Statistical Assertions for Validating Patterns and Finding Bugs in Quantum Programs

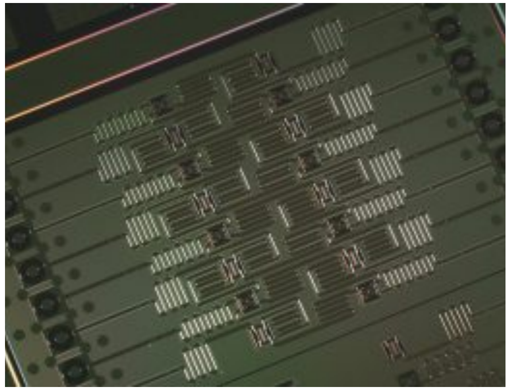
Yipeng Huang and Margaret Martonosi

PRINCETON UNIVERSITY

This work is funded in part by EPiQC, an NSF Expedition in Computing, under grant 1730082

Motivation: Race to practical quantum computation

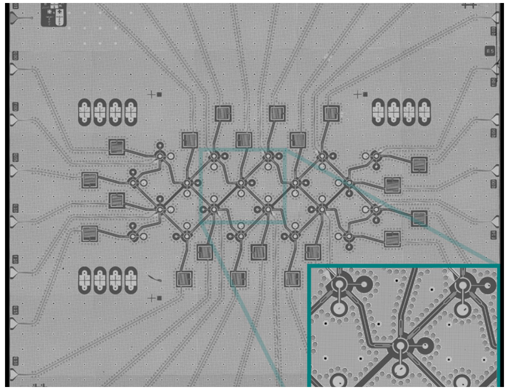
Superconducting qubits



IBM

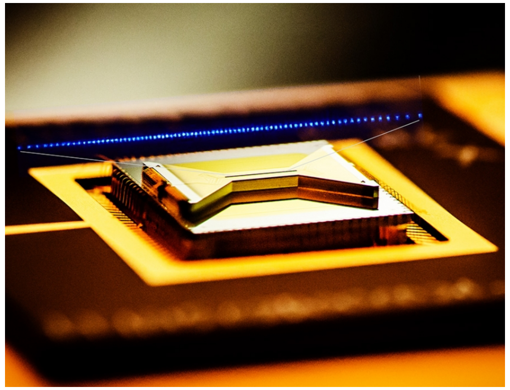
Google

Intel



Rigetti

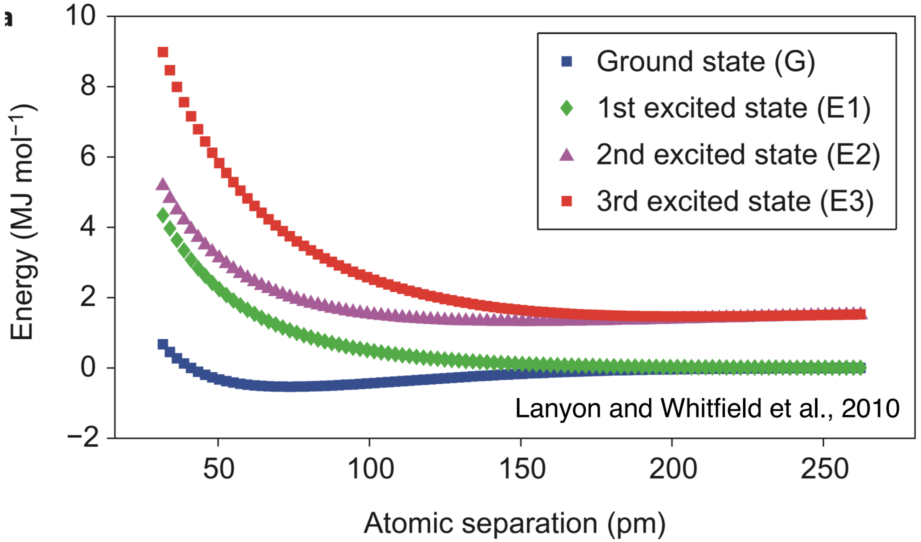
Trapped ion qubits



University of Maryland / IonQ

Many research teams now competing towards more reliable and more numerous qubits.

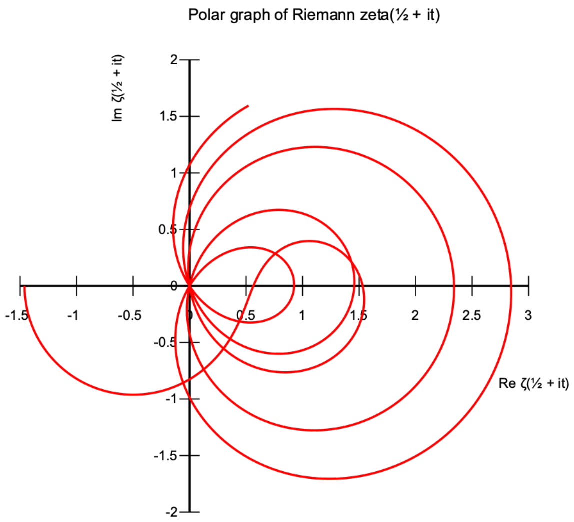
Motivation: Race to practical quantum computation



Lanyon and Whitfield et al., 2010

Quantum algorithms for chemical simulations

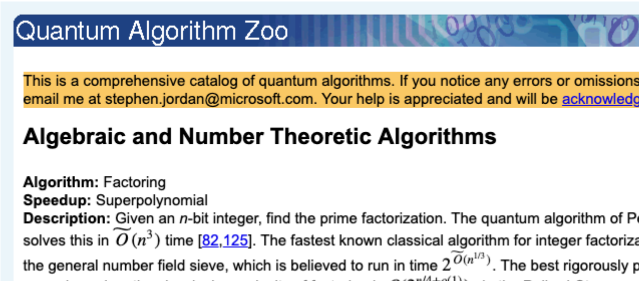
- Calculate properties of molecules directly from governing equations
- Use quantum mechanical computer to simulate quantum mechanics!



Shor’s quantum algorithm for factoring integers

- Factor large integers to primes in polynomial time complexity
- Surpasses any known classical algorithm taking exponential time complexity

Hundreds of algorithms @ QuantumAlgorithmZoo.org



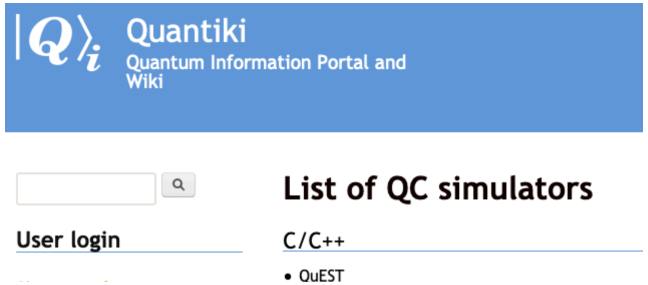
Quantum Algorithm Zoo

This is a comprehensive catalog of quantum algorithms. If you notice any errors or omission
email me at stephen.jordan@microsoft.com. Your help is appreciated and will be acknowledg

Algebraic and Number Theoretic Algorithms

Algorithm: Factoring
Speedup: Superpolynomial
Description: Given an *n*-bit integer, find the prime factorization. The quantum algorithm of P
solves this in $\widetilde{O}(n^3)$ time [82,125]. The fastest known classical algorithm for integer factoriza
the general number field sieve, which is believed to run in time $2^{\widetilde{O}(n^{1/3})}$. The best rigorously p

Hundreds of
quantum algorithm
specifications



Quantiki
Quantum Information Portal and Wiki

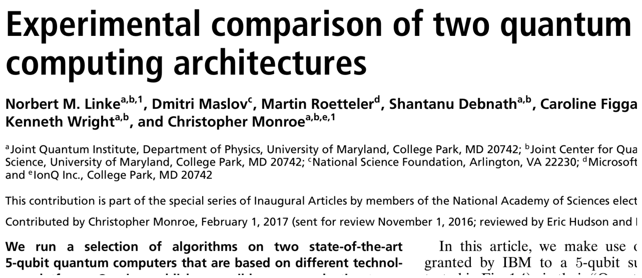
User login

List of QC simulators

C/C++

- QuEST

Dozens of
quantum
programming languages
and open source
software packages



Experimental comparison of two quantum computing architectures

Norbert M. Linke[a,b,1], Dmitri Maslov[c], Martin Roetteler[d], Shantanu Debnath[a,b], Caroline Figga
Kenneth Wright[a,b], and Christopher Monroe[a,b,e,1]

