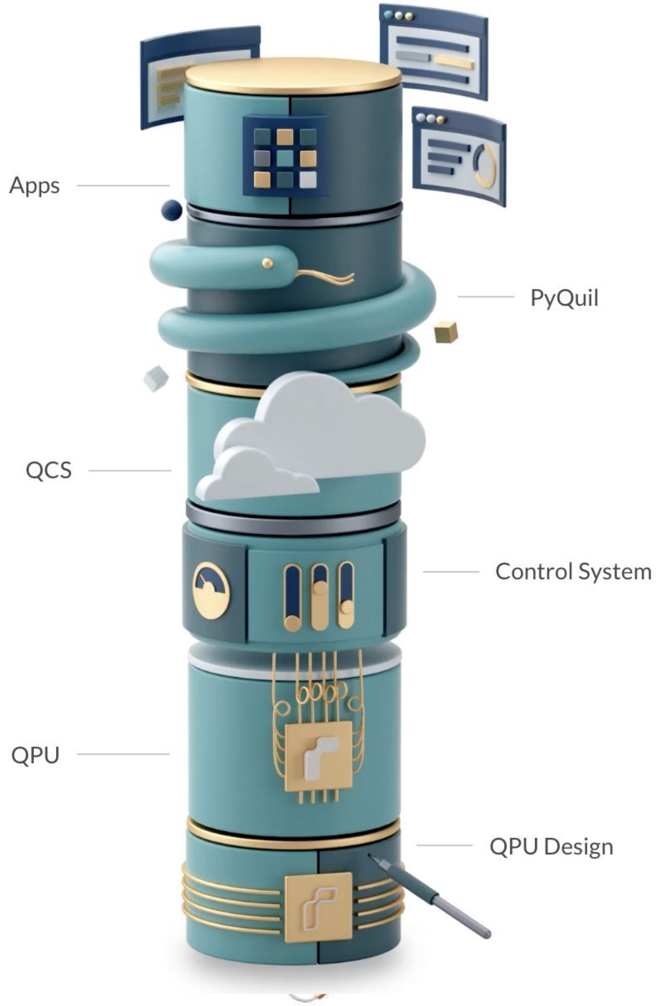
Quantum computing with near term devices

CERN - Quantum Computing for High Energy Physics Workshop
November 5, 2018

Will Zeng

rigetti



rigetti

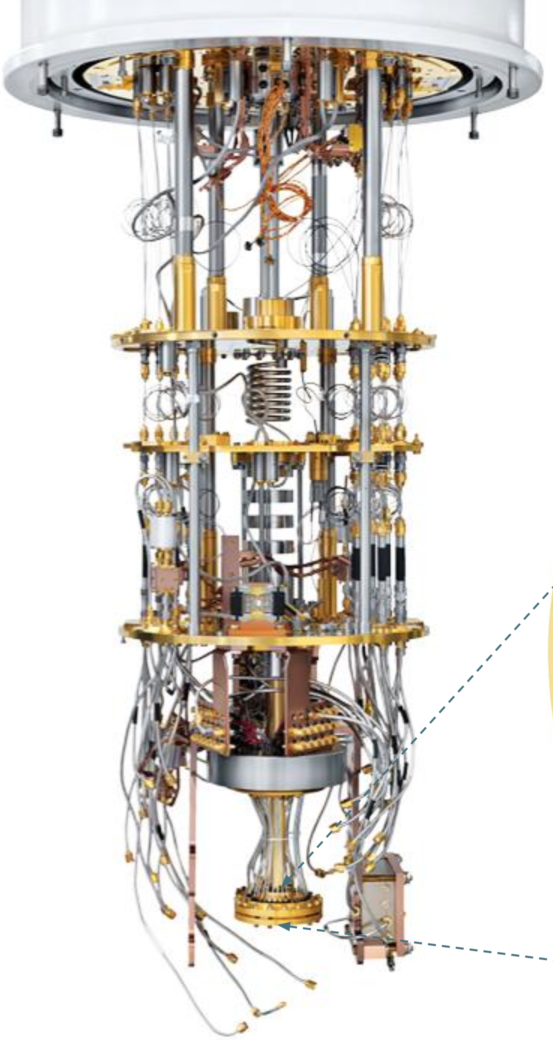
The world's first **full-stack quantum computing** company.

8-qubit and 19-qubit QPUs released on our cloud platform in 2017

100+ employees w/ $119M raised

Home of Fab-1, the world's first commercial quantum integrated circuit fab

Located in Berkeley, Calif. (R&D Lab) and Fremont, Calif.



The first quantum processors are here today

> **Superconducting processors** operating at 10mK

> Compute w/ individual microwave photons

> **New programming model** w/ potential for huge linear algebra

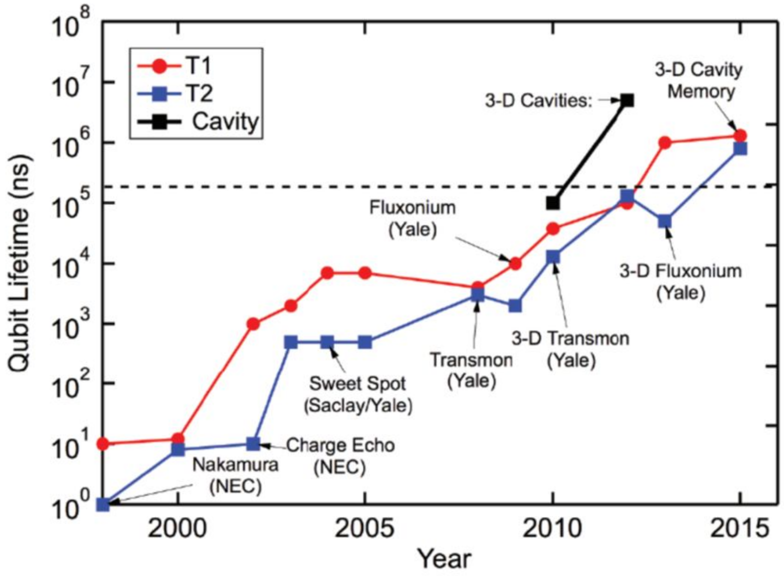
> Need to **improve both quantum memory size and performance**

Why now?

NISQ Hardware
+
Hybrid Software

Superconducting qubits have now gotten good enough to scale

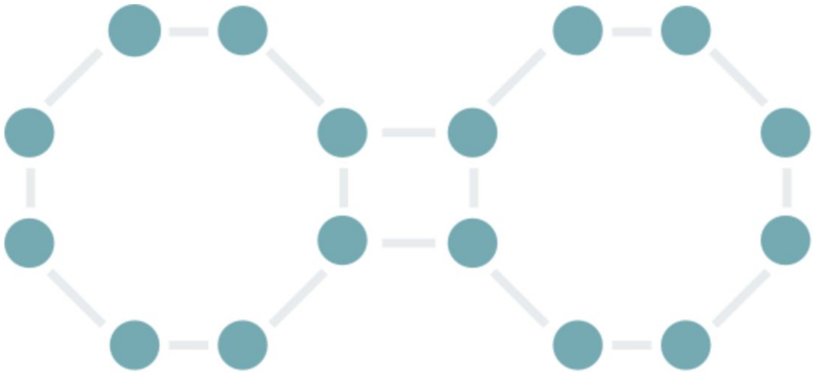
Superconducting qubit performance has increased by > 10^6 in the last 15 years



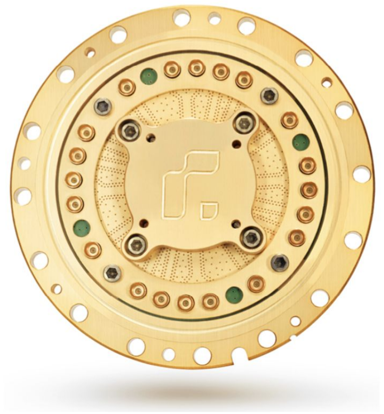
"Schoelkopf's Law"

Towards 128Q

Rigetti is building towards a 128 qubit system by scaling out a tileable lattice of qubits.



