

Building a Compiler for Quantum Computers

Matthew Treinish
Software Engineer - IBM Research

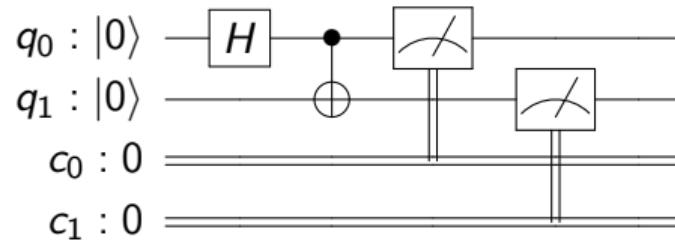
mtreinish@kortar.org

`mtreinish` on Freenode

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

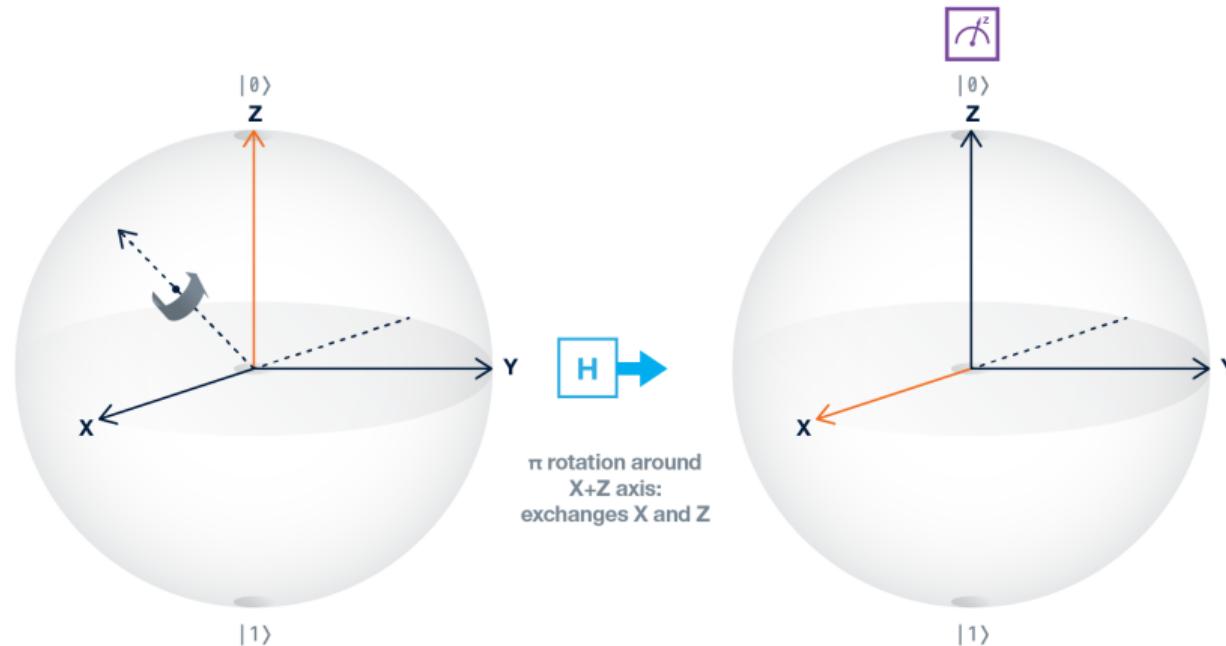
January 17, 2020

Quantum Circuits



Quantum Gates

- ▶ Quantum gates
- ▶ Gates are reversible
- ▶ Each gate can be represented as a unitary matrix



OpenQASM¹²

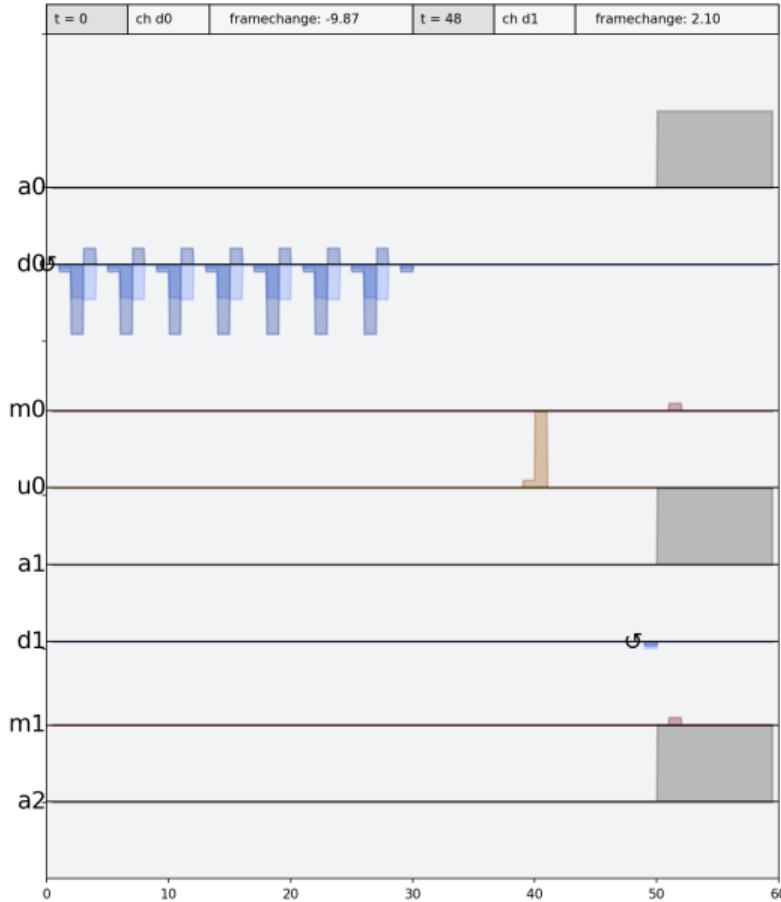
```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg c[2];
h q[0];
cx q[0],q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];
```

