Quantum Algorithms for Beginners

Chris Cade, University of Bristol

Quantum Information for Developers
ETH Zurich
2018

Quantum Algorithms

- There are a ***lot*** of quantum algorithms
  - The ‘Quantum Algorithm Zoo’ cites 392 papers on quantum algorithms
- Mostly, they solve specific mathematical problems
  - E.g. Factoring, matrix inversion
- Often cleverly combine smaller quantum sub-routines

This lecture

- Will focus on a few important algorithms / sub-routines:
  - Grover's search, phase estimation, factoring, matrix inversion (HHL), Hamiltonian simulation
- Mostly give high-level overviews
  - Hopefully enough detail to be able to implement Grover's search (and understand what's happening!)
- For a more complete introduction:
  - Ashley Montanaro's lecture notes www.people.maths.bris.ac.uk/~csxam/teaching
  - Ronald de Wolf's lecture notes www.homepages.cwi.nl/~rdewolf/qcnotes.pdf
  - *Quantum Computation and Quantum Information* by Nielsen and Chuang

We might divide quantum algorithms into two categories:

Polynomial speedup	Exponential speedup
Grover's Search	**Integer Factoring**
Quantum walks	**Matrix Inversion**
Graph algorithms	**Phase Estimation**
Minimum finding	Quantum Fourier Transform
	Quantum Simulation

We might divide quantum algorithms into two categories:

Polynomial speedup	Exponential speedup
Grover's Search	**Integer Factoring**
Quantum walks	**Matrix Inversion**
Graph algorithms	**Phase Estimation**
Minimum finding	Quantum Fourier Transform
	Quantum Simulation

We might divide quantum algorithms into two categories:

Polynomial speedup	Exponential speedup
Grover's Search	Integer Factoring
Quantum walks	Matrix Inversion
Graph algorithms	Phase Estimation
	Quantum Fourier Transform
Minimum finding	
	Quantum Simulation

We might divide quantum algorithms into two categories:

Polynomial speedup

- **Grover's Search**
- Quantum walks
- Graph algorithms
- Minimum finding

Exponential speedup

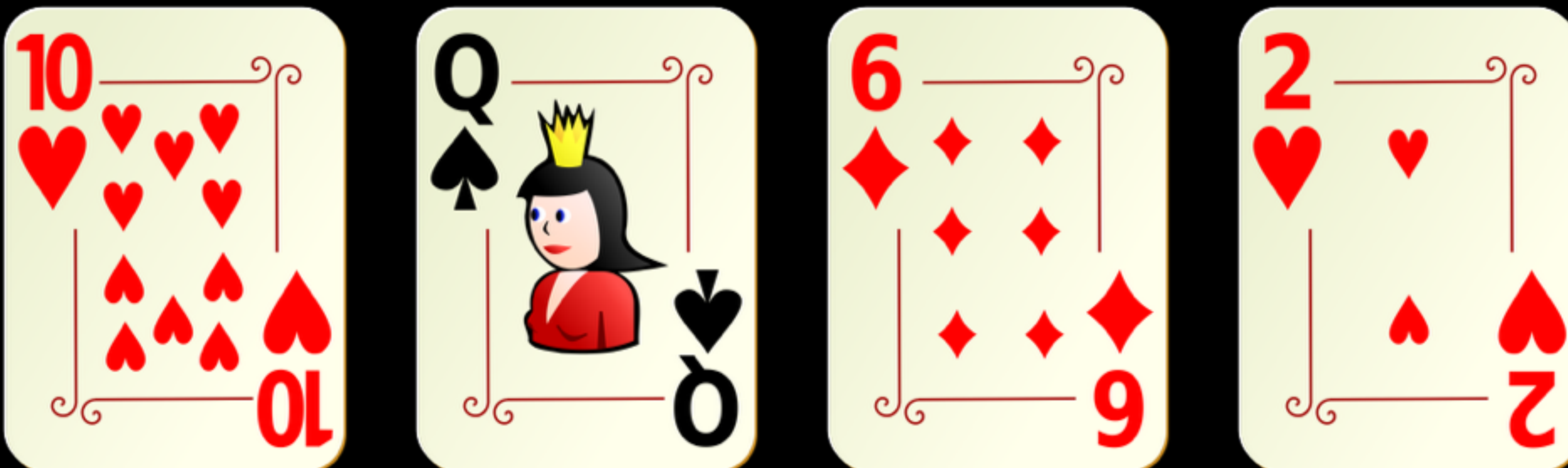
- **Integer Factoring**
- **Matrix Inversion**
- **Phase Estimation**
- Quantum Fourier Transform
- Quantum Simulation

We might divide quantum algorithms into two categories:

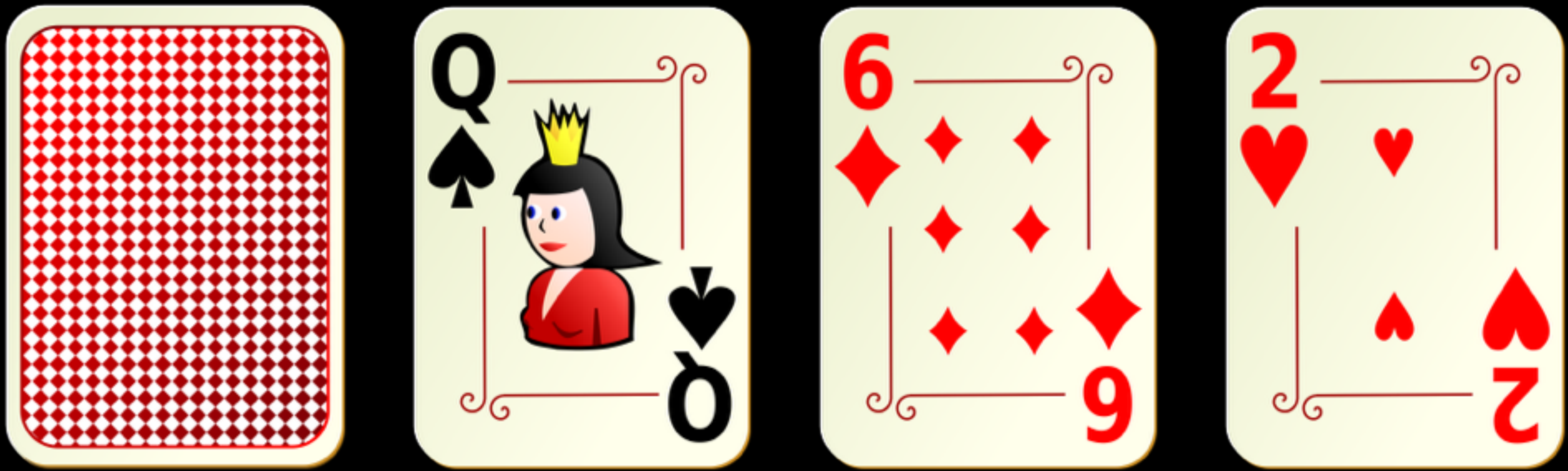
Polynomial speedup	Exponential speedup
Grover's Search	Integer Factoring
Quantum walks	Matrix Inversion
Graph algorithms	Phase Estimation
Minimum finding	Quantum Fourier Transform
	Quantum Simulation

