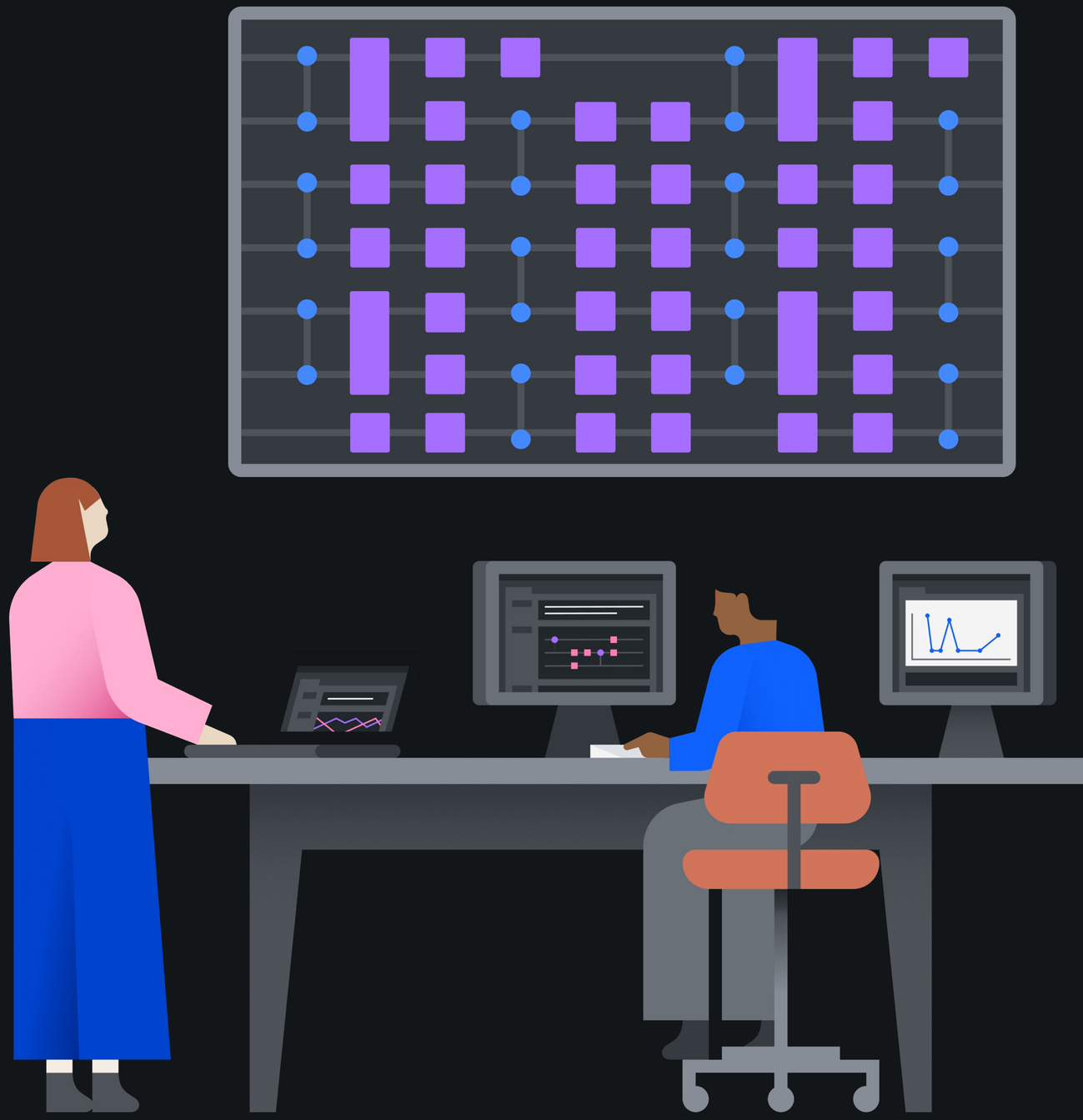
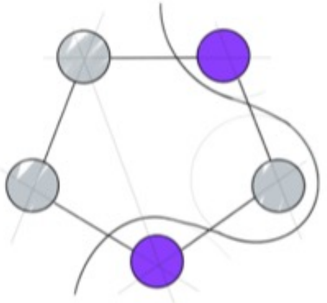
Optimizing quantum circuits with Qiskit

Luciano Bello
Qiskit Developer
IBM Quantum

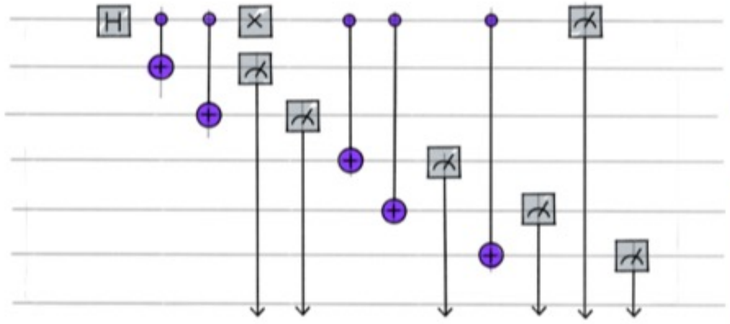
@microluciano
1ucian0
0x1ucian0



IBM

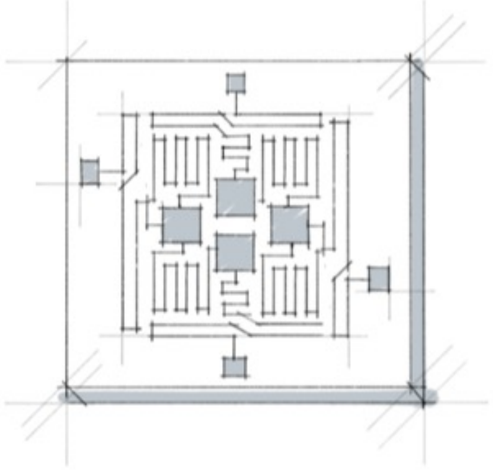


Problem



Quantum Algorithm

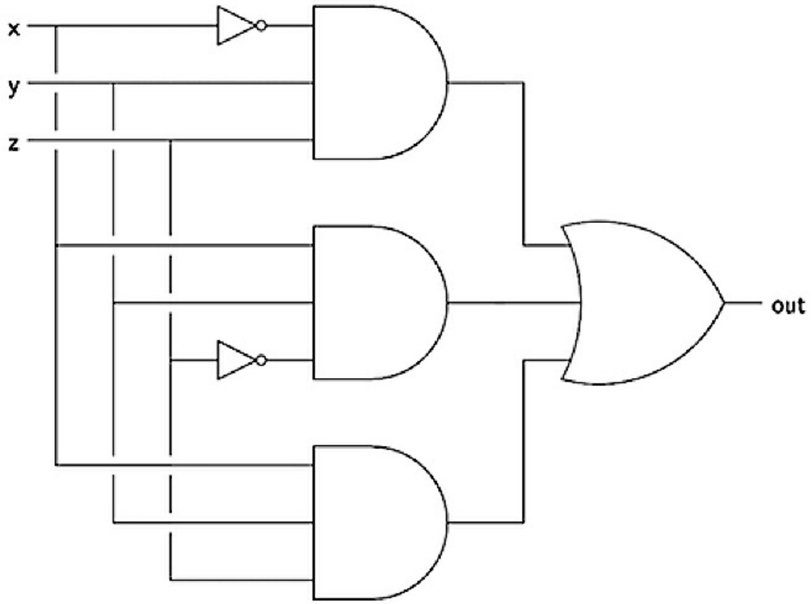
Quantum Circuits

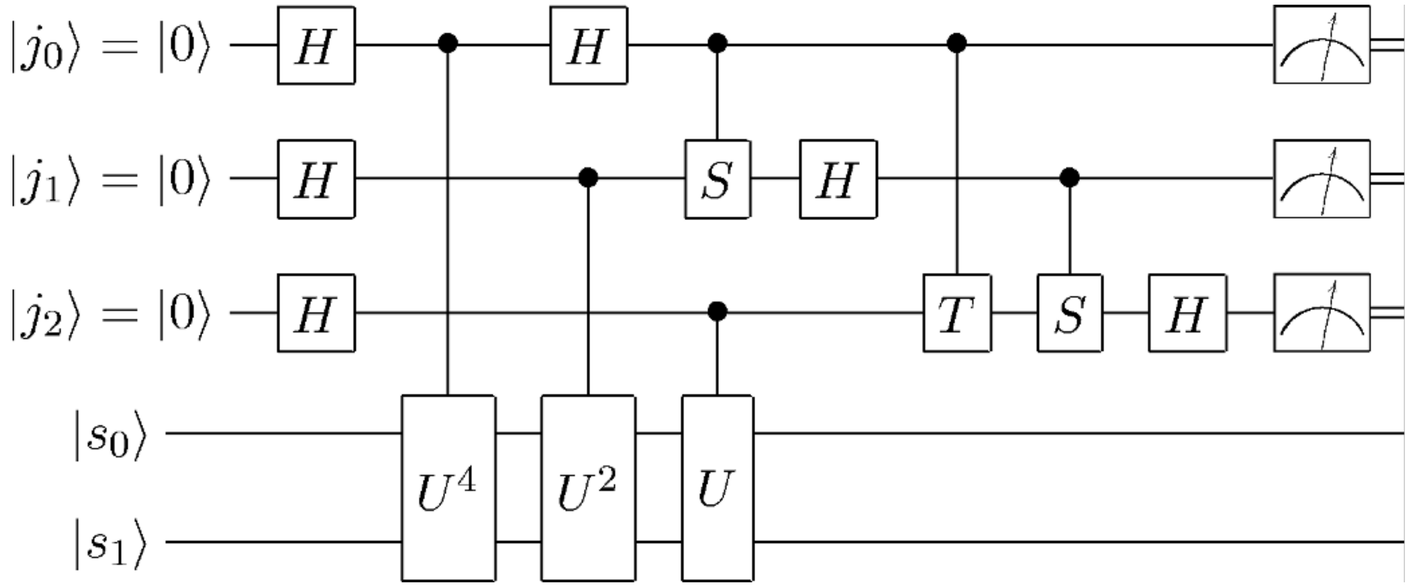


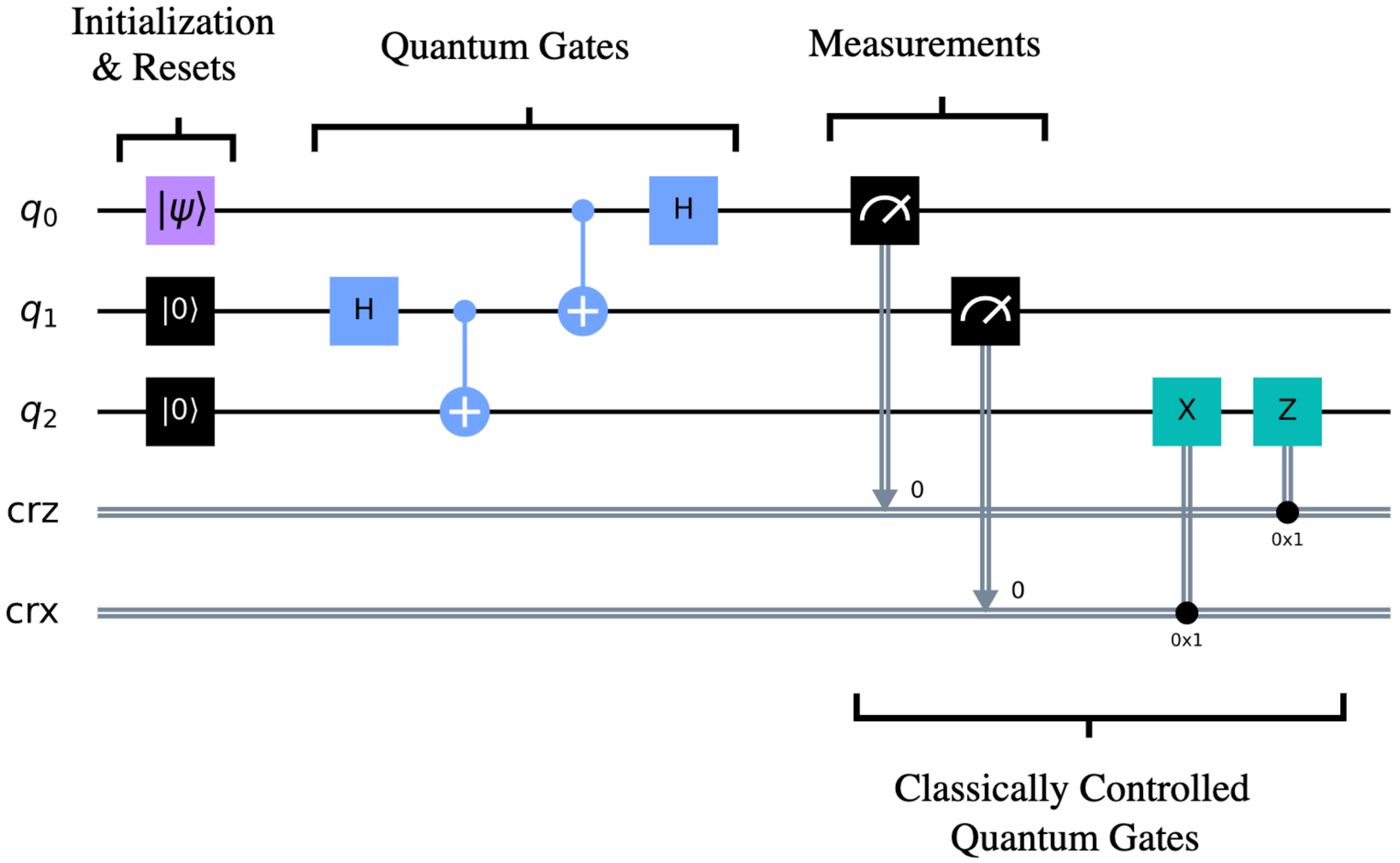
Compiled Quantum Circuit

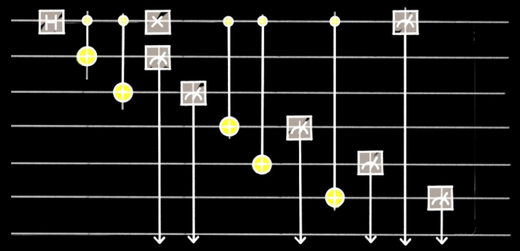
Hardware

Simulator


x
y
z
out

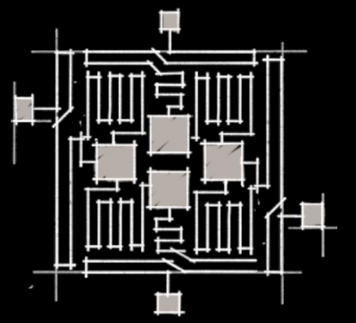

$|j_0\rangle = |0\rangle$
$|j_1\rangle = |0\rangle$
$|j_2\rangle = |0\rangle$
$|s_0\rangle$
$|s_1\rangle$
H
U^4
U^2
U
S
T


Initialization & Resets
Quantum Gates
Measurements
q_0
q_1
q_2
crz
crx
$|\psi\rangle$
|0⟩
|0⟩
H
H
X
Z
0
0
0x1
0x1
Classically Controlled Quantum Gates



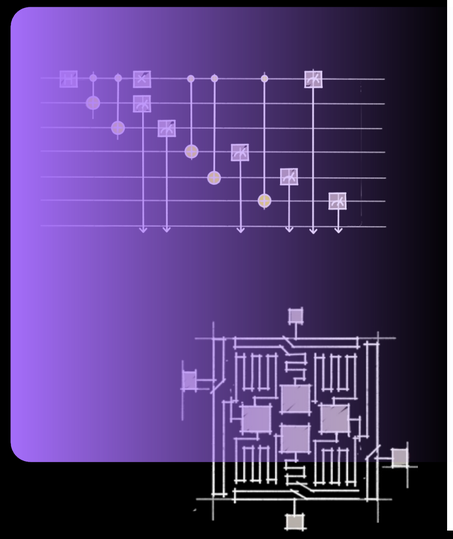
quantum circuit

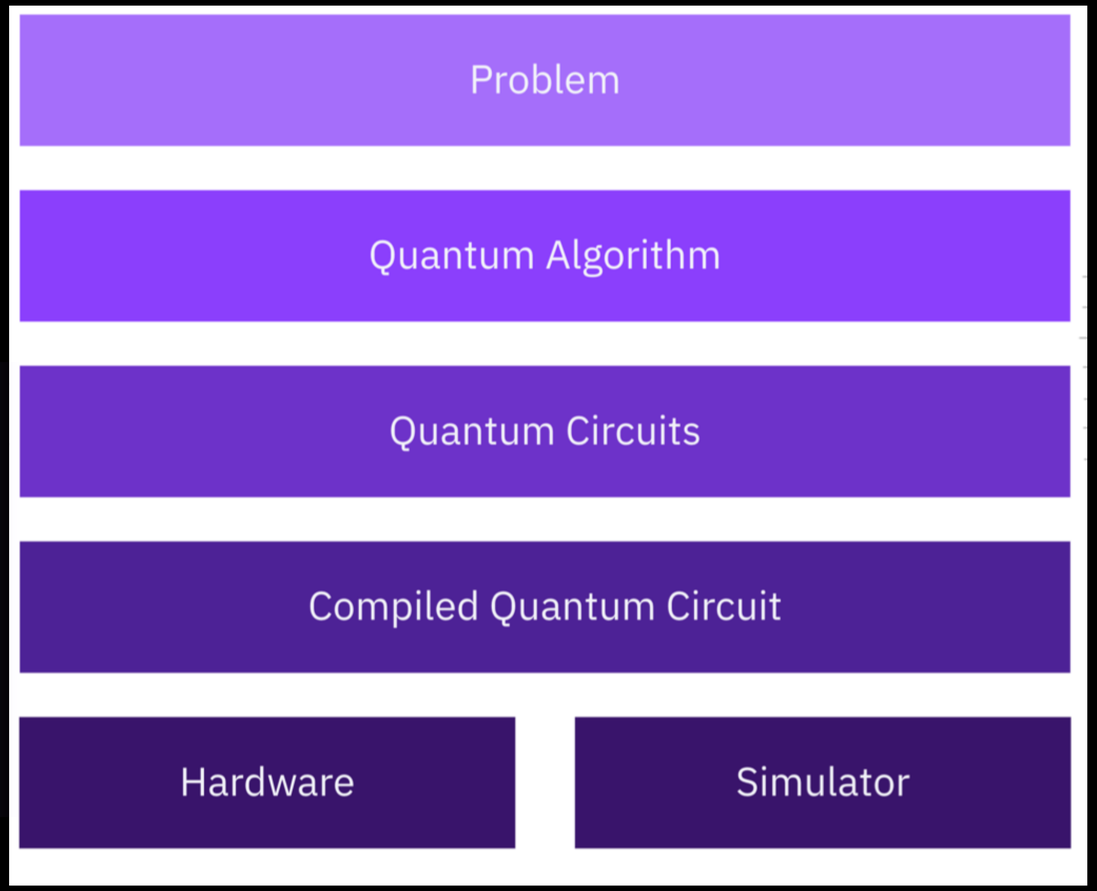
target description



compiled quantum circuit

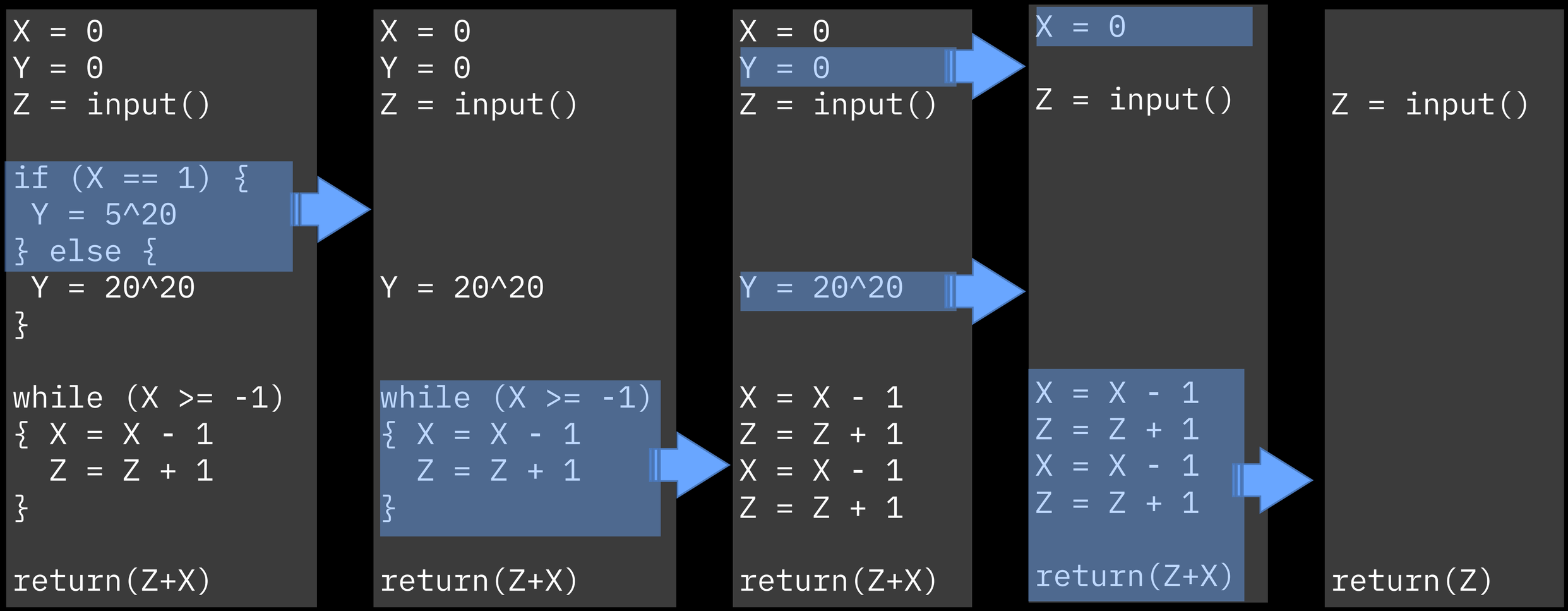
- Why compiling?
- Transpiler architecture
- Performance





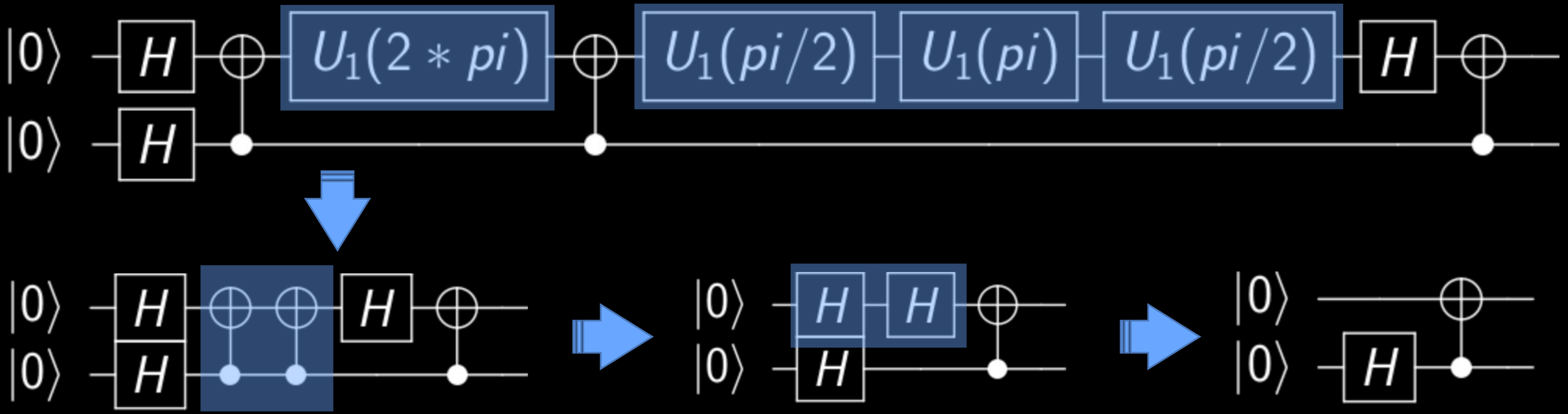
Why compiling?

