

# Building a Compiler for Quantum Computers

Matthew Treinish

Software Engineer - IBM Research

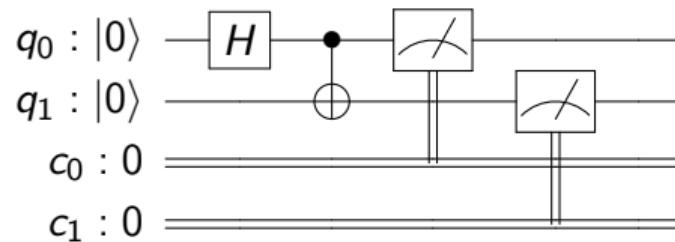
[mtreinish@kortar.org](mailto:mtreinish@kortar.org)

`mtreinish` on Freenode

<https://github.com/mtreinish/quantum-compilers>

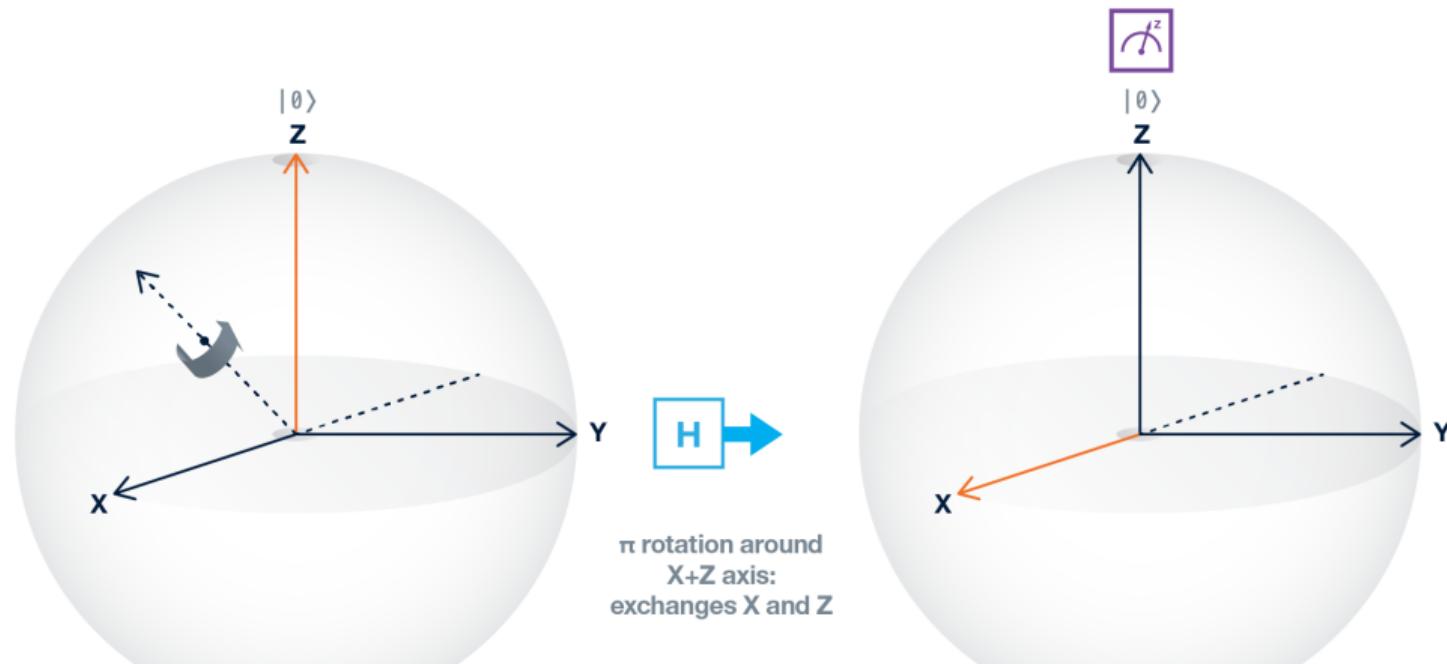
January 17, 2020

# Quantum Circuits



# Quantum Gates

- ▶ Quantum gates
- ▶ Gates are reversible
- ▶ Each gate represents a unitary matrix that gets multiplied against state of qubits it operates on





# OpenQASM<sup>12</sup>

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg c[2];
h q[0];
cx q[0],q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];
```

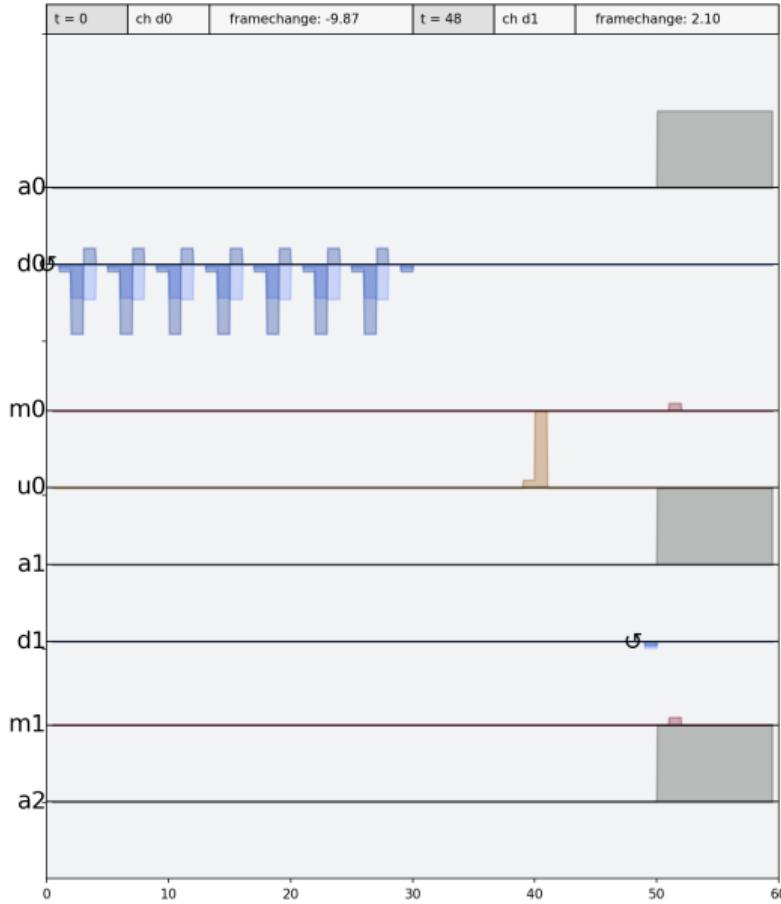
