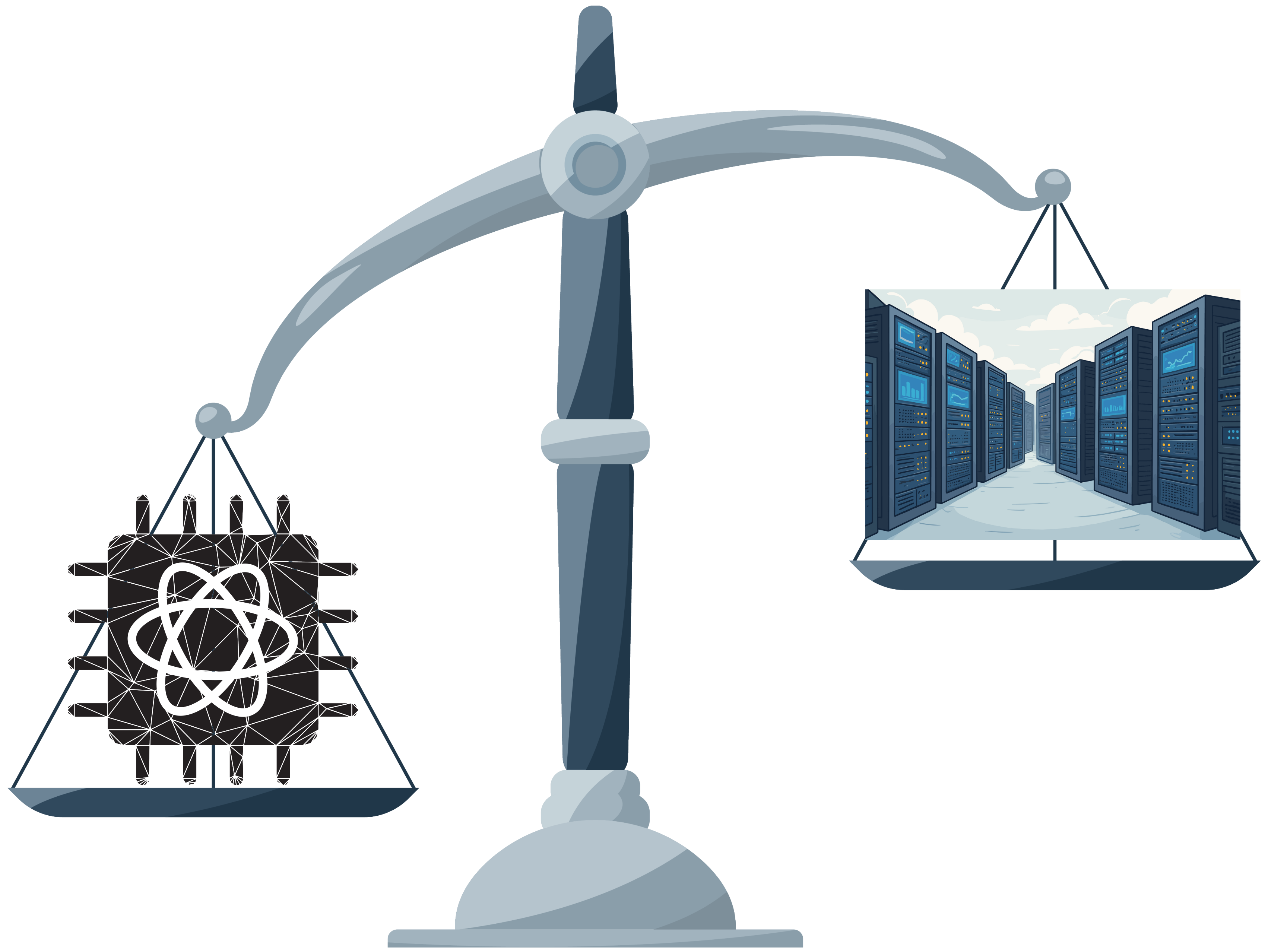
Quantum Factoring
(and why you should try to factor P^2Q)

Seyoon Ragavan (MIT)

Based on joint work with Gregory D. Kahanamoku-Meyer*, Vinod Vaikuntanathan*, and Katherine Van Kirk†
*MIT, †Harvard



Is there a problem that is "very easy" with a quantum computer, but difficult without one?



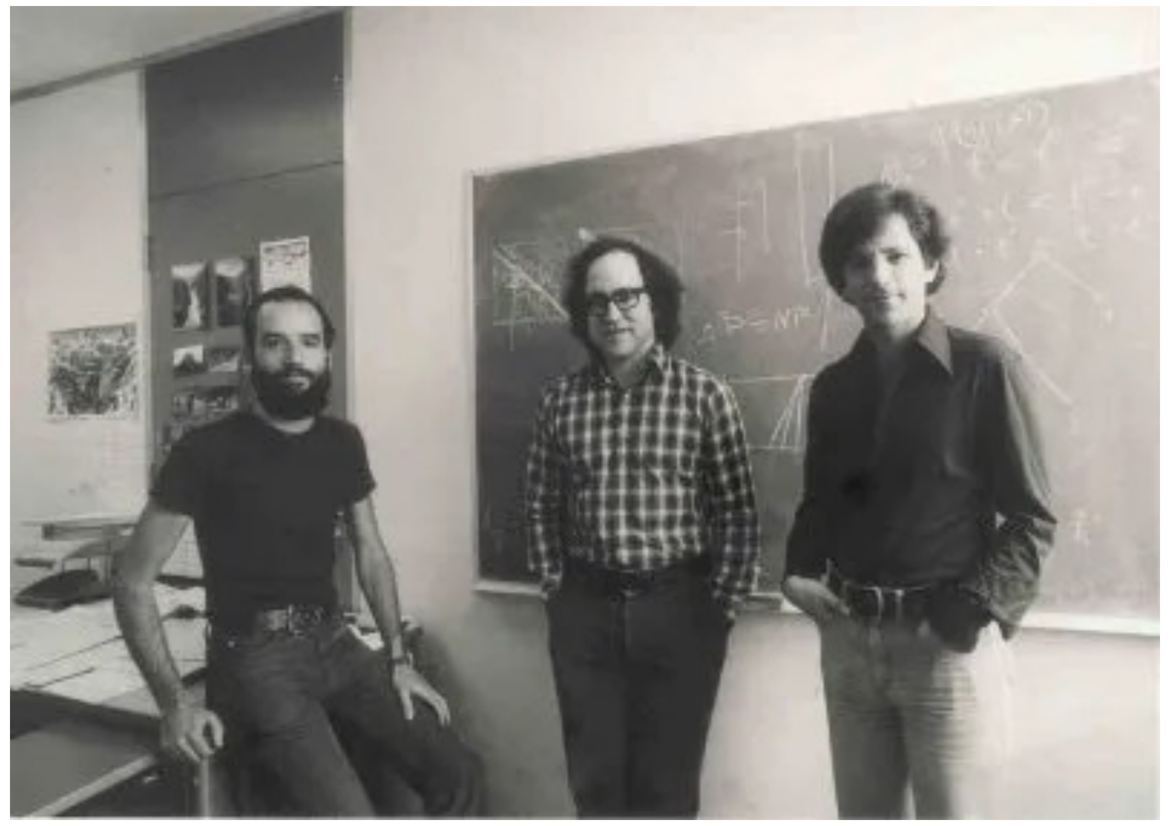
One of Few Candidates: Integer Factorisation

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

One of Few Candidates: Integer Factorisation

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

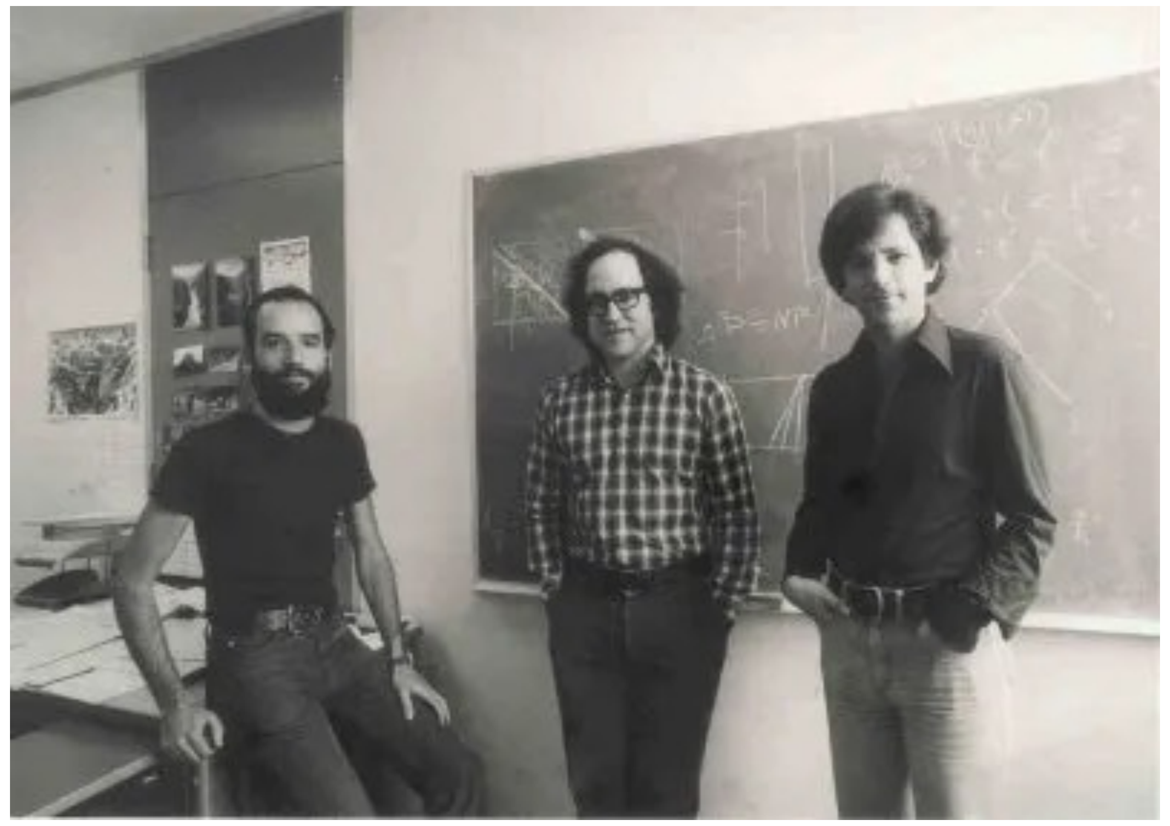
- Security of RSA encryption relies on this problem being hard