Optimizations



Why compiling?

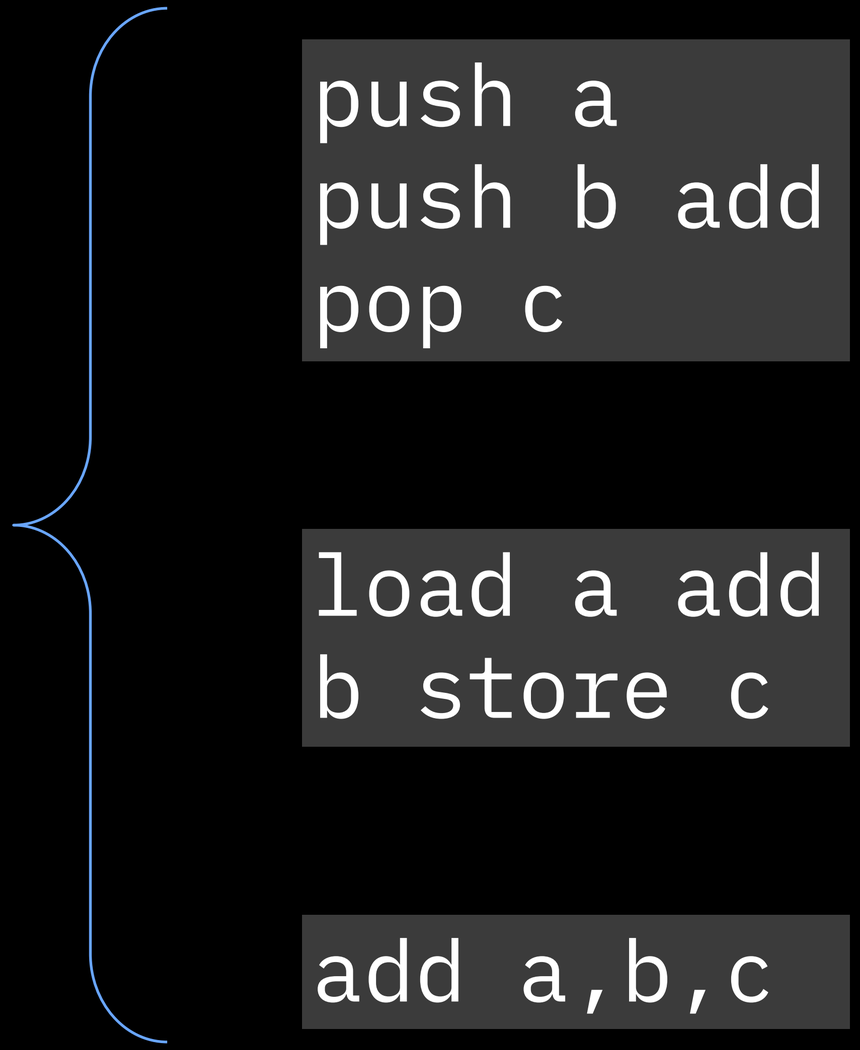
Optimizations



Why compiling?

Optimizations
Instruction set

```
c = a+b
```



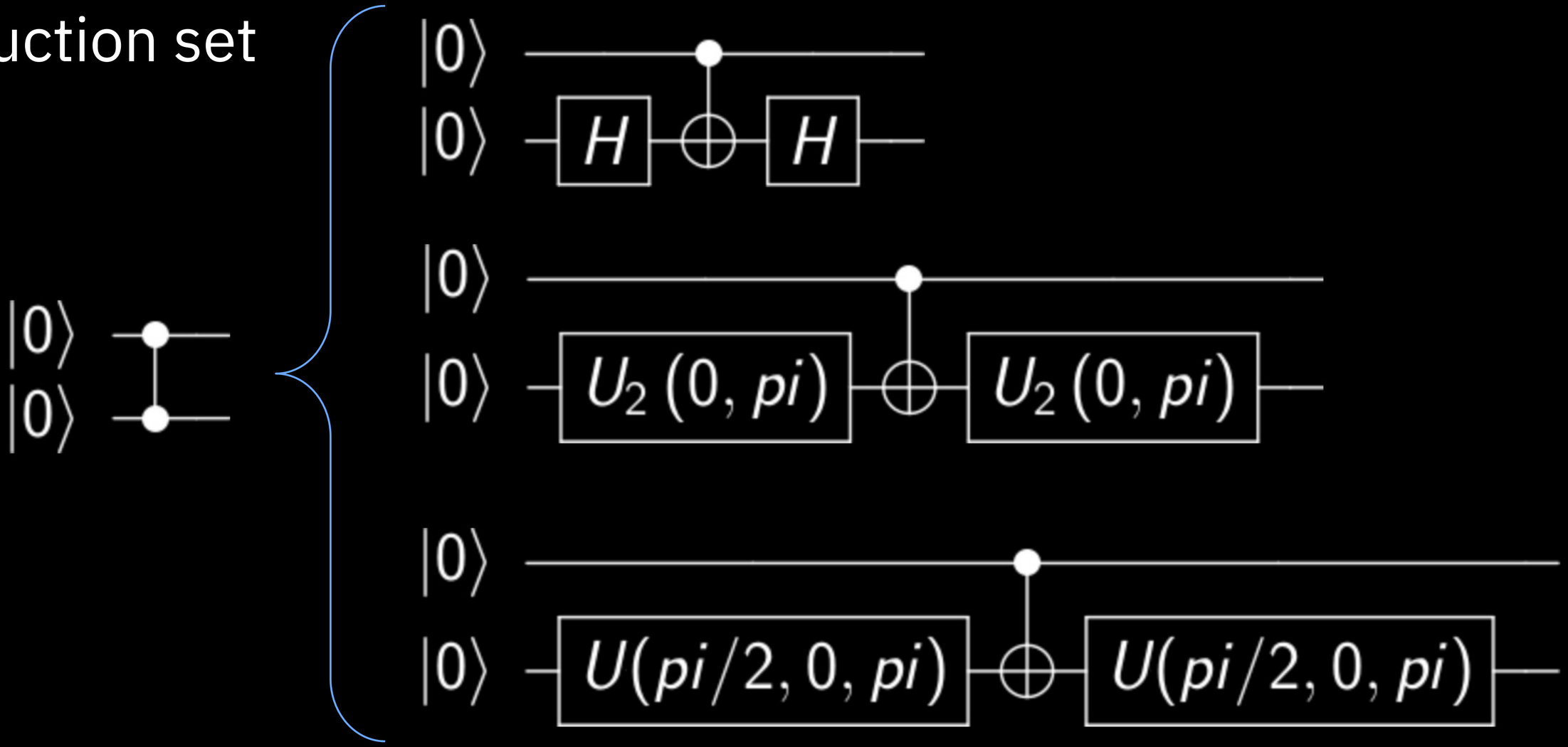
```
push a
push b add
pop c
```

```
load a add
b store c
```

```
add a,b,c
```

Why compiling?

Optimizations
Instruction set



Why compiling?

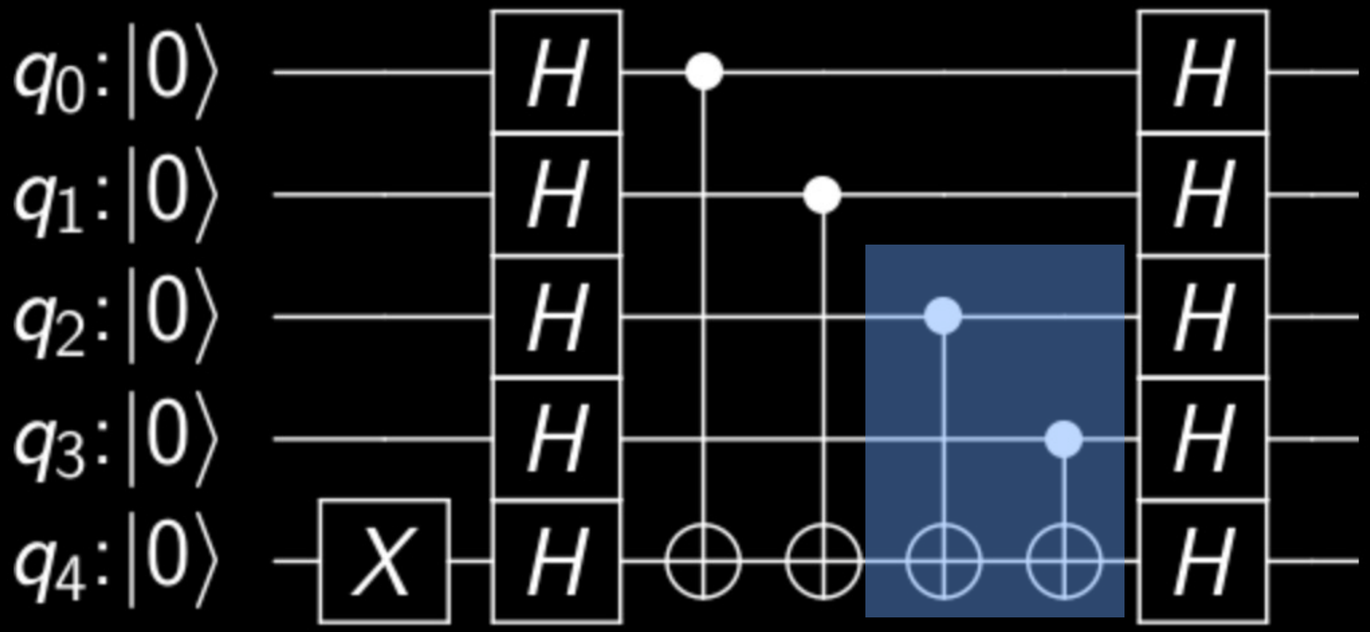
Falcon
27 Qubits
ECR, CX, Rz, √X, X

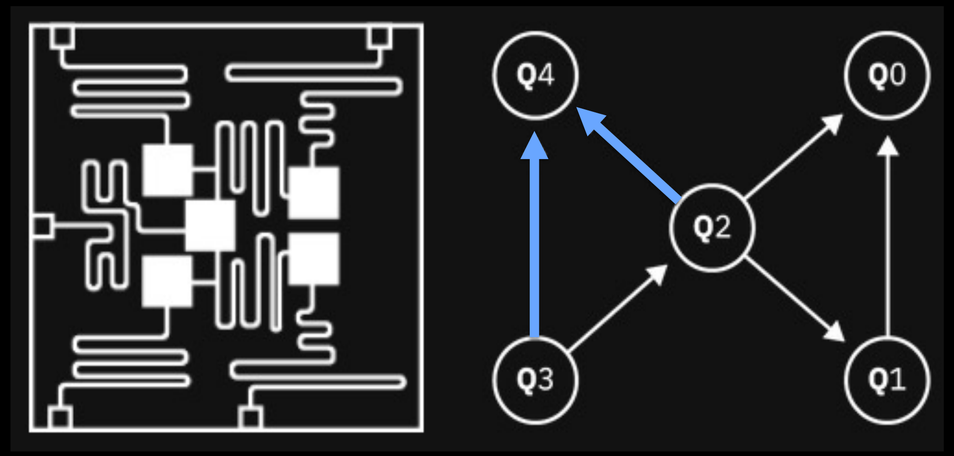
Eagle
127 Qubits
ECR, Rz, √X, X

Heron
133 Qubits
CZ, Rz, √X, X

Why compiling?

Optimizations
Instruction set
Hardware connectivity

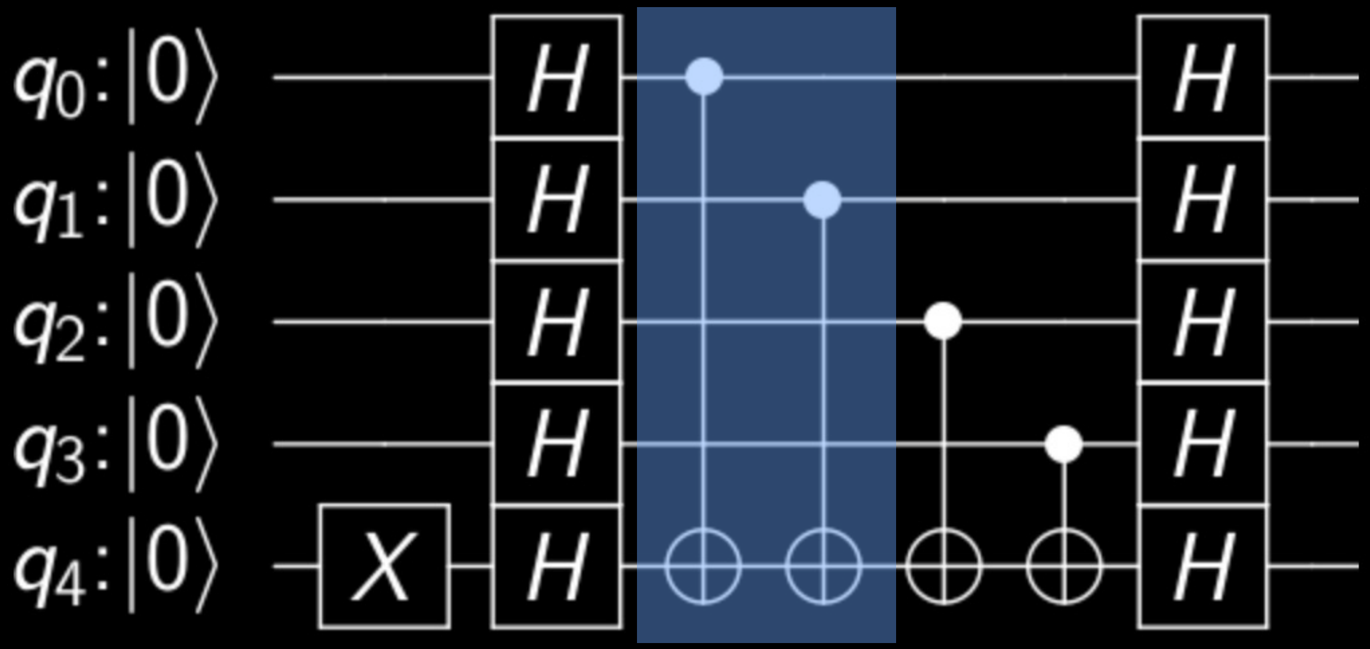


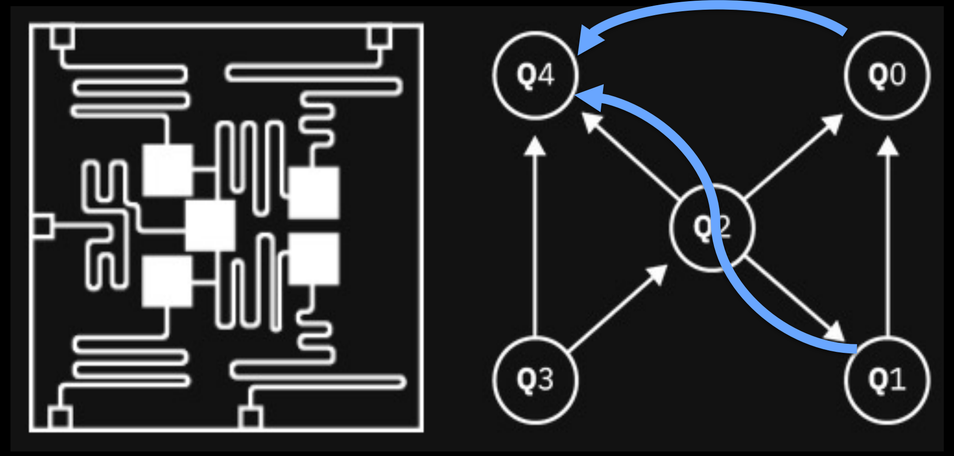


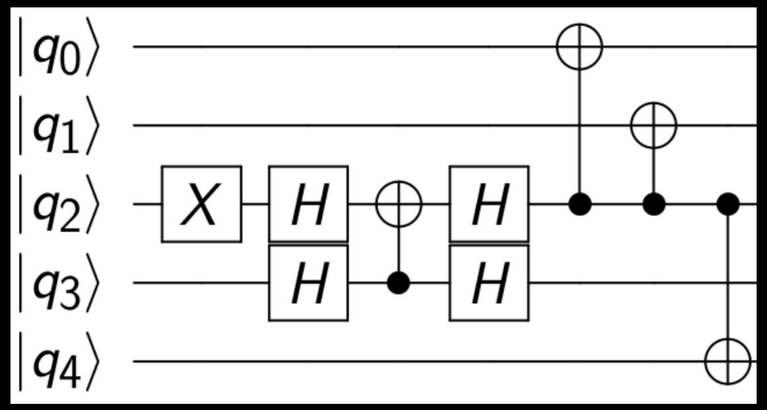
Why compiling?

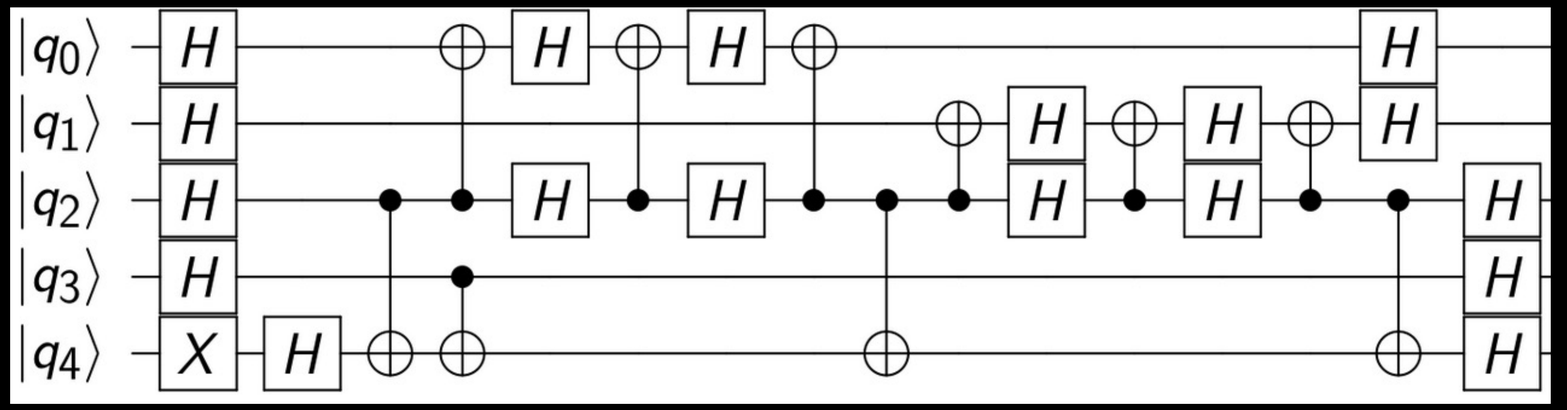
Optimizations
Instruction set
Hardware connectivity

