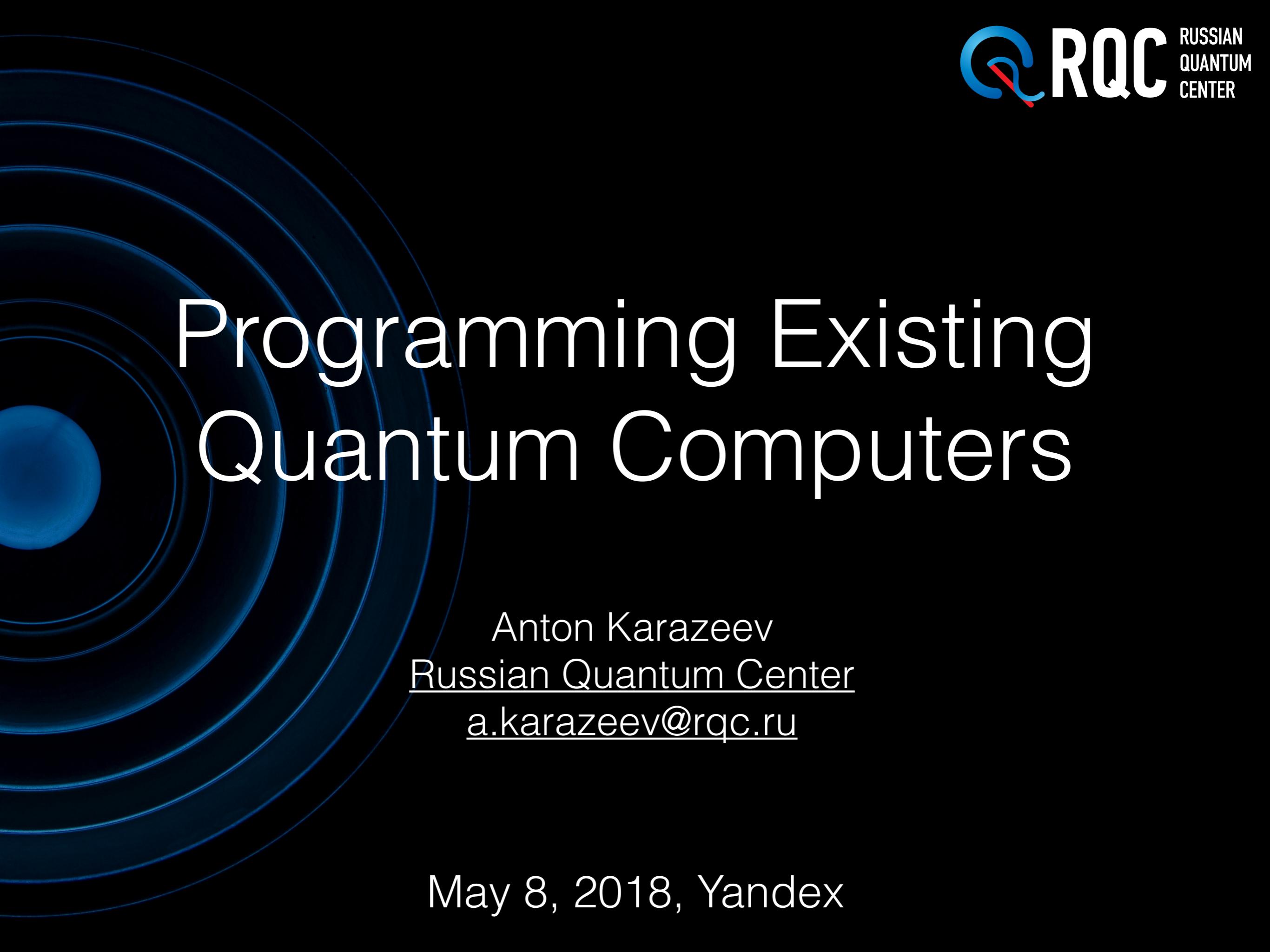


Programming Existing Quantum Computers



Anton Karazeev
Russian Quantum Center
a.karazeev@rqc.ru

May 8, 2018, Yandex

IBM Quantum Experience

▼ IBM Q 5.1 [ibmqx4] ACTIVE: USERS

Last Calibration: 2018-05-07 05:09:03
Fridge Temperature: 0.021 K

[More details](#)

	Q0	Q1	Q2	Q3	Q4
Frequency (GHz)	5.24	5.31	5.35	5.41	5.19
T1 (μs)	46.70	56.30	37.80	32.90	53.30
T2 (μs)	13.50	47.70	46.00	16.60	37.60
Gate error (10^{-3})	0.77	0.94	1.12	3.09	0.77
Readout error (10^{-2})	4.80	4.40	6.70	3.30	4.70
MultiQubit gate error (10^{-2})	2.21	2.85	11.23	4.35	
CX1_0	CX2_0	CX3_2	CX4_2		
CX2_1	CX3_4				
3.32	7.50				

► IBM Q 5 [ibmqx2] MAINTENANCE

Grover N=2 A=01 ↕ Add a description New Save Save as

[Switch to Qasm Editor](#) Backend: ibmqx2 i My Units: 18 i Experiment Units: 3 i Run Simulate

q[0] |0> —————
q[1] |0> — H —————
q[2] |0> — H — S — H —+— H —————
q[3] |0>
q[4] |0>

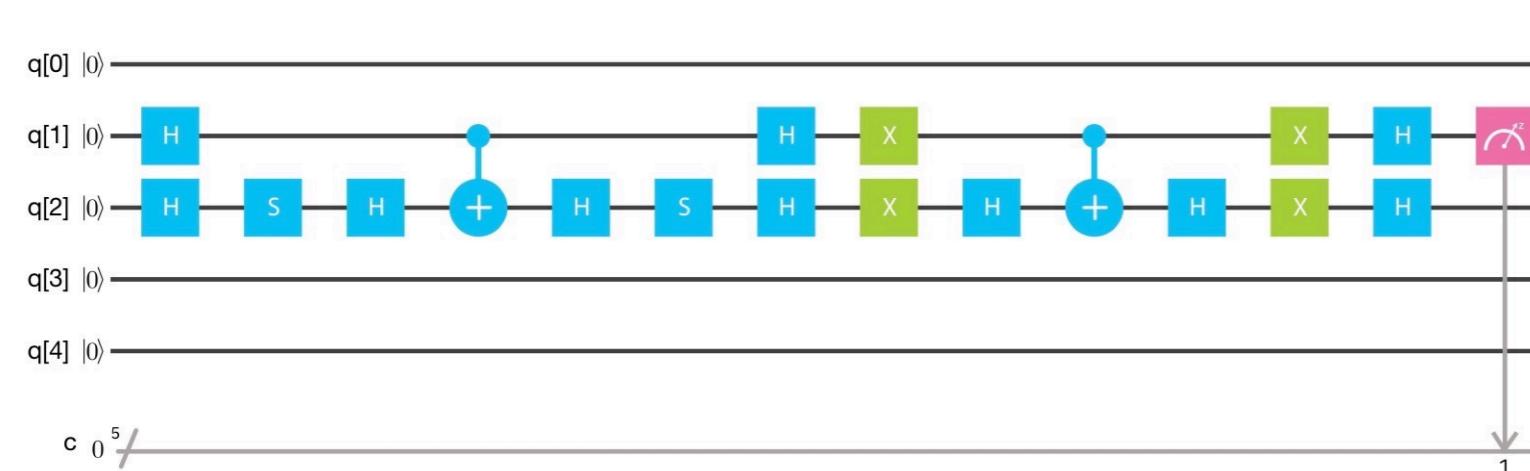
c 0 5 → 1

GATES i Advanced

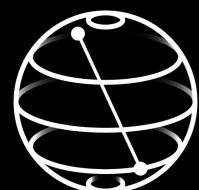
id	X	Y	Z	H
S	S^\dagger	+	T	T^\dagger

BARRIER

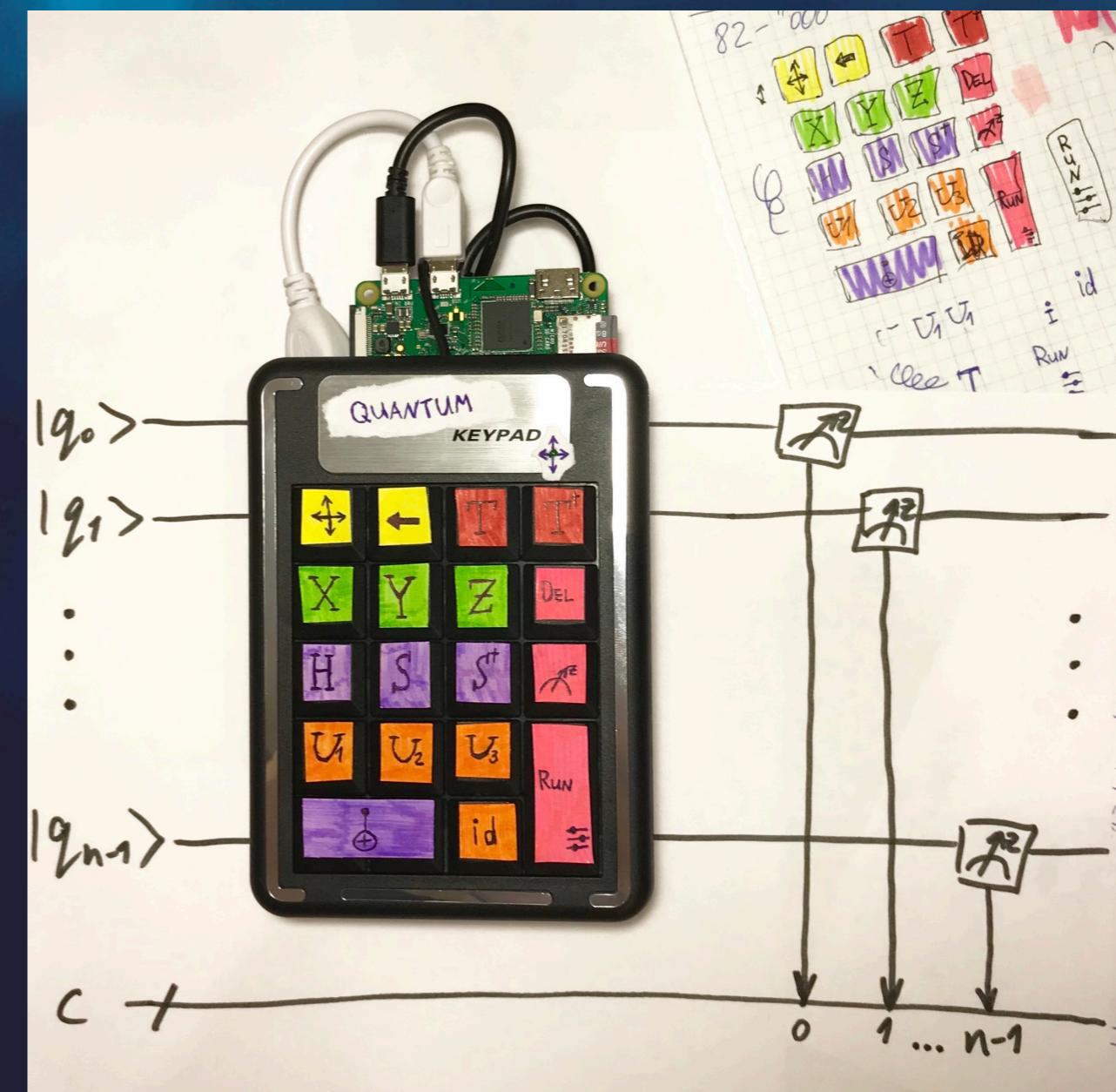
OPERATIONS



IBM Quantum Experience



<https://qiskit.org/modelq/>



<https://akarazeev.github.io>

IBM Quantum Experience

- Wide range of **quantum devices** with free access to users (5 and 16 qubits). IBM Q 20 is available only if you are a partner of IBM or a member of the **IBMQ Network**.
- **Quantum processor with 50 qubits** was announced in the end of 2017.
- QISKit (Quantum Information Software Kit) — open source SDK for working with **OpenQASM** and the **IBMQ quantum processors**.
- Create **quantum computing programs**, compile, and execute them online in a real quantum processors or simulators.

Rigetti

RIGETTI COMPUTING INTRODUCES

Forest^{1.3}

An API for quantum computing in the cloud.

NEW

19Q Processor

REQUEST UPGRADED ACCESS >

Get started for free

Full Name

Email address

EMAIL API KEY

Forest is a developer environment for **quantum programming**.

Forest provides free developer access for up to 26-qubits of our simulator the Quantum Virtual Machine™ and private access to our quantum hardware systems for select partners.