One of Few Candidates: Integer Factorisation

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

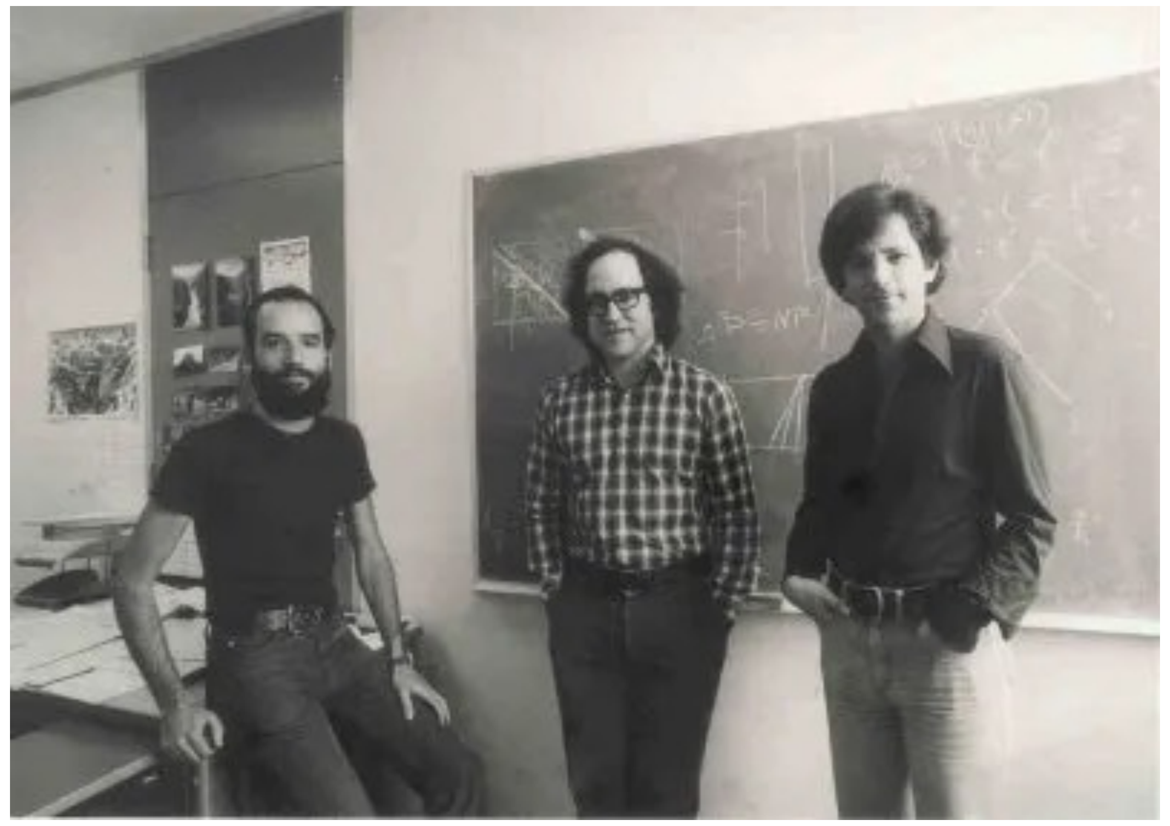
- Security of RSA encryption relies on this problem being hard
- Useful notation: $L_N[\alpha, c] = \exp\left((c + o(1))(\log N)^{\alpha}(\log\log N)^{1-\alpha}\right)$



One of Few Candidates: Integer Factorisation

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

- Security of RSA encryption relies on this problem being hard
- Useful notation: $L_N[\alpha, c] = \exp\left((c + o(1))(\log N)^\alpha(\log\log N)^{1-\alpha}\right)$
- $c > 0$ and $\alpha \in \left[0, 1\right]$
  - $\alpha = 0$: $(\log N)^c \to$ the dream
  - $\alpha = 1$: $N^c \to$ brute force



Classical Factoring Algorithms

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

Factoring general N

- Brute force: $N^{\Theta(1)} = L_N\left[1, \Theta(1)\right]$

Classical Factoring Algorithms

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

Factoring general N

- Brute force: $N^{\Theta(1)} = L_N\left[1, \Theta(1)\right]$
- Quadratic sieve (many works from Fermat to Pomerance '82): $L_N\left[1/2, 1\right]$

Classical Factoring Algorithms

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

Factoring general N

- Brute force: $N^{\Theta(1)} = L_N\left[1, \Theta(1)\right]$
- Quadratic sieve (many works from Fermat to Pomerance '82): $L_N\left[1/2, 1\right]$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $L_N\left[1/3, (64/9)^{1/3}\right]$

Classical Factoring Algorithms

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

Factoring general N

- Brute force: $N^{\Theta(1)} = L_N\left[1, \Theta(1)\right]$
- Quadratic sieve (many works from Fermat to Pomerance '82): $L_N\left[1/2, 1\right]$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $L_N\left[1/3, (64/9)^{1/3}\right]$

Factoring N with a small prime factor Q

Classical Factoring Algorithms

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

<u>Factoring general N</u>

- Brute force: $N^{\Theta(1)} = L_N\left[1, \Theta(1)\right]$
- Quadratic sieve (many works from Fermat to Pomerance '82): $L_N\left[1/2, 1\right]$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $L_N\left[1/3, (64/9)^{1/3}\right]$

<u>Factoring N with a small prime factor Q</u>

- Lenstra elliptic curve method ('87): $L_Q\left[1/2, \sqrt{2}\right]$

Classical Factoring Algorithms

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

<u>Factoring general N</u>

- Brute force: $N^{\Theta(1)} = L_N\left[1, \Theta(1)\right]$
- Quadratic sieve (many works from Fermat to Pomerance '82): $L_N\left[1/2, 1\right]$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $L_N\left[1/3, (64/9)^{1/3}\right]$

<u>Factoring N with a small prime factor Q</u>

- Lenstra elliptic curve method ('87): $L_Q\left[1/2, \sqrt{2}\right]$
- Mulder '24 (ANTS Selfridge Prize): if $N = P^2Q$, can factor in time $L_Q\left[1/2, 1\right]$

Classical Factoring Algorithms

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

Factoring general N

- Brute force: $N^{\Theta(1)} = L_N\left[1, \Theta(1)\right]$
- Quadratic sieve (many works from Fermat to Pomerance '82): $L_N\left[1/2, 1\right]$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $L_N\left[1/3, (64/9)^{1/3}\right]$

Factoring N with a small prime factor Q

- Lenstra elliptic curve method ('87): $L_Q\left[1/2, \sqrt{2}\right]$
- Mulder '24 (ANTS Selfridge Prize): if $N = P^2Q$, can factor in time $L_Q\left[1/2, 1\right]$
  - Uses class groups of binary quadratic forms

The Quantum Factoring Landscape

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

The Quantum Factoring Landscape

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

Factoring general N

- Shor '94: $\tilde{O}((\log N)^2)$ time, $\tilde{O}(\log N)$ quantum space/memory

The Quantum Factoring Landscape

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

<u>Factoring general N</u>

- Shor '94: $\tilde{O}((\log N)^2)$ time, $\tilde{O}(\log N)$ quantum space/memory
- Regev '23 and **R,** Vaikuntanathan '24: $\tilde{O}((\log N)^{3/2})$ time, $\tilde{O}(\log N)$ quantum space

The Quantum Factoring Landscape

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

Factoring general N

- Shor '94: $\tilde{O}((\log N)^2)$ time, $\tilde{O}(\log N)$ quantum space/memory
- Regev '23 and **R,** Vaikuntanathan '24: $\tilde{O}((\log N)^{3/2})$ time, $\tilde{O}(\log N)$ quantum space

Factoring $N = P^2Q$

The Quantum Factoring Landscape

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

Factoring general N

- Shor '94: $\tilde{O}((\log N)^2)$ time, $\tilde{O}(\log N)$ quantum space/memory
- Regev '23 and **R,** Vaikuntanathan '24: $\tilde{O}((\log N)^{3/2})$ time, $\tilde{O}(\log N)$ quantum space

Factoring $N = P^2Q$

- Li, Peng, Du, Suter '12: $\tilde{O}(\log N)$ time and quantum space

The Quantum Factoring Landscape

Given a large integer N, find its prime factorisation in $\mathsf{poly}(\log N)$ time.

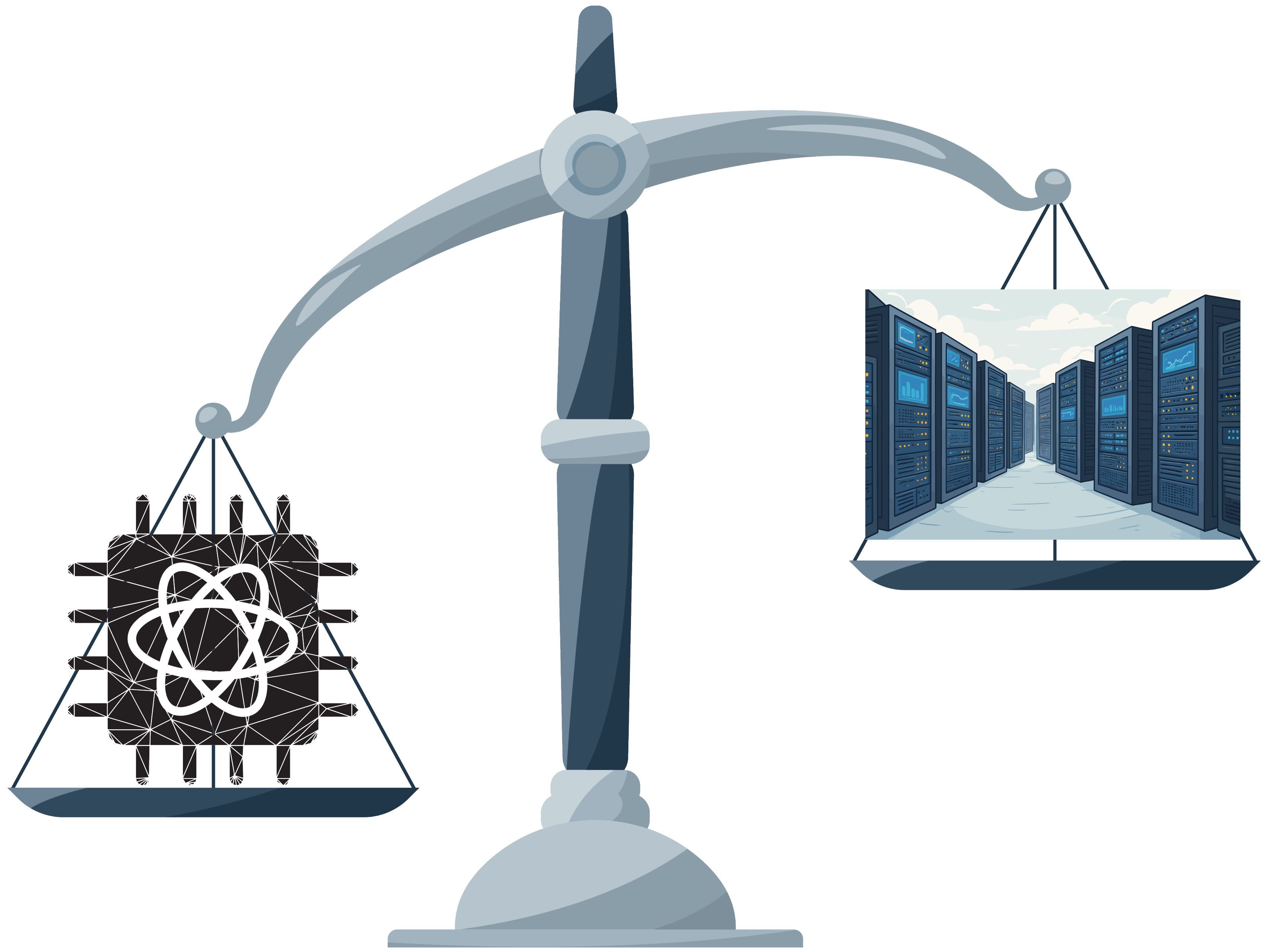
Factoring general N

- Shor '94: $\tilde{O}((\log N)^2)$ time, $\tilde{O}(\log N)$ quantum space/memory
- Regev '23 and **R,** Vaikuntanathan '24: $\tilde{O}((\log N)^{3/2})$ time, $\tilde{O}(\log N)$ quantum space

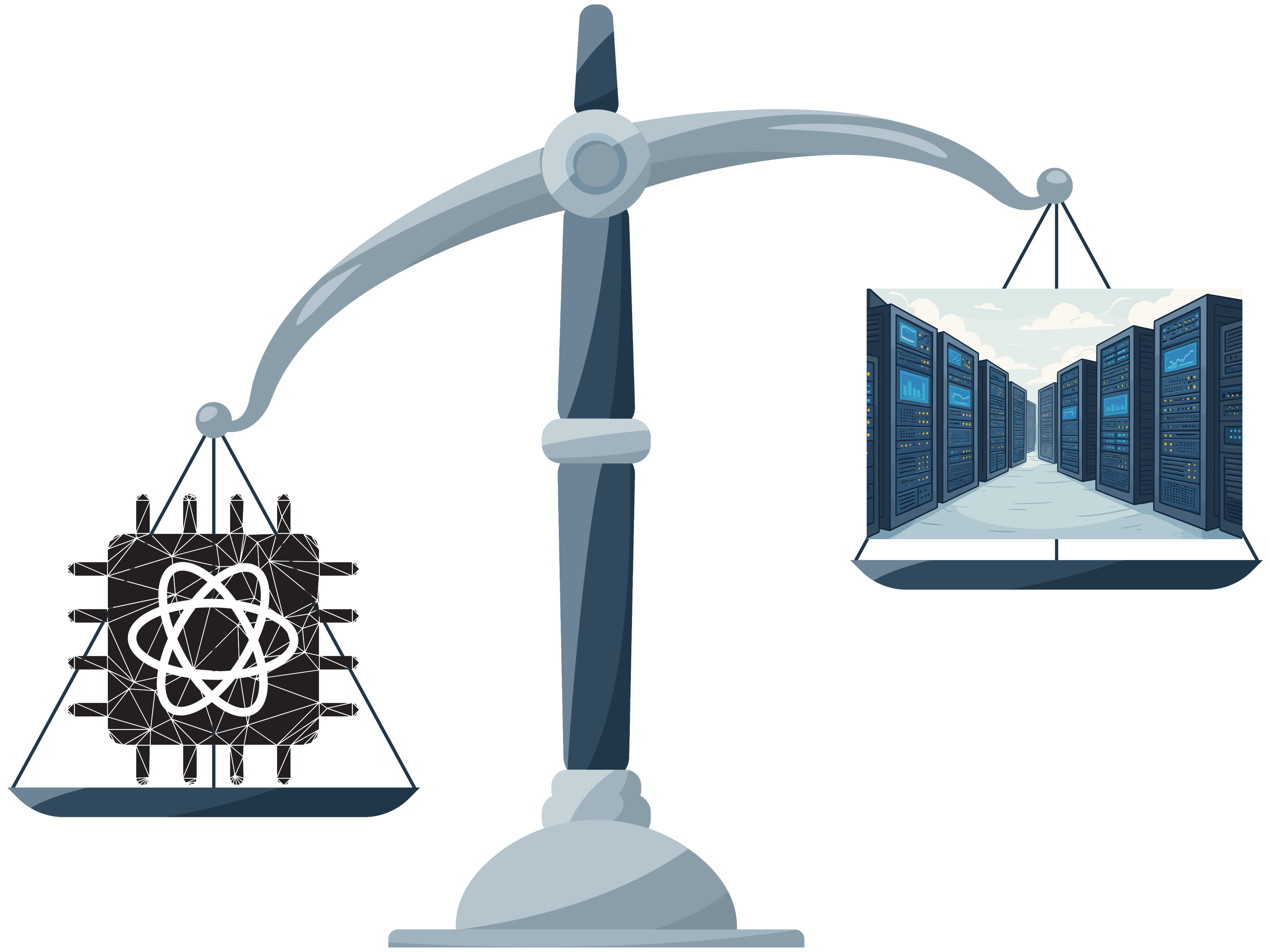
Factoring $N = P^2 Q$

- Li, Peng, Du, Suter '12: $\tilde{O}(\log N)$ time and quantum space
- Kahanamoku-Meyer, **R**, Vaikuntanathan, Van Kirk '25: $\tilde{O}(\log N)$ time, $\tilde{O}(\log Q)$ quantum space

Is factoring $N = P^2Q$ with $Q \ll N$ "very easy" with a quantum computer, but difficult without one?

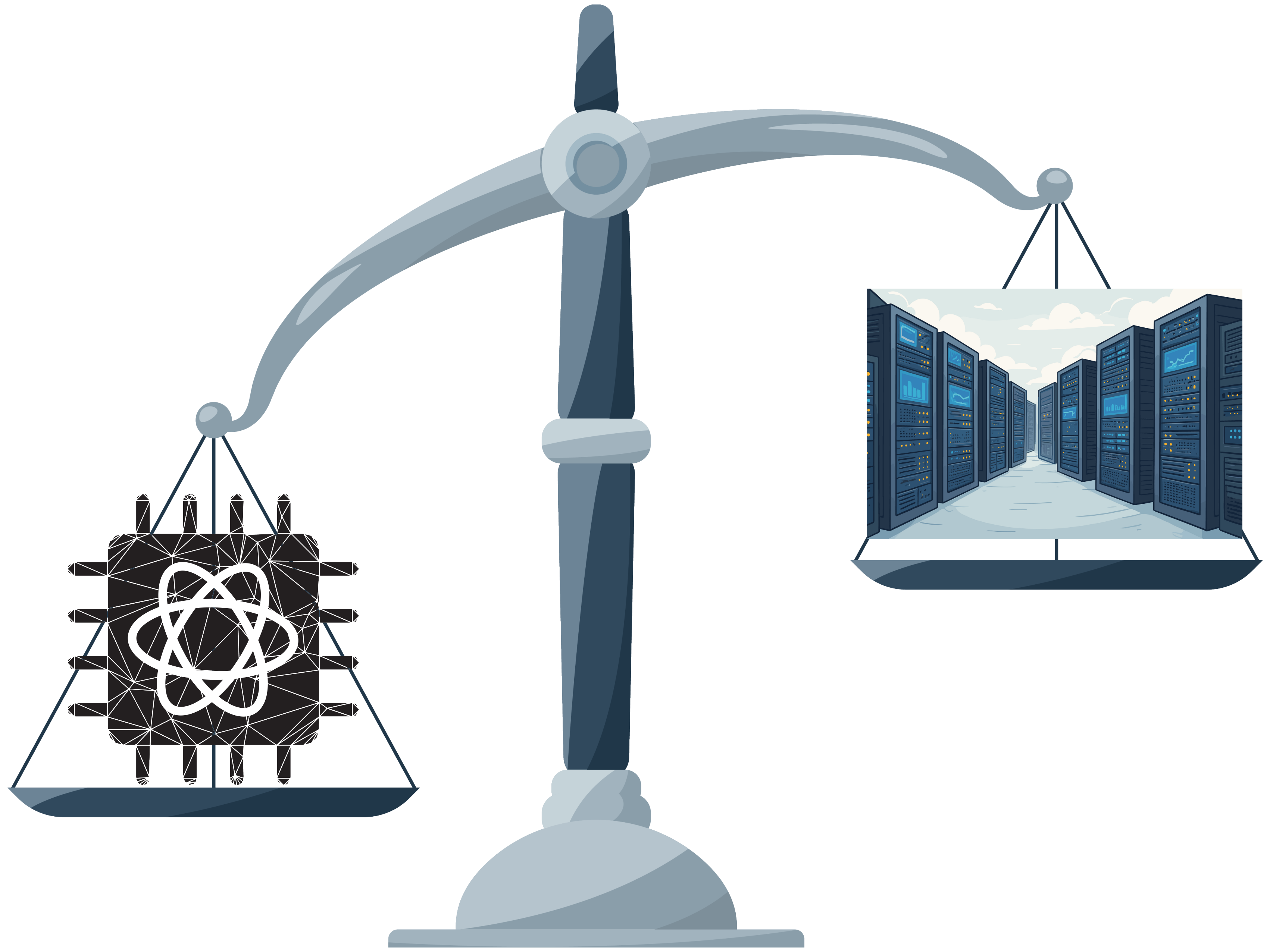


Is factoring $N = P^2Q$ with $Q \ll N$ "very easy" with a quantum computer, but difficult without one?



K**R**VV'25: $\tilde{O}(\log N)$ time, $\tilde{O}(\log Q)$ space

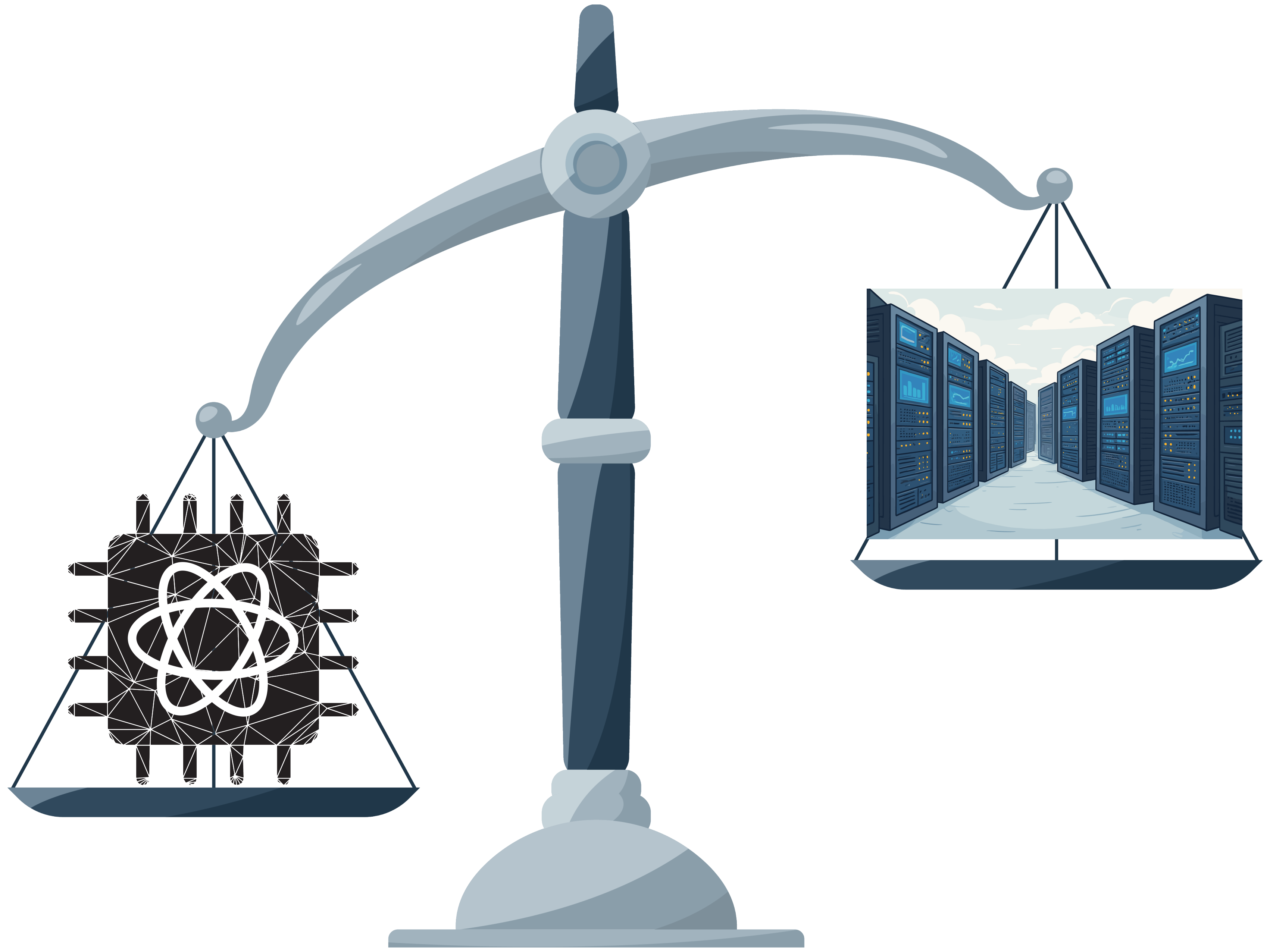
Is factoring $N = P^2Q$ with $Q \ll N$ "very easy" with a quantum computer, but difficult without one?



