Qiskit Fall Fest 2025

Qiskit 101

A guided introduction to the Qiskit workflow

Ángel Rodríguez
PhD Student
Materials Physics Center

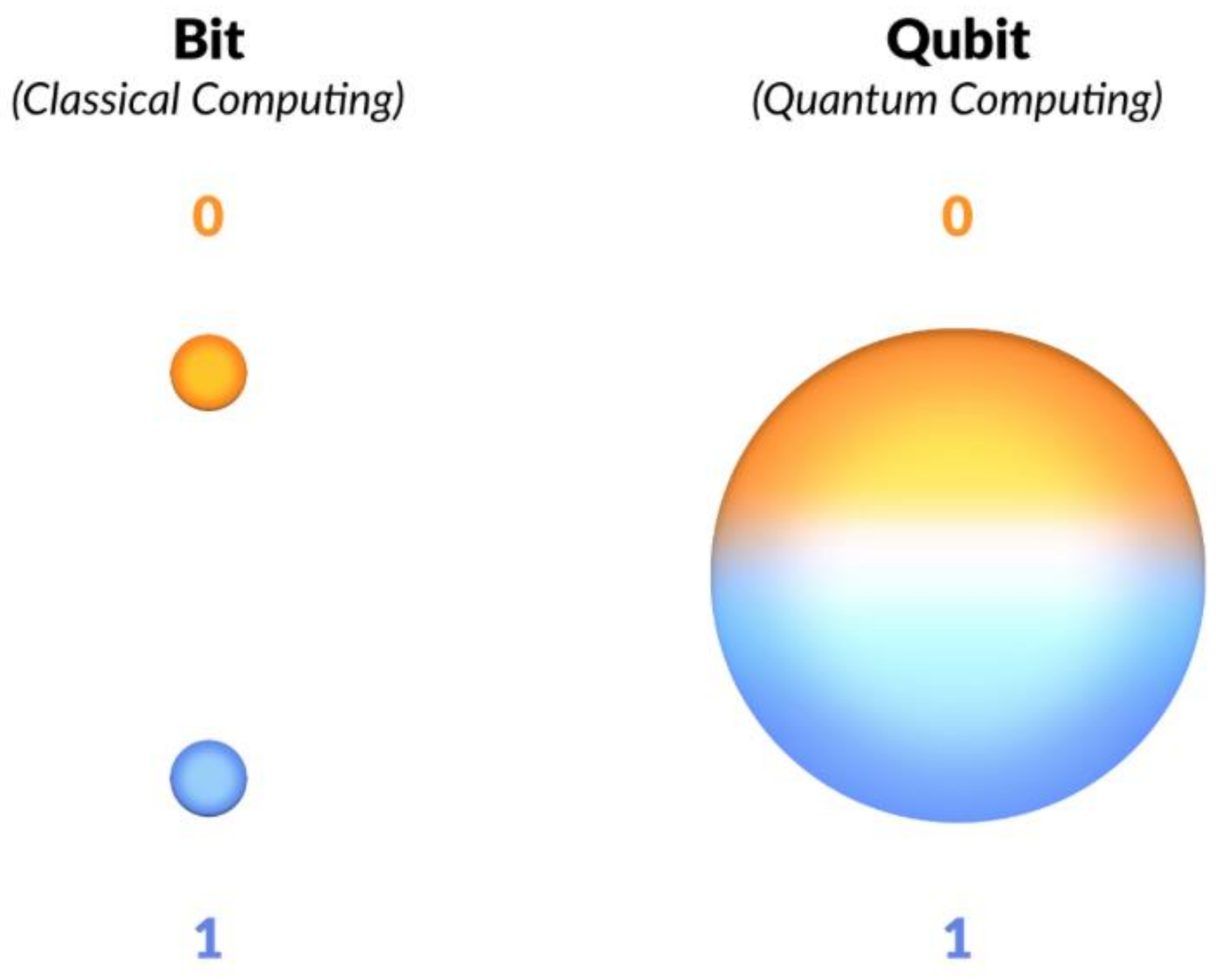
Benjamin Tirado
PhD Student
Centro de Física de Materiales
(CSIC – UPV/EHU)

Century of Quantum

Crash course on quantum computing

WTF is a qubit?

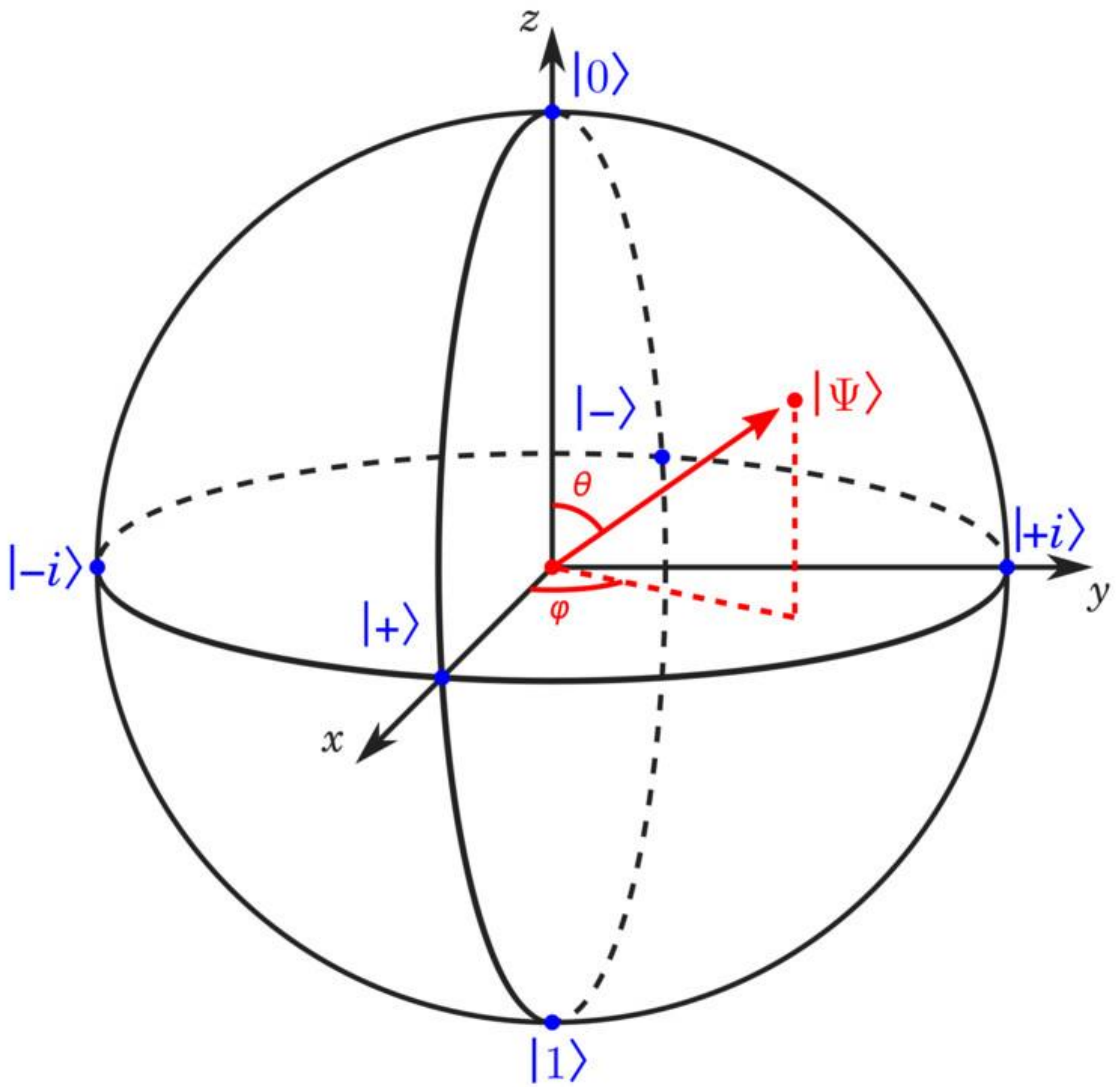
It's the basic unit of information in quantum computing



Crash course on quantum computing

Bloch sphere

It represents the state of a qubit in quantum computing

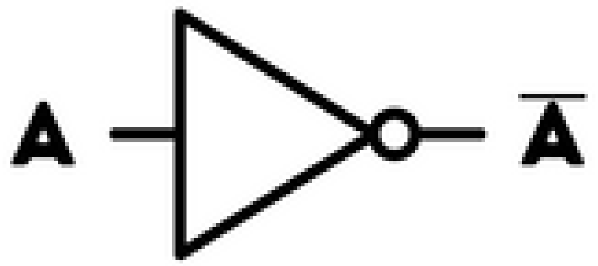


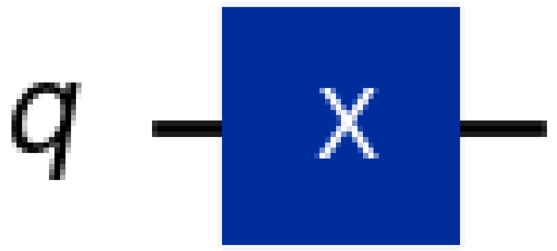
Crash course on quantum computing

Quantum gates

They manipulate the state of the qubit (resulting in rotations along the Bloch sphere)

X Gate (Bit flip): Quantum analogue to the classical NOT gate





INPUT	OUTPUT
0	1
1	0

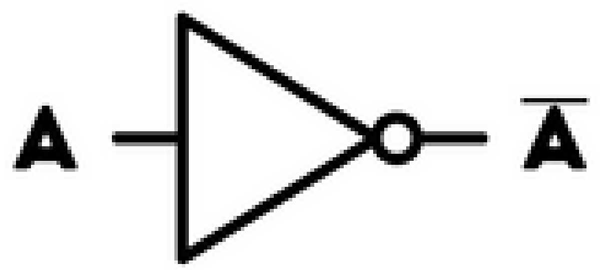
INPUT	OUTPUT
$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$

Crash course on quantum computing

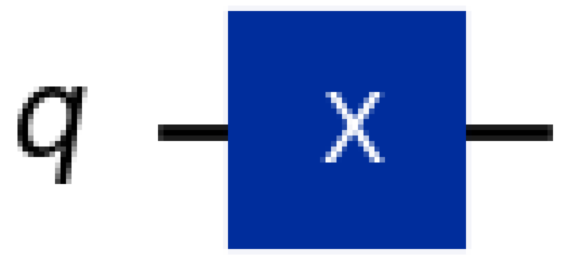
Quantum gates

They manipulate the state of the qubit (resulting in rotations along the Bloch sphere)

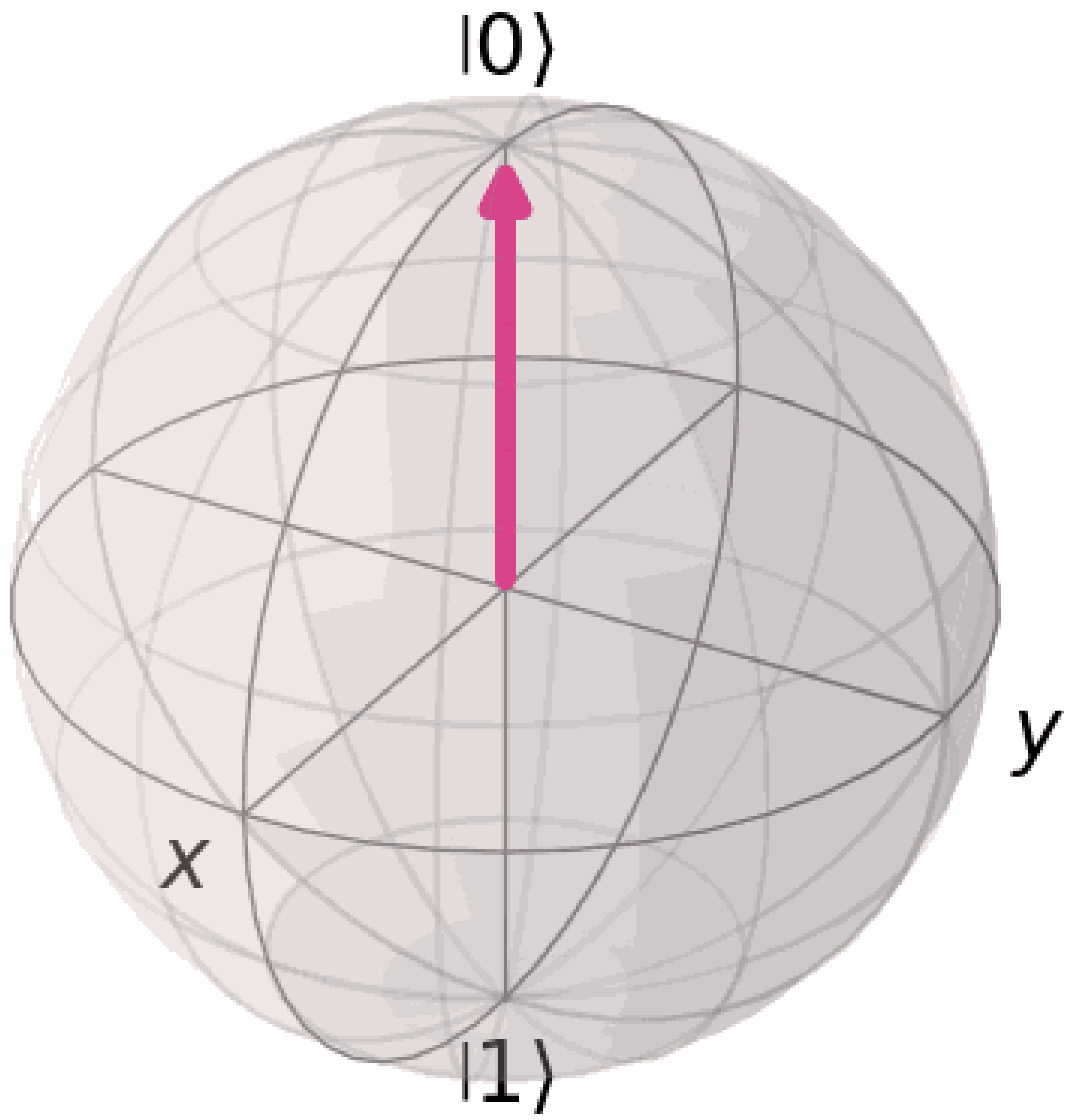
X Gate (Bit flip): Quantum analogue to the classical NOT gate



INPUT	OUTPUT
0	1
1	0



INPUT	OUTPUT
$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$

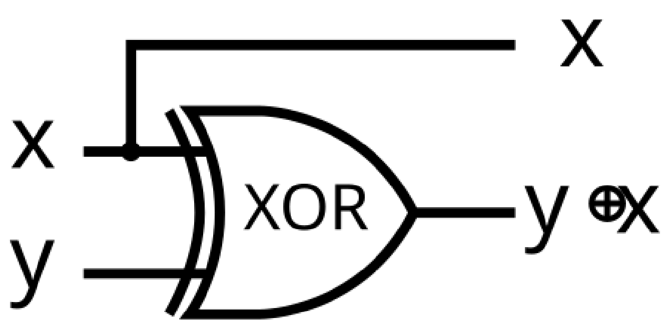


Crash course on quantum computing

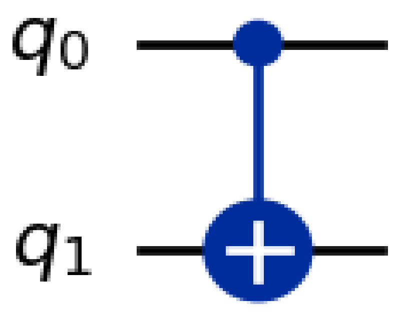
Quantum gates

They manipulate the state of the qubit (resulting in rotations along the Bloch sphere)

CX/CNOT Gate: Quantum analogue to the classical XOR gate



INPUT		OUTPUT	
x	y	x	y
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0



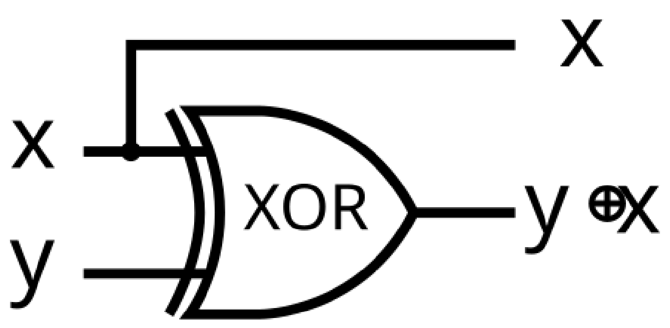
INPUT		OUTPUT	
q_0	q_1	q_0	q_1
$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$

Crash course on quantum computing

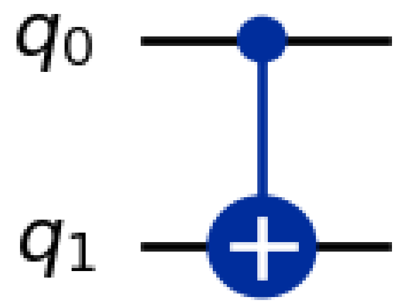
Quantum gates

They manipulate the state of the qubit (resulting in rotations along the Bloch sphere)

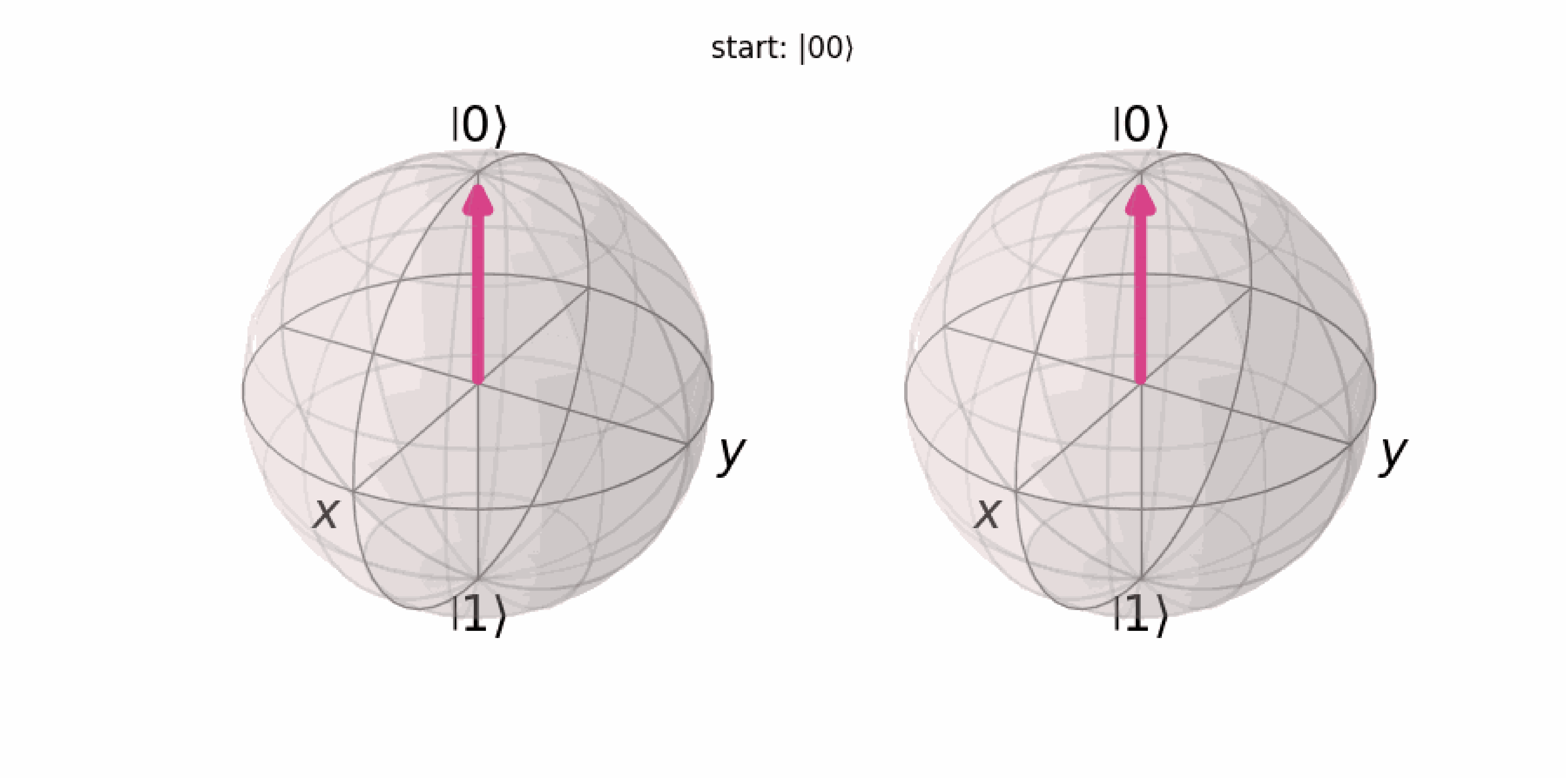
CX/CNOT Gate: Quantum analogue to the classical XOR gate



