Parallel (Public-Key) Cryptanalysis

Joppe W. Bos

NXP

Summer school
on real-world crypto and privacy

What is Parallel cryptanalysis?

From the concise Oxford Dictionary (ninth edition)

Parallel
"*Computing* involving the simultaneous performance of operations"

Cryptanalysis
"the art or process of solving cryptograms by analysis; code-breaking"

What is Parallel cryptanalysis?

The working rebuilt bombe at Bletchley Park museum.
Picture by Antoine Taveneaux

From the concise Oxford Dictionary (ninth edition)

Parallel
"*Computing* involving the simultaneous performance of operations"

Cryptanalysis
"the art or process of solving cryptograms by analysis; code-breaking"

Parallel cryptanalysis can be applied in many different settings
- ✓ Brute force
- ✓ Public-key / symmetric cryptography
- ✓ Computation of higher-order correlation power analysis attacks

What is Parallel cryptanalysis?

The working rebuilt bombe at Bletchley Park museum.
Picture by Antoine Taveneaux

From the concise Oxford Dictionary (ninth edition)

Parallel
"*Computing* involving the simultaneous performance of operations"

Cryptanalysis
"the art or process of solving cryptograms by analysis; code-breaking"

Parallel cryptanalysis can be applied in many different settings
- ✓ Brute force
- ✓ Public-key / symmetric cryptography
- ✓ Computation of higher-order correlation power analysis attacks

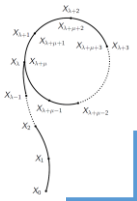
Use *parallel cryptanalysis* to solve mathematical problems which form the theoretical foundation of many public-key cryptographic schemes.

Outline

Integer factorization

- RSA

  Still the most widely used public-key cryptosystem

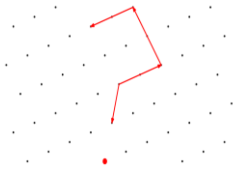


Discrete Logarithm

- DH
- ElGamal
- DSA

  In this talk Elliptic curve DLP

- ECDH(E)
- ECDSA



Post-Quantum

- Lattice-based
  R-LWE, NTRU
- Hash-based
  Merkle trees
- Code-based
  McEliece

Most common problems used to base public-key cryptography systems on

Goals

1. Show the *best* methods to cryptanalyze public-key cryptography
   a) Explain some of the details
   b) Effort estimates (security assessment)

2. From a computational and parallel point of view

3. Public-key cryptography is fun!

Tiny keys
Fast crypto
No security

Huge keys
Slow crypto
Too much (?) security

Tiny keys	Huge keys
Fast crypto	Slow crypto
No security	Too much (?) security

Security assessment

Finding the “optimal” key size is difficult

<u>Approach: Use the **best** parallelizable algorithms</u>

Tiny keys

Fast crypto

No security

Security assessment

Huge keys

Slow crypto

Too much (?) security

Finding the “optimal” key size is difficult

Approach: Use the **best** parallelizable algorithms

Does it make sense to say the best attack?

Fastest	(time)
Minimize power consumption	(green)
Minimize investment	(re-use existing hardware)
Et cetera	(invent your own characterization for best)

Tiny keys
Fast crypto
No security

Security assessment

Huge keys
Slow crypto
Too much (?) security

Finding the “optimal” key size is difficult

Approach: Use the **best** parallelizable algorithms

Does it make sense to say the best attack?

Fastest	(time)
Minimize power consumption	(green)
Minimize investment	(re-use existing hardware)
Et cetera	(invent your own characterization for best)

This presentation: Minimize wall-clock time using commonly available compute power

Integer Factorization

Many **generic** integer factoring algorithms follow the same **old** approach.

Idea: an odd integer n can be written as the **difference of two squares**

For instance, idea behind
- Fermat factorization method
- Quadratic sieve (and variants)
- Number field sieve

Given a composite odd $n \in \mathbb{Z}$, find non-trivial factors p and q such that $p \cdot q = n \quad (q > p)$.

Integer Factorization

Many **generic** integer factoring algorithms follow the same **old** approach.

Idea: an odd integer n can be written as the **difference of two squares**

For instance, idea behind
- Fermat factorization method
- Quadratic sieve (and variants)
- Number field sieve

Given a composite odd $n \in \mathbb{Z}$, find non-trivial factors p and q such that $p \cdot q = n \quad (q > p)$.

$$\begin{aligned} p \cdot q &= \left(\frac{p+q}{2} - \frac{q-p}{2}\right) \cdot \left(\frac{p+q}{2} + \frac{q-p}{2}\right) \\ &= (x-y) \cdot (x+y) \\ &= x^2 - y^2 \end{aligned}$$

Since p and q are odd the **average** $\frac{p+q}{2}$ is an integer and $\frac{q-p}{2}$ is the **distance** from this average to p (or q)

Integer Factorization: Fermat factorization method

Good at finding large divisors

Given n, try to find x and y such that $x^2 - n = y^2$, start with $\lceil \sqrt{n} \rceil$

Example.

$$2279 = 43 \times 53$$

$$\lceil \sqrt{2279} \rceil = 48$$

$$48^2 - 2279 = 25 = 5^2$$

$$(48 + 5)(48 - 5) = 43 \times 53 = 2279$$

Integer Factorization: Fermat factorization method

Good at finding large divisors

Given n, try to find x and y such that $x^2 - n = y^2$, start with $\lceil \sqrt{n} \rceil$

Example.

$$2279 = 43 \times 53$$
$$\lceil \sqrt{2279} \rceil = 48$$
$$48^2 - 2279 = 25 = 5^2$$
$$(48 + 5)(48 - 5) = 43 \times 53 = 2279$$

Generalization

Find x and y such that

$$x^2 \equiv y^2 \pmod{n} \text{ and } x \not\equiv y \pmod{n}$$

then with high probability

$$\gcd(x - y, n) \neq 1 \text{ or } n$$

Integer Factorization

1. **Polynomial selection**
   Degree $d > 1$, integer $m \approx n^{1/d}$, radix-m representation of $n = f_d m^d + \cdots + f_1 m + f_0$
   Leads to $f_a(X) = \sum_{i=1}^{d} f_i X^i \in \mathbb{Z}[X]$ with $f_a(m) \equiv 0 \pmod{n}$ (one can do better!) and $f_r(X) = X - m$

2. **Relation collection**
   Find co-prime $a, b \in \mathbb{Z} \times \mathbb{Z}_{\geq 0}$ such that $bf_r(a/b)$ and $b^d f_a(a/b)$ factors into small primes ("smooth")

3. **Matrix step**
   Find even sum of small prime exponent vectors, solve linear dependencies between the relations
   (find random elements of the null-space of the matrix)

4. **Square root**
   Compute the square root of a large element of the number field

Integer Factorization

1. **Polynomial selection**
   Degree $d > 1$, integer $m \approx n^{1/d}$, radix-m representation of $n = f_d m^d + \cdots + f_1 m + f_0$
   Leads to $f_a(X) = \sum_{i=1}^{d} f_i X^i \in \mathbb{Z}[X]$ with $f_a(m) \equiv 0 \pmod{n}$ (one can do better!) and $f_r(X) = X - m$
   40 CPU years

2. **Relation collection**
   Find co-prime $a, b \in \mathbb{Z} \times \mathbb{Z}_{\geq 0}$ such that $bf_r(a/b)$ and $b^d f_a(a/b)$ factors into small primes ("smooth")
   1500 CPU years, easy to run in parallel

3. **Matrix step**
   Find even sum of small prime exponent vectors, solve linear dependencies between the relations (find random elements of the null-space of the matrix)
   152 CPU years, not easy to run in parallel

4. **Square root**
   Compute the square root of a large element of the number field
   < 1 CPU day

Thorsten Kleinjung et al.: **Factorization of a 768-bit RSA modulus**. *CRYPTO 2010*

Integer Factorization

1. **Polynomial selection**
   Degree $d > 1$, integer $m \approx n^{1/d}$, radix-m representation of $n = f_d m^d + \cdots + f_1 m + f_0$
   Leads to $f_a(X) = \sum_{i=1}^{d} f_i X^i \in \mathbb{Z}[X]$ with $f_a(m) \equiv 0 \pmod{n}$ (one can do better!) and $f_r(X) = X - m$
   40 CPU years

2. **Relation collection**
   Find co-prime $a, b \in \mathbb{Z} \times \mathbb{Z}_{\geq 0}$ such that $bf_r(a/b)$ and $b^d f_a(a/b)$ factors into small primes (“smooth”)
   1500 CPU years, easy to run in parallel, **memory requirement: $1 \sim 8$ GB**
   (Disclaimer: oversieving)

3. **Matrix step**
   Find even sum of small prime exponent vectors, solve linear dependencies between the relations (find random elements of the null-space of the matrix)
   152 CPU years, not easy to run in parallel, **memory requirement: 180 GB / $\approx$ 900 GB / 180 GB**

4. **Square root**
   Compute the square root of a large element of the number field
   $<$ 1 CPU day

Thorsten Kleinjung et al.: **Factorization of a 768-bit RSA modulus**. *CRYPTO 2010*

Integer Factorization

1. **Polynomial selection**
   Degree $d > 1$, integer $m \approx n^{1/d}$, radix-m representation of $n = f_d m^d + \cdots + f_1 m + f_0$
   Leads to $f_a(X) = \sum_{i=1}^{d} f_i X^i \in \mathbb{Z}[X]$ with $f_a(m) \equiv 0 \pmod{n}$ (one can do better!) and $f_r(X) = X - m$
   40 CPU years

2. **Relation collection**
   Find co-prime $a, b \in \mathbb{Z} \times \mathbb{Z}_{\geq 0}$ such that $bf_r(a/b)$ and $b^d f_a(a/b)$ factors into small primes ("smooth")
   1500 CPU years, easy to run in parallel, **memory requirement: 1 ~ 8 GB**
   (Disclaimer: oversieving)

3. **Matrix step**
   Find even sum of small prime exponent vectors, solve linear dependencies between the relations
   (find random elements of the null-space of the matrix)
   152 CPU years, not easy to run in parallel, **memory requirement: 180 GB / ≈ 900 GB / 180 GB**

4. **Square root**
   Compute the square root of a large element of the number field
   $<$ 1 CPU day

Thorsten Kleinjung et al.: **Factorization of a 768-bit RSA modulus**. *CRYPTO 2010*

Relation Collection

Sieving identifies many pairs (a_i, b_i) such that $bf_r(a/b)$ and $b^d f_a(a/b)$ have many small factors (memory intensive)

Cofactorization (to check: is $bf_r(a/b)$ $\boldsymbol{B_r}$**-smooth** and $b^d f_a(a/b)$ $\boldsymbol{B_a}$**-smooth**)

1. Polynomial evaluation
2. Compositeness test (Miller-Rabin)
3. Trial division
4. Pollard $p-1$ (stage 1 & 2)
5. Elliptic curve factorization method (stage 1 & 2) using twisted Edwards curves

Relation Collection

Sieving identifies many pairs (a_i, b_i) such that $bf_r(a/b)$ and $b^d f_a(a/b)$ have many small factors (memory intensive)

Cofactorization (to check: is $bf_r(a/b)$ $\boldsymbol{B_r}$**-smooth** and $b^d f_a(a/b)$ $\boldsymbol{B_a}$**-smooth**)

1. Polynomial evaluation
2. Compositeness test (Miller-Rabin)
3. Trial division
4. Pollard $p-1$ (stage 1 & 2)
5. Elliptic curve factorization method (stage 1 & 2) using twisted Edwards curves

Modular arithmetic
(Montgomery multiplication)
Exact division

Relation Collection

Sieving identifies many pairs (a_i, b_i) such that $bf_r(a/b)$ and $b^d f_a(a/b)$ have many small factors (memory intensive)

Cofactorization (to check: is $bf_r(a/b)$ $\boldsymbol{B_r}$**-smooth** and $b^d f_a(a/b)$ $\boldsymbol{B_a}$**-smooth**)

1. Polynomial evaluation
2. Compositeness test (Miller-Rabin)
3. Trial division
4. Pollard $p - 1$ (stage 1 & 2)
5. Elliptic curve factorization method (stage 1 & 2) using twisted Edwards curves

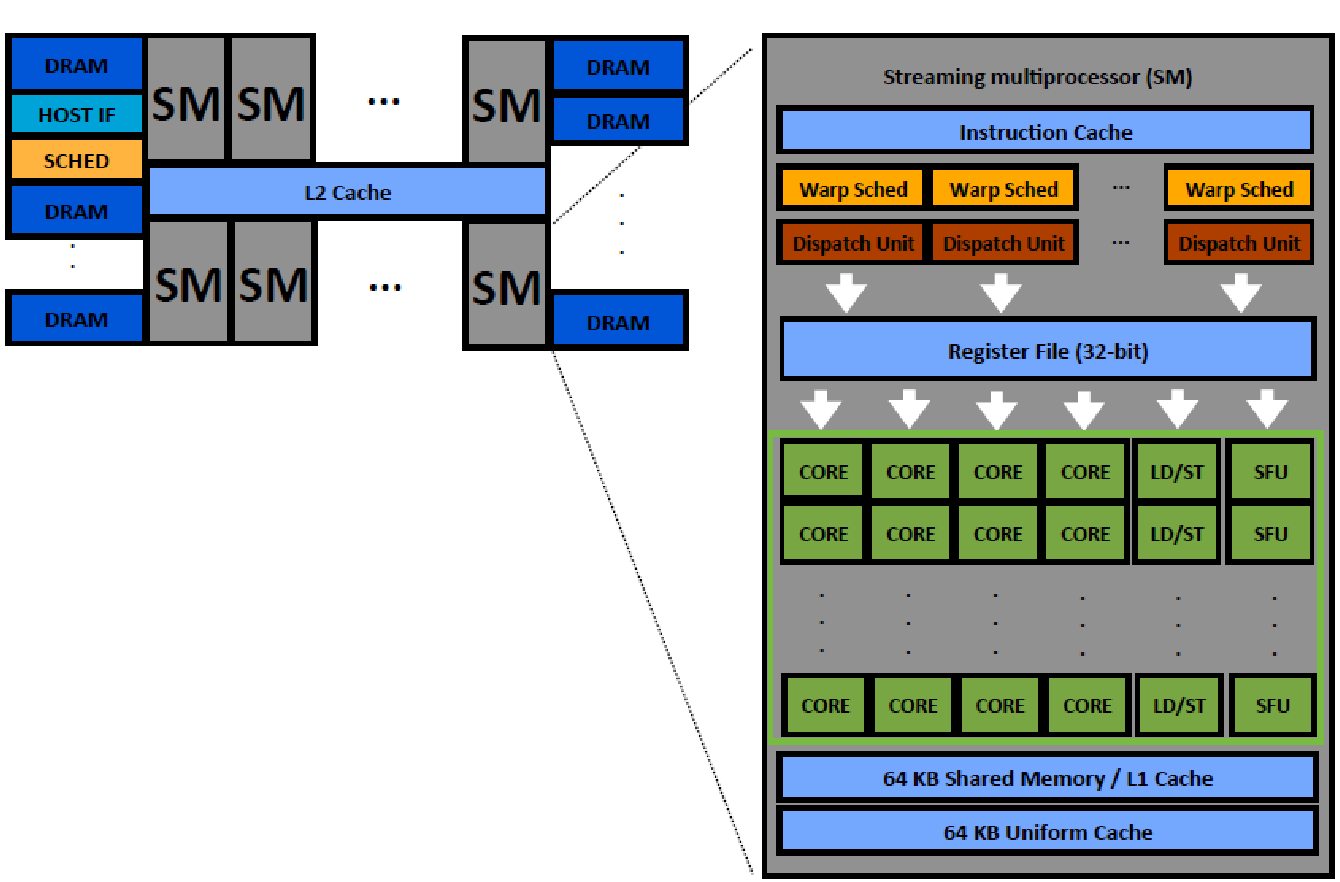
Modular arithmetic
(Montgomery multiplication)
Exact division

Cofactorization can check many pairs (a_i, b_i) simultaneously. Can we offload this to another device?
Possible answer: Graphics processing unit

Andrea Miele, Joppe W. Bos, Thorsten Kleinjung, Arjen K. Lenstra: **Cofactorization on Graphics Processing Units.** *CHES 2014*

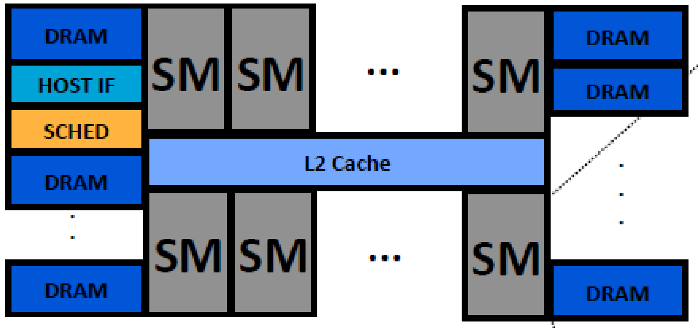
Graphics processing unit (Nvidia platform)

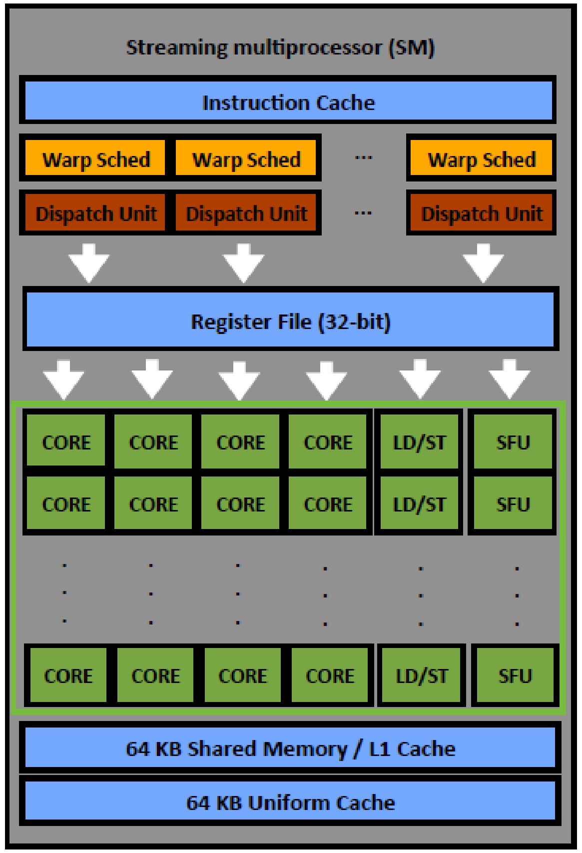
- Modern GPUs are massively parallel 32-bit many-core architectures
- One integer or floating point instruction/clock cycle per thread/core
- Usually run thousands of threads



Graphics processing unit (Nvidia platform)

- Modern GPUs are massively parallel 32-bit many-core architectures
- One integer or floating point instruction/clock cycle per thread/core
- Usually run thousands of threads



