

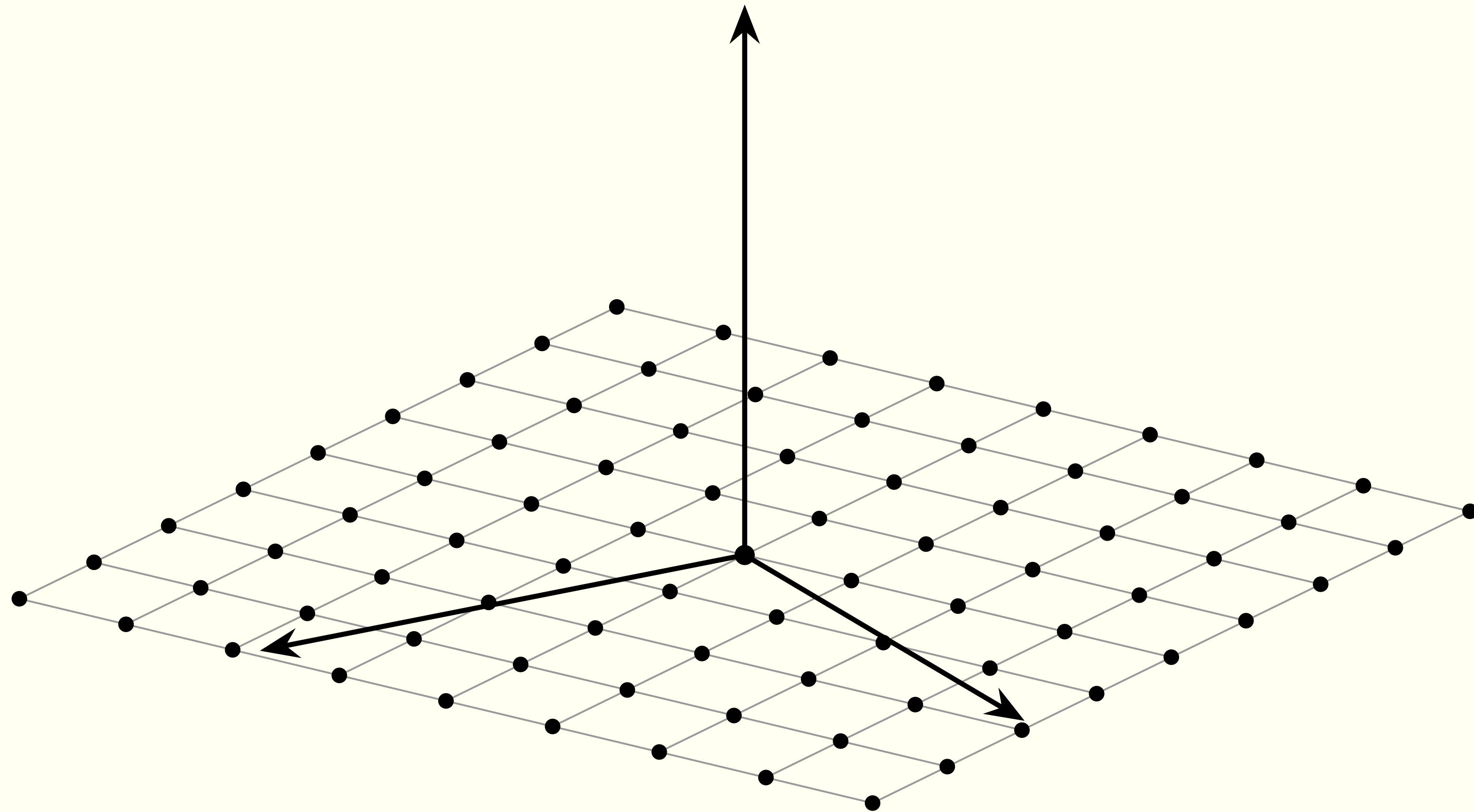
The code intersection problem and applications to quantum complexity

John Bostanci

Based on work with Andrew Huang and Vinod Vaikuntanathan

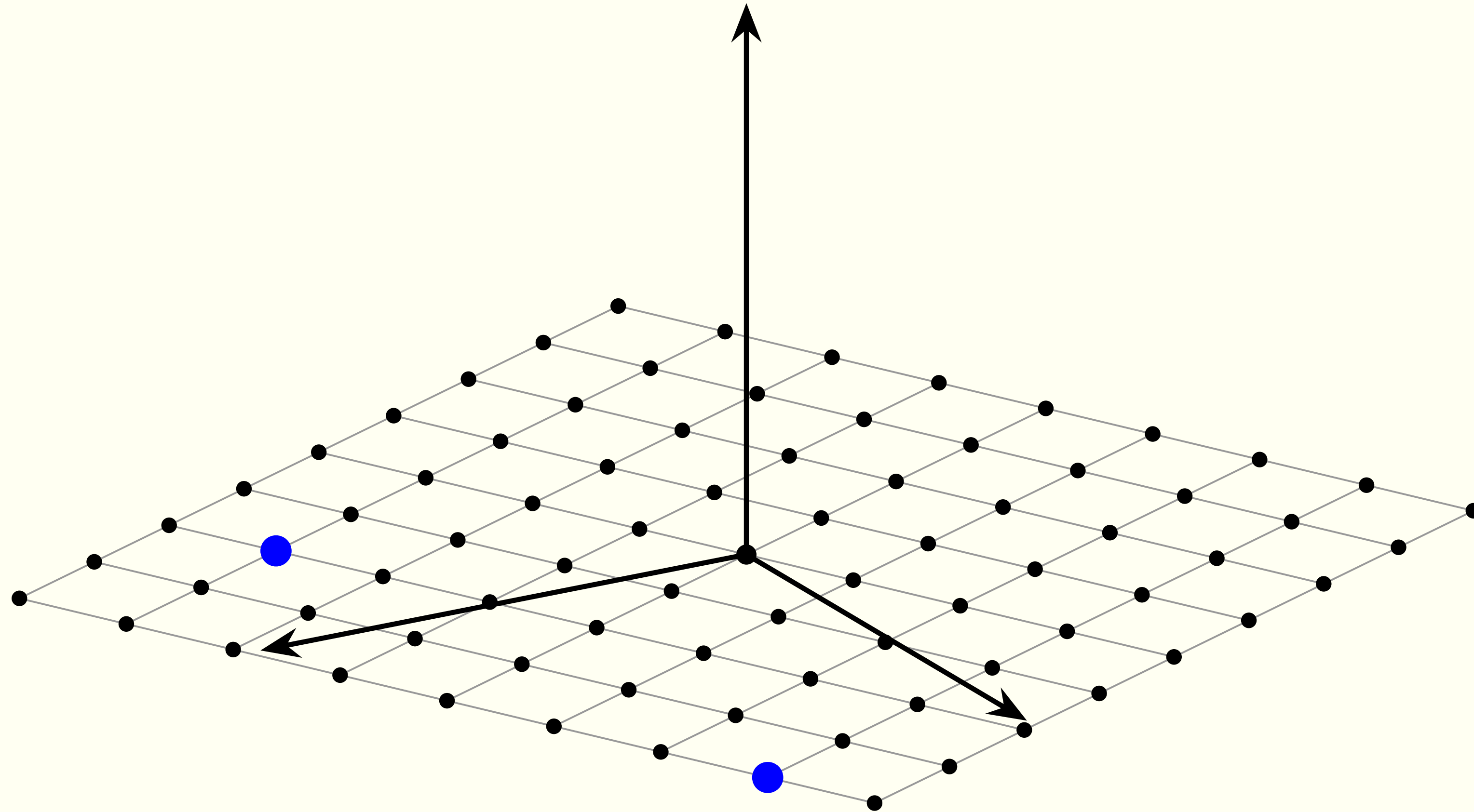
Regev's reduction and the short vector problem

Before 2005, Oded Regev was studying algorithms for the shortest vector problem.



Regev's reduction and the short vector problem

Before 2005, Oded Regev was studying algorithms for the shortest vector problem.
Input: A basis for a lattice (i.e., a collection of linearly independent vectors)

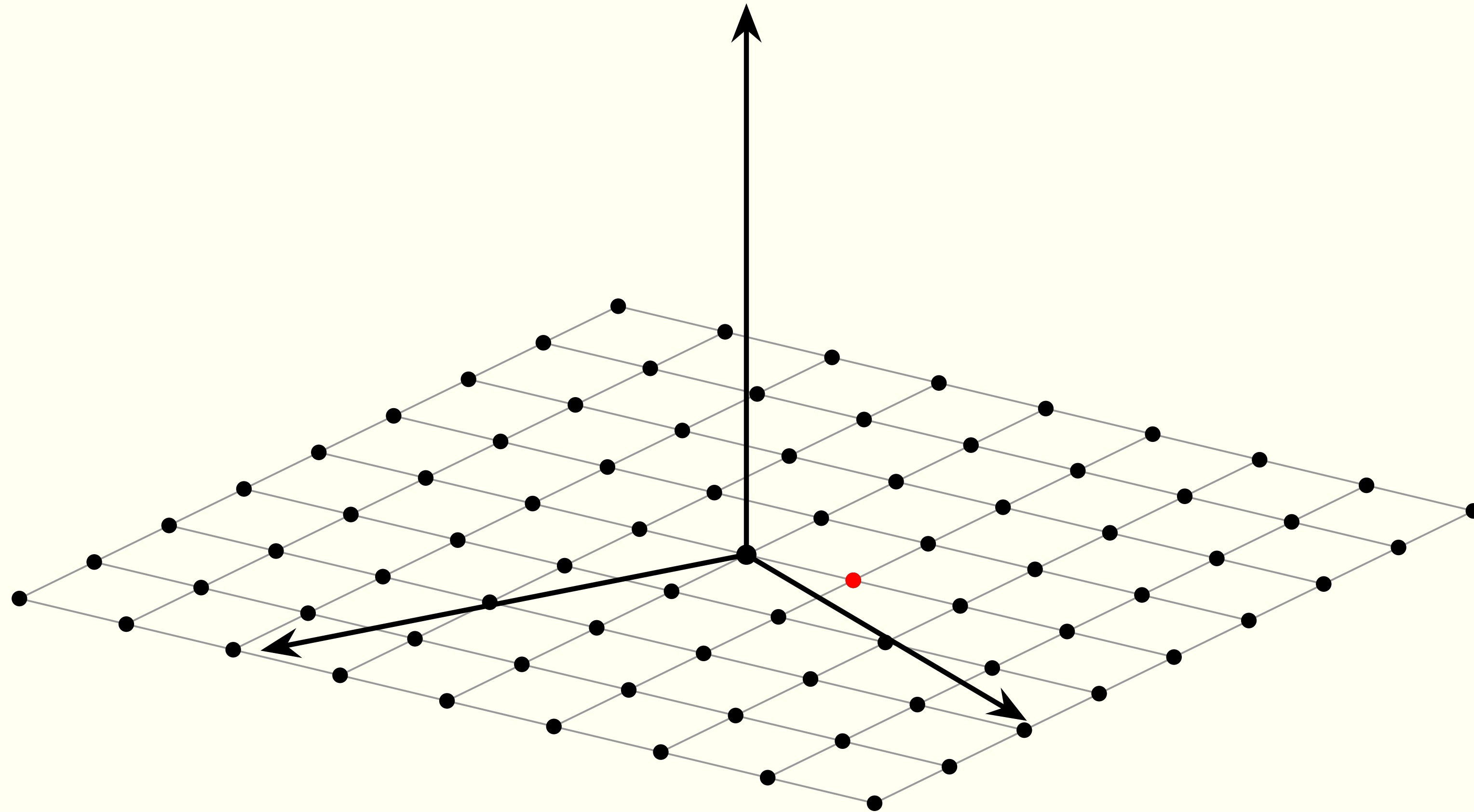


Regev's reduction and the short vector problem

Before 2005, Oded Regev was studying algorithms for the shortest vector problem.

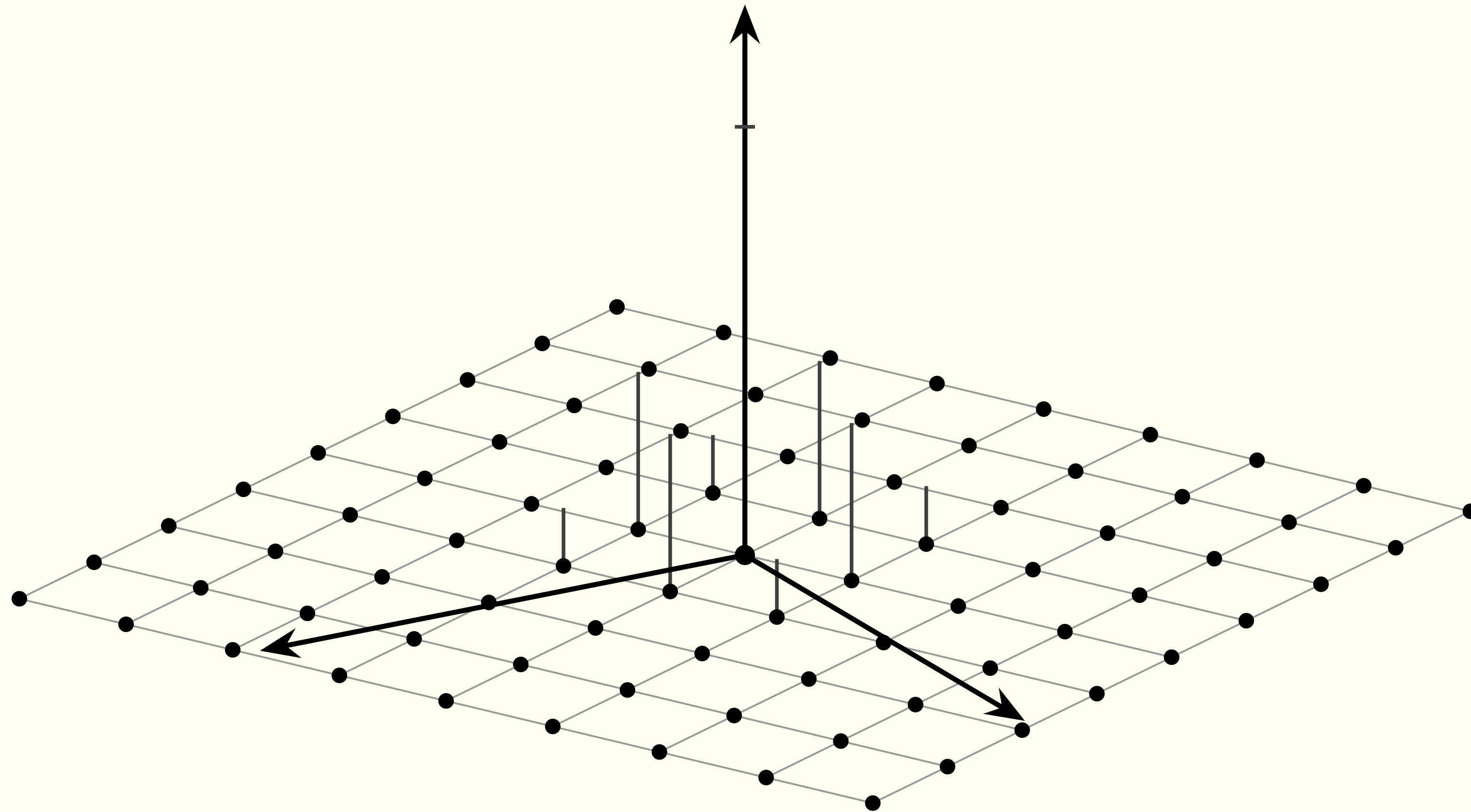
Input: A basis for a lattice (i.e., a collection of linearly independent vectors)

Output: A vector in the lattice that is close to the origin.