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

¹<https://arxiv.org/abs/1707.03429>

²<https://github.com/Qiskit/openqasm>

Pulse Level Programming

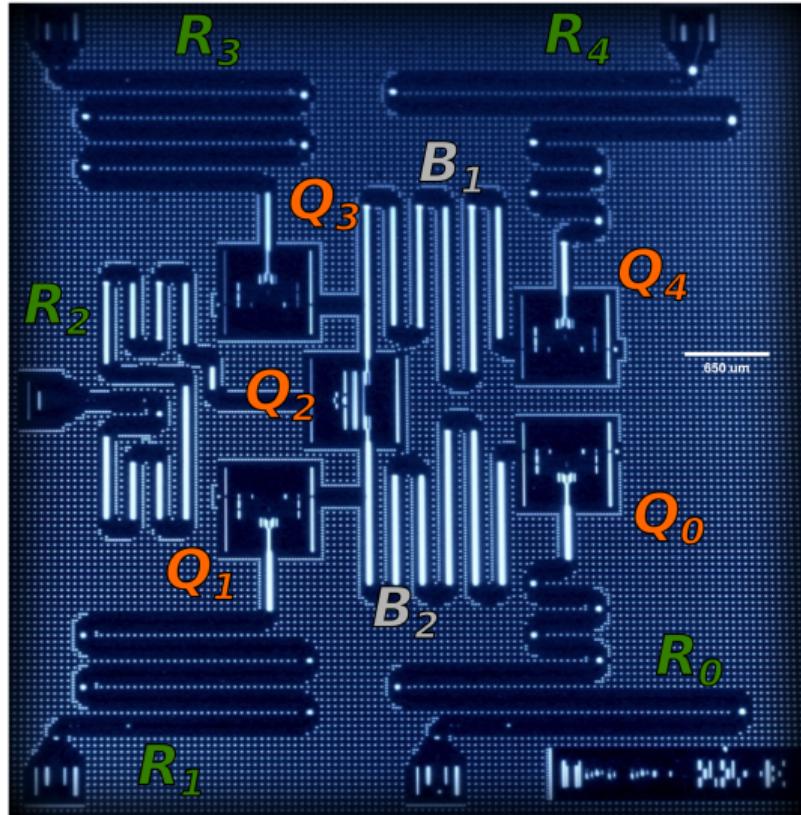


- ▶ A layer below the circuit model is to use pulses
- ▶ Each quantum gate is defined as a pulse that gets applied to the qubits
- ▶ Enables you to tweak the definition of gates
- ▶ Mostly used for physics research or hardware characterization

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

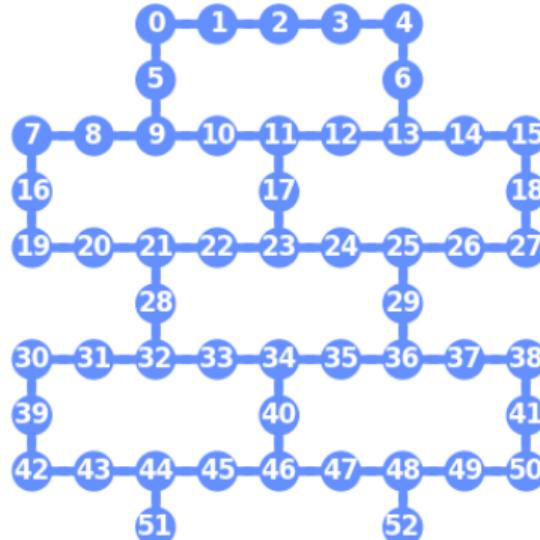
IBM Q 5 Yorktown



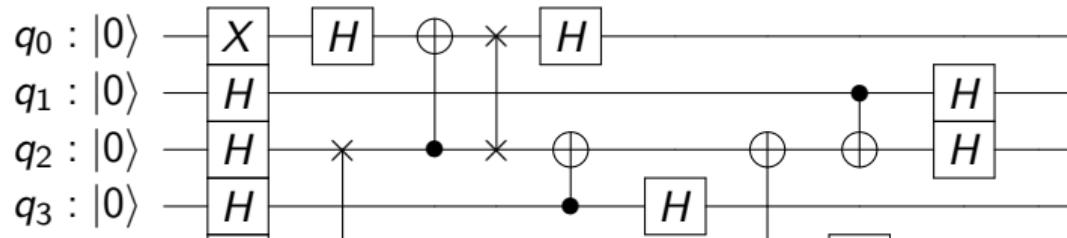
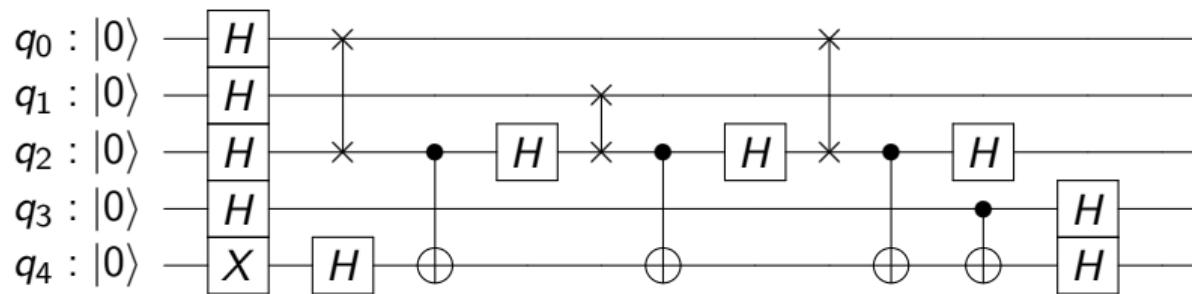
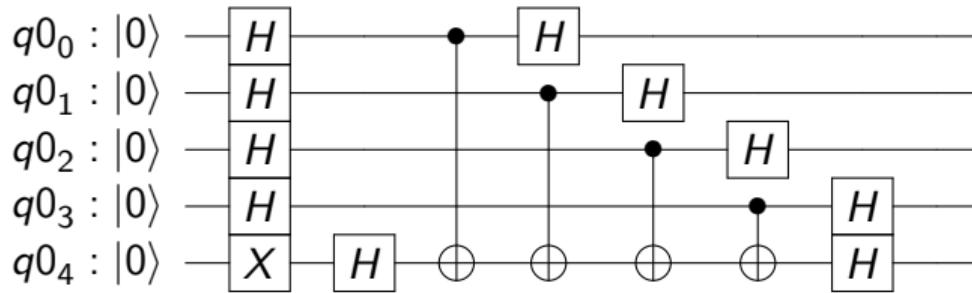
Qubit Connectivity