- ▶ Open Quantum Assembly Language
- ▶ An option
- ▶ Mostly used as a transport layer to save circuits or share circuits

---

<sup>1</sup><https://arxiv.org/abs/1707.03429>

<sup>2</sup><https://github.com/Qiskit/openqasm>

# Pulse Level Programming

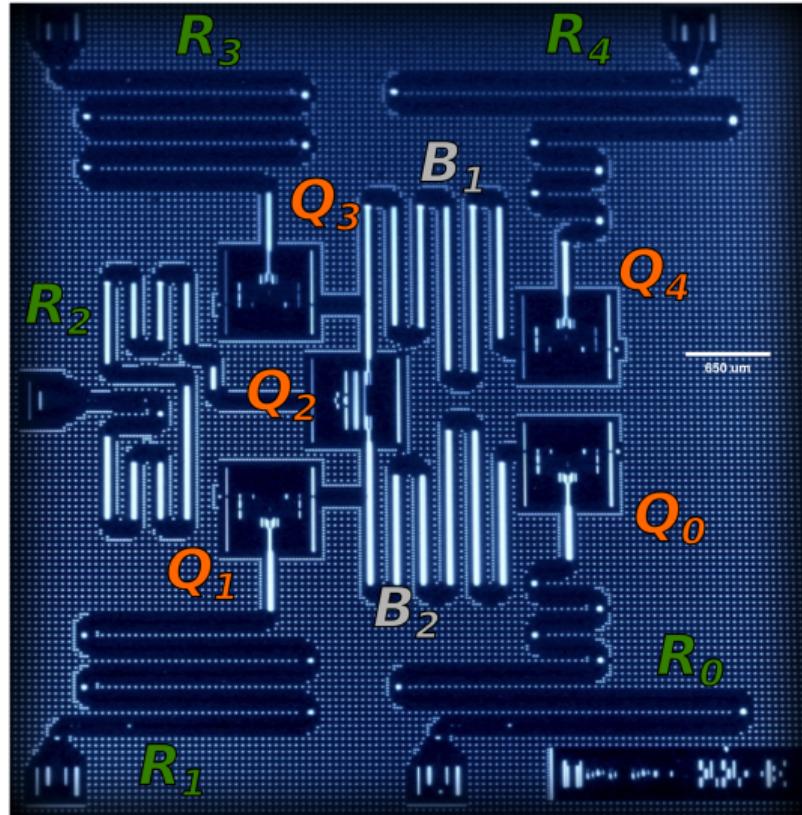


- ▶ A layer below the circuit model is to use pulses
- ▶ Each quantum gate is defined as a pulse that gets applied to the qubits
- ▶ Enables you to tweak the definition of gates
- ▶ Mostly used for physics research or hardware characterization

# Qubit Connectivity

- ▶ Qubits in a device have limited connectivity
- ▶ For multi-qubit gates this means we can only run them between those qubits