[a]Joint Quantum Institute, Department of Physics, University of Maryland, College Park, MD 20742; [b]Joint Center for Qua
Science, University of Maryland, College Park, MD 20742; [c]National Science Foundation, Arlington, VA 22230; [d]Microsoft
and [e]IonQ Inc., College Park, MD 20742

This contribution is part of the special series of Inaugural Articles by members of the National Academy of Sciences elect

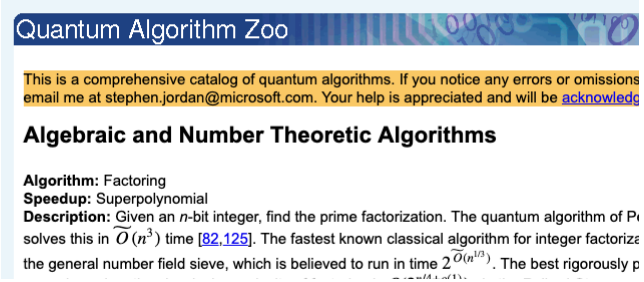
Contributed by Christopher Monroe, February 1, 2017 (sent for review November 1, 2016; reviewed by Eric Hudson and

We run a selection of algorithms on two state-of-the-art 5-qubit quantum computers that are based on different technol-

In this article, we make use
granted by IBM to a 5-qubit s

Very few
quantum algorithms
actually written
and tested
as program code

GAP!

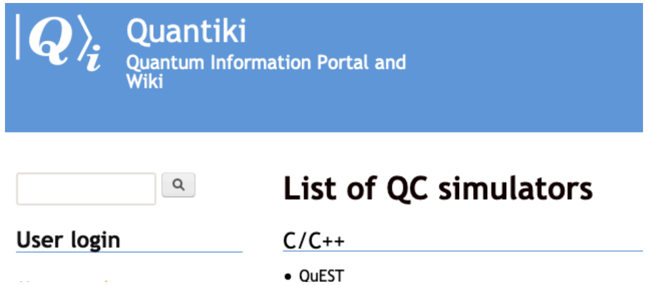


Quantum Algorithm Zoo

This is a comprehensive catalog of quantum algorithms. If you notice any errors or omissions
email me at stephen.jordan@microsoft.com. Your help is appreciated and will be acknowledg

Algebraic and Number Theoretic Algorithms

Algorithm: Factoring
Speedup: Superpolynomial
Description: Given an *n*-bit integer, find the prime factorization. The quantum algorithm of P
solves this in $\widetilde{O}(n^3)$ time [82,125]. The fastest known classical algorithm for integer factoriz
the general number field sieve, which is believed to run in time $2^{\widetilde{O}(n^{1/3})}$. The best rigorously p



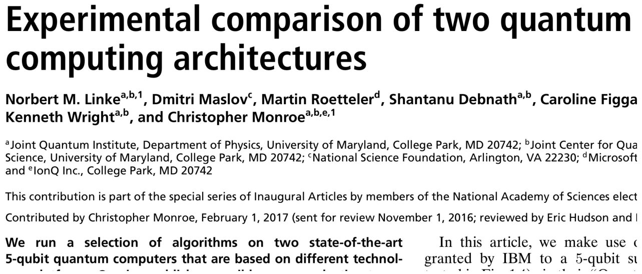
Quantiki
Quantum Information Portal and Wiki

User login

List of QC simulators

C/C++

- QuEST



Experimental comparison of two quantum computing architectures

Norbert M. Linke[a,b,1], Dmitri Maslov[c], Martin Roetteler[d], Shantanu Debnath[a,b], Caroline Figga
Kenneth Wright[a,b], and Christopher Monroe[a,b,e,1]

[a]Joint Quantum Institute, Department of Physics, University of Maryland, College Park, MD 20742; [b]Joint Center for Qua
Science, University of Maryland, College Park, MD 20742; [c]National Science Foundation, Arlington, VA 22230; [d]Microsoft
and [e]IonQ Inc., College Park, MD 20742

This contribution is part of the special series of Inaugural Articles by members of the National Academy of Sciences elect

Contributed by Christopher Monroe, February 1, 2017 (sent for review November 1, 2016; reviewed by Eric Hudson and

We run a selection of algorithms on two state-of-the-art
5-qubit quantum computers that are based on different technol-

In this article, we make use
granted by IBM to a 5-qubit s

Hundreds of
quantum algorithm
specifications

Dozens of
quantum
programming languages
and open source
software packages

Very few
quantum algorithms
actually written
and tested
as program code

Software tools gap:
Need higher level
programming
languages,
optimizing compilers,
debuggers

Hardware and simulator infrastructure gap:
Need scalable
simulators,
and more abundant &
reliable qubits

Outline: This paper addresses three challenges in quantum debugging

1

Programmers cannot easily read variable values while a quantum program runs.

Stop programs early at various points & observe values, in real hardware or simulation.

2

Quantum states are difficult to understand, so they offer limited help for debugging.

Use statistical assertions to decide if states are classical, superposition, or entangled.

3

Programmers don't yet have guidelines for where & what to check in programs.

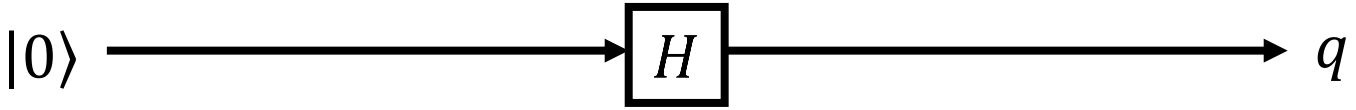
Use a bug taxonomy & program patterns in benchmark quantum algorithms as a guide.

Outline: This paper addresses three challenges in quantum debugging

1

Programmers cannot easily read variable values while a quantum program runs.

Quantum computing primer to understand debugging challenge



Classical value

Deterministic

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$$

$$|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$$

Hadamard gate

A quantum operator

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$$

$$q = H|0\rangle$$

Quantum qubit

Superposition

$$q = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

QC variables' ability to be simultaneously in several values underlies power of quantum computing.

