20th ANNUAL HIGH CONFIDENCE SOFTWARE AND SYSTEMS CONFERENCE

A Verified Optimizer for Quantum Circuits

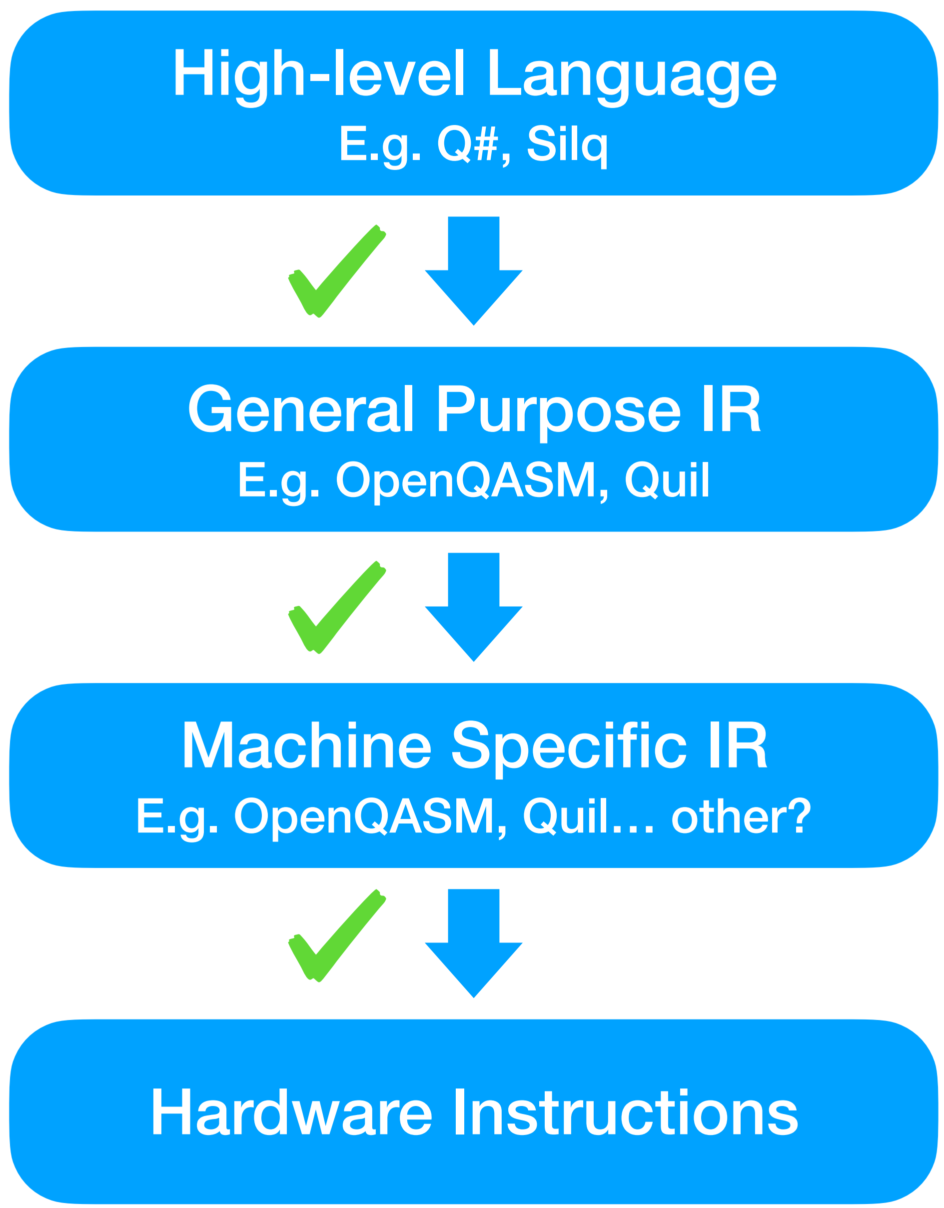
Kesha Hietala, ***Robert Rand****, Shih-Han Hung,
Xiaodi Wu and Michael Hicks

*University of Chicago

HCSS

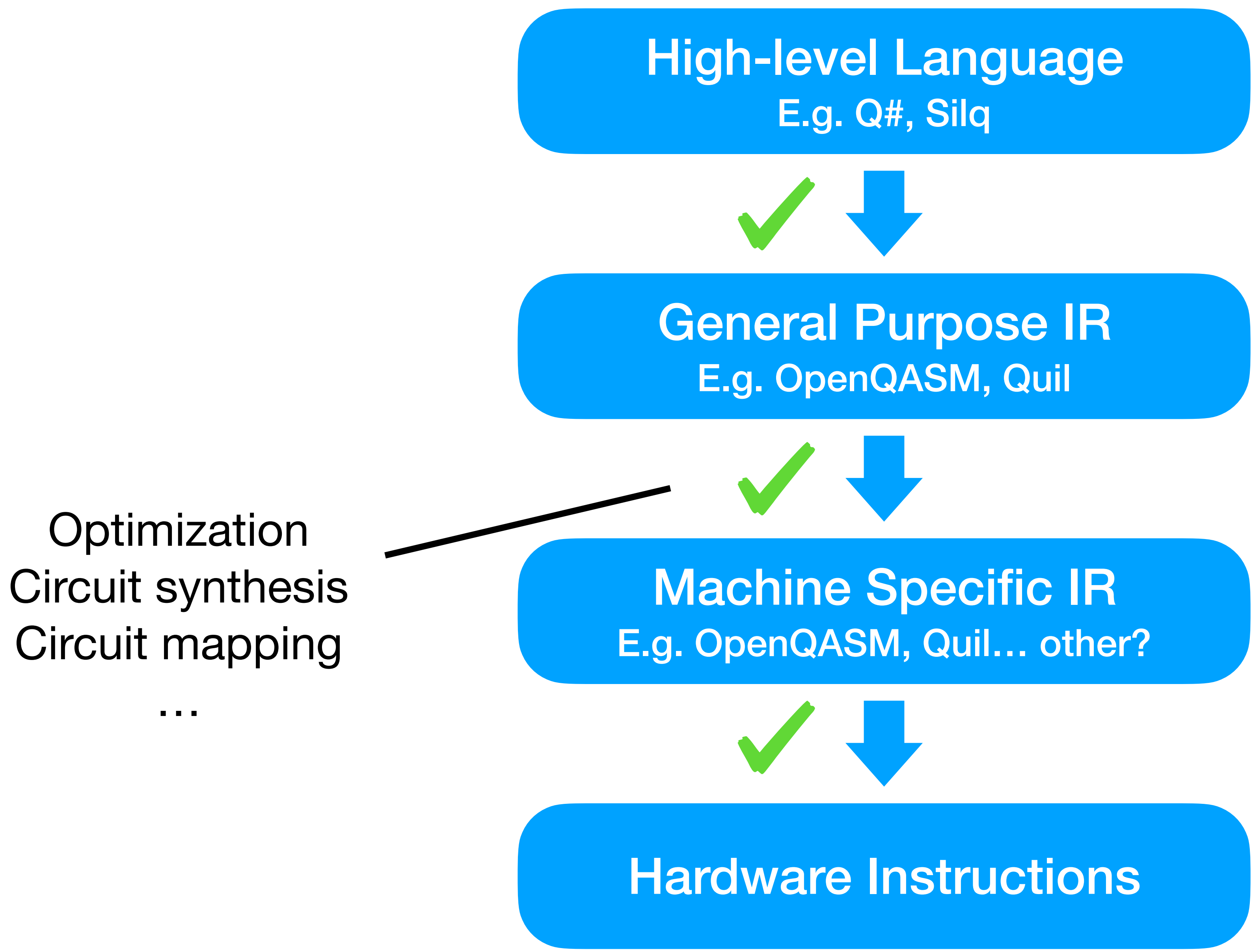
Verified compiler stack

- End goal: *verified compiler stack* for quantum programs



Verified compiler stack

- End goal: *verified compiler stack* for quantum programs



Verified compiler stack

- End goal: *verified compiler stack* for quantum programs

Means that we've *formally verified* that the transformation is semantics-preserving.

High-level Language
E.g. Q#, Silq

↓

General Purpose IR
E.g. OpenQASM, Quil

↓

Optimization
Circuit synthesis
Circuit mapping
…

Machine Specific IR
E.g. OpenQASM, Quil… other?

↓

Hardware Instructions

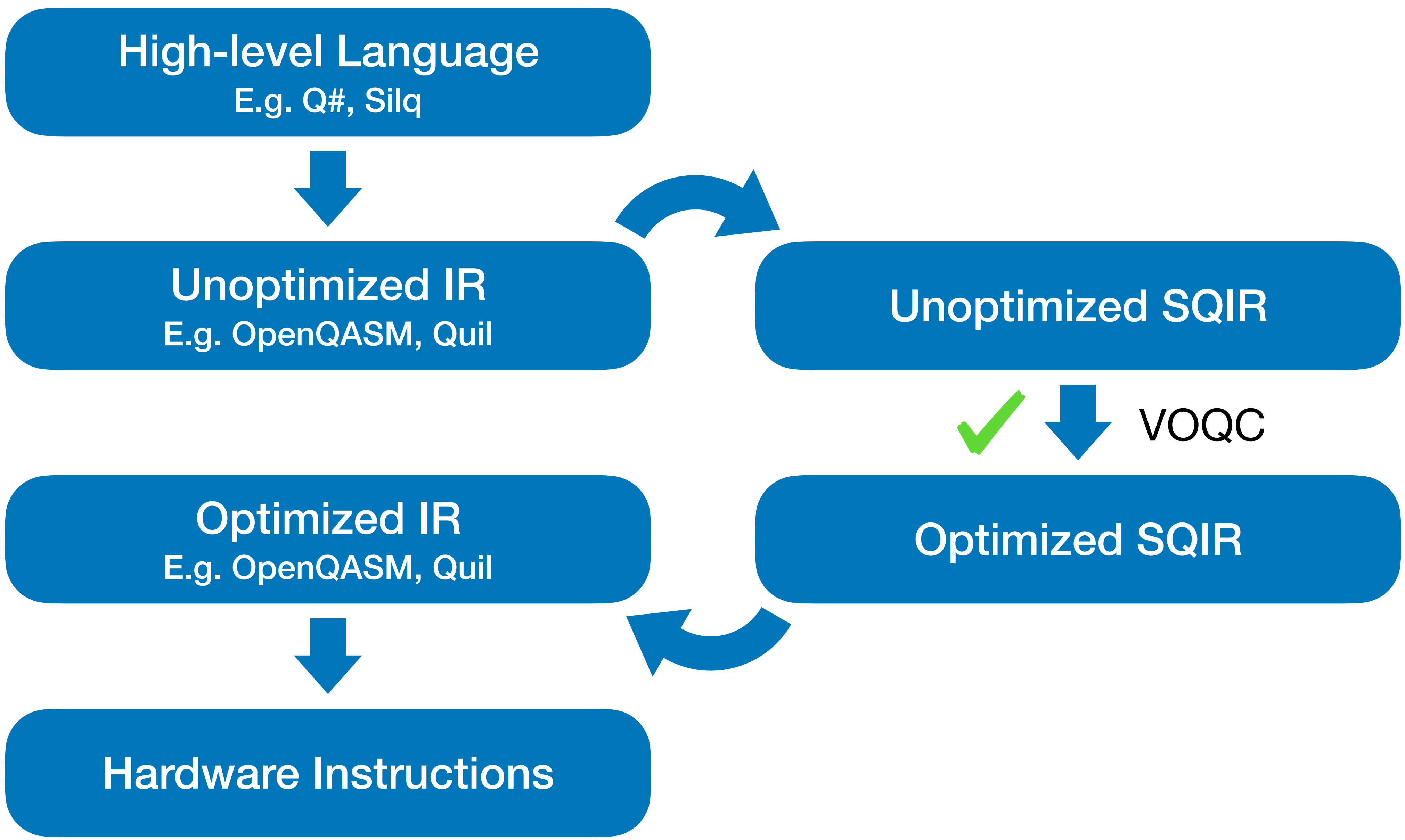
Small Quantum IR

- Intermediate goal: An IR suitable for representing and reasoning about quantum programs

$$
\begin{aligned}
U &:= U_1;\ U_2 \mid G\ q \mid G\ q_1\ q_2 \\
P &:= \texttt{skip} \mid P_1;\ P_2 \mid U \mid \texttt{meas}\ q\ P_1\ P_2
\end{aligned}
$$

Verified Compiler Stack

- End goal: *verified compiler stack* for quantum programs



Why is quantum hard?

Why is quantum hard?

- *Superposition*: Every quantum bit (or *qubit*) can be in a combination of 0 and 1 states simultaneously.

Why is quantum hard?

- *Superposition*: Every quantum bit (or *qubit*) can be in a combination of 0 and 1 states simultaneously.
- *Entanglement*: Qubits can be bound to one another in such a way that operating on qubit **x** influence qubit **y**.

Why is quantum hard?

- *Superposition*: Every quantum bit (or *qubit*) can be in a combination of 0 and 1 states simultaneously.
- *Entanglement*: Qubits can be bound to one another in such a way that operating on qubit **x** influence qubit **y**.
- *Measurement*: Inspecting a qubit induces a *probabilistic transition* while perturbing the system.

Why is quantum hard?

- *Superposition*: Every quantum bit (or *qubit*) can be in a combination of 0 and 1 states simultaneously.
- *Entanglement*: Qubits can be bound to one another in such a way that operating on qubit **x** influence qubit **y**.
- ~~*Measurement*: Inspecting a qubit induces a *probabilistic transition* while perturbing the system.~~

Why is quantum hard?

- *Superposition*: Every quantum bit (or *qubit*) can be in a combination of 0 and 1 states simultaneously.

- *Entanglement*: Qubits can be bound to one another in such a way that operating on qubit **x** influence qubit **y**.

- ~~*Measurement*: Inspecting a qubit induces a *probabilistic transition* while perturbing the system~~.

Deferred.

Why is quantum hard? (operationally)

Why is quantum hard? (operationally)

- A n-qubit quantum state is represented as a length 2^n vector of complex numbers.

Why is quantum hard? (operationally)

- A n-qubit quantum state is represented as a length 2^n vector of complex numbers.

- Every operation on the quantum state can alter the entire vector.

Qubits

Qubits

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Qubits

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Qubits

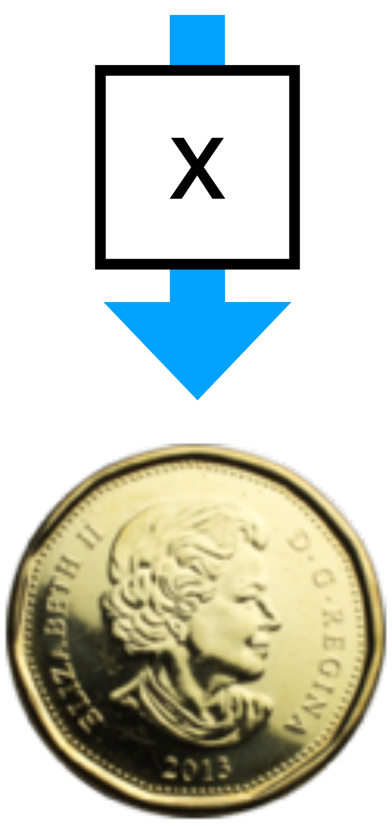
$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Qubits

$$\begin{pmatrix}1\\0\end{pmatrix} \qquad \begin{pmatrix}0\\1\end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$$

