Introduction to
quantum algorithms
and introduction to
code-based cryptography

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Data ("state") stored in n bits:
an element of $\{0, 1\}^n$,
often viewed as representing
an element of $\{0, 1, \ldots, 2^n - 1\}$.

Introduction to
quantum algorithms
and introduction to
code-based cryptography

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Data ("state") stored in n bits:
an element of $\{0, 1\}^n$,
often viewed as representing
an element of $\{0, 1, \ldots, 2^n - 1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
Retrieving this vector is tough!

Introduction to
quantum algorithms
and introduction to
code-based cryptography

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Data ("state") stored in n bits: an element of $\{0,1\}^n$, often viewed as representing an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits: a nonzero element of $\mathbf{C}^{2^n}$. Retrieving this vector is tough!

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then **measuring** the qubits produces an element of $\{0,1,\ldots,2^n-1\}$ and destroys the state. Measurement produces element q with probability $|a_q|^2/\sum_r |a_r|^2$.

ction to
n algorithms
oduction to
sed cryptography

. Bernstein
ty of Illinois at Chicago &
che Universiteit Eindhoven

Data ("state") stored in n bits:
an element of $\{0,1\}^n$,
often viewed as representing
an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
Retrieving this vector is tough!

If n qubits have state
$(a_0, a_1, \ldots, a_{2^n-1})$ then
measuring the qubits produces
an element of $\{0,1,\ldots,2^n-1\}$
and destroys the state.
Measurement produces element q
with probability $|a_q|^2 / \sum_r |a_r|^2$.

Some ex

$(1, 0, 0, 0$
"$|0\rangle$" in
Measure

ns

o

graphy

n

is at Chicago &

siteit Eindhoven

Data ("state") stored in n bits:
an element of $\{0,1\}^n$,
often viewed as representing
an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
Retrieving this vector is tough!

If n qubits have state
$(a_0, a_1, \ldots, a_{2^n-1})$ then
measuring the qubits produces
an element of $\{0,1,\ldots,2^n-1\}$
and destroys the state.
Measurement produces element q
with probability $|a_q|^2/\sum_r |a_r|^2$.

Some examples of

$(1,0,0,0,0,0,0,0$

"$|0\rangle$" in standard

Measurement proc

ago &
hoven

Data ("state") stored in n bits:
an element of $\{0,1\}^n$,
often viewed as representing
an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
Retrieving this vector is tough!

If n qubits have state
$(a_0, a_1, \ldots, a_{2^n-1})$ then
measuring the qubits produces
an element of $\{0,1,\ldots,2^n-1\}$
and destroys the state.
Measurement produces element q
with probability $|a_q|^2 / \sum_r |a_r|^2$.

Some examples of 3-qubit st

$(1,0,0,0,0,0,0,0)$ is
"$|0\rangle$" in standard notation.
Measurement produces 0.

Data ("state") stored in n bits:
an element of $\{0,1\}^n$,
often viewed as representing
an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
Retrieving this vector is tough!

If n qubits have state
$(a_0, a_1, \ldots, a_{2^n-1})$ then
measuring the qubits produces
an element of $\{0,1,\ldots,2^n-1\}$
and destroys the state.
Measurement produces element q
with probability $|a_q|^2/\sum_r |a_r|^2$.

Some examples of 3-qubit states:

$(1,0,0,0,0,0,0,0)$ is
"$|0\rangle$" in standard notation.
Measurement produces 0.

Data ("state") stored in n bits:
an element of $\{0, 1\}^n$,
often viewed as representing
an element of $\{0, 1, \ldots, 2^n - 1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
Retrieving this vector is tough!

If n qubits have state
$(a_0, a_1, \ldots, a_{2^n-1})$ then
measuring the qubits produces
an element of $\{0, 1, \ldots, 2^n - 1\}$
and destroys the state.
Measurement produces element q
with probability $|a_q|^2 / \sum_r |a_r|^2$.

Some examples of 3-qubit states:

$(1, 0, 0, 0, 0, 0, 0, 0)$ is
"$|0\rangle$" in standard notation.
Measurement produces 0.

$(0, 0, 0, 0, 0, 0, 1, 0)$ is
"$|6\rangle$" in standard notation.
Measurement produces 6.

Data ("state") stored in n bits:
an element of $\{0,1\}^n$,
often viewed as representing
an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
Retrieving this vector is tough!

If n qubits have state
$(a_0, a_1, \ldots, a_{2^n-1})$ then
measuring the qubits produces
an element of $\{0,1,\ldots,2^n-1\}$
and destroys the state.
Measurement produces element q
with probability $|a_q|^2 / \sum_r |a_r|^2$.

Some examples of 3-qubit states:

$(1,0,0,0,0,0,0,0)$ is
"$|0\rangle$" in standard notation.
Measurement produces 0.

$(0,0,0,0,0,0,1,0)$ is
"$|6\rangle$" in standard notation.
Measurement produces 6.

$(0,0,0,0,0,0,-7i,0) = -7i|6\rangle$:
Measurement produces 6.

Data ("state") stored in n bits:
an element of $\{0,1\}^n$,
often viewed as representing
an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
Retrieving this vector is tough!

If n qubits have state
$(a_0, a_1, \ldots, a_{2^n-1})$ then
measuring the qubits produces
an element of $\{0,1,\ldots,2^n-1\}$
and destroys the state.
Measurement produces element q
with probability $|a_q|^2/\sum_r |a_r|^2$.

Some examples of 3-qubit states:

$(1,0,0,0,0,0,0,0)$ is
"$|0\rangle$" in standard notation.
Measurement produces 0.

$(0,0,0,0,0,0,1,0)$ is
"$|6\rangle$" in standard notation.
Measurement produces 6.

$(0,0,0,0,0,0,-7i,0) = -7i|6\rangle$:
Measurement produces 6.

$(0,0,4,0,0,0,8,0) = 4|2\rangle + 8|6\rangle$:
Measurement produces
2 with probability 20%,
6 with probability 80%.

state") stored in n bits:
ent of $\{0,1\}^n$,
ewed as representing
ent of $\{0,1,\ldots,2^n-1\}$.

ored in n qubits:
ro element of $\mathbf{C}^{2^n}$.
ng this vector is tough!

its have state
$\ldots, a_{2^n-1})$ then
ing the qubits produces
ent of $\{0,1,\ldots,2^n-1\}$
troys the state.
ement produces element q
bability $|a_q|^2/\sum_r |a_r|^2$.

Some examples of 3-qubit states:

$(1,0,0,0,0,0,0,0)$ is
"$|0\rangle$" in standard notation.
Measurement produces 0.

$(0,0,0,0,0,0,1,0)$ is
"$|6\rangle$" in standard notation.
Measurement produces 6.

$(0,0,0,0,0,0,-7i,0) = -7i|6\rangle$:
Measurement produces 6.

$(0,0,4,0,0,0,8,0) = 4|2\rangle + 8|6\rangle$:
Measurement produces
2 with probability 20%,
6 with probability 80%.

Fast qua

$(a_0, a_1, a$
$(a_1, a_0, a$
is compl
hence "c

ored in n bits:
$1\}^n$,
presenting
$1, \ldots, 2^n - 1\}$.

qubits:
of $\mathbf{C}^{2^n}$.
ctor is tough!

tate
then
bits produces
$1, \ldots, 2^n - 1\}$
tate.
duces element q
$_q|^2 / \sum_r |a_r|^2$.

Some examples of 3-qubit states:

$(1, 0, 0, 0, 0, 0, 0, 0)$ is
"$|0\rangle$" in standard notation.
Measurement produces 0.

$(0, 0, 0, 0, 0, 0, 1, 0)$ is
"$|6\rangle$" in standard notation.
Measurement produces 6.

$(0, 0, 0, 0, 0, 0, -7i, 0) = -7i|6\rangle$:
Measurement produces 6.

$(0, 0, 4, 0, 0, 0, 8, 0) = 4|2\rangle + 8|6\rangle$:
Measurement produces
2 with probability 20%,
6 with probability 80%.

Fast quantum ope

$(a_0, a_1, a_2, a_3, a_4, a$
$(a_1, a_0, a_3, a_2, a_5, a$
is complementing
hence "complemen

oits:

{

$- 1\}$.

gh!

uces

$- 1\}$

ent q

$a_r|^2$.

Some examples of 3-qubit states:

$(1,0,0,0,0,0,0,0)$ is
"$|0\rangle$" in standard notation.
Measurement produces 0.

$(0,0,0,0,0,0,1,0)$ is
"$|6\rangle$" in standard notation.
Measurement produces 6.

$(0,0,0,0,0,0,-7i,0) = -7i|6\rangle$:
Measurement produces 6.

$(0,0,4,0,0,0,8,0) = 4|2\rangle + 8|6\rangle$:
Measurement produces
2 with probability 20%,
6 with probability 80%.

Fast quantum operations, pa

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is complementing index bit 0
hence "complementing qubi

Some examples of 3-qubit states:

$(1, 0, 0, 0, 0, 0, 0, 0)$ is
"$|0\rangle$" in standard notation.
Measurement produces 0.

$(0, 0, 0, 0, 0, 0, 1, 0)$ is
"$|6\rangle$" in standard notation.
Measurement produces 6.

$(0, 0, 0, 0, 0, 0, -7i, 0) = -7i|6\rangle$:
Measurement produces 6.

$(0, 0, 4, 0, 0, 0, 8, 0) = 4|2\rangle + 8|6\rangle$:
Measurement produces
2 with probability 20%,
6 with probability 80%.

Fast quantum operations, part 1

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is complementing index bit 0,
hence "complementing qubit 0".

Some examples of 3-qubit states:

$(1, 0, 0, 0, 0, 0, 0, 0)$ is
"$|0\rangle$" in standard notation.
Measurement produces 0.

$(0, 0, 0, 0, 0, 0, 1, 0)$ is
"$|6\rangle$" in standard notation.
Measurement produces 6.

$(0, 0, 0, 0, 0, 0, -7i, 0) = -7i|6\rangle$:
Measurement produces 6.

$(0, 0, 4, 0, 0, 0, 8, 0) = 4|2\rangle + 8|6\rangle$:
Measurement produces
2 with probability 20%,
6 with probability 80%.

<u>Fast quantum operations, part 1</u>

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is complementing index bit 0,
hence "complementing qubit 0".

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
is measured as (q_0, q_1, q_2),
representing $q = q_0 + 2q_1 + 4q_2$,
with probability $|a_q|^2/\sum_r |a_r|^2$.

$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is measured as $(q_0 \oplus 1, q_1, q_2)$,
representing $q \oplus 1$,
with probability $|a_q|^2/\sum_r |a_r|^2$.

xamples of 3-qubit states:

0, 0, 0, 0, 0) is
standard notation.
ement produces 0.

0, 0, 0, 1, 0) is
standard notation.
ement produces 6.

0, 0, 0, $-7i$, 0) $= -7i|6\rangle$:
ement produces 6.

0, 0, 0, 8, 0) $= 4|2\rangle + 8|6\rangle$:
ement produces
probability 20%,
probability 80%.

Fast quantum operations, part 1

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is complementing index bit 0,
hence "complementing qubit 0".

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
is measured as (q_0, q_1, q_2),
representing $q = q_0 + 2q_1 + 4q_2$,
with probability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is measured as $(q_0 \oplus 1, q_1, q_2)$,
representing $q \oplus 1$,
with probability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_0, a_1,$ a
$(a_4, a_5,$ a
is "comp
$(q_0, q_1,$ c

3-qubit states:

) is

notation.

duces 0.

) is

notation.

duces 6.

$, 0) = -7i|6\rangle$:

duces 6.

$) = 4|2\rangle + 8|6\rangle$:

duces

20%,

80%.

Fast quantum operations, part 1

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is complementing index bit 0,
hence "complementing qubit 0".

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
is measured as (q_0, q_1, q_2),
representing $q = q_0 + 2q_1 + 4q_2$,
with probability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is measured as $(q_0 \oplus 1, q_1, q_2)$,
representing $q \oplus 1$,
with probability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_0, a_1, a_2, a_3, a_4,$ a
$(a_4, a_5, a_6, a_7, a_0,$ a
is "complementing
$(q_0, q_1, q_2) \mapsto (q_0$

tates:

$i|6\rangle$:

$8|6\rangle$:

Fast quantum operations, part 1

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is complementing index bit 0,
hence "complementing qubit 0".

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
is measured as (q_0, q_1, q_2),
representing $q = q_0 + 2q_1 + 4q_2$,
with probability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is measured as $(q_0 \oplus 1, q_1, q_2)$,
representing $q \oplus 1$,
with probability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2"
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus$

Fast quantum operations, part 1

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is complementing index bit 0,
hence "complementing qubit 0".

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
is measured as (q_0, q_1, q_2),
representing $q = q_0 + 2q_1 + 4q_2$,
with probability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is measured as $(q_0 \oplus 1, q_1, q_2)$,
representing $q \oplus 1$,
with probability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

Fast quantum operations, part 1

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is complementing index bit 0,
hence "complementing qubit 0".

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
is measured as (q_0, q_1, q_2),
representing $q = q_0 + 2q_1 + 4q_2$,
with probability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is measured as $(q_0 \oplus 1, q_1, q_2)$,
representing $q \oplus 1$,
with probability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7)$
is "swapping qubits 0 and 2":
$(q_0, q_1, q_2) \mapsto (q_2, q_1, q_0)$.

Fast quantum operations, part 1

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is complementing index bit 0,
hence "complementing qubit 0".

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
is measured as (q_0, q_1, q_2),
representing $q = q_0 + 2q_1 + 4q_2$,
with probability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is measured as $(q_0 \oplus 1, q_1, q_2)$,
representing $q \oplus 1$,
with probability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7)$
is "swapping qubits 0 and 2":
$(q_0, q_1, q_2) \mapsto (q_2, q_1, q_0)$.

Complementing qubit 2
= swapping qubits 0 and 2
∘ complementing qubit 0
∘ swapping qubits 0 and 2.

Similarly: swapping qubits i, j.

antum operations, part 1

$a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$a_3, a_2, a_5, a_4, a_7, a_6)$
ementing index bit 0,
complementing qubit 0".

$a_2, a_3, a_4, a_5, a_6, a_7)$
ured as (q_0, q_1, q_2),
ting $q = q_0 + 2q_1 + 4q_2$,
bability $|a_q|^2 / \sum_r |a_r|^2$.

$a_3, a_2, a_5, a_4, a_7, a_6)$
ured as $(q_0 \oplus 1, q_1, q_2)$,
ting $q \oplus 1$,
bability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7)$
is "swapping qubits 0 and 2":
$(q_0, q_1, q_2) \mapsto (q_2, q_1, q_0)$.

Complementing qubit 2
= swapping qubits 0 and 2
∘ complementing qubit 0
∘ swapping qubits 0 and 2.

Similarly: swapping qubits i, j.

$(a_0, a_1,$
$(a_0, a_1,$
is a "rev
"control
$(q_0, q_1,$

erations, part 1

$a_5, a_6, a_7) \mapsto$
$a_4, a_7, a_6)$
index bit 0,
nting qubit 0".

$a_5, a_6, a_7)$
$, q_1, q_2),$
$q_0 + 2q_1 + 4q_2,$
$_q|^2 / \sum_r |a_r|^2.$

$a_4, a_7, a_6)$
$\oplus 1, q_1, q_2),$
,
$_q|^2 / \sum_r |a_r|^2.$

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1).$

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7)$
is "swapping qubits 0 and 2":
$(q_0, q_1, q_2) \mapsto (q_2, q_1, q_0).$

Complementing qubit 2
= swapping qubits 0 and 2
 ∘ complementing qubit 0
 ∘ swapping qubits 0 and 2.

Similarly: swapping qubits i, j.

$(a_0, a_1, a_2, a_3, a_4, a$
$(a_0, a_1, a_3, a_2, a_4, a$
is a "reversible XC
"controlled NOT g
$(q_0, q_1, q_2) \mapsto (q_0$

art 1

$\mapsto$

0,

t 0".

$4q_2$,

$a_r|^2$.

$q_2)$,

$a_r|^2$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7)$
is "swapping qubits 0 and 2":
$(q_0, q_1, q_2) \mapsto (q_2, q_1, q_0)$.

Complementing qubit 2
= swapping qubits 0 and 2
$\circ$ complementing qubit 0
$\circ$ swapping qubits 0 and 2.

Similarly: swapping qubits i, j.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
$(a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6)$
is a "reversible XOR gate" =
"controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1, q_1,$

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7)$
is "swapping qubits 0 and 2":
$(q_0, q_1, q_2) \mapsto (q_2, q_1, q_0)$.

Complementing qubit 2
= swapping qubits 0 and 2
$\circ$ complementing qubit 0
$\circ$ swapping qubits 0 and 2.

Similarly: swapping qubits i, j.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6)$
is a "reversible XOR gate" =
"controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1, q_1, q_2)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7)$
is "swapping qubits 0 and 2":
$(q_0, q_1, q_2) \mapsto (q_2, q_1, q_0)$.

Complementing qubit 2
= swapping qubits 0 and 2
∘ complementing qubit 0
∘ swapping qubits 0 and 2.

Similarly: swapping qubits i, j.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6)$
is a "reversible XOR gate" =
"controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1, q_1, q_2)$.

Example with more qubits:

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$\mapsto (a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6,$
$a_8, a_9, a_{11}, a_{10}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{16}, a_{17}, a_{19}, a_{18}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{24}, a_{25}, a_{27}, a_{26}, a_{28}, a_{29}, a_{31}, a_{30})$.

$a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$

$a_6, a_7, a_0, a_1, a_2, a_3)$

plementing qubit 2":

$q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

$a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$

$a_2, a_6, a_1, a_5, a_3, a_7)$

pping qubits 0 and 2":

$q_2) \mapsto (q_2, q_1, q_0)$.

menting qubit 2

ping qubits 0 and 2

nplementing qubit 0

apping qubits 0 and 2.

y: swapping qubits i, j.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$

$(a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6)$

is a "reversible XOR gate" =

"controlled NOT gate":

$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1, q_1, q_2)$.

Example with more qubits:

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$

$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$

$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$

$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$

$\mapsto (a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6,$

$a_8, a_9, a_{11}, a_{10}, a_{12}, a_{13}, a_{15}, a_{14},$

$a_{16}, a_{17}, a_{19}, a_{18}, a_{20}, a_{21}, a_{23}, a_{22},$

$a_{24}, a_{25}, a_{27}, a_{26}, a_{28}, a_{29}, a_{31}, a_{30})$.

$(a_0, a_1,$

$(a_0, a_1,$

is a "To

"control

$(q_0, q_1,$

$a_5, a_6, a_7) \mapsto$

$a_1, a_2, a_3)$

g qubit 2":

$, q_1, q_2 \oplus 1)$.

$a_5, a_6, a_7) \mapsto$

$a_5, a_3, a_7)$

ts 0 and 2":

$, q_1, q_0)$.

ubit 2

s 0 and 2

ng qubit 0

its 0 and 2.

g qubits i, j.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6)$
is a "reversible XOR gate" $=$
"controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1, q_1, q_2)$.

Example with more qubits:

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$\mapsto (a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6,$
$a_8, a_9, a_{11}, a_{10}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{16}, a_{17}, a_{19}, a_{18}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{24}, a_{25}, a_{27}, a_{26}, a_{28}, a_{29}, a_{31}, a_{30})$.

$(a_0, a_1, a_2, a_3, a_4, a$

$(a_0, a_1, a_2, a_3, a_4, a$

is a "Toffoli gate"

"controlled contro

$(q_0, q_1, q_2) \mapsto (q_0$

$\mapsto$

:

1).

$\mapsto$

”:

2.

$, j.$

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6)$
is a “reversible XOR gate” =
“controlled NOT gate”:
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1, q_1, q_2)$.

Example with more qubits:

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$\mapsto (a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6,$
$a_8, a_9, a_{11}, a_{10}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{16}, a_{17}, a_{19}, a_{18}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{24}, a_{25}, a_{27}, a_{26}, a_{28}, a_{29}, a_{31}, a_{30})$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
$(a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)$
is a “Toffoli gate” =
“controlled controlled NOT
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1 q_2, q$

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6)$
is a "reversible XOR gate" $=$
"controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1, q_1, q_2)$.

Example with more qubits:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$\mapsto (a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6,$
$a_8, a_9, a_{11}, a_{10}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{16}, a_{17}, a_{19}, a_{18}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{24}, a_{25}, a_{27}, a_{26}, a_{28}, a_{29}, a_{31}, a_{30})$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)$
is a "Toffoli gate" $=$
"controlled controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1 q_2, q_1, q_2)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6)$
is a "reversible XOR gate" =
"controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1, q_1, q_2)$.

Example with more qubits:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$\mapsto (a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6,$
$a_8, a_9, a_{11}, a_{10}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{16}, a_{17}, a_{19}, a_{18}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{24}, a_{25}, a_{27}, a_{26}, a_{28}, a_{29}, a_{31}, a_{30})$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)$
is a "Toffoli gate" =
"controlled controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1 q_2, q_1, q_2)$.

Example with more qubits:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$\mapsto (a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{31}, a_{30})$.

$a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$a_3, a_2, a_4, a_5, a_7, a_6)$
versible XOR gate" $=$
lled NOT gate":
$q_2) \mapsto (q_0 \oplus q_1, q_1, q_2)$.

e with more qubits:
$a_2, a_3, a_4, a_5, a_6, a_7,$
$_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$_1, a_3, a_2, a_4, a_5, a_7, a_6,$
$_{11}, a_{10}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{19}, a_{18}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{27}, a_{26}, a_{28}, a_{29}, a_{31}, a_{30})$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)$
is a "Toffoli gate" $=$
"controlled controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1 q_2, q_1, q_2)$.

Example with more qubits:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$\mapsto (a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{31}, a_{30})$.

Reversib

Say p is
of $\{0, 1,$

General
these fas
to obtai
$(a_0, a_1, .$
$(a_{p^{-1}(0)}$

$a_5, a_6, a_7) \mapsto$

$a_5, a_7, a_6)$

OR gate" =

gate":

$\oplus q_1, q_1, q_2)$.

e qubits:

$a_5, a_6, a_7,$

$, a_{13}, a_{14}, a_{15},$

$_{20}, a_{21}, a_{22}, a_{23},$

$_{28}, a_{29}, a_{30}, a_{31})$

$_4, a_5, a_7, a_6,$

$, a_{13}, a_{15}, a_{14},$

$_{20}, a_{21}, a_{23}, a_{22},$

$_{28}, a_{29}, a_{31}, a_{30})$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)$
is a "Toffoli gate" =
"controlled controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1 q_2, q_1, q_2)$.

Example with more qubits:

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$\mapsto (a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{31}, a_{30})$.

Reversible comput

Say p is a permuta
of $\{0, 1, \ldots, 2^n -$

General strategy to
these fast quantum
to obtain index pe
$(a_0, a_1, \ldots, a_{2^n-1})$
$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots$

$\mapsto$

$=$

$q_2)$.

a_{15},

$2, a_{23}$,

$0, a_{31})$

6,

a_{14},

$3, a_{22}$,

$1, a_{30})$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)$
is a "Toffoli gate" $=$
"controlled controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1 q_2, q_1, q_2)$.

Example with more qubits:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$\mapsto (a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{31}, a_{30})$.

Reversible computation

Say p is a permutation
of $\{0, 1, \ldots, 2^n - 1\}$.

General strategy to compose
these fast quantum operatio
to obtain index permutation
$(a_0, a_1, \ldots, a_{2^n-1}) \mapsto$
$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n$

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)$
is a "Toffoli gate" =
"controlled controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1 q_2, q_1, q_2)$.

Example with more qubits:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$\mapsto (a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{31}, a_{30})$.

Reversible computation

Say p is a permutation
of $\{0, 1, \ldots, 2^n - 1\}$.

General strategy to compose
these fast quantum operations
to obtain index permutation
$(a_0, a_1, \ldots, a_{2^n-1}) \mapsto$
$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$:

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)$
is a "Toffoli gate" =
"controlled controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1 q_2, q_1, q_2)$.

Example with more qubits:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$\mapsto (a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{31}, a_{30})$.

Reversible computation

Say p is a permutation
of $\{0, 1, \ldots, 2^n - 1\}$.

General strategy to compose
these fast quantum operations
to obtain index permutation
$(a_0, a_1, \ldots, a_{2^n-1}) \mapsto$
$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$:

1. Build a traditional circuit
to compute $j \mapsto p(j)$
using NOT/XOR/AND gates.

2. Convert into reversible gates:
e.g., convert AND into Toffoli.

$a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$

$a_2, a_3, a_4, a_5, a_7, a_6)$

ffoli gate" =

led controlled NOT gate":

$q_2) \mapsto (q_0 \oplus q_1 q_2, q_1, q_2)$.

e with more qubits:

$a_2, a_3, a_4, a_5, a_6, a_7,$

$_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$

$a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$

$a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$

$_1, a_2, a_3, a_4, a_5, a_7, a_6,$

$_{10}, a_{11}, a_{12}, a_{13}, a_{15}, a_{14},$

$a_{18}, a_{19}, a_{20}, a_{21}, a_{23}, a_{22},$

$a_{26}, a_{27}, a_{28}, a_{29}, a_{31}, a_{30})$.

Reversible computation

Say p is a permutation of $\{0, 1, \ldots, 2^n - 1\}$.

General strategy to compose these fast quantum operations to obtain index permutation $(a_0, a_1, \ldots, a_{2^n-1}) \mapsto (a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$:

1. Build a traditional circuit to compute $j \mapsto p(j)$ using NOT/XOR/AND gates.

2. Convert into reversible gates: e.g., convert AND into Toffoli.

Example

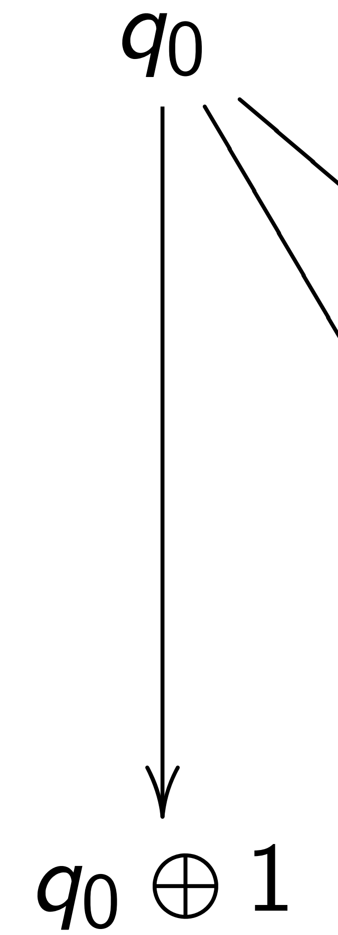
$(a_0, a_1,$

$(a_7, a_0,$

permuta

1. Build

to comp



$a_5, a_6, a_7) \mapsto$

$a_5, a_7, a_6)$

$=$

lled NOT gate":

$\oplus\, q_1 q_2, q_1, q_2)$.

e qubits:

$a_5, a_6, a_7,$

$, a_{13}, a_{14}, a_{15},$

$_{20}, a_{21}, a_{22}, a_{23},$

$_{28}, a_{29}, a_{30}, a_{31})$

$_4, a_5, a_7, a_6,$

$, a_{13}, a_{15}, a_{14},$

$_{20}, a_{21}, a_{23}, a_{22},$

$_{28}, a_{29}, a_{31}, a_{30})$.

<u>Reversible computation</u>

Say p is a permutation of $\{0, 1, \ldots, 2^n - 1\}$.

General strategy to compose these fast quantum operations to obtain index permutation $(a_0, a_1, \ldots, a_{2^n-1}) \mapsto (a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$:

1. Build a traditional circuit to compute $j \mapsto p(j)$ using NOT/XOR/AND gates.

2. Convert into reversible gates: e.g., convert AND into Toffoli.

Example: Let's co

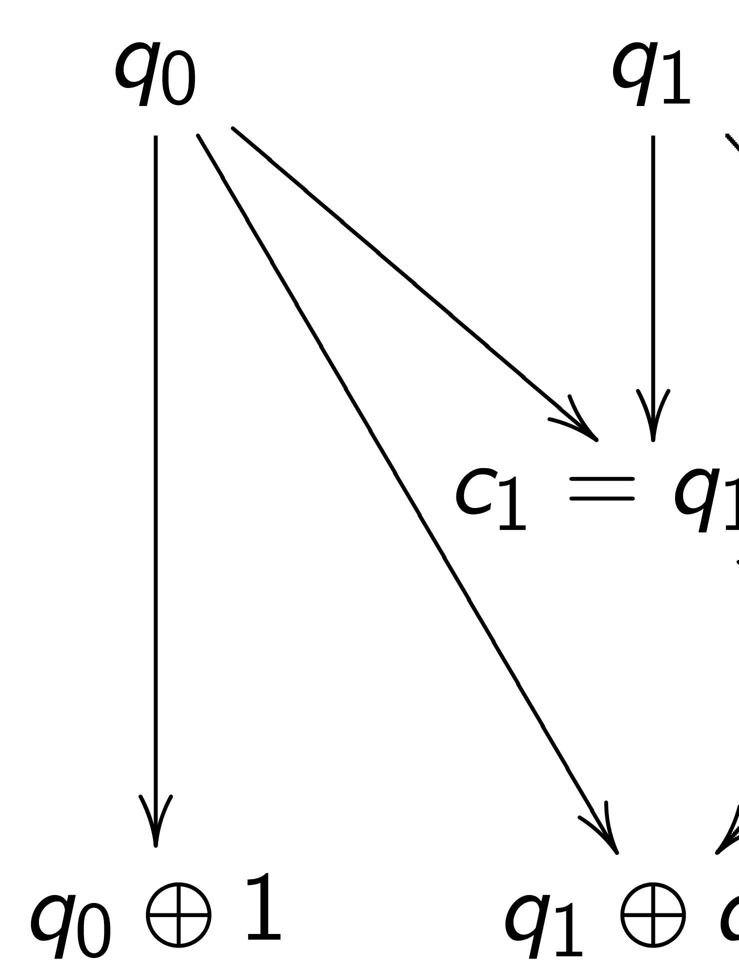
$(a_0, a_1, a_2, a_3, a_4, a$

$(a_7, a_0, a_1, a_2, a_3, a$

permutation $q \mapsto$

1. Build a traditio

to compute $q \mapsto q$



$\mapsto$

gate":

$_1, q_2)$.

a_{15},

$_2, a_{23}$,

$_0, a_{31})$

$_6$,

a_{14},

$_3, a_{22}$,

$_1, a_{30})$.

Reversible computation

Say p is a permutation of $\{0, 1, \ldots, 2^n - 1\}$.

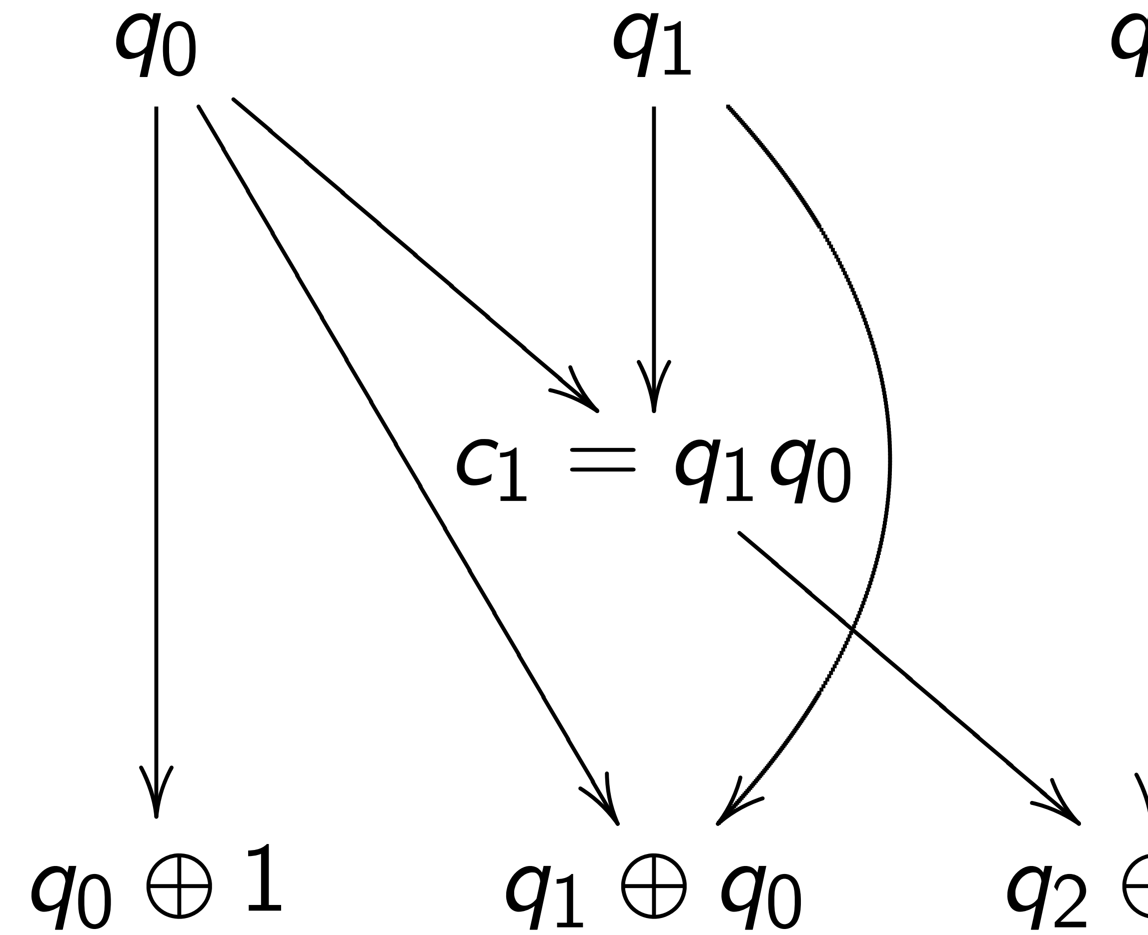
General strategy to compose these fast quantum operations to obtain index permutation $(a_0, a_1, \ldots, a_{2^n-1}) \mapsto (a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$:

1. Build a traditional circuit to compute $j \mapsto p(j)$ using NOT/XOR/AND gates.

2. Convert into reversible gates: e.g., convert AND into Toffoli.

Example: Let's compute $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$;
permutation $q \mapsto q + 1$ mod

1. Build a traditional circuit to compute $q \mapsto q + 1$ mod



Reversible computation

Say p is a permutation of $\{0, 1, \ldots, 2^n - 1\}$.

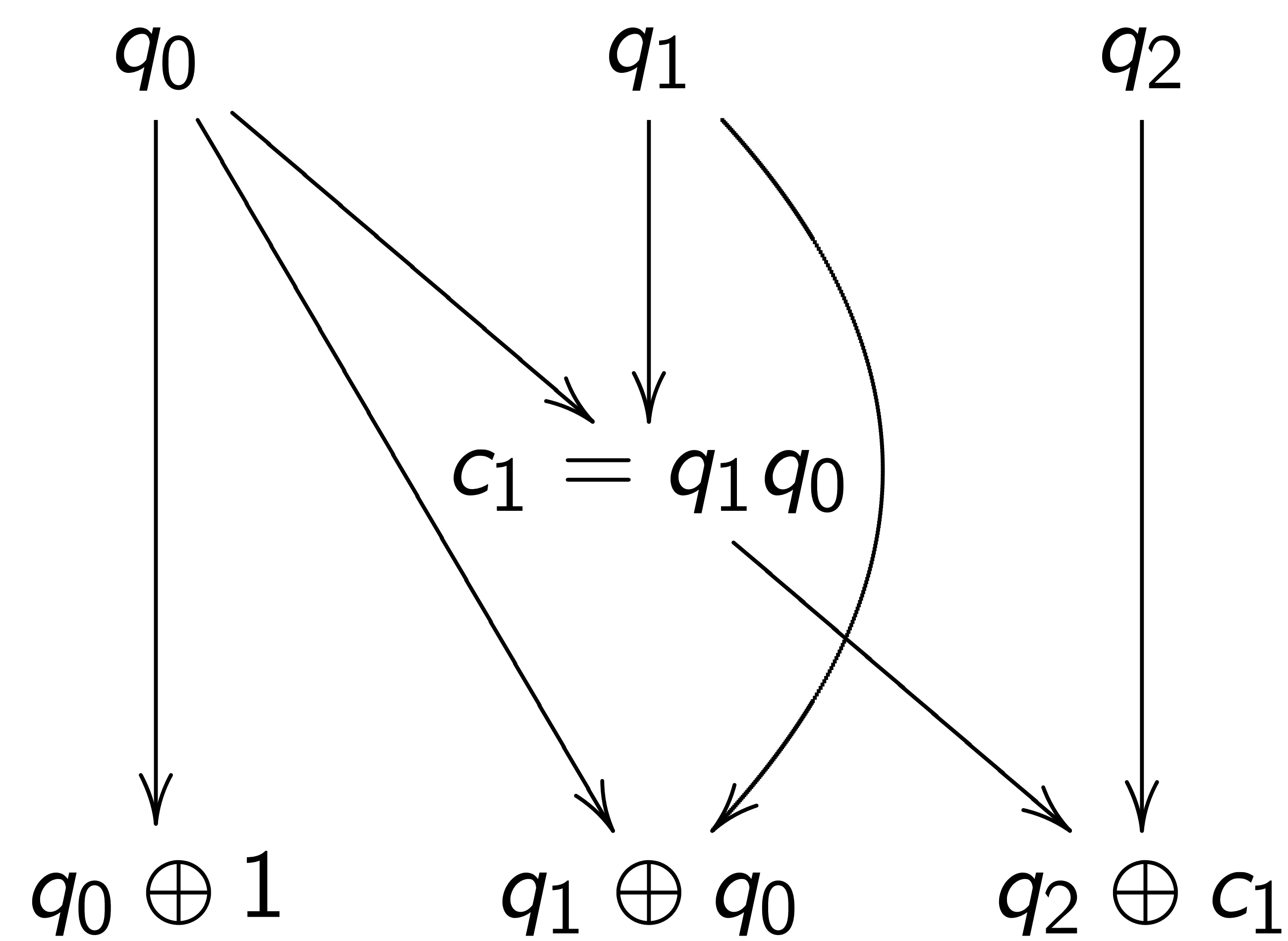
General strategy to compose these fast quantum operations to obtain index permutation $(a_0, a_1, \ldots, a_{2^n-1}) \mapsto (a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$:

1. Build a traditional circuit to compute $j \mapsto p(j)$ using NOT/XOR/AND gates.

2. Convert into reversible gates: e.g., convert AND into Toffoli.

Example: Let's compute $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto (a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$; permutation $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit to compute $q \mapsto q + 1 \bmod 8$.



le computation

a permutation
$\ldots, 2^n - 1\}$.

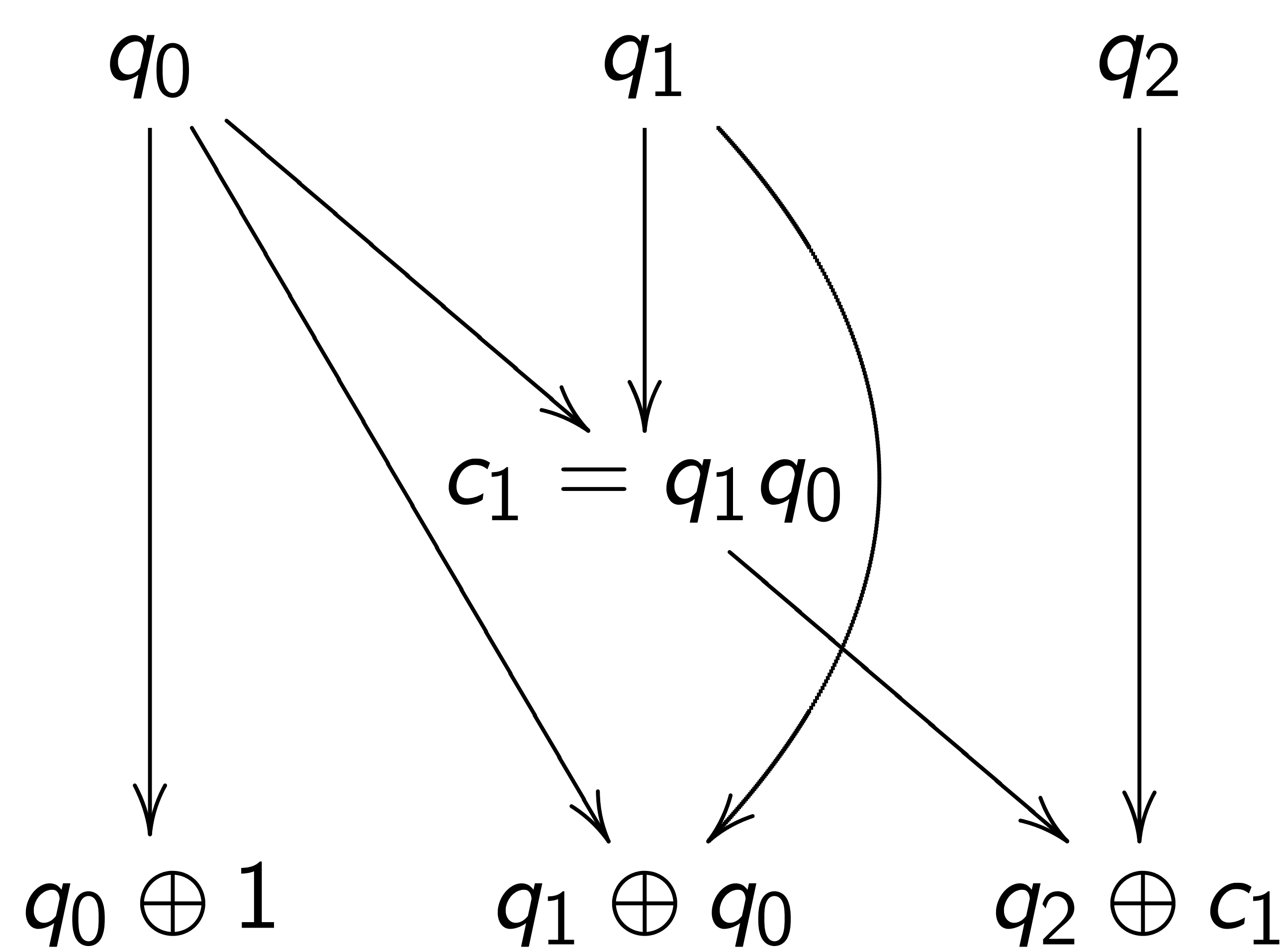
strategy to compose
st quantum operations
n index permutation
$\ldots, a_{2^n-1}) \mapsto$
$, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$:

a traditional circuit
ute $j \mapsto p(j)$
OT/XOR/AND gates.

ert into reversible gates:
nvert AND into Toffoli.

Example: Let's compute
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$;
permutation $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit
to compute $q \mapsto q + 1 \bmod 8$.



2. Conv

Toffoli f
$(a_0, a_1, a$
$(a_0, a_1, a$

ation

ation
1}.

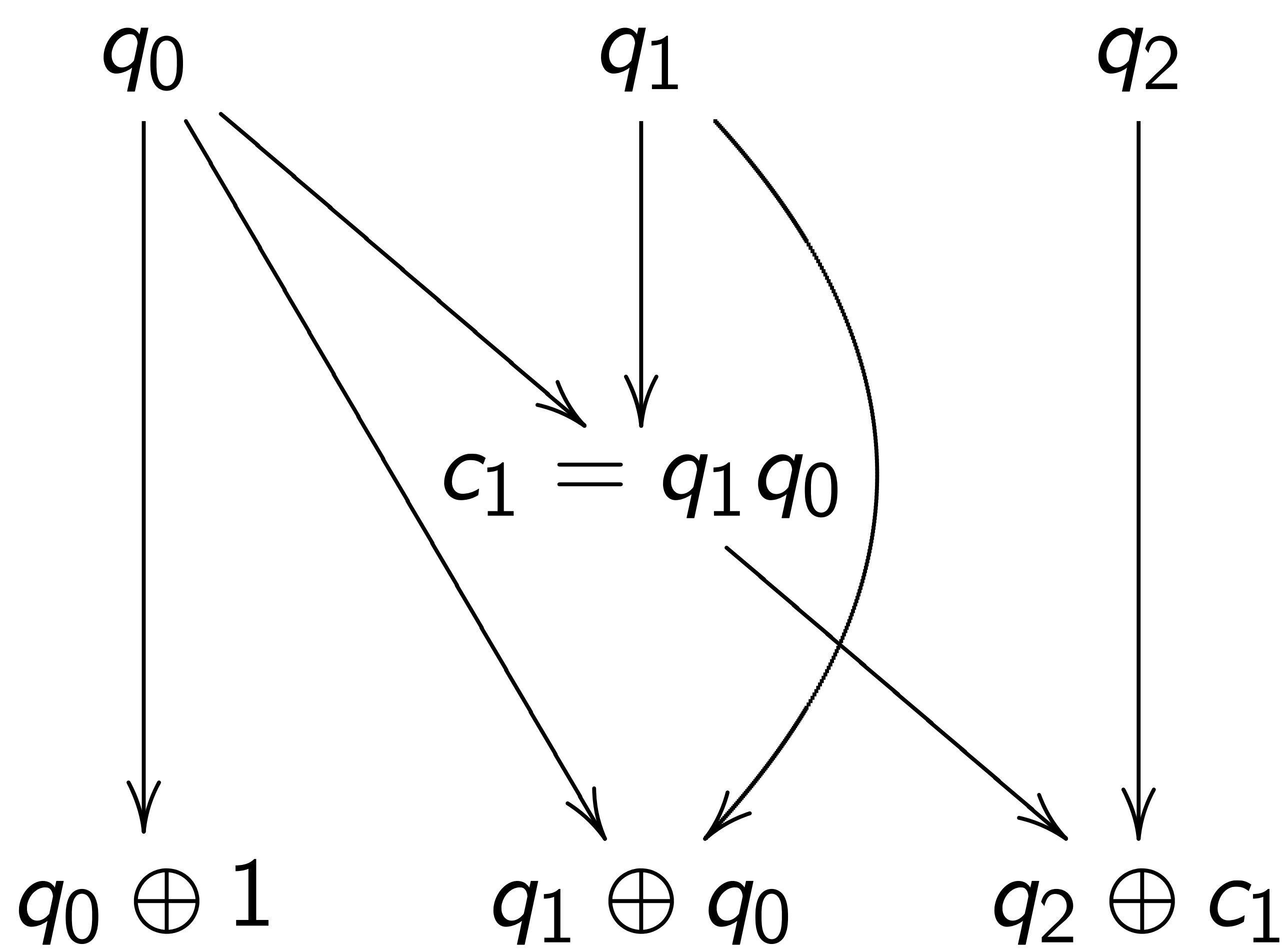
o compose
m operations
ermutation
) $\mapsto$
$\ldots, a_{p^{-1}(2^n-1)})$:

onal circuit
(j)
AND gates.

versible gates:
into Toffoli.

Example: Let's compute
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$;
permutation $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit
to compute $q \mapsto q + 1 \bmod 8$.



2. Convert into re

Toffoli for $q_2 \leftarrow q$
$(a_0, a_1, a_2, a_3, a_4, a$
$(a_0, a_1, a_2, a_7, a_4, a$

e

ons

$_{n-1})$):

:

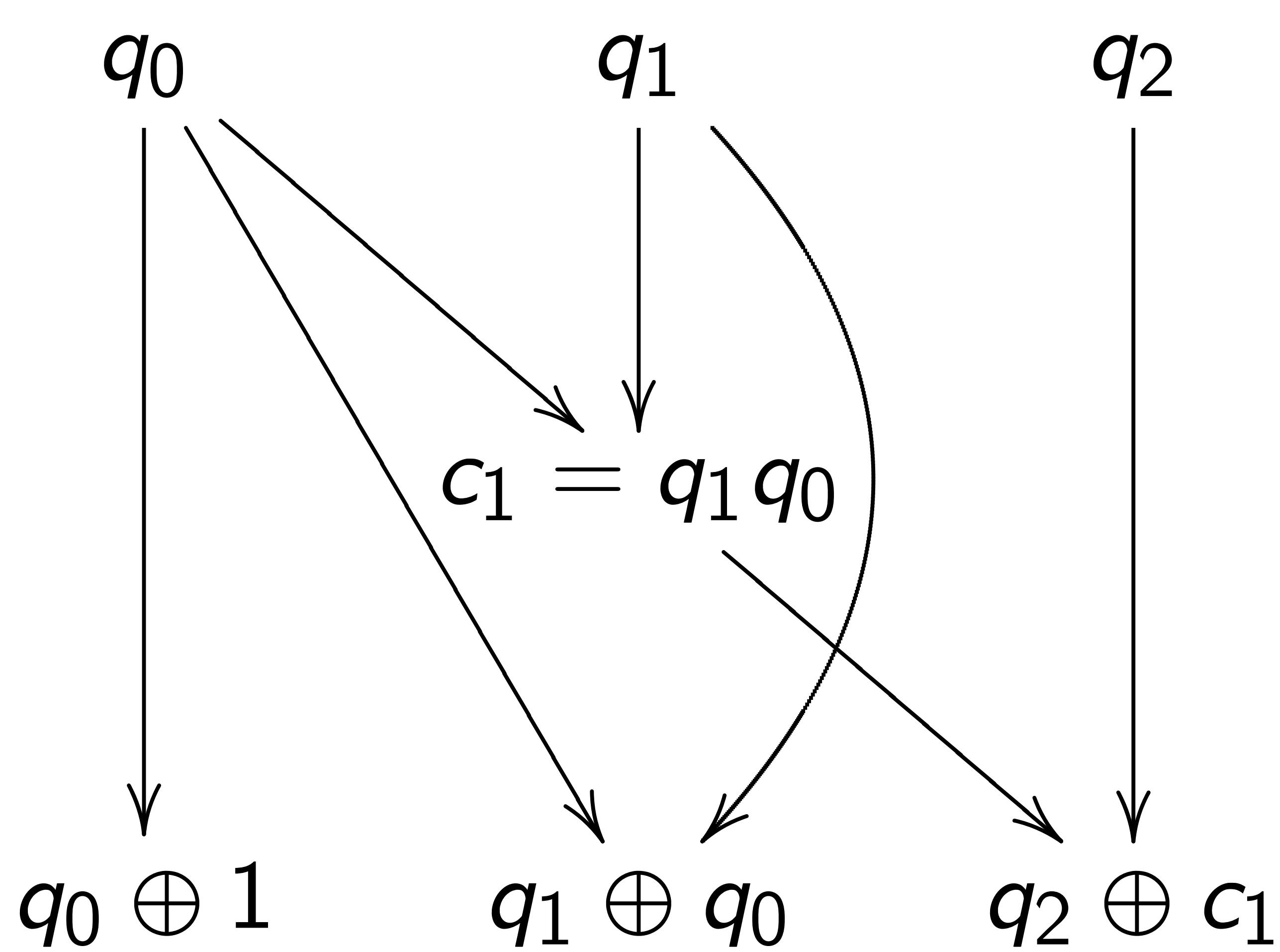
es.

ates:

oli.

Example: Let's compute
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$;
permutation $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit
to compute $q \mapsto q + 1 \bmod 8$.



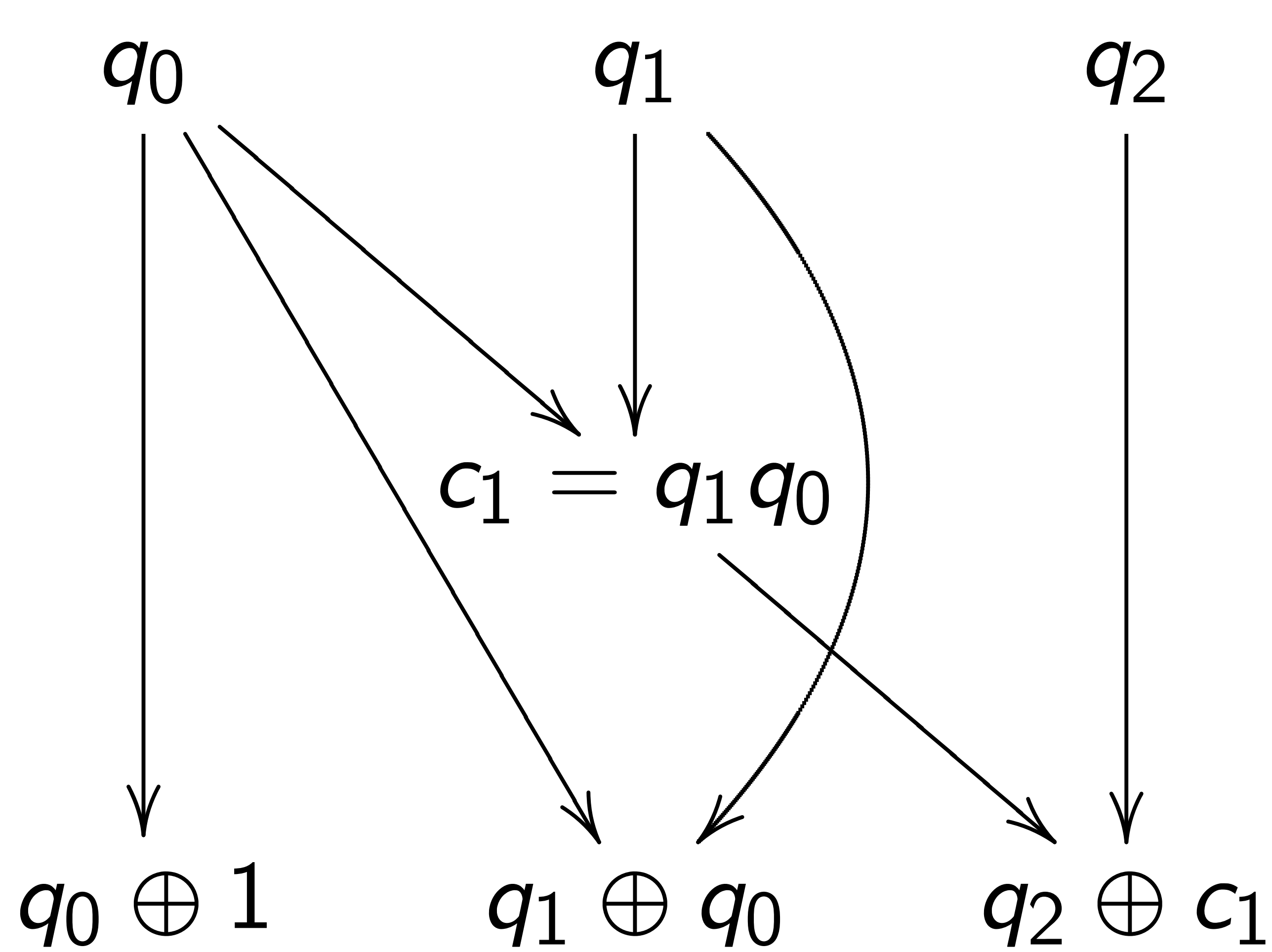
2. Convert into reversible ga

Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3)$.

Example: Let's compute

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$;
permutation $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit to compute $q \mapsto q + 1 \bmod 8$.

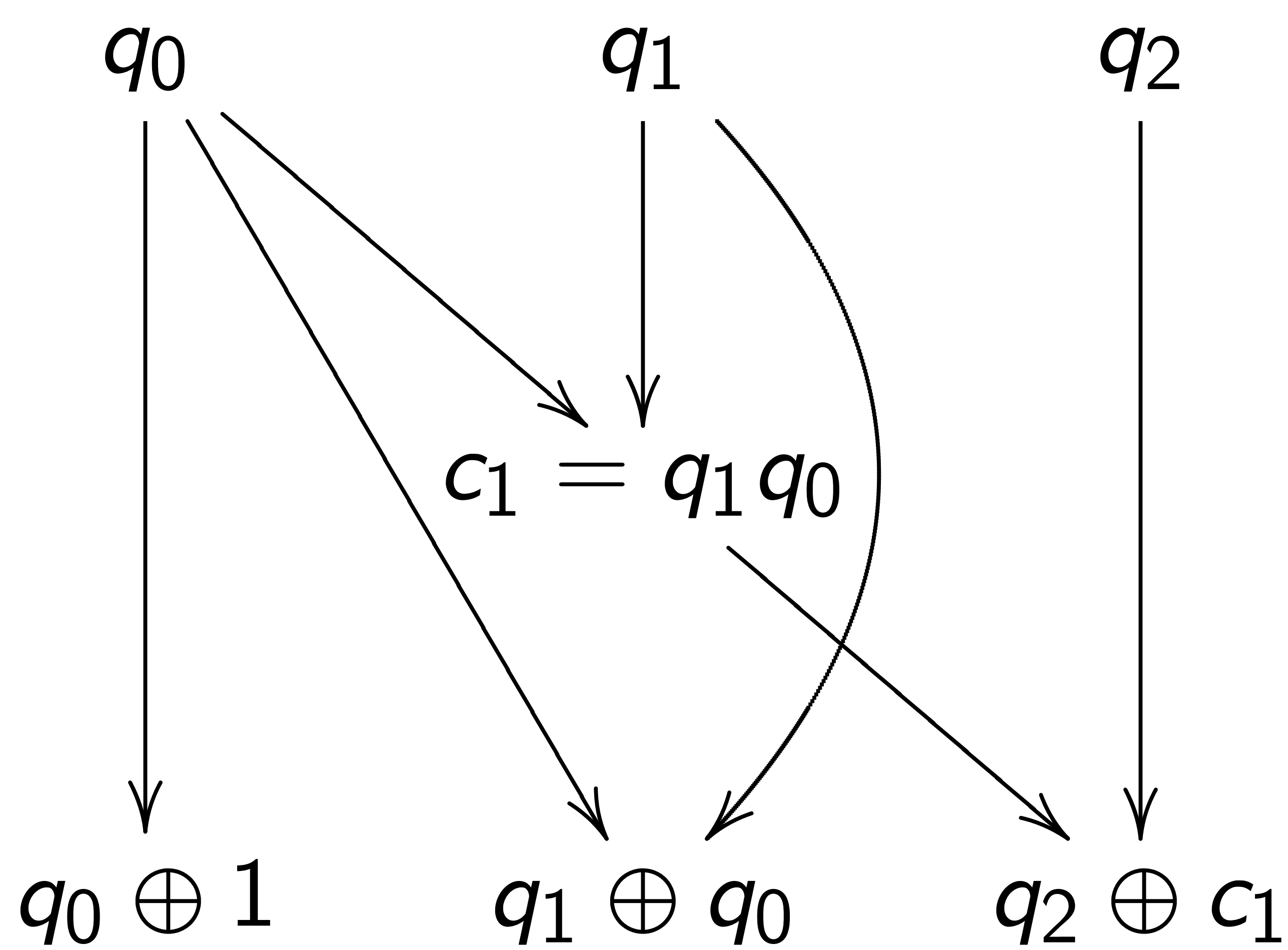


2. Convert into reversible gates.

Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3)$.

Example: Let's compute
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$;
permutation $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit to compute $q \mapsto q + 1 \bmod 8$.



2. Convert into reversible gates.

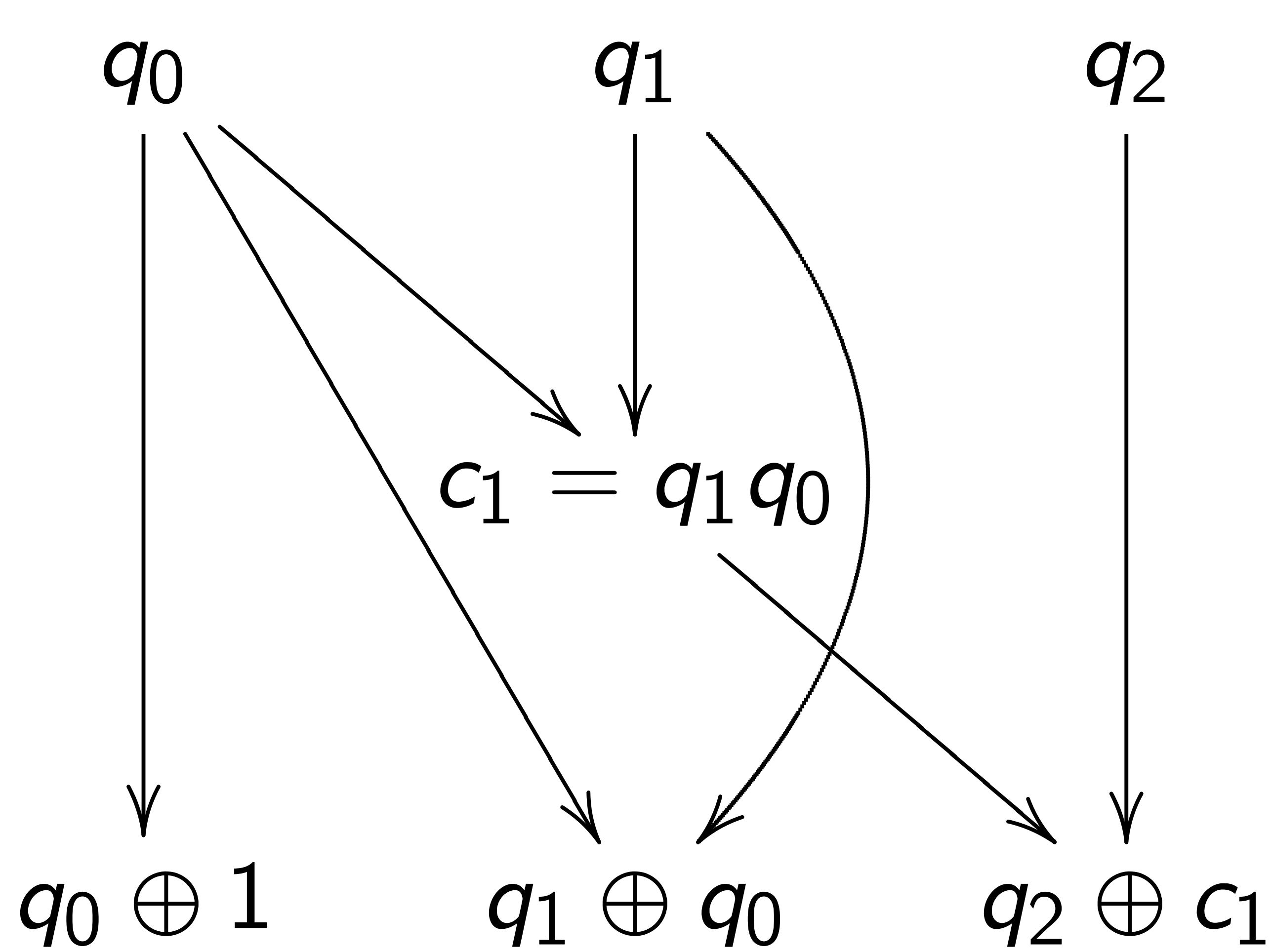
Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3)$.

Controlled NOT for $q_1 \leftarrow q_1 \oplus q_0$:
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3) \mapsto$
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5)$.

Example: Let's compute

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$;
permutation $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit to compute $q \mapsto q + 1 \bmod 8$.



2. Convert into reversible gates.

Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3)$.

Controlled NOT for $q_1 \leftarrow q_1 \oplus q_0$:
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3) \mapsto$
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5)$.

NOT for $q_0 \leftarrow q_0 \oplus 1$:
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$.

e: Let's compute

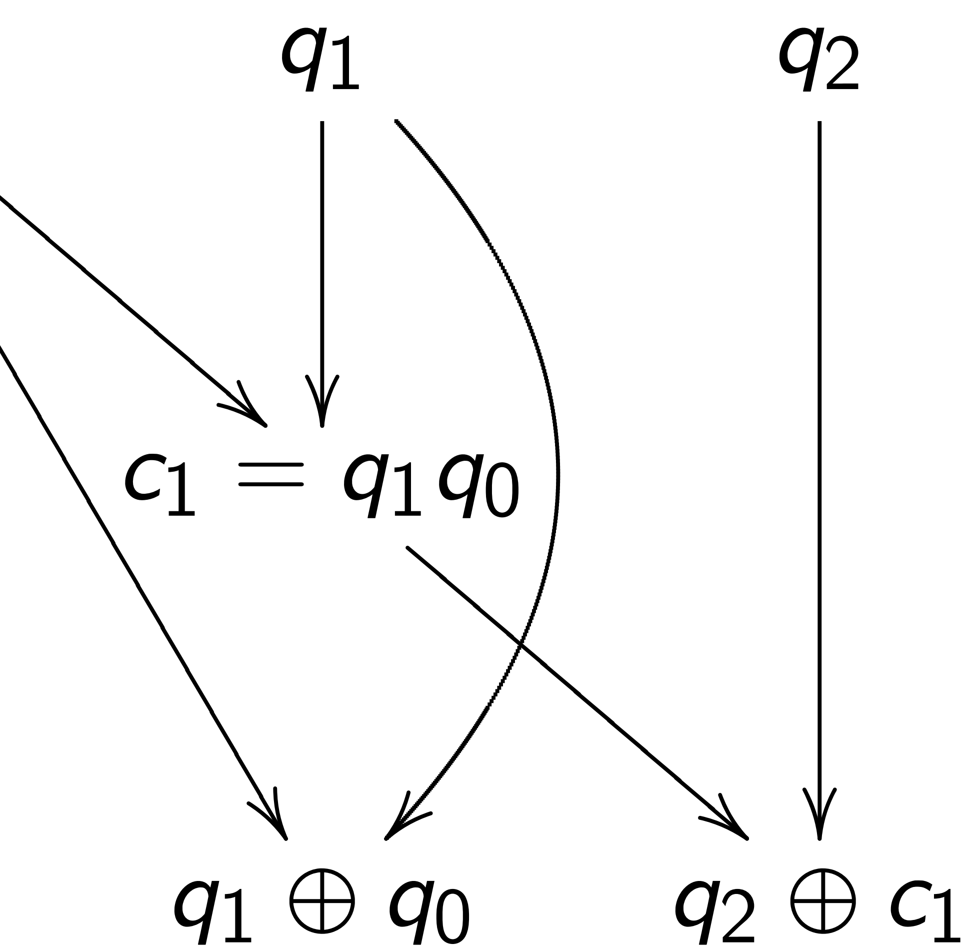
$a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$

$a_1, a_2, a_3, a_4, a_5, a_6)$;

tion $q \mapsto q + 1 \bmod 8$.

a traditional circuit

ute $q \mapsto q + 1 \bmod 8$.



2. Convert into reversible gates.

Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3)$.

Controlled NOT for $q_1 \leftarrow q_1 \oplus q_0$:
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3) \mapsto$
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5)$.

NOT for $q_0 \leftarrow q_0 \oplus 1$:
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$.

This per

was dece

It didn't

For large

need ma

Really w

mpute

$a_5, a_6, a_7) \mapsto$

$a_4, a_5, a_6)$;

$q + 1 \bmod 8$.

nal circuit

$q + 1 \bmod 8$.

2. Convert into reversible gates.

Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3)$.

Controlled NOT for $q_1 \leftarrow q_1 \oplus q_0$:
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3) \mapsto$
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5)$.

NOT for $q_0 \leftarrow q_0 \oplus 1$:
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$.

This permutation
was deceptively ea

It didn't need mar

For large *n*, most
need many operat
Really want *fast* c

$\to$

d 8.

:

8.

q_2

$\oplus c_1$

2. Convert into reversible gates.

Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3)$.

Controlled NOT for $q_1 \leftarrow q_1 \oplus q_0$:
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3) \mapsto$
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5)$.

NOT for $q_0 \leftarrow q_0 \oplus 1$:
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$.

This permutation example
was deceptively easy.

It didn't need many operatio

For large n, most permutatio
need many operations $\Rightarrow$ slo
Really want *fast* circuits.

2. Convert into reversible gates.

Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3)$.

Controlled NOT for $q_1 \leftarrow q_1 \oplus q_0$:
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3) \mapsto$
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5)$.

NOT for $q_0 \leftarrow q_0 \oplus 1$:
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$.

This permutation example was deceptively easy.

It didn't need many operations.

For large n, most permutations p need many operations $\Rightarrow$ slow. Really want *fast* circuits.

2. Convert into reversible gates.

Toffoli for $q_2 \leftarrow q_2 \oplus q_1 q_0$:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3)$.

Controlled NOT for $q_1 \leftarrow q_1 \oplus q_0$:
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3) \mapsto$
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5)$.

NOT for $q_0 \leftarrow q_0 \oplus 1$:
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$.

This permutation example was deceptively easy.

It didn't need many operations.

For large n, most permutations p need many operations $\Rightarrow$ slow.
Really want *fast* circuits.

Also, it didn't need extra storage: circuit operated "in place" after computation $c_1 \leftarrow q_1 q_0$ was merged into $q_2 \leftarrow q_2 \oplus c_1$.

Typical circuits aren't in-place.

ert into reversible gates.

or $q_2 \leftarrow q_2 \oplus q_1 q_0$:

$a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$

$a_2, a_7, a_4, a_5, a_6, a_3)$.

ed NOT for $q_1 \leftarrow q_1 \oplus q_0$:

$a_2, a_7, a_4, a_5, a_6, a_3) \mapsto$

$a_2, a_1, a_4, a_3, a_6, a_5)$.

r $q_0 \leftarrow q_0 \oplus 1$:

$a_2, a_1, a_4, a_3, a_6, a_5) \mapsto$

$a_1, a_2, a_3, a_4, a_5, a_6)$.

This permutation example was deceptively easy.

It didn't need many operations.

For large n, most permutations p need many operations $\Rightarrow$ slow. Really want *fast* circuits.

Also, it didn't need extra storage: circuit operated "in place" after computation $c_1 \leftarrow q_1 q_0$ was merged into $q_2 \leftarrow q_2 \oplus c_1$.

Typical circuits aren't in-place.

Start fro

inputs b

$b_{i+1} = 1$

$b_{i+2} = 1$

...

$b_T = 1 \oplus$

specified

versible gates.

$_2 \oplus q_1 q_0$:

$a_5, a_6, a_7) \mapsto$

$a_5, a_6, a_3)$.

or $q_1 \leftarrow q_1 \oplus q_0$:

$a_5, a_6, a_3) \mapsto$

$a_3, a_6, a_5)$.

$\oplus 1$:

$a_3, a_6, a_5) \mapsto$

$a_4, a_5, a_6)$.

This permutation example
was deceptively easy.

It didn't need many operations.

For large n, most permutations p
need many operations $\Rightarrow$ slow.
Really want *fast* circuits.

Also, it didn't need extra storage:
circuit operated "in place" after
computation $c_1 \leftarrow q_1 q_0$ was
merged into $q_2 \leftarrow q_2 \oplus c_1$.

Typical circuits aren't in-place.

Start from any cir
inputs $b_1, b_2, \ldots,$
$b_{i+1} = 1 \oplus b_{f(i+1)}$
$b_{i+2} = 1 \oplus b_{f(i+2)}$
...
$b_T = 1 \oplus b_{f(T)} b_{g($
specified outputs.

ates.

$\to$

$_1 \oplus q_0$:

$\to$

$\to$

This permutation example was deceptively easy.

It didn't need many operations.

For large n, most permutations p need many operations $\Rightarrow$ slow. Really want *fast* circuits.

Also, it didn't need extra storage: circuit operated "in place" after computation $c_1 \leftarrow q_1 q_0$ was merged into $q_2 \leftarrow q_2 \oplus c_1$.

Typical circuits aren't in-place.

Start from any circuit:
inputs $b_1, b_2, \ldots, b_i$;
$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;
$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;
$\ldots$
$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;
specified outputs.

This permutation example was deceptively easy.

It didn't need many operations.

For large n, most permutations p need many operations $\Rightarrow$ slow. Really want *fast* circuits.

Also, it didn't need extra storage: circuit operated "in place" after computation $c_1 \leftarrow q_1 q_0$ was merged into $q_2 \leftarrow q_2 \oplus c_1$.

Typical circuits aren't in-place.

Start from any circuit:

inputs $b_1, b_2, \ldots, b_i$;

$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;

...

$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;

specified outputs.

This permutation example was deceptively easy.

It didn't need many operations.

For large n, most permutations p need many operations $\Rightarrow$ slow. Really want *fast* circuits.

Also, it didn't need extra storage: circuit operated "in place" after computation $c_1 \leftarrow q_1 q_0$ was merged into $q_2 \leftarrow q_2 \oplus c_1$.

Typical circuits aren't in-place.

Start from any circuit:
inputs $b_1, b_2, \ldots, b_i$;
$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;
$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;
...
$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;
specified outputs.

Reversible but dirty:
inputs $b_1, b_2, \ldots, b_T$;
$b_{i+1} \leftarrow 1 \oplus b_{i+1} \oplus b_{f(i+1)} b_{g(i+1)}$;
$b_{i+2} \leftarrow 1 \oplus b_{i+2} \oplus b_{f(i+2)} b_{g(i+2)}$;
...
$b_T \leftarrow 1 \oplus b_T \oplus b_{f(T)} b_{g(T)}$.
Same outputs if all of $b_{i+1}, \ldots, b_T$ started as 0.

rmutation example
eptively easy.

need many operations.

e n, most permutations p
ny operations $\Rightarrow$ slow.
vant *fast* circuits.

didn't need extra storage:
perated "in place" after
ation $c_1 \leftarrow q_1 q_0$ was
into $q_2 \leftarrow q_2 \oplus c_1$.

circuits aren't in-place.

Start from any circuit:
inputs $b_1, b_2, \ldots, b_i$;
$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;
$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;
...
$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;
specified outputs.

Reversible but dirty:
inputs $b_1, b_2, \ldots, b_T$;
$b_{i+1} \leftarrow 1 \oplus b_{i+1} \oplus b_{f(i+1)} b_{g(i+1)}$;
$b_{i+2} \leftarrow 1 \oplus b_{i+2} \oplus b_{f(i+2)} b_{g(i+2)}$;
...
$b_T \leftarrow 1 \oplus b_T \oplus b_{f(T)} b_{g(T)}$.
Same outputs if all of
$b_{i+1}, \ldots, b_T$ started as 0.

Reversib
after fini
set non-
by repea
on non-c

Original
(inputs)
(inputs, c

Dirty rev
(inputs, z
(inputs, c

Clean re
(inputs, z
(inputs, z

example
sy.

y operations.

permutations p
ions $\Rightarrow$ slow.
ircuits.

d extra storage:
n place" after
$q_1 q_0$ was
$q_2 \oplus c_1$.

en't in-place.

Start from any circuit:

inputs $b_1, b_2, \ldots, b_i$;

$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;

...

$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;

specified outputs.

Reversible but dirty:

inputs $b_1, b_2, \ldots, b_T$;

$b_{i+1} \leftarrow 1 \oplus b_{i+1} \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} \leftarrow 1 \oplus b_{i+2} \oplus b_{f(i+2)} b_{g(i+2)}$;

...

$b_T \leftarrow 1 \oplus b_T \oplus b_{f(T)} b_{g(T)}$.

Same outputs if all of

$b_{i+1}, \ldots, b_T$ started as 0.

Reversible and clea
after finishing dirt
set non-outputs ba
by repeating same
on non-outputs in

Original computat
(inputs) $\mapsto$
(inputs, dirt, outpu

Dirty reversible co
(inputs, zeros, zero
(inputs, dirt, outpu

Clean reversible co
(inputs, zeros, zero
(inputs, zeros, outp

ons.

ons p

ow.

orage:

after

s

ce.

Start from any circuit:

inputs $b_1, b_2, \ldots, b_i$;

$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;

...

$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;

specified outputs.

Reversible but dirty:

inputs $b_1, b_2, \ldots, b_T$;

$b_{i+1} \leftarrow 1 \oplus b_{i+1} \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} \leftarrow 1 \oplus b_{i+2} \oplus b_{f(i+2)} b_{g(i+2)}$;

...

$b_T \leftarrow 1 \oplus b_T \oplus b_{f(T)} b_{g(T)}$.

Same outputs if all of

$b_{i+1}, \ldots, b_T$ started as 0.

Reversible and clean:

after finishing dirty computa

set non-outputs back to 0,

by repeating same operation

on non-outputs in reverse or

Original computation:

(inputs) $\mapsto$

(inputs, dirt, outputs).

Dirty reversible computation

(inputs, zeros, zeros) $\mapsto$

(inputs, dirt, outputs).

Clean reversible computatio

(inputs, zeros, zeros) $\mapsto$

(inputs, zeros, outputs).

Start from any circuit:

inputs $b_1, b_2, \ldots, b_i$;

$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;

$\ldots$

$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;

specified outputs.

Reversible but dirty:

inputs $b_1, b_2, \ldots, b_T$;

$b_{i+1} \leftarrow 1 \oplus b_{i+1} \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} \leftarrow 1 \oplus b_{i+2} \oplus b_{f(i+2)} b_{g(i+2)}$;

$\ldots$

$b_T \leftarrow 1 \oplus b_T \oplus b_{f(T)} b_{g(T)}$.

Same outputs if all of

$b_{i+1}, \ldots, b_T$ started as 0.

Reversible and clean:

after finishing dirty computation,

set non-outputs back to 0,

by repeating same operations

on non-outputs in reverse order.

Original computation:

$(\text{inputs}) \mapsto$

$(\text{inputs}, \text{dirt}, \text{outputs})$.

Dirty reversible computation:

$(\text{inputs}, \text{zeros}, \text{zeros}) \mapsto$

$(\text{inputs}, \text{dirt}, \text{outputs})$.

Clean reversible computation:

$(\text{inputs}, \text{zeros}, \text{zeros}) \mapsto$

$(\text{inputs}, \text{zeros}, \text{outputs})$.

om any circuit:

$_1, b_2, \ldots, b_i$;

$1 \oplus b_{f(i+1)} b_{g(i+1)}$;

$1 \oplus b_{f(i+2)} b_{g(i+2)}$;

$\oplus\, b_{f(T)} b_{g(T)}$;

l outputs.

le but dirty:

$_1, b_2, \ldots, b_T$;

$1 \oplus b_{i+1} \oplus b_{f(i+1)} b_{g(i+1)}$;

$1 \oplus b_{i+2} \oplus b_{f(i+2)} b_{g(i+2)}$;

$\oplus\, b_T \oplus b_{f(T)} b_{g(T)}$.

utputs if all of

$, b_T$ started as 0.

Reversible and clean:
after finishing dirty computation,
set non-outputs back to 0,
by repeating same operations
on non-outputs in reverse order.

Original computation:
(inputs) $\mapsto$
(inputs, dirt, outputs).

Dirty reversible computation:
(inputs, zeros, zeros) $\mapsto$
(inputs, dirt, outputs).

Clean reversible computation:
(inputs, zeros, zeros) $\mapsto$
(inputs, zeros, outputs).

Given fa
and fast
build fas
$(x,$ zeros

cuit:

b_i;

$)\,b_{g(i+1)}$;

$)\,b_{g(i+2)}$;

(T);

ty:

b_T;

$\oplus\ b_{f(i+1)} b_{g(i+1)}$;

$\oplus\ b_{f(i+2)} b_{g(i+2)}$;

$_{f(T)} b_{g(T)}$.

ll of

ed as 0.

Reversible and clean:
after finishing dirty computation,
set non-outputs back to 0,
by repeating same operations
on non-outputs in reverse order.

Original computation:
(inputs) $\mapsto$
(inputs, dirt, outputs).

Dirty reversible computation:
(inputs, zeros, zeros) $\mapsto$
(inputs, dirt, outputs).

Clean reversible computation:
(inputs, zeros, zeros) $\mapsto$
(inputs, zeros, outputs).

Given fast circuit f

and fast circuit for

build fast reversibl

$(x, \text{zeros}) \mapsto (p(x)$

$g(i{+}1)$;
$g(i{+}2)$;

Reversible and clean:
after finishing dirty computation,
set non-outputs back to 0,
by repeating same operations
on non-outputs in reverse order.

Original computation:
(inputs) $\mapsto$
(inputs, dirt, outputs).

Dirty reversible computation:
(inputs, zeros, zeros) $\mapsto$
(inputs, dirt, outputs).

Clean reversible computation:
(inputs, zeros, zeros) $\mapsto$
(inputs, zeros, outputs).

Given fast circuit for p
and fast circuit for p^{-1},
build fast reversible circuit fo
$(x, \text{zeros}) \mapsto (p(x), \text{zeros})$.

Reversible and clean:
after finishing dirty computation,
set non-outputs back to 0,
by repeating same operations
on non-outputs in reverse order.

Original computation:
$(\text{inputs}) \mapsto$
$(\text{inputs}, \text{dirt}, \text{outputs})$.

Dirty reversible computation:
$(\text{inputs}, \text{zeros}, \text{zeros}) \mapsto$
$(\text{inputs}, \text{dirt}, \text{outputs})$.

Clean reversible computation:
$(\text{inputs}, \text{zeros}, \text{zeros}) \mapsto$
$(\text{inputs}, \text{zeros}, \text{outputs})$.

Given fast circuit for p
and fast circuit for p^{-1},
build fast reversible circuit for
$(x, \text{zeros}) \mapsto (p(x), \text{zeros})$.

Reversible and clean:
after finishing dirty computation,
set non-outputs back to 0,
by repeating same operations
on non-outputs in reverse order.

Original computation:
(inputs) $\mapsto$
(inputs, dirt, outputs).

Dirty reversible computation:
(inputs, zeros, zeros) $\mapsto$
(inputs, dirt, outputs).

Clean reversible computation:
(inputs, zeros, zeros) $\mapsto$
(inputs, zeros, outputs).

Given fast circuit for p
and fast circuit for p^{-1},
build fast reversible circuit for
$(x, \text{zeros}) \mapsto (p(x), \text{zeros})$.

Replace reversible bit operations
with Toffoli gates etc.
permuting $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

Permutation on first 2^n entries is
$(a_0, a_1, \ldots, a_{2^n-1}) \mapsto$
$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$.

Typically prepare vectors
supported on first 2^n entries
so don't care how permutation
acts on last $2^{n+z} - 2^n$ entries.

le and clean:
ishing dirty computation,
outputs back to 0,
ting same operations
outputs in reverse order.

computation:
$\mapsto$
dirt, outputs).

versible computation:
zeros, zeros) $\mapsto$
dirt, outputs).

versible computation:
zeros, zeros) $\mapsto$
zeros, outputs).

Given fast circuit for p
and fast circuit for p^{-1},
build fast reversible circuit for
$(x, \text{zeros}) \mapsto (p(x), \text{zeros})$.

Replace reversible bit operations
with Toffoli gates etc.
permuting $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

Permutation on first 2^n entries is
$(a_0, a_1, \ldots, a_{2^n-1}) \mapsto$
$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$.

Typically prepare vectors
supported on first 2^n entries
so don't care how permutation
acts on last $2^{n+z} - 2^n$ entries.

Warning
$\approx$ numb
in origin

This can
than nu
in the o

an:
y computation,
ack to 0,
operations
reverse order.

ion:

ts).

mputation:
s) $\mapsto$
ts).

omputation:
s) $\mapsto$
outs).

Given fast circuit for p
and fast circuit for p^{-1},
build fast reversible circuit for
$(x, \text{zeros}) \mapsto (p(x), \text{zeros})$.

Replace reversible bit operations
with Toffoli gates etc.
permuting $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

Permutation on first 2^n entries is
$(a_0, a_1, \ldots, a_{2^n-1}) \mapsto$
$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$.

Typically prepare vectors
supported on first 2^n entries
so don't care how permutation
acts on last $2^{n+z} - 2^n$ entries.

Warning: Number
$\approx$ number of **bit o**
in original p, p^{-1} c

This can be much
than number of **bi**
in the original circ

tion,

s

rder.

n:

n:

Given fast circuit for p
and fast circuit for p^{-1},
build fast reversible circuit for
$(x, \text{zeros}) \mapsto (p(x), \text{zeros})$.

Replace reversible bit operations
with Toffoli gates etc.
permuting $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

Permutation on first 2^n entries is
$(a_0, a_1, \ldots, a_{2^n-1}) \mapsto$
$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$.

Typically prepare vectors
supported on first 2^n entries
so don't care how permutation
acts on last $2^{n+z} - 2^n$ entries.

Warning: Number of **qubits**
$\approx$ number of **bit operations**
in original p, p^{-1} circuits.

This can be much larger
than number of **bits stored**
in the original circuits.

Given fast circuit for p
and fast circuit for p^{-1},
build fast reversible circuit for
$(x, \text{zeros}) \mapsto (p(x), \text{zeros})$.

Replace reversible bit operations
with Toffoli gates etc.
permuting $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

Permutation on first 2^n entries is
$(a_0, a_1, \ldots, a_{2^n-1}) \mapsto$
$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$.

Typically prepare vectors
supported on first 2^n entries
so don't care how permutation
acts on last $2^{n+z} - 2^n$ entries.

Warning: Number of **qubits**
$\approx$ number of **bit operations**
in original p, p^{-1} circuits.

This can be much larger
than number of **bits stored**
in the original circuits.

Given fast circuit for p
and fast circuit for p^{-1},
build fast reversible circuit for
$(x, \text{zeros}) \mapsto (p(x), \text{zeros})$.

Replace reversible bit operations
with Toffoli gates etc.
permuting $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

Permutation on first 2^n entries is
$(a_0, a_1, \ldots, a_{2^n-1}) \mapsto$
$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$.

Typically prepare vectors
supported on first 2^n entries
so don't care how permutation
acts on last $2^{n+z} - 2^n$ entries.

Warning: Number of **qubits**
$\approx$ number of **bit operations**
in original p, p^{-1} circuits.

This can be much larger
than number of **bits stored**
in the original circuits.

Many useful techniques
to compress into fewer qubits,
but often these lose time.
Many subtle tradeoffs.

Given fast circuit for p and fast circuit for p^{-1}, build fast reversible circuit for $(x, \text{zeros}) \mapsto (p(x), \text{zeros})$.

Replace reversible bit operations with Toffoli gates etc. permuting $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

Permutation on first 2^n entries is $(a_0, a_1, \ldots, a_{2^n-1}) \mapsto (a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$.

Typically prepare vectors supported on first 2^n entries so don't care how permutation acts on last $2^{n+z} - 2^n$ entries.

Warning: Number of **qubits** $\approx$ number of **bit operations** in original p, p^{-1} circuits.

This can be much larger than number of **bits stored** in the original circuits.

Many useful techniques to compress into fewer qubits, but often these lose time. Many subtle tradeoffs.

Crude "poly-time" analyses don't care about this, but serious cryptanalysis is much more precise.

st circuit for p
circuit for p^{-1},
st reversible circuit for
$) \mapsto (p(x), \text{zeros})$.

reversible bit operations
ffoli gates etc.
ng $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

ation on first 2^n entries is
$\ldots, a_{2^n-1}) \mapsto$
$, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$.

y prepare vectors
ed on first 2^n entries
care how permutation
last $2^{n+z} - 2^n$ entries.

Warning: Number of **qubits**
$\approx$ number of **bit operations**
in original p, p^{-1} circuits.

This can be much larger
than number of **bits stored**
in the original circuits.

Many useful techniques
to compress into fewer qubits,
but often these lose time.
Many subtle tradeoffs.

Crude "poly-time" analyses
don't care about this,
but serious cryptanalysis
is much more precise.

Fast qua

"Hadam
(a_0, a_1)

for p
p^{-1},
e circuit for
), zeros).

bit operations
etc.
$\to \mathbf{C}^{2^{n+z}}$.

rst 2^n entries is
) $\mapsto$
$\ldots, a_{p^{-1}(2^n-1)})$.

vectors
2^n entries
permutation
$- 2^n$ entries.

Warning: Number of **qubits**
$\approx$ number of **bit operations**
in original p, p^{-1} circuits.

This can be much larger
than number of **bits stored**
in the original circuits.

Many useful techniques
to compress into fewer qubits,
but often these lose time.
Many subtle tradeoffs.

Crude “poly-time” analyses
don’t care about this,
but serious cryptanalysis
is much more precise.

Fast quantum ope

“Hadamard”:
$(a_0, a_1) \mapsto (a_0 + a$

or

tions

ries is

$_{-1})$).

;

ion

es.

Warning: Number of **qubits** $\approx$ number of **bit operations** in original p, p^{-1} circuits.

This can be much larger than number of **bits stored** in the original circuits.

Many useful techniques to compress into fewer qubits, but often these lose time. Many subtle tradeoffs.

Crude "poly-time" analyses don't care about this, but serious cryptanalysis is much more precise.

Fast quantum operations, pa

"Hadamard":

$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1$

Warning: Number of **qubits**
$\approx$ number of **bit operations**
in original p, p^{-1} circuits.

This can be much larger
than number of **bits stored**
in the original circuits.

Many useful techniques
to compress into fewer qubits,
but often these lose time.
Many subtle tradeoffs.

Crude "poly-time" analyses
don't care about this,
but serious cryptanalysis
is much more precise.

Fast quantum operations, part 2

"Hadamard":
$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)$.

Warning: Number of **qubits** $\approx$ number of **bit operations** in original p, p^{-1} circuits.

This can be much larger than number of **bits stored** in the original circuits.

Many useful techniques to compress into fewer qubits, but often these lose time. Many subtle tradeoffs.

Crude "poly-time" analyses don't care about this, but serious cryptanalysis is much more precise.

Fast quantum operations, part 2

"Hadamard":
$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)$.

$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3)$.

Warning: Number of **qubits** $\approx$ number of **bit operations** in original p, p^{-1} circuits.

This can be much larger than number of **bits stored** in the original circuits.

Many useful techniques to compress into fewer qubits, but often these lose time. Many subtle tradeoffs.

Crude "poly-time" analyses don't care about this, but serious cryptanalysis is much more precise.

Fast quantum operations, part 2

"Hadamard":
$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)$.

$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3)$.

Same for qubit 1:
$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_2, a_1 + a_3, a_0 - a_2, a_1 - a_3)$.

Warning: Number of **qubits** $\approx$ number of **bit operations** in original p, p^{-1} circuits.

This can be much larger than number of **bits stored** in the original circuits.

Many useful techniques to compress into fewer qubits, but often these lose time. Many subtle tradeoffs.

Crude "poly-time" analyses don't care about this, but serious cryptanalysis is much more precise.

Fast quantum operations, part 2

"Hadamard":
$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)$.

$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3)$.

Same for qubit 1:
$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_2, a_1 + a_3, a_0 - a_2, a_1 - a_3)$.

Qubit 0 and then qubit 1:
$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0+a_1, a_0-a_1, a_2+a_3, a_2-a_3) \mapsto$
$(a_0 + a_1 + a_2 + a_3, a_0 - a_1 + a_2 - a_3,$
$a_0 + a_1 - a_2 - a_3, a_0 - a_1 - a_2 + a_3)$.

g: Number of **qubits**
er of **bit operations**
al p, p^{-1} circuits.

n be much larger
mber of **bits stored**
riginal circuits.

seful techniques
ress into fewer qubits,
n these lose time.
ubtle tradeoffs.

poly-time" analyses
re about this,
ous cryptanalysis
more precise.

Fast quantum operations, part 2

"Hadamard":
$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)$.

$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3)$.

Same for qubit 1:
$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_2, a_1 + a_3, a_0 - a_2, a_1 - a_3)$.

Qubit 0 and then qubit 1:
$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3) \mapsto$
$(a_0 + a_1 + a_2 + a_3, a_0 - a_1 + a_2 - a_3,$
$a_0 + a_1 - a_2 - a_3, a_0 - a_1 - a_2 + a_3)$.

Repeat
(1, 0, 0, .

Measurin
always p

Measurin
can proc
Pr[outpu

of **qubits**
operations
circuits.

larger
its stored
uits.

iques
ewer qubits,
se time.
offs.

analyses
his,
nalysis
ise.

<u>Fast quantum operations, part 2</u>

"Hadamard":
$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)$.

$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3)$.

Same for qubit 1:
$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_2, a_1 + a_3, a_0 - a_2, a_1 - a_3)$.

Qubit 0 and then qubit 1:
$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3) \mapsto$
$(a_0 + a_1 + a_2 + a_3, a_0 - a_1 + a_2 - a_3,$
$a_0 + a_1 - a_2 - a_3, a_0 - a_1 - a_2 + a_3)$.

Repeat n times: e
$(1, 0, 0, \ldots, 0) \mapsto$ (

Measuring $(1, 0, 0,$
always produces 0

Measuring $(1, 1, 1,$
can produce any o
$\Pr[\text{output} = q] =$

s

s

ts,

Fast quantum operations, part 2

"Hadamard":

$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)$.

$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3)$.

Same for qubit 1:
$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_2, a_1 + a_3, a_0 - a_2, a_1 - a_3)$.

Qubit 0 and then qubit 1:
$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3) \mapsto$
$(a_0 + a_1 + a_2 + a_3, a_0 - a_1 + a_2 - a_3,$
$a_0 + a_1 - a_2 - a_3, a_0 - a_1 - a_2 + a_3)$.

Repeat n times: e.g.,
$(1, 0, 0, \ldots, 0) \mapsto (1, 1, 1, \ldots$

Measuring $(1, 0, 0, \ldots, 0)$
always produces 0.

Measuring $(1, 1, 1, \ldots, 1)$
can produce any output:
$\Pr[\text{output} = q] = 1/2^n$.

Fast quantum operations, part 2

"Hadamard":

$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)$.

$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3)$.

Same for qubit 1:
$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_2, a_1 + a_3, a_0 - a_2, a_1 - a_3)$.

Qubit 0 and then qubit 1:
$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0+a_1, a_0-a_1, a_2+a_3, a_2-a_3) \mapsto$
$(a_0 + a_1 + a_2 + a_3, a_0 - a_1 + a_2 - a_3,$
$a_0 + a_1 - a_2 - a_3, a_0 - a_1 - a_2 + a_3)$.

Repeat n times: e.g.,
$(1, 0, 0, \ldots, 0) \mapsto (1, 1, 1, \ldots, 1)$.

Measuring $(1, 0, 0, \ldots, 0)$
always produces 0.

Measuring $(1, 1, 1, \ldots, 1)$
can produce any output:
$\Pr[\text{output} = q] = 1/2^n$.

Fast quantum operations, part 2

"Hadamard":

$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1)$.

$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3)$.

Same for qubit 1:
$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_2, a_1 + a_3, a_0 - a_2, a_1 - a_3)$.

Qubit 0 and then qubit 1:
$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_1, a_0 - a_1, a_2 + a_3, a_2 - a_3) \mapsto$
$(a_0 + a_1 + a_2 + a_3, a_0 - a_1 + a_2 - a_3,$
$a_0 + a_1 - a_2 - a_3, a_0 - a_1 - a_2 + a_3)$.

Repeat n times: e.g.,
$(1, 0, 0, \ldots, 0) \mapsto (1, 1, 1, \ldots, 1)$.

Measuring $(1, 0, 0, \ldots, 0)$
always produces 0.

Measuring $(1, 1, 1, \ldots, 1)$
can produce any output:
$\Pr[\text{output} = q] = 1/2^n$.

Aside from "normalization"
(irrelevant to measurement),
have Hadamard = Hadamard^{-1},
so easily work backwards
from "uniform superposition"
$(1, 1, 1, \ldots, 1)$ to "pure state"
$(1, 0, 0, \ldots, 0)$.

antum operations, part 2

ard":

$\mapsto (a_0 + a_1, a_0 - a_1)$.

$a_2, a_3) \mapsto$

$, a_0 - a_1, a_2 + a_3, a_2 - a_3)$.

r qubit 1:

$a_2, a_3) \mapsto$

$, a_1 + a_3, a_0 - a_2, a_1 - a_3)$.

and then qubit 1:

$a_2, a_3) \mapsto$

$a_0 - a_1, a_2 + a_3, a_2 - a_3) \mapsto$

$+ a_2 + a_3, a_0 - a_1 + a_2 - a_3,$

$- a_2 - a_3, a_0 - a_1 - a_2 + a_3)$.

Repeat n times: e.g.,
$(1, 0, 0, \ldots, 0) \mapsto (1, 1, 1, \ldots, 1)$.

Measuring $(1, 0, 0, \ldots, 0)$
always produces 0.

Measuring $(1, 1, 1, \ldots, 1)$
can produce any output:
$\Pr[\text{output} = q] = 1/2^n$.

Aside from "normalization"
(irrelevant to measurement),
have Hadamard = Hadamard^{-1},
so easily work backwards
from "uniform superposition"
$(1, 1, 1, \ldots, 1)$ to "pure state"
$(1, 0, 0, \ldots, 0)$.

Simon's

Assume:
satisfies
for every
Can we
given a

erations, part 2

$_1, a_0 - a_1)$.

$a_2 + a_3, a_2 - a_3)$.

$a_0 - a_2, a_1 - a_3)$.

qubit 1:

$_2 + a_3, a_2 - a_3) \mapsto$

$a_0 - a_1 + a_2 - a_3,$

$a_0 - a_1 - a_2 + a_3)$.

Repeat n times: e.g.,
$(1, 0, 0, \ldots, 0) \mapsto (1, 1, 1, \ldots, 1)$.

Measuring $(1, 0, 0, \ldots, 0)$
always produces 0.

Measuring $(1, 1, 1, \ldots, 1)$
can produce any output:
$\Pr[\text{output} = q] = 1/2^n$.

Aside from "normalization"
(irrelevant to measurement),
have Hadamard = Hadamard^{-1},
so easily work backwards
from "uniform superposition"
$(1, 1, 1, \ldots, 1)$ to "pure state"
$(1, 0, 0, \ldots, 0)$.

Simon's algorithm

Assume: nonzero
satisfies $f(x) = f($
for every $x \in \{0, 1$
Can we find this p
given a fast circuit

art 2

).

$- a_3$).

$- a_3$).

$a_3) \mapsto$

$a_2 - a_3$,

$a_2 + a_3$).

Repeat n times: e.g., $(1, 0, 0, \ldots, 0) \mapsto (1, 1, 1, \ldots, 1)$.

Measuring $(1, 0, 0, \ldots, 0)$ always produces 0.

Measuring $(1, 1, 1, \ldots, 1)$ can produce any output: $\Pr[\text{output} = q] = 1/2^n$.

Aside from "normalization" (irrelevant to measurement), have Hadamard = Hadamard^{-1}, so easily work backwards from "uniform superposition" $(1, 1, 1, \ldots, 1)$ to "pure state" $(1, 0, 0, \ldots, 0)$.

Simon's algorithm

Assume: nonzero $s \in \{0, 1\}$ satisfies $f(x) = f(x \oplus s)$ for every $x \in \{0, 1\}^n$. Can we find this period s, given a fast circuit for f?

Repeat n times: e.g.,
$(1, 0, 0, \ldots, 0) \mapsto (1, 1, 1, \ldots, 1)$.

Measuring $(1, 0, 0, \ldots, 0)$
always produces 0.

Measuring $(1, 1, 1, \ldots, 1)$
can produce any output:
$\Pr[\text{output} = q] = 1/2^n$.

Aside from "normalization"
(irrelevant to measurement),
have Hadamard = Hadamard^{-1},
so easily work backwards
from "uniform superposition"
$(1, 1, 1, \ldots, 1)$ to "pure state"
$(1, 0, 0, \ldots, 0)$.

Simon's algorithm

Assume: nonzero $s \in \{0, 1\}^n$
satisfies $f(x) = f(x \oplus s)$
for every $x \in \{0, 1\}^n$.
Can we find this period s,
given a fast circuit for f?

Repeat n times: e.g.,
$(1, 0, 0, \ldots, 0) \mapsto (1, 1, 1, \ldots, 1)$.

Measuring $(1, 0, 0, \ldots, 0)$
always produces 0.

Measuring $(1, 1, 1, \ldots, 1)$
can produce any output:
$\Pr[\text{output} = q] = 1/2^n$.

Aside from "normalization"
(irrelevant to measurement),
have Hadamard $=$ Hadamard^{-1},
so easily work backwards
from "uniform superposition"
$(1, 1, 1, \ldots, 1)$ to "pure state"
$(1, 0, 0, \ldots, 0)$.

Simon's algorithm

Assume: nonzero $s \in \{0, 1\}^n$
satisfies $f(x) = f(x \oplus s)$
for every $x \in \{0, 1\}^n$.
Can we find this period s,
given a fast circuit for f?

We don't have enough data
if f has many periods.
Assume: $\{\text{periods}\} = \{0, s\}$.

Repeat n times: e.g.,
$(1, 0, 0, \ldots, 0) \mapsto (1, 1, 1, \ldots, 1)$.

Measuring $(1, 0, 0, \ldots, 0)$
always produces 0.

Measuring $(1, 1, 1, \ldots, 1)$
can produce any output:
$\Pr[\text{output} = q] = 1/2^n$.

Aside from "normalization"
(irrelevant to measurement),
have Hadamard = Hadamard^{-1},
so easily work backwards
from "uniform superposition"
$(1, 1, 1, \ldots, 1)$ to "pure state"
$(1, 0, 0, \ldots, 0)$.

Simon's algorithm

Assume: nonzero $s \in \{0, 1\}^n$
satisfies $f(x) = f(x \oplus s)$
for every $x \in \{0, 1\}^n$.
Can we find this period s,
given a fast circuit for f?

We don't have enough data
if f has many periods.
Assume: $\{\text{periods}\} = \{0, s\}$.

Traditional solution:
Compute f for many inputs,
sort, analyze collisions.
Success probability is very low
until #inputs approaches $2^{n/2}$.

n times: e.g.,

$\ldots, 0) \mapsto (1, 1, 1, \ldots, 1)$.

ng $(1, 0, 0, \ldots, 0)$

produces 0.

ng $(1, 1, 1, \ldots, 1)$

duce any output:

ut $= q] = 1/2^n$.

om "normalization"

nt to measurement),

damard = Hadamard^{-1},

work backwards

niform superposition"

$\ldots, 1)$ to "pure state"

$\ldots, 0)$.

Simon's algorithm

Assume: nonzero $s \in \{0,1\}^n$ satisfies $f(x) = f(x \oplus s)$ for every $x \in \{0,1\}^n$. Can we find this period s, given a fast circuit for f?

We don't have enough data if f has many periods. Assume: $\{\text{periods}\} = \{0, s\}$.

Traditional solution: Compute f for many inputs, sort, analyze collisions. Success probability is very low until #inputs approaches $2^{n/2}$.

Simon's

far fewer

if n is la

reversibi

.g.,
$(1, 1, 1, \ldots, 1)$.
$, \ldots, 0)$
.
$, \ldots, 1)$
utput:
$1/2^n$.
alization"
surement),
Hadamard^{-1},
kwards
perposition"
"pure state"

Simon's algorithm

Assume: nonzero $s \in \{0, 1\}^n$
satisfies $f(x) = f(x \oplus s)$
for every $x \in \{0, 1\}^n$.
Can we find this period s,
given a fast circuit for f?

We don't have enough data
if f has many periods.
Assume: $\{\text{periods}\} = \{0, s\}$.

Traditional solution:
Compute f for many inputs,
sort, analyze collisions.
Success probability is very low
until #inputs approaches $2^{n/2}$.

Simon's algorithm
far fewer qubit op
if n is large and
reversibility overhe

$, 1)$.

,

d^{-1},

”

e”

Simon’s algorithm

Assume: nonzero $s \in \{0, 1\}^n$
satisfies $f(x) = f(x \oplus s)$
for every $x \in \{0, 1\}^n$.
Can we find this period s,
given a fast circuit for f?

We don’t have enough data
if f has many periods.
Assume: $\{\text{periods}\} = \{0, s\}$.

Traditional solution:
Compute f for many inputs,
sort, analyze collisions.
Success probability is very low
until #inputs approaches $2^{n/2}$.

Simon’s algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

Simon's algorithm

Assume: nonzero $s \in \{0,1\}^n$ satisfies $f(x) = f(x \oplus s)$ for every $x \in \{0,1\}^n$. Can we find this period s, given a fast circuit for f?

We don't have enough data if f has many periods. Assume: $\{\text{periods}\} = \{0, s\}$.

Traditional solution: Compute f for many inputs, sort, analyze collisions. Success probability is very low until #inputs approaches $2^{n/2}$.

Simon's algorithm uses far fewer qubit operations if n is large and reversibility overhead is low.

Simon's algorithm

Assume: nonzero $s \in \{0,1\}^n$ satisfies $f(x) = f(x \oplus s)$ for every $x \in \{0,1\}^n$.
Can we find this period s, given a fast circuit for f?

We don't have enough data if f has many periods.
Assume: $\{\text{periods}\} = \{0, s\}$.

Traditional solution:
Compute f for many inputs, sort, analyze collisions.
Success probability is very low until #inputs approaches $2^{n/2}$.

Simon's algorithm uses far fewer qubit operations if n is large and reversibility overhead is low.

Say f maps n bits to m bits using z "ancilla" bits for reversibility.

Prepare $n + m + z$ qubits in pure zero state:
vector $(1, 0, 0, \ldots)$.

Simon's algorithm

Assume: nonzero $s \in \{0,1\}^n$ satisfies $f(x) = f(x \oplus s)$ for every $x \in \{0,1\}^n$. Can we find this period s, given a fast circuit for f?

We don't have enough data if f has many periods. Assume: $\{\text{periods}\} = \{0, s\}$.

Traditional solution: Compute f for many inputs, sort, analyze collisions. Success probability is very low until #inputs approaches $2^{n/2}$.

Simon's algorithm uses far fewer qubit operations if n is large and reversibility overhead is low.

Say f maps n bits to m bits using z "ancilla" bits for reversibility.

Prepare $n + m + z$ qubits in pure zero state: vector $(1, 0, 0, \ldots)$.

Use n-fold Hadamard to move first n qubits into uniform superposition: $(1, 1, 1, \ldots, 1, 0, 0, \ldots)$ with 2^n entries 1, others 0.

algorithm

nonzero $s \in \{0,1\}^n$
$f(x) = f(x \oplus s)$
$x \in \{0,1\}^n$.
find this period s,
fast circuit for f?

t have enough data
many periods.
$\{\text{periods}\} = \{0, s\}$.

nal solution:
e f for many inputs,
alyze collisions.
probability is very low
nputs approaches $2^{n/2}$.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

Say f maps n bits to m bits using
z "ancilla" bits for reversibility.

Prepare $n + m + z$ qubits
in pure zero state:
vector $(1, 0, 0, \ldots)$.

Use n-fold Hadamard
to move first n qubits
into uniform superposition:
$(1, 1, 1, \ldots, 1, 0, 0, \ldots)$
with 2^n entries 1, others 0.

Apply fa
for rever
1 in pos
moves to

Note syr
1 at $(q,$
1 at $(q \oplus$

$s \in \{0,1\}^n$
$x \oplus s)$
$\}^n$.
period s,
t for f?

ough data
ods.
$\} = \{0, s\}$.

n:
any inputs,
ions.
y is very low
roaches $2^{n/2}$.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

Say f maps n bits to m bits using
z "ancilla" bits for reversibility.

Prepare $n + m + z$ qubits
in pure zero state:
vector $(1, 0, 0, \ldots)$.

Use n-fold Hadamard
to move first n qubits
into uniform superposition:
$(1, 1, 1, \ldots, 1, 0, 0, \ldots)$
with 2^n entries 1, others 0.

Apply fast vector p
for reversible f co
1 in position $(q, 0,$
moves to position

Note symmetry be
1 at $(q, f(q), 0)$ ar
1 at $(q \oplus s, f(q), 0$

n

.

ow

/2.

Simon's algorithm uses far fewer qubit operations if n is large and reversibility overhead is low.

Say f maps n bits to m bits using z "ancilla" bits for reversibility.

Prepare $n + m + z$ qubits in pure zero state: vector $(1, 0, 0, \ldots)$.

Use n-fold Hadamard to move first n qubits into uniform superposition: $(1, 1, 1, \ldots, 1, 0, 0, \ldots)$ with 2^n entries 1, others 0.

Apply fast vector permutatio for reversible f computation 1 in position $(q, 0, 0)$ moves to position $(q, f(q), 0$

Note symmetry between 1 at $(q, f(q), 0)$ and 1 at $(q \oplus s, f(q), 0)$.

Simon's algorithm uses far fewer qubit operations if n is large and reversibility overhead is low.

Say f maps n bits to m bits using z "ancilla" bits for reversibility.

Prepare $n + m + z$ qubits in pure zero state: vector $(1, 0, 0, \ldots)$.

Use n-fold Hadamard to move first n qubits into uniform superposition: $(1, 1, 1, \ldots, 1, 0, 0, \ldots)$ with 2^n entries 1, others 0.

Apply fast vector permutation for reversible f computation: 1 in position $(q, 0, 0)$ moves to position $(q, f(q), 0)$.

Note symmetry between 1 at $(q, f(q), 0)$ and 1 at $(q \oplus s, f(q), 0)$.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

Say f maps n bits to m bits using
z "ancilla" bits for reversibility.

Prepare $n + m + z$ qubits
in pure zero state:
vector $(1, 0, 0, \ldots)$.

Use n-fold Hadamard
to move first n qubits
into uniform superposition:
$(1, 1, 1, \ldots, 1, 0, 0, \ldots)$
with 2^n entries 1, others 0.

Apply fast vector permutation
for reversible f computation:
1 in position $(q, 0, 0)$
moves to position $(q, f(q), 0)$.

Note symmetry between
1 at $(q, f(q), 0)$ and
1 at $(q \oplus s, f(q), 0)$.

Apply n-fold Hadamard.

Simon's algorithm uses far fewer qubit operations if n is large and reversibility overhead is low.

Say f maps n bits to m bits using z "ancilla" bits for reversibility.

Prepare $n+m+z$ qubits in pure zero state: vector $(1, 0, 0, \ldots)$.

Use n-fold Hadamard to move first n qubits into uniform superposition: $(1, 1, 1, \ldots, 1, 0, 0, \ldots)$ with 2^n entries 1, others 0.

Apply fast vector permutation for reversible f computation: 1 in position $(q, 0, 0)$ moves to position $(q, f(q), 0)$.

Note symmetry between 1 at $(q, f(q), 0)$ and 1 at $(q \oplus s, f(q), 0)$.

Apply n-fold Hadamard.

Measure. By symmetry, output is orthogonal to s.

Simon's algorithm uses far fewer qubit operations if n is large and reversibility overhead is low.

Say f maps n bits to m bits using z "ancilla" bits for reversibility.

Prepare $n + m + z$ qubits in pure zero state: vector $(1, 0, 0, \ldots)$.

Use n-fold Hadamard to move first n qubits into uniform superposition: $(1, 1, 1, \ldots, 1, 0, 0, \ldots)$ with 2^n entries 1, others 0.

Apply fast vector permutation for reversible f computation: 1 in position $(q, 0, 0)$ moves to position $(q, f(q), 0)$.

Note symmetry between 1 at $(q, f(q), 0)$ and 1 at $(q \oplus s, f(q), 0)$.

Apply n-fold Hadamard.

Measure. By symmetry, output is orthogonal to s.

Repeat $n + 10$ times. Use Gaussian elimination to (probably) find s.

algorithm uses
r qubit operations
rge and
lity overhead is low.

aps n bits to m bits using
a" bits for reversibility.

$n + m + z$ qubits
zero state:
$1, 0, 0, \ldots)$.

old Hadamard
first n qubits
form superposition:
$\ldots, 1, 0, 0, \ldots)$
entries 1, others 0.

Apply fast vector permutation
for reversible f computation:
1 in position $(q, 0, 0)$
moves to position $(q, f(q), 0)$.

Note symmetry between
1 at $(q, f(q), 0)$ and
1 at $(q \oplus s, f(q), 0)$.

Apply n-fold Hadamard.

Measure. By symmetry,
output is orthogonal to s.

Repeat $n + 10$ times.
Use Gaussian elimination
to (probably) find s.

Example

$f(0) = 4$
$f(1) = 7$
$f(2) = 2$
$f(3) = 3$
$f(4) = 7$
$f(5) = 4$
$f(6) = 3$
$f(7) = 2$

uses
erations

ead is low.

to m bits using
r reversibility.

z qubits

.

ard
bits
rposition:
$\ldots)$
others 0.

Apply fast vector permutation
for reversible f computation:
1 in position $(q, 0, 0)$
moves to position $(q, f(q), 0)$.

Note symmetry between
1 at $(q, f(q), 0)$ and
1 at $(q \oplus s, f(q), 0)$.

Apply n-fold Hadamard.

Measure. By symmetry,
output is orthogonal to s.

Repeat $n + 10$ times.
Use Gaussian elimination
to (probably) find s.

Example, 3 bits to

$f(0) = 4$.
$f(1) = 7$.
$f(2) = 2$.
$f(3) = 3$.
$f(4) = 7$.
$f(5) = 4$.
$f(6) = 3$.
$f(7) = 2$.

using

ity.

Apply fast vector permutation for reversible f computation: 1 in position $(q, 0, 0)$ moves to position $(q, f(q), 0)$.

Note symmetry between 1 at $(q, f(q), 0)$ and 1 at $(q \oplus s, f(q), 0)$.

Apply n-fold Hadamard.

Measure. By symmetry, output is orthogonal to s.

Repeat $n + 10$ times. Use Gaussian elimination to (probably) find s.

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$

Apply fast vector permutation
for reversible f computation:
1 in position $(q, 0, 0)$
moves to position $(q, f(q), 0)$.

Note symmetry between
1 at $(q, f(q), 0)$ and
1 at $(q \oplus s, f(q), 0)$.

Apply n-fold Hadamard.

Measure. By symmetry,
output is orthogonal to s.

Repeat $n + 10$ times.
Use Gaussian elimination
to (probably) find s.

Example, 3 bits to 3 bits:

$f(0) = 4$.
$f(1) = 7$.
$f(2) = 2$.
$f(3) = 3$.
$f(4) = 7$.
$f(5) = 4$.
$f(6) = 3$.
$f(7) = 2$.

Apply fast vector permutation
for reversible f computation:
1 in position $(q, 0, 0)$
moves to position $(q, f(q), 0)$.

Note symmetry between
1 at $(q, f(q), 0)$ and
1 at $(q \oplus s, f(q), 0)$.

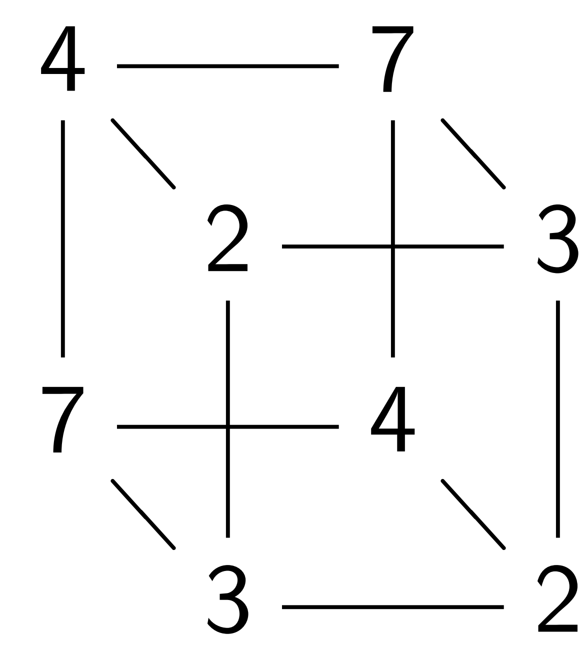
Apply n-fold Hadamard.

Measure. By symmetry,
output is orthogonal to s.

Repeat $n + 10$ times.
Use Gaussian elimination
to (probably) find s.

Example, 3 bits to 3 bits:

$f(0) = 4$.
$f(1) = 7$.
$f(2) = 2$.
$f(3) = 3$.
$f(4) = 7$.
$f(5) = 4$.
$f(6) = 3$.
$f(7) = 2$.



Apply fast vector permutation for reversible f computation: 1 in position $(q, 0, 0)$ moves to position $(q, f(q), 0)$.

Note symmetry between 1 at $(q, f(q), 0)$ and 1 at $(q \oplus s, f(q), 0)$.

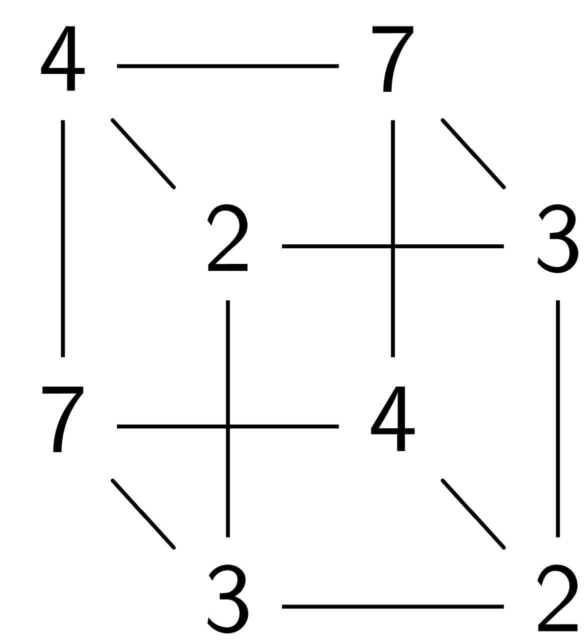
Apply n-fold Hadamard.

Measure. By symmetry, output is orthogonal to s.

Repeat $n + 10$ times.
Use Gaussian elimination to (probably) find s.

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



Complete table shows that $f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm for f, using 6 qubits.

st vector permutation
sible f computation:
ition $(q, 0, 0)$
o position $(q, f(q), 0)$.

mmetry between
$f(q), 0)$ and
$\oplus s, f(q), 0)$.

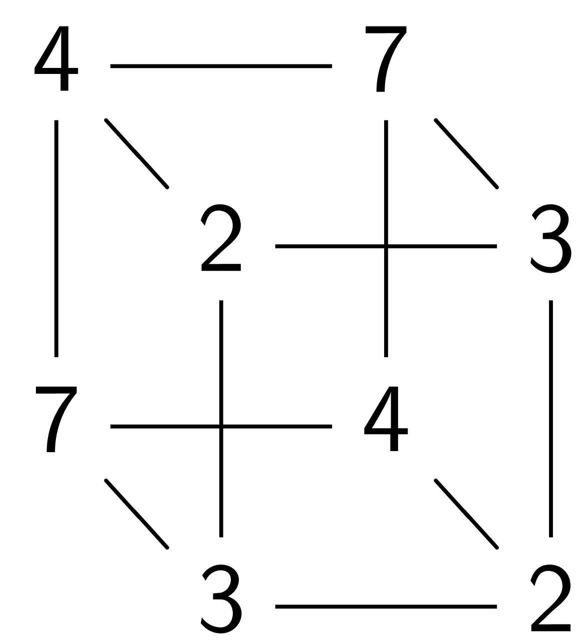
-fold Hadamard.

. By symmetry,
s orthogonal to s.

$n + 10$ times.
ssian elimination
ably) find s.

Example, 3 bits to 3 bits:

$f(0) = 4$.
$f(1) = 7$.
$f(2) = 2$.
$f(3) = 3$.
$f(4) = 7$.
$f(5) = 4$.
$f(6) = 3$.
$f(7) = 2$.



Complete table shows that $f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm for f, using 6 qubits.

Step 1.

$1, 0, 0, 0$
$0, 0, 0, 0$
$0, 0, 0, 0$
$0, 0, 0, 0$
$0, 0, 0, 0$
$0, 0, 0, 0$
$0, 0, 0, 0$
$0, 0, 0, 0$

permutation

mputation:

$, 0)$

$(q, f(q), 0)$.

etween

nd

$0))$.

mard.

metry,

nal to s.

nes.

ination

s.

Example, 3 bits to 3 bits:

$f(0) = 4$.

$f(1) = 7$.

$f(2) = 2$.

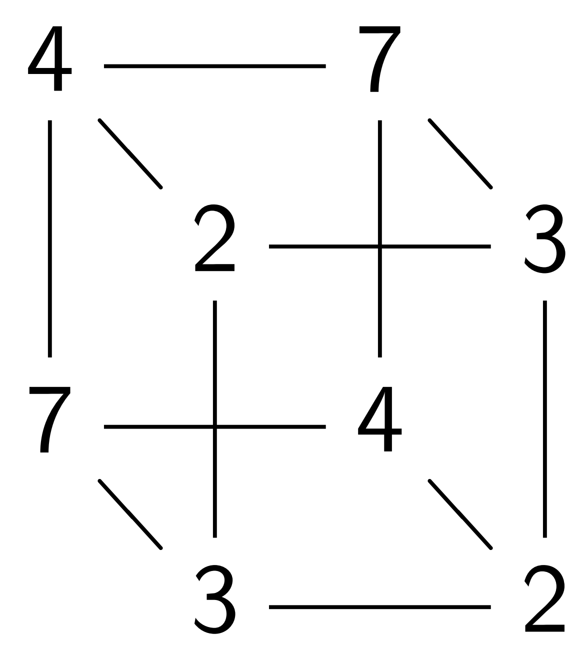
$f(3) = 3$.

$f(4) = 7$.

$f(5) = 4$.

$f(6) = 3$.

$f(7) = 2$.



Complete table shows that $f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm for f, using 6 qubits.

Step 1. Set up pu

1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

on

:

0)).

Example, 3 bits to 3 bits:

$f(0) = 4.$

$f(1) = 7.$

$f(2) = 2.$

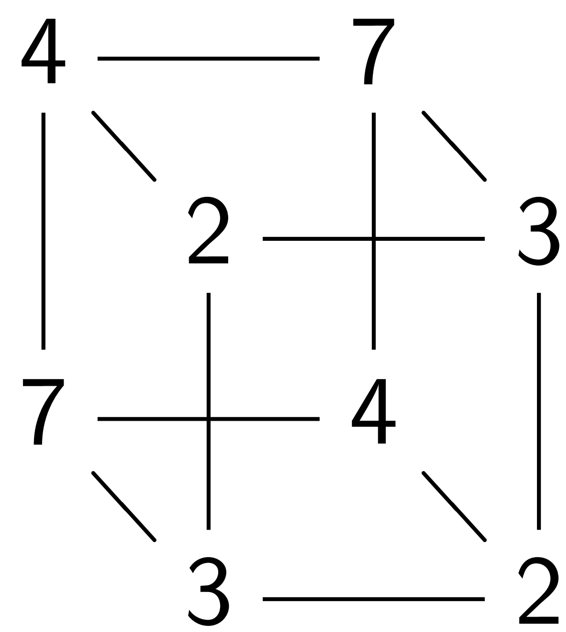
$f(3) = 3.$

$f(4) = 7.$

$f(5) = 4.$

$f(6) = 3.$

$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm
for f, using 6 qubits.

Step 1. Set up pure zero sta

1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Example, 3 bits to 3 bits:

$f(0) = 4.$

$f(1) = 7.$

$f(2) = 2.$

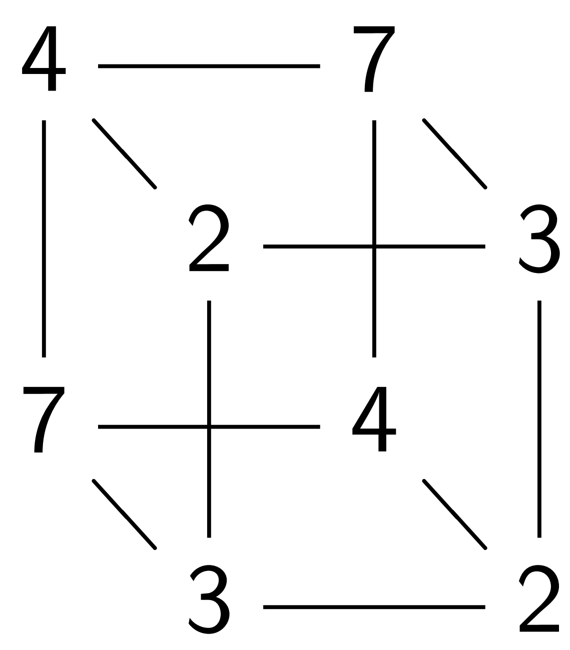
$f(3) = 3.$

$f(4) = 7.$

$f(5) = 4.$

$f(6) = 3.$

$f(7) = 2.$



Complete table shows that

$f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm

for f, using 6 qubits.

Step 1. Set up pure zero state:

1, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0.

Example, 3 bits to 3 bits:

$f(0) = 4.$

$f(1) = 7.$

$f(2) = 2.$

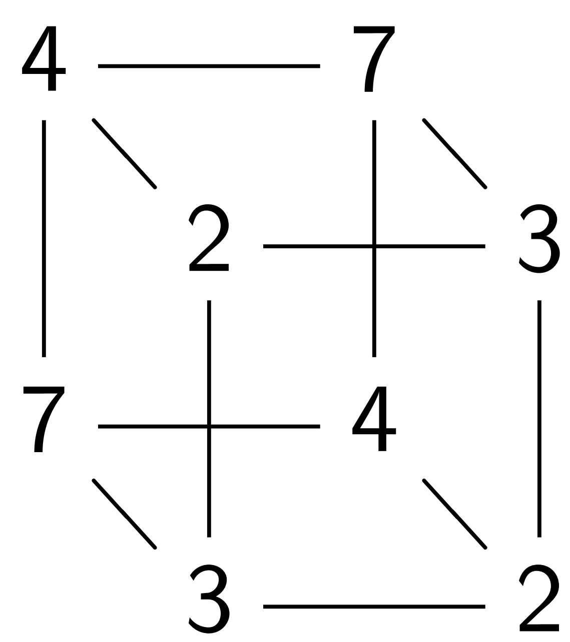
$f(3) = 3.$

$f(4) = 7.$

$f(5) = 4.$

$f(6) = 3.$

$f(7) = 2.$



Complete table shows that

$f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm

for f, using 6 qubits.

Step 2. Hadamard on qubit 0:

$\color{blue}{1}, \color{blue}{1}, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

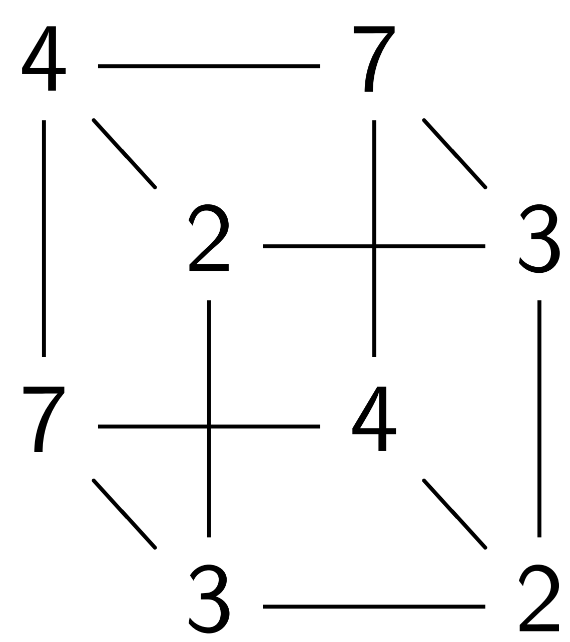
$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0.$

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

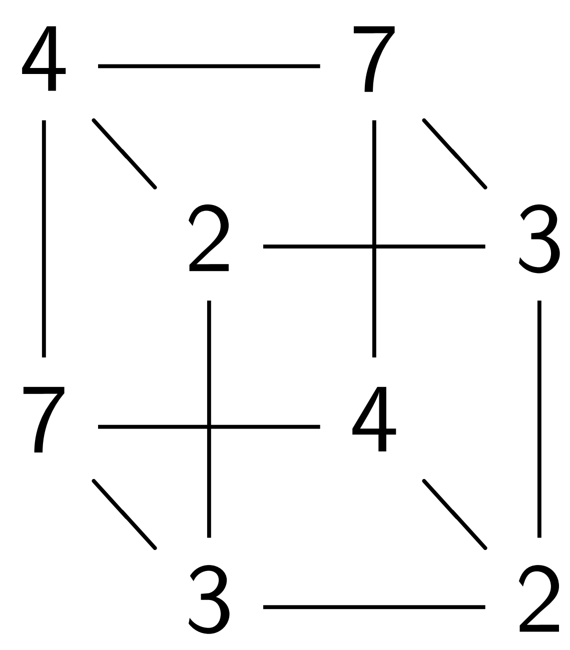
Let's watch Simon's algorithm
for f, using 6 qubits.

Step 3. Hadamard on qubit 1:

1, 1, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm
for f, using 6 qubits.

Step 4. Hadamard on qubit 2:

1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Example, 3 bits to 3 bits:

$f(0) = 4.$

$f(1) = 7.$

$f(2) = 2.$

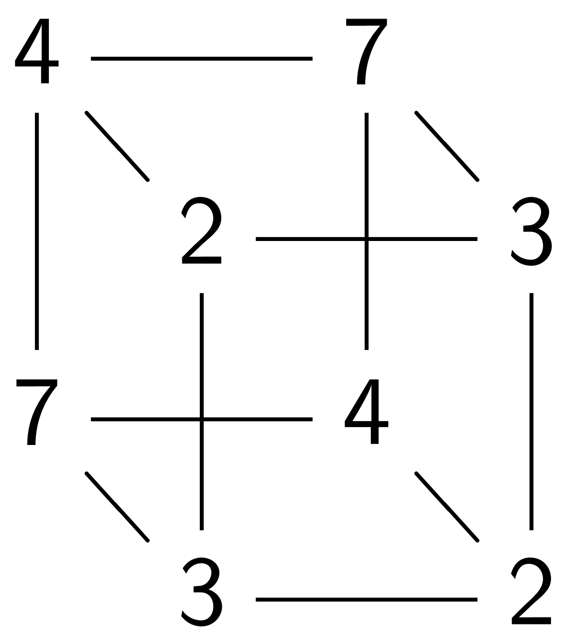
$f(3) = 3.$

$f(4) = 7.$

$f(5) = 4.$

$f(6) = 3.$

$f(7) = 2.$



Complete table shows that

$f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm

for f, using 6 qubits.

Step 5. $(q, 0) \mapsto (q, f(q))$:

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 1, 0, 0, 0, 0, 1,

0, 0, 0, 1, 0, 0, 1, 0,

1, 0, 0, 0, 0, 1, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

0, 1, 0, 0, 1, 0, 0, 0.

Example, 3 bits to 3 bits:

$f(0) = 4.$

$f(1) = 7.$

$f(2) = 2.$

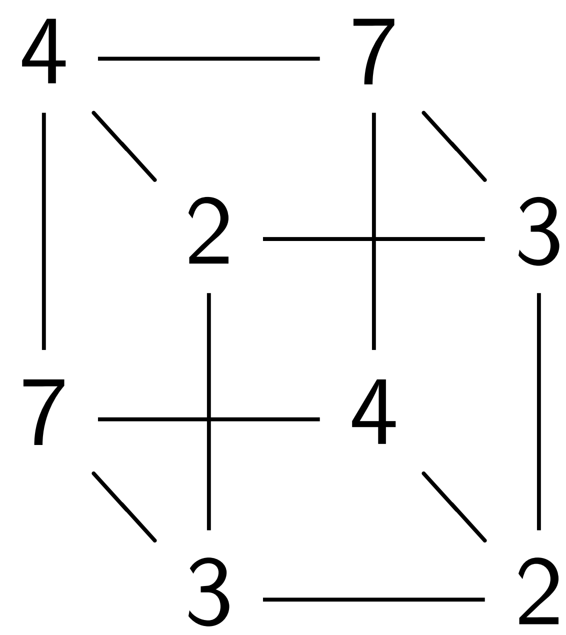
$f(3) = 3.$

$f(4) = 7.$

$f(5) = 4.$

$f(6) = 3.$

$f(7) = 2.$



Complete table shows that

$f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm

for f, using 6 qubits.

Step 6. Hadamard on qubit 0:

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 1, 1, 0, 0, 1, \overline{1},$

$0, 0, 1, \overline{1}, 0, 0, 1, 1,$

$1, 1, 0, 0, 1, \overline{1}, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$1, \overline{1}, 0, 0, 1, 1, 0, 0.$

Notation: $\overline{1} = -1.$

Example, 3 bits to 3 bits:

$f(0) = 4.$

$f(1) = 7.$

$f(2) = 2.$

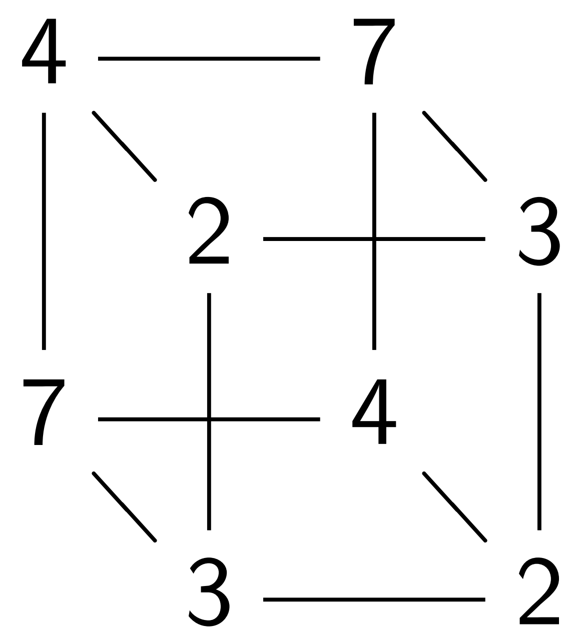
$f(3) = 3.$

$f(4) = 7.$

$f(5) = 4.$

$f(6) = 3.$

$f(7) = 2.$



Complete table shows that $f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm for f, using 6 qubits.

Step 7. Hadamard on qubit 1:

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$1, 1, \overline{1}, \overline{1}, 1, \overline{1}, \overline{1}, 1,$

$1, \overline{1}, \overline{1}, 1, 1, 1, \overline{1}, \overline{1},$

$1, 1, 1, 1, 1, \overline{1}, 1, \overline{1},$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$1, \overline{1}, 1, \overline{1}, 1, 1, 1, 1.$

Example, 3 bits to 3 bits:

$f(0) = 4.$

$f(1) = 7.$

$f(2) = 2.$

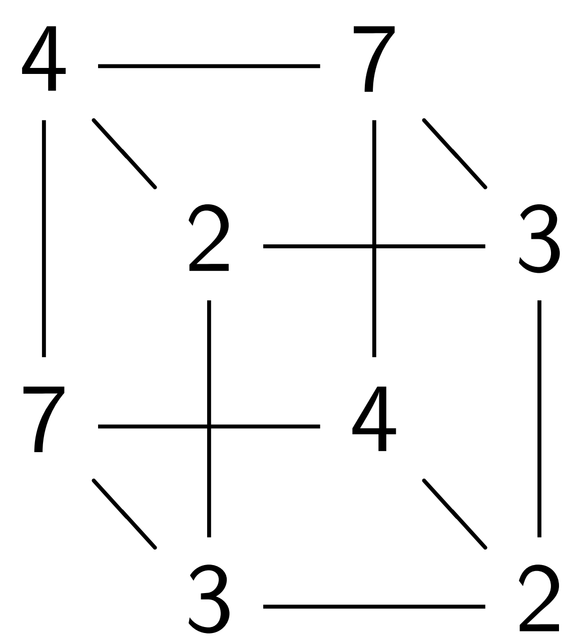
$f(3) = 3.$

$f(4) = 7.$

$f(5) = 4.$

$f(6) = 3.$

$f(7) = 2.$



Complete table shows that

$f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm

for f, using 6 qubits.

Step 8. Hadamard on qubit 2:

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, \overline{2},$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$

$2, 0, 2, 0, 0, 2, 0, 2,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2}.$

Example, 3 bits to 3 bits:

$f(0) = 4.$

$f(1) = 7.$

$f(2) = 2.$

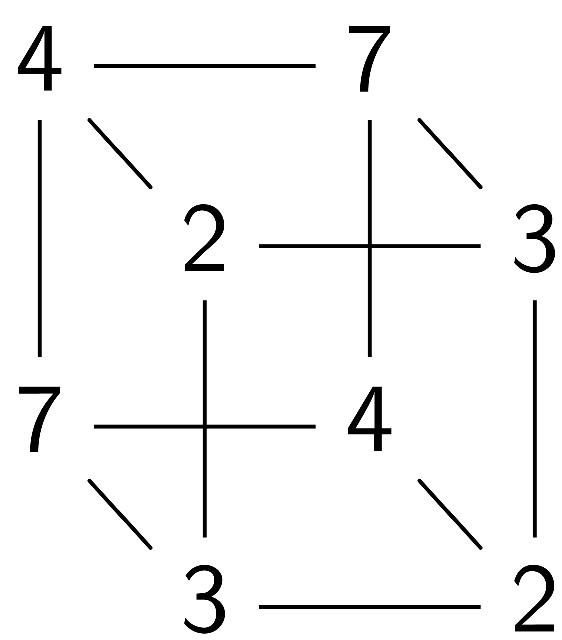
$f(3) = 3.$

$f(4) = 7.$

$f(5) = 4.$

$f(6) = 3.$

$f(7) = 2.$



Complete table shows that

$f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm

for f, using 6 qubits.

Step 8. Hadamard on qubit 2:

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, \overline{2},$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$

$2, 0, 2, 0, 0, 2, 0, 2,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2}.$

Step 9. Measure.

First 3 qubits are uniform random

vector orthogonal to 101: i.e.,

000, 010, 101, or 111.

e, 3 bits to 3 bits:

4.

7.

2.

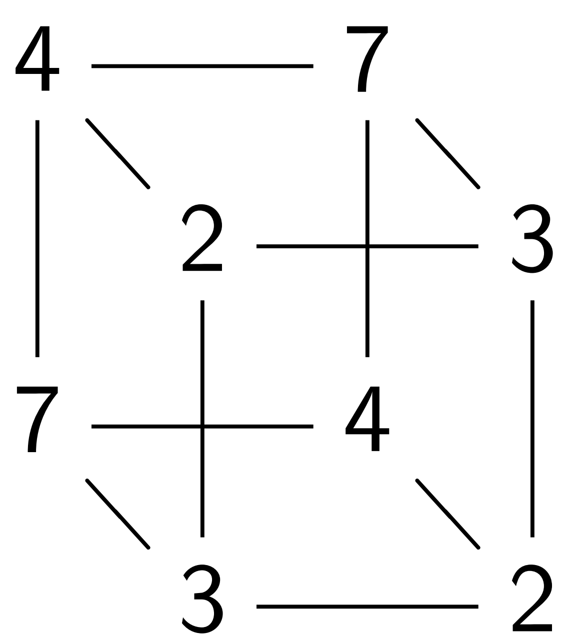
3.

7.

4.

3.

2.



te table shows that

$f(x \oplus 5)$ for all x.

tch Simon's algorithm

sing 6 qubits.

Step 8. Hadamard on qubit 2:

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, \overline{2},$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$

$2, 0, 2, 0, 0, 2, 0, 2,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2}.$

Step 9. Measure.
First 3 qubits are uniform random
vector orthogonal to 101: i.e.,
000, 010, 101, or 111.

Grover's

Assume:

has $f(s)$

Traditio

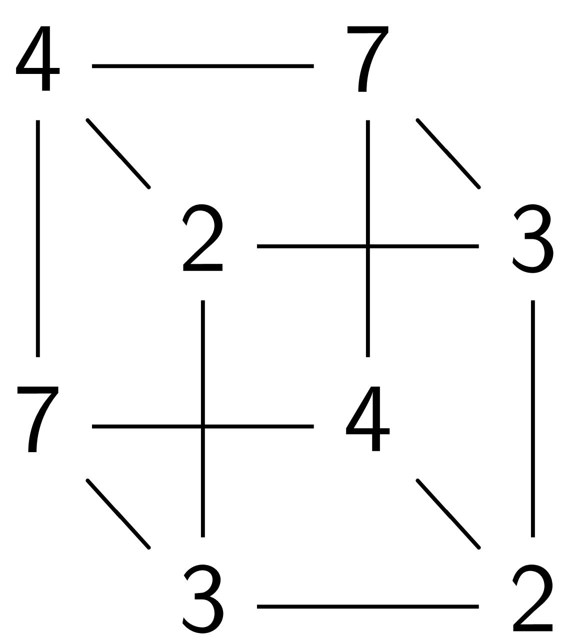
compute

hope to

Success

until #i

3 bits:



ows that

or all x.

's algorithm

its.

Step 8. Hadamard on qubit 2:

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, \overline{2},$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$

$2, 0, 2, 0, 0, 2, 0, 2,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

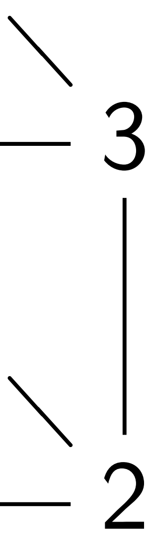
$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2}.$

Step 9. Measure.
First 3 qubits are uniform random vector orthogonal to 101: i.e., 000, 010, 101, or 111.

Grover's algorithm

Assume: unique s
has $f(s) = 0$.

Traditional algorit
compute f for ma
hope to find outpu
Success probabilit
until #inputs appr



m

Step 8. Hadamard on qubit 2:

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, \overline{2},$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$

$2, 0, 2, 0, 0, 2, 0, 2,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2}.$

Step 9. Measure.
First 3 qubits are uniform random vector orthogonal to 101: i.e., 000, 010, 101, or 111.

Grover's algorithm

Assume: unique $s \in \{0, 1\}^n$
has $f(s) = 0$.

Traditional algorithm to find
compute f for many inputs,
hope to find output 0.
Success probability is very lo
until #inputs approaches 2^n

Step 8. Hadamard on qubit 2:

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, \overline{2},$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$

$2, 0, 2, 0, 0, 2, 0, 2,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2}.$

Step 9. Measure.
First 3 qubits are uniform random vector orthogonal to 101: i.e., 000, 010, 101, or 111.

Grover's algorithm

Assume: unique $s \in \{0, 1\}^n$ has $f(s) = 0$.

Traditional algorithm to find s: compute f for many inputs, hope to find output 0.
Success probability is very low until #inputs approaches 2^n.

Step 8. Hadamard on qubit 2:

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, \overline{2},$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$

$2, 0, 2, 0, 0, 2, 0, 2,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2}.$

Step 9. Measure.
First 3 qubits are uniform random vector orthogonal to 101: i.e., 000, 010, 101, or 111.

Grover's algorithm

Assume: unique $s \in \{0, 1\}^n$ has $f(s) = 0$.

Traditional algorithm to find s: compute f for many inputs, hope to find output 0.
Success probability is very low until #inputs approaches 2^n.

Grover's algorithm takes only $2^{n/2}$ reversible computations of f.
Typically: reversibility overhead is small enough that this easily beats traditional algorithm.

Hadamard on qubit 2:

$, 0, 0, 0, 0,$

$, 0, 0, 0, 0,$

$, 0, \overline{2}, 0, \overline{2},$

$, 0, \overline{2}, 0, 2,$

$, 0, 2, 0, 2,$

$, 0, 0, 0, 0,$

$, 0, 0, 0, 0,$

$, 0, \overline{2}, 0, \overline{2}.$

Measure.

qubits are uniform random

rthogonal to 101: i.e.,

0, 101, or 111.

Grover's algorithm

Assume: unique $s \in \{0, 1\}^n$ has $f(s) = 0$.

Traditional algorithm to find s: compute f for many inputs, hope to find output 0. Success probability is very low until #inputs approaches 2^n.

Grover's algorithm takes only $2^{n/2}$ reversible computations of f. Typically: reversibility overhead is small enough that this easily beats traditional algorithm.

Start fro

over all

Step 1:

$b_q = -a$

$b_q = a_q$

This is f

Step 2:

Negate

This is a

Repeat

about 0.

Measure

With hig

d on qubit 2:

uniform random
to 101: i.e.,
111.

Grover's algorithm

Assume: unique $s \in \{0,1\}^n$
has $f(s) = 0$.

Traditional algorithm to find s:
compute f for many inputs,
hope to find output 0.
Success probability is very low
until #inputs approaches 2^n.

Grover's algorithm takes only $2^{n/2}$
reversible computations of f.
Typically: reversibility overhead
is small enough that this
easily beats traditional algorithm.

Start from uniform
over all n-bit strin

Step 1: Set $a \leftarrow b$
$b_q = -a_q$ if $f(q)$
$b_q = a_q$ otherwise
This is fast.

Step 2: "Grover d
Negate a around i
This is also fast.

Repeat Step 1 + S
about $0.58 \cdot 2^{0.5n}$

Measure the n qub
With high probabi

2:

ndom

e.,

Grover's algorithm

Assume: unique $s \in \{0,1\}^n$ has $f(s) = 0$.

Traditional algorithm to find s: compute f for many inputs, hope to find output 0.
Success probability is very low until #inputs approaches 2^n.

Grover's algorithm takes only $2^{n/2}$ reversible computations of f.
Typically: reversibility overhead is small enough that this easily beats traditional algorithm.

Start from uniform superpos over all n-bit strings q.

Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this fi

Grover's algorithm

Assume: unique $s \in \{0,1\}^n$ has $f(s) = 0$.

Traditional algorithm to find s: compute f for many inputs, hope to find output 0. Success probability is very low until #inputs approaches 2^n.

Grover's algorithm takes only $2^{n/2}$ reversible computations of f. Typically: reversibility overhead is small enough that this easily beats traditional algorithm.

Start from uniform superposition over all n-bit strings q.

Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion". Negate a around its average. This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits. With high probability this finds s.

algorithm

unique $s \in \{0, 1\}^n$
$= 0$.

nal algorithm to find s:
e f for many inputs,
find output 0.
probability is very low
nputs approaches 2^n.

algorithm takes only $2^{n/2}$
e computations of f.
y: reversibility overhead
enough that this
eats traditional algorithm.

Start from uniform superposition over all n-bit strings q.

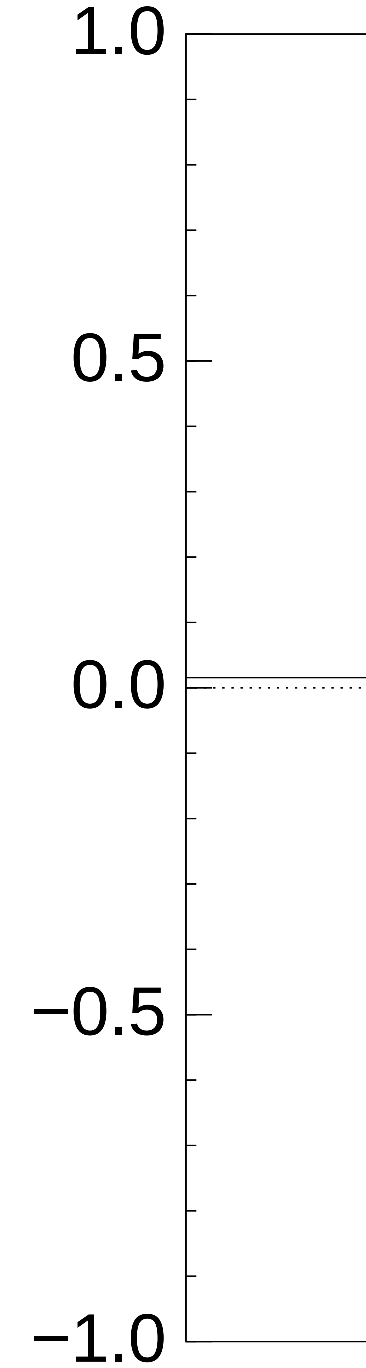
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normaliz
for an ex
after 0 s



$\in \{0,1\}^n$

hm to find s:
ny inputs,
ut 0.
y is very low
roaches 2^n.

takes only $2^{n/2}$
ations of f.
ility overhead
at this
onal algorithm.

Start from uniform superposition over all n-bit strings q.

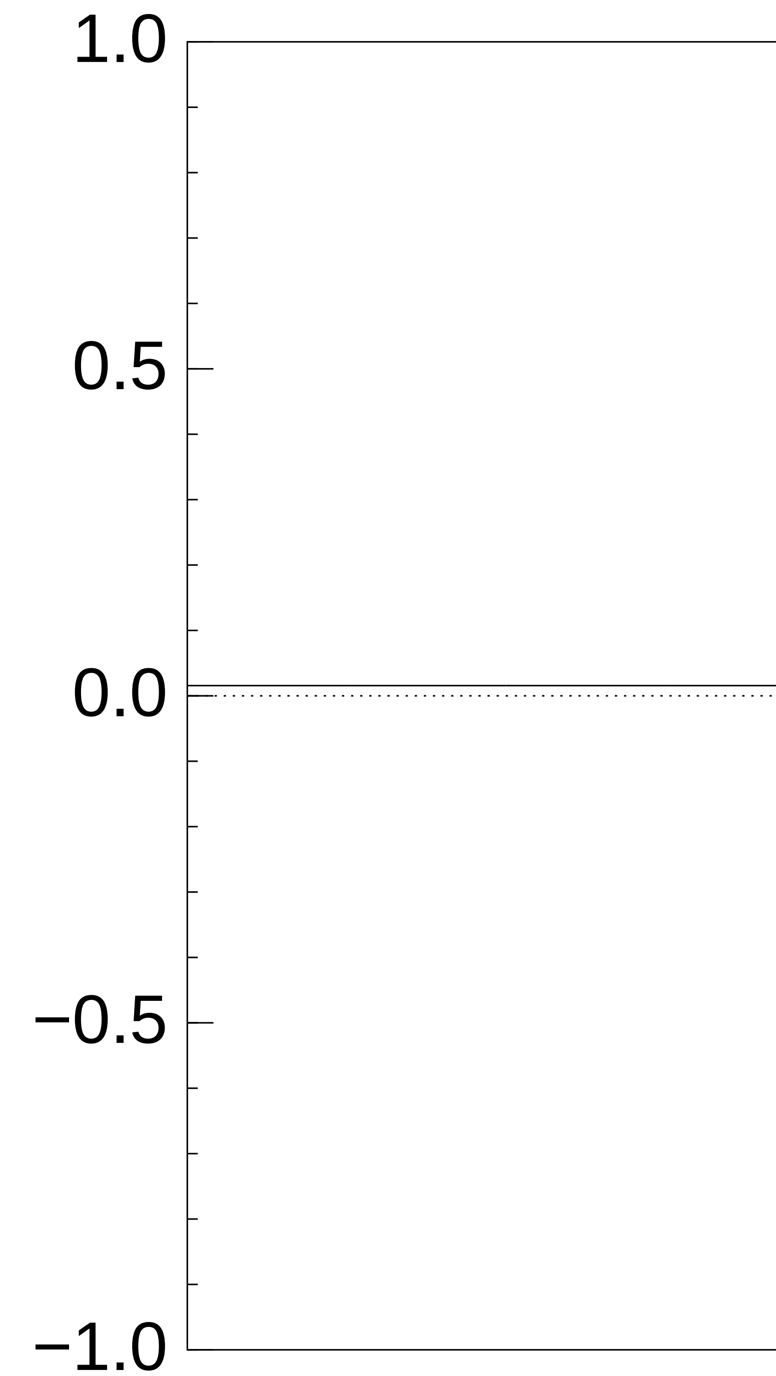
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph
for an example wit
after 0 steps:



l s:

ow

'.

y $2^{n/2}$

'.

ead

ithm.

Start from uniform superposition over all n-bit strings q.

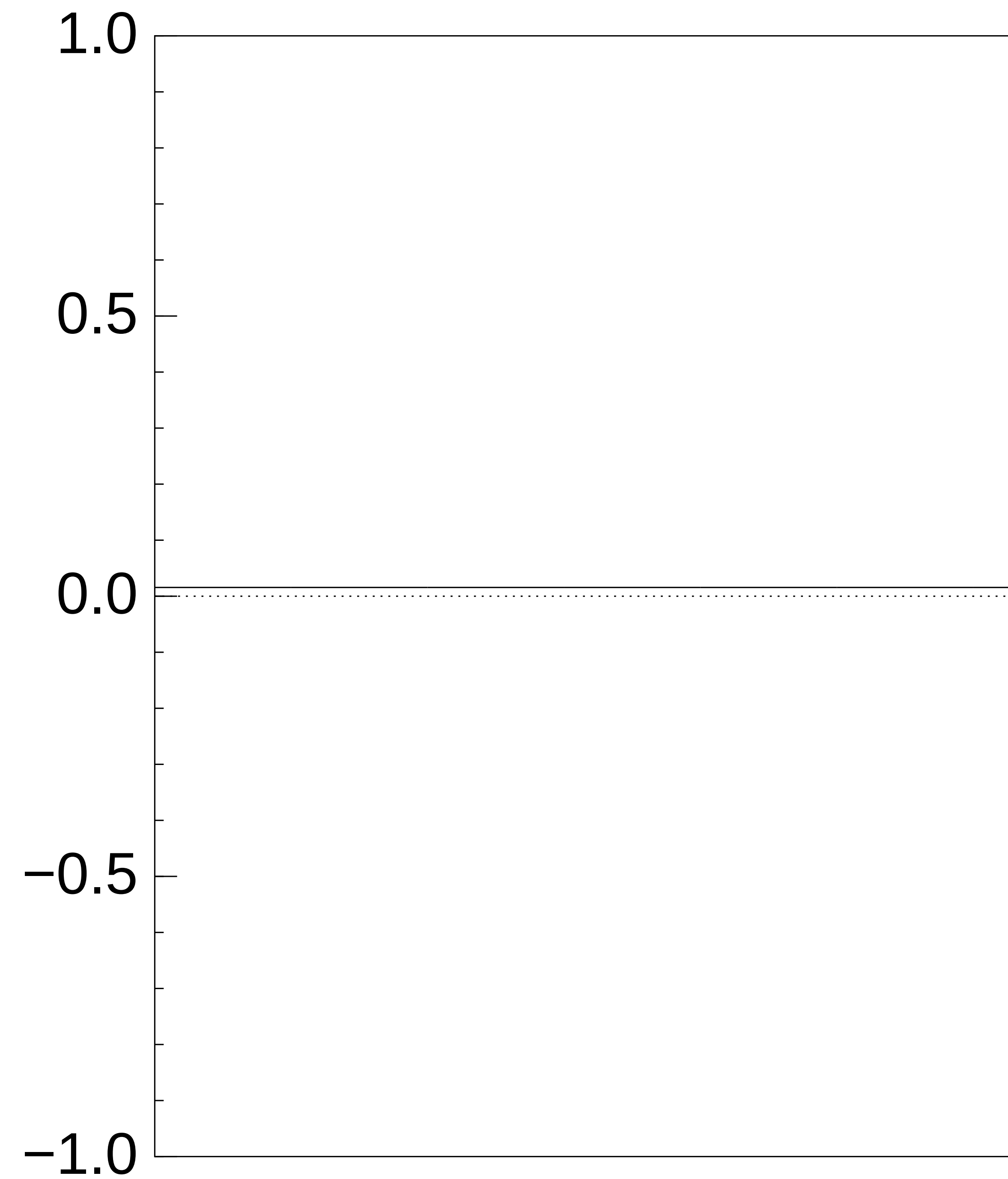
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after 0 steps:



Start from uniform superposition over all n-bit strings q.

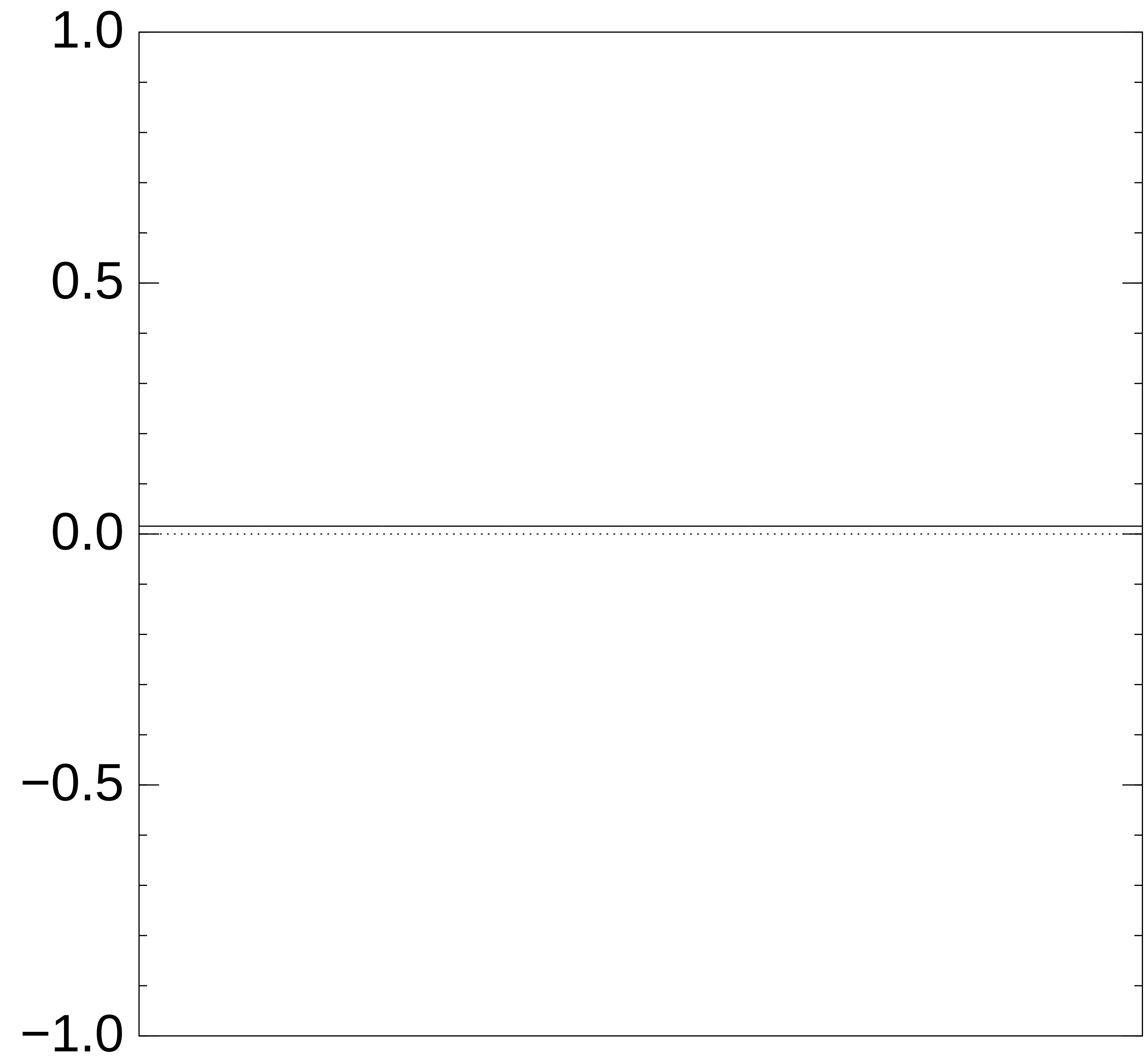
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after 0 steps:



Start from uniform superposition over all n-bit strings q.

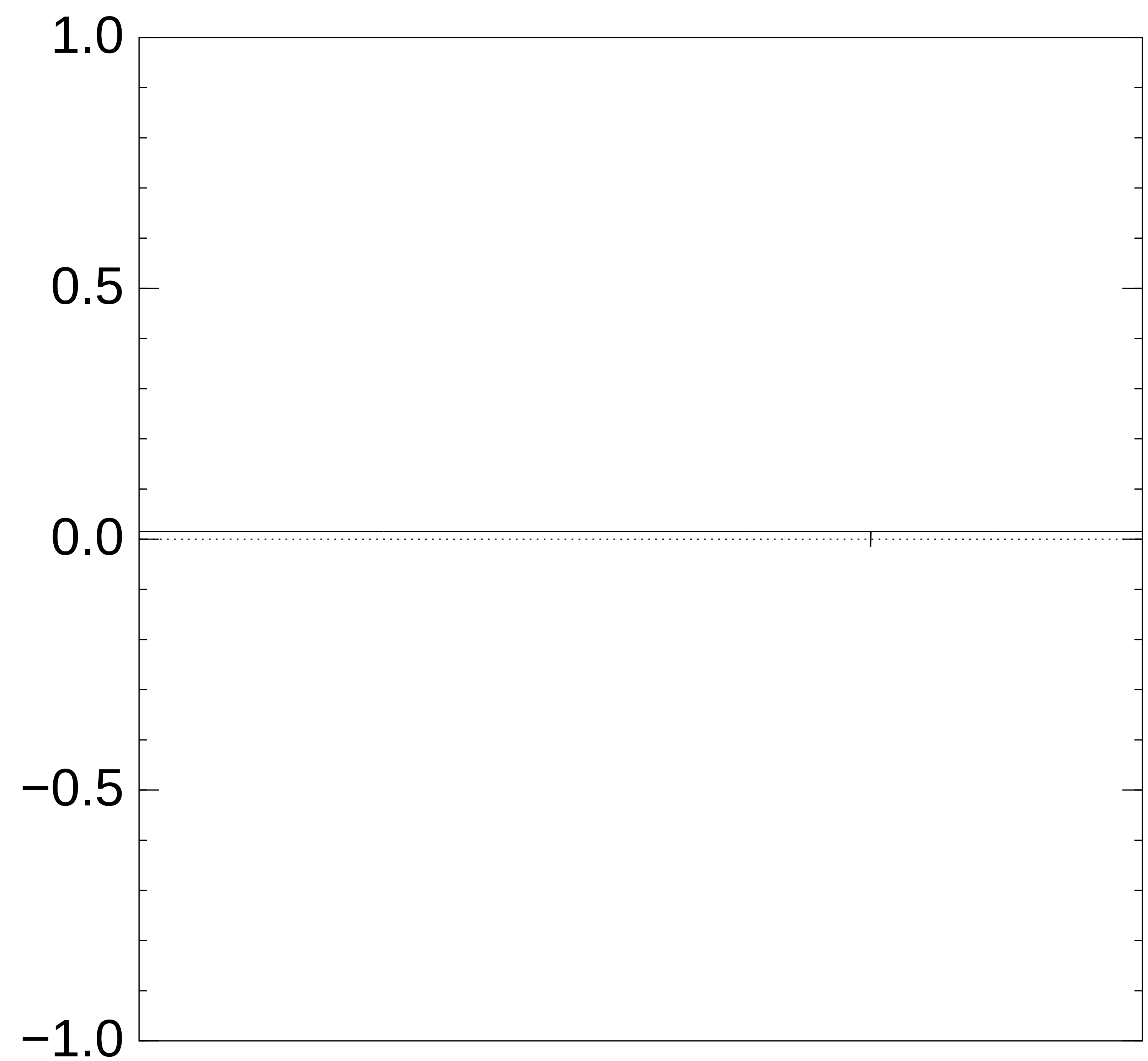
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after Step 1:



Start from uniform superposition over all n-bit strings q.

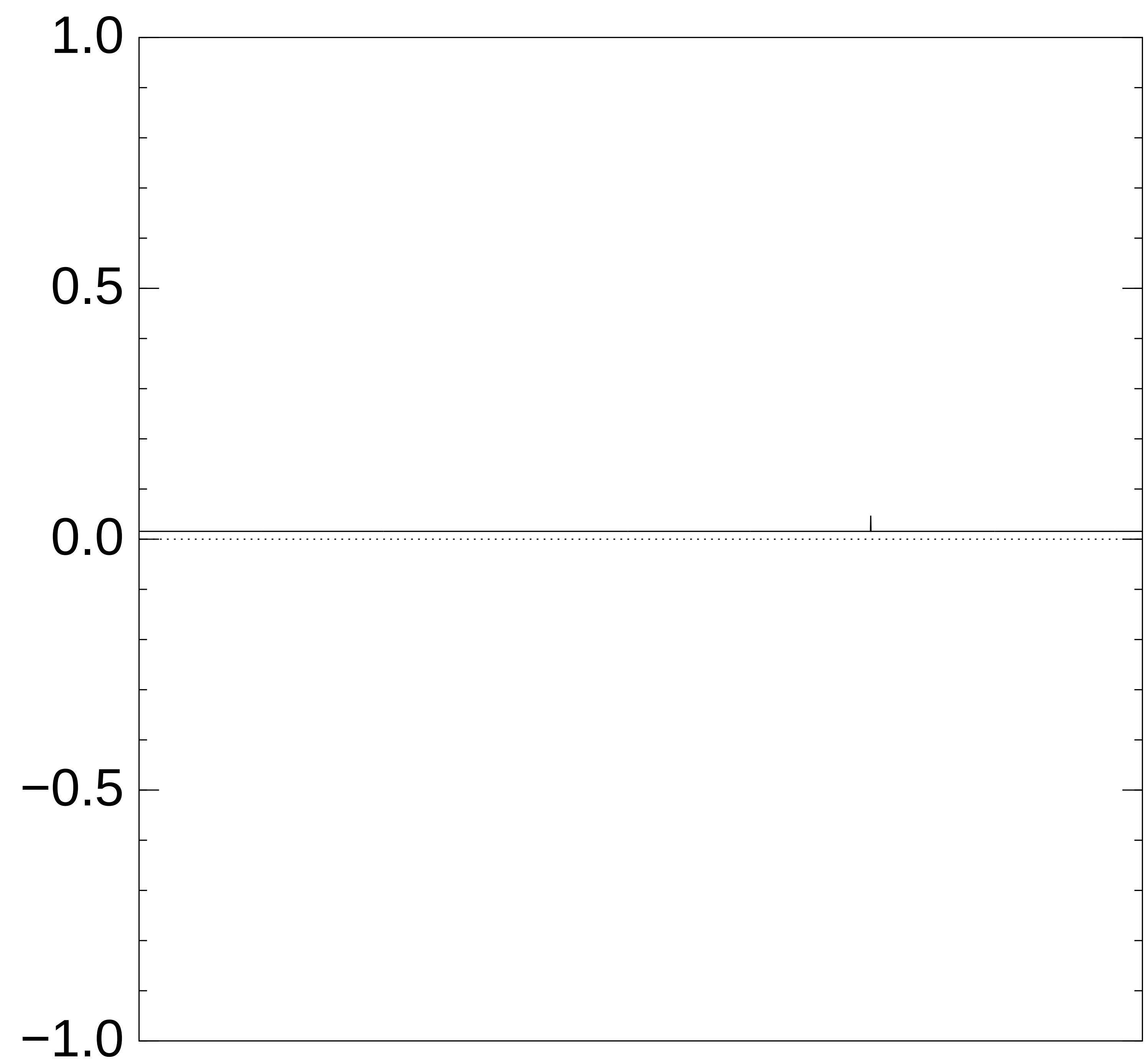
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after Step 1 + Step 2:



Start from uniform superposition over all n-bit strings q.

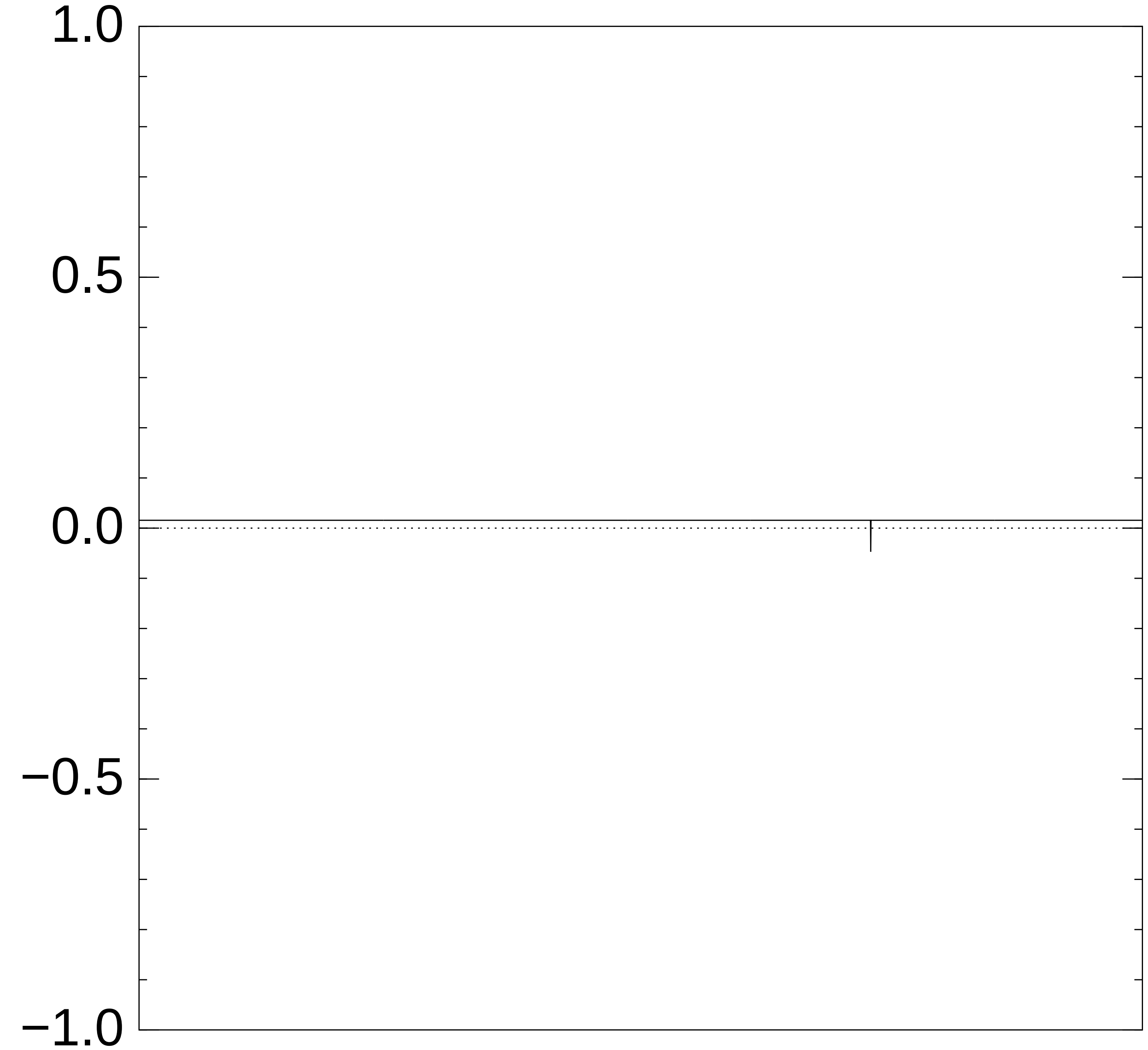
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after Step 1 + Step 2 + Step 1:



Start from uniform superposition over all n-bit strings q.

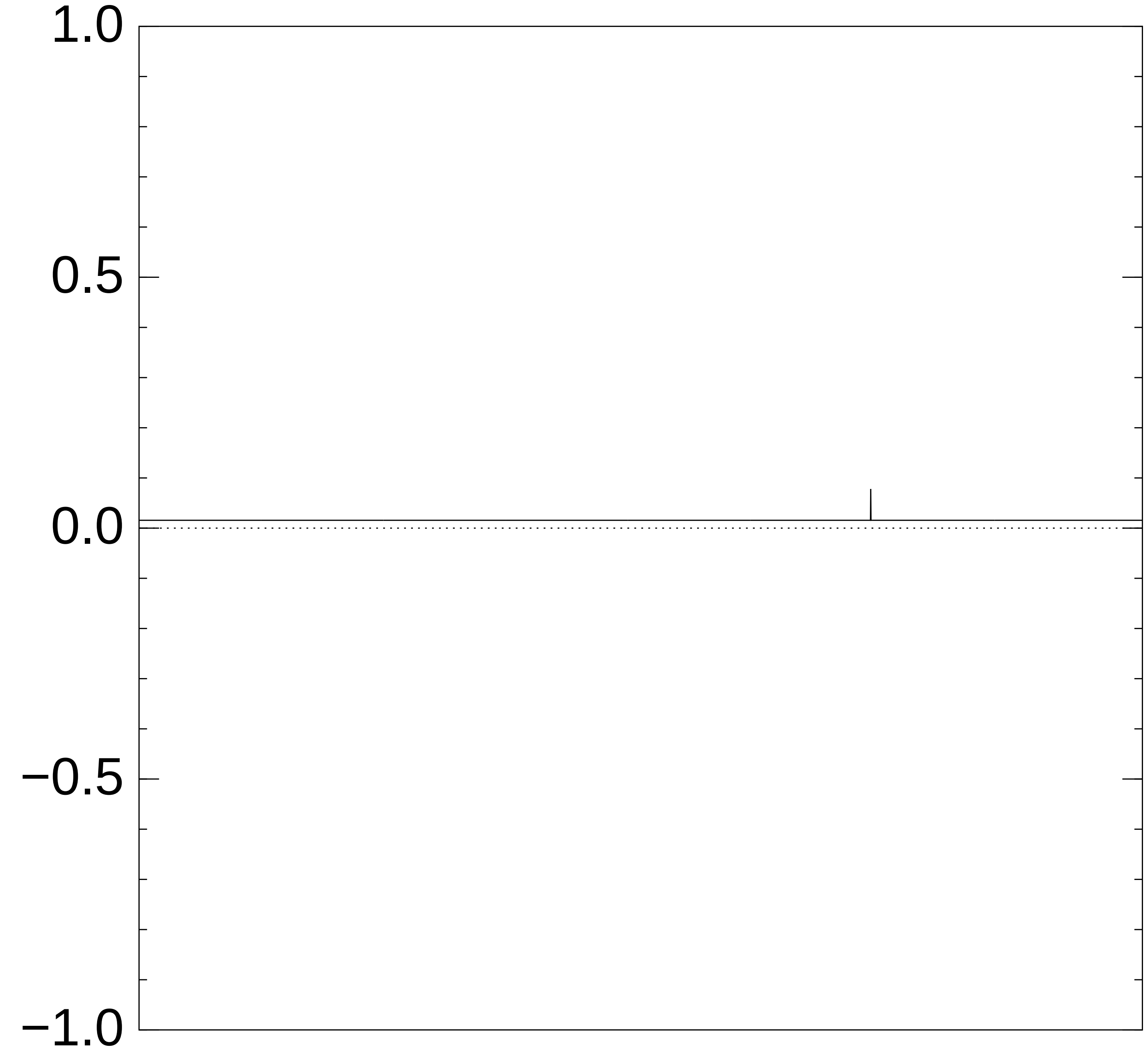
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $2 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

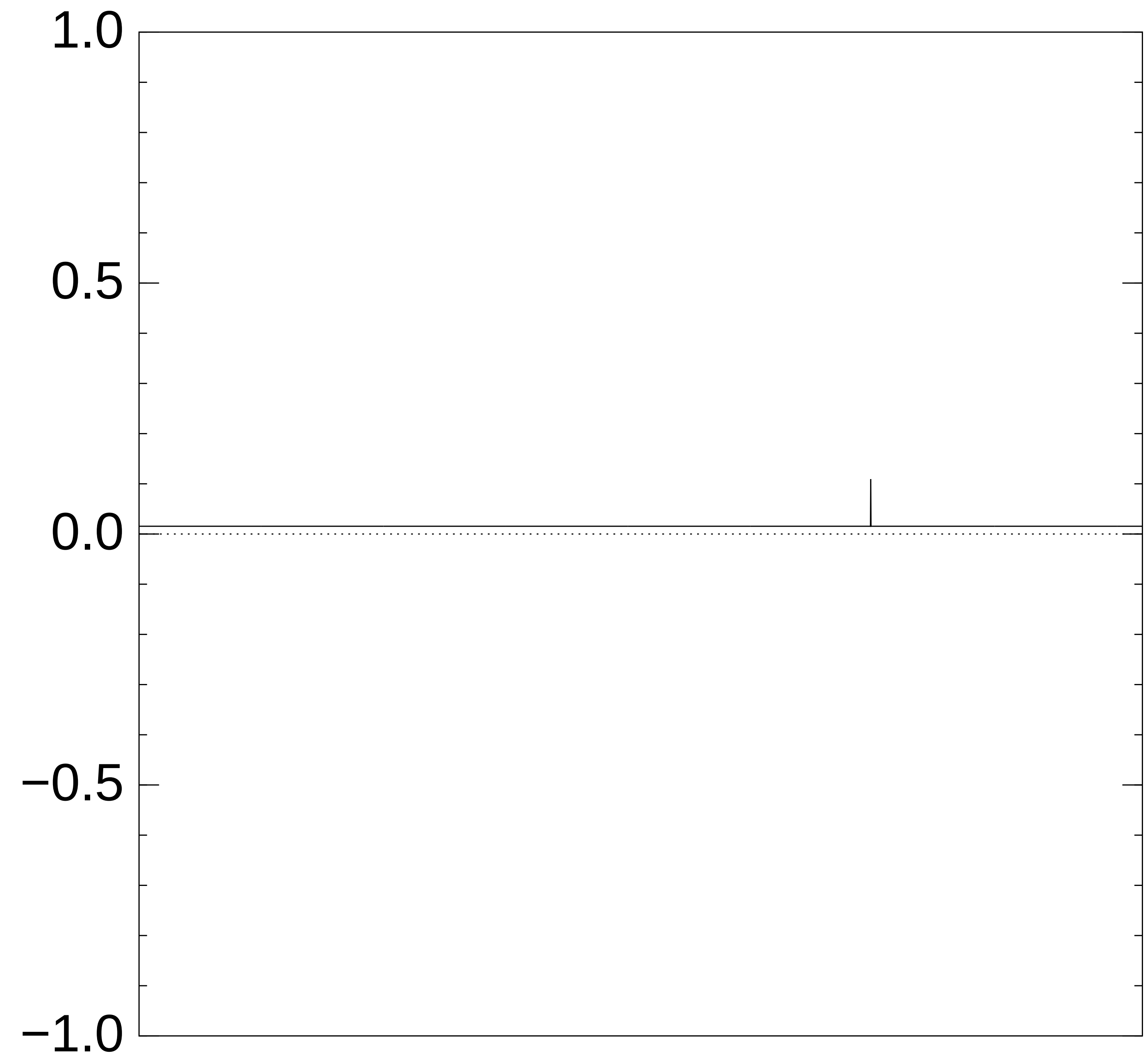
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $3 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

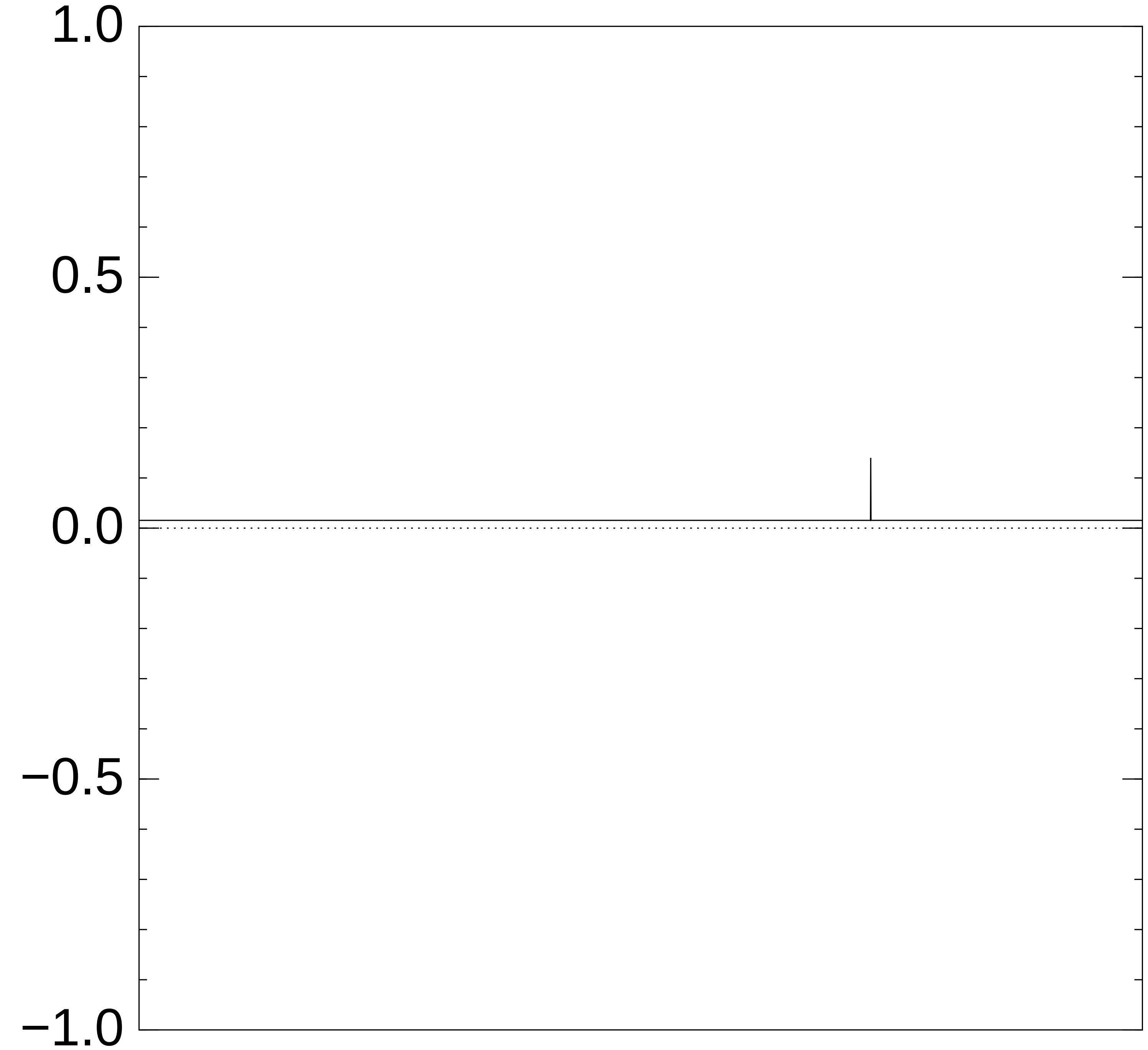
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $4 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

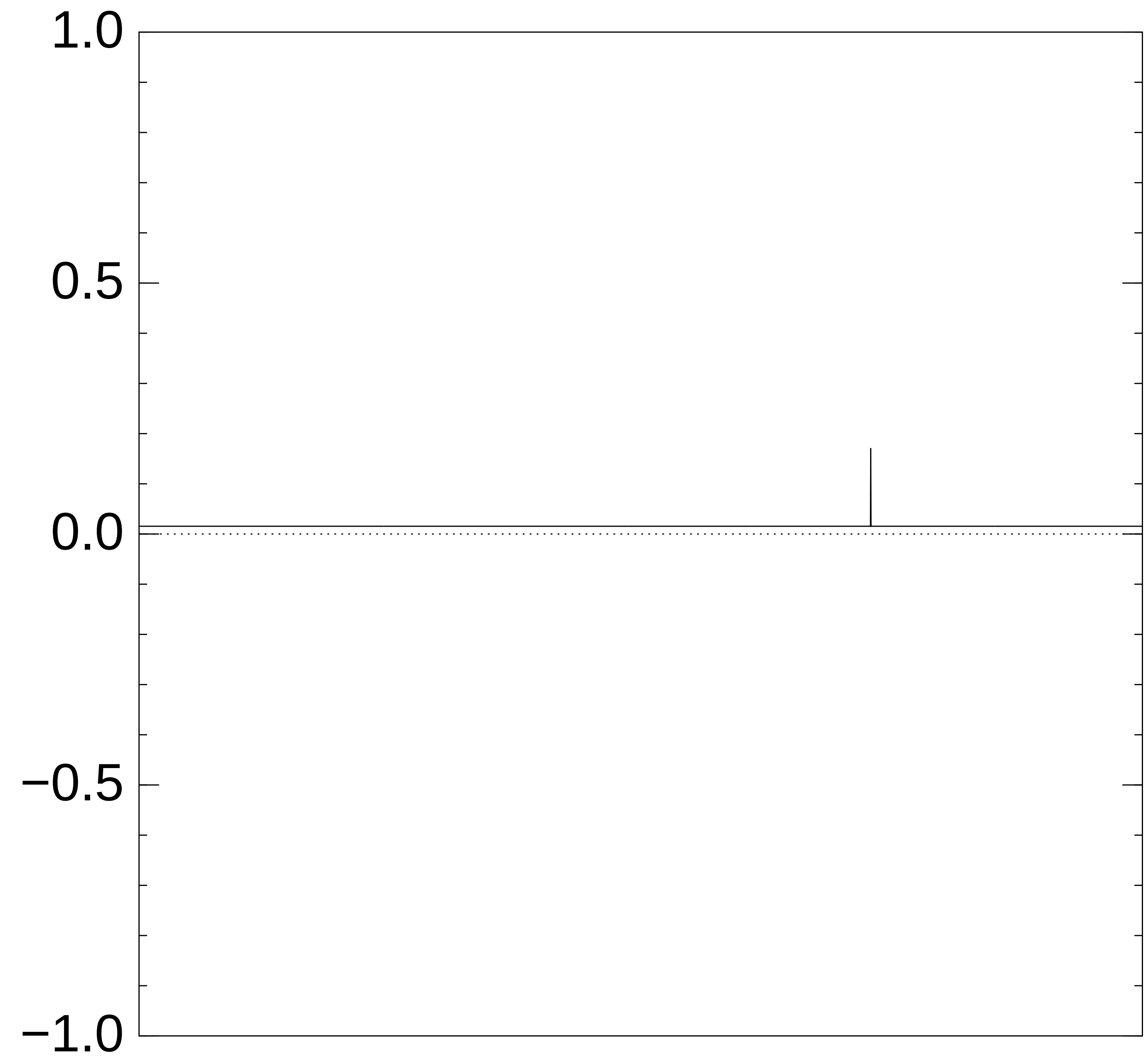
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $5 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

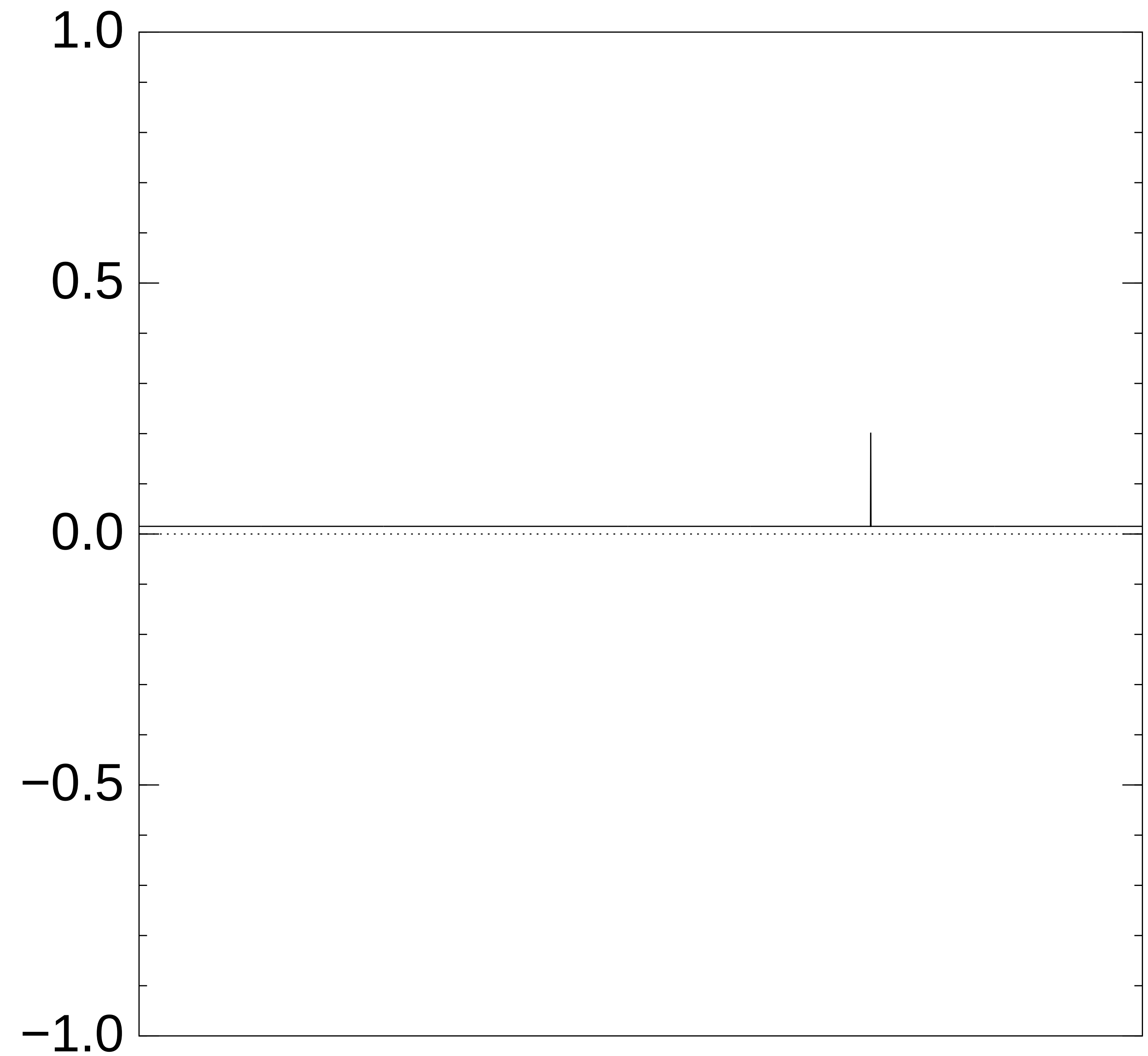
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $6 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

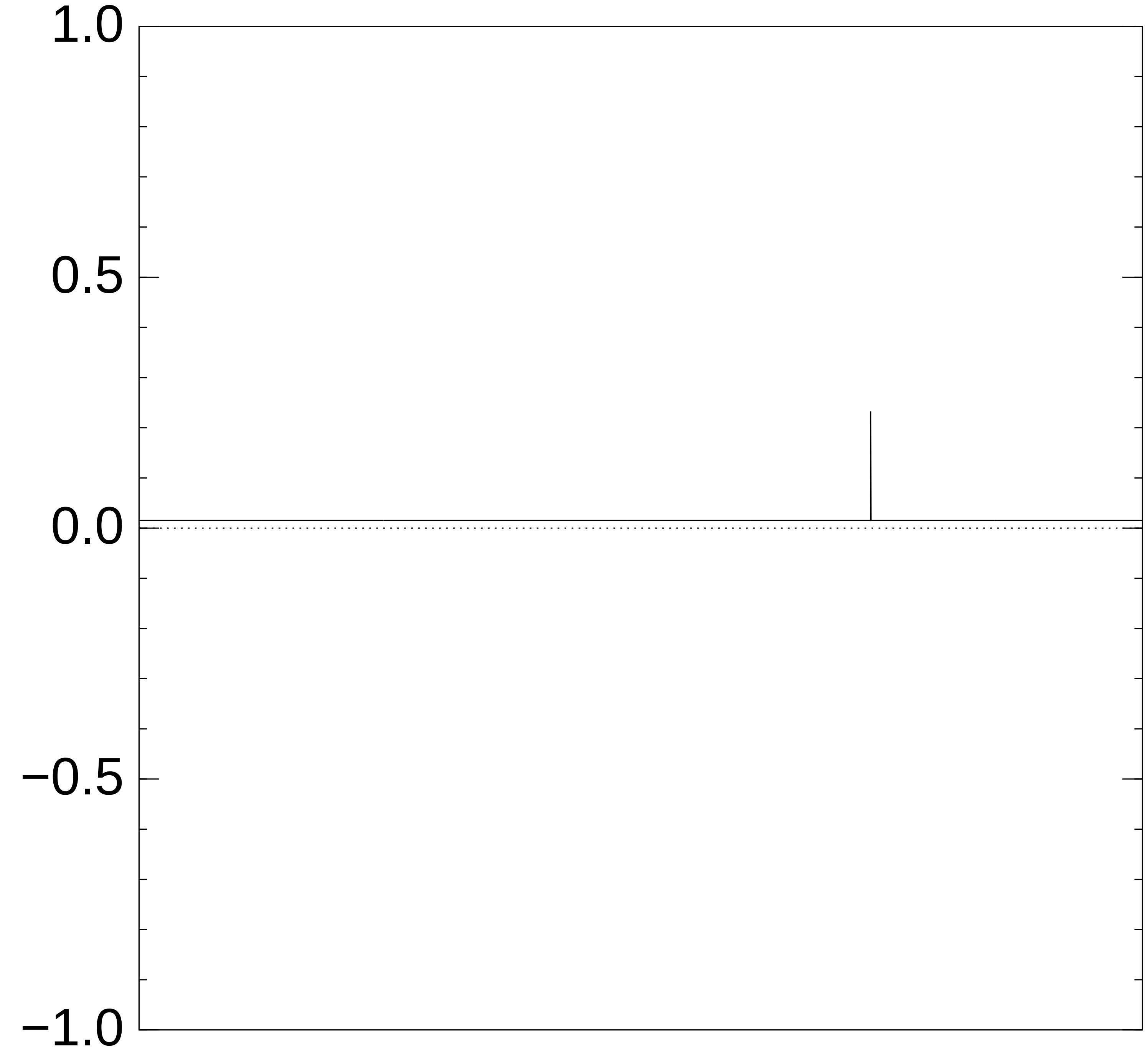
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $7 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

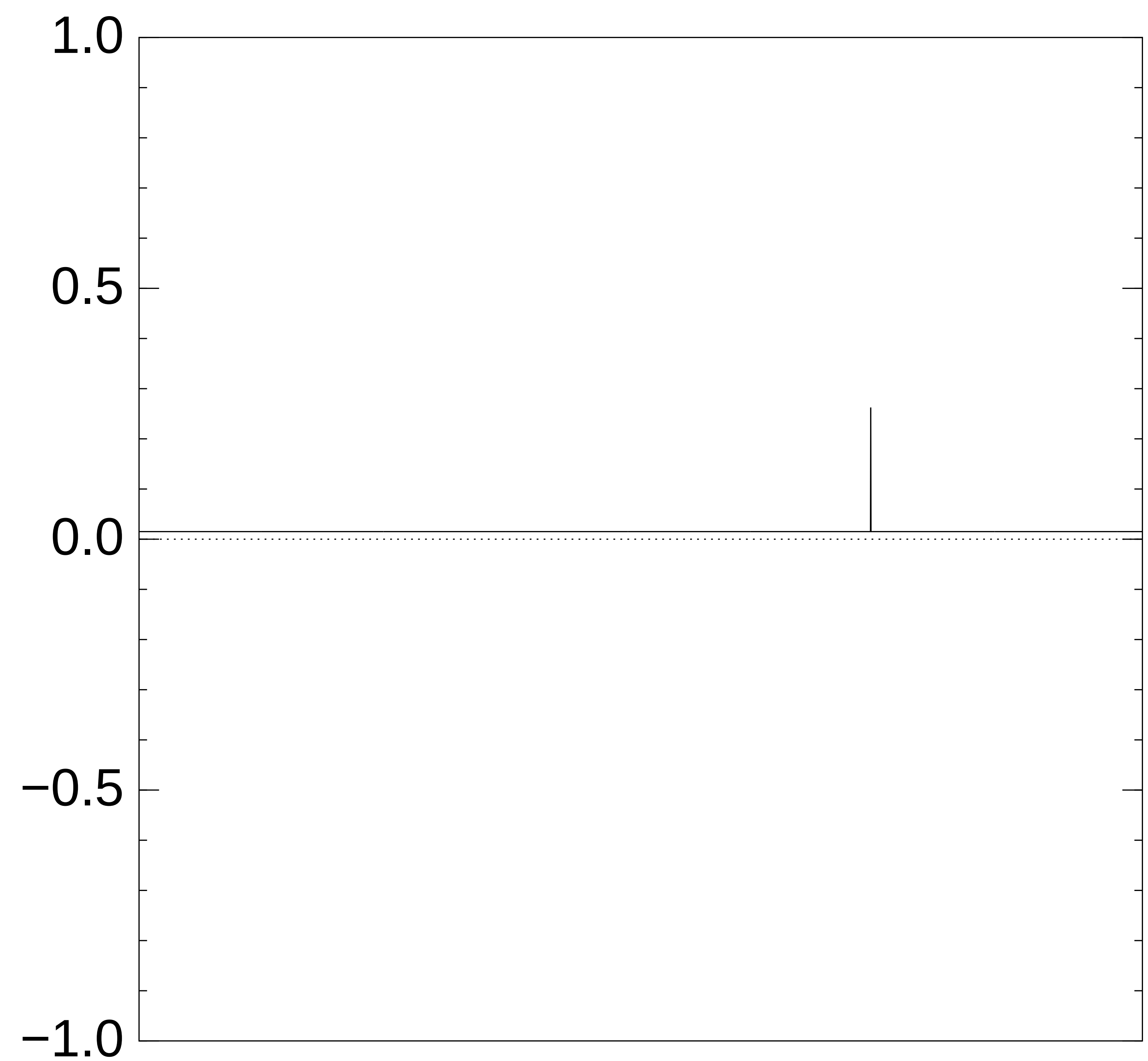
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $8 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

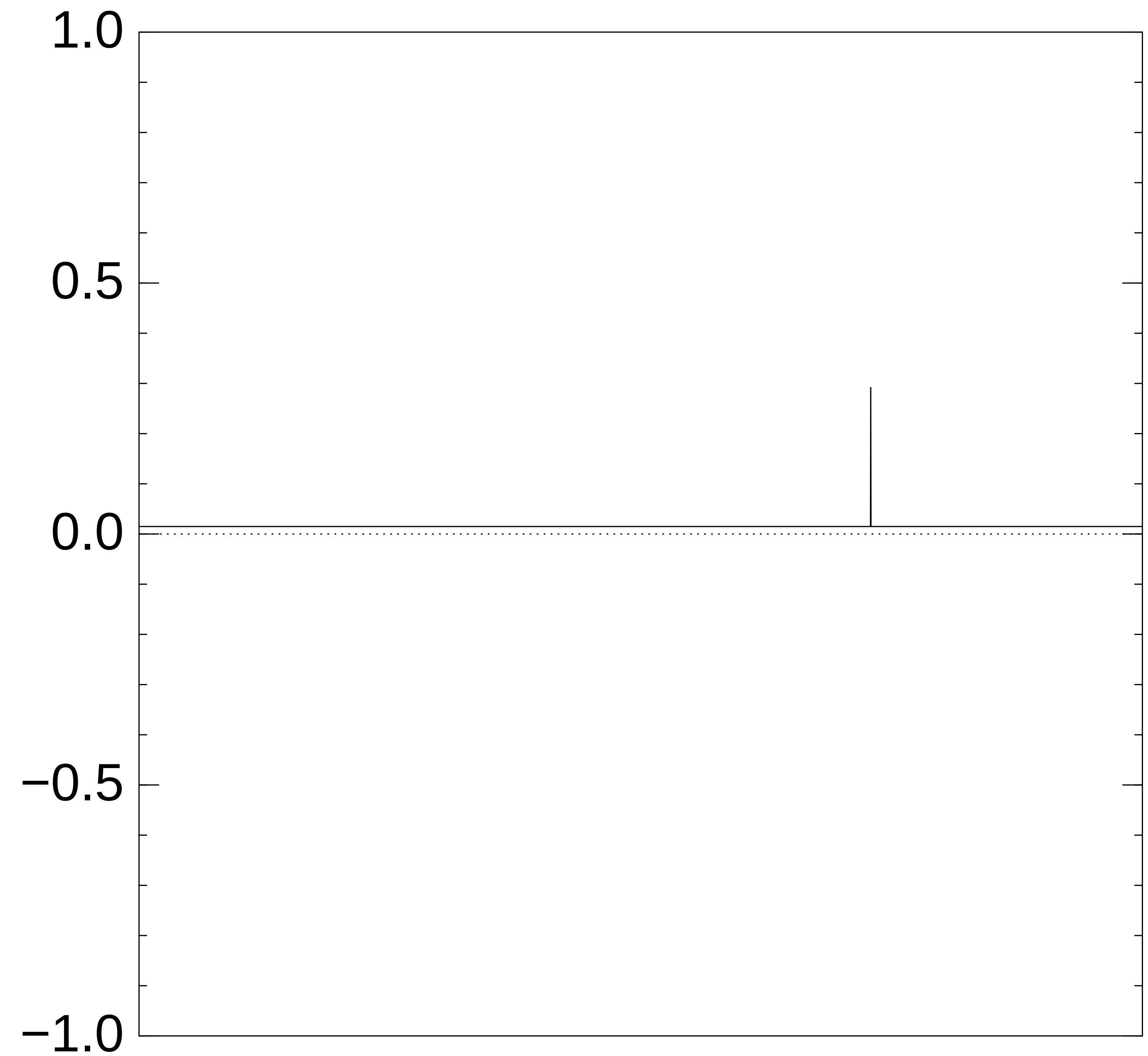
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $9 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

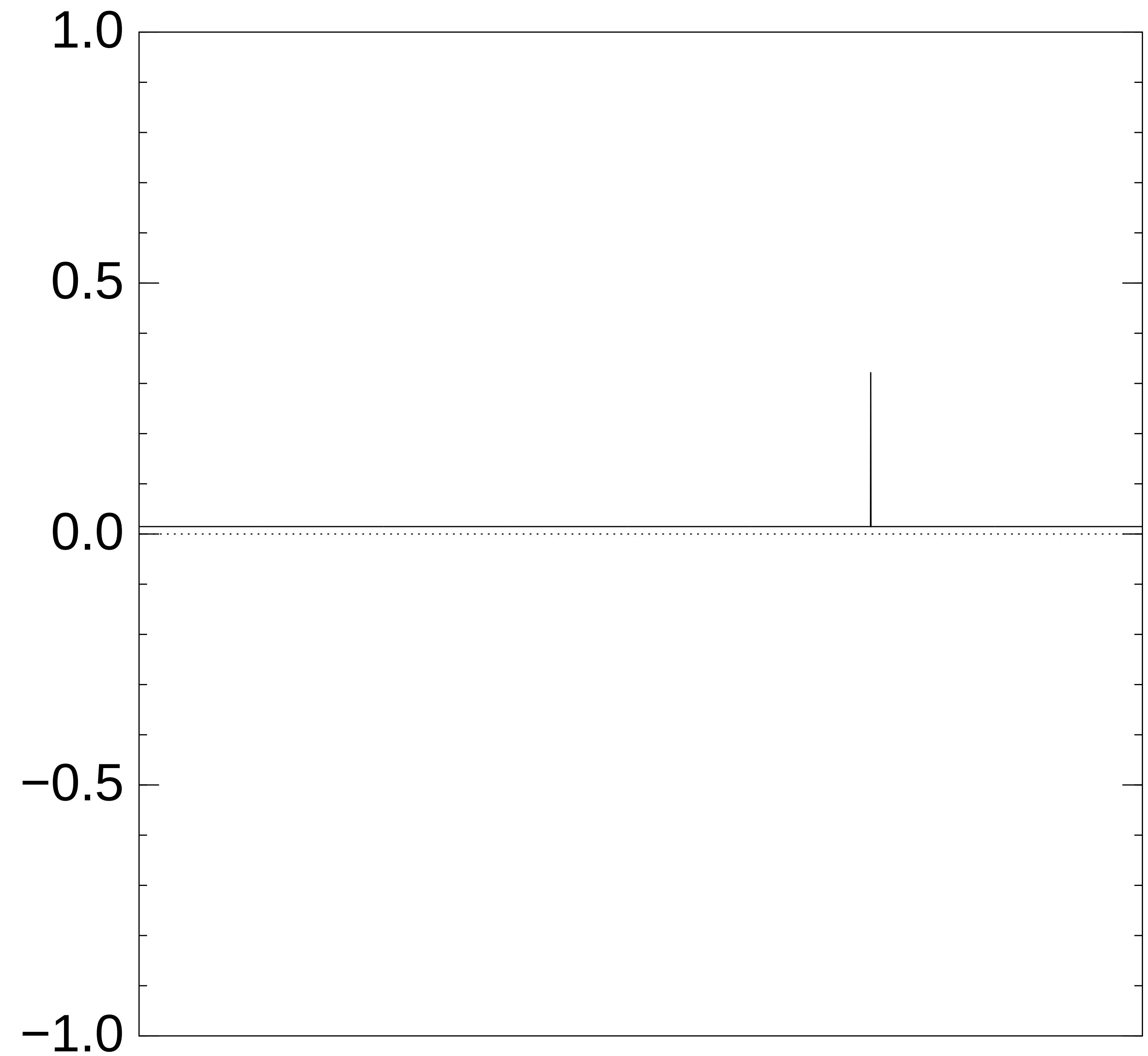
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $10 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

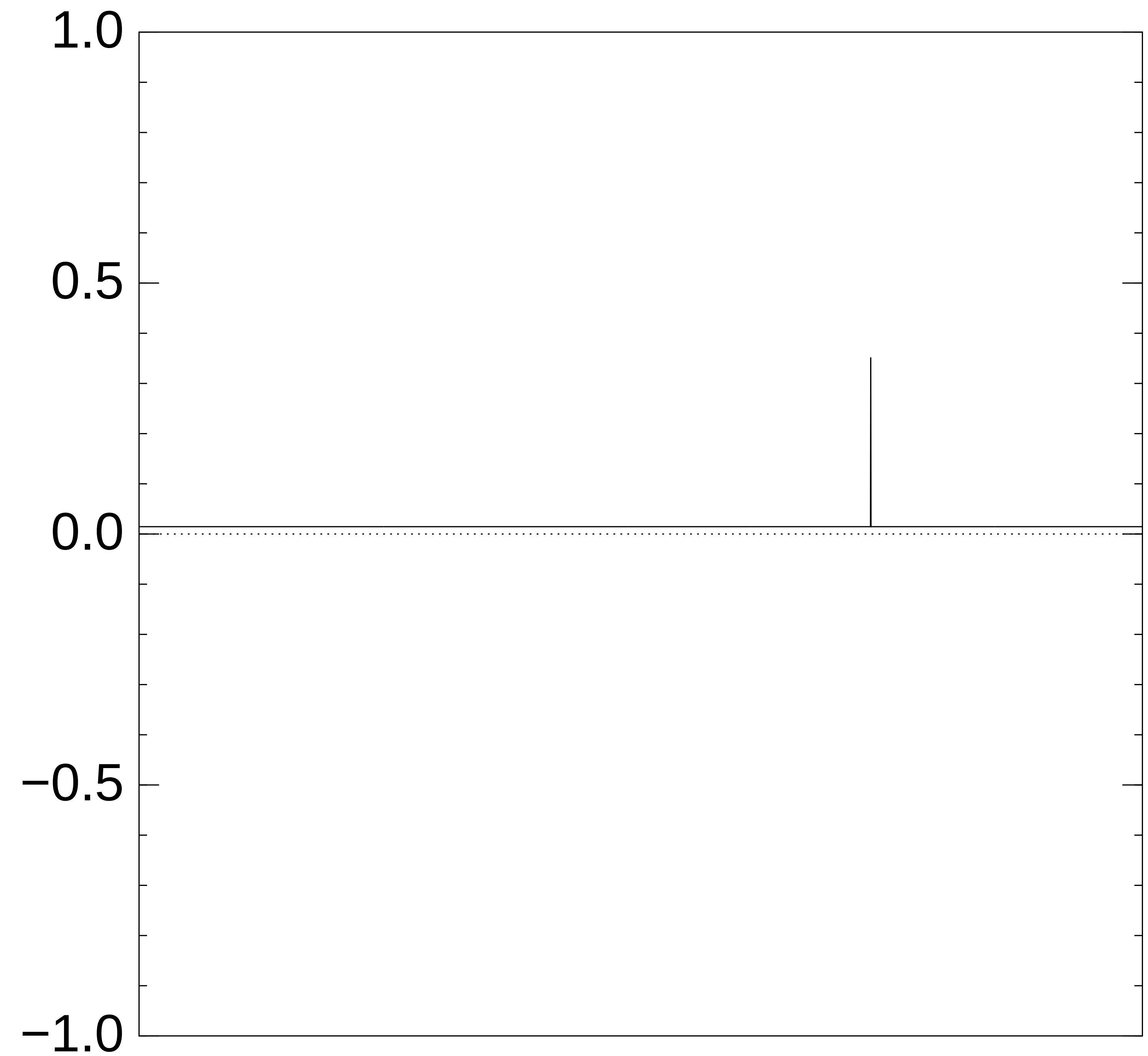
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $11 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

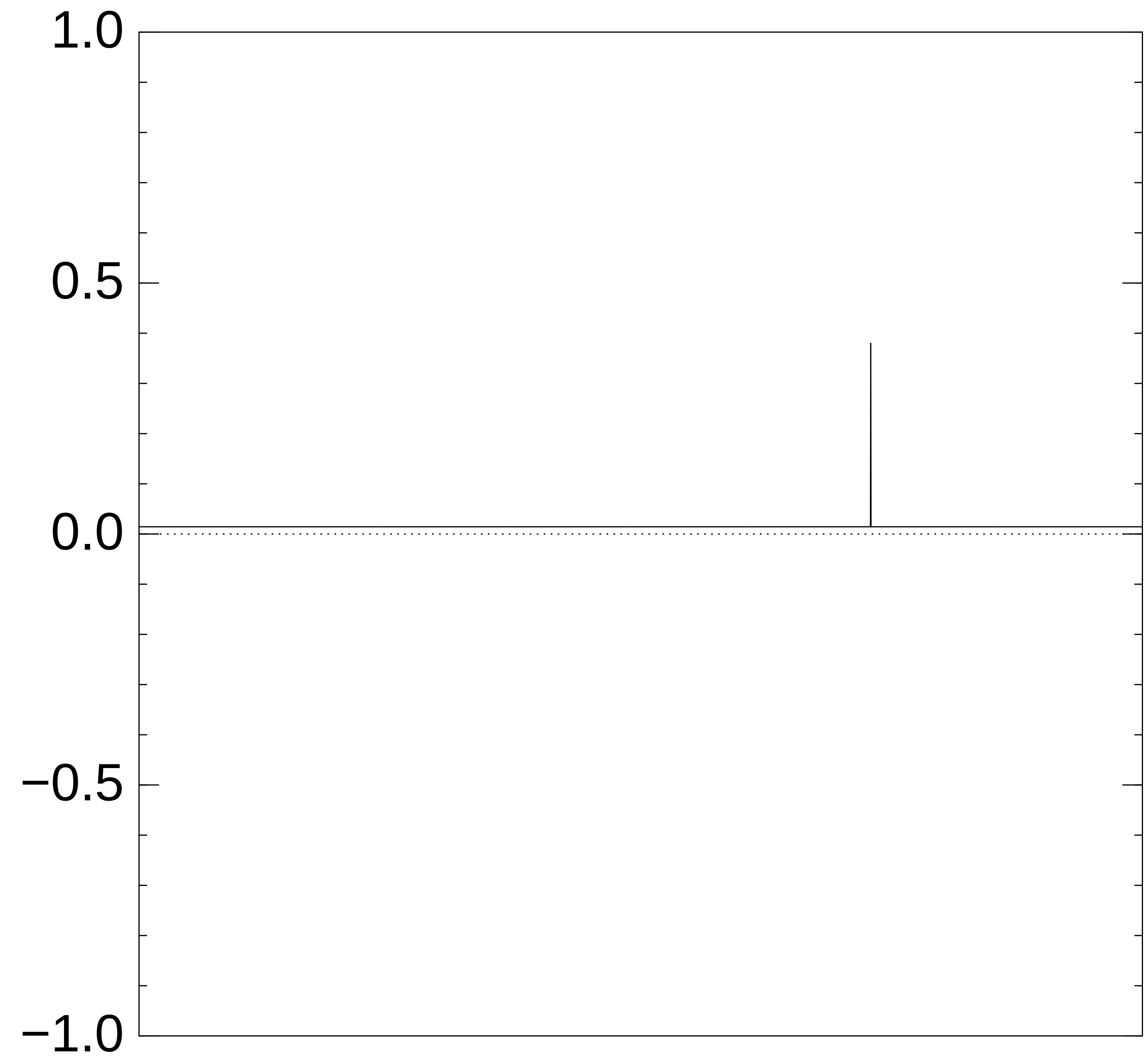
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $12 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

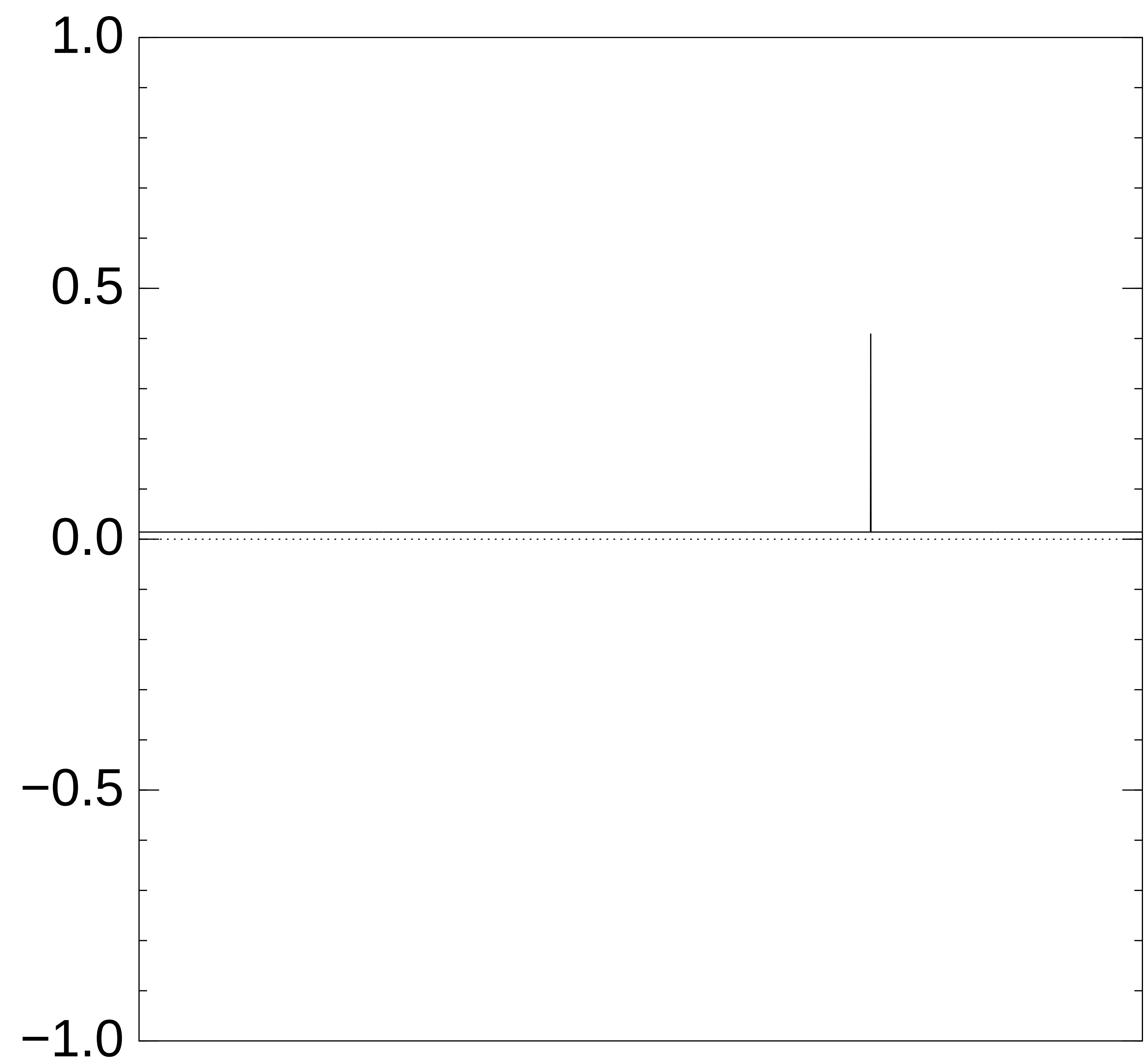
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $13 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

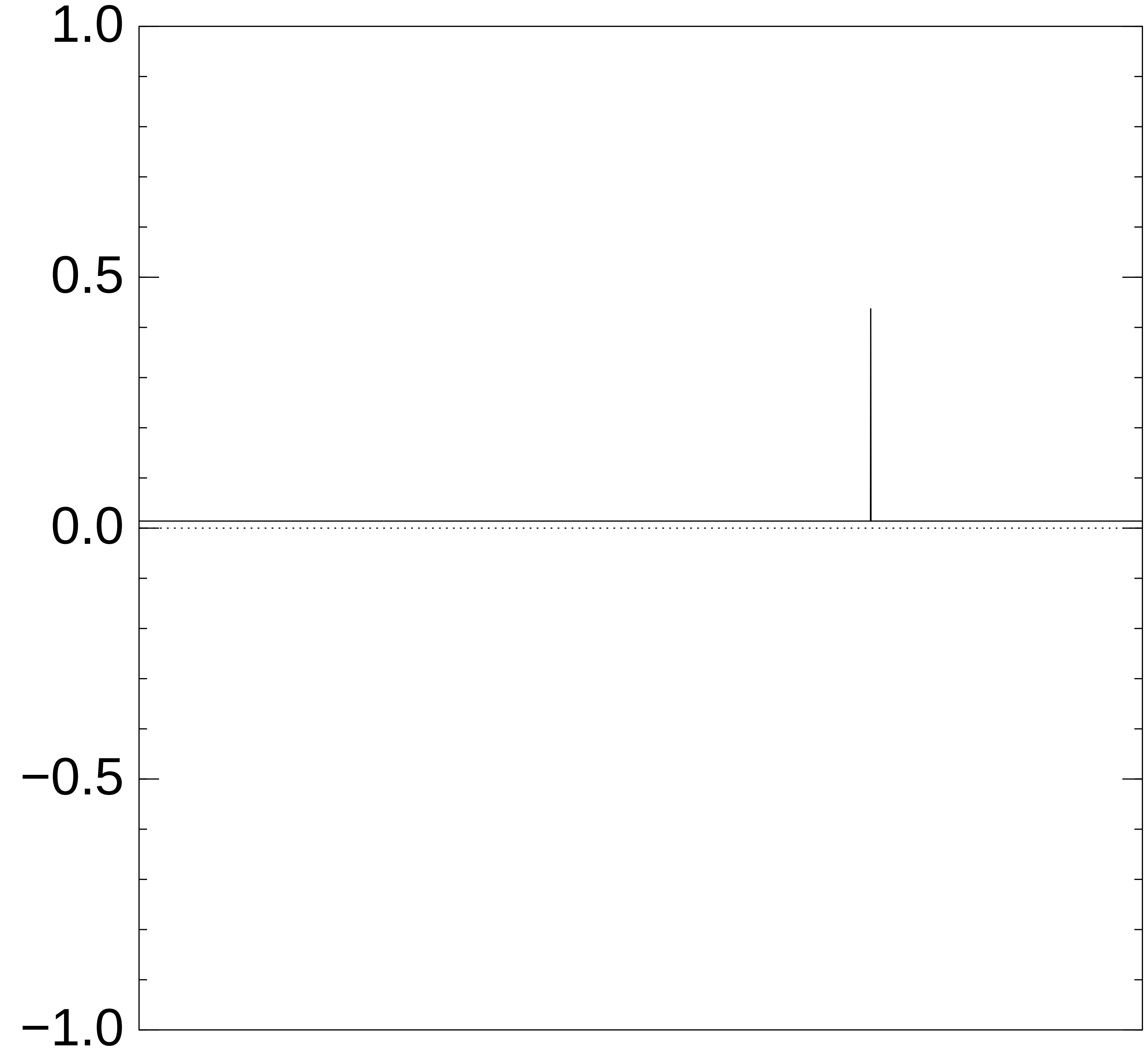
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $14 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