Grover's Algorithm

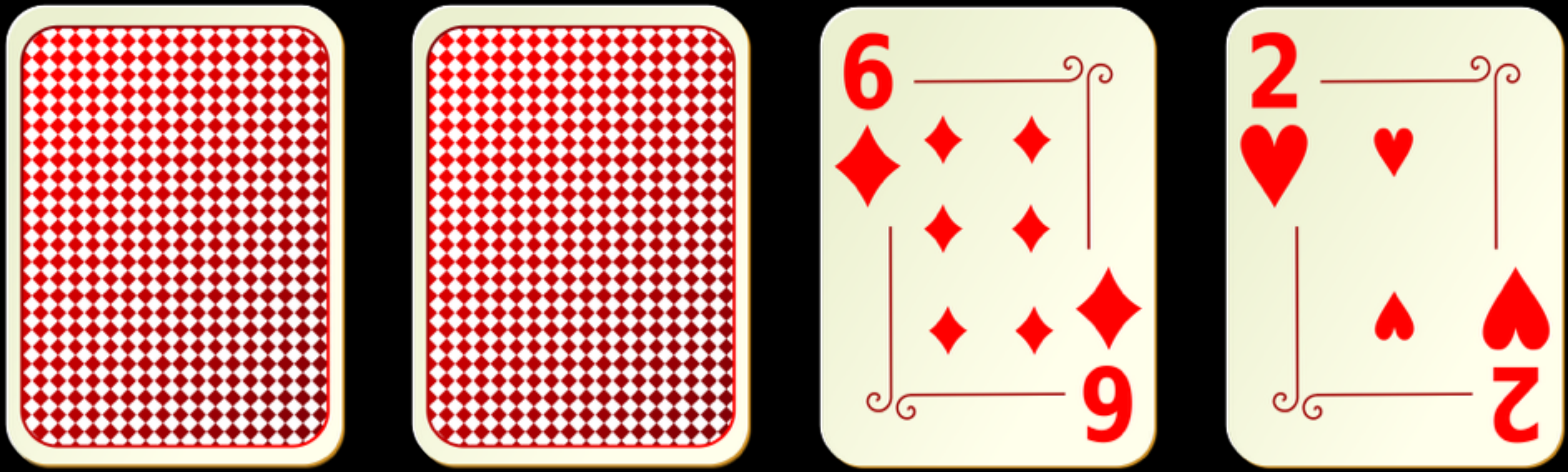
Grover's Algorithm



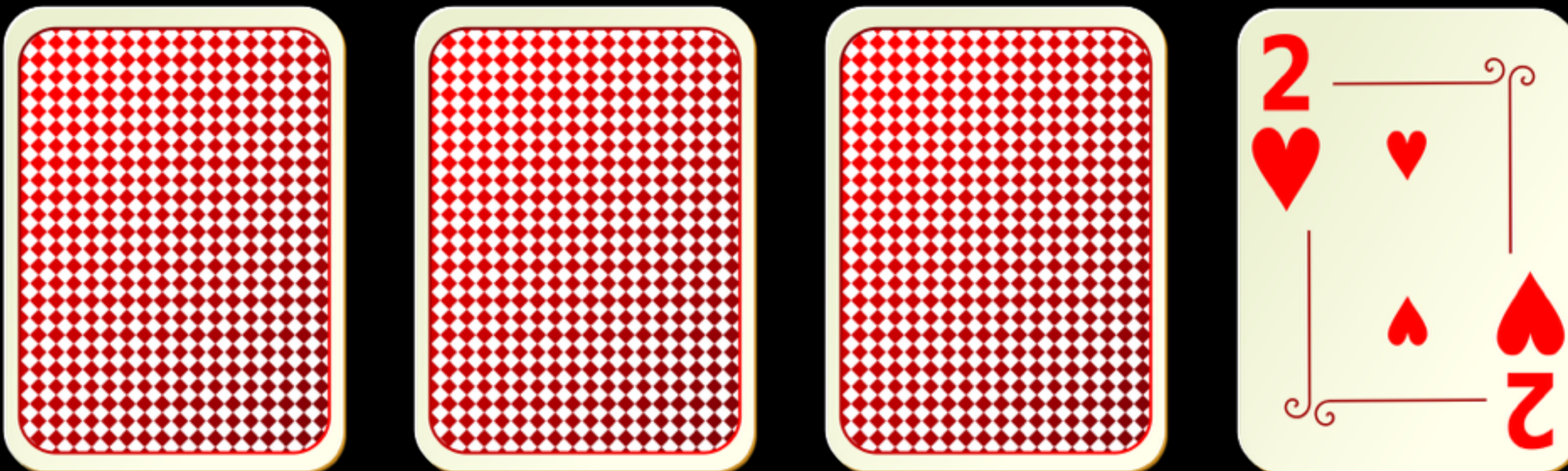
Grover's Algorithm



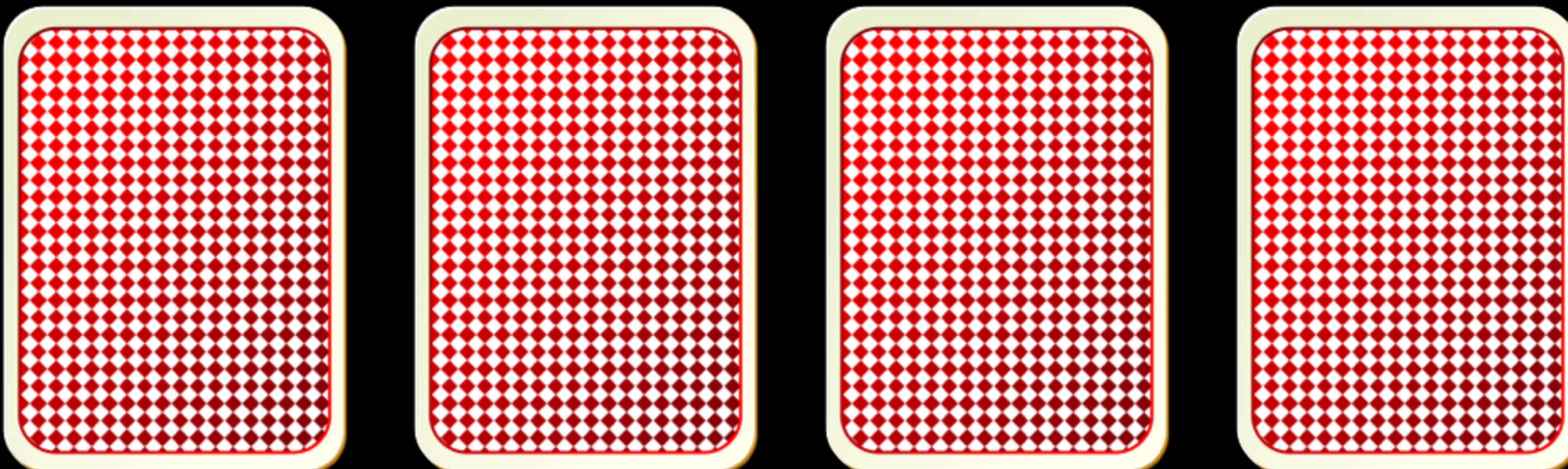
Grover's Algorithm



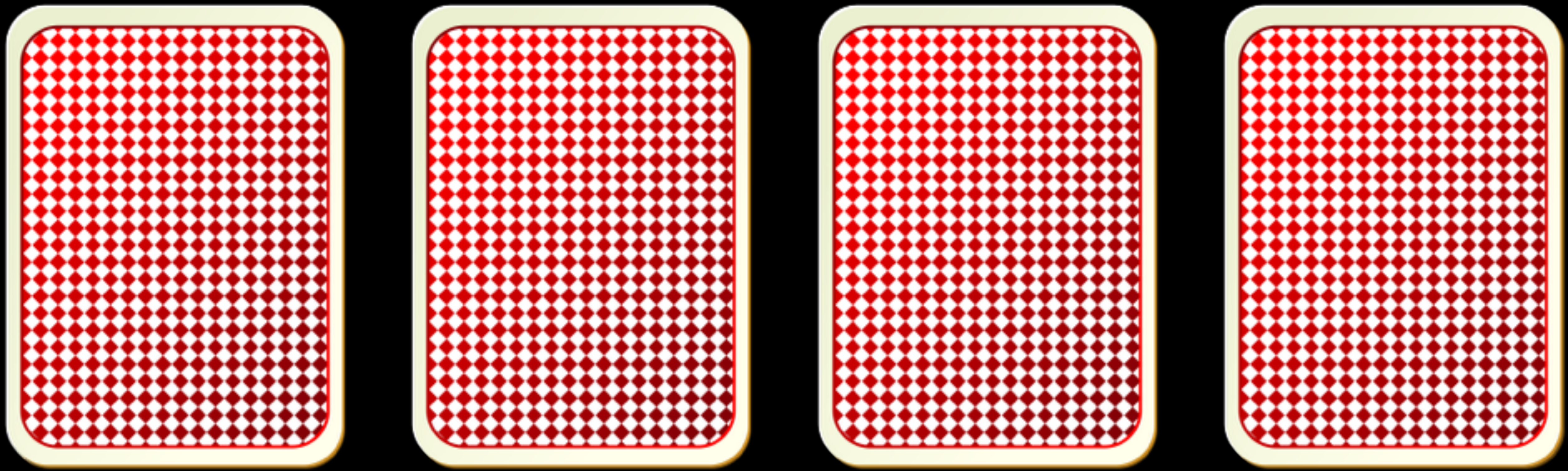
Grover's Algorithm



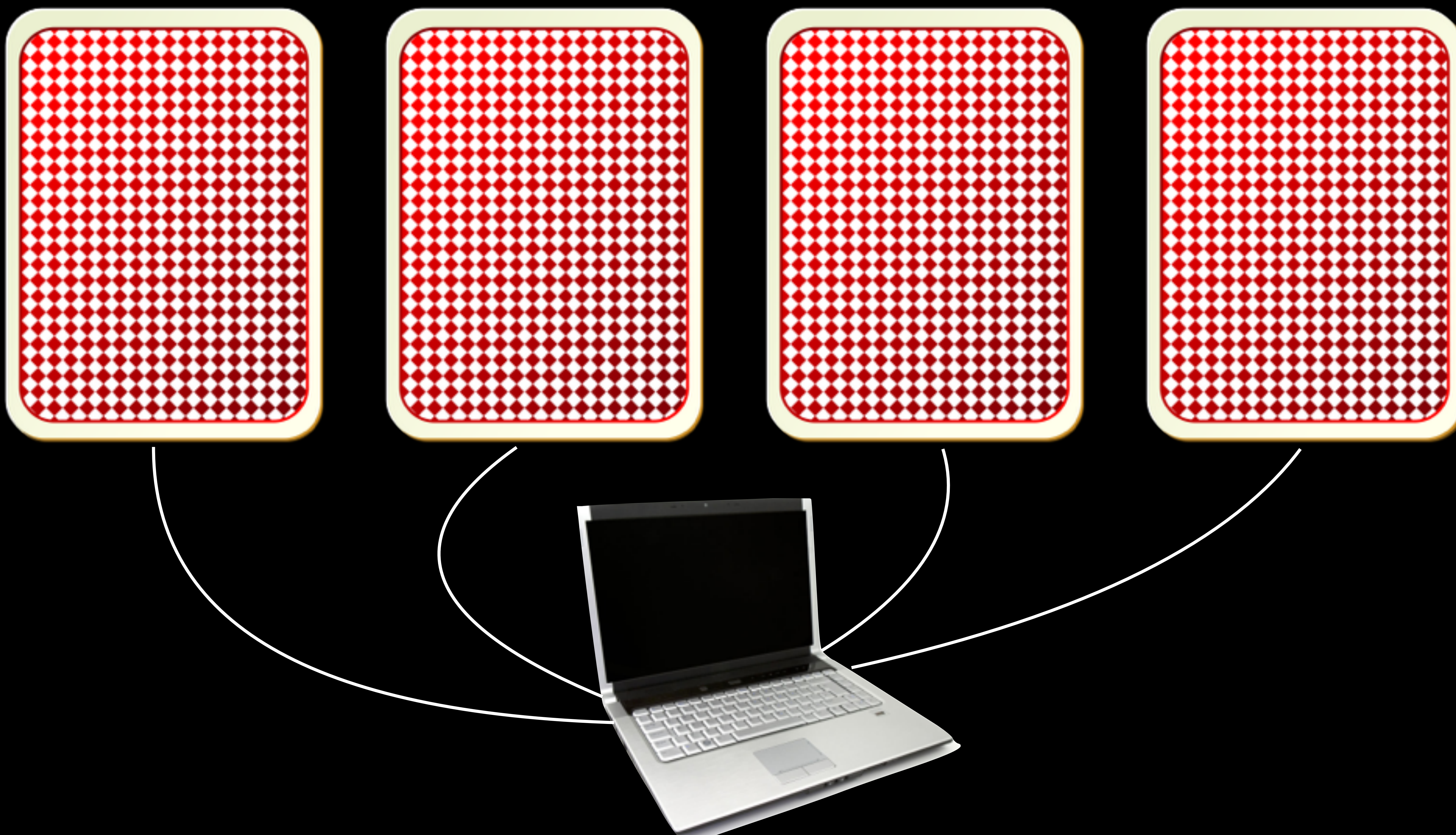
Grover's Algorithm



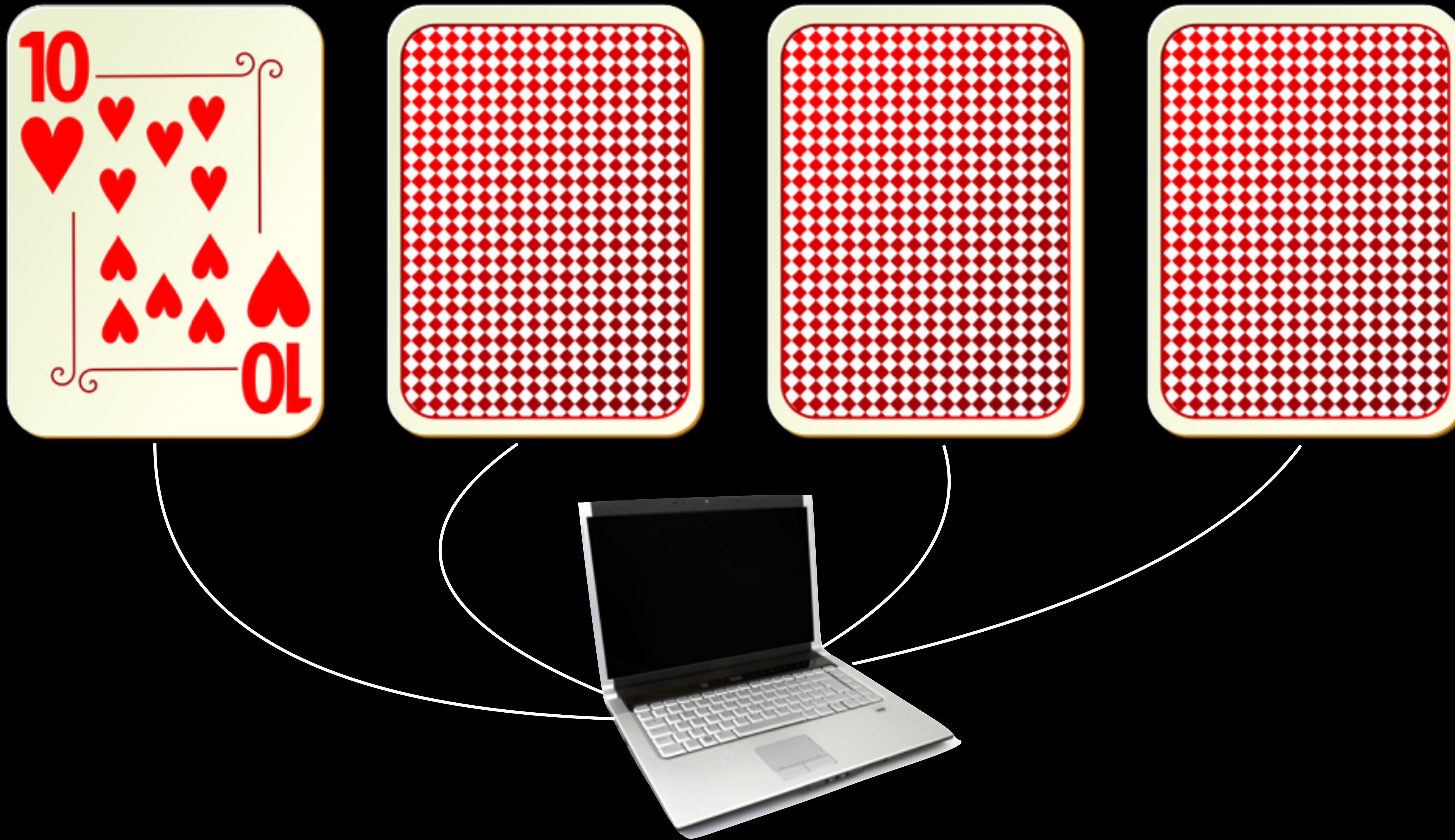
Grover's Algorithm



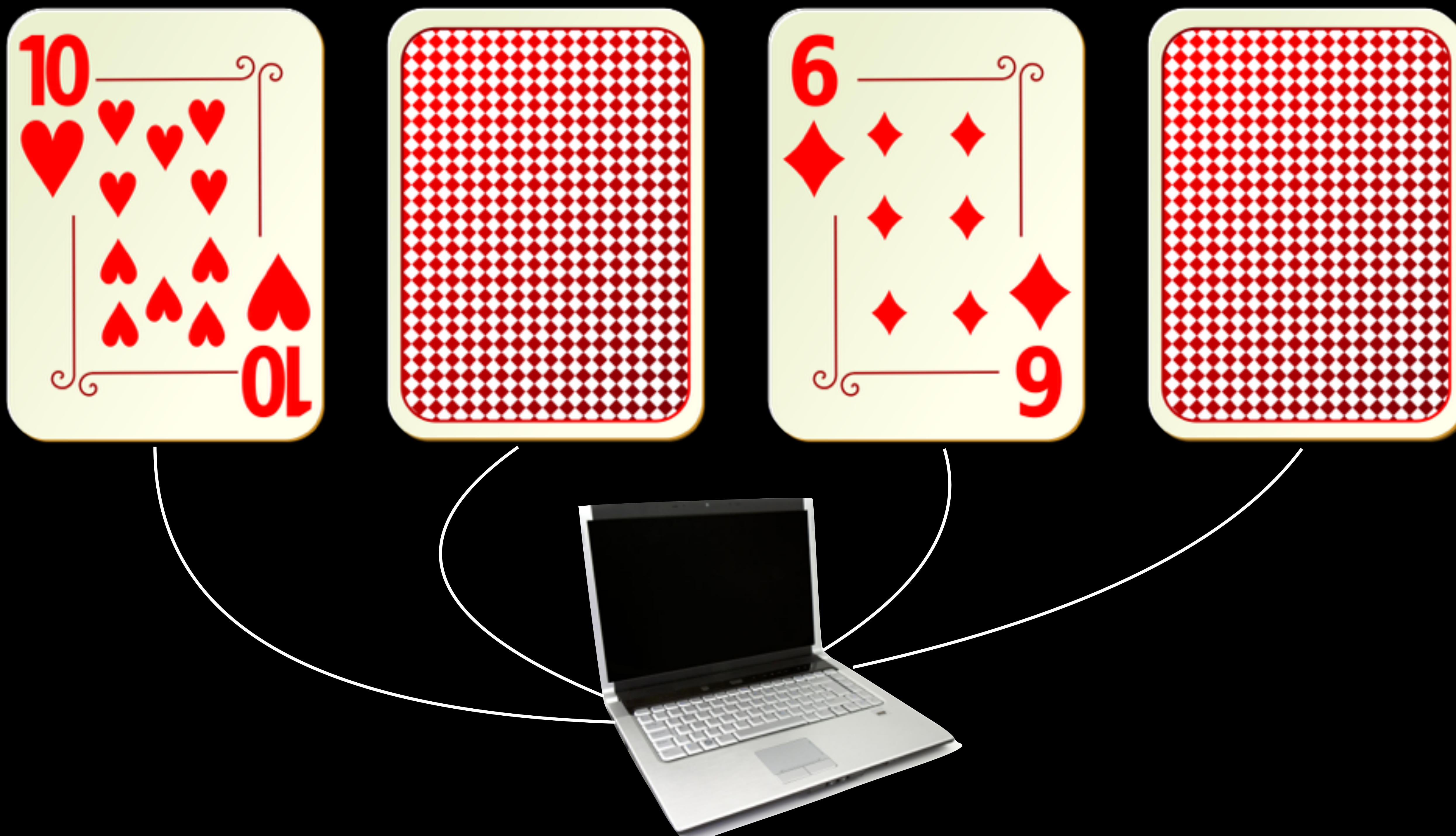
Grover's Algorithm



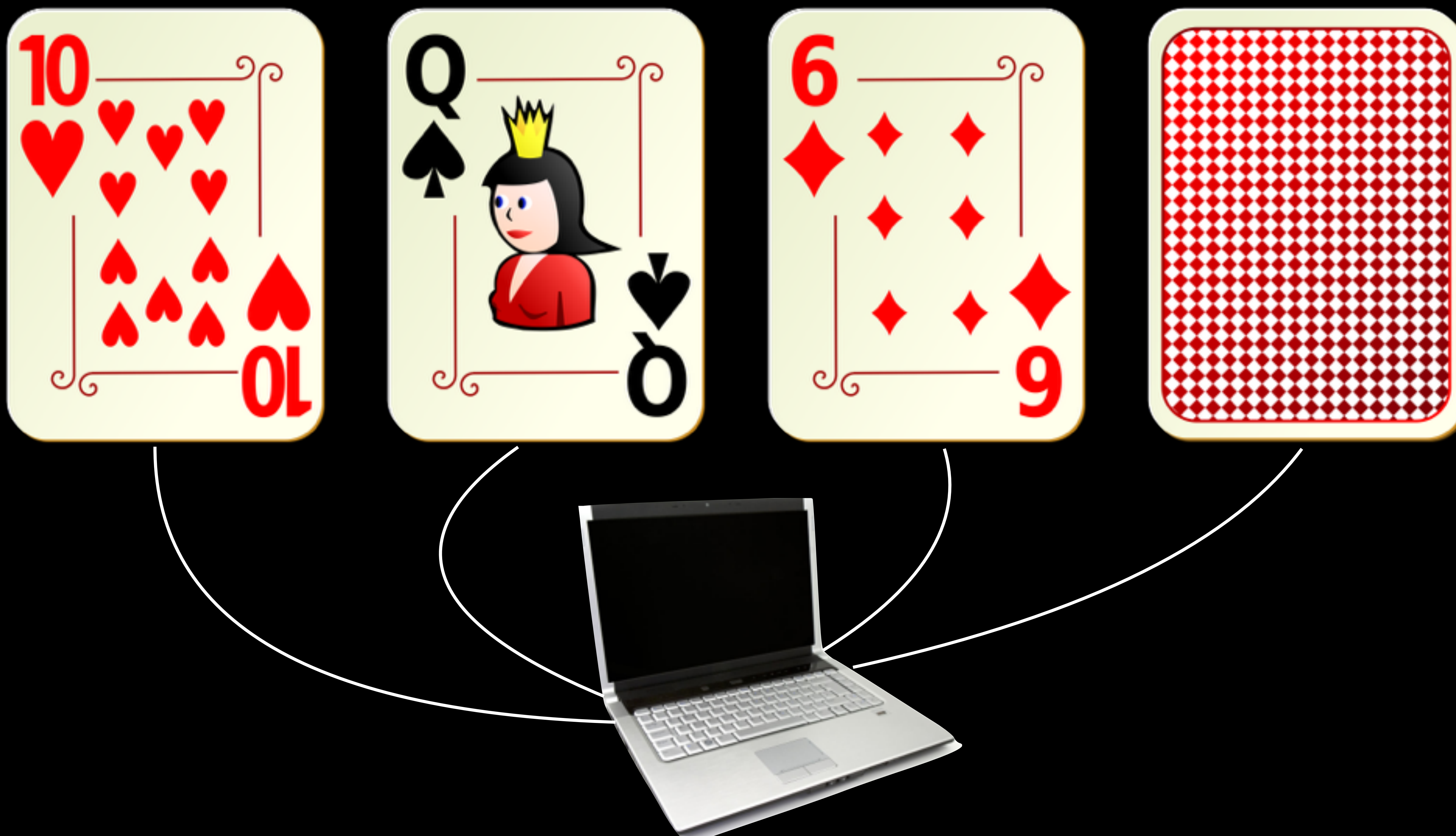
Grover's Algorithm



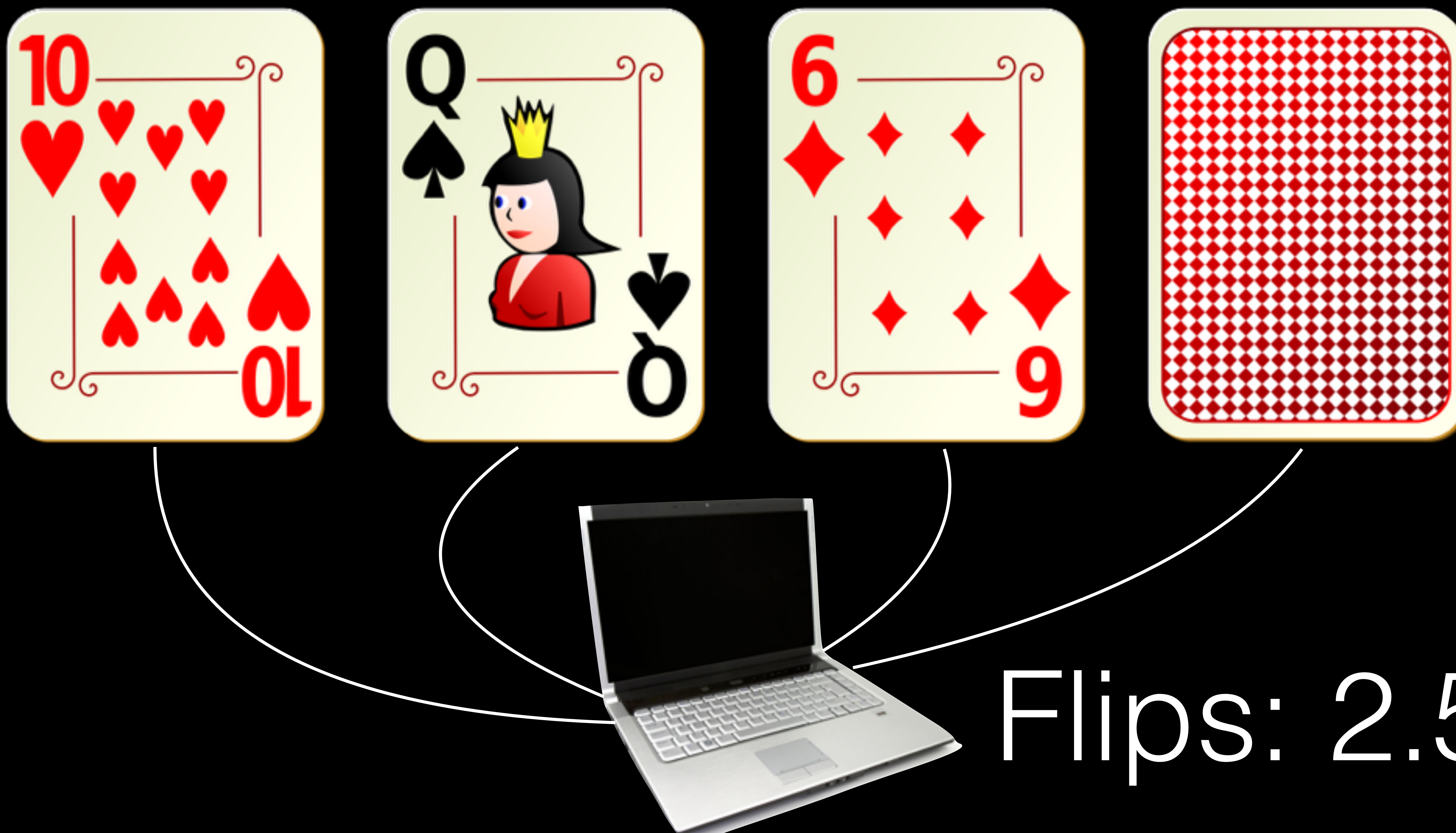
Grover's Algorithm



Grover's Algorithm



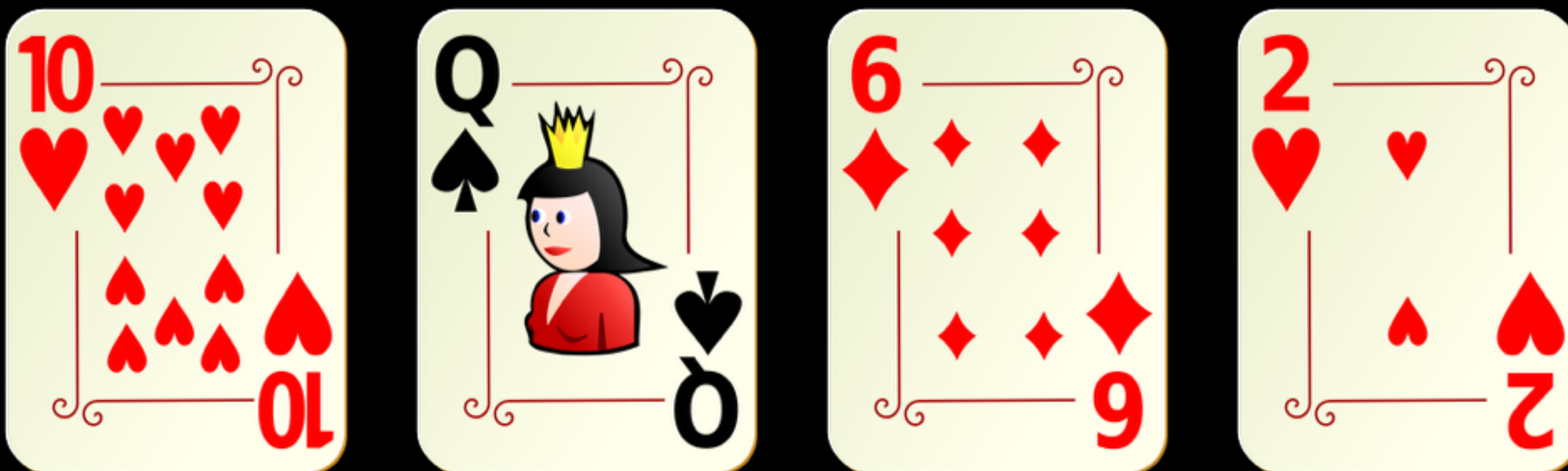
Grover's Algorithm



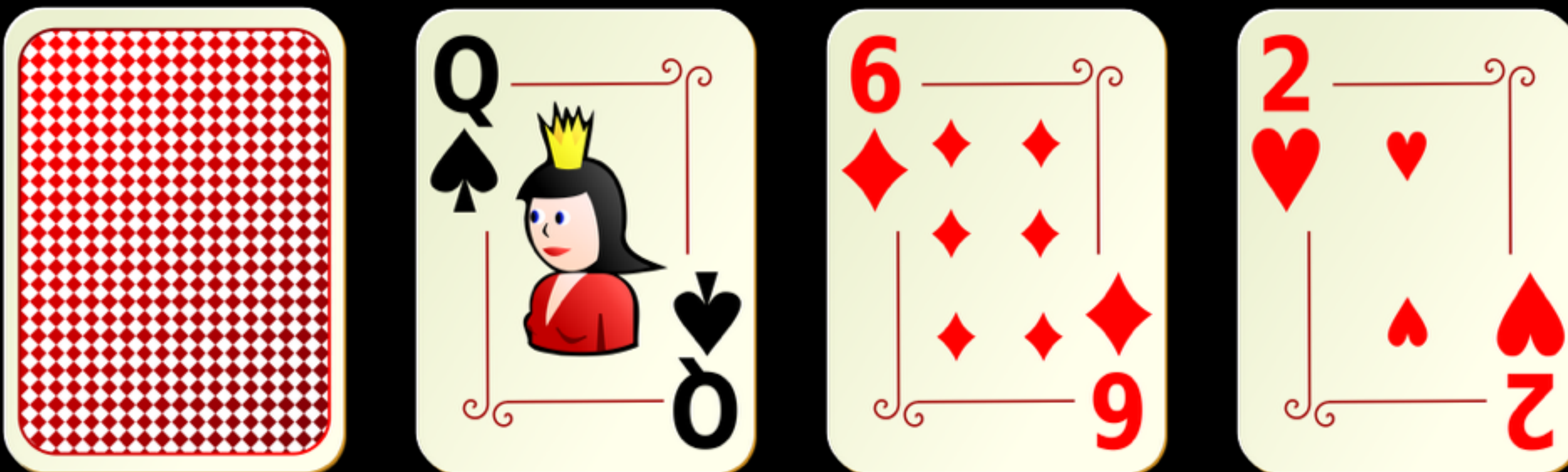
Flips: 2.5

Grover's Algorithm

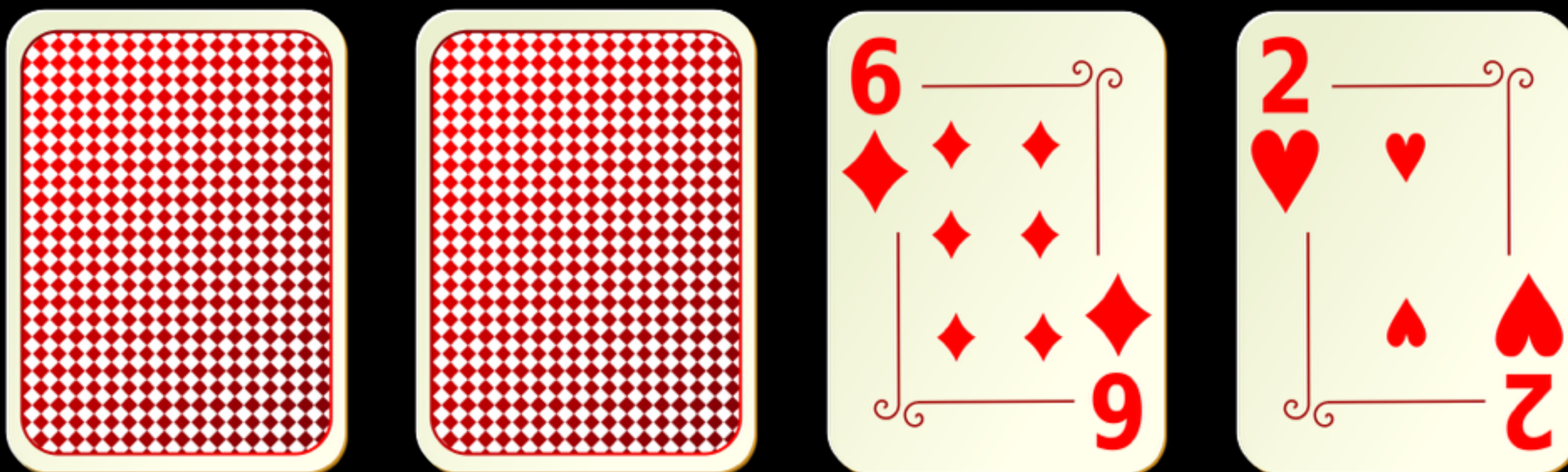
Grover's Algorithm



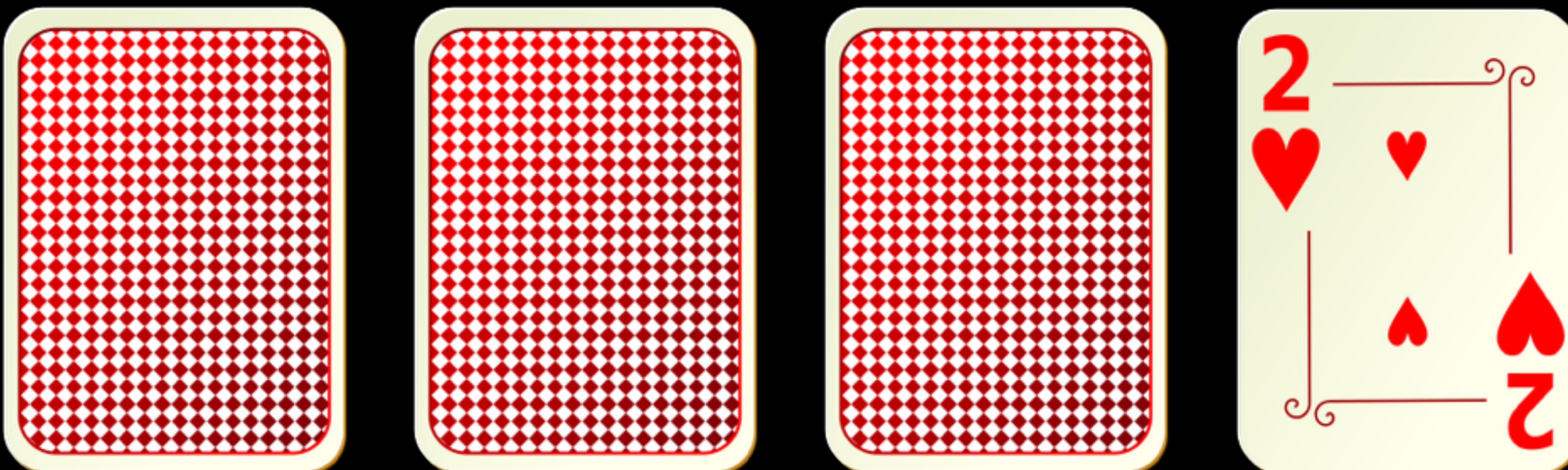
Grover's Algorithm



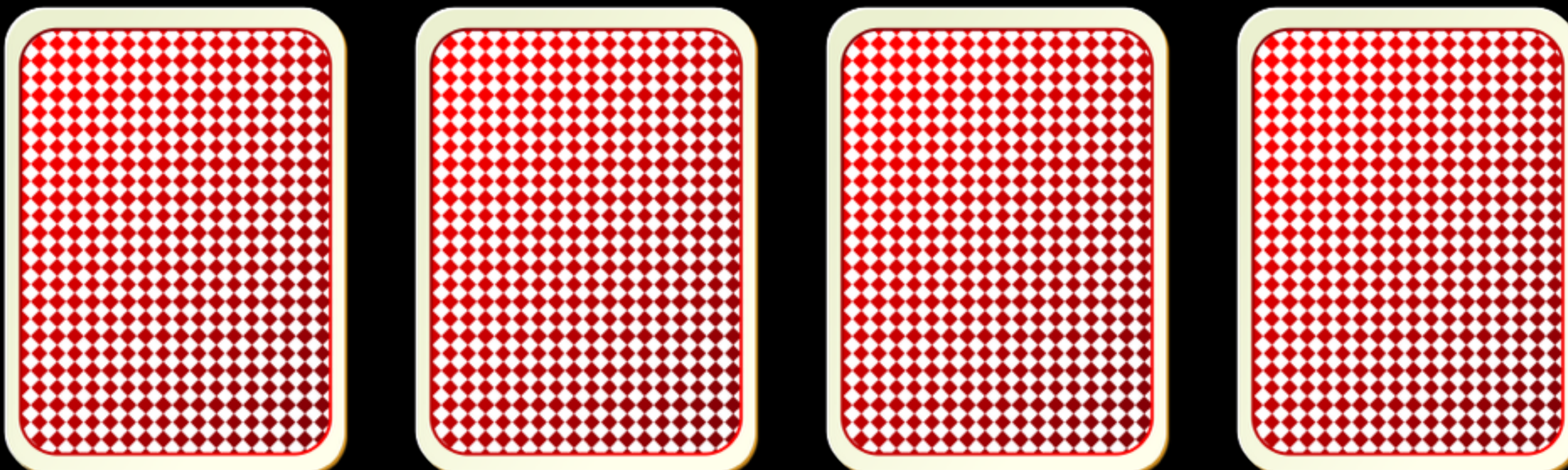
Grover's Algorithm



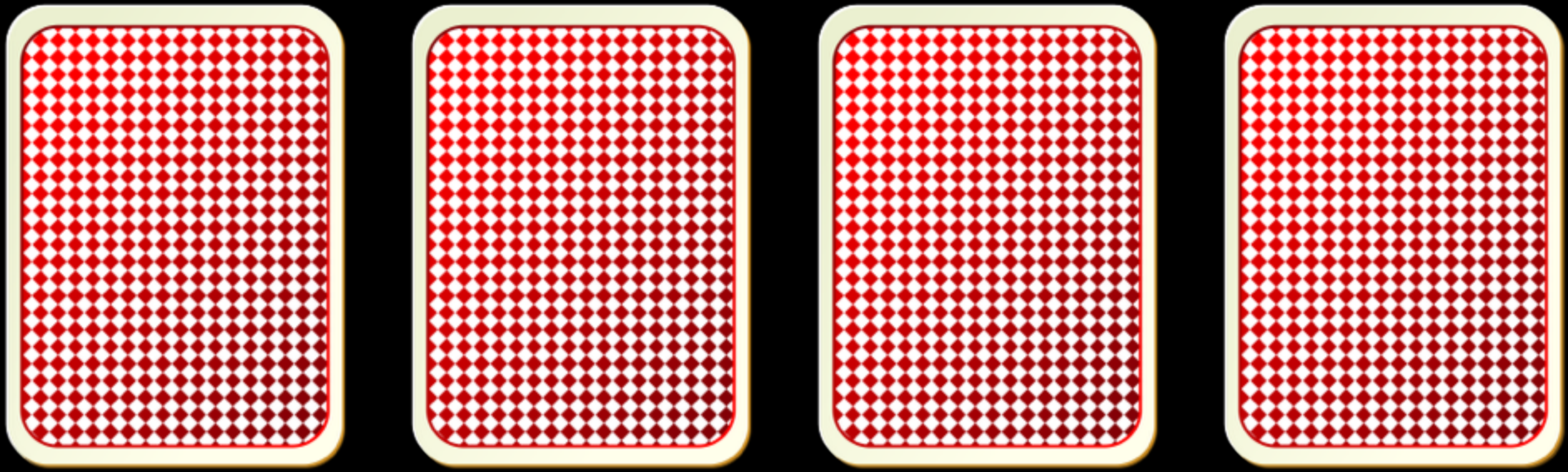
Grover's Algorithm



Grover's Algorithm



Grover's Algorithm



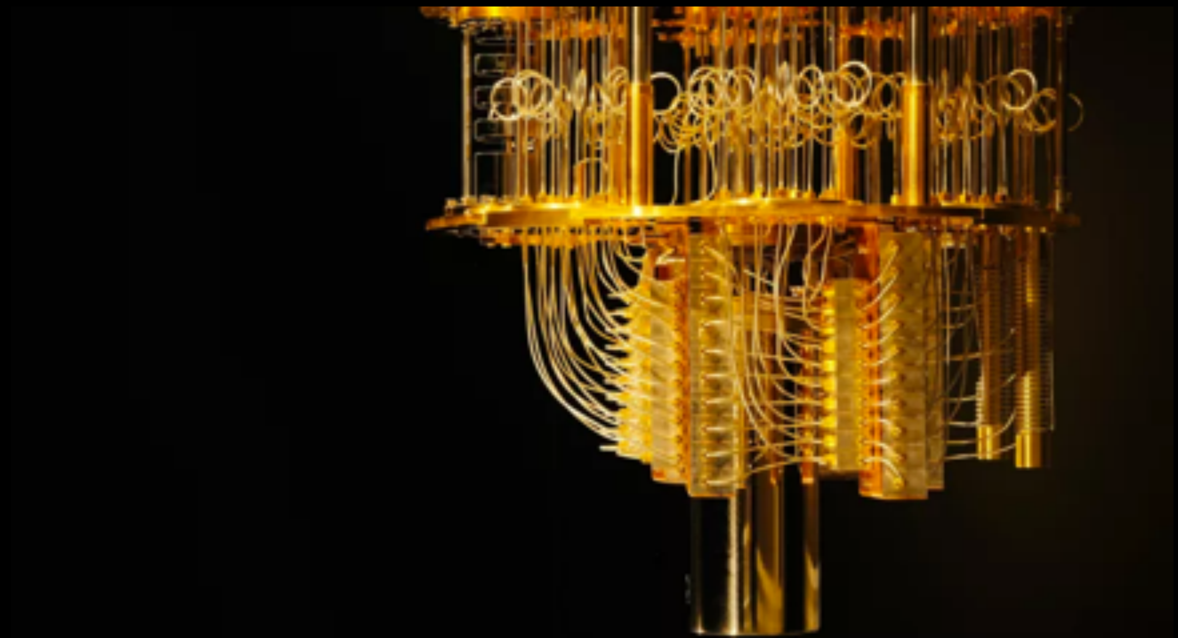


Photo: IBM Research

Grover's Algorithm

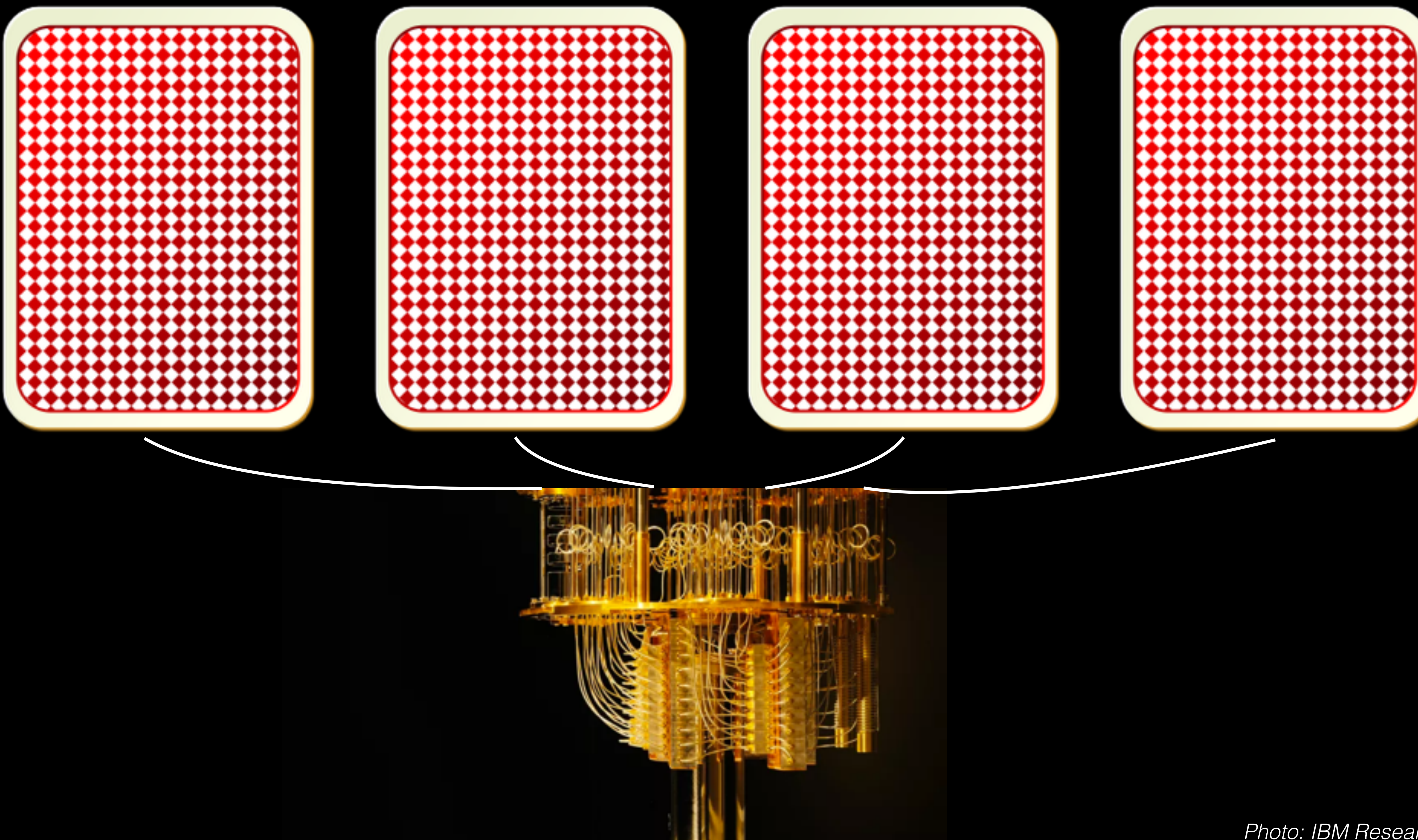


Photo: IBM Research

Grover's Algorithm

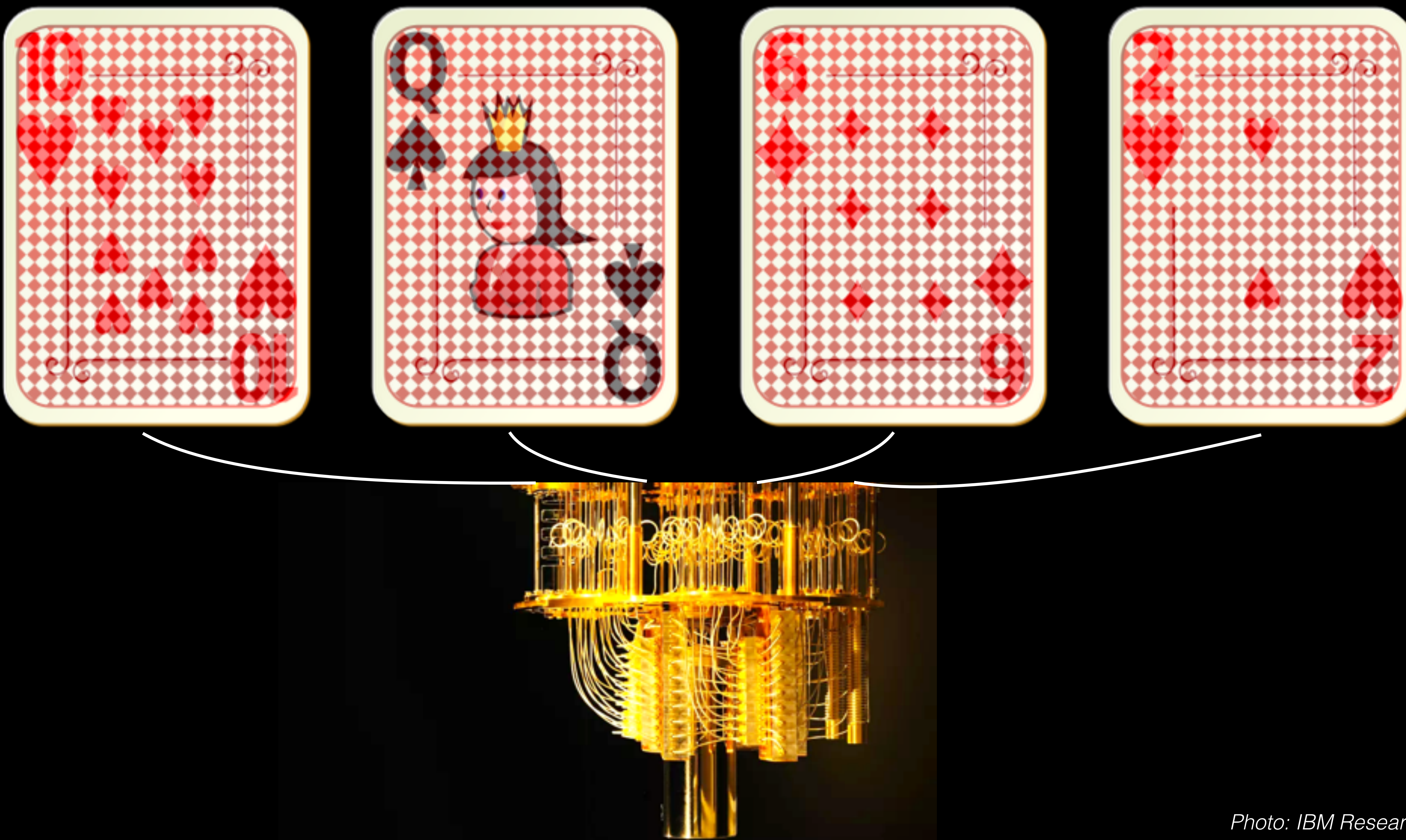
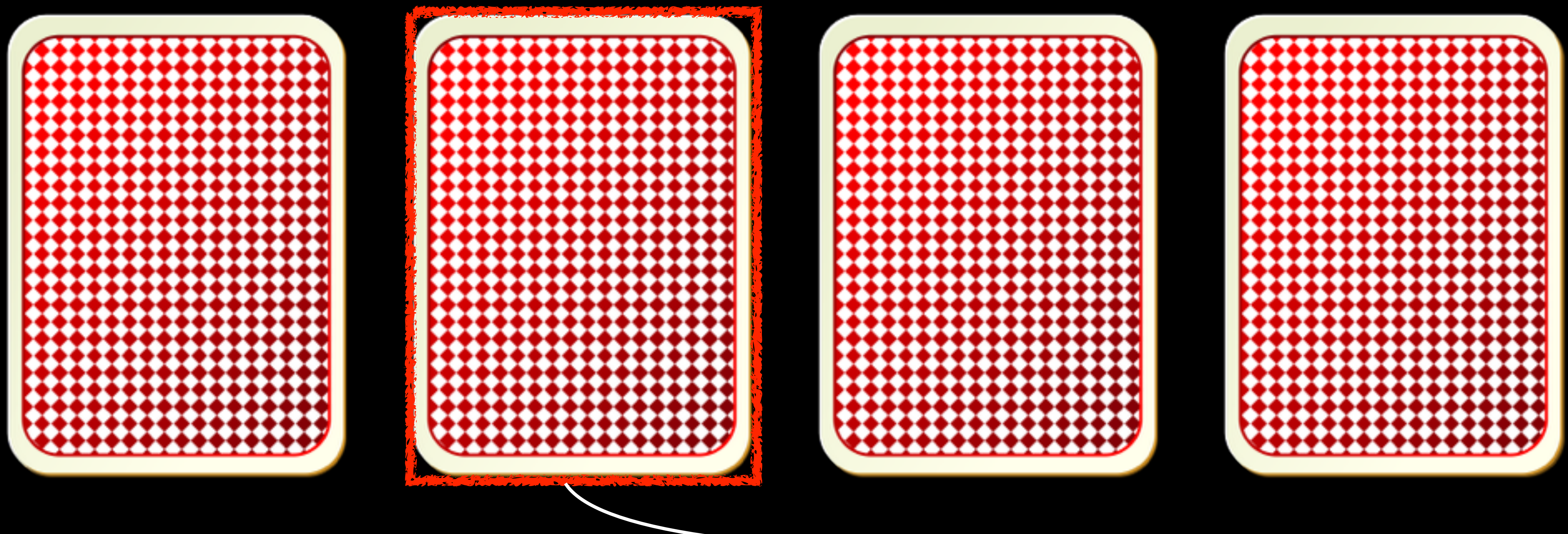


Photo: IBM Research

Grover's Algorithm



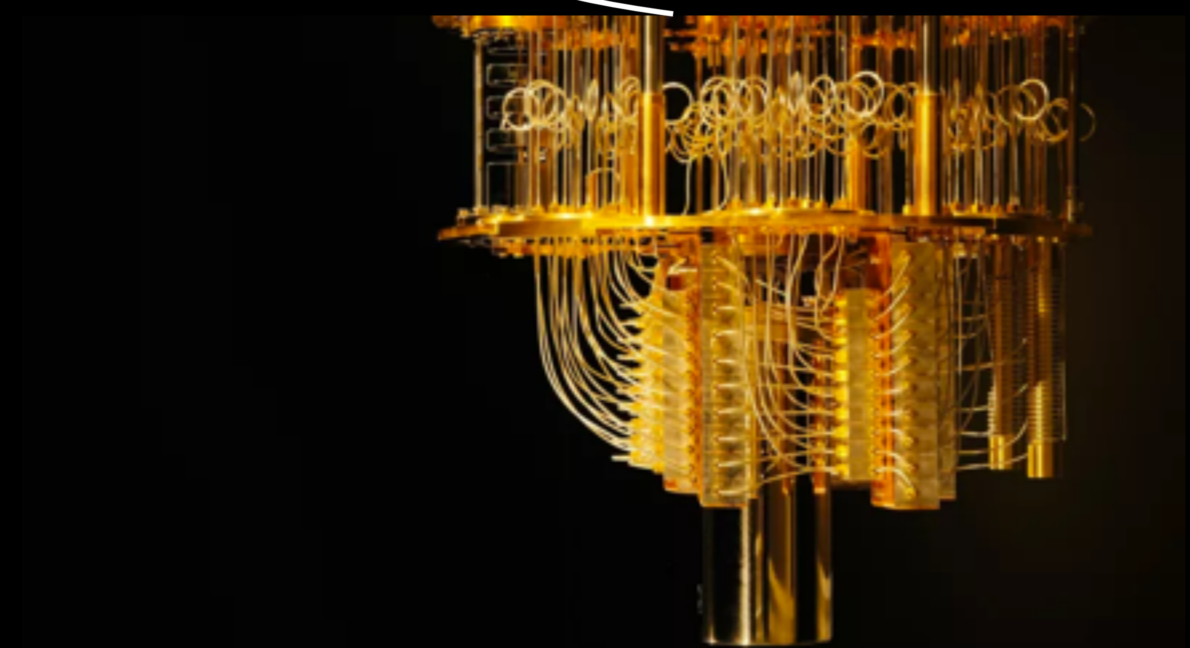
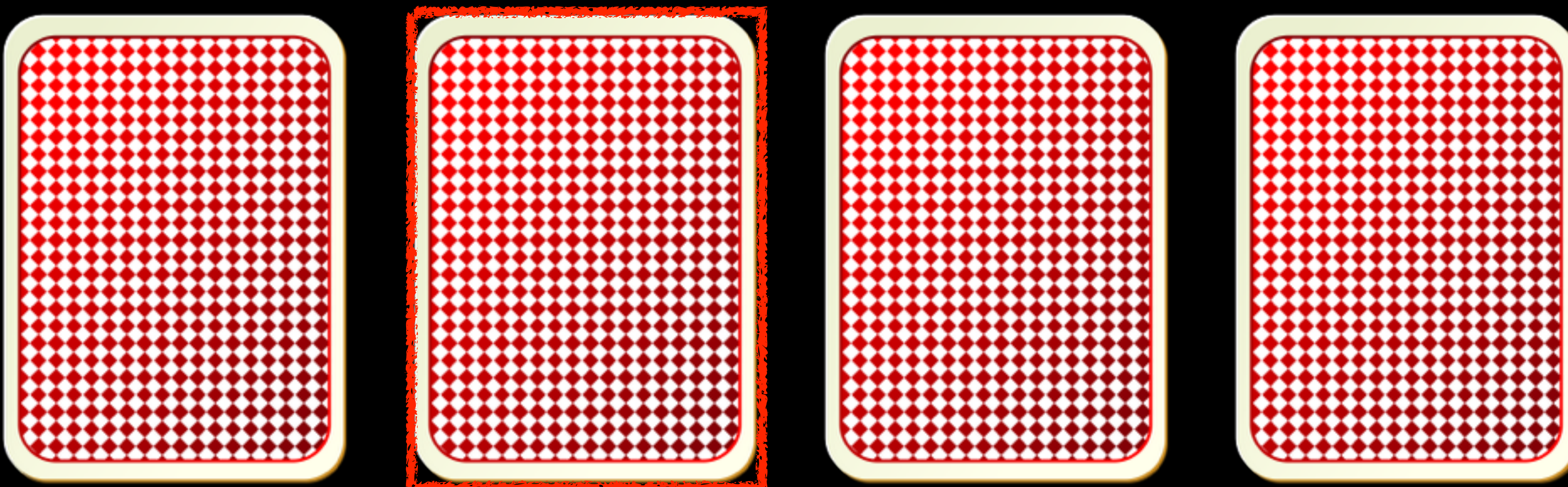
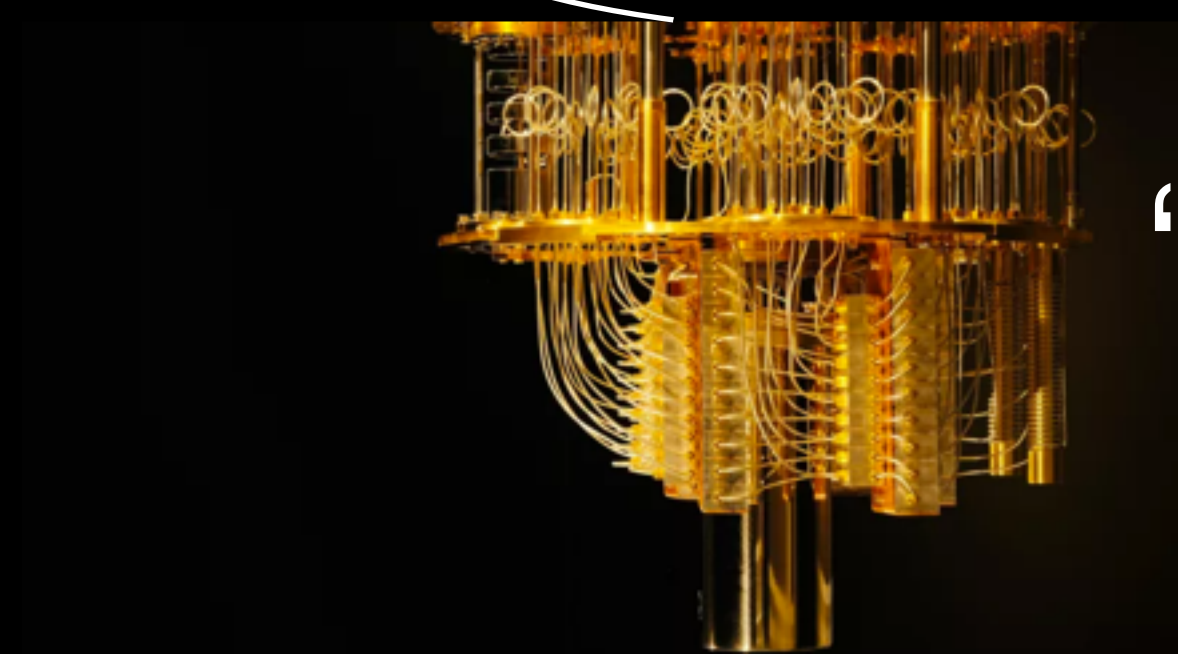


Photo: IBM Research

Grover's Algorithm





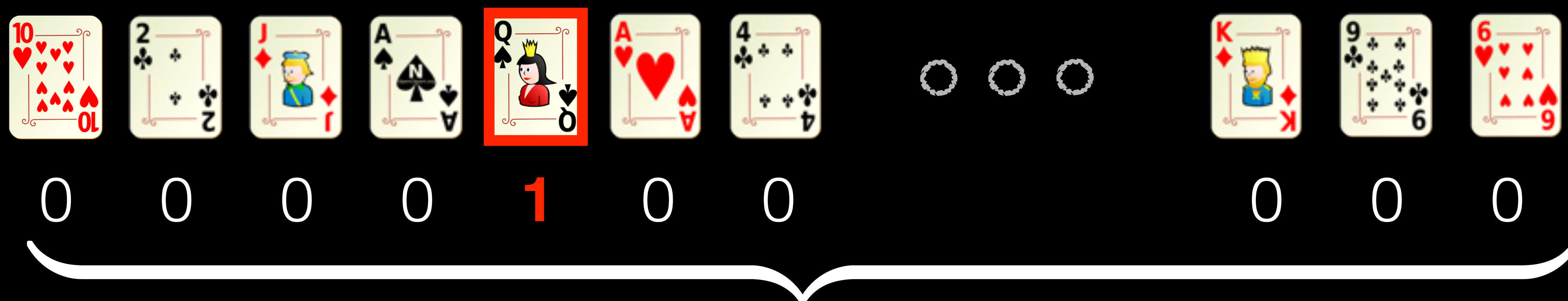
'Flips': 1

Photo: IBM Research

Solves the 'Unstructured Search' problem

Works in the black box / query setting

Find **1** marked item out of **N** items



N items

- Classical Computer: N queries

- Quantum Computer: $\sim\sqrt{N}$ queries

Reminder

Reminder

We represent quantum states using ket notation

Reminder

We represent quantum states using ket notation

$|\psi\rangle$ represents a column vector

Reminder

We represent quantum states using ket notation

$|\psi\rangle$ represents a column vector

$\langle\psi|$ is its complex conjugate — a row vector

Reminder

We represent quantum states using ket notation

$|\psi\rangle$ represents a column vector

$\langle\psi|$ is its complex conjugate — a row vector

We represent an integer x as the quantum state $|x\rangle$

Reminder

We represent quantum states using ket notation

$|\psi\rangle$ represents a column vector

$\langle\psi|$ is its complex conjugate — a row vector

We represent an integer x as the quantum state $|x\rangle$

e.g. $|5\rangle$

Reminder

We represent quantum states using ket notation

$|\psi\rangle$ represents a column vector

$\langle\psi|$ is its complex conjugate — a row vector

We represent an integer x as the quantum state $|x\rangle$

e.g. $|5\rangle$

The inner product between two states/vectors is written as

Reminder

We represent quantum states using ket notation

$|\psi\rangle$ represents a column vector

$\langle\psi|$ is its complex conjugate — a row vector

We represent an integer x as the quantum state $|x\rangle$

e.g. $|5\rangle$

The inner product between two states/vectors is written as

$$\langle\psi|\phi\rangle$$

Reminder

We represent quantum states using ket notation

$|\psi\rangle$ represents a column vector

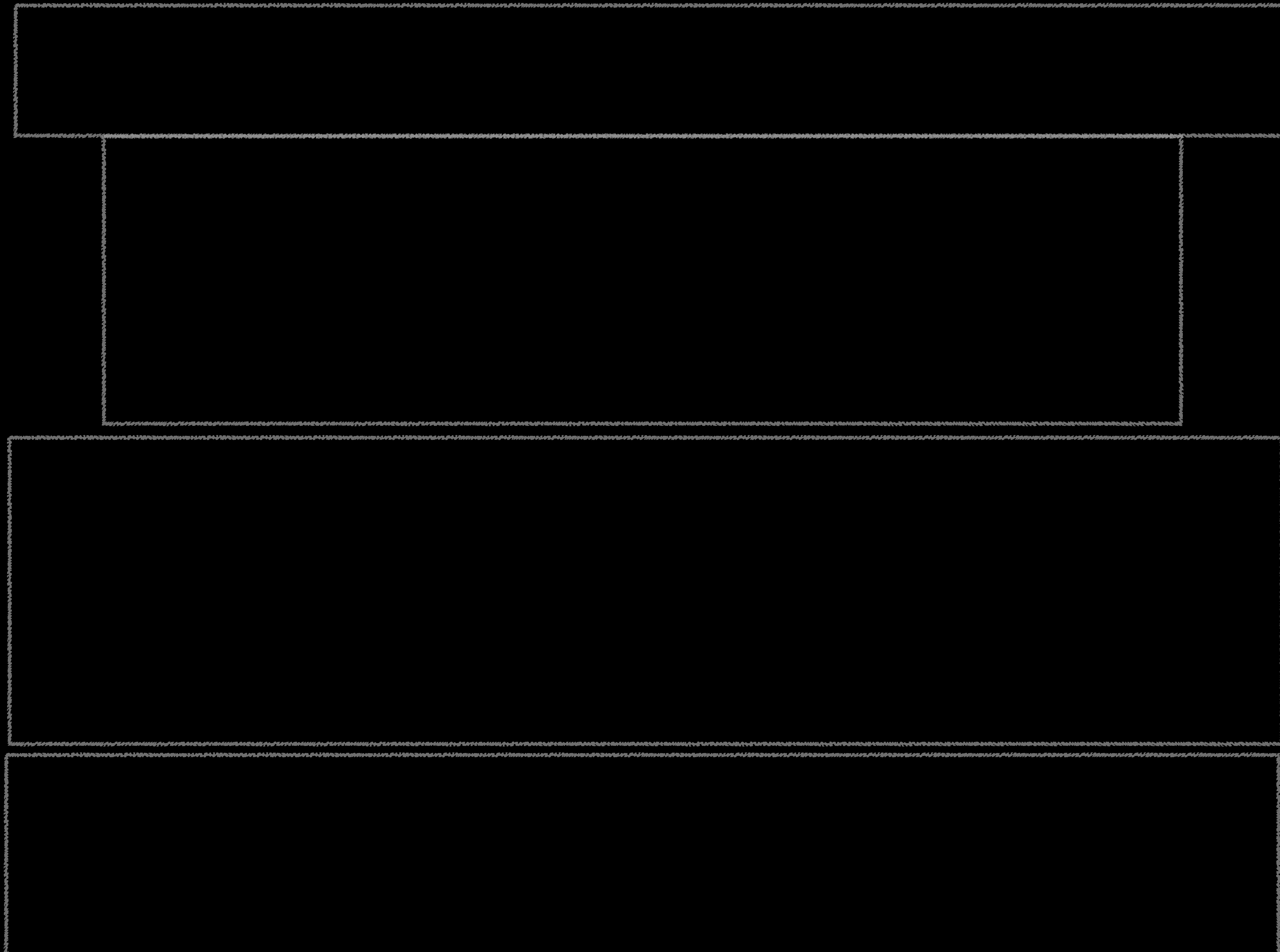
$\langle\psi|$ is its complex conjugate — a row vector

We represent an integer x as the quantum state $|x\rangle$

e.g. $|5\rangle$

The inner product between two states/vectors is written as

$$\langle\psi|\phi\rangle = \cos\theta$$



Operations on quantum states are via *unitary operators*

(Matrices that preserve the norm of the vector)

Operations on quantum states are via *unitary operators*

(Matrices that preserve the norm of the vector)

E.g. the Hadamard gate:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Operations on quantum states are via *unitary operators*

(Matrices that preserve the norm of the vector)

E.g. the Hadamard gate:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Operations on quantum states are via *unitary operators*

(Matrices that preserve the norm of the vector)

E.g. the Hadamard gate:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Operations on quantum states are via *unitary operators*

(Matrices that preserve the norm of the vector)

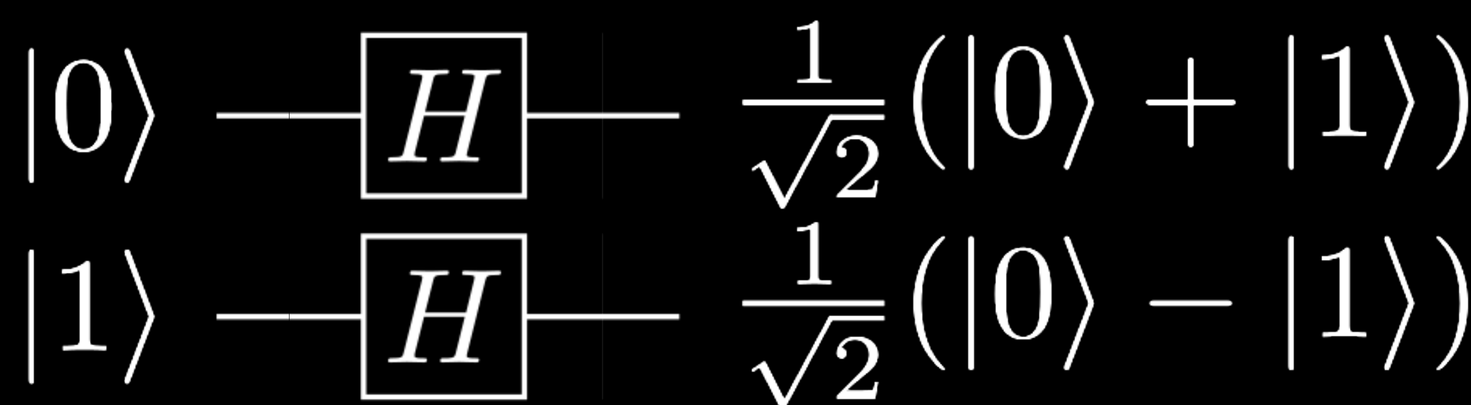
E.g. the Hadamard gate:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

We represent sequences of operations as circuits:



Operations on quantum states are via *unitary operators*

(Matrices that preserve the norm of the vector)