Watch Video 55:38



Open Source Software



Superconducting Quantum Processors



Example Algorithms



Python Development Tools

Rigetti

Quantum Virtual Machine

The QVM is a high performance simulation environment for developing and testing quantum programs.

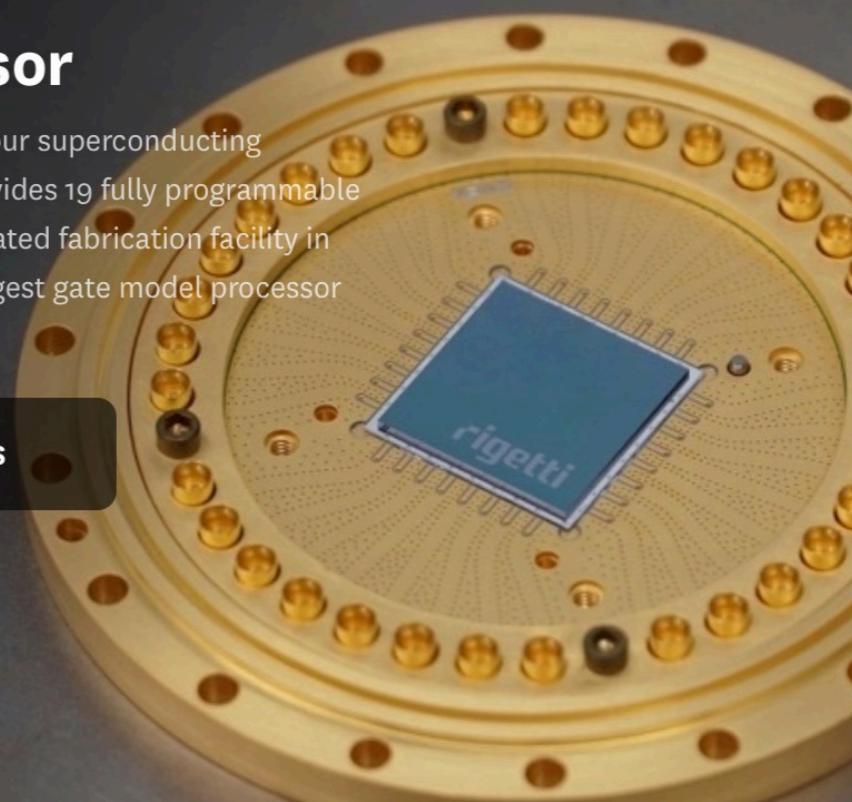
-  Free access to up to 26 simulated qubits
-  Customizable noise models
-  Private access for 30+ qubits



19Q Processor

The latest generation of our superconducting quantum processors provides 19 fully programmable qubits. Built in our dedicated fabrication facility in Fremont, CA, it is our largest gate model processor available ever.

[REQUEST ACCESS](#)



<https://www.rigetti.com/forest>

Rigetti

- Free access up to 26 **simulated qubits**.
- Access by request to **superconducting quantum processor** with 19 fully programmable qubits (19Q).
- Open source Python library pyQuil for constructing, analysing, and running **quantum programs**.
- Grove — a repository with implemented **quantum algorithms** using Forest API (including quantum Fourier transform, QAOA, phase estimations, etc.)



Strawberry Fields (by Xanadu)

STRAWBERRY FIELDS INTERACTIVE

Gates Settings Algorithms ?