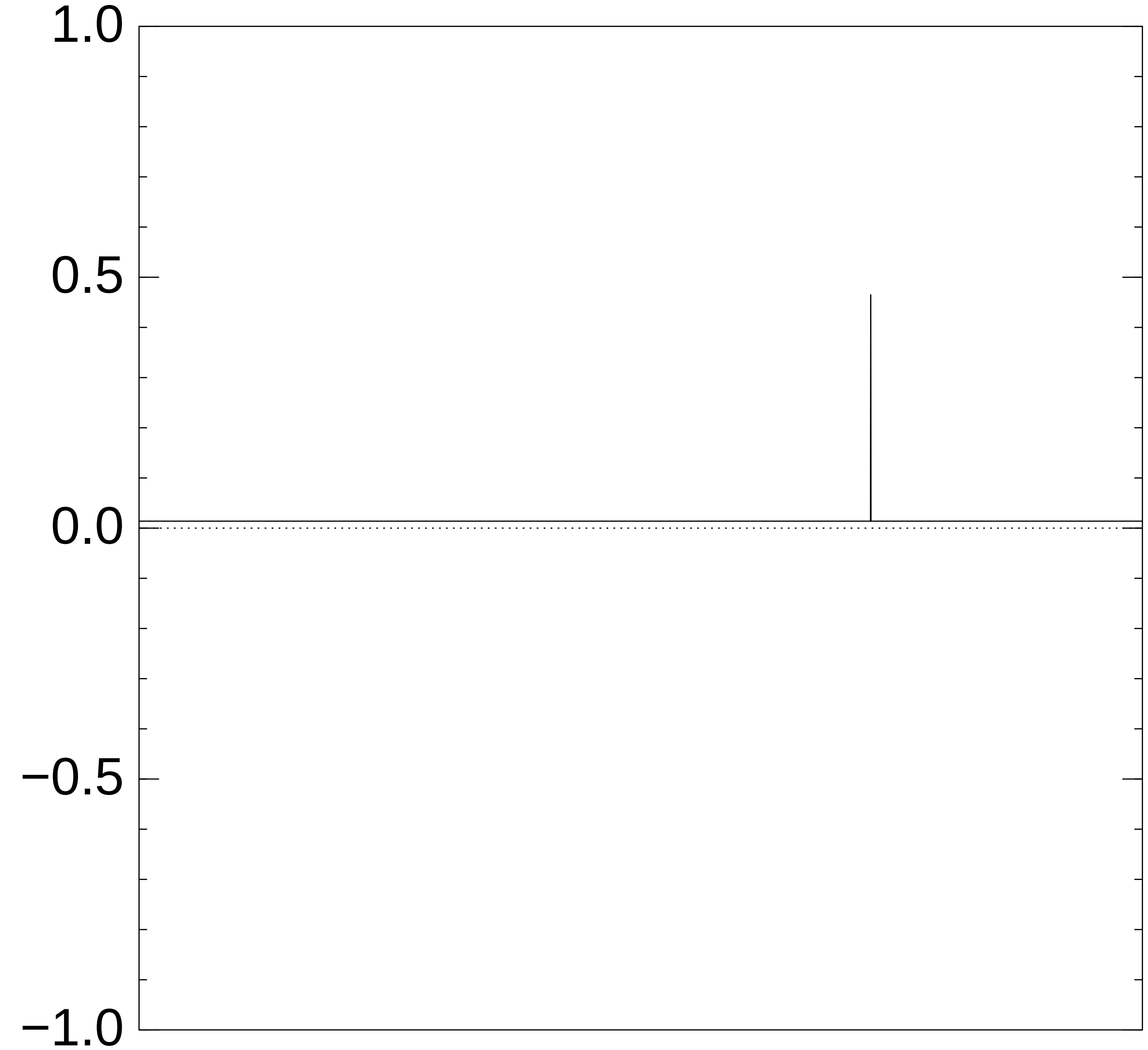
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $15 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

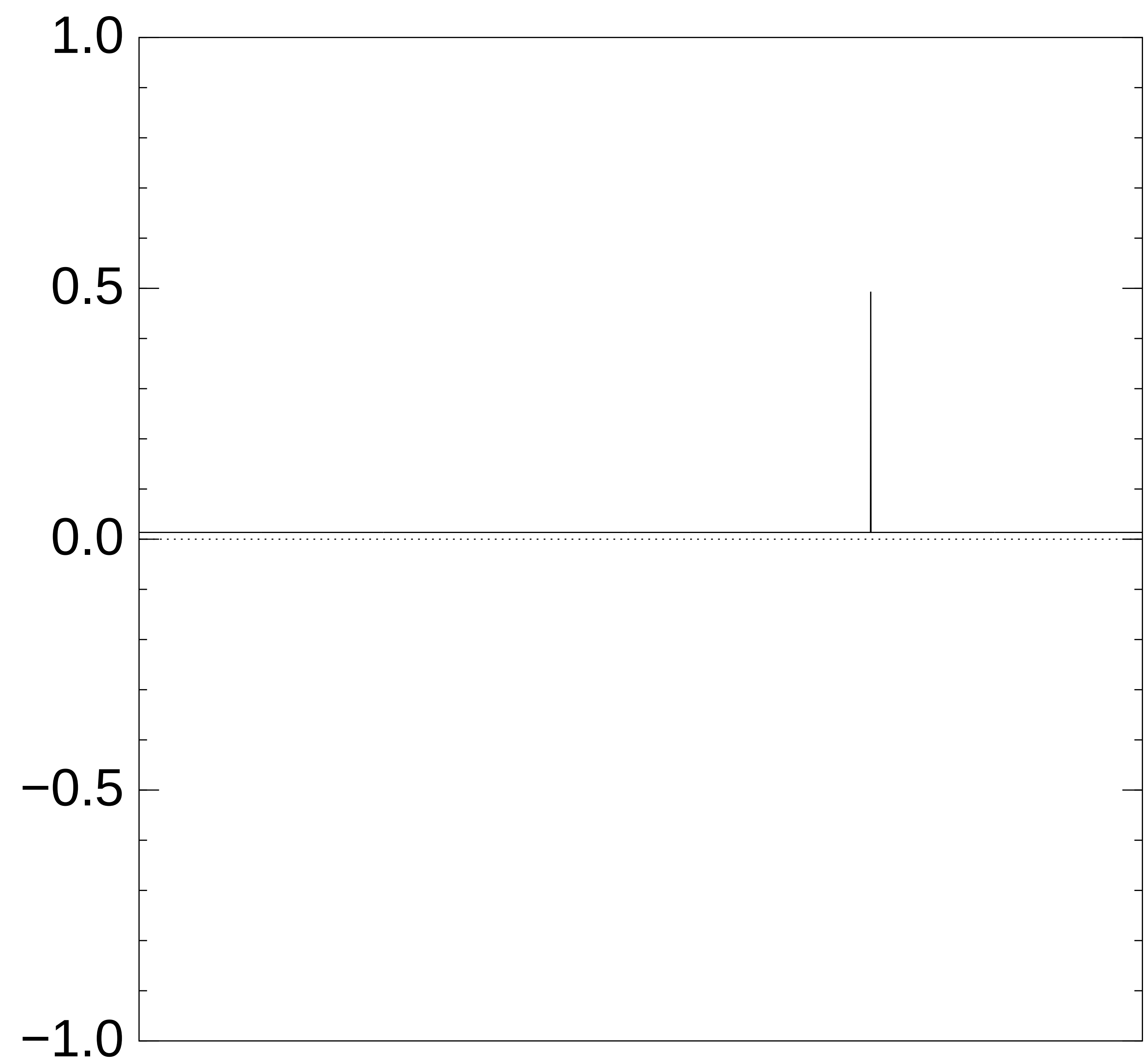
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $16 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

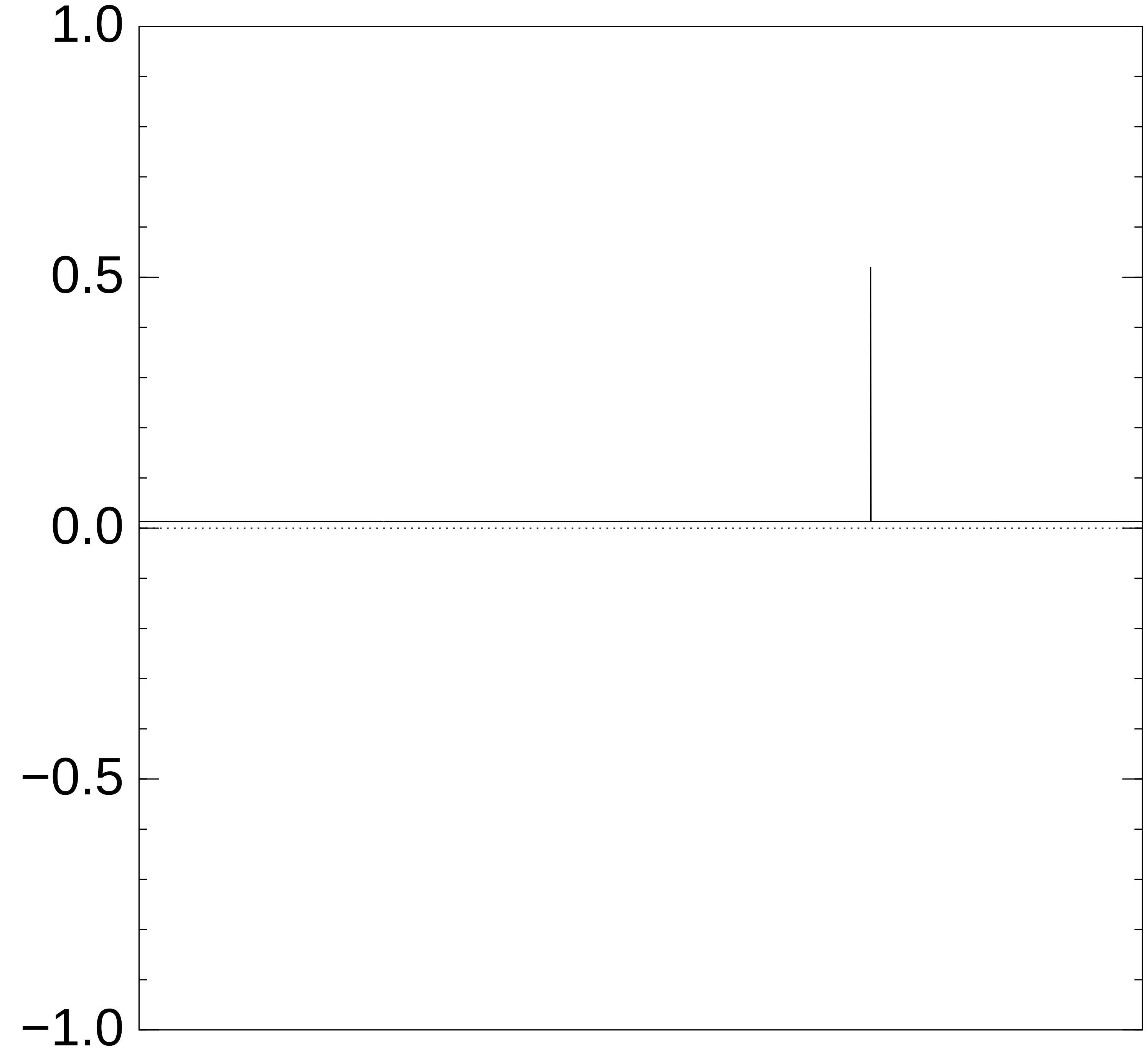
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $17 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

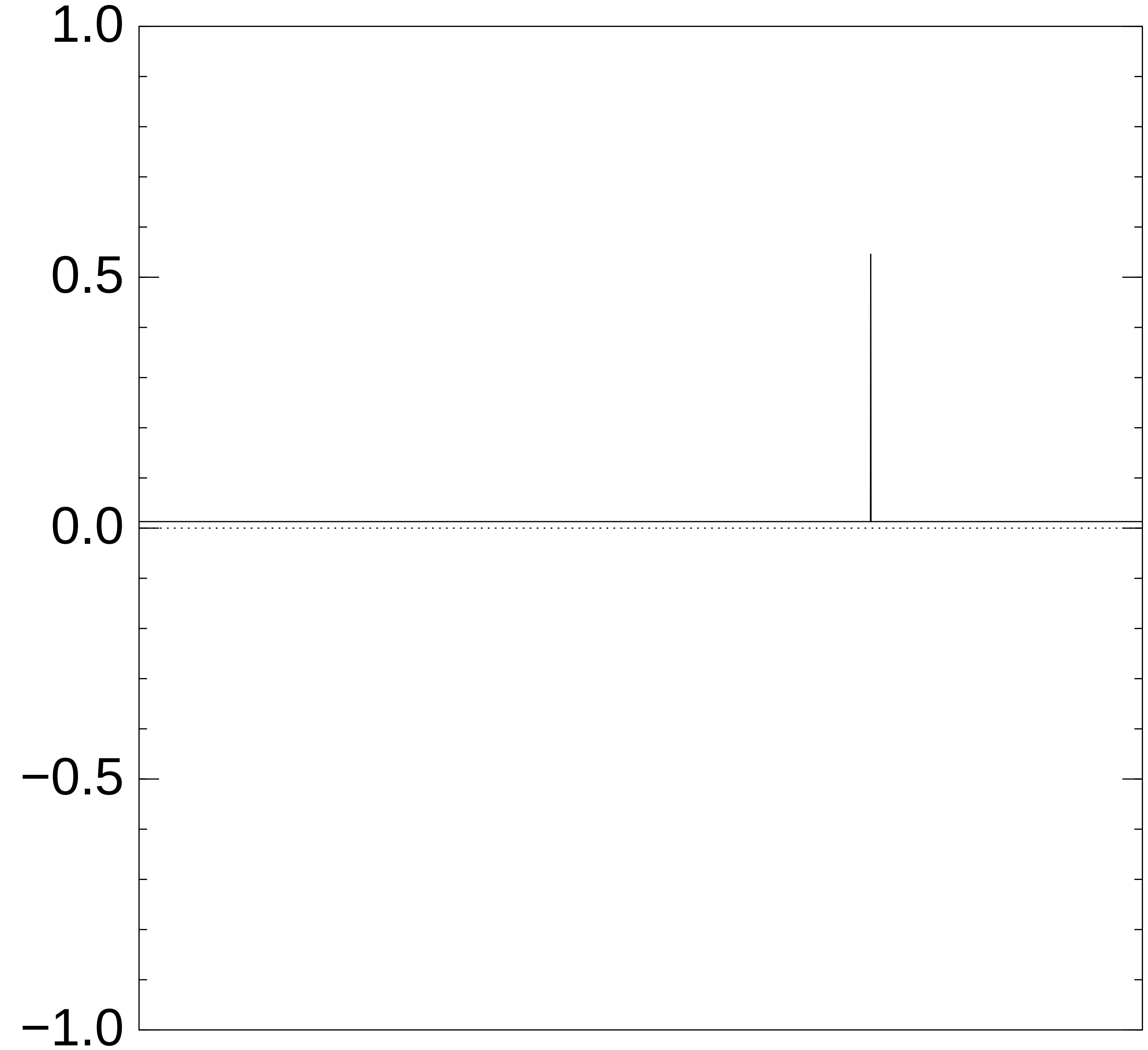
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $18 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

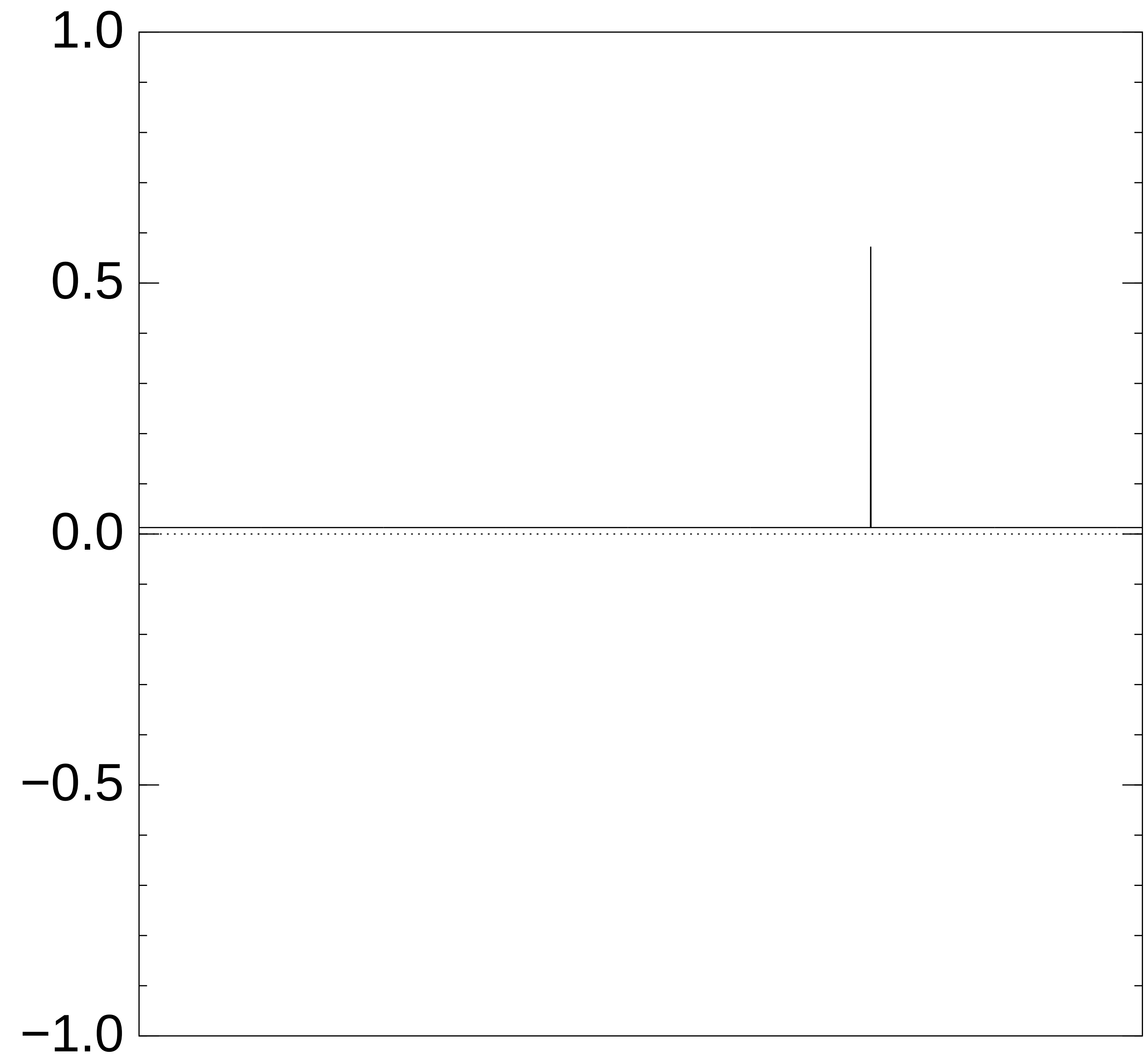
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $19 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

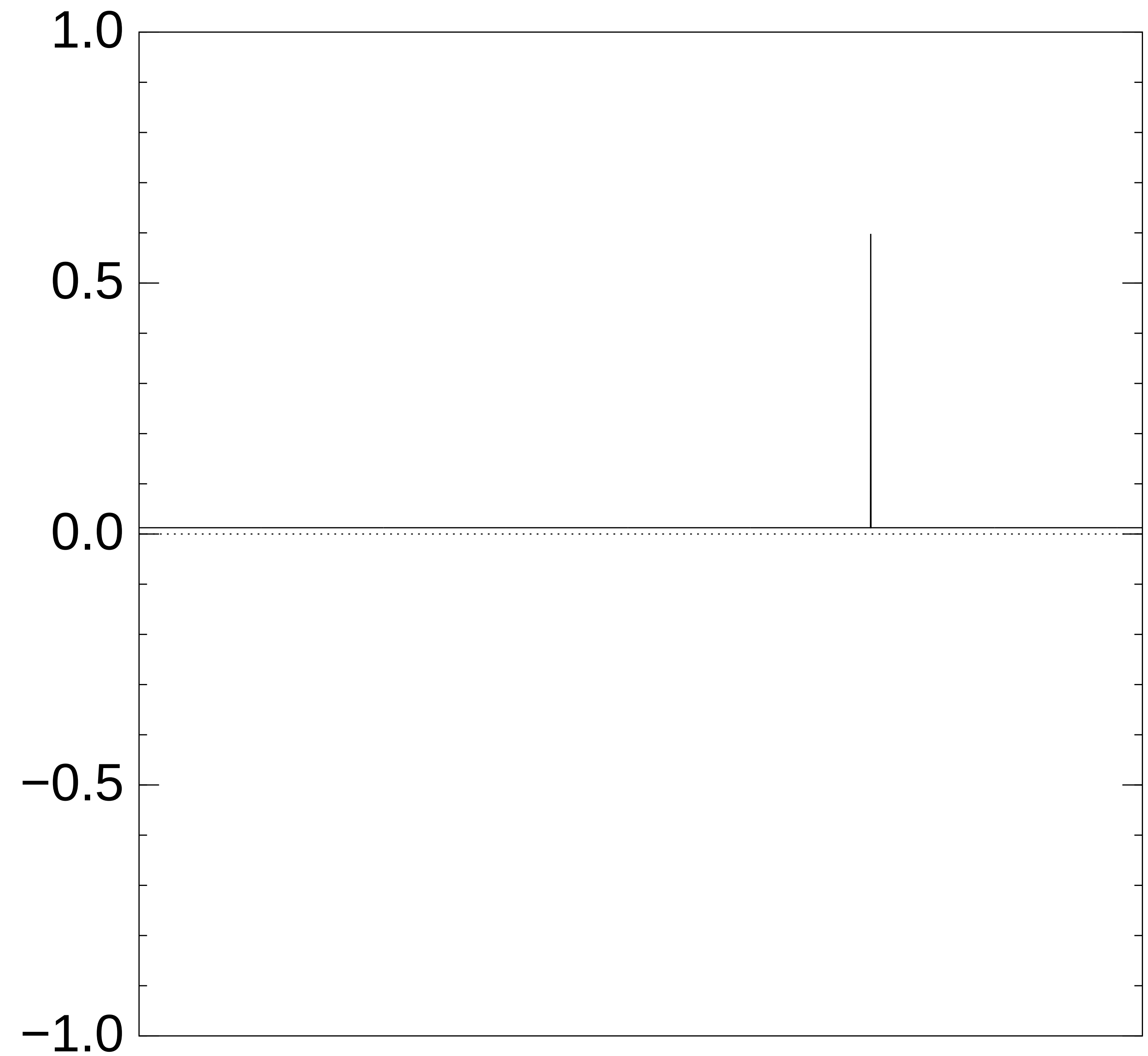
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $20 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

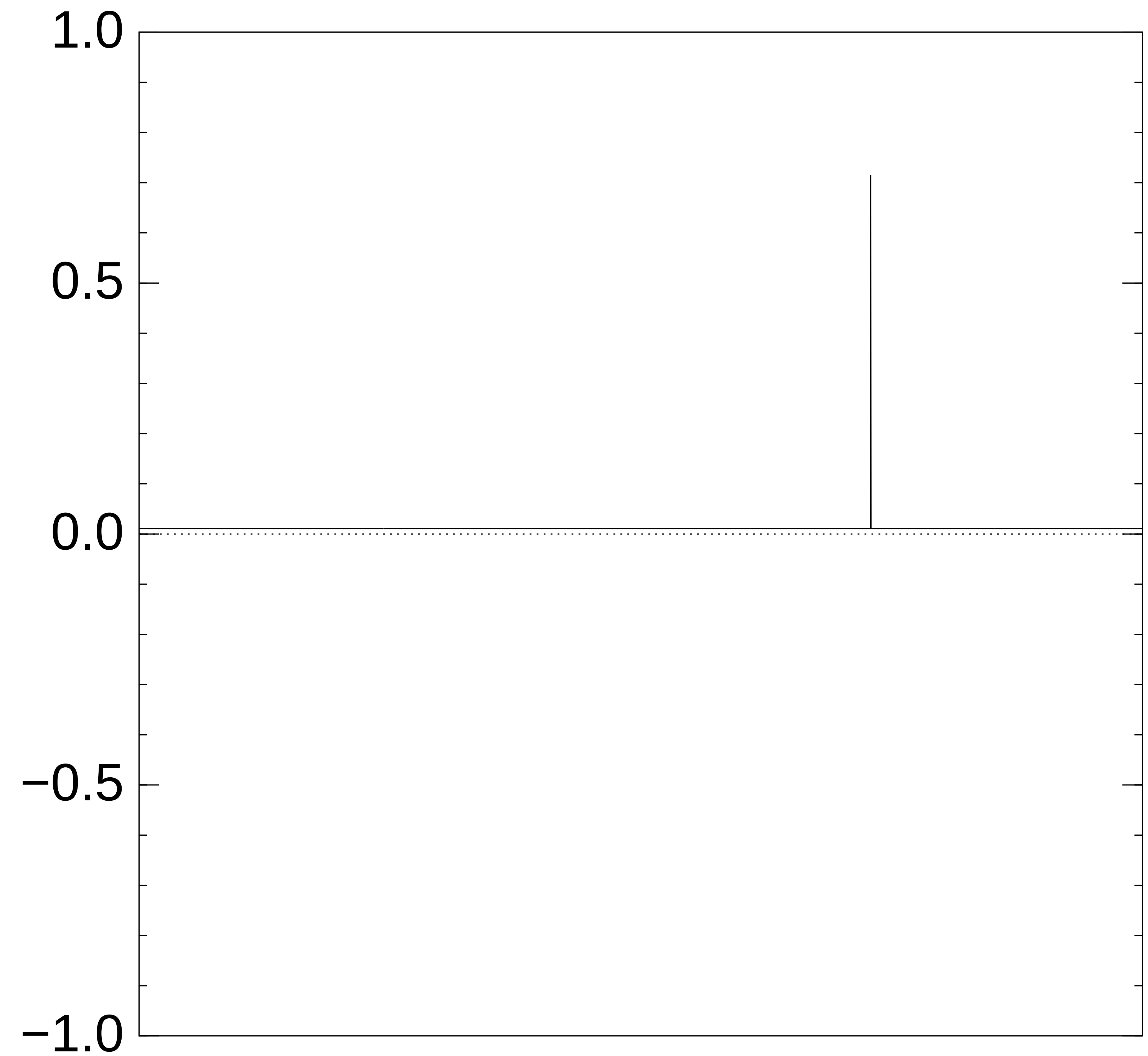
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $25 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

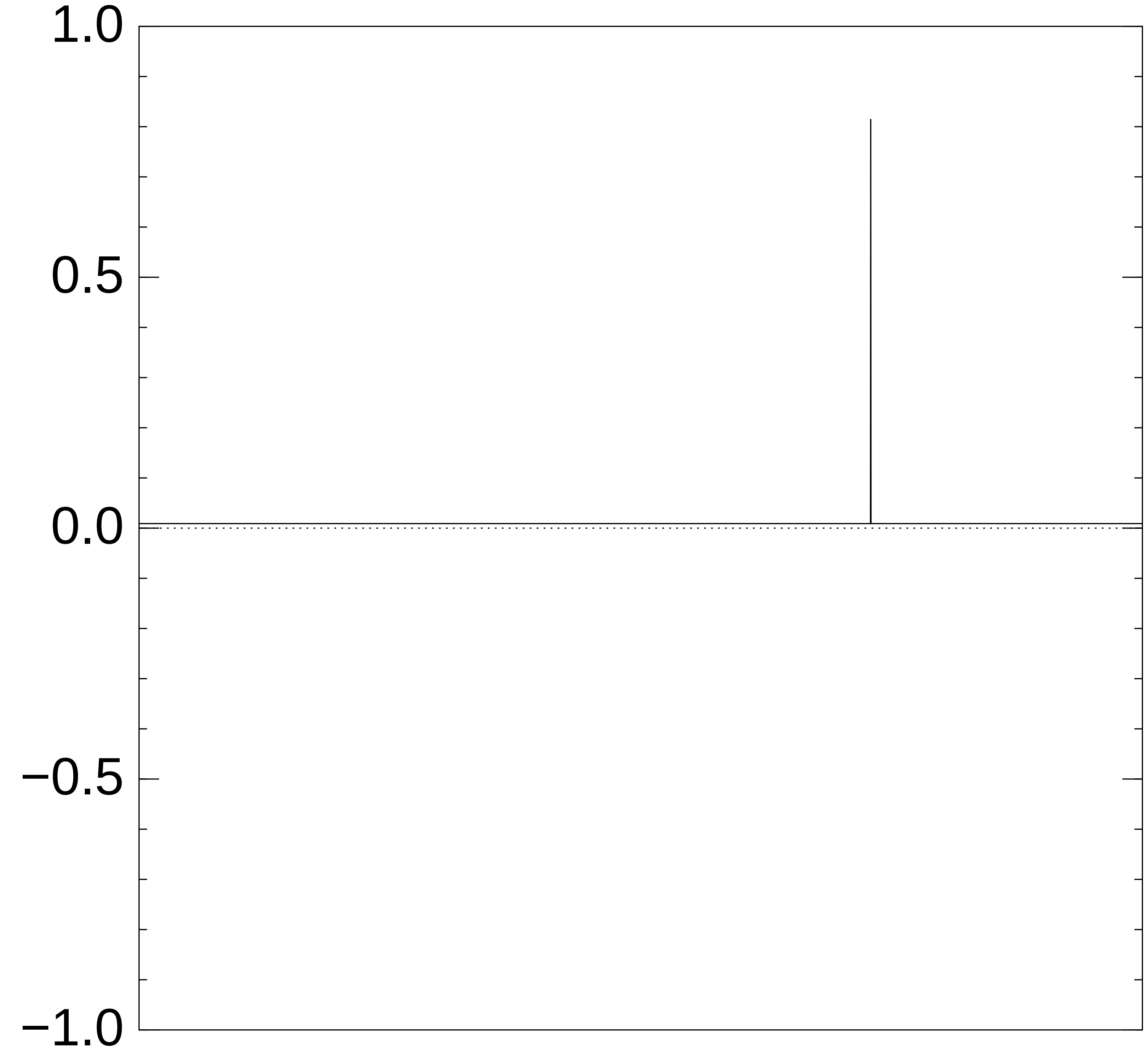
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $30 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

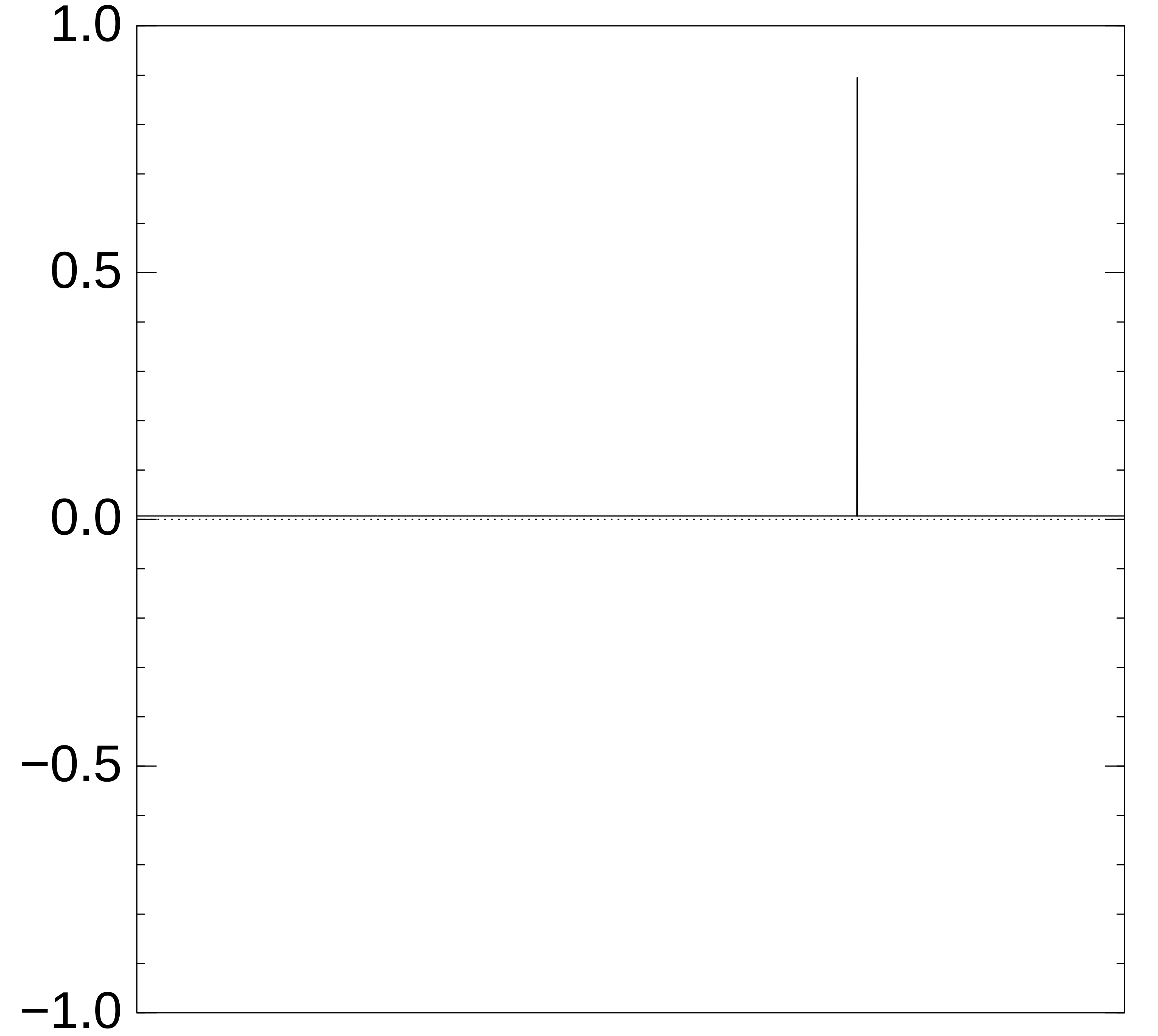
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $35 \times$ (Step 1 + Step 2):



Good moment to stop, measure.

Start from uniform superposition over all n-bit strings q.

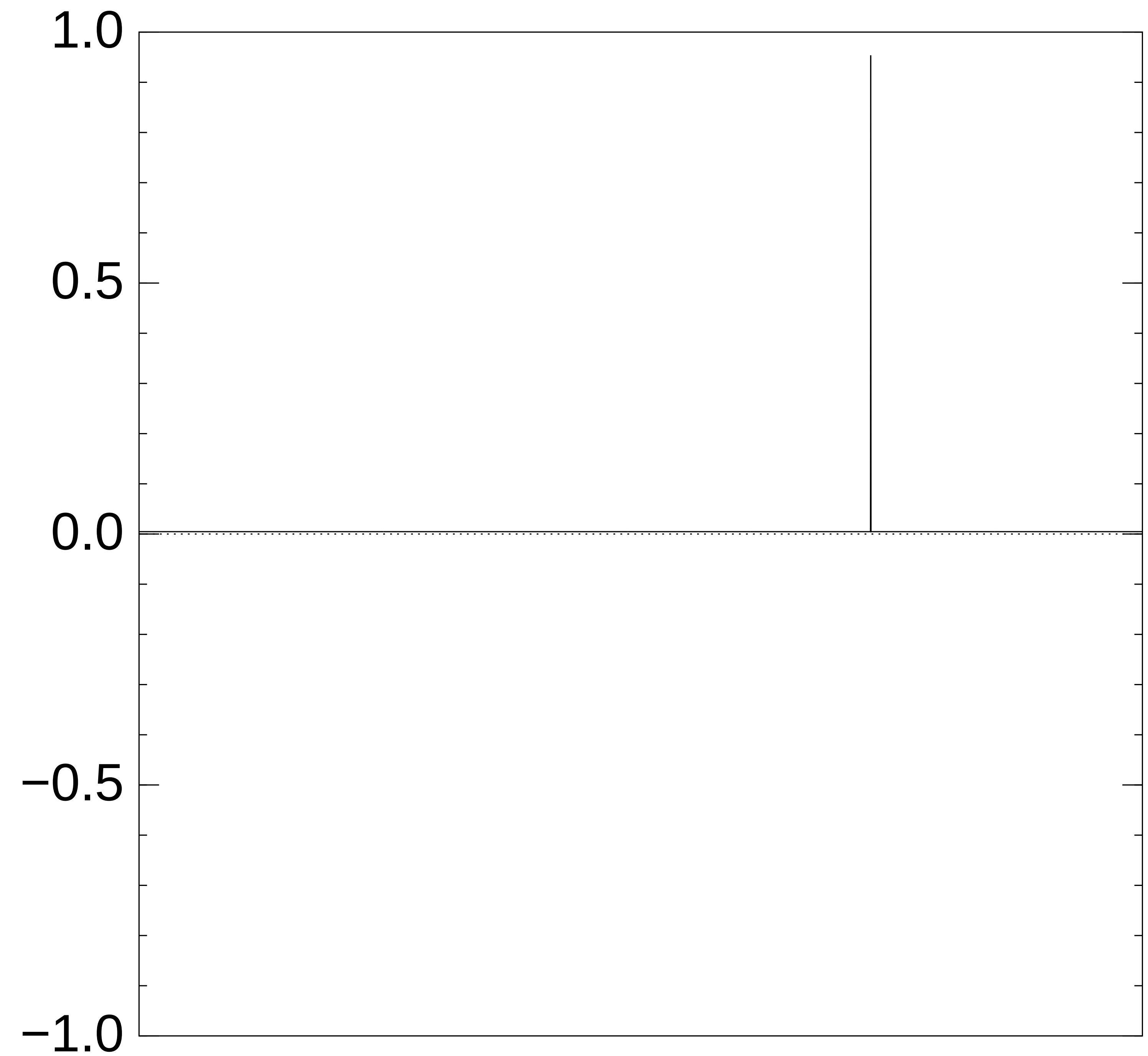
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $40 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

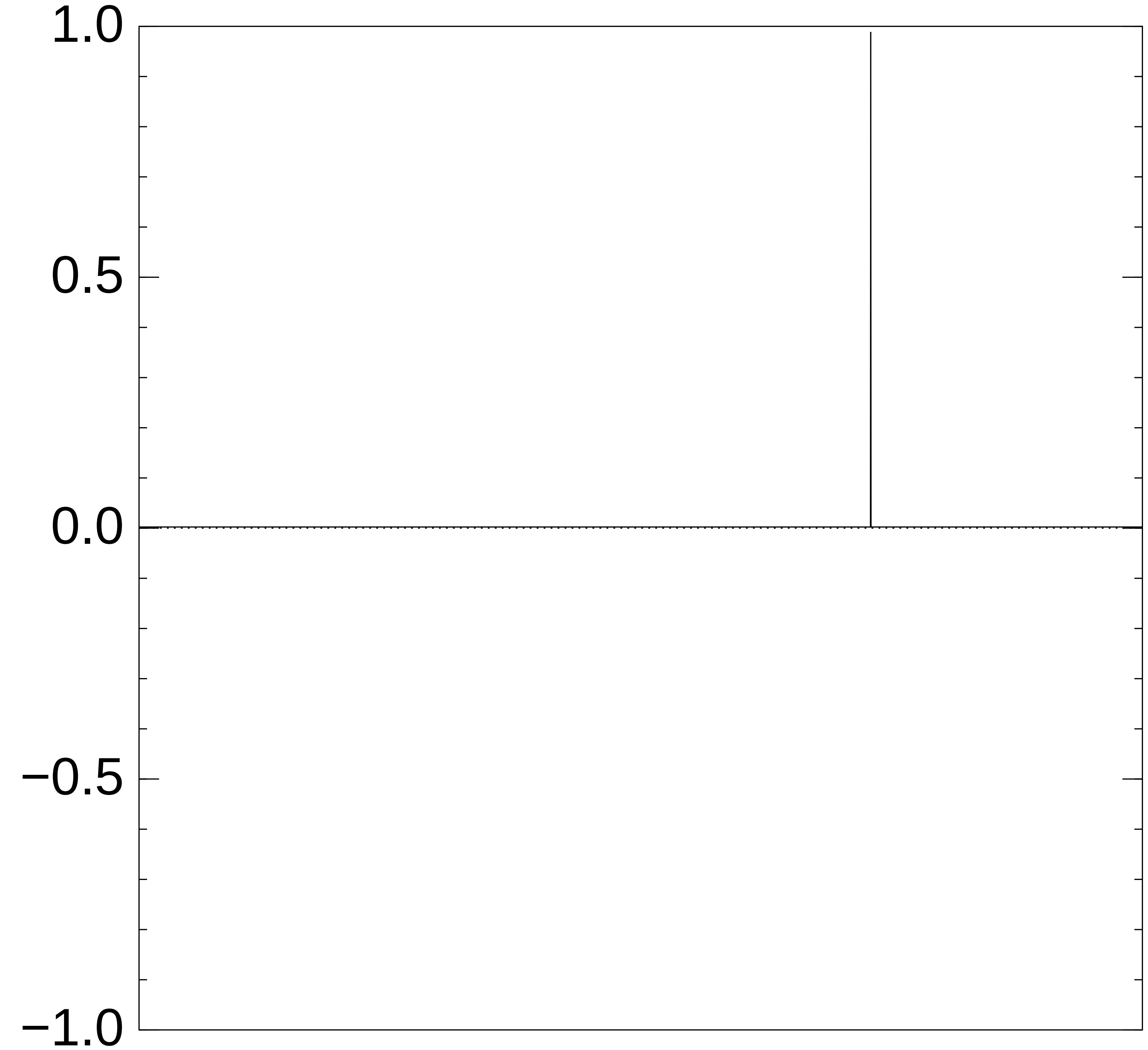
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $45 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

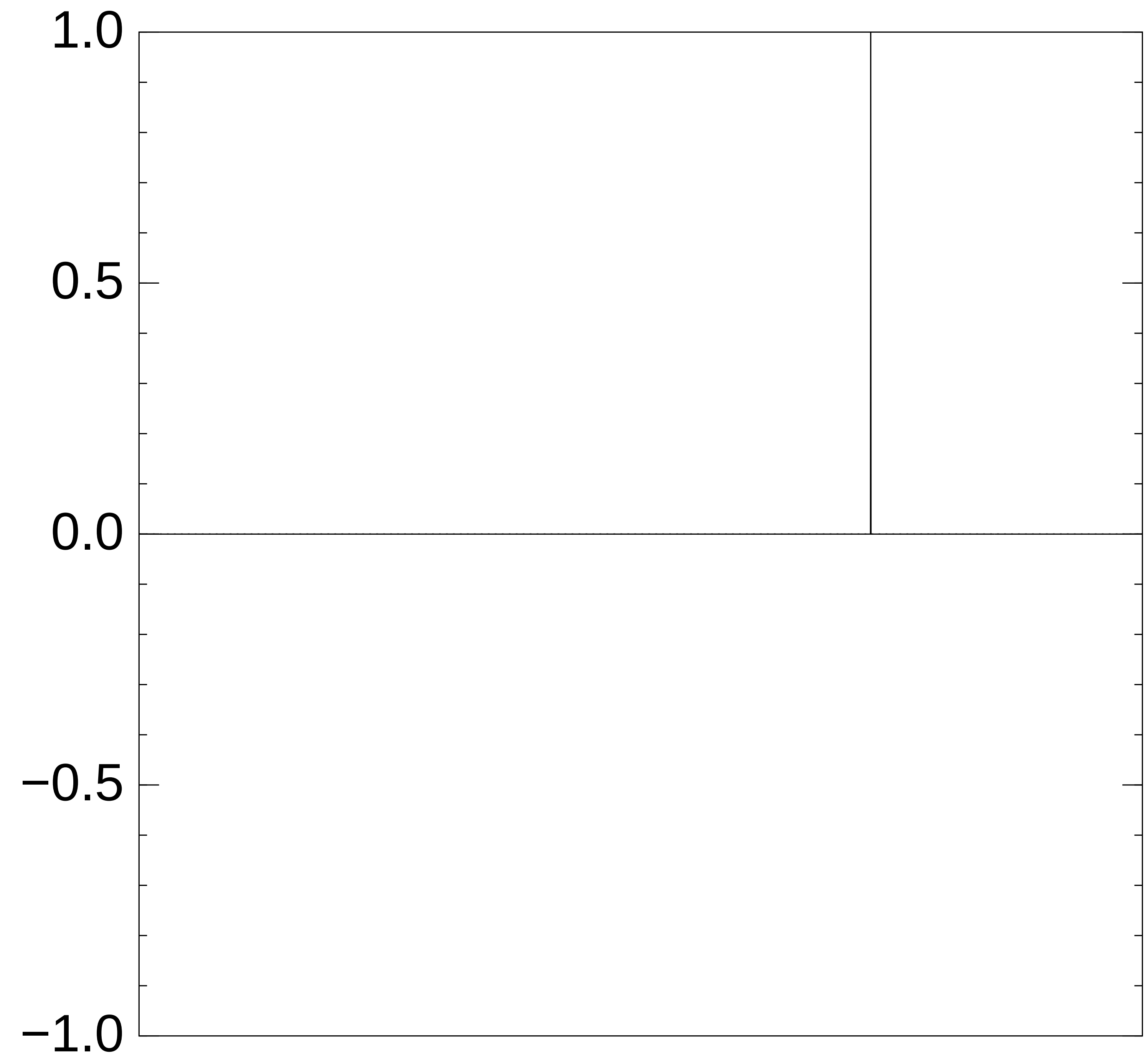
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $50 \times$ (Step 1 + Step 2):



Traditional stopping point.

Start from uniform superposition over all n-bit strings q.

