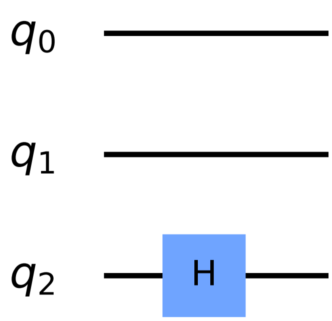
In [1]:
```
import numpy as np
from numpy import pi
# importing Qiskit
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.providers.ibmq import least_busy
from qiskit.tools.monitor import job_monitor
from qiskit.visualization import plot_histogram, plot_bloch_multivector
%config InlineBackend.figure_format = 'svg' # Makes the images look nice
```

In [2]:
```
qc = QuantumCircuit(3)
```

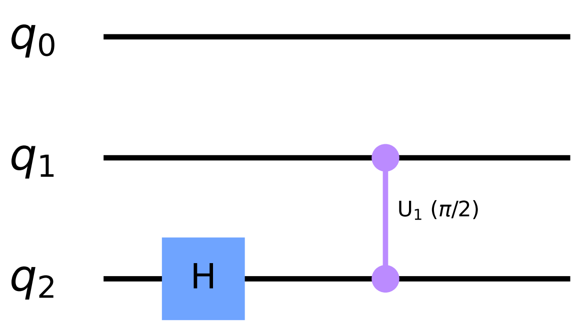
In [3]:
```
qc.h(2)
qc.draw('mpl')
```

Out[3]:



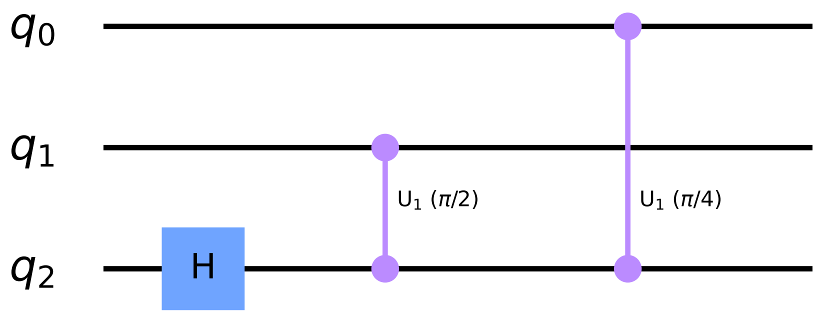
In [4]:
```
qc.cu1(pi/2, 1, 2) # CROT from qubit 1 to qubit 2
qc.draw('mpl')
```

Out[4]:



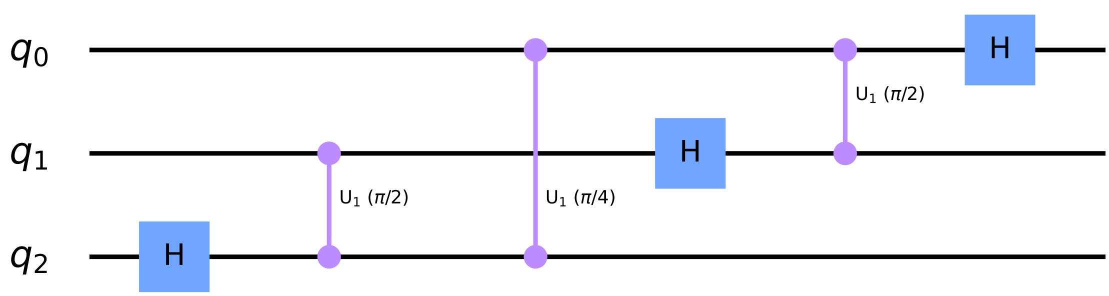
In [5]:
```
qc.cu1(pi/4, 0, 2) # CROT from qubit 2 to qubit 0
qc.draw('mpl')
```

Out[5]:



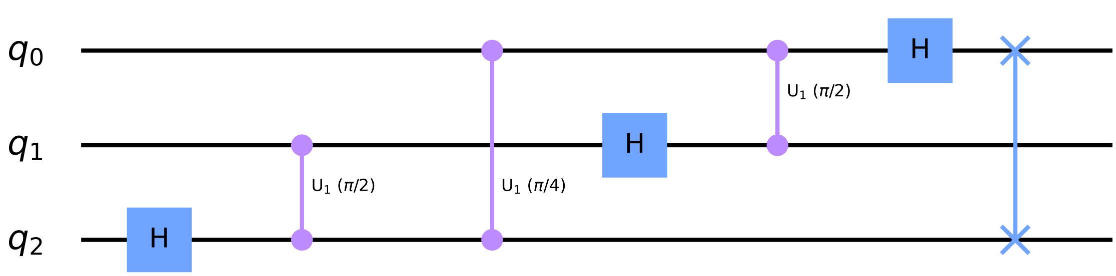
In [6]:
```
qc.h(1)
qc.cu1(pi/2, 0, 1) # CROT from qubit 0 to qubit 1
qc.h(0)
qc.draw('mpl')
```