Regev's reduction and the short vector problem

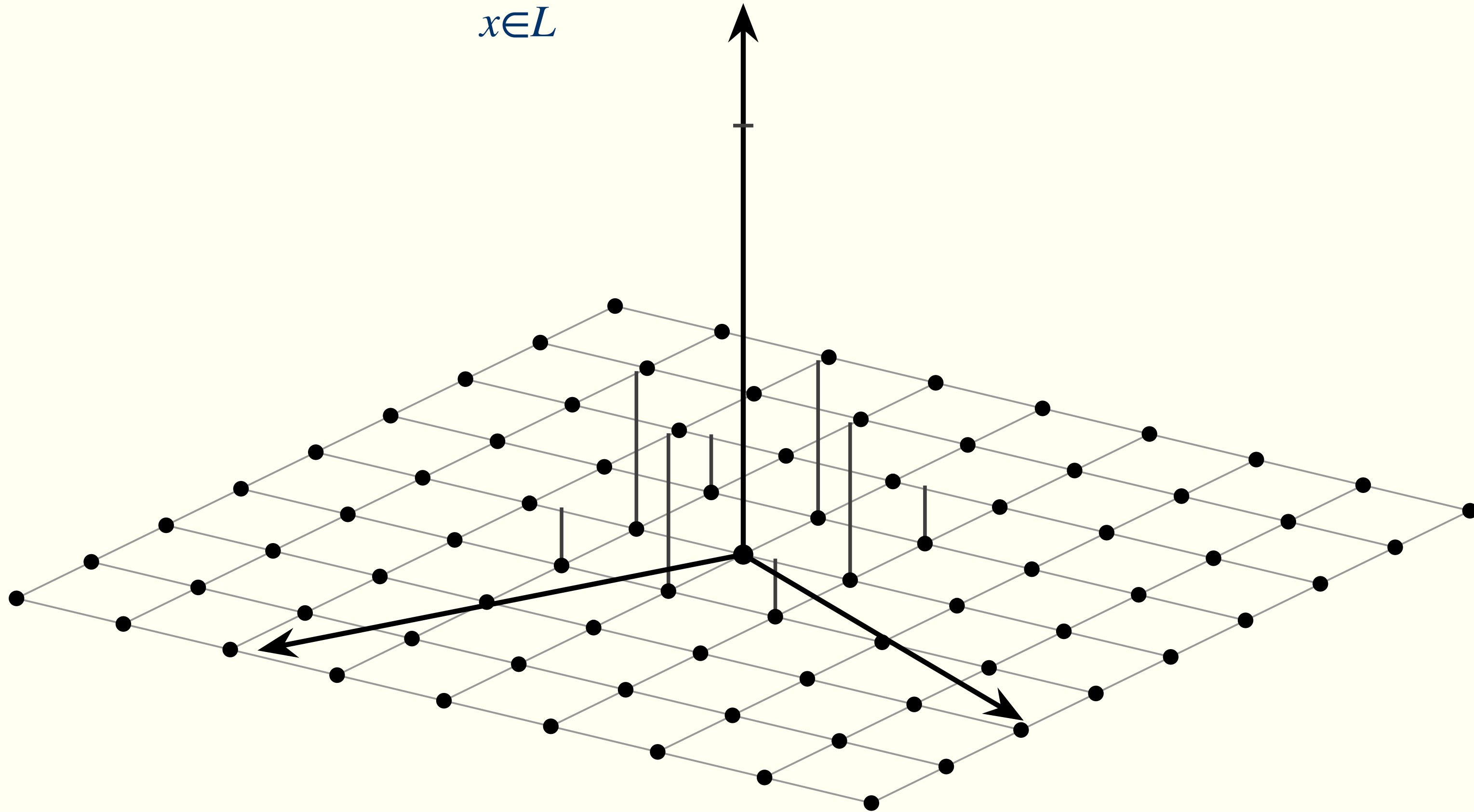
Starting point: Can we sample points from the lattice, according to a Gaussian distribution?



Regev's reduction and the short vector problem

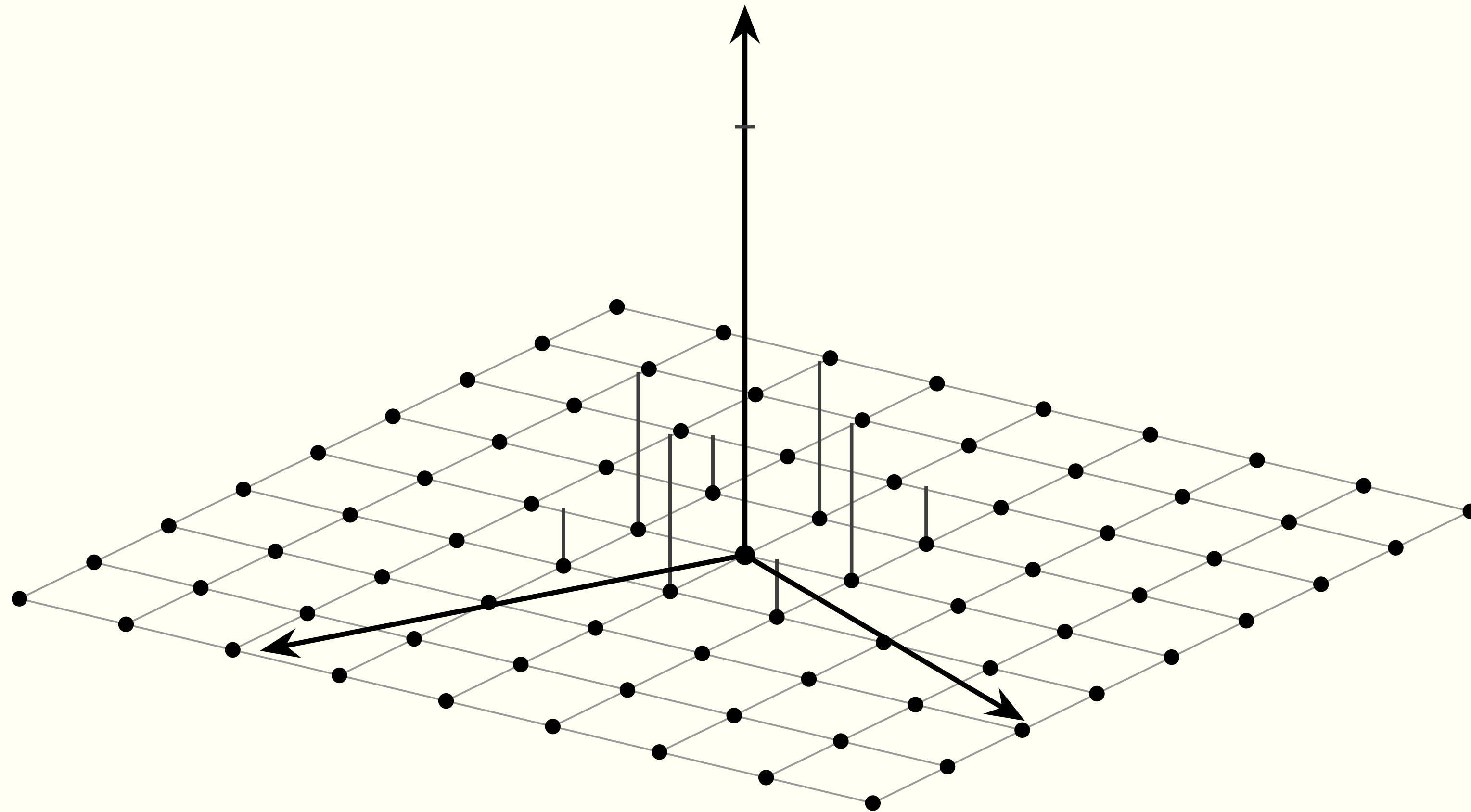
Since we're quantum, we can imagine this as being represented by a quantum state

$$\sum_{x \in L} \sqrt{\text{Gaussian}(x)} |x\rangle$$



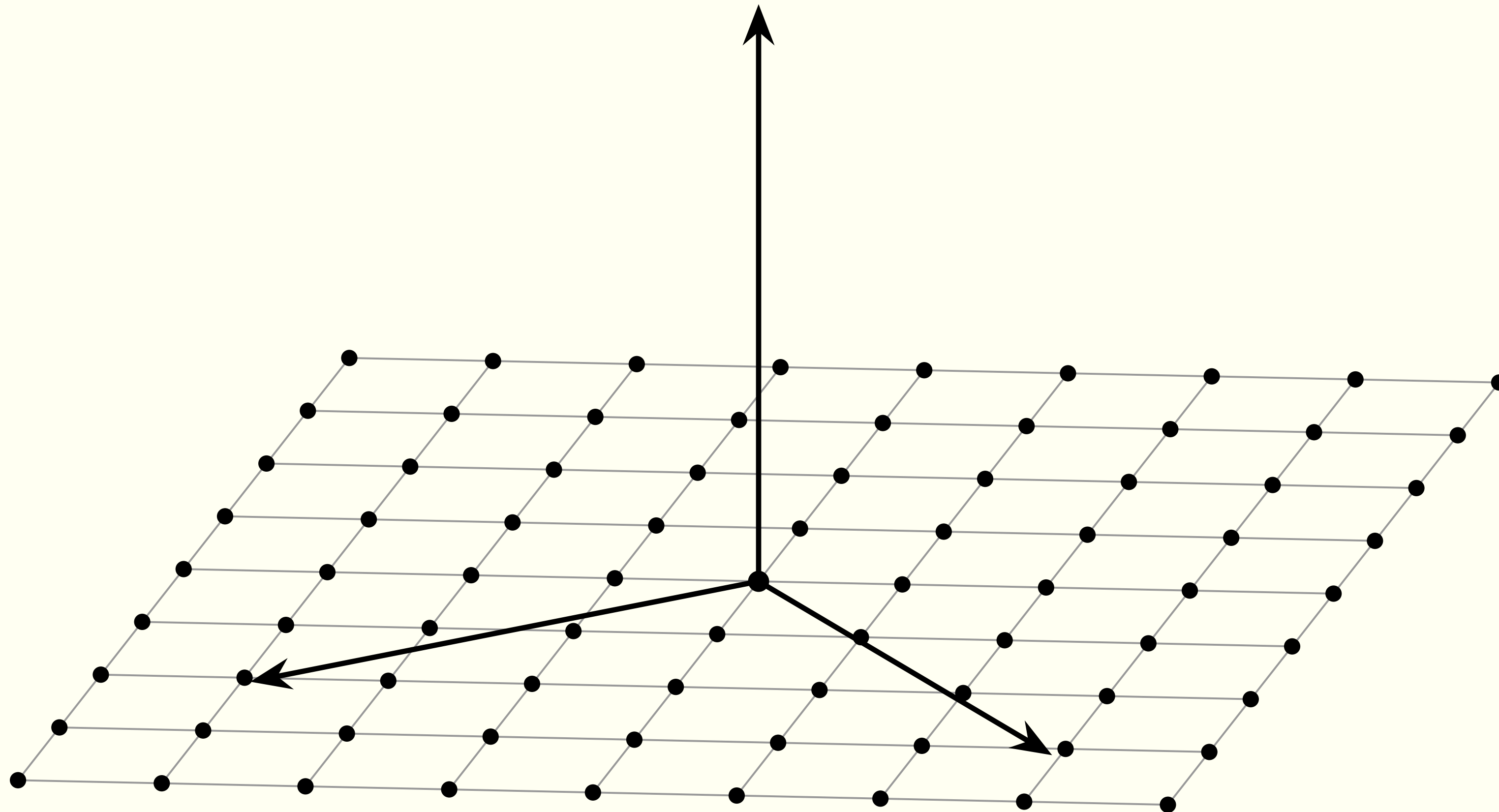
Regev's reduction and the short vector problem

Regev noticed that if you took the Fourier transform of this state, you would get a state with a very interesting form!



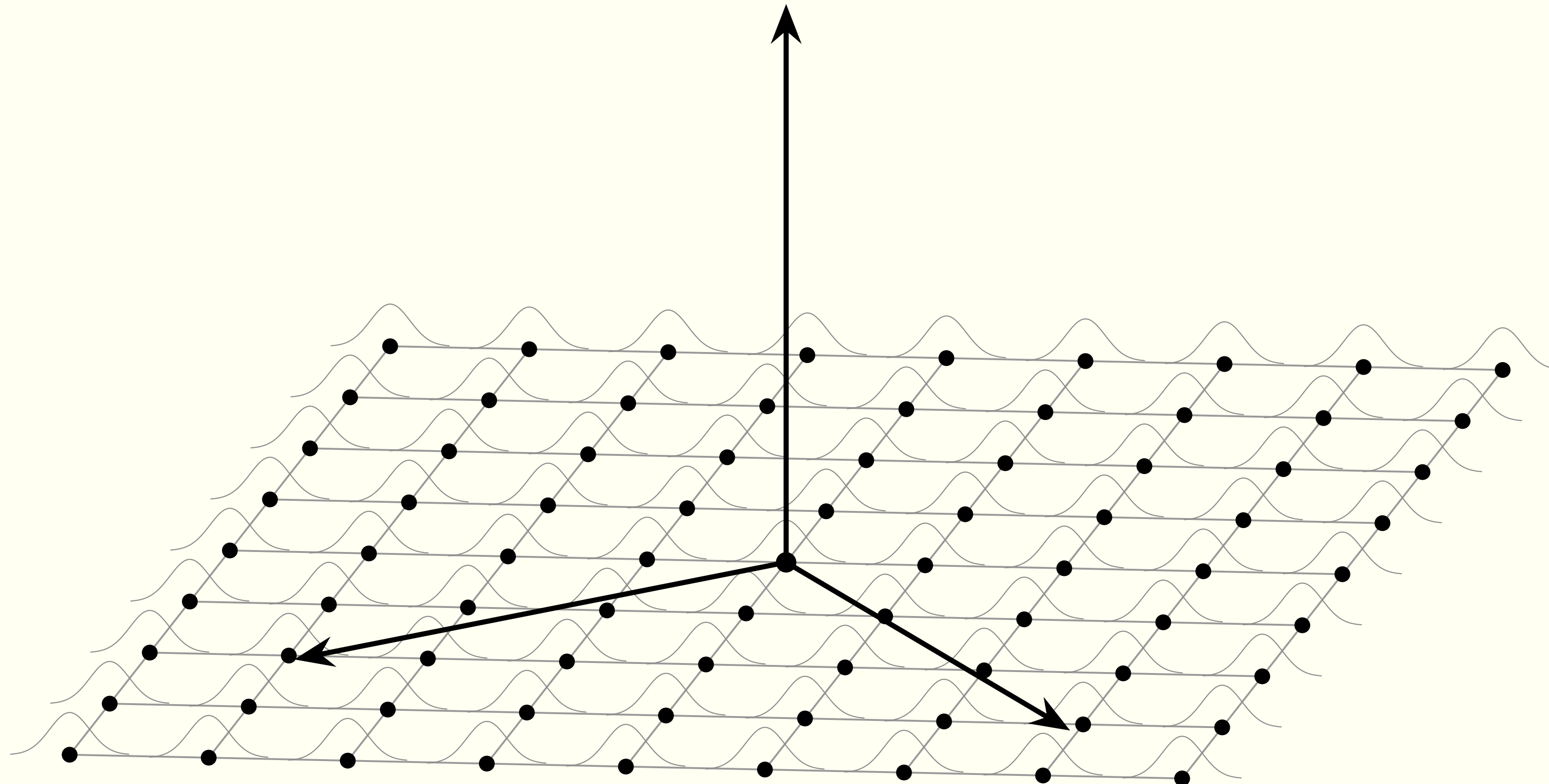
Regev's reduction and the short vector problem

Regev noticed that if you took the Fourier transform of this state, you would get a state with a very interesting form!