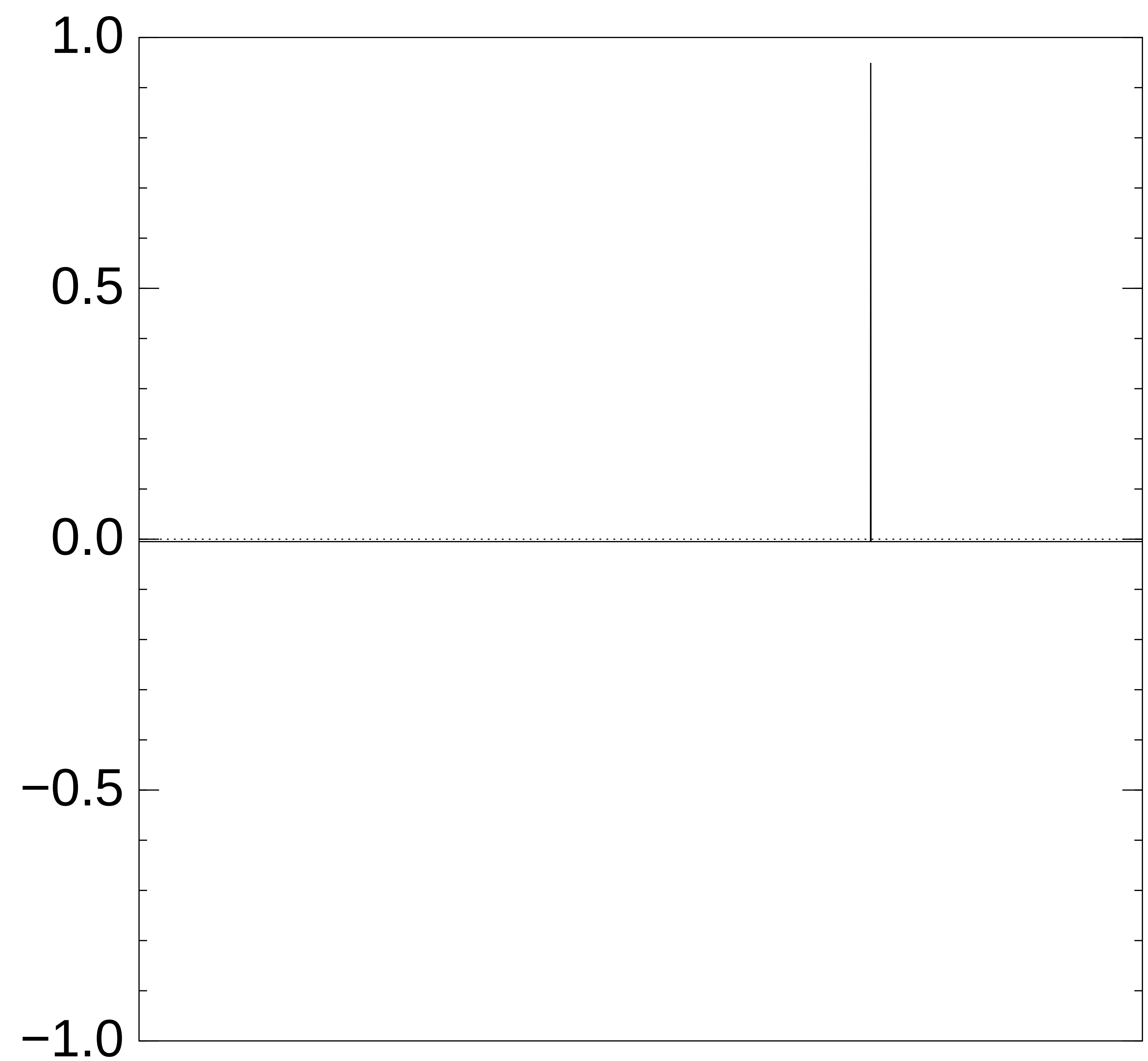
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $60 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

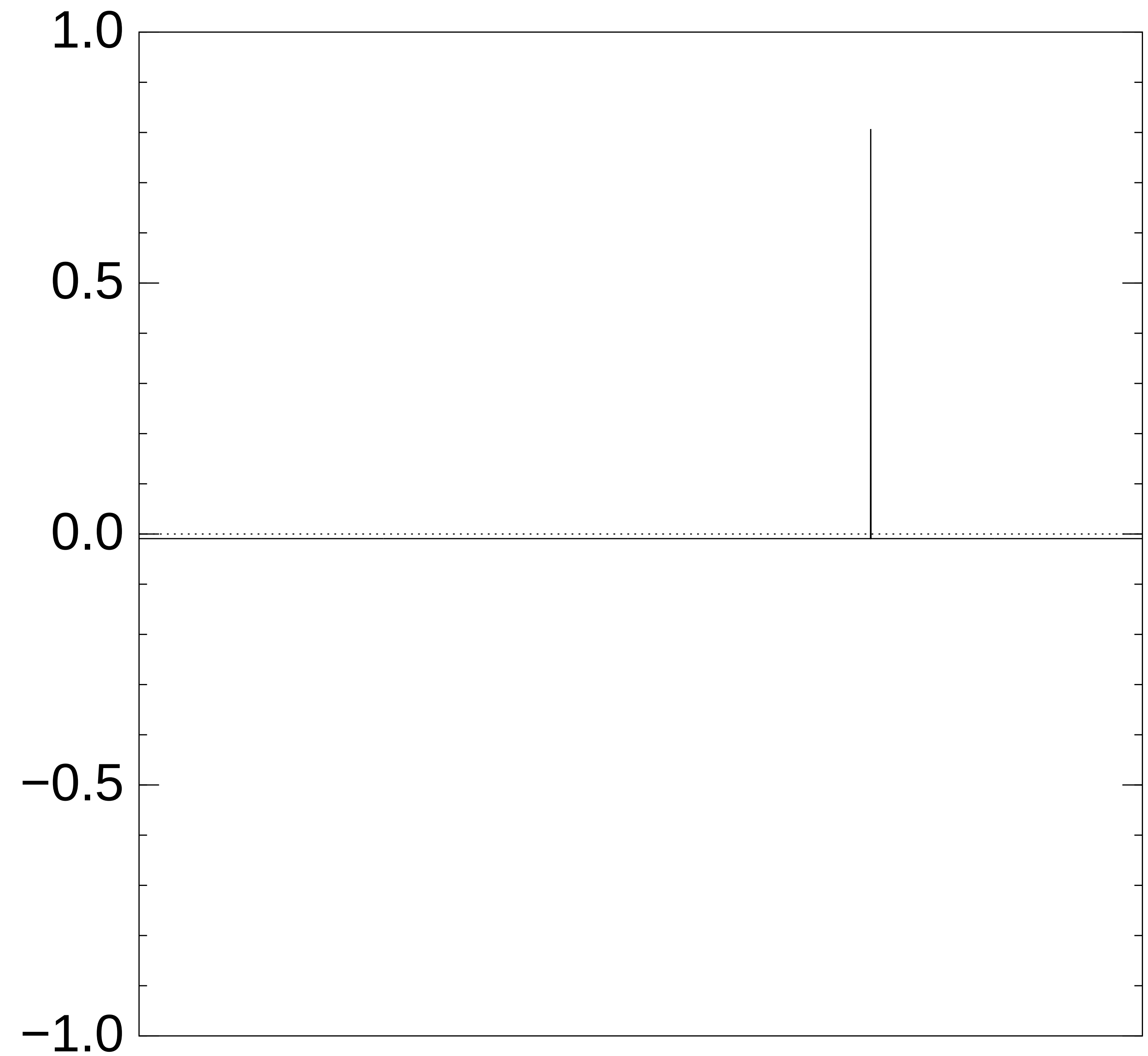
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $70 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

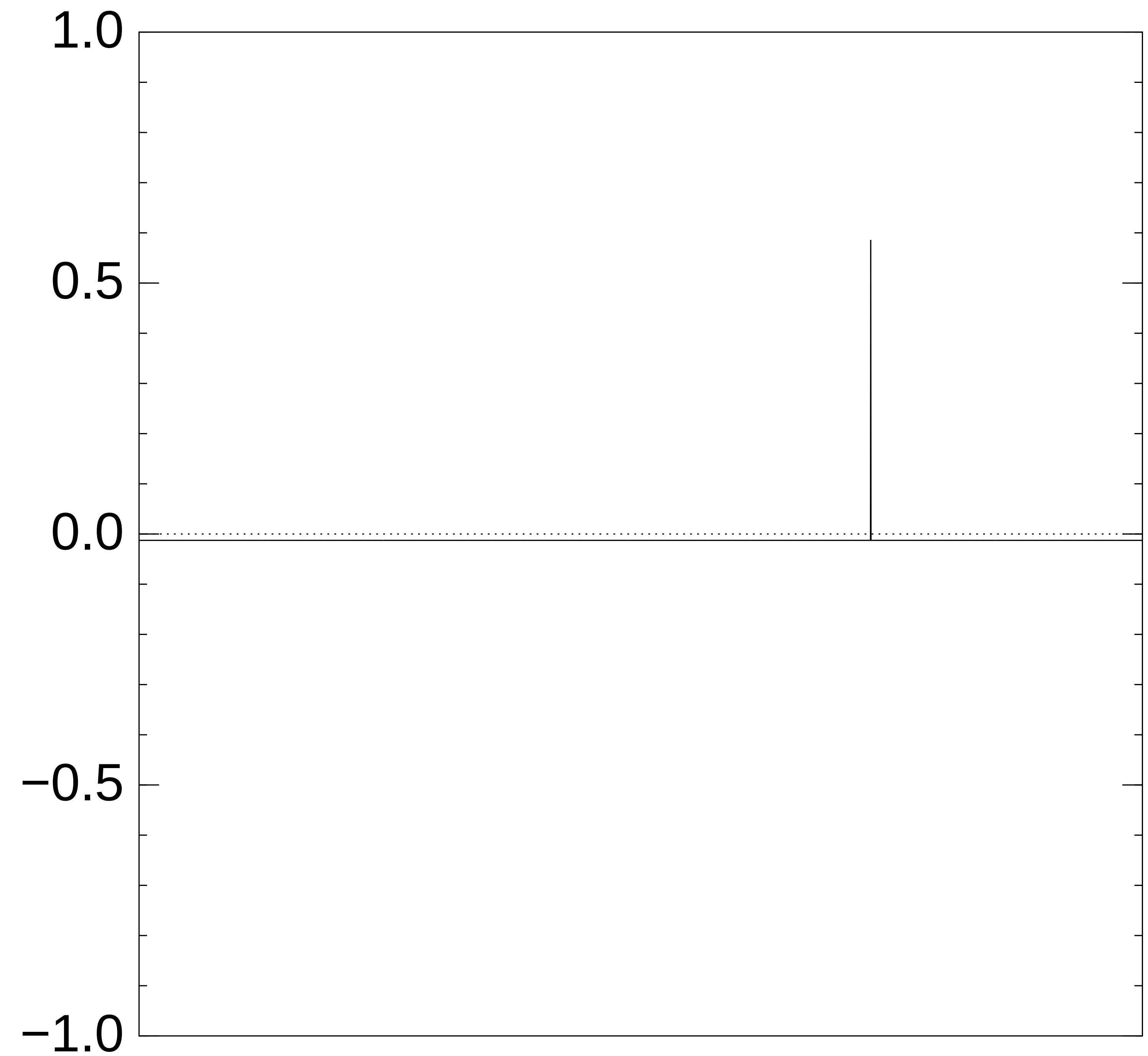
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $80 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

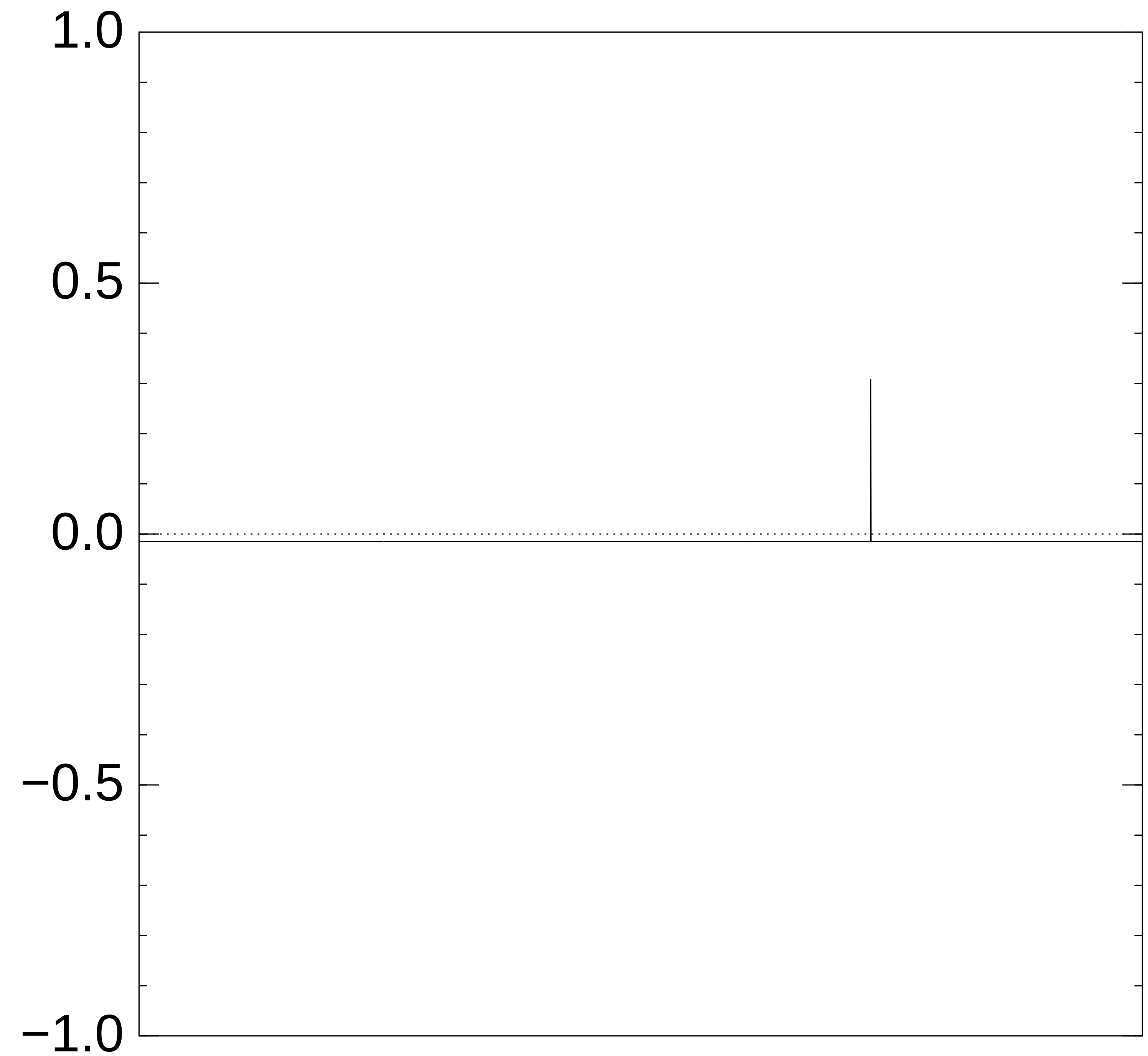
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $90 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

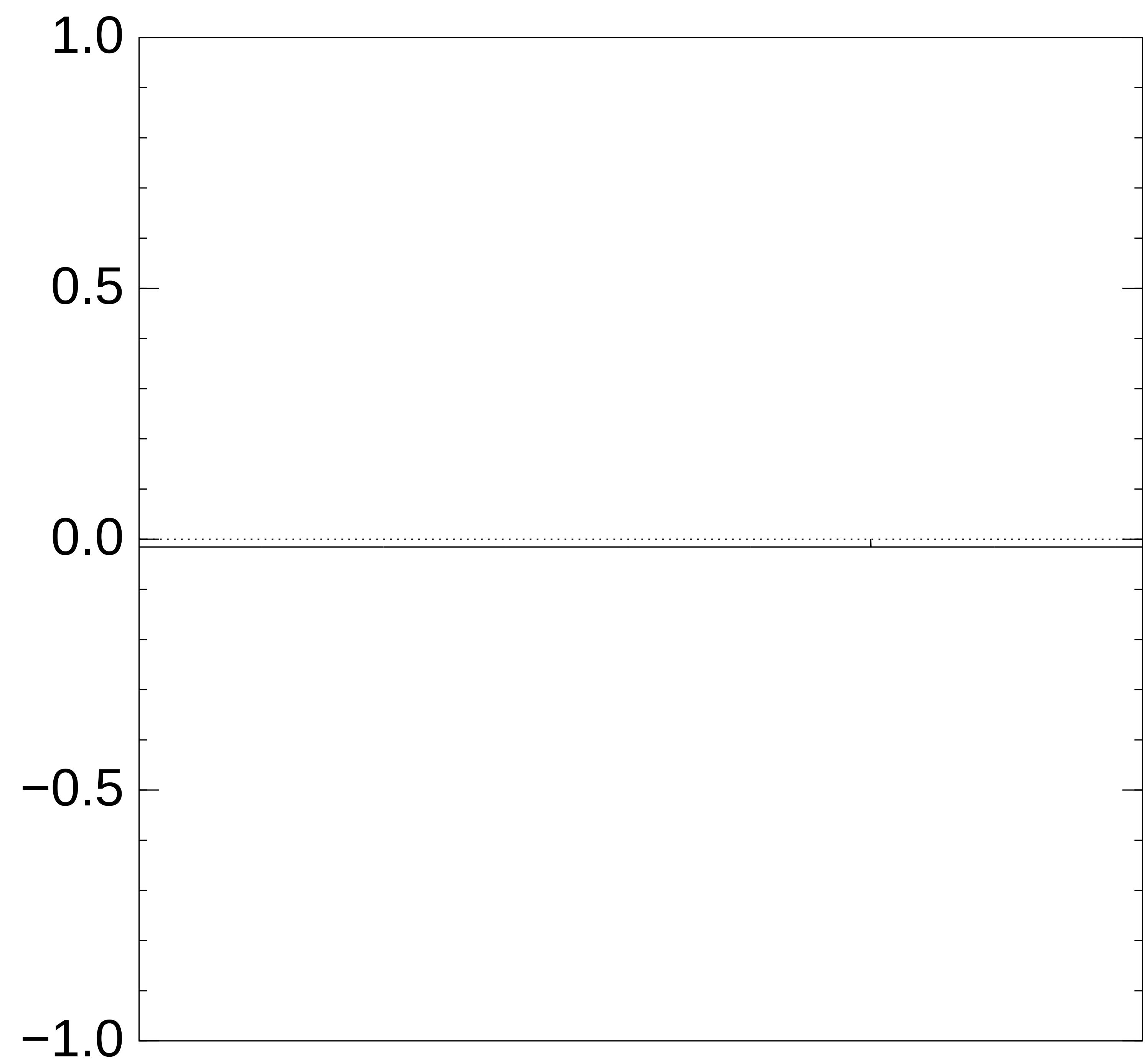
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

Repeat Step 1 + Step 2 about $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this finds s.

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $100 \times (\text{Step } 1 + \text{Step } 2)$:



Very bad stopping point.

om uniform superposition
n-bit strings q.

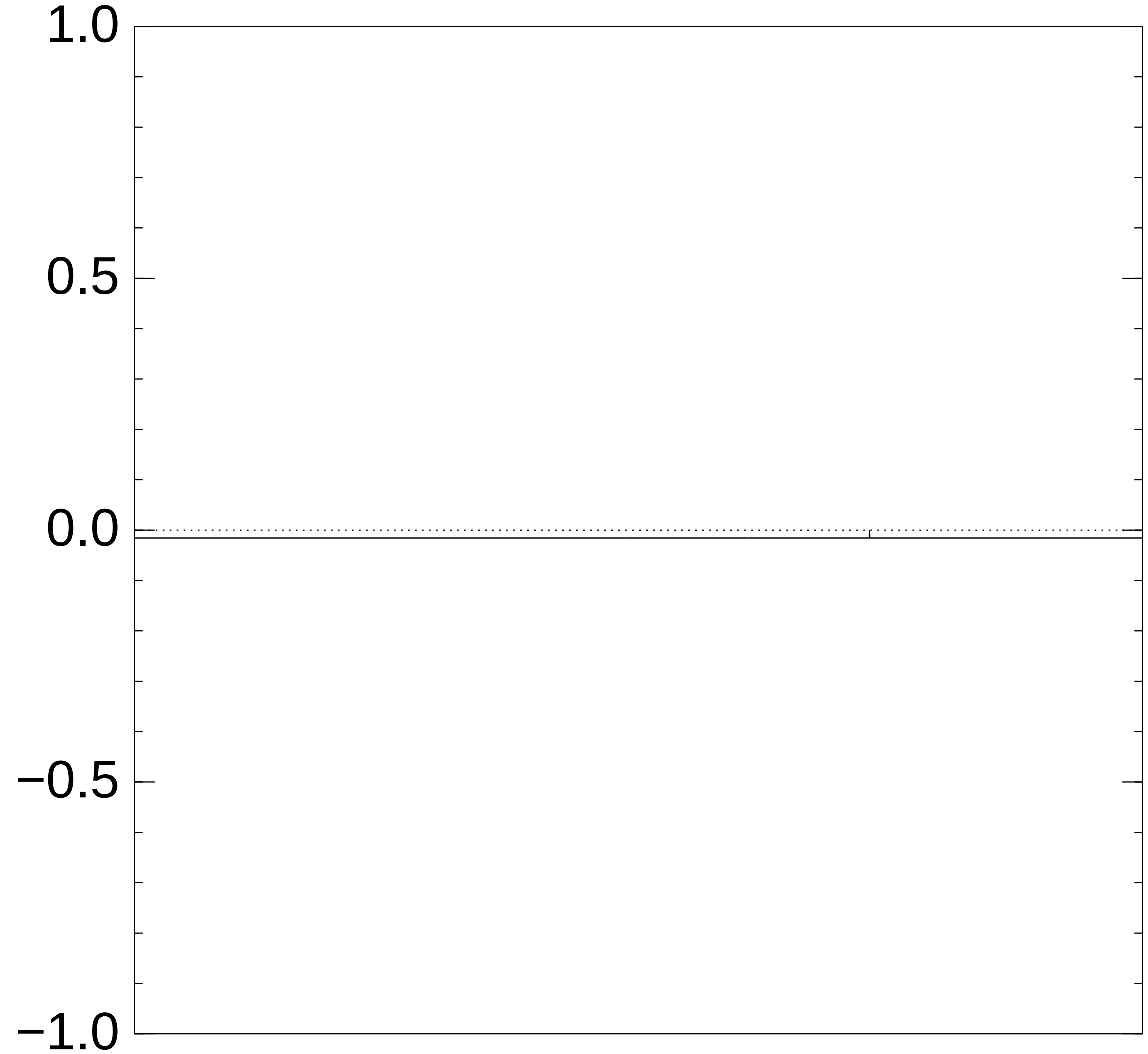
Set $a \leftarrow b$ where
a_q if $f(q) = 0$,
otherwise.
fast.

"Grover diffusion".
a around its average.
also fast.

Step 1 + Step 2
$58 \cdot 2^{0.5n}$ times.

the n qubits.
gh probability this finds s.

Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $100 \times$ (Step $1 +$ Step 2):



Very bad stopping point.

$q \mapsto a_q$
by a vec
(with fix
(1) a_q fo
(2) a_q fo

Step 1 +
act linea

Easily co
and pow
to under
of state
$\Rightarrow$ Proba
after $\approx$(

n superposition
gs q.

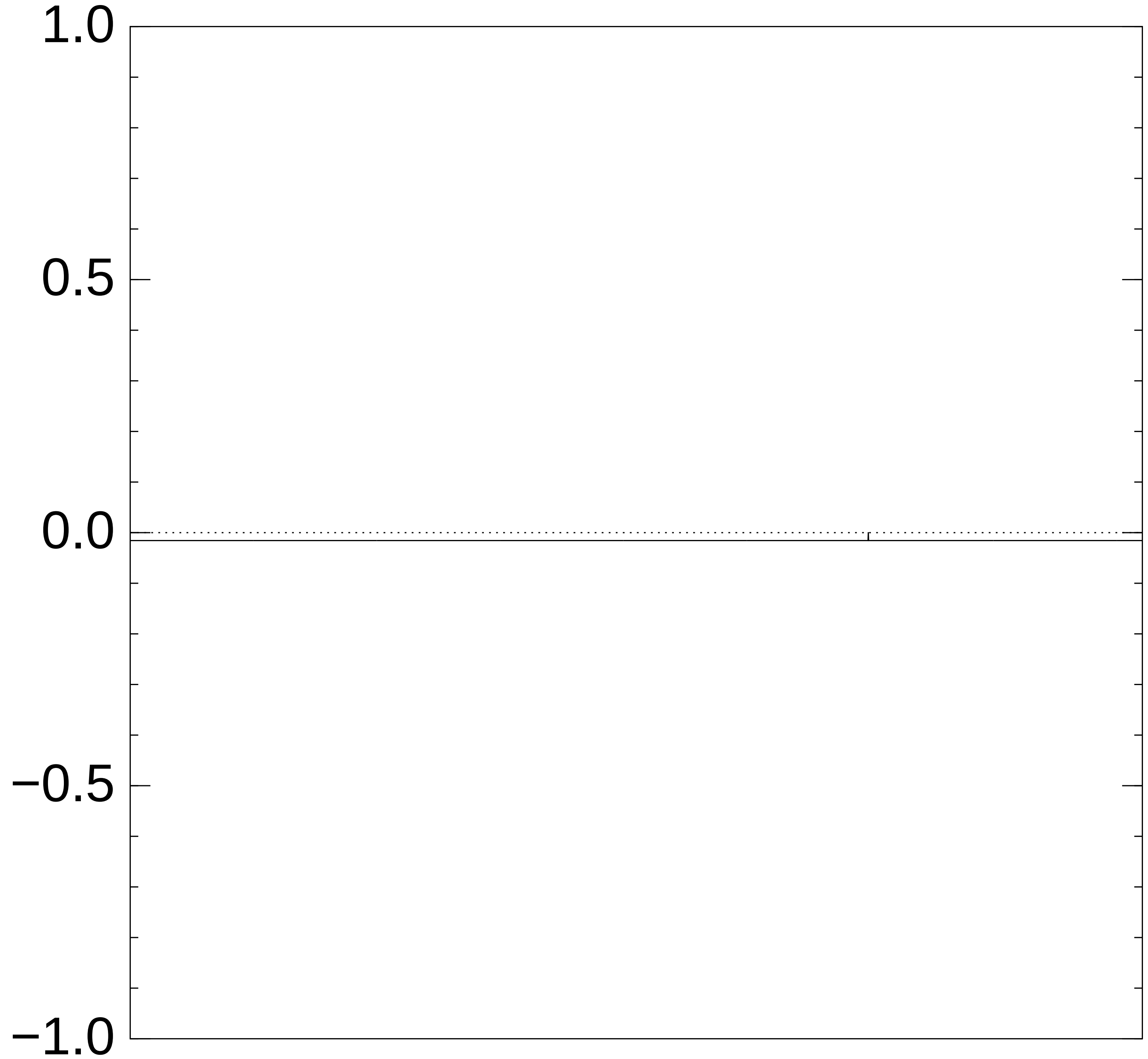
o where
$= 0$,
.

iffusion".
ts average.

Step 2
times.

bits.
lity this finds s.

Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $100 \times (\text{Step } 1 + \text{Step } 2)$:



Very bad stopping point.

$q \mapsto a_q$ is complet
by a vector of two
(with fixed multipl
(1) a_q for roots q;
(2) a_q for non-roo

Step 1 + Step 2
act linearly on this

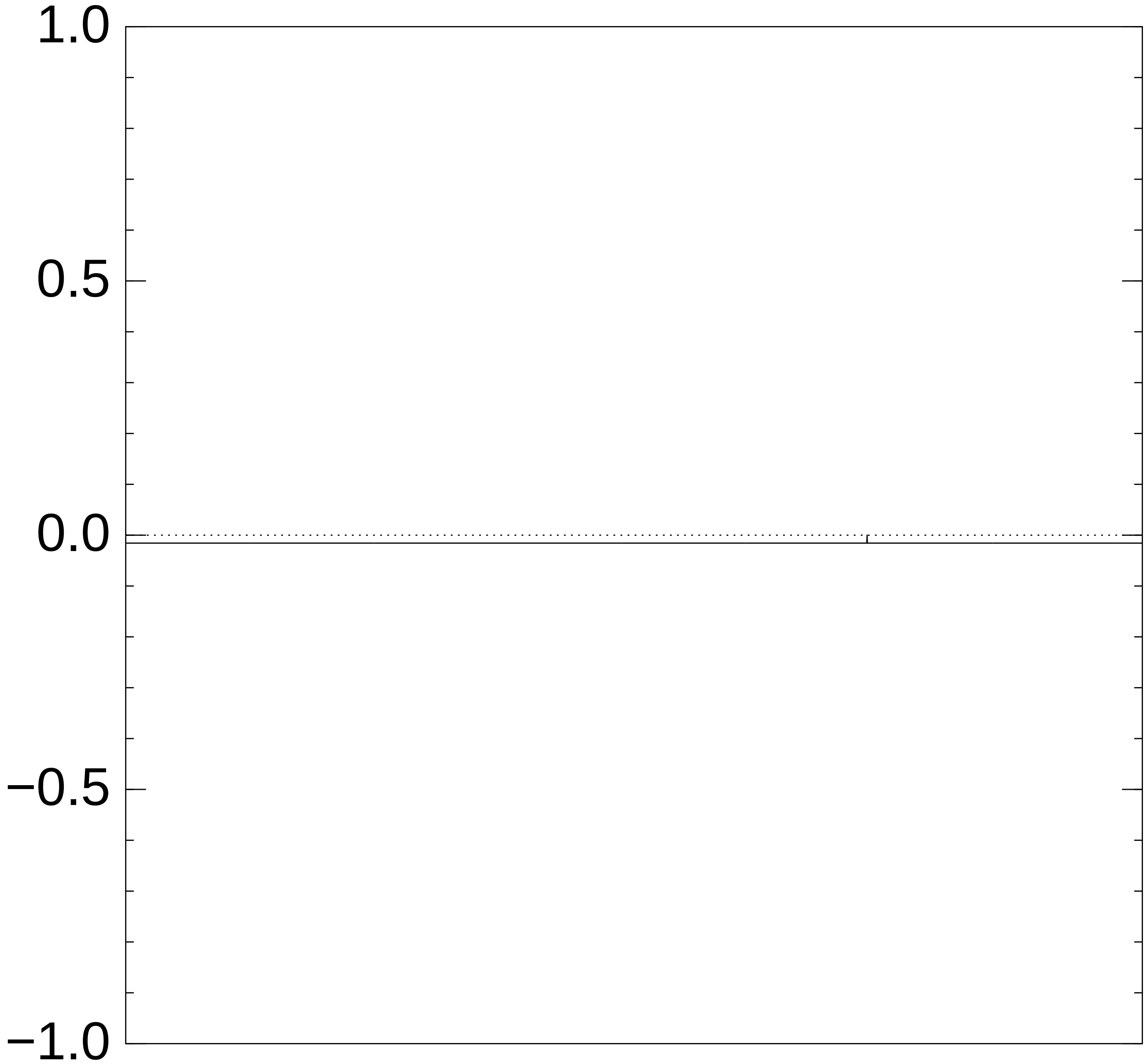
Easily compute eig
and powers of this
to understand evo
of state of Grover'
$\Rightarrow$ Probability is $\approx$
after $\approx(\pi/4)2^{0.5n}$

sition

.

nds s.

Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $100 \times (\text{Step 1} + \text{Step 2})$:



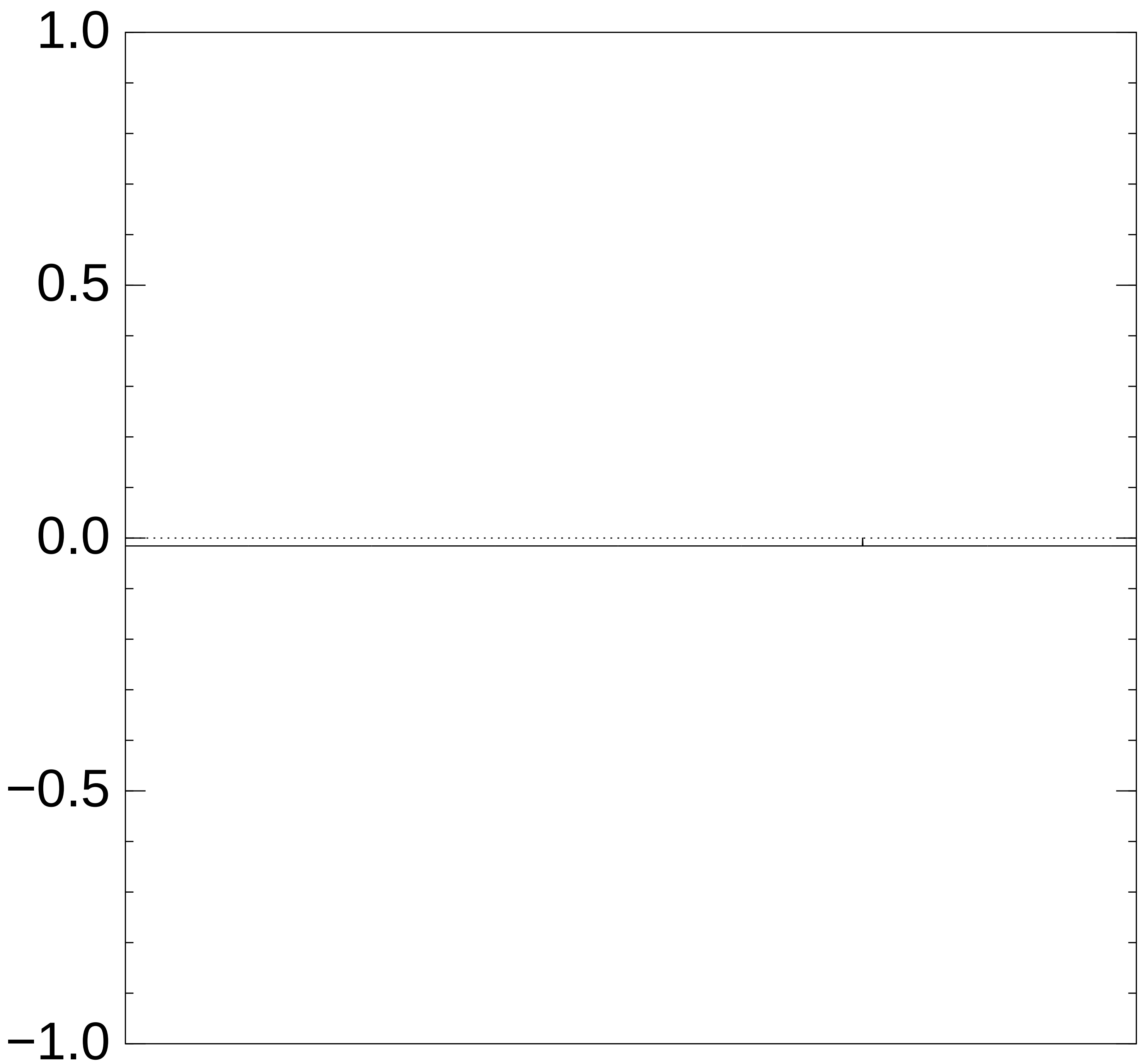
Very bad stopping point.

$q \mapsto a_q$ is completely describ
by a vector of two numbers
(with fixed multiplicities):
(1) a_q for roots q;
(2) a_q for non-roots q.

Step 1 + Step 2
act linearly on this vector.

Easily compute eigenvalues
and powers of this linear ma
to understand evolution
of state of Grover's algorithr
$\Rightarrow$ Probability is ≈ 1
after $\approx(\pi/4)2^{0.5n}$ iterations

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $100 \times$ (Step 1 + Step 2):



Very bad stopping point.

$q \mapsto a_q$ is completely described by a vector of two numbers (with fixed multiplicities):
(1) a_q for roots q;
(2) a_q for non-roots q.

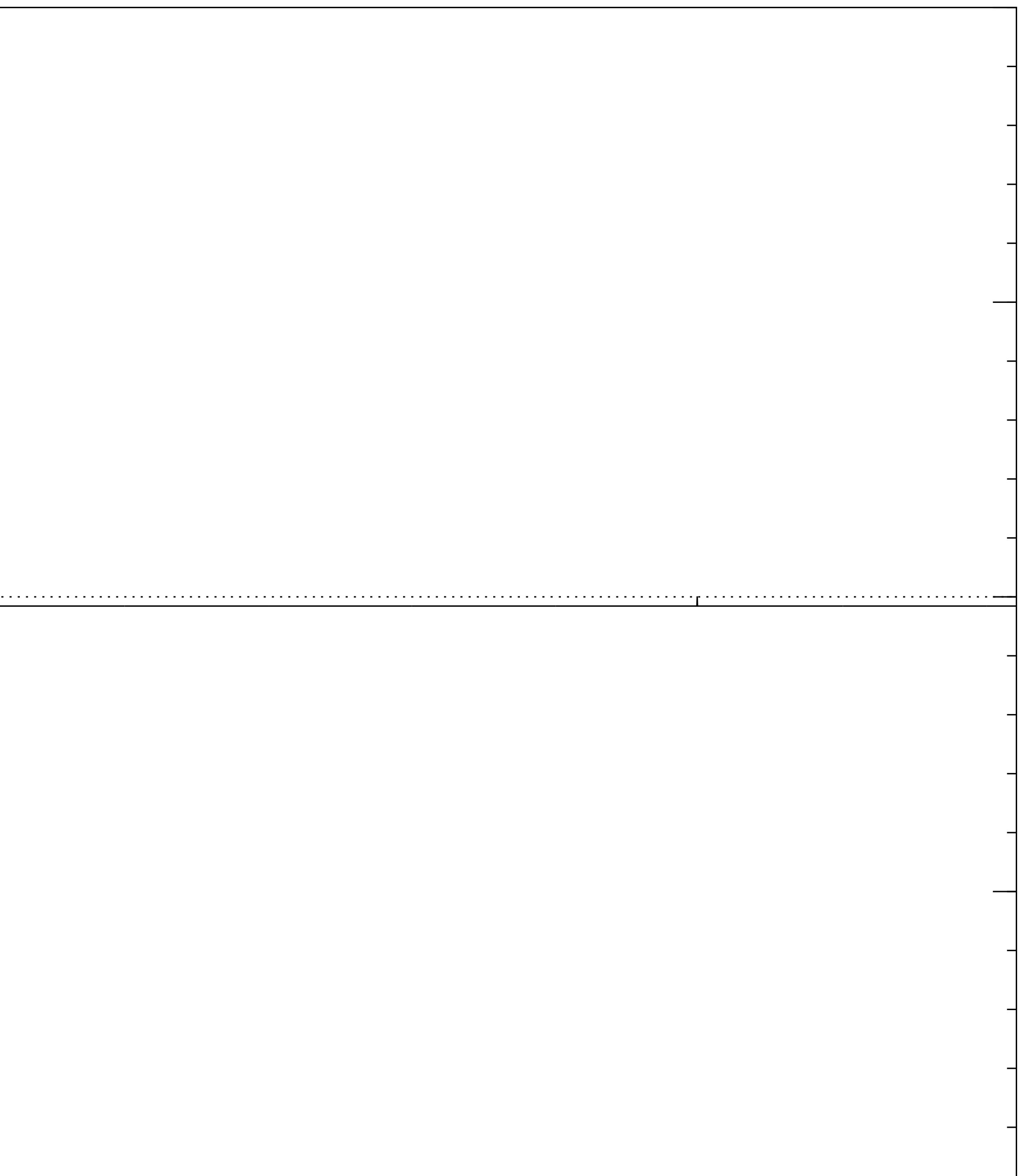
Step 1 + Step 2 act linearly on this vector.

Easily compute eigenvalues and powers of this linear map to understand evolution of state of Grover's algorithm.
⇒ Probability is ≈1 after $\approx(\pi/4)2^{0.5n}$ iterations.

zed graph of $q \mapsto a_q$

xample with $n = 12$

0 × (Step 1 + Step 2):



d stopping point.

$q \mapsto a_q$ is completely described by a vector of two numbers (with fixed multiplicities):

(1) a_q for roots q;

(2) a_q for non-roots q.

Step 1 + Step 2 act linearly on this vector.

Easily compute eigenvalues and powers of this linear map to understand evolution of state of Grover's algorithm.

⇒ Probability is ≈1 after $\approx(\pi/4)2^{0.5n}$ iterations.

Notes o

Textboo

Proof o

New

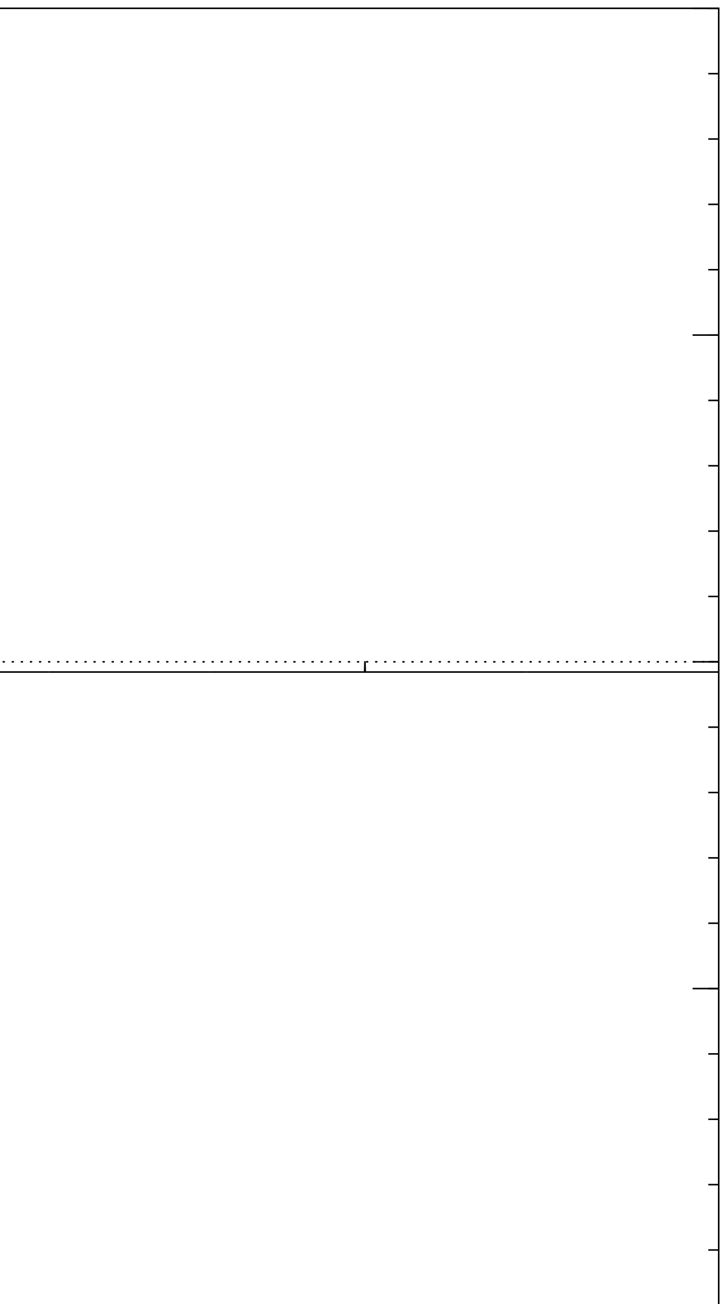
Proof

Mislead

that bes

best *pro*

of $q \mapsto a_q$
th $n = 12$
$1 + \text{Step } 2$):



point.

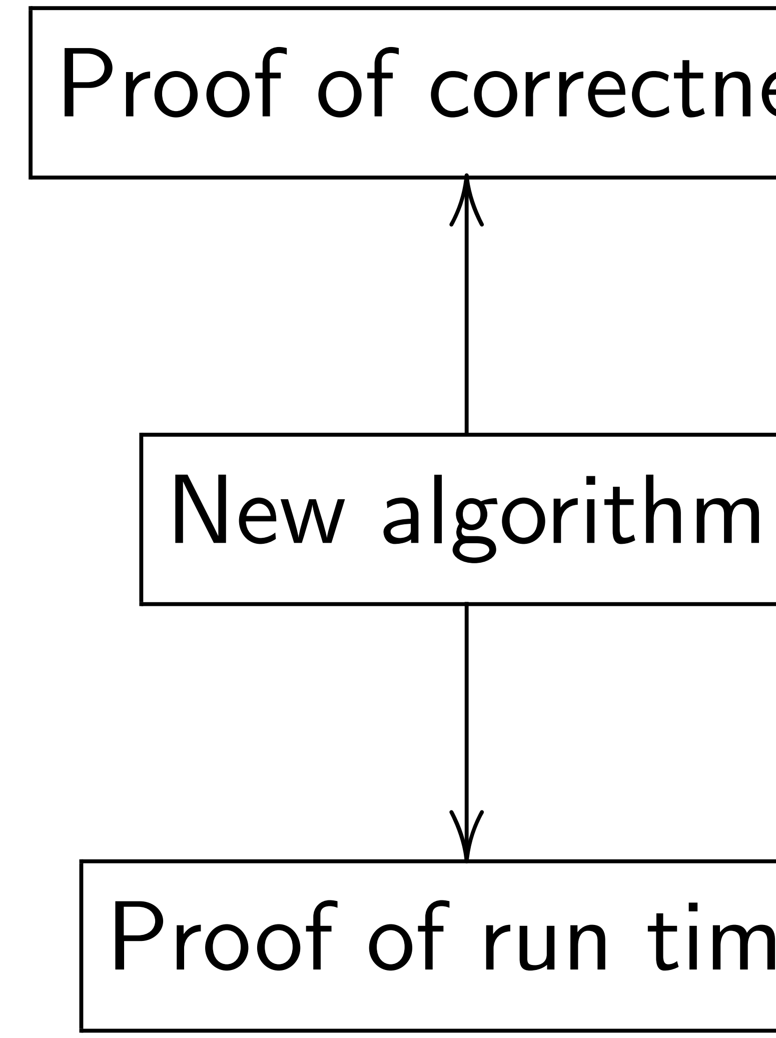
$q \mapsto a_q$ is completely described
by a vector of two numbers
(with fixed multiplicities):
(1) a_q for roots q;
(2) a_q for non-roots q.

Step 1 + Step 2
act linearly on this vector.

Easily compute eigenvalues
and powers of this linear map
to understand evolution
of state of Grover's algorithm.
$\Rightarrow$ Probability is ≈ 1
after $\approx (\pi/4) 2^{0.5n}$ iterations.

<u>Notes on provabili</u>

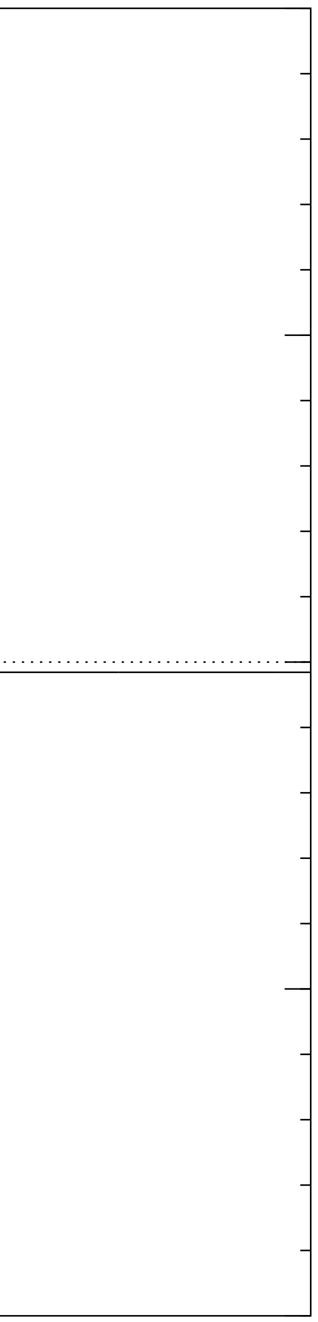
Textbook algorith



Mislead students i
that best algorith
best *proven* algori

q

2):



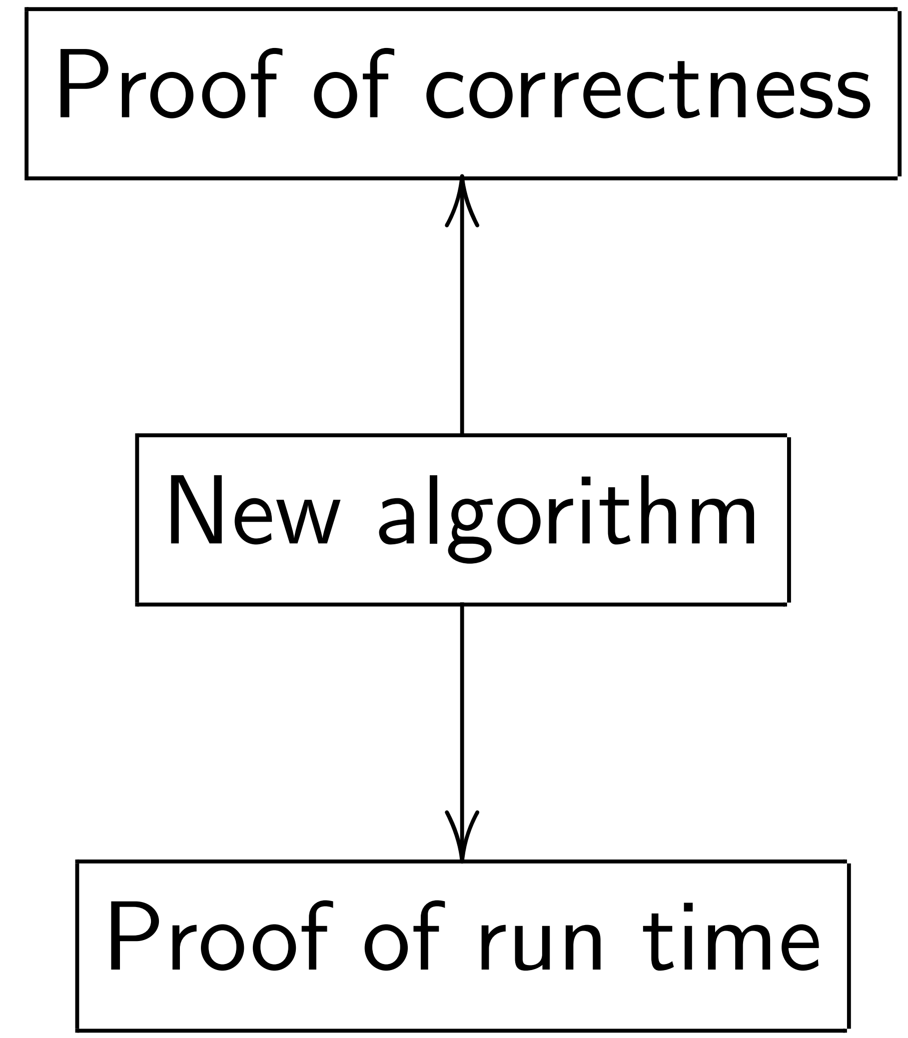
$q \mapsto a_q$ is completely described by a vector of two numbers (with fixed multiplicities):
(1) a_q for roots q;
(2) a_q for non-roots q.

Step 1 + Step 2 act linearly on this vector.

Easily compute eigenvalues and powers of this linear map to understand evolution of state of Grover's algorithm.
$\Rightarrow$ Probability is ≈ 1 after $\approx(\pi/4)2^{0.5n}$ iterations.

Notes on provability

Textbook algorithm analysis



Mislead students into thinkin
that best algorithm =
best *proven* algorithm.

$q \mapsto a_q$ is completely described by a vector of two numbers (with fixed multiplicities):

(1) a_q for roots q;

(2) a_q for non-roots q.

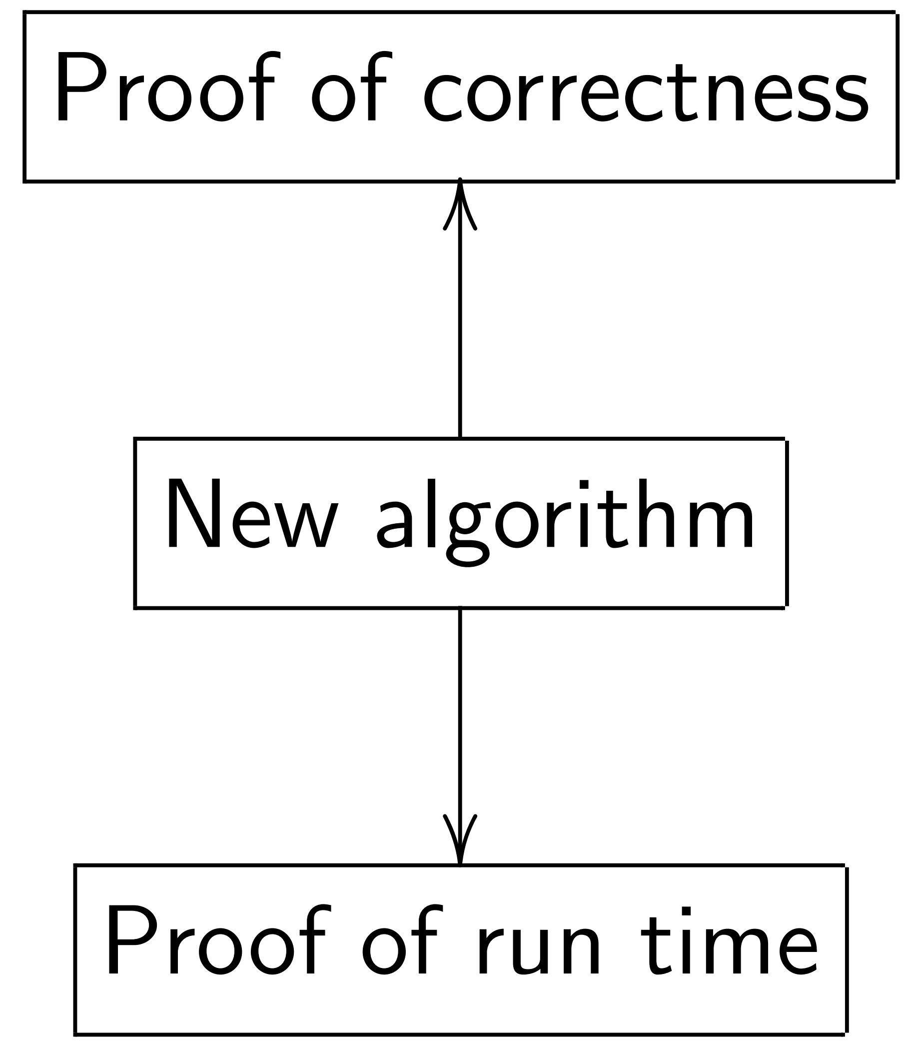
Step 1 + Step 2 act linearly on this vector.

Easily compute eigenvalues and powers of this linear map to understand evolution of state of Grover's algorithm.

$\Rightarrow$ Probability is ≈ 1 after $\approx (\pi/4) 2^{0.5n}$ iterations.

Notes on provability

Textbook algorithm analysis:



Mislead students into thinking that best algorithm = best *proven* algorithm.

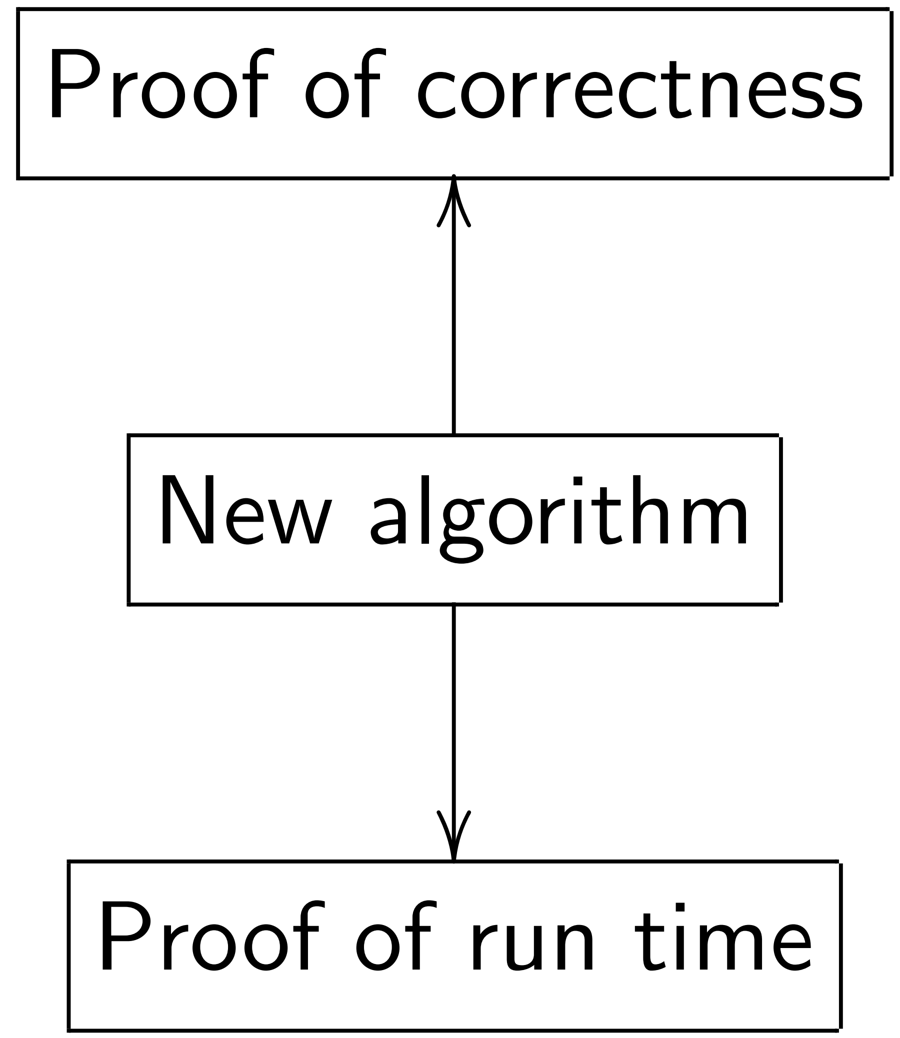
is completely described
tor of two numbers
ked multiplicities):
or roots q;
or non-roots q.

+ Step 2
arly on this vector.

ompute eigenvalues
vers of this linear map
rstand evolution
of Grover's algorithm.
ability is ≈ 1
$\pi/4)2^{0.5n}$ iterations.

Notes on provability

Textbook algorithm analysis:



Mislead students into thinking that best algorithm = best *proven* algorithm.

Reality:
cryptana
are almo

tely described
numbers
licities):

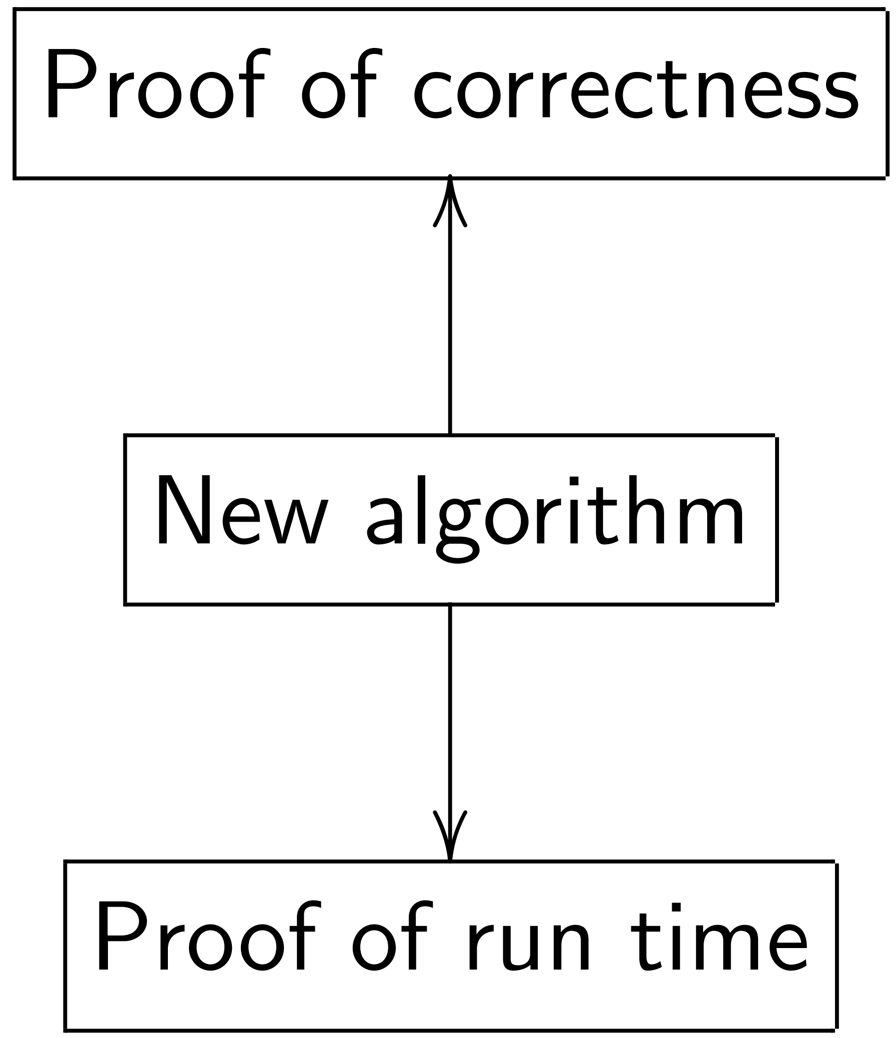
ts q.

s vector.

genvalues
s linear map
lution
s algorithm.
≈ 1
iterations.

Notes on provability

Textbook algorithm analysis:



Mislead students into thinking that best algorithm = best *proven* algorithm.