- ▶ Need to use SWAP gates to move state around

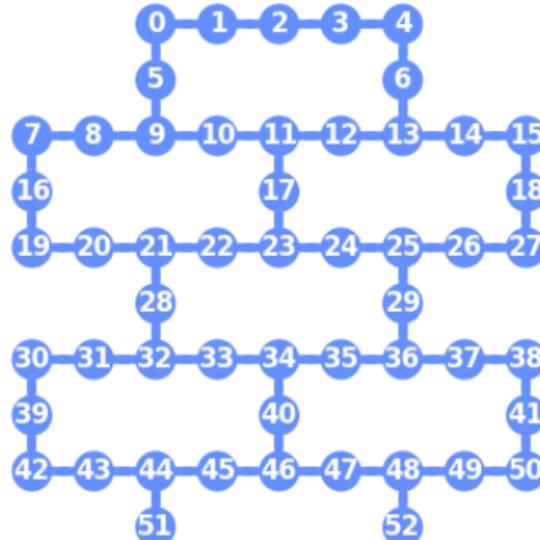
IBM Q 5 Yorktown



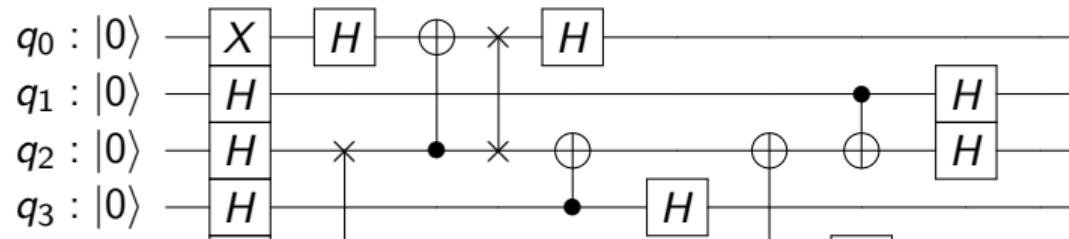
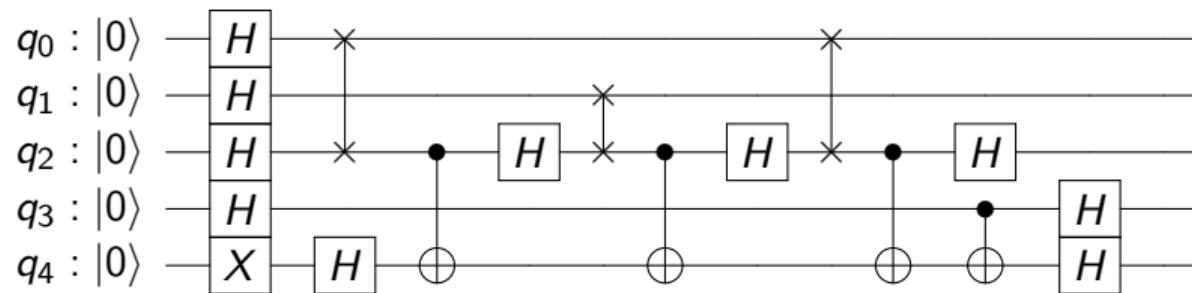
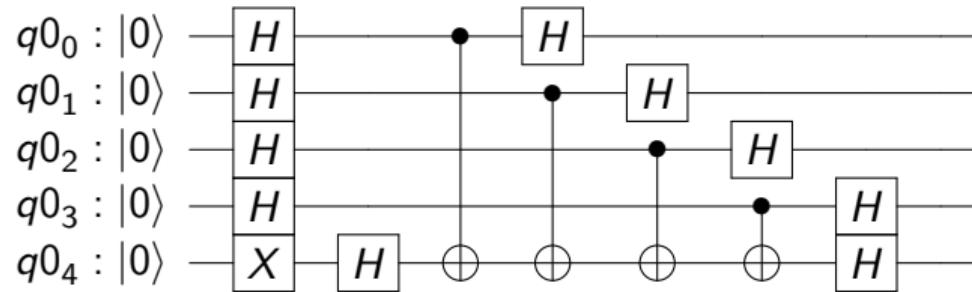
# Qubit Connectivity

## IBM Q 53 Rochester Coupling Map

- ▶ Qubits in a device have limited connectivity
- ▶ For multi-qubit gates this means we can only run them between those qubits
- ▶ Need to use SWAP gates to move state around



# Mapping



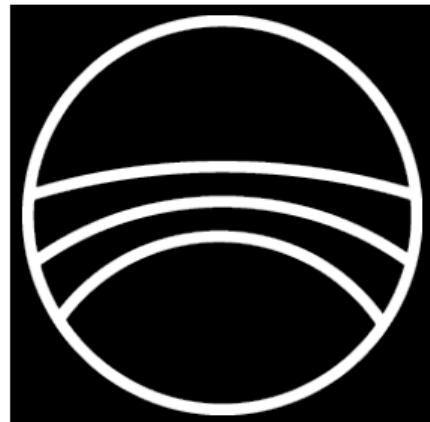
## Basis Gates

# Unrolling



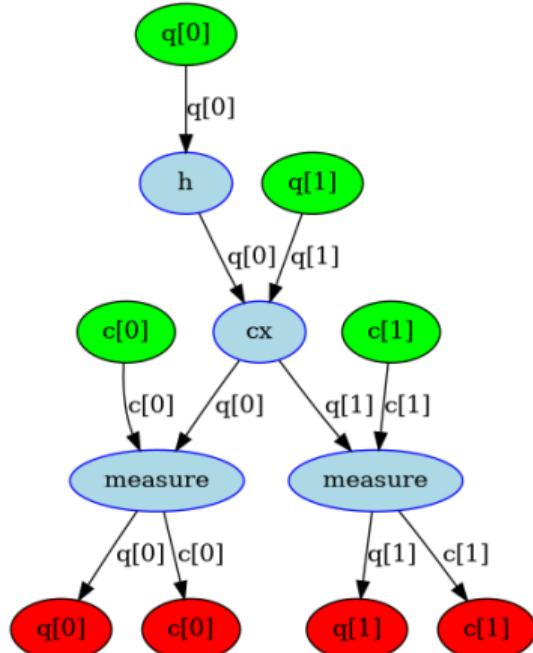
# Qiskit Terra

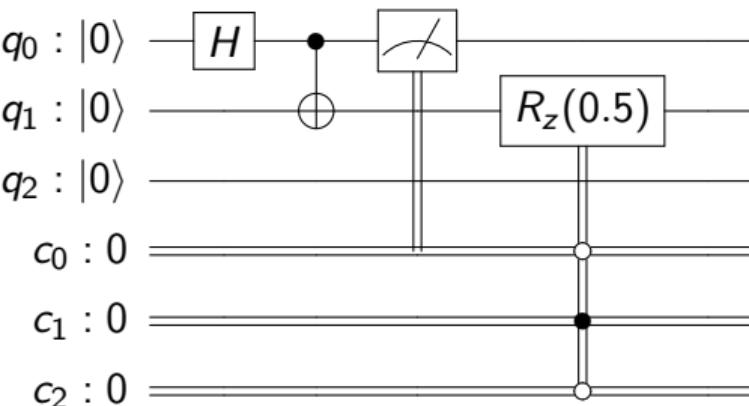
- ▶ Is the base layer for working with quantum computers provides interface to hardware and simulators