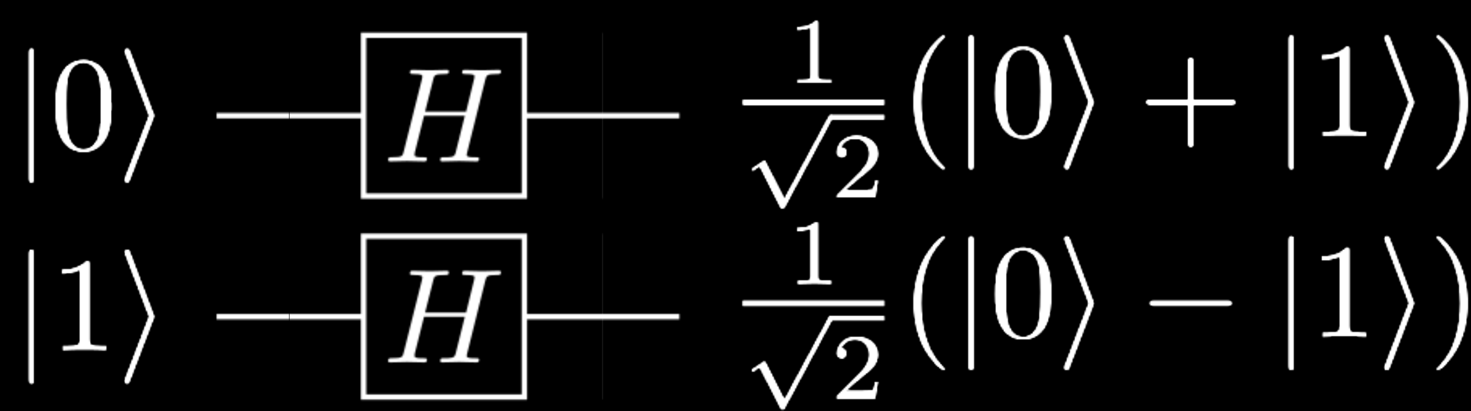
E.g. the Hadamard gate:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

We represent sequences of operations as circuits:



Given a state $|\psi\rangle$, the probability of measuring $|\phi\rangle$ is:

Operations on quantum states are via *unitary operators*

(Matrices that preserve the norm of the vector)

E.g. the Hadamard gate:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

We represent sequences of operations as circuits:

$$|0\rangle \; — \boxed{H} — \; \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$|1\rangle \; — \boxed{H} — \; \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Given a state $|\psi\rangle$, the probability of measuring $|\phi\rangle$ is:

$$\Pr[\text{measure } |\phi\rangle] = |\langle\phi|\psi\rangle|^2$$

Ingredients for Grover’s Algorithm:

Ingredients for Grover's Algorithm:

1. An 'Oracle'

$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

Ingredients for Grover's Algorithm:

1. An 'Oracle'

$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

2. An operator

$$U_0|x\rangle \mapsto \begin{cases} -|x\rangle & \text{if } x = 0^n \\ |x\rangle & \text{Otherwise} \end{cases}$$

Ingredients for Grover's Algorithm:

1. An 'Oracle'

$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

2. An operator

$$U_0|x\rangle \mapsto \begin{cases} -|x\rangle & \text{if } x = 0^n \\ |x\rangle & \text{Otherwise} \end{cases}$$

3. Some Hadamards

$$H^{\otimes n}$$

Hadamard gates on all n qubits

Ingredients:

1. An 'Oracle'

$$U_f|x\rangle \mapsto \begin{cases} -|x\rangle & \text{If } x \text{ is the marked item} \\ |x\rangle & \text{Otherwise} \end{cases}$$

2. An operator

$$U_0|x\rangle \mapsto \begin{cases} -|x\rangle & \text{If } x = 0^n \\ |x\rangle & \text{Otherwise} \end{cases}$$

3. Some Hadamards

$$H^{\otimes n}$$

Ingredients:

1. An 'Oracle'

$$U_f|x\rangle \mapsto \begin{cases} -|x\rangle & \text{If } x \text{ is the marked item} \\ |x\rangle & \text{Otherwise} \end{cases}$$

2. An operator

$$U_0|x\rangle \mapsto \begin{cases} -|x\rangle & \text{If } x = 0^n \\ |x\rangle & \text{Otherwise} \end{cases}$$

3. Some Hadamards

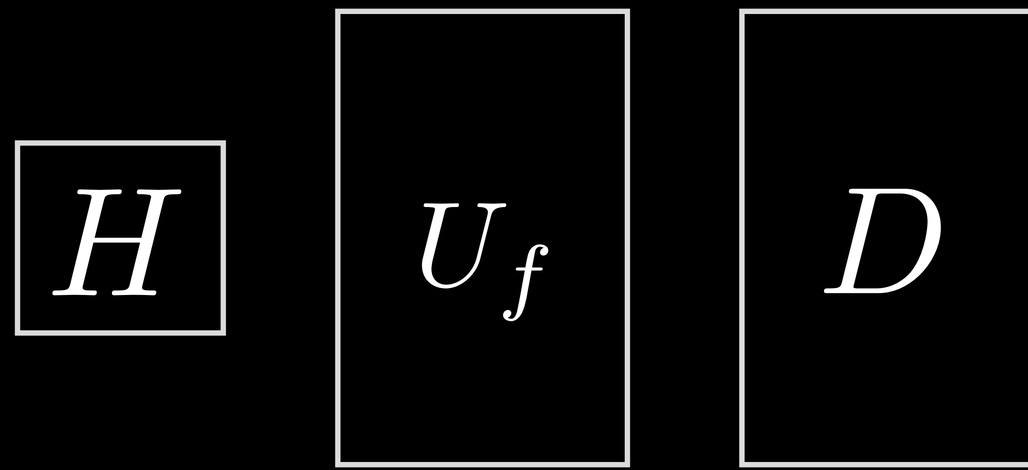
$$H^{\otimes n}$$

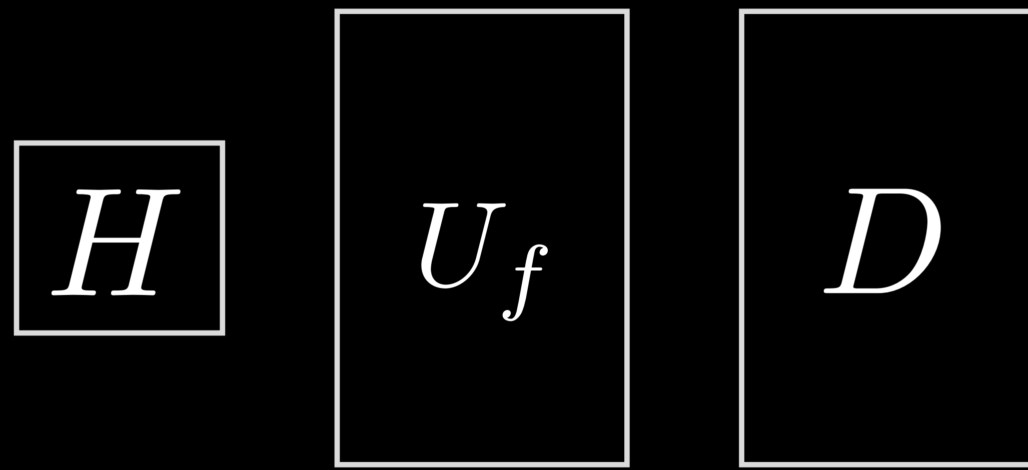
Using U_0 we define the operator

$$D = H^{\otimes n} U_0 H^{\otimes n}$$

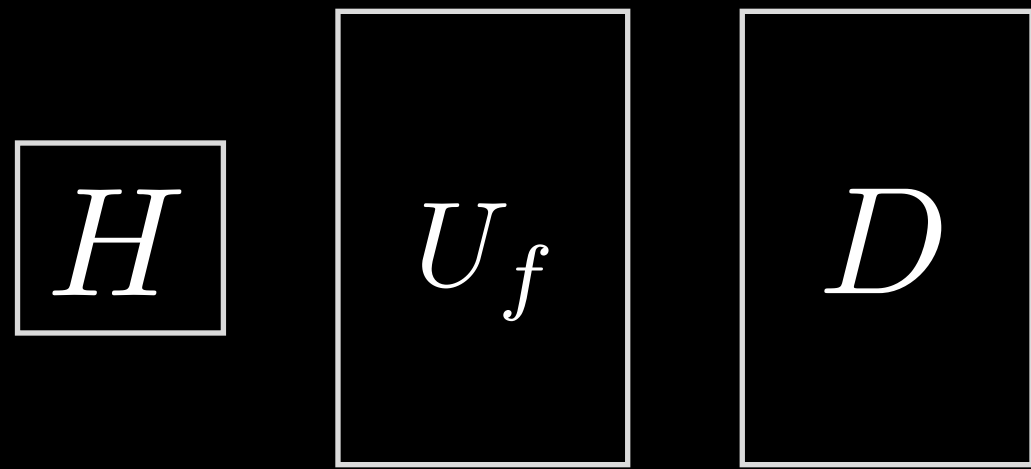
H


H
U_f

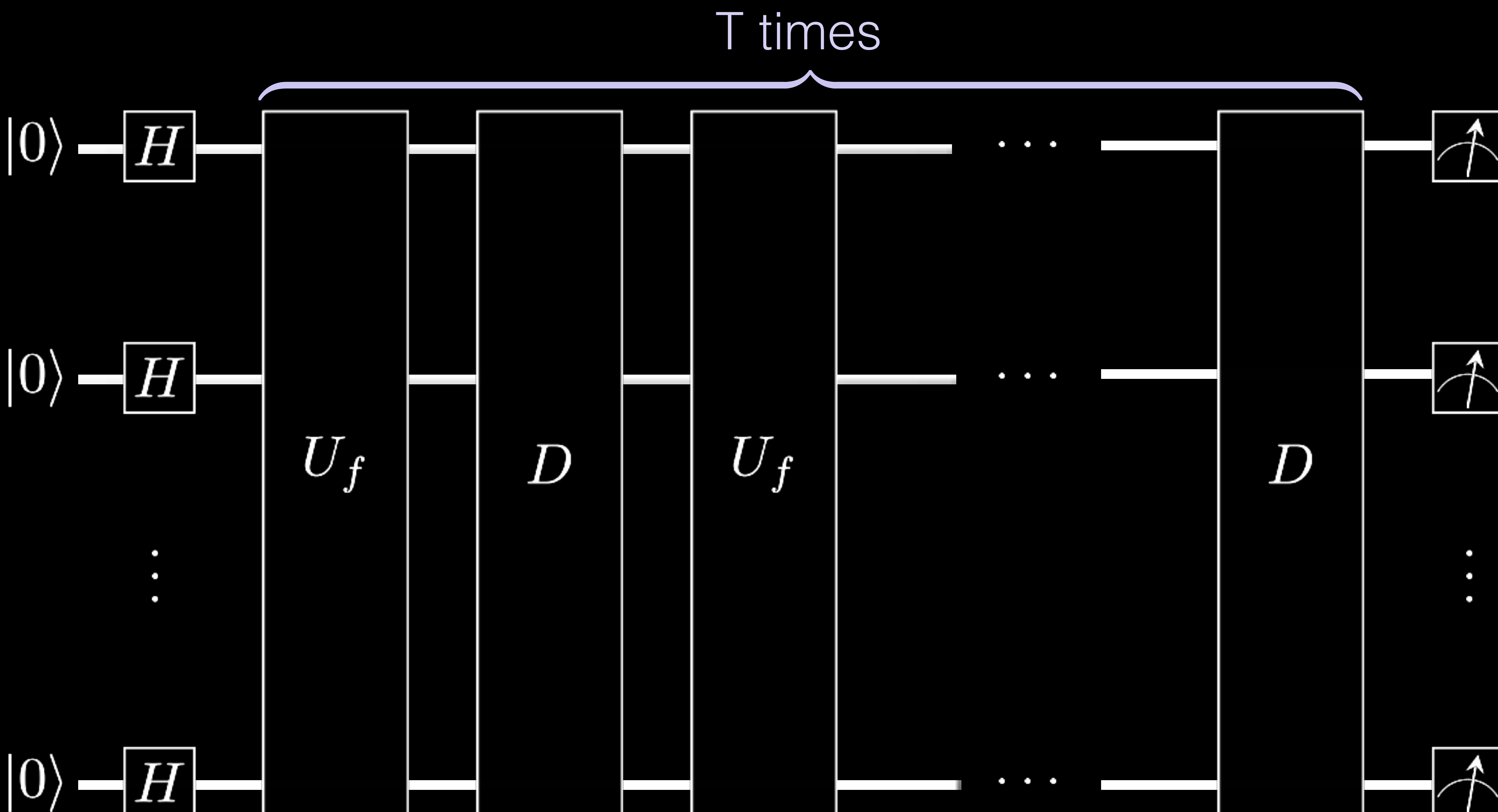

H
U_f
D



$$D = H^{\otimes n} U_0 H^{\otimes n}$$



$$D = H^{\otimes n} U_0 H^{\otimes n}$$



Why does this work?

Why does this work?

U_f and D are **reflections**

Why does this work?

U_f and D are **reflections**

Why does this work?

U_f and D are **reflections**

$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

Why does this work?

U_f and D are **reflections**

$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

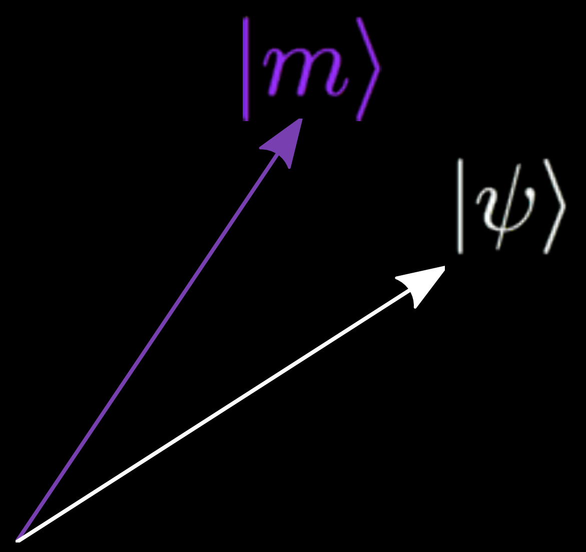
Reflection about the marked item $|m\rangle$

Why does this work?

U_f and D are **reflections**

$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

Reflection about the marked item $|m\rangle$

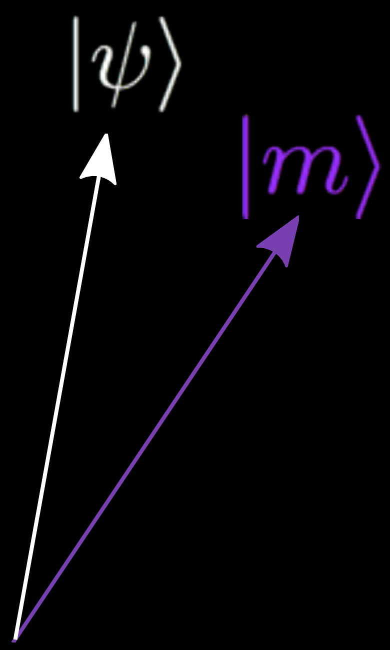


Why does this work?

U_f and D are **reflections**

$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

Reflection about the marked item $|m\rangle$

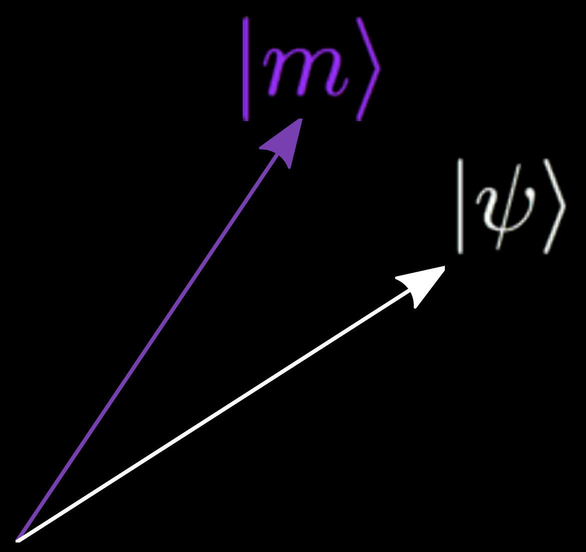


Why does this work?

U_f and D are **reflections**

$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

Reflection about the marked item $|m\rangle$

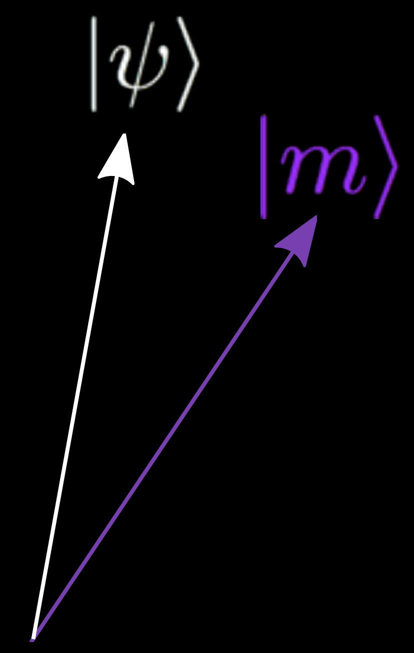


Why does this work?

U_f and D are **reflections**

$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

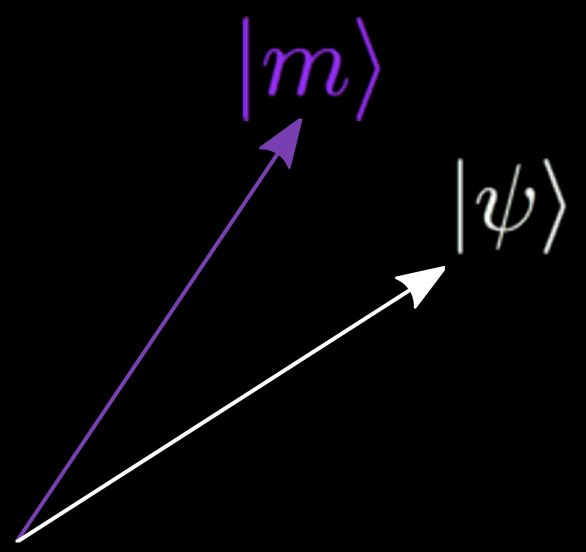
Reflection about the marked item $|m\rangle$



Why does this work?

U_f and D are **reflections**

$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$



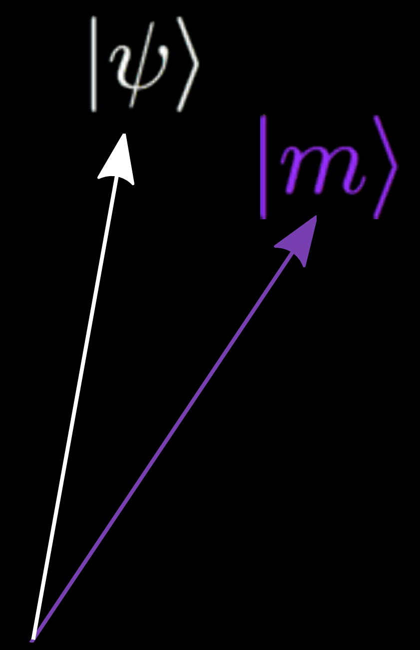
Reflection about the marked item $|m\rangle$

Why does this work?

U_f and D are **reflections**

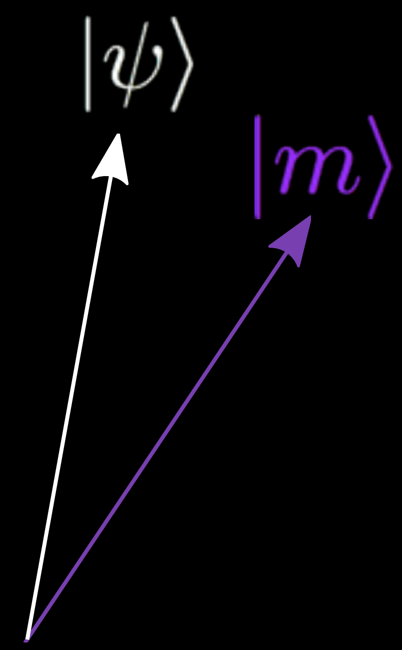
$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

Reflection about the marked item $|m\rangle$



Why does this work?

U_f and D are **reflections**



$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

Reflection about the marked item $|m\rangle$

Why does this work?

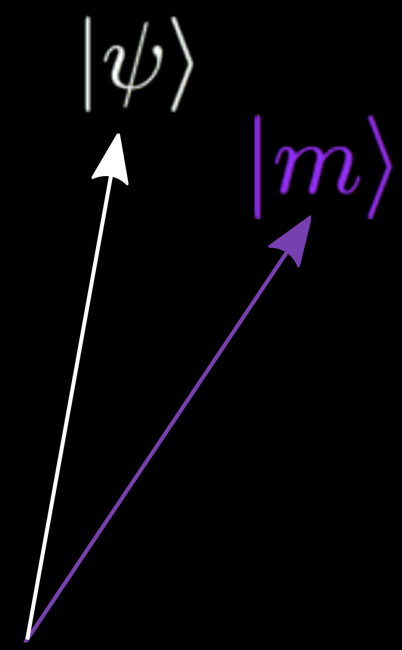
U_f and D are **reflections**

$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

Reflection about the marked item $|m\rangle$

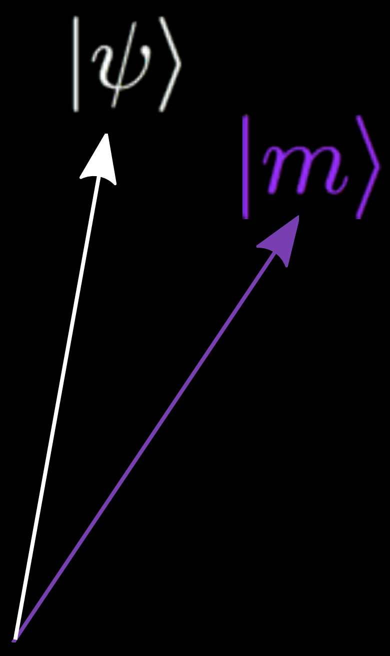
$$D = \quad H^{\otimes n} U_0 H^{\otimes n}$$

$$U_0|x\rangle \mapsto \begin{matrix} -|x\rangle & \text{If } x = 0^n \\ |x\rangle & \text{Otherwise} \end{matrix}$$



Why does this work?

U_f and D are **reflections**



$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

Reflection about the marked item $|m\rangle$

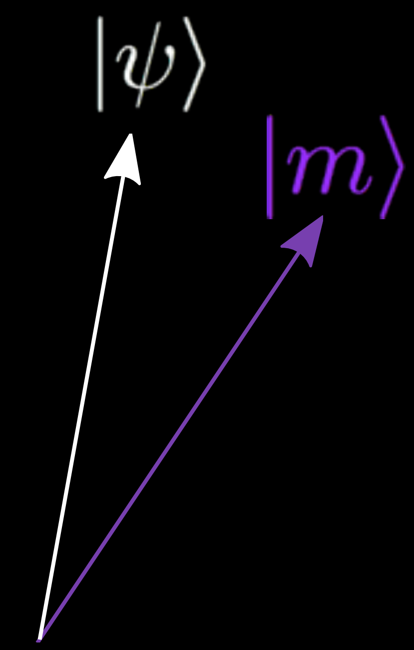
$$H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2^n}} \sum_x |x\rangle = |+\rangle$$

$$D = \quad H^{\otimes n} U_0 H^{\otimes n}$$

$$U_0|x\rangle \mapsto \begin{matrix} -|x\rangle & \text{If } x = 0^n \\ |x\rangle & \text{Otherwise} \end{matrix}$$

Why does this work?

U_f and D are **reflections**



$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

Reflection about the marked item $|m\rangle$

$$H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2^n}}\sum_x |x\rangle = |+\rangle$$

$$D = H^{\otimes n} U_0 H^{\otimes n}$$

$$U_0|x\rangle \mapsto \begin{matrix} -|x\rangle & \text{If } x = 0^n \\ |x\rangle & \text{Otherwise} \end{matrix}$$

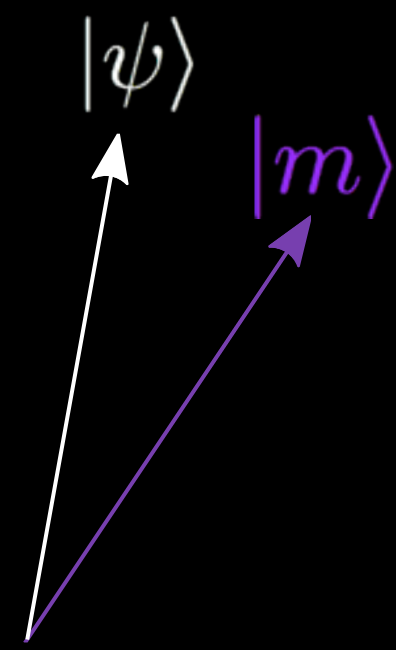
Reflection about the state orthogonal to the uniform superposition, $|+^\perp\rangle$

Why does this work?

U_f and D are **reflections**

$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

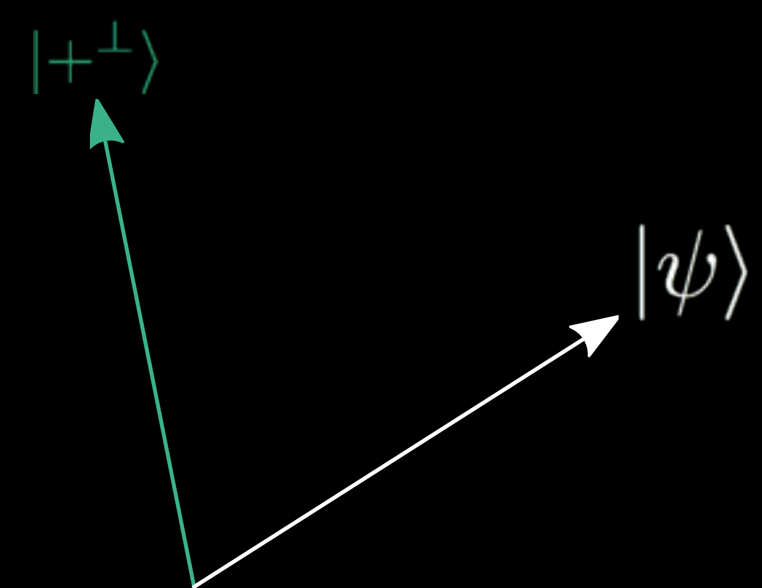
Reflection about the marked item $|m\rangle$



$$H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2^n}}\sum_x |x\rangle = |+\rangle$$

$$D = \quad H^{\otimes n} U_0 H^{\otimes n}$$

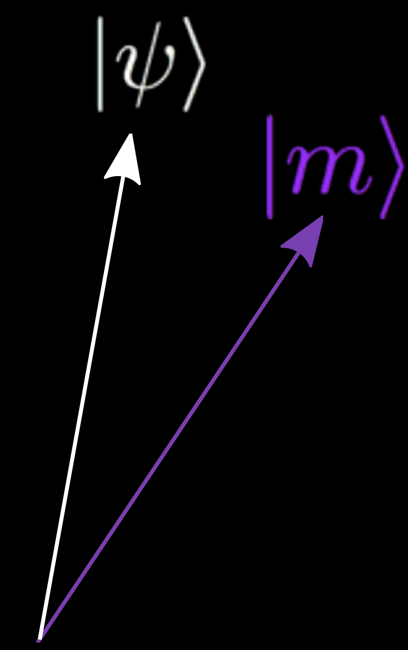
$$U_0|x\rangle \mapsto \begin{matrix} -|x\rangle & \text{If } x = 0^n \\ |x\rangle & \text{Otherwise} \end{matrix}$$



Reflection about the state orthogonal to the uniform superposition, $|+^{\perp}\rangle$

Why does this work?

U_f and D are **reflections**



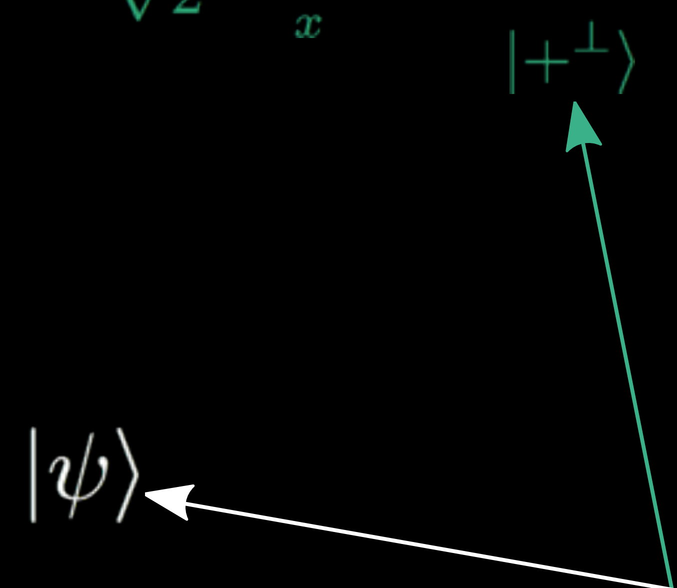
$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

Reflection about the marked item $|m\rangle$

$$H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2^n}}\sum_x |x\rangle = |+\rangle$$

$$D = H^{\otimes n} U_0 H^{\otimes n}$$

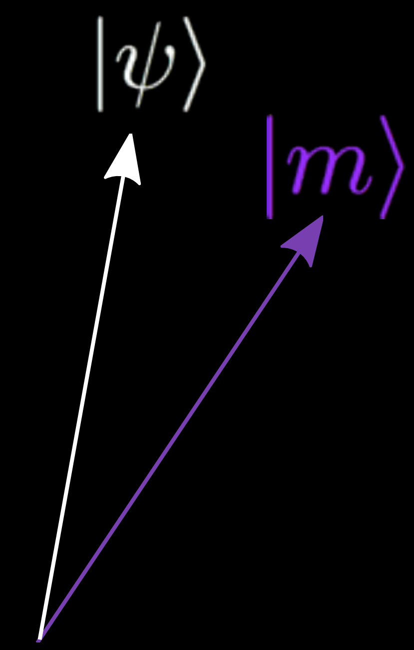
$$U_0|x\rangle \mapsto \begin{matrix} -|x\rangle & \text{If } x = 0^n \\ |x\rangle & \text{Otherwise} \end{matrix}$$



Reflection about the state orthogonal to the uniform superposition, $|+^{\perp}\rangle$

Why does this work?

U_f and D are **reflections**



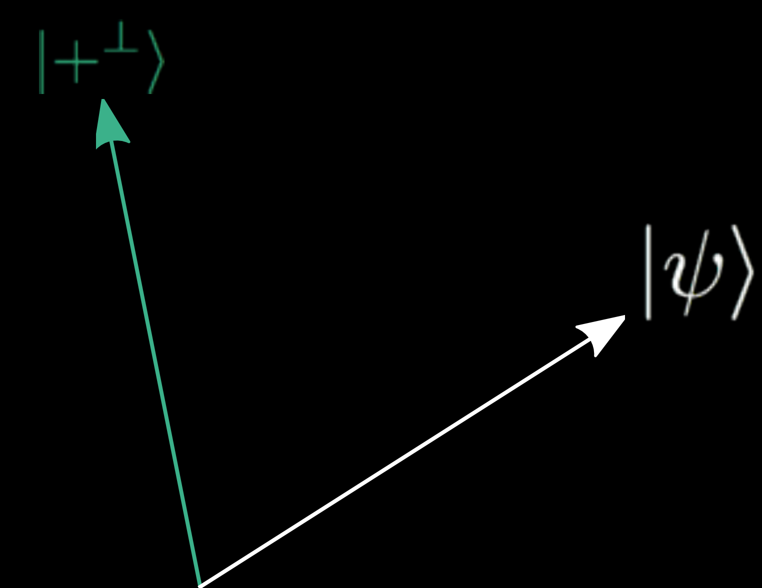
$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

Reflection about the marked item $|m\rangle$

$$H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2^n}}\sum_x |x\rangle = |+\rangle$$

$$D = H^{\otimes n} U_0 H^{\otimes n}$$

$$U_0|x\rangle \mapsto \begin{matrix} -|x\rangle & \text{If } x = 0^n \\ |x\rangle & \text{Otherwise} \end{matrix}$$



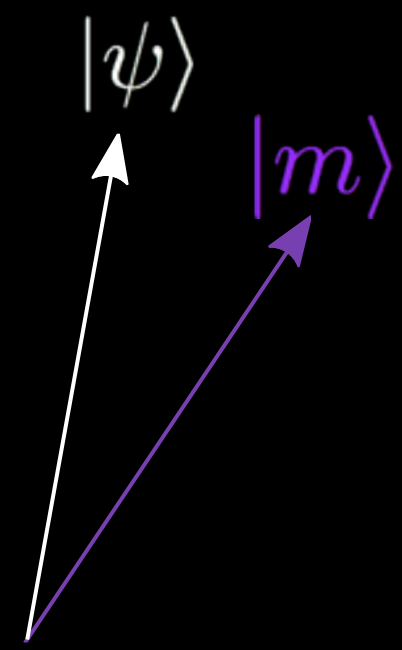
Reflection about the state orthogonal to the uniform superposition, $|+^\perp\rangle$

Why does this work?

U_f and D are **reflections**

$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

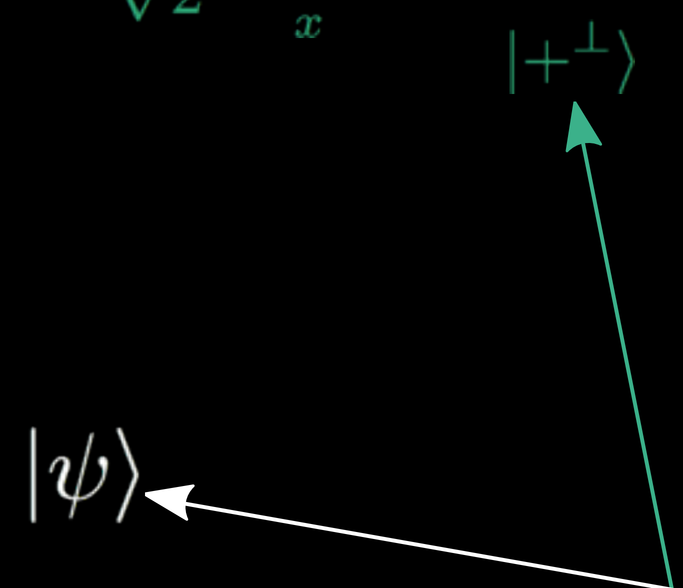
Reflection about the marked item $|m\rangle$



$$H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2^n}}\sum_x |x\rangle = |+\rangle$$

$$D = H^{\otimes n} U_0 H^{\otimes n}$$

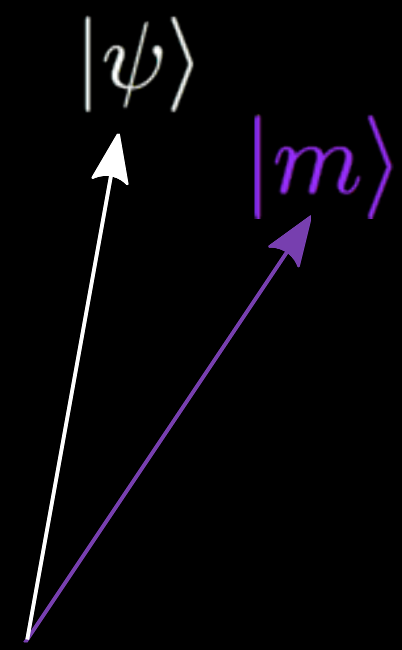
$$U_0|x\rangle \mapsto \begin{matrix} -|x\rangle & \text{If } x = 0^n \\ |x\rangle & \text{Otherwise} \end{matrix}$$



Reflection about the state orthogonal to the uniform superposition, $|+^\perp\rangle$

Why does this work?

