The Jacobi Factoring Circuit

Classically Hard Factoring in Sublinear Quantum Space and Depth

Gregory D. Kahanamoku-Meyer*, Seyoon Ragavan*, Vinod Vaikuntanathan*, Katherine Van Kirk†
*MIT, †Harvard




Integer Factorisation

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

Classical Factoring: A Crash Course

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

<u>Factoring general integers</u>

- Brute force: $\exp(O(n))$

Classical Factoring: A Crash Course

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

<u>Factoring general integers</u>

- Brute force: $\exp(O(n))$
- Quadratic sieve (many works from Fermat to Pomerance '82): $\exp(\tilde{O}(n^{1/2}))$

Classical Factoring: A Crash Course

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

Factoring general integers

- Brute force: $\exp(O(n))$
- Quadratic sieve (many works from Fermat to Pomerance '82): $\exp(\tilde{O}(n^{1/2}))$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $\exp(\tilde{O}(n^{1/3}))$

Classical Factoring: A Crash Course

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

<u>Factoring general integers</u>

- Brute force: $\exp(O(n))$
- Quadratic sieve (many works from Fermat to Pomerance '82): $\exp(\tilde{O}(n^{1/2}))$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $\exp(\tilde{O}(n^{1/3}))$

<u>Factoring special-form integers</u>

Classical Factoring: A Crash Course

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

Factoring general integers

- Brute force: $\exp(O(n))$
- Quadratic sieve (many works from Fermat to Pomerance '82): $\exp(\tilde{O}(n^{1/2}))$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $\exp(\tilde{O}(n^{1/3}))$

Factoring special-form integers

- Lenstra ECM ('87): $\exp\left(\tilde{O}((\log P)^{1/2})\right)$ where P is a factor of N

Classical Factoring: A Crash Course

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

Factoring general integers

- Brute force: $\exp(O(n))$
- Quadratic sieve (many works from Fermat to Pomerance '82): $\exp(\tilde{O}(n^{1/2}))$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $\exp(\tilde{O}(n^{1/3}))$

Factoring special-form integers

- Lenstra ECM ('87): $\exp\left(\tilde{O}((\log P)^{1/2})\right)$ where P is a factor of N
- Boneh, Durfee, Howgrave-Graham ('99): if $N = P^r Q$ and $r \geq \log P$, can factor in $\mathsf{poly}(n)$ time

Classical Factoring: A Crash Course

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

Factoring general integers

- Brute force: $\exp(O(n))$
- Quadratic sieve (many works from Fermat to Pomerance '82): $\exp(\tilde{O}(n^{1/2}))$
- General number field sieve (Pollard '88; Buhler-Lenstra-Pomerance '93): $\exp(\tilde{O}(n^{1/3}))$

Factoring special-form integers

- Lenstra ECM ('87): $\exp\left(\tilde{O}((\log P)^{1/2})\right)$ where P is a factor of N
- Boneh, Durfee, Howgrave-Graham ('99): if $N = P^r Q$ and $r \geq \log P$, can factor in $\mathsf{poly}(n)$ time

For more: see Pomerance's survey "A Tale of Two Sieves"!

Integer Factorisation

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

- Fastest classical algorithm for general N: $\exp\left(\tilde{O}(n^{1/3})\right)$ time

Integer Factorisation

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

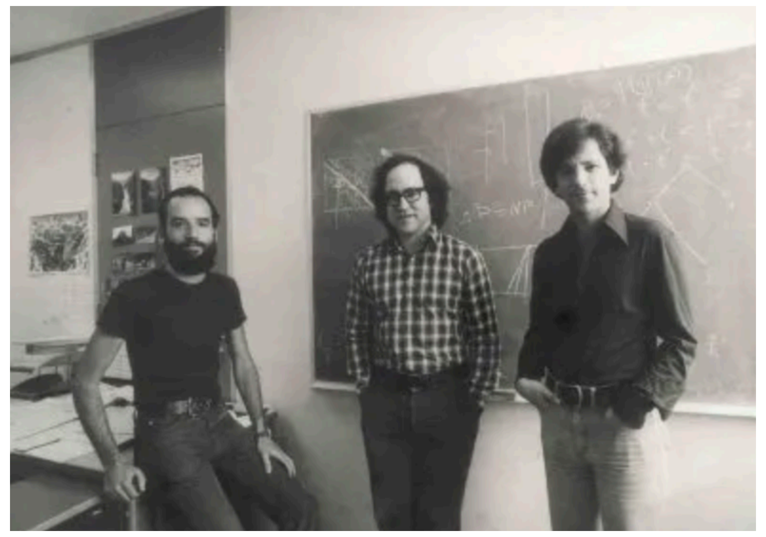
- Fastest classical algorithm for general N: $\exp\left(\tilde{O}(n^{1/3})\right)$ time
- **Quantum algorithms:** $\mathsf{poly}(n)$ **time!** (Shor 1994)

Implication 1: Breaking Cryptography

- RSA public-key cryptography:

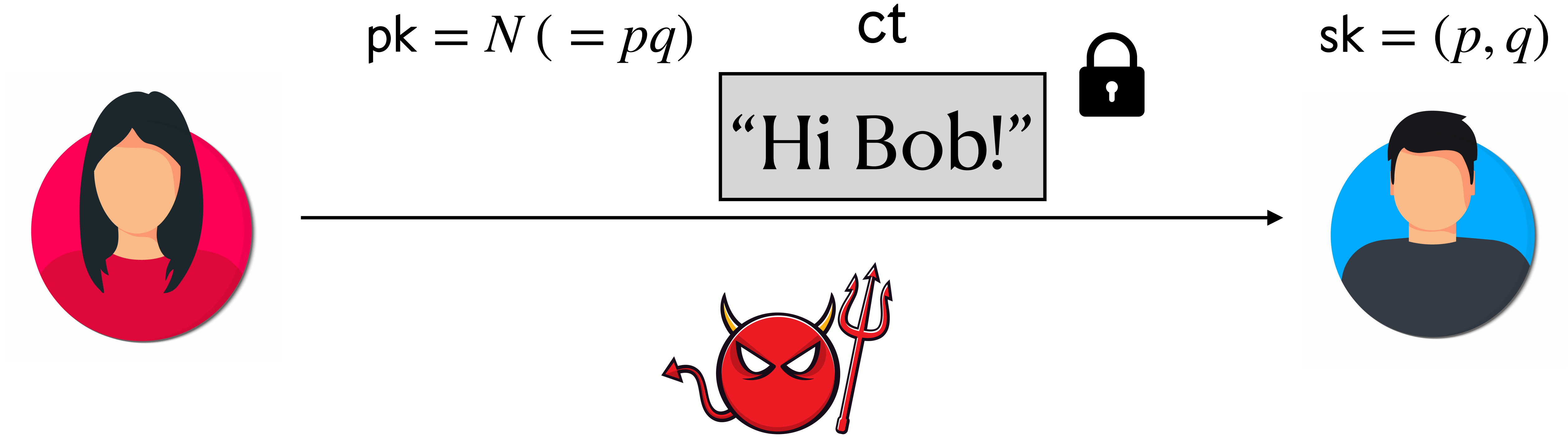
pk $= N\ (= pq)$

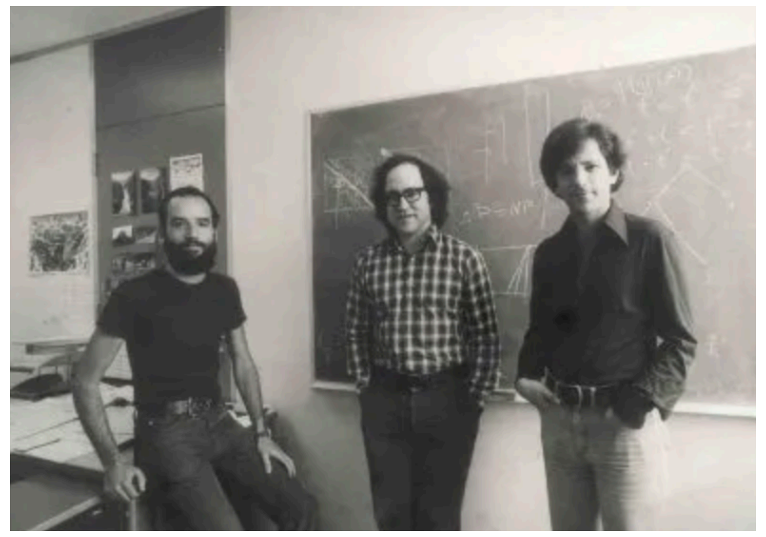
sk $= (p, q)$



Implication 1: Breaking Cryptography

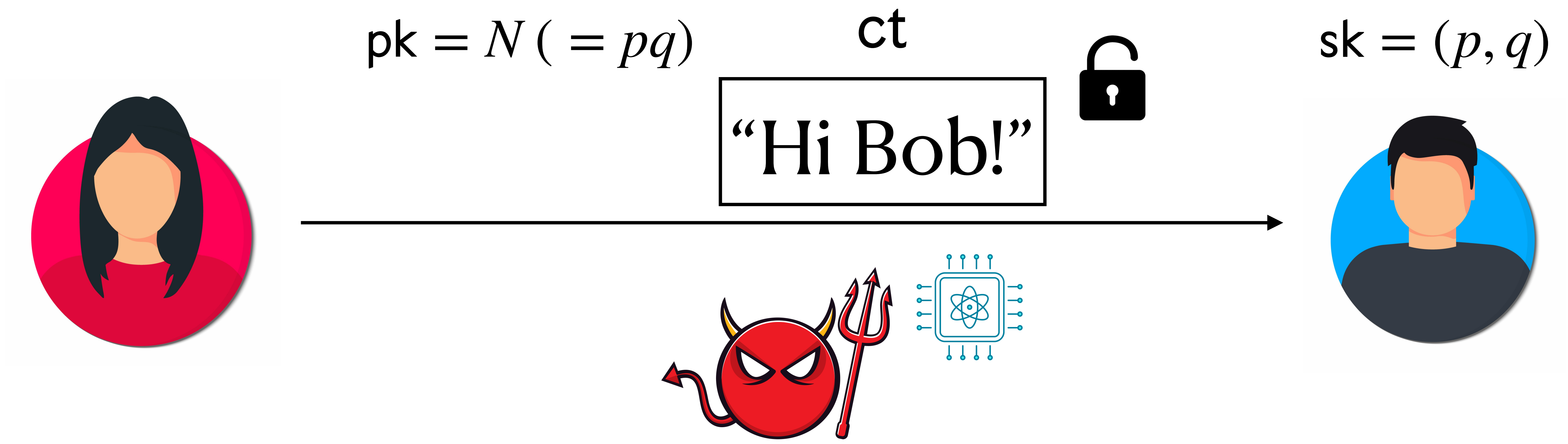
- RSA public-key cryptography:

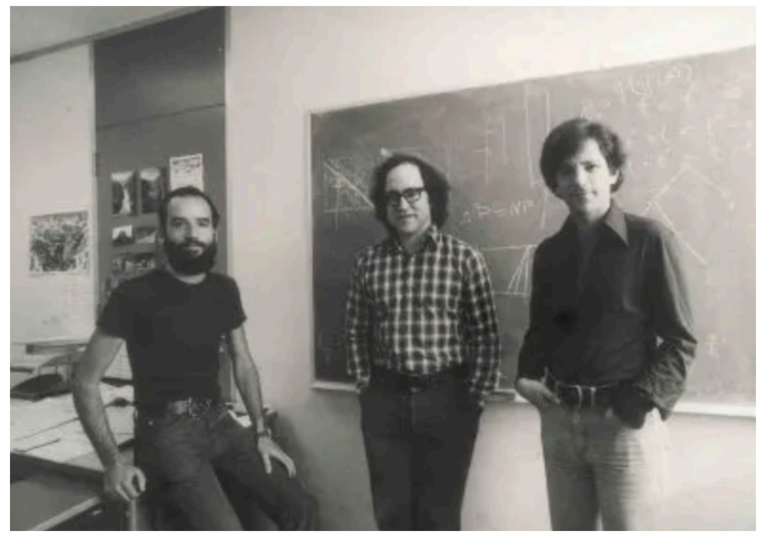




Implication 1: Breaking Cryptography

- RSA public-key cryptography:



