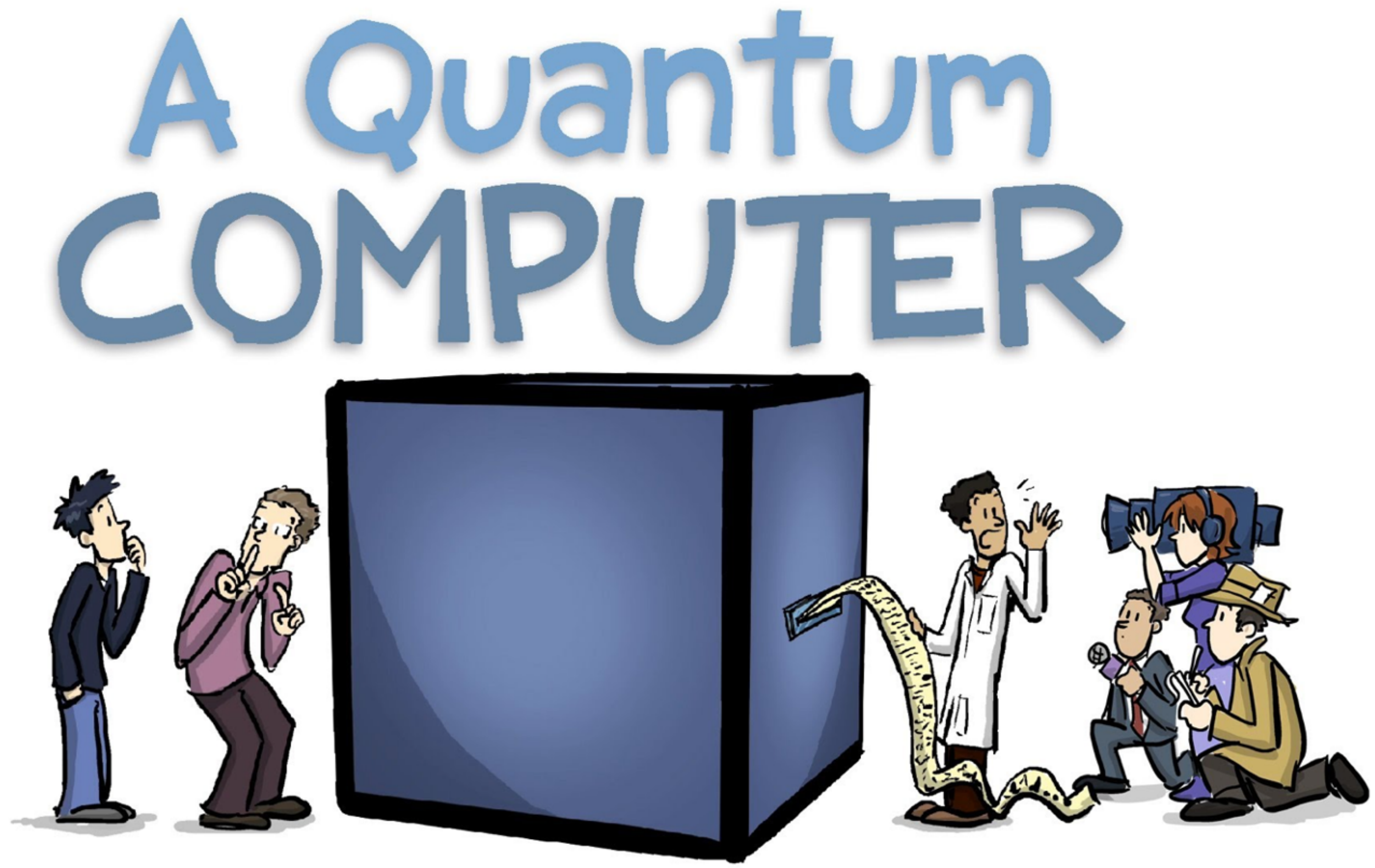
An Invitation to the intersection of
Quantum Computing & Programming Languages

Tutorial at POPL 2021



Xiaodi Wu
QuICS & UMD

UNIVERSITY OF MARYLAND

JOINT CENTER FOR QUANTUM INFORMATION AND COMPUTER SCIENCE

About this Tutorial:

Goal: An Invitation due to limited time

Cover Some Basic Quantum Computing & PL

Provide References / Pointers for further study

About this Tutorial:

Goal: An Invitation due to limited time
Cover Some Basic Quantum Computing & PL
Provide References / Pointers for further study

Format: Tutorial divided into 3 parts:

(1) Introduction to Quantum Computing and Potential Roles of Programming Languages (25 min + 5 Q & A)

(2) A Mini-Course of Quantum Hoare Logic on Quantum While Language (30 min + 5 Q & A)

(3) Discussion on existing and potential Programming Language research opportunities (20 min + 5 Q & A)

About the Speaker:

Wu: assistant professor at UMD working on quantum computing from CS perspective in general.

About the Speaker:

Wu: assistant professor at UMD **working on quantum computing from CS perspective in general.**

Teaching in Q. Computing

Past Courses

This is a collection of courses that I have taught in the past for your references. Please be cautious as thes

University of Maryland, College Park (2017 - present)

- Complexity Theory (CMSC 652): graduate-level theory core course
  - Fall 2017
- Introduction to Quantum Computing (CMSC/PHYS 457): undergraduate-level introduction to quar
  - Spring 2018, Spring 2020, Spring 2021
- Introduction to Quantum Information Processing (CMSC 657): graduate-level introduction to quan
  - Fall 2018, Fall 2019

University of Oregon (2015 - 2017)

- Intermediate Data Structure (CIS 313): undergraduate CS major theory course.
  - Winter 2016, Fall 2016, Winter 2017.
- Introduction to Quantum Information Processing (CIS 410/510): senior undergraduate / graduate
  - Spring 2016, Spring 2017.

Mini-Library on Quantum Information and Computation

This page is meant to be a collection of representative and available references for the study and research of the **theoretica** as possible and will be regularly maintained. Send me an email if you have any good suggestion.

Expository Writings and Lecture Notes by myself

- **Tutorial at POPL 2021**: An Invitation to the Intersection of Quantum Computing and Programming Languages
  - (Part I) A brief introduction to quantum computing and potential roles of programming languages
  - (Part II) A mini-course on the verification of quantum while languages based on quantum Hoare logic
  - (Part III) A discussion of existing and possible research directions at the intersection of quantum computing an
- **Lecture Notes (Fall 2019)**
  - Quantum Approximate Optimization Algorithm (QAOA)
  - Introduction to Quantum Hoare Logic (slides)
- **Lecture Notes (Fall 2018)**
  - Quantum Interactive Proofs and QIP=PSPACE
  - Quantum Algorithms for Linear Equation Systems
  - Quantum Algorithms for Semidefinite Programs

Scientific Reports from Relevant Research Communities

- National Academies of Sciences, Engineering, and Medicine. 2019. Quantum Computing: Progress and Prospects. Wa
- National Academies of Sciences, Engineering, and Medicine. 2020. Manipulating Quantum Systems: An Assessment
- Quantum Frontiers Report on community input to the Nation's Strategy for Quantum Information Science, October,
- Next Steps in Quantum Computing: Computer Science's Role: Computing Community Consortium Workshop Report
- More Reports at Quantum|Gov.

General Study: Courses, Lecture Notes & Textbooks

- **Self-learning Materials for Beginners**
  - Why now is the right time to study quantum computing by A. Harrow.
  - S. Aaronson: @UWaterloo Quantum Computing since Democritus
  - M. Nielsen's Quantum Computing for the determined: 22 short (5-15 mins) youtube videos, each explaining a ba
  - 12th Canadian Summer School on Quantum Information Lecture Notes YouTubes

About the Speaker:

Wu: assistant professor at UMD **working on quantum computing from CS perspective in general.**

Teaching in Q. Computing

Past Courses

This is a collection of courses that I have taught in the past for your references. Please be cautious as thes

University of Maryland, College Park (2017 - present)

- Complexity Theory (CMSC 652): graduate-level theory core course
  - Fall 2017
- Introduction to Quantum Computing (CMSC/PHYS 457): undergraduate-level introduction to quan
  - Spring 2018, Spring 2020, Spring 2021
- Introduction to Quantum Information Processing (CMSC 657): graduate-level introduction to quan
  - Fall 2018, Fall 2019

University of Oregon (2015 - 2017)

- Intermediate Data Structure (CIS 313): undergraduate CS major theory course.
  - Winter 2016, Fall 2016, Winter 2017.
- Introduction to Quantum Information Processing (CIS 410/510): senior undergraduate / graduate
  - Spring 2016, Spring 2017.

Mini-Library on Quantum Information and Computation

This page is meant to be a collection of representative and available references for the study and research of the **theoretica** as possible and will be regularly maintained. Send me an email if you have any good suggestion.

Expository Writings and Lecture Notes by myself

- **Tutorial at POPL 2021**: An Invitation to the Intersection of Quantum Computing and Programming Languages
  - (Part I) A brief introduction to quantum computing and potential roles of programming languages
  - (Part II) A mini-course on the verification of quantum while languages based on quantum Hoare logic
  - (Part III) A discussion of existing and possible research directions at the intersection of quantum computing an
- **Lecture Notes (Fall 2019)**
  - Quantum Approximate Optimization Algorithm (QAOA)
  - Introduction to Quantum Hoare Logic (slides)
- **Lecture Notes (Fall 2018)**
  - Quantum Interactive Proofs and QIP=PSPACE
  - Quantum Algorithms for Linear Equation Systems
  - Quantum Algorithms for Semidefinite Programs

Scientific Reports from Relevant Research Communities

- National Academies of Sciences, Engineering, and Medicine. 2019. Quantum Computing: Progress and Prospects. Wa
- National Academies of Sciences, Engineering, and Medicine. 2020. Manipulating Quantum Systems: An Assessment
- Quantum Frontiers Report on community input to the Nation's Strategy for Quantum Information Science, October,
- Next Steps in Quantum Computing: Computer Science's Role: Computing Community Consortium Workshop Report
- More Reports at Quantum|Gov.

General Study: Courses, Lecture Notes & Textbooks

- **Self-learning Materials for Beginners**
  - Why now is the right time to study quantum computing by A. Harrow.
  - S. Aaronson: @UWaterloo Quantum Computing since Democritus
  - M. Nielsen's Quantum Computing for the determined: 22 short (5-15 mins) youtube videos, each explaining a b
  - 12th Canadian Summer School on Quantum Information Lecture Notes YouTubes

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

Outline

(1) Introduction to Quantum Computing and Potential Roles of Programming Languages (25 min + 5 Q & A)

(2) A Mini-Course of Quantum Hoare Logic on Quantum While Language (30 min + 5 Q & A)

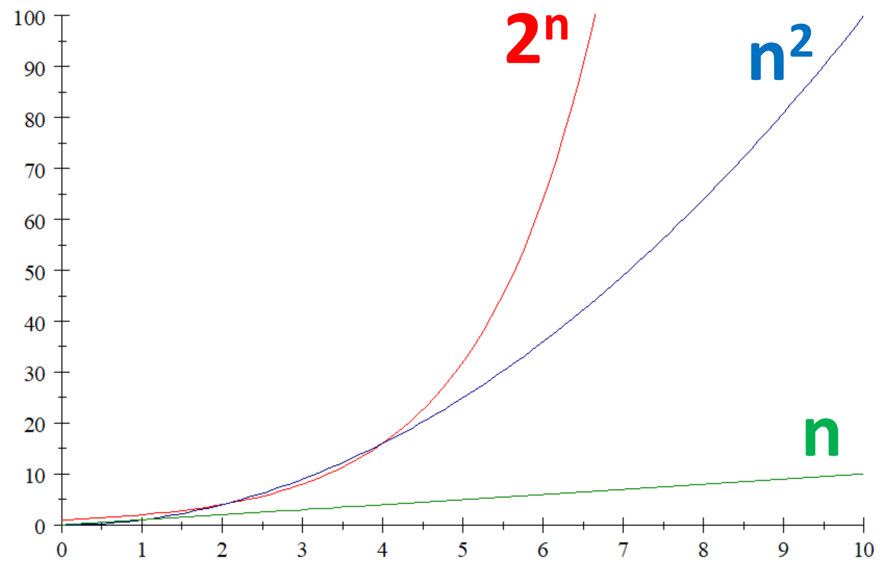
(3) Discussion on existing and potential Programming Language research opportunities (20 min + 5 Q & A)

Reference: tutorial slides and some references are available at https://www.cs.umd.edu/~xwu/mini_lib.html

What Quantum Computing is NOT

Credit: Scott Aaronson

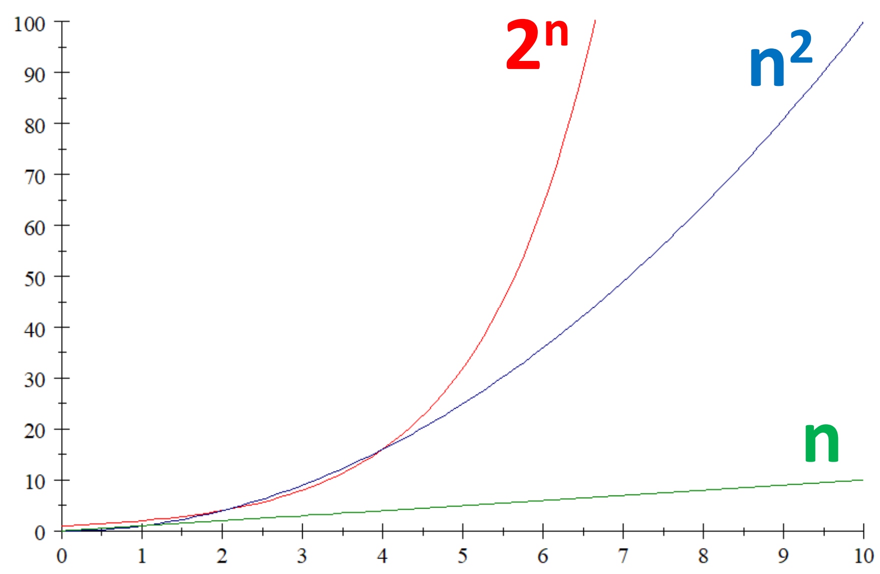
It Isn't Just Today's Computers But Smaller or Faster



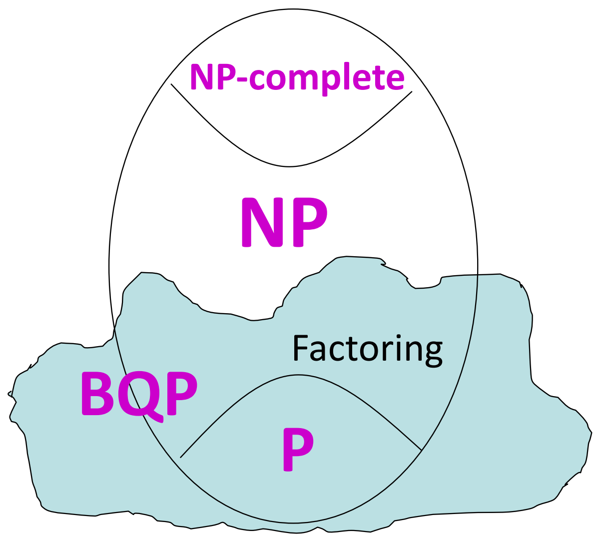
What Quantum Computing is NOT

Credit: Scott Aaronson

It Isn't Just Today's Computers But Smaller or Faster



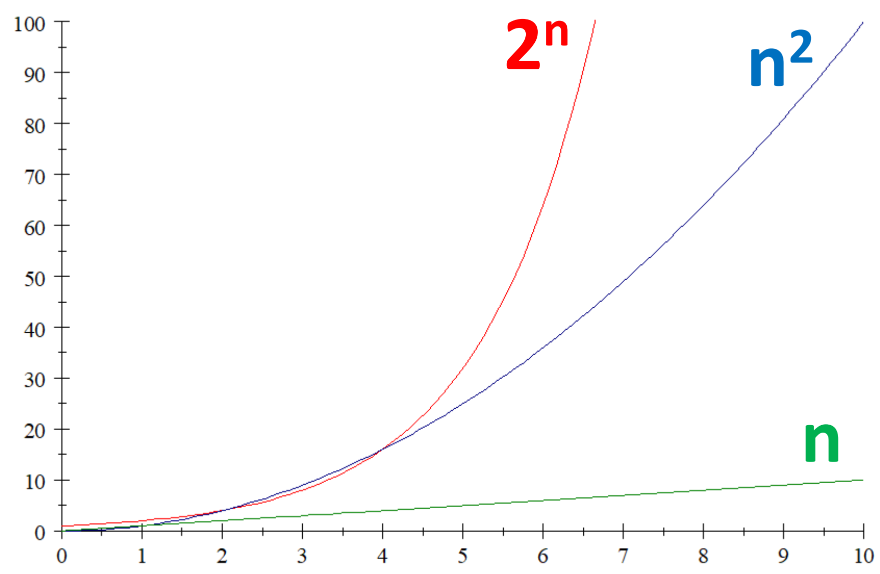
It Isn't A Magic Bullet That Solves All Problems Instantly



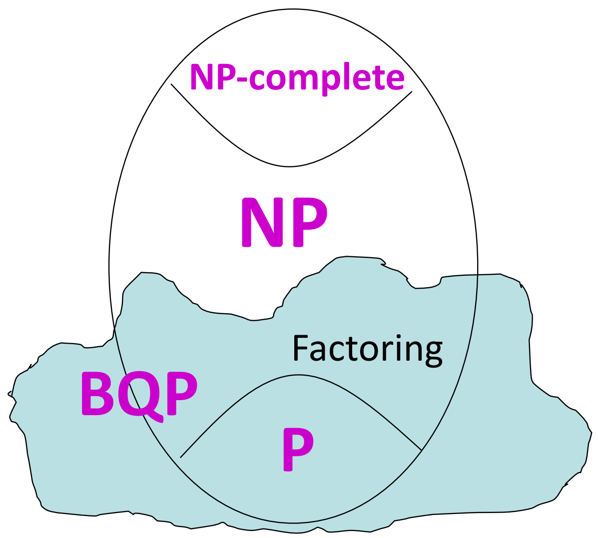
What Quantum Computing is NOT

Credit: Scott Aaronson

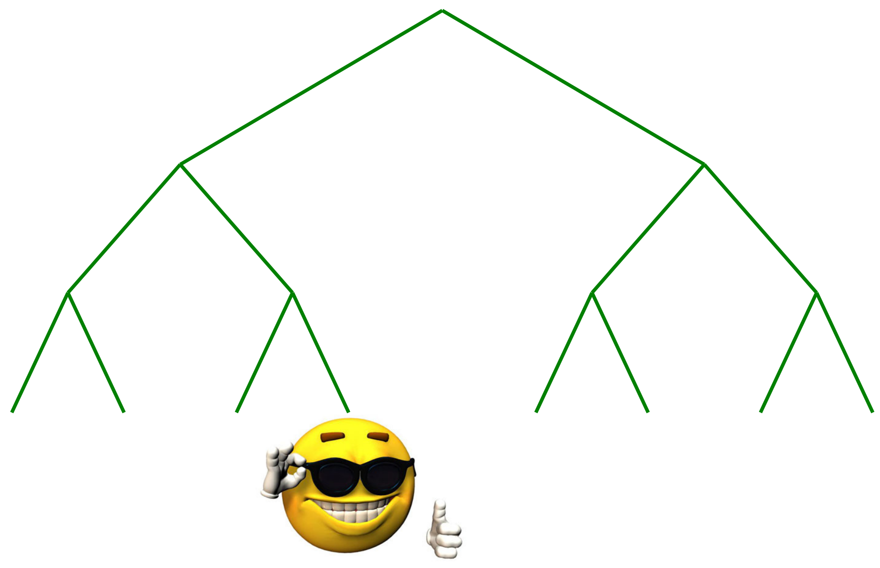
It Isn't Just Today's Computers But Smaller or Faster



It Isn't A Magic Bullet That Solves All Problems Instantly



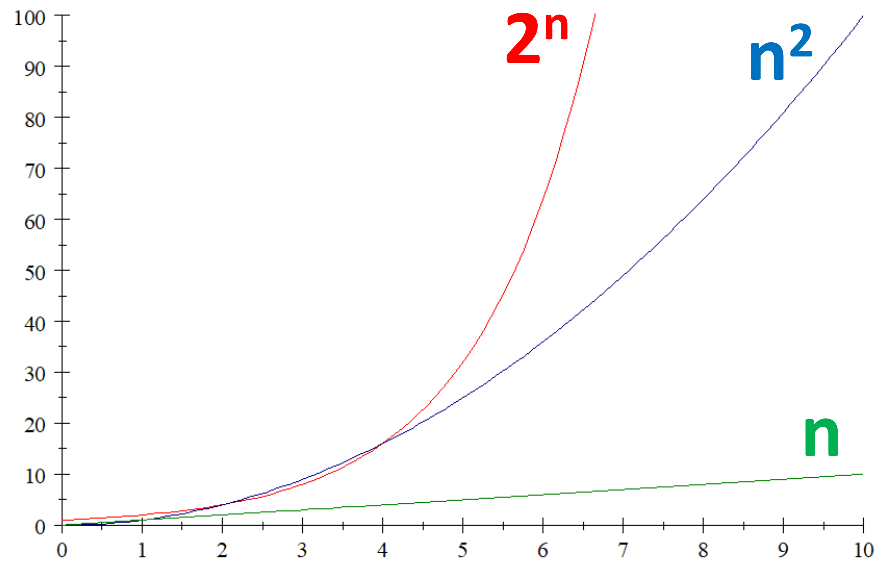
It Isn't A Simple Matter of Trying All Possible Answers In Parallel



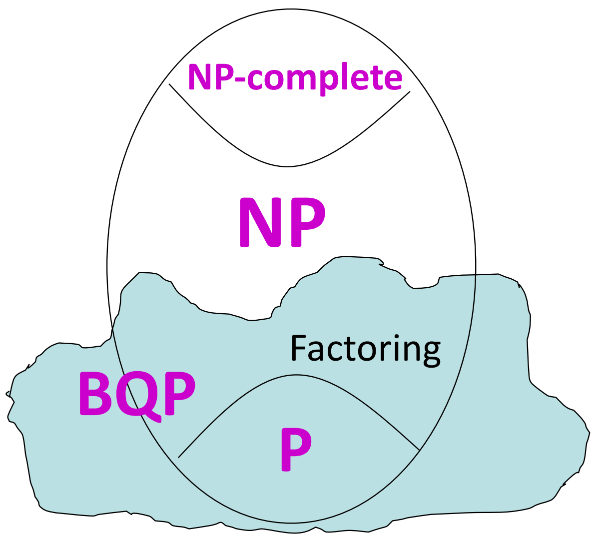
What Quantum Computing is NOT

Credit: Scott Aaronson

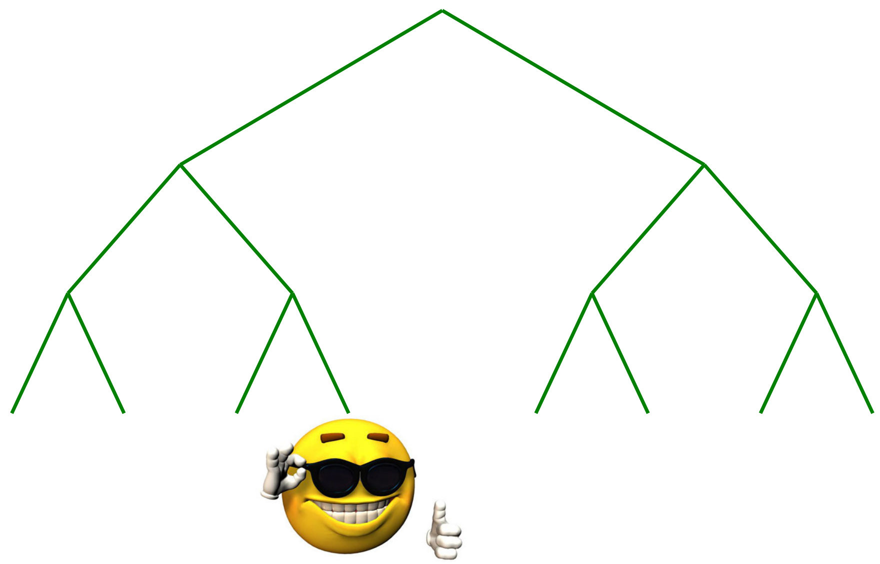
It Isn't Just Today's Computers But Smaller or Faster



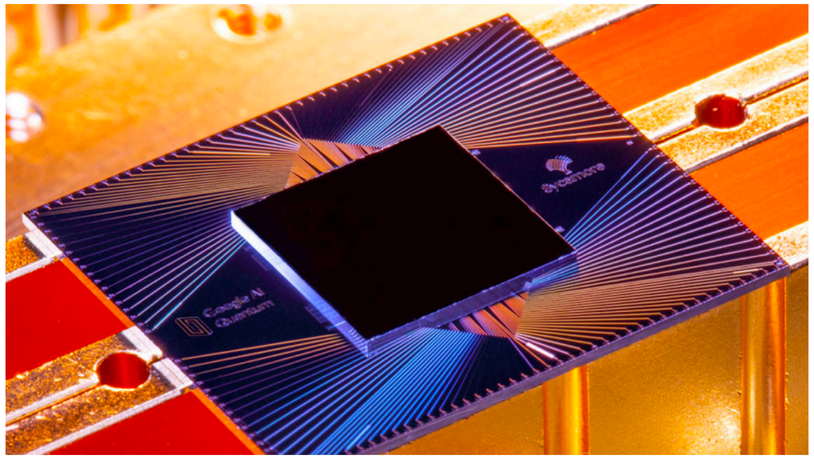
It Isn't A Magic Bullet That Solves All Problems Instantly



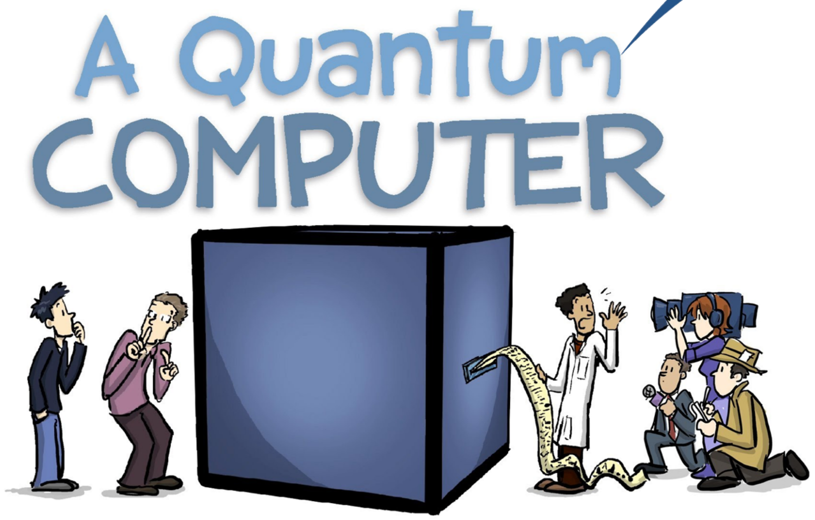
It Isn't A Simple Matter of Trying All Possible Answers In Parallel



But Nor Is It Science Fiction



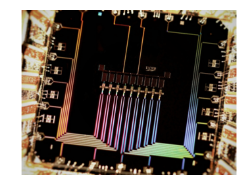
Roadmap in 2010s



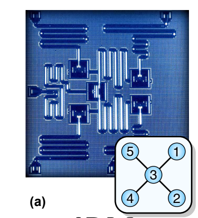
(2012)

Ion-Trap (UMD)

Super-conducting



Google

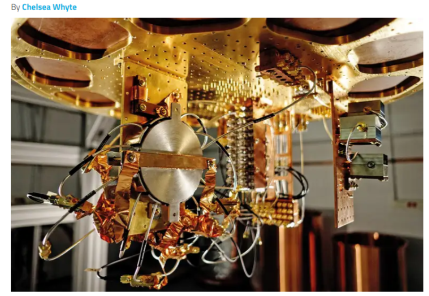


IBM

(2017)

IBM will soon launch a 53-qubit quantum computer

Frederic Lardinois @fredericl / 8:00 am EDT • September 18, 2019

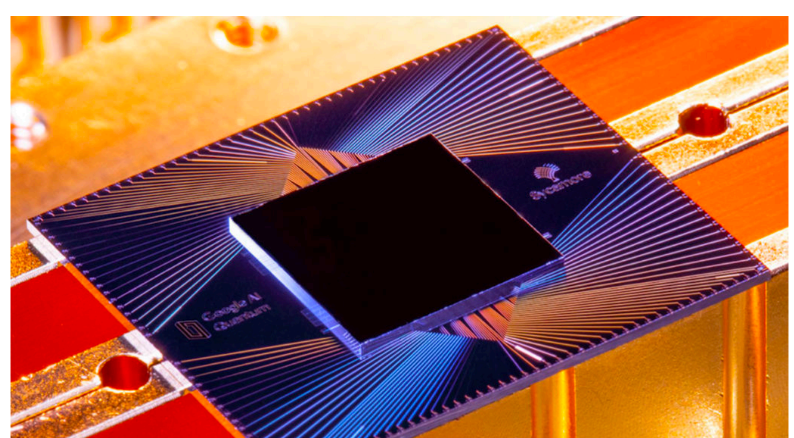


Google has reached quantum supremacy – here's what it should do next

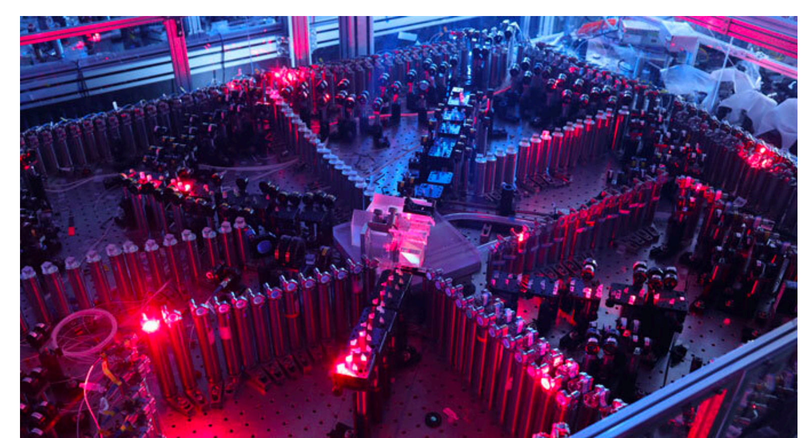
TECHNOLOGY | ANALYSIS 26 September 2019

By Chelsea Whyte

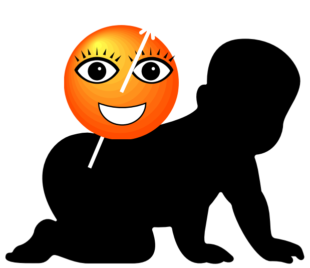
(2019)



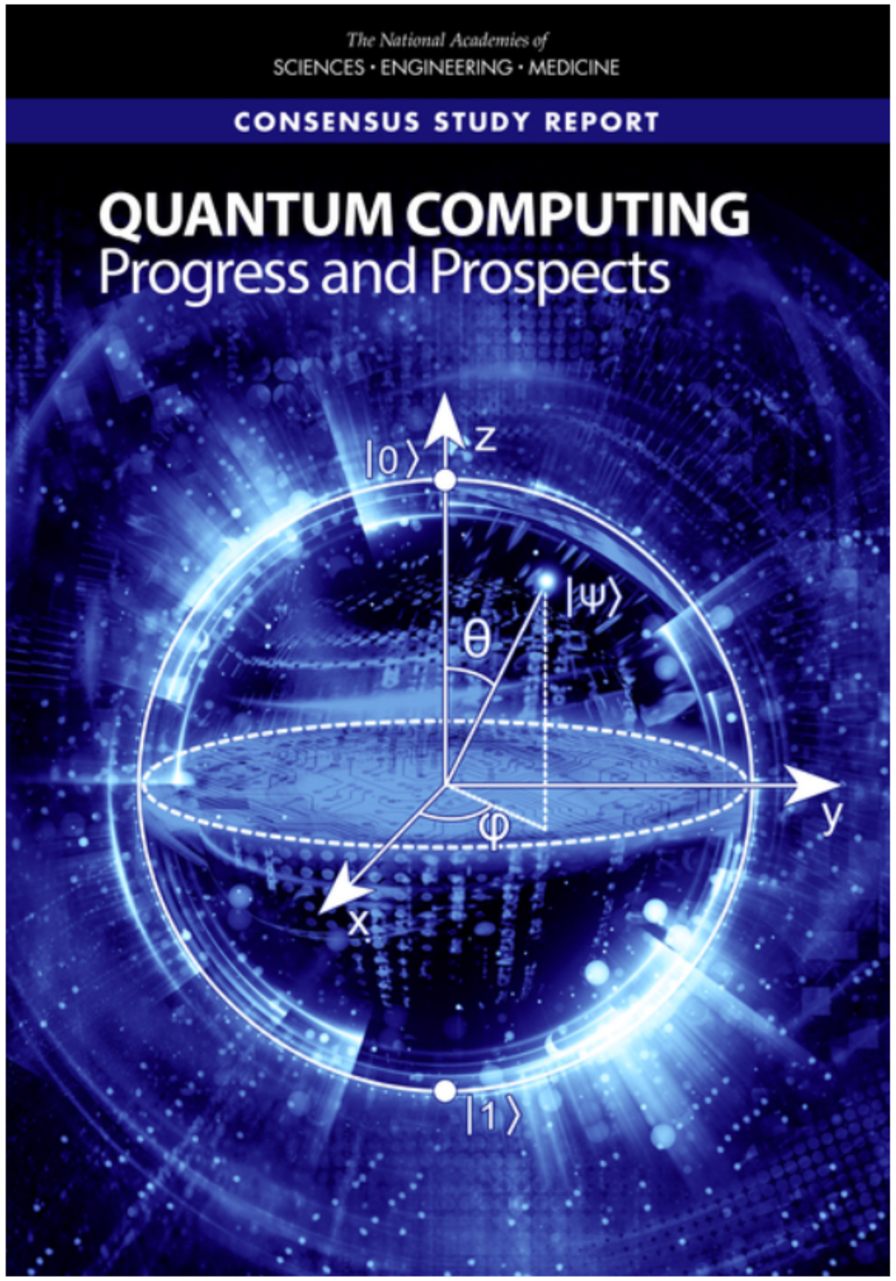
Google Supremacy: RCS (2019)



USTC: Boson Sampling (2020)

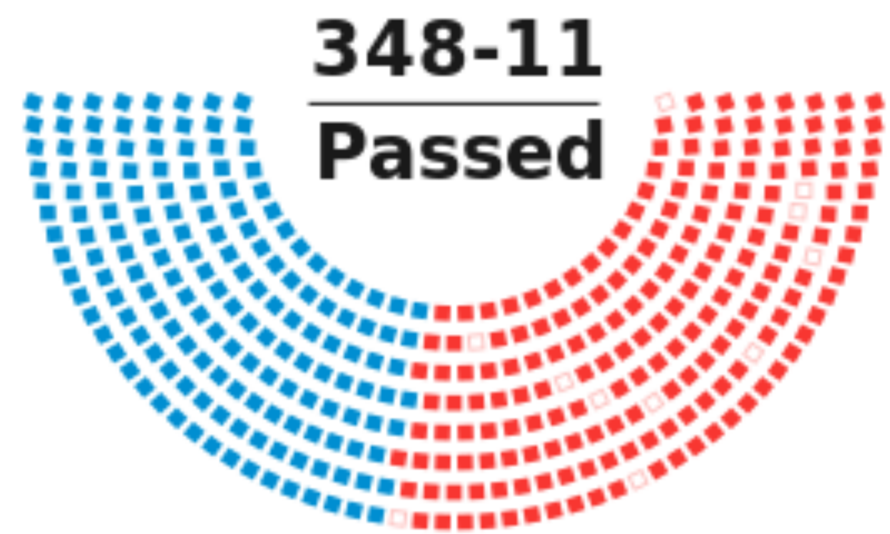


Surge of Interests from Gov, Academia, & Industry



MARK A. HOROWITZ,
Stanford University, *Chair*
ALÁN ASPURU-GUZIK,
University of Toronto
DAVID D. AWSCHALOM,
University of Chicago
BOB BLAKLEY,
Citigroup
DAN BONEH,
Stanford University
SUSAN N. COPPERSMITH,
University of Wisconsin, Madison
JUNGSANG KIM,
Duke University
JOHN M. MARTINIS,
Google, Inc.
MARGARET MARTONOSI,
Princeton University
MICHELE MOSCA,
University of Waterloo
WILLIAM D. OLIVER,
Massachusetts Institute of Technology
KRYSTA SVORE,
Microsoft Research
UMESH V. VAZIRANI,
University of California, Berkeley

H. R. 6227
National Quantum Initiative Act

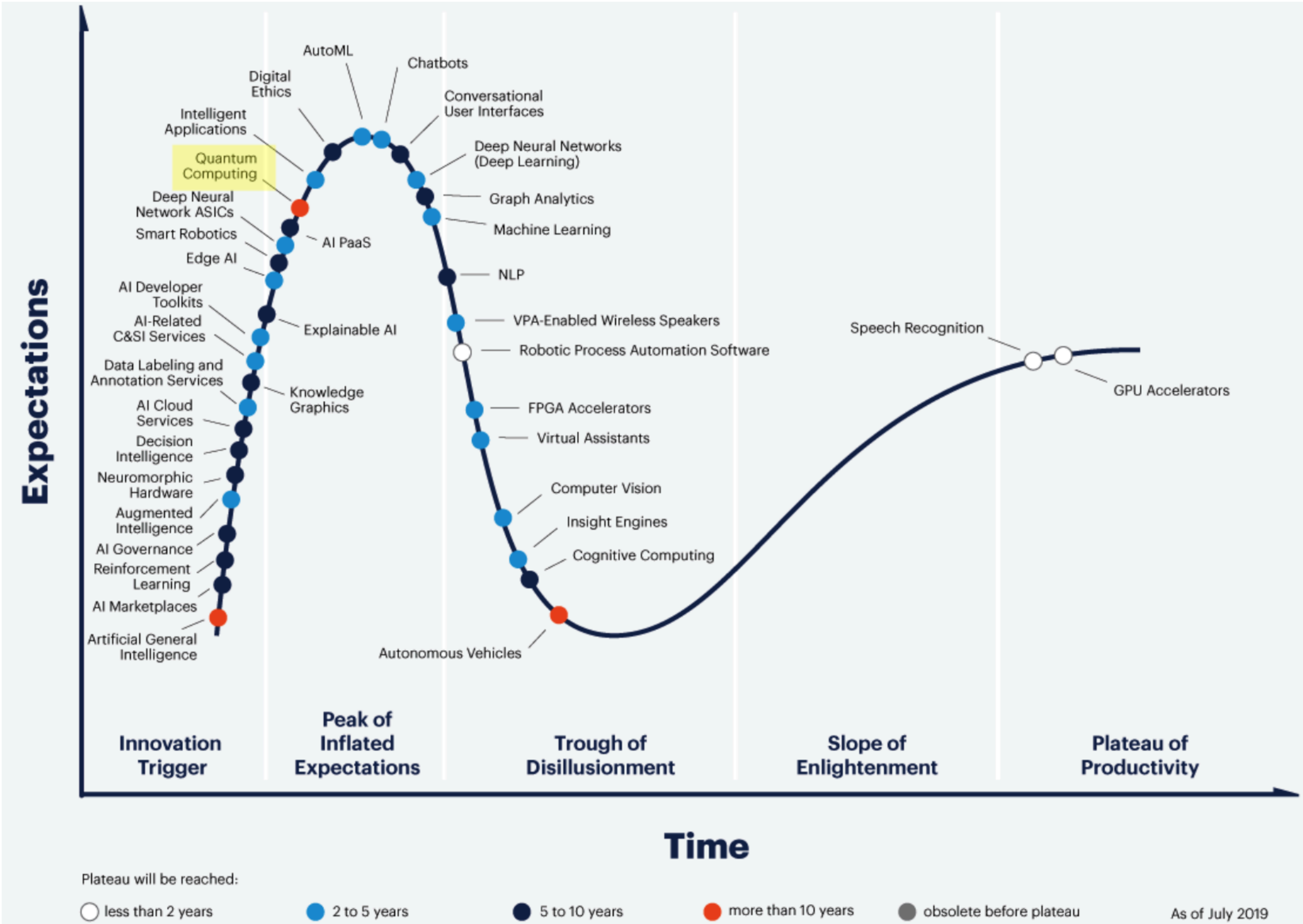


House Vote #442 -- 12/19/18

Gov: US (NSF, DOE + National Labs, DoD, NIST), China, Europe, ….
Industry: Google, IBM, Microsoft, Amazon, Alibaba, Tecent, Baidu, ….
Academia: #faculty in quantum computing ++

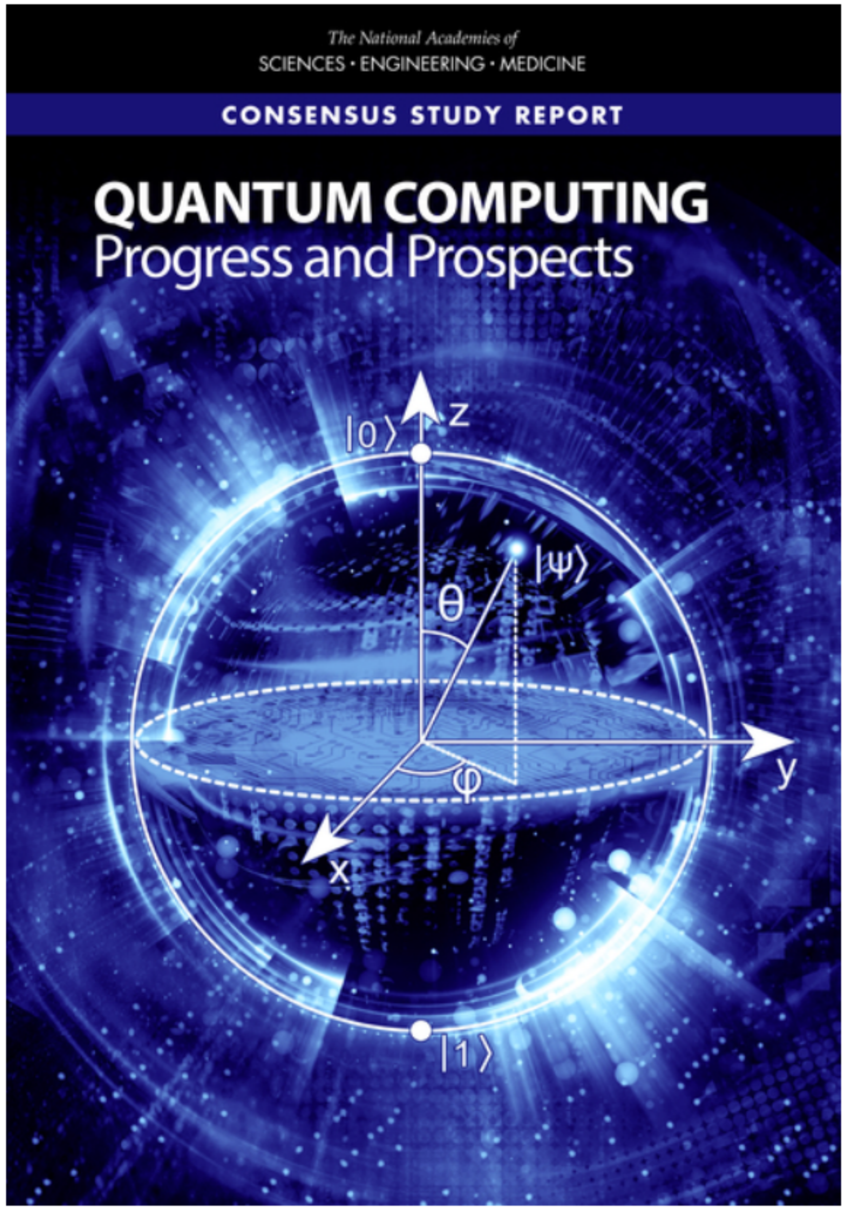
US GOV Policy & Efforts: ⟨quantum|gov⟩

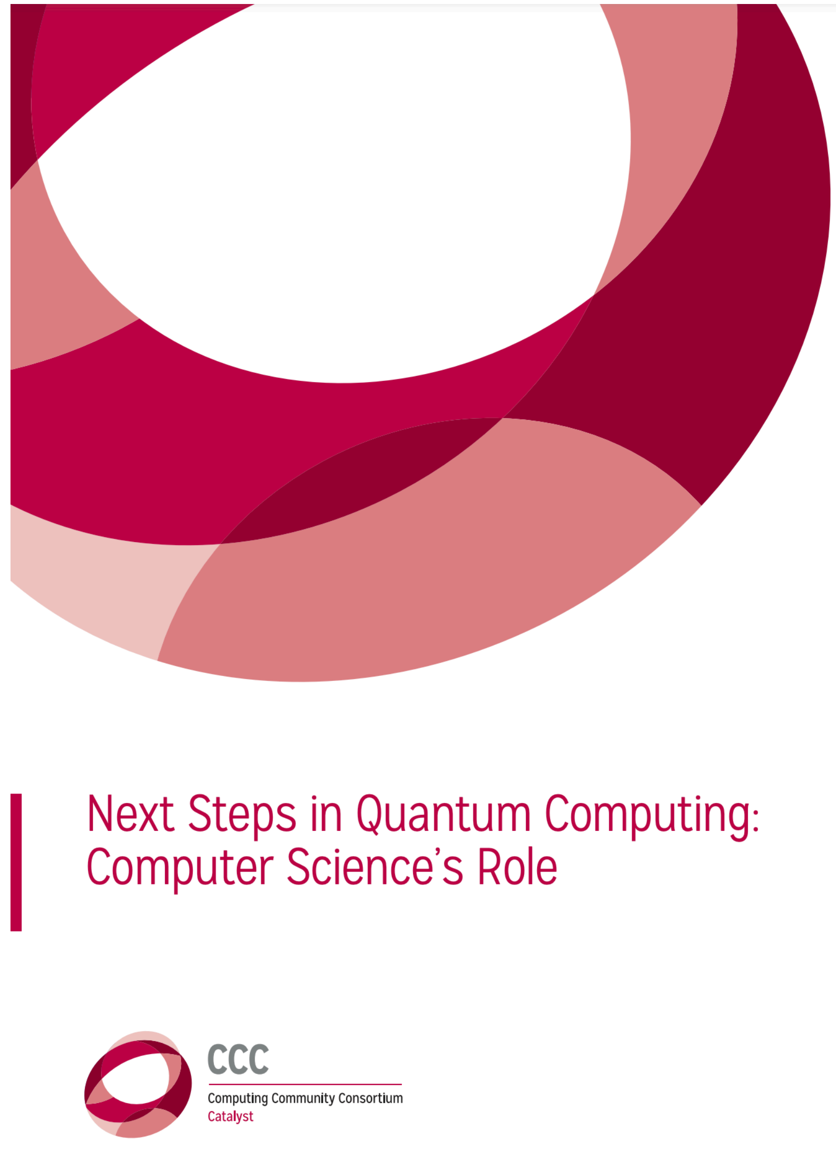
Quantum Computing: still too early to call!