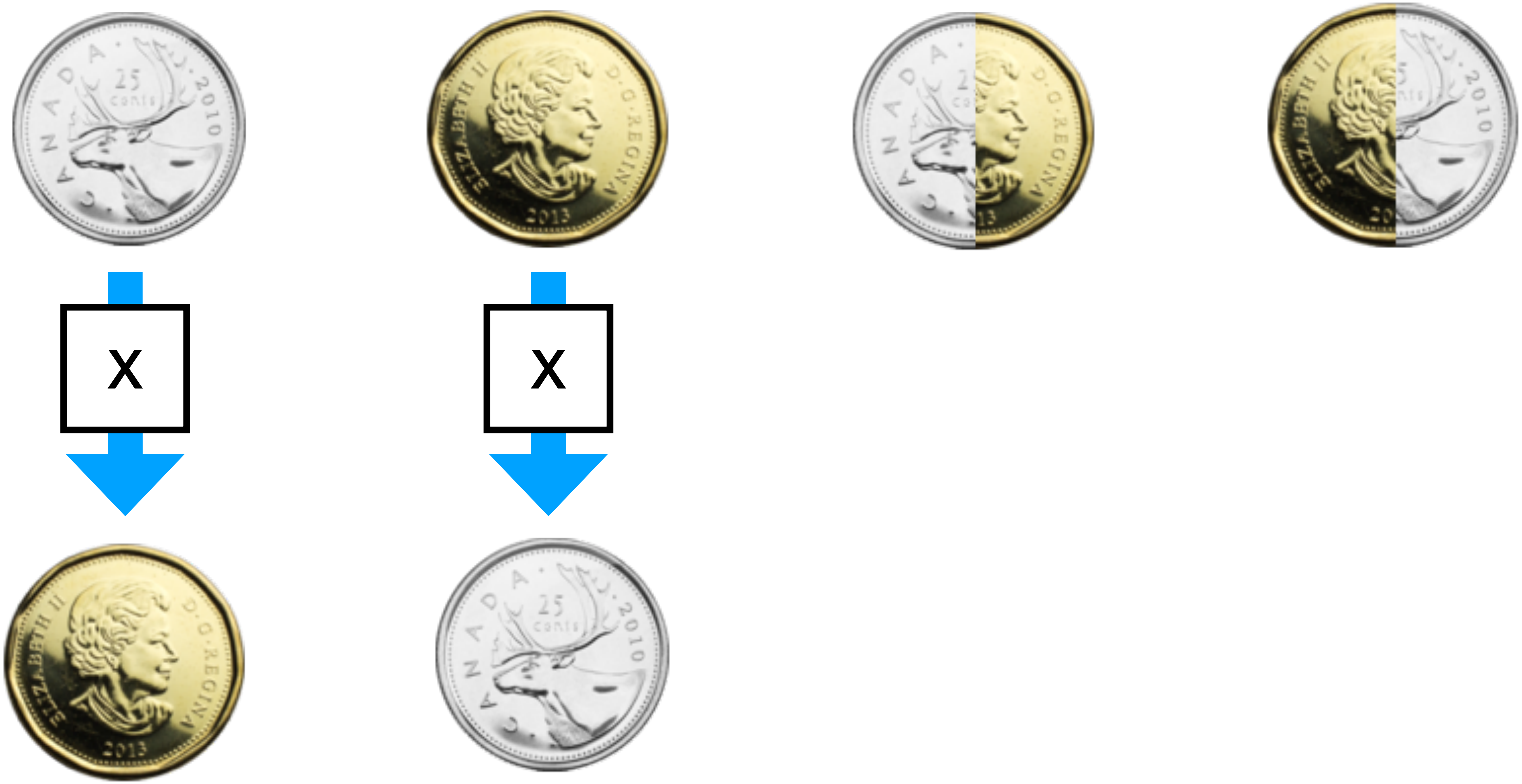

Qubits

$$\begin{pmatrix}1\\0\end{pmatrix} \qquad \begin{pmatrix}0\\1\end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$$



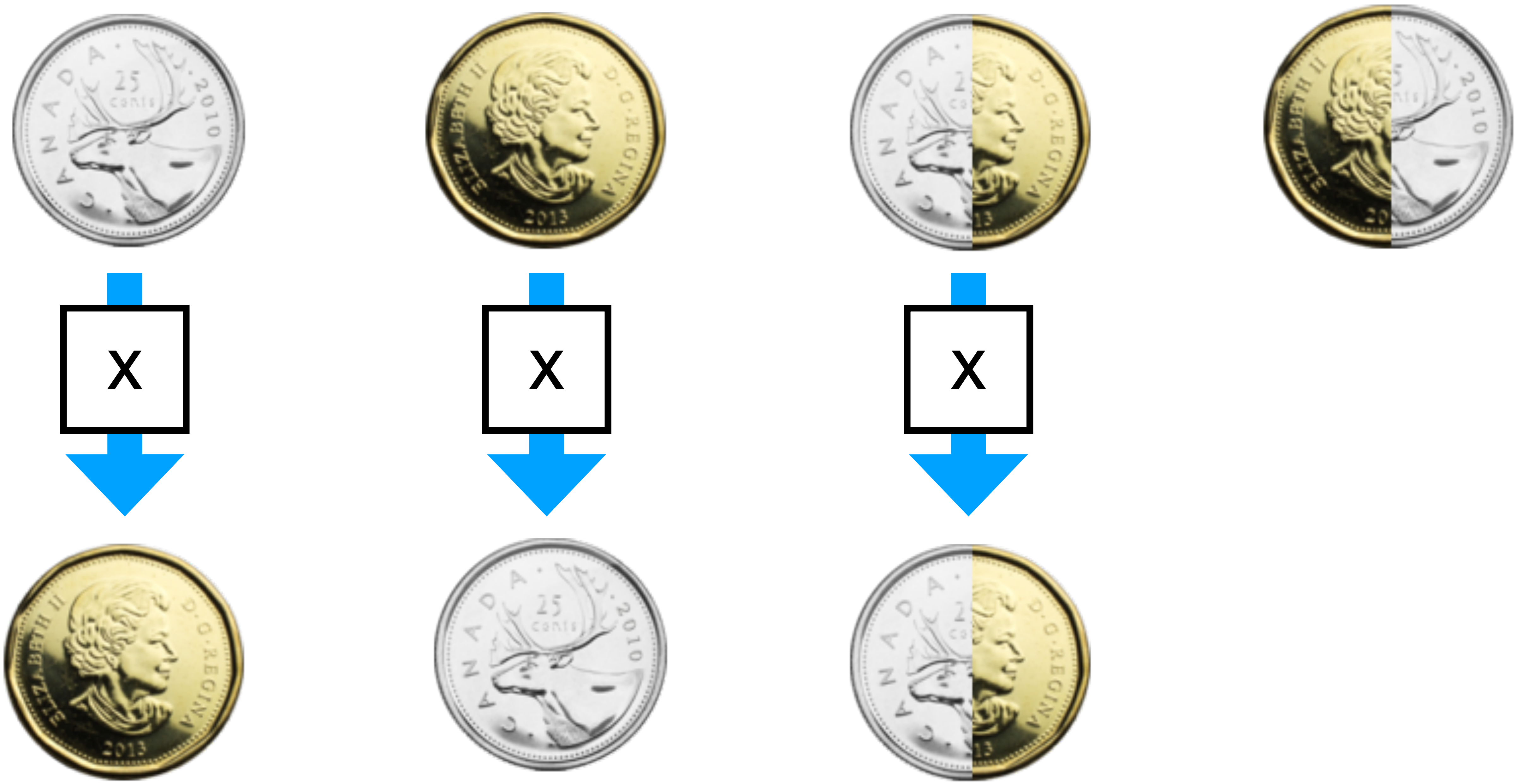
Qubits

$$\begin{pmatrix}1\\0\end{pmatrix} \qquad \begin{pmatrix}0\\1\end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$$



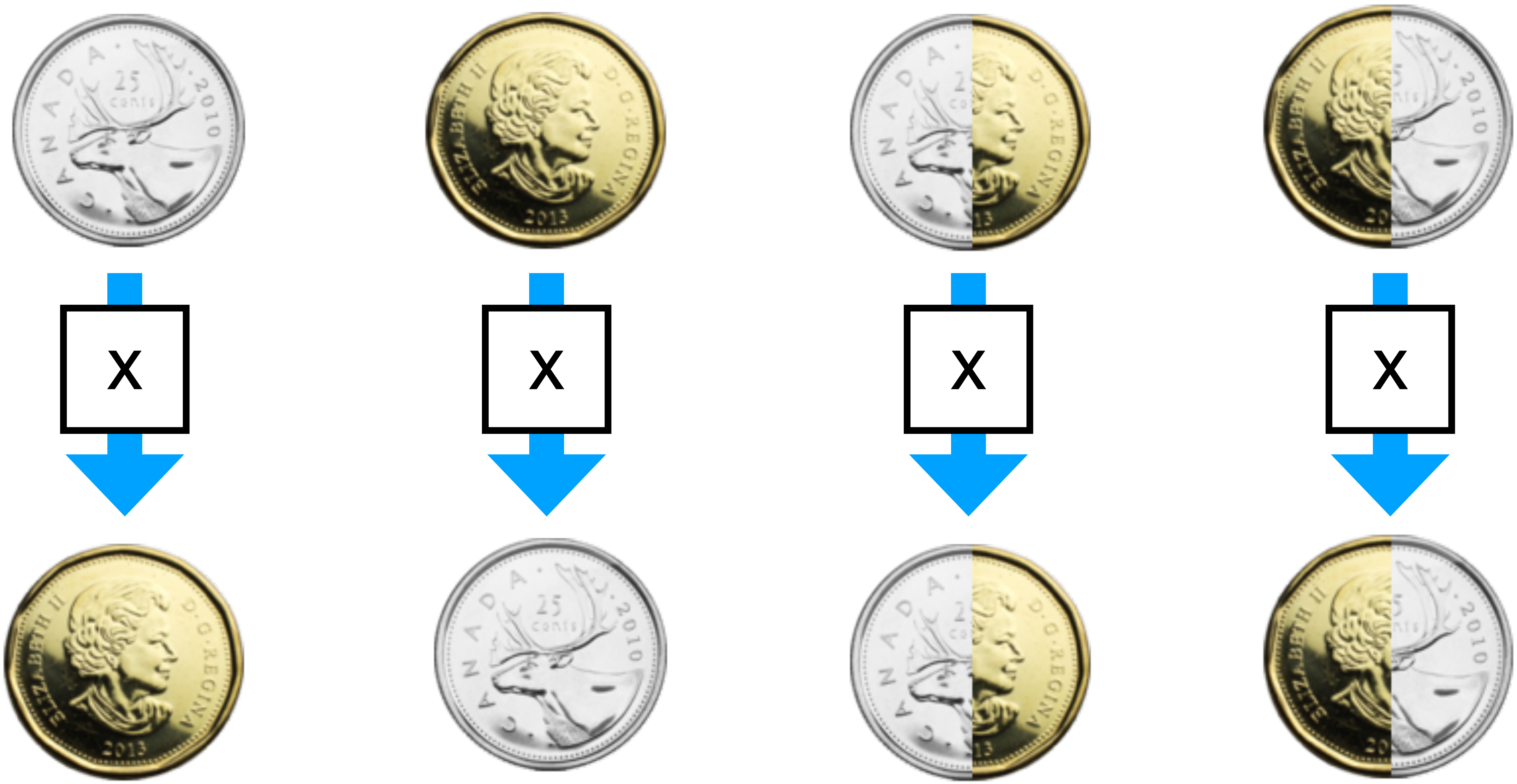
Qubits

$$\begin{pmatrix}1\\0\end{pmatrix} \qquad \begin{pmatrix}0\\1\end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$$



