How to program a quantum computer

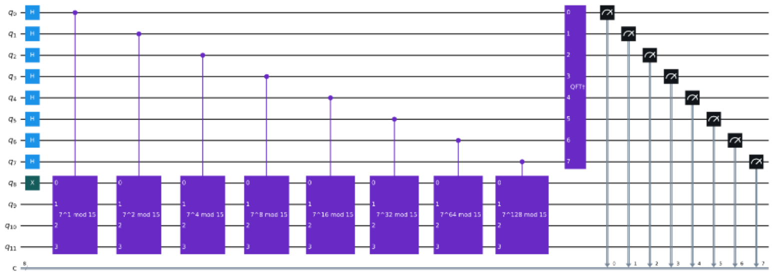
Dominik Hahn

Leverhulme-Peierls Fellow

UNIVERSITY OF OXFORD

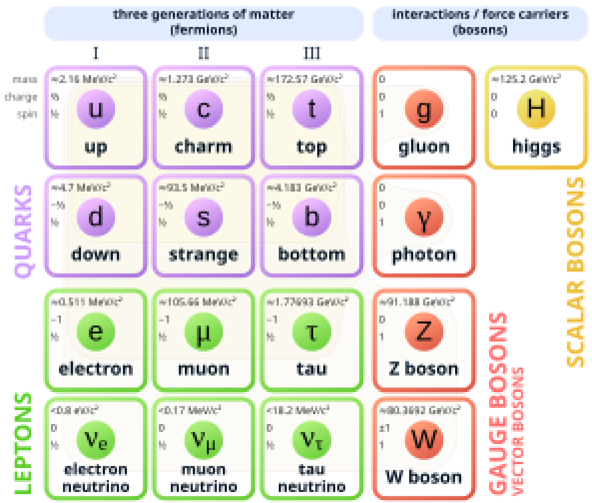
Possible applications for quantum computers

Prime factorization/ Shor's algorithm



Qiskit

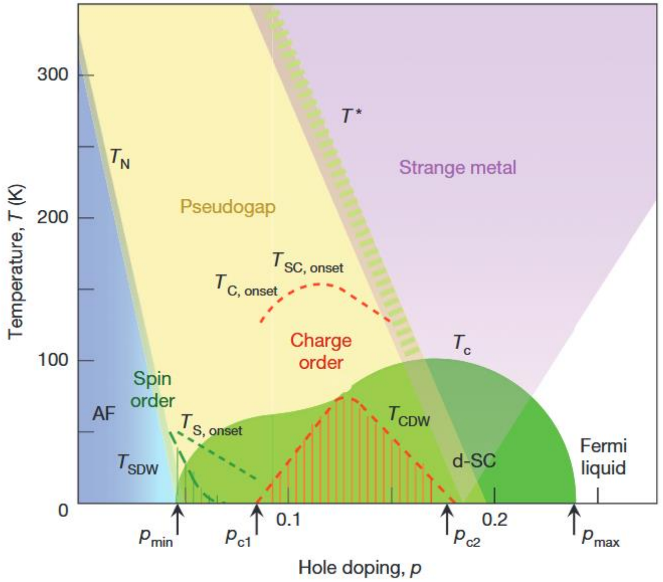
High energy physics



Wikipedia

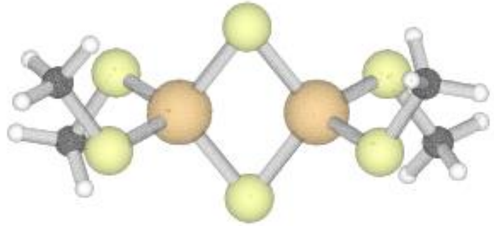
Simulation of strongly interacting quantum systems

Strongly correlated electron systems



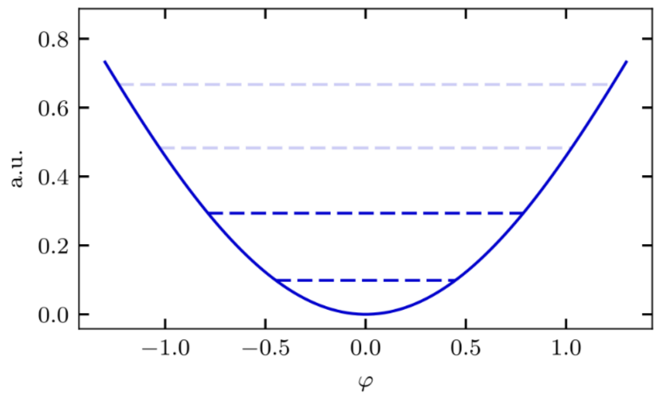
Keimer et al. Nature 518, 179-186 (2015)

Quantum chemistry



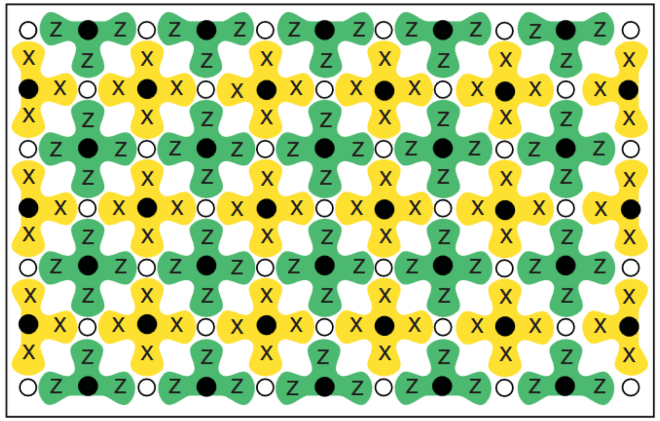
Robledo-Moreno et al. ,arXiv 2405.05068(2024)

DiVincenzo’s criteria



We need the following ingredients to build a quantum computer

DiVincenzo, arXIv:cond-mat/9612126



- A scalable systems with well-defined qubits
- Long enough coherence times
- An universal set of quantum gates
- Measurements

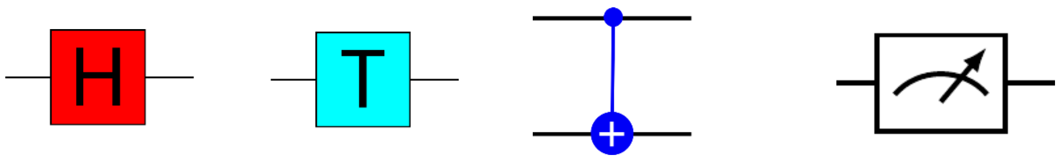
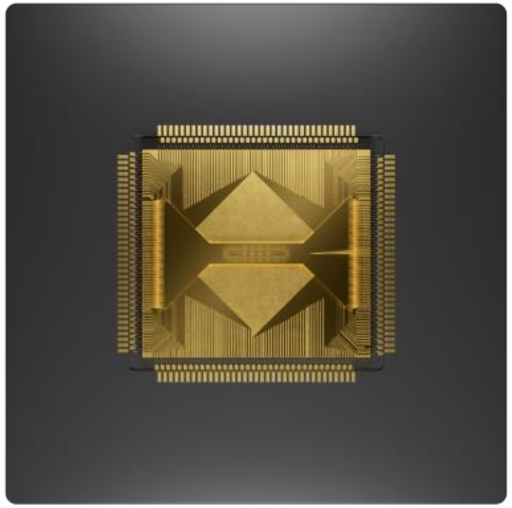


Image source: Fowler et al. PRA 86,0323324(2013)

Physical realizations

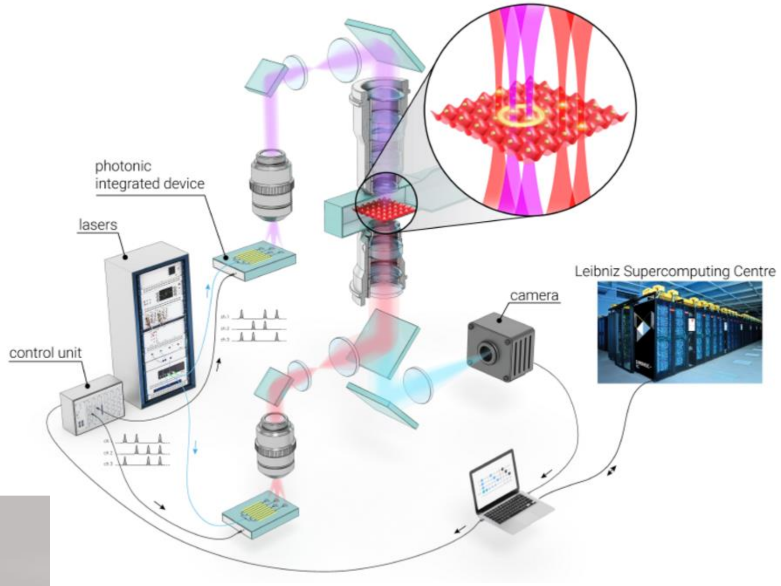
There a lots of different approaches!

