CPEN 400Q / EECE 571Q Lecture 08
Grover search, and quantum resource estimation

Tuesday 1 February 2022

Announcements

- Assignment 2 due Monday 14 Feb 23:59
- Project details coming next week
- Return to in-person classes next week
  - SCRF 209
  - Lectures will be recorded (screen/audio) and posted ASAP
  - TA office hours + tutorial sessions will remain online
  - My office hours are online until my office (KAIS 3043) has chairs

Announcements

UBC QUANTUM CLUB

Upcoming UBC Quantum Club Event

Workshop: Fundamentals of Quantum Computing

Feb 5th, Saturday 2:00-5:00 pm, Online - Zoom

Join the workshop to get hands-on experience on coding a real quantum computer. Topics covered will include: **Qubits, Quantum Gates, Quantum Circuits, Entanglement and Quantum Teleportation Algorithm**

This will build foundations for future workshops on Quantum Machine Learning and Mentorship projects.

How to Sign Up

If you scan the QR or follow the link to our beacons page, you will find the the forms that allow you to register to any of our latest events!

https://beacons.page/ubcquantum

Announcements

QHack 2022: quantum machine learning hackathon with PennyLane coding challenges (and serious prizes!)

Visit `https://qhack.ai`

Learning outcomes

- List and describe the various layers of resource estimation of quantum algorithms
- List and describe the stages of the quantum compilation process

First: quiz 3 solution, and revisit Grover's algorithm.

Quiz 3

Quiz 3

Normally, we've been implementing oracles using an auxiliary qubit and phase kickback:

$$O|\mathbf{x}\rangle|-\rangle = (-1)^{f(\mathbf{x})}|\mathbf{x}\rangle|-\rangle$$

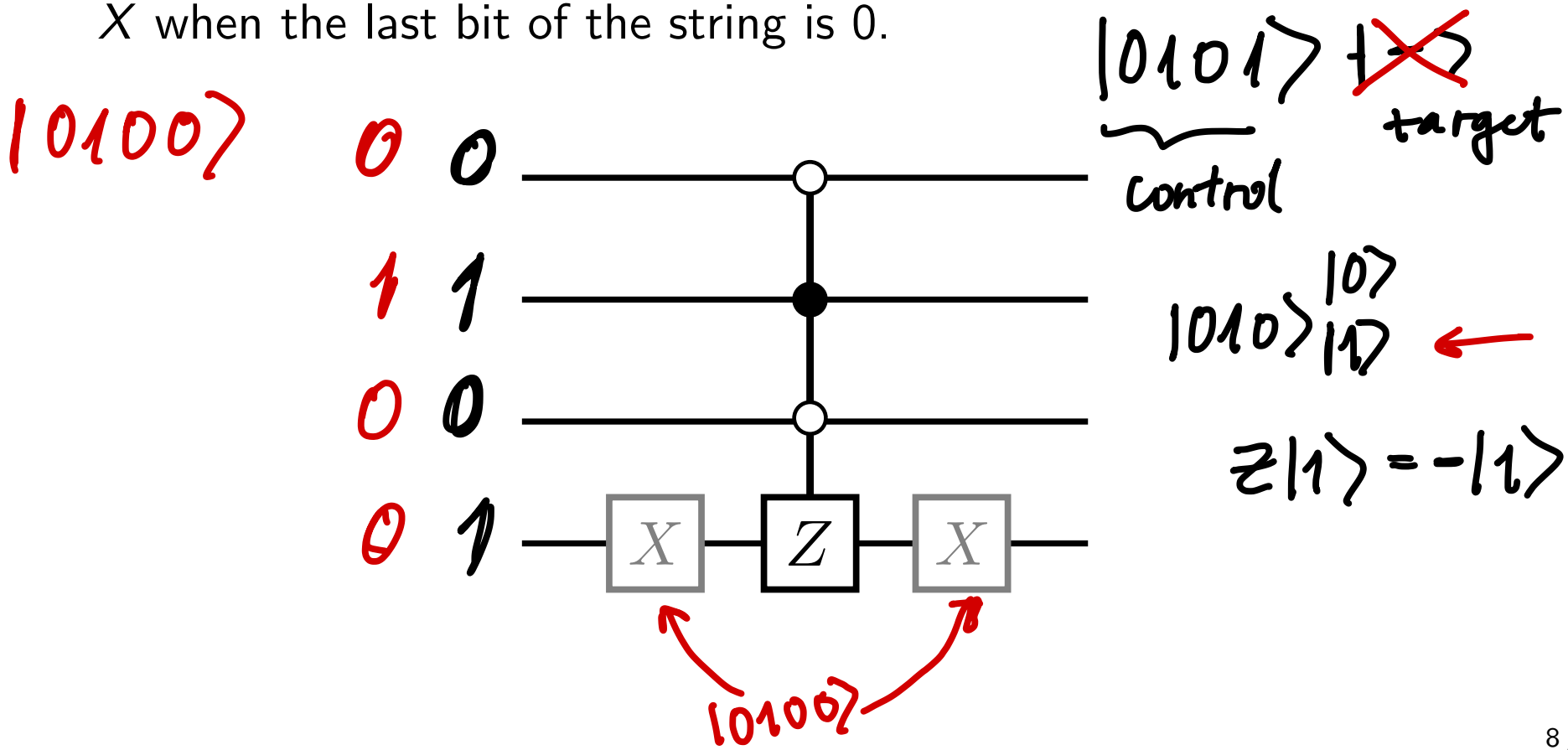
How to implement an oracle in PennyLane without the auxiliary qubit?

$$O|\mathbf{x}\rangle = (-1)^{f(\mathbf{x})}|\mathbf{x}\rangle$$

Quiz 3

Z adds a phase; CZ adds a phase only to state $|11\rangle$; this generalizes to CCZ on state $|111\rangle$, etc.

Need to apply multi-controlled Z to flip sign of correct basis state, but can't control on all qubits at the same time; use a "corrective" X when the last bit of the string is 0.



Quiz 3

My solution:

```
def oracle(special_string=None, wires=None):
    if special_string[-1] == "0":
        qml.PauliX(wires=wires[-1])

    qml.ControlledQubitUnitary(
        qml.PauliZ.matrix,
        control_wires=wires[:-1],
        wires=wires[-1],
        control_values=special_string[:-1],
    )

    if special_string[-1] == "0":
        qml.PauliX(wires=wires[-1])
```

Quiz 3

Alternative solutions:

```
def oracle(special_string=None, wires=None):
    special_idx = int('0b' + special_string, 2)

    oracle_matrix = np.eye(2 ** len(wires))
    oracle_matrix[special_idx, special_idx] = -1

    qml.QubitUnitary(oracle_matrix, wires=wires)
```

```
def oracle(special_string=None, wires=None):
    special_idx = int('0b' + special_string, 2)

    diagonal = np.ones(2 ** len(wires))
    diagonal[special_idx] = -1

    qml.DiagonalQubitUnitary(diagonal, wires=wires)
```

Grover's algorithm revisited

Grover's quantum search algorithm

Grover is **search algorithm** that can find an n-bit special element $|\mathbf{s}\rangle$ using $O(\sqrt{2^n})$ queries of an oracle.

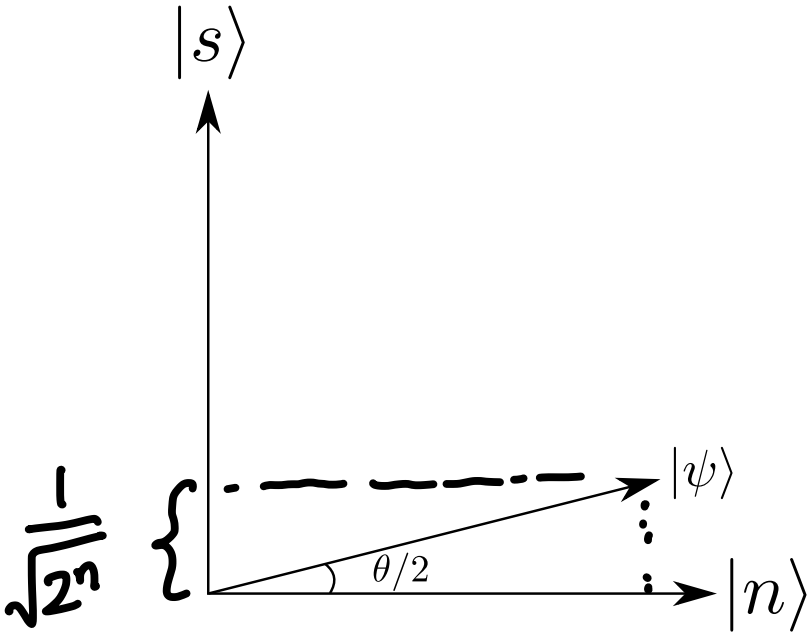
We start from the uniform superposition:

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x}\in\{0,1\}^n} |\mathbf{x}\rangle$$

and use amplitude amplification to obtain

$$|\psi'\rangle = (\text{big number})|\mathbf{s}\rangle + (\text{small number}) \sum_{\mathbf{x}\neq\mathbf{s}} |\mathbf{x}\rangle$$

Grover's algorithm: geometric visualization



We started by writing the uniform superposition as a linear combination of the special element, and things that are not the special element.

$$|\psi\rangle = \sin\left(\frac{\theta}{2}\right)|s\rangle + \cos\left(\frac{\theta}{2}\right)|n\rangle$$

