Lecture

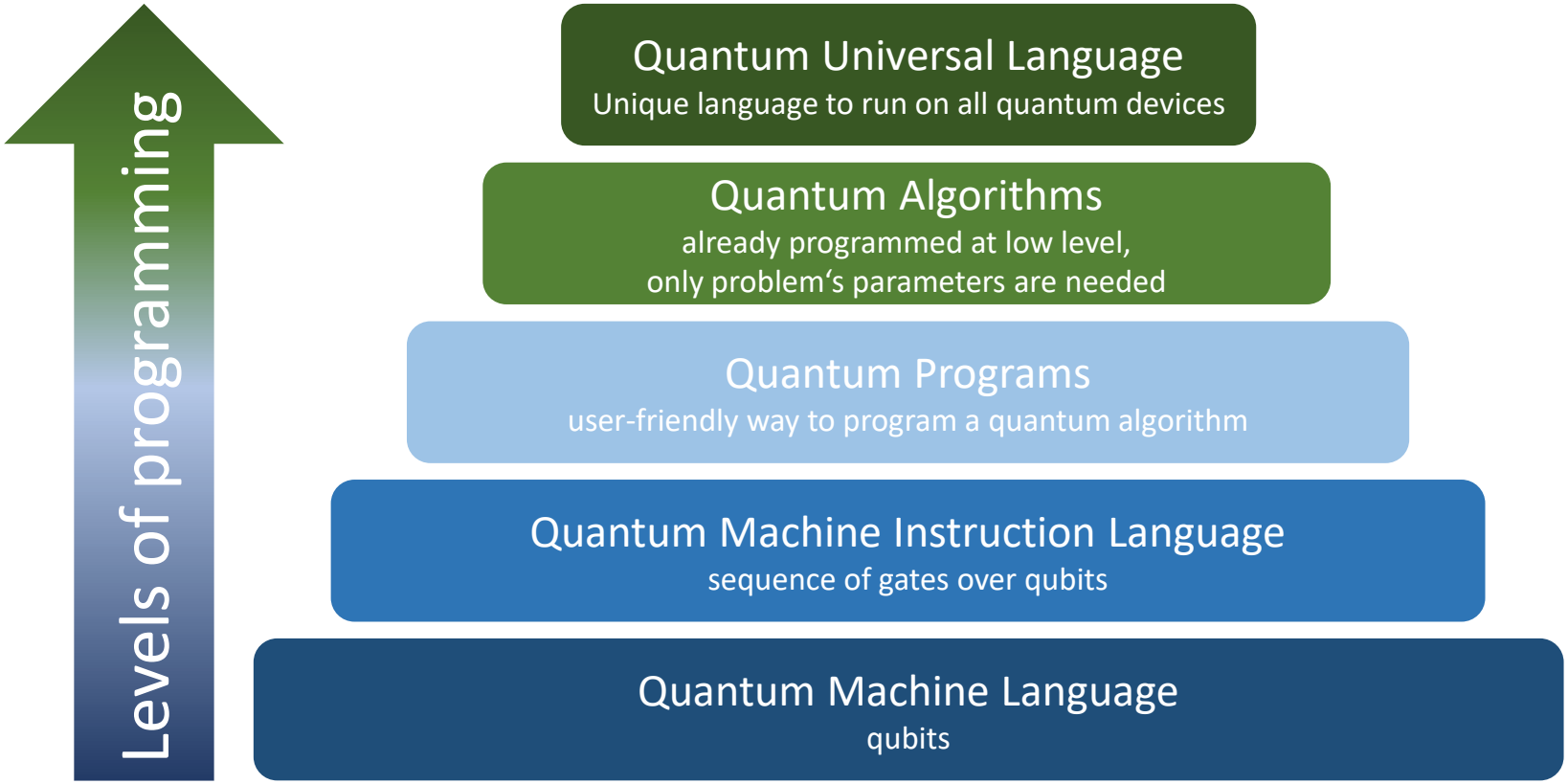
# Quantum Computing

(CS5070)

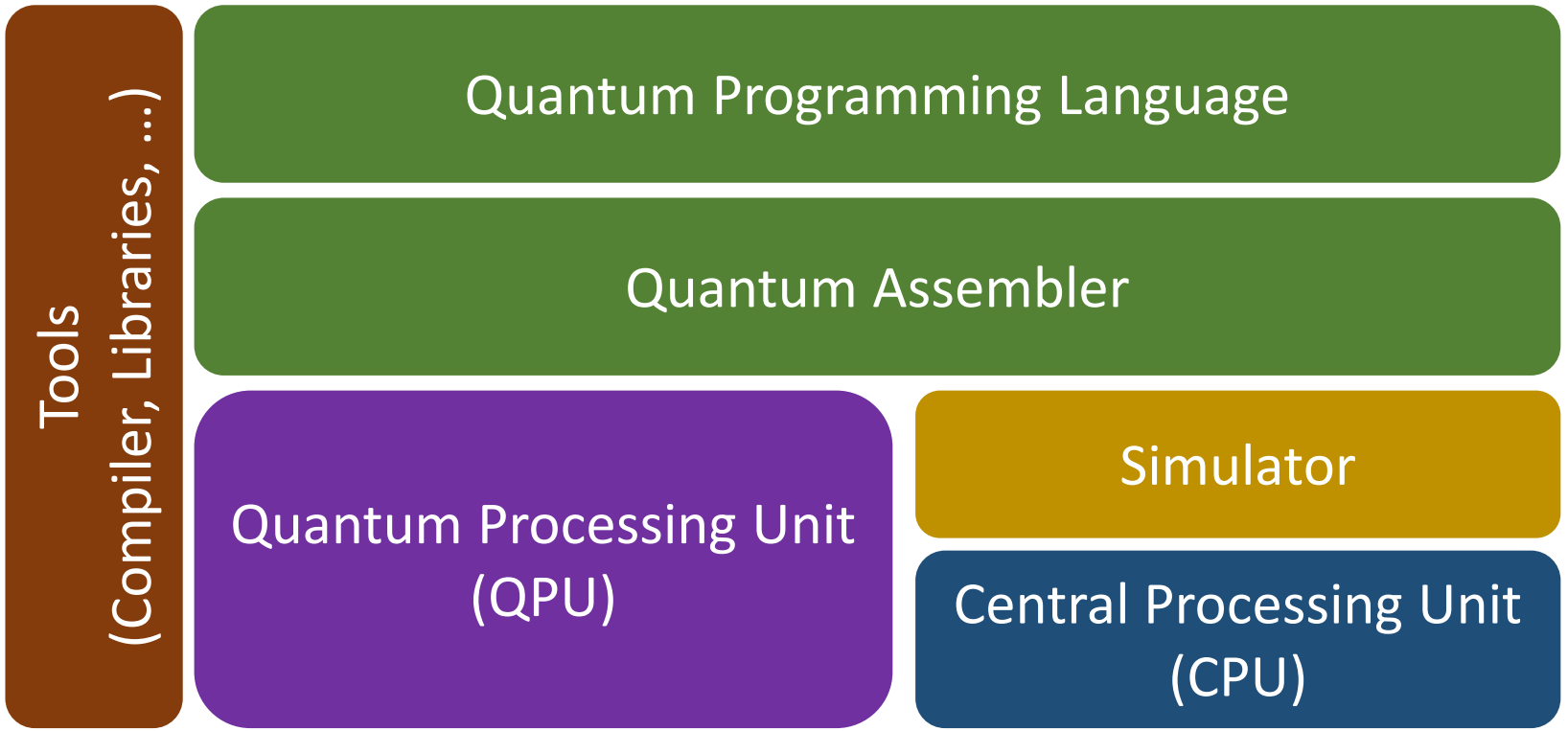
# Qiskit and Deutsch-Jozsa Algorithm

**Professor Dr. rer. nat. habil. Sven Groppe**
https://www.ifis.uni-luebeck.de/~groppe

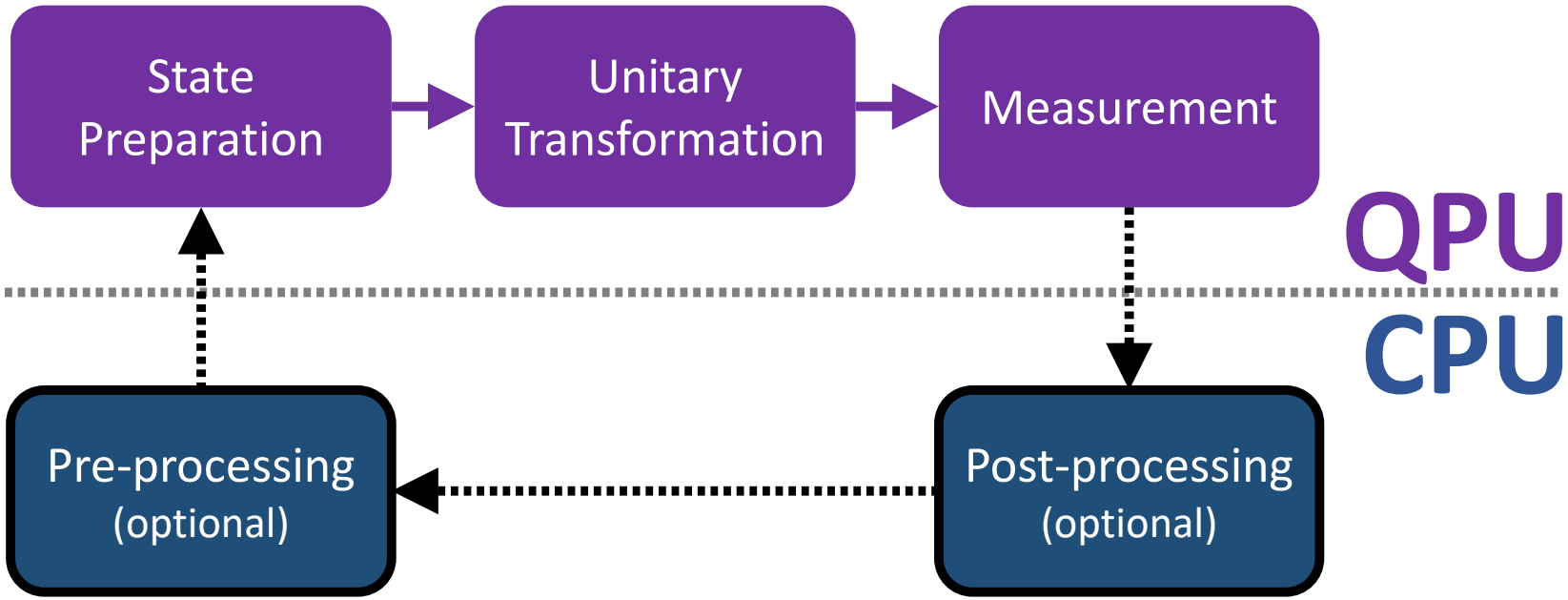
# Levels of Quantum Programming



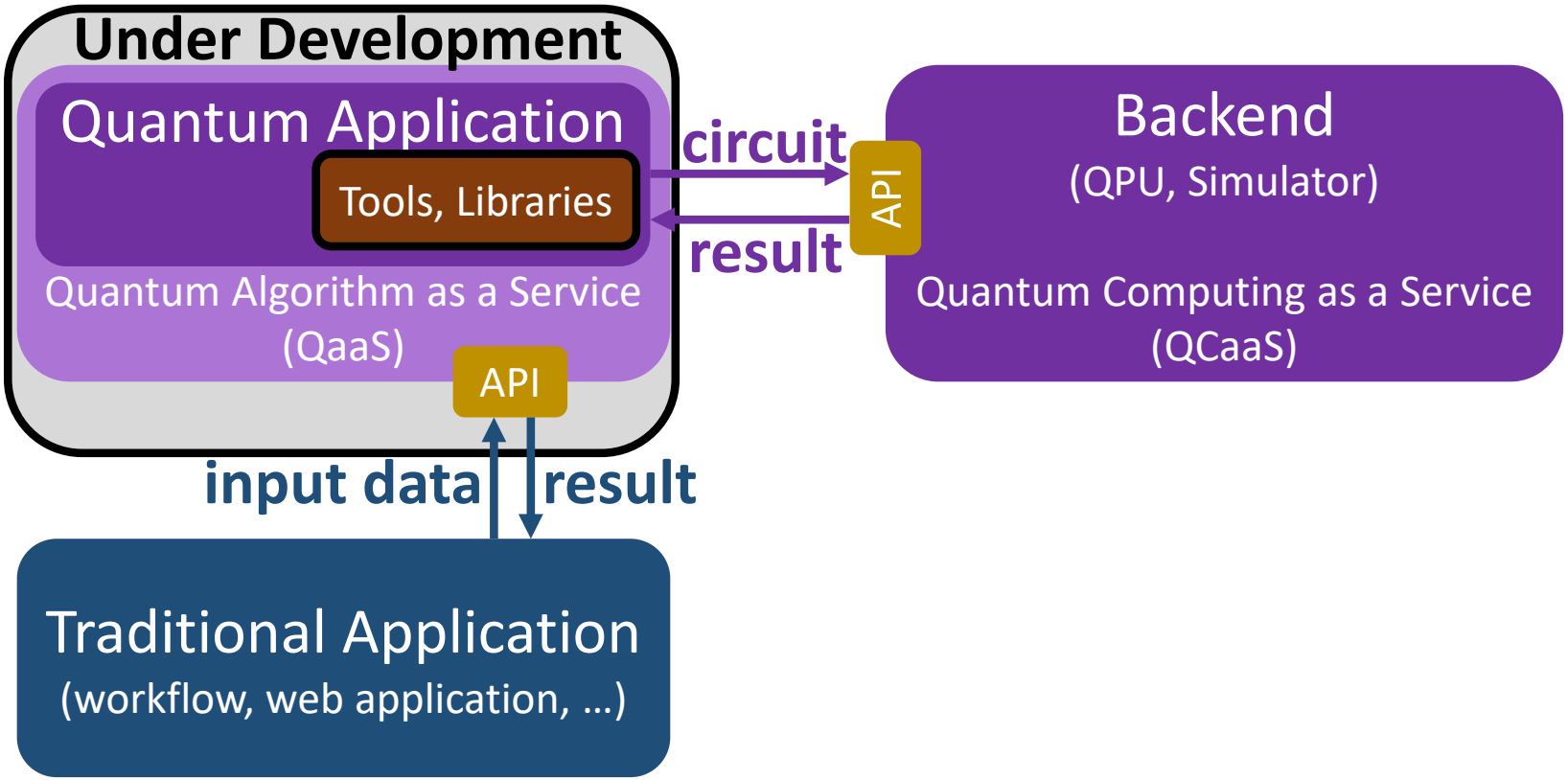
# Language and Tools Stack



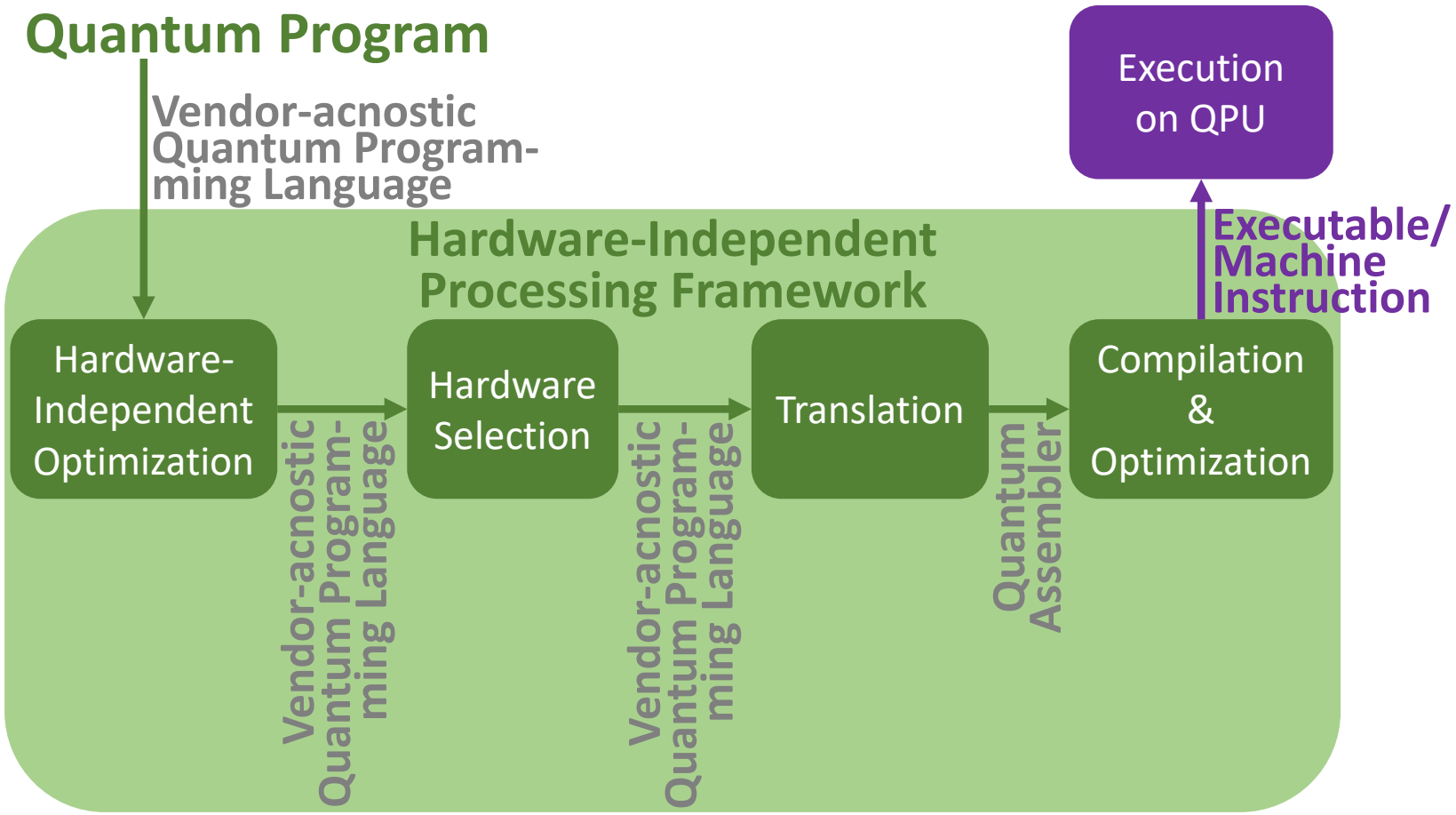
# CPU and QPU Processing Steps



# Quantum Algorithms as a Service



# Hardware Independent Translation



# Overview Quantum Programming Languages & their Stacks

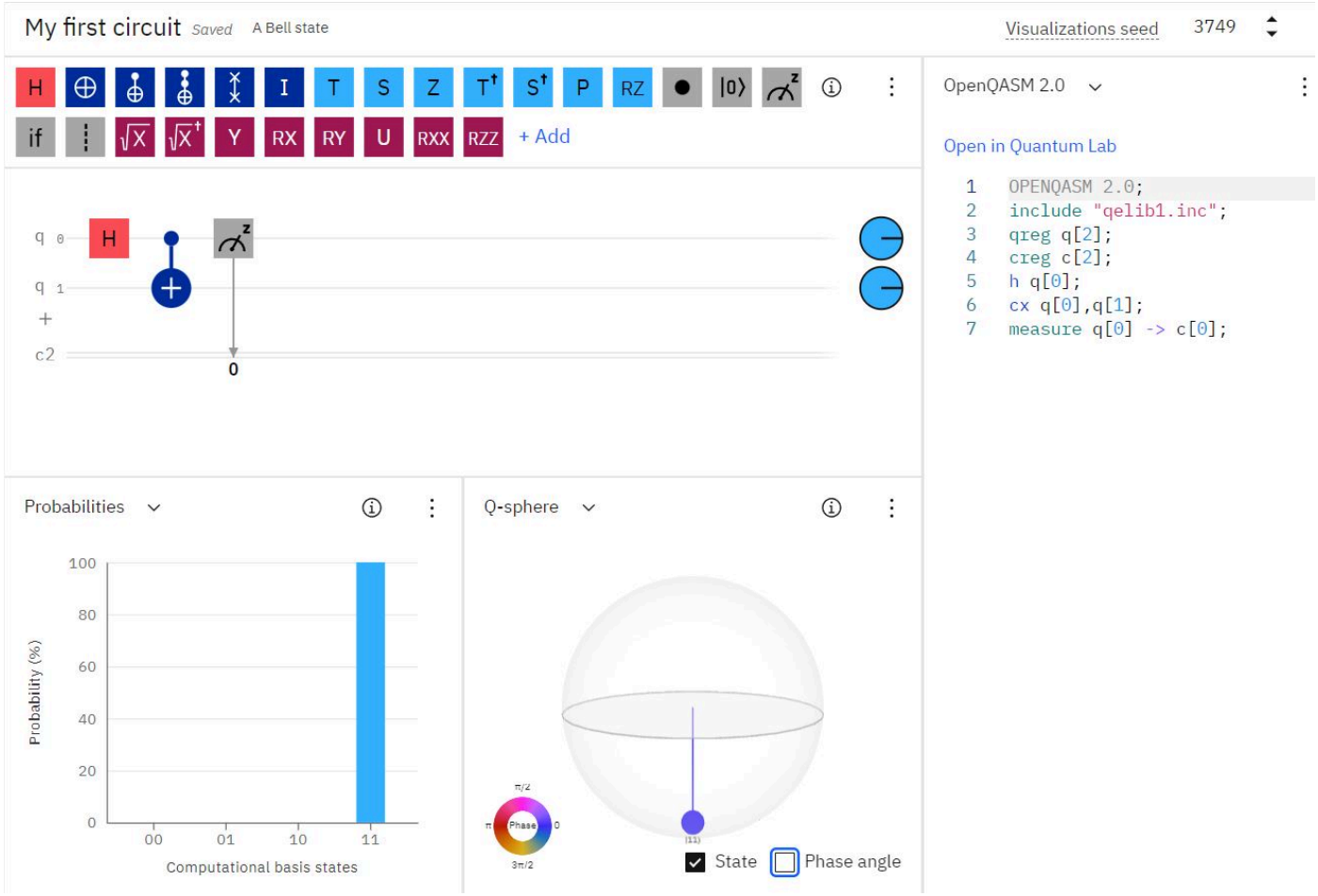
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Quantum Universal Languages** | XACC<br>ProjectQ (+ AQT devices, AWS Braket, IonQ) | | | | | | | |
| **Provider** | IBM | Rigetti | DWave | Xanadu | Google | Microsoft | Qilimanjaro | ETH Zürich |
| **Full-stack Libraries** | Qiskit | Forest | | Straw-berry Fields | Cirq | Quantum Development Kit | | Silq |
| **Quantum algorithms** | Qiskit ml,finance chemistry,opt. | Grove | DWave Ocean | | OpenFermion -Cirq | Q# | | |
| **Quantum circuits** | Qiskit Terra | pyquil | | | Cirq | | Qibo | |
| **Assembly language** | Open QASM | Quil | QMASM | Blackbird | Open QASM/ Quil | Internal Quantum Simulator | Open QASM | Internal Quantum Simulator |
| **Hardware** | Quantum Device | | | | | | | Silq2Qiskit |

This list is not complete. "Quantum Language" refers to specialized programming languages and also to libraries to write quantum programs on top of well-known programming languages for classical computation.

# Qiskit

- Installation
  - Local installation via Python, Anaconda and Jupyter
  - Cloud
    - **IBM Quantum (Lab)**
      - Graphically build quantum circuits
      - Jupyter-notebooks for Qiskit without installation
      - Quantum system details like status, topology, calibration data, and access details of IBM quantum systems
      - Running quantum circuits on real quantum computers
      - Step-by-step tutorials and guides
    - **Strangeworks**
      - Hardware-agnostic, software-inclusive, collaborative environment for quantum development
      - Support of many important quantum frameworks without installation and configuration
      - Running quantum circuits on real quantum computers

# IBM Quantum Composer



# Qiskit Library/Python

- Importing important packages, creating and drawing first quantum circuit with one qubit initialized with $|0\rangle$...

## Program:

```
from qiskit import QuantumCircuit
import kaleidoscope.qiskit
from kaleidoscope import bloch_sphere
from qiskit.visualization import array_to_latex
from IPython.display import Latex

circuit = QuantumCircuit(1)
circuit.draw('latex', initial_state=True)
```

## Output:

$q : |0\rangle$ ——

# Printing the state vector & visualizing in the bloch sphere...

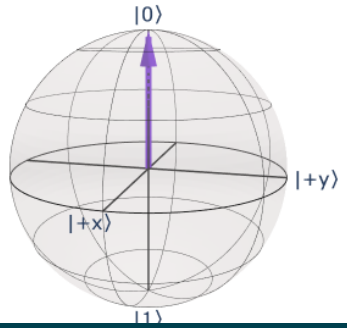
- Determine state vector of circuit for printing and visualizing

## Program:

```
display(array_to_latex(circuit.statevector(), prefix="State = "))
bloch_sphere(circuit.statevector())
```

## Output:

$$State = \begin{bmatrix} 1 & 0 \end{bmatrix}$$



# Extending the circuit...

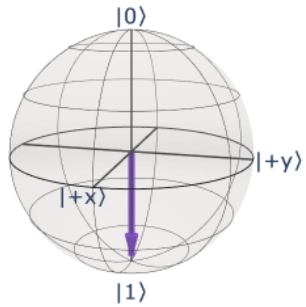
- Add a NOT(X) gate to the circuit resulting in $|1\rangle$...

