



HPCA 2025 Tutorial

Janus 3.0: A Software Framework for Analyzing, Optimizing, Verifying, and Implementing Quantum Circuit



Organizers: Jianwei Yin, **Liqiang Lu**, Siwei Tan, Tianyao Chu

College of Computer Science and Technology
Zhejiang University (ZJU)

https://janusq.github.io/HPCA_2025_Tutorial/



Liqiang Lu

liqianglu@zju.edu.cn

Liqiang Lu is a ZJU100 Young Professor in the College of Computer Science, Zhejiang University. His research interests include quantum computing, computer architecture, deep learning accelerator, and software-hardware codesign. He has authored more than 30 scientific publications in premier international journals and conferences in related domains, including ISCA, MICRO, HPCA, ASPLOS, FCCM, DAC, IEEE Micro, and TCAD. He also serves as a TPC member in the premier conferences in the related domain, including DAC, ICCAD, FPT, HPCC, etc.

[HPCA 2025] Debin Xiang, Qifan Jiang, **Liqiang Lu**, et al. “Choco-Q: Commute Hamiltonian-based QAOA for Constrained Binary Optimization”.

[ASPLOS 2024] Siwei Tan, **Liqiang Lu**, Hanyu Zhang, et al. “QuFEM: Fast and Accurate Quantum Readout Calibration Using the Finite Element Method”.

[MICRO 2021] **Liqiang Lu**, Yicheng Jin, Hangrui Bi, et al. “Sanger: A Co-Design Framework for Enabling Sparse Attention using Reconfigurable Architecture”.

[ISCA 2021] **Liqiang Lu**, Naiqing Guan, Yuyue Wang, et al. “TENET: A Framework for Modeling Tensor Dataflow Based on Relation-centric Notation”.



ACES Lab

ADVANCED COMPUTING AND EMERGING SERVICE LAB



ACES-Q Research Group is a part of Advanced Computing and Emerging Service Lab in Zhejiang University led by Prof. Jianwei Yin. Our group focuses on Quantum Computing and consists of 3 faculty members and 16 students from the College of Computer Science and Technique and the School of Software Technology, Zhejiang University.

Milestone of Janus Quantum



JanusQ Software 2.0

- Cluster I/O
- Quantum compilation



Achievements published in top conferences

- MICRO & ASPLOS
- ICCAD quantum chemistry competition

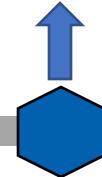


HPCA & Janus 3.0 tutorial

- Constrained binary optimization QAOA
- Second time of Janus tutorial



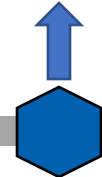
HPCA 2025



2023/02



2022/07



2023/10



2024/04



2025/03



2025/06

Hybrid quantum-classical SAT solver

HPCA 2023

Revealed with
JanusQ cloud
in HPCA 2023



ASPLOS & Janus 2.0 tutorial

- Ultra-fast readout calibration
- First China quantum tutorial



Follow our work accepted by DAC



THE CHIPS
TO SYSTEMS
CONFERENCE

SHAPING THE NEXT GENERATION OF ELECTRONICS

Outline of Presentation



- **Tutorial Overview**
- Background Knowledge
- Mathematical Model of Quantum Computing
- Janus (Taiyuan) Quantum Cloud Platform
- Installing JanusQ

Janus Quantum Cloud



浙江大学
ZHEJIANG UNIVERSITY

Coding interface of JanusQ

A

QuCode

Run Program QFT manipulation Export

Program editor

```
✓ QFT
var sender = qint.new(3, 'S')
var receiver = qint.new(3, 'R')
var ancillary = qint.new(3, 'A')

qc.write(0x0)

let label = 'genData'
qc.startLabel(label)
sender.ry(135, 0x2 | 0x4)
qc.disableDisplay(label)
qc.endLabel(label)

qc.nop()

label = 'InvQFT'
qc.startLabel(label)
sender.invQFT()
qc.endLabel(label)

qc.nop()
label = 'send'
qc.startLabel(label)
sender.exchange(receiver)
qc.endLabel(label)
qc.disableDisplay(label)
qc.nop()

label = 'QFT'
qc.startLabel(label)
receiver.QFT()
```

A-1

Circuit reuse

A-2

Self-defined Gate

A-3

Console

```
Freq 1 = 7
Freq 2 = 7
Freq 3 = 0
Freq 4 = 7
```

B

Quantum circuit view

C

Evolution view

D

Variable State

Probability Magnitude Entanglement

C-1

C-2

D-1

D-2

D-3

D-4

State view

The interface includes a QuCode editor with quantum circuit code, a Quantum circuit view showing a multi-qubit circuit with various gates like H, S, and CNOT, an Evolution view showing state transitions over time, a Variable State view showing probability and magnitude distributions for qubits S, R, and A, and a State view showing the overall state of the system.

Execution modes

switching mode

Choose mode

- Quantum cluster Quantum computer python simulator
 JavaScript simulator analysis

cancel confirm

Provided quantum processors

Resources

计算机名字 N36U19_0

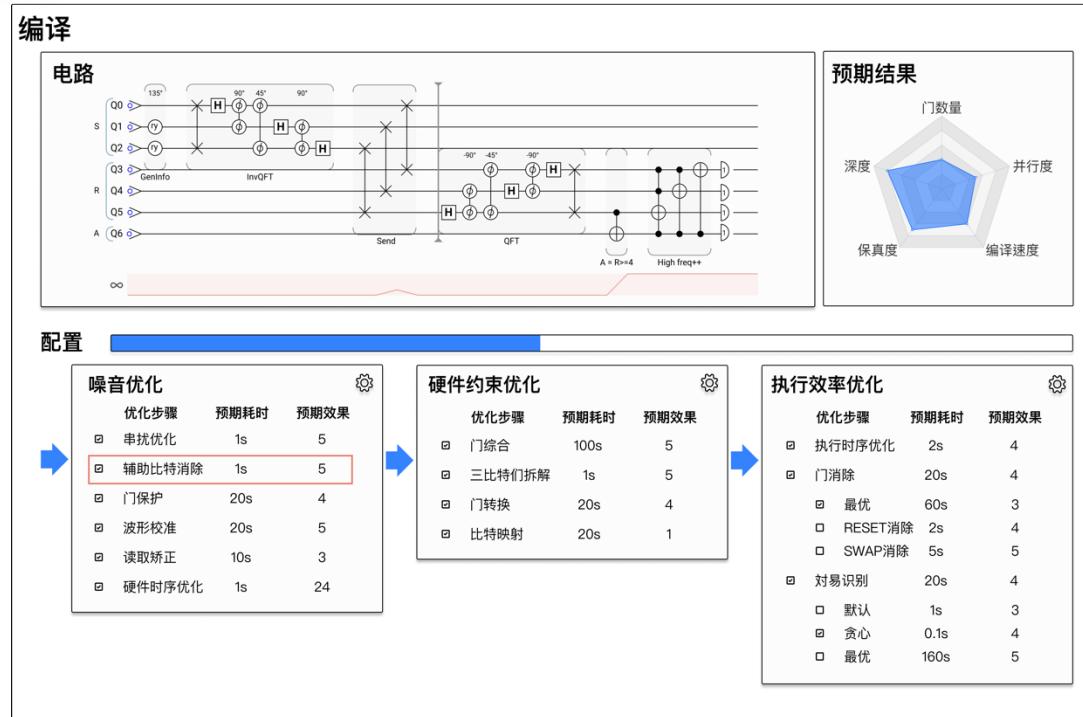
ONLINE STATUS 10 QUBITS

计算机名字 N36U19_1

ONLINE STATUS 13 QUBITS

The Resources section displays two quantum processor units, N36U19_0 and N36U19_1, both currently online. N36U19_0 has 10 qubits, and N36U19_1 has 13 qubits, represented by a grid of blue circles.

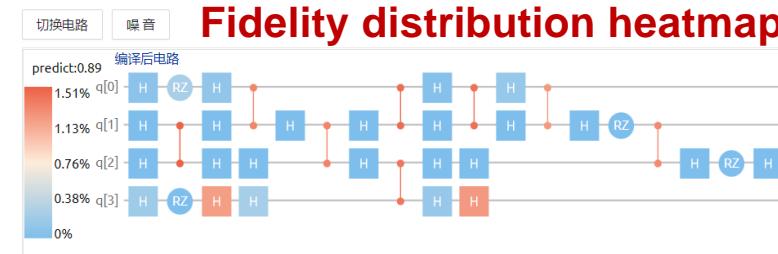
Quantum computing compilation optimization



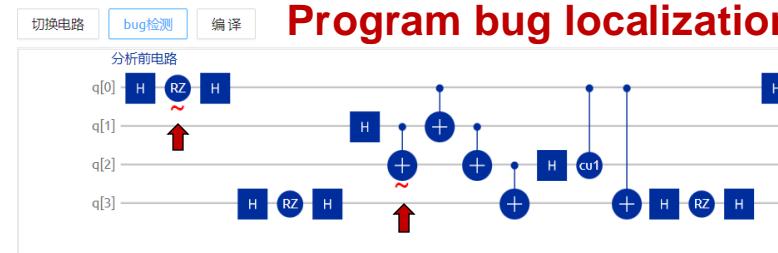
Support selection of compilation optimization options

1. Noise optimization, reducing computation error rates
2. Quantum gate optimization, reducing the number of quantum gates
3. Compilation speed, reducing compilation time

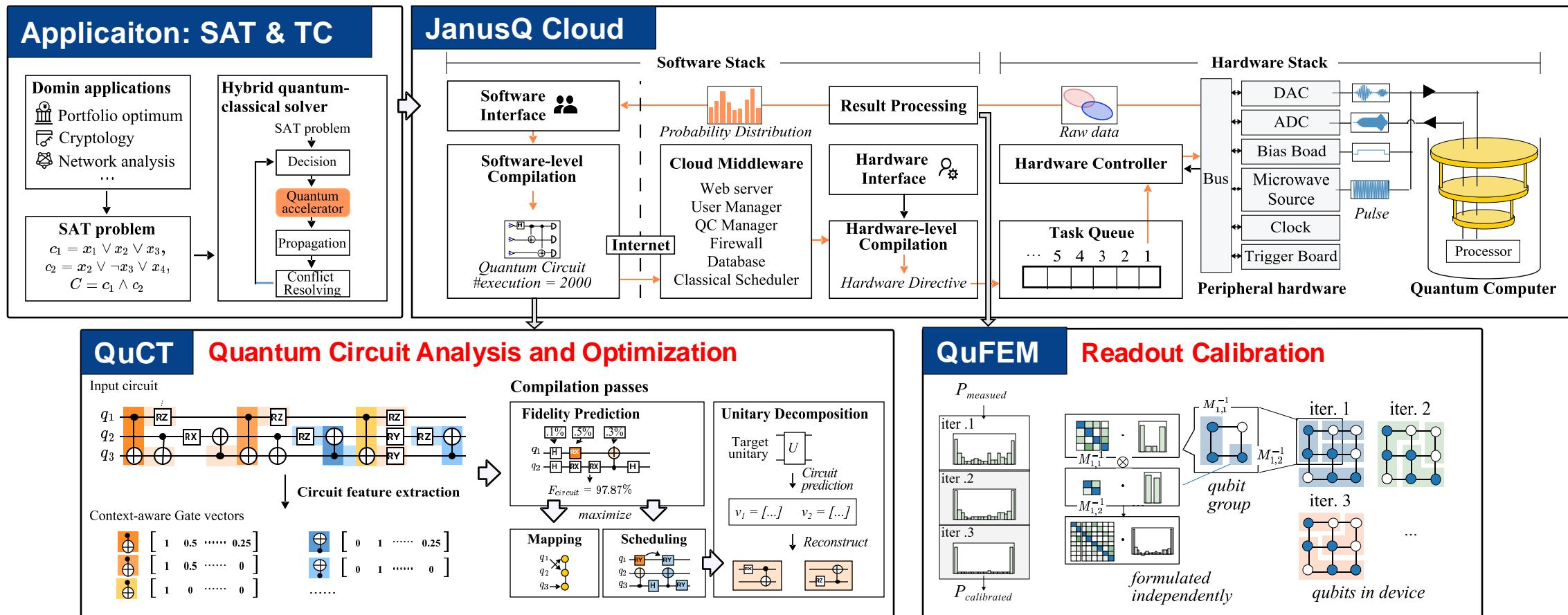
Fidelity prediction and optimization



Program verification and correction

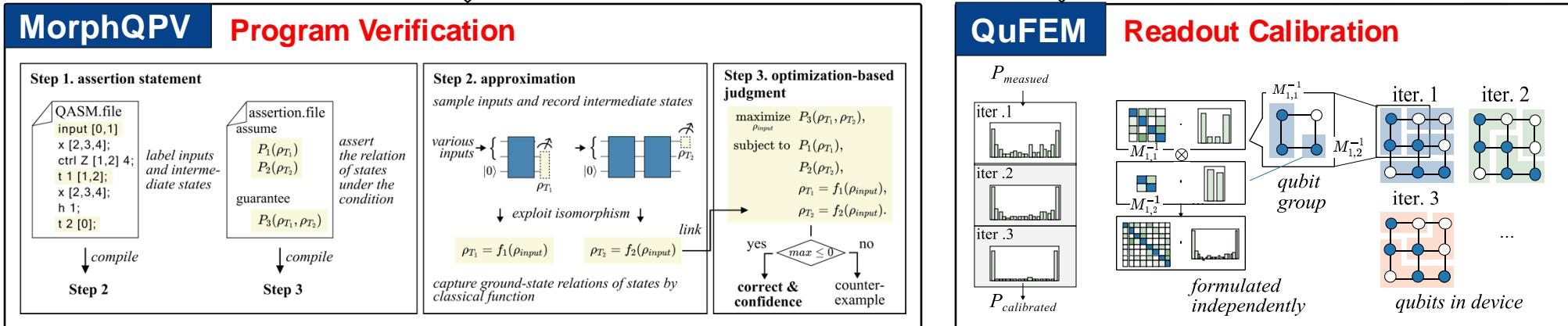
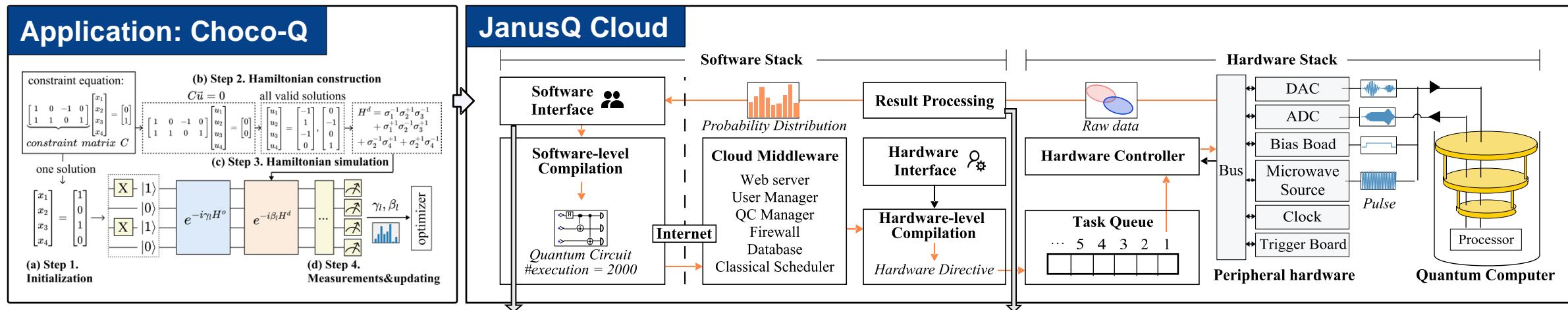


Janus 2.0: A Software Framework for Analyzing, Optimizing and Implementing Quantum Circuit





Janus 3.0: A Software Framework for Analyzing, Optimizing, Verifying, and Implementing Quantum Circuit