One Mode ⓘ Two Mode ⓘ Measurement ⓘ

S D X Z P
F K V R

CX BS S
P HD

Clear All Editing is disabled on preset algorithms. Click 'Edit Preset' to create an editable copy. Edit Preset

$|a\rangle$ $|0\rangle$ $|0\rangle$

mode 0 mode 1 mode 0 mode 1

Add Outputs ⓘ Simulation Type ⓘ Cutoff Dimension ⓘ

Select an Output Fock (Tensorflow) - 5 + Go

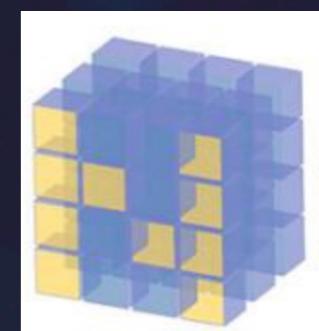
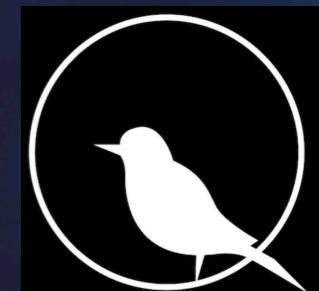
Blackbird Code × Fock States ×

<https://strawberryfields.ai>



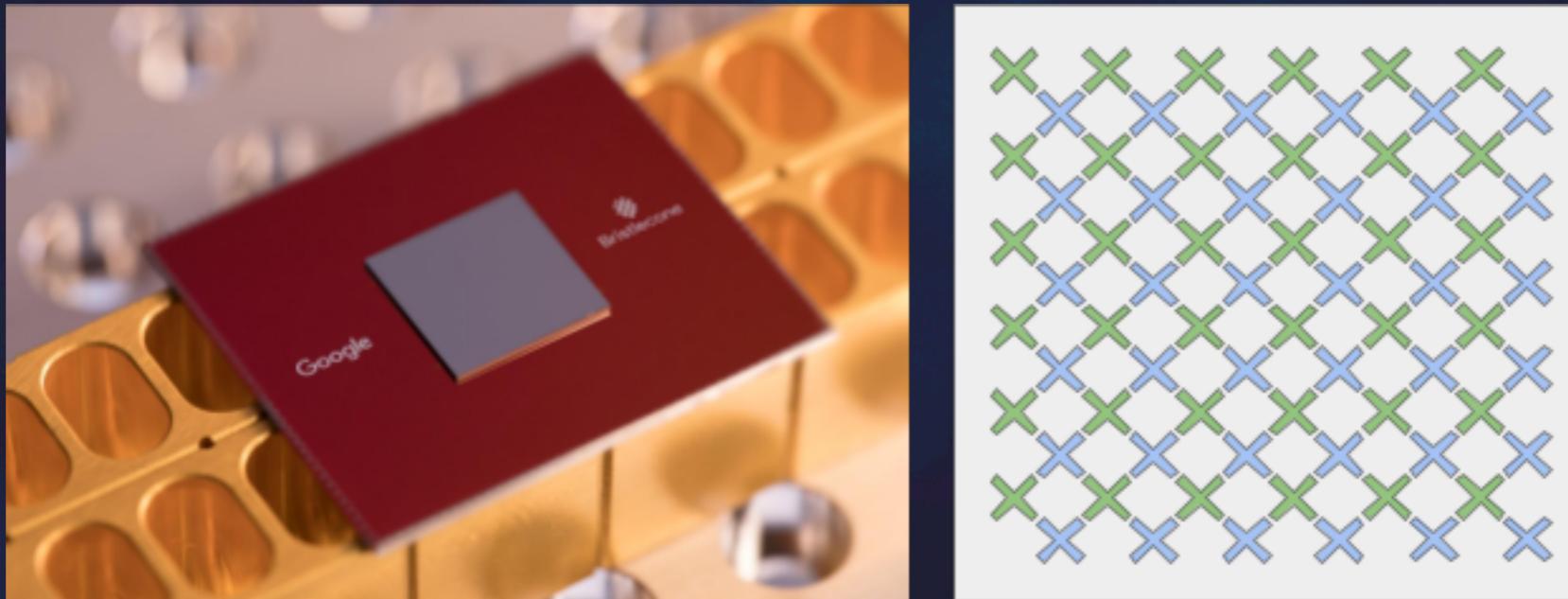
Strawberry Fields (by Xanadu)

- They developed a full stack software solution for simulating **quantum photonics** and **continuous variable** quantum computing.
- Integrated support for **TensorFlow** — creating a framework that combines the latest advances in **{deep, machine} learning** with **quantum computation**.
- Strawberry Fields includes a suite of **quantum simulators** implemented using **NumPy** and **TensorFlow** — these convert and optimise **Blackbird** code for classical simulation.
- Future releases will target **experimental backends**.



Google Quantum Research

- 72 qubits based on superconducting circuits in 2018.



Bristlecone processor. [Source](#)

Quantum companies

Company	Number of qubits	Access	Libraries/API	Description
IBM Q	50	Devices with 5 and 16 qubits, simulators with 20 and 32 qubits	QISKit Manuals are available	Actively trying to achieve quantum supremacy
Rigetti	19	By request to 19Q processor, free access up to 26 simulated qubits	Forest Library with implemented quantum algorithms called Grove	Quantum startup. Now they have quantum chips and all kinds of quantum software
D-Wave Systems	2000	You have to contact D-Wave engineers and collaborate with them	Few libraries are available here but it's hard to use them without help from D-Wave	The only company offering a quantum computer to marketplace
Google	72	-	-	Leaders by the numbers of qubits
Microsoft	-	-	Q# language and compiler	-
Xanadu	-	Strawberry Fields interface	Strawberry Fields Quantum programming language Blackbird	They are developing a quantum photonic processor

Series of seminars

A series of jupyter notebooks dedicated to introduction to Quantum Computing <http://rqc.ru/> Edit

quantum-computing russian-quantum-center seminars Manage topics

26 commits 2 branches 0 releases 1 contributor Apache-2.0

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

akarazeev	Fix probs	Latest commit d0c49c4 17 days ago
01_Introduction	Fix probs	17 days ago
02_IBM_QISKit	Fix latex	18 days ago
03_Rigetti_pyQuil	Fix X and I	17 days ago
img	Add img folder	18 days ago
.gitignore	Up	19 days ago
LICENSE	Rename LICENSE	19 days ago
README.md	Up README	18 days ago
requirements.txt	Add requirements	18 days ago