Regev's reduction and the short vector problem

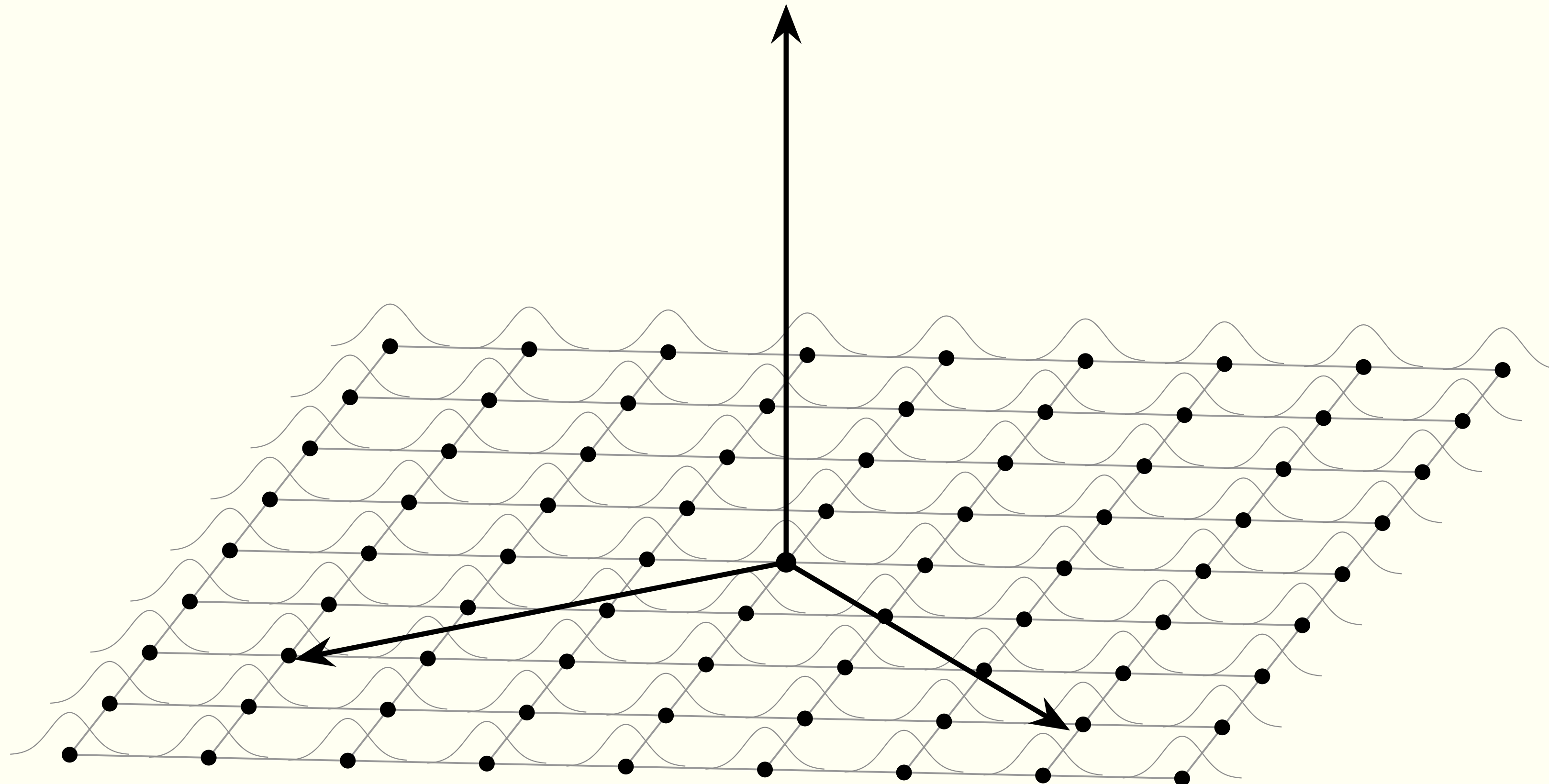
Regev noticed that if you took the Fourier transform of this state, you would get a state with a very interesting form!



Regev's reduction and the short vector problem

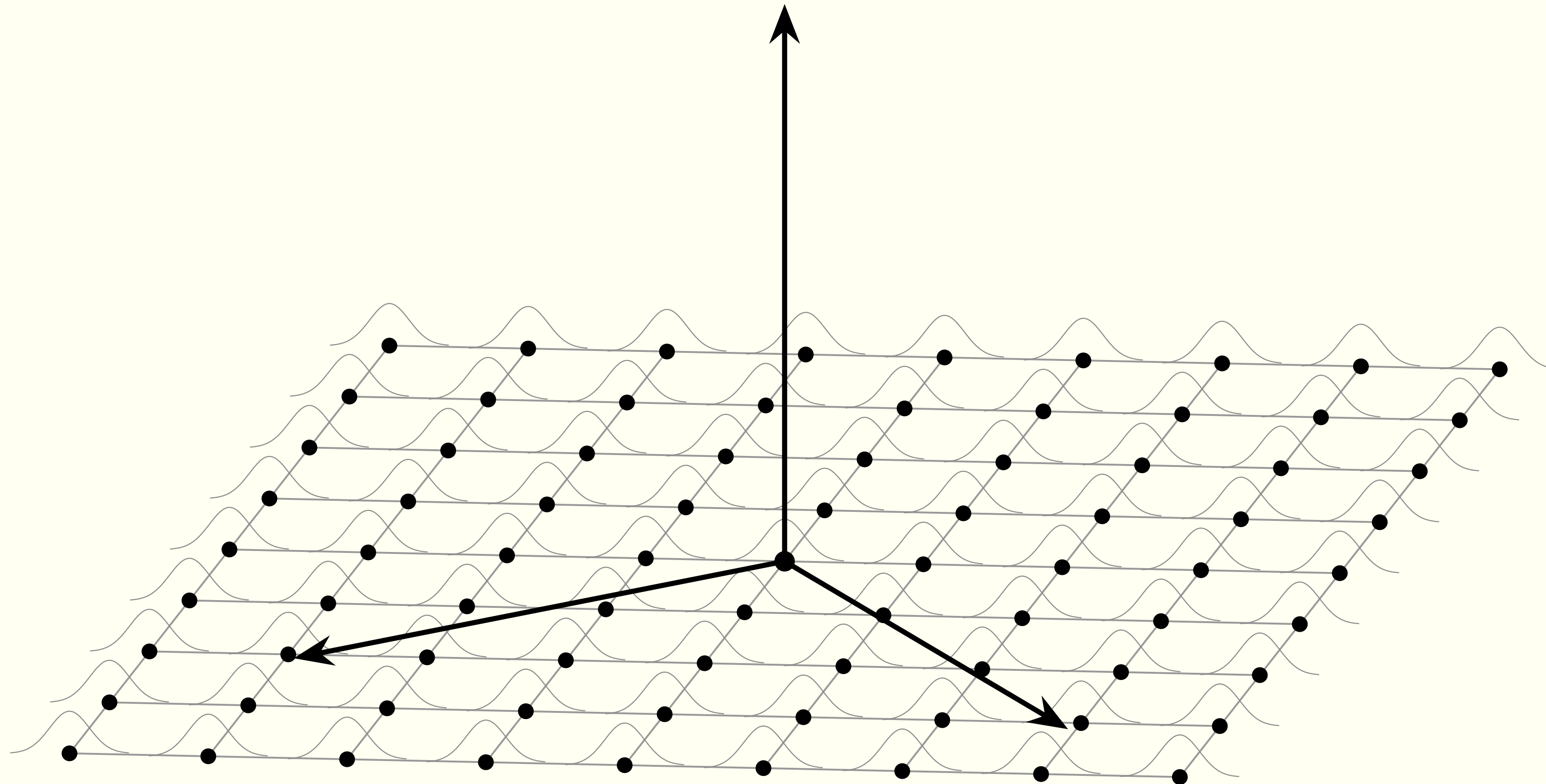
Regev noticed that if you took the Fourier transform of this state, you would get a state with a very interesting form!

→ It looks like the dual lattice, plus a bit of Gaussian noise!



Regev's reduction and the short vector problem

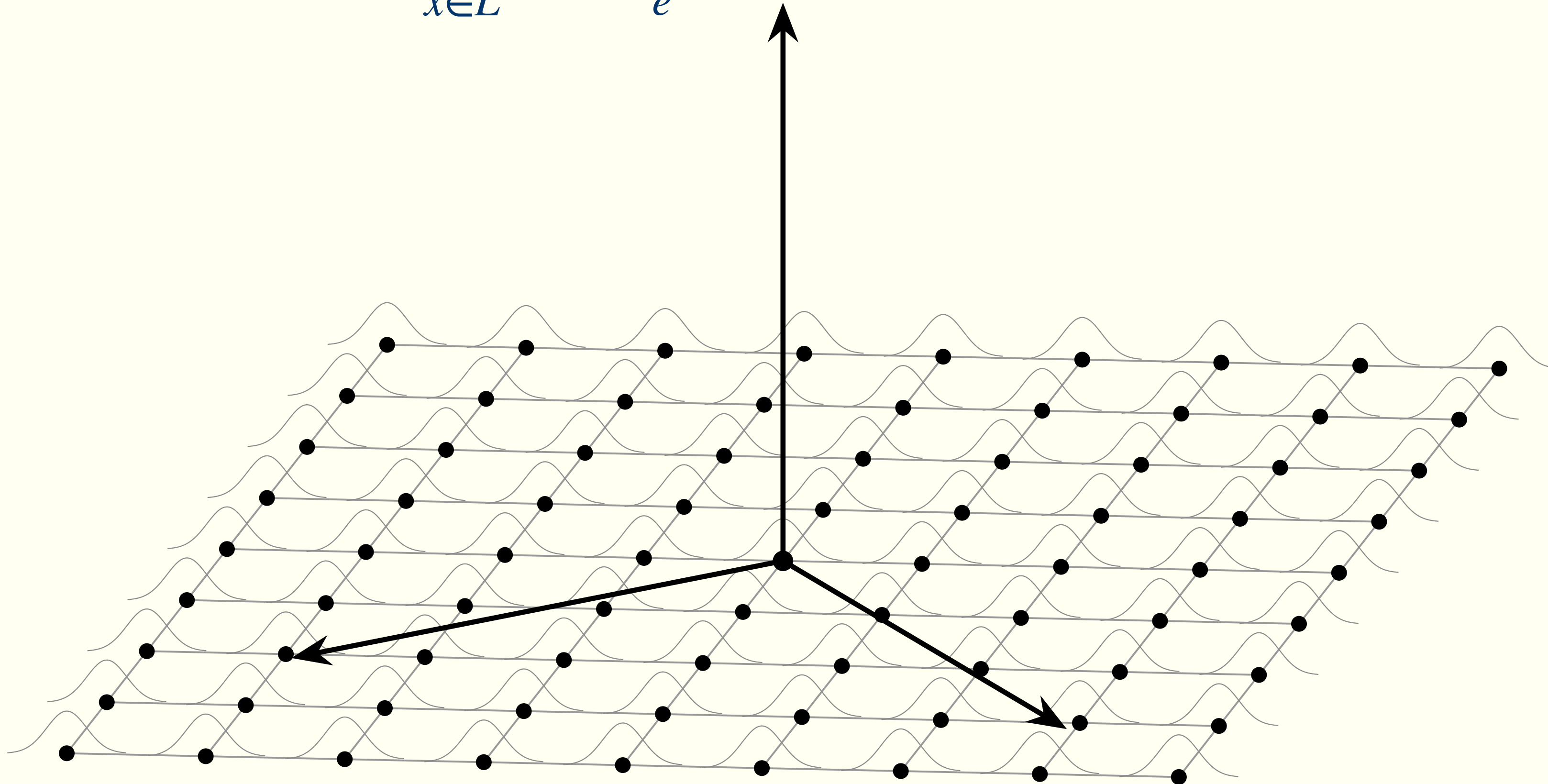
An aside, the next question Regev asked was: How can we prepare this state?



Regev's reduction and the short vector problem

An aside, the next question Regev asked was: How can we prepare this state?

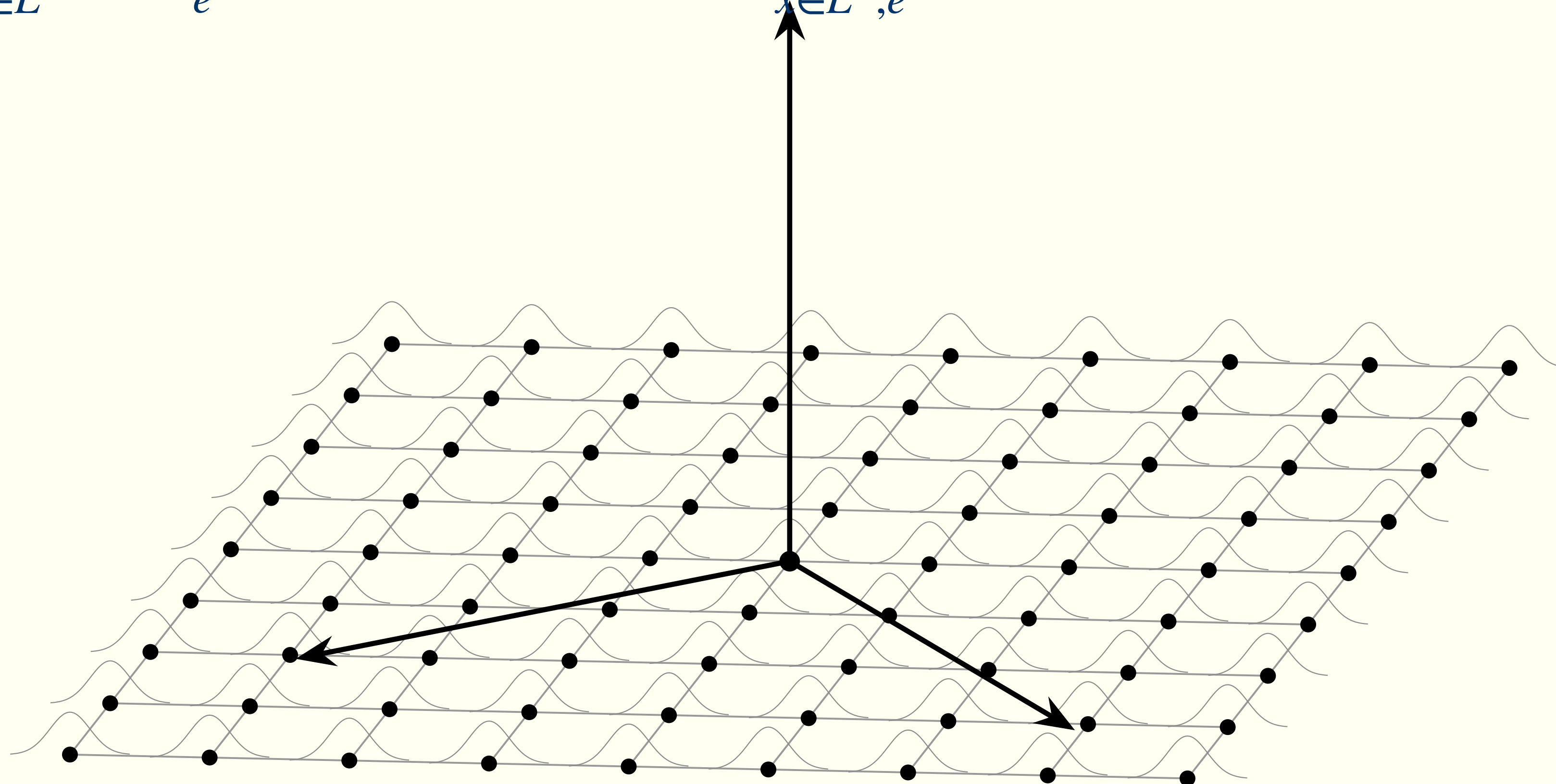
$$\sum_{x \in L^\perp} |x\rangle \sum_e \sqrt{\text{gaussian}(e)} |e\rangle$$



Regev's reduction and the short vector problem

An aside, the next question Regev asked was: How can we prepare this state?

$$\sum_{x \in L^\perp} |x\rangle \sum_e \sqrt{\text{gaussian}(e)} |e\rangle \rightarrow \sum_{x \in L^\perp, e} |x\rangle |e\rangle \sqrt{\text{gaussian}(e)} |x + e\rangle$$

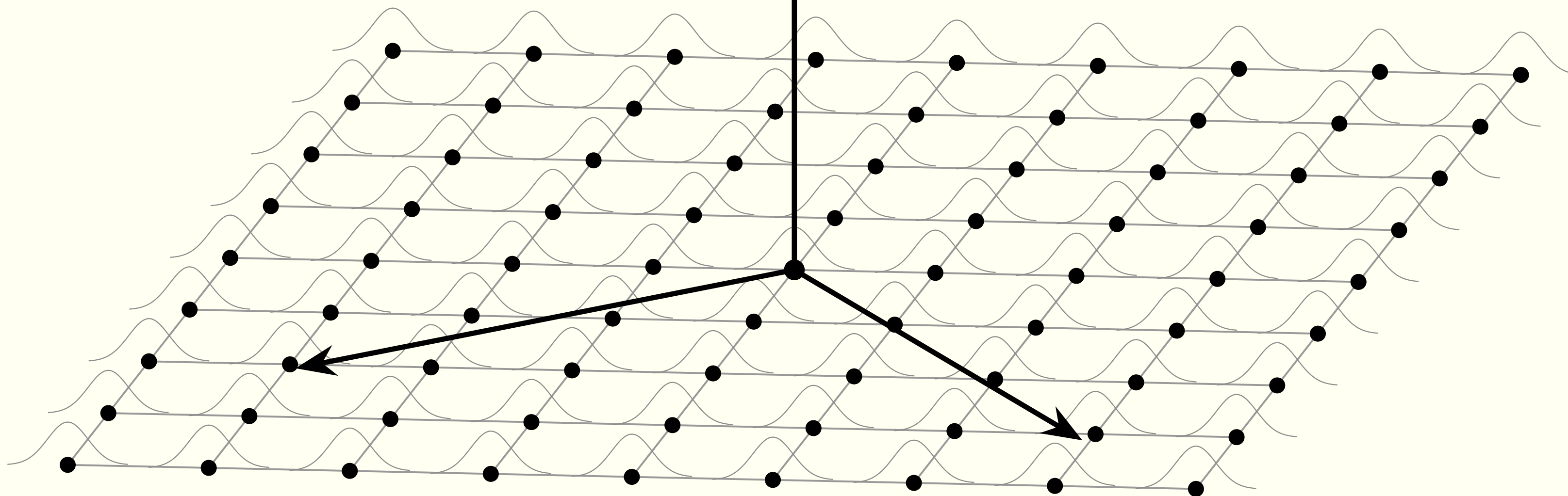


Regev's reduction and the short vector problem

An aside, the next question Regev asked was: How can we prepare this state?

$$\sum_{x \in L^\perp} |x\rangle \sum_e \sqrt{\text{gaussian}(e)} |e\rangle \rightarrow \sum_{x \in L^\perp, e} |x\rangle |e\rangle \sqrt{\text{gaussian}(e)} |x + e\rangle$$

Uncomputing is exactly the LWE problem!



