Space-Efficient and Noise-Robust Quantum Factoring

Seyoon Ragavan and Vinod Vaikuntanathan

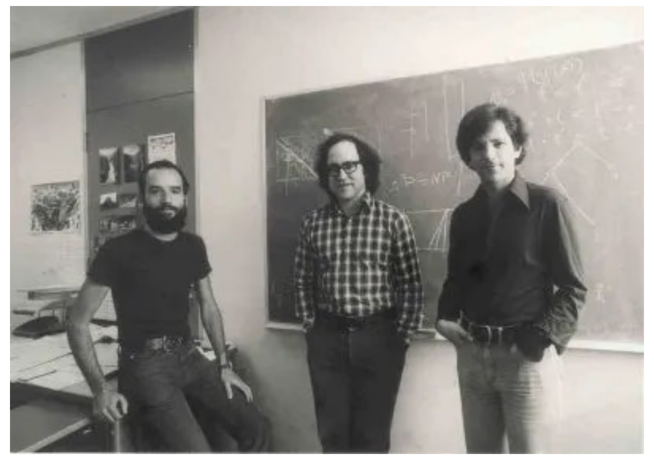
MIT CSAIL

Factoring

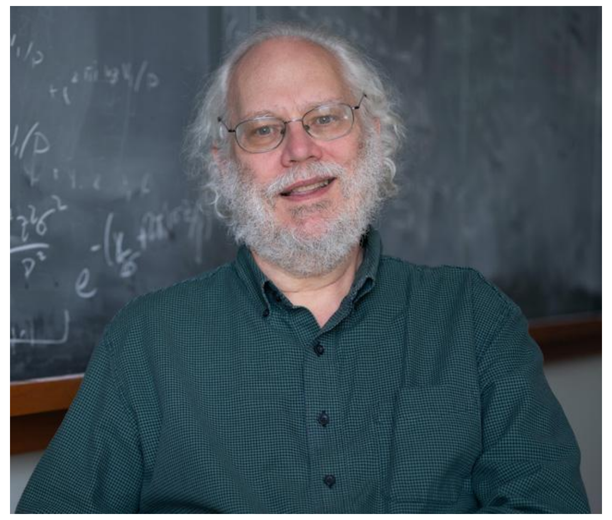
Given an n-bit integer $N < 2^n$, find its prime factorisation in poly(n) time.

(In cryptography, e.g. RSA: $N = pq$ is a product of two equal-sized primes.)

"The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic."



- Hugely important in cryptography
- Important application of a (future) quantum computer



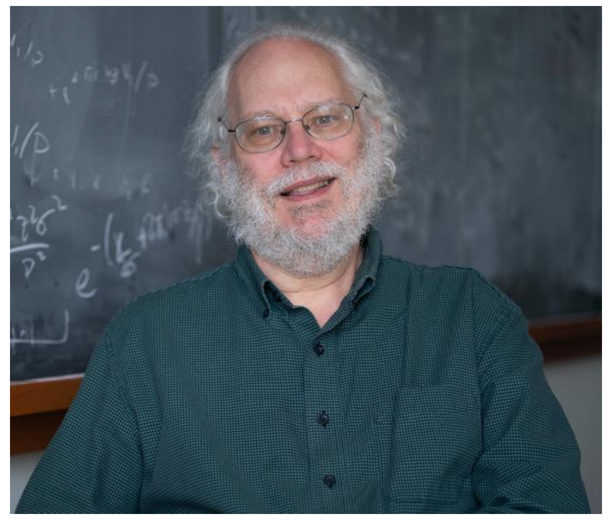
Factoring

Given an n-bit integer $N < 2^n$, find its prime factorisation in $\mathsf{poly}(n)$ time.
(In cryptography, e.g. RSA: $N = pq$ is a product of two equal-sized primes.)

Much work on fast classical algorithms:
cf. survey by Pomerance ("A Tale of Two Sieves")

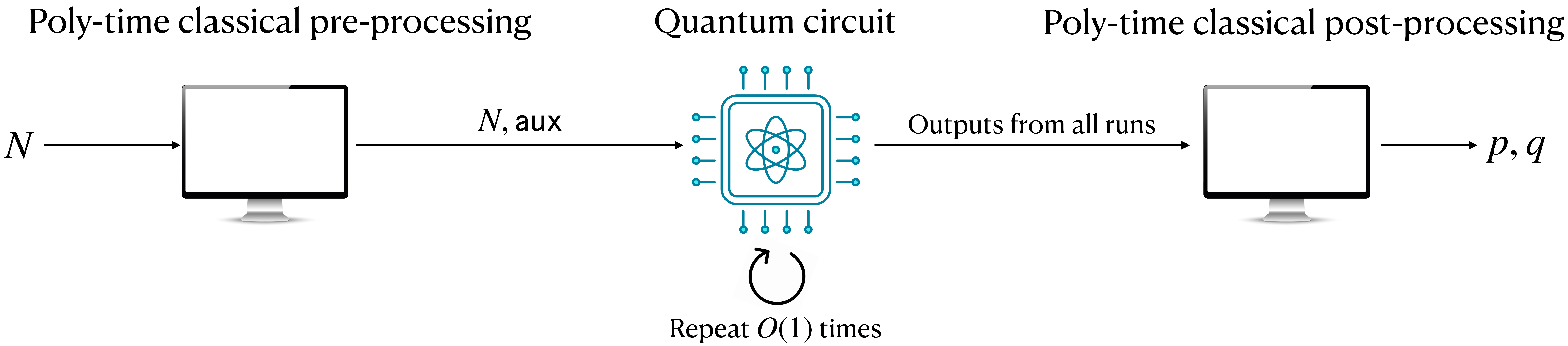
Best known: Number field sieve which runs in $2^{\widetilde{O}\left(\sqrt[3]{n}\right)}$ time

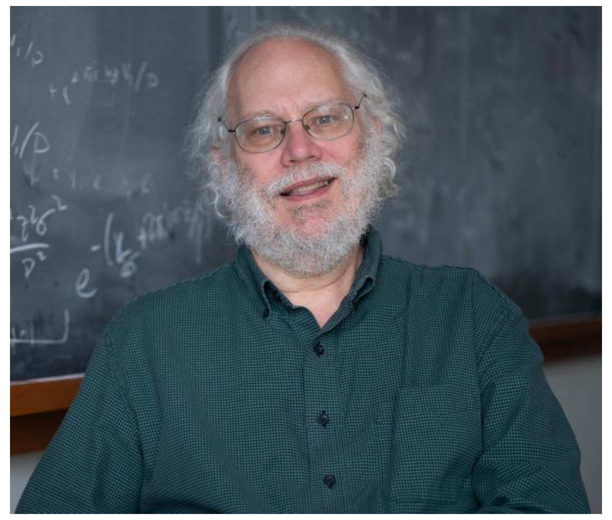
(Pollard 1988; Buhler-Lenstra-Pomerance 1993)



Enter quantum circuits!

Algorithm	Number of gates
Shor 1994	$O(n^2 \log n)$

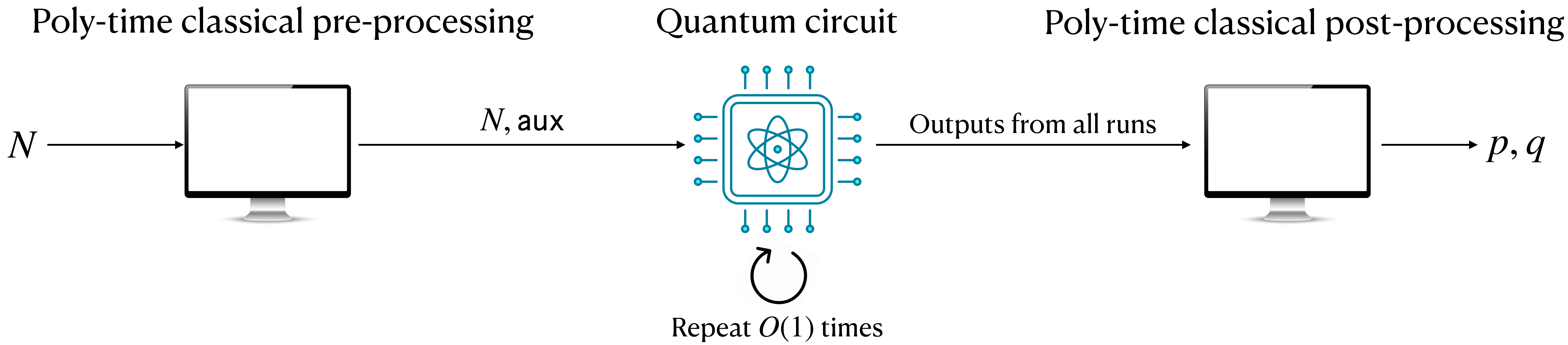


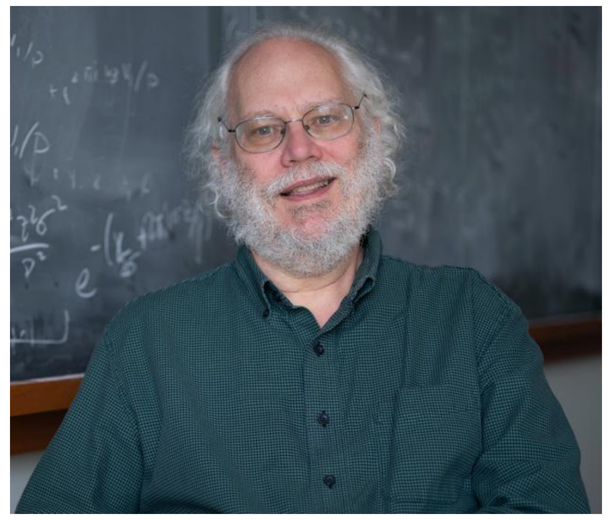


Enter quantum circuits!

Algorithm	Number of gates	Number of qubits
Shor 1994	$O(n^2 \log n)$	$O(n \log n)$

"Idling is just as expensive as doing operations... memory isn't cheap."





Enter quantum circuits!

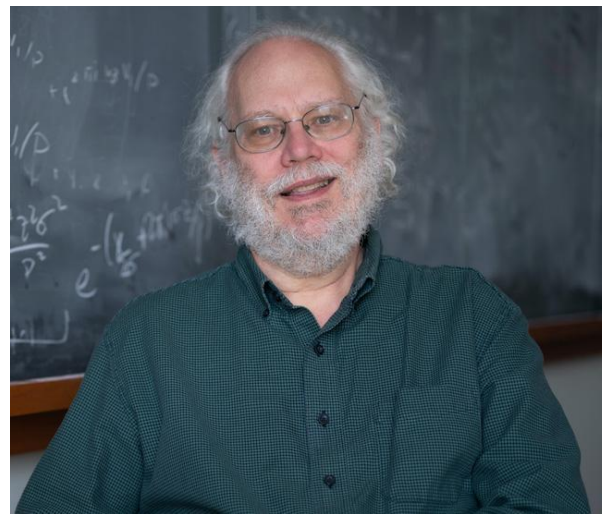
Why improve this?

Algorithm	Number of gates	Number of qubits
Shor 1994	$O(n^2 \log n)$	$O(n \log n)$

How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits

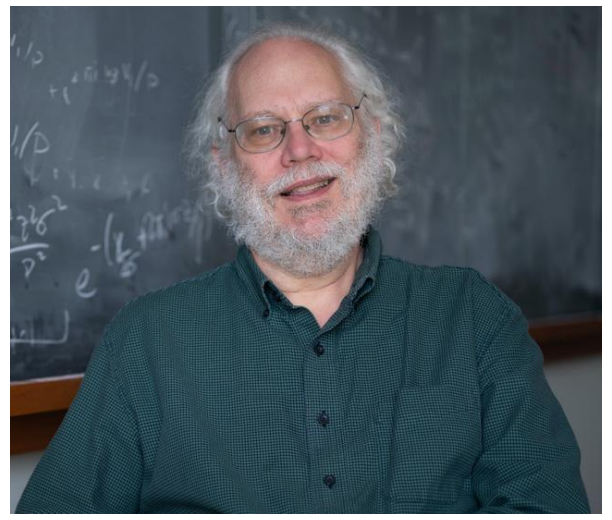
Craig Gidney[1] and Martin Ekerå[2]

[1]Google Inc., Santa Barbara, California 93117, USA
[2]KTH Royal Institute of Technology, SE-100 44 Stockholm, Sweden
Swedish NCSA, Swedish Armed Forces, SE-107 85 Stockholm, Sweden



Enter quantum circuits!

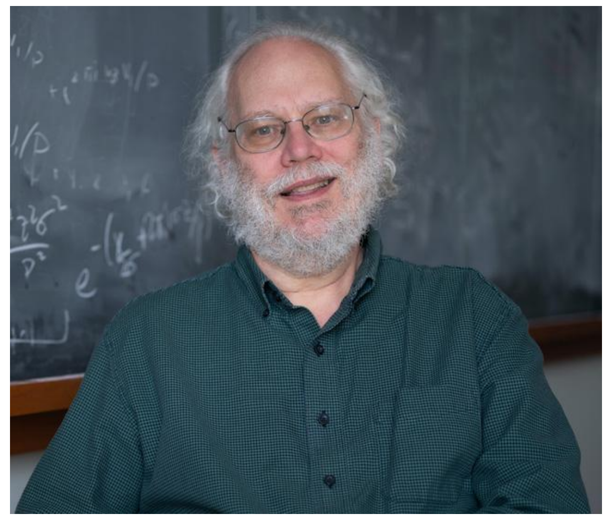
Algorithm	Number of gates	Number of qubits
Shor 1994	$O(n^2 \log n)$	$O(n \log n)$
Regev 2023	$O(n^{3/2} \log n)$	



Enter quantum circuits!

Algorithm	Number of gates	Number of qubits
Shor 1994	$O(n^2 \log n)$	$O(n \log n)$
Regev 2023	$O(n^{3/2} \log n)$	

- *Shor's algorithm also naturally extends to discrete logarithms over any group.*
- *Follow-up work by Ekerå and Gärtner: Regev's speedup can be adapted to discrete logarithms over* $\mathbb{Z}_N^*$



Enter quantum circuits!