<https://github.com/RQC-QApp/Seminars>

Series of seminars

README.md

Введение в квантовые вычисления. Семинары



Содержание

- Введение:
 - i. [\[intro_to qc.ipynb\]](#)
 - Основные понятия из линейной алгебры
 - Принципы квантовой механики
- Вычисления с помощью QISKit'a и облачной платформы IBM Q Experience:
 - i. Настройка соединения с API сервисов IBMQX: [\[ibm_setup.md\]](#)
 - ii. [\[ibm_intro.ipynb\]](#)
 - Базовые операции, гейты
 - Представление результатов
- Вычисления с помощью pyQuil'a, Grove и облачной платформы Forest от Rigetti:
 - i. Настройка соединения с API сервисов Rigetti Forest: [\[rigetti_setup.md\]](#)
 - ii. [\[rigetti_intro.ipynb\]](#)
 - Базовые операции, гейты
 - Представление результатов

Зависимости

Для работы с семинарами необходимо наличие следующих Python-пакетов:

- NumPy >=1.13.3
- SciPy >=1.0.0
- pyQuil >=1.6.2
- QISKit >=0.4.9

<https://github.com/RQC-QApp/Seminars>

Series of seminars



- Anton Karazeev: a.karazeev@rqc.ru or t.me/akarazeev
- Based on [Introduction to Quantum Computing](#)
- The latest version of this notebook is available [here](#)

От классического бита к кубиту / From Bit to Qubit

Вероятностные биты как векторные пространства / Probabilistic Bits as Vector Spaces

Возможные результаты измерения бита представим в виде ортонормированных базисных векторов $\vec{0}$ и $\vec{1}$. И назовём их **исходами**.

Например, в двумерном пространстве базисные векторы можно представить следующим образом: $\vec{0} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ и $\vec{1} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

Выбор обусловлен следующим:

- Они **нормированы** (их длины равны единице):

$$(\vec{0}, \vec{0}) = \vec{0}^T \cdot \vec{0} = (1 \ 0) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 1 \cdot 1 + 0 \cdot 0 = 1 \rightarrow \text{length}(\vec{0}) = \sqrt{1} = 1$$

$$(\vec{1}, \vec{1}) = \vec{1}^T \cdot \vec{1} = (0 \ 1) \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 0 \cdot 0 + 1 \cdot 1 = 1 \rightarrow \text{length}(\vec{1}) = \sqrt{1} = 1$$

- И **ортогональны** (скалярное произведение равно 0):

$$(\vec{1}, \vec{0}) = (\vec{0}, \vec{1}) = \vec{0}^T \cdot \vec{1} = (1 \ 0) \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 1 \cdot 0 + 0 \cdot 1 = 0$$

Обозначения

\vec{v}^T означает **транспонирование** вектора \vec{v} .

Операция (\vec{u}, \vec{v}) называется операцией **скалярного произведения** векторов \vec{u} и \vec{v} .

Длиной вектора \vec{v} называется величина $|\vec{v}| = \sqrt{(\vec{v}, \vec{v})}$.

Примеры

Транспонирование: пусть $\vec{v} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, тогда $\vec{v}^T = (1 \ 0)$. В случае с матрицами: пусть $X = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, тогда $X^T = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$.

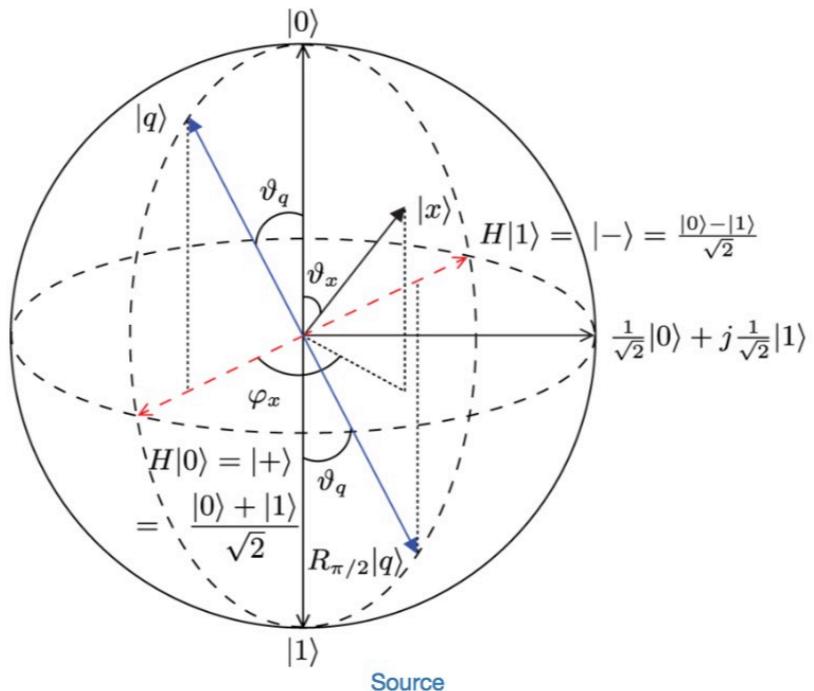
Скалярное произведение: пусть $\vec{u} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$, $\vec{v} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$, тогда $(\vec{u}, \vec{v}) = \vec{u}^T \cdot \vec{v} = (u_1 \ u_2) \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 \cdot v_1 + u_2 \cdot v_2$.