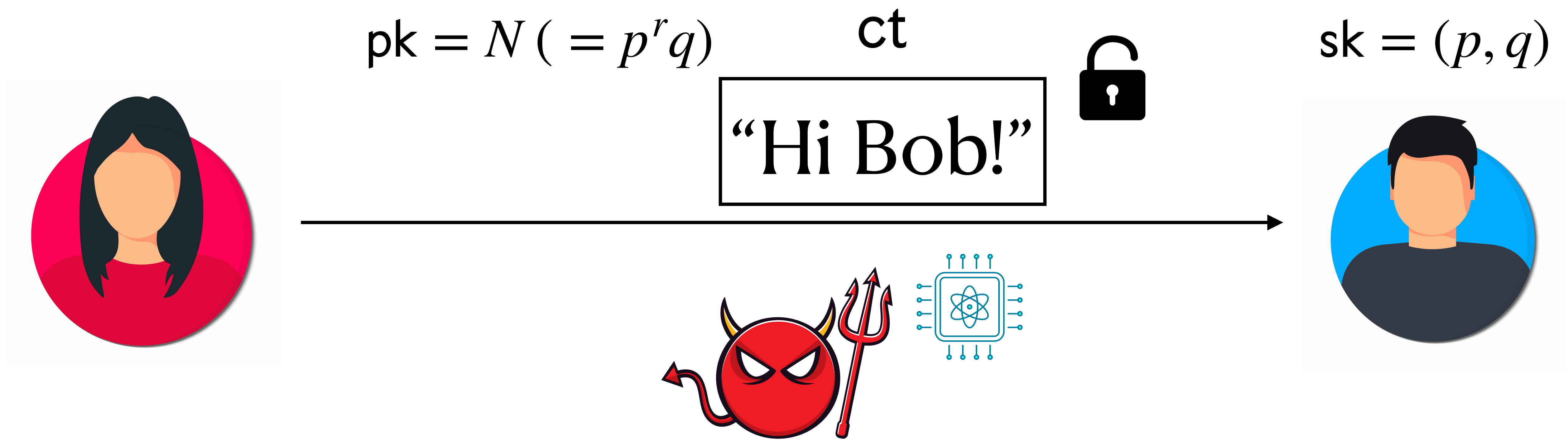


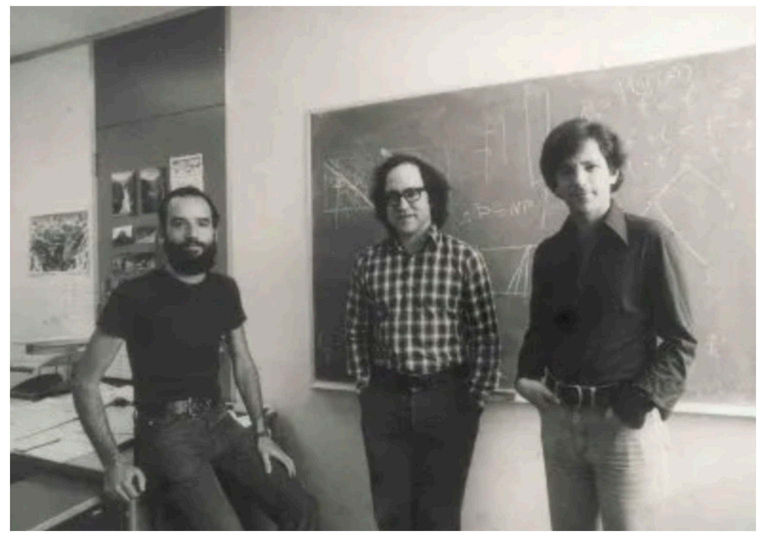
- **Completely broken if the eavesdropper has a large quantum computer (learn Bob's secret key by factoring $N = pq$)**

Implication 1: Breaking Cryptography

- Okamoto-Uchiyama ($r = 2$) or Takagi ($r > 2$) cryptography:

Goal: faster decryption than RSA



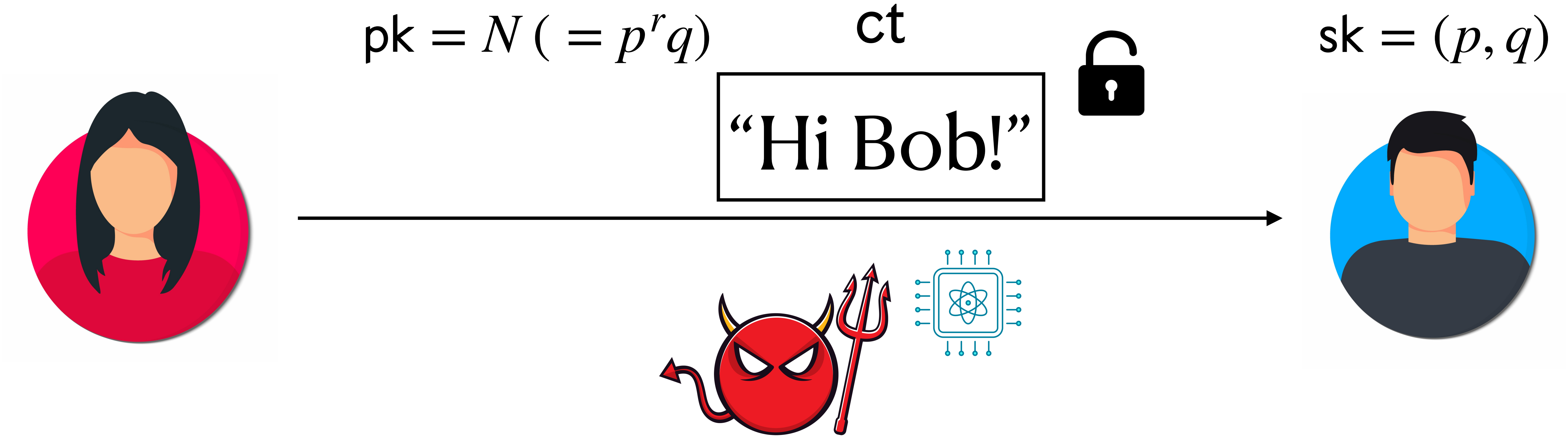


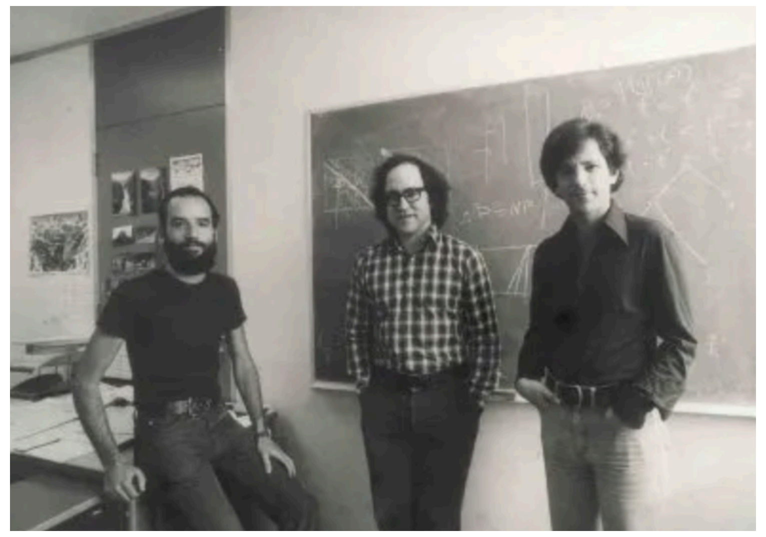
- **Completely broken if the eavesdropper has a large quantum computer (learn Bob's secret key by factoring $N = p^r q$)**

Implication 1: Breaking Cryptography

- Okamoto-Uchiyama ($r = 2$) or Takagi ($r > 2$) cryptography:

Goal: faster decryption than RSA

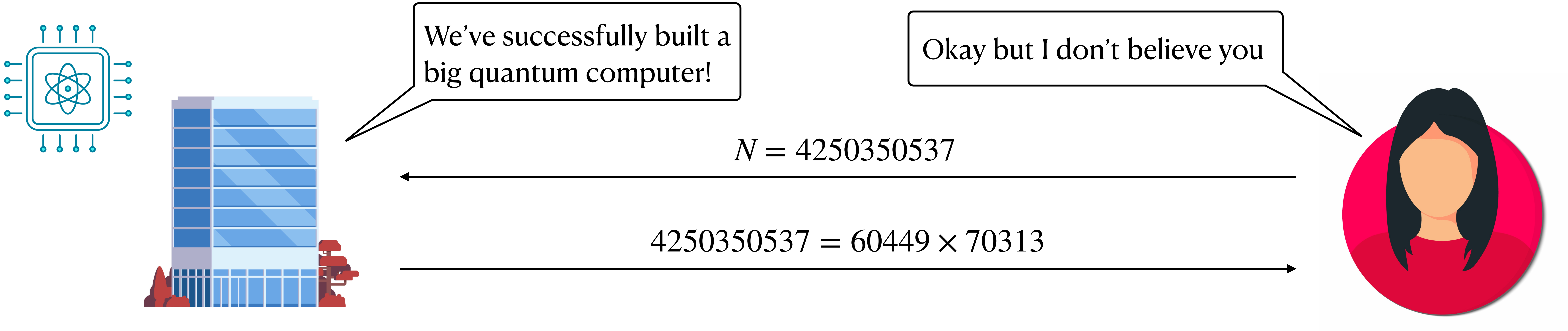




- **Completely broken if the eavesdropper has a large quantum computer (learn Bob's secret key by factoring $N = p^r q$)**

Coming up: even better quantum circuits for factoring $p^r q$ ($r > 1$)

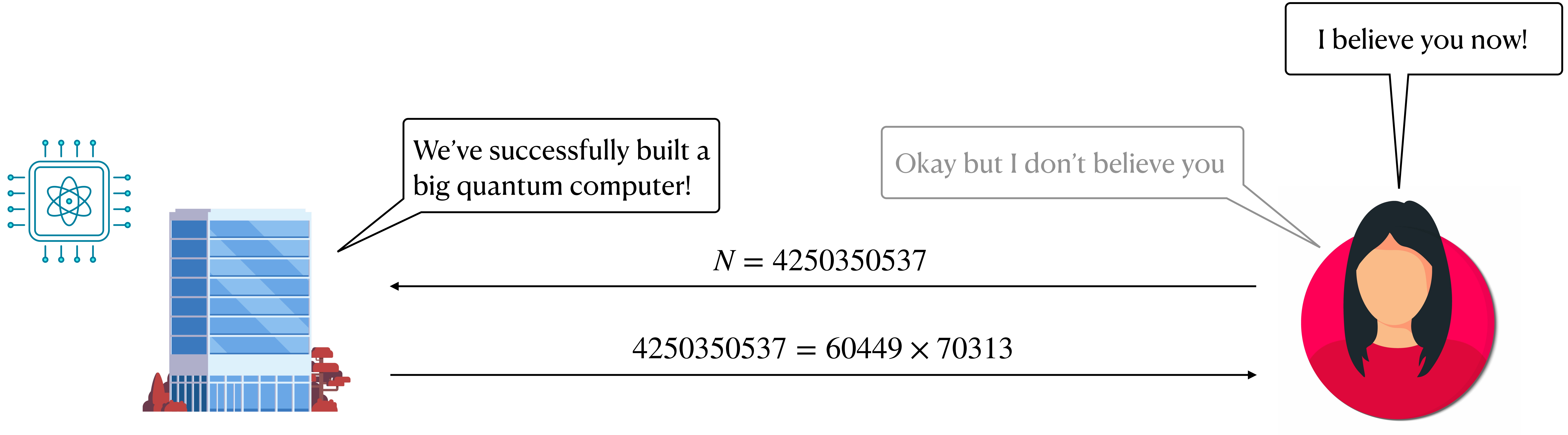
Implication 2: Proofs of Quantumness



Q: How can XYZABC Labs convince Alice that they really do have a large quantum computer?

One answer: By factoring a large integer of Alice's choice!

Implication 2: Proofs of Quantumness



Q: How can XYZABC Labs convince Alice that they really do have a large quantum computer?

One answer: By factoring a large integer of Alice's choice!

The Current State of Affairs

- Known for 30 years: if we had a large-scale quantum computer, we could verifiably demonstrate quantum advantage (by factoring large integers)
- **Slight catch: no-one has managed to build a large-scale quantum computer yet**

The Current State of Affairs

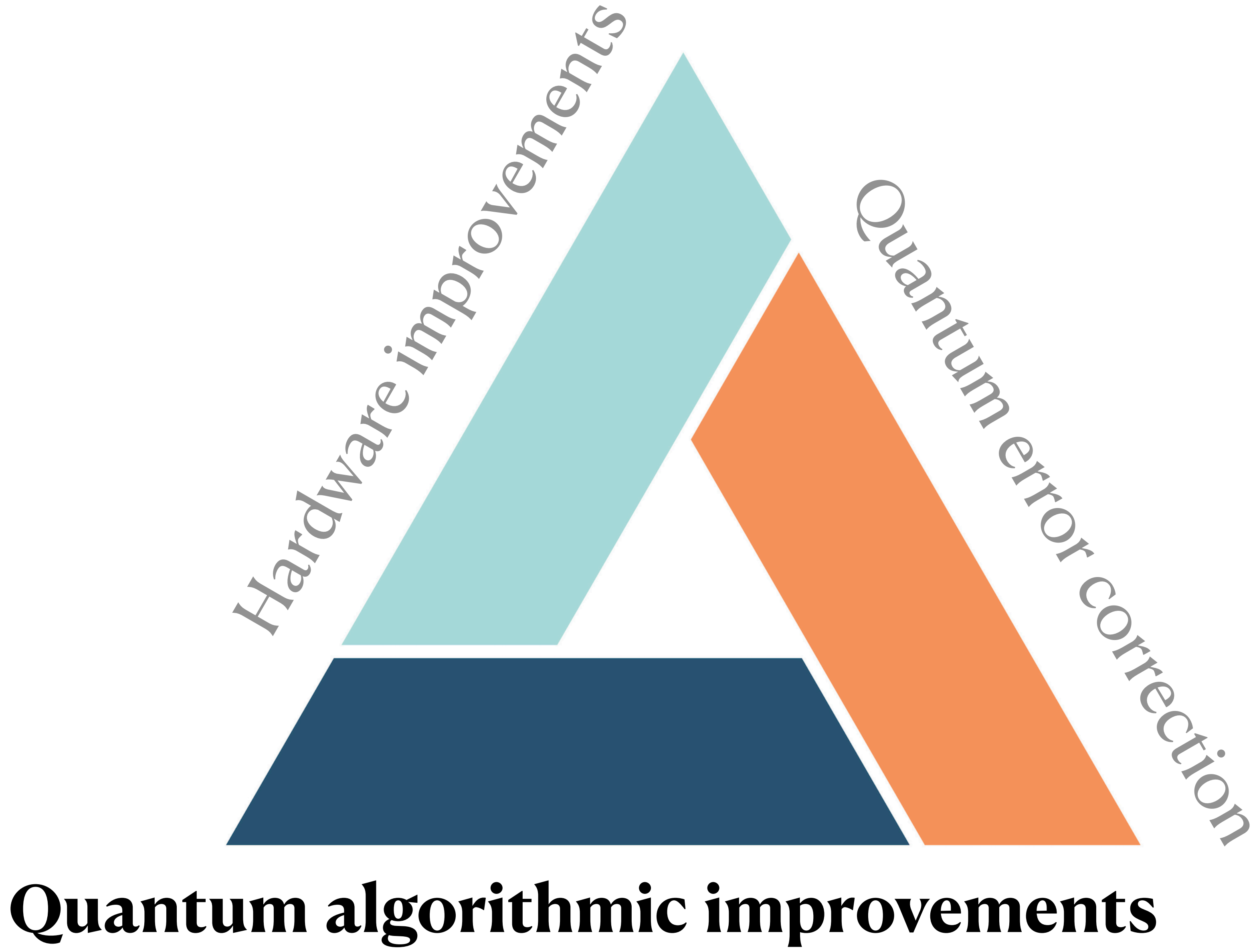
- Known for 30 years: if we had a large-scale quantum computer, we could verifiably demonstrate quantum advantage (by factoring large integers)
- **Slight catch: no-one has managed to build a large-scale quantum computer yet**

*"Our qubits are constantly trying to fall apart…
It's like you're trying to write in the sand and
the wind is blowing it away."*

The Current State of Affairs

- Known for 30 years: if we had a large-scale quantum computer, we could verifiably demonstrate quantum advantage (by factoring large integers)
- **Slight catch: no-one has managed to build a large-scale quantum computer yet**

"Our qubits are constantly trying to fall apart... It's like you're trying to write in the sand and the wind is blowing it away."



Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$

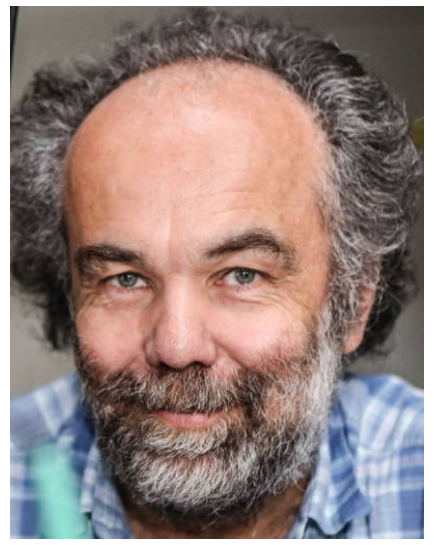
n is the number of bits in the input N

$\tilde{O}(\cdot)$ hides constant and poly$(\log n)$ factors

All results in this talk are using fast integer multiplication (multiply n-bit integers in $\tilde{O}(n)$ time)

Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$



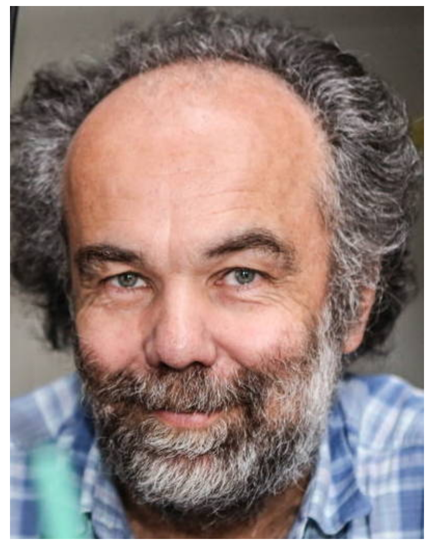
Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$

Any input: would break RSA cryptography, and suffice as a proof of quantumness

$N = P^2Q$: only suffices as a proof of quantumness



Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$




Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$

Note: KCVY is not a factoring algorithm!

- 3-round interactive proof of quantumness (building on previous work by BCMVV18) assuming factoring is classically hard




Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$

Note: KCVY is not a factoring algorithm!

- 3-round interactive proof of quantumness (building on previous work by BCMVV18) assuming factoring is classically hard
- Pro: easier than actually factoring




Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$

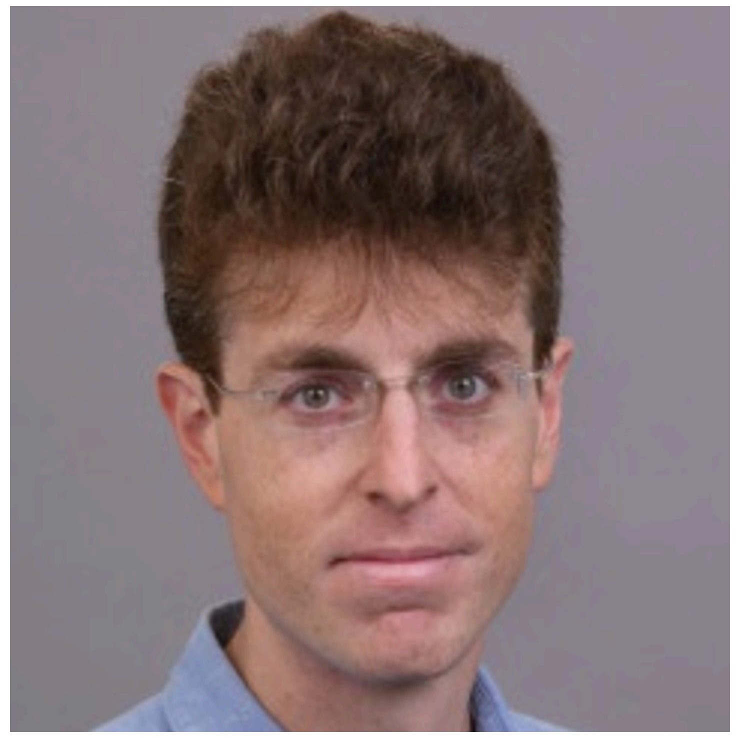
Note: KCVY is not a factoring algorithm!

- 3-round interactive proof of quantumness (building on previous work by BCMVV18) assuming factoring is classically hard
- Pro: easier than actually factoring
- Con: quantum prover needs to store state between rounds




Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$
Regev (2023)	Any	$\tilde{O}(n^{1.5})$	$O(n^{1.5})$	$\tilde{O}(n^{0.5})$



Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$
Regev (2023)	Any	$\tilde{O}(n^{1.5})$	$O(n^{1.5})$	$\tilde{O}(n^{0.5})$
RV (2024)	Any	$\tilde{O}(n^{1.5})$	$\tilde{O}(n)$	$\tilde{O}(n^{0.5})$

Proofs of Quantumness from Factoring

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$
Regev (2023)	Any	$\tilde{O}(n^{1.5})$	$O(n^{1.5})$	$\tilde{O}(n^{0.5})$
RV (2024)	Any	$\tilde{O}(n^{1.5})$	$\tilde{O}(n)$	$\tilde{O}(n^{0.5})$
K**R**VV (2024)	$N = P^2Q\ (Q < 2^m)$	$\tilde{O}(n)$	$\tilde{O}(m)$	$\tilde{O}(n/m + m)$

This work: LPDS with space and depth proportional to $\log Q$ **rather than** $\log N$




Setting Parameters

How should we set $m = \log Q$?

- Balancing act:
  - If Q is too large: our quantum circuit is no better than the LPDS12 circuit

Setting Parameters

How should we set $m = \log Q$?

- Balancing act:
  - If Q is too large: our quantum circuit is no better than the LPDS12 circuit
  - If Q is too small: classical algorithms could exploit this structure to run faster than general-purpose classical factoring algorithms

Setting Parameters

How should we set $m = \log Q$?

- Balancing act:
  - If Q is too large: our quantum circuit is no better than the LPDS12 circuit
  - If Q is too small: classical algorithms could exploit this structure to run faster than general-purpose classical factoring algorithms
    - General NFS: $\exp(\tilde{O}(n^{1/3}))$
    - Lenstra elliptic curve method/Mulder '24: $\exp(\tilde{O}(m^{1/2}))$

Setting Parameters

How should we set $m = \log Q$?

- Balancing act:
  - If Q is too large: our quantum circuit is no better than the LPDS12 circuit
  - If Q is too small: classical algorithms could exploit this structure to run faster than general-purpose classical factoring algorithms
    - General NFS: $\exp(\tilde{O}(n^{1/3}))$
    - Lenstra elliptic curve method/Mulder '24: $\exp(\tilde{O}(m^{1/2}))$
- Sweet spot: $m = \tilde{O}(n^{2/3}) \rightarrow$ gates $\tilde{O}(n)$, space $\tilde{O}(n^{2/3})$, depth $\tilde{O}(n^{2/3})$

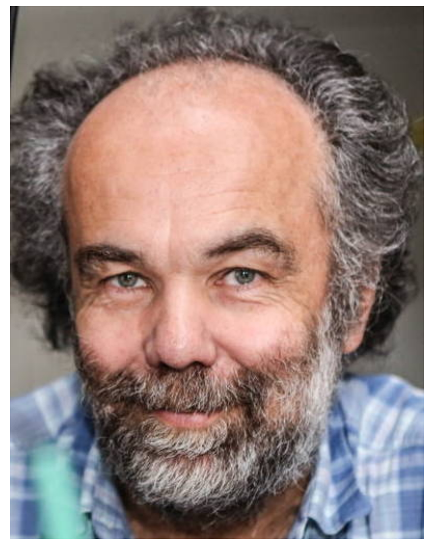
Our Result

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$
Regev (2023)	Any	$\tilde{O}(n^{1.5})$	$O(n^{1.5})$	$\tilde{O}(n^{0.5})$
RV (2024)	Any	$\tilde{O}(n^{1.5})$	$\tilde{O}(n)$	$\tilde{O}(n^{0.5})$
K**R**VV (2024)	$N = P^2Q \ (Q < 2^{n^{2/3}})$	$\tilde{O}(n)$	$\tilde{O}(n^{2/3})$	$\tilde{O}(n^{2/3})$

An algorithm that factors special-form integers (that are still classically as hard as RSA integers to factor) in <u>sublinear</u> space and depth!

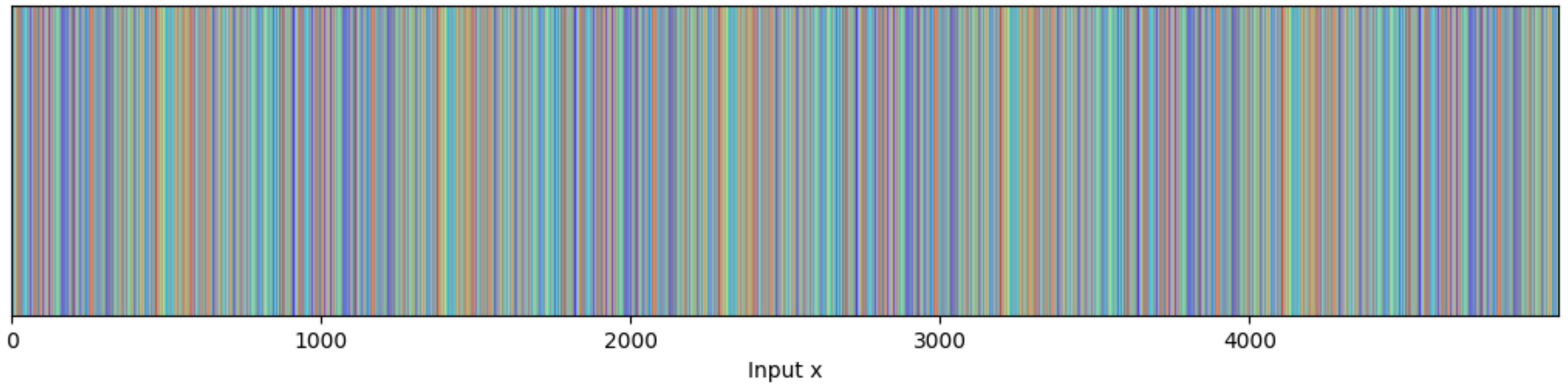



Factoring P^2Q with LPDS12: A Sketch



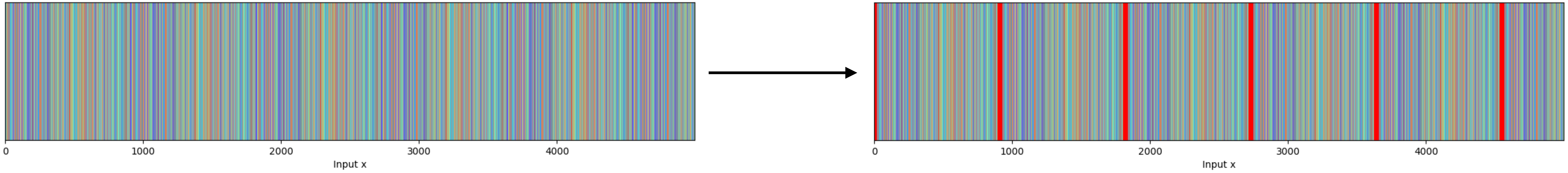
Preliminary: Quantum Period Finding

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



Preliminary: Quantum Period Finding

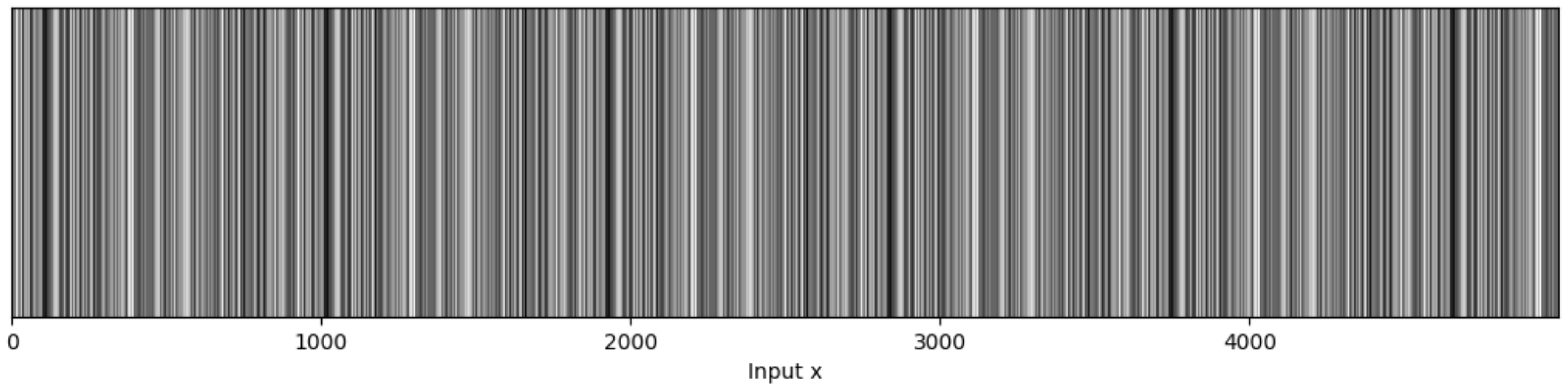
- Strictly periodic function $f : \mathbb{Z} \to \mathbb{Z}$ with unknown period T
    - $x \equiv y \pmod{T} \Leftrightarrow f(x) = f(y)$
- Informal theorem statement: can quantumly recover a **uniformly random** multiple of $1/T$ (and hence T itself) using essentially only the gates/space needed to compute $f(x)$ for $|x| \leq \mathsf{poly}(T)$