Algorithm	Number of gates	Number of qubits
Shor 1994	$O(n^2 \log n)$	$O(n \log n)$
Regev 2023	$O(n^{3/2} \log n)$	$O(n^{3/2})$

Our Result 1: Reducing Qubit Complexity

Algorithm	Number of gates	Number of qubits
Shor 1994	$O(n^2 \log n)$	$O(n \log n)$
Regev 2023	$O(n^{3/2} \log n)$	$O(n^{3/2})$
Our Result: The Best of Both Worlds	$O(n^{3/2} \log n)$	$O(n \log n)$

Our Result 1: Reducing Qubit Complexity

Algorithm	Number of gates	Number of qubits
Shor 1994	$O(n^2 \log n)$	$O(n \log n)$
Regev 2023	$O(n^{3/2} \log n)$	$O(n^{3/2})$
Our Result: The Best of Both Worlds	$O(n^{3/2} \log n)$	$O(n \log n)$

Concretely for $n = 2048$, using schoolbook multiplication:

- Regev: $\approx 3n^{3/2} \approx 278000$ qubits
- Our result: $\approx 10.4n \approx 21300$ qubits

Our Result 2: Better Noise-Tolerance

Algorithm	Number of gates	Number of qubits	Number of runs
Shor 1994	$O(n^2 \log n)$	$O(n \log n)$	$O(1)$ clean runs
Regev 2023	$O(n^{3/2} \log n)$	$O(n^{3/2})$	$O(n^{1/2})$ clean runs
Our result	$O(n^{3/2} \log n)$	$O(n \log n)$	

- Shor: needs to run $O(n^2 \log n \cdot 1) = \widetilde{O}(n^2)$ gates without logical error → per-gate error prob. $\widetilde{O}(n^{-2})$

Our Result 2: Better Noise-Tolerance

Algorithm	Number of gates	Number of qubits	Number of runs
Shor 1994	$O(n^2 \log n)$	$O(n \log n)$	$O(1)$ clean runs
Regev 2023	$O(n^{3/2} \log n)$	$O(n^{3/2})$	$O(n^{1/2})$ clean runs
Our result	$O(n^{3/2} \log n)$	$O(n \log n)$	

- Shor: needs to run $O(n^2 \log n \cdot 1) = \widetilde{O}(n^2)$ gates without logical error → per-gate error prob. $\widetilde{O}(n^{-2})$
- Regev's classical post-processing: all $O(n^{1/2})$ runs need to be free of logical errors

Our Result 2: Better Noise-Tolerance

Algorithm	**Number of gates**	**Number of qubits**	**Number of runs**
Shor 1994	$O(n^2 \log n)$	$O(n \log n)$	$O(1)$ clean runs
Regev 2023	$O(n^{3/2} \log n)$	$O(n^{3/2})$	$O(n^{1/2})$ clean runs
Our result	$O(n^{3/2} \log n)$	$O(n \log n)$	

- Shor: needs to run $O(n^2 \log n \cdot 1) = \widetilde{O}(n^2)$ gates without logical error → per-gate error prob. $\widetilde{O}(n^{-2})$
- Regev's classical post-processing: all $O(n^{1/2})$ runs need to be free of logical errors
  - Total # of gates is $O(n^{3/2} \log n \cdot n^{1/2}) = \widetilde{O}(n^2)$ → ***per-gate error prob. no better than Shor!***

Our Result 2: Better Noise-Tolerance

Algorithm	Number of gates	Number of qubits	Number of runs
Shor 1994	$O(n^2 \log n)$	$O(n \log n)$	$O(1)$ clean runs
Regev 2023	$O(n^{3/2} \log n)$	$O(n^{3/2})$	$O(n^{1/2})$ clean runs
Our result	$O(n^{3/2} \log n)$	$O(n \log n)$	$O(n^{1/2})$ dirty runs

- Shor: needs to run $O(n^2 \log n \cdot 1) = \widetilde{O}(n^2)$ gates without logical error → per-gate error prob. $\widetilde{O}(n^{-2})$
- Regev's classical post-processing: all $O(n^{1/2})$ runs need to be free of logical errors
  - Total # of gates is $O(n^{3/2} \log n \cdot n^{1/2}) = \widetilde{O}(n^2)$ → ***per-gate error prob. no better than Shor!***
- **Our result: Improved post-processing to tolerate a constant failure probability per run**

Our Result 2: Better Noise-Tolerance

Algorithm	Number of gates	Number of qubits	Number of runs
Shor 1994	$O(n^2 \log n)$	$O(n \log n)$	$O(1)$ clean runs
Regev 2023	$O(n^{3/2} \log n)$	$O(n^{3/2})$	$O(n^{1/2})$ clean runs
Our result	$O(n^{3/2} \log n)$	$O(n \log n)$	$O(n^{1/2})$ dirty runs

- Shor: needs to run $O(n^2 \log n \cdot 1) = \widetilde{O}(n^2)$ gates without logical error $\rightarrow$ per-gate error prob. $\widetilde{O}(n^{-2})$
- Regev's classical post-processing: all $O(n^{1/2})$ runs need to be free of logical errors
  - Total # of gates is $O(n^{3/2} \log n \cdot n^{1/2}) = \widetilde{O}(n^2) \rightarrow$ ***per-gate error prob. no better than Shor!***
- **Our result: Improved post-processing to tolerate a constant failure probability per run**
  - Per-gate error probability only needs to be $\widetilde{O}(n^{-3/2})$, *better than both Shor and Regev*!

Our Result 2: Better Noise-Tolerance

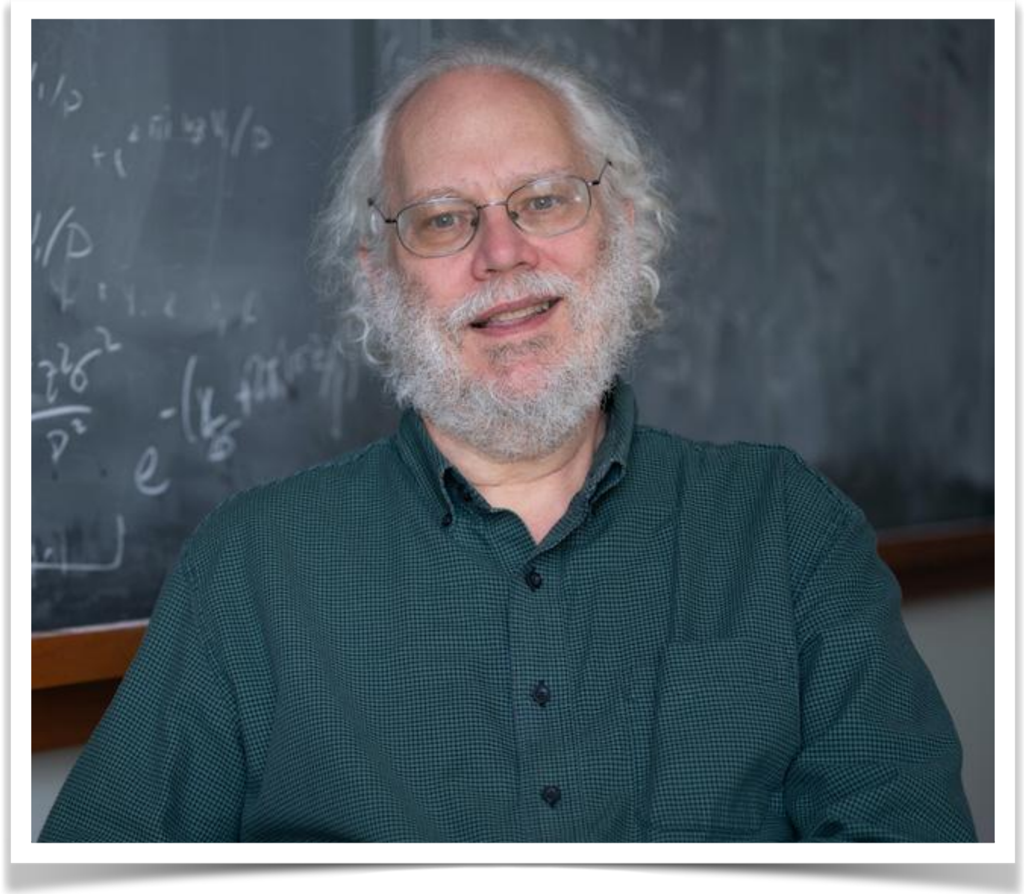
Algorithm	**Number of gates**	**Number of qubits**	**Number of runs**
Shor 1994	$O(n^2 \log n)$	$O(n \log n)$	$O(1)$ clean runs
Regev 2023	$O(n^{3/2} \log n)$	$O(n^{3/2})$	$O(n^{1/2})$ clean runs
Our result	$O(n^{3/2} \log n)$	$O(n \log n)$	$O(n^{1/2})$ dirty runs

- Shor: needs to run $O(n^2 \log n \cdot 1) = \widetilde{O}(n^2)$ gates without logical error → per-gate error prob. $\widetilde{O}(n^{-2})$
- Regev's classical post-processing: all $O(n^{1/2})$ runs need to be free of logical errors
  - Total # of gates is $O(n^{3/2} \log n \cdot n^{1/2}) = \widetilde{O}(n^2)$ → ***per-gate error prob. no better than Shor!***
- **Our result: Improved post-processing to tolerate a constant failure probability per run**
- Similar result in a concurrent work by Ekerå and Gärtner.

Our Results 1 and 2: Summary

Algorithm	Number of gates	Number of qubits	Number of runs
Shor 1994	$O(n^2 \log n)$	$O(n \log n)$	$O(1)$ clean runs
Regev 2023	$O(n^{3/2} \log n)$	$O(n^{3/2})$	$O(n^{1/2})$ clean runs
Our result	$O(n^{3/2} \log n)$	$O(n \log n)$	$O(n^{1/2})$ dirty runs

Shor and Regev: A Sketch



Shor overview: finding square roots of 1

- Goal: find $z \not\equiv \pm 1 \mod N$ such that $z^2 \equiv 1 \mod N$
  - N divides $z^2 - 1 = (z-1)(z+1)$ but not either factor individually
  - Hence $\gcd(z-1, N)$ is a nontrivial divisor of N
- Most factoring algorithms - **quantum or classical** - boil down to this

Shor overview: reduction to period-finding

Let's focus on $N = 3763$**.** *(* $= 53 \times 71$*, but we don't know this yet)*

Shor overview: reduction to period-finding

Let's focus on $N = 3763$**.** *(* $= 53 \times 71$*, but we don't know this yet)*

- Choose a random square as a base e.g. 4
- Powers of 4 mod 3763: $4^0 = 1, 4^1 = 4, 4^2 = 16, 64, 256, 1024, 333, 1332, \ldots$
  - Eventually repeats
  - Let r be the first positive index where $4^r \equiv 1 \bmod 3763$

Shor overview: reduction to period-finding

Let's focus on $N = 3763$. *($= 53 \times 71$, but we don't know this yet)*

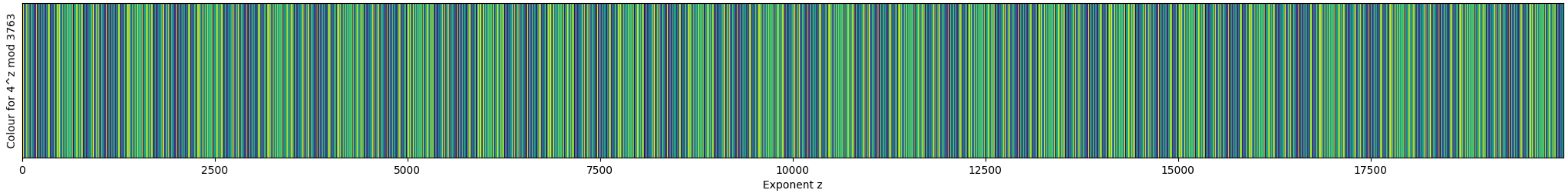
- Choose a random square as a base e.g. 4
- Powers of 4 mod 3763: $4^0 = 1, 4^1 = 4, 4^2 = 16, 64, 256, 1024, 333, 1332, \ldots$
  - Eventually repeats
  - Let r be the first positive index where $4^r \equiv 1 \bmod 3763$
- But $4^r = (2^r)^2$, so 2^r is a square root of $1 \bmod 3763$
  - With some luck: $2^r \not\equiv \pm 1 \bmod 3763$ so this would give us a factor!

Shor overview: reduction to period-finding

Let's focus on $N = 3763$**.** *(* $= 53 \times 71$*, but we don't know this yet)*

- Choose a random square as a base e.g. 4
- Powers of 4 mod 3763: $4^0 = 1, 4^1 = 4, 4^2 = 16, 64, 256, 1024, 333, 1332, \ldots$
  - Eventually repeats
  - Let r be the first positive index where $4^r \equiv 1 \bmod 3763$
- But $4^r = (2^r)^2$, so 2^r is a square root of $1 \bmod 3763$
  - With some luck: $2^r \not\equiv \pm 1 \bmod 3763$ so this would give us a factor!
  - *"Luck" is with respect to the randomly chosen base*

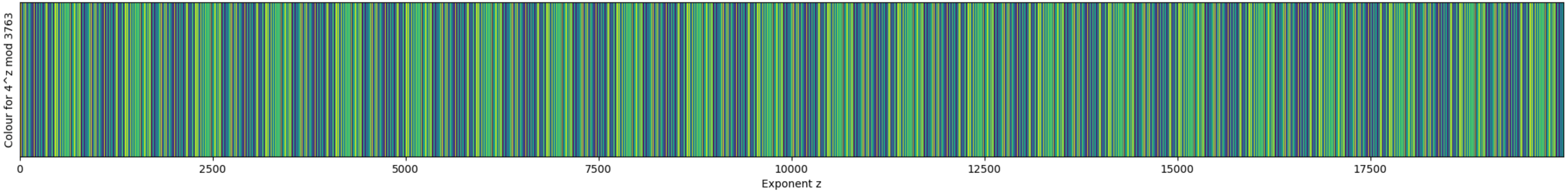
Shor overview: quantum period finding



We can find this period quickly with a quantum computer!

1. Superposition over many values of z
   a. z may be as large as N but we only need $\log N \leq n$ qubits
   b. $\log z$ **turns out to be the crucial metric for circuit size too**

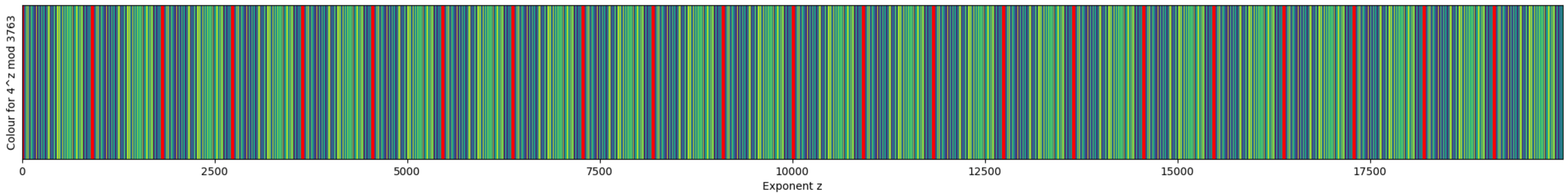
Shor overview: quantum period finding



We can find this period quickly with a quantum computer!

1. Superposition over many values of z
2. Compute 4^z mod 3763 in superposition
    a. This will basically give us the above picture

Shor overview: quantum period finding



We can find this period quickly with a quantum computer!