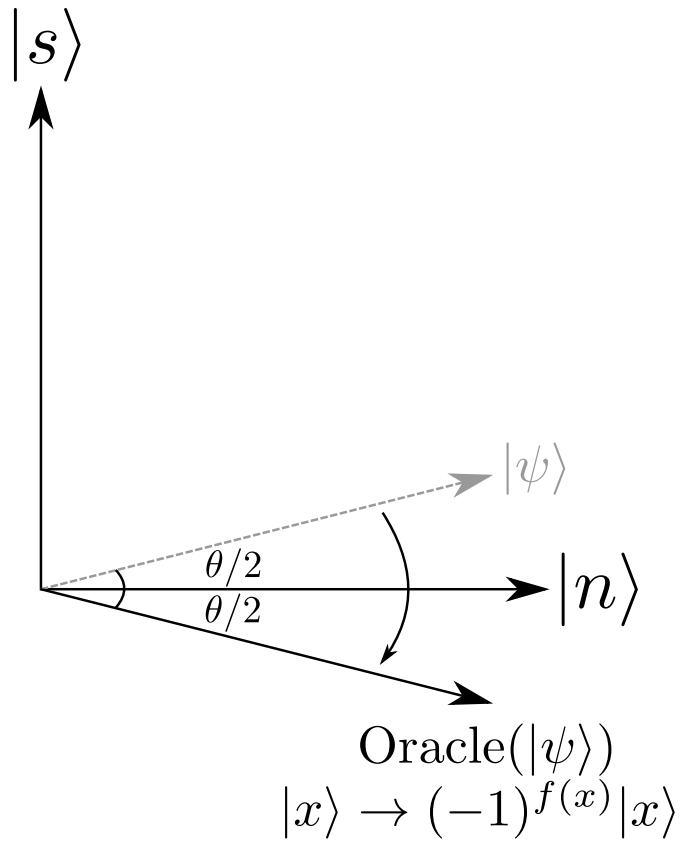
We can represent this geometrically in 2D space.

$$|\psi\rangle = \frac{1}{\sqrt{2^n}}\left(|00\cdots0\rangle + |00\cdots01\rangle + \cdots + |111\cdots1\rangle\right)$$

$$\langle s|\psi\rangle = \frac{1}{\sqrt{2^n}} \quad \Rightarrow \quad \sin\frac{\theta}{2} = \frac{1}{\sqrt{2^n}}$$

Grover's algorithm: geometric visualization

Two special operations: the oracle, and the diffusion operator.



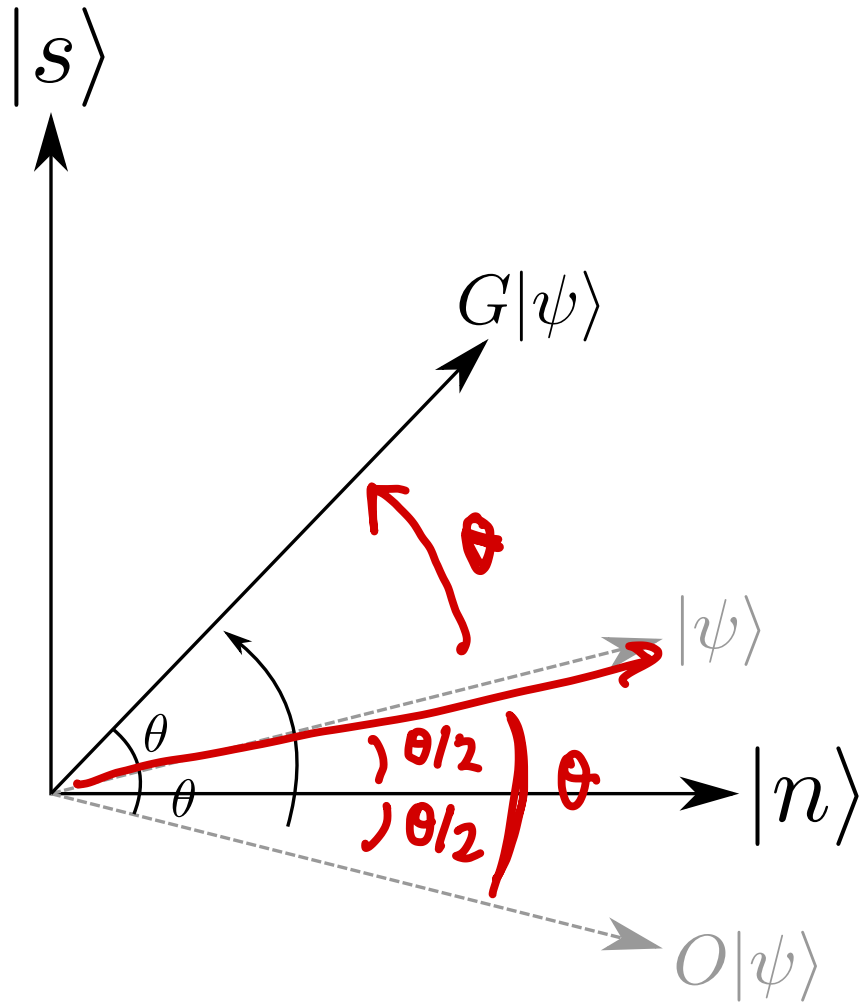
The oracle sends

$$|\mathbf{s}\rangle \rightarrow -|\mathbf{s}\rangle$$
$$|\mathbf{x}\rangle \rightarrow |\mathbf{x}\rangle, \mathbf{x} \neq \mathbf{s}$$

It flips the phase of the special state only. This looks like a reflection in the 2-dimensional space.

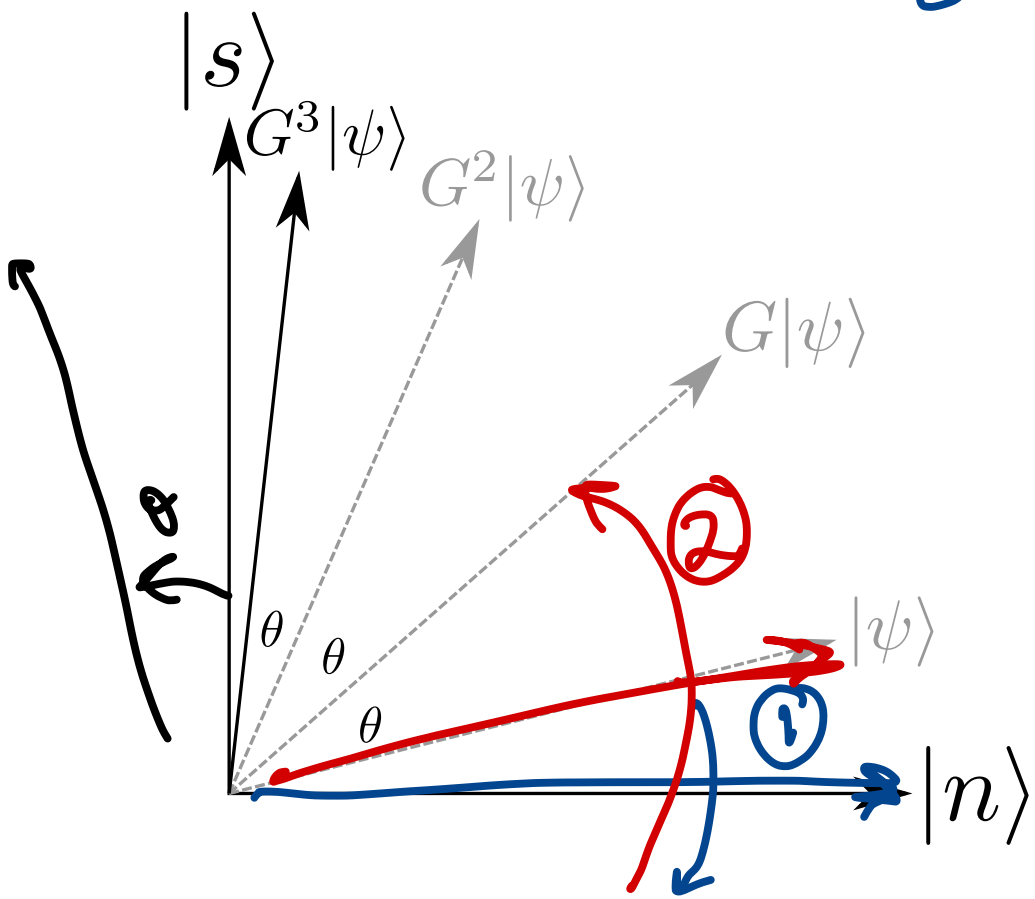
The diffusion operator

The diffusion operator spreads amplitude. It reflects us about the uniform superposition state, i.e., keep uniform superposition constant and change the sign on stuff orthogonal to it.



Finding the optimal number of iterations

Applying oracle then diffusion effects a rotation in this 2-dimensional space. We iterate this procedure many times to get closer to the special state.



Finding the optimal number of iterations

We start at

$$|\psi\rangle = \sin\left(\frac{\theta}{2}\right)|s\rangle + \cos\left(\frac{\theta}{2}\right)|n\rangle$$

After one rotation,

$$|\psi\rangle = \sin\left(\frac{3\theta}{2}\right)|s\rangle + \cos\left(\frac{3\theta}{2}\right)|n\rangle$$

$$\frac{\theta}{2} = \frac{1}{\sqrt{2^n}}$$

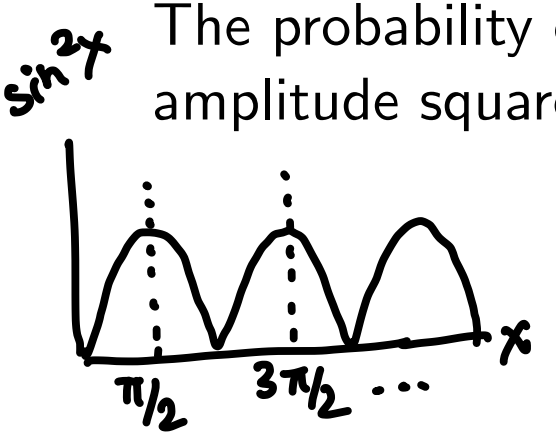
After k rotations,

$$|\psi\rangle = \sin\left(\frac{(2k+1)\theta}{2}\right)|s\rangle + \cos\left(\frac{(2k+1)\theta}{2}\right)|n\rangle$$

What value of k maximizes the probability of observing the special state when we take a measurement?

