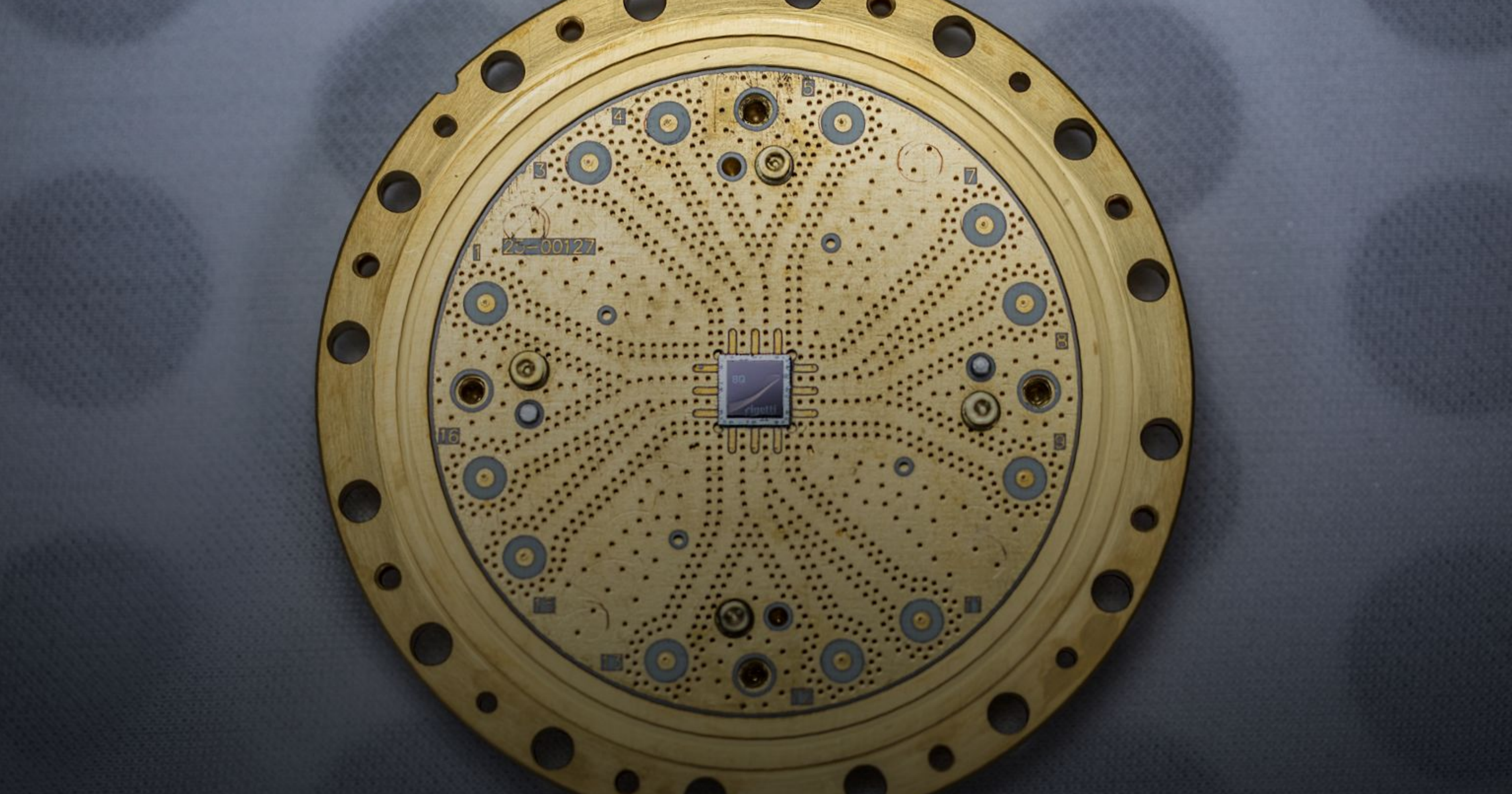
Quantum Cloud Computing

Johannes Otterbach

TU Kaiserslautern
January 22, 2018

Rigetti 8-Qubit Quantum Processor



Rigetti 8-Qubit Quantum Processor

- **> Scalable Gate-model Quantum Processors**
  - > Superconducting Microwave Circuits
- **> Focus on near-term applications**
  - > Quantum/Classical Hybrid Algorithms
- **> Build towards fault-tolerance**
- **> Access over the cloud**
  - > Quantum computers as co-processors