The new 128-qubit chip is based on a scalable 16-qubit form factor.



1994

Robust hybrid algorithms can run on smaller processors

1992-4
First Quantum Algorithms w/ Exponential Speedup
(Deutsch-Jozsa, Shor's Factoring, Discrete Log, ...)

1996
First Quantum Database Search Algorithm (Grover's)

2007
Quantum Linear Equation Solving (Harrow, Hassidim, Lloyd)

2008
Quantum Algorithms for SVM's & Principal Component Analysis

These algorithms require
Big, Perfect Quantum Computers

> 10,000,000 qubits for Shor's algorithms
to factor a 2048 bit number

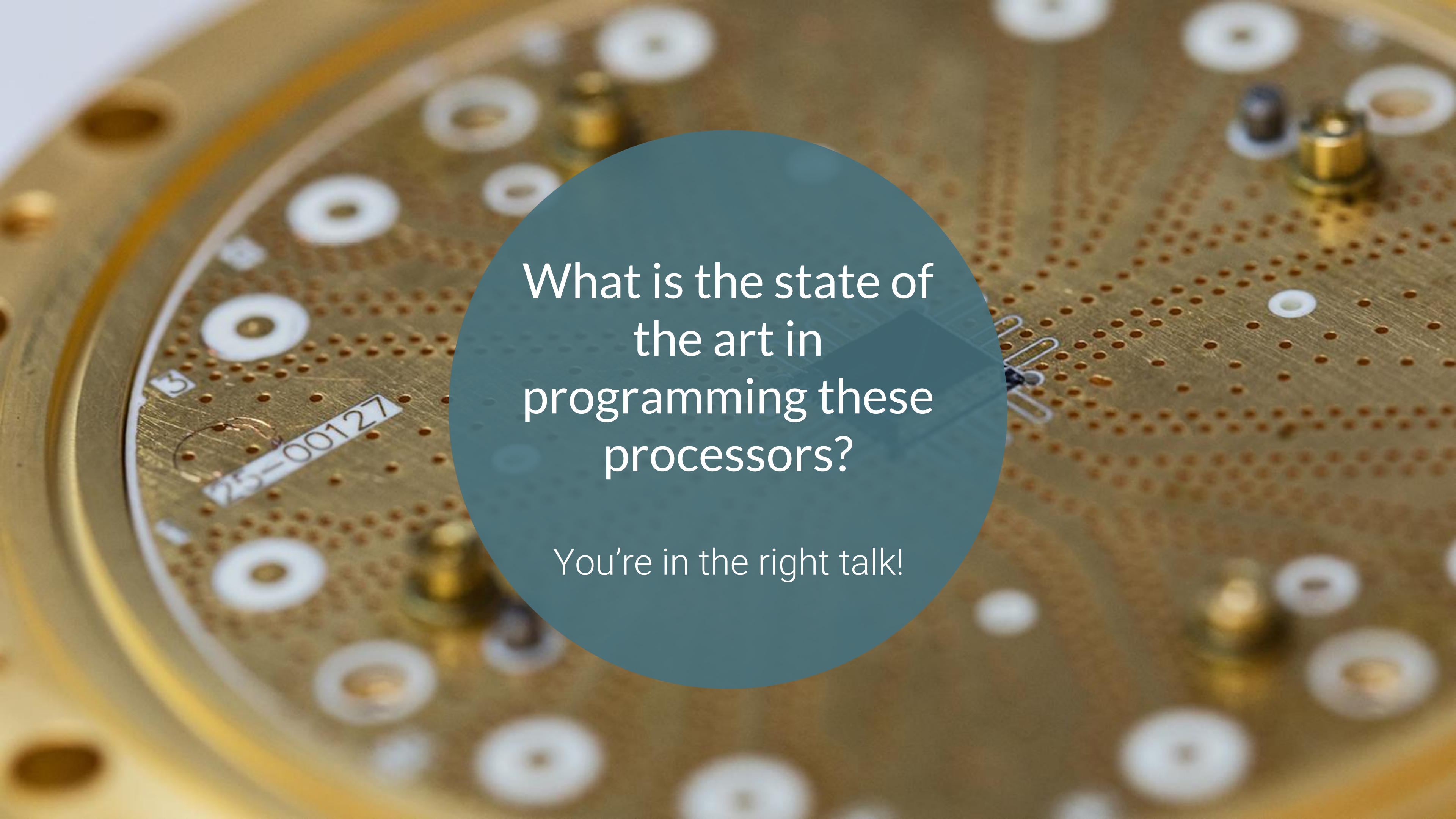
2013
Practical Quantum Chemistry Algorithms (VQE)

2016
Practical Quantum Optimization Algorithms (QAOA)
Simulations on Near-term Quantum Supremacy

Hybrid quantum/classical algorithms

Noise Robust, empirical speedups

TODAY



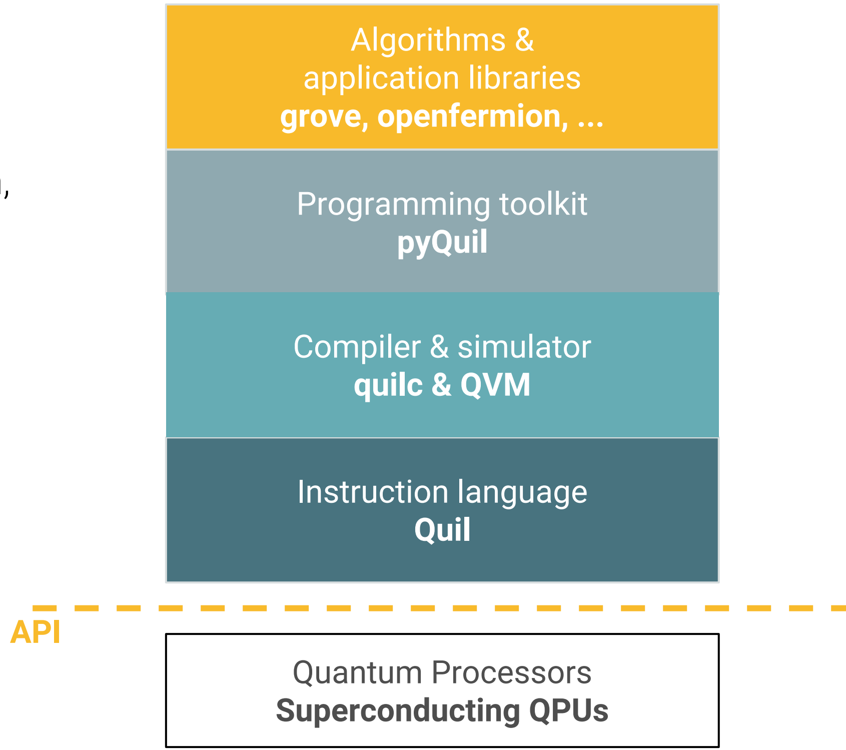
What is the state of the art in programming these processors?

You’re in the right talk!

This talk: Programming Rigetti Quantum Computers

1. The Quil programming model
2. PyQuil: Wavefunction, QuantumComputer, Compilation, Binary Patching
3. What’s next!

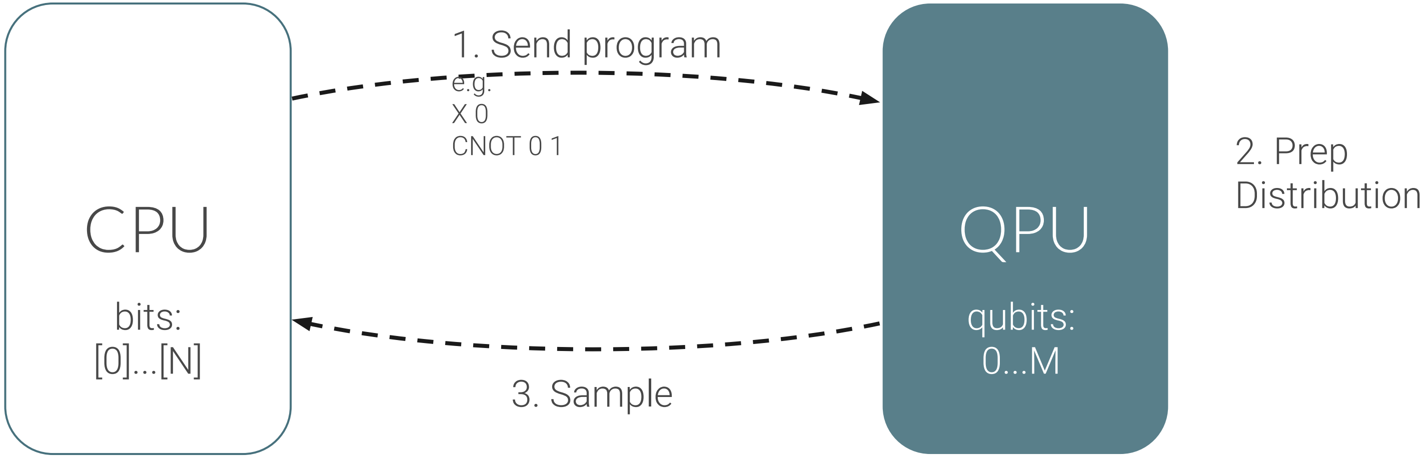
rigetti Forest SDK



Integrated quantum/classical computing

A motivation for Quil

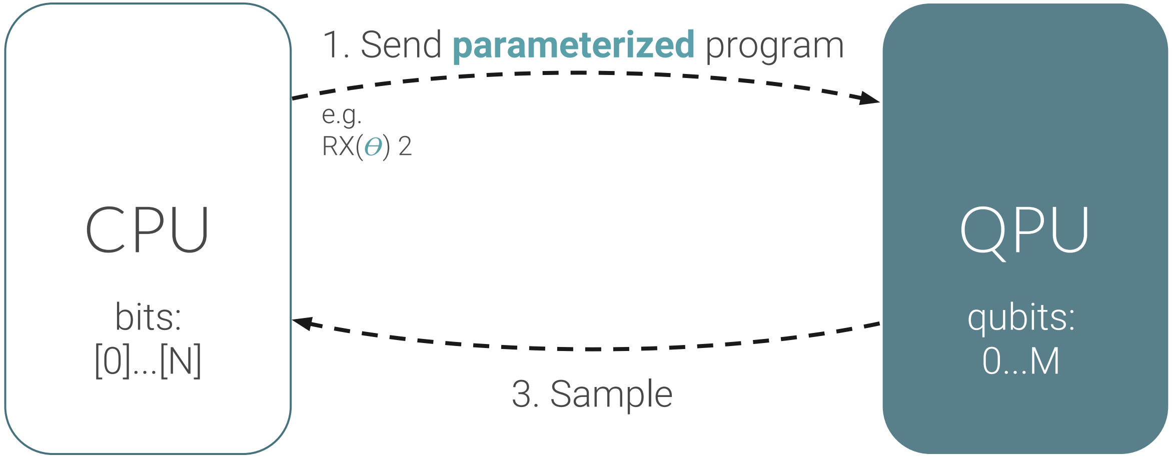
Quantum programming is preparing and sampling from complicated distributions

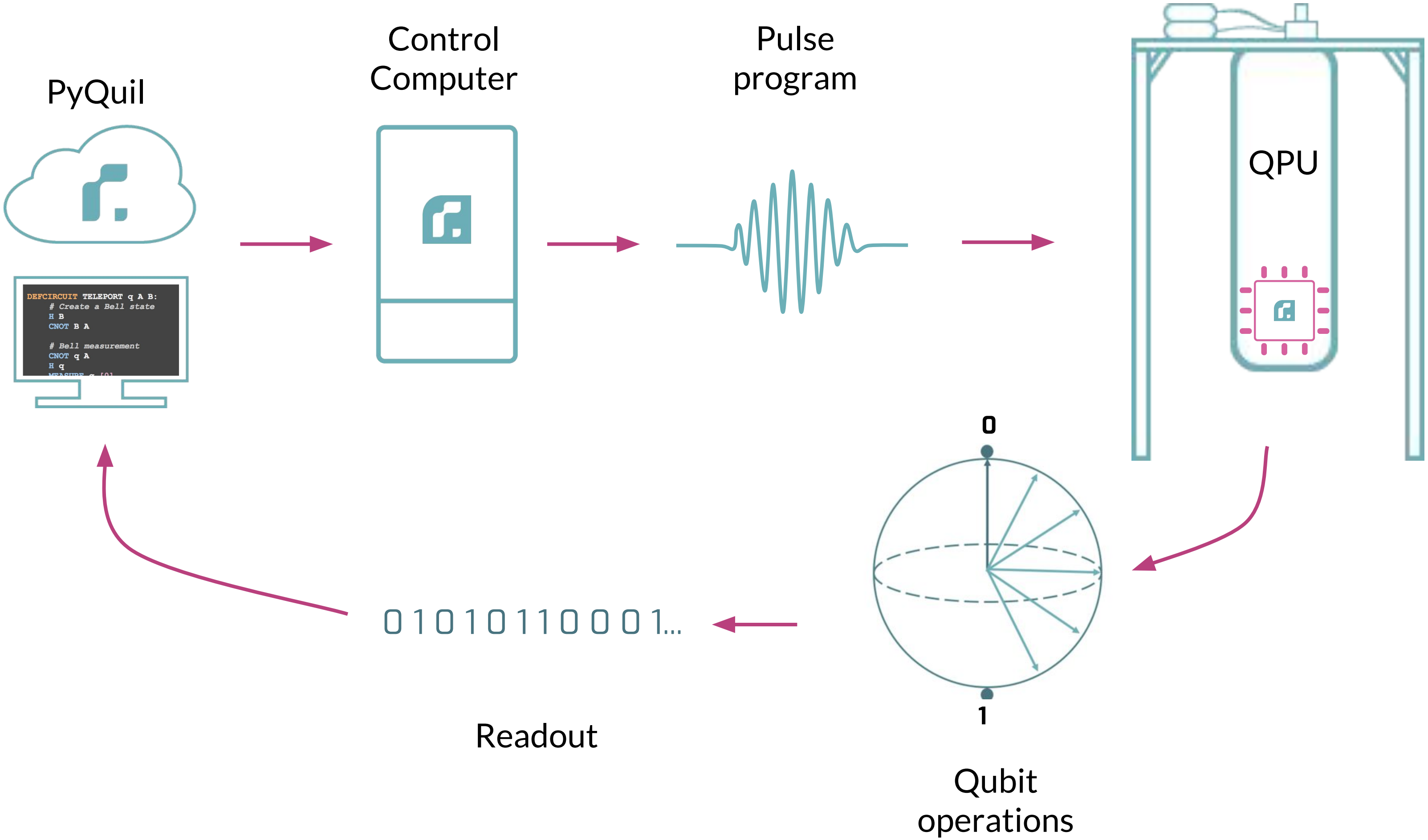


Integrated quantum/classical computing

A motivation for Quil

We parameterize and ***learn*** the quantum program to make it more robust




PyQuil
DEFCIRCUIT TELEPORT q A B:
Create a Bell state
H B
CNOT B A
Bell measurement
CNOT q A
H q
Control
Computer
Pulse
program
QPU
0
1
0 1 0 1 0 1 1 0 0 0 1...
Readout
Qubit
operations

The Quil Programming Model

Targets a **Quantum Abstract Machine (QAM)** with a syntax for representing state transitions

Ψ: Quantum state (qubits) → quantum instructions

C: Classical state (bits) → classical and measurement instructions

κ: Execution state (program) → control instructions (e.g., jumps)

```
# Quil Example
H 3
MEASURE 3 [4]
JUMP-WHEN @END [5]
.
.
.
```

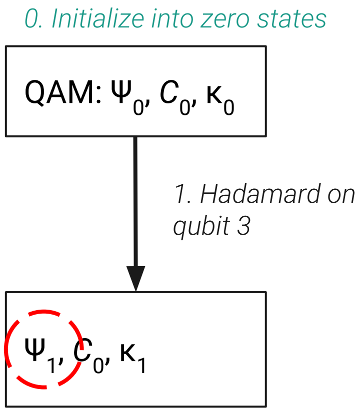
The Quil Programming Model

Targets a **Quantum Abstract Machine (QAM)** with a syntax for representing state transitions

Ψ: Quantum state (qubits) → quantum instructions

C: Classical state (bits) → classical and measurement instructions

κ: Execution state (program) → control instructions (e.g., jumps)




```
# Quil Example
H 3
MEASURE 3 [4]
JUMP-WHEN @END [5]
.
.
.
```

The Quil Programming Model

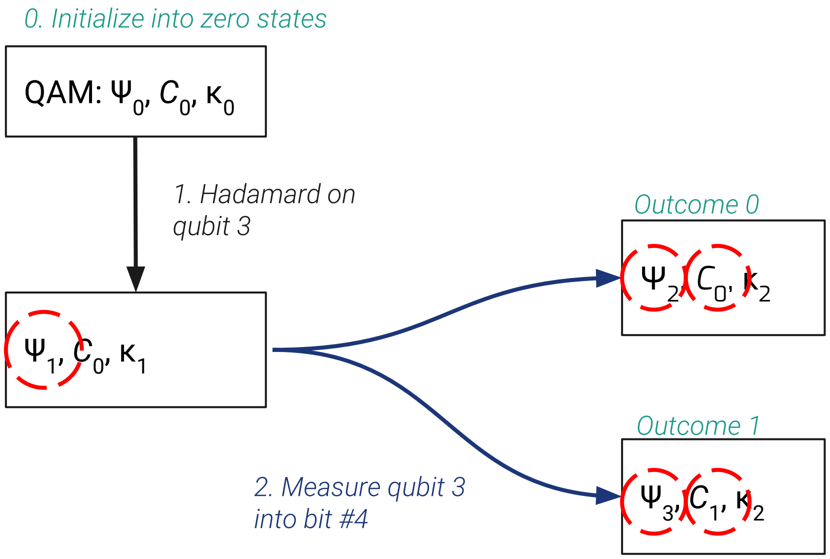
Targets a **Quantum Abstract Machine (QAM)** with a syntax for representing state transitions

Ψ: Quantum state (qubits) → quantum instructions

C: Classical state (bits) → classical and measurement instructions

κ: Execution state (program) → control instructions (e.g., jumps)

```
# Quil Example
H 3
MEASURE 3 [4]
JUMP-WHEN @END [5]
.
.
.
```



The Quil Programming Model

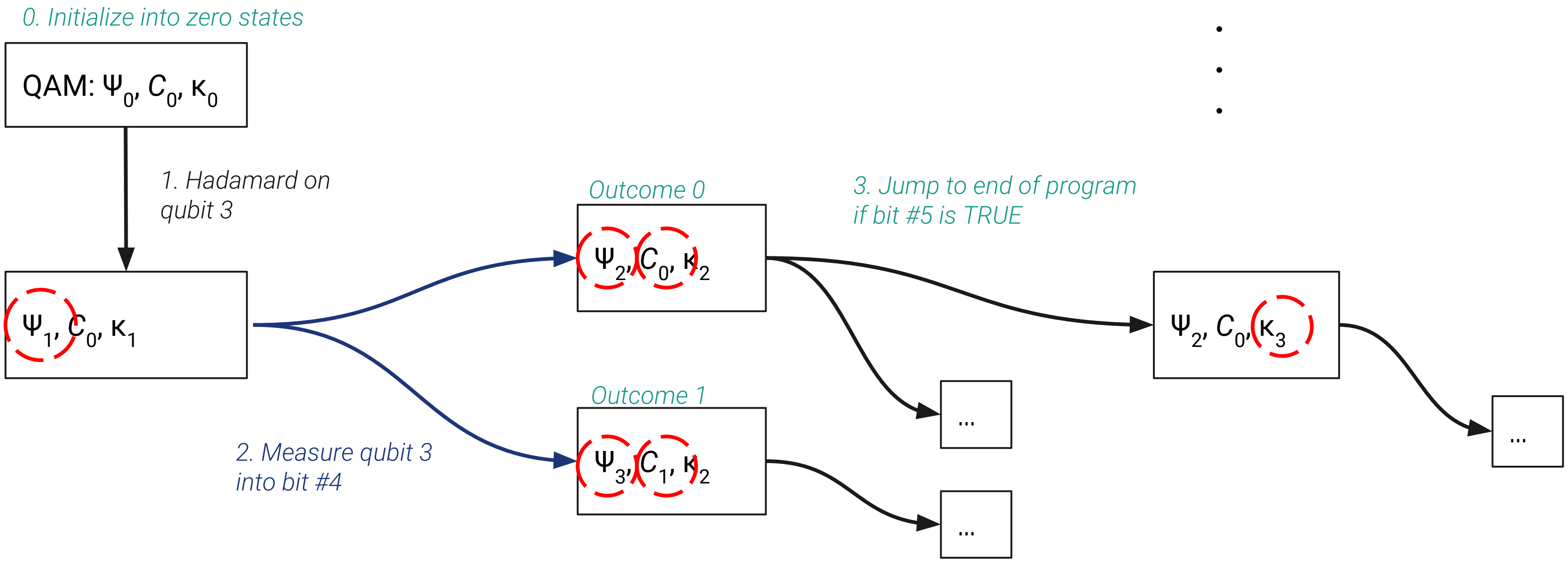
Targets a **Quantum Abstract Machine (QAM)** with a syntax for representing state transitions

Ψ: Quantum state (qubits) → quantum instructions

C: Classical state (bits) → classical and measurement instructions

κ: Execution state (program) → control instructions (e.g., jumps)

```
# Quil Example
H 3
MEASURE 3 [4]
JUMP-WHEN @END [5]
.
.
.
```



A Python library for quantum programming using Quil. http://forest.rigetti.com

Edit

forest | quil | quantum-computing | quantum | rigetti-forest | Manage topics

587 commits | 4 branches | 32 releases | 47 contributors | Apache-2.0

pyQuil

stable

Search docs

- Installation and Getting Started
- Forest 2.0: Migration Guide
- Programs and Gates
- The Quantum Virtual Machine (QVM)
- The Wavefunction Simulator
- The Quil Compiler
- Noise and Quantum Computation
- Advanced Usage
- Exercises
- Source Code Documentation
- Changelog

Docs » Welcome to the Docs for the Forest SDK!

Edit on GitHub

Welcome to the Docs for the Forest SDK!

The Rigetti Forest Software Development Kit includes pyQuil, the Rigetti Quil Compiler (quilc), and the Quantum Virtual Machine (qvm).

Longtime users of Rigetti Forest will notice a few changes. First, the SDK now contains a downloadable compiler and a QVM. Second, the SDK contains pyQuil 2.0, with significant updates to previous versions. As a result, programs written using previous versions of the Forest toolkit will need to be updated to pyQuil 2.0 to be compatible with the QVM or compiler.

After installing the SDK and updating pyQuil in Installation and Getting Started, see Forest 2.0: Migration Guide to get caught up on what's new!

Quantum Cloud Services will provide users with a dedicated Quantum Machine Image, which will come prepackaged with the Forest SDK. We're releasing a Preview to the Forest SDK now, so

pyQuil

Quantum Programming and Interfacing with Rigetti devices

pyQuil is:

1. A library with functions to easily generate quil programs
2. Interface to quilc & the QVM.
3. Contains a circuit simulator
4. Objects for controlling execution of quil programs: QPU or QVM.

Main Objects

`QuantumComputer`

QPU or Simulator

Compilation mode

Noise modeling for simulator

`Program`

Gates

Generators of Gates (PauliTerm)

Composition

`list_quantum_computers`

Getting information about Live Chips

rigetti Forest SDK

Algorithms & application libraries
grove, openfermion, ...

Programming toolkit
pyQuil

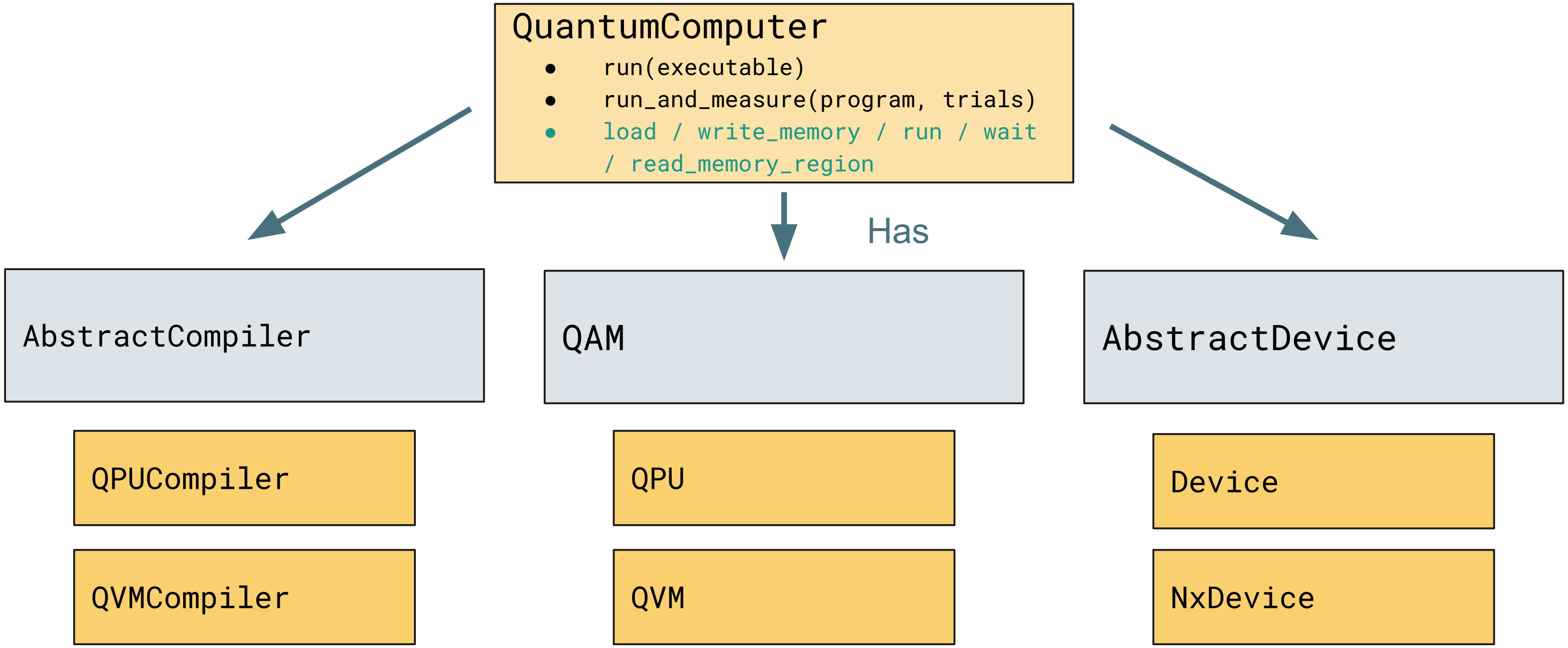
Compiler & simulator
quilc & QVM

Instruction language
Quil

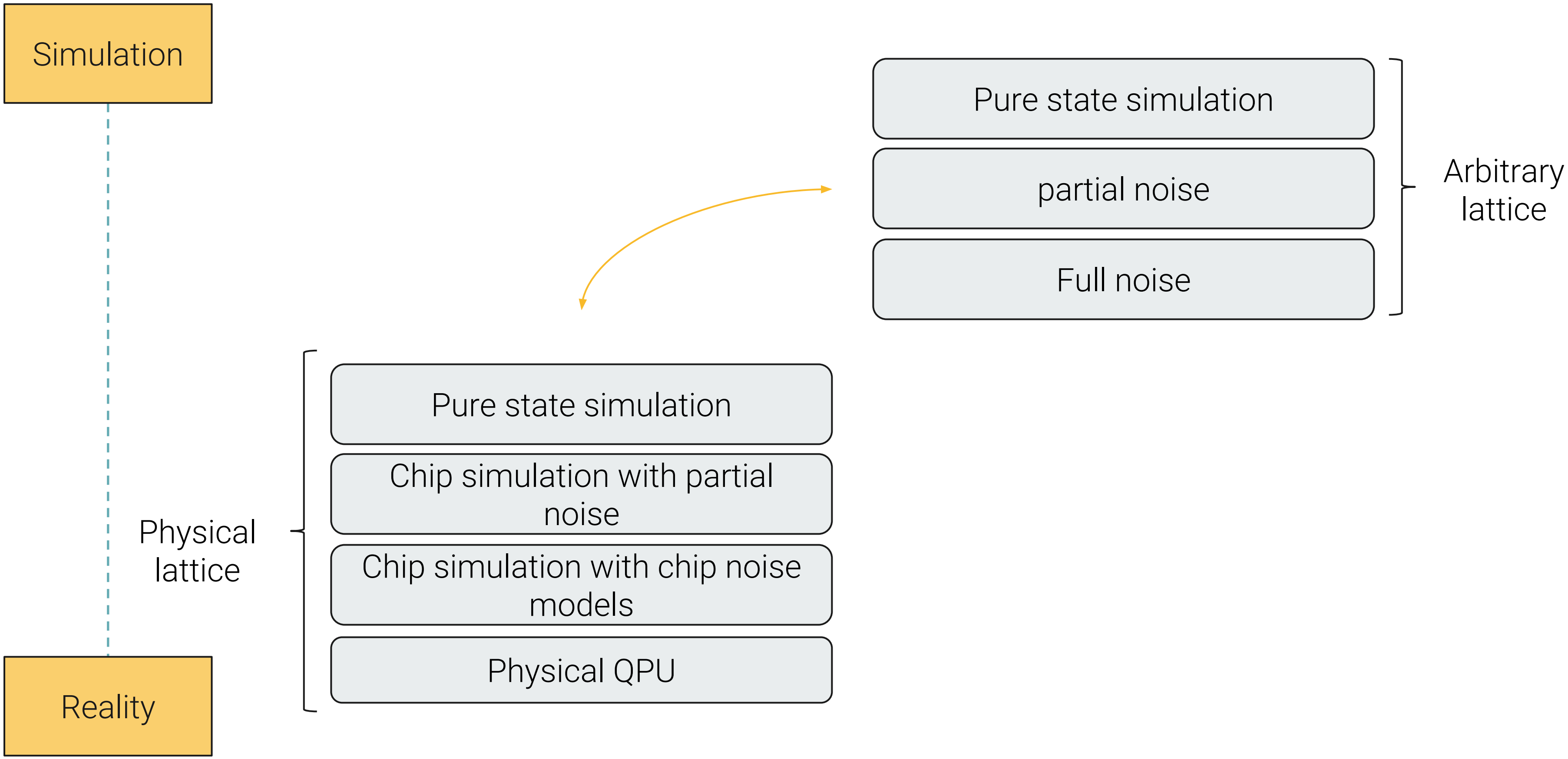
API

Quantum Processors
Superconducting QPUs

The `QuantumComputer` object



The `QuantumComputer` object: A hierarchy of realism



pyQuil

The `QuantumComputer` object: Using the WavefunctionSimulator

Simulation

Reality

```python
from pyquil import Program
from pyquil.gates import *
from pyquil.api import WavefunctionSimulator

def ghz_state(qubits):
    """Create a GHZ state on the given list of qubits by applying a Hadamard gate to the
   first qubit followed by a chain of CNOTs
    """
    program = Program()
    program += H(qubits[0])
    for q1, q2 in zip(qubits, qubits[1:]):
        program += CNOT(q1, q2)
    return program

program = ghz_state(qubits=[0, 1, 2])
print(program)

wfn = WavefunctionSimulator().wavefunction(program)
print(wfn)  # (0.7071067812+0j)|000> + (0.7071067812+0j)|111>
```

pyQuil

The `QuantumComputer` object: Using the WavefunctionSimulator

Simulation

Reality

```
from pyquil import Program
from pyquil.gates import *
from pyquil.api import WavefunctionSimulator

def ghz_state(qubits):
    """Create a GHZ state on the given list of qubits by applying a Hadamard gate to the
   first qubit followed by a chain of CNOTs
    """
    program = Program()
    program += H(qubits[0])
    for q1, q2 in zip(qubits, qubits[1:]):
        program += CNOT(q1, q2)
    return program

program = ghz_state(qubits=[0, 1, 2])
print(program)

wfn = WavefunctionSimulator().wavefunction(program)
print(wfn)  # (0.7071067812+0j)|000> + (0.7071067812+0j)|111>
```

The QuantumComputer object: Using the WavefunctionSimulator

Simulation

Reality

```
from pyquil import Program
from pyquil.gates import *
from pyquil.api import WavefunctionSimulator

def ghz_state(qubits):
    """Create a GHZ state on the given list of qubits by applying a Hadamard gate to the
   first qubit followed by a chain of CNOTs
    """
    program = Program()
    program += H(qubits[0])
    for q1, q2 in zip(qubits, qubits[1:]):
        program += CNOT(q1, q2)
    return program

program = ghz_state(qubits=[0, 1, 2])
print(program)

wfn = WavefunctionSimulator().wavefunction(program)
print(wfn)  # (0.7071067812+0j)|000> + (0.7071067812+0j)|111>
```

pyQuil

The `QuantumComputer` object: A hierarchy of realism

Simulation

Reality

```
from pyquil import get_qc

qc = get_qc('3q-qvm')  # 3-qubit qvm (fully connected lattice of qubits)

qc = get_qc('20q-qvm')  # 20-qubit qvm (fully connected lattice of qubits)

qc = get_qc('20q-noisy-qvm')  # 20-qubit qvm (fully connected lattice of qubits)

qc = get_qc('Apsen-xxx-noisy-qvm')  # Aspen topology simulated with chip noise

qc = get_qc('aspen-xx')  # runs on the QPU
```