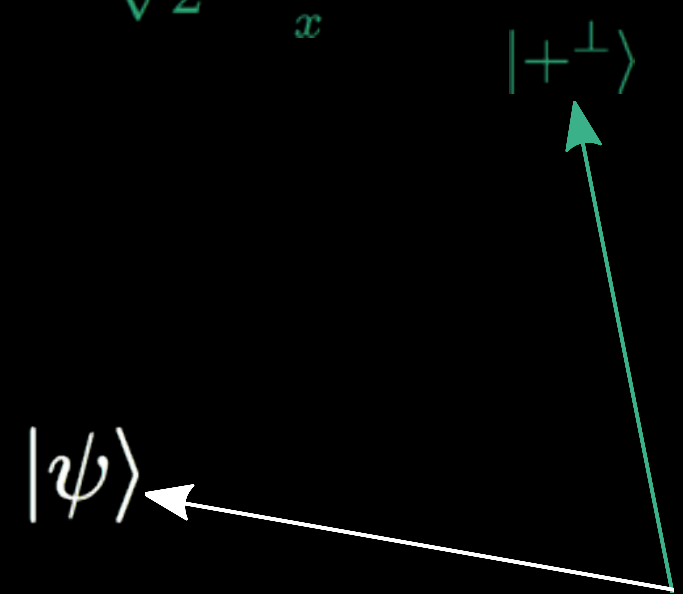
U_f and D are **reflections**



$$U_f|x\rangle \mapsto \begin{cases} |x\rangle & \text{if } x \text{ is the marked item} \\ -|x\rangle & \text{Otherwise} \end{cases}$$

Reflection about the marked item $|m\rangle$

$$H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2^n}}\sum_x |x\rangle = |+\rangle$$

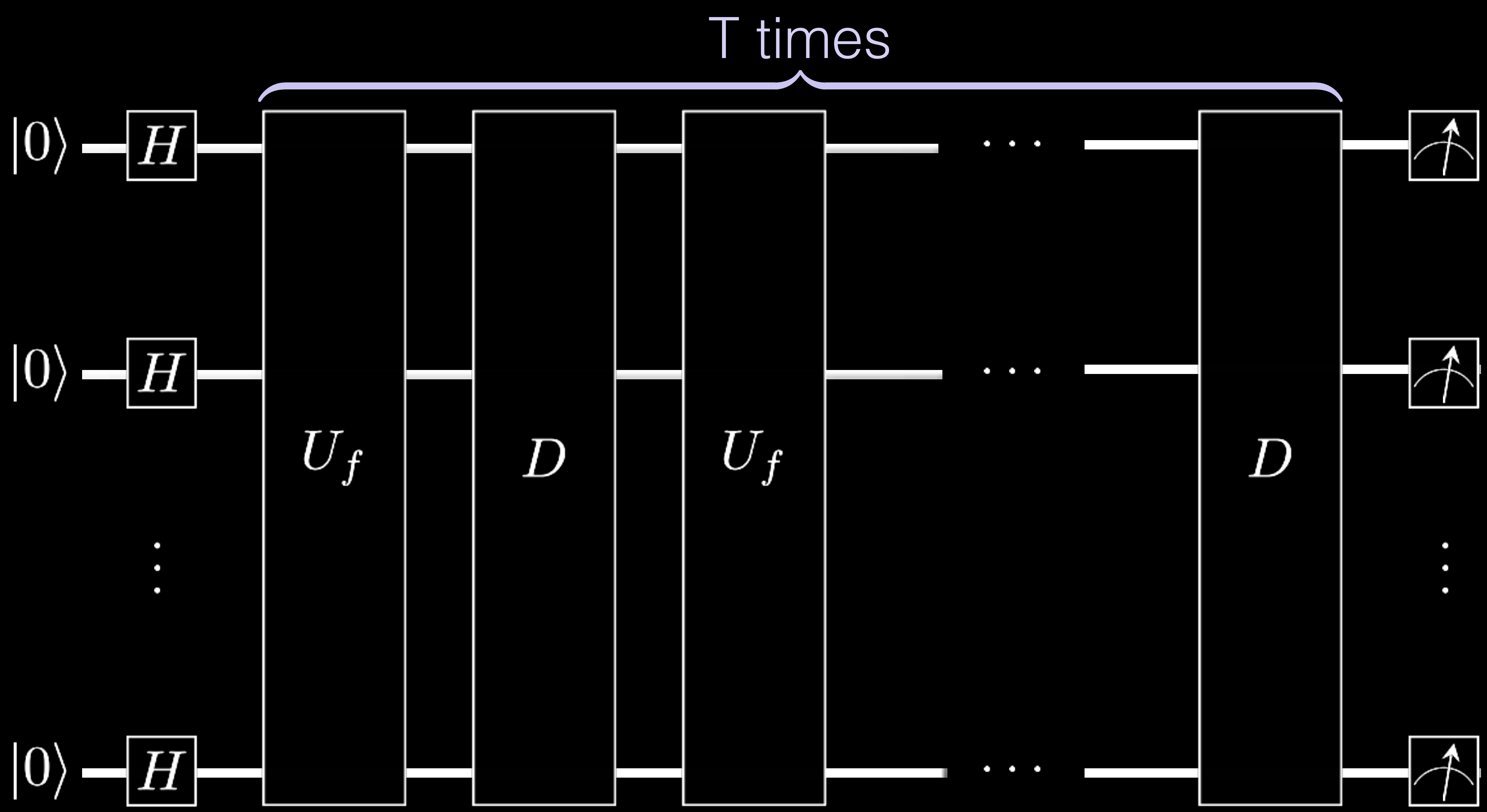


$$D = H^{\otimes n} U_0 H^{\otimes n}$$

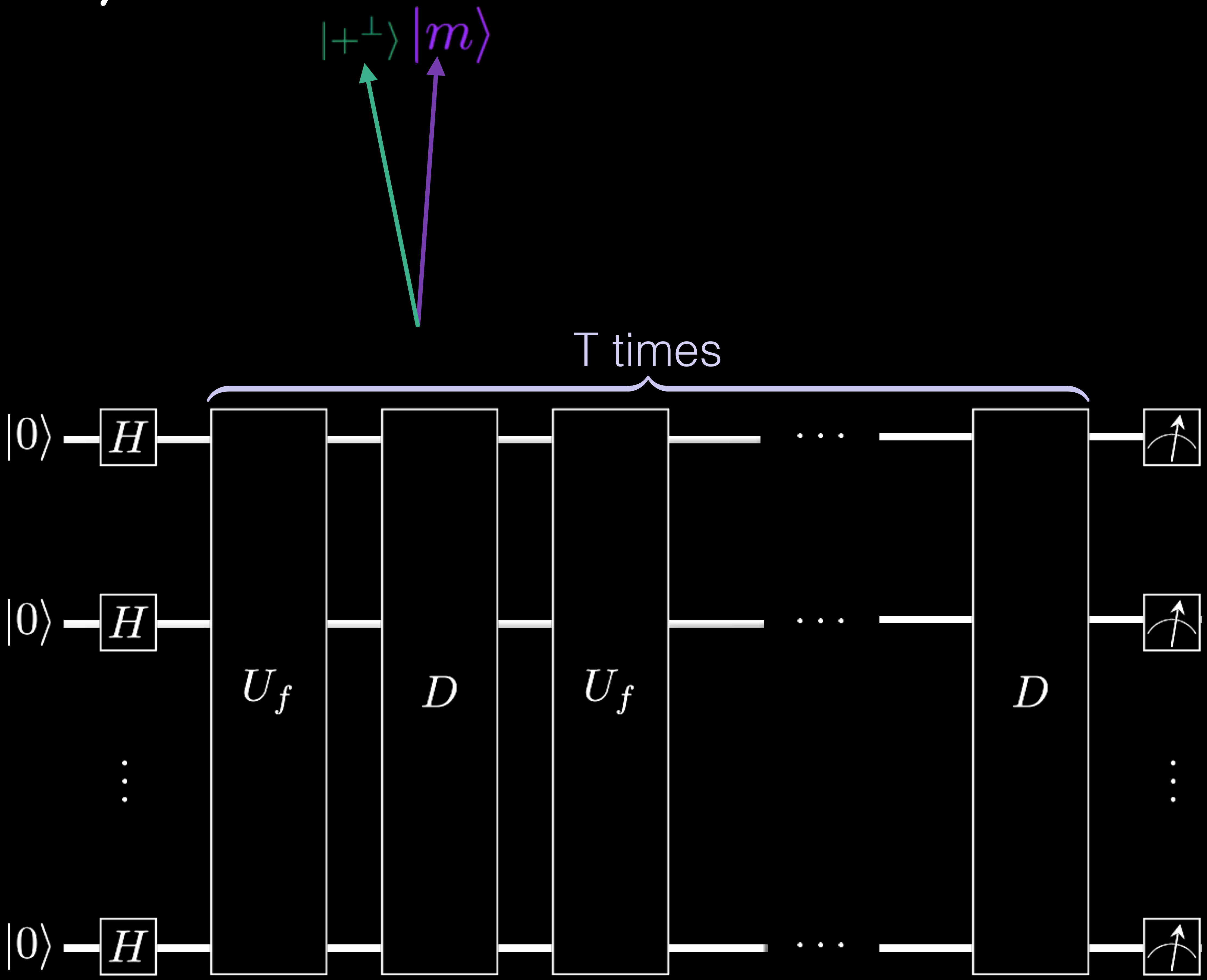
$$U_0|x\rangle \mapsto \begin{matrix} -|x\rangle & \text{If } x = 0^n \\ |x\rangle & \text{Otherwise} \end{matrix}$$

Reflection about the state orthogonal to the uniform superposition, $|+^{\perp}\rangle$

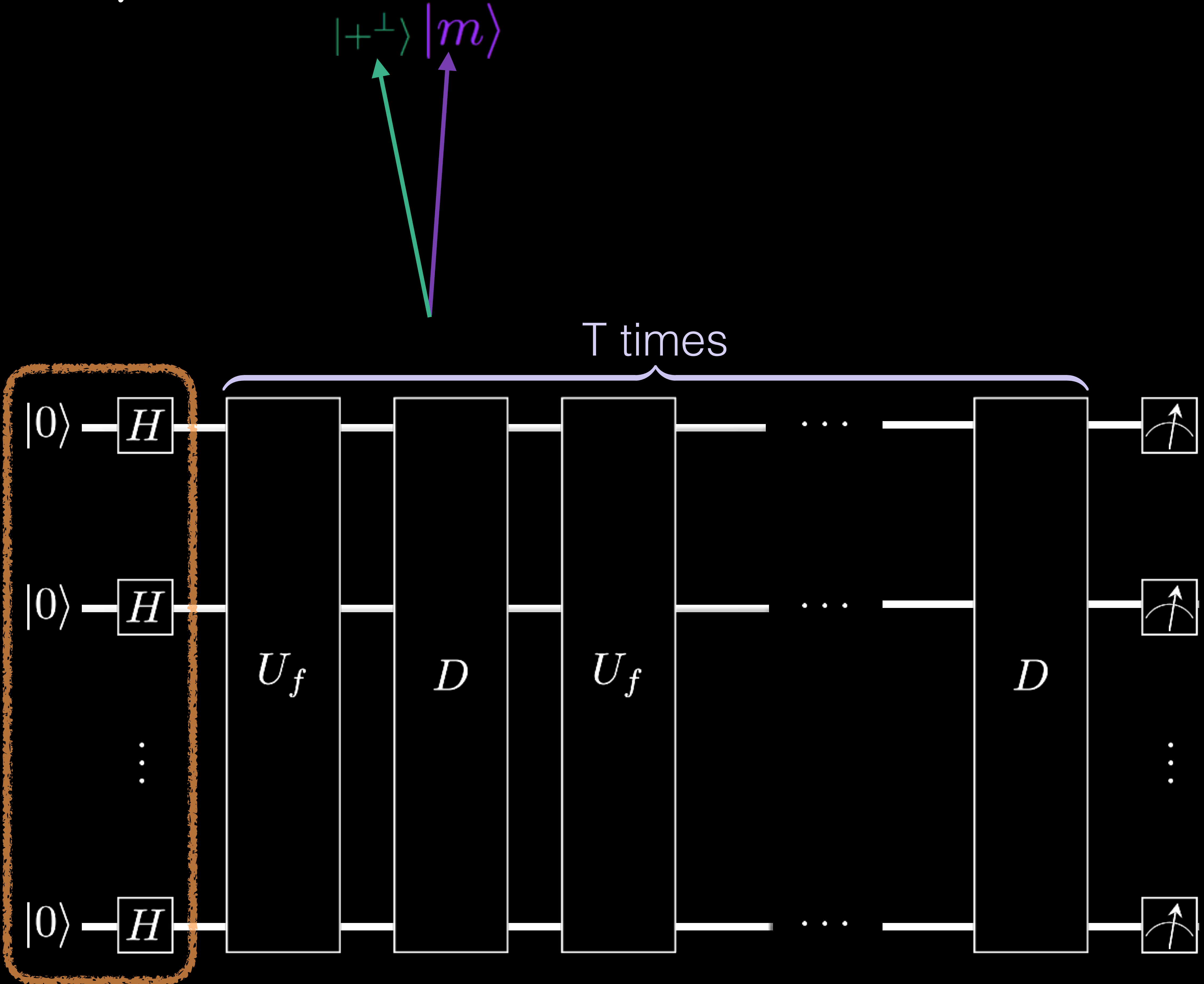
Why does this work?



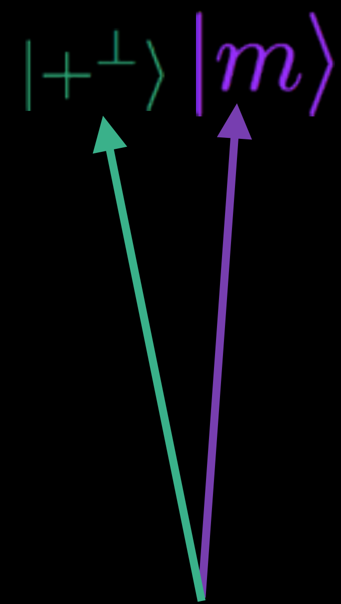
Why does this work?



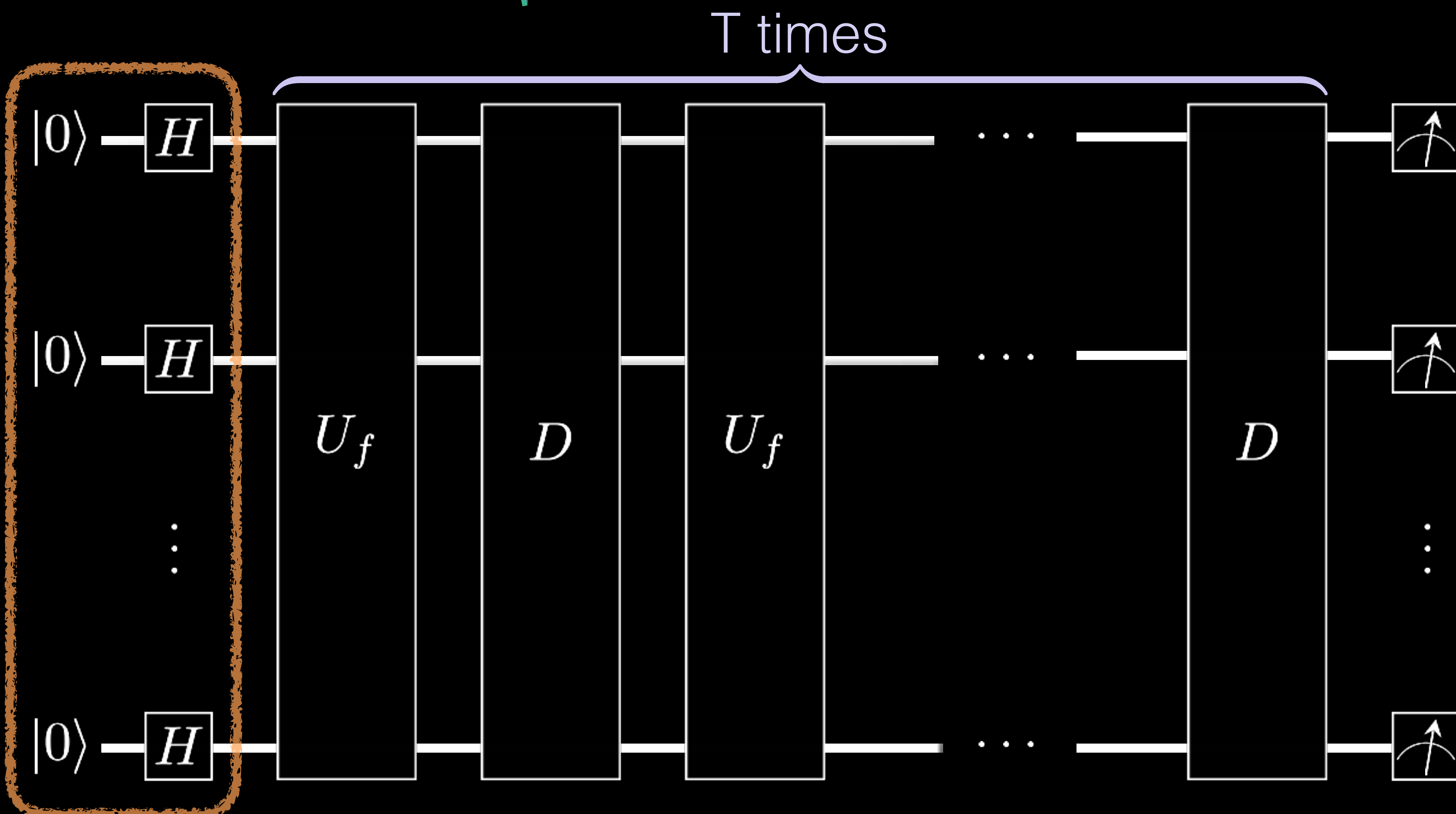
Why does this work?



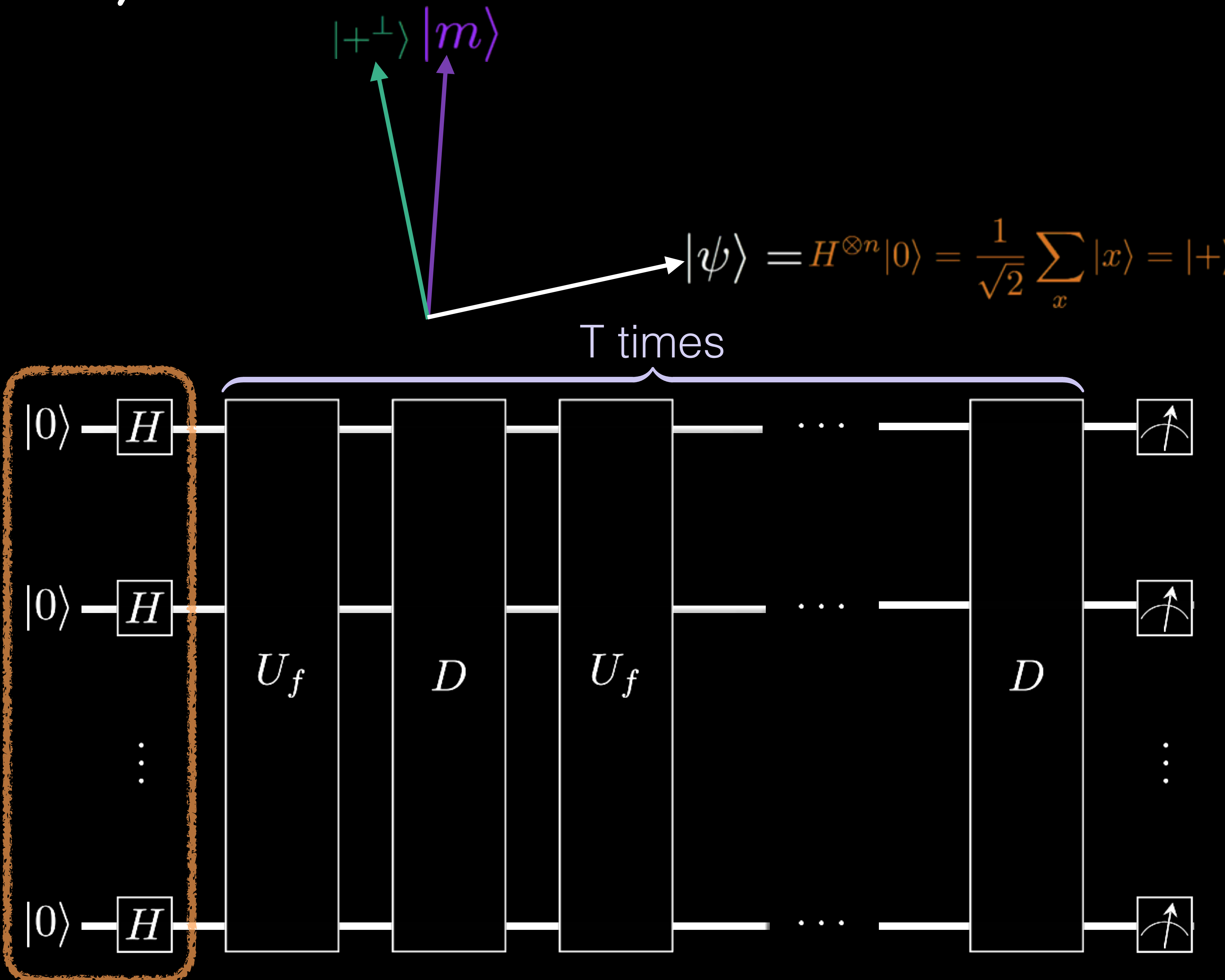
Why does this work?



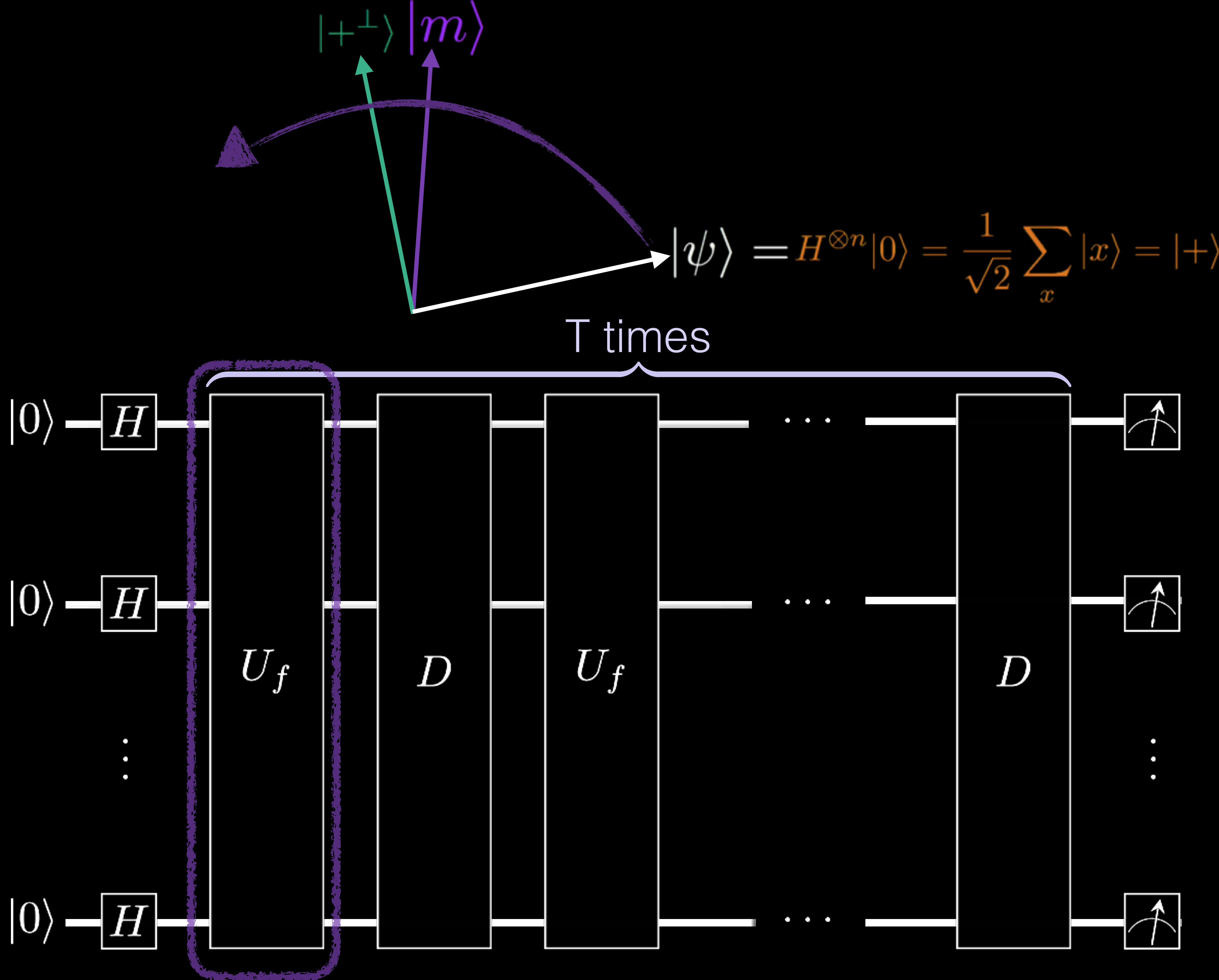
$$H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2}}\sum_x |x\rangle = |+\rangle$$



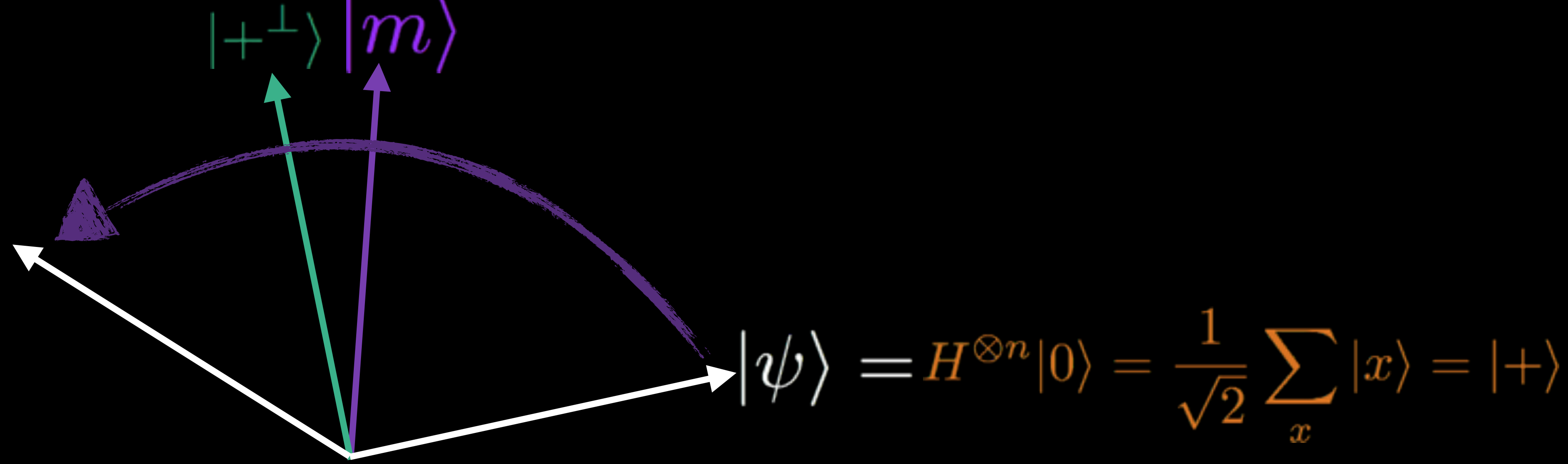
Why does this work?

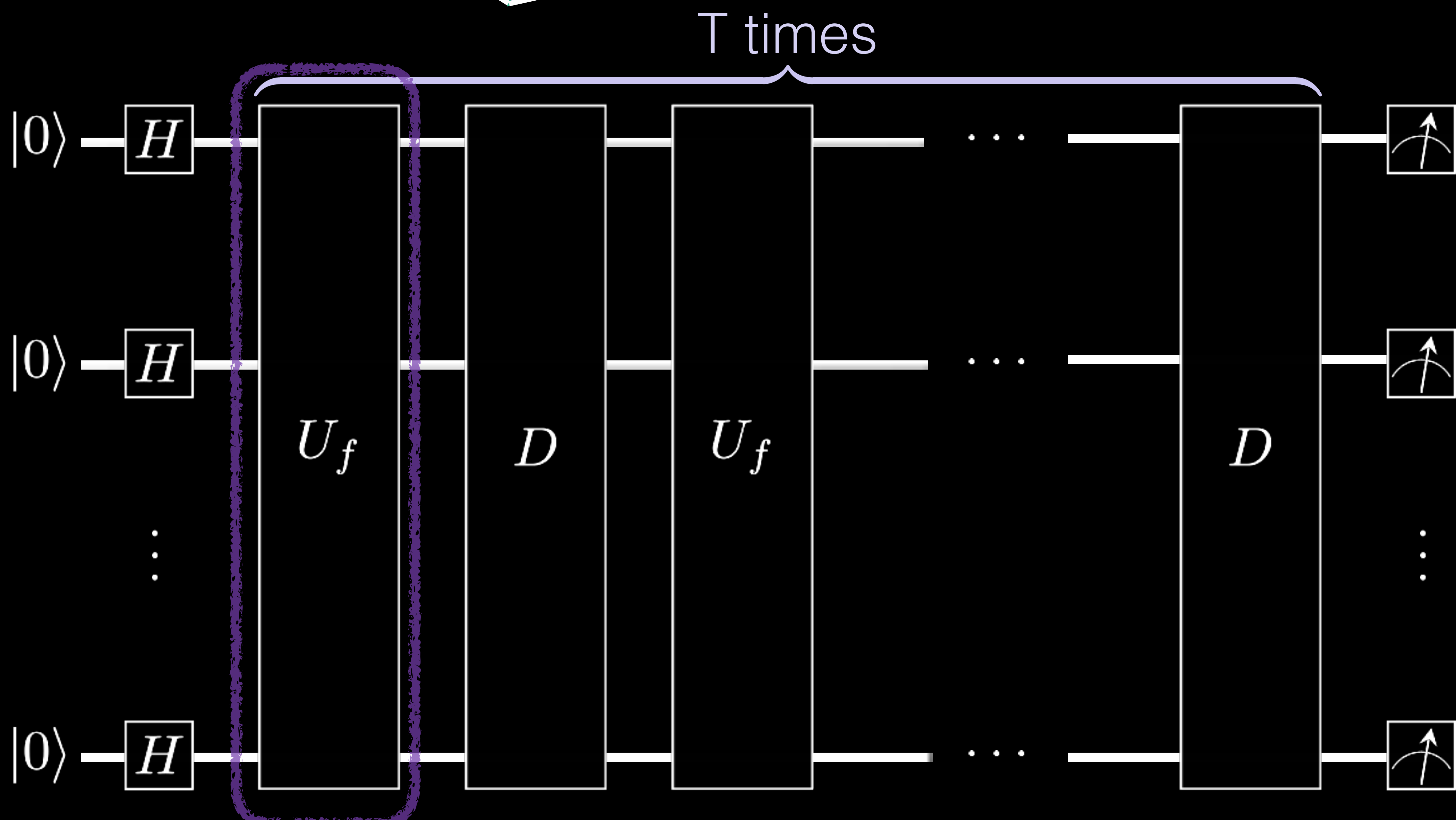


Why does this work?

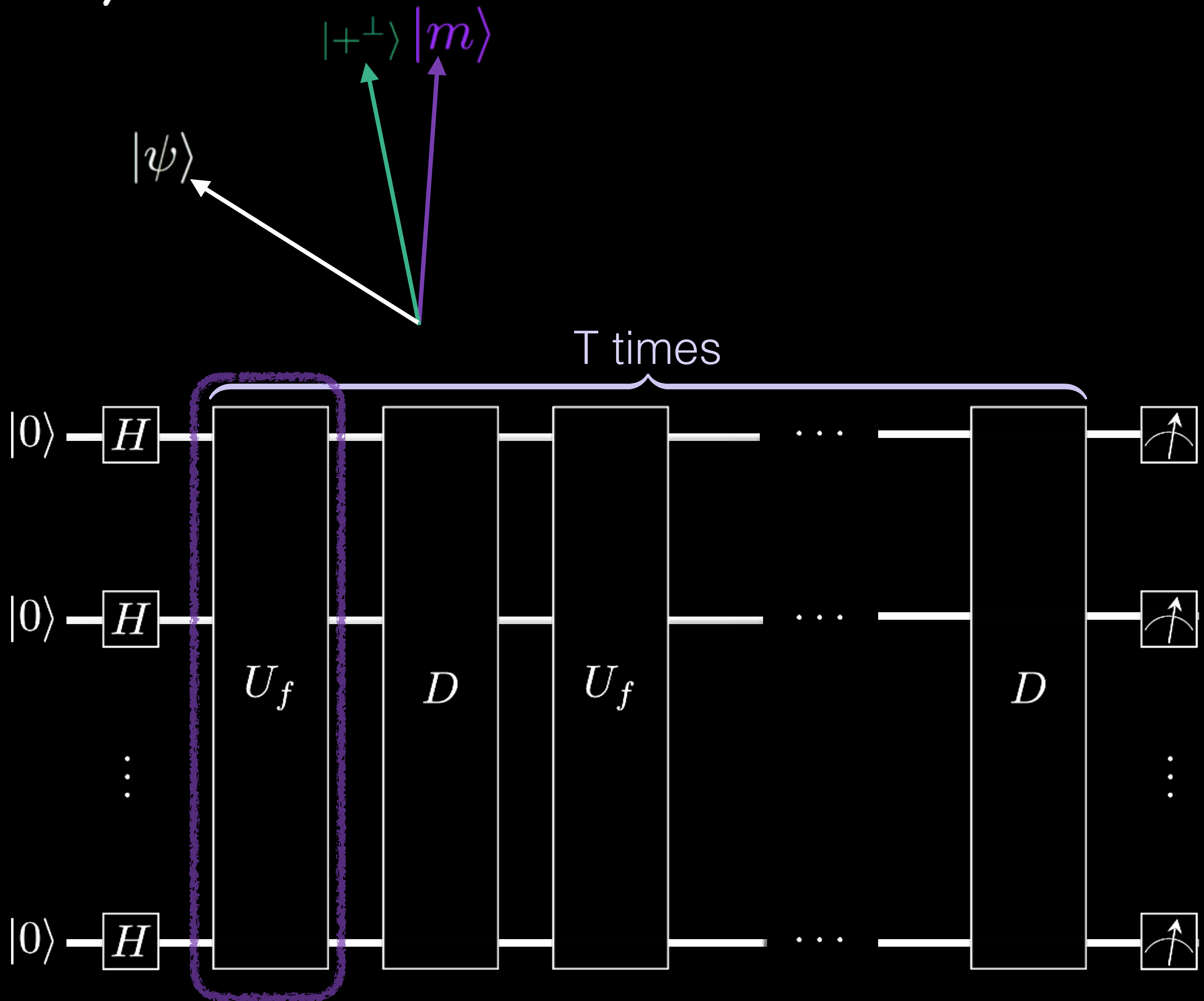


Why does this work?

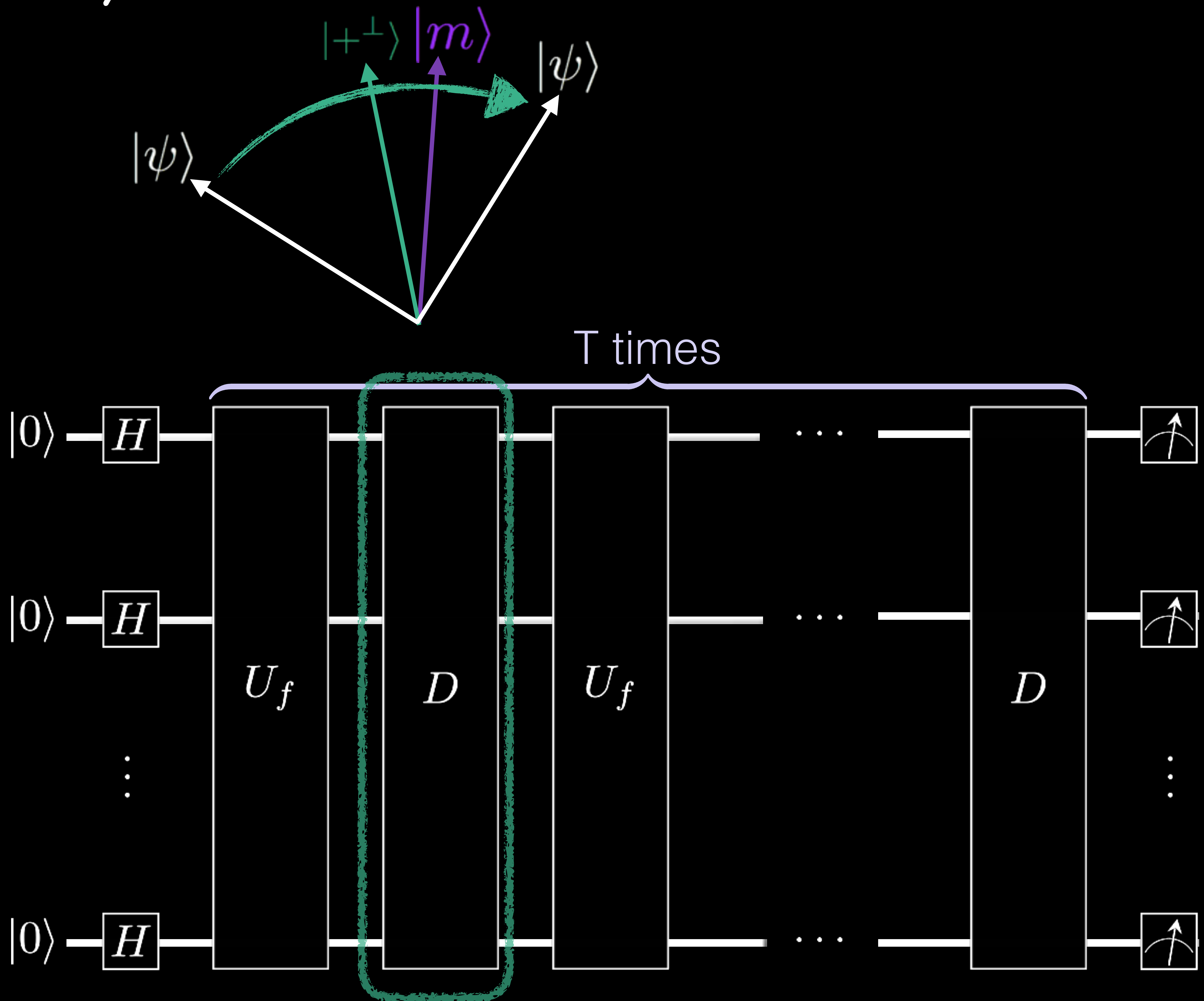




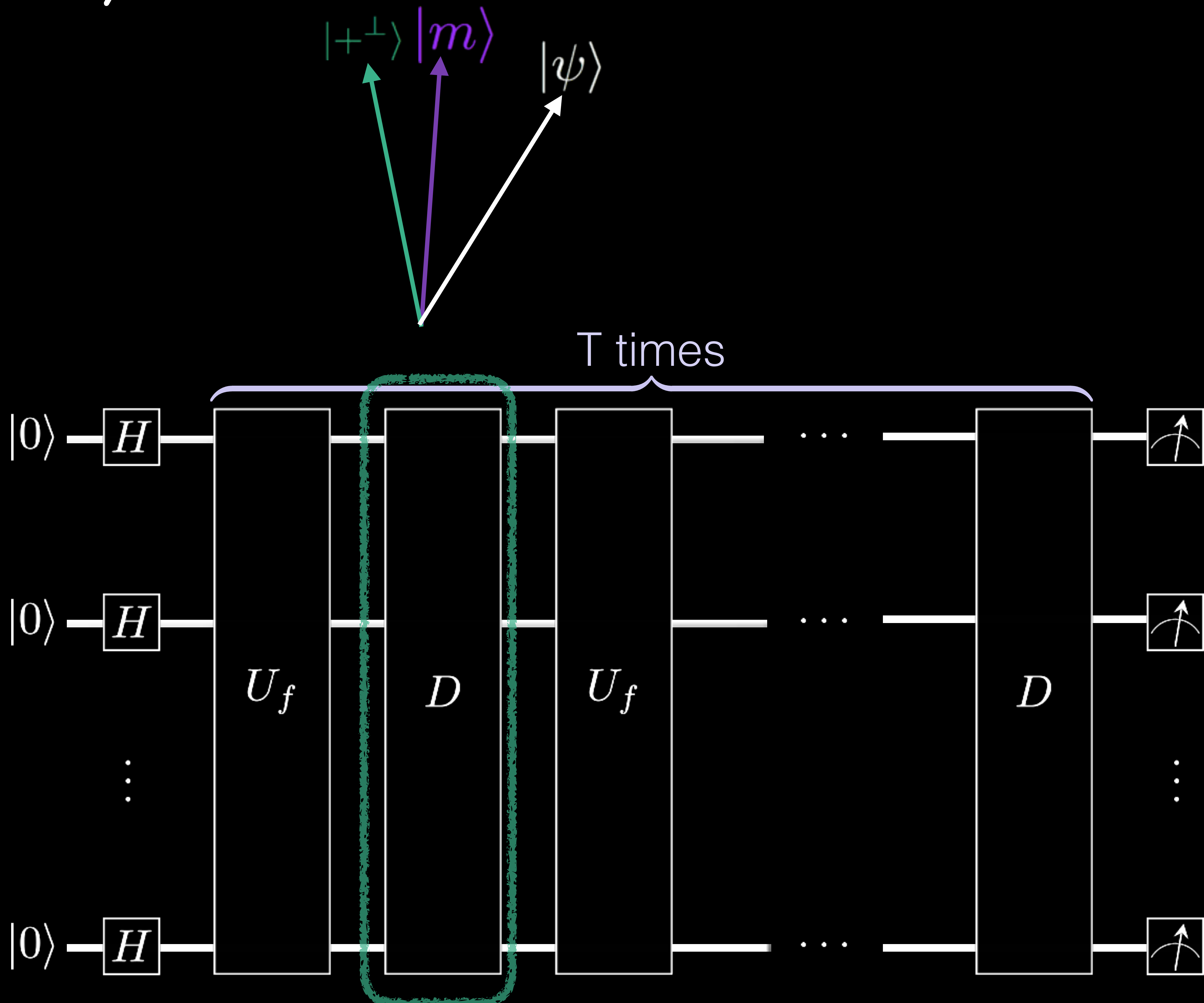
Why does this work?



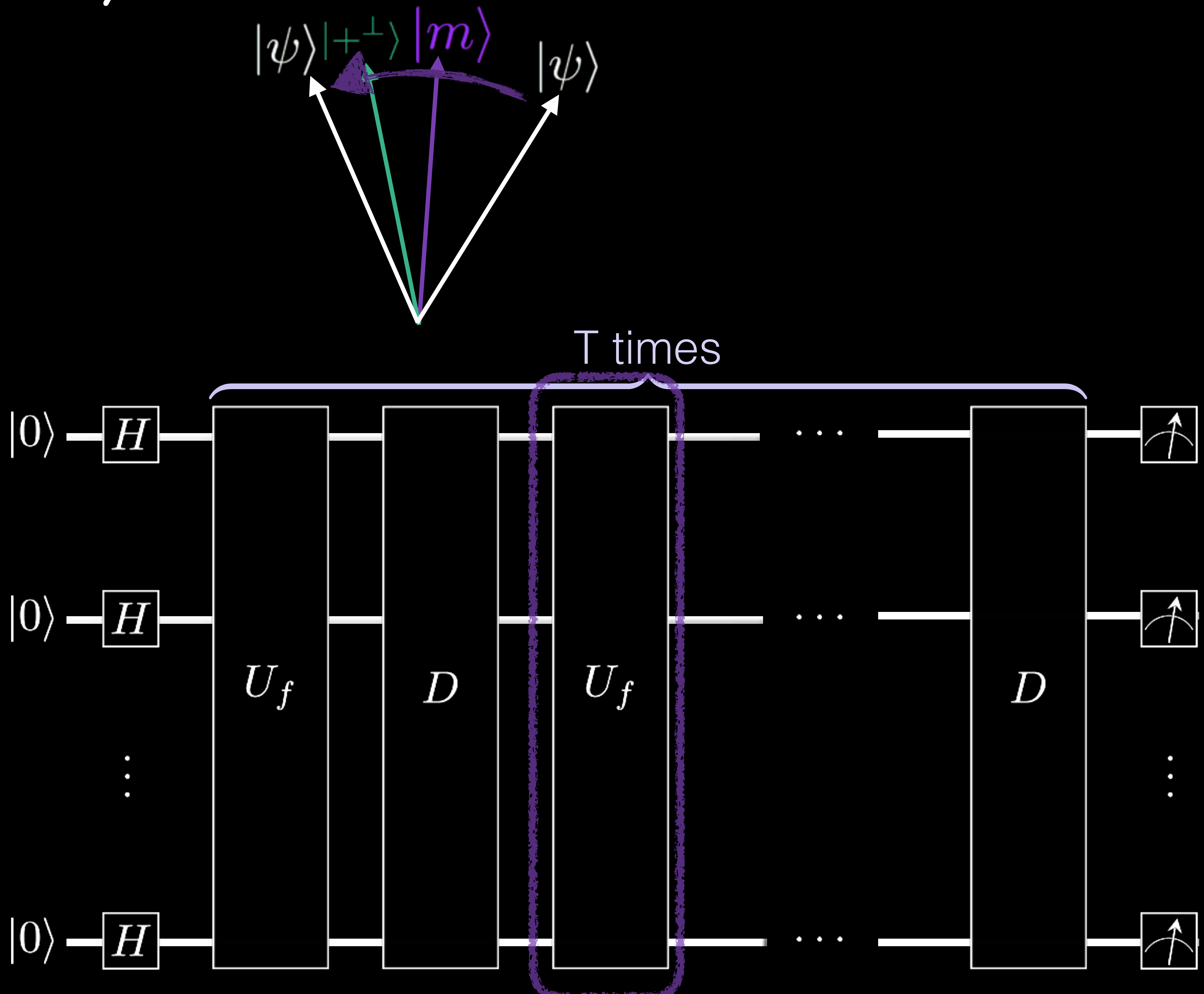
Why does this work?