Organizers



浙江大學
ZHEJIANG UNIVERSITY



Jianwei Yin
Professor

zjuyjw@cs.zju.edu.cn



Liqiang Lu
Assistant professor

liqianglu@zju.edu.cn



Siwei Tan
Assistant professor

siweitan@zju.edu.cn



Tianyao Chu
Ph.D. Student

tianyao_chu@zju.edu.cn

Outline

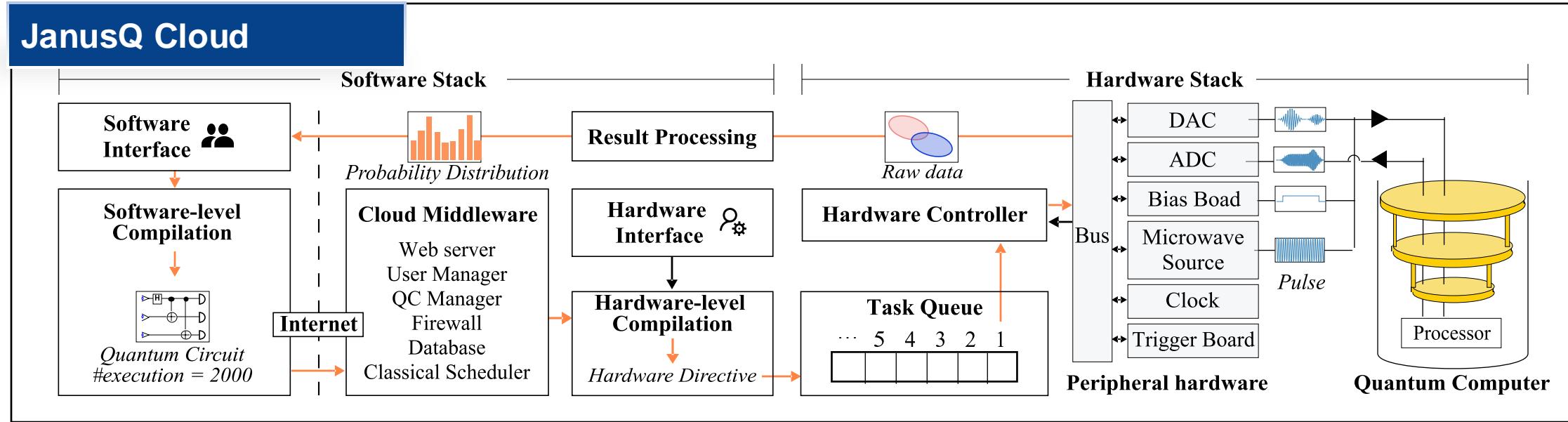


Topic	Presenter	Time
Topic-1. Introduction of Janus quantum cloud platform	Liqiang Lu	1:00pm – 1:30pm
Topic-2. QuCT Details (a) Vectorization model and code examples (b) Fidelity optimization & Unitary decomposition using gate vectors (c) Extending the framework by yourself: other downstream tasks!	Tianyao Chu	1:30pm – 2:15pm
Topic-3. MorphQPV Details (a) MorphQPV overview (b) Assertion statement (c) Implementation details of the automatic verification and repair	Siwei Tan	2:15pm – 3:00pm
Afternoon Break		3:00pm – 3:30pm

Outline



Topic	Presenter	Time
Topic-4. QuFEM Details (a) Characterization of readout error (b) Readout calibration using QuFEM	Kaiwen Zhou	3:30pm – 4:00pm
Topic-5. Choco-Q Details (a) Introduction of constained binary optimization problem (b) Choco-Q overview (c) Solve real-world problems with Choco-Q	Liqiang Lu	4:00pm – 4:45pm
Topic-6. Q & A	All	4:45pm – 5:00pm
Total		1:00pm – 5:00pm

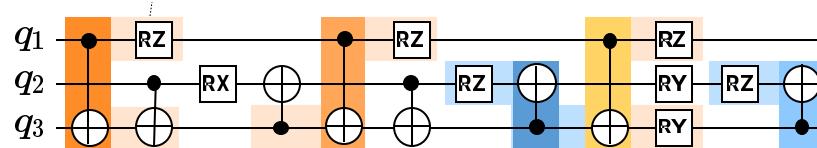


Janus Quantum Infrastructure:

- How do you use the code editor on the cloud platform?
- How do we submit the task to quantum hardware via API?
- How do we take advantage of JanusQ architecture?

QuCT C: Contextual, T: Topological

Input circuit

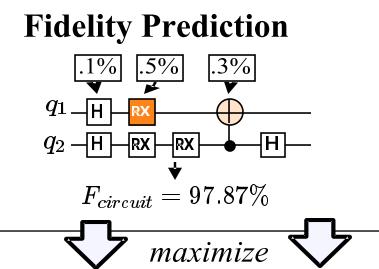


Circuit feature extraction

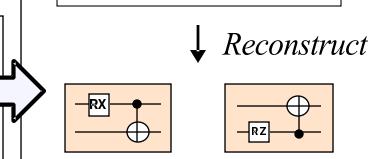
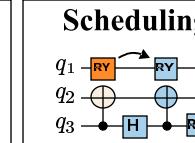
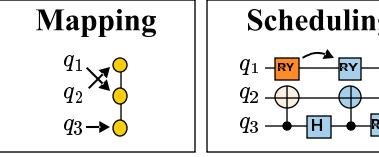
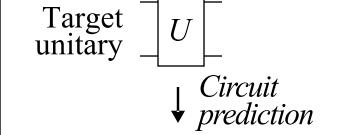
Context-aware Gate vectors

	$\begin{bmatrix} 1 & 0.5 & \dots & 0.25 \end{bmatrix}$		$\begin{bmatrix} 0 & 1 & \dots & 0.25 \end{bmatrix}$
	$\begin{bmatrix} 1 & 0.5 & \dots & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 1 & \dots & 0 \end{bmatrix}$
	$\begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}$	

Compilation passes



Unitary Decomposition



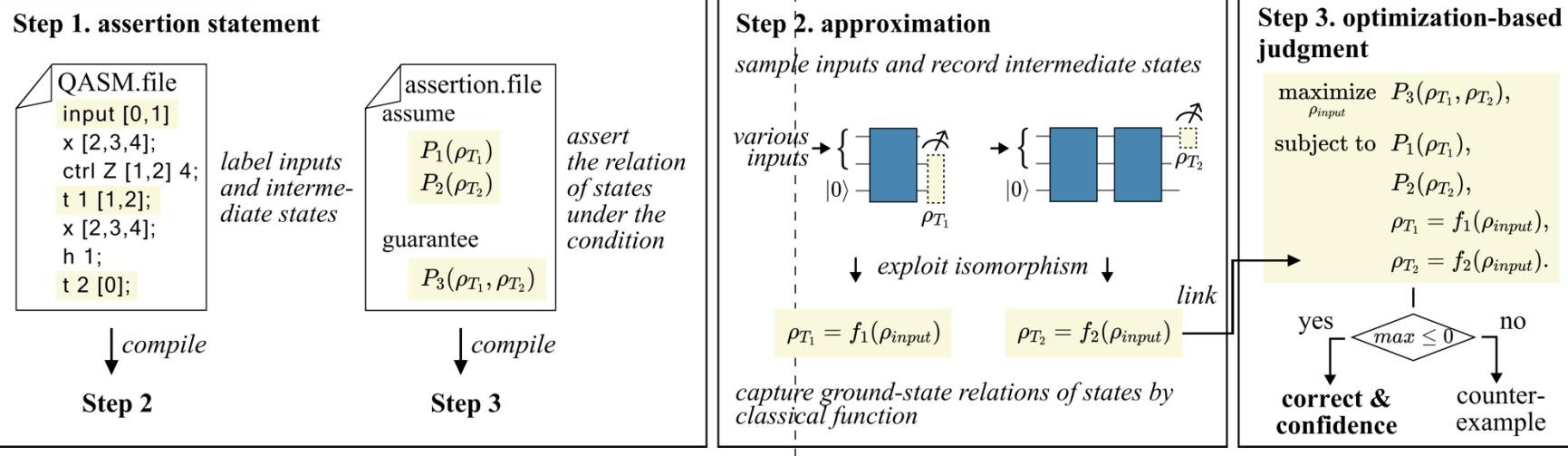
QuCT: Topology-aware quantum circuit optimizer (MICRO 2023)

- Why is topological information important in circuit analysis and optimization?
- How do we use topological information to analyze and optimize noise?
- How can we speed up the unitary decomposition with an upstream model?

Related paper: QuCT: A Framework for Analyzing Quantum Circuit by Extracting Contextual and Topological Features. [MICRO 2023]



MorphQPV

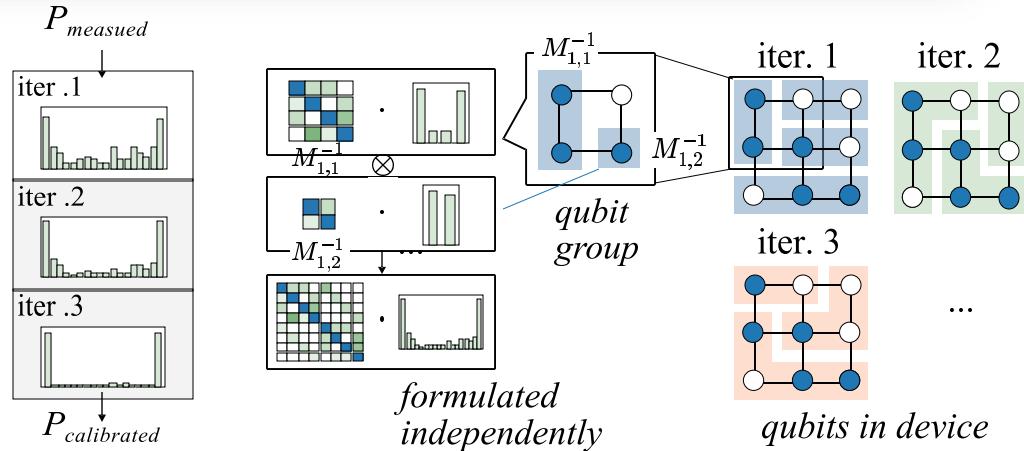


MorphQPV: Isomorphism-based quantum program verification (ASPLOS 2024)

- What is program verification in quantum circuit analysis? Why is it difficult?
- How do we use isomorphism relationship to verify quantum programs?
- How do we realize the assertion statement in quantum programs?

Related paper: MorphQPV: Exploiting Isomorphism in Quantum Programs to Facilitate Confident Verification. [ASPLOS 2024]

QuFEM FEM: Finite Element Method

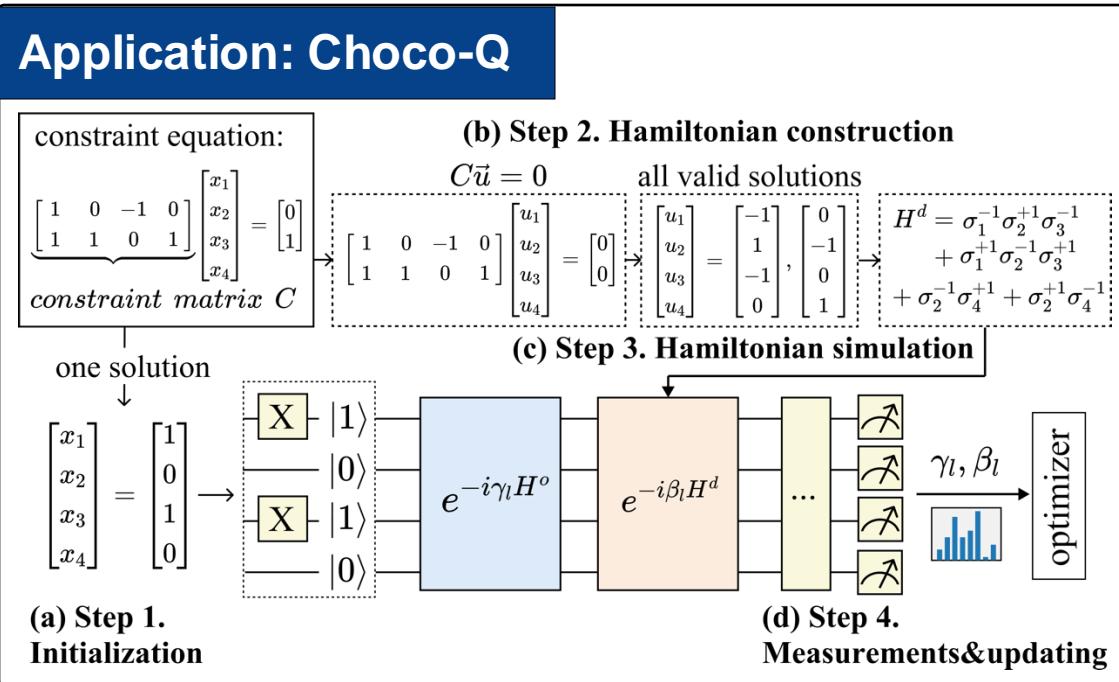


QuFEM: Readout Calibration based on finite element method (ASPLOS 2024)

- What is readout error and the difficulty of calibrating it?
- What is the finite element method?
- How do we mitigate the readout noise using the finite element method?

Related papers: QuFEM: Fast and Accurate Quantum Readout Calibration Using the Finite Element Method. [ASPLOS 2024]

Application: Choco-Q



Choco-Q: Constrained Binary Optimization (HPCA 2025)

- What is constrained binary optimization?
- How do we use commute Hamiltonian to squeeze the searching space of QAOA?

Related papers:

Choco-Q: Commute Hamiltonian-based QAOA for Constrained Binary Optimization. [HPCA 2025]

Outline of Presentation



- Tutorial Overview
- **Background Knowledge**
- Mathematical Model of Quantum Computing
- Janus (Taiyuan) Quantum Cloud Platform
- Installing JanusQ

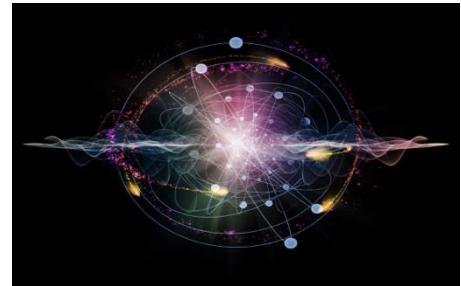


浙江大學
ZHEJIANG UNIVERSITY

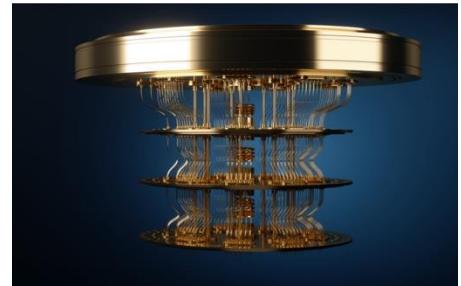
Background Knowledge



Development of
Classical Computing



Motivation of
Quantum Computing



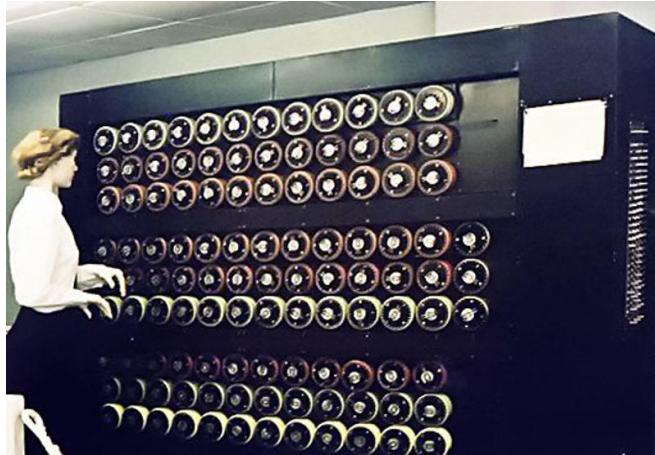
Development of
Quantum Computing

Development Of Classical Computing

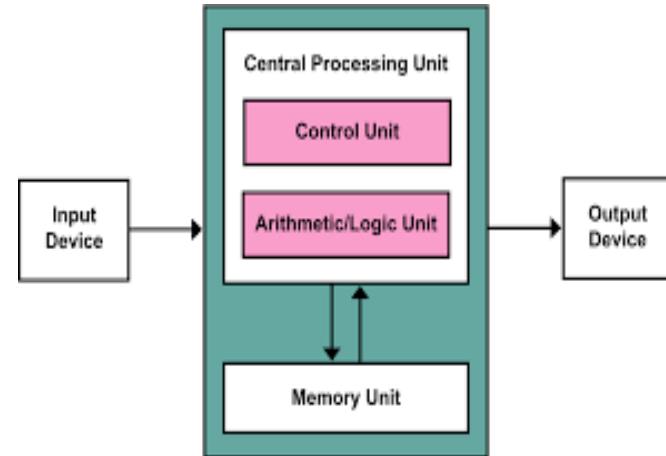


浙江大學
ZHEJIANG UNIVERSITY

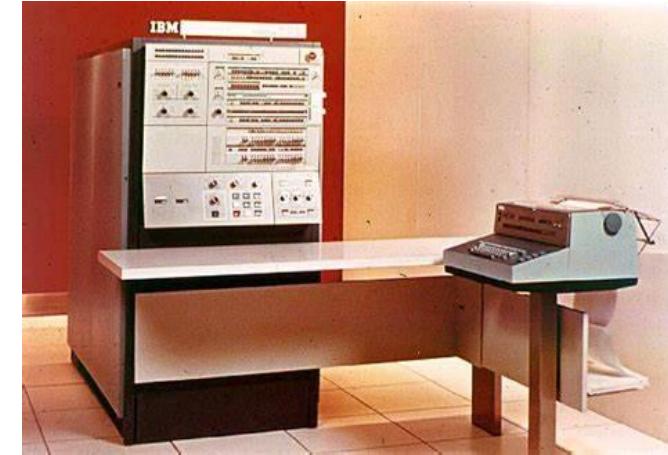
Domain-specific calculator (1939)