1. Superposition over many values of z
2. Compute 4^z mod 3763 in superposition
3. This signal has frequency $1/r$ (r is the period), which is exactly what we want to recover!
    a. Apply QFT to the z register and measure to find a noisy multiple of $1/r$
    b. Classical post-processing → recover r

Regev's key idea: add dimensions

Before: exploited periodicity of

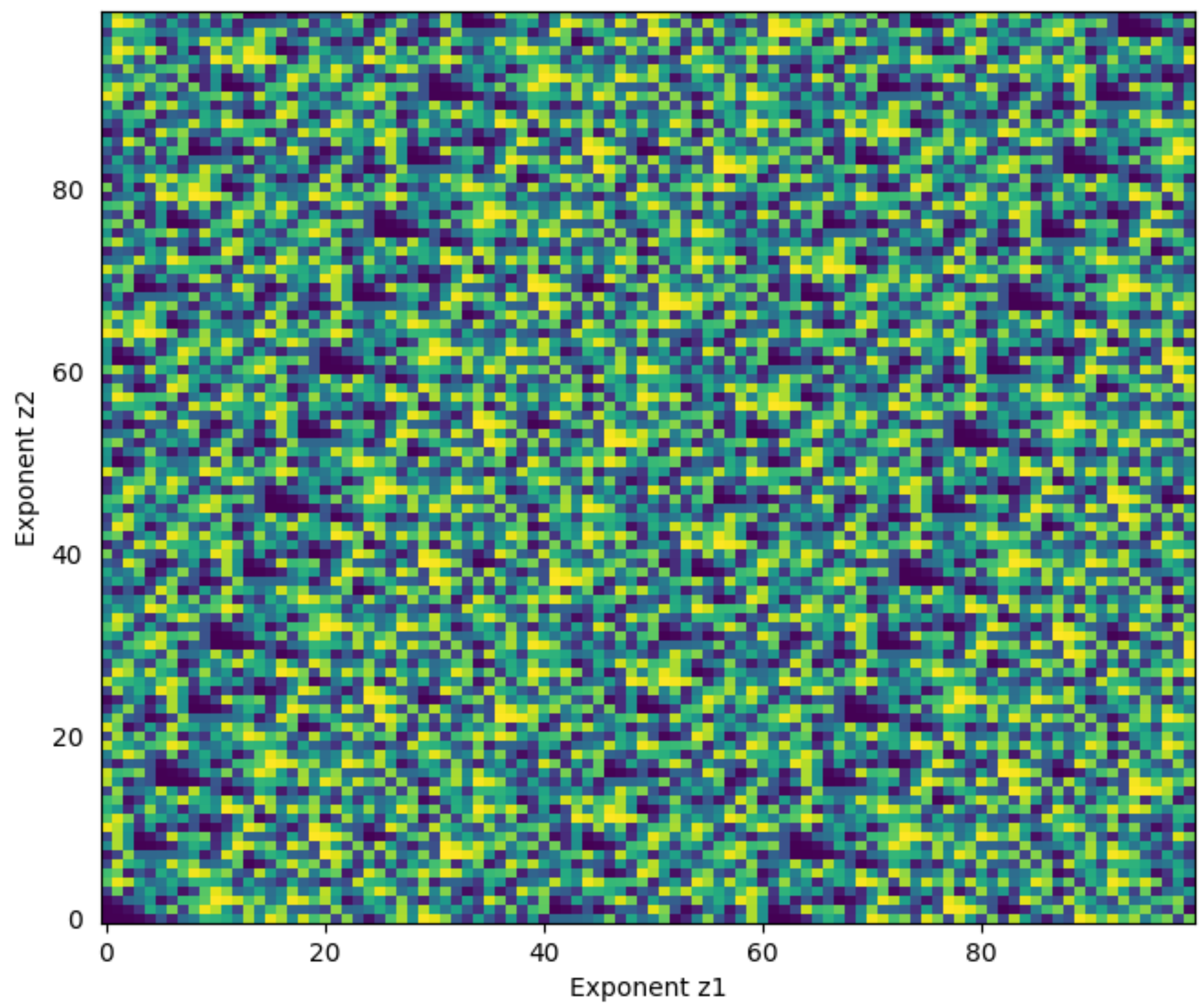
$$z \mapsto 4^z \bmod 3763.$$

Now: let's look at

$$(z_1, z_2) \mapsto 4^{z_1} 9^{z_2} \bmod 3763.$$

Regev's key idea: add dimensions

Q: How does this help us?

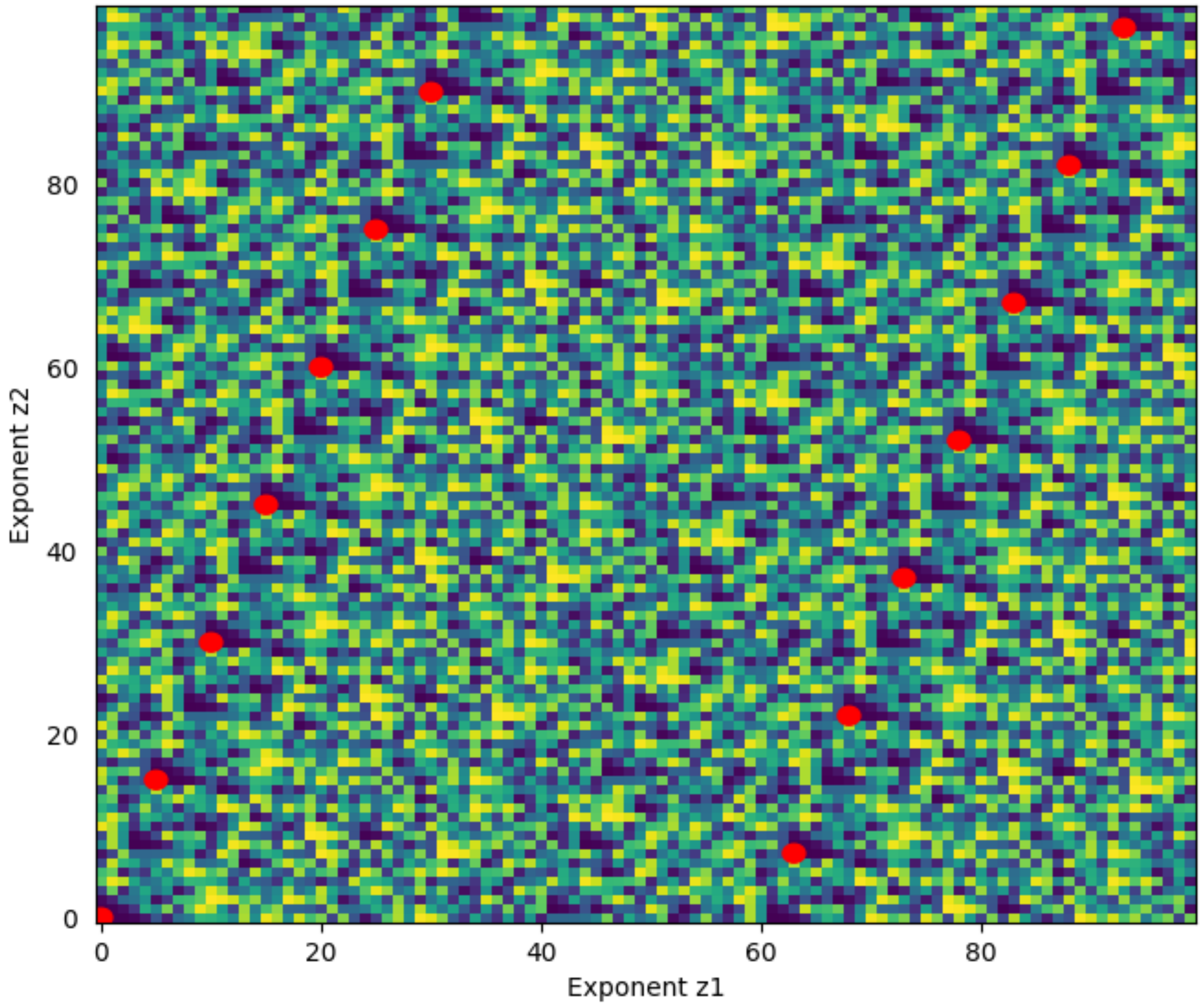


Regev's key idea: add dimensions

Q: How does this help us?

A: There are many "2D periods" now.

Crucially, these periods are much closer to (0, 0) than in Shor!



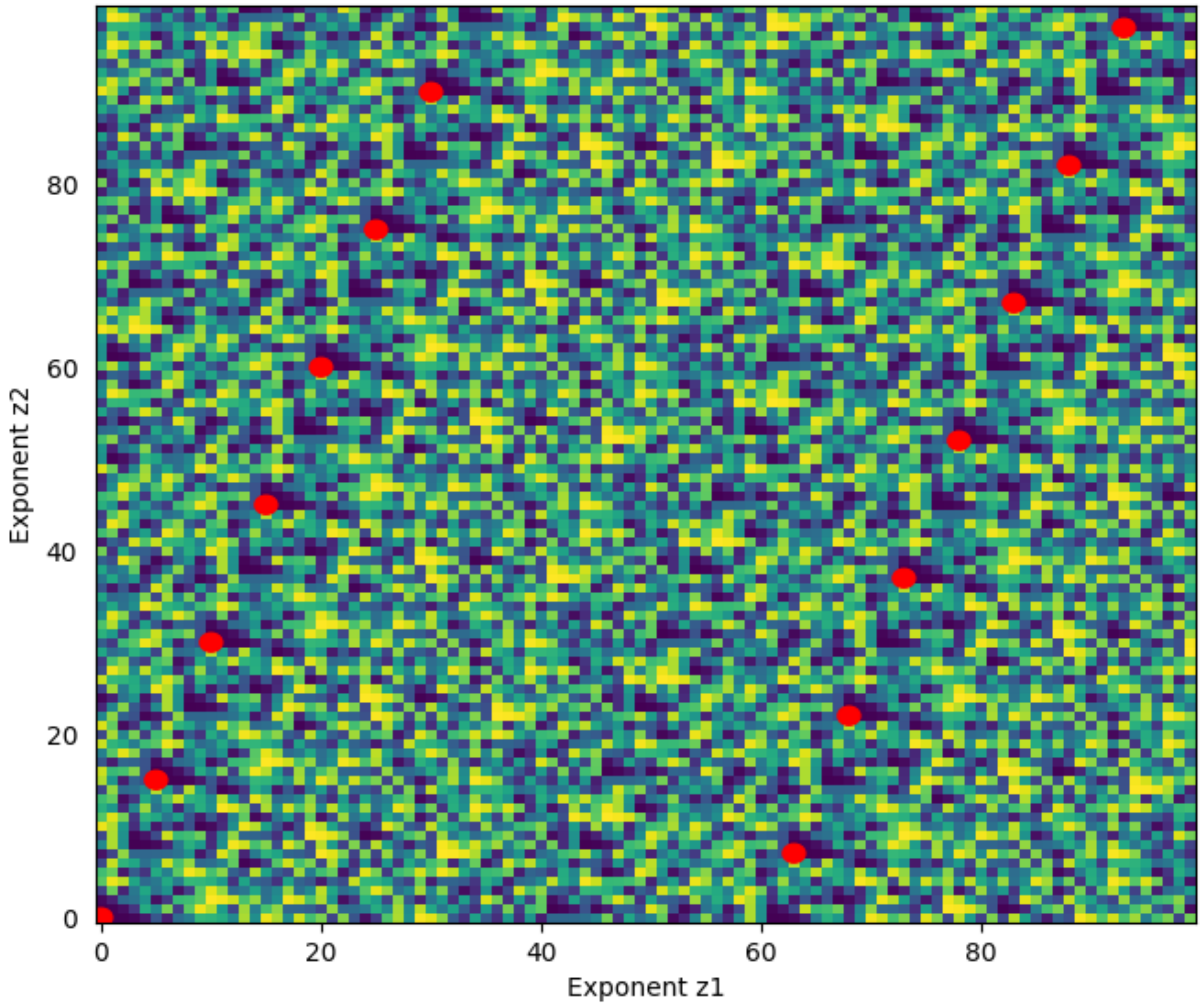
Regev's key idea: add dimensions

Q: How does this help us?

A: There are many "2D periods" now.

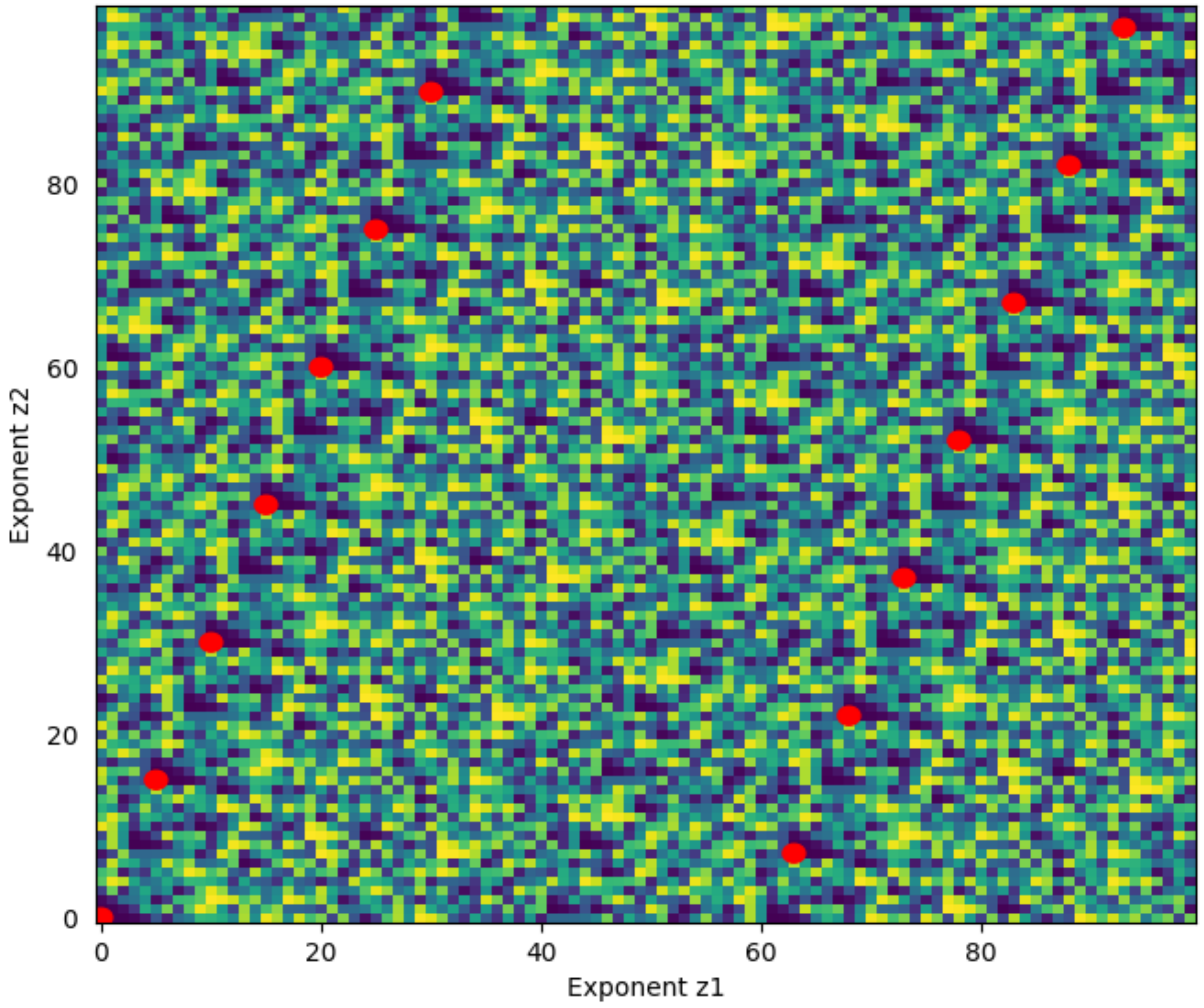
Crucially, these periods are much closer to (0, 0) **than in Shor!**

Quantum circuit follows the same blueprint as Shor.