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 Terra¹

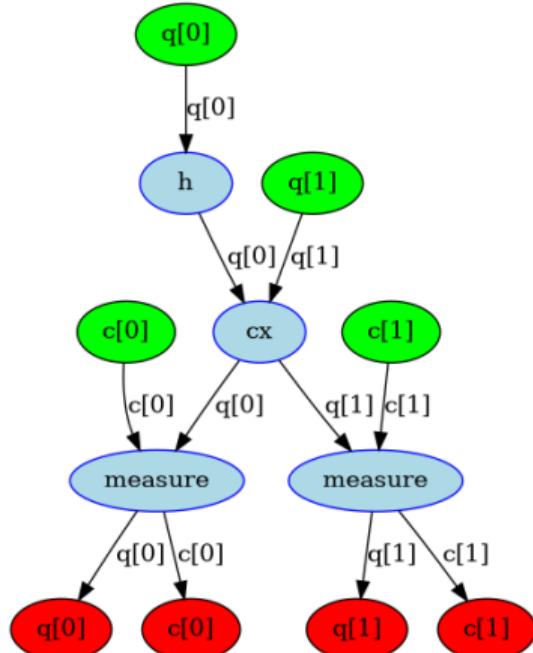
- ▶ Is the base layer for working with quantum computers provides interface to hardware and simulators
- ▶ Provides an SDK for working with quantum circuits
- ▶ Compiles circuits to run on different backends
- ▶ Designed to be backend agnostic and work with any quantum hardware or simulator
- ▶ Written in Python
- ▶ Apache 2.0 Licensed

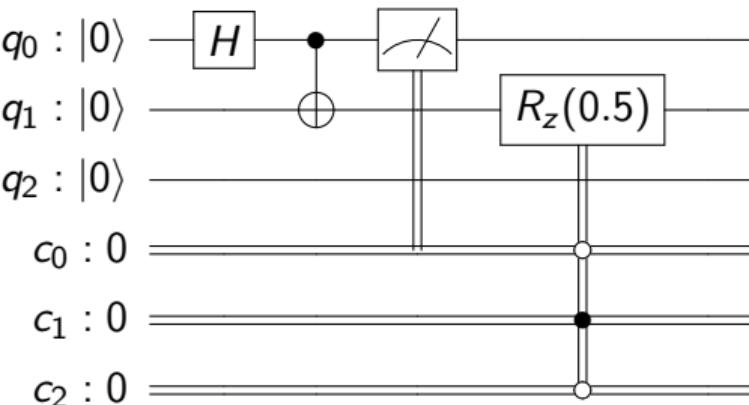


¹<https://github.com/Qiskit/qiskit-terra>

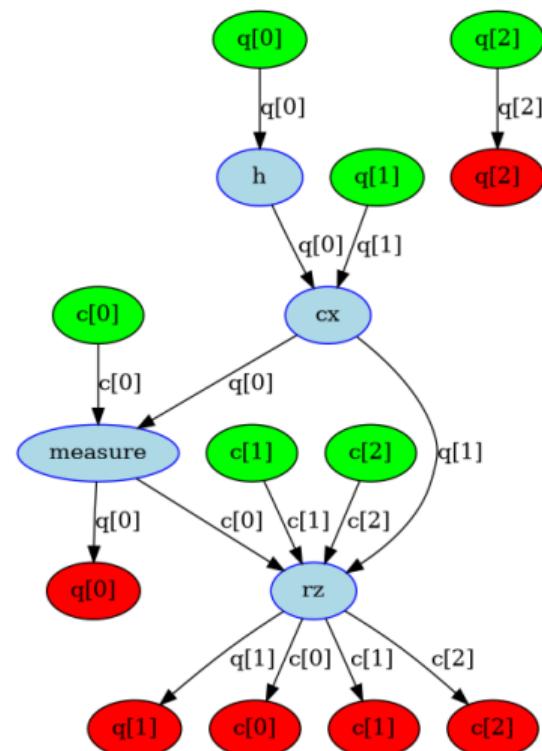
The DAG

- ▶ 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

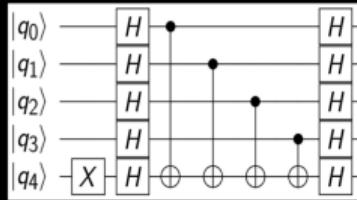




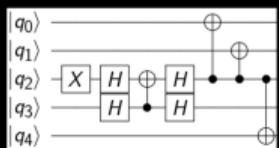
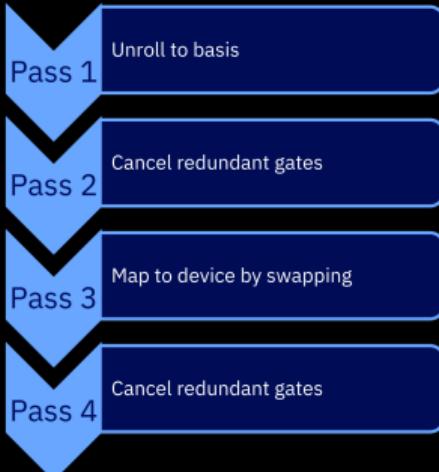
\rightarrow



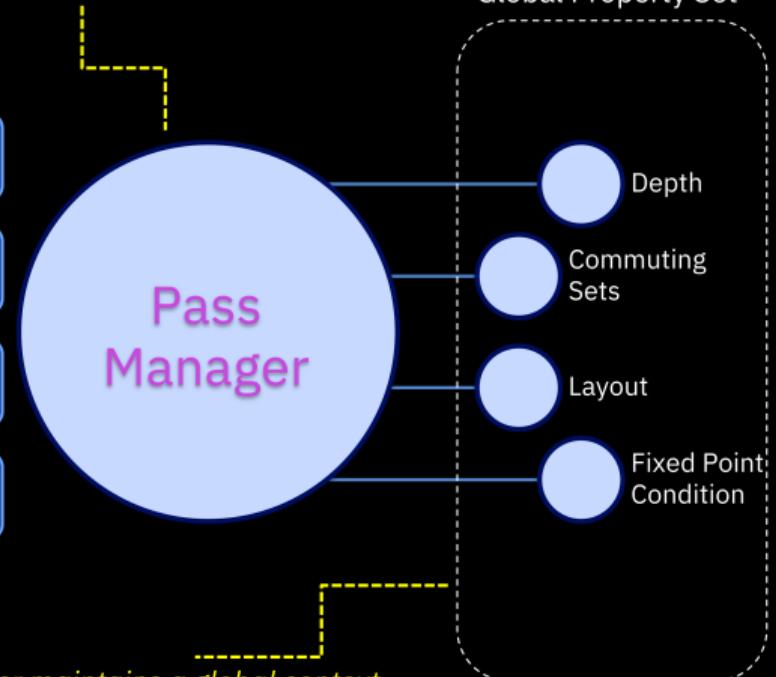
Transpiler Architecture



Each pass does one small, well-defined task.

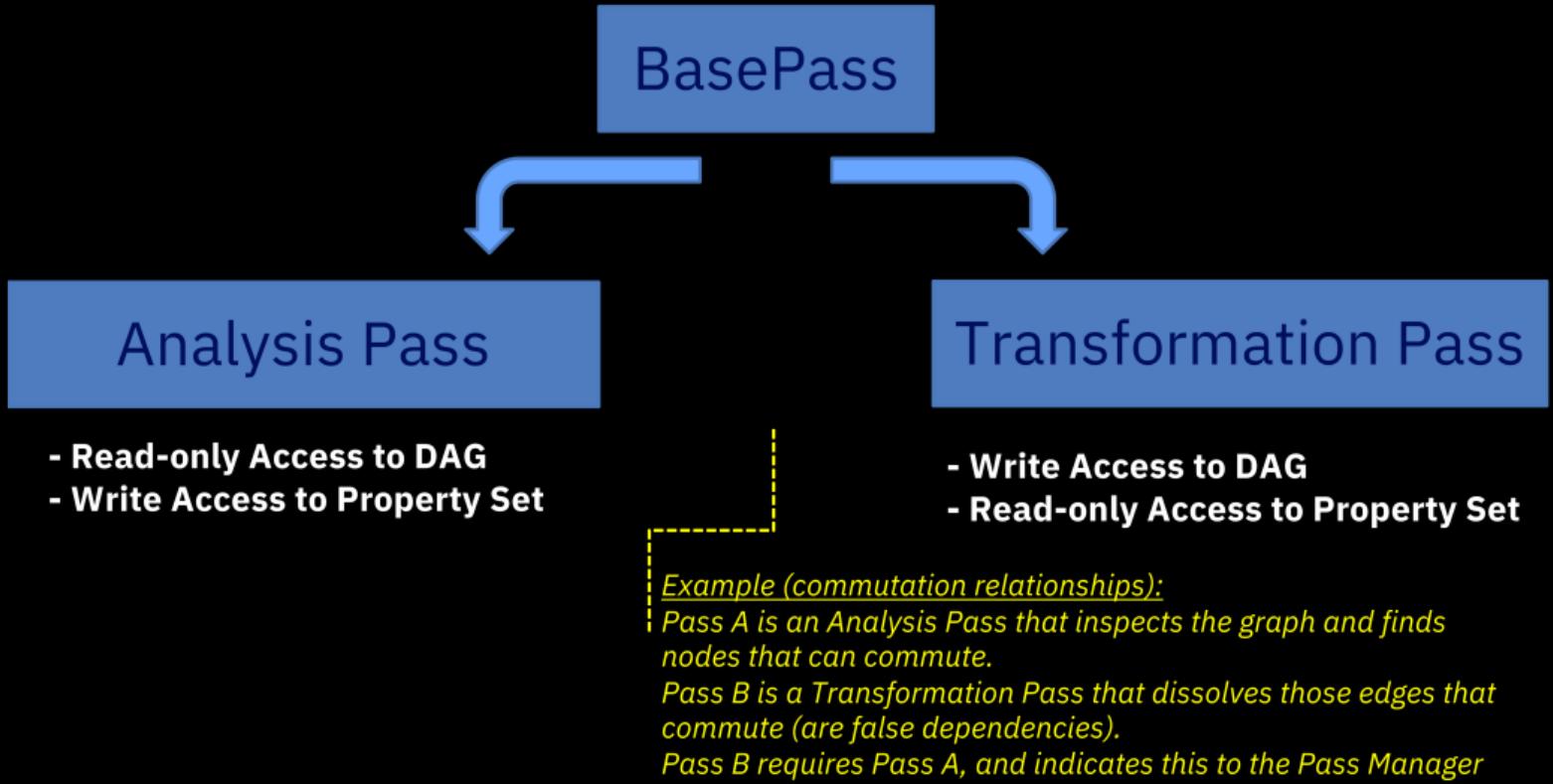


The complexity of scheduling is solely in the pass manager, to keep passes simple