Superconducting qubits (Google, IBM, Oxford Quantum circuits)



Trapped Ions (Quantinuum, IonQ, Oxford Ionics)

Topological quantum computing (Microsoft)



Ultracold atoms (QuEra Computing, Pasqal, planqc)

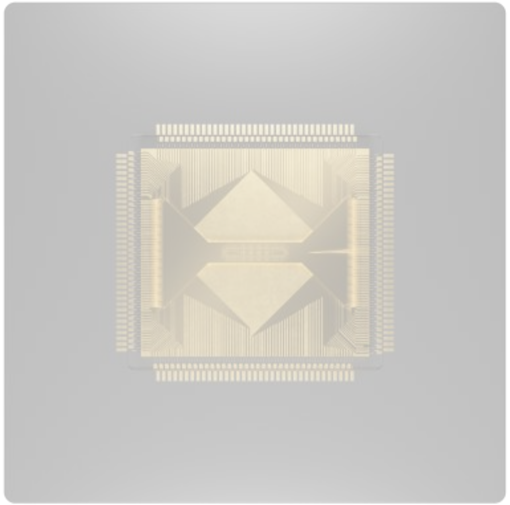
Photonic quantum computing, Silicon quantum computing …

Image sources: Google, Quantinuum, Munich Quantum Valley, Microsoft Azure

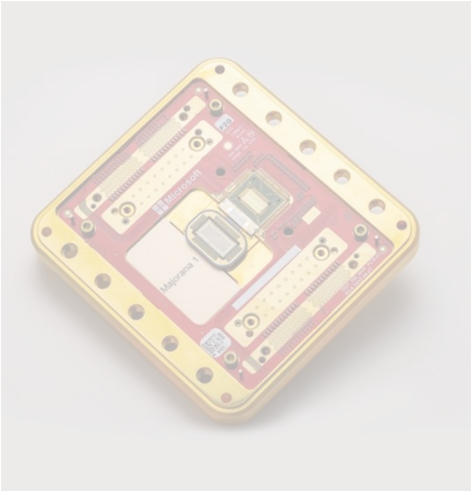
Physical realizations

There a lots of different approaches!

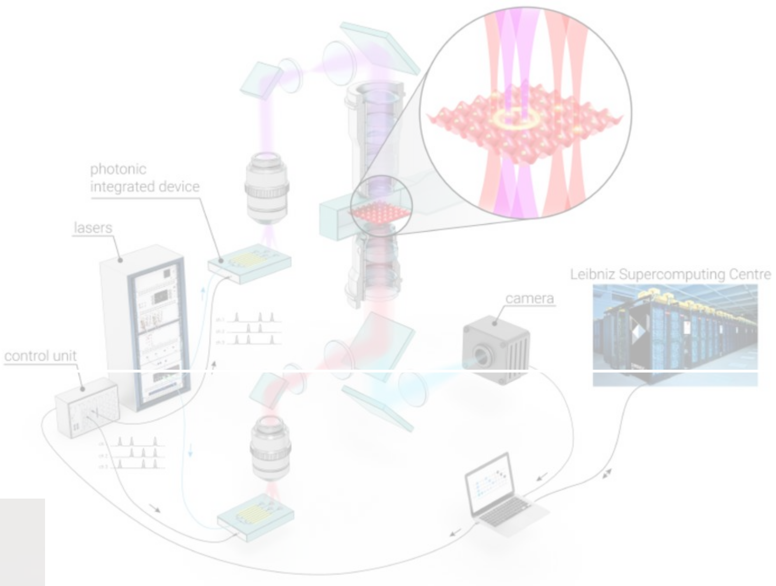
Superconducting qubits (Google, IBM, Oxford Quantum circuits)



Trapped Ions (Quantinuum, IonQ, Oxford Ionics)



Topological quantum computing (Microsoft)

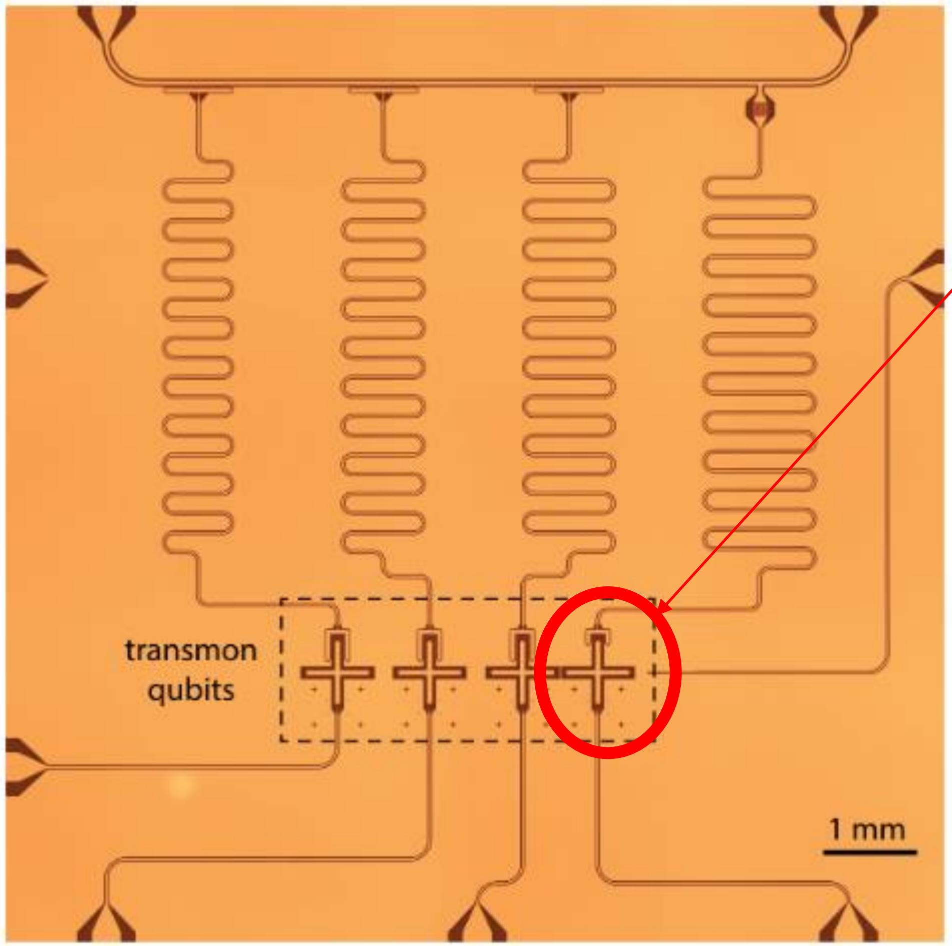


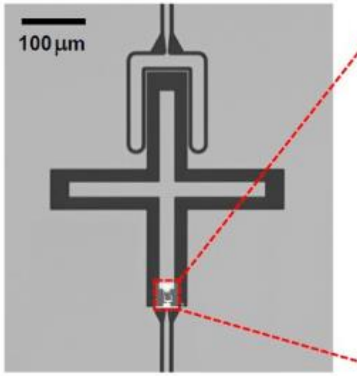
Ultracold atoms (QuEra Computing, Pasqal, planqc)

Photonic quantum computing, Silicon quantum computing ...