- ▶ Provides an SDK for working with quantum circuits
- ▶ Compiles circuits to run on different backends
- ▶ Designed to be backend agnostic and work with any quantum hardware or simulator
- ▶ Written in Python



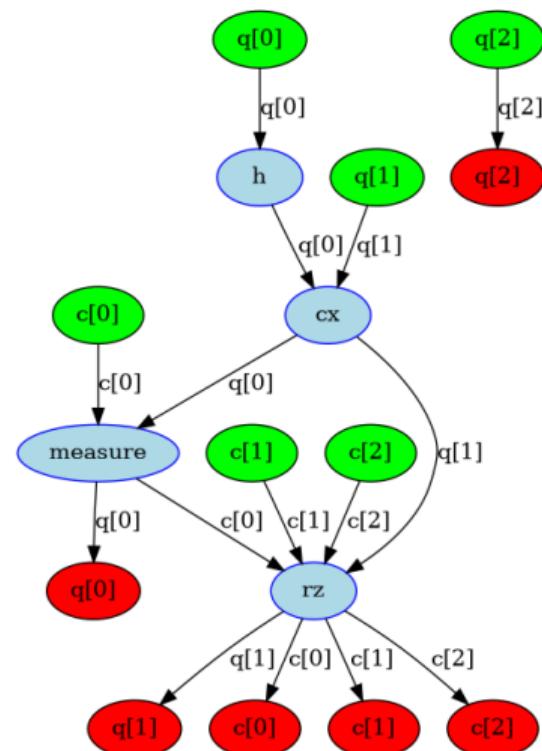
# The DAG

- ▶ Compiler represents circuits as a DAG
- ▶ Each node in the DAG is an operation, an input, or an output
- ▶ Each edge indicates data flow between nodes
- ▶ Makes flow of information between operations explicit

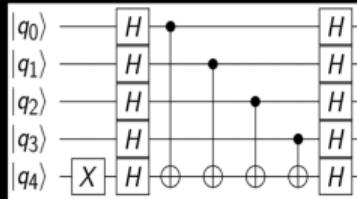




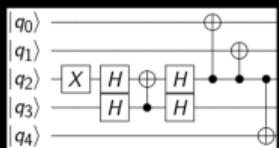
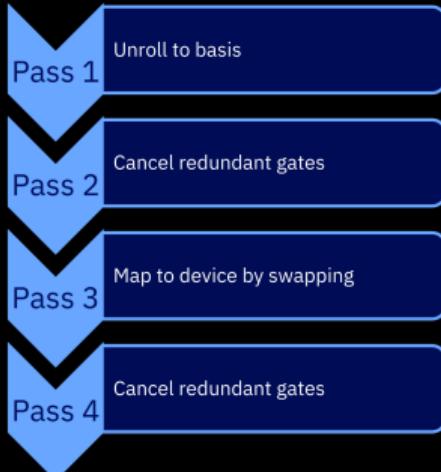
$\rightarrow$



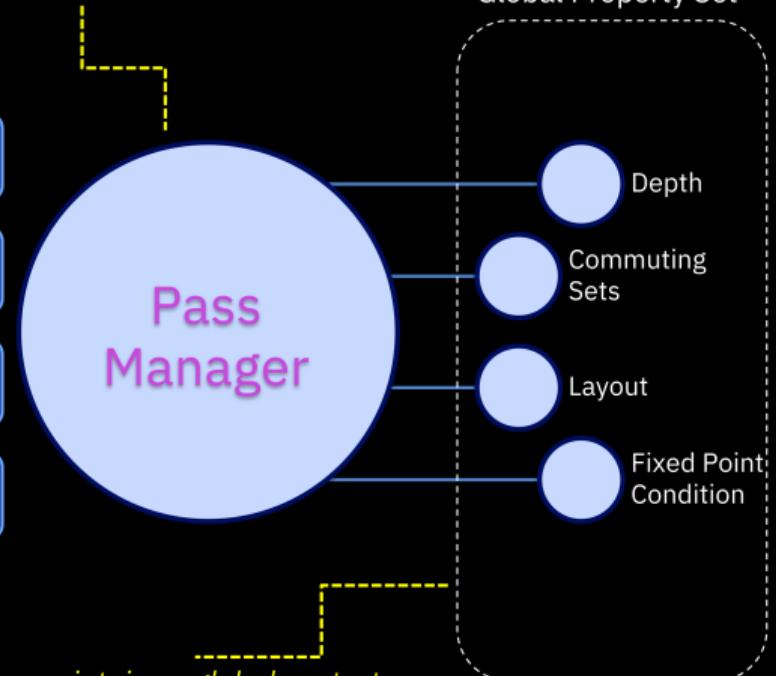