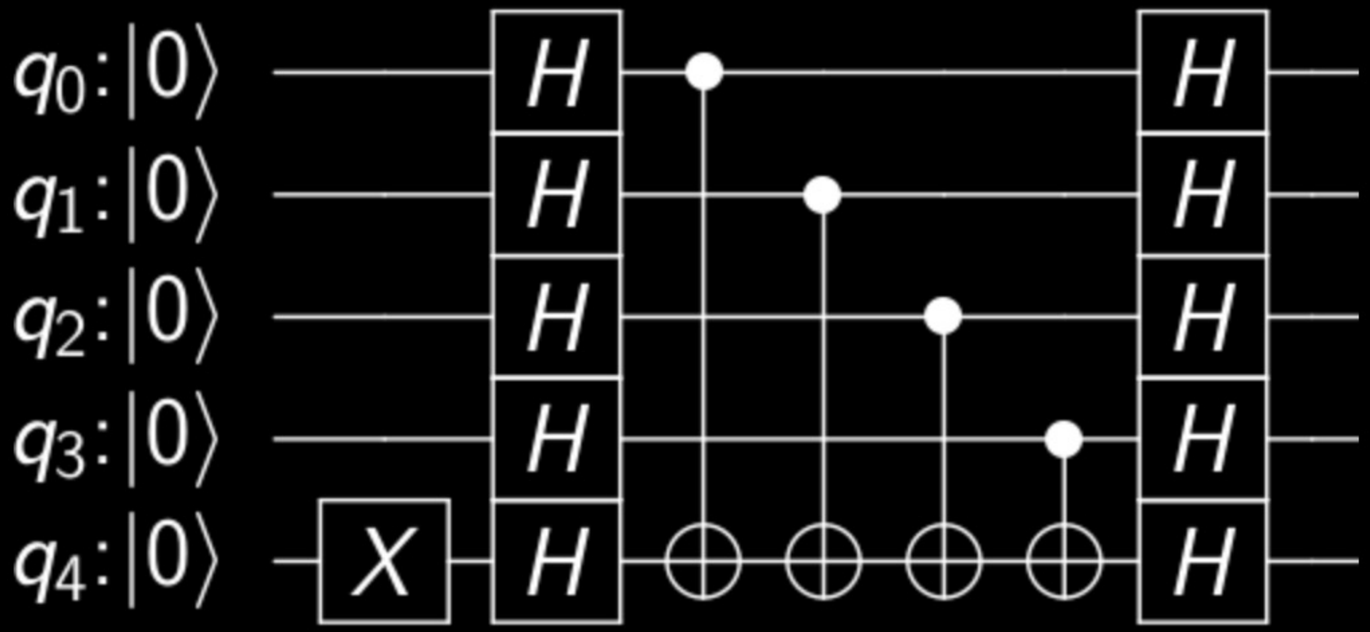
The Routing Problem

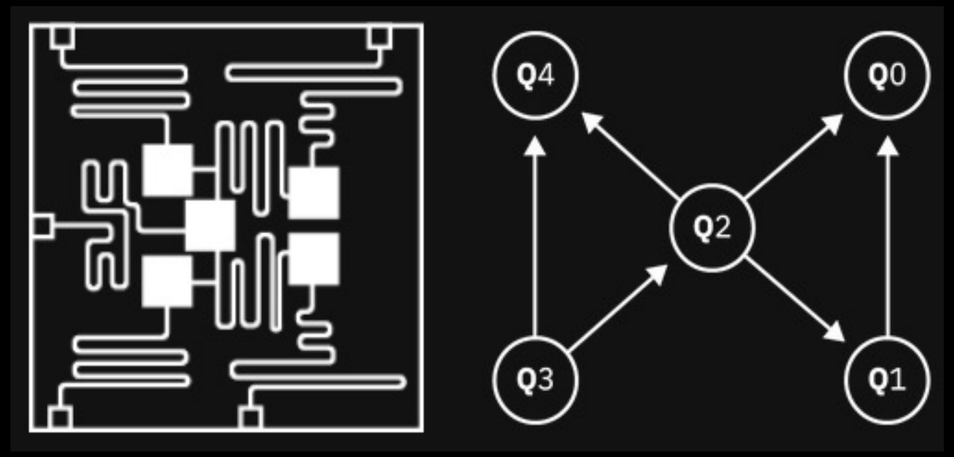


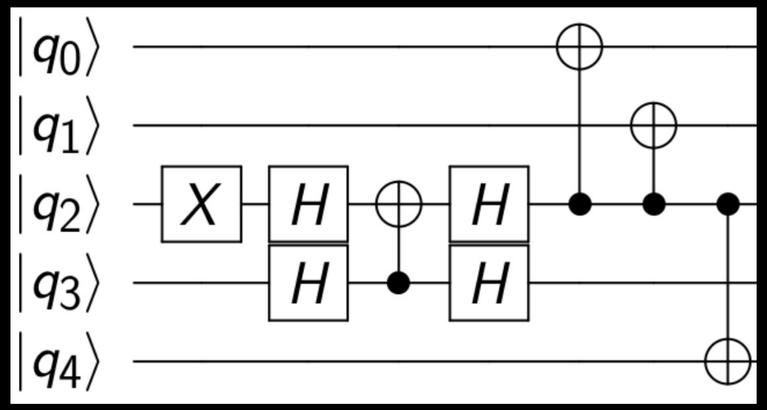


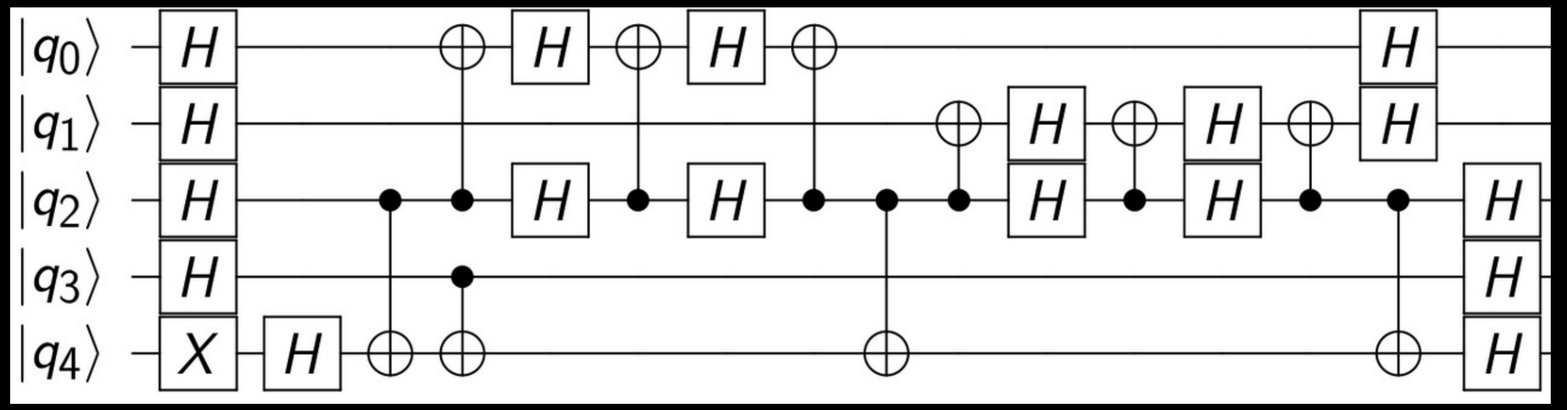

$|q_0\rangle$
$|q_1\rangle$
$|q_2\rangle$
$|q_3\rangle$
$|q_4\rangle$
X
H
H
H
H

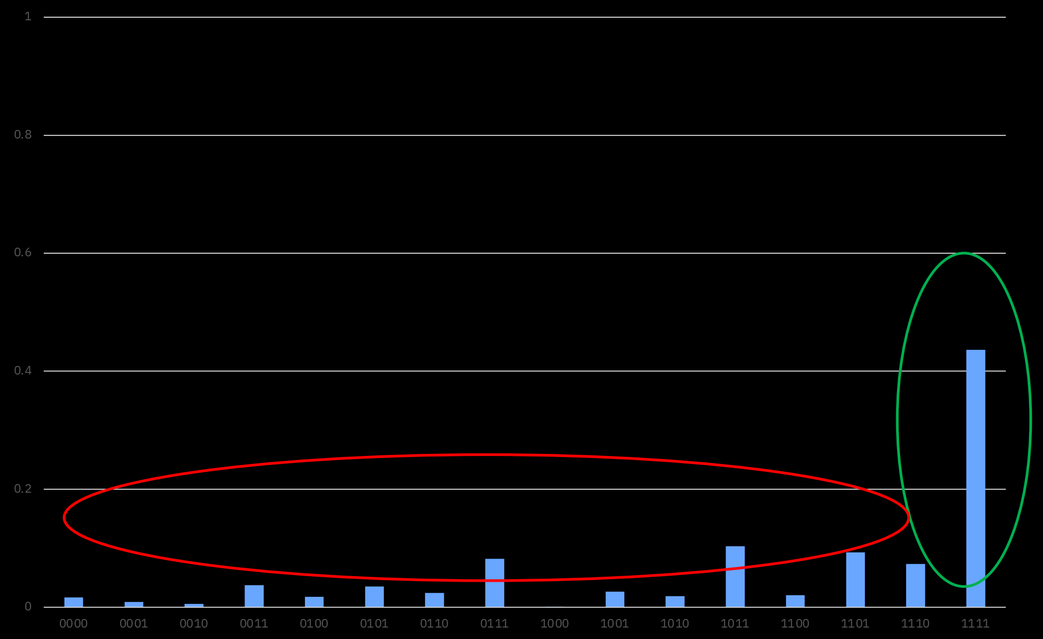

$|q_0\rangle$
$|q_1\rangle$
$|q_2\rangle$
$|q_3\rangle$
$|q_4\rangle$
H
X
H

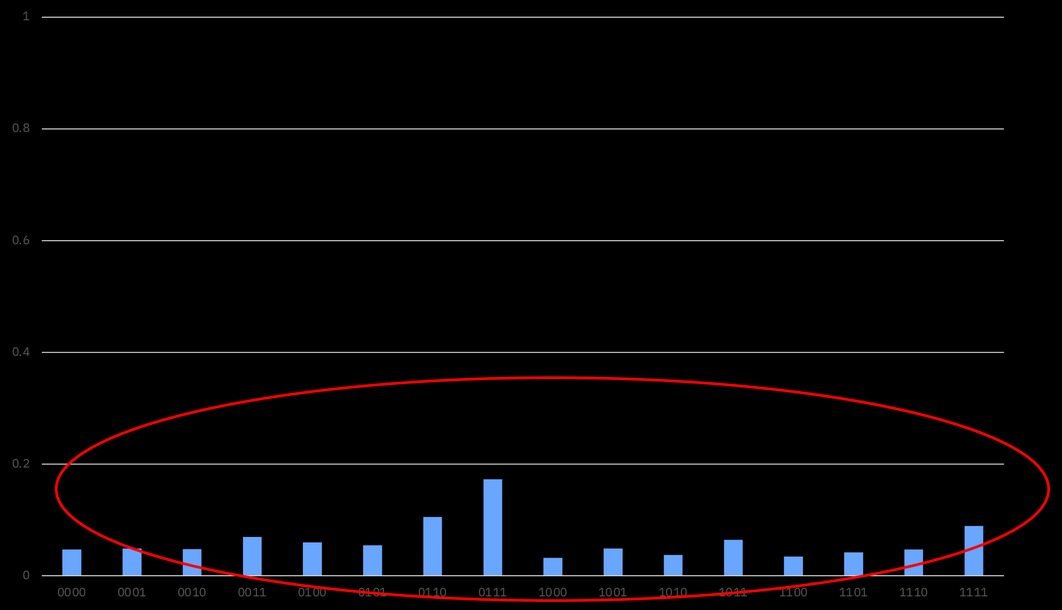

$q_0 : |0\rangle$
$q_1 : |0\rangle$
$q_2 : |0\rangle$
$q_3 : |0\rangle$
$q_4 : |0\rangle$
X
H


Q4
Q0
Q2
Q3
Q1


$|q_0\rangle$
$|q_1\rangle$
$|q_2\rangle$
$|q_3\rangle$
$|q_4\rangle$
X
H
H
H
H

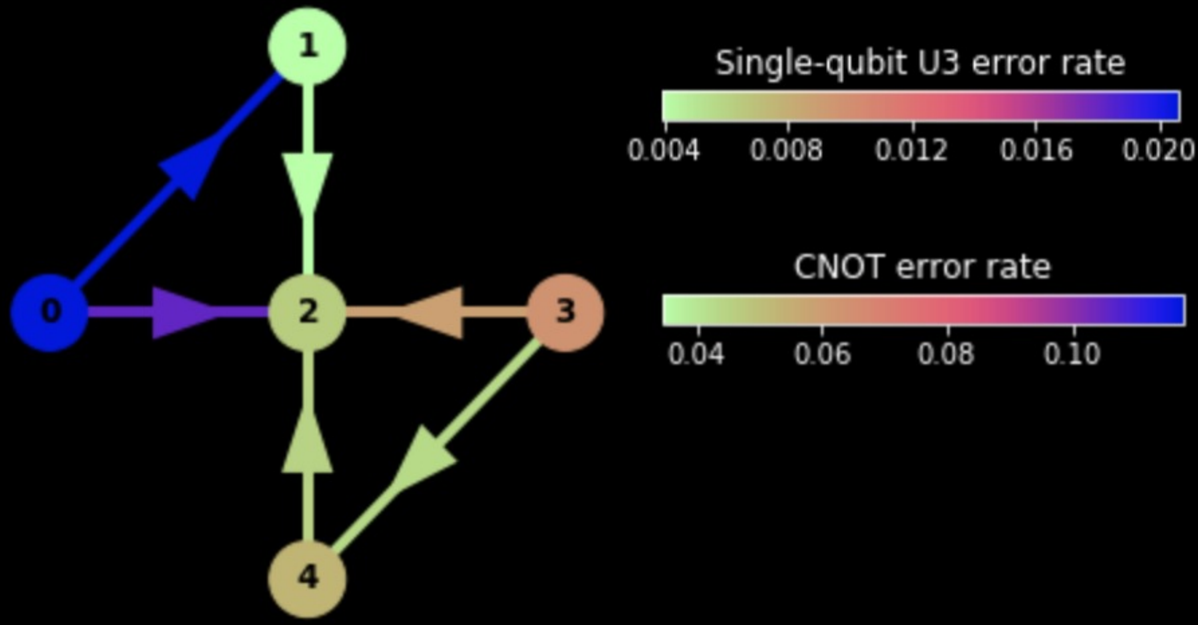

$|q_0\rangle$
$|q_1\rangle$
$|q_2\rangle$
$|q_3\rangle$
$|q_4\rangle$
H
X

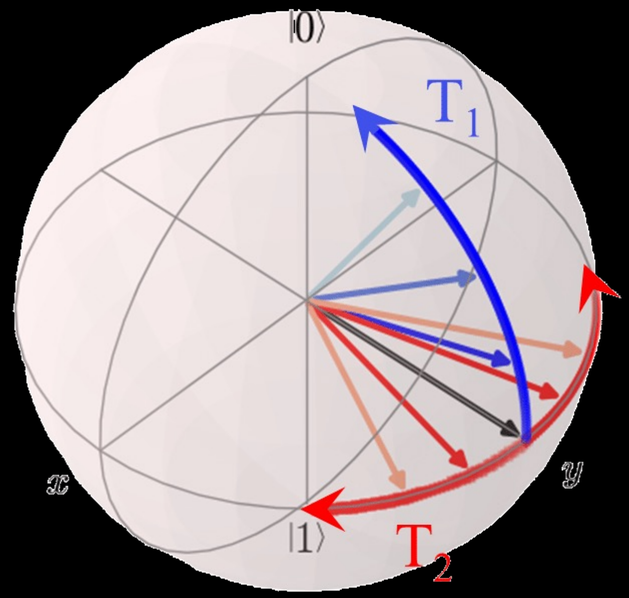

1
0.8
0.6
0.4
0.2
0
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111


1
0.8
0.6
0.4
0.2
0
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

Noise everywhere!

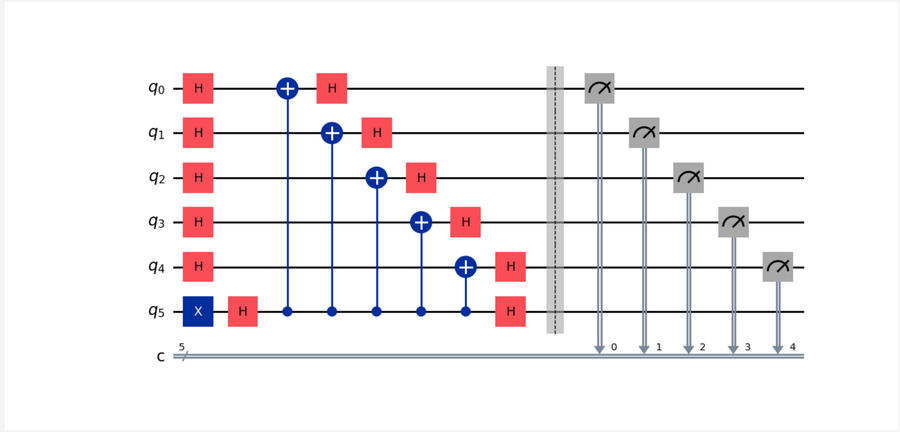
- Gate error
  - Single qubit gate
  - Multiple qubit gates
- Readout error
- Decoherence time

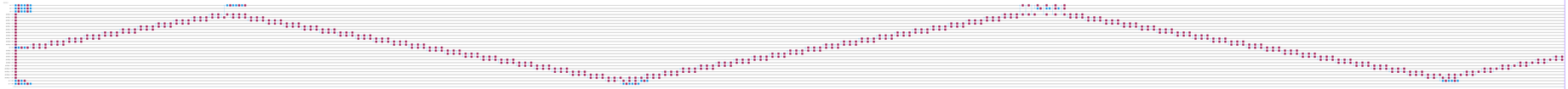


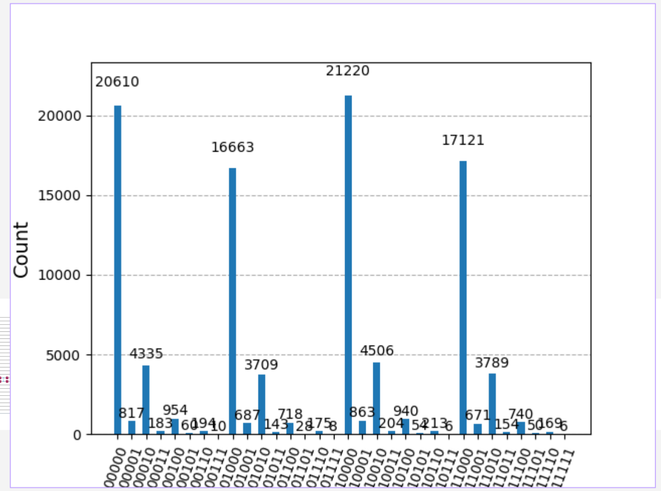


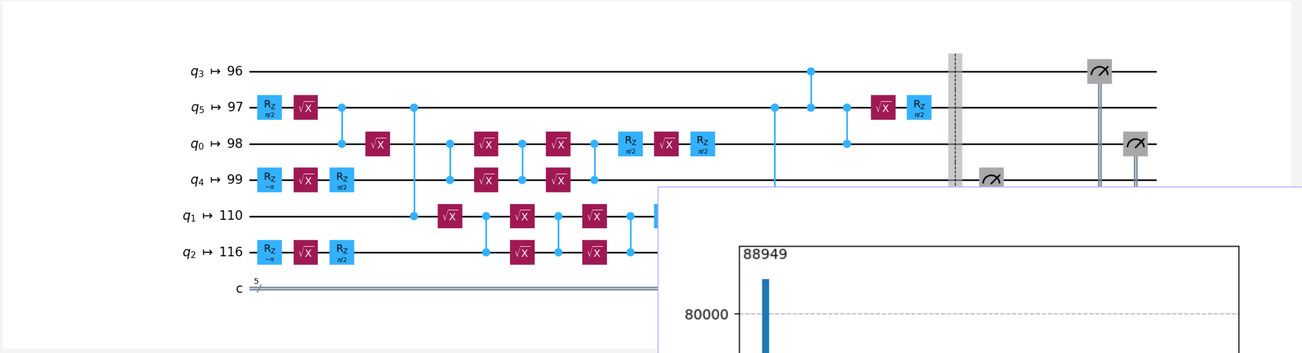
Why compiling?

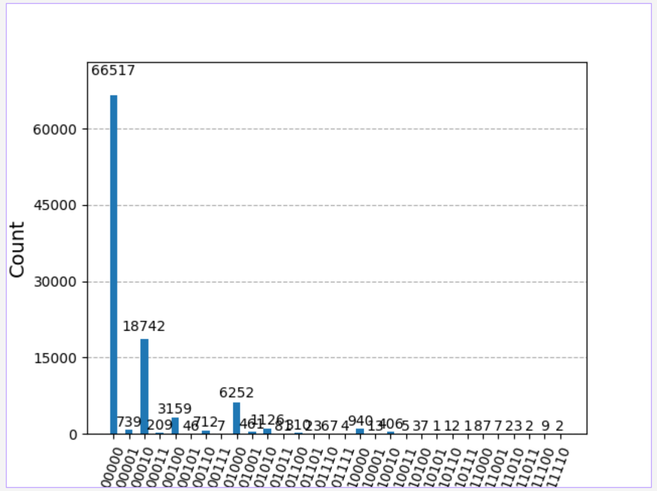
- Optimizations
- Instruction set
- Hardware connectivity
- Noise awareness