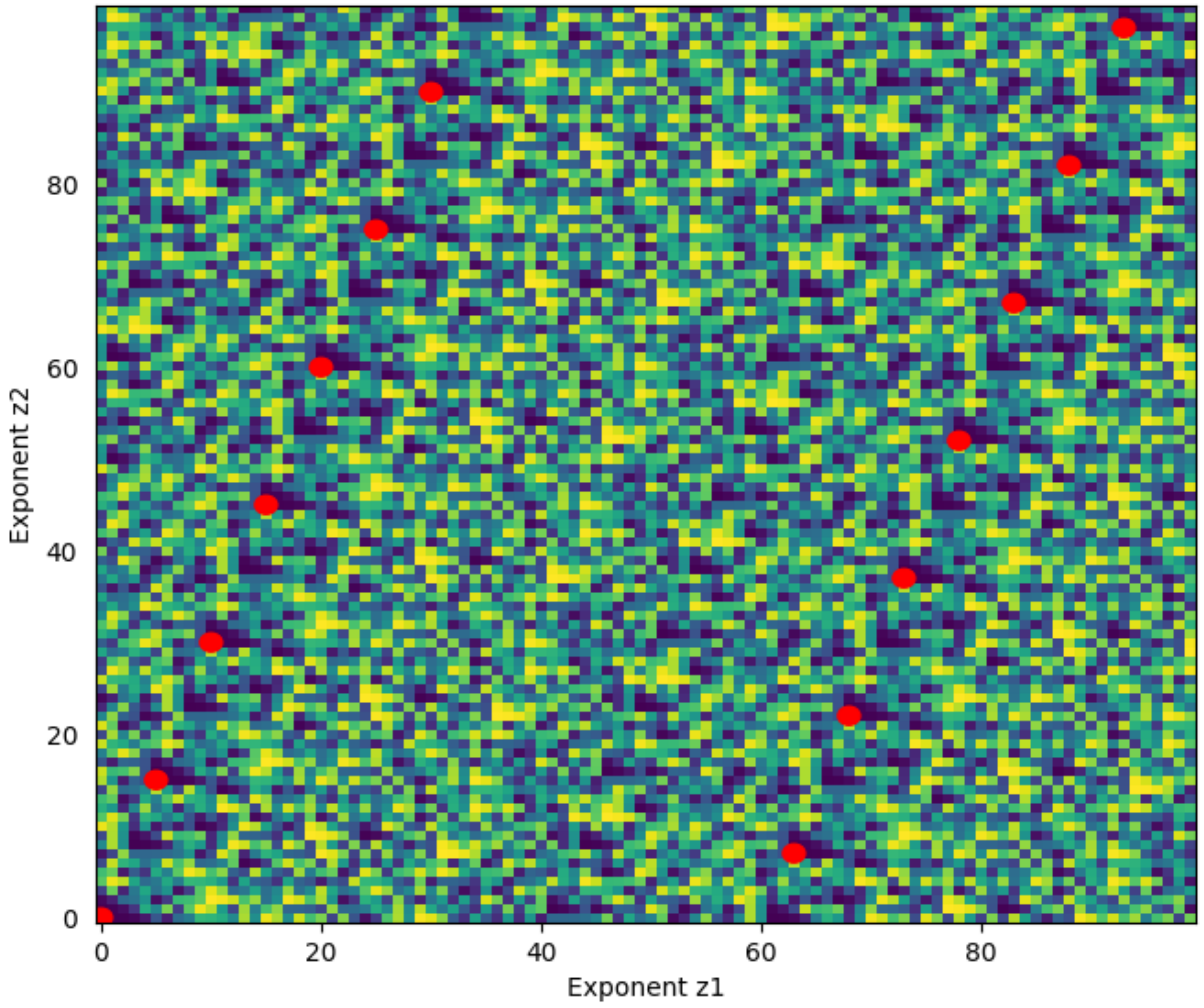
Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d}$ (we considered $d = 2$)



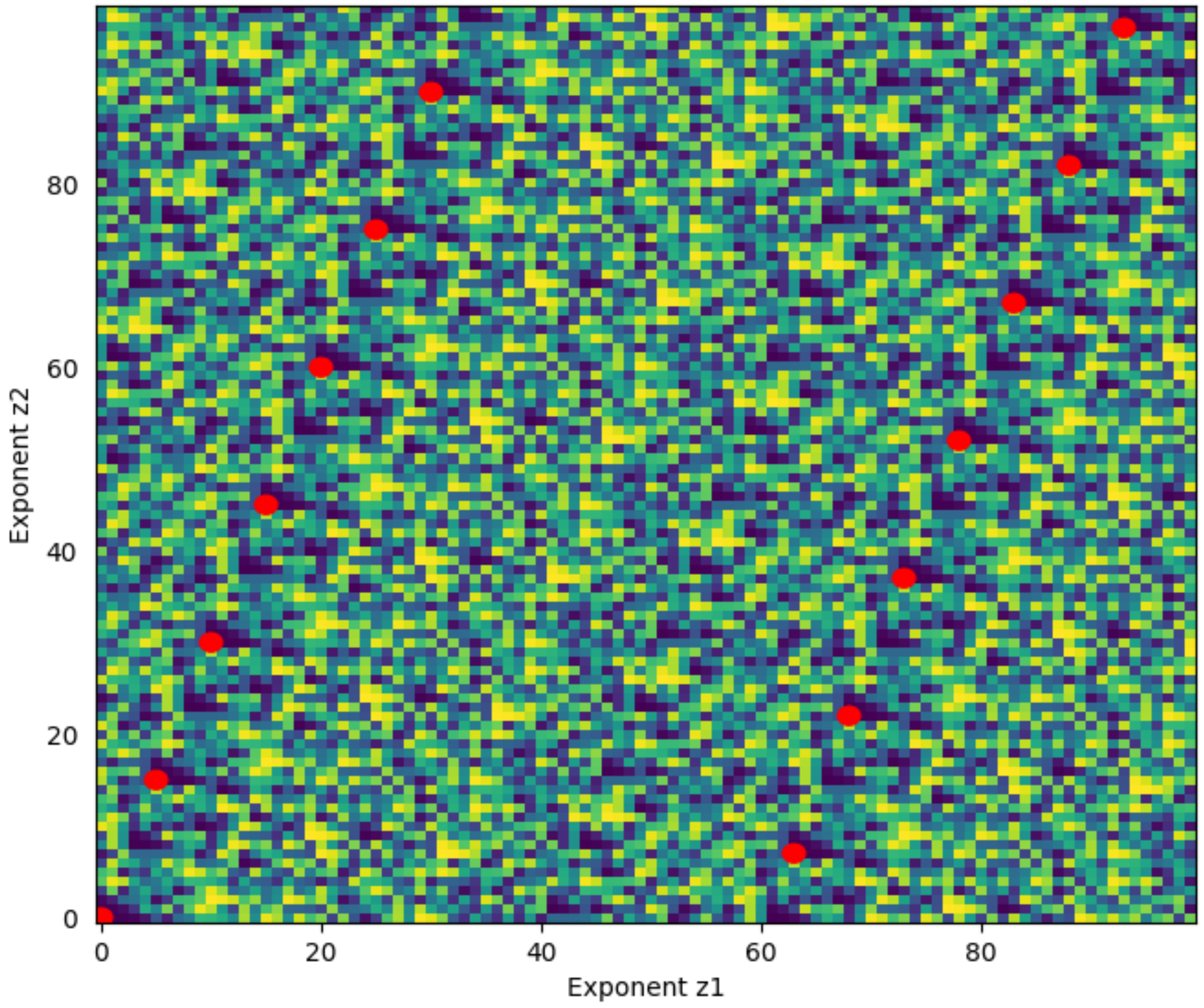
Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d}$ (we considered $d = 2$)
  - Regev's circuit: compute $a_1^{z_1} \ldots a_d^{z_d}$ mod N in superposition for $z_i \leq 2^{n/d}$



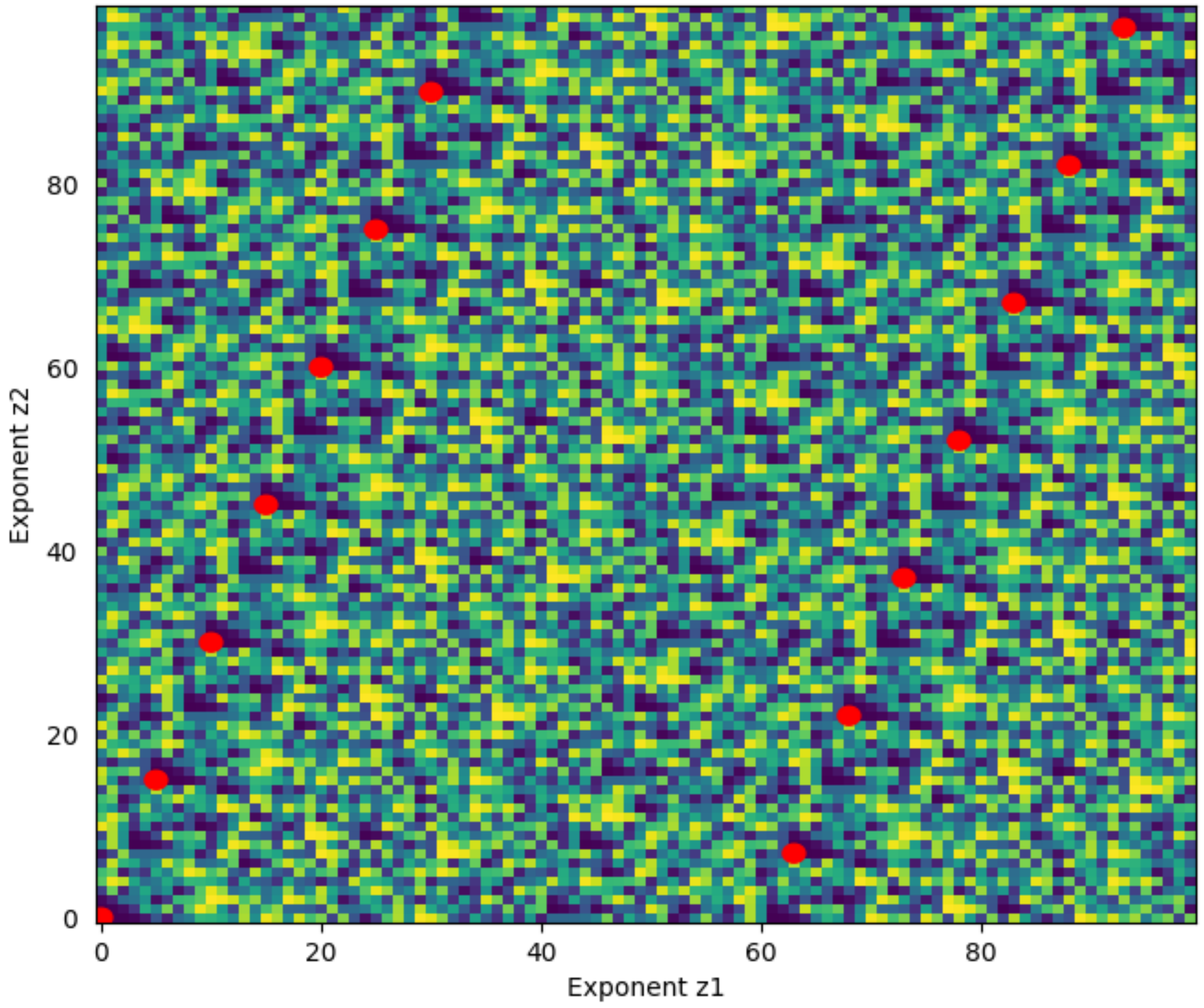
Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d}$ (we considered $d = 2$)
  - Regev's circuit: compute $a_1^{z_1} \ldots a_d^{z_d} \bmod N$ in superposition for $z_i \leq 2^{n/d}$
  - Repeated squaring → at least n/d multiplications needed



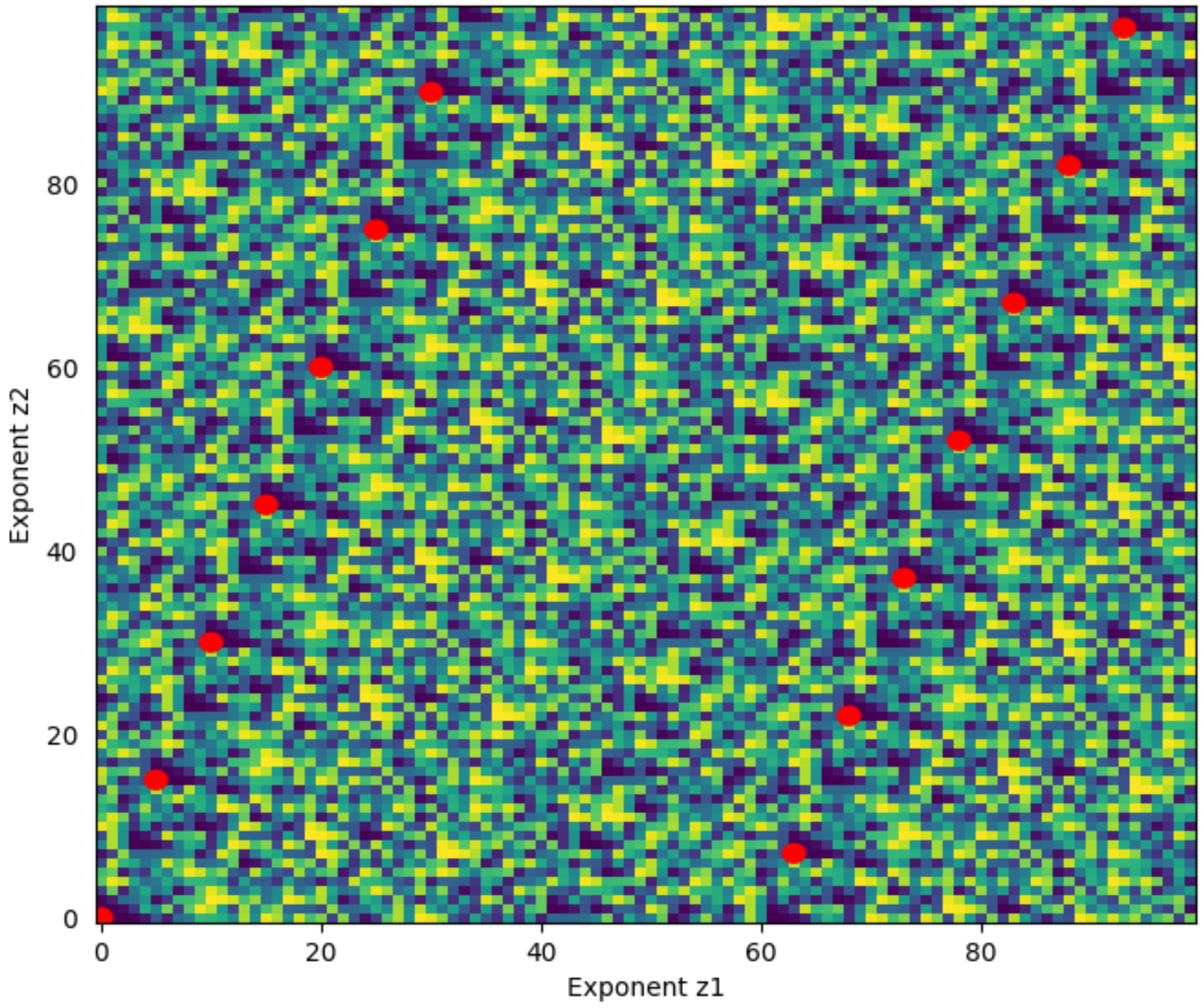
Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d}$ (we considered $d = 2$)
  - Regev's circuit: compute $a_1^{z_1} \dots a_d^{z_d}$ mod N in superposition for $z_i \leq 2^{n/d}$
  - Repeated squaring → <u>at least</u> n/d multiplications needed
  - Regev magic → actually enough! *(Assuming the a_i's are small)*