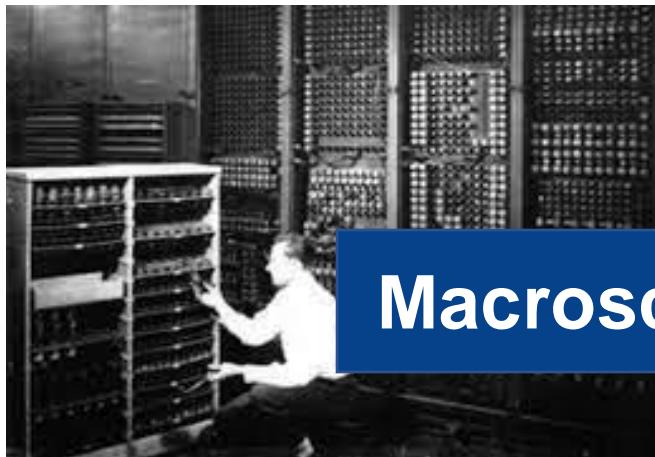
Von Neumann architecture (1947)



IBM360 Integrated Circuit (1964)



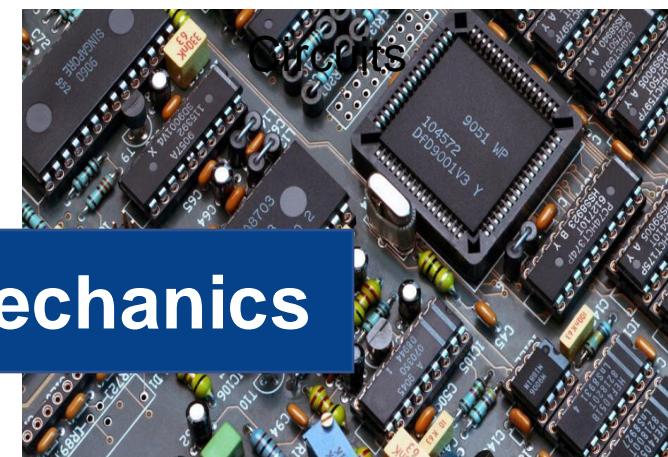
Vacuum Tube Computer ENIAC (1942)



Transistor Computer TRADIC



Large Scale Integrated Circuits

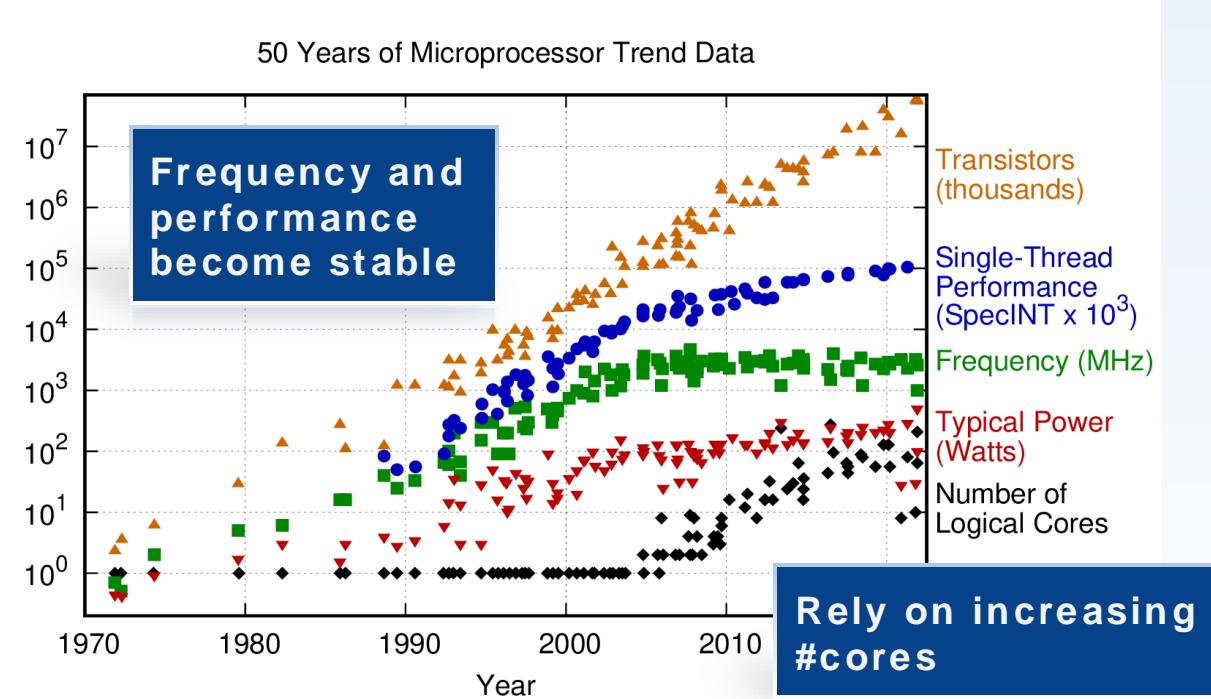


Macroscopic Effects of Quantum Mechanics

Motivation Of Quantum Computing



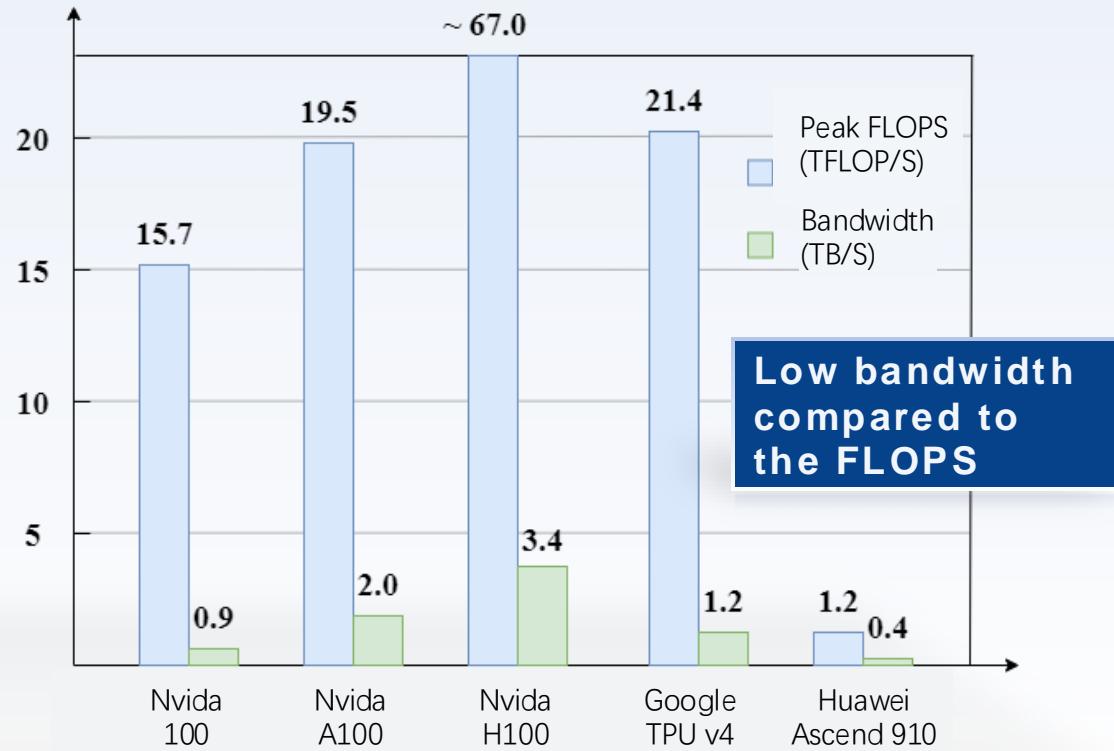
Computation barrier



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonne, O. Shacham, K. Olukolu, L. Hammond, and C. Balen. New plot and data collected for 2010-2021 by K. Rupp.

- The research costs and cycles of advanced chip processes are continuously increasing, Moore's Law is approaching obsolescence.
- The computing systems cannot rely solely on the development of traditional single chips. Instead, it requires new chip design methods and computing principles.

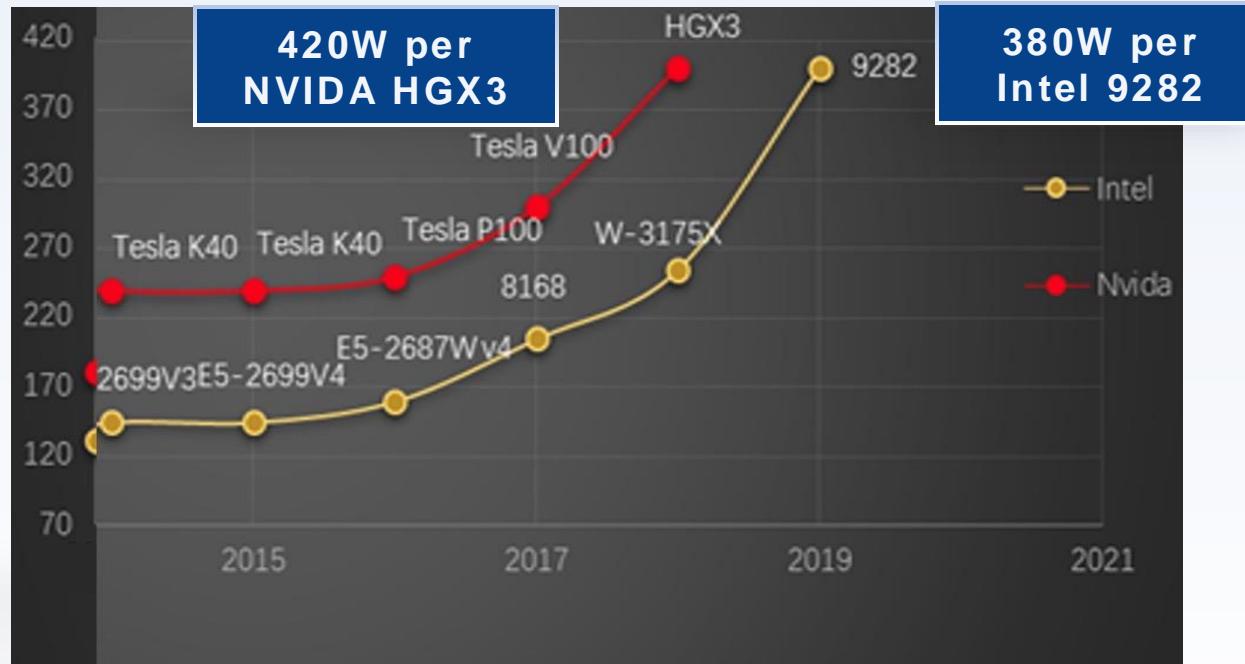
Storage barrier



- Computation power and bandwidth is not matched.
- Latency of memory access is high, limiting CPU performance.
- Quantum computing is in memory.
- Non-von Neumann architectures.

Power wall

Thermal design power exponentially increases.



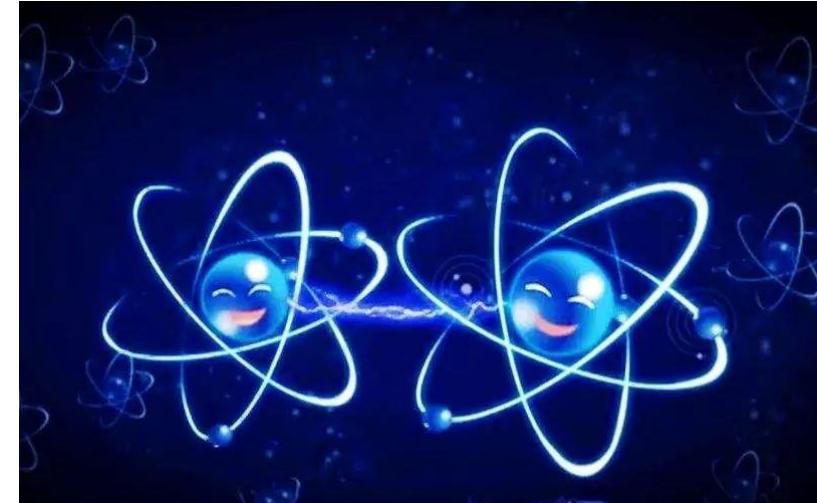
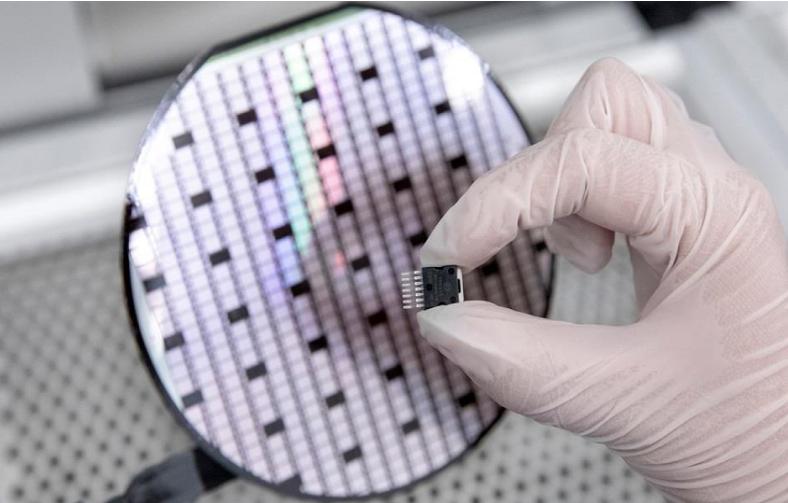
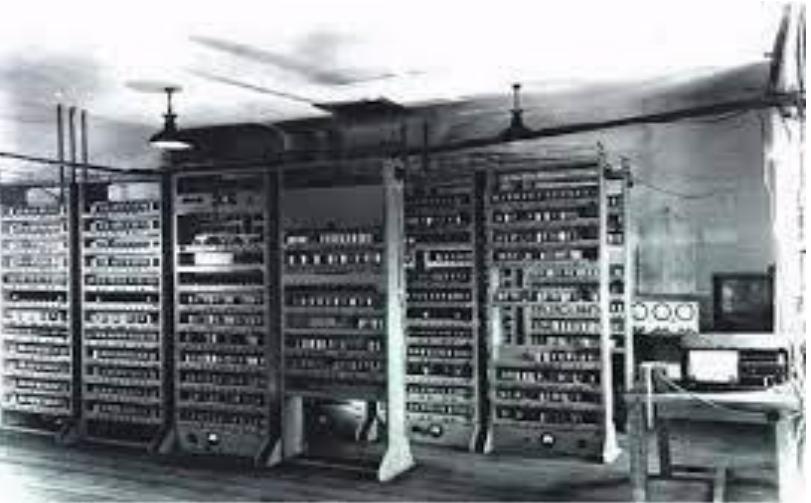
From <https://www.blueocean-china.net/faq3/241.html>

- The power consumption **increases exponentially with computing power.**
- Energy consumption **restricts computing power.**
- Systematical resource scheduling at the architectural level.

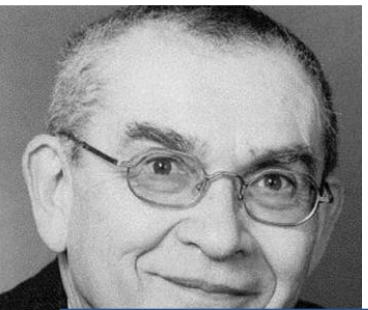
Proposal of Quantum Computing



浙江大學
ZHEJIANG UNIVERSITY



- Classical physics is no longer able to fully describe the underlying physical mechanisms at its core.