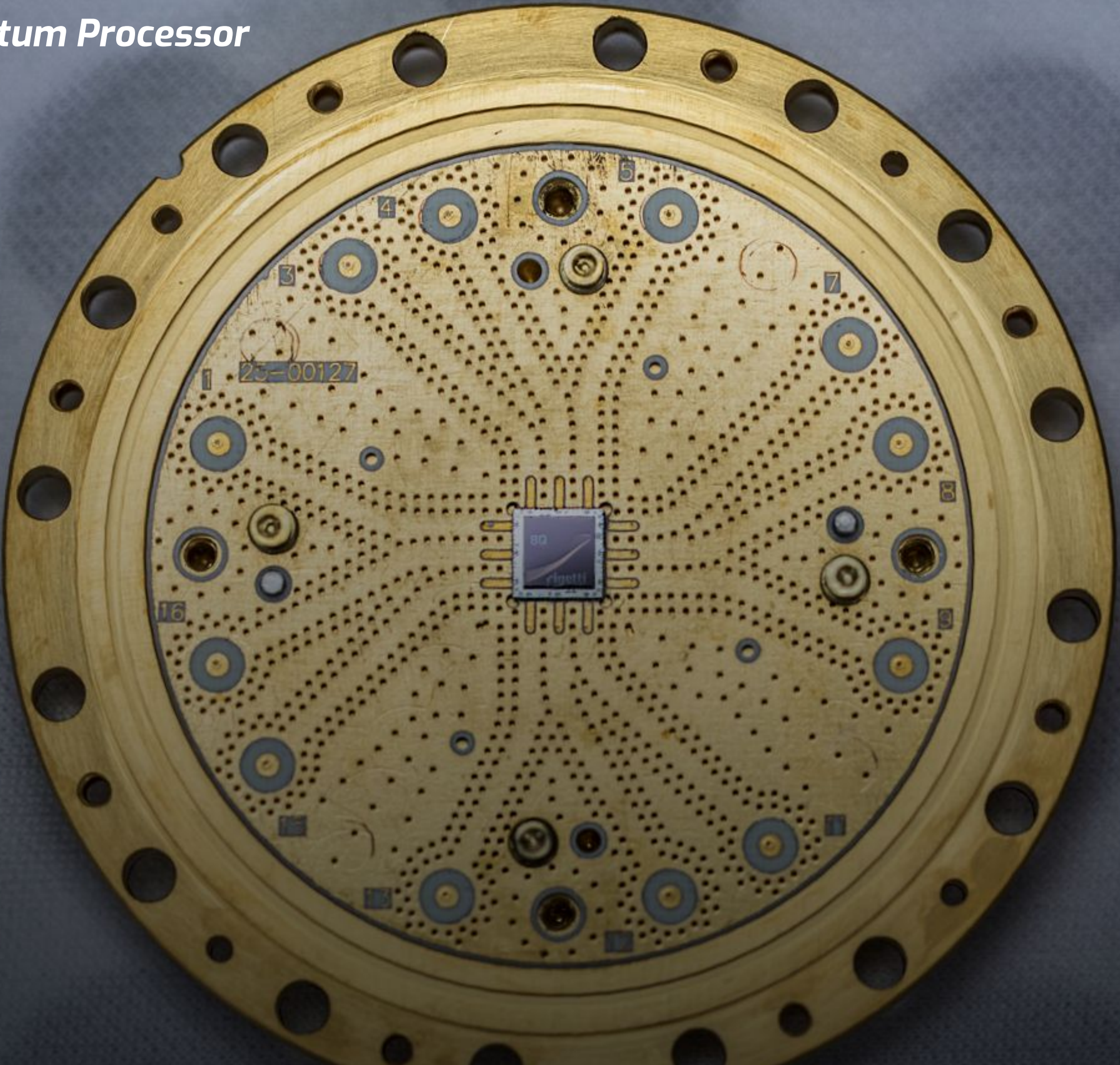
Rigetti 8-Qubit Quantum Processor

Founded in 2013 by Chad Rigetti

Fab-1: Fremont, CA

~100 Employees

Forest: Quantum computing over the cloud



Venture backe
startu

R&D lab: Berkeley, C

Full-stack scalab
superconductin
qubi

30-qubit Quantu
Virtual Machine

Full Stack **Quantum Computing**

Quantum Processor

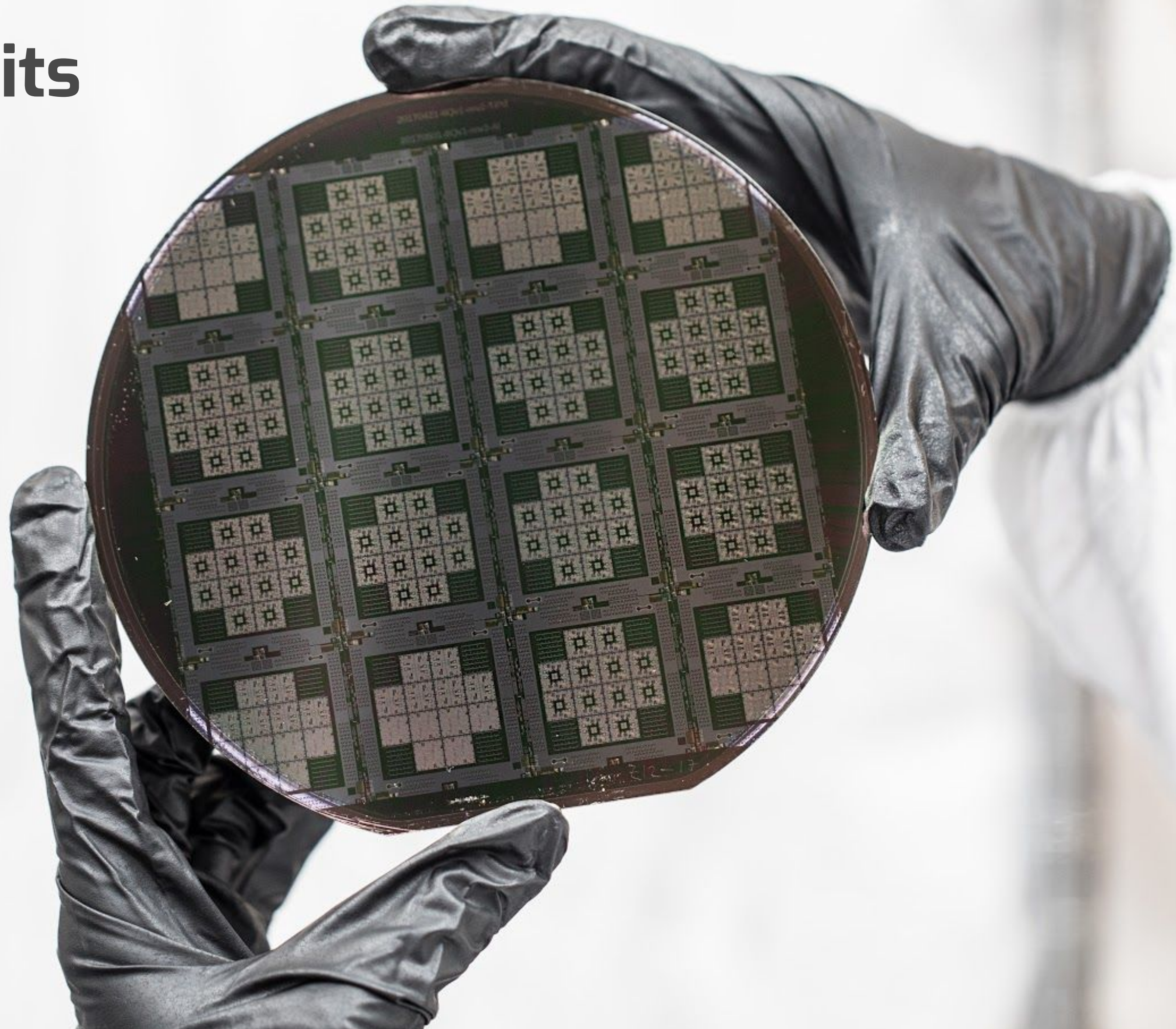
Hardware

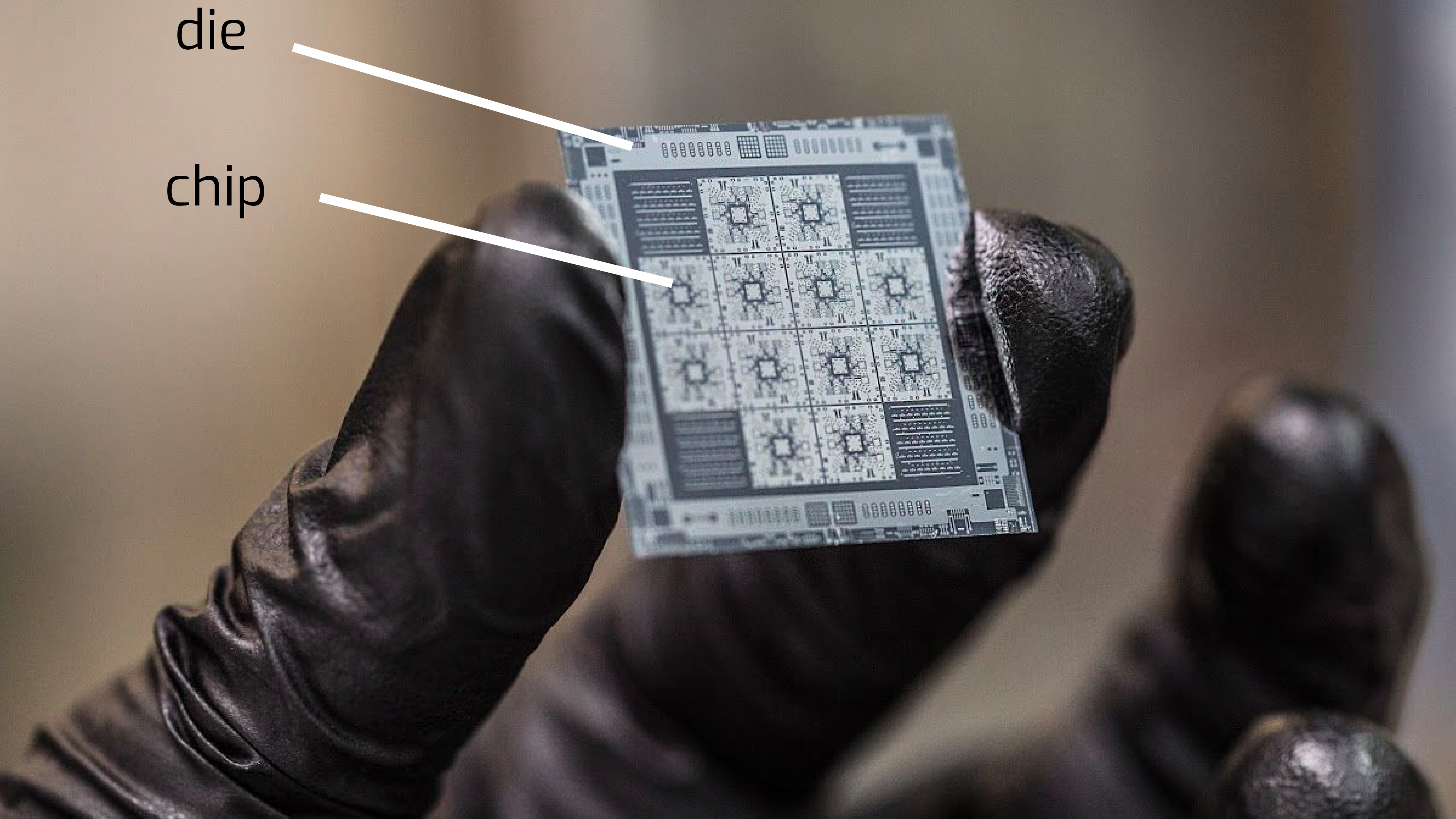
Cloud based Quantum Operating System

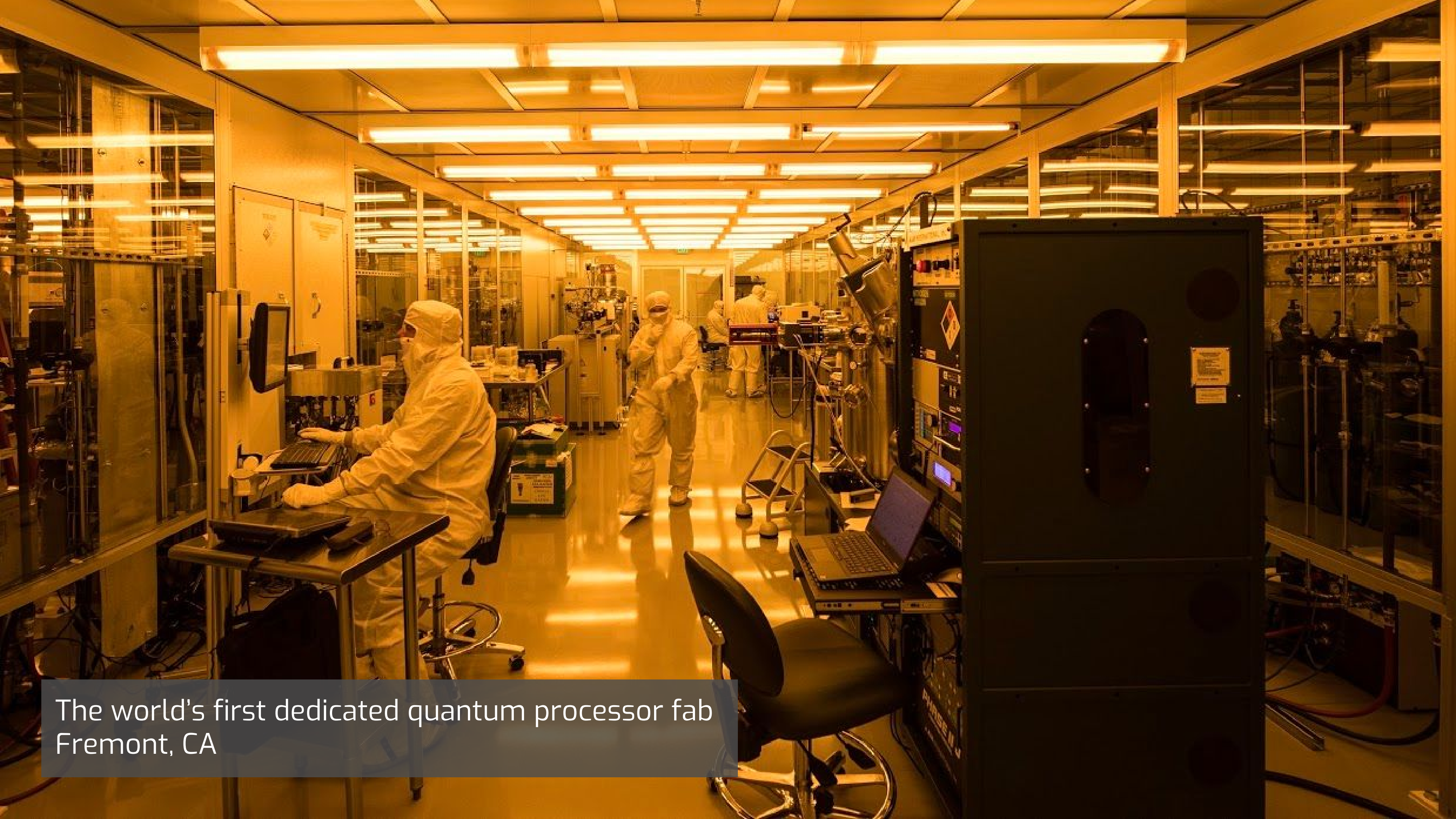
Applications

What do our **Qubits** look like ?

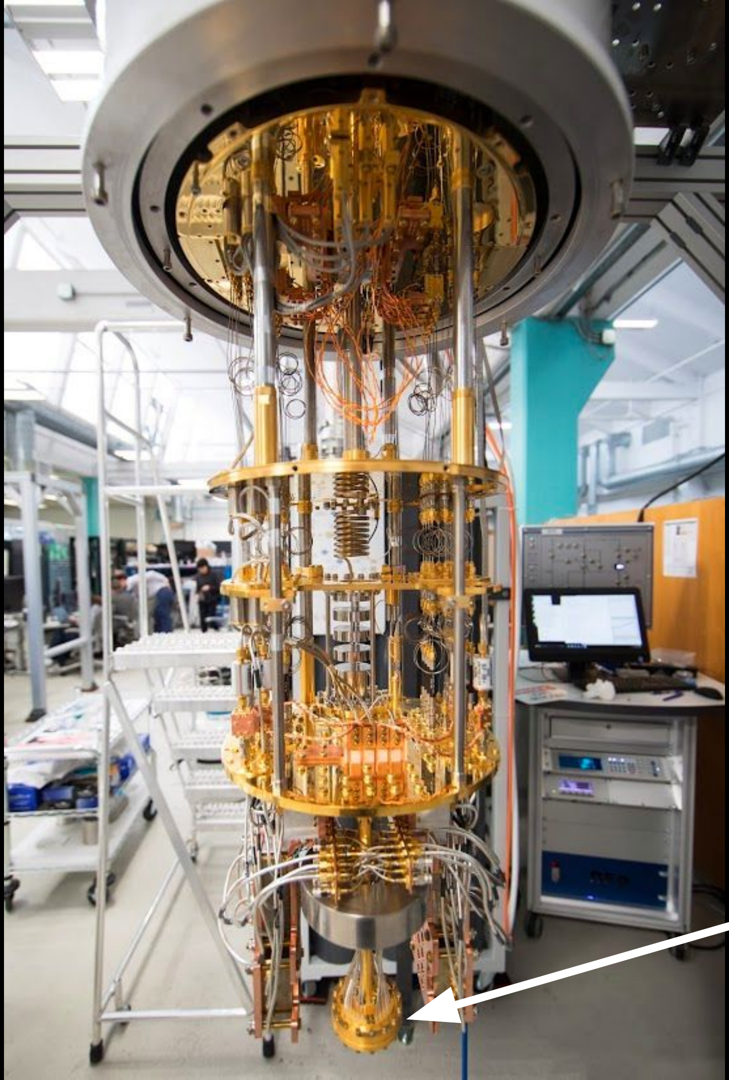
- Superconducting circuits
- Operated near absolute zero
- Aluminum on silicon
- Microwave signal delivery