Finding the optimal number of iterations

The probability of measuring and observing the special state is the amplitude squared:



$$\Pr(s) = \sin^2\left(\frac{(2k+1)\theta}{2}\right)$$

Note that $\theta/2$ is the overlap between uniform superposition and $|\mathbf{s}\rangle$:

$$\sin\frac{\theta}{2} = \frac{1}{\sqrt{2^n}} \Rightarrow \frac{\theta}{2} = \arcsin\frac{1}{\sqrt{2^n}}$$

If $N = 2^n$ is large, can make an approximation:

$$\frac{\theta}{2} = \arcsin\frac{1}{\sqrt{2^n}} \rightarrow \frac{\theta}{2} \approx \frac{1}{\sqrt{2^n}}$$

Finding the optimal number of iterations

Classical: $O(2^n)$

Quantum: $O(\sqrt{2^n})$

$$\Pr(\mathbf{s}) = \sin^2\left(\frac{(2k+1)\theta}{2}\right) = \sin^2\left(\frac{(2k+1)}{\sqrt{2^n}}\right)$$

This is maximized when the argument of the sin is $\pi/2$:

$$\frac{(2k+1)}{\sqrt{2^n}} = \frac{\pi}{2}$$

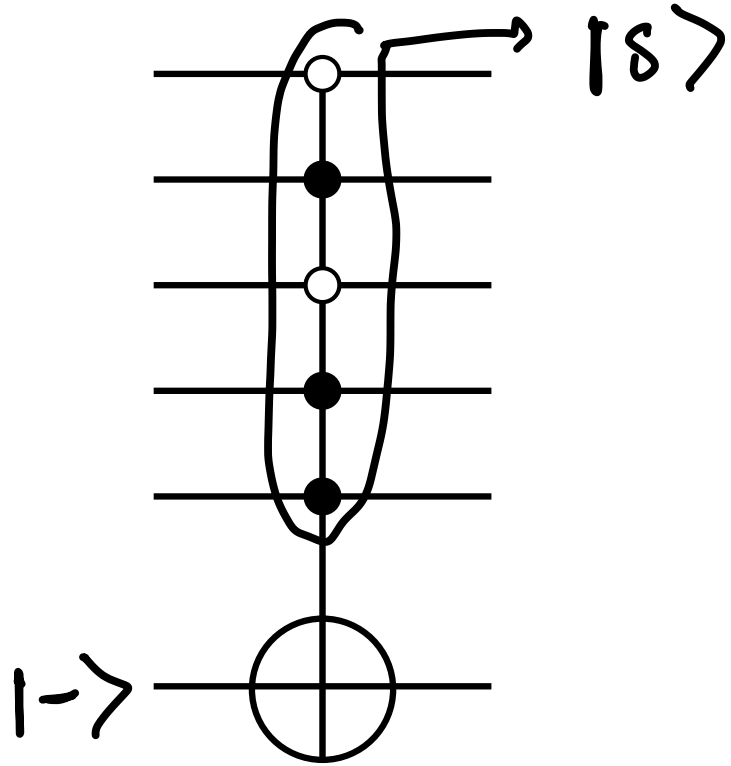
$$(2k+1) = \frac{\pi}{2}\sqrt{2^n}$$

$$2k = \frac{\pi}{2}\sqrt{2^n} - 1$$

$$k = \frac{\pi}{4}\sqrt{2^n} - \frac{1}{2}$$

The oracle

We can implement the oracle using phase kickback and a multi-controlled NOT, controlled on the value of the special state.



The diffusion operator

The uniform superposition is $|++\cdots+\rangle$.

$H|0\rangle = |+\rangle$
$H|1\rangle = |-\rangle$

This is a state in the Hadamard basis ($\{|+\rangle, |-\rangle\}$).

$|+--+\cdots\rangle$

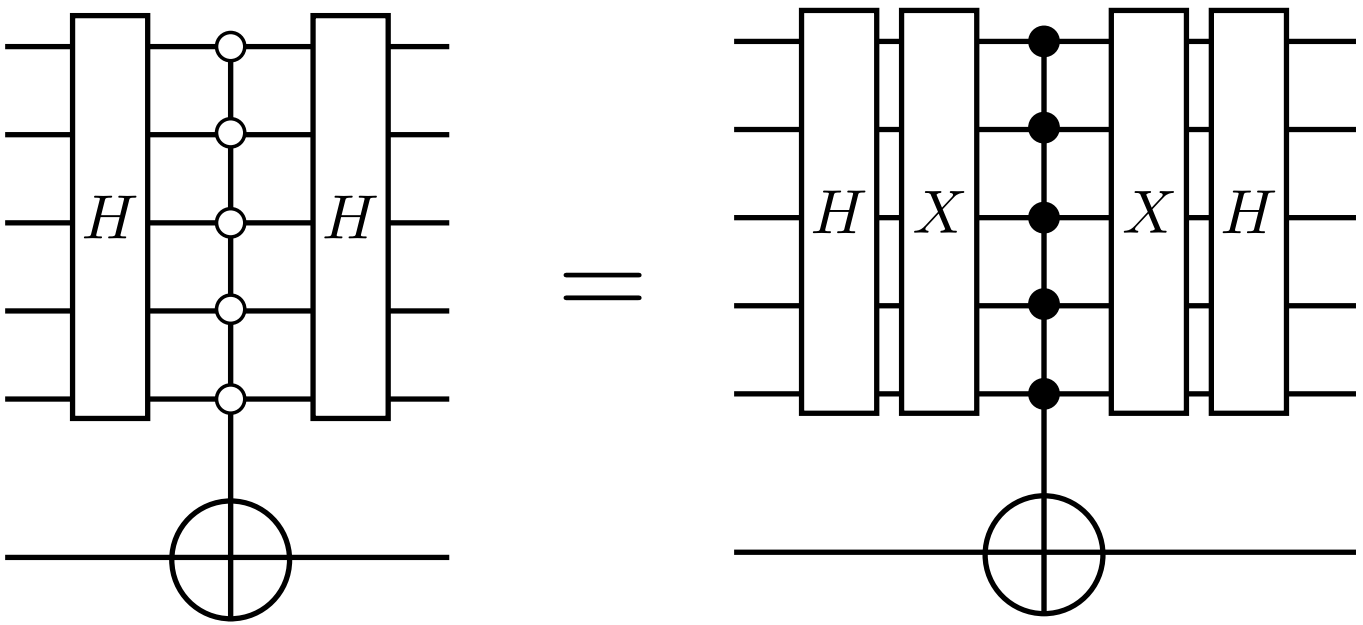
All other Hadamard basis states are orthogonal to it.

The diffusion operator needs to send:

- $|++\cdots+\rangle \to |++\cdots+\rangle$
- $|\mathbf{h}\rangle \to -|\mathbf{h}\rangle$ for $\mathbf{h} \in \{+,-\}^n, \quad \mathbf{h} \neq +^n$

The diffusion operator

This circuit is a bit mysterious, but it does the job *up to a global phase of -1*:

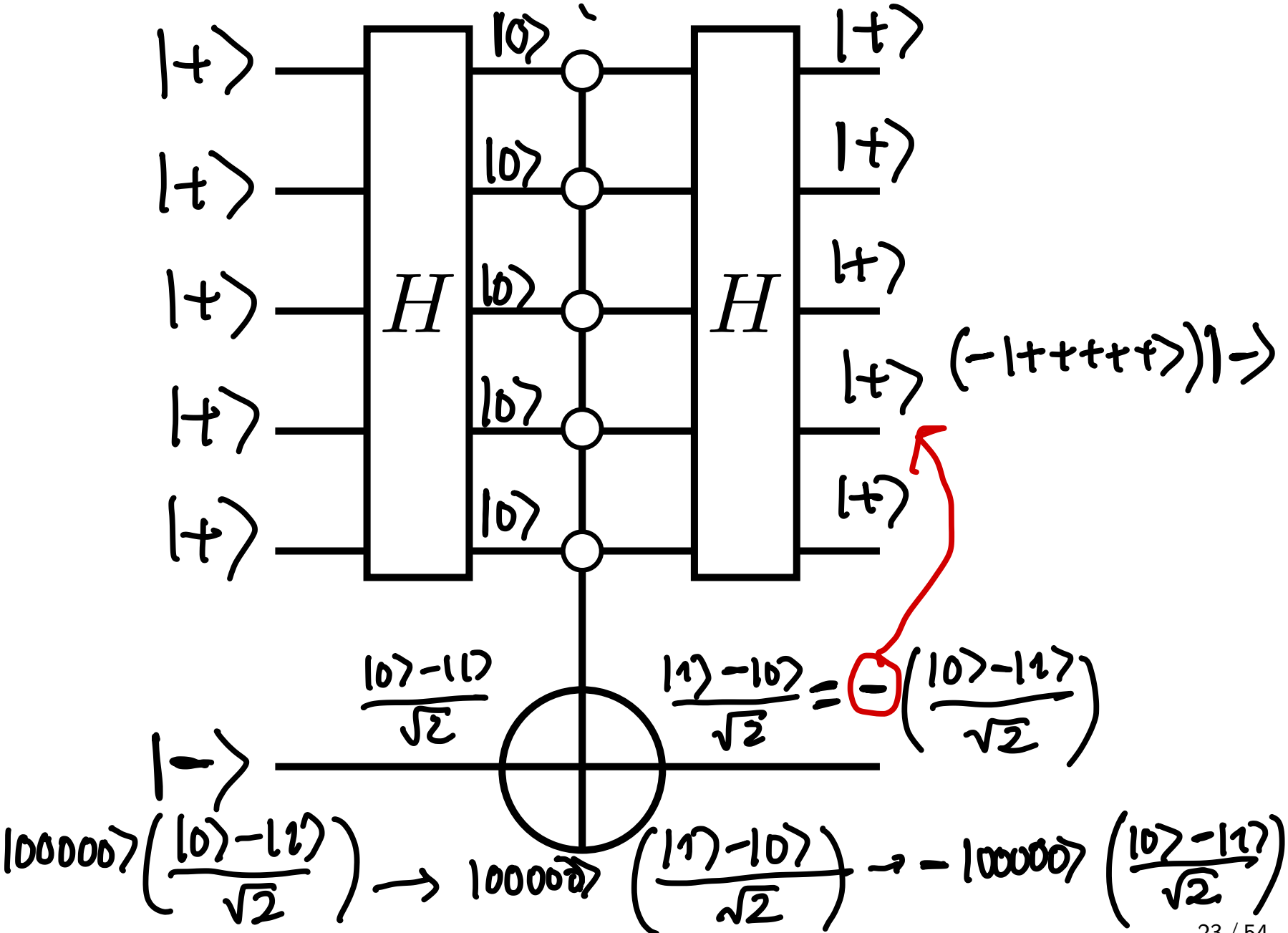


$|++\cdots+\rangle \to -|+\cdots+\rangle$

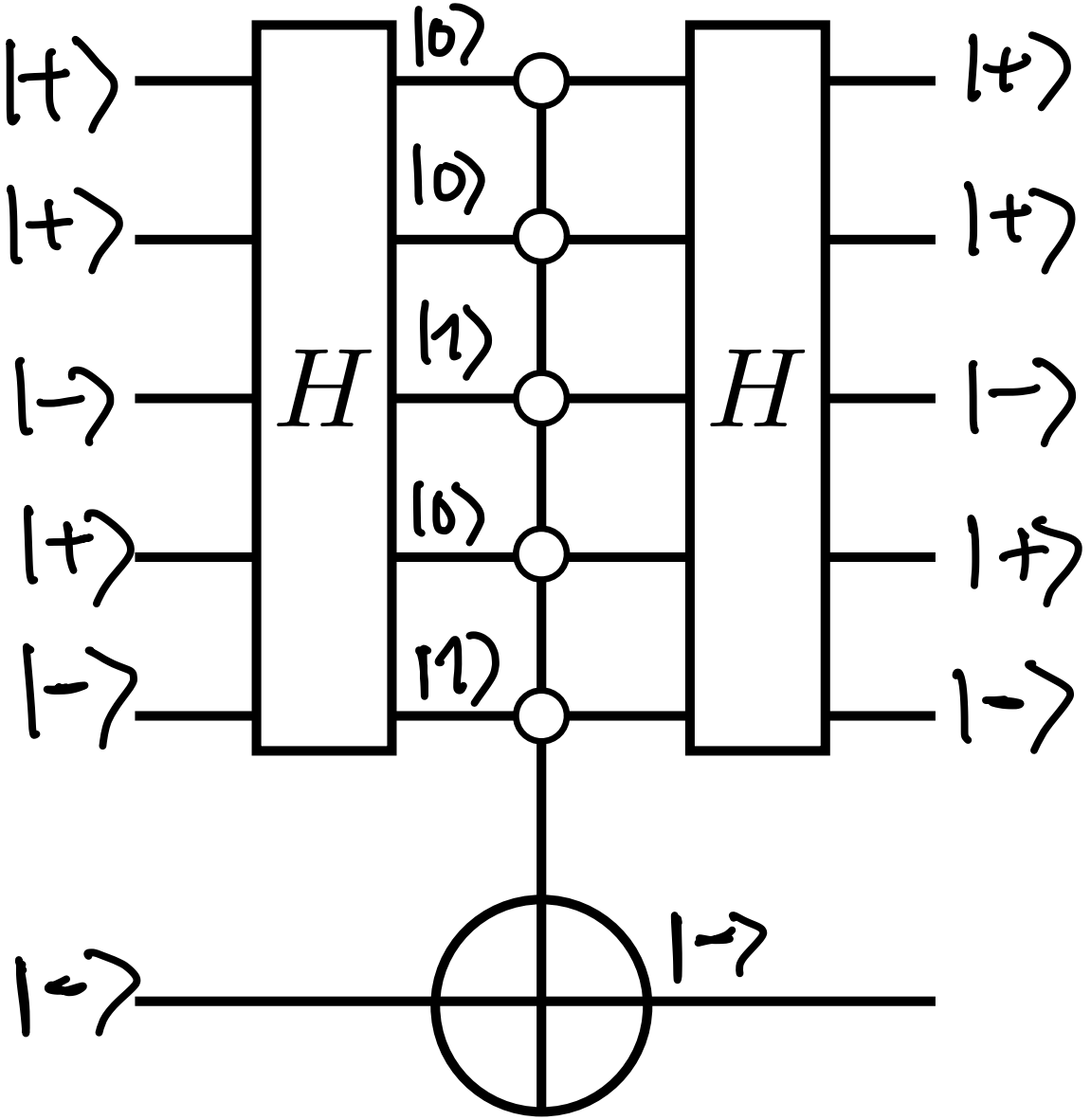
$|else\rangle \to |else\rangle$

Let's test it on some Hadamard basis states.