Yuri Manin

- Algebraic Geometry
- Discrete Geometry (Diophantine Geometry)
- Manin (1980) and Feynman

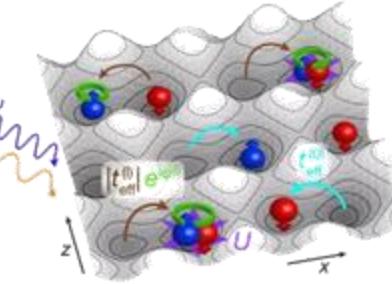


Richard Feynman

- Feynman Path Integral, Feynman Diagram, Feynman Parton Model
- Quantum Electrodynamics, Nobel Prize in Physics

generalize to other domains such as the database search and cryptography

Basic Concepts of Quantum Computing

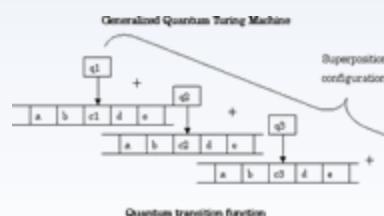
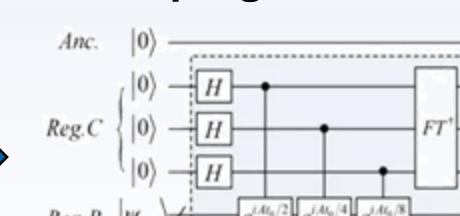
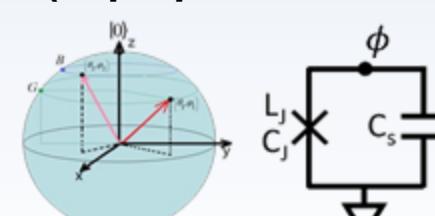


Physical
Simulation

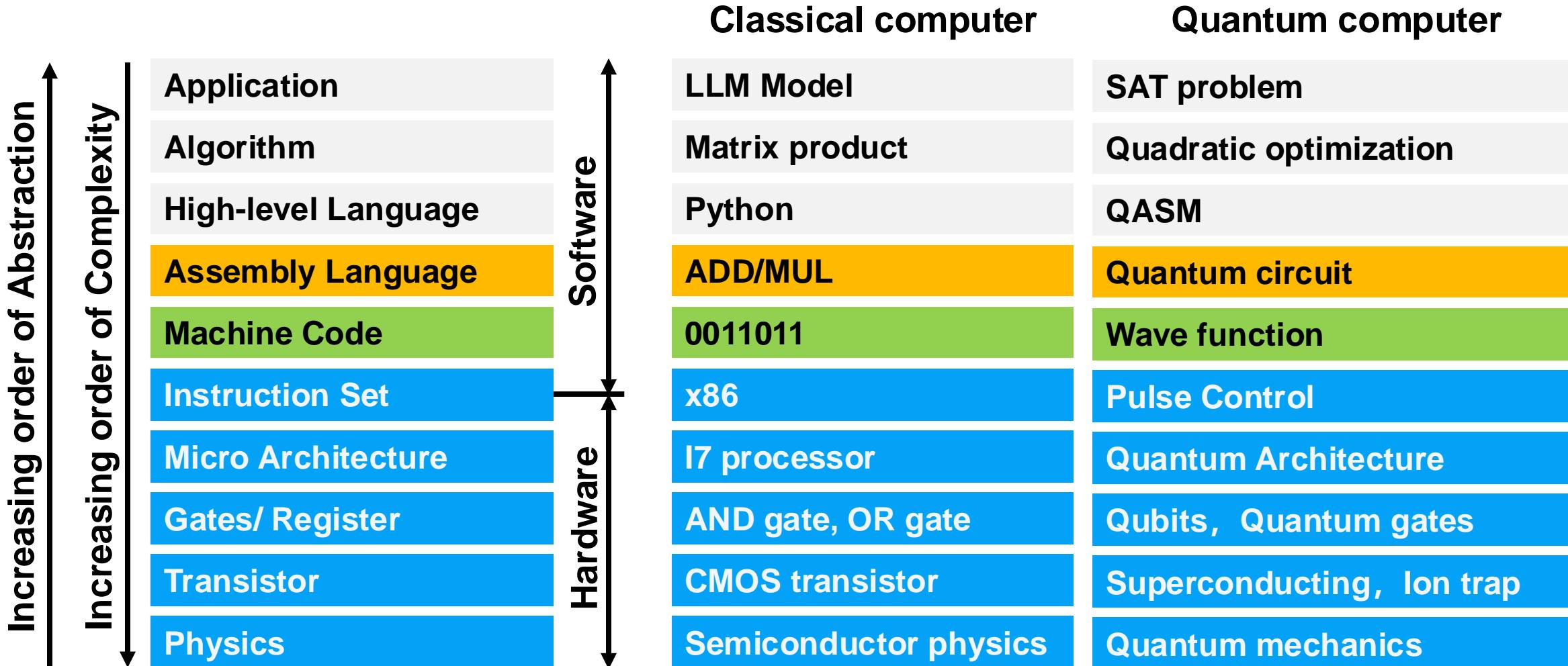


Factor

Application

		Computing theory	Encoding	Physical implementation
Classical program		Classical Turing machine	Program based on binary encoding	CMOS (0/1)
Quantum program	Quantum	Quantum Turing machine	Circuit-based quantum program	Superconducting, photon (superposition state)
	lead to high computational advantages via parallelism.	 $\Lambda : \mathfrak{S}(\mathcal{H}) \rightarrow \mathfrak{S}(\mathcal{H})$ $\mathcal{H} = \mathcal{H}_0 \otimes \mathcal{H}_E \otimes \mathcal{H}_G$	 $ 0\rangle$ $ 0\rangle$ $ 0\rangle$ $ \psi_{\Lambda_0}\rangle$ U_{PE}	 Josephson junction
		suffer from complexity due to quantum collapse and entanglement		has low energy consumption due to reversibility and superconductivity.

Quantum Computing Architecture



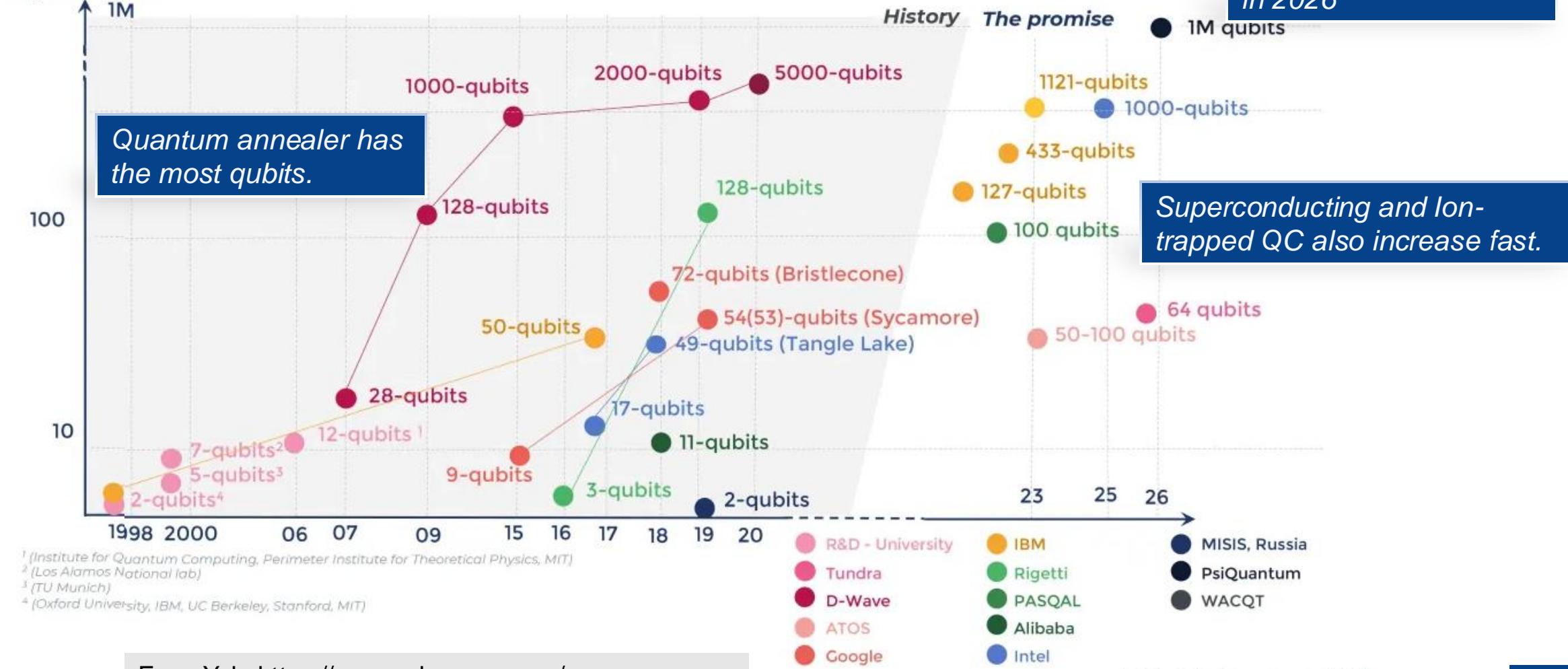
From <https://www.secplicity.org/2018/09/19/understanding-the-layers-of-a-computer-system/>

Classical and quantum computing has similar architecture

Development of Quantum Computer

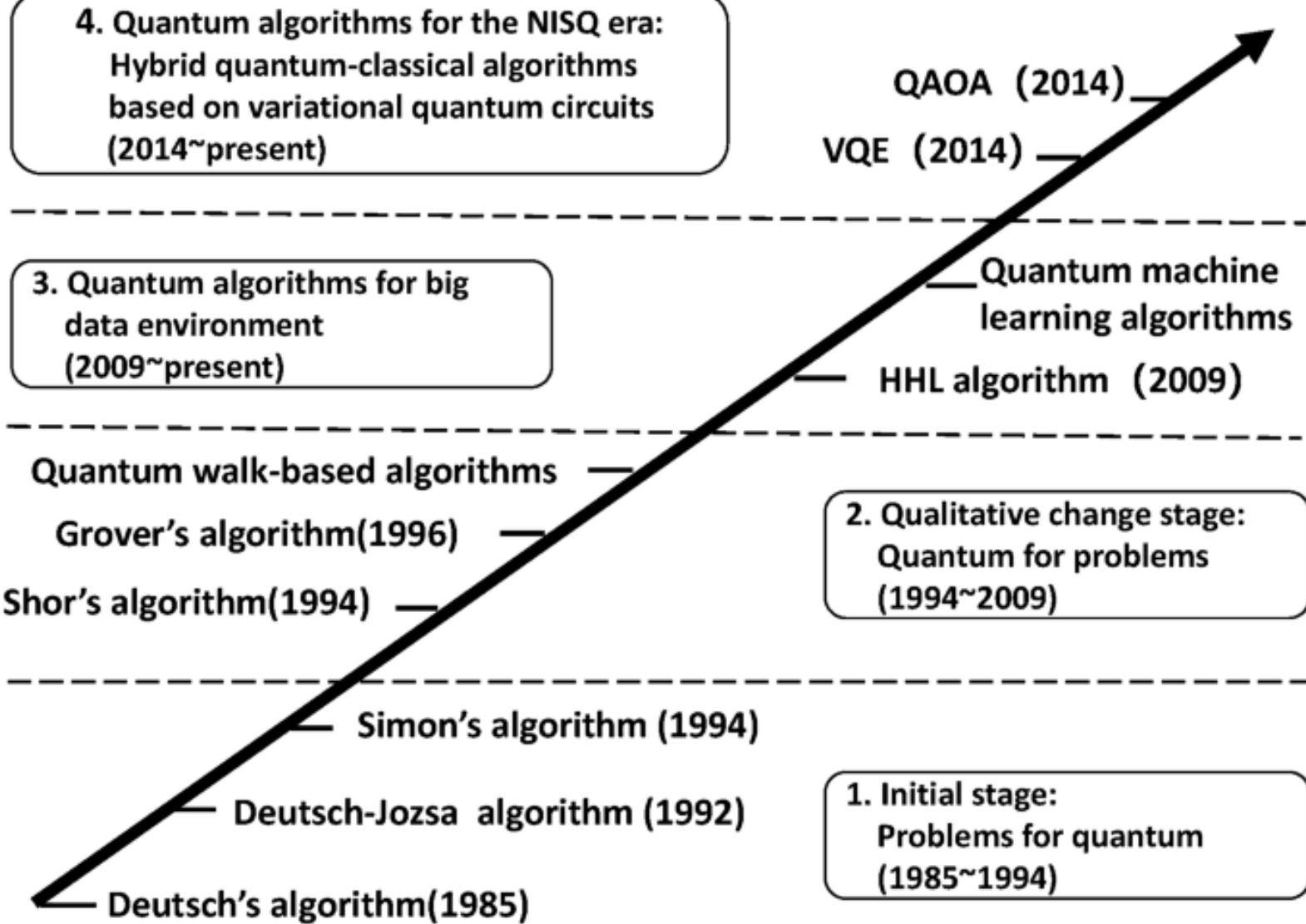


Graph below shows physical qubit roadmap (Note: for a quantum computer, 50 logical qubits minimum are required → it means 50 000 physical qubits)
Physical qubits



From Yole <https://www.yolegroup.com/press-release/quantum-technologies-the-market-is-growing/>

Development of Quantum Algorithm



Zhang, S., Li, L. A brief introduction to quantum algorithms. *CCF Trans. HPC* 4, 2022

Outline of Presentation

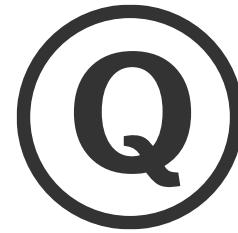


- Tutorial Overview
- Background Knowledge
- **Mathematical Model of Quantum Computing**
- Janus (Taiyuan) Quantum Cloud Platform
- Installing JanusQ



浙江大學
ZHEJIANG UNIVERSITY

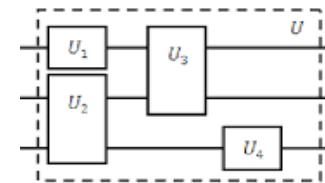
Mathematical Model of Quantum Computing



Qubits



Quantum Evolution



Quantum Circuit

Quantum Bit (Qubit)



Single qubit

A **qubit** has two bases $|0\rangle$ and $|1\rangle$. The information stored in its **superposition state** is represented as a **2-dimension state vector** $|\varphi\rangle$.

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

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

$$|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$$|\varphi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

$$\text{subject to } |\alpha|^2 + |\beta|^2 = 1$$

Quantum Bit (Qubit)



Single qubit

A **qubit** has two bases $|0\rangle$ and $|1\rangle$. The information stored in its **superposition state** is represented as a **2-dimension state vector** $|\varphi\rangle$.