The convolution trick

One way to see why Regev's reduction works without doing the QFT manually is via the convolution trick, which states that given two states $|\psi\rangle, |\phi\rangle$:

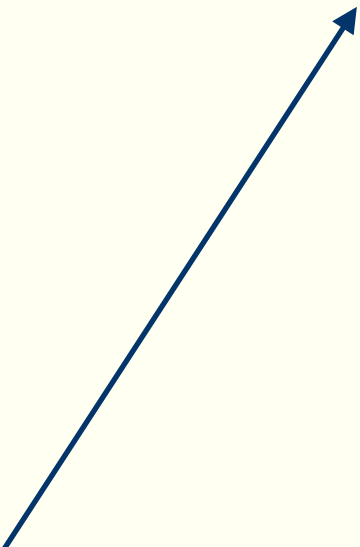
$$|\psi\rangle \odot |\phi\rangle = \text{QFT}^\dagger \left(\text{QFT} |\psi\rangle \star \text{QFT} |\phi\rangle \right)$$

The convolution trick

One way to see why Regev's reduction works without doing the QFT manually is via the convolution trick, which states that given two states $|\psi\rangle, |\phi\rangle$:

$$|\psi\rangle \odot |\phi\rangle = \text{QFT}^\dagger (\text{QFT} |\psi\rangle \star \text{QFT} |\phi\rangle)$$

Element-wise product
(Being in the lattice times being Gaussian)

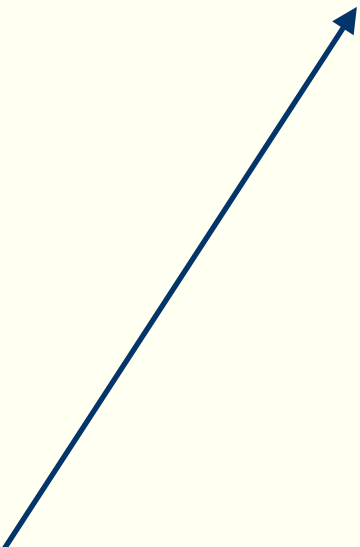


The convolution trick

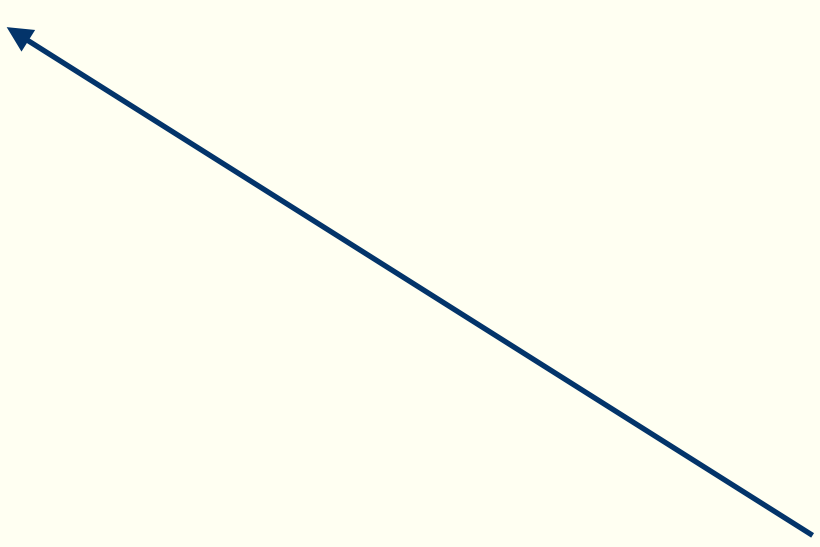
One way to see why Regev's reduction works without doing the QFT manually is via the convolution trick, which states that given two states $|\psi\rangle, |\phi\rangle$:

$$|\psi\rangle \odot |\phi\rangle = \text{QFT}^\dagger (\text{QFT} |\psi\rangle \star \text{QFT} |\phi\rangle)$$

Element-wise product
(Being in the lattice times being Gaussian)

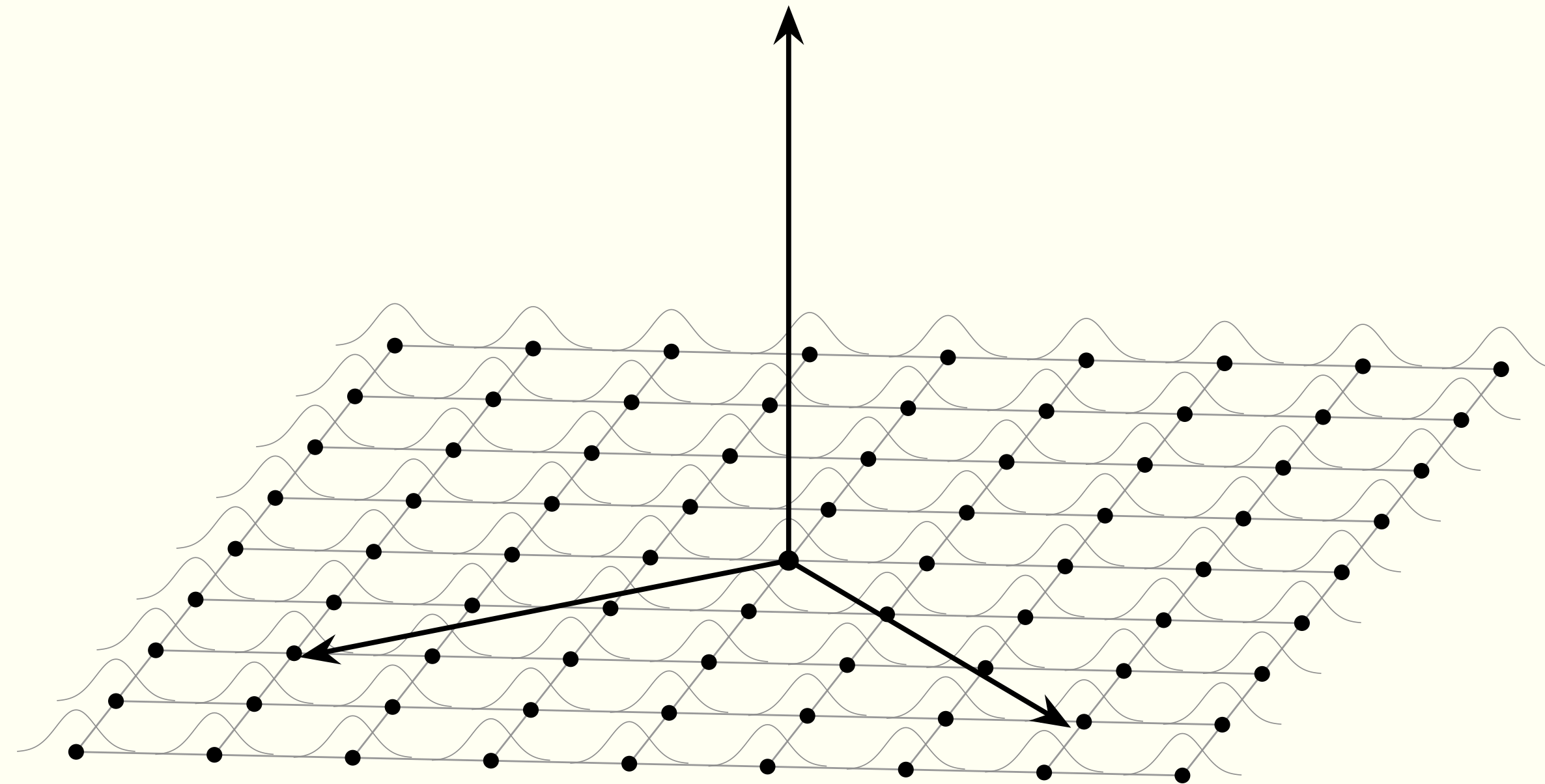


Convolution
(Dual lattice points, convolved with noise)



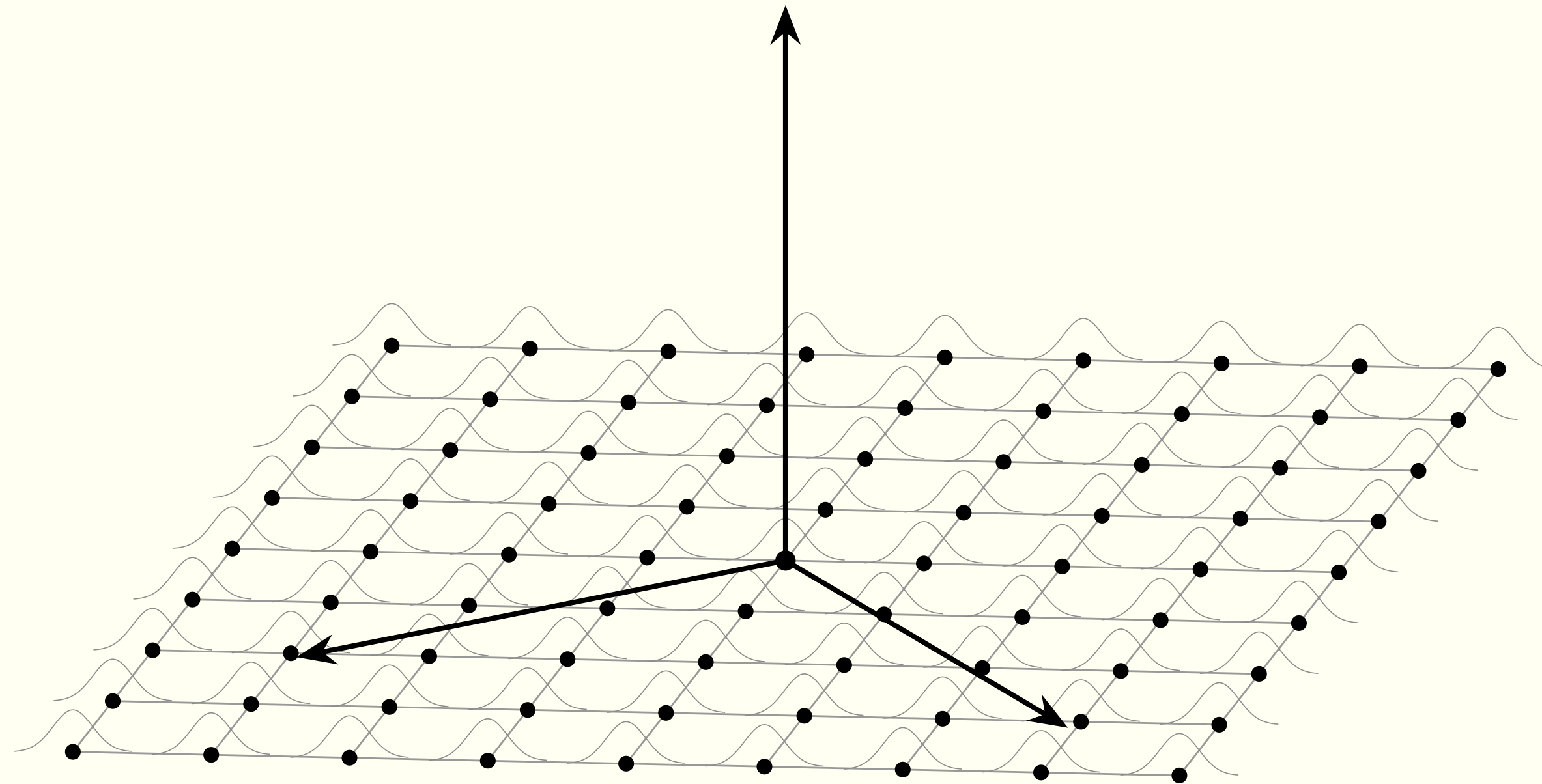
Yamakawa-Zhandry's insight

Before 2022, Yamakawa and Zhandry were wondering about a similar question:
Can quantum algorithms solve problems that classical computers provably can't?



Yamakawa-Zhandry's insight

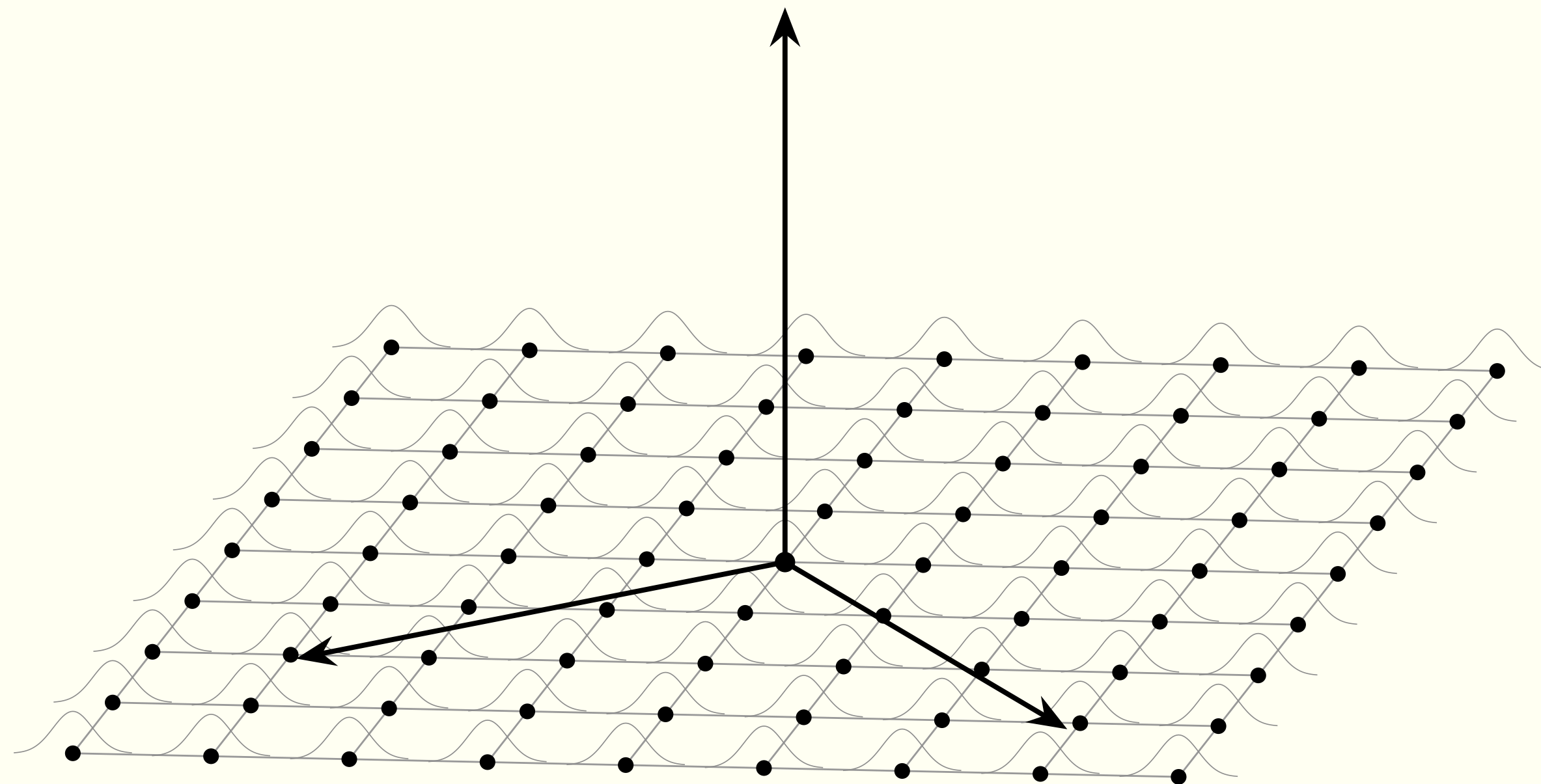
Before 2022, Yamakawa and Zhandry were wondering about a similar question:
Can quantum algorithms solve problems that classical computers provably can't?