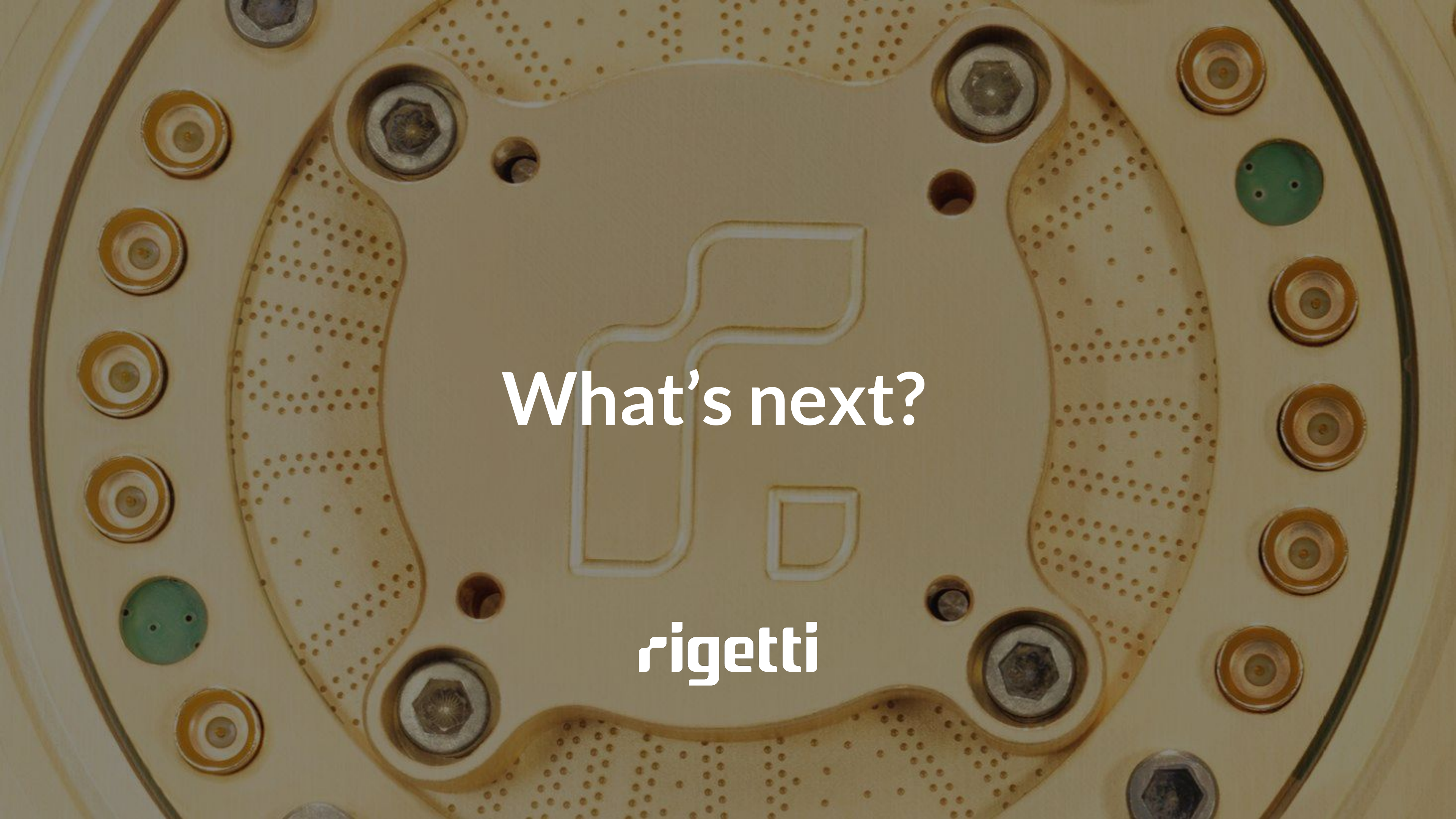
pyQuil

The `QuantumComputer` object: A hierarchy of realism -custom noise

Simulation

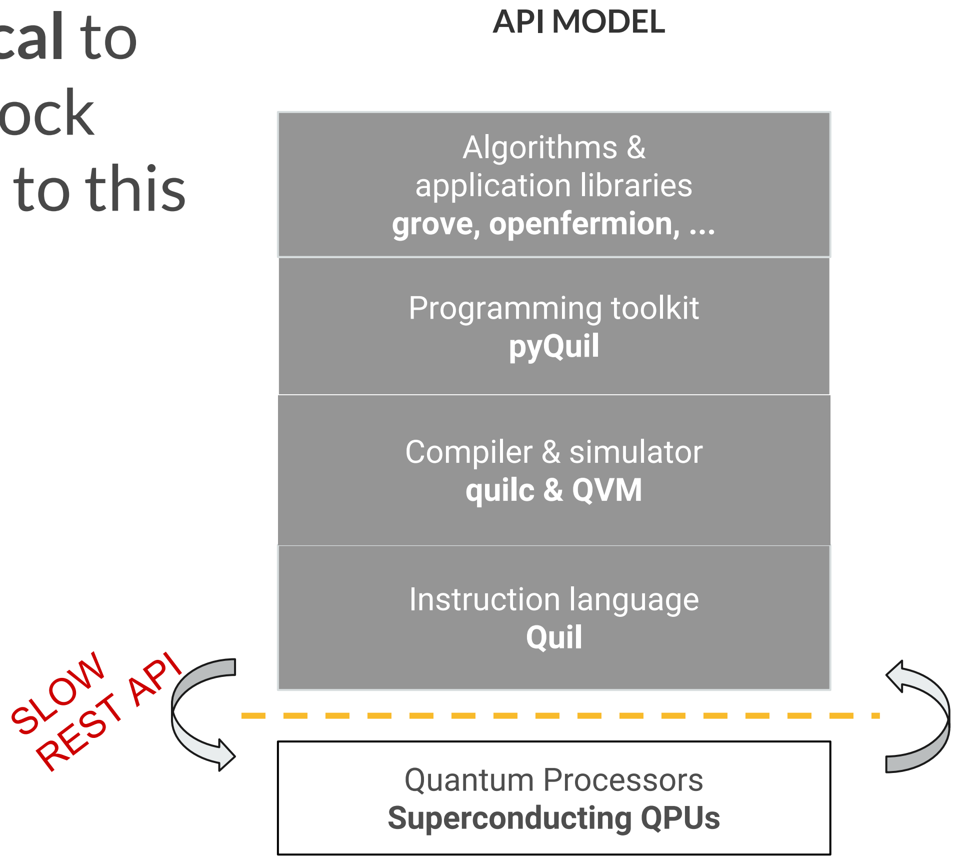
Reality

```
# NxDevice takes a networkx graph as the topology
fully_connected_device = NxDevice(topology=nx.complete_graph(n_qubits))

# generates gate objects with specifications of noise
gates = gates_in_isa(fully_connected_device.get_isa())

# only implement measurement noise
noise_model = _decoherence_noise_model(gates, T1=np.infty, T2=np.infty,
                                                   gate_time_1q=0, gate_time_2q=0,
                                                   ro_fidelity=q0_p00)
# construct QC object with customized everything!
qc = QuantumComputer(name='2q-qvm',
                     qam=QVM(connection=ForestConnection(),
                      noise_model=noise_model),
                      compiler=MyCompiler(),
                      device=fully_connected_device)
```


What's next?

Job to Job latency is critical to hybrid algorithms. Wall clock time is often proportional to this latency.

How can this be reduced?



Rigetti Quantum Cloud Services

No install access to dedicated Quantum Machine Images

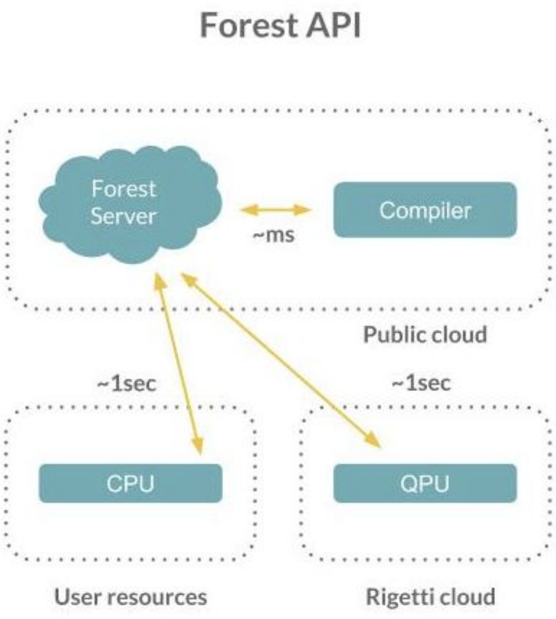
Open source, Python SDK

Fast hybrid programming

Signup for beta access at rigetti.com/qcs

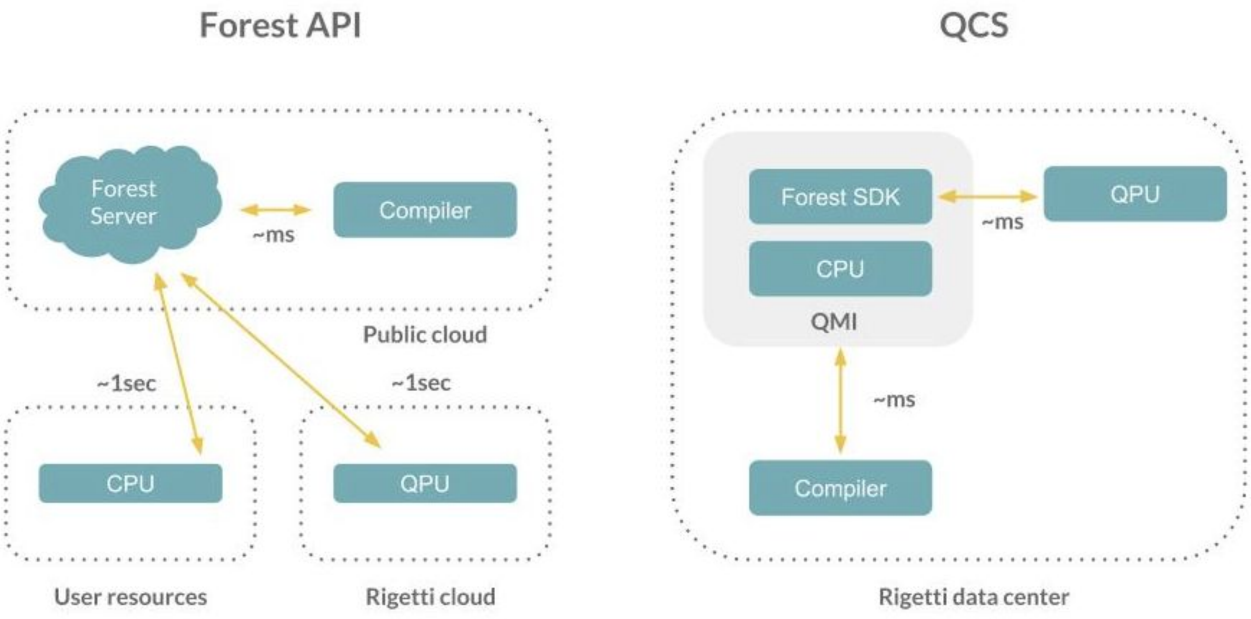
Hybrid computing with the **Quantum Machine Image**