## Program:

```
circuit.x(0)
circuit.draw('latex', initial_state=True)
display(array_to_latex(circuit.statevector(), prefix="State = "))
bloch_sphere(circuit.statevector())
```

## Output:

$$State = \begin{bmatrix} 0 & 1 \end{bmatrix}$$



# Matrix representations of gates

- Printing the matrix representations of gates (here X gate)...

## Program:

```
from qiskit.circuit.library import *
xgate = XGate()
print('X gate matrix:')
array_to_latex(xgate.to_matrix())
```

## Output:

X gate matrix:

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

# A circuit with a Hadamard transform...

## Program:

```
circuit = QuantumCircuit(1)
circuit.h(0)
circuit.draw('latex', initial_state=True)
```

## Output:

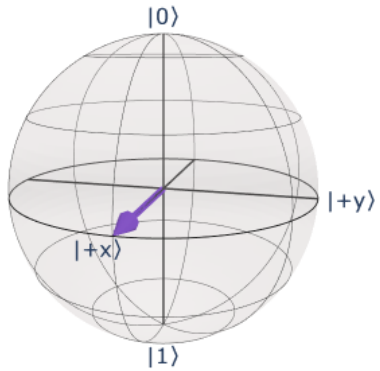
$q : |0\rangle$ — H —

## Program:

```
display(array_to_latex(circuit.statevector(),
                       prefix="|+\\rangle = "))
bloch_sphere(circuit.statevector())
```

## Output:

$$|+\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

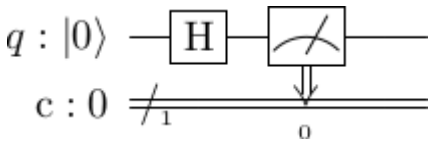


# Adding a measurement...

## Program:

```
circuit = QuantumCircuit(1,1)
circuit.h(0)
circuit.measure(0,0)
circuit.draw('latex', initial_state=True)
```

## Output:




- Open QASM output of the circuit...

## Program:

```
circuit.qasm()
```

## Output:

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[1];
creg c[1];
h q[0];
measure q[0] -> c[0];
```

# Simulating a circuit

- Initialize simulator, execute a job with 1000 shots and plot the resultant histogram...

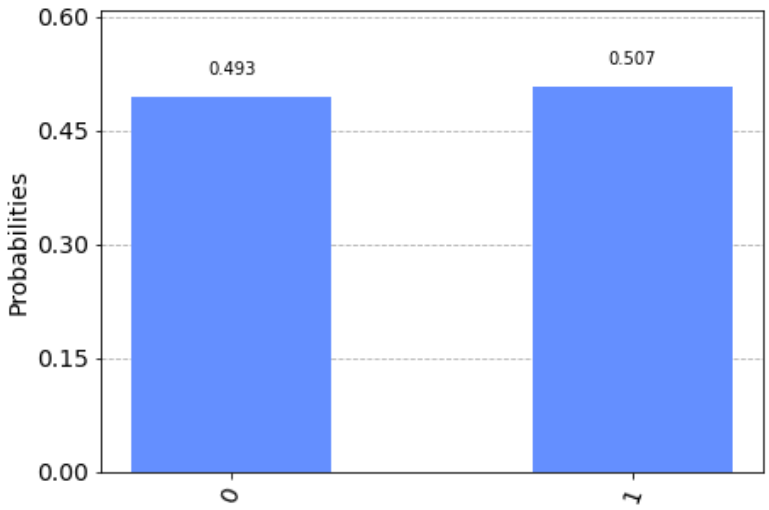
## Program:

```
from qiskit import BasicAer,
                   execute
from qiskit.visualization
     import plot_histogram

local_simulator =
  BasicAer.get_backend('qasm_simulator')

job = execute(circuit,
              backend=local_simulator,
              shots=1000)
plot_histogram(job.result().get_counts())
```

## Output:

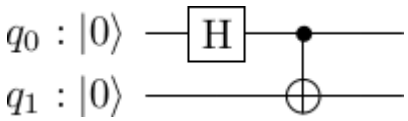


# Circuits with 2 Qubits (Correlated Bell State)

## Program:

```
circuit = QuantumCircuit(2)
circuit.h(0)
circuit.cx(0, 1)
circuit.draw('latex', initial_state=True)
```

## Output:




## Program:

```
array_to_latex(circuit.statevector(),
  prefix="\\text{CNOT}|0{+}\\rangle = ")
```

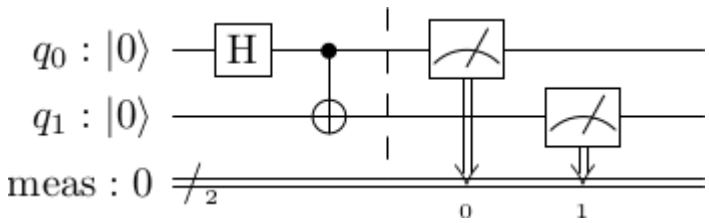
## Output:

$$\text{CNOT}|0+\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}$$

## Program:

```
circuit.measure_all()
circuit.draw('latex', initial_state=True)
```

## Output:



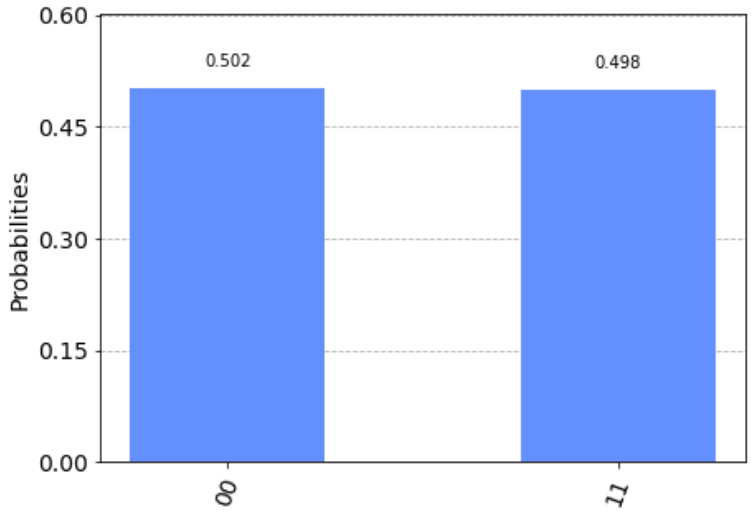
# Simulating the Correlated Bell State

## Program:

```
local_simulator =
  BasicAer.get_backend('qasm_simulator')