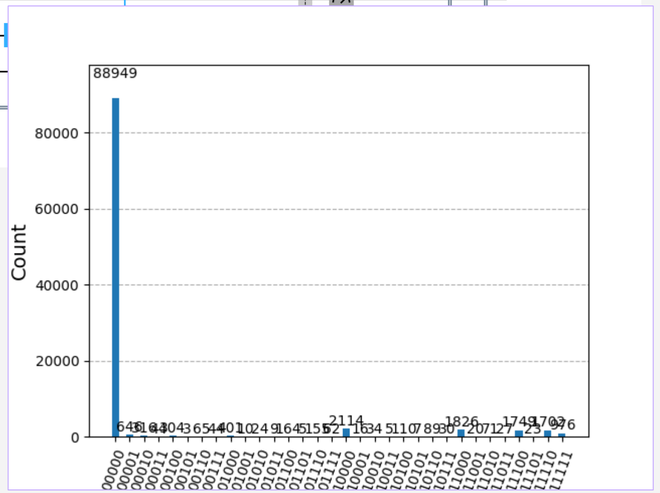


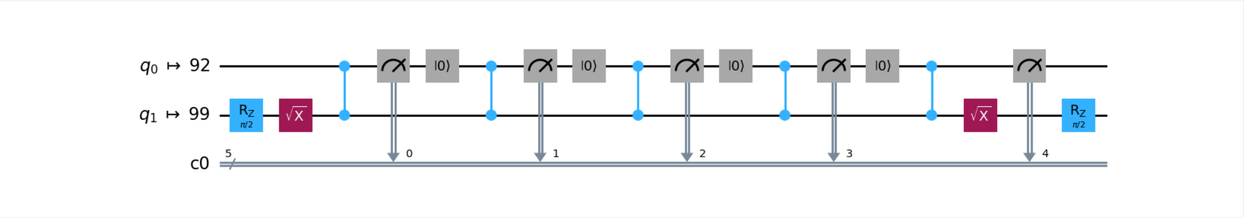




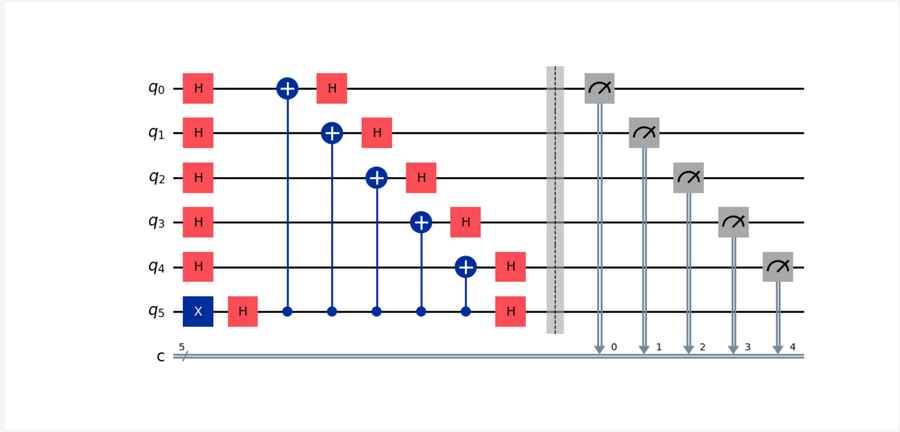


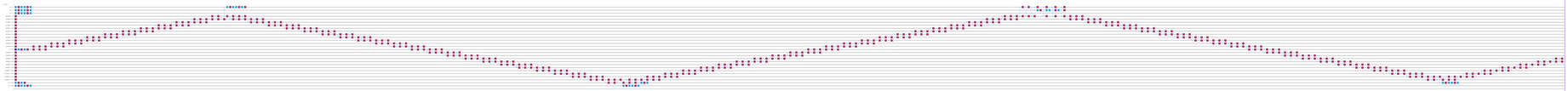


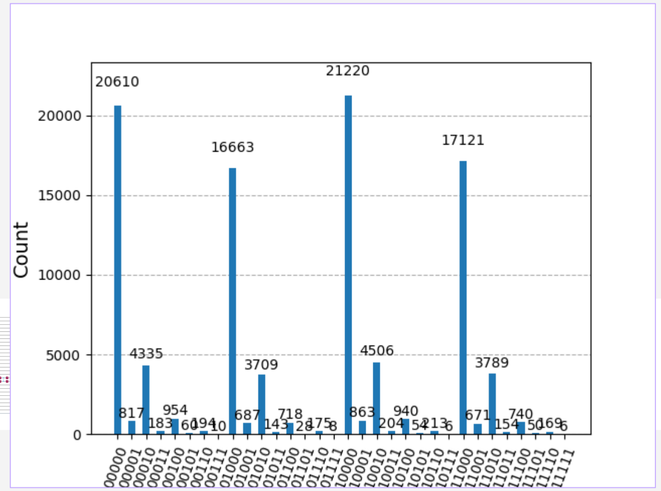


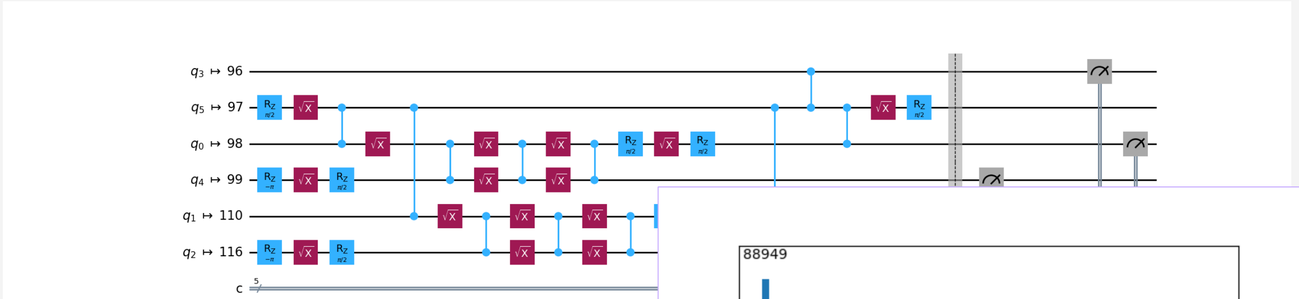


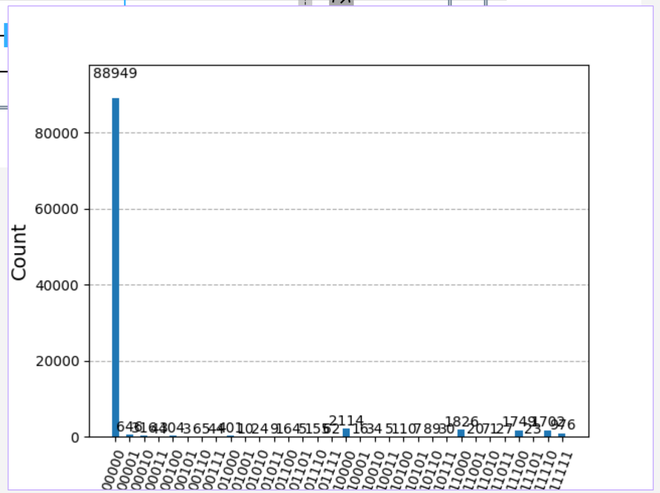
3	High Optimization
2	Medium optimization
1	Light optimization
0	No optimization

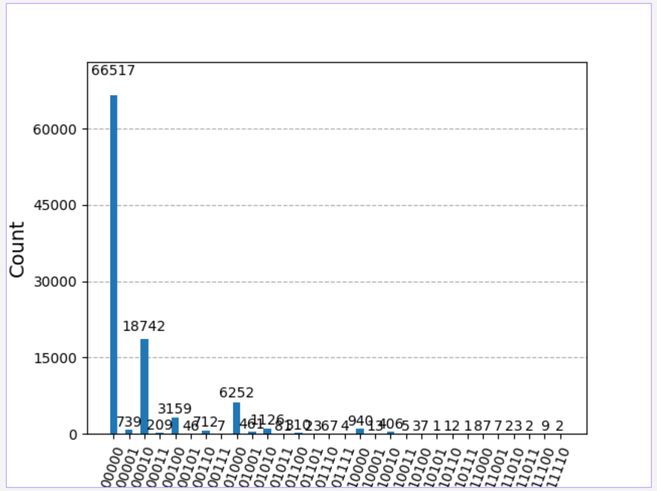


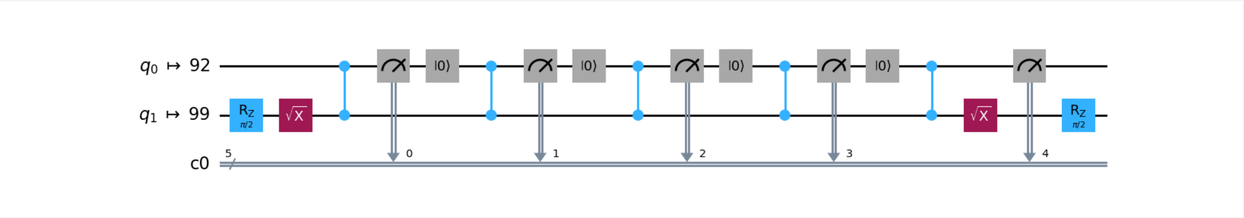


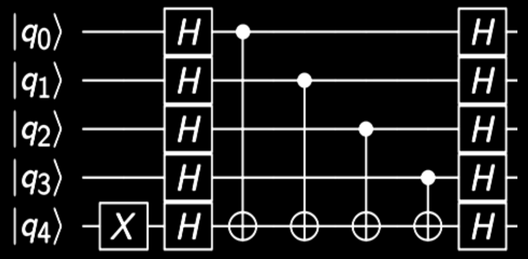












Preset Pass Manager

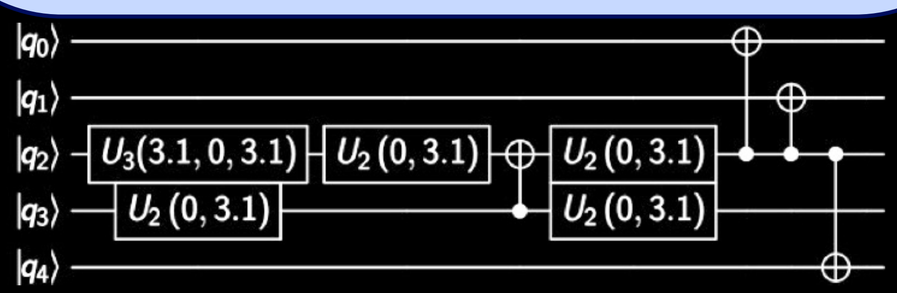
3	**High** Optimization
2	**Medium** optimization
1	**Light** optimization
0	**No** optimization

Pass Manager

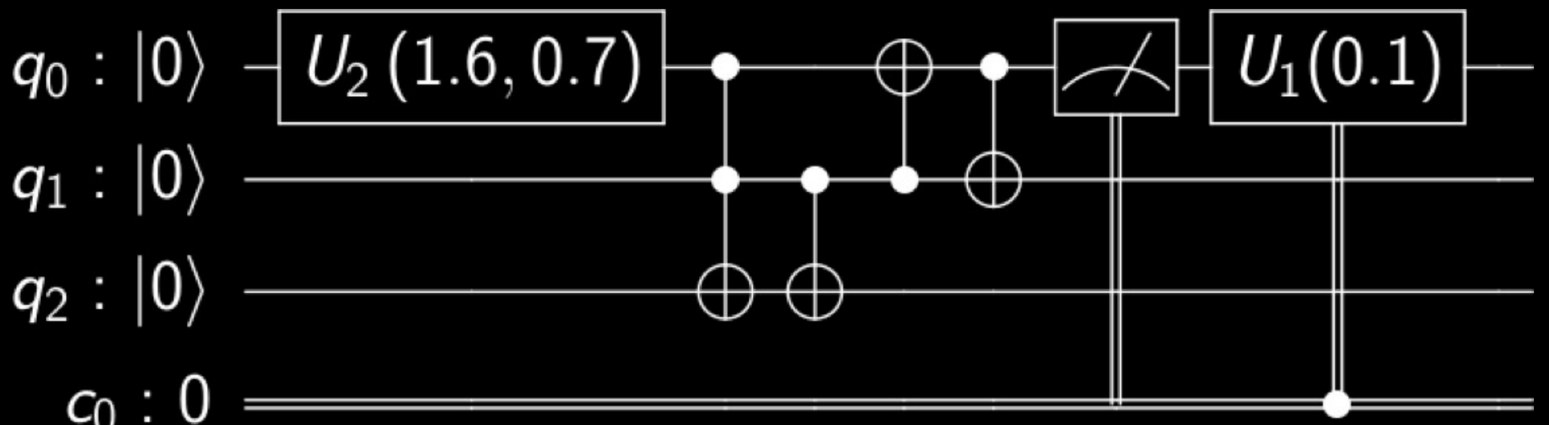
- Pass 1: Unroll to basis
- Pass 2: Cancel redundant gates
- Pass 3: Map to device by swapping
- Pass 4: Cancel redundant gates

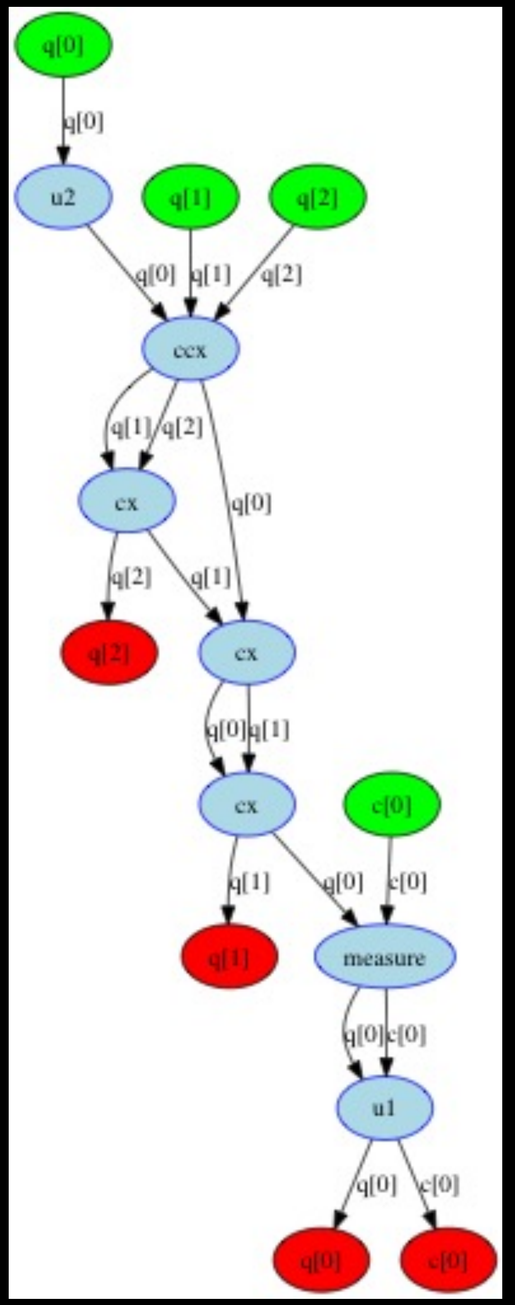
Property Set

- Depth
- Commuting Sets
- Layout
- Fixed Point Condition

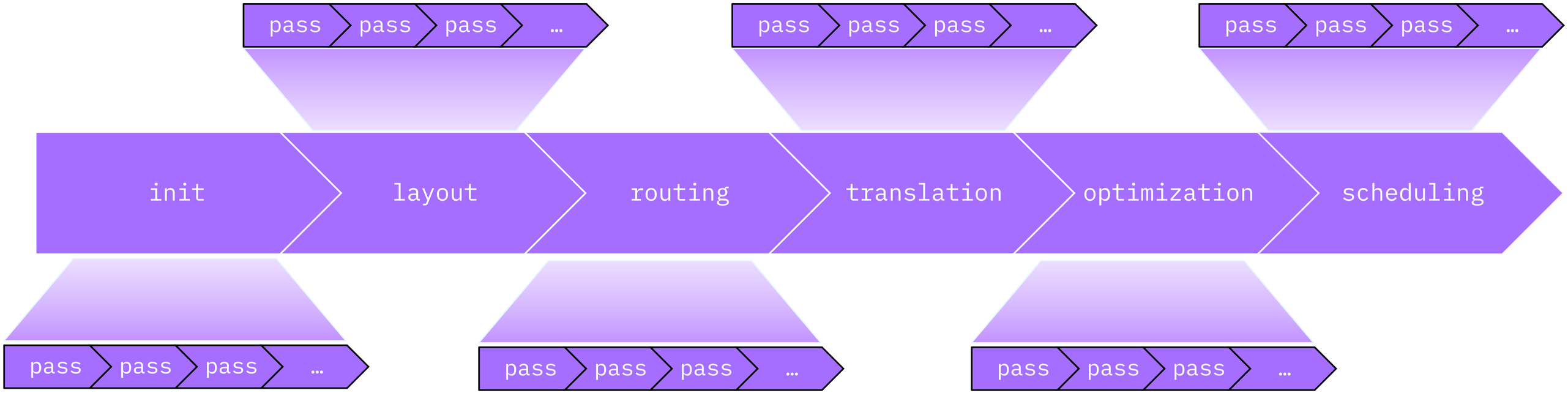


Directed Acyclic Graph



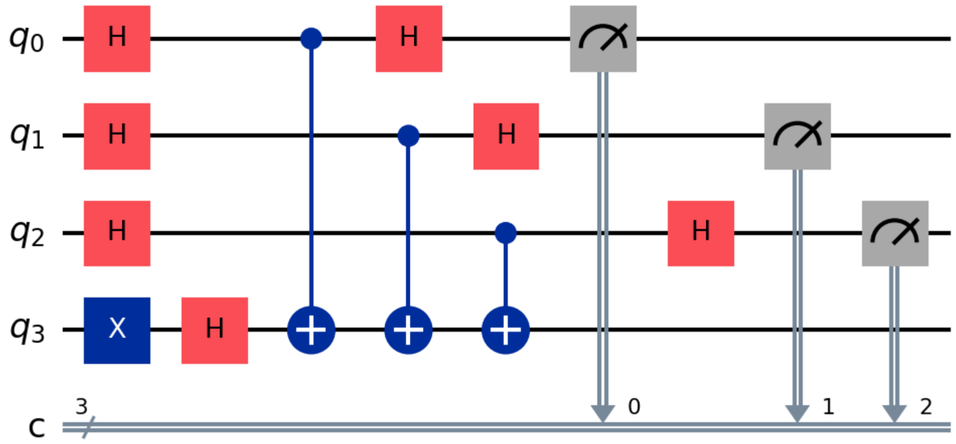


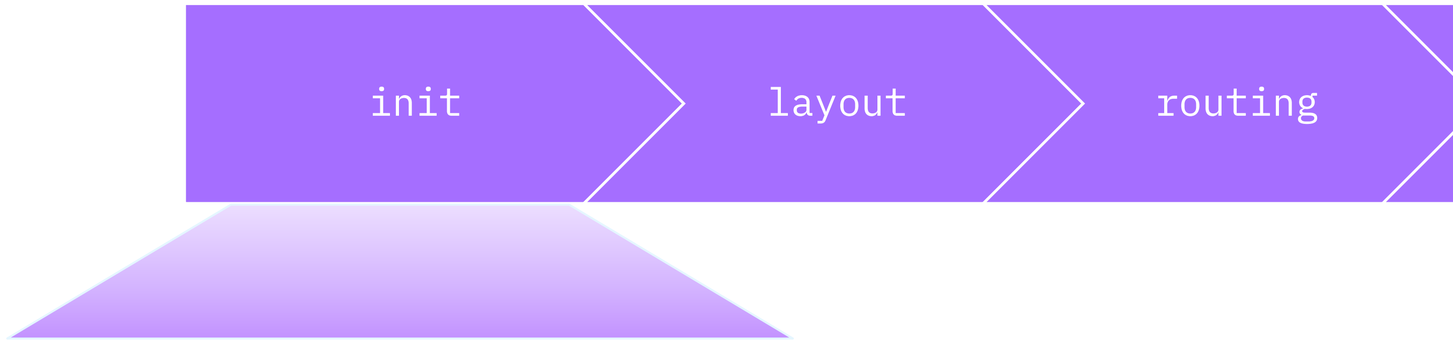
Stages of a Pass Manager



Stages of a Pass Manager - Plugins

```
pip install qiskit-qubit-reuse
```

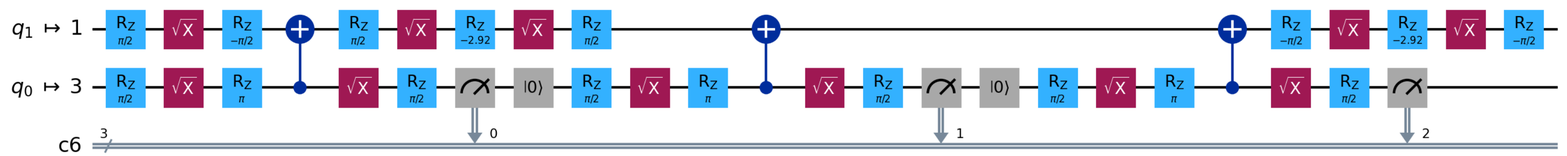





```
transpile(qc, target, init_method="qubit_reuse")
```