INPUT		OUTPUT	
x	y	x	y
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0



INPUT		OUTPUT	
q_0	q_1	q_0	q_1
$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$

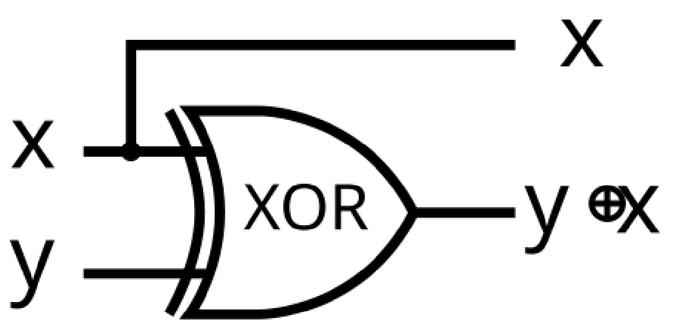


Crash course on quantum computing

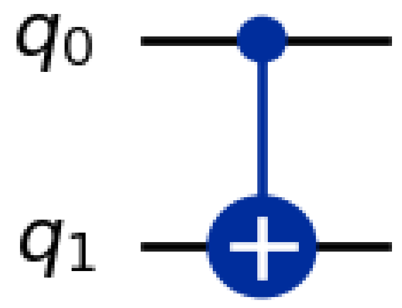
Quantum gates

They manipulate the state of the qubit (resulting in rotations along the Bloch sphere)

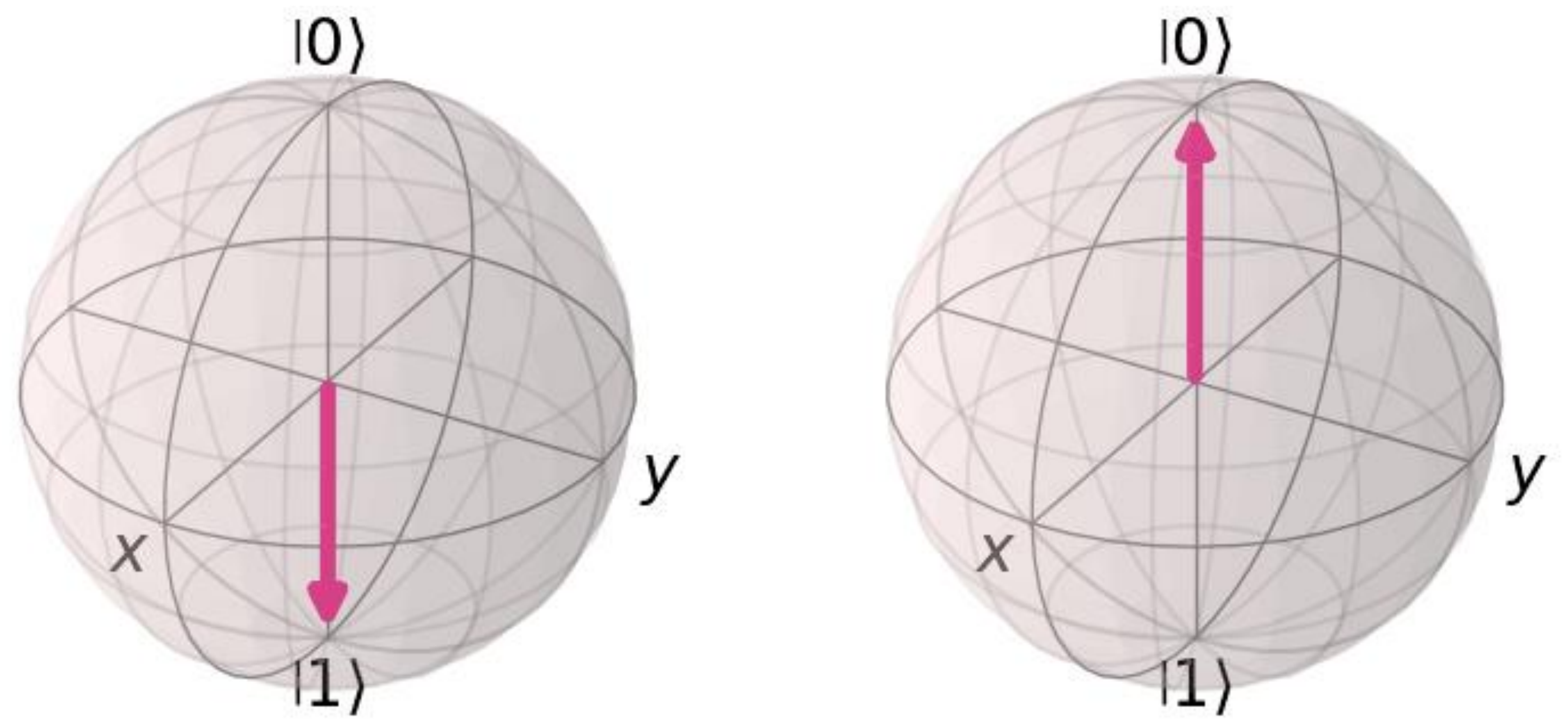
CX/CNOT Gate: Quantum analogue to the classical XOR gate



INPUT		OUTPUT	
x	y	x	y
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0



INPUT		OUTPUT	
q_0	q_1	q_0	q_1
$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$

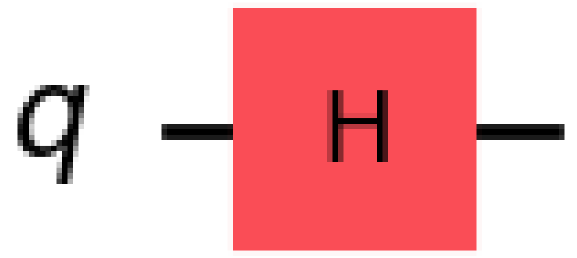


Crash course on quantum computing

Quantum gates

They manipulate the state of the qubit (resulting in rotations along the Bloch sphere)

Hadamard gate: Puts the qubit in an equal superposition of the states $|0\rangle$ and $|1\rangle$



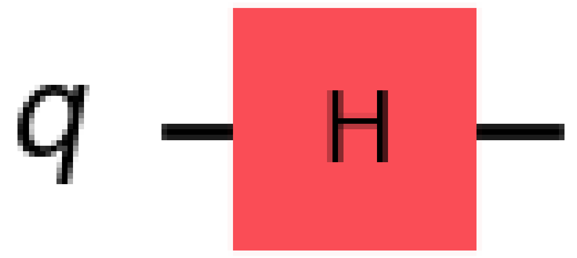
INPUT	OUTPUT
$\|0\rangle$	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$
$\|1\rangle$	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$

Crash course on quantum computing

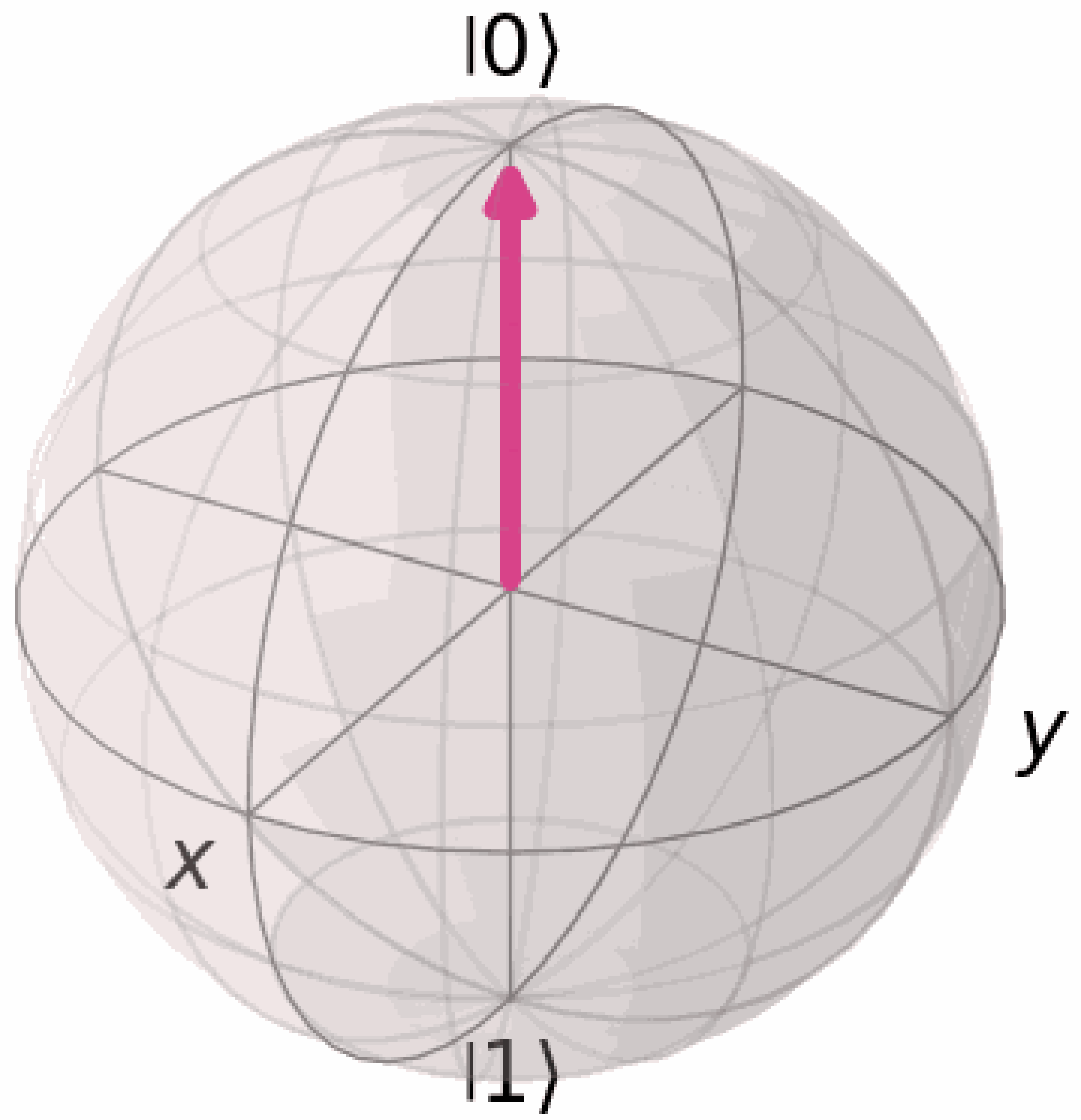
Quantum gates

They manipulate the state of the qubit (resulting in rotations along the Bloch sphere)

Hadamard gate: Puts the qubit in an equal superposition of the states $|0\rangle$ and $|1\rangle$



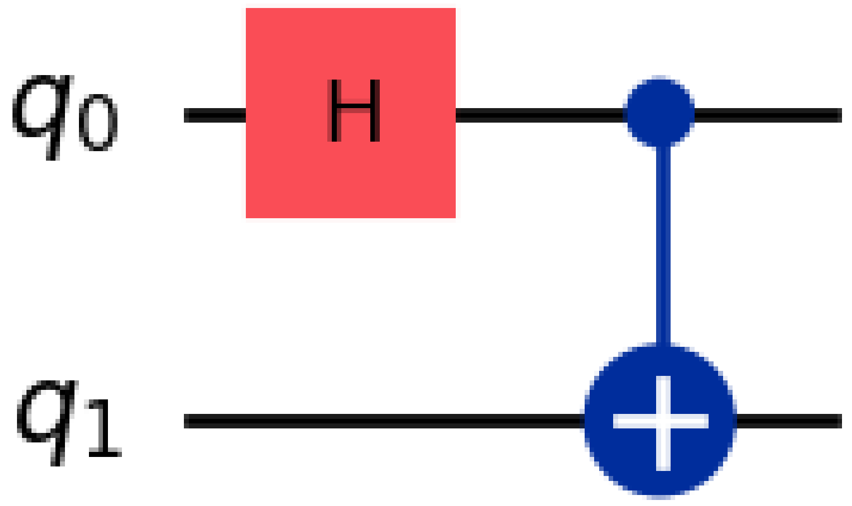
INPUT	OUTPUT
$\|0\rangle$	$\frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$
$\|1\rangle$	$\frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$



Crash course on quantum computing

Entanglement

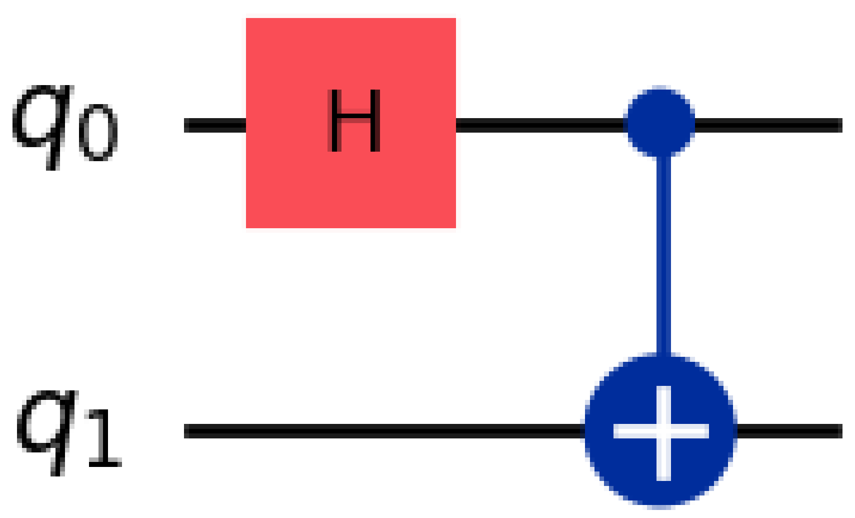
Quantum property by which the state of two qubits become interconnected (i.e., the state of one of them cannot be described independently of the other)



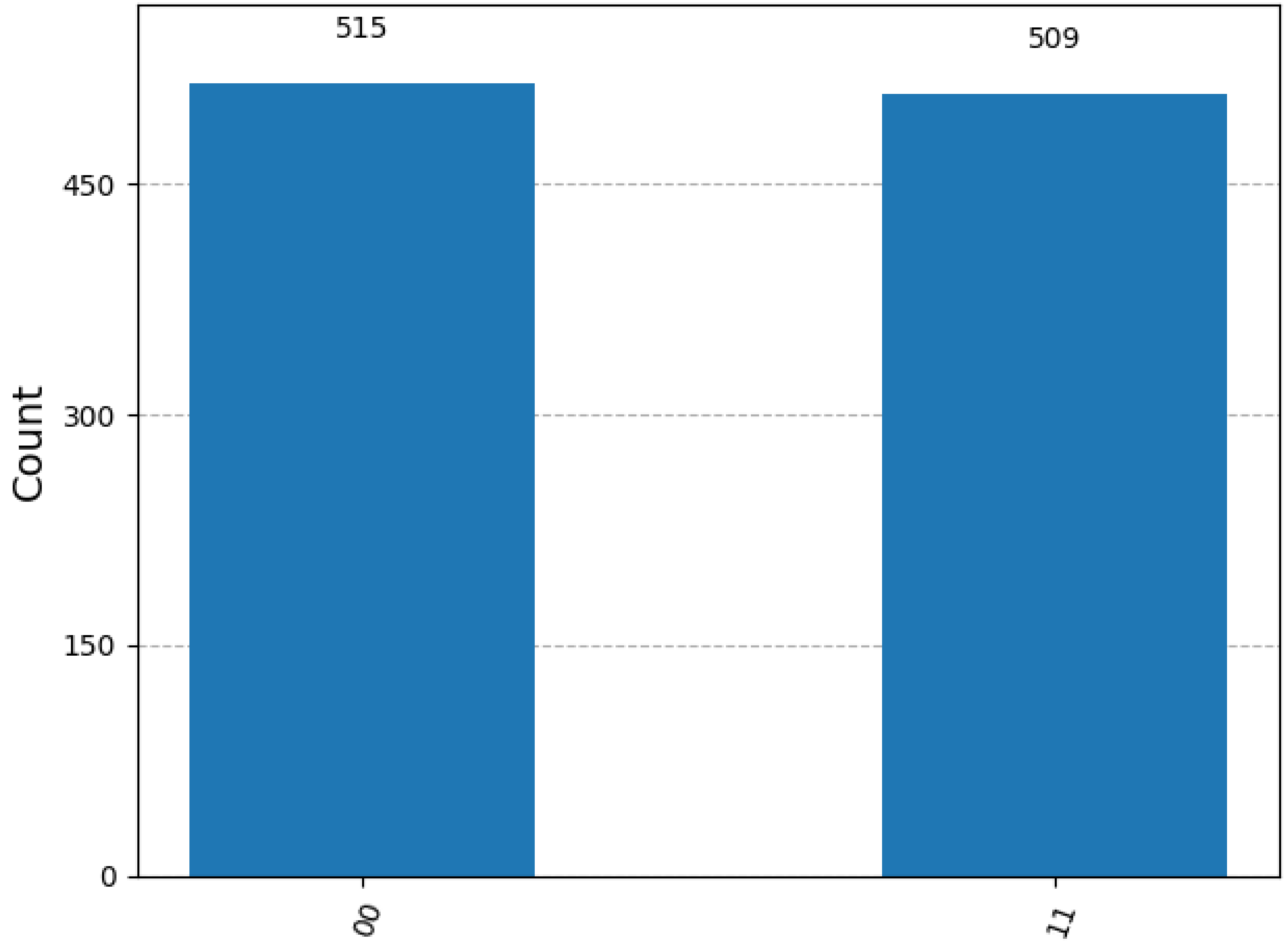
INPUT	OUTPUT
$\|00\rangle$	$\frac{1}{\sqrt{2}}(\|00\rangle + \|11\rangle)$

Crash course on quantum computing

Entanglement Quantum property by which the state of two qubits become interconnected (i.e., the state of one of them cannot be described independently of the other)



INPUT	OUTPUT
$\|00\rangle$	$\frac{1}{\sqrt{2}}(\|00\rangle + \|11\rangle)$

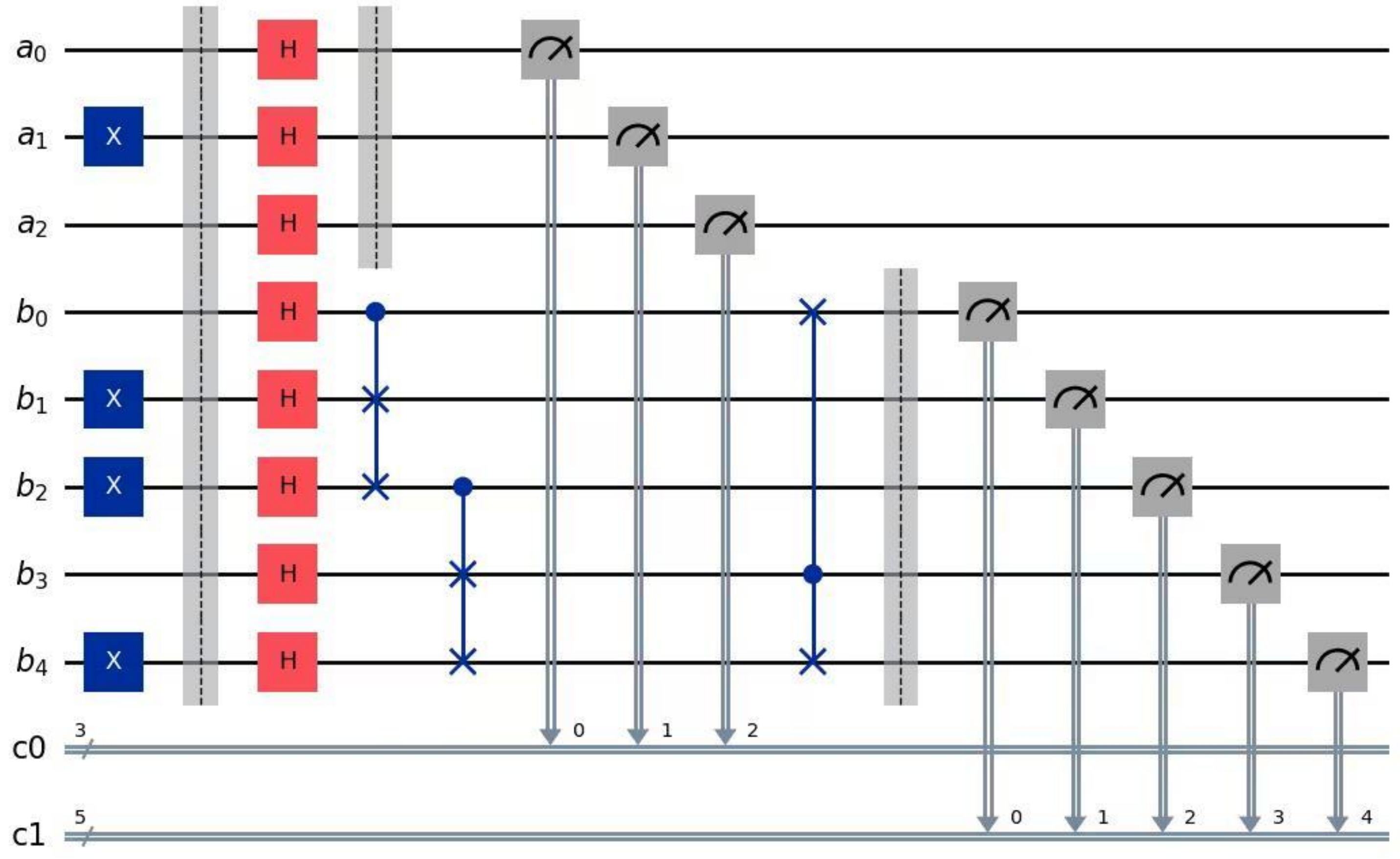


What is Qiskit?