job = execute(circuit, backend=local_simulator,
              shots=5000)
plot_histogram(job.result().get_counts())
```

## Output:

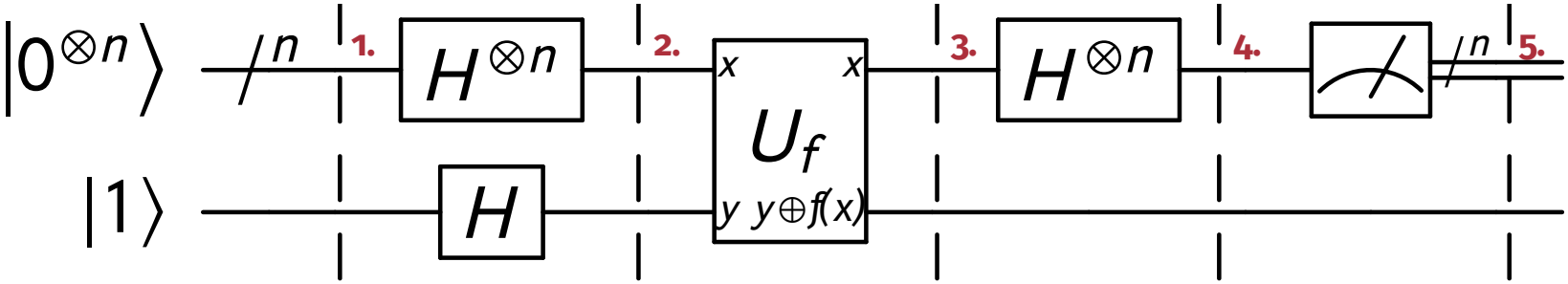


# Deutsch-Jozsa Algorithm

- **Purpose:** shows that the power of state superposition in a quantum algorithm can significantly reduce the runtime complexity compared to its classical counterpart by **exponential speedup**
- **Problem** to be solved
  - **Objective:** Determine the nature of a function, $f : \{0,1\}^n \rightarrow \{0,1\}$, which takes as input an $n$-bit integer and returns either $0$ or $1$
  - We are guaranteed that this function
    - is either constant (returning either $0$ or $1$ on all inputs), or
    - perfectly balanced (returning $0$ on half of the inputs and $1$ on the other half)
  - **Goal:** Determine of which type (constant or balanced) the function is with minimum number of function executions

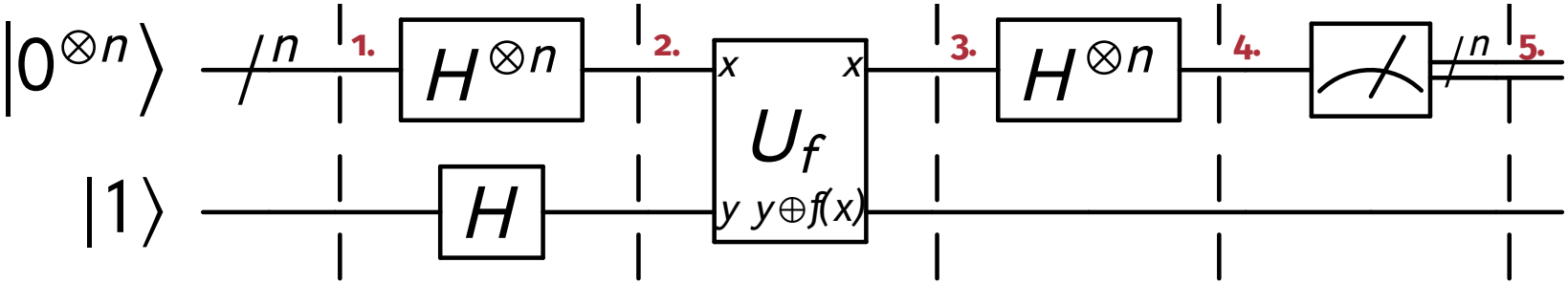
- References:
  - $n = 1$, Deutsch algorithm: **D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. Proceedings of the Royal Society A. 400 (1818): 97–117, 1985**
  - $n > 1$, Deutsch-Jozsa algorithm: **D. Deutsch, R. Jozsa. Rapid solutions of problems by quantum computation. Proceedings of the Royal Society of London A. 439: 553–558, 1992**

# Deutsch-Jozsa Algorithm - Idea & Circuit



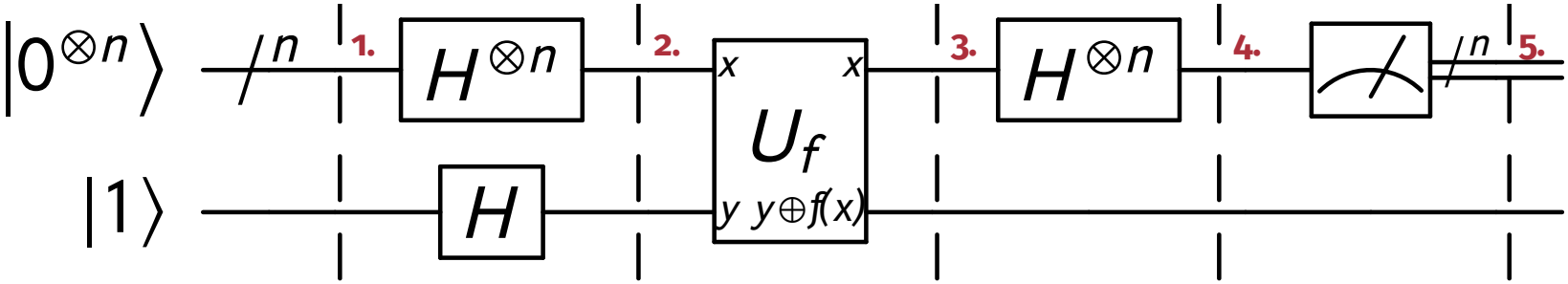

- **Step 1 & 2:** Bring $n$ qubits into superposition with all values $0, 1, ..., 2^n - 1$ having the same probability

# Deutsch-Jozsa Algorithm - Idea & Circuit



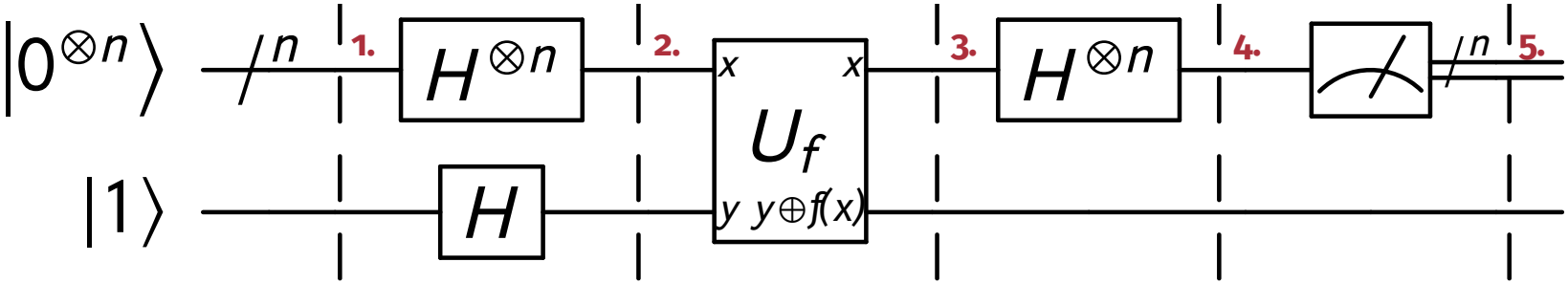

- **Step 1 & 2:** Bring $n$ qubits into superposition with all values $0, 1, ..., 2^n - 1$ having the same probability
- **Step 3:** Apply the oracle function on all input values simultaneously
  - **Constant** functions do not change the state (up to a global phase)
  - **Balanced** functions cause state changes for half of the input values

# Deutsch-Jozsa Algorithm - Idea & Circuit




- **Step 1 & 2:** Bring $n$ qubits into superposition with all values $0, 1, ..., 2^n - 1$ having the same probability
- **Step 3:** Apply the oracle function on all input values simultaneously
  - **Constant** functions do not change the state (up to a global phase)
  - **Balanced** functions cause state changes for half of the input values
- **Step 4 & 5:** Interfere the result and check output
  - **Constant** functions: No quantum state change & Hadamard is reversible
    $\Rightarrow$ Output of the circuit = Input of the circuit
  - **Balanced** functions: Quantum state changes of previous step
    $\Rightarrow$ Output of the circuit $\neq$ Input of the circuit

# Deutsch-Jozsa Algorithm - Details of **Step 1**




## $n = 1$:

$$|\psi_1\rangle = |0\rangle|1\rangle = |0\rangle \otimes |1\rangle$$

$$= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$= |01\rangle$$

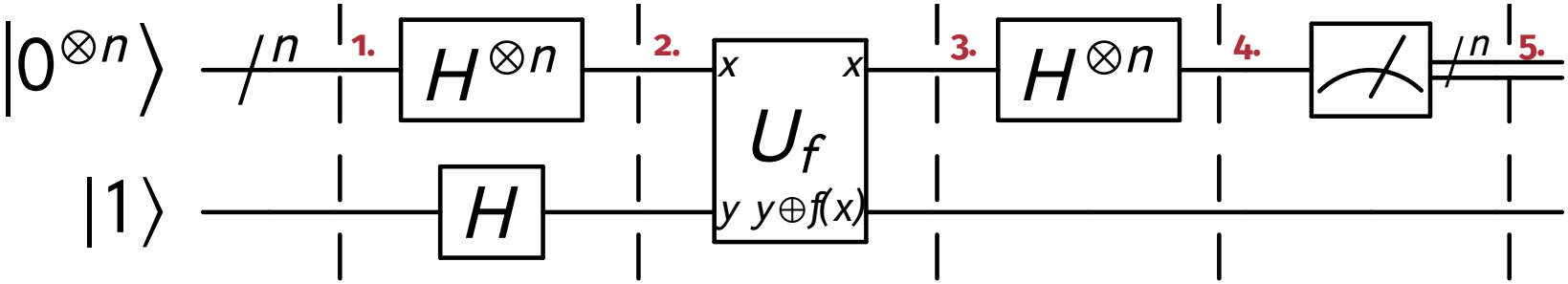
## **Arbitrary** $n \geq 1$:

$$|\psi_1\rangle = |0\rangle \cdots |0\rangle|1\rangle = |0\rangle \otimes \cdots \otimes |0\rangle \otimes |1\rangle$$

$$= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \cdots \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$= |0 \cdots 01\rangle$$

# Deutsch-Jozsa Algorithm - Details of **Step 2**




## $n = 1$:

$$H_2 = H \otimes H = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$|\psi_2\rangle = H_2 \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$$

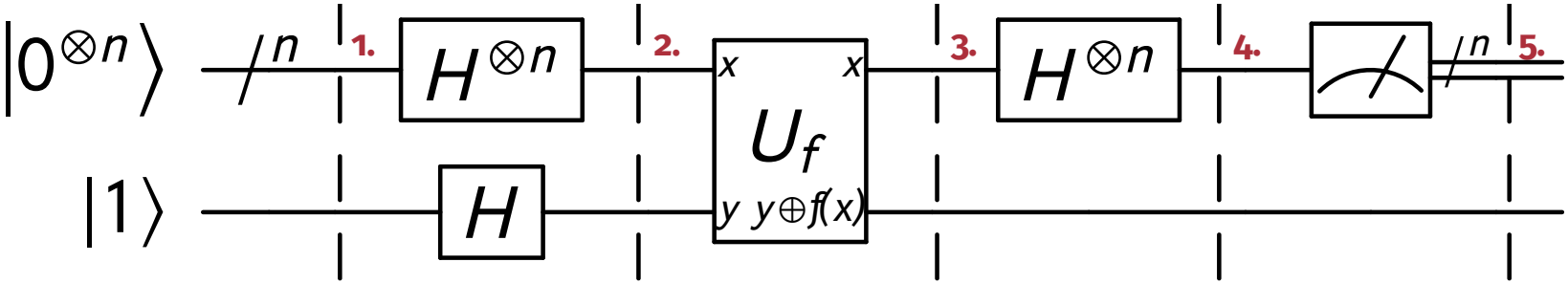
$$= \frac{1}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{(|0\rangle+|1\rangle)(|0\rangle-|1\rangle)}{2}$$

## **Arbitrary** $n \geq 1$:

$$H_{n+1} = H \otimes \cdots \otimes H$$

$$|\psi_2\rangle = H_{n+1} |0 \cdots 01\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle(|0\rangle - |1\rangle)$$

# Deutsch-Jozsa Algorithm - Details of **Step 3**




## $n = 1$:

Apply the oracle function in gate mapping $|x\rangle|b\rangle \mapsto |x\rangle|b \oplus f(x)\rangle$ with $|x\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|b\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$:

| $\|b \oplus f(x)\rangle$ | $b = 0$ | $b = 1$ |
|---|---|---|
| $f(x) = 0$ | $\|0\rangle$ | $-\|1\rangle$ |
| $f(x) = 1$ | $\|1\rangle$ | $-\|0\rangle$ |

$|\psi_3\rangle = |x\rangle|b \oplus f(x)\rangle$
(where $\oplus$ is addition modulo 2)

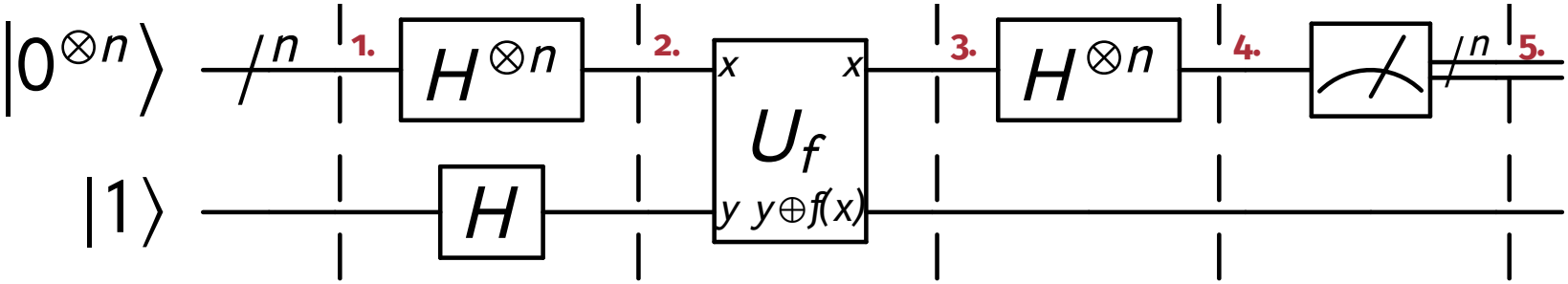
## Arbitrary $n \geq 1$:

The result of applying the quantum oracle is $\frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle(|f(x)\rangle - |1 \oplus f(x)\rangle)$.

For each $x$, $f(x)$ is either 0 or 1. Testing these two possibilities, we see the above state is equal to
$|\psi_3\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle(|0\rangle - |1\rangle)$.

At this point the last qubit $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ may be ignored and therefore below is remained:
$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle$.

# Deutsch-Jozsa Algorithm - Details of **Step 3**




## $n = 1$: Examples of oracles:

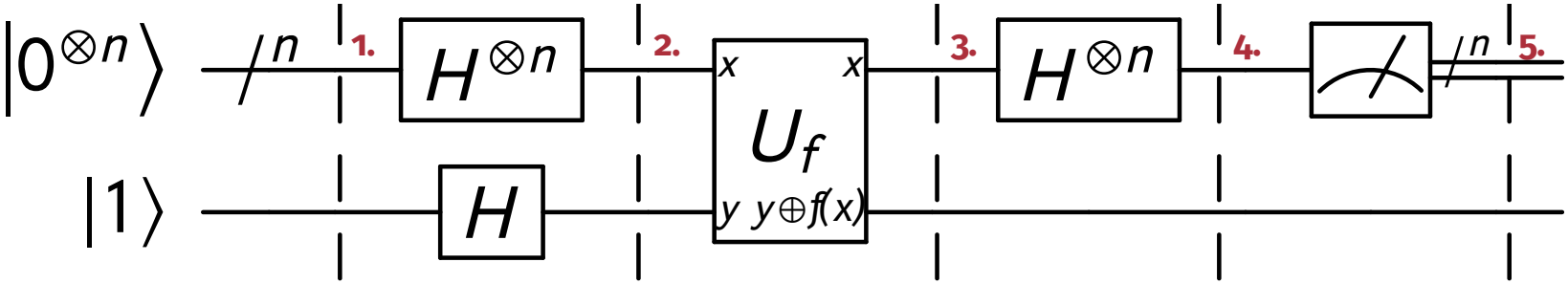
| $\lvert x\rangle\lvert b \oplus f(x)\rangle$ | $f(0) = 0$ | $f(0) = 1$ |
|---|---|---|
| $f(1) = 0$ | $\frac{1}{2}\begin{bmatrix}1\\-1\\1\\-1\end{bmatrix}$ | $\frac{1}{2}\begin{bmatrix}-1\\1\\1\\-1\end{bmatrix}$ |
| $f(1) = 1$ | $\frac{1}{2}\begin{bmatrix}1\\-1\\-1\\1\end{bmatrix}$ | $\frac{1}{2}\begin{bmatrix}-1\\1\\-1\\1\end{bmatrix}$ |

## Arbitrary $n \geq 1$: Example oracle:

| $f(x_1, x_2) = x_1 \oplus x_2$ | $x_1 = 0$ | $x_1 = 1$ |
|---|---|---|
| $x_2 = 0$ | 0 | 1 |
| $x_2 = 1$ | 1 | 0 |

$$\psi_3 = \frac{|00\rangle\otimes(|0\oplus0\rangle-|1\oplus0\oplus0\rangle)+|01\rangle\otimes(|0\oplus1\rangle-|1\oplus0\oplus1\rangle)}{2\cdot\sqrt{2}}$$
$$+\frac{|10\rangle\otimes(|1\oplus0\rangle-|1\oplus1\oplus0\rangle)+|11\rangle\otimes(|1\oplus1\rangle-|1\oplus1\oplus1\rangle)}{2\cdot\sqrt{2}}$$
$$=\frac{|00\rangle\otimes(|0\rangle-|1\rangle)-|01\rangle\otimes(|0\rangle-|1\rangle)-|10\rangle\otimes(|0\rangle-|1\rangle)+|11\rangle\otimes(|0\rangle-|1\rangle)}{2\cdot\sqrt{2}}$$