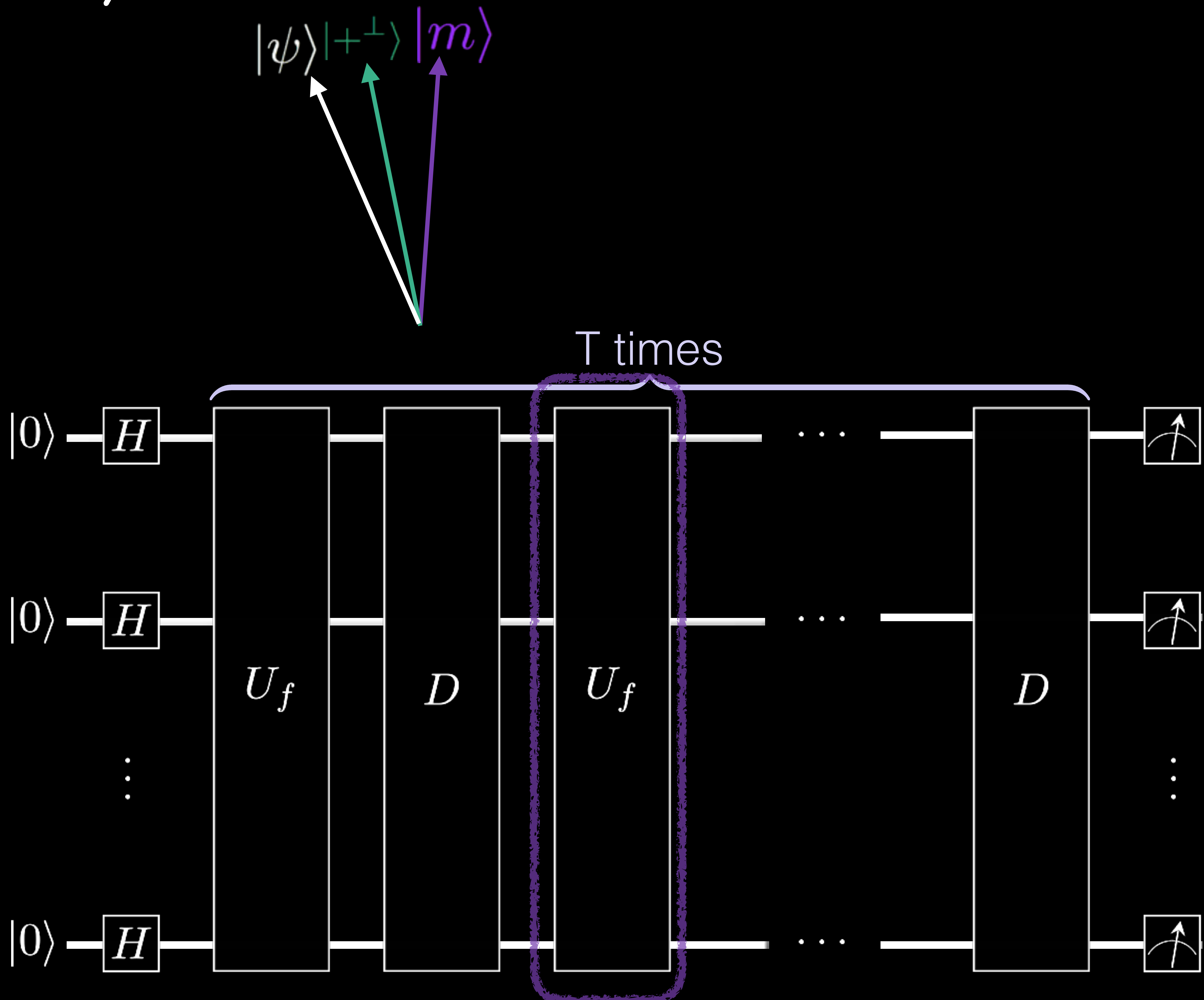
Why does this work?



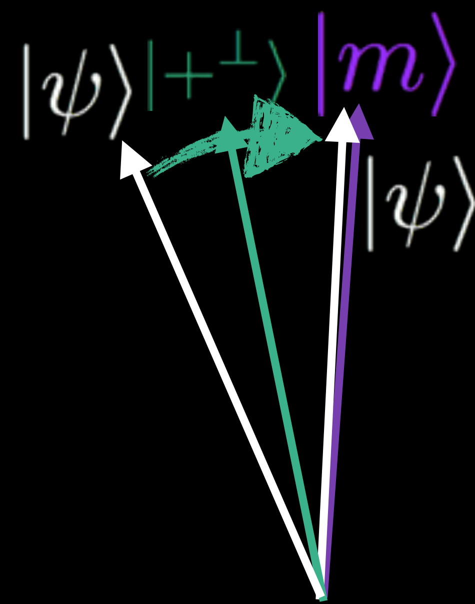
Why does this work?

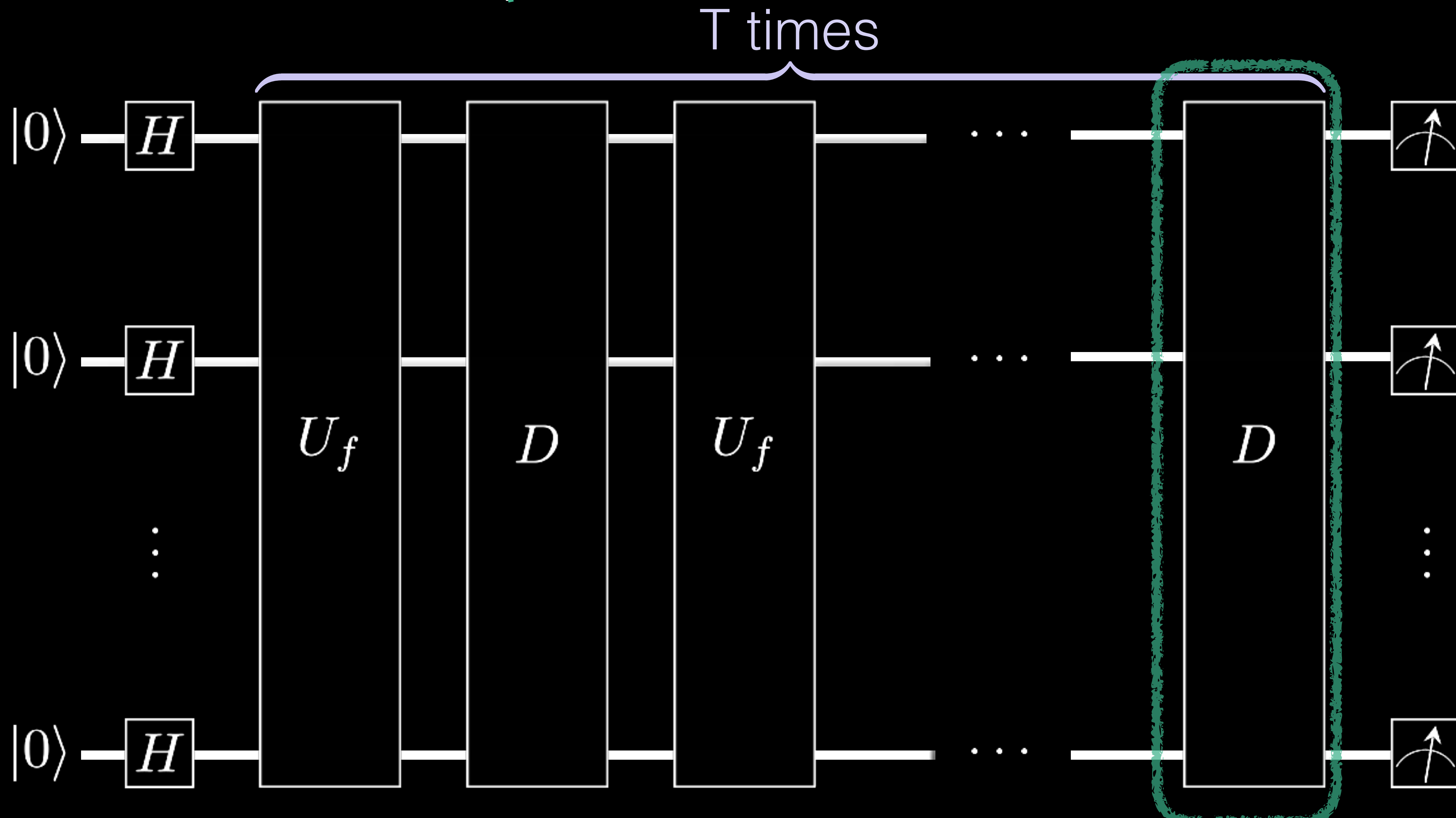


Why does this work?

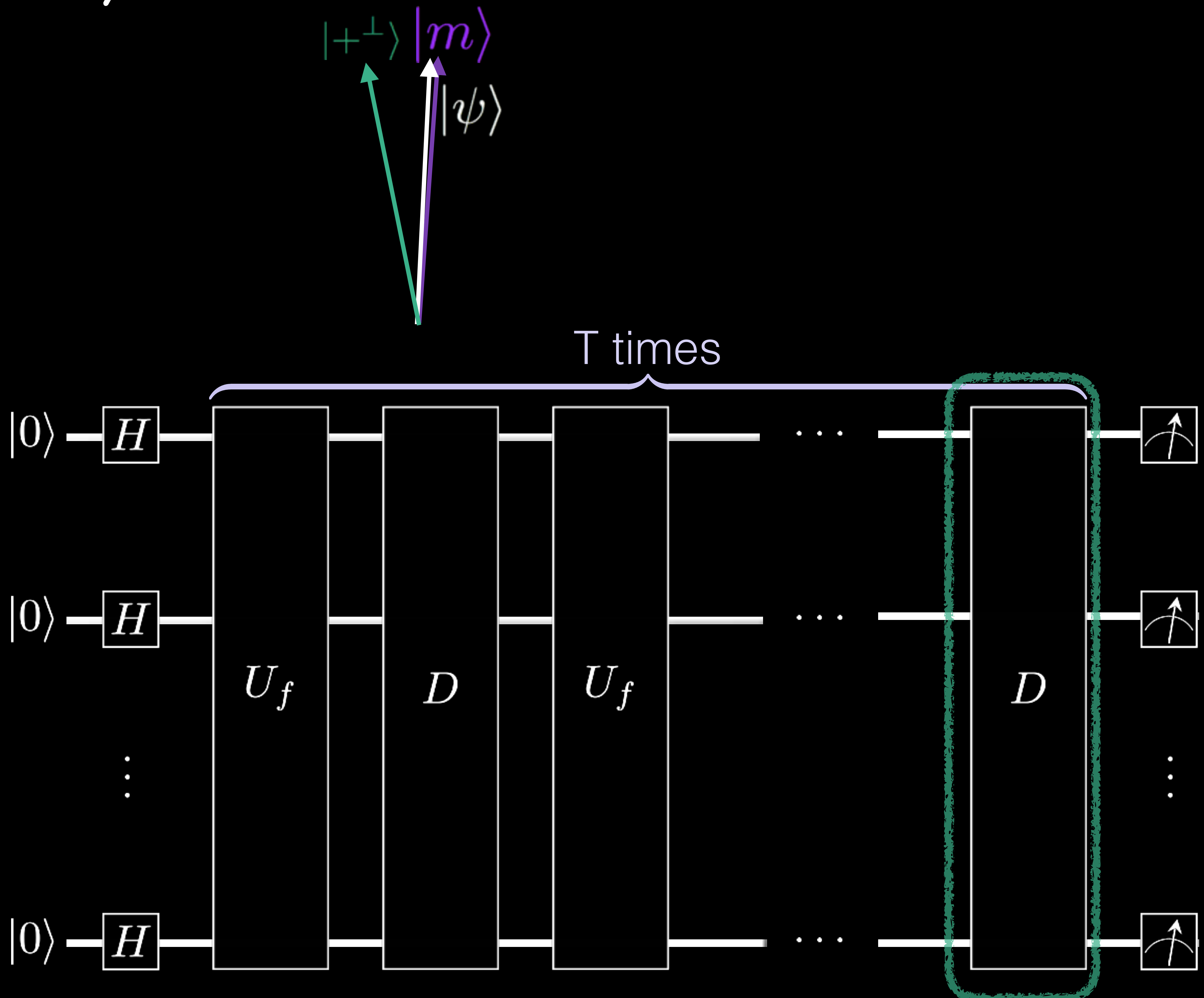


Why does this work?

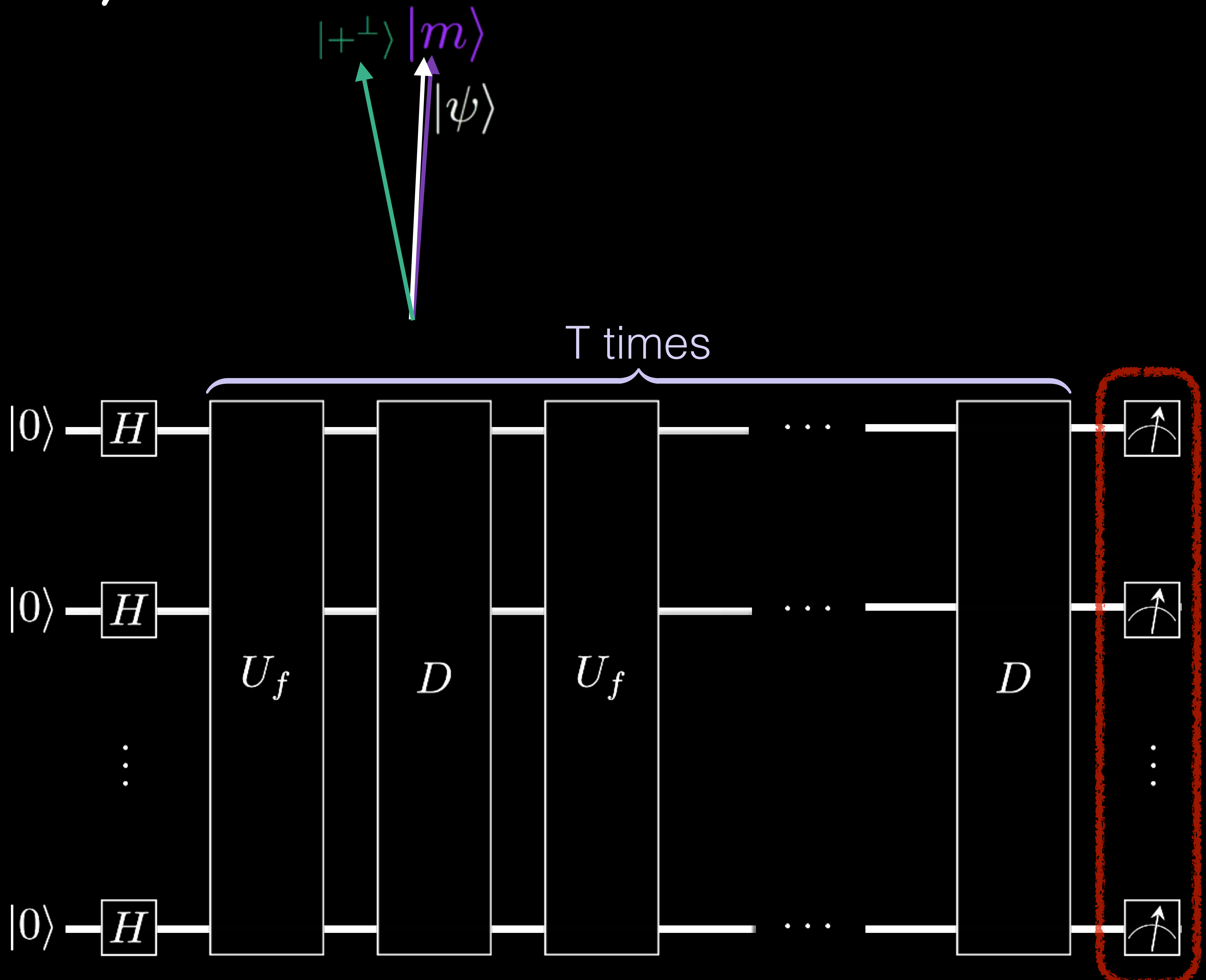




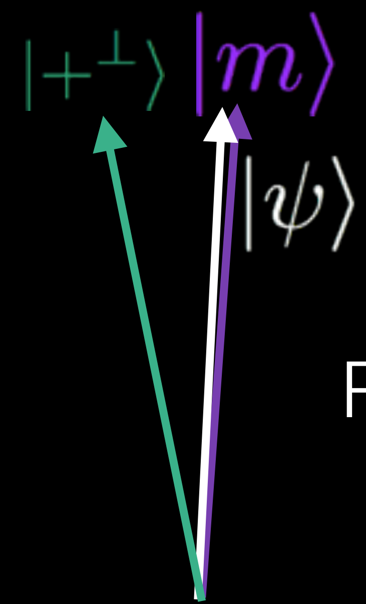
Why does this work?



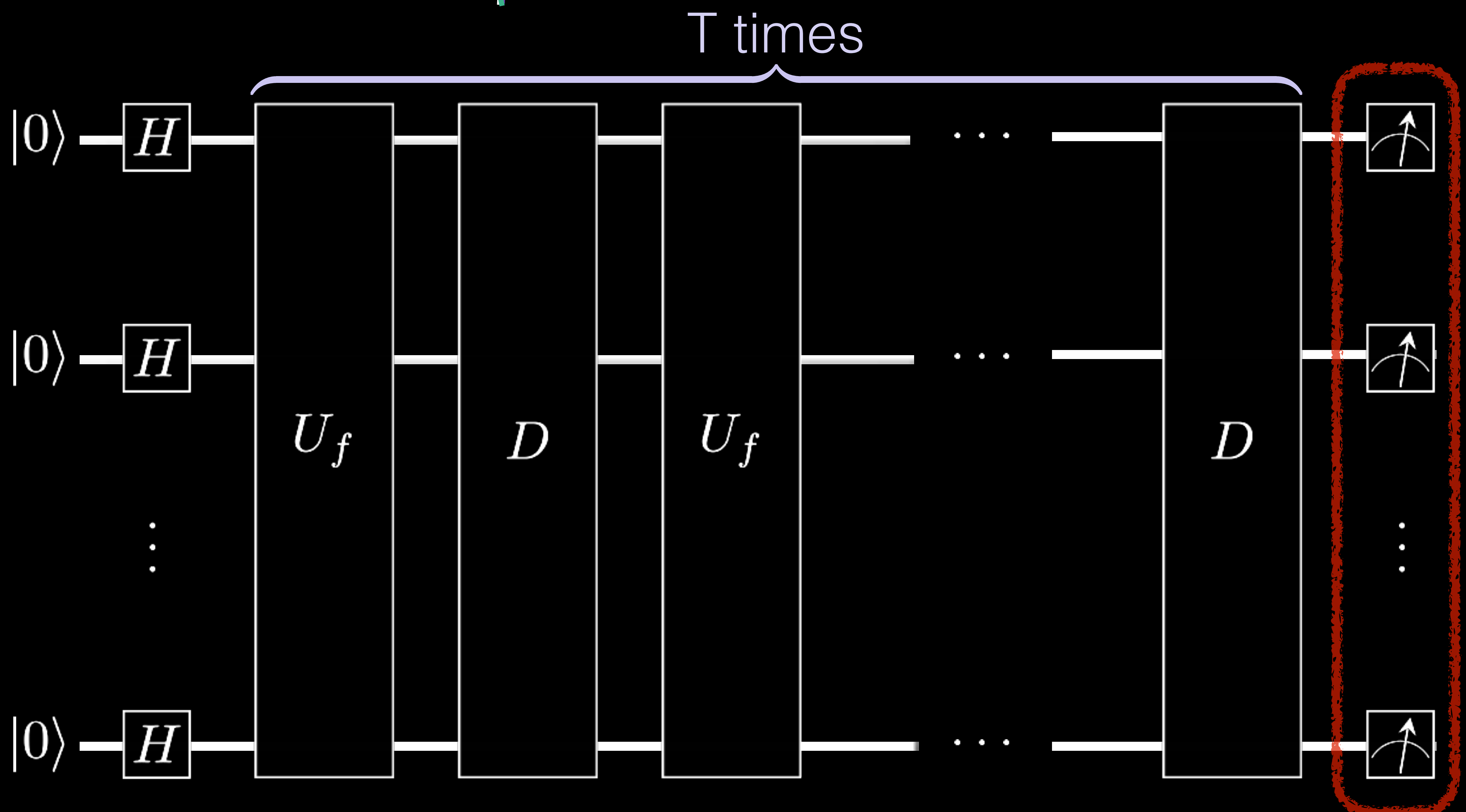
Why does this work?



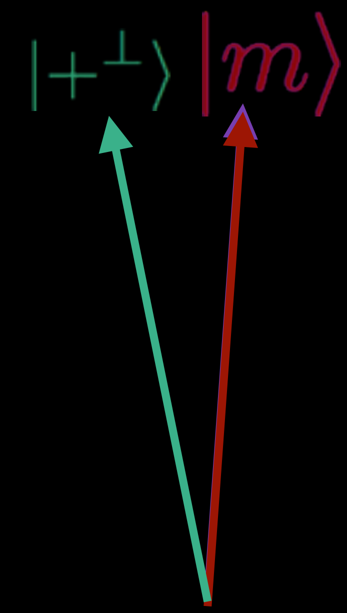
Why does this work?



Probability of measuring $|m\rangle = |\langle m|\psi\rangle|^2$

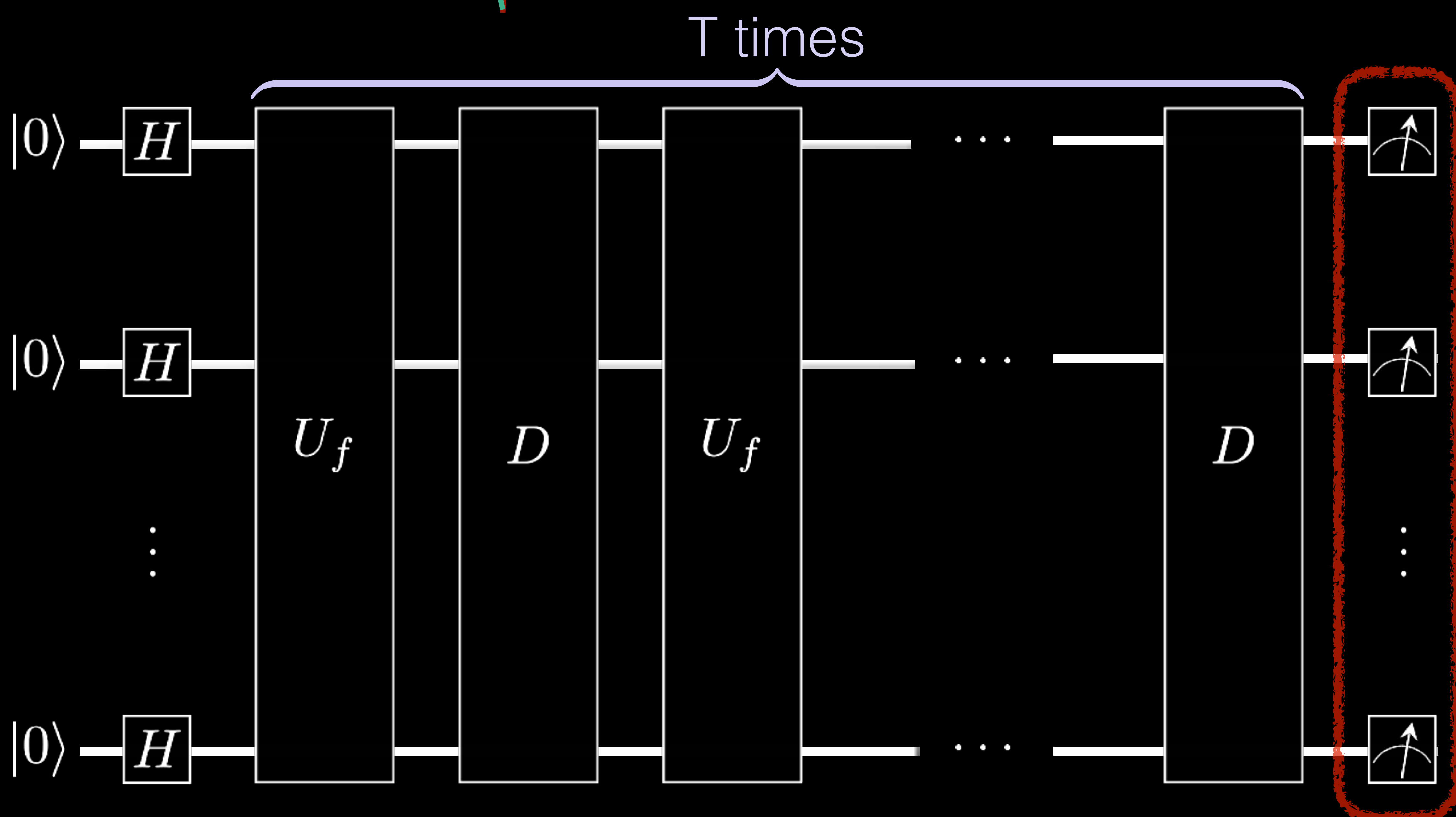


Why does this work?



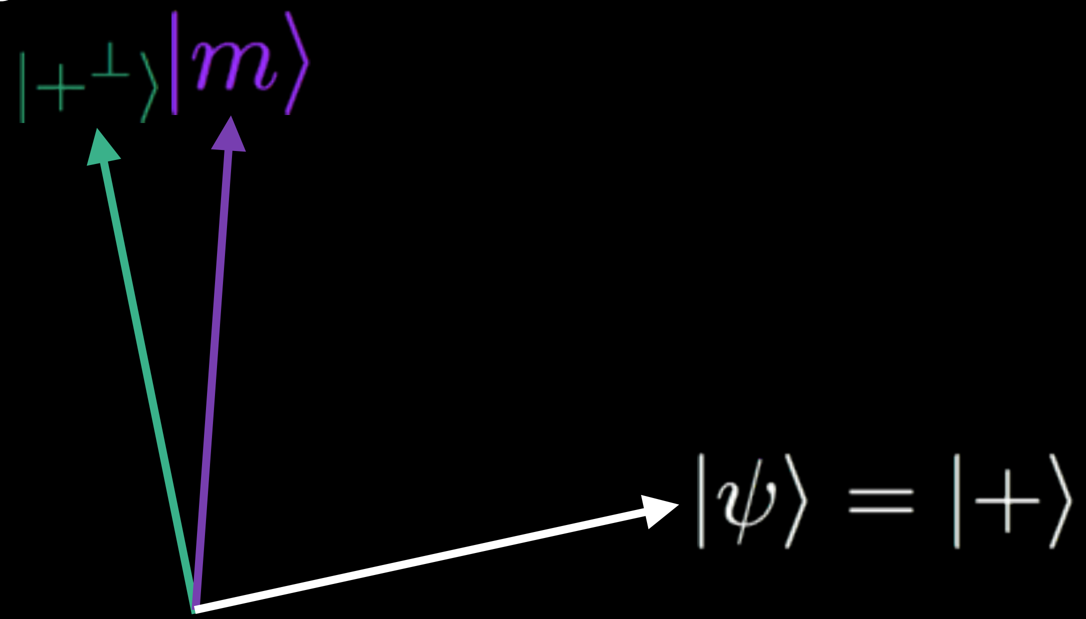
With high probability, the algorithm outputs **m**

Probability of measuring $|m\rangle = |\langle m|\psi\rangle|^2$

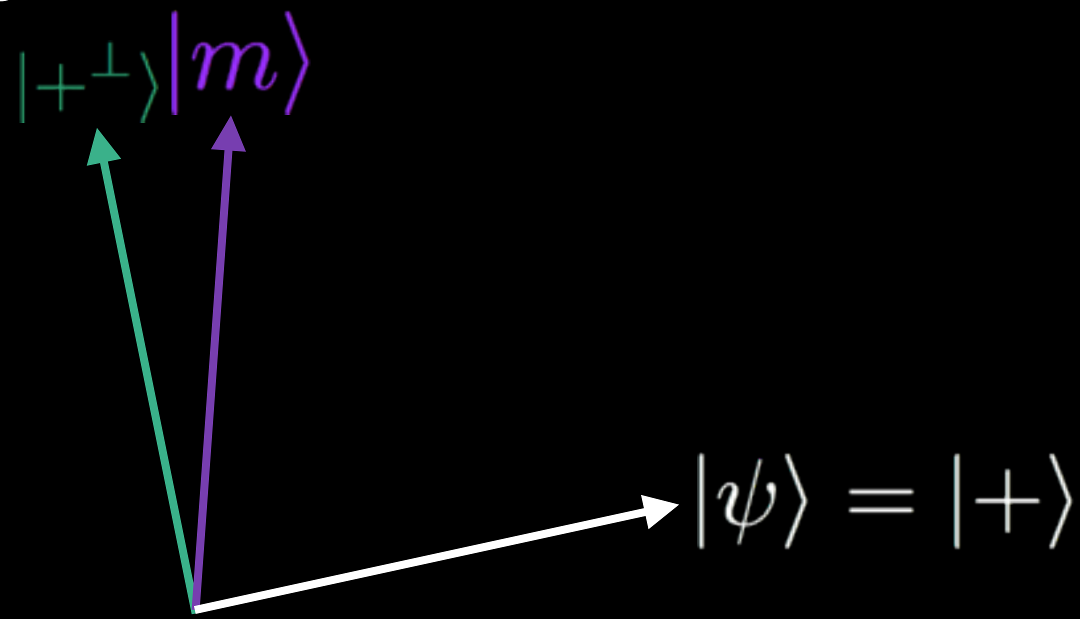


How long does it take?

How long does it take?

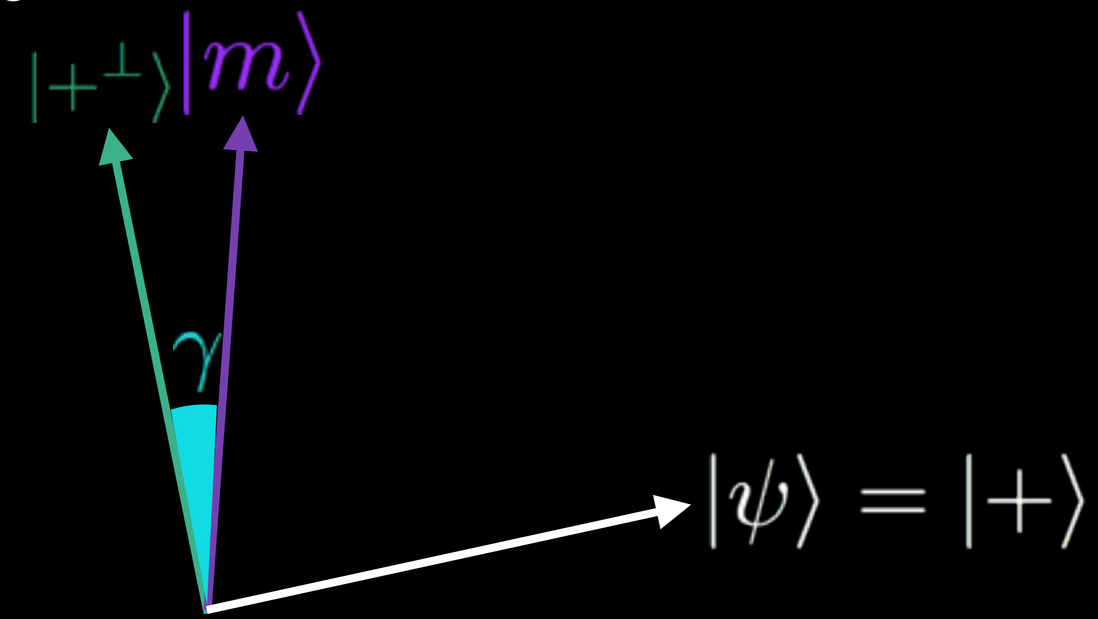


How long does it take?



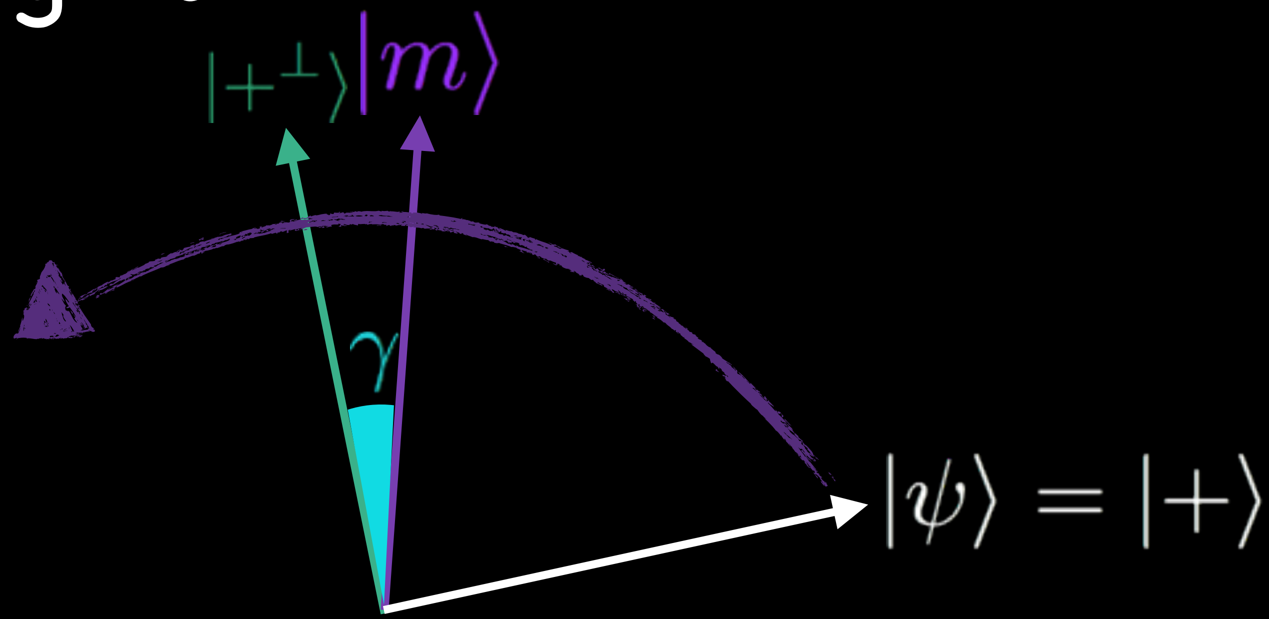
How much closer do we get after each iteration?

How long does it take?



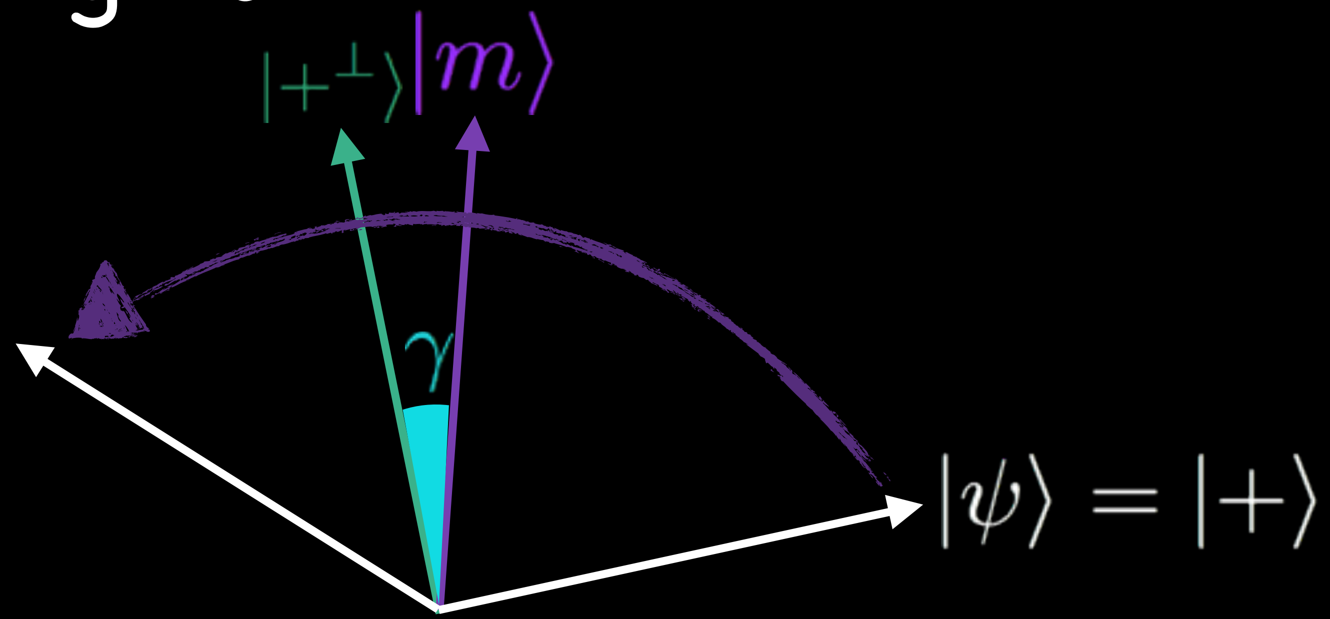
How much closer do we get after each iteration?

How long does it take?