Qubits

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$$



Transformations

$$\begin{pmatrix}1\\0\end{pmatrix}$$

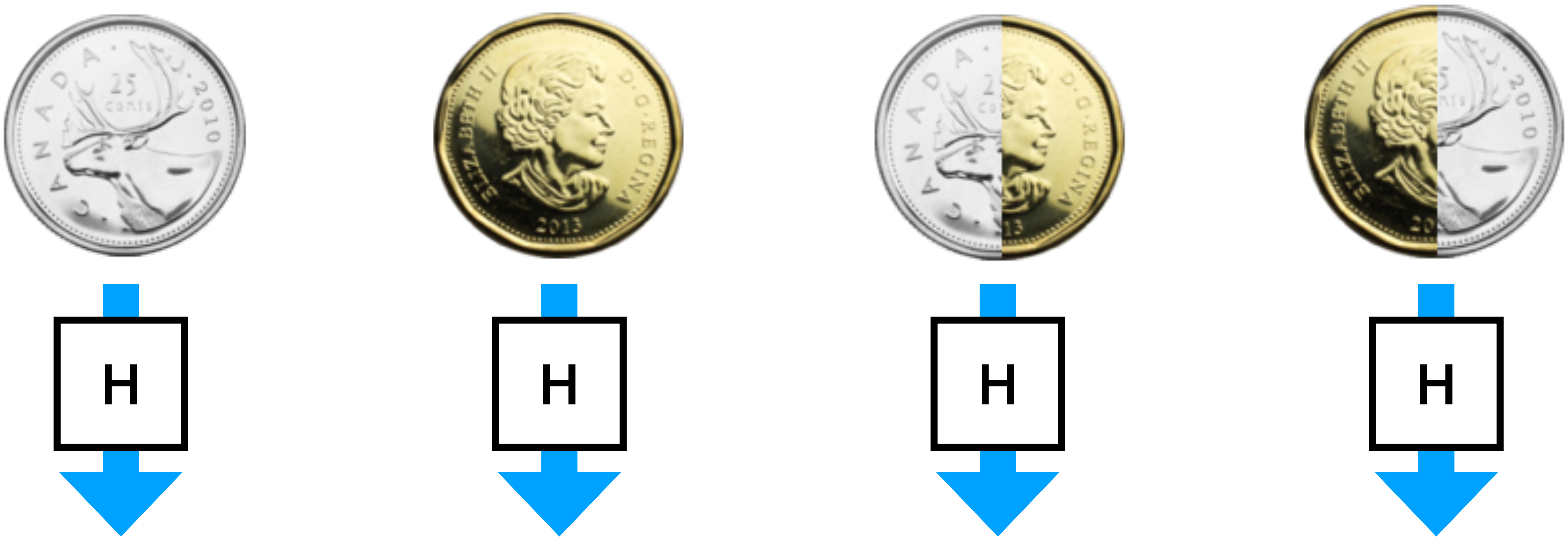
$$\begin{pmatrix}0\\1\end{pmatrix}$$

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$$

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$$

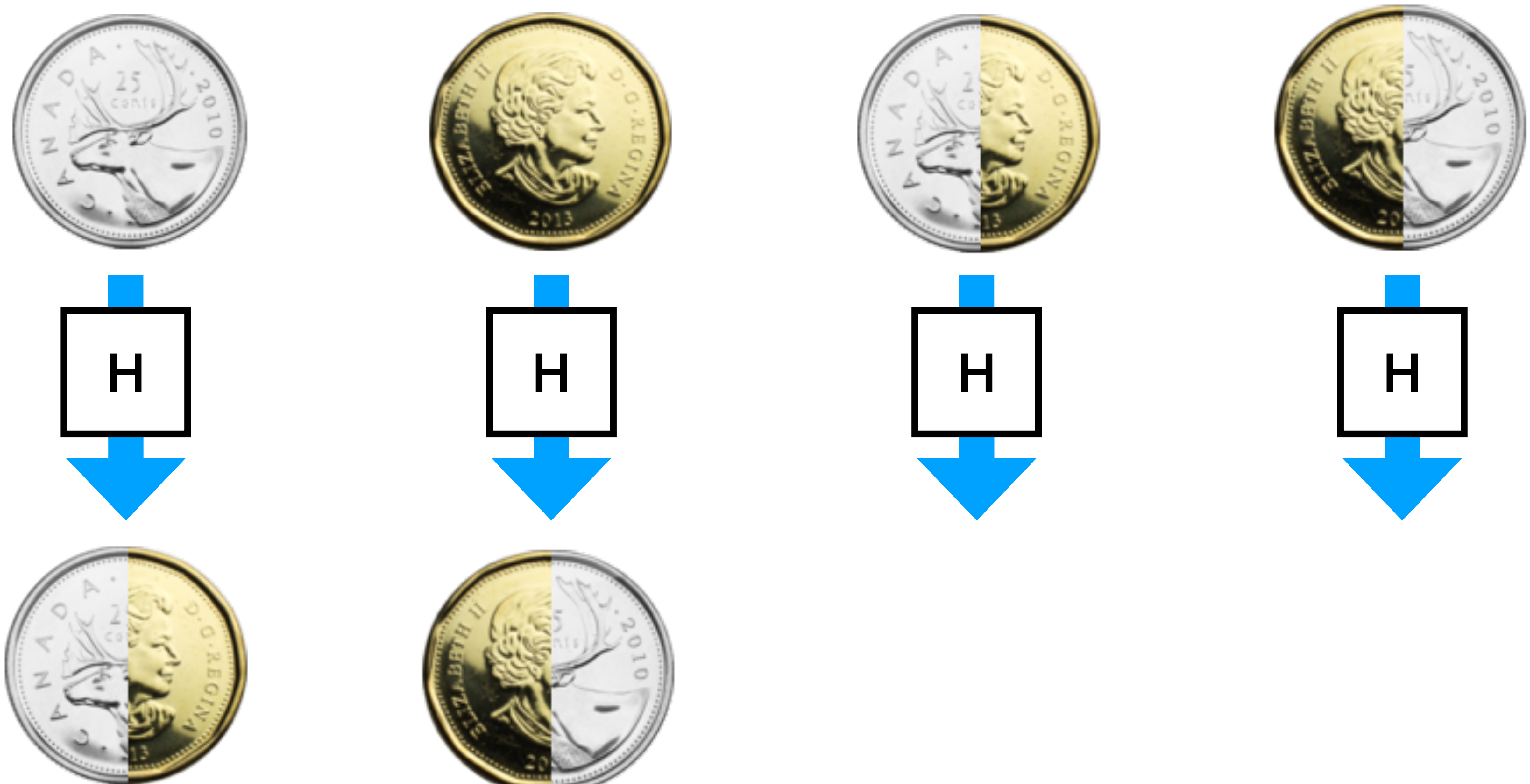
Transformations

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$$



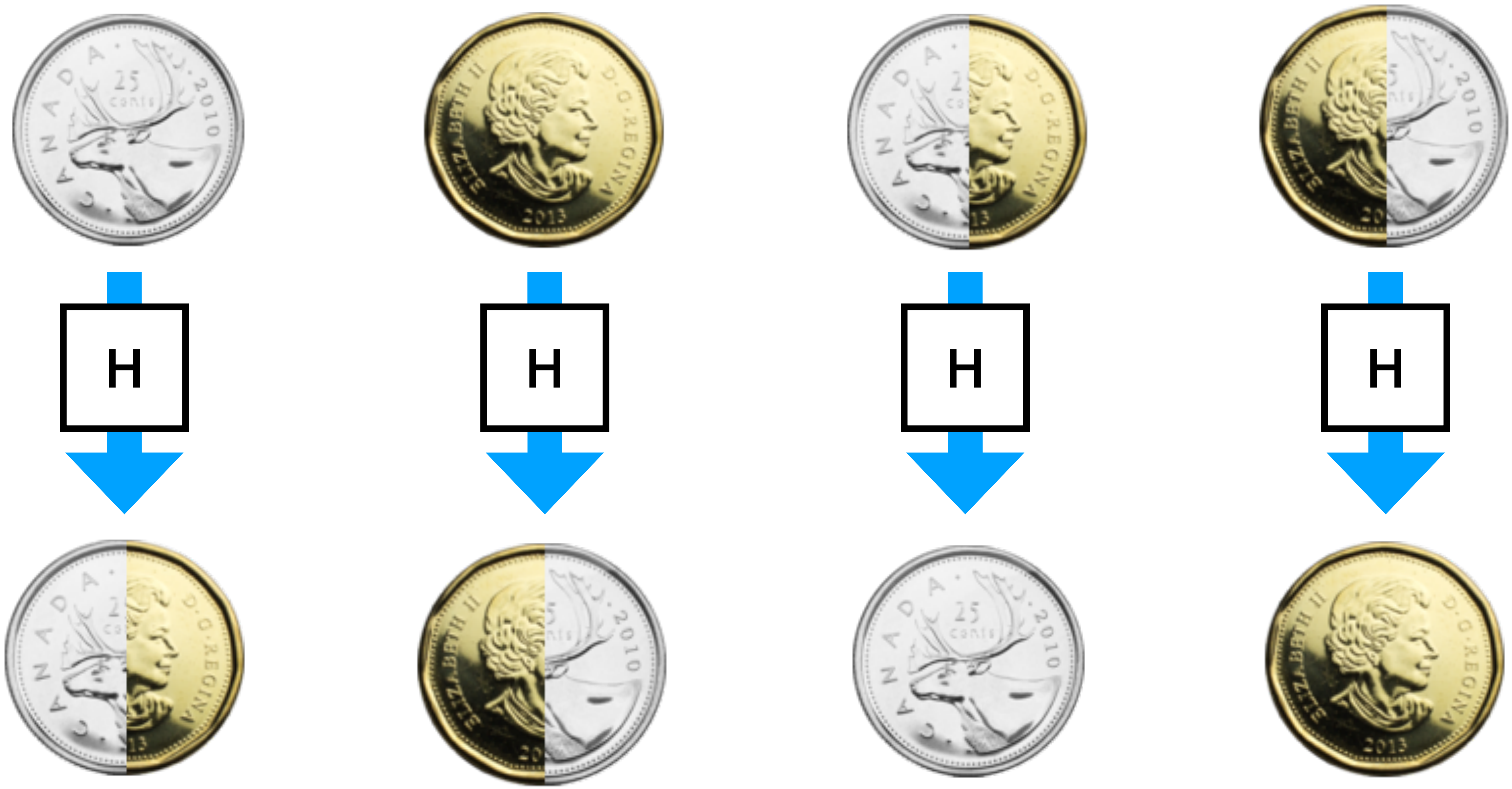
Transformations

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$$



Transformations

$$\begin{pmatrix}1\\0\end{pmatrix} \qquad \begin{pmatrix}0\\1\end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$$



Entanglement

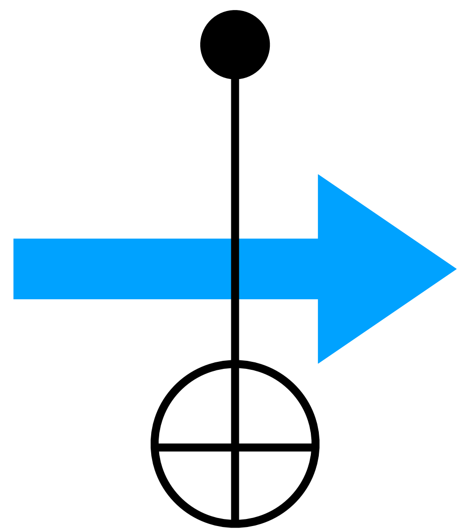
$$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$$

$$\begin{pmatrix}1\\0\end{pmatrix}$$

Entanglement

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$$

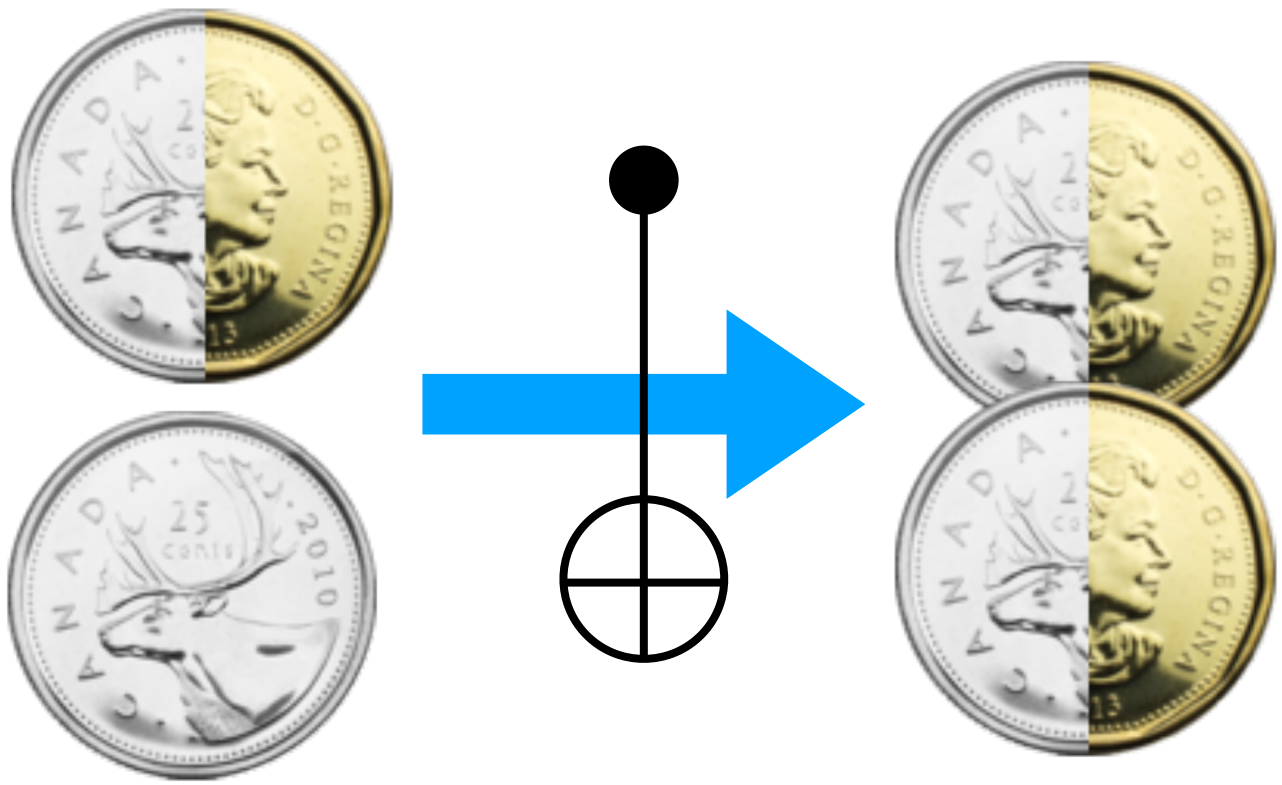
$$\begin{pmatrix}1\\0\end{pmatrix}$$



Entanglement

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$$

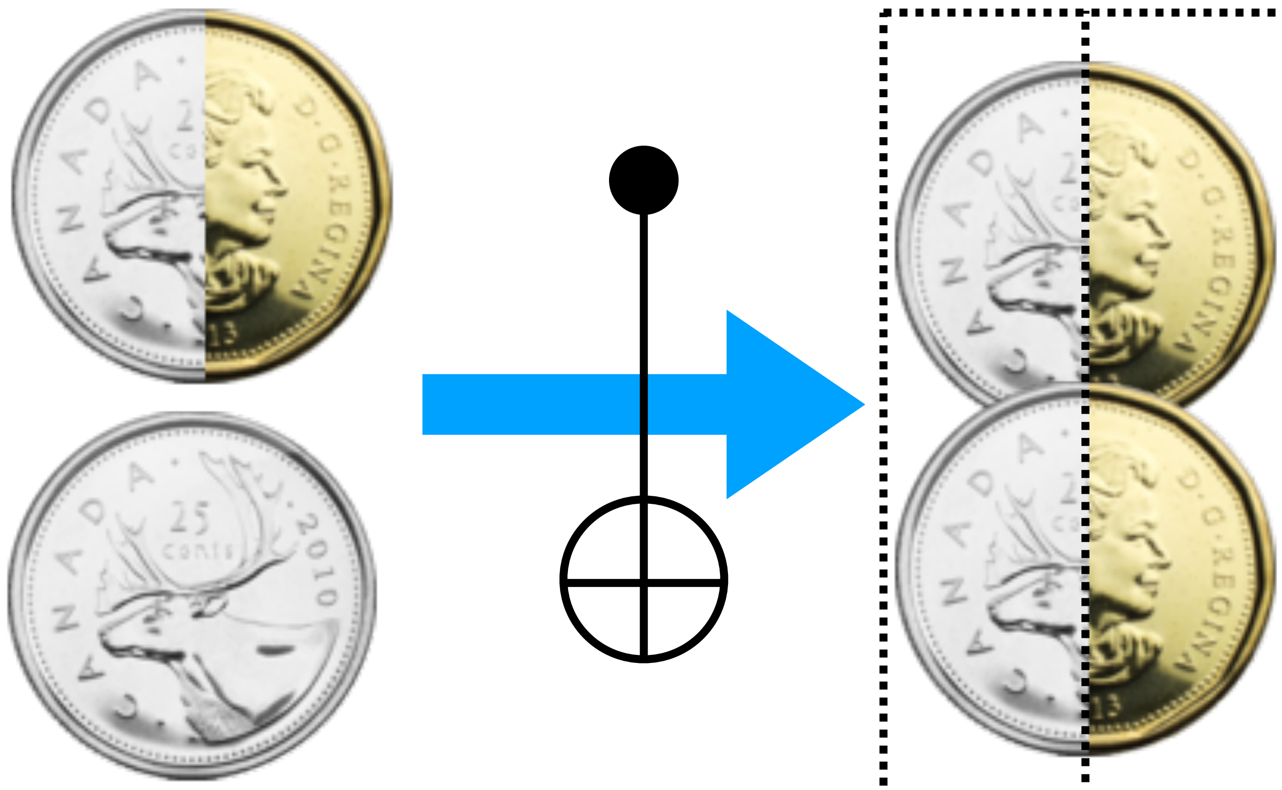
$$\begin{pmatrix}1\\0\end{pmatrix}$$



Entanglement

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$$

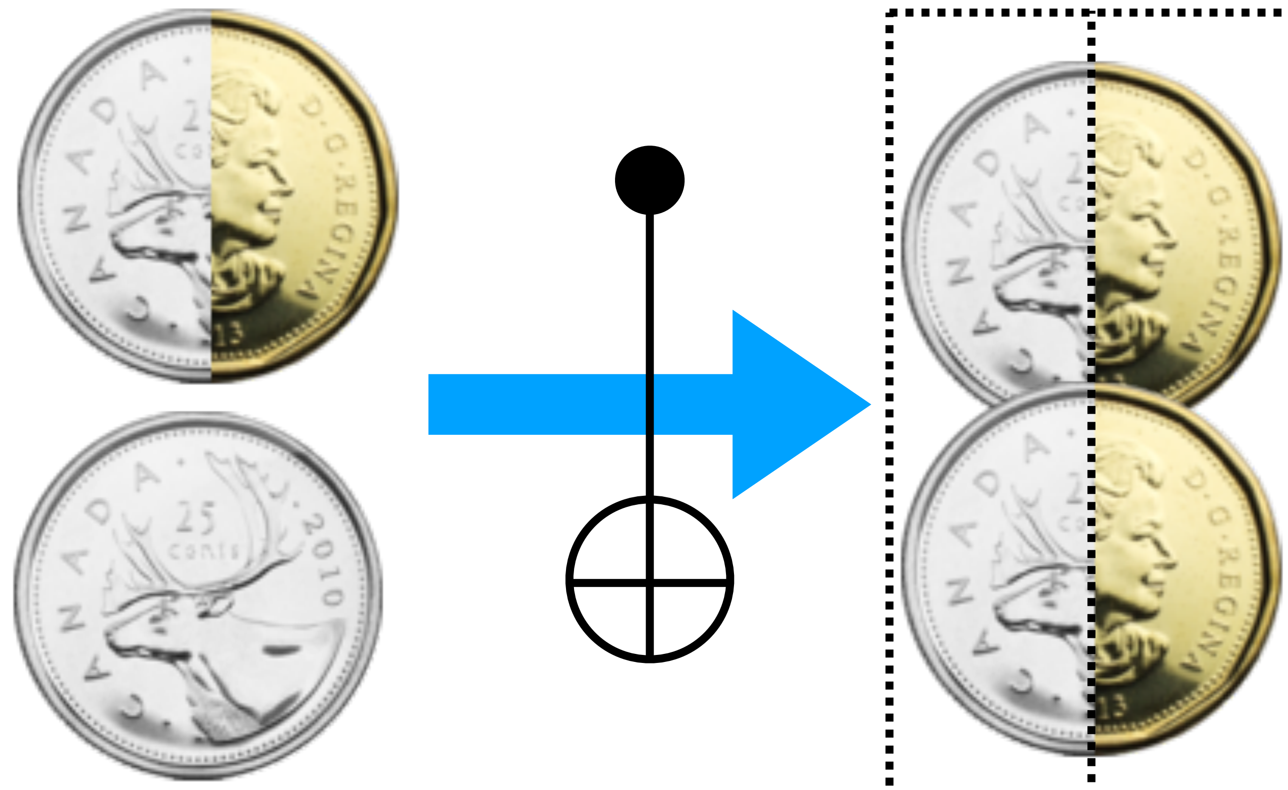
$$\begin{pmatrix}1\\0\end{pmatrix}$$



Entanglement

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$$

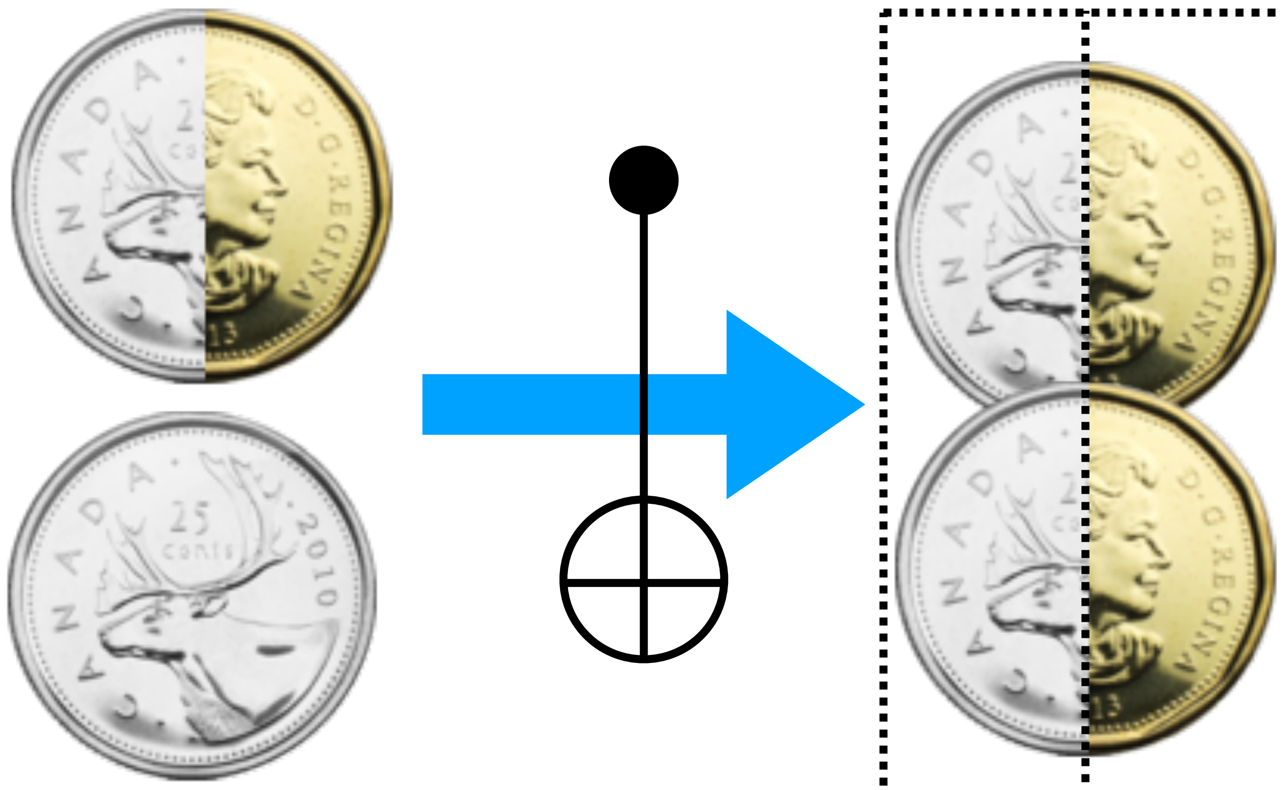
$$\begin{pmatrix}1\\0\end{pmatrix}$$



Bell pair

Entanglement

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} \qquad \begin{pmatrix}1\\0\end{pmatrix}$$