They ended up with an algorithm that looks a lot like Regev's reduction, but with a few important modifications.

Yamakawa-Zhandry's insight

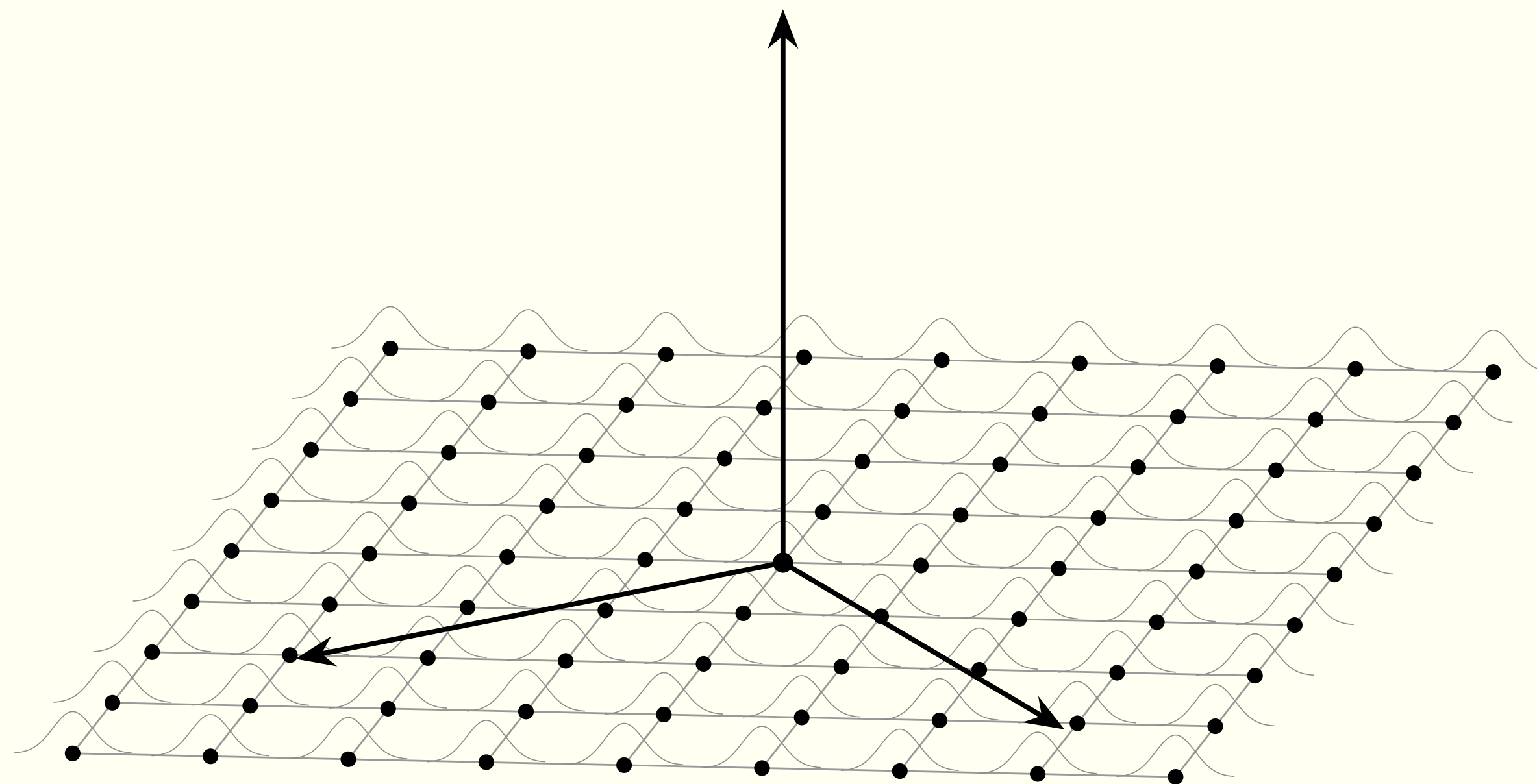
First, let's have the dual lattice corresponds to a good error correcting code.
→ A quantum algorithm can perform Regev's reduction to solve the SVP.



Yamakawa-Zhandry's insight

First, let's have the dual lattice corresponds to a good error correcting code.

→ A quantum algorithm can perform Regev's reduction to solve the SVP, but it's not clear how to get classical hardness, especially unconditionally!

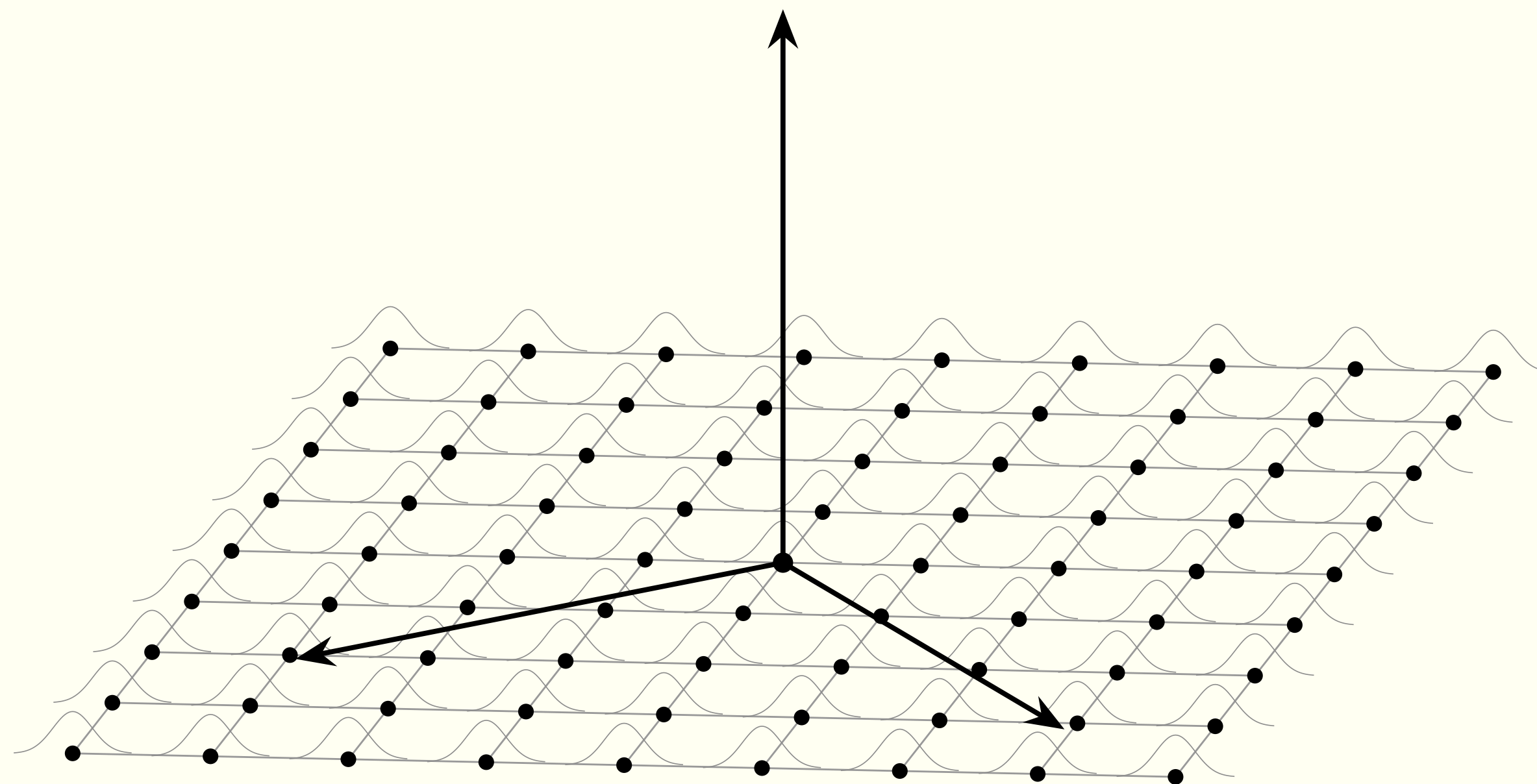


Yamakawa-Zhandry's insight

First, let's have the dual lattice corresponds to a good error correcting code.

→ A quantum algorithm can perform Regev's reduction to solve the SVP, but it's not clear how to get classical hardness, especially unconditionally!

Yamakawa and Zhandry's insight: You have to look at the noise you're adding!



Yamakawa-Zhandry's insight

Now, let's look at what Regev's reduction does:

$$\sum_{x \in L^\perp} |x\rangle \sum_e \sqrt{\text{gaussian}(e)} |e\rangle \rightarrow \sum_{x \in L^\perp, e} \sqrt{\text{gaussian}(e)} |x + e\rangle \rightarrow \text{Fourier transform}$$

Yamakawa-Zhandry's insight

Now, let's look at what Regev's reduction does:

$$\sum_{x \in L^\perp} |x\rangle \sum_e \sqrt{\text{gaussian}(e)} |e\rangle \rightarrow \sum_{x \in L^\perp, e} \sqrt{\text{gaussian}(e)} |x + e\rangle \rightarrow \text{Fourier transform}$$

The noise vector e is something that **only** appears because we took a Fourier transform! So it's something a quantum algorithm will see that a classical algorithm might not get to.

Yamakawa-Zhandry's insight

Now, let's look at what Regev's reduction does:

$$\sum_{x \in L^\perp} |x\rangle \sum_e \sqrt{\text{gaussian}(e)} |e\rangle \rightarrow \sum_{x \in L^\perp, e} \sqrt{\text{gaussian}(e)} |x + e\rangle \rightarrow \text{Fourier transform}$$

We need to find a good choice for the noise vector e so that it satisfies:

(a) Low weight (so we can decode it).

Yamakawa-Zhandry's insight

Now, let's look at what Regev's reduction does:

$$\sum_{x \in L^\perp} |x\rangle \sum_e \sqrt{\text{gaussian}(e)} |e\rangle \rightarrow \sum_{x \in L^\perp, e} \sqrt{\text{gaussian}(e)} |x + e\rangle \rightarrow \text{Fourier transform}$$

We need to find a good choice for the noise vector e so that it satisfies:

- (a) Low weight (so we can decode it).
- (b) Fourier transform corresponds to something hard to solve classically.

Yamakawa-Zhandry's insight

Take a bunch of random functions $f_1, \dots, f_n$ and let $|f_i^{-1}(1)\rangle$ be the uniform superposition over pre-images of 1. Because they're random, the following holds:

$$\langle 0 | H^{\otimes n} | f_i^{-1}(1) \rangle \approx \frac{1}{2}$$

Yamakawa-Zhandry's insight

Take a bunch of random functions $f_1, \dots, f_n$ and let $|f_i^{-1}(1)\rangle$ be the uniform superposition over pre-images of 1. Because they're random, the following holds:

$$\langle 0 | H^{\otimes n} | f_i^{-1}(1) \rangle \approx \frac{1}{2}$$

→ The Fourier transform of the (tensor product of the) states mostly supported on low-weight string, so Regev's reduction will work.

Yamakawa-Zhandry's insight

Take a bunch of random functions $f_1, \dots, f_n$ and let $|f_i^{-1}(1)\rangle$ be the uniform superposition over pre-images of 1. Because they're random, the following holds:

$$\langle 0 | H^{\otimes n} | f_i^{-1}(1) \rangle \approx \frac{1}{2}$$

→ The Fourier transform of the (tensor product of the) states mostly supported on low-weight string, so Regev's reduction will work.

→ After doing the inverse Fourier transform, this problem corresponds to the problem of outputting a point $(x_1, \dots, x_n)$ in the lattice that also satisfies $f_i(x_i) = 1$ for all i . Maybe this is a hard problem classically?

The code intersection problem

Input: A description of an error correcting code $C \subseteq \mathbb{F}_q^n$, and oracle access to functions $f_1, \dots, f_n : \mathbb{F}_q \mapsto \{0,1\}$, and a hash value $x \in \{0,1\}^n$.

Output: A codeword $\mathbf{v} \in C$ such that $f_i(\mathbf{v}_i) = x_i$.

The code intersection problem

Input: A description of an error correcting code $C \subseteq \mathbb{F}_q^n$, and oracle access to functions $f_1, \dots, f_n : \mathbb{F}_q \mapsto \{0,1\}$, and a hash value $x \in \{0,1\}^n$.

Output: A codeword $\mathbf{v} \in C$ such that $f_i(\mathbf{v}_i) = x_i$.

The quantum algorithm:

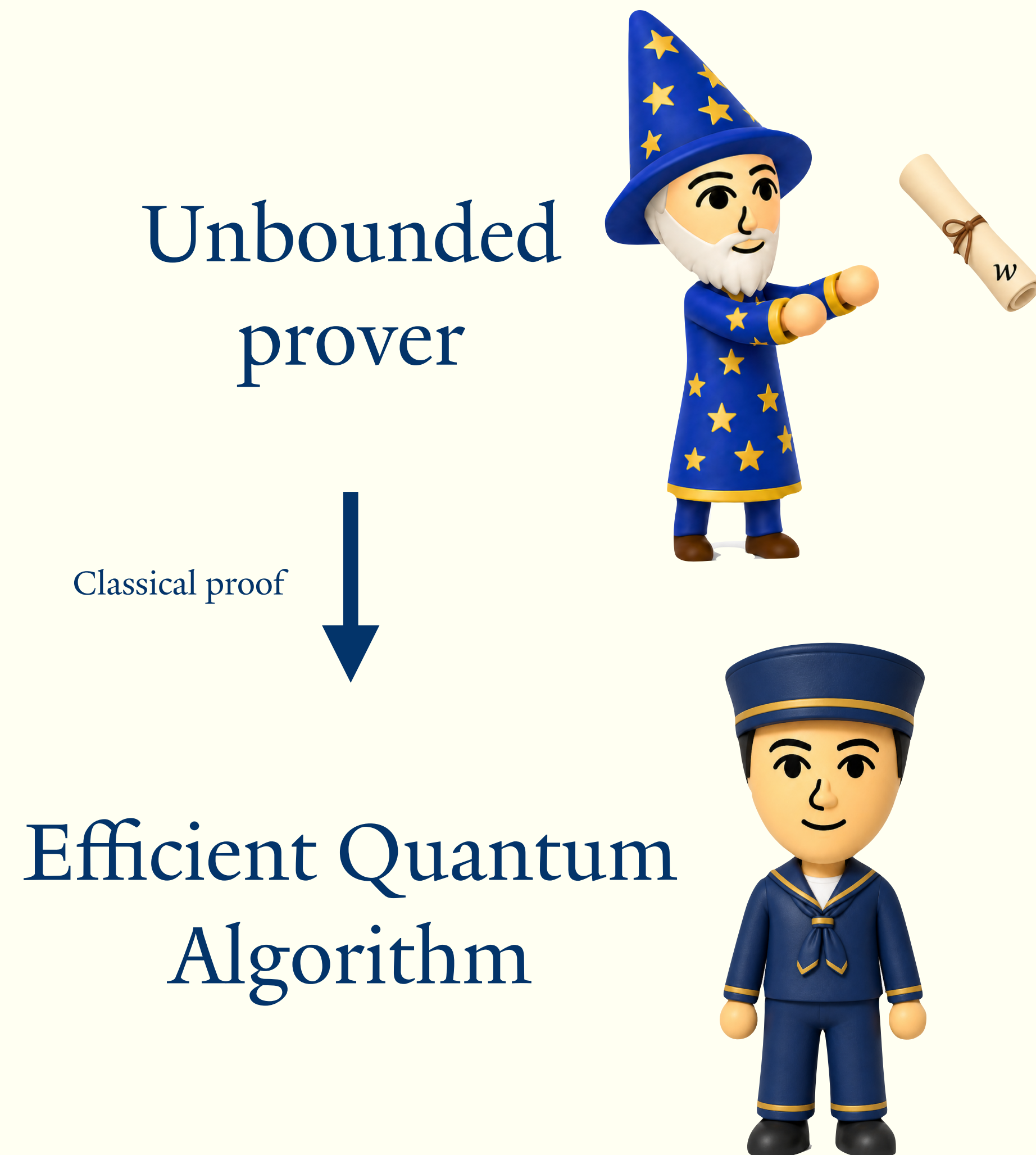
1. Measure each of the functions on the uniform superposition until you see outcome x_i .
2. Perform Regev's reduction using this state (and uniform superposition over the codewords).
3. Measure the resulting state.

Classical versus quantum proofs and advice

Another way we might want to study quantum computation is through the lens of “proofs”

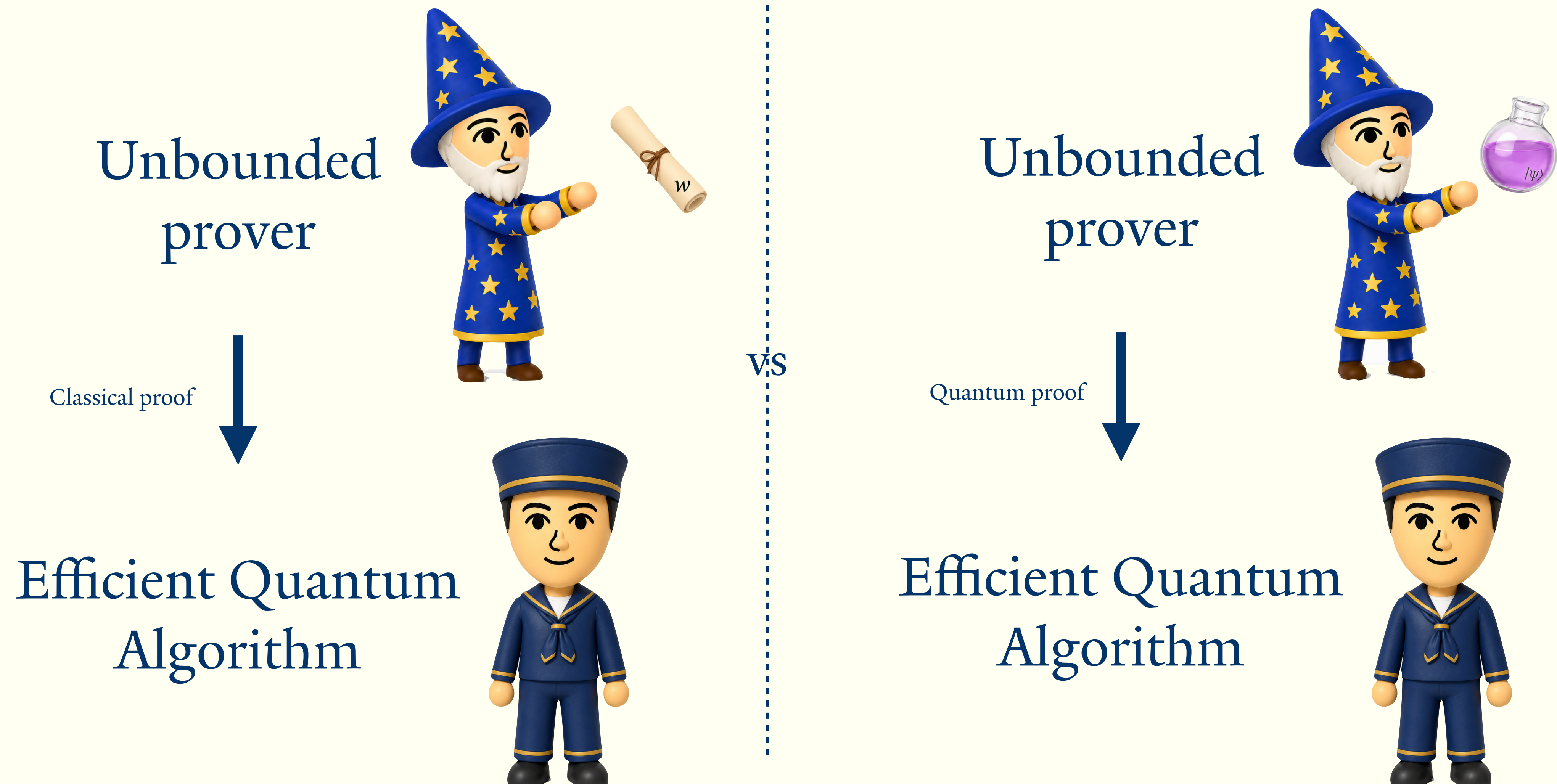
Classical versus quantum proofs and advice

Another way we might want to study quantum computation is through the lens of “proofs”



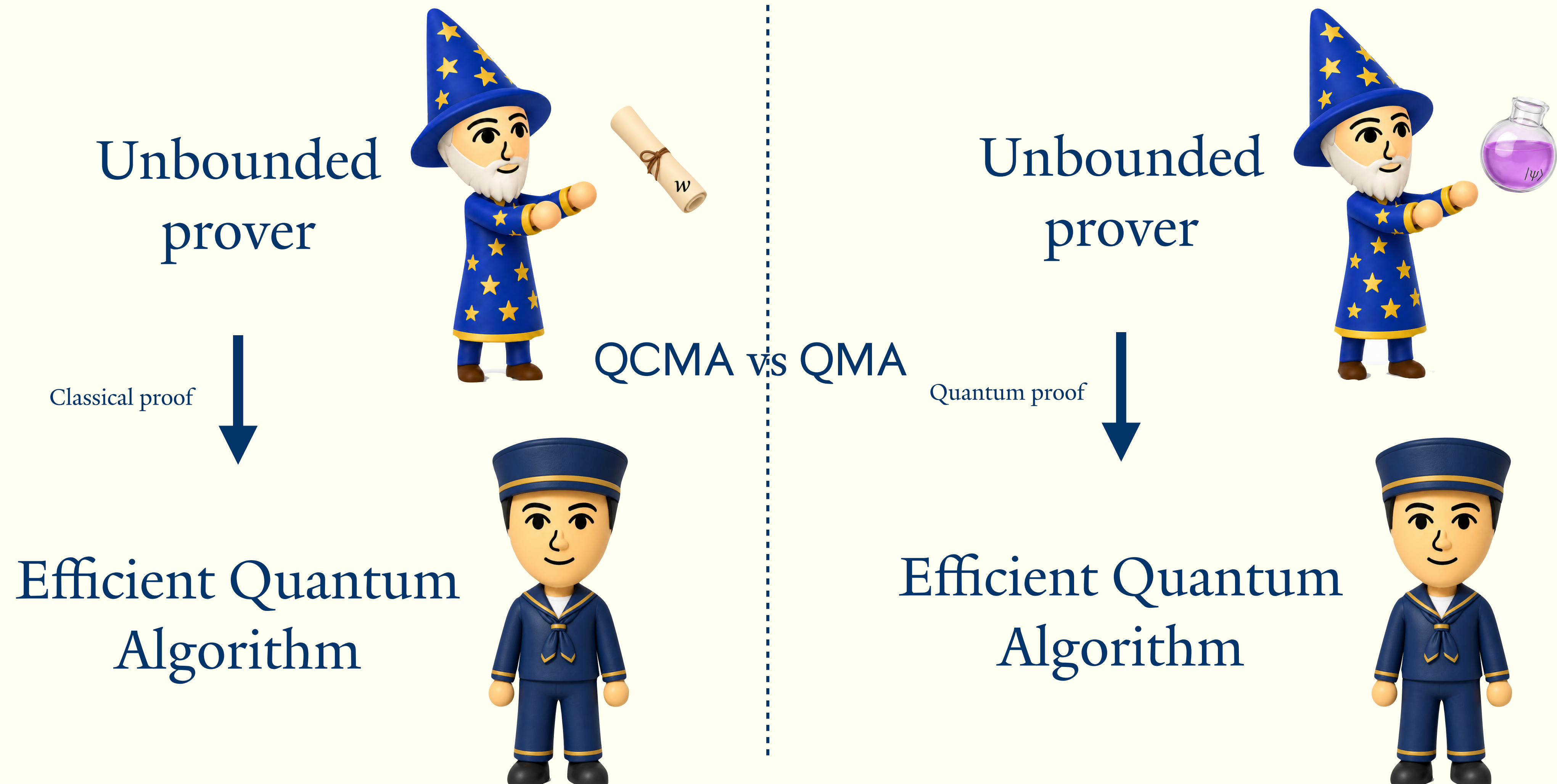
Classical versus quantum proofs and advice

Another way we might want to study quantum computation is through the lens of “proofs”



Classical versus quantum proofs and advice

Another way we might want to study quantum computation is through the lens of “proofs”



Classical versus quantum proofs and advice

To compare these classes, we have to slightly modify the code intersection problem.

Classical versus quantum proofs and advice

To compare these classes, we have to slightly modify the code intersection problem.

Lets define a signed oracle $O_{S,f_1,\dots,f_n}$ as follows:

Classical versus quantum proofs and advice

To compare these classes, we have to slightly modify the code intersection problem.

Lets define a signed oracle $O_{S,f_1,\dots,f_n}$ as follows:

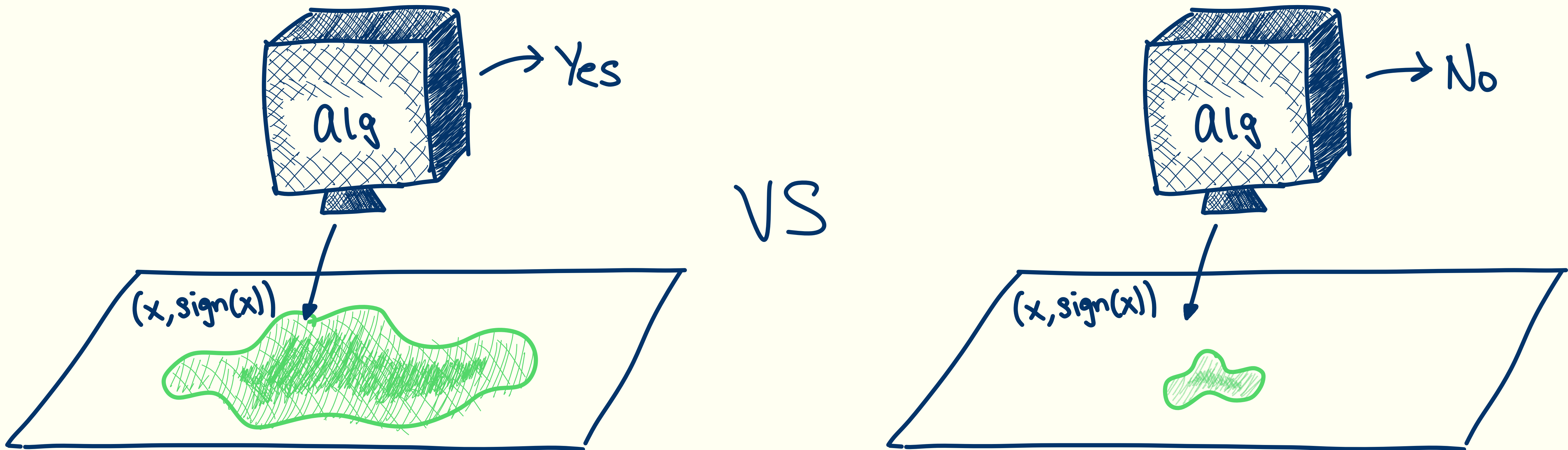
Fix a linear code $C \subseteq \mathbb{F}_q^n$, and functions $f_1, \dots, f_n : \mathbb{F}_q \mapsto \{0,1\}$, and $S \subseteq \{0,1\}^n \times \mathbb{F}_q^n$. Then the oracle $O_{S,f_1,\dots,f_n}$ outputs 1 on $(x, \mathbf{v})$ if and only if:

- (1) $\mathbf{v} \in C$
- (2) $f_i(\mathbf{v}_i) = x_i$ for all i .
- (3) $(x, \mathbf{v}) \in C$

Classical versus quantum proofs and advice

Input: Oracle access to a signed set oracle $O_{S,f_1,\dots,f_n}$ (for a fixed code C).

Output: Is $|O_{S,f_1,\dots,f_n}| \geq 2^n$ or $|O_{S,f_1,\dots,f_n}| \leq 2^n/3$?



Classical versus quantum proofs and advice

The Yamakawa-Zhandry algorithm gives us a way to solve this problem, given the pre-images of the individual hash functions,

$$\bigotimes_{i=1}^n |f_i^{-1}(0)\rangle \otimes |f_i^{-1}(1)\rangle$$

Classical versus quantum proofs and advice

The Yamakawa-Zhandry algorithm gives us a way to solve this problem, given the pre-images of the individual hash functions,

$$\bigotimes_{i=1}^n |f_i^{-1}(0)\rangle \otimes |f_i^{-1}(1)\rangle$$

Let's assume for the sake of contradiction that there was a verifier with classical proof that also helped solve this problem.

Classical versus quantum proofs and advice

Let $A(w)$ be the verifier that solves the code intersection problem, taking a classical witness. Consider the following algorithm:

Classical versus quantum proofs and advice

Let $A(w)$ be the verifier that solves the code intersection problem, taking a classical witness. Consider the following algorithm:

Guess a witness w , let $\Delta \leftarrow \emptyset$. For i from 1 to $2^{n/3}$:

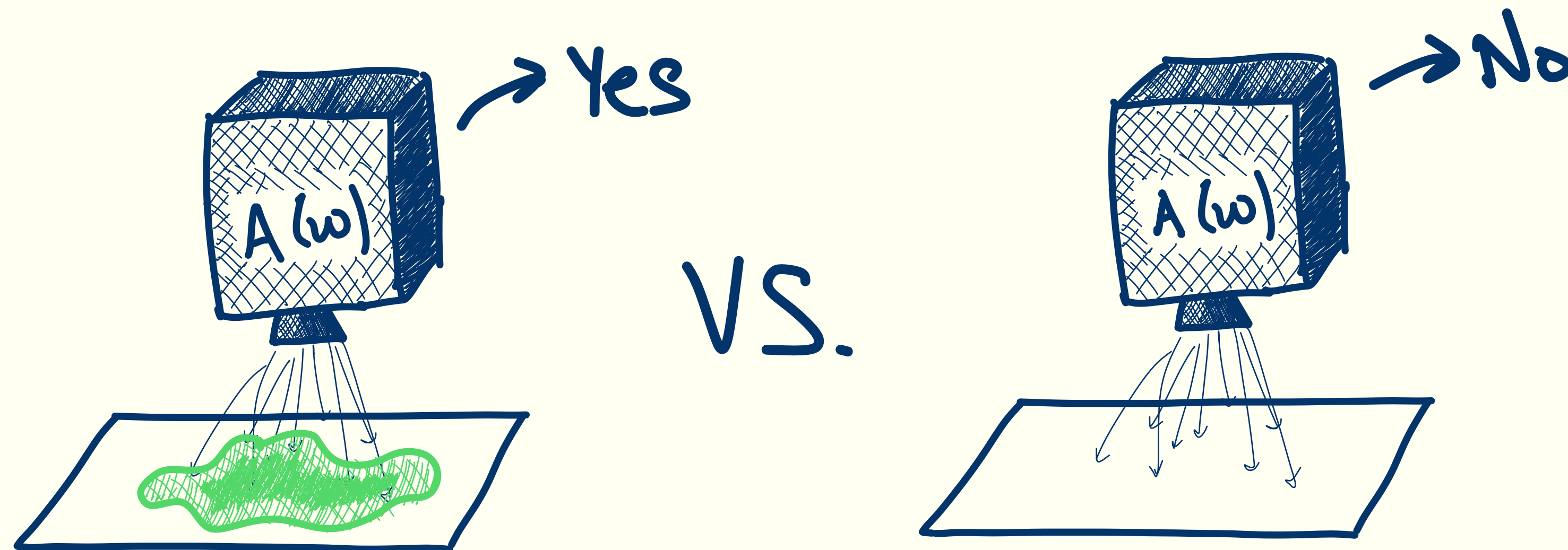
- (a) Run $A^\Delta(w)$, pick a random query, and measure it to get $(x_i, \mathbf{v}_i)$.
- (b) Set $\Delta \leftarrow \Delta \cup \{(x_i, \mathbf{v}_i)\}$

Output the set Δ .