Quantum computing primer to understand debugging challenge

Classical value
Deterministic

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$$

$$|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$$

Hadamard gate
A quantum operator

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$$

$$q = H|0\rangle$$

Quantum qubit
Superposition

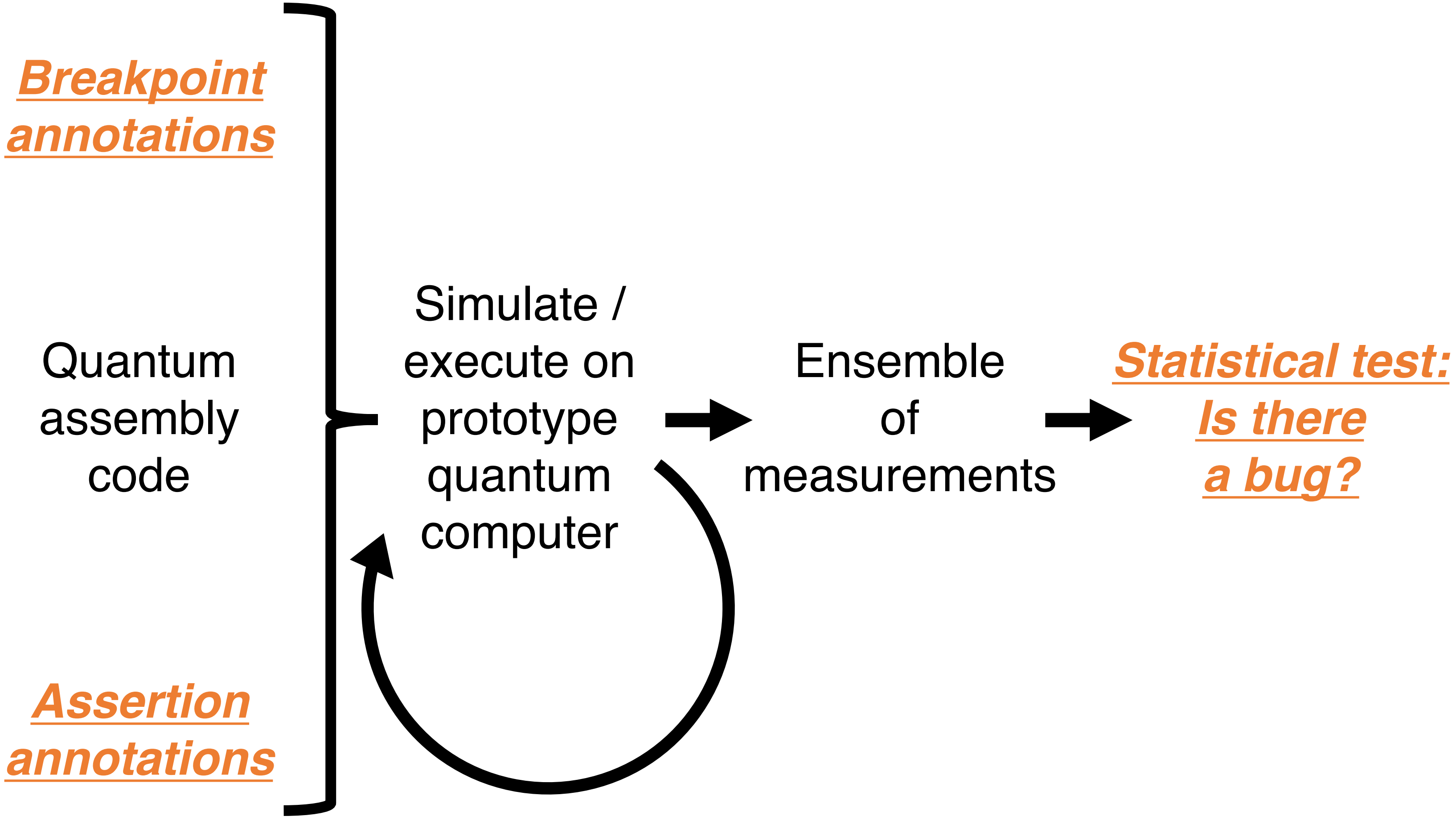
$$q = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

Measurement
Collapses state

$$m = \begin{cases} 0, P = {}^{1}\!/_{2} \\ 1, P = {}^{1}\!/_{2} \end{cases}$$

We cannot pause a quantum computer and "printf debug," because measurement collapses state.

Toolchain for debugging programs with tests on measurements



Breakpoint annotations

Quantum assembly code

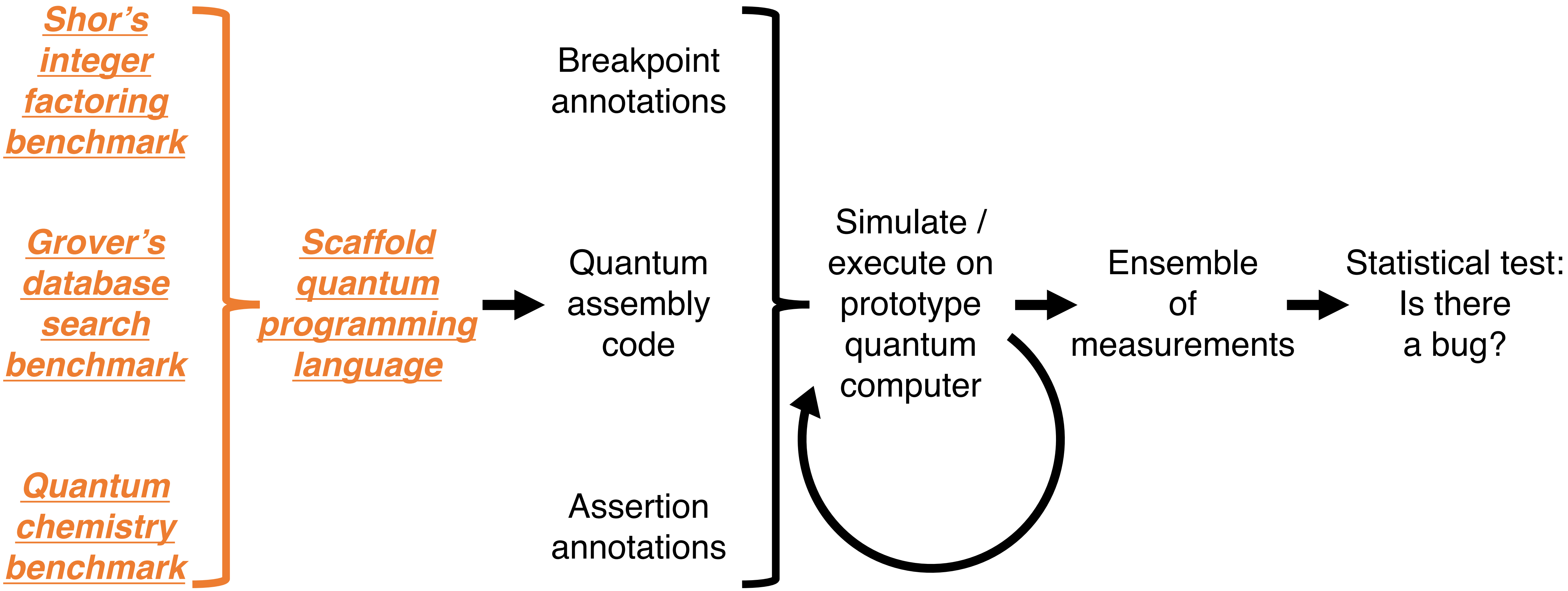
Assertion annotations

Simulate / execute on prototype quantum computer

Ensemble of measurements

Statistical test: Is there a bug?

Toolchain for debugging programs with tests on measurements



Shor's integer factoring benchmark

Grover's database search benchmark

Quantum chemistry benchmark

Scaffold quantum programming language

Breakpoint annotations

Quantum assembly code

Assertion annotations

Simulate / execute on prototype quantum computer

Ensemble of measurements

Statistical test: Is there a bug?

Outline: This paper addresses three challenges in quantum debugging

1

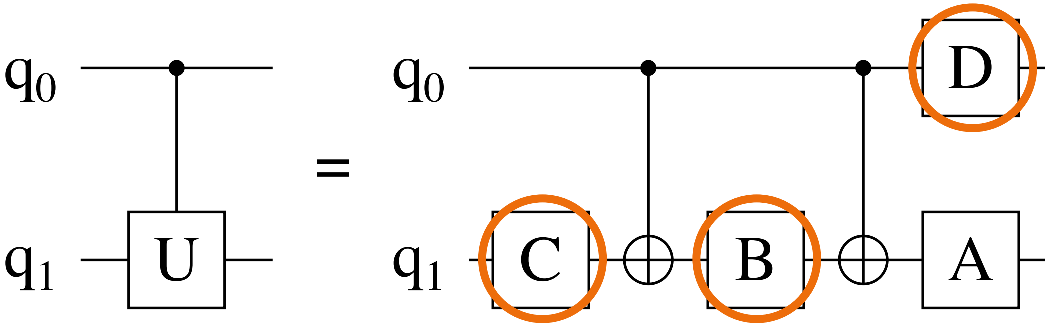
Programmers cannot easily read variable values while a quantum program runs.

Stop programs early at various points & observe values, in real hardware or simulation.

2

Quantum states are difficult to understand, so they offer limited help for debugging.