Open-source quantum software for quantum computing and algorithms.

What is Qiskit?

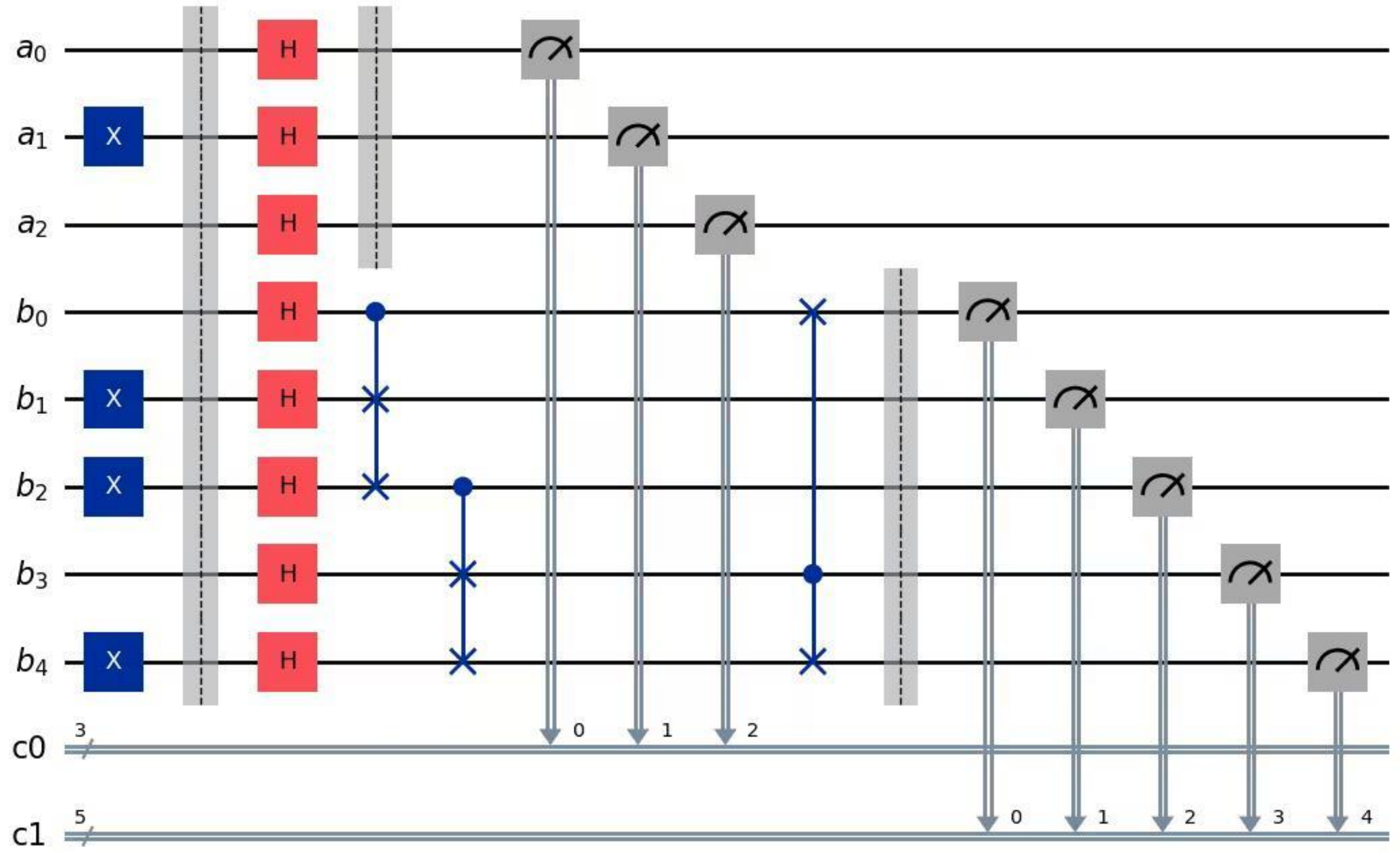
Open-source quantum software for quantum computing and algorithms.



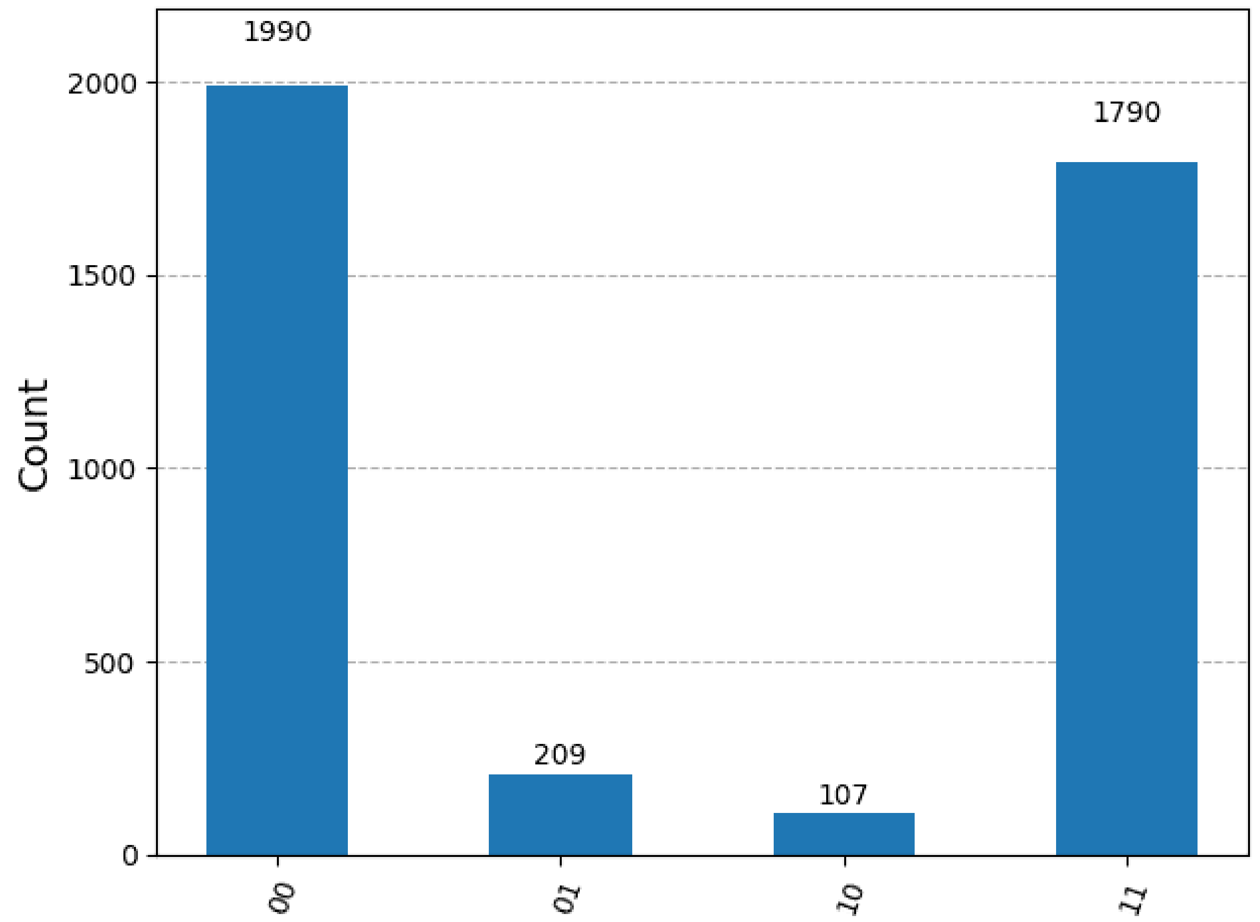
Design quantum algorithms

What is Qiskit?

Open-source quantum software for quantum computing and algorithms.



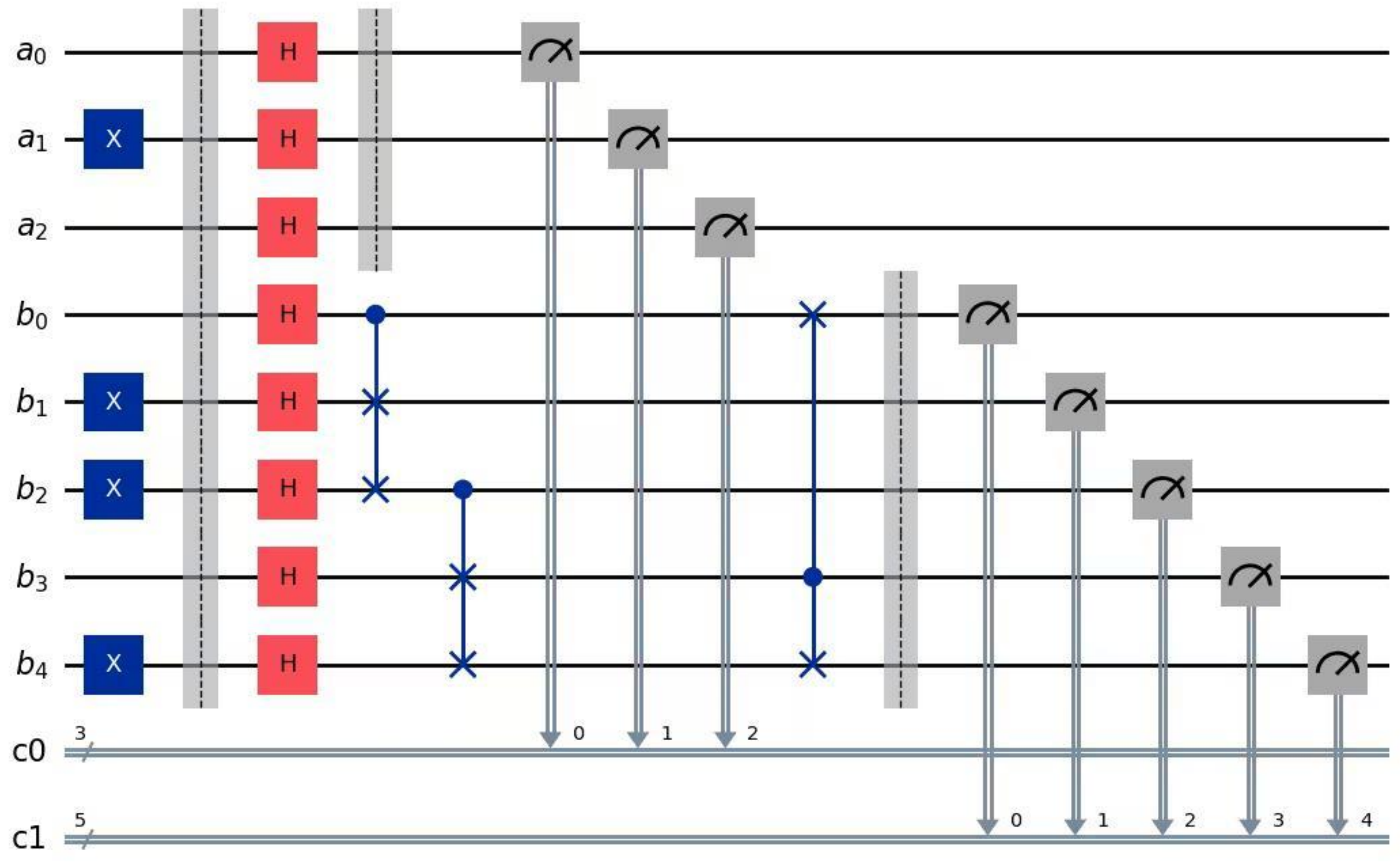
Design quantum algorithms

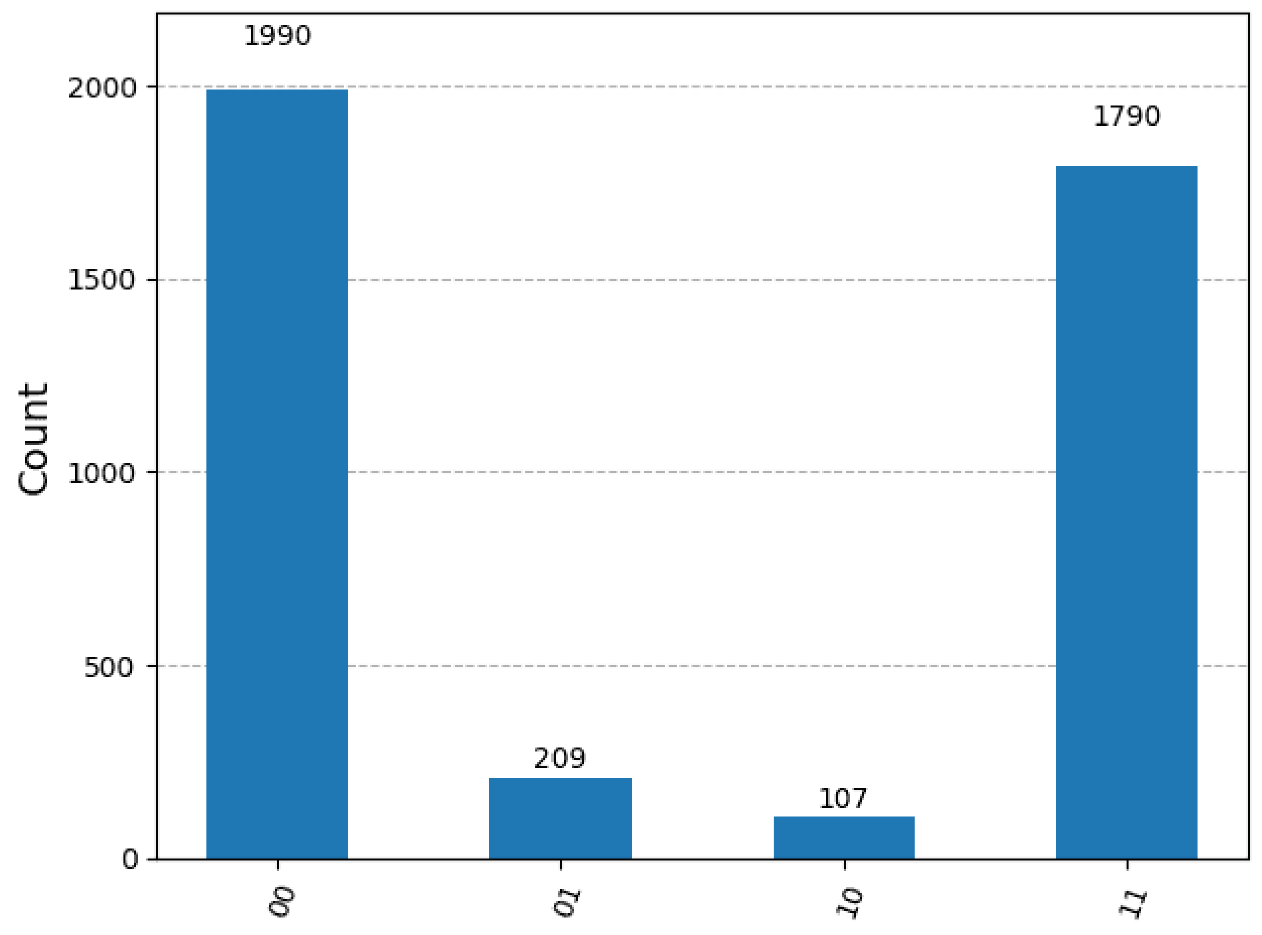


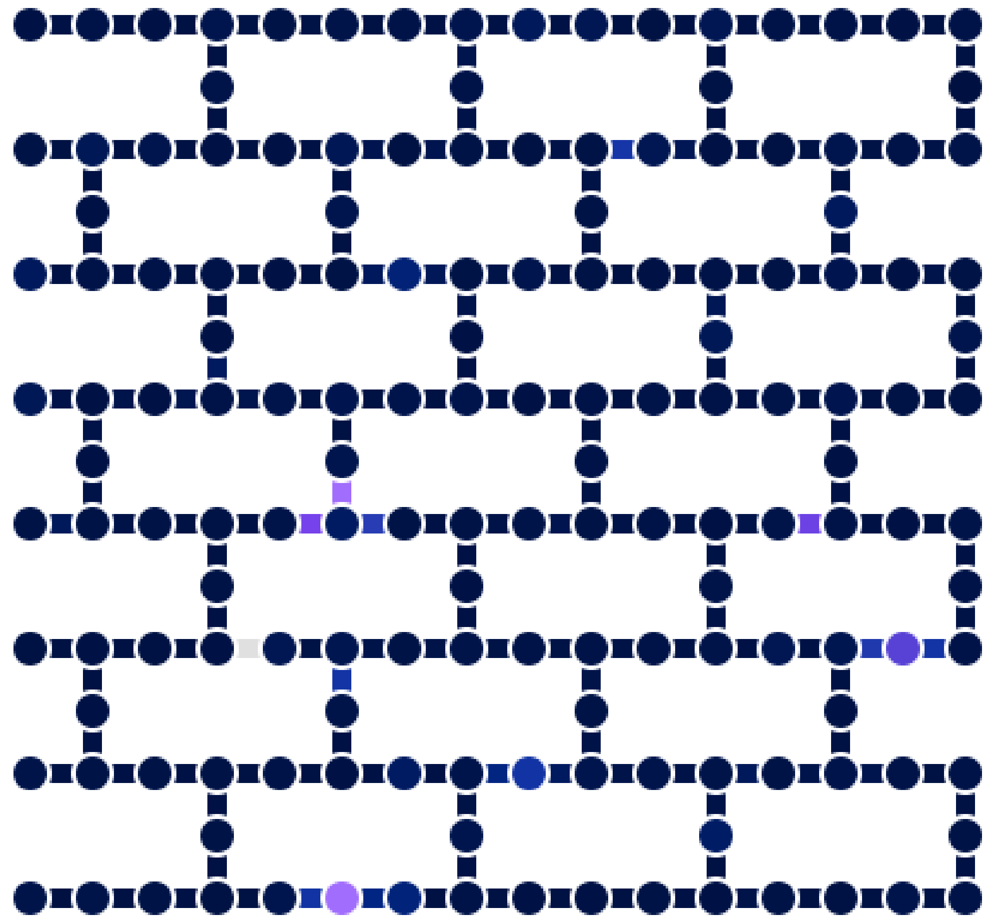
Visualize results

What is **Qiskit?**

Open-source **quantum software** for quantum computing and algorithms.







