Quantum Algorithms: A Top-Down Approach

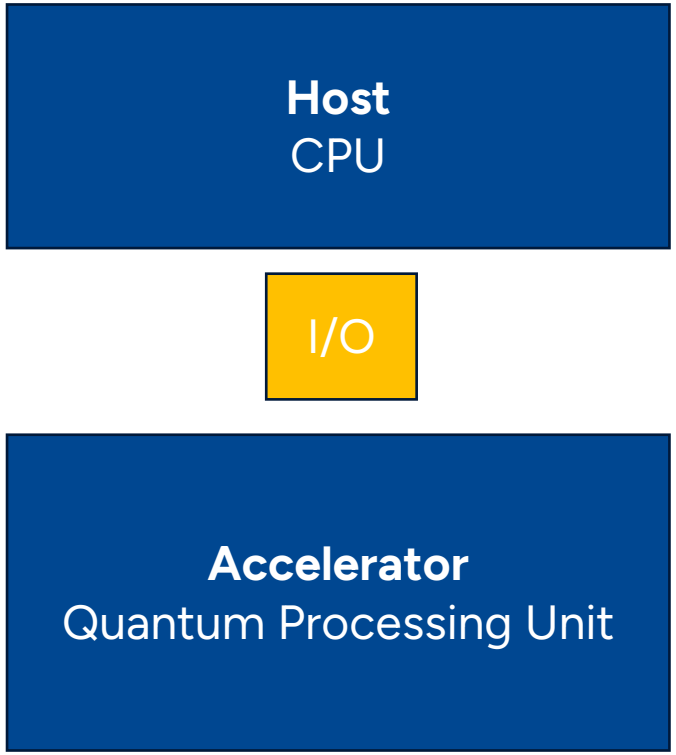
Stefano Markidis

KTH Royal Institute of Technology

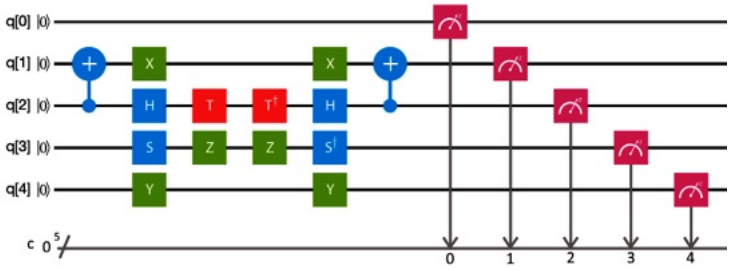
How a Quantum Computer Looks Today – IBM System

https://www.popsci.com/technology/in-photos-journey-to-the-center-of-a-quantum-computer/

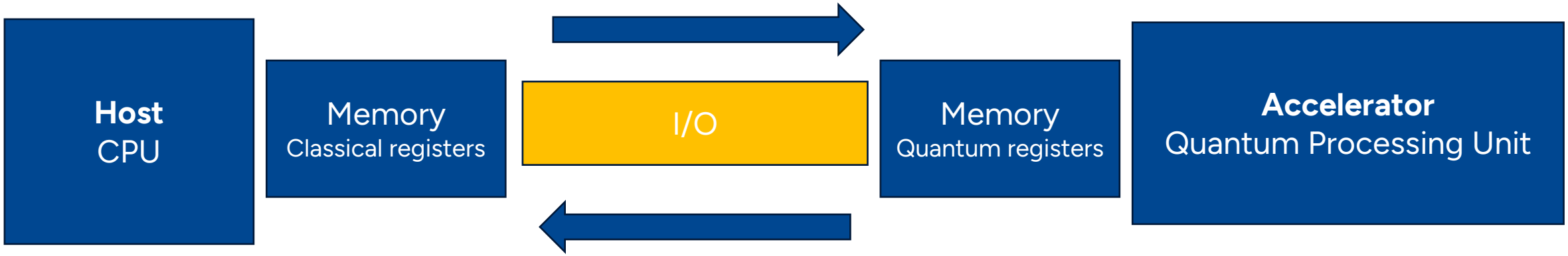
Quantum Processor Unit (QPU) Abstraction



The Host-Accelerator Quantum Machine Model



Execute Quantum Circuit (Kernel)



Measurements / Sampling

Similar to classical Von Neumann architectures, note the presence of memories on both the CPU and QPU sides.