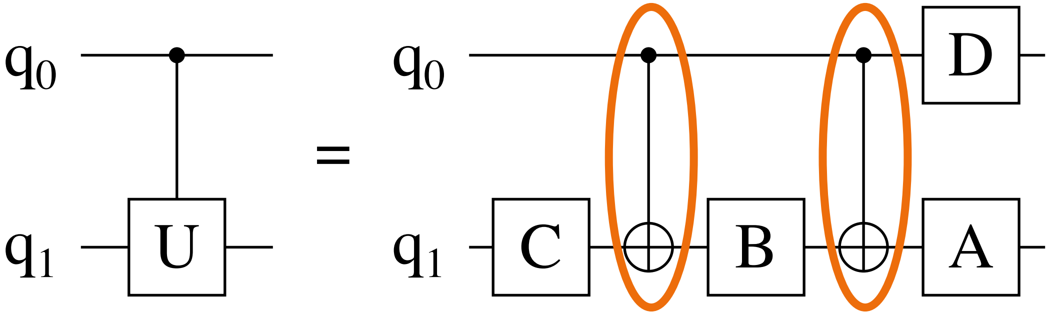
Even simple quantum programming bugs lead to non-obvious symptoms



Elementary single-qubit operations

```
Rz(q1, +angle/2); // C
CNOT(q0, q1);
Rz(q1, -angle/2); // B
CNOT(q0, q1);
Rz(q0, +angle/2); // D
```

Even simple quantum programming bugs lead to non-obvious symptoms

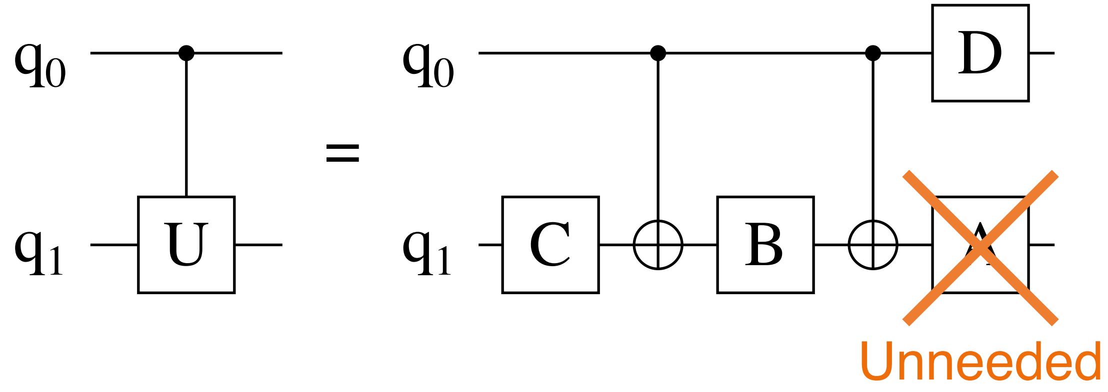


Elementary two-qubit operations

```
Rz(q1, +angle/2); // C
CNOT(q0, q1);
Rz(q1, -angle/2); // B
CNOT(q0, q1);
Rz(q0, +angle/2); // D
```

Correct,
operation A unneeded

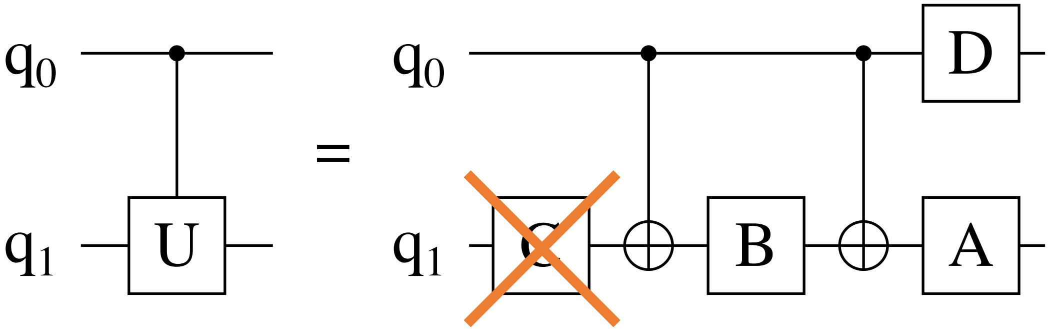
Even simple quantum programming bugs lead to non-obvious symptoms



```
Rz(q1, +angle/2); // C
CNOT(q0, q1);
Rz(q1, -angle/2); // B
CNOT(q0, q1);
Rz(q0, +angle/2); // D
```

**Correct,
operation A unneeded**

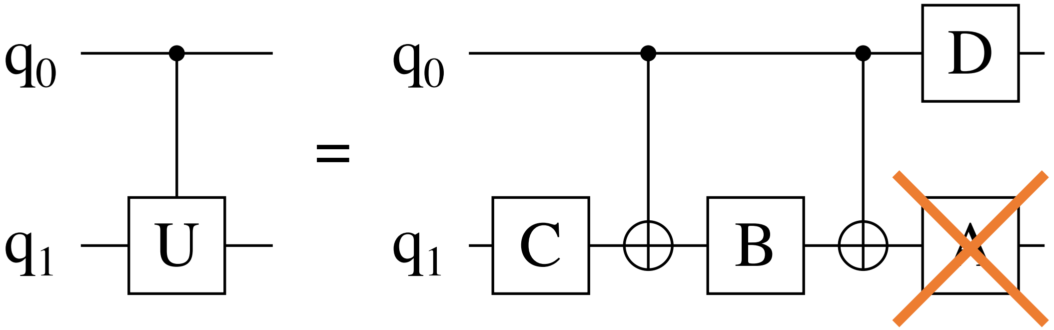
Even simple quantum programming bugs lead to non-obvious symptoms



Unneeded

`Rz(q1, +angle/2); // C`
`CNOT(q0, q1);`
`Rz(q1, -angle/2); // B`
`CNOT(q0, q1);`
`Rz(q0, +angle/2); // D`
 	
Correct,
operation A unneeded
	`CNOT(q0, q1);`
	`Rz(q1, -angle/2); // B`
	`CNOT(q0, q1);`
	`Rz(q1, +angle/2); // A`
	`Rz(q0, +angle/2); // D`
	
	Correct,
	operation C unneeded	

Even simple quantum programming bugs lead to non-obvious symptoms



Unneeded?

But signs on angles wrong!

```
Rz(q1, +angle/2); // C
CNOT(q0, q1);
Rz(q1, -angle/2); // B
CNOT(q0, q1);
Rz(q0, +angle/2); // D
```

**Correct,
operation A unneeded**

```
CNOT(q0, q1);
Rz(q1, -angle/2); // B
CNOT(q0, q1);
Rz(q1, +angle/2); // A
Rz(q0, +angle/2); // D
```

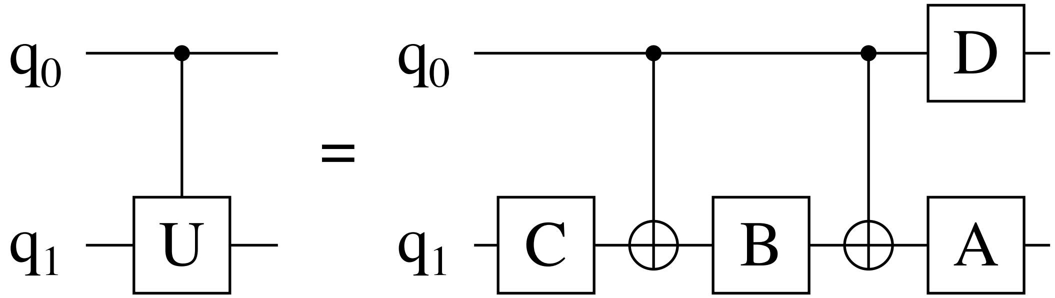
**Correct,
operation C unneeded**

```
Rz(q1, -angle/2);
CNOT(q0, q1);
Rz(q1, +angle/2);
CNOT(q0, q1);
Rz(q0, +angle/2); // D
```

**Incorrect,
angles flipped**

Many ways to translate basic quantum operations to program code—many details to get right!

Even simple quantum programming bugs lead to non-obvious symptoms



```
Rz(q1, +angle/2); // C
CNOT(q0, q1);
Rz(q1, -angle/2); // B
CNOT(q0, q1);
Rz(q0, +angle/2); // D
```

**Correct,
operation A unneeded**

$$|11\rangle \rightarrow e^{i*angle}|11\rangle$$

```
CNOT(q0, q1);
Rz(q1, -angle/2); // B
CNOT(q0, q1);
Rz(q1, +angle/2); // A
Rz(q0, +angle/2); // D
```

**Correct,
operation C unneeded**

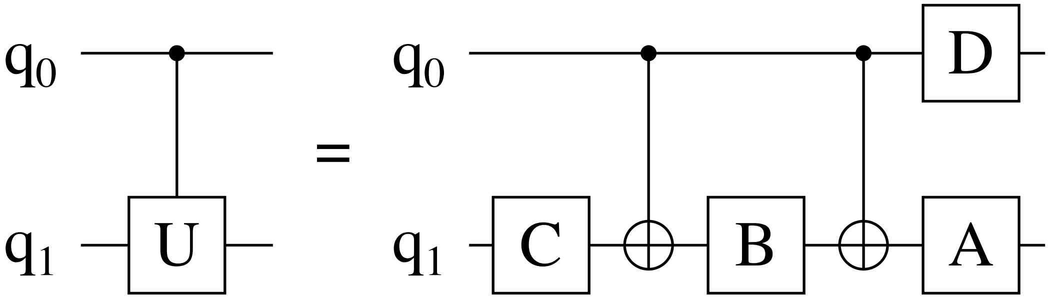
$$|11\rangle \rightarrow e^{i*angle}|11\rangle$$

```
Rz(q1, -angle/2);
CNOT(q0, q1);
Rz(q1, +angle/2);
CNOT(q0, q1);
Rz(q0, +angle/2); // D
```

**Incorrect,
angles flipped**

$$|11\rangle \rightarrow e^{-i*angle}|11\rangle$$

Even simple quantum programming bugs lead to non-obvious symptoms



```
Rz(q1, +angle/2); // C
CNOT(q0, q1);
Rz(q1, -angle/2); // B
CNOT(q0, q1);
Rz(q0, +angle/2); // D
```

Correct,
operation A unneeded

$|11\rangle \rightarrow e^{i*angle}|11\rangle$

```
CNOT(q0, q1);
Rz(q1, -angle/2); // B
CNOT(q0, q1);
Rz(q1, +angle/2); // A
Rz(q0, +angle/2); // D
```

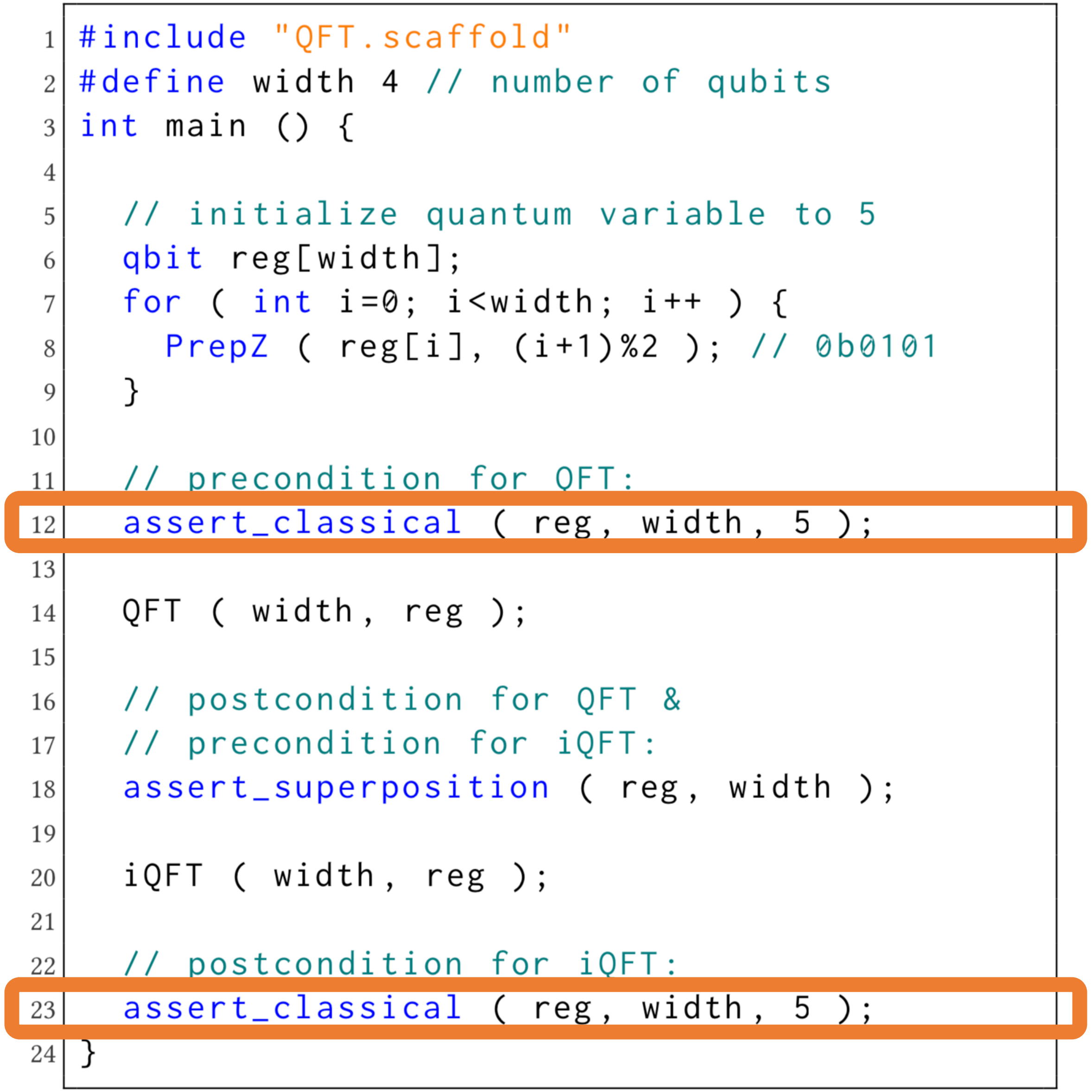
Correct,
operation C unneeded

$|11\rangle \rightarrow e^{i*angle}|11\rangle$

```
Rz(q1, -angle/2);
CNOT(q0, q1);
Rz(q1, +angle/2);
CNOT(q0, q1);
Rz(q0, +angle/2); // D
```

Incorrect,
angles flipped

$|11\rangle \rightarrow e^{-i*angle}|11\rangle$



```
#include "QFT.scaffold"
#define width 4 // number of qubits
int main () {

  // initialize quantum variable to 5
  qbit reg[width];
  for ( int i=0; i<width; i++ ) {
    PrepZ ( reg[i], (i+1)%2 ); // 0b0101
  }

  // precondition for QFT:
  assert_classical ( reg, width, 5 );

  QFT ( width, reg );

  // postcondition for QFT &
  // precondition for iQFT:
  assert_superposition ( reg, width );

  iQFT ( width, reg );

  // postcondition for iQFT:
  assert_classical ( reg, width, 5 );
}
```

Listing 1: Test harness for quantum Fourier transform.

Assertions on classical & superposition states help us decide whether programs are correct

Testbench for quantum Fourier transform, consisting of controlled-rotations

QFT and iQFT should be inverses, but bug in controlled-rotations would lead to flawed inversion

Flawed inversion caught in failure of classical assertion based on Chi-squared tests

Outline: This paper addresses three challenges in quantum debugging

1

Programmers cannot easily read variable values while a quantum program runs.

Stop programs early at various points & observe values, in real hardware or simulation.

2

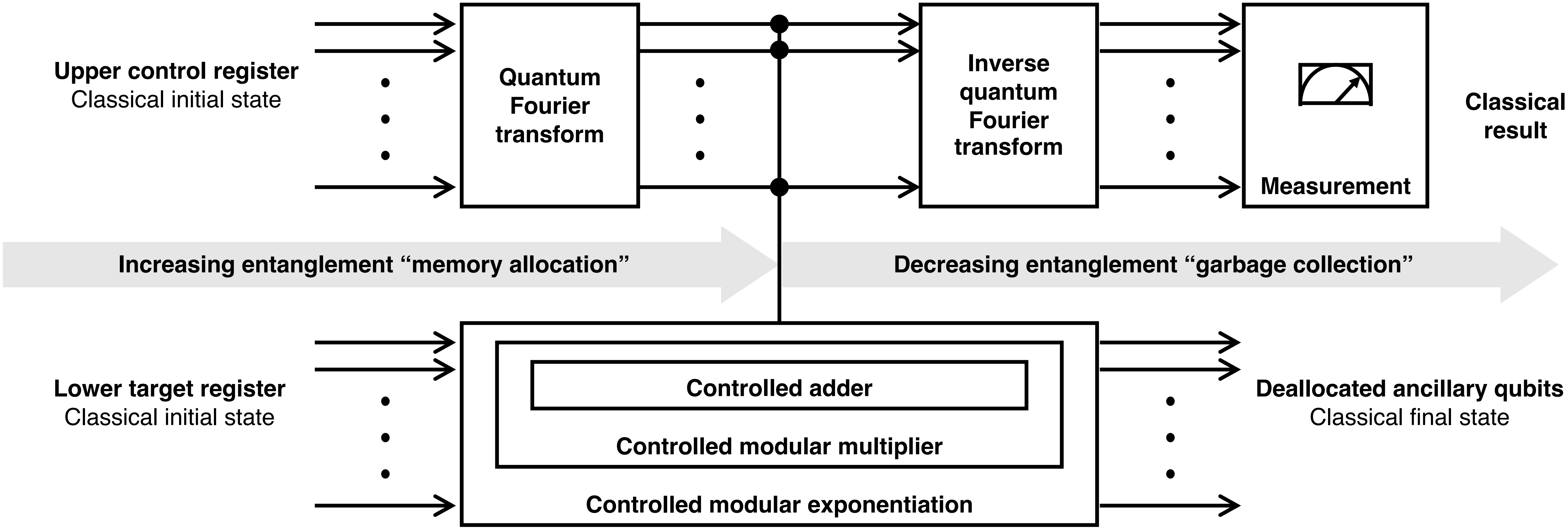
Quantum states are difficult to understand, so they offer limited help for debugging.

Use statistical assertions to decide if states are classical, superposition, or entangled.

3

Programmers don't yet have guidelines for where & what to check in programs.

Structure of quantum algorithm primitives tells programmers what to check



Upper control register
Classical initial state

Quantum Fourier transform

Inverse quantum Fourier transform

Measurement

Classical result

Increasing entanglement "memory allocation"

Decreasing entanglement "garbage collection"

Lower target register
Classical initial state

Controlled adder

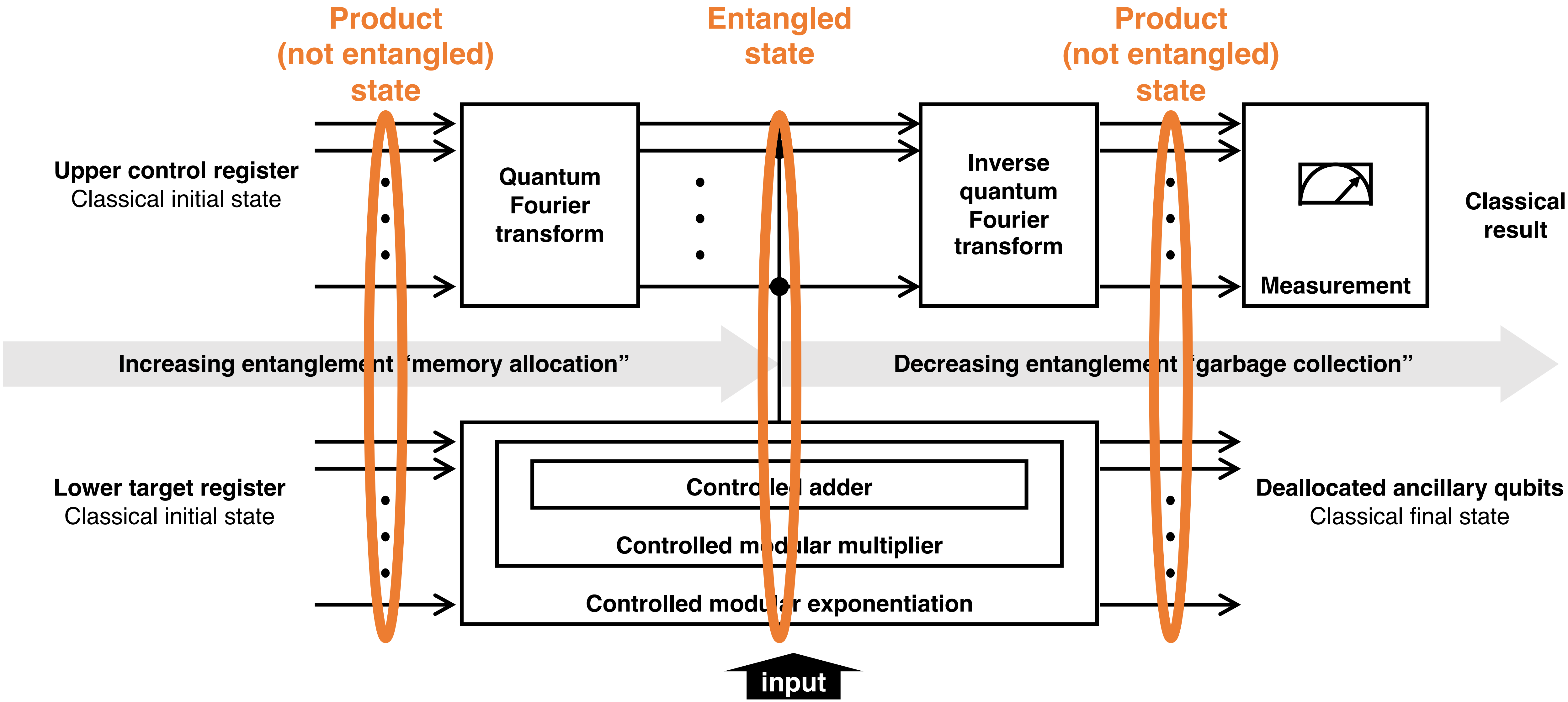
Controlled modular multiplier

Controlled modular exponentiation

Deallocated ancillary qubits
Classical final state

Bring up Shor's algorithm w/ library of quantum program modules, unit tests, & integration tests.

Structure of quantum algorithm primitives tells programmers what to check



Product (not entangled) state

Entangled state

Product (not entangled) state

Upper control register
Classical initial state

Quantum Fourier transform

Inverse quantum Fourier transform

Measurement

Classical result

Increasing entanglement “memory allocation”

Decreasing entanglement “garbage collection”

Lower target register
Classical initial state

Controlled adder

Controlled modular multiplier

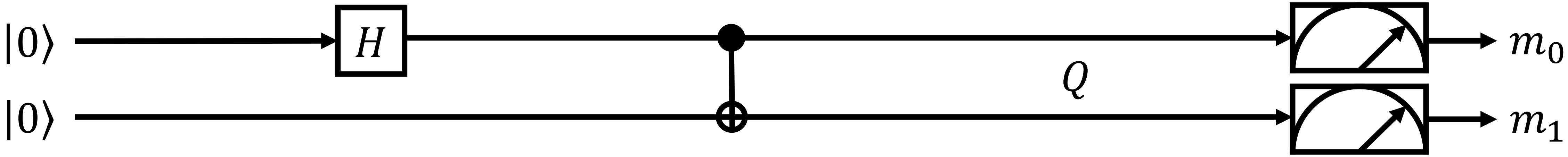
Controlled modular exponentiation

Deallocated ancillary qubits
Classical final state

input

k, the algorithm iteration	**0**	**1**	**2**	**3**	**…**
$a = 7^{2^k} \mod 15$	7	4	1	1	…
$a^{-1};\ a \times a^{-1} \equiv 1 \mod 15$	12	4	1	1	…

Entanglement tutorial to understand programming patterns



Two qubits
Tensor product

$$|0\rangle \otimes |0\rangle = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} = |00\rangle$$