Image sources: Google, Quantinuum, Munich Quantum Valley, Microsoft Azure

Superconducting qubits





Josephson junction shunted between a capacitor

$$\hat{H} = 4E_C\left(\hat{n} - n_g\right)^2 - E_J \cos\hat{\varphi}$$

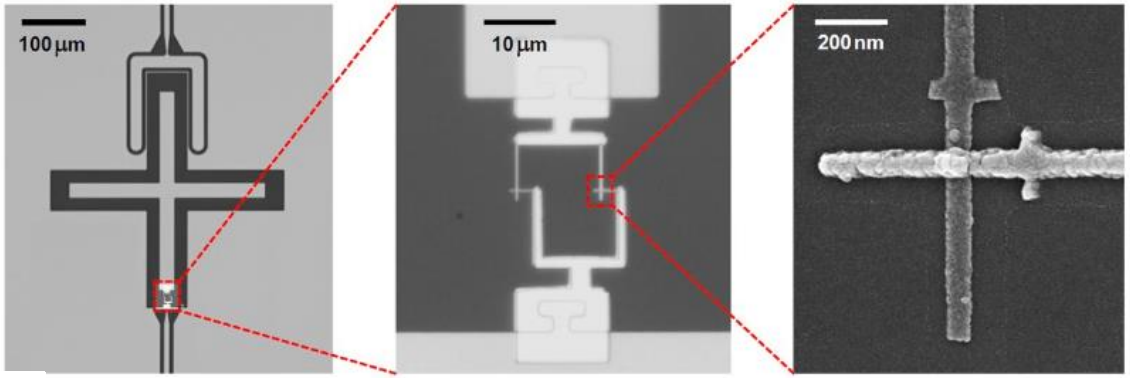
E_J Josephson energy
E_C Capacity energy
$\hat{n}$ Number of Cooper pairs
$\hat{\varphi}$ Josephson phase

Anharmonic oscillator!
Large Capacity energy to be more noise resilient

Image source: Roth et al. arXiv2106.11352

Superconducting qubits

$$\hat{H} = 4E_C\,(\hat{n} - n_g)^2 - E_J \cos\hat{\varphi}$$



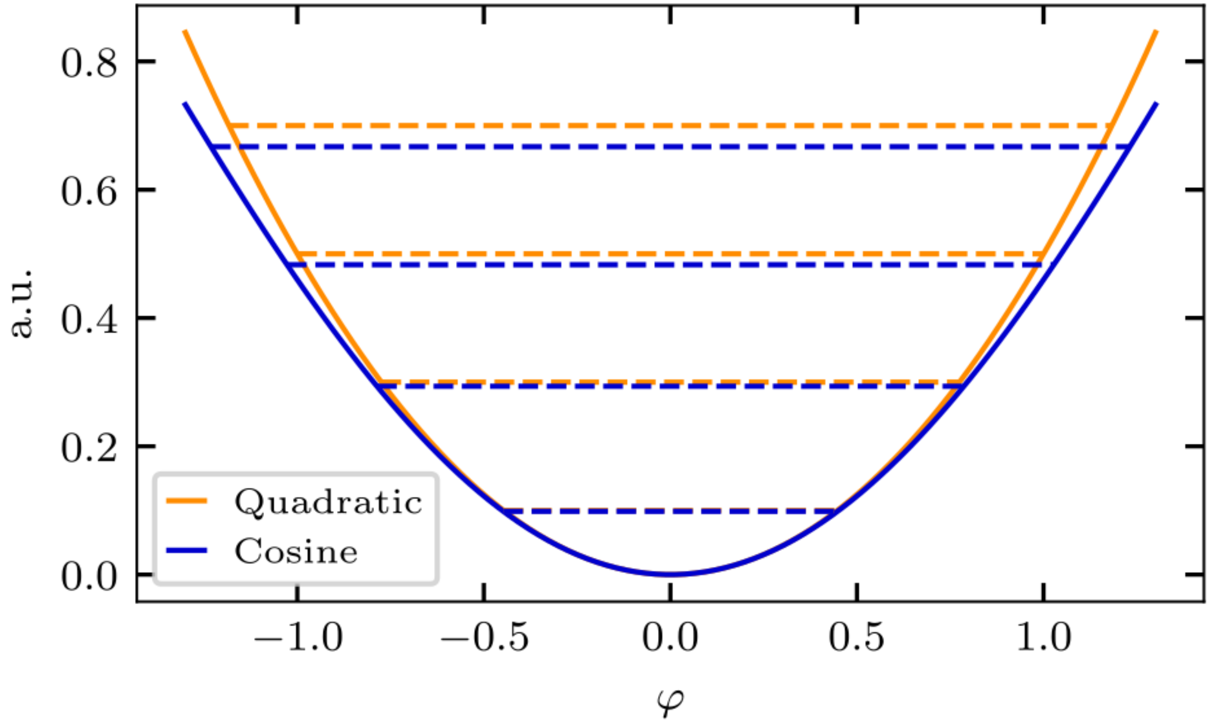
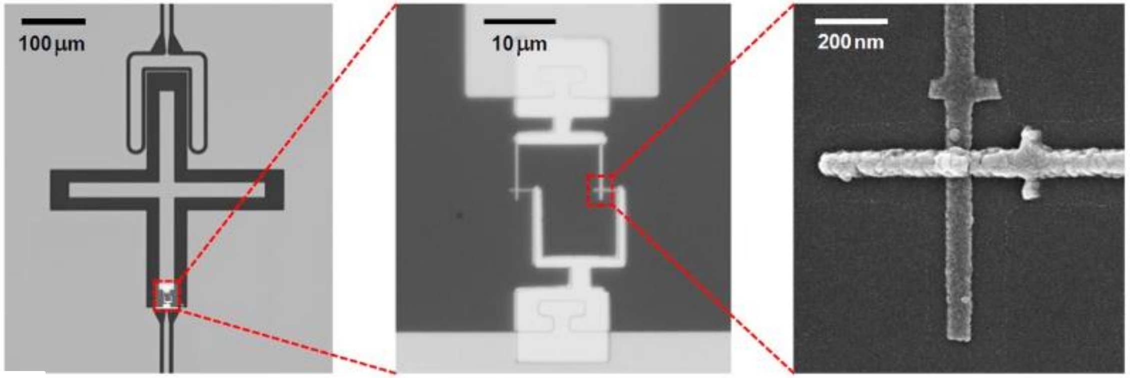
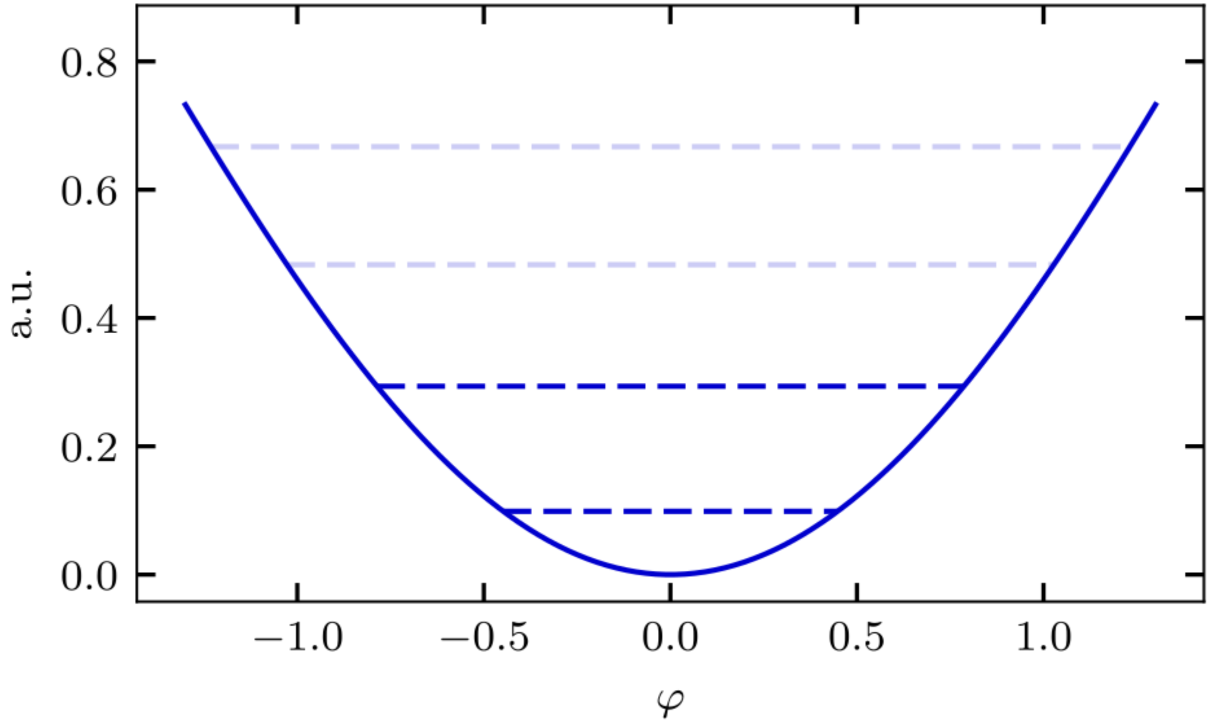