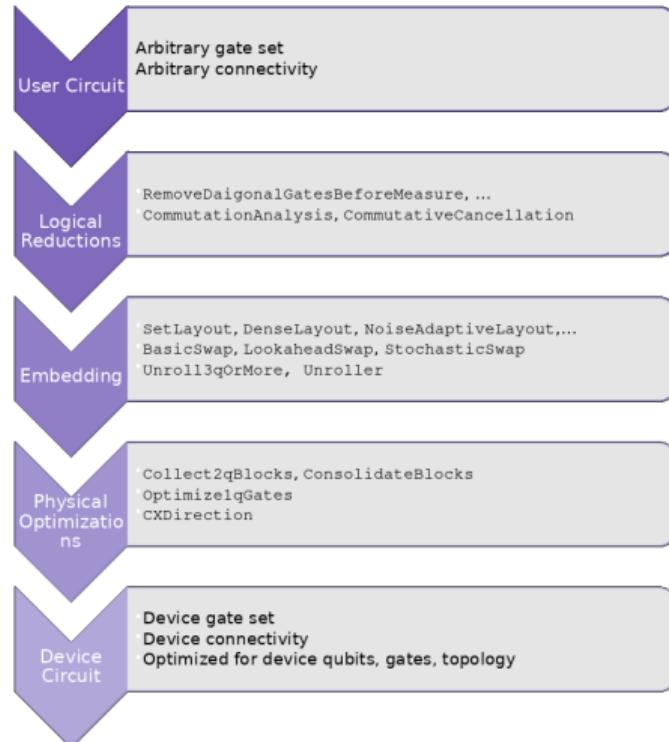
The pass manager maintains a global context about the circuit as it runs through the pipeline

Transpiler Architecture



Passmanager Stages

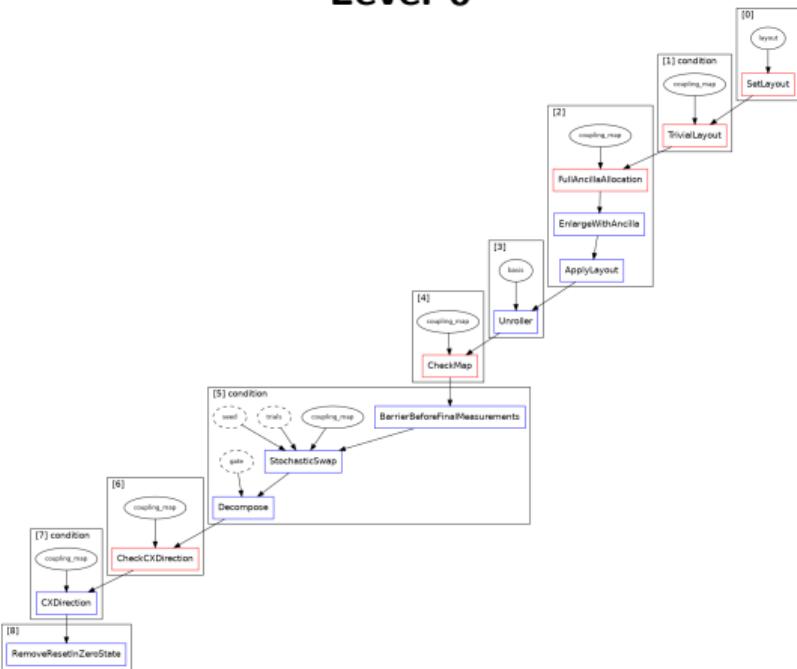
- ▶ Stages
 - ▶ Logical Reductions
 - ▶ Embedding
 - ▶ Layout
 - ▶ Mapping/Routing
 - ▶ Unrolling
 - ▶ Physical Reductions
- ▶ Pluggable and extensible



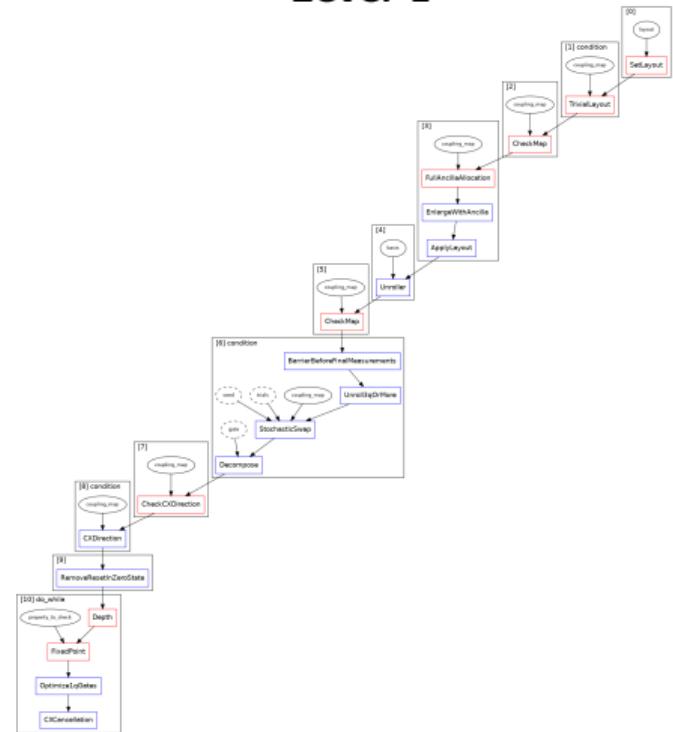
Preset Pass Managers

- ▶ Out of the box 4 optimization levels (0-3) with preset pass managers
- ▶ Increasing levels of optimization vs execute time

Level 0



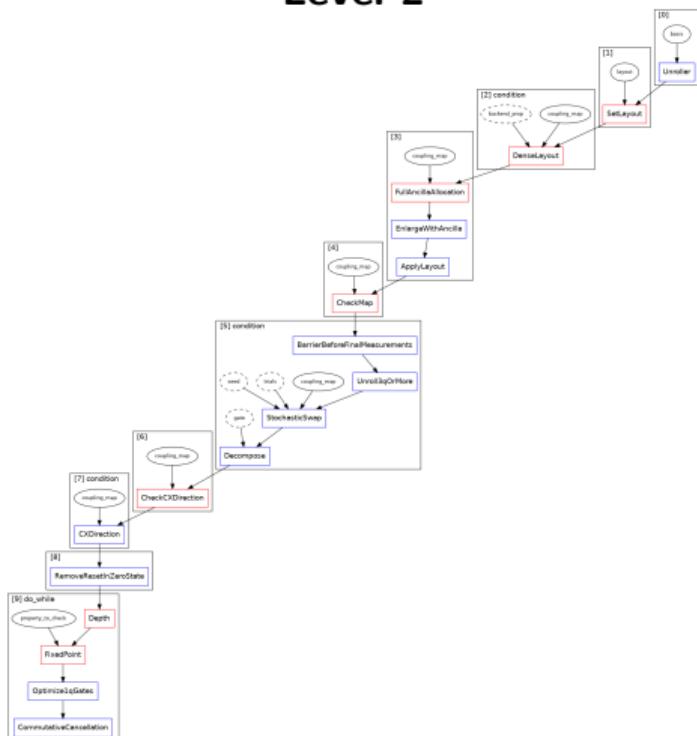
Level 1



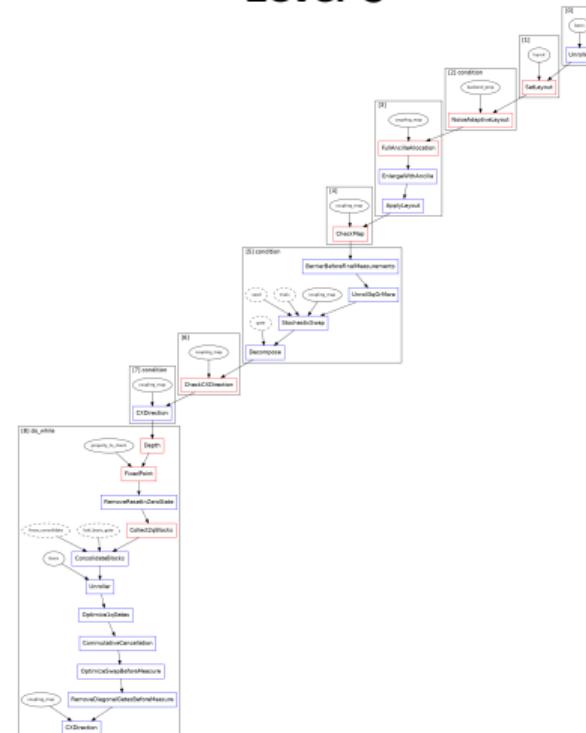
Preset Pass Managers

- ▶ Out of the box 4 optimization levels (0-3) with preset pass managers
 - ▶ Increasing levels of optimization vs execute time