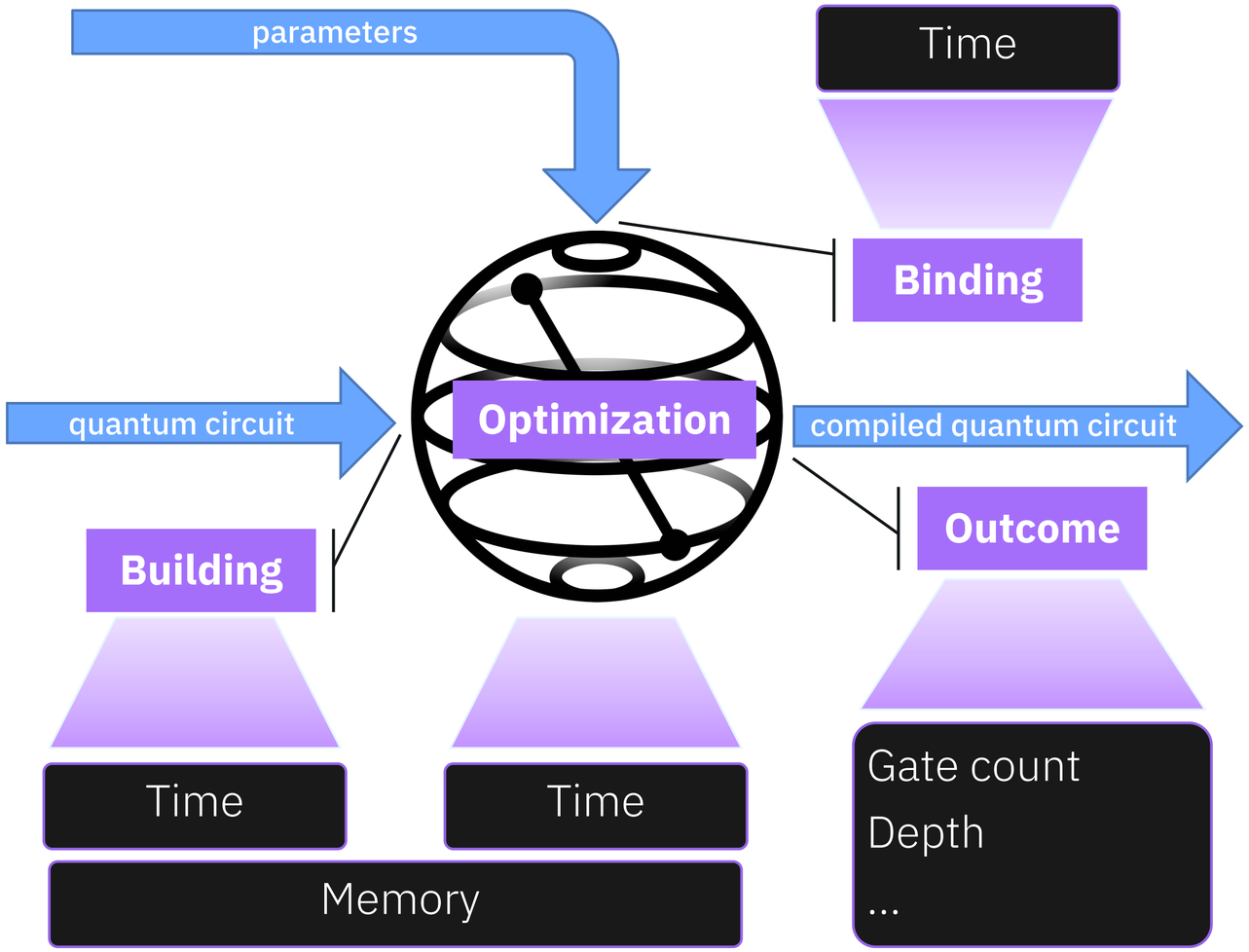
Stages of a Pass Manager - Plugins

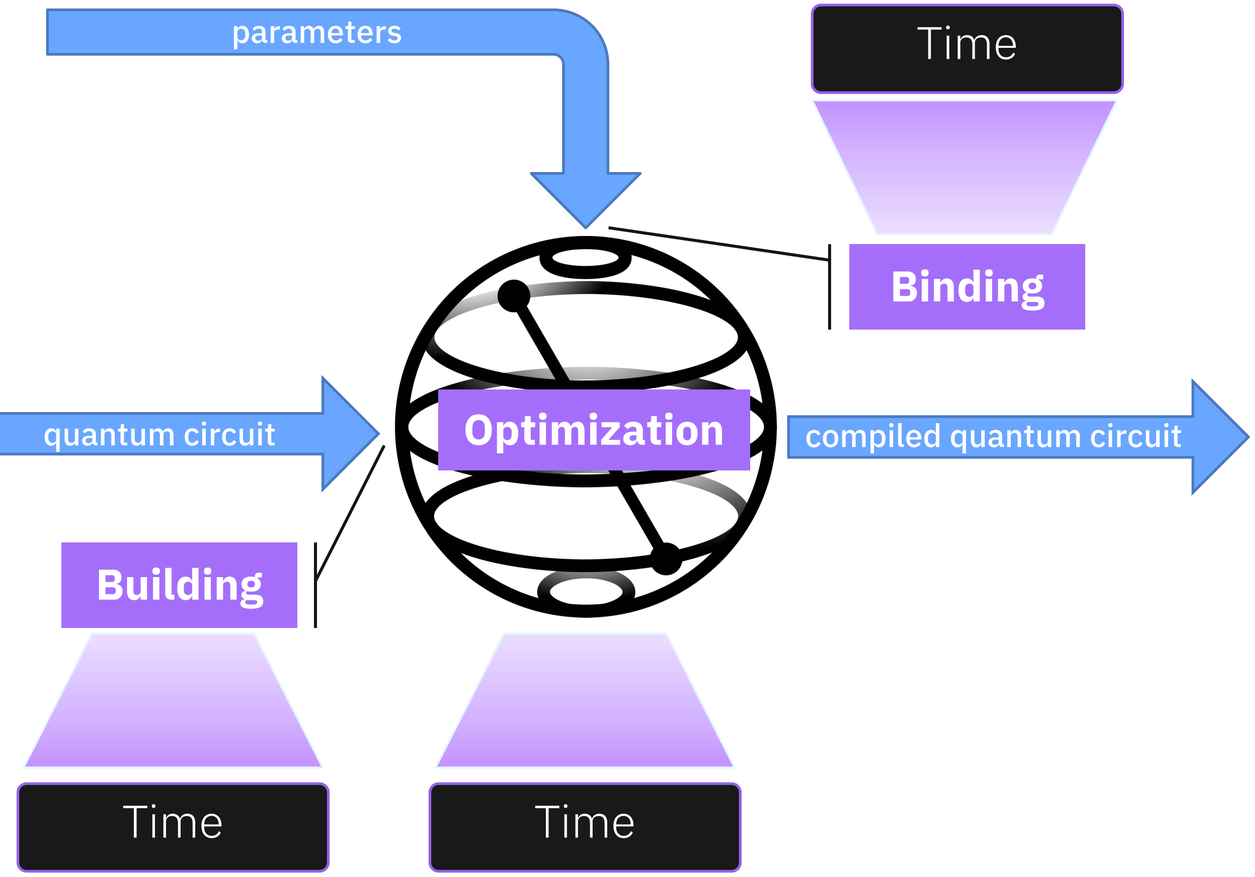
```
pip install qiskit-qubit-reuse
```

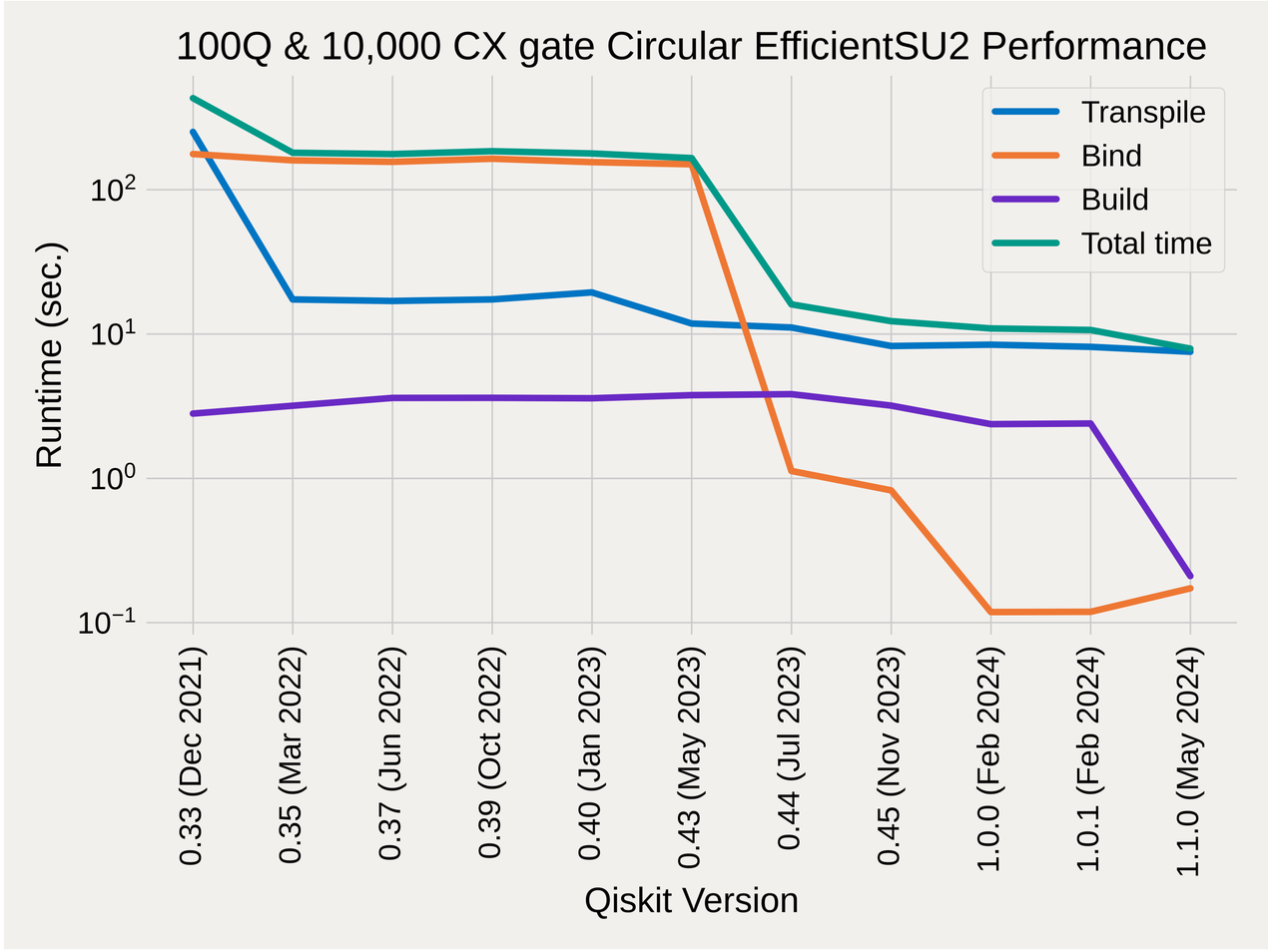


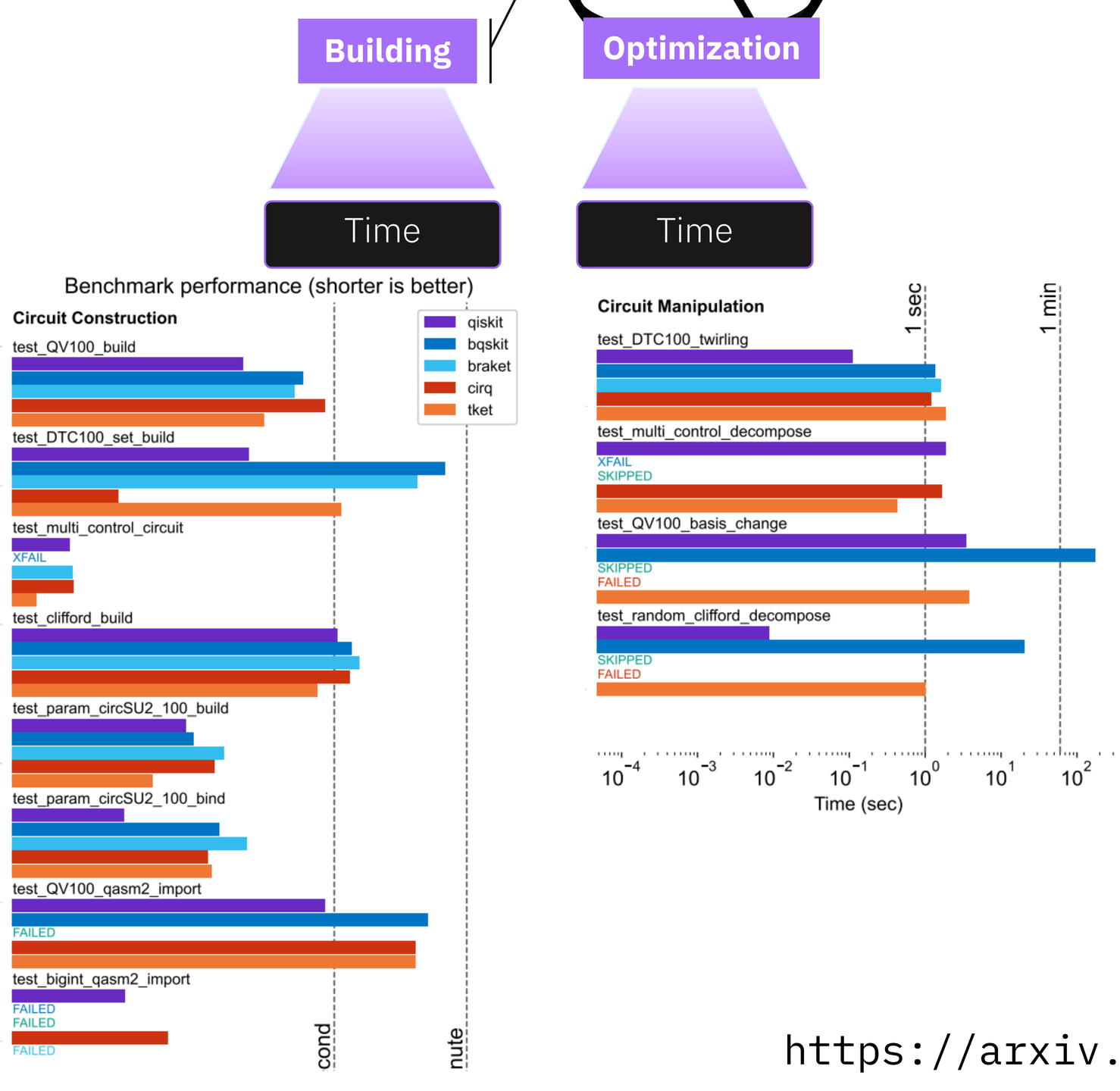

```
transpile(qc, target, init_method="qubit_reuse")
```

Performance

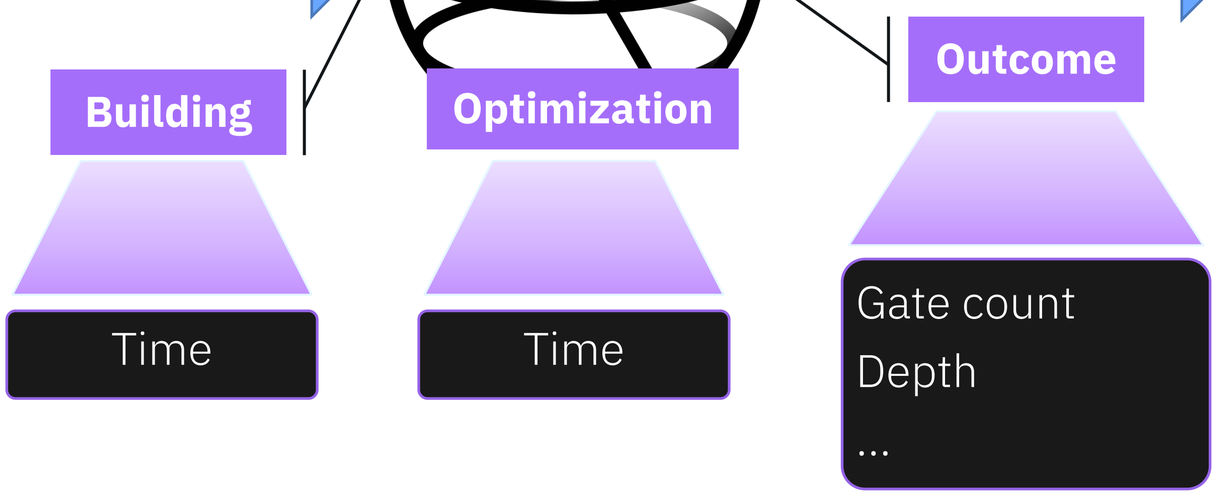



parameters
Time
Binding
quantum circuit
Optimization
compiled quantum circuit
Building
Time
Time


100Q & 10,000 CX gate Circular EfficientSU2 Performance
Transpile
Bind
Build
Total time
Runtime (sec.)
10^2
10^1
10^0
10^{-1}
0.33 (Dec 2021)
0.35 (Mar 2022)
0.37 (Jun 2022)
0.39 (Oct 2022)
0.40 (Jan 2023)
0.43 (May 2023)
0.44 (Jul 2023)
0.45 (Nov 2023)
1.0.0 (Feb 2024)
1.0.1 (Feb 2024)
1.1.0 (May 2024)
Qiskit Version

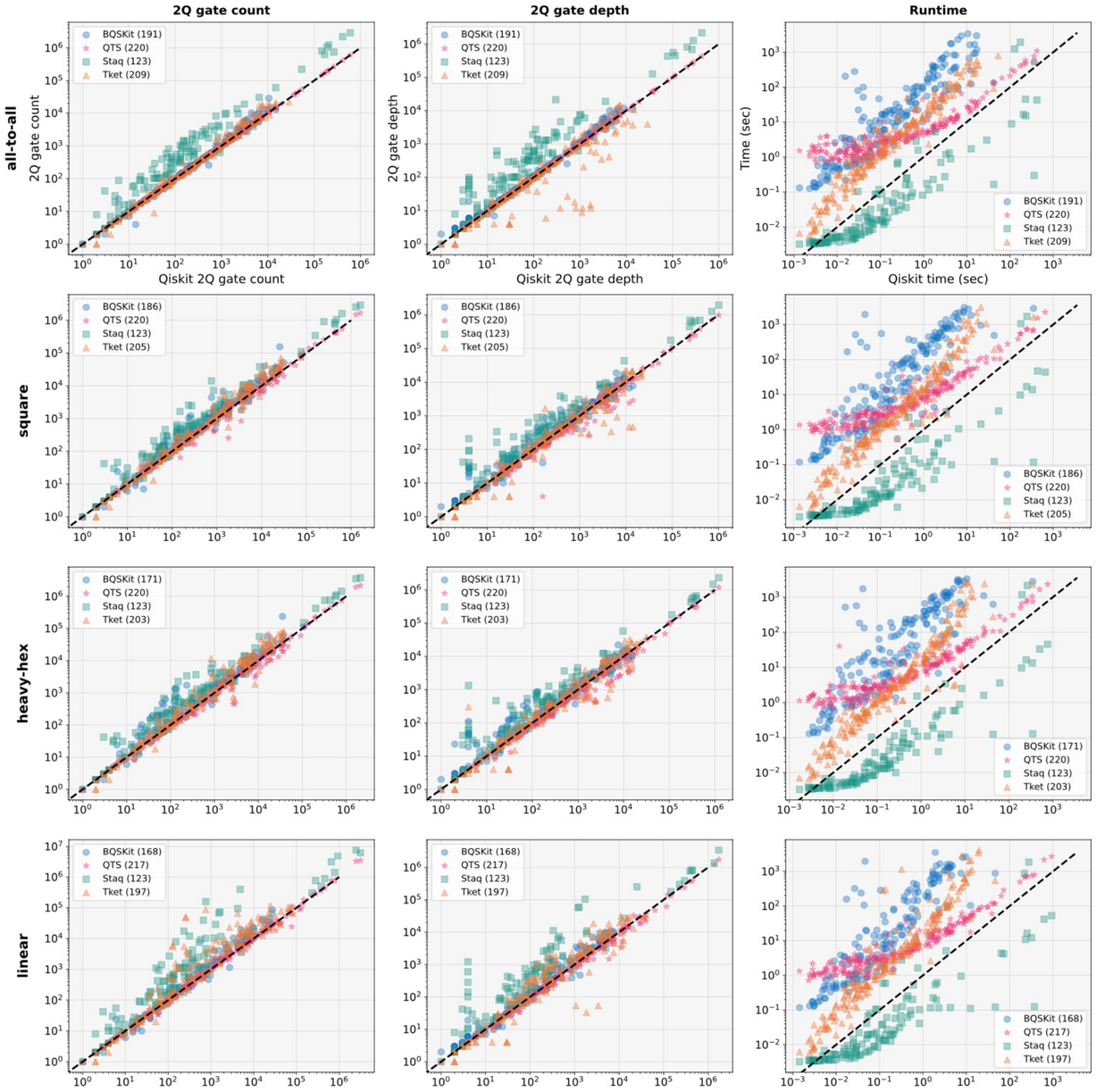


https://arxiv.org/abs/2409.08844

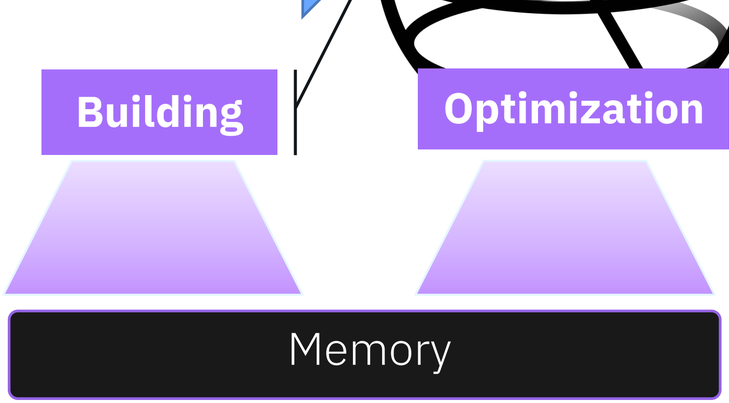


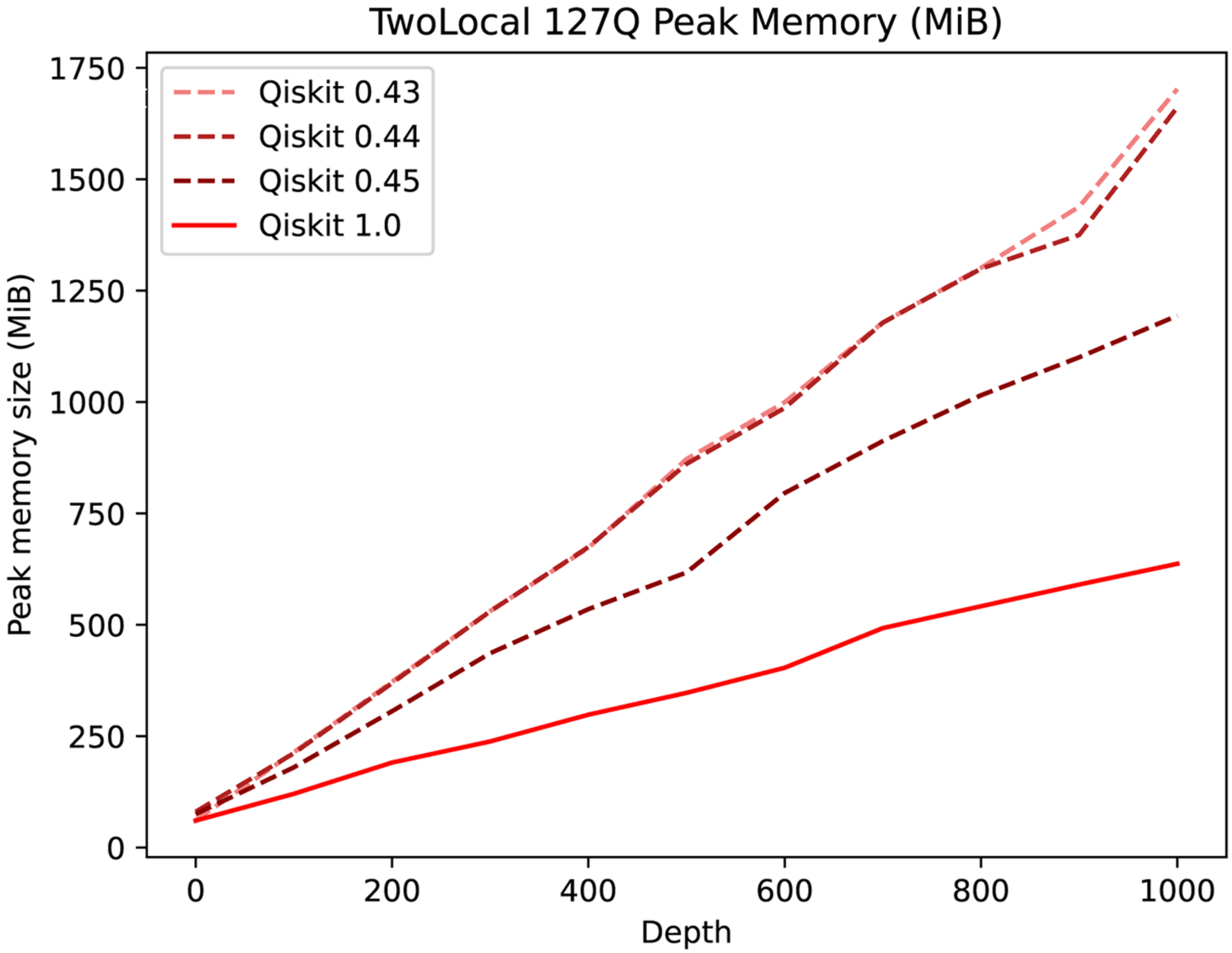
- Qiskit compared to Tket:
  - **53%** mean reduction on 2Q gate count
  - **29%** mean transpilation time reduction

- Qiskit compared to BQSKit:
  - **29%** mean reduction on 2Q gate count
  - **572%** mean transpilation time reduction



https://arxiv.org/abs/2409.08844





https://arxiv.org/abs/2409.08844

112

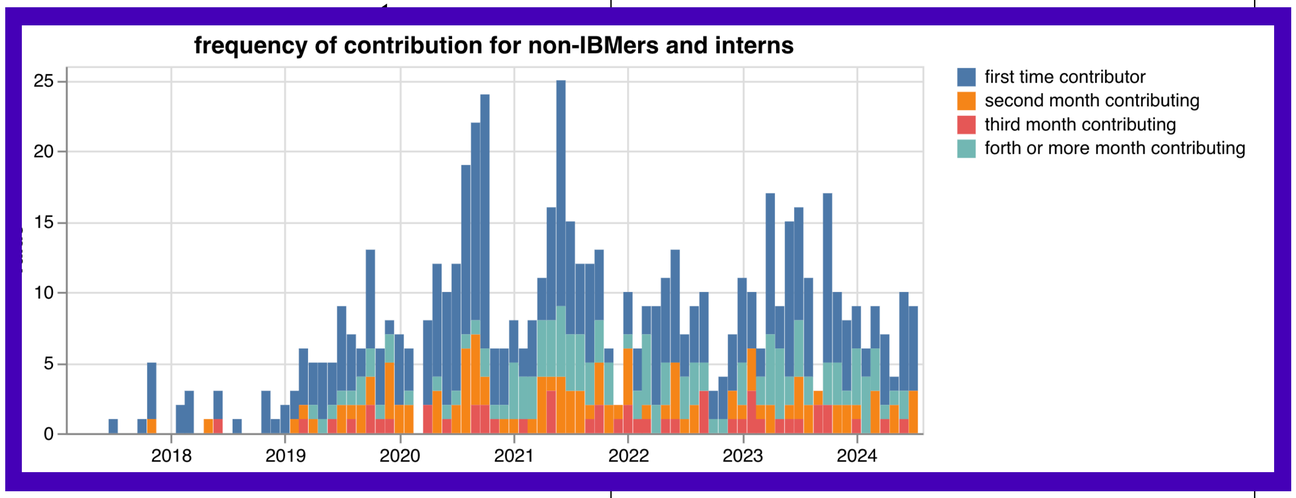
releases, including patch and pre-releases.

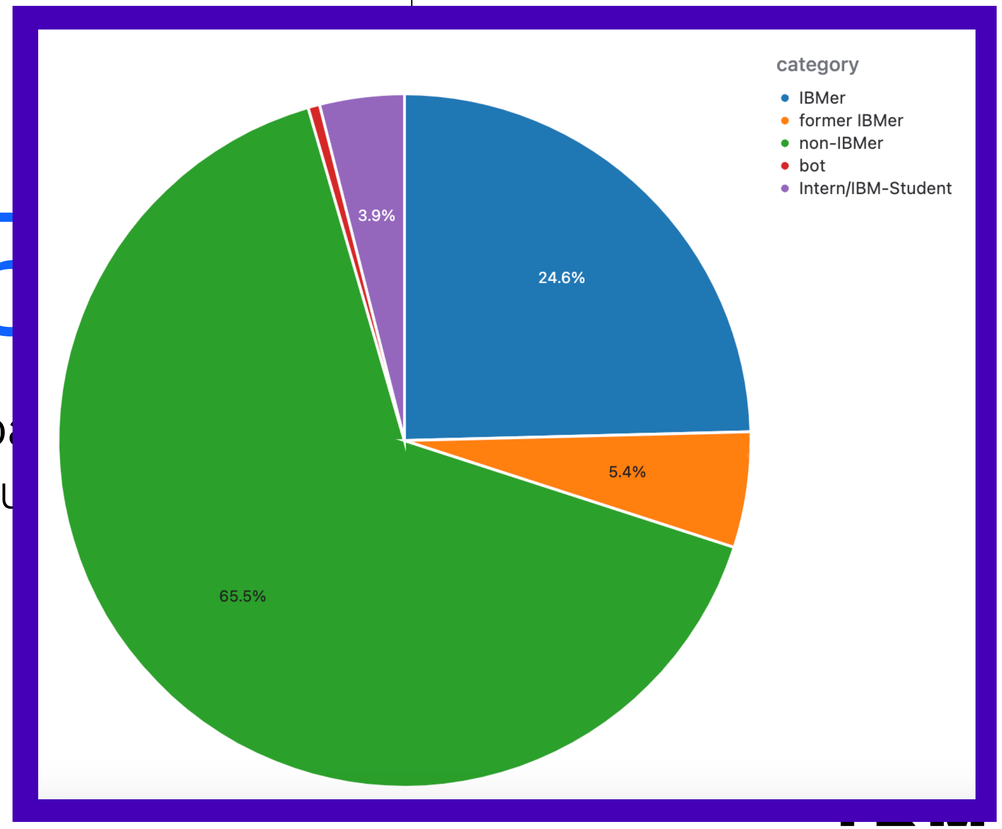
546

contributors, many first-OSS timers. 2.2k forks and 4.4k GitHub stars.

+1600

research papers extending, using, and benchmarking Qiskit



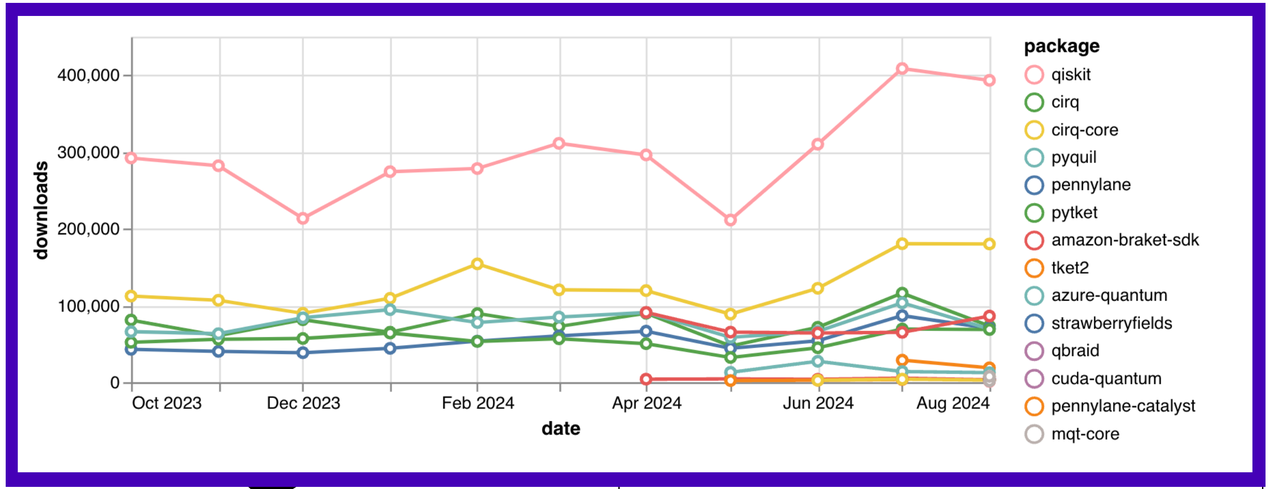


7.5

downloa
since Au

position in Unitary Fund
and
t used
Quantum Framework

qisk.it/good-first-issues
qisk.it/help-wanted



Qiskit

546

contributors, many first-OSS timers. 2.2k forks and 4.4k GitHub stars.

+1600

research papers extending, using, and benchmarking Qiskit

1st

position in Unitary Fund surveys (2022 and 2023) for most used Quantum Framework

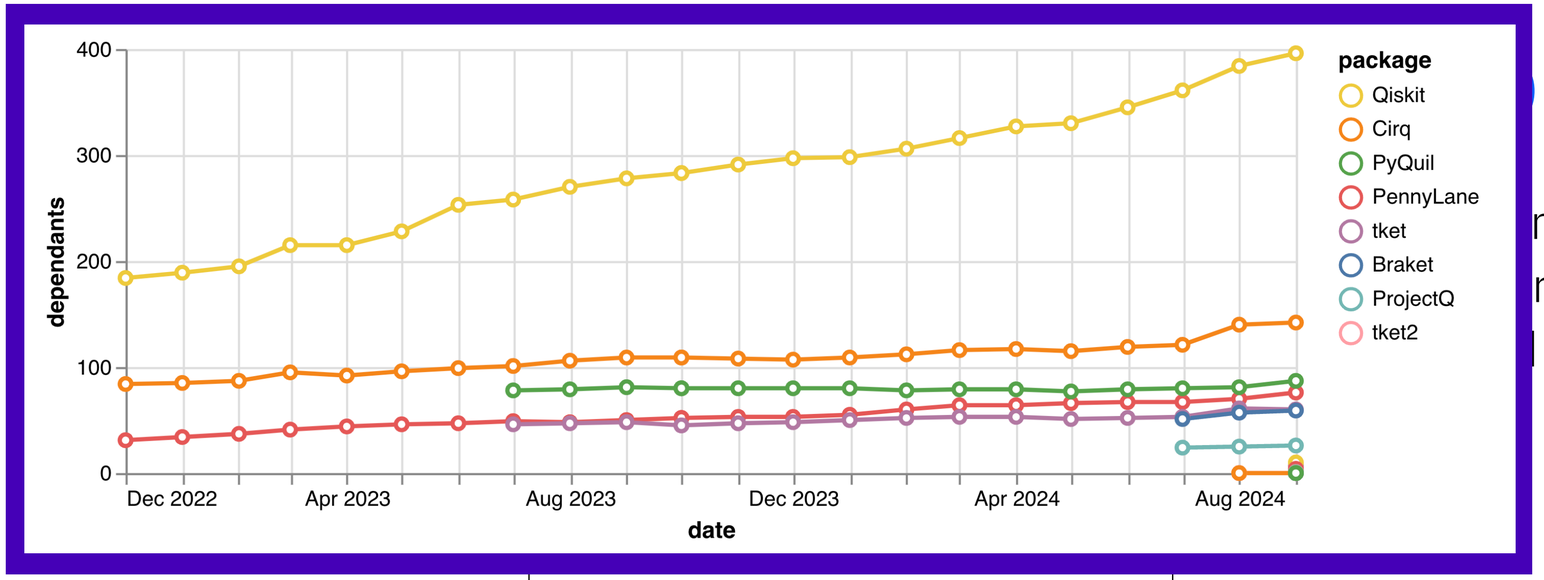
7.5 million

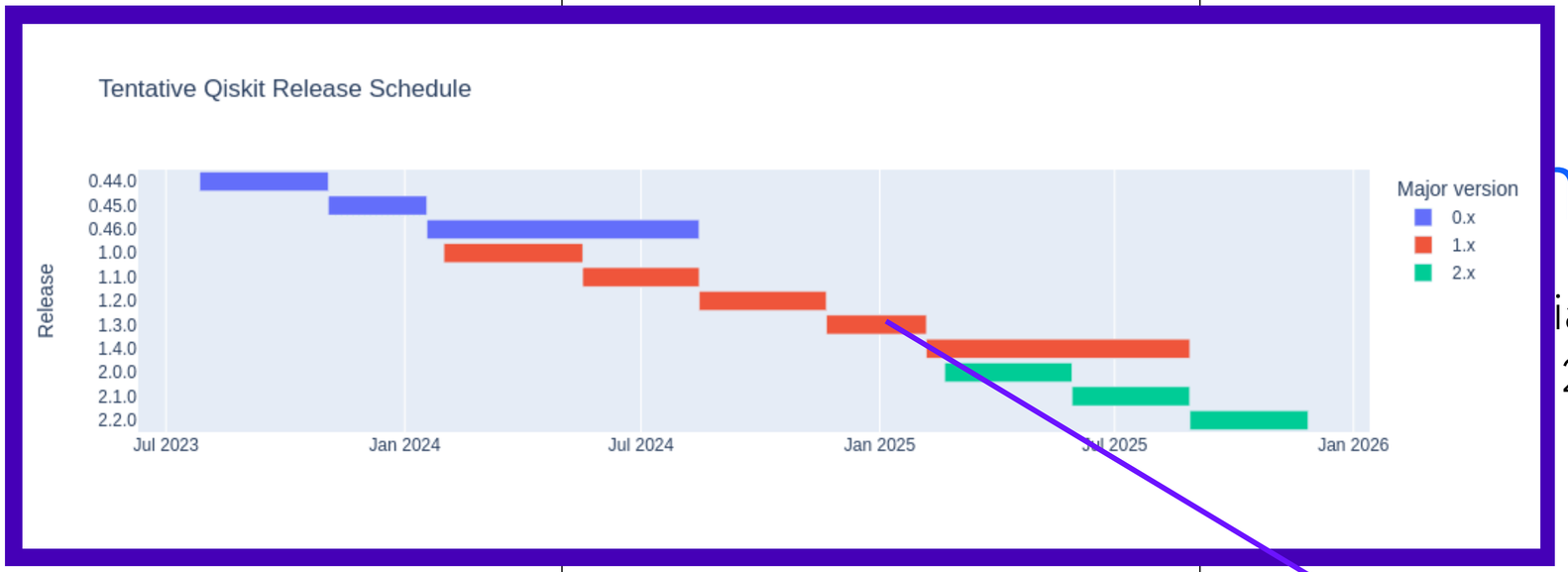
downloads via PyPI since August 2017

396

dependants, i.e packages that directly or indirectly depend on Qiskit

IBM





many
rs.
4.4k

illion

ia PyPI
2017

+1600

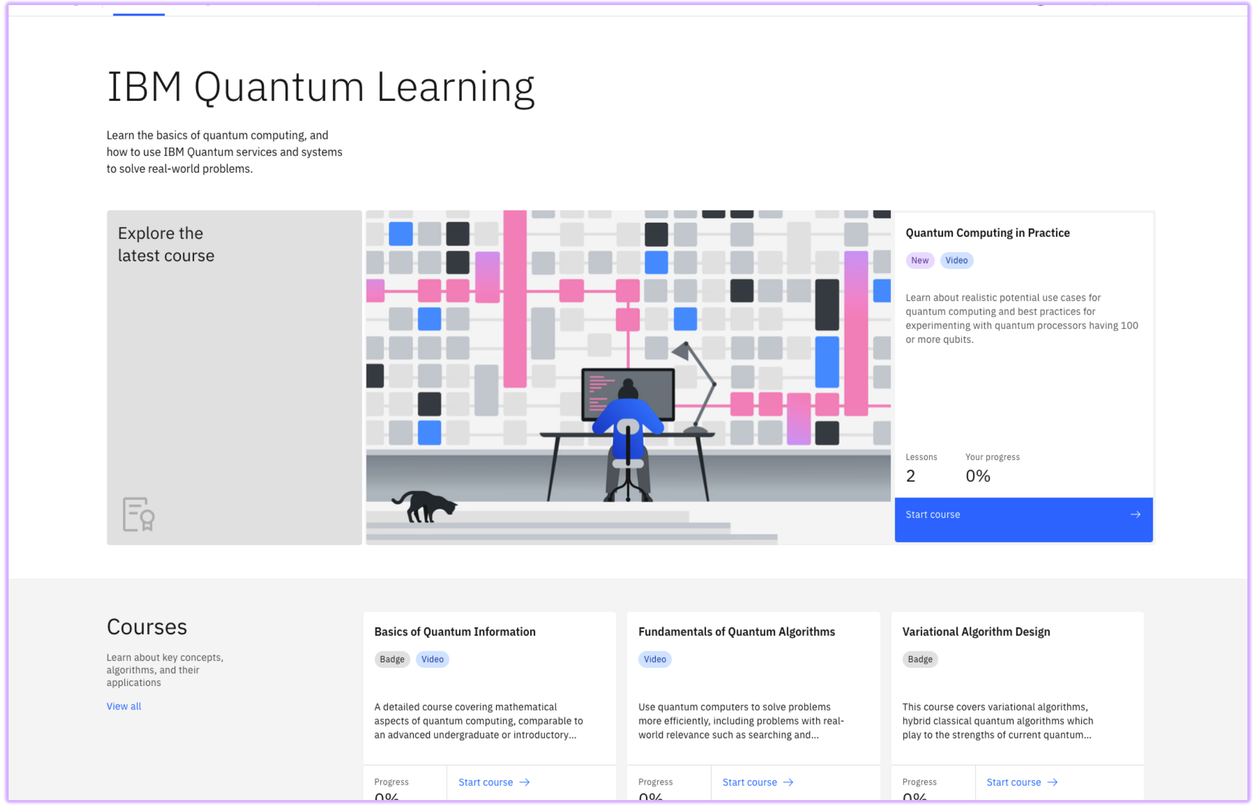
research papers extending, using, and benchmarking Qiskit

396

dependants, i.e packages that directly or indirectly depend on Qiskit

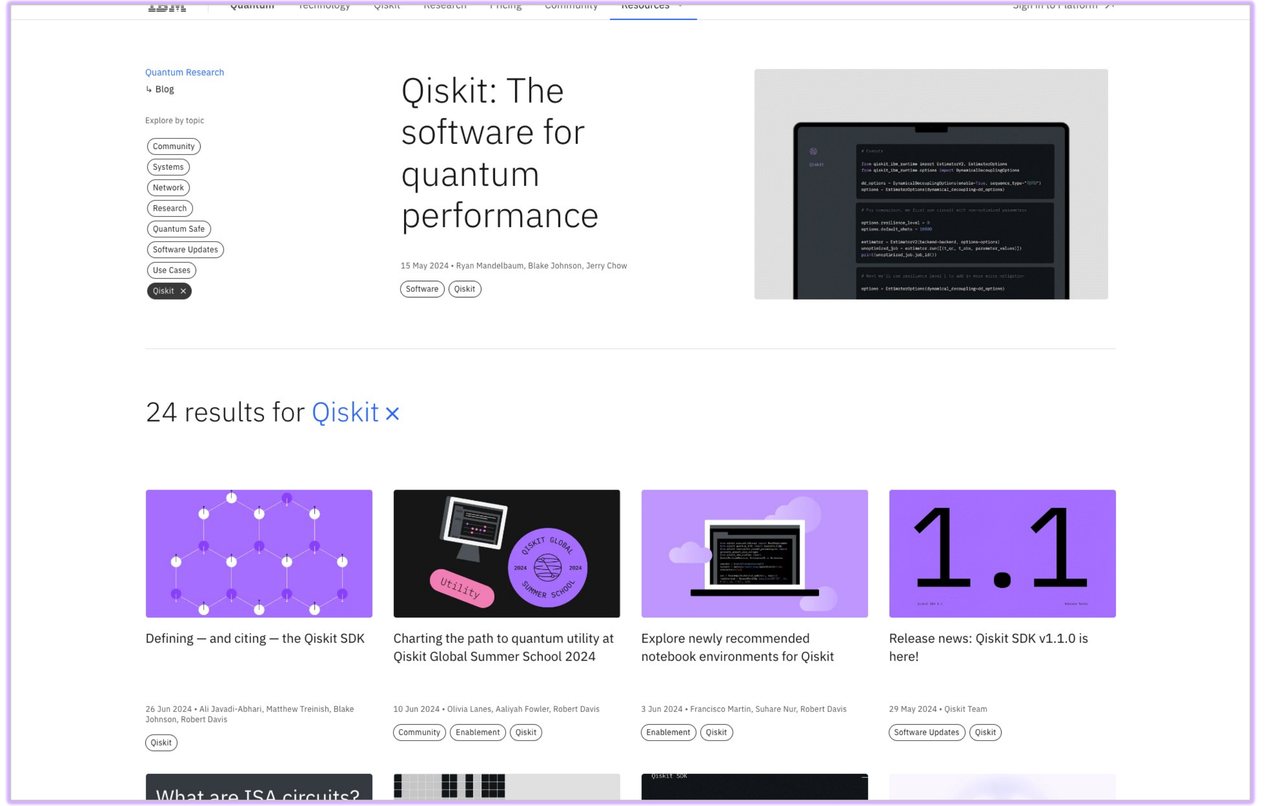
```
pip install "qiskit==1.3.0b1"
```

IBM Quantum Learning – to learn, experiment, and prototype quantum algorithms and applications

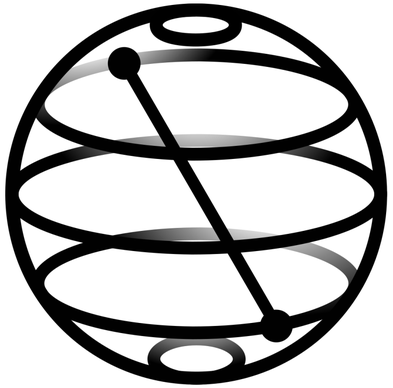


learning.quantum.ibm.com

IBM Quantum Blog – for news, announcements, and release reviews



www.ibm.com/quantum/blog



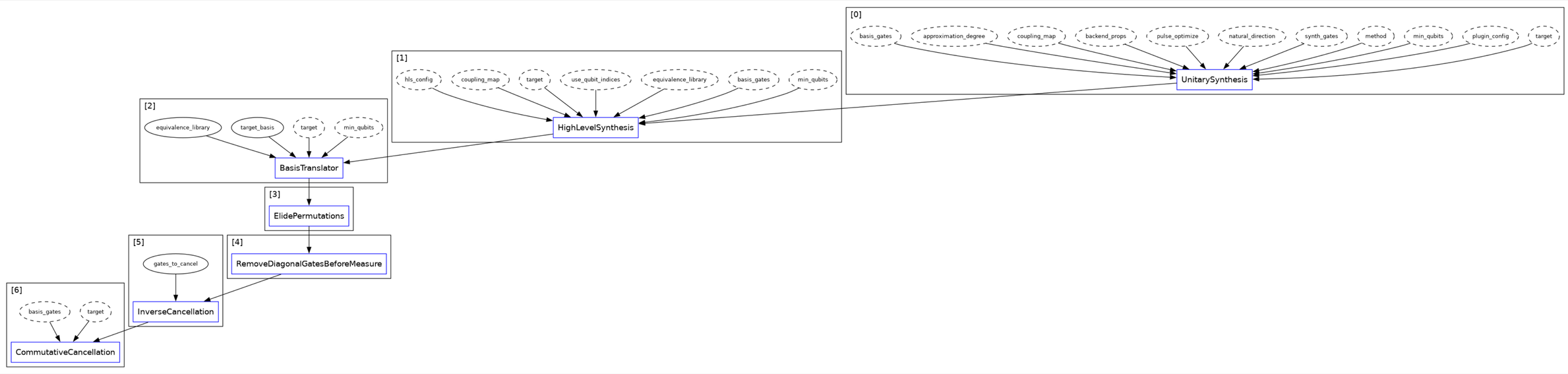
www.ibm.com/quantum/qiskit

github.com/Qiskit/

Preset Pass Managers

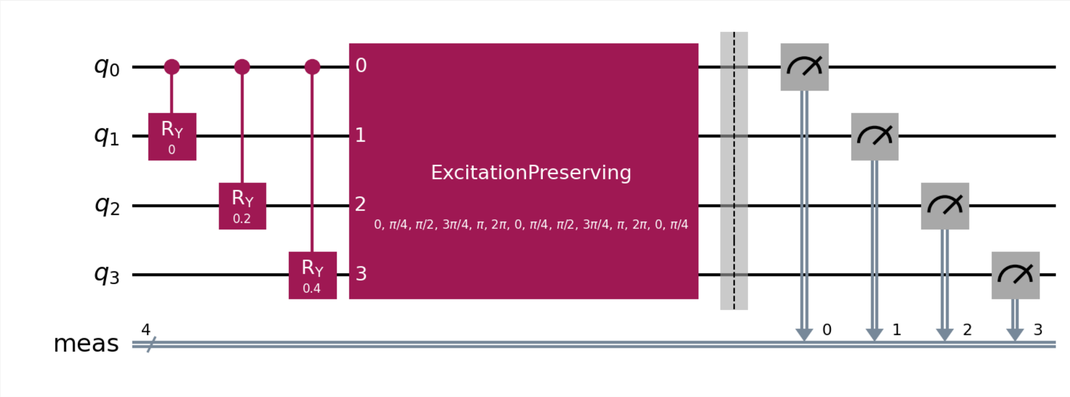
Init stage

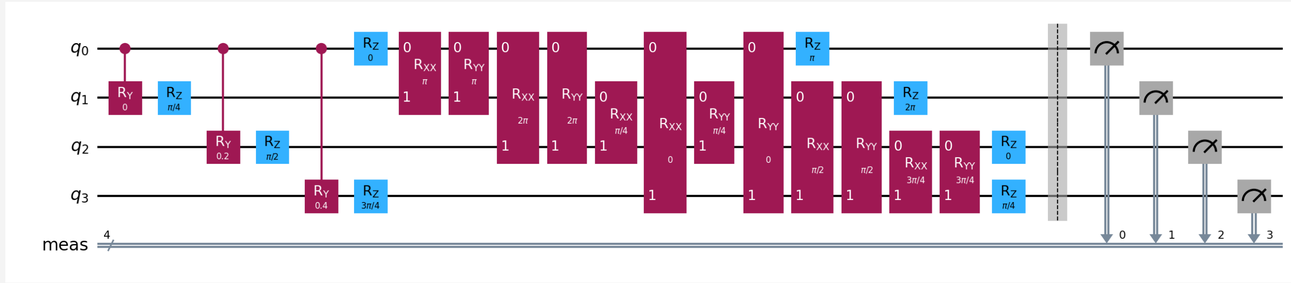
- Stage is broken into two phases:
  - Decompose larger (>=3 qubit) gates to 1 and 2 qubit gates
  - Logical optimizations
- At the end of the stage, we are able to run layout (which models the circuit as a graph)



Init stage

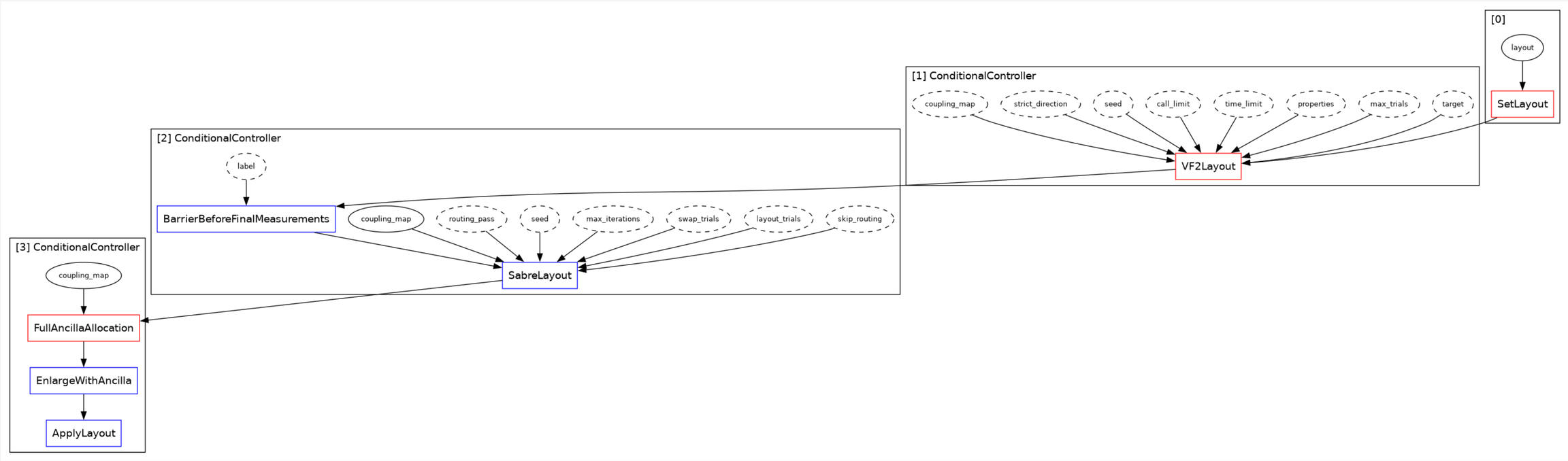
- Stage is broken into two phases:
  - Decompose larger (>=3 qubit) gates to 1 and 2 qubit gates
  - Logical optimizations
- At the end of the stage, we are able to run layout (which models the circuit as a graph)



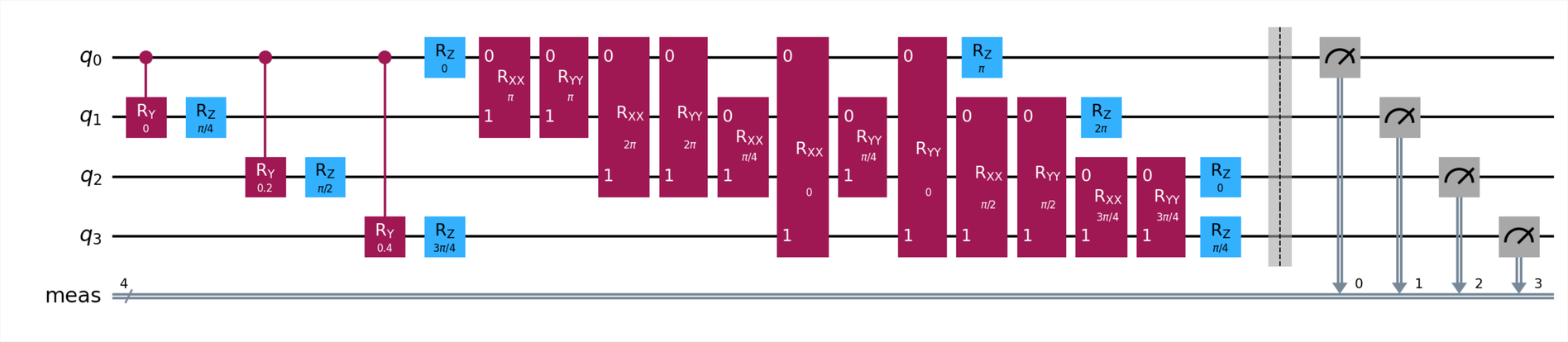


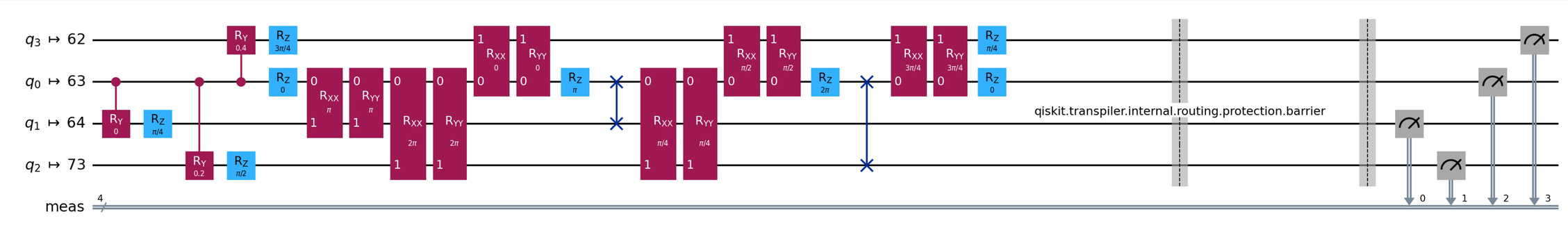
Layout stage

- This stage takes the virtual qubits from abstract circuit and maps them to a qubit as defined in the target
- Layout is of critical importance because not all qubits perform the same.
- This stage typically tries multiple techniques to find the best layout. In Qiskit we use two passes by default: `VF2Layout` and `Sabre`
- A poor layout can induce more Swaps because of the connectivity



Layout stage





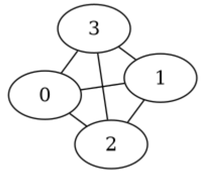
VF2Layout

This pass creates a graph from the 2 qubit interactions in the circuit and tries to find an isomorphic subgraph of the connectivity graph of the target. If a subgraph is found than that indicates a layout which does not require swaps can be used. If a layout is found, then the pass will search for additional layouts and pick the one with the lowest predicted error rate.

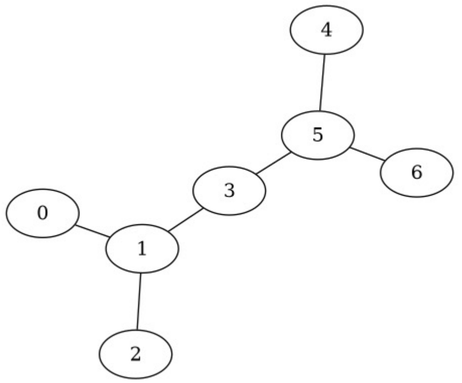
It leverages the rustworkx's library's VF2 algorithm implementation to efficiently search for an isomorphism.

https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.passes.VF2Layout

Interaction Graph



Connectivity Graph



Sabre

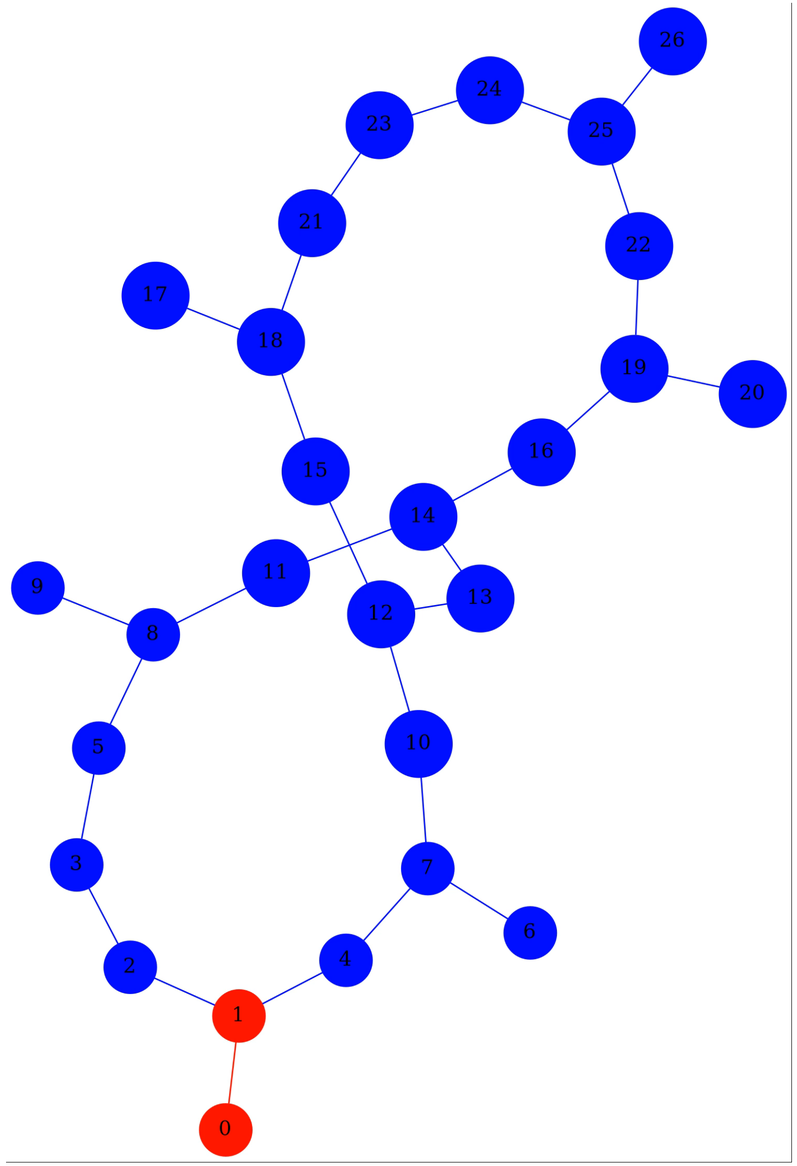
If `VF2Layout` does not find a layout then we run `SabreLayout` pass. This pass is based on the SABRE (SWAP-based BidiREctional) algorithm originally detailed in:

https://arxiv.org/pdf/1809.02573.pdf.

The SABRE algorithm works by starting with an initial random guess, then running a routing algorithm on it to insert swap gates and instead of inserting a swap it swaps the qubits in the layout. It fully "routes" the circuit then reverses the edge direction in the dag and repeats. This is performed multiple times, doing this will minimize the number of swaps needed.

The implementation in Qiskit has significantly changed and improved on the algorithm from the original paper.

https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.passes.SabreLayout



Sabre

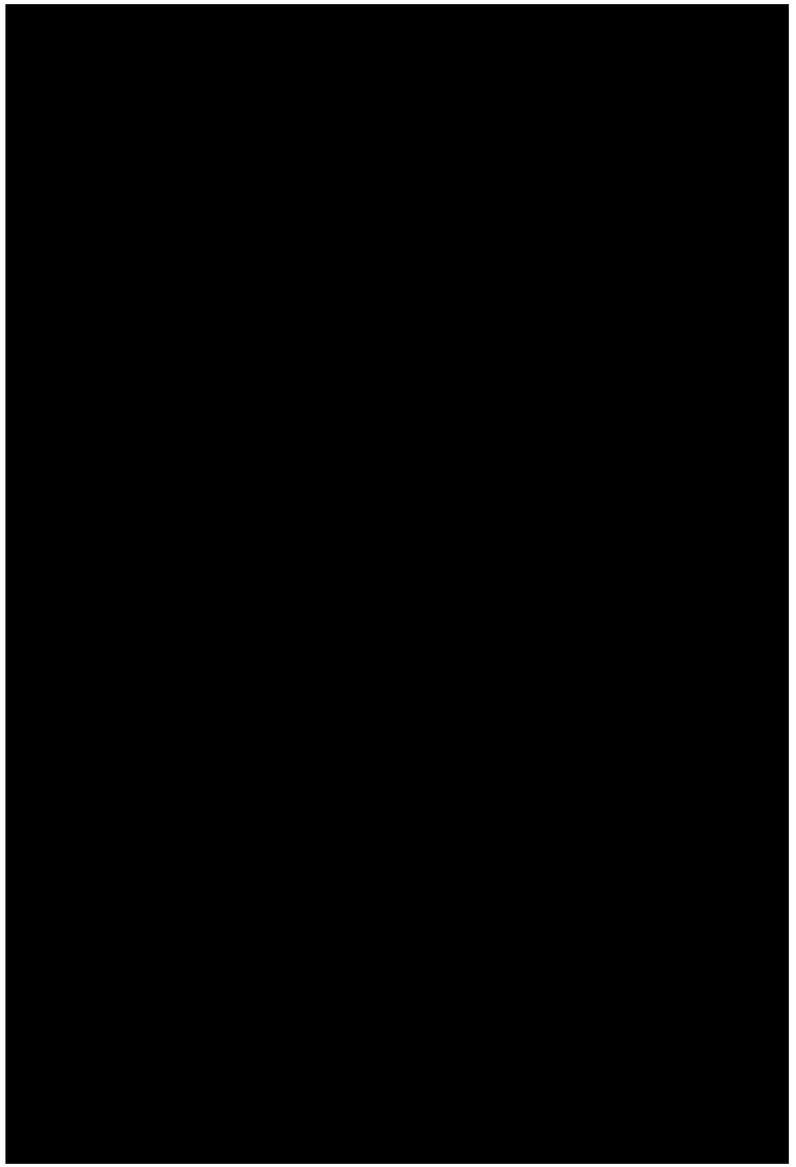
If `VF2Layout` does not find a layout then we run `SabreLayout` pass. This pass is based on the SABRE (SWAP-based BidiREctional) algorithm originally detailed in:

https://arxiv.org/pdf/1809.02573.pdf.

The SABRE algorithm works by starting with an initial random guess, then running a routing algorithm on it to insert swap gates and instead of inserting a swap it swaps the qubits in the layout. It fully "routes" the circuit then reverses the edge direction in the dag and repeats. This is performed multiple times, doing this will minimize the number of swaps needed.

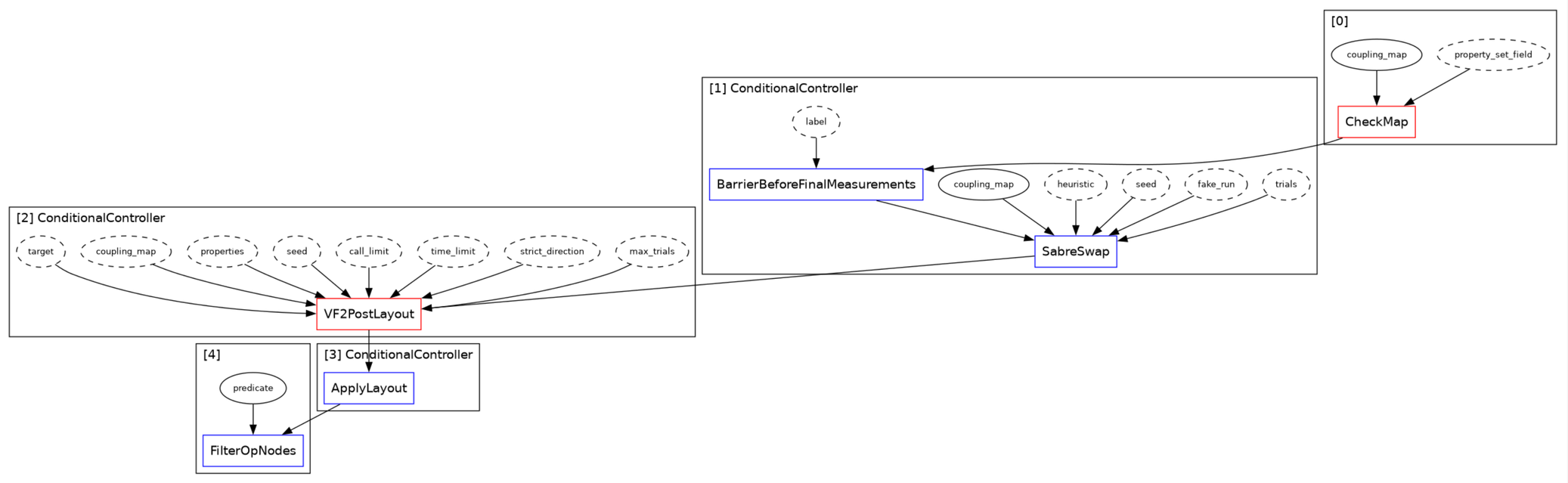
The implementation in Qiskit has significantly changed and improved on the algorithm from the original paper.

https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.passes.SabreLayout



Routing stage

- This stage takes the circuit with a layout set and inserts Swap gates where necessary.
- The default pass for this is `SabreSwap`.
- If running with defaults however this pass is skipped for efficiency, and it's run as part of `SabreLayout` in the Layout stage.
- We re-run layout with `VF2PostLayout` after swap insertion to search for a better layout



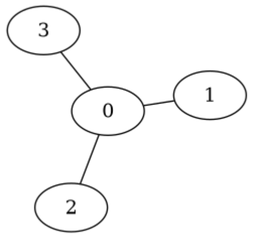
VF2PostLayout

Just as with VF2Layout this pass creates a graph from the 2 qubit interactions in the circuit and tries to find an isomorphic subgraph of the connectivity graph of the target. The difference is by running after routing we know that there is at least one isomorphic subgraph. This pass then searches for other layouts and picks the one found with the lowest predicted error rate.

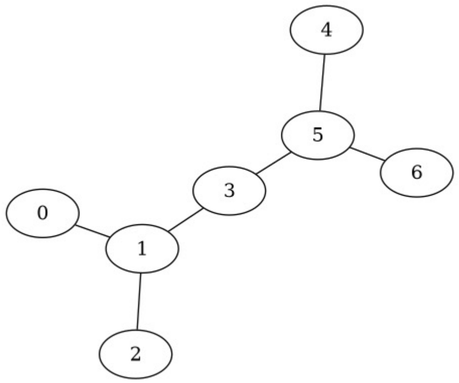
https://journals.aps.org/prxquantum/abstract/10.1103/PRXQuantum.4.010327

https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.passes.VF2PostLayout

Interaction Graph



Connectivity Graph

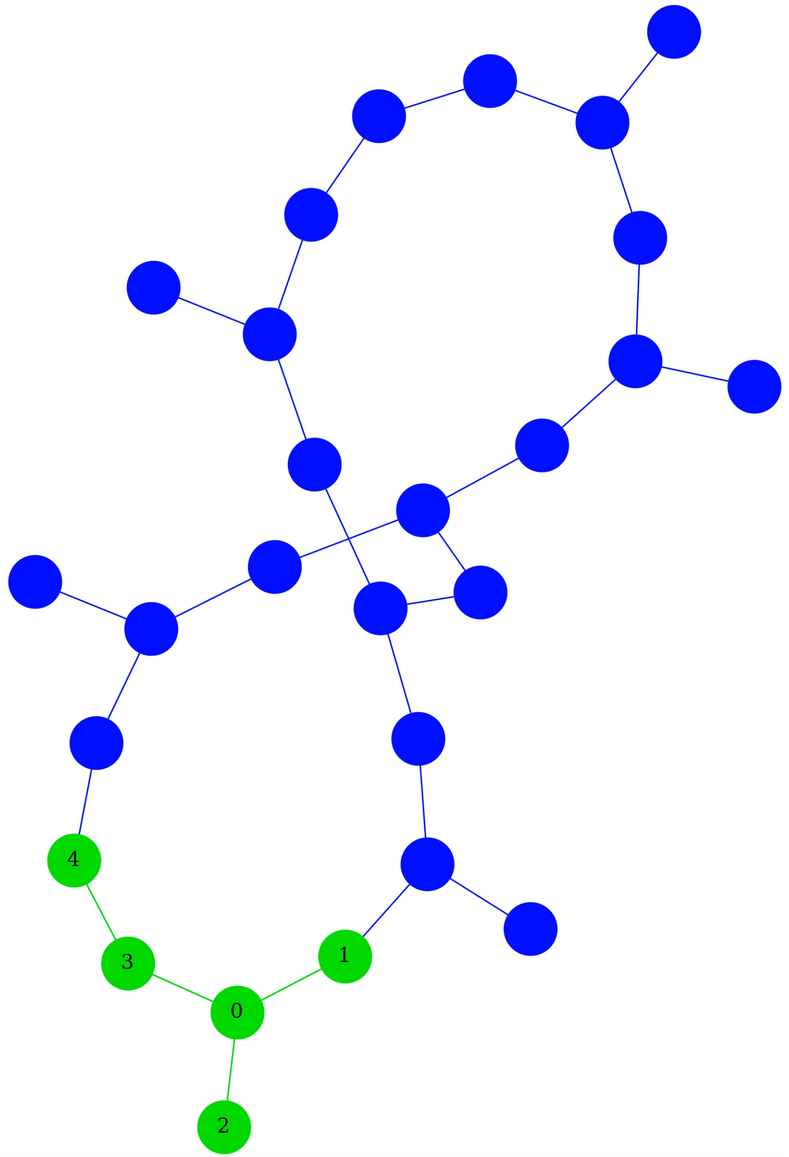


VF2PostLayout

Just as with VF2Layout this pass creates a graph from the 2 qubit interactions in the circuit and tries to find an isomorphic subgraph of the connectivity graph of the target. The difference is by running after routing we know that there is at least one isomorphic subgraph. This pass then searches for other layouts and picks the one found with the lowest predicted error rate.