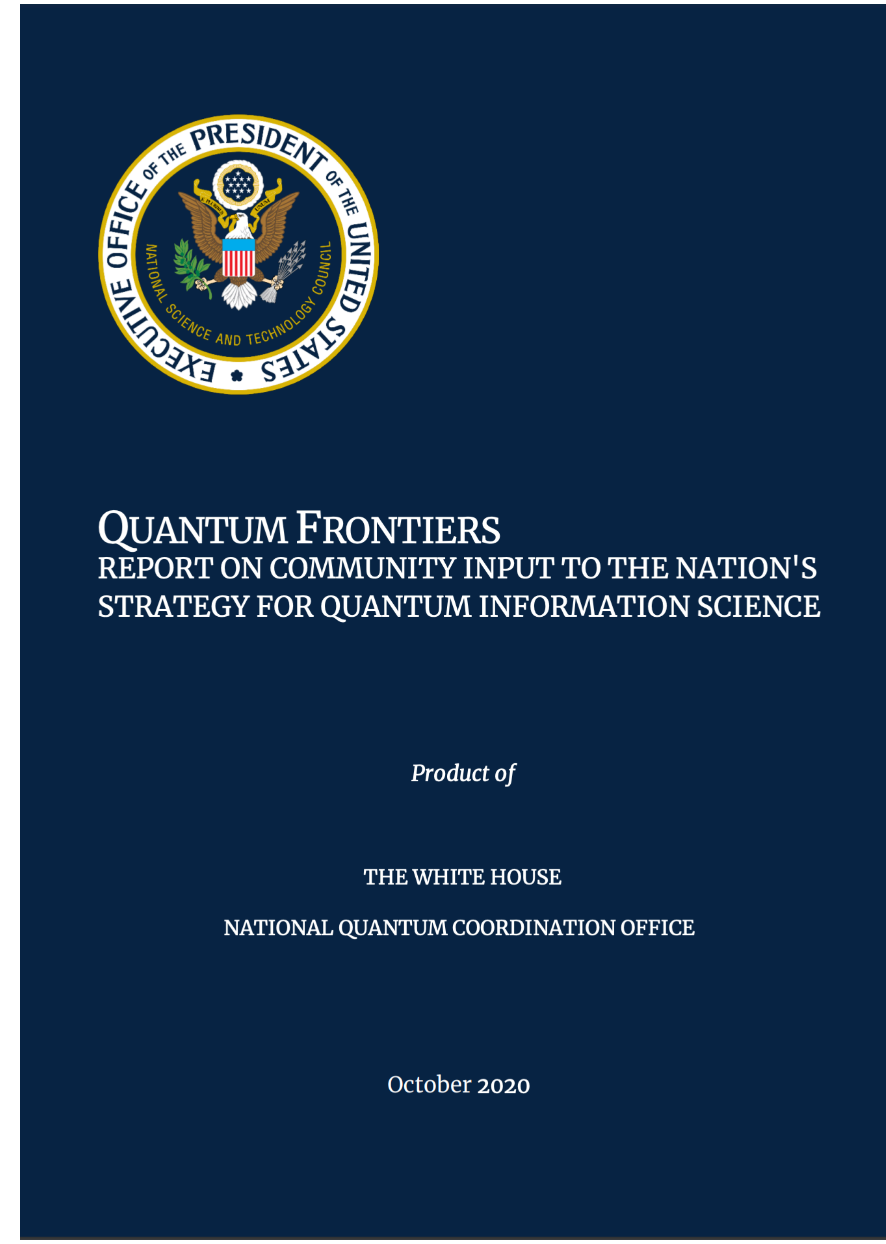


The 2019 Gartner Hype Cycle for Artificial Intelligence, with quantum computing highlighted in yellow. Credit: Gartner

Scientific Reports from relevant research communities

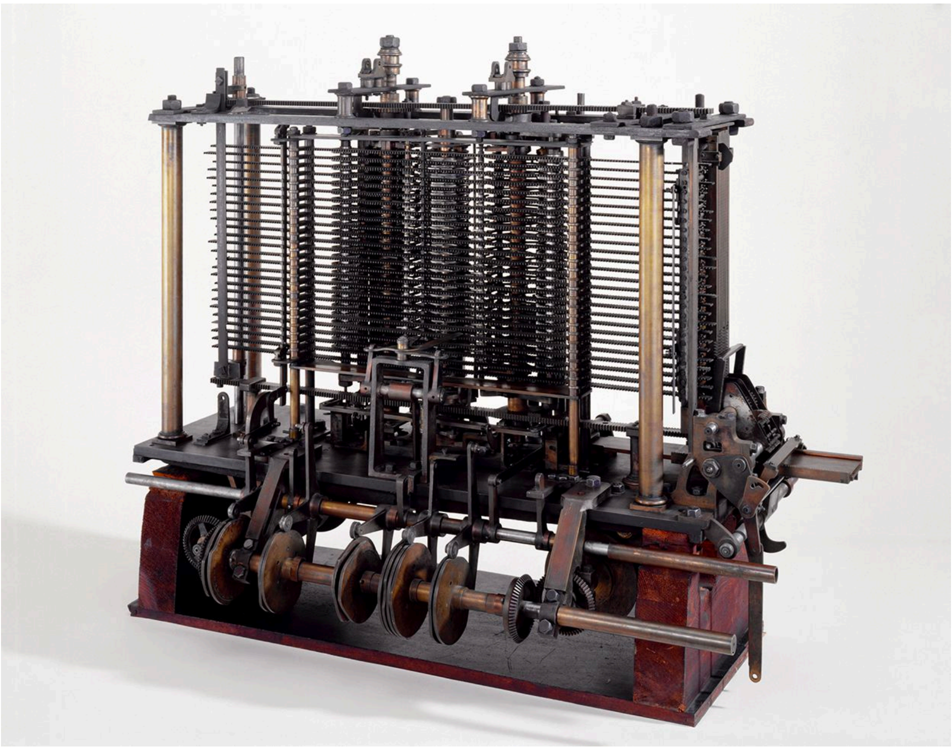






Reference: links are available at https://www.cs.umd.edu/~xwu/mini_lib.html

What is Quantum Computing?



A Mechanical Computer

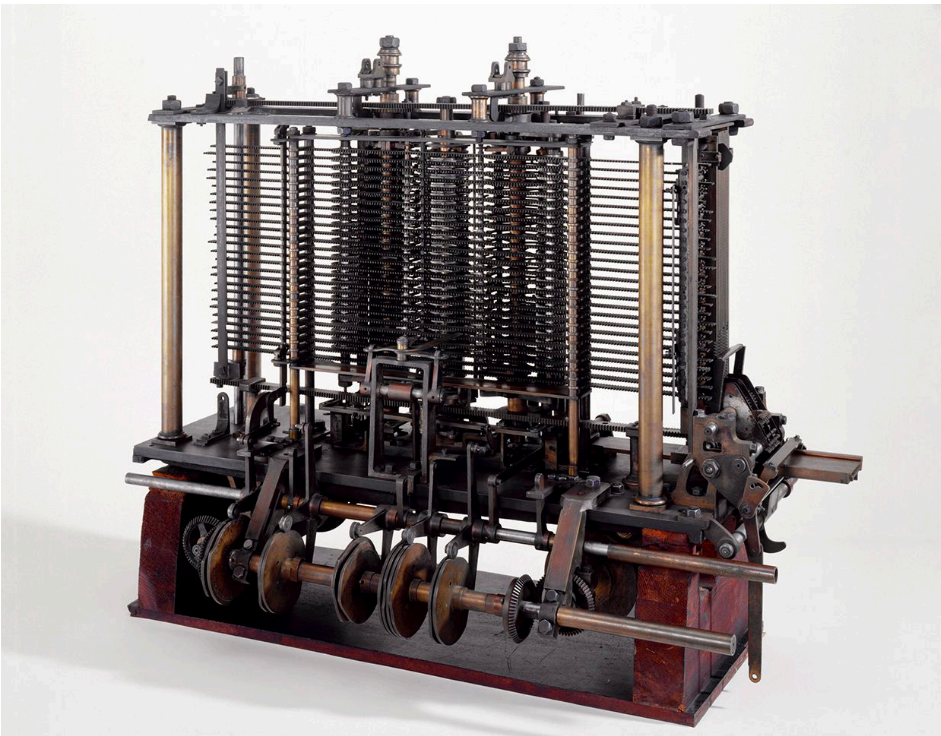
An **Operation** O – – – –> A **Physical Evolution** P

Computation:

Evolution of the Machine: $P_1, P_2, P_3, \cdots$

The accumulative evolution carries some computation!

What is Quantum Computing?



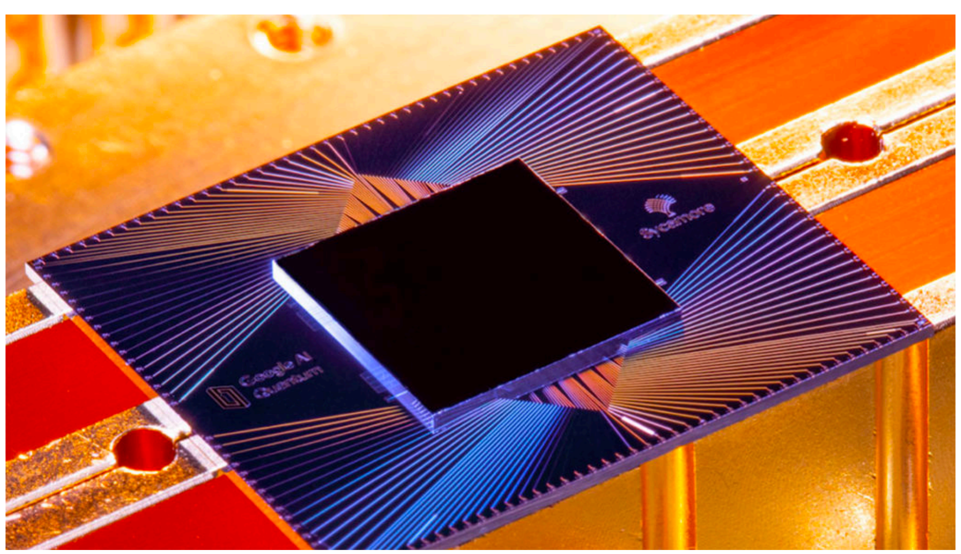
A Mechanical Computer

An **Operation** O – – – –> A **Physical Evolution** P

Computation:

Evolution of the Machine: $P_1, P_2, P_3, \cdots$

The accumulative evolution carries some computation!



A Quantum Computer

An **Operation** O –> A **Quantum Physical Evolution** Q

Computation:

Evolution of the Machine: $Q_1, Q_2, Q_3, \cdots$

The accumulative evolution carries some computation!

What is Quantum Computing good at?

Assume a *unit* operation requires a *unit* time on respective machines.

What is Quantum Computing **good** at?

Assume a *unit* operation requires a *unit* time on respective machines.

Computation can be carried out by $P_1, P_2, \cdots, P_T$

Classical Computing (T)

Computation can be carried out by $Q_1, Q_2, \cdots, Q_T$

Quantum Computing (T)

What is Quantum Computing **good** at?

Assume a *unit* operation requires a *unit* time on respective machines.

Computation can be carried out by $P_1, P_2, \cdots, P_T$

Classical Computing (T)

Computation can be carried out by $Q_1, Q_2, \cdots, Q_T$

Quantum Computing (T)

What is Quantum Computing good at?

Assume a *unit* operation requires a *unit* time on respective machines.

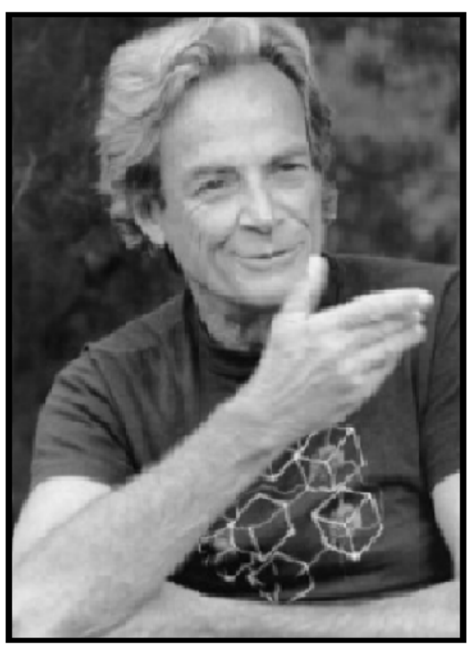
Computation can be carried out by $P_1, P_2, \cdots, P_T$

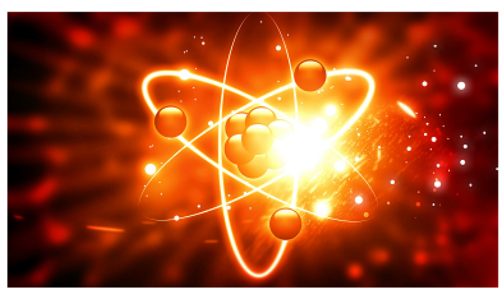
Classical Computing (T)

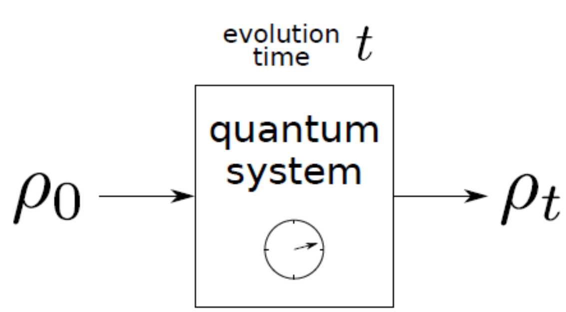
Computation can be carried out by $Q_1, Q_2, \cdots, Q_T$

Quantum Computing (T)

Quantum Simulation







Nature isn't classical, and if you want to make a simulation of Nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem, because it doesn't look so easy.

Richard Feynman, 1982

Simulating quantum systems is critical for the scientific discovery for natural science include physics, chemistry, biology, material science, and so on. And nowadays, it consumes a significant amount of our HPC computing power.

What is Quantum Computing good at?

31074182404900437213507500358885679300373460228427275457201619488232064405180815045563468296717232867824379162728380334154710731085019195485290073377248227835257423864540146917366024776523466091

=

163473364580925384844313388386509085984178367003309231218111085238933310010450815121211816751157

×

190087128166482211312685157393541397547189678996851549366663853908802710380210449895719126146557 1

000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000001000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000

- Linear systems
- Graph problems (minimum spanning tree, connectivity, shortest path, triangle finding, etc.)
- Formula evaluation
- Decomposing groups (abelian, dihedral, etc.)
- ……

What is Quantum Computing **good** at?

3107418240490043721350750035888567
9300373460228427275457201619488232
0644051808150455634682967172328678
2437916272838033415471073108501919
5485290073377248227835257423864540
14691736602477652346609
=
163473364580925384844313388386509
085984178367003309231218111085238
9333100104508151212118167511579
×
190087128166482211312685157393541
397547189678996851549366663853908
8027103802104498957191261465571

00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000001000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000

- Linear systems
- Graph problems (minimum spanning tree, connectivity, shortest path, triangle finding, etc.)
- Formula evaluation
- Decomposing groups (abelian, dihedral, etc.)
- ……

It was a good surprise that quantum physics can help solve classical problems that look nothing like quantum physics at all!

Any high-level intuition why?

What is Quantum Computing good at?

3107418240490043721350750035888567
9300373460228427275457201619488232
0644051808150455634682967172328678
2437916272838033415471073108501919
5485290073377248227835257423864540
14691736602477652346609
=
1634733645809253848443133883865090
8598417836700330923121811108523893
33100104508151212118167511579
×
1900871281664822113126851573935413
9754718967899685154936666385390880
2710380210449895719126146557 1

000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000001000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000

- Linear systems
- Graph problems (minimum spanning tree, connectivity, shortest path, triangle finding, etc.)
- Formula evaluation
- Decomposing groups (abelian, dihedral, etc.)
- ……

It was a good surprise that quantum physics can help solve classical problems that look nothing like quantum physics at all!

Any high-level intuition why?

Quantum Duality :

Particle + Wave

What is Quantum Computing good at?

3107418240490043721350750035888567
9300373460228427275457201619488232
0644051808150455634682967172328678
2437916272838033415471073108501919
5485290073377248227835257423864540
14691736602477652346609
=
163473364580925384844313388386509
085984178367003309231218111085238
9333100104508151212118167511579
×
190087128166482211312685157393541
397547189678996851549366663853908
8027103802104498957191261465571

- Linear systems
- Graph problems (minimum spanning tree, connectivity, shortest path, triangle finding, etc.)
- Formula evaluation
- Decomposing groups (abelian, dihedral, etc.)
- ……

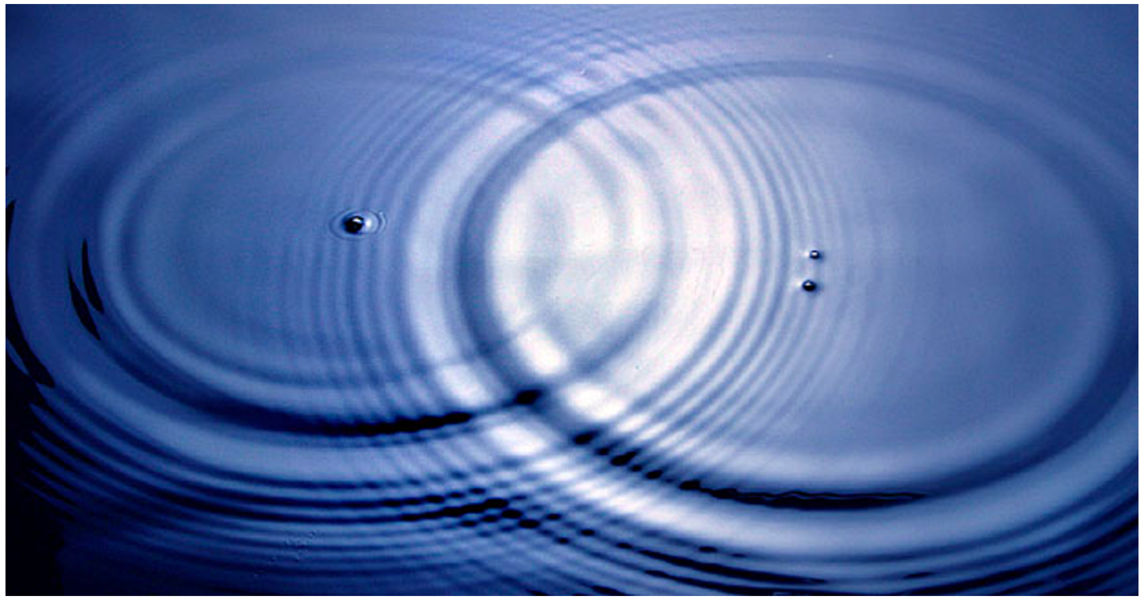
It was a good surprise that quantum physics can help solve classical problems that look nothing like quantum physics at all!

Any high-level intuition why?

Quantum Duality :

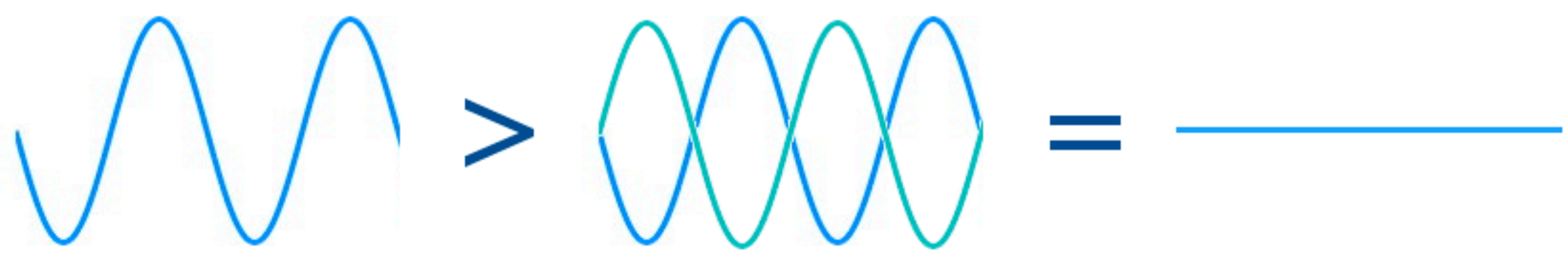
Particle + **Wave**

Interference of Waves :

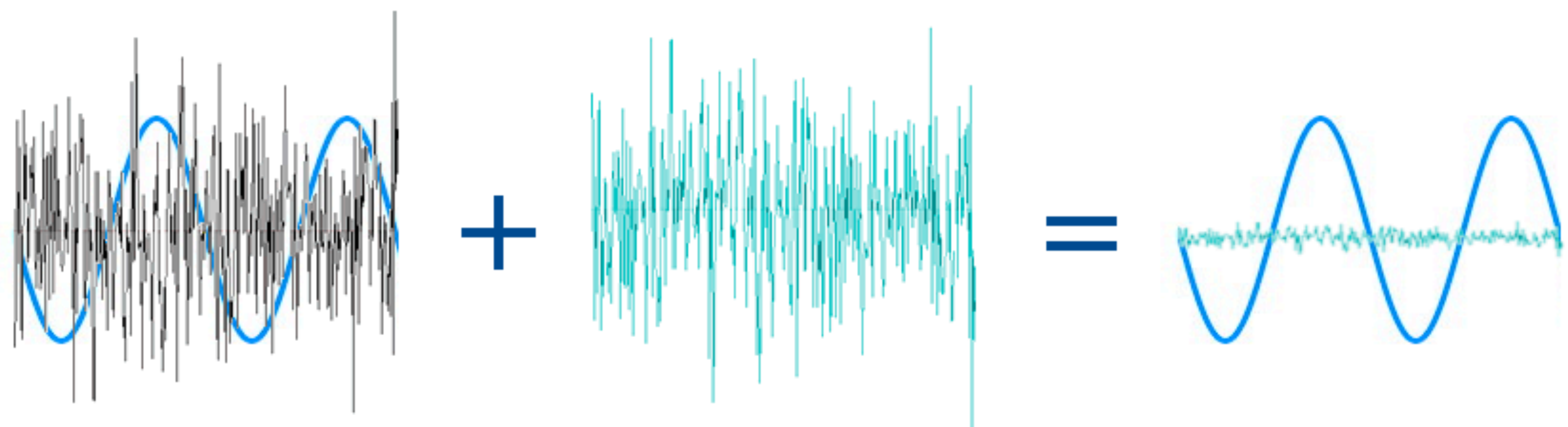


Make Interference Work:

Waves of equal amplitude and opposite phase cancel out

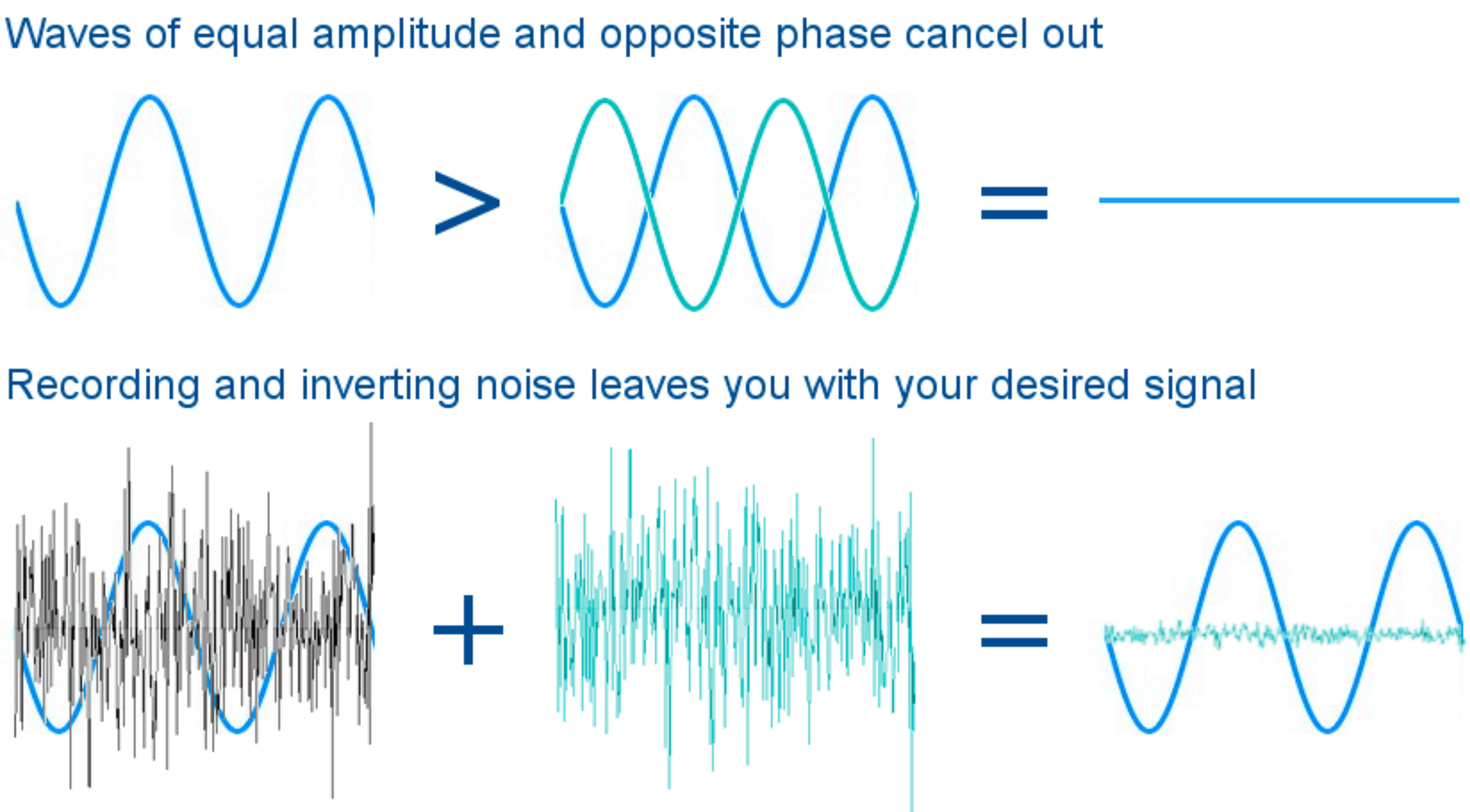


Recording and inverting noise leaves you with your desired signal



Active Noise-Canceling!

Make Interference Work:

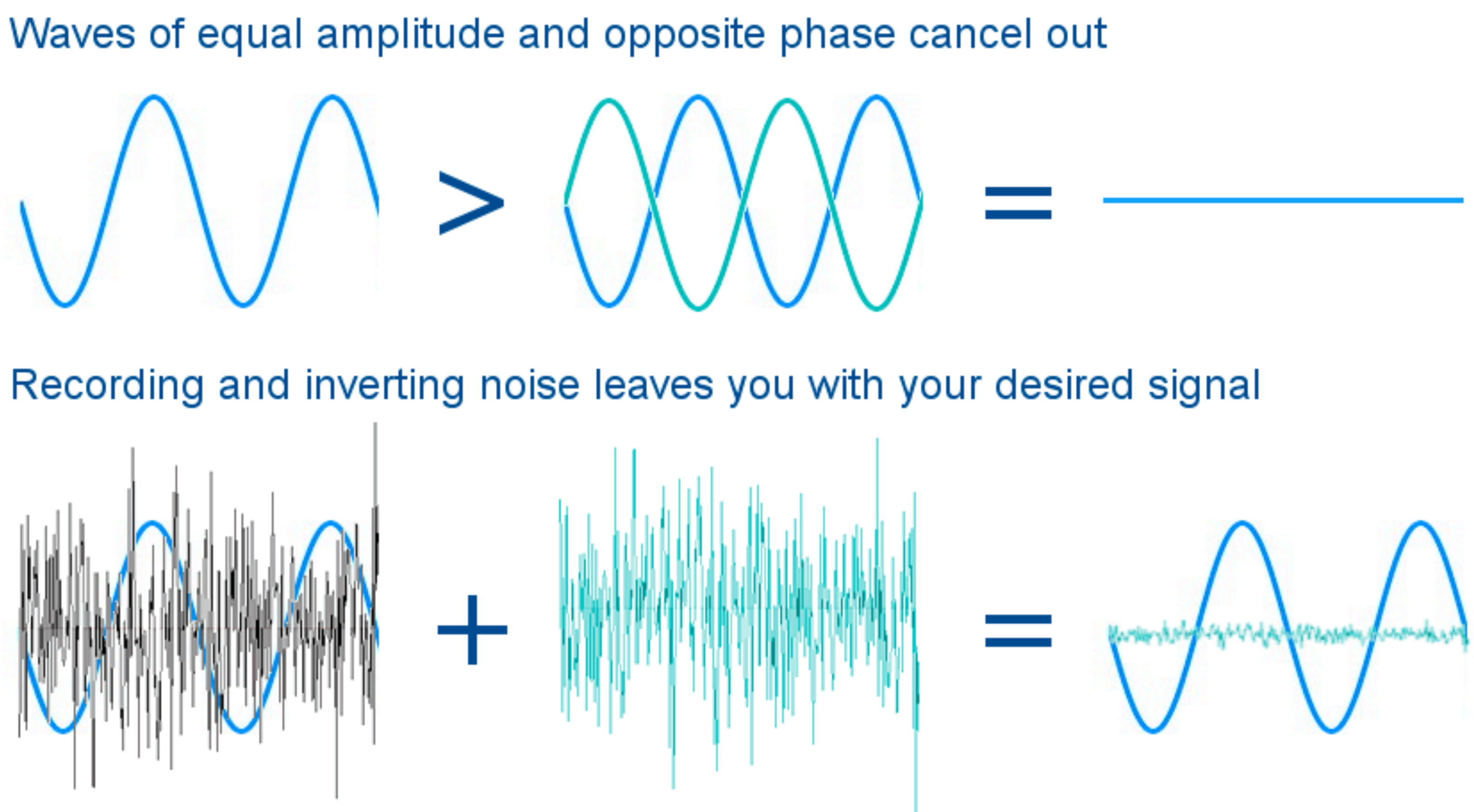


Active Noise-Canceling!

Make Interference Work for Computation:

Quantum Computation: Get computational paths leading to *incorrect* answers to **interfere destructively** and cancel each other out.

Make Interference Work:



Active Noise-Canceling!

Make Interference Work for Computation:

Quantum Computation: Get computational paths leading to *incorrect* answers to **interfere destructively** and cancel each other out.

Quantum vs Randomized:

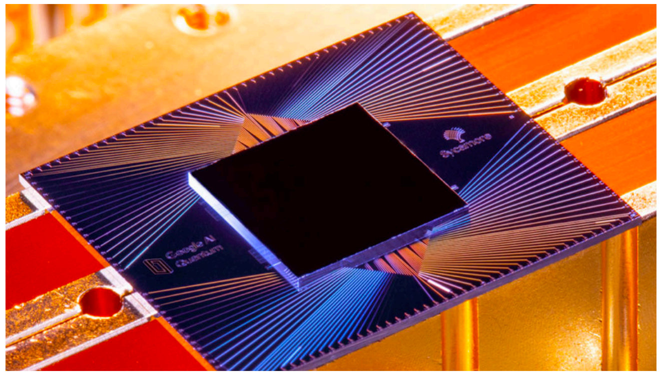
Randomized Computation: Probabilities of computational paths leading to *incorrect* answers only **add up**, never cancel out.

A Rough Timeline of Quantum Applications

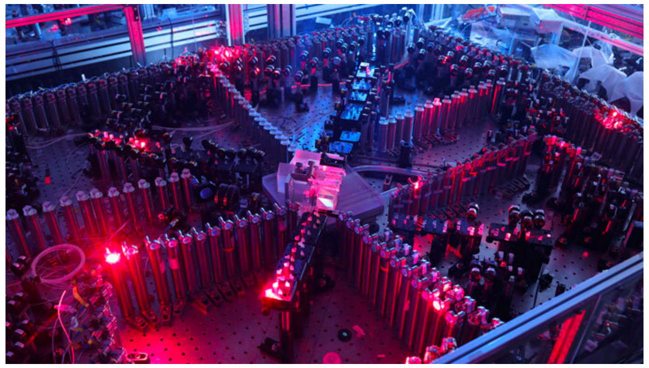
NOW: Quantum Supremacy

Computational tasks, *not necessarily useful*, which is feasible to implement w/ current q. machines, but hard to simulate by classical computation.

A **Milestone** Toward Useful Quantum Computation



Google: Random Circuit Sampling



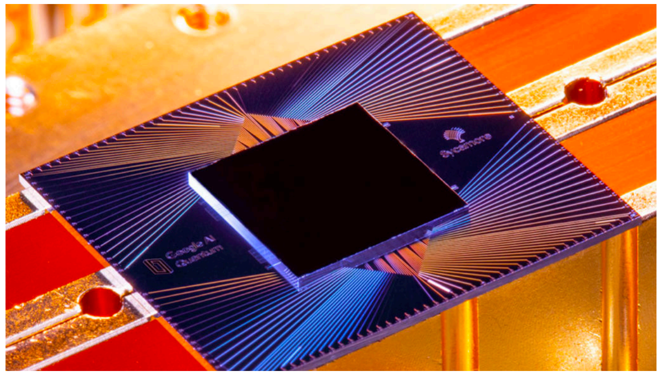
USTC: Boson Sampling

A Rough Timeline of Quantum Applications

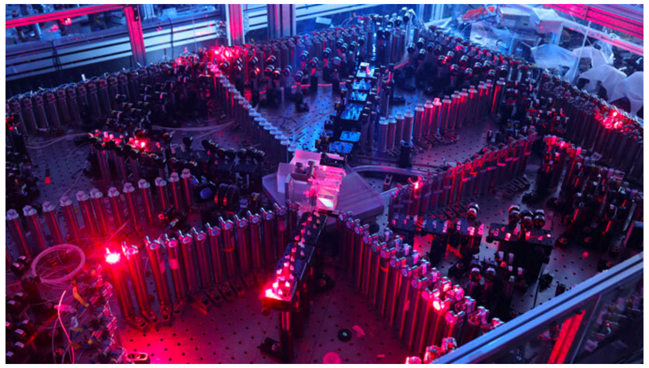
NOW: **Quantum Supremacy**

Computational tasks, *not necessarily useful*, which is feasible to implement w/ current q. machines, but hard to simulate by classical computation.

A **Milestone** Toward Useful Quantum Computation



Google: Random Circuit Sampling



USTC: Boson Sampling

NISQ: **Noise Intermediate-Scale Quantum machines** **~ near future**

50 ~ 200, ~ 1000 controllable but noisy qubits, no fault-tolerant qubits

Or special-purpose quantum machines, like analog quantum simulators

Quantum Simulation

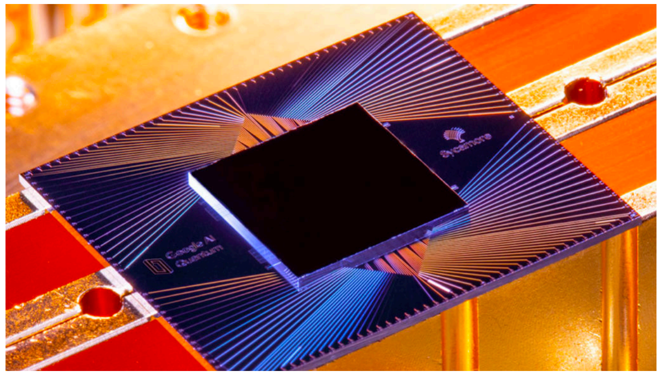
Variational Q. Methods

A Rough Timeline of Quantum Applications

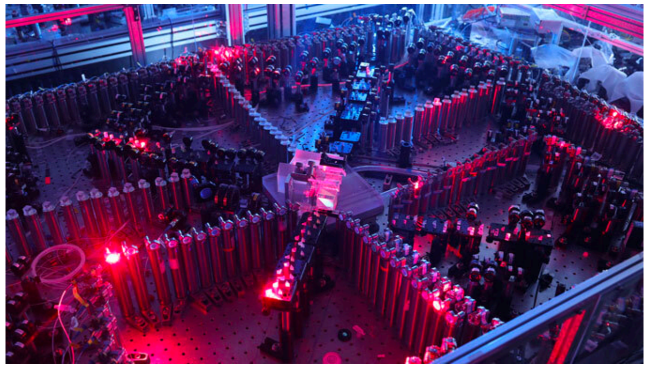
NOW: Quantum Supremacy

Computational tasks, *not necessarily useful*, which is feasible to implement w/ current q. machines, but hard to simulate by classical computation.

A **Milestone** Toward Useful Quantum Computation



Google: Random Circuit Sampling



USTC: Boson Sampling

NISQ: Noise Intermediate-Scale Quantum machines ~ near future

50 ~ 200, ~ 1000 controllable but noisy qubits, no fault-tolerant qubits

Or special-purpose quantum machines, like analog quantum simulators

Quantum Simulation

Variational Q. Methods

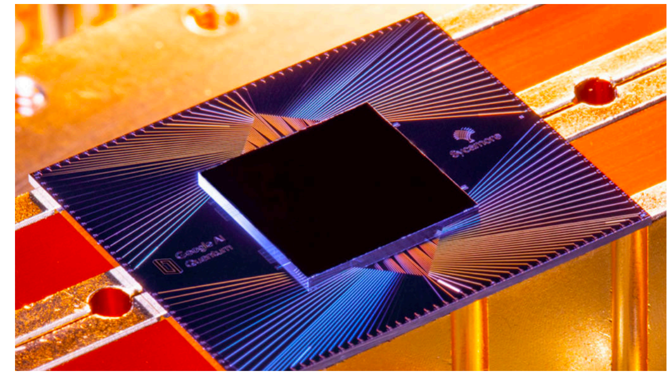
Other quantum applications not in the computation domain: *quantum sensing*, *quantum communication*

A Rough Timeline of Quantum Applications

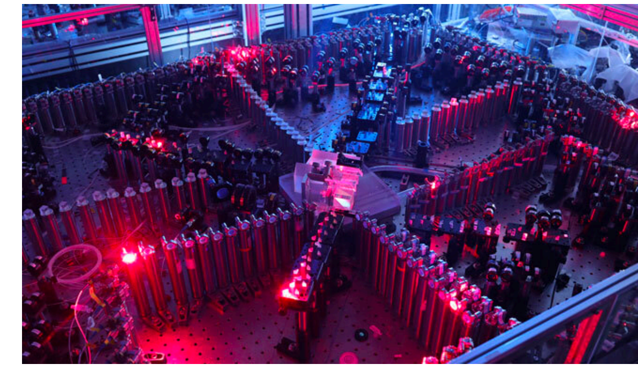
NOW: Quantum Supremacy

Computational tasks, *not necessarily useful*, which is feasible to implement w/ current q. machines, but hard to simulate by classical computation.

A **Milestone** Toward Useful Quantum Computation



Google: Random Circuit Sampling



USTC: Boson Sampling

NISQ: Noise Intermediate-Scale Quantum machines ~ near future

50 ~ 200, ~ 1000 controllable but noisy qubits, no fault-tolerant qubits

Or special-purpose quantum machines, like analog quantum simulators

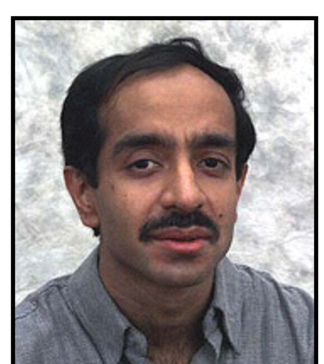
Quantum Simulation

Variational Q. Methods

Other quantum applications not in the computation domain: *quantum sensing, quantum communication*

Fault-Tolerant QC: ~ unknown future, a lot of uncertainty here

3107418240490043721350750035888567
9300373460228427275457201619488232
0644051808150455634682967172328678
2437916272838033415471073108501919
5485290073377248227835257423864540
14691736602477652346609
=
1634733645809253848443133883865090
859841783670033092312181110852389
3331001045081512121181675111579
×
1900871281664822113126851573935413
975471896789968515493666638539088
0271038021044989571912614655571



000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000001000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000

- Linear systems
- Graph problems (minimum spanning tree, connectivity, shortest path, triangle finding, etc.)
- Formula evaluation
- Decomposing groups (abelian, dihedral, etc.)
- ……

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!

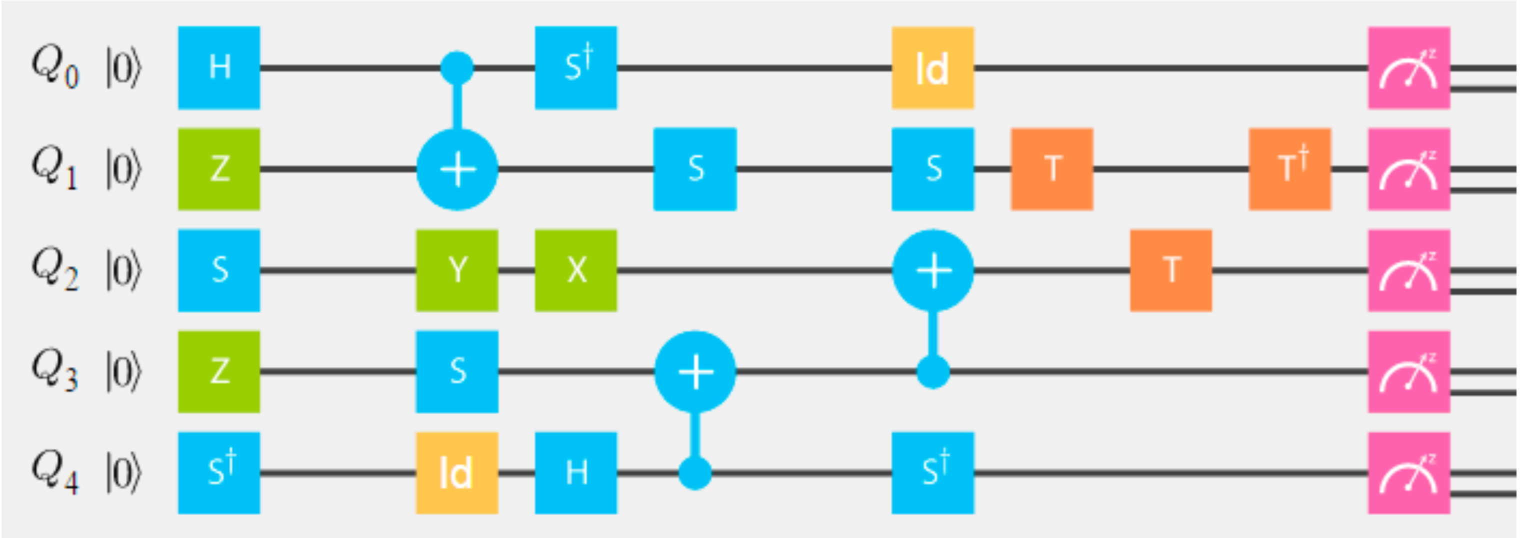
THEORY: **quantum lambda-calculus, functional quantum PL, q. while language semantics in various pictures, q. Hoare logic and verification, …**

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!

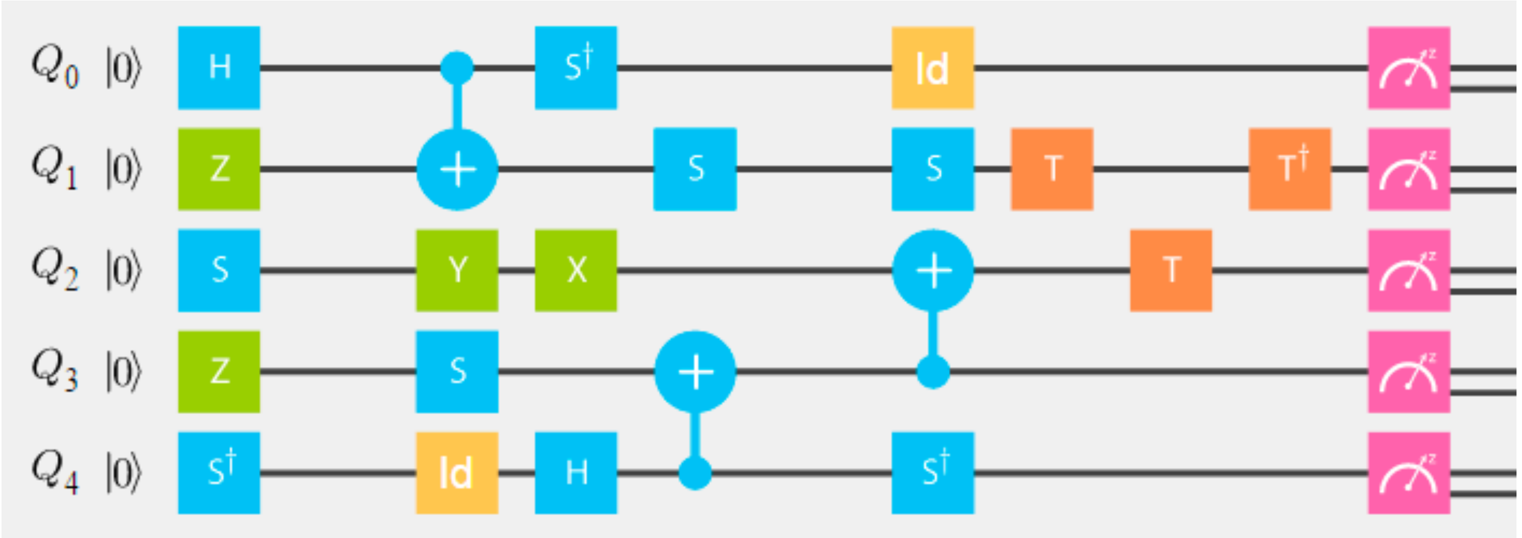
THEORY: **quantum lambda-calculus, functional quantum PL, q. while language semantics in various pictures, q. Hoare logic and verification, …**

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

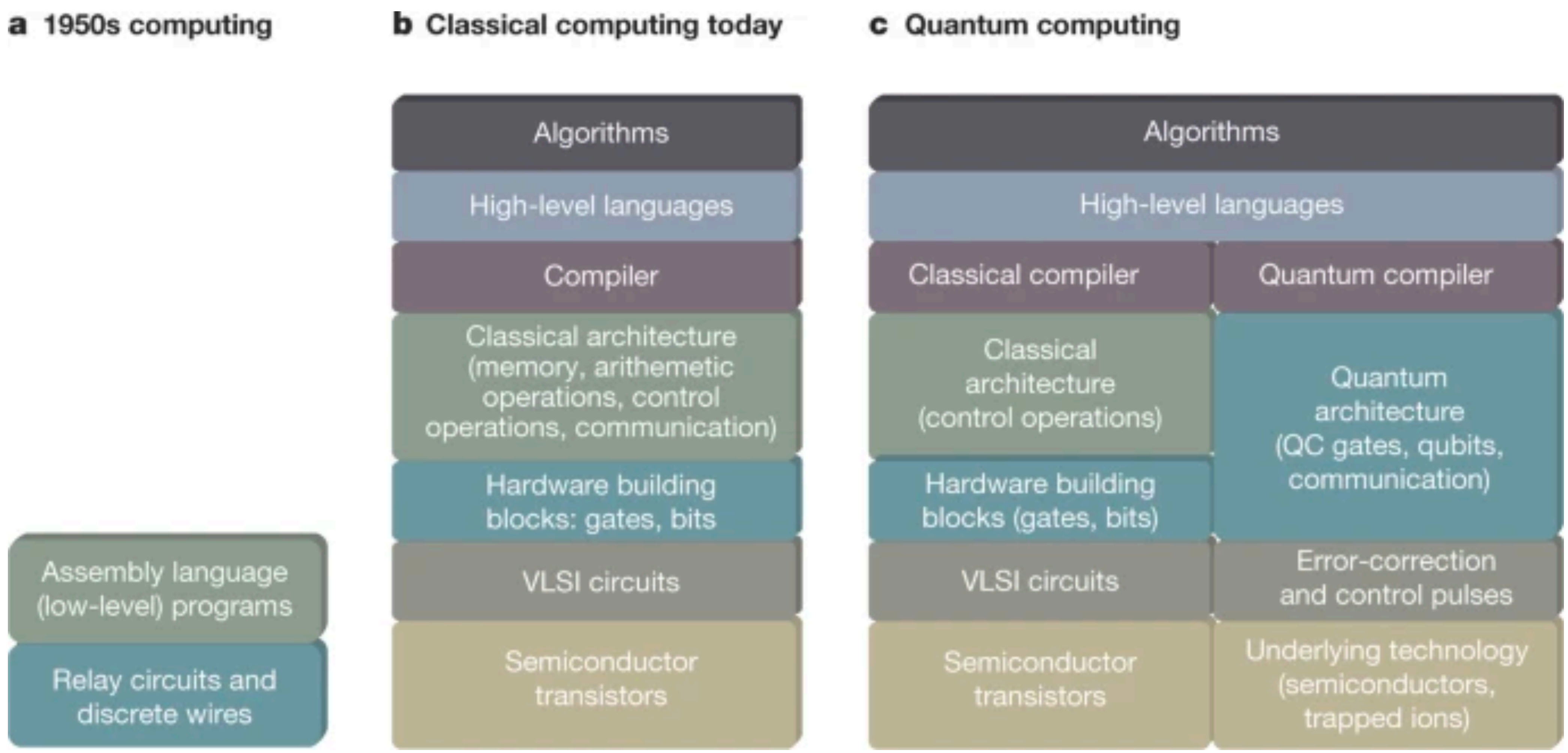
(2) lack of scalable verification: very hard to write **correct** programs



Verifying the circuit by observation …. not scalable …

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

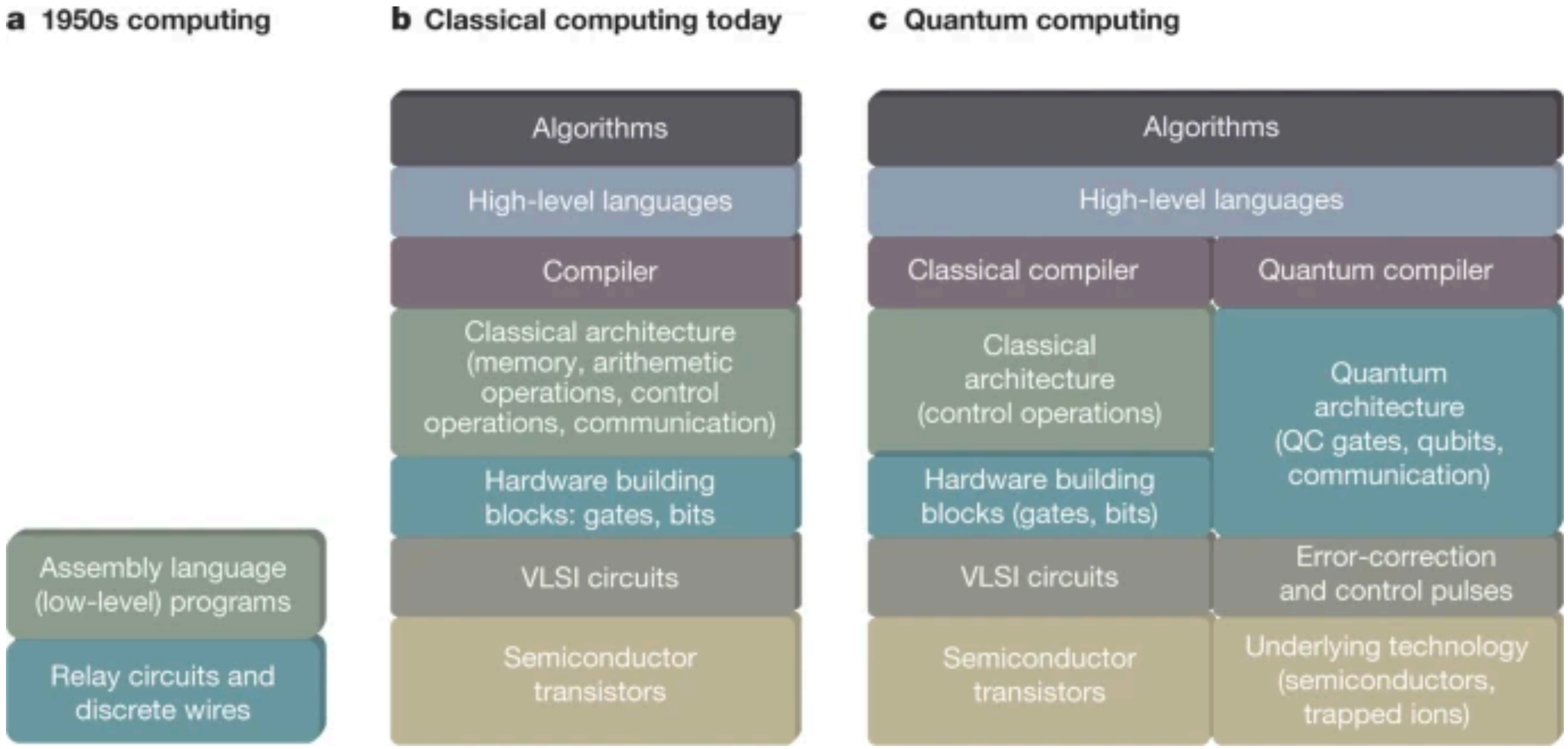
How to Develop Software for Q. Computing, e.g., compiler, system?



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?



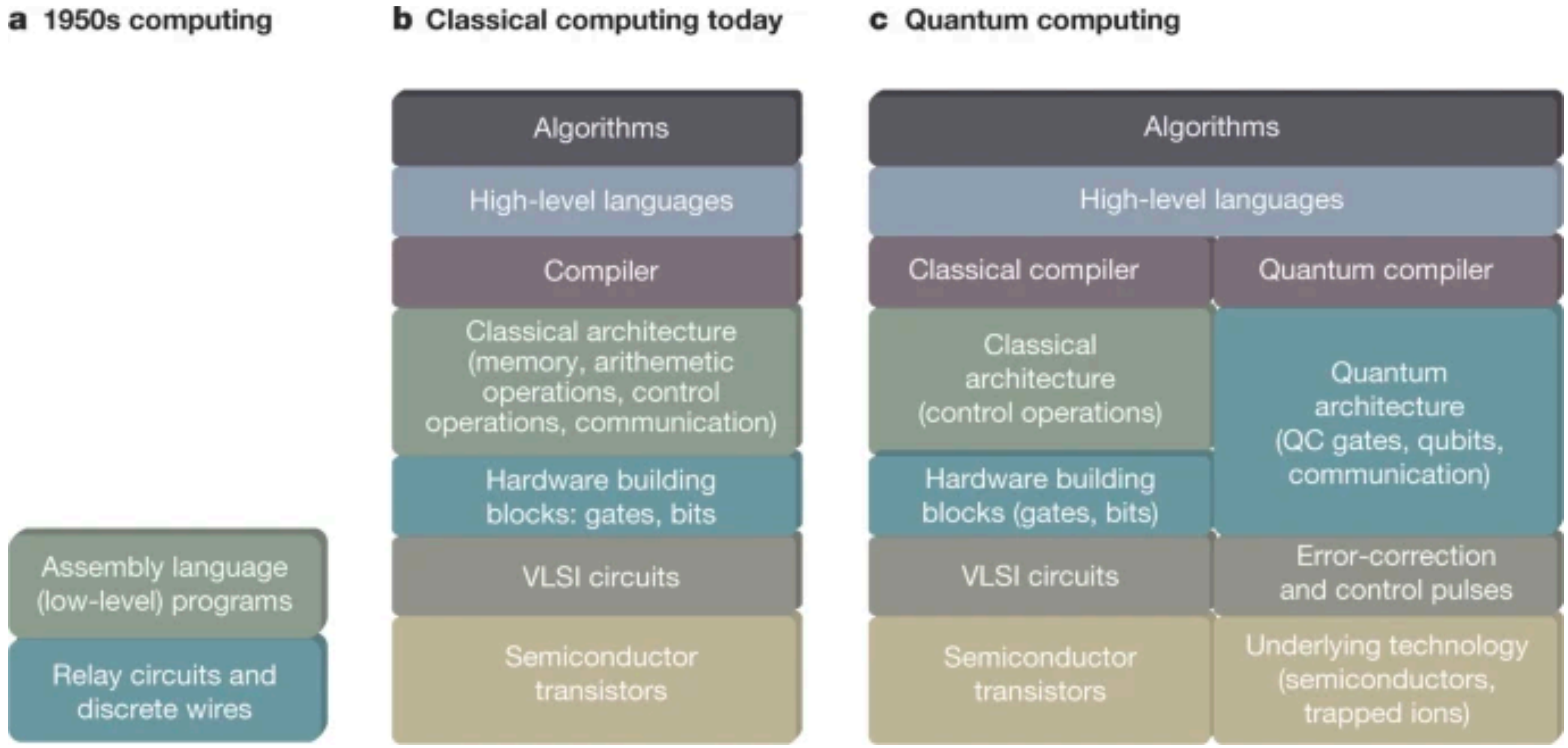
F. Chong, D. Franklin, M. Martonosi, Nature 549, 180

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

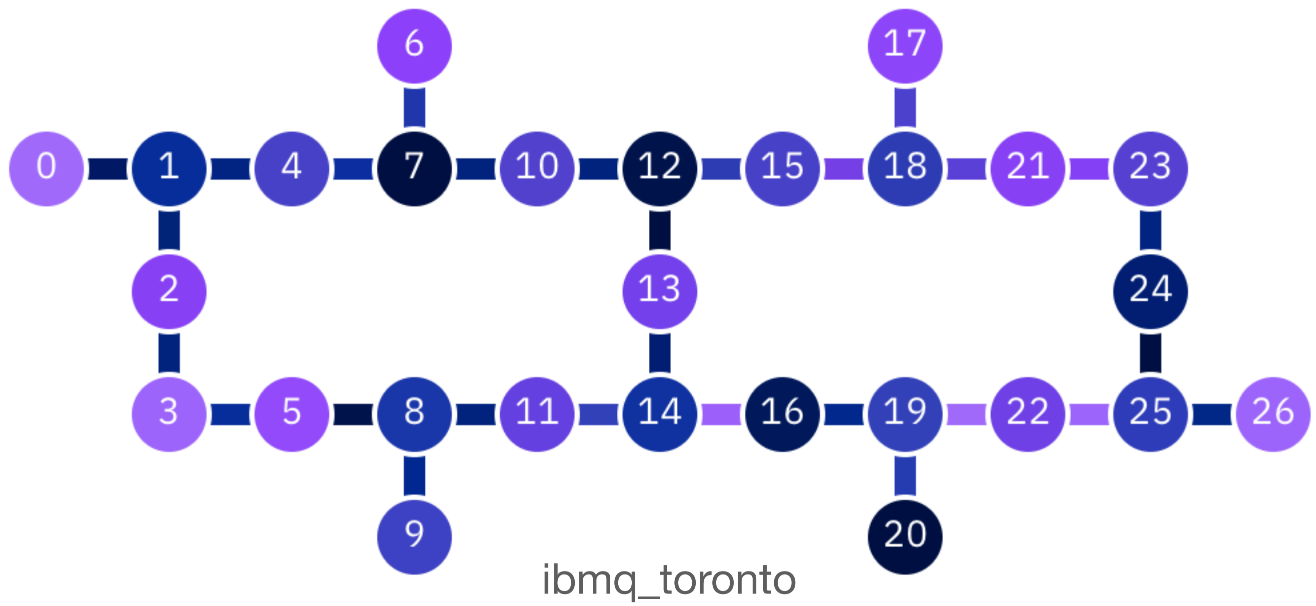
Large Design Space for System Software for Quantum Computers.

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)

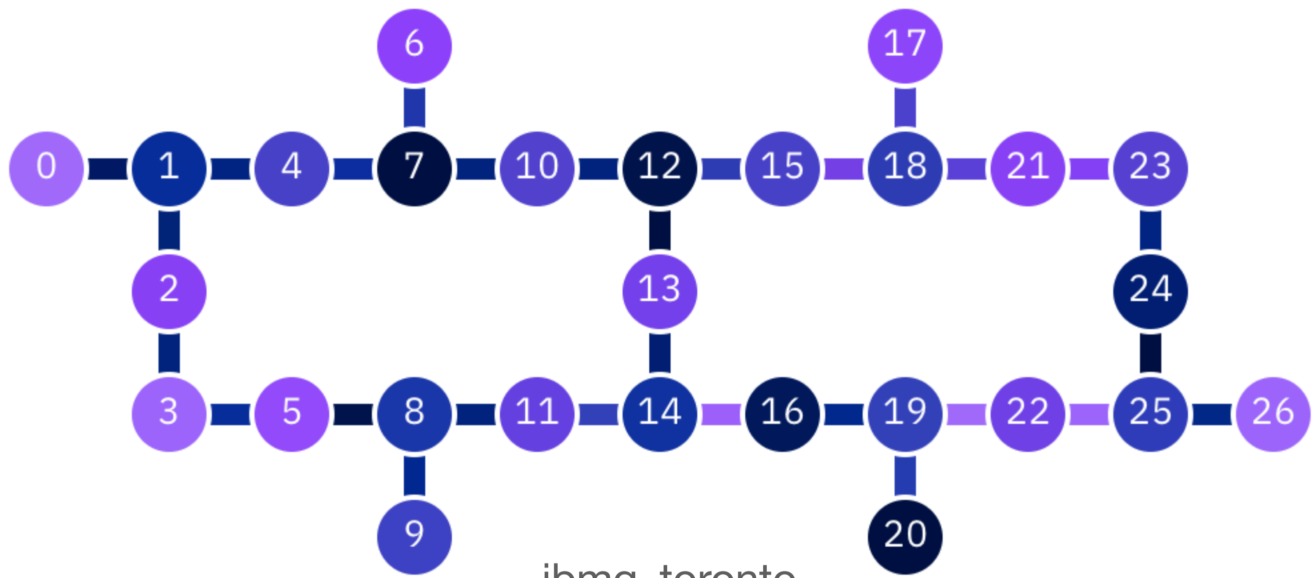
How to Design and Implement Architecture for Quantum Computing?



Mapping, Error Mitigation, …

approximate computing

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



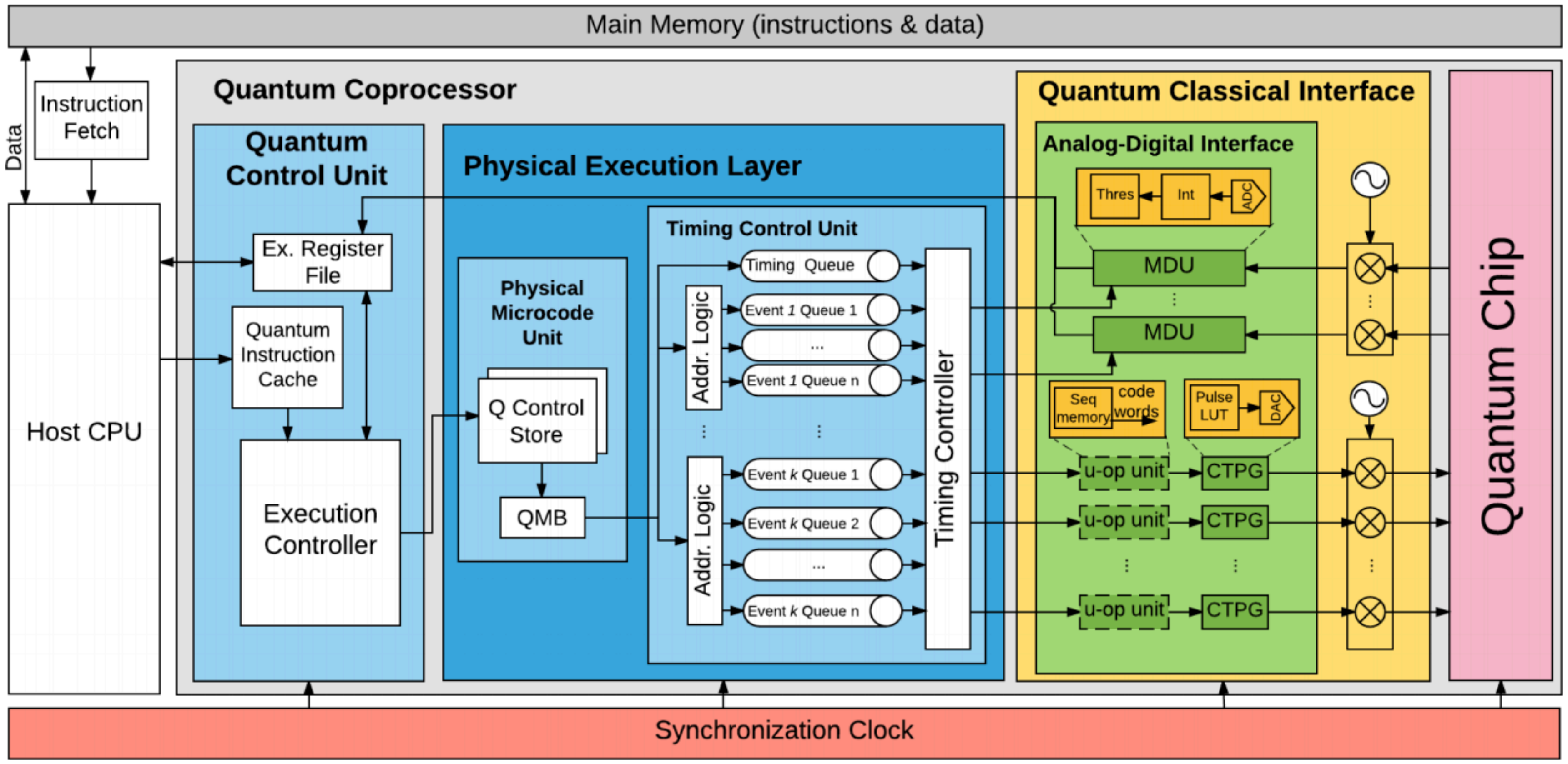
ibmq_toronto

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:

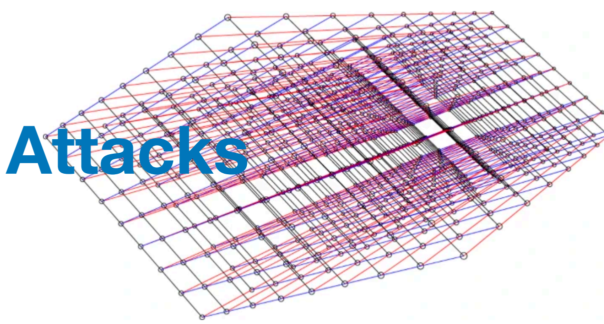
Relational Quantum Hoare Logic (Unruh; Barthe et al.)

Quantum Cryptanalysis:

Resource estimation of Complex Quantum Attack Programs

Post-Quantum Cryptography:

Classical Cryptographic Systems Resilient to Quantum Attacks

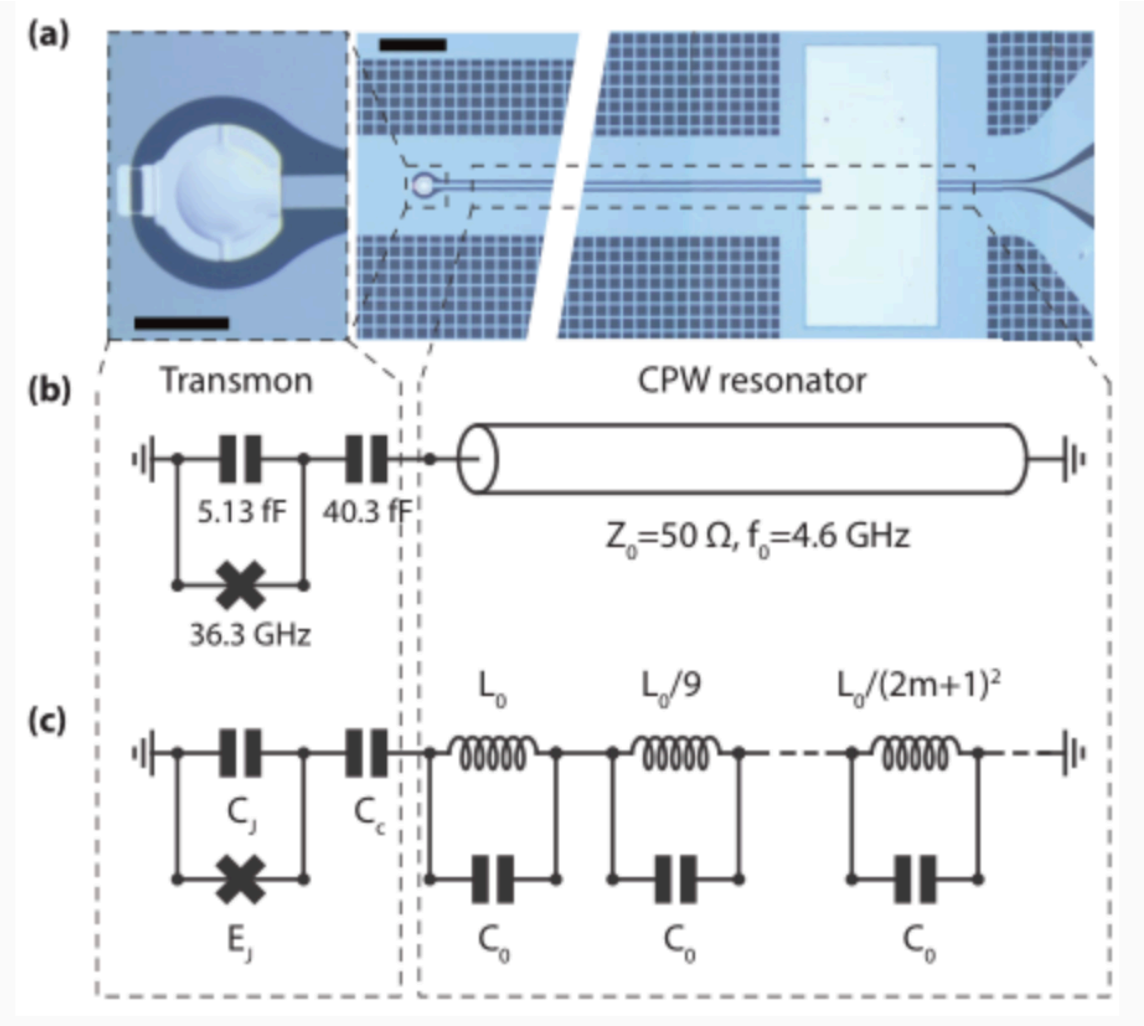


For Classical Cryptographic Systems

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

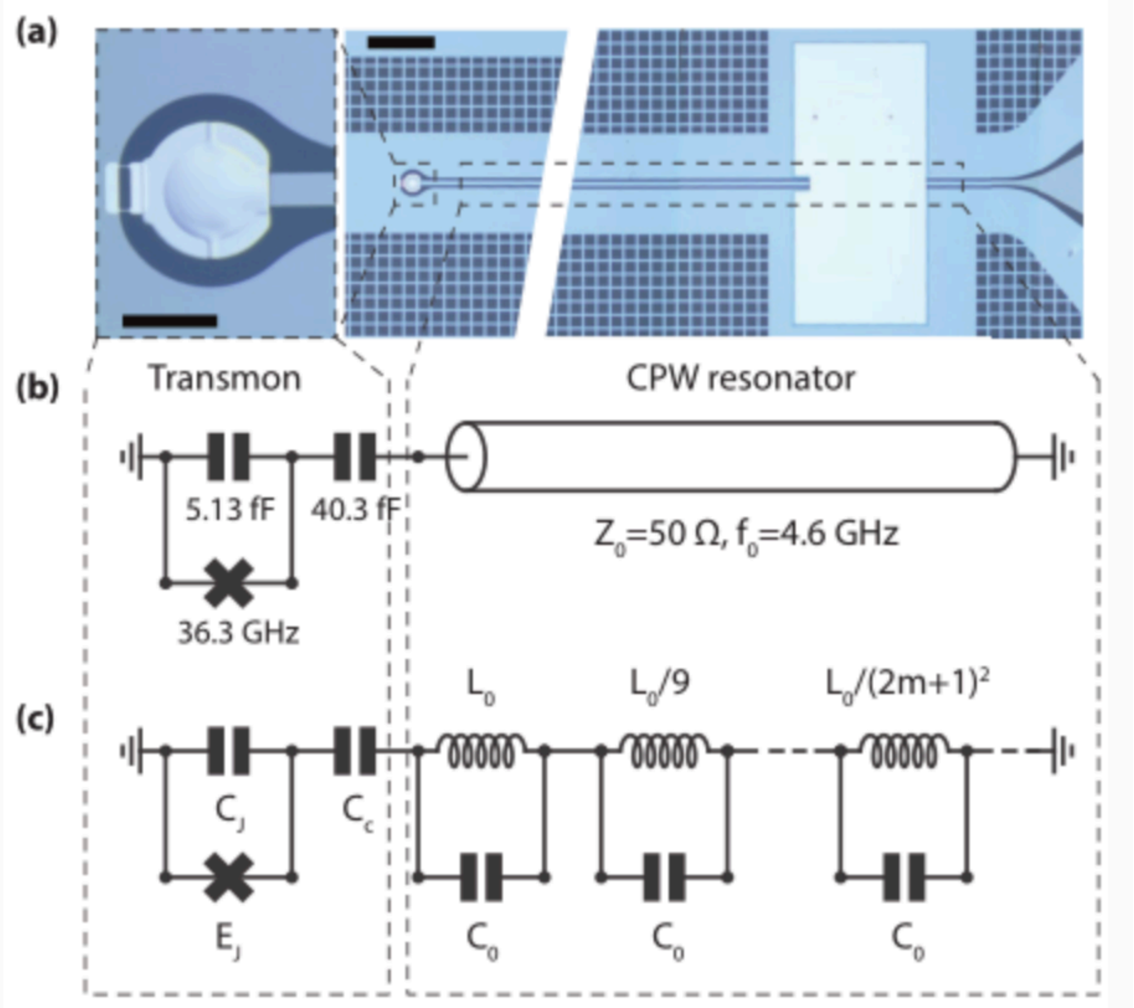
Formally generated security analysis will provide not only efficient and high assurance proofs that can replace the tedious and error-prone analysis for experts, but also independently verifiable proofs that can be used by security practitioners without much quantum knowledge.

How to Scale and Automate the Design of Quantum Hardware ?

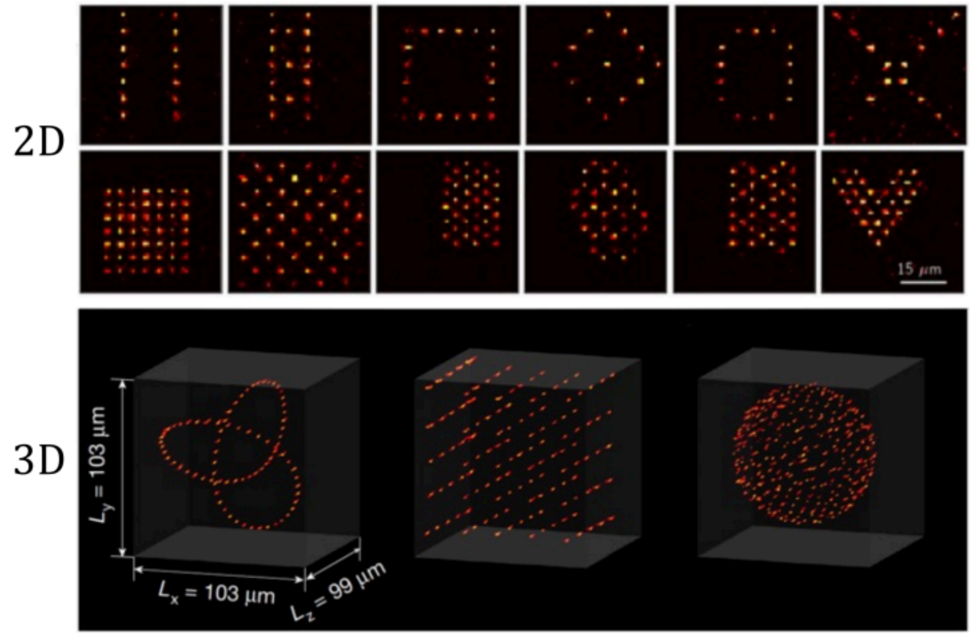


Superconducting Credit: arXiv:1704.06208

How to Scale and Automate the Design of Quantum Hardware ?

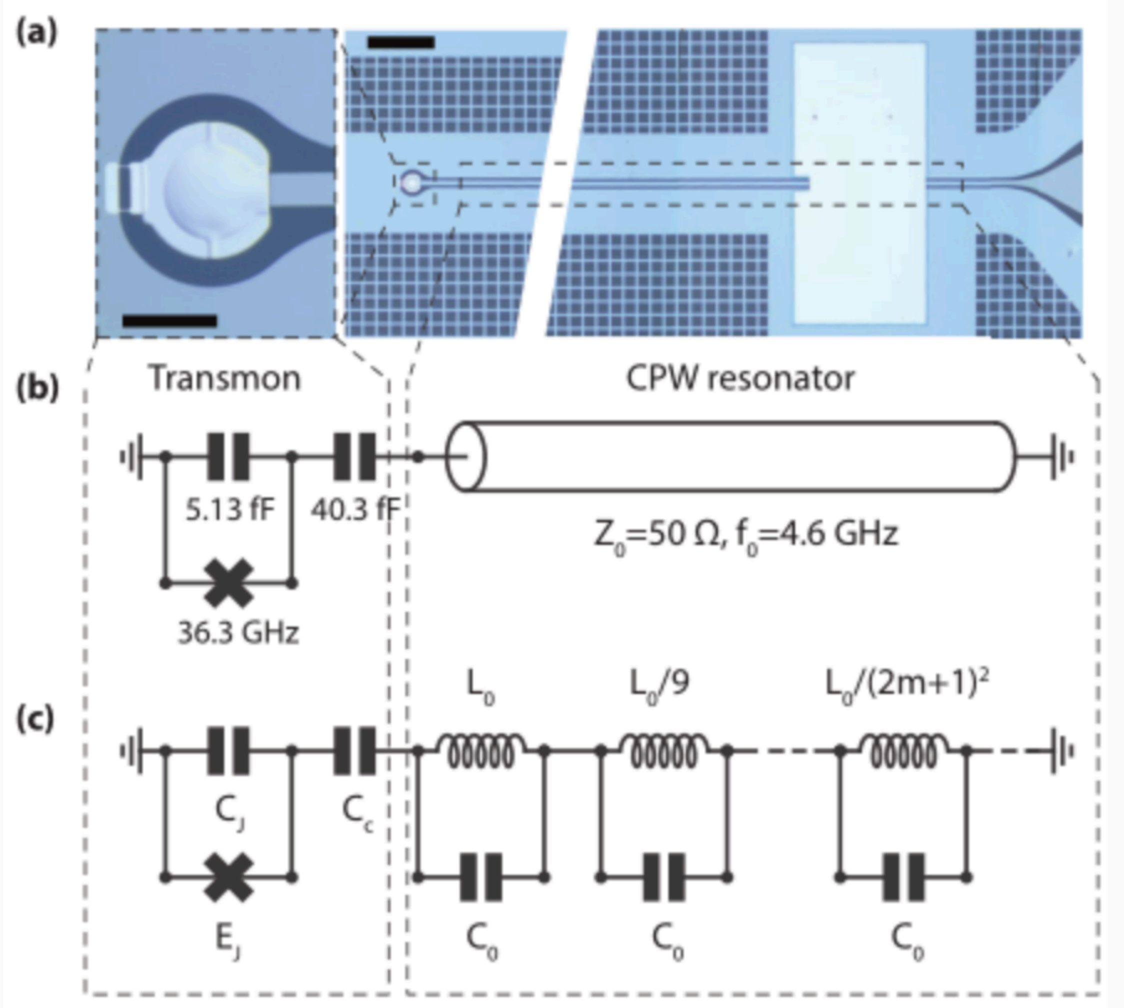


Superconducting Credit: arXiv:1704.06208

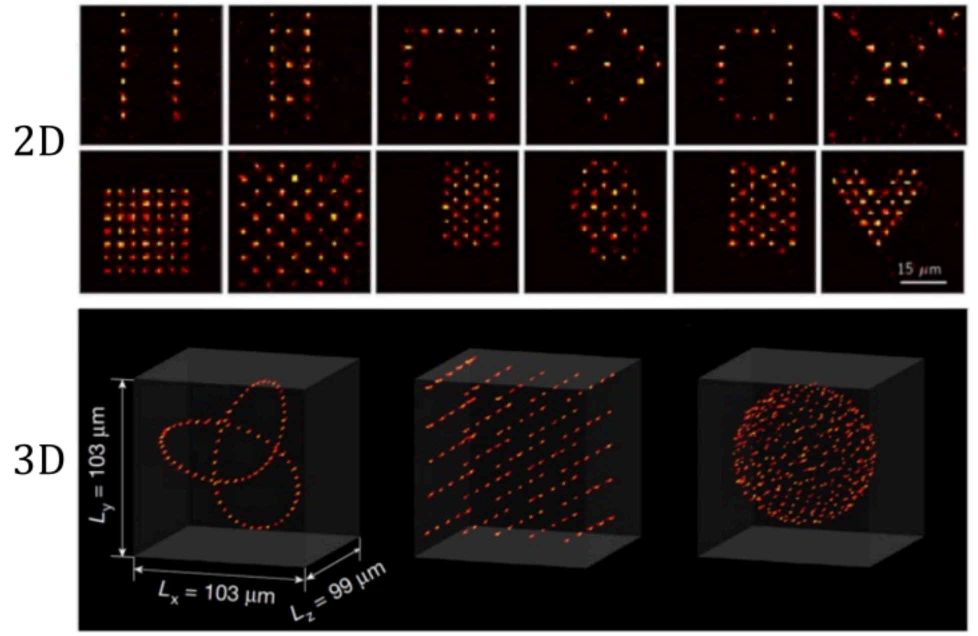


Neutral Atoms Credit: arXiv:2006.12326

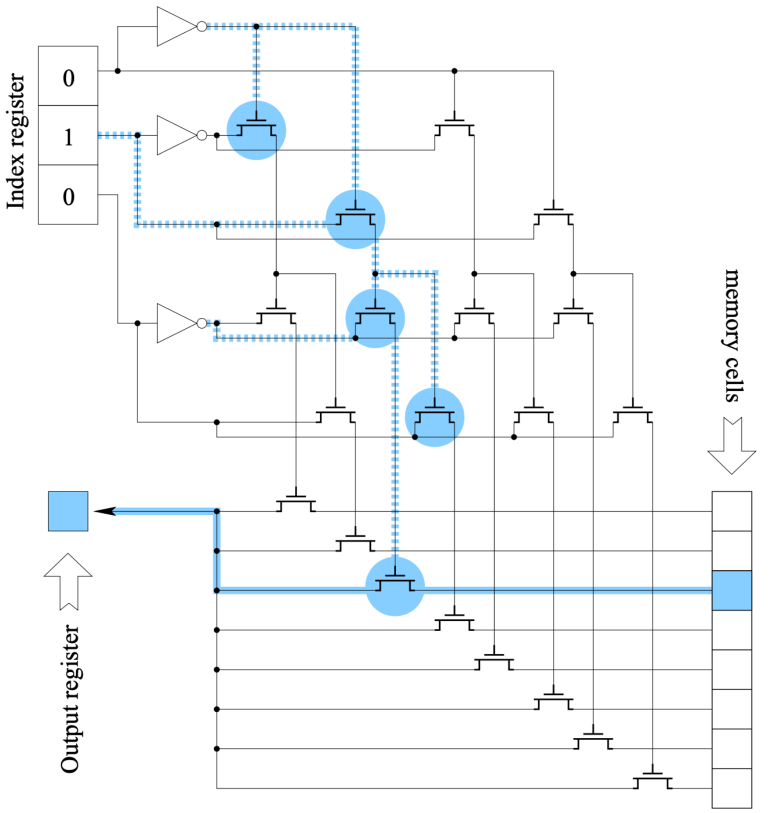
How to Scale and Automate the Design of Quantum Hardware ?



Superconducting Credit: arXiv:1704.06208

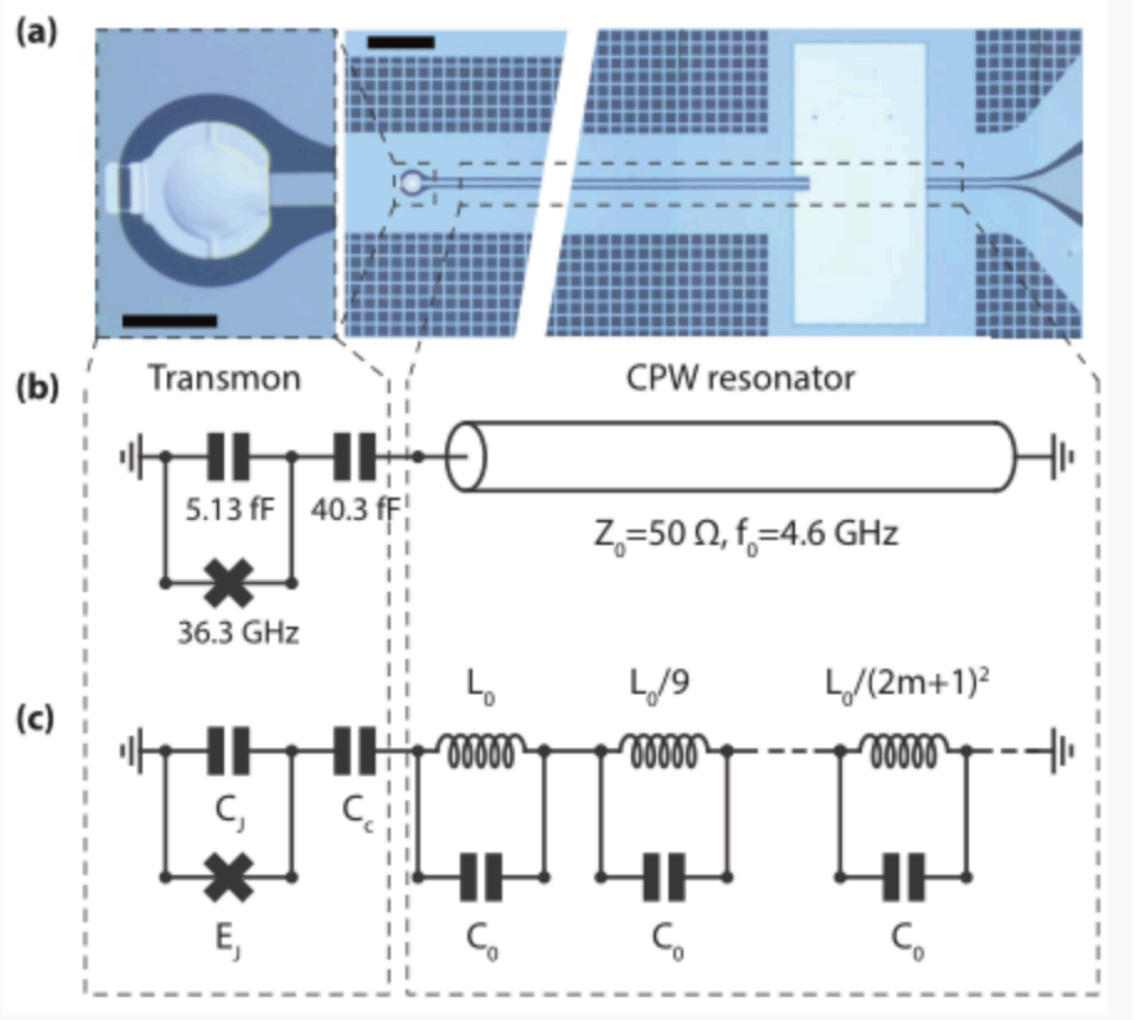


Neutral Atoms Credit: arXiv:2006.12326

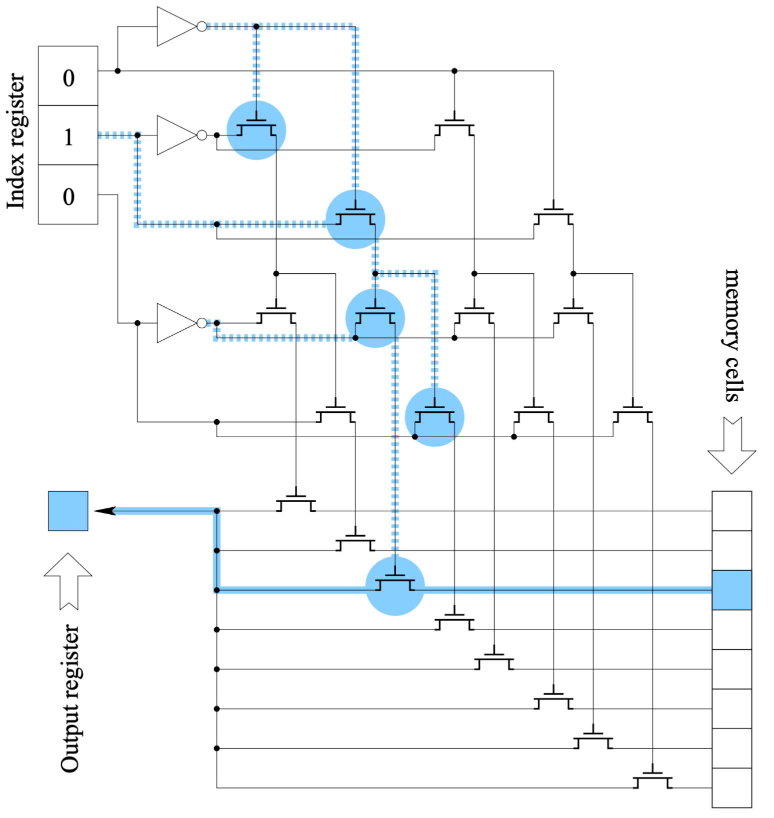


QRAM Architecture
Credit: ArXiv 0807.4994

How to Scale and Automate the Design of Quantum Hardware ?

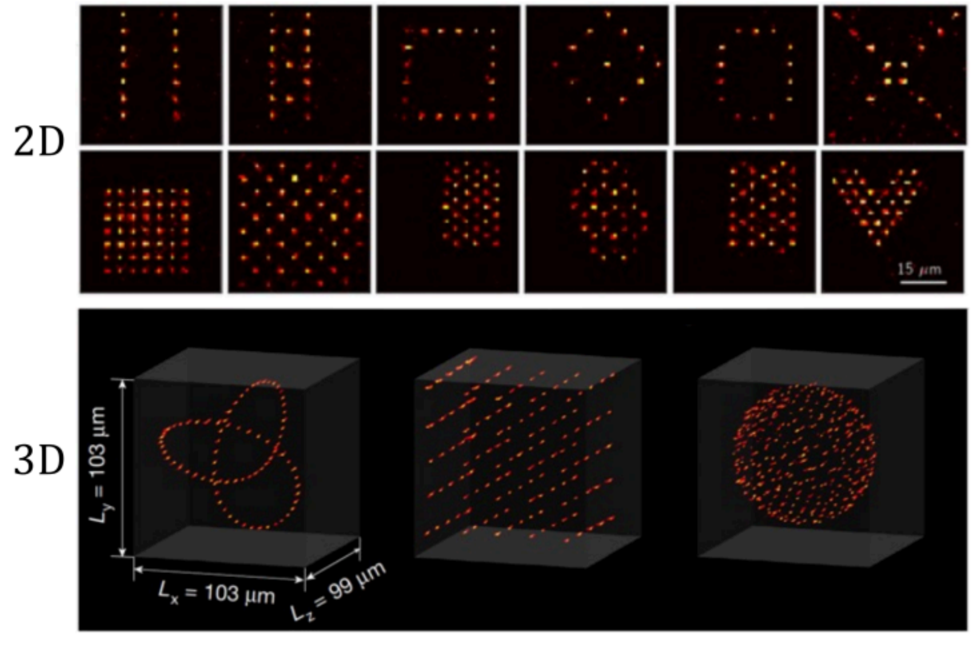


Superconducting Credit: arXiv:1704.06208



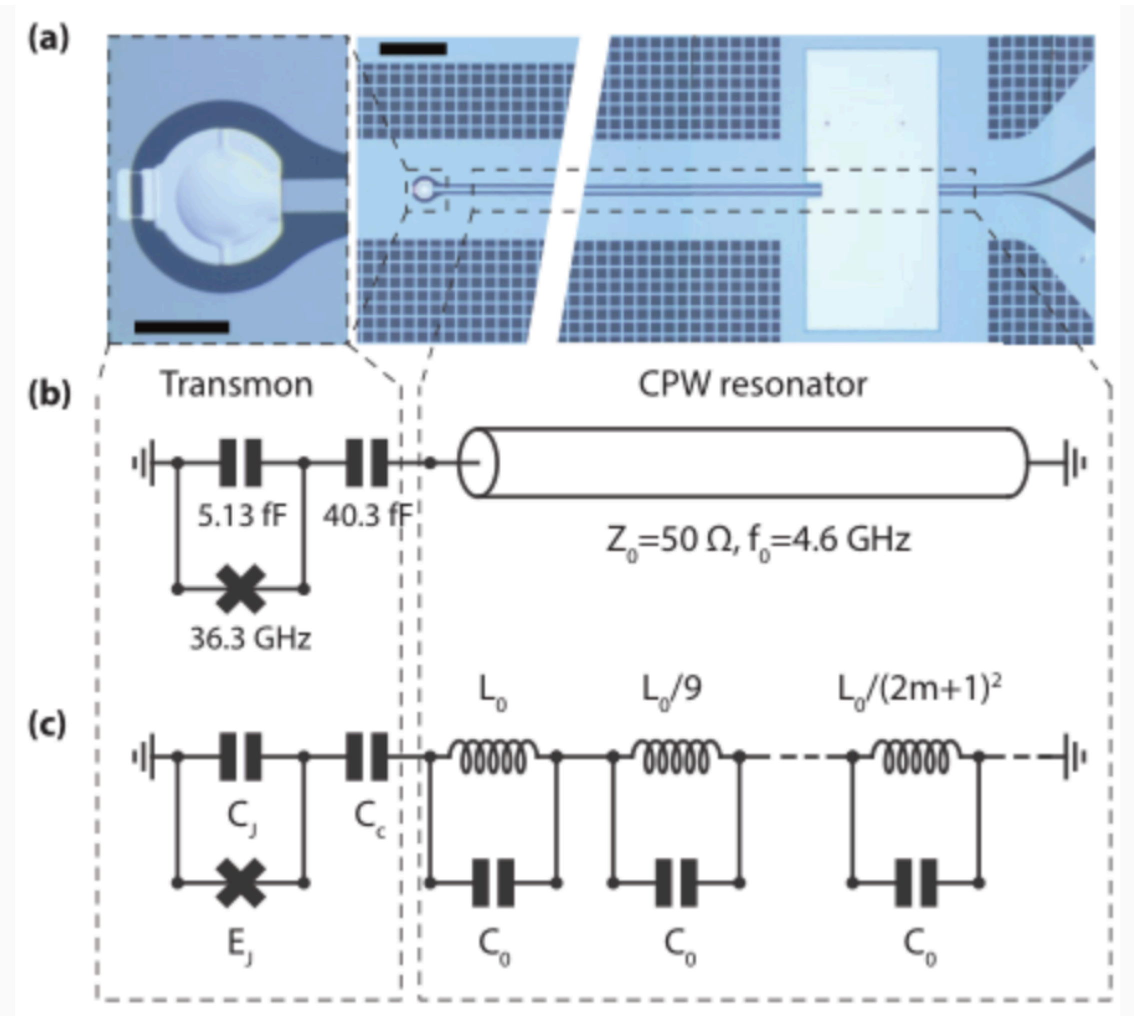
QRAM Architecture
Credit: ArXiv 0807.4994

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

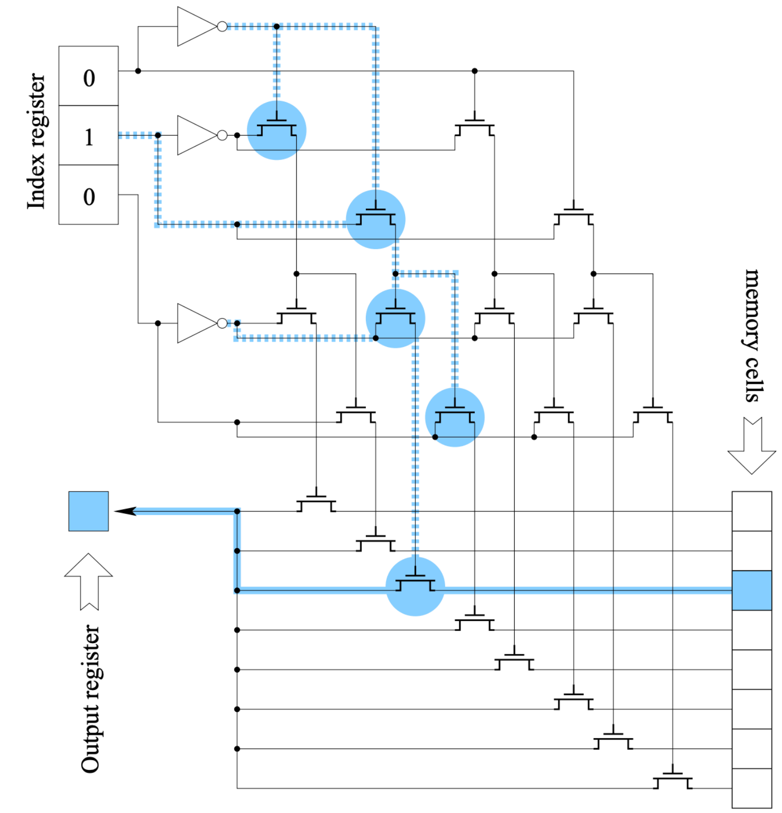


Neutral Atoms Credit: arXiv:2006.12326

How to Scale and Automate the Design of Quantum Hardware ?

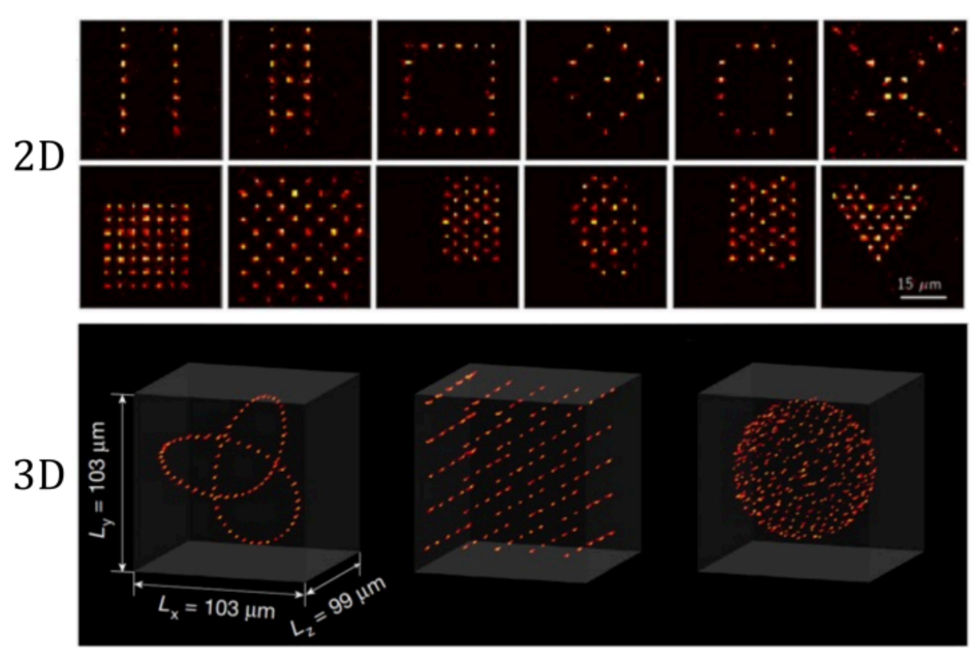


Superconducting Credit: arXiv:1704.06208



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

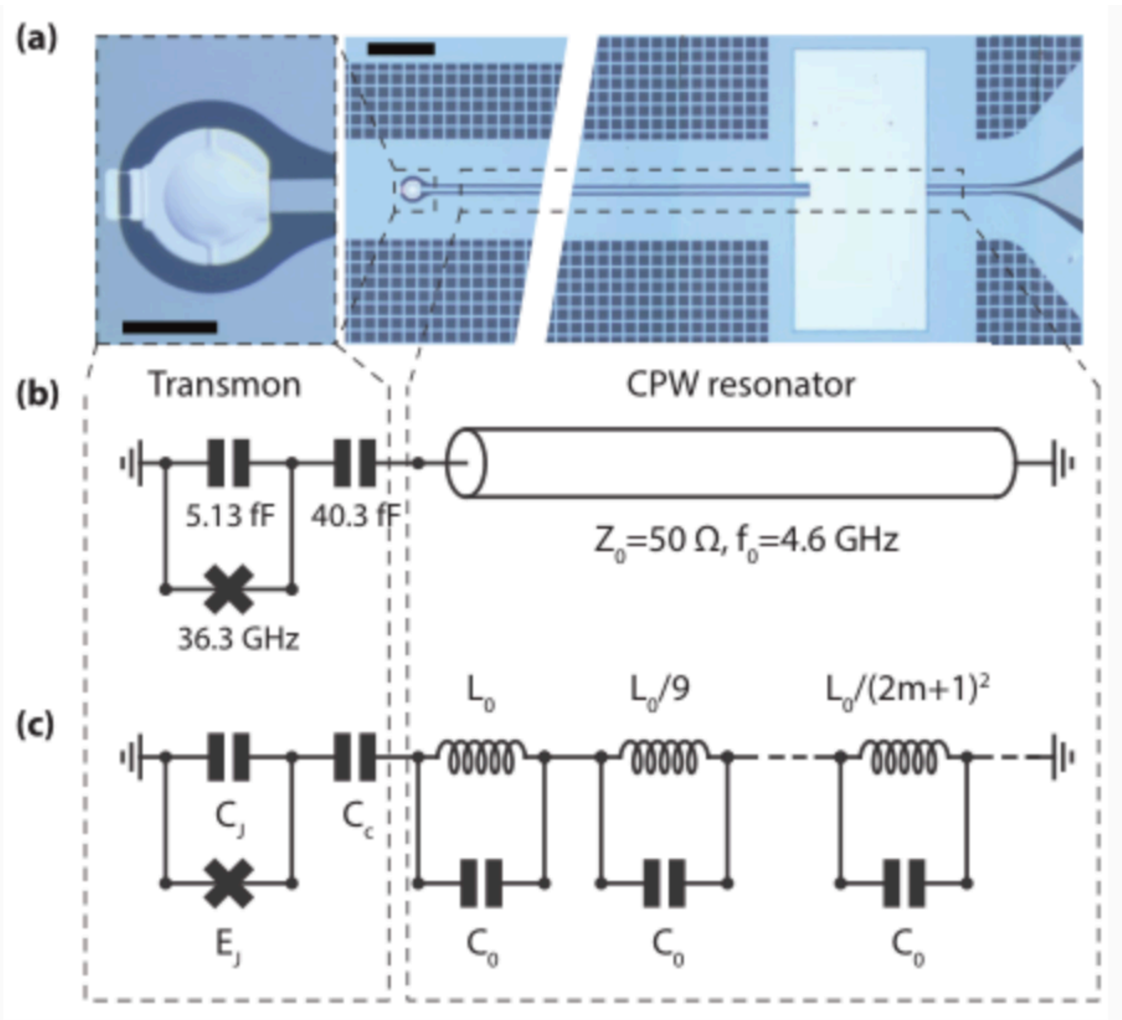
Pat Hanrahan
Stanford University, USA
hanrahan@cs.stanford.edu

SNAPL 2019

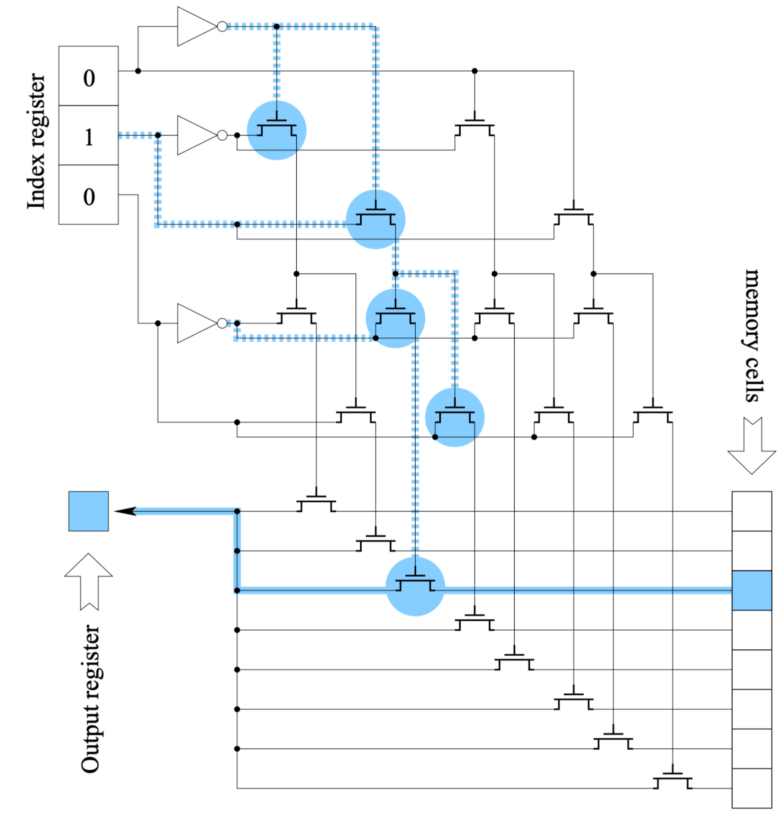
Verilog

HDL

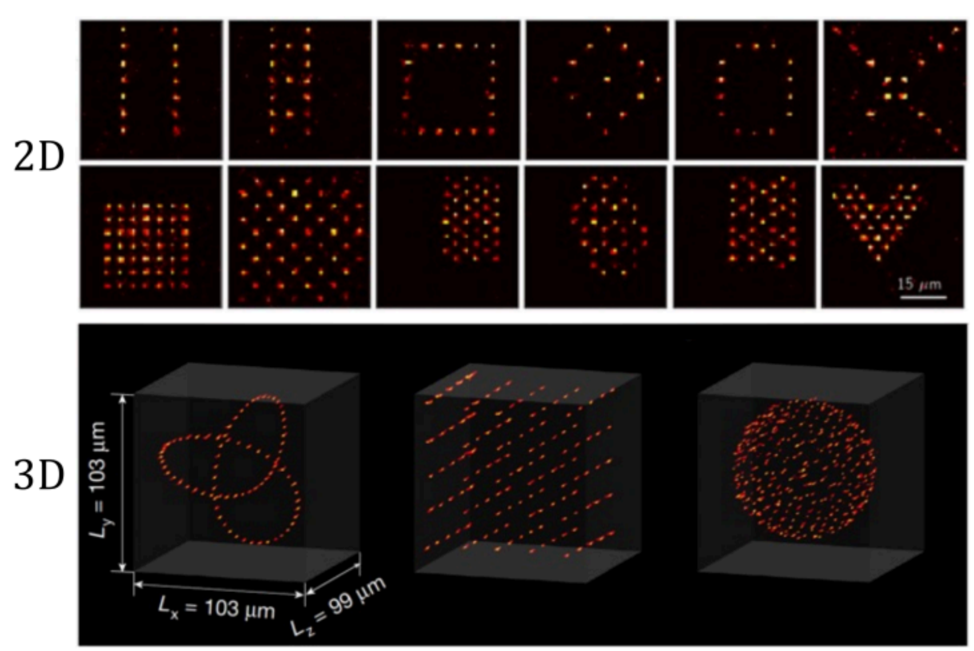
How to Scale and Automate the Design of Quantum Hardware ?



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

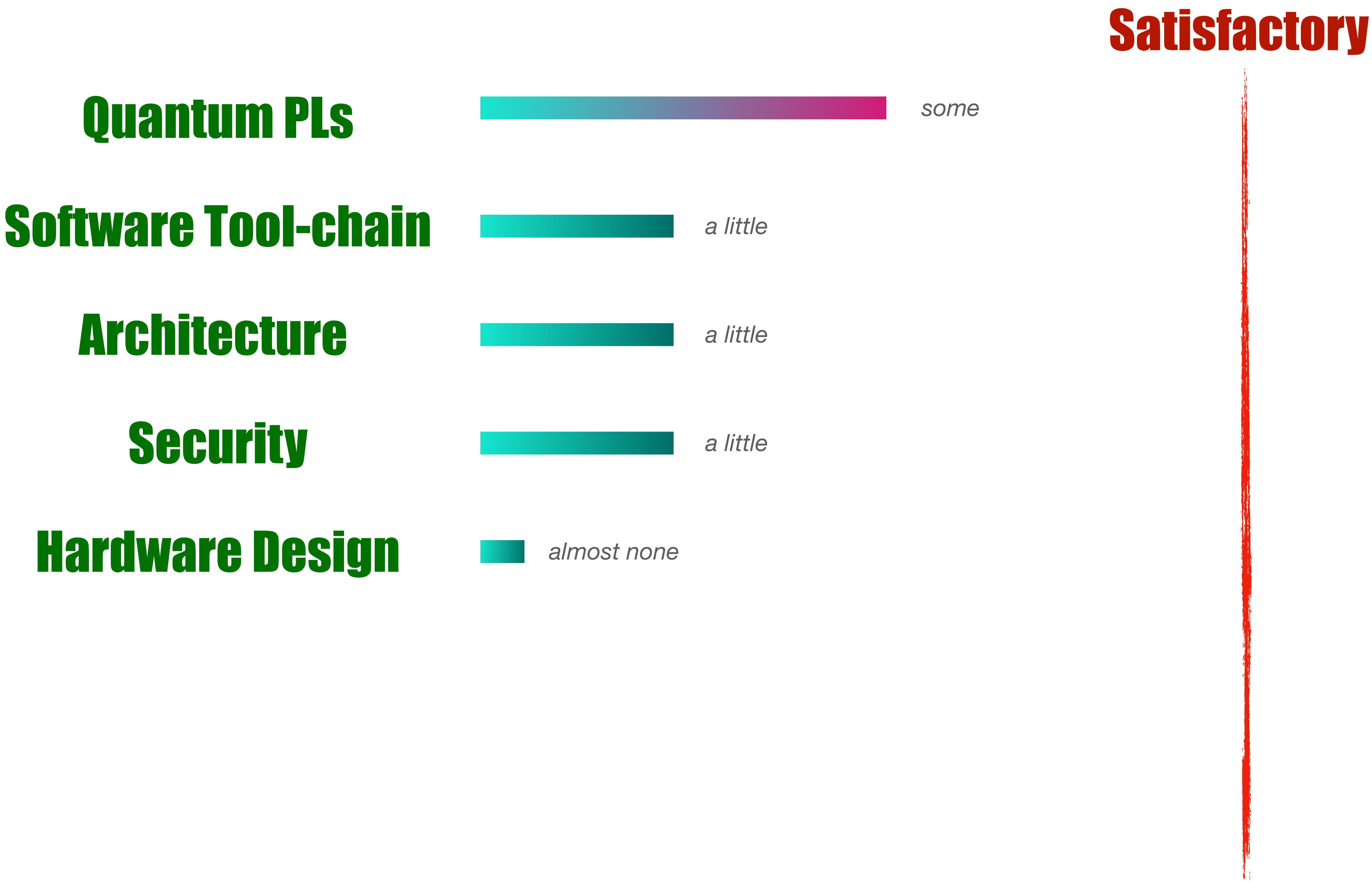
Verilog
HDL

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

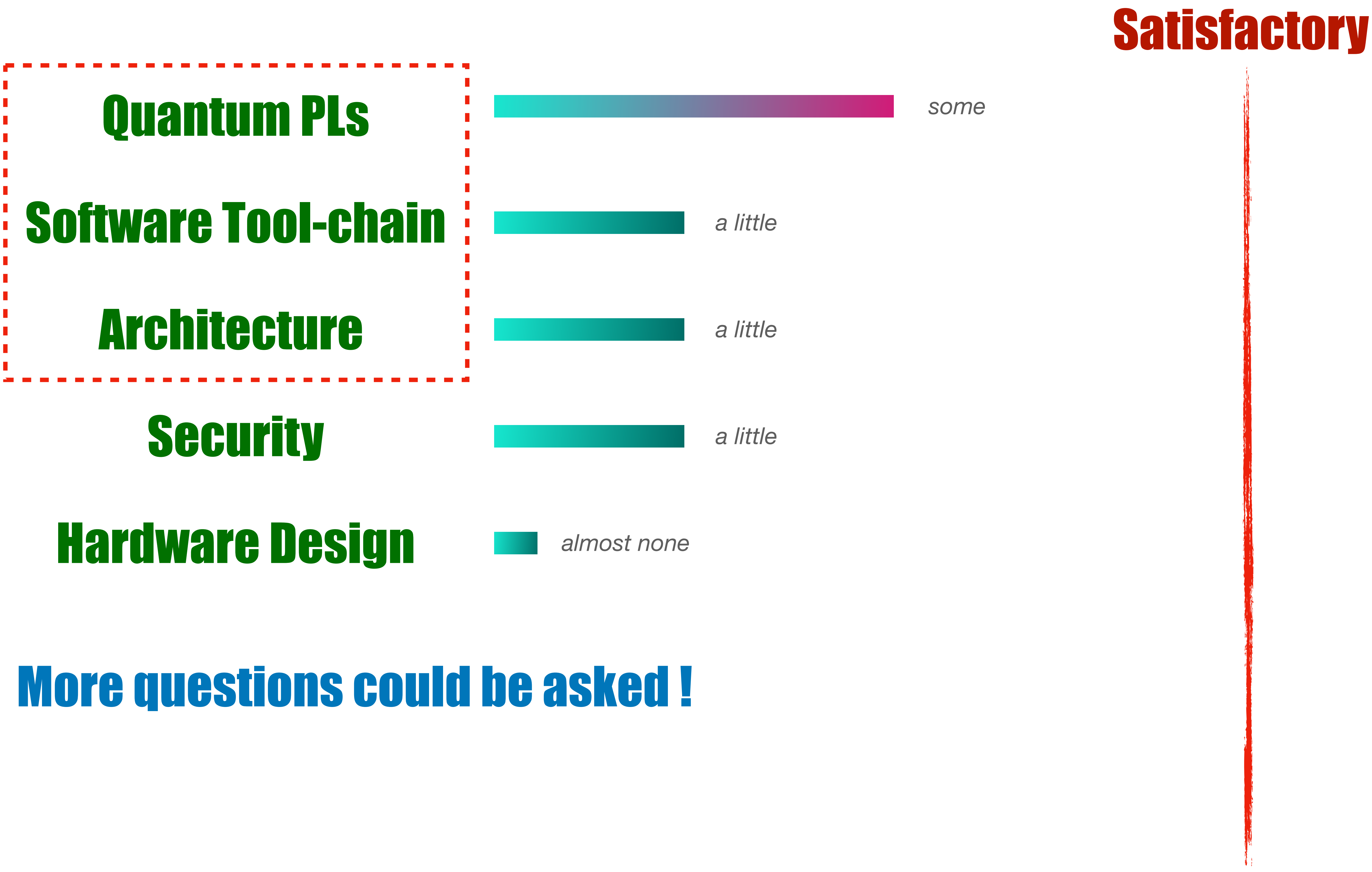
Summary

Satisfactory

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

More questions could be asked !

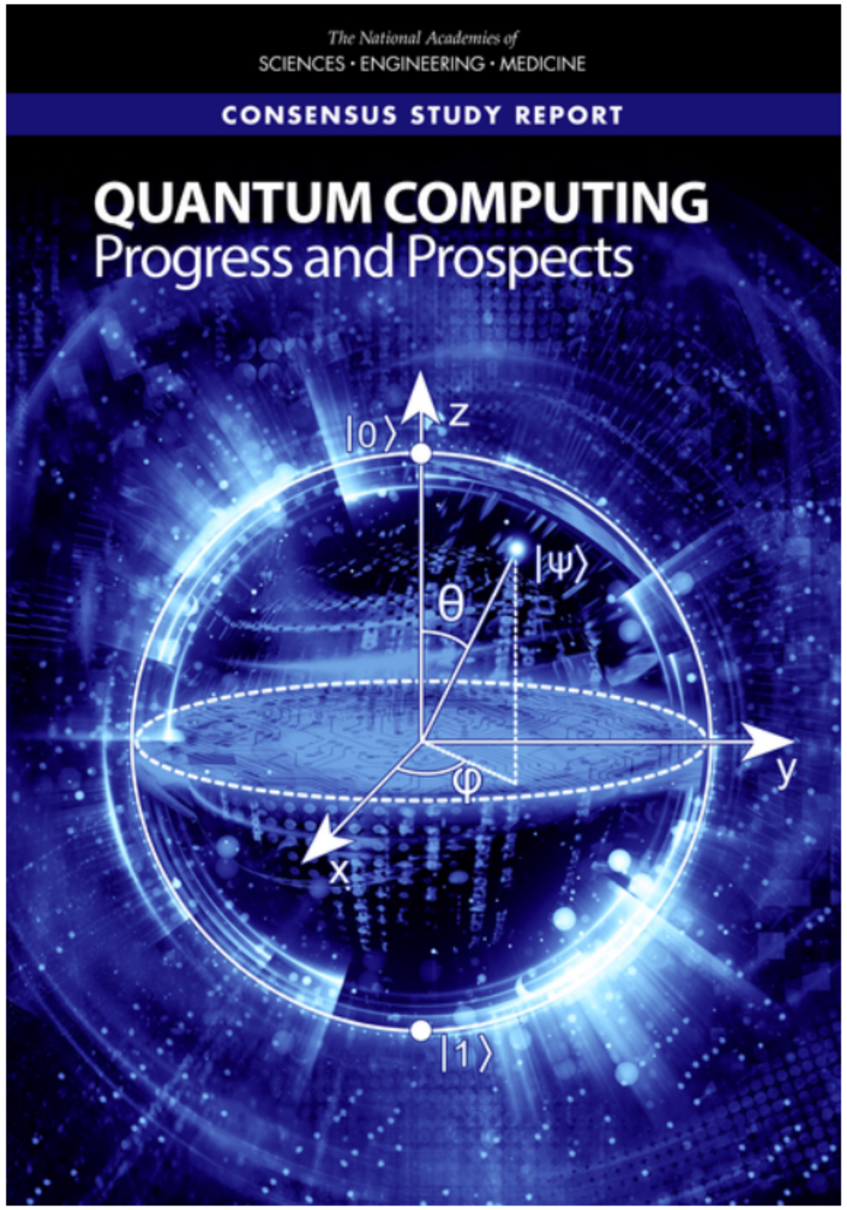
Summary

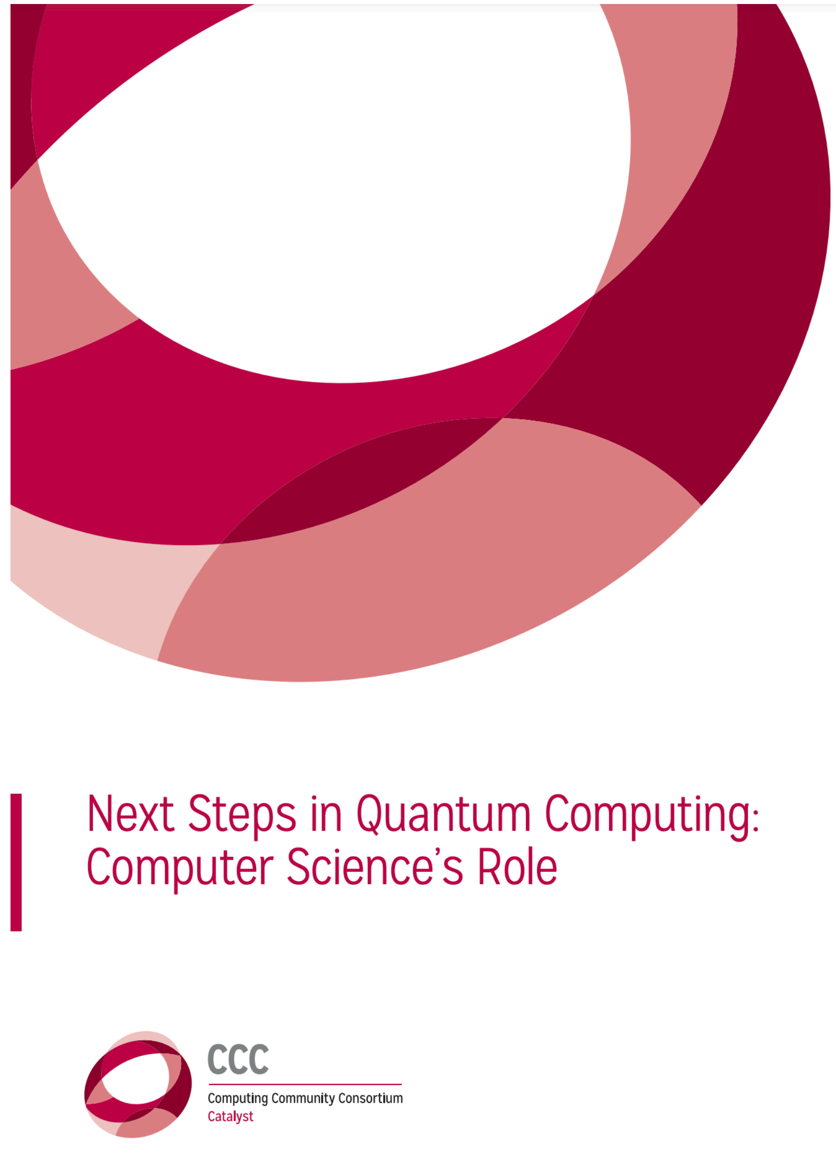


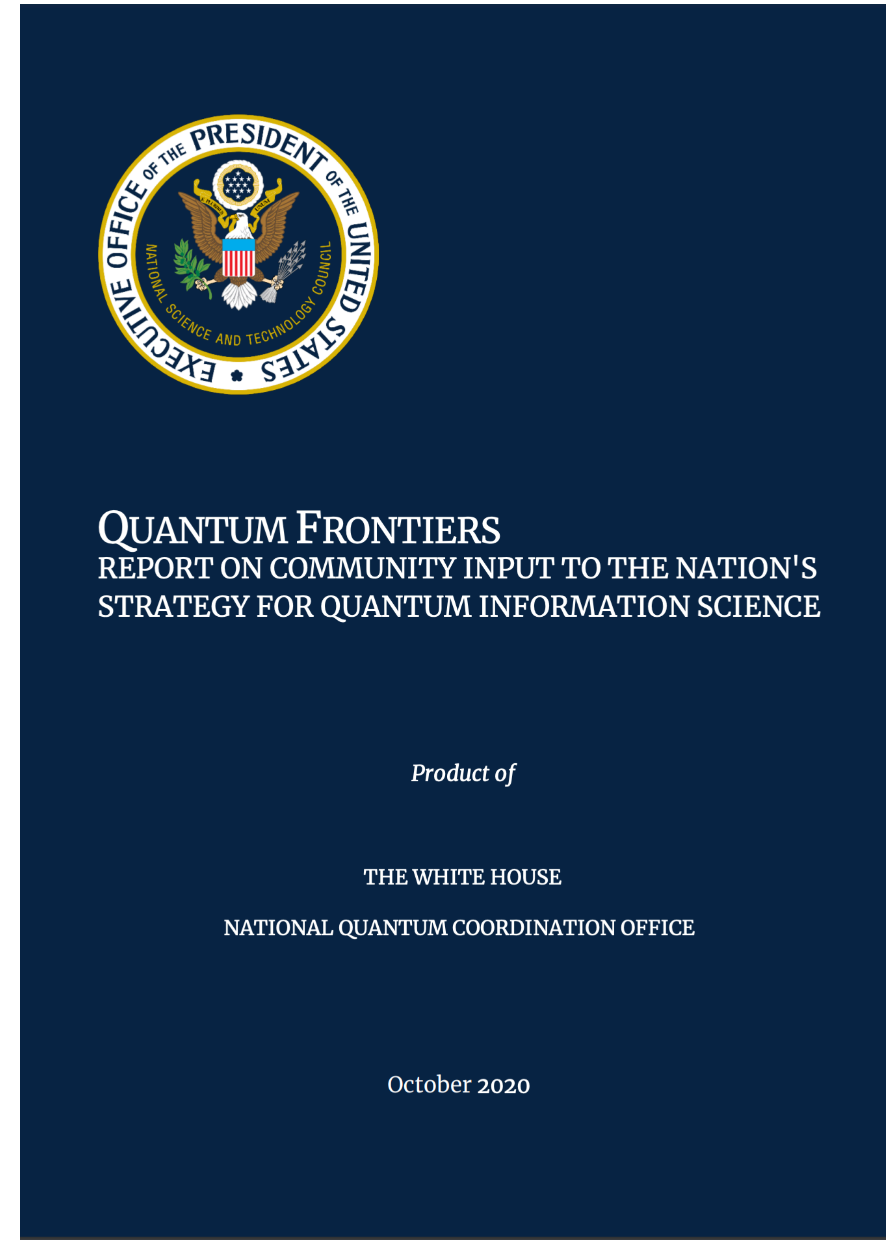
More questions could be asked !

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

Further Readings: Thank You! Q & A







Reference: links are available at https://www.cs.umd.edu/~xwu/mini_lib.html

Outline

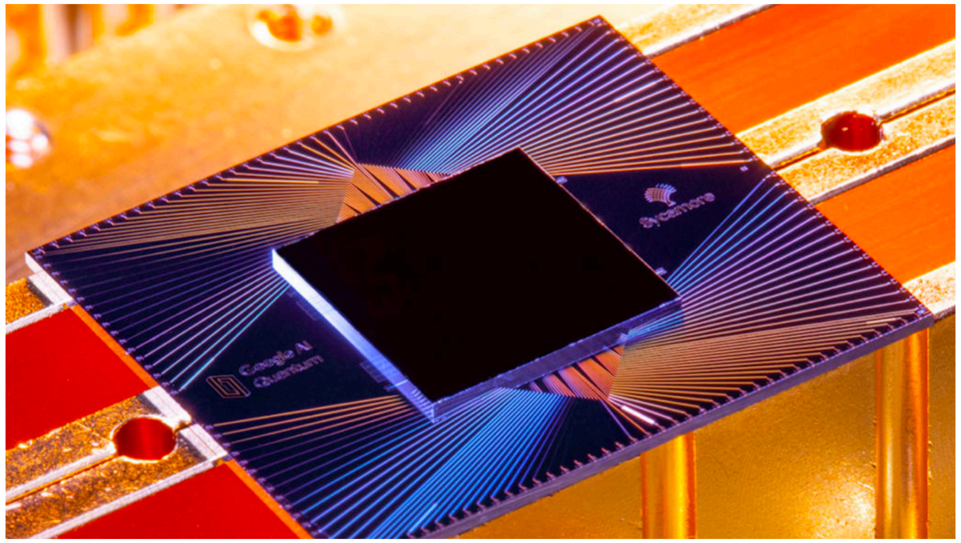
(1) Introduction to Quantum Computing and Potential Roles of Programming Languages (25 min + 5 Q & A)

(2) A Mini-Course of Quantum Hoare Logic on Quantum While Language (30 min + 5 Q & A)

(3) Discussion on existing and potential Programming Language research opportunities (20 min + 5 Q & A)

Reference: tutorial slides and some references are available at https://www.cs.umd.edu/~xwu/mini_lib.html

What is Quantum Computing?



A Quantum Computer

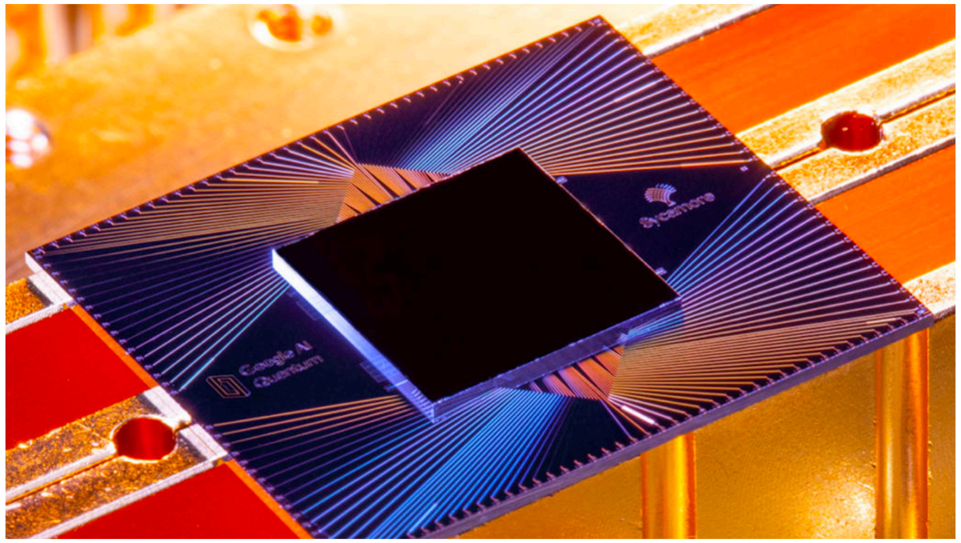
An **Operation** O —> A **Quantum** **Physical Evolution** Q

Computation:

Evolution of the Machine: $Q_1, Q_2, Q_3, \cdots$

The accumulative evolution carries some computation!

What is Quantum Computing?



A Quantum Computer

An **Operation** O —> A **Quantum** **Physical Evolution** Q

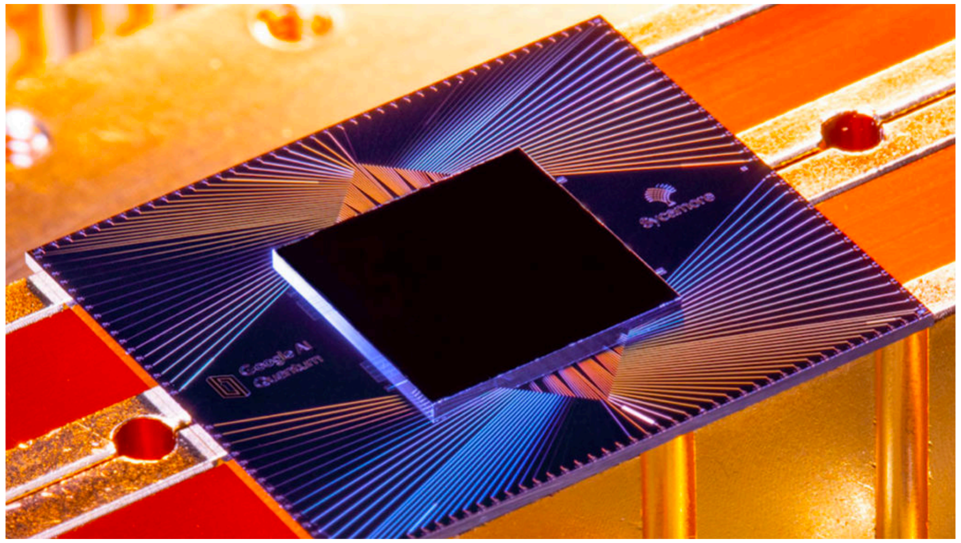
Computation:

Evolution of the Machine: $Q_1, Q_2, Q_3, \cdots$

The accumulative evolution carries some computation!

Consider quantum machines of finite-dimension. Hilbert space -> Euclidean space

What is Quantum Computing?



A Quantum Computer

An **Operation** O —> A **Quantum** **Physical Evolution** Q

Computation:

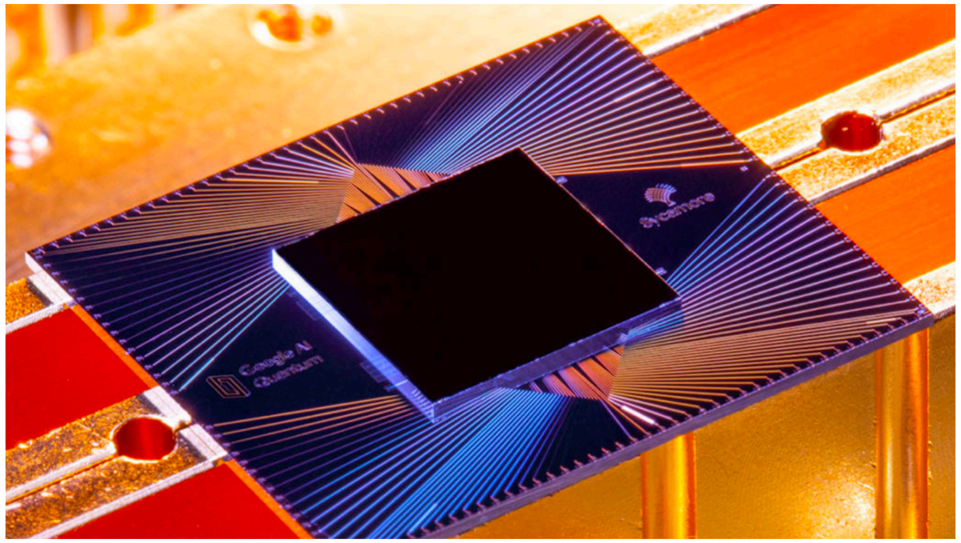
Evolution of the Machine: $Q_1, Q_2, Q_3, \cdots$

The accumulative evolution carries some computation!

Consider quantum machines of finite-dimension. Hilbert space -> Euclidean space

The Math Model of Quantum Machines comes from the math model of Q_is.
(semantics)

What is Quantum Computing?



A Quantum Computer

An **Operation** O —> A **Quantum Physical Evolution** Q

Computation:

Evolution of the Machine: $Q_1, Q_2, Q_3, \cdots$

The accumulative evolution carries some computation!

Consider quantum machines of finite-dimension. Hilbert space -> Euclidean space

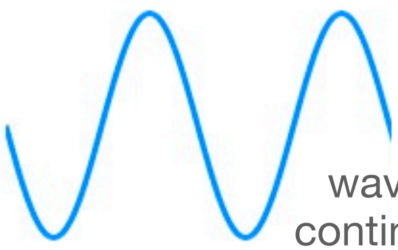
The Math Model of Quantum Machines comes from the math model of Q_is.
(semantics)

Four Postulates for Quantum Mechanics:

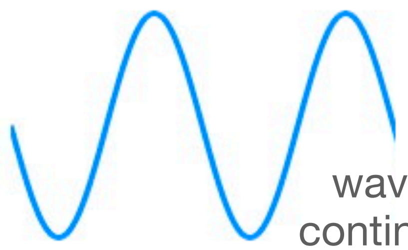
- **State Space postulate**
- **Evolution postulate — No-Cloning theorem**
- **Composite System postulate**
- **Measurement postulate**

State Space postulate: (pure) quantum state represented by unit complex vectors

State Space postulate: (pure) quantum state represented by unit complex vectors

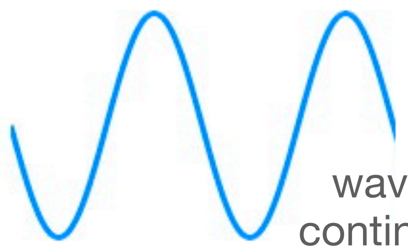


State Space postulate: (pure) quantum state represented by unit complex vectors


wave in
continuous
domain

A *quantum bit* (**qubit**) refers to a quantum system of dimension 2

State Space postulate: (pure) quantum state represented by unit complex vectors

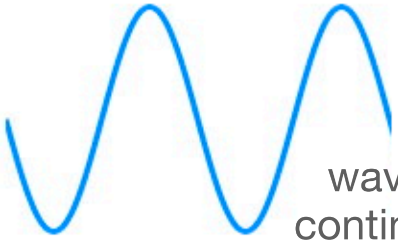


A *quantum bit* (**qubit**) refers to a quantum system of dimension 2

classical 0 and 1: $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

classical bits are special cases of quantum.

State Space postulate: (pure) quantum state represented by unit complex vectors



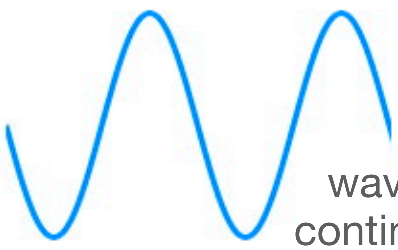
A *quantum bit* (**qubit**) refers to a quantum system of dimension 2

classical 0 and 1: $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

classical bits are special cases of quantum.

Dirac Notation

State Space postulate: (pure) quantum state represented by unit complex vectors



A *quantum bit* (**qubit**) refers to a quantum system of dimension 2

classical 0 and 1: $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

classical bits are special cases of quantum.

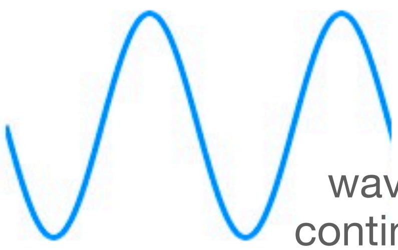
Dirac Notation

A general qubit:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad \text{with } |\alpha|^2 + |\beta|^2 = 1.$$

α, β are general **complex** numbers. Constraint due to Born's **probability amplitude** interpretation.

State Space postulate: (pure) quantum state represented by unit complex vectors



A *quantum bit* (**qubit**) refers to a quantum system of dimension 2

classical 0 and 1: $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

classical bits are special cases of quantum.

Dirac Notation

A general qubit:

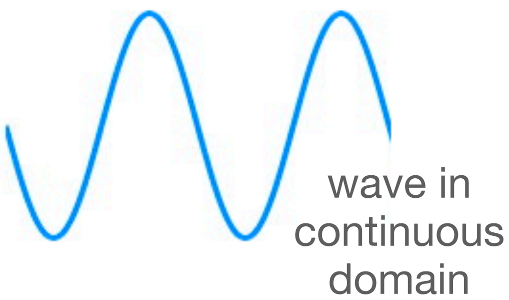
$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad \text{with } |\alpha|^2 + |\beta|^2 = 1.$$

α, β are general **complex** numbers. Constraint due to Born's **probability amplitude** interpretation.

Example:

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

State Space postulate: (pure) quantum state represented by unit complex vectors



A *quantum bit* (**qubit**) refers to a quantum system of dimension 2

classical 0 and 1: $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

classical bits are special cases of quantum.

Dirac Notation

A general qubit:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad \text{with } |\alpha|^2 + |\beta|^2 = 1.$$

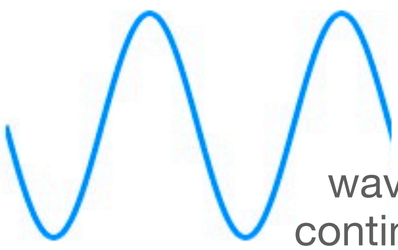
α, β are general **complex** numbers. Constraint due to Born's **probability amplitude** interpretation.

Example: $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ $\quad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$

Evolution postulate: evolution of quantum systems is **unitary**

Unitary evolution is a simple consequence of being linear and preserving ℓ_2 norm

State Space postulate: (pure) quantum state represented by unit complex vectors



A *quantum bit* (**qubit**) refers to a quantum system of dimension 2

classical 0 and 1: $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

classical bits are special cases of quantum.

Dirac Notation

A general qubit:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad \text{with } |\alpha|^2 + |\beta|^2 = 1.$$

α, β are general **complex** numbers. Constraint due to Born's **probability amplitude** interpretation.

Example: $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$

Evolution postulate: evolution of quantum systems is **unitary**

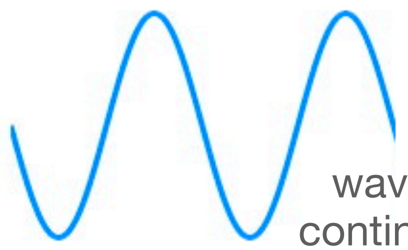
Unitary evolution is a simple consequence of being linear and preserving ℓ_2 norm

Precisely, $|\psi\rangle \mapsto U|\psi\rangle$ since $U|\psi\rangle$ is also a quantum state, so that

$$\langle\psi|U^\dagger U|\psi\rangle = 1, \forall |\psi\rangle \implies U^\dagger U = I$$

unitary (reversible)

State Space postulate: (pure) quantum state represented by unit complex vectors



A *quantum bit* (**qubit**) refers to a quantum system of dimension 2

classical 0 and 1: $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

classical bits are special cases of quantum.

Dirac Notation

A general qubit:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad \text{with } |\alpha|^2 + |\beta|^2 = 1.$$

α, β are general **complex** numbers. Constraint due to Born's **probability amplitude** interpretation.

Example: $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ $\quad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$

Evolution postulate: evolution of quantum systems is **unitary**

Unitary evolution is a simple consequence of being linear and preserving ℓ_2 norm

Precisely, $|\psi\rangle \mapsto U|\psi\rangle$ since $U|\psi\rangle$ is also a quantum state, so that

$$\langle\psi|U^\dagger U|\psi\rangle = 1, \forall|\psi\rangle \implies U^\dagger U = I$$

unitary (reversible)

Example: $H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ $\quad H|0\rangle = |+\rangle, H|1\rangle = |-\rangle$

Composite System postulate: joint system (A,B) in the **tensor-product** of A and B

The representation of two qubits lies in $\mathbb{C}^2 \otimes \mathbb{C}^2$ (dim-4), where $\mathbb{C}^2$ (dim-2) is for a qubit.

Composite System postulate: joint system (A,B) in the **tensor-product** of A and B

The representation of two qubits lies in $\mathbb{C}^2 \otimes \mathbb{C}^2$ (dim-4), where $\mathbb{C}^2$ (dim-2) is for a qubit.

So $|00\rangle = |0\rangle \otimes |0\rangle$ $\quad |01\rangle \quad |10\rangle \quad |11\rangle$

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Composite System postulate: joint system (A,B) in the **tensor-product** of A and B

The representation of two qubits lies in $\mathbb{C}^2 \otimes \mathbb{C}^2$ (dim-4), where $\mathbb{C}^2$ (dim-2) is for a qubit.

So $|00\rangle = |0\rangle \otimes |0\rangle$ $\qquad |01\rangle \qquad |10\rangle \qquad |11\rangle$

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

A **n**-qubit system requires 2^n dimensional space. Exponential cost in classical simulation!

Composite System postulate: joint system (A,B) in the **tensor-product** of A and B

The representation of two qubits lies in $\mathbb{C}^2 \otimes \mathbb{C}^2$ (dim-4), where $\mathbb{C}^2$ (dim-2) is for a qubit.

So $|00\rangle = |0\rangle \otimes |0\rangle$ $\quad |01\rangle \quad |10\rangle \quad |11\rangle$

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

A **n**-qubit system requires 2^n dimensional space. Exponential cost in classical simulation!

Examples of Common Quantum Gates

- Pauli gates: **Single-qubit Gate**

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

- Hadarmard gate:

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

- Rotation about $x-$axis of the Bloch sphere:

$$R_x(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

- The controlled-NOT (CNOT) gate: **Two-qubit Gate**

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Composite System postulate: joint system (A,B) in the **tensor-product** of A and B

The representation of two qubits lies in $\mathbb{C}^2 \otimes \mathbb{C}^2$ (dim-4), where $\mathbb{C}^2$ (dim-2) is for a qubit.

So $|00\rangle = |0\rangle \otimes |0\rangle$ $\quad |01\rangle \quad |10\rangle \quad |11\rangle$

$$\begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1\\0\\0\\0\end{pmatrix} \quad \begin{pmatrix}0\\1\\0\\0\end{pmatrix} \quad \begin{pmatrix}0\\0\\1\\0\end{pmatrix} \quad \begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

A **n**-qubit system requires 2^n dimensional space. Exponential cost in classical simulation!

Examples of Common Quantum Gates

- Pauli gates: **Single-qubit Gate**

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

- Hadarmard gate:

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

- Rotation about $x-$axis of the Bloch sphere:

$$R_x(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

- The controlled-NOT (CNOT) gate: **Two-qubit Gate**

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

NO-CLONING Theorem

Assume a cloning procedure U, then

$$U|0\rangle|0\rangle = |0\rangle|0\rangle \qquad U|1\rangle|0\rangle = |1\rangle|1\rangle$$

Composite System **postulate:** joint system (A,B) in the **tensor-product** of A and B

The representation of two qubits lies in $\mathbb{C}^2 \otimes \mathbb{C}^2$ (dim-4), where $\mathbb{C}^2$ (dim-2) is for a qubit.

So $|00\rangle = |0\rangle \otimes |0\rangle$

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

$|01\rangle$ $\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$ $|10\rangle$ $\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$ $|11\rangle$ $\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$

A **n**-qubit system requires 2^n dimensional space. Exponential cost in classical simulation!

Examples of Common Quantum Gates

- Pauli gates: **Single-qubit Gate**

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

- Hadarmard gate:

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

- Rotation about $x-$axis of the Bloch sphere:

$$R_x(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

- The controlled-NOT (CNOT) gate: **Two-qubit Gate**

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

NO-CLONING Theorem

Assume a cloning procedure U, then

$$U|0\rangle|0\rangle = |0\rangle|0\rangle \qquad U|1\rangle|0\rangle = |1\rangle|1\rangle$$

Consider an arbitrary state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$

$$U|\psi\rangle|0\rangle = \alpha|0\rangle|0\rangle + \beta|1\rangle|1\rangle$$
$$\neq |\psi\rangle|\psi\rangle$$

CONTRADICTION!

Measurement postulate: how to read classical info out of q. system?

This information reading procedure will distribute/collapse the underlying q. systems.

Measurement postulate: how to read classical info out of q. system?

This information reading procedure will distribute/collapse the underlying q. systems.

- A *measurement* is modelled as a set of operators $M = \{M_m\}$ with $\sum_m M_m^\dagger M_m = I$.
- If a quantum system was in pure state $|\psi\rangle$ before the measurement, then:
  - the probability that measurement outcome is λ:

    $$p(m) = ||M_m|\psi\rangle||^2$$

    where $||\cdot||$ is the length of vector.
  - the state of the system after the measurement:

    $$\frac{M_m|\psi\rangle}{\sqrt{p(m)}}$$

Measurement postulate: how to read classical info out of q. system?

This information reading procedure will distribute/collapse the underlying q. systems.

- A *measurement* is modelled as a set of operators $M = \{M_m\}$ with $\sum_m M_m^\dagger M_m = I$.
- If a quantum system was in pure state $|\psi\rangle$ before the measurement, then:
  - the probability that measurement outcome is λ:

$$p(m) = ||M_m|\psi\rangle||^2$$

    where $||\cdot||$ is the length of vector.
  - the state of the system after the measurement:

$$\frac{M_m|\psi\rangle}{\sqrt{p(m)}}$$

Examples

Consider $|0\rangle$

Measured in $\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$

-> $|0\rangle$ w/ prob. 1 (recover classical)

Measurement postulate: how to read classical info out of q. system?

This information reading procedure will distribute/collapse the underlying q. systems.

- A *measurement* is modelled as a set of operators $M = \{M_m\}$ with $\sum_m M_m^\dagger M_m = I$.
- If a quantum system was in pure state $|\psi\rangle$ before the measurement, then:
  - the probability that measurement outcome is λ:

$$p(m) = || M_m |\psi\rangle ||^2$$

    where $|| \cdot ||$ is the length of vector.
  - the state of the system after the measurement:

$$\frac{M_m|\psi\rangle}{\sqrt{p(m)}}$$

Examples Consider $|0\rangle$

Measured in $\{ |0\rangle\langle 0|, |1\rangle\langle 1| \}$

-> $|0\rangle$ w/ prob. 1 (recover classical)

Measured in $\{ |+\rangle\langle +|, |-\rangle\langle -| \}$

-> $|+\rangle$ w/ prob. 0.5

-> $|-\rangle$ w/ prob. 0.5

Measurement postulate: how to read classical info out of q. system?

This information reading procedure will distribute/collapse the underlying q. systems.

- A *measurement* is modelled as a set of operators $M = \{M_m\}$ with $\sum_m M_m^\dagger M_m = I$.
- If a quantum system was in pure state $|\psi\rangle$ before the measurement, then:
  - the probability that measurement outcome is λ:
  $$p(m) = ||M_m|\psi\rangle||^2$$
  where $||\cdot||$ is the length of vector.
  - the state of the system after the measurement:
  $$\frac{M_m|\psi\rangle}{\sqrt{p(m)}}$$

Examples Consider $|0\rangle$

Measured in $\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$

-> $|0\rangle$ w/ prob. 1 (recover classical)

Measured in $\{|+\rangle\langle +|, |-\rangle\langle -|\}$

-> $|+\rangle$ w/ prob. 0.5

-> $|-\rangle$ w/ prob. 0.5

More advanced math formulation of ensemble of quantum states

Density matrices

- In the n-dimensional Hilbert space $\mathbb{C}^n$, an operator is represented by an $n \times n$ complex matrix A.
- The trace of an operator A is $tr(A) = \sum_i A_{ii}$ (the sum of the entries on the main diagonal).
- A positive semidefinite matrix ρ is called a *partial density matrix* if $tr(\rho) \leq 1$; in particular, a *density matrix* ρ is a partial density matrix with $tr(\rho) = 1$.

- For any mixed state $\{(p_1, |\psi_1\rangle), ..., (p_k, |\psi_k\rangle)\}$,

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|$$

Measurement postulate: how to read classical info out of q. system?

This information reading procedure will distribute/collapse the underlying q. systems.

- A *measurement* is modelled as a set of operators $M = \{M_m\}$ with $\sum_m M_m^{\dagger} M_m = I$.
- If a quantum system was in pure state $|\psi\rangle$ before the measurement, then:
  - the probability that measurement outcome is λ:

$$p(m) = ||M_m|\psi\rangle||^2$$

    where $||\cdot||$ is the length of vector.
  - the state of the system after the measurement:

$$\frac{M_m|\psi\rangle}{\sqrt{p(m)}}$$

Examples Consider $|0\rangle$

Measured in $\{ |0\rangle\langle 0|, |1\rangle\langle 1| \}$

-> $|0\rangle$ w/ prob. 1 (recover classical)

Measured in $\{ |+\rangle\langle +|, |-\rangle\langle -| \}$

-> $|+\rangle$ w/ prob. 0.5

-> $|-\rangle$ w/ prob. 0.5

More advanced math formulation of ensemble of quantum states

Density matrices

- In the n-dimensional Hilbert space $\mathbb{C}^n$, an operator is represented by an $n \times n$ complex matrix A.
- The trace of an operator A is $tr(A) = \sum_i A_{ii}$ (the sum of the entries on the main diagonal).
- A positive semidefinite matrix ρ is called a *partial density matrix* if $tr(\rho) \leq 1$; in particular, a *density matrix* ρ is a partial density matrix with $tr(\rho) = 1$.

- For any mixed state $\{(p_1, |\psi_1\rangle), ..., (p_k, |\psi_k\rangle)\}$,

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|$$

Example:

$$\{(\frac{2}{3}, |0\rangle), (\frac{1}{3}, |-\rangle)\} \rightarrow \rho = \frac{2}{3}|0\rangle\langle 0| + \frac{1}{3}|-\rangle\langle -| = \frac{1}{6}\begin{pmatrix} 5 & -1 \\ -1 & 1 \end{pmatrix}$$

Quantum While-Language

Syntax

A ***core*** language for imperative quantum programming

$$
\begin{aligned}
S ::= \mathbf{skip} \mid q &:= |0\rangle \\
&\mid S_1; S_2 \\
&\mid \overline{q} := U[\overline{q}] \\
&\mid \mathbf{if}\ \left(\Box m \cdot M[\overline{q}] = m \rightarrow S_m\right)\ \mathbf{fi} \\
&\mid \mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ S\ \mathbf{od}
\end{aligned}
$$

Quantum While-Language

Syntax

A ***core*** language for imperative quantum programming

$$
\begin{aligned}
S ::= \mathbf{skip} \mid & \, q := |0\rangle \\
\mid & \, S_1; S_2 \\
\mid & \, \overline{q} := U[\overline{q}] \\
\mid & \, \mathbf{if}\ (\square m \cdot M[\overline{q}] = m \rightarrow S_m)\ \mathbf{fi} \\
\mid & \, \mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ S\ \mathbf{od}
\end{aligned}
$$

Classically, one has

$u := t$ t ~ expression.

However, due to no-cloning,

1) initialization
2) unitary operation

Quantum While-Language

Syntax

Quantum Data, Classical Control

A ***core*** language for imperative quantum programming

$$
\begin{aligned}
S ::= \mathbf{skip}\ & |\ q := |0\rangle \\
& |\ S_1; S_2 \\
& |\ \overline{q} := U[\overline{q}] \\
& |\ \mathbf{if}\ (\square m \cdot M[\overline{q}] = m \rightarrow S_m)\ \mathbf{fi} \\
& |\ \mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ S\ \mathbf{od}
\end{aligned}
$$

Classically, one has

$u := t$ t ~ expression.

However, due to no-cloning,

1) initialization
2) unitary operation

Quantum While-Language

Syntax

Quantum Data, Classical Control

A ***core*** language for imperative quantum programming

$$
\begin{aligned}
S ::= \mathbf{skip} \;&|\; q := |0\rangle \\
&|\; S_1; S_2 \\
&|\; \overline{q} := U[\overline{q}] \\
&|\; \mathbf{if}\ (\square m \cdot M[\overline{q}] = m \rightarrow S_m)\ \mathbf{fi} \\
&|\; \mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ S\ \mathbf{od}
\end{aligned}
$$

Classically, one has
$u := t$ t ~ expression.

However, due to no-cloning,
1) initialization
2) unitary operation

Classical control requires reading information out of quantum systems.

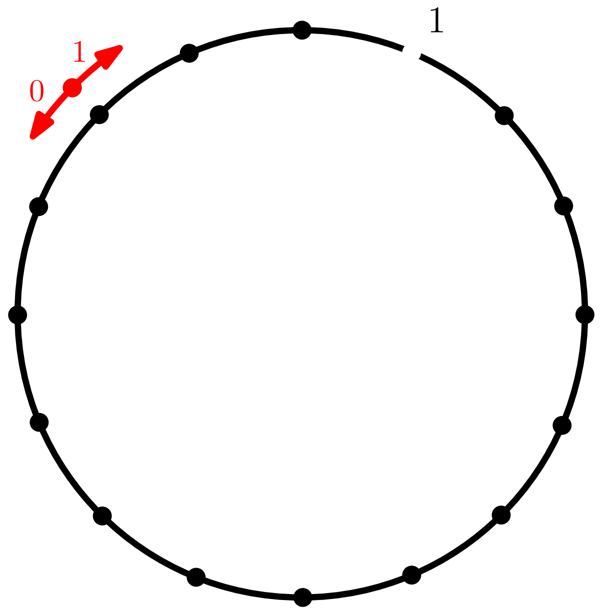
However, by measuring the guard, it leads to

(1) a probabilistic choice of branches

(2) a collapse of the guard state before entering each branch

Quantum 1-D Loop Walk

$$
\begin{aligned}
QW \equiv\, & c := |L\rangle; \\
& p := |0\rangle; \\
& \textbf{while } M[p] = no \textbf{ do} \\
& \quad c := H[c]; \\
& \quad c, p := S[c, p] \\
& \textbf{od}
\end{aligned}
$$



Operator Definition

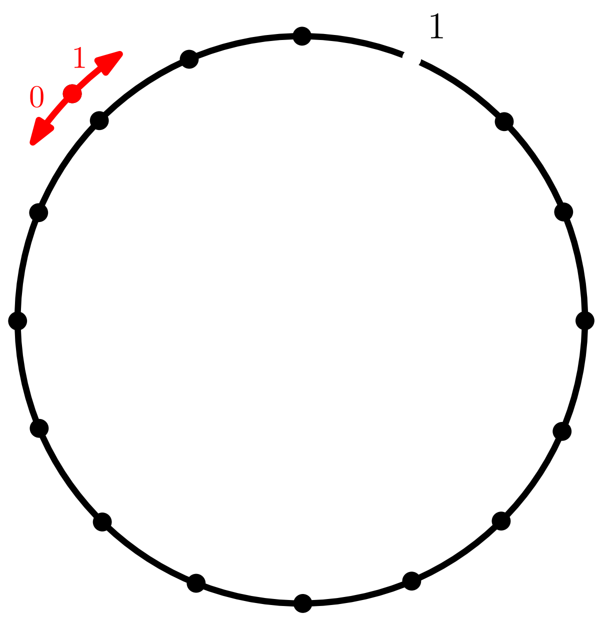
$$S = \sum_{i=0}^{n-1} |L\rangle\langle L| \otimes |i \ominus 1\rangle\langle i| + \sum_{i=0}^{n-1} |R\rangle\langle R| \otimes |i \oplus 1\rangle\langle i|.$$

Quantum 1-D Loop Walk

$$
\begin{aligned}
QW \equiv\ & c := |L\rangle; \\
& p := |0\rangle; \\
& \textbf{while } M[p] = no \textbf{ do} \\
& \quad c := H[c]; \\
& \quad c, p := S[c, p] \\
& \textbf{od}
\end{aligned}
$$

coin space = {L, R}

position space = {0, …, n-1}



Operator Definition

$$S = \sum_{i=0}^{n-1} |L\rangle\langle L| \otimes |i \ominus 1\rangle\langle i| + \sum_{i=0}^{n-1} |R\rangle\langle R| \otimes |i \oplus 1\rangle\langle i|.$$

Quantum 1-D Loop Walk

$QW \equiv c := |L\rangle;$ coin space = {L, R}

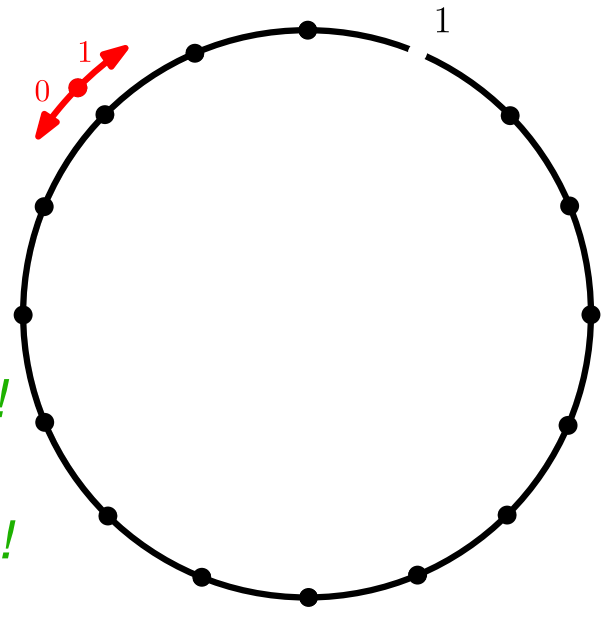
$p := |0\rangle;$ position space = {0, …, n-1}

while $M[p] = no$ **do**

$c := H[c];$ Create a new coin in superposition!

$c, p := S[c, p]$ Random walk based on that coin!

od



Operator Definition

$$S = \sum_{i=0}^{n-1} |L\rangle\langle L| \otimes |i \ominus 1\rangle\langle i| + \sum_{i=0}^{n-1} |R\rangle\langle R| \otimes |i \oplus 1\rangle\langle i|.$$

Quantum 1-D Loop Walk

$QW \equiv c := |L\rangle;$ coin space = {L, R}

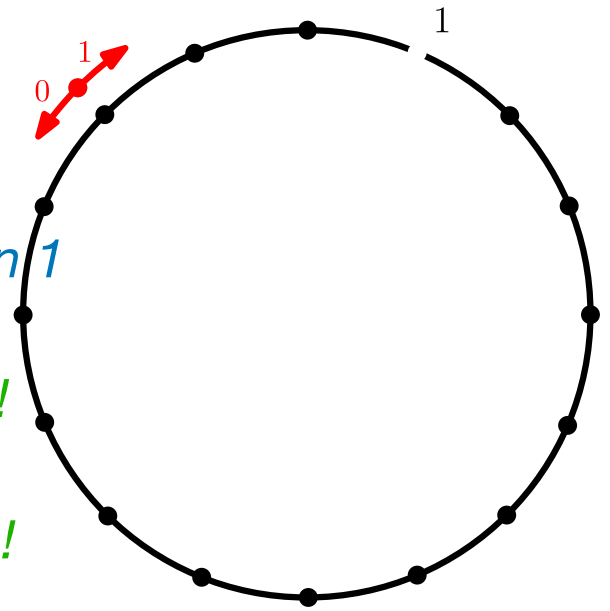
$p := |0\rangle;$ position space = {0, …, n-1}

while $M[p] = no$ **do** Terminal of loop: position 1

$c := H[c];$ Create a new coin in superposition!

$c, p := S[c, p]$ Random walk based on that coin!

od



Operator Definition

$$S = \sum_{i=0}^{n-1} |L\rangle\langle L| \otimes |i \ominus 1\rangle\langle i| + \sum_{i=0}^{n-1} |R\rangle\langle R| \otimes |i \oplus 1\rangle\langle i|.$$

Quantum 1-D Loop Walk

Goal: reason about this program

$$\begin{aligned}QW \equiv\, & c := |L\rangle; \\ & p := |0\rangle; \\ & \textbf{while } M[p] = no \textbf{ do} \\ & \quad c := H[c]; \\ & \quad c, p := S[c, p] \\ & \textbf{od}\end{aligned}$$

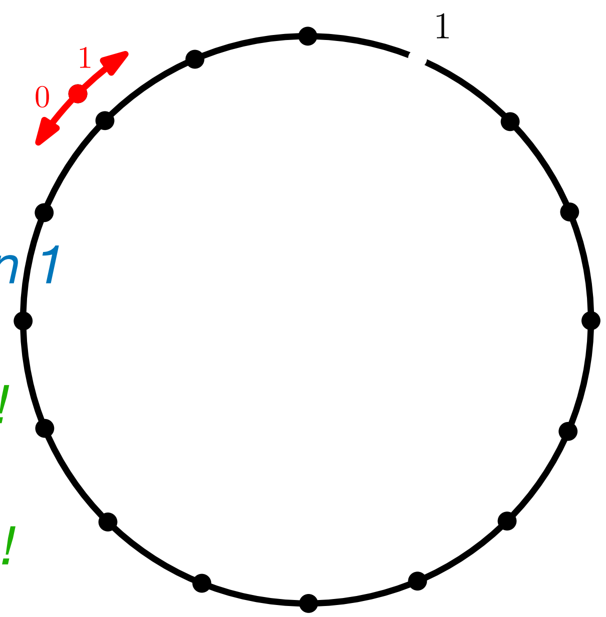
coin space = {L, R}

position space = {0, …, n-1}

Terminal of loop: position 1

Create a new coin in superposition!

Random walk based on that coin!



Operator Definition

$$S = \sum_{i=0}^{n-1} |L\rangle\langle L| \otimes |i \ominus 1\rangle\langle i| + \sum_{i=0}^{n-1} |R\rangle\langle R| \otimes |i \oplus 1\rangle\langle i|.$$

Semantics of Quantum While-Language

Operational Semantics

A *configuration*: $\langle S, \rho\rangle$

- S is a quantum program or E (the empty program)
- ρ is a partial density operator in

$$\mathcal{H}_{\text{all}} = \bigotimes_{\text{all } q} \mathcal{H}_q$$

$$(Sk) \qquad \frac{}{\langle \textbf{skip}, \rho\rangle \to \langle E, \rho\rangle}$$

$$(Ini) \qquad \frac{}{\langle q := |0\rangle, \rho\rangle \to \langle E, \rho_0^q\rangle}$$

- $type(q) = \textbf{Boolean}$:

$$\rho_0^q = |0\rangle_q\langle 0|\rho|0\rangle_q\langle 0| + |0\rangle_q\langle 1|\rho|1\rangle_q\langle 0|$$

- $type(q) = \textbf{integer}$:

$$\rho_0^q = \sum_{n=-\infty}^{\infty} |0\rangle_q\langle n|\rho|n\rangle_q\langle 0|$$

Semantics of Quantum While-Language

Operational Semantics

A *configuration*: $\langle S, \rho \rangle$

- S is a quantum program or E (the empty program)
- ρ is a partial density operator in

$$\mathcal{H}_{\text{all}} = \bigotimes_{\text{all } q} \mathcal{H}_q$$

$$(Sk) \qquad \frac{}{\langle \mathbf{skip}, \rho \rangle \to \langle E, \rho \rangle}$$

$$(Ini) \qquad \frac{}{\langle q := |0\rangle, \rho \rangle \to \langle E, \rho_0^q \rangle}$$

- $type(q) = \mathbf{Boolean}$:

$$\rho_0^q = |0\rangle_q\langle 0|\rho|0\rangle_q\langle 0| + |0\rangle_q\langle 1|\rho|1\rangle_q\langle 0|$$

- $type(q) = \mathbf{integer}$:

$$\rho_0^q = \sum_{n=-\infty}^{\infty} |0\rangle_q\langle n|\rho|n\rangle_q\langle 0|$$

$$(Uni) \qquad \frac{}{\langle \overline{q} := U[\overline{q}], \rho \rangle \to \langle E, U\rho U^\dagger \rangle}$$

$$(Seq) \qquad \frac{\langle S_1, \rho \rangle \to \langle S_1', \rho' \rangle}{\langle S_1; S_2, \rho \rangle \to \langle S_1'; S_2, \rho' \rangle}$$

$$\text{Convention}: \ E; S_2 = S_2.$$

$$(IF) \qquad \frac{}{\langle \mathbf{if}\ (\Box m \cdot M[\overline{q}] = m \to S_m)\ \mathbf{fi}, \rho \rangle \to \langle S_m, M_m \rho M_m^\dagger \rangle}$$

for each outcome m

Semantics of Quantum While-Language

Operational Semantics

A *configuration*: $\langle S, \rho \rangle$

- S is a quantum program or E (the empty program)
- ρ is a partial density operator in

$$\mathcal{H}_{\text{all}} = \bigotimes_{\text{all } q} \mathcal{H}_q$$

$$(Sk) \qquad \frac{}{\langle \mathbf{skip}, \rho \rangle \rightarrow \langle E, \rho \rangle}$$

$$(Ini) \qquad \frac{}{\langle q := |0\rangle, \rho \rangle \rightarrow \langle E, \rho_0^q \rangle}$$

- $type(q) =$ **Boolean**:

$$\rho_0^q = |0\rangle_q \langle 0|\rho|0\rangle_q \langle 0| + |0\rangle_q \langle 1|\rho|1\rangle_q \langle 0|$$

- $type(q) =$ **integer**:

$$\rho_0^q = \sum_{n=-\infty}^{\infty} |0\rangle_q \langle n|\rho|n\rangle_q \langle 0|$$

$$(Uni) \qquad \frac{}{\langle \overline{q} := U[\overline{q}], \rho \rangle \rightarrow \langle E, U\rho U^\dagger \rangle}$$

$$(Seq) \qquad \frac{\langle S_1, \rho \rangle \rightarrow \langle S_1', \rho' \rangle}{\langle S_1; S_2, \rho \rangle \rightarrow \langle S_1'; S_2, \rho' \rangle}$$

$$\text{Convention}: \ E; S_2 = S_2.$$

$$(IF) \qquad \frac{}{\langle \mathbf{if} \ (\square m \cdot M[\overline{q}] = m \rightarrow S_m) \ \mathbf{fi}, \rho \rangle \rightarrow \langle S_m, M_m \rho M_m^\dagger \rangle}$$

for each outcome m

Loop:

$$(L0) \qquad \frac{}{\langle \mathbf{while} \ M[\overline{q}] = 1 \ \mathbf{do} \ S \ \mathbf{od}, \rho \rangle \rightarrow \langle E, M_0 \rho M_0^\dagger \rangle}$$

$$(L1) \qquad \frac{}{\langle \mathbf{while} \ M[\overline{q}] = 1 \ \mathbf{do} \ S, \rho \rangle \rightarrow \langle S; \mathbf{while} \ M[\overline{q}] = 1 \ \mathbf{do} \ S, M_1 \rho M_1^\dagger \rangle}$$

Semantics of Quantum While-Language

Operational Semantics

A *configuration*: $\langle S, \rho\rangle$

- S is a quantum program or E (the empty program)
- ρ is a partial density operator in

$$\mathcal{H}_{\text{all}} = \bigotimes_{\text{all } q} \mathcal{H}_q$$

(Sk) $$\frac{}{\langle \mathbf{skip}, \rho\rangle \rightarrow \langle E, \rho\rangle}$$

(Ini) $$\frac{}{\langle q := |0\rangle, \rho\rangle \rightarrow \langle E, \rho_0^q\rangle}$$

- $type(q) = $ **Boolean**:

$$\rho_0^q = |0\rangle_q\langle 0|\rho|0\rangle_q\langle 0| + |0\rangle_q\langle 1|\rho|1\rangle_q\langle 0|$$

- $type(q) = $ **integer**:

$$\rho_0^q = \sum_{n=-\infty}^{\infty} |0\rangle_q\langle n|\rho|n\rangle_q\langle 0|$$

(Uni) $$\frac{}{\langle \overline{q} := U[\overline{q}], \rho\rangle \rightarrow \langle E, U\rho U^\dagger\rangle}$$

(Seq) $$\frac{\langle S_1, \rho\rangle \rightarrow \langle S_1', \rho'\rangle}{\langle S_1; S_2, \rho\rangle \rightarrow \langle S_1'; S_2, \rho'\rangle}$$

$$\text{Convention}: \ E; S_2 = S_2.$$

(IF) $$\frac{}{\langle \mathbf{if}\ (\Box m \cdot M[\overline{q}] = m \rightarrow S_m)\ \mathbf{fi}, \rho\rangle \rightarrow \langle S_m, M_m\rho M_m^\dagger\rangle}$$

for each outcome m

Loop:

(L0) $$\frac{}{\langle \mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ S\ \mathbf{od}, \rho\rangle \rightarrow \langle E, M_0\rho M_0^\dagger\rangle}$$

(L1) $$\frac{}{\langle \mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ S, \rho\rangle \rightarrow \langle S; \mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ S, M_1\rho M_1^\dagger\rangle}$$

Capture the **Collapse** of the **Guard** state.

Semantics of Quantum While-Language

Denotational Semantics

Semantic function of quantum program S:

$$[\![S]\!] : \mathcal{D}(\mathcal{H}_{\text{all}}) \rightarrow \mathcal{D}(\mathcal{H}_{\text{all}})$$

$$[\![S]\!](\rho) = \sum \{|\rho' : \langle S, \rho\rangle \rightarrow^* \langle E, \rho'\rangle|\} \text{ for all } \rho \in \mathcal{D}(\mathcal{H}_{\text{all}})$$

Semantics of Quantum While-Language

Denotational Semantics

Semantic function of quantum program S:

$$[\![S]\!] : \mathcal{D}(\mathcal{H}_{\text{all}}) \to \mathcal{D}(\mathcal{H}_{\text{all}})$$

$$[\![S]\!](\rho) = \sum\{|\rho' : \langle S, \rho\rangle \to^* \langle E, \rho'\rangle|\} \text{ for all } \rho \in \mathcal{D}(\mathcal{H}_{\text{all}})$$

Observation:

$$tr([\![S]\!](\rho)) \leq tr(\rho)$$

for any quantum program S and all $\rho \in \mathcal{D}(\mathcal{H}_{\text{all}})$.

- $tr(\rho) - tr([\![S]\!](\rho))$ is the probability that program S diverges from input state ρ.

Quantum Predicate & Hoare Triple

- A *quantum predicate* is a Hermitian operator (obsevable) P such that $0 \sqsubseteq P \sqsubseteq I$.

[1] E. D'Hondt and P. Panangaden, Quantum weakest preconditions, *Mathematical Structures in Computer Science* 2006.

Quantum Predicate & Hoare Triple

- A *quantum predicate* is a Hermitian operator (obsevable) P such that $0 \sqsubseteq P \sqsubseteq I$.

  [1] E. D'Hondt and P. Panangaden, Quantum weakest preconditions, *Mathematical Structures in Computer Science* 2006.

Continuous logic
[0, 1]
Matrix Upgrade

Quantum Predicate & Hoare Triple

- A *quantum predicate* is a Hermitian operator (obsevable) P such that $0 \sqsubseteq P \sqsubseteq I$.

  [1] E. D'Hondt and P. Panangaden, Quantum weakest preconditions, *Mathematical Structures in Computer Science* 2006.

Continuous logic
[0, 1]
Matrix Upgrade

- A *correctness formula* is a statement of the form:

$$\{P\}S\{Q\}$$

  where:

  - S is a quantum program
  - P and Q are quantum predicates.
  - Operator P is called the *precondition* and Q the *postcondition*.

Quantum Predicate & Hoare Triple

- A *quantum predicate* is a Hermitian operator (obsevable) P such that $0 \sqsubseteq P \sqsubseteq I$.

  [1] E. D'Hondt and P. Panangaden, Quantum weakest preconditions, *Mathematical Structures in Computer Science* 2006.

Continuous logic
[0, 1]
Matrix Upgrade

- A *correctness formula* is a statement of the form:

$$\{P\}S\{Q\}$$

  where:

  - S is a quantum program
  - P and Q are quantum predicates.
  - Operator P is called the *precondition* and Q the *postcondition*.

Similar as Classical
Hoare triple w/
different semantics

Quantum Predicate & Hoare Triple

- A *quantum predicate* is a Hermitian operator (obsevable) P such that $0 \sqsubseteq P \sqsubseteq I$.

  [1] E. D'Hondt and P. Panangaden, Quantum weakest preconditions, *Mathematical Structures in Computer Science* 2006.

Continuous logic
[0, 1]
Matrix Upgrade

- A *correctness formula* is a statement of the form:

$$\{P\}S\{Q\}$$

  where:
  - S is a quantum program
  - P and Q are quantum predicates.
  - Operator P is called the *precondition* and Q the *postcondition*.

Similar as Classical
Hoare triple w/
different semantics

1. $\{P\}S\{Q\}$ is true in the sense of *total correctness*:

$$\models_{\text{tot}} \{P\}S\{Q\}$$

   if

$$tr(P\rho) \leq tr(Q[\![S]\!](\rho)) \text{ for all } \rho.$$

Quantum Predicate & Hoare Triple

- A *quantum predicate* is a Hermitian operator (obsevable) P such that $0 \sqsubseteq P \sqsubseteq I$.

  [1] E. D'Hondt and P. Panangaden, Quantum weakest preconditions, *Mathematical Structures in Computer Science* 2006.

Continuous logic
[0, 1]
Matrix Upgrade

- A *correctness formula* is a statement of the form:

$$\{P\}S\{Q\}$$

  where:
  - S is a quantum program
  - P and Q are quantum predicates.
  - Operator P is called the *precondition* and Q the *postcondition*.

Similar as Classical Hoare triple w/ different semantics

1. $\{P\}S\{Q\}$ is true in the sense of *total correctness*:

$$\models_{\text{tot}} \{P\}S\{Q\}$$

if

$$tr(P\rho) \leq tr(Q[\![S]\!](\rho)) \text{ for all } \rho.$$

Semantics

Quantum Predicate & Hoare Triple

- A *quantum predicate* is a Hermitian operator (obsevable) P such that $0 \sqsubseteq P \sqsubseteq I$.

  [1] E. D'Hondt and P. Panangaden, Quantum weakest preconditions, *Mathematical Structures in Computer Science* 2006.

Continuous logic
[0, 1]
Matrix Upgrade

- A *correctness formula* is a statement of the form:

$$\{P\}S\{Q\}$$

  where:
  - S is a quantum program
  - P and Q are quantum predicates.
  - Operator P is called the *precondition* and Q the *postcondition*.

Similar as Classical Hoare triple w/ different semantics

1. $\{P\}S\{Q\}$ is true in the sense of *total correctness*:

$$\models_{\text{tot}} \{P\}S\{Q\}$$

if

Pre-S State *Post-S State*

$$tr(P\rho) \leq tr(Q[\![S]\!](\rho)) \text{ for all } \rho.$$

Semantics

Quantum Predicate & Hoare Triple

- A *quantum predicate* is a Hermitian operator (obsevable) P such that $0 \sqsubseteq P \sqsubseteq I$.

  [1] E. D'Hondt and P. Panangaden, Quantum weakest preconditions, *Mathematical Structures in Computer Science* 2006.

Continuous logic
[0, 1]
Matrix Upgrade

- A *correctness formula* is a statement of the form:

$$\{P\}S\{Q\}$$

where:

  - S is a quantum program
  - P and Q are quantum predicates.
  - Operator P is called the *precondition* and Q the *postcondition*.

Similar as Classical Hoare triple w/ different semantics

1. $\{P\}S\{Q\}$ is true in the sense of *total correctness*:

$$\models_{\text{tot}} \{P\}S\{Q\}$$

if

Pre-S State *Post-S State*

$$tr(P\rho) \leq tr(Q[\![S]\!](\rho)) \text{ for all } \rho.$$

Semantics

2. $\{P\}S\{Q\}$ is true in the sense of *partial correctness*:

$$\models_{\text{par}} \{P\}S\{Q\},$$

if

$$tr(P\rho) \leq tr(Q[\![S]\!](\rho)) + [tr(\rho) - tr([\![S]\!](\rho))]$$

Quantum Predicate & Hoare Triple

- A *quantum predicate* is a Hermitian operator (obsevable) P such that $0 \sqsubseteq P \sqsubseteq I$.

  [1] E. D'Hondt and P. Panangaden, Quantum weakest preconditions, *Mathematical Structures in Computer Science* 2006.

Continuous logic
[0, 1]
Matrix Upgrade

- A *correctness formula* is a statement of the form:

$$\{P\}S\{Q\}$$

where:

- S is a quantum program
- P and Q are quantum predicates.
- Operator P is called the *precondition* and Q the *postcondition*.

Similar as Classical Hoare triple w/ different semantics

1. $\{P\}S\{Q\}$ is true in the sense of *total correctness*:

$$\models_{\text{tot}} \{P\}S\{Q\}$$

if

Pre-S State   Post-S State

$$tr(P\rho) \leq tr(Q[\![S]\!](\rho)) \text{ for all } \rho.$$

Semantics

2. $\{P\}S\{Q\}$ is true in the sense of *partial correctness*:

$$\models_{\text{par}} \{P\}S\{Q\},$$

if

$$tr(P\rho) \leq tr(Q[\![S]\!](\rho)) + [tr(\rho) - tr([\![S]\!](\rho))]$$

Divergence

Quantum Hoare logic for Partial Correctness

(*Axiom Sk*) $\{P\}\textbf{Skip}\{P\}$

(*Axiom Ini*)

$type(q) = \textbf{Boolean}$:

$$\{|0\rangle_q\langle 0|P|0\rangle_q\langle 0| + |1\rangle_q\langle 0|P|0\rangle_q\langle 1|\}q := |0\rangle\{P\}$$

$type(q) = \textbf{integer}$:

$$\{\sum_{n=-\infty}^{\infty} |n\rangle_q\langle 0|P|0\rangle_q\langle n|\}q := |0\rangle\{P\}$$

(*Axiom Uni*) $\{U^\dagger P U\}\overline{q} := U[\overline{q}]\{P\}$

(*Rule Seq*)
$$\frac{\{P\}S_1\{Q\} \qquad \{Q\}S_2\{R\}}{\{P\}S_1; S_2\{R\}}$$

(*Rule IF*)
$$\frac{\{P_m\}S_m\{Q\} \text{ for all } m}{\{\sum_m M_m^\dagger P_m M_m\}\textbf{if } (\Box m \cdot M[\overline{q}] = m \to S_m) \textbf{ fi}\{Q\}}$$

(*Rule LP*)
$$\frac{\{Q\}S\{M_0^\dagger P M_0 + M_1^\dagger Q M_1\}}{\{M_0^\dagger P M_0 + M_1^\dagger Q M_1\}\textbf{while } M[\overline{q}] = 1 \textbf{ do } S\{P\}}$$

(*Rule Ord*)
$$\frac{P \sqsubseteq P' \quad \{P'\}S\{Q'\} \quad Q' \sqsubseteq Q}{\{P\}S\{Q\}}$$

Quantum Hoare logic for Partial Correctness

(*Axiom Sk*) $\{P\}\textbf{Skip}\{P\}$

(*Axiom Ini*)

$type(q) = \textbf{Boolean}$:

$$\{|0\rangle_q\langle 0|P|0\rangle_q\langle 0| + |1\rangle_q\langle 0|P|0\rangle_q\langle 1|\}q := |0\rangle\{P\}$$

$type(q) = \textbf{integer}$:

$$\{\sum_{n=-\infty}^{\infty} |n\rangle_q\langle 0|P|0\rangle_q\langle n|\}q := |0\rangle\{P\}$$

(*Axiom Uni*) $\{U^\dagger PU\}\overline{q} := U[\overline{q}]\{P\}$

(*Rule Seq*) $$\frac{\{P\}S_1\{Q\} \quad \{Q\}S_2\{R\}}{\{P\}S_1;S_2\{R\}}$$

(*Rule IF*) $$\frac{\{P_m\}S_m\{Q\} \text{ for all } m}{\{\sum_m M_m^\dagger P_m M_m\}\textbf{if } (\Box m \cdot M[\overline{q}] = m \rightarrow S_m) \textbf{ fi}\{Q\}}$$

(*Rule LP*) $$\frac{\{Q\}S\{M_0^\dagger PM_0 + M_1^\dagger QM_1\}}{\{M_0^\dagger PM_0 + M_1^\dagger QM_1\}\textbf{while } M[\overline{q}] = 1 \textbf{ do } S\{P\}}$$

(*Rule Ord*) $$\frac{P \sqsubseteq P' \quad \{P'\}S\{Q'\} \quad Q' \sqsubseteq Q}{\{P\}S\{Q\}}$$

Parts of Classical Hoare Logic

AXIOM 2: ASSIGNMENT

$$\{p[u := t]\}\ u := t\ \{p\}$$

Quantum Hoare logic for Partial Correctness

(*Axiom Sk*) $\{P\}\mathbf{Skip}\{P\}$

(*Axiom Ini*)

type$(q) = $ **Boolean** :

$$\{|0\rangle_q\langle 0|P|0\rangle_q\langle 0| + |1\rangle_q\langle 0|P|0\rangle_q\langle 1|\}q := |0\rangle\{P\}$$

type$(q) = $ **integer** :

$$\{\sum_{n=-\infty}^{\infty} |n\rangle_q\langle 0|P|0\rangle_q\langle n|\}q := |0\rangle\{P\}$$

(*Axiom Uni*) $\{U^\dagger PU\}\overline{q} := U[\overline{q}]\{P\}$

(*Rule Seq*)

$$\frac{\{P\}S_1\{Q\} \quad \{Q\}S_2\{R\}}{\{P\}S_1; S_2\{R\}}$$

(*Rule IF*)

$$\frac{\{P_m\}S_m\{Q\} \text{ for all } m}{\{\sum_m M_m^\dagger P_m M_m\}\mathbf{if}\ (\Box m \cdot M[\overline{q}] = m \rightarrow S_m)\ \mathbf{fi}\{Q\}}$$

(*Rule LP*)

$$\frac{\{Q\}S\{M_0^\dagger PM_0 + M_1^\dagger QM_1\}}{\{M_0^\dagger PM_0 + M_1^\dagger QM_1\}\mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ S\{P\}}$$

(*Rule Ord*)

$$\frac{P \sqsubseteq P' \quad \{P'\}S\{Q'\} \quad Q' \sqsubseteq Q}{\{P\}S\{Q\}}$$

Parts of Classical Hoare Logic

AXIOM 2: ASSIGNMENT

$$\{p[u := t]\}\ u := t\ \{p\}$$

RULE 4: CONDITIONAL

$$\frac{\{p \wedge B\}\ S_1\ \{q\}, \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q\}}$$

RULE 5: LOOP

$$\frac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{p \wedge \neg B\}}$$

Quantum Hoare logic for Partial Correctness

(*Axiom Sk*) $\{P\}\mathbf{Skip}\{P\}$

(*Axiom Ini*)

$type(q) = \mathbf{Boolean}$:

$$\{|0\rangle_q\langle 0|P|0\rangle_q\langle 0| + |1\rangle_q\langle 0|P|0\rangle_q\langle 1|\}q := |0\rangle\{P\}$$

$type(q) = \mathbf{integer}$:

$$\{\sum_{n=-\infty}^{\infty} |n\rangle_q\langle 0|P|0\rangle_q\langle n|\}q := |0\rangle\{P\}$$

(*Axiom Uni*) $\{U^\dagger PU\}\overline{q} := U[\overline{q}]\{P\}$

(*Rule Seq*) $$\frac{\{P\}S_1\{Q\} \quad \{Q\}S_2\{R\}}{\{P\}S_1; S_2\{R\}}$$

(*Rule IF*) $$\frac{\{P_m\}S_m\{Q\} \text{ for all } m}{\{\sum_m M_m^\dagger P_m M_m\}\mathbf{if}\ (\square m \cdot M[\overline{q}] = m \rightarrow S_m)\ \mathbf{fi}\{Q\}}$$

(*Rule LP*) $$\frac{\{Q\}S\{M_0^\dagger PM_0 + M_1^\dagger QM_1\}}{\{M_0^\dagger PM_0 + M_1^\dagger QM_1\}\mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ S\{P\}}$$

(*Rule Ord*) $$\frac{P \sqsubseteq P' \quad \{P'\}S\{Q'\} \quad Q' \sqsubseteq Q}{\{P\}S\{Q\}}$$

Parts of Classical Hoare Logic

AXIOM 2: ASSIGNMENT

$$\{p[u := t]\}\ u := t\ \{p\}$$

RULE 4: CONDITIONAL

$$\frac{\{p \wedge B\}\ S_1\ \{q\}, \{p \wedge \neg B\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q\}}$$

RULE 5: LOOP

$$\frac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}\ \{p \wedge \neg B\}}$$

Theorem (Soundness and Completeness)

For any quantum program S and quantum predicates P, Q,

$$\models_{\text{par}} \{P\}S\{Q\} \text{ if and only if } \vdash_{PD} \{P\}S\{Q\}.$$

Ying. TOPLAS, 2011.

Quantum Hoare logic for Total Correctness

Proof System for Total Correctness

Let P be a quantum predicate and $\epsilon > 0$. A function

$$t : \mathcal{D}(\mathcal{H}_{\text{all}}) \text{ (density operators)} \rightarrow \mathbb{N}$$

is called a (P, ϵ)-*ranking function* of quantum loop:

$$\textbf{while } M[\overline{q}] = 1 \textbf{ do } S \textbf{ od}$$

if for all ρ:

1. $t(\llbracket S \rrbracket(M_1 \rho M_1^\dagger)) \leq t(\rho)$;
2. $tr(P\rho) \geq \epsilon$ implies $t(\llbracket S \rrbracket(M_1 \rho M_1^\dagger)) < t(\rho)$

$$(\textit{Rule LT})\ \frac{\begin{array}{c}(1)\ \{Q\}S\{M_0^\dagger P M_0 + M_1^\dagger Q M_1\} \\ (2) \text{ for any } \epsilon > 0,\ t_\epsilon \text{ is a } (M_1^\dagger Q M_1, \epsilon)-\text{ranking} \\ \text{function of loop}\end{array}}{\{M_0^\dagger P M_0 + M_1^\dagger Q M_1\}\textbf{while } M[\overline{q}] = 1 \textbf{ do } S \textbf{ od}\{P\}}$$

Theorem (Soundness and Completeness)

For any quantum program S and quantum predicates P Q,

$$\models_{\text{tot}} \{P\}S\{Q\} \text{ if and only if } \vdash_{TD} \{P\}S\{Q\}.$$

[2] M. S. Ying, Floyd-Hoare logic for quantum programs, *ACM Transactions on Programming Languages and Systems* 2011

Quantum Hoare logic and Invariants : POPL17

$$QW \equiv c := |L\rangle;$$ coin space = {L, R}

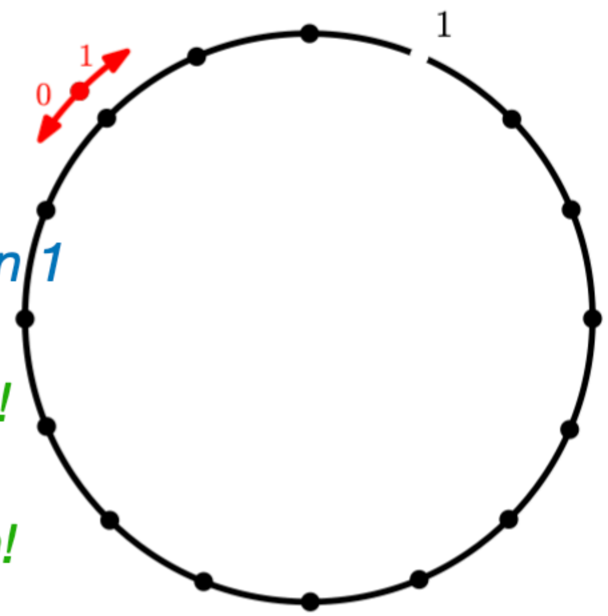
$$p := |0\rangle;$$ position space = {0, …, n-1}

$$\textbf{while } M[p] = no \textbf{ do}$$ Terminal of loop: position 1

$$c := H[c];$$ Create a new coin in superposition!

$$c, p := S[c, p]$$ Random walk based on that coin!

od



Quantum Hoare logic and Invariants : POPL17

$$
\begin{aligned}
QW \equiv\, & c := |L\rangle; \\
& p := |0\rangle; \\
& \textbf{while } M[p] = no \textbf{ do} \\
& \quad c := H[c]; \\
& \quad c,p := S[c,p] \\
& \textbf{od}
\end{aligned}
$$

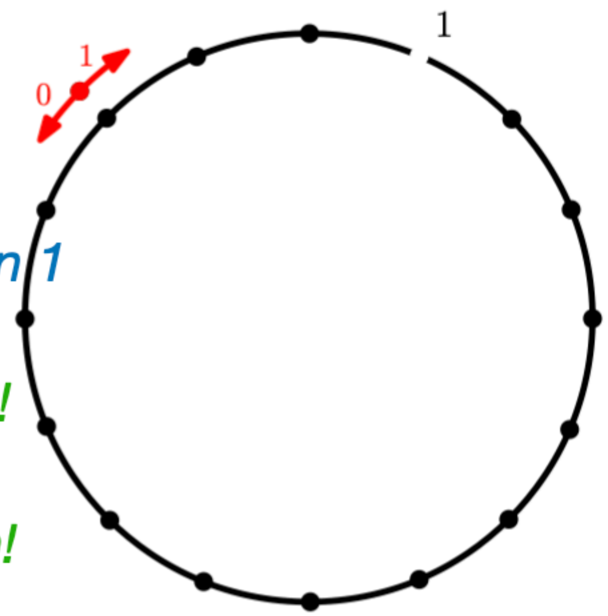
coin space = {L, R}

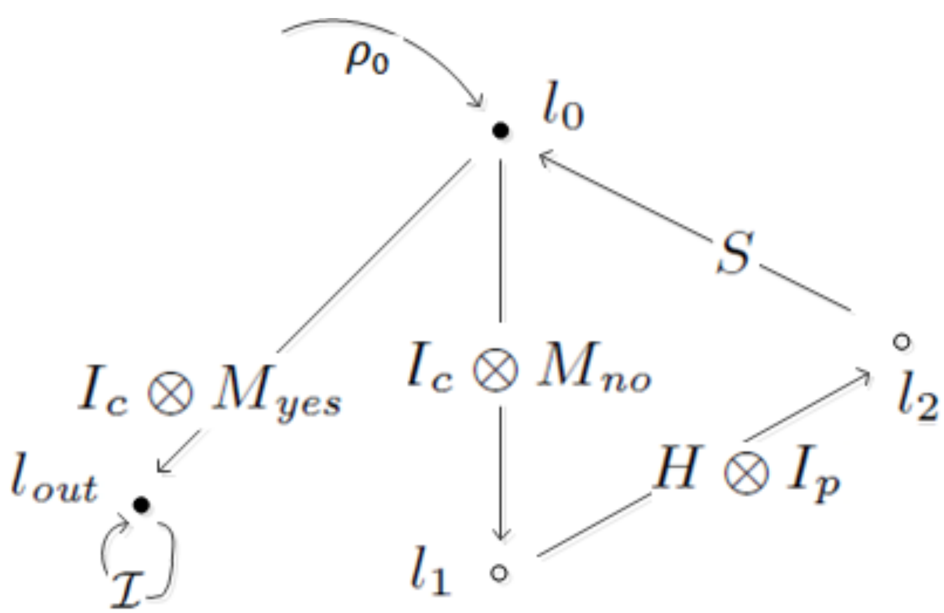
position space = {0, …, n-1}

Terminal of loop: position 1

Create a new coin in superposition!

Random walk based on that coin!





Control - Flow - Graph

Quantum Hoare logic and Invariants : POPL17

$$QW \equiv c := |L\rangle;$$ coin space = {L, R}

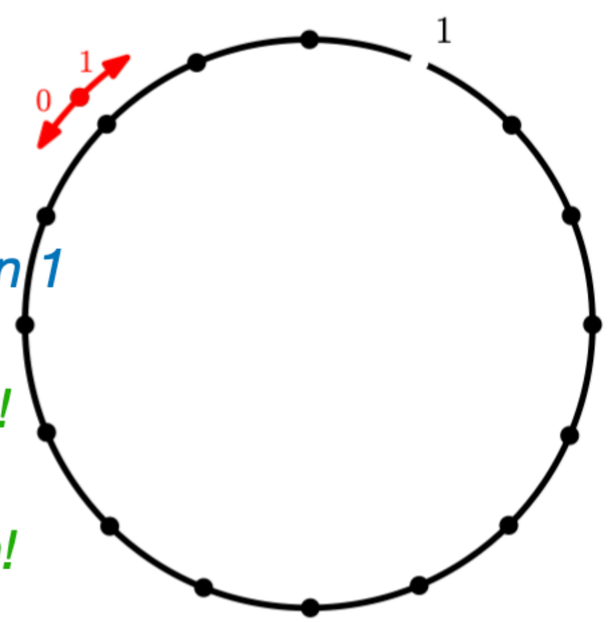
$$p := |0\rangle;$$ position space = {0, ..., n-1}

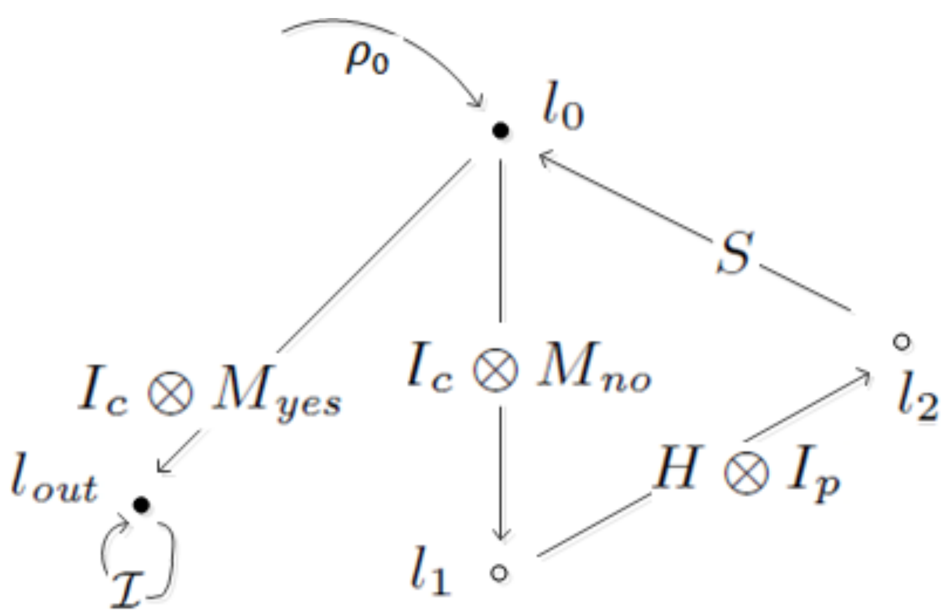
$$\textbf{while } M[p] = no \textbf{ do}$$ Terminal of loop: position 1

$$c := H[c];$$ Create a new coin in superposition!

$$c, p := S[c, p]$$ Random walk based on that coin!

od





Control - Flow - Graph

Invariants

- A set Π of paths is *prime* if for each

$$\pi = l_1 \stackrel{\mathcal{E}_1}{\to} \dots \stackrel{\mathcal{E}_{n-1}}{\to} l_n \in \Pi$$

its proper initial segments $l_1 \stackrel{\mathcal{E}_1}{\to} \dots \stackrel{\mathcal{E}_{k-1}}{\to} l_k \notin \Pi$ for all $k < n$.

- Let $\mathcal{G} = \langle \mathcal{H}, L, l_0, \to \rangle$, Θ a quantum predicate (initial condition), $l \in L$. An *invariant* at l is a quantum predicate O such that for any density operator ρ, any prime set Π of paths from l_0 to l:

$$tr(\Theta\rho) \leq 1 - tr\left(\mathcal{E}_\Pi(\rho)\right) + tr\left(O\mathcal{E}_\Pi(\rho)\right)$$

where $\mathcal{E}_\Pi = \sum \{|\mathcal{E}_\pi : \pi \in \Pi|\}$.

Quantum Hoare logic and Invariants : POPL17

$QW \equiv c := |L\rangle;$ — coin space = {L, R}

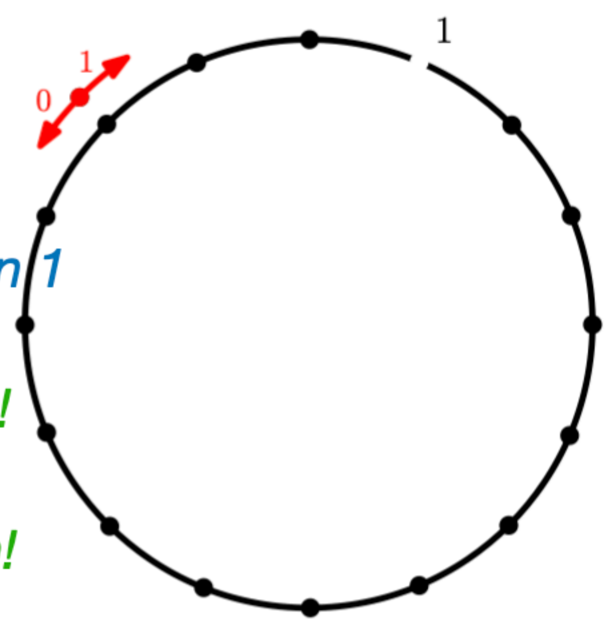
$\quad p := |0\rangle;$ — position space = {0, ..., n-1}

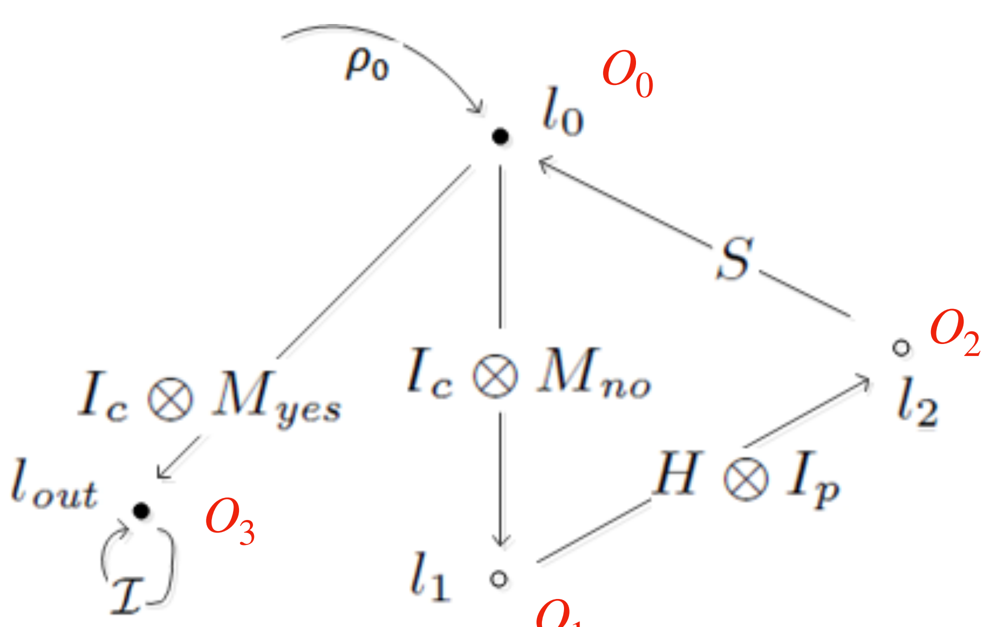
$\quad$ **while** $M[p] = no$ **do** — Terminal of loop: position 1

$\qquad c := H[c];$ — Create a new coin in superposition!

$\qquad c, p := S[c, p]$ — Random walk based on that coin!

$\quad$ **od**





Control - Flow - Graph

Invariants

- A set Π of paths is *prime* if for each

$$\pi = l_1 \xrightarrow{\mathcal{E}_1} \dots \xrightarrow{\mathcal{E}_{n-1}} l_n \in \Pi$$

its proper initial segments $l_1 \xrightarrow{\mathcal{E}_1} \dots \xrightarrow{\mathcal{E}_{k-1}} l_k \notin \Pi$ for all $k < n$.

- Let $\mathcal{G} = \langle \mathcal{H}, L, l_0, \rightarrow \rangle$, Θ a quantum predicate (initial condition), $l \in L$. An *invariant* at l is a quantum predicate O such that for any density operator ρ, any prime set Π of paths from l_0 to l:

$$tr(\Theta\rho) \leq 1 - tr\left(\mathcal{E}_\Pi(\rho)\right) + tr\left(O\mathcal{E}_\Pi(\rho)\right)$$

where $\mathcal{E}_\Pi = \sum \{|\mathcal{E}_\pi : \pi \in \Pi|\}$.

Finding Quantum Invariants

Theorem (Partial Correctness)

Let P be a quantum program. If O is an invariant at l_{out}^{P} in $\mathcal{S}_P$, then

$$\models_{par} \{\Theta\}P\{O\}$$

Finding Quantum Invariants

Theorem (Partial Correctness)

Let P be a quantum program. If O is an invariant at l_{out}^{P} in $\mathcal{S}_P$, then

$$\models_{par} \{\Theta\}P\{O\}$$

Inductive Assertion Maps

- Given $\mathcal{G} = \langle \mathcal{H}, L, l_0, \rightarrow \rangle$ with a cutset C and initial condition Θ.
- An *assertion map* is a mapping η from each cutpoint $l \in C$ to a quantum predicate $\eta(l)$.
- Π_l: the set of all basic paths from l to some cutpoint.
- l_π: the last location in a path π.
- An assertion map η is *inductive* if:
  - **Initiation**: for any density operator ρ:

$$tr(\Theta\rho) \leq 1 - tr\left(\mathcal{E}_{\Pi_{l_0}}(\rho)\right) + \sum_{\pi\in\Pi_{l_0}} tr\left(\eta(l_\pi)\mathcal{E}_\pi(\rho)\right);$$

  - **Consecution**: for any density operator ρ, each cutpoint $l \in C$:

$$tr(\eta(l)\rho) \leq 1 - tr\left(\mathcal{E}_{\Pi_l}(\rho)\right) + \sum_{\pi\in\Pi_l} tr\left(\eta(l_\pi)\mathcal{E}_\pi(\rho)\right).$$

Reducing Global Constraints Into Local Ones

Finding Quantum Invariants

Theorem (Partial Correctness)

Let P be a quantum program. If O is an invariant at l_{out}^{P} in $\mathcal{S}_P$, then

$$\models_{par} \{\Theta\}P\{O\}$$

Inductive Assertion Maps

- Given $\mathcal{G} = \langle \mathcal{H}, L, l_0, \rightarrow \rangle$ with a cutset C and initial condition Θ.
- An *assertion map* is a mapping η from each cutpoint $l \in C$ to a quantum predicate $\eta(l)$.
- Π_l: the set of all basic paths from l to some cutpoint.
- l_π: the last location in a path π.
- An assertion map η is *inductive* if:
  - **Initiation**: for any density operator ρ:

$$tr(\Theta\rho) \leq 1 - tr\left(\mathcal{E}_{\Pi_{l_0}}(\rho)\right) + \sum_{\pi \in \Pi_{l_0}} tr\left(\eta(l_\pi)\mathcal{E}_\pi(\rho)\right);$$

  - **Consecution**: for any density operator ρ, each cutpoint $l \in C$:

$$tr(\eta(l)\rho) \leq 1 - tr\left(\mathcal{E}_{\Pi_l}(\rho)\right) + \sum_{\pi \in \Pi_l} tr\left(\eta(l_\pi)\mathcal{E}_\pi(\rho)\right).$$

Reducing Global Constraints Into Local Ones

Reduce to a SDP (Semi-Definite Programming) Problem

- Assume $C = \{l_0, l_1, ..., l_m\}$.
- Write $O_i = \eta(l_i)$ for $i = 0, 1, ...m$.
- $\mathcal{E}_{ij}^* = \sum\{|\mathcal{E}_\pi^* : \text{basic path } l_i \overset{\pi}{\Rightarrow} l_j\,|\}$ for $i, j = 0, 1, ..., m$.

SDPs for Quantum Invariants

Theorem

Invariant Generation Problem is equivalent to find complex matrices $O_0, O_1, ..., O_m$ satisfying the constraints:

$$0 \sqsubseteq \sum_j \mathcal{E}_{0j}^*(O_j) + A,$$

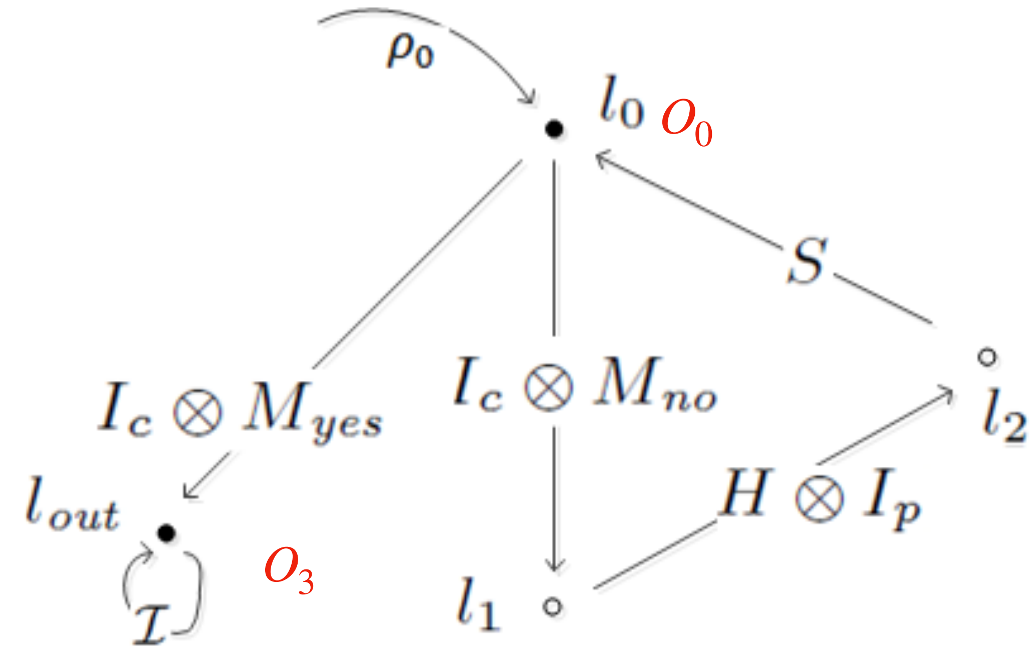
$$0 \sqsubseteq \sum_{j \neq i} \mathcal{E}_{ij}^*(O_j) + (\mathcal{E}_{ii}^* - \mathcal{I})(O_i) + A_i \ (i = 0, 1, ..., m),$$

$$0 \sqsubseteq O_i \sqsubseteq I \ (i = 0, 1, ..., m),$$

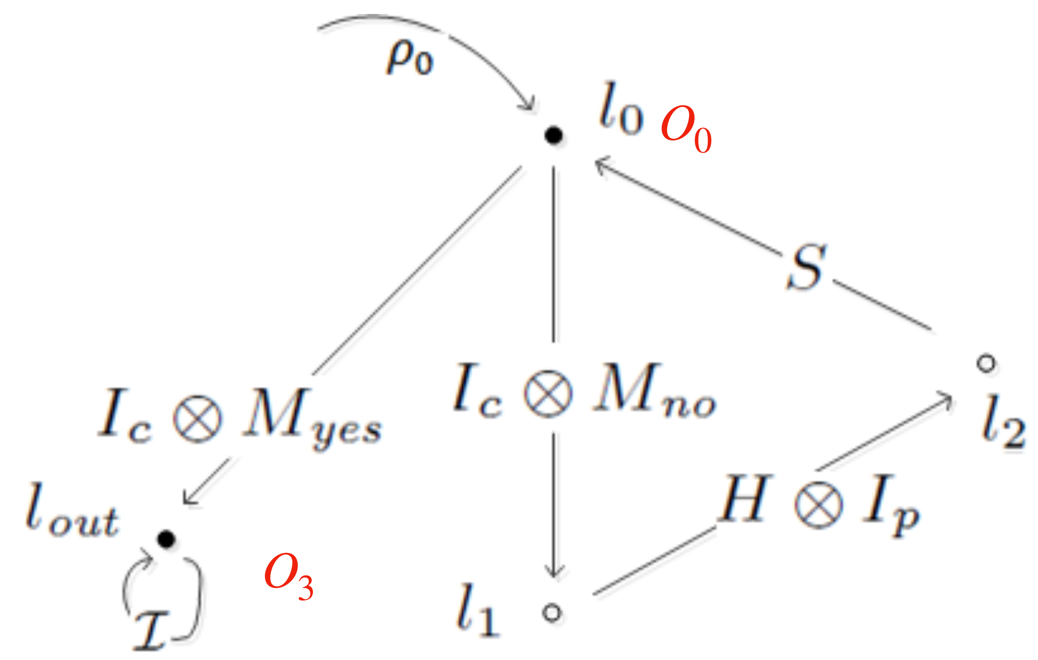
where:

$$\begin{cases} A = I - \sum_j \mathcal{E}_{0j}^*(I) - \Theta, \\ A_i = I - \sum_j \mathcal{E}_{ij}^*(I) \ (i = 0, 1, ..., m). \end{cases}$$

$$
\begin{aligned}
QW \equiv\, & c := |L\rangle; \\
& p := |0\rangle; \\
& \textbf{while } M[p] = no \textbf{ do} \\
& \quad c := H[c]; \\
& \quad c, p := S[c, p] \\
& \textbf{od}
\end{aligned}
$$



$$
\begin{aligned}
QW \equiv\, & c := |L\rangle; \\
& p := |0\rangle; \\
& \textbf{while } M[p] = no \textbf{ do} \\
& \quad c := H[c]; \\
& \quad c, p := S[c, p] \\
& \textbf{od}
\end{aligned}
$$



Invariant SDPs for Quantum 1-D Loop Walk

Choose cut-set $C = \{l_0, l_3\}$ with $l_3 = l_{out}$. $\Theta = I$. Invariants O_0 and O_3 satisfy the following constraints:

$$
0 \sqsubseteq \mathcal{E}^*_{00}(O_0) + \mathcal{E}^*_{03}(O_3) - \Theta, \tag{1}
$$

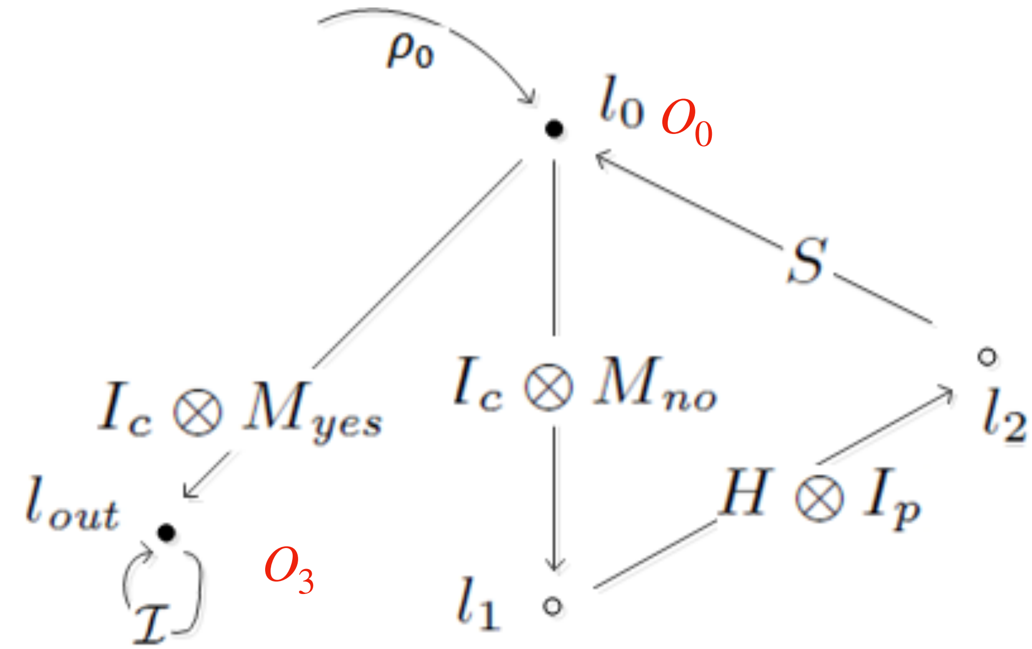
$$
0 \sqsubseteq (\mathcal{E}^*_{00} - \mathcal{I})(O_0) + \mathcal{E}^*_{03}(O_3), \tag{2}
$$

$$
0 \sqsubseteq (\mathcal{E}^*_{33} - \mathcal{I})(O_3) - (I - \mathcal{E}^*_{33}(I)), \tag{3}
$$

$$
0 \sqsubseteq O_0, O_3 \sqsubseteq I \tag{4}
$$

$\mathbb{E}_{00} = E_{00} \circ E^{\dagger}_{00}$, $\mathbb{E}_{03} = E_{03} \circ E^{\dagger}_{03}$, $\mathbb{E}_{33} = \mathcal{I}$,
$E_{00} = S(H \otimes I_p)(I_c \otimes M_{no})$, $E_{03} = I_c \otimes M_{yes}$, and I_c, I_p identities.

$$
\begin{aligned}
QW \equiv\, & c := |L\rangle; \\
& p := |0\rangle; \\
& \textbf{while } M[p] = no \textbf{ do} \\
& \quad c := H[c]; \\
& \quad c, p := S[c, p] \\
& \textbf{od}
\end{aligned}
$$



Invariant SDPs for Quantum 1-D Loop Walk

Choose cut-set $C = \{l_0, l_3\}$ with $l_3 = l_{out}$. $\Theta = I$. Invariants O_0 and O_3 satisfy the following constraints:

$$
\begin{aligned}
0 &\sqsubseteq \mathcal{E}^*_{00}(O_0) + \mathcal{E}^*_{03}(O_3) - \Theta, && (1)\\
0 &\sqsubseteq (\mathcal{E}^*_{00} - \mathcal{I})(O_0) + \mathcal{E}^*_{03}(O_3), && (2)\\
0 &\sqsubseteq (\mathcal{E}^*_{33} - \mathcal{I})(O_3) - (I - \mathcal{E}^*_{33}(I)), && (3)\\
0 &\sqsubseteq O_0, O_3 \sqsubseteq I && (4)
\end{aligned}
$$

$\mathbb{E}_{00} = E_{00} \circ E^\dagger_{00}$, $\mathbb{E}_{03} = E_{03} \circ E^\dagger_{03}$, $\mathbb{E}_{33} = \mathcal{I}$,
$E_{00} = S(H \otimes I_p)(I_c \otimes M_{no})$, $E_{03} = I_c \otimes M_{yes}$, and I_c, I_p identities.

Using SDP Solver

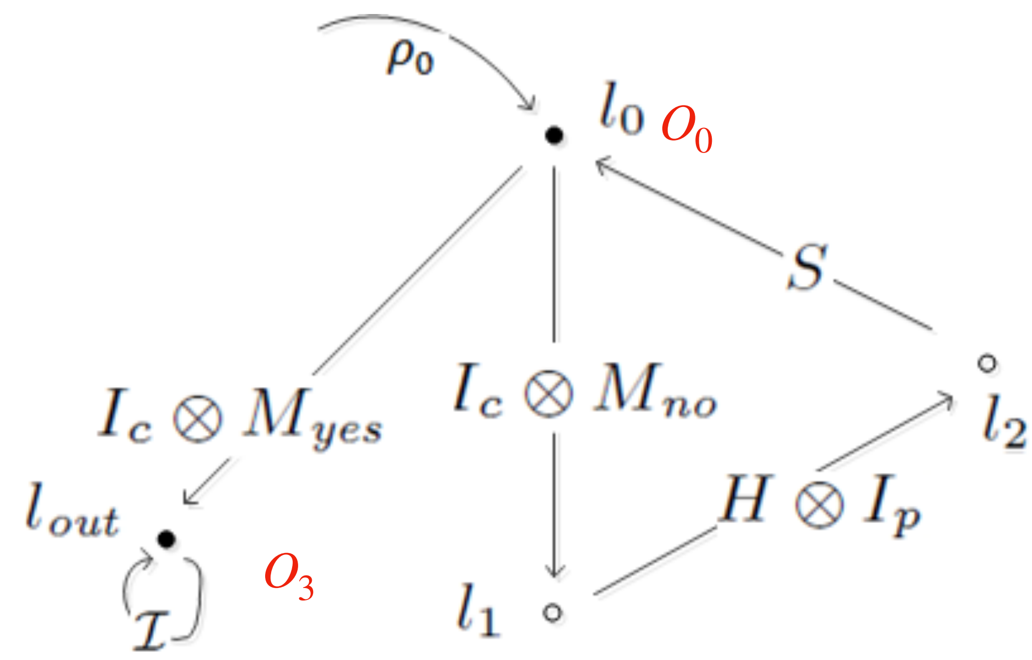
$$O_3 = I_c \otimes |1\rangle\langle 1|$$

$$\Downarrow$$

$$\mathrm{tr}(O_3 \rho_{out}) \geq \mathrm{tr}(\Theta \rho_{in}) = 1$$

Namely, QW always terminates at the position $|1\rangle$ regardless of the input state ρ_0.

$$
\begin{aligned}
QW \equiv\, & c := |L\rangle; \\
& p := |0\rangle; \\
& \textbf{while } M[p] = no \textbf{ do} \\
& \quad c := H[c]; \\
& \quad c, p := S[c, p] \\
& \textbf{od}
\end{aligned}
$$



Invariant SDPs for Quantum 1-D Loop Walk

Choose cut-set $C = \{l_0, l_3\}$ with $l_3 = l_{out}$. $\Theta = I$. Invariants O_0 and O_3 satisfy the following constraints:

$$0 \sqsubseteq \mathcal{E}_{00}^*(O_0) + \mathcal{E}_{03}^*(O_3) - \Theta, \tag{1}$$

$$0 \sqsubseteq (\mathcal{E}_{00}^* - \mathcal{I})(O_0) + \mathcal{E}_{03}^*(O_3), \tag{2}$$

$$0 \sqsubseteq (\mathcal{E}_{33}^* - \mathcal{I})(O_3) - (I - \mathcal{E}_{33}^*(I)), \tag{3}$$

$$0 \sqsubseteq O_0, O_3 \sqsubseteq I \tag{4}$$

$\mathbb{E}_{00} = E_{00} \circ E_{00}^{\dagger}$, $\mathbb{E}_{03} = E_{03} \circ E_{03}^{\dagger}$, $\mathbb{E}_{33} = \mathcal{I}$,
$E_{00} = S(H \otimes I_p)(I_c \otimes M_{no})$, $E_{03} = I_c \otimes M_{yes}$, and I_c, I_p identities.

Using SDP Solver

$$O_3 = I_c \otimes |1\rangle\langle 1|$$

⬇

$$\text{tr}(O_3 \rho_{out}) \geq \text{tr}(\Theta \rho_{in}) = 1$$

Namely, QW always terminates at the position $|1\rangle$ regardless of the input state ρ_0.

Drawback: all these matrices are *exponentially* large.

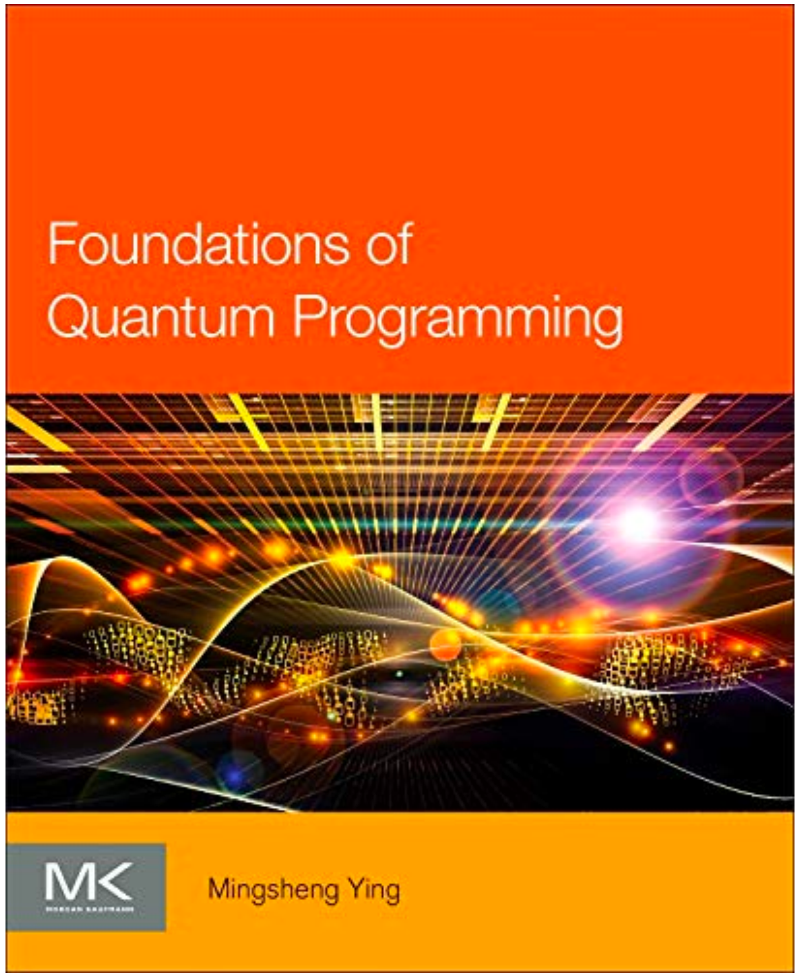
Further Readings: Thank You! Q & A

Applications

- Quantum walk on an n-circle.
- Quantum Metropolis sampling on n-qubits.
- Repeat-Until-Success.
- Quantum Search.
- Quantum Bernoulli Factory.
- Recursively written Quantum Fourier Transformation.

References

- M. S. Ying. Floyd-Hoare Logic for Quantum Programs, *TOPLAS*, 2011.
- M. S. Ying. Foundations of Quantum Programming, Morgan Kaufmann, 2016.
- M. S. Ying, S. G. Ying and X. Wu, Invariants of quantum programs: characterizations and generation, *POPL* 2017.
- Y. Li, and M. S. Ying. Algorithmic Analysis of Termination Problems for Quantum Programs, *POPL*, 2018.
- L. Zhou, N. Yu, and M. S. Ying. An Applied Quantum Hoare Logic, *PLDI*, 2019.
- S. H. Hung, Y. Peng, X. Wang, S. Zhu, and X. Wu. On the Theory and Practice of Invariant-based Verification of Quantum Programs, manuscript, 2020.



Outline

(1) Introduction to Quantum Computing and Potential Roles of Programming Languages (25 min + 5 Q & A)

(2) A Mini-Course of Quantum Hoare Logic on Quantum While Language (30 min + 5 Q & A)

(3) Discussion on existing and potential Programming Language research opportunities (20 min + 5 Q & A)

Reference: tutorial slides and some references are available at https://www.cs.umd.edu/~xwu/mini_lib.html

Summary from Part I

Satisfactory

Quantum PLs — *some*

Software Tool-chain — *a little*

Architecture — *a little*

Security — *a little*

Hardware Design — *almost none*

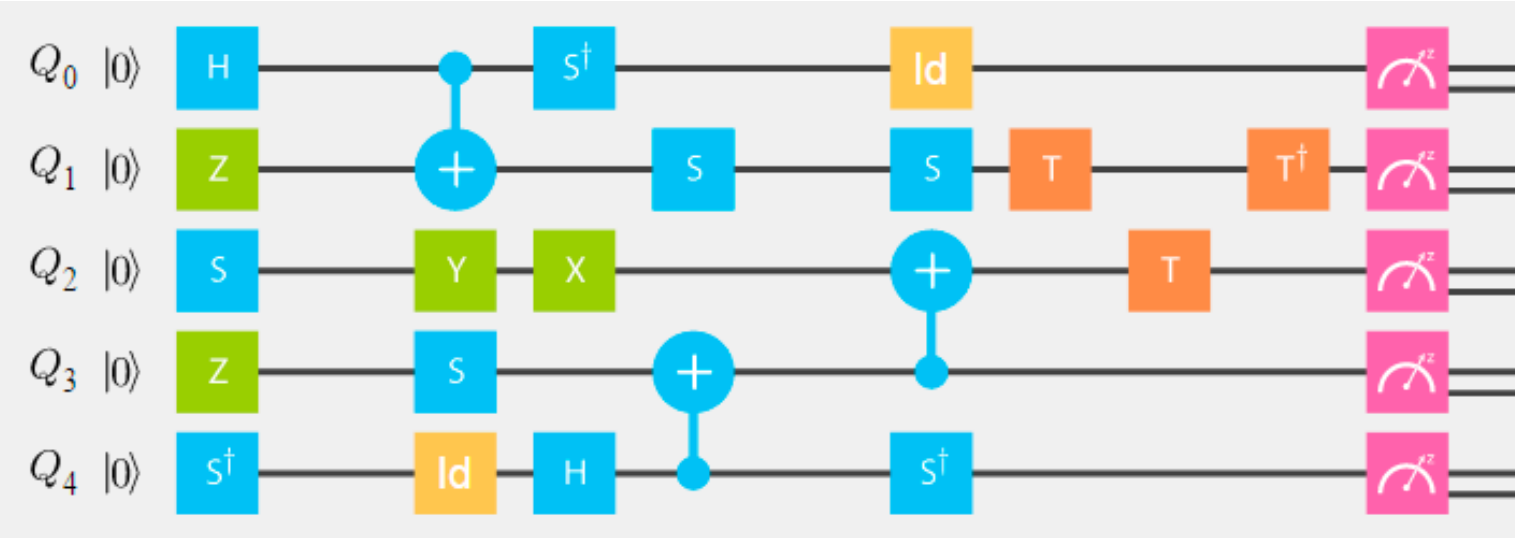
From the implementation perspective

Highlight some concrete problems! (Not a survey)

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



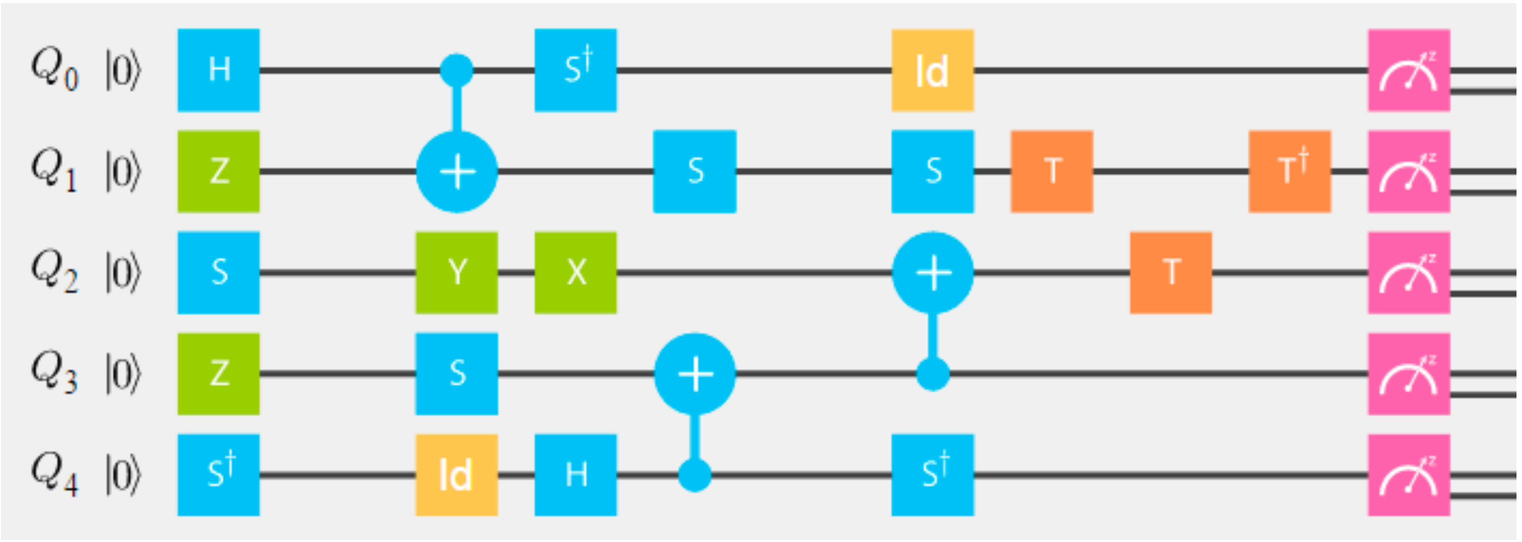
Verifying the circuit
by observation
…. not scalable …

(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

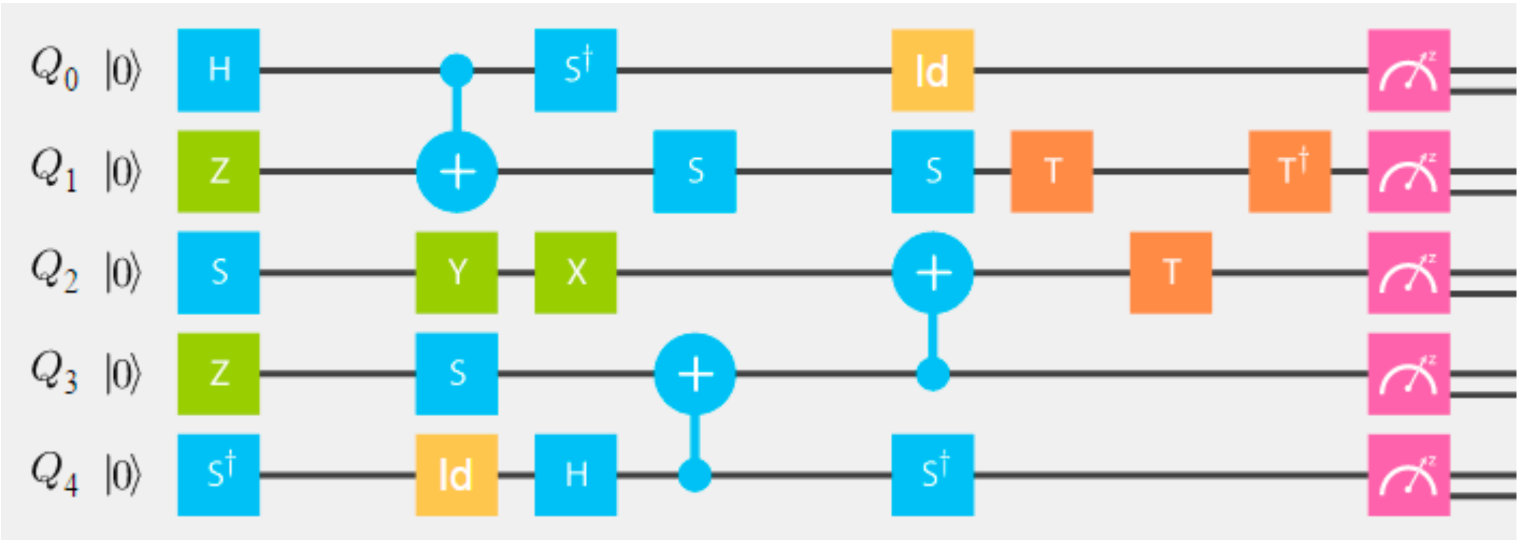
Existing work on type enforced **correctness** in QPLs

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

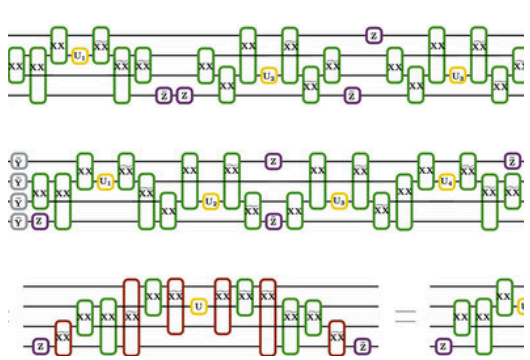
Need to be very resilient to hardware errors

For NISQ machines, all operations could be erroneous

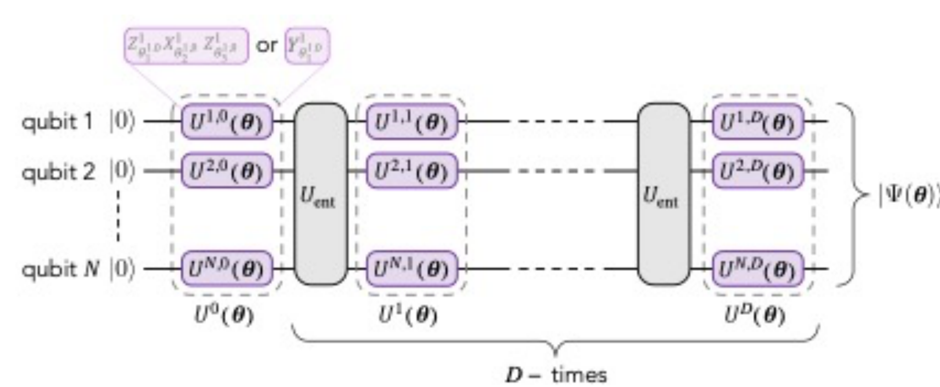
Need also to be scalable

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

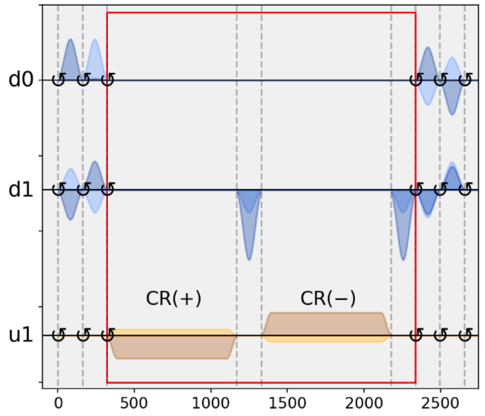
GAP: most of existing tool-chains compile to circuits with non-native gates on the hardware. Lead to very inefficient use of NISQ machines.

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

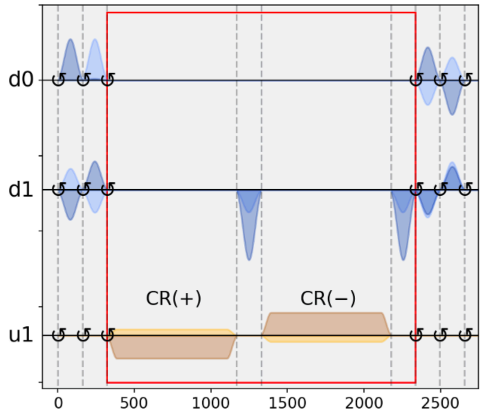


Compilation of Quantum Application: Analog Machines

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**

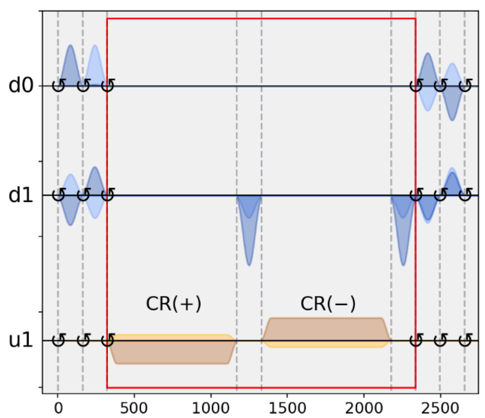
Shi et al. Proceedings of the IEEE, Jun 2020

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?

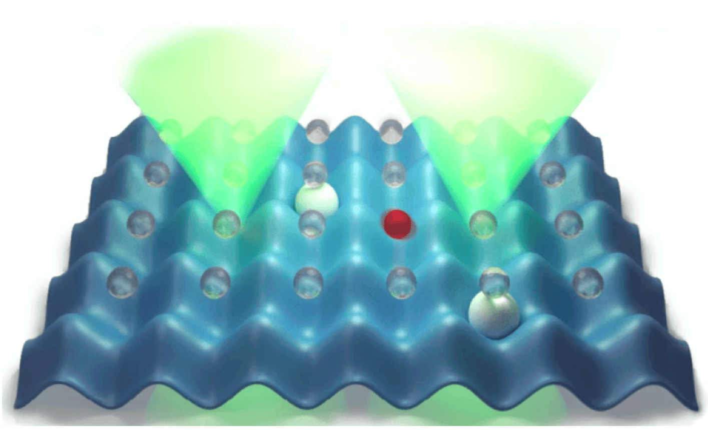
Recent study suggests : compilation to *control pulses, qutrits*, or so



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

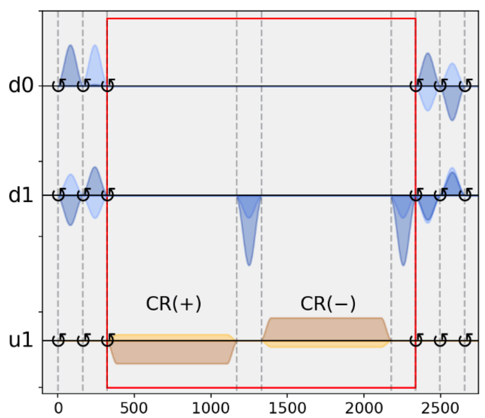
Unexplored yet. But would be of great interests!

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?

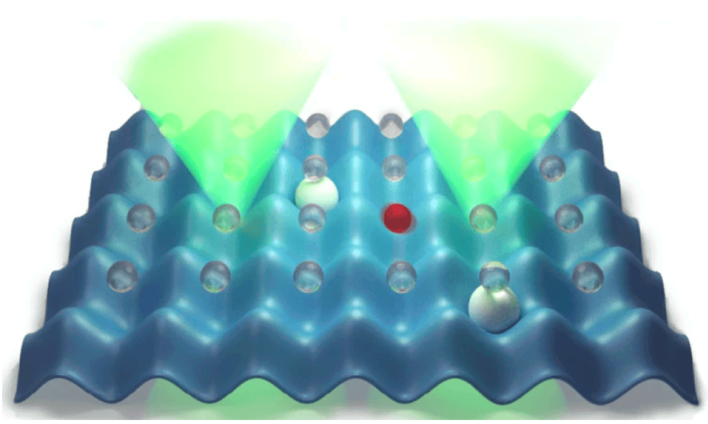
Recent study suggests : compilation to *control pulses, qutrits*, or so



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?



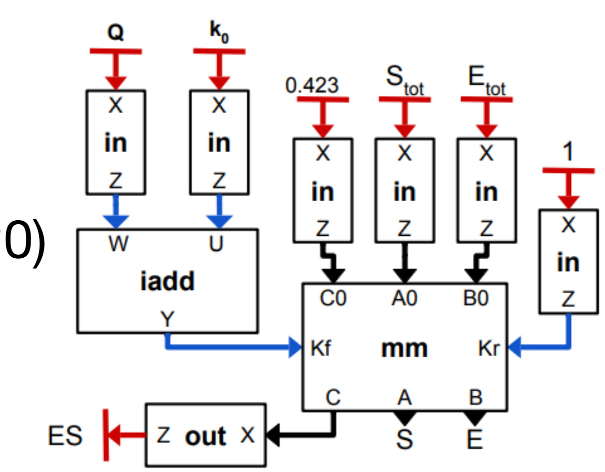
Analog machine modeled after the physics to simulate

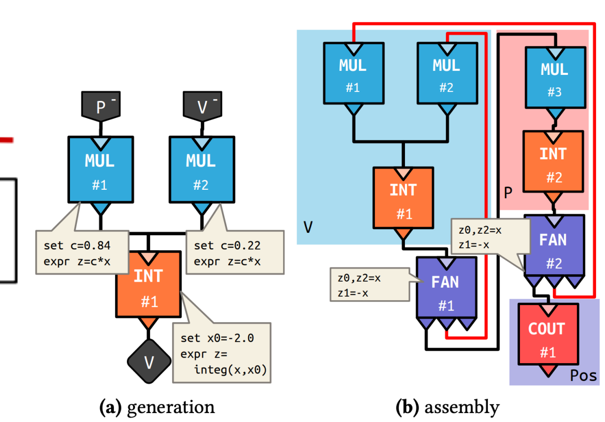
Unexplored yet. But would be of great interests!

Classical Examples:

Achour et al. (PLDI16)

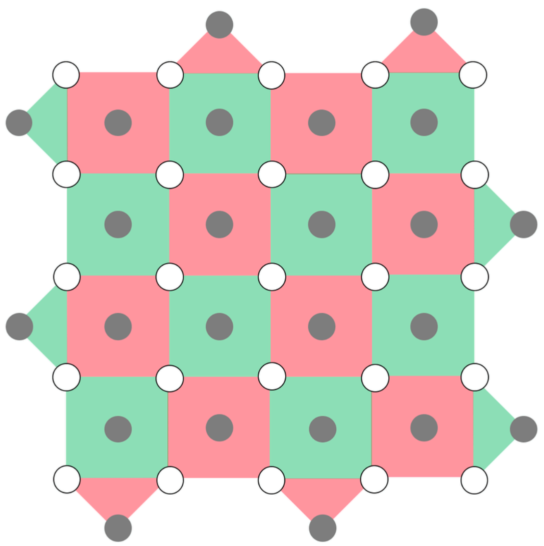
Achour & Rinard (ASPLOS 20)



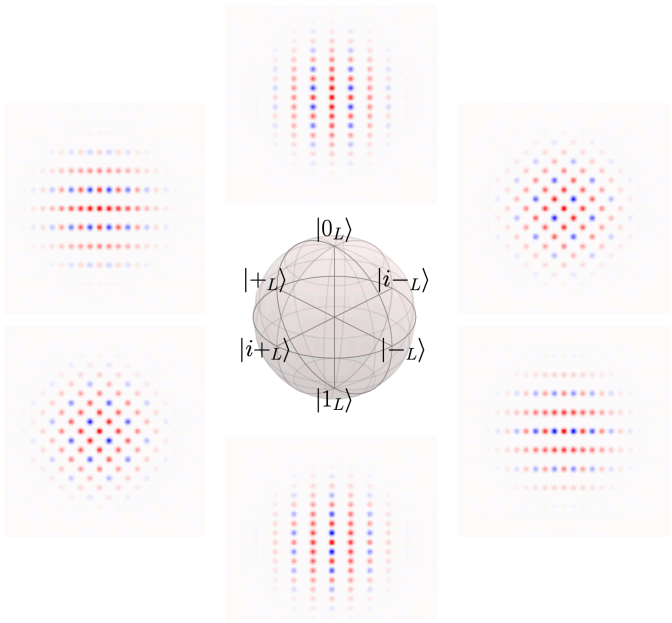


ERROR

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*

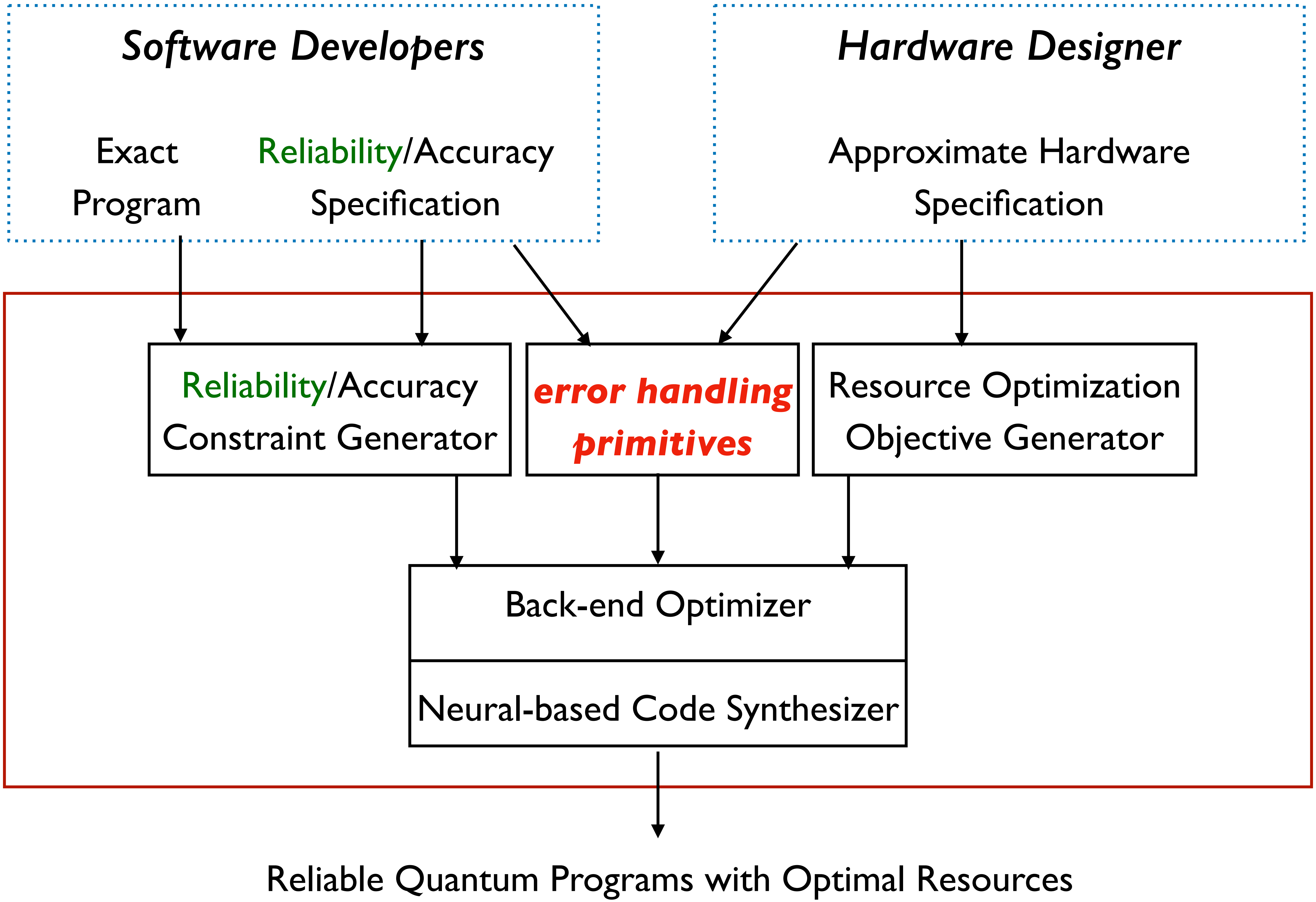
Methodology

- 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* !

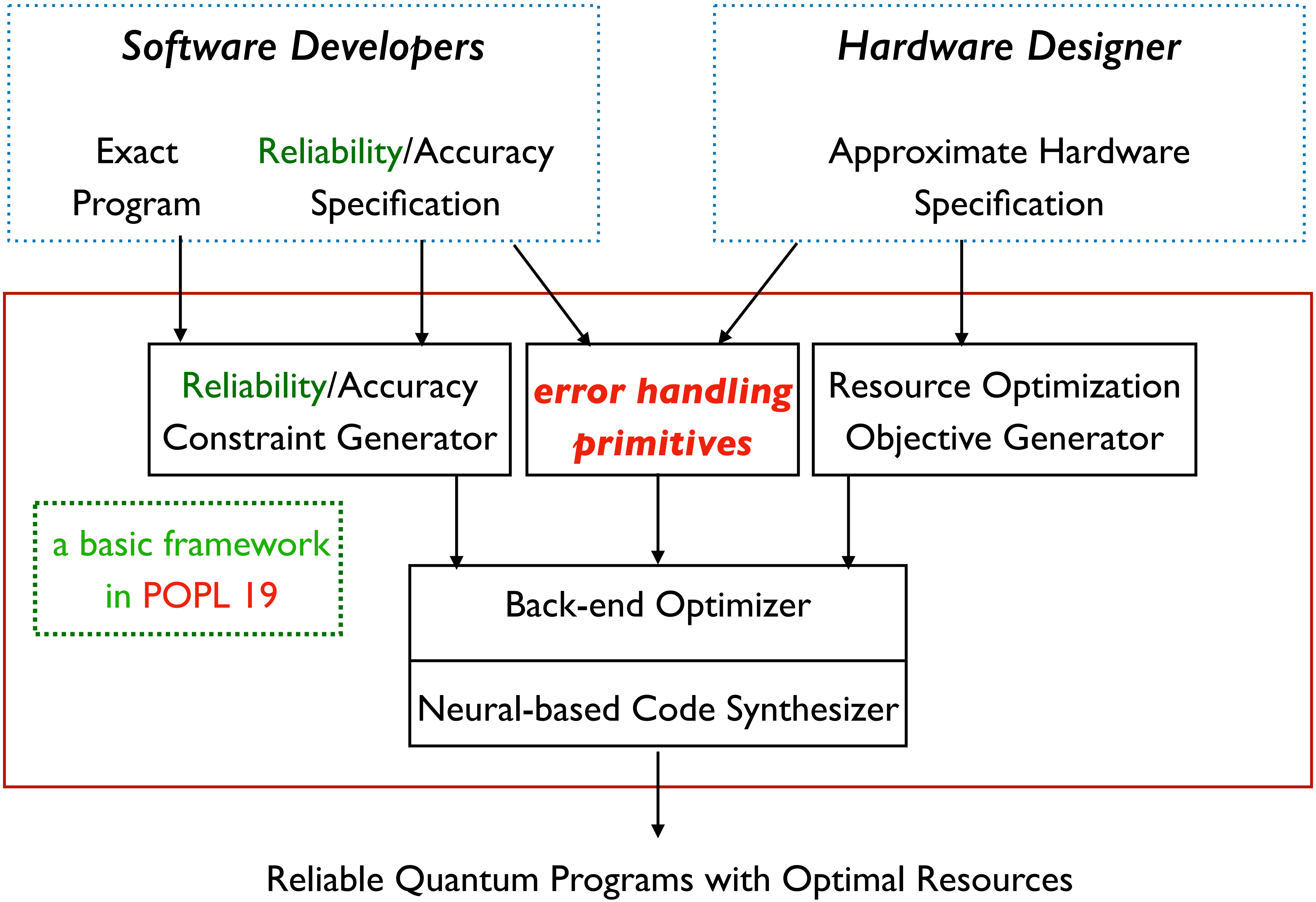
- Return possibly inaccurate/approximate results!
  - *unreliable hardware*
  - *limited computational resource*
- **Good** when *approximate results* are sufficient for applications!
  - *vision, machine learning; also with guarantees for critical data*
- Various techniques developed in classical PL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

Overview

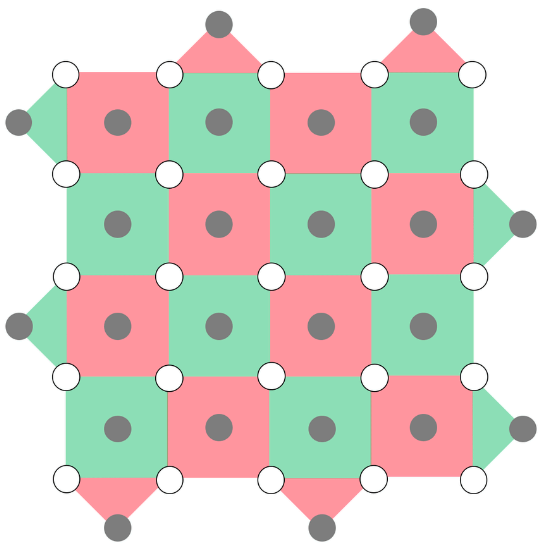


Software Developers
Exact Program
Reliability/Accuracy Specification
Hardware Designer
Approximate Hardware Specification
Reliability/Accuracy Constraint Generator
error handling primitives
Resource Optimization Objective Generator
Back-end Optimizer
Neural-based Code Synthesizer
Reliable Quantum Programs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

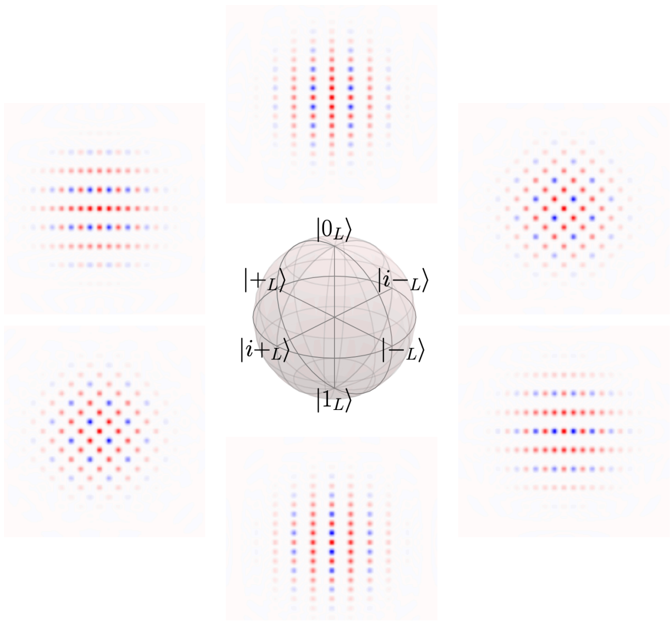
Overview


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



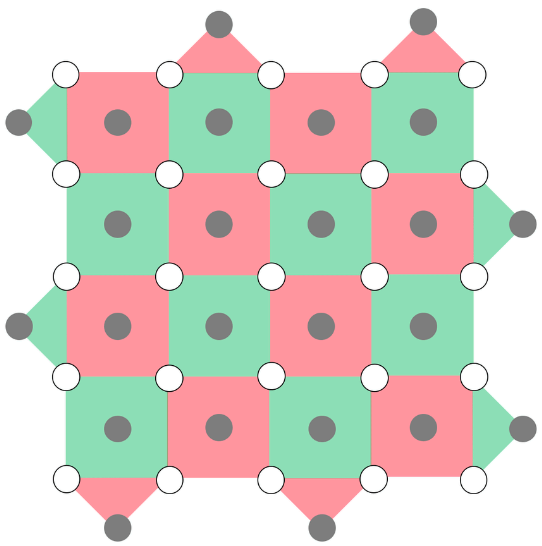
Quantum Error Correction
Fight
Quantum Decoherence



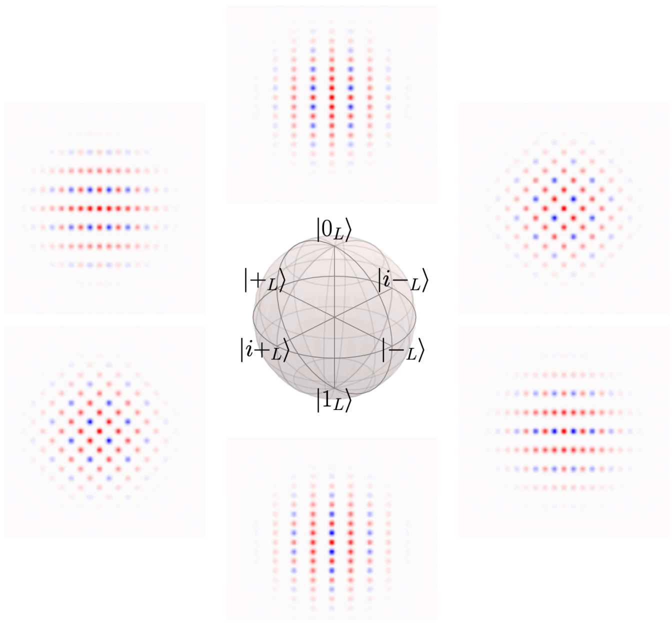
ERROR

Nature

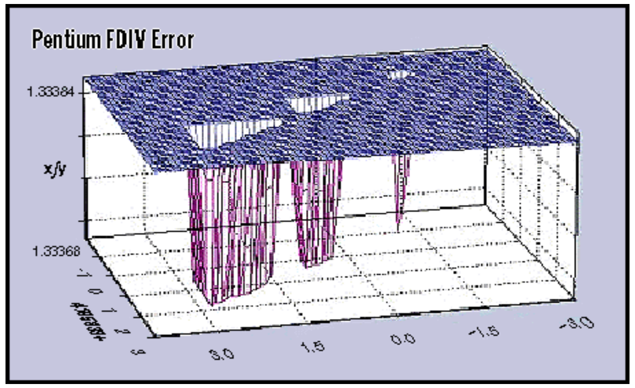
Human



Quantum Error Correction
Fight
Quantum Decoherence



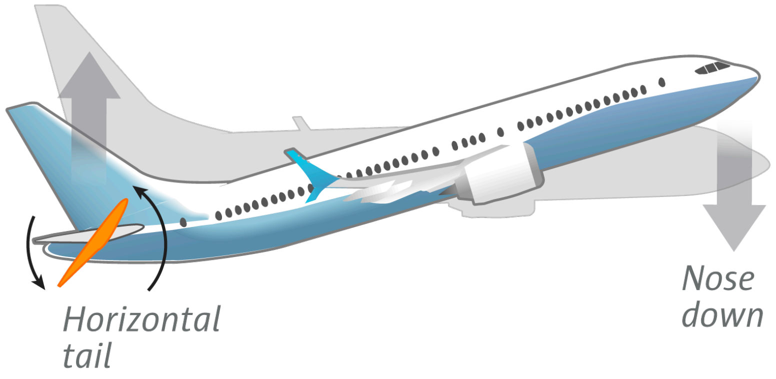
ERROR



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

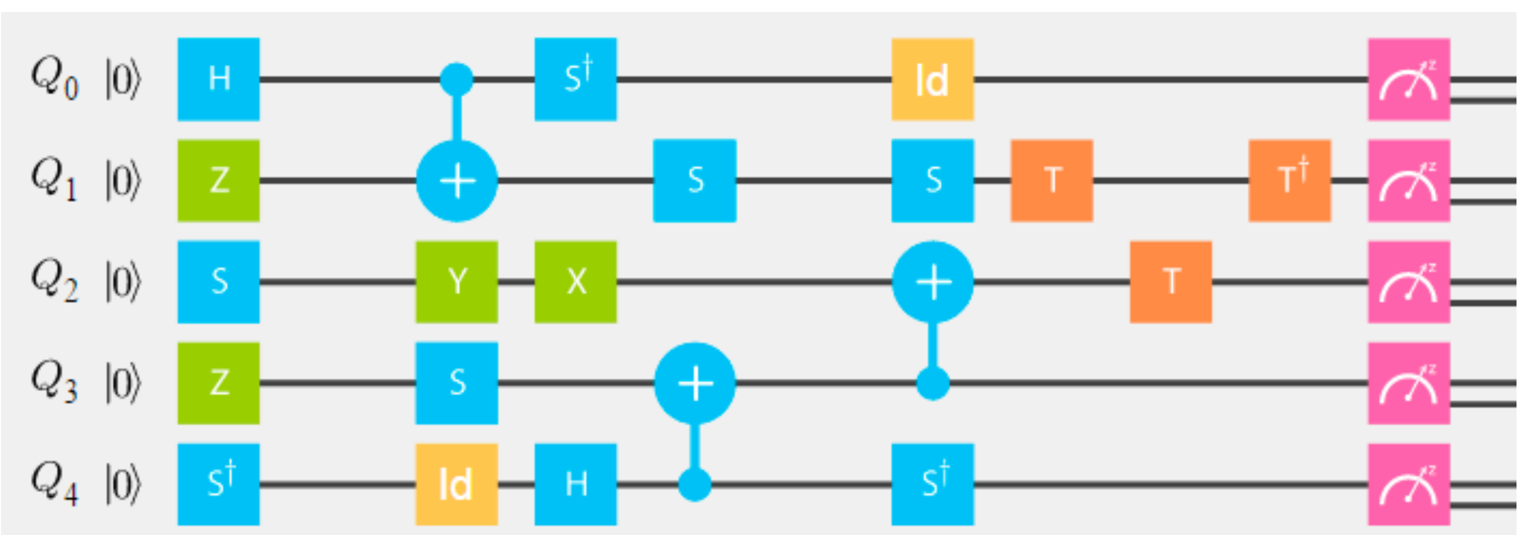
Human Errors in Quantum Software Engineering

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

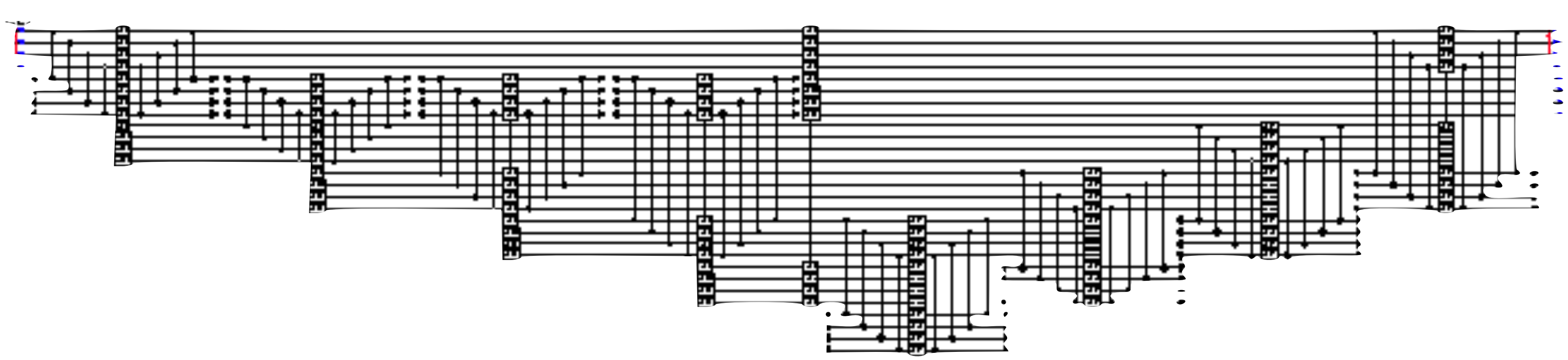
Quantum case : Significantly More **CHALLENGING** than Classical

- 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…



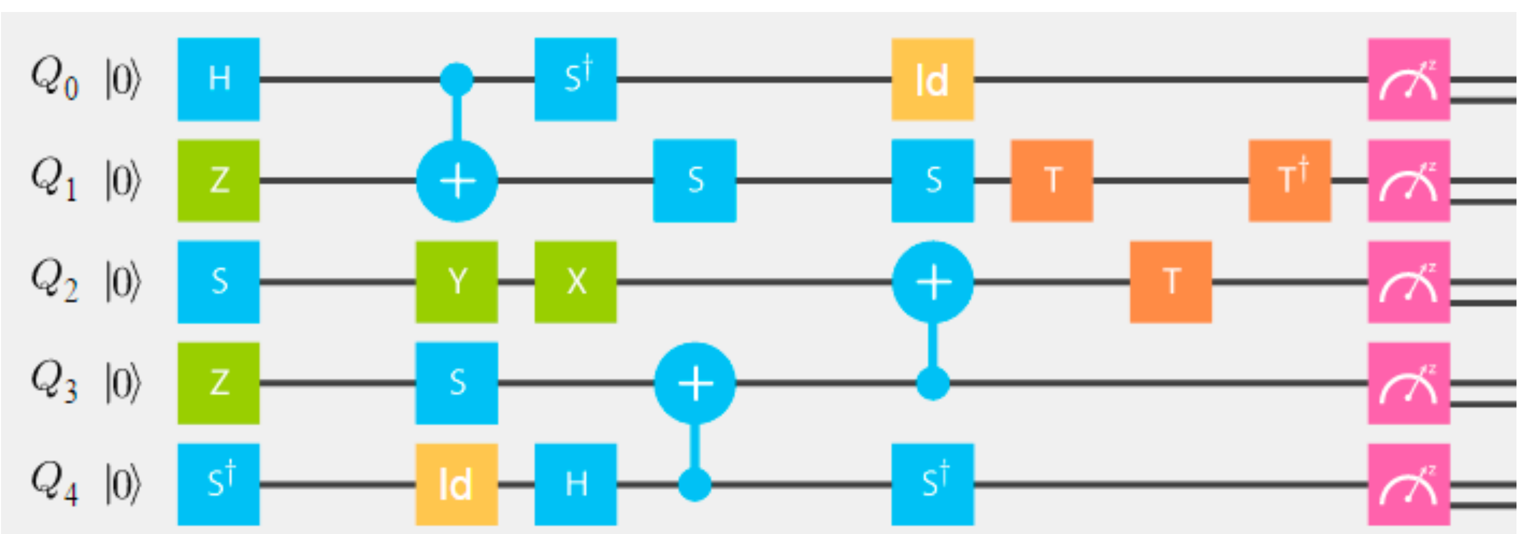
Human Errors in Quantum Software Engineering

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

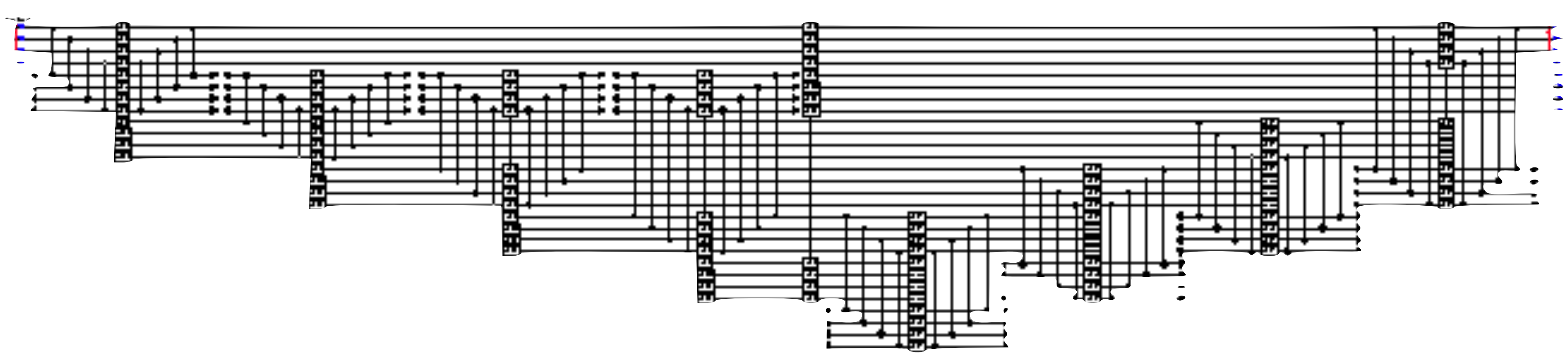
Quantum case : Significantly More **CHALLENGING** than Classical

- 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!

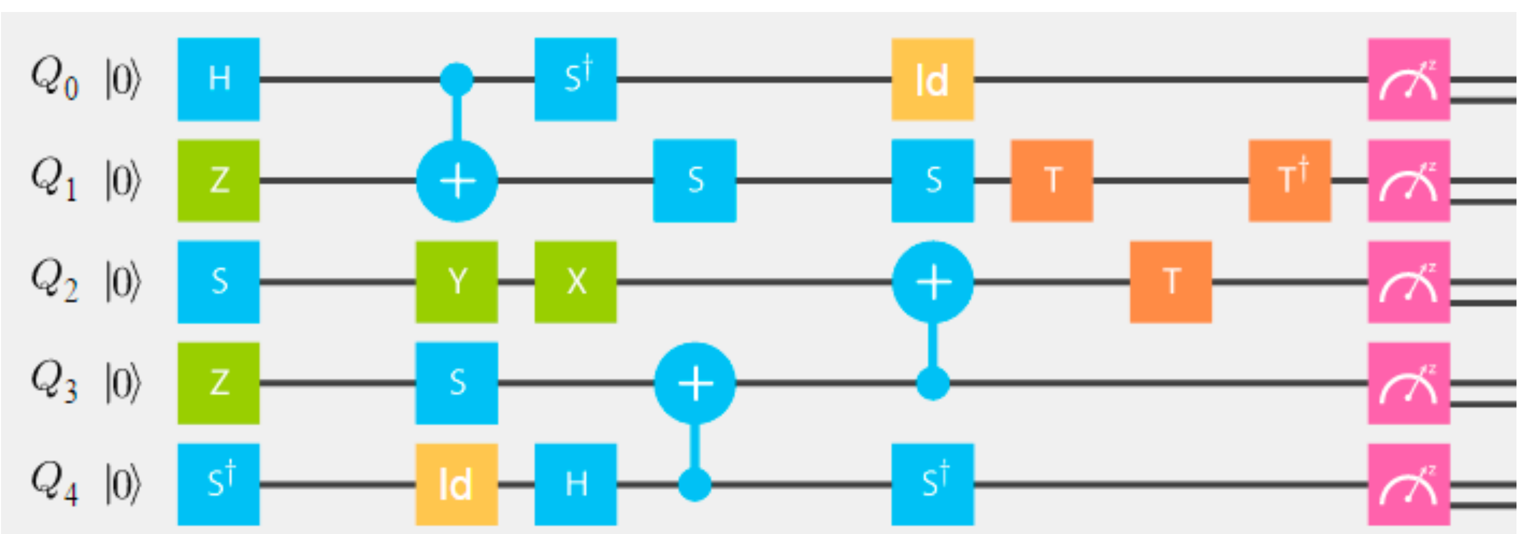
Human Errors in Quantum Software Engineering

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

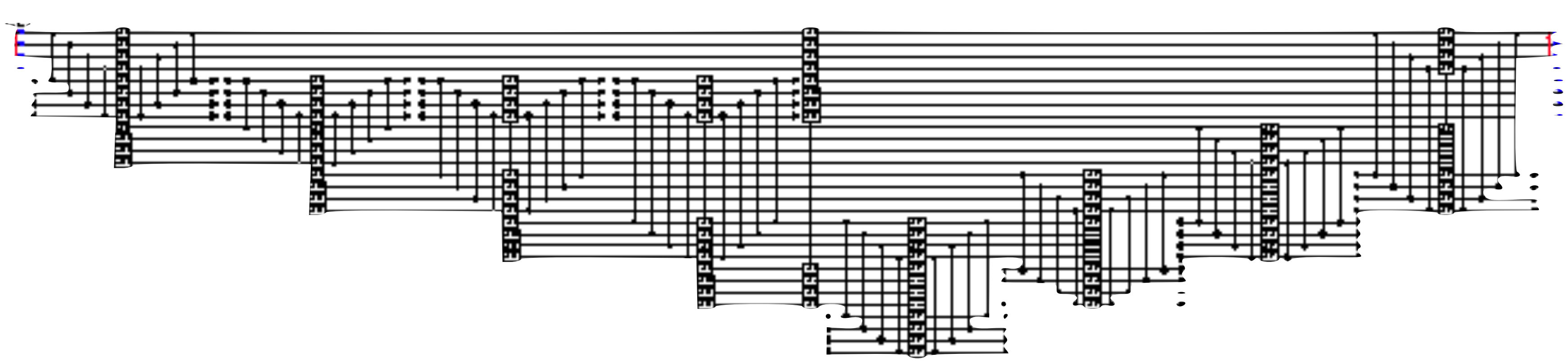
Quantum case : Significantly More **CHALLENGING** than Classical

- 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!

WARNING

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

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

TONY HOARE

Microsoft Research Ltd., Cambridge, UK

Journal of the ACM, Vol 50, 2003

Certified software: a solution to validation of q. software

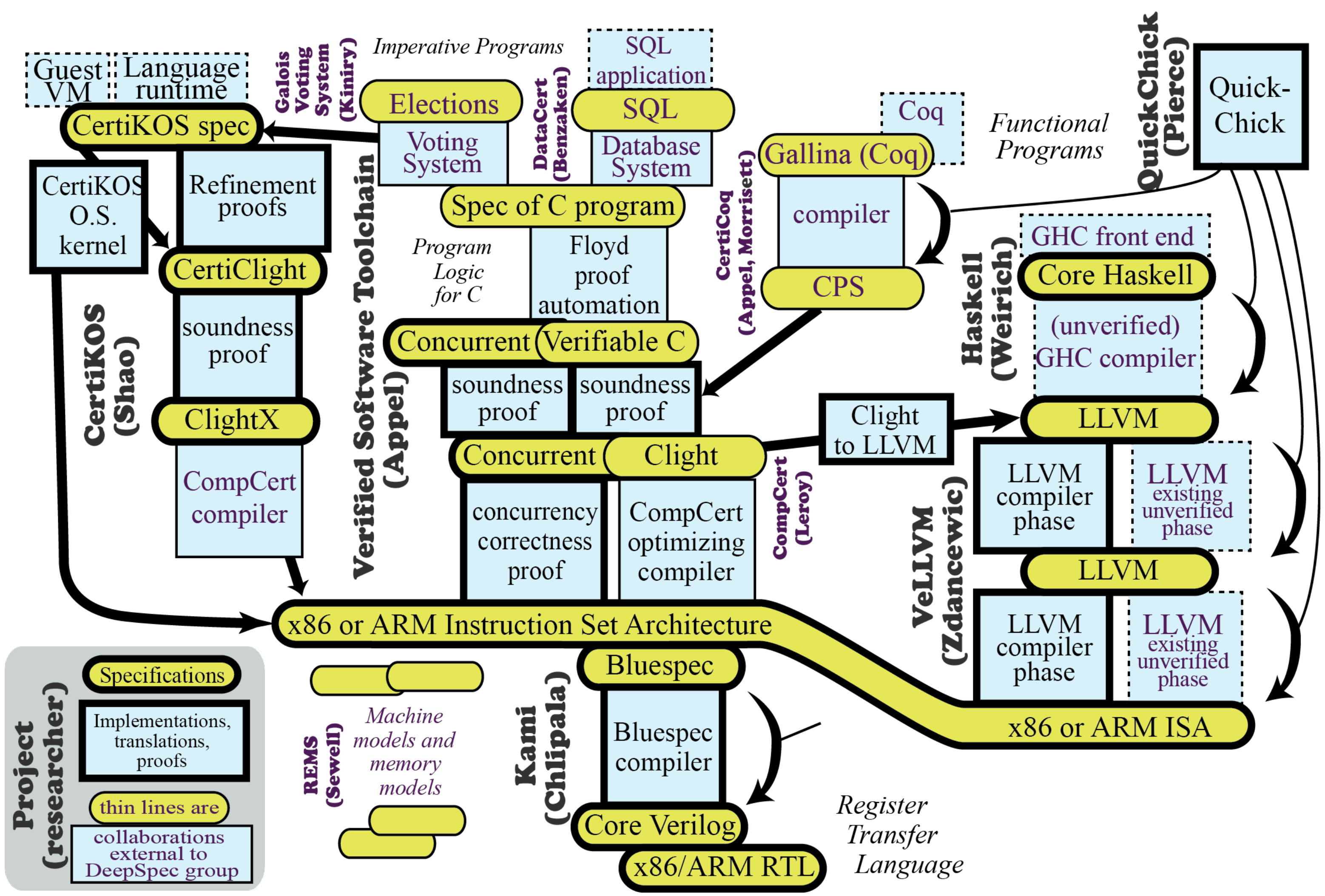
The Verifying Compiler: A Grand Challenge for Computing Research

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



Verified Software Toolchain

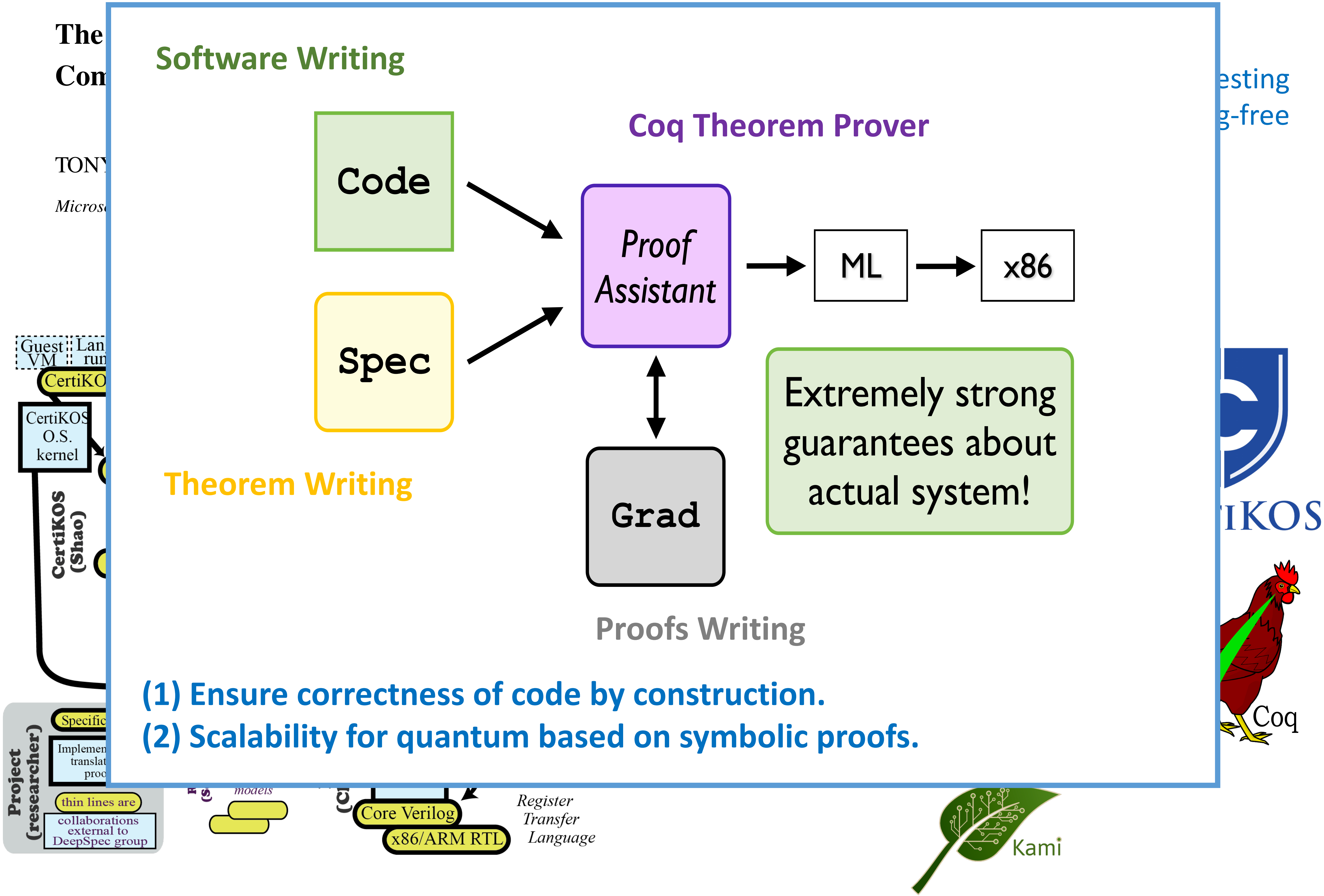
CERTIKOS

Vellvm verified LLVM

Certi Coq

Kami

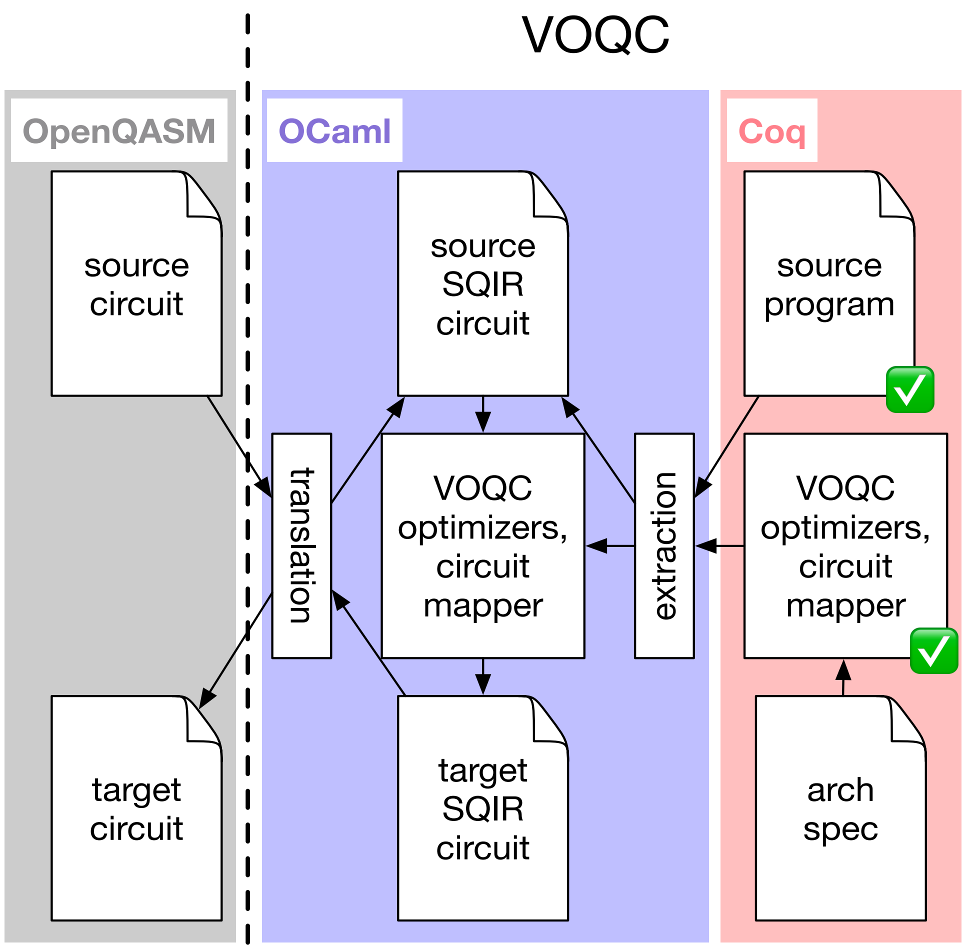
Certified software: a solution to validation of q. software

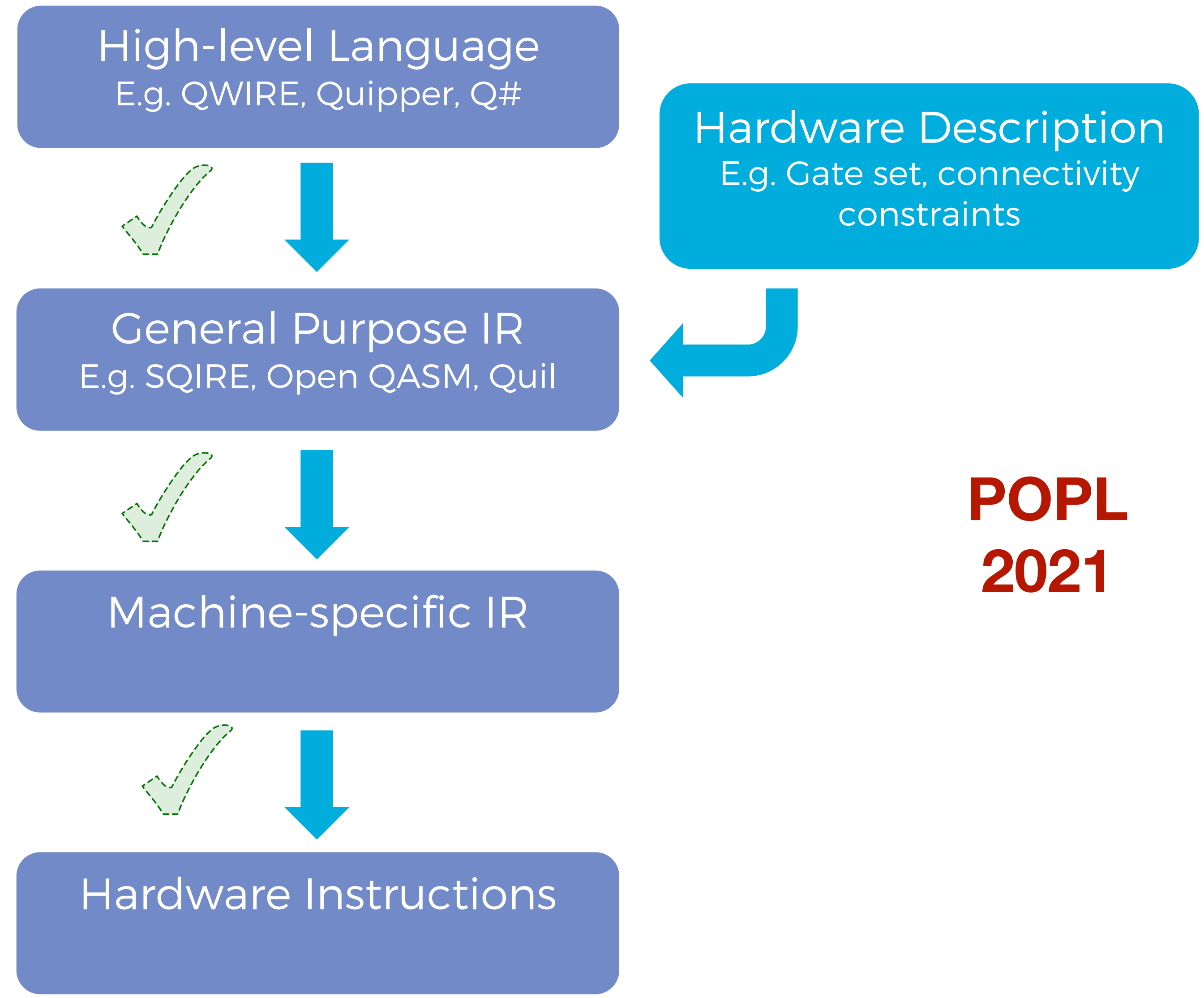


Joint Center for Quantum Information and Computer Science

PLUM

(Verified Optimizer for Quantum Circuits)



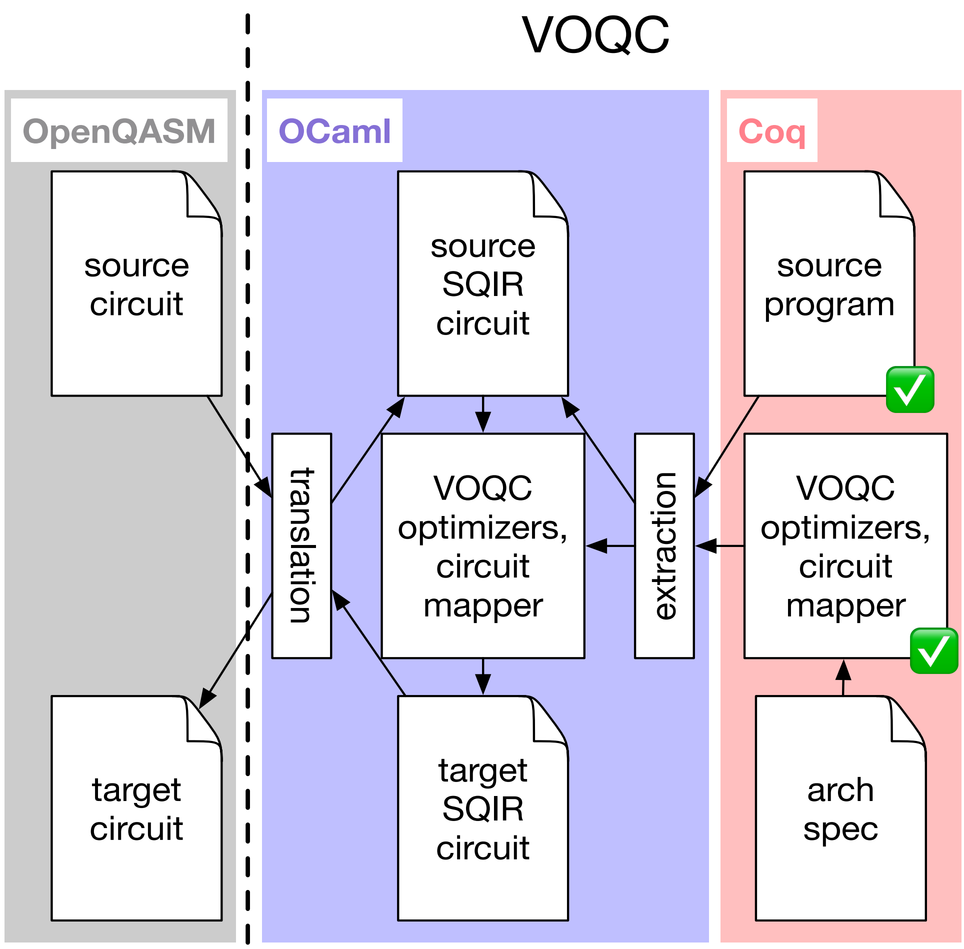


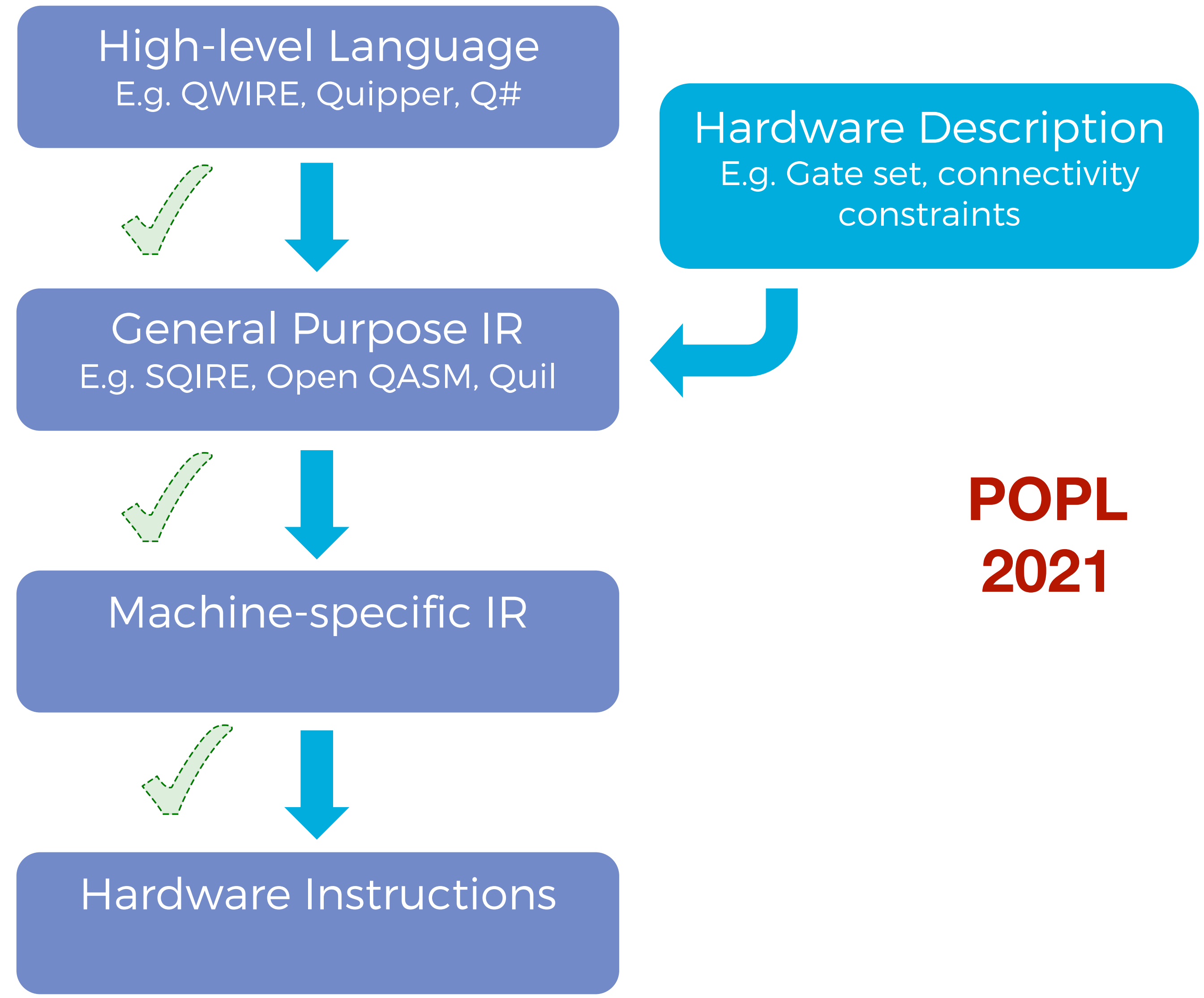
POPL 2021

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

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

(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.

About Today's Tutorial:

Goal: Some Basic Quantum Computing & PL + References

(1) Introduction to Quantum Computing and Potential Roles of Programming Languages (25 min + 5 Q & A)

(2) A Mini-Course of Quantum Hoare Logic on Quantum While Language (30 min + 5 Q & A)

(3) Discussion on existing and potential Programming Language research opportunities (20 min + 5 Q & A)

Reference: tutorial slides and some references are available at https://www.cs.umd.edu/~xwu/mini_lib.html