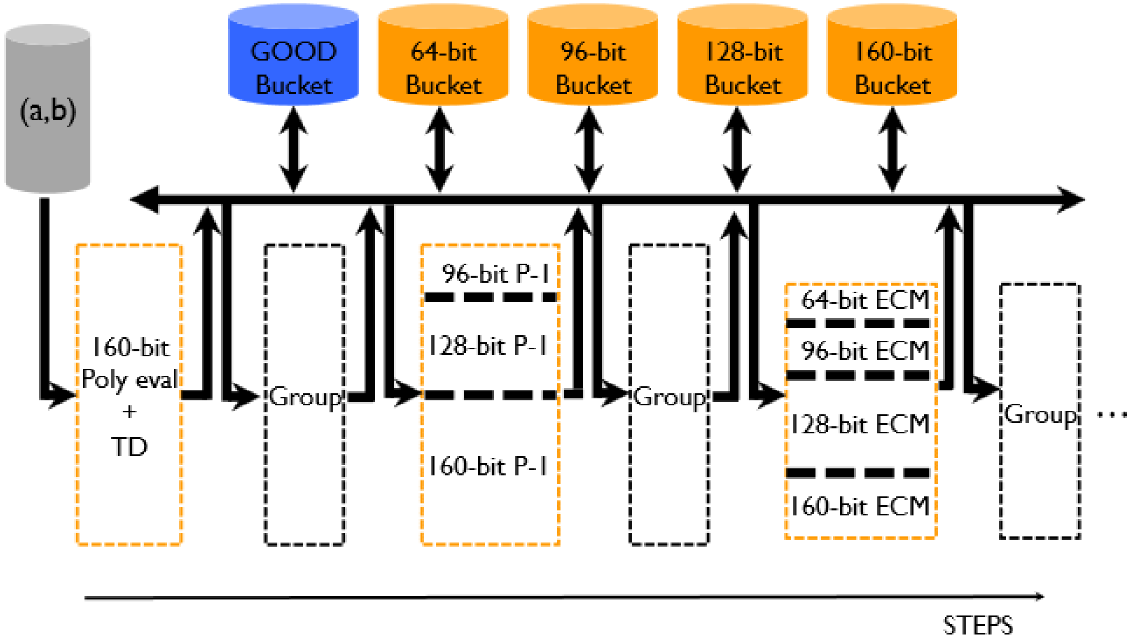


	NVIDIA FERMI (GTX 500 family)	NVIDIA KEPLER (GTX 700 family)
Cores	Up to 512	Up to 2880
SMs	Up to 16	Up to 48
Freq	Up to 1544 MHz	Up to 980 MHz
DRAM	Up to 3GB (192 GB/s)	Up to 6 GB (336 GB/s)

Relation Collection on GPUs

- GPUs have been considered as cryptanalytic coprocessors before (e.g., for ECM)
- First time for the entire relation collection phase

Transfer batch of (a_i, b_i) from CPU to GPU
Repeat in parallel until all (a_i, b_i) have been processed {
  Thread receives (a_i, b_i)
  Polynomial evaluation + Trial Division
  Perform compositeness test and put results in correct bucket
  Pick composite from bucket and perform dedicated Pollard $p - 1$
  Perform compositeness test and put results in correct bucket
  for $(i = 0; i < n; i++)$ {
    Pick composite from bucket and perform dedicated ECM
    Perform compositeness test and put results in correct bucket
  }
}
Transfer good pair to CPU, throw away the rest

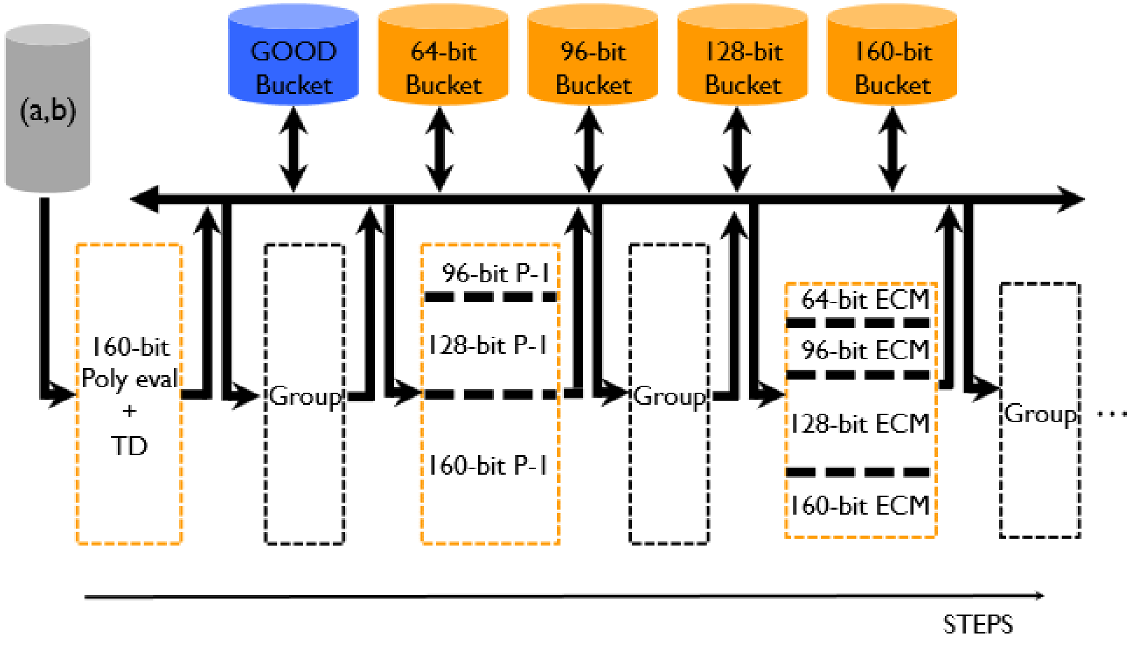


Relation Collection on GPUs

- GPUs have been considered as cryptanalytic coprocessors before (e.g., for ECM)
- First time for the entire relation collection phase

All the Pollard $p-1$ and ECM algorithms run concurrently
→ must use the same parameters
→ **how to optimize?**

Transfer batch of (a_i, b_i) from CPU to GPU
Repeat in parallel until all (a_i, b_i) have been processed {
  Thread receives (a_i, b_i)
  Polynomial evaluation + Trial Division
  Perform compositeness test and put results in correct bucket
  Pick composite from bucket and perform dedicated Pollard $p-1$
  Perform compositeness test and put results in correct bucket
  for $(i = 0; i < n; i++)$ {
    Pick composite from bucket and perform dedicated ECM
    Perform compositeness test and put results in correct bucket
  }
}
Transfer good pair to CPU, throw away the rest



Parameter determination

Observation: Varying the bounds of the Pollard $p - 1$ factoring (within reasonable ranges) does **not noticeably affect the yield**

Explanation: All missed prime factors are found by the subsequent ECM attempts.

Parameter determination

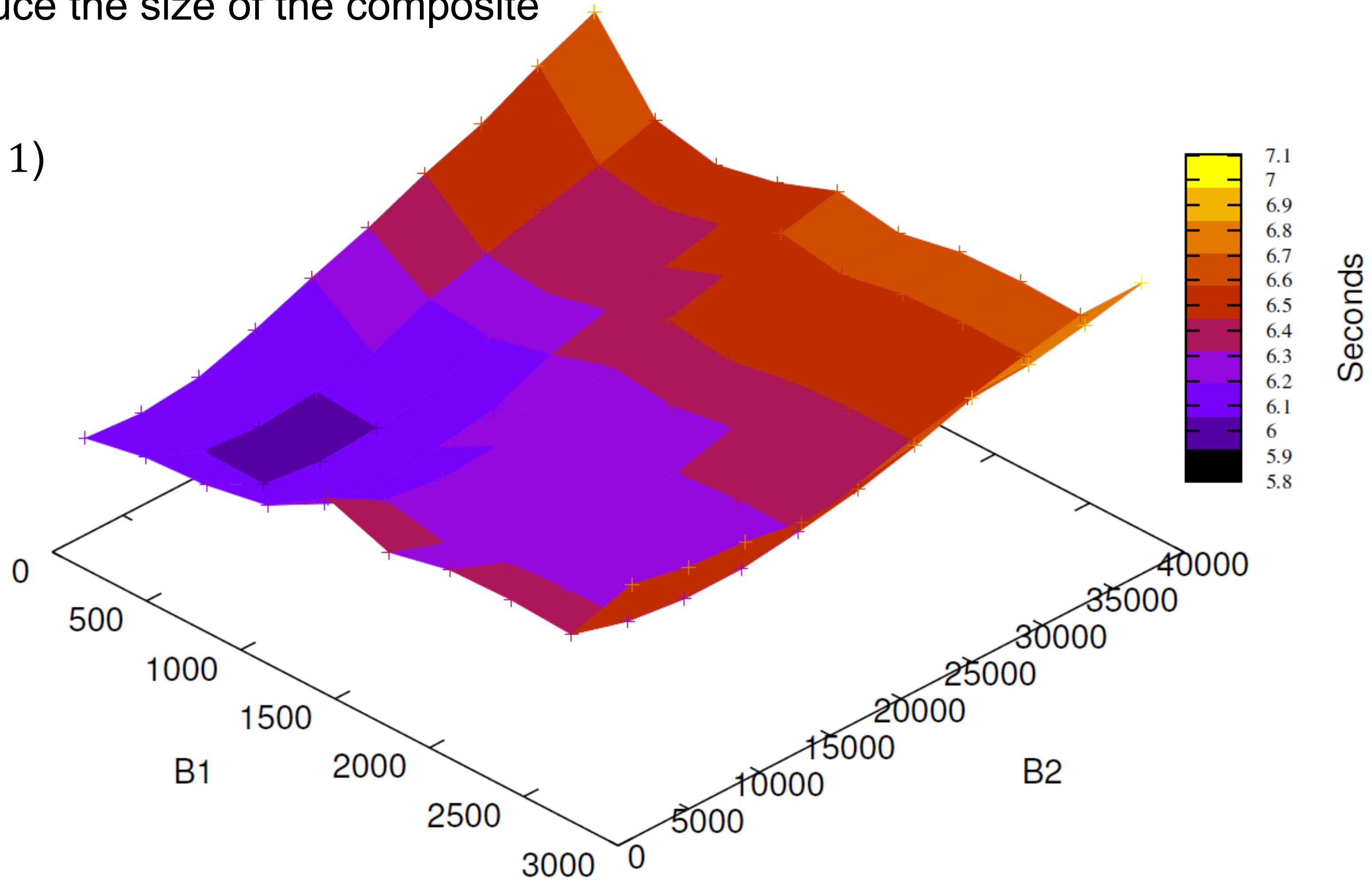
Observation: Varying the bounds of the Pollard $p-1$ factoring (within reasonable ranges) does **not noticeably affect the yield**
Explanation: All missed prime factors are found by the subsequent ECM attempts.

However, early removal of small primes reduce the size of the composite
→ reducing the ECM run time
→ **reduces the overall run time**
(if not too much time is spent on Pollard $p-1$)

Parameter determination

Observation: Varying the bounds of the Pollard $p-1$ factoring (within reasonable ranges) does **not noticeably affect the yield**
Explanation: All missed prime factors are found by the subsequent ECM attempts.

However, early removal of small primes reduce the size of the composite
→ reducing the ECM run time
→ **reduces the overall run time**
(if not too much time is spent on Pollard $p-1$)



The time difference for the entire cofactorization when the yield is *fixed* at 95% when varying the B_1 and B_2 bounds for Pollard $p-1$ on the rational side

Results

CPU used: **Intel i7-3770K CPU**, with 4 cores, 3.5 GHz with 16 GB of memory
GPU used: **NVIDIA GeForce GTX 580**, with 512 CUDA cores, 1.5 GHz with 1.5 GB of global memory

Target number: RSA-768 (same polynomial as used for the factorization)

Processing multiple special primes with desired yield 99%.

Large primes	Number of pairs after sieving	Setting	Total seconds	Relations found	Relations per second
4	$\approx 5 \cdot 10^7$	CPU only	1602	6855	4.28
		GPU + CPU			

Results

CPU used: **Intel i7-3770K CPU**, with 4 cores, 3.5 GHz with 16 GB of memory
GPU used: **NVIDIA GeForce GTX 580**, with 512 CUDA cores, 1.5 GHz with 1.5 GB of global memory

Target number: RSA-768 (same polynomial as used for the factorization)

Processing multiple special primes with desired yield 99%.

Large primes	Number of pairs after sieving	Setting	Total seconds	Relations found	Relations per second
4	$\approx 5 \cdot 10^7$	CPU only	1602	6855	4.28
		GPU + CPU	1300	8302	6.39

Results

CPU used: **Intel i7-3770K CPU**, with 4 cores, 3.5 GHz with 16 GB of memory
GPU used: **NVIDIA GeForce GTX 580**, with 512 CUDA cores, 1.5 GHz with 1.5 GB of global memory

Target number: RSA-768 (same polynomial as used for the factorization)

Processing multiple special primes with desired yield 99%.

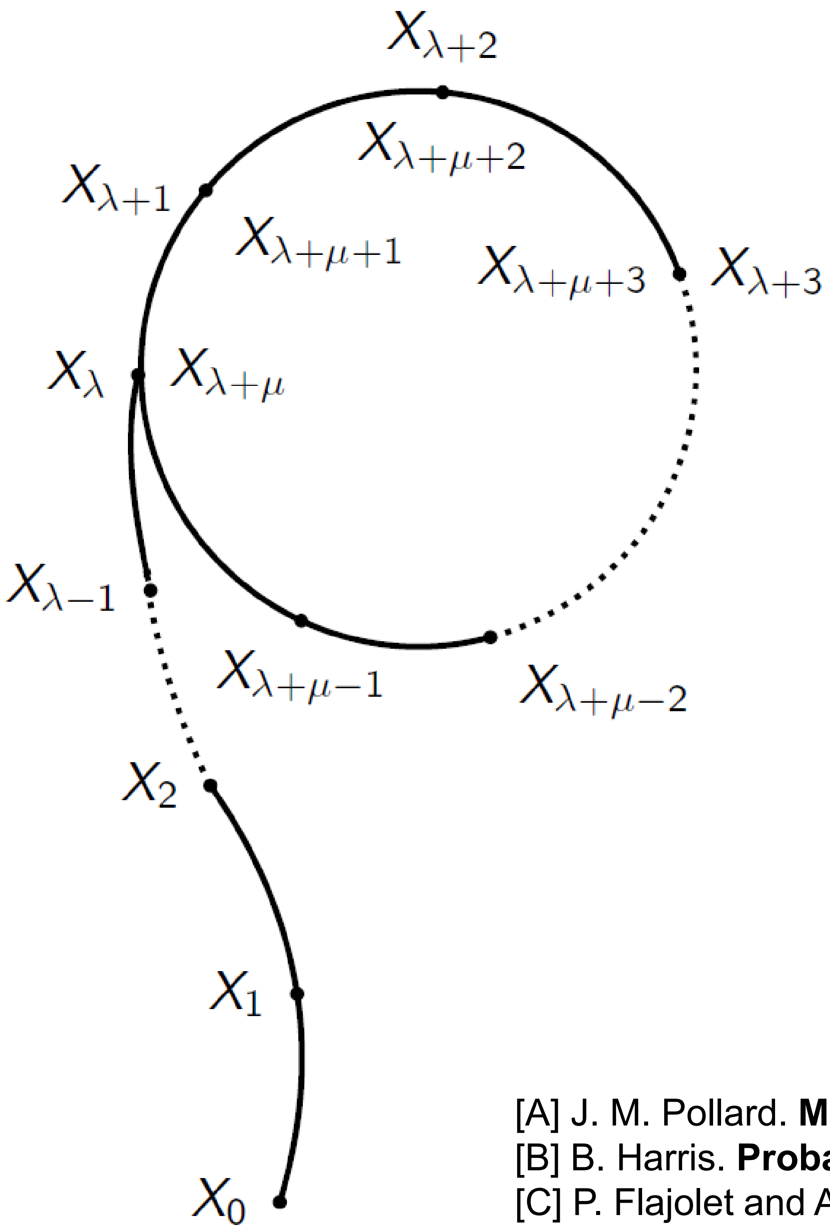
Large primes	Number of pairs after sieving	Setting	Total seconds	Relations found	Relations per second
4	$\approx 5 \cdot 10^7$	CPU only	1602	6855	4.28
		GPU + CPU	1300	8302	6.39

- ✓ Latency down by a factor 1.23
- ✓ Number of relations found up by 21.1%
- ✓ Yield / second up by a factor 1.49x

Not considered

- ➢ Purchase cost GPU versus CPU
- ➢ Power comparison GPU versus CPU

Pollard Rho

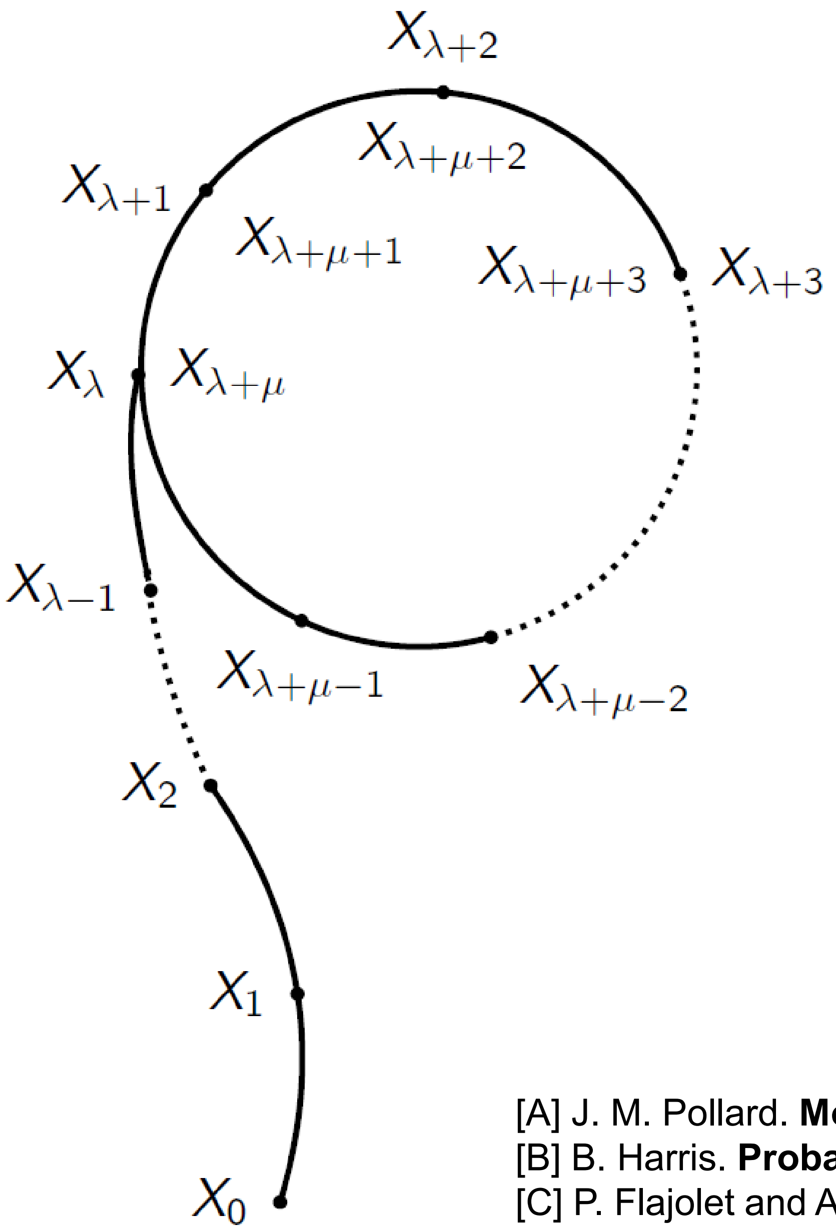


Given prime $p > 3,\ G \in E(\mathbf{F}_p)$ of order $n,\ H \in \langle G \rangle$
find integer m such that $mG = H$

