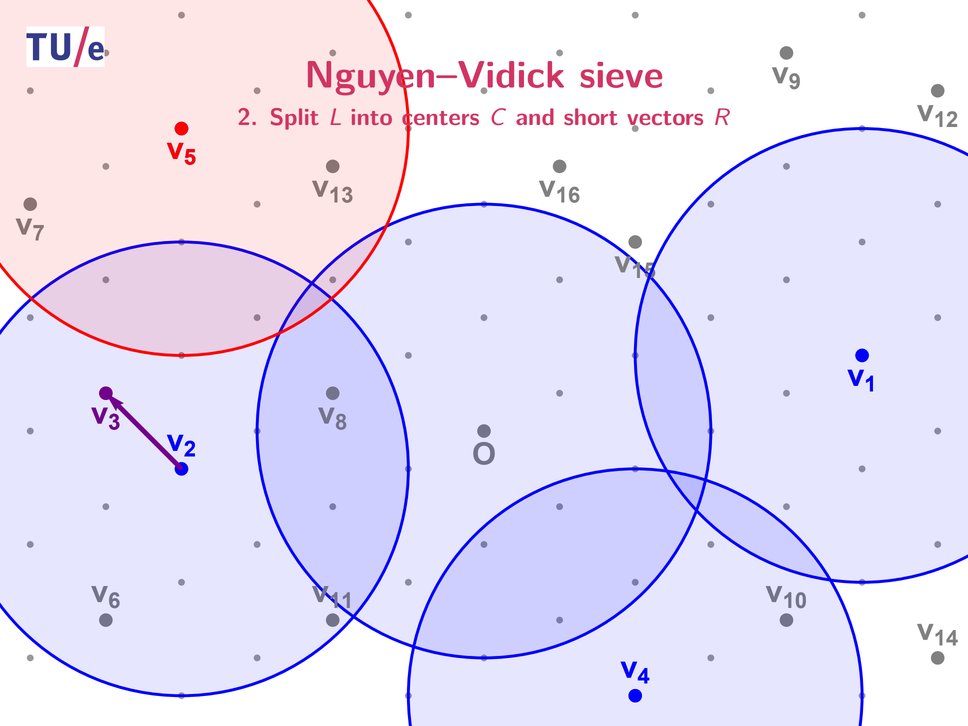
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



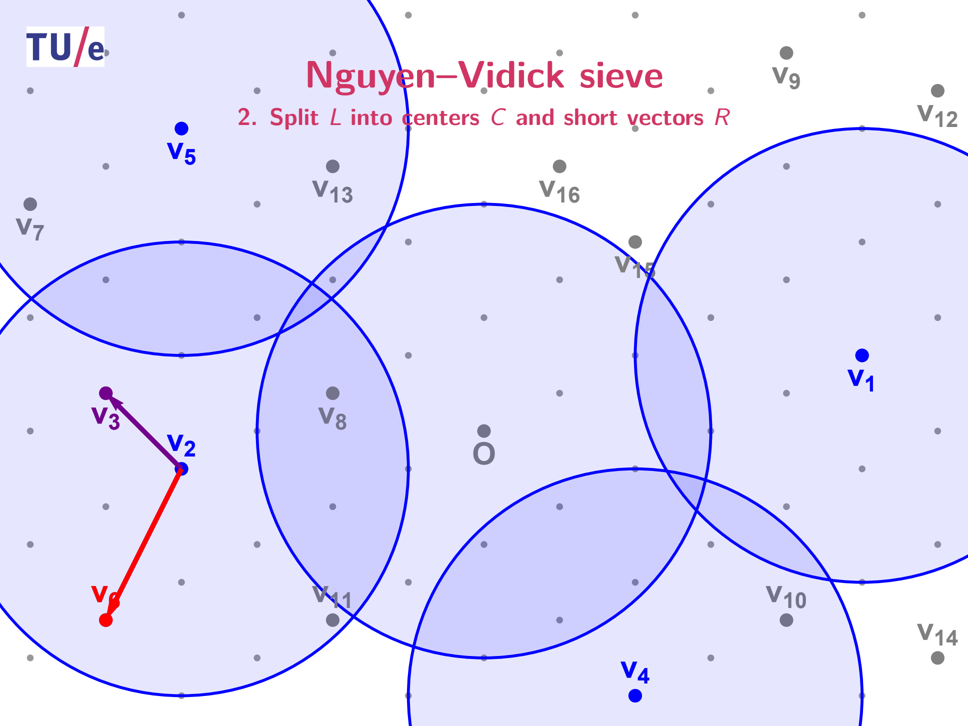
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



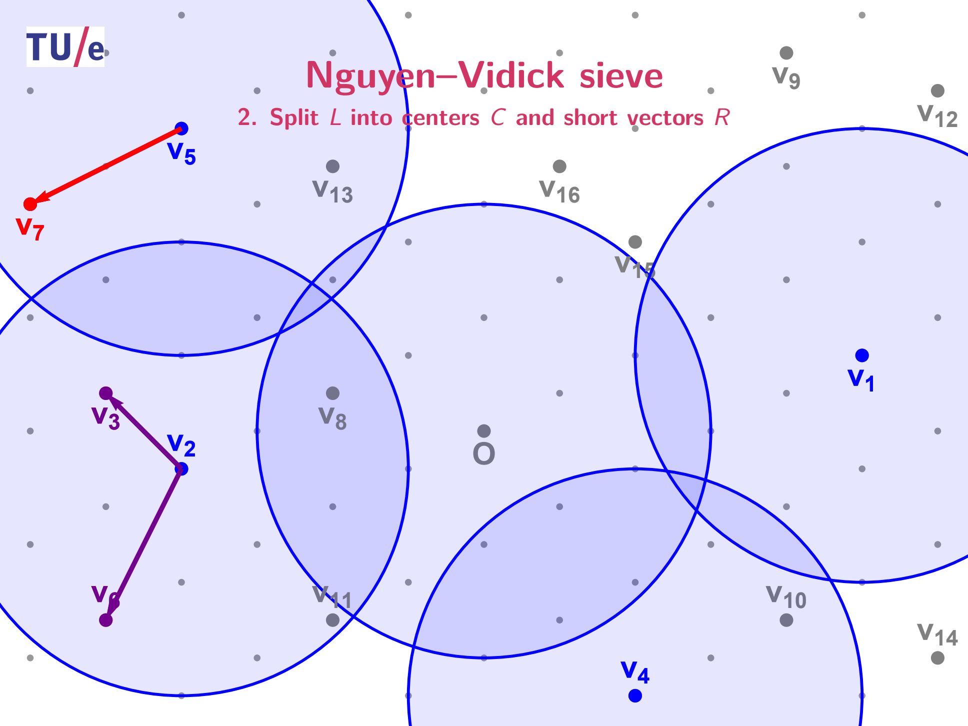
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



Nguyen–Vidick sieve

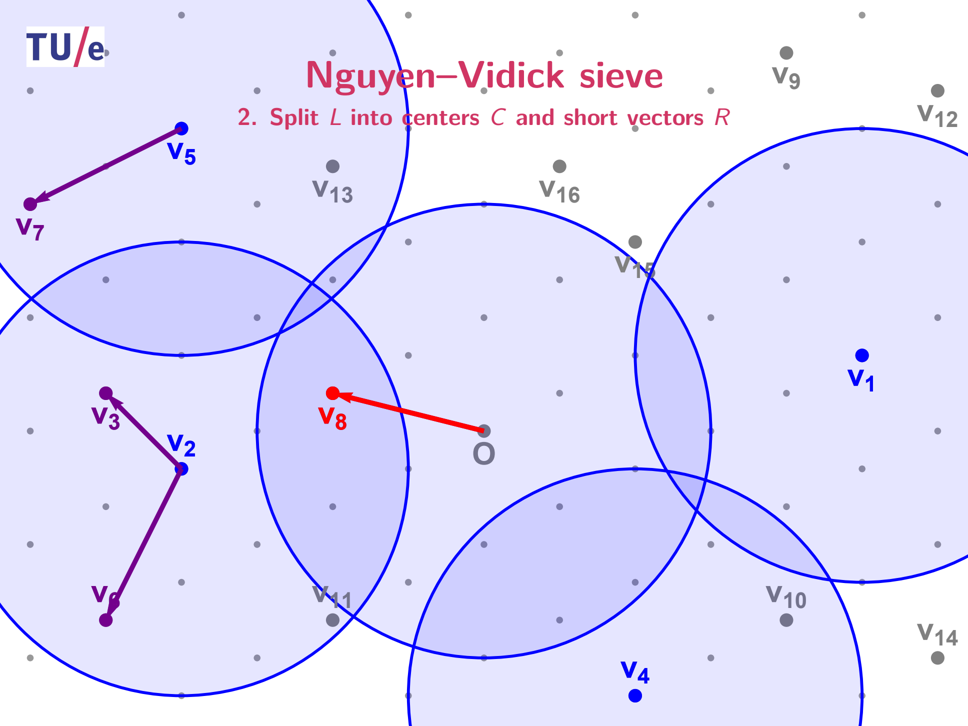
2. Split L into centers C and short vectors R



TU/e

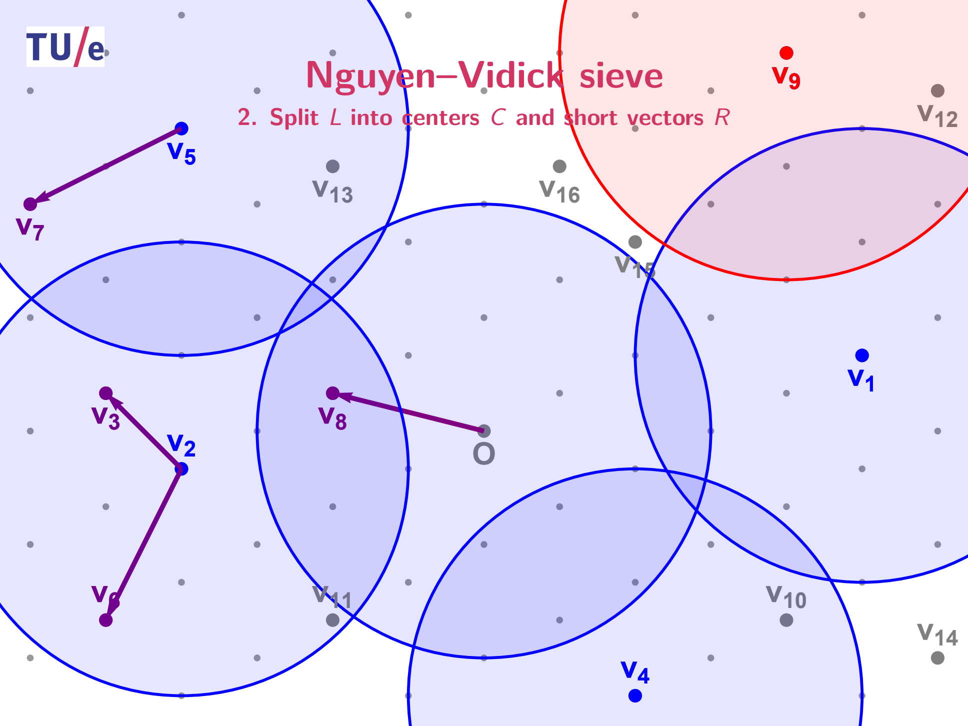
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



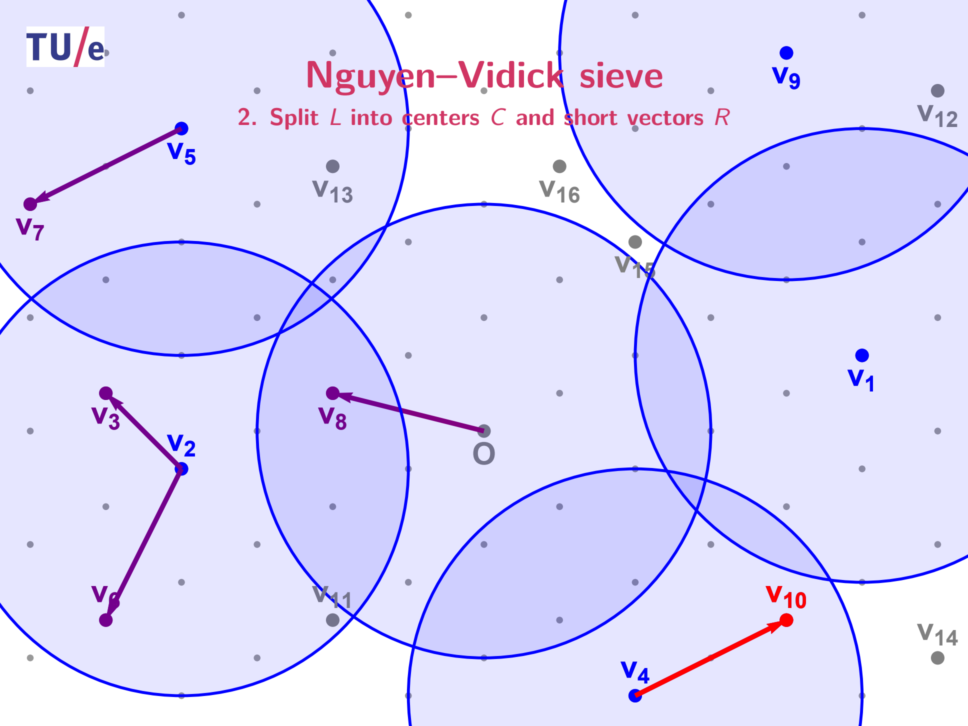
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



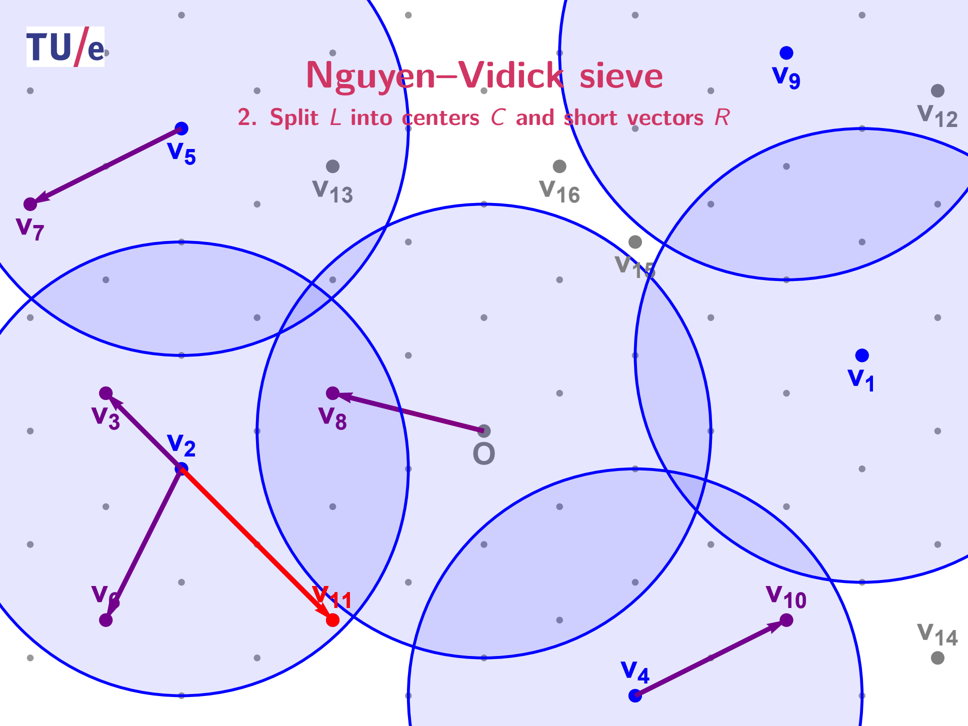
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



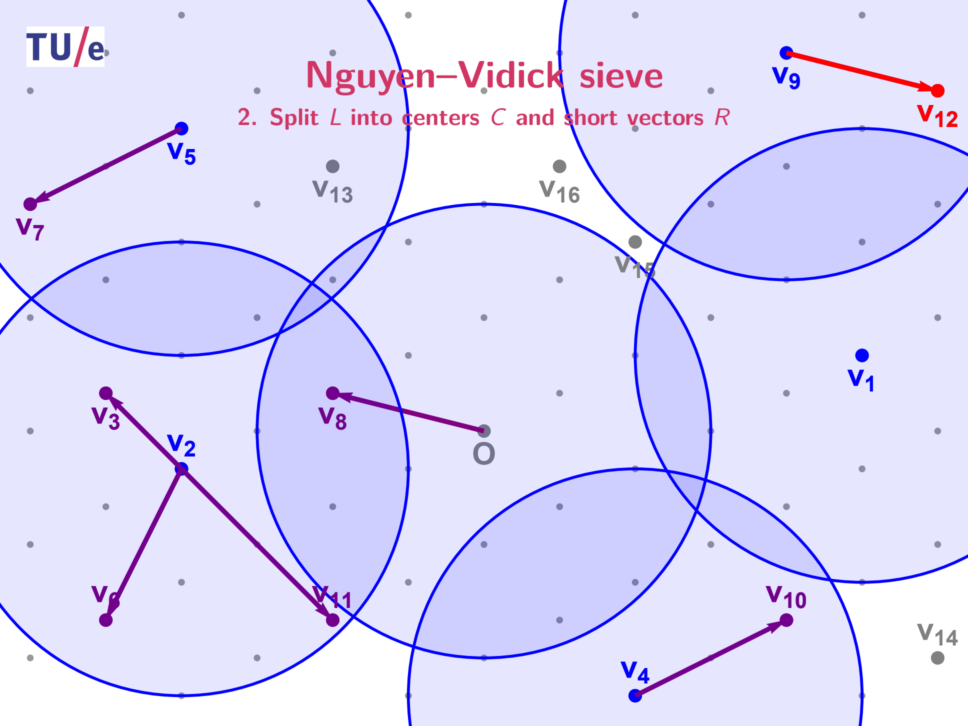
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



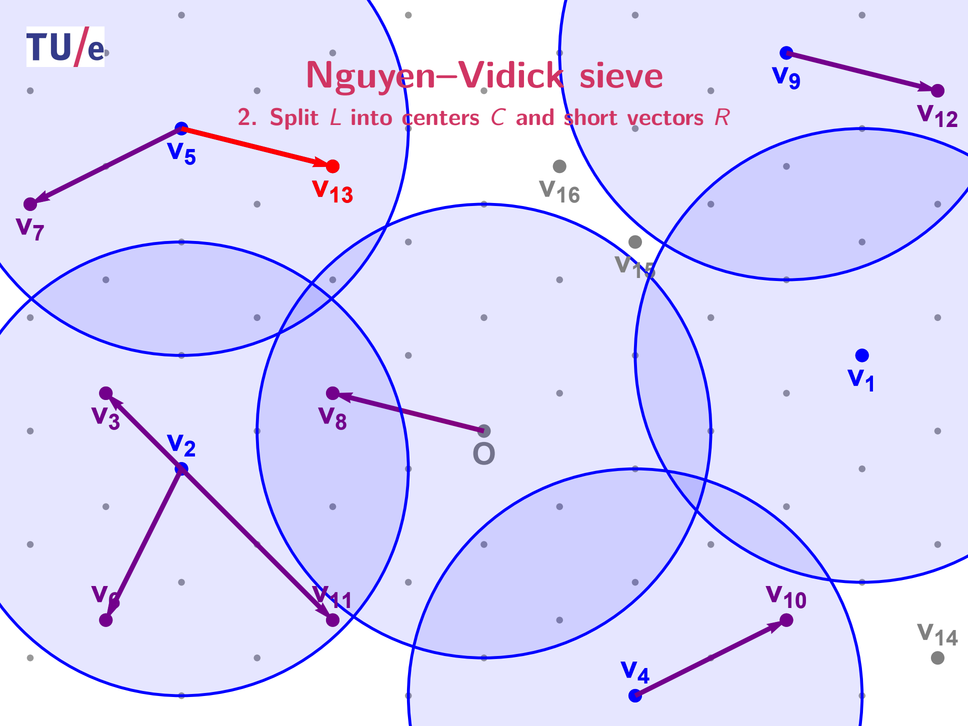
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



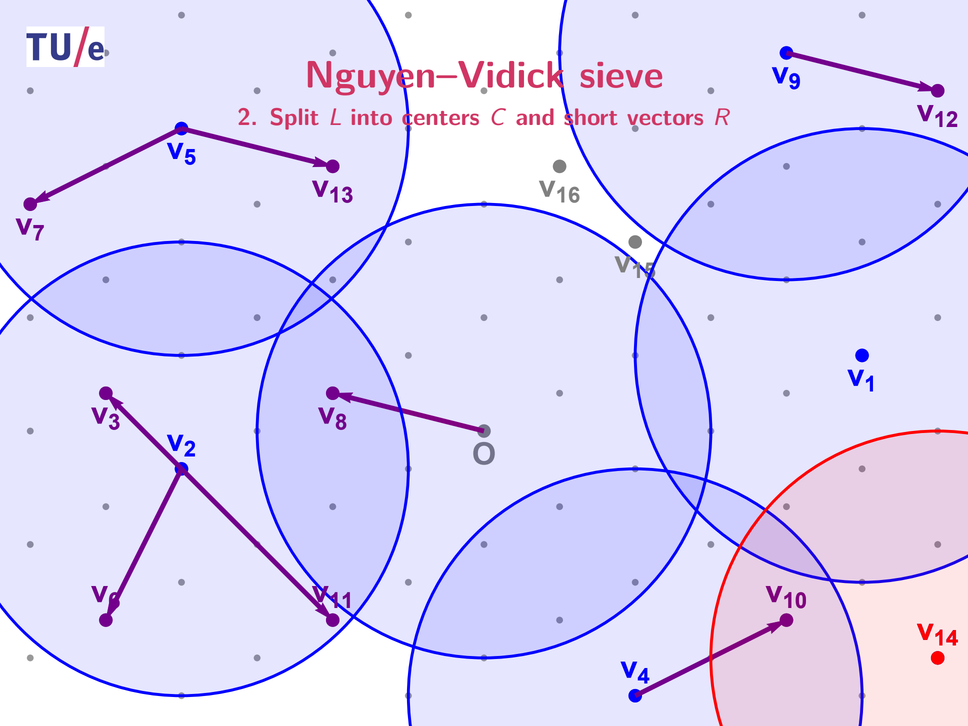
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



Nguyen–Vidick sieve

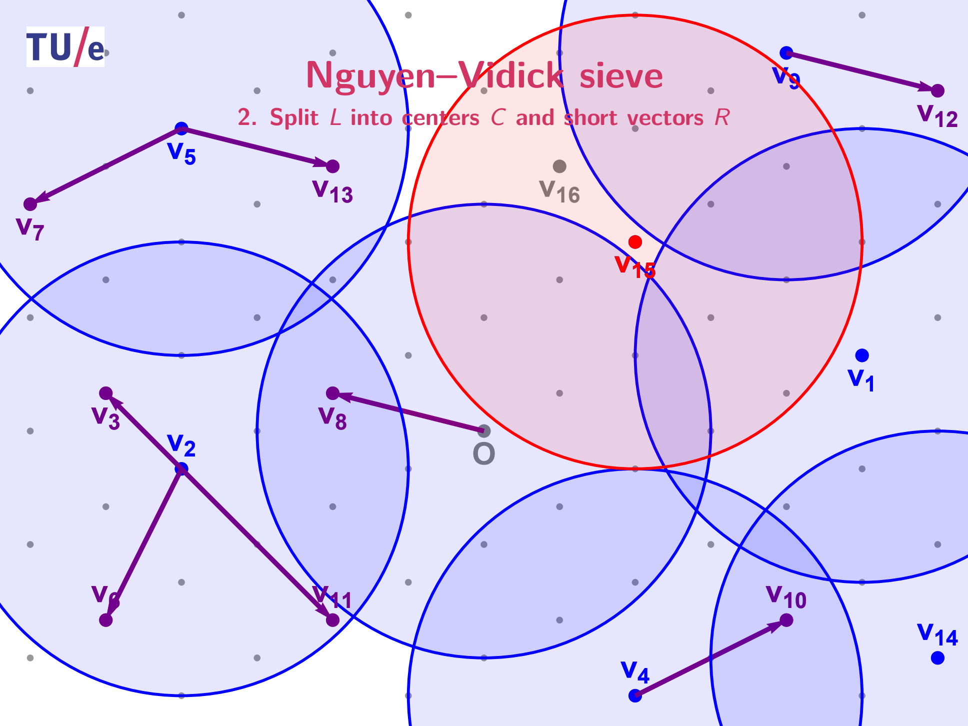
2. Split L into centers C and short vectors R



TU/e

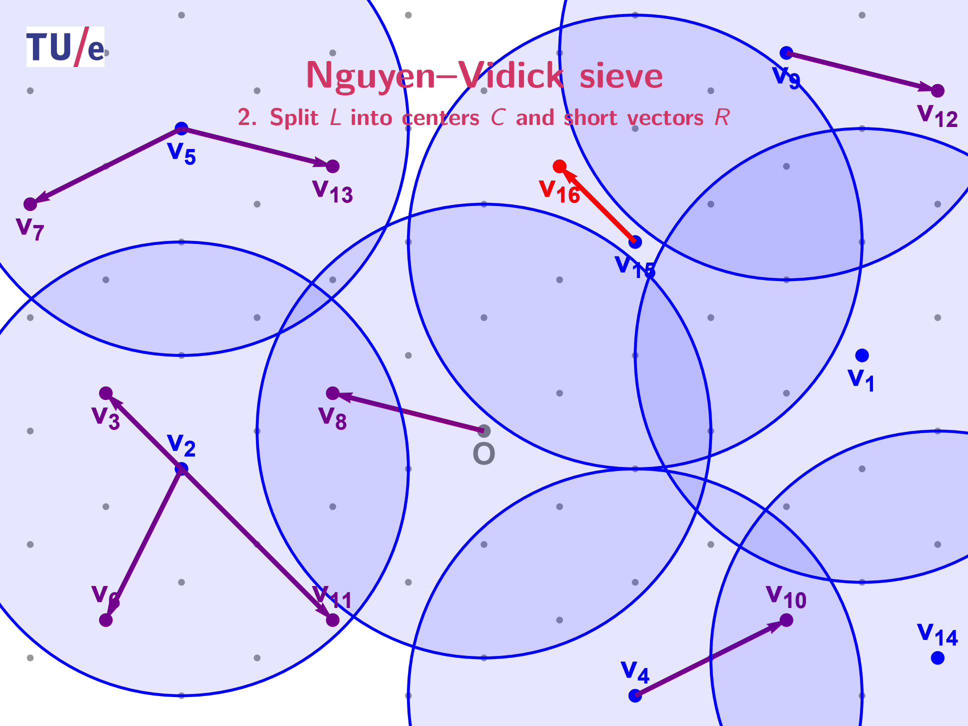
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



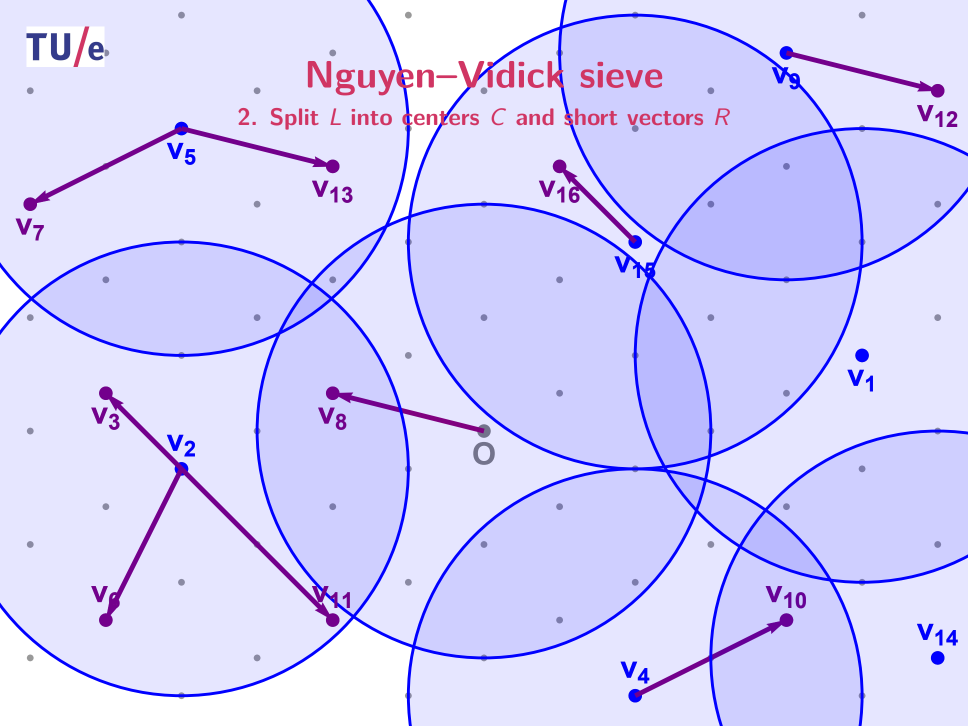
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



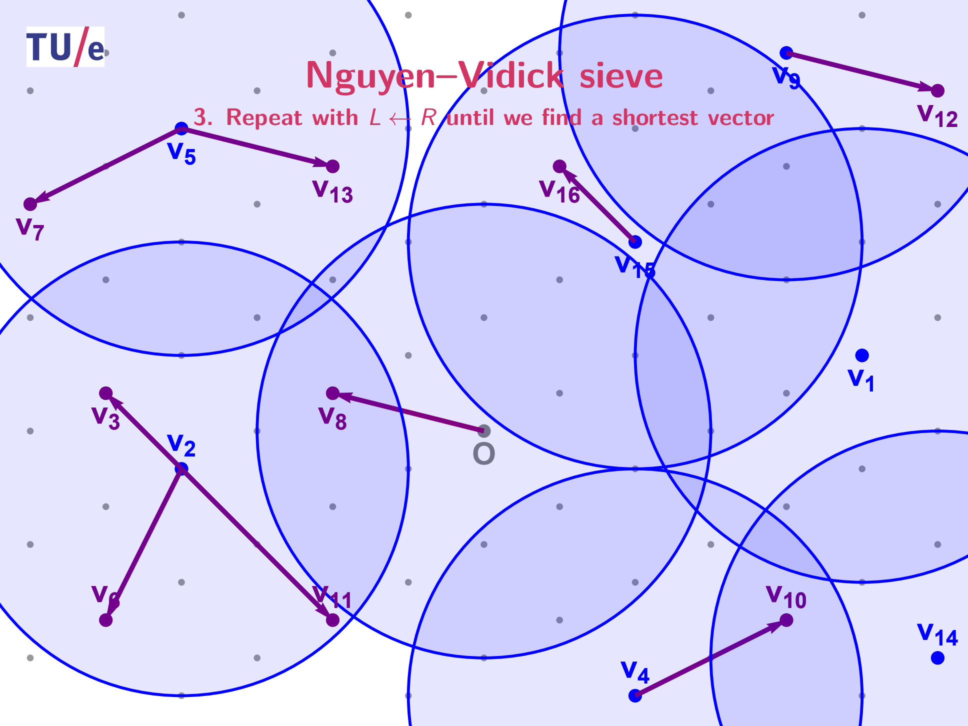
Nguyen–Vidick sieve

2. Split L into centers C and short vectors R



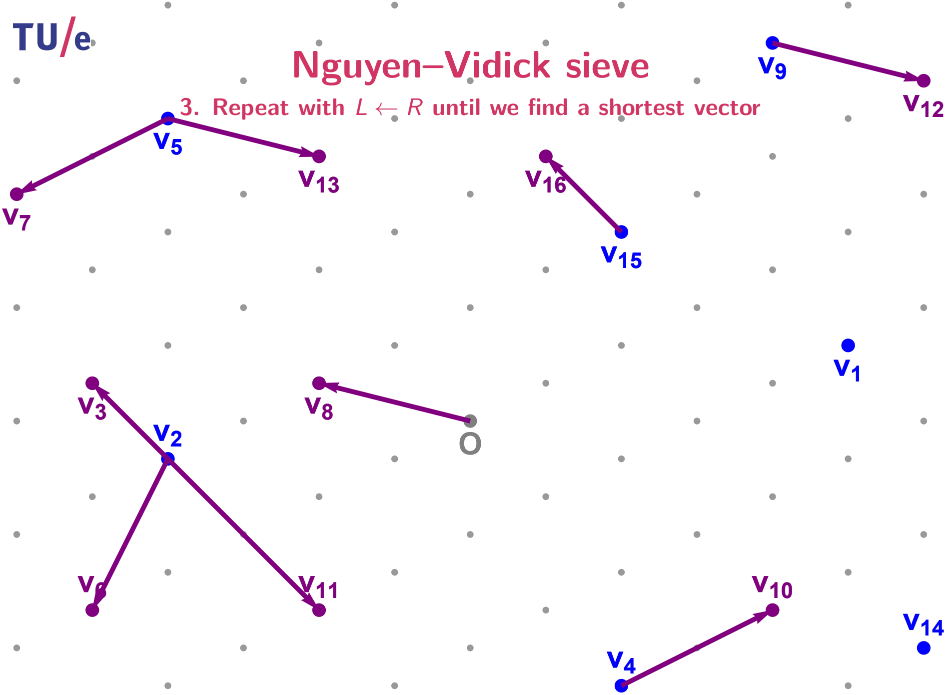
Nguyen–Vidick sieve

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



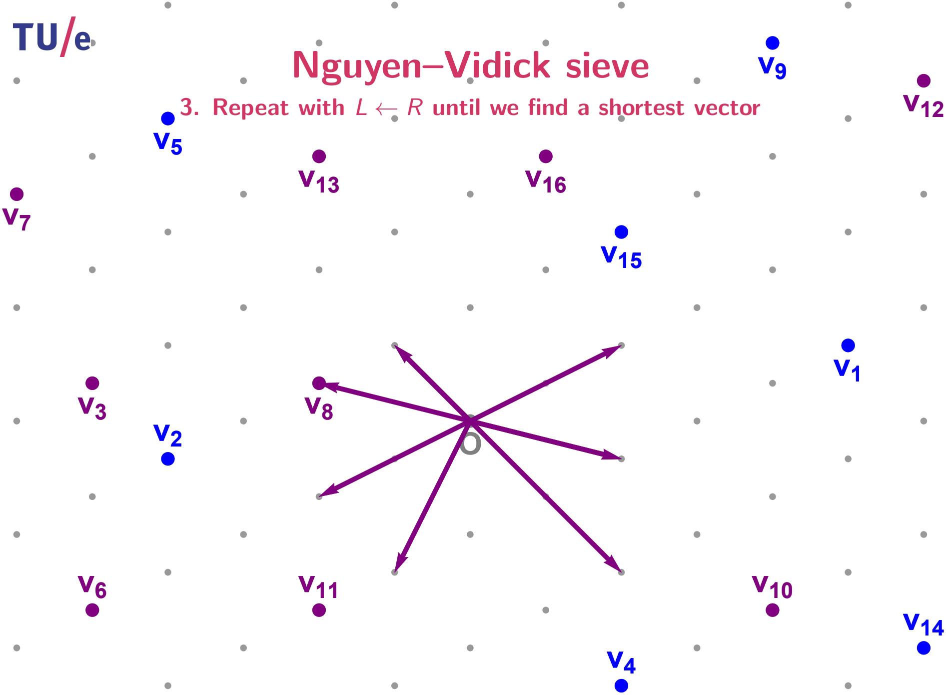
Nguyen–Vidick sieve

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



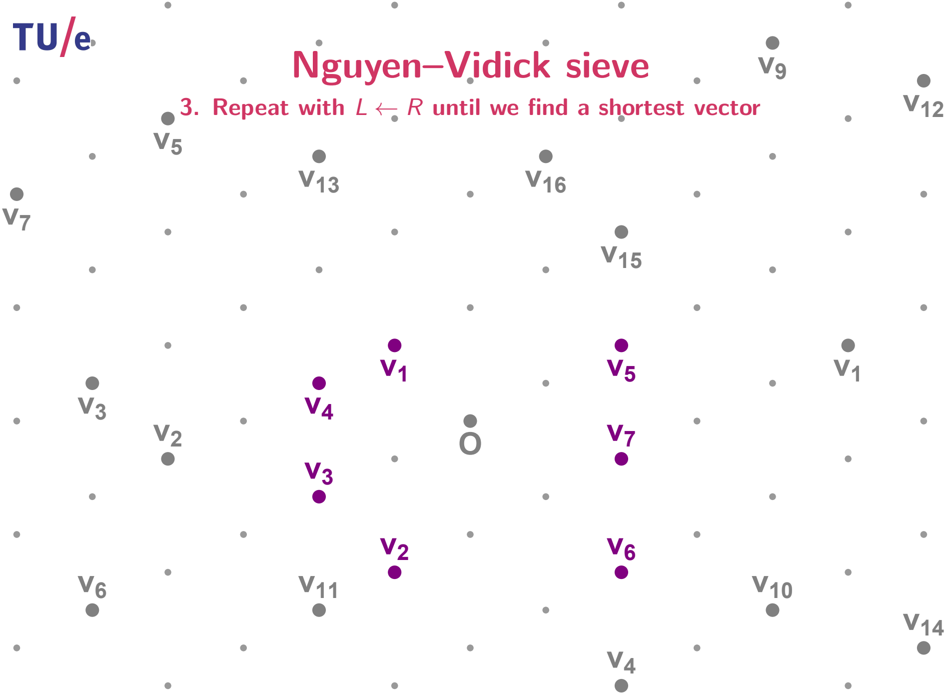
Nguyen–Vidick sieve

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

- Space complexity: $\sqrt{4/3}^n \approx 2^{0.21n+o(n)}$ vectors
  - Need $\sqrt{4/3}^n$ vectors to cover all corners of $\mathbb{R}^n$