$$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\0\\0\\1\end{pmatrix}$$

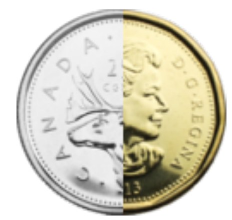
Bell pair

Linear Algebra

Linear Algebra

 $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$

 $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$

 $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$

 $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$

Linear Algebra

$$\begin{pmatrix}1\\0\end{pmatrix}$$

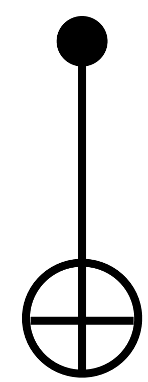
$$\begin{pmatrix}0\\1\end{pmatrix}$$

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$$

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$$

$$\boxed{X} \quad \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$$

$$\boxed{H} \quad \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}$$

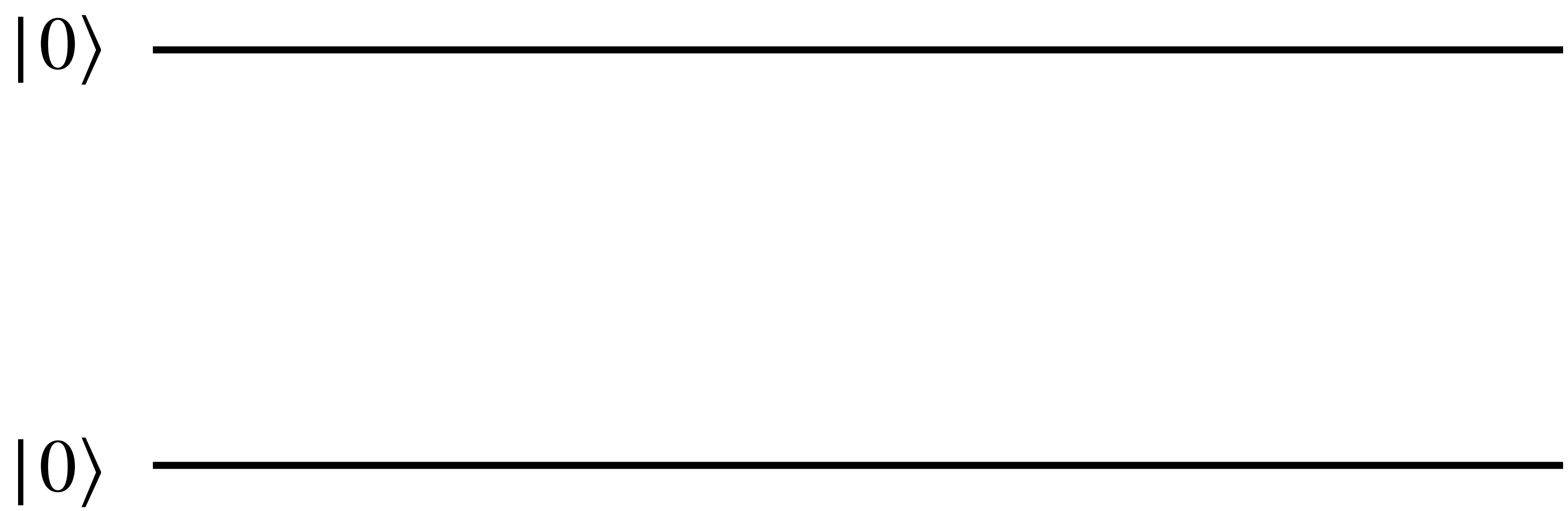


$$\begin{pmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{pmatrix}$$

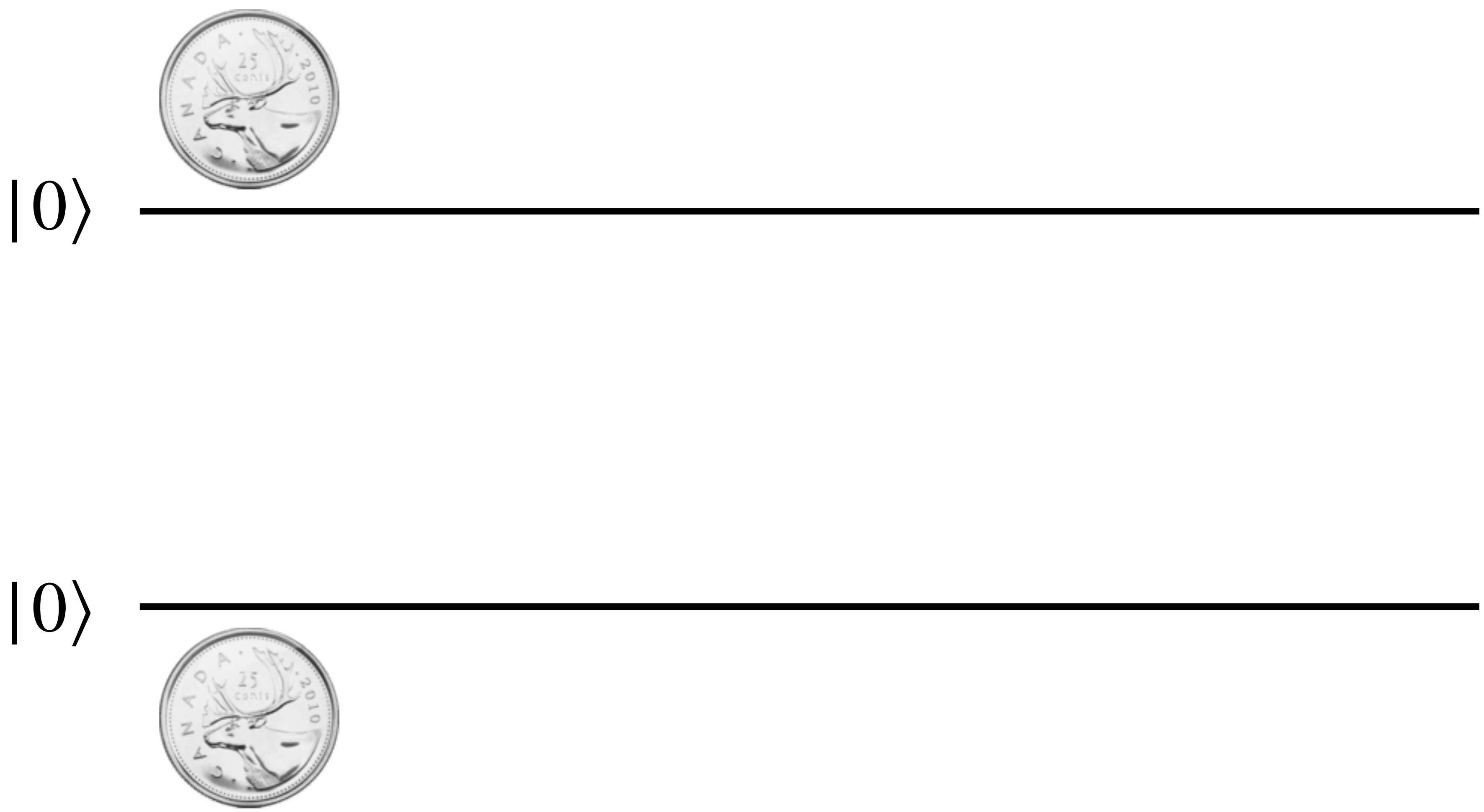
Circuits

Circuits

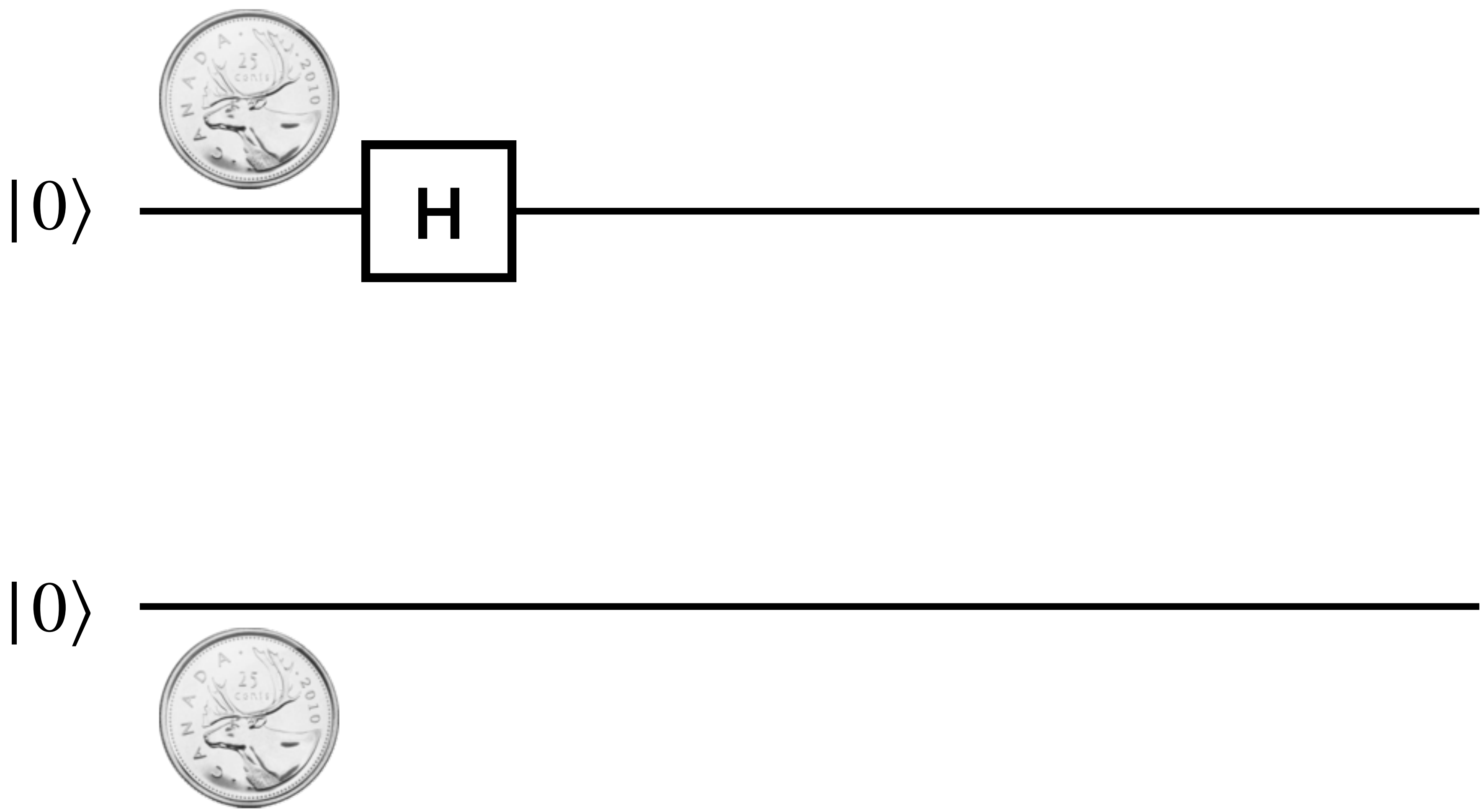
Circuits



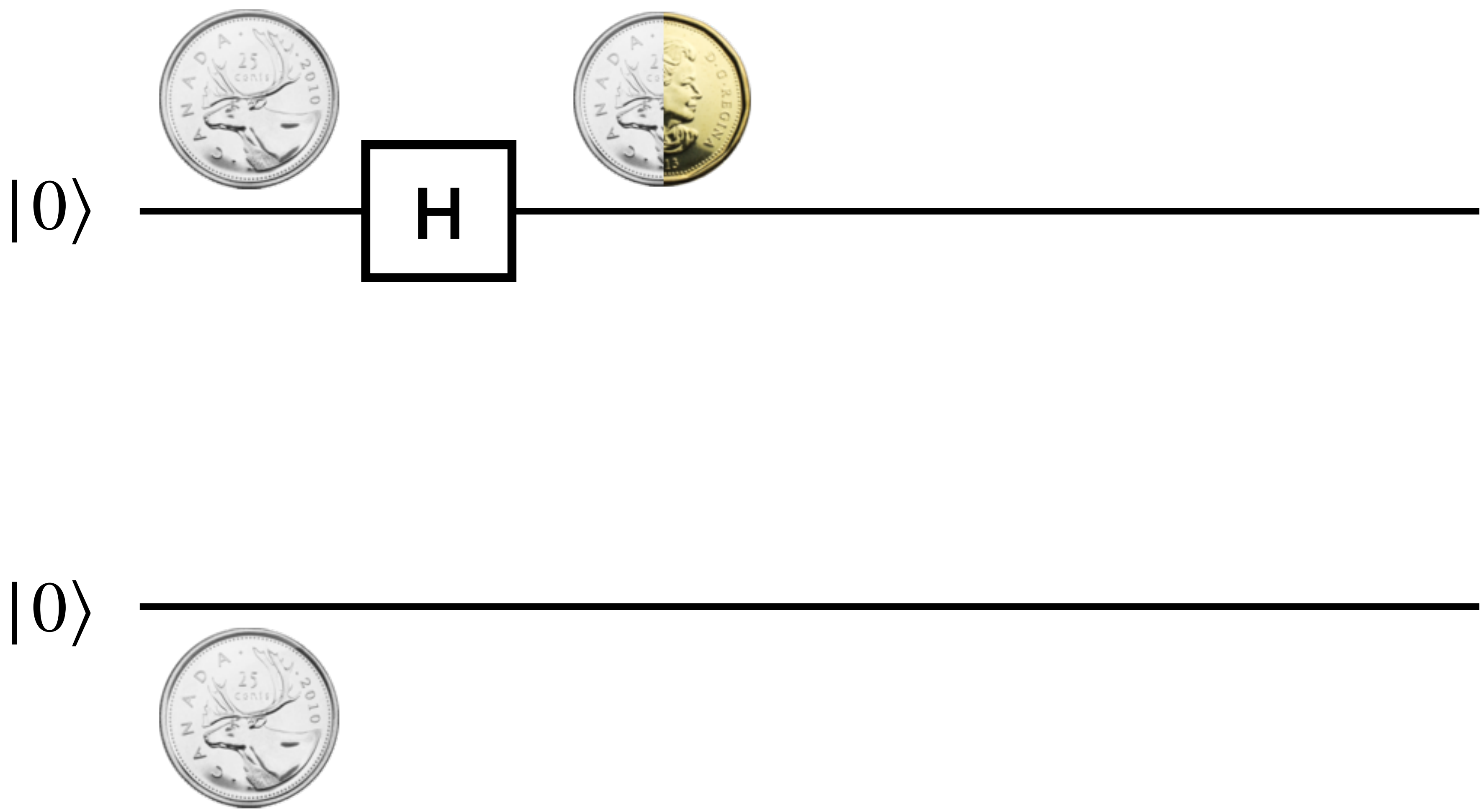
Circuits



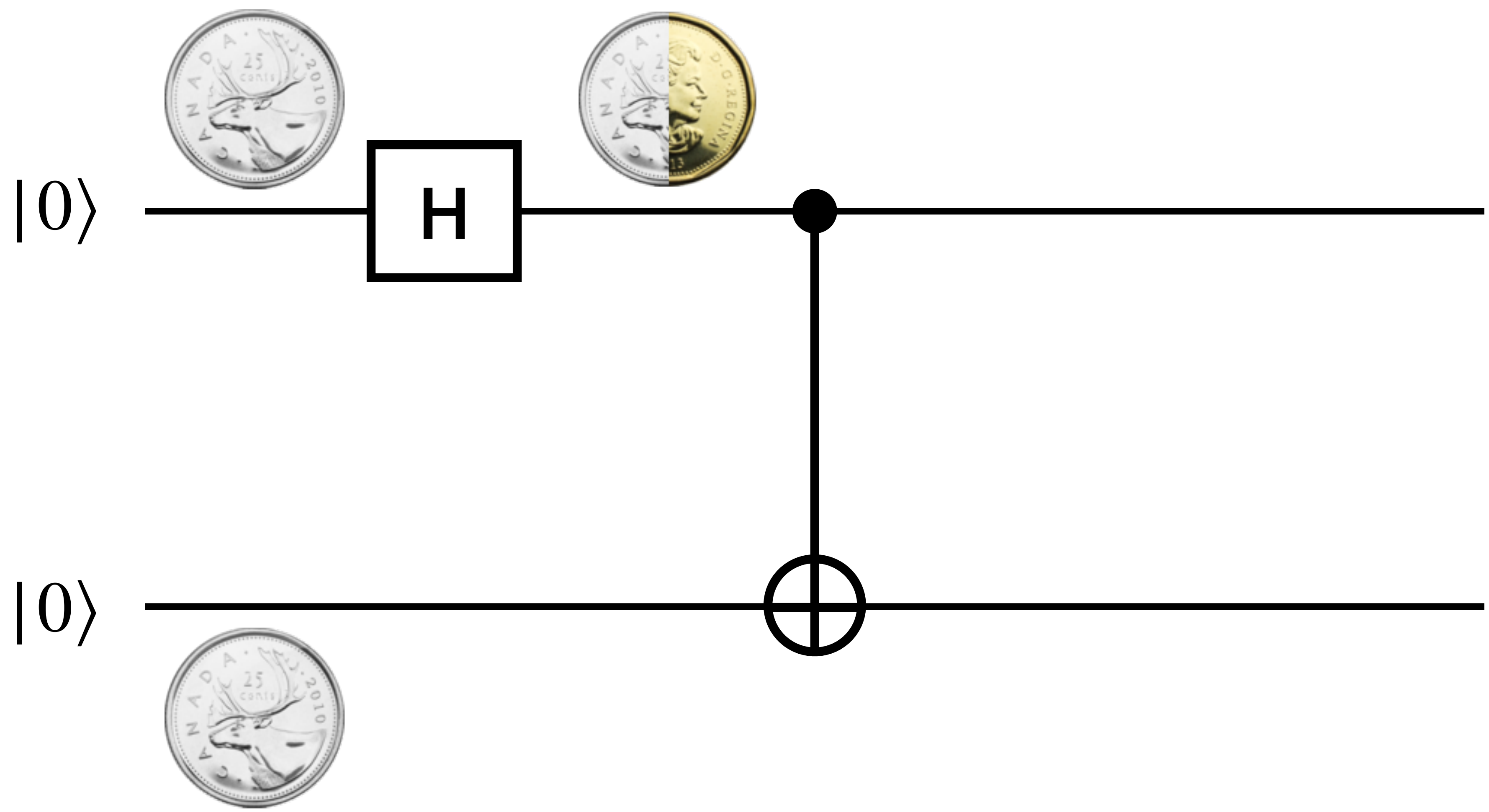
Circuits



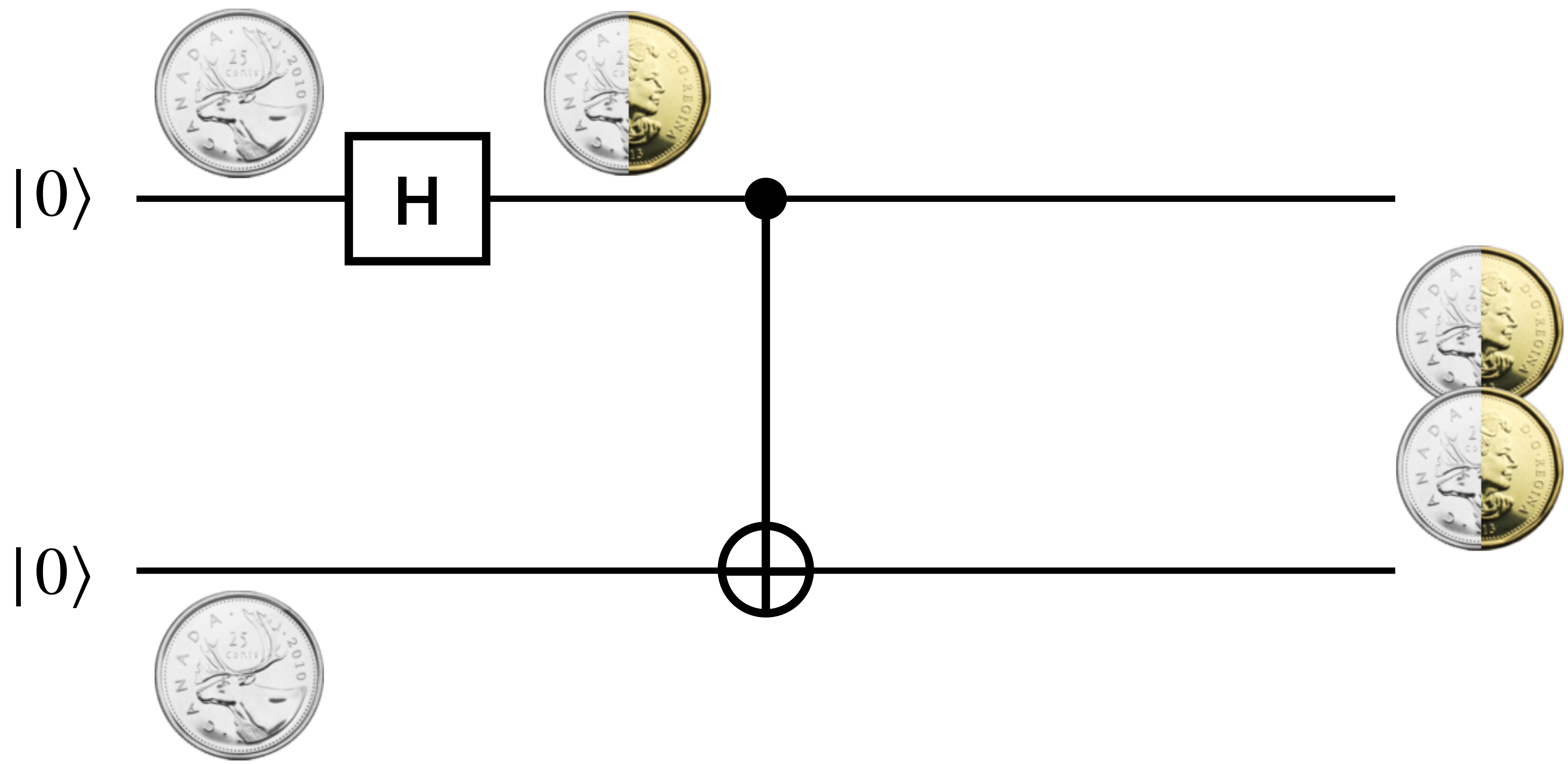
Circuits



Circuits



Circuits



SQIR

- Syntax

$$U \;:=\; U_1;\ U_2 \mid G\ q \mid G\ q_1\ q_2$$
$$P \;:=\; \texttt{skip} \mid P_1;\ P_2 \mid U \mid \texttt{meas}\ q\ P_1\ P_2$$

- Semantics assumes a **global register** of size *d*
  - A unitary program corresponds to a unitary matrix of size $2^d \times 2^d$
  - A non-unitary program corresponds to a function between density matrices of size $2^d \times 2^d$

SQIR

- Syntax

$$U \ := U_1;\ U_2 \mid G\ q \mid G\ q_1\ q_2$$

$$\cancel{P \ := \texttt{skip} \mid P_1;\ P_2 \mid U \mid \texttt{meas}\ q\ P_1\ P_2}$$

- Semantics assumes a **global register** of size *d*
  - A unitary program corresponds to a unitary matrix of size $2^d \times 2^d$
  - ~~A non-unitary program corresponds to a function between density matrices of size~~ $\cancel{2^d \times 2^d}$

Semantics

Unitary program → unitary matrix

$$[\![X]\!] = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad [\![H]\!] = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$[\![U_1;\ U_2]\!]_d = [\![U_2]\!]_d \times [\![U_1]\!]_d$$

$$[\![G_1\ q]\!]_d = \begin{cases} apply_1(G_1, q, d) & \text{well-typed} \\ 0_{2^d} & \text{otherwise} \end{cases}$$

$$[\![G_1\ q_1\ q_2]\!]_d = \begin{cases} apply_2(G_1, q_1, q_2, d) & \text{well-typed} \\ 0_{2^d} & \text{otherwise} \end{cases}$$

Semantics

Unitary program → unitary matrix

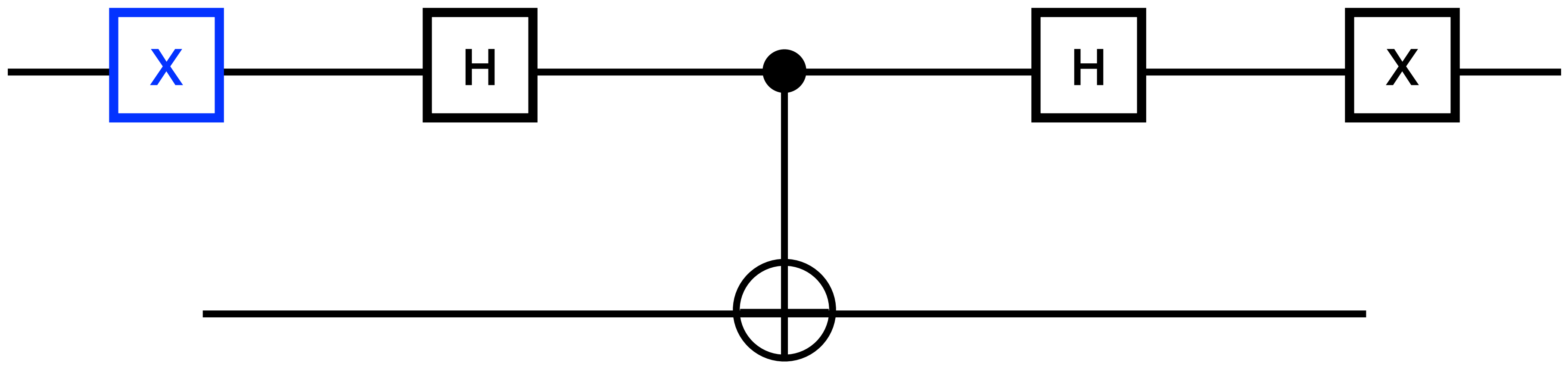
$$[\![X]\!] = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad [\![H]\!] = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$[\![U_1;\ U_2]\!]_d = \ [\![U_2]\!]_d \times [\![U_1]\!]_d$$

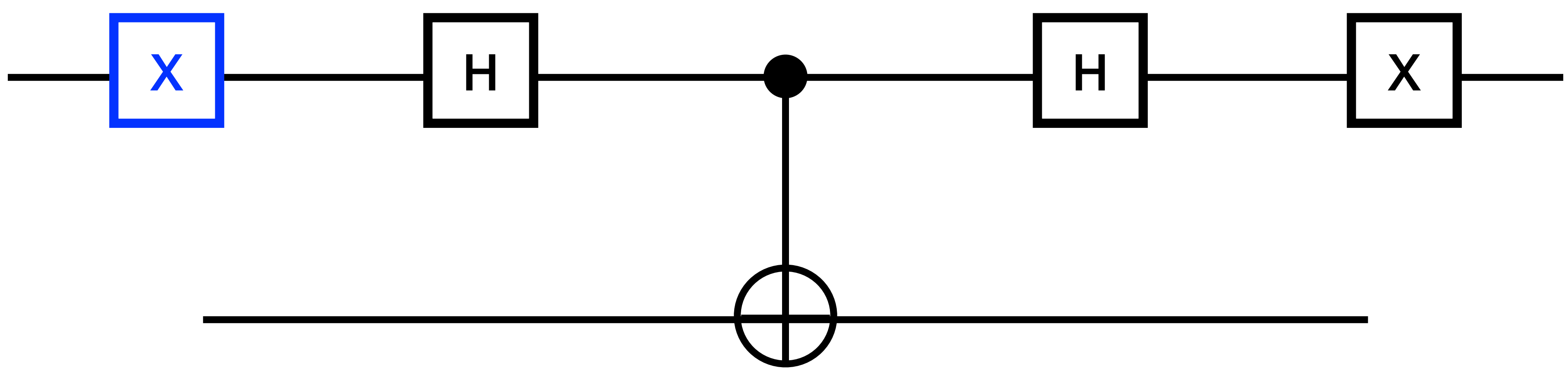
$$[\![H \quad 1]\!]_4 = \begin{cases} I_2 \otimes H \otimes I_4 & \text{well-typed} \\ 0_{2^4} & \text{otherwise} \end{cases}$$

$$[\![G_1\ q_1\ q_2]\!]_d = \begin{cases} apply_2(G_1, q_1, q_2, d) & \text{well-typed} \\ 0_{2^d} & \text{otherwise} \end{cases}$$

X/Z-Propagation

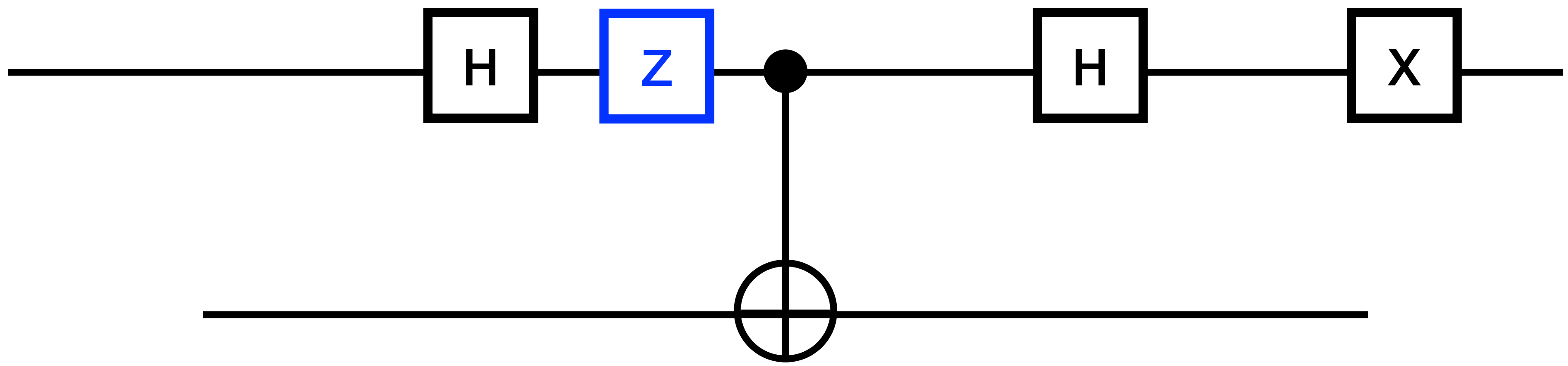


X/Z-Propagation



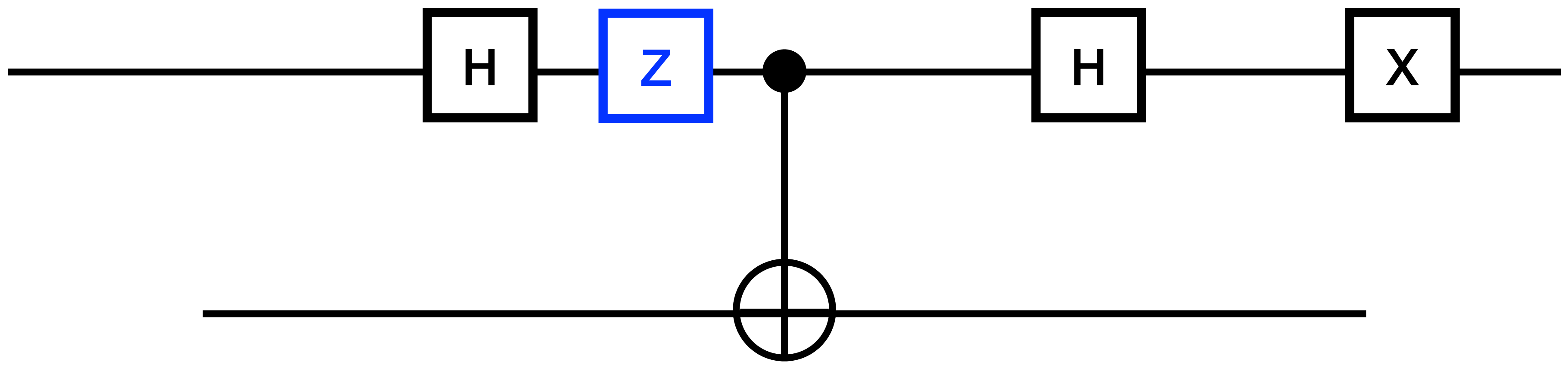

```
Lemma X_H_slide: X q; H q ≡ H q; Z q.
```

X/Z-Propagation




```
Lemma X_H_slide: X q; H q ≡ H q; Z q.
```

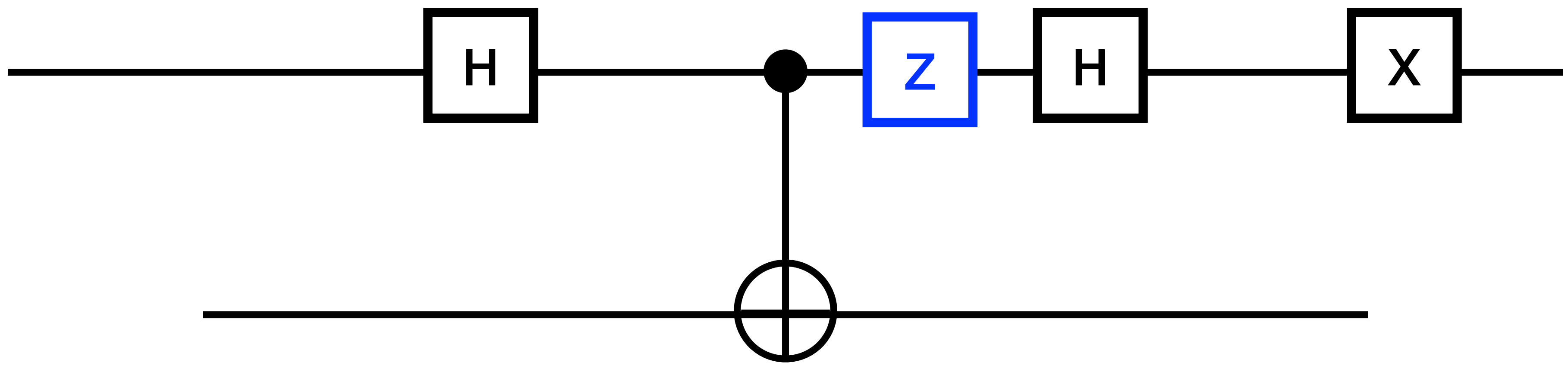
X/Z-Propagation




```
Lemma X_H_slide: X q; H q ≡ H q; Z q.
```

```
Lemma Z_CNOT_slide:
  Z q; CNOT q q' ≡ CNOT q q'; Z q
```

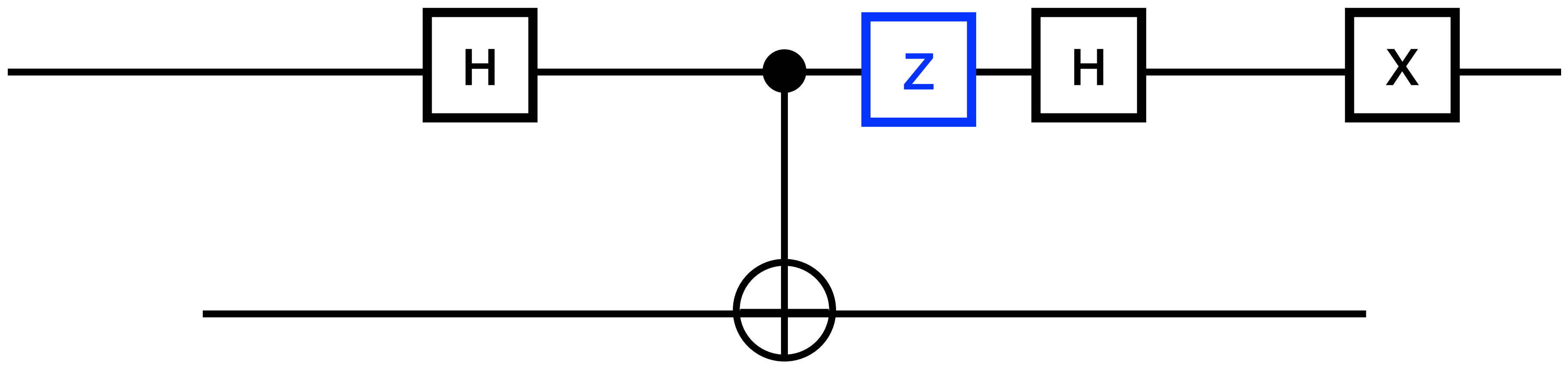
X/Z-Propagation




```
Lemma X_H_slide: X q; H q ≡ H q; Z q.
```

```
Lemma Z_CNOT_slide:
  Z q; CNOT q q' ≡ CNOT q q'; Z q
```

X/Z-Propagation

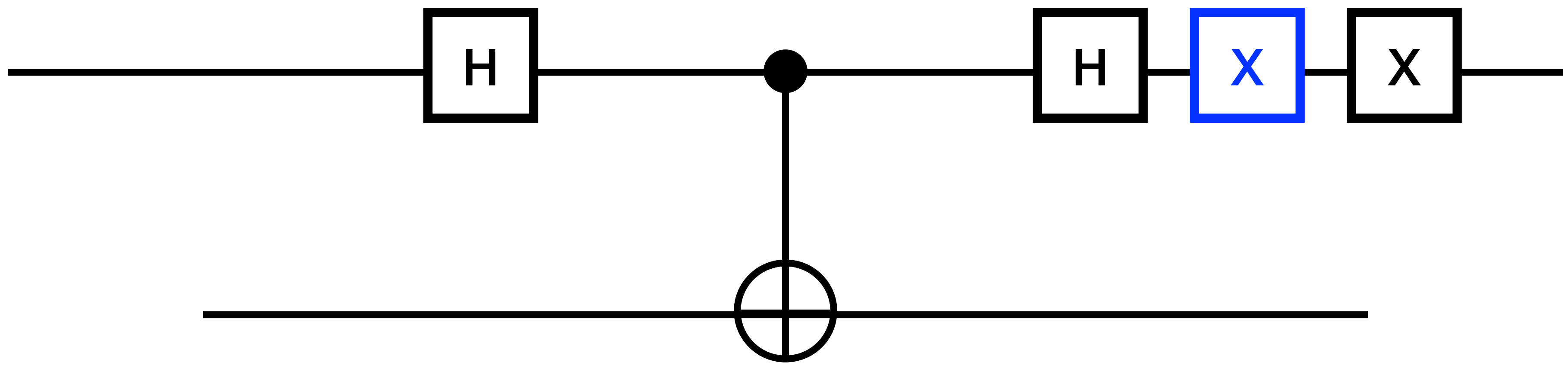



```
Lemma X_H_slide: X q; H q ≡ H q; Z q.
```

```
Lemma Z_CNOT_slide:
  Z q; CNOT q q' ≡ CNOT q q'; Z q
```

```
Lemma Z_H_slide: Z q; H q ≡ X q; Z q.
```

X/Z-Propagation

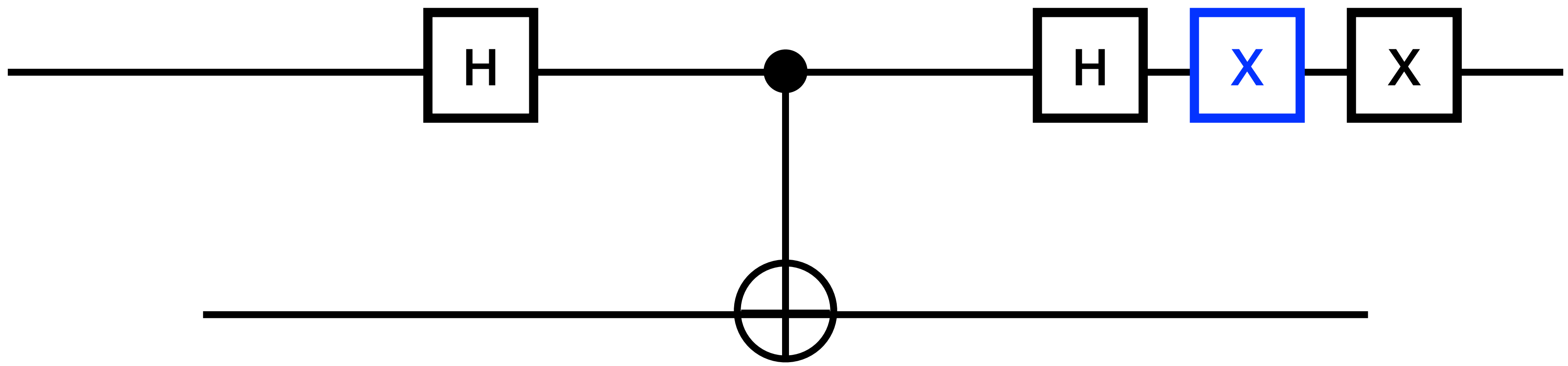



```
Lemma X_H_slide: X q; H q ≡ H q; Z q.
```

```
Lemma Z_CNOT_slide:
  Z q; CNOT q q' ≡ CNOT q q'; Z q
```

```
Lemma Z_H_slide: Z q; H q ≡ X q; Z q.
```

X/Z-Propagation



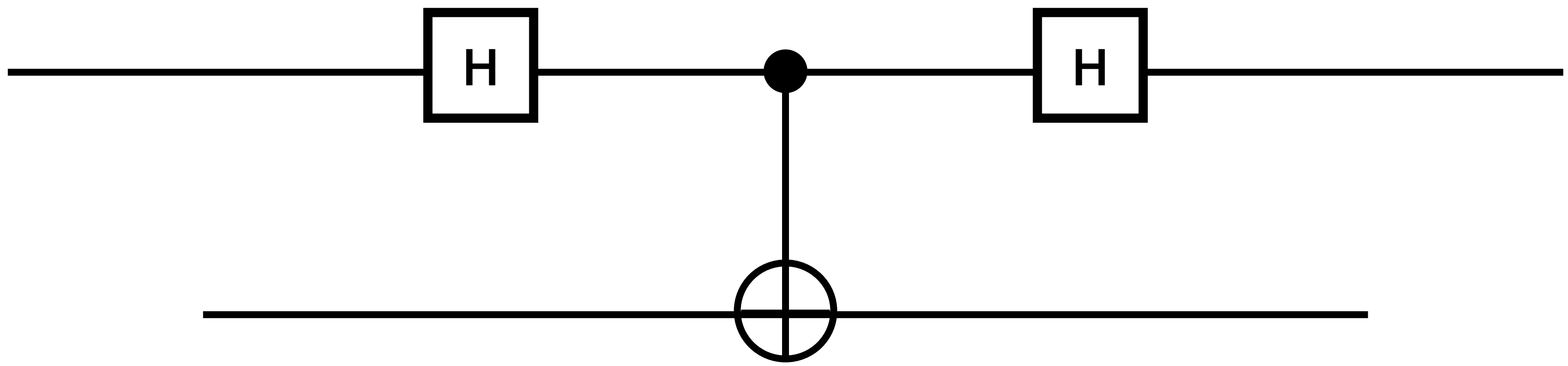

```
Lemma X_H_slide: X q; H q ≡ H q; Z q.
```

```
Lemma Z_CNOT_slide:
  Z q; CNOT q q' ≡ CNOT q q'; Z q
```

```
Lemma Z_H_slide: Z q; H q ≡ X q; Z q.
```

```
Lemma X_X_id: X q; X q ≡ I q
```

X/Z-Propagation



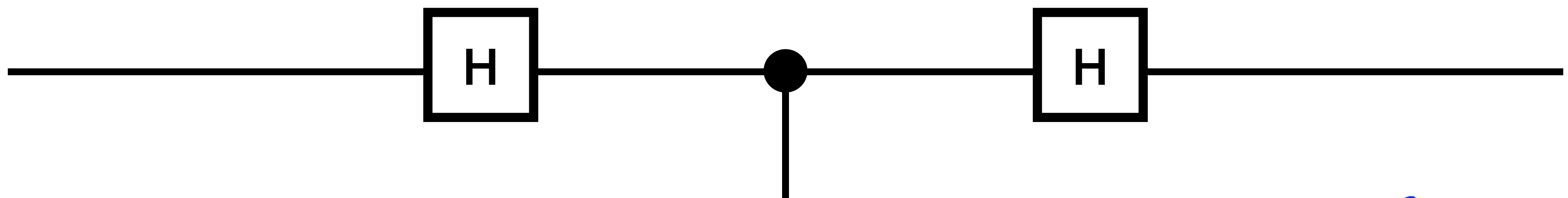

```
Lemma X_H_slide: X q; H q ≡ H q; Z q.
```

```
Lemma Z_CNOT_slide:
  Z q; CNOT q q' ≡ CNOT q q'; Z q
```

```
Lemma Z_H_slide: Z q; H q ≡ X q; Z q.
```

```
Lemma X_X_id: X q; X q ≡ I q
```

X/Z-Propagation




```
Theorem propagateZX_sound :
propagate_ZX P ≡ P.
```

```
Lemma X_H_slide: X q; H q ≡ H q; Z q.
```

```
Lemma Z_CNOT_slide:
  Z q; CNOT q q' ≡ CNOT q q'; Z q
```

```
Lemma Z_H_slide: Z q; H q ≡ X q; Z q.
```

```
Lemma X_X_id: X q; X q ≡ I q
```

Verifying Matrix Equivalences

- Proving matrix equivalences in Coq is tedious
- E.g. $X\ n;\ CNOT\ m\ n \equiv CNOT\ m\ n;\ X\ n$

$$\Updownarrow$$

$$apply_1(X, n, d) \times apply_2(CNOT, m, n, d) = apply_2(CNOT, m, n, d) \times apply_1(X, n, d).$$

Verifying Matrix Equivalences

- Proving matrix equivalences in Coq is tedious
- E.g. $$X\ n;\ CNOT\ m\ n \equiv CNOT\ m\ n;\ X\ n$$

$$\Updownarrow$$

$$apply_1(X, n, d) \times apply_2(CNOT, m, n, d) = apply_2(CNOT, m, n, d) \times apply_1(X, n, d).$$

$$apply_1(X, n, d) = I_{2^n} \otimes \sigma_x \otimes I_{2^q}$$

$$apply_2(CNOT, m, n, d) = I_{2^m} \otimes |1\rangle\langle 1| \otimes I_{2^p} \otimes \sigma_x \otimes I_{2^q} + I_{2^m} \otimes |0\rangle\langle 0| \otimes I_{2^p} \otimes I_2 \otimes I_{2^q}$$

Verifying Matrix Equivalences

- Proving matrix equivalences in Coq is tedious
- E.g. $X\ n;\ CNOT\ m\ n \equiv CNOT\ m\ n;\ X\ n$

$$apply_1(X, n, d)\ \times\ apply_2(CNOT, m, n, d) = apply_2(CNOT, m, n, d)\ \times\ apply_1(X, n, d).$$

Verifying Matrix Equivalences

- Proving matrix equivalences in Coq is tedious
- E.g. $X\ n;\ CNOT\ m\ n \equiv CNOT\ m\ n;\ X\ n$

$$\Updownarrow$$

$$apply_1(X, n, d)\ \times\ apply_2(CNOT, m, n, d) = apply_2(CNOT, m, n, d)\ \times\ apply_1(X, n, d).$$

$$\xrightarrow{\texttt{normalize}}$$

Verifying Matrix Equivalences

- Proving matrix equivalences in Coq is tedious
- E.g. $X\ n;\ CNOT\ m\ n \equiv CNOT\ m\ n;\ X\ n$

$$\Updownarrow$$

$$apply_1(X, n, d) \times apply_2(CNOT, m, n, d) = apply_2(CNOT, m, n, d) \times apply_1(X, n, d)$$

normalize

normalize

Verifying Matrix Equivalences

- Proving matrix equivalences in Coq is tedious
- E.g. $X\ n;\ CNOT\ m\ n \equiv CNOT\ m\ n;\ X\ n$