- Originally an integer factorization method (1975)
- Three years later turned into approach for solving dlps [A]
- Perform a pseudo-random walk through the set of points (length tail ≈ length cycle ≈ $\sqrt{\pi n/4}$ [B,C])

[A] J. M. Pollard. **Monte Carlo methods for index computation (mod *p*)**. *Mathematics of Computation*, 1978
[B] B. Harris. **Probability distributions related to random mappings**. *The Annals of Mathematical Statistics*, 1960
[C] P. Flajolet and A. M. Odlyzko. **Random mapping statistics**. *Eurocrypt 1989*

Pollard Rho



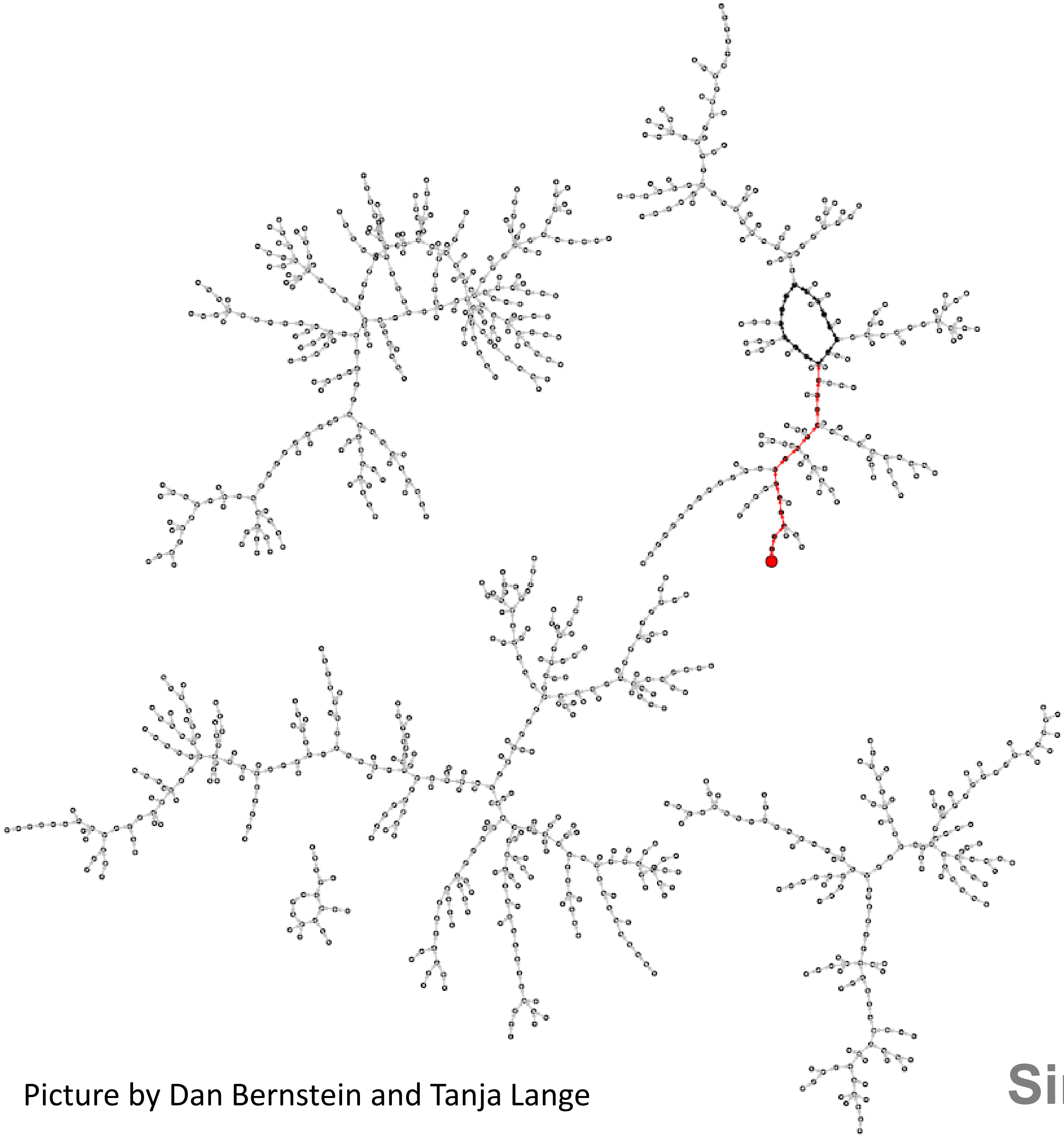
Given prime $p > 3$, $G \in E(\mathbf{F}_p)$ of order n, $H \in \langle G \rangle$
find integer m such that $mG = H$

- Originally an integer factorization method (1975)
- Three years later turned into approach for solving dlps [A]
- Perform a pseudo-random walk through the set of points
  (length tail ≈ length cycle ≈ $\sqrt{\pi n / 4}$ [B,C])

r-adding walk

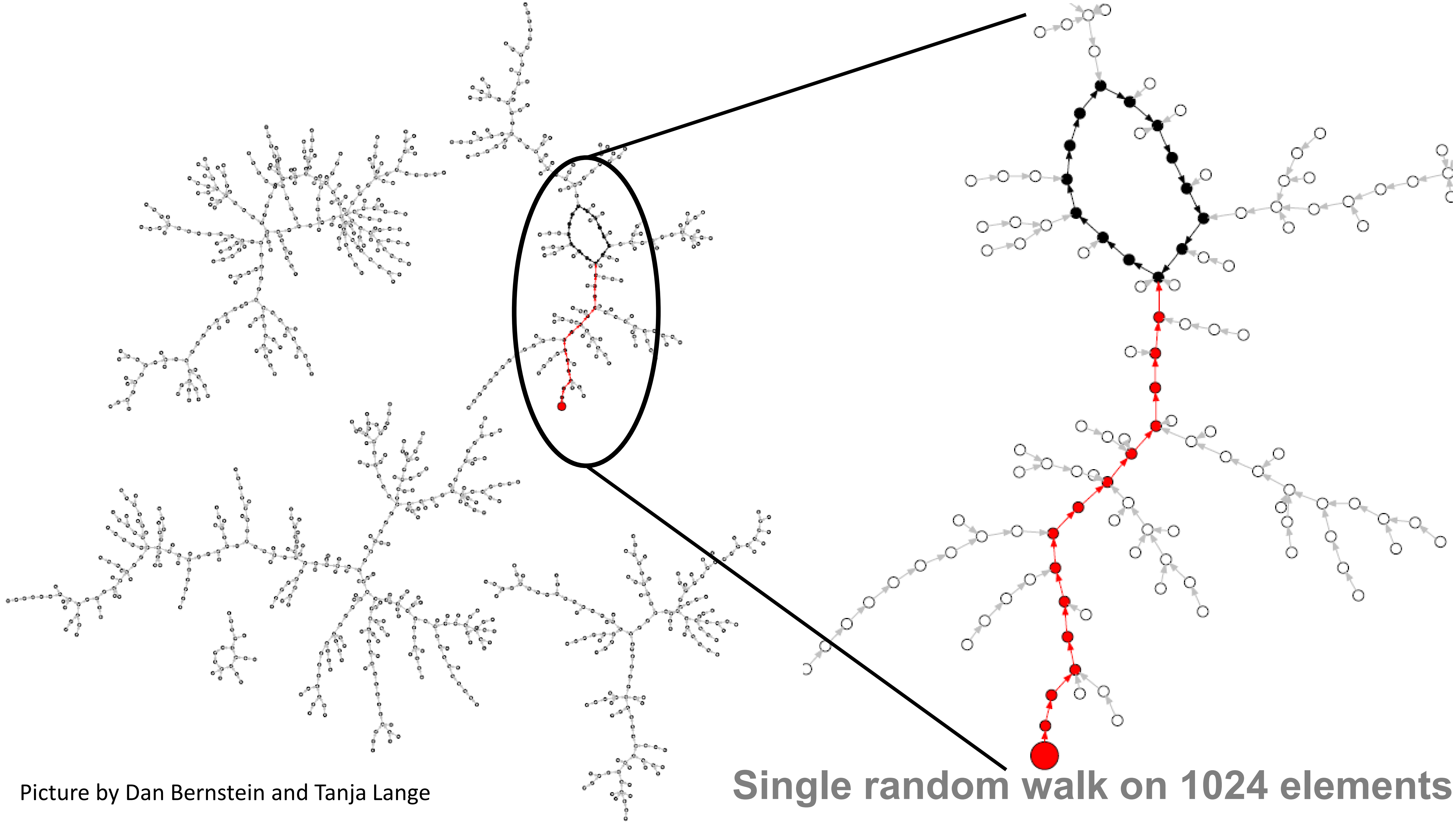
- Define index function $\ell : \langle G \rangle \mapsto [0, \ldots, r-1]$
- ℓ-induced r-partition $\langle G \rangle = \cup_{i=0}^{r-1} g_i$ where $g_i = \{x : x \in \langle G \rangle, \ell(x) = i\}$
  (all of approximately the same cardinality)
- Select random multipliers u_i and v_i to define r points $f_i = u_i G + v_i H$
- One step is defined as: $P_{i+1} = P_i + f_{\ell(P_i)}$

[A] J. M. Pollard. **Monte Carlo methods for index computation (mod *p*)**. *Mathematics of Computation*, 1978
[B] B. Harris. **Probability distributions related to random mappings**. *The Annals of Mathematical Statistics*, 1960
[C] P. Flajolet and A. M. Odlyzko. **Random mapping statistics**. *Eurocrypt 1989*



Picture by Dan Bernstein and Tanja Lange

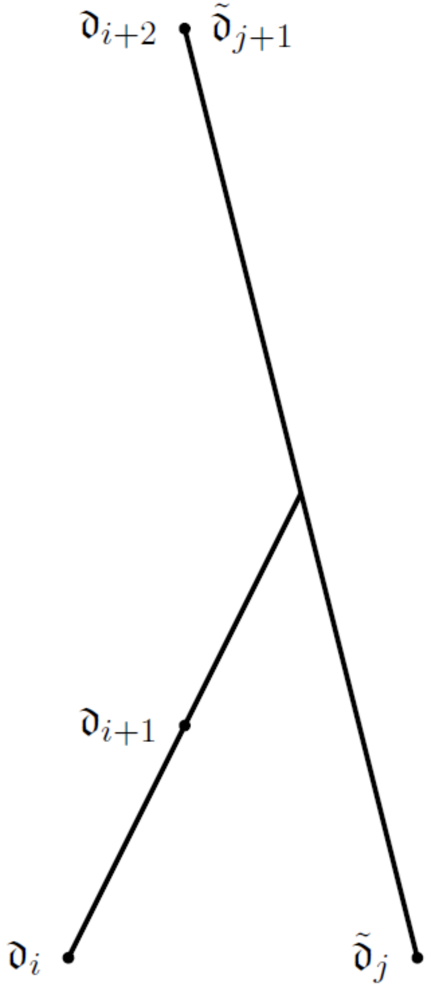
Single random walk on 1024 elements



Picture by Dan Bernstein and Tanja Lange

Single random walk on 1024 elements

Parallelization of Pollard Rho



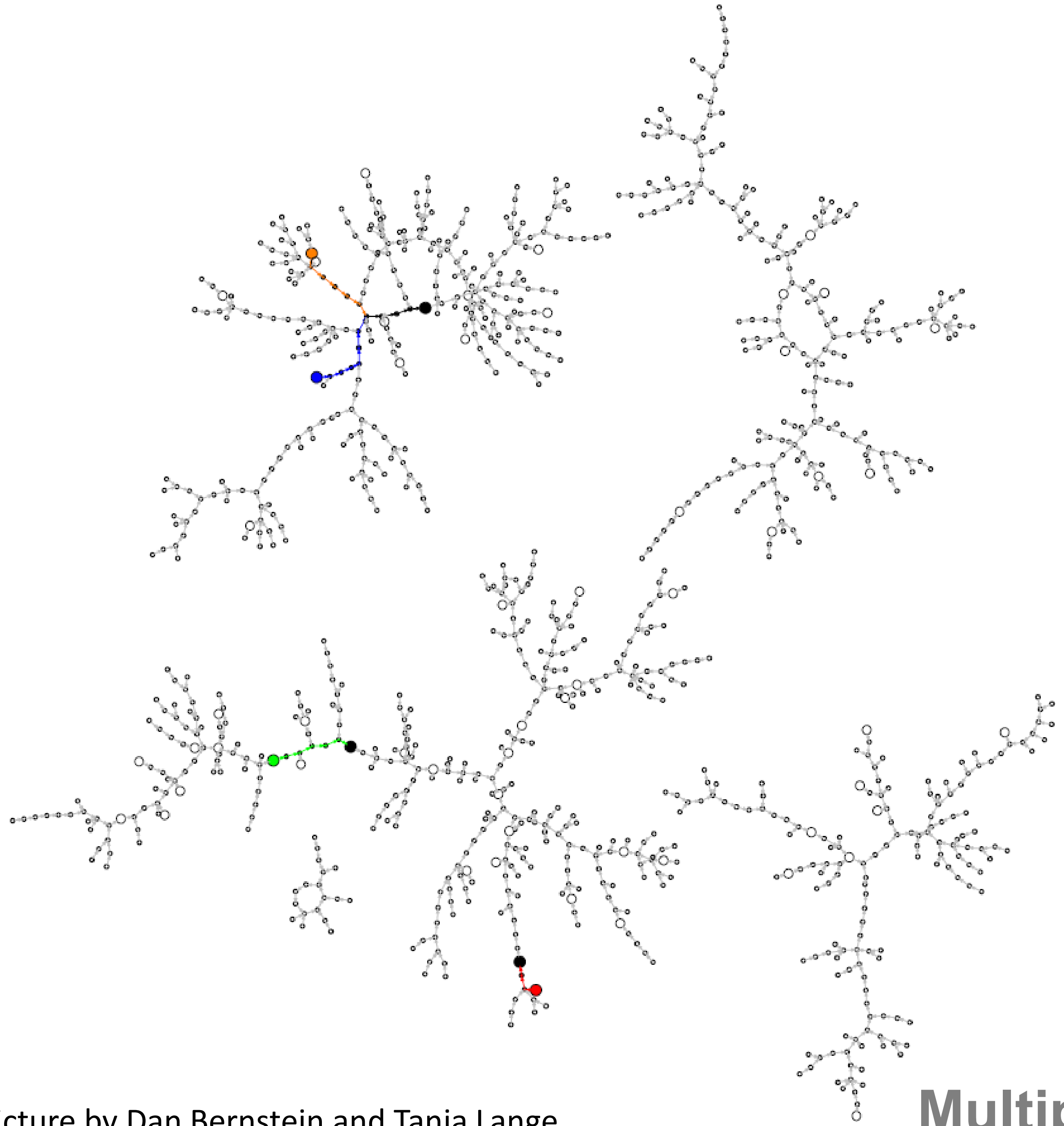
<u>Can we compute Pollard rho using multiple computational resources?</u>

What happens if we run Pollard rho m times in parallel?
→ $\sqrt{m}$ speedup

Can we do better?

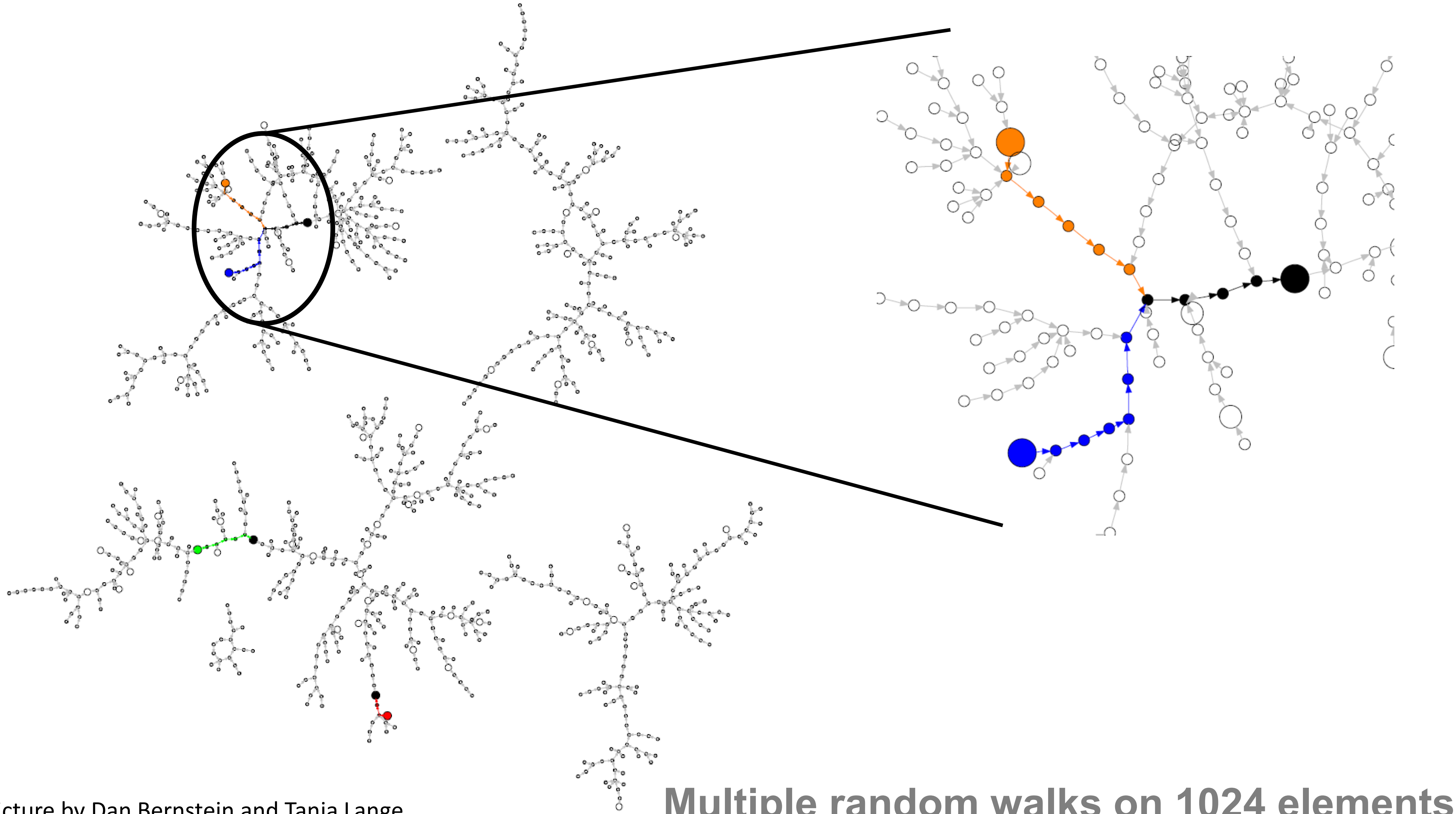
Let the m parallel instance "work together"
→ share some points (distinguished points)
(Collected in a central database, collision search is performed here)
→ factor m speedup

P. C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 1999



Picture by Dan Bernstein and Tanja Lange

Multiple random walks on 1024 elements



Picture by Dan Bernstein and Tanja Lange

Multiple random walks on 1024 elements

Using Pollard Rho to solve ECDLPs

Advantages of Pollard rho

- ✓ Very low memory requirement (can run virtually on any device!)
- ✓ Can store a batch of distinguished points locally and sent them to the central database in batches.