# Transpiler Architecture



*Each pass does one small, well-defined task.*

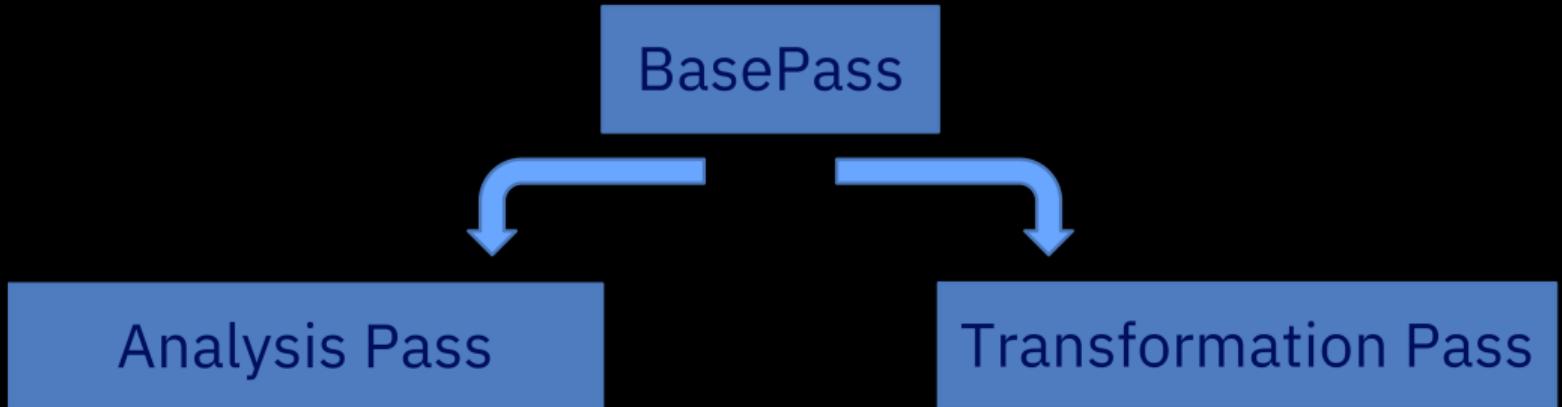


*The complexity of scheduling is solely in the pass manager, to keep passes simple*



*The pass manager maintains a global context about the circuit as it runs through the pipeline*

# Transpiler Architecture



- Read-only Access to DAG
- Write Access to Property Set

- Write Access to DAG
- Read-only Access to Property Set

Example (commutation relationships):

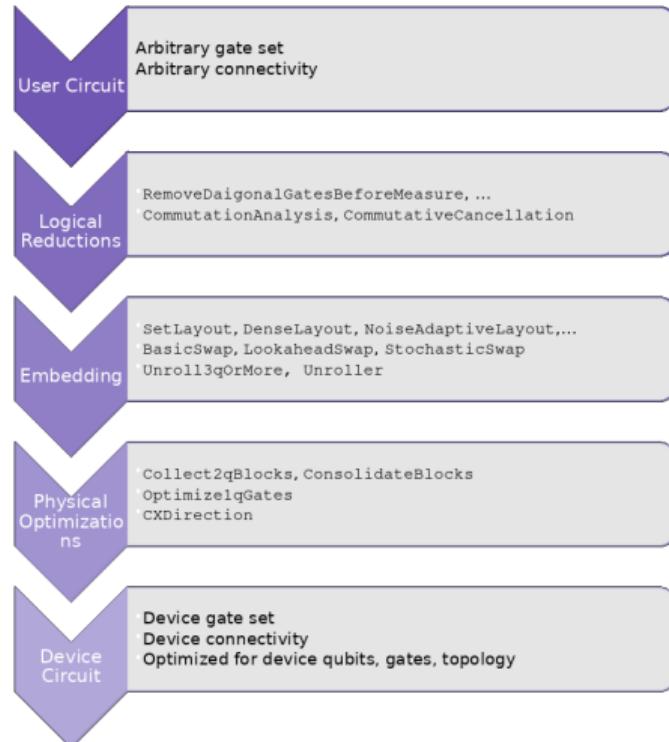
Pass A is an Analysis Pass that inspects the graph and finds nodes that can commute.

Pass B is a Transformation Pass that dissolves those edges that commute (are false dependencies).