Image sources: Roth et al. arXiv2106.11352

Superconducting qubits

$$\hat{H} = 4E_C\left(\hat{n} - n_g\right)^2 - E_J\cos\hat{\varphi}$$

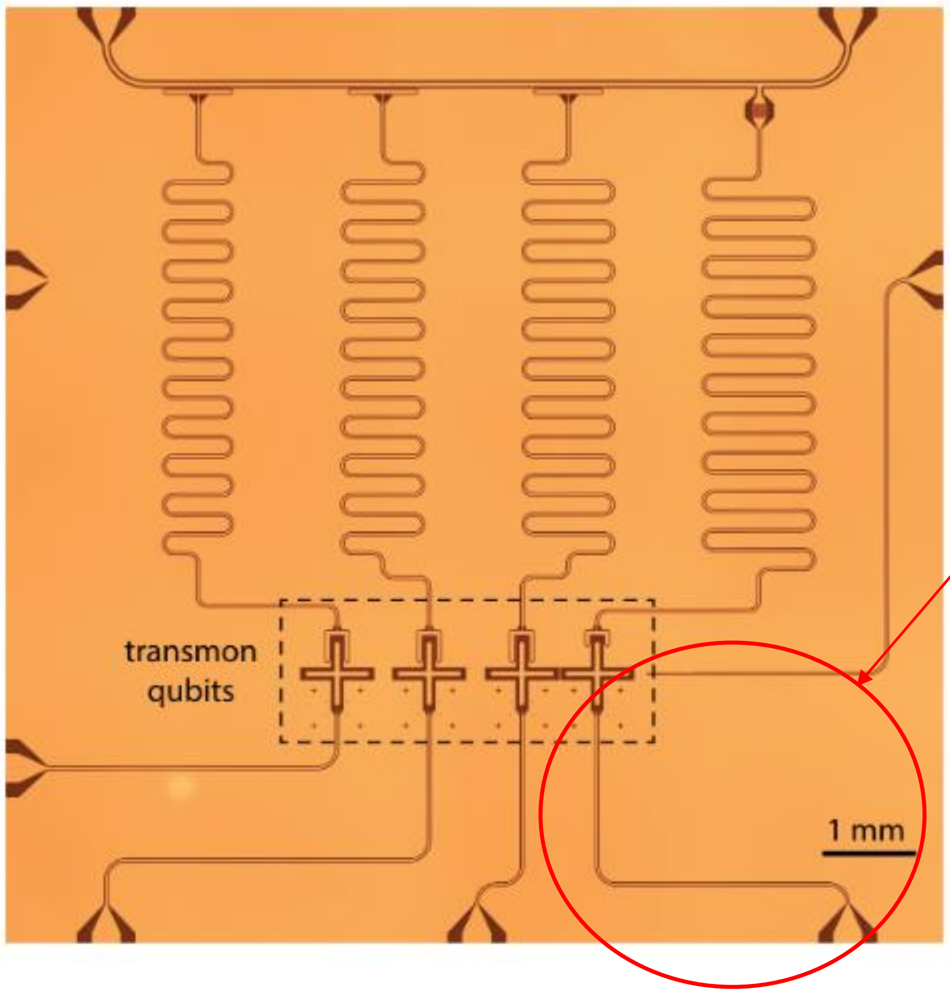




- The lowest two eigenstates are the states of the qubit
- Cosine potential leads to slight anharmonicity

Image sources: Roth et al. arXiv2106.11352

Applying quantum operations



- Coupling of the transmon qubits to resonators → Hybridization between photons and qubit state
- Pulse in the resonator allows to change state of the qubit
- Anharmonicity prevents excitation of higher eigenstates

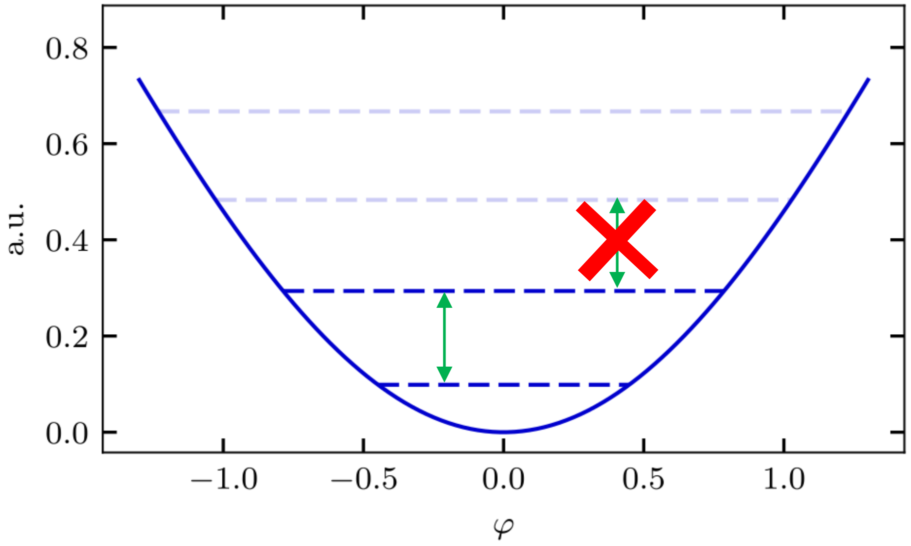
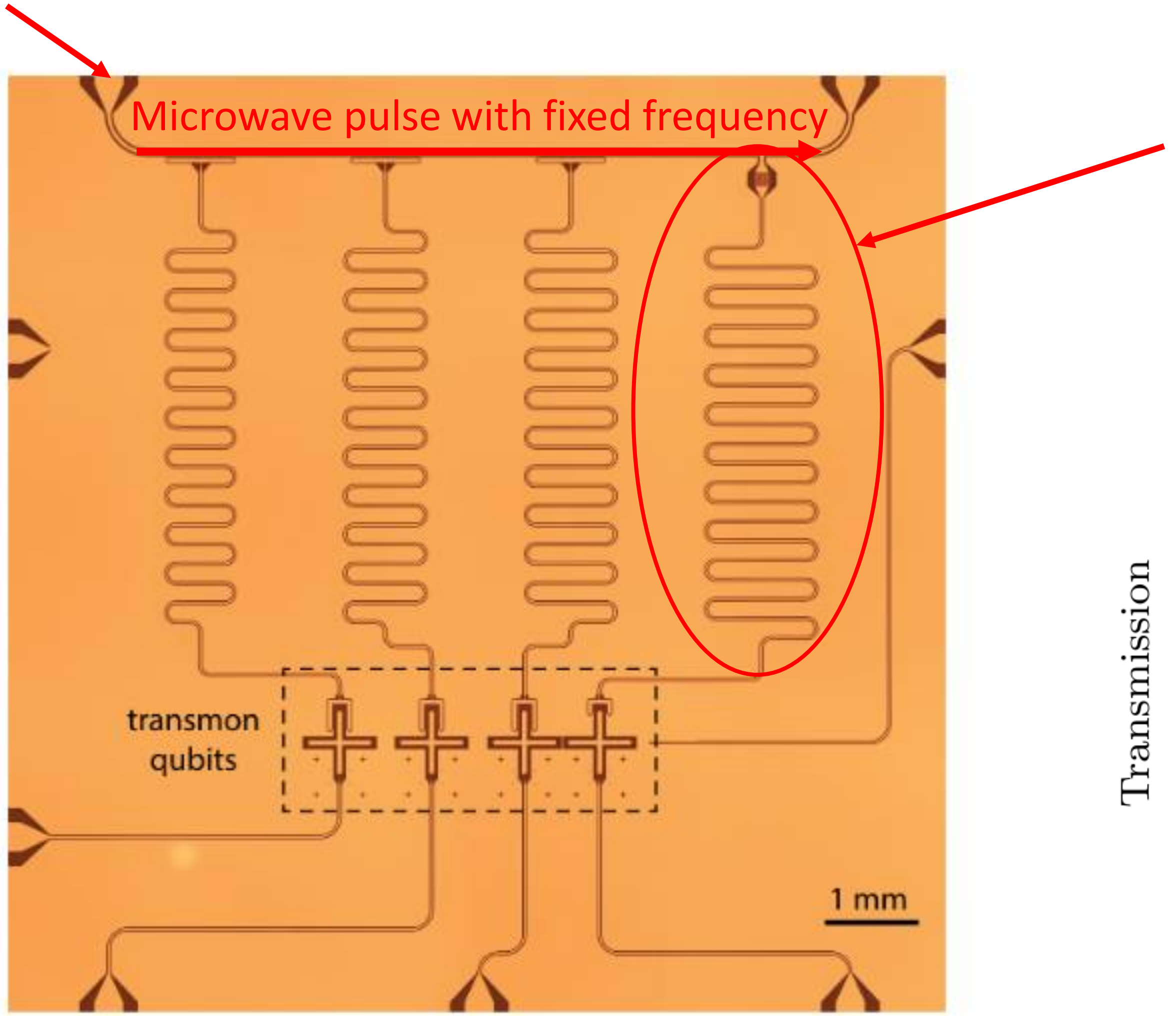