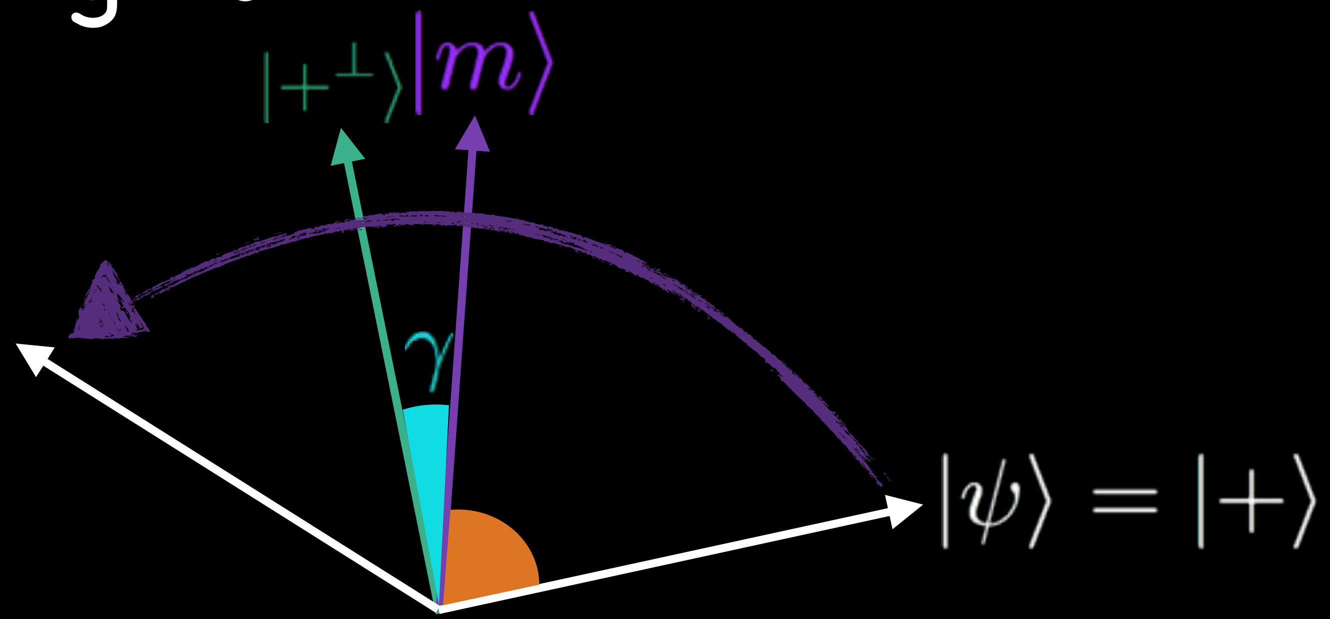
How much closer do we get after each iteration?

How long does it take?



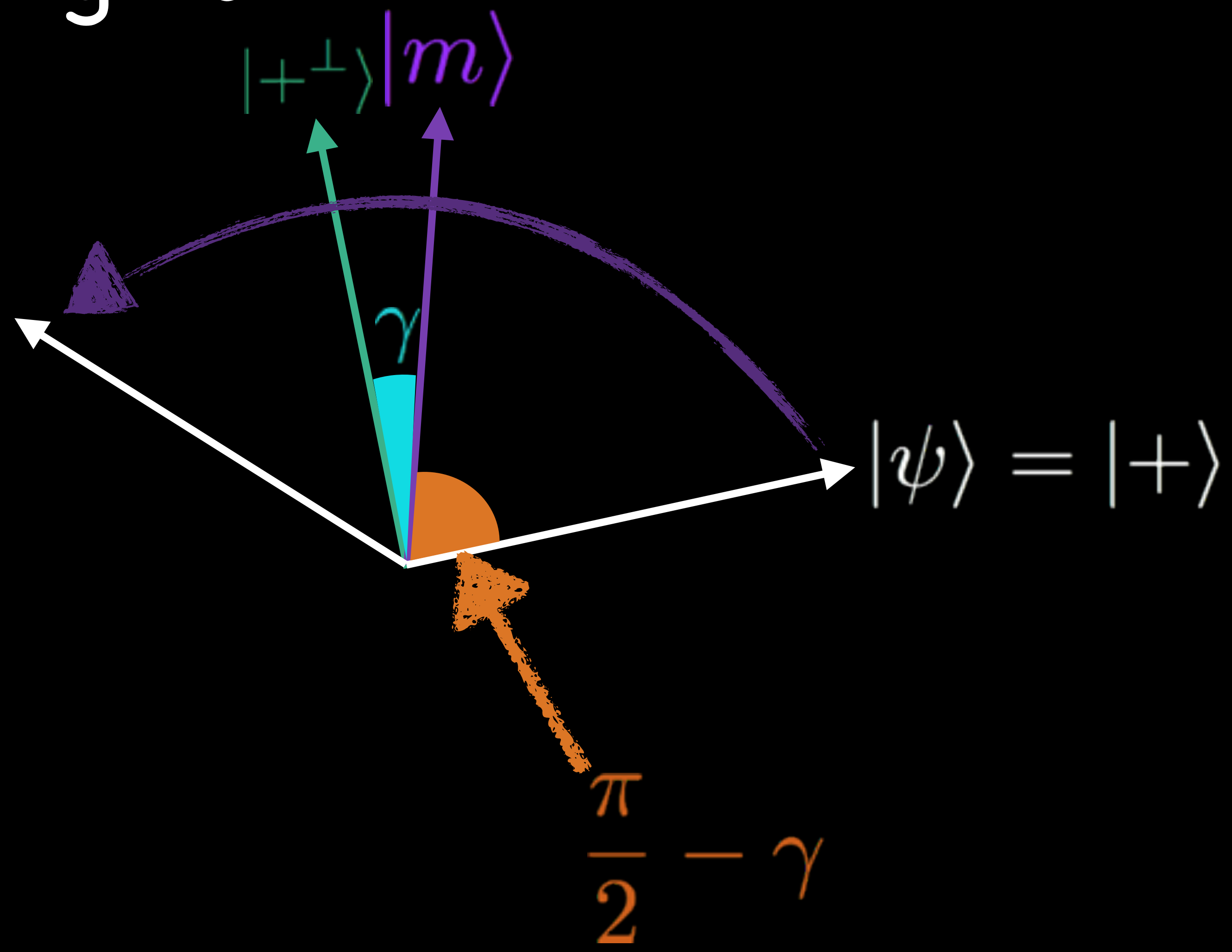
How much closer do we get after each iteration?

How long does it take?



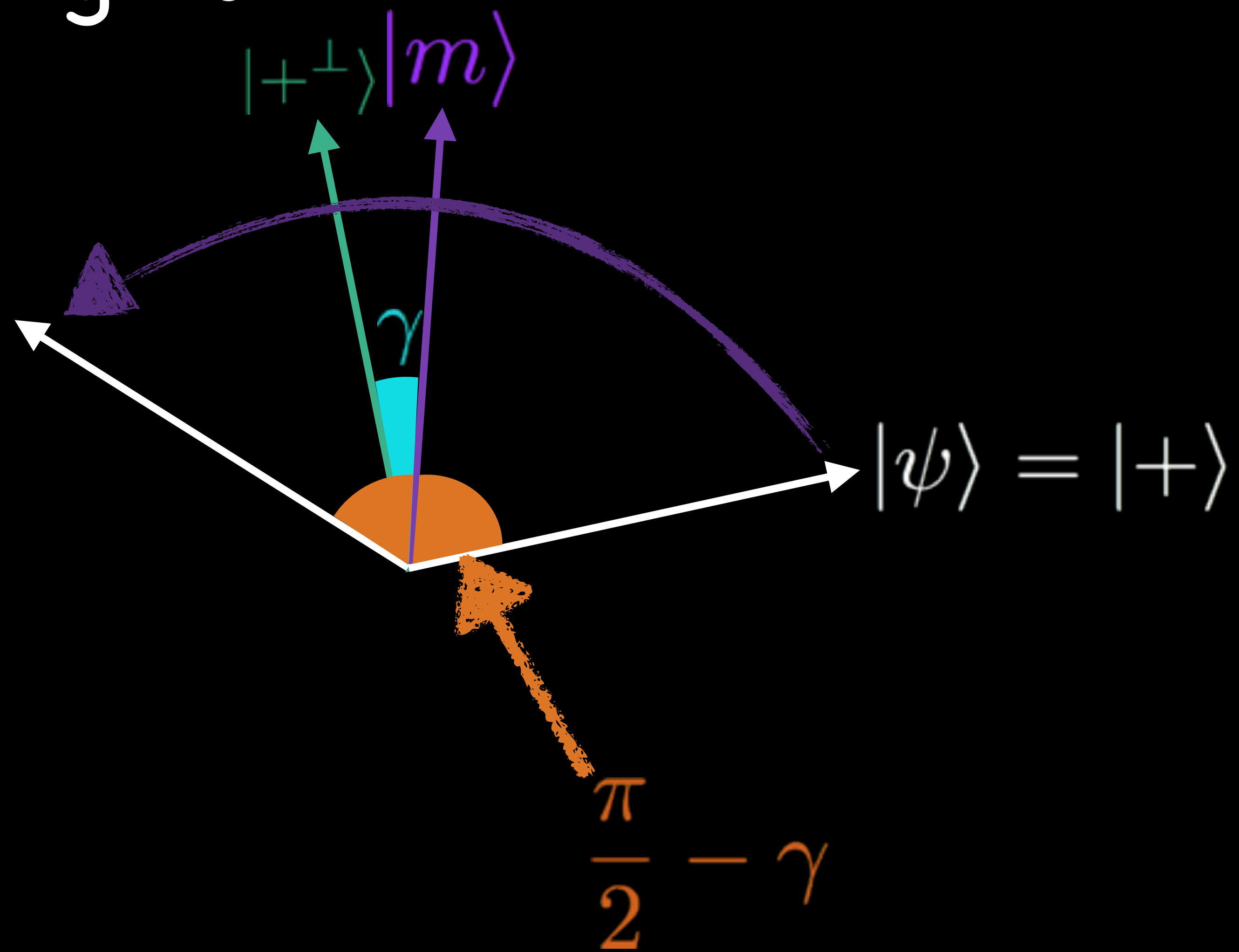
How much closer do we get after each iteration?

How long does it take?



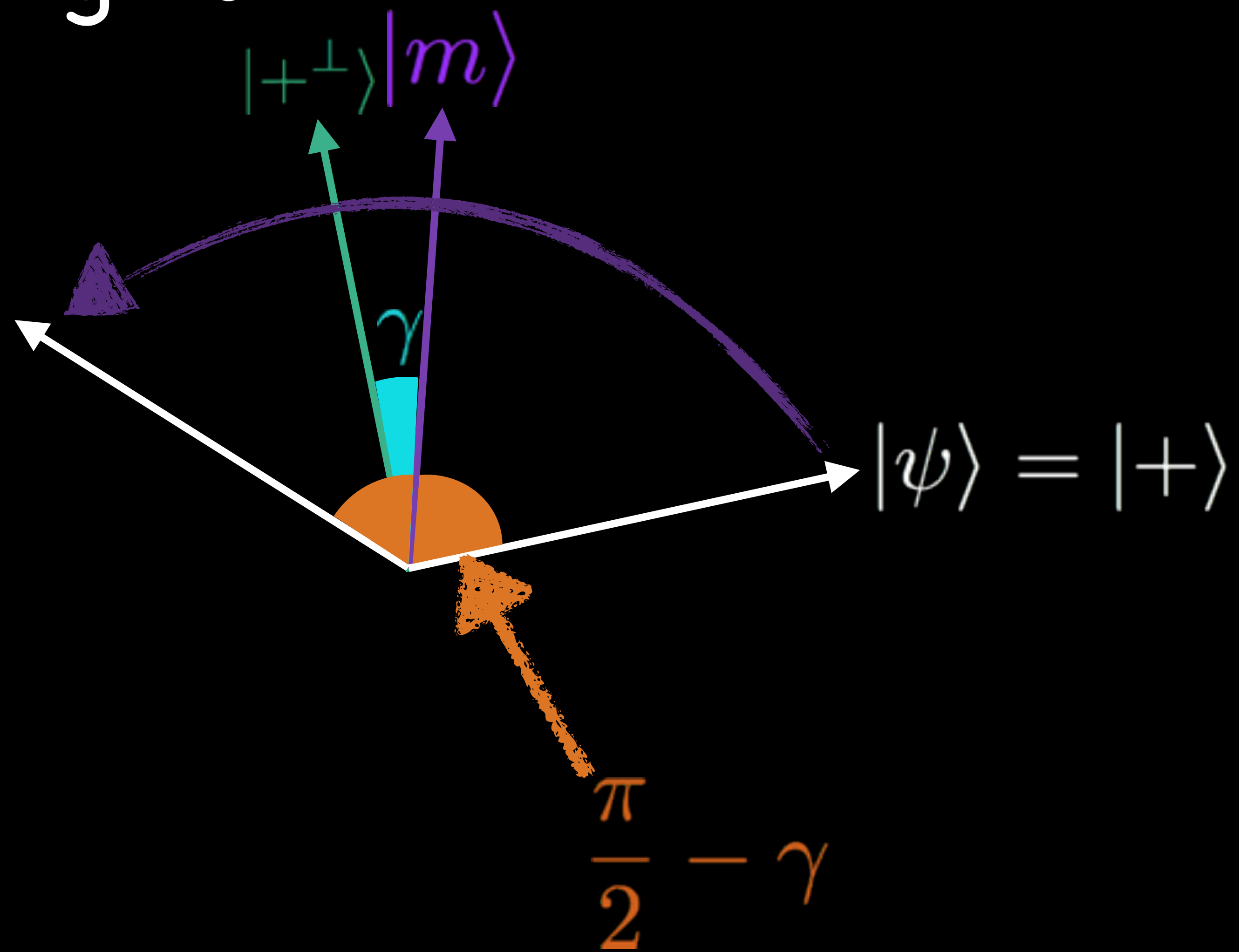
How much closer do we get after each iteration?

How long does it take?



How much closer do we get after each iteration?

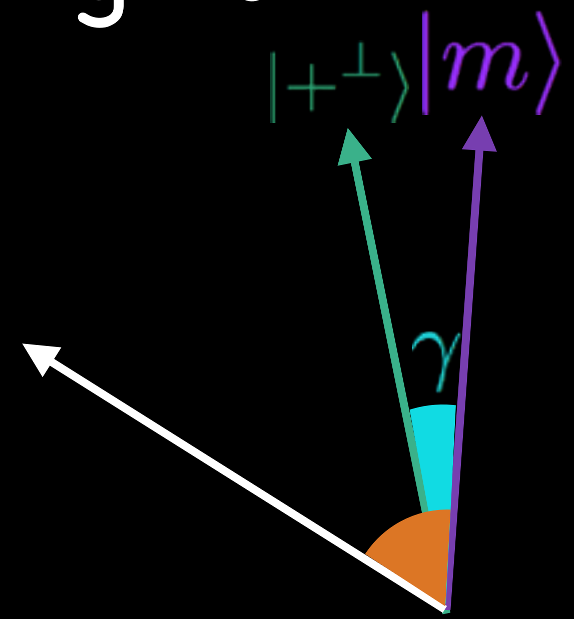
How long does it take?



How much closer do we get after each iteration?

$$\left(\frac{\pi}{2} - \gamma\right) + \left(\frac{\pi}{2} - \gamma\right)$$

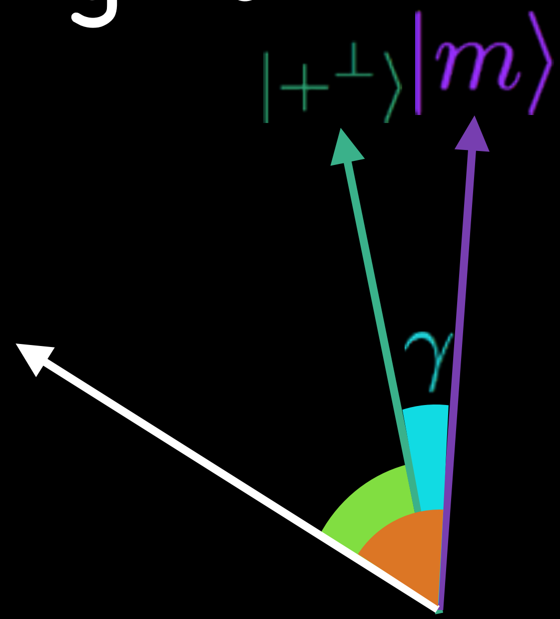
How long does it take?



How much closer do we get after each iteration?

$$\left(\frac{\pi}{2} - \gamma\right) + \left(\frac{\pi}{2} - \gamma\right)$$

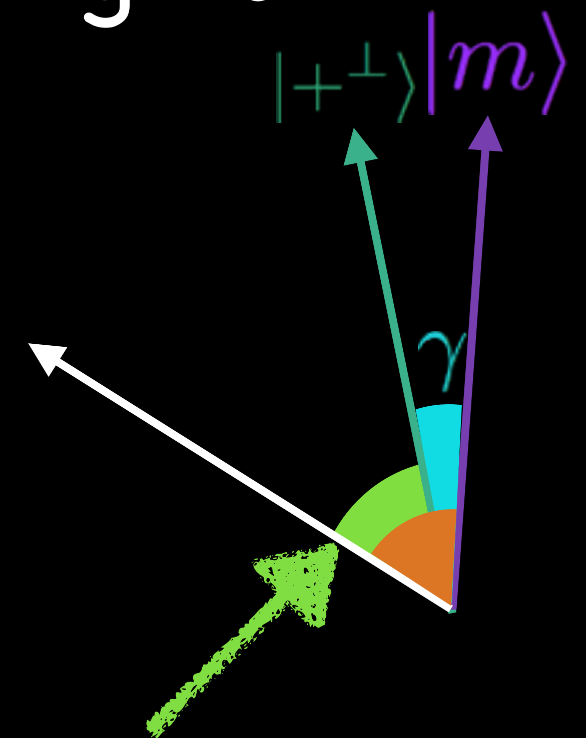
How long does it take?



How much closer do we get after each iteration?

$$\left(\frac{\pi}{2} - \gamma\right) + \left(\frac{\pi}{2} - \gamma\right)$$

How long does it take?

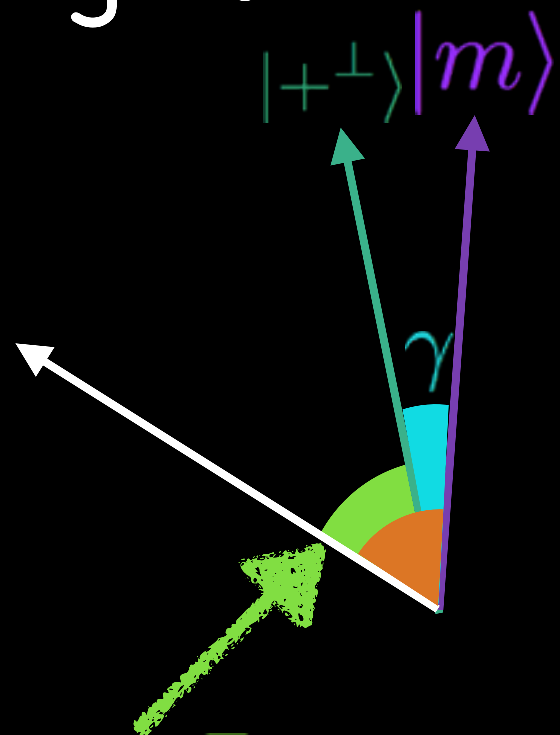


$$\left(\frac{\pi}{2} - \gamma\right) - \gamma$$

How much closer do we get after each iteration?

$$\left(\frac{\pi}{2} - \gamma\right) + \left(\frac{\pi}{2} - \gamma\right)$$

How long does it take?

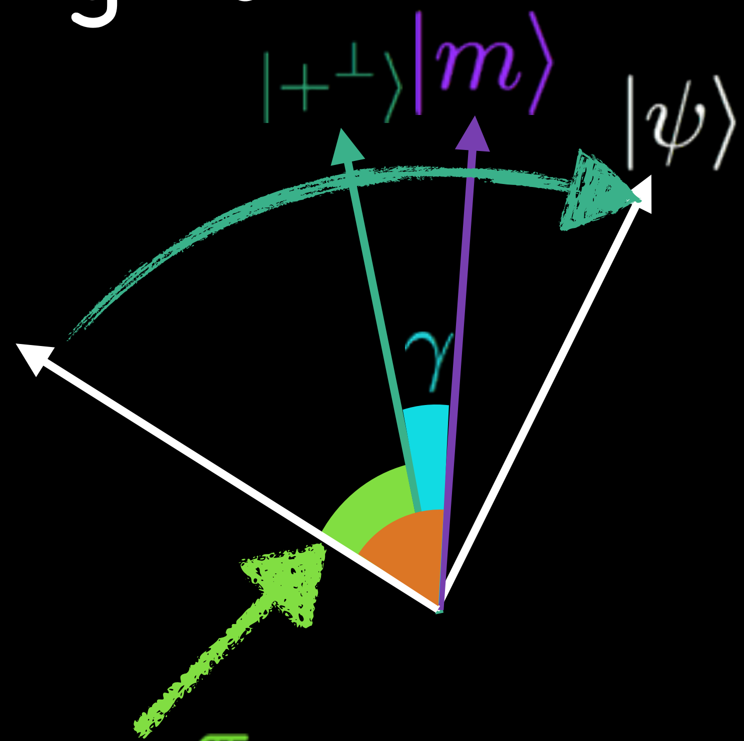


$$\left(\frac{\pi}{2} - \gamma\right) - \gamma = \frac{\pi}{2} - 2\gamma$$

How much closer do we get after each iteration?

$$\left(\frac{\pi}{2} - \gamma\right) + \left(\frac{\pi}{2} - \gamma\right)$$

How long does it take?

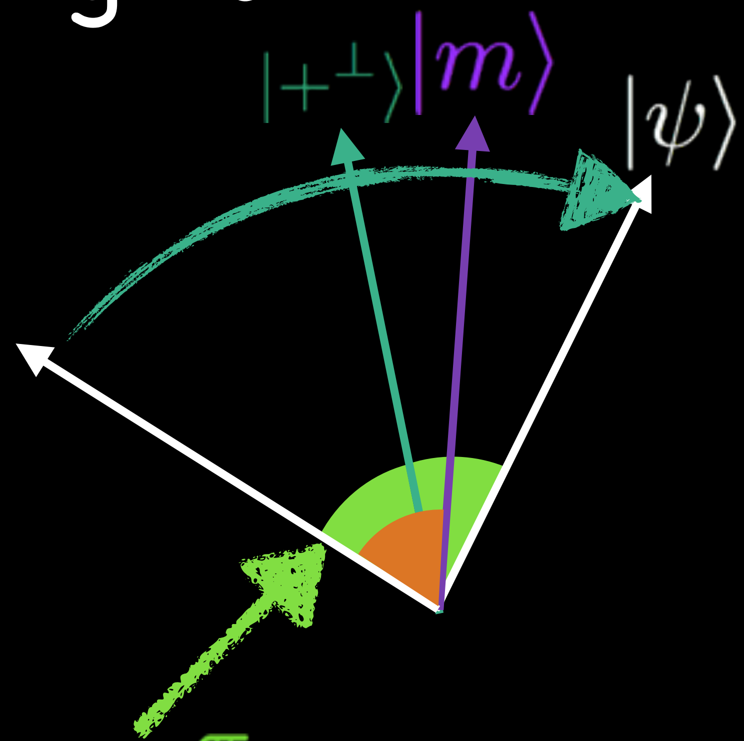


$$\left(\frac{\pi}{2}-\gamma\right)-\gamma=\frac{\pi}{2}-2\gamma$$

How much closer do we get after each iteration?

$$\left(\frac{\pi}{2}-\gamma\right)+\left(\frac{\pi}{2}-\gamma\right)$$

How long does it take?

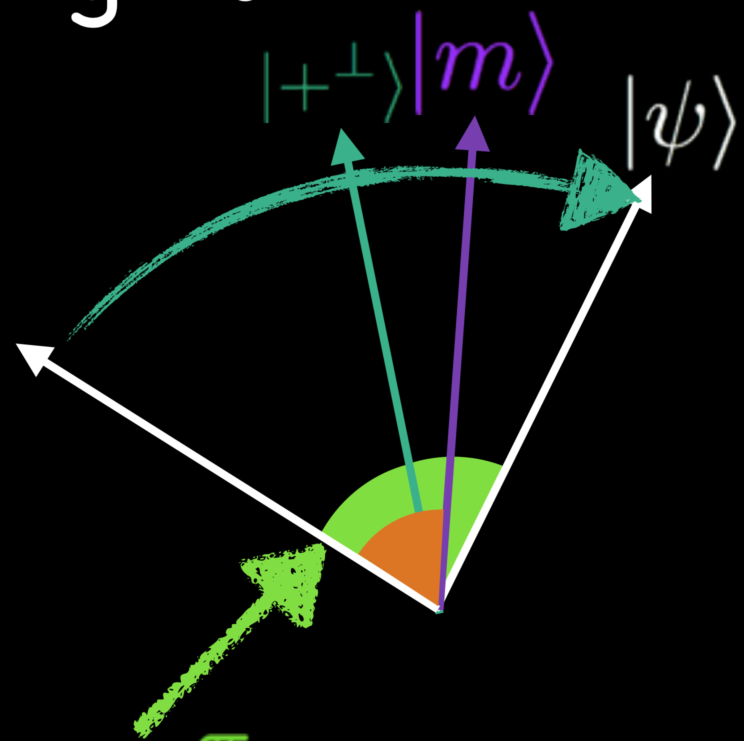


$$\left(\frac{\pi}{2} - \gamma\right) - \gamma = \frac{\pi}{2} - 2\gamma$$

How much closer do we get after each iteration?

$$\left(\frac{\pi}{2} - \gamma\right) + \left(\frac{\pi}{2} - \gamma\right)$$

How long does it take?

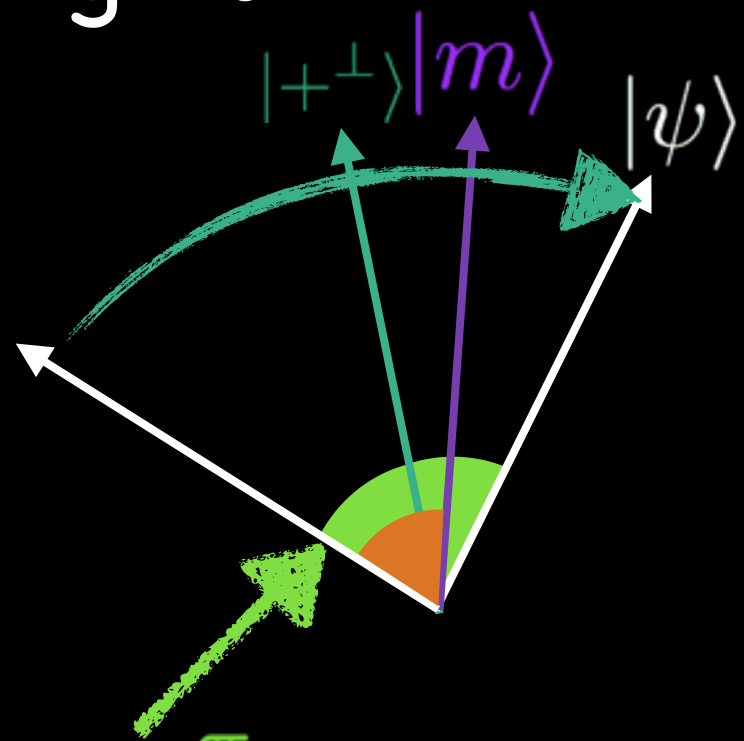


$$\left(\frac{\pi}{2} - \gamma\right) - \gamma = \frac{\pi}{2} - 2\gamma$$

How much closer do we get after each iteration?

$$\left(\frac{\pi}{2} - \gamma\right) + \left(\frac{\pi}{2} - \gamma\right) - \left(\frac{\pi}{2} - 2\gamma\right) - \left(\frac{\pi}{2} - 2\gamma\right)$$

How long does it take?

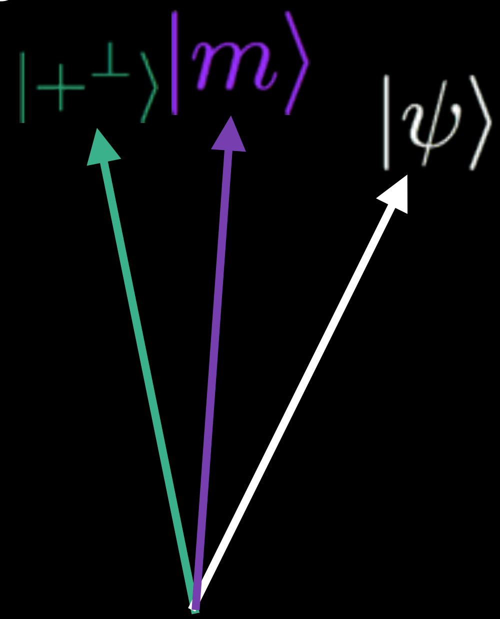


$$\left(\frac{\pi}{2}-\gamma\right)-\gamma=\frac{\pi}{2}-2\gamma$$

How much closer do we get after each iteration?

$$\left(\frac{\pi}{2}-\gamma\right)+\left(\frac{\pi}{2}-\gamma\right)-\left(\frac{\pi}{2}-2\gamma\right)-\left(\frac{\pi}{2}-2\gamma\right)$$
$$=2\gamma$$

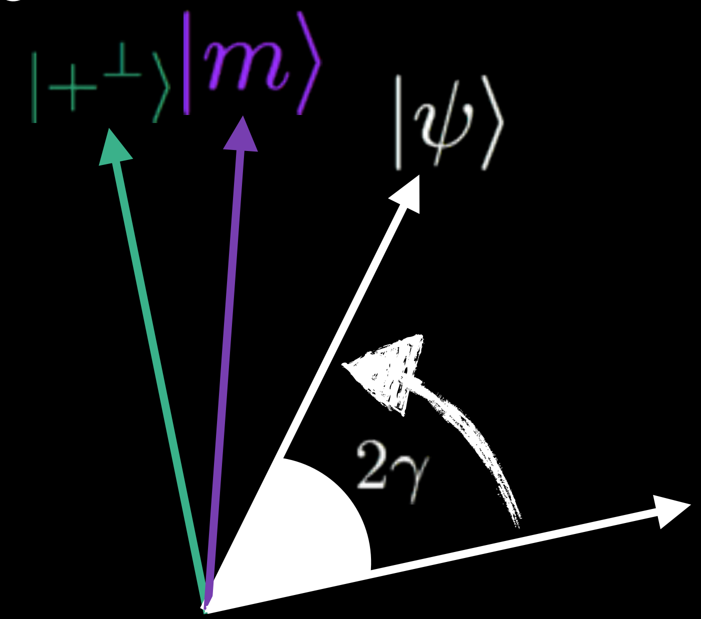
How long does it take?



How much closer do we get after each iteration?

$$\left(\frac{\pi}{2}-\gamma\right)+\left(\frac{\pi}{2}-\gamma\right)-\left(\frac{\pi}{2}-2\gamma\right)-\left(\frac{\pi}{2}-2\gamma\right)$$
$$=2\gamma$$

How long does it take?

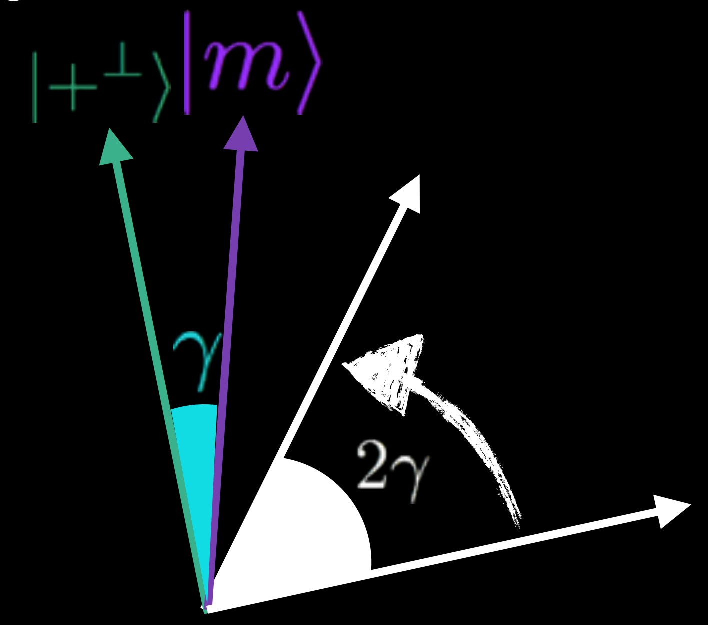


How much closer do we get after each iteration?

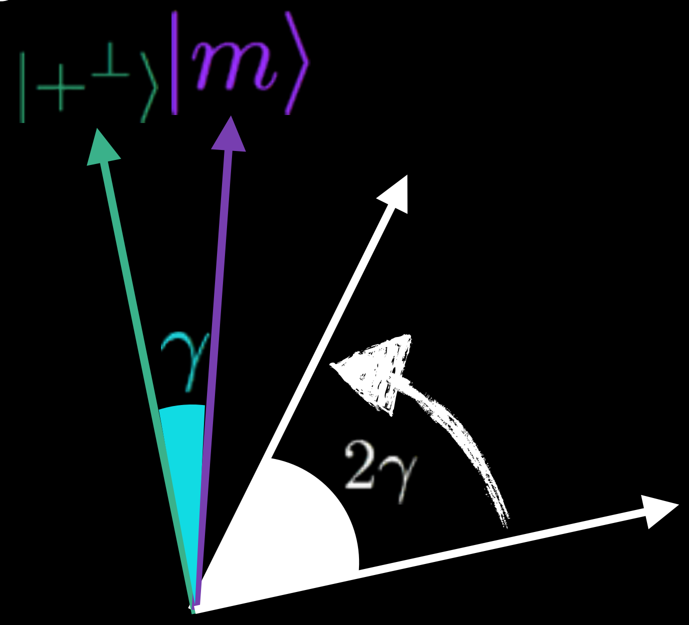
$$\left(\frac{\pi}{2}-\gamma\right)+\left(\frac{\pi}{2}-\gamma\right)-\left(\frac{\pi}{2}-2\gamma\right)-\left(\frac{\pi}{2}-2\gamma\right)$$
$$=2\gamma$$

How long does it take?

How long does it take?

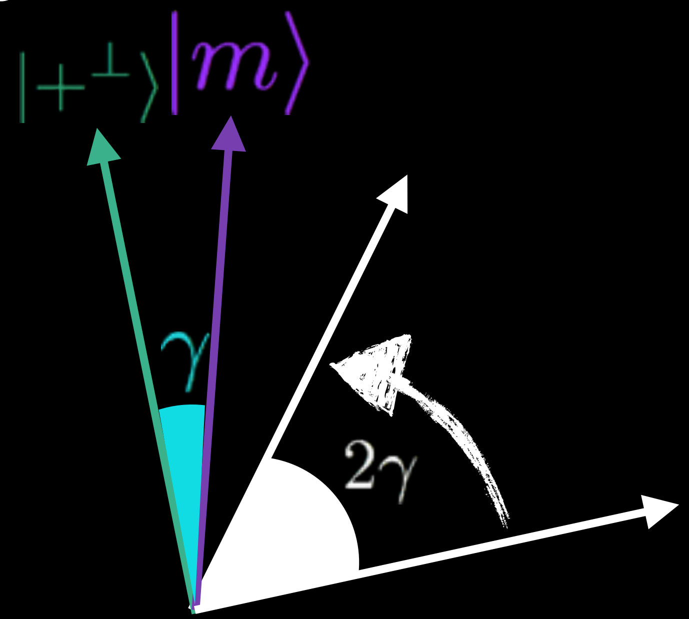


How long does it take?



$$\cos\gamma = \langle +^{\perp}|m\rangle$$

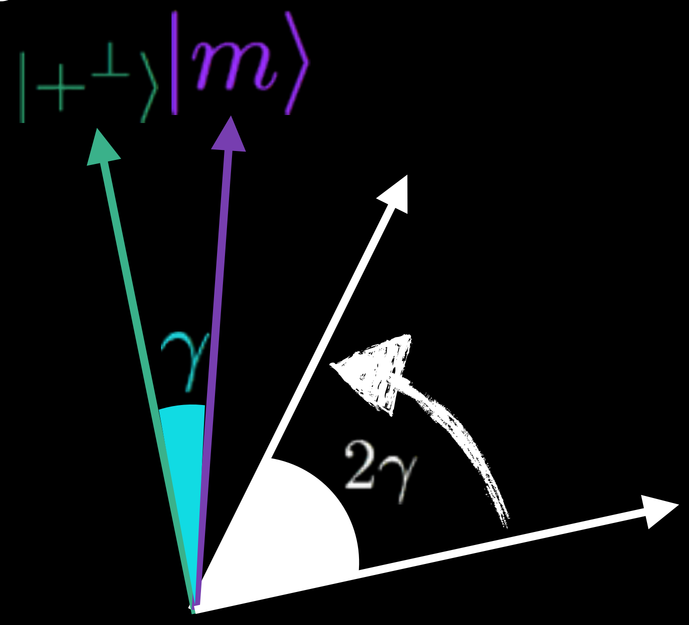
How long does it take?



$$\cos\gamma = \langle +^\perp | m\rangle$$

$$\sin\gamma = \langle + | m\rangle$$

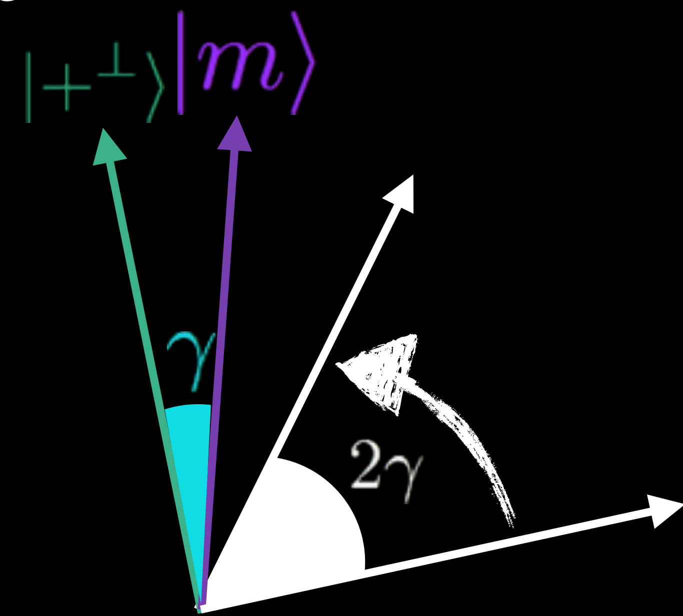
How long does it take?



$$\cos\gamma = \langle +^{\perp}|m\rangle$$

$$\sin\gamma = \langle +|m\rangle = \frac{1}{\sqrt{N}}$$

How long does it take?

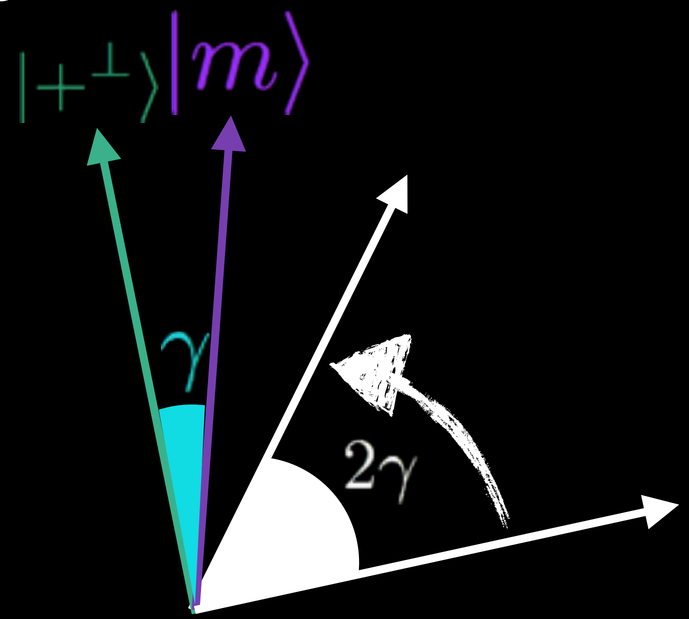


$$\cos\gamma = \langle +^{\perp} | m \rangle$$

$$\sin\gamma = \langle + | m \rangle = \frac{1}{\sqrt{N}}$$

$$\gamma \approx \frac{1}{\sqrt{N}}$$

How long does it take?