Preliminary: General Quantum Period Finding

Hales-Hallgren '98, May-Schlieper '22

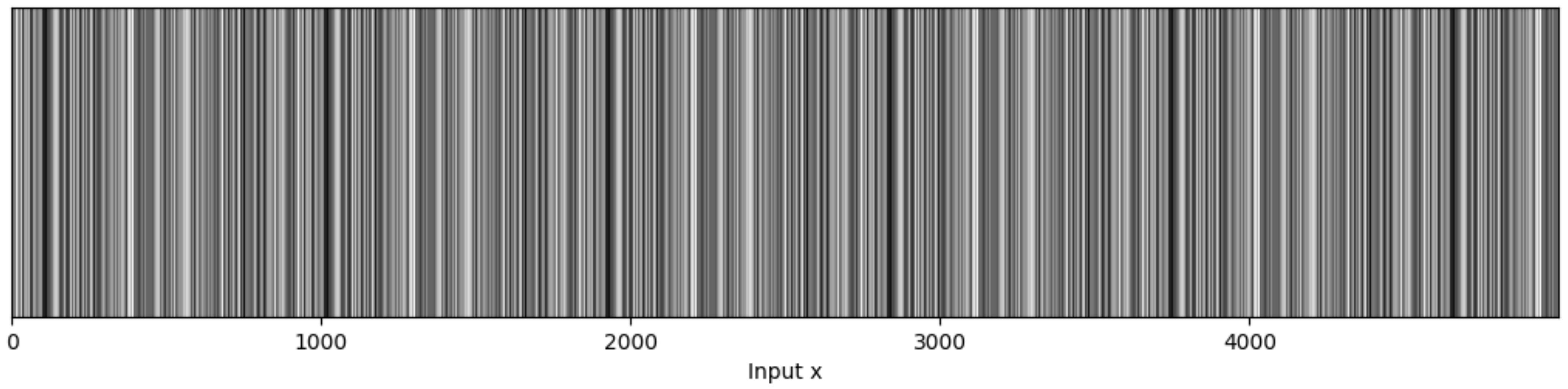
- ~~Strictly~~ periodic function $f : \mathbb{Z} \to \{-1,1\}$ with unknown period T
  - $x \equiv y \pmod{T} \Rightarrow f(x) = f(y)$



Preliminary: General Quantum Period Finding

Hales-Hallgren '98, May-Schlieper '22

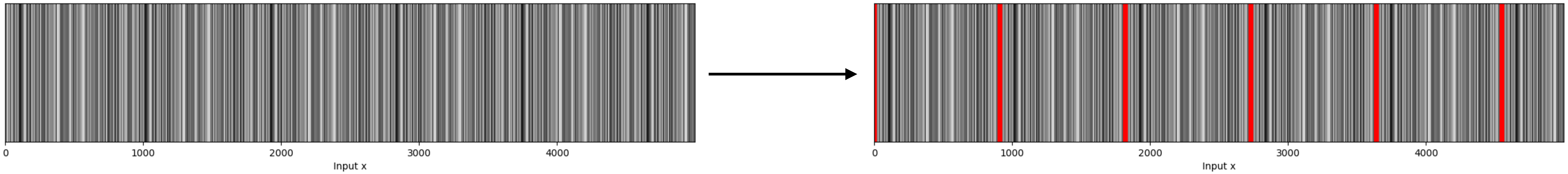
- ~~Strictly~~ periodic function $f : \mathbb{Z} \to \{-1,1\}$ with unknown period T
  - $x \equiv y \pmod{T} \Rightarrow f(x) = f(y)$
- Linearity of the Fourier transform → the same algorithm still outputs a **(not necessarily uniform) random** multiple of $1/T$



Preliminary: General Quantum Period Finding

Hales-Hallgren '98, May-Schlieper '22

- ~~Strictly~~ periodic function $f : \mathbb{Z} \to \{-1,1\}$ with unknown period T
  - $x \equiv y \pmod{T} \Rightarrow f(x) = f(y)$
- Linearity of the Fourier transform → the same algorithm still outputs a **(not necessarily uniform) random** multiple of $1/T$
- Informal theorem statement: for "reasonable" f, this is still sufficient to recover T



What is a Reasonable f?

Hales-Hallgren '98, May-Schlieper '22

- Approximate probability of obtaining $\frac{a}{T}$, for $a \in [T]$):

$$\left|\hat{f}(a)\right|^2 = \frac{1}{T}\left|\sum_{x=0}^{T-1} f(x) \cdot \exp\left(\frac{2\pi i a x}{T}\right)\right|^2$$

What is a Reasonable f?

Hales-Hallgren '98, May-Schlieper '22

- Approximate probability of obtaining $\frac{a}{T}$, for $a \in [T]$):

$$\left|\hat{f}(a)\right|^2 = \frac{1}{T}\left|\sum_{x=0}^{T-1} f(x) \cdot \exp\left(\frac{2\pi i a x}{T}\right)\right|^2$$

- Obvious failure mode: when $\hat{f}$ concentrates on multiples of some divisor $d > 1$ of T (equivalently, f has a smaller period T/d)

What is a Reasonable f?

Hales-Hallgren '98, May-Schlieper '22

- Approximate probability of obtaining $\frac{a}{T}$, for $a \in [T]$):

$$\left|\hat{f}(a)\right|^2 = \frac{1}{T}\left|\sum_{x=0}^{T-1} f(x) \cdot \exp\left(\frac{2\pi i a x}{T}\right)\right|^2$$

- Obvious failure mode: when $\hat{f}$ concentrates on multiples of some divisor $d > 1$ of T (equivalently, f has a smaller period T/d)
- As long as this is not the case, we can succeed after taking enough samples!

What is a Reasonable f?

Hales-Hallgren '98, May-Schlieper '22

- Approximate probability of obtaining $\frac{a}{T}$, for $a \in [T]$):

$$\left|\hat{f}(a)\right|^2 = \frac{1}{T}\left|\sum_{x=0}^{T-1} f(x) \cdot \exp\left(\frac{2\pi i a x}{T}\right)\right|^2$$

- Obvious failure mode: when $\hat{f}$ concentrates on multiples of some divisor $d > 1$ of T (equivalently, f has a smaller period T/d)
- As long as this is not the case, we can succeed after taking enough samples!
  - Example: random periodic f (by Chernoff bound)

What is a Reasonable f?

Hales-Hallgren '98, May-Schlieper '22

- Approximate probability of obtaining $\frac{a}{T}$, for $a \in [T]$):

$$\left|\hat{f}(a)\right|^2 = \frac{1}{T}\left|\sum_{x=0}^{T-1} f(x)\cdot\exp\left(\frac{2\pi i a x}{T}\right)\right|^2$$

- Obvious failure mode: when $\hat{f}$ concentrates on multiples of some divisor $d > 1$ of T (equivalently, f has a smaller period T/d)
- As long as this is not the case, we can succeed after taking enough samples!
  - Example: random periodic f (by Chernoff bound)
- Even better, we can recover T from one sample if $\hat{f}$ concentrates on values a such that $\gcd(a, T) = 1$

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 $N = p_1^{\alpha_1} \ldots p_r^{\alpha_r}$, define:

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

- 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 $\tilde{O}(\log N)$

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

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

* modulo minor technical caveats; requires a, b odd

Algorithms Computing the Jacobi Symbol

ft. Euclid, 2000 years ago

Jacobi Properties

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

  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

<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: $\gcd(a, b) = \gcd(b, a)$

* 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: $\gcd(a, b) = \gcd(b, a)$

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, $O(n(\log n)^2 \log\log n)$.

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 $N = P^2Q$?

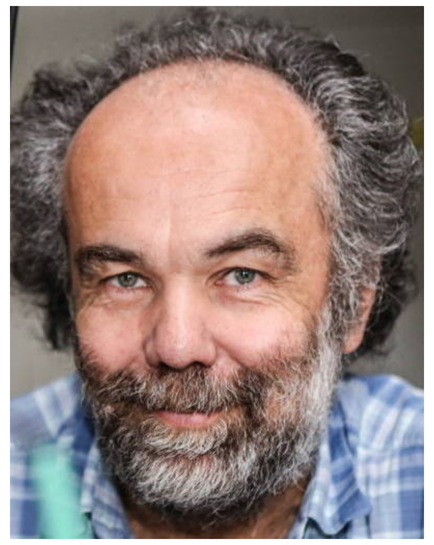
$$\left(\frac{a}{N}\right) = \left(\frac{a}{P}\right)^2\left(\frac{a}{Q}\right) = \left(\frac{a}{Q}\right)$$, which is periodic* with period Q!

* 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)

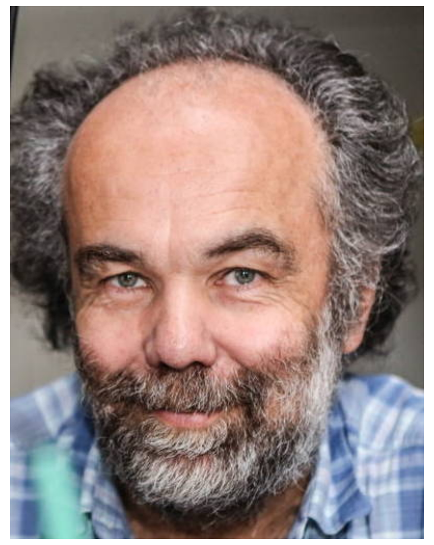
- We know $\left(\frac{a}{N}\right)$ is periodic with period Q!
- So quantum period finding → recover Q (and hence P)



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)
- Gate complexity: cost of computing $\left(\frac{a}{N}\right)$ for $a \leq \mathrm{poly}(Q)$, which is $\tilde{O}(\log N)$

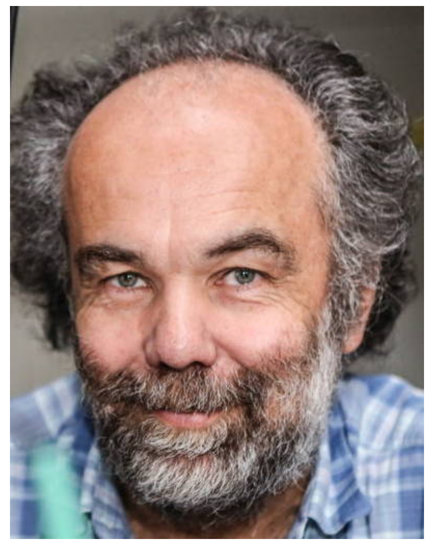


Quantumly Factoring $N = P^2Q$

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)
- Gate complexity: cost of computing $\left(\frac{a}{N}\right)$ for $a \leq \text{poly}(Q)$, which is $\tilde{O}(\log N)$
- Space and depth (if naively implemented): also $\tilde{O}(\log N)$



Is the Jacobi Function "Reasonable"?

- Generalised quantum period finding: we recover $\frac{a}{Q}$ with probability

$$\left|\hat{f}(a)\right|^2 = \frac{1}{Q}\left|\sum_{x=0}^{Q-1}\left(\frac{x}{Q}\right)\cdot\exp\left(\frac{2\pi i a x}{Q}\right)\right|^2$$

Is the Jacobi Function "Reasonable"?

- Generalised quantum period finding: we recover $\frac{a}{Q}$ with probability

$$\left|\hat{f}(a)\right|^2 = \frac{1}{Q}\left|\sum_{x=0}^{Q-1}\left(\frac{x}{Q}\right)\cdot\exp\left(\frac{2\pi iax}{Q}\right)\right|^2$$

- Typically (and in Shor's factoring algorithm): may need > 1 samples (runs of the quantum circuit) to recover the period

Is the Jacobi Function "Reasonable"?

- Generalised quantum period finding: we recover $\frac{a}{Q}$ with probability

$$\left|\hat{f}(a)\right|^2 = \frac{1}{Q}\left|\sum_{x=0}^{Q-1}\left(\frac{x}{Q}\right)\cdot\exp\left(\frac{2\pi i a x}{Q}\right)\right|^2$$

- Typically (and in Shor's factoring algorithm): may need > 1 samples (runs of the quantum circuit) to recover the period
  - Troublesome case: we sample a such that $\gcd(a, Q) > 1$

Is the Jacobi Function "Reasonable"?

- Generalised quantum period finding: we recover $\frac{a}{Q}$ with probability

$$\left|\hat{f}(a)\right|^2 = \frac{1}{Q}\left|\sum_{x=0}^{Q-1}\left(\frac{x}{Q}\right)\cdot\exp\left(\frac{2\pi iax}{Q}\right)\right|^2$$

- Typically (and in Shor's factoring algorithm): may need > 1 samples (runs of the quantum circuit) to recover the period
  - Troublesome case: we sample a such that $\gcd(a, Q) > 1$
- **Theorem (follows from Gauss sums): 1 sample always suffices for Jacobi!**

Is the Jacobi Function "Reasonable"?