$$=\frac{1}{2}\left(|00\rangle-|01\rangle-|10\rangle+|11\rangle\right)\otimes\frac{1}{\sqrt{2}}\left(|0\rangle-|1\rangle\right)$$
$$=\frac{1}{\sqrt{2}}\left(|0\rangle-|1\rangle\right)\otimes\frac{1}{\sqrt{2}}\left(|0\rangle-|1\rangle\right)\otimes\frac{1}{\sqrt{2}}\left(|0\rangle-|1\rangle\right)$$

# Deutsch-Jozsa Algorithm - Details of **Step 4**




## $n = 1$:

$$H \otimes I = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$|\psi_4\rangle = (H \otimes I)|\psi_3\rangle$
$= (H \otimes I)|x\rangle|b \oplus f(x)\rangle$

## **Arbitrary** $n \geq 1$:

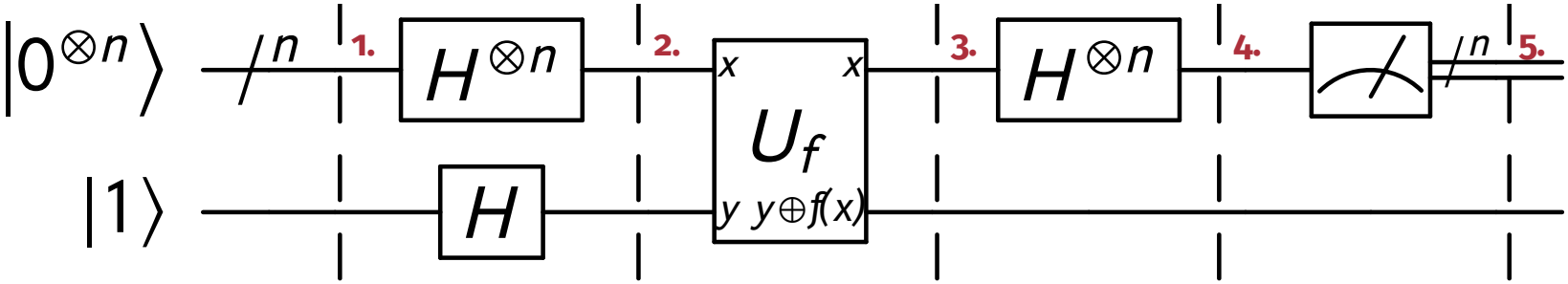
Hadamards transform to:

$$|\psi_4\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \left[ \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} (-1)^{x \cdot y} |y\rangle \right]$$
$$= \frac{1}{2^n} \sum_{y=0}^{2^n-1} \left[ \sum_{x=0}^{2^n-1} (-1)^{f(x)} (-1)^{x \cdot y} \right] |y\rangle$$

where

$x \cdot y = x_0\, y_0 \oplus x_1\, y_1 \oplus \cdots \oplus x_{n-1}\, y_{n-1}$ is the sum of the bitwise product (where $\oplus$ is addition modulo 2).

# Deutsch-Jozsa Algorithm - Details of **Step 4**




## $n = 1$: Examples of oracles:

| $(H \otimes I)\|x\rangle\|b \oplus f(x)\rangle$ | $f(0) = 0$ | $f(0) = 1$ |
|---|---|---|
| $f(1) = 0$ | $\frac{1}{2\cdot\sqrt{2}} \begin{bmatrix} 2 \\ -2 \\ 0 \\ 0 \end{bmatrix}$ | $\frac{1}{2\cdot\sqrt{2}} \begin{bmatrix} 0 \\ 0 \\ -2 \\ 2 \end{bmatrix}$ |
| $f(1) = 1$ | $\frac{1}{2\cdot\sqrt{2}} \begin{bmatrix} 0 \\ 0 \\ 2 \\ -2 \end{bmatrix}$ | $\frac{1}{2\cdot\sqrt{2}} \begin{bmatrix} -2 \\ 2 \\ 0 \\ 0 \end{bmatrix}$ |

## Arbitrary $n \geq 1$: Example oracle:

| $f(x_1, x_2) = x_1 \oplus x_2$ | $x_1 = 0$ | $x_1 = 1$ |
|---|---|---|
| $x_2 = 0$ | 0 | 1 |
| $x_2 = 1$ | 1 | 0 |

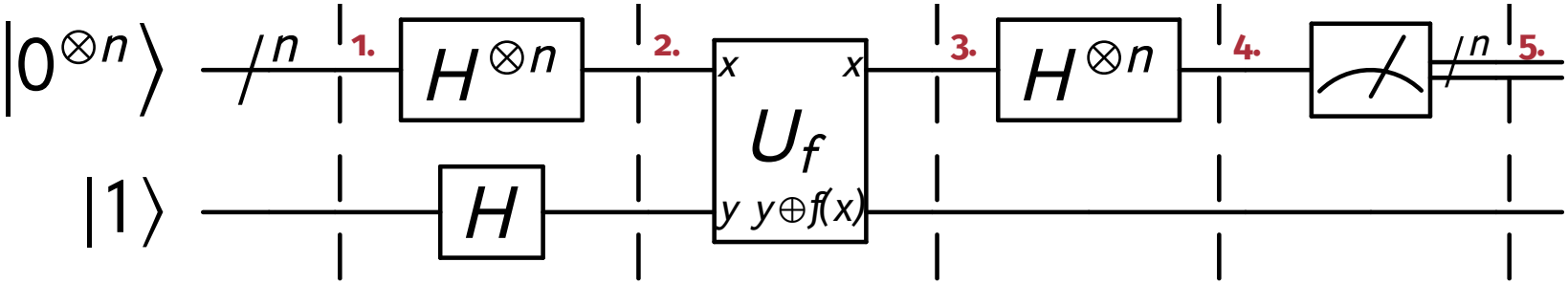
$\psi_3 = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$

The result of applying Hadamards on the first and second qubit is:

$$|\psi_4\rangle = |1\rangle \otimes |1\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$\Rightarrow$ It will be $|11\rangle$ measured in the next step!

# Deutsch-Jozsa Algorithm - Details of **Step 5**




## $n = 1$: Examples of oracles:

$|0\rangle$ is measured (neglecting noise) as 1st qubit for $\begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \\ 0 \\ 0 \end{bmatrix}$ $\begin{matrix} 00 \\ 01 \\ 10 \\ 11 \end{matrix}$, rest is analogous

| | $f(0) = 0$ | $f(0) = 1$ |
|---|---|---|
| $f(1) = 0$ | $\|0\rangle \Rightarrow$ constant | $\|1\rangle \Rightarrow$ balanced |
| $f(1) = 1$ | $\|1\rangle \Rightarrow$ balanced | $\|0\rangle \Rightarrow$ constant |

## Arbitrary $n \geq 1$:

The probability of measuring $|0\rangle^{\otimes n}$ is:

$$\left| \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \right|^2$$

which evaluates to

- 1 (i.e., $|0\rangle^{\otimes n}$ is measured) if $f(x)$ is constant (constructive interference) and
- 0 (i.e., $|0\rangle^{\otimes n}$ is **not** measured) if $f(x)$ is balanced (destructive interference).

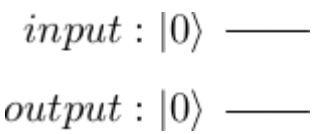
# Constant Oracles in Qiskit

## Oracle 0 : Constant 0

### Program:

```
from qiskit import QuantumRegister
input = QuantumRegister(1, name='input')
output = QuantumRegister(1, name='output')
constant0 = QuantumCircuit(input, output,
                           name='oracle')
oracle0 = constant0.to_instruction()
constant0.draw('latex', initial_state=True)
```

### Output:

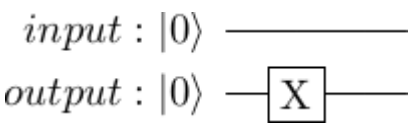



## Oracle 1 : Constant 1

### Program:

```
input = QuantumRegister(1, name='input')
output = QuantumRegister(1, name='output')
constant1 = QuantumCircuit(input, output,
                           name='oracle')
constant1.x(output)
oracle1 = constant1.to_instruction()
constant1.draw('latex', initial_state=True)
```

### Output:



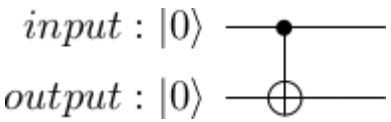
# Balanced Oracles in Qiskit

## Oracle I: Identity

### Program:

```
input = QuantumRegister(1, name='input')
output = QuantumRegister(1, name='output')
identity = QuantumCircuit(input, output,
                          name='oracle')
identity.cx(input, output)
oracleI = identity.to_instruction()
identity.draw('latex', initial_state=True)
```

### Output:

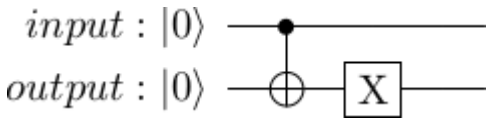



## Oracle X: Invert/Not/X

### Program:

```
input = QuantumRegister(1, name='input')
output = QuantumRegister(1, name='output')
invert = QuantumCircuit(input, output,
                        name='oracle')
invert.cx(input, output)
invert.x(output)
oracleX = identity.to_instruction()
invert.draw('latex', initial_state=True)
```

### Output:



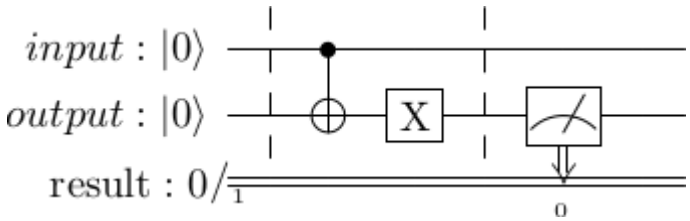
# Run an Oracle

## Oracle X: Invert

### Program:

```
from qiskit import ClassicalRegister
result = ClassicalRegister(1, name='result')
circuit = QuantumCircuit(input, output, result)
circuit.barrier()
circuit.compose(invert, inplace=True)
circuit.barrier()
circuit.measure(output, result)
circuit.draw('latex', initial_state=True)
```

### Output:




### Program:

```
execute(circuit, backend=local_simulator,
        shots=1000).result().get_counts()
```

### Output:

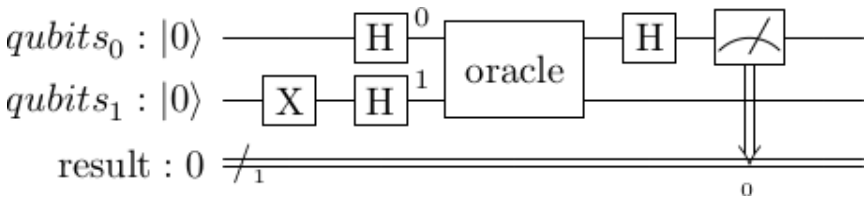
```
{'1': 1000}
```

# Simulating Deutsch's Algorithm

## Program:

```
qr = QuantumRegister(2, name='qubits')
cr = ClassicalRegister(1, name='result')
circuit = QuantumCircuit(qr, cr)
circuit.x(qr[1])
circuit.h(qr)
# oracle0, oracle1, oracleI or oracleX:
circuit.append(oracle0, [qr[0], qr[1]])
circuit.h(qr[0])
circuit.measure(qr[0], cr[0])
circuit.draw('latex', initial_state=True,
             justify="right")
```

## Output:




## Program:

```
counts = execute(circuit,
            backend=local_simulator,
            shots=1).result().get_counts()
counts['BALANCED'] = counts.pop('1', None)
counts['CONSTANT'] = counts.pop('0', None)
counts
```

## Output:

```
{'BALANCED': None,
 'CONSTANT': 1}
```

# Running Deutsch's Algorithm on a real Quantum Computer

## Program:

```
import qiskit.tools.jupyter
from qiskit import IBMQ
from qiskit.providers.ibmq import least_busy
provider = IBMQ.load_account()
least_busy_device = least_busy(provider.backends(simulator=False,
    filters=lambda b: b.configuration().n_qubits >= 2))
least_busy_device
```

## Output:

```
IBMQBackend('ibmq_lima') from IBMQ(hub='ibm-q', group='open', project='main')
```

## Some information about ibmq_lima:

| | |
|---|---|
| **n_qubits** | 5 |
| **quantum_volume*** | 8 |
| **pending_jobs** | 2 |
| **basis_gates** | ['id', 'rz', 'sx', 'x', 'cx', 'reset'] |
| **max_shots** | 20000 |
| **max_experiments** | 100 |

* Qiskit Textbook: Measuring Quantum Volume

# Job Status and Result

## Program:

```
job.status()
```

## Output:

```
JobStatus.RUNNING: 'job is actively running'
```

## Program:

```
job.wait_for_final_state()
job.status()
```

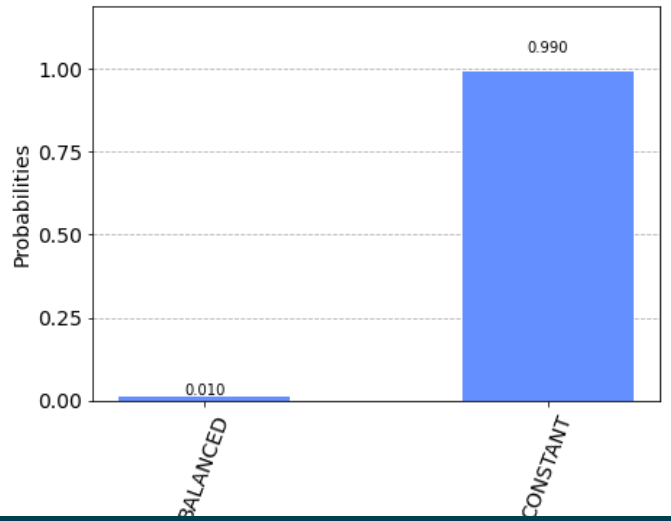
## Output:

```
JobStatus.DONE: 'job has successfully run'
```

## Program:

```
counts = job.result().get_counts()
counts['BALANCED'] = counts.pop('1', None)
# wrong answer:
counts['CONSTANT'] = counts.pop('0', None)
plot_histogram(counts)
```

## Output:

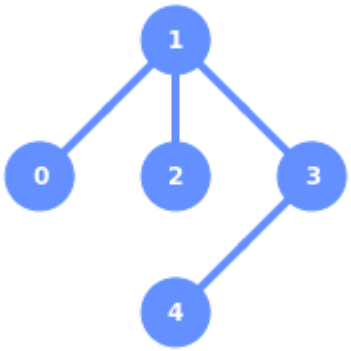


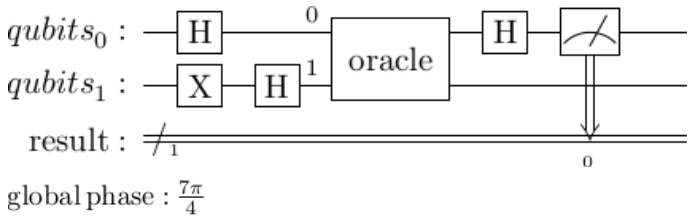
# Physical Mapping of Circuit

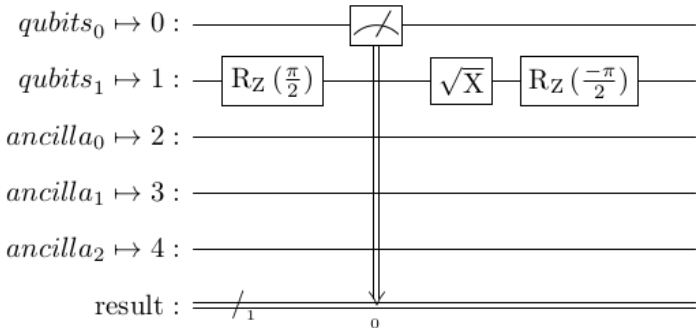
## Program:

```
from qiskit.visualization import plot_gate_map
from qiskit import transpile
display(plot_gate_map(least_busy_device))
job = execute(circuit, backend=least_busy_device, shots=1000, initial_layout=[0, 4])
display(circuit.draw('latex'))
transpile(circuit, least_busy_device).draw('latex')
```

## Output:



# Summary and Conclusions

- Qiskit
  - introduction
  - gates, circuits
  - output, visualizations (e.g., bloch sphere)
- Deutsch-Jozsa algorithm
  - problem to solve
  - why it works...
  - realization in Qiskit
  - running on a real quantum computer