Reality: state-of-th
cryptanalytic algor
are almost never p

ped

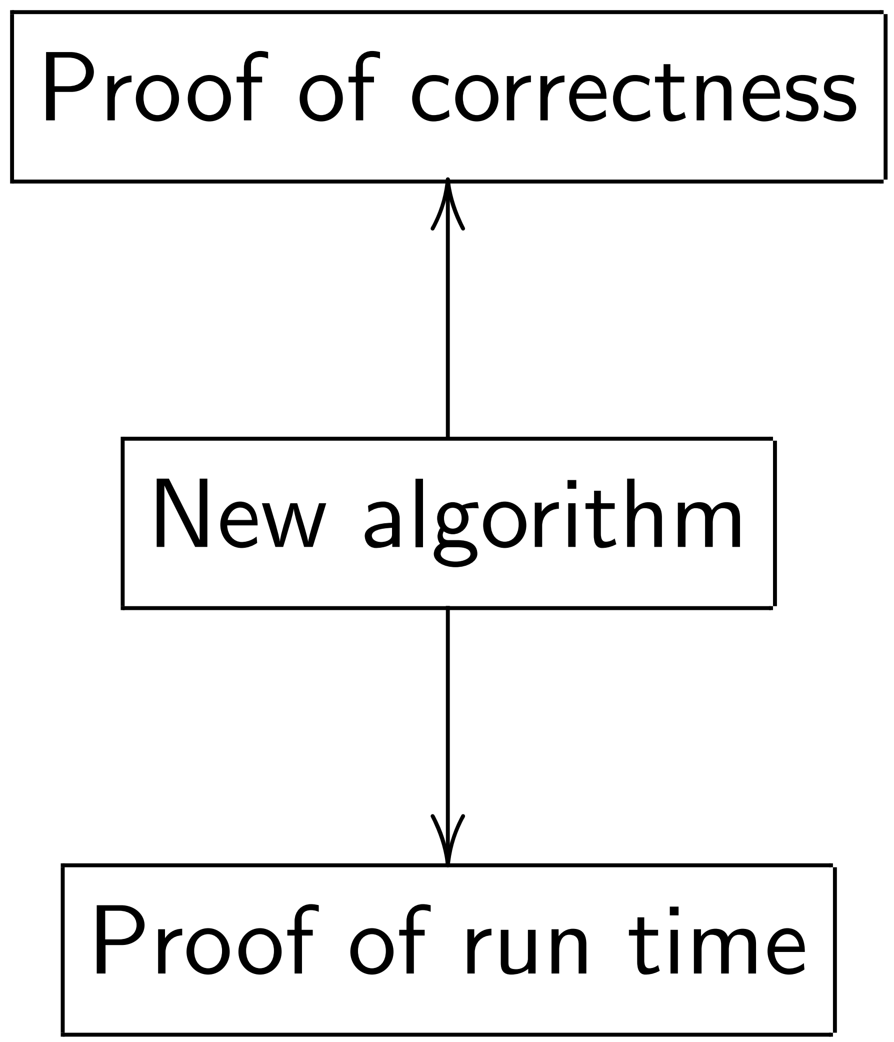
p

m.

.

Notes on provability

Textbook algorithm analysis:

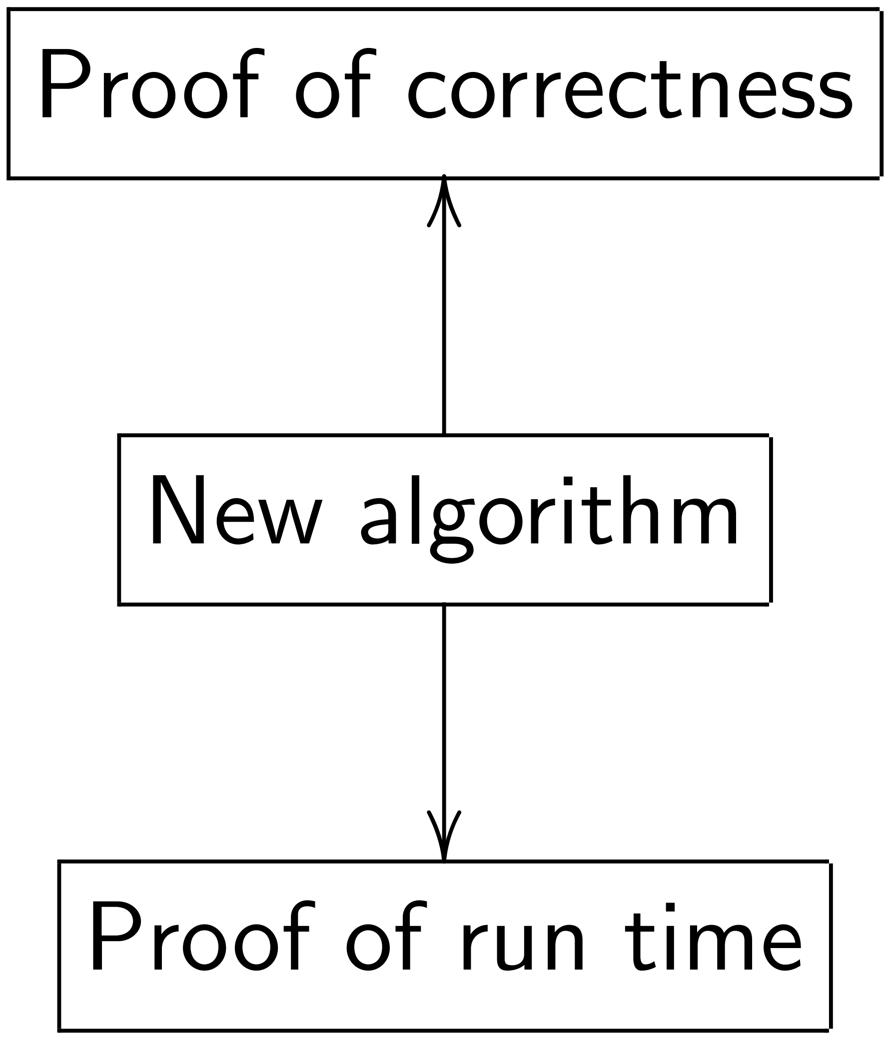


Mislead students into thinking that best algorithm = best *proven* algorithm.

Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Notes on provability

Textbook algorithm analysis:

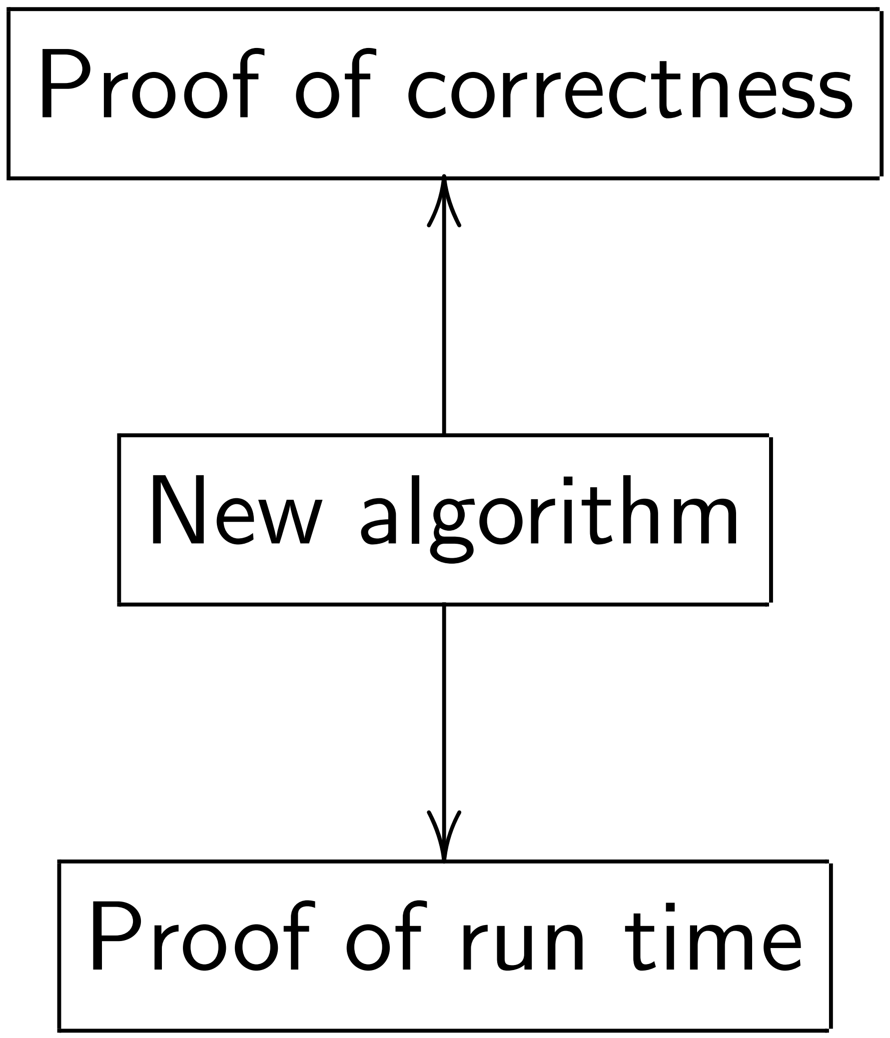


Mislead students into thinking that best algorithm = best *proven* algorithm.

Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Notes on provability

Textbook algorithm analysis:



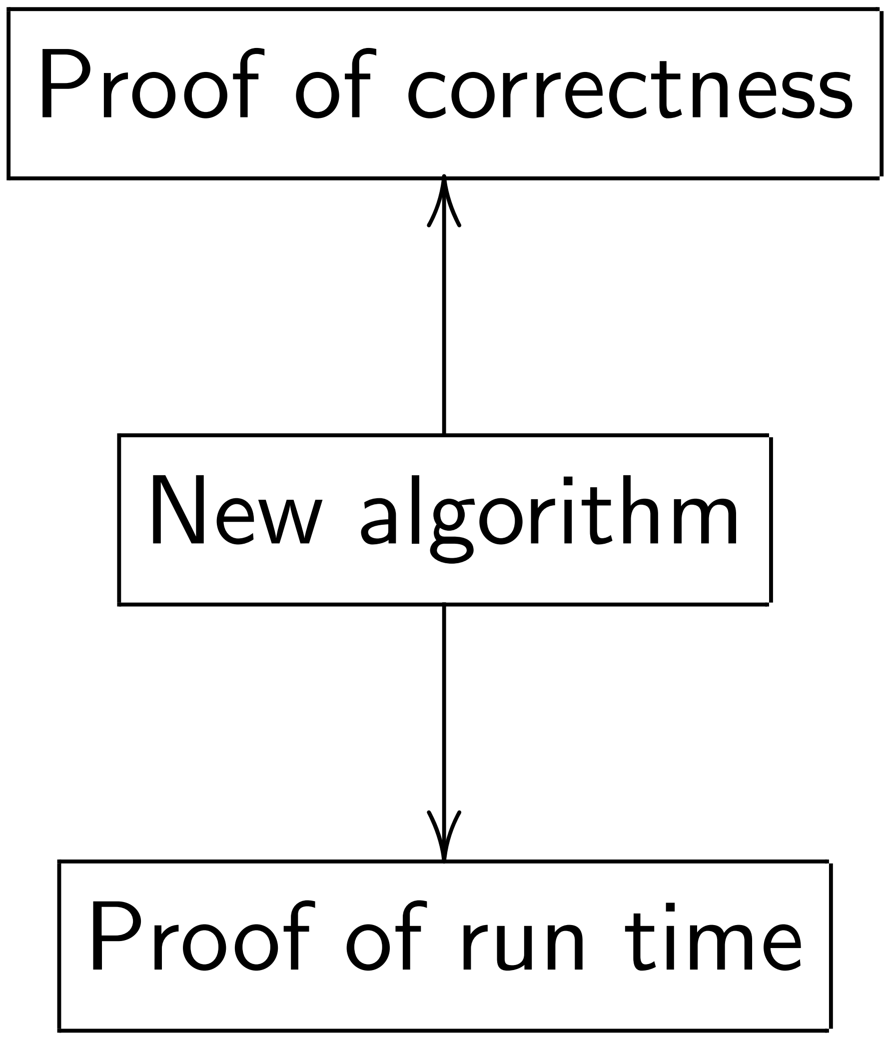
Mislead students into thinking that best algorithm = best *proven* algorithm.

Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Ignorant response: "Work harder, find proofs!"

Notes on provability

Textbook algorithm analysis:



Mislead students into thinking that best algorithm = best *proven* algorithm.

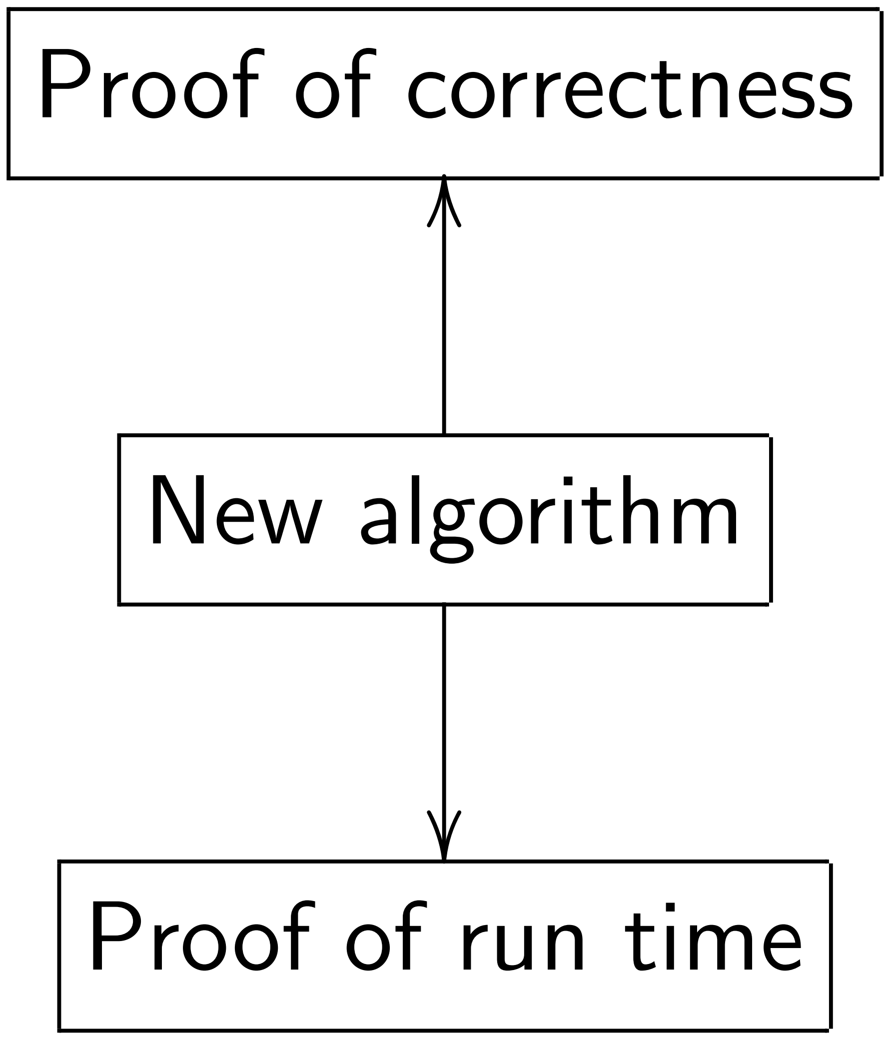
Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Ignorant response: "Work harder, find proofs!"

Consensus of the experts: proofs probably do not *exist* for most of these algorithms. So demanding proofs is silly.

Notes on provability

Textbook algorithm analysis:



Mislead students into thinking that best algorithm = best *proven* algorithm.

Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

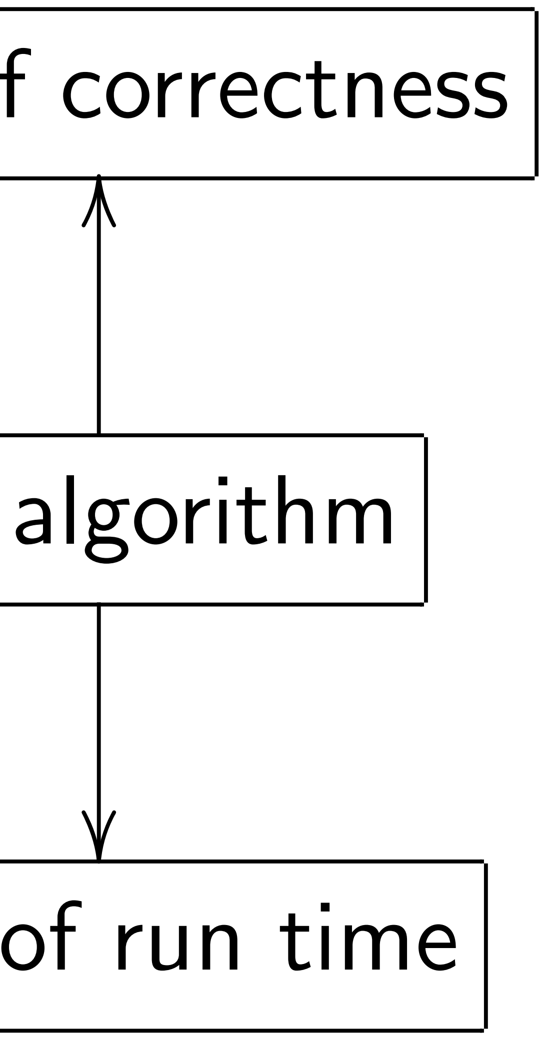
Ignorant response: "Work harder, find proofs!"

Consensus of the experts: proofs probably do not *exist* for most of these algorithms. So demanding proofs is silly.

Without proofs, how do we analyze correctness+speed? Answer: Real algorithm analysis relies critically on heuristics and **computer experiments**.

n provability

k algorithm analysis:



students into thinking
t algorithm =
ven algorithm.

Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Ignorant response: “Work harder, find proofs!”

Consensus of the experts: proofs probably do not *exist* for most of these algorithms. So demanding proofs is silly.

Without proofs, how do we analyze correctness+speed? Answer: Real algorithm analysis relies critically on heuristics and **computer experiments**.

What ab
Want to
quantum
to figure
against

ty

m analysis:

ess

e

nto thinking

m =

thm.

Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Ignorant response: "Work harder, find proofs!"

Consensus of the experts: proofs probably do not *exist* for most of these algorithms. So demanding proofs is silly.

Without proofs, how do we analyze correctness+speed? Answer: Real algorithm analysis relies critically on heuristics and **computer experiments**.

What about quant

Want to analyze,

quantum algorithm

to figure out safe

against *future* qua

:

ng

Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Ignorant response: "Work harder, find proofs!"

Consensus of the experts: proofs probably do not *exist* for most of these algorithms. So demanding proofs is silly.

Without proofs, how do we analyze correctness+speed? Answer: Real algorithm analysis relies critically on heuristics and **computer experiments**.

What about quantum algori
Want to analyze, optimize
quantum algorithms *today*
to figure out safe crypto
against *future* quantum atta

Reality: state-of-the-art
cryptanalytic algorithms
are almost never proven.

Ignorant response:
"Work harder, find proofs!"

Consensus of the experts:
proofs probably do not *exist*
for most of these algorithms.
So demanding proofs is silly.

Without proofs, how do we
analyze correctness+speed?
Answer: Real algorithm analysis
relies critically on heuristics and
computer experiments.

What about quantum algorithms?
Want to analyze, optimize
quantum algorithms *today*
to figure out safe crypto
against *future* quantum attack.

Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Ignorant response: "Work harder, find proofs!"

Consensus of the experts: proofs probably do not *exist* for most of these algorithms. So demanding proofs is silly.

Without proofs, how do we analyze correctness+speed? Answer: Real algorithm analysis relies critically on heuristics and **computer experiments**.

What about quantum algorithms? Want to analyze, optimize quantum algorithms *today* to figure out safe crypto against *future* quantum attack.

1. Simulate *tiny* q. computer?

$\Rightarrow$ Huge extrapolation errors.

Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Ignorant response: "Work harder, find proofs!"

Consensus of the experts: proofs probably do not *exist* for most of these algorithms. So demanding proofs is silly.

Without proofs, how do we analyze correctness+speed? Answer: Real algorithm analysis relies critically on heuristics and **computer experiments**.

What about quantum algorithms? Want to analyze, optimize quantum algorithms *today* to figure out safe crypto against *future* quantum attack.

1. Simulate *tiny* q. computer?
$\Rightarrow$ Huge extrapolation errors.

2. Faster algorithm-specific simulation? Yes, sometimes.

Reality: state-of-the-art cryptanalytic algorithms are almost never proven.

Ignorant response: "Work harder, find proofs!"

Consensus of the experts: proofs probably do not *exist* for most of these algorithms. So demanding proofs is silly.

Without proofs, how do we analyze correctness+speed? Answer: Real algorithm analysis relies critically on heuristics and **computer experiments**.

What about quantum algorithms? Want to analyze, optimize quantum algorithms *today* to figure out safe crypto against *future* quantum attack.

1. Simulate *tiny* q. computer? ⇒ Huge extrapolation errors.

2. Faster algorithm-specific simulation? Yes, sometimes.

3. Fast **trapdoor simulation.** Simulator (like prover) knows more than the algorithm does. Tung Chou has implemented this, found errors in two publications.

state-of-the-art
alytic algorithms
ost never proven.

response:
harder, find proofs!"

us of the experts:
robably do not *exist*
of these algorithms.
anding proofs is silly.

proofs, how do we
correctness+speed?
Real algorithm analysis
tically on heuristics and
er experiments.

What about quantum algorithms?
Want to analyze, optimize
quantum algorithms *today*
to figure out safe crypto
against *future* quantum attack.

1. Simulate *tiny* q. computer?
$\Rightarrow$ Huge extrapolation errors.

2. Faster algorithm-specific
simulation? Yes, sometimes.

3. Fast **trapdoor simulation.**
Simulator (like prover) knows
more than the algorithm does.
Tung Chou has implemented this,
found errors in two publications.

Post-qua

Grover's
128-bit
2^{64} quar

he-art
rithms
roven.

d proofs!"

experts:
o not *exist*
algorithms.
ofs is silly.

ow do we
s+speed?
rithm analysis
heuristics and
ments.

What about quantum algorithms?
Want to analyze, optimize
quantum algorithms *today*
to figure out safe crypto
against *future* quantum attack.

1. Simulate *tiny* q. computer?
$\Rightarrow$ Huge extrapolation errors.

2. Faster algorithm-specific
simulation? Yes, sometimes.

3. Fast **trapdoor simulation.**
Simulator (like prover) knows
more than the algorithm does.
Tung Chou has implemented this,
found errors in two publications.

Post-quantum cry

Grover's algorithm
128-bit AES key u
2^{64} quantum AES

.

.

.

lysis
and

What about quantum algorithms?
Want to analyze, optimize quantum algorithms *today* to figure out safe crypto against *future* quantum attack.

1. Simulate *tiny* q. computer?
⇒ Huge extrapolation errors.

2. Faster algorithm-specific simulation? Yes, sometimes.

3. Fast **trapdoor simulation.**
Simulator (like prover) knows more than the algorithm does.
Tung Chou has implemented this, found errors in two publications.

<u>Post-quantum cryptography</u>

Grover's algorithm finds 128-bit AES key using 2^{64} quantum AES evaluatio

What about quantum algorithms?
Want to analyze, optimize quantum algorithms *today* to figure out safe crypto against *future* quantum attack.

1. Simulate *tiny* q. computer?
$\Rightarrow$ Huge extrapolation errors.

2. Faster algorithm-specific simulation? Yes, sometimes.

3. Fast **trapdoor simulation.**
Simulator (like prover) knows more than the algorithm does.
Tung Chou has implemented this, found errors in two publications.

Post-quantum cryptography

Grover's algorithm finds 128-bit AES key using 2^{64} quantum AES evaluations.

What about quantum algorithms?
Want to analyze, optimize quantum algorithms *today* to figure out safe crypto against *future* quantum attack.

1. Simulate *tiny* q. computer?
⇒ Huge extrapolation errors.

2. Faster algorithm-specific simulation? Yes, sometimes.

3. Fast **trapdoor simulation.** Simulator (like prover) knows more than the algorithm does. Tung Chou has implemented this, found errors in two publications.

Post-quantum cryptography

Grover's algorithm finds 128-bit AES key using 2^{64} quantum AES evaluations.

Sensible risk management: Assume that this is feasible— or will be feasible in, e.g., 2025.

What about quantum algorithms?
Want to analyze, optimize quantum algorithms *today* to figure out safe crypto against *future* quantum attack.

1. Simulate *tiny* q. computer?
⇒ Huge extrapolation errors.

2. Faster algorithm-specific simulation? Yes, sometimes.

3. Fast **trapdoor simulation.**
Simulator (like prover) knows more than the algorithm does.
Tung Chou has implemented this, found errors in two publications.

Post-quantum cryptography

Grover's algorithm finds 128-bit AES key using 2^{64} quantum AES evaluations.

Sensible risk management:
Assume that this is feasible—
or will be feasible in, e.g., 2025.
"AES-128 is dead."

What about quantum algorithms? Want to analyze, optimize quantum algorithms *today* to figure out safe crypto against *future* quantum attack.

1. Simulate *tiny* q. computer?
⇒ Huge extrapolation errors.

2. Faster algorithm-specific simulation? Yes, sometimes.

3. Fast **trapdoor simulation.** Simulator (like prover) knows more than the algorithm does. Tung Chou has implemented this, found errors in two publications.

Post-quantum cryptography

Grover's algorithm finds 128-bit AES key using 2^{64} quantum AES evaluations.

Sensible risk management: Assume that this is feasible— or will be feasible in, e.g., 2025. "AES-128 is dead."

Fix: Switch to AES-256.

What about quantum algorithms? Want to analyze, optimize quantum algorithms *today* to figure out safe crypto against *future* quantum attack.

1. Simulate *tiny* q. computer? ⇒ Huge extrapolation errors.

2. Faster algorithm-specific simulation? Yes, sometimes.

3. Fast **trapdoor simulation.** Simulator (like prover) knows more than the algorithm does. Tung Chou has implemented this, found errors in two publications.

Post-quantum cryptography

Grover's algorithm finds 128-bit AES key using 2^{64} quantum AES evaluations.

Sensible risk management: Assume that this is feasible— or will be feasible in, e.g., 2025. "AES-128 is dead."

Fix: Switch to AES-256.

AES-256 has 14 rounds. Maybe 12 rounds are enough for 2^{128} post-quantum security? Maybe 10 rounds are enough?

bout quantum algorithms?
analyze, optimize
n algorithms *today*
e out safe crypto
future quantum attack.

late *tiny* q. computer?
e extrapolation errors.

er algorithm-specific
on? Yes, sometimes.

trapdoor simulation.
or (like prover) knows
an the algorithm does.
hou has implemented this,
rrors in two publications.

<u>Post-quantum cryptography</u>

Grover's algorithm finds
128-bit AES key using
2^{64} quantum AES evaluations.

Sensible risk management:
Assume that this is feasible—
or will be feasible in, e.g., 2025.
"AES-128 is dead."

Fix: Switch to AES-256.

AES-256 has 14 rounds.
Maybe 12 rounds are enough
for 2^{128} post-quantum security?
Maybe 10 rounds are enough?

Shor's a
(similar
factors R
finding p

Number
≈ numb
to comp

tum algorithms?
optimize
ns *today*
crypto
ntum attack.

q. computer?
tion errors.

m-specific
sometimes.

simulation.
over) knows
orithm does.
plemented this,
o publications.

Post-quantum cryptography

Grover’s algorithm finds
128-bit AES key using
2^{64} quantum AES evaluations.

Sensible risk management:
Assume that this is feasible—
or will be feasible in, e.g., 2025.
“AES-128 is dead.”

Fix: Switch to AES-256.

AES-256 has 14 rounds.
Maybe 12 rounds are enough
for 2^{128} post-quantum security?
Maybe 10 rounds are enough?

Shor’s algorithm
(similar to Simon’
factors RSA modu
finding period of

Number of qubit o
$\approx$ number of bit o
to compute 2^x mo

thms?

ack.

er?

s.

r

n.

ys

es.

d this,

ons.

Post-quantum cryptography

Grover's algorithm finds
128-bit AES key using
2^{64} quantum AES evaluations.

Sensible risk management:
Assume that this is feasible—
or will be feasible in, e.g., 2025.
"AES-128 is dead."

Fix: Switch to AES-256.

AES-256 has 14 rounds.
Maybe 12 rounds are enough
for 2^{128} post-quantum security?
Maybe 10 rounds are enough?

Shor's algorithm
(similar to Simon's algorithn
factors RSA modulus N by
finding period of $x \mapsto 2^x$ mo

Number of qubit operations
$\approx$ number of bit operations
to compute $2^x \bmod N$.

Post-quantum cryptography

Grover's algorithm finds
128-bit AES key using
2^{64} quantum AES evaluations.

Sensible risk management:
Assume that this is feasible—
or will be feasible in, e.g., 2025.
"AES-128 is dead."

Fix: Switch to AES-256.

AES-256 has 14 rounds.
Maybe 12 rounds are enough
for 2^{128} post-quantum security?
Maybe 10 rounds are enough?

Shor's algorithm
(similar to Simon's algorithm)
factors RSA modulus N by
finding period of $x \mapsto 2^x \bmod N$.

Number of qubit operations
$\approx$ number of bit operations
to compute $2^x \bmod N$.

Post-quantum cryptography

Grover's algorithm finds 128-bit AES key using 2^{64} quantum AES evaluations.

Sensible risk management: Assume that this is feasible— or will be feasible in, e.g., 2025. "AES-128 is dead."

Fix: Switch to AES-256.

AES-256 has 14 rounds. Maybe 12 rounds are enough for 2^{128} post-quantum security? Maybe 10 rounds are enough?

Shor's algorithm (similar to Simon's algorithm) factors RSA modulus N by finding period of $x \mapsto 2^x \bmod N$.

Number of qubit operations $\approx$ number of bit operations to compute $2^x \bmod N$.

$\approx 2^{64}$ qubit operations when N is around 1 gigabyte.

Post-quantum cryptography

Grover's algorithm finds 128-bit AES key using 2^{64} quantum AES evaluations.

Sensible risk management: Assume that this is feasible— or will be feasible in, e.g., 2025. "AES-128 is dead."

Fix: Switch to AES-256.

AES-256 has 14 rounds. Maybe 12 rounds are enough for 2^{128} post-quantum security? Maybe 10 rounds are enough?

Shor's algorithm (similar to Simon's algorithm) factors RSA modulus N by finding period of $x \mapsto 2^x \bmod N$.

Number of qubit operations $\approx$ number of bit operations to compute $2^x \bmod N$.

$\approx 2^{64}$ qubit operations when N is around 1 gigabyte.

Shor also finds $\log_g h$ by finding period of $(x, y) \mapsto g^x h^y$.

Post-quantum cryptography

Grover's algorithm finds 128-bit AES key using 2^{64} quantum AES evaluations.

Sensible risk management: Assume that this is feasible— or will be feasible in, e.g., 2025. "AES-128 is dead."

Fix: Switch to AES-256.

AES-256 has 14 rounds. Maybe 12 rounds are enough for 2^{128} post-quantum security? Maybe 10 rounds are enough?

Shor's algorithm (similar to Simon's algorithm) factors RSA modulus N by finding period of $x \mapsto 2^x \bmod N$.

Number of qubit operations $\approx$ number of bit operations to compute $2^x \bmod N$.

$\approx 2^{64}$ qubit operations when N is around 1 gigabyte.

Shor also finds $\log_g h$ by finding period of $(x, y) \mapsto g^x h^y$.

"RSA is dead. ECC is dead."

Post-quantum cryptography

Grover's algorithm finds 128-bit AES key using 2^{64} quantum AES evaluations.

Sensible risk management: Assume that this is feasible— or will be feasible in, e.g., 2025. "AES-128 is dead."

Fix: Switch to AES-256.

AES-256 has 14 rounds. Maybe 12 rounds are enough for 2^{128} post-quantum security? Maybe 10 rounds are enough?

Shor's algorithm (similar to Simon's algorithm) factors RSA modulus N by finding period of $x \mapsto 2^x \bmod N$.

Number of qubit operations $\approx$ number of bit operations to compute $2^x \bmod N$.

$\approx 2^{64}$ qubit operations when N is around 1 gigabyte.

Shor also finds $\log_g h$ by finding period of $(x, y) \mapsto g^x h^y$.

"RSA is dead. ECC is dead." But some systems seem safe.

antum cryptography

algorithm finds
AES key using
ntum AES evaluations.

risk management:
that this is feasible—
e feasible in, e.g., 2025.
28 is dead."

itch to AES-256.

6 has 14 rounds.
12 rounds are enough
post-quantum security?
10 rounds are enough?

Shor's algorithm
(similar to Simon's algorithm)
factors RSA modulus N by
finding period of $x \mapsto 2^x \bmod N$.

Number of qubit operations
$\approx$ number of bit operations
to compute $2^x \bmod N$.

$\approx 2^{64}$ qubit operations
when N is around 1 gigabyte.

Shor also finds $\log_g h$ by
finding period of $(x, y) \mapsto g^x h^y$.

"RSA is dead. ECC is dead."
But some systems seem safe.

Hash-ba
Example
public-ke

Code-ba
Example
hidden-G
public-ke

Lattice-
Example

Multiva
equatio
Example
1996 Pa
public-ke

ptography

n finds
sing
evaluations.

agement:
s feasible—
in, e.g., 2025.
"

S-256.

ounds.
are enough
ntum security?
are enough?

Shor's algorithm
(similar to Simon's algorithm)
factors RSA modulus N by
finding period of $x \mapsto 2^x \bmod N$.

Number of qubit operations
$\approx$ number of bit operations
to compute $2^x \bmod N$.

$\approx 2^{64}$ qubit operations
when N is around 1 gigabyte.

Shor also finds $\log_g h$ by
finding period of $(x, y) \mapsto g^x h^y$.

"RSA is dead. ECC is dead."
But some systems seem safe.

Hash-based signa
Example: 1979 M
public-key signatu

Code-based cryp
Example: 1978 M
hidden-Goppa-cod
public-key encrypt

Lattice-based cry
Example: 1998 "N

Multivariate-qua
equations crypto
Example:
1996 Patarin "HF
public-key signatu

ns.

—

025.

h

rity?

h?

Shor's algorithm
(similar to Simon's algorithm)
factors RSA modulus N by
finding period of $x \mapsto 2^x \bmod N$.

Number of qubit operations
$\approx$ number of bit operations
to compute $2^x \bmod N$.

$\approx 2^{64}$ qubit operations
when N is around 1 gigabyte.

Shor also finds $\log_g h$ by
finding period of $(x, y) \mapsto g^x h^y$.

"RSA is dead. ECC is dead."
But some systems seem safe.

Hash-based signatures.
Example: 1979 Merkle hash
public-key signature system.

Code-based cryptography.
Example: 1978 McEliece
hidden-Goppa-code
public-key encryption system

Lattice-based cryptograph
Example: 1998 "NTRU".

Multivariate-quadratic-
equations cryptography.
Example:
1996 Patarin "HFE^{v-}"
public-key signature system.

Shor's algorithm
(similar to Simon's algorithm)
factors RSA modulus N by
finding period of $x \mapsto 2^x \bmod N$.

Number of qubit operations
$\approx$ number of bit operations
to compute $2^x \bmod N$.

$\approx 2^{64}$ qubit operations
when N is around 1 gigabyte.

Shor also finds $\log_g h$ by
finding period of $(x, y) \mapsto g^x h^y$.

"RSA is dead. ECC is dead."
But some systems seem safe.

Hash-based signatures.
Example: 1979 Merkle hash-tree
public-key signature system.

Code-based cryptography.
Example: 1978 McEliece
hidden-Goppa-code
public-key encryption system.

Lattice-based cryptography.
Example: 1998 "NTRU".

Multivariate-quadratic-equations cryptography.
Example:
1996 Patarin "HFE^{v-}"
public-key signature system.

lgorithm
to Simon's algorithm)
RSA modulus N by
period of $x \mapsto 2^x \bmod N$.

of qubit operations
er of bit operations
ute $2^x \bmod N$.

bit operations
is around 1 gigabyte.

o finds $\log_g h$ by
period of $(x, y) \mapsto g^x h^y$.

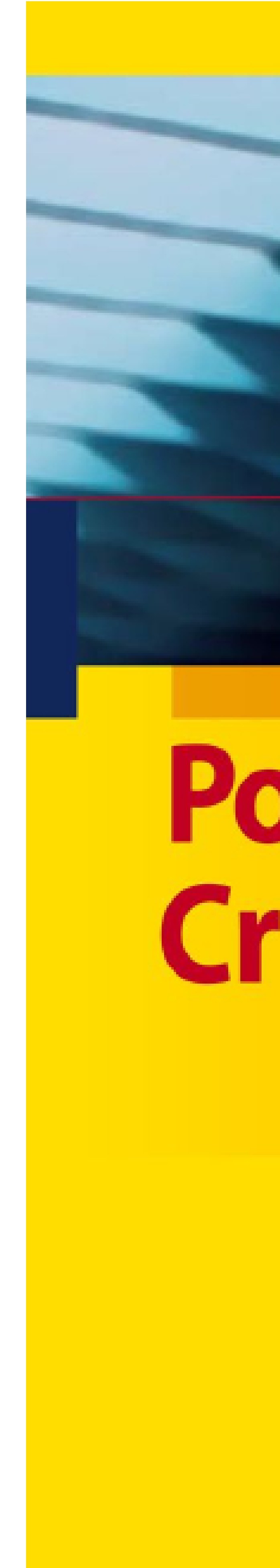
dead. ECC is dead."
ne systems seem safe.

Hash-based signatures.
Example: 1979 Merkle hash-tree public-key signature system.

Code-based cryptography.
Example: 1978 McEliece hidden-Goppa-code public-key encryption system.

Lattice-based cryptography.
Example: 1998 "NTRU".

Multivariate-quadratic-equations cryptography.
Example:
1996 Patarin "$\text{HFE}^{\text{v}-}$"
public-key signature system.



s algorithm)
lus N by
$x \mapsto 2^x \bmod N$.

operations
perations
d N.

ions
1 gigabyte.

$_g\, h$ by
$x, y) \mapsto g^x h^y$.

C is dead.”
seem safe.

Hash-based signatures.
Example: 1979 Merkle hash-tree public-key signature system.

Code-based cryptography.
Example: 1978 McEliece hidden-Goppa-code public-key encryption system.

Lattice-based cryptography.
Example: 1998 “NTRU”.

Multivariate-quadratic-equations cryptography.
Example:
1996 Patarin “HFE^{v-}”
public-key signature system.

n)

od N.

e.

$^x h^y$.

”

e.

Hash-based signatures.

Example: 1979 Merkle hash-tree public-key signature system.

Code-based cryptography.

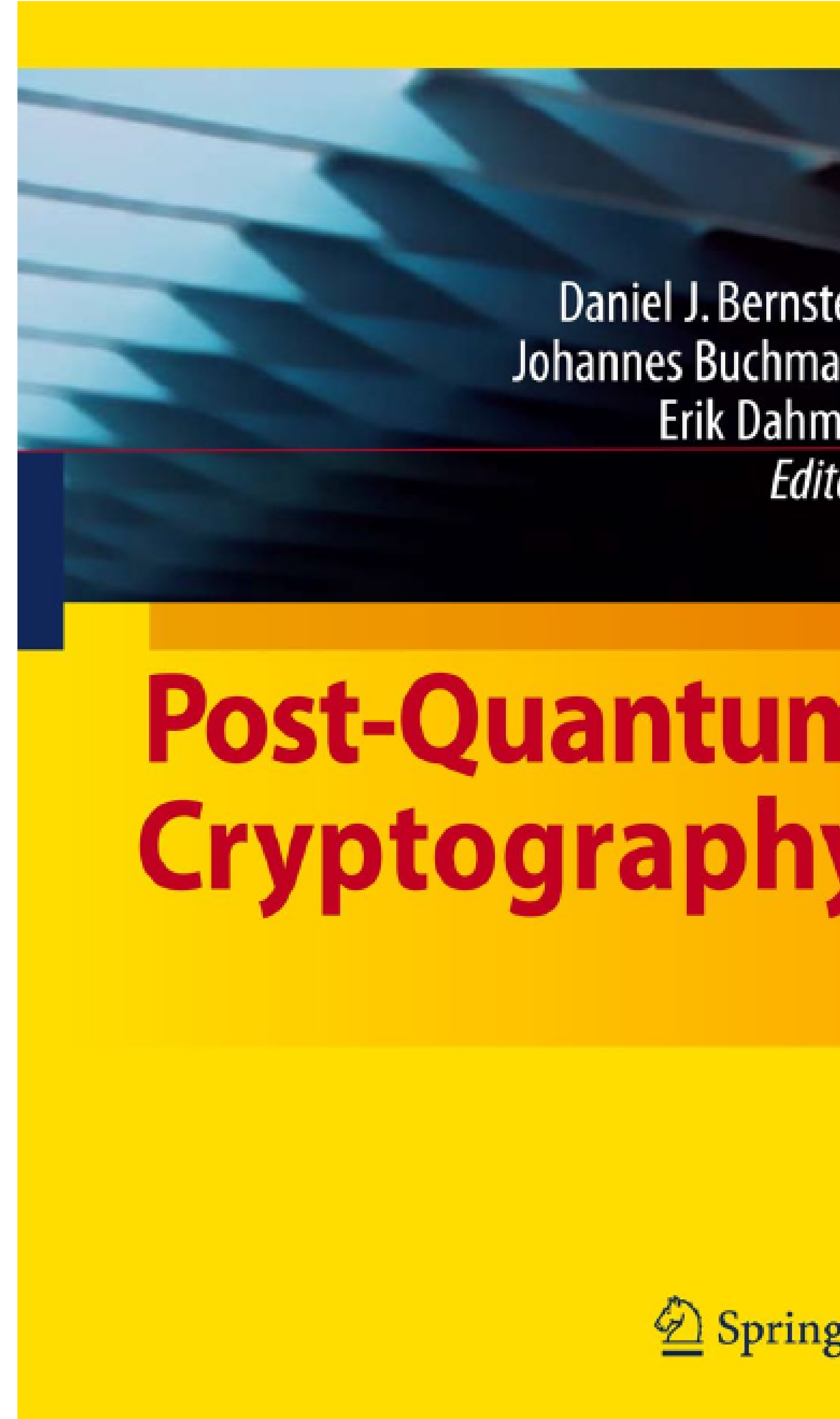
Example: 1978 McEliece hidden-Goppa-code public-key encryption system.

Lattice-based cryptography.

Example: 1998 “NTRU”.

Multivariate-quadratic-equations cryptography.

Example:
1996 Patarin “HFE^{v-}”
public-key signature system.

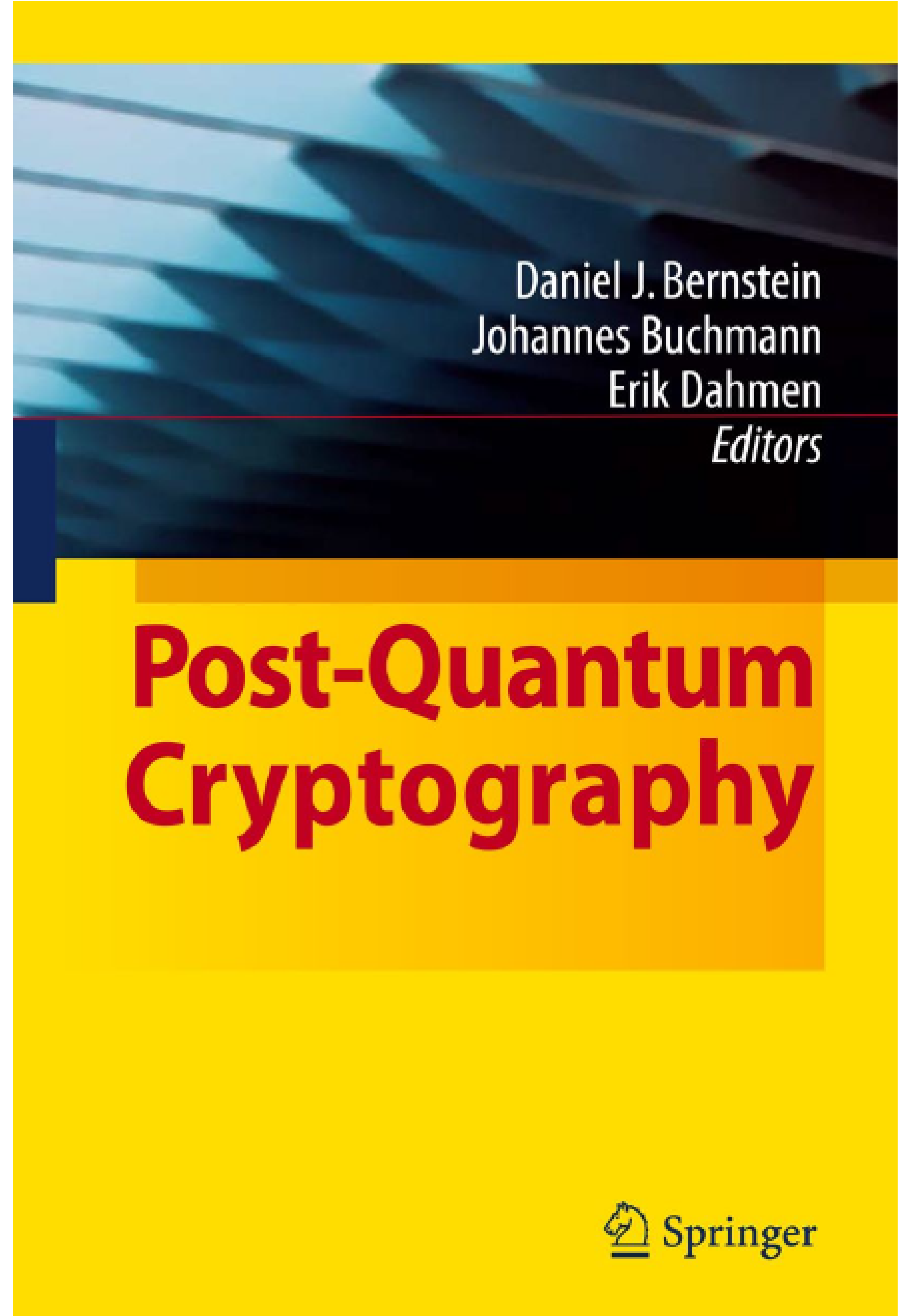


Hash-based signatures.
Example: 1979 Merkle hash-tree public-key signature system.

Code-based cryptography.
Example: 1978 McEliece hidden-Goppa-code public-key encryption system.

Lattice-based cryptography.
Example: 1998 "NTRU".

Multivariate-quadratic-equations cryptography.
Example: 1996 Patarin "HFE^{v-}" public-key signature system.

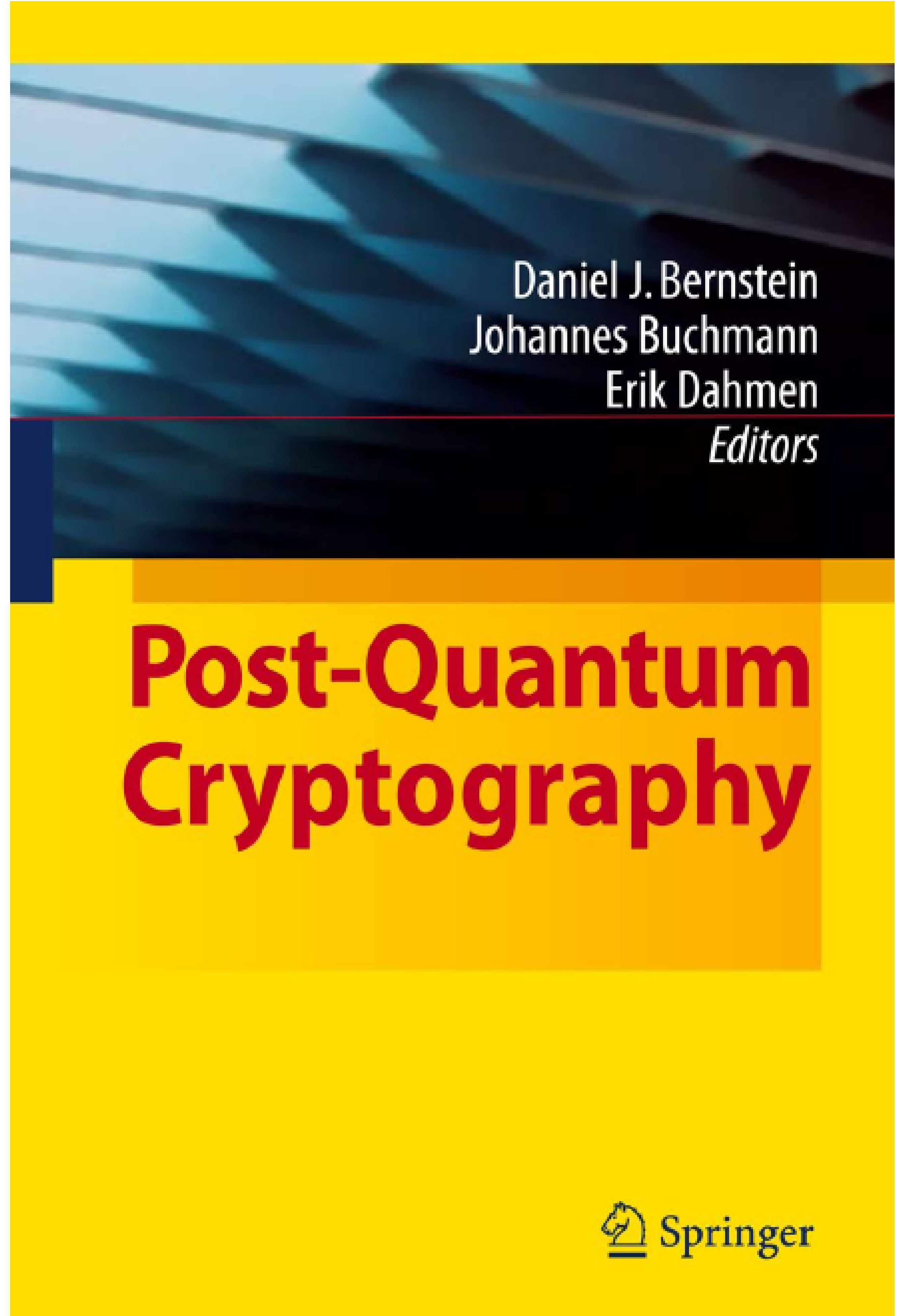


ased signatures.
e: 1979 Merkle hash-tree
ey signature system.

ased cryptography.
e: 1978 McEliece
Goppa-code
ey encryption system.

-based cryptography.
e: 1998 “NTRU”.

riate-quadratic-
ns cryptography.
e:
tarin “HFE^{v-}”
ey signature system.



The 197

(with 19

Receiver
500 × 10
Specifies

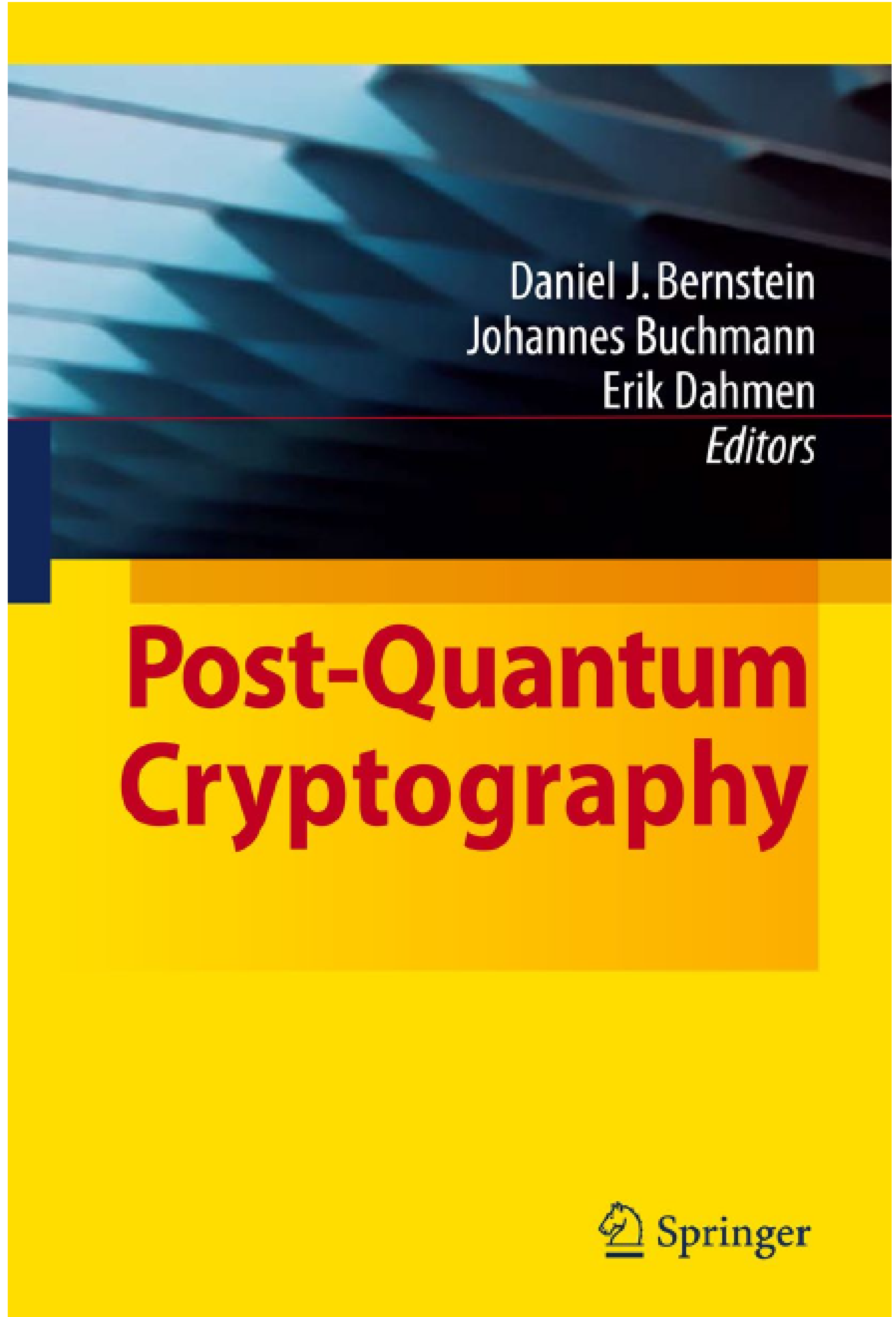
atures.
erkle hash-tree
re system.

tography.
cEliece
e
ion system.

yptography.
NTRU".

dratic-
graphy.

E^{v-}"
re system.



The 1978 McEliec

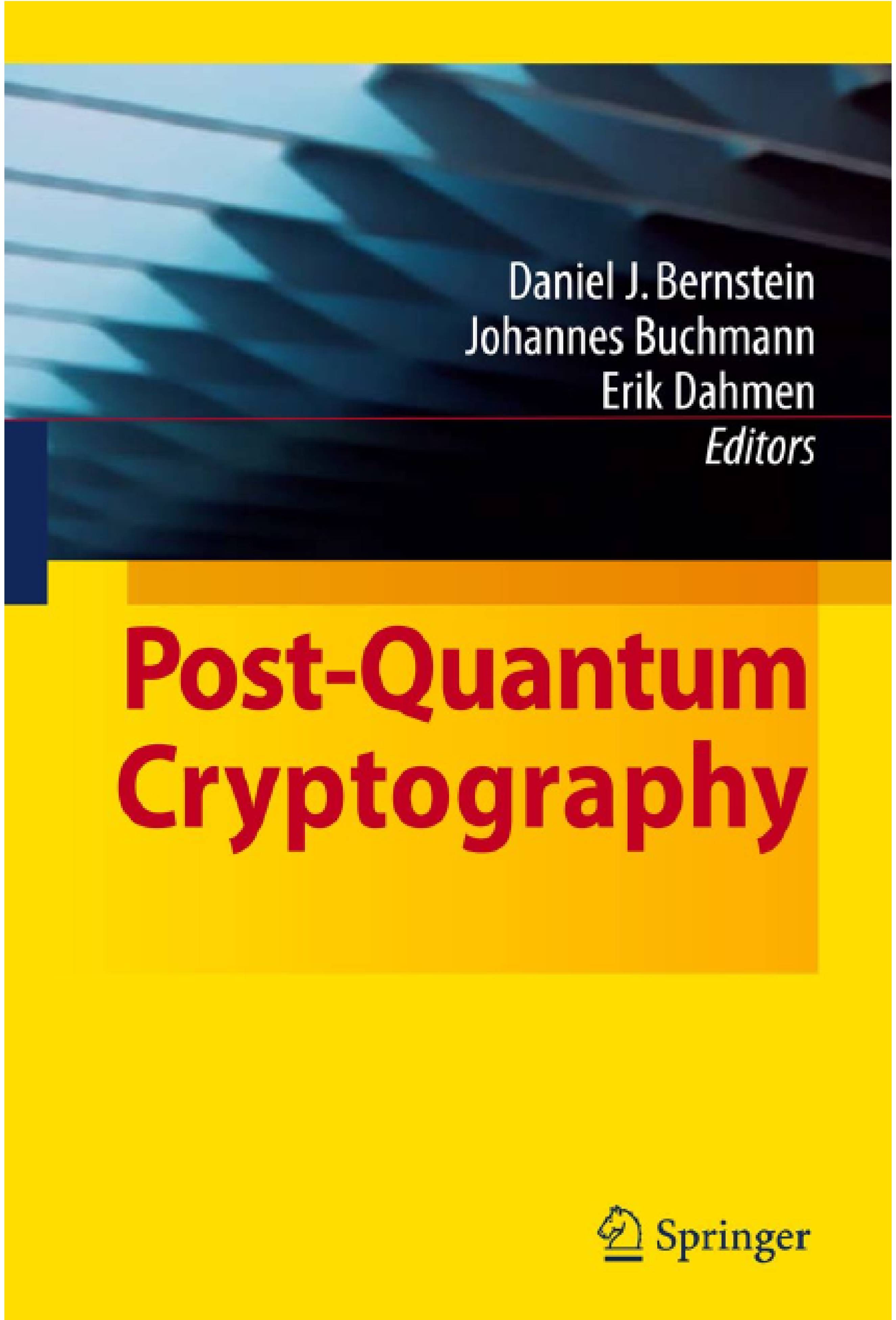
(with 1986 Nieder

Receiver's public k
500×1024 matrix
Specifies linear $\mathbf{F}_2^{1\ldots}$

-tree

n.

ny.



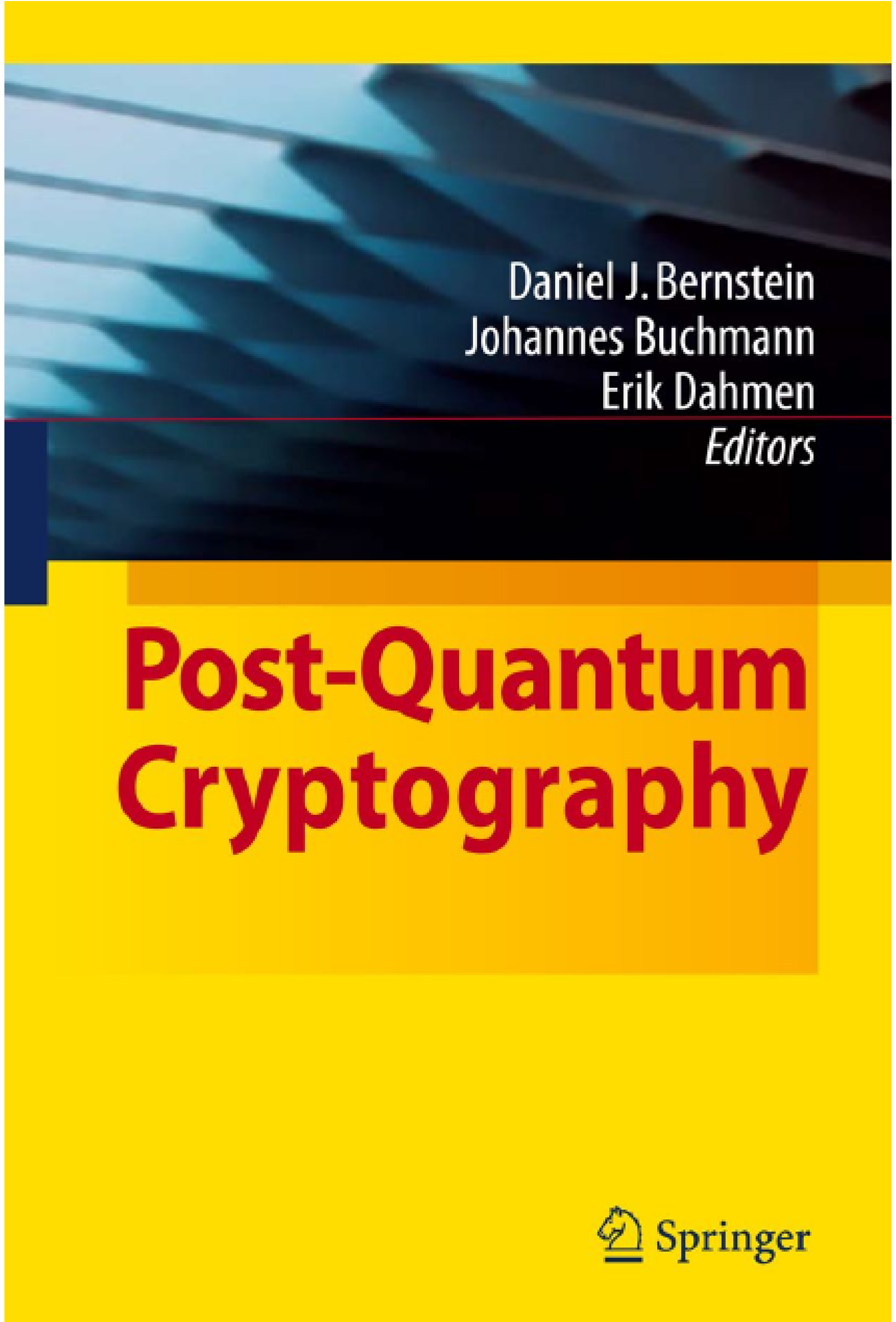
The 1978 McEliece cryptosy

(with 1986 Niederreiter spee

Receiver's public key: "rand

500×1024 matrix K over $\mathbf{F}$

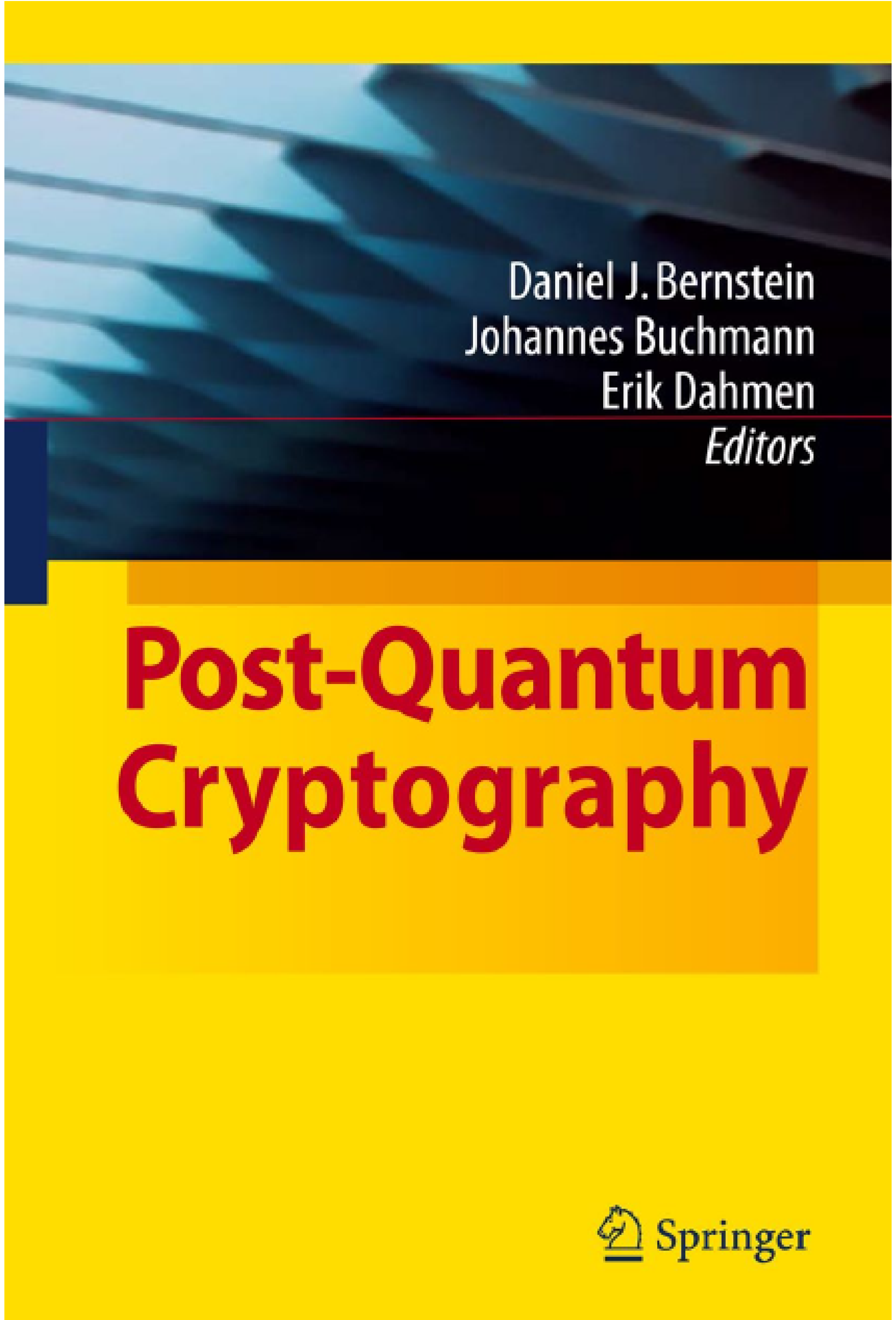
Specifies linear $\mathbf{F}_2^{1024} \rightarrow \mathbf{F}_2^{50}$



The 1978 McEliece cryptosystem

(with 1986 Niederreiter speedup)

Receiver's public key: "random" 500×1024 matrix K over $\mathbf{F}_2$. Specifies linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

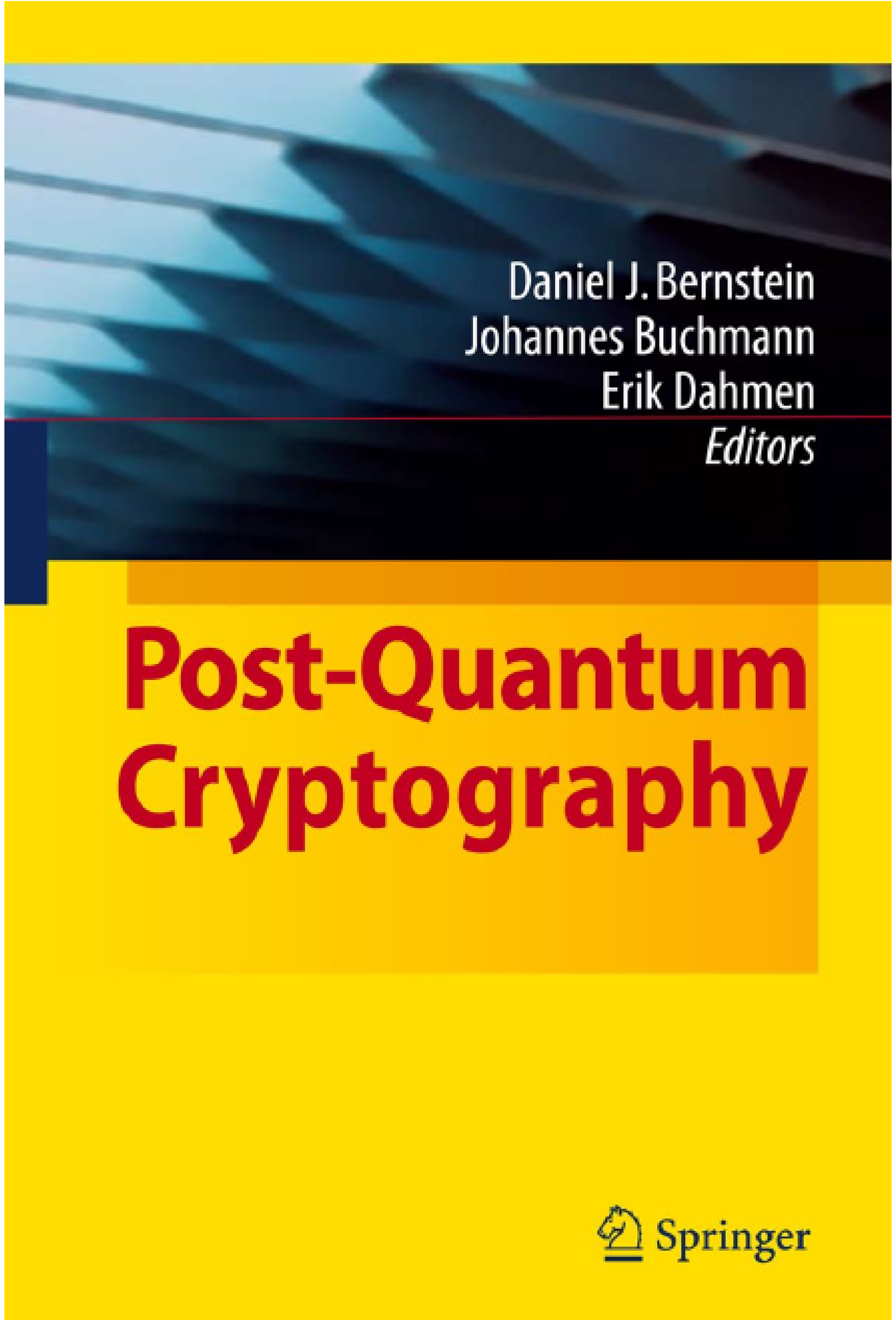


The 1978 McEliece cryptosystem

(with 1986 Niederreiter speedup)

Receiver's public key: "random" 500×1024 matrix K over $\mathbf{F}_2$. Specifies linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

Messages suitable for encryption: 1024-bit strings of weight 50. $\{e \in \mathbf{F}_2^{1024} : \#\{i : e_i = 1\} = 50\}$.



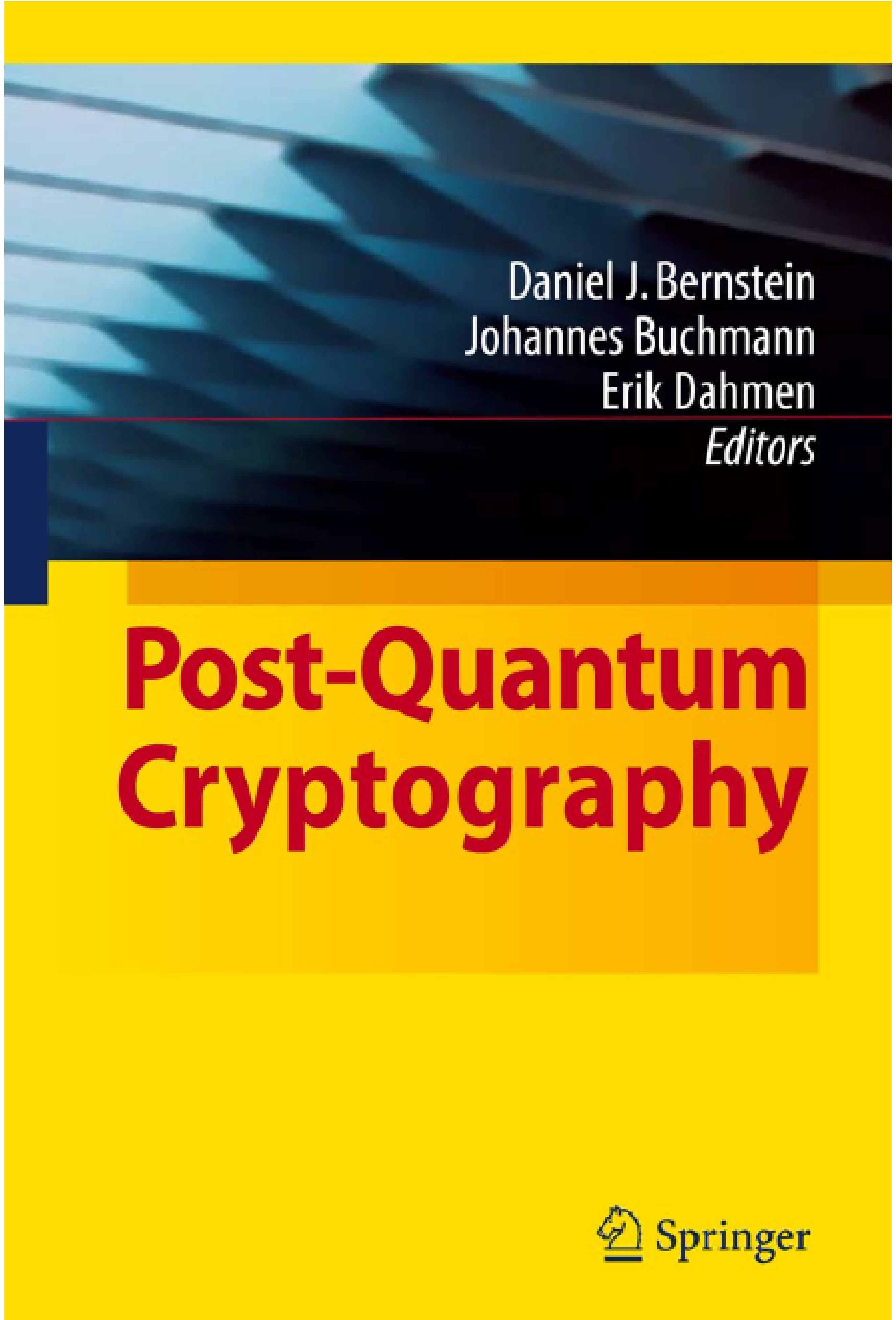
The 1978 McEliece cryptosystem

(with 1986 Niederreiter speedup)

Receiver's public key: "random" 500×1024 matrix K over $\mathbf{F}_2$. Specifies linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

Messages suitable for encryption: 1024-bit strings of weight 50. $\{e \in \mathbf{F}_2^{1024} : \#\{i : e_i = 1\} = 50\}$.

Encryption of e is $Ke \in \mathbf{F}_2^{500}$.



The 1978 McEliece cryptosystem

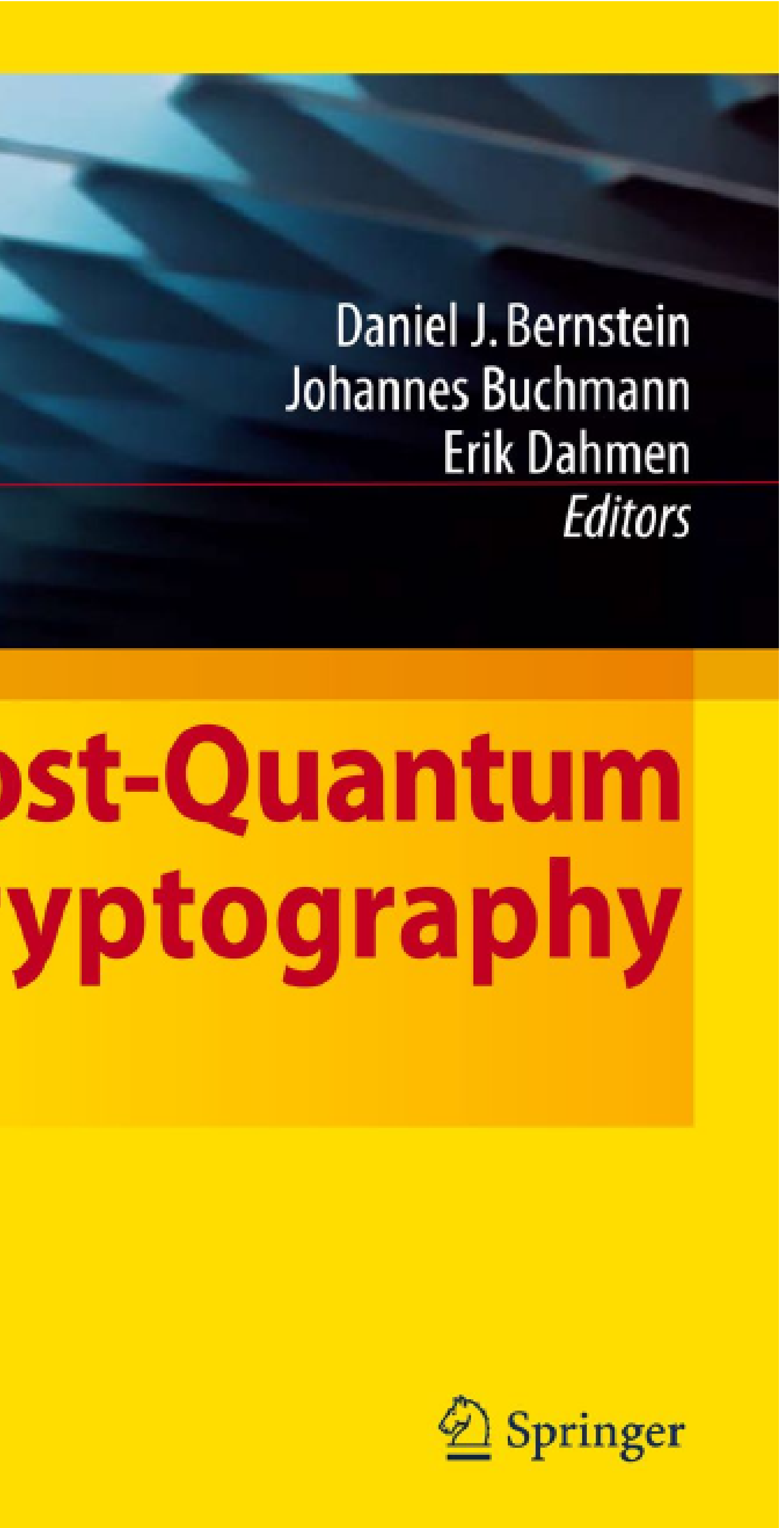
(with 1986 Niederreiter speedup)

Receiver's public key: "random" 500×1024 matrix K over $\mathbf{F}_2$. Specifies linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

Messages suitable for encryption: 1024-bit strings of weight 50. $\{e \in \mathbf{F}_2^{1024} : \#\{i : e_i = 1\} = 50\}$.

Encryption of e is $Ke \in \mathbf{F}_2^{500}$.

"Padding": Choose random e; send Ke; use SHA-256(e, Ke) as AES-256-GCM key to encrypt actual message of any length.



The 1978 McEliece cryptosystem

(with 1986 Niederreiter speedup)

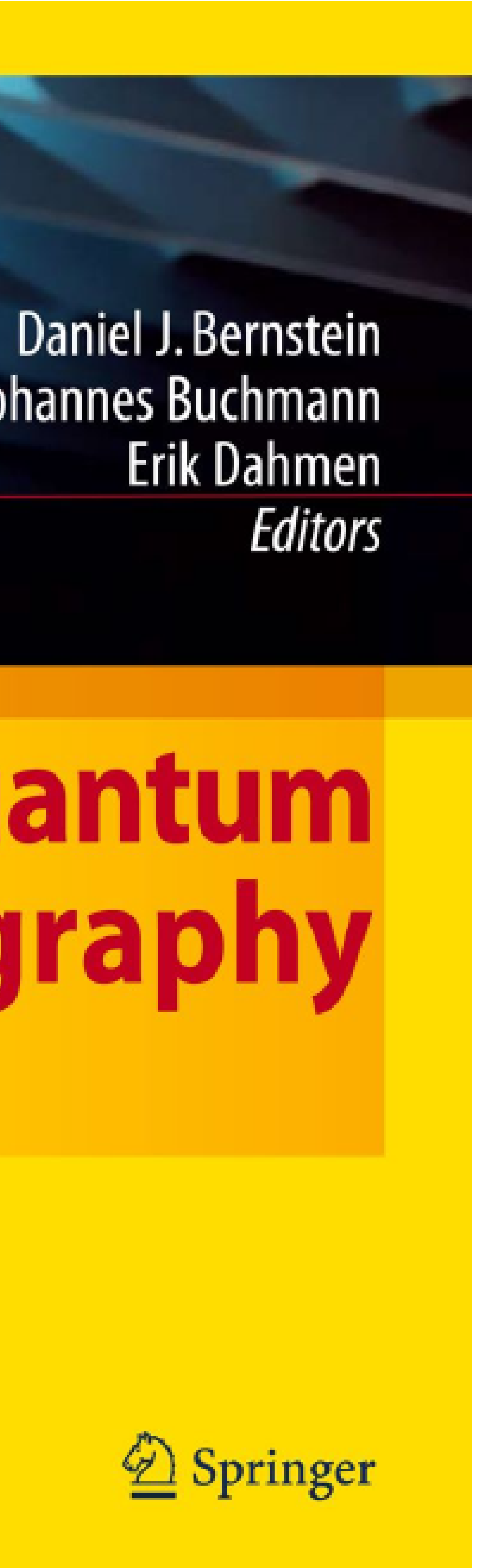
Receiver's public key: "random" 500×1024 matrix K over $\mathbf{F}_2$. Specifies linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

Messages suitable for encryption: 1024-bit strings of weight 50. $\{e \in \mathbf{F}_2^{1024} : \#\{i : e_i = 1\} = 50\}$.

Encryption of e is $Ke \in \mathbf{F}_2^{500}$.

"Padding": Choose random e; send Ke; use SHA-256(e, Ke) as AES-256-GCM key to encrypt actual message of any length.

Attacker
easily wo
from Ke
such tha



The 1978 McEliece cryptosystem

(with 1986 Niederreiter speedup)

Receiver's public key: "random" 500×1024 matrix K over $\mathbf{F}_2$. Specifies linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

Messages suitable for encryption: 1024-bit strings of weight 50. $\{e \in \mathbf{F}_2^{1024} : \#\{i : e_i = 1\} = 50\}$.

Encryption of e is $Ke \in \mathbf{F}_2^{500}$.

"Padding": Choose random e; send Ke; use SHA-256(e, Ke) as AES-256-GCM key to encrypt actual message of any length.

Attacker, by linear
easily works backw
from Ke to *some*
such that $Kv = K$

The 1978 McEliece cryptosystem

(with 1986 Niederreiter speedup)

Receiver's public key: "random" 500×1024 matrix K over $\mathbf{F}_2$. Specifies linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

Messages suitable for encryption: 1024-bit strings of weight 50. $\{e \in \mathbf{F}_2^{1024} : \#\{i : e_i = 1\} = 50\}$.

Encryption of e is $Ke \in \mathbf{F}_2^{500}$.

"Padding": Choose random e; send Ke; use SHA-256(e, Ke) as AES-256-GCM key to encrypt actual message of any length.

Attacker, by linear algebra, easily works backwards from Ke to *some* $v \in \mathbf{F}_2^{1024}$ such that $Kv = Ke$.

The 1978 McEliece cryptosystem

(with 1986 Niederreiter speedup)

Receiver's public key: "random" 500×1024 matrix K over $\mathbf{F}_2$. Specifies linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

Messages suitable for encryption: 1024-bit strings of weight 50. $\{e \in \mathbf{F}_2^{1024} : \#\{i : e_i = 1\} = 50\}$.

Encryption of e is $Ke \in \mathbf{F}_2^{500}$.

"Padding": Choose random e; send Ke; use SHA-256(e, Ke) as AES-256-GCM key to encrypt actual message of any length.

Attacker, by linear algebra, easily works backwards from Ke to *some* $v \in \mathbf{F}_2^{1024}$ such that $Kv = Ke$.

The 1978 McEliece cryptosystem

(with 1986 Niederreiter speedup)

Receiver's public key: "random" 500×1024 matrix K over $\mathbf{F}_2$. Specifies linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

Messages suitable for encryption: 1024-bit strings of weight 50. $\{e \in \mathbf{F}_2^{1024} : \#\{i : e_i = 1\} = 50\}$.

Encryption of e is $Ke \in \mathbf{F}_2^{500}$.

"Padding": Choose random e; send Ke; use SHA-256(e, Ke) as AES-256-GCM key to encrypt actual message of any length.

Attacker, by linear algebra, easily works backwards from Ke to *some* $v \in \mathbf{F}_2^{1024}$ such that $Kv = Ke$.

i.e. Attacker finds *some* element $v \in e + \operatorname{Ker} K$. Note that $\# \operatorname{Ker} K \geq 2^{524}$.

Attacker wants to decode v: to find element of $\operatorname{Ker} K$ at distance only 50 from v. Presumably unique, revealing e.

The 1978 McEliece cryptosystem

(with 1986 Niederreiter speedup)

Receiver's public key: "random" 500×1024 matrix K over $\mathbf{F}_2$. Specifies linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

Messages suitable for encryption: 1024-bit strings of weight 50. $\{e \in \mathbf{F}_2^{1024} : \#\{i : e_i = 1\} = 50\}$.

Encryption of e is $Ke \in \mathbf{F}_2^{500}$.

"Padding": Choose random e; send Ke; use SHA-256(e, Ke) as AES-256-GCM key to encrypt actual message of any length.

Attacker, by linear algebra, easily works backwards from Ke to *some* $v \in \mathbf{F}_2^{1024}$ such that $Kv = Ke$.

i.e. Attacker finds *some* element $v \in e + \operatorname{Ker} K$. Note that $\# \operatorname{Ker} K \geq 2^{524}$.

Attacker wants to decode v: to find element of $\operatorname{Ker} K$ at distance only 50 from v. Presumably unique, revealing e.

But decoding isn't easy!

8 McEliece cryptosystem

986 Niederreiter speedup)

's public key: "random"
024 matrix K over $\mathbf{F}_2$.
s linear $\mathbf{F}_2^{1024} \to \mathbf{F}_2^{500}$.

es suitable for encryption:
strings of weight 50.
^{024} : $\#\{i : e_i = 1\} = 50\}$.

on of e is $Ke \in \mathbf{F}_2^{500}$.

g": Choose random e;
e; use SHA-256(e, Ke) as
6-GCM key to encrypt
nessage of any length.

Attacker, by linear algebra,
easily works backwards
from Ke to *some* $v \in \mathbf{F}_2^{1024}$
such that $Kv = Ke$.

i.e. Attacker finds *some*
element $v \in e + \operatorname{Ker} K$.
Note that $\# \operatorname{Ker} K \geq 2^{524}$.

Attacker wants to decode v:
to find element of $\operatorname{Ker} K$
at distance only 50 from v.
Presumably unique, revealing e.

But decoding isn't easy!

Informat

Choose
$S \subseteq \{1,$

For typic
that $\mathbf{F}_2^S$
is inverti

e cryptosystem

reiter speedup)

key: "random"

K over $\mathbf{F}_2$.

$024} \to \mathbf{F}_2^{500}$.

for encryption:

f weight 50.

$e_i = 1\} = 50\}$.

$Ke \in \mathbf{F}_2^{500}$.

se random e;

A-256(e, Ke) as

y to encrypt

any length.

Attacker, by linear algebra, easily works backwards from Ke to *some* $v \in \mathbf{F}_2^{1024}$ such that $Kv = Ke$.

i.e. Attacker finds *some* element $v \in e + \operatorname{Ker} K$. Note that $\# \operatorname{Ker} K \geq 2^{524}$.

Attacker wants to decode v: to find element of $\operatorname{Ker} K$ at distance only 50 from v. Presumably unique, revealing e.

But decoding isn't easy!

Information-set de

Choose random si
$S \subseteq \{1, 2, 3, \ldots, 1$

For typical K: Go
that $\mathbf{F}_2^S \hookrightarrow \mathbf{F}_2^{1024}$
is invertible.

ystem

edup)

om"

$_2$.

^{0}.

ption:

0.

= 50}.

^{0}.

e;

e) as

pt

h.

Attacker, by linear algebra, easily works backwards from Ke to *some* $v \in \mathbf{F}_2^{1024}$ such that $Kv = Ke$.

i.e. Attacker finds *some* element $v \in e + \operatorname{Ker} K$. Note that $\# \operatorname{Ker} K \geq 2^{524}$.

Attacker wants to decode v: to find element of $\operatorname{Ker} K$ at distance only 50 from v. Presumably unique, revealing e.

But decoding isn't easy!

Information-set decoding

Choose random size-500 sub
$S \subseteq \{1, 2, 3, \ldots, 1024\}$.

For typical K: Good chance
that $\mathbf{F}_2^S \hookrightarrow \mathbf{F}_2^{1024} \xrightarrow{K} \mathbf{F}_2^{500}$
is invertible.

Attacker, by linear algebra, easily works backwards from Ke to *some* $v \in \mathbf{F}_2^{1024}$ such that $Kv = Ke$.

i.e. Attacker finds *some* element $v \in e + \operatorname{Ker} K$. Note that $\# \operatorname{Ker} K \geq 2^{524}$.

Attacker wants to decode v: to find element of $\operatorname{Ker} K$ at distance only 50 from v. Presumably unique, revealing e.

But decoding isn't easy!

Information-set decoding

Choose random size-500 subset $S \subseteq \{1, 2, 3, \ldots, 1024\}$.

For typical K: Good chance that $\mathbf{F}_2^S \hookrightarrow \mathbf{F}_2^{1024} \xrightarrow{K} \mathbf{F}_2^{500}$ is invertible.

Attacker, by linear algebra, easily works backwards from Ke to *some* $v \in \mathbf{F}_2^{1024}$ such that $Kv = Ke$.

i.e. Attacker finds *some* element $v \in e + \operatorname{Ker} K$. Note that $\# \operatorname{Ker} K \geq 2^{524}$.

Attacker wants to decode v: to find element of $\operatorname{Ker} K$ at distance only 50 from v. Presumably unique, revealing e.

But decoding isn't easy!

Information-set decoding

Choose random size-500 subset $S \subseteq \{1, 2, 3, \ldots, 1024\}$.

For typical K: Good chance that $\mathbf{F}_2^S \hookrightarrow \mathbf{F}_2^{1024} \xrightarrow{K} \mathbf{F}_2^{500}$ is invertible.

Hope $e \in \mathbf{F}_2^S$; chance $\approx 2^{-53}$. Apply inverse map to Ke, revealing e if $e \in \mathbf{F}_2^S$.

Attacker, by linear algebra, easily works backwards from Ke to *some* $v \in \mathbf{F}_2^{1024}$ such that $Kv = Ke$.

i.e. Attacker finds *some* element $v \in e + \operatorname{Ker} K$. Note that $\# \operatorname{Ker} K \geq 2^{524}$.

Attacker wants to decode v: to find element of $\operatorname{Ker} K$ at distance only 50 from v. Presumably unique, revealing e.

But decoding isn't easy!

Information-set decoding

Choose random size-500 subset $S \subseteq \{1, 2, 3, \ldots, 1024\}$.

For typical K: Good chance that $\mathbf{F}_2^S \hookrightarrow \mathbf{F}_2^{1024} \xrightarrow{K} \mathbf{F}_2^{500}$ is invertible.

Hope $e \in \mathbf{F}_2^S$; chance $\approx 2^{-53}$. Apply inverse map to Ke, revealing e if $e \in \mathbf{F}_2^S$.

If $e \notin \mathbf{F}_2^S$, try again. $\approx 2^{80}$ bit operations in total.

Attacker, by linear algebra, easily works backwards from Ke to *some* $v \in \mathbf{F}_2^{1024}$ such that $Kv = Ke$.

i.e. Attacker finds *some* element $v \in e + \operatorname{Ker} K$. Note that $\# \operatorname{Ker} K \geq 2^{524}$.

Attacker wants to decode v: to find element of $\operatorname{Ker} K$ at distance only 50 from v. Presumably unique, revealing e.

But decoding isn't easy!

Information-set decoding

Choose random size-500 subset $S \subseteq \{1, 2, 3, \ldots, 1024\}$.

For typical K: Good chance that $\mathbf{F}_2^S \hookrightarrow \mathbf{F}_2^{1024} \xrightarrow{K} \mathbf{F}_2^{500}$ is invertible.

Hope $e \in \mathbf{F}_2^S$; chance $\approx 2^{-53}$. Apply inverse map to Ke, revealing e if $e \in \mathbf{F}_2^S$.

If $e \notin \mathbf{F}_2^S$, try again. $\approx 2^{80}$ bit operations in total.

Bad estimate by McEliece: $\approx 2^{64}$.

r, by linear algebra,
orks backwards
e to *some* $v \in \mathbf{F}_2^{1024}$
nt $Kv = Ke$.

acker finds *some*
$v \in e + \operatorname{Ker} K$.
at $\# \operatorname{Ker} K \geq 2^{524}$.

r wants to decode v:
element of $\operatorname{Ker} K$
nce only 50 from v.
ably unique, revealing e.

oding isn't easy!

Information-set decoding

Choose random size-500 subset $S \subseteq \{1, 2, 3, \ldots, 1024\}$.

For typical K: Good chance that $\mathbf{F}_2^S \hookrightarrow \mathbf{F}_2^{1024} \xrightarrow{K} \mathbf{F}_2^{500}$ is invertible.

Hope $e \in \mathbf{F}_2^S$; chance $\approx 2^{-53}$.
Apply inverse map to Ke, revealing e if $e \in \mathbf{F}_2^S$.

If $e \notin \mathbf{F}_2^S$, try again.
$\approx 2^{80}$ bit operations in total.

Bad estimate by McEliece: $\approx 2^{64}$.

Analyzin
1962 Pra
1988 Le
1989 Kr
1989 Du
1990 Co
1990 va
1991 Co
1993 Ch
1993 Ch
1994 va
1994 Ca
1998 Ca
1998 Ca

r algebra,

vards

$v \in \mathbf{F}_2^{1024}$

Ke.

some

Ker K.

$K \geq 2^{524}$.

decode v:

Ker K

0 from v.

e, revealing e.

t easy!

Information-set decoding

Choose random size-500 subset $S \subseteq \{1, 2, 3, \ldots, 1024\}$.

For typical K: Good chance that $\mathbf{F}_2^S \hookrightarrow \mathbf{F}_2^{1024} \xrightarrow{K} \mathbf{F}_2^{500}$ is invertible.

Hope $e \in \mathbf{F}_2^S$; chance $\approx 2^{-53}$. Apply inverse map to Ke, revealing e if $e \in \mathbf{F}_2^S$.

If $e \notin \mathbf{F}_2^S$, try again. $\approx 2^{80}$ bit operations in total.

Bad estimate by McEliece: $\approx 2^{64}$.

Analyzing and opt

1962 Prange. 198

1988 Lee–Brickell.

1989 Krouk. 1989

1989 Dumer.

1990 Coffey–Good

1990 van Tilburg.

1991 Coffey–Good

1993 Chabanne–C

1993 Chabaud.

1994 van Tilburg.

1994 Canteaut–Ch

1998 Canteaut–Ch

1998 Canteaut–Se

g e.

Information-set decoding

Choose random size-500 subset $S \subseteq \{1, 2, 3, \ldots, 1024\}$.

For typical K: Good chance that $\mathbf{F}_2^S \hookrightarrow \mathbf{F}_2^{1024} \xrightarrow{K} \mathbf{F}_2^{500}$ is invertible.

Hope $e \in \mathbf{F}_2^S$; chance $\approx 2^{-53}$. Apply inverse map to Ke, revealing e if $e \in \mathbf{F}_2^S$.

If $e \notin \mathbf{F}_2^S$, try again. $\approx 2^{80}$ bit operations in total.

Bad estimate by McEliece: $\approx 2^{64}$.

Analyzing and optimizing at

1962 Prange. 1981 Omura.

1988 Lee–Brickell. 1988 Lec

1989 Krouk. 1989 Stern.

1989 Dumer.

1990 Coffey–Goodman.

1990 van Tilburg. 1991 Dur

1991 Coffey–Goodman–Farr

1993 Chabanne–Courteau.

1993 Chabaud.

1994 van Tilburg.

1994 Canteaut–Chabanne.

1998 Canteaut–Chabaud.

1998 Canteaut–Sendrier.

Information-set decoding

Choose random size-500 subset $S \subseteq \{1, 2, 3, \ldots, 1024\}$.

For typical K: Good chance that $\mathbf{F}_2^S \hookrightarrow \mathbf{F}_2^{1024} \xrightarrow{K} \mathbf{F}_2^{500}$ is invertible.

Hope $e \in \mathbf{F}_2^S$; chance $\approx 2^{-53}$.
Apply inverse map to Ke, revealing e if $e \in \mathbf{F}_2^S$.

If $e \notin \mathbf{F}_2^S$, try again.
$\approx 2^{80}$ bit operations in total.

Bad estimate by McEliece: $\approx 2^{64}$.

Analyzing and optimizing attacks:

1962 Prange. 1981 Omura.
1988 Lee–Brickell. 1988 Leon.
1989 Krouk. 1989 Stern.
1989 Dumer.
1990 Coffey–Goodman.
1990 van Tilburg. 1991 Dumer.
1991 Coffey–Goodman–Farrell.
1993 Chabanne–Courteau.
1993 Chabaud.
1994 van Tilburg.
1994 Canteaut–Chabanne.
1998 Canteaut–Chabaud.
1998 Canteaut–Sendrier.

tion-set decoding

random size-500 subset
2, 3, . . . , 1024}.

cal K: Good chance
$\hookrightarrow \mathbf{F}_2^{1024} \xrightarrow{K} \mathbf{F}_2^{500}$
ible.

$\in \mathbf{F}_2^S$; chance $\approx 2^{-53}$.
verse map to Ke,
g e if $e \in \mathbf{F}_2^S$.

$_2^S$, try again.
operations in total.

mate by McEliece: $\approx 2^{64}$.

Analyzing and optimizing attacks:

1962 Prange. 1981 Omura.
1988 Lee–Brickell. 1988 Leon.
1989 Krouk. 1989 Stern.
1989 Dumer.
1990 Coffey–Goodman.
1990 van Tilburg. 1991 Dumer.
1991 Coffey–Goodman–Farrell.
1993 Chabanne–Courteau.
1993 Chabaud.
1994 van Tilburg.
1994 Canteaut–Chabanne.
1998 Canteaut–Chabaud.
1998 Canteaut–Sendrier.

2008 Be
mo
att
2009 Be
Pe
2009 Be
2009 Fi
2010 Be
2011 Ma
2011 Be
2012 Be
2013 Be
Me
2015 Ma

coding

ze-500 subset

024}.

od chance

$\xrightarrow{K} \mathbf{F}_2^{500}$

nce $\approx 2^{-53}$.

to Ke,

$\mathbf{F}_2^S$.

n.

ns in total.

McEliece: $\approx 2^{64}$.

Analyzing and optimizing attacks:

1962 Prange. 1981 Omura.

1988 Lee–Brickell. 1988 Leon.

1989 Krouk. 1989 Stern.

1989 Dumer.

1990 Coffey–Goodman.

1990 van Tilburg. 1991 Dumer.

1991 Coffey–Goodman–Farrell.

1993 Chabanne–Courteau.

1993 Chabaud.

1994 van Tilburg.

1994 Canteaut–Chabanne.

1998 Canteaut–Chabaud.

1998 Canteaut–Sendrier.

2008 Bernstein–La

more speedu

attack **actua**

2009 Bernstein–La

Peters–van T

2009 Bernstein: p

2009 Finiasz–Send

2010 Bernstein–La

2011 May–Meurer

2011 Becker–Coro

2012 Becker–Joux

2013 Bernstein–Je

Meurer: post

2015 May–Ozerov

oset

3.

r

≈2^{64}.

Analyzing and optimizing attacks:

1962 Prange. 1981 Omura.

1988 Lee–Brickell. 1988 Leon.

1989 Krouk. 1989 Stern.

1989 Dumer.

1990 Coffey–Goodman.

1990 van Tilburg. 1991 Dumer.

1991 Coffey–Goodman–Farrell.

1993 Chabanne–Courteau.

1993 Chabaud.

1994 van Tilburg.

1994 Canteaut–Chabanne.

1998 Canteaut–Chabaud.

1998 Canteaut–Sendrier.

2008 Bernstein–Lange–Peter
more speedups; ≈2^{60} c
attack **actually carrie**

2009 Bernstein–Lange–
Peters–van Tilborg.

2009 Bernstein: post-quantu

2009 Finiasz–Sendrier.

2010 Bernstein–Lange–Peter

2011 May–Meurer–Thomae.

2011 Becker–Coron–Joux.

2012 Becker–Joux–May–Me

2013 Bernstein–Jeffery–Lang
Meurer: post-quantum

2015 May–Ozerov.

Analyzing and optimizing attacks:

1962 Prange. 1981 Omura.
1988 Lee–Brickell. 1988 Leon.
1989 Krouk. 1989 Stern.
1989 Dumer.
1990 Coffey–Goodman.
1990 van Tilburg. 1991 Dumer.
1991 Coffey–Goodman–Farrell.
1993 Chabanne–Courteau.
1993 Chabaud.
1994 van Tilburg.
1994 Canteaut–Chabanne.
1998 Canteaut–Chabaud.
1998 Canteaut–Sendrier.

2008 Bernstein–Lange–Peters: more speedups; $\approx 2^{60}$ cycles; attack **actually carried out**.
2009 Bernstein–Lange–Peters–van Tilborg.
2009 Bernstein: post-quantum.
2009 Finiasz–Sendrier.
2010 Bernstein–Lange–Peters.
2011 May–Meurer–Thomae.
2011 Becker–Coron–Joux.
2012 Becker–Joux–May–Meurer.
2013 Bernstein–Jeffery–Lange–Meurer: post-quantum.
2015 May–Ozerov.

ng and optimizing attacks:
ange. 1981 Omura.
e–Brickell. 1988 Leon.
ouk. 1989 Stern.
umer.
ffey–Goodman.
n Tilburg. 1991 Dumer.
ffey–Goodman–Farrell.
abanne–Courteau.
abaud.
n Tilburg.
nteaut–Chabanne.
nteaut–Chabaud.
nteaut–Sendrier.

2008 Bernstein–Lange–Peters: more speedups; $\approx 2^{60}$ cycles; attack **actually carried out**.
2009 Bernstein–Lange–Peters–van Tilborg.
2009 Bernstein: post-quantum.
2009 Finiasz–Sendrier.
2010 Bernstein–Lange–Peters.
2011 May–Meurer–Thomae.
2011 Becker–Coron–Joux.
2012 Becker–Joux–May–Meurer.
2013 Bernstein–Jeffery–Lange–Meurer: post-quantum.
2015 May–Ozerov.

Modern

Easily re
a larger
$(n/2) \times$
e.g., 180

imizing attacks:
1 Omura.
1988 Leon.
Stern.

lman.
1991 Dumer.
lman–Farrell.
ourteau.

habanne.
habaud.
endrier.

2008 Bernstein–Lange–Peters: more speedups; $\approx 2^{60}$ cycles; attack **actually carried out**.
2009 Bernstein–Lange–Peters–van Tilborg.
2009 Bernstein: post-quantum.
2009 Finiasz–Sendrier.
2010 Bernstein–Lange–Peters.
2011 May–Meurer–Thomae.
2011 Becker–Coron–Joux.
2012 Becker–Joux–May–Meurer.
2013 Bernstein–Jeffery–Lange–Meurer: post-quantum.
2015 May–Ozerov.

Modern McEliece

Easily rescue syste
a larger public key
$(n/2) \times n$ matrix
e.g., 1800×3600.

tacks:

on.

mer.

ell.

2008 Bernstein–Lange–Peters: more speedups; $\approx 2^{60}$ cycles; attack **actually carried out**.

2009 Bernstein–Lange–Peters–van Tilborg.

2009 Bernstein: post-quantum.

2009 Finiasz–Sendrier.

2010 Bernstein–Lange–Peters.

2011 May–Meurer–Thomae.

2011 Becker–Coron–Joux.

2012 Becker–Joux–May–Meurer.

2013 Bernstein–Jeffery–Lange–Meurer: post-quantum.

2015 May–Ozerov.

Modern McEliece

Easily rescue system by usin
a larger public key: "random
$(n/2) \times n$ matrix K over $\mathbf{F}_2$
e.g., 1800×3600.

2008 Bernstein–Lange–Peters: more speedups; $\approx 2^{60}$ cycles; attack **actually carried out**.

2009 Bernstein–Lange–Peters–van Tilborg.

2009 Bernstein: post-quantum.

2009 Finiasz–Sendrier.

2010 Bernstein–Lange–Peters.

2011 May–Meurer–Thomae.

2011 Becker–Coron–Joux.

2012 Becker–Joux–May–Meurer.

2013 Bernstein–Jeffery–Lange–Meurer: post-quantum.

2015 May–Ozerov.

Modern McEliece

Easily rescue system by using a larger public key: "random" $(n/2) \times n$ matrix K over $\mathbf{F}_2$. e.g., 1800×3600.

2008 Bernstein–Lange–Peters: more speedups; $\approx 2^{60}$ cycles; attack **actually carried out**.
2009 Bernstein–Lange–Peters–van Tilborg.
2009 Bernstein: post-quantum.
2009 Finiasz–Sendrier.
2010 Bernstein–Lange–Peters.
2011 May–Meurer–Thomae.
2011 Becker–Coron–Joux.
2012 Becker–Joux–May–Meurer.
2013 Bernstein–Jeffery–Lange–Meurer: post-quantum.
2015 May–Ozerov.

Modern McEliece

Easily rescue system by using a larger public key: "random" $(n/2) \times n$ matrix K over $\mathbf{F}_2$. e.g., 1800×3600.

Larger weight $w \approx n/(2 \lg n)$. e.g. $e \in \mathbf{F}_2^{3600}$ of weight 150.

2008 Bernstein–Lange–Peters: more speedups; $\approx 2^{60}$ cycles; attack **actually carried out**.

2009 Bernstein–Lange–Peters–van Tilborg.

2009 Bernstein: post-quantum.

2009 Finiasz–Sendrier.

2010 Bernstein–Lange–Peters.

2011 May–Meurer–Thomae.

2011 Becker–Coron–Joux.

2012 Becker–Joux–May–Meurer.

2013 Bernstein–Jeffery–Lange–Meurer: post-quantum.

2015 May–Ozerov.

Modern McEliece

Easily rescue system by using a larger public key: "random" $(n/2) \times n$ matrix K over $\mathbf{F}_2$. e.g., 1800×3600.

Larger weight $w \approx n/(2 \lg n)$. e.g. $e \in \mathbf{F}_2^{3600}$ of weight 150.

1962 attack cost: $2^{(1+o(1))w}$.

2008 Bernstein–Lange–Peters: more speedups; $\approx 2^{60}$ cycles; attack **actually carried out**.

2009 Bernstein–Lange–Peters–van Tilborg.

2009 Bernstein: post-quantum.

2009 Finiasz–Sendrier.

2010 Bernstein–Lange–Peters.

2011 May–Meurer–Thomae.

2011 Becker–Coron–Joux.

2012 Becker–Joux–May–Meurer.

2013 Bernstein–Jeffery–Lange–Meurer: post-quantum.

2015 May–Ozerov.

Modern McEliece

Easily rescue system by using a larger public key: "random" $(n/2) \times n$ matrix K over $\mathbf{F}_2$. e.g., 1800×3600.

Larger weight $w \approx n/(2 \lg n)$. e.g. $e \in \mathbf{F}_2^{3600}$ of weight 150.

1962 attack cost: $2^{(1+o(1))w}$.

After extensive research, 2015 attack cost: $2^{(1+o(1))w}$.

2008 Bernstein–Lange–Peters: more speedups; $\approx 2^{60}$ cycles; attack **actually carried out**.

2009 Bernstein–Lange–Peters–van Tilborg.

2009 Bernstein: post-quantum.

2009 Finiasz–Sendrier.

2010 Bernstein–Lange–Peters.

2011 May–Meurer–Thomae.

2011 Becker–Coron–Joux.

2012 Becker–Joux–May–Meurer.

2013 Bernstein–Jeffery–Lange–Meurer: post-quantum.

2015 May–Ozerov.

Modern McEliece

Easily rescue system by using a larger public key: "random" $(n/2) \times n$ matrix K over $\mathbf{F}_2$. e.g., 1800×3600.

Larger weight $w \approx n/(2 \lg n)$. e.g. $e \in \mathbf{F}_2^{3600}$ of weight 150.

1962 attack cost: $2^{(1+o(1))w}$.

After extensive research, 2015 attack cost: $2^{(1+o(1))w}$.

Post-quantum: $2^{(0.5+o(1))w}$. e.g. $\approx 2^{26}$ Grover iterations to search 2^{53} choices of S.