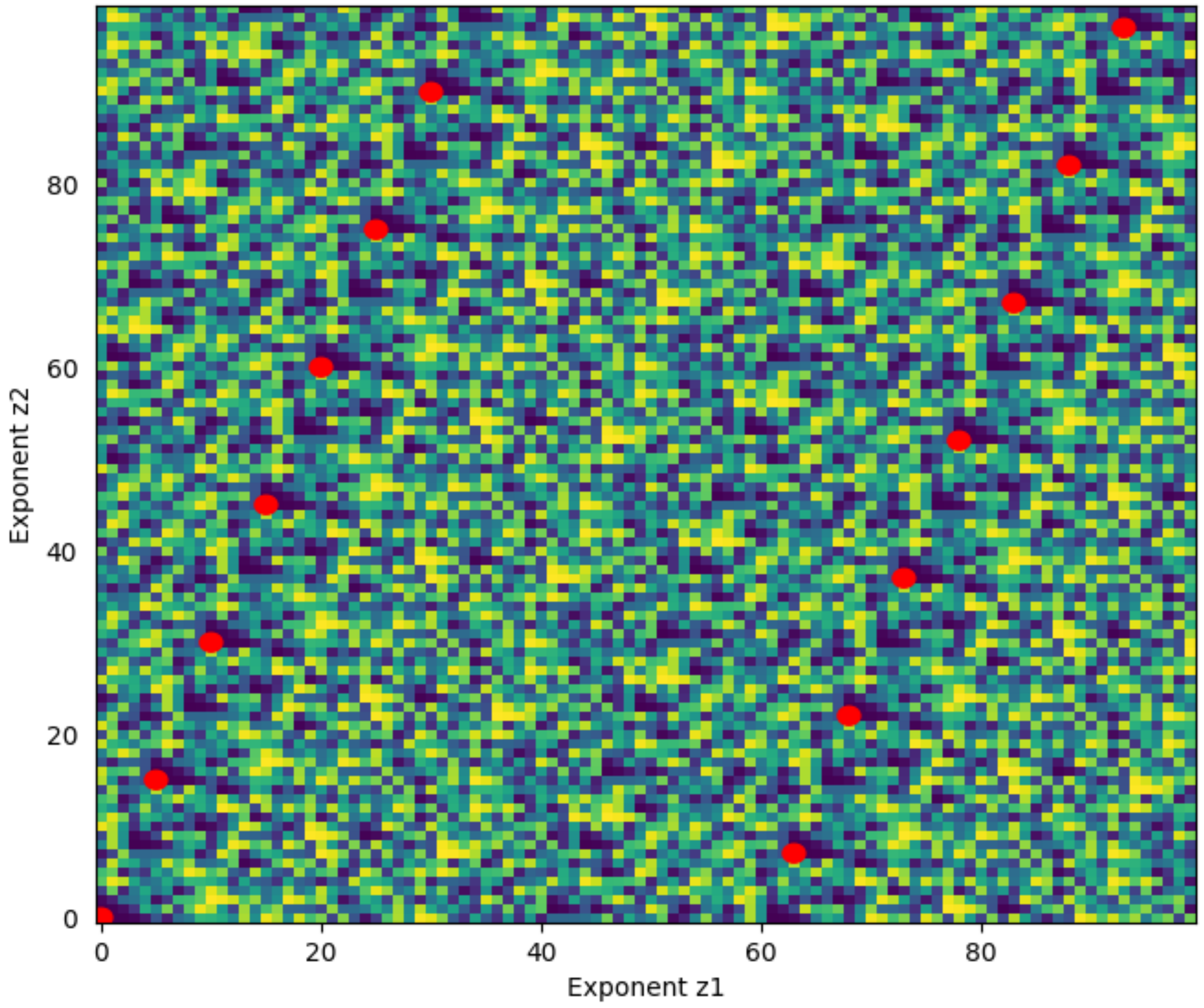
Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d}$ (we considered $d = 2$)
  - Regev's circuit: compute $a_1^{z_1} \dots a_d^{z_d} \bmod N$ in superposition for $z_i \leq 2^{n/d}$
  - Repeated squaring → <u>at least</u> n/d multiplications needed
  - Regev magic → actually enough! *(Assuming the a_i's are small)*
  - In comparison, Shor needs n multiplications → we save a factor of d



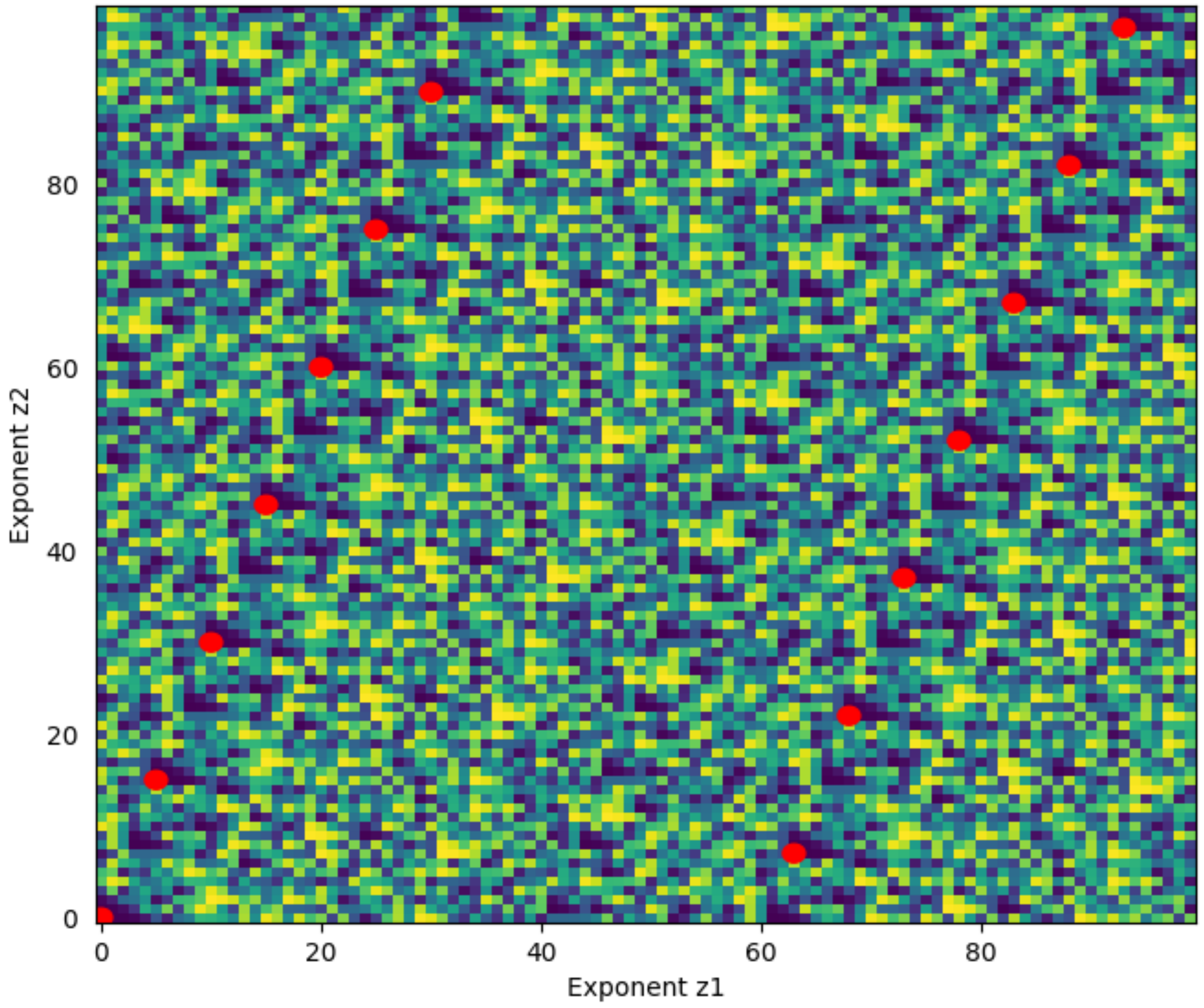
Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d} \rightarrow d \times$ faster



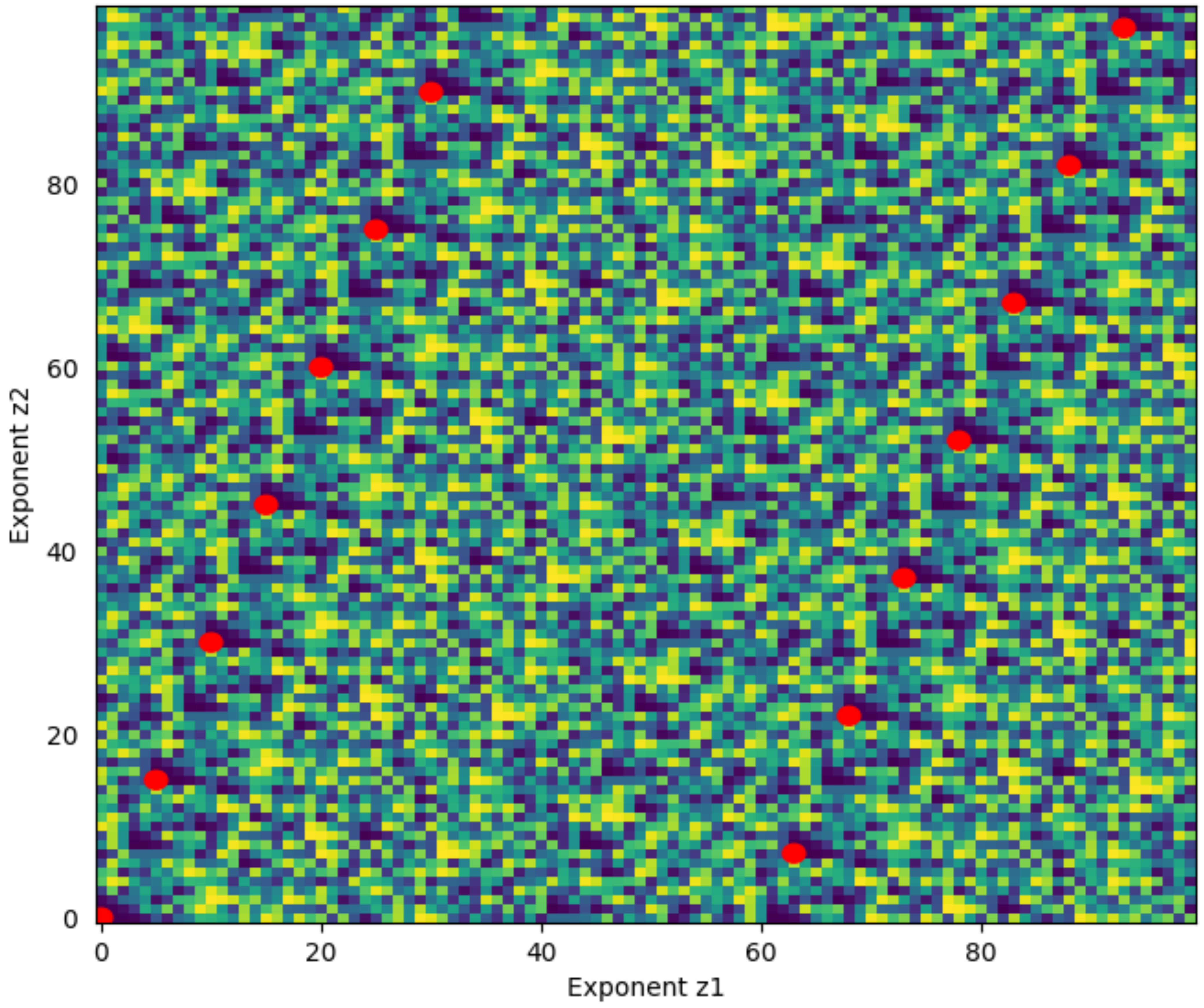
Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d} \rightarrow d \times$ faster
- Q: Why not set $d = n$?
  - Would give a $O(n \log n)$ size quantum circuit for factoring



Regev's algorithm: efficiency

- Size of periods $\approx 2^{n/d} \rightarrow d \times$ faster
- Q: Why not set $d = n$?
  - Would give a $O(n \log n)$ size quantum circuit for factoring
- A: Classical post-processing needs to solve a d-dimensional lattice problem with approximation factor $2^{O(n/d)}$
  - Sweet spot: $d = \sqrt{n}$ (LLL)