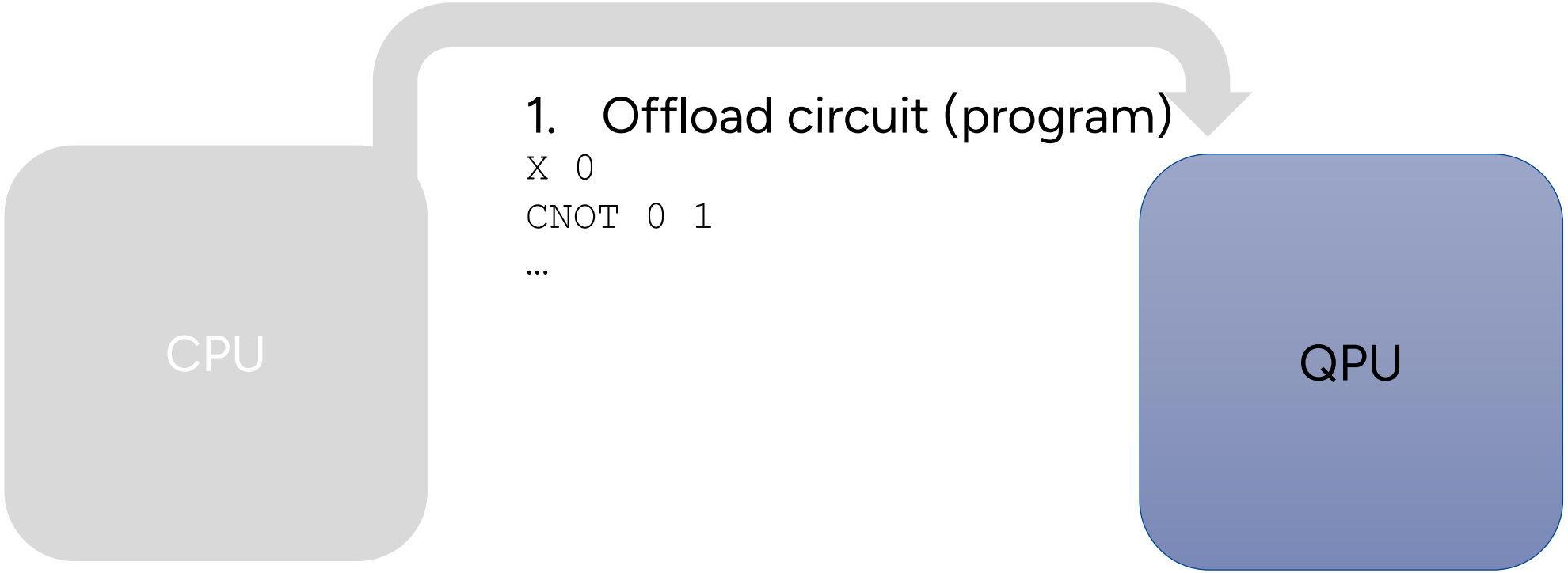
QPU (and Accelerators) Characteristics

- Usually, a program running on a CPU will issue QPU instructions and later retrieve the results.
  - Some tasks are very well suited to the QPU, and others are not.
- The QPU runs on a separate clock from the CPU and usually has its own dedicated hardware interfaces to external devices (such as optical outputs).
- A typical QPU has its own special RAM, which the CPU cannot efficiently access.
- When a computation is done, a projection of the result is returned to the CPU, and most of the QPU's internal working data is discarded.

Programming a Quantum Computer

1. Express our program as a quantum circuit (define the combination of quantum gates acting on the qubit)
2. Offload our code (definition of our quantum circuit)
3. Prepare the distribution
4. Sample the distribution (measure many times)

1 . Express our Code as a Quantum Circuit and Offload to QPU



QPU & FPGAs (loose analogy for QPU)

The closest classical accelerator to QPU is **FPGAs**, where we can provide configuration of hardware and network (our code is the classical circuit for FPGAs.)