die
chip

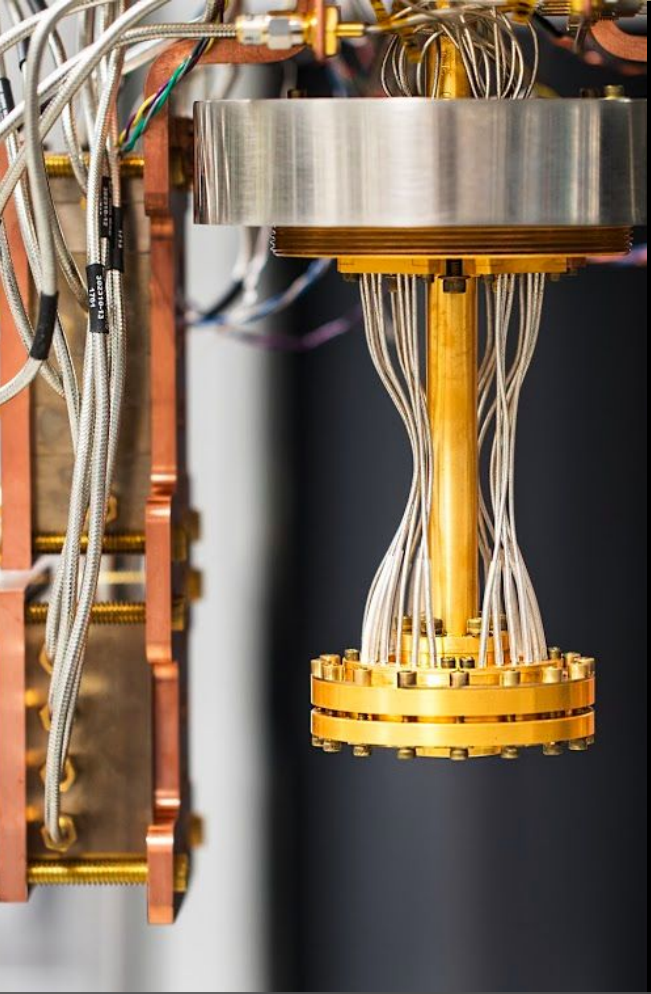


The world's first dedicated quantum processor fab
Fremont, CA



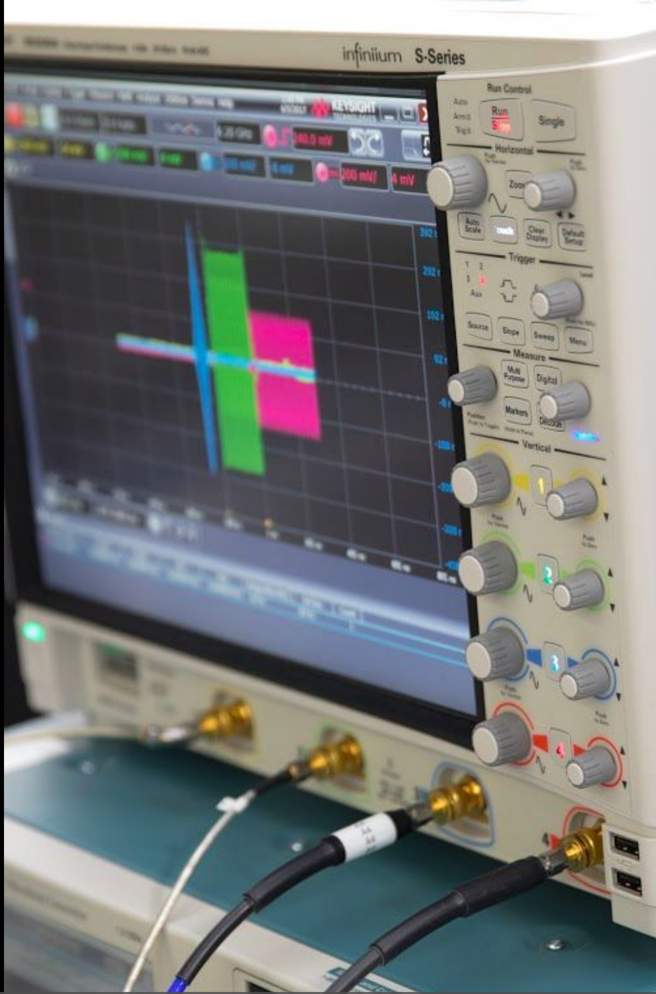
Signal Delivery

The Chip



> Packaging

> Cooling



> Control Electronics

FOREST: Tools for experimental quantum programming

forest.rigetti.com

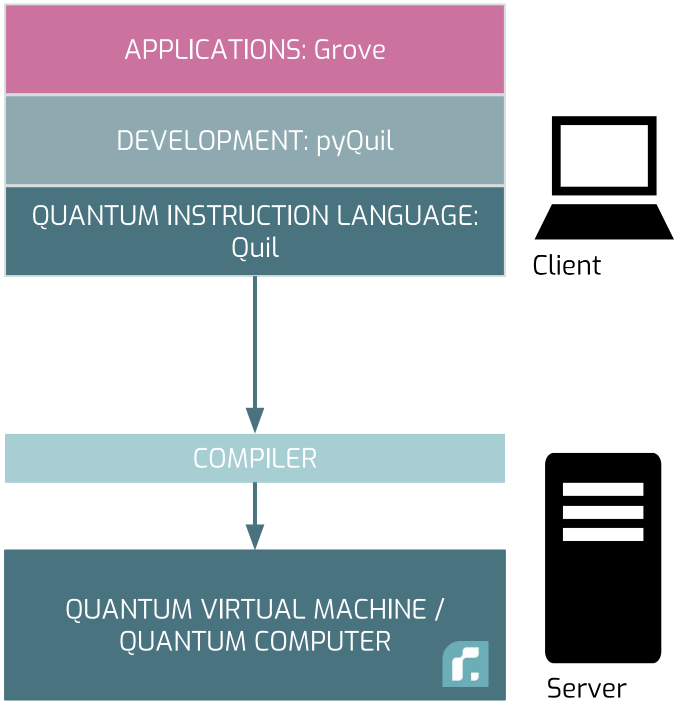
> Write applications…

> using tools…

> that build quantum programs…

> that compile onto quantum hardware…

> that execute on a real or virtual quantum processor.



Open-sourced on github under Apache v2.0 license

github.com/rigetticomputing/pyquil

github.com/rigetticomputing/grove

Simulator in public-beta
Quantum HW

forest.rigetti.com

Robert Smith, Michael Curtis, William Zeng. *A Practical Quantum Instruction Set Architecture*. arXiv:1608.03355

Quantum Algorithms

Simulating quantum systems

1981
Original proposal by Feynman:
Simulating physics with computers

"The physical world is quantum mechanical, and therefore the proper problem is the simulation of quantum physics."

1990

Quantum Algorithms

1994
Shor's factoring algorithm (Shor)

1995
Phase Estimation (PE) introduced (Kitaev)

1997
Hamiltonian Simulation by PE (Lloyd)

2002
Map of fermions to paulis (Somma)

2005
Molecular ground states w/ PE (Aspuru-Guzik)

2010
H_2 ground state using simulated QC (Whitfield)

2013
Hybrid Quantum-Classical Algorithms VQE (McClean)

2015
Approximate Combinatorial Optimization (Farhi)

TODAY

Simulating quantum systems

1981
Original proposal by Feynman:
Simulating physics with computers

"The physical world is quantum mechanical, and therefore the proper problem is the simulation of quantum physics."