Nguyen–Vidick sieve

Overview

- Space complexity: $\sqrt{4/3}^n \approx 2^{0.21n+o(n)}$ vectors
  - Need $\sqrt{4/3}^n$ vectors to cover all corners of $\mathbb{R}^n$
- Time complexity: $(4/3)^n \approx 2^{0.42n+o(n)}$
  - Comparing a target vector to all centers: $2^{0.21n+o(n)}$
  - Repeating this for each list vector: $2^{0.21n+o(n)}$
  - Repeating the whole sieving procedure: $\mathrm{poly}(n)$

Nguyen–Vidick sieve

Overview

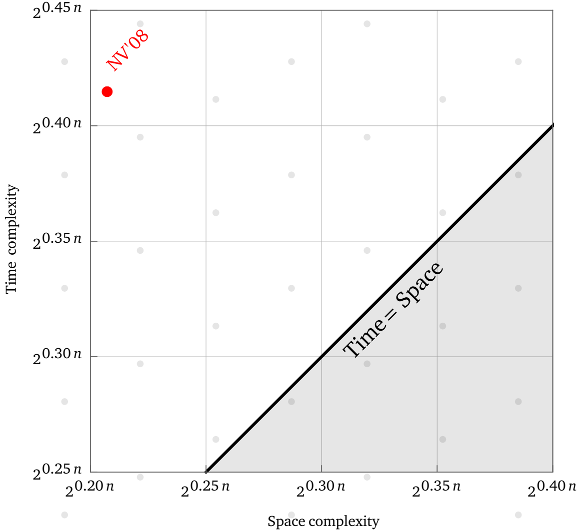
- Space complexity: $\sqrt{4/3}^n \approx 2^{0.21n+o(n)}$ vectors
  - Need $\sqrt{4/3}^n$ vectors to cover all corners of $\mathbb{R}^n$
- Time complexity: $(4/3)^n \approx 2^{0.42n+o(n)}$
  - Comparing a target vector to all centers: $2^{0.21n+o(n)}$
  - Repeating this for each list vector: $2^{0.21n+o(n)}$
  - Repeating the whole sieving procedure: $\mathrm{poly}(n)$

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

The NV-sieve runs in time $2^{0.42n+o(n)}$ and space $2^{0.21n+o(n)}$.

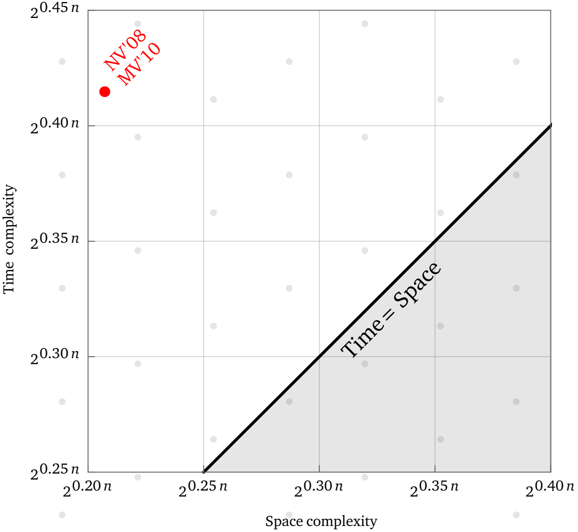
Nguyen–Vidick sieve

Space/time trade-off



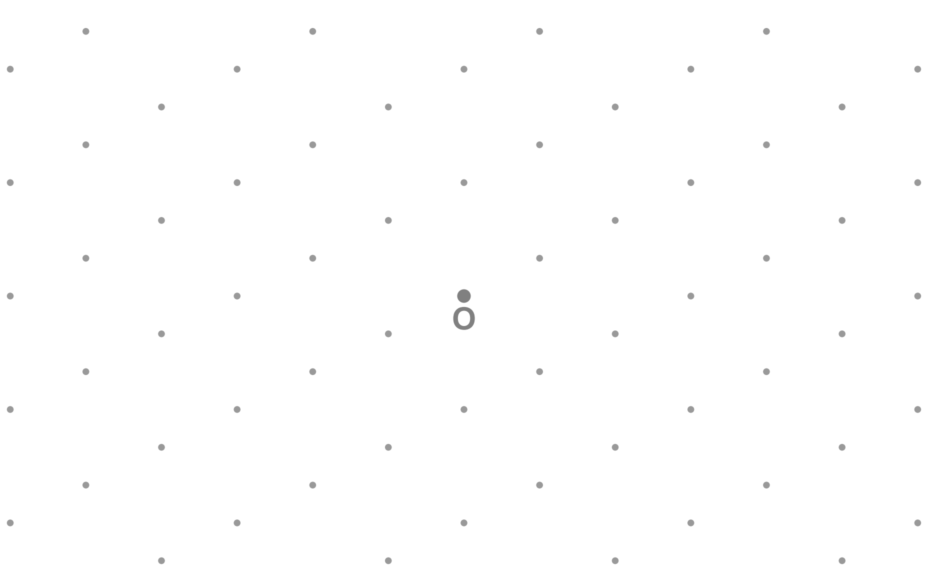
GaussSieve

Space/time trade-off



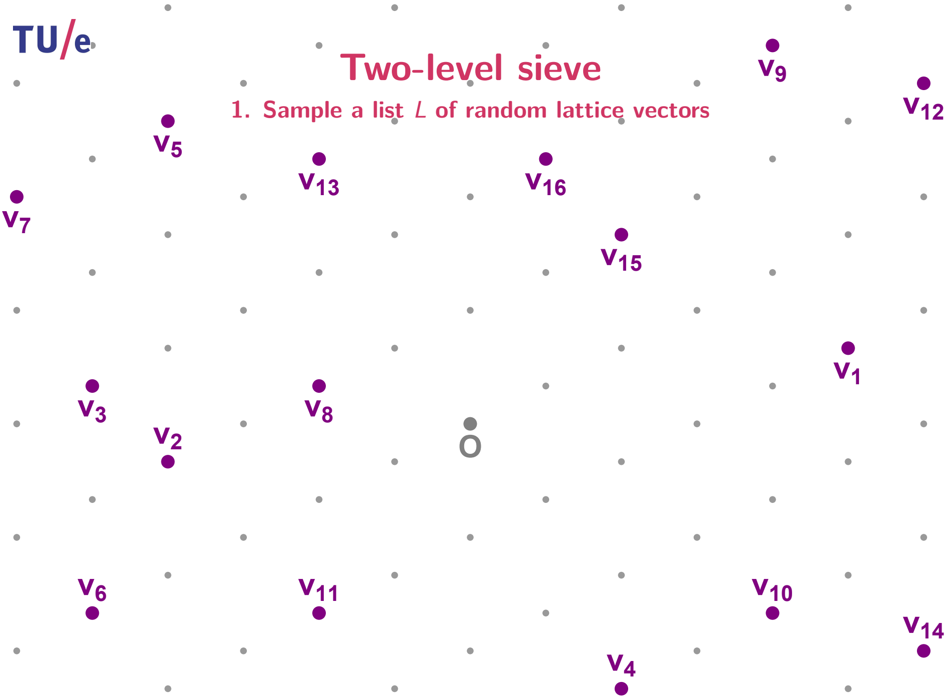
Two-level sieve

1. Sample a list L of random lattice vectors



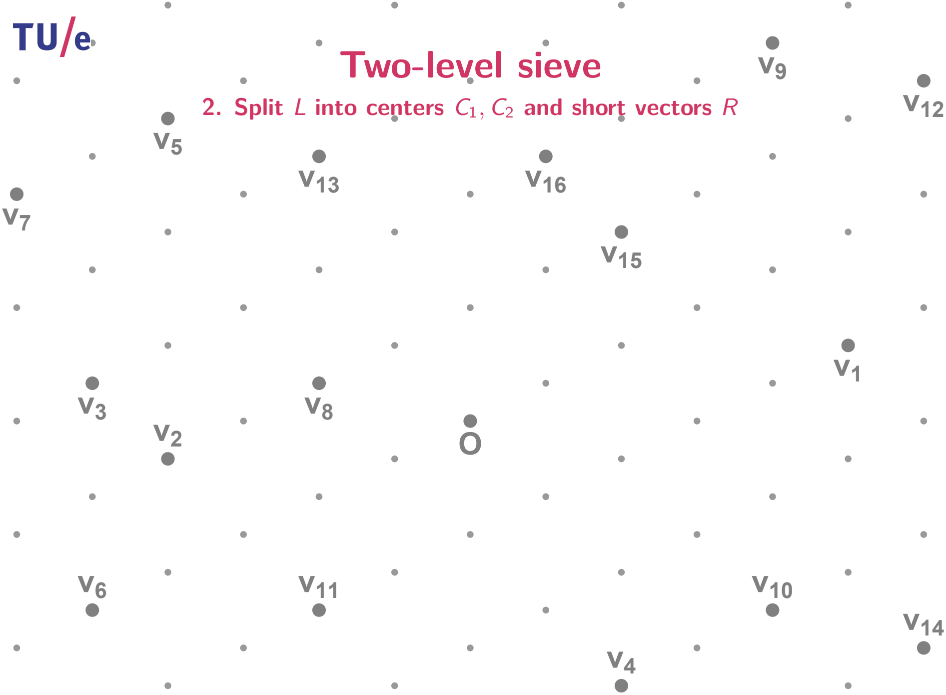
Two-level sieve

1. Sample a list L of random lattice vectors



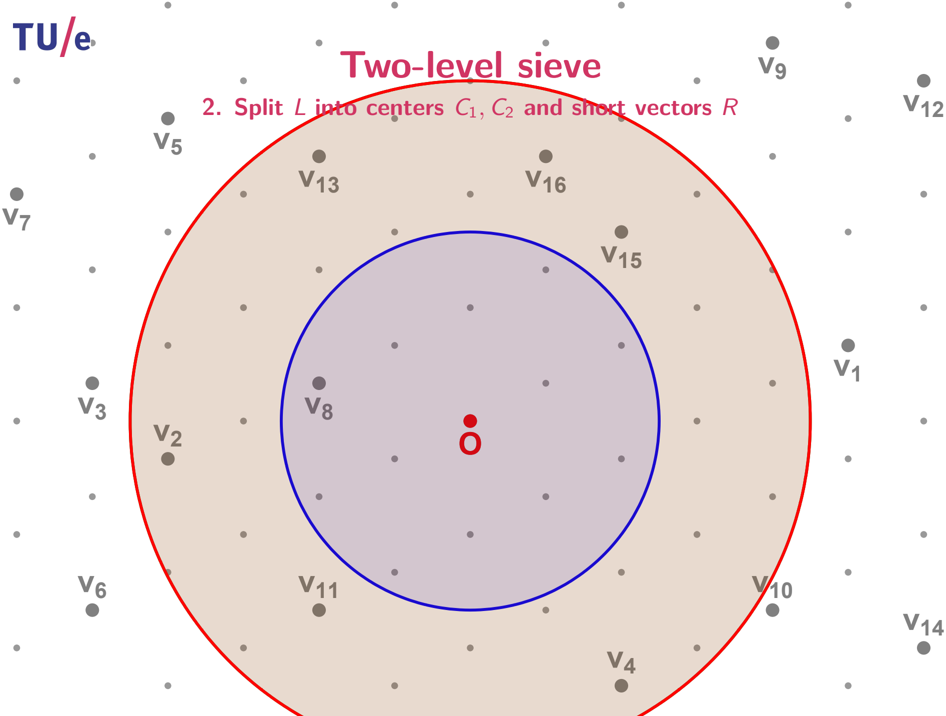
Two-level sieve

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



Two-level sieve

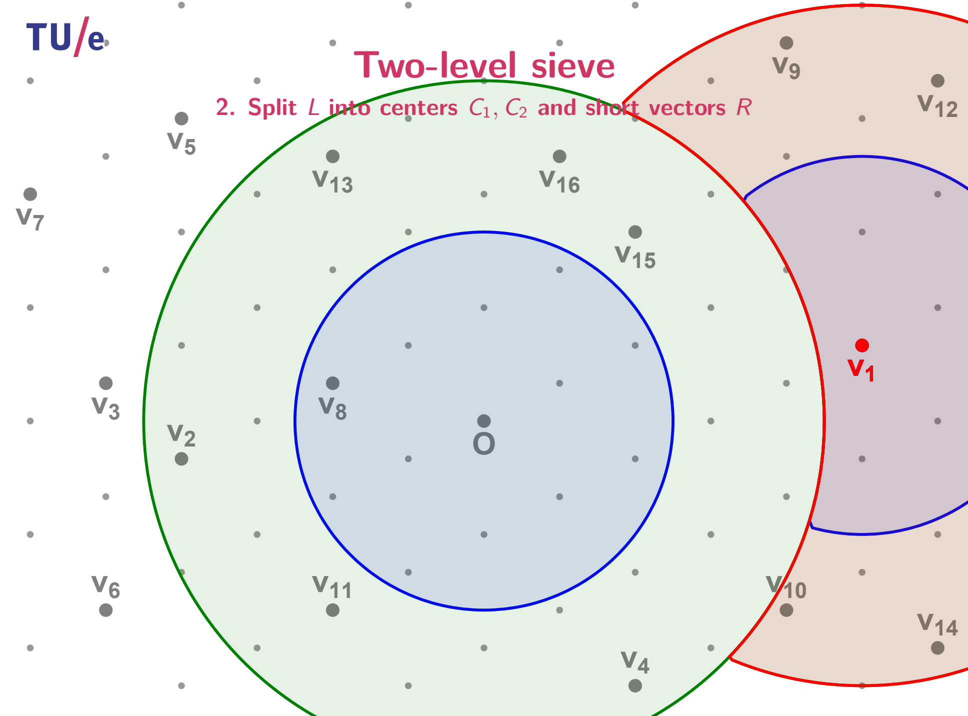
2. Split L into centers C_1, C_2 and short vectors R