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

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

$$|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$$|\varphi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

$$\text{subject to } |\alpha|^2 + |\beta|^2 = 1$$

Multiple qubits

N qubits has 2^N bases $|00\cdots 0\rangle$, $|00\cdots 1\rangle\cdots$, $|11\cdots 1\rangle$. The information stored in their **superposition state** is represented as a **2^N -dimension state vector**.

$$|00\cdots 0\rangle = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad |00\cdots 1\rangle = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad \cdots \quad |11\cdots 1\rangle = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

$$|\varphi\rangle = \alpha_0|00\cdots 0\rangle + \alpha_1|00\cdots 1\rangle + \cdots + \alpha_{2^N}|11\cdots 1\rangle$$

$$|\varphi\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{2^N} \end{bmatrix}$$

$$\text{subject to } |\alpha_0|^2 + |\alpha_1|^2 + \cdots + |\alpha_{2^N}|^2 = 1$$



Unitary matrix

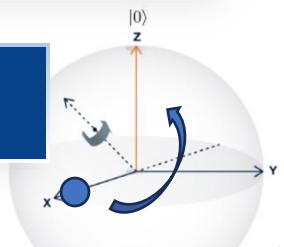
A quantum evolution caused by **quantum gates** is represented as a **unitary matrix (unitary)**, which is a square matrix whose conjugate transpose is its inverse.

$$UU^\dagger = I$$

The evolution of qubit state $|\varphi\rangle$ is represented as:

Bloch sphere

X+Z axes

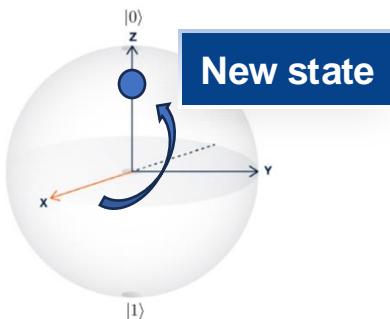


State

$$|\varphi'\rangle = U|\varphi\rangle$$

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

$\xrightarrow{\pi \text{ rotation around X+Z axes}}$



New state



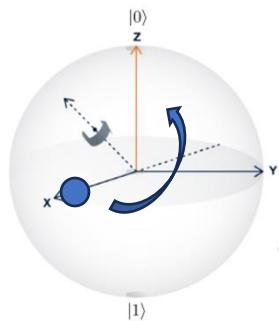
Unitary matrix

A quantum evolution caused by **quantum gates** is represented as a **unitary matrix (unitary)**, which is a square matrix whose conjugate transpose is its inverse.

$$UU^\dagger = I$$

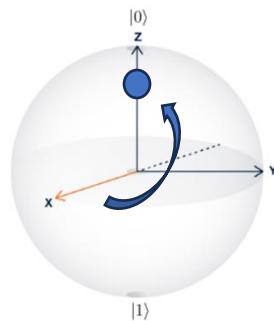
The evolution of qubit state $|\varphi\rangle$ is represented as:

$$|\varphi'\rangle = U|\varphi\rangle$$



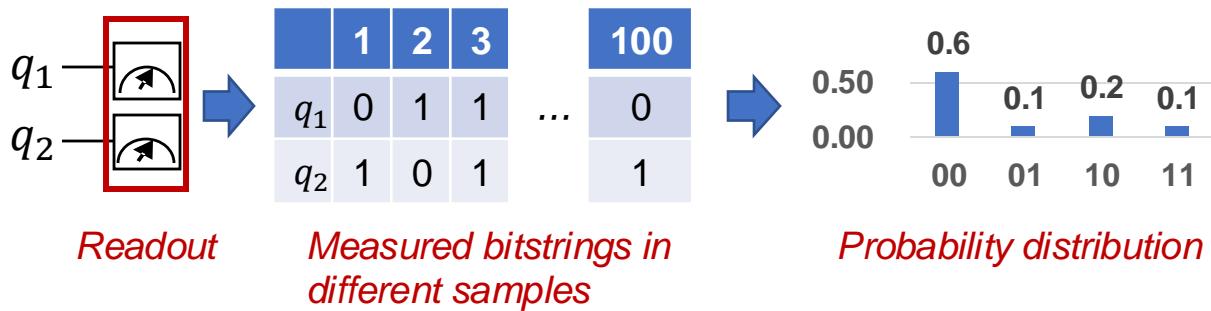
$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

π rotation around
X+Z axis:
Exchanges X and
Z



Quantum readout

A sampling of a quantum state is a bitstring. Multiple sampling of this state composes a probability distribution of measuring different bitstrings.



Quantum Circuit



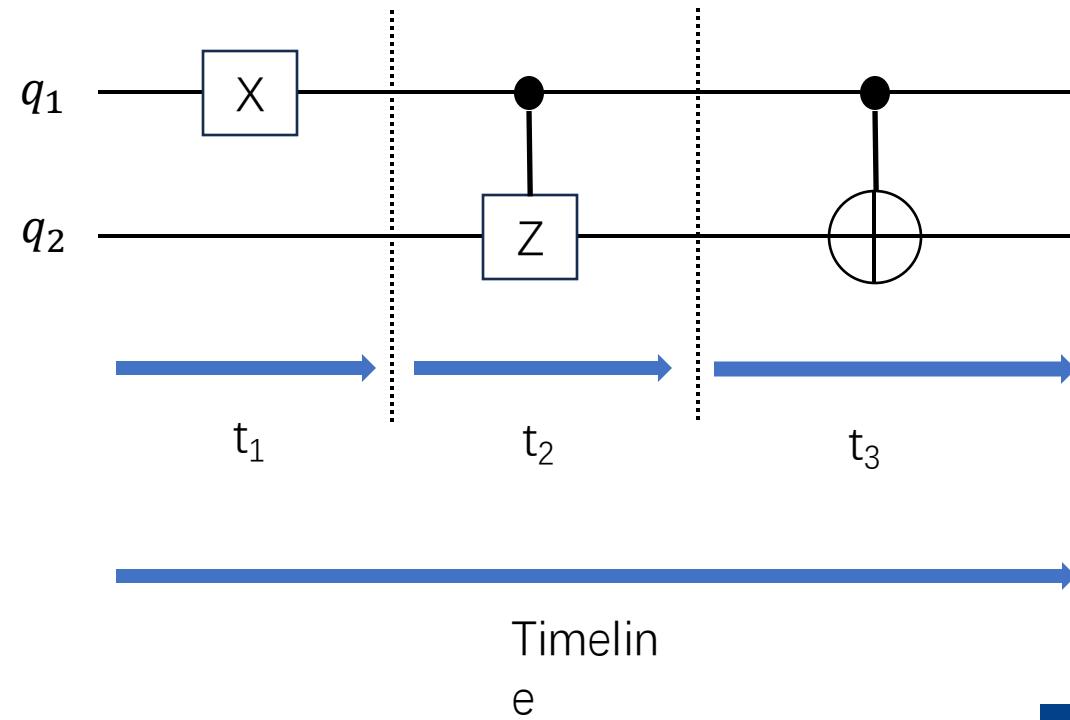
Quantum gates

In the quantum circuit model of computation, a **quantum gate** is a **basic quantum circuit operating** on a small number of qubits.

Operator	Gate(s)	Matrix
Pauli-X (X)		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Controlled Not (CNOT, CX)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled Z (CZ)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

Qubit timeline

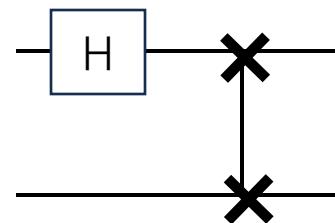
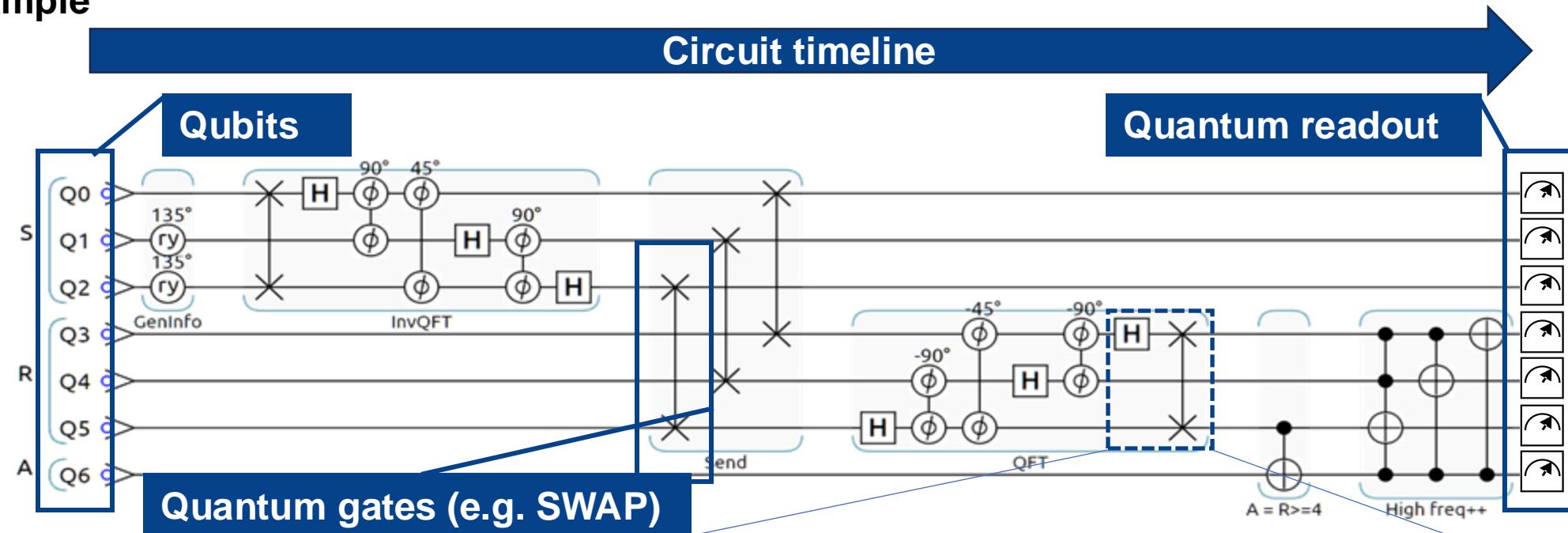
Each line in a quantum circuit represents a **qubit**. The quantum gate in a line is applied on the same qubits **from left to right in time direction**.



Quantum Circuit



For example



=

$$SWAP \cdot (H \otimes I) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

Implementation of Quantum Circuit

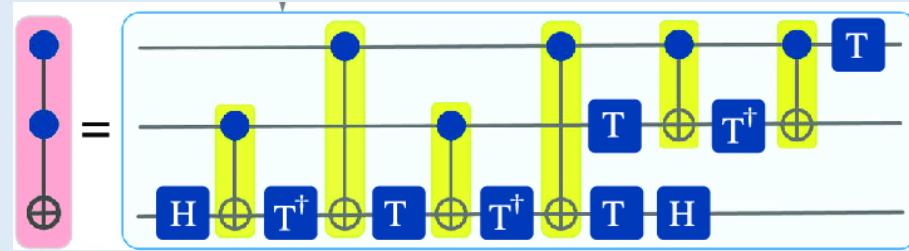
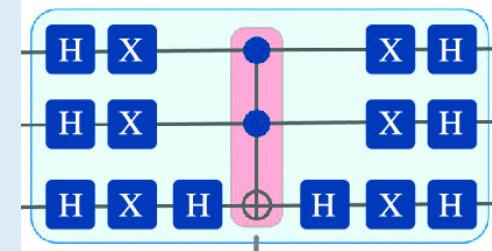


On superconducting quantum computer

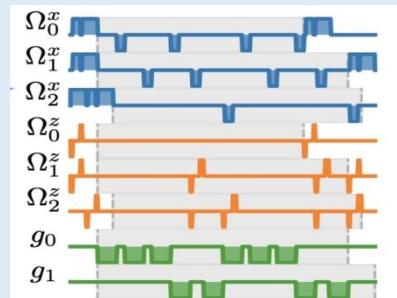
Step 1. Circuit statement

```
OPENQASM 2.0;  
qreg qubits[3];  
H qubits[1];H qubits[2];  
H qubits[3];  
X qubits[1];X qubits[2];  
X qubits[3];  
H qubits[3];  
Toffoli qubits[1], qubits[2], qubits[3];  
H qubits[3];  
X qubits[1];X qubits[2];  
X qubits[3];  
H qubits[1];H qubits[2];  
H qubits[3];
```

Step 2. Circuit compilation



Step 3. Circuit execution



Pulse generation

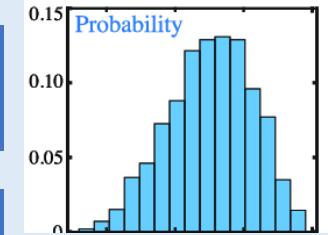


Quantum device

Step 4. Result processing

Error mitigation

Visualization



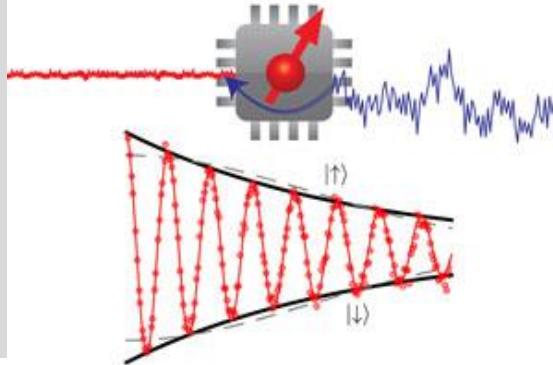
Probability distribution

Challenge in Quantum Computing



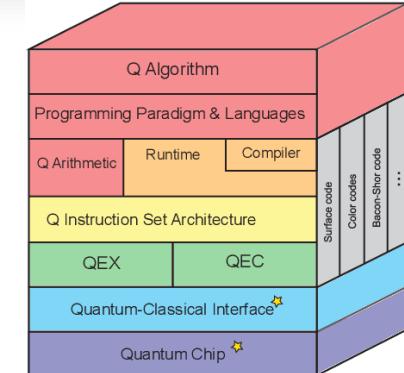
Noise

- Coherence error
- Gate error
- State preparation error
- Readout error
- Crosstalk

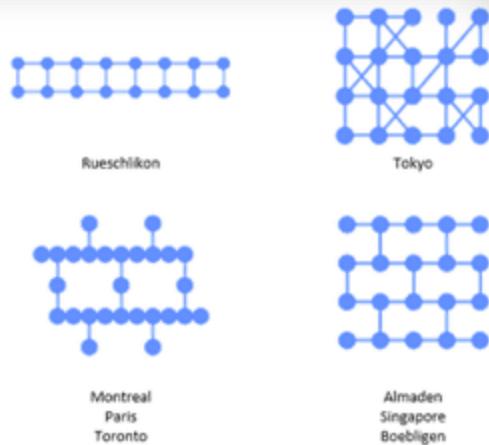


Instructions

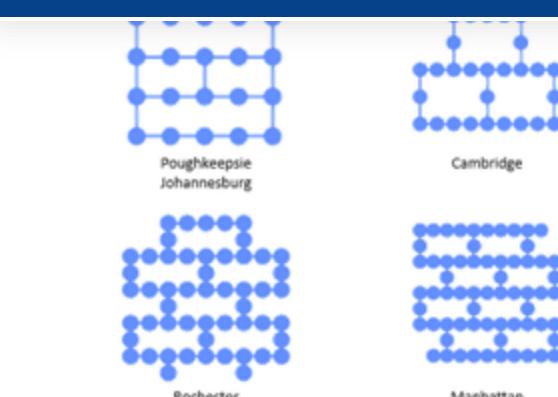
- Multi-level compilation
- Micro-architecture instructions
- Quantum-classical communication



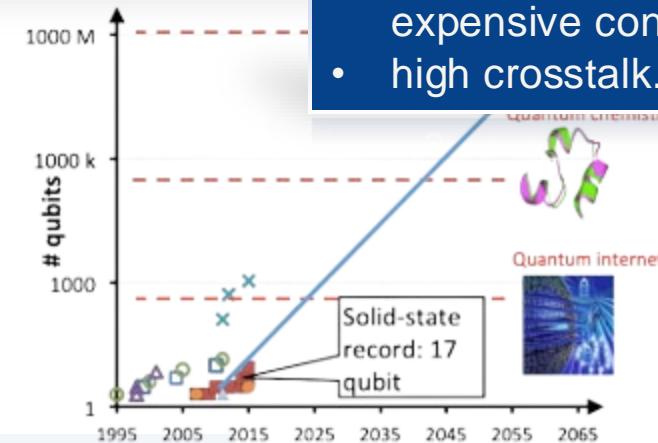
Topology



Locality in communication



Scalability



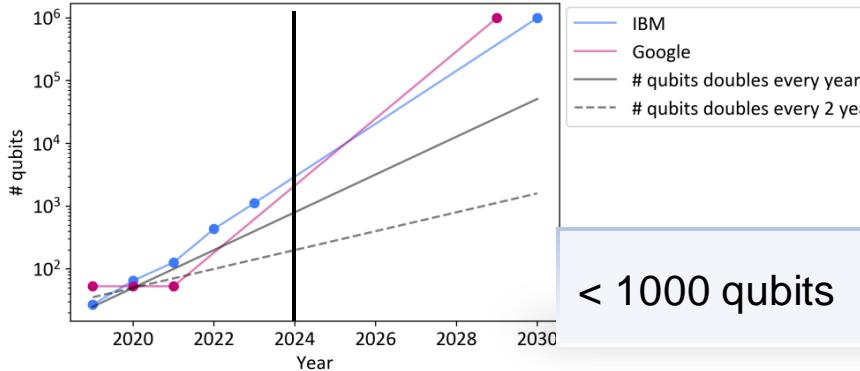
- poor fabrication techniques
- expensive control systems
- high crosstalk.