Quantum Algorithms

1990

1994
Shor's factoring algorithm (Shor)

1995
Phase Estimation (PE) introduced (Kitaev)

1997
Hamiltonian Simulation by PE (Lloyd)

2002
Map of fermions to paulis (Somma)

2005
Molecular ground states w/ PE (Aspuru-Guzik)

2010
H_2 ground state using simulated QC (Whitfield)

2013
Hybrid Quantum-Classical Algorithms VQE (McClean)

2015
Approximate Combinatorial Optimization (Farhi)

TODAY

2013

2013
Molecular ground states w/ VQE
Implementation on a photonics processor
(Peruzzo)

2014
Quantum Combinatorial Optimization (QAOA)
(Farhi), eg. MAX-CUT, MaxE3Lin2

2015
VQE + PE on superconducting qubits,
Small quantum machine learning examples

2016
Broader applications of VQE (Troyer, **Rubin**)

2017
Machine Learning (Aspuru-Guzik, **Otterbach**)

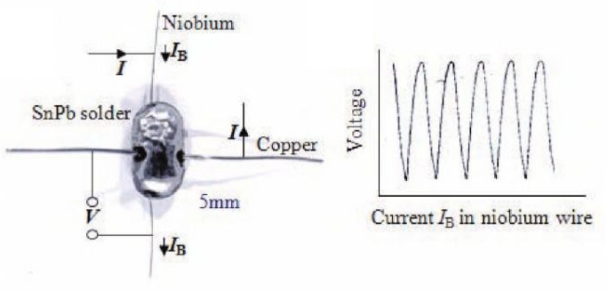
TODAY

Simulating quantum systems

Transmons
&
2-Qubit Gates

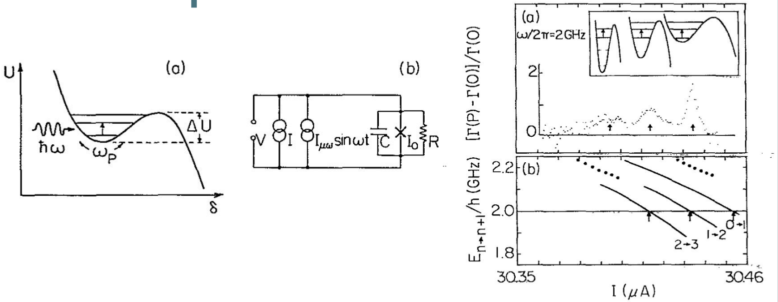
Early years of superconducting qubits

Josephson junctions



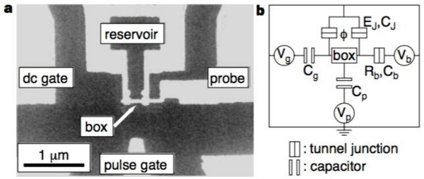
J. Clarke, 1966

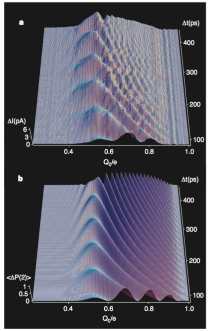
Macroscopic coherence



J. Martinis, M. Devoret, J. Clarke, *PRB*, 1987

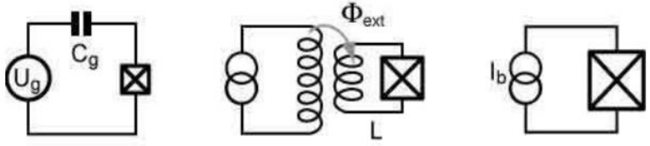
Coherent control





Y. Nakamura *et al.*, *Nature*, 1999

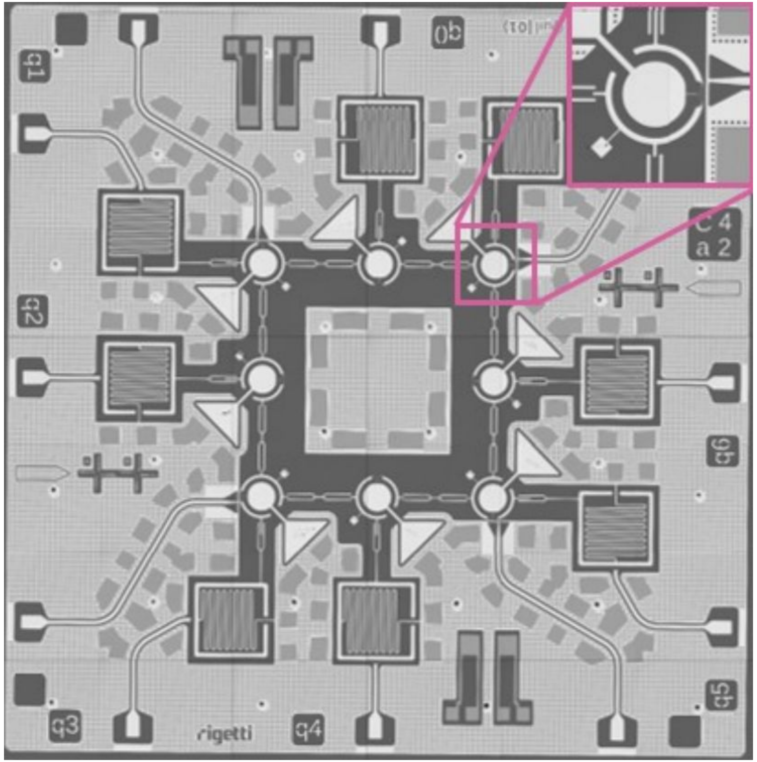
Qubit taxonomy



M. Devoret, A. Wallraff, J. Martinis, arXiv:0411174 (2004)

Our Implementation

Aluminum circuit on Silicon



5.5mm

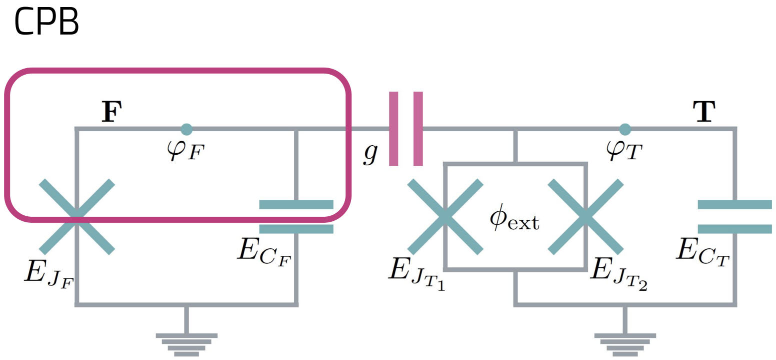
Circuit Quantum Electrodynamics (cQED)

- Superconducting circuits at very low temps (0.01 Kelvin)
- Qubit = Circuit element made with Josephson Junctions (JJ's)

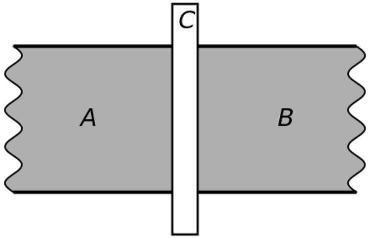
Blais, cond-mat/0402216
Wallraff, cond-mat/0407325

cQED - the transmon qubit(s)

- Josephson Junction - Cooper Pair Box
- Transmon regime: $E_J \gg E_C$
- JJ's are nonlinear
- Nonlinearity→two-level subspace→qubit
- Qubits coupled to linear resonators for state readout
- Jaynes-Cummings Hamiltonian: Atomic transition coupled to cavity mode
- Two types of transmons: Fixed & Tunable



Josephson Effect



$$i\hbar\frac{\partial}{\partial t}\begin{pmatrix}\psi_A\\ \psi_B\end{pmatrix} = \begin{pmatrix}qV/2 & K\\ K & -qV/2\end{pmatrix}\begin{pmatrix}\psi_A\\ \psi_B\end{pmatrix}$$

- Fundamental commutation relation of cQED

$$[\hat{\Phi}, \hat{Q}] = i\hbar$$

- Equations of Motion

$$I_C = \dot{\rho}_A = -\dot{\rho}_B = I_0 \sin(\phi)$$

$$\dot{\phi} = \dot{\theta}_A - \dot{\theta}_B = \frac{qV}{\hbar}$$

Fixed SC Qubit

- Capacitor

$$I_C = C\frac{dV_C}{dt} = \frac{\hbar}{q}\ddot{\phi}$$

- Junction

$$I_J = I_0 \sin(\phi)$$

- Equation of motion

$$\frac{\hbar}{q}\ddot{\phi} + I_0 \sin(\phi) = I_b$$

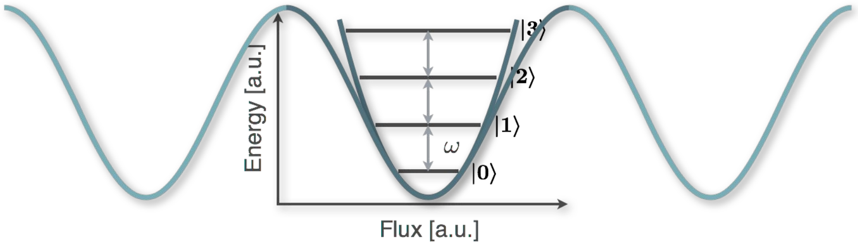
- Using Lagrangian formalism and fundamental commutation relation to arrive at Hamiltonian

$$H = E_C \hat{n}^2 - E_J \cos(\hat{\phi}) - E_J \frac{I_b}{I_0}\hat{\phi}$$

Washboard and SC-Qubits

- Taylor expansion: Harmonic Oscillator

$$H = E_C\hat{n}^2 + \frac{1}{2}E_J\cos(\phi_0)(\hat{\phi} - \phi_0)^2$$



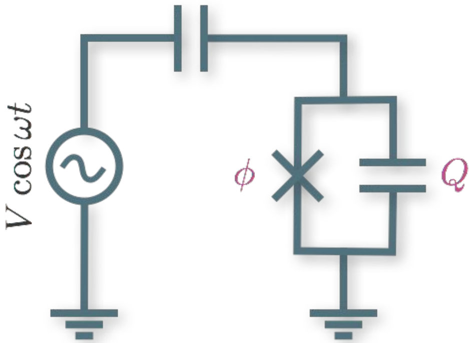
Washboard and SC-Qubits

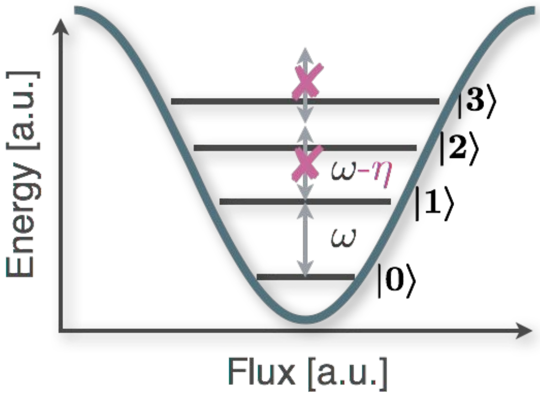
- Taylor expansion: Harmonic Oscillator

$$H = E_C\hat{n}^2 + \frac{1}{2}E_J\cos(\phi_0)(\hat{\phi} - \phi_0)^2$$

- Higher order correction: Anharmonic spectrum
- Ground- and 1st excited state form Qubit states
- Transmon Regime

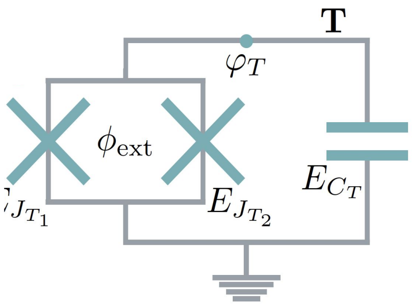
$$E_J \gg E_C$$





Tunable SC Qubit

- Phase sensitive SQUID
- Hamiltonian



$$H = 4E_C\hat{n}^2 - E_{J_1}\cos(\hat{\phi} - \varphi_{\text{ext}}) - E_{J_2}\cos(\hat{\phi})$$

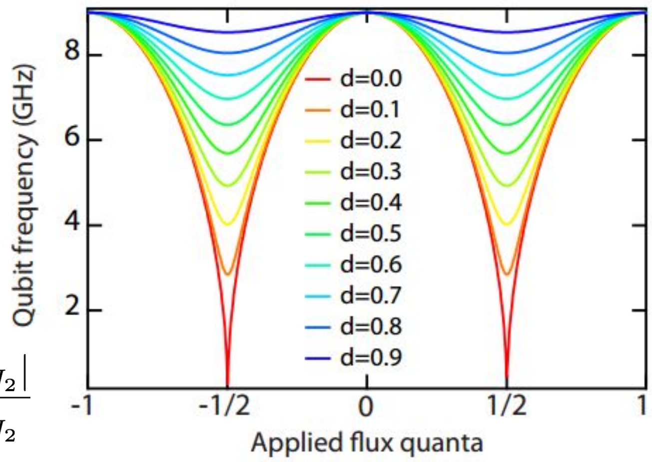
Symmetric junctions: $E_{J1} = E_{J2}$

Phases cancel completely at $\boldsymbol{\Phi=\Phi_0/2}$

Asymmetric junctions: E_{J1} != E_{J2}

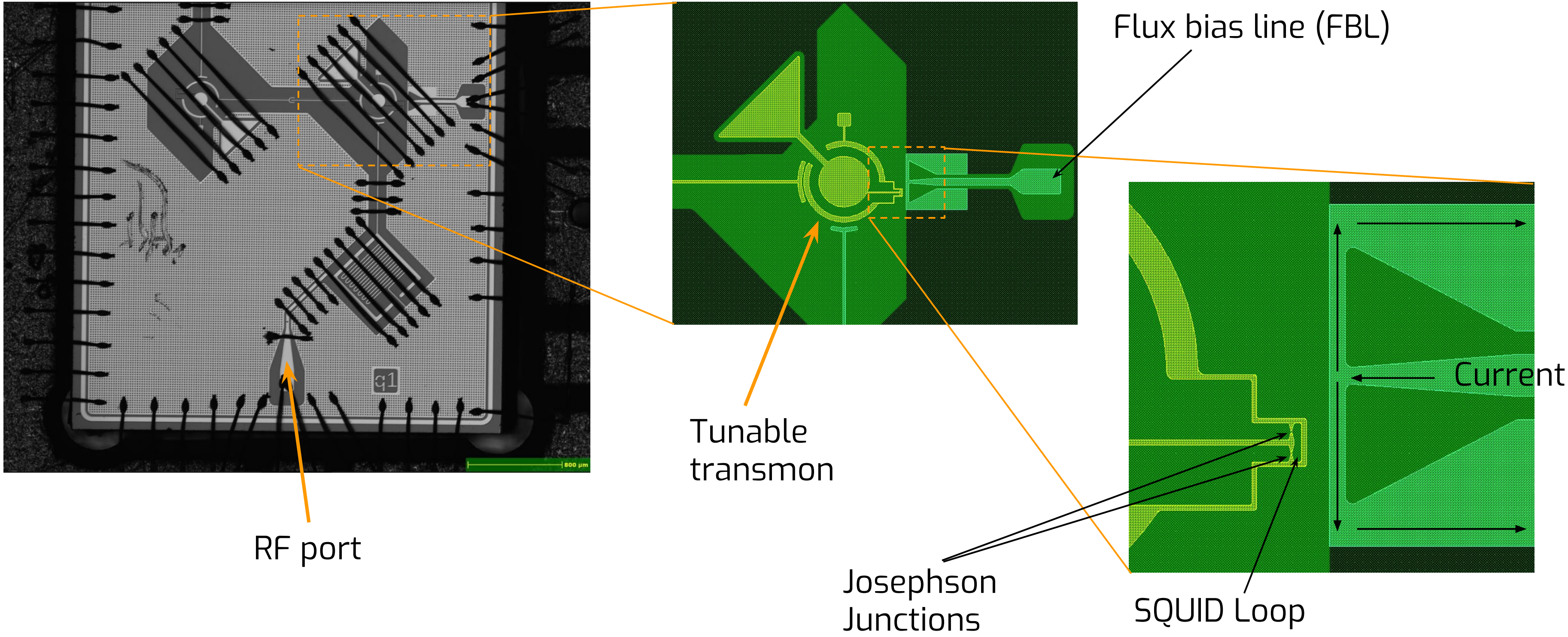
Phases cancel incompletely at $\boldsymbol{\Phi=\Phi_0/2}$

$$d = \frac{|E_{J_1} - E_{J_2}|}{E_{J_1} + E_{J_2}}$$

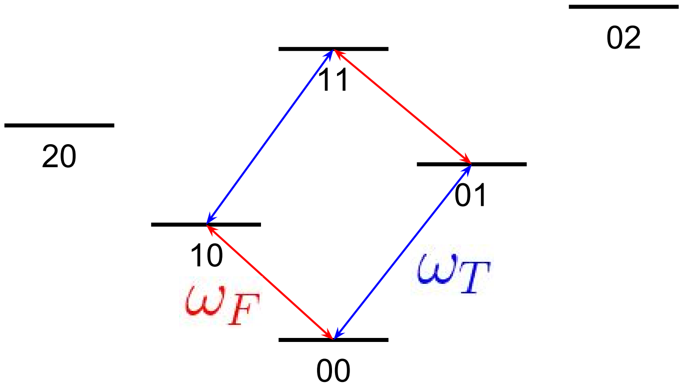


mage from M. Reed, *Entanglement and Quantum Error Correction using Superconducting Qubits*, 2013, arXiv:1311.6759

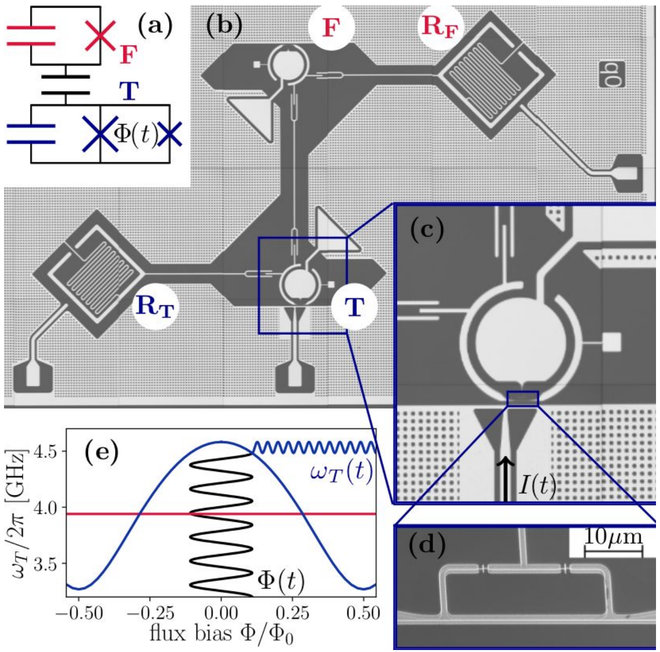
Experimental Device



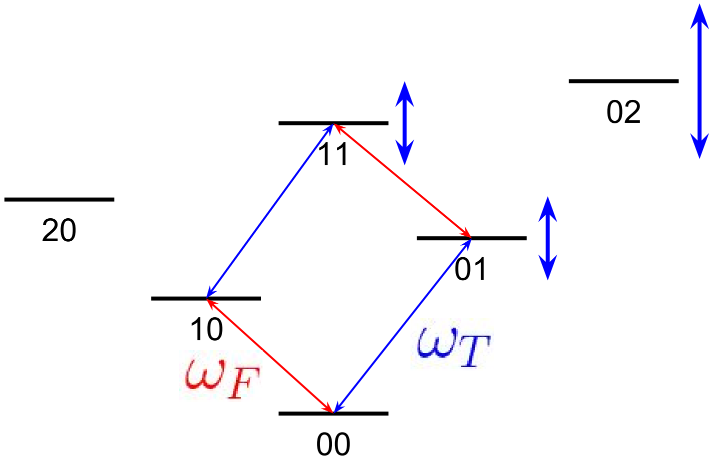
Parametric Entangling Gates



$$\Phi(t) = \overline{\Phi} + \widetilde{\Phi}\cos(\omega_p t + \theta_p)$$

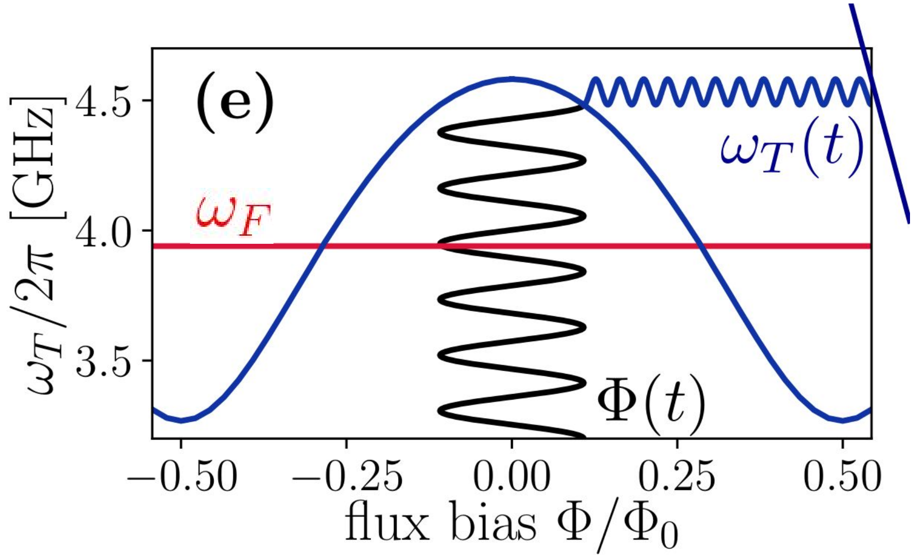


Parametric Entangling Gates

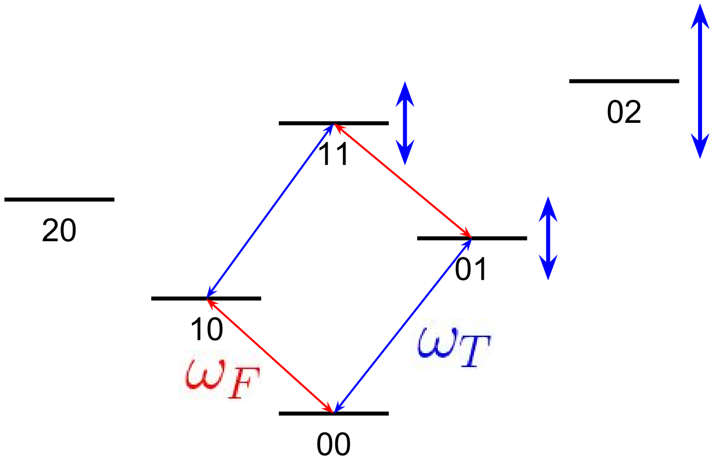


$$\Phi(t) = \overline{\Phi} + \widetilde{\Phi}\cos(\omega_p t + \theta_p)$$

$$\omega_T(t) \approx \overline{\omega_T}(\widetilde{\Phi}) + \widetilde{\omega_T}(\widetilde{\Phi})\cos(2\omega_p t + 2\theta_p)$$

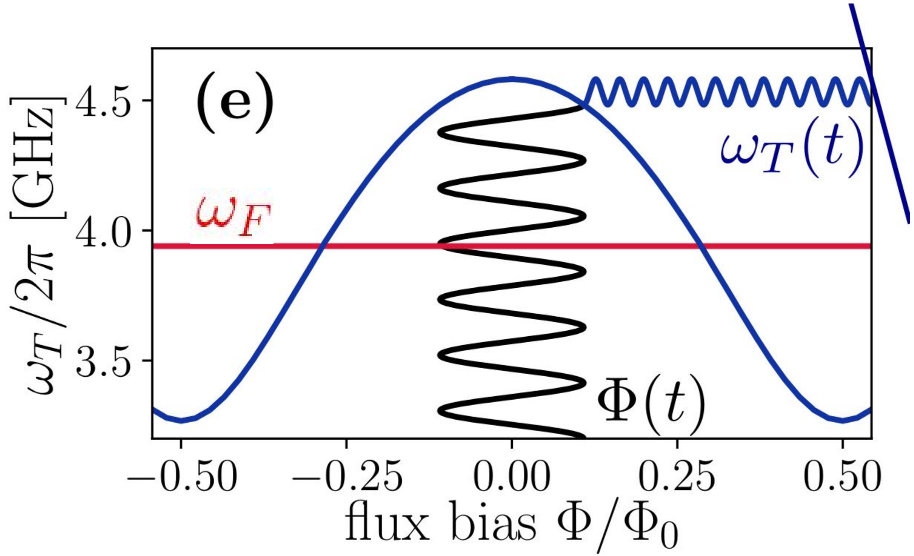


Parametric Entangling Gates



$$\Phi(t) = \overline{\Phi} + \widetilde{\Phi}\cos(\omega_p t + \theta_p)$$

$$\omega_T(t) \approx \overline{\omega_T}(\widetilde{\Phi}) + \widetilde{\omega_T}(\widetilde{\Phi})\cos(2\omega_p t + 2\theta_p)$$

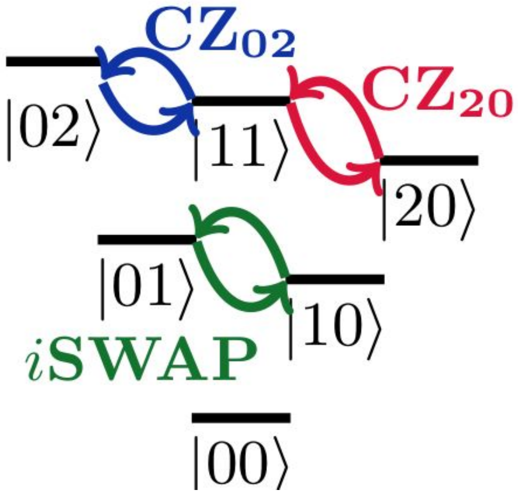


Resonance when

$$(\omega_T - \overline{\omega_T}) + 2\omega_p = \Delta_{ij}$$

for some Δ between levels *i*, *j*

Parametric Entangling Gates



$$iSWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Resonance when

$$(\omega_T - \overline{\omega_T}) + 2\omega_p = \Delta_{ij}$$

for some Δ between levels *i*, *j*

Randomized Benchmarking

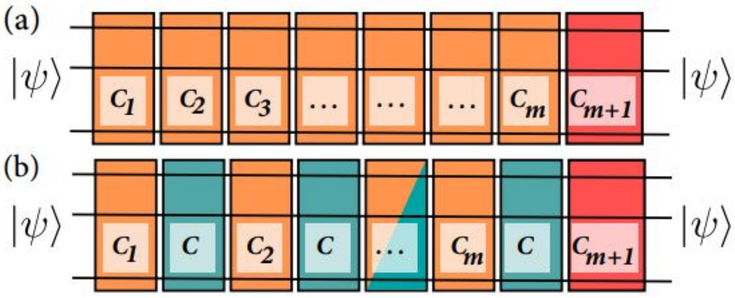
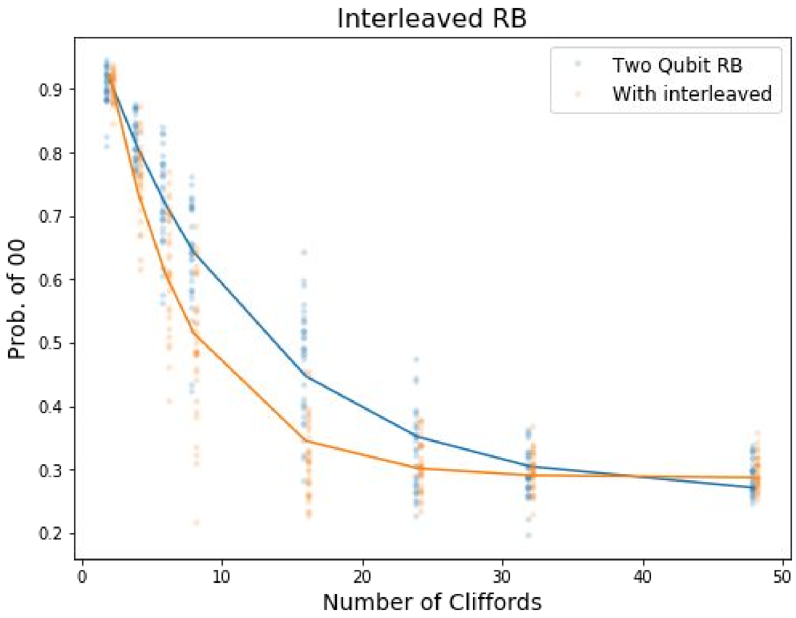


Image from Magesan



Interleaved RB

A. Run sequences of 2Q Cliffords

B. Run sequences with gate *C* interleaved

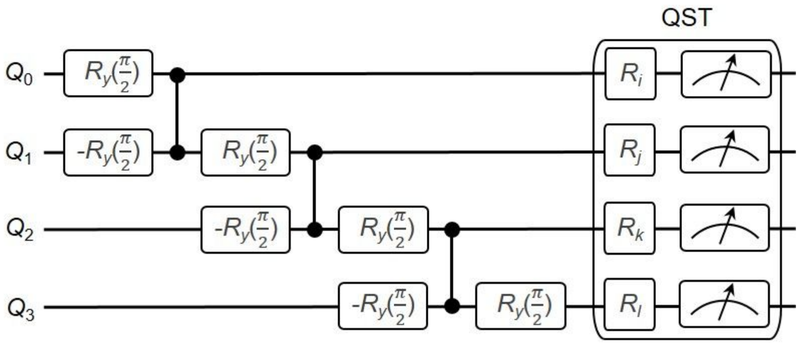
Attempt to isolate the infidelity due to *C*