Level 2



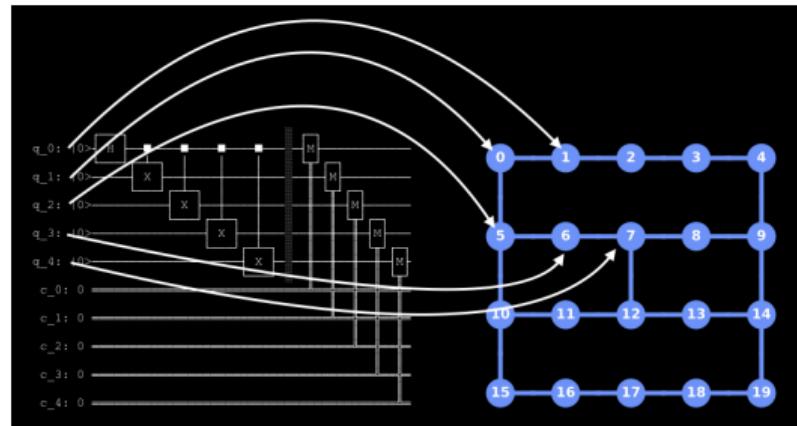
Level 3



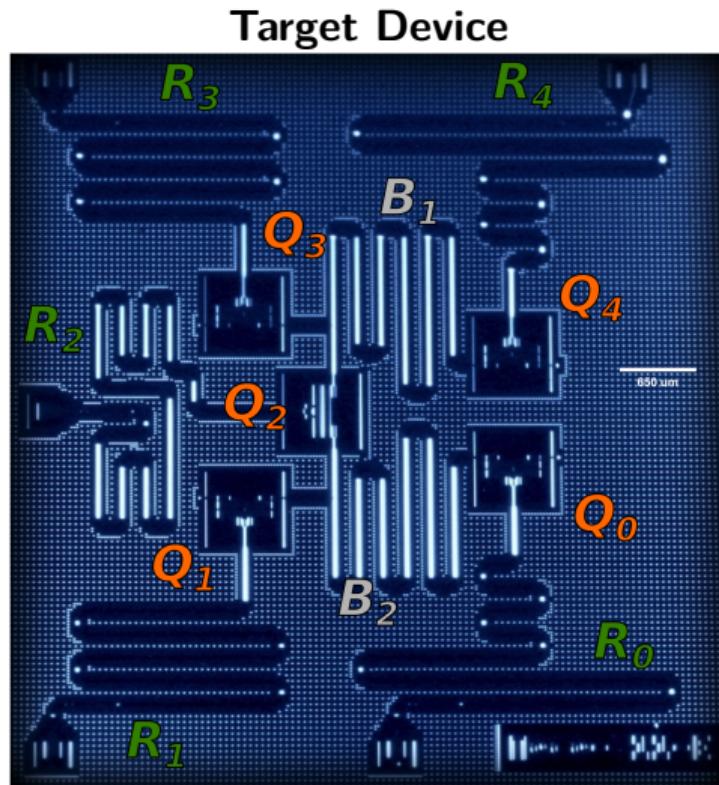
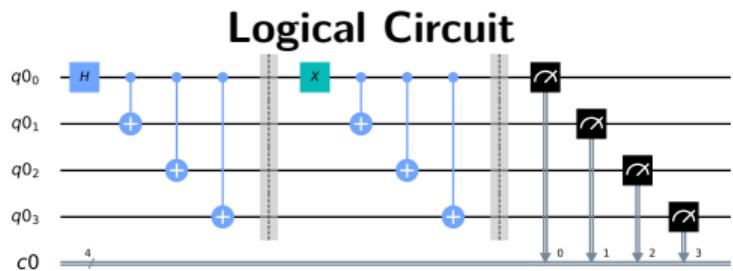
The Unroller

Layout

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

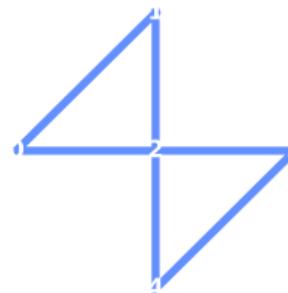
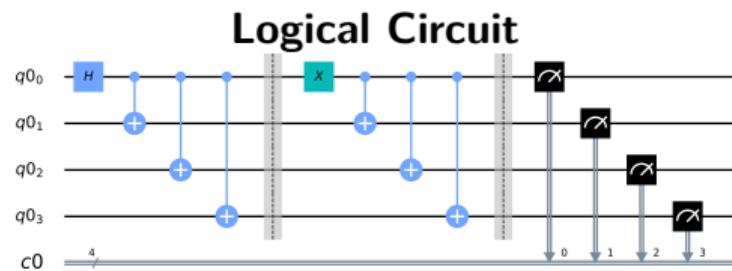


Layout Example

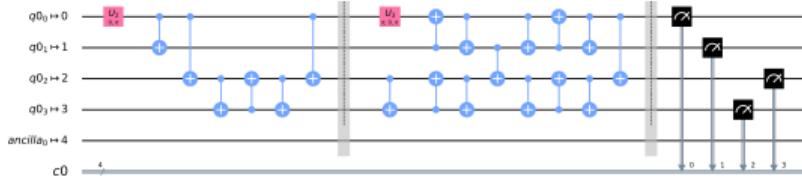


Layout Example

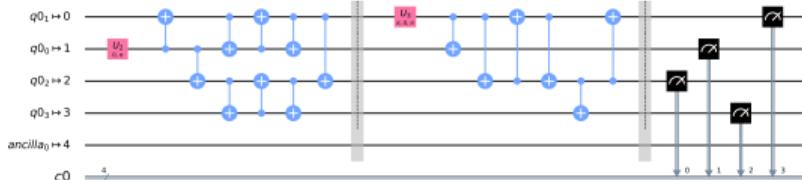
Target Device



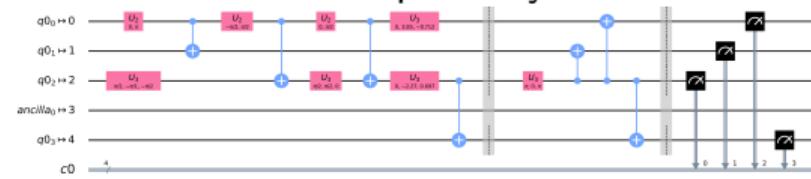
optimization_level=1
TrivialLayout



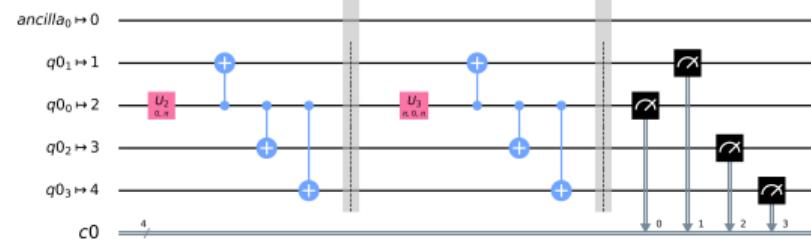
optimization_level=2
DenseLayout



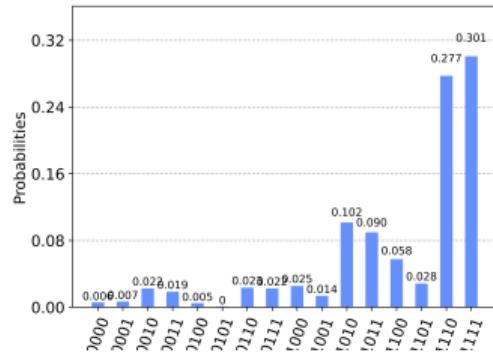
optimization_level=3
NoiseAdaptiveLayout