Pass B requires Pass A, and indicates this to the Pass Manager

# Passmanager Stages

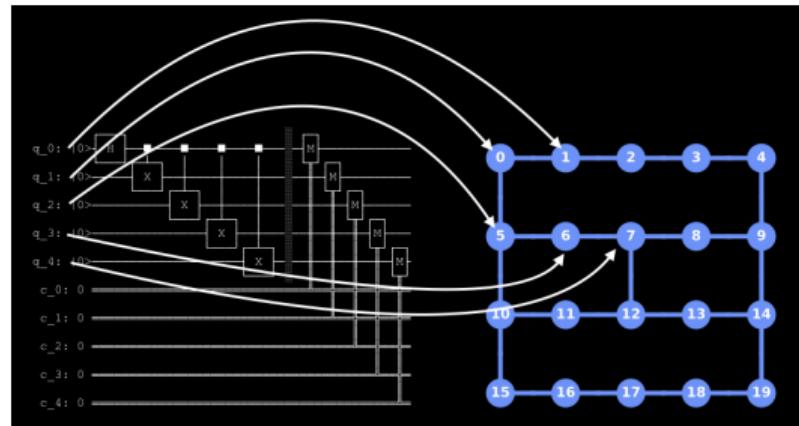
- ▶ Stages
  - ▶ Logical Reductions
  - ▶ Embedding
    - ▶ Layout
    - ▶ Mapping/Routing
    - ▶ Unrolling
  - ▶ Physical Reductions
- ▶ Pluggable and extensible



# The Unroller

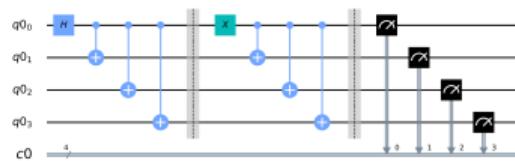
# Layout

- ▶ Initial mapping to physical qubits
- ▶ Multiple options included
  - ▶ TrivialLayout
  - ▶ DenseLayout
  - ▶ NoiseAdaptiveLayout

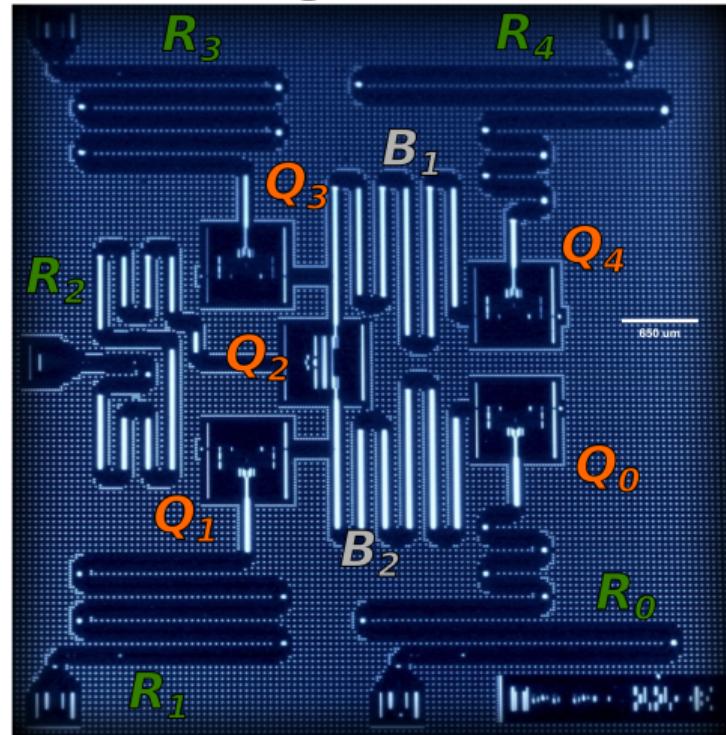


# Layout Example

Logical Circuit



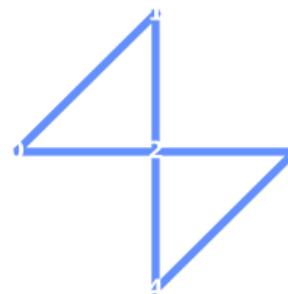
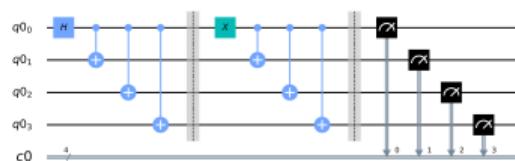
Target Device



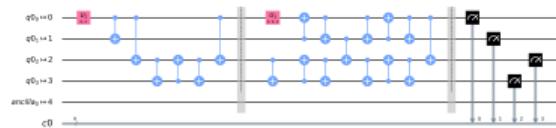
# Layout Example

Target Device