Constrained by physical implementation

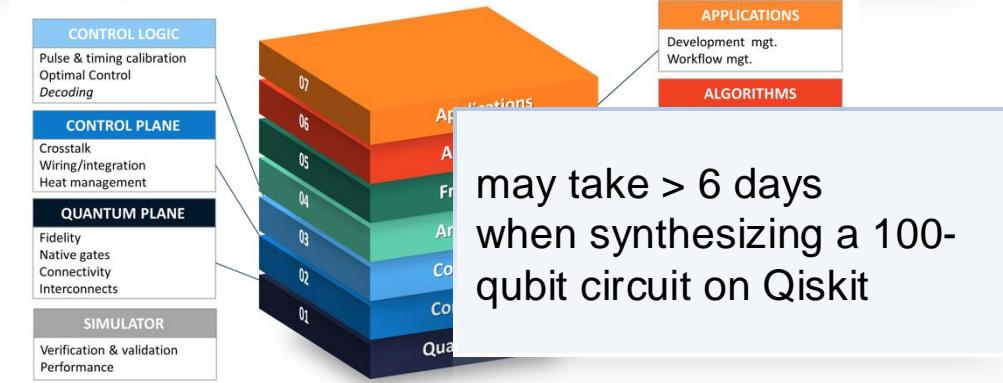
Results of Challenges



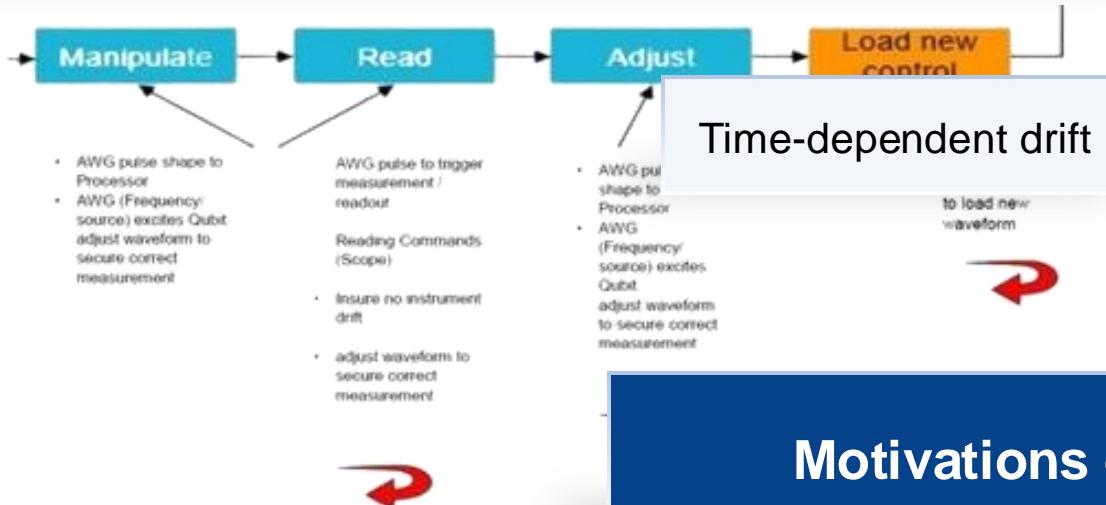
Limited Hardware Resource



Compilation Complexity



Difficulty Of Hardware Calibration



Rare quantum advantage

- Limited qubits resources
- High error rate
- Hard to ensure quantum advantage in real applications

Long calibration time (e.g., readout calibration)



Motivations of JanusQ

Outline of Presentation



- Tutorial Overview
- Background Knowledge
- Mathematical Model of Quantum Computing
- **Janus (Taiyuan) Quantum Cloud Platform**
- Installing JanusQ

Janus (Taiyuan) Quantum Cloud Platform



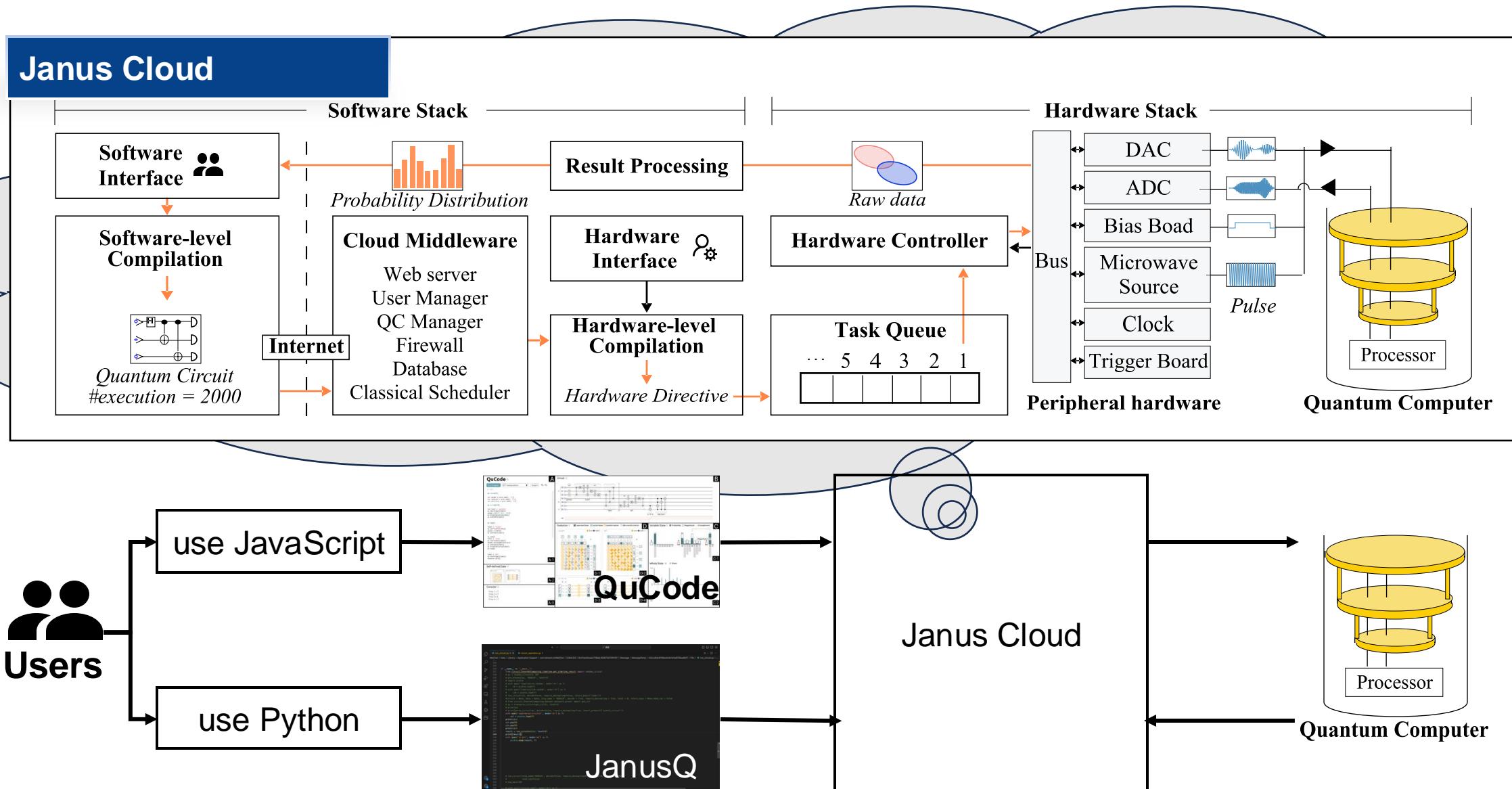


Tai Yuan is a Chinese gold who is the wife of Pan Gu (the god that created the world). She was delivered of Yin and Yang, which is entangled like a quantum superposition state

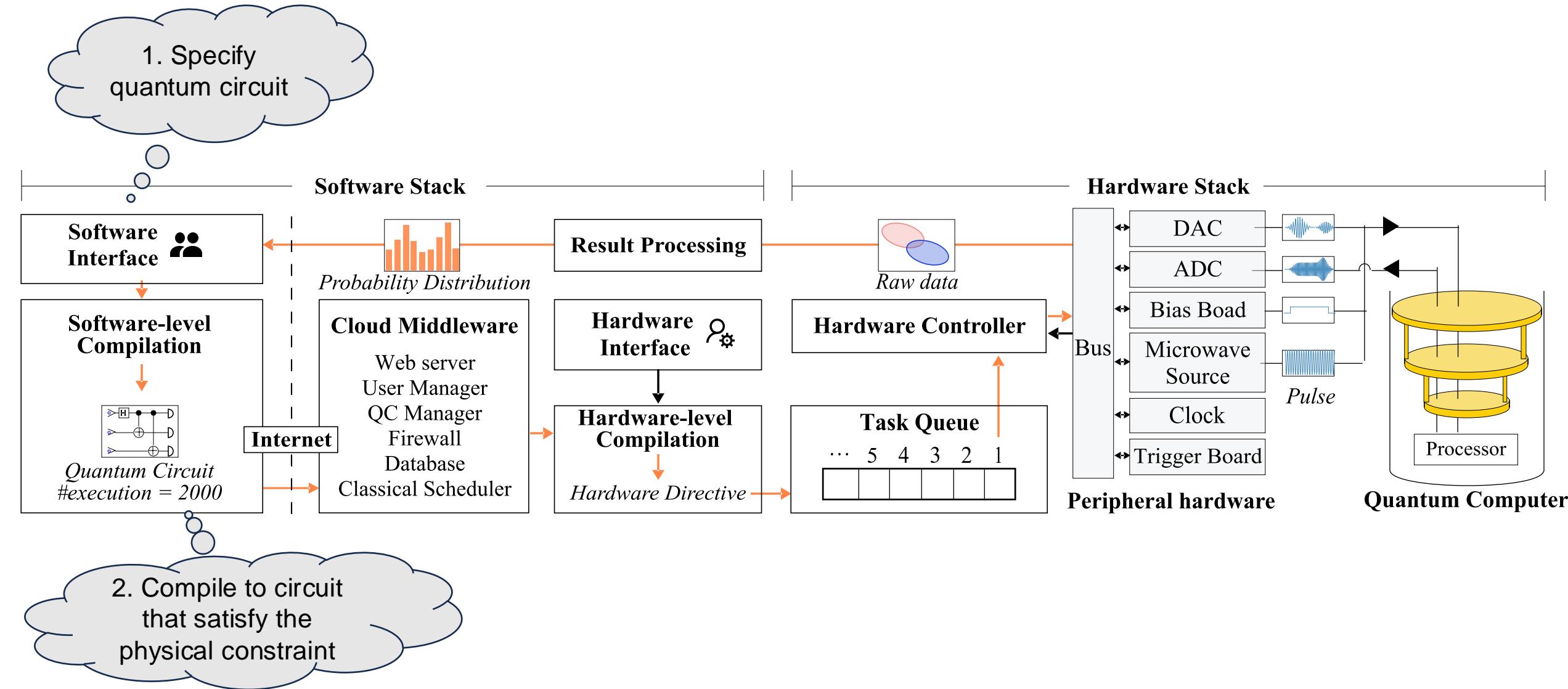


Janus is a two-faced god of the ancient Romans. He is the god of beginning, transition, time, change, dualism, and end. He has two faces simultaneously.