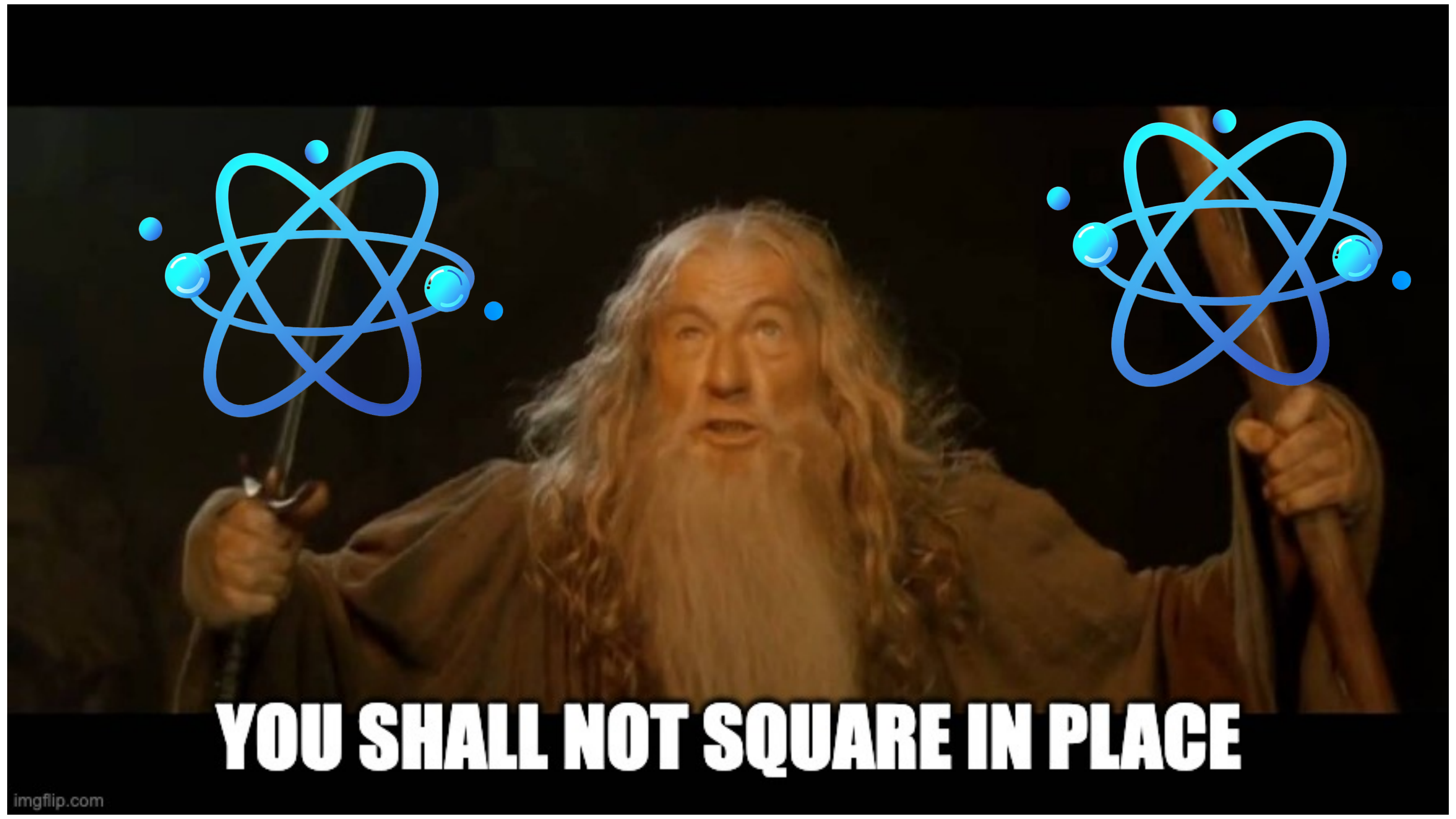
Our Result 1: Improving Space Complexity

The qubit problem

- Performance bottleneck: computing $(z_1, \ldots, z_d) \mapsto a_1^{z_1} \ldots a_d^{z_d} \bmod N$
- Repeated squaring mod N cannot be done in place!

The qubit problem

- Performance bottleneck: computing $(z_1, \ldots, z_d) \mapsto a_1^{z_1} \ldots a_d^{z_d} \bmod N$
- Repeated squaring mod N cannot be done in place!
  - Quantum circuits need to be reversible, but squaring mod N is not e.g. $-1, 1$



The qubit problem

- Performance bottleneck: computing $(z_1, \ldots, z_d) \mapsto a_1^{z_1} \ldots a_d^{z_d} \bmod N$
- Repeated squaring mod N cannot be done in place!
- Instead, Regev has to square *out-of-place*:

$$|a\rangle \rightsquigarrow |a, a^2\rangle \rightsquigarrow |a, a^2, a^4\rangle \rightsquigarrow |a, a^2, a^4, a^8\rangle$$

The qubit problem

- Performance bottleneck: computing $(z_1, \ldots, z_d) \mapsto a_1^{z_1} \ldots a_d^{z_d} \bmod N$
- Repeated squaring mod N cannot be done in place!
- Instead, Regev has to square *out-of-place*:

$$|a\rangle \rightsquigarrow |a, a^2\rangle \rightsquigarrow |a, a^2, a^4\rangle \rightsquigarrow |a, a^2, a^4, a^8\rangle$$

Each squaring adds n qubits $\rightarrow O\left(\frac{n}{d} \times n\right) = O(n^{3/2})$ qubits total.

Aside: qubit complexity of Shor

- Does this mean Shor also requires $O\left(\frac{n}{d} \times n\right) = O(n^2)$ qubits? **No!**
  - Can precompute $a^1, a^2, a^4, a^8, a^{16}, \ldots$ classically
  - Quantum part: instead of squaring, multiply these together

Aside: qubit complexity of Shor

- Does this mean Shor also requires $O\left(\frac{n}{d} \times n\right) = O(n^2)$ qubits? **No!**
  - Can precompute $a^1, a^2, a^4, a^8, a^{16}, \ldots$ classically
  - Quantum part: instead of squaring, multiply these together
    - *Unlike squaring, multiplication can be done reversibly (not obvious... stay tuned)*

Aside: qubit complexity of Shor

- Does this mean Shor also requires $O\left(\frac{n}{d} \times n\right) = O(n^2)$ qubits? **No!**
  - Can precompute $a^1, a^2, a^4, a^8, a^{16}, \ldots$ classically
  - Quantum part: instead of squaring, multiply these together
- **Why doesn't the same precomputation trick work for Regev?**

Aside: qubit complexity of Shor

- Does this mean Shor also requires $O\left(\frac{n}{d} \times n\right) = O(n^2)$ qubits? **No!**
  - Can precompute $a^1, a^2, a^4, a^8, a^{16}, \ldots$ classically
  - Quantum part: instead of squaring, multiply these together
- **Why doesn't the same precomputation trick work for Regev?**
  - # of precomputed bits: d bases, n/d exponents, n-bit answers → $O(n^2)$

Aside: qubit complexity of Shor

- Does this mean Shor also requires $O\left(\frac{n}{d} \times n\right) = O(n^2)$ qubits? **No!**
  - Can precompute $a^1, a^2, a^4, a^8, a^{16}, \ldots$ classically
  - Quantum part: instead of squaring, multiply these together
- **Why doesn't the same precomputation trick work for Regev?**
  - # of precomputed bits: d bases, n/d exponents, n-bit answers → $O(n^2)$
  - Any circuit using these results should require $O(n^2)$ gates

Idea 1: Fibonacci exponentiation

- What if we used two accumulators?

$$|a, b\rangle \rightarrow |a, ab \bmod N\rangle$$

Idea 1: Fibonacci exponentiation

- What if we used two accumulators?

$$|a, b\rangle \to |a, ab \bmod N\rangle$$

- Observation by Kaliski 2017:

$$|a, a\rangle \rightsquigarrow |a, a^2\rangle \rightsquigarrow |a^3, a^2\rangle \rightsquigarrow |a^3, a^5\rangle \rightsquigarrow |a^8, a^5\rangle$$

Idea 1: Fibonacci exponentiation

- What if we used two accumulators?

$$|a, b\rangle \to |a, ab \bmod N\rangle$$

- Observation by Kaliski 2017:

$$|a, a\rangle \rightsquigarrow |a, a^2\rangle \rightsquigarrow |a^3, a^2\rangle \rightsquigarrow |a^3, a^5\rangle \rightsquigarrow |a^8, a^5\rangle$$

We can efficiently compute a^{F_k} for any Fibonacci number F_k!

Idea 2: multiplying in-place using inverses

- How to implement $|a, b\rangle \to |a, ab \bmod N\rangle$?
  - Not reversible if $\gcd(a, N) > 1$! (e.g. $2 \times 1 \equiv 2 \times 4 \bmod 6$)

Idea 2: multiplying in-place using inverses

- How to implement $|a, b\rangle \to |a, ab \bmod N\rangle$?
  - Not reversible if $\gcd(a, N) > 1$! (e.g. $2 \times 1 \equiv 2 \times 4 \bmod 6$)
- Solution based on ideas by Shor: “certify” that $\gcd(a, N) = 1$ by providing $a^{-1} \bmod N$

Idea 2: multiplying in-place using inverses

- How to implement $|a, b\rangle \to |a, ab \bmod N\rangle$?
  - Not reversible if $\gcd(a, N) > 1$! (e.g. $2 \times 1 \equiv 2 \times 4 \bmod 6$)
- Solution based on ideas by Shor: "certify" that $\gcd(a, N) = 1$ by providing $a^{-1} \bmod N$
  - So instead, we implement $|a, a^{-1}, b, b^{-1}\rangle \to |a, a^{-1}, ab, (ab)^{-1}\rangle$

Idea 3: greedy Fibonacci decomposition

- Compute a^z mod N using Fibonacci exponentiation → need to decompose z as a sum of Fibonacci numbers, in superposition

Idea 3: greedy Fibonacci decomposition

- Compute a^z mod N using Fibonacci exponentiation → need to decompose z as a sum of Fibonacci numbers, in superposition
- Zeckendorf 1972: straightforward greedy algorithm
  - Cost of this greedy procedure: $O(n^{3/2})$ gates, $O(n)$ space overhead

Our Space Improvement

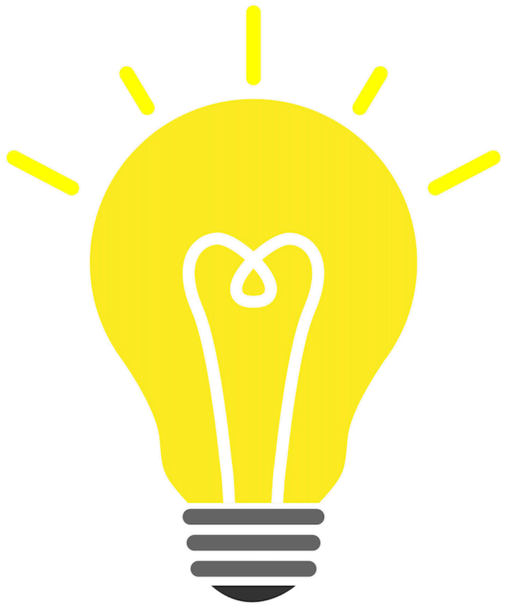
Concrete Results and Summary

- Regev's original circuit: $\approx 3n^{3/2}$ qubits, $\approx 6n^{1/2}$ multiplications mod N
- With our space optimisations: $\approx 10.4n$ qubits, $\approx 45.7n^{1/2}$ multiplications mod N

Our Space Improvement

Concrete Results and Summary

- Regev's original circuit: $\approx 3n^{3/2}$ qubits, $\approx 6n^{1/2}$ multiplications mod N
- With our space optimisations: $\approx 10.4n$ qubits, $\approx 45.7n^{1/2}$ multiplications mod N



Squaring mod N *cannot be implemented in place, while multiplication* mod N *can* $\rightarrow$ *can exponentiate with two registers using Fibonacci exponentiation,* ***without consuming additional space per step.***

Our Result 2: Improving Noise Tolerance

Output of Regev's Circuit

- Define the following lattices:

$$\mathscr{L} = \left\{ \mathbf{z} \in \mathbb{Z}^d : a_1^{z_1} \ldots a_d^{z_d} \equiv 1 \bmod N \right\}$$

$$\mathscr{L}^* = \left\{ \mathbf{y} \in \mathbb{R}^d : \langle \mathbf{y}, \mathbf{z} \rangle \in \mathbb{Z} \, \forall \mathbf{z} \in \mathscr{L} \right\}$$

- Regev's circuit: outputs a vector from $\mathscr{L}^*$, plus exponentially small ℓ_2 noise

Output of Regev's Circuit

- Define the following lattices:

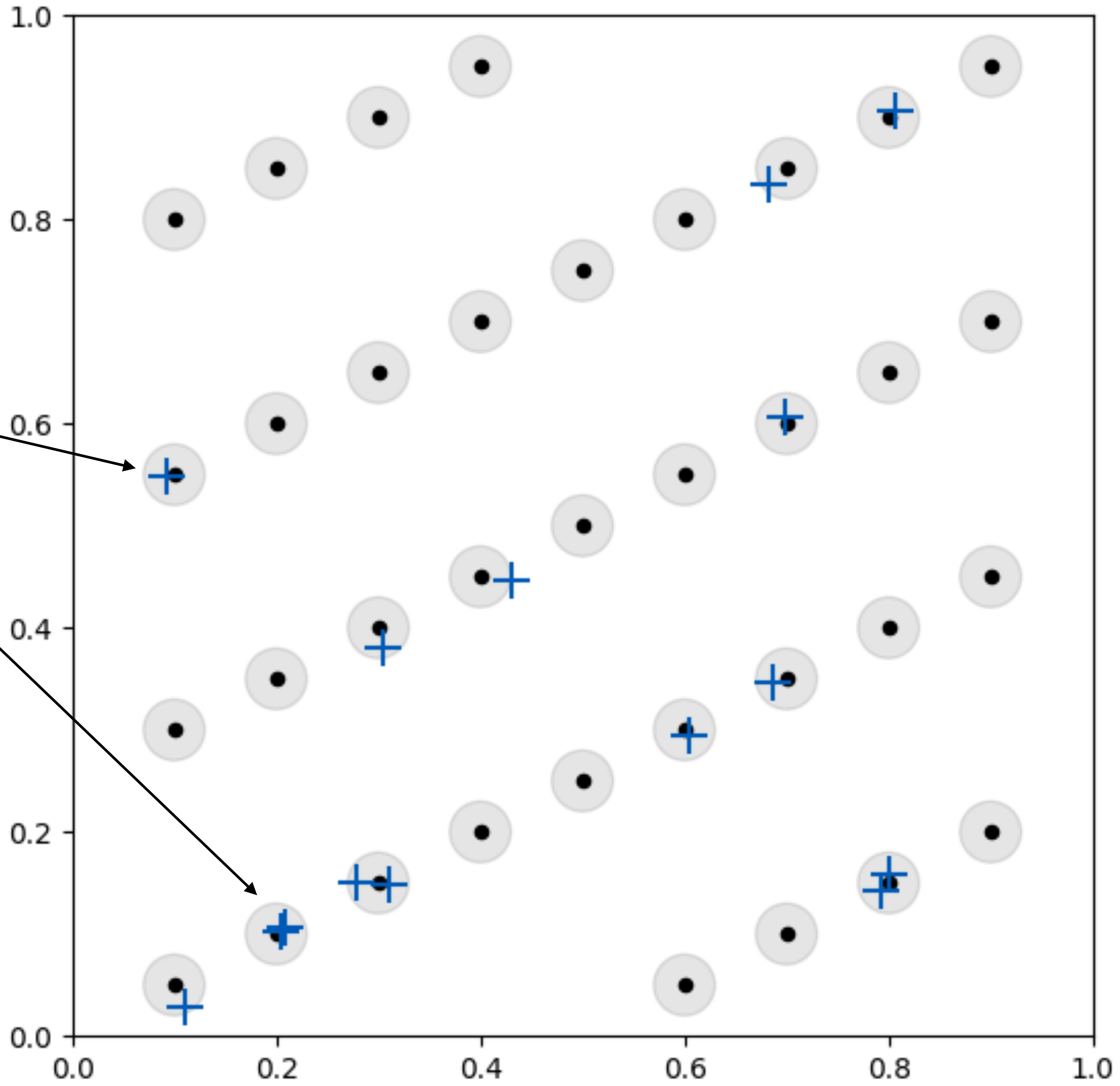
$$\mathscr{L} = \left\{ \mathbf{z} \in \mathbb{Z}^d : a_1^{z_1} \ldots a_d^{z_d} \equiv 1 \bmod N \right\}$$

$$\mathscr{L}^* = \left\{ \mathbf{y} \in \mathbb{R}^d : \langle \mathbf{y}, \mathbf{z} \rangle \in \mathbb{Z} \, \forall \mathbf{z} \in \mathscr{L} \right\}$$