Example of FPGA

https://www.amazon.com/Altera-Cyclone-FPGA-Development-Board/dp/B0188D4ENO

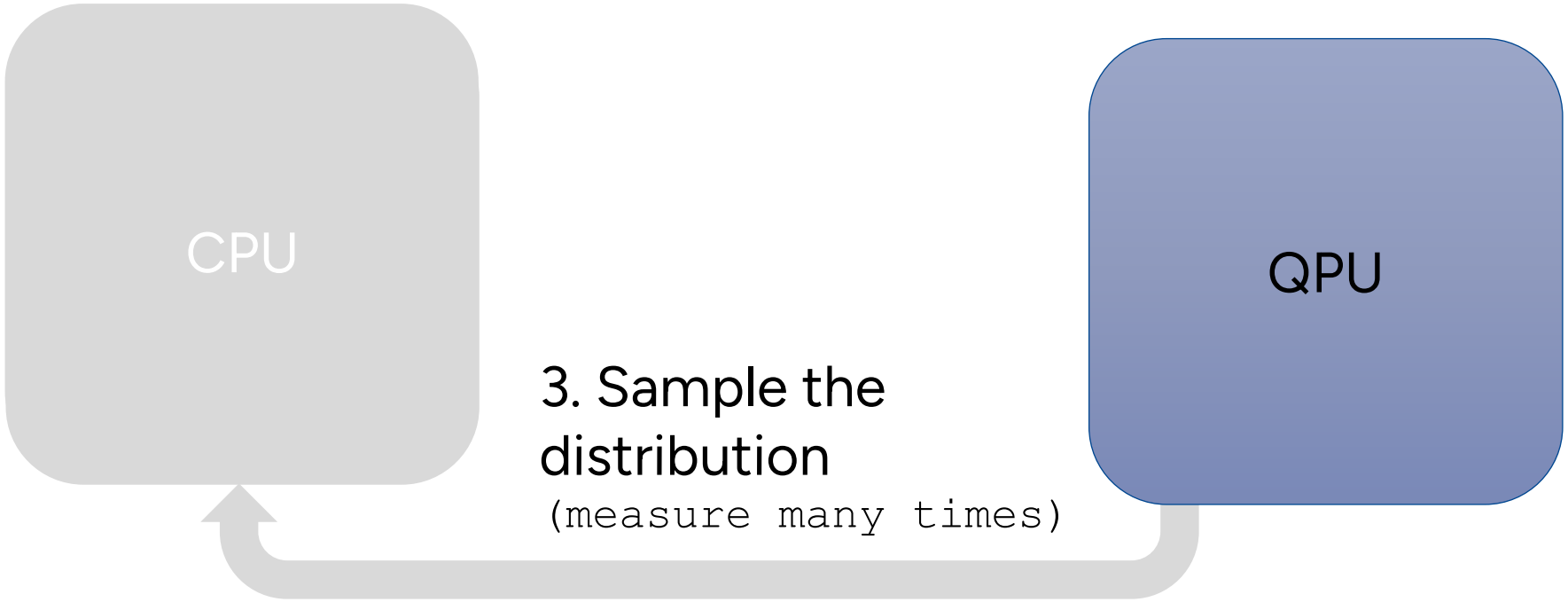
3. Prepare Distribution

CPU

QPU

2. Prepare the distribution (apply transformations to input)

4. Sample the Distribution



Programming with Offloading Model: IBM Qiskit Example

```
q = QuantumRegister(2,'q')
c = ClassicalRegister(2,'c')

def firstBellState():
    circuit = QuantumCircuit(q,c)

    circuit.h(q[0]) # Hadamard gate
    circuit.cx(q[0],q[1]) # CNOT gate
    circuit.measure(q,c) # Qubit Measurment

    print(circuit)

    job = execute(circuit, backend, shots=8192)

    job_monitor(job)
    counts = job.result().get_counts()

    print(counts)
```

1 Define variables on the host and accelerator

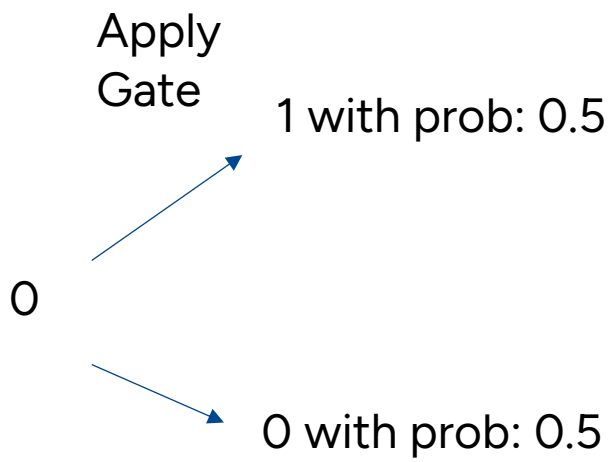
2 Define the kernel as a quantum circuit acting on quantum registers

3 Launch the Kernel

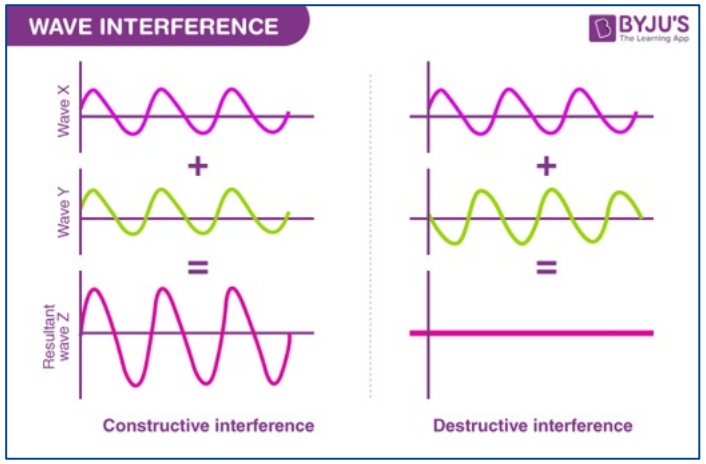
Quantum Applications Mechanism

- Initialize the system in a classical state, e.g., a zero state
- Apply gates to execute an algorithm
  - Spawn a superposition of quantum states
    - Use special gates: HAD (Hadamard), RY (Rotation Y), …
  - Use (wave) destructive/constructive interference to eliminate/reinforce potential solutions
    - Change the phases to interfere

superposition

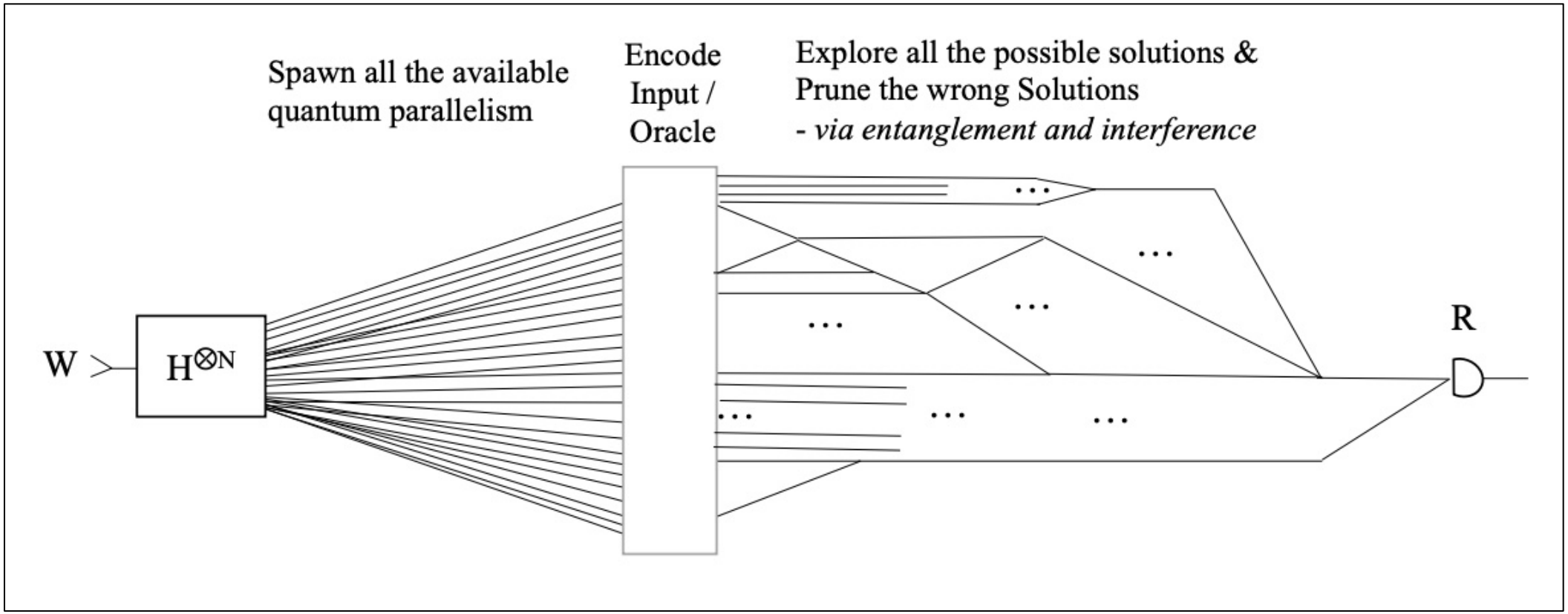


interference



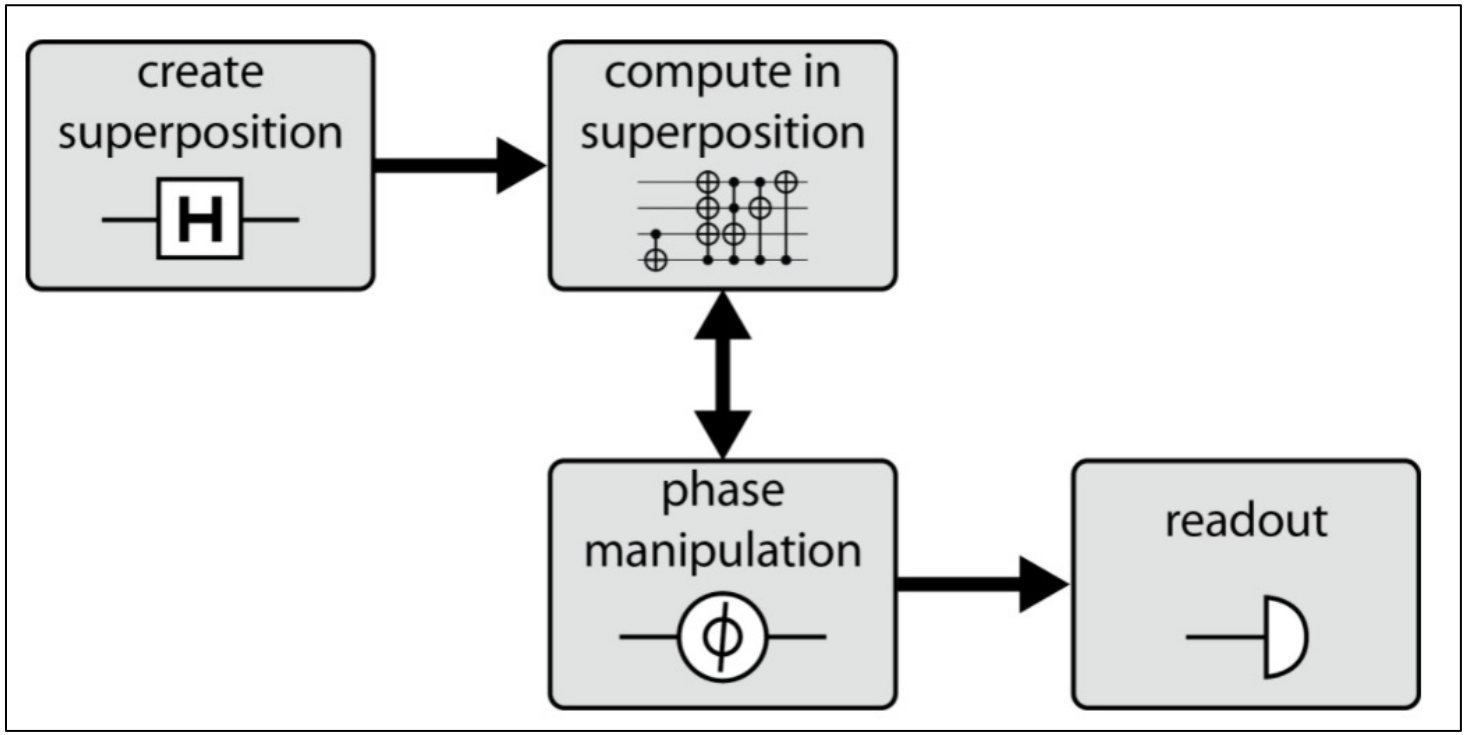
Credits: https://byjus.com/physics/constructive-interference/

Quantum Algorithms



Quantum Algorithms Compose Quantum Primitives

- Quantum algorithms are a composition of different quantum primitives
  - Think about libraries and templates in CS



- Primitives associated with the ***computation in superposition*** enable us to perform computations using the implicit parallelism of superposition.
- **Phase manipulation**: Primitives that realize phase manipulation ensure that our results can actually be read practically.

Example of Quantum Primitives

Quantum Arithmetics	Compute in superposition
Amplitude amplification	Phase manipulation
Quantum Fourier Transform	Phase manipulation
Phase estimation	Phase manipulation

The Abstraction of Qubit

- The quantum bit (qbit) is an extension of the probabilistic bit, which is a generalization of the bit
  - **Bit** = scalar
  - **Probabilistic bit** = vector column with probability encoded in real numbers >=0 and <=1
  - **Quantum bit** = probabilistic bit where the probability is encoded in the squared modulus of a complex number (called amplitude)

Quantum Bit = Qubit

- Similar to the probabilistic bit, but there are main essential differences
  - It is expressed by a vector and not a scalar
    - As the probabilistic bit
  - Its entries are expressed by a complex number instead of a real number
    - This complex number is called amplitude.
  - The modulus square of the complex number is the probability

$$\begin{bmatrix} \frac{i}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \end{bmatrix}$$

Same Probability of Observing 0 and 1

Probabilistic Bit

$$\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

Quantum Bit

$$\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \quad \dots \quad \begin{bmatrix} \frac{i}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \end{bmatrix}$$

Qubit as 2-dimensional Complex Vector

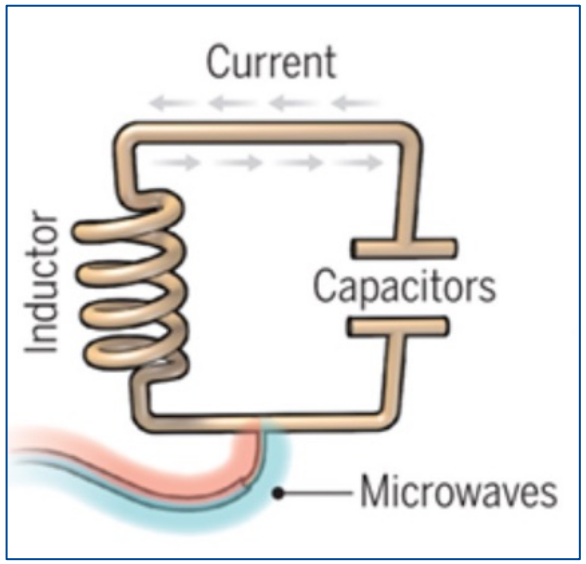
- We represent a qubit as a 2-by-1 matrix with complex numbers

$$\begin{matrix}\mathbf{0}\\ \mathbf{1}\end{matrix}\begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$$

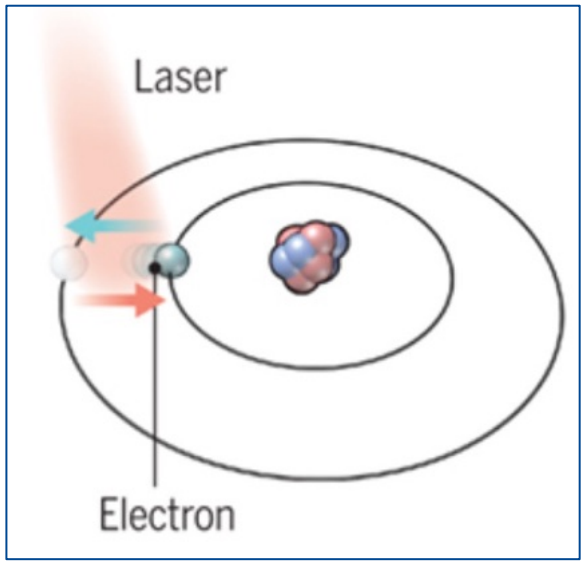
- where $|c_0|^2 + |c_1|^2 = 1$.
- A **classical bit** is a **special type of qubit**.
- $|c_0|^2$ is to be interpreted as the probability that after measuring the qubit, it will be found in state $|0\rangle$.
- $|c_1|^2$ is to be interpreted as the probability that after measuring the qubit it will be found in state $|1\rangle$.
- Whenever **we measure a qubit, it automatically becomes a bit**.
    - We will never measure a general qubit.

Implementation of a Qubit

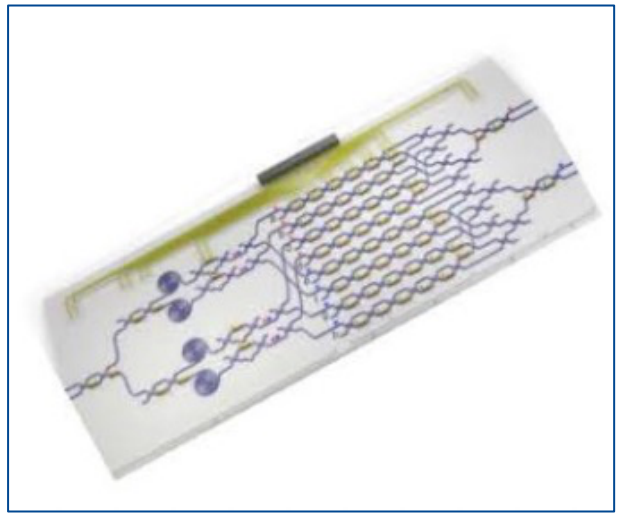
Superconducting Loops



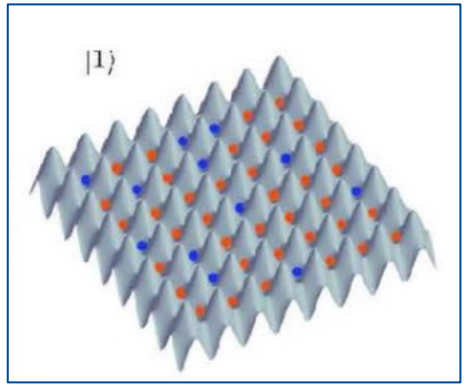
Trapped Ions



Photonics



Neutral Atoms



The Abstraction of Quantum Gate - I

- Transformations acting on one or multiple qubits
  - Each qubit = wire
- The transformation on a qubit can be expressed by matrix multiplication with the probabilistic qubit
- Valid transformations are only those respecting QM laws
- Transformation must:
  - Be reversible, e.g., matrix A can be inverted
  - Preserve the normalization of the probabilistic qubit
    - Also, conserve the “inner product” or the “geometry”
  - These two properties make A unitary

$q_1 = A\, q_0$

A is a matrix

The Abstraction of Quantum Gate - II

- The quantum gates can act on
  - single qubit = 1 wires
  - Two or multiple qubits
    - Act on the combined input qubit (2 or more qubits)
    - Controlled version of a single qubit gate

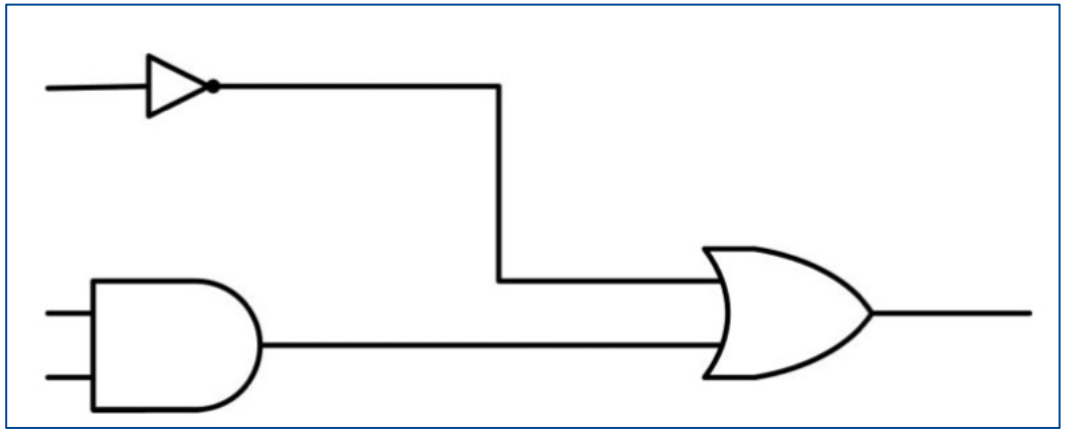
Single-qubit Gate: 2x2 matrix

Two-qubit Gate: 4x4 matrix

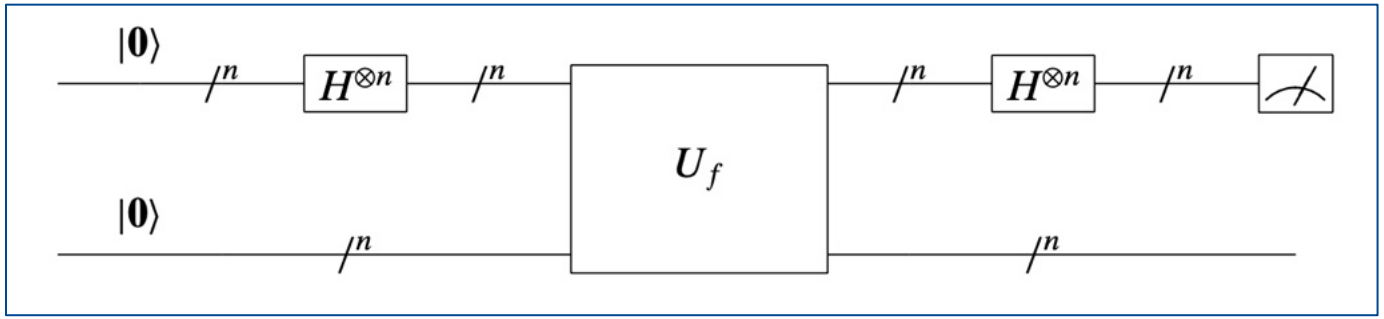
Circuits = Combine Gate Sequence

- Combinations of sequential and parallel operations, such as gates/matrices, will be referred to as **circuits**.
- Circuits can be decomposed into the sequential and parallel compositions of simple gates.

Classical circuit



Quantum circuit



Only a Few Gates Implemented on Quantum Computers

Examples of basis gates on IBM Brisbane and Torino Quantum Systems

```
cands = service.backends(simulator=False, operational=True, min_num_qubits=6)
for b in cands: print(b.name, b.num_qubits)

A = service.least_busy(simulator=False, operational=True, min_num_qubits=6)
B = next(b for b in cands if b.name != A.name)
A, B
```

```
ibm_torino 133
ibm_brisbane 127
(<IBMBackend('ibm_brisbane')>, <IBMBackend('ibm_torino')>)
```

```
cfgA = A.configuration(); cfgB = B.configuration()
print("A basis_gates:", cfgA.basis_gates)
print("B basis_gates:", cfgB.basis_gates)
cmapA = A.coupling_map; cmapB = B.coupling_map
```

```
A basis_gates: ['ecr', 'id', 'rz', 'sx', 'x']
B basis_gates: ['cz', 'id', 'rz', 'sx', 'x']
```

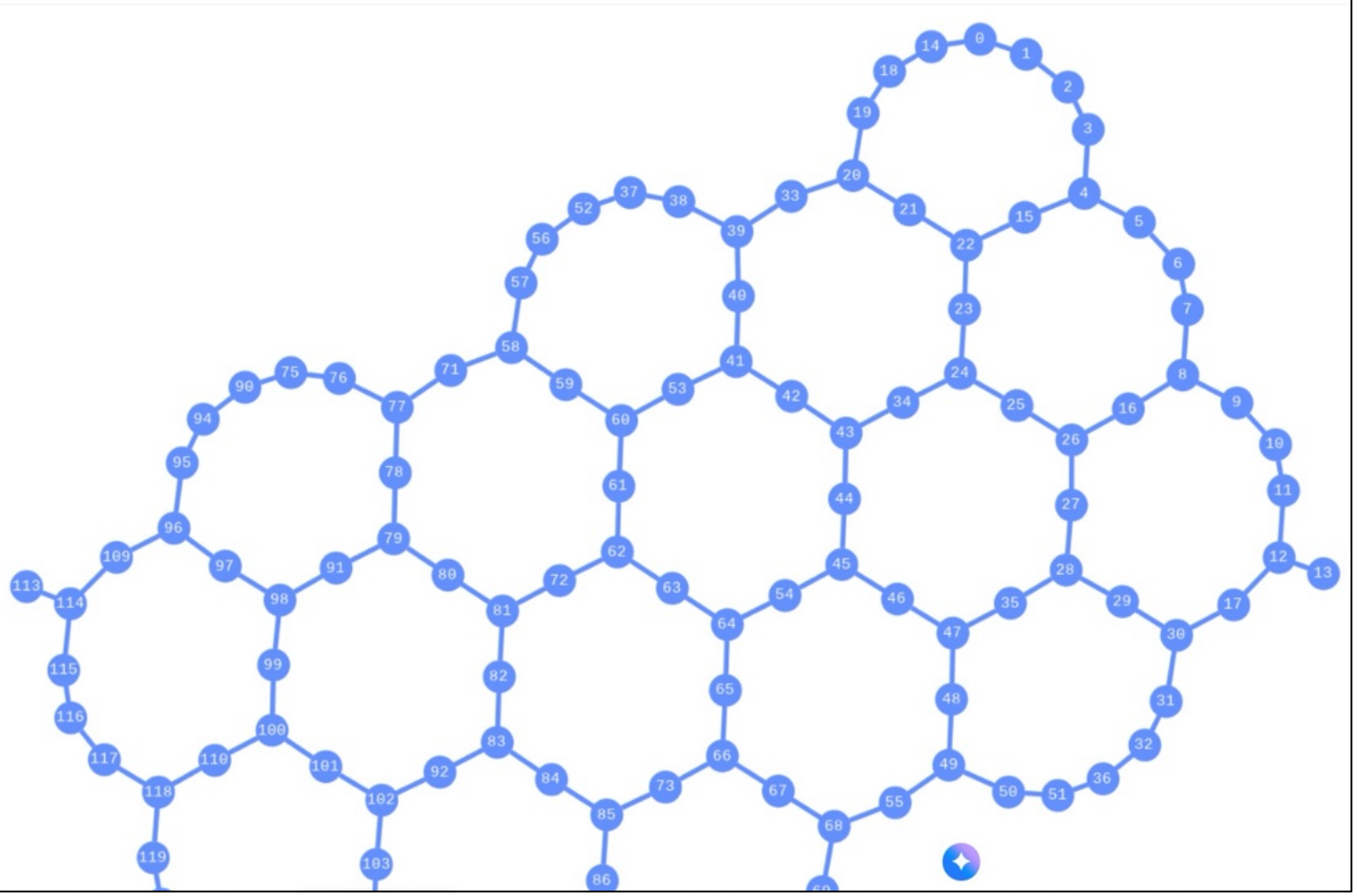
```
from qiskit.visualization import plot_coupling_map
plot_coupling_map(A.num_qubits, None, cmapA.get_edges())
```

- Use basis gates and equivalences to express the transformations of the given gates