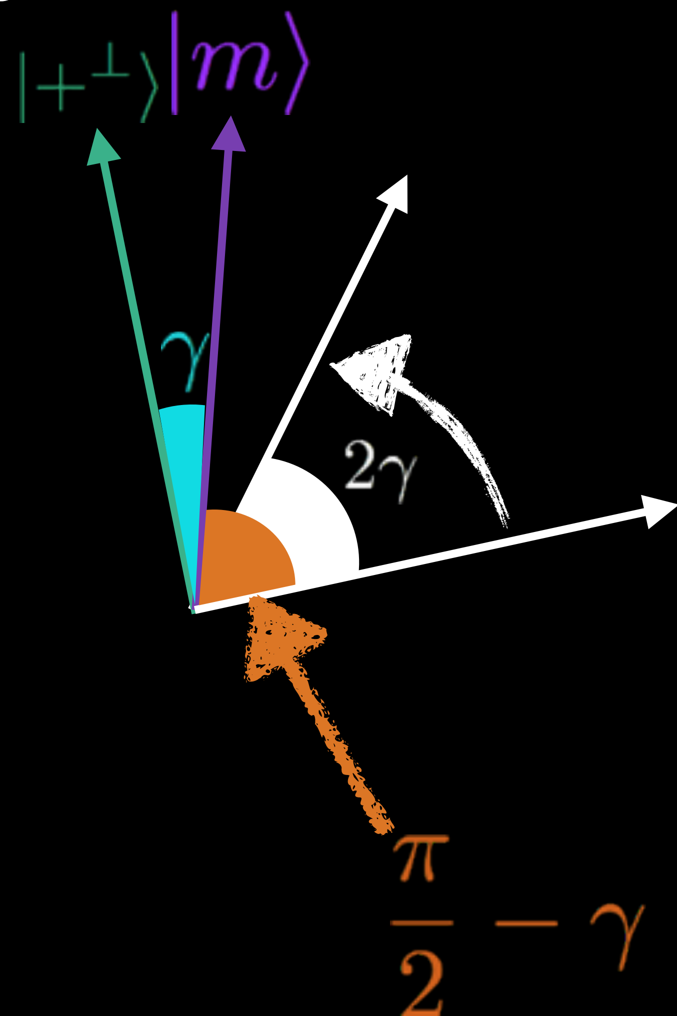
$$\cos\gamma = \langle +^{\perp}|m\rangle$$

$$\sin\gamma = \langle +|m\rangle = \frac{1}{\sqrt{N}}$$

$$\gamma \approx \frac{1}{\sqrt{N}}$$

How many iterations to get as close as possible?

How long does it take?



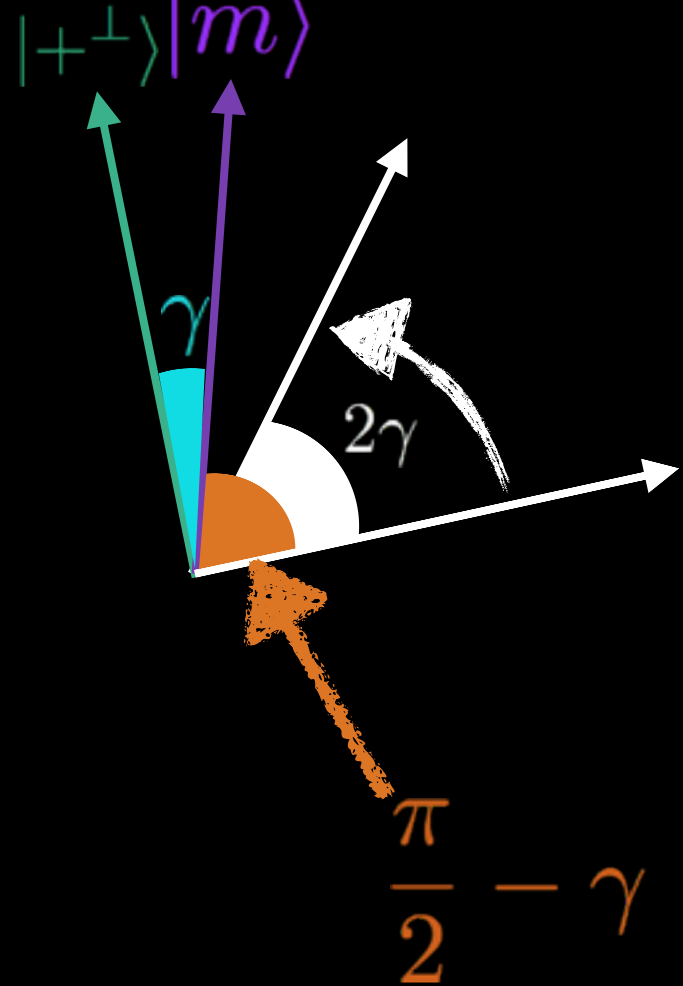
$$\cos\gamma = \langle +^{\perp}|m\rangle$$

$$\sin\gamma = \langle +|m\rangle = \frac{1}{\sqrt{N}}$$

$$\gamma \approx \frac{1}{\sqrt{N}}$$

How many iterations to get as close as possible?

How long does it take?



$$\cos\gamma = \langle +^{\perp} | m \rangle$$

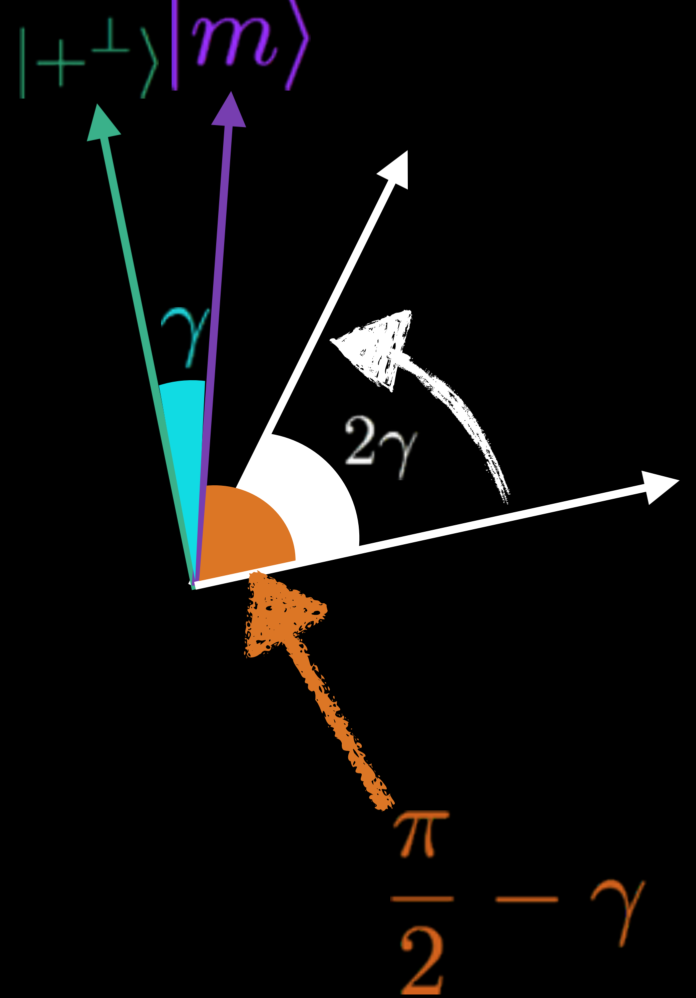
$$\sin\gamma = \langle + | m \rangle = \frac{1}{\sqrt{N}}$$

$$\gamma \approx \frac{1}{\sqrt{N}}$$

How many iterations to get as close as possible?

$$\frac{\pi}{2} - \gamma$$

How long does it take?



$$\cos\gamma = \langle +^\perp | m\rangle$$

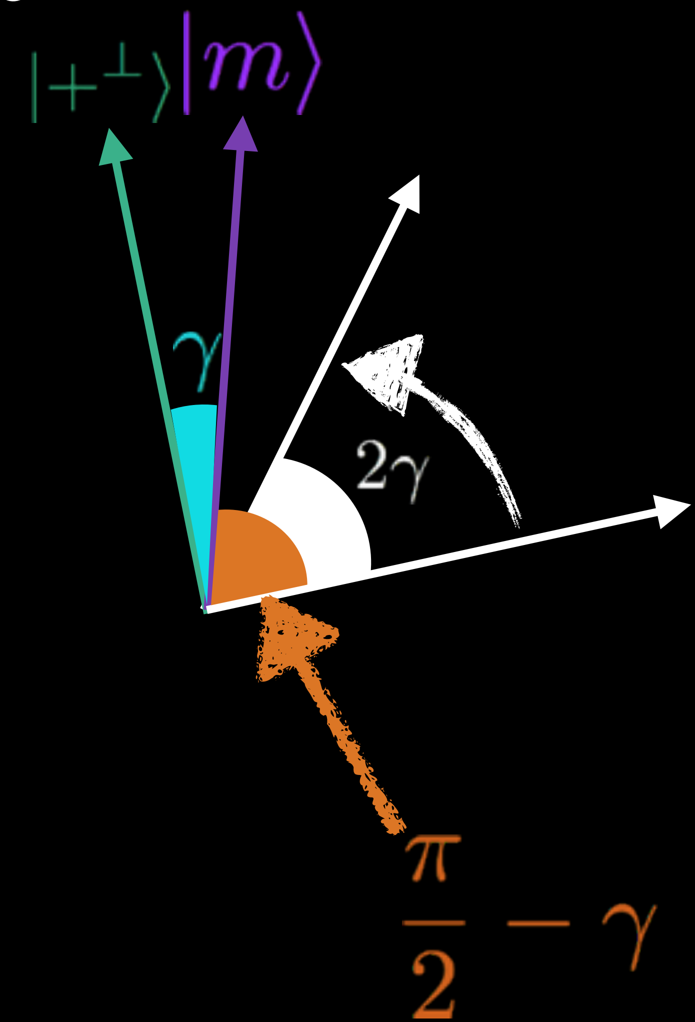
$$\sin\gamma = \langle + | m\rangle = \frac{1}{\sqrt{N}}$$

$$\gamma \approx \frac{1}{\sqrt{N}}$$

How many iterations to get as close as possible?

$$\frac{\frac{\pi}{2} - \gamma}{2\gamma}$$

How long does it take?



$$\cos\gamma = \langle +^\perp | m \rangle$$

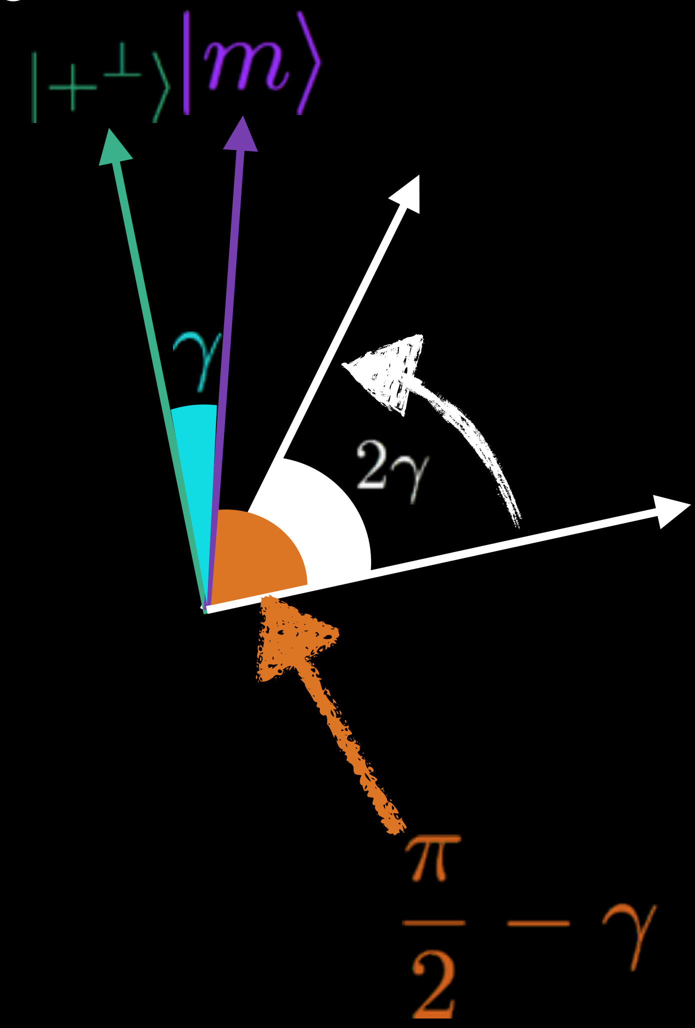
$$\sin\gamma = \langle + | m \rangle = \frac{1}{\sqrt{N}}$$

$$\boxed{\gamma \approx \frac{1}{\sqrt{N}}}$$

How many iterations to get as close as possible?

$$\frac{\frac{\pi}{2} - \gamma}{2\gamma} = \frac{\pi}{4\gamma} - \frac{1}{2} \approx \sqrt{N}$$

How long does it take?



$$\cos\gamma = \langle +^{\perp} | m \rangle$$

$$\sin\gamma = \langle + | m \rangle = \frac{1}{\sqrt{N}}$$

$$\gamma \approx \frac{1}{\sqrt{N}}$$

How many iterations to get as close as possible?

$$\frac{\frac{\pi}{2} - \gamma}{2\gamma} = \frac{\pi}{4\gamma} - \frac{1}{2} \approx \sqrt{N}$$

Summary: Grover's search

Summary: Grover’s search

- Grover’s search is very general!

Summary: Grover's search

- Grover's search is very general!

- A 'query' might involving running some other classical/quantum algorithm

Summary: Grover's search

- Grover's search is very general!
- A 'query' might involving running some other classical/quantum algorithm
- Gives a generic square-root speedup

Summary: Grover's search

- Grover's search is very general!
- A 'query' might involving running some other classical/quantum algorithm
- Gives a generic square-root speedup
    - But not always faster than classical!

Phase Estimation

Phase Estimation

- An important quantum computing primitive

Phase Estimation

- An important quantum computing primitive
- Often used as an ingredient in more complex algorithms:
    - Integer factorisation
    - Matrix inversion
    - Quantum counting
    - Quantum walks
    - …

Phase Estimation

Phase Estimation

$$A\vec{\mathbf{x}} = \lambda\vec{\mathbf{x}}$$

Phase Estimation

$$A\vec{\mathbf{x}} = \lambda\vec{\mathbf{x}}$$

Eigenvector

Phase Estimation

$$A\vec{\mathbf{x}} = \lambda\vec{\mathbf{x}}$$

Eigenvector

Eigenvalue

Phase Estimation

$$A\vec{\mathbf{x}} = \lambda\vec{\mathbf{x}}$$

Eigenvector

Eigenvalue

For a *unitary* U:

Phase Estimation

$$A\vec{\mathbf{x}} = \lambda\vec{\mathbf{x}}$$

Eigenvector

Eigenvalue

For a *unitary* U:

$$U|x\rangle = e^{2\pi i\theta}|x\rangle$$

Phase Estimation

$$A\vec{\mathbf{x}} = \lambda\vec{\mathbf{x}}$$

Eigenvector

Eigenvalue

For a *unitary* U:

$$U|x\rangle = e^{2\pi i\theta}|x\rangle$$

Phase Estimation

$$A\vec{\mathbf{x}} = \lambda\vec{\mathbf{x}}$$

Eigenvector

Eigenvalue

For a *unitary* U:

$$U|x\rangle = e^{2\pi i\theta}|x\rangle$$

Phase Estimation

$$A\vec{\mathbf{x}} = \lambda\vec{\mathbf{x}}$$

Eigenvector

Eigenvalue

Phase

For a *unitary* U:

$$U|x\rangle = e^{2\pi i\theta}|x\rangle$$

Phase Estimation

$$A\vec{\mathbf{x}} = \lambda\vec{\mathbf{x}}$$

Eigenvector

Eigenvalue

Phase

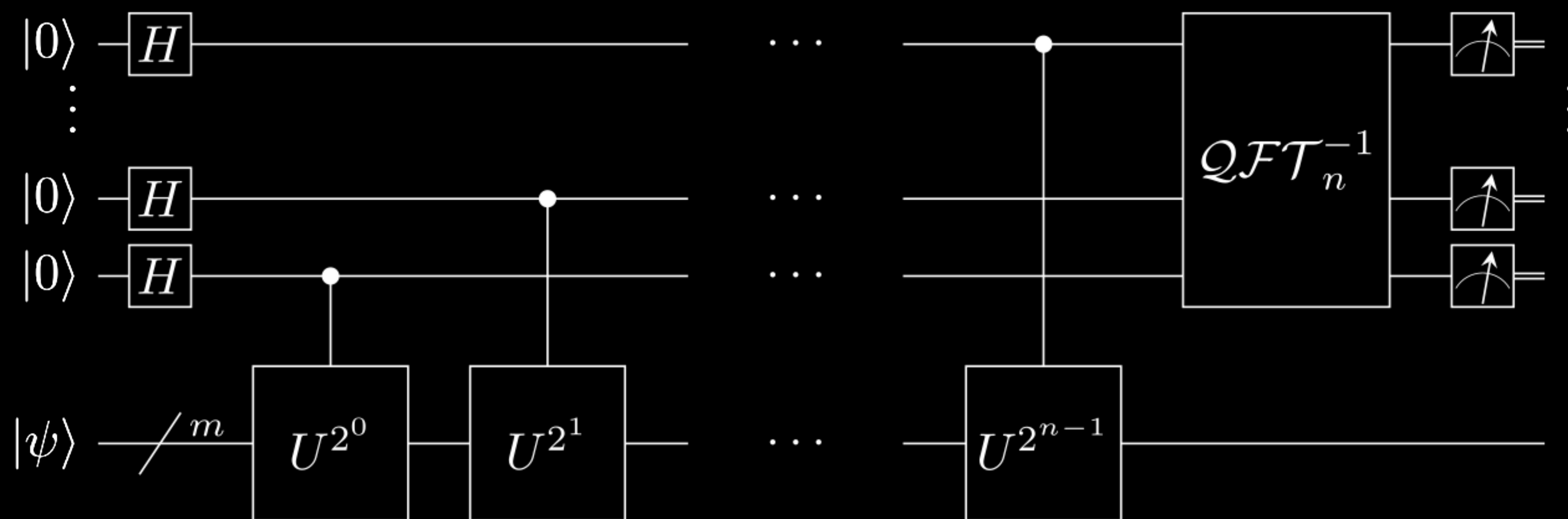
For a *unitary* U:

$$U|x\rangle = e^{2\pi i\theta}|x\rangle$$

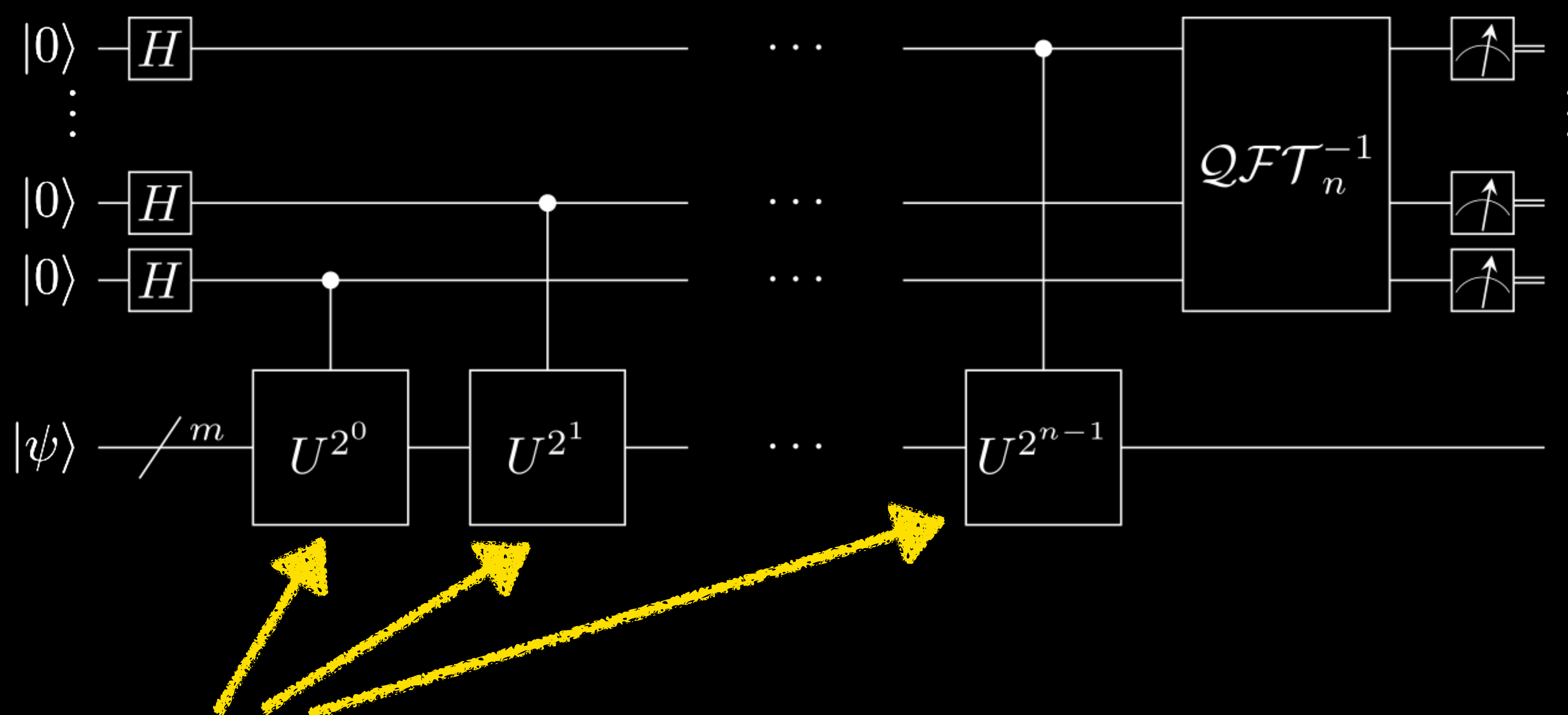
Phase Estimation: Given U and $|x\rangle$, estimate θ

Phase Estimation Circuit

Phase Estimation Circuit

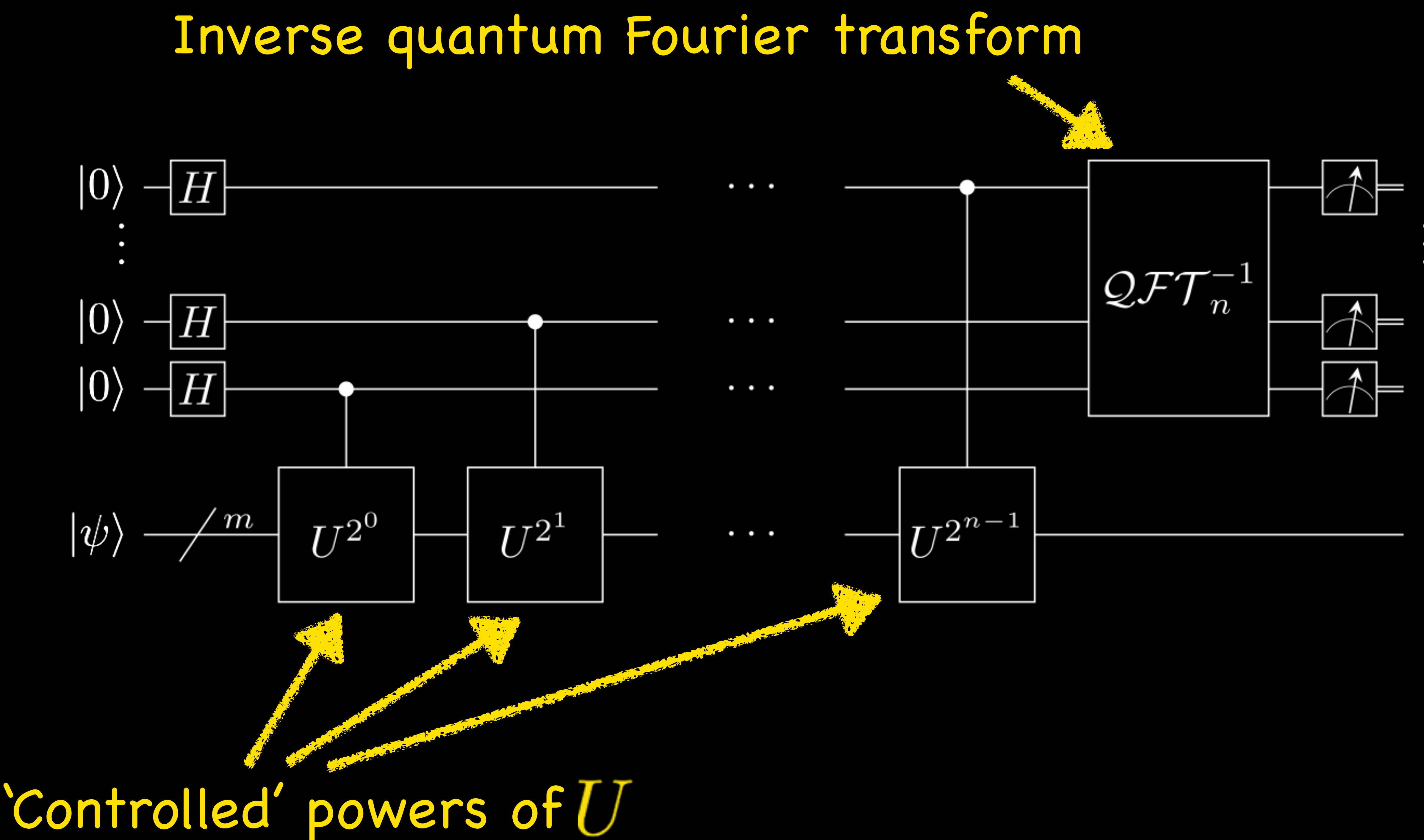


Phase Estimation Circuit

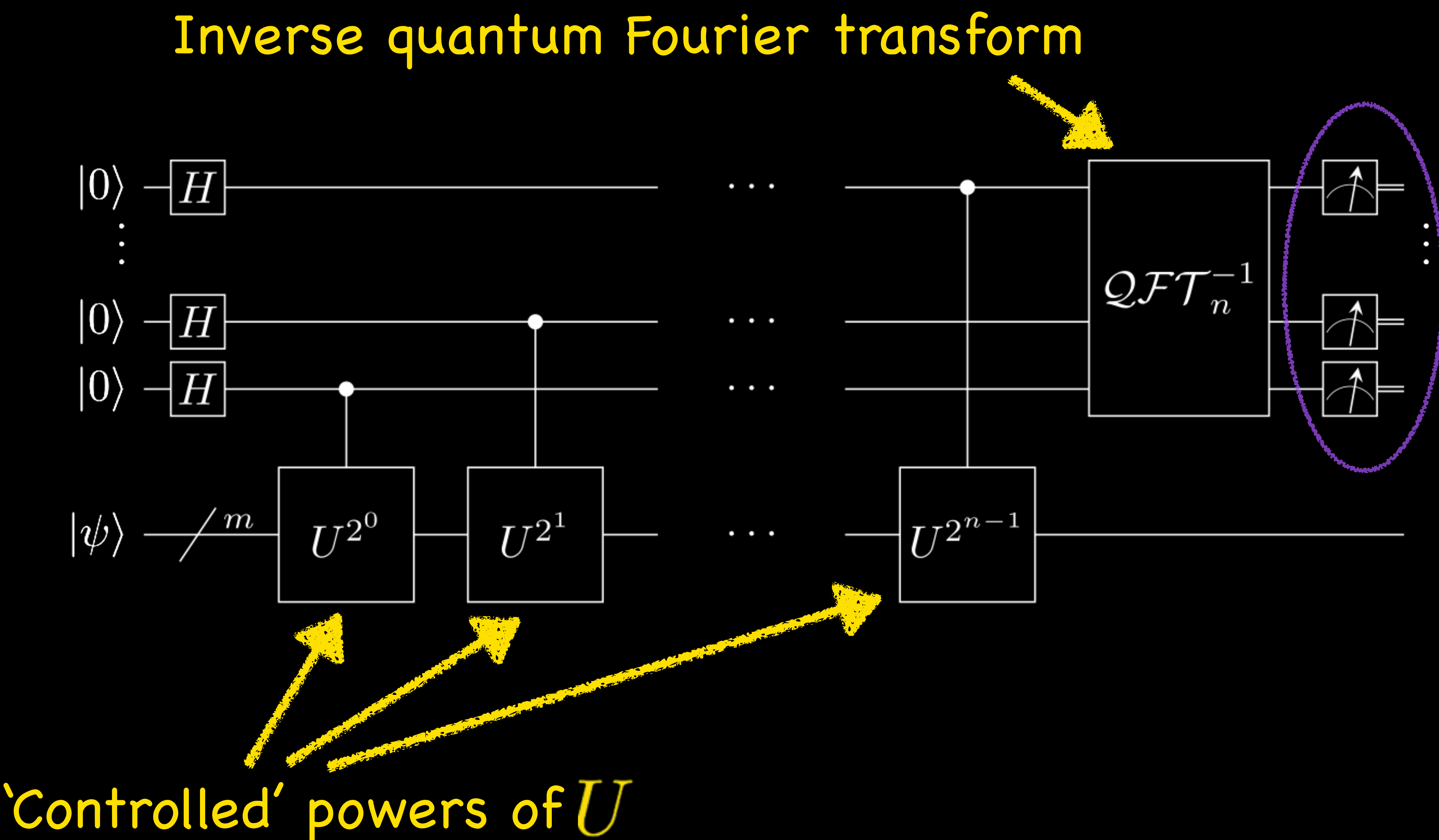


'Controlled' powers of U

Phase Estimation Circuit

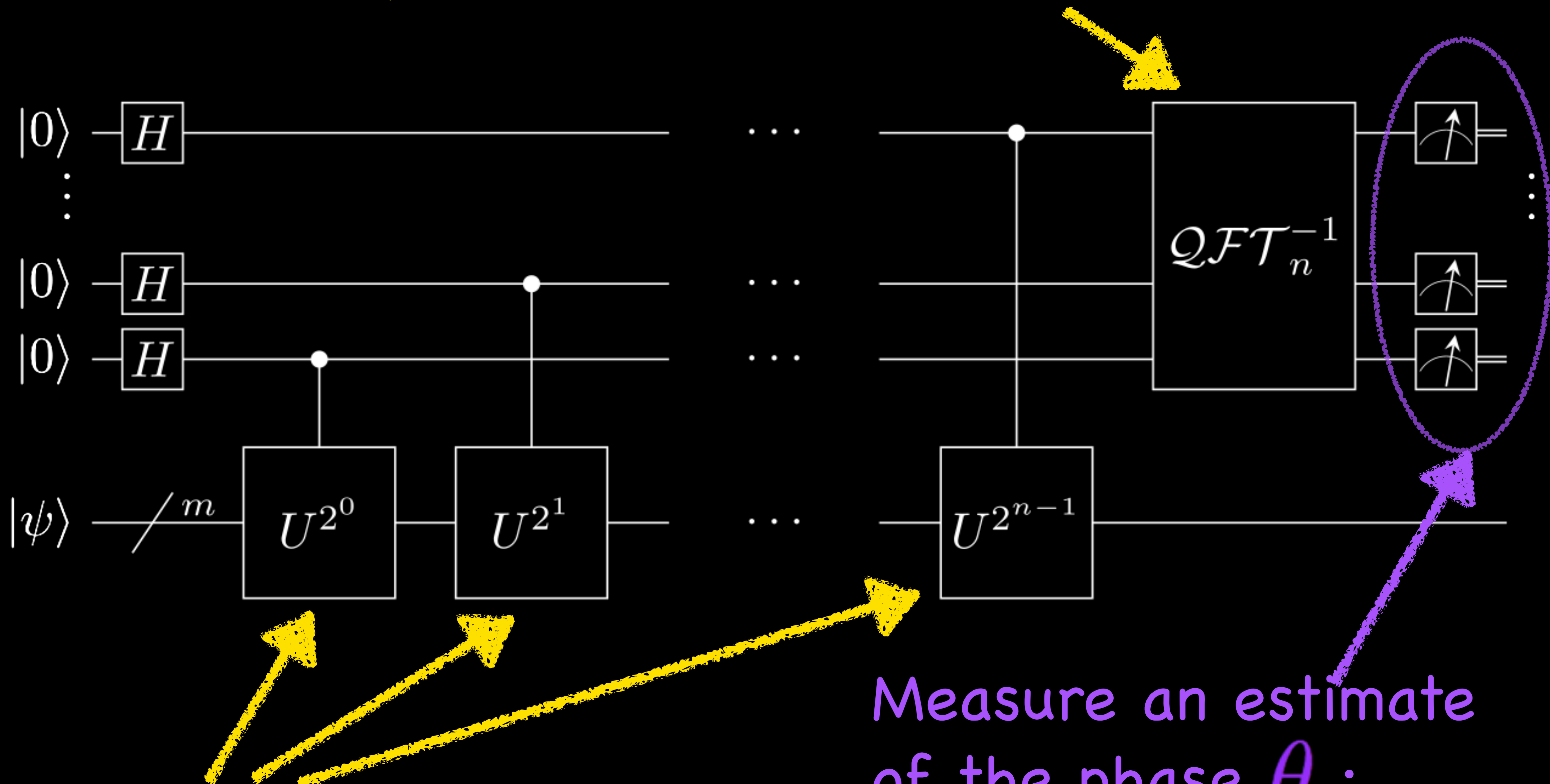


Phase Estimation Circuit



Phase Estimation Circuit

Inverse quantum Fourier transform



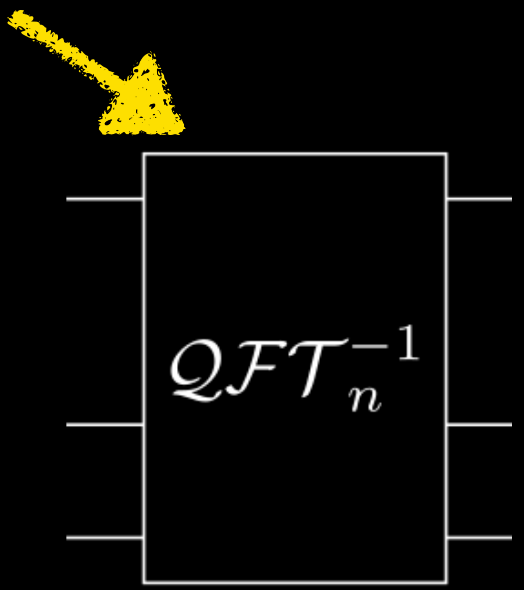
'Controlled' powers of U

Measure an estimate of the phase θ :

$$U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$$

Phase Estimation Circuit

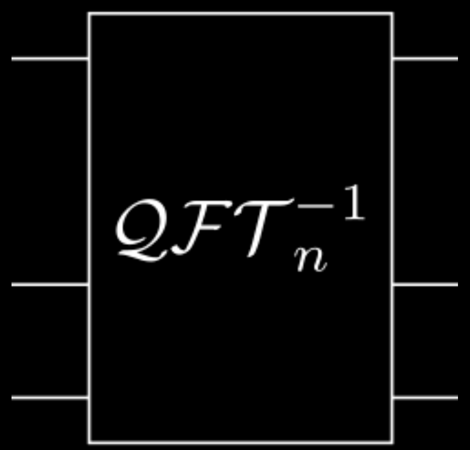
Inverse quantum Fourier transform



Phase Estimation Circuit

Inverse quantum Fourier transform

Quantum Fourier transform (QFT)

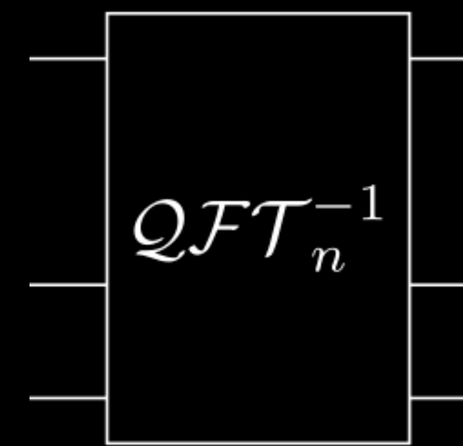


Phase Estimation Circuit

Inverse quantum Fourier transform

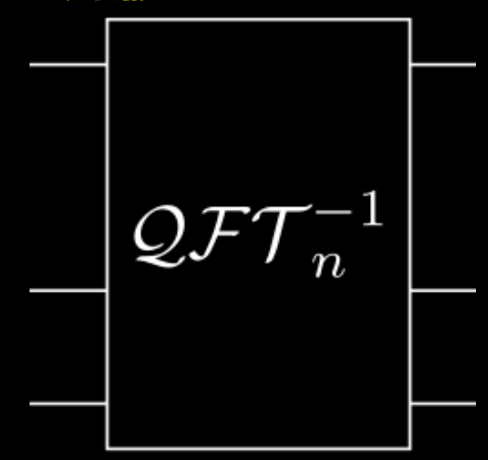
Quantum Fourier transform (QFT)

- Can be seen as a generalisation of the Hadamard gate



Phase Estimation Circuit

Inverse quantum Fourier transform

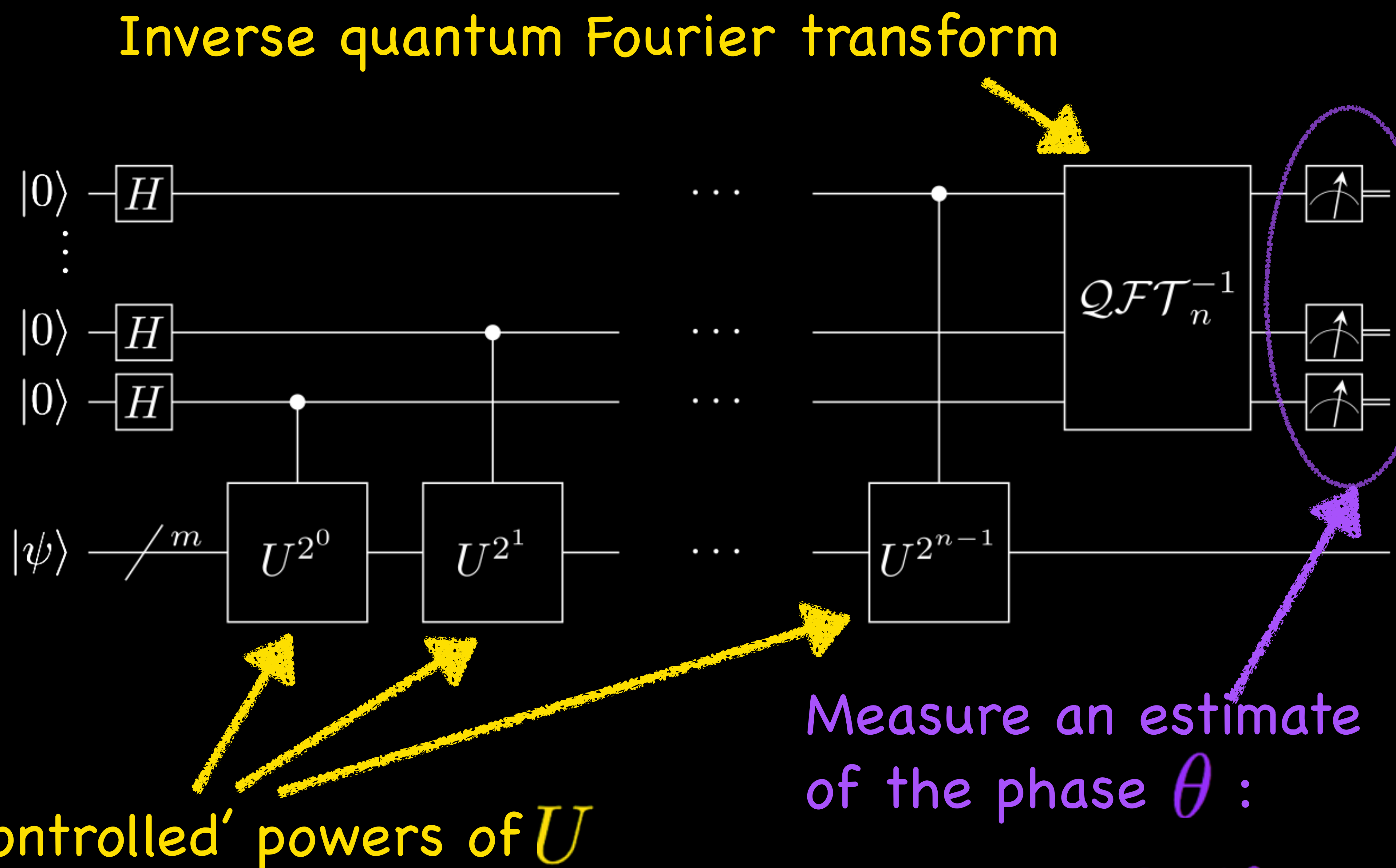


Quantum Fourier transform (QFT)

- Can be seen as a generalisation of the Hadamard gate
- Formally:

$$QFT_N|x\rangle = \frac{1}{\sqrt{N}} \sum_{y \in \{0,\ldots,N-1\}} e^{\frac{2\pi i x y}{N}} |y\rangle$$

Phase Estimation Circuit



$$U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$$

Factoring

Arguably the most famous application of quantum computers

Factoring

Arguably the most famous application of quantum computers

N

Factoring

Arguably the most famous application of quantum computers

$$N = a \times b$$

Factoring

Arguably the most famous application of quantum computers

$$N = a \times b$$

15

Factoring

Arguably the most famous application of quantum computers

$$N = a \times b$$

15 = 3 x 5

Factoring

Arguably the most famous application of quantum computers

$$N = a \times b$$

15 = 3 x 5

Important modern crypto-systems (e.g. RSA) rely on this problem being intractable for computers.