What devices can we use to solve ECDLPs?


[free busy full down disabled absent]
$ cluster-nodes
Master: M
Slave: S

Using Pollard Rho to solve ECDLPs

Advantages of Pollard rho

- ✓ Very low memory requirement (can run virtually on any device!)
- ✓ Can store a batch of distinguished points locally and sent them to the central database in batches.

What devices can we use to solve ECDLPs?

- Joppe W. Bos, Marcelo E. Kaihara, Thorsten Kleinjung, Arjen K. Lenstra, Peter L. Montgomery: **Solving a 112-bit Prime Elliptic Curve Discrete Logarithm Problem on Game Consoles using Sloppy Reduction**. *International Journal of Applied Cryptography*, 2012

Using Pollard Rho to solve <u>ECDLPs</u>

<u>Advantages of Pollard rho</u>

- ✓ Very low memory requirement (can run virtually on any device!)
- ✓ Can store a batch of distinguished points locally and sent them to the central database in batches.

What devices can we use to solve ECDLPs?

- Joppe W. Bos, Marcelo E. Kaihara, Thorsten Kleinjung, Arjen K. Lenstra, Peter L. Montgomery: **Solving a 112-bit Prime Elliptic Curve Discrete Logarithm Problem on Game Consoles using Sloppy Reduction**. *International Journal of Applied Cryptography*, 2012
- Daniel V. Bailey, Lejla Batina, Daniel J. Bernstein, Peter Birkner, Joppe W. Bos, Hsieh-Chung Chen, Chen-Mou Cheng, Gauthier van Damme, Giacomo de Meulenaer, Luis Julian Dominguez Perez, Junfeng Fan, Tim Güneysu, Frank Gurkaynak, Thorsten Kleinjung, Tanja Lange, Nele Mentens, Ruben Niederhagen, Christof Paar, Francesco Regazzoni, Peter Schwabe, Leif Uhsadel, Anthony Van Herrewege, Bo-Yin Yang: **Breaking ECC2K-130.** *Cryptology ePrint Archive, Report 2009/541*, IACR, 2009

Pollard Rho on Mobile Devices

Fun exercise (back in 2010) use Pollard rho with negation map to solve 115-bit ECDLP

Apple iPad family (2015)	250 million sold
Apple iPhone family (2015)	700 million sold
Android active monthly users (2014)	1000 million

Ipad (2010) Apple A4 (= ARM Cortext A8, 1.0 GHz, single-core)	$530 \cdot 10^3$ iterations per second

Idea use their compute power when they are charging (night time)

Effort: $\sqrt{\frac{\pi \cdot 2^{115}}{4}} \approx 1.8 \cdot 10^{17}$ iterations expected $\rightarrow 10^4$ Ipad years

Pollard Rho on Mobile Devices

Fun exercise (back in 2010) use Pollard rho with negation map to solve 115-bit ECDLP

Apple iPad family (2015)	250 million sold
Apple iPhone family (2015)	700 million sold
Android active monthly users (2014)	1000 million

Ipad (2010) Apple A4 (= ARM Cortext A8, 1.0 GHz, single-core)	$530 \cdot 10^3$ iterations per second

Idea use their compute power when they are charging (night time)

Effort: $\sqrt{\frac{\pi \cdot 2^{115}}{4}} \approx 1.8 \cdot 10^{17}$ iterations expected $\rightarrow 10^4$ Ipad years $\overset{?}{\rightarrow} 10^3$ modern Ipad years

Newer models have multiple cores, 64-bit architecture, higher clock-speeds + better implementation

Celliptic

celliptic.epfl.ch

Search

Most Visited Getting Started Suggested Sites Web Slice Gallery

Grid computing on the move.

Grid computing system designed for cryptographic computation only based on smartphones and tablets.

Participate

100% FREE

Lattice-based cryptosystems -- Motivation

- Shortest Vector Problem (SVP) used as a theoretical foundation in many PQ-crypto schemes
  - Lattice based encryption / signature schemes, fully homomorphic encryption
  - Often compute in an ideal lattice for performance reasons

$$R = \mathbb{Z}[X]/(X^n + 1)$$

- Exact SVP is known to be NP-hard under randomized reductions (In most applications approximations are enough)

- How efficient can we find short vectors in ideal lattices?

SVP solvers

Asymptotic rigorous proven runtimes (ignoring poly-log factors in the exponent)		
	Time	Memory
Voronoi	2^{2n}	2^{n}
List Sieve	$2^{2.465n}$	$2^{1.233n}$
Enumeration	$2^{O(n\log(n))}$	$\mathrm{poly}(n)$

SVP solvers

Asymptotic rigorous proven runtimes (ignoring poly-log factors in the exponent)		
	Time	Memory
Voronoi	2^{2n}	2^{n}
List Sieve	$2^{2.465n}$	$2^{1.233n}$
Enumeration	$2^{O(n\log(n))}$	$\mathrm{poly}(n)$

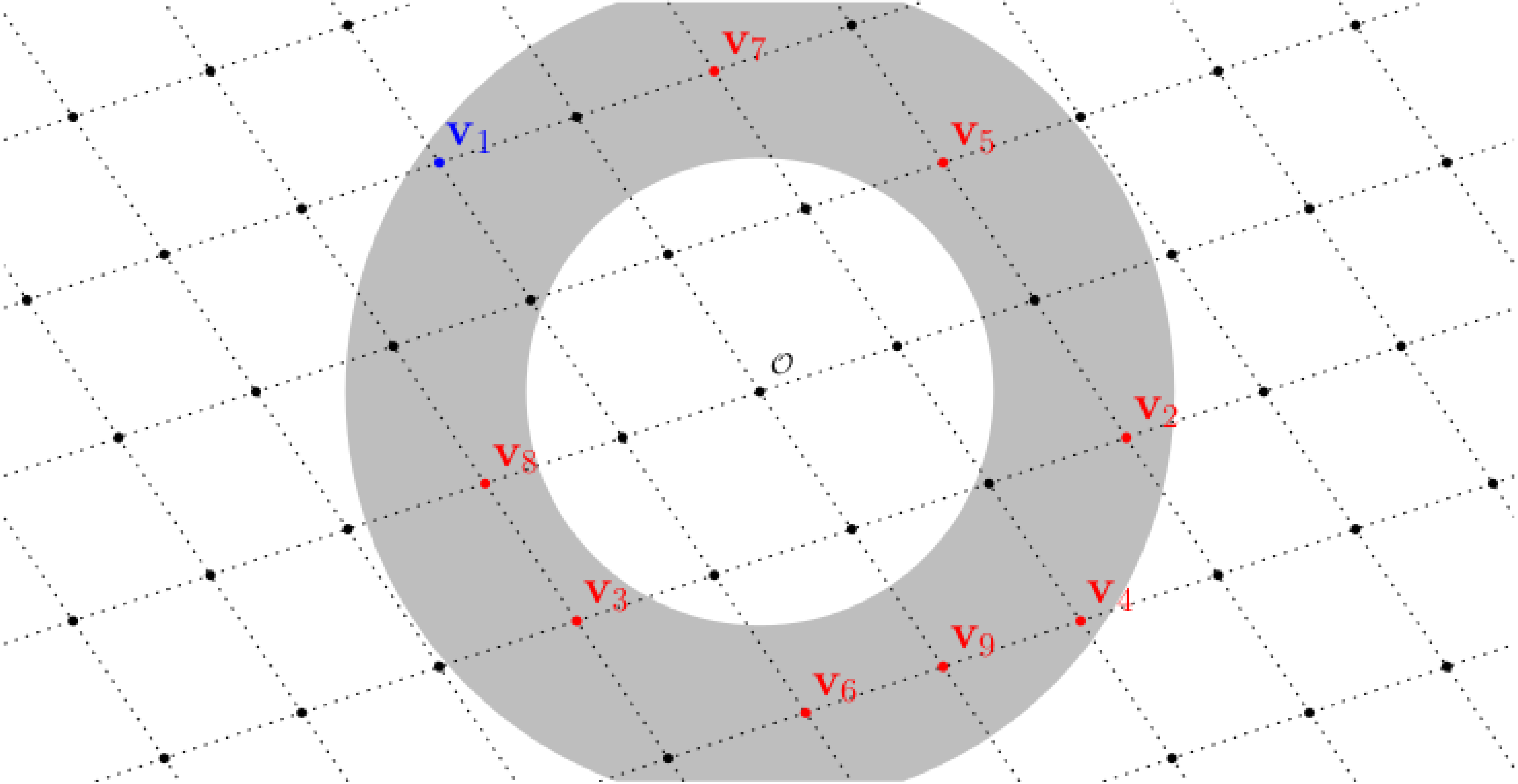
Asymptotic heuristic runtimes		
BKZ 2.0	$n \cdot N \cdot \mathrm{svp}(k)$	$\mathrm{poly}(n)$
+ Enumeration with extreme pruning	$n \cdot N \cdot 2^{O(k^2)}$	$\mathrm{poly}(n)$
Gauss Sieve	"$2^{0.48n}$"	$2^{0.2075n}$
Decomposition	$2^{0.3374n}$	$2^{0.2925n}$
Voronoi	"up to dimension 8"	

SVP solvers

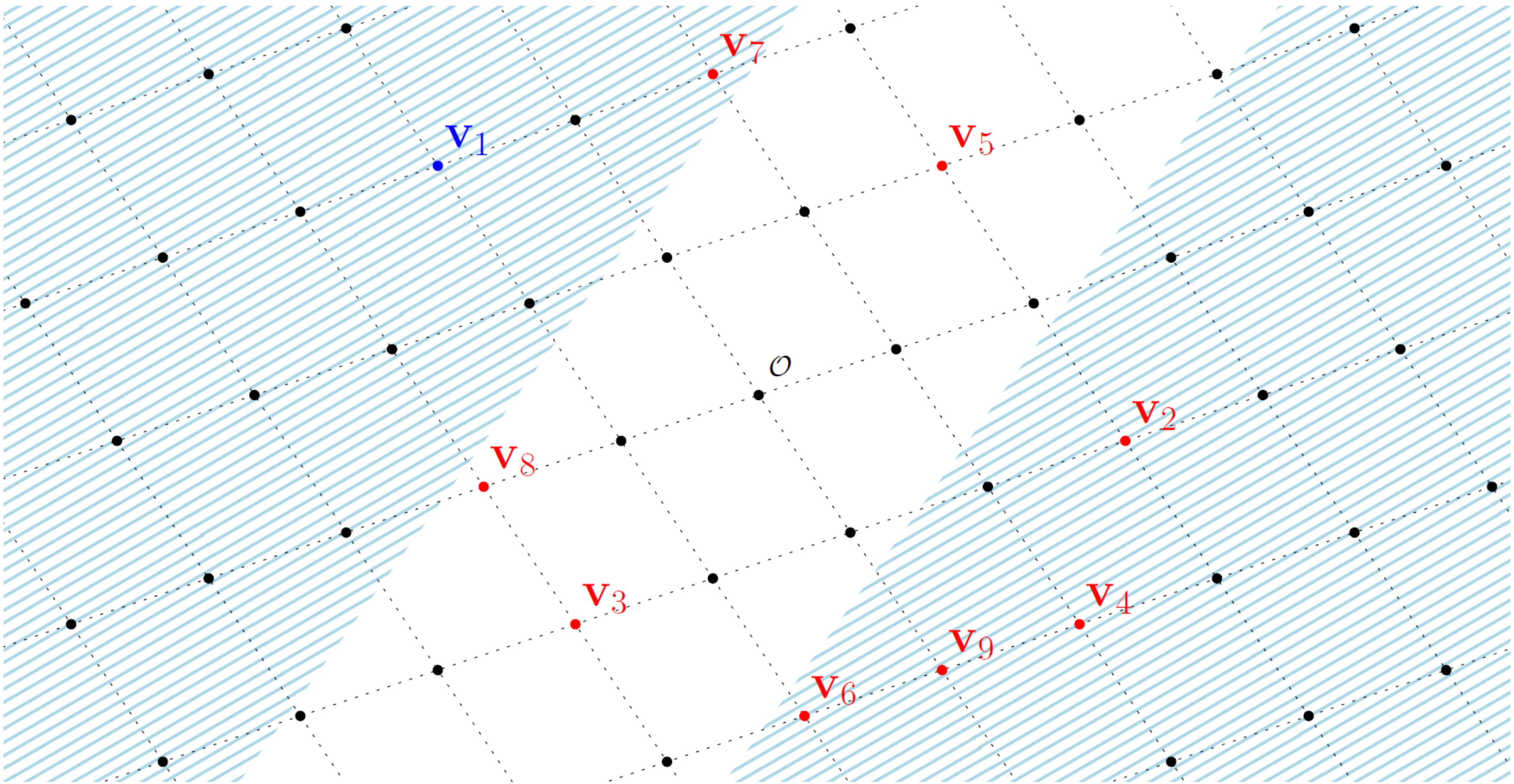
Asymptotic rigorous proven runtimes (ignoring poly-log factors in the exponent)		
	Time	Memory
Voronoi	2^{2n}	2^{n}
List Sieve	$2^{2.465n}$	$2^{1.233n}$
Enumeration	$2^{O(n\log(n))}$	$\mathrm{poly}(n)$

Asymptotic heuristic runtimes		
BKZ 2.0	$n \cdot N \cdot \mathrm{svp}(k)$	$\mathrm{poly}(n)$
+ Enumeration with extreme pruning	$n \cdot N \cdot 2^{O(k^2)}$	$\mathrm{poly}(n)$
Gauss Sieve	"$2^{0.48n}$"	$2^{0.2075n}$
Decomposition	$2^{0.3374n}$	$2^{0.2925n}$
Voronoi	"up to dimension 8"	

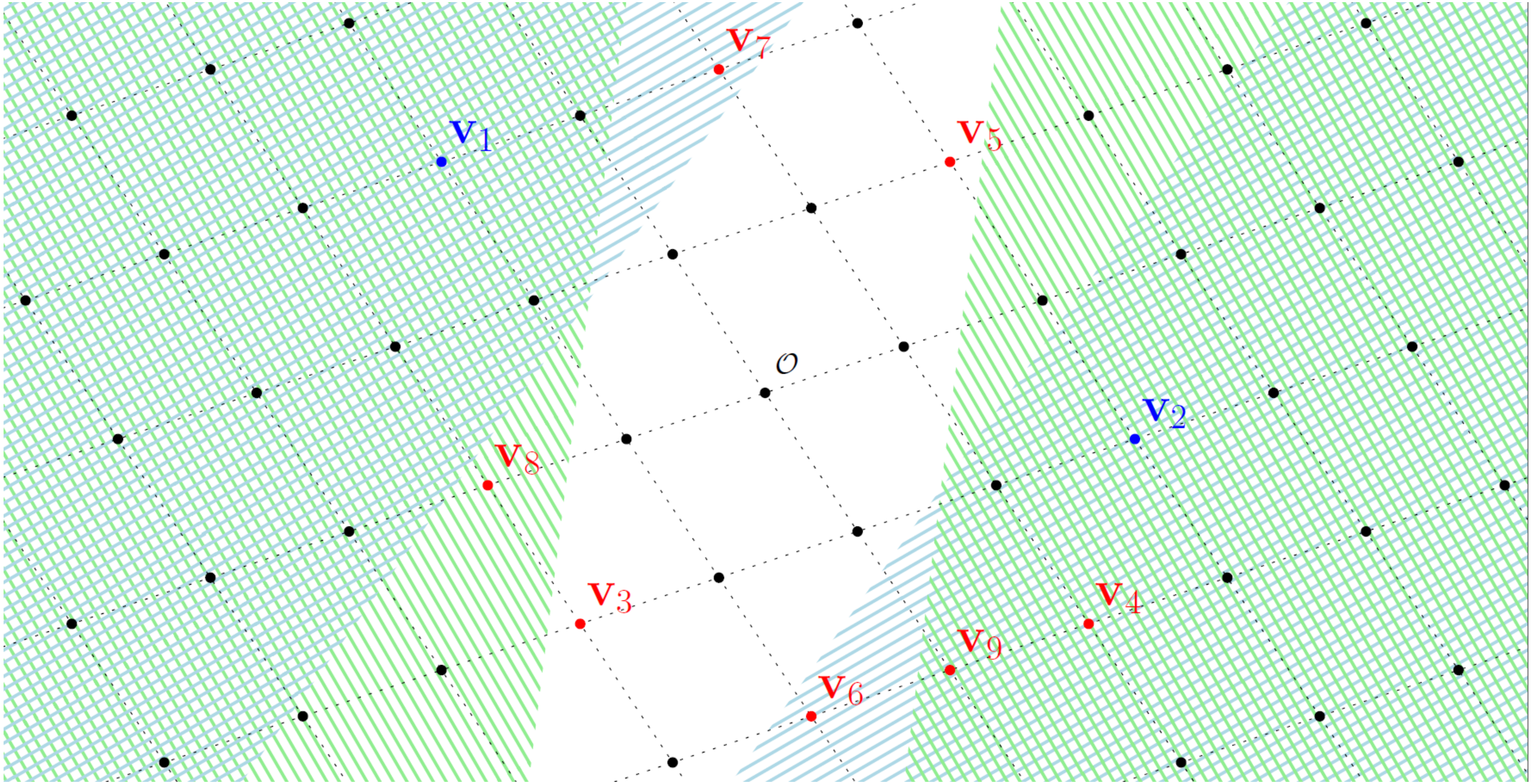
Only sieving algorithms take advantage of the ideal lattice structure