- Regev's circuit: outputs a vector from $\mathscr{L}^*$, plus exponentially small ℓ_2 noise
- **Role of the dual lattice is analogous to Shor's and Simon's algorithms**
  - Shor: $\mathscr{L} = \{0, r, 2r, \ldots\}, \mathscr{L}^* = \{0, 1/r, 2/r, \ldots\}$
  - Simon (in $\mathbb{Z}_2^n$): $\mathscr{L} = \{0^n, \mathbf{s}\}, \mathscr{L}^* = \left\{ \mathbf{y} \in \mathbb{Z}_2^n : \langle \mathbf{y}, \mathbf{s} \rangle = 0 \right\}$

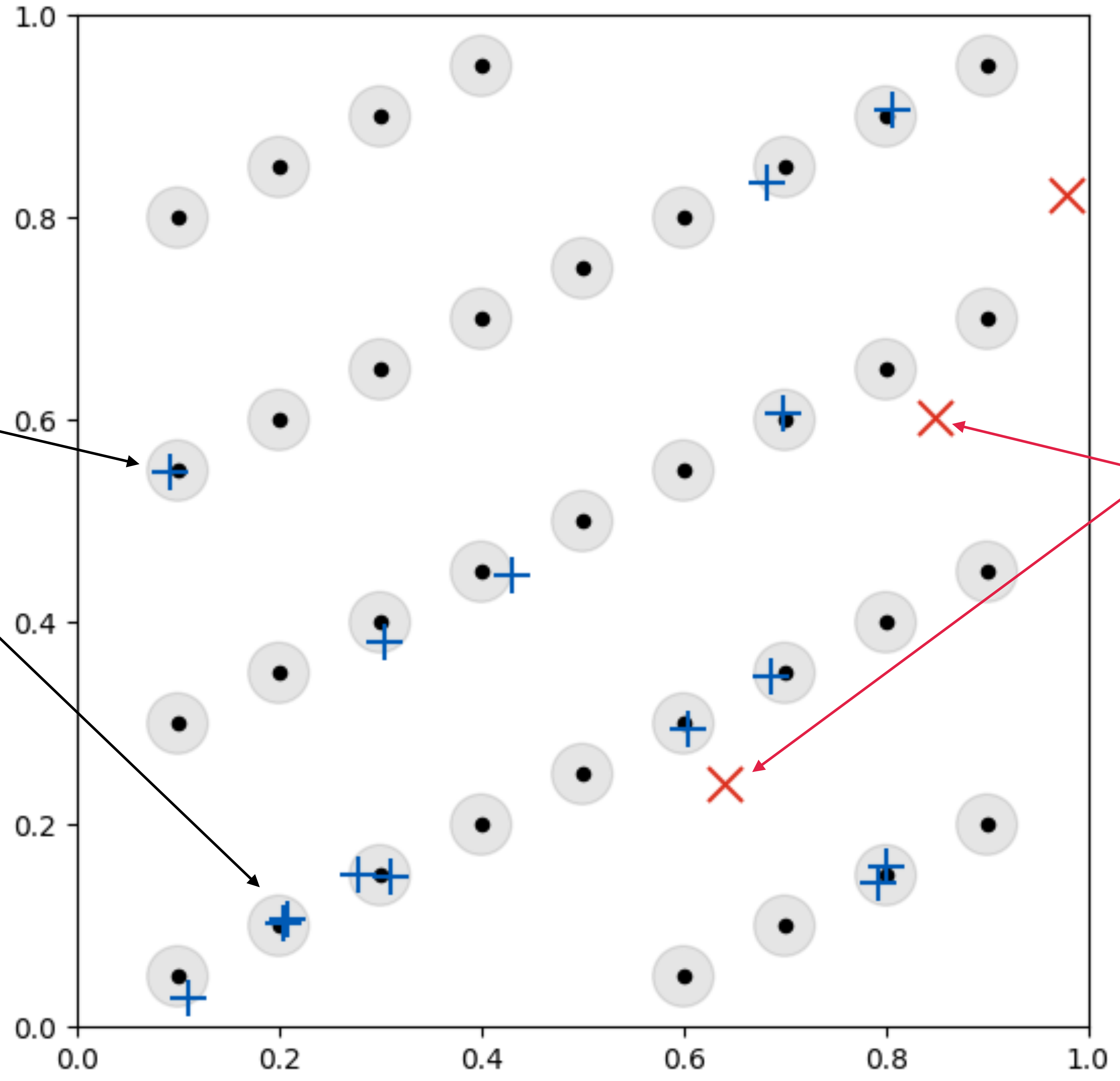
Regev's classical post-processing

Regev: given $O(d)$ samples from $\mathscr{L}^*$ with ℓ_2 noise, roughly outputs a basis for $\mathscr{L}$



Regev's classical post-processing

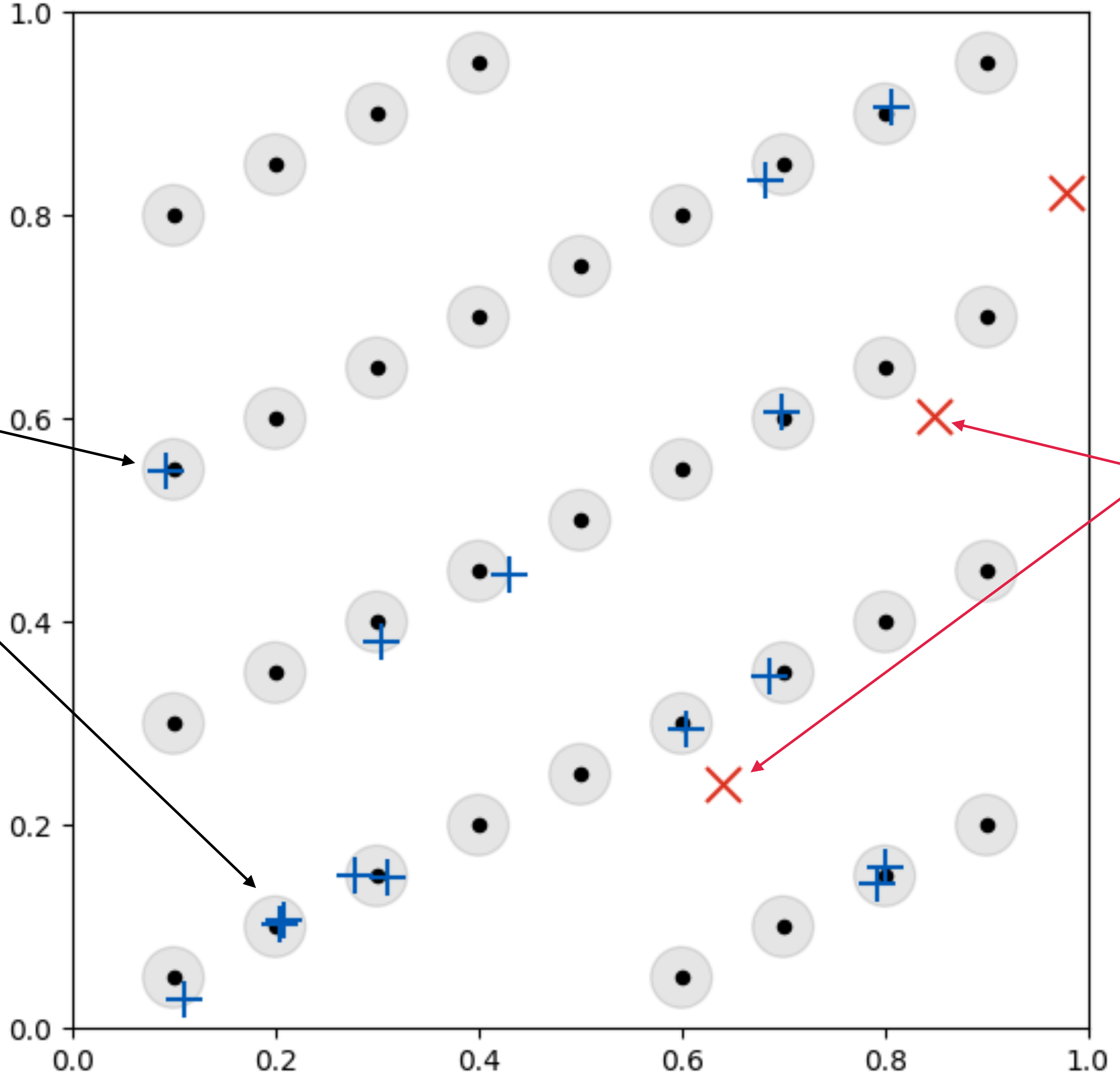
Regev: given $O(d)$ samples from $\mathscr{L}^*$ with ℓ_2 noise, roughly outputs a basis for $\mathscr{L}$



With logical errors: some Hamming errors as well (say uniform over $[0, 1]^d$)

Regev's classical post-processing

Regev: given $O(d)$ samples from $\mathscr{L}^*$ with ℓ_2 noise, roughly outputs a basis for $\mathscr{L}$

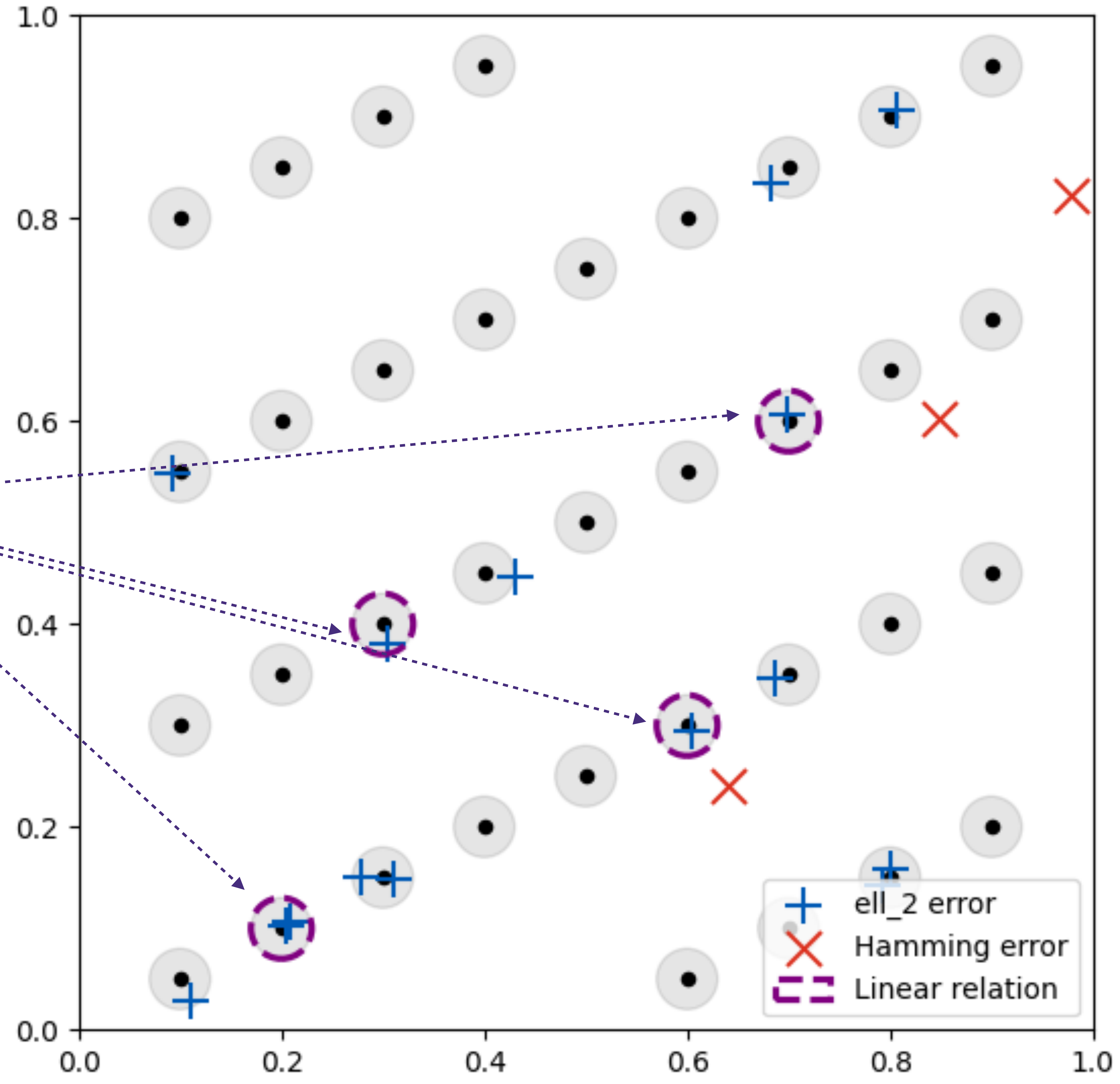


With logical errors: some Hamming errors as well (say uniform over $[0, 1]^d$)

Our result: we can detect and filter out these Hamming errors, then use Regev's procedure to handle the ℓ_2 errors