Image sources: Roth et al. arXiv2106.11352

Measurements



- Coupling of resonator induces shift in the transition frequency

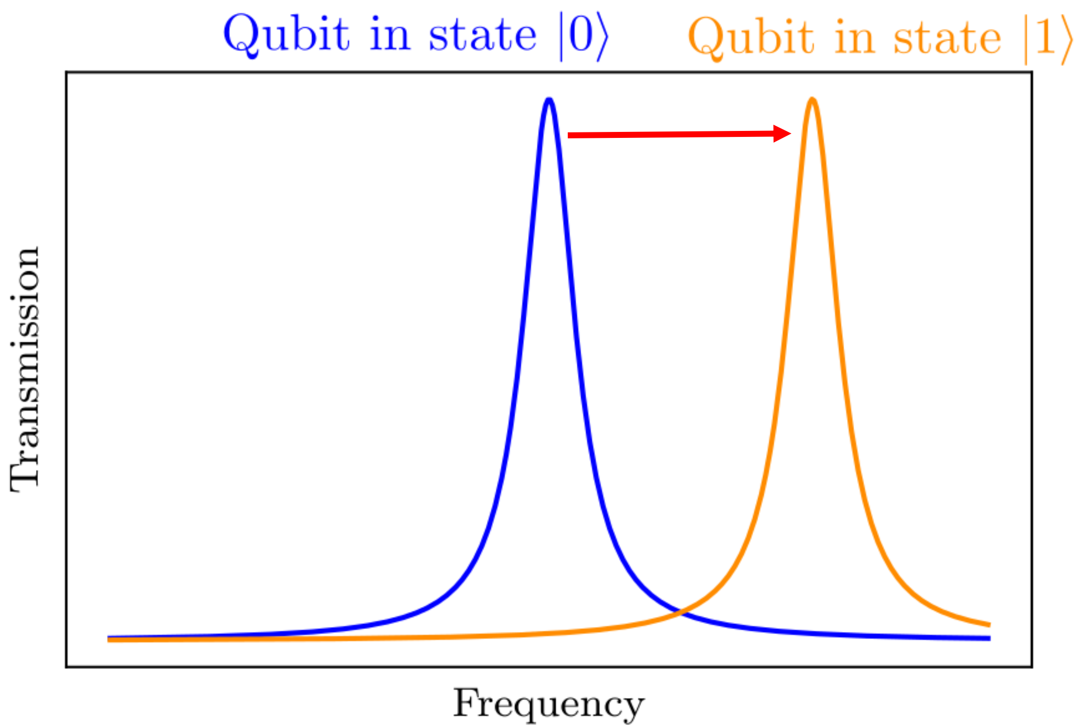
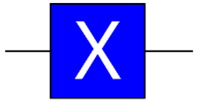
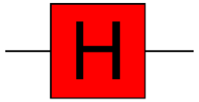
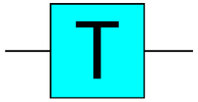


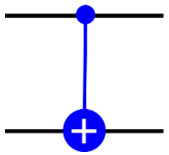
Image sources: Roth et al. arXiv2106.11352

Basic operations on a quantum computer

- Graphic notation to illustrate quantum circuits
- Qubits: Denoted by a single line
- Quantum operations:
- Unitary gates
    - X gate 

$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
    - Hadamard gate 

$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
    - T gate 

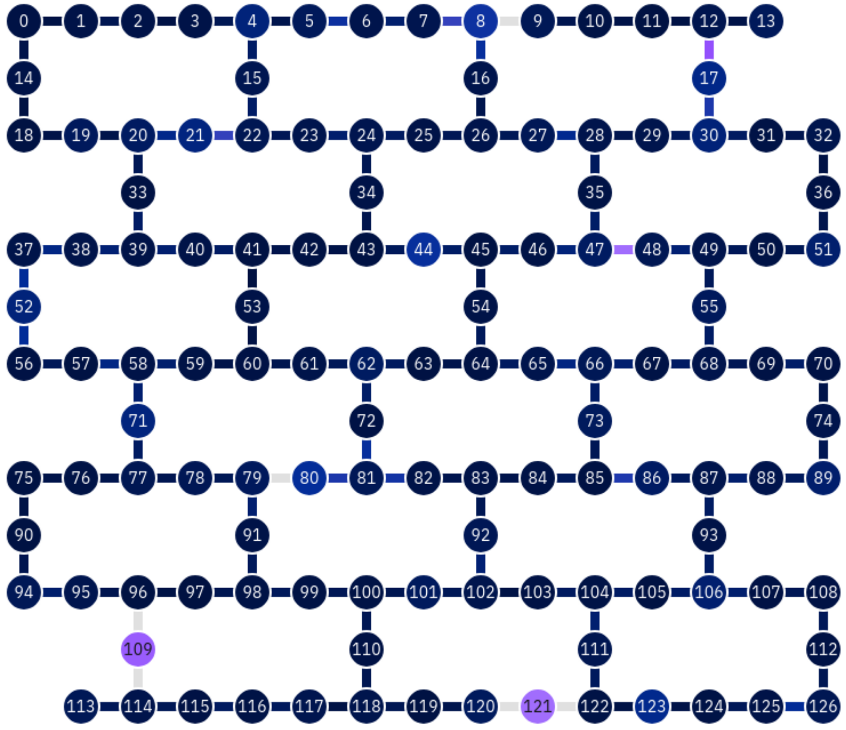
$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$
    - CNOT gate 
$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

→ These gates are sufficient to generate arbitrary quantum gates

- Measurements 

Programming quantum computers

- Quantum circuits can be programmed using a python library (qiskit)
- Quantum circuits are compiled using native gate operations
- Small scale simulation on classical computers possible
- Access to real hardware via Cloud services (AWS Braket, IBM quantum)



Programming quantum computers is similar to programing classical computers!

Creating a Bell state

- Create a Bell state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$

```
from qiskit import QuantumCircuit
```

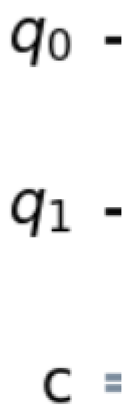
```
circuit = QuantumCircuit(2,2)
```

Create a quantum circuit object

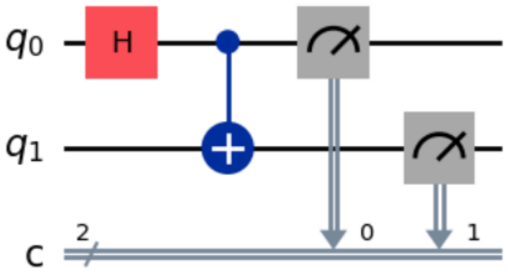
Put the first qubit in a superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$

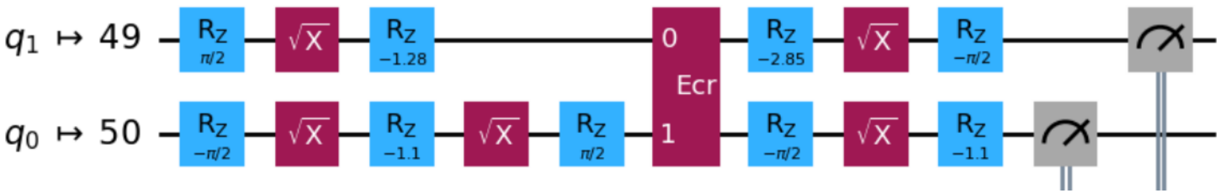
Controlled operation on the second qubit