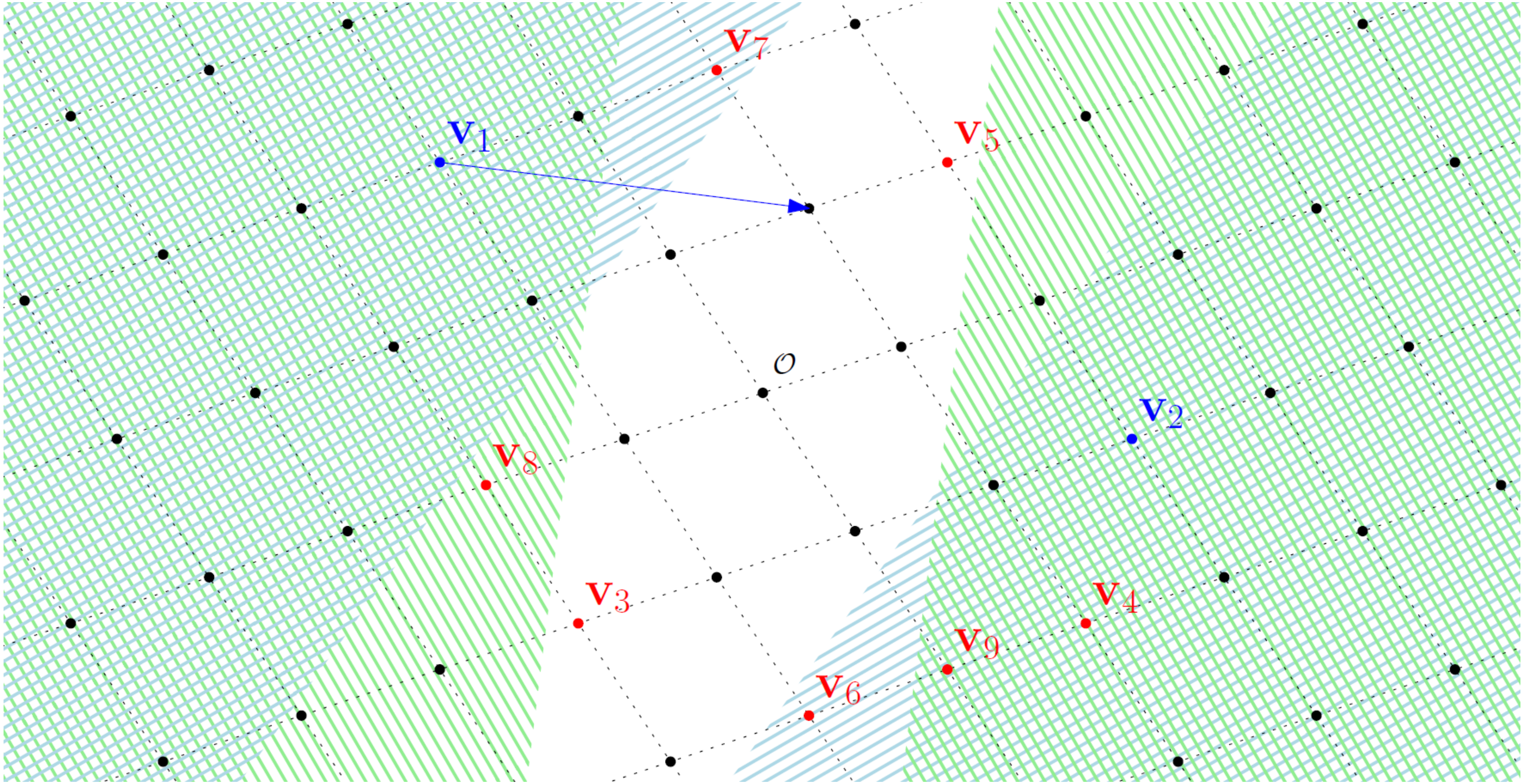
Sample a list of vectors and Gauss reduce all vectors with respect to each other



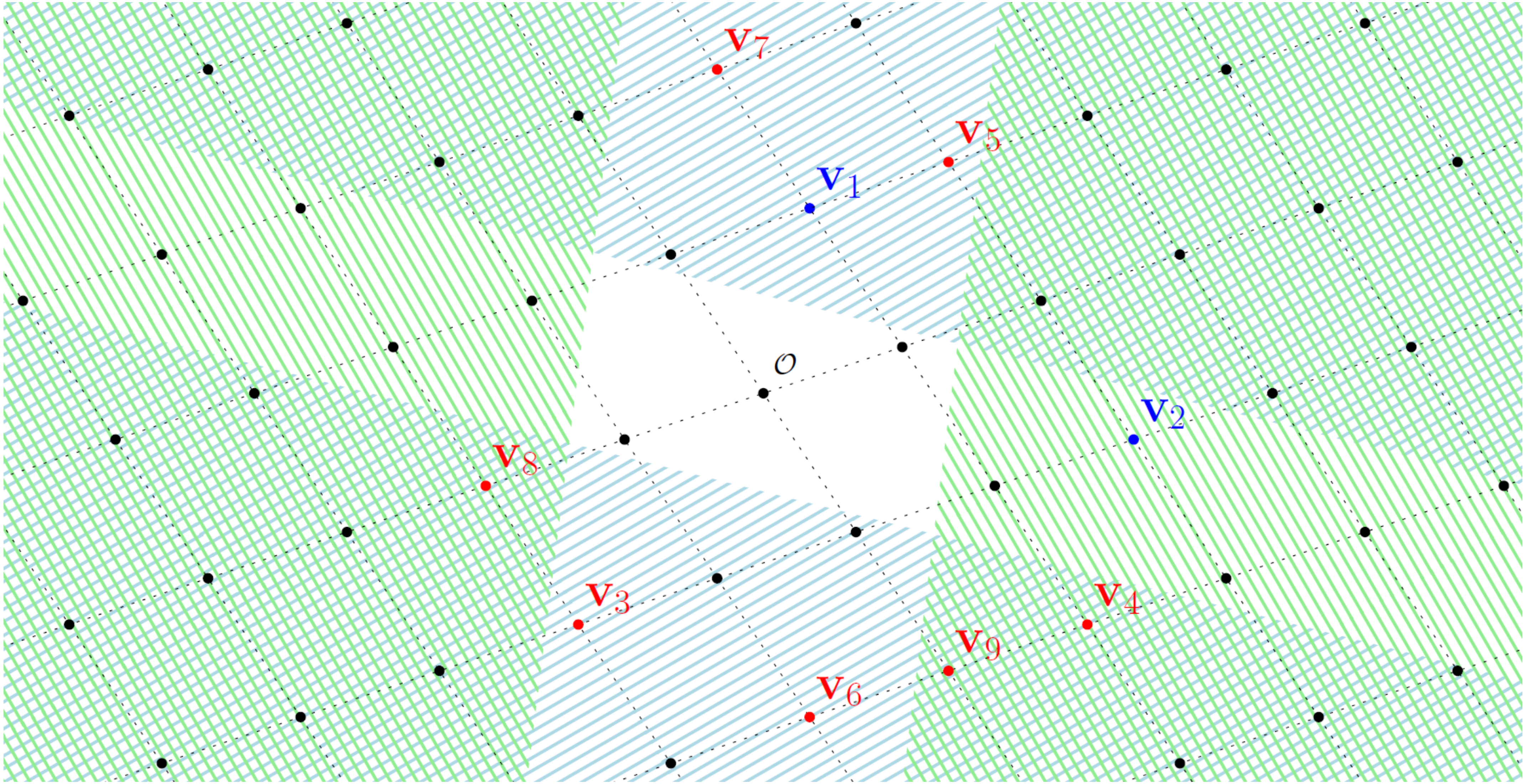
Each vector corresponds to two half spaces.
If a vector is in half-space of another previous vector, it can be reduced.



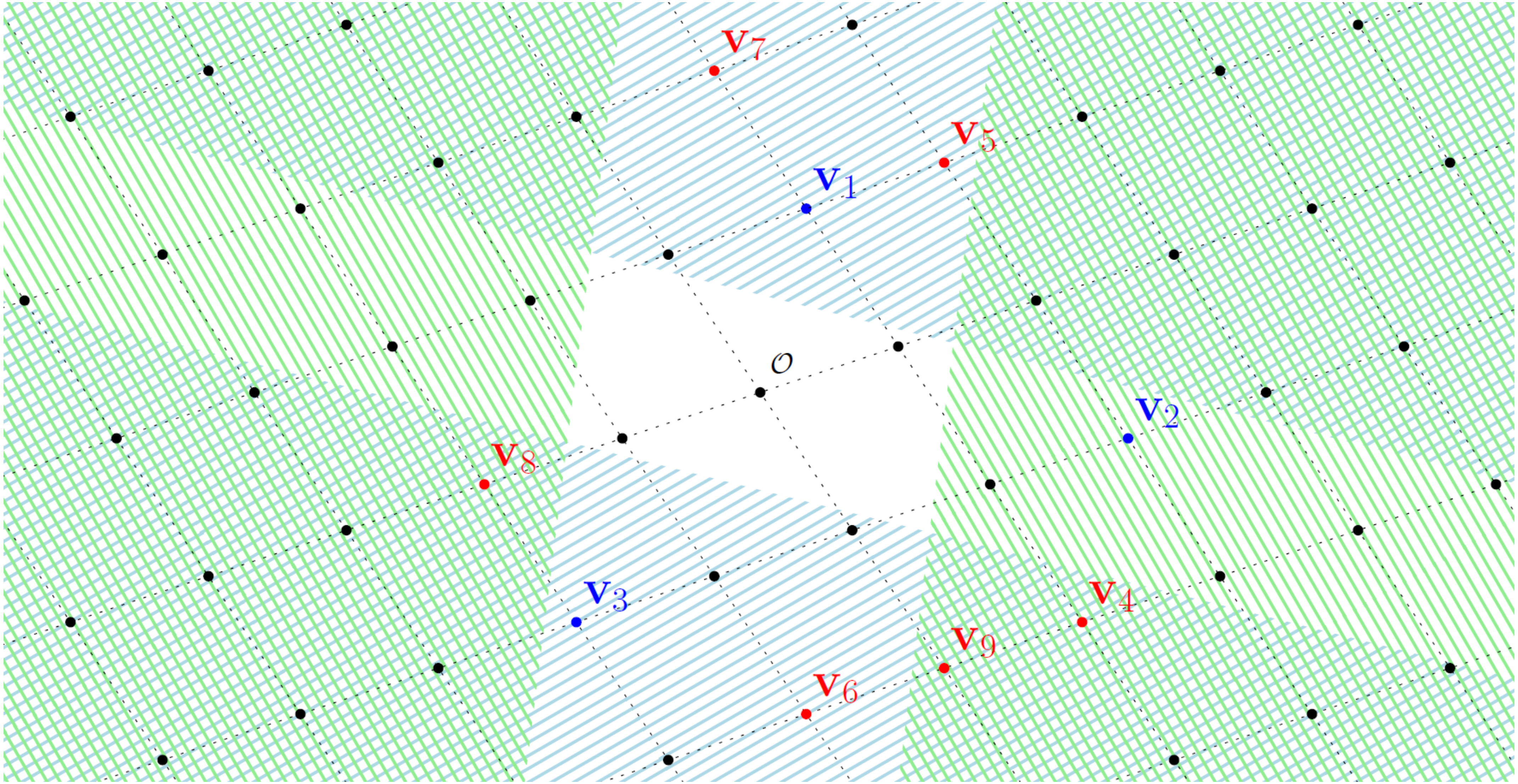
Each vector corresponds to two half spaces.
If a vector is in a half-space of another previous vector, it can be reduced.



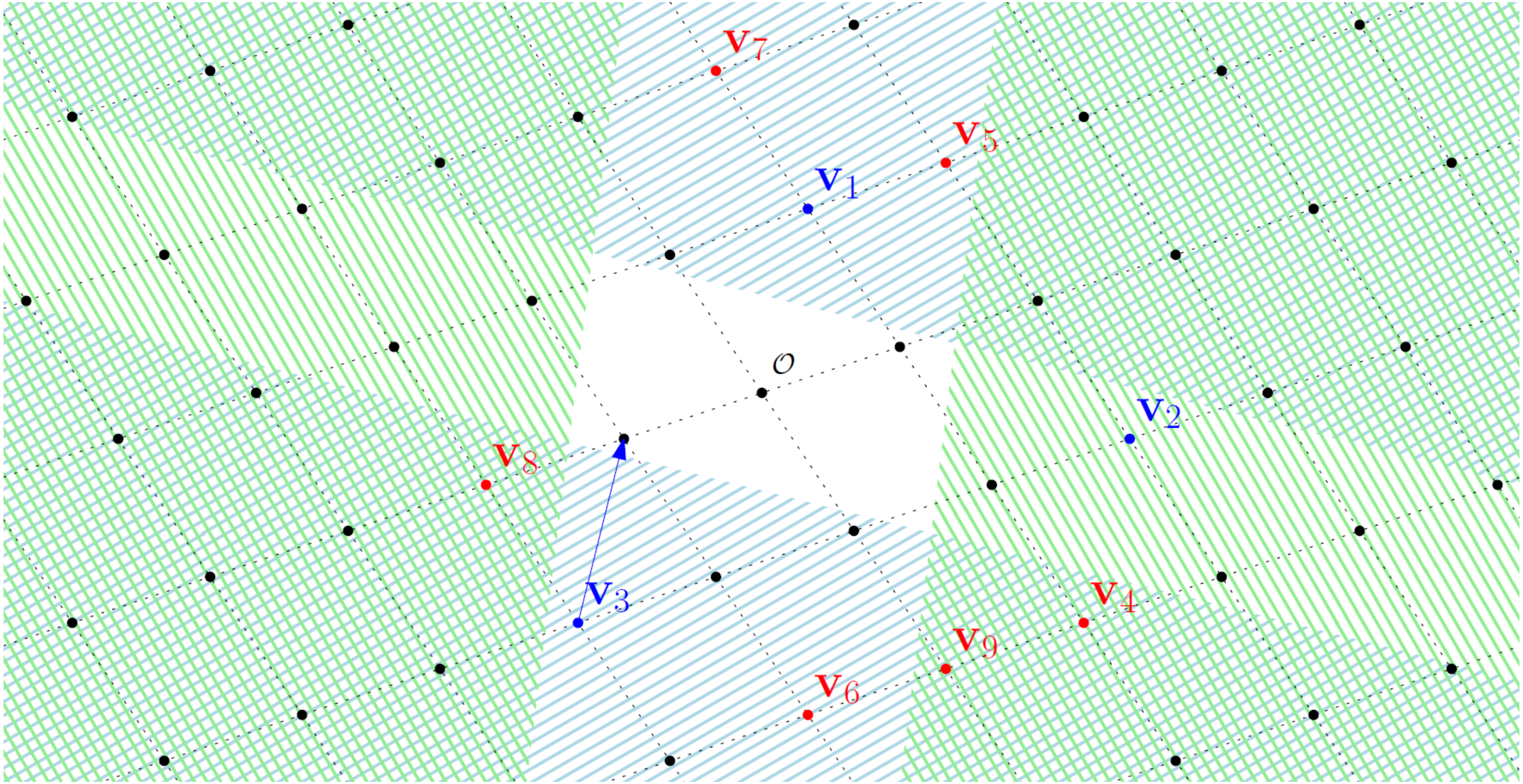
When two vectors can reduce each other, the shorter one reduces the longer one.



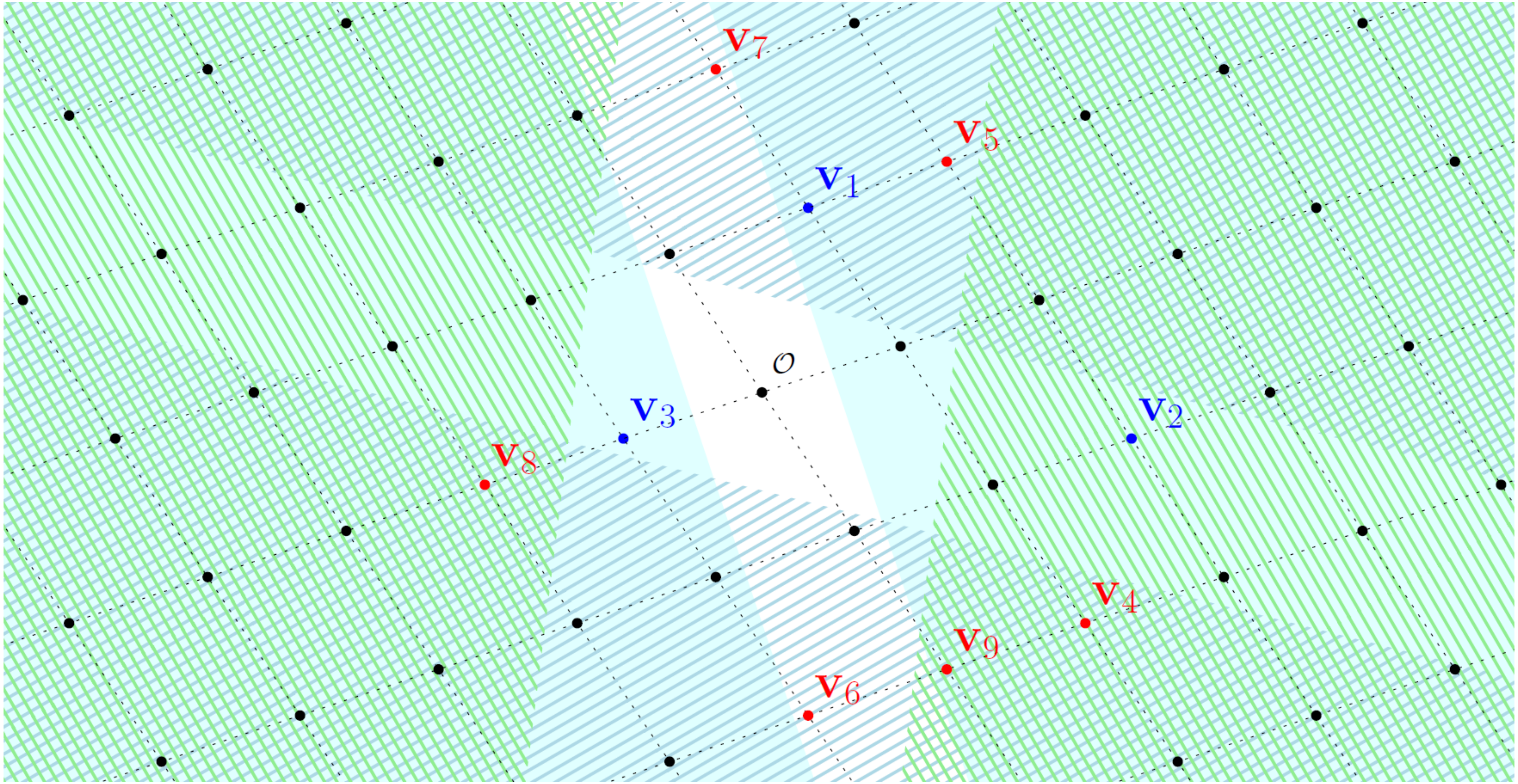
When two vectors can reduce each other, the shorter one reduces the longer one. The half-spaces increasingly cover more space.