KRVV'25: $\tilde{O}(\log N)$ time, $\tilde{O}(\log Q)$ space

- Number field sieve: $L_N\left[1/3, (64/9)^{1/3}\right]$

Is factoring $N = P^2 Q$ with $Q \ll N$ "very easy" with a quantum computer, but difficult without one?

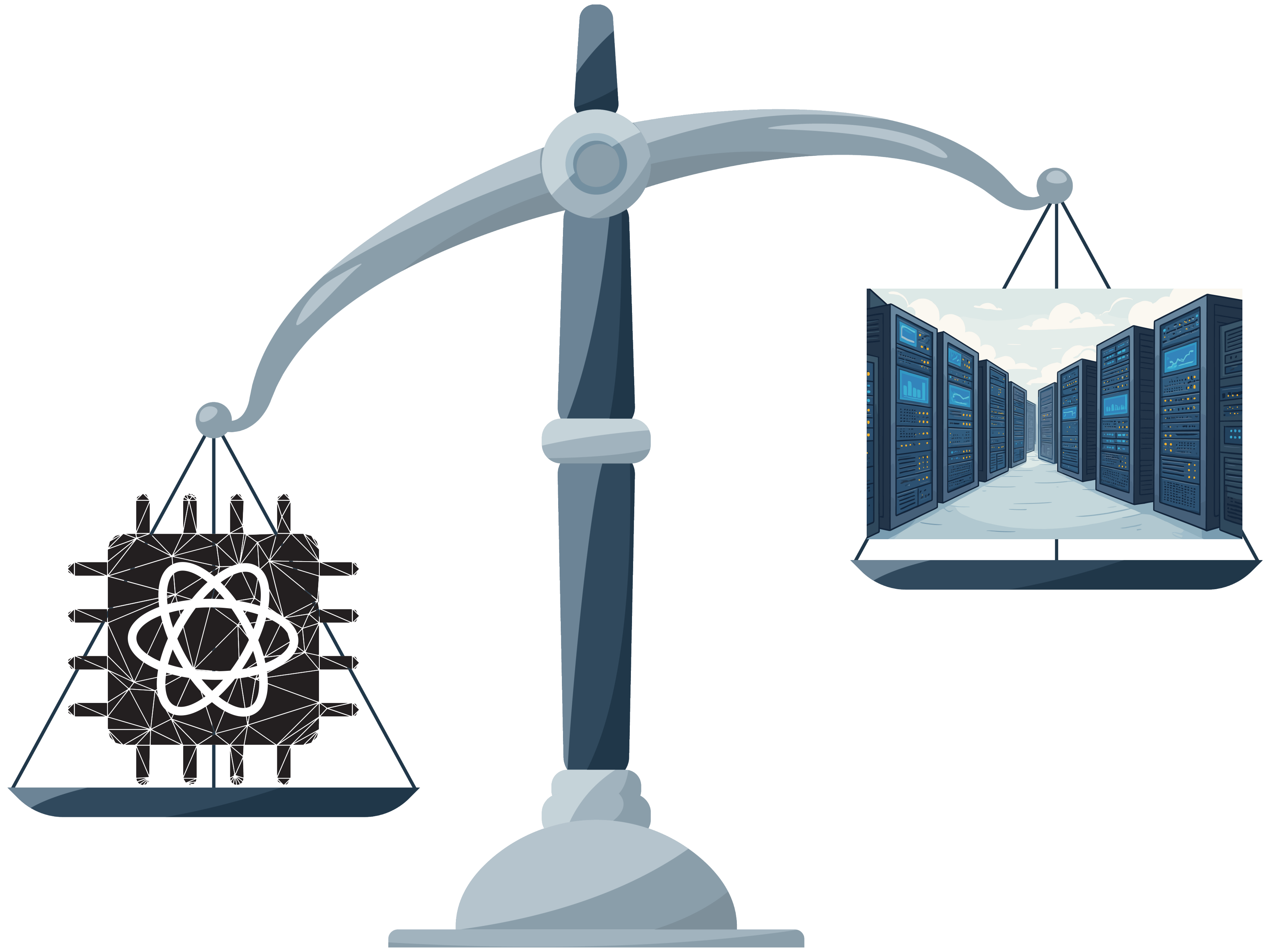


KRVV'25: $\tilde{O}(\log N)$ time, $\tilde{O}(\log Q)$ space

- Number field sieve: $L_N\left[1/3, (64/9)^{1/3}\right]$
- Mulder '24: $L_Q\left[1/2, 1\right]$

Is factoring $N = P^2Q$ with $Q \ll N$ "very easy" with a quantum computer, but difficult without one?

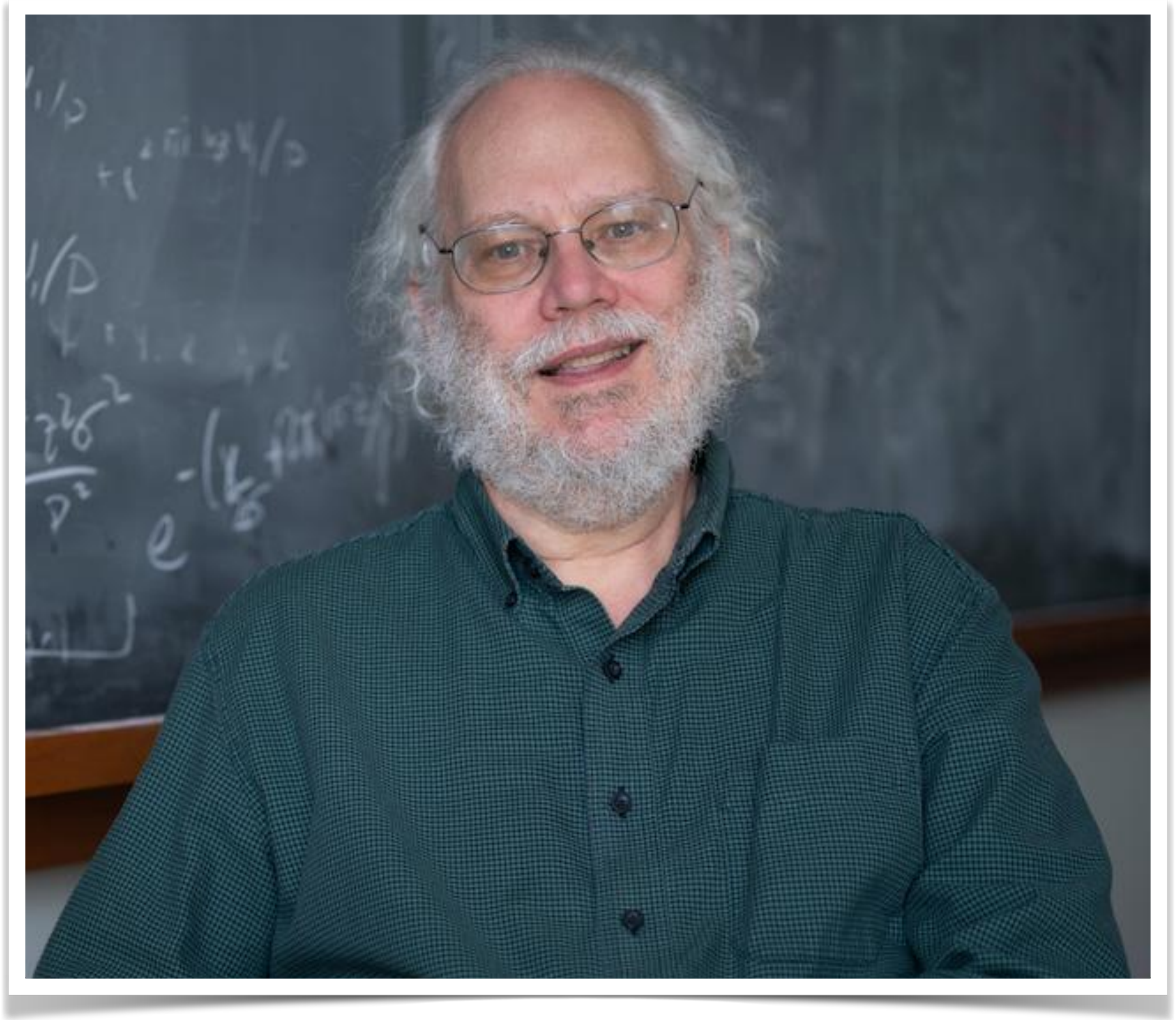
KRVV'25: $\tilde{O}(\log N)$ time, $\tilde{O}(\log Q)$ space



- Number field sieve: $L_N\left[1/3, (64/9)^{1/3}\right]$
- Mulder '24: $L_Q\left[1/2, 1\right]$
- **Anything better?**