Not all the Qubits are Connected

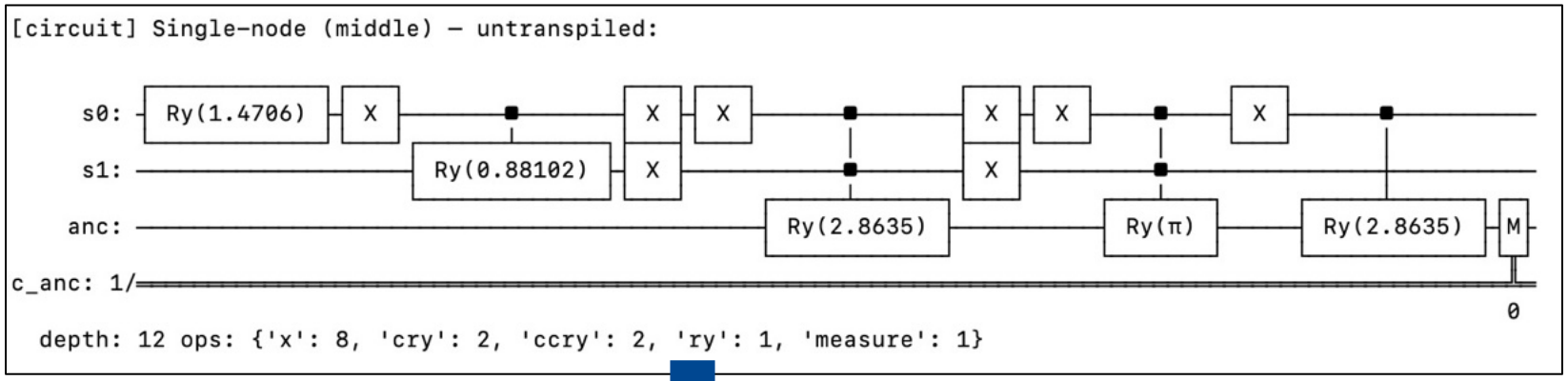
Topology of qubits connection on IBM Brisbane

```
from qiskit.visualization import plot_coupling_map
plot_coupling_map(A.num_qubits, None, cmapA.get_edges())
```

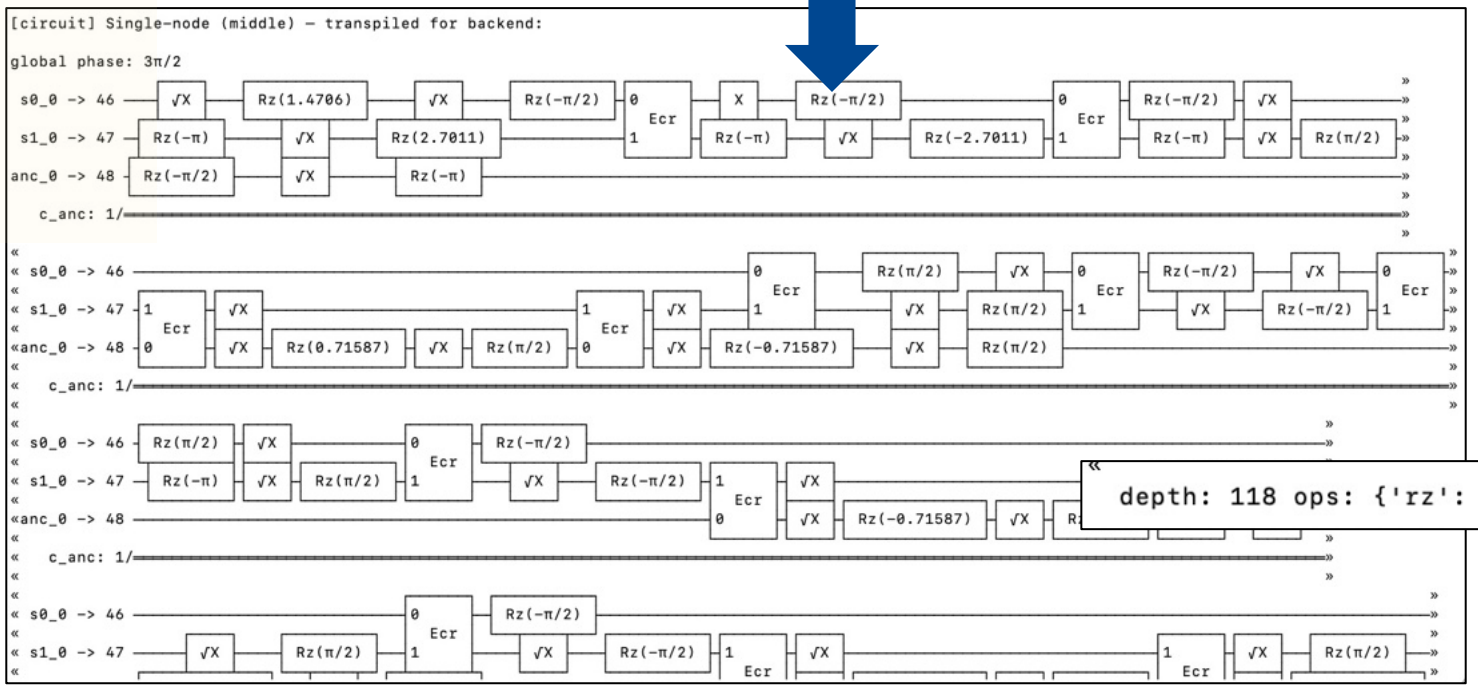


- When we use a C-operation, it might be that the qubits we use are not connected
- We need a sort of “routing”
  - Possible to add quantum gates to perform the routing

The Need for Compilation/Transpilation when Running on Quantum Computers



Transpilation



Transpilation = Automatic translation from the source quantum circuit to use basis gates and the underlying topology

Conclusions

- Quantum computing systems are complex experimental facilities for computation.
- Abstractions help us in dealing with complexity
- Qubit = probabilistic bit with probability encoded in amplitude
- Quantum gates = apply a unitary transformation on probabilistic qubits
- Algorithms combine qubits and quantum gates