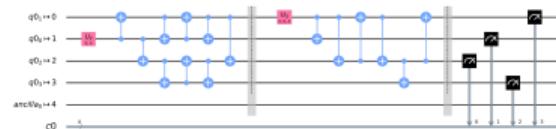
Logical Circuit



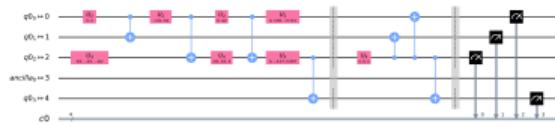
optimization\_level=1  
TrivialLayout



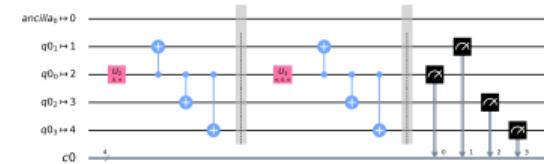
optimization\_level=2  
DenseLayout



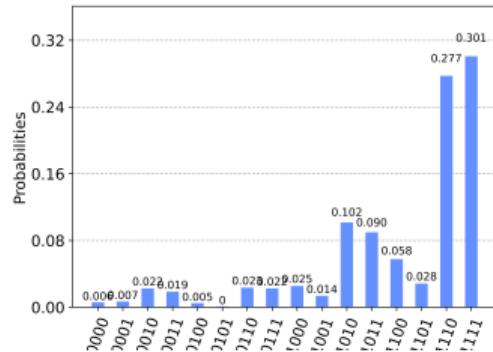
optimization\_level=3  
NoiseAdaptiveLayout



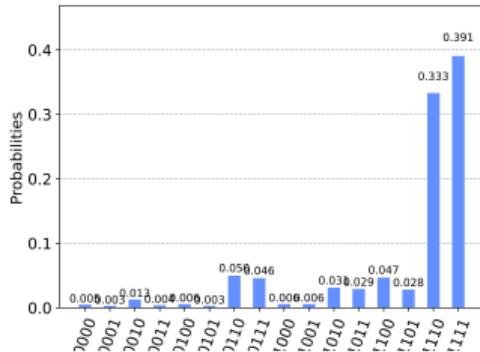
Custom Layout  
(optimization\_level=1)



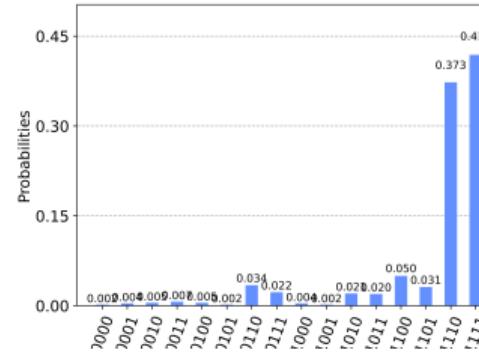
**optimization\_level=1**  
**TrivialLayout**



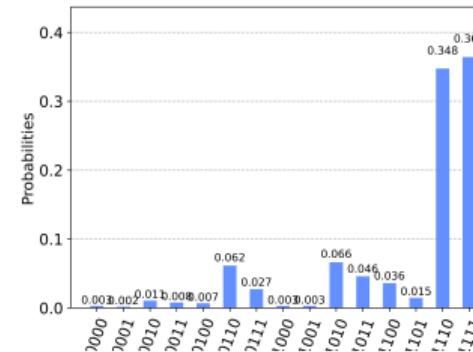
**optimization\_level=2**  
**DenseLayout**



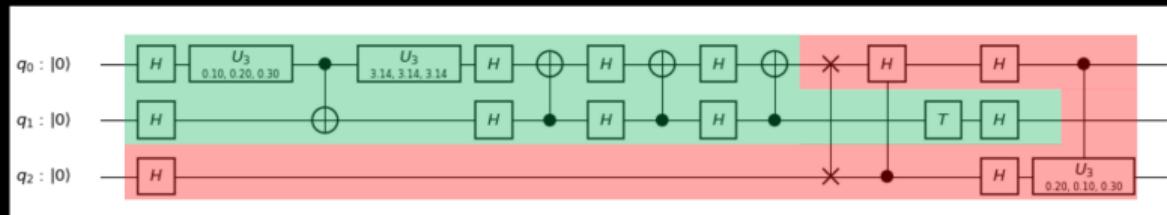
**optimization\_level=3**  
**NoiseAdaptiveLayout**



**Custom Layout  
(optimization\_level=1)**