Out[6]:



In [7]:
```
qc.swap(0,2)
qc.draw('mpl')
```

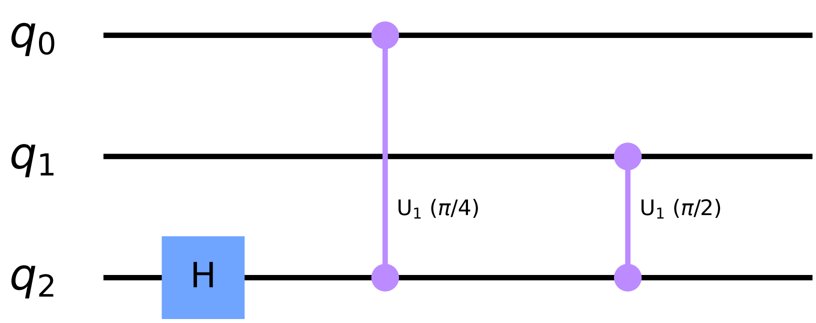
Out[7]:



In [8]:
```
def qft_rotations(circuit, n):
    if n == 0: # Exit function if circuit is empty
        return circuit
    n -= 1 # Indexes start from 0
    circuit.h(n) # Apply the H-gate to the most significant qubit
    for qubit in range(n):
        # For each less significant qubit, we need to do a
        # smaller-angled controlled rotation:
        circuit.cu1(pi/2**(n-qubit), qubit, n)
```

In [10]:
```
qc = QuantumCircuit(3)
qft_rotations(qc,3)
qc.draw('mpl')
```

Out[10]:

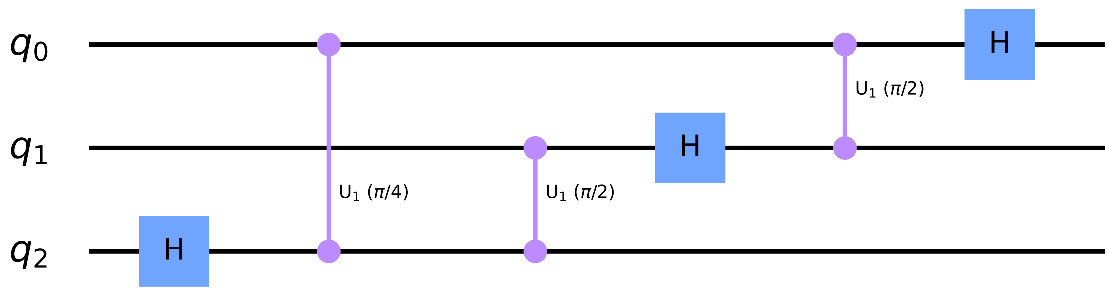


In [11]:
```
from qiskit_textbook.widgets import scalable_circuit
scalable_circuit(qft_rotations)
```

In [12]:

```
def qft_rotations(circuit, n):
    """Performs qft on the first n qubits in circuit (without swaps)"""
    if n == 0:
        return circuit
    n -= 1
    circuit.h(n)
    for qubit in range(n):
        circuit.cu1(pi/2**(n-qubit), qubit, n)
    # At the end of our function, we call the same function again on
    # the next qubits (we reduced n by one earlier in the function)
    qft_rotations(circuit, n)

# Let's see how it looks:
qc = QuantumCircuit(3)
qft_rotations(qc,3)
qc.draw('mpl')
```

Out[12]:

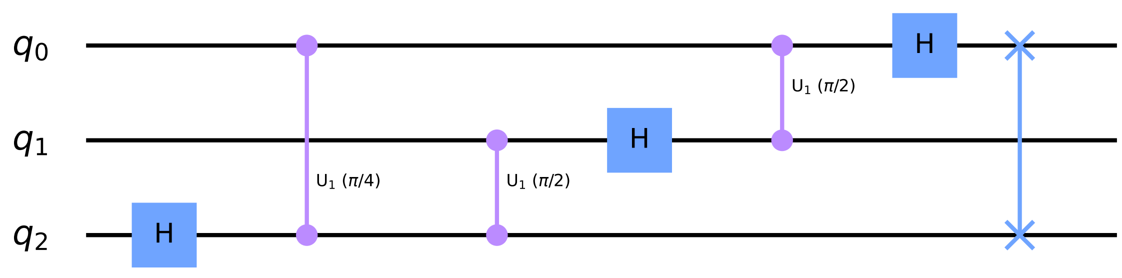


In [13]:

```
scalable_circuit(qft_rotations)
```

In [15]:
```
def swap_registers(circuit, n):
    for qubit in range(n//2):
        circuit.swap(qubit, n-qubit-1)
    return circuit

def qft(circuit, n):
    """QFT on the first n qubits in circuit"""
    qft_rotations(circuit, n)
    swap_registers(circuit, n)
    return circuit

# Let's see how it looks:
qc = QuantumCircuit(3)
qft(qc,3)
qc.draw('mpl')
```

Out[15]:



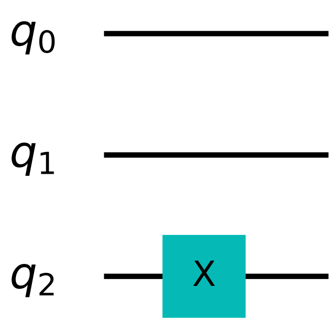
In [16]:
```
scalable_circuit(qft)
```

In [18]:
```
bin(4)
```

Out[18]: '0b100'

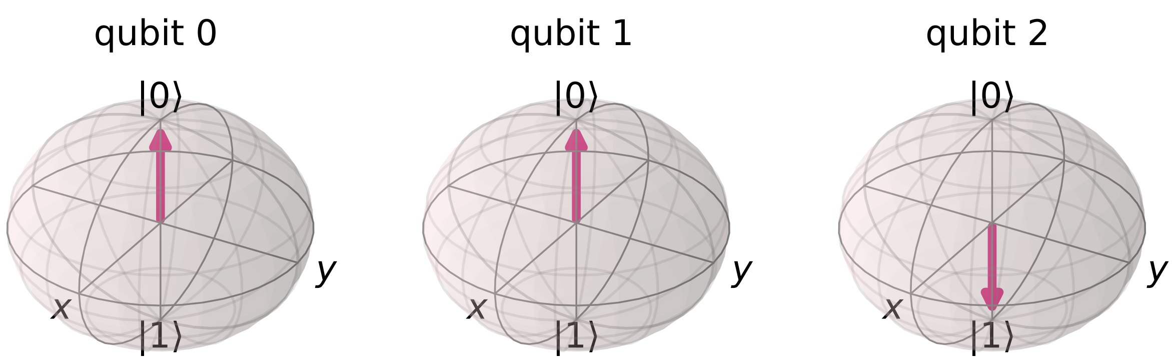
In [20]:
```
# Create the circuit
qc = QuantumCircuit(3)

# Encode the state 4
qc.x(2)
%config InlineBackend.figure_format = 'svg' # Makes the images fit
qc.draw('mpl')
```

Out[20]:



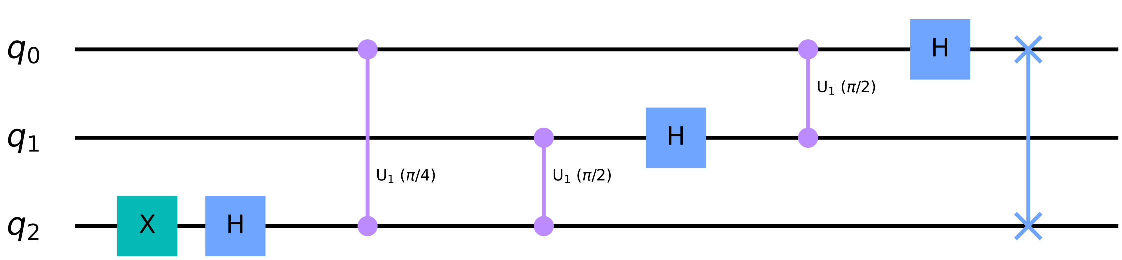
In [21]:
```
backend = Aer.get_backend("statevector_simulator")
statevector = execute(qc, backend=backend).result().get_statevector()
plot_bloch_multivector(statevector)
```

Out[21]:



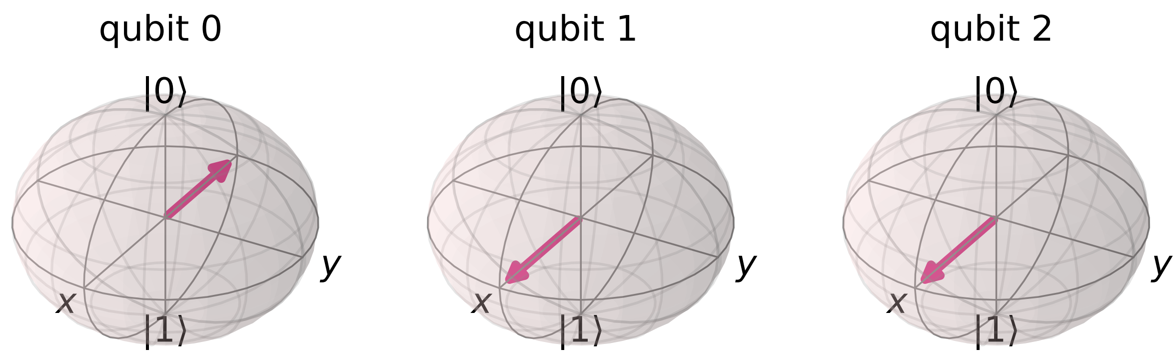
In [22]:
```
qft(qc,3)
qc.draw('mpl')
```

Out[22]:



In [23]:
```
statevector = execute(qc, backend=backend).result().get_statevector()
plot_bloch_multivector(statevector)
```

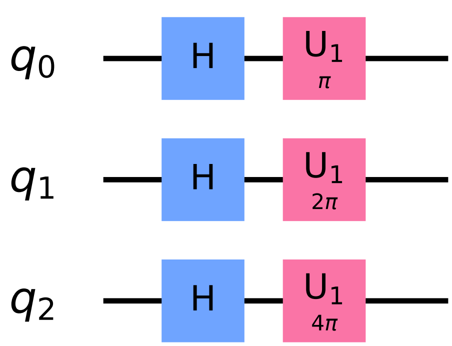
Out[23]:



In [24]:
```
def inverse_qft(circuit, n):
    """Does the inverse QFT on the first n qubits in circuit"""
    # First we create a QFT circuit of the correct size:
    qft_circ = qft(QuantumCircuit(n), n)
    # Then we take the inverse of this circuit
    invqft_circ = qft_circ.inverse()
    # And add it to the first n qubits in our existing circuit
    circuit.append(invqft_circ, circuit.qubits[:n])
    return circuit.decompose() # .decompose() allows us to see the indiv
idual gates
```

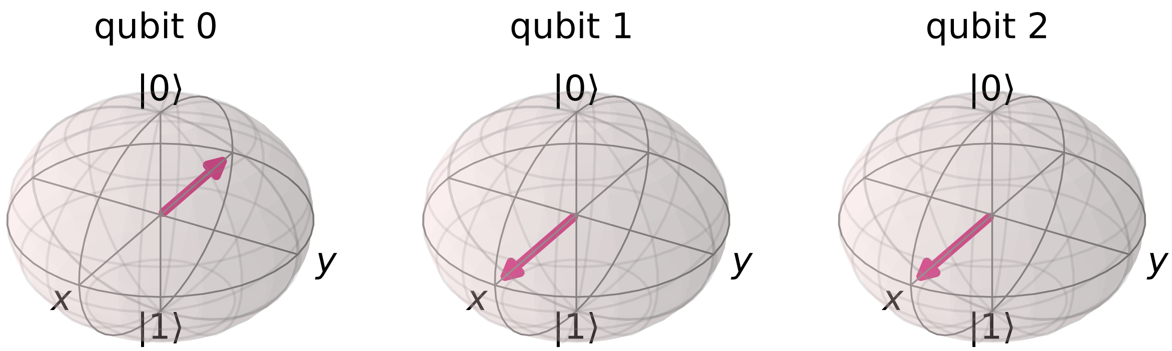
In [25]:
```
nqubits = 3
number = 4
qc = QuantumCircuit(nqubits)
for qubit in range(nqubits):
    qc.h(qubit)
qc.u1(number*pi/4,0)
qc.u1(number*pi/2,1)
qc.u1(number*pi,2)

qc.draw('mpl')
```

Out[25]:



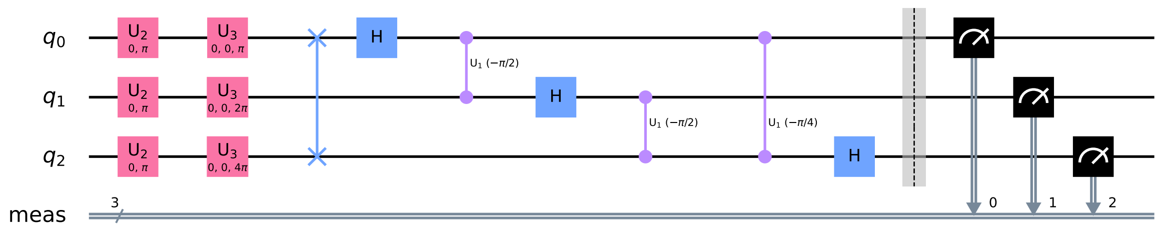
In [26]:
```
backend = Aer.get_backend("statevector_simulator")
statevector = execute(qc, backend=backend).result().get_statevector()
plot_bloch_multivector(statevector)
```

Out[26]:



In [27]:
```
qc = inverse_qft(qc,nqubits)
qc.measure_all()
qc.draw('mpl')
```

Out[27]:

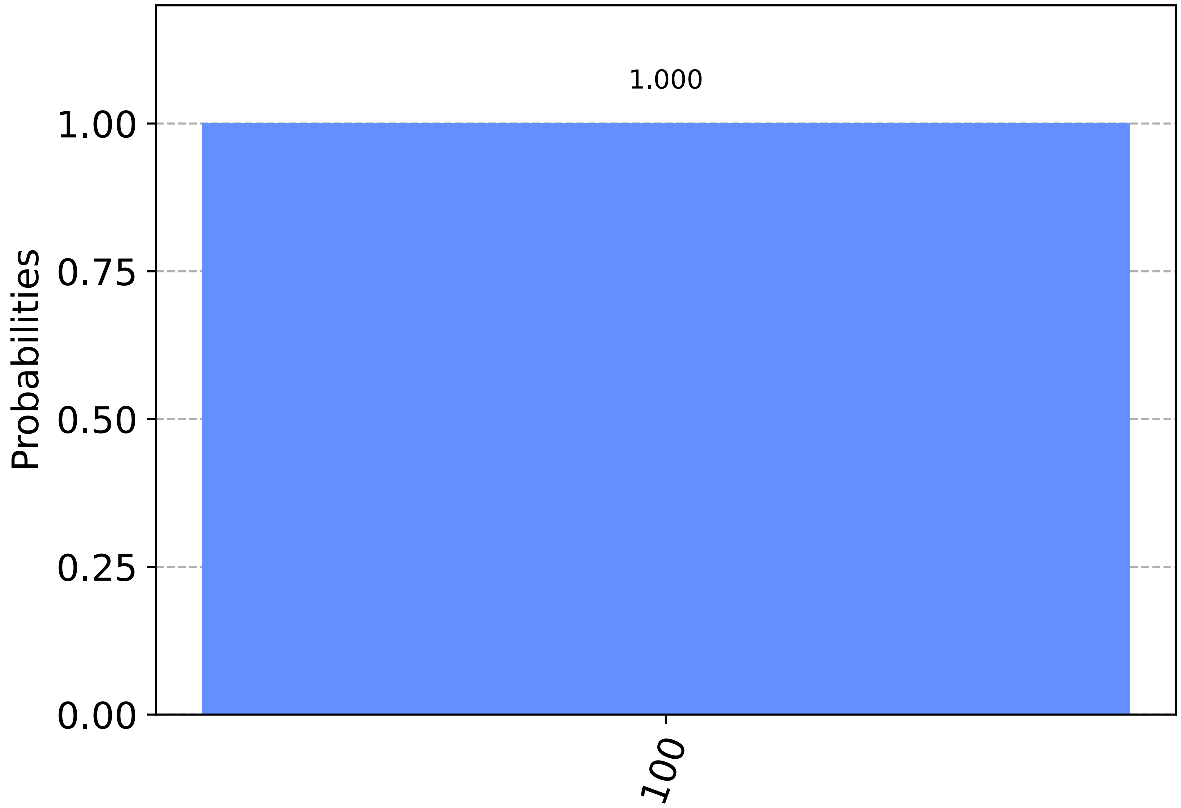


In [29]:
```
shots = 2048
job = execute(qc, backend=backend, shots=shots, optimization_level=3)
job_monitor(job)
```

```
Job Status: job has successfully run
```

In [30]:
```
counts = job.result().get_counts()
plot_histogram(counts)
```

Out[30]:



In [ ]: