# A Verified Optimizer for Quantum Circuits





**Kesha Hietala**
University of Maryland
kesha@cs.umd.edu

Robert Rand
University of Chicago
rand@uchicago.edu

Shih-Han Hung
University of Maryland
shung@umd.edu

Xiaodi Wu
University of Maryland
xwu@cs.umd.edu

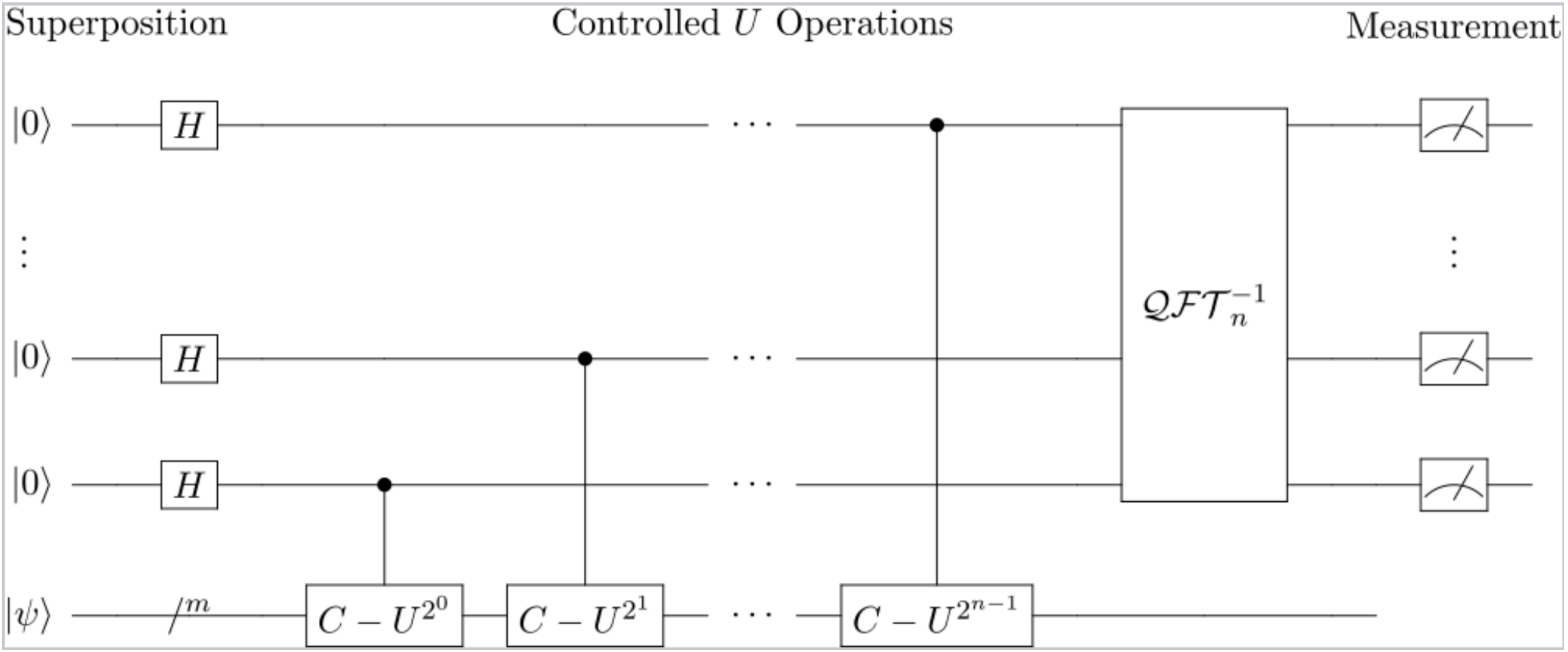
Michael Hicks
University of Maryland
mwh@cs.umd.edu

POPL 2021


Image from https://www.ibm.com/quantum-computing/

# Writing Quantum Programs is Hard

- Quantum indeterminacy ⇒ quantum programs are **probabilistic**
- Quantum programs are often written as **circuits**

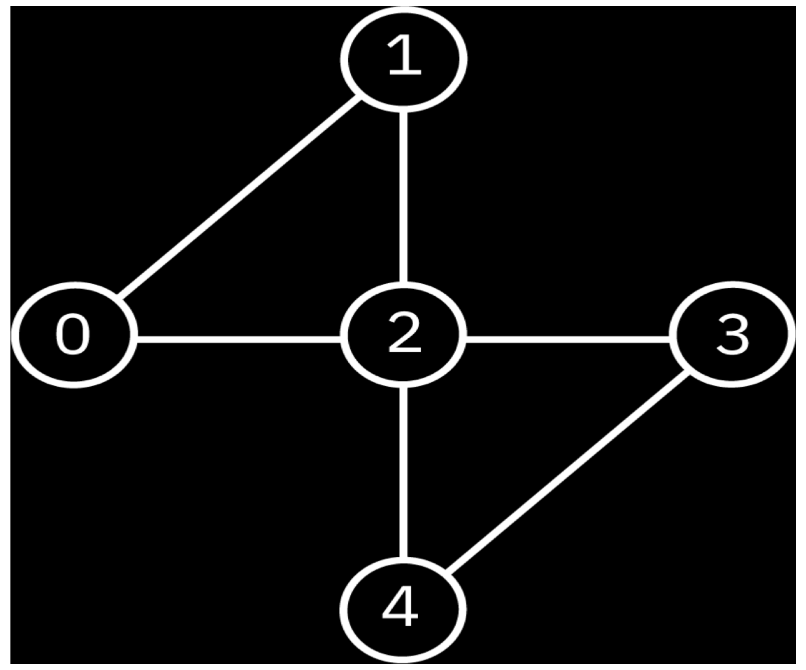



- Quantum programs use **new primitives**
    - E.g. "prepare a uniform superposition", "perform a Fourier transform"

Image from https://en.wikipedia.org/wiki/Quantum_phase_estimation_algorithm

# Quantum Machines are Limited

- Machines today have a **few, unreliable qubits**
  - Typically 15-50 qubits in total
  - In the near future, we can expect machines with a few hundred qubits, able to run up to 1000 two-qubit gates
- They also have **hardware-specific constraints**
  - Limited set of available operations
  - Only allow two-qubit gates between certain pairs of qubits




*Noisy, Intermediate Scale Quantum (NISQ)* Computing -Preskill



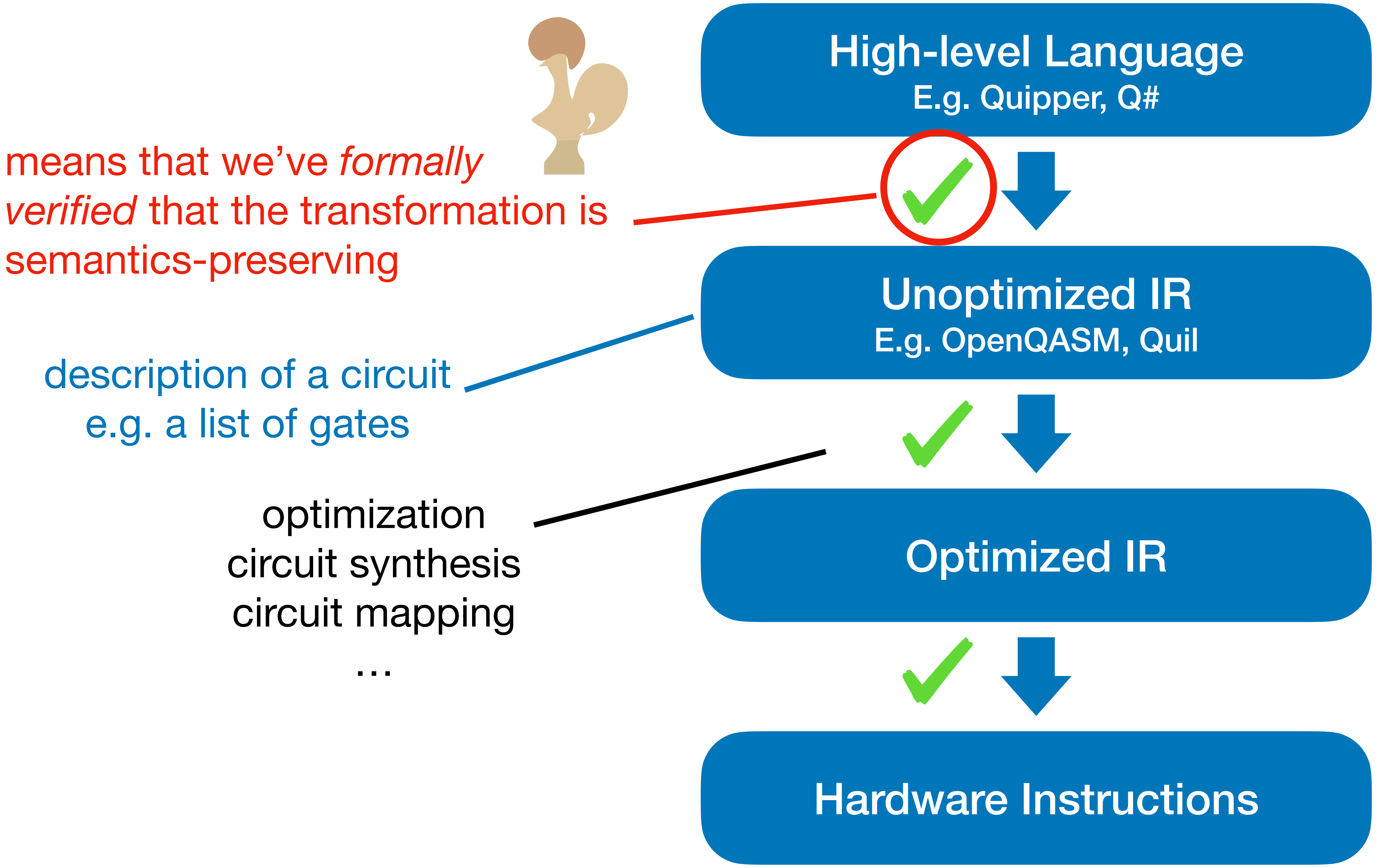
Image from https://github.com/Qiskit/ibmq-device-information

# Quantum Compilers are Complicated

- Quantum compilers need to perform **sophisticated transformations** to account for limited resources, hardware constraints
- These transformation are hard to write… and harder to debug
  - Is an unexpected result due to a program bug? machine error? quantum indeterminacy?

# Verified Compiler Stack

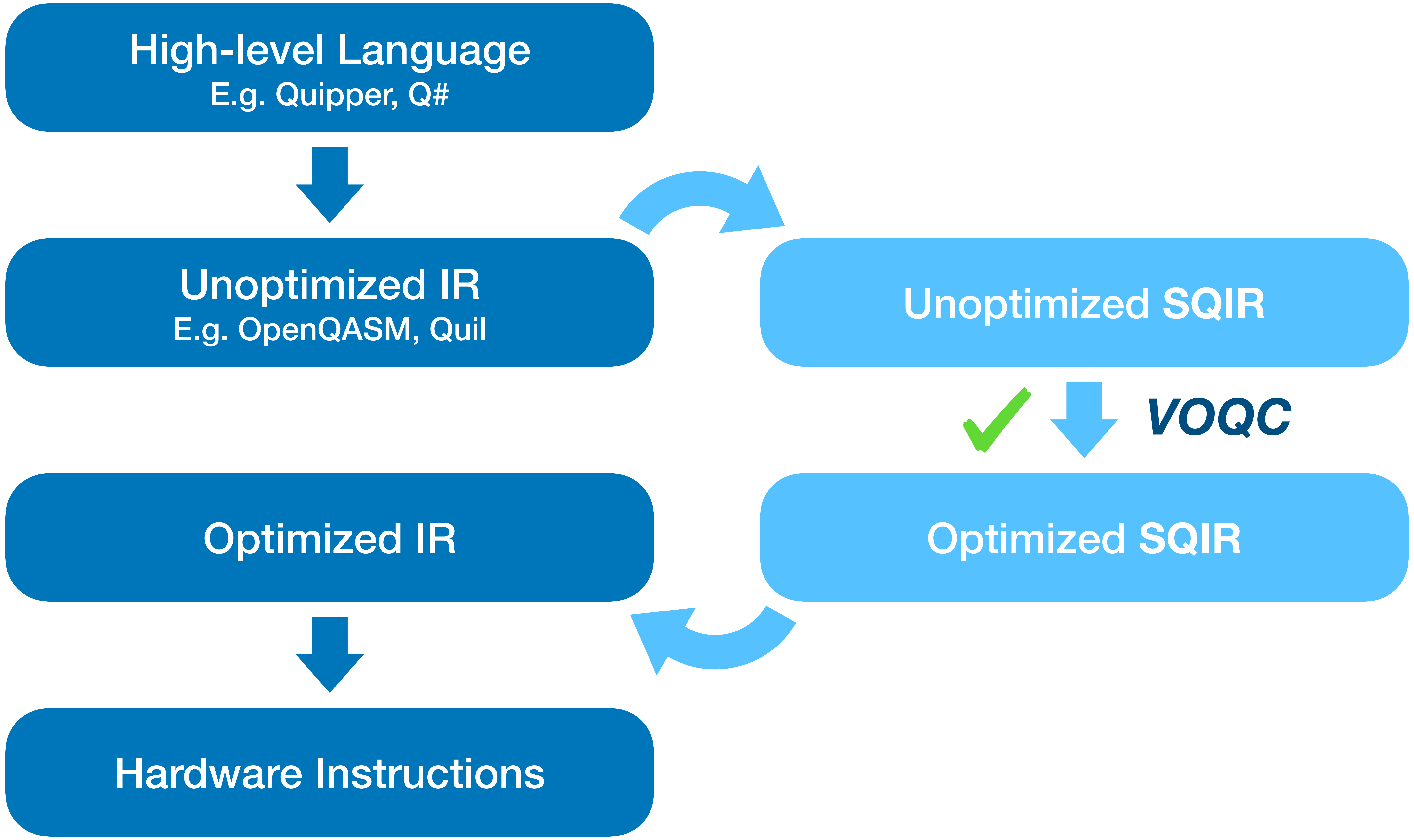
- End goal: *verified compiler stack* for quantum programs




- Challenge: The semantics of **quantum programs is very different** from classical programs
  - States represented as matrices of complex numbers
  - Programs involve probabilities, trigonometry
- Requires development of **new frameworks, libraries, and automation**

# SQIR and VOQC

- Our paper: **VOQC**, a *Verified Optimizer for Quantum Circuits*, which is built on top of **SQIR**, a *Small Quantum Intermediate Representation* designed for proof



# SQIR and VOQC

- SQIR and VOQC are implemented in around 11k lines of Coq code
  - 3.5k for core SQIR, source program proofs
  - 7.5k for VOQC libraries, optimizations, circuit mapper
  - We extend QWIRE's matrix & complex number libraries by 3k lines
- Long version of the paper available at https://arxiv.org/abs/1912.02250
- Code available at https://github.com/inQWIRE/SQIR
- Artifact available at https://zenodo.org/record/4268896

# Outline

- Intro to Quantum Programming
- SQIR
- VOQC
- Future Work

# Qubits

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$|0\rangle \qquad\qquad\qquad |1\rangle$$

$$|\alpha|^2 + |\beta|^2 = 1$$

*Superposition*: Qubits can be in multiple states (0 or 1) at once

# Measurement

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

$|\alpha|^2$ $\qquad$ $|\beta|^2$

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$|0\rangle$ $\qquad$ $|1\rangle$

*Measurement*: Looking at a qubit probabilistically turns it into a bit.

# Measurement

$|+\rangle$ state

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$\left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2} \qquad \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$$

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$|0\rangle \qquad |1\rangle$$

*Measurement*: Looking at a qubit probabilistically turns it into a bit.

# Operators

A unitary operator transforms, or *evolves*, a state

$$H \; |0\rangle = |+\rangle$$

$$H \; |+\rangle = |0\rangle$$

This is the *Hadamard* operator, H

(which is its own inverse)

# Operators

Operators are represented as *unitary matrixes*

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix} = \begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{pmatrix}$$

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{pmatrix} = \begin{pmatrix}1\\ 0\end{pmatrix}$$

# Multiple Qubits

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}$$

$$|+\rangle \otimes |0\rangle \quad = \quad |+\rangle\,|0\rangle \quad \text{or} \quad |+0\rangle$$

*Multi-qubit states* are constructed via the tensor product

# Measurement 2.0

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$\xrightarrow{\frac{1}{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \xrightarrow{\frac{1}{2}} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

# Measurement 2.0

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

$\frac{1}{2}$ $\qquad$ $\frac{1}{2}$

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$|00\rangle \qquad |11\rangle$$

# Entanglement

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} \qquad\qquad \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad\qquad ? \otimes ?$$

*Entangled* qubits are not probabilistically independent — they cannot be decomposed. Connection at a distance!

# Multi-Qubit Unitaries

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$\text{CNOT}\, |+0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

# More Unitaries

A *universal sets* of unitaries can be used to approximate any unitary operator using a finite sequence of gates

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$ **bit flip**

$$|0\rangle \mapsto |1\rangle$$
$$|1\rangle \mapsto |0\rangle$$

$$Rz(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$ **phase shift**

$$|0\rangle \mapsto |0\rangle$$
$$|1\rangle \mapsto e^{i\theta}|1\rangle$$

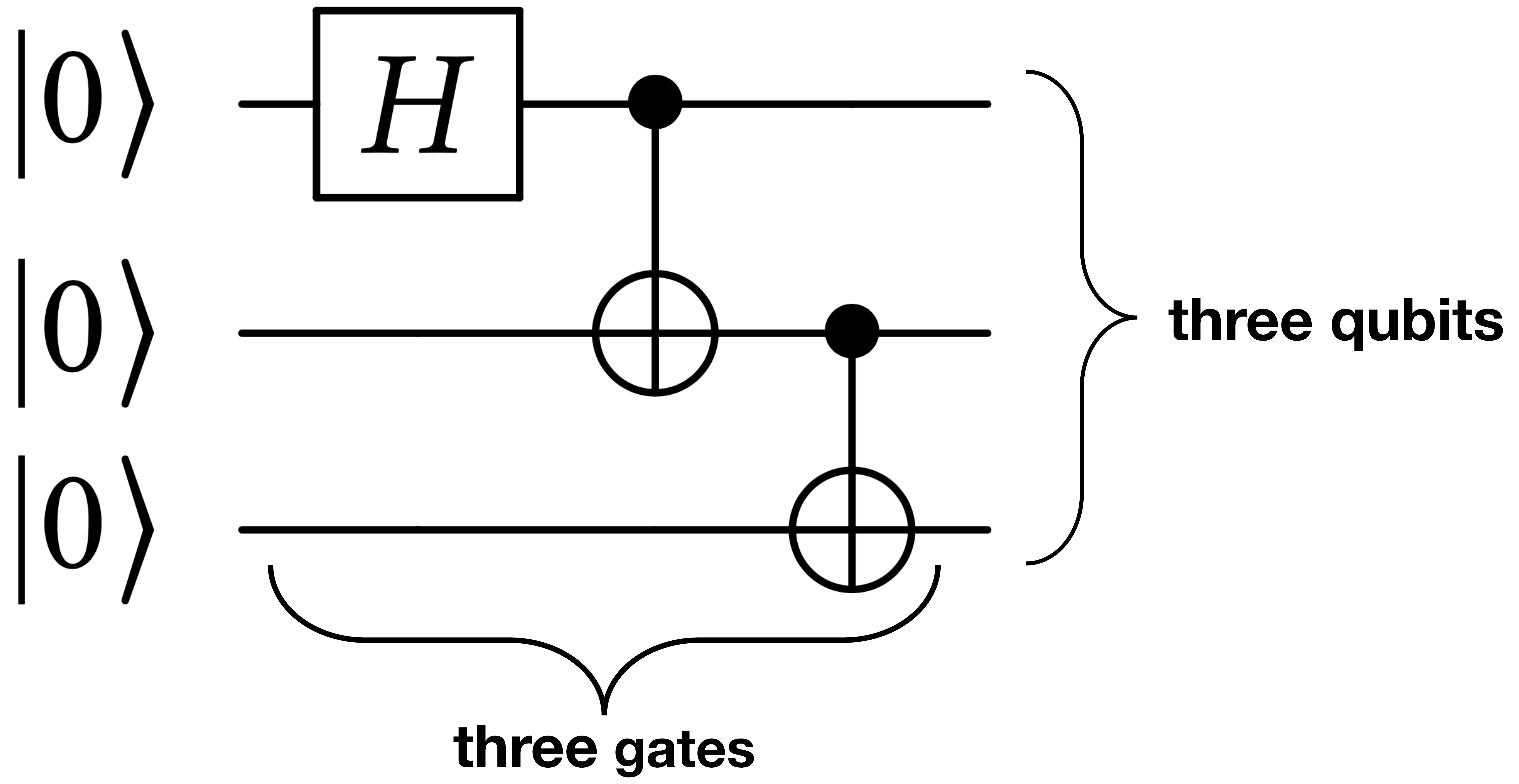
# General Quantum States

- So far we have seen *pure states*
  - *E.g.* $|0\rangle, |1\rangle, |+\rangle$
- A *mixed state* is a (classical) probability distribution over pure states
  - E.g. $\begin{cases} |0\rangle & \text{with probability 1/2} \\ |1\rangle & \text{with probability 1/2} \end{cases}$
- Density matrices allow us to describe both pure and mixed states

$$\rho = |0\rangle\langle 0| = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \qquad \rho = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| = \begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \end{pmatrix}$$

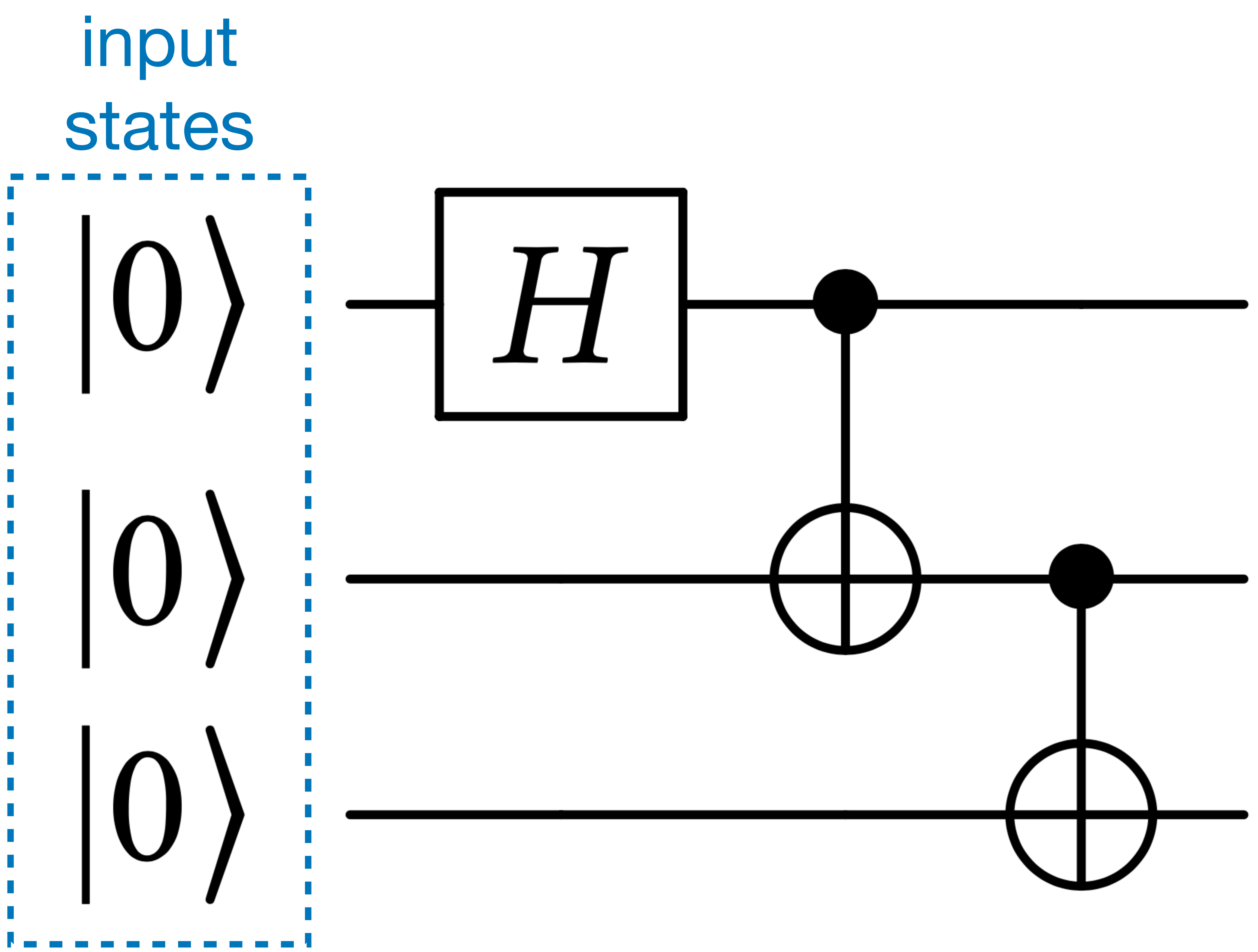
# Circuits

Quantum programs are often written as circuits



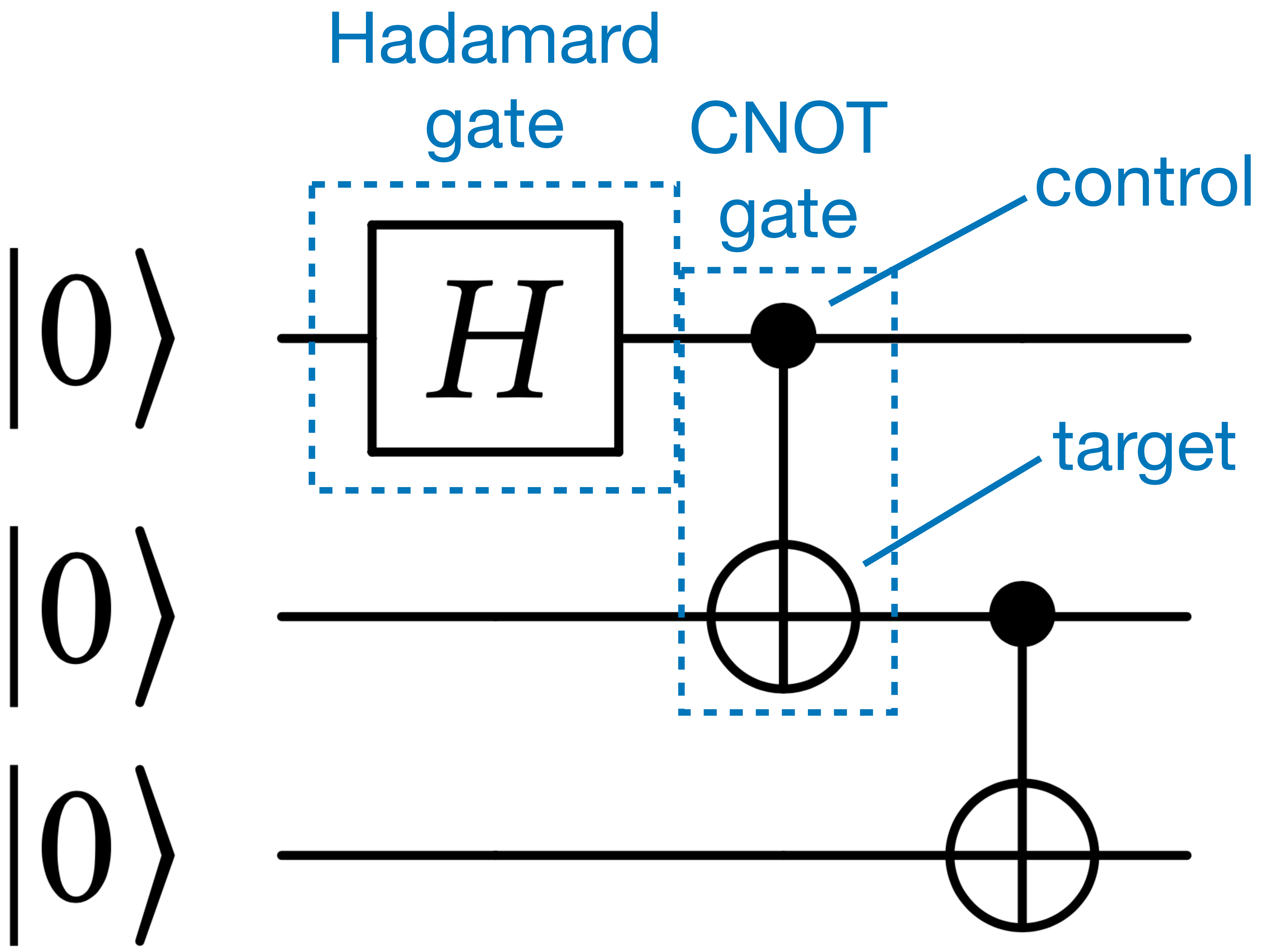
# Circuits

Quantum programs are often written as circuits

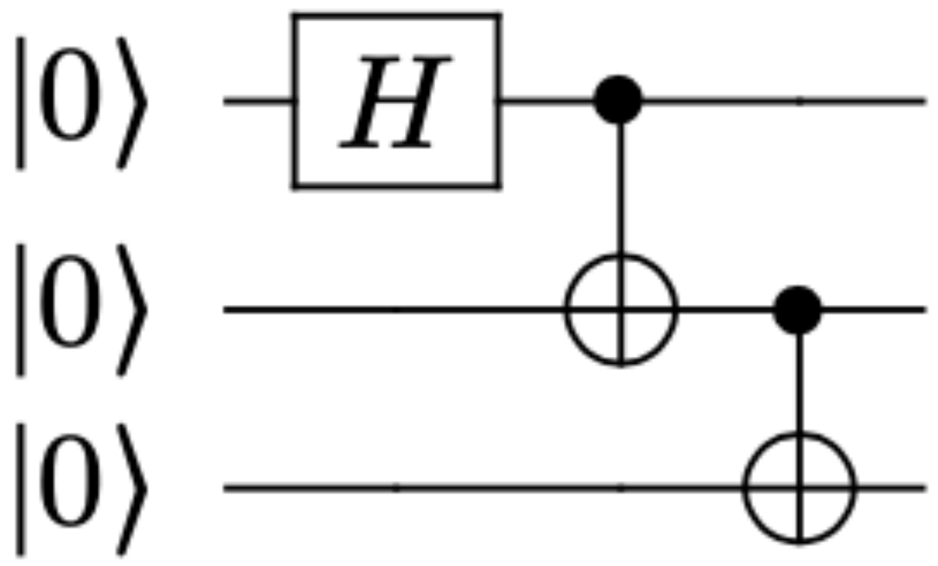


# Circuits

Quantum programs are often written as circuits



# Quantum Programming




**Circuit**

```
H 0
CNOT 0 1
CNOT 1 2
```

**Quil (Rigetti)**

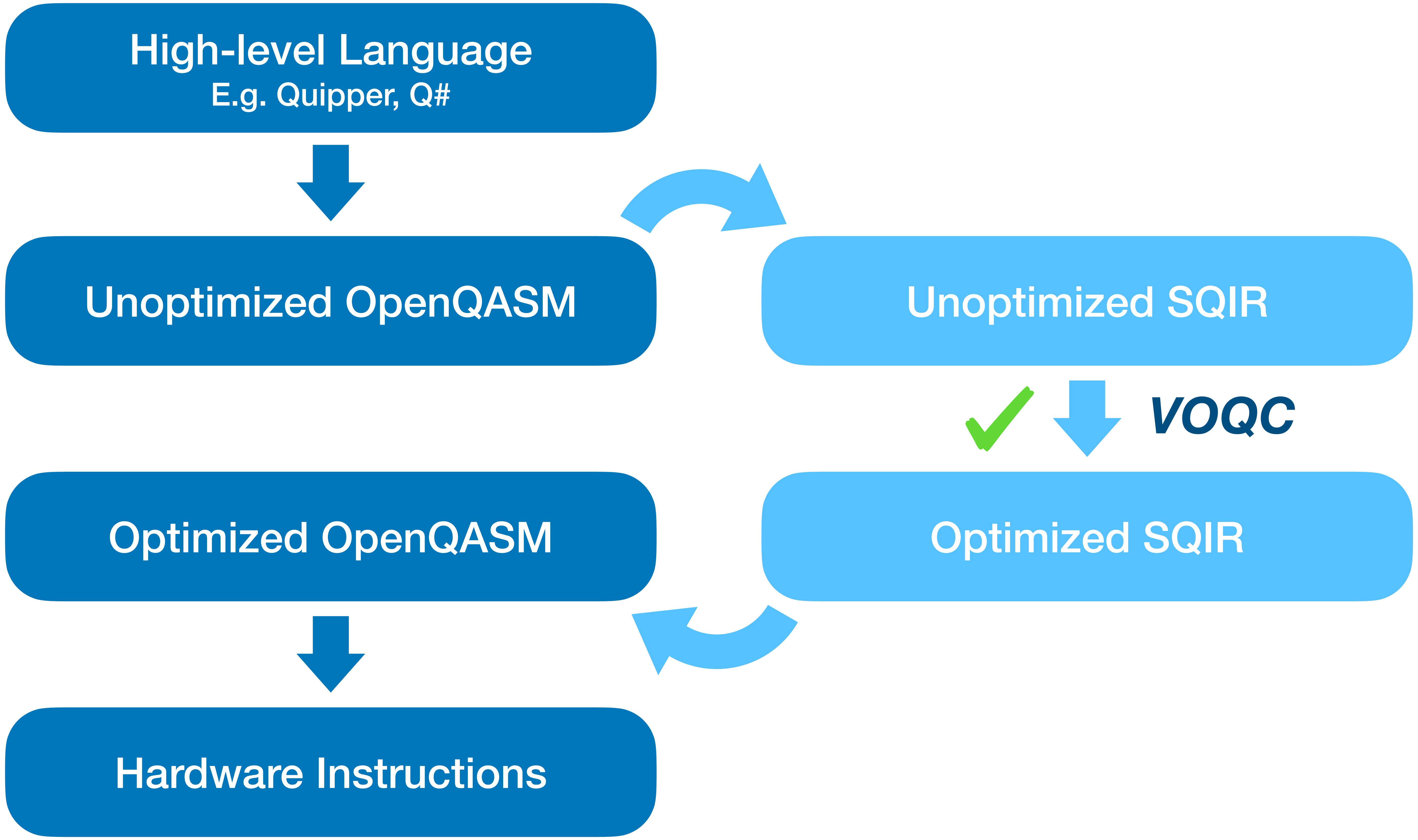
```python
def ghz_state(qubits):
  program = Program()
  program += H(qubits[0])
  for q1,q2 in zip(qubits, qubits[1:]):
    program += CNOT(q1, q2)
  return program
```

**PyQuil (Rigetti)**

Many “high-level” quantum programming languages (e.g. PyQuil, Cirq, Qiskit, Quipper, QWIRE) are libraries for constructing circuits

# SQIR: Small Quantum Intermediate Representation

- SQIR programs, embedded in Coq, are assigned a *denotational semantics* of matrices
- Two variations of SQIR
  - *Unitary SQIR*: No measurement
  - *Full SQIR*: Adds branching measurement operator



# Unitary SQIR

- Semantics parameterized by *gate set G* and *dimension d of a global register*

$$U \ ::= \ U_1;\ U_2 \mid G\ q \mid G\ q_1\ q_2$$

**E.g.** $apply_1(X, q, d) = I_{2^q} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes I_{2^{(d-q-1)}}$

- The denotation (semantics) of *U* is a $2^d \times 2^d$ unitary matrix

$$[\![U_1;\ U_2]\!]_d = [\![U_2]\!]_d \times [\![U_1]\!]_d$$

$$[\![G_1\ q]\!]_d = \begin{cases} apply_1(G_1,\ q,\ d) & \text{well-typed} \\ 0_{2^d} & \text{otherwise} \end{cases}$$

$q < d$

$$[\![G_2\ q_1\ q_2]\!]_d = \begin{cases} apply_2(G_2,\ q_1,\ q_2,\ d) & \text{well-typed} \\ 0_{2^d} & \text{otherwise} \end{cases}$$

$q_1 < d \ \wedge \ q_2 < d \ \wedge \ q_1 \neq q_2$

# Non-Unitary SQIR

- Semantics parameterized by *gate set G* and *dimension d of a global register*

$$P \ ::= \ \texttt{skip} \mid P_1;\ P_2 \mid U \mid \texttt{meas}\ q\ P_1\ P_2$$

- The denotation of *P* is a function over $2^d \times 2^d$ density matrices

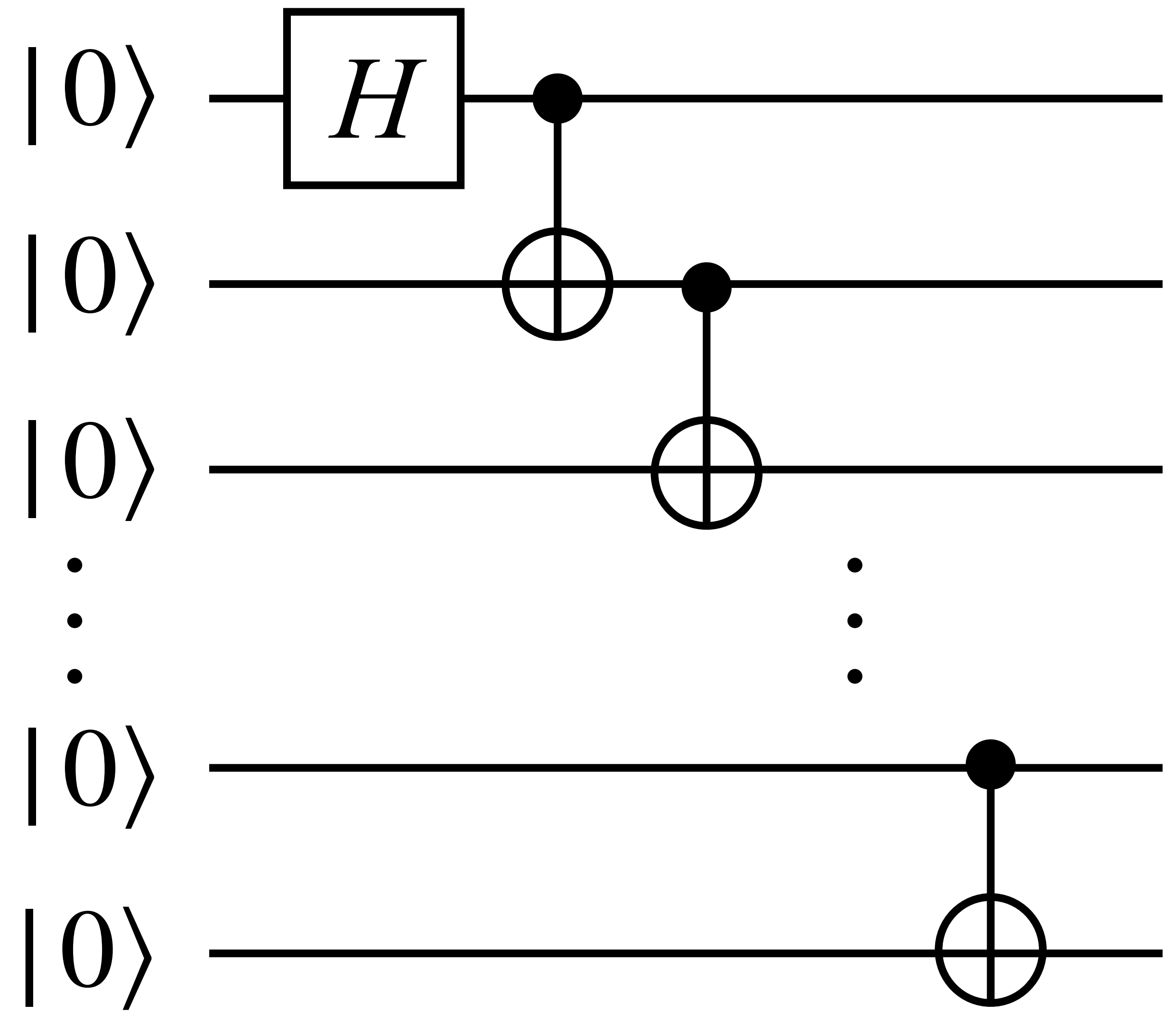
$$\{\!|\mathsf{skip}|\!\}_d(\rho) = \rho$$
$$\{\!|P_1;\ P_2|\!\}_d(\rho) = (\{\!|P_2|\!\}_d \circ \{\!|P_1|\!\}_d)(\rho)$$
$$\{\!|U|\!\}_d(\rho) = [\![U]\!]_d \times \rho \times [\![U]\!]_d^\dagger$$
$$\{\!|\mathsf{meas}\ q\ P_1\ P_2|\!\}_d(\rho) = \{\!|P_2|\!\}_d(|0\rangle_q\langle 0| \times \rho \times |0\rangle_q\langle 0|) + \{\!|P_1|\!\}_d(|1\rangle_q\langle 1| \times \rho \times |1\rangle_q\langle 1|)$$

Standard semantics; also used in QHL[1] and QWIRE[2]

[1] Ying. *Floyd-Hoare logic for quantum programs*. TOPLAS 2012.
[2] Paykin et al. *QWIRE: A core language for quantum circuits*. POPL 2017.

# SQIR Metaprogramming

- SQIR programs just express circuits. We can express parameterized circuit families using Coq as a meta programming language




```
Fixpoint ghz (n : ℕ) : ucom base n :=
  match n with
  | 0 ⇒ SKIP
  | 1 ⇒ H 0
  | S n' ⇒ ghz n'; CNOT (n'-1) n'
  end.
```

- The `ghz` Coq function returns a SQIR program (of type `ucom base n`) whose semantics is the `n`-qubit GHZ state

# Proofs of Correctness in Coq

- We might like to prove that evaluating `ghz n` on $|0\rangle^{\otimes n}$ produces $|GHZ^n\rangle$
  - where $|GHZ^n\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes n} + |1\rangle^{\otimes n})$

```
Definition GHZ (n : ℕ) : Vector (2 ^ n) :=
  match n with
  | 0    ⇒ I 1
  | S n' ⇒ 1/√2 * |0⟩^⊗n + 1/√2 * |1⟩^⊗n
  end.

Lemma ghz_correct : ∀ n : ℕ,
  n > 0 → [[ghz n]]_n × |0⟩^⊗n = GHZ n.
Proof.
...
Qed.
```

# Designed for Proof

- SQIR was conceived as a simplified version of QWIRE[1]; we use QWIRE's libraries for matrices and complex numbers

- SQIR proofs are simpler that QWIRE's because we:
  1. Reference **qubits using concrete indices** (`CNOT 2 1` vs. `CNOT x y`)
     - Easy to map gate arguments to the right column/row in the matrix
     - Disjointness is *syntactic*; important for proving equivalences
  2. **Separate the unitary core from the full language** with measurement
     - Unitary matrix semantics simpler than *density matrix* formulation
  3. Assign a **denotation of the zero-matrix to ill-typed programs**
     - E.g., `CNOT 1 1`, which violates no-cloning

[1]Paykin, Rand and Zdancewic. *QWIRE: A core language for quantum circuits*. POPL 2017.