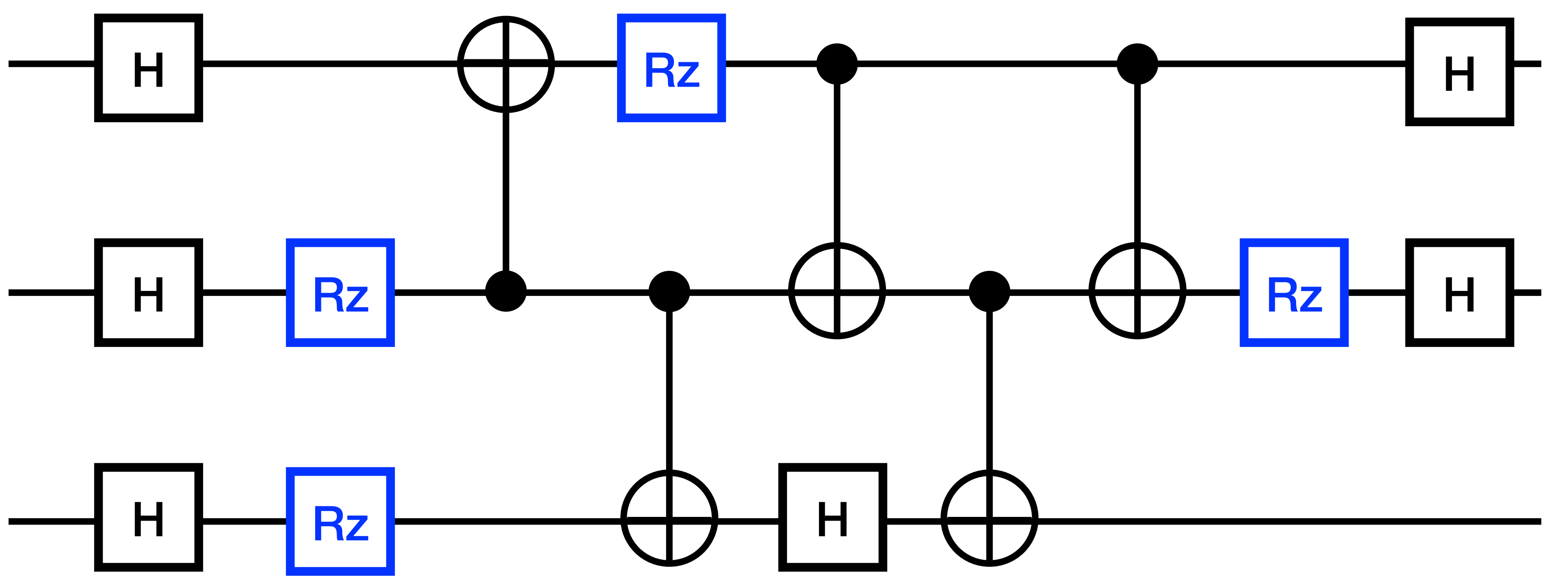
$$\Updownarrow$$

$$apply_1(X,n,d) \times apply_2(CNOT,m,n,d) = apply_2(CNOT,m,n,d) \times apply_1(X,n,d)$$

$$\searrow^{\texttt{normalize}} \qquad\qquad \swarrow^{\texttt{normalize}}$$

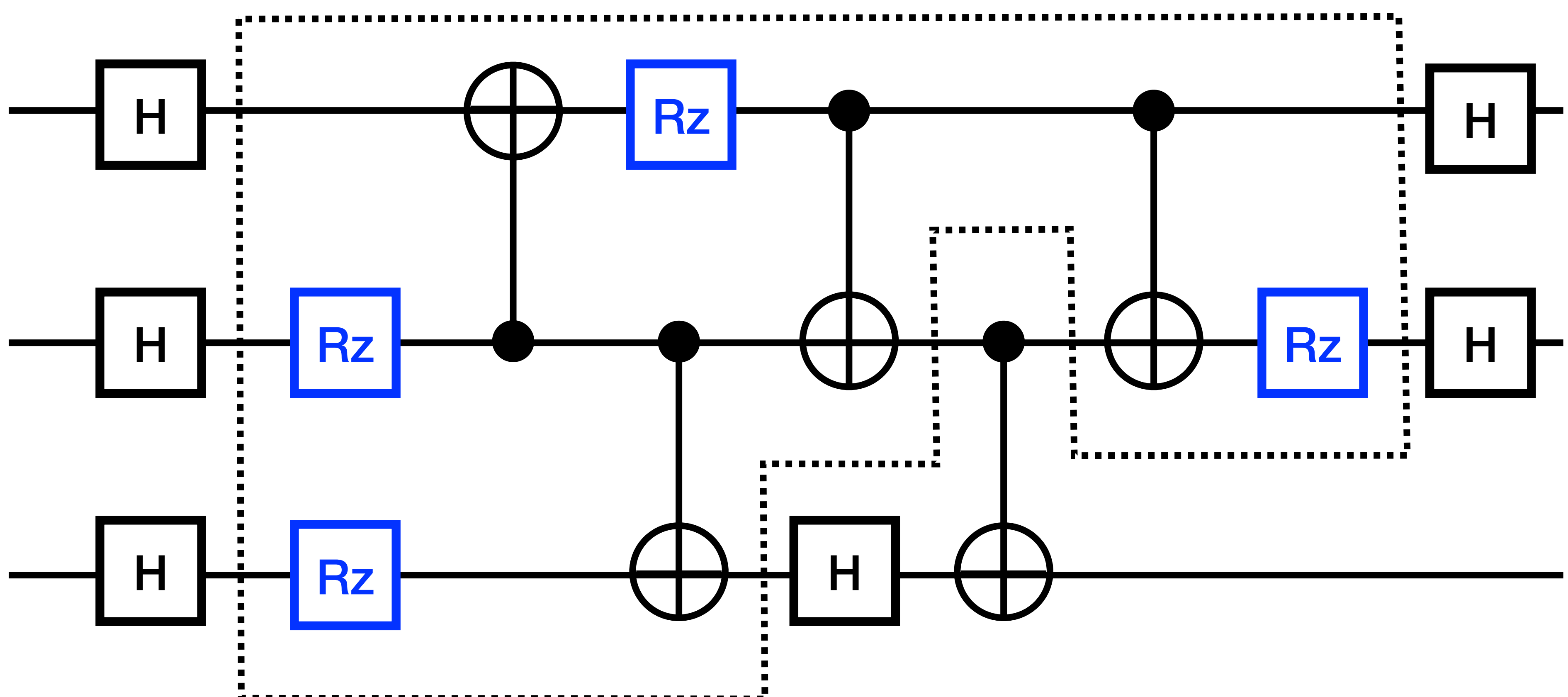
$$I_{2^m} \otimes |1\rangle\langle 1| \otimes I_{2^p} \otimes I_2 \otimes I_{2^q} + I_{2^m} \otimes |0\rangle\langle 0| \otimes I_{2^p} \otimes \sigma_x \otimes I_{2^q}$$

Rotation Merging



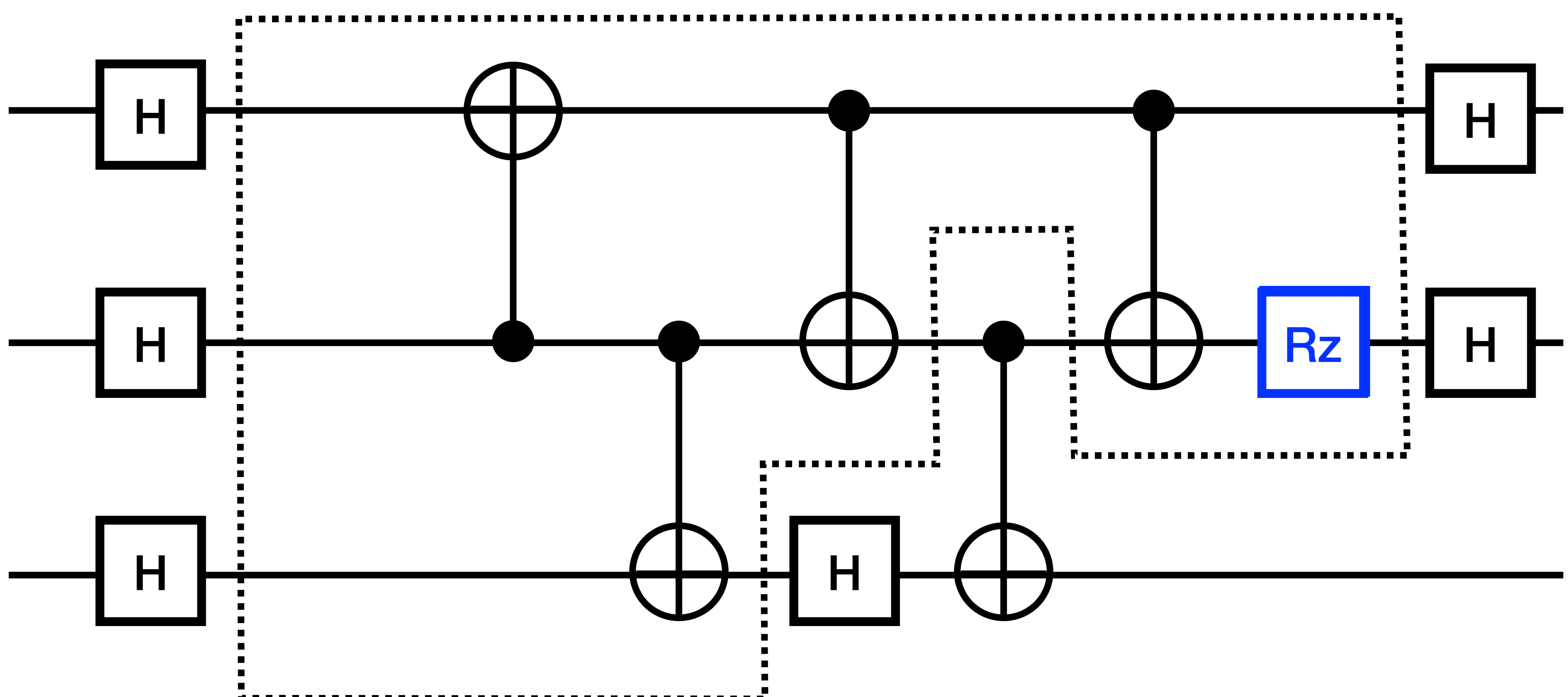
[Nam, Ross, Su, Childs, Maslov, 2018]

Rotation Merging



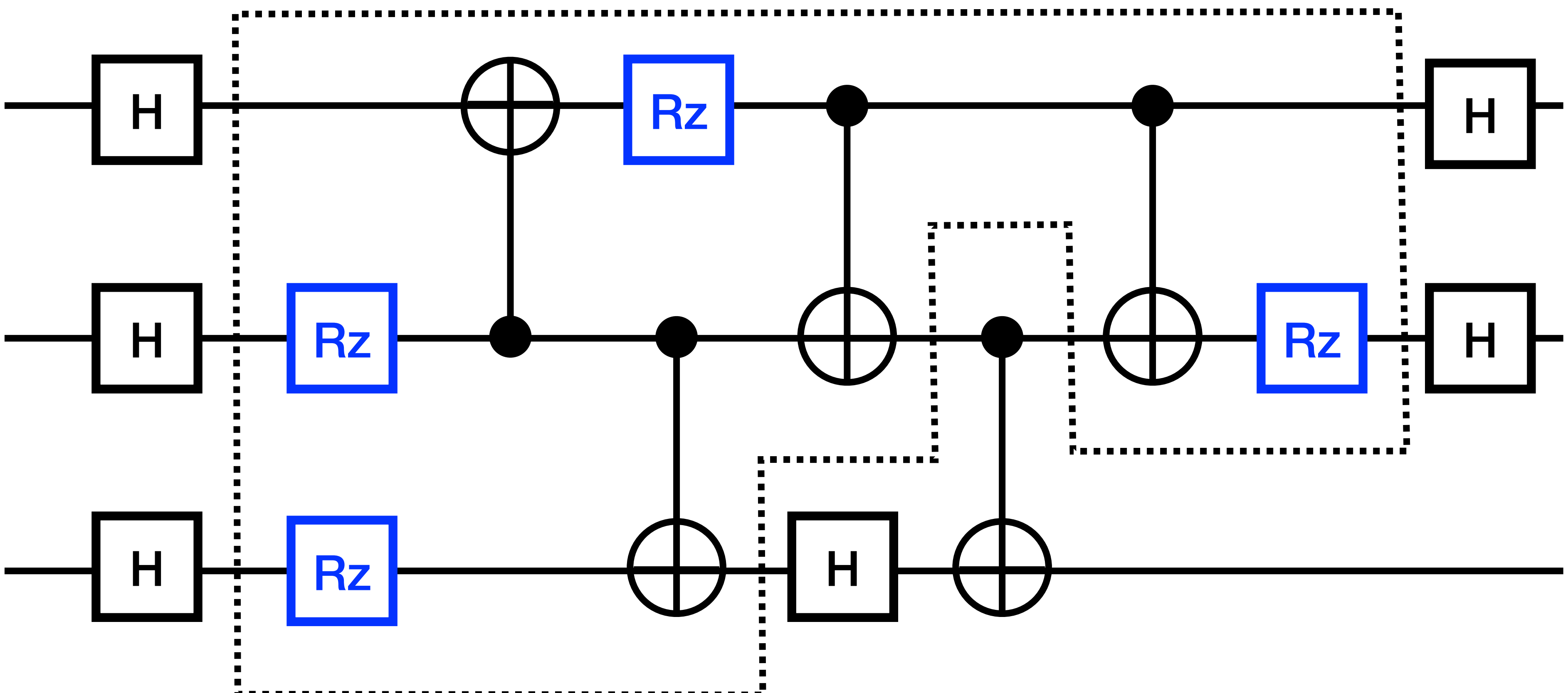
[Nam, Ross, Su, Childs, Maslov, 2018]

Rotation Merging



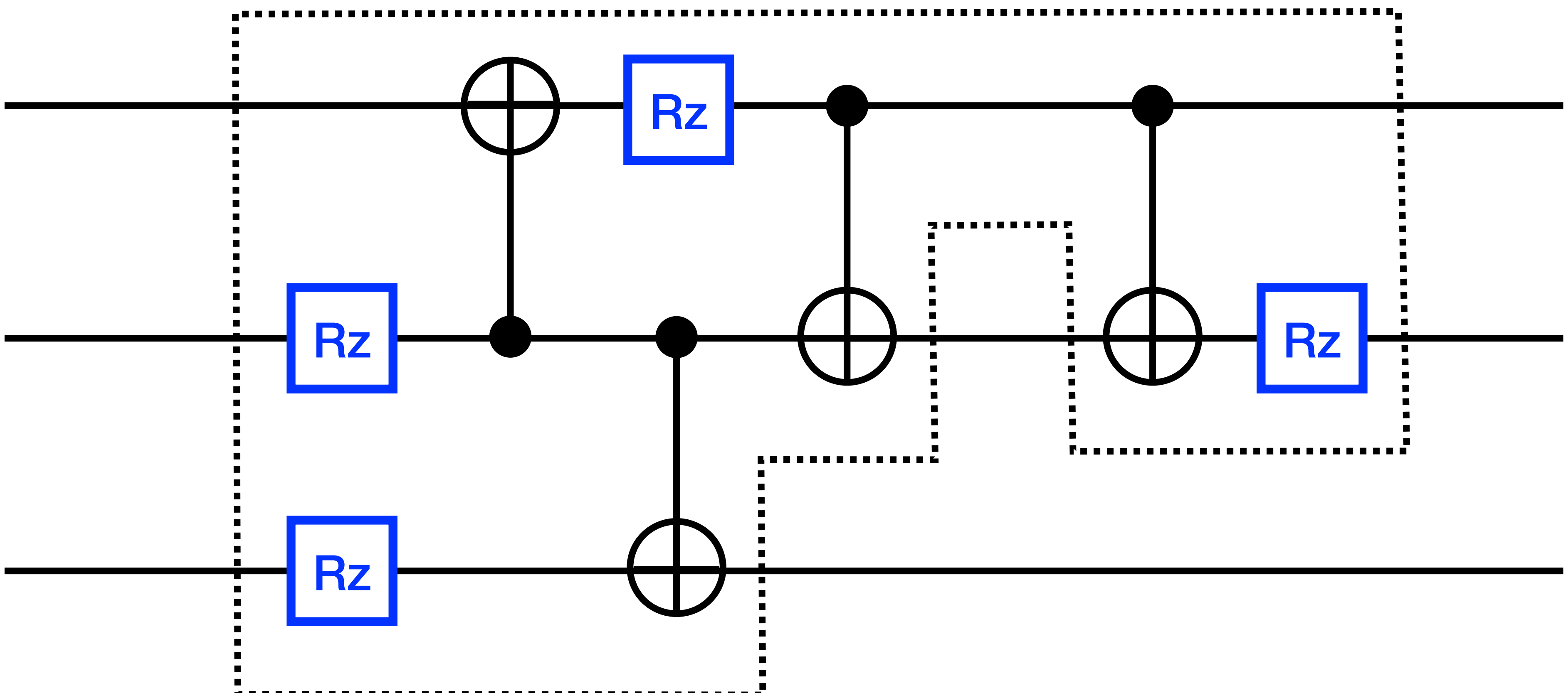
[Nam, Ross, Su, Childs, Maslov, 2018]