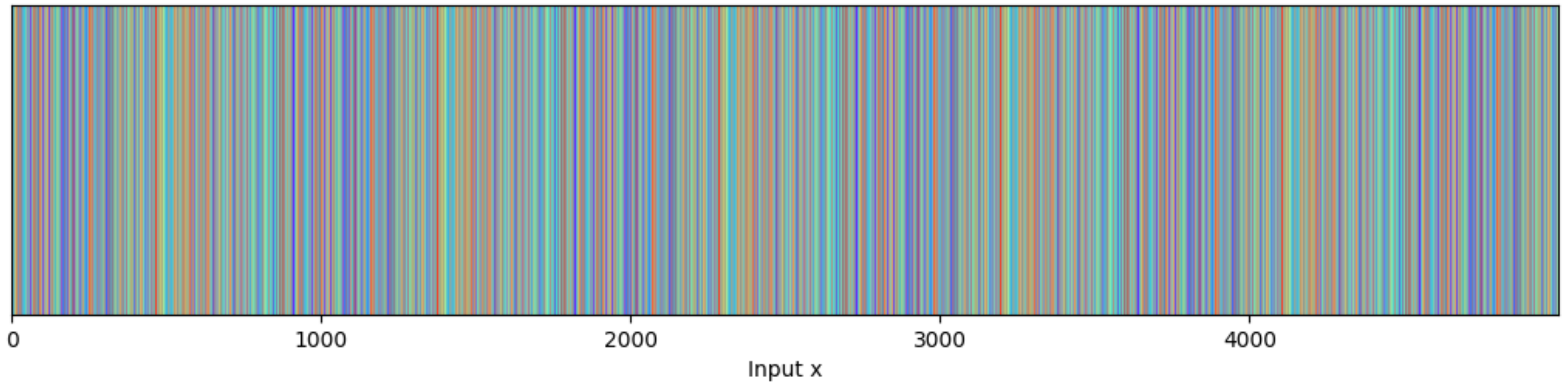

THE WORLD
NEEDS YOU TO FIND
NEW CLASSICAL
ALGORITHMS FOR FACTORING P^2Q
imgflip.com

Shor's Algorithm: A Sketch



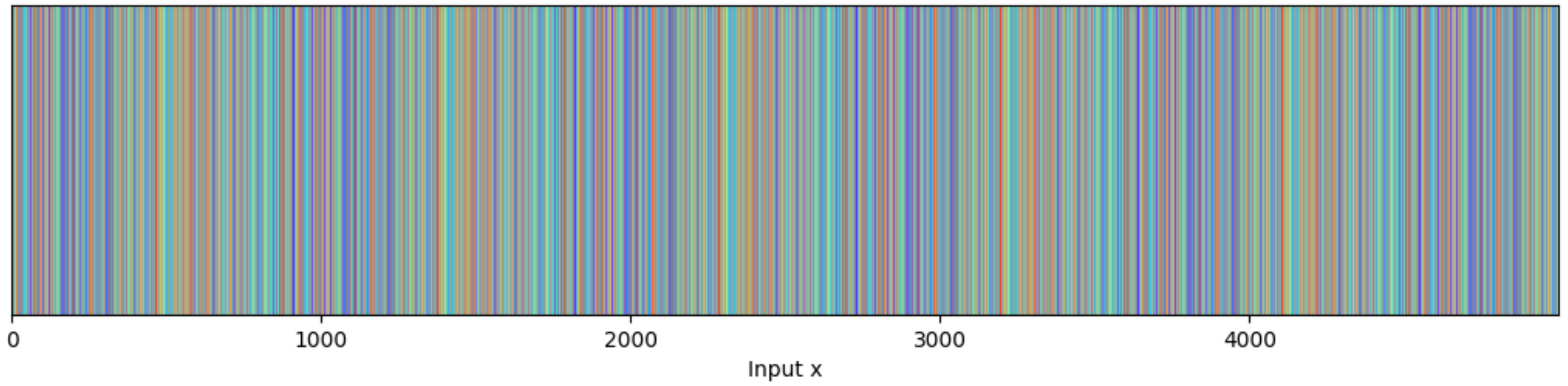
Preliminary: Quantum Period Finding

- Periodic function $f : \mathbb{Z} \to \mathbb{Z}$ with unknown period T
  - $x \equiv y \pmod{T} \Rightarrow f(x) = f(y)$



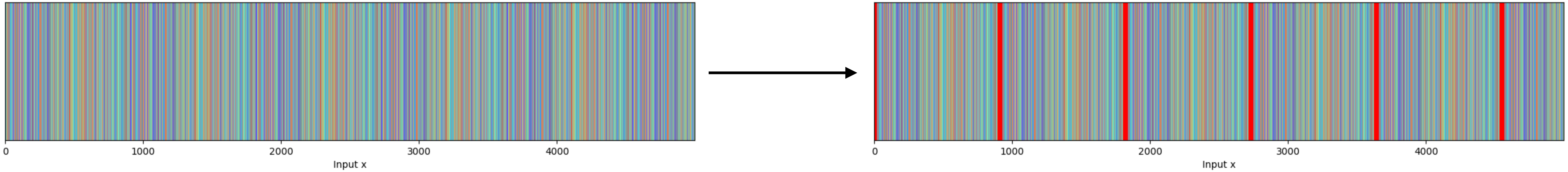
Preliminary: Quantum Period Finding

- Periodic function $f : \mathbb{Z} \to \mathbb{Z}$ with unknown period T
  - $x \equiv y \pmod{T} \Rightarrow f(x) = f(y)$
  - T is exponentially large



Preliminary: Quantum Period Finding

- Periodic function $f : \mathbb{Z} \to \mathbb{Z}$ with unknown period T
  - $x \equiv y \pmod{T} \Rightarrow f(x) = f(y)$
  - T is exponentially large
- Informal theorem statement: can quantumly recover a uniformly random multiple of $1/T$ (and hence T itself) using essentially only the time/space needed to compute $f(x)$ for $|x| \le \mathrm{poly}(T)$



Shor overview: finding square roots of 1

- Goal: find $z \not\equiv \pm 1 \bmod N$ such that $z^2 \equiv 1 \bmod N$
  - 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

Shor overview: reduction to period-finding

- Choose a random integer a as a base e.g. 2

Shor overview: reduction to period-finding

- Choose a random integer a as a base e.g. 2
- The powers $a^0, a^2, a^4, a^6 \ldots$ eventually repeat mod N

Shor overview: reduction to period-finding

- Choose a random integer a as a base e.g. 2
- The powers $a^0, a^2, a^4, a^6 \ldots$eventually repeat mod N
  - Let $r < N$ be the period i.e. the first positive index where $a^{2r} \equiv 1 \bmod N$
  - We can find r using quantum period finding!

Shor overview: reduction to period-finding

- Choose a random integer a as a base e.g. 2
- The powers $a^0, a^2, a^4, a^6\ldots$eventually repeat mod N
  - Let $r < N$ be the period i.e. the first positive index where $a^{2r} \equiv 1 \mod N$
  - We can find r using quantum period finding!
- Now a^r is a square root of $1 \mod N$

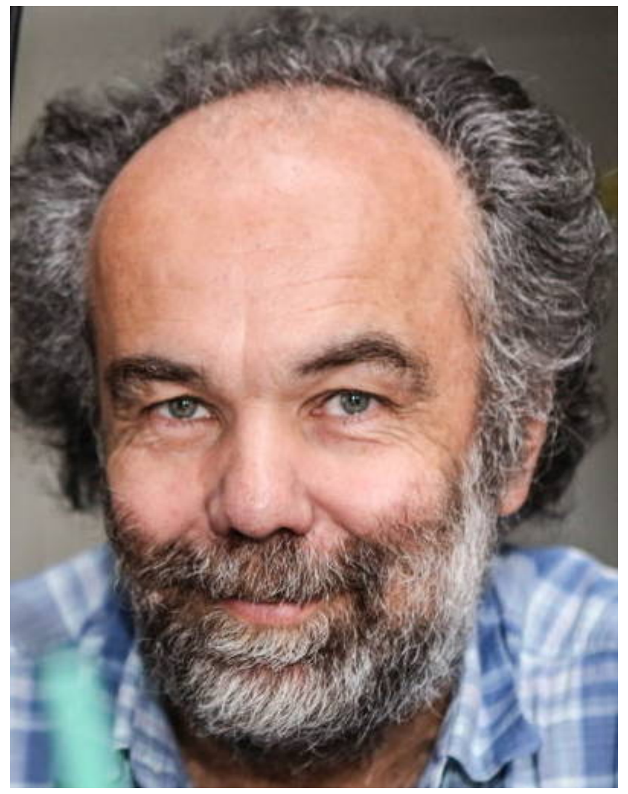
Shor overview: reduction to period-finding

- Choose a random integer a as a base e.g. 2
- The powers $a^0, a^2, a^4, a^6 \ldots$ eventually repeat mod N
    - Let $r < N$ be the period i.e. the first positive index where $a^{2r} \equiv 1 \mod N$
    - We can find r using quantum period finding!
- Now a^r is a square root of $1 \mod N$
    - With some luck: $a^r \not\equiv \pm 1 \mod N$ so this would give us a factor!

Shor overview: reduction to period-finding

- Choose a random integer a as a base e.g. 2
- The powers $a^0, a^2, a^4, a^6 \ldots$ eventually repeat mod N
  - Let $r < N$ be the period i.e. the first positive index where $a^{2r} \equiv 1 \mod N$
  - We can find r using quantum period finding!
- Now a^r is a square root of $1 \mod N$
  - With some luck: $a^r \not\equiv \pm 1 \mod N$ so this would give us a factor!
  - *"Luck" is with respect to the randomly chosen base*

Factoring P^2Q with LPDS12: A Sketch

Preliminary: The Legendre Symbol

- a is a *quadratic residue* modulo an odd prime p if there exists integer x such that $a \equiv x^2 \pmod{p}$

Preliminary: The Legendre Symbol

- a is a *quadratic residue* modulo an odd prime p if there exists integer x such that $a \equiv x^2 \pmod{p}$
- Legendre symbol essentially indicates whether this is the case:

$$\left(\frac{a}{p}\right) = \begin{cases} 1, \text{ if } a \text{ is a nonzero quadratic residue modulo } p\text{; and} \\ -1, \text{ if } a \text{ is not a quadratic residue modulo } p\text{; and} \\ 0, \text{ if } a \text{ divisible by } p\,. \end{cases}$$

Preliminary: The Jacobi Symbol

- Essentially generalises the Legendre symbol to odd composite moduli

Preliminary: The Jacobi Symbol

- Essentially generalises the Legendre symbol to odd composite moduli
- For $N = p_1^{\alpha_1} \ldots p_r^{\alpha_r}$, define:

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \cdots \left(\frac{a}{p_r}\right)^{\alpha_r}$$

Preliminary: The Jacobi Symbol

- Essentially generalises the Legendre symbol to odd composite moduli
- For $N = p_1^{\alpha_1} \dots p_r^{\alpha_r}$, define:

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \dots \left(\frac{a}{p_r}\right)^{\alpha_r}$$

- Note for intuition: the quadratic residue characterisation does **not** carry over from the Legendre symbol

Preliminary: The Jacobi Symbol

- Essentially generalises the Legendre symbol to odd composite moduli
- For $N = p_1^{\alpha_1} \dots p_r^{\alpha_r}$, define:

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \dots \left(\frac{a}{p_r}\right)^{\alpha_r}$$

- Note for intuition: the quadratic residue characterisation does **not** carry over from the Legendre symbol
  - Could have $\left(\frac{a}{N}\right) = 1$ without a being a quadratic residue modulo N

Preliminary: The Jacobi Symbol

- Essentially generalises the Legendre symbol to odd composite moduli
- For $N = p_1^{\alpha_1} \ldots p_r^{\alpha_r}$, define:

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \ldots \left(\frac{a}{p_r}\right)^{\alpha_r}$$

- Useful property: $a \equiv b \pmod{N} \Rightarrow \left(\frac{a}{N}\right) = \left(\frac{b}{N}\right)$

Preliminary: The Jacobi Symbol

- Essentially generalises the Legendre symbol to odd composite moduli
- For $N = p_1^{\alpha_1} \ldots p_r^{\alpha_r}$, define:

$$\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \ldots \left(\frac{a}{p_r}\right)^{\alpha_r}$$

- Useful property: $a \equiv b \pmod{N} \Rightarrow \left(\frac{a}{N}\right) = \left(\frac{b}{N}\right)$
- Theorem (from Euclid to Schönhage 1971): can compute $\left(\frac{a}{N}\right)$ efficiently without knowing the factorisation of N — in fact, in time $\tilde{O}(\log N)$

Factoring from Jacobi Symbol Periodicity

- For RSA integers ($N = PQ$): product of two periodic functions with smaller periods but itself only has period N

$$\left(\frac{a}{N}\right) = \left(\frac{a}{P}\right)\left(\frac{a}{Q}\right)$$

Factoring from Jacobi Symbol Periodicity

- For RSA integers ($N = PQ$): product of two periodic functions with smaller periods but itself only has period N

$$\left(\frac{a}{N}\right) = \left(\frac{a}{P}\right)\left(\frac{a}{Q}\right)$$

- What about $N = P^2Q$?

$$\left(\frac{a}{N}\right) = \left(\frac{a}{P}\right)^2\left(\frac{a}{Q}\right) = \left(\frac{a}{Q}\right)$$, which is periodic* with period Q!