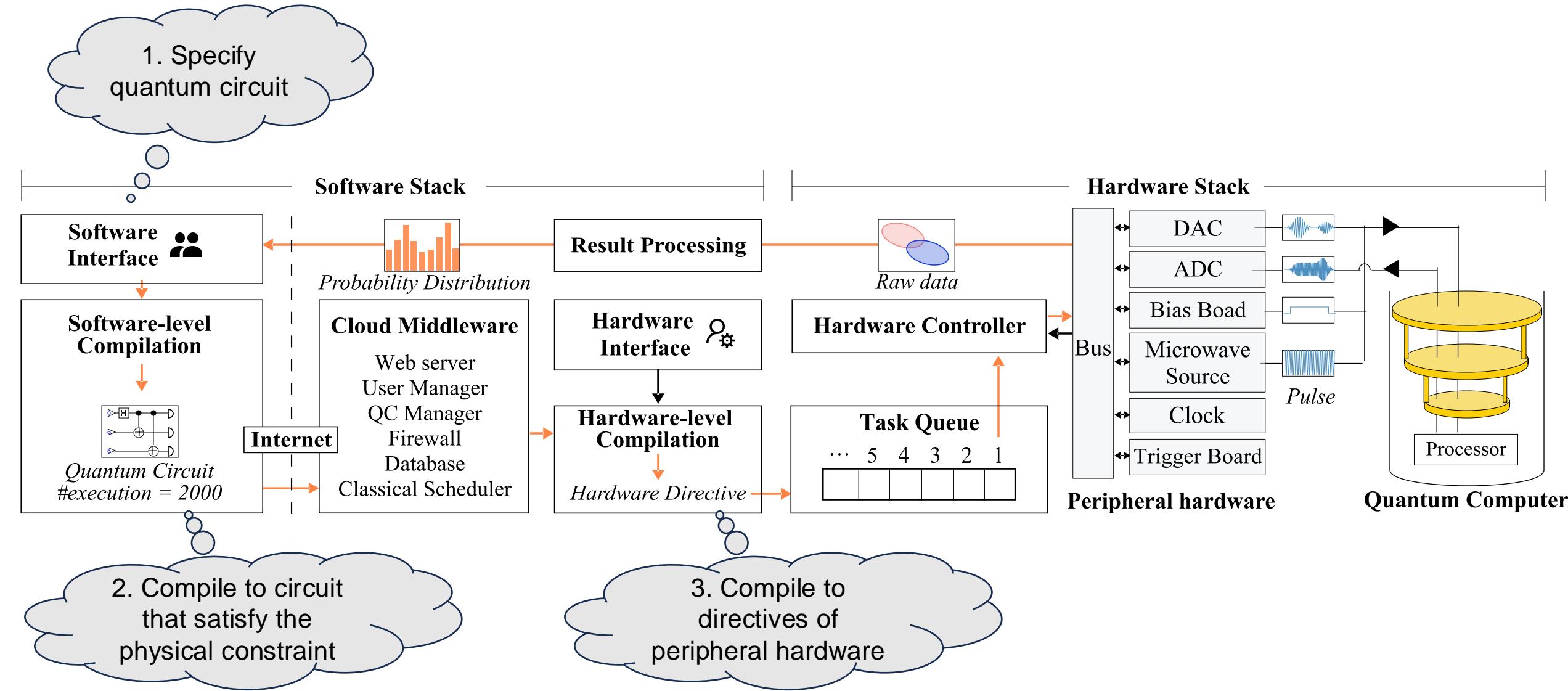
What we can do on Janus Platform



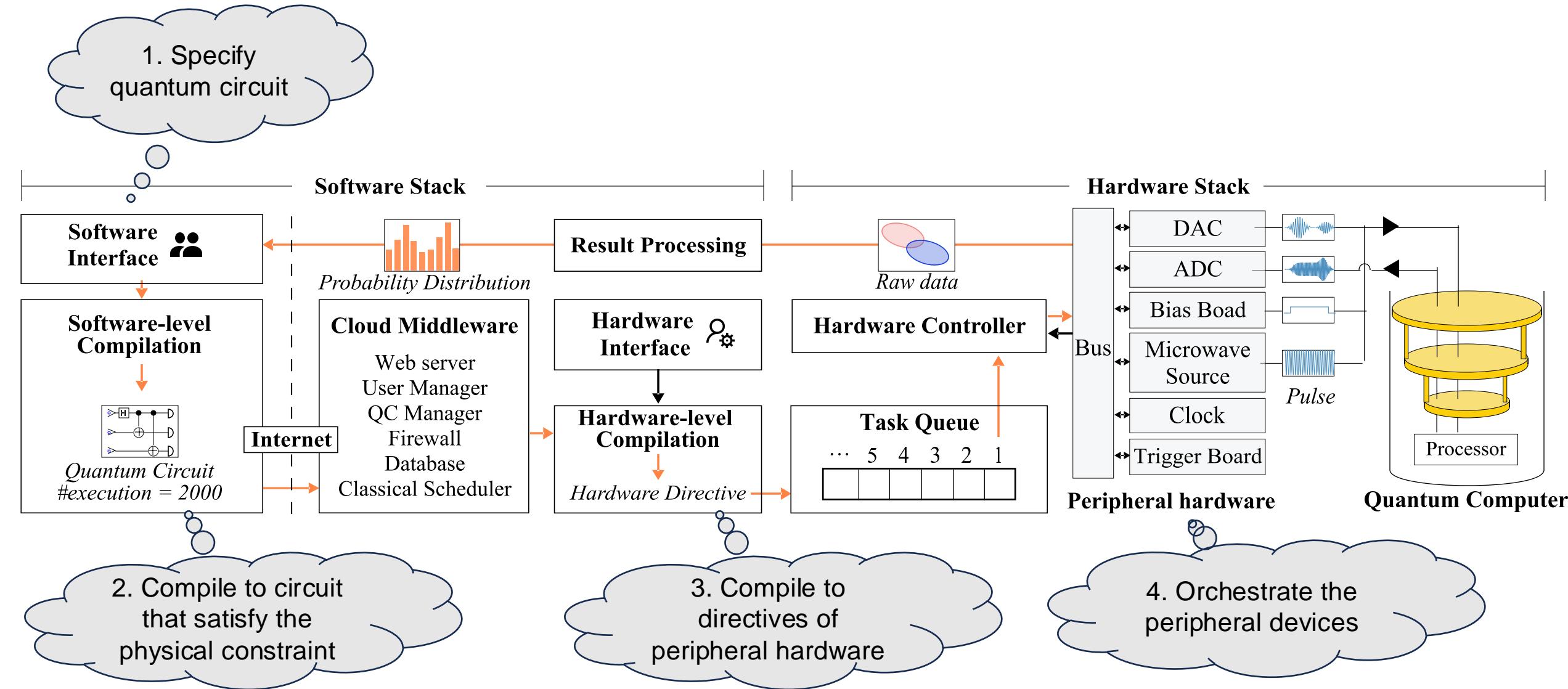
What we can do on Janus Platform



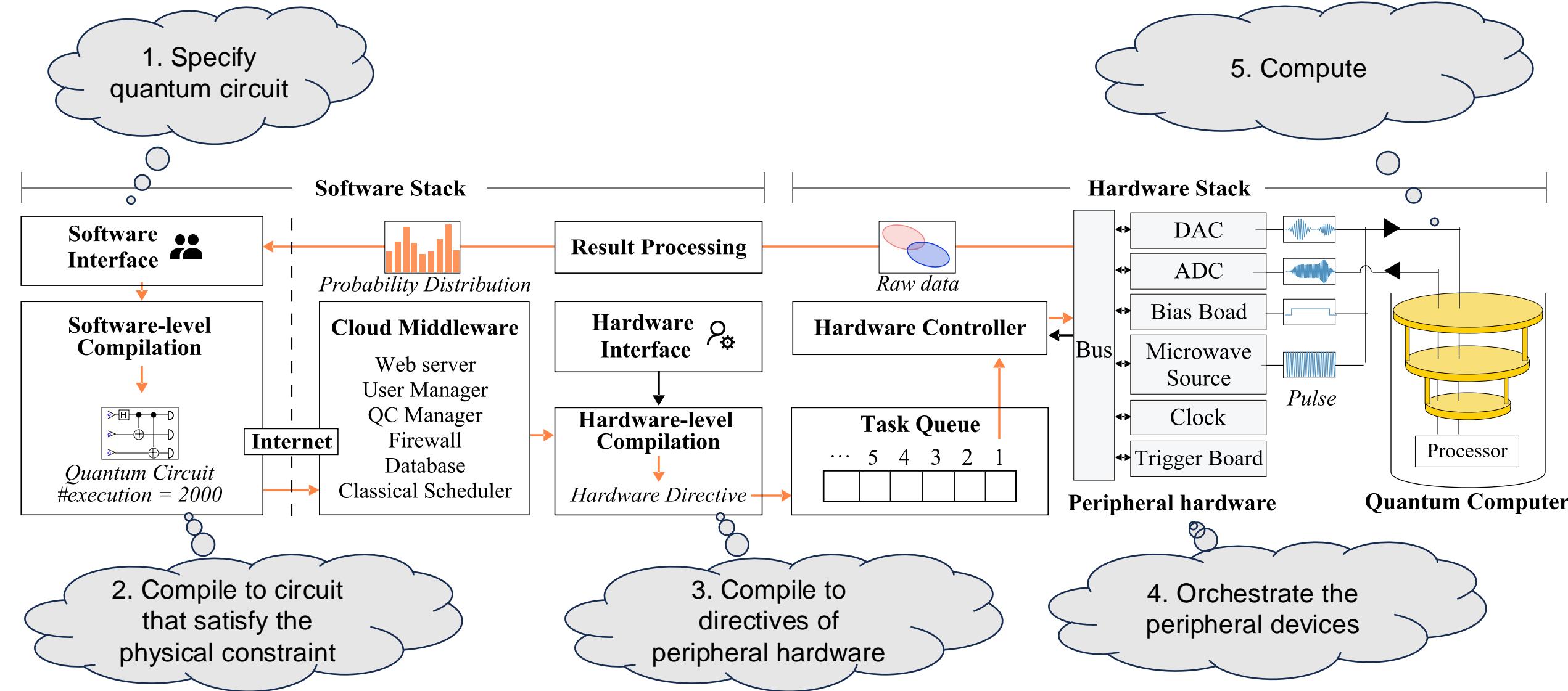
What we can do on Janus Platform



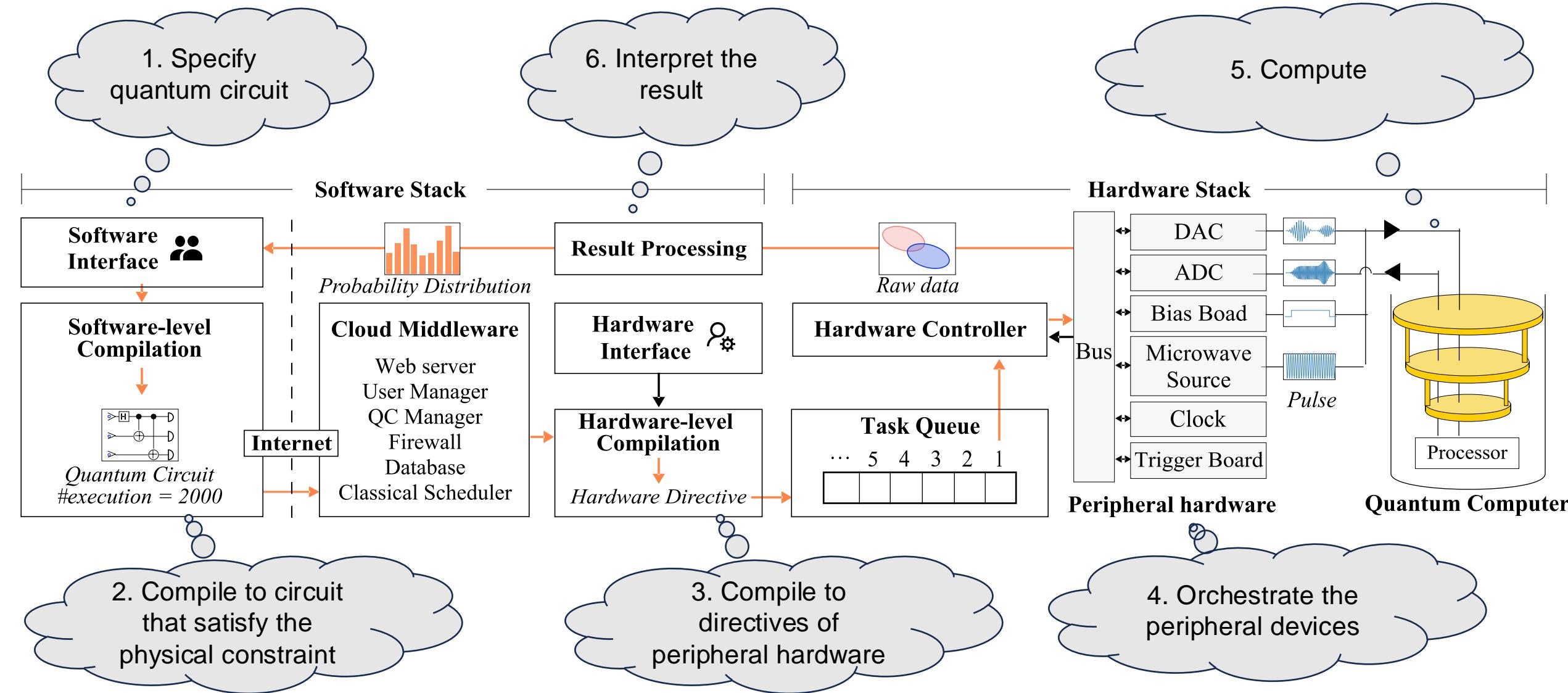
What we can do on Janus Platform



What we can do on Janus Platform



What we can do on Janus Platform



Creating an Account



浙江大學
ZHEJIANG UNIVERSITY

Go to Janus Cloud

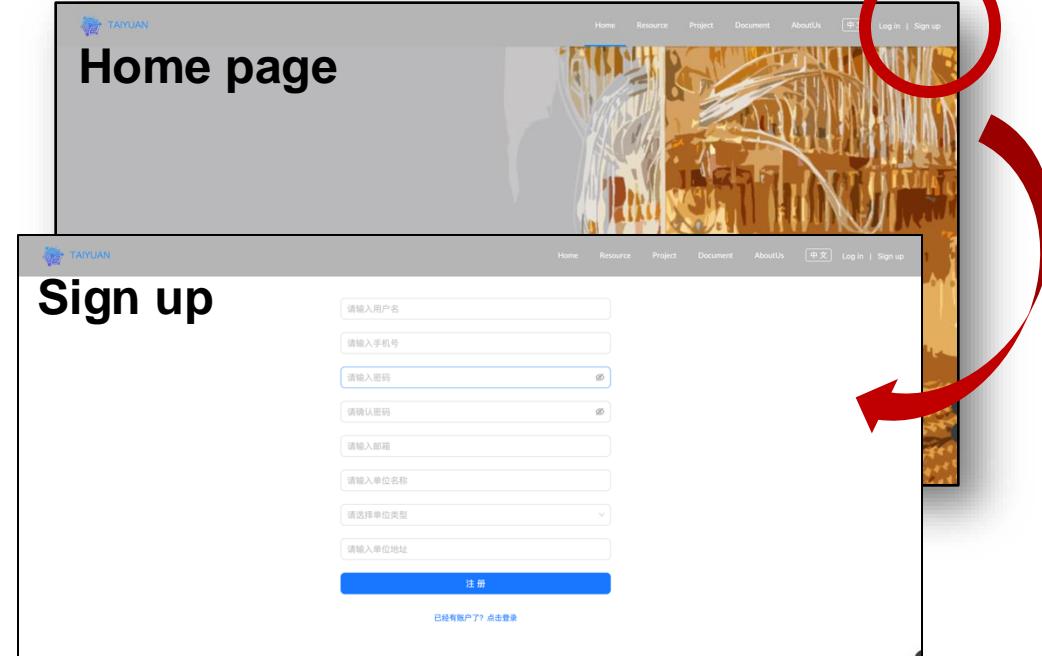
Open <http://janusq.zju.edu.cn/home>



QR code

Sign up

Click sign up button



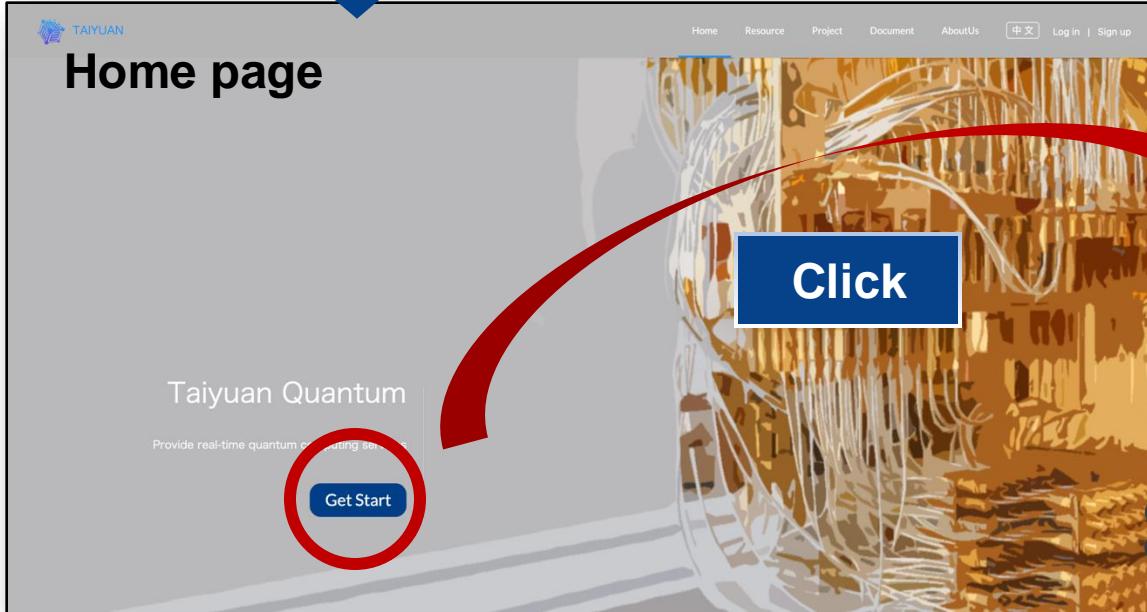
- After signing up, you will get an API key for task submission.
- During the tutorial, you can access the platform without registration.

Running Program On the Website

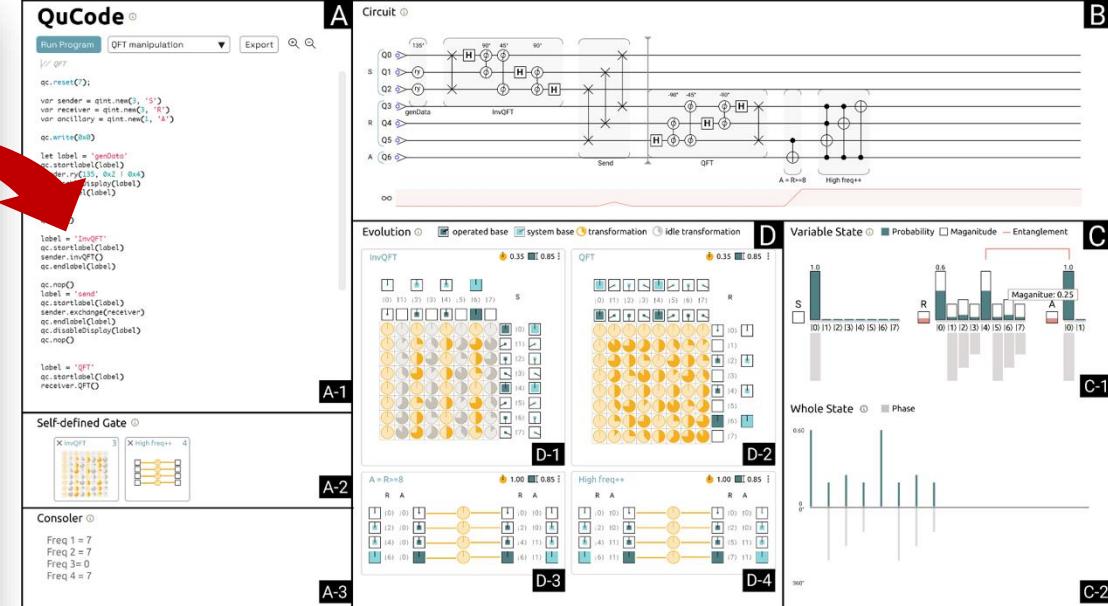


Submitting on website

Open <http://janusq.zju.edu.cn/home>



Programming on the Code editor

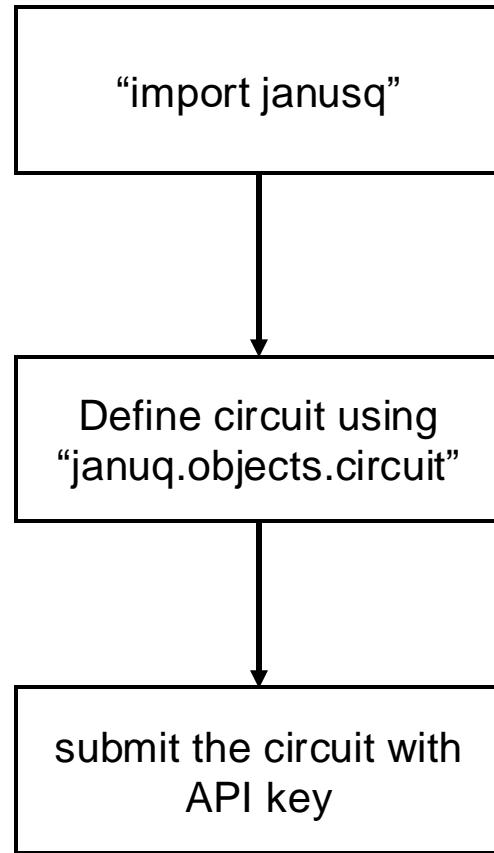


- Click the “Get Start” button to experience JanusQ.