Design quantum algorithms

Visualize results

Run in real quantum processing units (QPUs)

IBM BasQ Basque Quantum

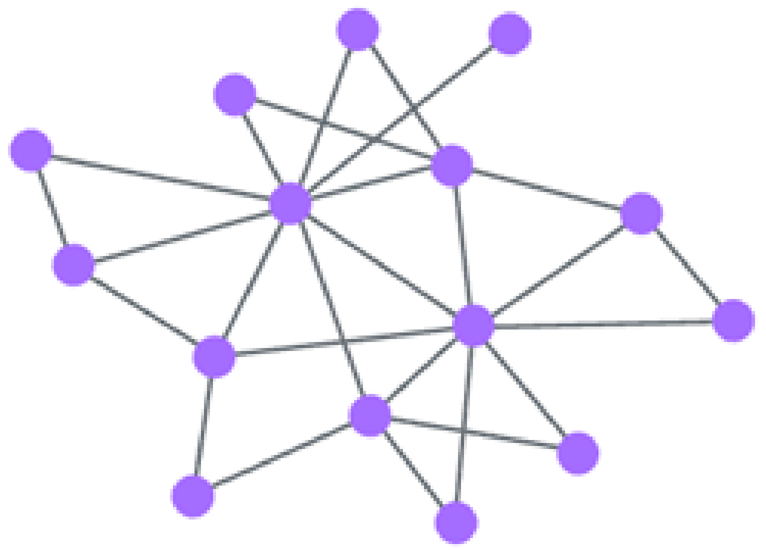
Century of Quantum

QISKIT 2025 2025 FALL FEST

Qiskit pattern

Step 1

Map classical inputs to a quantum problem



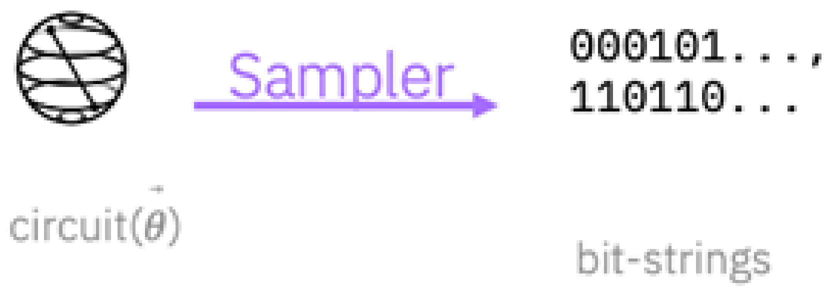
Step 2

Optimize problem for quantum execution.

```
PassManager([UnitarySynthesis(),
             BasisTranslator(),
             EnlargeWithAncilla(),
             AISwap(),
             Collect1qRuns(),
             Optimize1qGates(),
             Collect2qBlocks(),
             ConsolidateBlocks()])
```

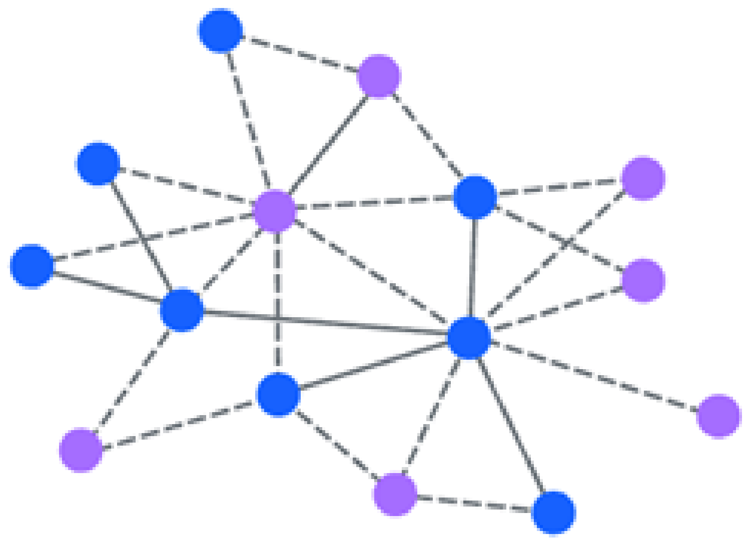
Step 3

Execute using Qiskit Runtime Primitives.



Step 4

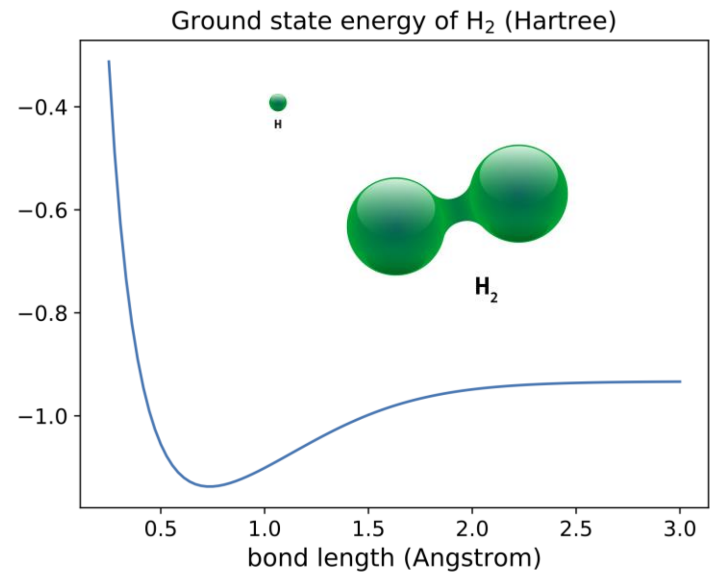
Post-process, return result in classical format.

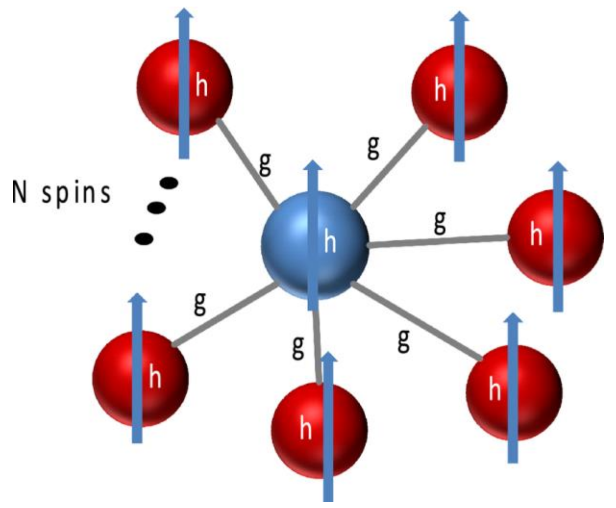


Map the problem

Involves **translating** the problem into the quantum computer.

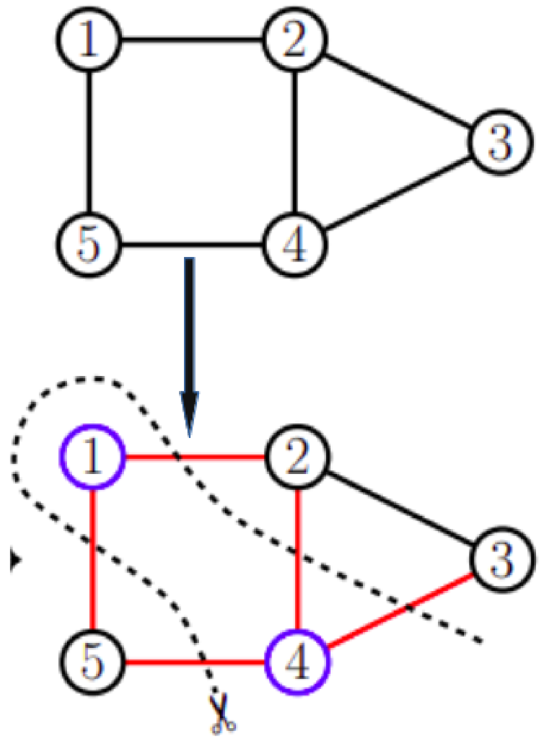
Quantum chemisty





Material physics

Optimization problems

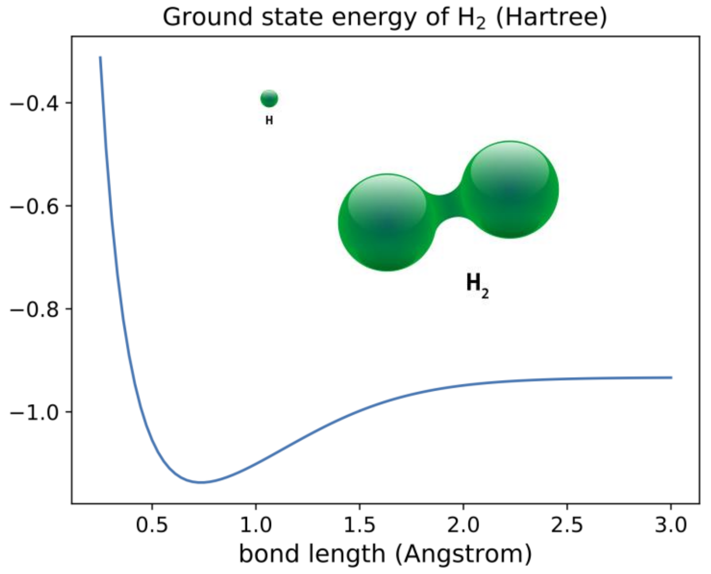


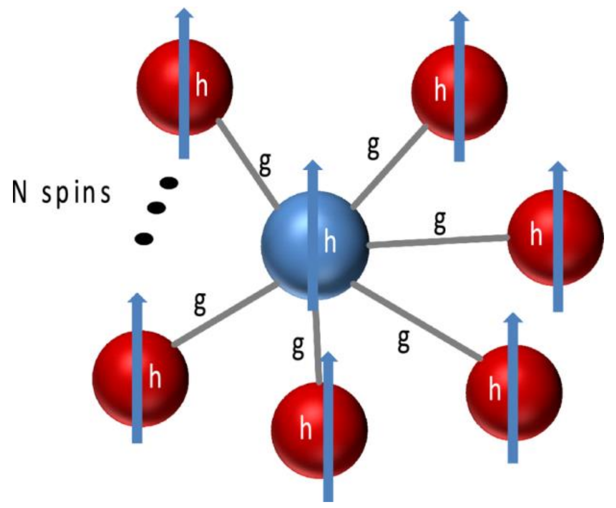
Identify the problem

Map the problem

Involves **translating** the problem into the quantum computer.

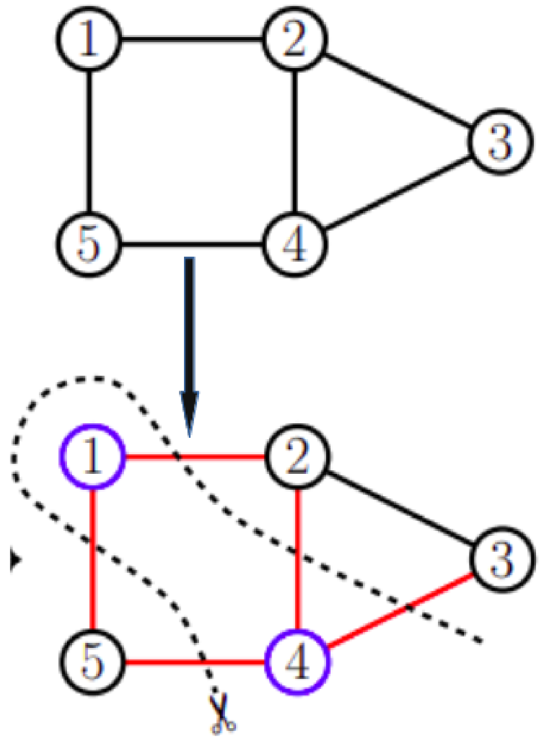
Quantum chemisty

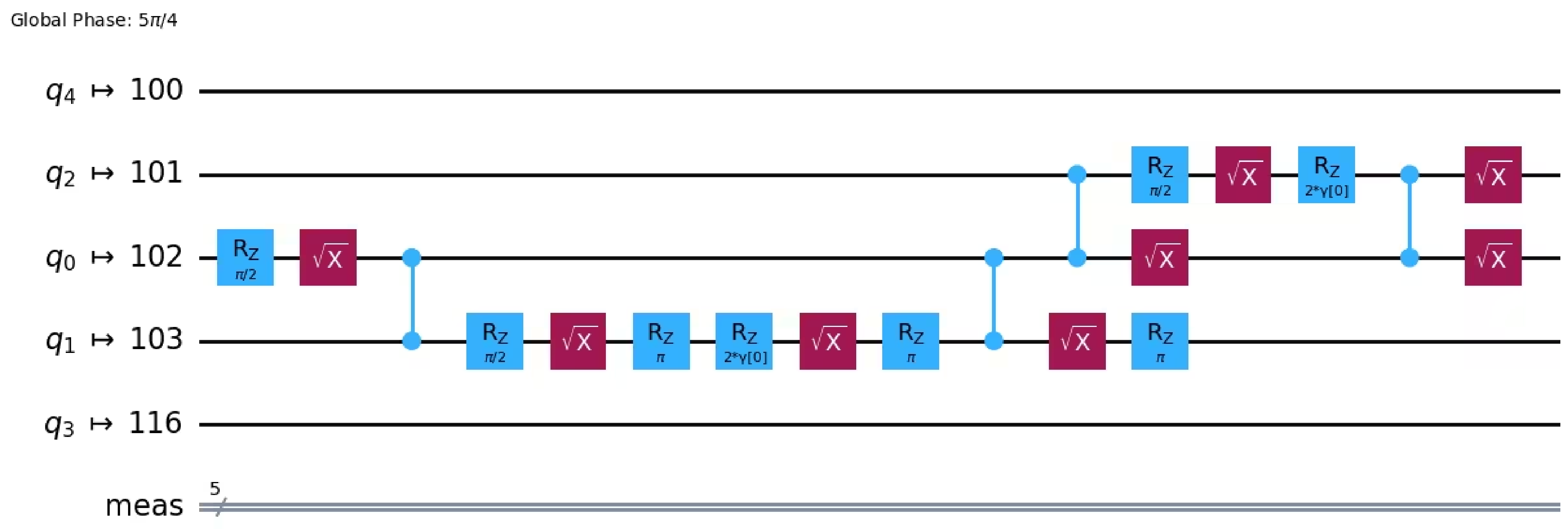




Material physics

Optimization problems





Identify the problem

Map it to qubits

Map the problem

Involves translating the problem into the quantum computer.

Map the problem

Involves **translating** the problem into the quantum computer.

```
from qiskit import QuantumCircuit
from qiskit.circuit.library import HGate, MCXGate

mcx_gate = MCXGate(3)
hadamard_gate = HGate()

qc = QuantumCircuit(4)
qc.append(hadamard_gate, [0])
qc.append(mcx_gate, [0, 1, 2, 3])
qc.draw("mpl")
```

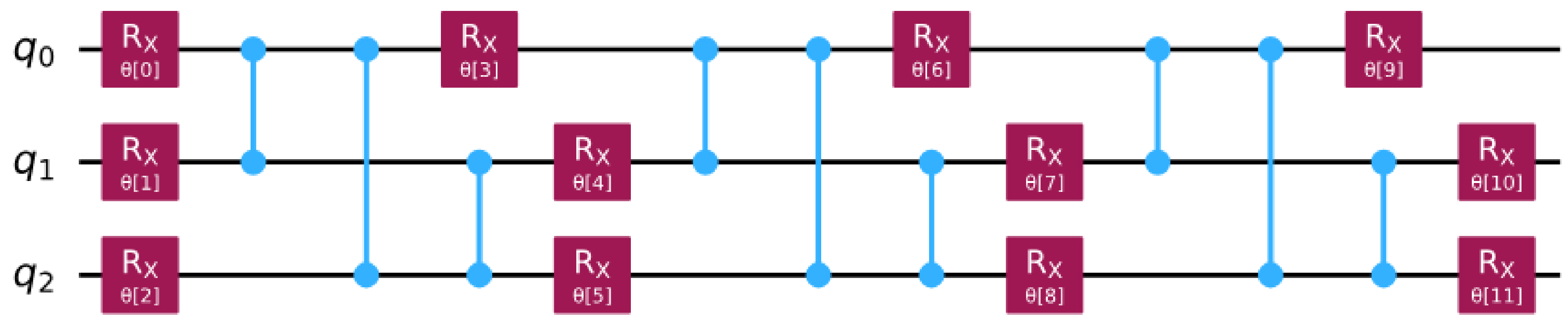
Construct your circuit

Map the problem

Involves **translating** the problem into the quantum computer.

```python
from qiskit import QuantumCircuit
from qiskit.circuit.library import HGate, MCXGate

mcx_gate = MCXGate(3)
hadamard_gate = HGate()

qc = QuantumCircuit(4)
qc.append(hadamard_gate, [0])
qc.append(mcx_gate, [0, 1, 2, 3])
qc.draw("mpl")
```

Construct your circuit



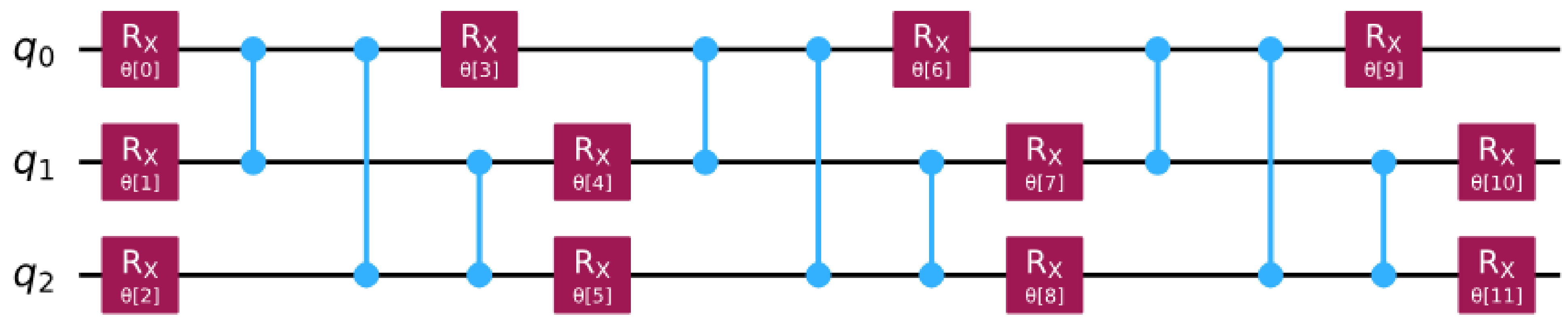
Visualize it

Map the problem

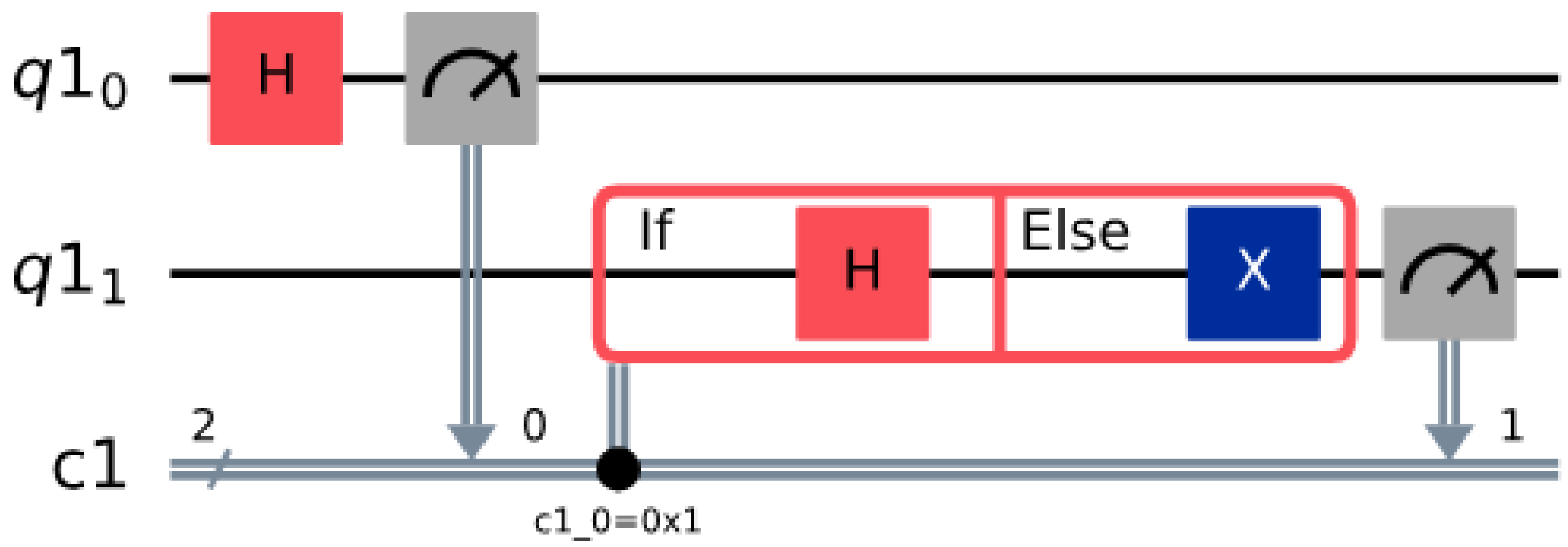
Involves **translating** the problem into the quantum computer.

```python
from qiskit import QuantumCircuit
from qiskit.circuit.library import HGate, MCXGate

mcx_gate = MCXGate(3)
hadamard_gate = HGate()

qc = QuantumCircuit(4)
qc.append(hadamard_gate, [0])
qc.append(mcx_gate, [0, 1, 2, 3])
qc.draw("mpl")
```