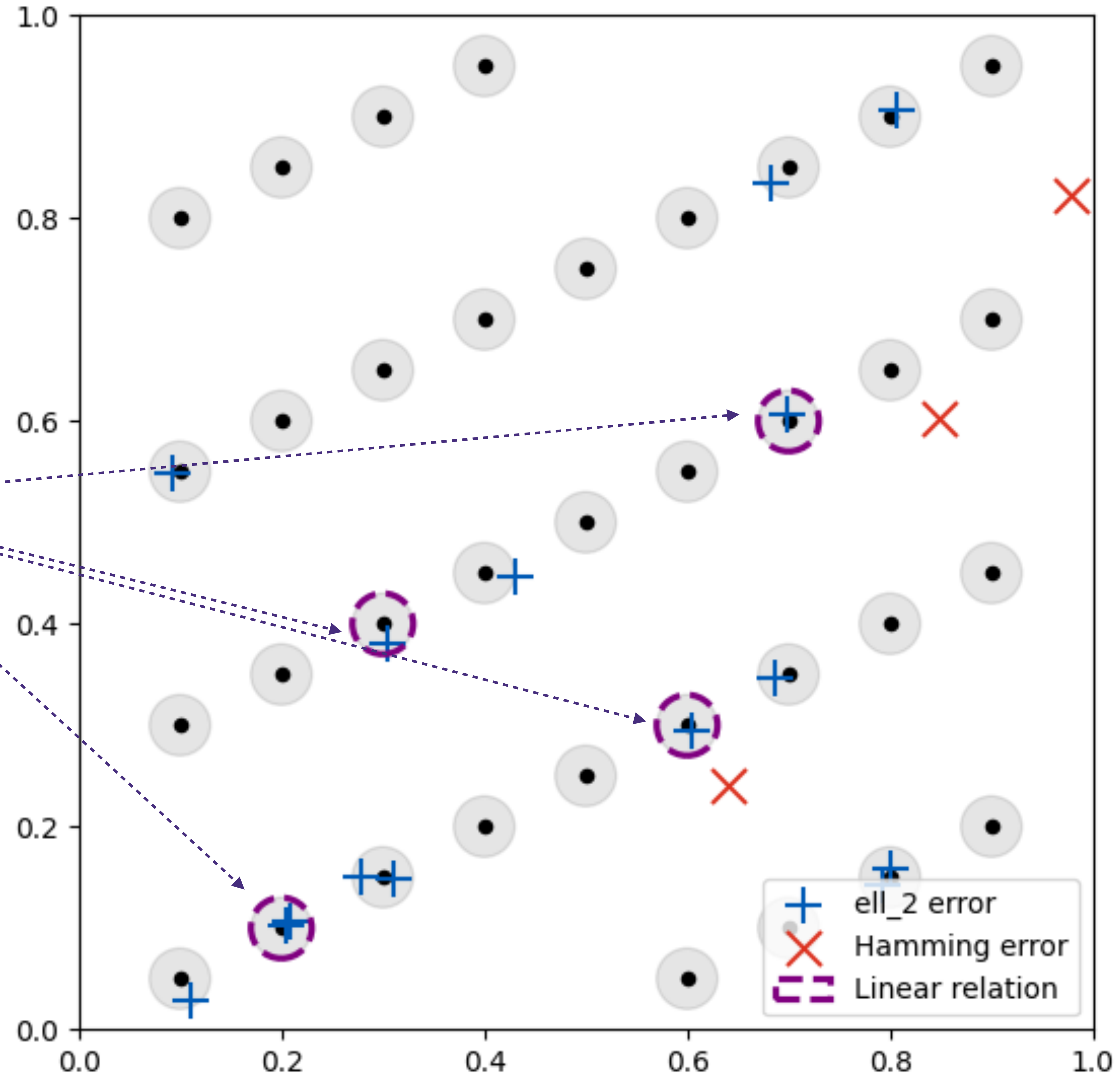
The Main Idea

Observation: the samples that only have ℓ_2 errors will share small approximate linear relations



The Main Idea

Observation: the samples that only have ℓ_2 errors will share small approximate linear relations



Moreover, w.h.p. there is no small approximate linear relation that includes a dirty sample (Hamming error)

Formulation as a Lattice Problem

- Finding an approximate linear relation between $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m \in [0, 1]^d$ is really an "approximate SIS" problem:

$$\textit{Find short } \mathbf{b} \in \mathbb{Z}^m \textit{ such that } [\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_m] \cdot \mathbf{b} \bmod 1 \in [0, 1]^d \textit{ is short}$$

Formulation as a Lattice Problem

- Finding an approximate linear relation between $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m \in [0, 1]^d$ is really an "approximate SIS" problem:

$$\textit{Find short } \mathbf{b} \in \mathbb{Z}^m \textit{ such that } [\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_m] \cdot \mathbf{b} \bmod 1 \in [0, 1]^d \textit{ is short}$$

- Can be solved with exponential approximation error using LLL

Formulation as a Lattice Problem

- Finding an approximate linear relation between $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m \in [0, 1]^d$ is really an "approximate SIS" problem:

$$\textit{Find short } \mathbf{b} \in \mathbb{Z}^m \textit{ such that } [\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_m] \cdot \mathbf{b} \bmod 1 \in [0, 1]^d \textit{ is short}$$

- Can be solved with exponential approximation error using LLL
- Our full algorithm:
  - Solve the above problem to obtain $\mathbf{b}$

Formulation as a Lattice Problem

- Finding an approximate linear relation between $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m \in [0, 1]^d$ is really an "approximate SIS" problem:

$$\textit{Find short } \mathbf{b} \in \mathbb{Z}^m \textit{ such that } [\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_m] \cdot \mathbf{b} \bmod 1 \in [0, 1]^d \textit{ is short}$$

- Can be solved with exponential approximation error using LLL
- Our full algorithm:
    - Solve the above problem to obtain $\mathbf{b}$
    - W.h.p., for any $i \in [m]$ such that $\mathbf{b}_i \neq 0$, $\mathbf{v}_i$ is only an ℓ_2 error

Formulation as a Lattice Problem

- Finding an approximate linear relation between $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m \in [0, 1]^d$ is really an "approximate SIS" problem:

$$\textit{Find short } \mathbf{b} \in \mathbb{Z}^m \textit{ such that } [\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_m] \cdot \mathbf{b} \bmod 1 \in [0, 1]^d \textit{ is short}$$

- Can be solved with exponential approximation error using LLL
- Our full algorithm:
  - Solve the above problem to obtain $\mathbf{b}$
  - W.h.p., for any $i \in [m]$ such that $\mathbf{b}_i \neq 0$, $\mathbf{v}_i$ is only an ℓ_2 error
  - Repeat until we have enough "good indices" i, and feed these samples to Regev's post-processing procedure

Our Final Post-Processing Result

Concrete Results and Summary

- Regev's original algorithm: $\approx 6n^{1/2}$ multiplications per run, $\approx n^{1/2}$ runs without logical error
    - Requires a multiplication circuit with logical error probability $\approx 0.17n^{-1}$

Our Final Post-Processing Result

Concrete Results and Summary

- Regev's original algorithm: $\approx 6n^{1/2}$ multiplications per run, $\approx n^{1/2}$ runs without logical error
  - Requires a multiplication circuit with logical error probability $\approx 0.17n^{-1}$
- Our algorithm: $\approx 12.0n^{1/2}$ multiplications, $\approx 10.8n^{1/2}$ runs with logical error probability of ≈ 0.09 <u>per run</u>
  - Requires a multiplication circuit with logical error probability $\approx 0.0077n^{-1/2}$

Our Final Post-Processing Result

Concrete Results and Summary

- Regev's original algorithm: $\approx 6n^{1/2}$ multiplications per run, $\approx n^{1/2}$ runs without logical error
  - Requires a multiplication circuit with logical error probability $\approx 0.17n^{-1}$
- Our algorithm: $\approx 12.0n^{1/2}$ multiplications, $\approx 10.8n^{1/2}$ runs with logical error probability of ≈ 0.09 per run
  - Requires a multiplication circuit with logical error probability $\approx 0.0077n^{-1/2}$
- Crossover point* is $n \approx 500$

** if considering the error per multiplication per qubit instead, crossover point would be closer to 2000*

Our Final Post-Processing Result

Concrete Results and Summary

- Regev's original algorithm: $\approx 6n^{1/2}$ multiplications per run, $\approx n^{1/2}$ runs without logical error → need multiplication with error prob. $\approx 0.17n^{-1}$
- Our algorithm: $\approx 12.0n^{1/2}$ multiplications, $\approx 10.8n^{1/2}$ runs with logical error probability of ≈ 0.09 <u>per run</u> → need multiplication with error prob. $\approx 0.0077n^{-1/2}$
- Crossover point is $n \approx 500$

The "good" samples (ℓ_2 errors) will share (approximate) linear relations that we can search for using LLL → filter out Hamming errors.

Future direction: why $\tilde{O}(n^{3/2})$ gates?

- With polynomial-time classical post-processing: seems like an artifact of Regev's specific algorithm

Future direction: why $\tilde{O}(n^{3/2})$ gates?

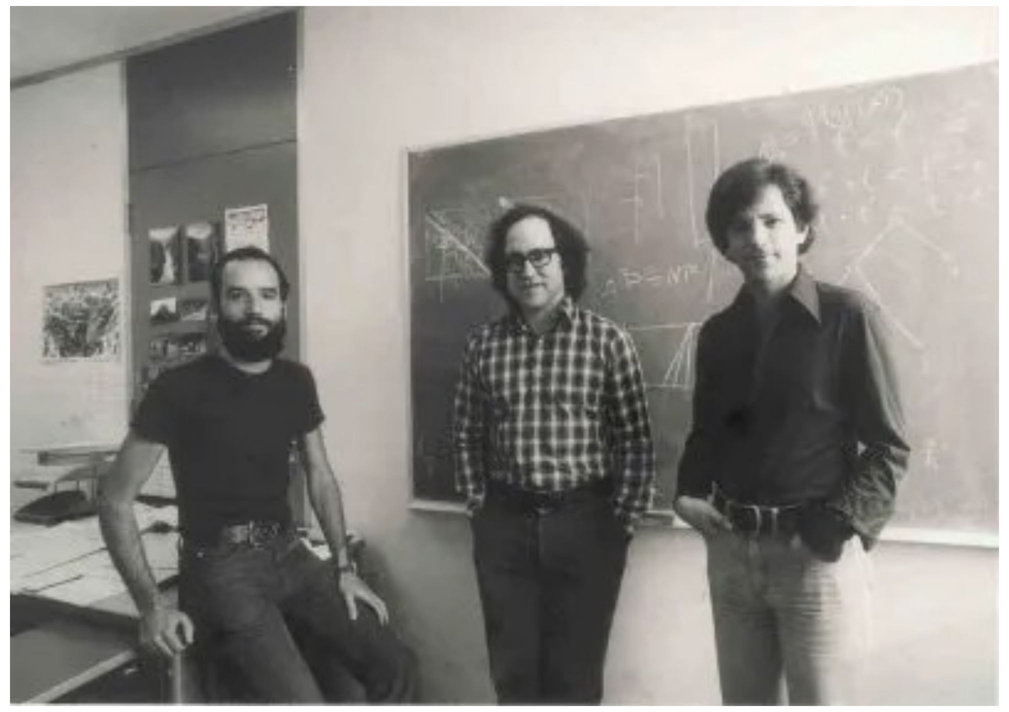
- With polynomial-time classical post-processing: seems like an artifact of Regev's specific algorithm
- What if we allowed superpolynomial-time classical post-processing?
  - Classical post-processing as slow as $2^{O(n^{1/3-\epsilon})}$ is still faster than the number field sieve!

Future direction: why $\tilde{O}(n^{3/2})$ gates?

- With polynomial-time classical post-processing: seems like an artifact of Regev's specific algorithm
- What if we allowed superpolynomial-time classical post-processing?
  - Classical post-processing as slow as $2^{O(n^{1/3-\epsilon})}$ is still faster than the number field sieve!
  - In this case, can set $d = n^{2/3-\epsilon}$ in Regev's algorithm and obtain a circuit of size $\tilde{O}(n^{4/3+\epsilon})$, but can we do better?

Future direction: concrete efficiency

Q: Does Regev's algorithm (and its follow-ups by us and Ekerå-Gärtner) bring us closer to breaking RSA-2048?

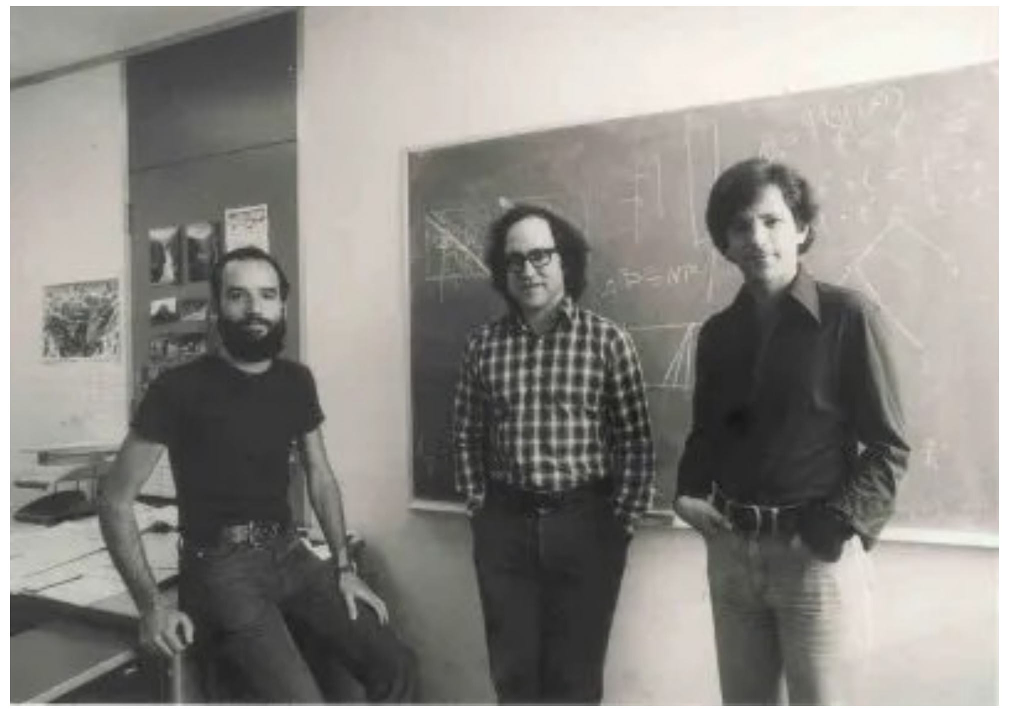


Future direction: concrete efficiency

Q: Does Regev's algorithm (and its follow-ups by us and Ekerå-Gärtner) bring us closer to breaking RSA-2048?

A: It might with further optimisations, but not yet.

Several constant and polylog-factor optimisations to Shor make it very effective for small problem sizes.



Bonus Slides

Modelling Hamming Errors

- Regev's circuit without logical errors: a point close to $\mathscr{L}^*$
- With logical errors: we assume uniform over $[0, 1]^d$ for simplicity
  - Does not neatly map to particular types of quantum errors (X, Z etc.)
  - Heuristic justification: circuit is complicated enough → any error should be "heavily scrambled" by later gates
- We also provide a more general condition for the error distribution: *"linear combinations of Hamming errors should not fall unreasonably close to $\mathscr{L}^*$"*

Regev's Number-Theoretic Assumption

- Regev: relies on $(z_1, \ldots, z_d) \mapsto a_1^{z_1} \ldots a_d^{z_d} \bmod N$ having periods of size $2^{O(n/d)}$
- **But these periods could just yield a trivial square root of 1** $\bmod N$
- Regev relies on a conjecture that *at least one* small period yields a non-trivial square root of 1
- Follow-up work by Pilatte proves* Regev's conjecture

** proves correctness for a variant of Regev's algorithm that is worse by polylog factors and likely impractical*