Эти исходы ($\vec{0}$ и $\vec{1}$) образуют двумерное векторное пространство, которое представляет **вероятностный бит**: $\vec{v} = a \cdot \vec{0} + b \cdot \vec{1}$, где a и b - вероятности того, что бит принимает значение 0 или 1 соответственно. Очевидно, что $a + b$ должно равняться 1.

Series of seminars

Сфера Блоха / Bloch Sphere

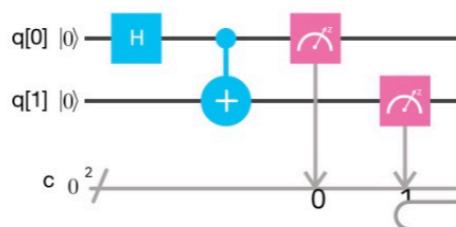
Этот переход к комплексным векторам означает следующее - вместо того, чтобы представлять вектор состояния на плоскости, мы будем представлять его на сфере.



Состояние из нескольких кубит может быть точно так же представлено, если взять тензорные произведения пространств и состояний. Таким образом, система из n кубит будет иметь 2^n возможных состояний.

Операции над кубитами / Qubits Operations

На примере IBM Q Experience [Composer](#)



Так выглядит простейшая квантовая цепь, состоящая из гейтов Адамара (H) и $CNOT$ (Гейт == Операция == Матрица). В конце стоят блоки, которые отвечают за **измерение** состояния кубит в базисе σ^z .

Series of seminars



• Anton Karazeev: a.karazeev@rqc.ru or t.me/akarazeev
• Based on [QISKit Tutorial](#)
• The latest version of this notebook is available [here](#)

IBM Q Experience

Введение

```
In [1]: from qiskit import QuantumProgram

# Создаём объект для квантовой программы.
qp = QuantumProgram()

# Теперь необходимо указать сколько нам понадобится квантовых регистров (кубит)
# и классических регистров.
#
# В обоих случаях указано 2. Помимо количества им присваиваются идентификаторы
# в рамках программы `qp`: "qr" и "cr" соответственно.
qr = qp.create_quantum_register("qr", 2)
cr = qp.create_classical_register("cr", 2)

# Создаём квантовую цепь с квантовыми регистрами `qr`, классическими регистрами `cr` и называем её "Bell".
qc = qp.create_circuit("Bell",[qr],[cr])

# Начинаем добавлять различные гейты (операции над кубитами).
#
# Синтаксис следующий:
# "<квантовая_цепь>.<гейт>(<квантовый/классический регистр, параметры, etc. - в зависимости от гейта>)".

# Гейт Адамара на нулевой кубит.
qc.h(qr[0])
# Controlled NOT (CNOT) гейт, который использует `qr[0]` кубит как управляющий, а кубит `qr[1]` как таргет.
qc.cx(qr[0], qr[1])
# Измерить нулевой кубит и записать измеренное значение в нулевой регистр.
qc.measure(qr[0], cr[0])
# Измерить `qr[1]` и записать в `cr[1]`.
qc.measure(qr[1], cr[1])

# Запустить выполнение программы "Bell" на бэкэнде (дефолтно программы запускаются на симуляторе
# "local_qasm_simulator").
result = qp.execute("Bell")

# Посмотреть на результаты программы в виде "{ '00': n1, '01': n2, ... }", где n1, n2, ... - число
# соответствующих исходов.
print(result.get_counts("Bell"))

{'11': 529, '00': 495}
```

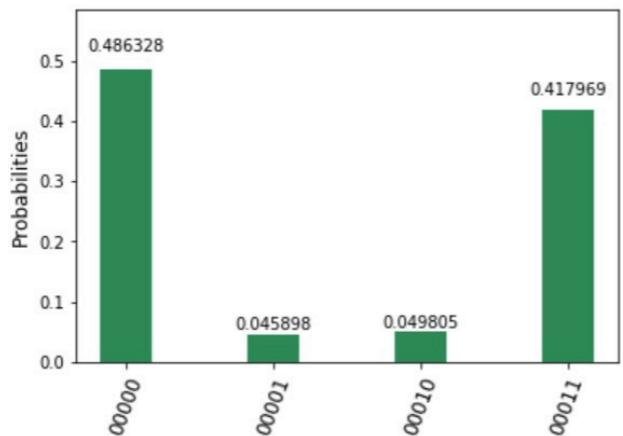
```
In [55]: %load_ext autoreload
%autoreload 2
import utils
```

Series of seminars

```
In [84]: # Проверяем статус программы.  
result_real.get_status()
```

```
Out[84]: 'COMPLETED'
```

```
In [86]: %%time  
# Гистограмма распределения исходов на настоящем квантовом устройстве.  
plot_histogram(result_real.get_counts('Bell'))
```



Про кредиты:

Experiment Units: 3 

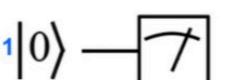
Experiments with 1024 shots or less = 3 units.

Experiments with more than 1024 = 5 units.

If you execute the experiment on a simulator, units are not consumed.