TU/e

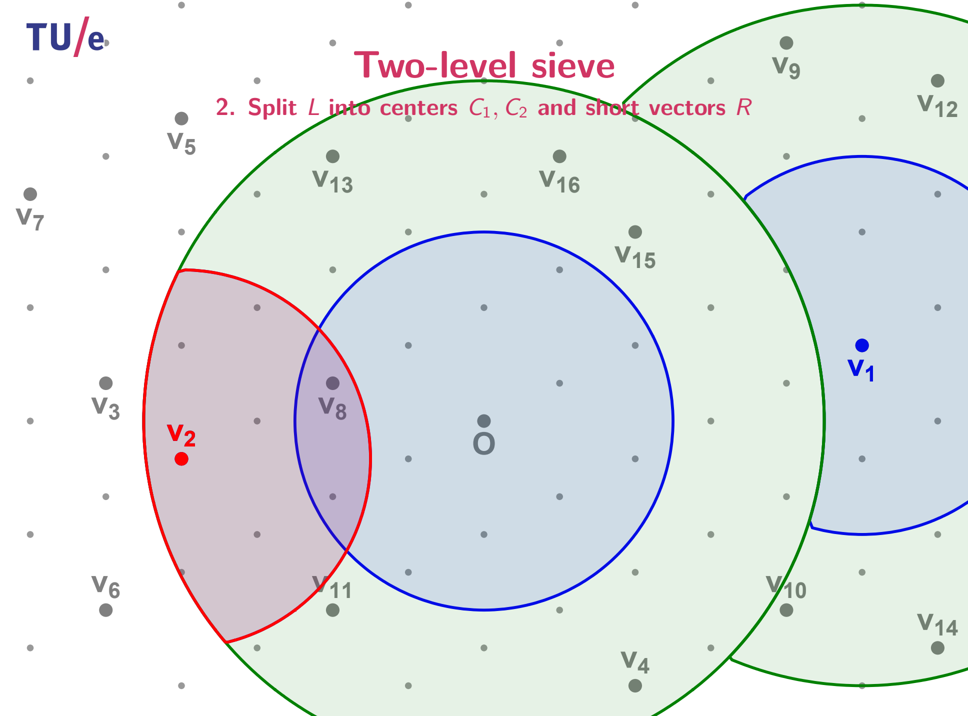
Two-level sieve

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



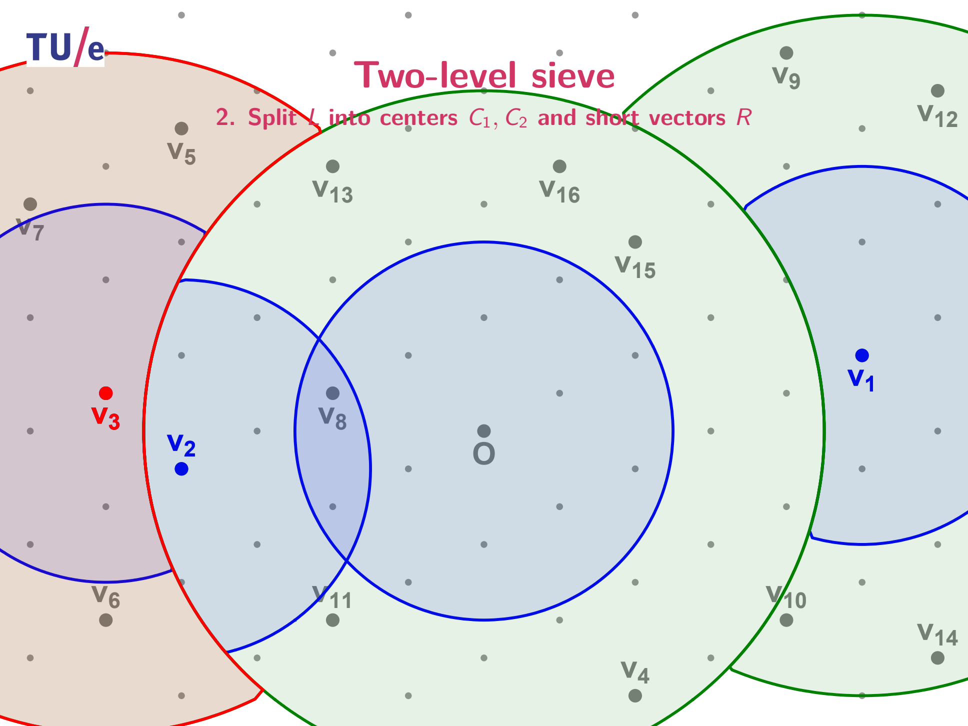
Two-level sieve

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



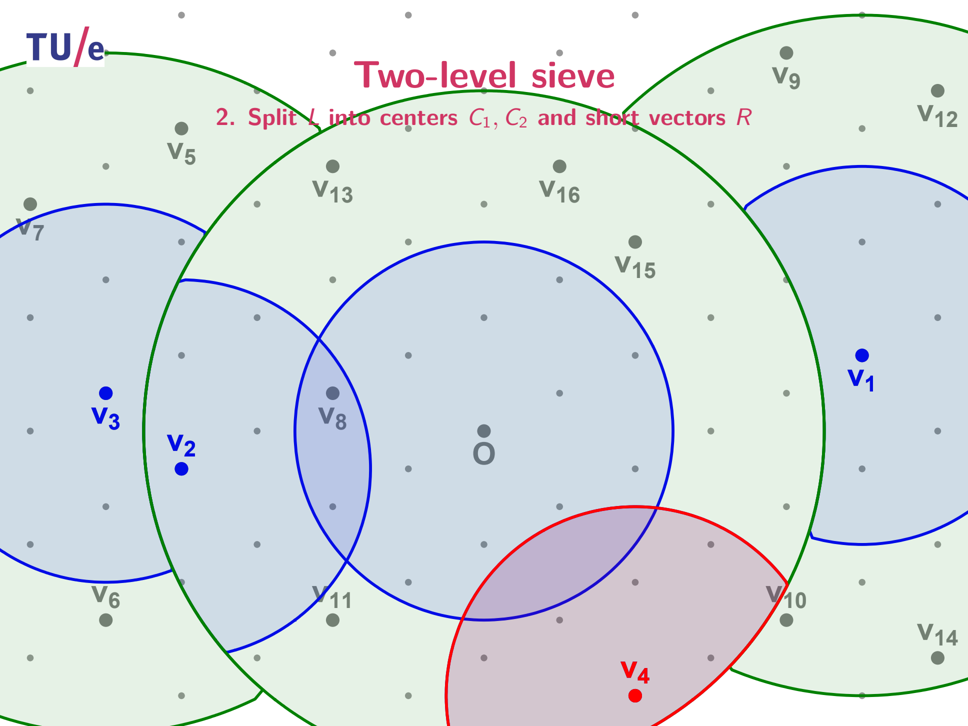
Two-level sieve

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



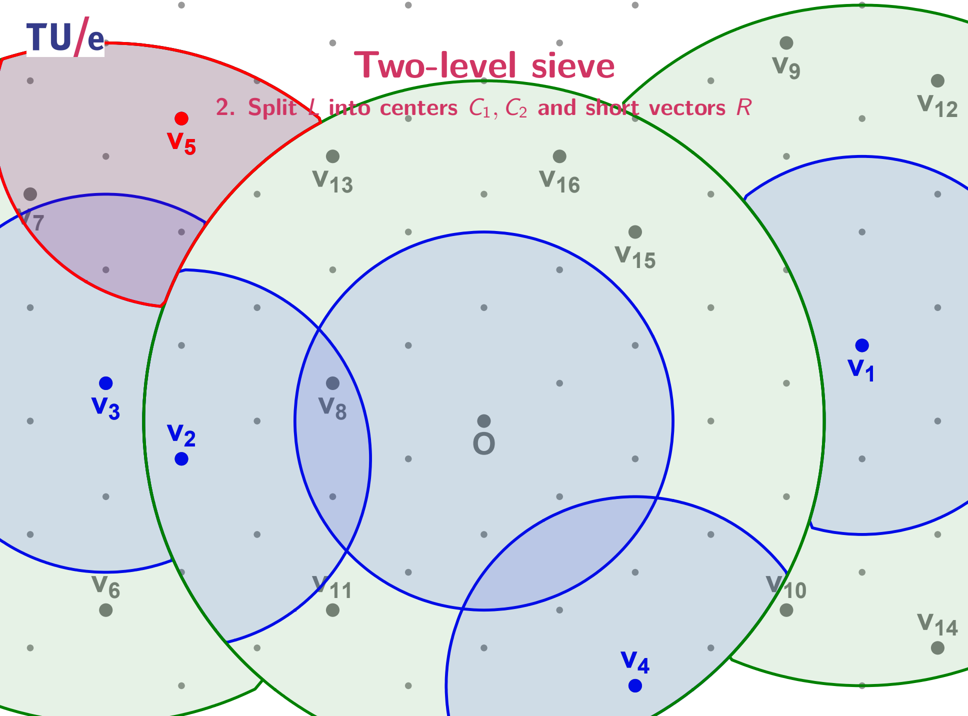
Two-level sieve

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



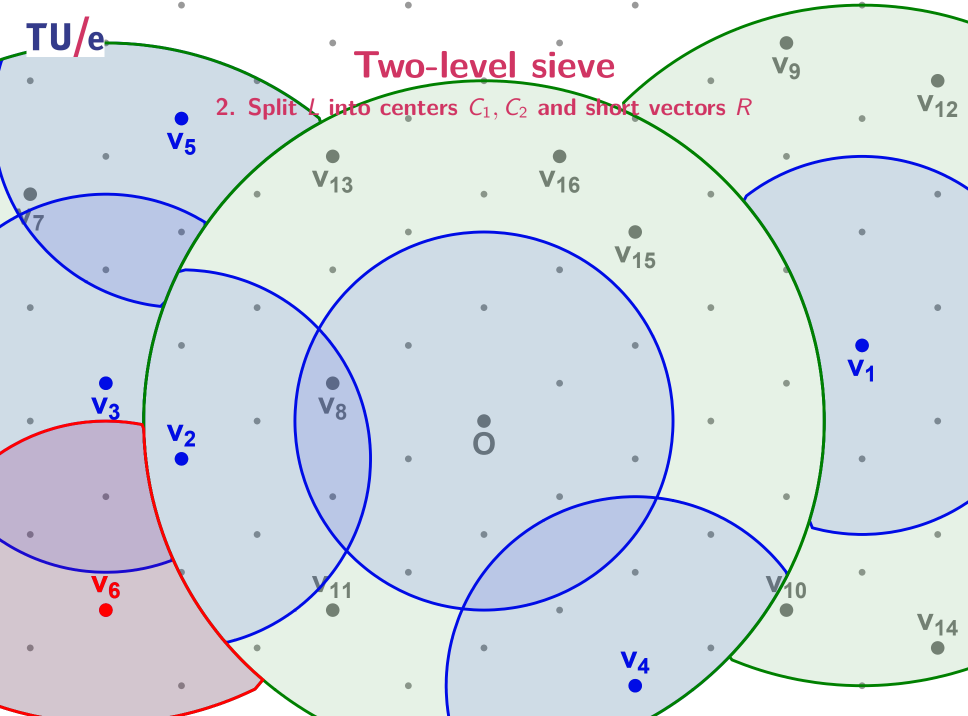
Two-level sieve

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



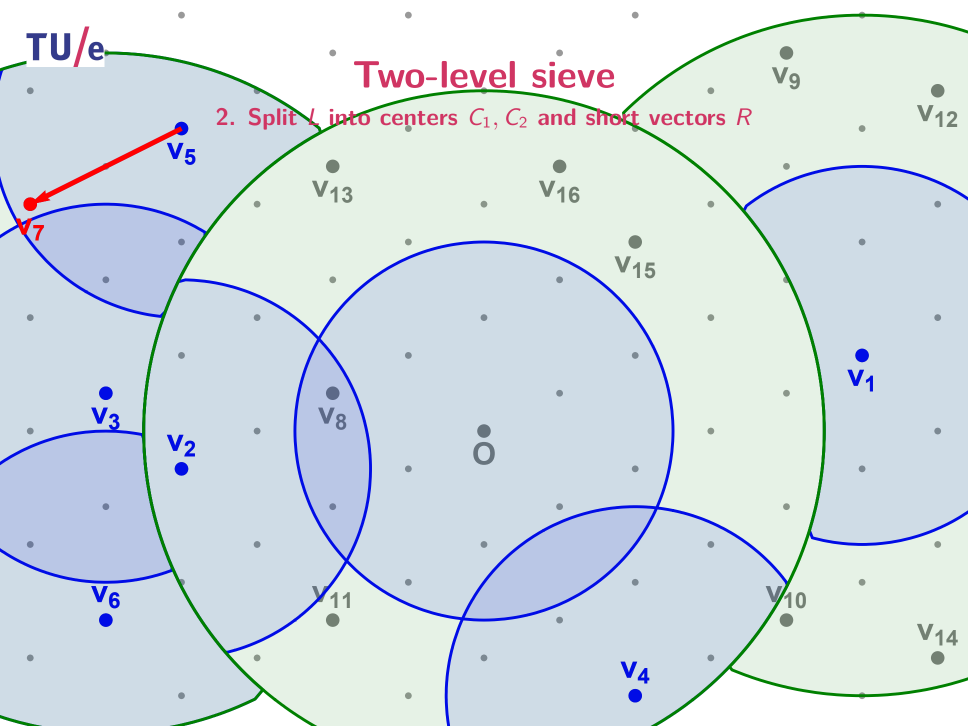
Two-level sieve

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



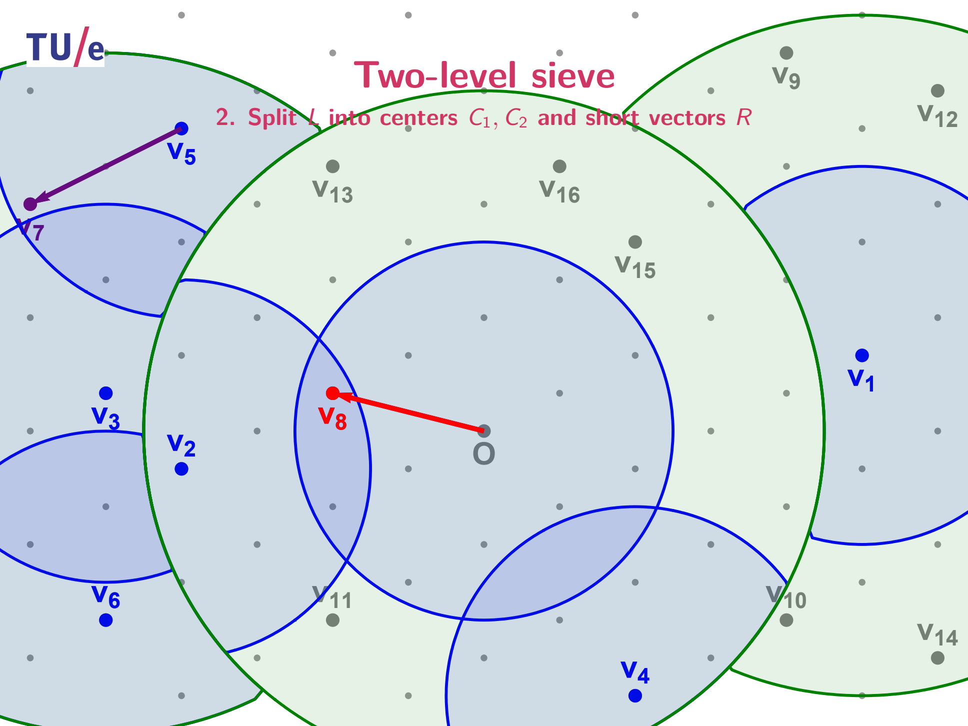
Two-level sieve

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



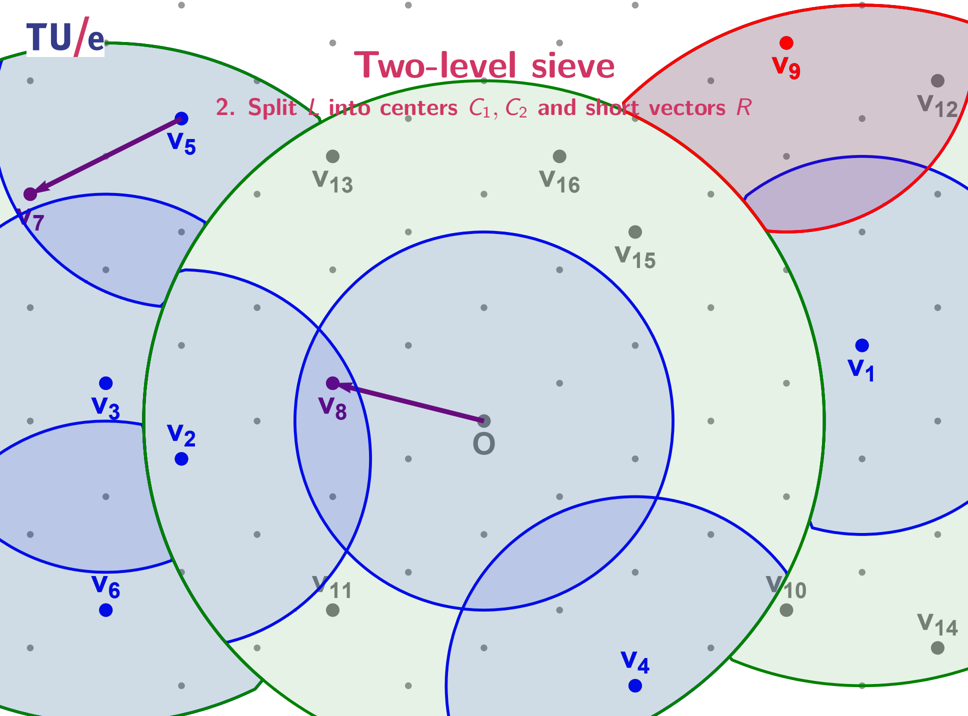
Two-level sieve

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



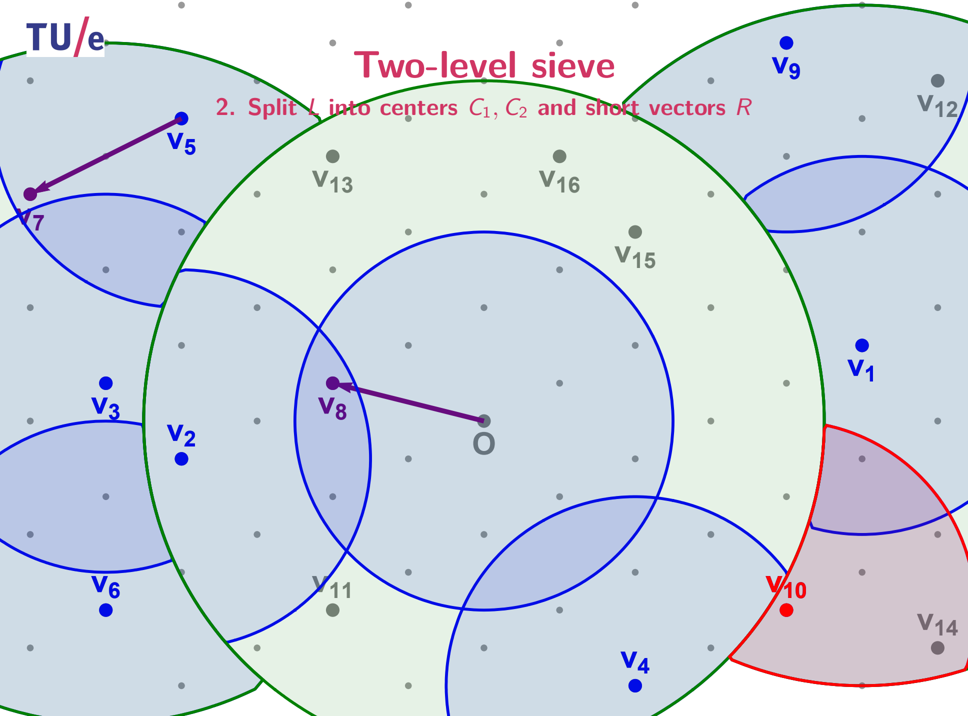
Two-level sieve

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



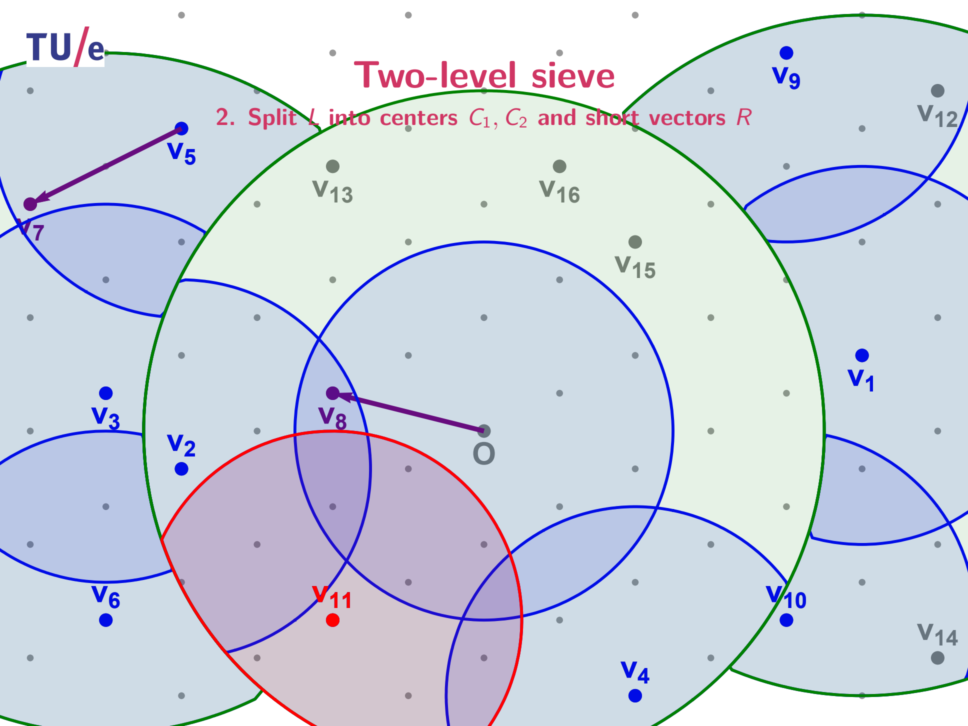
Two-level sieve

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



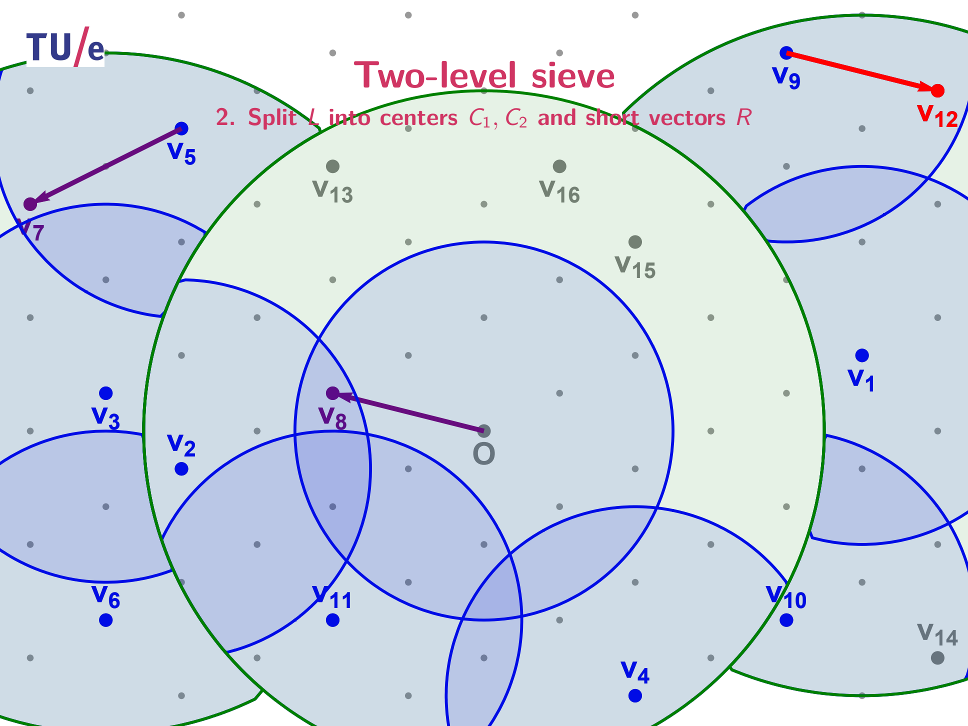
Two-level sieve

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



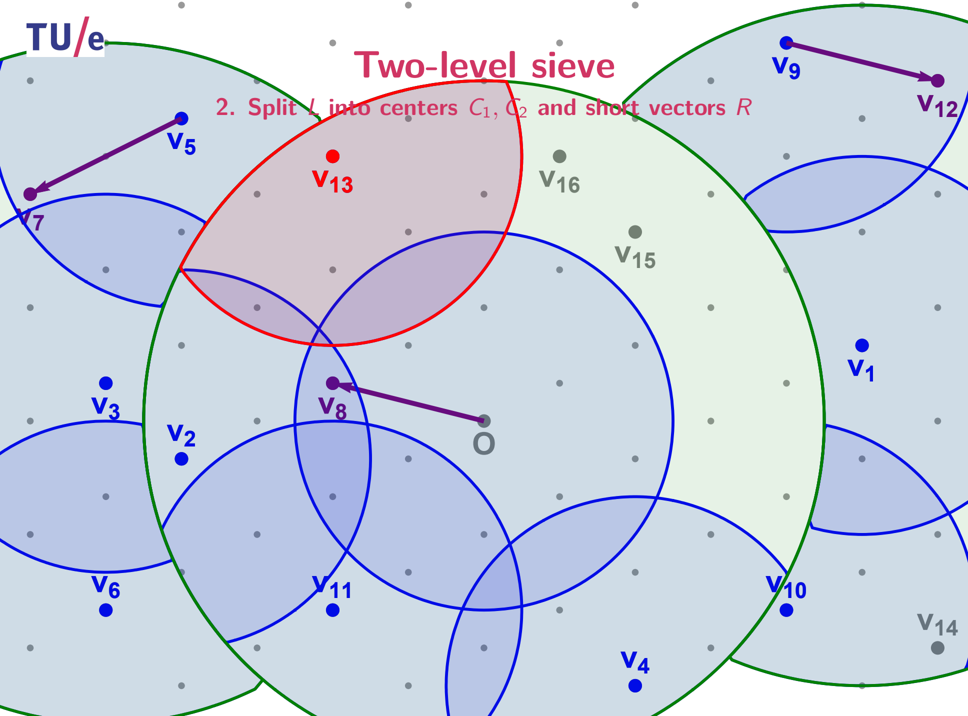
Two-level sieve

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



Two-level sieve

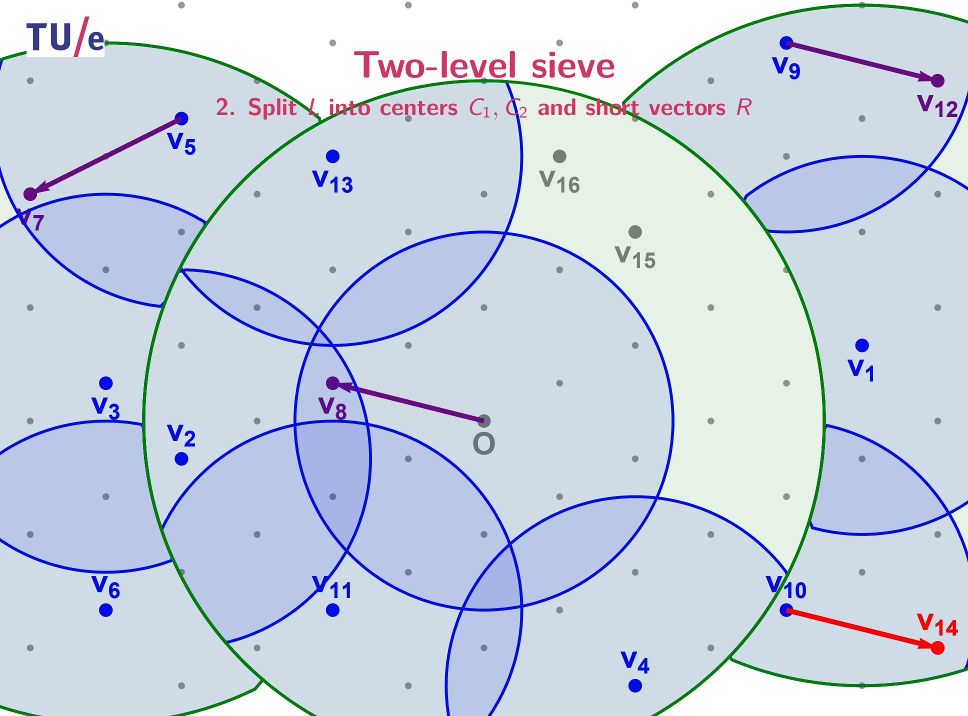
2. Split L into centers C_1, C_2 and short vectors R



TU/e

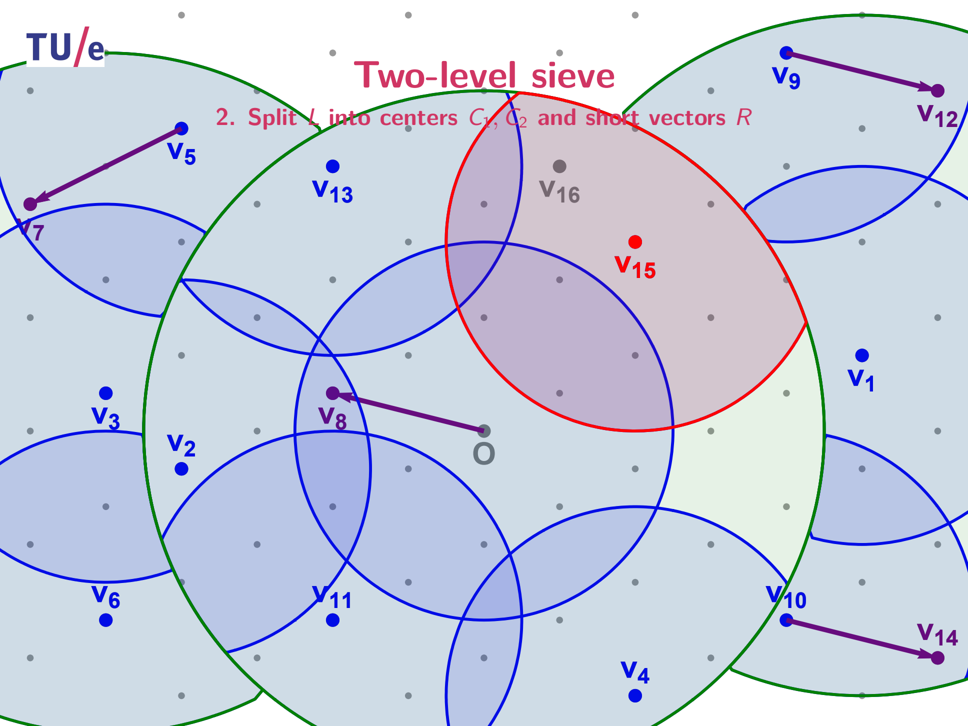
Two-level sieve

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



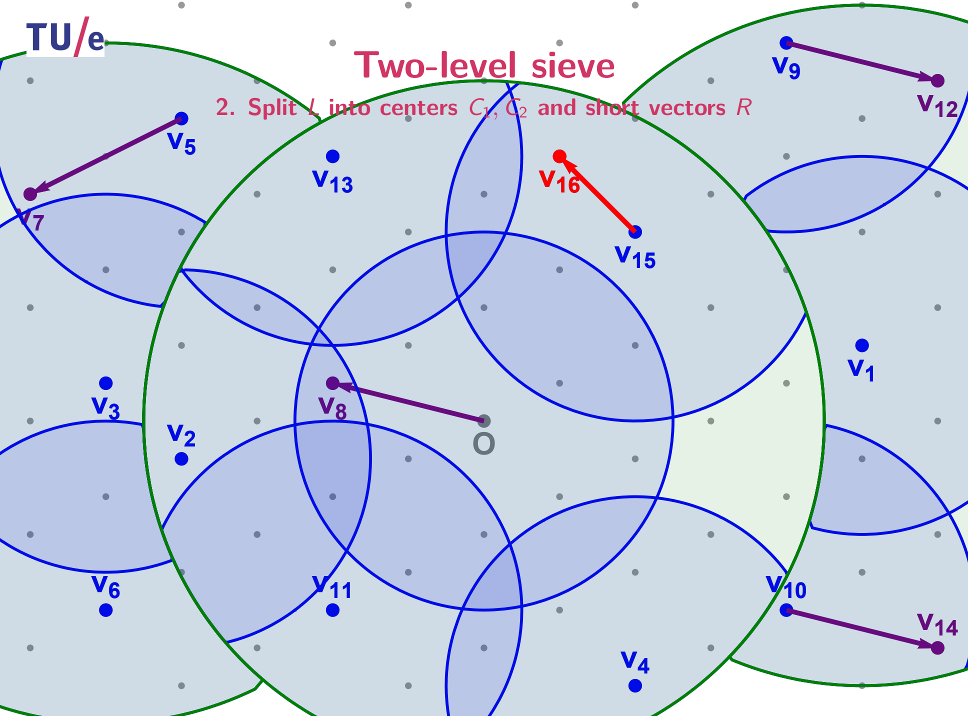
Two-level sieve

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



Two-level sieve

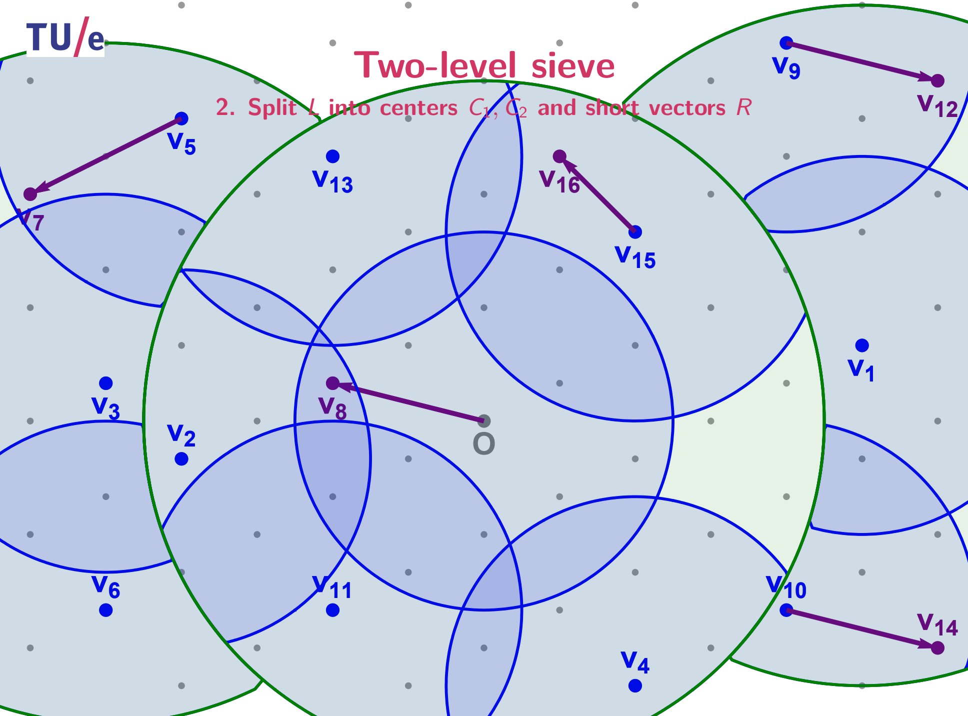
2. Split L into centers C_1, C_2 and short vectors R



TU/e

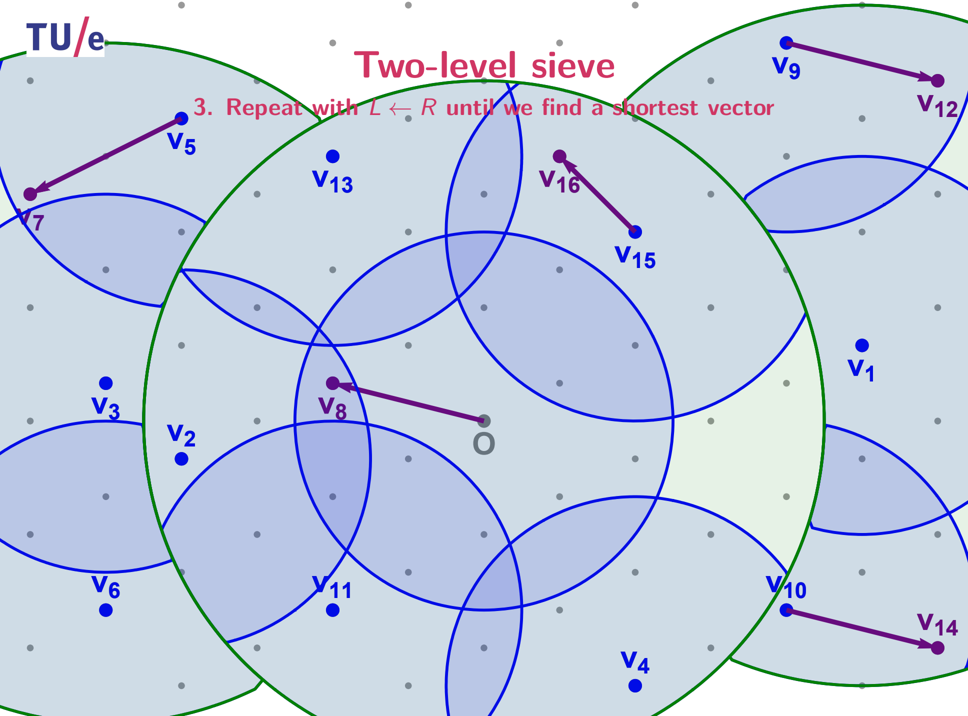
Two-level sieve

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



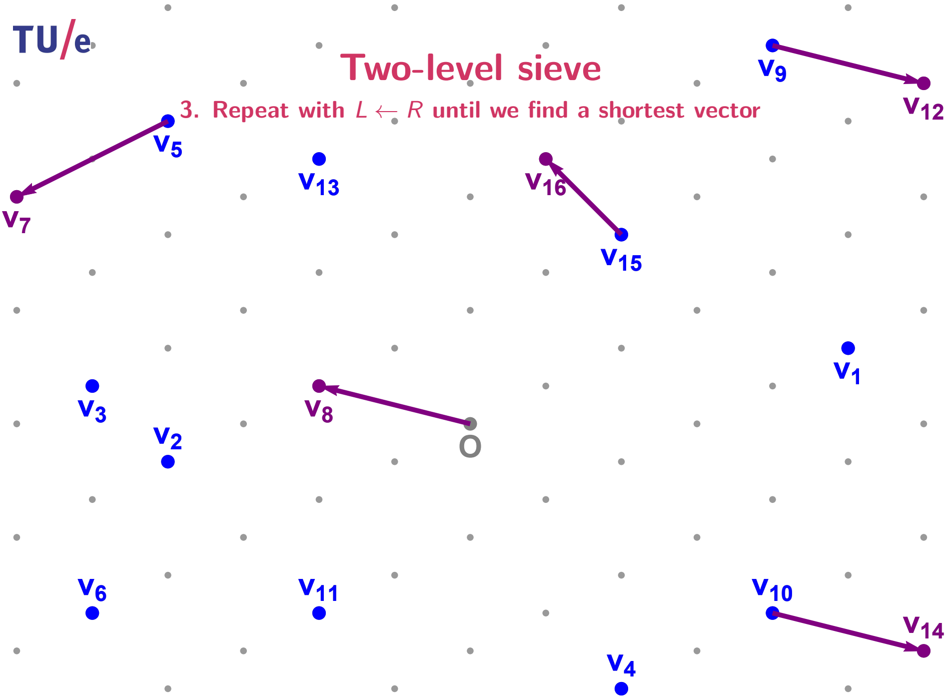
Two-level sieve

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



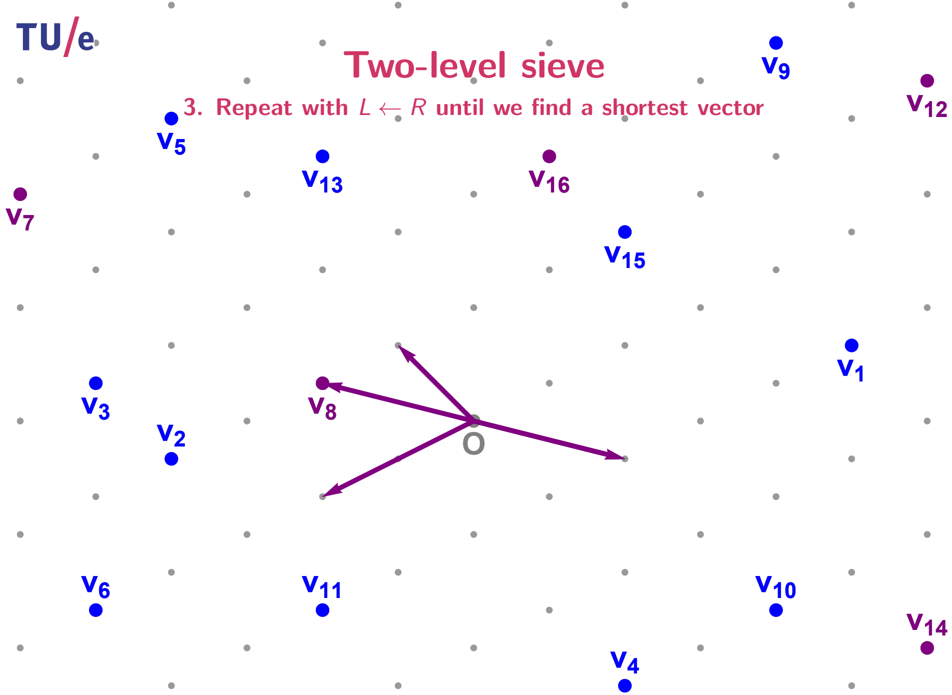
Two-level sieve

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



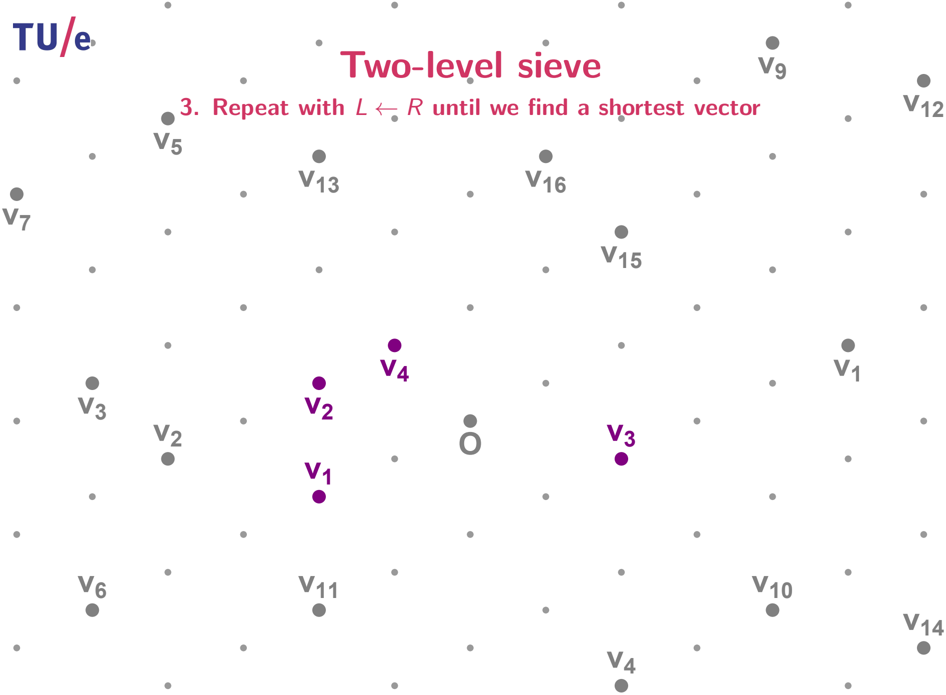
Two-level sieve

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



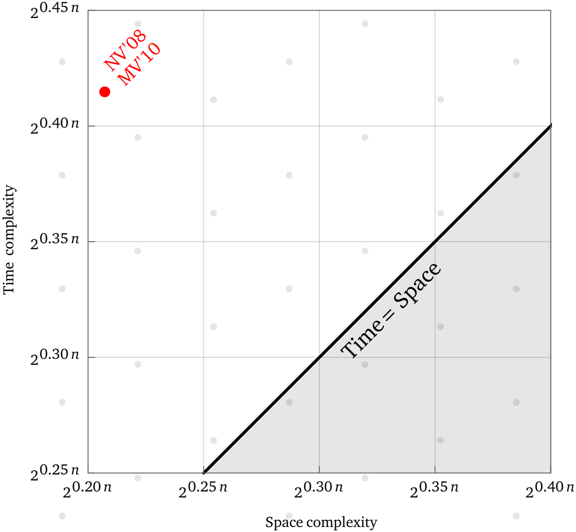
Two-level sieve

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



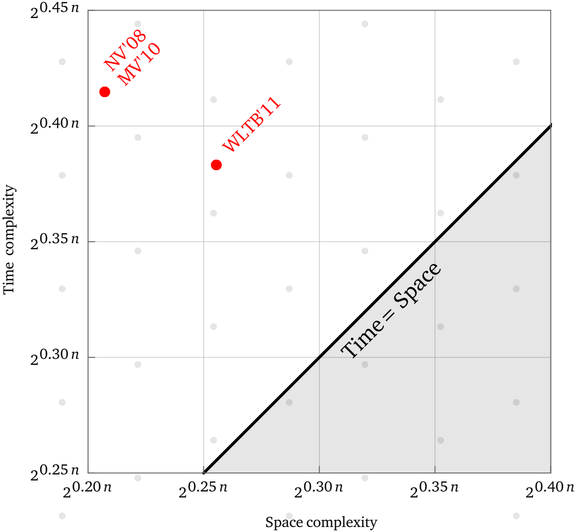
Two-level sieve

Space/time trade-off



Two-level sieve

Space/time trade-off



Three-level sieve

Overview

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

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

Three-level sieve

Overview

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

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

Theorem (Wang–Liu–Tian–Bi, ASIACCS'11)

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

Three-level sieve

Overview

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

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

Theorem (Wang–Liu–Tian–Bi, ASIACCS'11)

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

Theorem (Zhang–Pan–Hu, SAC'13)

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

Three-level sieve

Overview

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

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

Theorem (Wang–Liu–Tian–Bi, ASIACCS'11)

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

Theorem (Zhang–Pan–Hu, SAC'13)

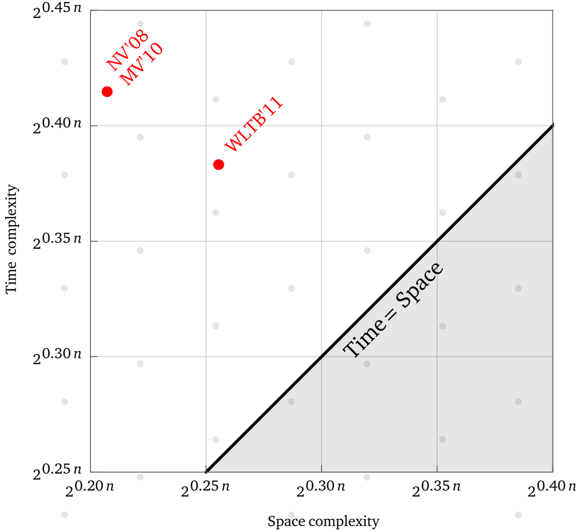
The three-level sieve runs in time $2^{0.3778n}$ and space $2^{0.2833n}$.