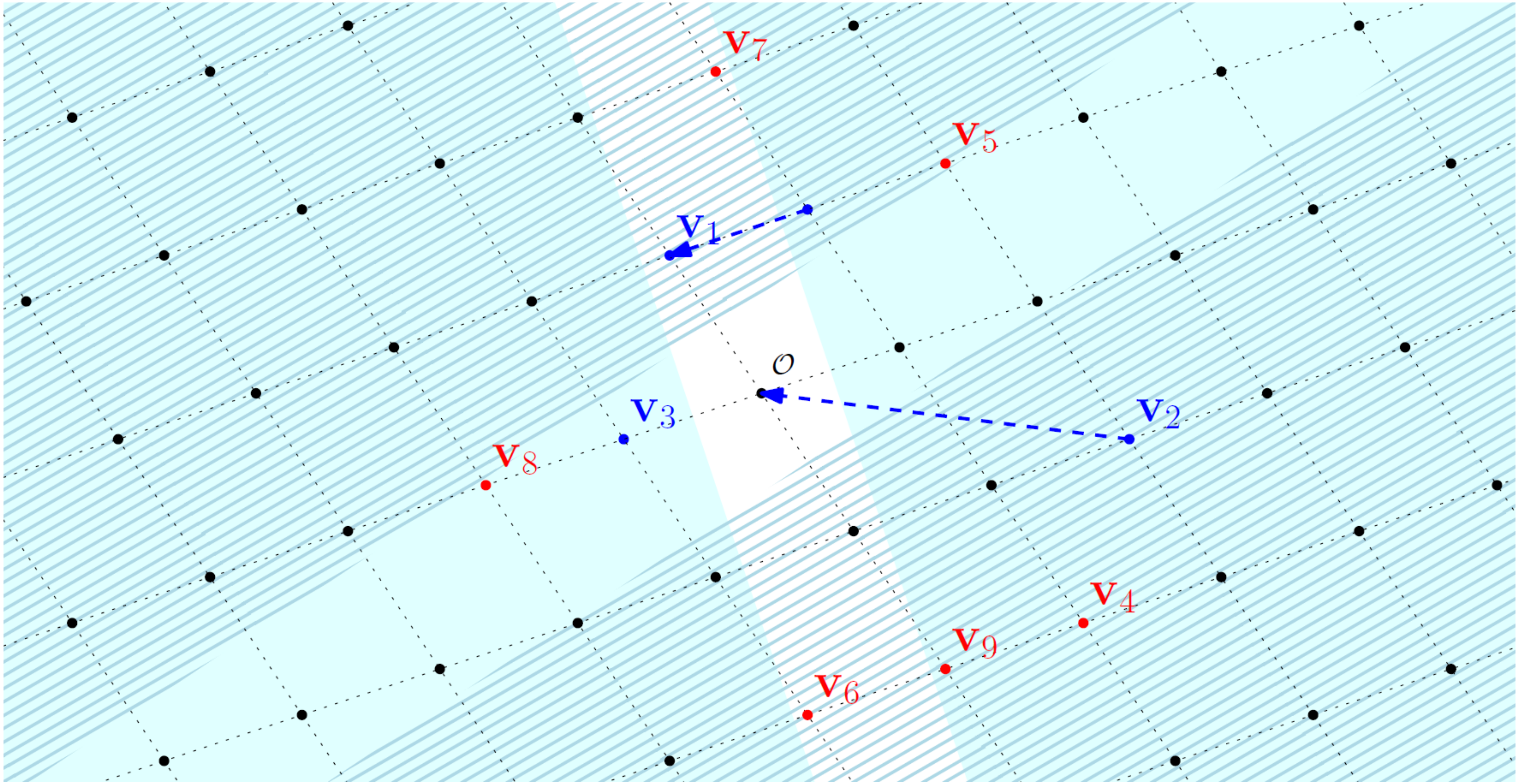
All vectors become pairwise Gauss reduced.



All vectors become pairwise Gauss reduced and the list consists of shorter and shorter vectors.

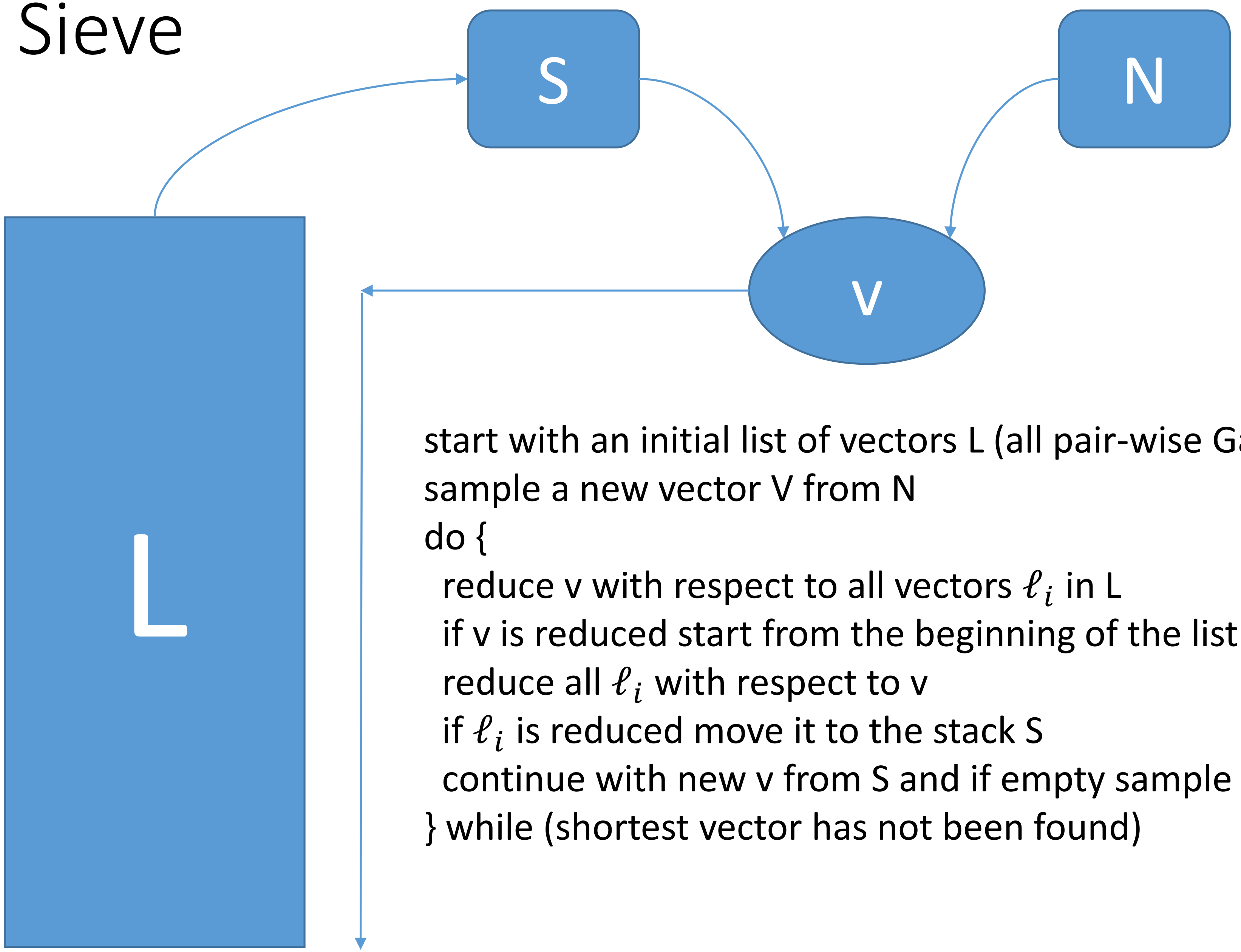


Repeat until we find a short vector or enough collisions.



Repeat until we find a short vector or enough collisions.
Nothing can be proven about the collisions.

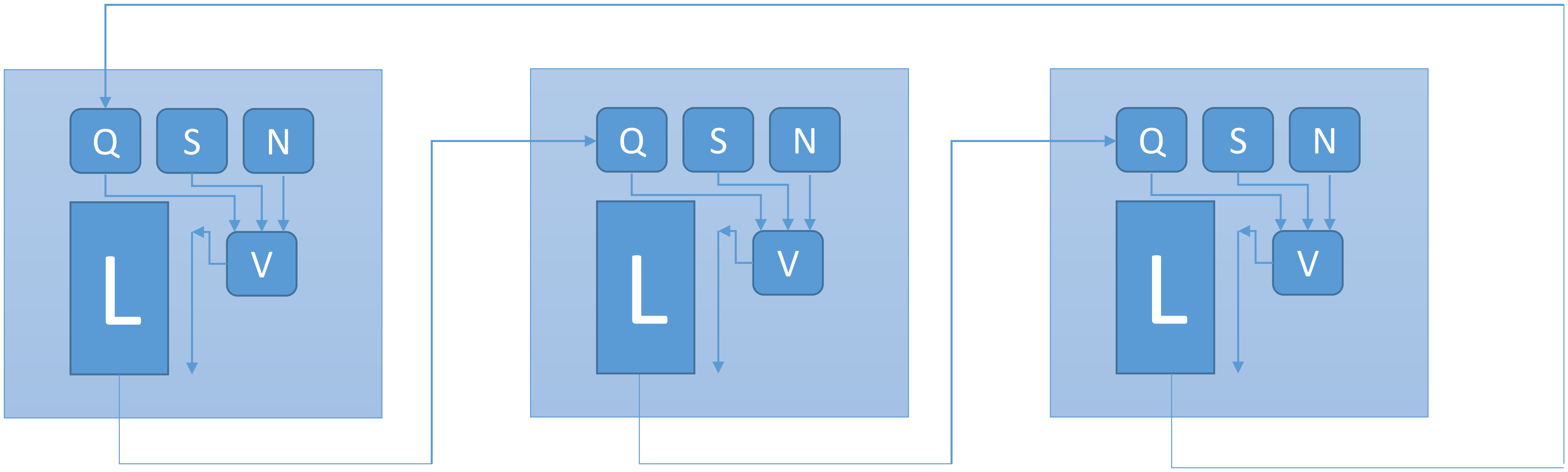
Gauss Sieve



start with an initial list of vectors L (all pair-wise Gauss reduced)
sample a new vector V from N
do {
  reduce v with respect to all vectors ℓ_i in L
  if v is reduced start from the beginning of the list L
  reduce all ℓ_i with respect to v
  if ℓ_i is reduced move it to the stack S
  continue with new v from S and if empty sample a new one from N
} while (shortest vector has not been found)

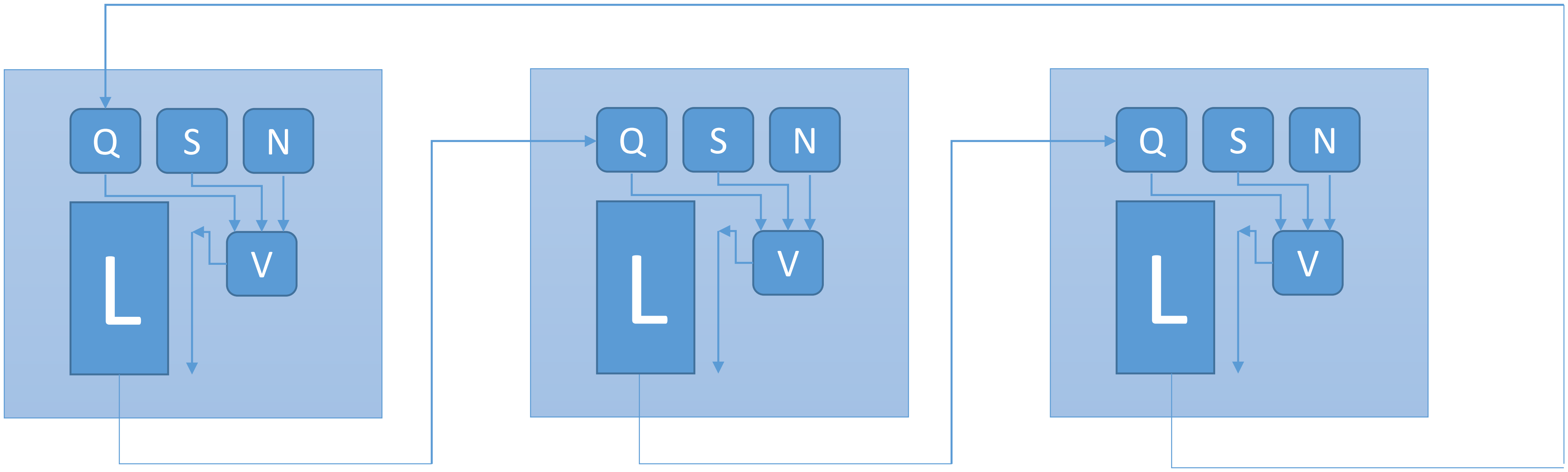
P. Voulgaris and D. Micciancio. Faster exponential time algorithms for the shortest vector problem. Electronic Colloquium on Computational Complexity, 2009

Parallel Gauss Sieve



B. Milde and M. Schneider. A parallel implementation of Gauss Sieve for the shortest vector problem in lattices. In Parallel Computing Technologies, 2011

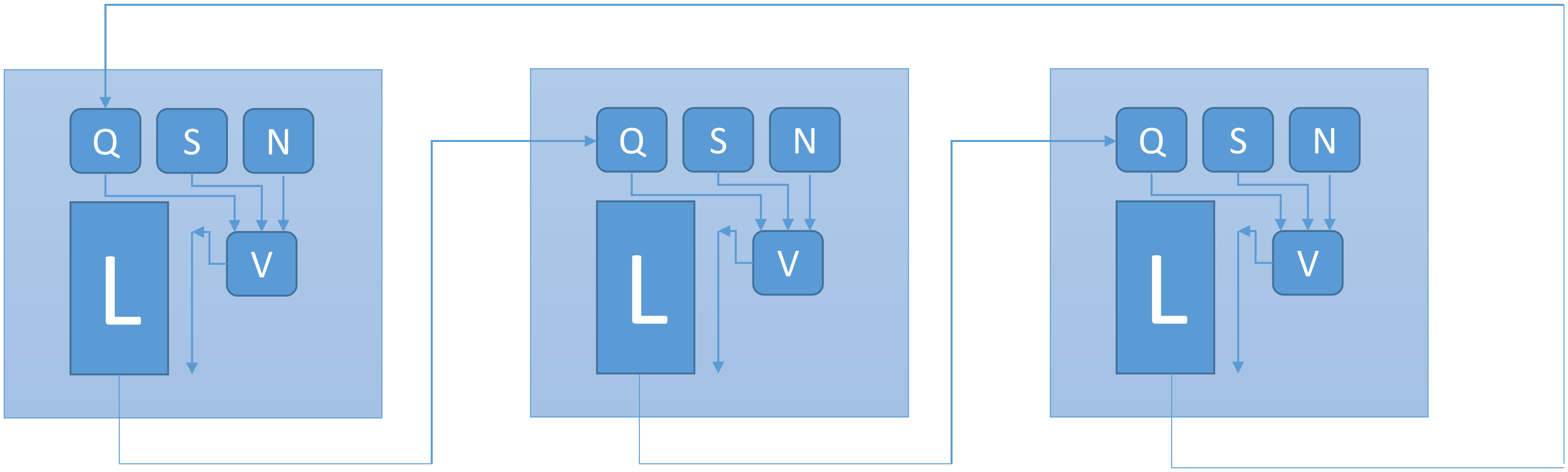
Parallel Gauss Sieve



Pros	Cons
Easy parallel algorithm	
Total list size $(\bigcup_i L_i)$ is distributed among nodes	

B. Milde and M. Schneider. A parallel implementation of Gauss Sieve for the shortest vector problem in lattices. In Parallel Computing Technologies, 2011

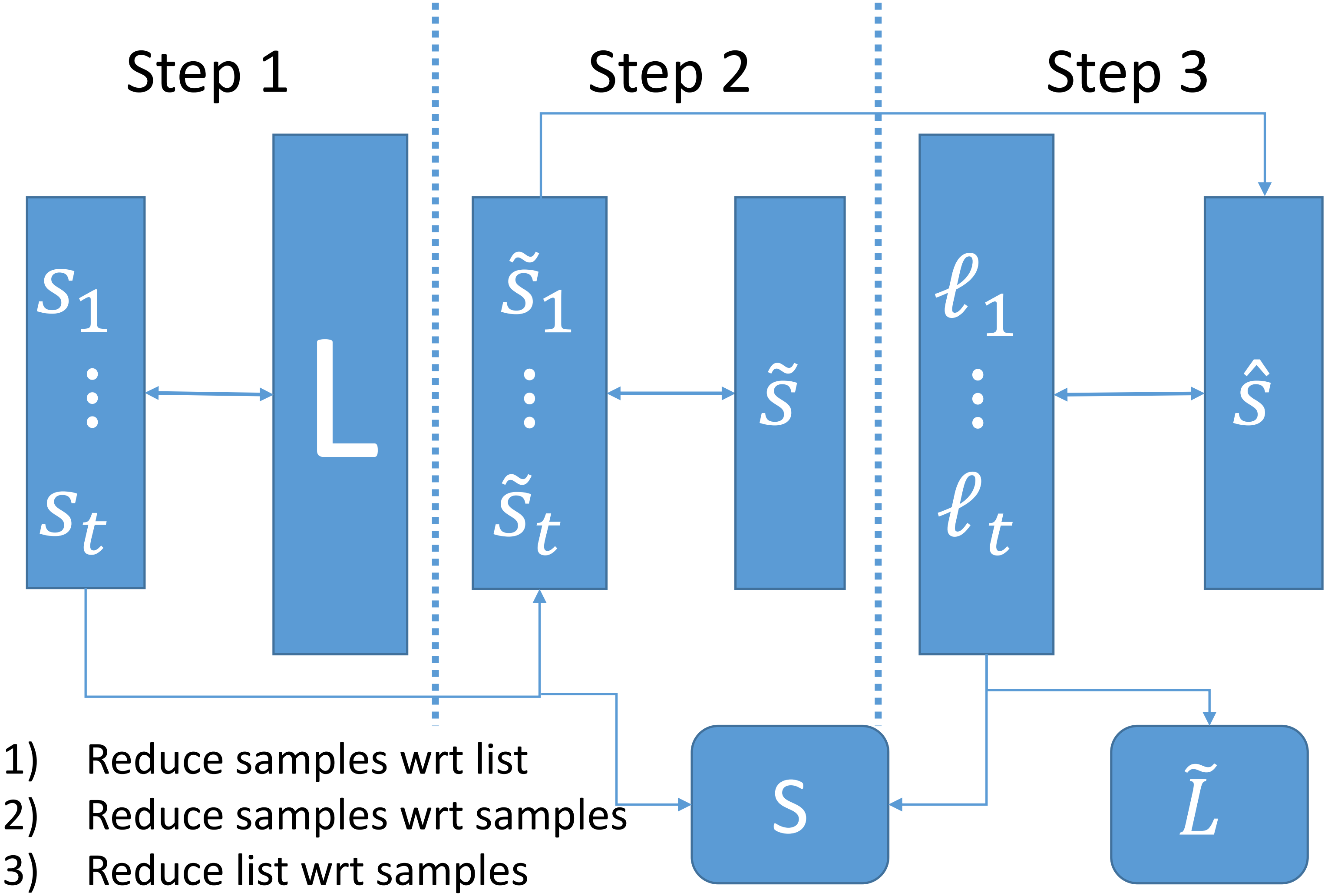
Parallel Gauss Sieve



Pros	Cons
Easy parallel algorithm	$\bigcup_i L_i$ are **not** necessarily pair-wise Gauss reduced
Total list size $(\bigcup_i L_i)$ is distributed among nodes	One node might sample a lot of new vectors: "traffic jams" + idle nodes
	Suggested solution: skip jams → more vectors in $(\bigcup_i L_i)$ are not pair-wise Gauss reduced → increased list size → increased running time