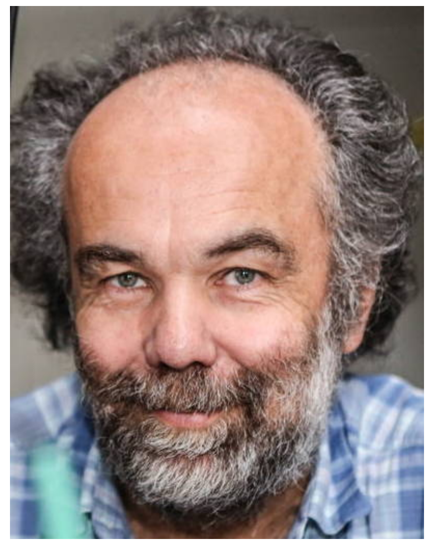
* modulo minor technical caveats; could have $\left(\frac{a}{P}\right) = 0$ for a tiny fraction of inputs a

Quantumly Factoring $N = P^2Q$

Li, Peng, Du, Suter (2012)

- We know $\left(\frac{a}{N}\right)$ is periodic with period Q!
- So quantum period finding → recover Q (and hence P)

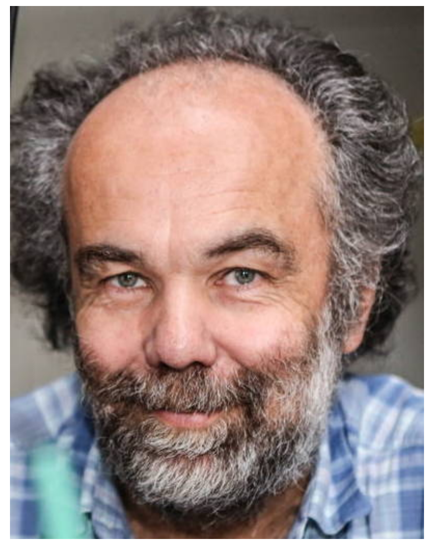


Quantumly Factoring $N = P^2Q$

Li, Peng, Du, Suter (2012)

- We know $\left(\frac{a}{N}\right)$ is periodic with period Q!
- So quantum period finding → recover Q (and hence P)
- Time: cost of computing $\left(\frac{a}{N}\right)$ for $a \leq \mathrm{poly}(Q)$, which is $\tilde{O}(\log N)$

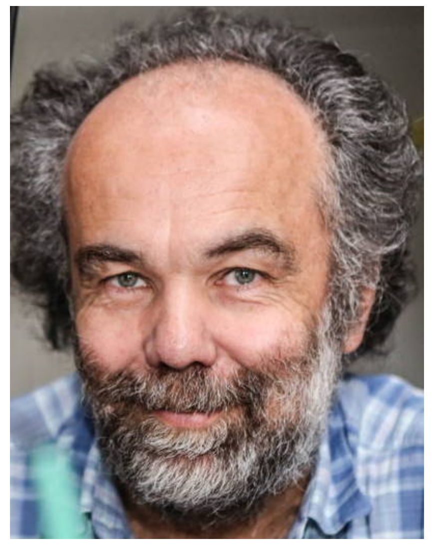


Quantumly Factoring $N = P^2Q$

Li, Peng, Du, Suter (2012)

- We know $\left(\frac{a}{N}\right)$ is periodic with period Q!
- So quantum period finding → recover Q (and hence P)
- Time: cost of computing $\left(\frac{a}{N}\right)$ for $a \leq \text{poly}(Q)$, which is $\tilde{O}(\log N)$
- Space (if naively implemented): also $\tilde{O}(\log N)$



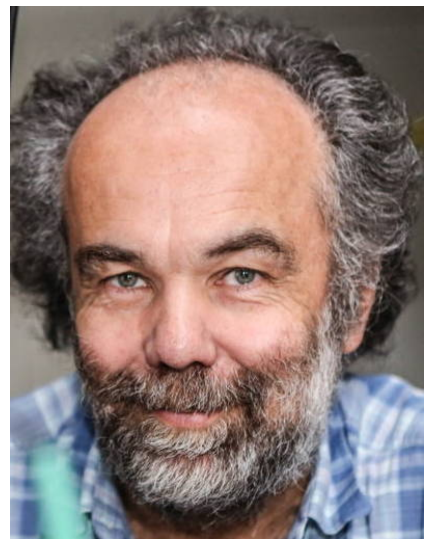



Quantumly Factoring $N = P^2Q$

Li, Peng, Du, Suter (2012)

- We know $\left(\frac{a}{N}\right)$ is periodic with period Q!
- So quantum period finding → recover Q (and hence P)
- Time: cost of computing $\left(\frac{a}{N}\right)$ for $a \leq \text{poly}(Q)$, which is $\tilde{O}(\log N)$
- Space (if naively implemented): also $\tilde{O}(\log N)$
  - K**R**VV'25: the space can be brought down to $\tilde{O}(\log Q)$



Outlook: The True Difficulty of Factoring

Outlook: The True Difficulty of Factoring

Things I do NOT expect to find in The Book

Outlook: The True Difficulty of Factoring

Things I do NOT expect to find in The Book

- The time needed to classically factor an integer N is $L_N\left[1/3, (64/9)^{1/3}\right]$

Outlook: The True Difficulty of Factoring

Things I do NOT expect to find in The Book

- The time needed to classically factor an integer N is $L_N\left[1/3, (64/9)^{1/3}\right]$
- The time needed to quantumly factor is $\tilde{O}\left((\log N)^{3/2}\right)$ (Regev)...

Outlook: The True Difficulty of Factoring

<u>Things I do NOT expect to find in The Book</u>

- The time needed to classically factor an integer N is $L_N\left[1/3, (64/9)^{1/3}\right]$
- The time needed to quantumly factor is $\tilde{O}\left((\log N)^{3/2}\right)$ (Regev)…
  - … unless N has a square factor, in which case we just need $\tilde{O}(\log N)$ time (Jacobi)

Thank you for your attention! Questions?

Bonus Slides

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

<u>Jacobi Properties</u>

- Periodicity: $\left(\frac{a}{b}\right) = \left(\frac{a \bmod b}{b}\right)$
- Reciprocity:* $\left(\frac{a}{b}\right) = (-1)^{f(a,b)}\left(\frac{b}{a}\right)$

  for a very simple f

* modulo minor technical caveats; requires a, b odd

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

Jacobi Properties

- Periodicity: $\left(\frac{a}{b}\right) = \left(\frac{a \bmod b}{b}\right)$
- Reciprocity:* $\left(\frac{a}{b}\right) = (-1)^{f(a,b)}\left(\frac{b}{a}\right)$

  for a very simple f

$$f(a,b) = \begin{cases} 0, \text{ if } a \equiv 1 \pmod 4 \text{ or } b \equiv 1 \pmod 4 \\ 1, \text{ if } a \equiv b \equiv 3 \pmod 4 \end{cases}$$

* modulo minor technical caveats; requires a, b odd

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

Jacobi Properties

- Periodicity: $\left(\frac{a}{b}\right) = \left(\frac{a \bmod b}{b}\right)$
- Reciprocity:* $\left(\frac{a}{b}\right) = (-1)^{f(a,b)}\left(\frac{b}{a}\right)$

  for a very simple f

Greatest Common Divisor (GCD) Properties

- Periodicity:
  $\gcd(a, b) = \gcd(a \bmod b, b)$
- Reciprocity: $\gcd(a, b) = \gcd(b, a)$

* modulo minor technical caveats; requires a, b odd

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

<u>Jacobi Properties</u>

- Periodicity: $\left(\frac{a}{b}\right) = \left(\frac{a \bmod b}{b}\right)$
- Reciprocity:* $\left(\frac{a}{b}\right) = (-1)^{f(a,b)}\left(\frac{b}{a}\right)$

  for a very simple f

<u>Greatest Common Divisor (GCD) Properties</u>

- Periodicity:

  $\gcd(a, b) = \gcd(a \bmod b, b)$
- Reciprocity: $\gcd(a, b) = \gcd(b, a)$

Extended Euclidean algorithm solves both these problems!

* modulo minor technical caveats; requires a, b odd

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

- Extended Euclidean recursion: if $a < b$, swap a, b. Else, update $a \leftarrow a \bmod b$.
  - Standard runtime: $O(\log a \log b)$

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

- Extended Euclidean recursion: if $a < b$, swap a, b. Else, update $a \leftarrow a \bmod b$.
  - Standard runtime: $O(\log a \log b)$
  - Schönhage 1971: complicated (and little-known!) divide-and-conquer algorithm that outputs the "transcript" of extended Euclidean in $\tilde{O}(\log a + \log b)$ time

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

- Extended Euclidean recursion: if $a < b$, swap a, b. Else, update $a \leftarrow a \bmod b$.
  - Standard runtime: $O(\log a \log b)$
  - Schönhage 1971: complicated (and little-known!) divide-and-conquer algorithm that outputs the "transcript" of extended Euclidean in $\tilde{O}(\log a + \log b)$ time

Acta Informatica 1, 139—144 (1971)

Schnelle Berechnung von Kettenbruchentwicklungen

A. Schönhage

Eingegangen am 16. September 1970

Summary. A method, given by D. E. Knuth for the computation of the greatest common divisor of two integers u, v and of the continued fraction for u/v is modified in such a way that only $O(n(\lg n)^2(\lg\lg n))$ elementary steps are used for $u, v < 2^n$.

Zusammenfassung. Ein von D. E. Knuth angegebenes Verfahren, für ganze Zahlen u, v den größten gemeinsamen Teiler und den Kettenbruch für u/v zu berechnen, wird so modifiziert, daß für n-stellige Zahlen nur $O(n(\lg n)^2(\lg\lg n))$ elementare Schritte gebraucht werden.

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

- Extended Euclidean recursion: if $a < b$, swap a, b. Else, update $a \leftarrow a \bmod b$.
  - Standard runtime: $O(\log a \log b)$
  - Schönhage 1971: complicated (and little-known!) divide-and-conquer algorithm that outputs the "transcript" of extended Euclidean in $\tilde{O}(\log a + \log b)$ time

Acta Informatica 1, 139—144 (1971)

Schnelle Berechnung von Kettenbruchentwicklungen

A. Schönhage

Eingegangen am 16. September 1970

Summary. A method, given by D. E. Knuth for the computation of the greatest common divisor of two integers u, v and of the continued fraction for u/v is modified in such a way that only $O(n(\lg n)^2(\lg\lg n))$ elementary steps are used for $u, v < 2^n$.

Zusammenfassung. Ein von D. E. Knuth angegebenes Verfahren, für ganze Zahlen u, v den größten gemeinsamen Teiler und den Kettenbruch für u/v zu berechnen, wird so modifiziert, daß für n-stellige Zahlen nur $O(n(\lg n)^2(\lg\lg n))$ elementare Schritte gebraucht werden.

A Unified Approach to HGCD Algorithms for polynomials and integers

Klaus Thull and Chee K. Yap*

Freie Universität Berlin
Fachbereich Mathematik
Arnimallee 2-6
D-1000 Berlin 33
West Germany

March, 1990

Abstract

We present a unified framework for the asymptotically fast Half-GCD (HGCD) algorithms, based on properties of the norm. Two other benefits of our approach are (a) a simplified correctness proof of the polynomial HGCD algorithm and (b) the first explicit integer HGCD algorithm. The integer HGCD algorithm turns out to be rather intricate.

Keywords: Integer GCD, Euclidean algorithm, Polynomial GCD, Half GCD algorithm, efficient algorithm.

MATHEMATICS OF COMPUTATION
Volume 77, Number 261, January 2008, Pages 589–607
S 0025-5718(07)02017-0
Article electronically published on September 12, 2007

ON SCHÖNHAGE'S ALGORITHM AND SUBQUADRATIC INTEGER GCD COMPUTATION

NIELS MÖLLER

Abstract. We describe a new subquadratic left-to-right GCD algorithm, inspired by Schönhage's algorithm for reduction of binary quadratic forms, and compare it to the first subquadratic GCD algorithm discovered by Knuth and Schönhage, and to the binary recursive GCD algorithm of Stehlé and Zimmermann. The new GCD algorithm runs slightly faster than earlier algorithms, and it is much simpler to implement. The key idea is to use a stop condition for HGCD that is based not on the size of the remainders, but on the size of the next difference. This subtle change is sufficient to eliminate the back-up steps that are necessary in all previous subquadratic left-to-right GCD algorithms. The subquadratic GCD algorithms all have the same asymptotic running time, $O(n(\log n)^2 \log\log n)$.