Rotation Merging



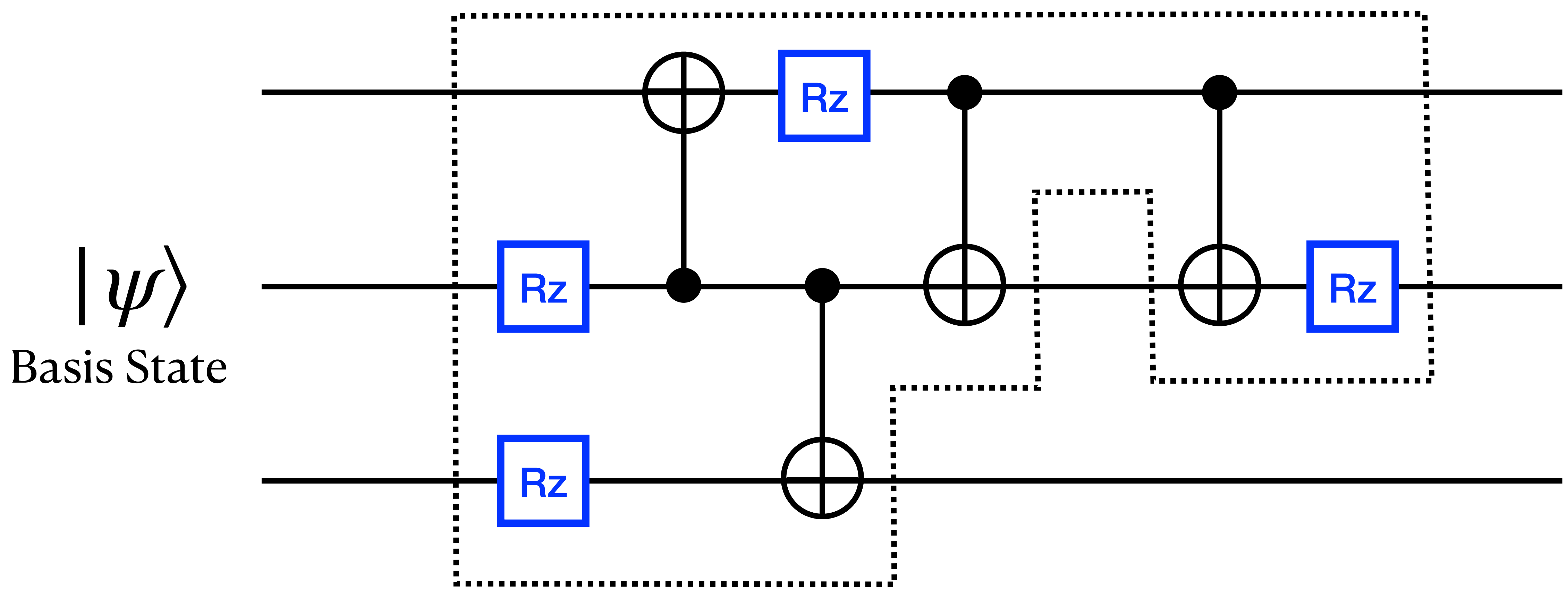
[Nam, Ross, Su, Childs, Maslov, 2018]

Rotation Merging



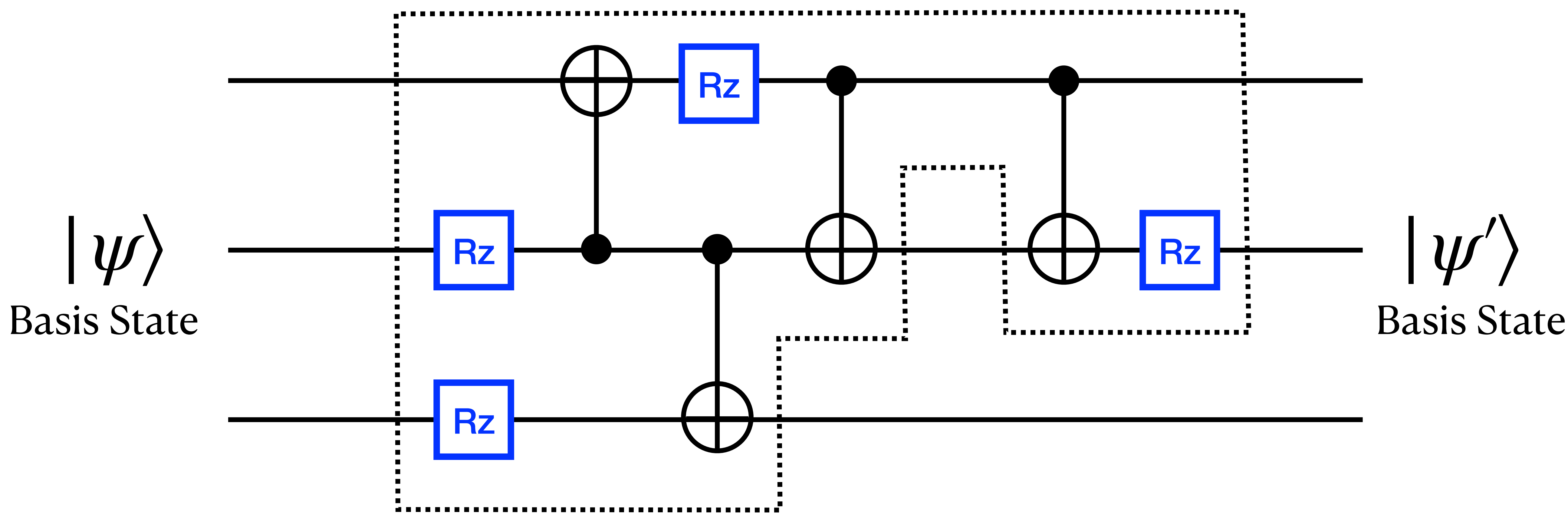
[Nam, Ross, Su, Childs, Maslov, 2018]

Rotation Merging



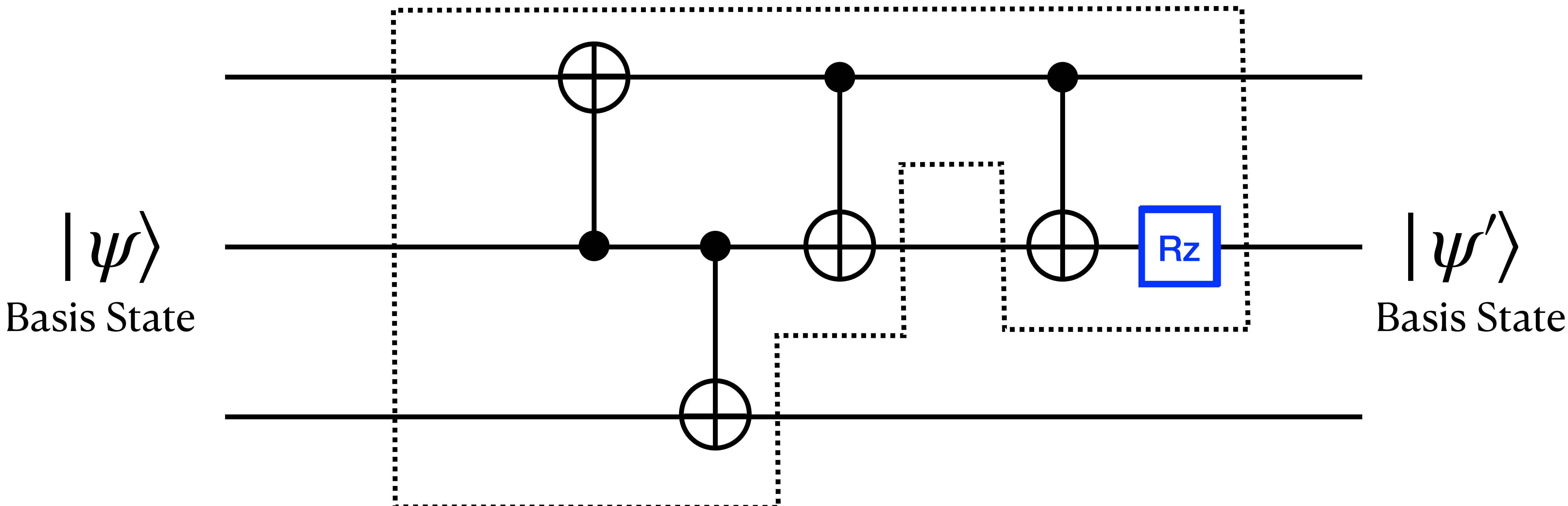
[Nam, Ross, Su, Childs, Maslov, 2018]

Rotation Merging



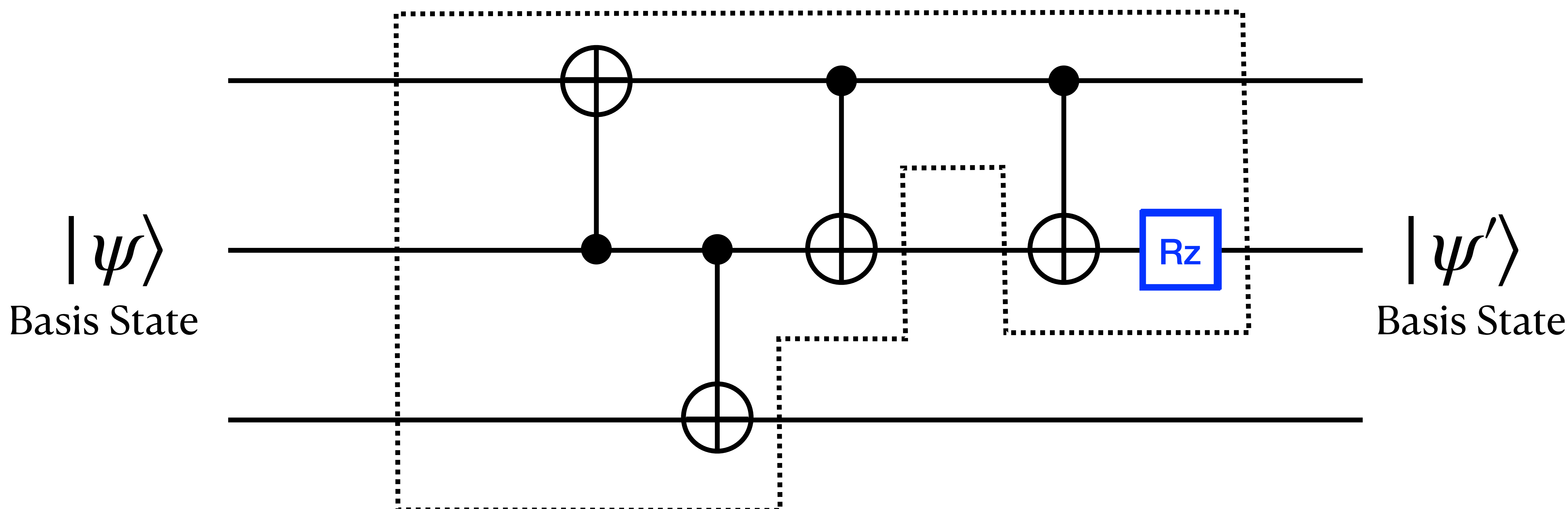
[Nam, Ross, Su, Childs, Maslov, 2018]

Rotation Merging



[Nam, Ross, Su, Childs, Maslov, 2018]

Rotation Merging



[Nam, Ross, Su, Childs, Maslov, 2018]

Performance

- Our *verified* optimizer vs. existing *unverified* optimizers

Geometric mean runtime						
Qiskit[1]	**tket**[2]	**Nam**[3] **(L)**	**Nam (H)**	**Amy**[4]	**PyZX**[5]	**VOQC**
2.128s	0.226s	0.002s	0.018s	0.007s	0.204s	0.012s

Avg. reduction in gate count			
Qiskit	**tket**	**Nam**	**VOQC**
10.4%	10.9%	26.4%	18.7%

Avg. reduction in T gates			
Amy	**PyZX**	**Nam**	**VOQC**
40.9%	43.8%	42.3%	42.3%

1 https://qiskit.org/
2 https://cqcl.github.io/pytket/build/html/index.html
3 https://arxiv.org/pdf/1710.07345.pdf
4 https://arxiv.org/pdf/1303.2042.pdf
5 https://github.com/Quantomatic/pyzx

Non-unitary Optimizations

Non-unitary Optimizations

```
Lemma Z_meas:
  Z q ; meas q then c1 else c2 ≡
        meas q then c1 else c2
```

Non-unitary Optimizations

```
Lemma Z_meas:
  Z q ; meas q then c1 else c2 ≡
        meas q then c1 else c2
```

```
Lemma X_meas:
  X q ; meas q then c1 else c2 ≡
  meas q then X q; c2 else X q; c1
```

Non-unitary Optimizations

```
Lemma Z_meas:
  Z q ; meas q then c1 else c2 ≡
        meas q then c1 else c2
```

```
Lemma X_meas:
  X q ; meas q then c1 else c2 ≡
  meas q then X q; c2 else X q; c1
```

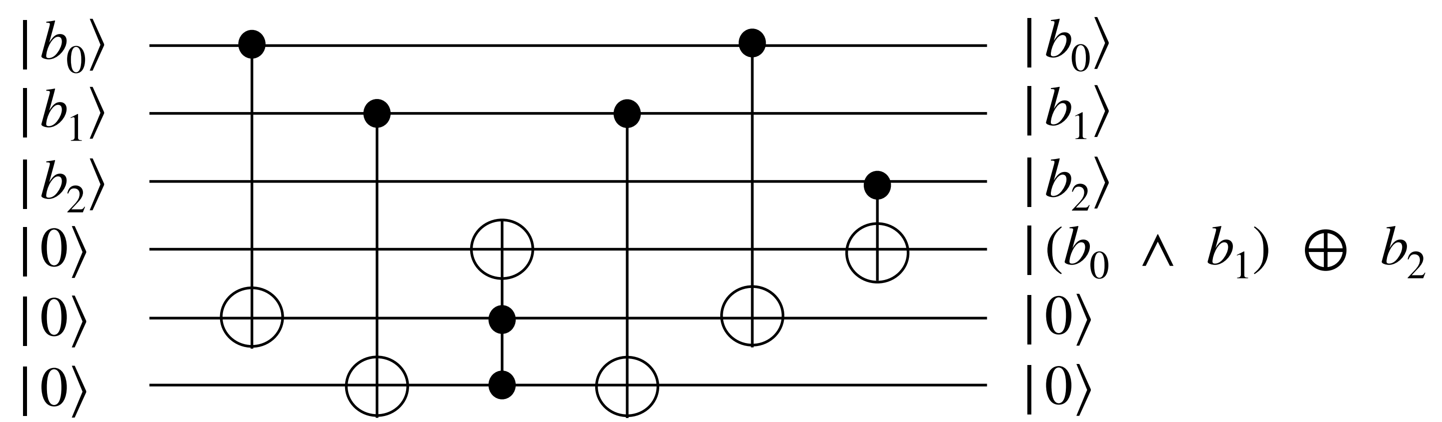
```
Lemma meas_CNOT:
  meas q then c1 else CNOT q q'; c2 ≡
  meas q then c1 else c2
```

Circuit Mapping

- Given an input program & description of machine connectivity, mapping produces a program that meets connectivity constraints
  - E.g. `CNOT 0 2` $\boxed{0} \leftrightarrow \boxed{1} \leftrightarrow \boxed{2}$ $\rightarrow$ `SWAP 0 1; CNOT 1 2`
- We prove that the output program is equivalent to the original, up to permutation of indices
  - Above, $[\![\texttt{CNOT}\ 0\ 2]\!]_3 = P \times [\![\texttt{SWAP}\ 0\ 1;\ \texttt{CNOT}\ 1\ 2]\!]_3$ where *P* implements the permutation $\{0 \rightarrow 1,\ 1 \rightarrow 0,\ 2 \rightarrow 2\}$

Boolean Oracle Compilation

$(b_0 \wedge b_1) \oplus b_2$



- Many quantum programs rely on *classical oracles*, classical functions evaluated on quantum data
- We have verified the compilation of Boolean formulas into quantum circuits with X, CNOT, and Toffoli gates

General Verification

- Scale invariant correctness proofs of variety of quantum algorithms:
    - GHZ state preparation
    - Deutsch-Jozsa algorithm
    - Simon's algorithm
    - Quantum phase estimation

In Progress

- Full proof of correctness for Shor’s algorithm
- Adding Python bindings to improve VOQC usability
- Extending oracle compilation to include arithmetic functions
- Implementing additional optimizations (e.g. those used in Qiskit)

Future Directions

- Verify approximate algorithms and optimizations
- Compile from high-level languages (Silq, Q#) to SQIR
- Compile from SQIR to low-level pulses
- Verify other parts of the software stack (e.g. resource estimation)

Future Directions

- Verify approximate algorithms and optimizations
- Compile from high-level languages (Silq, Q#) to SQIR
- Compile from SQIR to low-level pulses
- Verify other parts of the software stack (e.g. resource estimation)

Paper: http://people.cs.uchicago.edu/~rand/voqc_draft.pdf

Future Directions

- Verify approximate algorithms and optimizations
- Compile from high-level languages (Silq, Q#) to SQIR
- Compile from SQIR to low-level pulses
- Verify other parts of the software stack (e.g. resource estimation)

Paper: http://people.cs.uchicago.edu/~rand/voqc_draft.pdf

Code: https://github.com/inQWIRE/SQIR