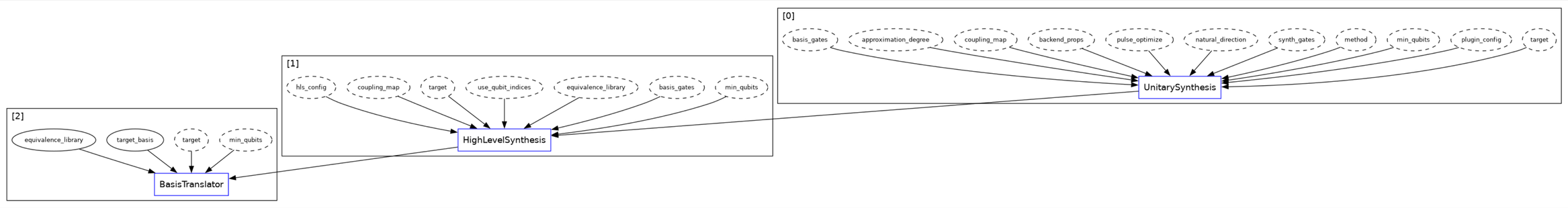
https://journals.aps.org/prxquantum/abstract/10.1103/PRXQuantum.4.010327

https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.passes.VF2PostLayout



Translation stage

- This stage takes the routed circuit which meets the connectivity constraints of the target and is responsible for making sure that all the operations are supported on the target
- This stage runs the same passes we used in the first phase of the init stage but for all operations instead of just those with >= 3 qubits

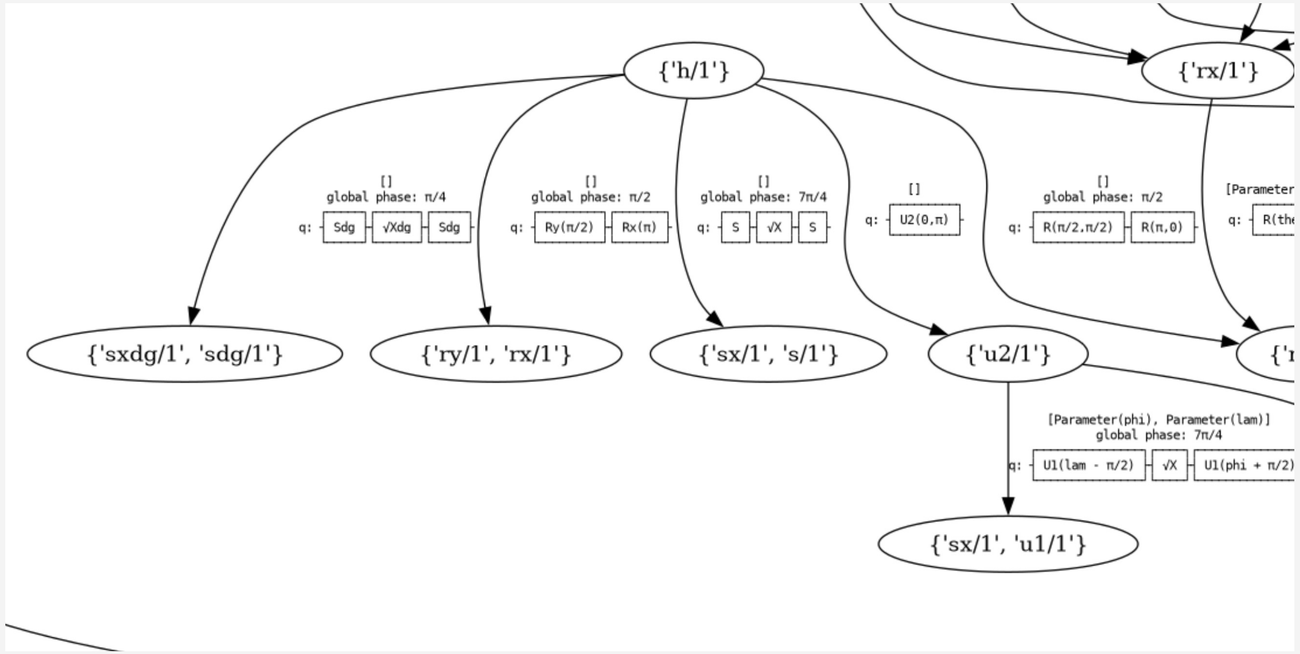


Basis Translator

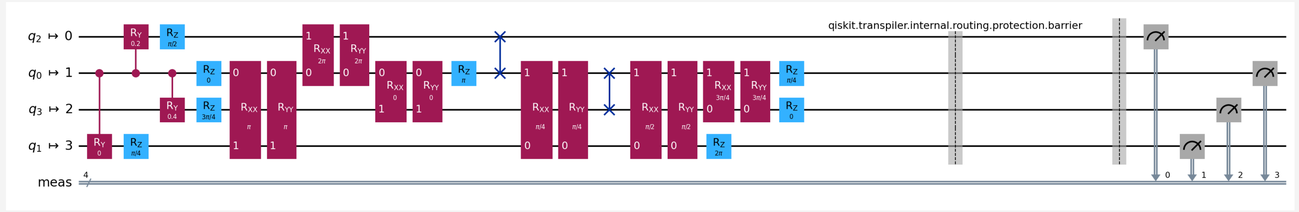
- The basis translator uses an equivalence library that builds a graph of equivalent circuits to a gate
- Dijkstra's algorithm is used to find a path from a given gate to gates in the target
- The gates in the circuits are recursively substituted along the path to build an equivalent circuit that conforms to the target
- Each gate is then replaced in the DAG by that equivalent circuit

Basis Translator

- The basis translator uses an equivalence library that builds a graph of equivalent circuits to a gate
- Dijkstra's algorithm is used to find a path from a given gate to gates in the target
- The gates in the circuits are recursively substituted along the path to build an equivalent circuit that conforms to the target
- Each gate is then replaced in the DAG by that equivalent circuit

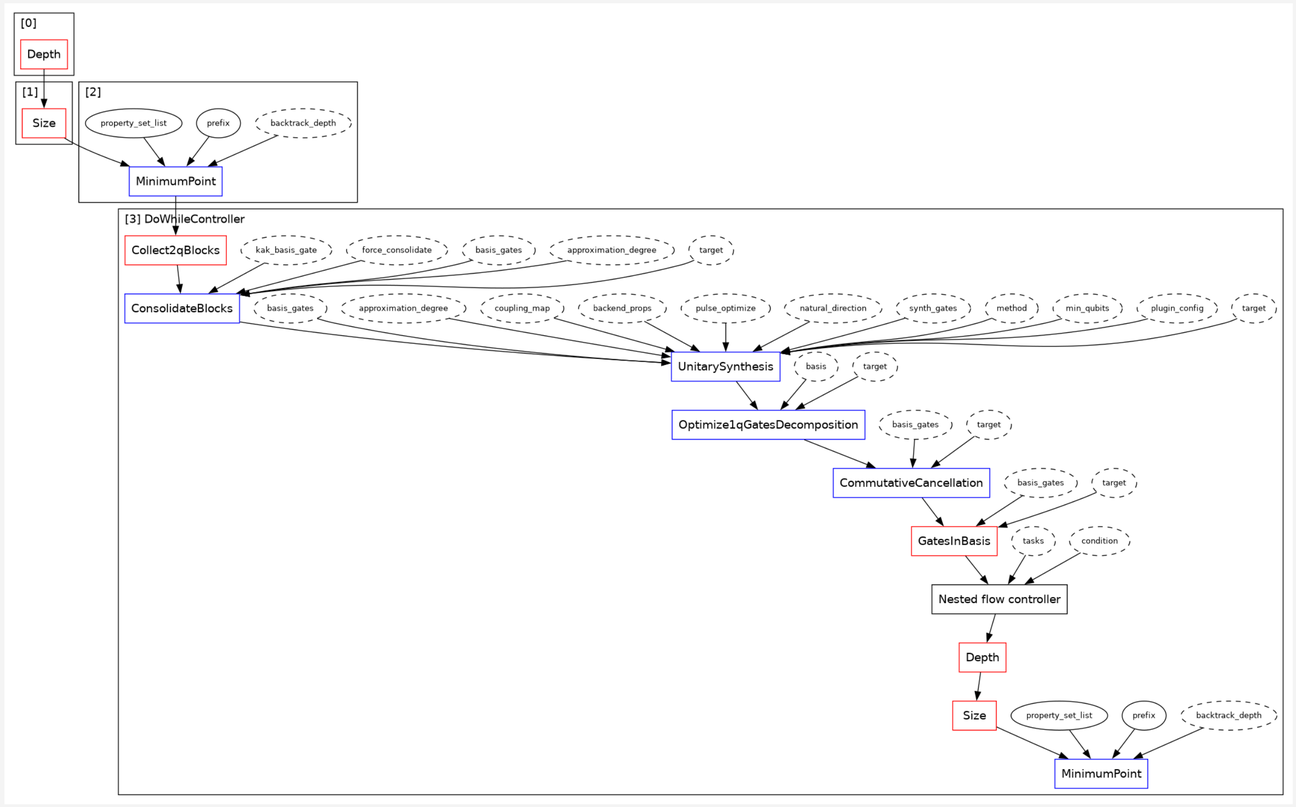


Basis Translator



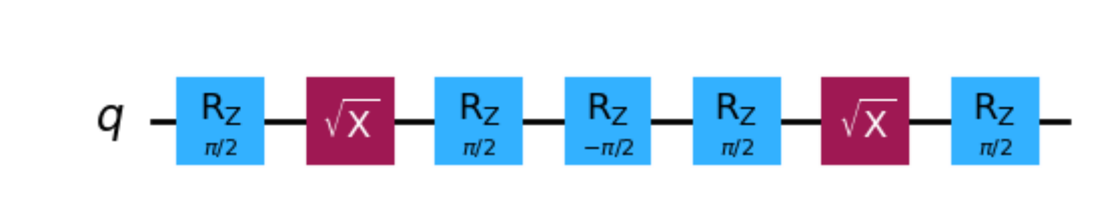
Optimization stage

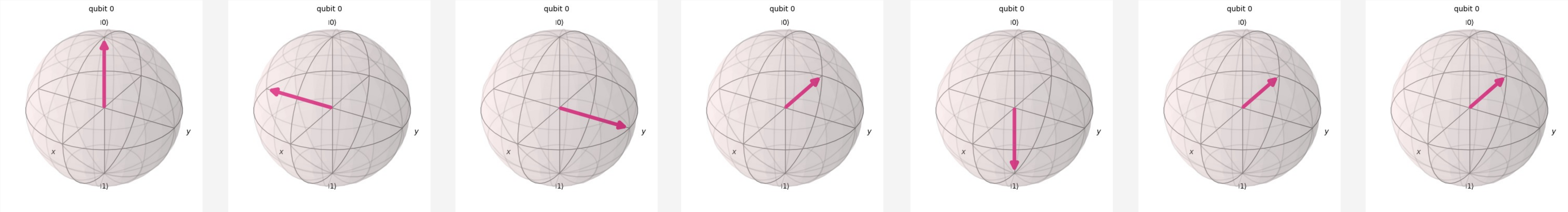
- This stage takes the mapped circuit and optimizes it to reduce unnecessary operations
- This stage runs in a loop to repeatedly run the optimization passes until a minimum point in depth and size is found



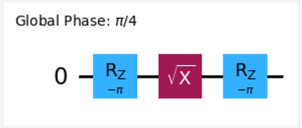
1 Qubit Unitary Peephole Optimization

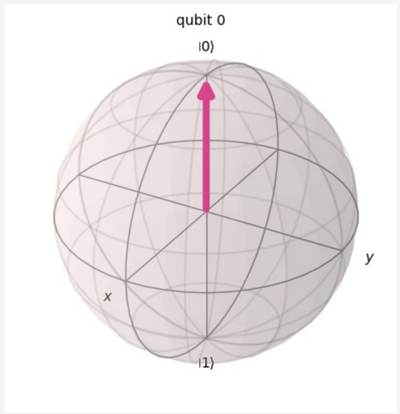
– This pass looks for sequences of single qubit gates

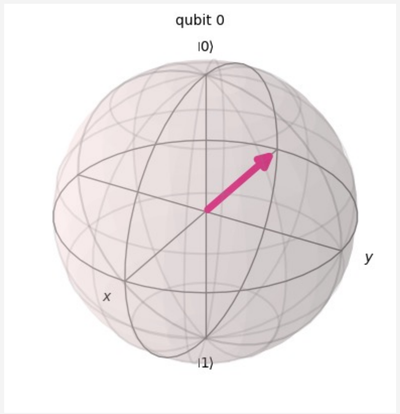




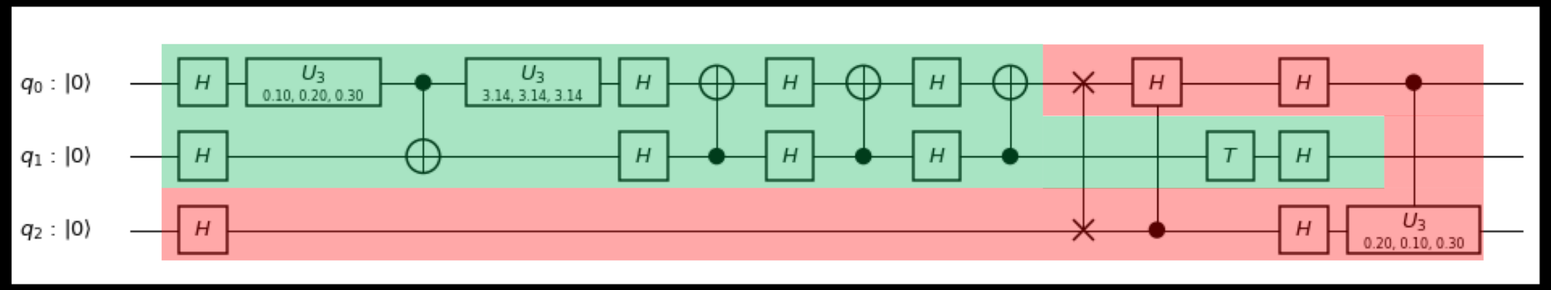
1 Qubit Unitary Peephole Optimization



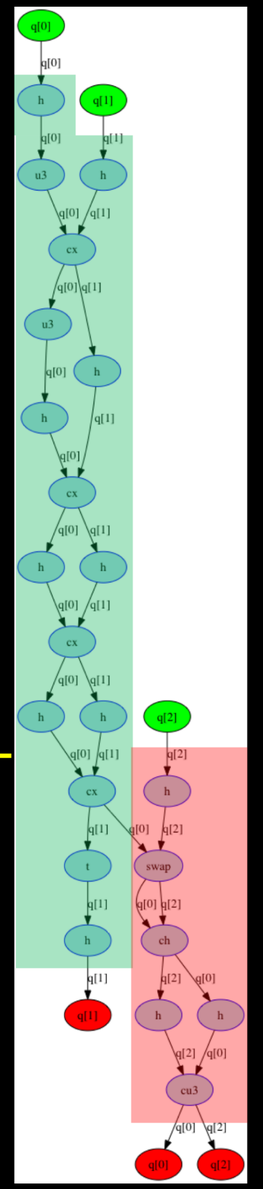




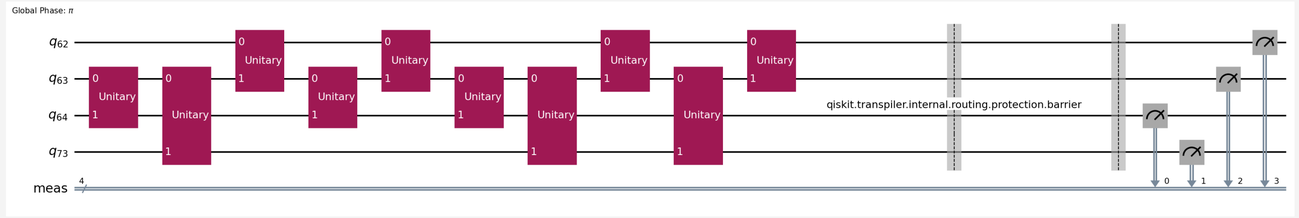
2-qubit Block Collection

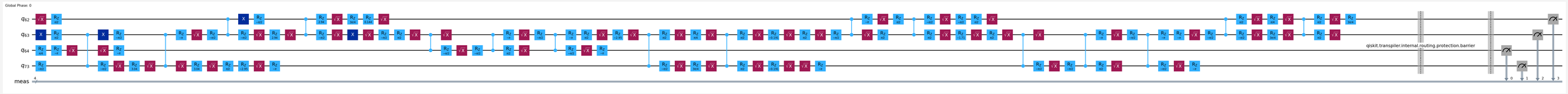


Efficient graph traversal on the DAG.
For each CNOT, collect ancestors and predecessors until a branch.



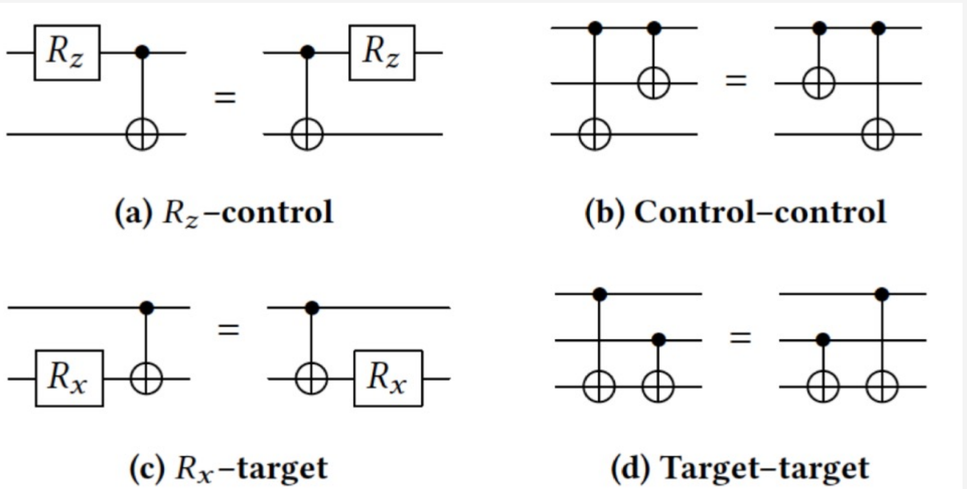
2 Qubit Unitary Peephole Optimization



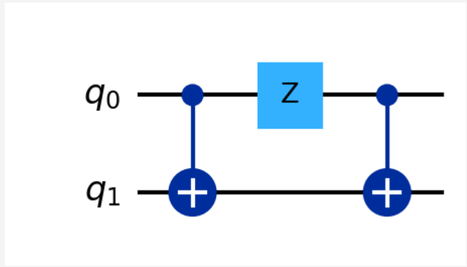


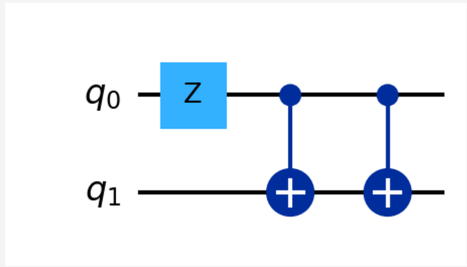
Commutative Cancellation

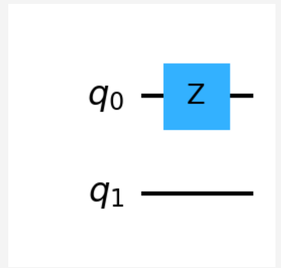
Examples of commuting gates:



Two consecutive gates are commutative ⇔
They can be exchanged without changing what they compute.

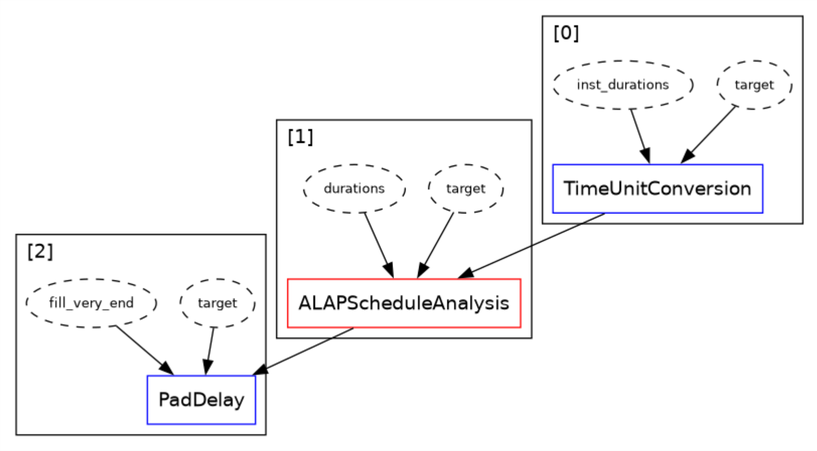


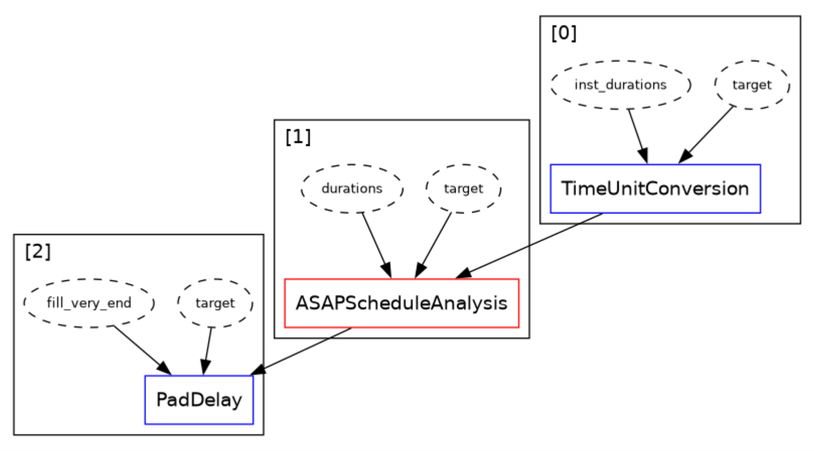




Scheduling stage

- This stage takes the mapped and optimized circuit and analyzes the runtime of the operations in the circuit and identify idle periods on each qubit
- It will then insert explicit operations to account for all the time in the circuit





Scheduling stage

- This stage takes the mapped and optimized circuit and analyzes the runtime of the operations in the circuit and identify idle periods on each qubit
- It will then insert explicit operations to account for all the time in the circuit