[Source](#)

Реализация квантовых цепей



Series of seminars



- Anton Karazeev: a.karazeev@rqc.ru or t.me/akarazeev
- Based on [Introduction to Quantum Computing](#)
- The latest version of this notebook is available [here](#)

Rigetti Forest

Введение

```
In [32]: from pyquil.quil import Program
from pyquil.gates import H, CNOT
from pyquil.api import QVMConnection

# Создаём объект для квантовой программы.
p = Program()

# "Мутирование" квантовой программы под действием оператора `H(0)` - гейт Адамара, который
# действует на нулевой кубит.
p.inst(H(0))
# Добавление CNOT гейта, который использует нулевой кубит как управляющий, а первый кубит как таргет.
p.inst(CNOT(0, 1))

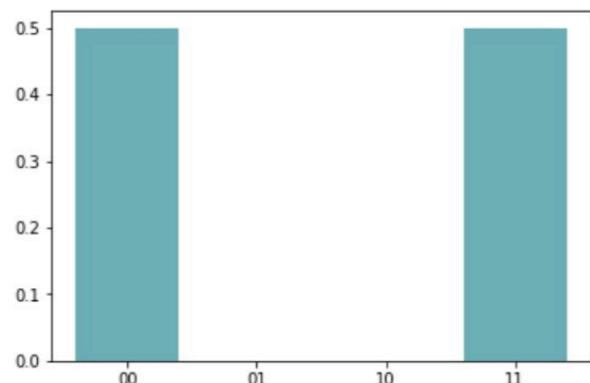
# Объект "квантовой виртуальной машины" - создаёт соединение с облачным бэкэндом.
qvm = QVMConnection()

# Запускаем квантовую программу `p` на бэкэнде.
result = qvm.wavefunction(p)

# Посмотреть на результаты программы в виде "{'00': p1, '01': p2, ...}", где p1, p2, ... - вероятности
# соответствующих исходов.
print(result.get_outcome_probs())

{'00': 0.4999999999999989, '01': 0.0, '10': 0.0, '11': 0.4999999999999989}
```

```
In [33]: # Построим гистограмму исходов.
result.plot()
```



Series of seminars

Измерение кубита

```
In [41]: # Указываем индексы классических регистров для записи.
classical_reg_index_0 = 0
classical_reg_index_1 = 1

# Указываем индексы квантовых регистров для измерения.
quantum_reg_index_0 = 0
quantum_reg_index_1 = 1

# Создаём простую программу, результаты измерений записываем в соответствующие регистры.
p = Program(I(0), X(1))

p.measure(quantum_reg_index_0, classical_reg_index_0)
p.measure(quantum_reg_index_1, classical_reg_index_1)

Out[41]: <pyquil.quil.Program at 0x10f005240>
```

```
In [42]: # Указываем классические регистры, состояние которых нам интересно после завершения программы.
classical_regs = [0, 1]

# Запускаем программу на квантовом симуляторе.
print(quantum_simulator.run(p, classical_regs, trials=4))

[[0, 1], [0, 1], [0, 1], [0, 1]]
```

Гейт Адамара

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

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

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle.$$

```
In [43]: from pyquil.gates import H

program = Program(H(0))
wavefunction = quantum_simulator.wavefunction(program)

print("H|0> = ", wavefunction)
print("Вероятности исходов:", wavefunction.get_outcome_probs())

H|0> = (0.7071067812+0j)|0> + (0.7071067812+0j)|1>
Вероятности исходов: {'0': 0.4999999999999989, '1': 0.4999999999999989}
```

Applications of QCs

- Quantum chemistry
- Simulations of physical systems
- Optimisation tasks such as MaxCut, k-SAT, etc.
- Grover's algorithm, Shor's algorithm
- Quantum PCA, SVM, etc.

What about Machine Learning algorithms?

- Quantum Neural Network (Hopfield Network)
- Unsupervised Machine Learning (Clustering)
- Quantum Classification (MNIST dataset)
- Quantum Approximate Optimisation Algorithm (QAOA)
- Restricted Boltzmann Machine: quantum training,
quantum state tomography

Quantum Machine Learning

Classical ML to classical problems	Classical ML to quantum problems	Quantum ML to classical problems	Quantum ML to quantum problems
	<ol style="list-style-type: none"> 1. Classical RBM for Quantum state tomography 	<ol style="list-style-type: none"> 1. Quantum RBM for MaxCut problem (using QAOA for RBM training) 	
We don't deal with such cases	<ol style="list-style-type: none"> 2. Identifying phases of matter using CNNs 	<ol style="list-style-type: none"> 2. Training of RBM on D-Wave processors required less iterations to achieve high accuracy of MNIST dataset classification 	We have ideas concerning such cases

Design tool flows in Classical and Quantum computers

a 1950s computing

Assembly language
(low-level) programs

Relay circuits and
discrete wires

Design tool flows in Classical and Quantum computers

a 1950s computing

b Classical computing today

Assembly language
(low-level) programs

Relay circuits and
discrete wires

Algorithms

High-level languages

Compiler

Classical architecture
(memory, arithmetic
operations, control
operations, communication)