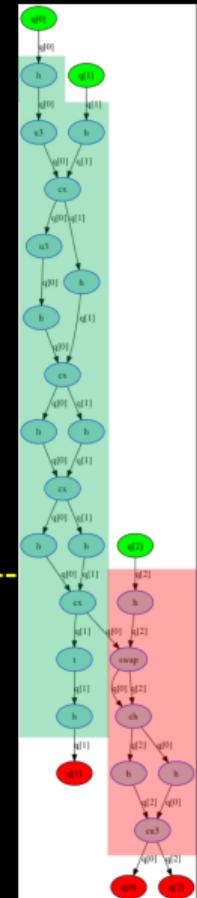


# 2-qubit Block Collection



Efficient graph traversal on the DAG.

For each CNOT, collect ancestors and predecessors until a branch.



# Optimize 1Q Operations



# Quantum Volume<sup>1</sup>

---

<sup>1</sup><https://arxiv.org/abs/1811.12926>

# Conclusions

- ▶ similar but different

## Where to get more information

- ▶ These Slides: <https://github.com/mtreinish/quantum-compilers>
- ▶ Qiskit: <https://qiskit.org/>
- ▶ Qiskit Terra on Github: <https://github.com/Qiskit/qiskit-terra>
- ▶ IBM Q Experience: <https://quantum-computing.ibm.com>
- ▶ Tutorials on Quantum Computing and Qiskit:  
<https://github.com/Qiskit/qiskit-tutorials>

## BACKUP SLIDES