- Generalised quantum period finding: we recover $\frac{a}{Q}$ with probability

$$\left|\hat{f}(a)\right|^2 = \frac{1}{Q}\left|\sum_{x=0}^{Q-1}\left(\frac{x}{Q}\right)\cdot\exp\left(\frac{2\pi i a x}{Q}\right)\right|^2$$

- Typically (and in Shor's factoring algorithm): may need > 1 samples (runs of the quantum circuit) to recover the period
  - Troublesome case: we sample a such that $\gcd(a, Q) > 1$
- **Theorem (follows from Gauss sums): 1 sample always suffices for Jacobi!**
  - Proof: Gauss sums $\rightarrow \hat{f}(a) = 0$ whenever $\gcd(a, Q) > 1$

Is the Jacobi Function "Reasonable"?

- Generalised quantum period finding: we recover $\frac{a}{Q}$ with probability

$$\left|\hat{f}(a)\right|^2 = \frac{1}{Q}\left|\sum_{x=0}^{Q-1}\left(\frac{x}{Q}\right)\cdot\exp\left(\frac{2\pi i a x}{Q}\right)\right|^2$$

- Typically (and in Shor's factoring algorithm): may need > 1 samples (runs of the quantum circuit) to recover the period
  - Troublesome case: we sample a such that $\gcd(a, Q) > 1$
- **Theorem (follows from Gauss sums): 1 sample always suffices for Jacobi!**
  - Proof: Gauss sums → $\hat{f}(a) = 0$ whenever $\gcd(a, Q) > 1$
  - Special case for intuition: if Q is prime, we have $\hat{f}(0) = \mathbb{E}\left[\left(\frac{x}{Q}\right)\right] = 0$

Is the Jacobi Function "Reasonable"?

- Generalised quantum period finding: we recover $\frac{a}{Q}$ with probability

$$\left|\hat{f}(a)\right|^2 = \frac{1}{Q}\left|\sum_{x=0}^{Q-1}\left(\frac{x}{Q}\right)\cdot\exp\left(\frac{2\pi i a x}{Q}\right)\right|^2$$

The Jacobi function isn't just "reasonably good" for general quantum period finding, it's actually magically well-suited to it — even more so than the periodic function used by Shor to factor!

- Typically (and in Shor's factoring algorithm), we need > 1 samples (runs of the quantum circuit) to recover the period
    - Troublesome case: we sample a such that $\gcd(a, Q) > 1$
- **Theorem (follows from Gauss sums): 1 sample always suffices for Jacobi!**
    - Proof: Gauss sums → $\hat{f}(a) = 0$ whenever $\gcd(a, Q) > 1$
    - Special case for intuition: if Q is prime, we have $\hat{f}(0) = 0$

Our Contribution: Pushing 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

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

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

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

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

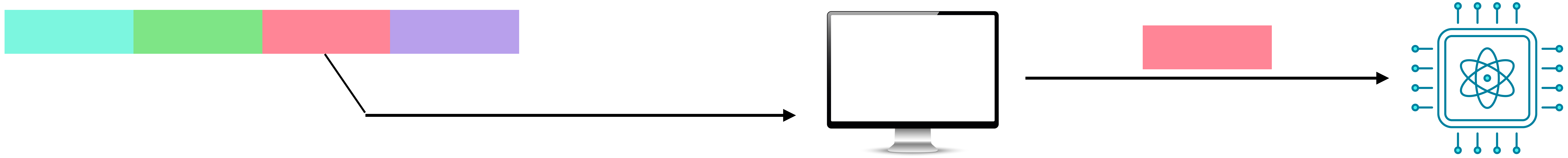
- Goal: compute $\left(\frac{a}{N}\right)$ for $a \leq \mathsf{poly}(Q)$
- How could we compute this without ever writing down all of N quantumly?

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 $\binom{a}{N}$ 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



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 $a \leq \mathsf{poly}(Q)$
- Recall:

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

Computing the Jacobi Symbol

It's all about $N \bmod a$

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

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

Our Result, Distilled

- <u>Theorem (K**R**VV24):</u> 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: $\left(\dfrac{a}{N}\right)$

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 (KRVV24): 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)

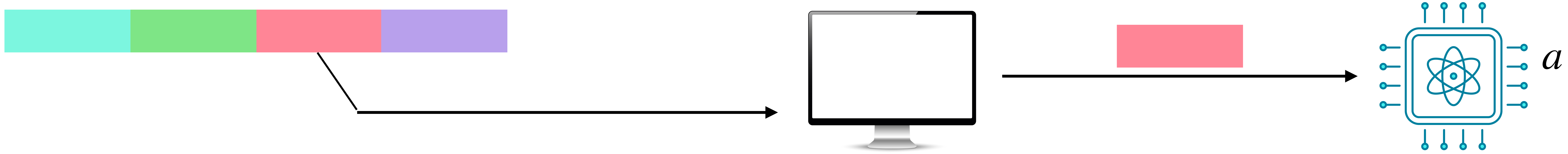
- **Corollary 2: we can factor $N = P^2Q$ in $\tilde{O}(\log N)$ gates and $\tilde{O}(\log Q)$ qubits**
  - **Just need the above theorem for $a \leq \mathsf{poly}(Q)$**

Computing $N \bmod a$ 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

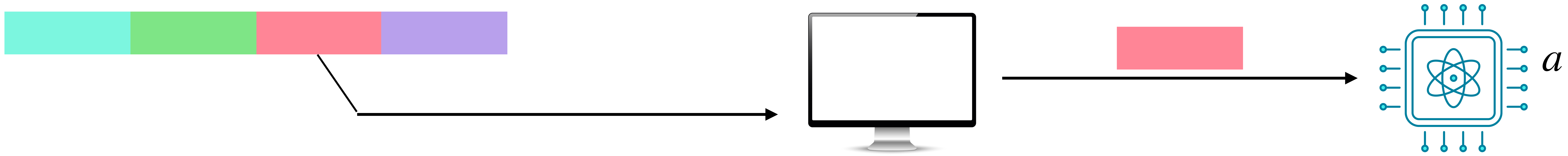
Computing $N \bmod a$ with Quantum Streaming

- 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

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

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

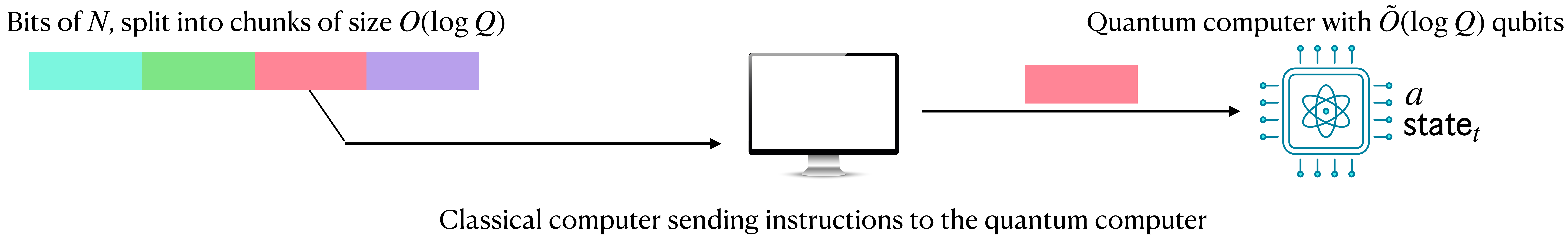


Classical computer sending instructions to the quantum computer

Computing $N \bmod a$ with Quantum Streaming

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

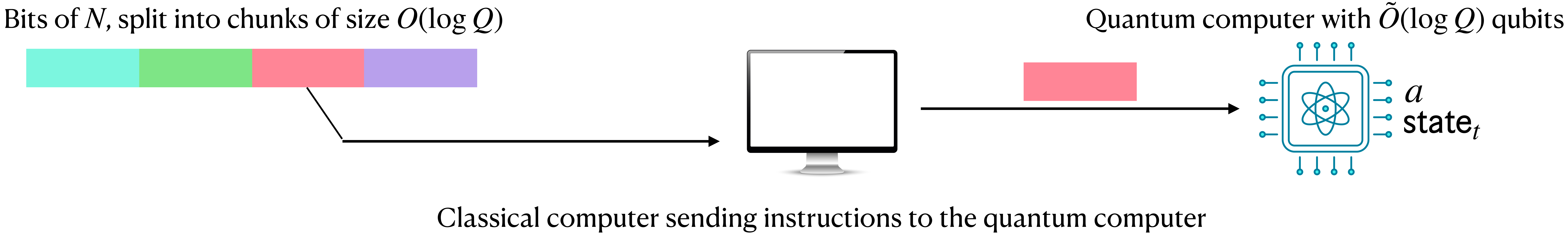
Notation: N has n bits, a has $m = O(\log Q)$ bits



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



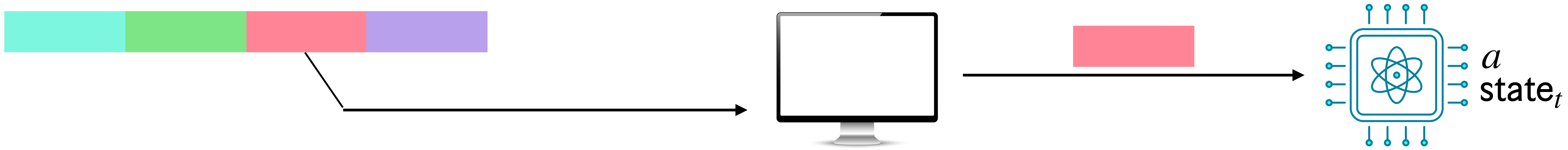
Computing N mod a with Quantum Streaming

- 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

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

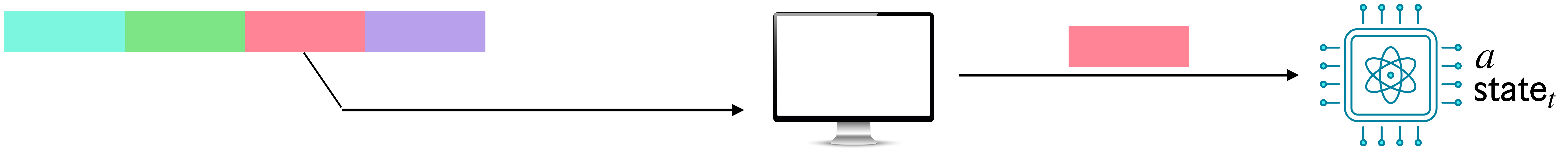
Computing N mod a with Quantum Streaming

- 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 $\mathsf{state}_{t_{\max}}$
  - 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 $N \equiv N_t \pmod{2^t}$
  - Equivalently: N_t agrees with N on the t lowest-order bits

Our Construction, Simplified

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

Our Construction, Simplified

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

Our Construction, Simplified

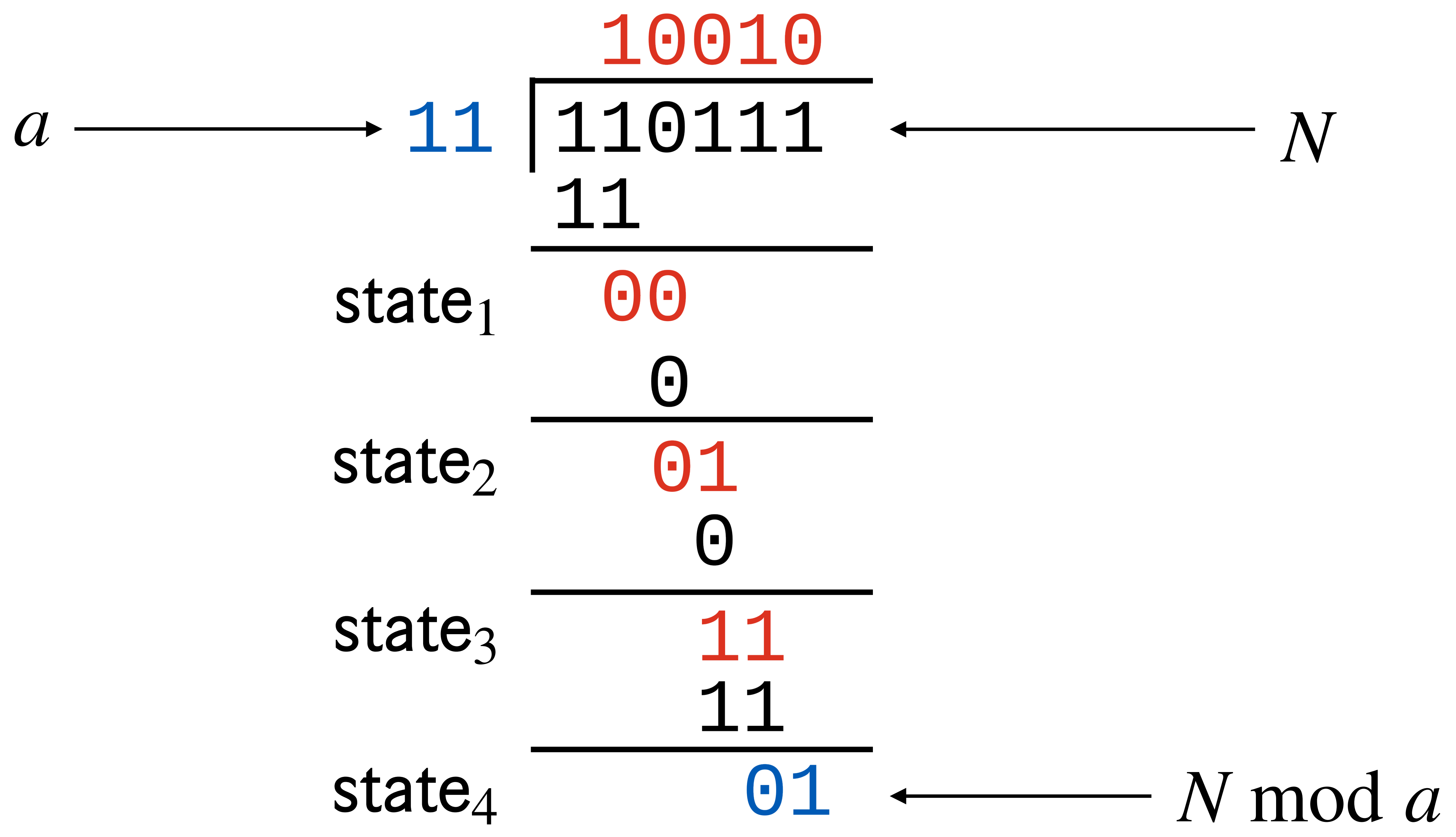
- At time $t = 0,\ldots,n-m$, let N_t be a multiple of a such that $N \equiv N_t \pmod{2^t}$
  - Equivalently: N_t agrees with N on the t lowest-order bits
- N_t is easily constructible from N_{t-1}
- Bits of N_t split into two parts:
  - t low-order bits: these match N → classically known → no need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