Construct your circuit



Visualize it



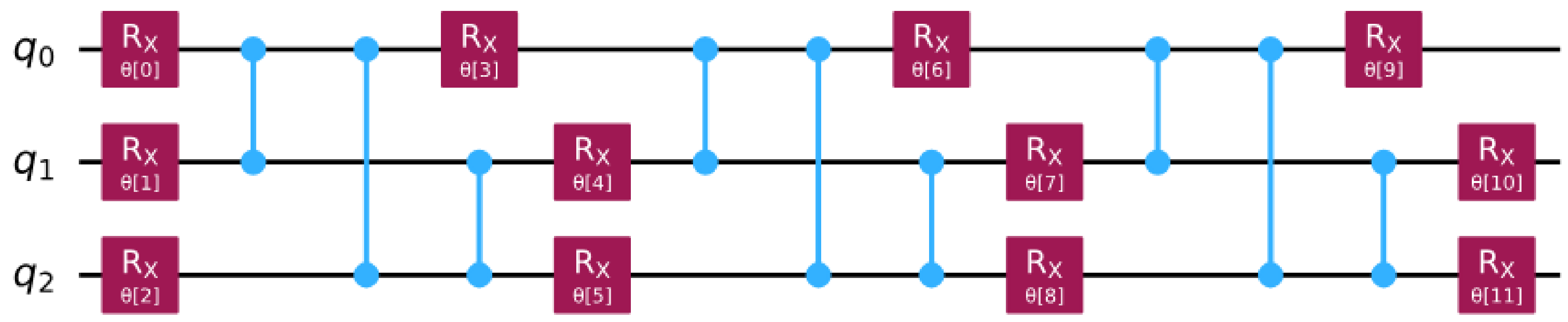
Add **classic logic** in between

Map the problem

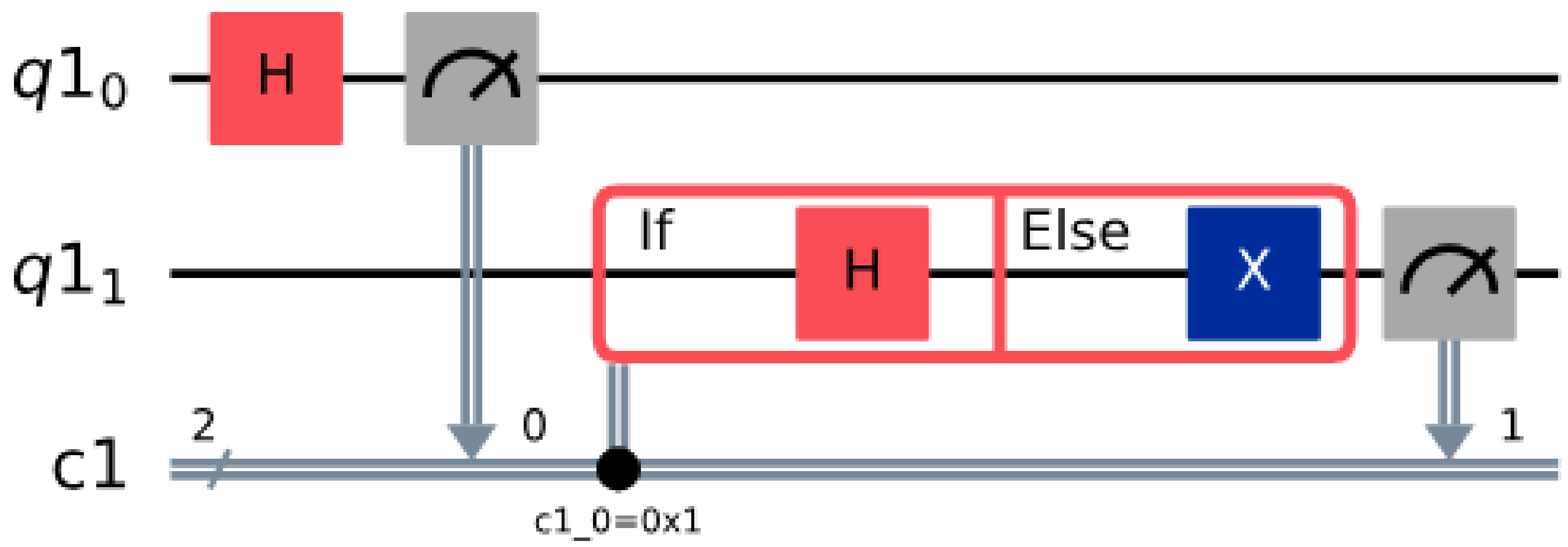
Involves **translating** the problem into the quantum computer.

```python
from qiskit import QuantumCircuit
from qiskit.circuit.library import HGate, MCXGate

mcx_gate = MCXGate(3)
hadamard_gate = HGate()

qc = QuantumCircuit(4)
qc.append(hadamard_gate, [0])
qc.append(mcx_gate, [0, 1, 2, 3])
qc.draw("mpl")
```

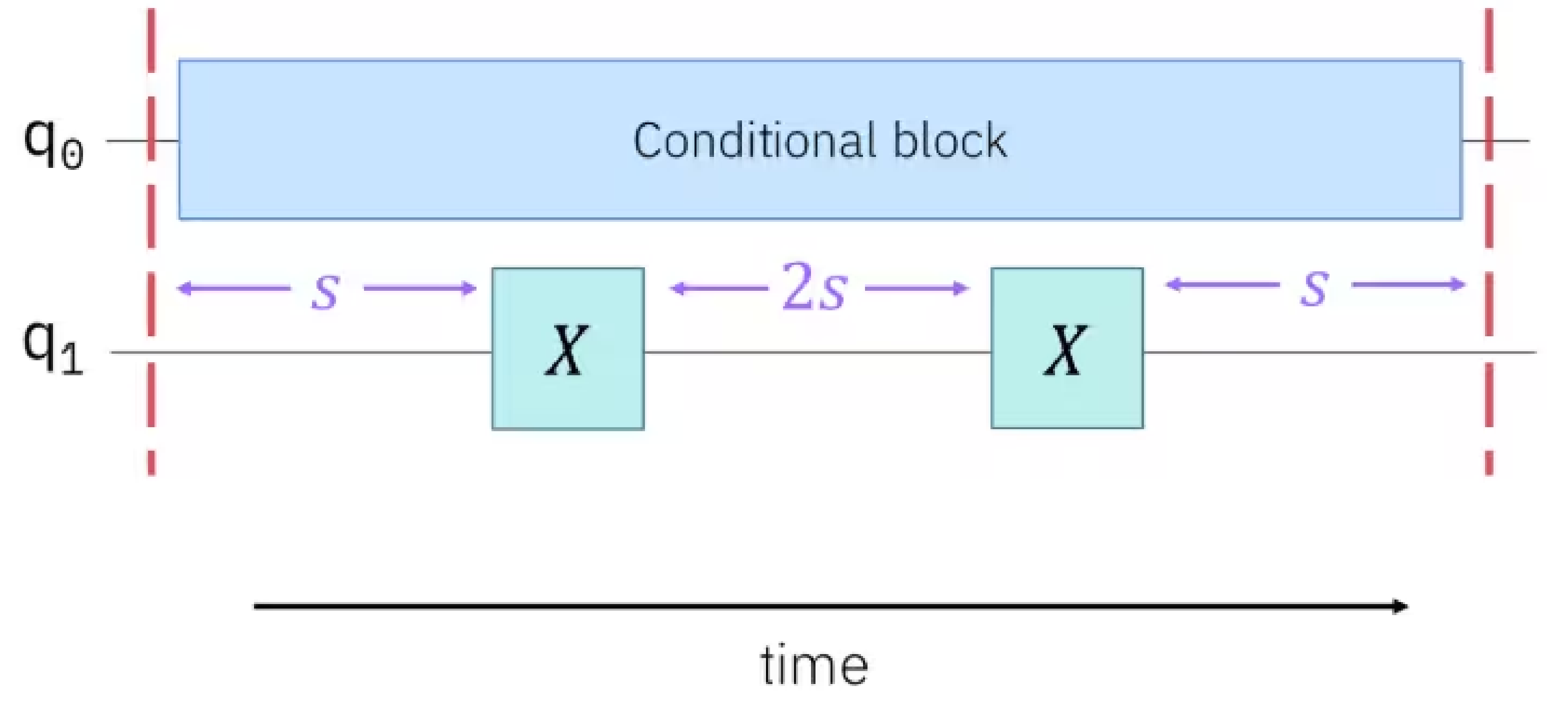
Construct your circuit



Visualize it



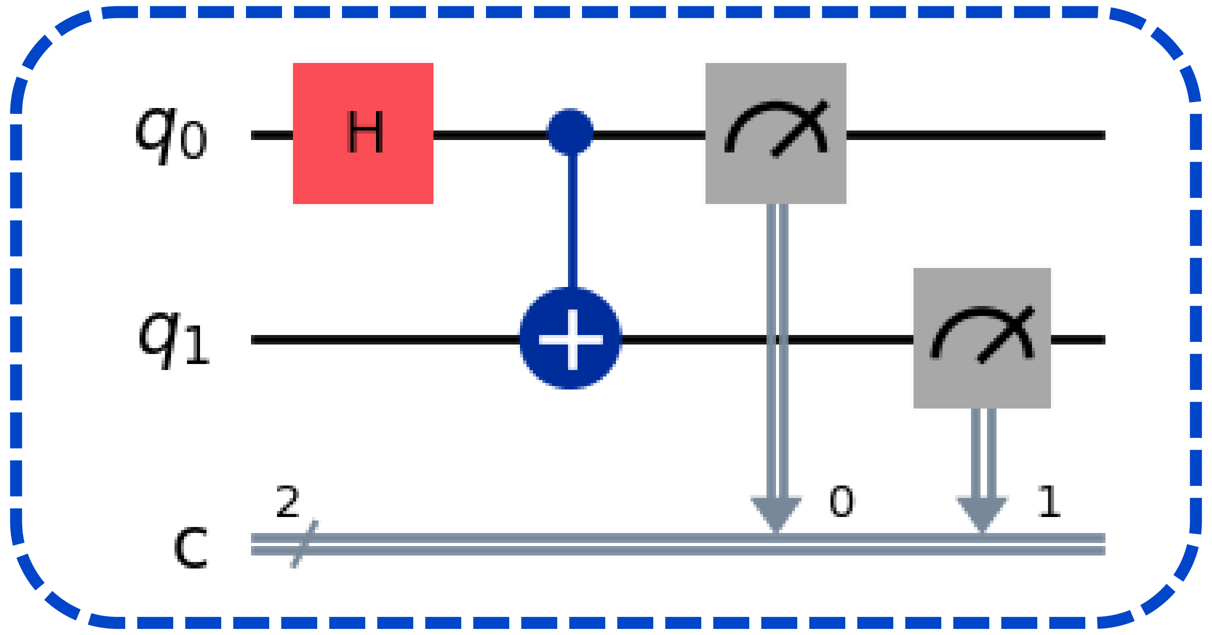
Add **classic logic** in between



Delay operations with stretch

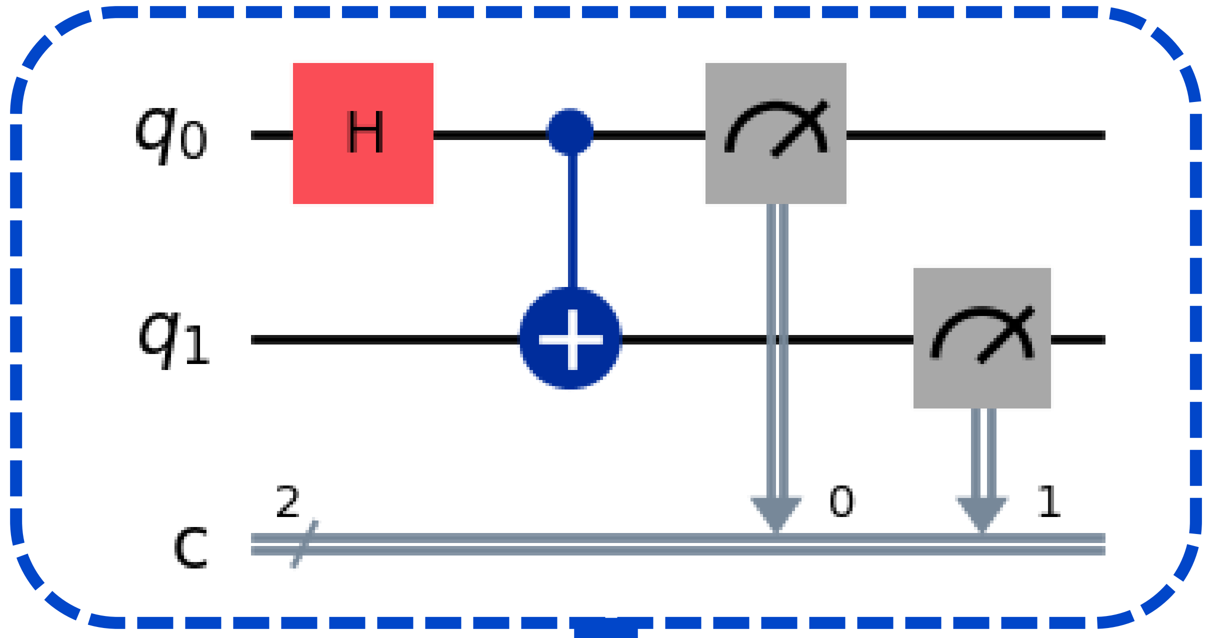
Optimize for hardware

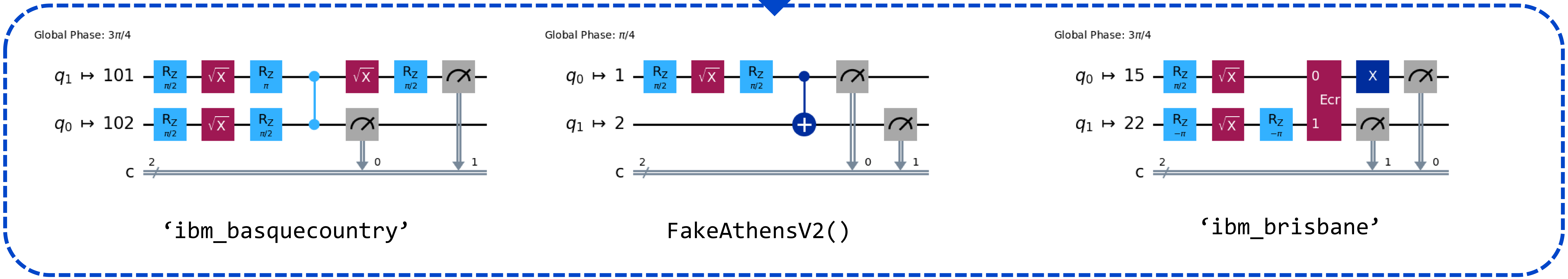
Transform the virtual circuit into a hardware-adapted circuit that matches the topology and constraints of a specific device and optimize it



Optimize for hardware

Transform the virtual circuit into a hardware-adapted circuit that matches the topology and constraints of a specific device and optimize it





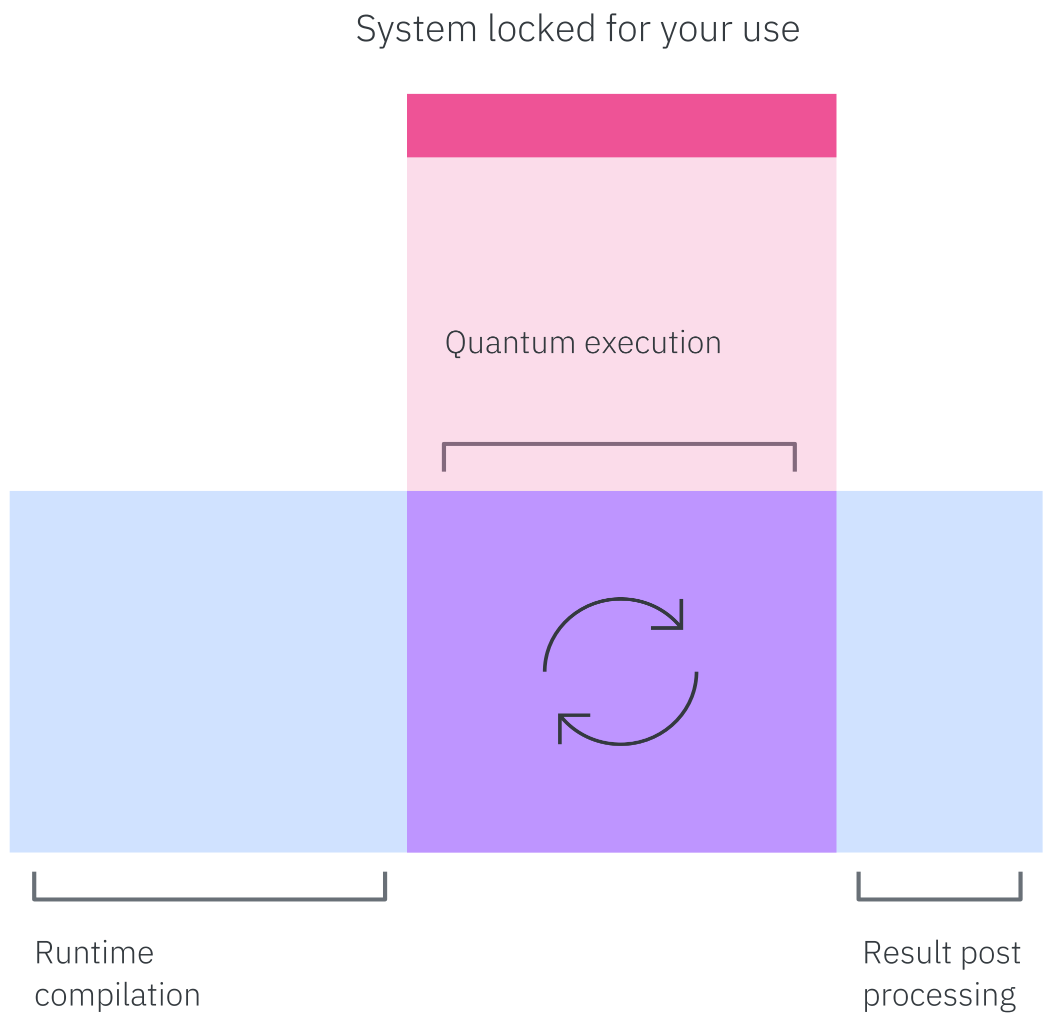
Execute on hardware

Run the designed circuits on hardware and produce outputs of the computation.

Execute on hardware

Run the designed circuits on hardware and produce outputs of the computation.

Job:
A single primitive request that contains all the context for executing your workload



Execute on hardware

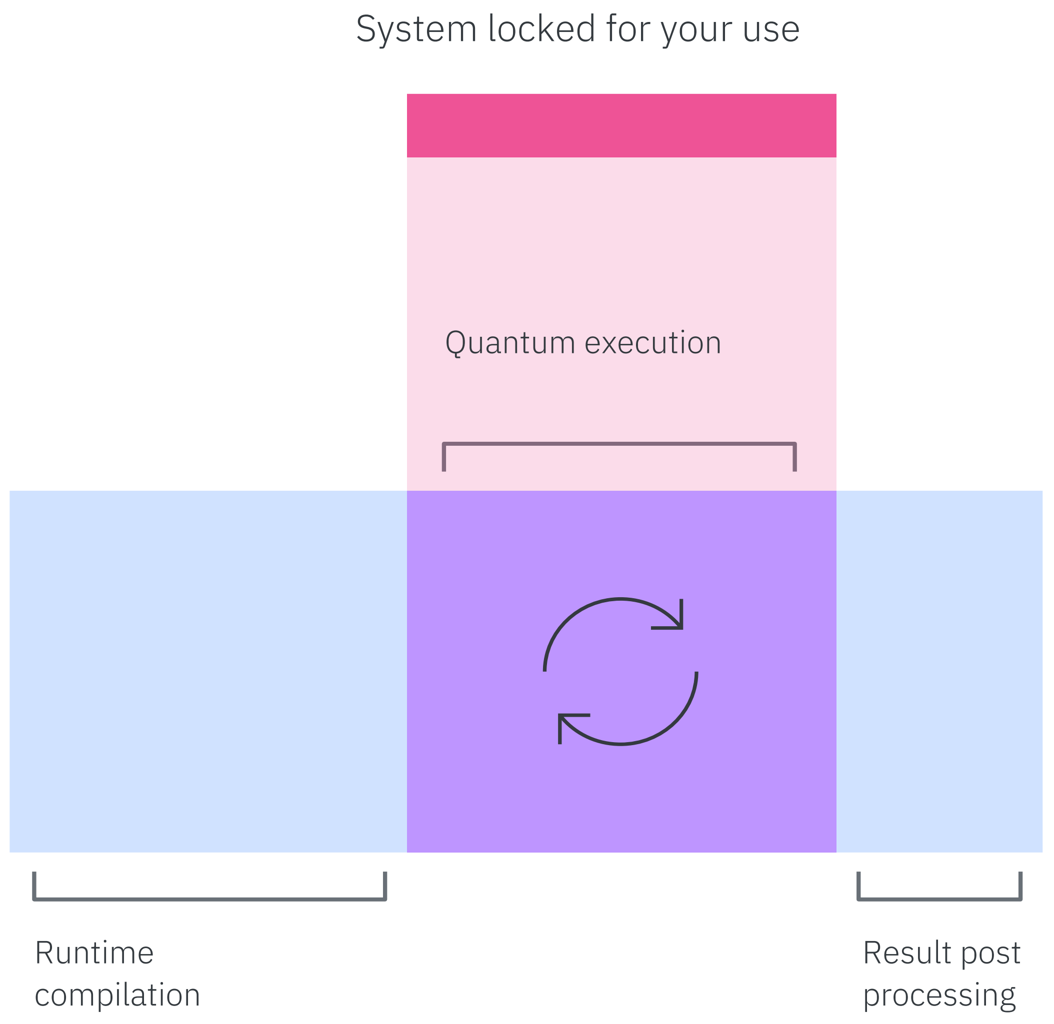
Run the designed circuits on hardware and produce outputs of the computation.