Product state
Can be factored

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

$$= \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle$$

Controlled-NOT
Two-qubit operator

$$\mathrm{CNOT} = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix}$$

Entangled state
Cannot be factored

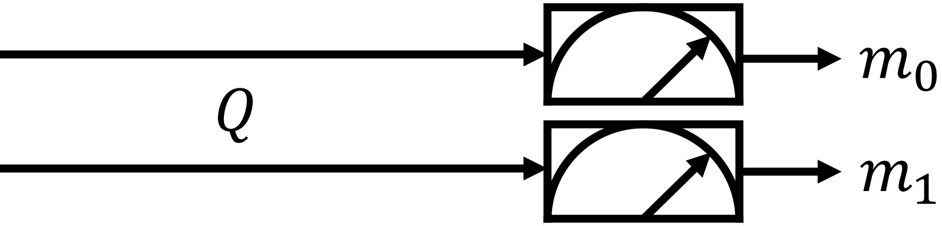
$$Q = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

Measurement
Results correlated

$$(m_0, m_1)$$

$$= \begin{cases}(0,0), P = {}^{1}\!/_{2}\\(1,1), P = {}^{1}\!/_{2}\end{cases}$$

Entanglement tutorial to understand programming patterns



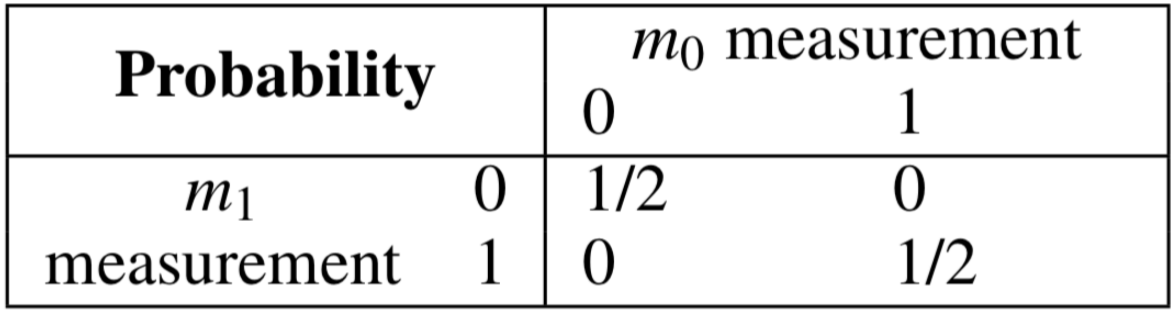
Entangled state
Cannot be factored

$$Q = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

Measurement
Results correlated

$$(m_0, m_1)$$

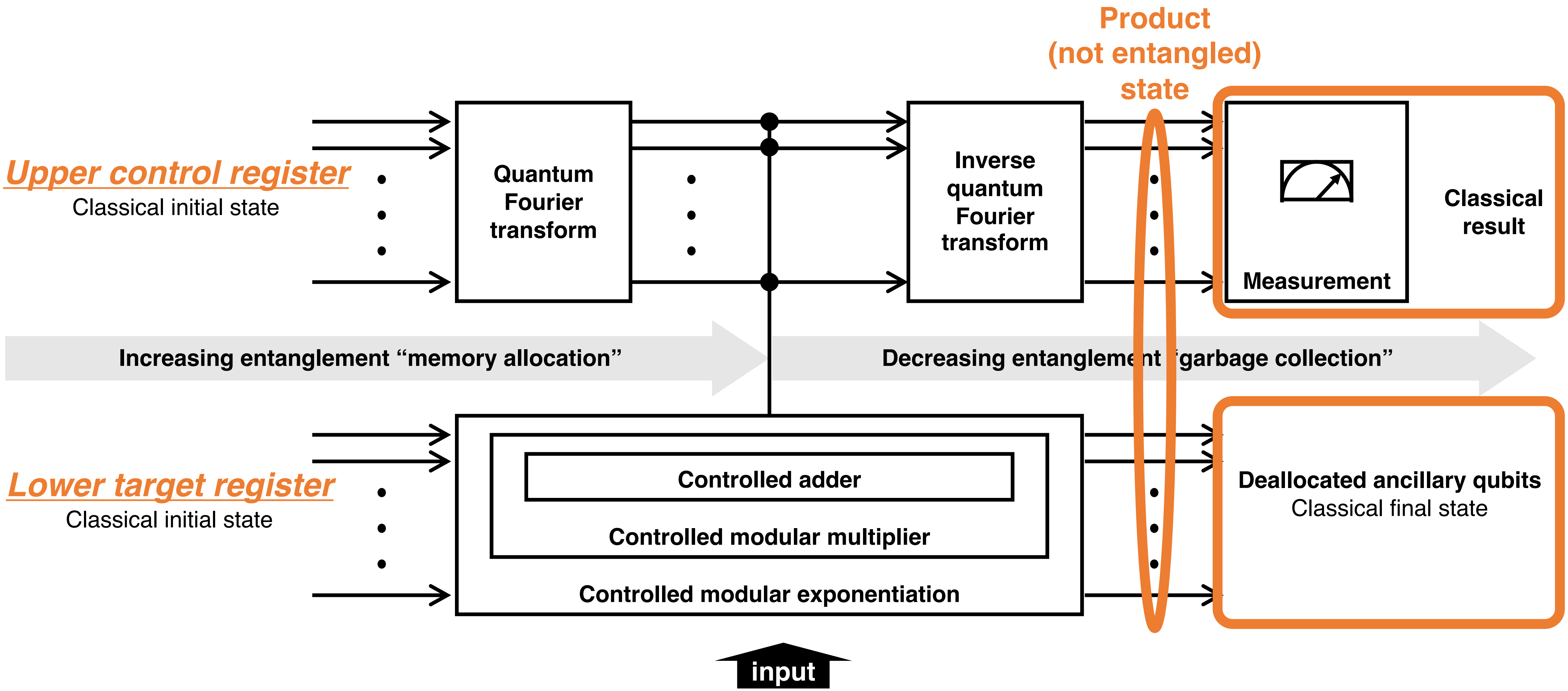
$$= \begin{cases} (0,0), P = {}^{1}/_{2} \\ (1,1), P = {}^{1}/_{2} \end{cases}$$



Probability		m_0 measurement	
		0	1
m_1 measurement	0	1/2	0
	1	0	1/2

Contingency table analysis + chi-squared statistical test decides if sets of variables are correlated

Structure of quantum algorithm primitives tells programmers what to check



Product (not entangled) state

Upper control register

Classical initial state

Quantum Fourier transform

Inverse quantum Fourier transform

Measurement

Classical result

Increasing entanglement "memory allocation"

Decreasing entanglement "garbage collection"

Lower target register

Classical initial state

Controlled adder

Controlled modular multiplier

Controlled modular exponentiation

Deallocated ancillary qubits

Classical final state

input

k, **the algorithm iteration**	**0**	**1**	**2**	**3**	**…**
$a = 7^{2^k} \mod 15$	7	4	1	1	…
$a^{-1}; a \times a^{-1} \equiv 1 \mod 15$	12	4	1	1	…

QC programming patterns + entanglement assertions help find bugs

probability		**Upper control register**							
		0	1	2	3	4	5	6	7
Lower target register	0	1/8	0	1/8	0	1/8	0	1/8	0
	2	1/64	1/64	1/64	1/64	1/64	1/64	1/64	1/64
	7	1/64	1/64	1/64	1/64	1/64	1/64	1/64	1/64
	8	1/64	1/64	1/64	1/64	1/64	1/64	1/64	1/64
	13	1/64	1/64	1/64	1/64	1/64	1/64	1/64	1/64

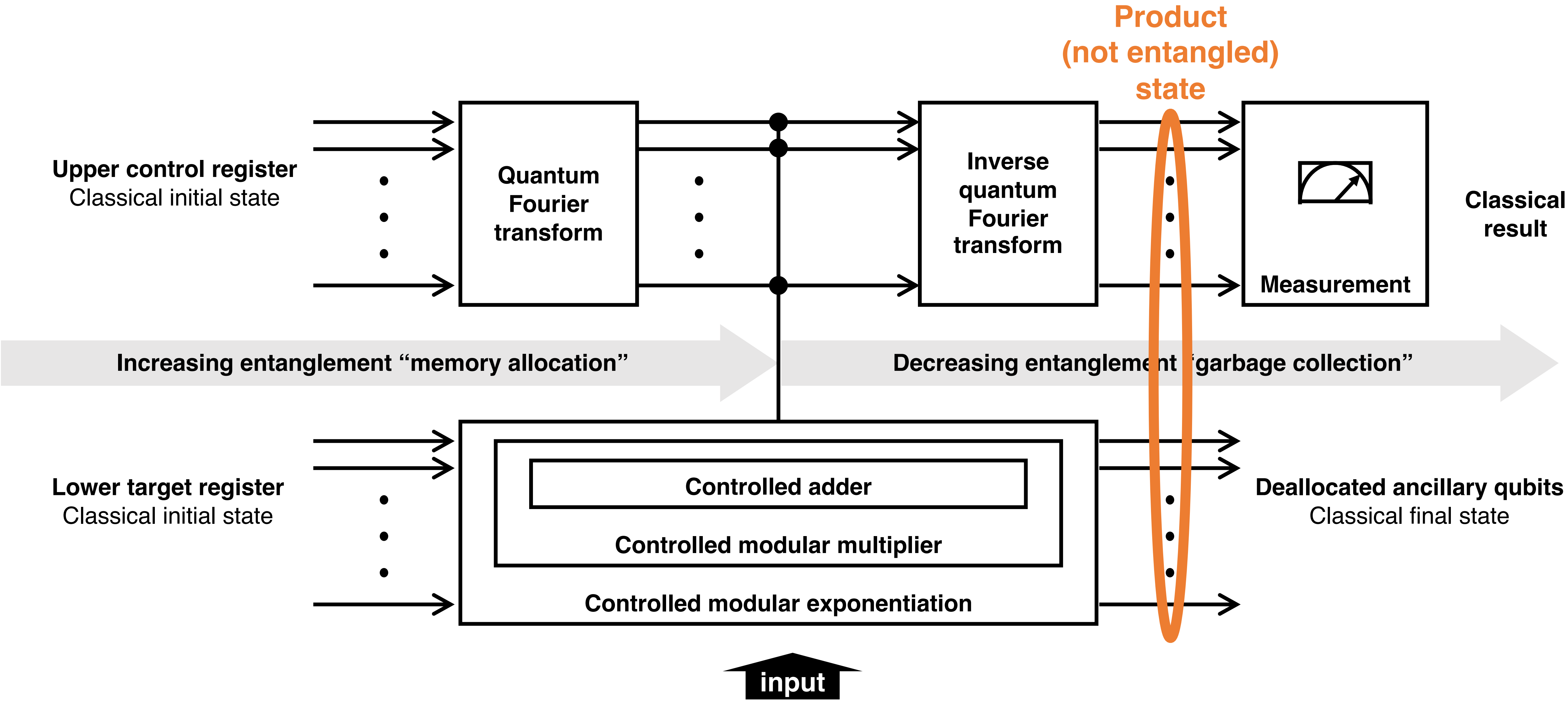
The two registers in the contingency table are correlated.

QC programming patterns + entanglement assertions help find bugs

probability		**Upper control register**							
		0	1	2	3	4	5	6	7
Lower target register	0	1/8	0	1/8	0	1/8	0	1/8	0
	~~2~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~
	~~7~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~
	~~8~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~
	~~13~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~	~~1/64~~

Chi-square test on table shows entanglement, causing assertion to fail.

Structure of quantum algorithm primitives tells programmers what to check



Product (not entangled) state

Upper control register
Classical initial state

Quantum Fourier transform

Inverse quantum Fourier transform

Measurement

Classical result

Increasing entanglement "memory allocation"

Decreasing entanglement "garbage collection"

Lower target register
Classical initial state

Controlled adder

Controlled modular multiplier

Controlled modular exponentiation

Deallocated ancillary qubits
Classical final state

input

k, **the algorithm iteration**	**0**	**1**	**2**	**3**	**…**
$a = 7^{2^k} \mod 15$	7	4	1	1	…
$a^{-1};\ a \times a^{-1} \equiv 1 \mod 15$	12	4	1	1	…

Outline: This paper addresses three challenges in quantum debugging

1

Programmers cannot easily read variable values while a quantum program runs.

Stop programs early at various points & observe values, in real hardware or simulation.

2

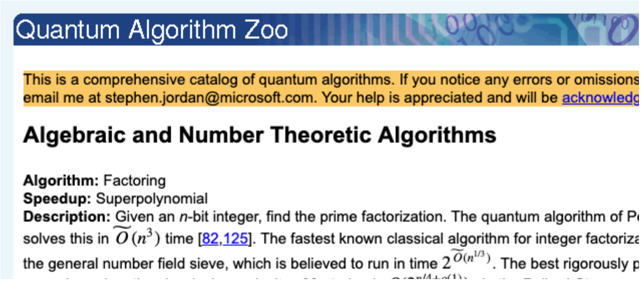
Quantum states are difficult to understand, so they offer limited help for debugging.

Use statistical assertions to decide if states are classical, superposition, or entangled.

3

Programmers don't yet have guidelines for where & what to check in programs.

Use a bug taxonomy & program patterns in benchmark quantum algorithms as a guide.

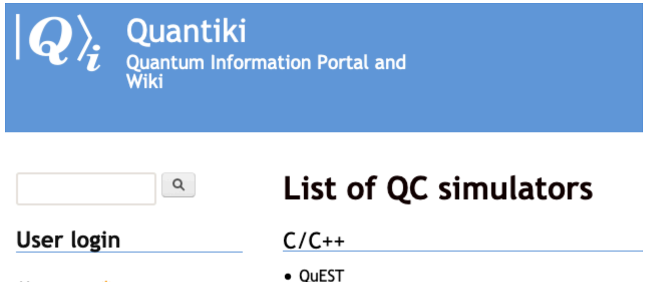


Quantum Algorithm Zoo

This is a comprehensive catalog of quantum algorithms. If you notice any errors or omissions
email me at stephen.jordan@microsoft.com. Your help is appreciated and will be acknowledg

Algebraic and Number Theoretic Algorithms

Algorithm: Factoring
Speedup: Superpolynomial
Description: Given an n-bit integer, find the prime factorization. The quantum algorithm of P
solves this in $\widetilde{O}(n^3)$ time [82,125]. The fastest known classical algorithm for integer factoriz
the general number field sieve, which is believed to run in time $2^{\widetilde{O}(n^{1/3})}$. The best rigorously p



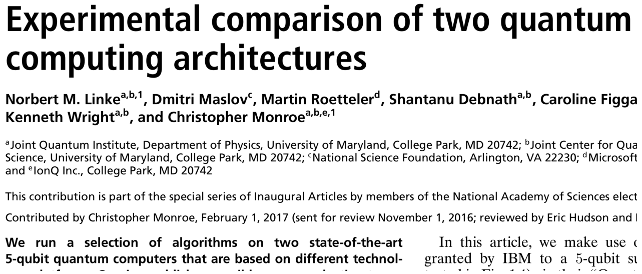
Quantiki
Quantum Information Portal and Wiki

List of QC simulators

User login

C/C++

- QuEST



Experimental comparison of two quantum computing architectures

Norbert M. Linke[a,b,1], Dmitri Maslov[c], Martin Roetteler[d], Shantanu Debnath[a,b], Caroline Figga
Kenneth Wright[a,b], and Christopher Monroe[a,b,e,1]

[a]Joint Quantum Institute, Department of Physics, University of Maryland, College Park, MD 20742; [b]Joint Center for Qua
Science, University of Maryland, College Park, MD 20742; [c]National Science Foundation, Arlington, VA 22230; [d]Microsoft
and [e]IonQ Inc., College Park, MD 20742

This contribution is part of the special series of Inaugural Articles by members of the National Academy of Sciences elect

Contributed by Christopher Monroe, February 1, 2017 (sent for review November 1, 2016; reviewed by Eric Hudson and

We run a selection of algorithms on two state-of-the-art 5-qubit quantum computers that are based on different technol-

In this article, we make use granted by IBM to a 5-qubit s

Hundreds of
quantum algorithm
specifications

Dozens of
quantum
programming languages
and open source
software packages

Very few
quantum algorithms
actually written
and tested
as program code

Software tools gap:
Need higher level
programming
languages,
optimizing compilers,
debuggers

Hardware and simulator infrastructure gap:
Need scalable
simulators,
and more abundant &
reliable qubits