B. Milde and M. Schneider. A parallel implementation of Gauss Sieve for the shortest vector problem in lattices. In Parallel Computing Technologies, 2011

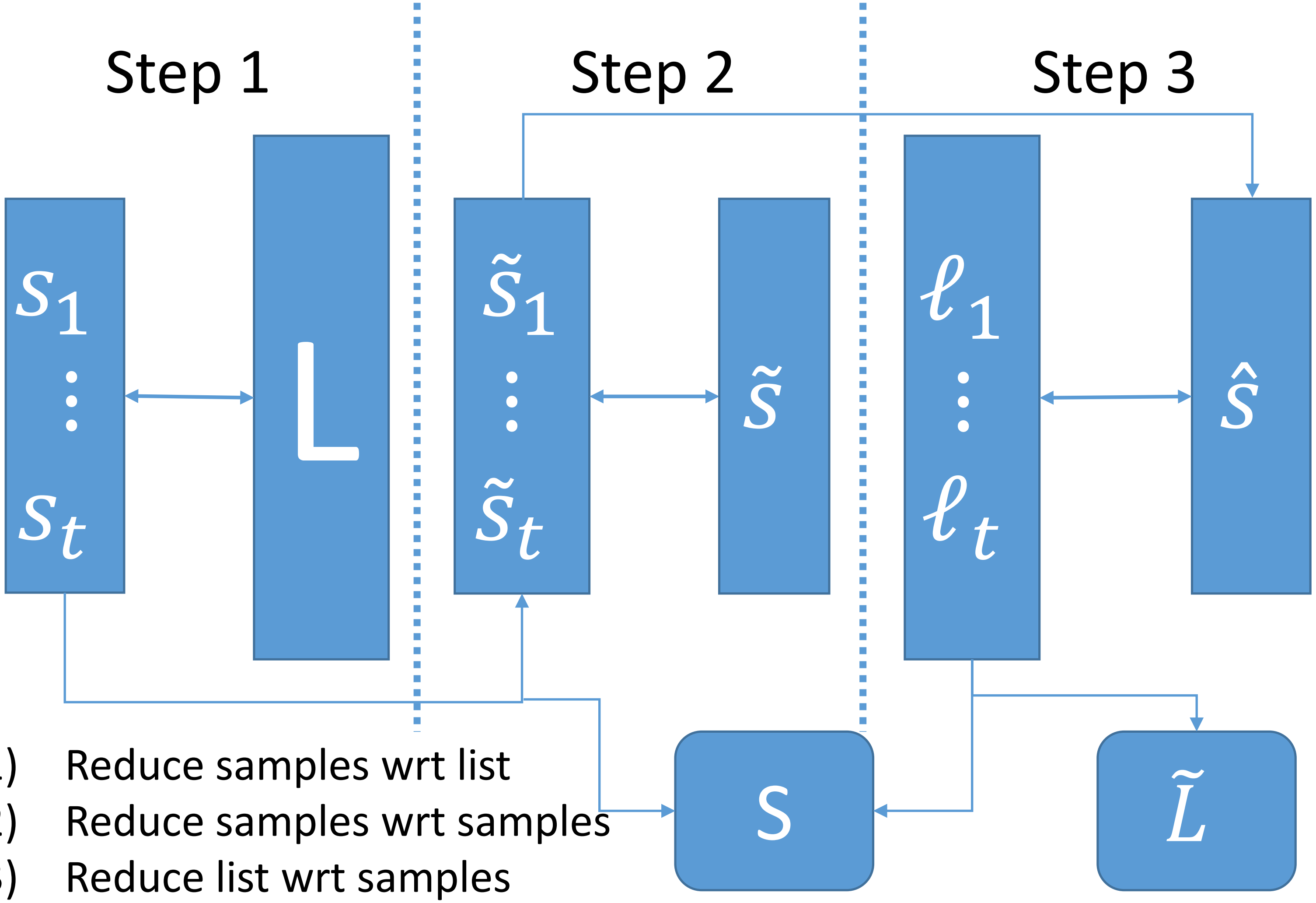
Parallel Gauss Sieve – another approach



1) Reduce samples wrt list
2) Reduce samples wrt samples
3) Reduce list wrt samples
4) Use S as new vectors and $\tilde{L}$ as the new list

T. Ishiguro, S. Kiyomoto, Y. Miyake, and T. Takagi. Parallel Gauss sieve algorithm: Solving the SVP challenge over a 128-dimensional ideal lattice. In PKC, 2014

Parallel Gauss Sieve – another approach



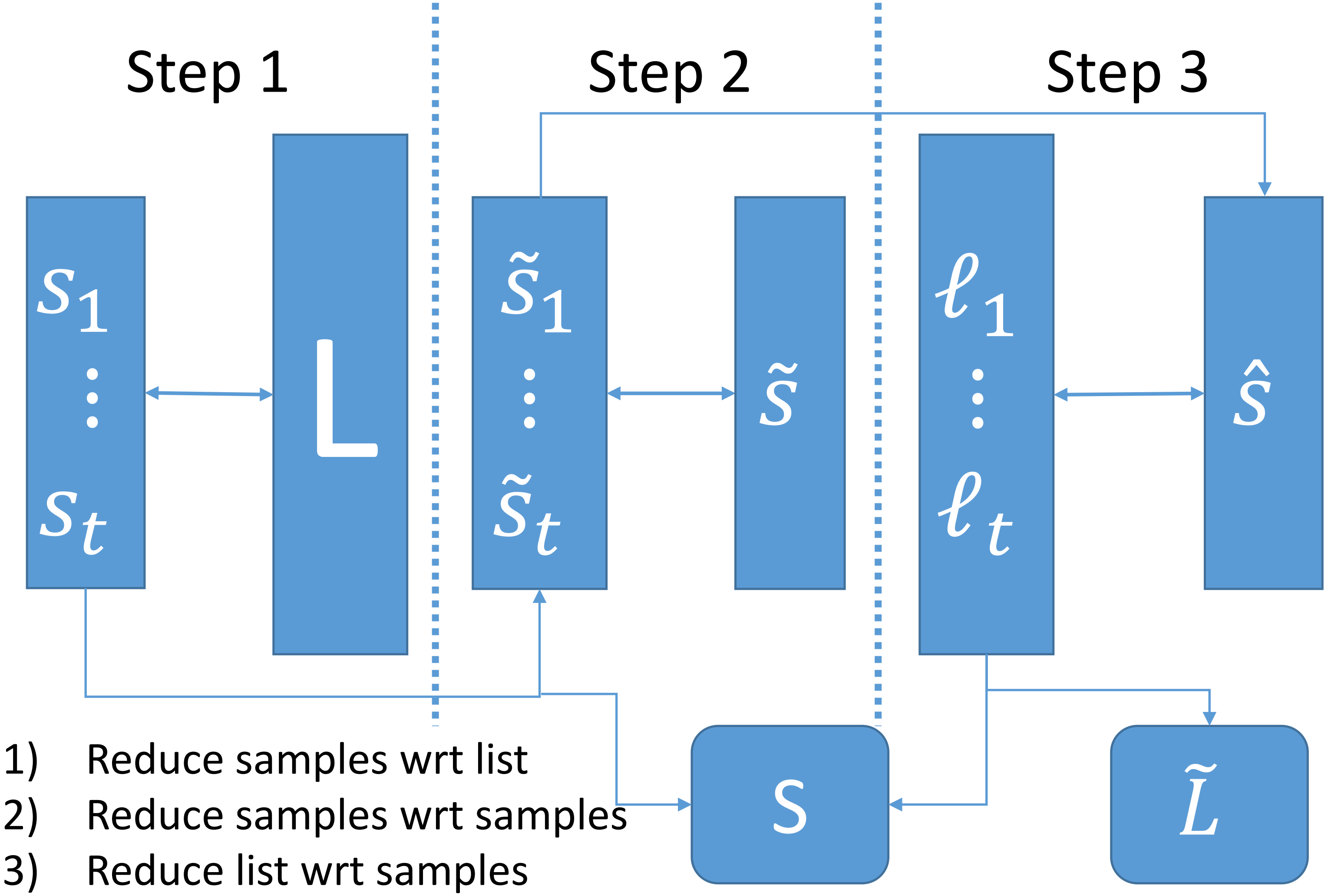
1) Reduce samples wrt list
2) Reduce samples wrt samples
3) Reduce list wrt samples
4) Use S as new vectors and $\tilde{L}$ as the new list

- ✓ After step 3 all vectors in $\tilde{L}$ are pairwise Gauss reduced
- ✓ Avoids the traffic jam problem

- ✓ Every node requires the complete list L and all samples S
- ✓ Conservative estimated max. list size for (non-ideal) dim. 128 is $2^{28} \rightarrow 64$ GB

T. Ishiguro, S. Kiyomoto, Y. Miyake, and T. Takagi. Parallel Gauss sieve algorithm: Solving the SVP challenge over a 128-dimensional ideal lattice. In PKC, 2014

Parallel Gauss Sieve – another approach



1) Reduce samples wrt list
2) Reduce samples wrt samples
3) Reduce list wrt samples
4) Use S as new vectors and $\tilde{L}$ as the new list

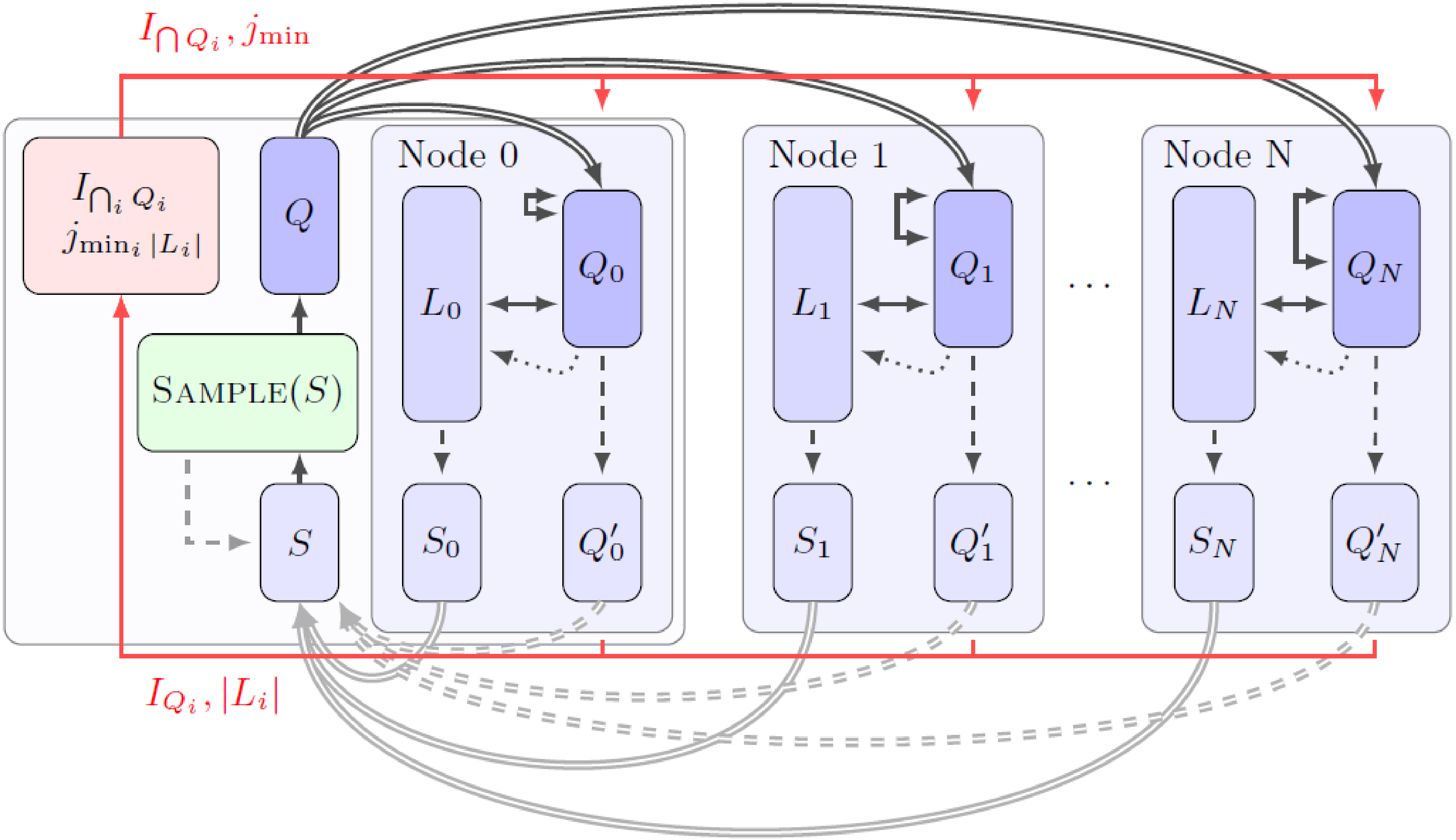
- ✓ After step 3 all vectors in $\tilde{L}$ are pairwise Gauss reduced
- ✓ Avoids the traffic jam problem

- ✓ Every node requires the complete list L and all samples S
- ✓ Conservative estimated max. list size for (non-ideal) dim. 128 is $2^{28} \rightarrow 64$ GB

- ✓ Used to solve ideal lattice challenge of dim. 128 in
  - $\approx$ 15 days on 1344 CPUs
  - $\approx$ 55 CPU years

T. Ishiguro, S. Kiyomoto, Y. Miyake, and T. Takagi. Parallel Gauss sieve algorithm: Solving the SVP challenge over a 128-dimensional ideal lattice. In PKC, 2014

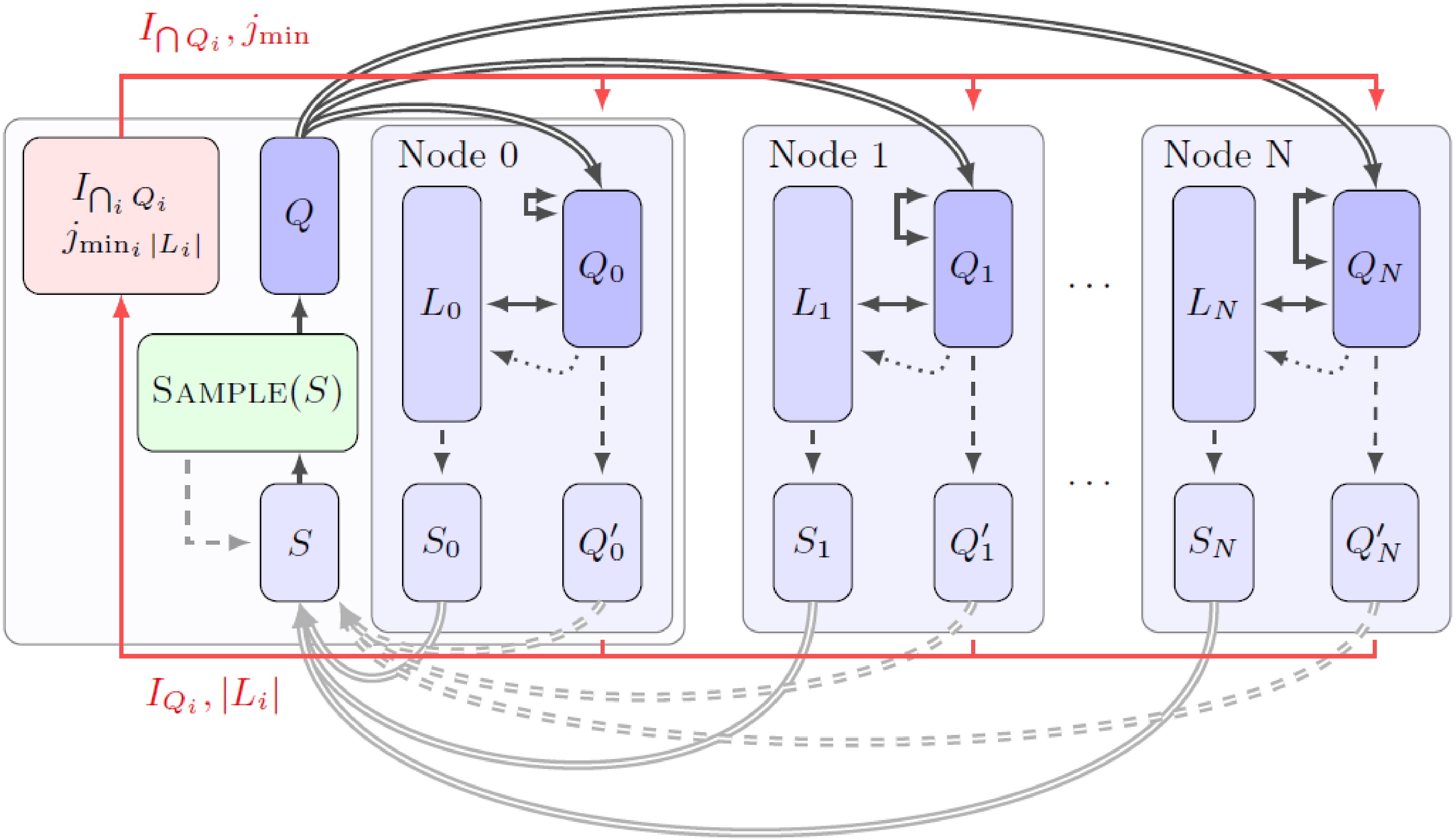
Parallel Gauss Sieve – combining both approaches



1) Collectively obtain new batch Q_i
2) Reduce vectors from Q_i wrt L_i and vice-versa
3) Reduce Q_i wrt to Q_i (divide work)
4) Reduced vectors from L_i go to S_i
5) Reduced vectors from Q_i go to Q'_i

J. W. Bos, M. Naehrig, J. van de Pol: **Sieving for Shortest Vectors in Ideal Lattices: a Practical Perspective,** Cryptology ePrint Archive, Report 2014/880, IACR, 2014.