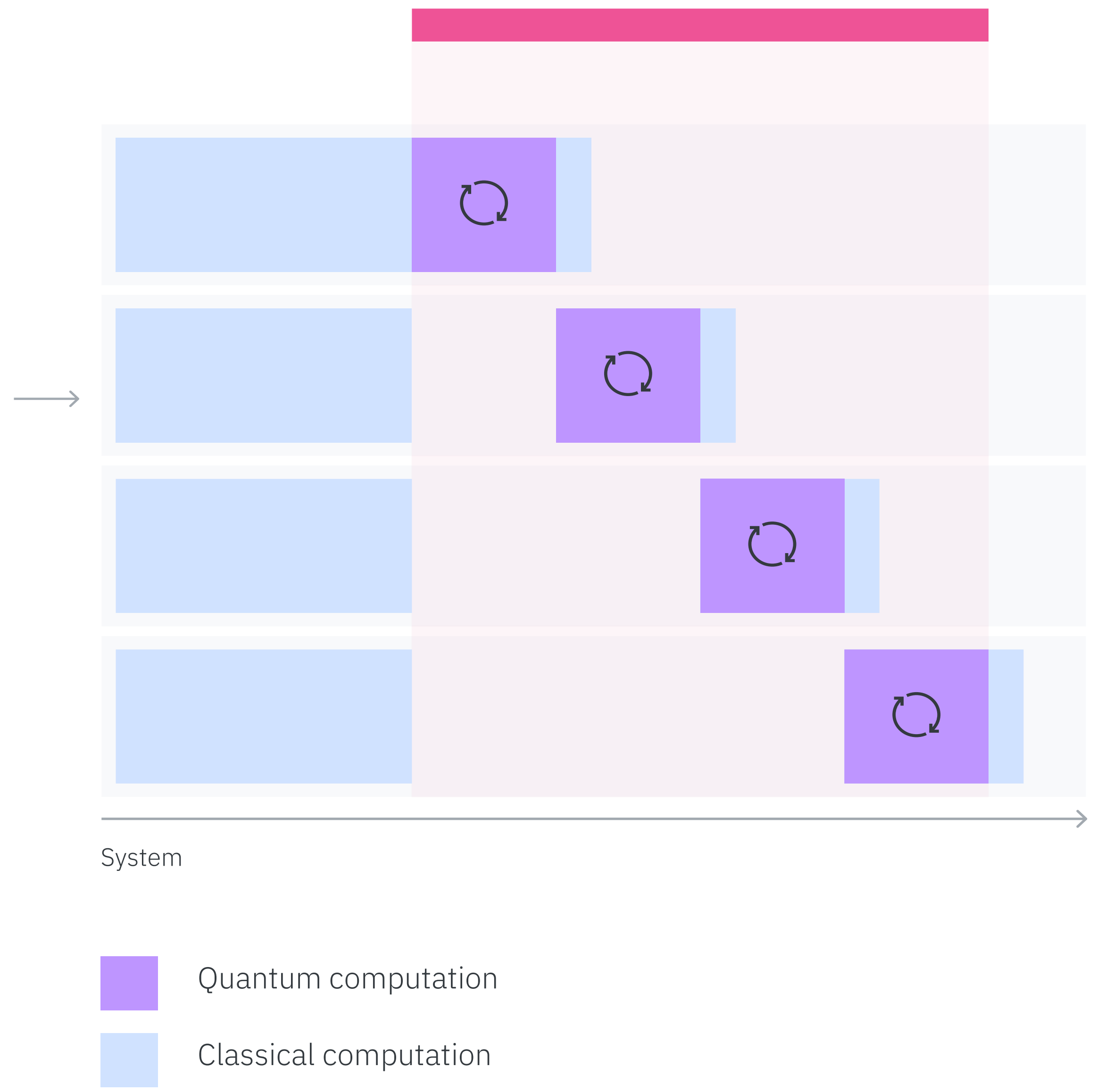
Job:

A single primitive request that contains all the context for executing your workload

Batch:

A multi-job manager for efficiently running an experiment that is comprised of bundle of independent jobs.



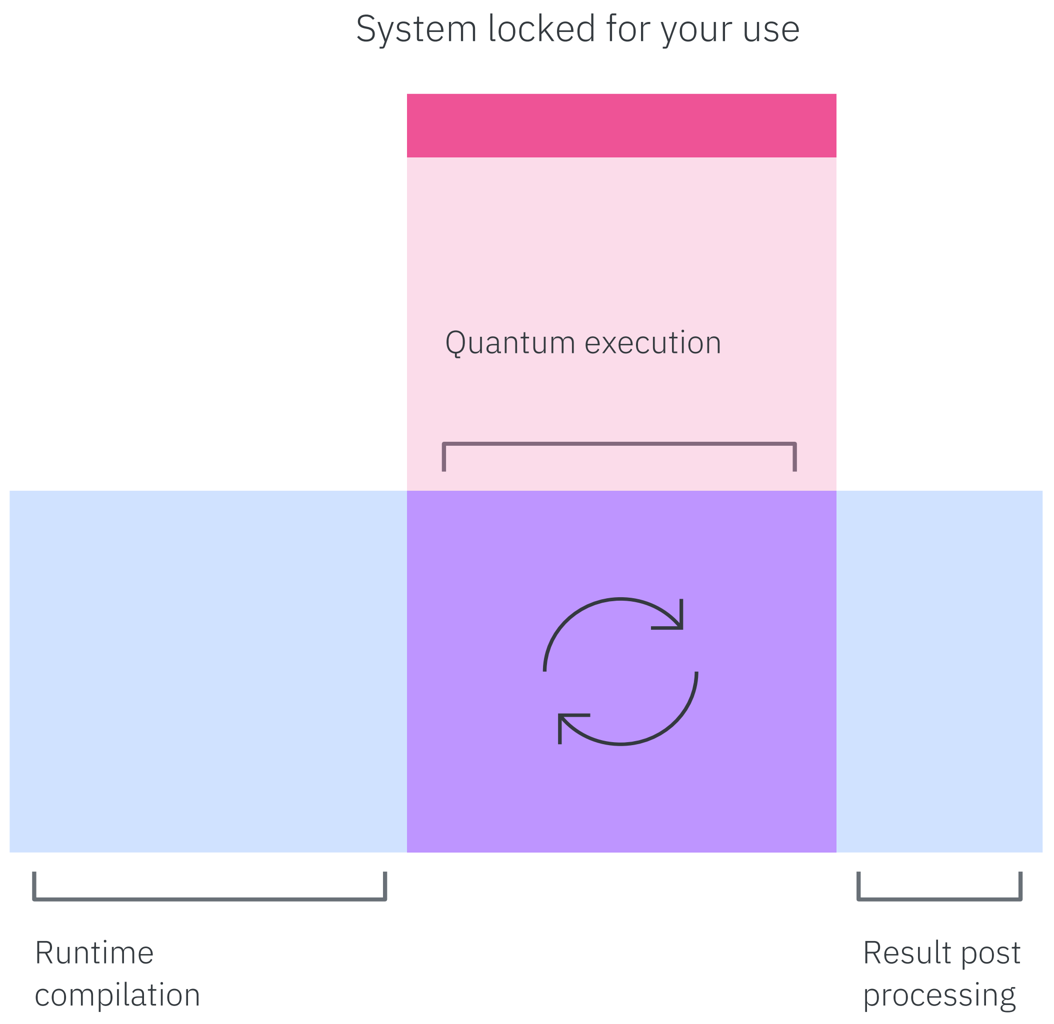


Execute on hardware

Run the designed circuits on hardware and produce outputs of the computation.

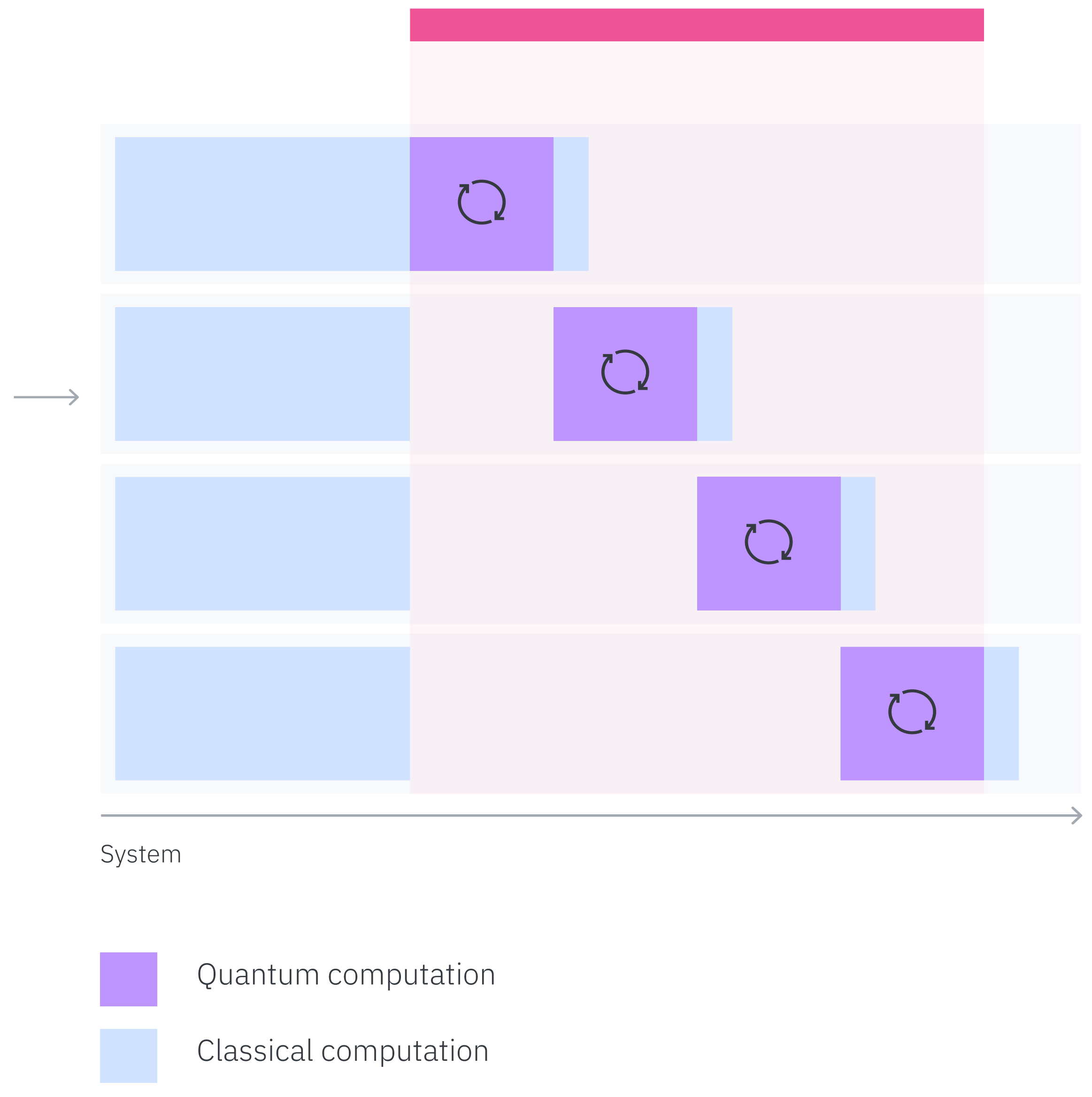
Job:

A single primitive request that contains all the context for executing your workload



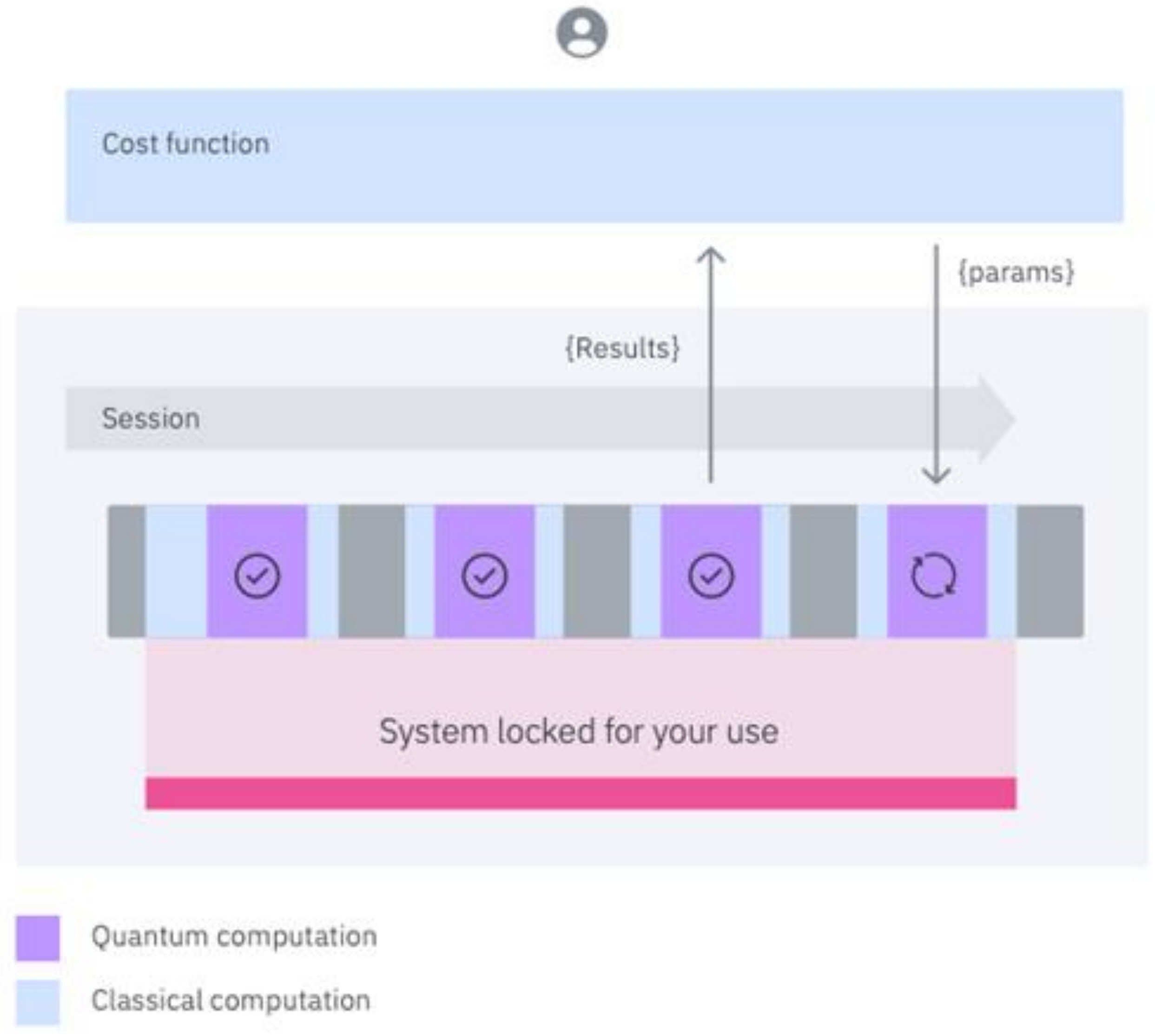
Batch:

A multi-job manager for efficiently running an experiment that is comprised of bundle of independent jobs.



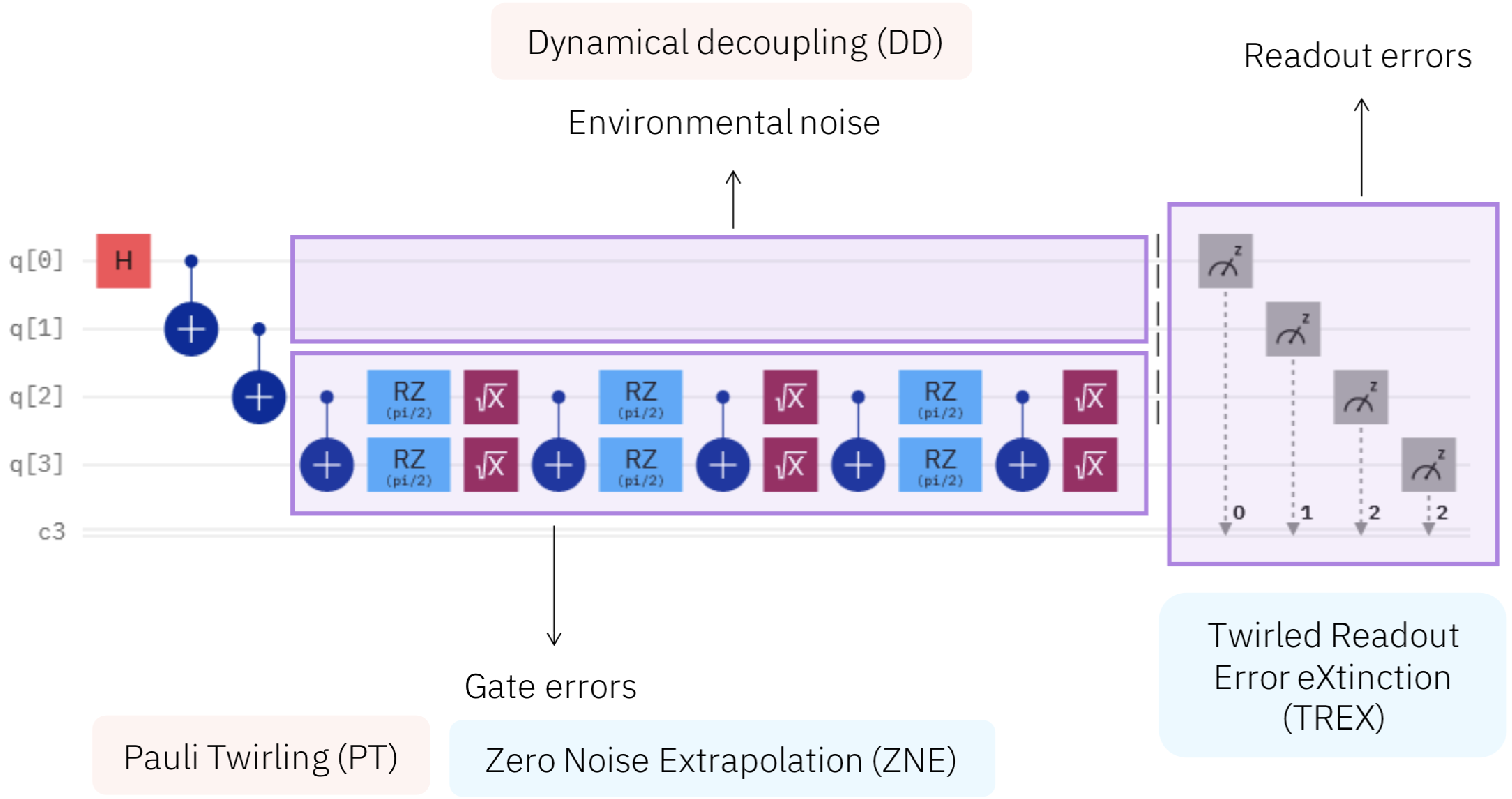
Session:

A dedicated window for running multi-job workload. Allows users to experiment with variational workloads in a more predictable way.