Conjecture (L, PhD thesis)

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

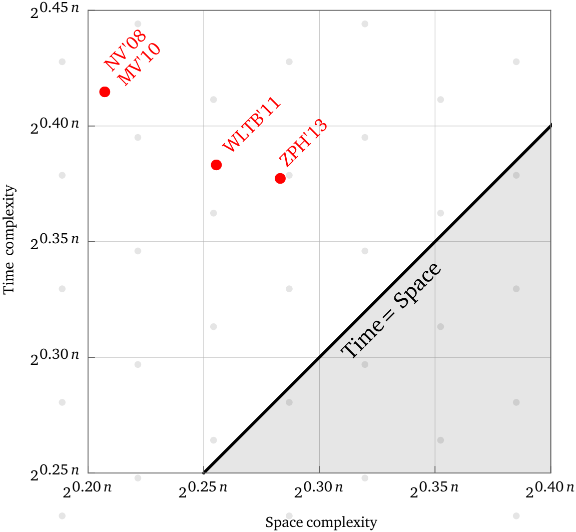
Three-level sieve

Space/time trade-off



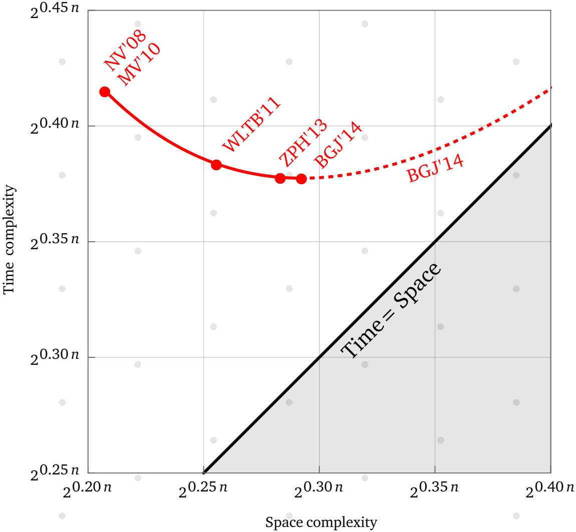
Three-level sieve

Space/time trade-off



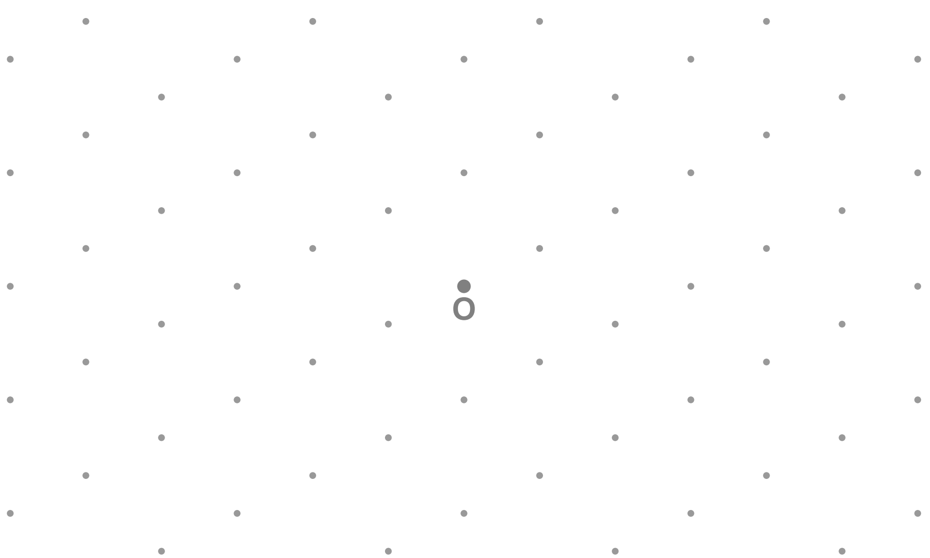
Decomposition approach

Space/time trade-off



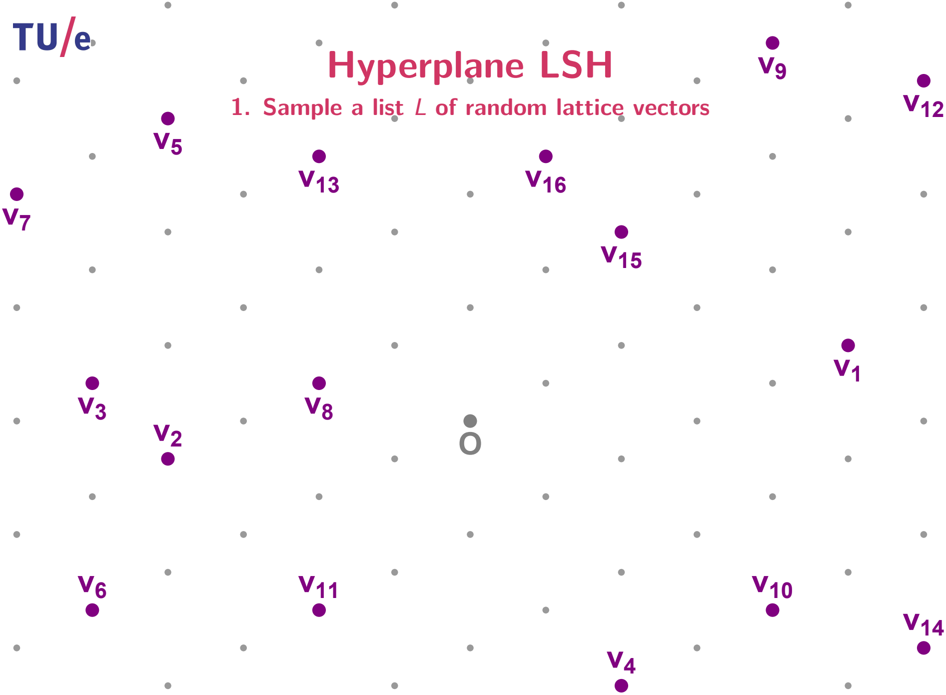
Hyperplane LSH

1. Sample a list L of random lattice vectors



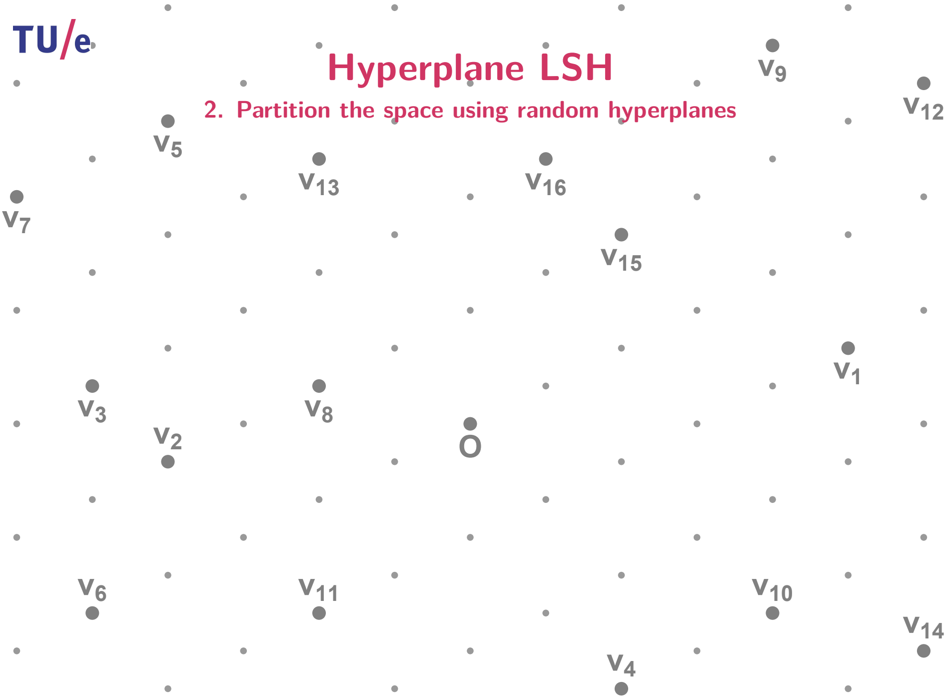
Hyperplane LSH

1. Sample a list L of random lattice vectors



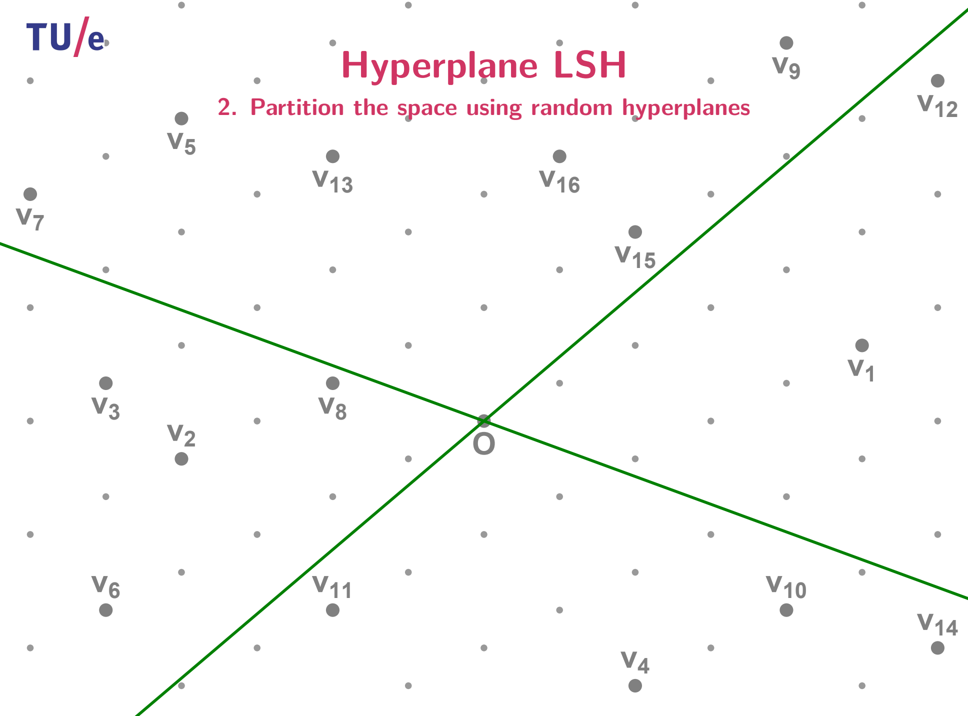
Hyperplane LSH

2. Partition the space using random hyperplanes



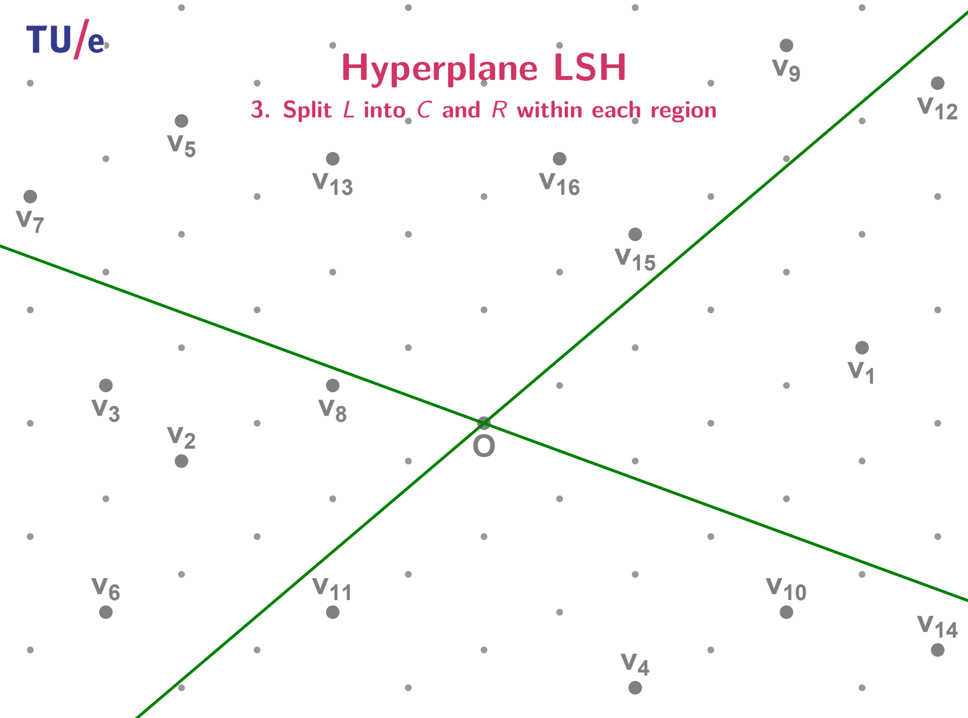
Hyperplane LSH

2. Partition the space using random hyperplanes



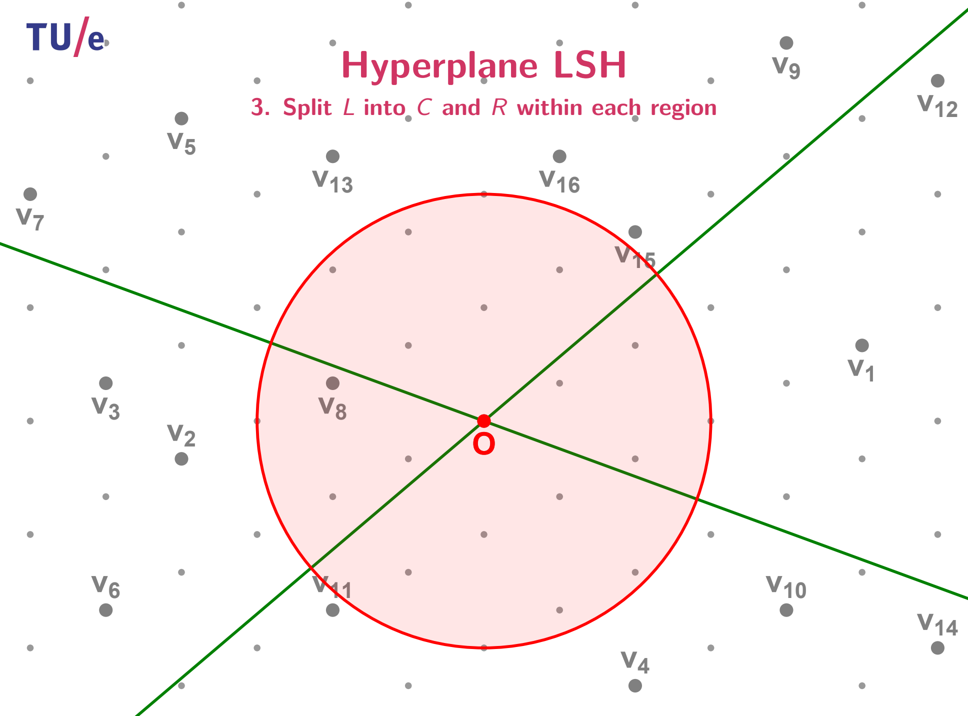
Hyperplane LSH

3. Split L into C and R within each region



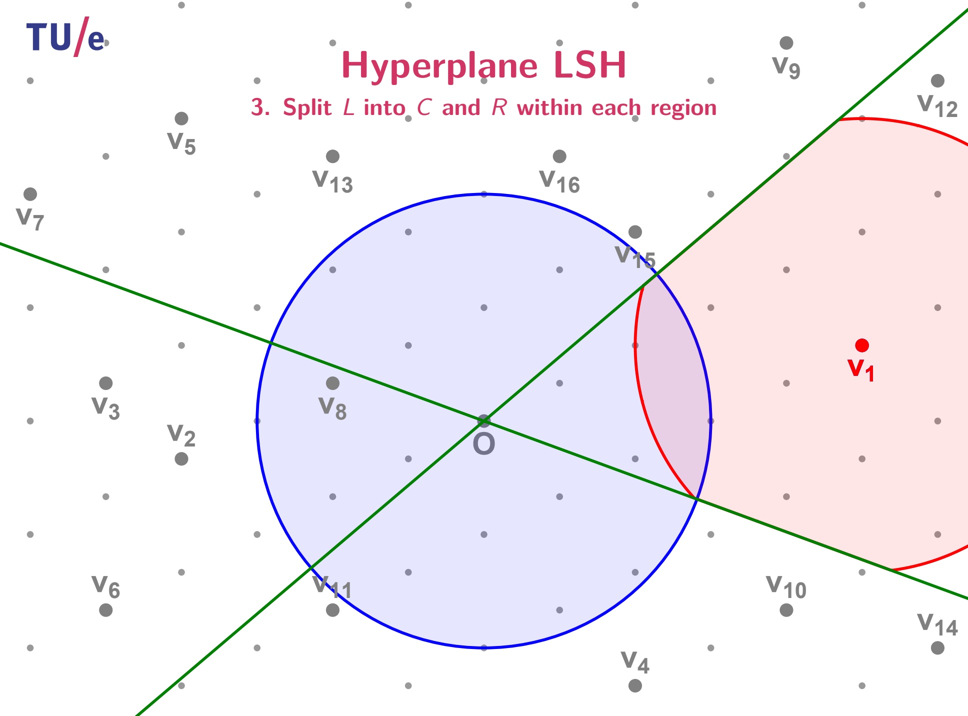
Hyperplane LSH

3. Split L into C and R within each region



Hyperplane LSH

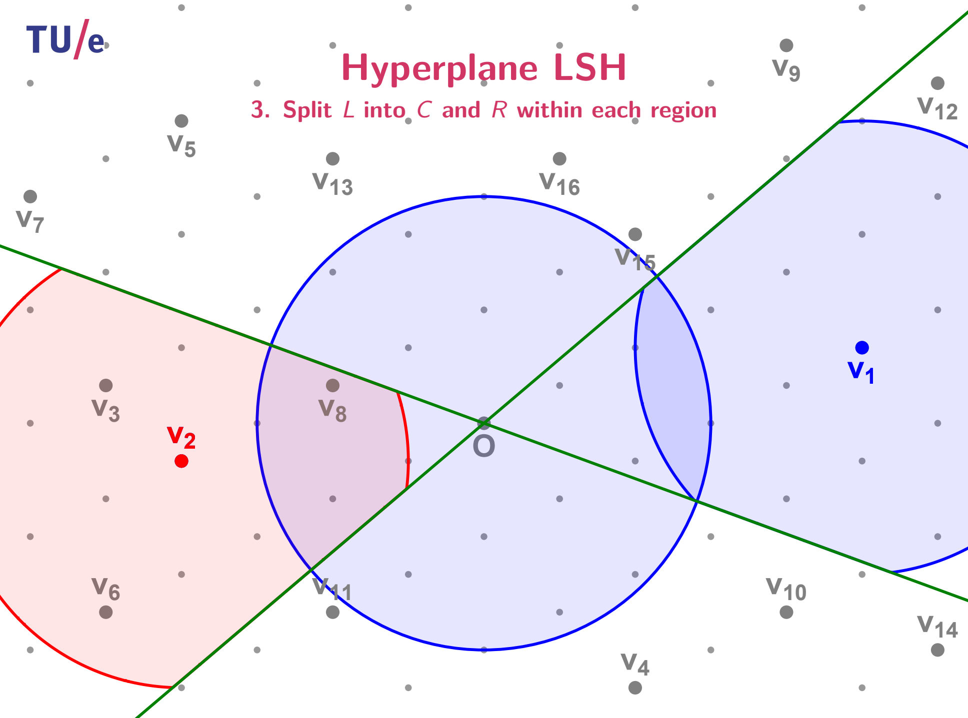
3. Split L into C and R within each region



TU/e

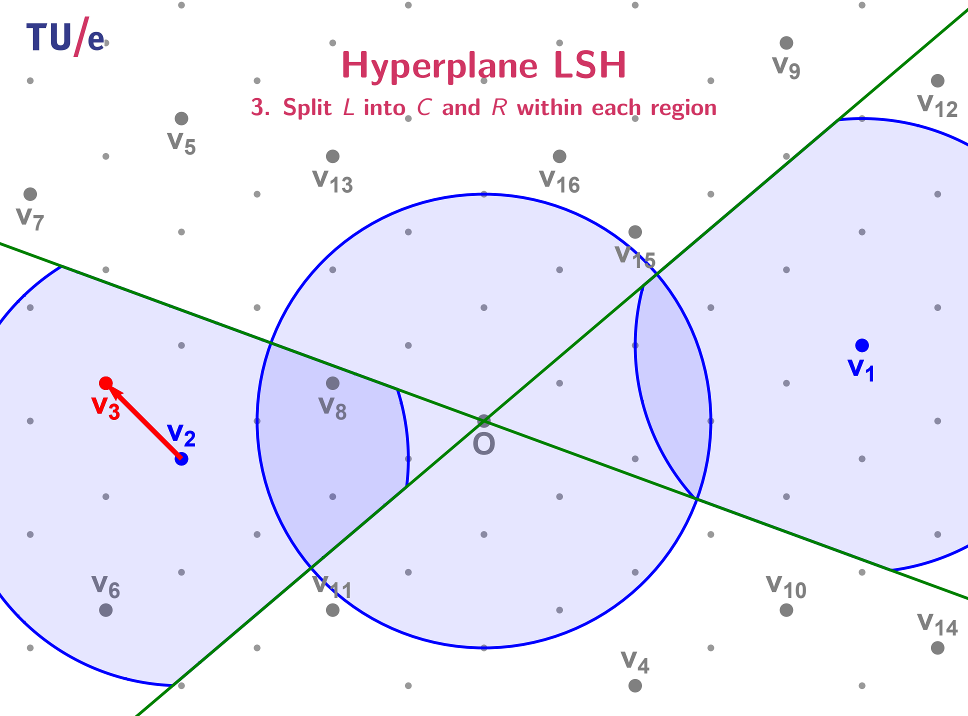
Hyperplane LSH

3. Split L into C and R within each region



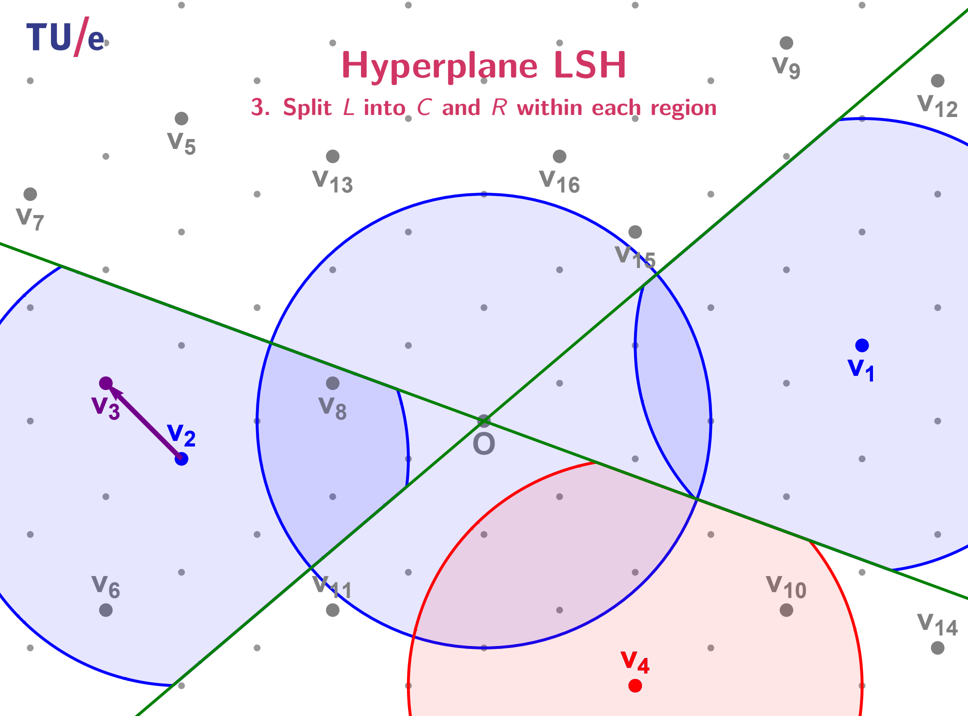
Hyperplane LSH

3. Split L into C and R within each region



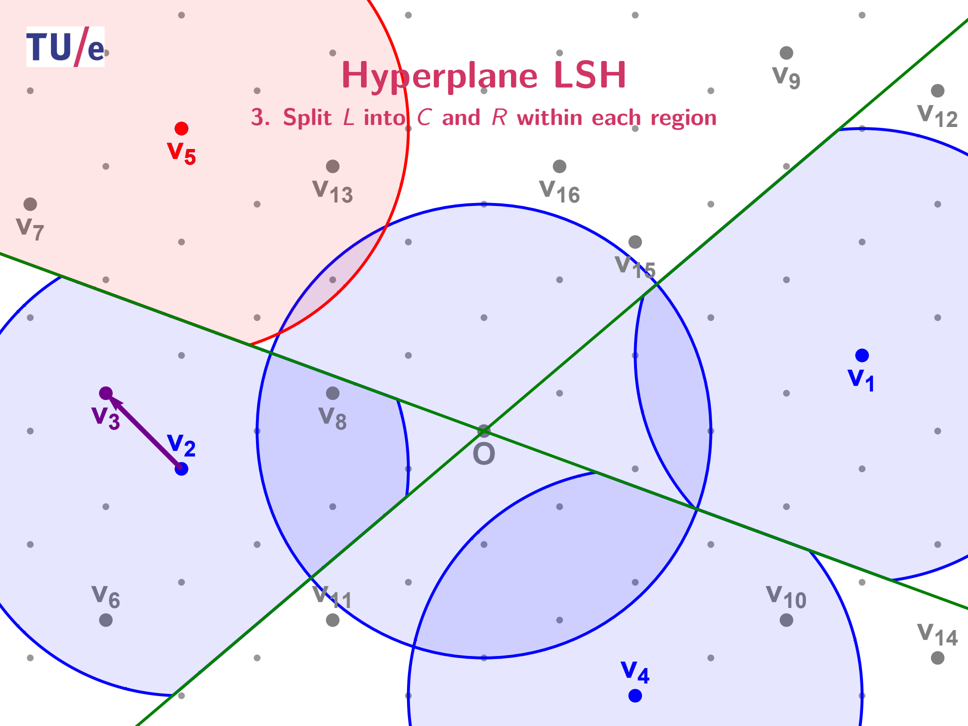
Hyperplane LSH

3. Split L into C and R within each region



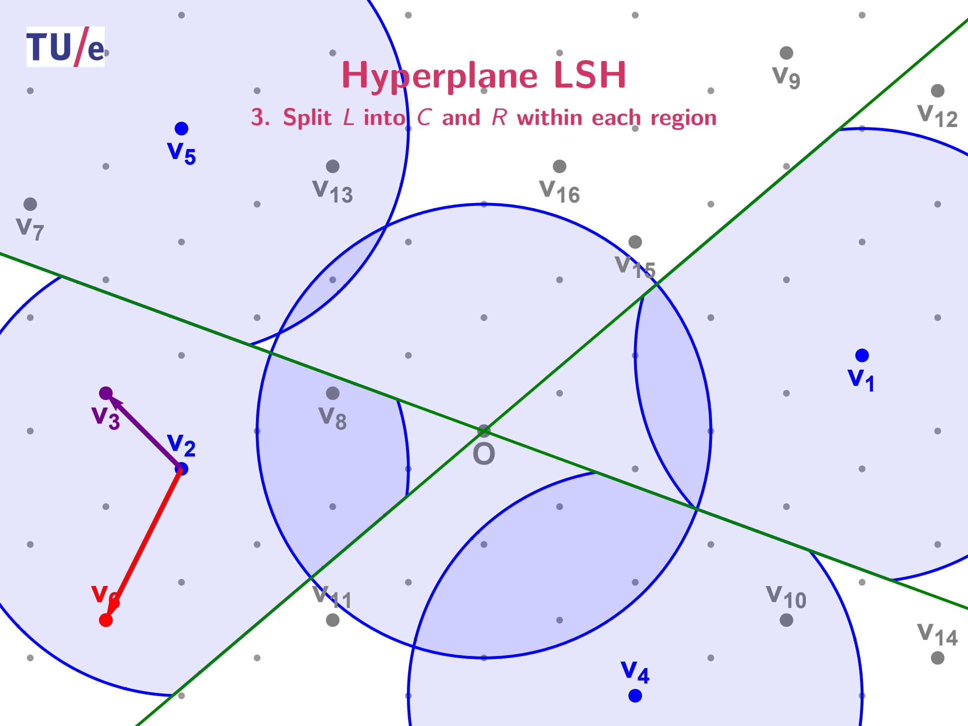
Hyperplane LSH

3. Split L into C and R within each region



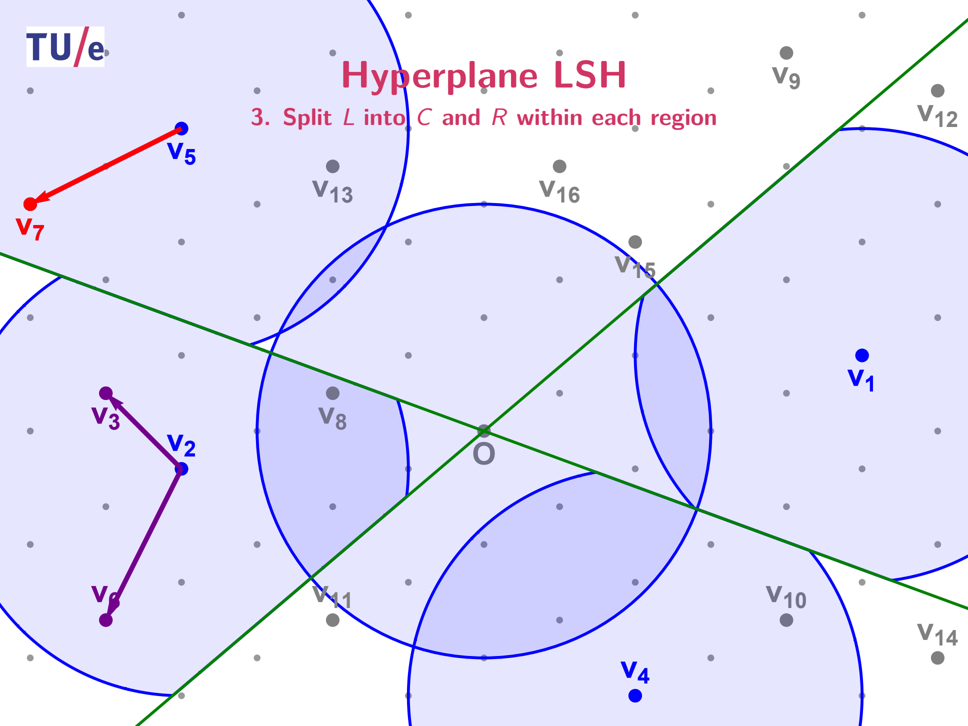
Hyperplane LSH

3. Split L into C and R within each region



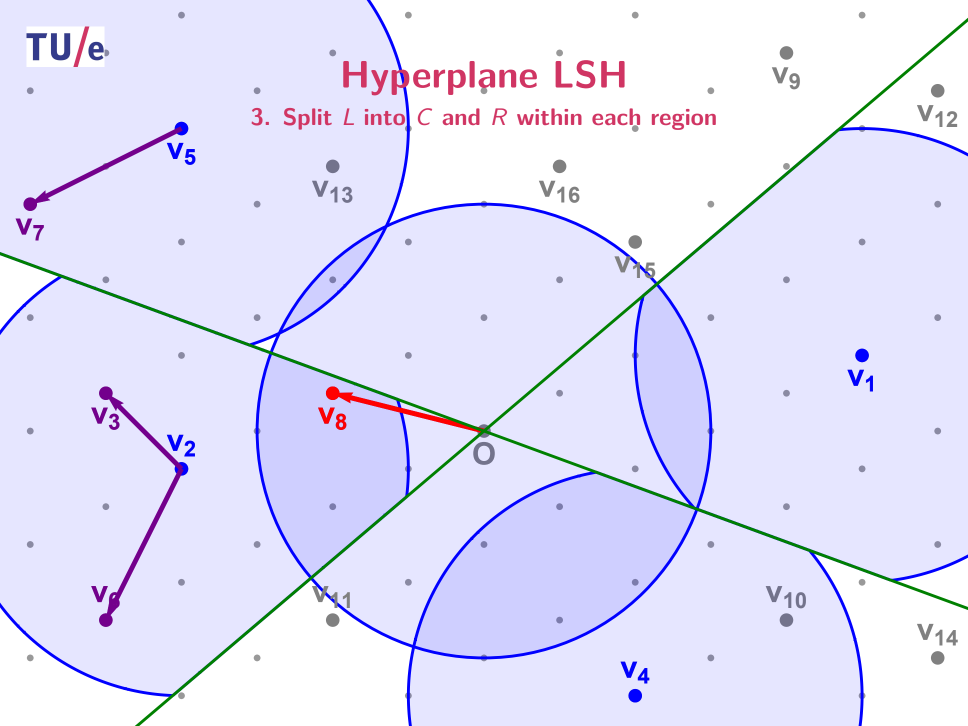
Hyperplane LSH

3. Split L into C and R within each region



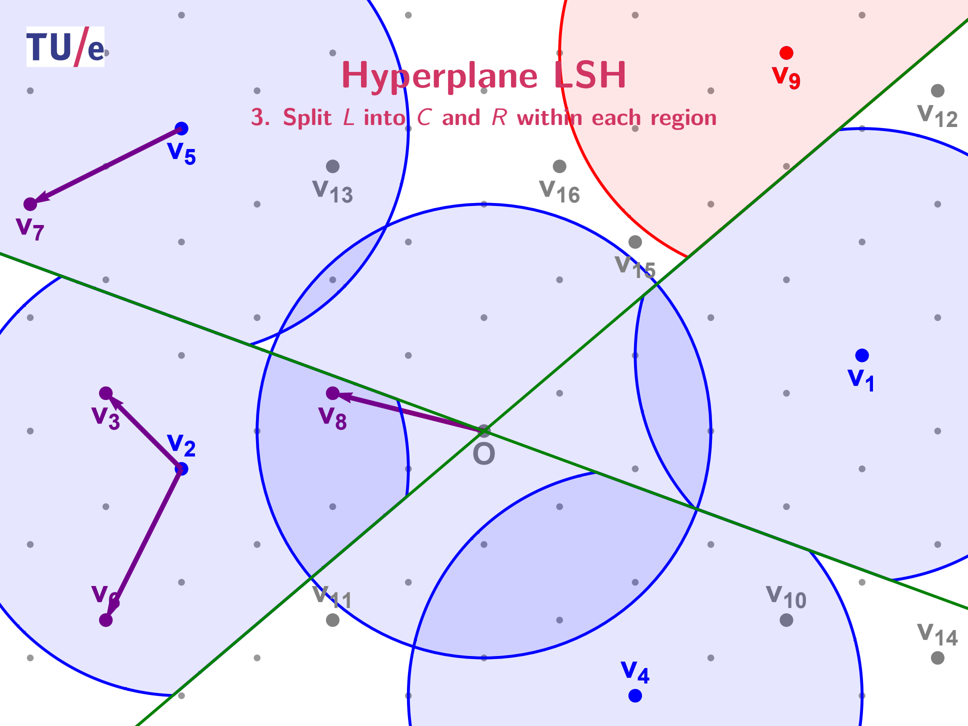
Hyperplane LSH

3. Split L into C and R within each region



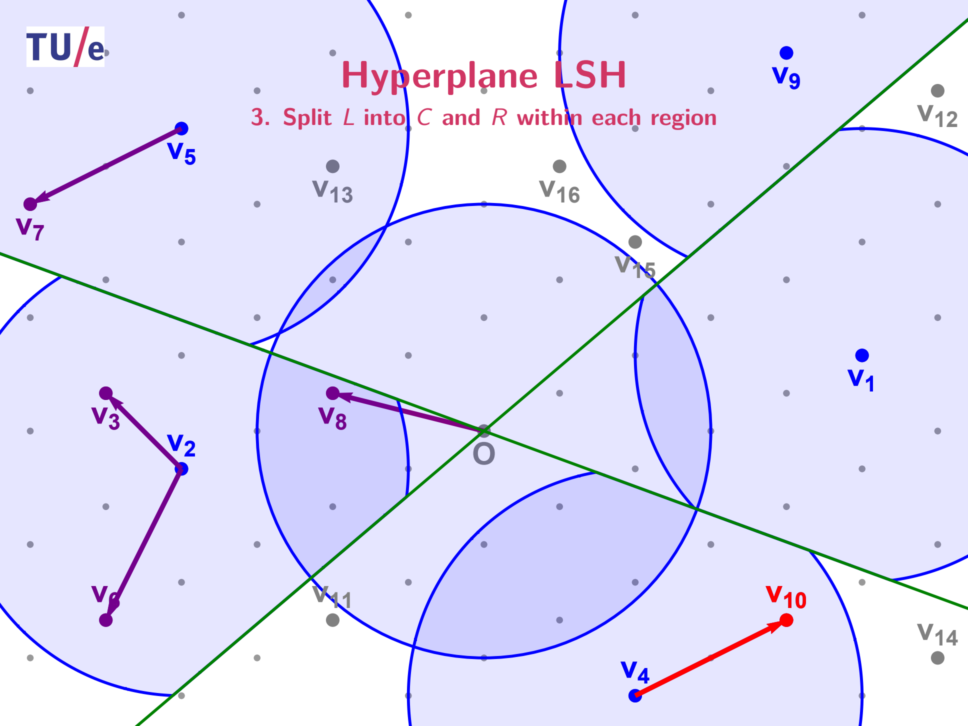
Hyperplane LSH

3. Split L into C and R within each region



Hyperplane LSH

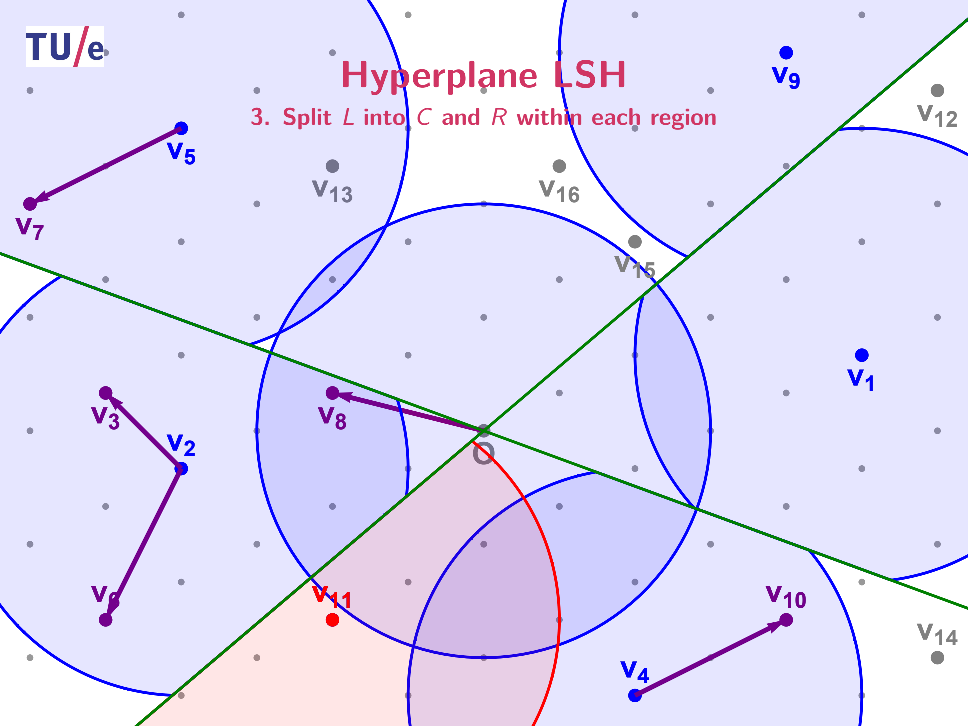
3. Split L into C and R within each region