From distinguishers to samplers

Assume for the sake of contradiction that some algorithm $A(w)$ distinguishes between the two cases. So, $A(w)$ outputs yes when given $O_{S,f_1,\dots,f_n}$, and no when given $\emptyset$.

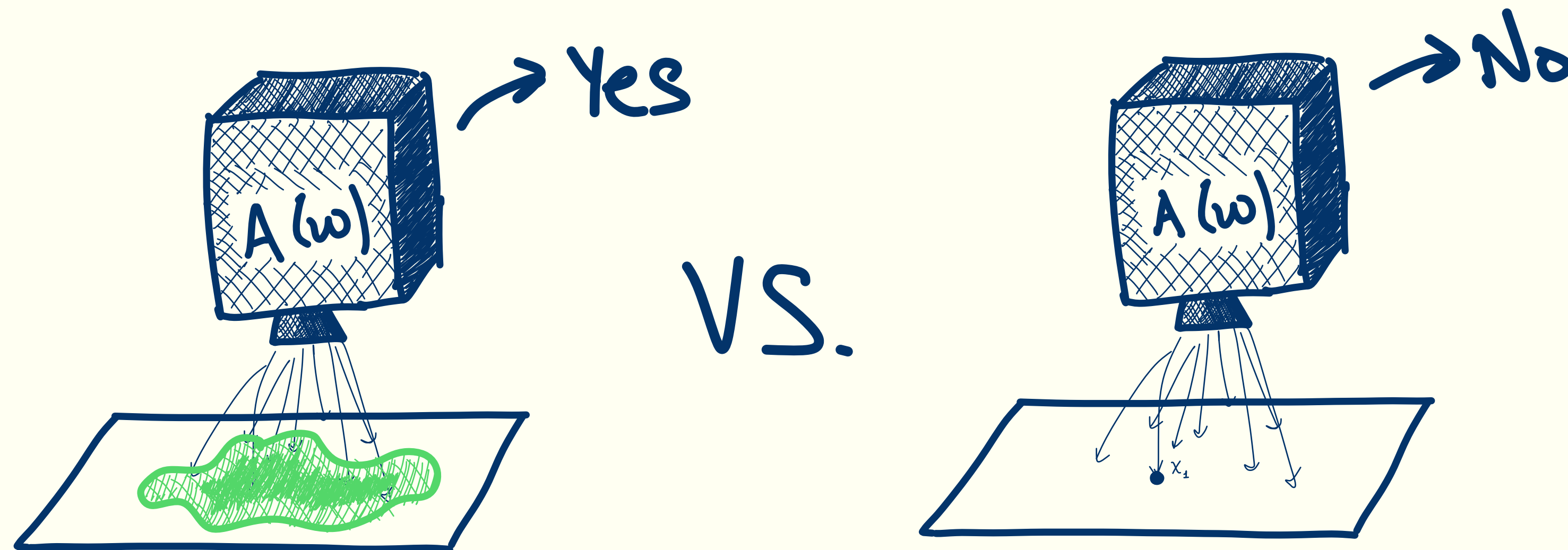
→ A random query of $A(w)$ must query a point of $O_{S,f_1,\dots,f_n}$ with at least $1/36t^2$ probability



From distinguishers to samplers

Assume for the sake of contradiction that some algorithm $A(w)$ distinguishes between the two cases. So, $A(w)$ outputs yes when given $O_{S,f_1,\dots,f_n}$, and no when given $\emptyset$.

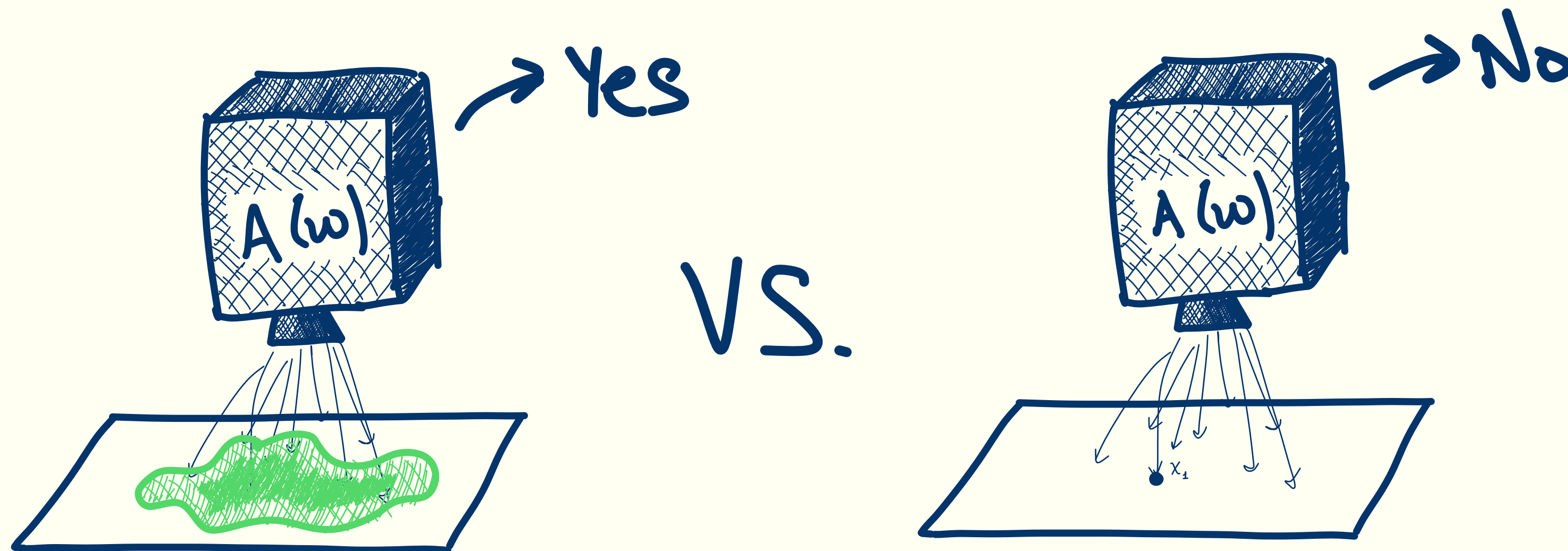
→ A random query of $A(w)$ must query a point of $O_{S,f_1,\dots,f_n}$ with at least $1/36t^2$ probability



From distinguishers to samplers

Assume for the sake of contradiction that some algorithm $A(w)$ distinguishes between the two cases. So, $A(w)$ outputs yes when given $O_{S,f_1,\dots,f_n}$, and no when given $\{(x_1, \mathbf{v}_1)\}$.

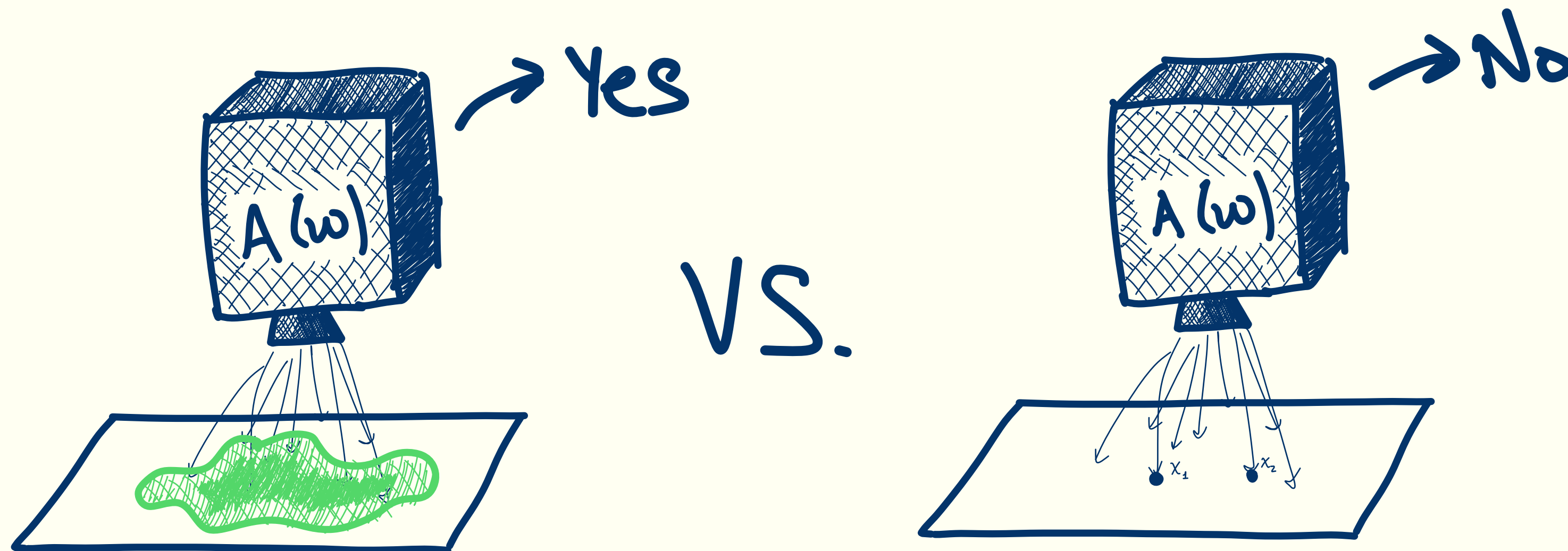
→ A random query of $A(w)$ must query a point of $O_{S,f_1,\dots,f_n} \setminus \{(x_1, \mathbf{v}_1)\}$ with at least $1/36t^2$ probability



From distinguishers to samplers

Assume for the sake of contradiction that some algorithm $A(w)$ distinguishes between the two cases. So, $A(w)$ outputs yes when given $O_{S,f_1,\dots,f_n}$, and no when given $\{(x_1, \mathbf{v}_1)\}$.

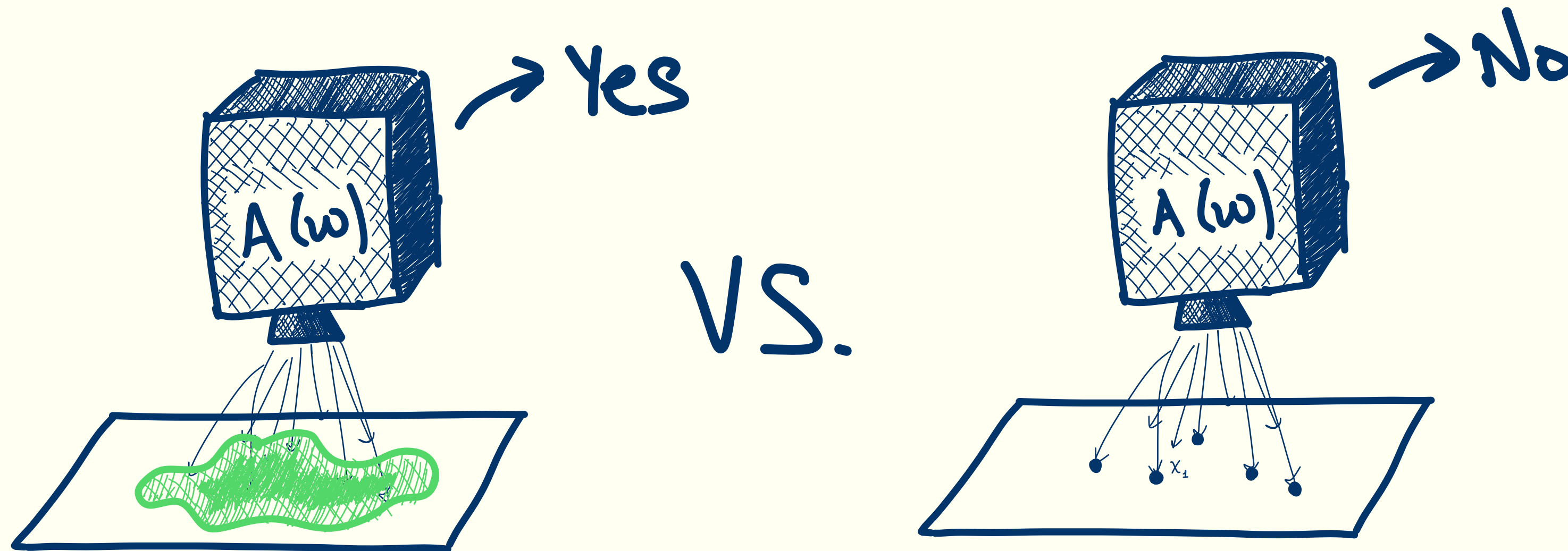
→ A random query of $A(w)$ must query a point of $O_{S,f_1,\dots,f_n} \setminus \{(x_1, \mathbf{v}_1)\}$ with at least $1/36t^2$ probability



From distinguishers to samplers

We can keep repeating this to get $2^n/3$ points from $O_{S,f_1,\dots,f_n}$, with probability at least $(1/36t^2)^{2^{n/3}}$. If we additionally guess the witness, we can sample $2^n/3$ points with probability

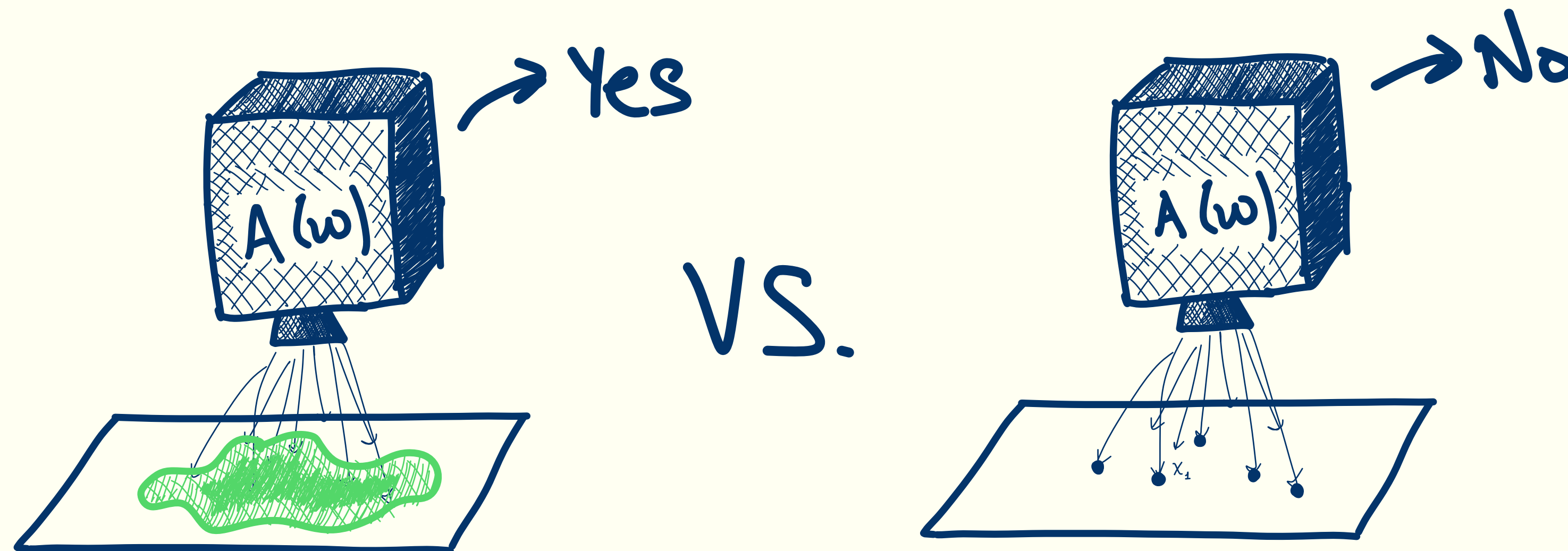
$$\geq 2^{-|w|} (1/36t^2)^{2^{n/3}}$$



From distinguishers to samplers

We can keep repeating this to get $2^n/3$ points from $O_{S,f_1,\dots,f_n}$, with probability at least $(1/36t^2)^{2^{n/3}}$. If we additionally guess the witness, we can sample $2^n/3$ points with probability

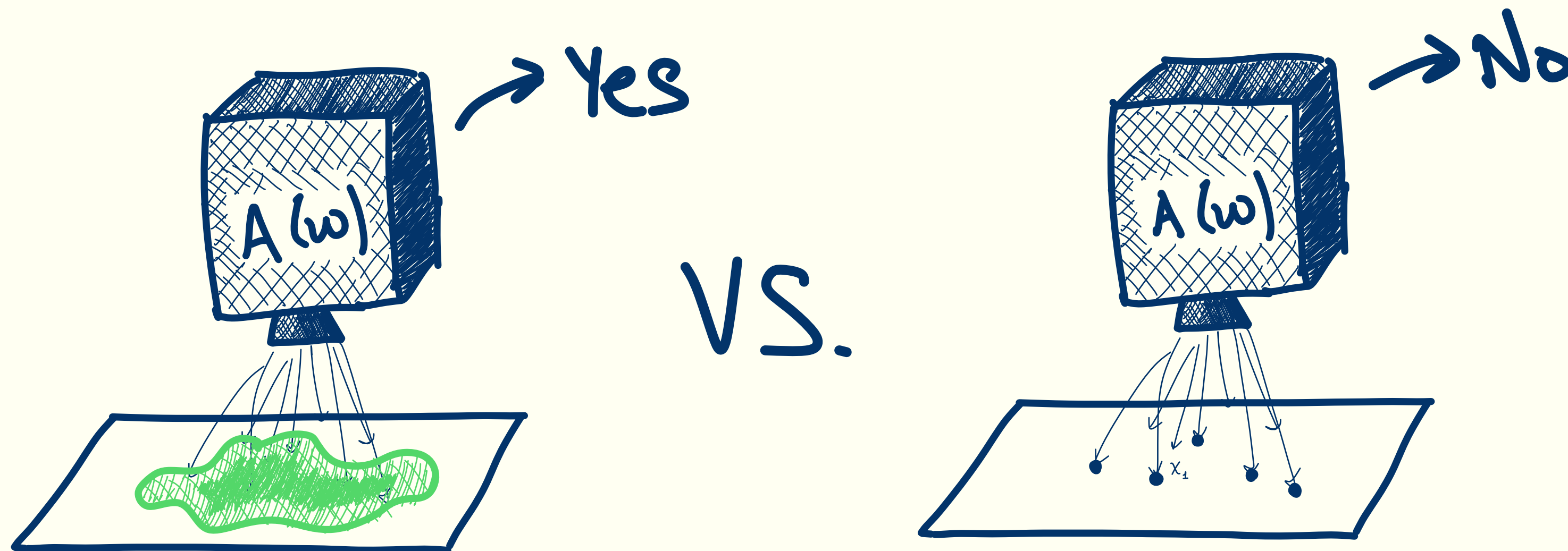
$$\geq 2^{-|w|} (1/36t^2)^{2^{n/3}}$$



Note that our sampler doesn't make any queries to S !

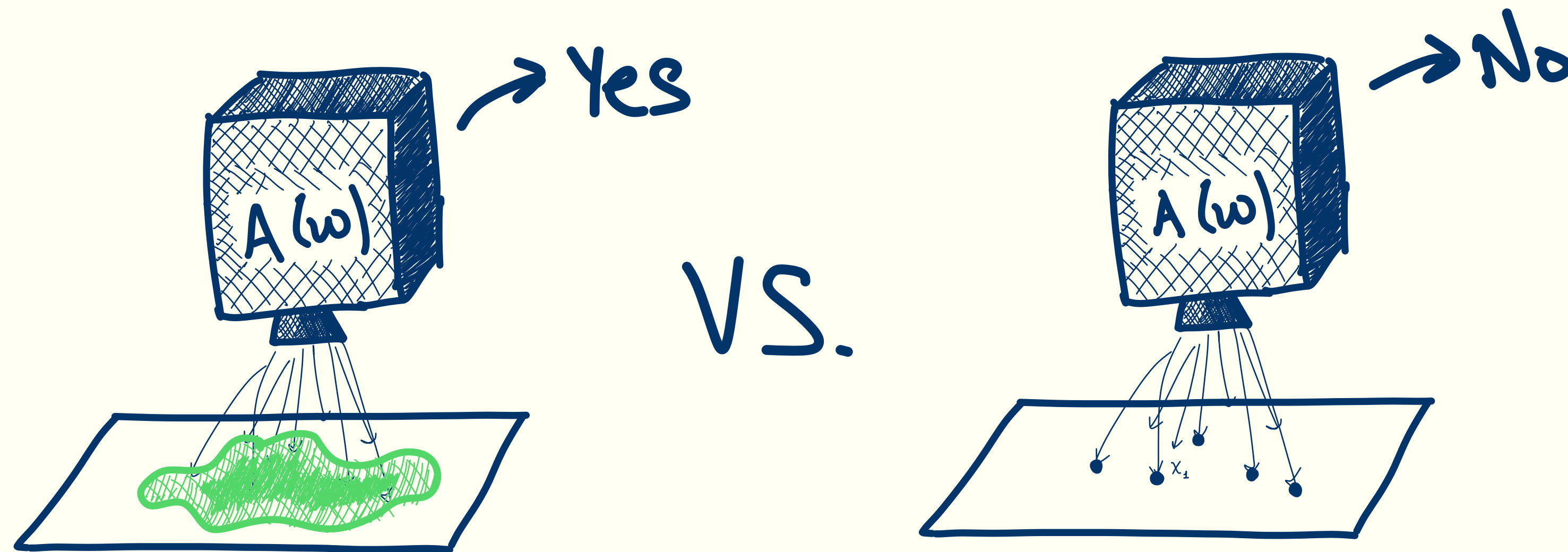
Ruling out classical proofs

If there is a QCMA verifier for set size verification, we get a sampler, making no queries to $O_{S,f_1,\dots,f_n}$, that samples from any set $O_{S,f_1,\dots,f_n}$ with probability $\geq 2^{-|w|} (1/36t^2)^{2^{n/3}}$



Ruling out classical proofs

If there is a QCMA verifier for set size verification, we get a sampler, making no queries to $O_{S,f_1,\dots,f_n}$, that samples from any set $O_{S,f_1,\dots,f_n}$ with probability $\geq 2^{-|w|} (1/36t^2)^{2^{n/3}}$



Let's think about what list recovery means for this collection of $2^n/3$ codewords!

Ruling out classical proofs

List recovery (informally): A code is (ℓ, L) -list recoverable if whenever you show me a collection of L codewords of a code, I can find $n\ell$ distinct symbols.

Ruling out classical proofs

List recovery (informally): A code is (ℓ, L) -list recoverable if whenever you show me a collection of L codewords of a code, I can find $n\ell$ distinct symbols.

Say that I started with a code that was $(\ell, 2\ell)$ -list recoverable, then this would mean
→ If you showed me $2^n/3$ pairs, $(x_i, \mathbf{v}_i)$, then the collection of $\mathbf{v}_i$'s must consist of at least $(n/2) \cdot 2^n/3$ many distinct symbols, and you guessed their hashes correctly.

Ruling out classical proofs

List recovery (informally): A code is (ℓ, L) -list recoverable if whenever you show me a collection of L codewords of a code, I can find $n\ell$ distinct symbols.

Say that I started with a code that was $(\ell, 2\ell)$ -list recoverable, then this would mean
→ If you showed me $2^n/3$ pairs, $(x_i, \mathbf{v}_i)$, then the collection of $\mathbf{v}_i$'s must consist of at least $(n/2) \cdot 2^n/3$ many distinct symbols, and you guessed their hashes correctly.

But any algorithm that guesses the hash values of $(n/2) \cdot 2^n/3$ many symbols, without looking at the oracle, must have success probability at most

$$\left(2^{-(n/2)}\right)^{2^n/3} \ll 2^{-|w|} \left(1/36t^2\right)^{2^n/3}$$

Ruling out classical proofs

List recovery (informally): A code is (ℓ, L) -list recoverable if whenever you show me a collection of L codewords of a code, I can find $n\ell$ distinct symbols.

Say that I started with a code that was $(\ell, 2\ell)$ -list recoverable, then this would mean
→ If you showed me $2^n/3$ pairs, $(x_i, \mathbf{v}_i)$, then the collection of $\mathbf{v}_i$'s must consist of at least $(n/2) \cdot 2^n/3$ many distinct symbols, and you guessed their hashes correctly.

But any algorithm that guesses the hash values of $(n/2) \cdot 2^n/3$ many symbols, without looking at the oracle, must have success probability at most

$$\left(2^{-(n/2)}\right)^{2^n/3} \ll 2^{-|w|} \left(1/36t^2\right)^{2^n/3}$$

Contradiction!

Ruling out classical proofs

If we think about the recipe we need to make this work, we need to find a code that has

Ruling out classical proofs

If we think about the recipe we need to make this work, we need to find a code that has

(a) the dual code has good distance and
(b) can be efficiently decoded

} For the Yamakawa-Zhandry algorithm to work

Ruling out classical proofs

If we think about the recipe we need to make this work, we need to find a code that has

- (a) the dual code has good distance and
 - (b) can be efficiently decoded
 - (c) nearly optimal list recovery
- } For the Yamakawa-Zhandry algorithm to work
- To get soundness against classical proofs

Ruling out classical proofs

If we think about the recipe we need to make this work, we need to find a code that has

- (a) the dual code has good distance and
 - (b) can be efficiently decoded
 - (c) nearly optimal list recovery
- } For the Yamakawa-Zhandry algorithm to work
- To get soundness against classical proofs

Trying to actually instantiate this leads to a very common struggle!

A common experience: Bias and pad

In our sketch, we needed extremely strong list decoding (basically optimal). This is a very common problem when using code intersection [GGJL25, CKMRST'26].

A common experience: Bias and pad

In our sketch, we needed extremely strong list decoding (basically optimal). This is a very common problem when using code intersection [GGJL25, CKMRST'26].

→ The original Yamakawa-Zhandry paper considers using folded Reed-Solomon codes, but these are not known to have the list recovery we need.

A common experience: Bias and pad

In our sketch, we needed extremely strong list decoding (basically optimal). This is a very common problem when using code intersection [GGJL25, CKMRST'26].

→ The original Yamakawa-Zhandry paper considers using folded Reed-Solomon codes, but these are not known to have the list recovery we need.

Therefore, we need to pick a better code!

A common experience: Bias and pad

In our sketch, we needed extremely strong list decoding (basically optimal). This is a very common problem when using code intersection [GGJL25, CKMRST'26].

→ The original Yamakawa-Zhandry paper considers using folded Reed-Solomon codes, but these are not known to have the list recovery we need.

Therefore, we need to pick a better code!

→ There are codes that have really really good list recoverability (one such family is called multiplicity codes), but the distance of their dual codes is not constant...

A common experience: Bias and pad

In our sketch, we needed extremely strong list decoding (basically optimal). This is a very common problem when using code intersection [GGJL25, CKMRST'26].

→ The original Yamakawa-Zhandry paper considers using folded Reed-Solomon codes, but these are not known to have the list recovery we need.

Therefore, we need to pick a better code!

→ There are codes that have really really good list recoverability (one such family is called multiplicity codes), but the distance of their dual codes is not constant...

So another tweak is needed... bias and pad!

A common experience: Bias and pad

Instead of picking random functions $f_1, \dots, f_n$, we can pick functions that are random, but have more 0's than 1's (or the other way around). Recall that:

$$|\langle 0^n | H^n | f^{-1}(0) \rangle|^2 = |f^{-1}(0)|/2^n,$$

A common experience: Bias and pad

Instead of picking random functions $f_1, \dots, f_n$, we can pick functions that are random, but have more 0's than 1's (or the other way around). Recall that:

$$|\langle 0^n | H^n | f^{-1}(0) \rangle|^2 = |f^{-1}(0)| / 2^n,$$

So if we ask for more 0's in the hash, our error will be smaller. Instead of asking for a random hash $x \in \{0,1\}^n$, we can pad x with a bunch of 0's so that the resulting error will be low weight, but the problem should still be hard!

Questions I have about code intersection

The Yamakawa-Zhandry algorithm and code intersection problem are fascinating!

Here are a few open questions related to it:

Questions I have about code intersection

1. Can you get a one-shot signature from it?

Questions I have about code intersection

1. Can you get a one-shot signature from it?

A one-shot signature is a pair:

sign_{sk} : Given x , output $\text{sign}_{sk}(x)$

Ver_{pk} : Given $(x, \text{sign}_{sk}(x))$ check if correct

But, the signing key is a quantum state,
and can't sign two different messages.

Questions I have about code intersection

1. Can you get a one-shot signature from it?
→ The idea is you sign x by finding $\mathbf{v}$ hashing to x , but how do you get the state?

Questions I have about code intersection

1. Can you get a one-shot signature from it?
→ The idea is you sign x by finding $\mathbf{v}$ hashing to x , but how do you get the state?
2. Is the Yamakawa-Zhandry algorithm (or a variant) rigid?

Questions I have about code intersection

1. Can you get a one-shot signature from it?
 - The idea is you sign x by finding $\mathbf{v}$ hashing to x , but how do you get the state?
2. Is the Yamakawa-Zhandry algorithm (or a variant) rigid?
 - Is the tensor product of pre-images (w/ phases) the only state that let's you find $\mathbf{v}$?
 - What about among product states? (Is the only product state the uniform superposition over the pre-images?)

Questions I have about code intersection

1. Can you get a one-shot signature from it?
 - The idea is you sign x by finding $\mathbf{v}$ hashing to x , but how do you get the state?
2. Is the Yamakawa-Zhandry algorithm (or a variant) rigid?
 - Is the tensor product of pre-images (w/ phases) the only state that let's you find $\mathbf{v}$?
 - What about among product states? (Is the only product state the uniform superposition over the pre-images?)

Typically rigidity results would look something like “If you solve code intersection, using the YZ algorithm, with probability $1 - \epsilon$, the starting state must be $O(\epsilon)$ close”

Questions I have about code intersection

1. Can you get a one-shot signature from it?
→ The idea is you sign x by finding $\mathbf{v}$ hashing to x , but how do you get the state?
2. Is the Yamakawa-Zhandry algorithm (or a variant) rigid?
→ Is the tensor product of pre-images (w/ phases) the only state that let's you find $\mathbf{v}$?
→ What about among product states? (Is the only product state the uniform superposition over the pre-images?)

Typically rigidity results would look something like “If you solve code intersection, using the YZ algorithm, with probability $1 - \epsilon$, the starting state must be $O(\epsilon)$ close”

3. Is the Yamakawa-Zhandry algorithm “complete”?
→ Often times we get separations using black-box versions of complete problems.
For example: A random function $\sim$ SAT, spectral Forrelation $\sim$ XZ-Hamiltonian

Questions I have about code intersection

1. Can you get a one-shot signature from it?
→ The idea is you sign x by finding $\mathbf{v}$ hashing to x , but how do you get the state?
2. Is the Yamakawa-Zhandry algorithm (or a variant) rigid?
→ Is the tensor product of pre-images (w/ phases) the only state that let's you find $\mathbf{v}$?
→ What about among product states? (Is the only product state the uniform superposition over the pre-images?)

Typically rigidity results would look something like “If you solve code intersection, using the YZ algorithm, with probability $1 - \epsilon$, the starting state must be $O(\epsilon)$ close”

3. Is the Yamakawa-Zhandry algorithm “complete”?
→ Often times we get separations using black-box versions of complete problems.
For example: A random function $\sim$ SAT, spectral Forrelation $\sim$ XZ-Hamiltonian
→ The version of the problem I presented here is in AM, is there a variant that is based on a “complete” problem for BQP or QMA?