Our Result: Pushing the Space and Depth Down to $\tilde{O}(\log Q)$

Why is There Any Hope for Sublinear Space?

- Recall: to solve period finding when the period is T, need to set up a superposition

$$\sum_{a=1}^{\mathrm{poly}(T)} |a\rangle |f(a)\rangle$$

Why is There Any Hope for Sublinear Space?

- Recall: to solve period finding when the period is T, need to set up a superposition

$$\sum_{a=1}^{\mathsf{poly}(T)} |a\rangle|f(a)\rangle$$

- Even just writing down $|a\rangle$ requires log $\mathsf{poly}(T) = O(\log T)$ qubits!

Why is There Any Hope for Sublinear Space?

- Recall: to solve period finding when the period is T, need to set up a superposition

$$\sum_{a=1}^{\mathsf{poly}(T)} |a\rangle|f(a)\rangle$$

- Even just writing down $|a\rangle$ requires $\log \mathsf{poly}(T) = O(\log T)$ qubits!
- In Shor (and Regev): the period is $\approx N \rightarrow$ stuck at $O(\log N)$ qubits

Why is There Any Hope for Sublinear Space?

- Recall: to solve period finding when the period is T, need to set up a superposition

$$\sum_{a=1}^{\mathsf{poly}(T)} |a\rangle |f(a)\rangle$$

- Even just writing down $|a\rangle$ requires log $\mathsf{poly}(T) = O(\log T)$ qubits!
- In Shor (and Regev): the period is $\approx N \rightarrow$ stuck at $O(\log N)$ qubits
- **<u>Hope 1:</u>** when factoring P^2Q with Jacobi: the period is just $Q \rightarrow O(\log Q)$ qubits could suffice!

Why is There Any Hope for Sublinear Space?

- Goal: compute $\left(\frac{a}{N}\right)$ for $a \leq \mathsf{poly}(Q)$
- How could we compute this without ever writing down all of N quantumly?

Why is There Any Hope for Sublinear Space?

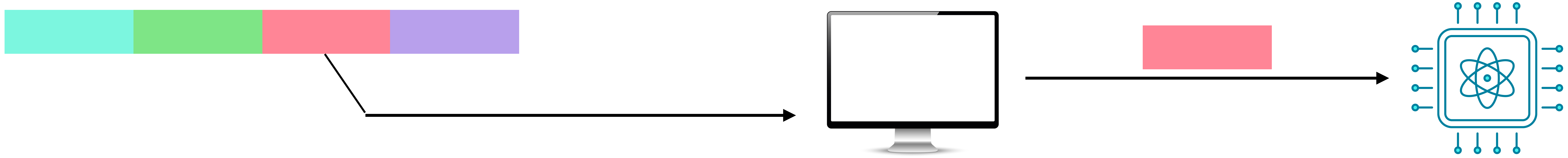
- Goal: compute $\left(\frac{a}{N}\right)$ for $a \leq \mathsf{poly}(Q)$
- How could we compute this without ever writing down all of N quantumly?
- **<u>Hope 2:</u>** N is classically known → could quantumly "stream" through bits of N to save space

Why is There Any Hope for Sublinear Space?

- Goal: compute $\left(\frac{a}{N}\right)$ for $a \leq \text{poly}(Q)$
- How could we compute this without ever writing down all of N quantumly?
- **<u>Hope 2:</u>** N is classically known → could quantumly "stream" through bits of N to save space

Bits of N, split into chunks of size $O(\log Q)$

Quantum computer with $\tilde{O}(\log Q)$ qubits



Classical computer sending instructions to the quantum computer

Computing the Jacobi Symbol

It's all about $N \bmod a$

- Our task: compute $\left(\frac{a}{N}\right)$ for $a \leq \mathsf{poly}(Q)$

Computing the Jacobi Symbol

It's all about $N \bmod a$

- Our task: compute $\left(\frac{a}{N}\right)$ for $a \leq \text{poly}(Q)$
- Recall:

$$\left(\frac{a}{N}\right) = (-1)^{f(a,N)}\left(\frac{N}{a}\right) = (-1)^{f(a,N)}\left(\frac{N \bmod a}{a}\right)$$

Computing the Jacobi Symbol

It's all about $N \bmod a$

- Our task: compute $\left(\frac{a}{N}\right)$ for $a \leq \mathsf{poly}(Q)$
- Recall:

$$\left(\frac{a}{N}\right) = (-1)^{f(a,N)}\left(\frac{N}{a}\right) = (-1)^{f(a,N)}\left(\frac{N \bmod a}{a}\right)$$

- Two step procedure:
  1. Compute $N \bmod a$

Computing the Jacobi Symbol

It's all about $N \bmod a$

- Our task: compute $\left(\frac{a}{N}\right)$ for $a \le \mathsf{poly}(Q)$
- Recall:

$$\left(\frac{a}{N}\right) = (-1)^{f(a,N)}\left(\frac{N}{a}\right) = (-1)^{f(a,N)}\left(\frac{N \bmod a}{a}\right)$$

- Two step procedure:

1. Compute $N \bmod a$
2. Now $(N \bmod a) < a < \mathsf{poly}(Q) \rightarrow$ can finish in $\tilde{O}(\log Q)$ gates/space using Schönhage

Computing the Jacobi Symbol

It's all about $N \bmod a$

- Our task: compute $\left(\frac{a}{N}\right)$ for $a \leq \text{poly}(Q)$
- Recall:

$$\left(\frac{a}{N}\right) = (-1)^{f(a,N)}\left(\frac{N}{a}\right) = (-1)^{f(a,N)}\left(\frac{N \bmod a}{a}\right)$$

- Two step procedure:
    1. Compute $N \bmod a$
    2. Now $(N \bmod a) < a < \text{poly}(Q) \rightarrow$ can finish in $\tilde{O}(\log Q)$ gates/space using Schönhage

The "only" bottleneck: computing $|a\rangle \mapsto |a\rangle|N \bmod a\rangle$

Our Result, Distilled

- Theorem (K**R**VV24): for quantum a and classically known N, we can compute

$$|a\rangle \mapsto |a\rangle|N \bmod a\rangle$$

in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits (enough qubits to write down a)

Our Result, Distilled

- Theorem (K**R**VV24): for quantum a and classically known N, we can compute

$$|a\rangle \mapsto |a\rangle|N \bmod a\rangle$$

in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits (enough qubits to write down a)

- Corollary 1: all of the following can be computed in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits for quantum a and classical N:
  - Jacobi symbol: $\left(\frac{a}{N}\right)$

Our Result, Distilled

- Theorem (K**R**VV24): for quantum a and classically known N, we can compute

$$|a\rangle \mapsto |a\rangle|N \bmod a\rangle$$

in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits (enough qubits to write down a)

- Corollary 1: all of the following can be computed in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits for quantum a and classical N:
  - Jacobi symbol: $\left(\frac{a}{N}\right)$
  - GCD: $\gcd(a, N)$
  - Modular inverse: $a^{-1} \bmod N$ (provided $\gcd(a, N) = 1$)

Our Result, Distilled

- Theorem (K**R**VV24): for quantum a and classically known N, we can compute

$$|a\rangle \mapsto |a\rangle|N \bmod a\rangle$$

in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits (enough qubits to write down a)

- Corollary 1: all of the following can be computed in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits for quantum a and classical N:
  - Jacobi symbol: $\left(\frac{a}{N}\right)$
  - GCD: $\gcd(a, N)$
  - Modular inverse: $a^{-1} \bmod N$ (provided $\gcd(a, N) = 1$)

Open question: other applications of these results?

Our Result, Distilled

- Theorem (K**R**VV24): for quantum a and classically known N, we can compute

$$|a\rangle \mapsto |a\rangle |N \bmod a\rangle$$

in $\tilde{O}(\log N)$ gates (near-linear) and $\tilde{O}(\log a)$ qubits (enough qubits to write down a)

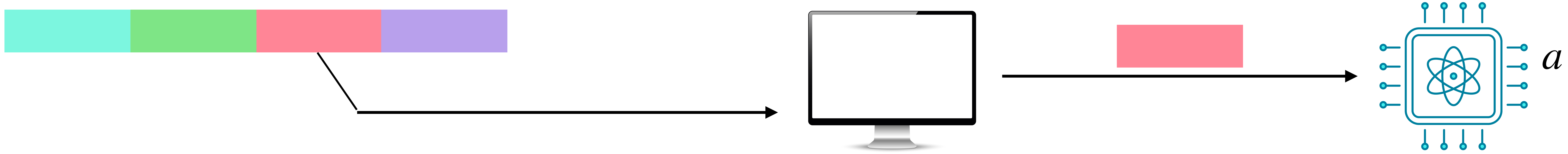
- **Corollary 2: we can factor $N = P^2Q$ in $\tilde{O}(\log N)$ gates and $\tilde{O}(\log Q)$ qubits**
  - **Just need the above theorem for $a \leq \mathsf{poly}(Q)$**

Computing N mod a with Quantum Streaming

Notation: N has n bits, a has $m = O(\log Q)$ bits

Bits of N, split into chunks of size $O(\log Q)$

Quantum computer with $\tilde{O}(\log Q)$ qubits

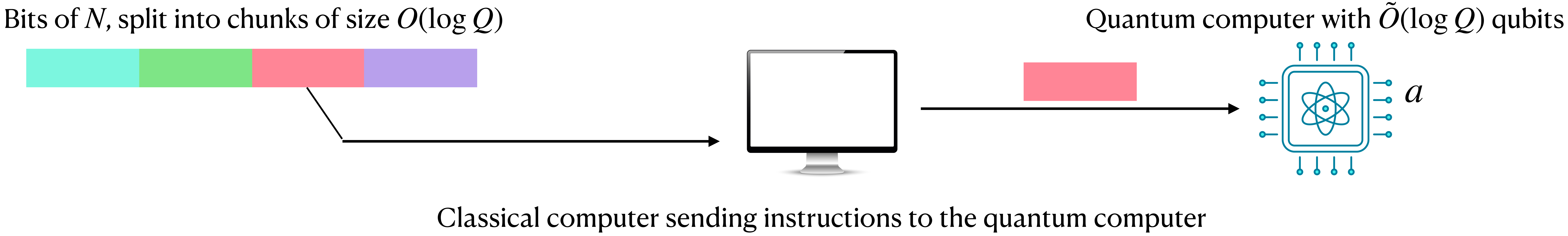


Classical computer sending instructions to the quantum computer

Computing $N \bmod a$ with Quantum Streaming

- Proceeds in $t_{\max} = O(n/m)$ time steps (one for each m-bit chunk of N)

Notation: N has n bits, a has $m = O(\log Q)$ bits



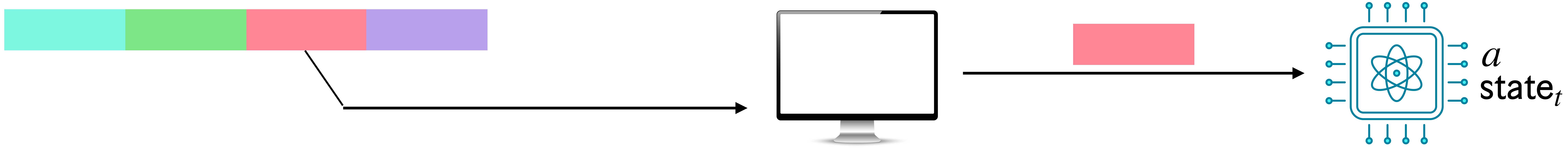
Computing $N \bmod a$ with Quantum Streaming

- Proceeds in $t_{\max} = O(n/m)$ time steps (one for each m-bit chunk of N)
  - After time step t: quantum computer has some state_t

Notation: N has n bits, a has $m = O(\log Q)$ bits

Bits of N, split into chunks of size $O(\log Q)$

Quantum computer with $\tilde{O}(\log Q)$ qubits

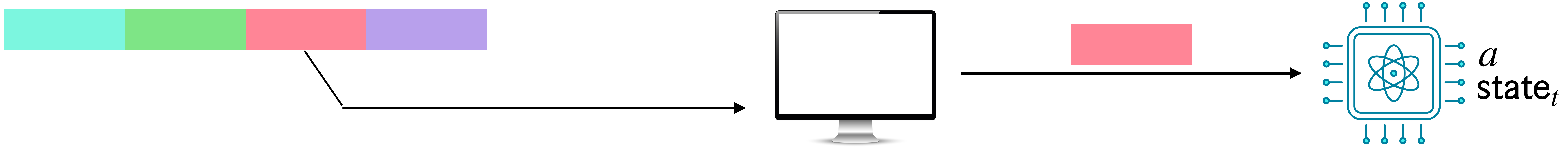


Classical computer sending instructions to the quantum computer

Computing N mod a with Quantum Streaming

- Proceeds in $t_{\max} = O(n/m)$ time steps (one for each m-bit chunk of N)
  - After time step t: quantum computer has some state_t
- Desiderata:
  - Correctness: N mod a is efficiently recoverable from $\mathsf{state}_{t_{\max}}$

Notation: N has n bits, a has $m = O(\log Q)$ bits

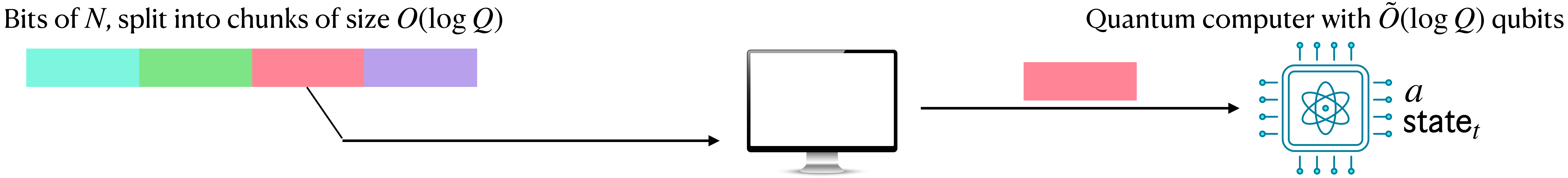


Classical computer sending instructions to the quantum computer

Computing N mod a with Quantum Streaming

- Proceeds in $t_{\max} = O(n/m)$ time steps (one for each m-bit chunk of N)
  - After time step t: quantum computer has some state_t
- Desiderata:
  - Correctness: N mod a is efficiently recoverable from $\mathsf{state}_{t_{\max}}$
  - Compactness: state_t has $O(m)$ bits for all t

Notation: N has n bits, a has $m = O(\log Q)$ bits



Classical computer sending instructions to the quantum computer

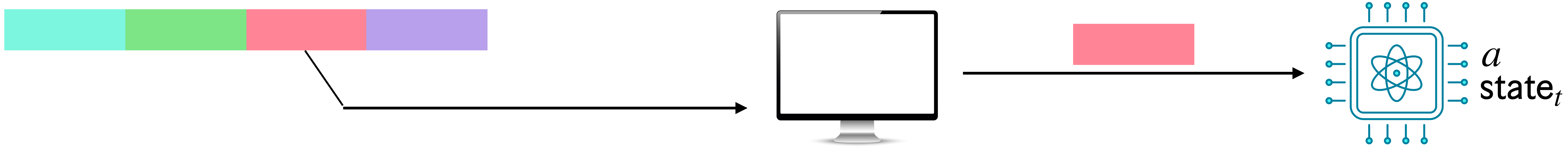
Computing N mod a with Quantum Streaming

- Proceeds in $t_{\max} = O(n/m)$ time steps (one for each m-bit chunk of N)
    - After time step t: quantum computer has some state_t
- Desiderata:
    - Correctness: N mod a is efficiently recoverable from $\mathsf{state}_{t_{\max}}$
    - Compactness: state_t has $O(m)$ bits for all t
    - **Reversibility:** state_{t-1} can be reconstructed (and therefore uncomputed) from state_t

Notation: N has n bits, a has $m = O(\log Q)$ bits

Bits of N, split into chunks of size $O(\log Q)$

Quantum computer with $\tilde{O}(\log Q)$ qubits



Classical computer sending instructions to the quantum computer

Our Construction, Simplified

- At time $t = 0, \ldots, n/m$, let N_t be a multiple of a such that $N \equiv N_t \pmod{2^{mt}}$
  - Equivalently: N_t agrees with N on the mt lowest-order bits

Our Construction, Simplified

- At time $t = 0, \ldots, n/m$, let N_t be a multiple of a such that $N \equiv N_t \pmod{2^{mt}}$
  - Equivalently: N_t agrees with N on the mt lowest-order bits
- N_t is constructible from N_{t-1} with essentially a multiplication of m-bit integers

Our Construction, Simplified

- At time $t = 0, \ldots, n/m$, let N_t be a multiple of a such that $N \equiv N_t \pmod{2^{mt}}$
  - Equivalently: N_t agrees with N on the mt lowest-order bits
- N_t is constructible from N_{t-1} with essentially a multiplication of m-bit integers
- Bits of N_t split into two parts:

Our Construction, Simplified

- At time $t = 0, \ldots, n/m$, let N_t be a multiple of a such that $N \equiv N_t \pmod{2^{mt}}$
  - Equivalently: N_t agrees with N on the mt lowest-order bits
- N_t is constructible from N_{t-1} with essentially a multiplication of m-bit integers
- Bits of N_t split into two parts:
  - tm low-order bits: these match N → classically known → no need to store quantumly

Our Construction, Simplified

- At time $t = 0, \ldots, n/m$, let N_t be a multiple of a such that $N \equiv N_t \pmod{2^{mt}}$
  - Equivalently: N_t agrees with N on the mt lowest-order bits
- N_t is constructible from N_{t-1} with essentially a multiplication of m-bit integers
- Bits of N_t split into two parts:
  - tm low-order bits: these match $N \rightarrow$ classically known $\rightarrow$ no need to store quantumly
  - Higher-order bits: this must be held quantumly, and is state_t

Our Construction, Simplified

- At time $t = 0, \ldots, n/m$, let N_t be a multiple of a such that $N \equiv N_t \pmod{2^{mt}}$
  - Equivalently: N_t agrees with N on the mt lowest-order bits
- N_t is constructible from N_{t-1} with essentially a multiplication of m-bit integers
- Bits of N_t split into two parts:
  - tm low-order bits: these match N → classically known → no need to store quantumly
  - Higher-order bits: this must be held quantumly, and is state_t
- It turns out that the final state $\mathsf{state}_{n/m}$ suffices to reconstruct N mod a

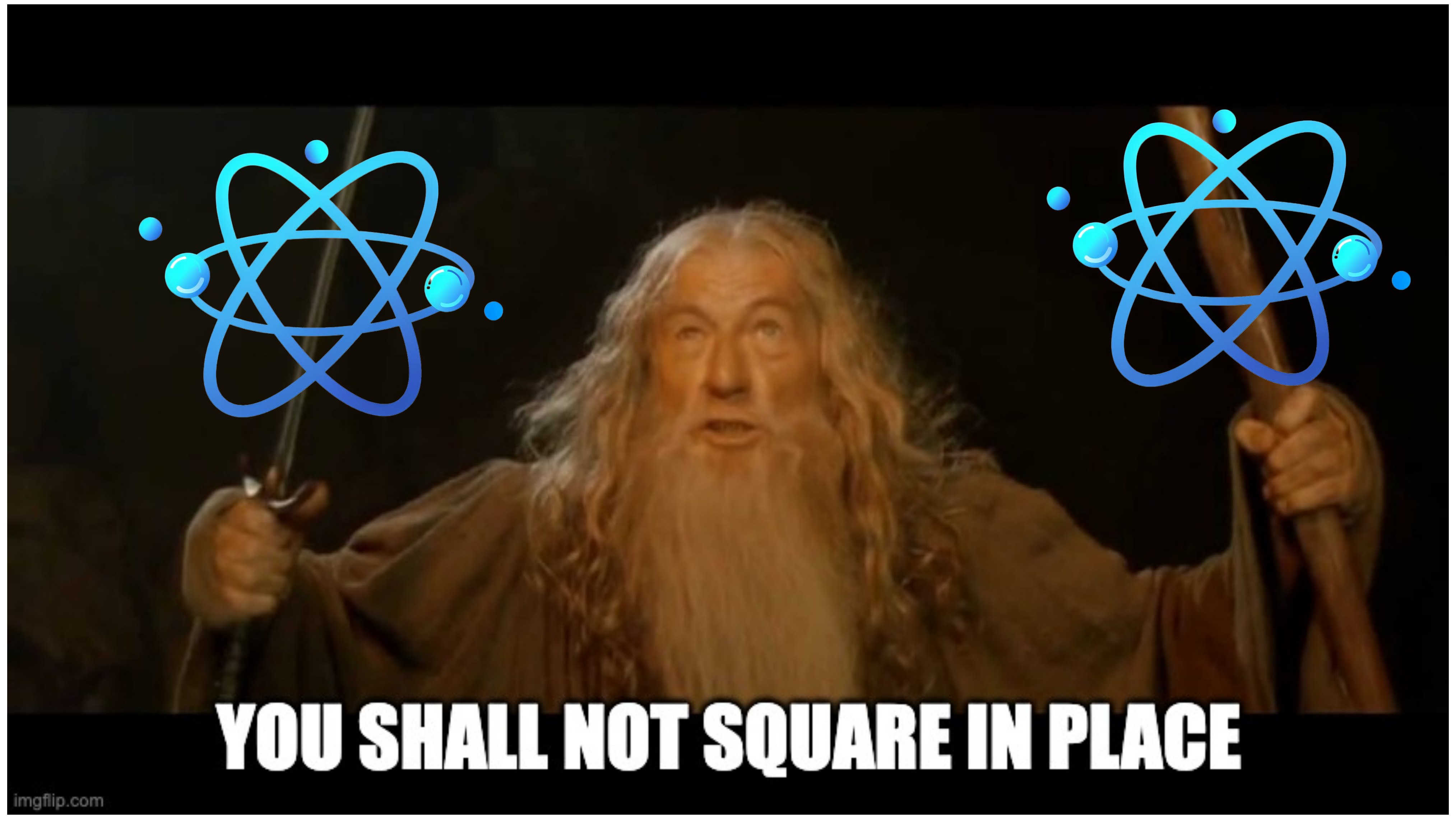
Our Result: Improving Space Complexity of Regev

Regev's 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!

Regev's 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. $-1, 1$



Regev's 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*:

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

Regev's 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.

Main Idea: Fibonacci Exponentiation

- What if we used two accumulators?

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

Main Idea: Fibonacci Exponentiation

- What if we used two accumulators?

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

- Observation by Kaliski 2017:

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

Main Idea: Fibonacci Exponentiation

- What if we used two accumulators?

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

- 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!

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*