TU/e

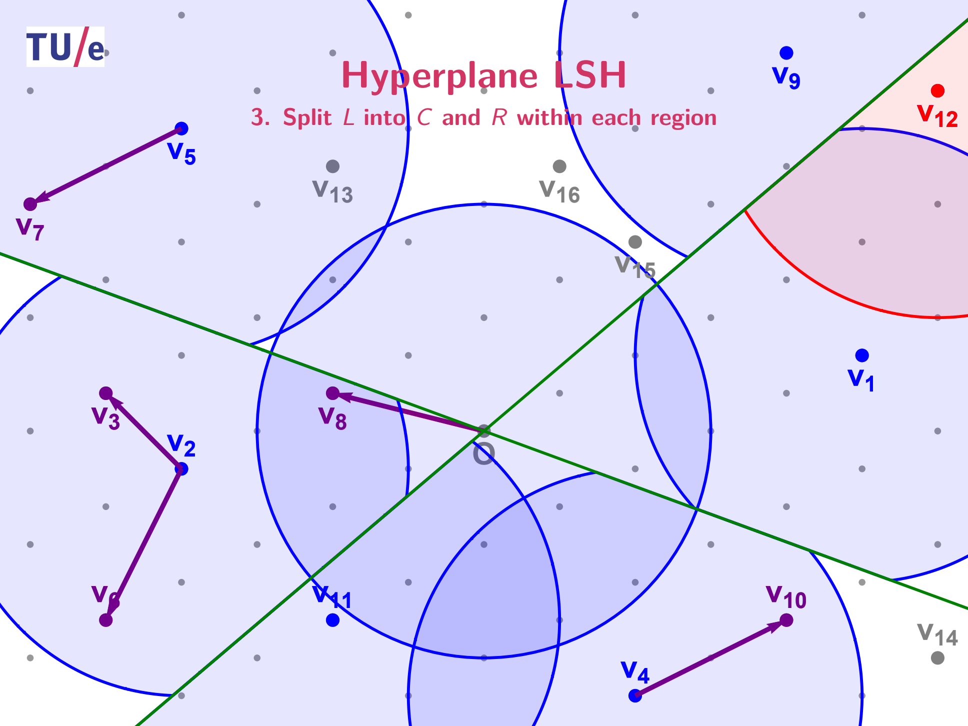
Hyperplane LSH

3. Split L into C and R within each region



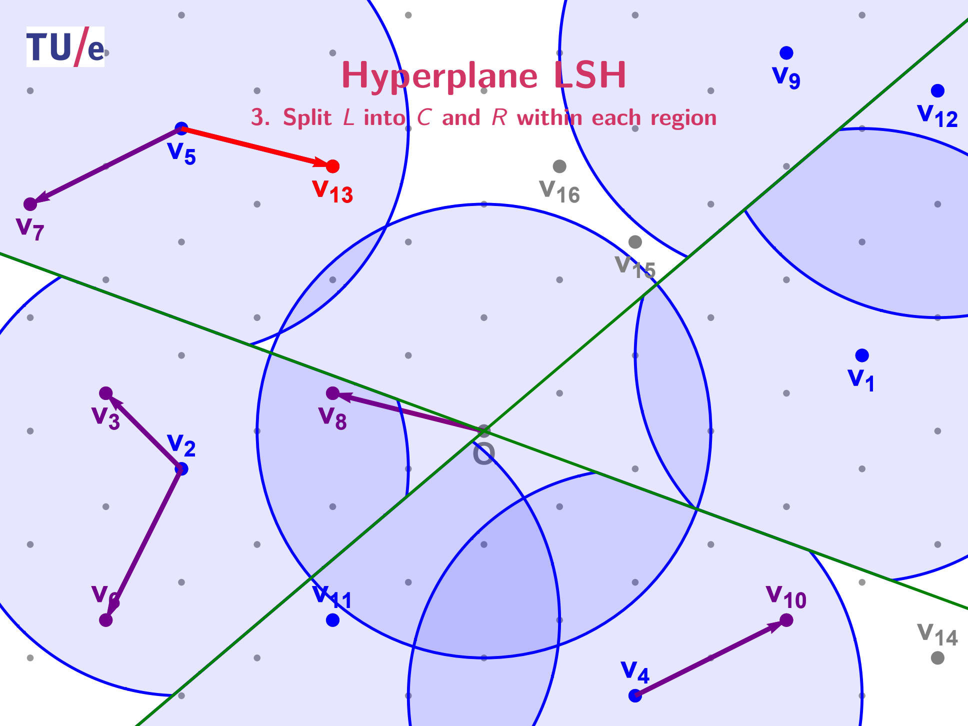
Hyperplane LSH

3. Split L into C and R within each region



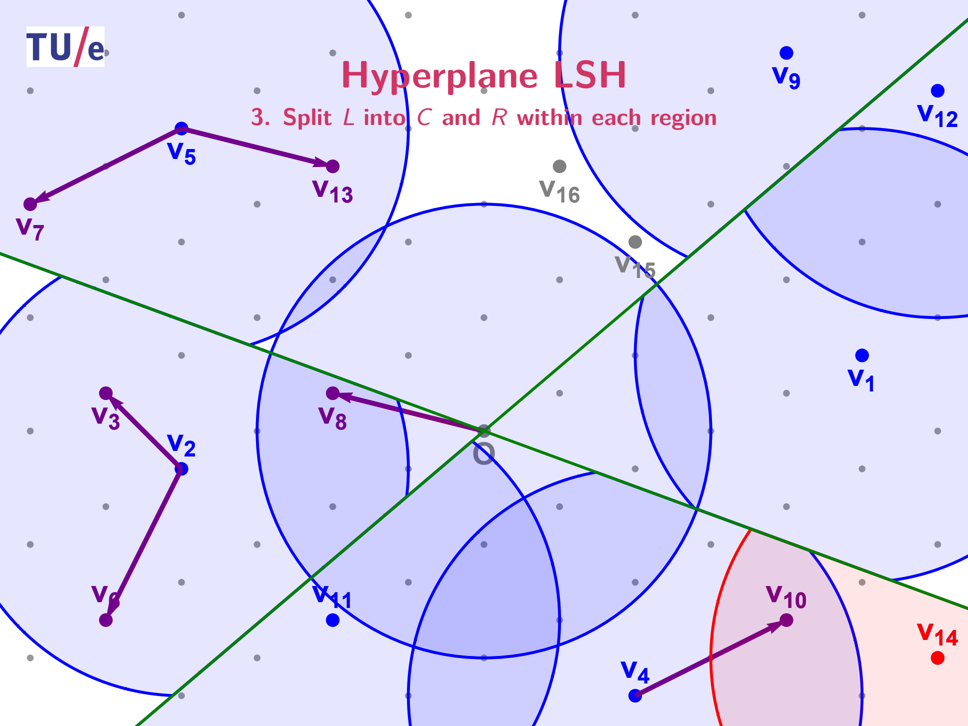
Hyperplane LSH

3. Split L into C and R within each region



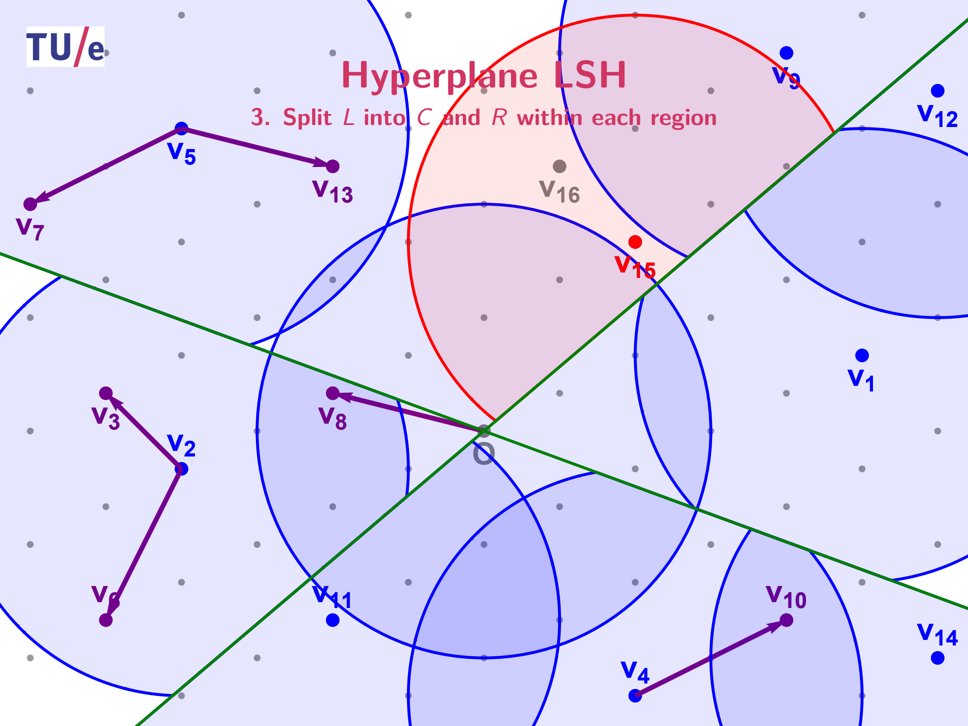
Hyperplane LSH

3. Split L into C and R within each region



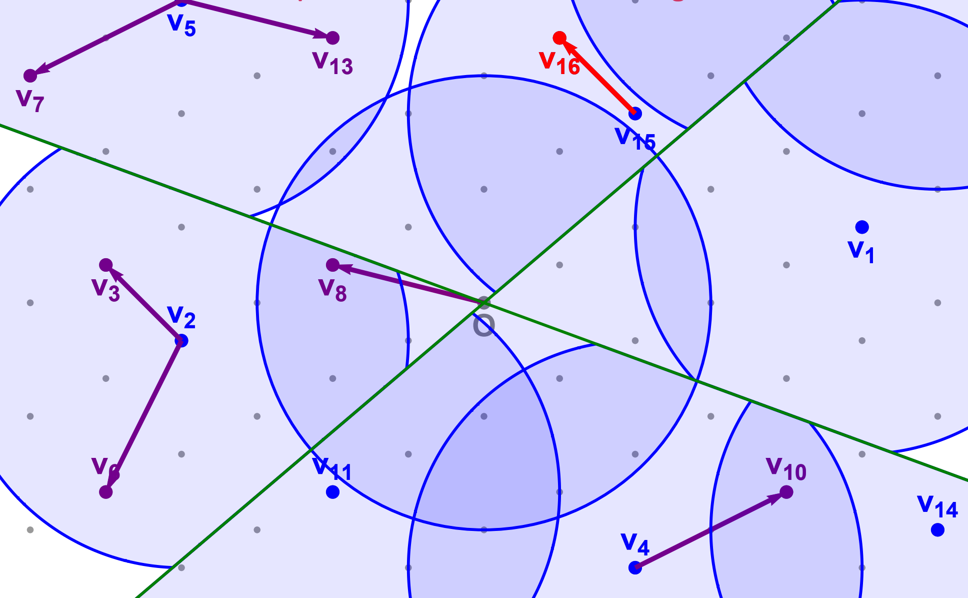
Hyperplane LSH

3. Split L into C and R within each region



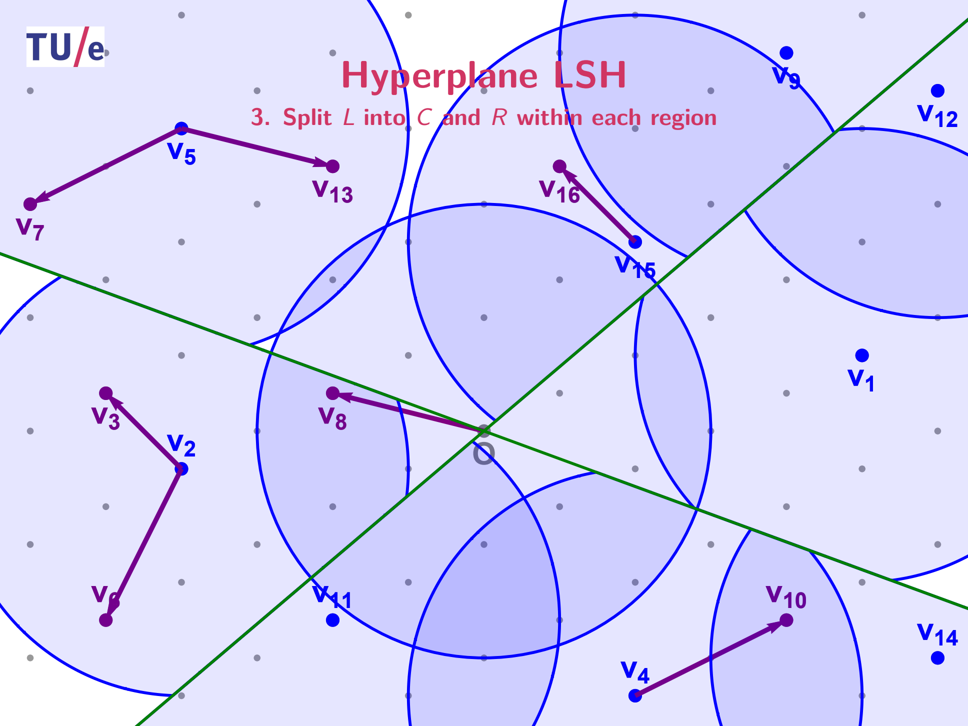
Hyperplane LSH

3. Split L into C and R within each region



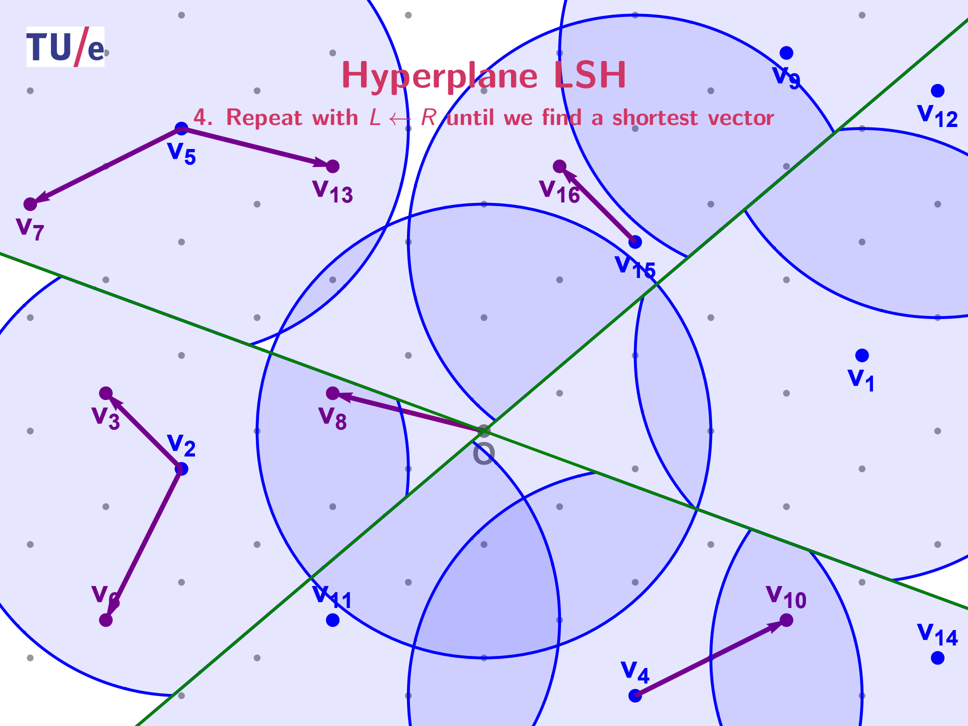
Hyperplane LSH

3. Split L into C and R within each region



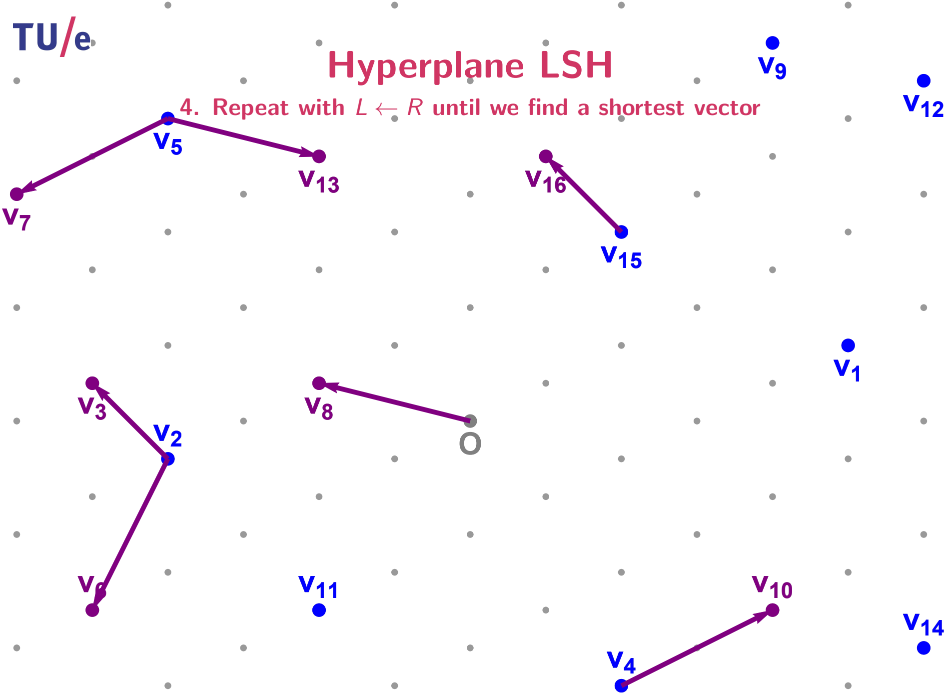
Hyperplane LSH

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



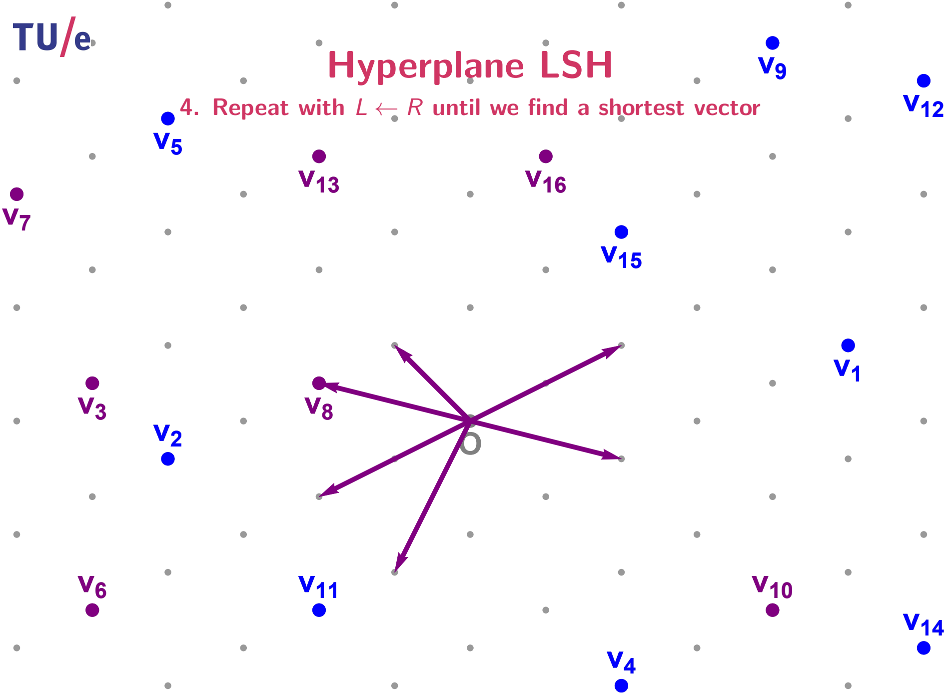
Hyperplane LSH

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



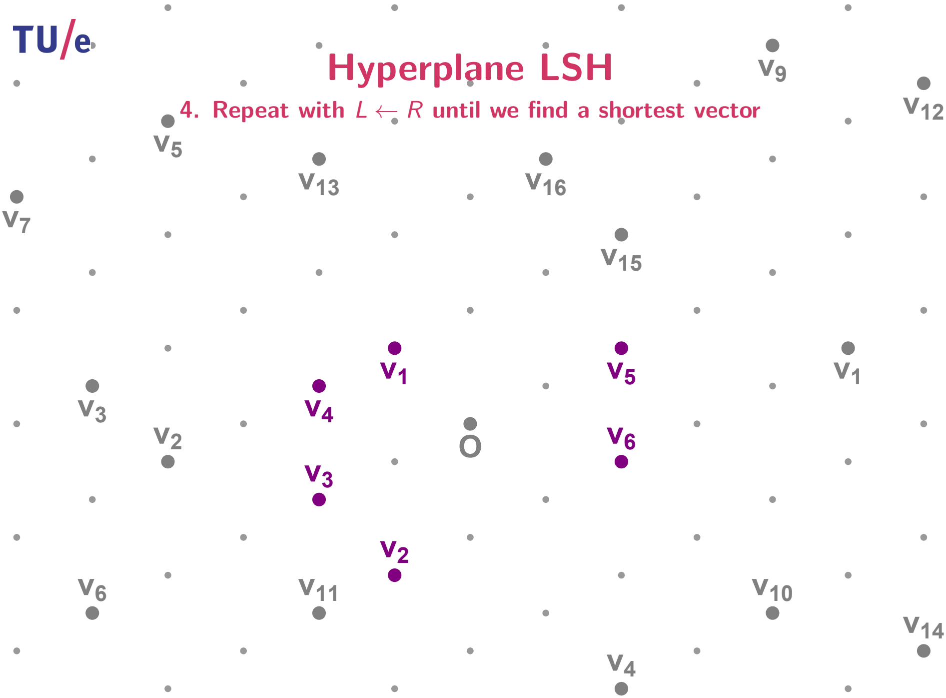
Hyperplane LSH

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

Overview

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"

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

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: $2^{0.208n+o(n)}$

Hyperplane LSH

Overview

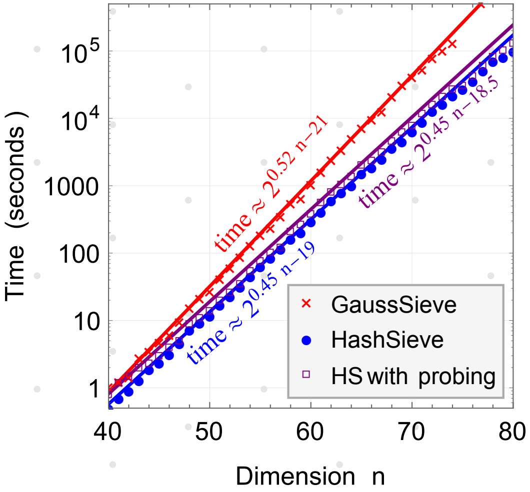
- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent "hash tables"
- Space complexity: $2^{0.337n+o(n)}$
  - Number of vectors: $2^{0.208n+o(n)}$
  - Number of hash tables: $2^{0.129n+o(n)}$
  - Each hash table contains 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 (L, CRYPTO'15)

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

Hyperplane LSH

Experimental results



Hyperplane LSH

Experimental results [MLB15]

Dimension	90	92	94	96	98	100
Hyperplanes (k)	20	20	21	21	22	22
Hash tables (t)	3126	3738	4470	465	533	637
Probing?	N	N	N	Y	Y	Y
BKZ-β preproc.	34	34	34	34	40	40
Used vectors ($\cdot 10^6$)	2.43	3.00	4.50	5.57	7.05	10.05
Time (hours)	0.86	1.72	3.74	6.52	10.03	18.19
Memory (GB)	310	380	872	95	113	256

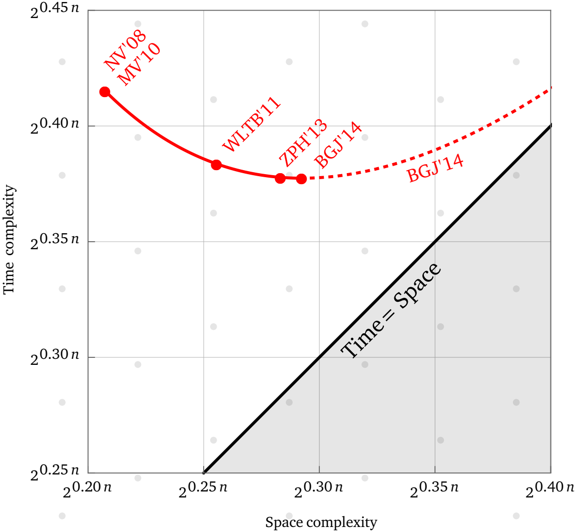
Hyperplane LSH

Experimental results [MLB15]

22	118	2782	0	Kenji Kashiwabara and Masaharu Fukase	vec	Other	2013-08-12	1.00811
23	118	2868	8	Yuanmi Chen and Phong Nguyen	vec	ENUM	2013-02-13	1.04441
24	116	2743	0	Thorsten Kleinjung	vec	Sieving	2014-05-2	1.00492
25	116	2764	0	Kenji Kashiwabara and Masaharu Fukase	vec	Other	2014-03-21	1.01287
26	116	2786	0	Kenji Kashiwabara and Masaharu Fukase	vec	Other	2013-08-3	1.02075
40	108	2508	0	Yuanmi Chen and Phong Nguyen	vec	Other	2010-06-16	0.95162
41	108	2755	0	Yuanmi Chen and Phong Nguyen	vec	Other	2010-05-30	1.04519
42	107	2626	0	Artur Mariano	vec	Sieving	2015-05-22	1.00056
43	107	2713	0	Artur Mariano	vec	Sieving	2015-05-22	1.03379
44	107	2716	0	Artur Mariano	vec	Sieving	2015-05-22	1.03490
45	107	2719	0	Artur Mariano	vec	Sieving	2015-05-22	1.03609
46	107	2720	0	Artur Mariano	vec	Sieving	2015-05-22	1.03657
47	107	2721	0	Artur Mariano	vec	Sieving	2015-05-22	1.03688
48	107	2722	0	Artur Mariano	vec	Sieving	2015-05-22	1.03721
49	107	2724	8	Po-Chun Kuo, Michael Schneider	vec	ENUM,BKZ	2011-03-12	1.03655