Measure the qubits



Execute the quantum circuit



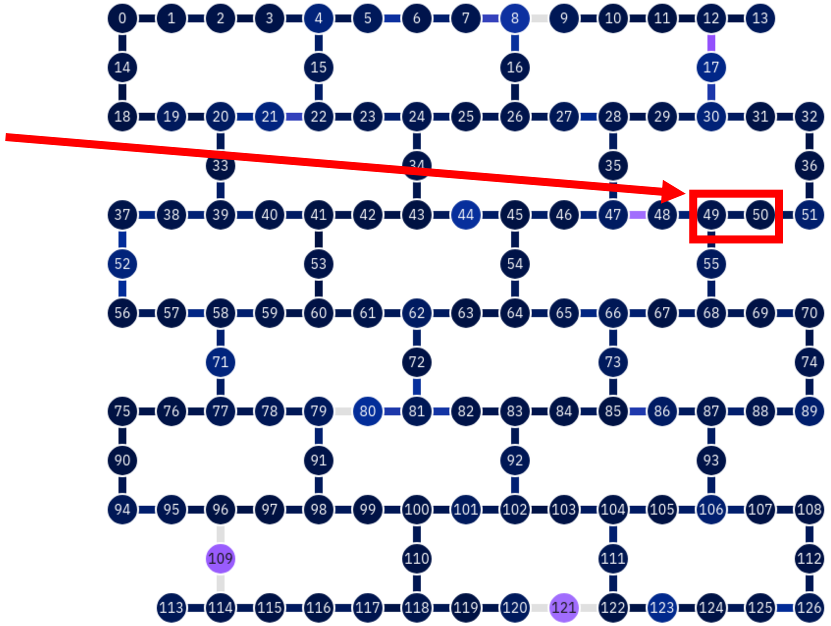



```
service = QiskitRuntimeService()
backend_real = service.least_busy(simulator=False)
```

```
from qiskit.transpiler import generate_preset_pass_manager
pm = generate_preset_pass_manager(backend=backend)
compiled_circuit = pm.run(circuit)
```

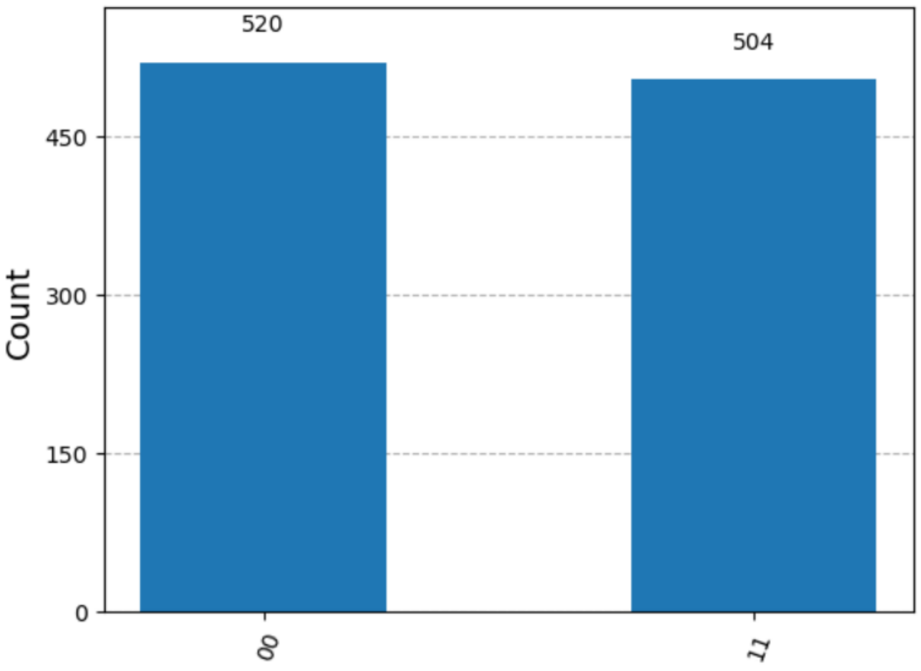
```
from qiskit_ibm_runtime import SamplerV2 as Sampler
sampler = Sampler(mode=backend)
job = sampler.run([(compiled_circuit)])
result=job.result()[0]
```

```
counts=pub_result.data.c.get_counts()
plot_histogram(counts)
```



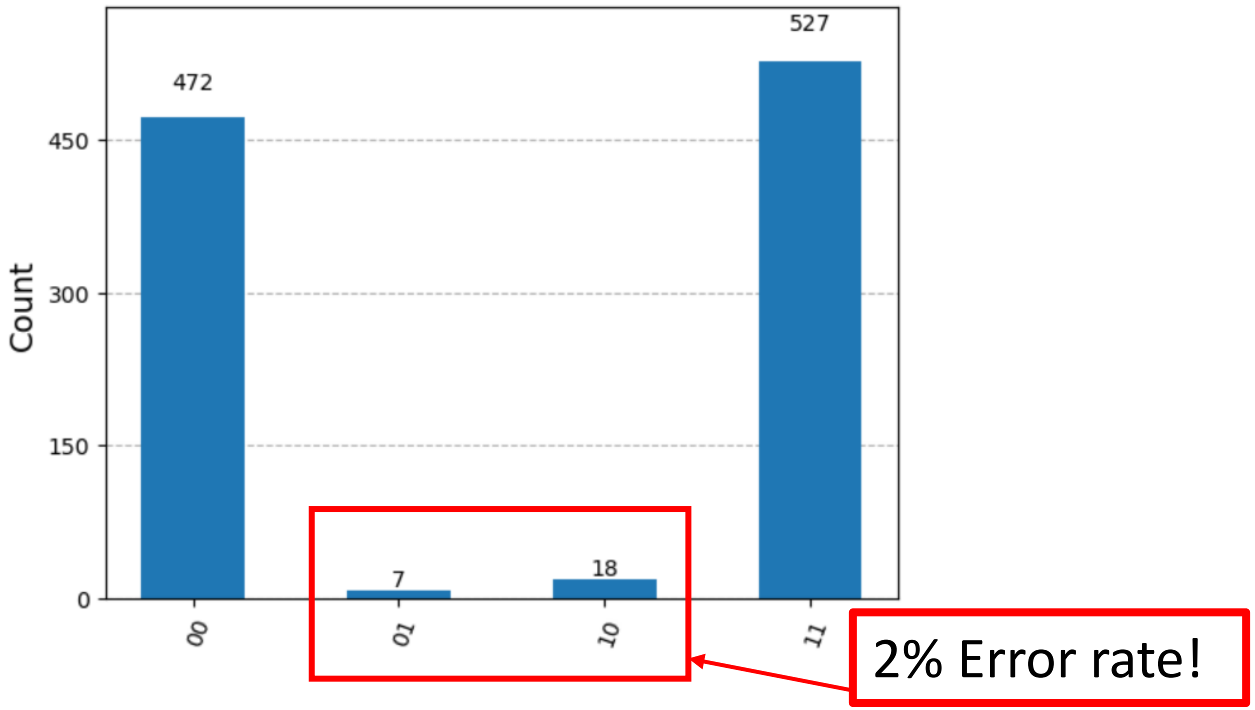
Results

Classical Simulation



50% probability to either measure $|00\rangle$ or $|11\rangle$

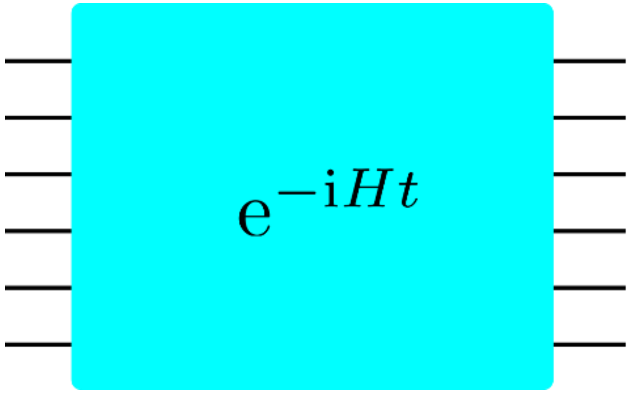
Result on a real quantum device



→ We need to deal with errors!

Advanced example: Quantum dynamics

- One of the most promising applications: Simulating other quantum systems
  Lloyd, Science 1996
- Basic approach: Trotter decomposition



- Dimension of the Hilbert space scales exponentially with number of qubits

→Classical simulation impossible for more than 30 qubits

→Benchmark for performance of quantum computers

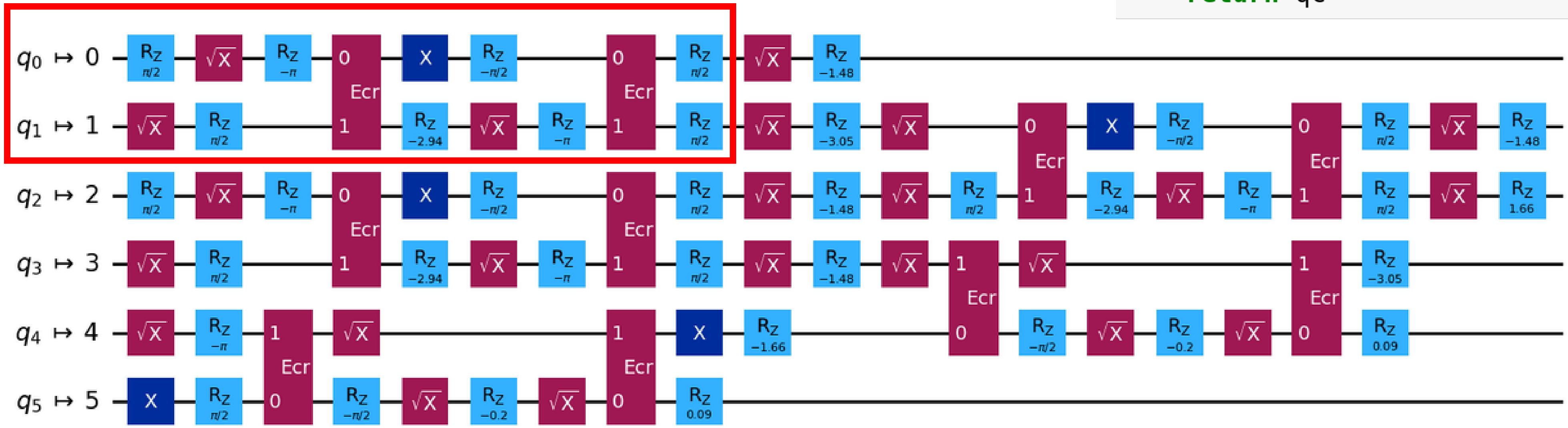
Time simulation of a spin model

Time evolution for the Hamiltonian: $\sum_i X_i X_{i+1} + 0.9\ Z_i$

Code for the Trotter evolution

```
def Trotterstep(qc,N,t):
    #Create a Trottergate
    H = (X^X) + 0.45*(Z^I)+0.45*(I^Z)
    pauli_ev_gate = PauliEvolutionGate(H, time=t)
    #Apply Gate on qubits
    #Apply it on even bonds
    for j in range(0, N-1, 2):
        qc.append(pauli_ev_gate,(j,j+1))
    #Apply it on odd bonds
    for j in range(1, N-1, 2):
        qc.append(pauli_ev_gate,(j,j+1))
    return qc
```

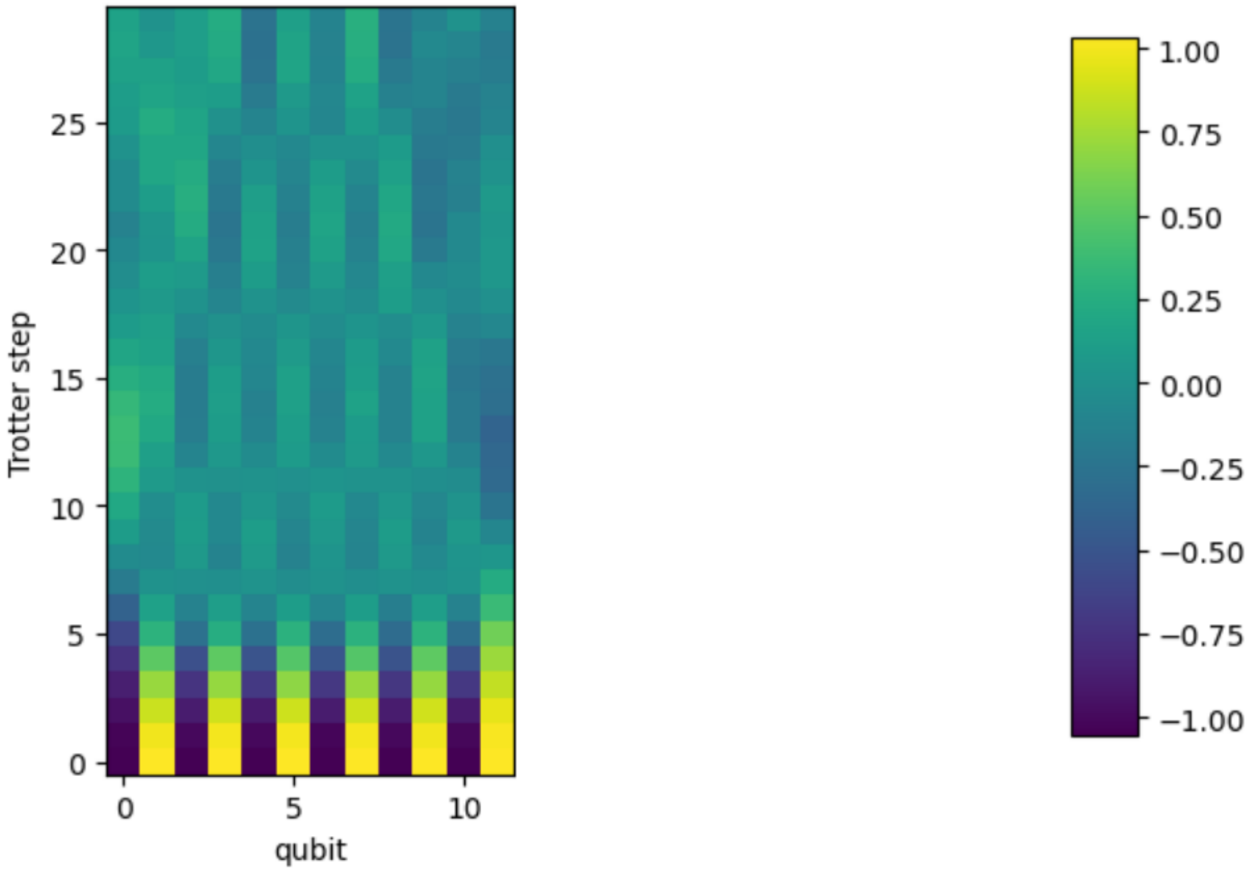
Compiled circuit for one Trotter step



Results for Time evolution

Time evolution for the Hamiltonian: $\sum_i X_i X_{i+1} + 0.45\, Z_i$
Results for the onsite magnetization of the results

Classical simulation **Quantum computer**

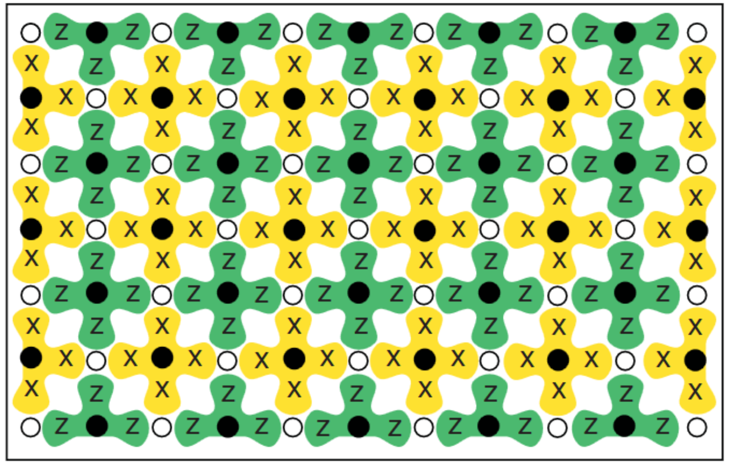


- Oscillations captured for short times
- Multiple steps: Worse accuracy
- Time evolutions is relatively robust in comparison to the previous experiment

Active area of research: Fighting the errors

Quantum error correction

- Encode redundant information to correct errors
- Requires small error rates and large number of qubits