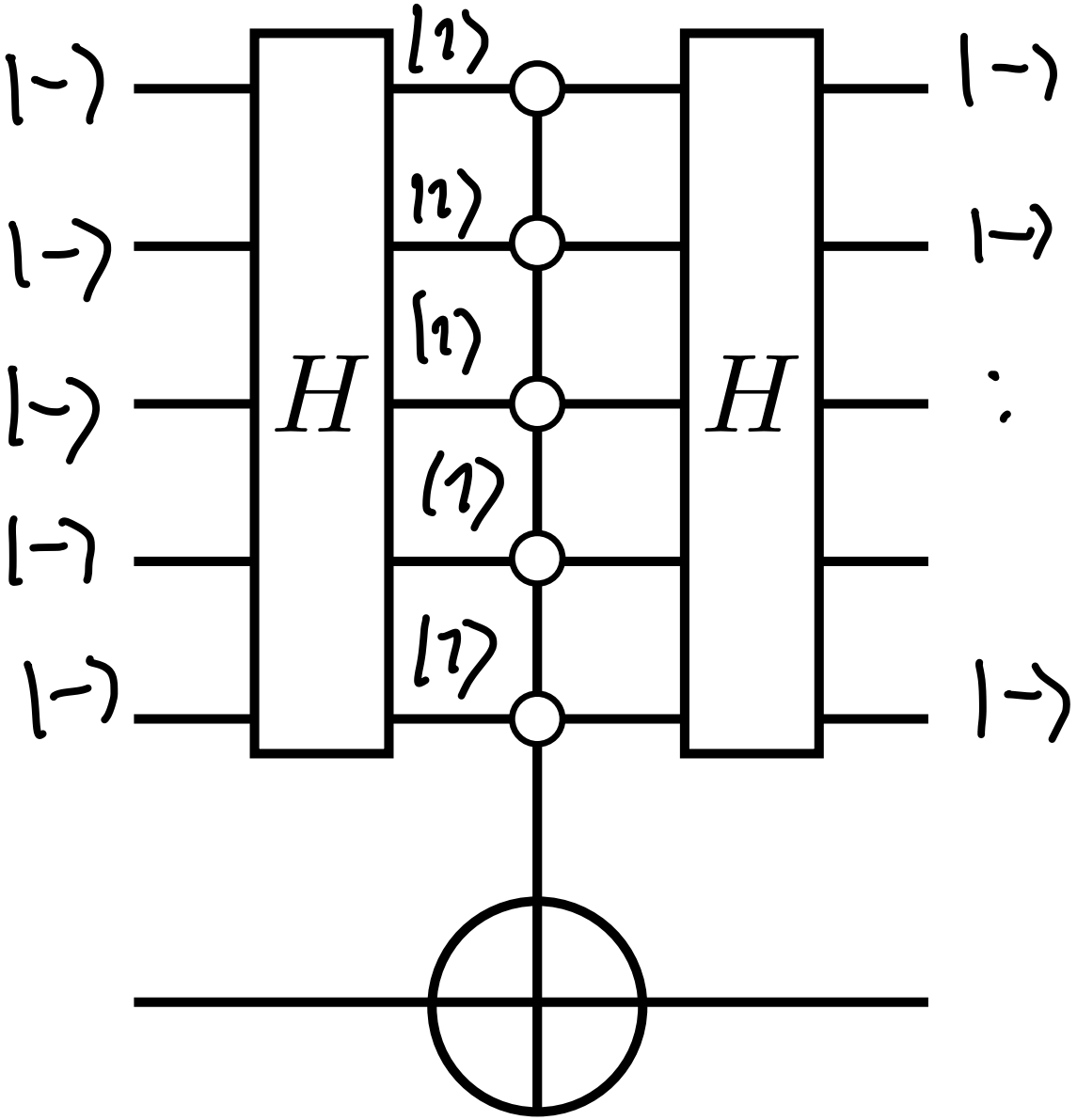
The diffusion operator



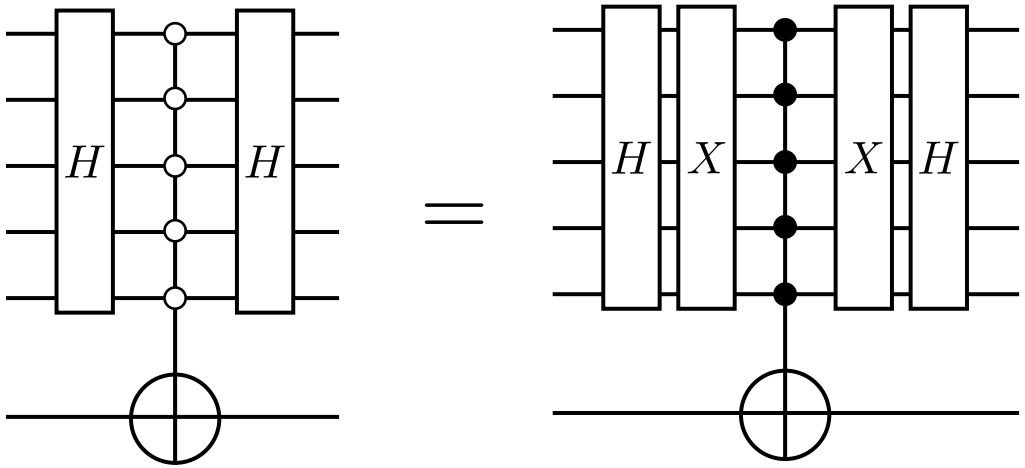
The diffusion operator



The diffusion operator



The diffusion operator



This sends

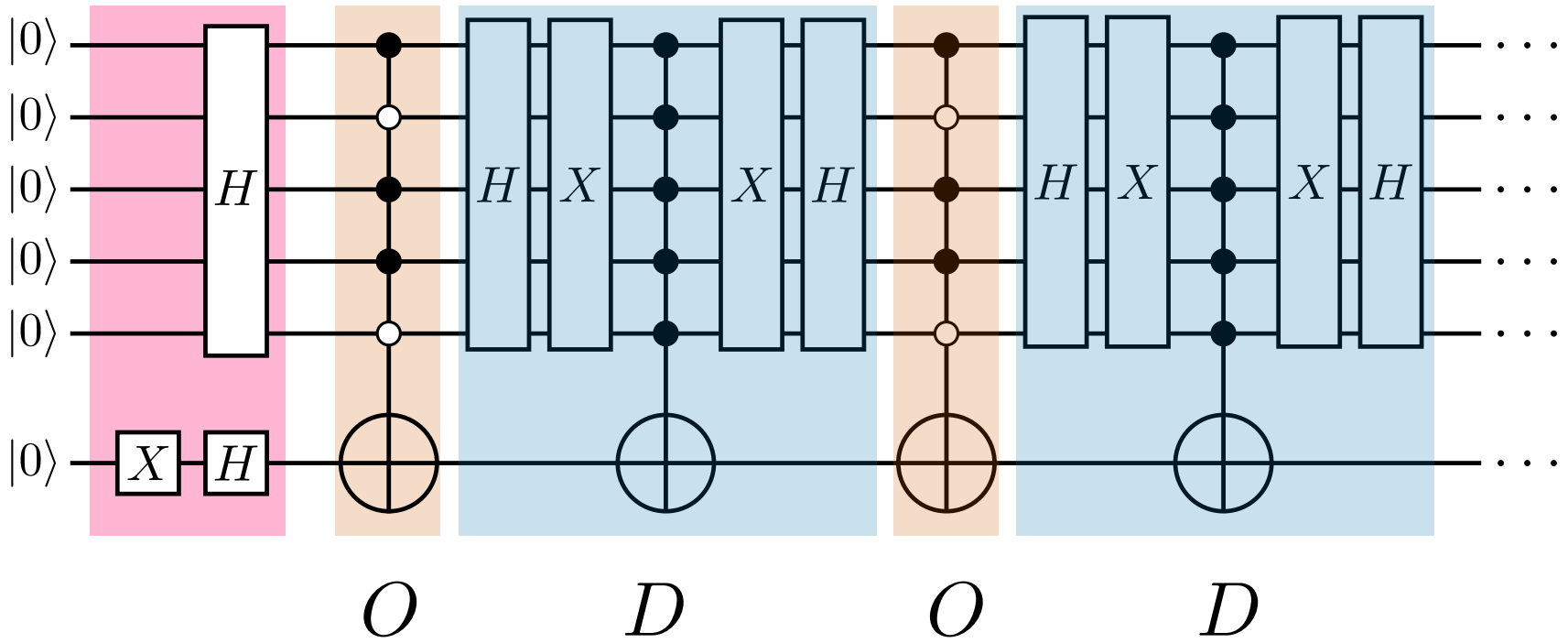
- $|++\cdots+\rangle \to -|++\cdots+\rangle$
- $|\mathbf{h}\rangle \to |\mathbf{h}\rangle$ for $\mathbf{h} \in \{+,-\}^n, \quad \mathbf{h} \neq +^n$

But this is just a global phase, so it is equivalent to

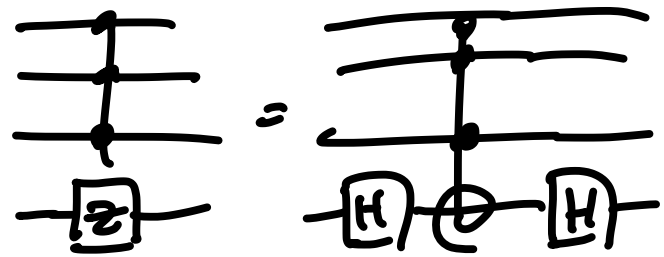
- $|++\cdots+\rangle \to |++\cdots+\rangle$
- $|\mathbf{h}\rangle \to -|\mathbf{h}\rangle$ for $\mathbf{h} \in \{+,-\}^n, \quad \mathbf{h} \neq +^n$

Grover's algorithm

Let's go back to the implementation.



Last time



Two interesting questions from last time:

1. "Although we need less oracle queries per experiment, but we have to repeat this procedure several times to extract the probability distributions through measurements. How does this affect the query complexity?"
2. How long does a Grover iteration take?

Question 1

> *"Although we need less oracle queries per experiment, but we have to repeat this procedure several times to extract the probability distributions through measurements. How does this affect the query complexity?"*

Answer: it does not affect the *query complexity*.

Each time, the success probability is $1 - O(1/\sqrt{2^n})$ (very high if 2^n is large) so can consider how many times you need to run to reach a desired success probability.

Question 2

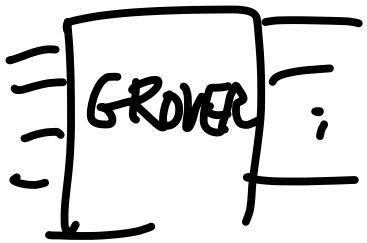
How long does a Grover iteration take?

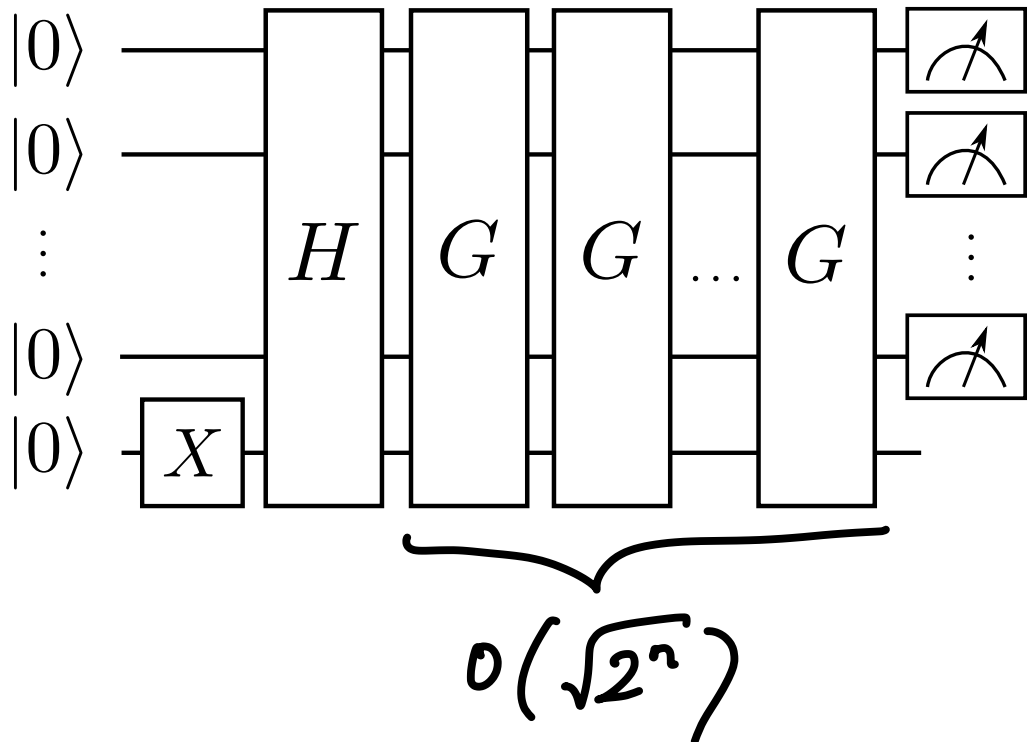
We will answer this today in the context of *quantum resource estimation*.

Quantum resource estimation

Resource estimation

We computed the *query complexity* of Grover search.