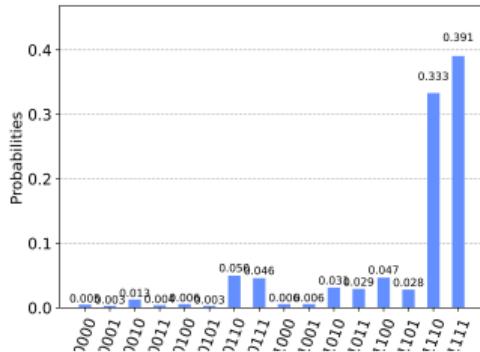
Custom Layout
(optimization_level=1)



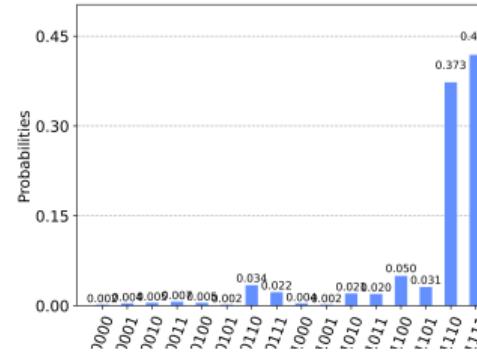
optimization_level=1
TrivialLayout



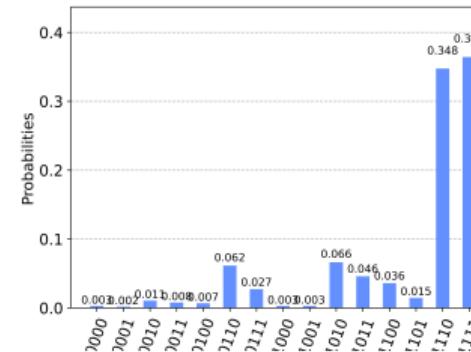
optimization_level=2
DenseLayout



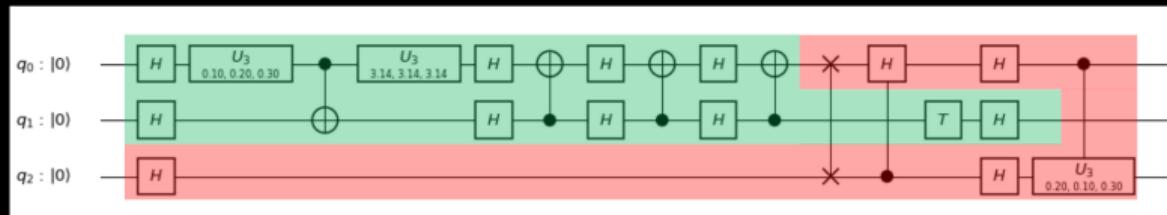
optimization_level=3
NoiseAdaptiveLayout



Custom Layout
(optimization_level=1)

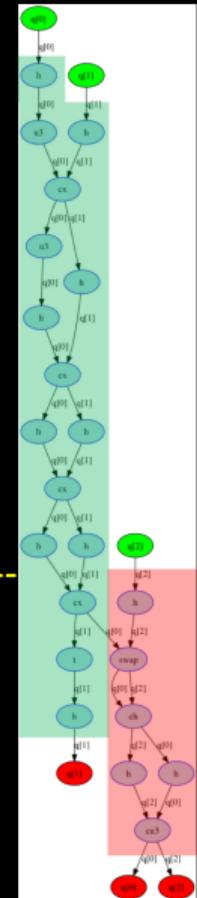


2-qubit Block Collection



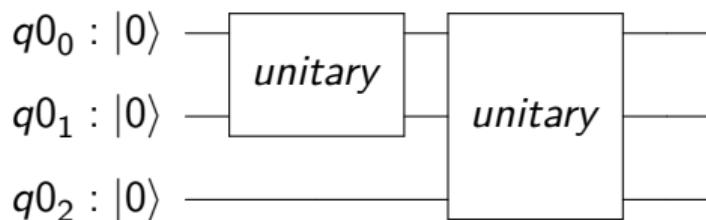
Efficient graph traversal on the DAG.

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



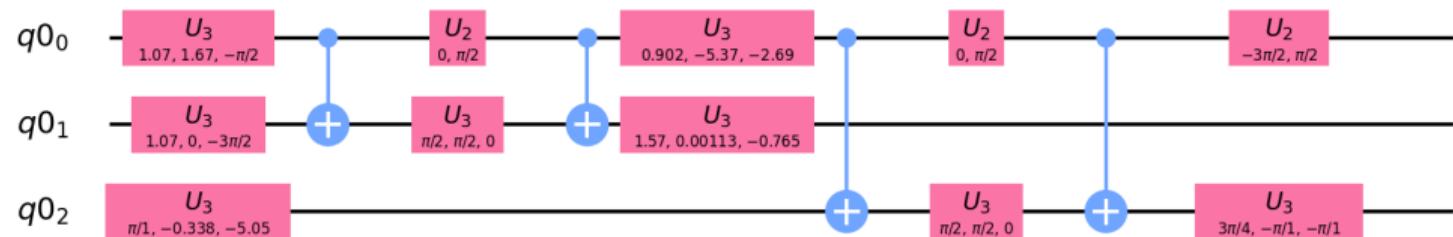
Consolidate Blocks

1. Loop over 2Q blocks collected from analysis pass
2. Run a simulation to find unitary matrix of each block
3. Replace block with unitary matrix



Consolidate Blocks

1. Loop over 2Q blocks collected from analysis pass
2. Run a simulation to find unitary matrix of each block
3. Replace block with unitary matrix



Optimize 1Q Operations

- ▶ For each run of single qubit gates find end state of rotations

Quantum Volume¹

¹<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>
- ▶ Tutorial on using Passmanager: https://github.com/Qiskit/qiskit-iqx-tutorials/blob/master/qiskit/advanced/terra/4_transpiler_passes_and_passmanager.ipynb