An Invitation to the intersection of
Quantum Computing & Programming Languages

Xiaodi Wu
QuICS & UMD

UNIVERSITY OF MARYLAND

JOINT CENTER FOR QUANTUM INFORMATION AND COMPUTER SCIENCE

Analysis of Algorithms Artificial Intelligence
Combinatorial Algorithms Compilers Computational Complexity
Computer Architecture Computer Hardware Cryptography
Data Structures Databases Education Error Correcting Codes Finite Automata Graphics
Interactive Computing Internet Communications List Processing Numerical Analysis
Numerical Methods Object Oriented Programming Operating Systems Personal Computing
Program Verification Programming
Programming Languages Proof Construction Software
Theory Software Engineering
Verification of Hardware and Software Models Computer Systems Machine Learning
Parallel Computation

Computational Thinking in Quantum Computing

Quantum Application

Algorithm & Complexity

Variational Methods

Programming Languages

How to effectively express quantum applications and do trouble shooting?

Computational Thinking in Quantum Computing

Quantum Application

Algorithm & Complexity

Variational Methods

Programming Languages

System

Architecture

Quantum Control

Quantum Hardware Design

How to effectively translate high-level descriptions of quantum applications to quantum machine instructions?

How to automate the design of quantum devices and its verification?

PL as a tool for non-PL expert to explore

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.

  — not easy to define the resource consumption… hence no good model for complexity theory

  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory
  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory
  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

- **(theory)** Logic is the subject of identifying true statements in a language of which you only know a few words. Different logics study different language fragments:

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory
  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

- **(theory)** Logic is the subject of identifying true statements in a language of which you only know a few words. Different logics study different language fragments:
  — propositional logic: and, or, not, if … then

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory
  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

- **(theory)** Logic is the subject of identifying true statements in a language of which you only know a few words. Different logics study different language fragments:
  — propositional logic: and, or, not, if … then
  — temporal logic: propositional logic + today, tomorrow, eventually, never, …

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory
  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

- **(theory)** Logic is the subject of identifying true statements in a language of which you only know a few words. Different logics study different language fragments:
  — propositional logic: and, or, not, if … then
  — temporal logic: propositional logic + today, tomorrow, eventually, never, …

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory
  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

- **(theory)** Logic is the subject of identifying true statements in a language of which you only know a few words. Different logics study different language fragments:
  — propositional logic: and, or, not, if … then
  — temporal logic: propositional logic + today, tomorrow, eventually, never, …

- **(Mind-set)** Languages affect the way we think.

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory
  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

- **(theory)** Logic is the subject of identifying true statements in a language of which you only know a few words. Different logics study different language fragments:
  — propositional logic: and, or, not, if … then
  — temporal logic: propositional logic + today, tomorrow, eventually, never, …

- **(Mind-set)** Languages affect the way we think.
  — many constructs or so might be hard to imagine with quantum circuit abstraction

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory
  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

- **(theory)** Logic is the subject of identifying true statements in a language of which you only know a few words. Different logics study different language fragments:
  — propositional logic: and, or, not, if … then
  — temporal logic: propositional logic + today, tomorrow, eventually, never, …

- **(Mind-set)** Languages affect the way we think.
  — many constructs or so might be hard to imagine with quantum circuit abstraction
  — famous classical example: merge-sort and recursion. How about quantum recursion?

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory
  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

- **(theory)** Logic is the subject of identifying true statements in a language of which you only know a few words. Different logics study different language fragments:
  — propositional logic: and, or, not, if … then
  — temporal logic: propositional logic + today, tomorrow, eventually, never, …

- **(Mind-set)** Languages affect the way we think.
  — many constructs or so might be hard to imagine with quantum circuit abstraction
  — famous classical example: merge-sort and recursion. How about quantum recursion?

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory
  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

- **(theory)** Logic is the subject of identifying true statements in a language of which you only know a few words. Different logics study different language fragments:
  — propositional logic: and, or, not, if … then
  — temporal logic: propositional logic + today, tomorrow, eventually, never, …

- **(Mind-set)** Languages affect the way we think.
  — many constructs or so might be hard to imagine with quantum circuit abstraction
  — famous classical example: merge-sort and recursion. How about quantum recursion?

- **(automation)** PL or formal methods (FM) can help us automate many tedious tasks and provide instance-specific solutions

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory
  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

- **(theory)** Logic is the subject of identifying true statements in a language of which you only know a few words. Different logics study different language fragments:
  — propositional logic: and, or, not, if … then
  — temporal logic: propositional logic + today, tomorrow, eventually, never, …

- **(Mind-set)** Languages affect the way we think.
  — many constructs or so might be hard to imagine with quantum circuit abstraction
  — famous classical example: merge-sort and recursion. How about quantum recursion?

- **(automation)** PL or formal methods (FM) can help us automate many tedious tasks and provide instance-specific solutions
  — many detailed optimization and verification can be conducted with better correctness guarantees

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory
  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

- **(theory)** Logic is the subject of identifying true statements in a language of which you only know a few words. Different logics study different language fragments:
  — propositional logic: and, or, not, if … then
  — temporal logic: propositional logic + today, tomorrow, eventually, never, …

- **(Mind-set)** Languages affect the way we think.
  — many constructs or so might be hard to imagine with quantum circuit abstraction
  — famous classical example: merge-sort and recursion. How about quantum recursion?

- **(automation)** PL or formal methods (FM) can help us automate many tedious tasks and provide instance-specific solutions
  — many detailed optimization and verification can be conducted with better correctness guarantees
  — the only way to scale up these tasks

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory
  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

- **(theory)** Logic is the subject of identifying true statements in a language of which you only know a few words. Different logics study different language fragments:
  — propositional logic: and, or, not, if … then
  — temporal logic: propositional logic + today, tomorrow, eventually, never, …

- **(Mind-set)** Languages affect the way we think.
  — many constructs or so might be hard to imagine with quantum circuit abstraction
  — famous classical example: merge-sort and recursion. How about quantum recursion?

- **(automation)** PL or formal methods (FM) can help us automate many tedious tasks and provide instance-specific solutions
  — many detailed optimization and verification can be conducted with better correctness guarantees
  — the only way to scale up these tasks

PL as a tool for non-PL expert to explore

- **(theory)** functional PL or so based on type theory and lambda calculus, Church's approach of understanding computability.
  — not easy to define the resource consumption… hence no good model for complexity theory
  — easy in programming; a lot of useful properties can be expressed conveniently; almost direct connection with compiler implementation.

- **(theory)** Logic is the subject of identifying true statements in a language of which you only know a few words. Different logics study different language fragments:
  — propositional logic: and, or, not, if … then
  — temporal logic: propositional logic + today, tomorrow, eventually, never, …

- **(Mind-set)** Languages affect the way we think.
  — many constructs or so might be hard to imagine with quantum circuit abstraction
  — famous classical example: merge-sort and recursion. How about quantum recursion?

- **(automation)** PL or formal methods (FM) can help us automate many tedious tasks and provide instance-specific solutions
  — many detailed optimization and verification can be conducted with better correctness guarantees
  — the only way to scale up these tasks

- **(computational thinking)** developing abstractions is at the heart of developing PL/FM techniques

The Role of Programming Languages

Like the role of PL played for any other computing models, many *similar first-principle questions* can be asked in the context of quantum computing as well!

The Role of Programming Languages

Like the role of PL played for any other computing models, many *similar first-principle questions* can be asked in the context of quantum computing as well!

But of course, quantum computing model demonstrates some *fundamental differences and unique needs*, which requires new techniques to deal with.

The Role of Programming Languages

Like the role of PL played for any other computing models, many *similar first-principle questions* can be asked in the context of quantum computing as well!

But of course, quantum computing model demonstrates some *fundamental differences and unique needs*, which requires new techniques to deal with.

Disclaimer: perspectives and claims are potentially limited or biased by personal knowledge.

How to Program Q. Applications, Debug, and Verify Correctness?

How to Develop Software for Q. Computing, e.g., compiler, system?

How to Design and Implement Architecture for Quantum Computing?

How to Handle Quantum Security Issues in Design&Implementation?

How to Scale and Automate the Design of Quantum Hardware ?

How to Program Q. Applications, Debug, and Verify Correctness?

The natural question with MOST investigation, but still a huge gap!

How to Program Q. Applications, Debug, and Verify Correctness?

The natural question with MOST investigation, but still a huge gap!

THEORY: quantum lambda-calculus, functional quantum PL, q. while language semantics in various pictures, q. Hoare logic and verification, …

How to Program Q. Applications, Debug, and Verify Correctness?

The natural question with MOST investigation, but still a huge gap!

THEORY: **quantum lambda-calculus, functional quantum PL, q. while language semantics in various pictures, q. Hoare logic and verification, …**

LANGUAGES: **Quipper** (embedded in Haskel), **Scaffold** (based on LLVM), **Q#** (based on F#, MSR), **QWIRE/SQIR** (embedded in Coq), **SILQ**, … **<- academia**

python-lib **Qiskit** (IBM), **Cirq** (Google), **Forrest** (Rigetti), **Braket** (AWS), **<- industry**

How to Program Q. Applications, Debug, and Verify Correctness?

The natural question with MOST investigation, but still a huge gap!

THEORY: **quantum lambda-calculus, functional quantum PL, q. while language semantics in various pictures, q. Hoare logic and verification, …**

LANGUAGES: **Quipper (**embedded in Haskel**), Scaffold (**based on LLVM**), Q# (**based on F#, MSR**),**
QWIRE/SQIR (embedded in Coq**), SILQ, …** **<- academia**
python-lib **Qiskit (**IBM**), Cirq (**Google**), Forrest (**Rigetti**), Braket (**AWS**),** **<- industry**

Gap: (1) too-low-level-abstraction: very hard to write **complex** programs

How to Program Q. Applications, Debug, and Verify Correctness?

The natural question with MOST investigation, but still a huge gap!

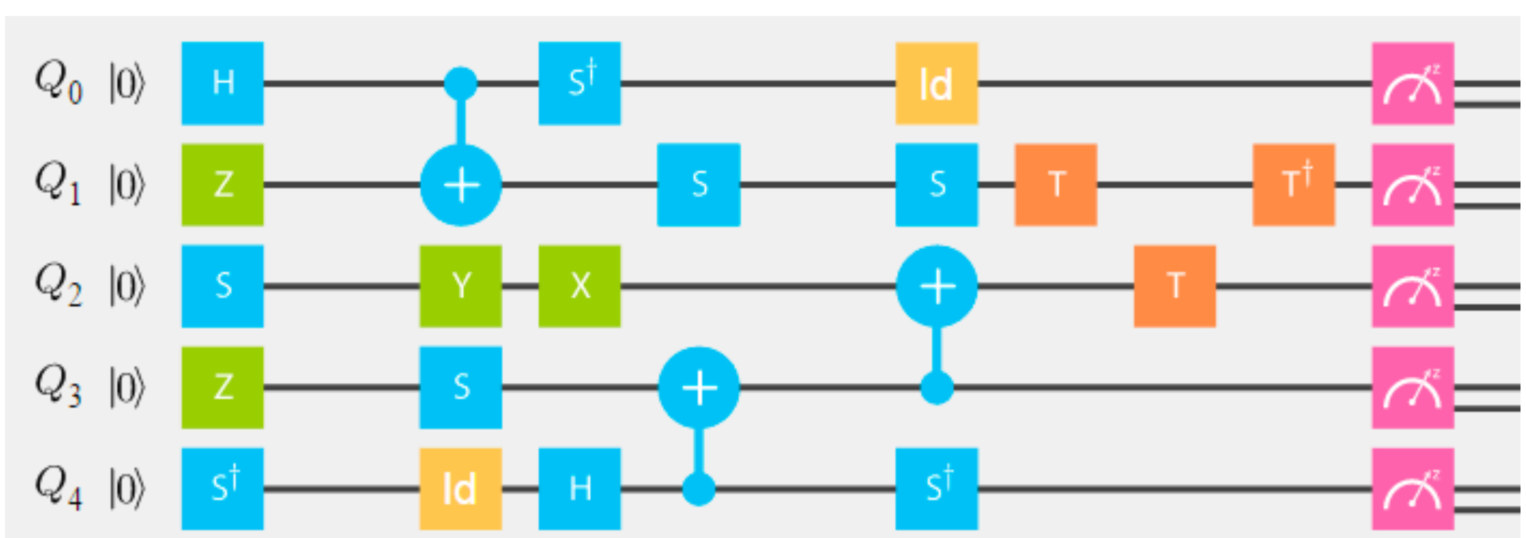
THEORY: **quantum lambda-calculus, functional quantum PL, q. while language semantics in various pictures, q. Hoare logic and verification, …**

LANGUAGES: **Quipper** (embedded in Haskel), **Scaffold** (based on LLVM), **Q#** (based on F#, MSR), **QWIRE/SQIR** (embedded in Coq), **SILQ, …** **<- academia**

python-lib **Qiskit** (IBM), **Cirq** (Google), **Forrest** (Rigetti), **Braket** (AWS), **<- industry**

Gap: (1) too-low-level-abstraction: very hard to write **complex** programs

(2) lack of scalable verification: very hard to write **correct** programs



Verifying the circuit by observation …. not scalable …

How to Program Q. Applications, Debug, and Verify Correctness?

The natural question with MOST investigation, but still a huge gap!

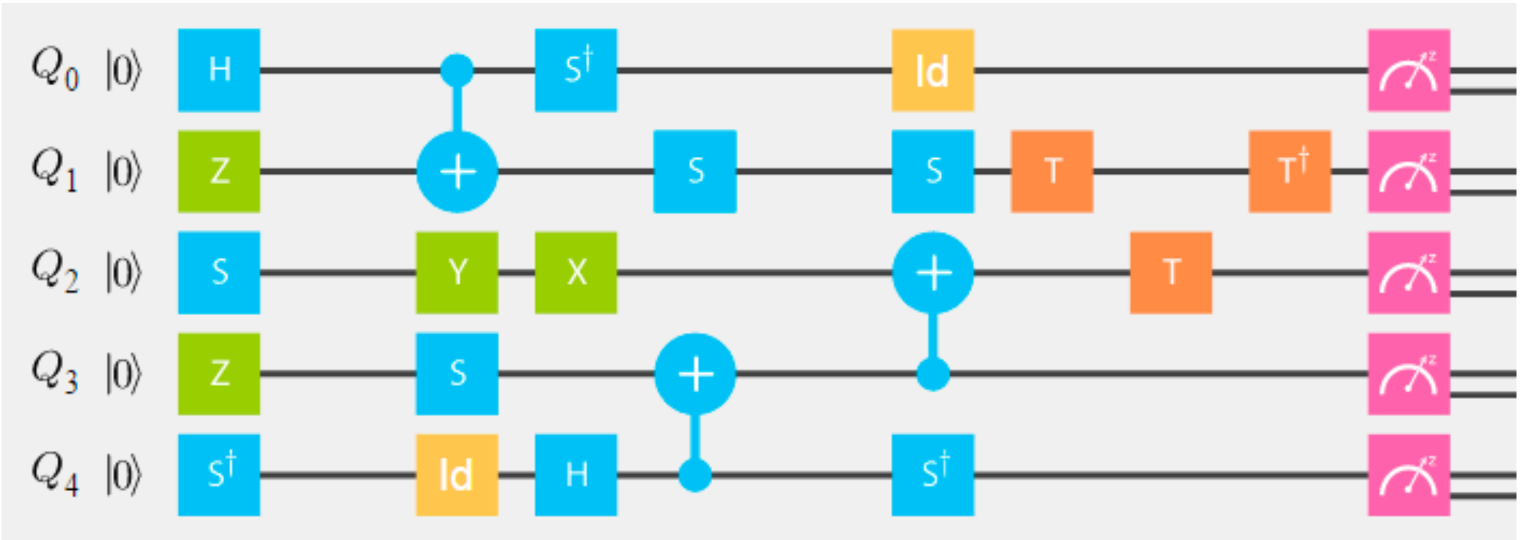
THEORY: **quantum lambda-calculus, functional quantum PL, q. while language semantics in various pictures, q. Hoare logic and verification, …**

LANGUAGES: **Quipper** (embedded in Haskel), **Scaffold** (based on LLVM), **Q#** (based on F#, MSR), **QWIRE/SQIR** (embedded in Coq), **SILQ**, … **<- academia**

python-lib **Qiskit** (IBM), **Cirq** (Google), **Forrest** (Rigetti), **Braket** (AWS), **<- industry**

Gap: (1) too-low-level-abstraction: very hard to write **complex** programs

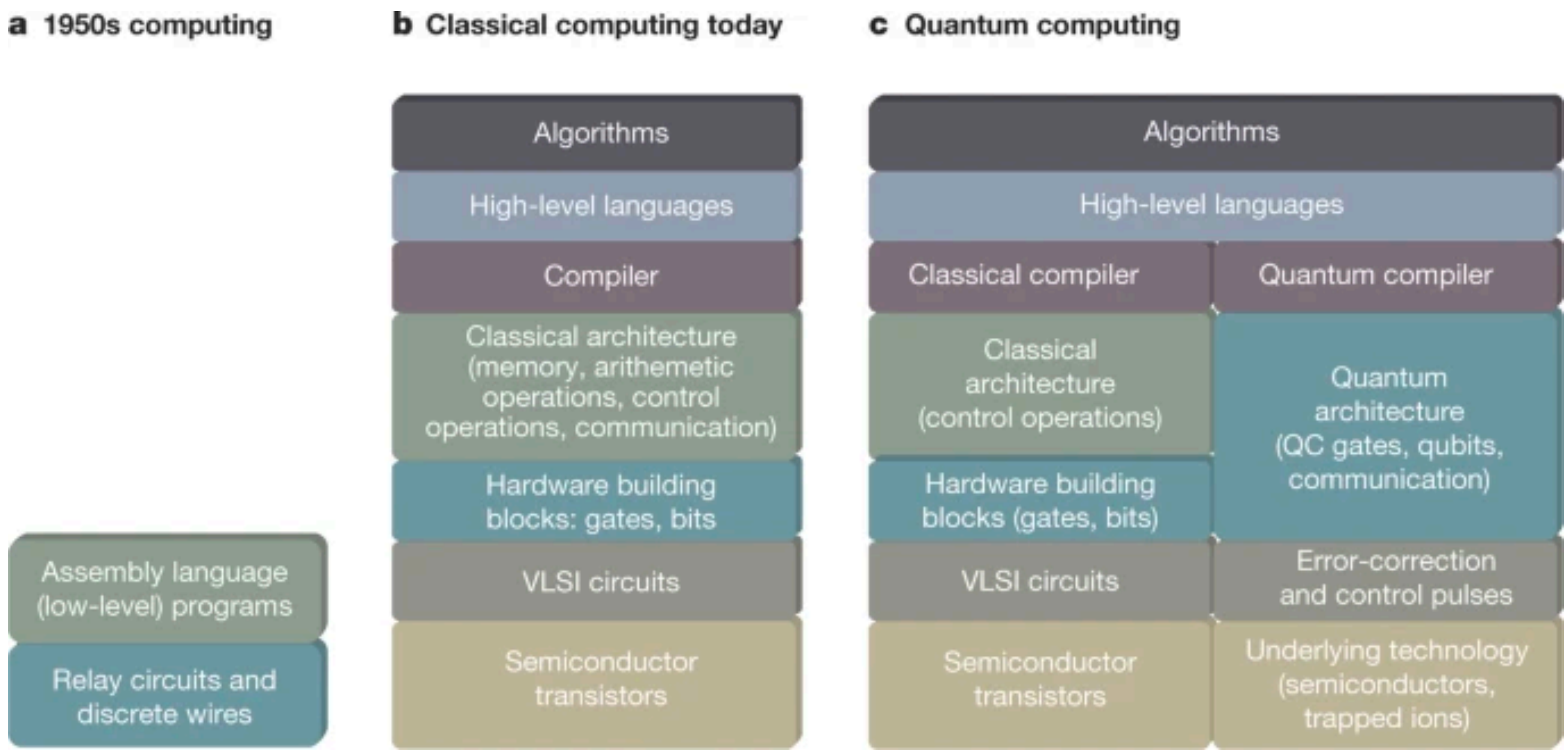
(2) lack of scalable verification: very hard to write **correct** programs



Verifying the circuit by observation …. not scalable …

(3) lack of many desirable analyses, automation, & optimization: a lot of burdens on the programmers

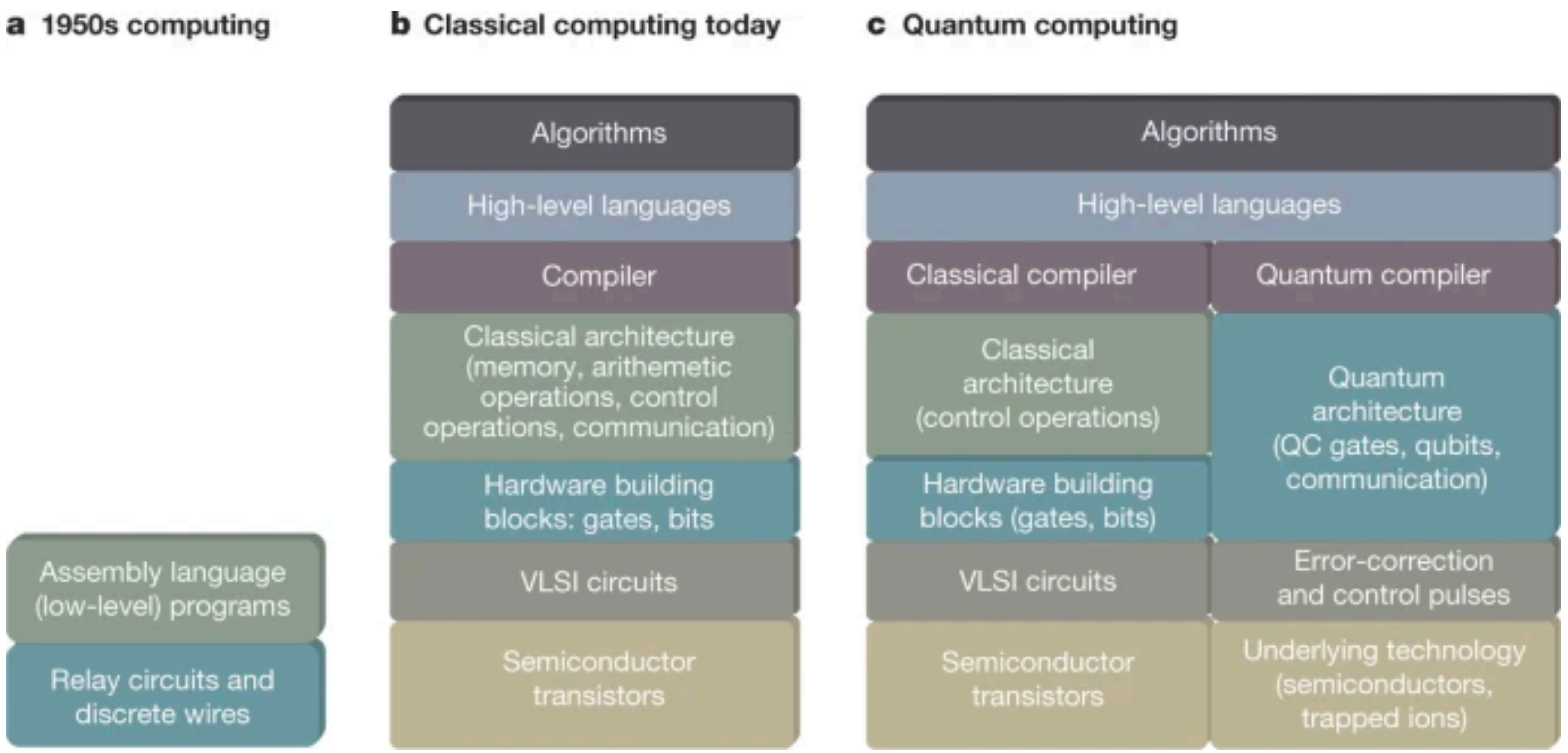
How to Develop Software for Q. Computing, e.g., compiler, system?



F. Chong, D. Franklin, M. Martonosi, Nature 549, 180

Large Design Space for System Software for Quantum Computers.

How to Develop Software for Q. Computing, e.g., compiler, system?



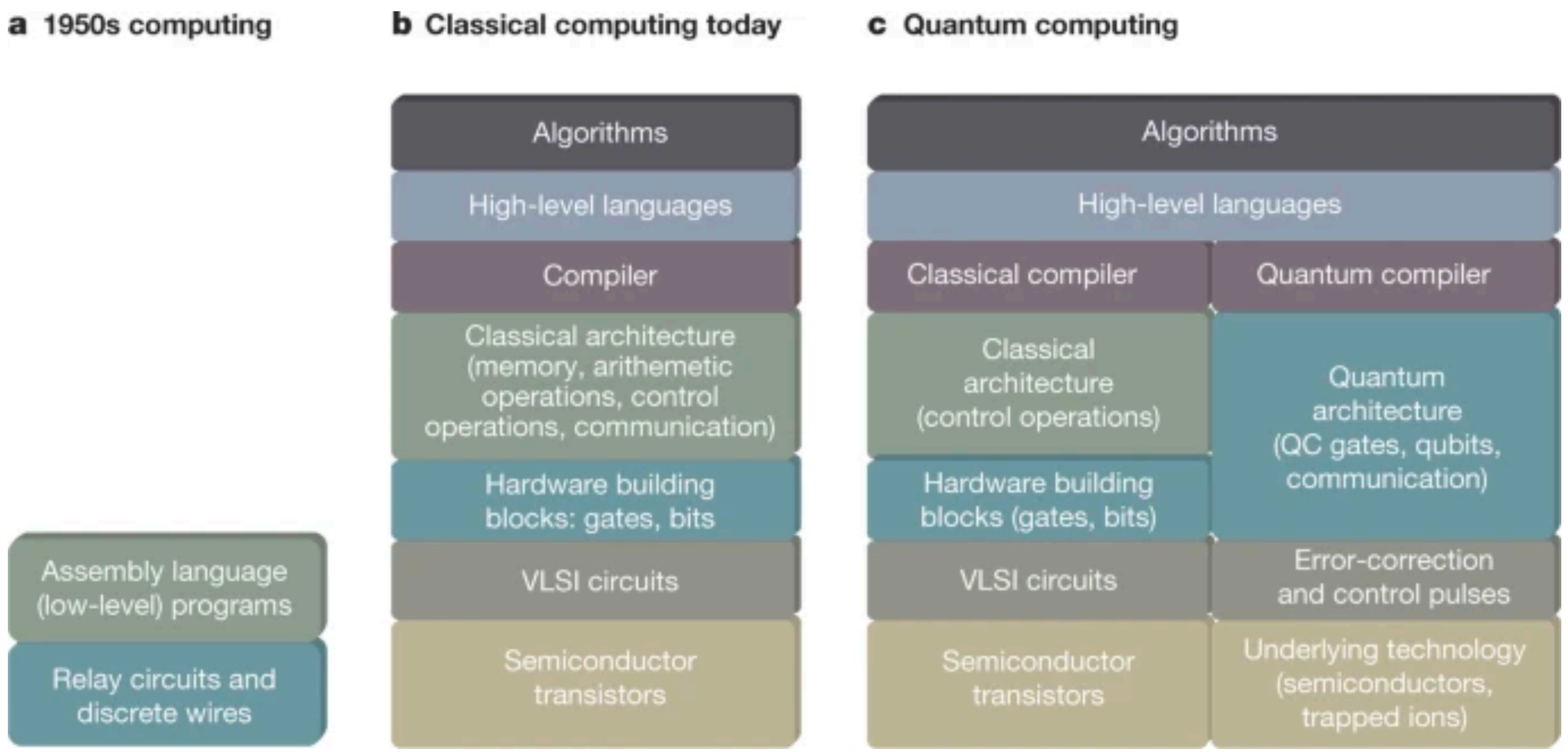
F. Chong, D. Franklin, M. Martonosi, Nature 549, 180

Large Design Space for System Software for Quantum Computers.

High-Assurance Software Tool-chain both desirable and challenging.

- standard software assurance techniques, e.g., black-box / unit test, expensive in q.
- quantum mechanics prohibits certain testing, e.g., assertions

How to Develop Software for Q. Computing, e.g., compiler, system?



F. Chong, D. Franklin, M. Martonosi, Nature 549, 180

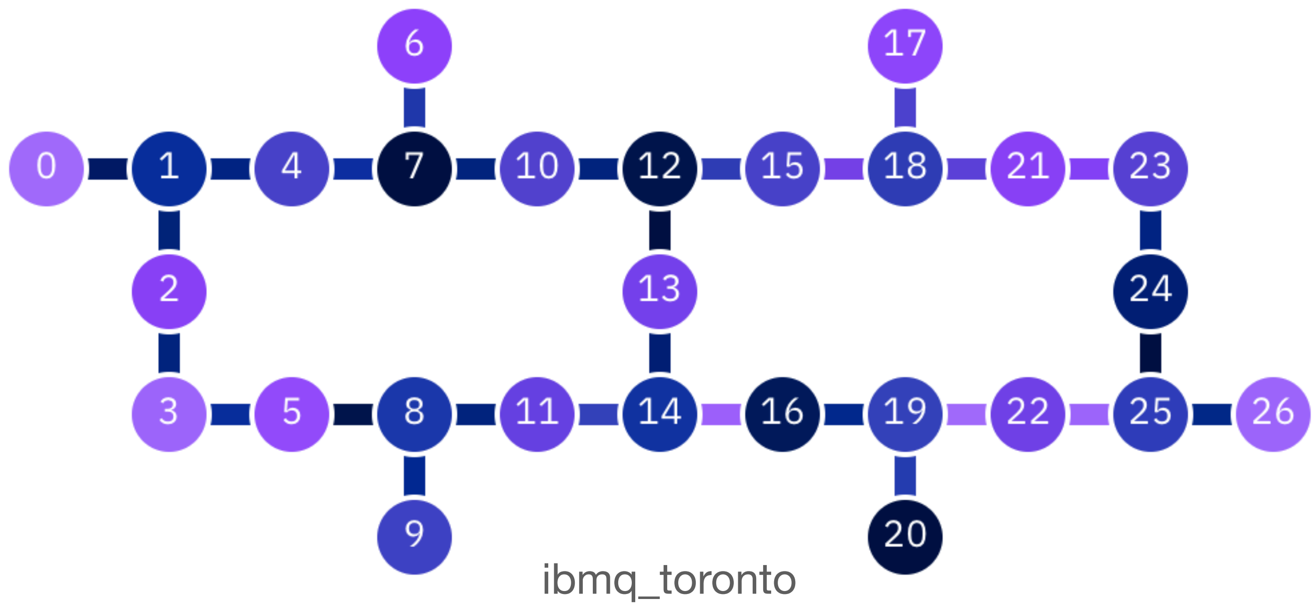
Large Design Space for System Software for Quantum Computers.

High-Assurance Software Tool-chain both desirable and challenging.

- standard software assurance techniques, e.g., black-box / unit test, expensive in q.
- quantum mechanics prohibits certain testing, e.g., assertions

A possible solution : fully certified software, e.g., VOQC (POPL 2021)

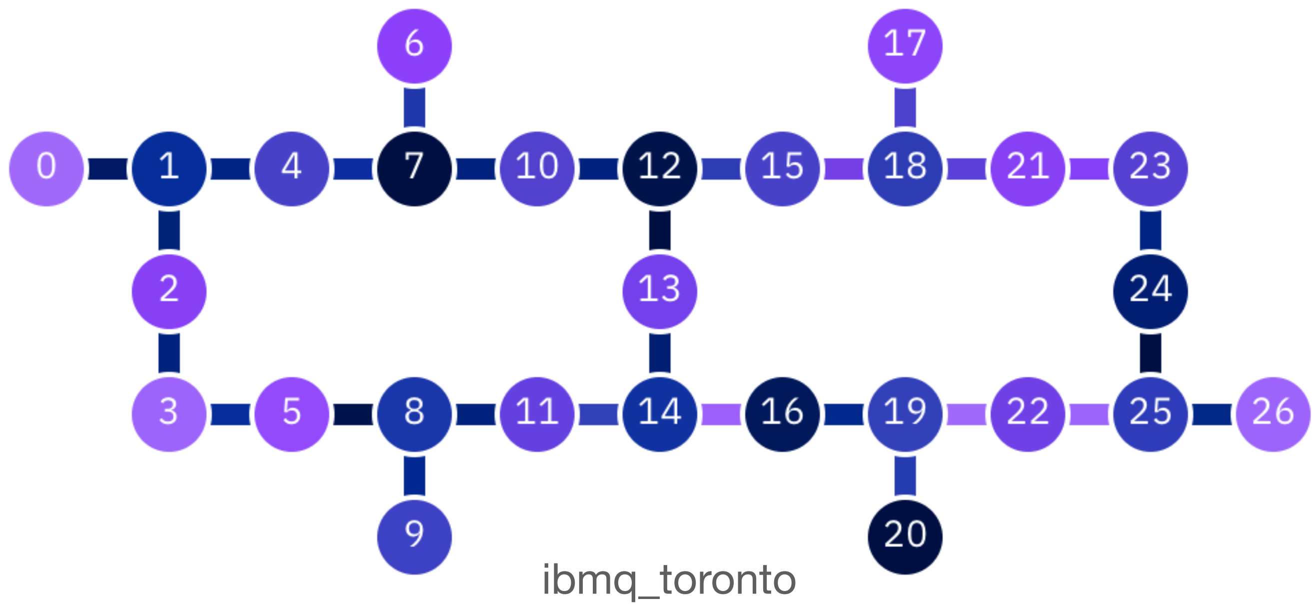
How to Design and Implement **Architecture** for Quantum Computing?



Mapping, Error Mitigation, …

approximate computing

How to Design and Implement Architecture for Quantum Computing?

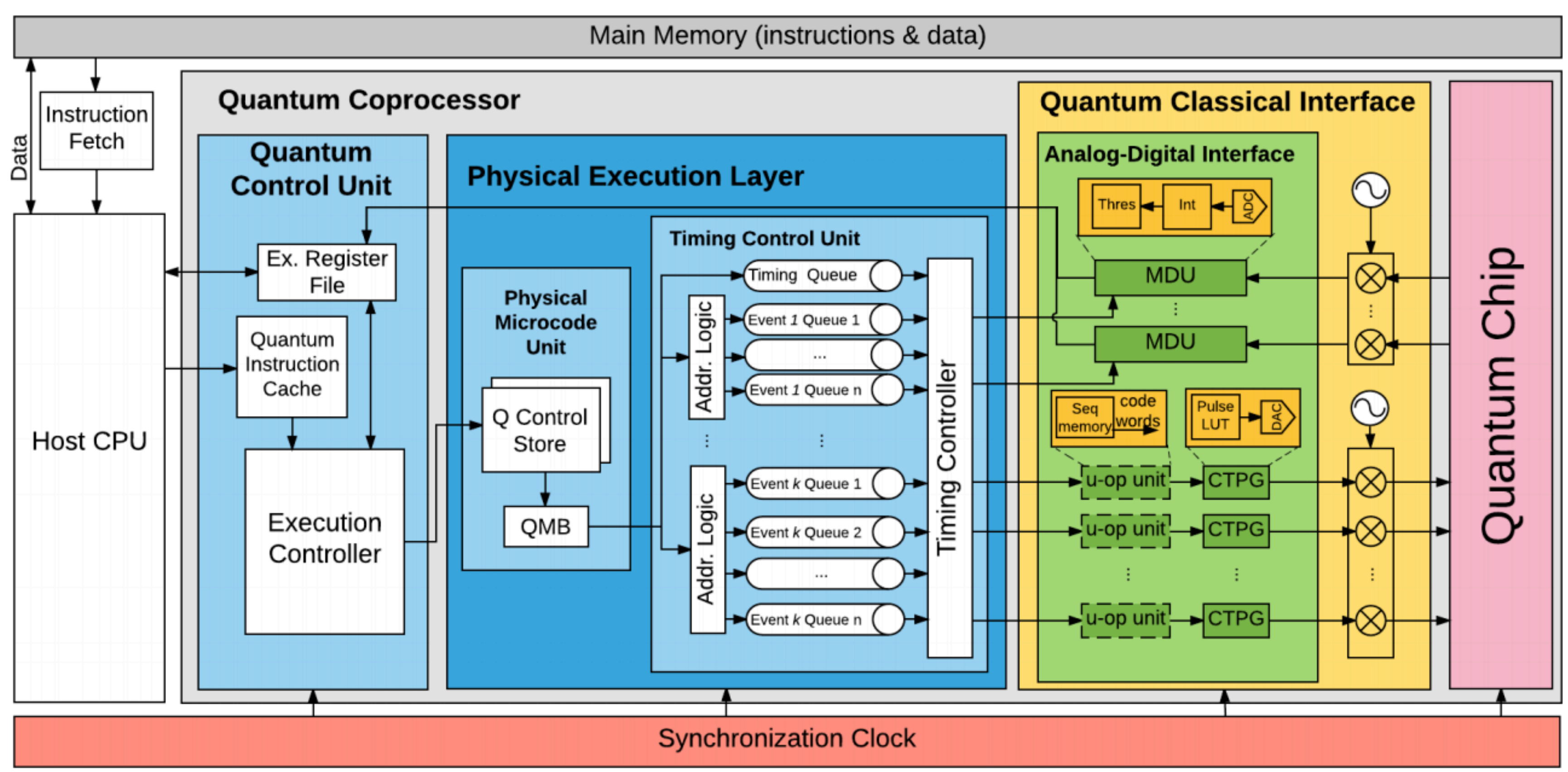


Mapping, Error Mitigation, …

approximate computing

A lot of controlling operations need to be located close to quantum chips for small responsive time.

ISA + Fast Compilation



X. Fu et al MICRO 17

How to Handle Quantum Security Issues in Design and Implementation?

Verification of Quantum Cryptography:

Relational Quantum Hoare Logic (Unruh; Barthe et al.)

How to Handle Quantum Security Issues in Design and Implementation?

Verification of Quantum Cryptography:

Relational Quantum Hoare Logic (Unruh; Barthe et al.)

Quantum Cryptanalysis:

Resource estimation of Complex Quantum Attack Programs

How to Handle Quantum Security Issues in Design and Implementation?

Verification of Quantum Cryptography:

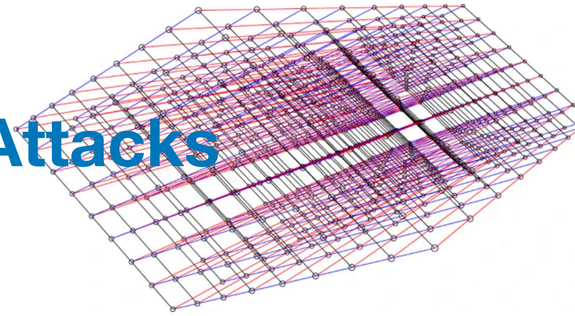
Relational Quantum Hoare Logic (Unruh; Barthe et al.)

Quantum Cryptanalysis:

Resource estimation of Complex Quantum Attack Programs

Post-Quantum Cryptography:

Classical Cryptographic Systems Resilient to Quantum Attacks

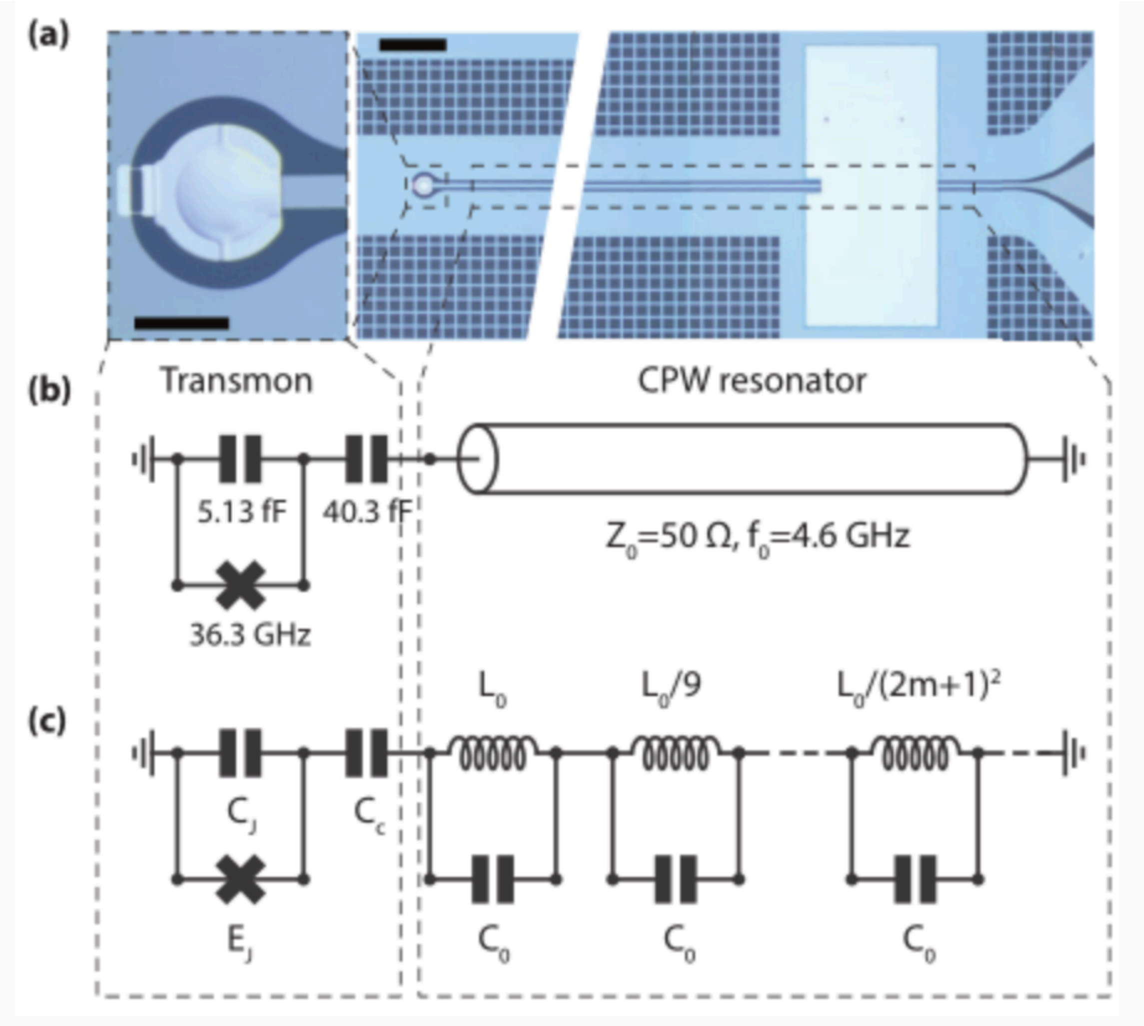


For Classical Cryptographic Systems

(1) Identify their post-quantum security
(2) automate the procedure to upgrade its post-quantum security
(3) formal post-quantum security proofs

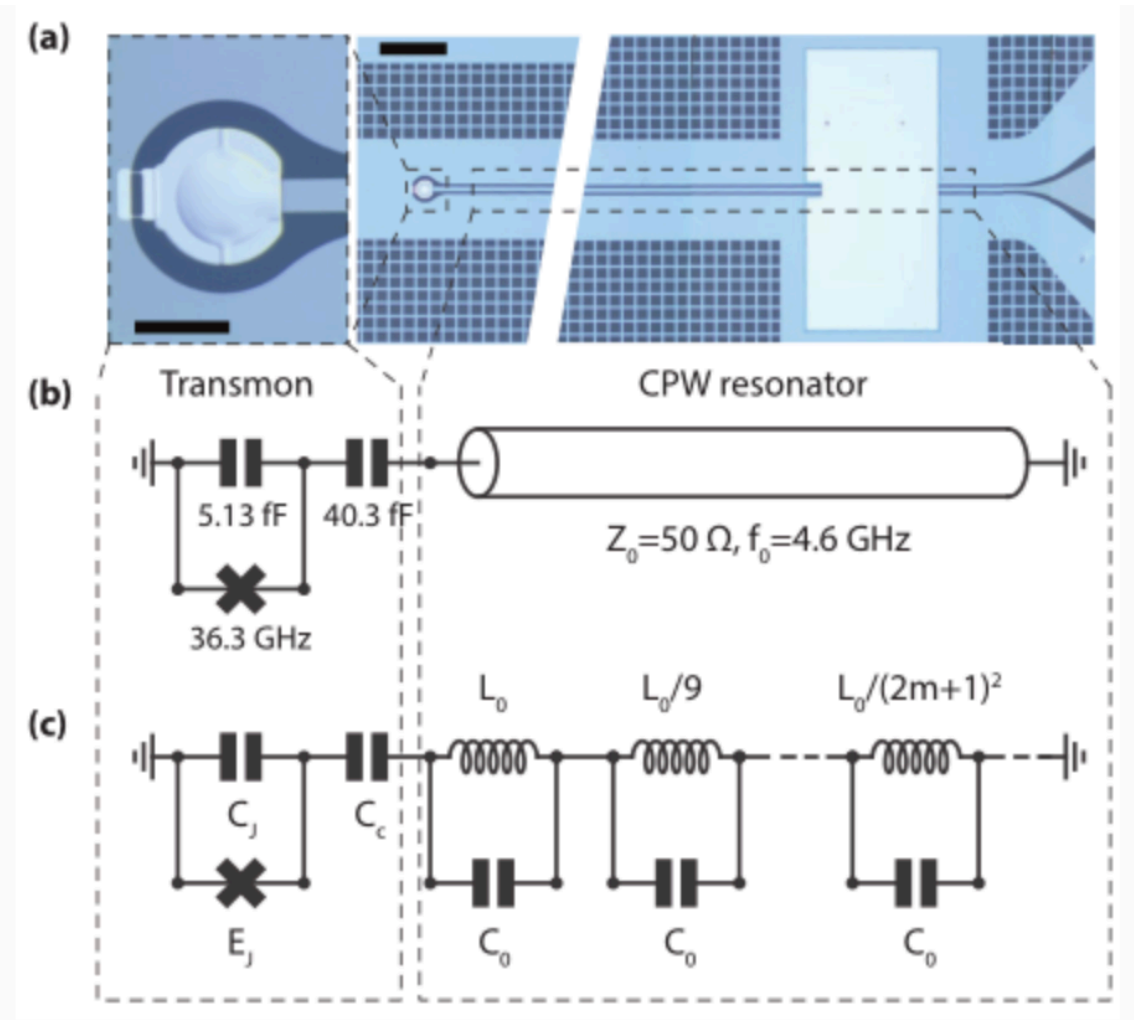
Formally generated security analysis will provide not only efficient and high assurance proofs that can replace the tedious and error-prone analysis for experts, but also independently verifiable proofs that can be used by security practitioners without much quantum knowledge.

How to Scale and Automate the Design of Quantum Hardware ?

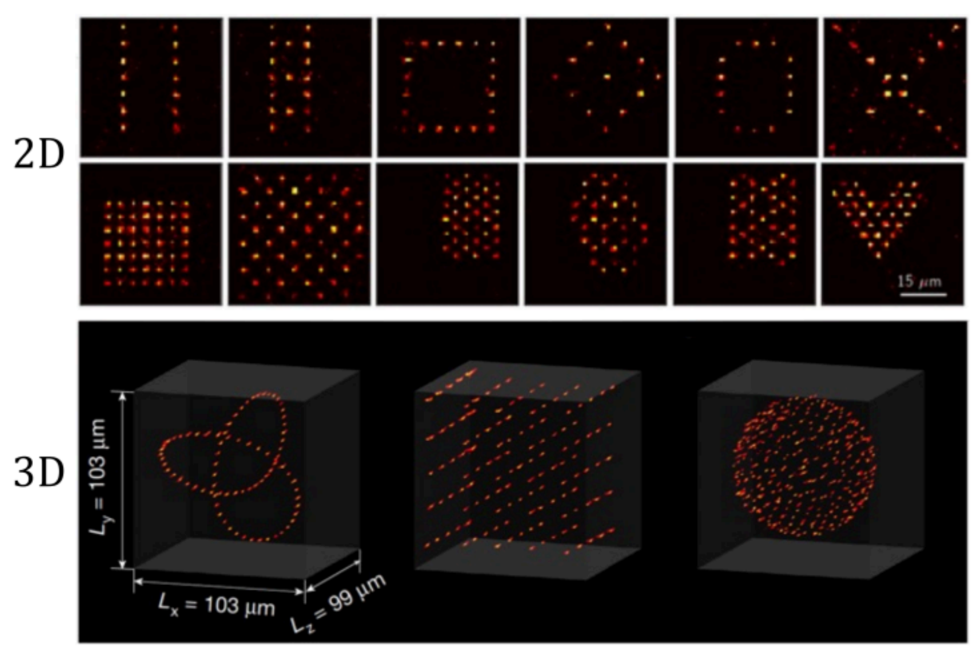


Superconducting Credit: arXiv:1704.06208

How to Scale and Automate the Design of Quantum Hardware ?

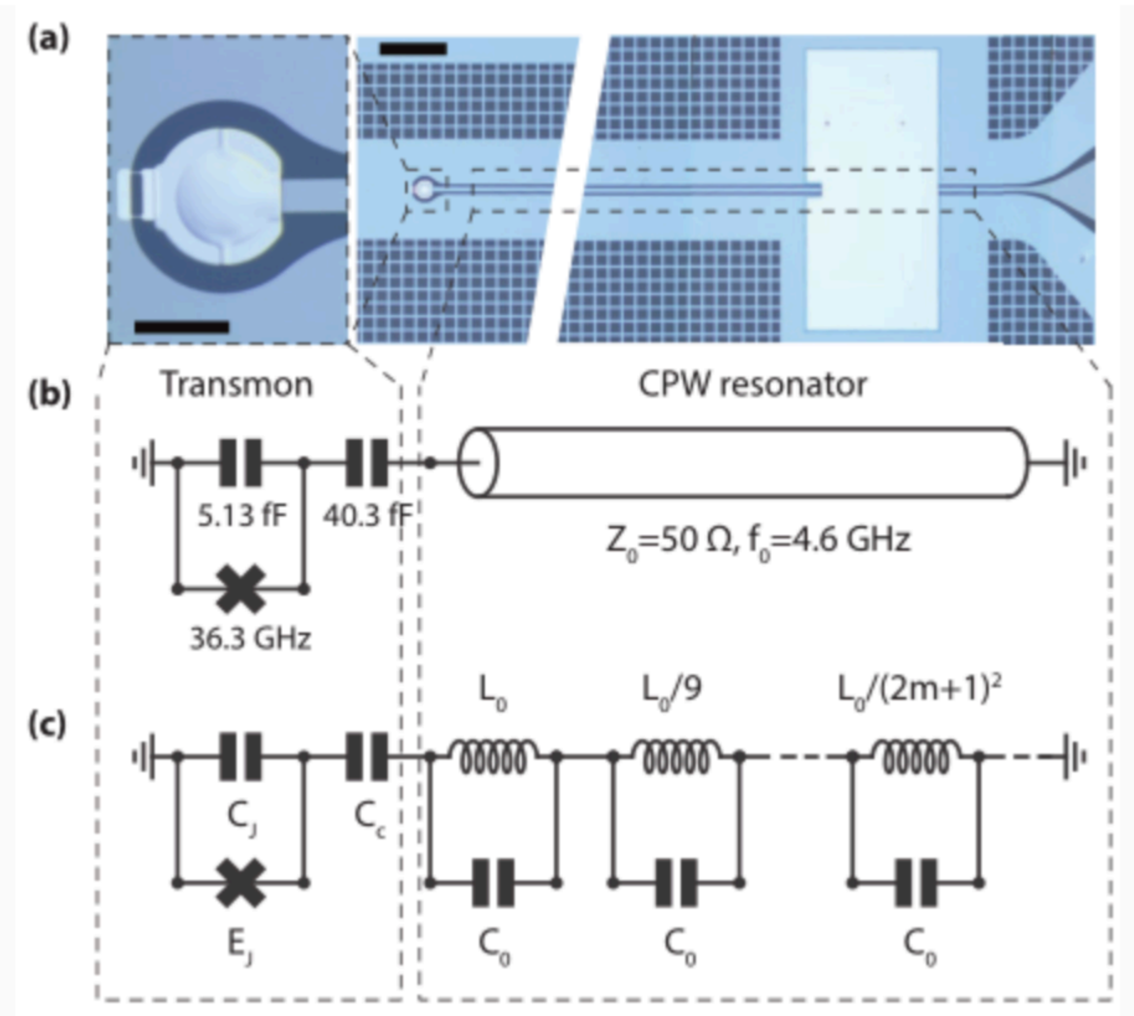


Superconducting Credit: arXiv:1704.06208

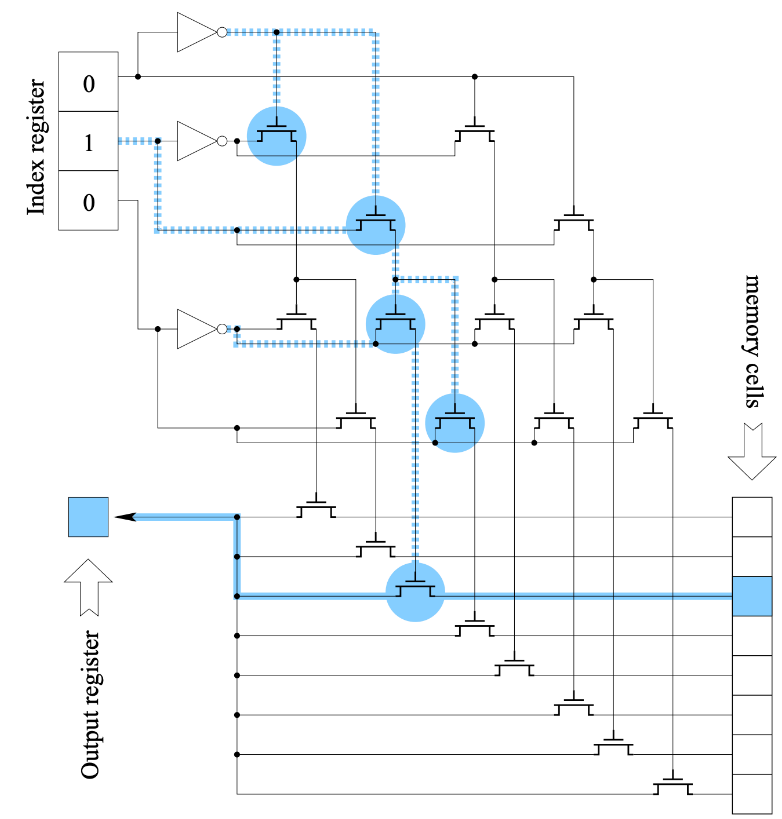


Neutral Atoms Credit: arXiv:2006.12326

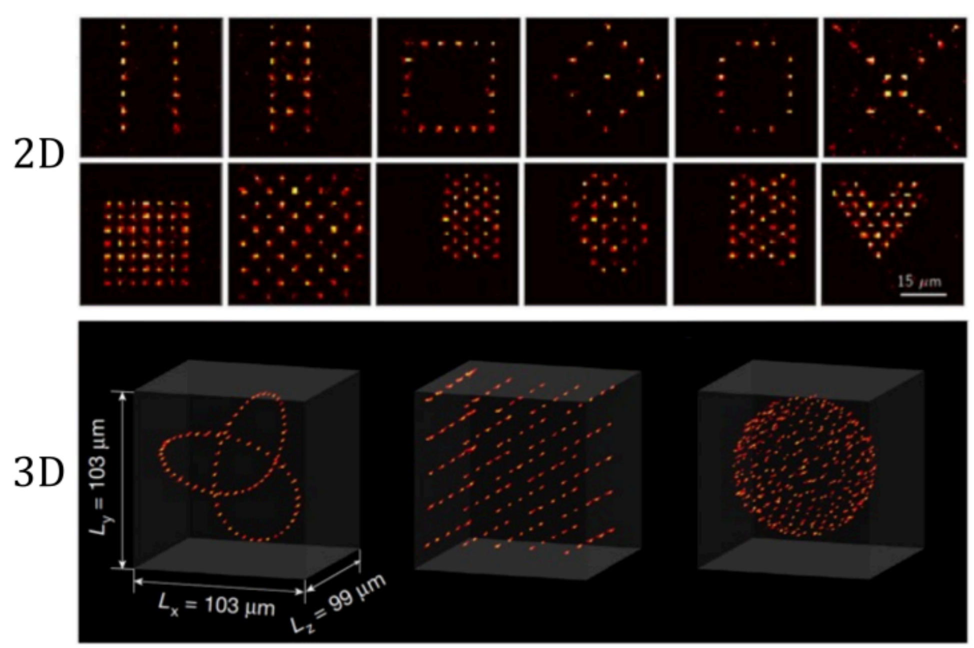
How to Scale and Automate the Design of Quantum Hardware ?



Superconducting Credit: arXiv:1704.06208

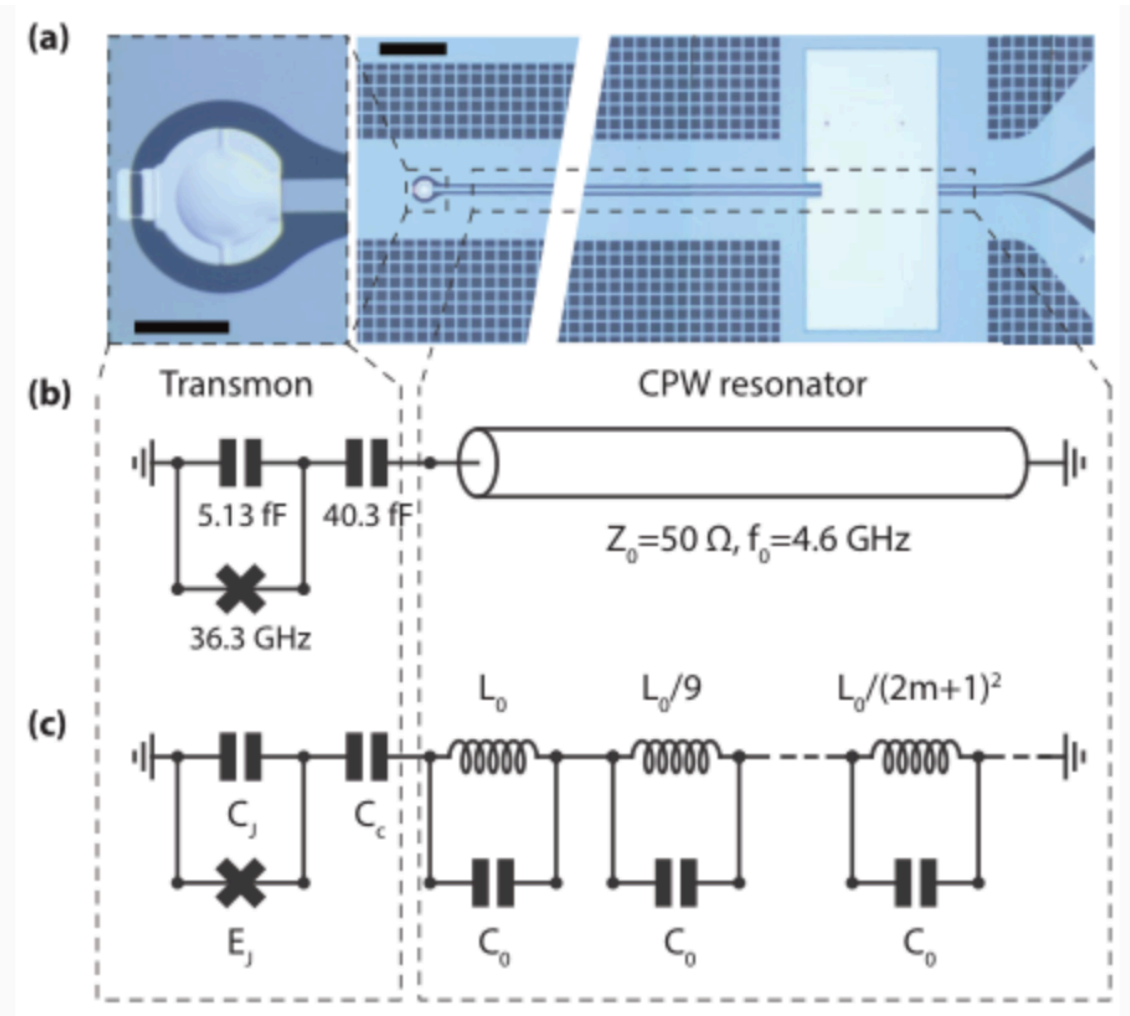


QRAM Architecture
Credit: ArXiv 0807.4994

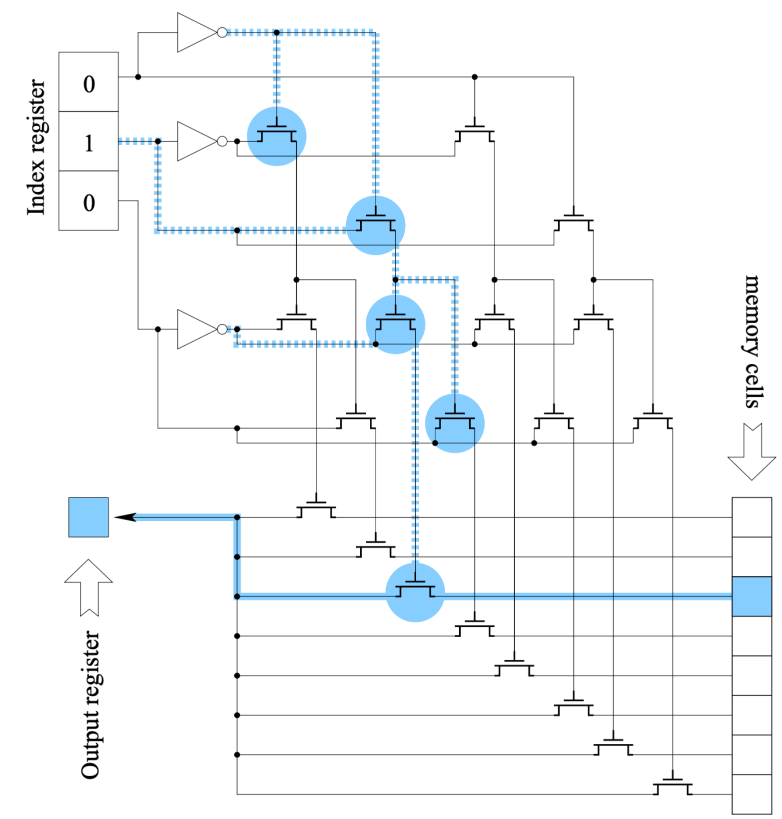


Neutral Atoms Credit: arXiv:2006.12326

How to Scale and Automate the Design of Quantum Hardware ?

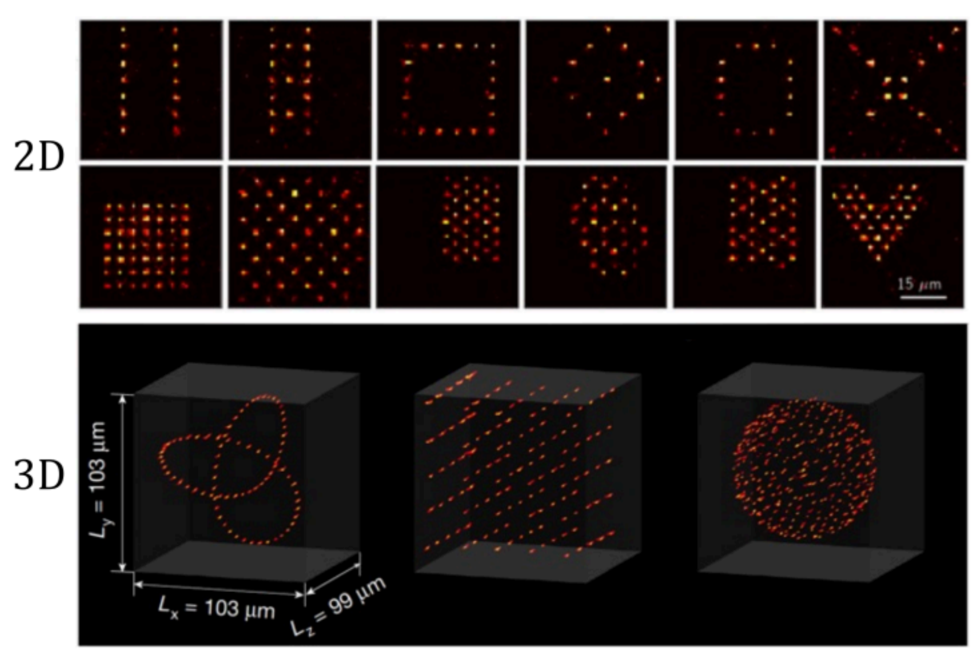


Superconducting Credit: arXiv:1704.06208



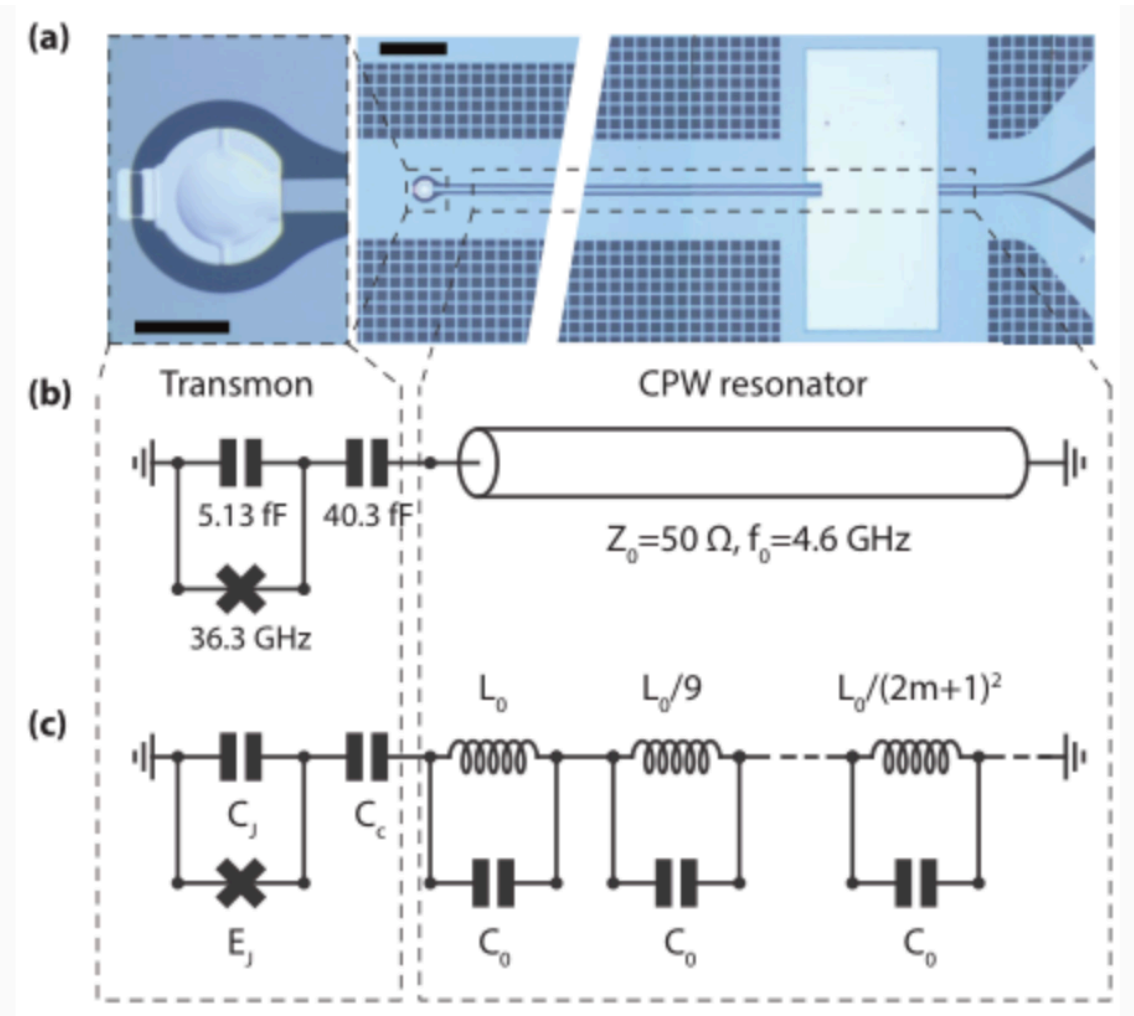
QRAM Architecture
Credit: ArXiv 0807.4994

Demonstrate A Lot of Design Choices
Hard to Scale without Automatic Tools

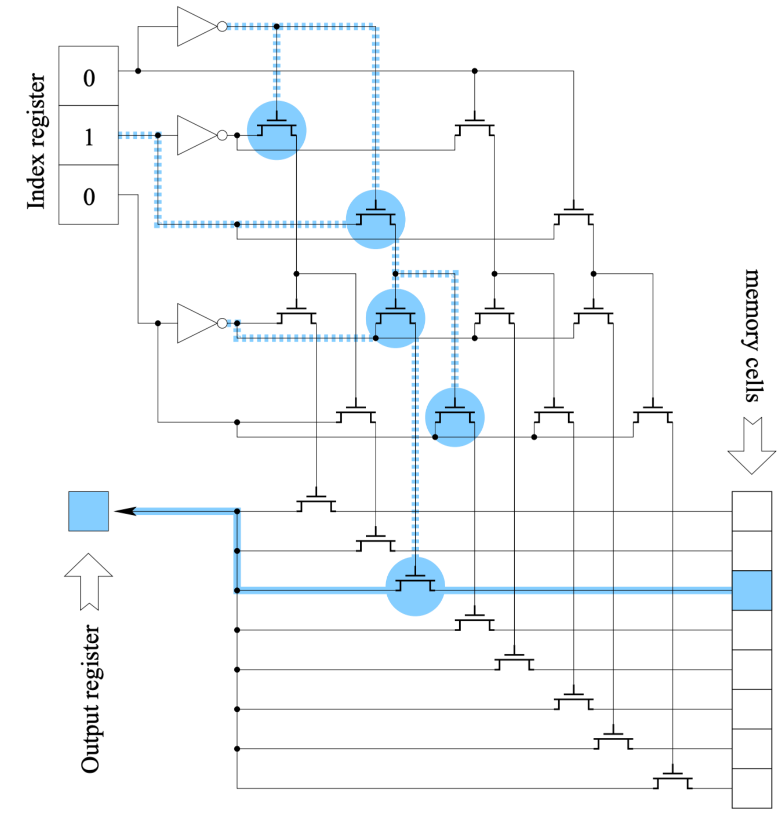


Neutral Atoms Credit: arXiv:2006.12326

How to Scale and Automate the Design of Quantum Hardware ?

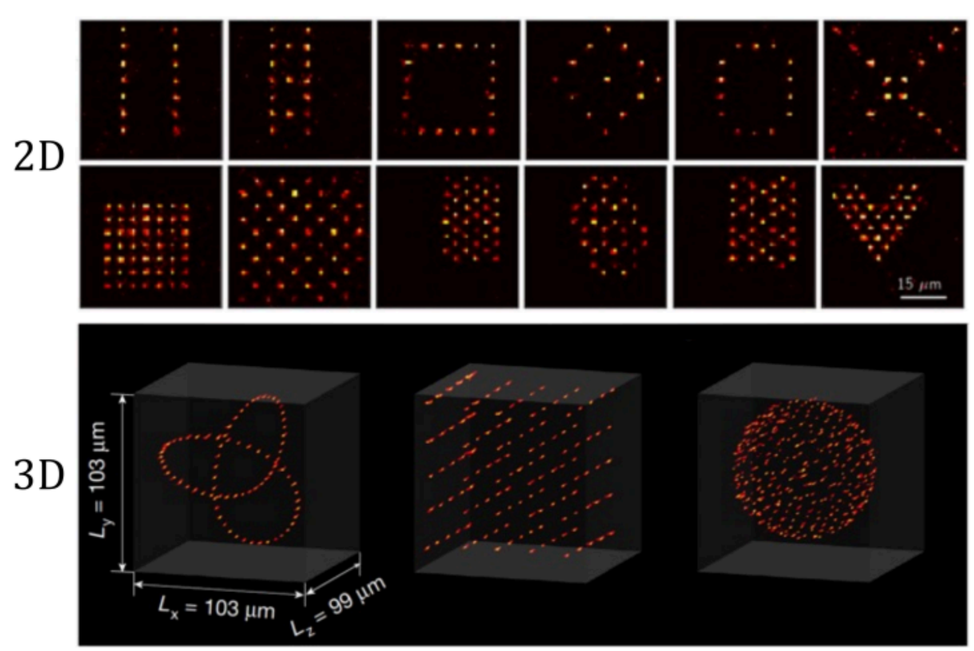


Superconducting Credit: arXiv:1704.06208



QRAM Architecture
Credit: ArXiv 0807.4994

Demonstrate A Lot of Design Choices
Hard to Scale without Automatic Tools



Neutral Atoms Credit: arXiv:2006.12326

A Golden Age of Hardware Description Languages: Applying Programming Language Techniques to Improve Design Productivity

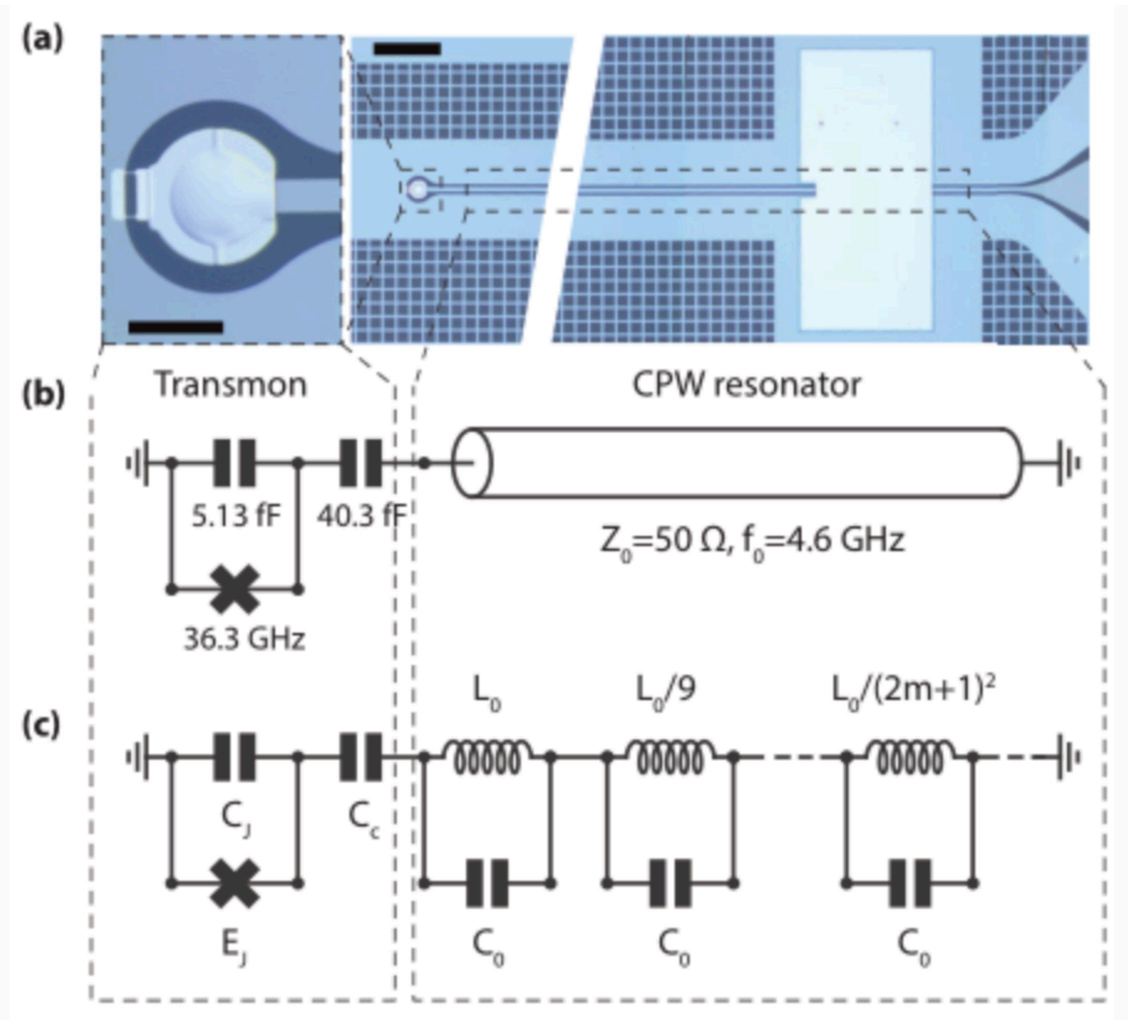
Lenny Truong
Stanford University, USA
lenny@cs.stanford.edu

Pat Hanrahan
Stanford University, USA
hanrahan@cs.stanford.edu

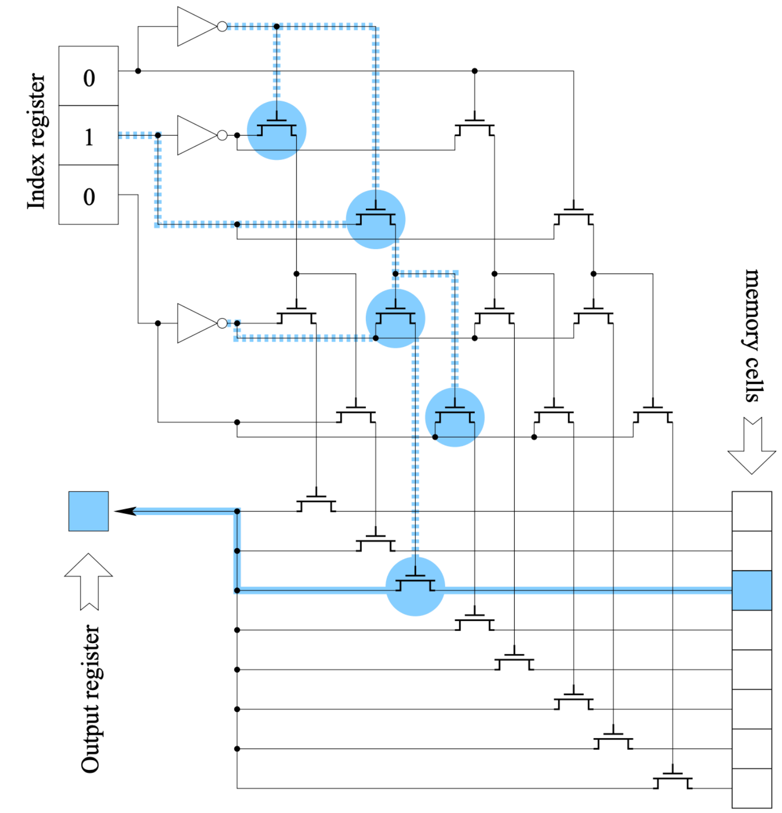
SNAPL 2019

Verilog
HDL

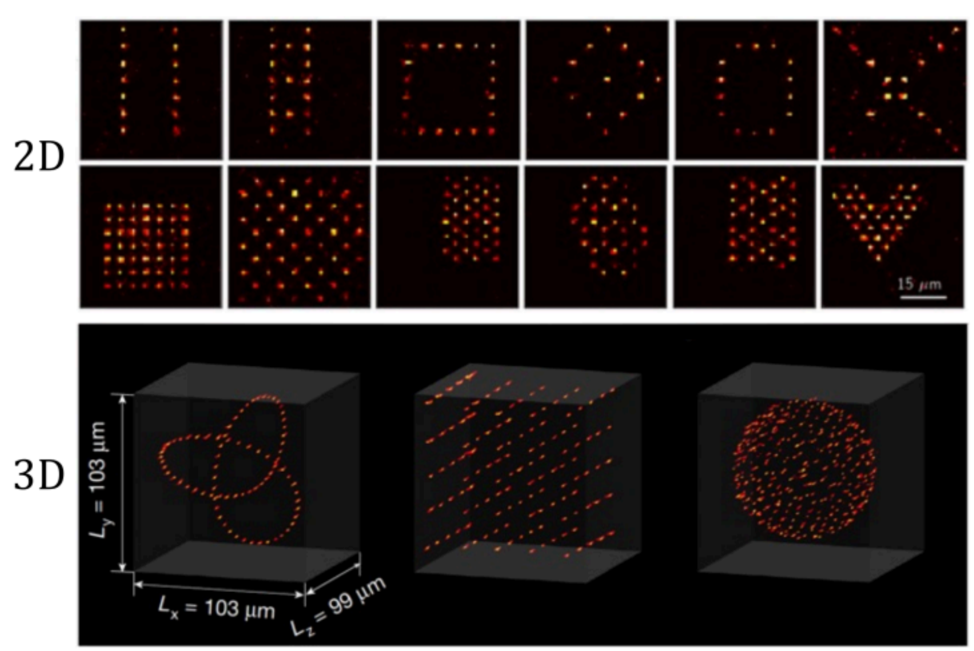
How to Scale and Automate the Design of Quantum Hardware ?



Superconducting Credit: arXiv:1704.06208



QRAM Architecture
Credit: ArXiv 0807.4994



Neutral Atoms Credit: arXiv:2006.12326

Demonstrate A Lot of Design Choices
Hard to Scale without Automatic Tools

A Golden Age of Hardware Description Languages: Applying Programming Language Techniques to Improve Design Productivity

Lenny Truong
Stanford University, USA
lenny@cs.stanford.edu

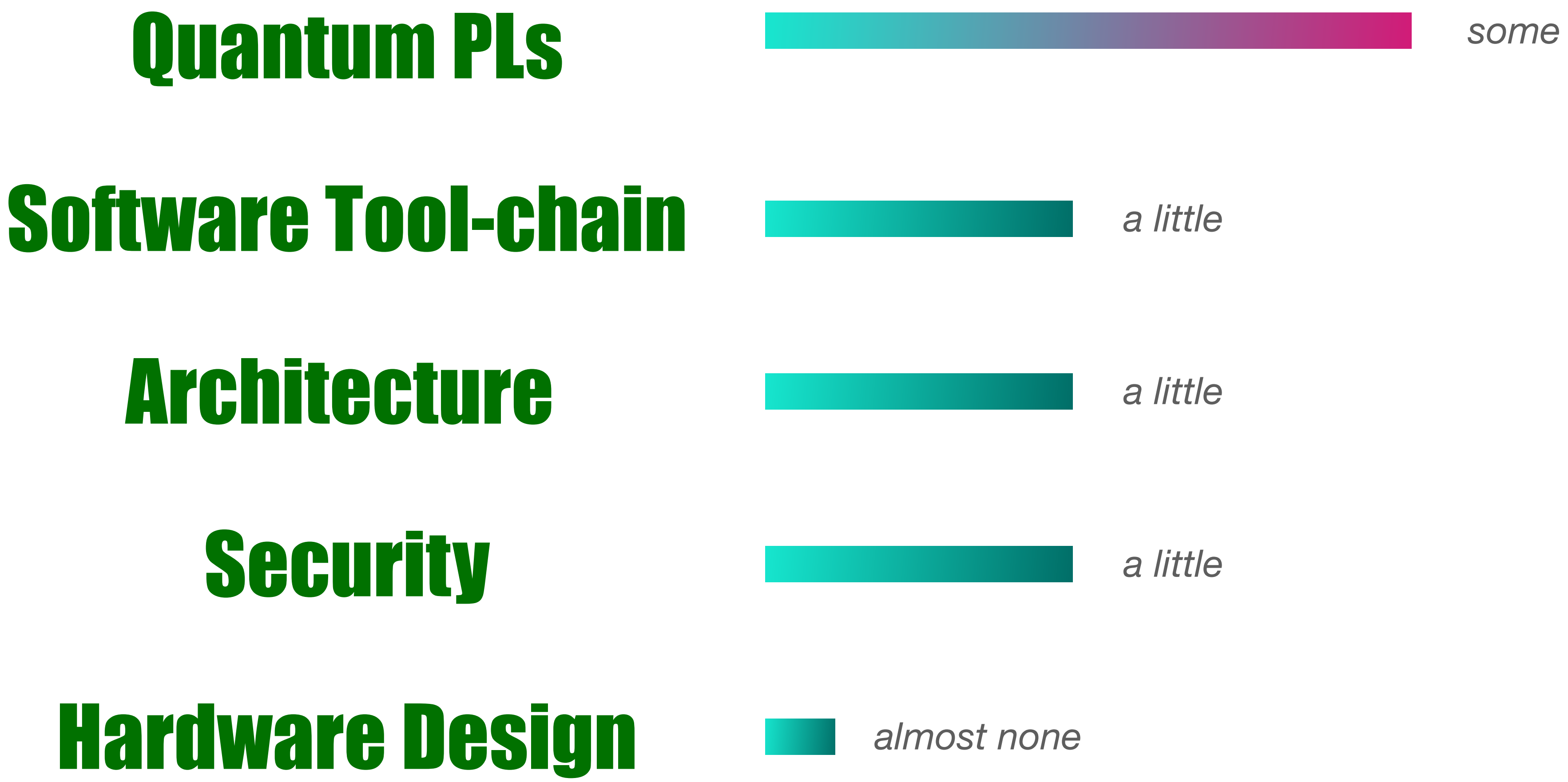
Pat Hanrahan
Stanford University, USA
hanrahan@cs.stanford.edu

SNAPL 2019

Verilog
HDL

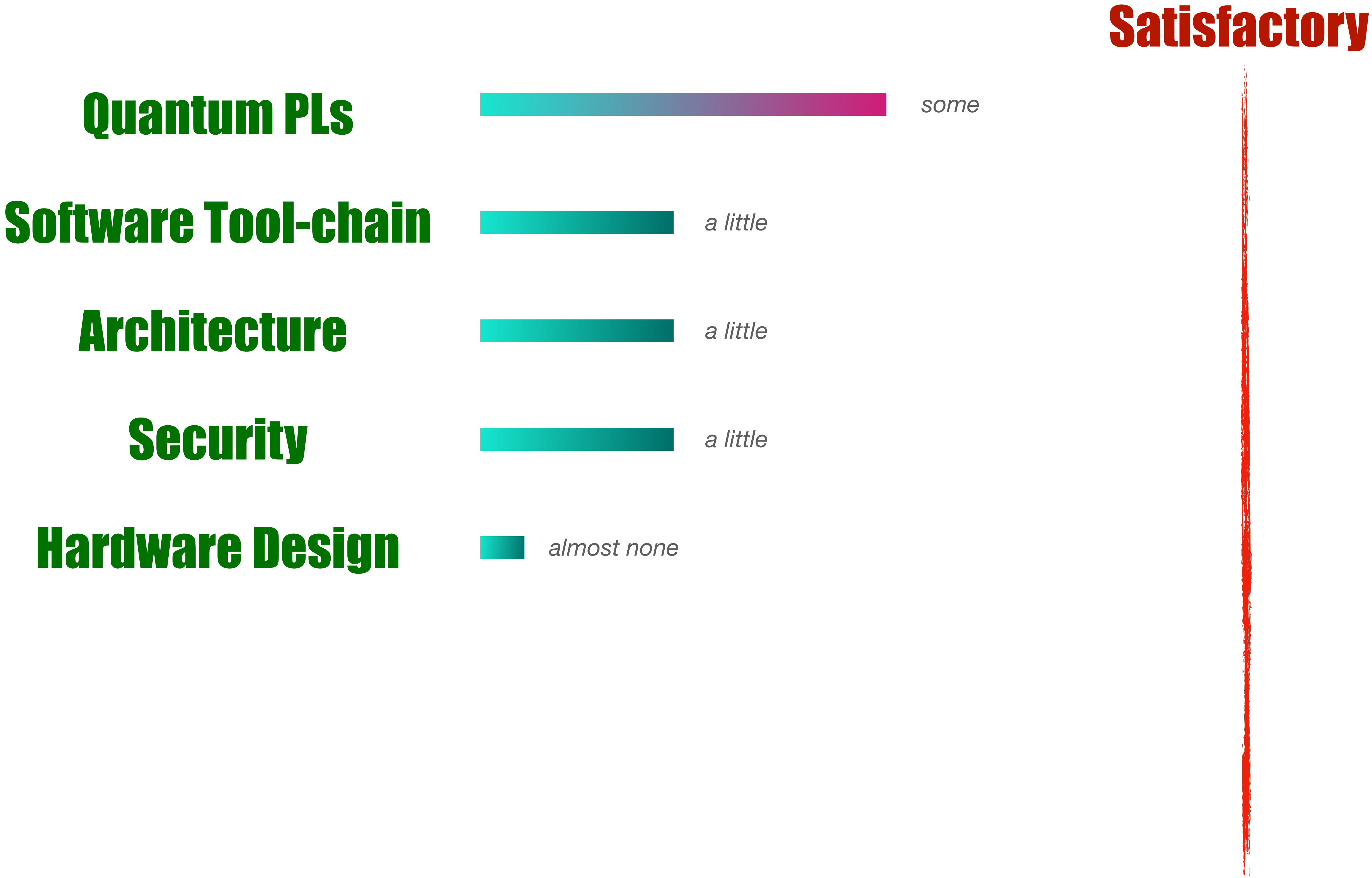
Applies to Quantum Hardware too!

Summary



Quantum PLs
some
Software Tool-chain
a little
Architecture
a little
Security
a little
Hardware Design
almost none

Summary


Satisfactory
Quantum PLs
some
Software Tool-chain
a little
Architecture
a little
Security
a little
Hardware Design
almost none

Summary

Satisfactory

Quantum PLs	*some*
Software Tool-chain	*a little*
Architecture	*a little*
Security	*a little*
Hardware Design	*almost none*

More questions could be asked !

Summary

Satisfactory

Quantum PLs — *some*

Software Tool-chain — *a little*

Architecture — *a little*

Security — *a little*

Hardware Design — *almost none*

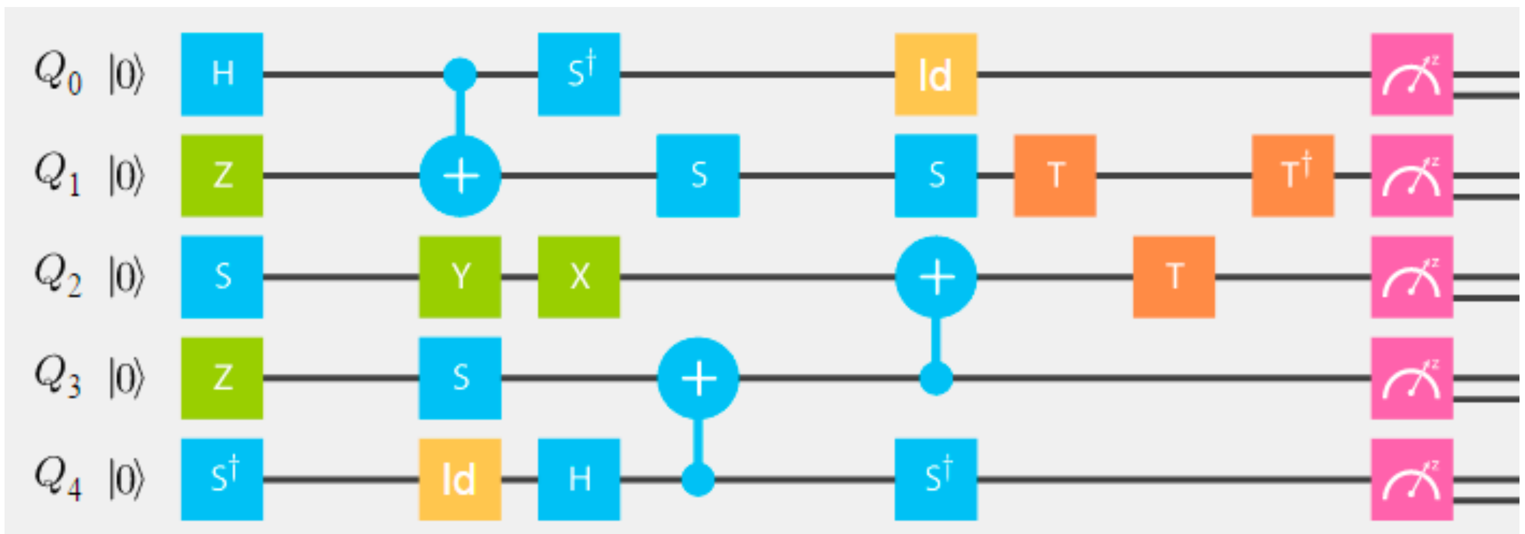
More questions could be asked !

More details will come back in Part III of the tutorial.

Design of Quantum Programming Languages

Gap: (1) too-low-level-abstraction: very hard to write **complex** programs

(2) lack of scalable verification: very hard to write **correct** programs



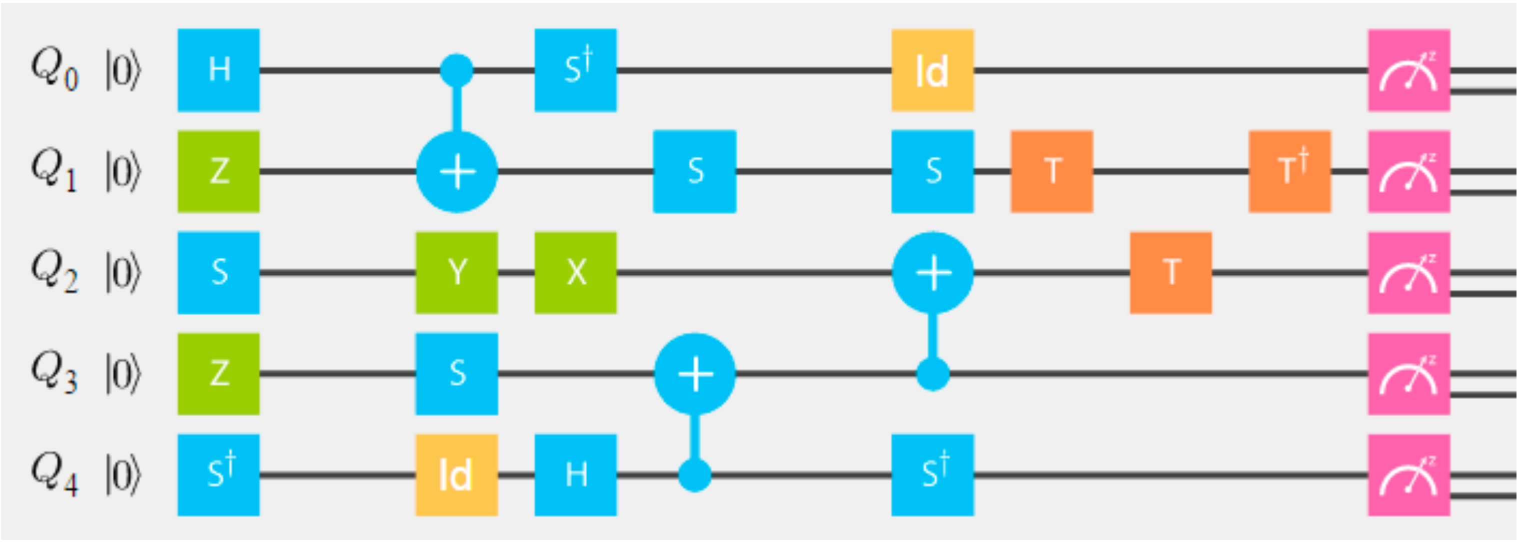
Verifying the circuit
by observation
…. not scalable …

(3) lack of many desirable analyses, automation, & optimization: a lot of burdens on the programmers

Design of Quantum Programming Languages

Gap: (1) too-low-level-abstraction: very hard to write **complex** programs

(2) lack of scalable verification: very hard to write **correct** programs



Verifying the circuit
by observation
…. not scalable …

(3) lack of many desirable analyses, automation, & optimization: a lot of burdens on the programmers

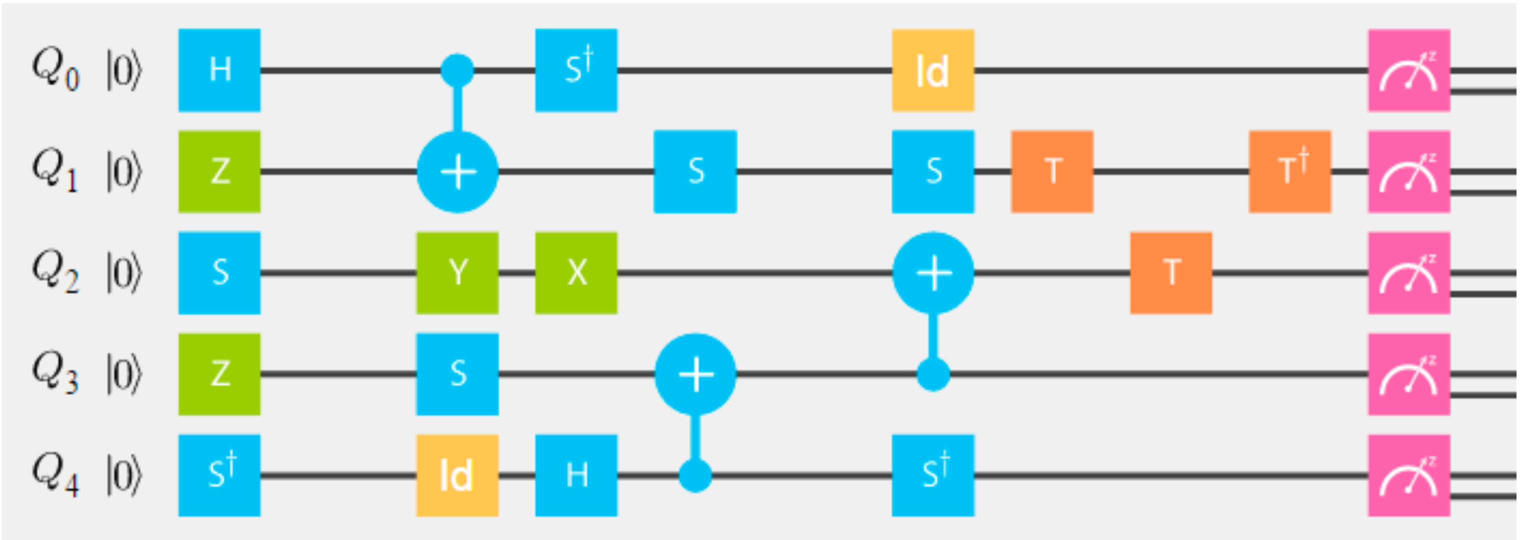
Existing work on type enforced **correctness** in QPLs

No-Cloning: use *linear* types for quantum variables (Quipper, QWIRE)

Design of Quantum Programming Languages

Gap: (1) too-low-level-abstraction: very hard to write **complex** programs

(2) lack of scalable verification: very hard to write **correct** programs



Verifying the circuit
by observation
…. not scalable …

(3) lack of many desirable analyses, automation, & optimization: a lot of burdens on the programmers

Existing work on type enforced **correctness** in QPLs

No-Cloning: use *linear* types for quantum variables (Quipper, QWIRE)

Ancilla: keep track of the scope of ancilla qubits (Quipper)

Design of QPLs: the level of abstraction

GAP: in the past discussion, we focus on *circuit-level-abstraction* on *bits*

Hard to code even *real numbers* and basic *arithmetic* operations

common as part of quantum algorithm design

information of the target applications.

Design of QPLs: the level of abstraction

GAP: in the past discussion, we focus on *circuit-level-abstraction* on *bits*

Hard to code even *real numbers* and basic *arithmetic* operations

common as part of quantum algorithm design

Question 1: high-level DSLs for classical computation in superposition?

Need to compile classical computation into **reversible computation**

Handle the ancilla qubits and potentially simpler error-correction issues.

information of the target applications.

Design of QPLs: the level of abstraction

GAP: in the past discussion, we focus on *circuit-level-abstraction* on *bits*

Hard to code even *real numbers* and basic *arithmetic* operations

common as part of quantum algorithm design

Question 1: high-level DSLs for classical computation in superposition?

Need to compile classical computation into **reversible computation**

Handle the ancilla qubits and potentially simpler error-correction issues.

Question 2: high-level abstractions for quantum applications?

Circuits pass little *structural information* of the target applications.

e.g., encoding, structural freedom or so for **automation** and **optimization**

Design of QPLs: the level of abstraction

GAP: in the past discussion, we focus on *circuit-level-abstraction* on *bits*

Hard to code even *real numbers* and basic *arithmetic* operations

common as part of quantum algorithm design

Question 1: high-level DSLs for classical computation in superposition?

Need to compile classical computation into **reversible computation**

Handle the ancilla qubits and potentially simpler error-correction issues.

Question 2: high-level abstractions for quantum applications?

Circuits pass little *structural information* of the target applications.

e.g., encoding, structural freedom or so for **automation** and **optimization**

Candidate applications: Quantum Simulation

Quantum Variational Methods

Design of QPLs: the level of abstraction

GAP: in the past discussion, we focus on *circuit-level-abstraction* on *bits*

Hard to code even *real numbers* and basic *arithmetic* operations

common as part of quantum algorithm design

Question 1: high-level DSLs for classical computation in superposition?

Need to compile classical computation into **reversible computation**

Handle the ancilla qubits and potentially simpler error-correction issues.

Question 2: high-level abstractions for quantum applications?

Circuits pass little *structural information* of the target applications.

e.g., encoding, structural freedom or so for **automation** and **optimization**

Candidate applications: Quantum Simulation
Quantum Variational Methods

Question 3: allow program analysis w/ high-level abstractions?

Design of QPLs: the support of high-level objects

GAP: existing QPLs focus on describing circuits, while not using other common high-level abstractions, e.g., *objects, data structures*.

Design of QPLs: the support of high-level objects

GAP: existing QPLs focus on describing circuits, while not using other common high-level abstractions, e.g., *objects, data structures*.

Question 4: allow programmers to use (classical) data structures?

Growing need to use complicated DS. (e.g. Ambainis's element distinctness)

Design of QPLs: the support of high-level objects

GAP: existing QPLs focus on describing circuits, while not using other common high-level abstractions, e.g., *objects, data structures*.

Question 4: allow programmers to use (classical) data structures?

Growing need to use complicated DS. (e.g. Ambainis's element distinctness)

But using classical DS in quantum faces many issues:

e.g., data manipulation is generally non-reversible, even if computation can be made so.

Reversibility alone does not guarantee correct quantum interference b/c workspace.

Efficiency issues about reimplementing DS w/ above constraints.

Design of QPLs: the support of high-level objects

GAP: existing QPLs focus on describing circuits, while not using other common high-level abstractions, e.g., *objects, data structures*.

Question 4: allow programmers to use (classical) data structures?

Growing need to use complicated DS. (e.g. Ambainis's element distinctness)

But using classical DS in quantum faces many issues:

e.g., data manipulation is generally non-reversible, even if computation can be made so.

Reversibility alone does not guarantee correct quantum interference b/c workspace.

Efficiency issues about reimplementing DS w/ above constraints.

However, well-defined classical problems that PL might help with.

Design of QPLs: the support of high-level objects

GAP: existing QPLs focus on describing circuits, while not using other common high-level abstractions, e.g., *objects, data structures*.

Question 4: allow programmers to use (classical) data structures?

Growing need to use complicated DS. (e.g. Ambainis's element distinctness)

But using classical DS in quantum faces many issues:

e.g., data manipulation is generally non-reversible, even if computation can be made so.

Reversibility alone does not guarantee correct quantum interference b/c workspace.

Efficiency issues about reimplementing DS w/ above constraints.

However, well-defined classical problems that PL might help with.

Question 5: allow programmers to define quantum object/DS?

Design of QPLs: the support of high-level objects

GAP: existing QPLs focus on describing circuits, while not using other common high-level abstractions, e.g., *objects, data structures*.

Question 4: allow programmers to use (classical) data structures?

Growing need to use complicated DS. (e.g. Ambainis's element distinctness)

But using classical DS in quantum faces many issues:

e.g., data manipulation is generally non-reversible, even if computation can be made so.

Reversibility alone does not guarantee correct quantum interference b/c workspace.

Efficiency issues about reimplementing DS w/ above constraints.

However, well-defined classical problems that PL might help with.

Question 5: allow programmers to define quantum object/DS?

Allow direct modeling of quantum hardware components (QRAM, Sensors)

Design of QPLs: the support of high-level objects

GAP: existing QPLs focus on describing circuits, while not using other common high-level abstractions, e.g., *objects, data structures*.

Question 4: allow programmers to use (classical) data structures?

Growing need to use complicated DS. (e.g. Ambainis's element distinctness)

But using classical DS in quantum faces many issues:

e.g., data manipulation is generally non-reversible, even if computation can be made so.

Reversibility alone does not guarantee correct quantum interference b/c workspace.

Efficiency issues about reimplementing DS w/ above constraints.

However, well-defined classical problems that PL might help with.

Question 5: allow programmers to define quantum object/DS?

Allow direct modeling of quantum hardware components (QRAM, Sensors)

Consider quantum stack ~ truly quantum recursion ~ quantum apps

Verifying Quantum Programs: Scalability & Settings

GAP: the drawback of q. Hoare logic make existing verification schemes not **scalable**. Moreover, how about verification in more general settings?

Verifying Quantum Programs: Scalability & Settings

GAP: the drawback of q. Hoare logic make existing verification schemes not **scalable**. Moreover, how about verification in more general settings?

Question 1: how to make verification of quantum programs scalable?

Hard questions also for classical programs. Solutions for special cases.

Verifying Quantum Programs: Scalability & Settings

GAP: the drawback of q. Hoare logic make existing verification schemes not **scalable**. Moreover, how about verification in more general settings?

Question 1: how to make verification of quantum programs scalable?

Hard questions also for classical programs. Solutions for special cases.

Verification w/ classical machines:

symbolic, abstract interpretation, or so, but certainly nontrivial!

Verifying Quantum Programs: Scalability & Settings

GAP: the drawback of q. Hoare logic make existing verification schemes not **scalable**. Moreover, how about verification in more general settings?

Question 1: how to make verification of quantum programs scalable?

Hard questions also for classical programs. Solutions for special cases.

Verification w/ classical machines:

symbolic, abstract interpretation, or so, but certainly nontrivial!

Verification w/ quantum machines:

Largely unexplored! Run-time verification or other possibility?

Verifying Quantum Programs: Scalability & Settings

GAP: the drawback of q. Hoare logic make existing verification schemes not **scalable**. Moreover, how about verification in more general settings?

Question 1: how to make verification of quantum programs scalable?

Hard questions also for classical programs. Solutions for special cases.

Verification w/ classical machines:

symbolic, abstract interpretation, or so, but certainly nontrivial!

Verification w/ quantum machines:

Largely unexplored! Run-time verification or other possibility?

Question 2: how to do verification of quantum internet applications?

Quantum Internet/Communication is another recent interest

Verifying Quantum Programs: Scalability & Settings

GAP: the drawback of q. Hoare logic make existing verification schemes not **scalable**. Moreover, how about verification in more general settings?

Question 1: how to make verification of quantum programs scalable?

Hard questions also for classical programs. Solutions for special cases.

Verification w/ classical machines:

symbolic, abstract interpretation, or so, but certainly nontrivial!

Verification w/ quantum machines:

Largely unexplored! Run-time verification or other possibility?

Question 2: how to do verification of quantum internet applications?

Quantum Internet/Communication is another recent interest

Develop Q Hoare logic for *parallel, concurrent, distributed* programs.

Some preliminary results exist. Essential difficulty exists due to quantum correlations.

Debugging Quantum Programs for NISQ

GAP: assertion-based debugging might in general distribute q. systems.

Li et al. (OOPSLA 2020) provides projection-based assertion scheme, which in principle resolves the issue for capable quantum computers. How about NISQ?

Debugging Quantum Programs for NISQ

GAP: assertion-based debugging might in general distribute q. systems.

Li et al. (OOPSLA 2020) provides projection-based assertion scheme, which in principle resolves the issue for capable quantum computers. How about NISQ?

Question 3: how to verify and debug NISQ applications?

Debugging Quantum Programs for NISQ

GAP: assertion-based debugging might in general distribute q. systems.

Li et al. (OOPSLA 2020) provides projection-based assertion scheme, which in principle resolves the issue for capable quantum computers. How about NISQ?

Question 3: how to verify and debug NISQ applications?

Need to develop new frameworks as program features are simple

e.g., only contains simple conditional and loops

Debugging Quantum Programs for NISQ

GAP: assertion-based debugging might in general distribute q. systems.

Li et al. (OOPSLA 2020) provides projection-based assertion scheme, which in principle resolves the issue for capable quantum computers. How about NISQ?

Question 3: how to verify and debug NISQ applications?

Need to develop new frameworks as program features are simple

e.g., only contains simple conditional and loops

Need to be very resilient to hardware errors

For NISQ machines, all operations could be erroneous

Debugging Quantum Programs for NISQ

GAP: assertion-based debugging might in general distribute q. systems.

Li et al. (OOPSLA 2020) provides projection-based assertion scheme, which in principle resolves the issue for capable quantum computers. How about NISQ?

Question 3: how to verify and debug NISQ applications?

Need to develop new frameworks as program features are simple

e.g., only contains simple conditional and loops

Need to be very resilient to hardware errors

For NISQ machines, all operations could be erroneous

Need also to be scalable

Classical simulation hard to scale; large q operations might contain more errors

Debugging Quantum Programs for NISQ

GAP: assertion-based debugging might in general distribute q. systems.

Li et al. (OOPSLA 2020) provides projection-based assertion scheme, which in principle resolves the issue for capable quantum computers. How about NISQ?

Question 3: how to verify and debug NISQ applications?

Need to develop new frameworks as program features are simple

e.g., only contains simple conditional and loops

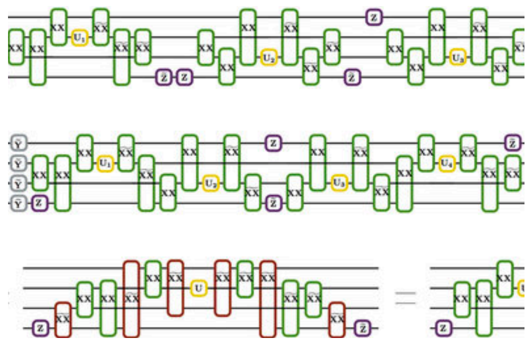
Need to be very resilient to hardware errors

For NISQ machines, all operations could be erroneous

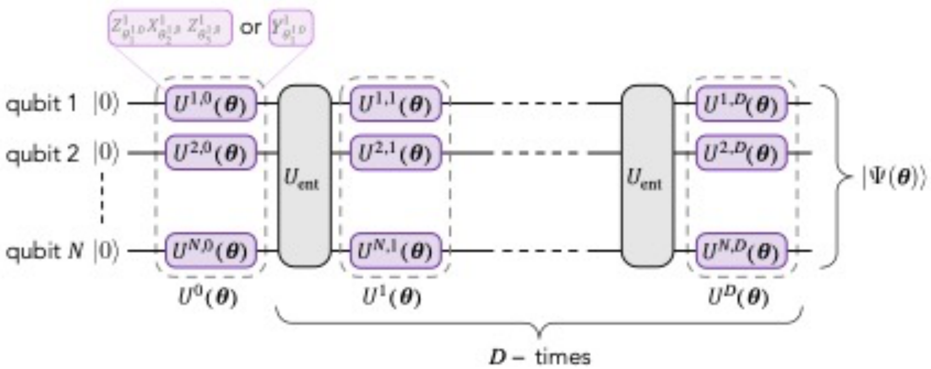
Need also to be scalable

Classical simulation hard to scale; large q operations might contain more errors

Likely to be application-specific



Quantum Simulation



Variational Quantum Methods

Compilation of Quantum Application: Analog Machines

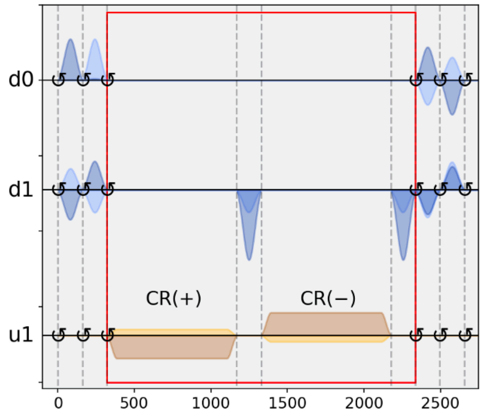
GAP: most of existing tool-chains compile to circuits with non-native gates on the hardware. Lead to very inefficient use of NISQ machines.

Compilation of Quantum Application: Analog Machines

GAP: most of existing tool-chains compile to circuits with non-native gates on the hardware. Lead to very inefficient use of NISQ machines.

Question 1: develop hardware-aware compilation?

Recent study suggests : compilation to *control pulses, qutrits*, or so

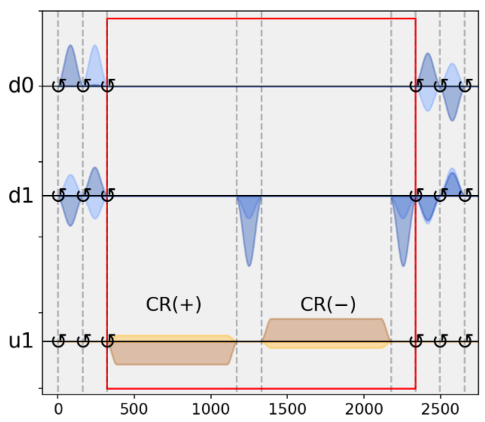


Compilation of Quantum Application: Analog Machines

GAP: most of existing tool-chains compile to circuits with non-native gates on the hardware. Lead to very inefficient use of NISQ machines.

Question 1: develop hardware-aware compilation?

Recent study suggests : compilation to *control pulses, qutrits*, or so



examples identified, but no systematic study for e.g., **efficiency**, and **verification**

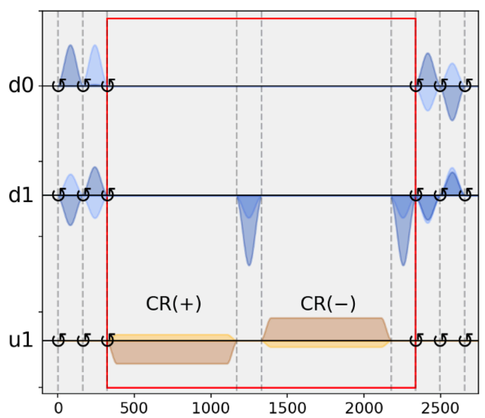
Shi et al. Proceedings of the IEEE, Jun 2020

Compilation of Quantum Application: Analog Machines

GAP: most of existing tool-chains compile to circuits with non-native gates on the hardware. Lead to very inefficient use of NISQ machines.

Question 1: develop hardware-aware compilation?

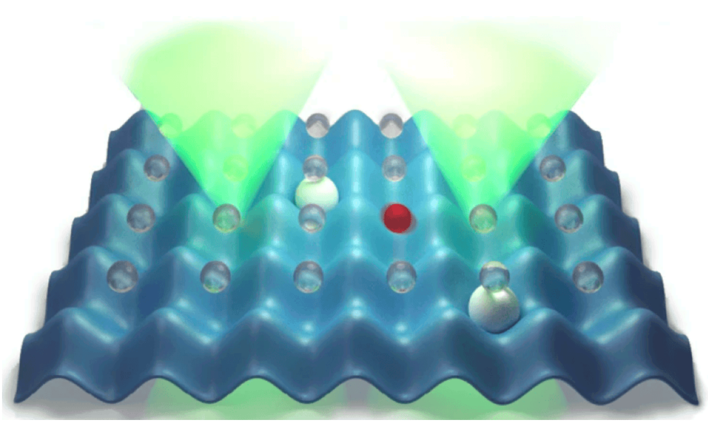
Recent study suggests : compilation to *control pulses, qutrits*, or so



examples identified, but no systematic study for e.g., **efficiency**, and **verification**

Shi et al. Proceedings of the IEEE, Jun 2020

Question 2: direct compilation to analog / special purpose q machines?



Analog machine modeled after the physics to simulate

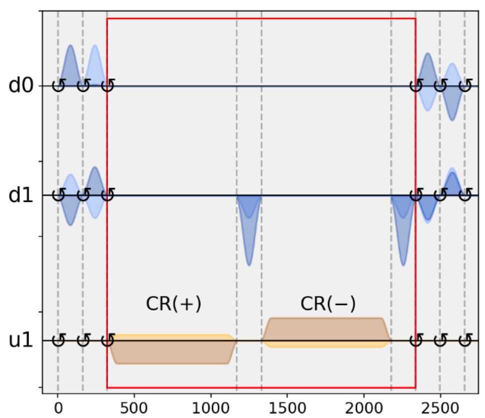
Unexplored yet. But would be of great interests!

Compilation of Quantum Application: Analog Machines

GAP: most of existing tool-chains compile to circuits with non-native gates on the hardware. Lead to very inefficient use of NISQ machines.

Question 1: develop hardware-aware compilation?

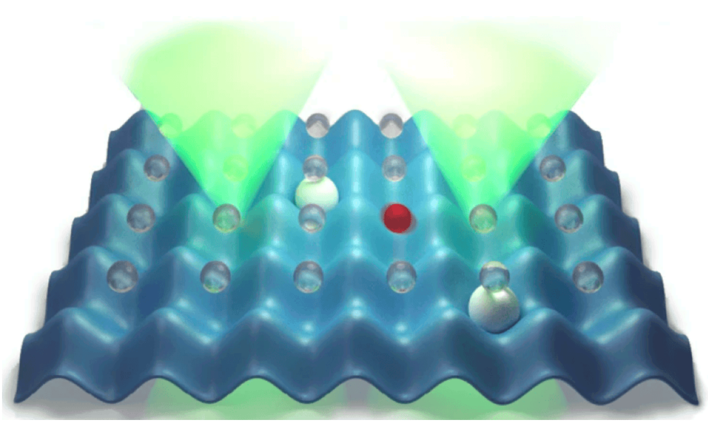
Recent study suggests : compilation to *control pulses, qutrits*, or so



examples identified, but no systematic study for e.g., **efficiency**, and **verification**

Shi et al. Proceedings of the IEEE, Jun 2020

Question 2: direct compilation to analog / special purpose q machines?



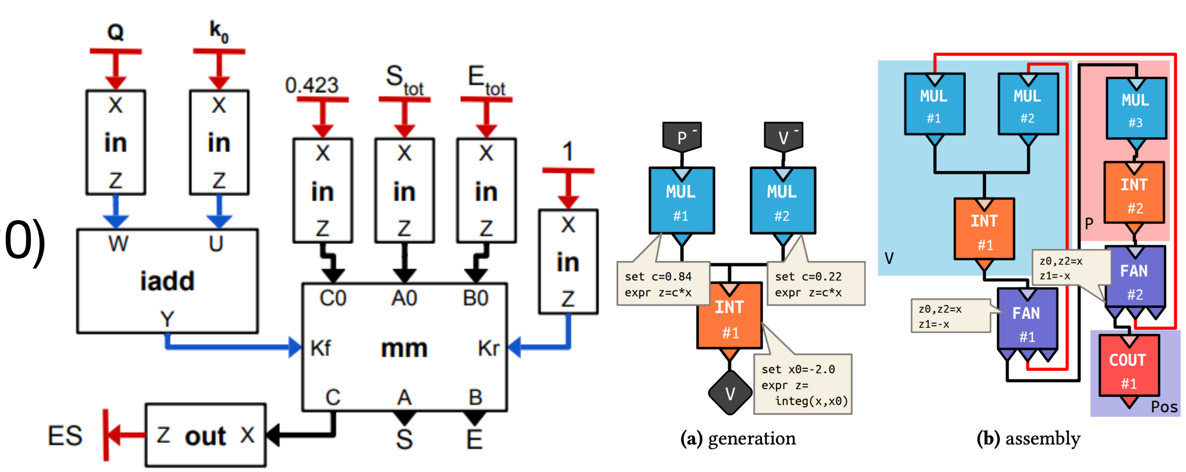
Analog machine modeled after the physics to simulate

Unexplored yet. But would be of great interests!

Classical Examples:

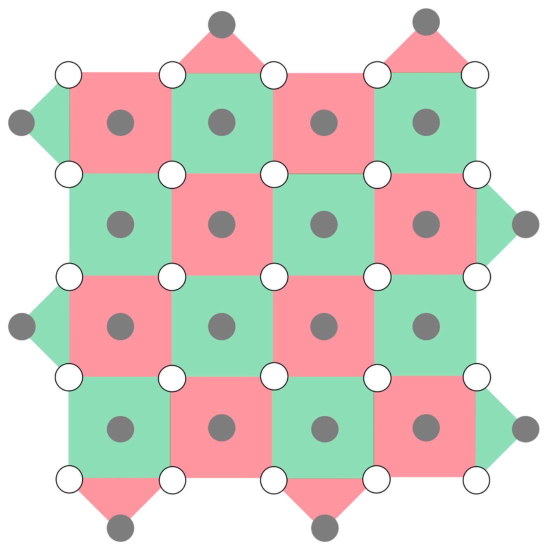
Achour et al. (PLDI16)

Achour & Rinard (ASPLOS 20)

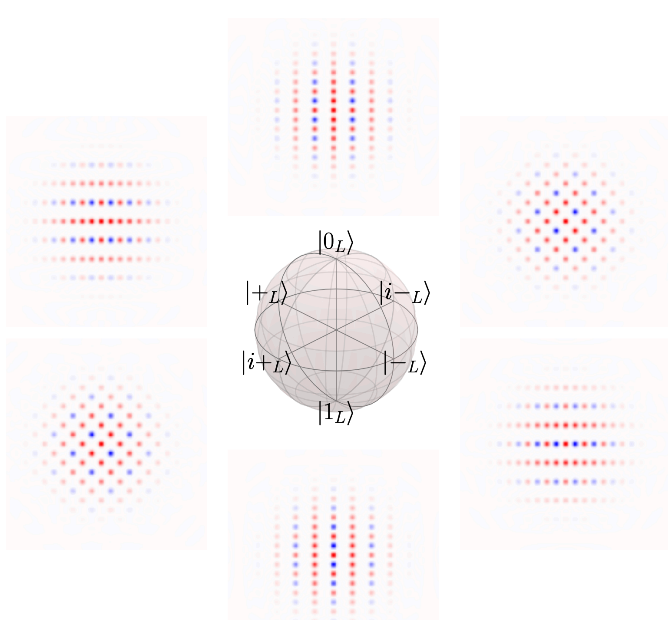


ERROR

Nature



Quantum Error Correction
Fight
Quantum Decoherence



ERROR

Approximate Computing & Quantum Computing

- General-purpose fault-tolerant quantum computers are *impractical* in the near term.
- *Near-term* practical quantum applications must focus on Noisy and Intermediate-Scale Quantum (*NISQ*) computers, where precisely controllable quits are *expensive*, *error-prone*, and *scarce*.

Approximate Computing & Quantum Computing

- General-purpose fault-tolerant quantum computers are *impractical* in the near term.
- *Near-term* practical quantum applications must focus on Noisy and Intermediate-Scale Quantum (*NISQ*) computers, where precisely controllable quits are *expensive*, *error-prone*, and *scarce*.

Goal: reliable quantum programs with resource optimization!

Approximate Computing & Quantum Computing

- General-purpose fault-tolerant quantum computers are *impractical* in the near term.
- *Near-term* practical quantum applications must focus on Noisy and Intermediate-Scale Quantum (*NISQ*) computers, where precisely controllable quits are *expensive*, *error-prone*, and *scarce*.

Goal: reliable quantum programs with resource optimization!

- Quantitive guarantee on the reliability/accuracy of quantum programs based on specific hardware information.

Approximate Computing & Quantum Computing

- General-purpose fault-tolerant quantum computers are *impractical* in the near term.
- *Near-term* practical quantum applications must focus on Noisy and Intermediate-Scale Quantum (*NISQ*) computers, where precisely controllable quits are *expensive*, *error-prone*, and *scarce*.

Goal: reliable quantum programs with resource optimization!

- Quantitive guarantee on the reliability/accuracy of quantum programs based on specific hardware information.
- High-level abstraction of error-handling primitives in quantum programs.

Approximate Computing & Quantum Computing

- General-purpose fault-tolerant quantum computers are *impractical* in the near term.
- *Near-term* practical quantum applications must focus on Noisy and Intermediate-Scale Quantum (*NISQ*) computers, where precisely controllable quits are *expensive*, *error-prone*, and *scarce*.

Goal: reliable quantum programs with resource optimization!

- Quantitive guarantee on the reliability/accuracy of quantum programs based on specific hardware information.
- High-level abstraction of error-handling primitives in quantum programs.
- Automatic error-resource-optimization on a per-program basis!

Methodology

Methodology

- Elevate the handling of errors to the level of programming language.

Methodology

- Elevate the handling of errors to the level of programming language.
- Reason *reliability/accuracy* of quantum programs via *static* analysis.

Methodology

- Elevate the handling of errors to the level of programming language.
- Reason *reliability/accuracy* of quantum programs via *static* analysis.
- Conduct *resource optimization* via *code synthesis* of quantum programs.

Methodology

- Elevate the handling of errors to the level of programming language.
- Reason *reliability/accuracy* of quantum programs via *static* analysis.
- Conduct *resource optimization* via *code synthesis* of quantum programs.

An important classical tool: *approximate computing* !

Methodology

- Elevate the handling of errors to the level of programming language.
- Reason *reliability/accuracy* of quantum programs via *static* analysis.
- Conduct *resource optimization* via *code synthesis* of quantum programs.

An important classical tool: *approximate computing* !

- Return possibly inaccurate/approximate results!

Methodology

- Elevate the handling of errors to the level of programming language.
- Reason *reliability/accuracy* of quantum programs via *static* analysis.
- Conduct *resource optimization* via *code synthesis* of quantum programs.

An important classical tool: *approximate computing* !

- Return possibly inaccurate/approximate results!
  - *unreliable hardware*

Methodology

- Elevate the handling of errors to the level of programming language.
- Reason *reliability/accuracy* of quantum programs via *static* analysis.
- Conduct *resource optimization* via *code synthesis* of quantum programs.

An important classical tool: *approximate computing* !

- Return possibly inaccurate/approximate results!
  - *unreliable hardware*
  - *limited computational resource*

Methodology

- Elevate the handling of errors to the level of programming language.
- Reason *reliability/accuracy* of quantum programs via *static* analysis.
- Conduct *resource optimization* via *code synthesis* of quantum programs.

An important classical tool: *approximate computing* !

- Return possibly inaccurate/approximate results!
  - *unreliable hardware*
  - *limited computational resource*
- **Good** when *approximate results* are sufficient for applications!

Methodology

- Elevate the handling of errors to the level of programming language.
- Reason *reliability/accuracy* of quantum programs via *static* analysis.
- Conduct *resource optimization* via *code synthesis* of quantum programs.

An important classical tool: *approximate computing* !

- Return possibly inaccurate/approximate results!
  - *unreliable hardware*
  - *limited computational resource*
- **Good** when *approximate results* are sufficient for applications!
  - *vision, machine learning; also with guarantees for critical data*

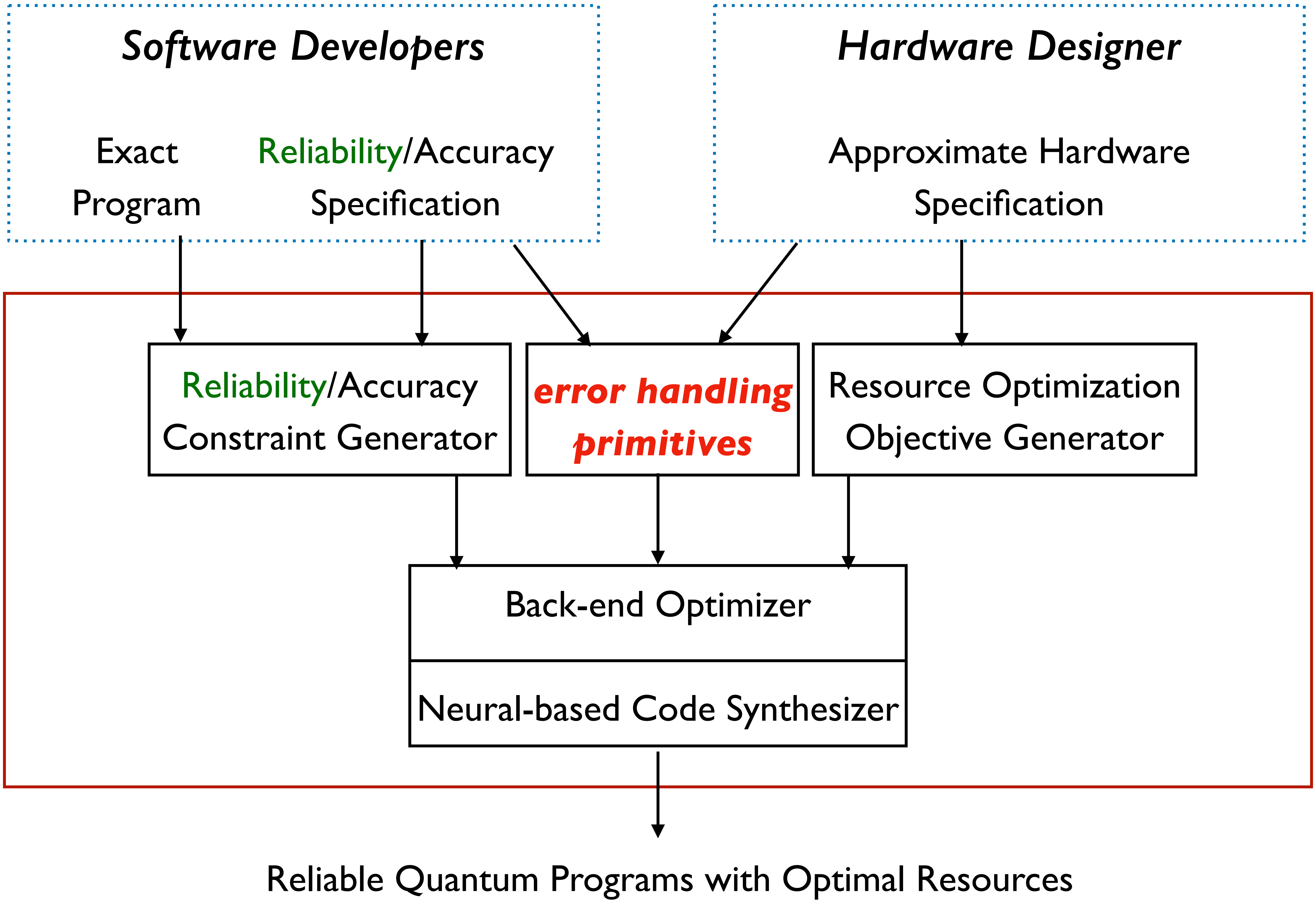
Methodology

- Elevate the handling of errors to the level of programming language.
- Reason *reliability/accuracy* of quantum programs via *static* analysis.
- Conduct *resource optimization* via *code synthesis* of quantum programs.

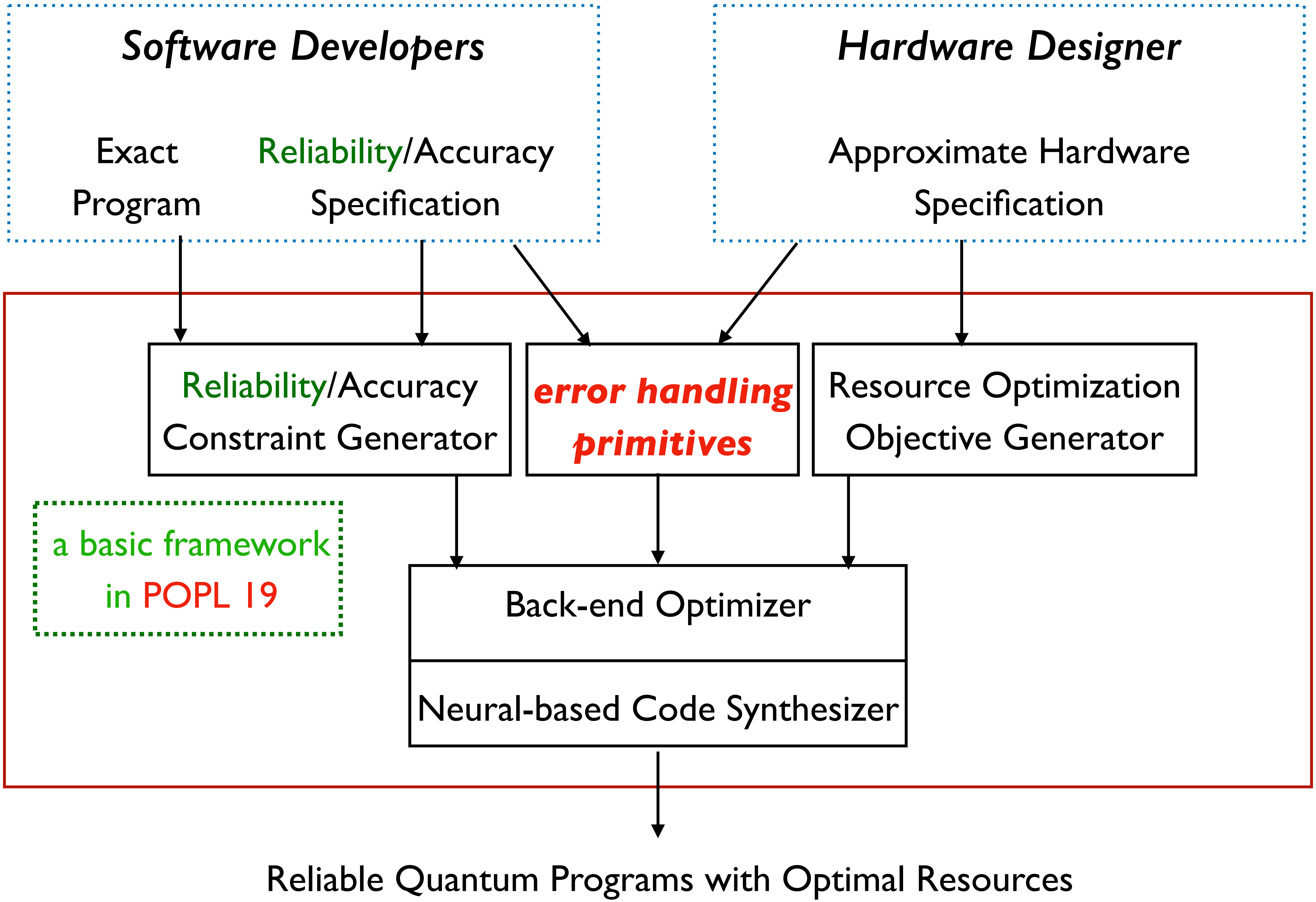
An important classical tool: *approximate computing* !

- Return possibly inaccurate/approximate results!
  - *unreliable hardware*
  - *limited computational resource*
- **Good** when *approximate results* are sufficient for applications!
  - *vision, machine learning; also with guarantees for critical data*
- Various techniques developed in classical PL literature.

Overview

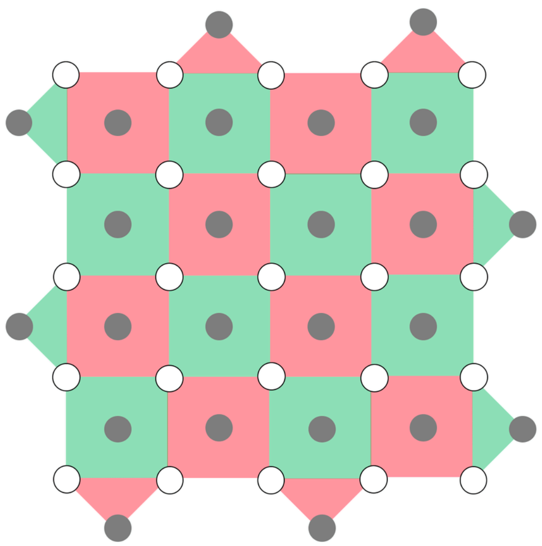

Software Developers
Exact Program
Reliability/Accuracy Specification
Hardware Designer
Approximate Hardware Specification
Reliability/Accuracy Constraint Generator
error handling primitives
Resource Optimization Objective Generator
Back-end Optimizer
Neural-based Code Synthesizer
Reliable Quantum Programs with Optimal Resources

Overview

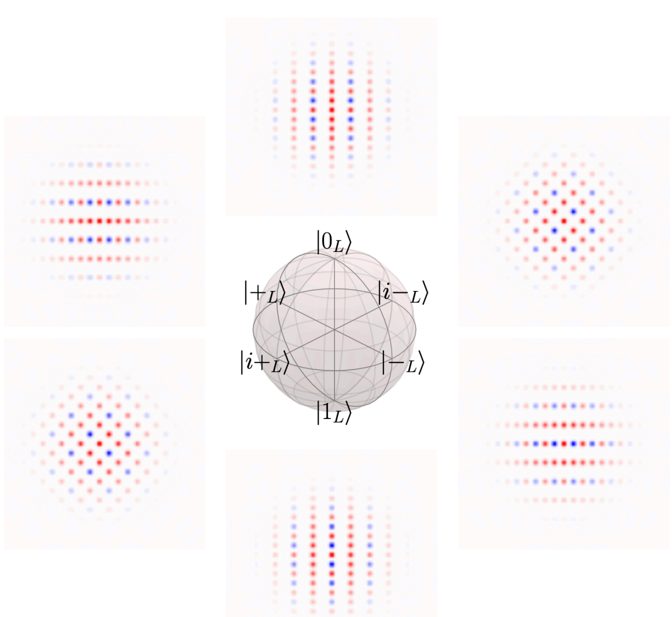


Software Developers
Exact Program
Reliability/Accuracy Specification
Hardware Designer
Approximate Hardware Specification
Reliability/Accuracy Constraint Generator
error handling primitives
Resource Optimization Objective Generator
a basic framework in POPL 19
Back-end Optimizer
Neural-based Code Synthesizer
Reliable Quantum Programs with Optimal Resources

Nature

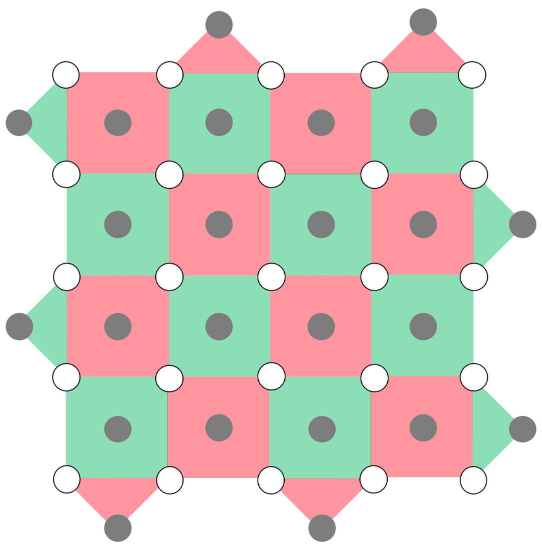


Quantum Error Correction
Fight
Quantum Decoherence

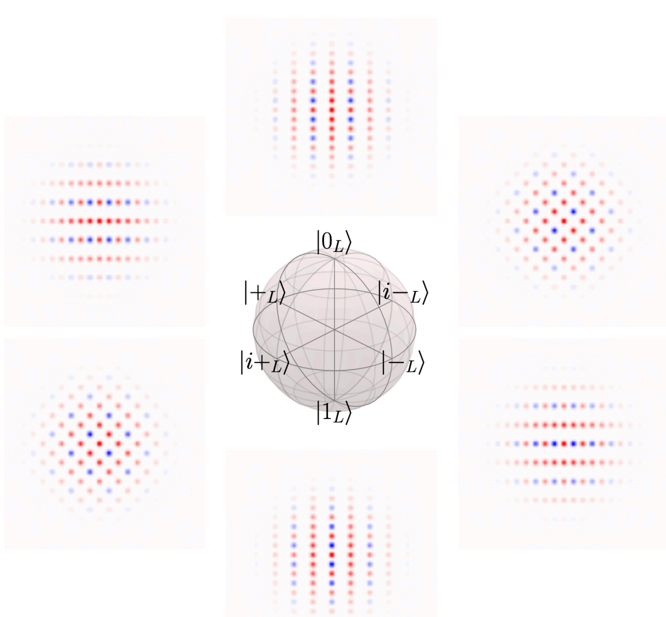


ERROR

Nature

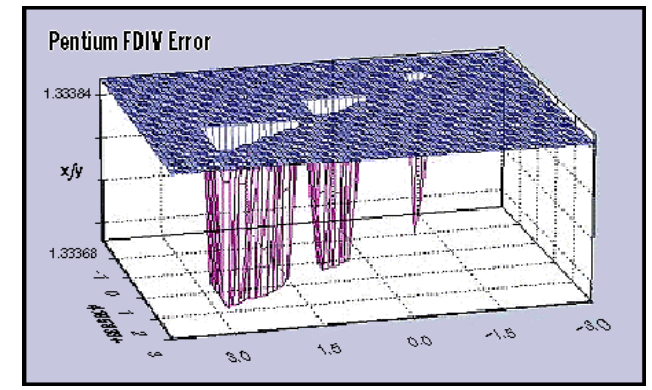


Quantum Error Correction
Fight
Quantum Decoherence



ERROR

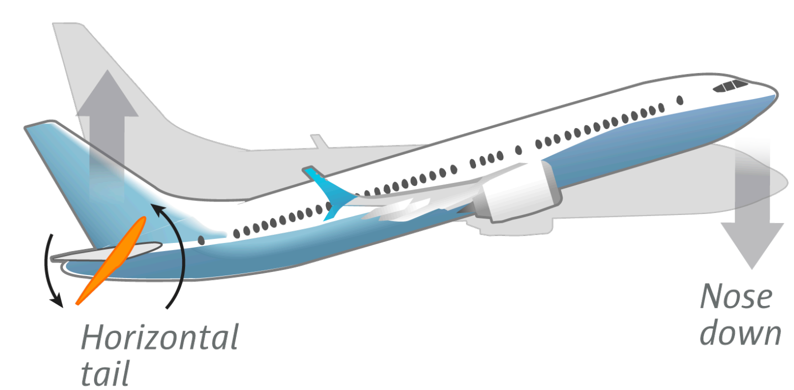
Human



Intel Pentium FPU error

Ariane 5

MCAS safety system engages



Human Errors in Quantum Software Engineering

Being careful cannot solve the human error problem in either classical or quantum.

Quantum case : Significantly More **CHALLENGING** than Classical

- standard software assurance techniques, e.g., black-box / unit test, expensive in q.
- quantum mechanics prohibits certain testing, e.g., assertions

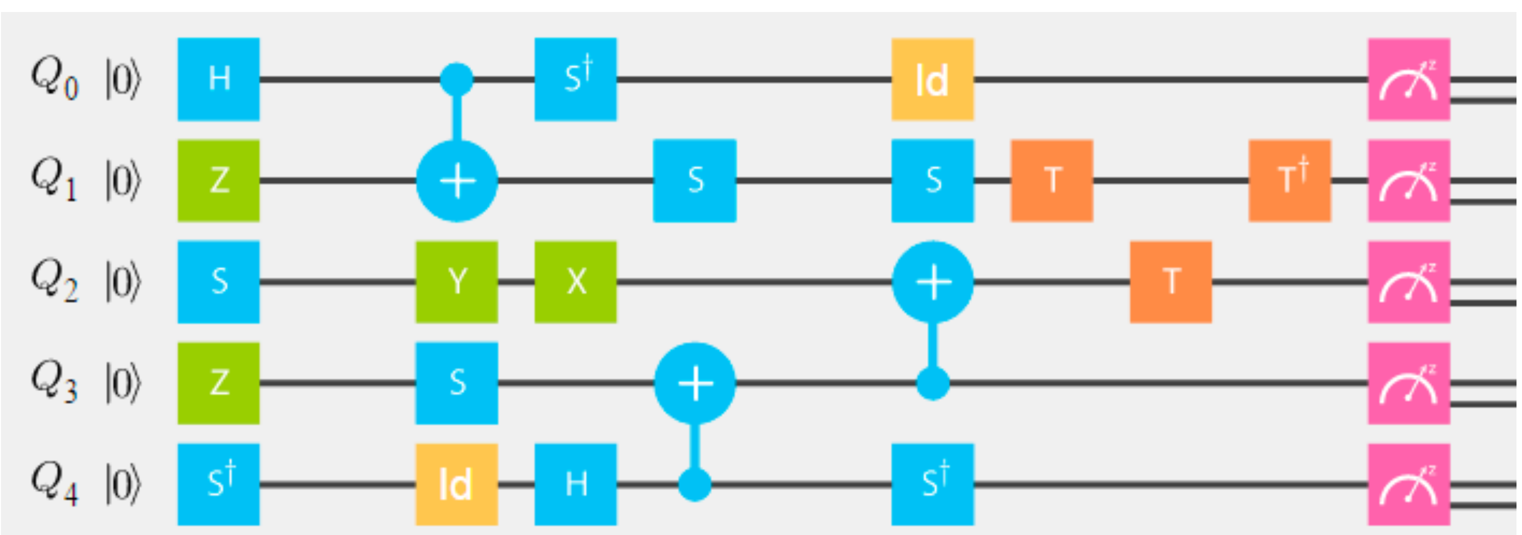
Human Errors in Quantum Software Engineering

Being careful cannot solve the human error problem in either classical or quantum.

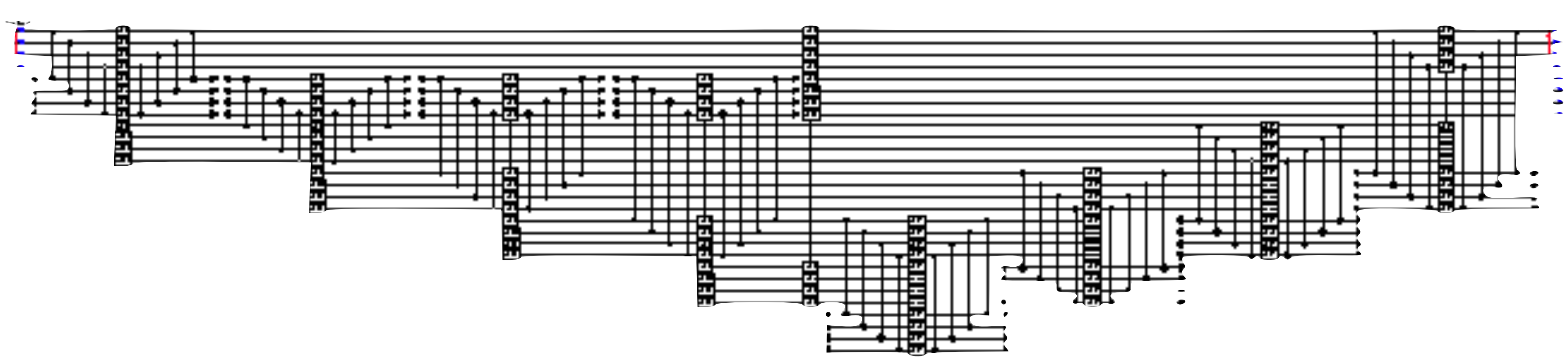
Quantum case : Significantly More **CHALLENGING** than Classical

- standard software assurance techniques, e.g., black-box / unit test, expensive in q.
- quantum mechanics prohibits certain testing, e.g., assertions

Reality: testing in quantum today



confirming the circuit by observation…. not scalable…



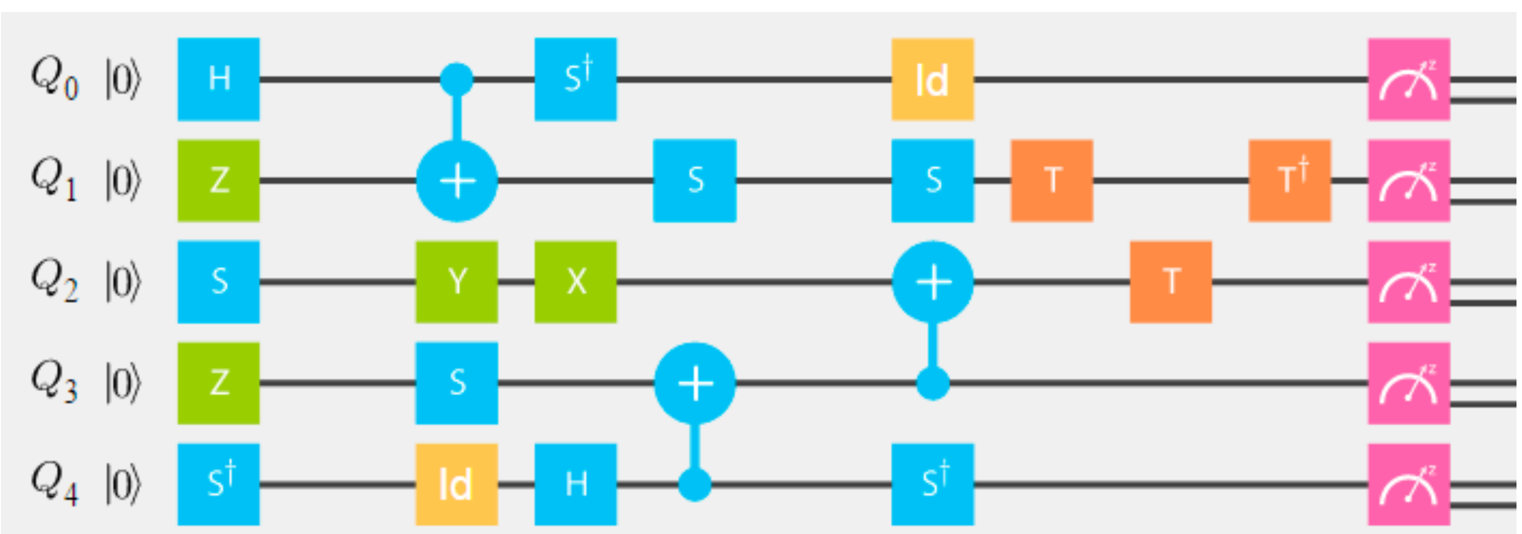
Human Errors in Quantum Software Engineering

Being careful cannot solve the human error problem in either classical or quantum.

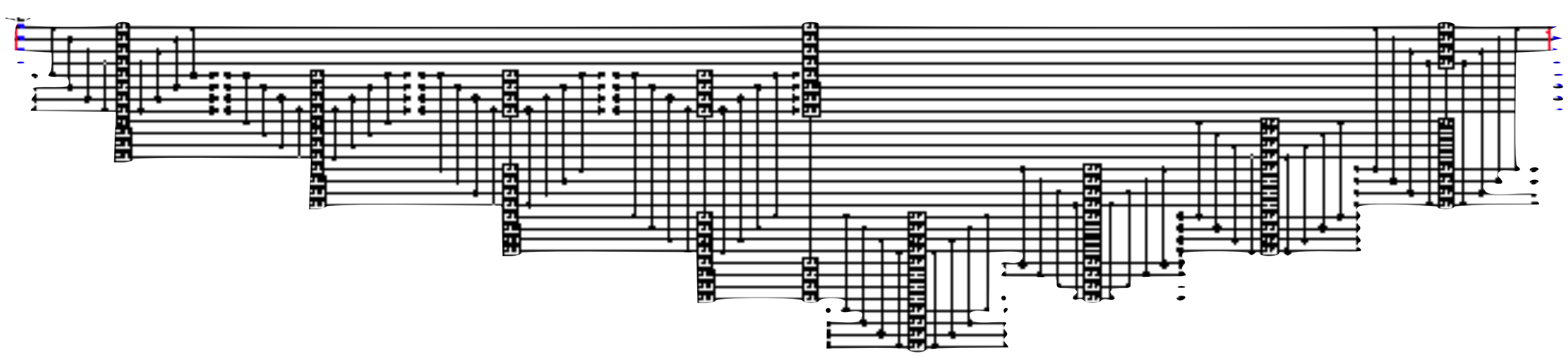
Quantum case : Significantly More **CHALLENGING** than Classical

- standard software assurance techniques, e.g., black-box / unit test, expensive in q.
- quantum mechanics prohibits certain testing, e.g., assertions

Reality: testing in quantum today



confirming the circuit by observation…. not scalable…



QISKIT Compiler ERRORs

Much **HARDER** to detect!

Serious Consequences!

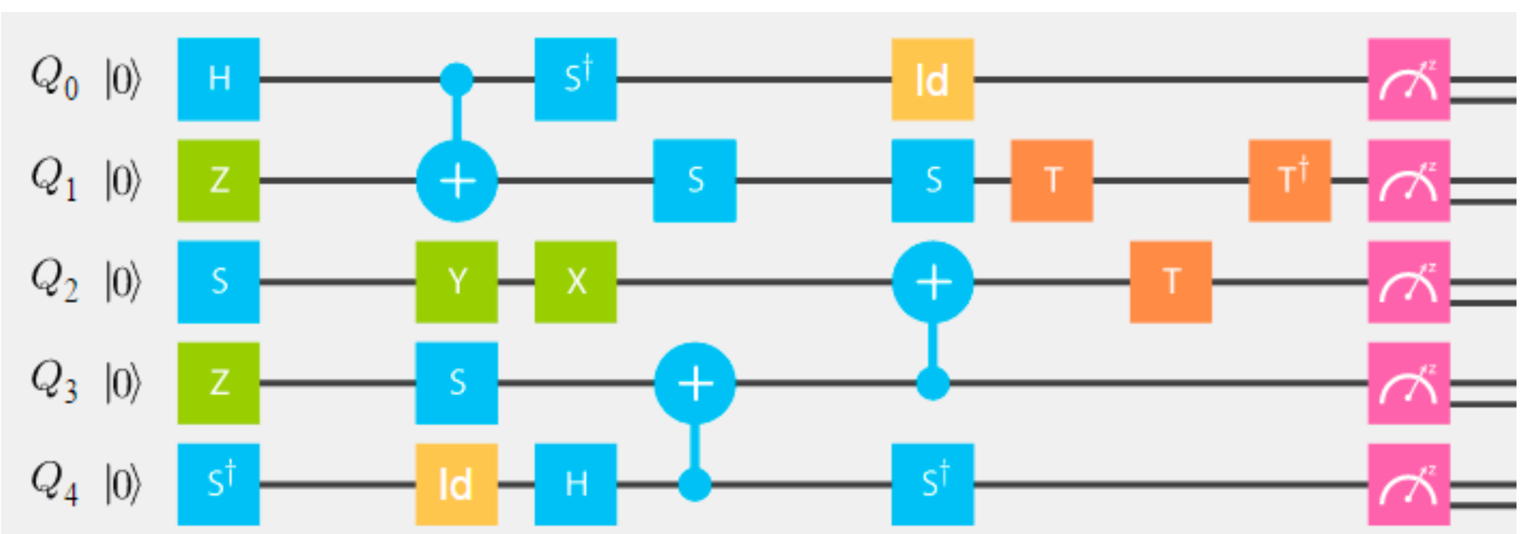
Human Errors in Quantum Software Engineering

Being careful cannot solve the human error problem in either classical or quantum.

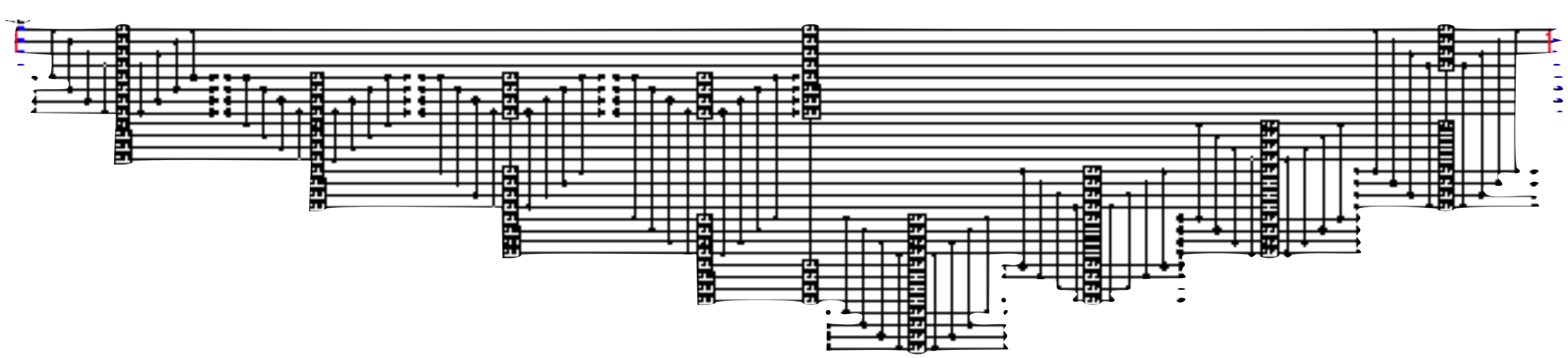
Quantum case : Significantly More **CHALLENGING** than Classical

- standard software assurance techniques, e.g., black-box / unit test, expensive in q.
- quantum mechanics prohibits certain testing, e.g., assertions

Reality: testing in quantum today



confirming the circuit by observation…. not scalable…



QISKIT Compiler ERRORs

Much **HARDER** to detect!

Serious Consequences!

Similar Concerns in classical !

More **SERIOUS** in quantum !

Certified software: a solution to validation of q. software

The Verifying Compiler: A Grand Challenge for Computing Research

TONY HOARE

Microsoft Research Ltd., Cambridge, UK

Journal of the ACM, Vol 50, 2003

Certified software: a solution to validation of q. software

The Verifying Compiler: A Grand Challenge for Computing Research

TONY HOARE

Microsoft Research Ltd., Cambridge, UK

Journal of the ACM, Vol 50, 2003

GCC : many bugs in software testing

CompCert: a certified “GCC”, bug-free

Certified software: a solution to validation of q. software

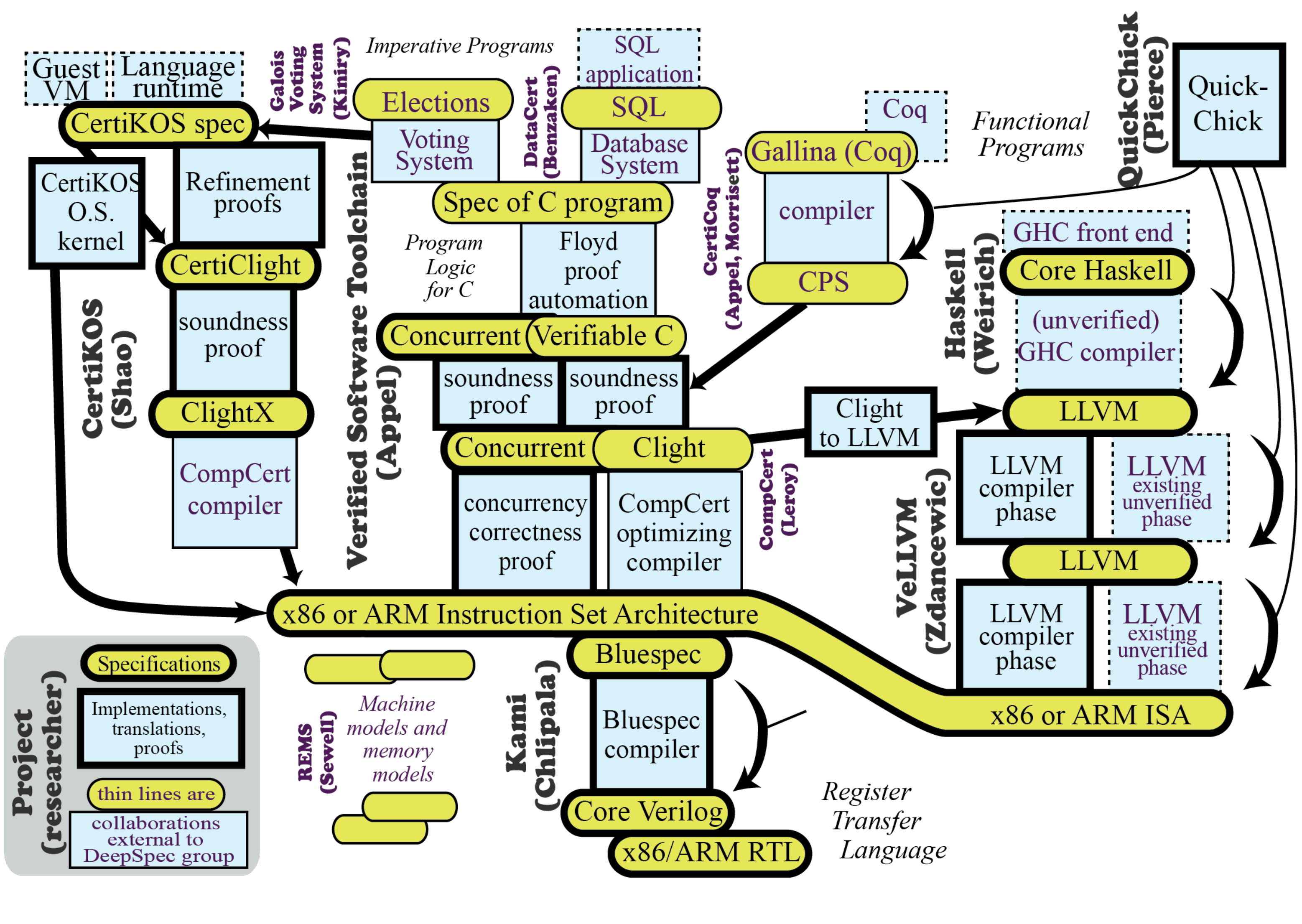
The Verifying Compiler: A Grand Challenge for Computing Research

TONY HOARE

Microsoft Research Ltd., Cambridge, UK

Journal of the ACM, Vol 50, 2003

GCC : many bugs in software testing
CompCert: a certified “GCC”, bug-free



Verified Software Toolchain

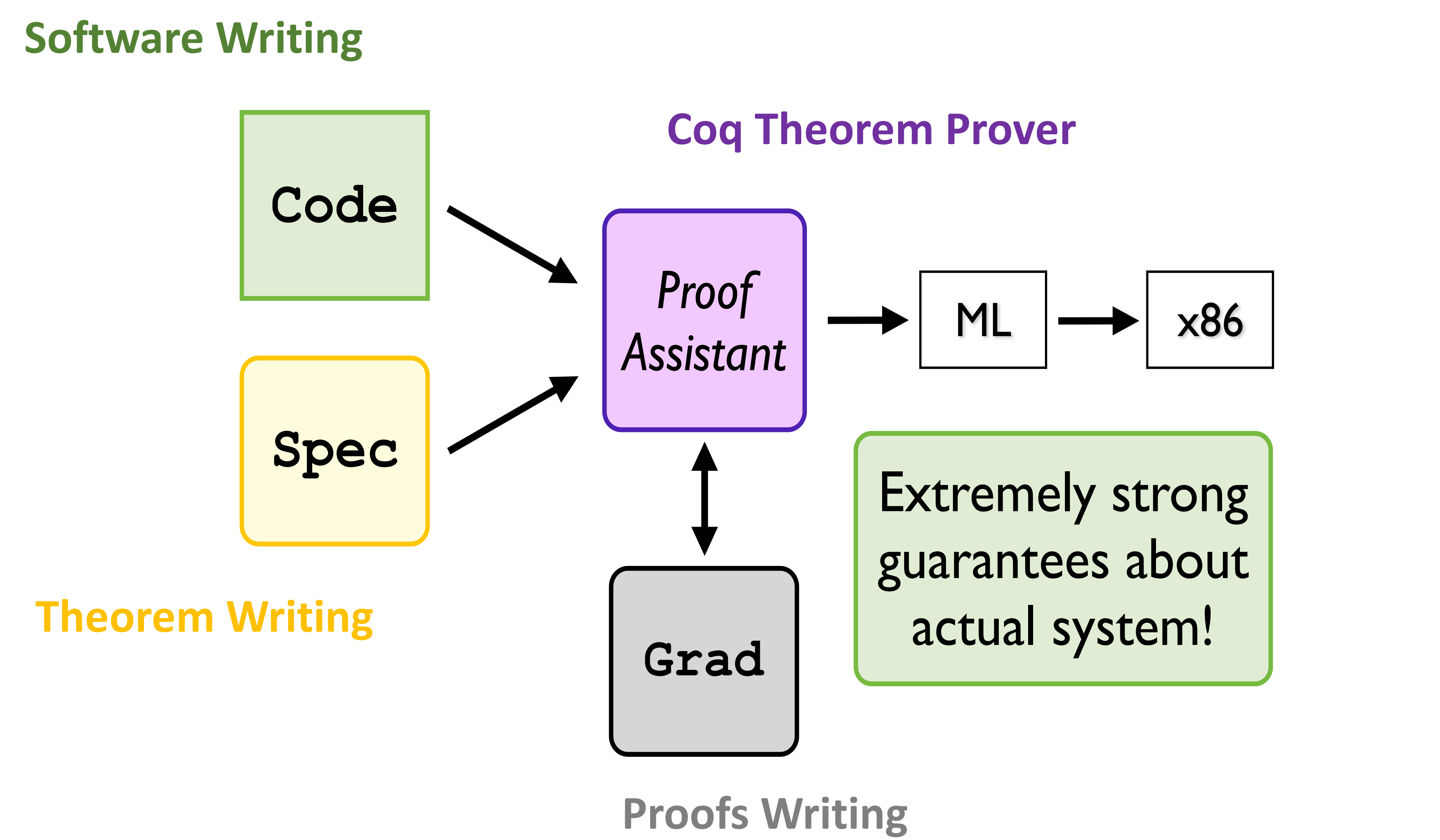
CERTIKOS

Vellvm verified LLVM

Certi Coq

Kami

Certified software: a solution to validation of q. software



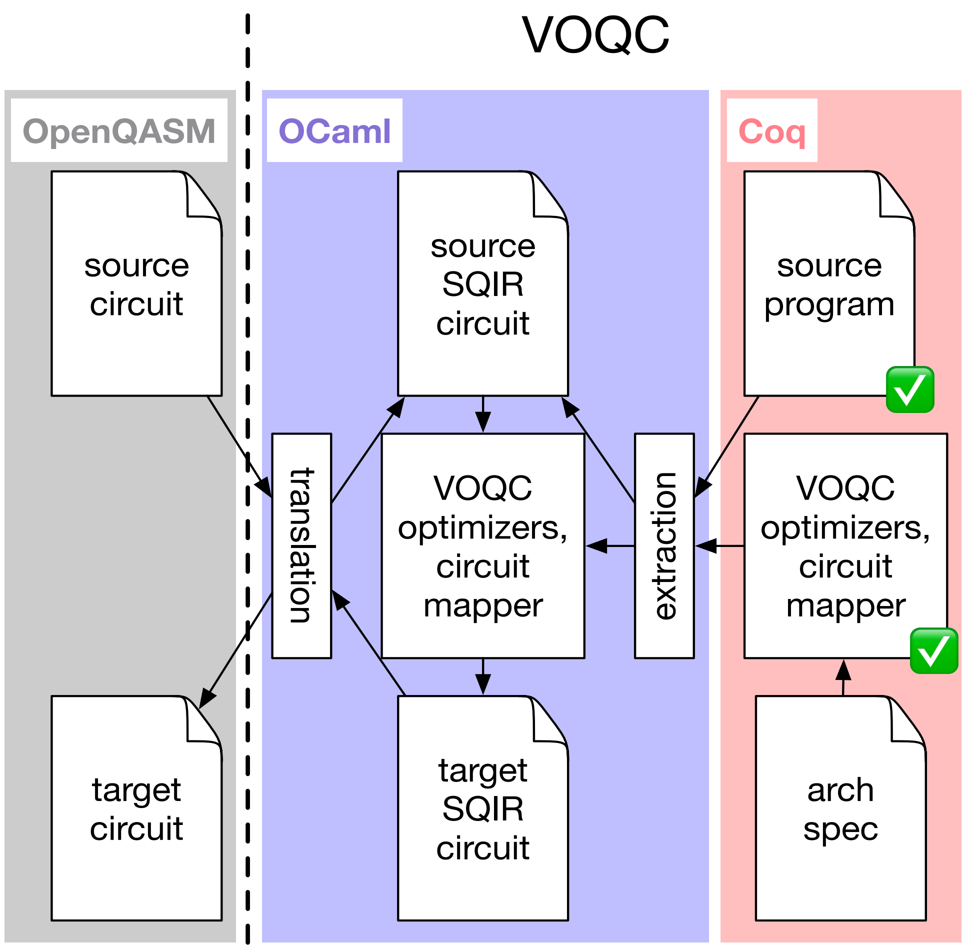
(1) Ensure correctness of code by construction.

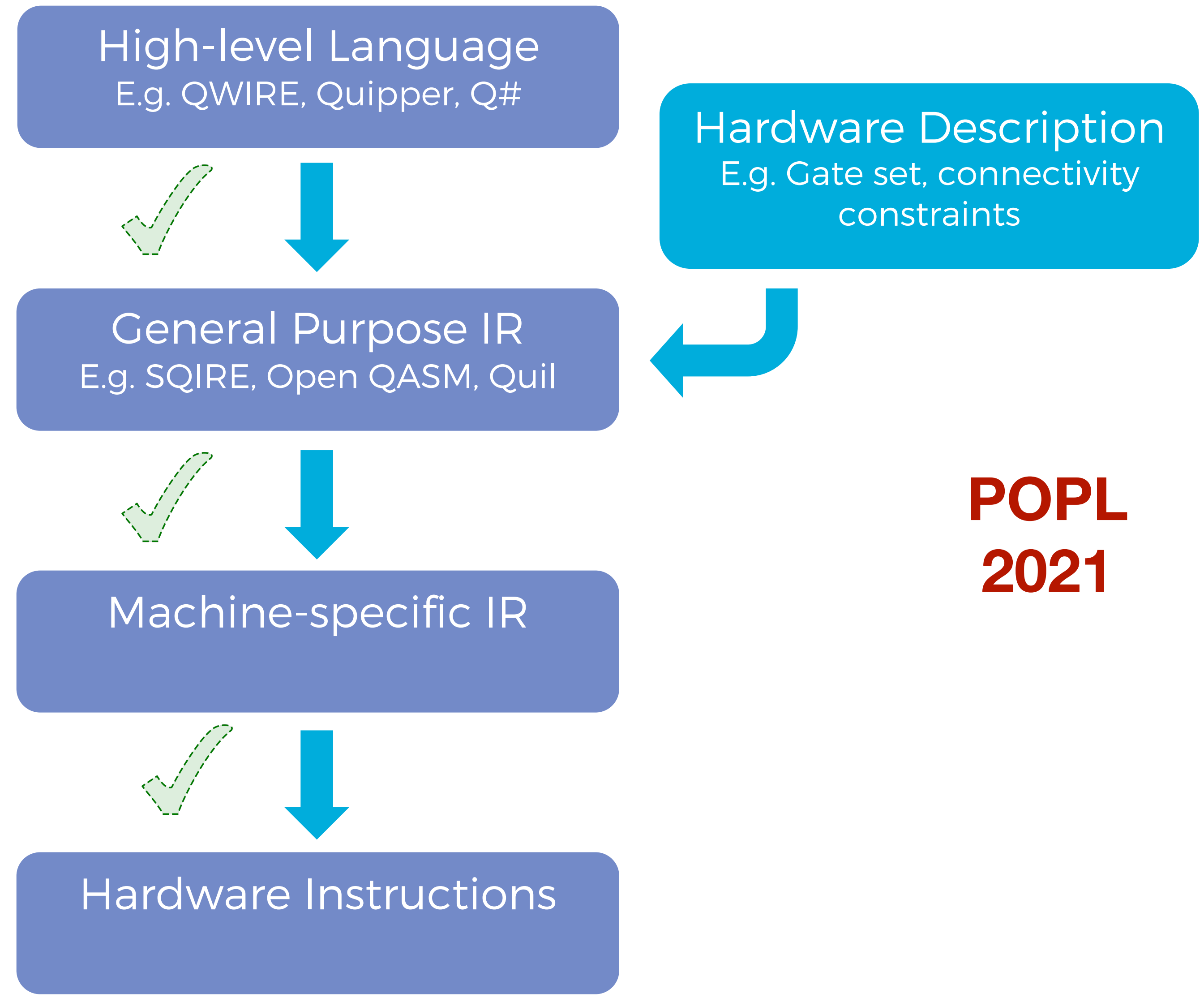
(2) Scalability for quantum based on symbolic proofs.

Joint Center for Quantum Information and Computer Science

PLUM

(Verified Optimizer for Quantum Circuits)





POPL 2021

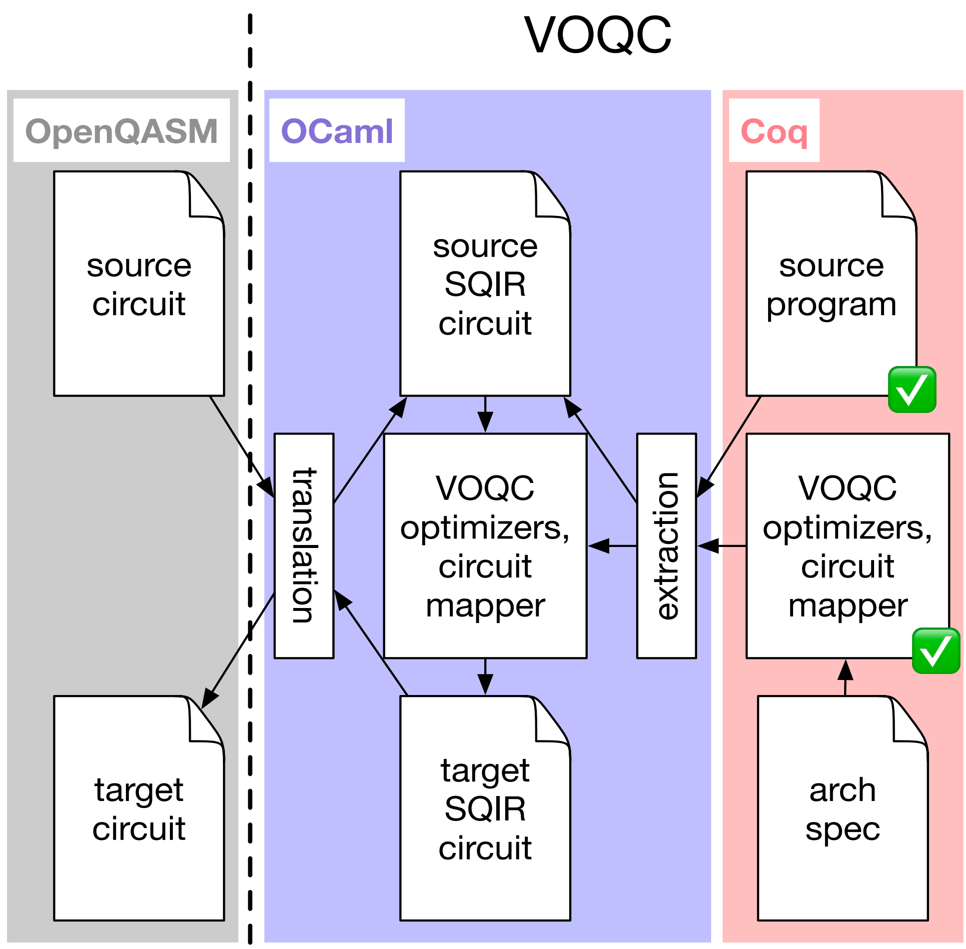
VOQC: a first step towards a fully certified quantum compiler.

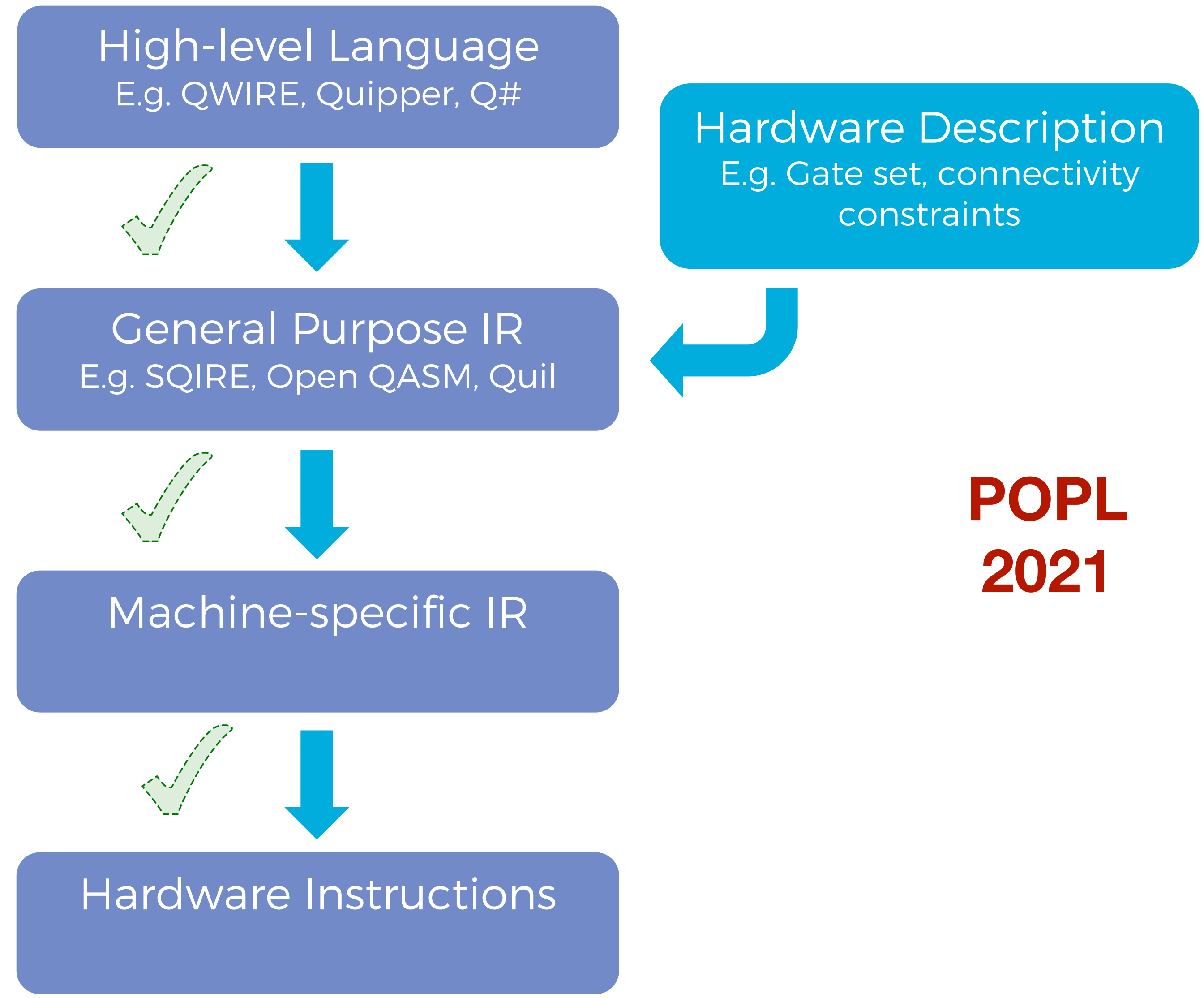
SQIRE: a simple quantum intermediate-representation embedded in Coq.

JOINT CENTER FOR QUANTUM INFORMATION AND COMPUTER SCIENCE

PLUM

(Verified Optimizer for Quantum Circuits)





POPL 2021

VOQC: a first step towards a fully certified quantum compiler.

SQIRE: a simple quantum intermediate-representation embedded in Coq.

Our infrastructure powerful enough:

an end-to-end implementation of Shor's algorithm & its correctness proof.