Factoring

Arguably the most famous application of quantum computers

$$N = a \times b$$

15 = 3 x 5

Important modern crypto-systems (e.g. RSA) rely on this problem being intractable for computers.

But a quantum computer can solve it quickly!

How quickly?

How quickly?

Recommended key size for RSA is 2048 bits

How quickly?

Recommended key size for RSA is 2048 bits

How quickly?

Recommended key size for RSA is 2048 bits

22701801293785014193580405120204586741061235962766583907094021879215171483119139894870133091111044901683400949483846818299518041763507948922590774925466088171879259465921026597046700449819899096862039460017743094473811056991294128542891880855362707407670722593737772666973440977361243336397308051763091506836310795312607239520365290032105848839507981452307299417185715796297454995023505316040919859193718023307414880446217922800831766040938656344571034778553457121080530736394535923932651866030515041060966437313323672831539323500067937107541955437362433248361242525945868802353916766181532375855504886901432221349733

How quickly?

Recommended key size for RSA is 2048 bits

22701801293785014193580405120204586741061235962766583907094021879215 1
71483119139894870133091111044901683400949483846818299518041763507948 9
22590774925466088171879259465921026597046700449819899096862039460017 7
43094473811056991294128542891880855362707407670722593737772666973440 9
77361243336397308051763091506836310795312607239520365290032105848839 5
07981452307299417185715796297454995023505316040919859193718023307414 8
80446217922800831766040938656344571034778553457121080530736394535923 9
32651866030515041060966437313323672831539323500067937107541955437362 4
3324836124252594586880235391676618153237585550488690143222134973 3

= ? x ?

How quickly?

Recommended key size for RSA is 2048 bits

22701801293785014193580405120204586741061235962766583907094021879215171483119139894870133091111044901683400949483846818299518041763507948922590774925466088171879259465921026597046700449819899096862039460017743094473811056991294128542891880855362707407670722593737772666973440977361243336397308051763091506836310795312607239520365290032105848839507981452307299417185715796297454995023505316040919859193718023307414880446217922800831766040938656344571034778553457121080530736394535923932651866030515041060966437313323672831539323500067937107541955437362433248361242525945868802353916766181532375855504886901432221349733

= ? x ?

Best known classical algorithm : ~ 1 Billion Years

How quickly?

Recommended key size for RSA is 2048 bits

22701801293785014193580405120204586741061235962766583907094021879215171483119139894870133091111044901683400949483846818299518041763507948922590774925466088171879259465921026597046700449819899096862039460017743094473811056991294128542891880855362707407670722593737772666973440977361243336397308051763091506836310795312607239520365290032105848839507981452307299417185715796297454995023505316040919859193718023307414880446217922800831766040938656344571034778553457121080530736394535923932651866030515041060966437313323672831539323500067937107541955437362433248361242525945868802353916766181532375855504886901432221349733

= ? x ?

Best known classical algorithm : ~ 1 Billion Years

Shor's algorithm : ~ 100 Seconds*

*For a quantum computer running at ~1GHz.
Source: "A Compare between Shor's quantum factoring algorithm and General Number Field Sieve", Hamdi et al.

Shor's Algorithm

Shor's Algorithm

Reduces the problem of factoring to the problem of **period finding**

Shor's Algorithm

Reduces the problem of factoring to the problem of **period finding**

Uses a quantum algorithm for fast period finding.

Shor's Algorithm

Reduces the problem of factoring to the problem of **period finding**

Uses a quantum algorithm for fast period finding.

Shor's Algorithm

1. If N is even, return $f=2$
2. If $N=p^k$ for p prime, return p
3. Randomly choose $1<q<N-1$
   3.1. If $f=gcd(q,N)>1$, return f
4. **Determine the order k of q modulo N**
   4.1. If k is odd, repeat from step 3
5. Write $k=2l$ and determine $q^l \bmod N$ with $1<r<N$
   5.1. If $1<f=gcd(r-1,N)<N$, return f
   5.2. If $1<f=gcd(r+1,N)<N$, return f
   5.3. Else, repeat from step 3

Shor's Algorithm

Reduces the problem of factoring to the problem of **period finding**

Uses a quantum algorithm for fast period finding.

Shor's Algorithm

1. If N is even, return $f=2$
2. If $N=p^k$ for p prime, return p
3. Randomly choose $1<q<N-1$
   3.1. If $f=gcd(q,N)>1$, return f
4. **Determine the order k of q modulo N**
   4.1. If k is odd, repeat from step 3
5. Write $k=2l$ and determine $q^l \mod N$ with $1<r<N$
   5.1. If $1<f=gcd(r-1,N)<N$, return f
   5.2. If $1<f=gcd(r+1,N)<N$, return f
   5.3. Else, repeat from step 3

Fast Quantum algorithm using Phase Estimation

Shor's Algorithm

Reduces the problem of factoring to the problem of **period finding**

Uses a quantum algorithm for fast period finding.

Shor's Algorithm

1. If N is even, return $f=2$
2. If $N=p^k$ for p prime, return p
3. Randomly choose $1<q<N-1$
    - 3.1. If $f=gcd(q,N)>1$, return f
4. **Determine the order k of q modulo N**
    - 4.1. If k is odd, repeat from step 3
5. Write $k=2l$ and determine $q^l \mod N$ with $1<r<N$
    - 5.1. If $1<f=gcd(r-1,N)<N$, return f
    - 5.2. If $1<f=gcd(r+1,N)<N$, return f
    - 5.3. Else, repeat from step 3

Fast Quantum algorithm using Phase Estimation

Shor's Algorithm

Reduces the problem of factoring to the problem of **period finding**

Uses a quantum algorithm for fast period finding.

Shor's Algorithm

1. If N is even, return $f=2$
2. If $N=p^k$ for p prime, return p
3. Randomly choose $1<q<N-1$
   3.1. If $f=gcd(q,N)>1$, return f
4. **Determine the order k of q modulo N**
   4.1. If k is odd, repeat from step 3
5. Write $k=2l$ and determine $q^l \bmod N$ with $1<r<N$
   5.1. If $1<f=gcd(r-1,N)<N$, return f
   5.2. If $1<f=gcd(r+1,N)<N$, return f
   5.3. Else, repeat from step 3

Fast Quantum algorithm using Phase Estimation

Shor's Algorithm

Reduces the problem of factoring to the problem of **period finding**

Uses a quantum algorithm for fast period finding.

Shor's Algorithm

1. If N is even, return $f=2$
2. If $N=p^k$ for p prime, return p
3. Randomly choose $1<q<N-1$
    3.1. If $f=gcd(q,N)>1$, return f
4. **Determine the order k of q modulo N**
    4.1. If k is odd, repeat from step 3
5. Write $k=2l$ and determine $q^l \bmod N$ with $1<r<N$
    5.1. If $1<f=gcd(r-1,N)<N$, return f
    5.2. If $1<f=gcd(r+1,N)<N$, return f
    5.3. Else, repeat from step 3

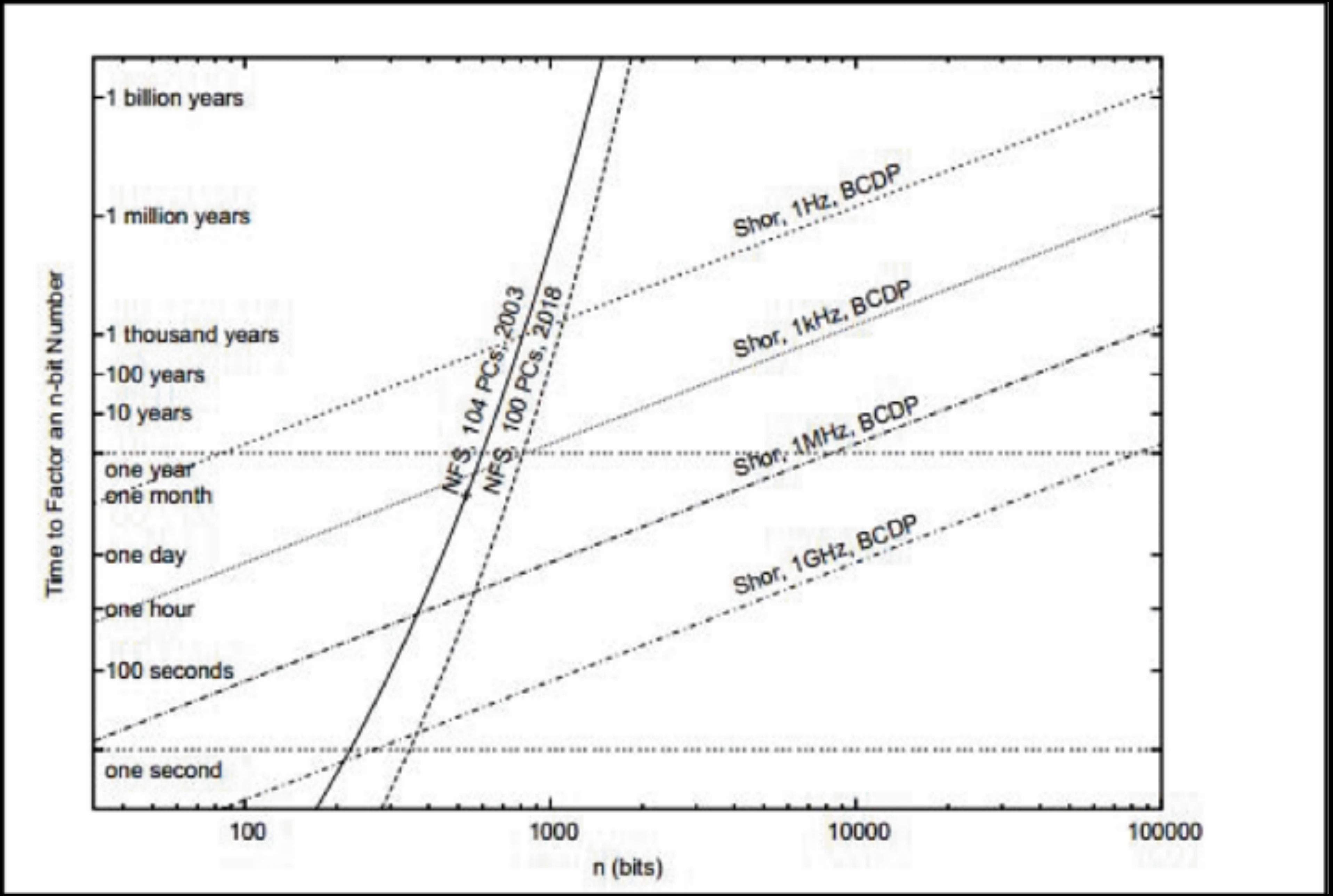
Fast Quantum algorithm using Phase Estimation

All other steps can be performed efficiently by a classical computer.

Given an *n*-bit integer:

Classical **number field sieve :** $O(2^{n^{1/3}})$

Shor's algorithm : $O(n^3)$



Source: "A Compare between Shor's quantum factoring algorithm and General Number Field Sieve", Hamdi et al.

Hamiltonian Simulation

Hamiltonian Simulation

- In quantum mechanics, physical systems are described by ‘Hamiltonians’.

Hamiltonian Simulation

- In quantum mechanics, physical systems are described by 'Hamiltonians'.
- For our purposes, these are Hermitian ($A = A^\dagger$) matrices, which we will write as H

Hamiltonian Simulation

- In quantum mechanics, physical systems are described by 'Hamiltonians'.
- For our purposes, these are Hermitian ($A = A^\dagger$) matrices, which we will write as H
- The evolution of quantum systems is governed by Schrödinger's equation:

$$i\hbar \frac{d}{dt}|\psi(t)\rangle = H(t)|\psi(t)\rangle$$

Hamiltonian Simulation

- In quantum mechanics, physical systems are described by 'Hamiltonians'.
- For our purposes, these are Hermitian ($A = A^\dagger$) matrices, which we will write as H
- The evolution of quantum systems is governed by Schrödinger's equation:

$$i\hbar \frac{d}{dt}|\psi(t)\rangle = H(t)|\psi(t)\rangle$$

- When H doesn't change over time, the solution to this equation is

$$|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$$

$$|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$$

This is a unitary matrix

$$|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$$

This is a unitary matrix

Hamiltonian Simulation:

Given a Hamiltonian H, construct a quantum circuit that approximates e^{-iHt}

$$|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$$

This is a unitary matrix

Hamiltonian Simulation:

Given a Hamiltonian H, construct a quantum circuit that approximates e^{-iHt}

There are a number of quantum algorithms that can do this efficiently for certain types of Hamiltonian

Why is this useful?

Why is this useful?

- Physical systems are quantum mechanical

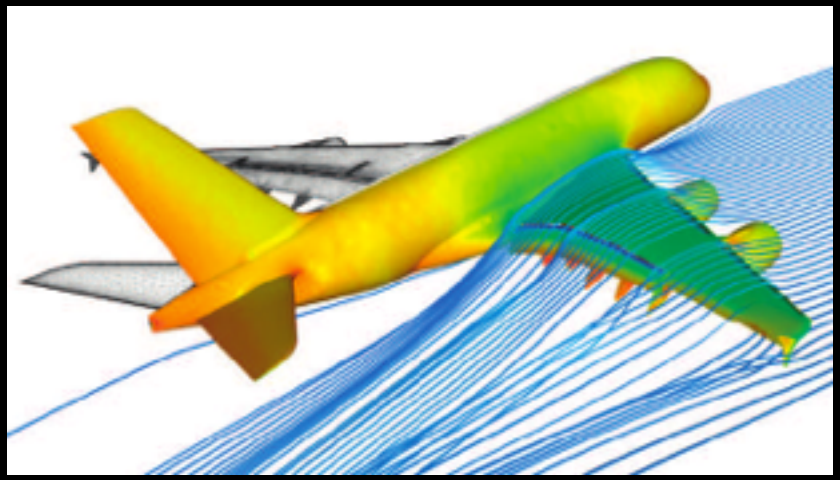
Why is this useful?

- Physical systems are quantum mechanical
- If we want to simulate them on a computer, we're going to need a quantum one

Why is this useful?

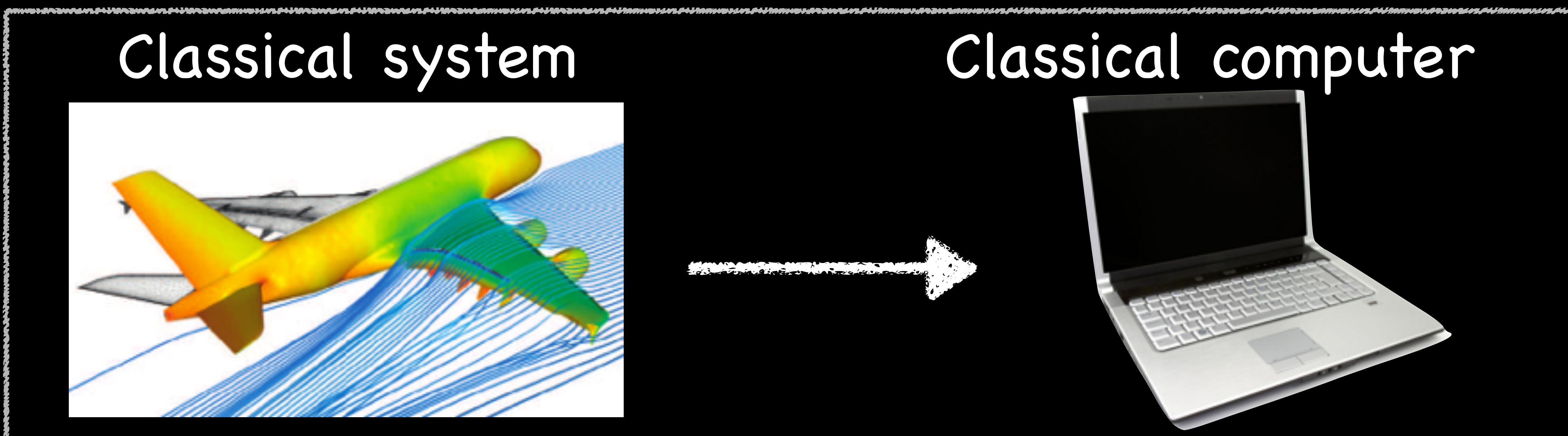
- Physical systems are quantum mechanical
- If we want to simulate them on a computer, we're going to need a quantum one

Classical system



Why is this useful?

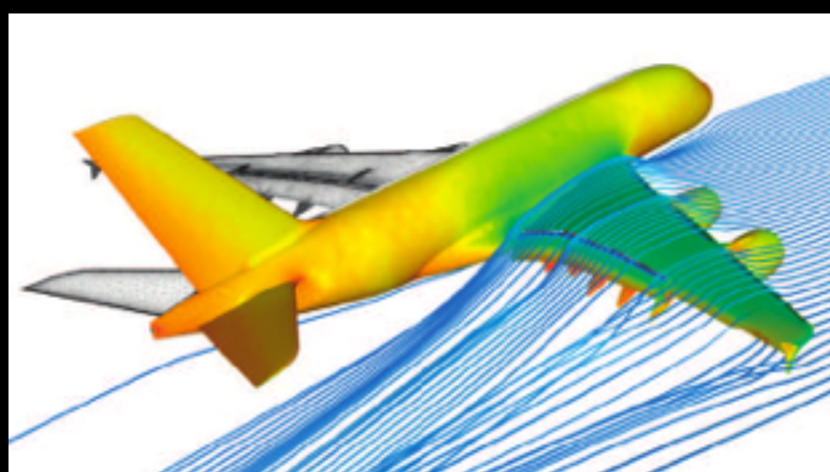
- Physical systems are quantum mechanical
- If we want to simulate them on a computer, we’re going to need a quantum one



Why is this useful?

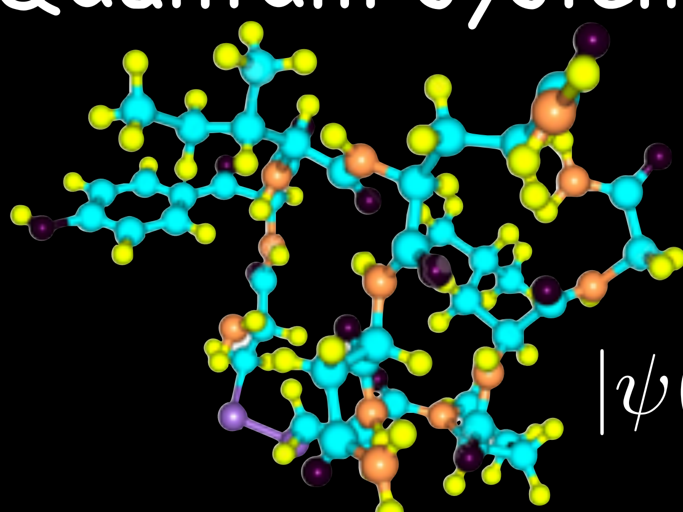
- Physical systems are quantum mechanical
- If we want to simulate them on a computer, we're going to need a quantum one

Classical system



Classical computer

Quantum system

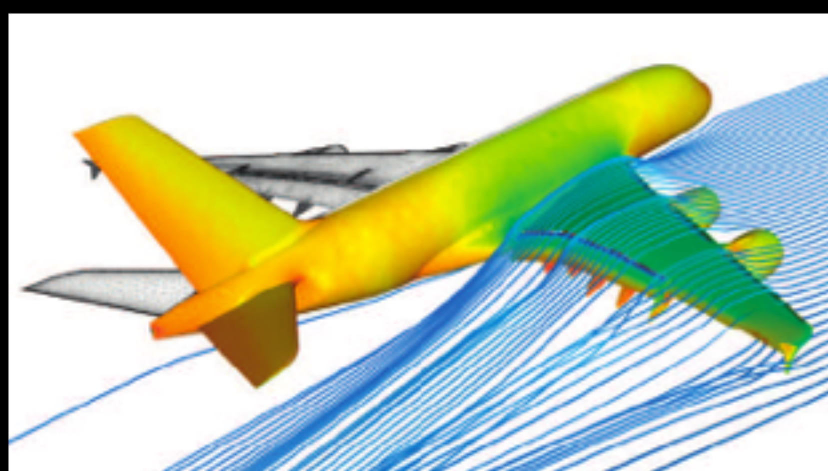


$$|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$$

Why is this useful?

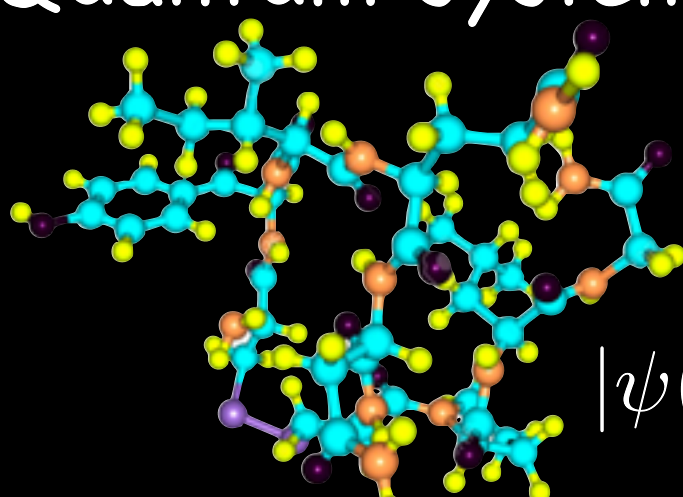
- Physical systems are quantum mechanical
- If we want to simulate them on a computer, we're going to need a quantum one

Classical system

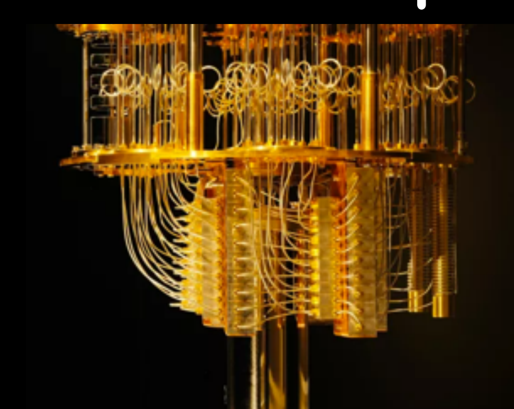


Classical computer

Quantum system



Quantum computer



$$|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$$

HHL

Named after Harrow, Hassidim, and Lloyd, who invented it in 2008

HHL

Named after Harrow, Hassidim, and Lloyd, who invented it in 2008

Attacks one of the most fundamental tasks in science — solving systems of linear equations:

HHL

Named after Harrow, Hassidim, and Lloyd, who invented it in 2008

Attacks one of the most fundamental tasks in science — solving systems of linear equations:

$$A\vec{\mathbf{x}} = \vec{\mathbf{b}}$$

HHL

Named after Harrow, Hassidim, and Lloyd, who invented it in 2008

Attacks one of the most fundamental tasks in science — solving systems of linear equations:

$$A\vec{\mathbf{x}} = \vec{\mathbf{b}}$$

Solve for $\vec{\mathbf{x}}$

HHL

Named after Harrow, Hassidim, and Lloyd, who invented it in 2008

Attacks one of the most fundamental tasks in science — solving systems of linear equations:

$$A\vec{\mathbf{x}} = \vec{\mathbf{b}}$$

Solve for $\vec{\mathbf{x}}$

Classically: Takes time polynomial in the size of the matrix

HHL

Named after Harrow, Hassidim, and Lloyd, who invented it in 2008

Attacks one of the most fundamental tasks in science — solving systems of linear equations:

$$A\vec{\mathbf{x}} = \vec{\mathbf{b}}$$

Solve for $\vec{\mathbf{x}}$

Classically: Takes time polynomial in the size of the matrix

HHL '*solves*' this problem in logarithmic time

HHL

HHL

$$A\vec{\mathbf{x}} = \vec{\mathbf{b}}$$

HHL

$$A\vec{\mathbf{x}} = \vec{\mathbf{b}}$$

Input: $|b\rangle, A$

HHL

$$A\vec{\mathbf{x}} = \vec{\mathbf{b}}$$

Input: $|b\rangle, A$

Output: quantum state $|x\rangle$

HHL

$$A\vec{\mathbf{x}} = \vec{\mathbf{b}}$$

Input: $|b\rangle, A$

Output: quantum state $|x\rangle$

So long as we can prepare $|b\rangle$, and only need to know global properties of $|x\rangle$, this is useful.

Rough Outline

Rough Outline

Input: A
$|b\rangle$

Rough Outline

Input: A → $U = e^{iA}$

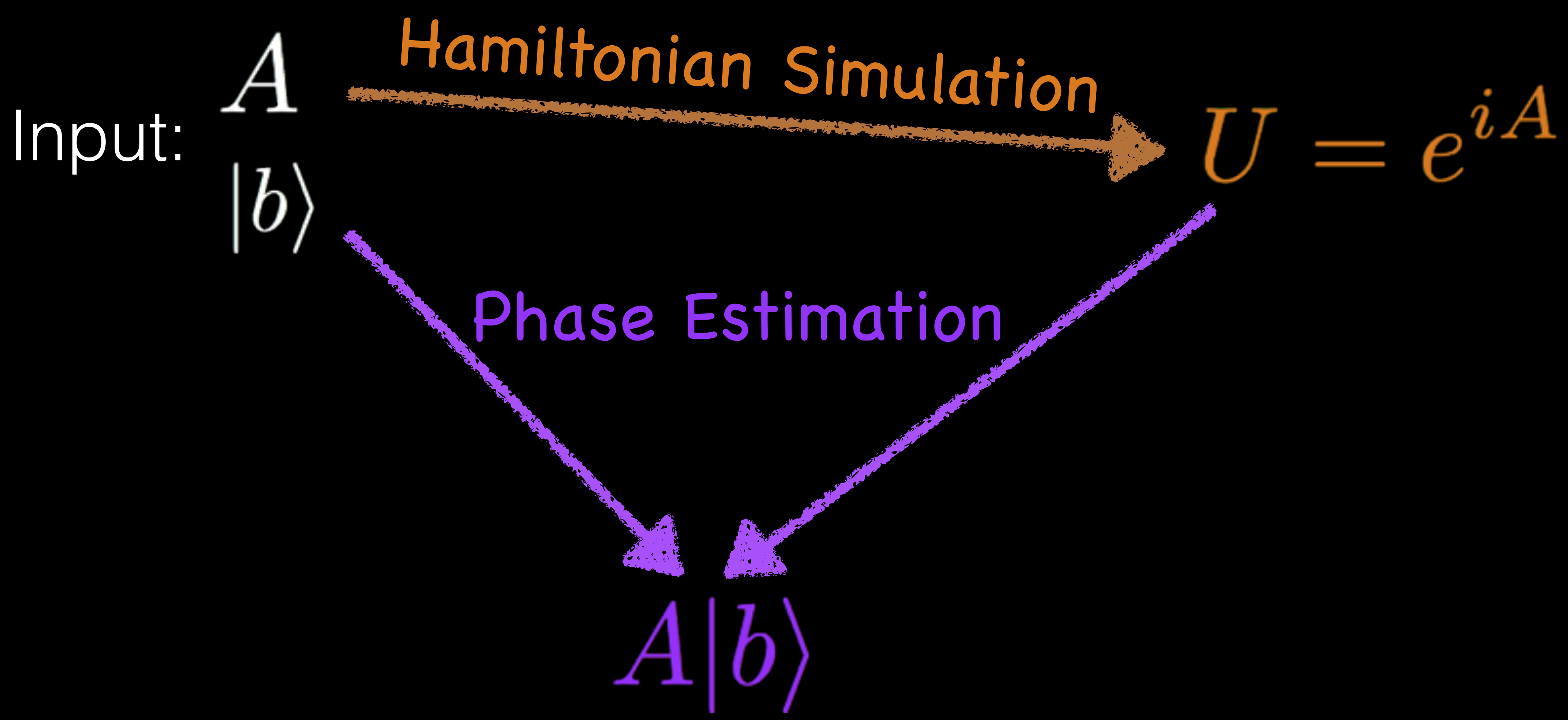
$|b\rangle$

Rough Outline

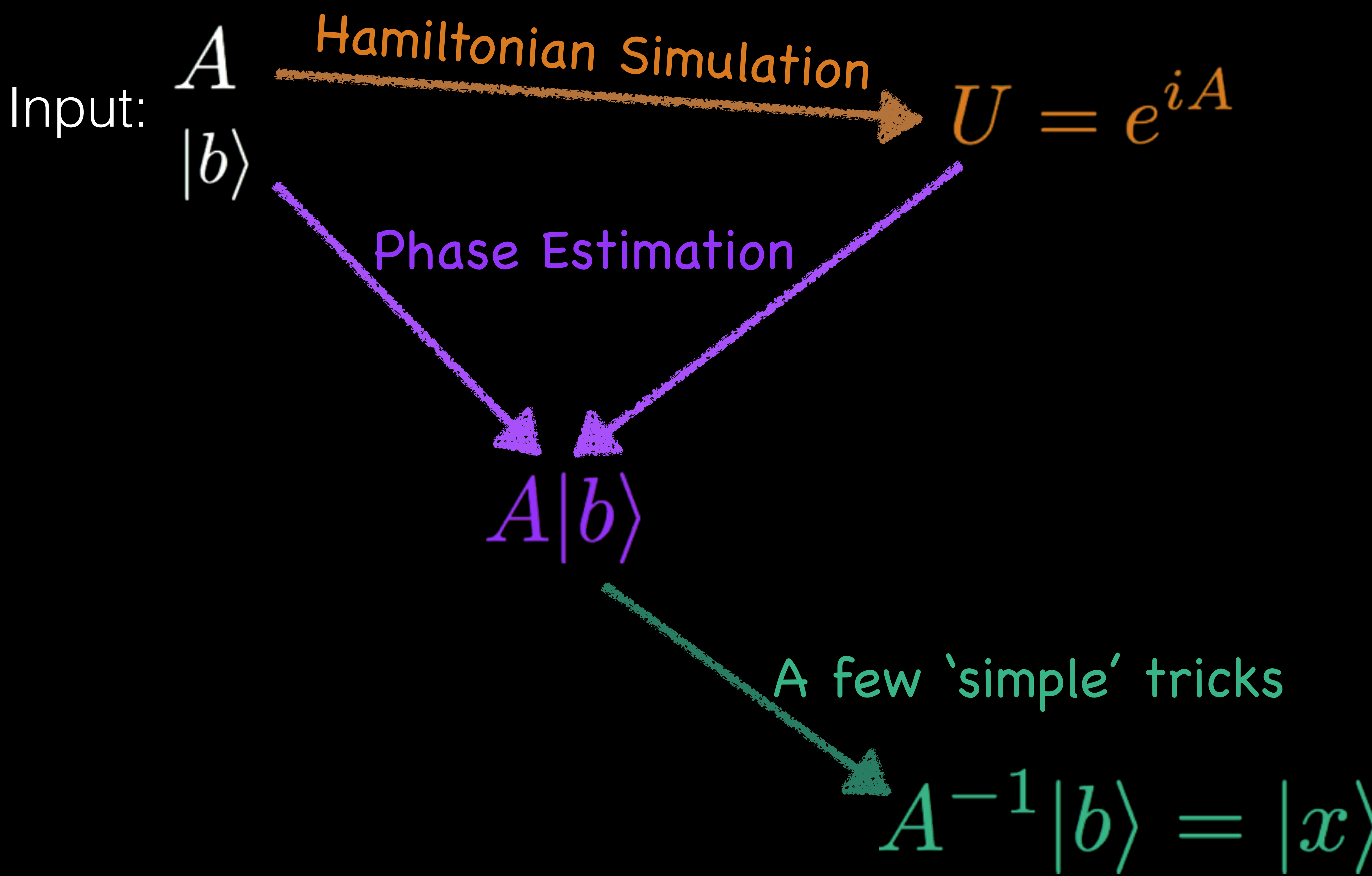
Input: A $|b\rangle$

Hamiltonian Simulation → $U = e^{iA}$

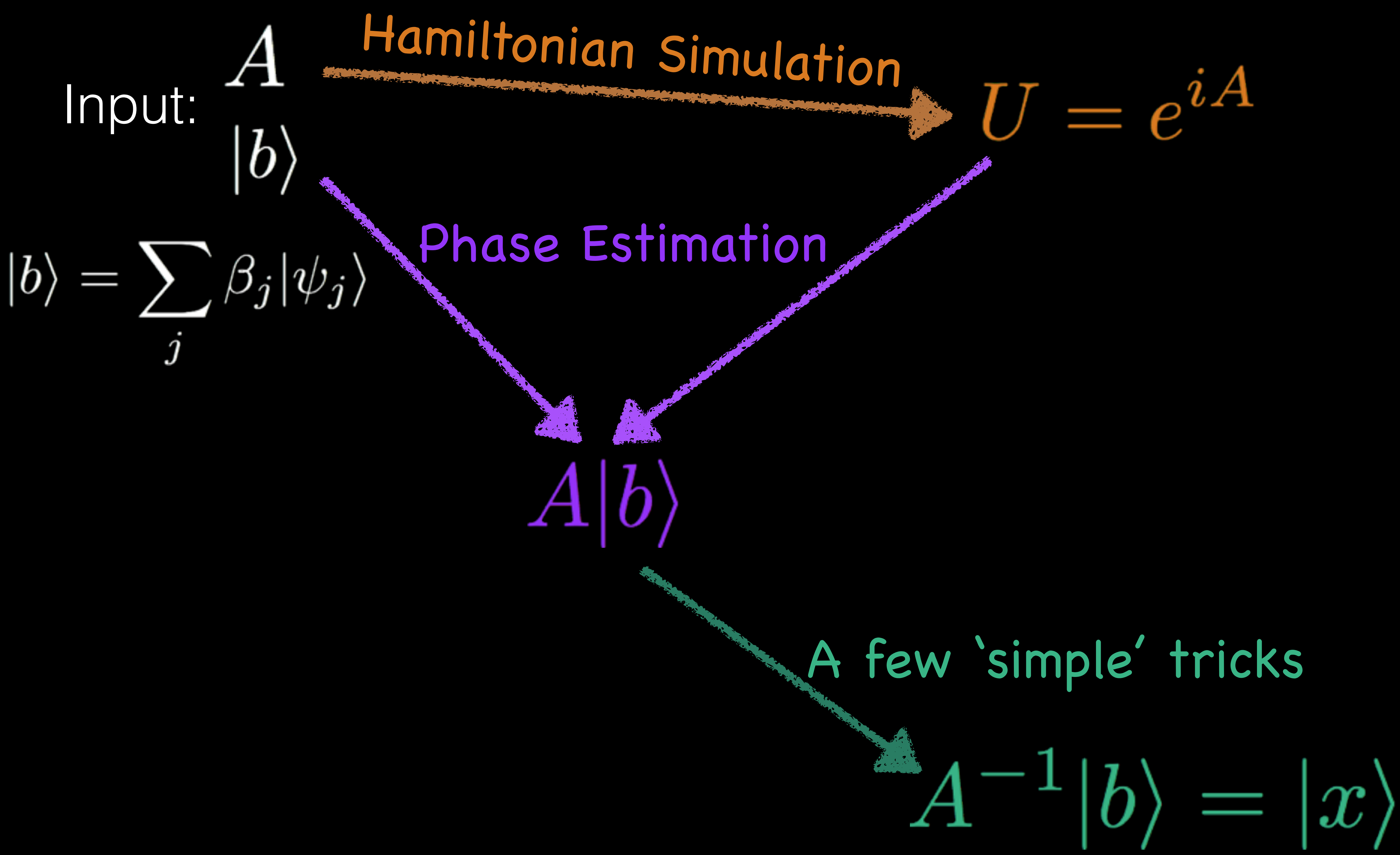
Rough Outline



Rough Outline



Rough Outline



Rough Outline

Input: A

$|b\rangle$

Hamiltonian Simulation

$U = e^{iA}$

$$|b\rangle = \sum_j \beta_j |\psi_j\rangle$$

Phase Estimation

$$A|b\rangle = \sum_j \beta_j |\lambda_j\rangle|\psi_j\rangle$$

A few ‘simple’ tricks

$$A^{-1}|b\rangle = |x\rangle$$

Rough Outline

Input: A, $|b\rangle$

$|b\rangle = \sum_j \beta_j |\psi_j\rangle$

Hamiltonian Simulation → $U = e^{iA}$

Phase Estimation → $A|b\rangle = \sum_j \beta_j |\lambda_j\rangle |\psi_j\rangle$

A few 'simple' tricks → $A^{-1}|b\rangle = |x\rangle$

$$\sum_j \beta_j |\psi_j\rangle \left(\sqrt{1 - \frac{1}{\lambda_j}} |0\rangle + \frac{1}{\lambda_j} |1\rangle \right)$$

Rough Outline

Input: A, $|b\rangle$

A → Hamiltonian Simulation → $U = e^{iA}$

$$|b\rangle = \sum_j \beta_j |\psi_j\rangle$$

Phase Estimation

$$A|b\rangle = \sum_j \beta_j |\lambda_j\rangle |\psi_j\rangle$$

A few 'simple' tricks

$$A^{-1}|b\rangle = |x\rangle$$

$$\sum_j \beta_j |\psi_j\rangle \left(\sqrt{1 - \frac{1}{\lambda_j}} |0\rangle + \frac{1}{\lambda_j} |1\rangle \right)$$

$$\sum_j \frac{\beta_j}{\lambda_j} |\psi_j\rangle = A^{-1}|b\rangle = |x\rangle$$

Applications

Applications

- Solving Systems of Differential Equations
  - E.g. Finite Element Method (FEM)
- Data fitting
- Various tasks in machine learning
  - E.g. clustering, support-vector machines, principal component analysis

Run-time

Run-time

For a system of ***n*** equations:

Run-time

For a system of ***n*** equations:

Classical : polynomial in ***n***

Run-time

For a system of ***n*** equations:

Classical : polynomial in ***n***

Quantum : logarithmic in ***n***

Run-time

For a system of ***n*** equations:

Classical : polynomial in ***n*** $\tilde{O}(n^3)$

Quantum : logarithmic in ***n*** $O(\kappa s(\log n)/\epsilon)$

Run-time

For a system of ***n*** equations:

Classical : polynomial in ***n*** $\tilde{O}(n^3)$

Quantum : logarithmic in ***n*** $O(\kappa s (\log n)/\epsilon)$

Condition number

Run-time

For a system of ***n*** equations:

Classical : polynomial in ***n*** $\tilde{O}(n^3)$

Quantum : logarithmic in ***n*** $O(\kappa s (\log n)/\epsilon)$

Condition number

Sparsity

Run-time

For a system of ***n*** equations:

Classical : polynomial in ***n*** $\tilde{O}(n^3)$

Quantum : logarithmic in ***n*** $O(\kappa s(\log n)/\epsilon)$

Condition number

Sparsity

Accuracy

Summary

- We saw some of the most frequently used quantum algorithms and sub-routines
    - Grover’s Search
    - Phase Estimation
    - Factoring
    - Hamiltonian Simulation
    - Matrix Inversion
- There are a lot more quantum algorithms, this is just a taster!
- Finding real-world applications is an ongoing challenge