Quantum error mitigation

- Learn the errors and interpolate to zero noise

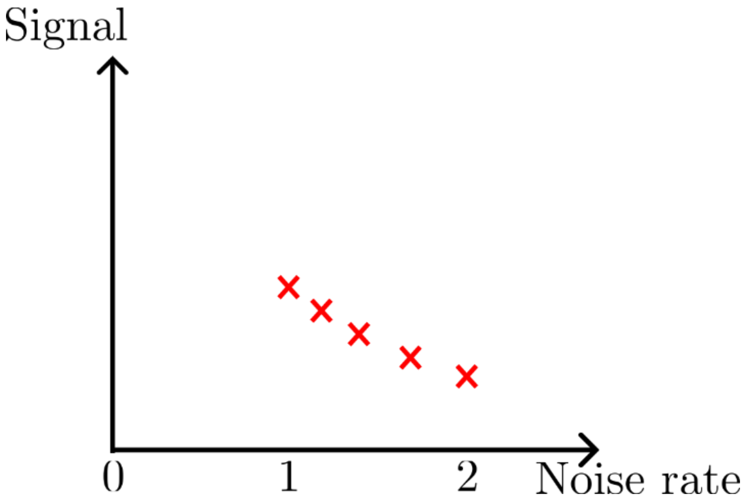
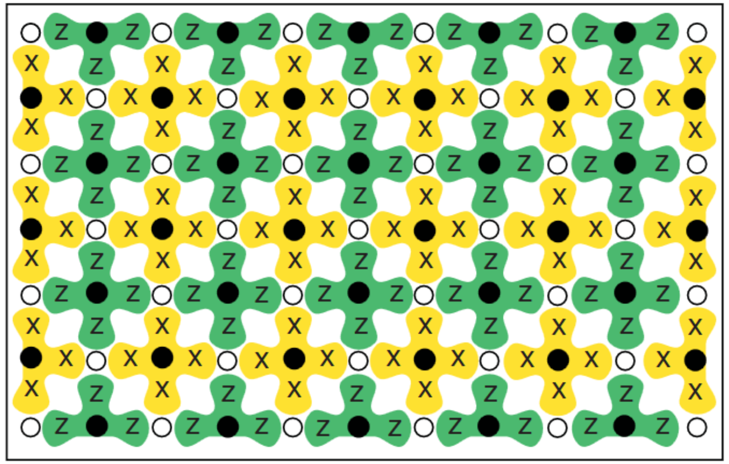


Image source: Fowler et al. PRA 86,0323324(2013)

Active area of research: Fighting the errors

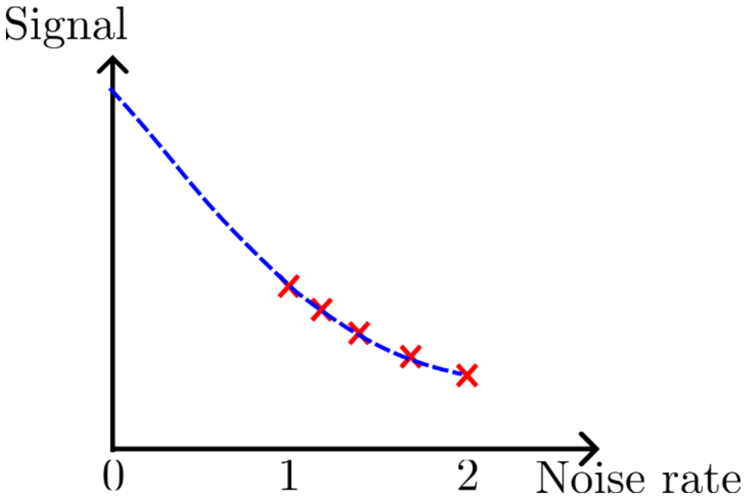
Quantum error correction

- Encode redundant information to correct errors
- Requires small error rates and large number of qubits



Quantum error mitigation

- Learn the errors and interpolate to zero noise

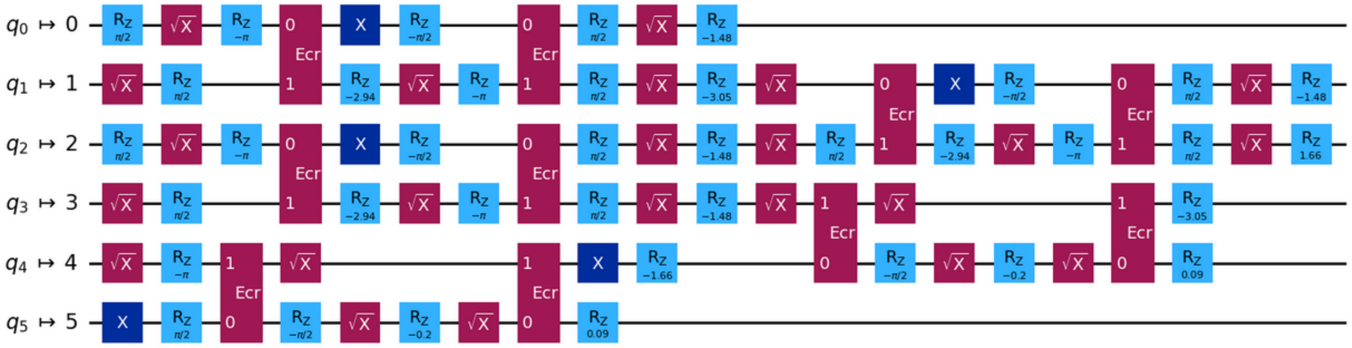


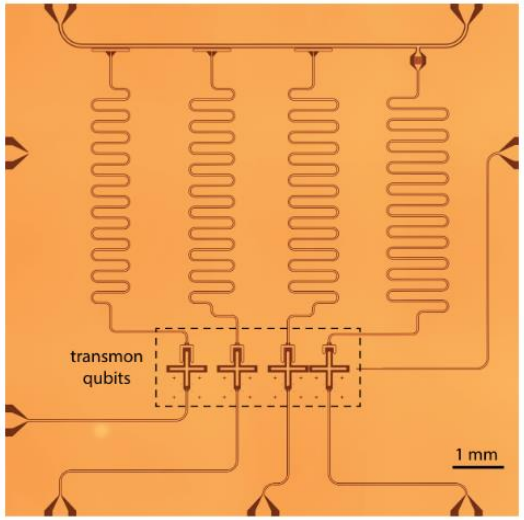
- Works fairly well for algorithms as time evolution dynamics

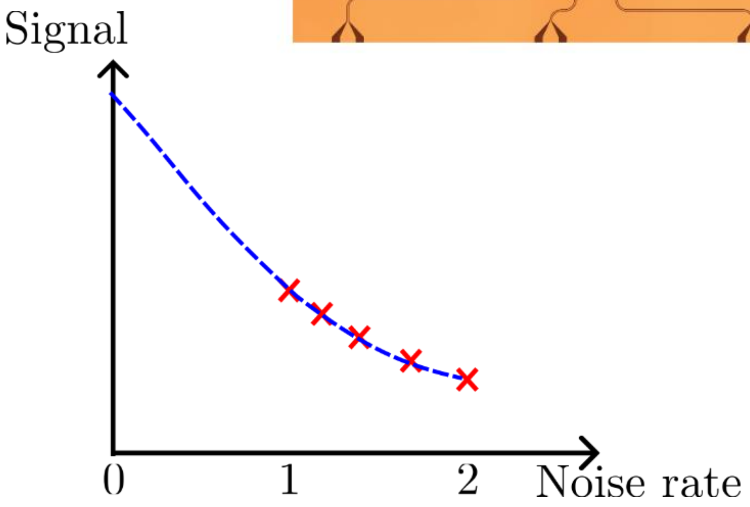
Image source: Fowler et al. PRA 86,0323324(2013)

Conclusion

- Programming quantum computer is “simple”
- Main challenge is understanding the errors







Do it yourself!

```
def Trotterstep(qc,N,t):
    #Create a Trottergate
    H = (X^X) + 0.45*(Z^I)+0.45*(I^Z)
    pauli_ev_gate = PauliEvolutionGate(H, time=t)
    #Apply Gate on qubits
    #Apply it on even bonds
    for j in range(0, N-1, 2):
        qc.append(pauli_ev_gate,(j,j+1))
    #Apply it on odd bonds
    for j in range(1, N-1, 2):
        qc.append(pauli_ev_gate,(j,j+1))
    return qc
```

Image source: Roth et al. arXiv2106.11352