Execute on hardware

User can toggle different error mitigation techniques to deal with the noise that the device introduces to the computation

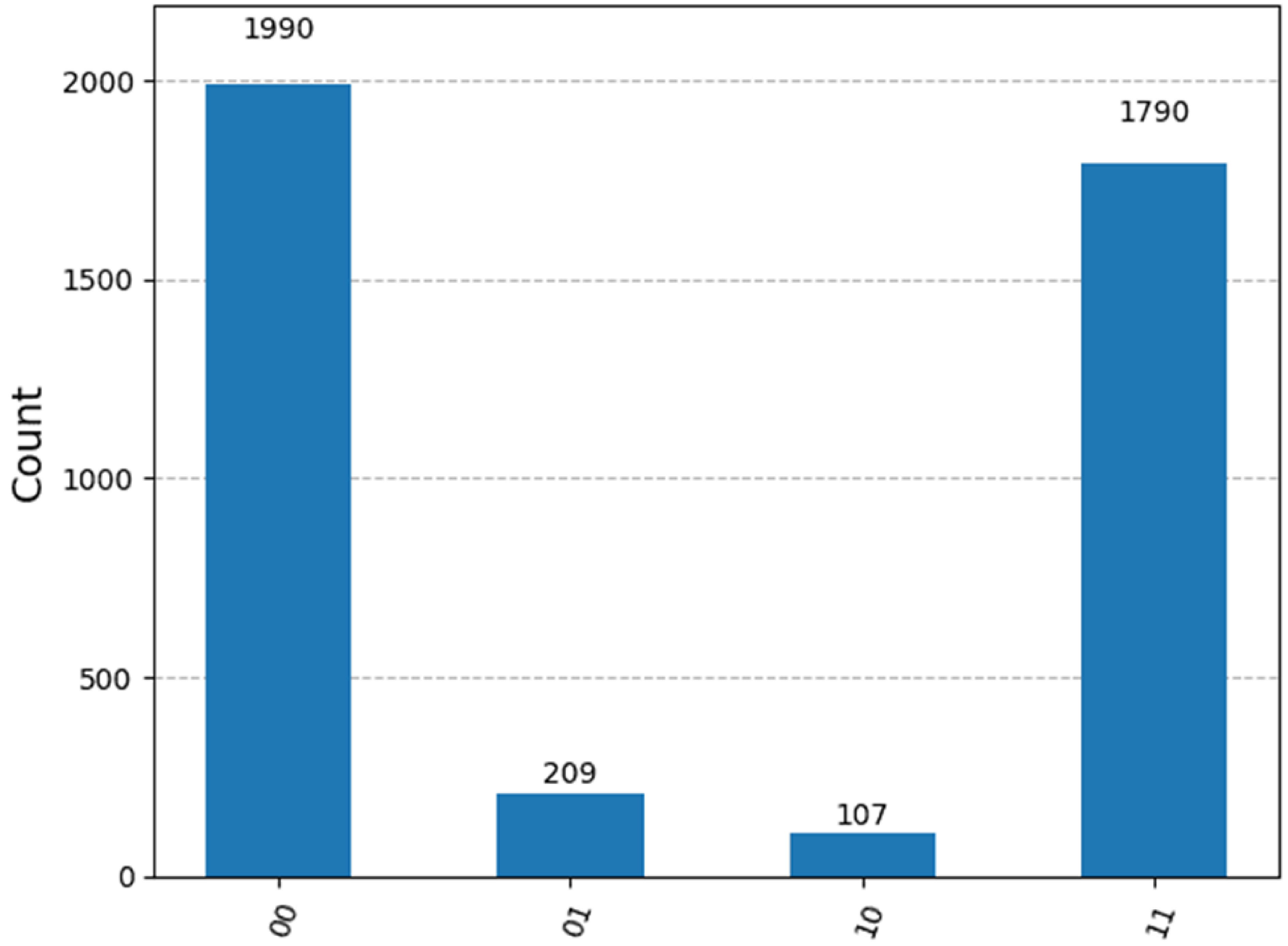


Post-process results

Combine the outputs of the hardware execution to obtain the target observables.

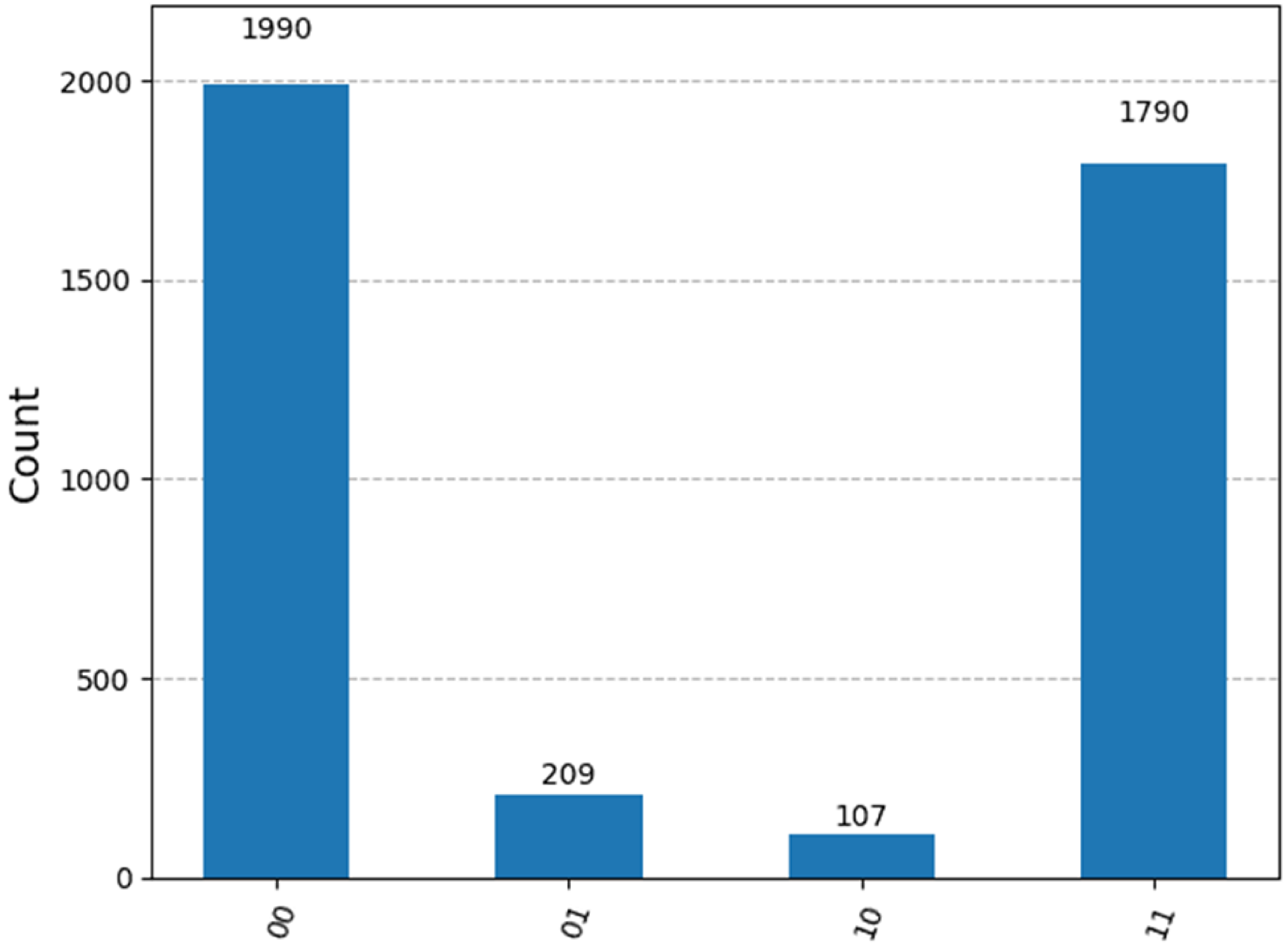
Post-process results

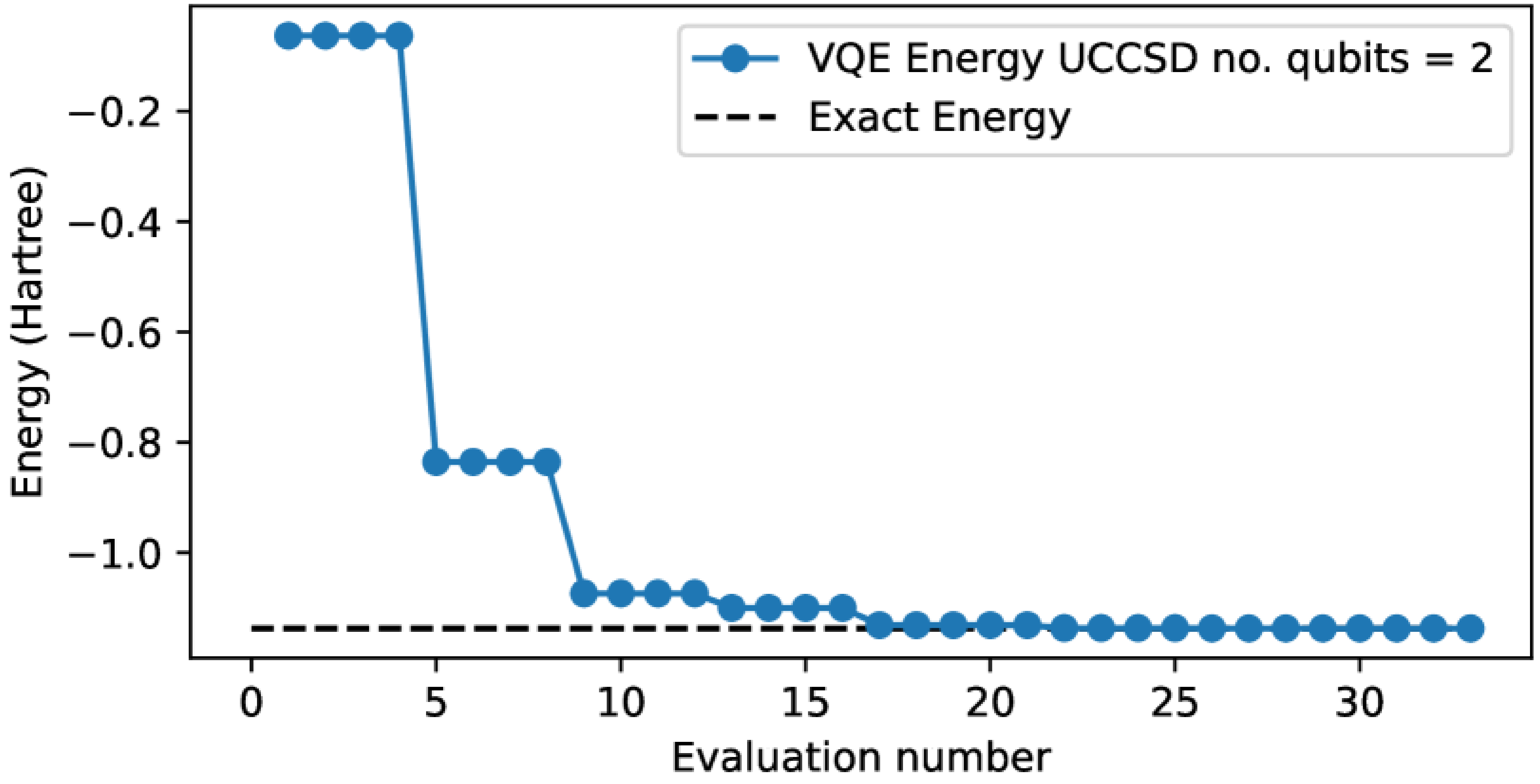
Combine the outputs of the hardware execution to obtain the target observables.



Post-process results

Combine the outputs of the hardware execution to obtain the target observables.





Example: Generate and run a Bell state

First of all, make sure you have Qiskit installed!

```
pip install qiskit
pip install qiskit[visualization]
pip install qiskit_aer
pip install qiskit_ibm_runtime
```

Example: Generate and run a Bell state

First of all, make sure you have Qiskit installed!

```
pip install qiskit
pip install qiskit[visualization]
pip install qiskit_aer
pip install qiskit_ibm_runtime
```

Once you installed it, we will import all packages needed

```
from qiskit import QuantumCircuit, QuantumRegister, generate_preset_pass_manager
from qiskit_aer import AerSimulator
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2
```

Example: Generate and run a Bell state

Now we will design the quantum circuit that generates the Bell state $|\phi^+\rangle$

```
qc = QuantumCircuit(2)
qc.h(0)
qc.cx(0, 1)
qc.measure_all()
qc.draw(output = 'mpl')
```

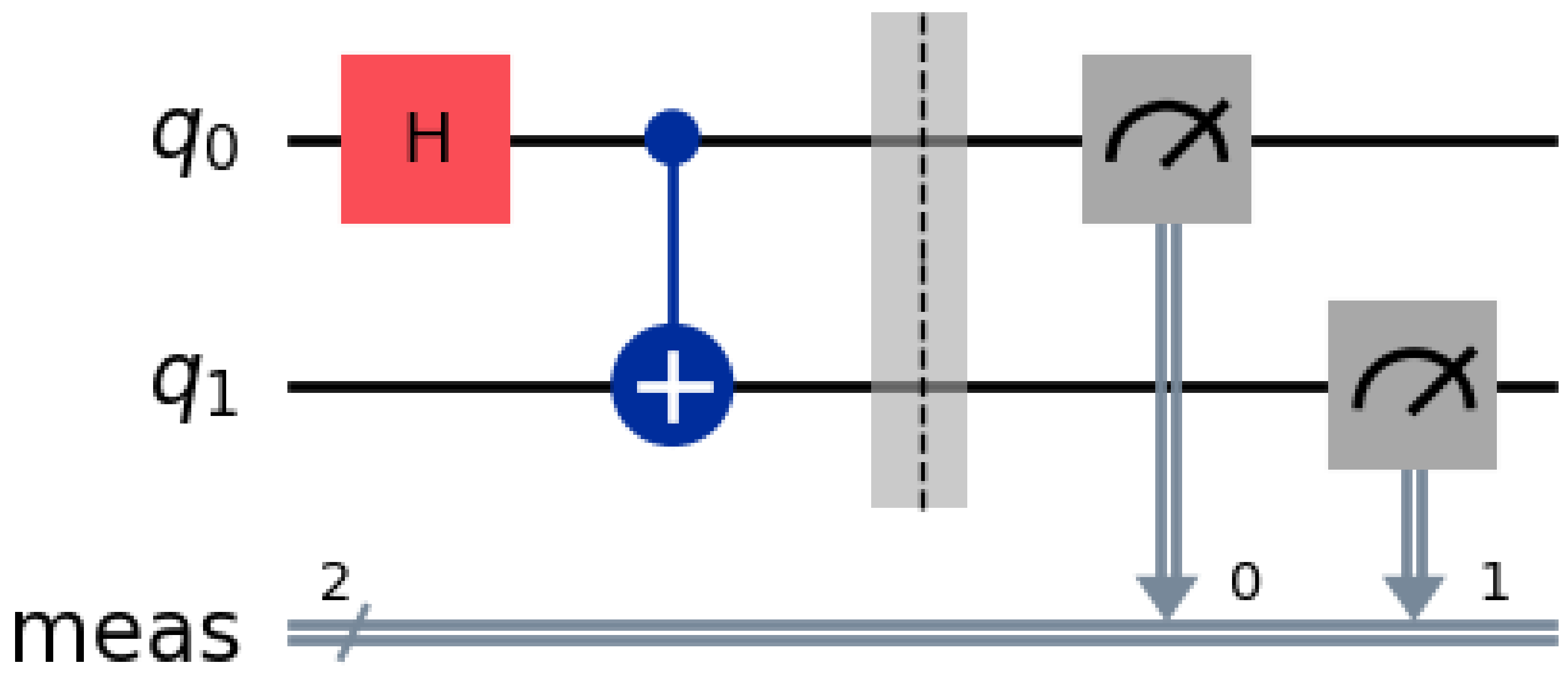
✓ 0.0s

Example: Generate and run a Bell state

Now we will design the quantum circuit that generates the Bell state $|\phi^+\rangle$

```
qc = QuantumCircuit(2)
qc.h(0)
qc.cx(0, 1)
qc.measure_all()
qc.draw(output = 'mpl')
```

✓ 0.0s



Example: Generate and run a Bell state

We can also use registers to create the circuit

```
q = QuantumRegister(size = 2, name = 'q')
c = ClassicalRegister(size = 2, name = 'c')
qc = QuantumCircuit(q, c)
qc.h(q[0])
qc.cx(q[0], q[1])
qc.measure(q[0], c[0])
qc.measure(q[1], c[1])
qc.draw(output = 'mpl')
```

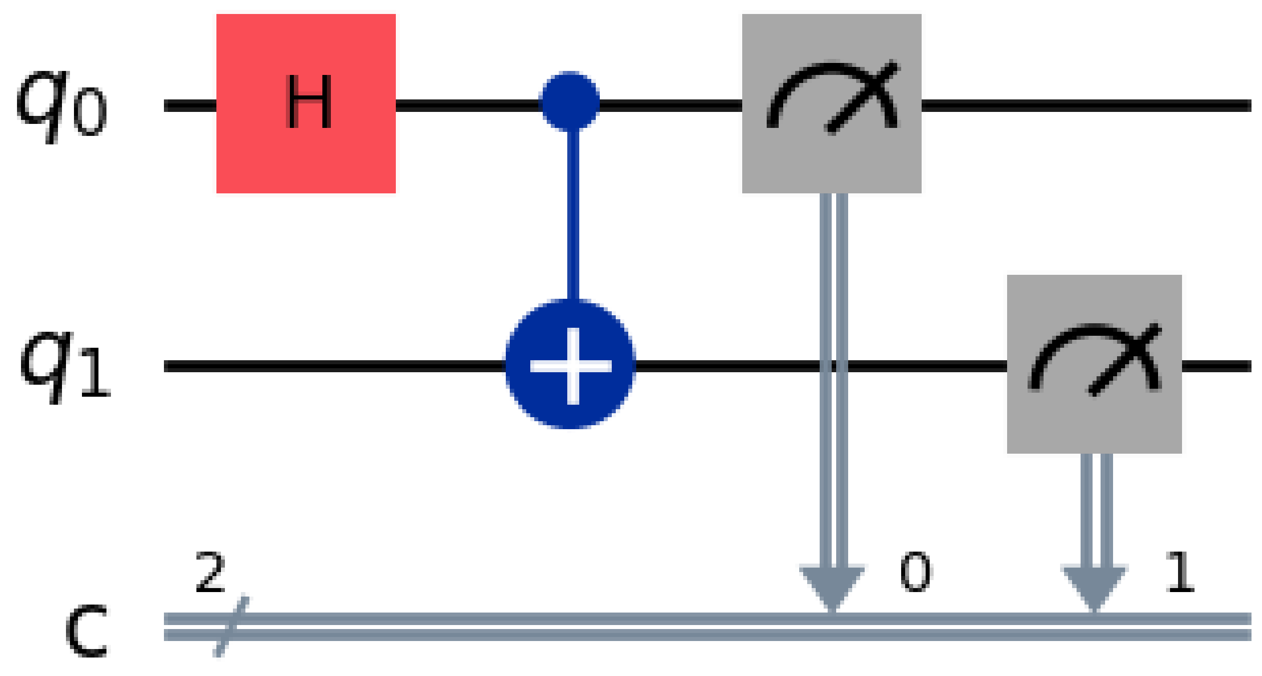
✓ 0.0s

Example: Generate and run a Bell state

We can also use registers to create the circuit

```
q = QuantumRegister(size = 2, name = 'q')
c = ClassicalRegister(size = 2, name = 'c')
qc = QuantumCircuit(q, c)
qc.h(q[0])
qc.cx(q[0], q[1])
qc.measure(q[0], c[0])
qc.measure(q[1], c[1])
qc.draw(output = 'mpl')
```

✓ 0.0s



Example: Generate and run a Bell state

We can run this circuit on a noiseless simulator to compare it with the real QPU run.

```
sampler = StatevectorSampler()
job = sampler.run([qc])
result = job.result()[0]
counts = result.data.c.get_counts()
counts
```

✓ 0.0s

```
{'00': 515, '11': 509}
```

```
plot_histogram(counts)
```

✓ 0.0s

Example: Generate and run a Bell state

We can run this circuit on a noiseless simulator to compare it with the real QPU run.