Hardware building
blocks: gates, bits

VLSI circuits

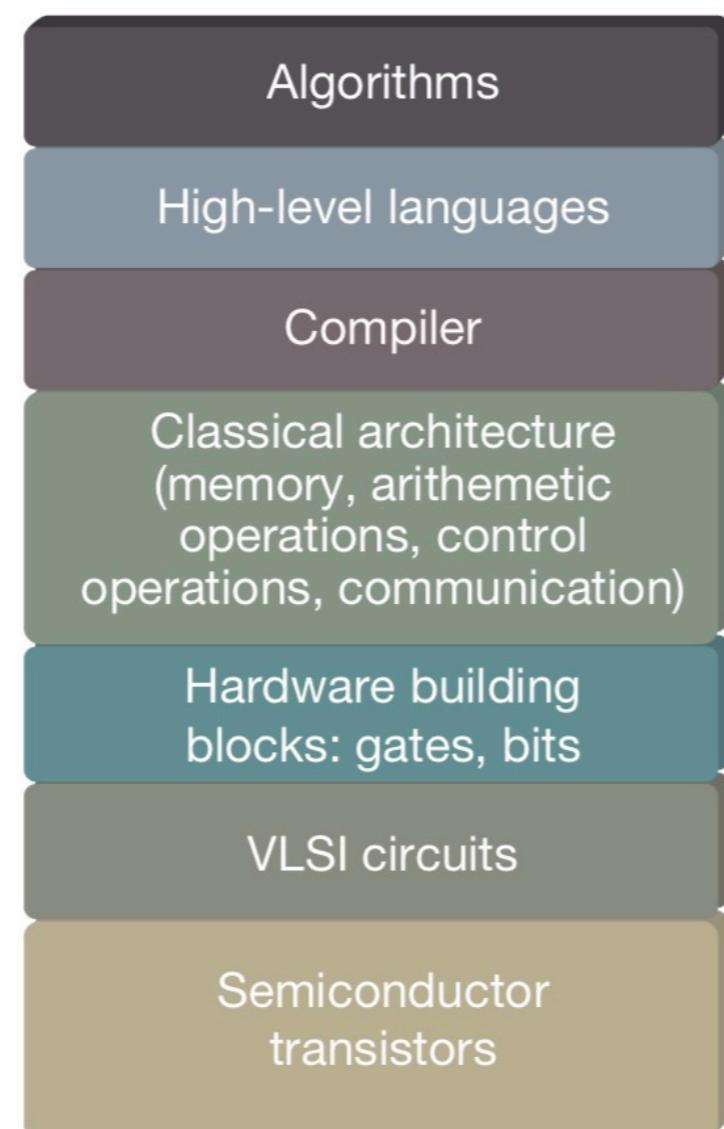
Semiconductor
transistors

Design tool flows in Classical and Quantum computers

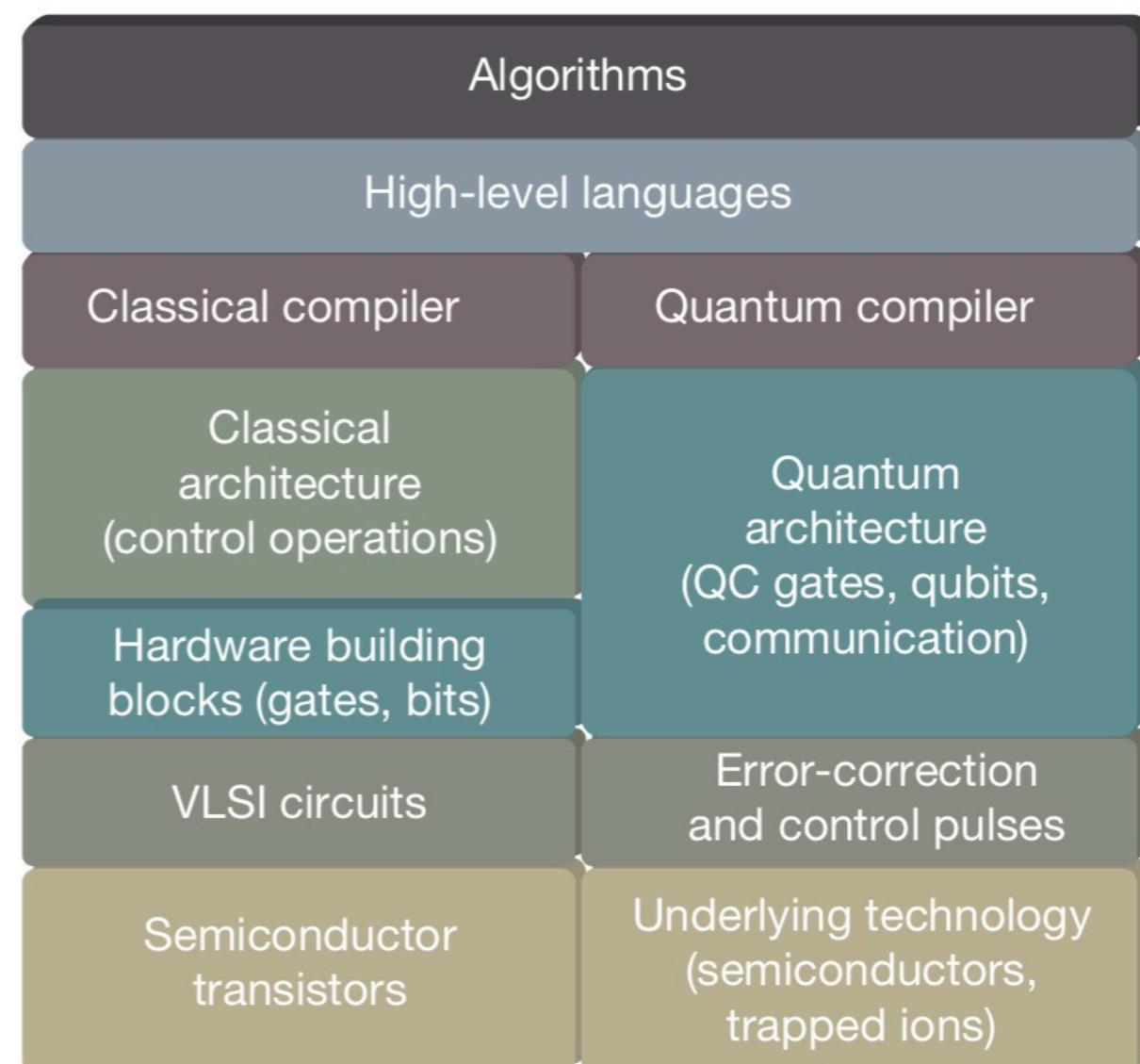
a 1950s computing



b Classical computing today



c Quantum computing



“The best way to predict the future is to create it.”

– Abraham Lincoln

Anton Karazeev
Russian Quantum Center
a.karazeev@rqc.ru