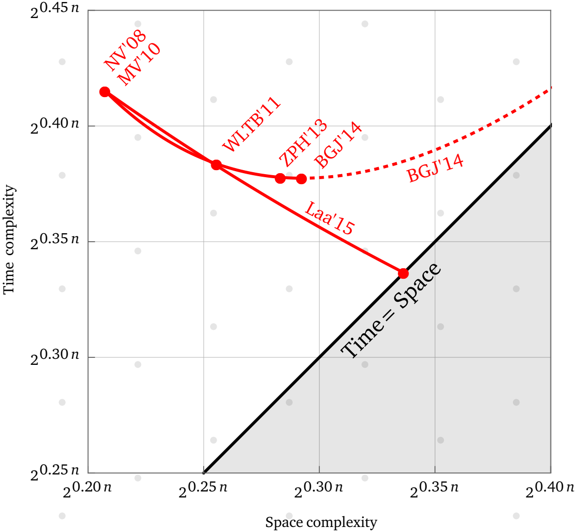
Hyperplane LSH

Space/time trade-off



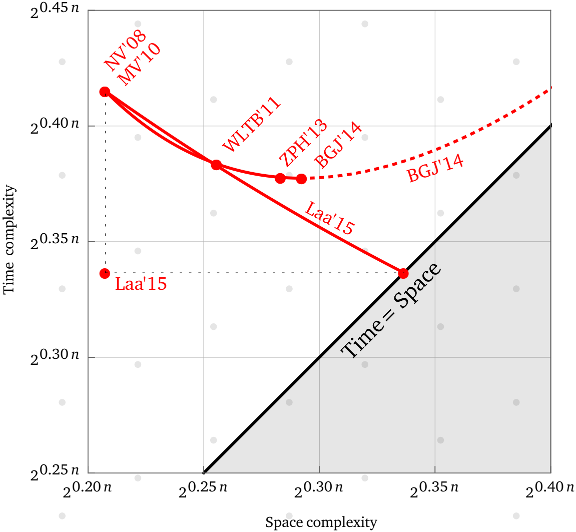
Hyperplane LSH

Space/time trade-off



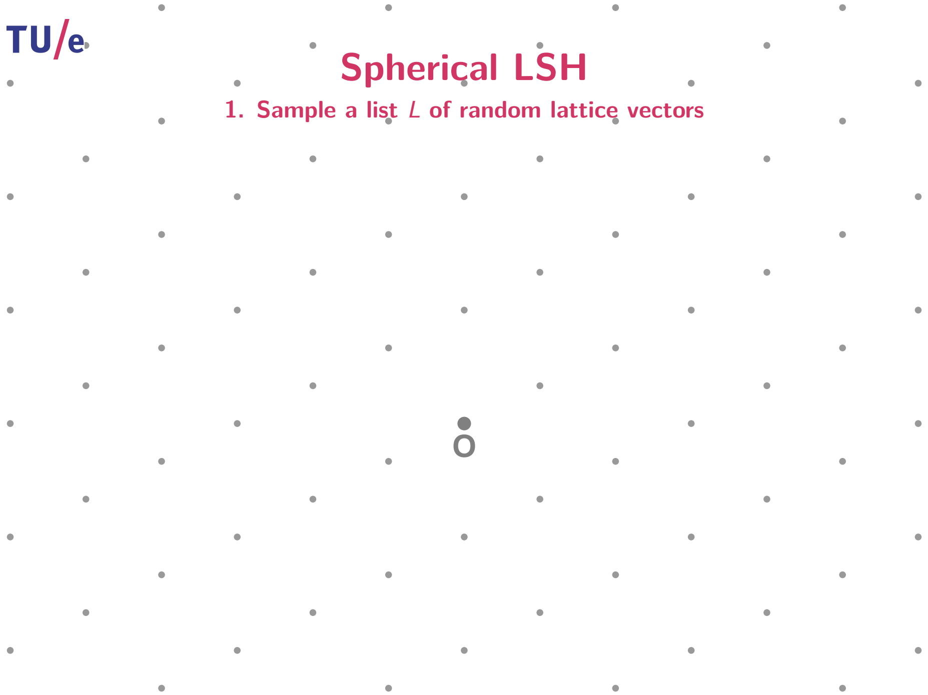
Hyperplane LSH

Space/time trade-off



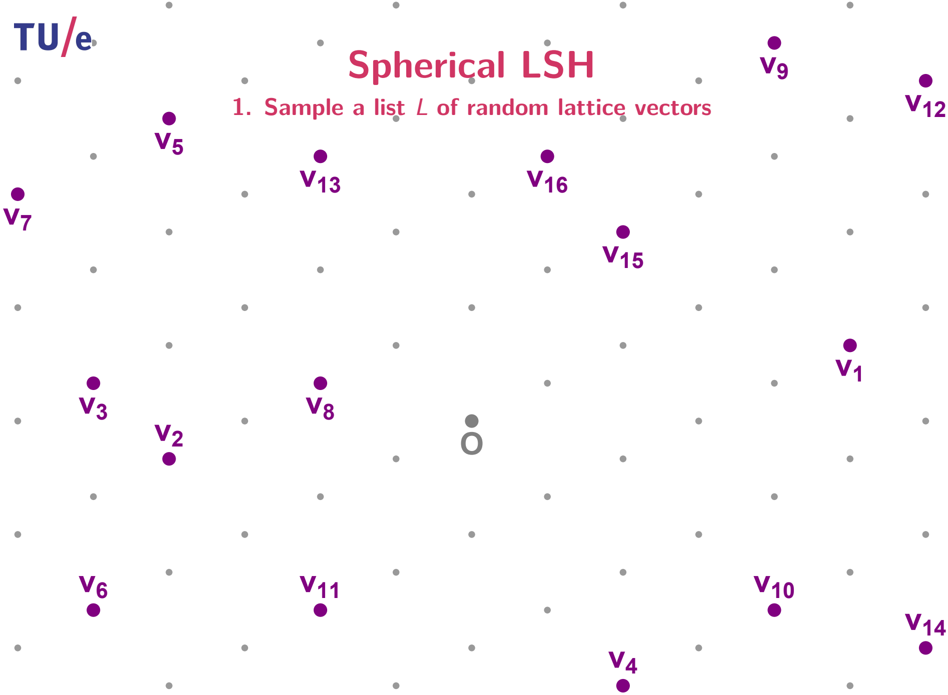
Spherical LSH

1. Sample a list L of random lattice vectors



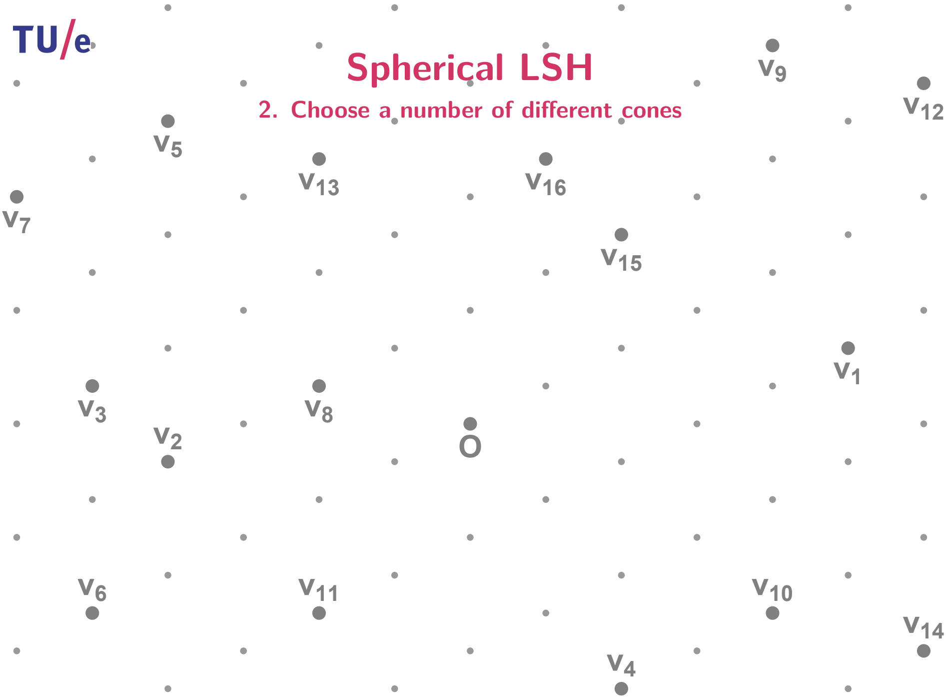
Spherical LSH

1. Sample a list L of random lattice vectors



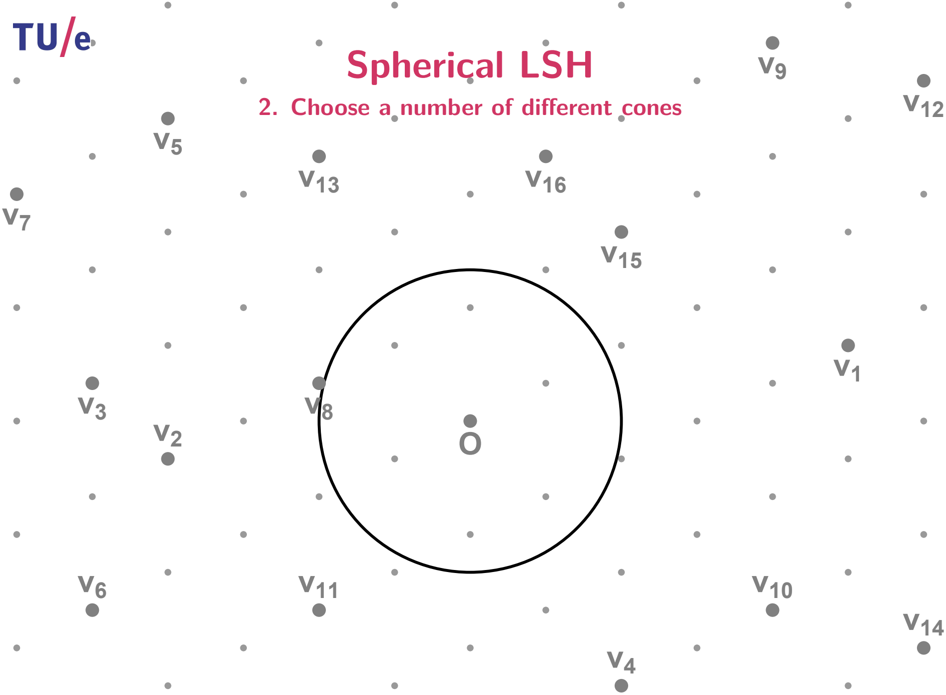
Spherical LSH

2. Choose a number of different cones



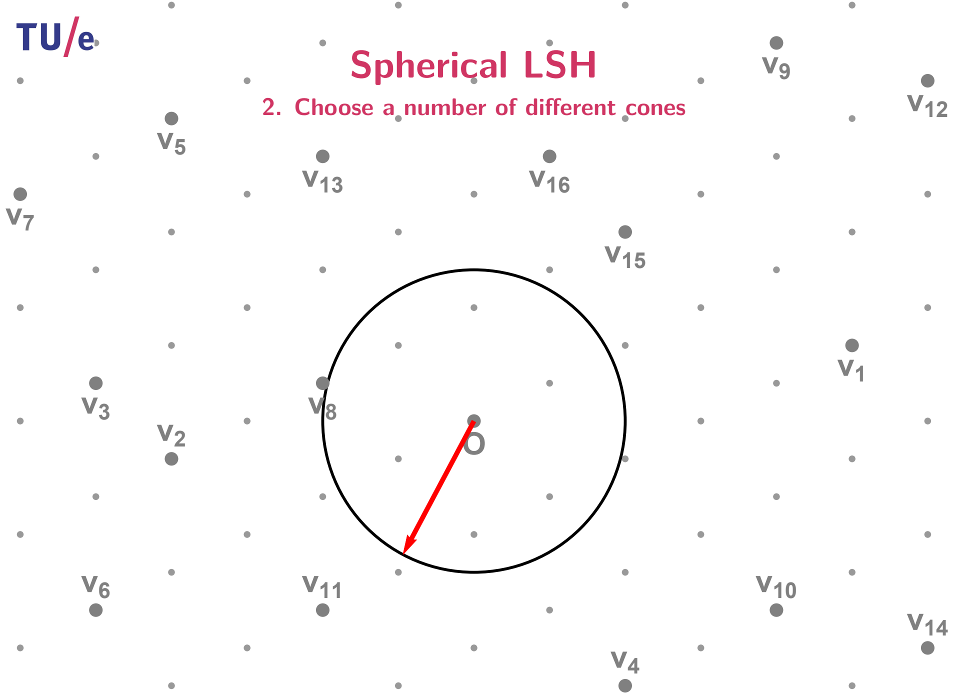
Spherical LSH

2. Choose a number of different cones



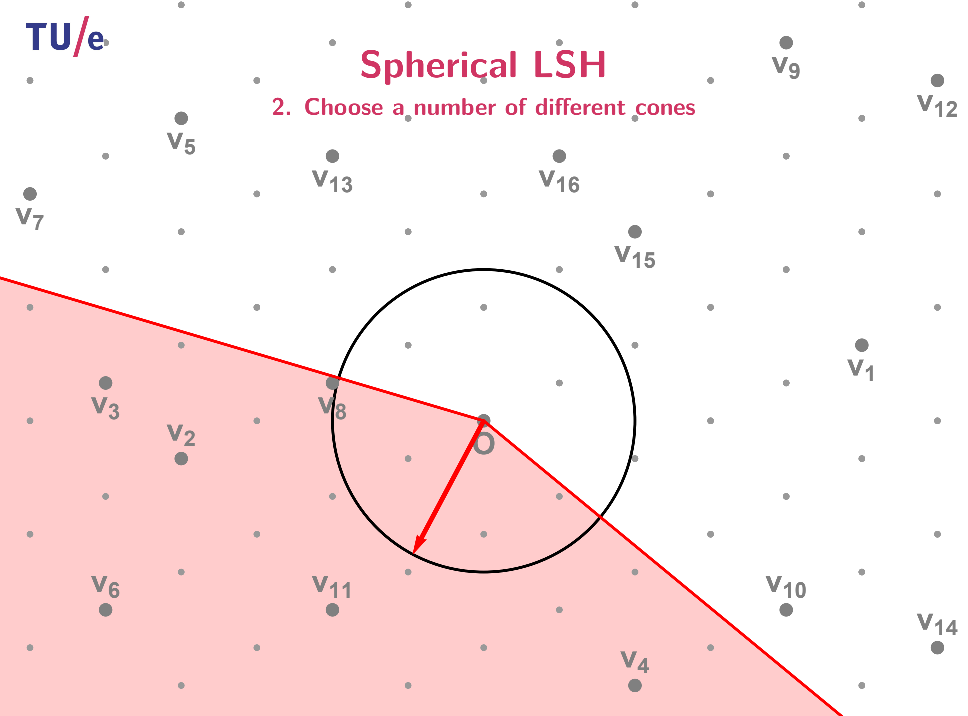
Spherical LSH

2. Choose a number of different cones



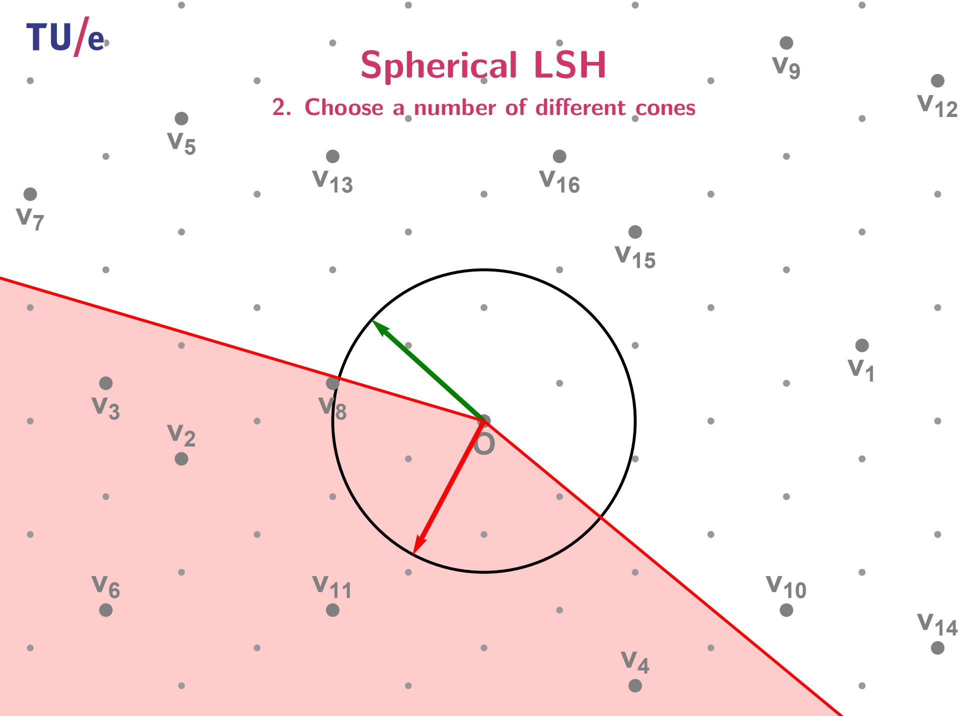
Spherical LSH

2. Choose a number of different cones



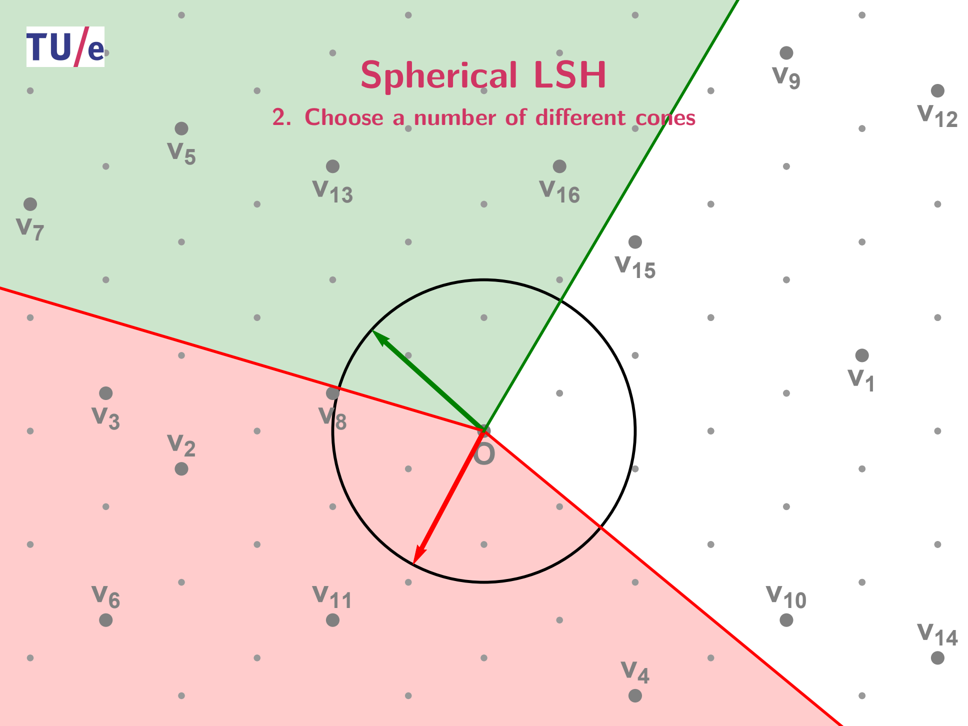
Spherical LSH

2. Choose a number of different cones



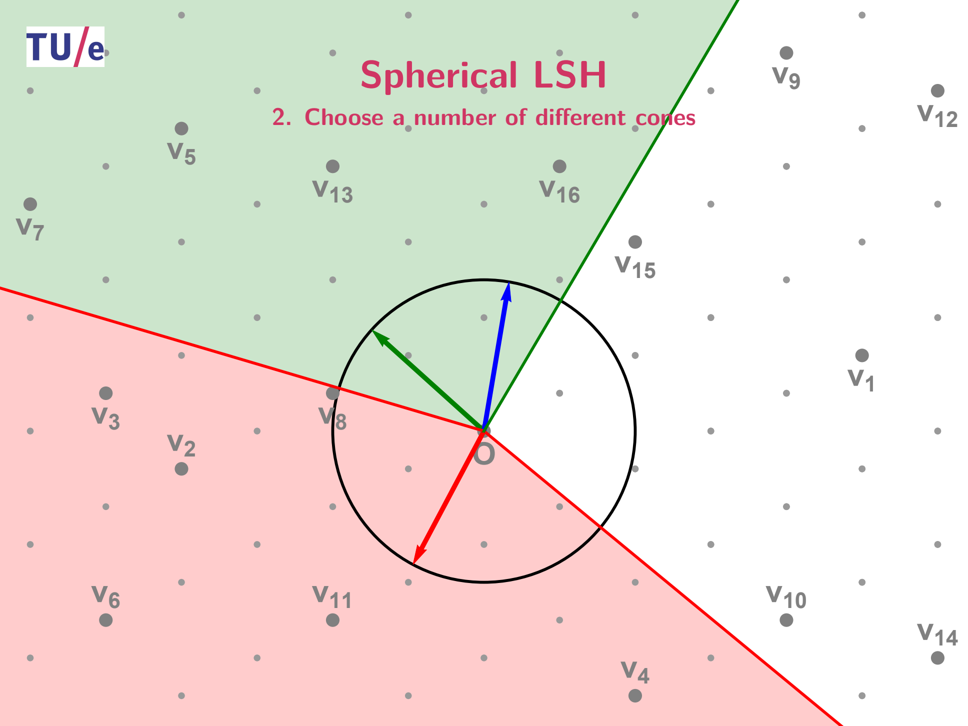
Spherical LSH

2. Choose a number of different cones



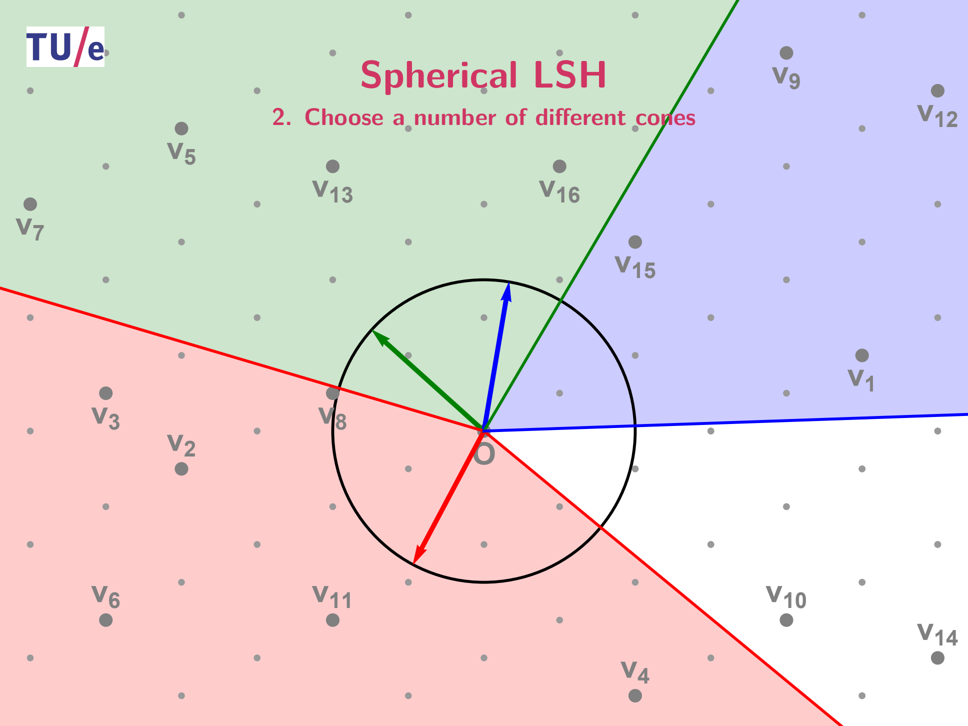
Spherical LSH

2. Choose a number of different cones



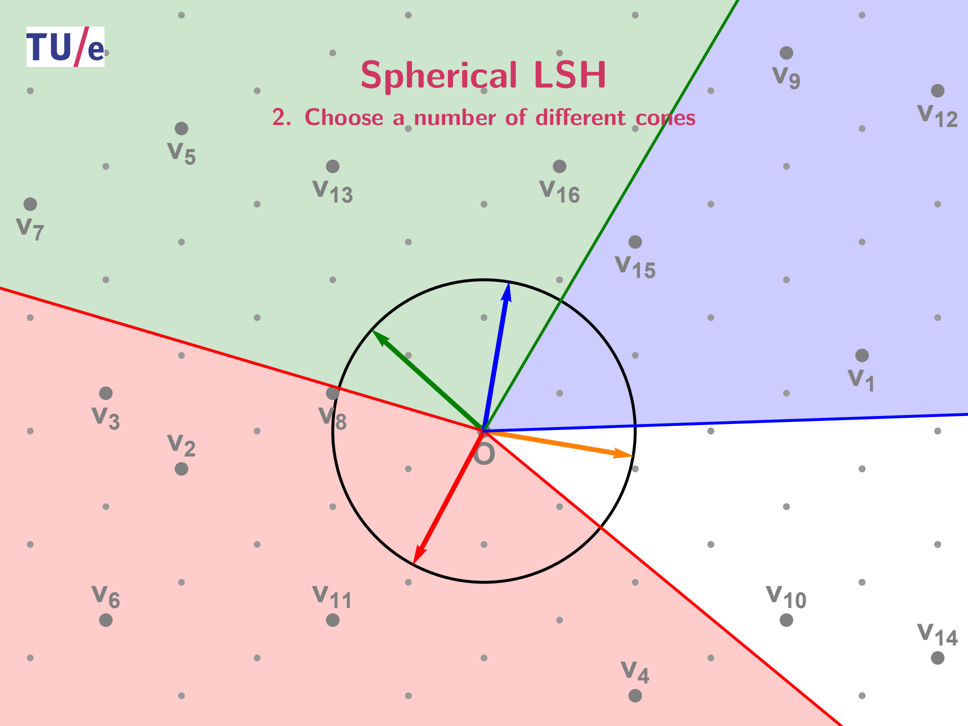
Spherical LSH

2. Choose a number of different cones



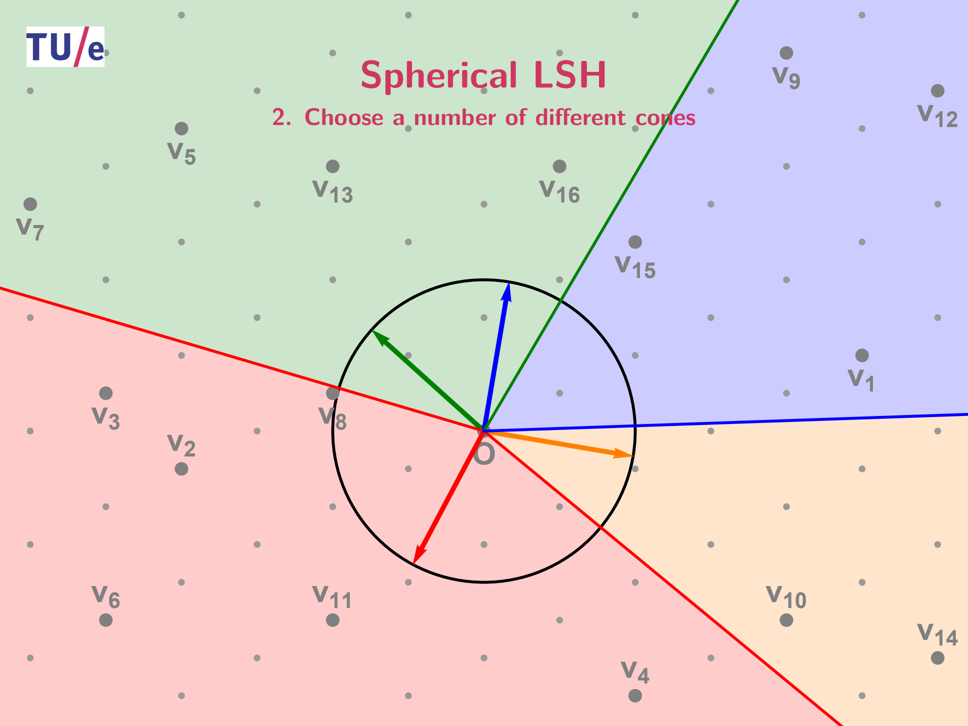
Spherical LSH

2. Choose a number of different cones



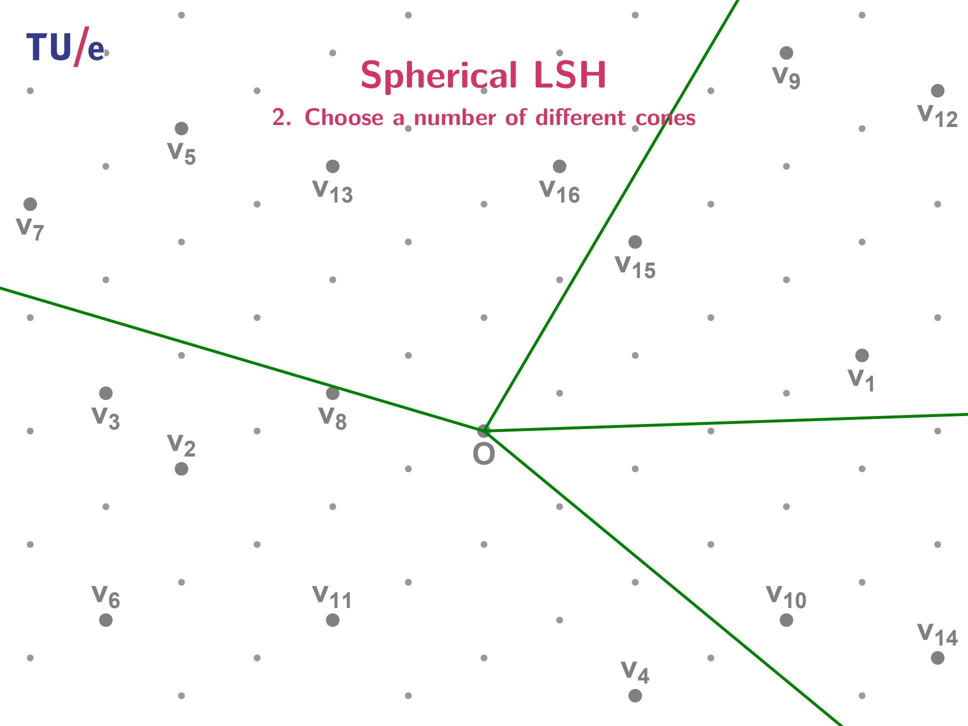
Spherical LSH

2. Choose a number of different cones



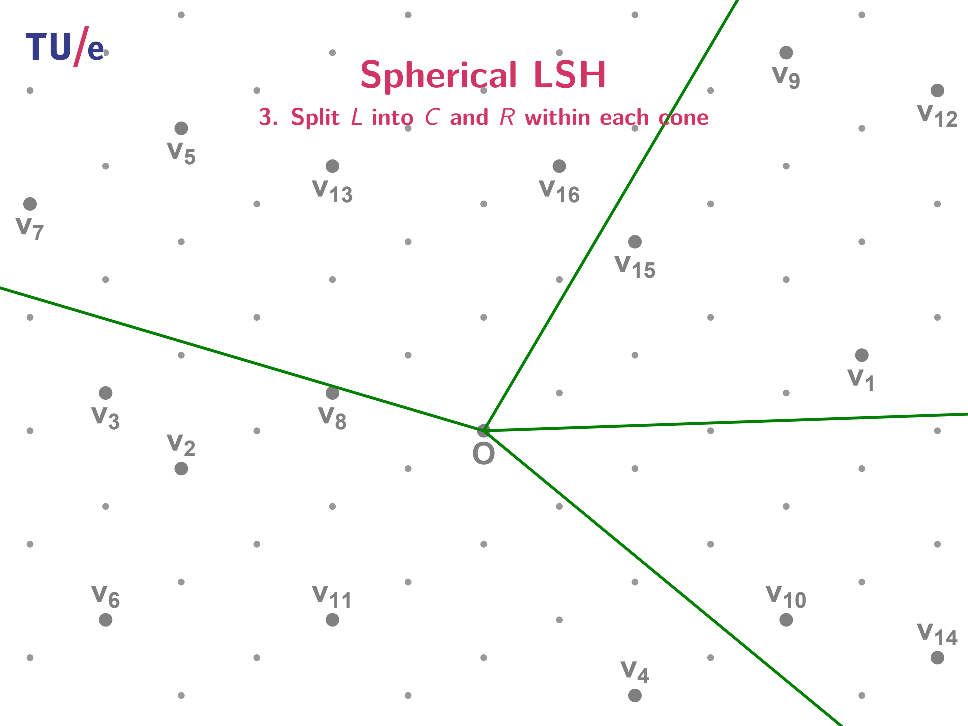
Spherical LSH

2. Choose a number of different cones



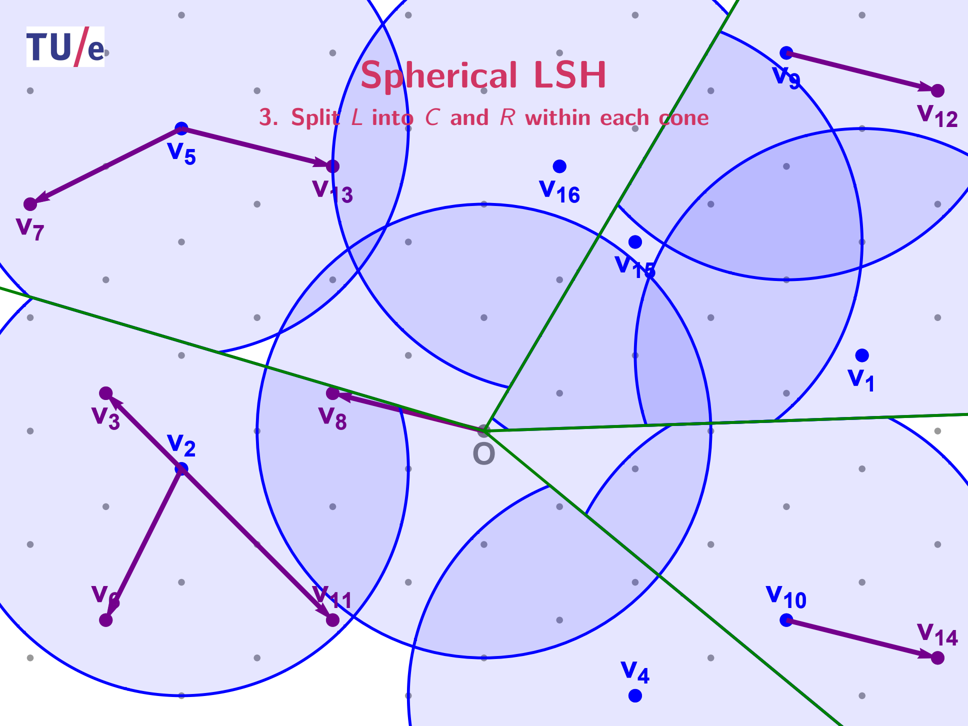
Spherical LSH

3. Split L into C and R within each cone



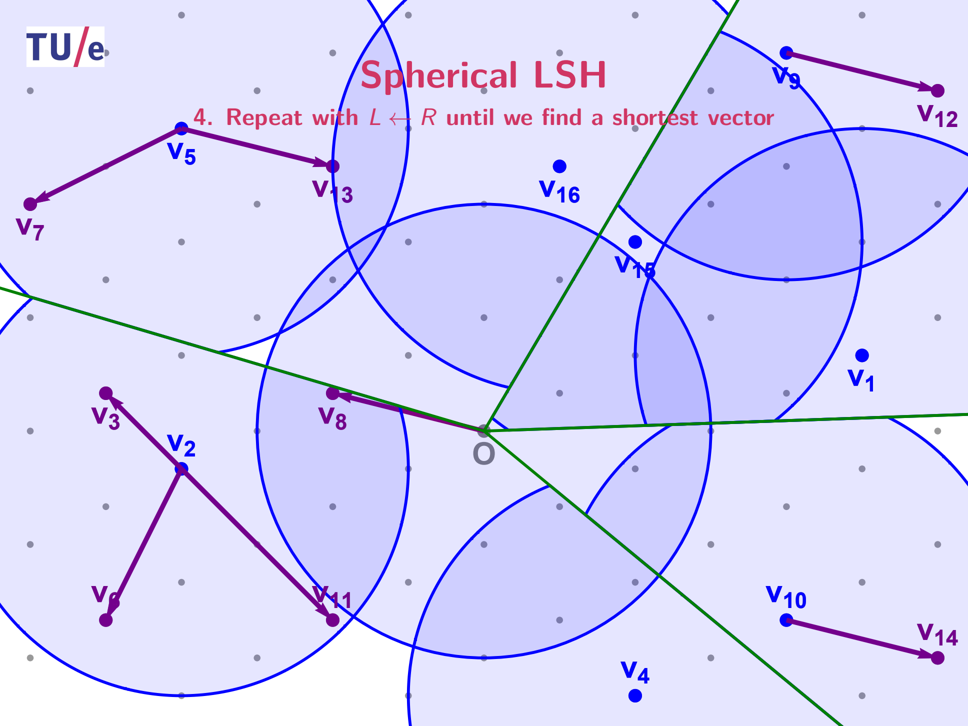
Spherical LSH

3. Split L into C and R within each cone