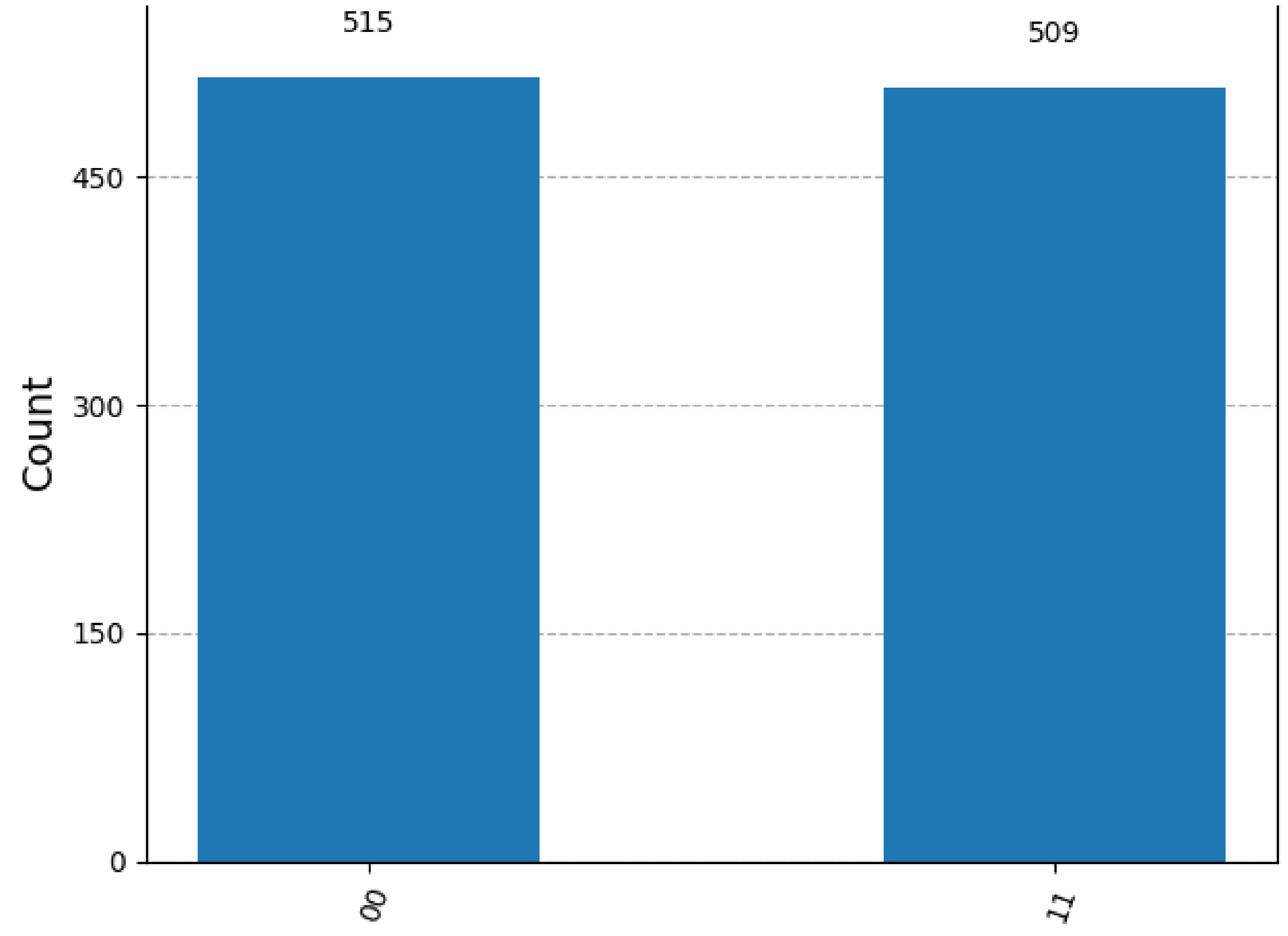
```
sampler = StatevectorSampler()
job = sampler.run([qc])
result = job.result()[0]
counts = result.data.c.get_counts()
counts
```

✓ 0.0s

```
{'00': 515, '11': 509}
```

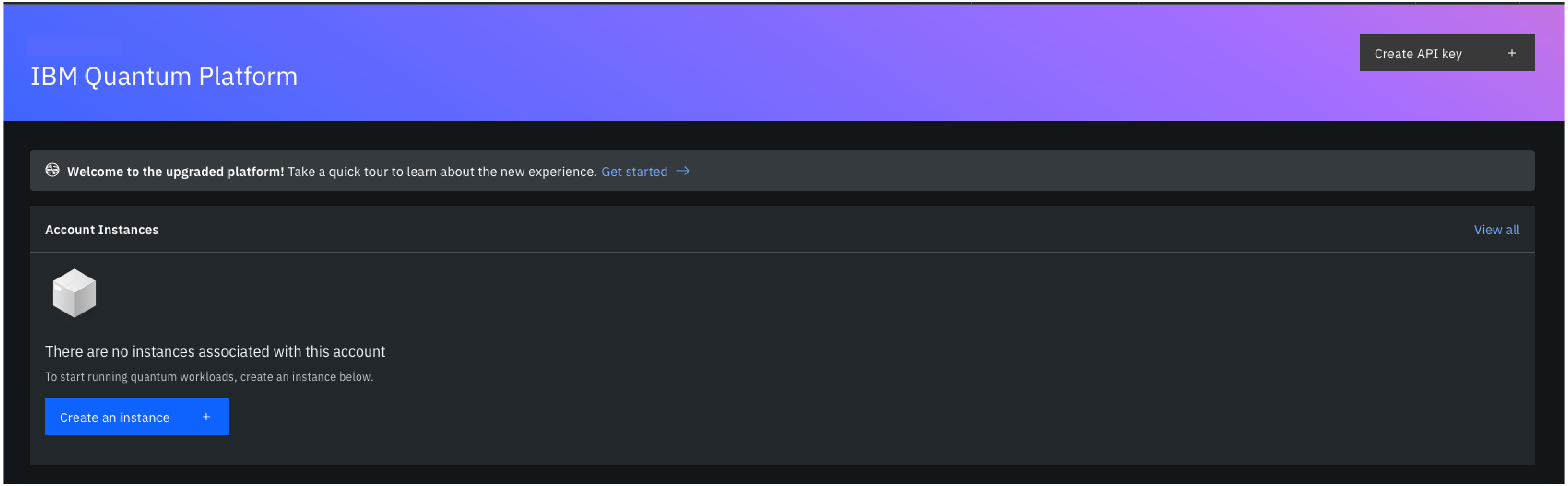
```
plot_histogram(counts)
```

✓ 0.0s



Example: Generate and run a Bell state

Before launching it to the QPU, we need to transpile the circuit to adapt it to the target hardware. To do this, we will start by retrieving our API token and instance CRN



Example: Generate and run a Bell state

Now we can set up our account in the notebook

```
your_api_key = "api_key"
your_crn = 'crn'

QiskitRuntimeService.save_account(
    channel="ibm_cloud",
    token=your_api_key,
    instance=your_crn,
    name="QFF25",
    overwrite=True
)

service = QiskitRuntimeService(name="QFF25")
```

Example: Generate and run a Bell state

Now we can set up our account in the notebook

```
your_api_key = "api_key"
your_crn = 'crn'

QiskitRuntimeService.save_account(
    channel="ibm_cloud",
    token=your_api_key,
    instance=your_crn,
    name="QFF25",
    overwrite=True
)

service = QiskitRuntimeService(name="QFF25")
```

And select our target QPU

```
service.backend('ibm_basquecountry')
```

Example: Generate and run a Bell state

Next, we will transpile and optimize our circuit for hardware execution

```
pm = generate_preset_pass_manager(optimization_level = 2,
                                  backend = backend)
isa_qc = pm.run(qc)
isa_qc.draw('mpl', idle_wires = False)
```

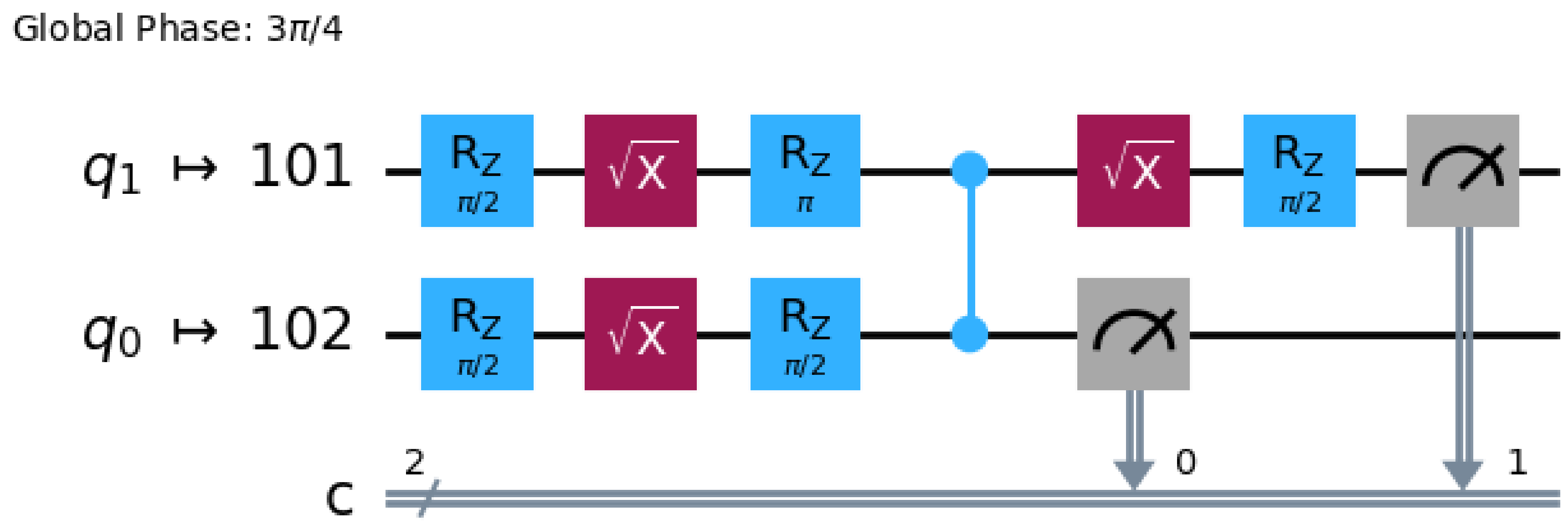
✓ 2.7s

Example: Generate and run a Bell state

Next, we will transpile and optimize our circuit for hardware execution

```
pm = generate_preset_pass_manager(optimization_level = 2,
                                  backend = backend)
isa_qc = pm.run(qc)
isa_qc.draw('mpl', idle_wires = False)
```

✓ 2.7s

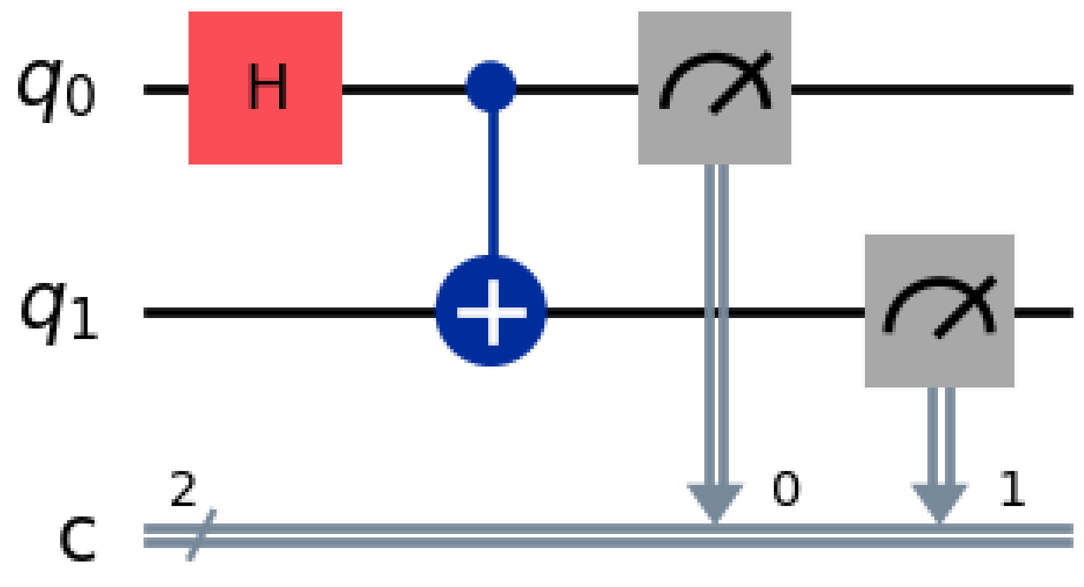


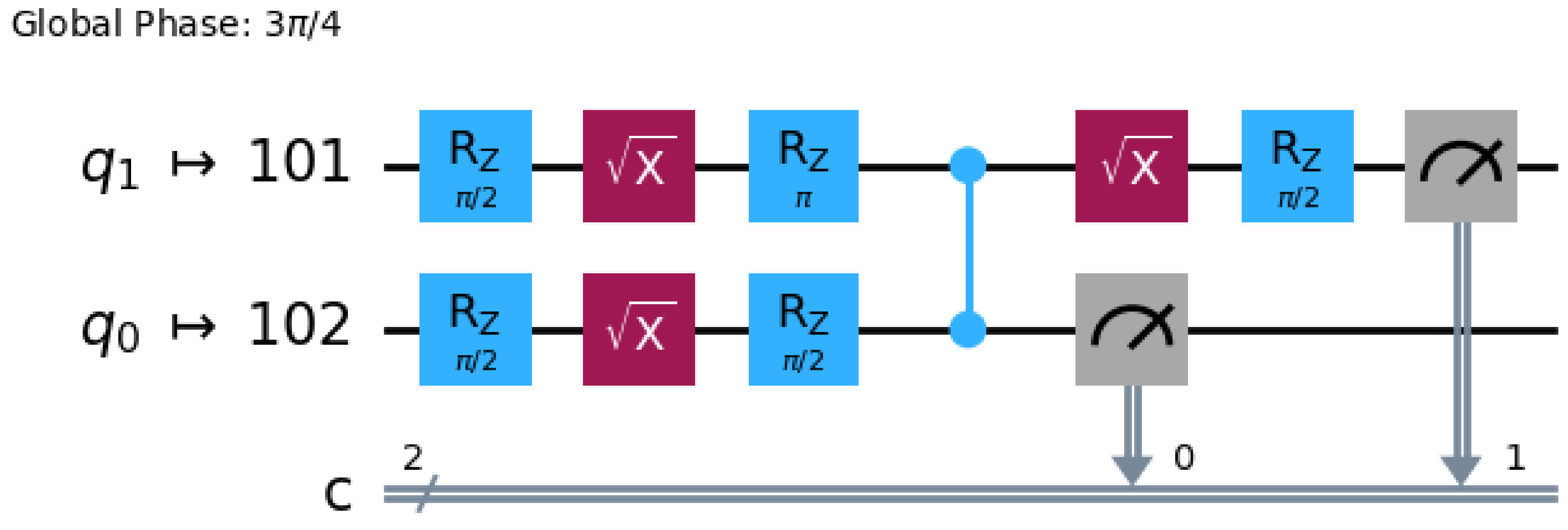
Example: Generate and run a Bell state

Next, we will transpile and optimize our circuit for hardware execution

```
pm = generate_preset_pass_manager(optimization_level = 2,
                                  backend = backend)
isa_qc = pm.run(qc)
isa_qc.draw('mpl', idle_wires = False)
```

✓ 2.7s





Example: Generate and run a Bell state

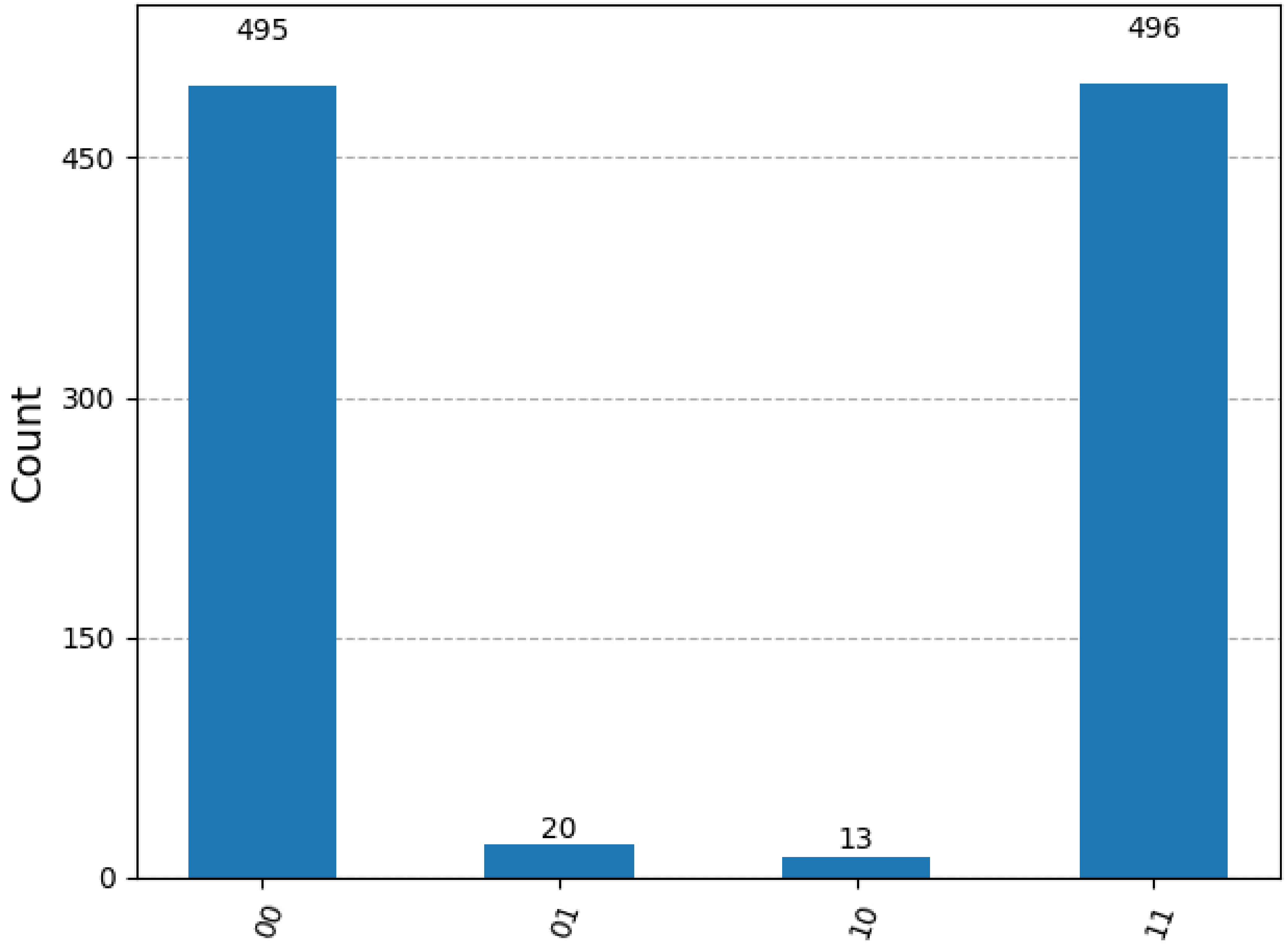
Finally, we are prepared to launch our first job to the real device!

```
session = Session(backend = backend, max_time = '1m')
sampler = SamplerV2(mode = session)
sampler.options.default_shots = 1024
job = sampler.run([isa_qc])
session.close()

result = job.result()[0]
counts = result.data.c.get_counts()
plot_histogram(counts)
```

✓ 19.1s

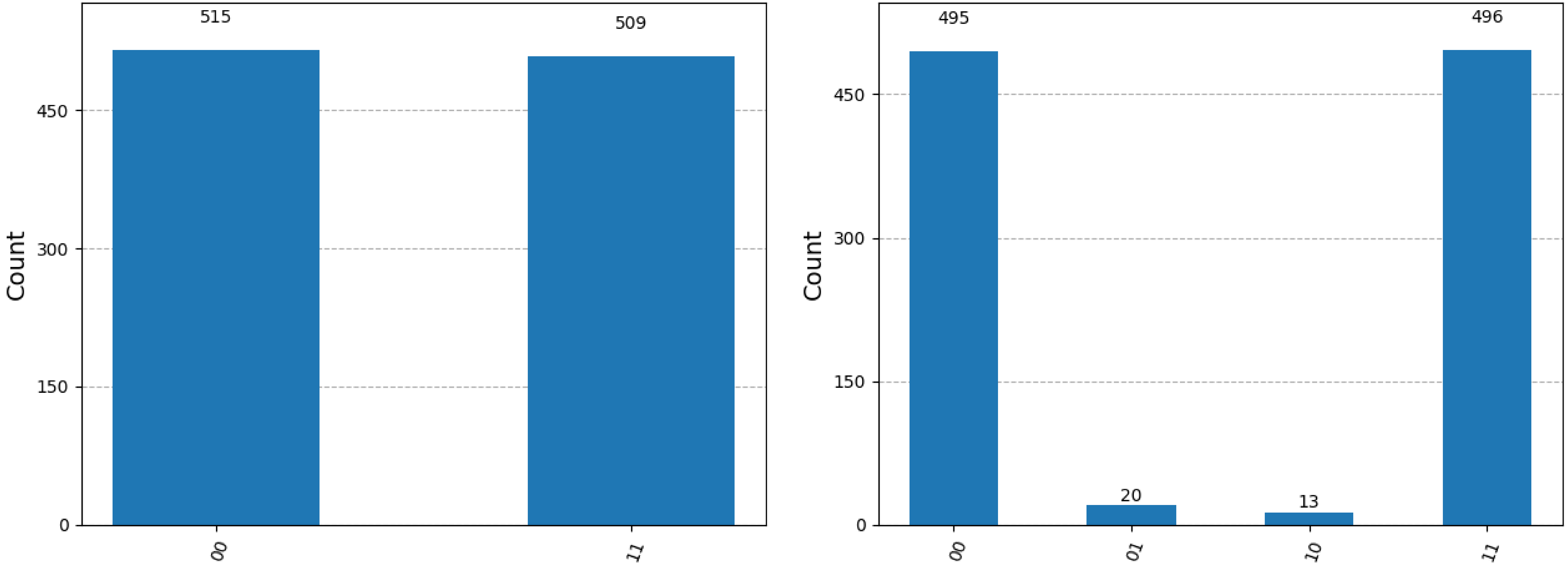
Example: Generate and run a Bell state

Finally, we are prepared to launch our first job to the real device!



Example: Generate and run a Bell state

Due to the presence of noise, new states appear in the output bitstring distribution



Thanks for your attention!