Our Construction, Simplified

- At time $t = 0, \ldots, n - m$, let N_t be a multiple of a such that $N \equiv N_t \pmod{2^t}$
  - Equivalently: N_t agrees with N on the t lowest-order bits
- N_t is easily constructible from N_{t-1}
- Bits of N_t split into two parts:
  - t 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 state_{n-m} suffices to reconstruct $N \bmod a$

Our Construction, In Detail

A Natural Attempt: Long Division

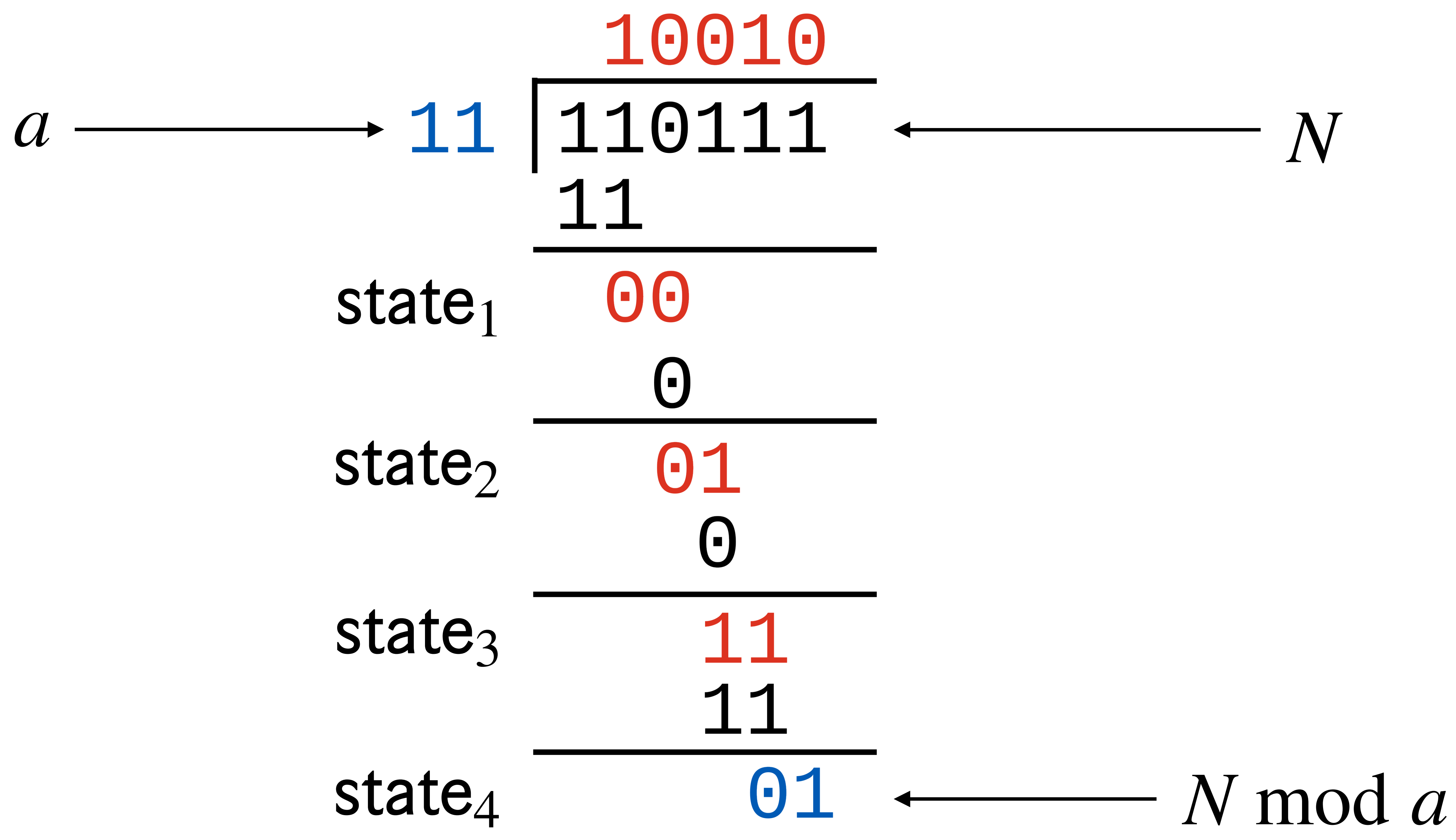


Required state

Excess state that we cannot clean up

Excess state that we can clean up

A Natural Attempt: Long Division



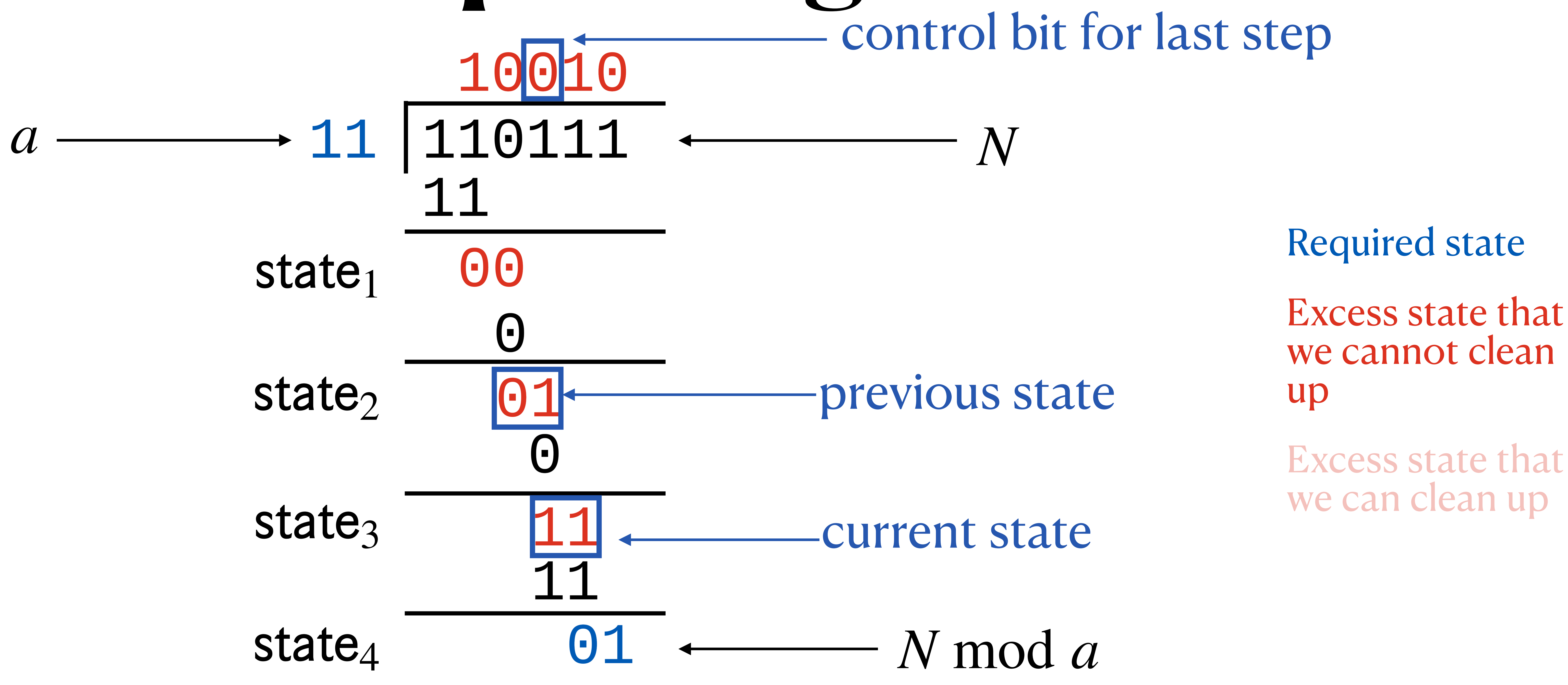
Required state

Excess state that we cannot clean up

Excess state that we can clean up

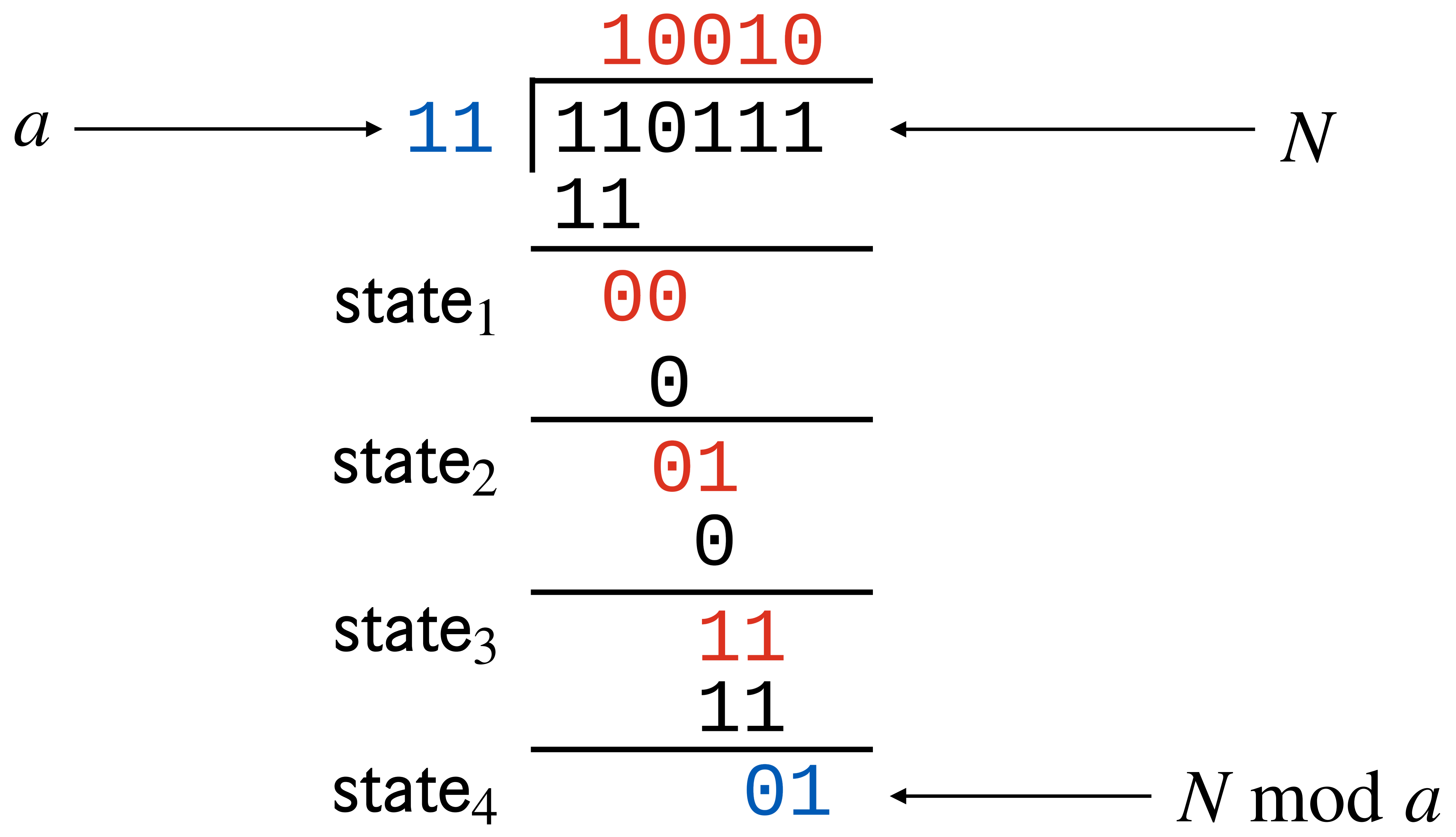
- Pro: only need to look at $O(m)$ bits of N at a time, and each state_t is compact

A Natural Attempt: Long Division



- Pro: only need to look at $O(m)$ bits of N at a time, and each state_t is compact
- Con: no reversibility → end up using $O(n)$ qubits anyway