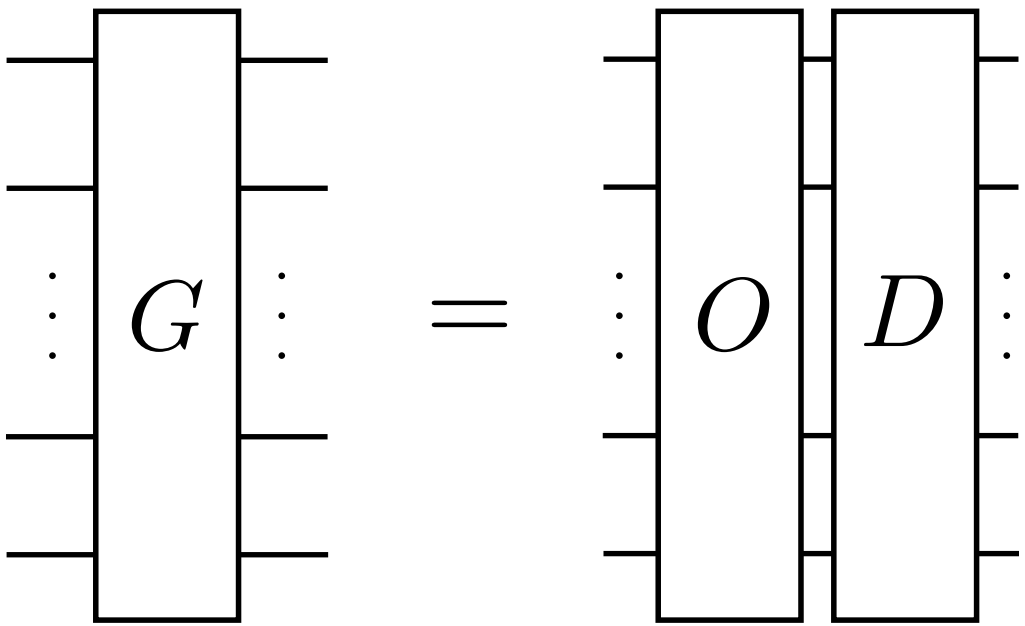




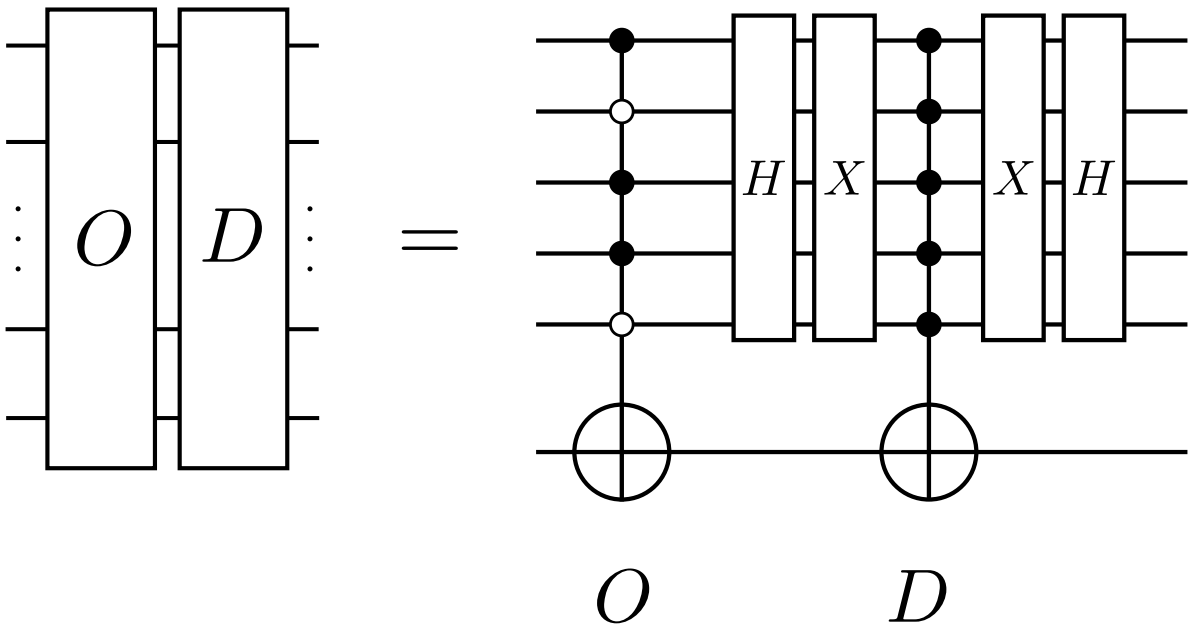
However, this only gives us the resources at a very high level.

Resource estimation

We know that:



Furthermore,



Resource estimation

There's still more!

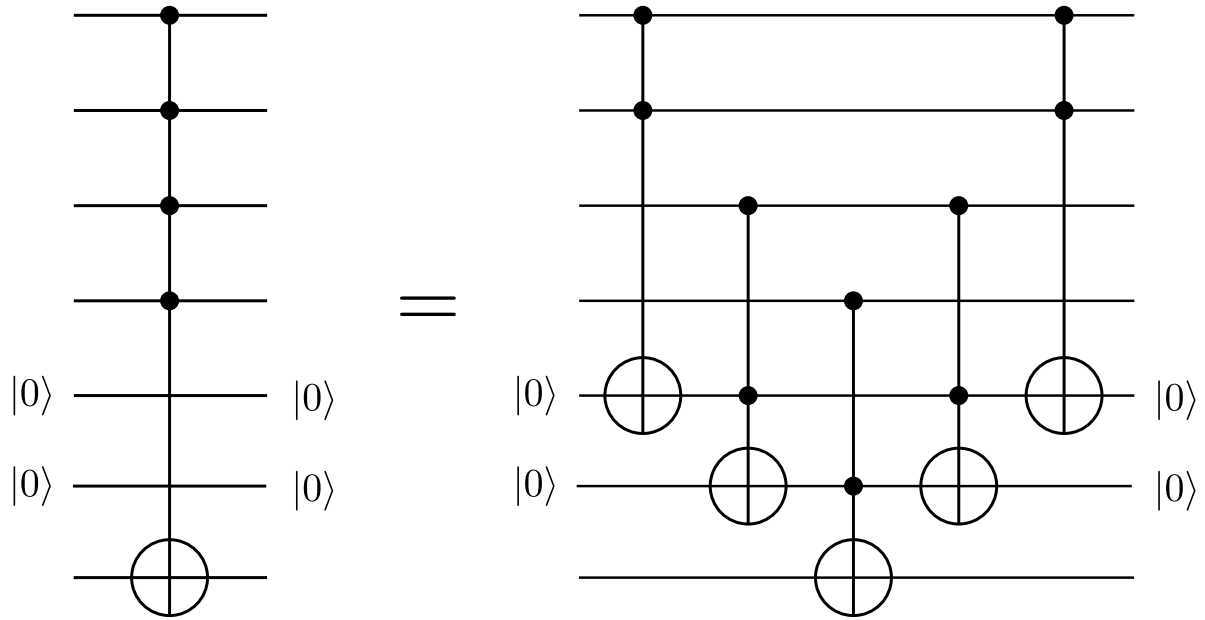


Image credit: Codebook node I.13

Resource estimation

It ends eventually:

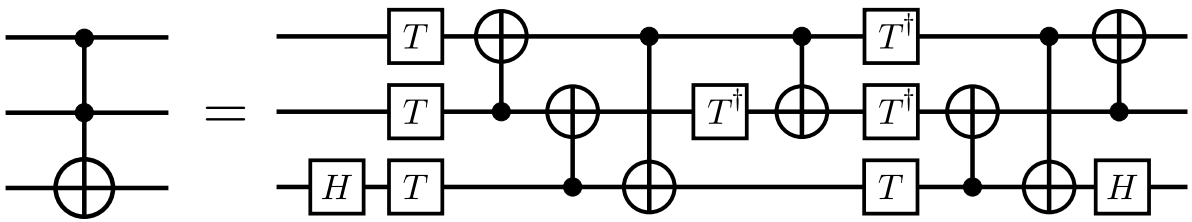


Image credit: Codebook node I.13

Resource estimation

To answer the question, "how long does it take", we must specify:

- what **level of abstraction** we are asking the question with respect to
- what our **metrics** are
- what **other types of resources** we have available (e.g., auxiliary qubits).

Resource estimation metrics

Often:

- Queries (high-level)
- Number of qubits / auxiliary qubits
- Number of gates
- Circuit depth

But we can also go deeper:

- Gates/depth after compilation to specific hardware
- Gates/depth after embedding into error-correcting code (fault-tolerant resource estimation)
- Actual run time

Circuit depth

The circuit depth is an important metric: informally it is the number of time steps required to run the circuit.

(Formally: represent the circuit as a directed acyclic graph, and compute the longest path).

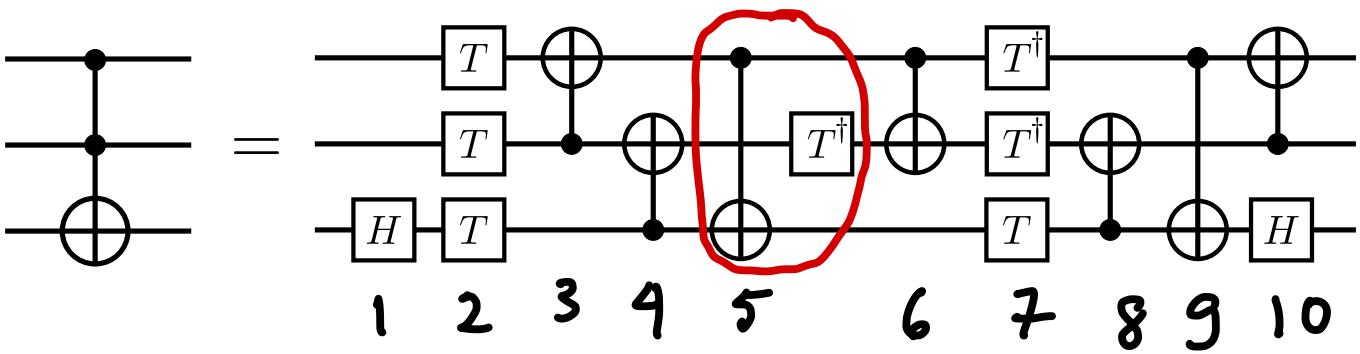


Image credit: Codebook node I.13

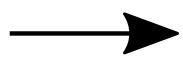
Quantum compilation

Compilation

Compilers are often taken for granted.

```c
#include <stdio.h>

void main() {
    printf("Hello, UBC!");
}
```



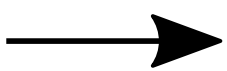
```
        .file   "comp.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello, UBC!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        nop
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

We don't want to do this by hand!

Quantum compilation

Quantum computers also need compilers!

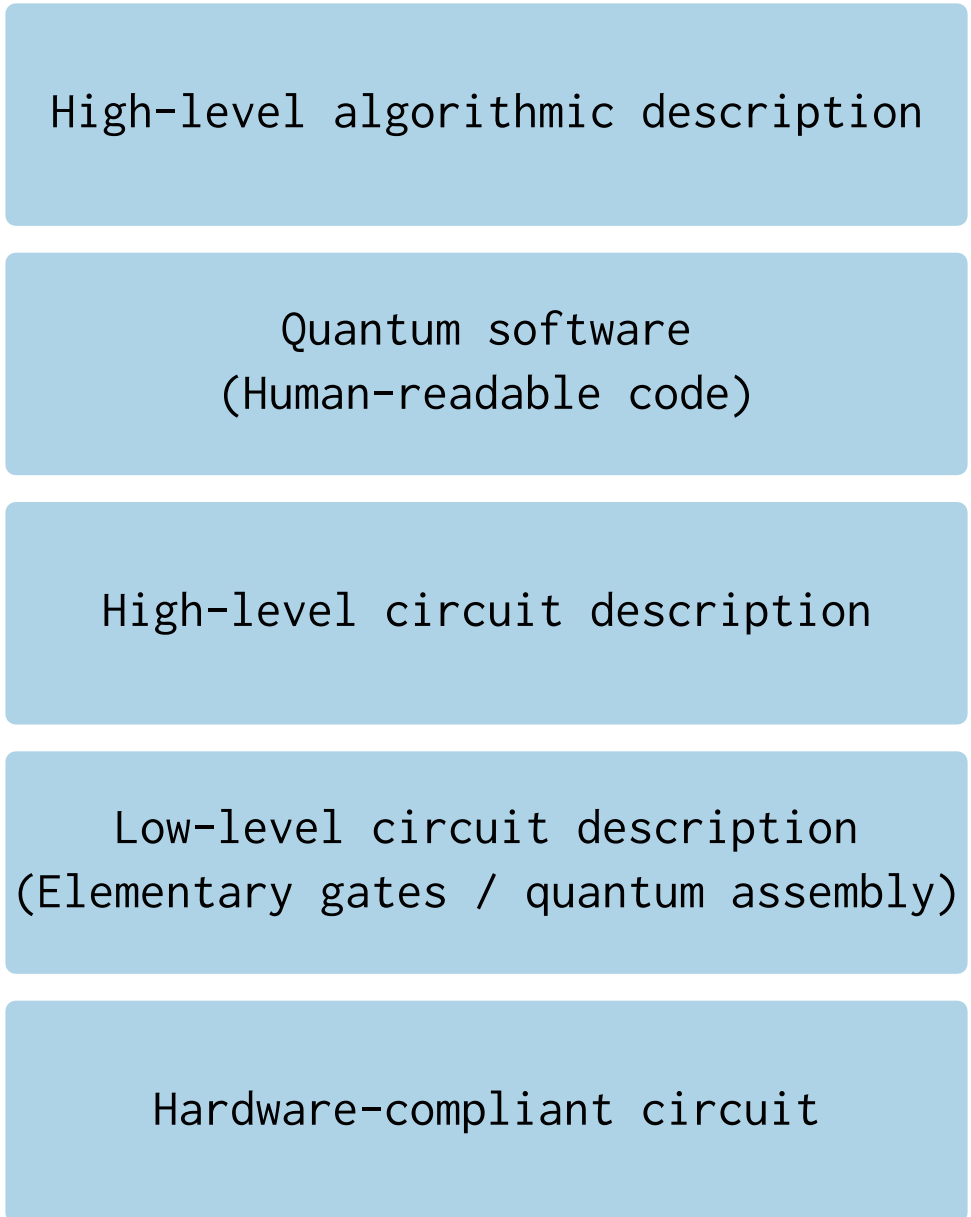
```
QFT(num_qubits=4)
```



```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[4];
h q[0];
cp(pi/2) q[0],q[1];
cp(pi/4) q[0],q[2];
cp(pi/8) q[0],q[3];
h q[1];
cp(pi/2) q[1],q[2];
cp(pi/4) q[1],q[3];
h q[2];
cp(pi/2) q[2],q[3];
h q[3];
swap q[0],q[3];
swap q[1],q[2];
```

Translate quantum algorithms into *elementary quantum gates*.

How to run a quantum algorithm on quantum hardware



How to run a quantum algorithm on quantum hardware

High-level algorithmic description

Add bits a and b with a half-adder

Quantum software
(Human-readable code)

High-level circuit description

Low-level circuit description
(Elementary gates / quantum assembly)

Hardware-compliant circuit

How to run a quantum algorithm on quantum hardware

High-level algorithmic description

Quantum software
(Human-readable code)

```
quantum_half_adder(a, b)
```

High-level circuit description

Low-level circuit description
(Elementary gates / quantum assembly)

Hardware-compliant circuit

How to run a quantum algorithm on quantum hardware

High-level algorithmic description

Quantum software
(Human-readable code)

High-level circuit description

Low-level circuit description
(Elementary gates / quantum assembly)

Hardware-compliant circuit

```
quantum_half_adder(a, b)
```

Available quantum computing software frameworks allow users to work mostly at this level.

How to run a quantum algorithm on quantum hardware

High-level algorithmic description

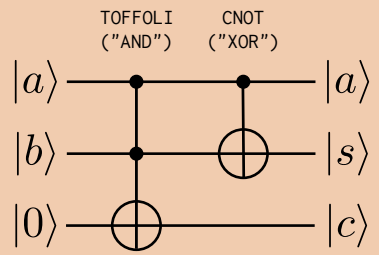
Quantum software
(Human-readable code)

High-level circuit description

Low-level circuit description
(Elementary gates / quantum assembly)

Hardware-compliant circuit

```
def quantum_half_adder(a, b):
    qubit register q[3]
    prepare_bit(a, q[0])
    prepare_bit(b, q[1])
    toffoli(q[0], q[1], q[2])
    cnot(q[0], q[1])
```



How to run a quantum algorithm on quantum hardware

High-level algorithmic description

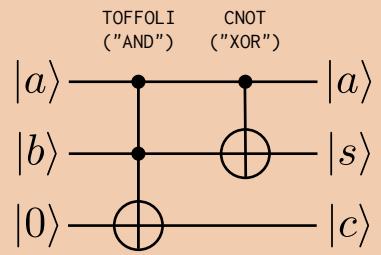
Quantum software
(Human-readable code)

High-level circuit description

Low-level circuit description
(Elementary gates / quantum assembly)

Hardware-compliant circuit

```
def quantum_half_adder(a, b):
    qubit register q[3]
    prepare_bit(a, q[0])
    prepare_bit(b, q[1])
    toffoli(q[0], q[1], q[2])
    cnot(q[0], q[1])
```



If you're developing quantum algorithms, you're probably working at this level.

How to run a quantum algorithm on quantum hardware

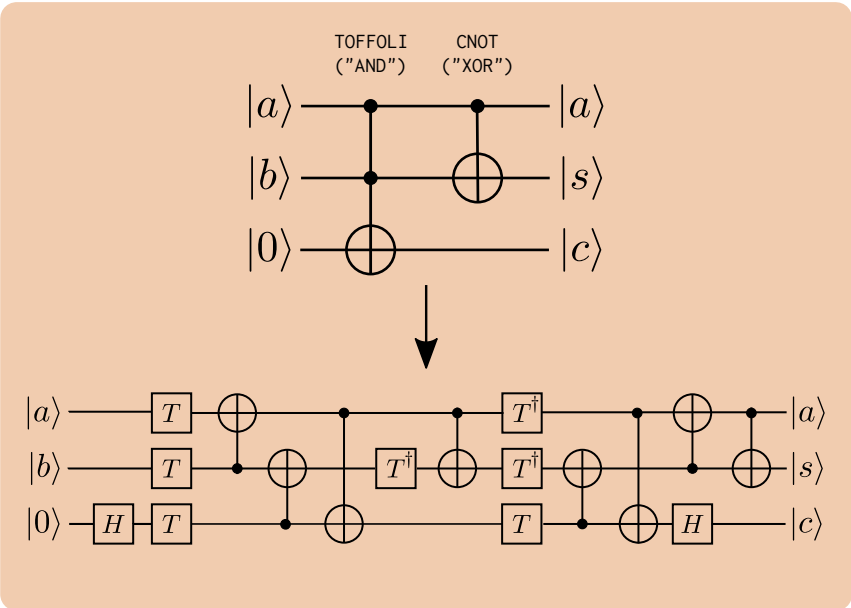
High-level algorithmic description

Quantum software
(Human-readable code)

High-level circuit description

Low-level circuit description
(Elementary gates / quantum assembly)

Hardware-compliant circuit



How to run a quantum algorithm on quantum hardware

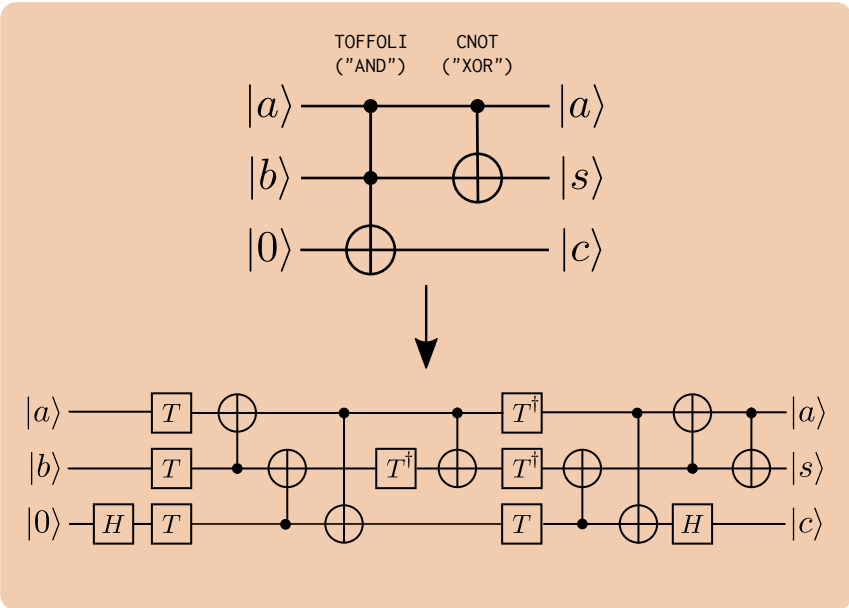
High-level algorithmic description

Quantum software
(Human-readable code)

High-level circuit description

Low-level circuit description
(Elementary gates / quantum assembly)

Hardware-compliant circuit



We should automate this.

How to run a quantum algorithm on quantum hardware

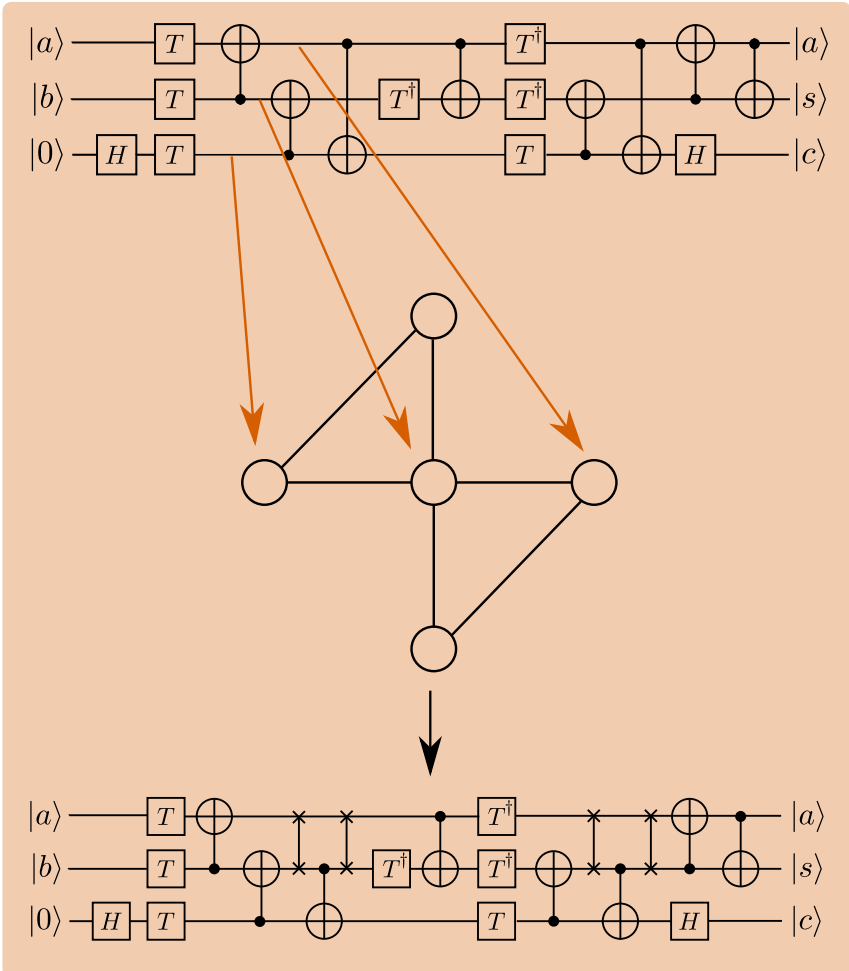
High-level algorithmic description

Quantum software
(Human-readable code)

High-level circuit description

Low-level circuit description
(Elementary gates / quantum assembly)

Hardware-compliant circuit



How to run a quantum algorithm on quantum hardware

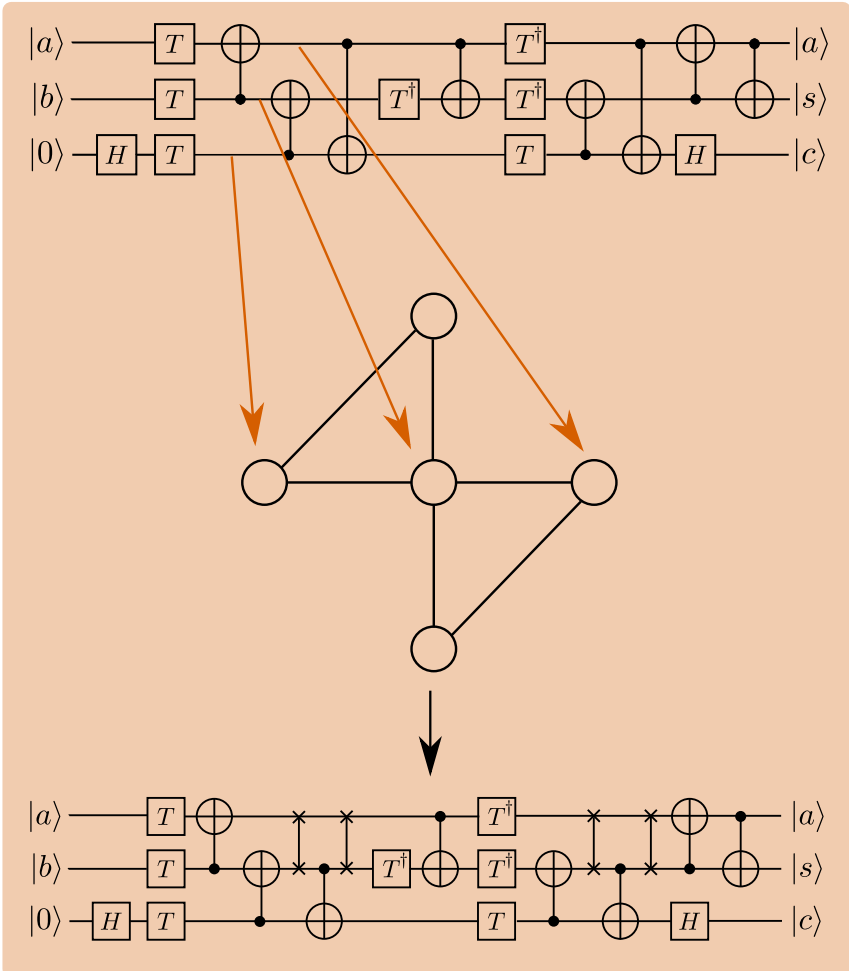
High-level algorithmic description

Quantum software
(Human-readable code)

High-level circuit description

Low-level circuit description
(Elementary gates / quantum assembly)

Hardware-compliant circuit



We should **really** automate this.

Quantum compilation

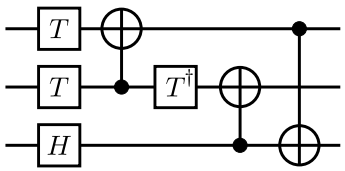
High-level algorithmic description

Quantum software
(Human-readable code)

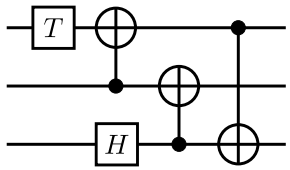
High-level circuit description

Low-level circuit description
(Elementary gates / quantum assembly)

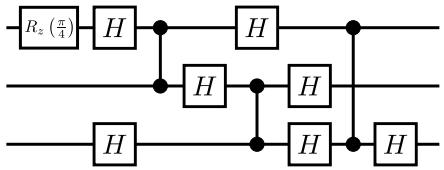
Hardware-compliant circuit



Synthesis

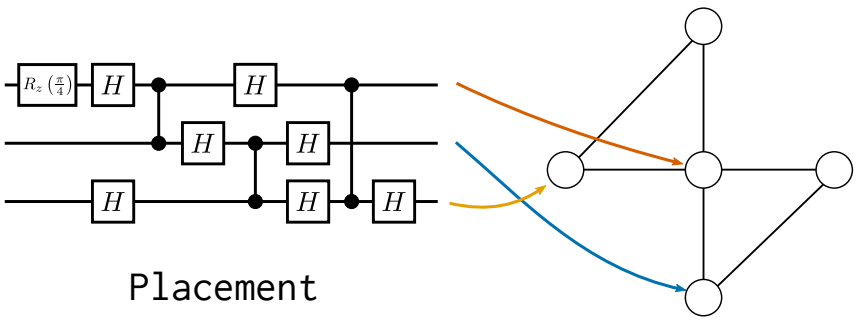


Optimization

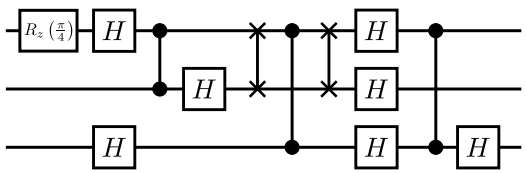


Transpilation

Verification



Placement



Routing

Next time

Content:

- Compilation with PennyLane: quantum transforms

Action items:

1. Continue with Assignment 2 (can do the first 3 problems now)

Recommended reading:

- Codebook nodes G.1-G.5, I.13
- Nielsen & Chuang 6.1