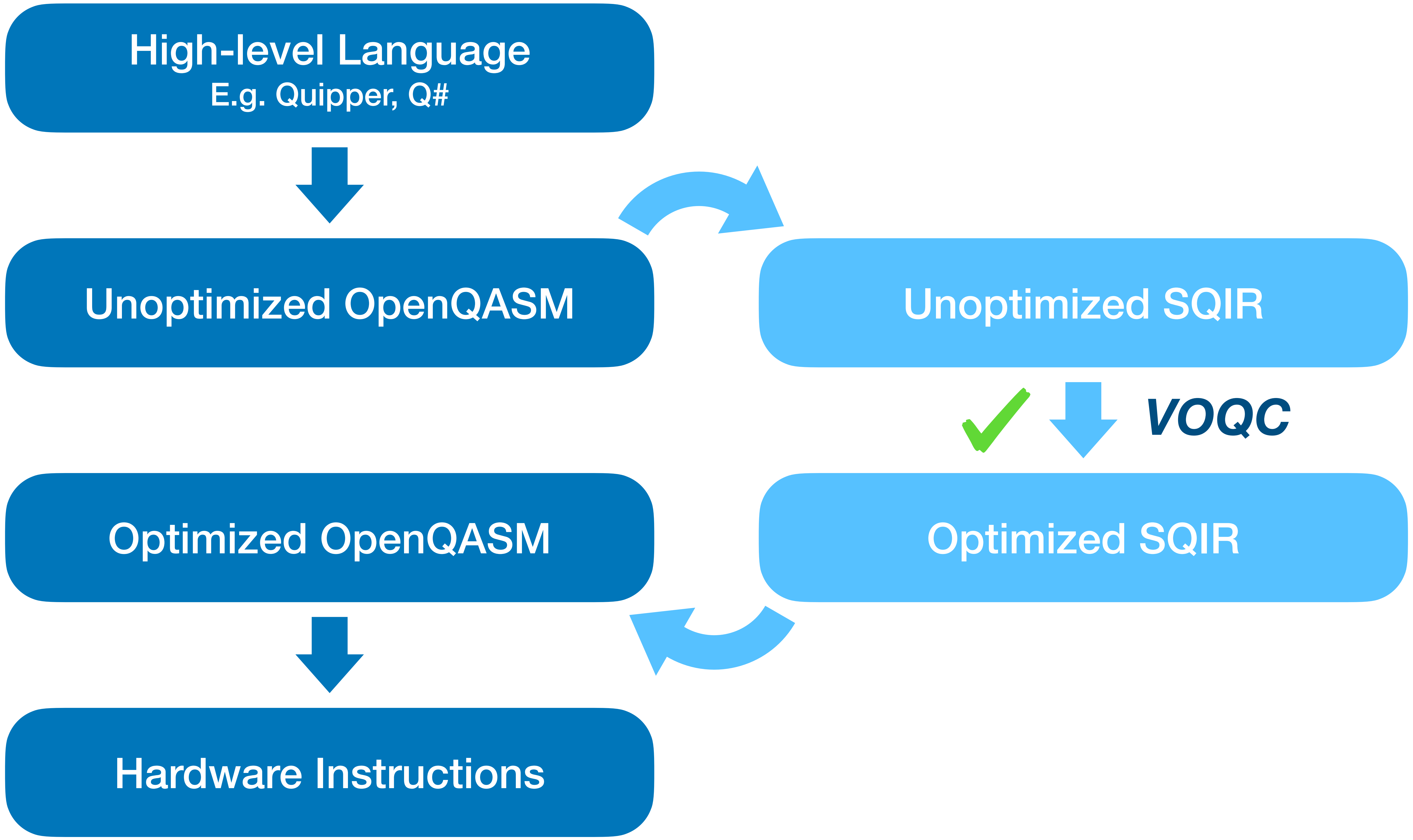
# Proofs so Far

- We have formally verified several source programs correct
  - Quantum teleportation / superdense coding
  - GHZ state preparation
  - Deutsch-Jozsa algorithm
  - Simon's algorithm
  - Grover's search algorithm
  - Quantum phase estimation (key part of Shor's algorithm)
- These proofs as well as the basic support of SQIR (lemmas, tactics, etc.) constitute about 3500 lines of Coq code
- For more details see https://arxiv.org/abs/2010.01240

# VOQC: A Verified Optimizer for Quantum Circuits

- Transformations are represented as Coq functions over SQIR circuits
  - Extracted to executable OCaml code

- We prove (verify) that transformations are *semantics-preserving*
  - Can also prove that the output program respects machine constraints



# VOQC in Sum

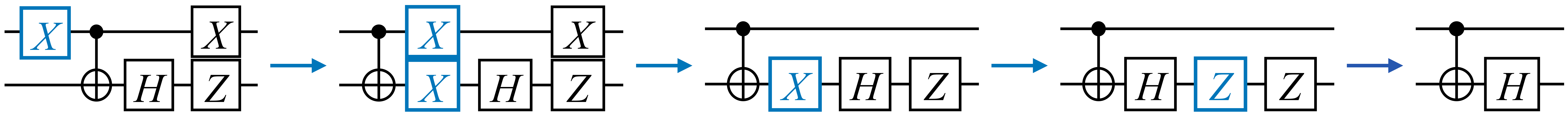
- Most of VOQC (2200 LOC) consists of verified implementations of optimizations developed by Nam et al.[1]
  - Replacement (peephole optimizations)
  - Propagation (commutation) and cancellation
  - Rotation merging (non-local coalescing)

- Some optimizations for non-unitary programs, inspired by Qiskit (800 LOC)
  - Remove $z$-rotations before measurement
  - Classical state propagation

- Another 2100 LOC for program manipulation; 2100 more for circuit mapping

[1]Nam, Ross, Su, Childs and Maslov. *Automated Optimization of Large Quantum Circuits with Continuous Parameters*. npj 2018.

# Example: X Propagation




- Based on Nam et al[1] “not propagation”
- We verify **semantics-preservation**
  - At each step, the denotation of the program (i.e. unitary matrix) does not change
- We prove this via induction on the structure of the input program
  - ~30 lines to implement optimization
  - ~270 lines to prove semantics-preservation

[1]Nam, Ross, Su, Childs and Maslov. *Automated Optimization of Large Quantum Circuits with Continuous Parameters*. npj 2018.

# Verifying Matrix Equivalences

- Many proofs use unitary equivalences; e.g., X propagation's proof uses:
  - X gates cancel: $X\ m; X\ m \equiv I\ m$
  - X commutes with CNOT control: $X\ m; CNOT\ m\ n \equiv CNOT\ m\ n; X\ m; X\ n$
  - X commutes with CNOT target: $X\ n; CNOT\ m\ n \equiv CNOT\ m\ n; X\ n$
  - H transforms X to Z: $X\ m; H\ m \equiv H\ m; Z\ m$

- We prove these as lemmas
  - Doing so is tedious, so we developed Coq tactics to convert matrix expressions into a *grid normal form* to facilitate automation

# Grid Normal Form

- Consider the equivalence $X\ n; CNOT\ m\ n \equiv CNOT\ m\ n; X\ n$
- Per our semantics, this requires proving

$$apply_1(X, n, d)\ \times\ apply_2(CNOT, m, n, d) = apply_2(CNOT, m, n, d)\ \times\ apply_1(X, n, d)$$

  - where

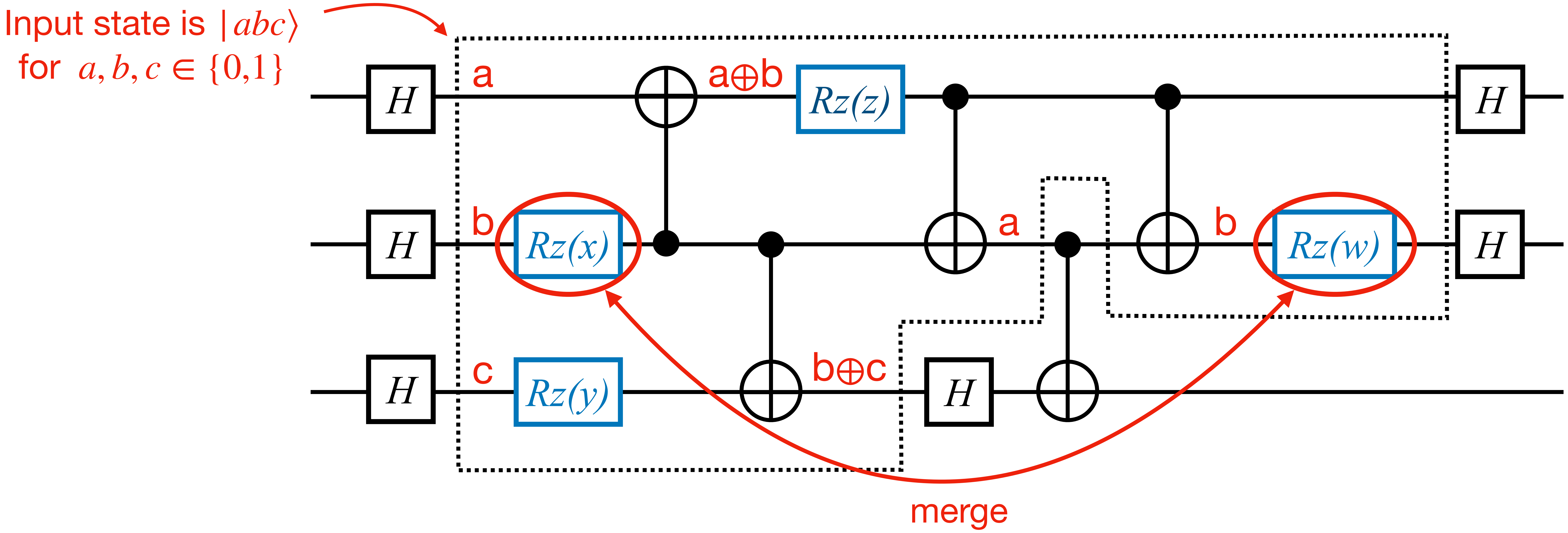
$$apply_1(X, n, d) = I_{2^n} \otimes \sigma_x \otimes I_{2^q}$$

$$apply_2(CNOT, m, n, d) = I_{2^m} \otimes |1\rangle\langle 1| \otimes I_{2^p} \otimes \sigma_x \otimes I_{2^q}\ + I_{2^m} \otimes |0\rangle\langle 0| \otimes I_{2^p} \otimes I_2 \otimes I_{2^q}$$

- Our automation reduces both sides of the equality to *grid normal form*

$$I_{2^m} \otimes |1\rangle\langle 1| \otimes I_{2^p} \otimes I_2 \otimes I_{2^q} + I_{2^m} \otimes |0\rangle\langle 0| \otimes I_{2^p} \otimes \sigma_x \otimes I_{2^q}$$

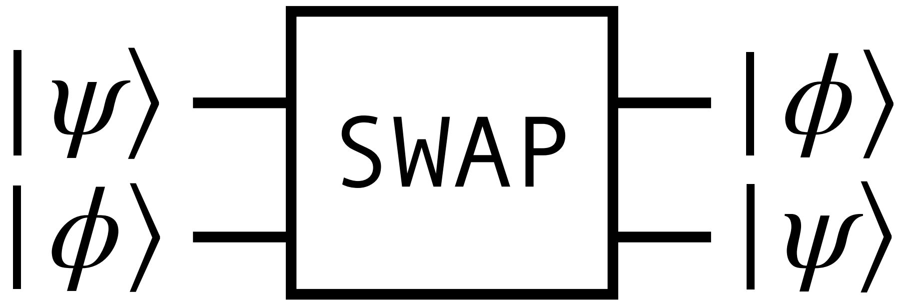
# More Interesting: Rotation Merging




- Based on Nam et al rotation merging
- Combines *Rz* gates in arbitrary {*Rz*, *CNOT*} sub-circuits
  - ~100 lines to implement optimization
  - ~920 lines to prove semantics-preservation

# Also: Circuit Mapping

- Given an input program & description of machine connectivity, *circuit mapping* produces a program that satisfies connectivity constraints
  - Usually uses `SWAP` gates to“move” qubits by exchanging their values




  - E.g `CNOT 0 2` ,   → `SWAP 0 1; CNOT 1 2`

- We prove that the output program is equivalent to the original, up to permutation of indices
  - Above, $[\![\texttt{CNOT 0 2}]\!]_3 = P \times [\![\texttt{SWAP 0 1; CNOT 1 2}]\!]_3$ where *P* implements the permutation $\{0 \rightarrow 1,\ \ 1 \rightarrow 0,\ \ 2 \rightarrow 2\}$

# Evaluation

1 https://qiskit.org/
2 https://cqcl.github.io/pytket/build/html/index.html
3 https://arxiv.org/pdf/1710.07345.pdf
4 https://arxiv.org/pdf/1303.2042.pdf
5 https://github.com/Quantomatic/pyzx

- Is VOQC any good? Maybe we just verified simple optimizations
- So: Compared our *verified* optimizer against existing *unverified* optimizers
  - IBM Qiskit Terra v0.15.12[1]
  - Cambridge CQC tket v0.6.0[2]
  - Nam et al,[3] both L and H levels (used by IonQ)
  - Amy et al[4]
  - PyZX v0.6.0[5]

# Benchmark

- Used **benchmark suite of Amy et al**[1]
  - 28 programs: Arithmetic circuits, implementations of multiple-control Toffoli gates, and Galois field multiplier circuits
  - Ranging from 45 to 13,593 gates and 5 to 96 qubits
  - Uses the Clifford+T gate set (CNOT, H, S and T)
- We measured **effectiveness in terms of gate reductions**
  - Both T gate and total
- Measured optimization time (not parsing or printing)

[1]Amy, Maslov and Mosca. *Polynomial-Time T-Depth Optimization of Clifford+T Circuits Via Matroid Partitioning.* TCAD 2013.

# Results

1 https://qiskit.org/
2 https://cqcl.github.io/pytket/build/html/index.html
3 https://arxiv.org/pdf/1710.07345.pdf
4 https://arxiv.org/pdf/1303.2042.pdf
5 https://github.com/Quantomatic/pyzx

| **Geo. mean compilation times** | | | | | | |
|---|---|---|---|---|---|---|
| **Qiskit**[1] | **tket**[2] | **Nam**[3] **(L)** | **Nam (H)** | **Amy**[4] | **PyZX**[5] | **VOQC** ✓ |
| 0.812s | 0.129s | 0.002s | 0.018s | 0.007s | 0.384s | 0.013s |

**VOQC is the same ballpark**

| **Geo. mean reduction in gate count** | | | |
|---|---|---|---|
| **Qiskit** | **tket** | **Nam (H)** | **VOQC** ✓ |
| 10.1% | 10.6% | 24.8% | 17.8% |

**VOQC only outperformed by Nam**

| **Geo mean. reduction in T gate count** | | | |
|---|---|---|---|
| **Amy** | **PyZX** | **Nam (H)** | **VOQC** ✓ |
| 39.7% | 42.6% | 41.4% | 41.4% |