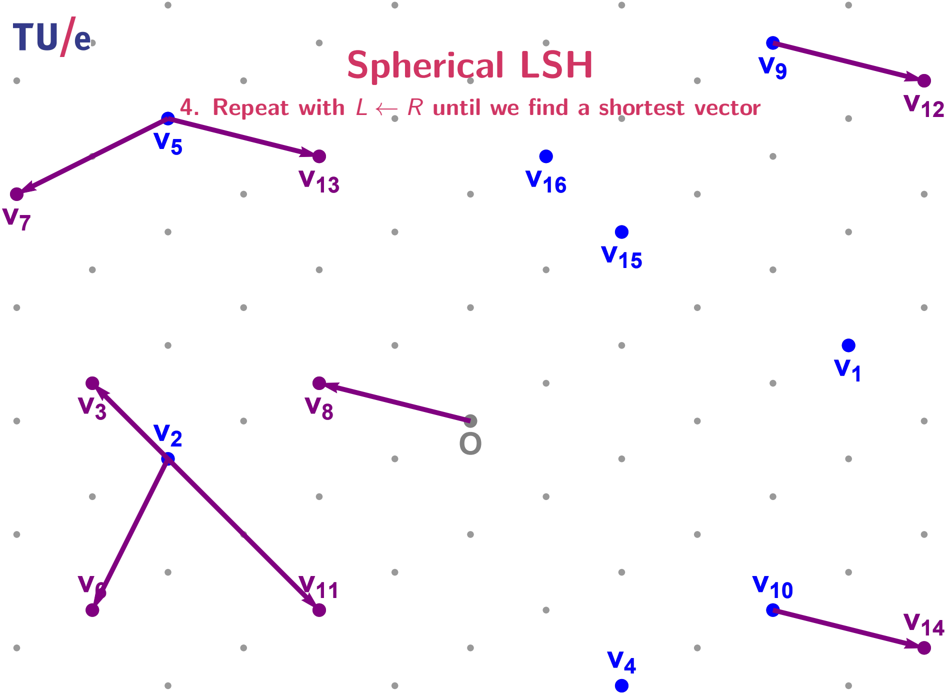
Spherical LSH

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



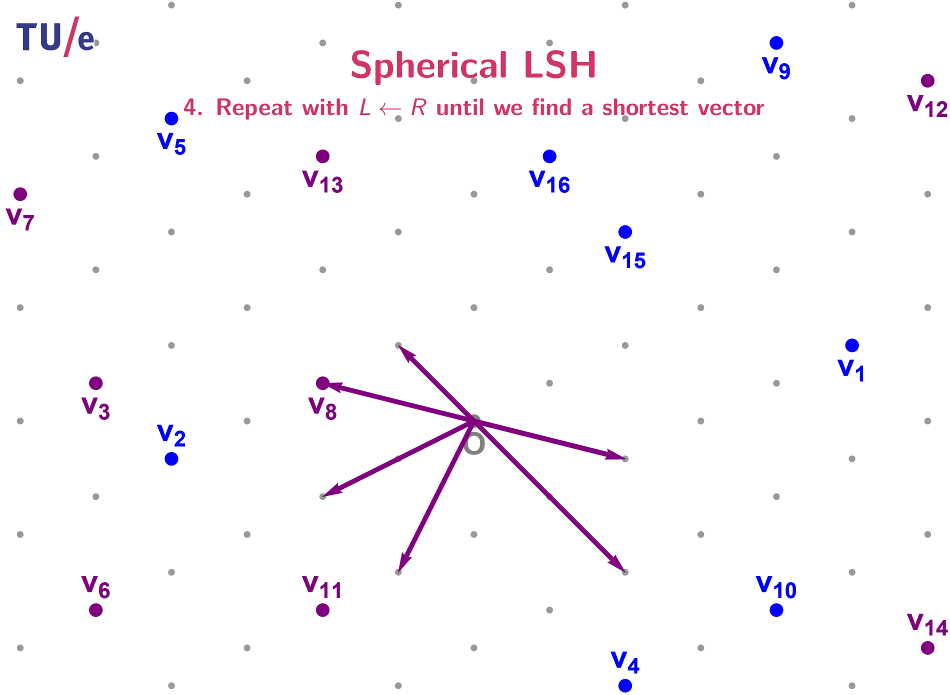
Spherical LSH

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



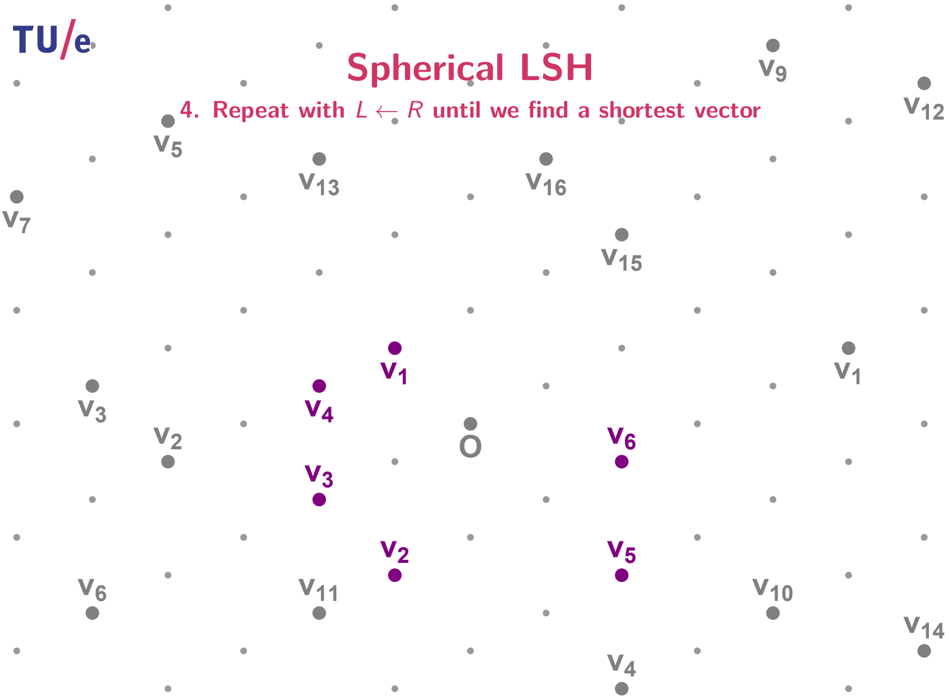
Spherical LSH

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



Spherical LSH

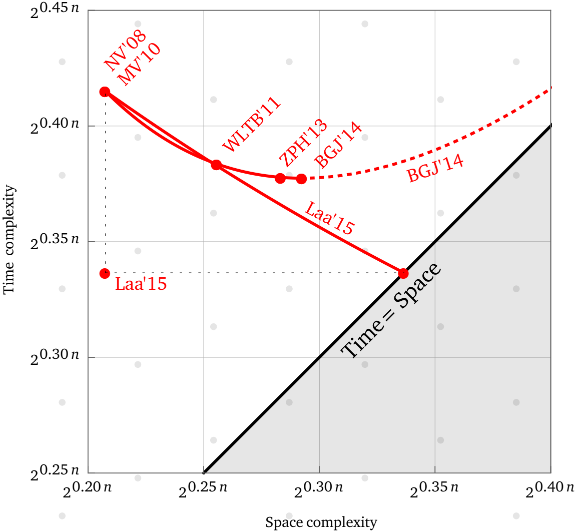
4. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

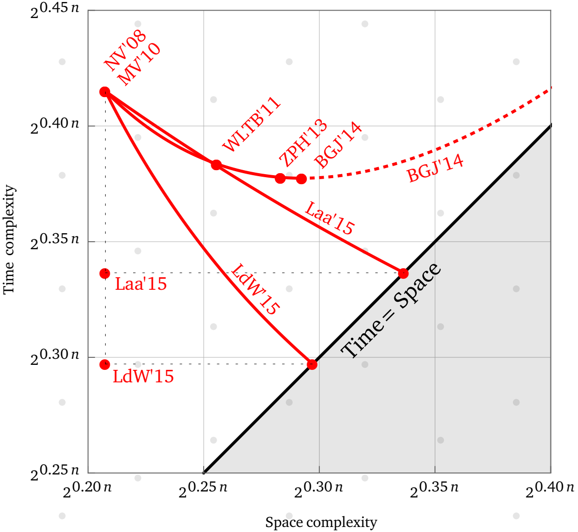
Spherical LSH

Space/time trade-off



Spherical LSH

Space/time trade-off

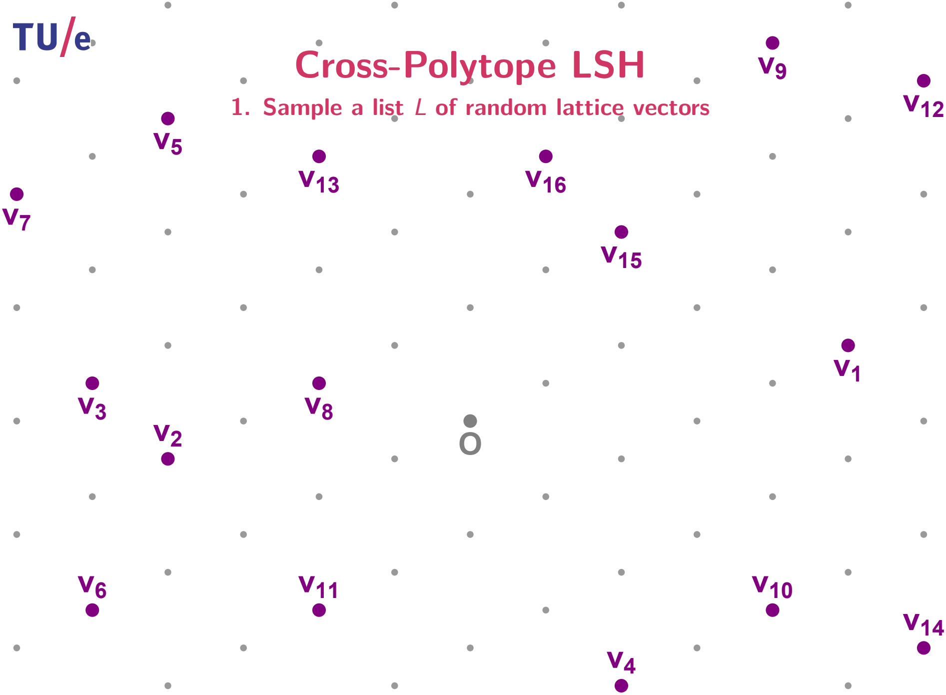


Cross-Polytope LSH

1. Sample a list L of random lattice vectors

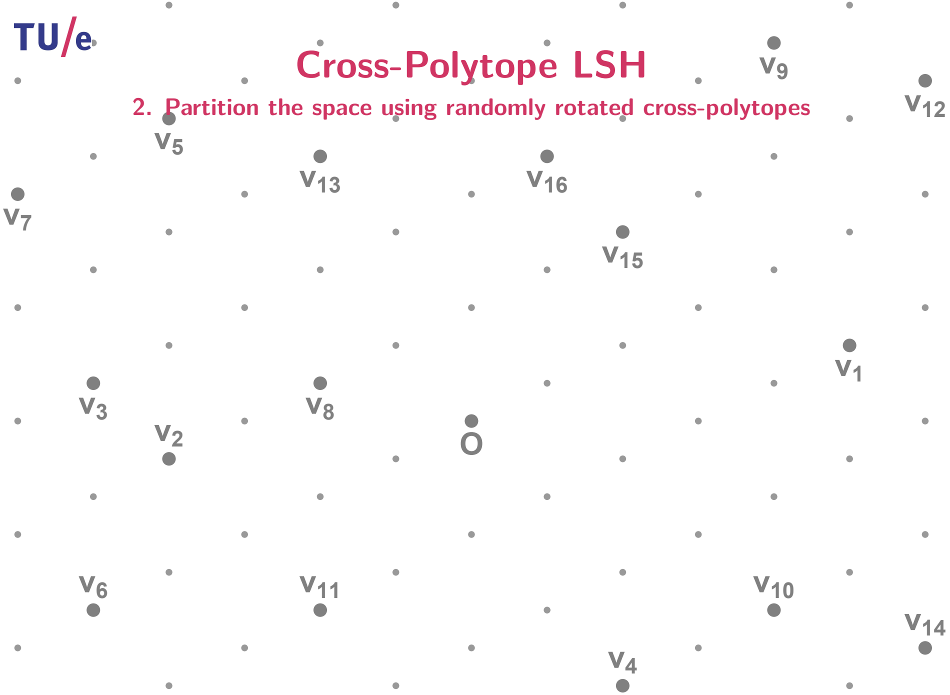
Cross-Polytope LSH

1. Sample a list L of random lattice vectors



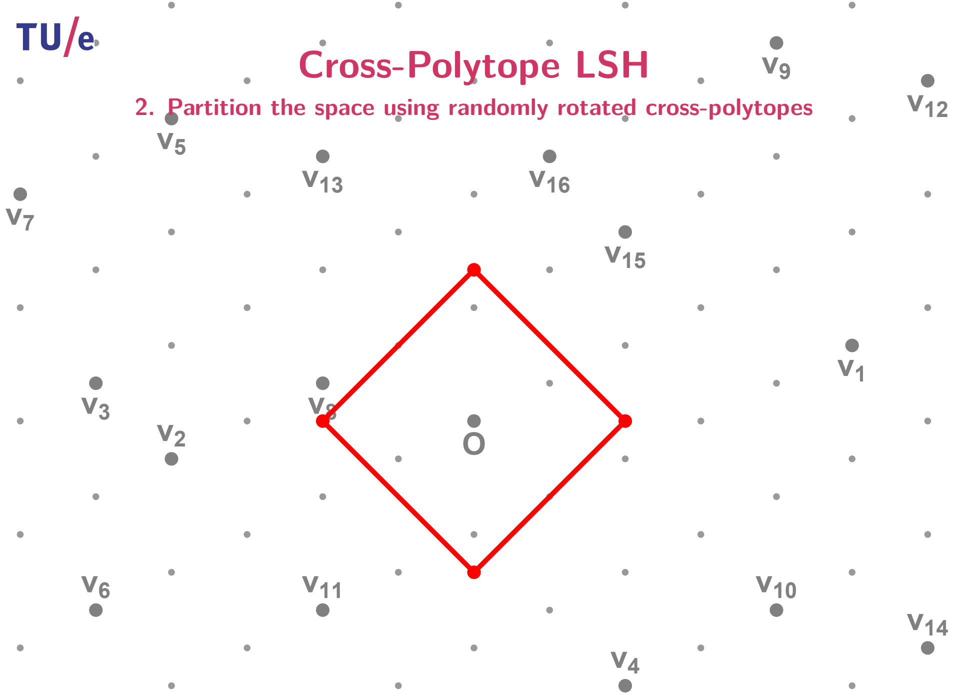
Cross-Polytope LSH

2. Partition the space using randomly rotated cross-polytopes



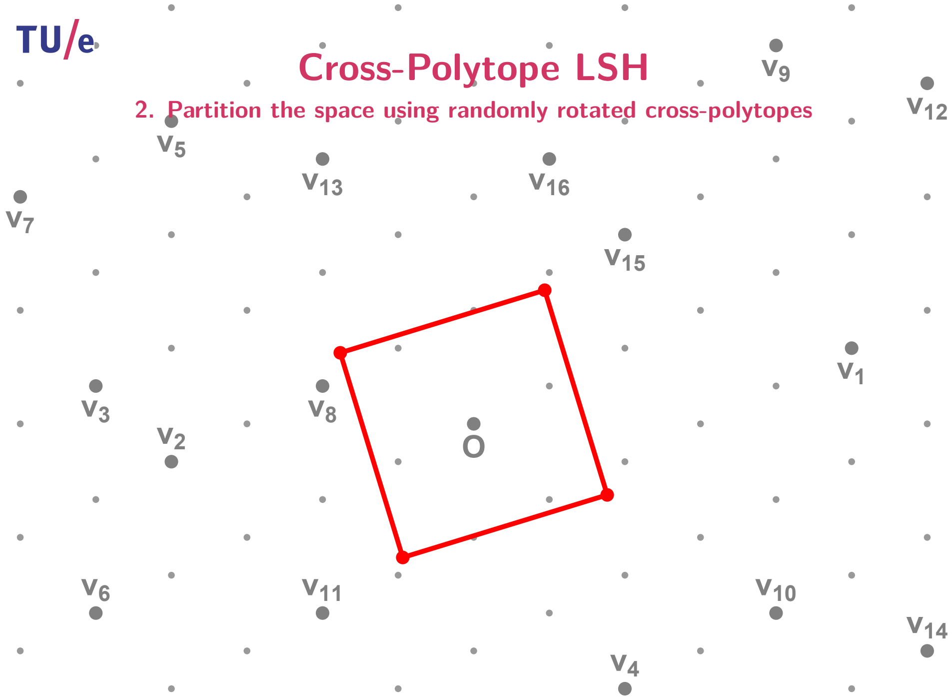
Cross-Polytope LSH

2. Partition the space using randomly rotated cross-polytopes



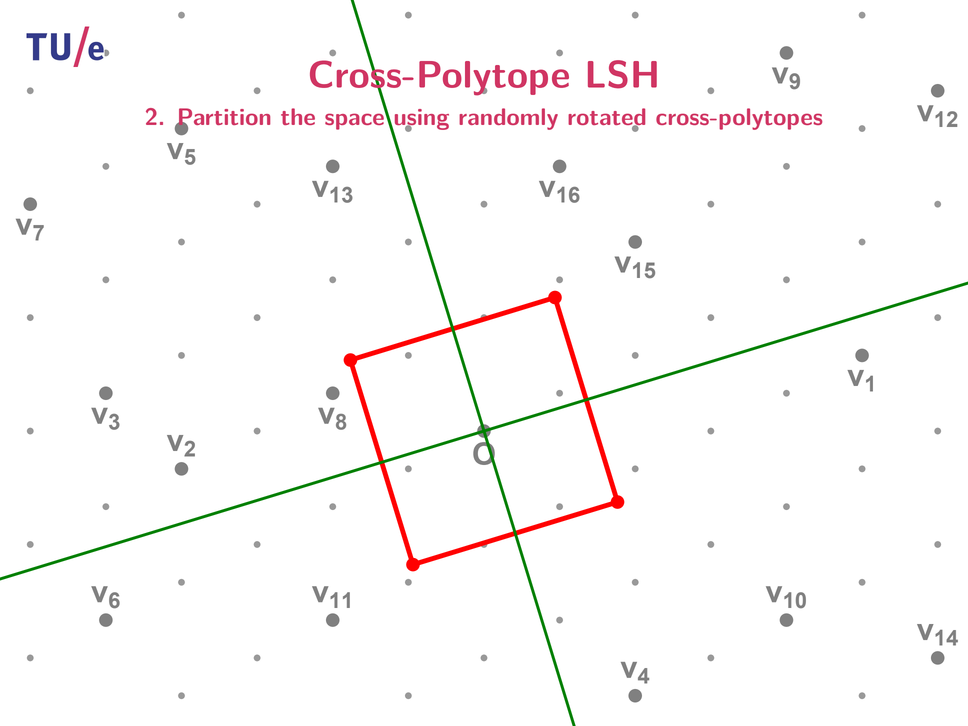
Cross-Polytope LSH

2. Partition the space using randomly rotated cross-polytopes



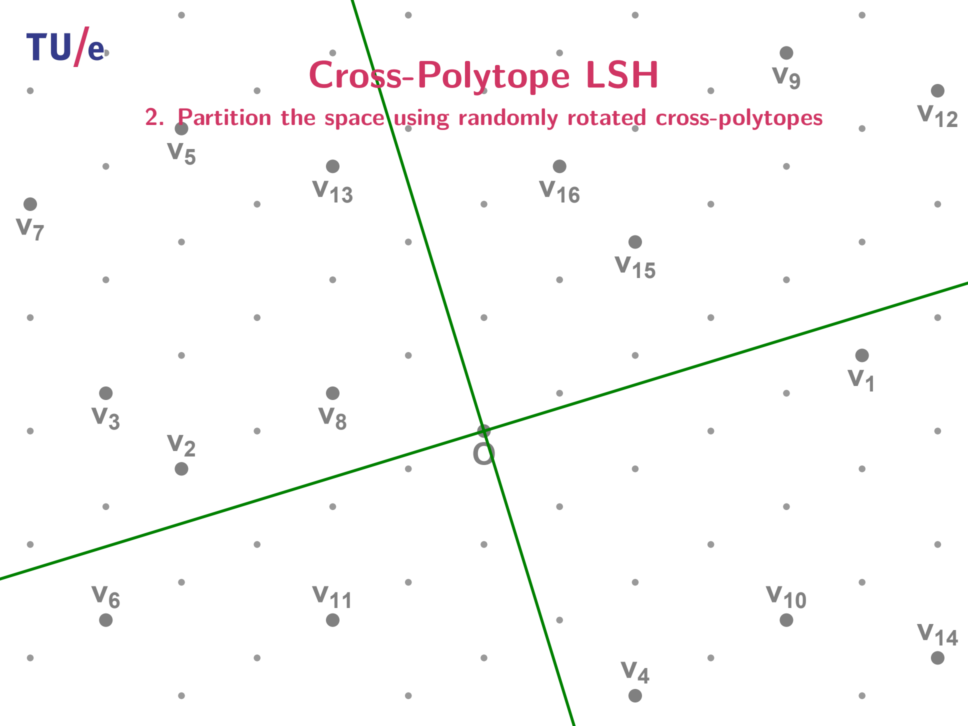
Cross-Polytope LSH

2. Partition the space using randomly rotated cross-polytopes



Cross-Polytope LSH

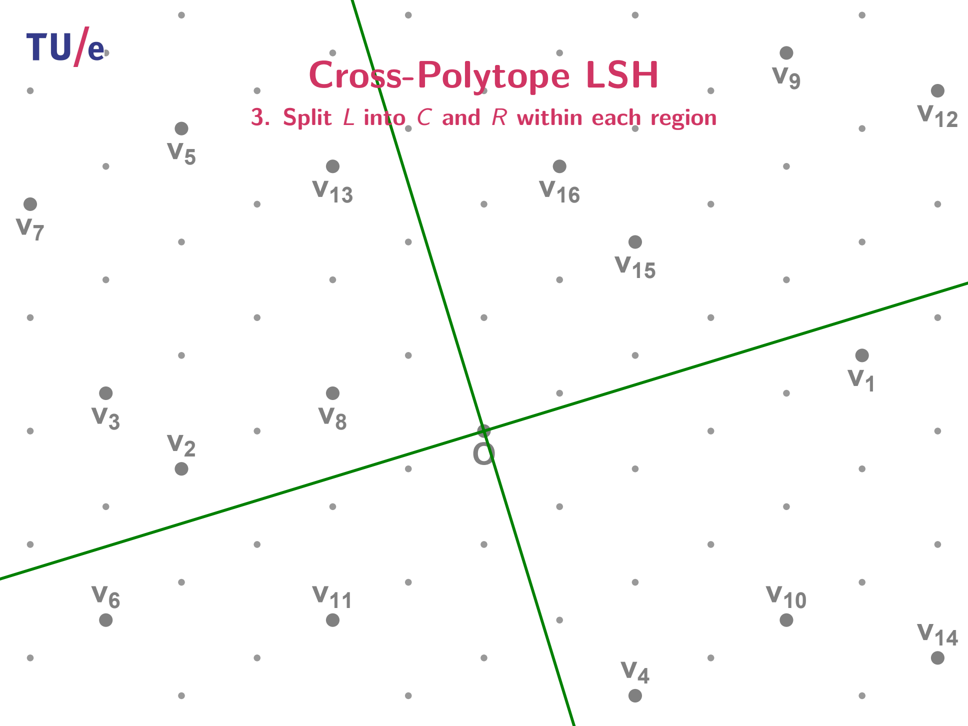
2. Partition the space using randomly rotated cross-polytopes



TU/e

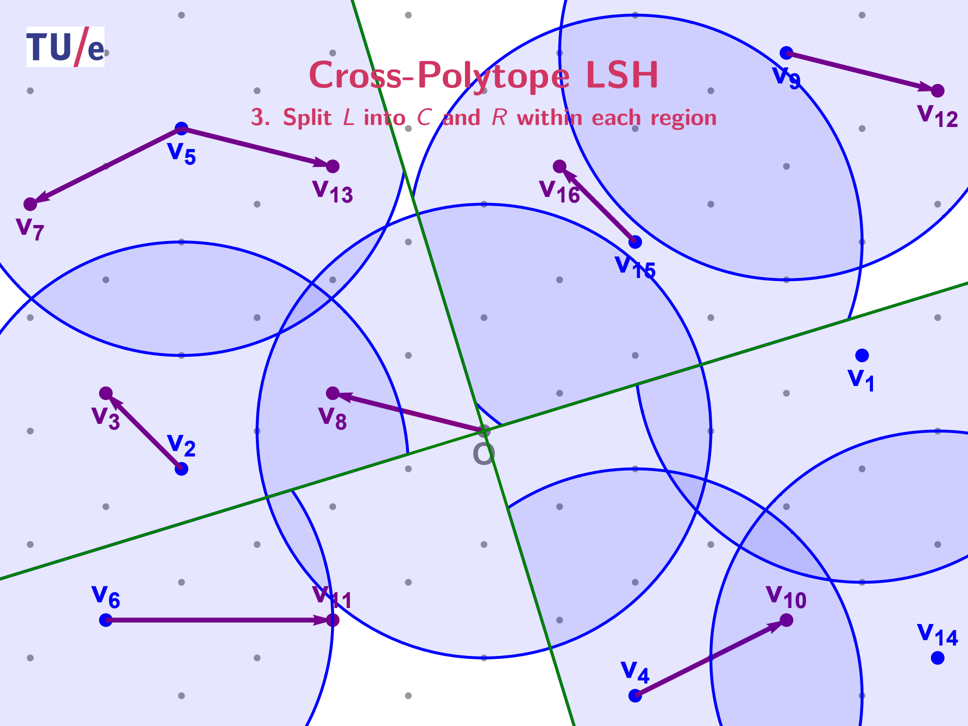
Cross-Polytope LSH

3. Split L into C and R within each region



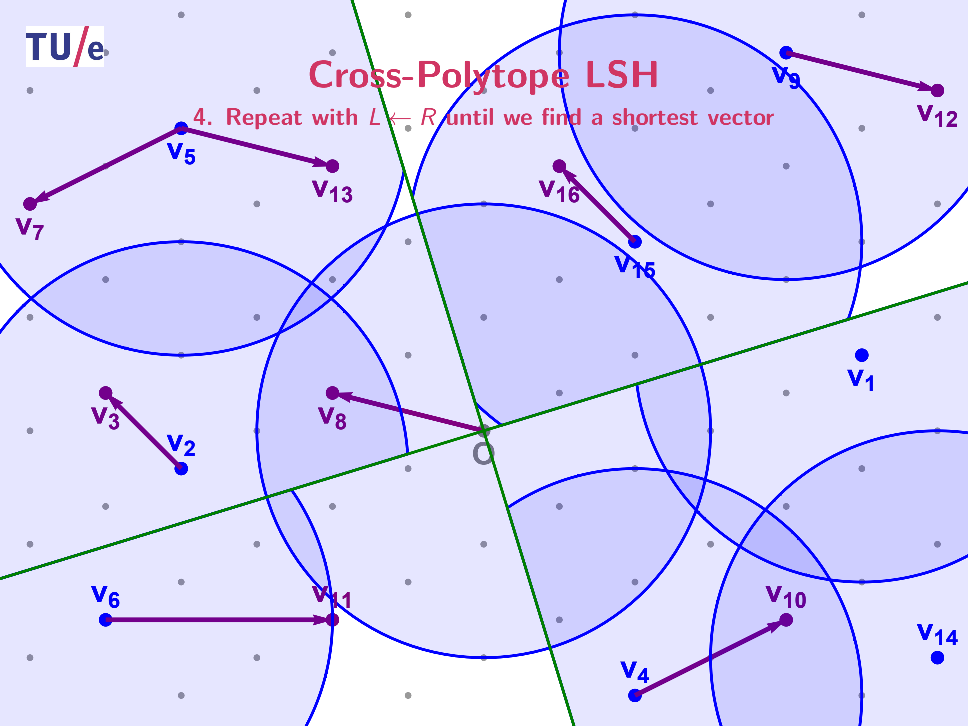
Cross-Polytope LSH

3. Split L into C and R within each region



Cross-Polytope LSH

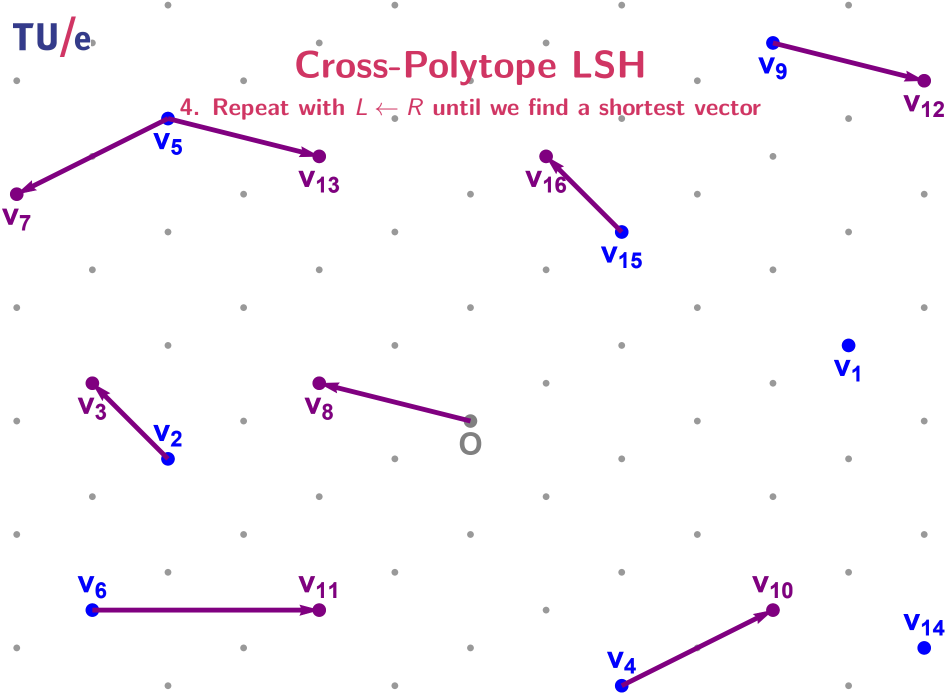
4. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