Long Division: A Bird's Eye View

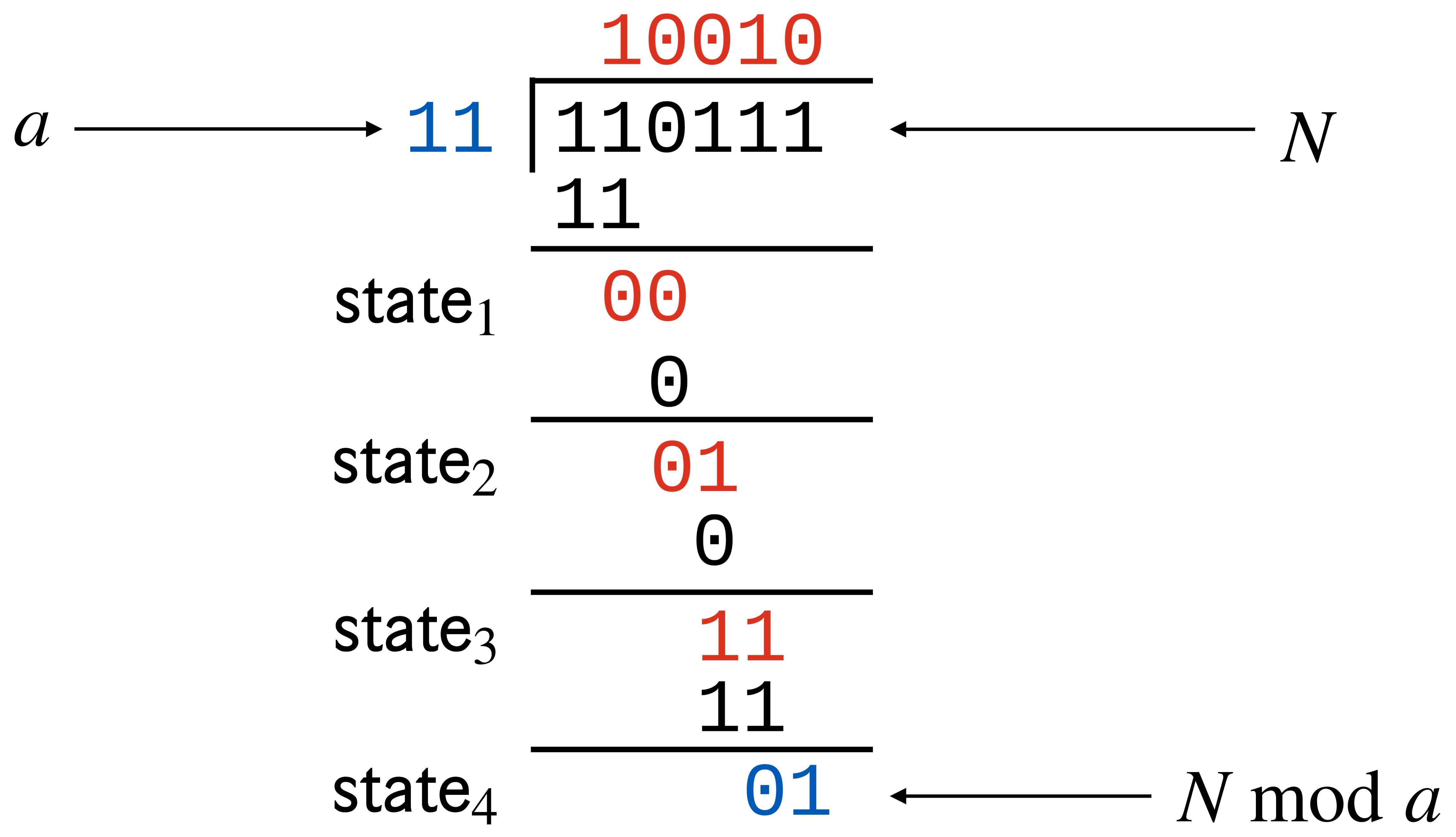


Required state

Excess state that we cannot clean up

Excess state that we can clean up

Long Division: A Bird's Eye View



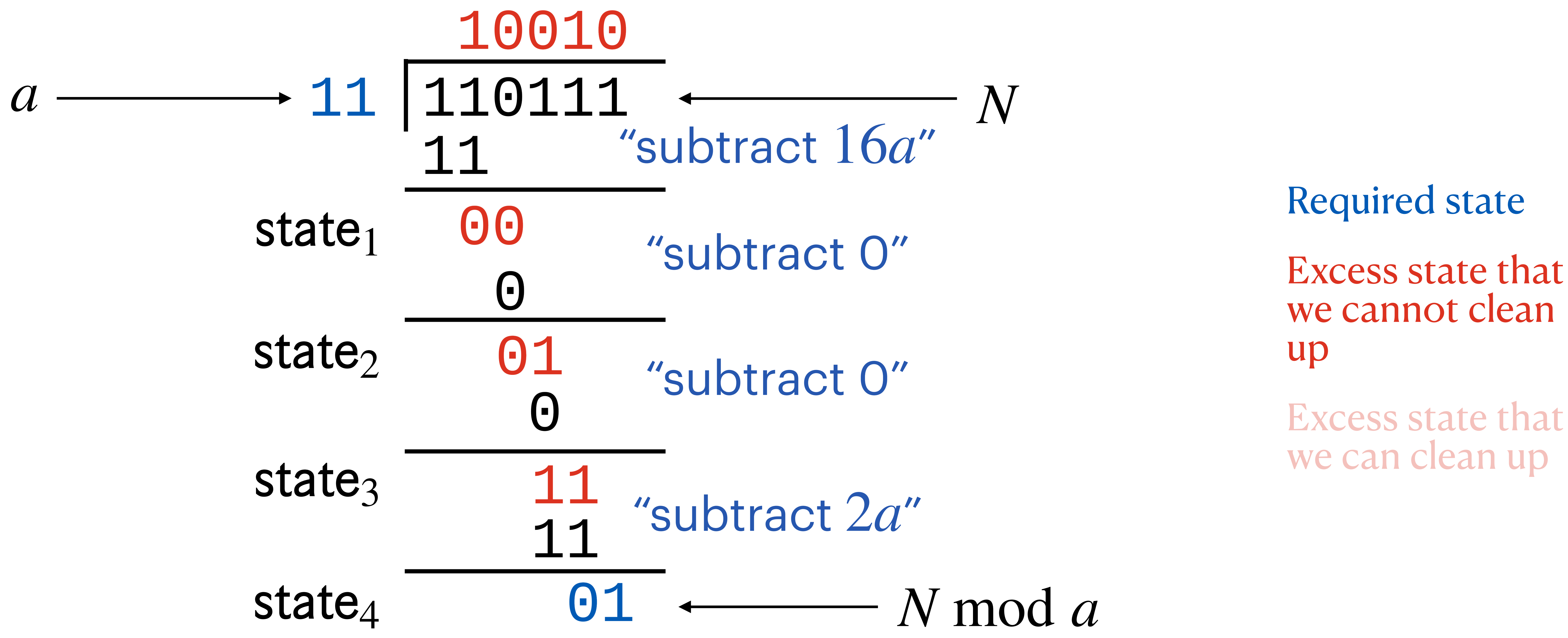
Required state

Excess state that we cannot clean up

Excess state that we can clean up

- Goal of long division: find k such that $N \approx ka$, and output $N - ka$

Long Division: A Bird's Eye View



- Goal of long division: find k such that $N \approx ka$, and output $N - ka$
- Does this by subtracting $2^r a$ from state, starting with large r (most significant bit of k) and going down to $r = 0$ (least significant bit of k)

Our Idea: "Backwards Long Division"

Long division: initialise $\mathsf{state} = N$ and subtract $2^r a$ from state, starting from large r (MSB of k) and going down to small r (LSB of k)

Our Idea: "Backwards Long Division"

Long division: initialise state $= N$ and subtract $2^r a$ from state, starting from large r (MSB of k) and going down to small r (LSB of k)

Multiple of a from long division: `110110`

N: `110111`

Our Idea: "Backwards Long Division"

Long division: initialise state $= N$ and subtract $2^r a$ from state, starting from large r (MSB of k) and going down to small r (LSB of k)

Multiple of a from long division:	110110
N:	110111
From backwards long division:	100111

Backwards long division: initialise state $= 0$ and **add $2^r a$ to state, starting from small r (LSB of k) and going up to large r (MSB of k)**

Our Idea: "Backwards Long Division"

Long division: initialise state $= N$ and subtract $2^r a$ from state, starting from large r (MSB of k) and going down to small r (LSB of k)

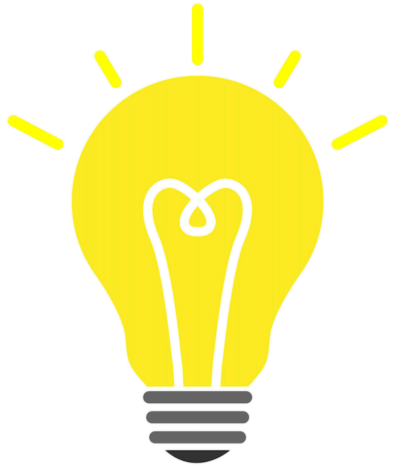
Multiple of a from long division: `110110`

N: `110111`

From backwards long division: `100111`

Backwards long division: initialise state $= 0$ and **add $2^r a$ to state, starting from small r (LSB of k) and going up to large r (MSB of k)**

Key observation: it is very easy at time t to decide based on state whether we just added $2^t a$; simply check whether state $\geq 2^t a$



Our Idea: "Backwards Long Division"

Long division: initialise state $= N$ and subtract $2^r a$ from state, starting from large r (MSB of k) and going down to small r (LSB of k)

Multiple of a from long division: 110110

N: 110111

From backwards long division: 100111

After backwards long division: once we have a multiple of a that matches N in the least significant bits, computing $N \bmod a$ is straightforward!

Backwards long division: initialise state $= 0$ and **add $2^r a$ to state, starting from small r (LSB of k) and going up to large r (MSB of k)**

Key observation: it is very easy at time t to decide based on state **whether we just added $2^t a$; simply check whether** state $\geq 2^t a$

Our Idea: "Backwards Long Division"

Long division: initialise state $= N$ and subtract $2^r a$ from state, starting from large r (MSB of k) and going down to small r (LSB of k)

Multiple of a from long division: 110110

N: 110111

From backwards long division: 100111

After backwards long division: once we have a multiple of a that matches N in the least significant bits, computing $N \bmod a$ is straightforward!

Backwards long division: initialise state $= 0$ and **add $2^r a$ to state, starting from small r (LSB of k) and going up to large r (MSB of k)**

Key observation: it is very easy at time t to decide based on state **whether we just added $2^t a$; simply check whether** state $\geq 2^t a$

Implementing Backwards Long Division

- Instead of constructing k from MSB to LSB, let's construct it from LSB to MSB

Key observation: it is very easy at time t to decide based on state whether we just added $2^t a$; simply check whether $\mathsf{state} \geq 2^t a$

Implementing Backwards Long Division

- Instead of constructing k from MSB to LSB, let's construct it from LSB to MSB
- Algorithm (assume for simplicity that a is odd):
  - Initialise $\mathsf{state} = 0$ (state will eventually be our multiple ka)

Key observation: it is very easy at time t to decide based on state **whether we just added** $2^t a$**; simply check whether** $\mathsf{state} \geq 2^t a$

Implementing Backwards Long Division

- Instead of constructing k from MSB to LSB, let's construct it from LSB to MSB
- Algorithm (assume for simplicity that a is odd):
  - Initialise state $= 0$ (state will eventually be our multiple ka)
  - At time t, add $2^t a$ to state if 

Key observation: it is very easy at time t to decide based on state **whether we just added $2^t a$; simply check whether** state $\geq 2^t a$

Implementing Backwards Long Division

- Instead of constructing k from MSB to LSB, let's construct it from LSB to MSB
- Algorithm (assume for simplicity that a is odd):
  - Initialise state $= 0$ (state will eventually be our multiple ka)
  - At time t, add $2^t a$ to state if 

  - Stop when N and state $(= ka)$ are "close" in some sense 

Key observation: it is very easy at time t to decide based on state **whether we just added $2^t a$; simply check whether** state $\geq 2^t a$

Implementing Backwards Long Division

- Instead of constructing k from MSB to LSB, let's construct it from LSB to MSB
- Algorithm (assume for simplicity that a is odd):
  - Initialise $\mathsf{state} = 0$ (state will eventually be our multiple ka)
  - At time t, add $2^t a$ to state if N and state differ in the tth least significant bit
  - Stop when N and state $(= ka)$ are "close" in some sense 

Key observation: it is very easy at time t to decide based on state whether we just added $2^t a$; simply check whether $\mathsf{state} \geq 2^t a$

Implementing Backwards Long Division

- Instead of constructing k from MSB to LSB, let's construct it from LSB to MSB
- Algorithm (assume for simplicity that a is odd):
  - Initialise state $= 0$ (state will eventually be our multiple ka)
  - At time t, add $2^t a$ to state if N and state differ in the tth least significant bit
  - Stop when N and state $(= ka)$ agree on the $n - m$ least significant bits

Key observation: it is very easy at time t to decide based on state whether we just added $2^t a$; simply check whether state $\geq 2^t a$

Implementing Backwards Long Division

- Instead of constructing k from MSB to LSB, let's construct it from LSB to MSB
- Algorithm (assume for simplicity that a is odd):
  - Initialise $\mathsf{state} = 0$ (state will eventually be our multiple ka)
  - At time t, add $2^t a$ to state if N and state differ in the tth least significant bit
  - Stop when N and state $(=ka)$ agree on the $n-m$ least significant bits
    - Equivalently: $N \equiv ka \pmod{2^{n-m}}$

Key observation: it is very easy at time t to decide based on state whether we just added $2^t a$; simply check whether $\mathsf{state} \geq 2^t a$

Tracing Through Our Algorithm

Recall:

state $= ka$

```
2^0 a =      11
    N =  110111
    k =       1
state =      11
```

Tracing Through Our Algorithm

Recall:

state $= ka$

2^0 a =	11	2^1 a =	11
N =	110111	N =	110111
k =	1	k =	01
state =	11	state =	011

Tracing Through Our Algorithm

Recall:

state $= ka$

2^0 a =	11	2^1 a =	11	2^2 a =	11
N =	110111	N =	110111	N =	110111
k =	1	k =	01	k =	101
state =	11	state =	011	state =	1111

Tracing Through Our Algorithm

Recall:

state = ka

2^0 a =	11	2^1 a =	11	2^2 a =	11	2^3 a =	11
N =	110111	N =	110111	N =	110111	N =	110111
k =	1	k =	01	k =	101	k =	1101
state =	11	state =	011	state =	1111	state =	100111

Tracing Through Our Algorithm

Recall:
state $= ka$

	Step 0	Step 1	Step 2	Step 3
	2^0 a = 11	2^1 a = 11	2^2 a = 11	2^3 a = 11
	N = 110111	N = 110111	N = 110111	N = 110111
	k = 1	k = 01	k = 101	k = 1101
	state = 11	state = 011	state = 1111	state = 100111

Tracing Through Our Algorithm

Recall:
state $= ka$

	Step 0	Step 1	Step 2	Step 3
2^t a =	2^0 a = 11	2^1 a = 11	2^2 a = 11	2^3 a = 11
N =	110111	110111	110111	110111
k =	1	01	101	1101
state =	11	011	1111	100111

- Our earlier observation: a simple comparison between state and $2^t a$ suffices for uncomputation at each step

Tracing Through Our Algorithm

Recall:

state $= ka$

2^0 a =	11	2^1 a =	11	2^2 a =	11	2^3 a =	11
N =	110111	N =	110111	N =	110111	N =	110111
k =	1	k =	01	k =	101	k =	1101
state =	11	state =	011	state =	1111	state =	100111

- Our earlier observation: a simple comparison between state and $2^t a$ suffices for uncomputation at each step
- Second observation:
  - The trailing bits of state and N are classically known; and

Tracing Through Our Algorithm

Recall:

state $= ka$

	Step 0	Step 1	Step 2	Step 3
2^t a =	2^0 a = 11	2^1 a = 11	2^2 a = 11	2^3 a = 11
N =	110111	110111	110111	110111
k =	1	01	101	1101
state =	11	011	1111	100111

- Our earlier observation: a simple comparison between state and $2^t a$ suffices for uncomputation at each step
- Second observation:
  - The trailing bits of state and N are classically known; and
  - We only need the leading-order bits of state to make the above comparison with $2^t a$

Tracing Through Our Algorithm

Recall:
state $= ka$

2^0 a =	11	2^1 a =	11	2^2 a =	11	2^3 a =	11
N =	110111	N =	110111	N =	110111	N =	110111
k =	1	k =	01	k =	101	k =	1101
state =	11	state =	011	state =	1111	state =	100111

- Our earlier observation: a simple comparison between state and $2^t a$ suffices for uncomputation at each step
- Second observation:
  - The trailing bits of state and N are classically known; and
  - We only need the leading-order bits of state to make the above comparison with $2^t a$

This now gets us down to $O(m)$ space!