Signup to QCS gives you your own QMI complete quantum development environment (think virtual machine)



Hybrid computing with the **Quantum Machine Image**

Signup to QCS gives you your own QMI complete quantum development environment (think virtual machine)



Quantum Approximate Optimization Algorithm

[QAOA] Hybrid algorithm used for constraint satisfaction problems

Given binary constraints:

$$z \in \{0,1\}^n$$

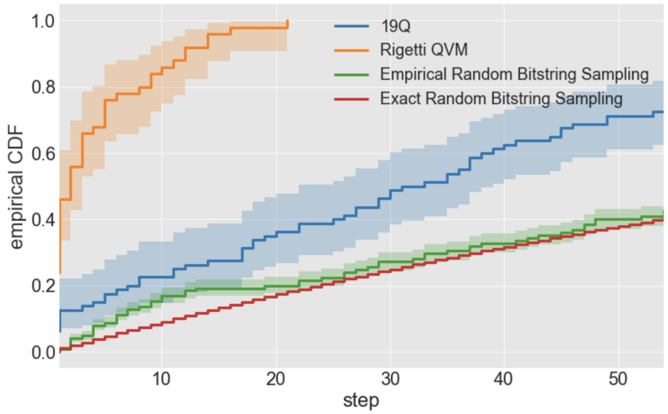
$$C_a(z) = \begin{cases} 1 & \text{if } z \text{ satisfies the constraint } a \\ 0 & \text{if } z \text{ does not .} \end{cases}$$

MAXIMIZE

$$C(z) = \sum_{a=1}^{m} C_a(z)$$

Traveling Salesperson   Scheduling   Clustering   Boltzmann Machine Training

Hadfield et al. 2017 [1709.03489]   Otterbach et al. 2017 [1712.05771]   Verdon et al. 2017 [1712.05304]



QAOA in **Forest**

In ***14*** lines of code

```
from pyquil.quil import Program
from pyquil.gates import H
from pyquil.paulis import sI, sX, sZ, exponentiate_commuting_pauli_sum
from pyquil.api import QPUConnection

graph = [(0, 1), (1, 2), (2, 3)]
nodes = range(4)

init_state_prog = sum([H(i) for i in nodes], Program())
h_cost = -0.5 * sum(sI(nodes[0]) - sZ(i) * sZ(j) for i, j in graph)
h_driver = -1. * sum(sX(i) for i in nodes)

def qaoa_ansatz(betas, gammas):
    return sum([exponentiate_commuting_pauli_sum(h_cost)(g) +
exponentiate_commuting_pauli_sum(h_driver)(b) \
        for g, b in zip(gammas, betas)], Program())

program = init_state_prog + qaoa_ansatz([0., 0.5], [0.75, 1.])

qvm = QPUConnection()
qvm.run_and_measure(program, qubits=nodes, trials=10)
```

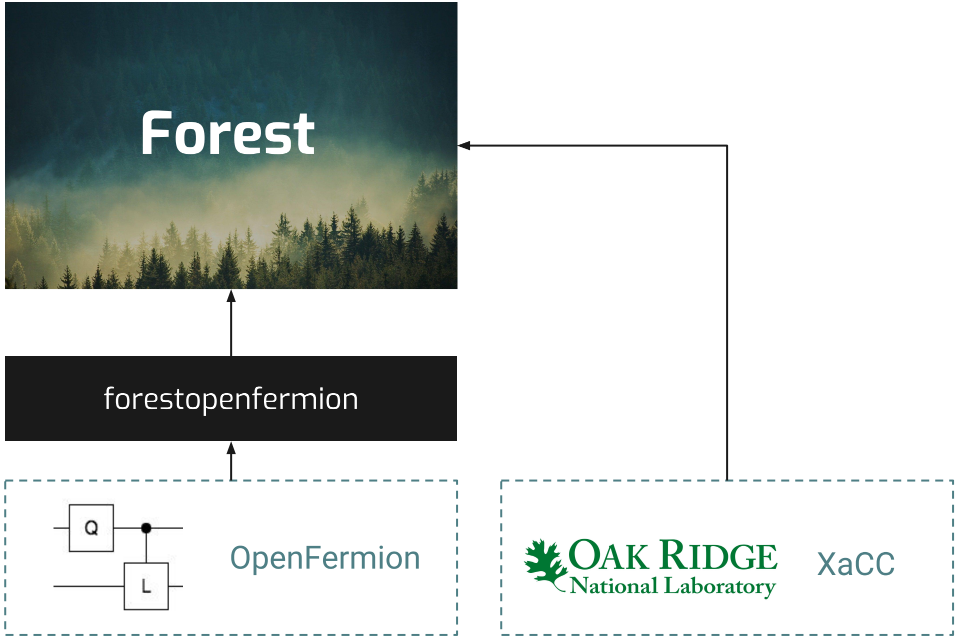
Open areas in quantum programming

> Debuggers

> Optimizing compilers

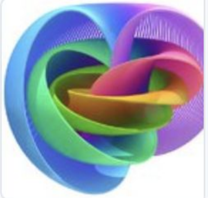
> Application specific packages

> **Adoption and implementations**



Unitary Fund

$2k grants no-strings attached for open source quantum/classical hybrid programming



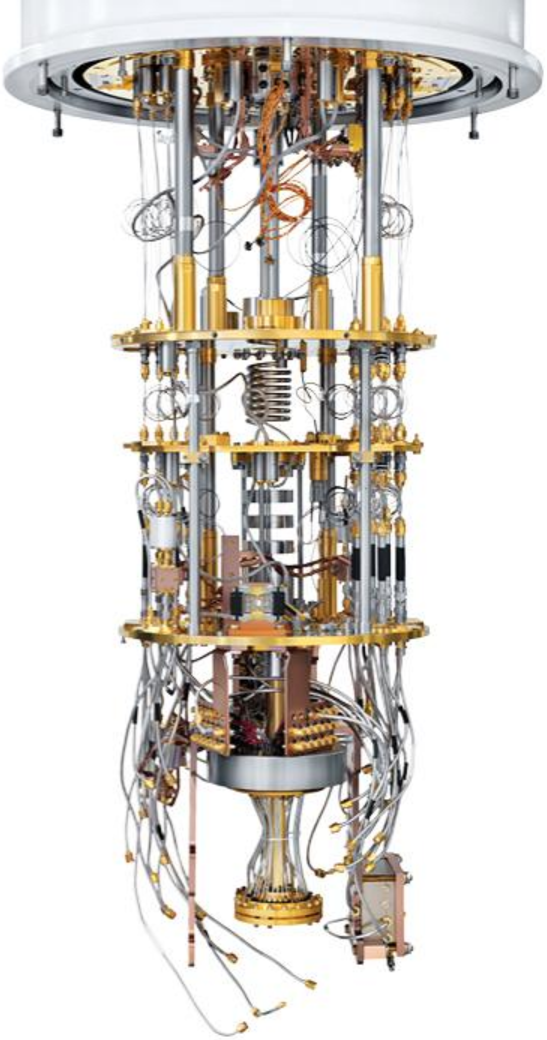
LSaldyt/unitary-proposal

unitary-proposal - My experimental proposal and materials for the unitary fund quantum computing grant

github.com

http://unitary.fund

* Platform agnostic: not Rigetti sponsored



$1M Quantum Advantage Prize

Using Rigetti QCS to solve valuable a business problem **better**, **faster**, or **cheaper** than otherwise possible.

More details online.

Links

QCS signup: https://www.rigetti.com/

Forest SDK: https://www.rigetti.com/forest

Documentation: https://www.pyquil.readthedocs.io