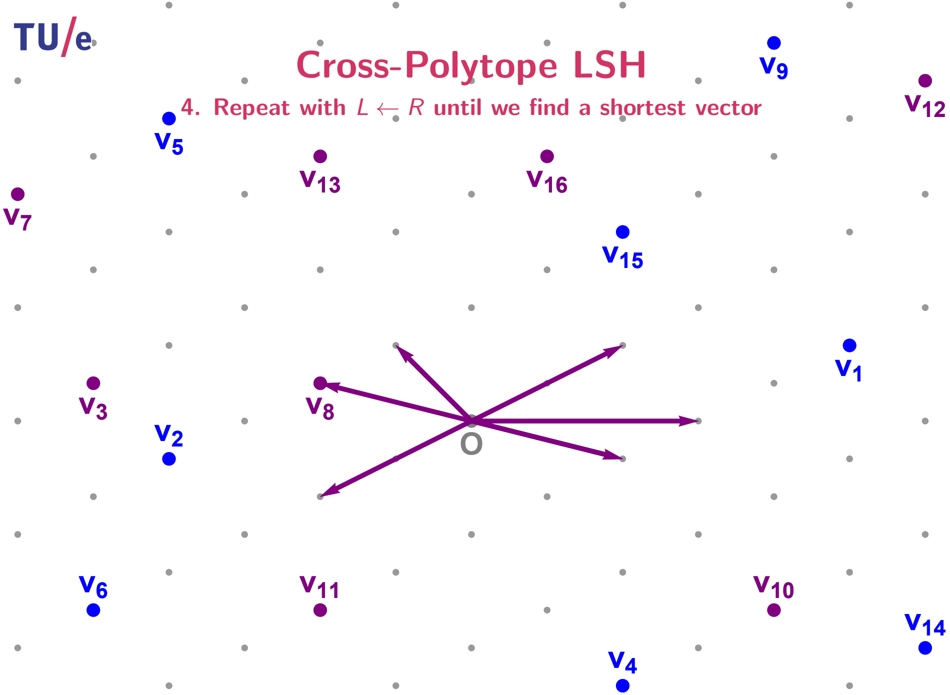
Cross-Polytope LSH

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



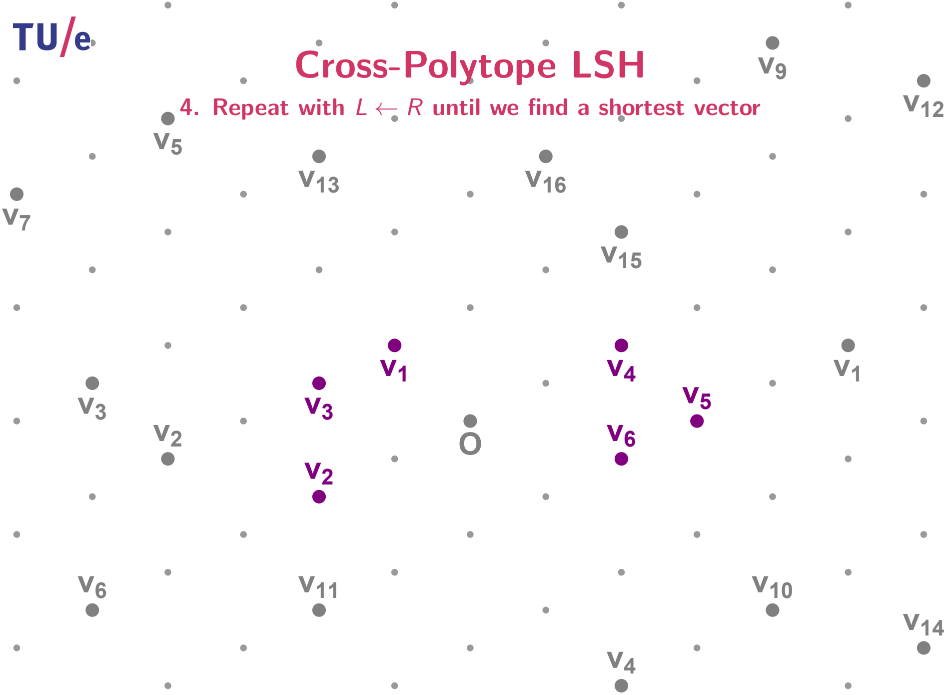
Cross-Polytope LSH

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



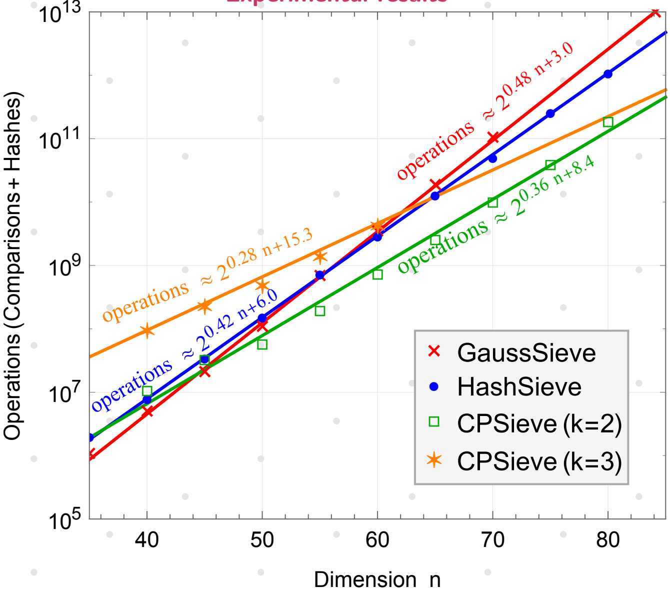
Cross-Polytope LSH

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



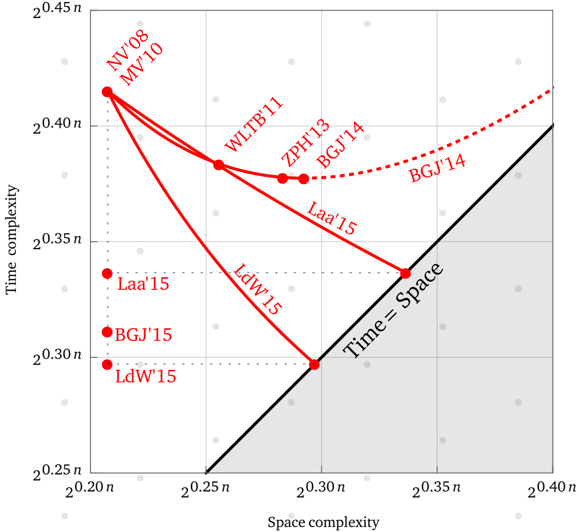
Cross-Polytope LSH

Experimental results



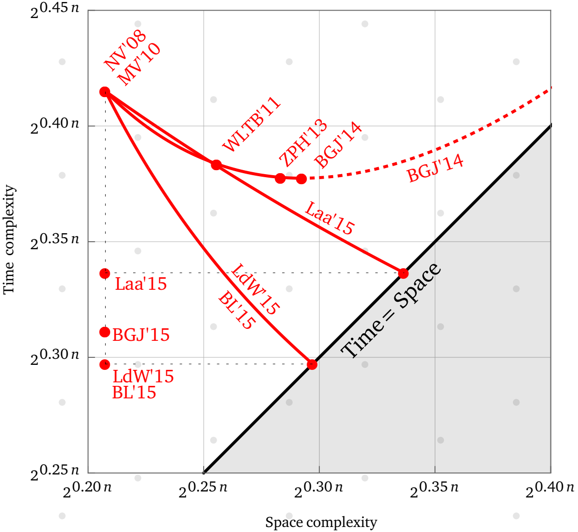
Cross-Polytope LSH

Space/time trade-off



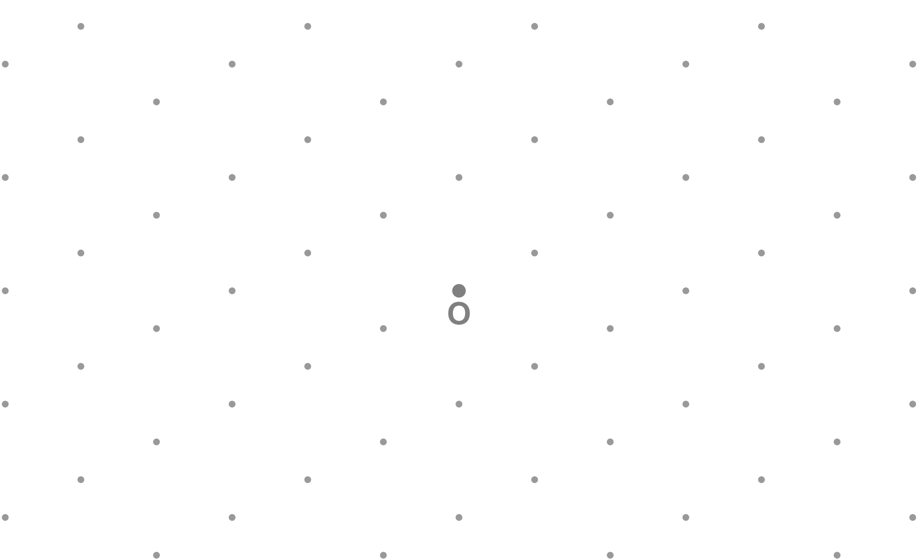
Cross-Polytope LSH

Space/time trade-off



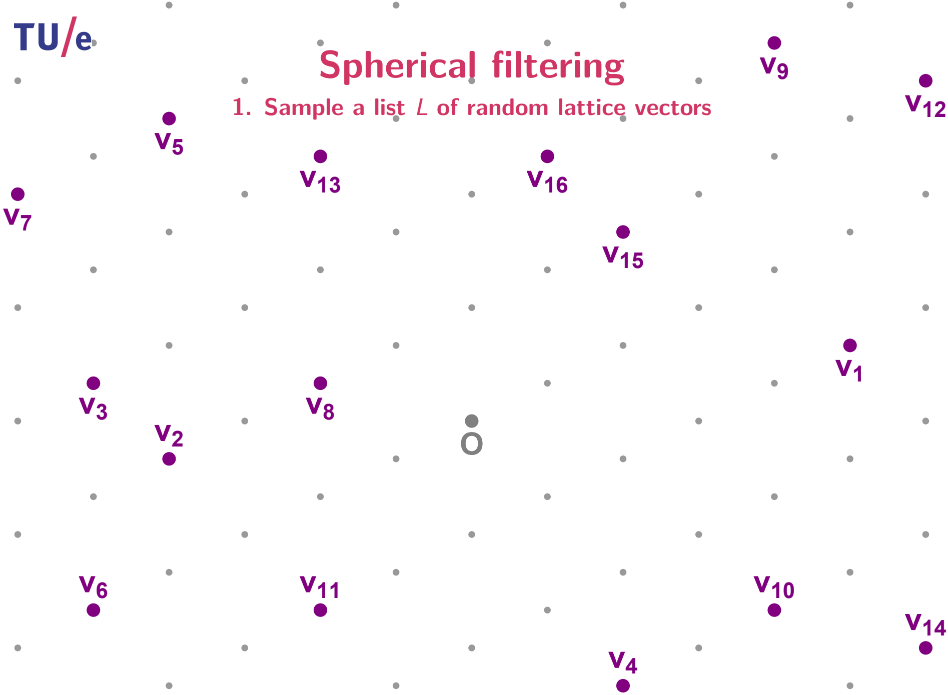
Spherical filtering

1. Sample a list L of random lattice vectors



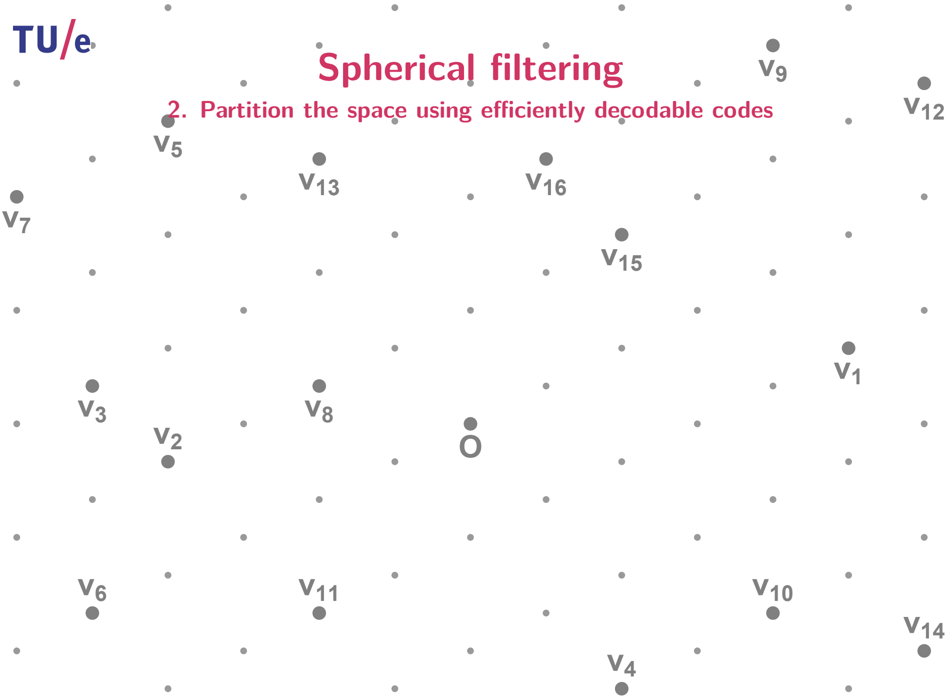
Spherical filtering

1. Sample a list L of random lattice vectors



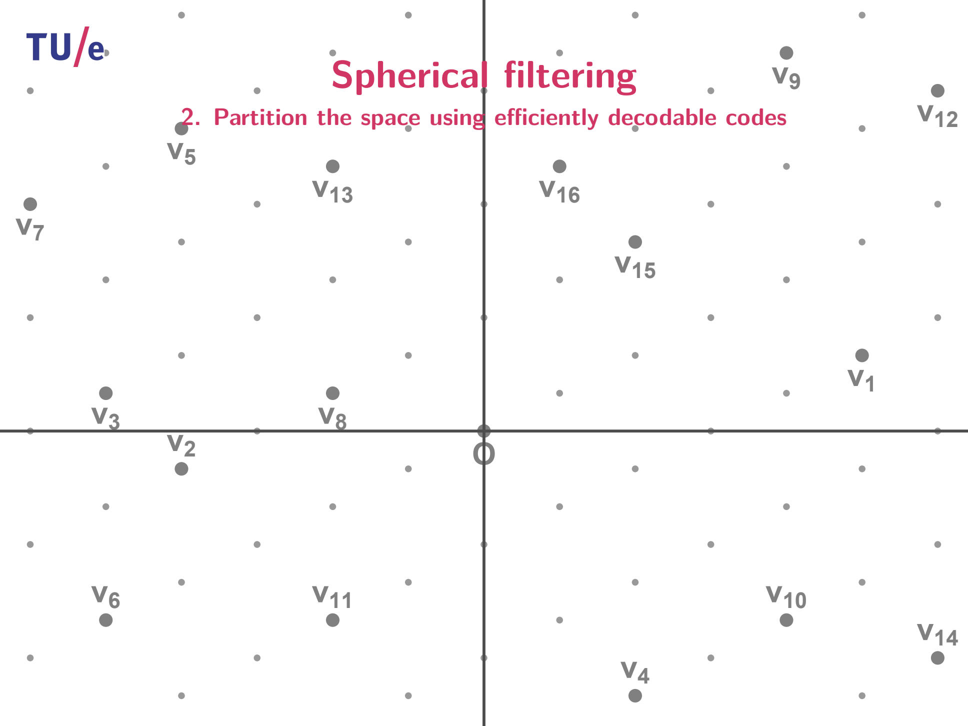
Spherical filtering

2. Partition the space using efficiently decodable codes



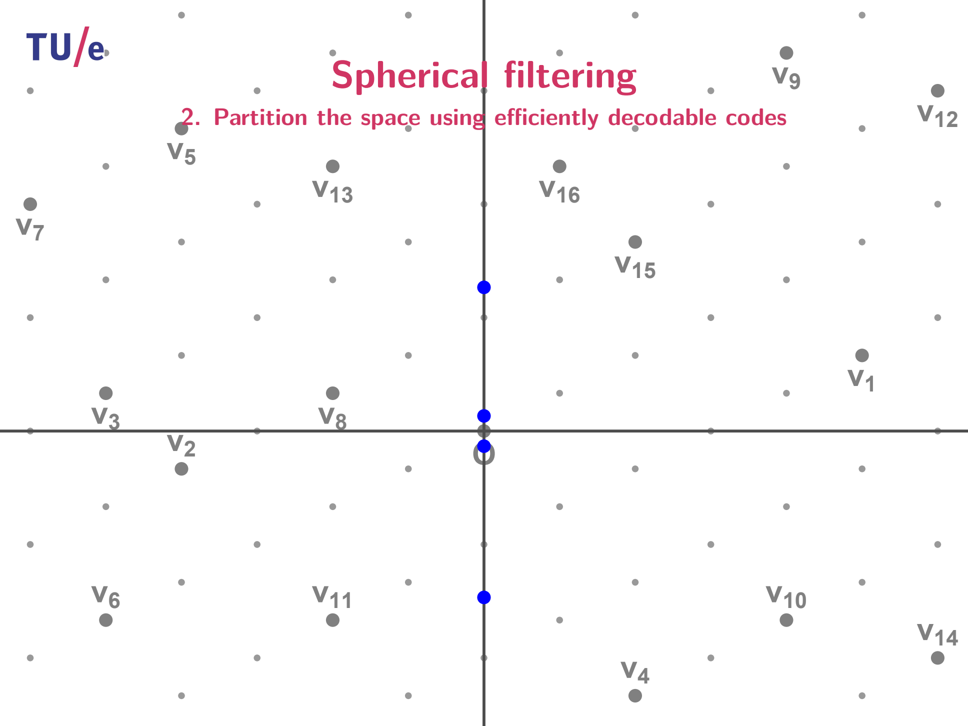
Spherical filtering

2. Partition the space using efficiently decodable codes



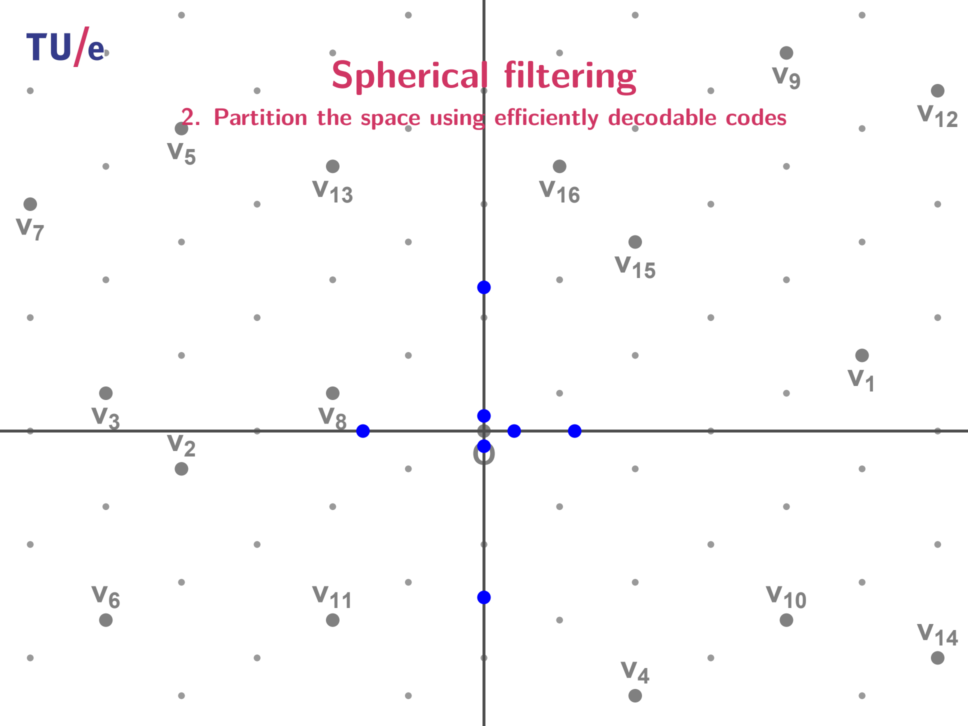
Spherical filtering

2. Partition the space using efficiently decodable codes



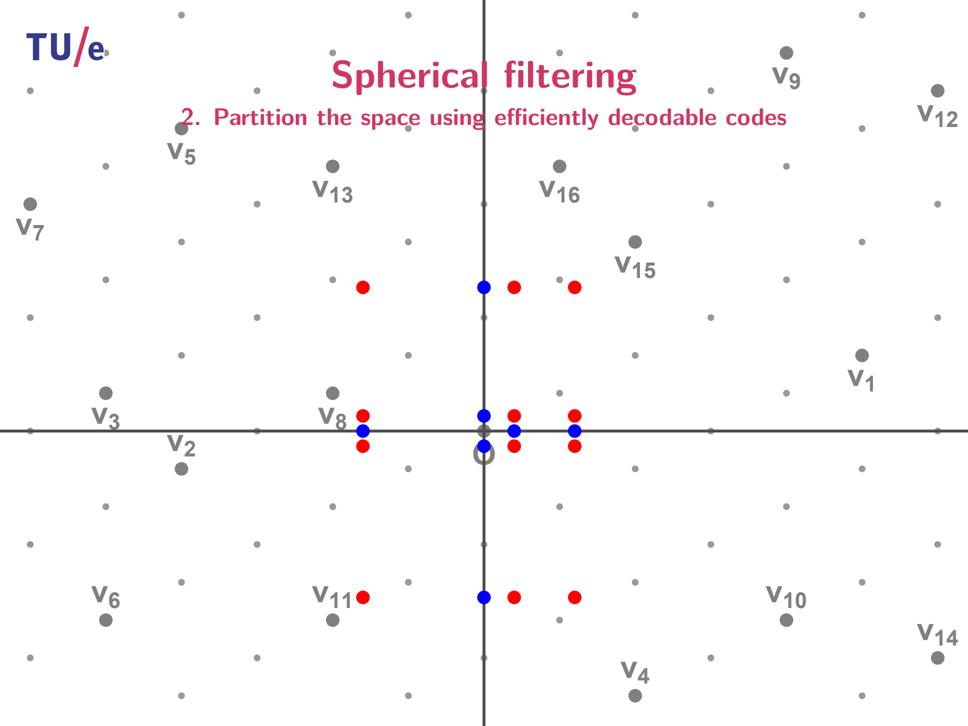
Spherical filtering

2. Partition the space using efficiently decodable codes



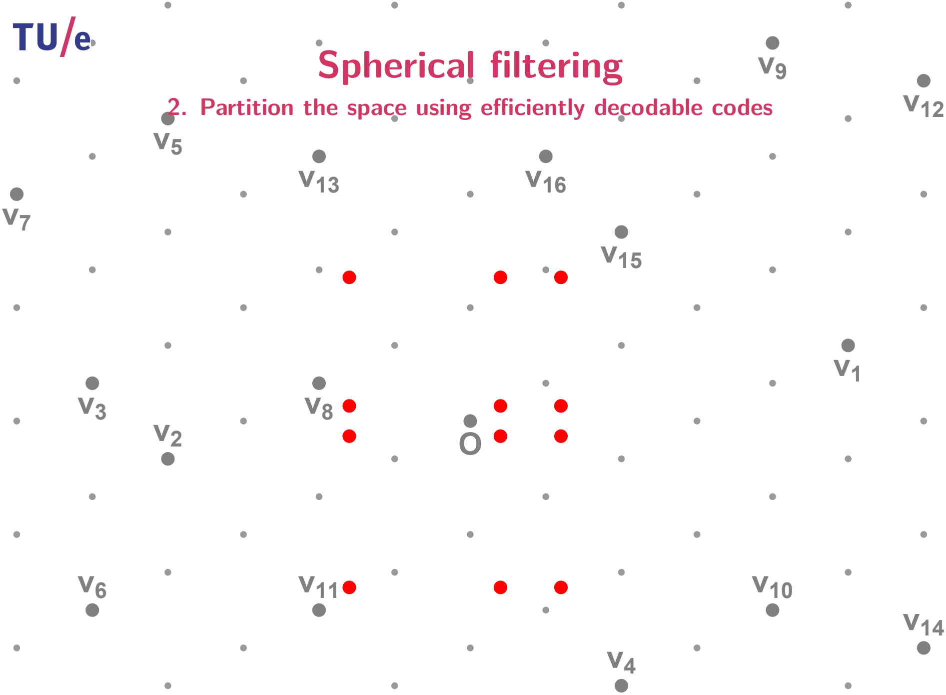
Spherical filtering

2. Partition the space using efficiently decodable codes



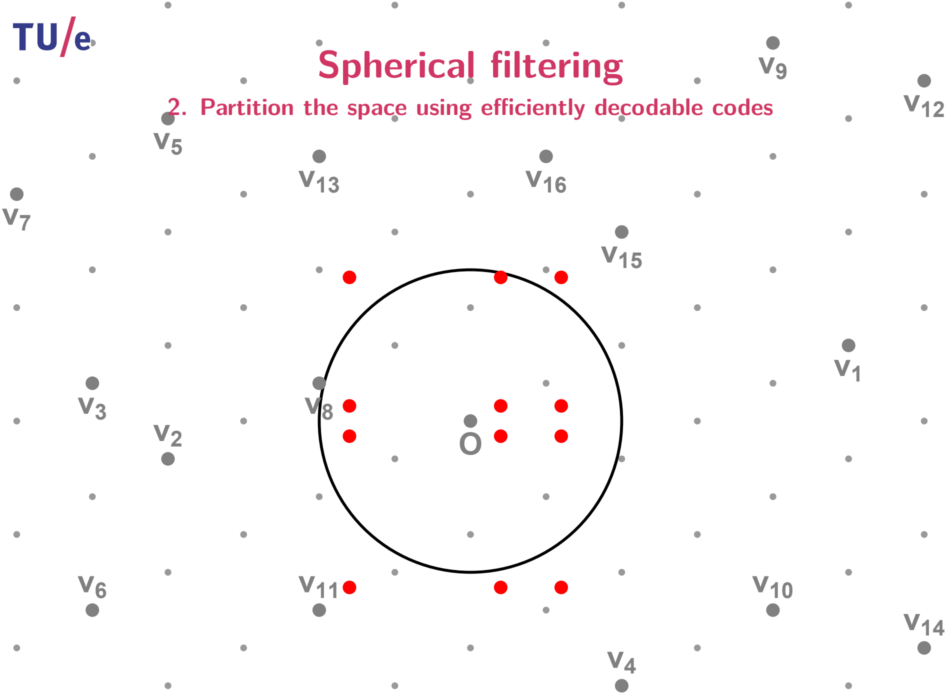
Spherical filtering

2. Partition the space using efficiently decodable codes



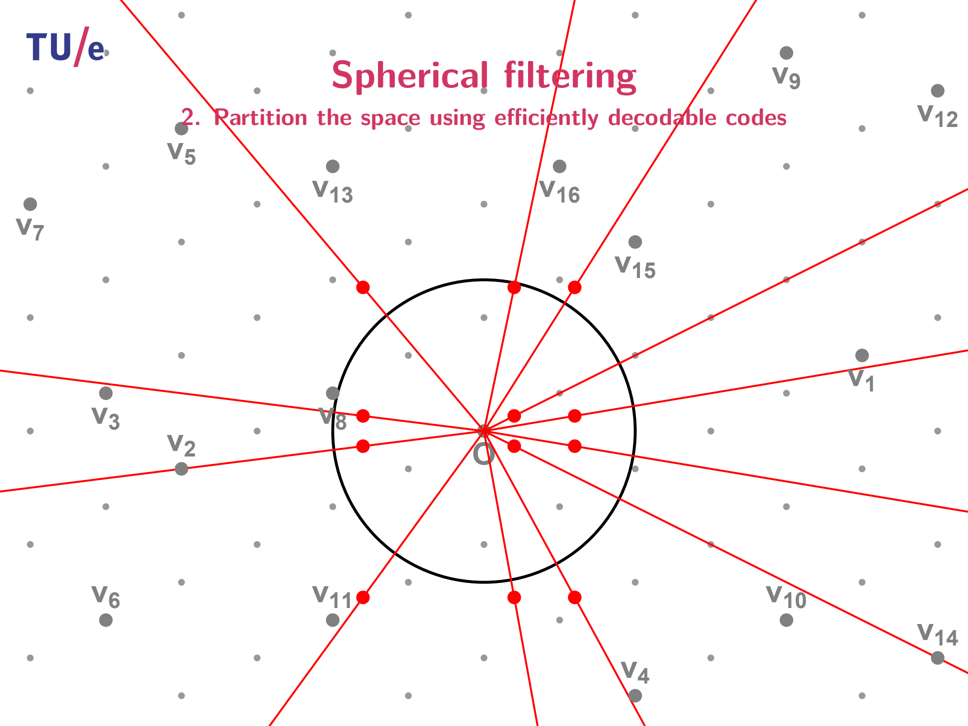
Spherical filtering

2. Partition the space using efficiently decodable codes



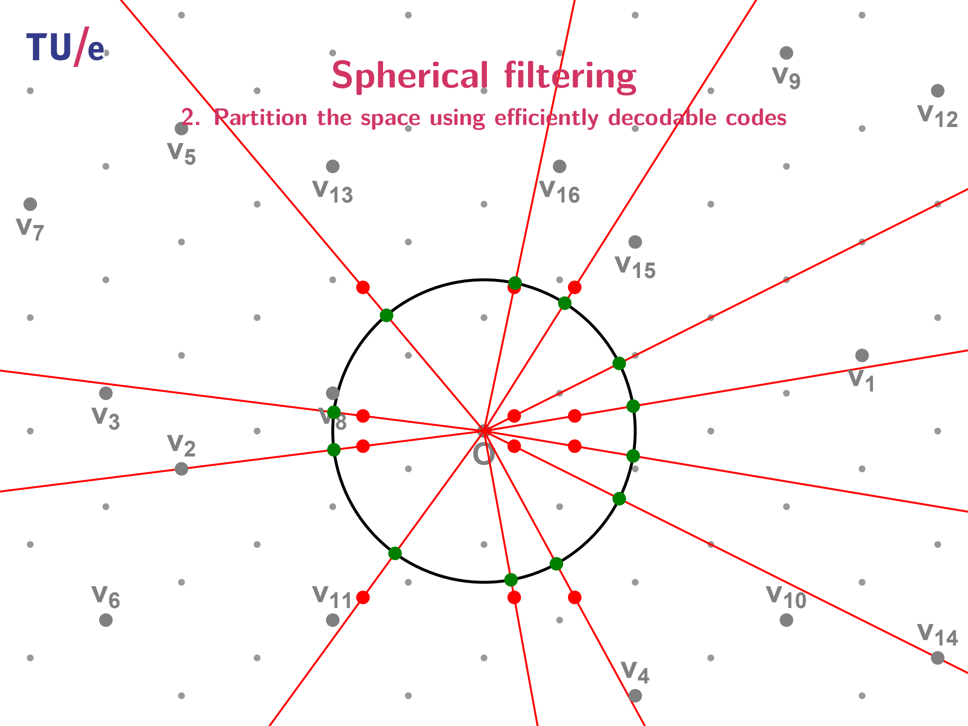
Spherical filtering

2. Partition the space using efficiently decodable codes



Spherical filtering

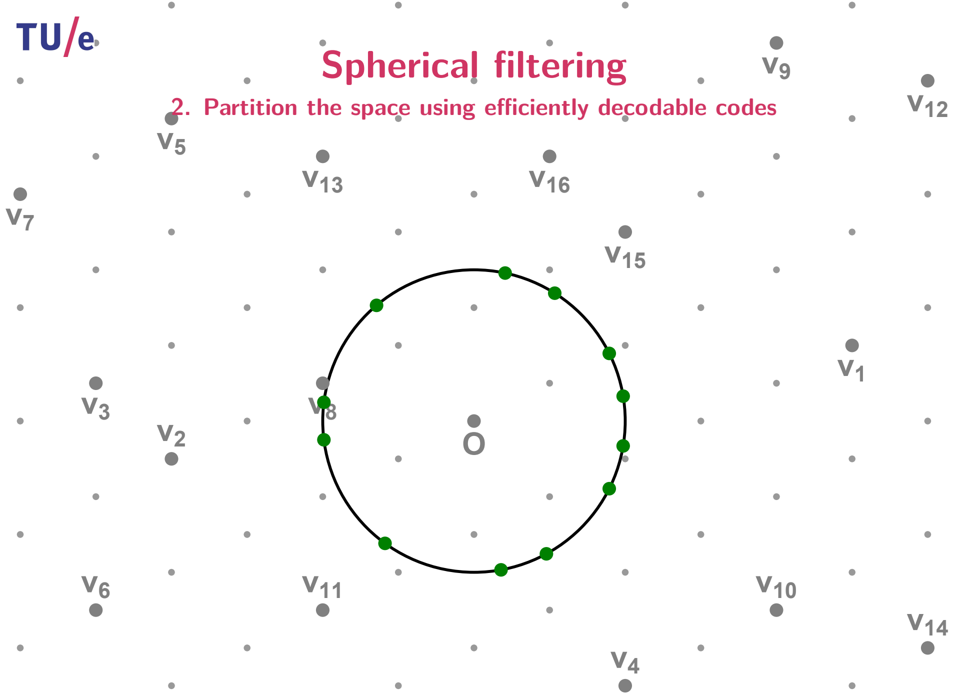
2. Partition the space using efficiently decodable codes



TU/e

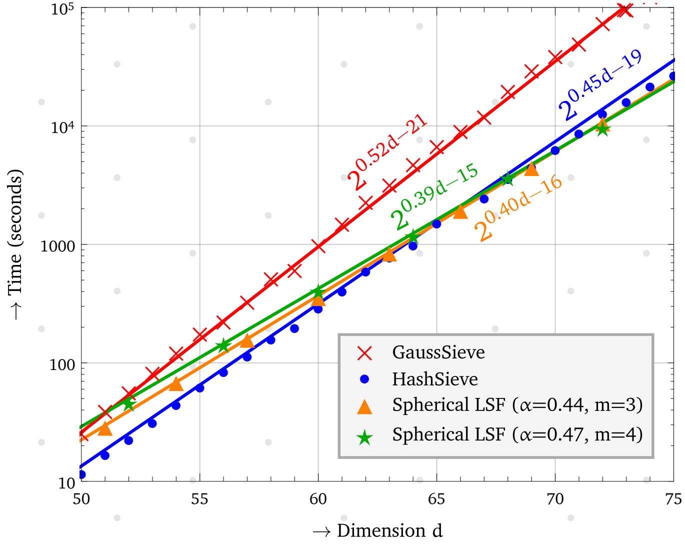
Spherical filtering

2. Partition the space using efficiently decodable codes



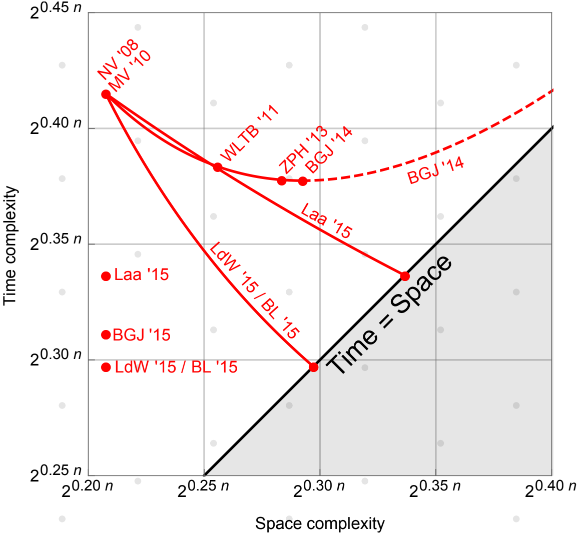
Spherical filtering

Experimental results



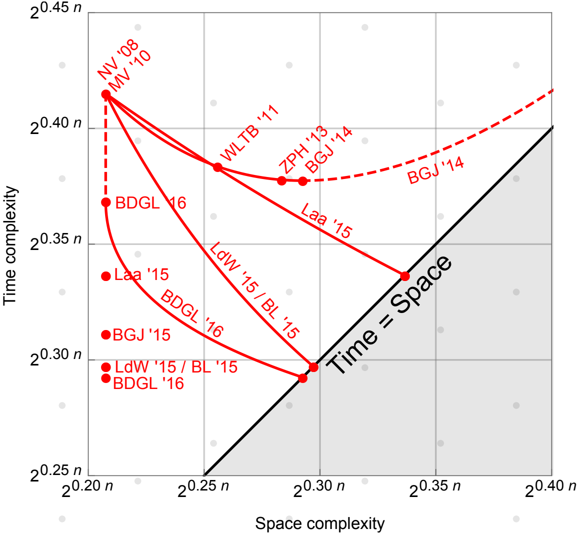
Spherical filtering

Space/time trade-off



Spherical filtering

Space/time trade-off



Implications for cryptography

Hardness of SVP in general

- Classically, dimension n costs $\approx 2^{0.29n}$ time and space
- Quantumly, dimension n costs $\approx 2^{0.25n}$ time and space [LMP15]

Implications for cryptography

Hardness of SVP in general

- Classically, dimension n costs $\approx 2^{0.29n}$ time and space
- Quantumly, dimension n costs $\approx 2^{0.25n}$ time and space [LMP15]

Ideal lattices used in cryptography

- Sieving with cross-polytope LSH improves by factor $\approx n$
- Other sieving algorithms improve by a factor $\approx n^{0.4}$
- Approximate solution suffices: further reduces SVP dimension
- LWE-based schemes: NNS can also improve BKW [Her15]

Implications for cryptography

Hardness of SVP in general

- Classically, dimension n costs $\approx 2^{0.29n}$ time and space
- Quantumly, dimension n costs $\approx 2^{0.25n}$ time and space [LMP15]

Ideal lattices used in cryptography

- Sieving with cross-polytope LSH improves by factor $\approx n$
- Other sieving algorithms improve by a factor $\approx n^{0.4}$
- Approximate solution suffices: further reduces SVP dimension
- LWE-based schemes: NNS can also improve BKW [Her15]

Many challenges still remain!

Questions

[vdP'12]