Running Program By Python API



Submitting by Python



Import package {

```
import matplotlib.pyplot as plt
from janusq.cloud_interface import submit, get_result
from janusq.objects.circuit import Circuit
```

Define a circuit {

```
qc = Circuit([], n_qubits = 4)
qc.h(0, 0)
qc.cx(0, 1, 1)
qc.cx(1, 2, 2)
qc.cx(2, 3, 3)
print(qc)
```

Submit circuit {

```
result = submit(circuit=qc, label= 'GHZ', shots= 3000,
run_type='simulator', API_TOKEN="")
```

Get result {

```
result = get_result(result['data']['result_id'],
run_type='simulator', result_format='probs')
print(result)
```

QuCode: Online Quantum Program Editor



浙江大學
ZHEJIANG UNIVERSITY

QuCode ⓘ

Run Program
QFT manipulation
Export

```
VV QFT
qc.reset(7);

var sender = qint.new(3, 'S')
var receiver = qint.new(3, 'R')
var ancillary = qint.new(1, 'A')

qc.write(0x0)

let label = 'genData'
qc.startlabel(label)
sender.ry(135, 0x2 | 0x4)
qc.disableDisplay(label)
qc.endlabel(label)

qc.nop()

label = 'InvQFT'
qc.startlabel(label)
sender.invQFT()
qc.endlabel(label)

qc.nop()
label = 'send'
qc.startlabel(label)
sender.exchange(receiver)
qc.endlabel(label)
qc.disableDisplay(label)
qc.nop()

label = 'QFT'
qc.startlabel(label)
receiver.QFT()
```

Program editor

Circuit reuse

Console

```
Freq 1 = 7
Freq 2 = 7
Freq 3 = 0
Freq 4 = 7
```

A
B

Circuit ⓘ

Quantum circuit view

D

C

A-1
B
C

A-2
D-1
C-1

A-3
D-2
C-2

Evolution view
D-3
D-4

Console
D-1
D-2

HPCA 2025

52

QuCode: Online Quantum Program Editor



浙江大學
ZHEJIANG UNIVERSITY

Program editor

A

Circuit ①

Evolution ②

	operated base	system base	transformation	idle transformation
InvQFT			0.35	0.85
QFT			0.35	0.85

Variable State ③

S

R

A

Whole State ④

B

C

D

D-1

D-2

D-3

D-4

A-1

A-2

A-3

A-4

A-5

A-6

A-7

A-8

A-9

A-10

A-11

A-12

A-13

A-14

A-15

A-16

A-17

A-18

A-19

A-20

A-21

A-22

A-23

A-24

A-25

A-26

A-27

A-28

A-29

A-30

A-31

A-32

A-33

A-34

A-35

A-36

A-37

A-38

A-39

A-40

A-41

A-42

A-43

A-44

A-45

A-46

A-47

A-48

A-49

A-50

A-51

A-52

A-53

A-54

A-55

A-56

A-57

A-58

A-59

A-60

A-61

A-62

A-63

A-64

A-65

A-66

A-67

A-68

A-69

A-70

A-71

A-72

A-73

A-74

A-75

A-76

A-77

A-78

A-79

A-80

A-81

A-82

A-83

A-84

A-85

A-86

A-87

A-88

A-89

A-90

A-91

A-92

A-93

A-94

A-95

A-96

A-97

A-98

A-99

A-100

A-101

A-102

A-103

A-104

A-105

A-106

A-107

A-108

A-109

A-110

A-111

A-112

A-113

A-114

A-115

A-116

A-117

A-118

A-119

A-120

A-121

A-122

A-123

A-124

A-125

A-126

A-127

A-128

A-129

A-130

A-131

A-133

A-135

A-137

A-139

A-141

A-143

A-145

A-147

A-149

A-151

A-153

A-155

A-157

A-159

A-161

A-163

A-165

A-167

A-169

A-171

A-173

A-175

A-177

A-179

A-181

<div data-bbox="235 3800 325 382

QuCode: Online Quantum Program Editor



浙江大學
ZHEJIANG UNIVERSITY

Program editor A Circuit B

Provide 14 examples:

1. QFT
2. teleportation
3. Grover
- ...

Run Program QFT manipulation Export

```
// QFT
qc.reset();
var sender = qint.new(3, 'S');
var receiver = qint.new(3, 'R');
var ancillary = qint.new(1, 'A');

qc.write(0x0);

let label = 'genData';
qc.startlabel(label)
sender.ry(135, 0x2 | 0x4)
qc.disableDisplay(label)
qc.endlabel(label)

qc.nop()

label = 'InvQFT'
qc.startlabel(label)
sender.invQFT()
qc.endlabel(label)

qc.nop()
label = 'send'
qc.startlabel(label)
sender.exchange(receiver)
qc.endlabel(label)
qc.disableDisplay(label)
qc.nop()

label = 'QFT'
qc.startlabel(label)
receiver.QFT()
```

A-1 A-2 A-3

D D-1 D-2 D-3 D-4

C C-1 C-2

Self-defined Gate

Consoler

Freq 1=7
Freq 2=7
Freq 3=0
Freq 4=7

Diagram showing a quantum circuit with various gates (H, P, S, T, CNOT) and measurements. Labels include 'Send', 'QFT', and 'A = R >= 8'. A red circle highlights the 'QFT manipulation' dropdown in the top menu.

Variable State

Probability

Magnitude

Entanglement

Whole State

Phase

Diagram showing the state of three qubits (S, R, A) across 8 basis states. The S qubit has a probability of 1.0. The R qubit has a probability of 0.6. The A qubit has a magnitude of 0.25. The phase diagram shows a periodic pattern with a period of 360 degrees.

QuCode: Online Quantum Program Editor



浙江大學
ZHEJIANG UNIVERSITY

QuCode ⓘ

Run Program QFT manipulation Export 🔍 🌐

```

// QFT
qc.reset(7);

var sender = qint.new(3, 'S')
var receiver = qint.new(3, 'R')
var ancillary = qint.new(1, 'A')

qc.write(0x0)

let label = 'genData'
qc.startlabel(label)
sender.ry[135, 0x2 | 0x4]
qc.disableDisplay(label)
qc.endlabel(label)

qc.nop()

label = 'InvQFT'
qc.startlabel(label)
sender.invQFT()
qc.endlabel(label)

qc.nop()
label = 'send'
qc.startlabel(label)
sender.exchange(receiver)
qc.endlabel(label)
qc.disableDisplay(label)
qc.nop()

label = 'QFT'
qc.startlabel(label)
receiver.QFT()
        
```

Circuit View

Grouping

Purity

A-1

D
D-1
D-2

C
C-1
C-2

A-2

D-3
D-4

A-3
Consoler ⓘ
 Freq 1 = 7
 Freq 2 = 7
 Freq 3 = 0
 Freq 4 = 7

QuCode: Online Quantum Program Editor



浙江大學
ZHEJIANG UNIVERSITY

QuCode

Run Program QFT manipulation Export 🔍 🔍

```

V// QFT

qc.reset(7);

var sender = qint.new(3, 'S')
var receiver = qint.new(3, 'R')
var ancillary = qint.new(1, 'A')

qc.write(@x0)

let label = 'genData'
qc.startlabel(label)
sender.ry(135, 0x2 | 0x4)
qc.disableDisplay(label)
qc.endlabel(label)

qc.nop()

label = 'InvQFT'
qc.startlabel(label)
sender.invQFT()
qc.endlabel(label)

qc.nop()
label = 'send'
qc.startlabel(label)
sender.exchange(receiver)
qc.endlabel(label)
qc.disableDisplay(label)
qc.nop()

label = 'QFT'
qc.startlabel(label)
receiver.QFT()
      
```

A

Circuit

Evolution

InvQFT

0.35 0.85

QFT

0.35 0.85

A = R>=8

1.00 0.85

High freq++

1.00 0.85

B

A-1

D-1

A-2

D-2

A-3

D-3

A-4

D-4

State View

partial trace

The whole state

The whole state

QuCode: Online Quantum Program Editor



浙江大學
ZHEJIANG UNIVERSITY

QuCode

Run Program QFT manipulation ▾ Export 🔍 🔍

```

V// QFT

qc.reset(7);

var sender = qint.new(3, 'S')
var receiver = qint.new(3, 'R')
var ancillary = qint.new(1, 'A')

qc.write(0x0)

let label = 'genData'
qc.startlabel(label)
sender.ry(135, 0x2 | 0x4)
qc.disableDisplay(label)
qc.endlabel(label)

qc.nop()

label = 'InvQFT'
qc.startlabel(label)
sender.invQFT()
qc.endlabel(label)

qc.nop()
label = 'send'
qc.startlabel(label)
sender.exchange(receiver)
qc.endlabel(label)
qc.disableDisplay(label)
qc.nop()

label = 'QFT'
qc.startlabel(label)
receiver.QFT()
      
```

A
B

Circuit

Evolution View

D
C

INVQFT

QFT

A = R >= 8

High freq++

D-1
D-2
D-3
D-4

C-1
C-2

Variable State

- Probability
- Magnitude
- Entanglement

Phase

1. Visualize the unitary
2. Visualize the change of qubit states

Simulate your program with JanusQ



GHZ state Calibration Chip: dense_5 Compile Predict fidelity Simulate with noise Calibrate readout error

QuCode Editor

```
1 var num_qubits = 5
2 reset(num_qubits)
3 var control = qint.new(1, 'control')
4 var expand = qint.new(4, 'expand')
5
6 startlabel('init')
7 // initialize the control qubit
8 control.hadamard([0])
9 endlabel('init')
10
11 startlabel('entanglement')
12 // entangle with qubit_1
13 cnot([0, 1])
14
15 // entangle with qubit_2
16 cnot([1, 2])
17
18 // entangle with qubit_3
19 cnot([2, 3])
20
21 // entangle with qubit_4
22 cnot([3, 4])
23 endlabel('entanglement')
```

Original Circuit

Compiled Circuit

2. Compile your quantum circuit

Console

Noisy Result

Calibrated Result

3. Simulate your circuit with noise

4. Calibrate readout error in result

Outline of Presentation



- Tutorial Overview
- Background Knowledge
- Mathematical Model of Quantum Computing
- Janus (Taiyuan) Quantum Cloud Platform
- **Installing JanusQ**



浙江大學
ZHEJIANG UNIVERSITY

Installing JanusQ



<https://github.com/JanusQ/JanusQ>



Language: Python 3, C++

Available platforms: Linux, Mac, Windows

HyQSAT has not been tested on Windows.

Hardware requirements:

- Classical computer: ≥ 16 GB Memory, Intel i5 10 Gen

Hardware used in the experiment

- A server with two AMD EPYC 2.25GHz 64-core CPUs and 1.6TB DDR 5 memory is preferable.
- Superconducting, ion-trapped quantum computer and D-Wave quantum annealer.

Installation:

From Docker (recommend)

Data collected from the quantum hardware are put into the framework.

From Source code

Wheel is provided in <https://github.com/JanusQ/JanusQ/tree/main/dist>. But HyQSAT is disabled.

Install from Docker (Preferred)



Step 1. download docker-desktop

```
https://www.docker.com/products/docker-desktop/
```

Windows Subsystem for Linux (WSL) are required for windows.

Step 2. pull JanusQ image

```
docker pull janusq/janusq3:latest
```



Step 3. start image

```
docker run -itd -p 8888:22 -p 9999:23 --name tutorial janusq/janusq3
```

Step 4. Use SSH/VSCode/PyCharm to connect the docker

```
ssh root@localhost -p 8888
```

Docker page of JanusQ:

<https://hub.docker.com/r/janusq/janusq3>

Step 5. Or use Jupyter Lab to connect the docker

```
http://localhost:9999/lab
```

The password of the docker is “root”.

Install from Source Code



Step 1. clone the source code

```
git clone git@github.com:JanusQ/JanusQ.git
```

Step 2. install requirements (virtual environment is preferred)

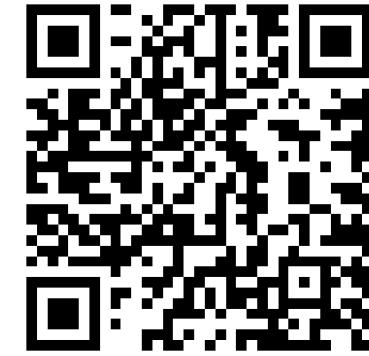
```
cd ./JanusQ  
conda create -n janusq python=3.10  
conda activate janusq  
pip install -r requirements.txt
```

Step 3. compile HyQSAT

```
cd ./janusq/application/hyqsat/solver  
cmake .  
make install
```

Step 3. install requirements for Choco-Q (Linux with CPU)

```
cd ./janusq/application/chocoq  
conda env create -f environment_linux_cpu.yml
```



Page of github:
<https://github.com/JanusQ/JanusQ>

Implemented based on C++.

*Change the dependency file
according to your platform*

Testing JanusQ



On “./JanusQ/examples/ipynb/1_1_install_janusq.ipynb”

Test QuCT

```
from janusq.analysis.vectorization import *
vec_model = RandomwalkModel()
vec_model.train()
```

Test MorphQPV

```
from janusq.verification.morphqpv import
MorphQC, Config

myconfig = Config()
myconfig.solver = 'sgd'
with MorphQC(config=myconfig) as morphQC:
    morphQC.add_tracepoint(0,1)
    morphQC.assume(0, IsPure())
    ...
```

Test QuFEM

```
from janusq.calibration.readout_mitigation.qufem
import EnumeratedProtocol

protocol = EnumeratedProtocol(4)
```

Test Choco-Q

```
from janusq.application.chocoq.chocoq.model import
LinearConstrainedBinaryOptimization as LcboModel

m = LcboModel()
x = m.addVars(5, name="x")
m.setObjective((x[0] + x[1])* x[3] + x[2], "max")
m.addConstr(x[0] + x[1] - x[2] == 0)
m.addConstr(x[2] + x[3] - x[4] == 1)
optimize = m.optimize()
```

Document and Example



On: [JanusQ/examples](#) or https://janusq.github.io/HPCA_2025_Tutorial/demo

- Getting Started
 - └ Install JanusQ
 - └ Submit to cloud
- QuCT
 - └ Vectorization model of QuCT
 - └ Fidelity prediction of QuCT on quantum simulators
 - └ Fidelity prediction of QuCT on real quantum devices
 - └ Fidelity optimization based on QuCT
 - └ Unitary decomposition based on QuCT
 - └ Extending framework for bug identification
- MorphQPV
 - └ Verify Quantum Program
- QuFEM
 - └ Readout calibration of QuFEM on quantum simulators
 - └ Readout calibration of QuFEM on real quantum devices
- Choco-Q
 - └ Constrained binary optimization with QAOA

Include 12 examples

Vectorization Model of QuCT

Author: Siwei Tan

Date: 7/4/2024

Based on paper "[QuCT: A Framework for Analyzing Quantum Circuit by Extracting Contextual and Topological Features](#)" (MICRO 2023)

In the current Noisy Intermediate-Scale Quantum era, quantum circuit analysis is an essential technique for designing high-performance quantum programs. Current analysis methods exhibit either accuracy limitations or high computational complexity for obtaining precise results. To reduce this tradeoff, we propose QuCT, a unified framework for extracting, analyzing, and optimizing quantum circuits. The main innovation of QuCT is to vectorize each gate with each element, quantitatively describing the degree of the interaction with neighboring gates. Extending from the vectorization model, we can develop multiple downstream models for fidelity prediction and unitary decomposition, etc. In this tutorial, we introduce the APIs of the vectorization model in QuCT.

```
In [1]:  
  
%matplotlib inline  
  
import os  
os.chdir("../")  
import logging  
logging.basicConfig(level=logging.WARN)  
  
import random  
import numpy as np  
  
from janusq.analysis.vectorization import RandomwalkModel, extract_device  
from janusq.objects.random_circuit import random_circuits, random_circuit  
from janusq.objects.backend import GridBackend
```

Vectorization Flow

Below is the workflow to vectorize a gate in the quantum circuit. The gate is vectorized by two steps. The first step runs random walks to extract circuit features in the neighbor of the gates. The second step uses a table comparison to generate the gate vector.



浙江大學
ZHEJIANG UNIVERSITY

Thanks for listening!