Efficiency of Our Algorithm

- Space: $O(m) = O(\log a)$

Efficiency of Our Algorithm

- Space: $O(m) = O(\log a)$
- Computation boils down to $O(n)$ additions of m-bit integers

Efficiency of Our Algorithm

- Space: $O(m) = O(\log a)$
- Computation boils down to $O(n)$ additions of m-bit integers
  - Gates: $O(mn)$
  - Depth: $\tilde{O}(n)$

Efficiency of Our Algorithm

- Space: $O(m) = O(\log a)$
- Computation boils down to $O(n)$ additions of m-bit integers
  - Gates: $O(mn)$
  - Depth: $\tilde{O}(n)$

This is already nice, but we can do better…

Pushing Down the Gates and Depth

- Idea: instead of setting one bit of k at a time, we will set m bits at a time

Pushing Down the Gates and Depth

- Idea: instead of setting one bit of k at a time, we will set m bits at a time
  - Enables us to benefit from fast integer multiplication algorithms (for m-bit inputs)

Pushing Down the Gates and Depth

- Idea: instead of setting one bit of k at a time, we will set m bits at a time
  - Enables us to benefit from fast integer multiplication algorithms (for m-bit inputs)
  - To decide what multiple of a to add at each step: start by computing $a^{-1} \bmod 2^m$

Pushing Down the Gates and Depth

- Idea: instead of setting one bit of k at a time, we will set m bits at a time
  - Enables us to benefit from fast integer multiplication algorithms (for m-bit inputs)
  - To decide what multiple of a to add at each step: start by computing $a^{-1} \bmod 2^m$

Two options to compute $a^{-1} \bmod 2^m$:

1. (Obnoxious; concretely inefficient) Use Schönhage to achieve this in $\tilde{O}(m)$ gates/space/depth

Pushing Down the Gates and Depth

- Idea: instead of setting one bit of k at a time, we will set m bits at a time
  - Enables us to benefit from fast integer multiplication algorithms (for m-bit inputs)
  - To decide what multiple of a to add at each step: start by computing $a^{-1} \bmod 2^m$

Two options to compute $a^{-1} \bmod 2^m$:

1. (Obnoxious; concretely inefficient) Use Schönhage to achieve this in $\tilde{O}(m)$ gates/space/depth
2. (Using the 2^m modulus; can be made concretely efficient) Recurse down to computing $a^{-1} \bmod 2^{m/2}$
   a. Recursion step just does some multiplications and bit shifts on m-bit integers

Pushing Down the Gates and Depth

- Idea: instead of setting one bit of k at a time, we will set m bits at a time
  - Enables us to benefit from fast integer multiplication algorithms (for m-bit inputs)
  - To decide what multiple of a to add at each step: start by computing $a^{-1} \bmod 2^m$
- Space usage: still $\tilde{O}(m)$

Pushing Down the Gates and Depth

- Idea: instead of setting one bit of k at a time, we will set m bits at a time
  - Enables us to benefit from fast integer multiplication algorithms (for m-bit inputs)
  - To decide what multiple of a to add at each step: start by computing $a^{-1} \bmod 2^m$
- Space usage: still $\tilde{O}(m)$
- Computational work: $O(n/m)$ multiplications of m-bit integers

Pushing Down the Gates and Depth

- Idea: instead of setting one bit of k at a time, we will set m bits at a time
  - Enables us to benefit from fast integer multiplication algorithms (for m-bit inputs)
  - To decide what multiple of a to add at each step: start by computing $a^{-1} \bmod 2^m$
- Space usage: still $\tilde{O}(m)$
- Computational work: $O(n/m)$ multiplications of m-bit integers
  - Gates: $O(n/m) \times \tilde{O}(m) = \tilde{O}(n)$
  - Depth: $O(n/m) \times \tilde{O}(1) = \tilde{O}(n/m)$

Summary and Open Questions

Summary of Our Algorithm

To compute $\left(\dfrac{a}{N}\right)$ for $a < 2^m$ (where $m = O(\log Q)$):

1. Compute $N \bmod a$

Summary of Our Algorithm

To compute $\left(\frac{a}{N}\right)$ for $a < 2^m$ (where $m = O(\log Q)$):

1. Compute $N \bmod a$
   - Technical challenge: achieving this in gates $\tilde{O}(n)$, space $\tilde{O}(m)$, and depth $\tilde{O}(n/m)$

Summary of Our Algorithm

To compute $\left(\dfrac{a}{N}\right)$ for $a < 2^m$ (where $m = O(\log Q)$):

1. Compute $N \bmod a$
   - Technical challenge: achieving this in gates $\tilde{O}(n)$, space $\tilde{O}(m)$, and depth $\tilde{O}(n/m)$
2. Finish by computing $\left(\dfrac{N \bmod a}{a}\right)$ using Schönhage's algorithm

Summary of Our Algorithm

To compute $\left(\frac{a}{N}\right)$ for $a < 2^m$ (where $m = O(\log Q)$):

1. Compute $N \bmod a$
   - Technical challenge: achieving this in gates $\tilde{O}(n)$, space $\tilde{O}(m)$, and depth $\tilde{O}(n/m)$
2. Finish by computing $\left(\frac{N \bmod a}{a}\right)$ using Schönhage's algorithm
   - Contributes an additional depth of $\tilde{O}(m)$ (dominant term in depth if $m \gg \sqrt{n}$)

Summary: Our Factoring Circuit for P^2Q

- Main theorem (K**R**VV24): for $N = P^2Q < 2^n$ such that Q is squarefree and $< 2^m$, we can recover P, Q from N in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m + m)$ depth

Summary: Our Factoring Circuit for P^2Q

- Main theorem (K**R**VV24): for $N = P^2Q < 2^n$ such that Q is squarefree and $< 2^m$, we can recover P, Q from N in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m + m)$ depth

Central workhorse: an efficient quantum circuit for computing $N \bmod a$ for classical $N < 2^n$ and quantum $a < 2^m$

Summary: Our Factoring Circuit for P^2Q

Authors	Types of inputs	Gates	Space	Depth
Shor (1994)	Any	$\tilde{O}(n^2)$	$\tilde{O}(n)$	$\tilde{O}(n)$
LPDS (2012)	$N = P^2Q$	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n)$
KCVY (2021)	N/A	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(1)$
Regev (2023)	Any	$\tilde{O}(n^{1.5})$	$O(n^{1.5})$	$\tilde{O}(n^{0.5})$
RV (2024)	Any	$\tilde{O}(n^{1.5})$	$\tilde{O}(n)$	$\tilde{O}(n^{0.5})$
K**R**VV (2024)	$N = P^2Q\ (Q < 2^m)$	$\tilde{O}(n)$	$\tilde{O}(m)$	$\tilde{O}(n/m + m)$

Open Questions

- (Ongoing work) What are the **concrete** costs of these algorithms?

Open Questions

- (Ongoing work) What are the **concrete** costs of these algorithms?
- Can error-correcting codes and fault tolerance be designed in a way that is especially suited to computing the Jacobi symbol?

Open Questions

- (Ongoing work) What are the **concrete** costs of these algorithms?
- Can error-correcting codes and fault tolerance be designed in a way that is especially suited to computing the Jacobi symbol?
- Better classical algorithms for factoring P^2Q when Q is small?

Open Questions

- (Ongoing work) What are the **concrete** costs of these algorithms?
- Can error-correcting codes and fault tolerance be designed in a way that is especially suited to computing the Jacobi symbol?
- Better classical algorithms for factoring P^2Q when Q is small?
- What other integers N can we factor using this algorithm (e.g. using number theory magic)?

Open Questions

- (Ongoing work) What are the **concrete** costs of these algorithms?
- Can error-correcting codes and fault tolerance be designed in a way that is especially suited to computing the Jacobi symbol?
- Better classical algorithms for factoring P^2Q when Q is small?
- What other integers N can we factor using this algorithm (e.g. using number theory magic)?
  - Current generalisation: can **completely** factor $N = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_r^{\alpha_r}$ for distinct $\alpha_1, \dots, \alpha_r$

Open Questions

- (Ongoing work) What are the **concrete** costs of these algorithms?
- Can error-correcting codes and fault tolerance be designed in a way that is especially suited to computing the Jacobi symbol?
- Better classical algorithms for factoring P^2Q when Q is small?
- What other integers N can we factor using this algorithm (e.g. using number theory magic)?
  - Current generalisation: can **completely** factor $N = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_r^{\alpha_r}$ for distinct $\alpha_1, \dots, \alpha_r$
- Other quantum factoring algorithms exploiting special structure in N? (Many such algorithms in the classical world)

Bonus Slides

Our Result, Distilled

- Theorem (K**R**VV24): for $a < 2^m$ (in our application, $m = O(\log Q)$) and classically known $N < 2^n$, we can compute

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

in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m)$ depth

Our Result, Distilled

- Theorem (K**R**VV24): for $a < 2^m$ (in our application, $m = O(\log Q)$) and classically known $N < 2^n$, we can compute

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

in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m)$ depth

- Corollary 1: all of the following can be computed in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m + m)$ depth for quantum a and classical N:
  - Jacobi symbol: $\left(\frac{a}{N}\right)$

Our Result, Distilled

- Theorem (K**R**VV24): for $a < 2^m$ (in our application, $m = O(\log Q)$) and classically known $N < 2^n$, we can compute

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

in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m)$ depth

- Corollary 1: all of the following can be computed in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m + m)$ depth 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 $a < 2^m$ (in our application, $m = O(\log Q)$) and classically known $N < 2^n$, we can compute

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

in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m)$ depth

- Corollary 1: all of the following can be computed in $\tilde{O}(n)$ gates, $\tilde{O}(m)$ qubits, and $\tilde{O}(n/m + m)$ depth 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?

From Backwards Long Division to $N \bmod a$

- The example we just worked out:
  - $N = 55, a = 3$ $(n = 6, m = 2)$

simplified presentation based on an observation by Daniel J. Bernstein

From Backwards Long Division to $N \bmod a$

- The example we just worked out:
  - $N = 55, a = 3$ $(n = 6, m = 2)$
  - End up with $ka = \mathsf{state} = 39 = 13 \times 3 \equiv N \pmod{2^{n-m} = 16}$

simplified presentation based on an observation by Daniel J. Bernstein

From Backwards Long Division to $N \bmod a$

- The example we just worked out:
  - $N = 55, a = 3$ $(n = 6, m = 2)$
  - End up with $ka = \textsf{state} = 39 = 13 \times 3 \equiv N \pmod{2^{n-m} = 16}$
- Now define $N' = \dfrac{N - \textsf{state}}{2^{n-m}} = \dfrac{N - ka}{2^{n-m}}$ (informally, this is the "remainder")

simplified presentation based on an observation by Daniel J. Bernstein

From Backwards Long Division to $N \bmod a$

- The example we just worked out:
  - $N = 55, a = 3$ ($n = 6, m = 2$)
  - End up with $ka = \mathsf{state} = 39 = 13 \times 3 \equiv N \pmod{2^{n-m} = 16}$
- Now define $N' = \dfrac{N - \mathsf{state}}{2^{n-m}} = \dfrac{N - ka}{2^{n-m}}$ (informally, this is the "remainder")
- Finally: $N \equiv N' \cdot \left(2^{n-m} \bmod a\right) \pmod{a}$, and the RHS is easily computable with $\tilde{O}(m)$ gates!

simplified presentation based on an observation by Daniel J. Bernstein

From Backwards Long Division to $N \bmod a$

- The example we just worked out:
  - $N = 55, a = 3$ ($n = 6, m = 2$)
  - End up with $ka = \mathsf{state} = 39 = 13 \times 3 \equiv N \pmod{2^{n-m} = 16}$
- Now define $N' = \dfrac{N - \mathsf{state}}{2^{n-m}} = \dfrac{N - ka}{2^{n-m}}$ (informally, this is the "remainder")
- Finally: $N \equiv N' \cdot \left(2^{n-m} \bmod a\right) \pmod{a}$, and the RHS is easily computable with $\tilde{O}(m)$ gates!

That's it! We computed $N \bmod a$!

simplified presentation based on an observation by Daniel J. Bernstein