**VOQC only outperformed by PyZX**

# No Bugs!

Bugs found in prior work[1,2] via translation validation

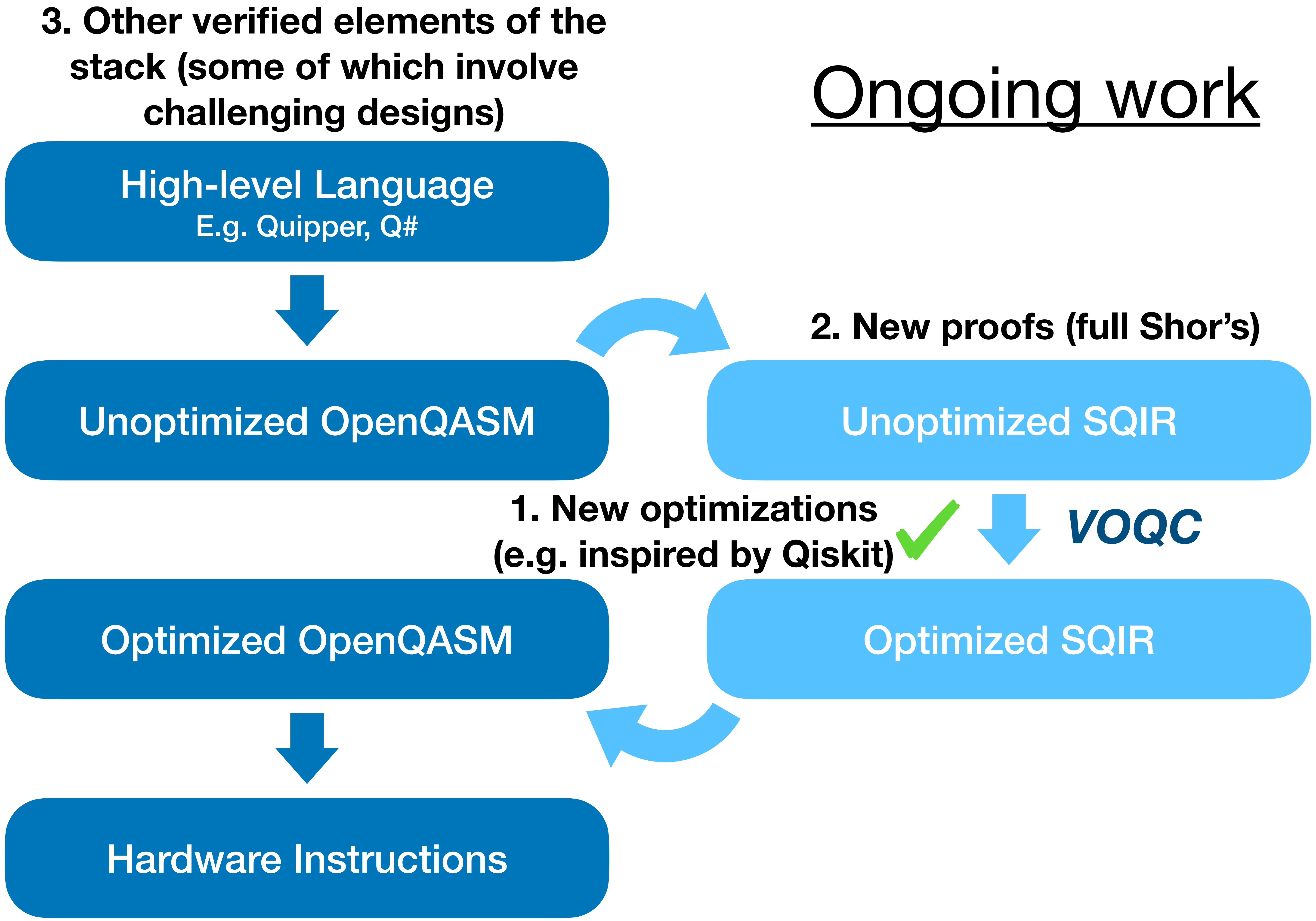
| Name | Total Gate Count | | | | | |
|---|---|---|---|---|---|---|
| | Original | Qiskit | t\|ket⟩ | Nam (L) | Nam (H) | VOQC |
| adder_8 | 900 | 805 | 775 | 646 | **606** | 682 |
| barenco_tof_3 | 58 | 51 | 51 | 42 | **40** | 50 |
| barenco_tof_4 | 114 | 100 | 100 | 78 | **72** | 95 |
| barenco_tof_5 | 170 | 149 | 149 | 114 | **104** | 140 |
| barenco_tof_10 | 450 | 394 | 394 | 294 | **264** | 365 |
| csla_mux_3 | 170 | 156 | **155** | 161 | **155** | 158 |
| csum_mux_9 | 420 | 382 | 361 | 294 | **266** | 308 |
| gf2^4_mult | 225 | 206 | 206 | **187** | **187** | 192 |
| gf2^5_mult | 347 | 318 | 319 | 296 | 296 | **291** |
| gf2^6_mult | 495 | 454 | 454 | **403** | **403** | 410 |
| gf2^7_mult | 669 | 614 | 614 | 555 | 555 | **549** |
| gf2^8_mult | 883 | 804 | 806 | 712 | 712 | **705** |
| gf2^9_mult | 1095 | 1006 | 1009 | 891 | 891 | **885** |
| gf2^10_mult | 1347 | 1238 | 1240 | **1070** | **1070** | 1084 |
| gf2^16_mult | 3435 | 3148 | 3150 | 2707 | 2707 | **2695** |
| gf2^32_mult | 13593 | 12506 | 12507 | 10601 | 10601 | **10577** |
| mod5_4 | 63 | 58 | 58 | **51** | **51** | 56 |
| mod_mult_55 | 119 | 106 | 102 | 91 | 91 | **90** |
| mod_red_21 | 278 | 227 | 224 | 184 | **180** | 214 |
| qcla_adder_10 | 521 | 469 | 460 | 411 | **399** | 438 |
| qcla_com_7 | 443 | 398 | 392 | **284** | **284** | 314 |
| qcla_mod_7 | 884 | 793 | 780 | 636 | 624 | **723** |
| rc_adder_6 | 200 | 170 | 172 | 142 | **140** | 157 |
| tof_3 | 45 | 40 | 40 | **35** | **35** | 40 |
| tof_4 | 75 | 66 | 66 | **55** | **55** | 65 |
| tof_5 | 105 | 92 | 92 | **75** | **75** | 90 |
| tof_10 | 255 | 222 | 222 | **175** | **175** | 215 |
| vbe_adder_3 | 150 | 138 | 139 | **89** | **89** | 101 |
| **Geo. Mean Reduction** | – | 10.1% | 10.6% | 23.3% | 24.8% | 17.8% |

| Name | *T*-Gate Count | | | | | |
|---|---|---|---|---|---|---|
| | Original | Amy | PyZX | Nam (L) | Nam (H) | VOQC |
| adder_8 | 399 | 215 | **173** | 215 | 215 | 215 |
| barenco_tof_3 | 28 | **16** | **16** | **16** | **16** | **16** |
| barenco_tof_4 | 56 | **28** | **28** | **28** | **28** | **28** |
| barenco_tof_5 | 84 | **40** | **40** | **40** | **40** | **40** |
| barenco_tof_10 | 224 | **100** | **100** | **100** | **100** | **100** |
| csla_mux_3 | 70 | 62 | **62** | 64 | 64 | 64 |
| csum_mux_9 | 196 | 112 | **84** | **84** | **84** | **84** |
| gf2^4_mult | 112 | **68** | **68** | **68** | **68** | **68** |
| gf2^5_mult | 175 | **111** | 115 | 115 | 115 | 115 |
| gf2^6_mult | 252 | **150** | **150** | **150** | **150** | **150** |
| gf2^7_mult | 343 | **217** | **217** | **217** | **217** | **217** |
| gf2^8_mult | 448 | **264** | **264** | **264** | **264** | **264** |
| gf2^9_mult | 567 | **351** | **351** | **351** | **351** | **351** |
| gf2^10_mult | 700 | **410** | **410** | **410** | **410** | **410** |
| gf2^16_mult | 1792 | **1040** | **1040** | **1040** | **1040** | **1040** |
| gf2^32_mult | 7168 | **4128** | **4128** | **4128** | **4128** | **4128** |
| mod5_4 | 28 | 16 | **8** | 16 | 16 | 16 |
| mod_mult_55 | 49 | 37 | **35** | **35** | **35** | **35** |
| mod_red_21 | 119 | **73** | **73** | **73** | **73** | **73** |
| qcla_adder_10 | 238 | **162** | **162** | **162** | **162** | 164 |
| qcla_com_7 | 203 | **95** | **95** | **95** | **95** | **95** |
| qcla_mod_7 | 413 | 249 | **237** | 237 | 235 | 249 |
| rc_adder_6 | 77 | 63 | **47** | **47** | **47** | **47** |
| tof_3 | 21 | **15** | **15** | **15** | **15** | **15** |
| tof_4 | 35 | **23** | **23** | **23** | **23** | **23** |
| tof_5 | 49 | **31** | **31** | **31** | **31** | **31** |
| tof_10 | 119 | **71** | **71** | **71** | **71** | **71** |
| vbe_adder_3 | 70 | **24** | **24** | **24** | **24** | **24** |
| **Geo. Mean Reduction** | – | 39.7% | 42.6% | 41.4% | 41.4% | 41.4% |

[1] Nam, Ross, Su, Childs and Maslov. *Automated Optimization of Large Quantum Circuits with Continuous Parameters*. npj 2018.
[2] Kissinger and van de Wetering. *PyZX: Large scale automated diagrammatic reasoning*. QPL 2019.

# Summary and Future Work

- **SQIR** and **VOQC**: Two building blocks of a verified quantum software stack
  - Powerful enough to verify **state-of-the-art optimizations**, and prove source programs **correct** (QPE; Grover's)
  - Resulted in **novel frameworks, libraries, automation** for quantum program proofs




github.com/inQWIRE/SQIR