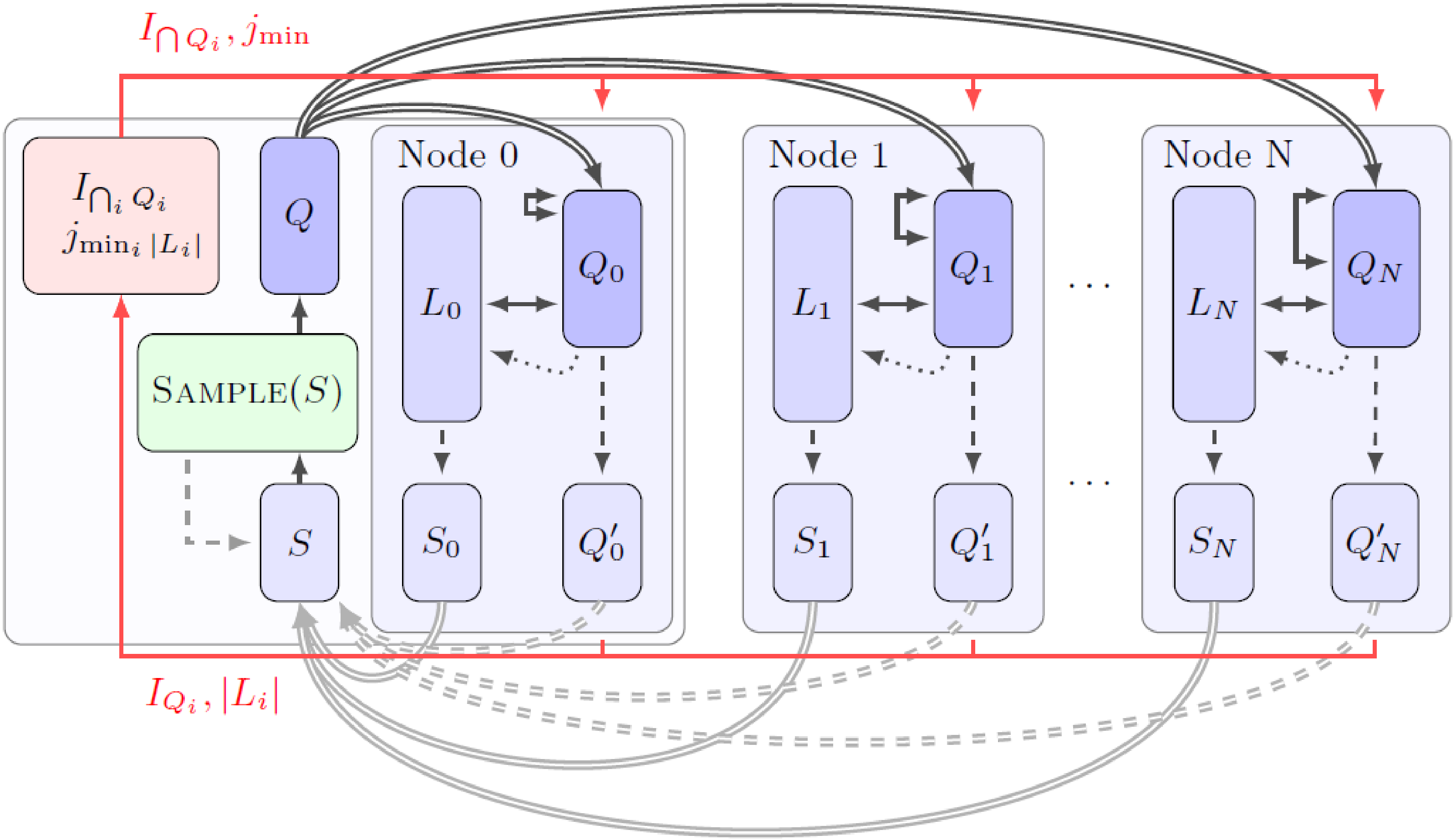
Parallel Gauss Sieve – combining both approaches



- Locally L_i is replaced by $L_i \backslash S_i$
- Compute j s.t. $|L_j|$ is minimal and update L_j as $L_j \cup \bigcap_i Q_i$
- This avoids traffic jams
- List size $(\bigcup_i L_i)$ is distributed among nodes
- All vectors are pairwise Gauss reduced

J. W. Bos, M. Naehrig, J. van de Pol: **Sieving for Shortest Vectors in Ideal Lattices: a Practical Perspective,** Cryptology ePrint Archive, Report 2014/880, IACR, 2014.

Parallel Gauss Sieve – combining both approaches



- The same vector $v \in Q$ might be reduced by different L_i at different nodes → collisions
- Propagate the vector with minimal norm

J. W. Bos, M. Naehrig, J. van de Pol: **Sieving for Shortest Vectors in Ideal Lattices: a Practical Perspective,** Cryptology ePrint Archive, Report 2014/880, IACR, 2014.

Ideal lattice

✓ Ideal lattice: additional structure → also ideals in a ring R

✓Most crypto settings restrict to

$$R = \mathbb{Z}[X]/(\Phi_m(X)),$$

$$\text{where } m = 2n, n = 2^{\ell}, \ell > 0 \text{ s.t. } \Phi_m(X) = X^n + 1$$

- If $a(X)$ belongs to an ideal then $X^i a$ for $i \in \mathbb{Z}$ also belongs to the ideal
- Negative exponents: $X^{-1} = -X^{n-1}$

Notation: An element $a \in R$ is of the form

$$a(X) = \sum_{i=0}^{n-1} a_i X^i$$

and given by the coefficient vector

$$\boldsymbol{a} = (a_0, a_1, \dots, a_{n-1})$$

Ideal lattice

Previous work: store one vector, represent n vectors.

Observation 1: Checking if all n^2 pairs of rotations of a vector $\boldsymbol{a}$ with a vector $\boldsymbol{b}$ are Gauss reduced can be done with only n comparisons and n scalar products.

Lemma 1.
Let $a, b \in R = R = \mathbb{Z}[X]/(X^n + 1)$ for n a power of 2 and $i, j \in \mathbb{Z}$. Then we have:

$$X^i \cdot (X^j \cdot \boldsymbol{a}) = X^{i+j} \cdot \boldsymbol{a}\,, \quad X^i \cdot (\boldsymbol{a} \cdot \boldsymbol{b}) = X^i \cdot \boldsymbol{a} + X^i \cdot \mathbf{b}\,, \quad X^n \cdot \boldsymbol{a} = -\boldsymbol{a},$$

$$\langle X^i \cdot \boldsymbol{a}, X^i \cdot \boldsymbol{b} \rangle = \langle \boldsymbol{a}, \boldsymbol{b} \rangle, \quad \langle X^i \cdot \boldsymbol{a}, X^j \cdot \boldsymbol{b} \rangle = \langle \boldsymbol{a}, -X^{n-i+j} \cdot \boldsymbol{b} \rangle.$$

Lemma 2.
Let $a, b \in R = \mathbb{Z}[X]/(X^n + 1)$ for n a power of 2 and $i, j \in \mathbb{Z}$.

If $2\left|\langle \boldsymbol{a}, X^\ell \cdot \boldsymbol{b} \rangle\right| \leq \min\{\langle \boldsymbol{a}, \boldsymbol{a} \rangle, \langle \boldsymbol{b}, \boldsymbol{b} \rangle\}$ for all $0 \leq \ell < n$, then $X^i \cdot \boldsymbol{a}$ and $X^j \cdot \boldsymbol{b}$ are Gauss reduced for all $i, j \in \mathbb{Z}$.

Ideal lattice

Observation 1. Checking if all n^2 pairs of rotations of a vector $\boldsymbol{a}$ with a vector $\boldsymbol{b}$ are Gauss reduced can be done with only n comparisons and n scalar products.

Observation 2. The n scalar products can be computed using a single ring product.

Define the reflex polynomial $b^{(R)}(X)$ as

$$b^{(R)}(X) = X^{n-1} \cdot b(X^{-1}) \text{ such that } \boldsymbol{b}^{(R)} = (b_{n-1}, b_{n-2}, \dots, b_0)$$

Lemma 3. Let

$$c(X) = a(X) \cdot \left(-X \cdot b^{(R)}(X)\right) \bmod (X^n + 1)$$

And let $c = (c_0, c_1, \dots, c_{n-1}) \in \mathbb{Z}^n$ be its coefficient vector. Then

$$c_i = \left\langle a, X^i \cdot b \right\rangle \text{ for } 0 \leq i < n.$$

Ideal lattice

Observation 1. Checking if all n^2 pairs of rotations of a vector $\boldsymbol{a}$ with a vector $\boldsymbol{b}$ are Gauss reduced can be done with only n comparisons and n scalar products.

Observation 2. The n scalar products can be computed using a single ring product.

Observation 3. Since the ring product is a negacyclic convolution we can use a (symbolic) FFT

Nussbaumer's symbolic FFT

Decompose $\mathbb{Z}[X]/(X^n+1)$ into two extensions. Let $n = 2^k = s \cdot r$ such that $s|r$. Then

$$\mathbb{Z}[X]/(X^n+1) \cong S = T[X]/(X^s - Z), \text{ where } T = \mathbb{Z}[Z]/(Z^r+1)$$

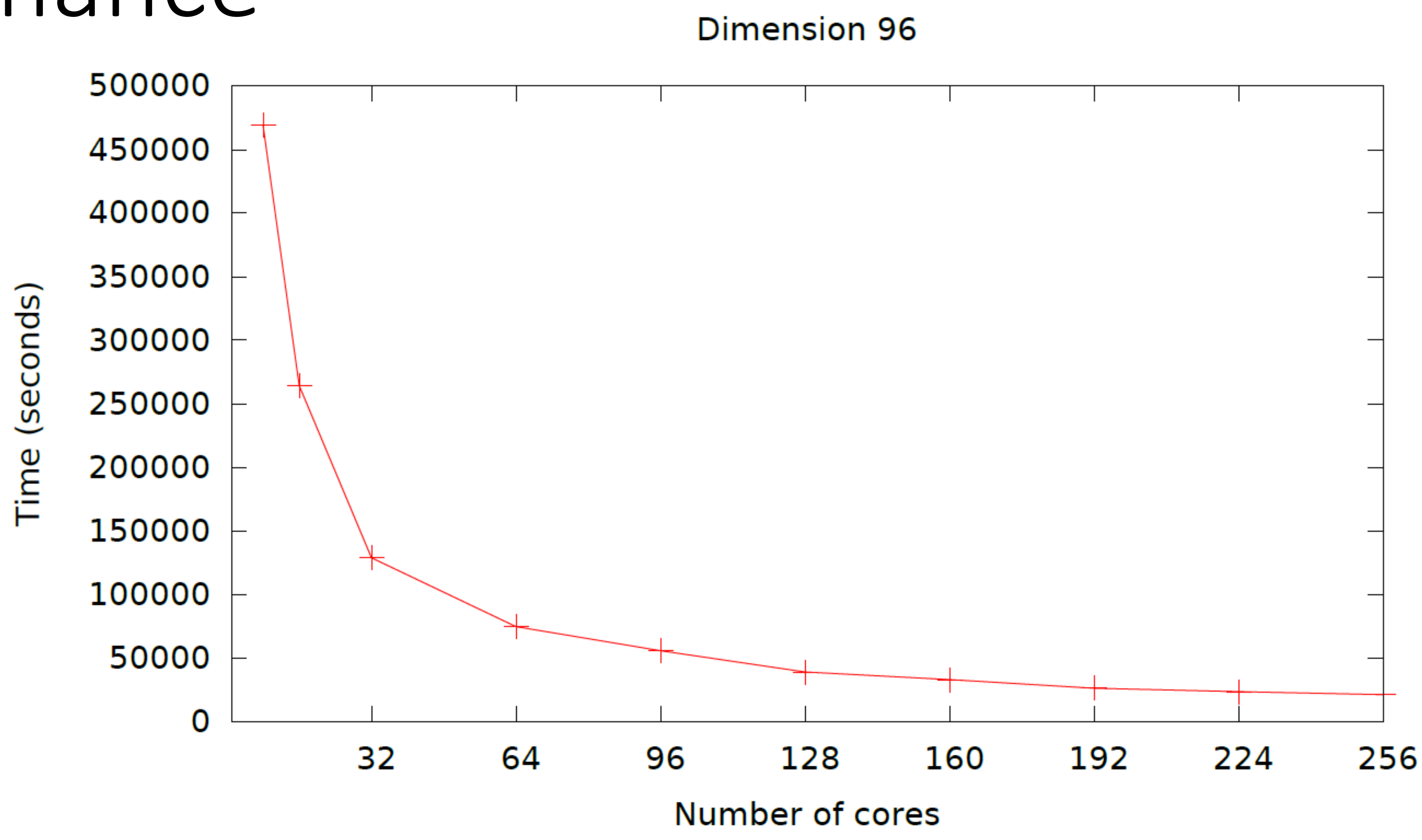
Note: $Z^{r/s}$ is an s^{th} root of -1 in T and $X^s = Z$ in S

Allows to compute the DFT symbolically in T

Use $\mathcal{O}(n \ln n)$ instead of $\mathcal{O}(n^2)$ arithmetic operations

H. J. Nussbaumer. Fast polynomial transform algorithms for digital convolution. Acoustics, Speech and Signal Processing, IEEE Transactions on, 1980

Performance

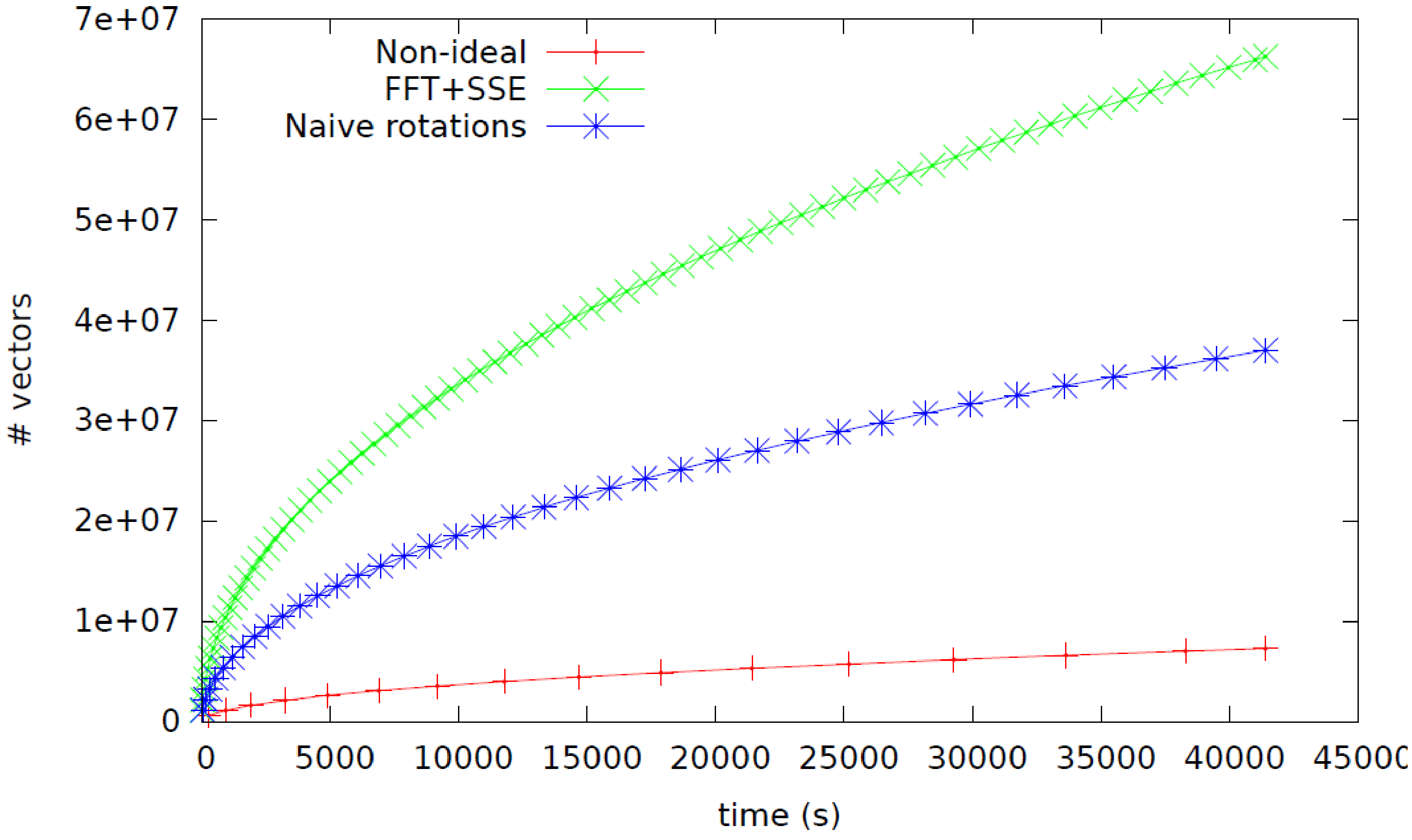


Lattices obtained from the SVP challenge, preprocess with BKZ with blocksize 30.

Speedup	
8 CPU versus 32 CPU	3.6
8 CPU versus 256 CPU	22.1

Experiments run on the BlueCrystal Phase 2 cluster of the Advanced Computing Research Centre at the University of Bristol

Performance



- Ishiguro et al. found a short vector in a dim. 128 ideal lattice in 14.88 days on 1334 CPUs ≈ 55 CPU years
- Our algorithm using FFT on the same lattice challenge on the same hardware (Bristol cluster) on 8.69 days on 1024 CPUs ≈ 25 CPU years
- More than twice as efficient
- Running challenge again with better load balancing, expect better results soon

- Source code available (public domain): http://www.joppebos.com/src/ParallelGaussSieve-1.0.tgz

Conclusions

- Number field sieve (*Integer factorization*)
  - Cofactorization step in parallel
  - When using the NVIDIA GeForce GTX 580 **1.5x improved yield** over quad-core Intel i7-3770K CPU
  - Matrix step is still difficult run in parallel

- Pollard rho (*Elliptic curve discrete logarithm*)
  - Highly-parallel and needs no memory → can utilize the power of low-cost and widely available devices
  - Example: mobile phones

- Gauss sieve (*shortest vector*)
  - Entire algorithm can be run in parallel, how does it scale exactly to thousands of nodes?
  - High communication cost, all nodes need to be online (?)

	Entire algorithm in parallel?	Can run on low-end devices?	Low communication?
Number field sieve	☒	☒	☑ ☒
Pollard rho	☑	☑	☑
Gauss sieve	☑	☒	☒