2Q gate type	Avg error per 2Q gate
iSwap	5.9%
CZ_{02}	8.5%
CZ_{20}	8.7%

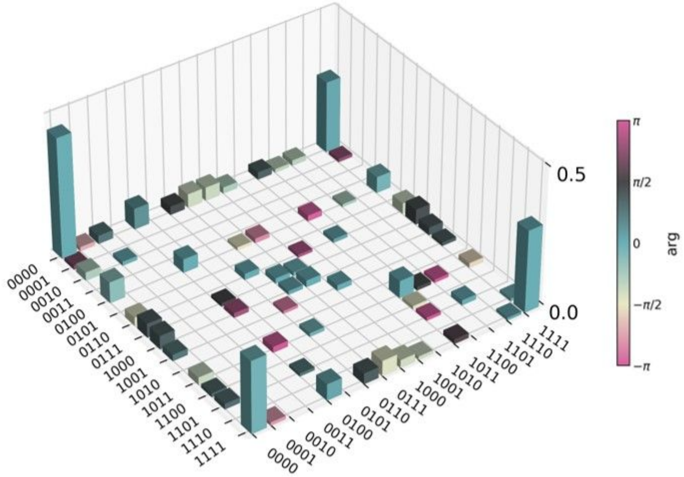
Magesan arXiv:1203.4550v2

4 qubit entangled state verification

Quantum Circuit



Quantum State Tomography



Fidelity of 79%

Quil and the Quantum Abstract Machine

A hybrid classical/quantum programming model.

FSM & QAM

FSM

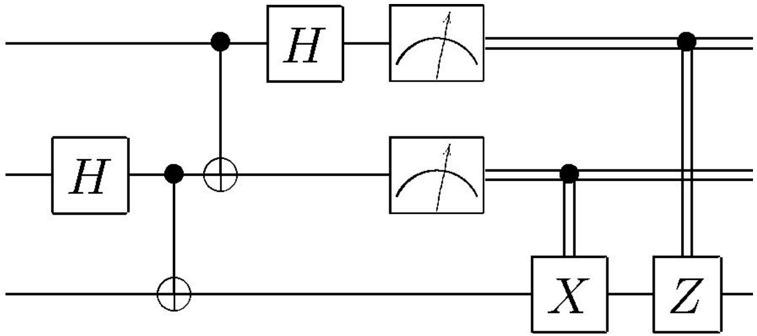
- Classic Bits 0 and 1 encode the state
- Universal Gate Sets
  - NOT + AND
  - NOT + OR
  - AND + XOR
  - ...
- Execution state (next instruction)

QAM

- Qubits $|0\rangle$ and $|1\rangle$
- Universal Quantum Gate Set
  - CNOT + Single Qubit Gates

- Classic Bits 0 and 1 encode the state
- Universal Gate Sets
  - NOT + AND
  - NOT + OR
  - AND + XOR
  - ...
- Execution state (next instruction)

Quil is **portable** and **hybrid**.



The Quil Programming Model

Targets a **Quantum Abstract Machine (QAM)**

> **Quil** is the instruction language and is how you interact with the machine

> It is a syntax for representing state transitions.

The Quil Programming Model

Targets a **Quantum Abstract Machine (QAM)**

> **Quil** is the instruction language and is how you interact with the machine

> It is 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

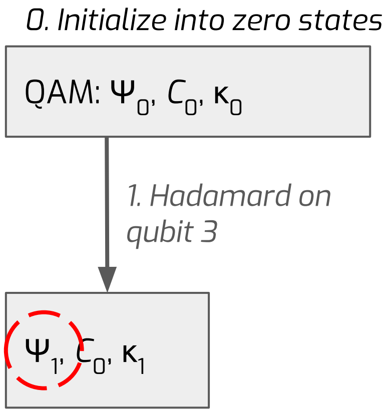
Targets a **Quantum Abstract Machine (QAM)**

> **Quil** is the instruction language and is how you interact with the machine
> It is 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]
.
.
.
```

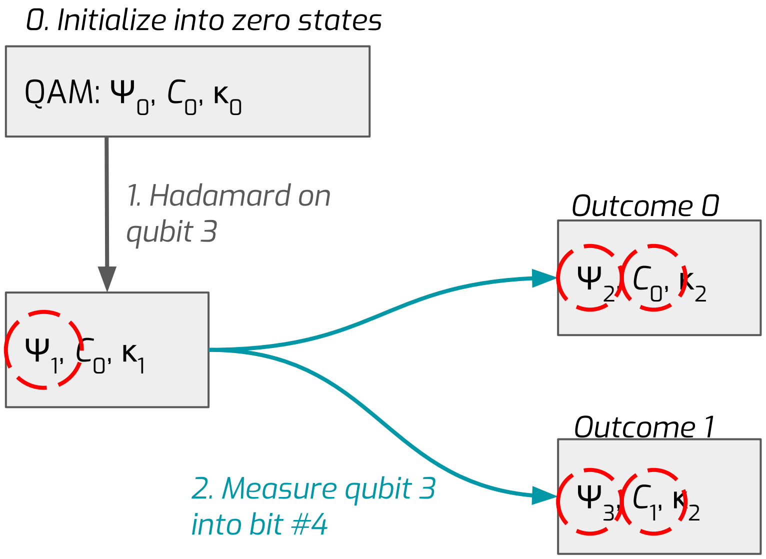
The Quil Programming Model

Targets a **Quantum Abstract Machine (QAM)**

> **Quil** is the instruction language and is how you interact with the machine
> It is 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]
.
.
.
```



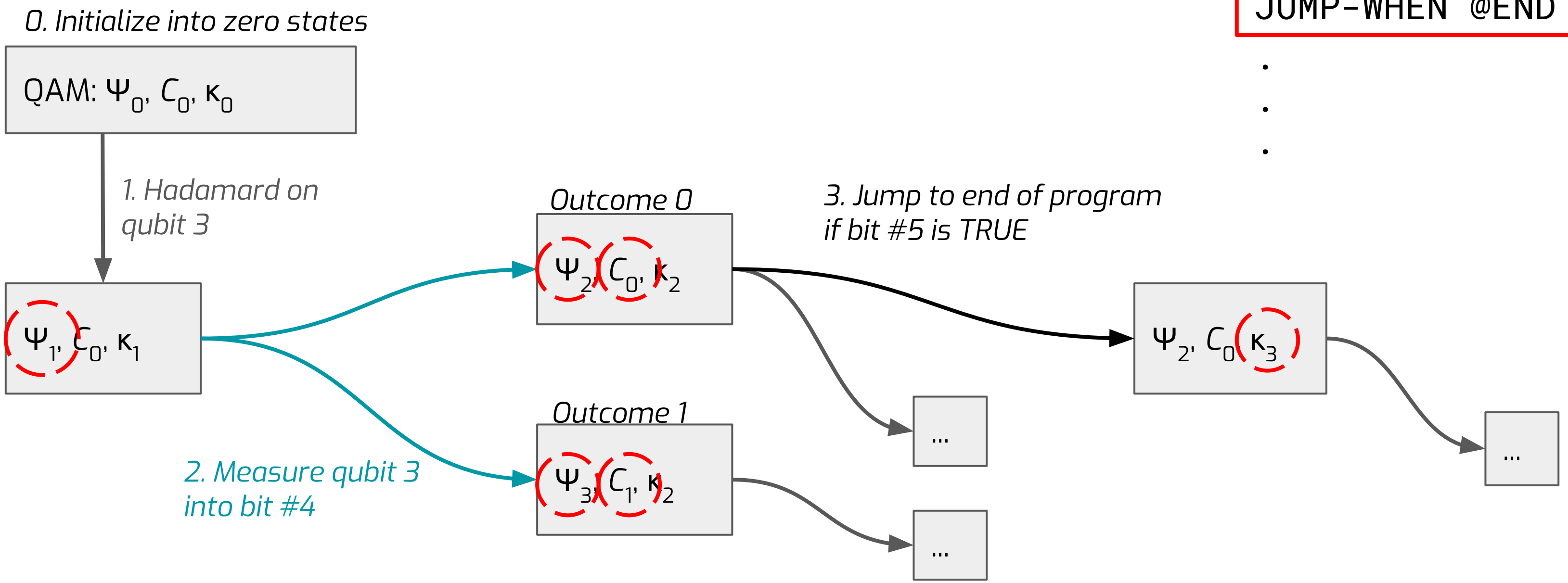
The Quil Programming Model

Targets a **Quantum Abstract Machine (QAM)**

> **Quil** is the instruction language and is how you interact with the machine
> It is a syntax for representing state transitions.

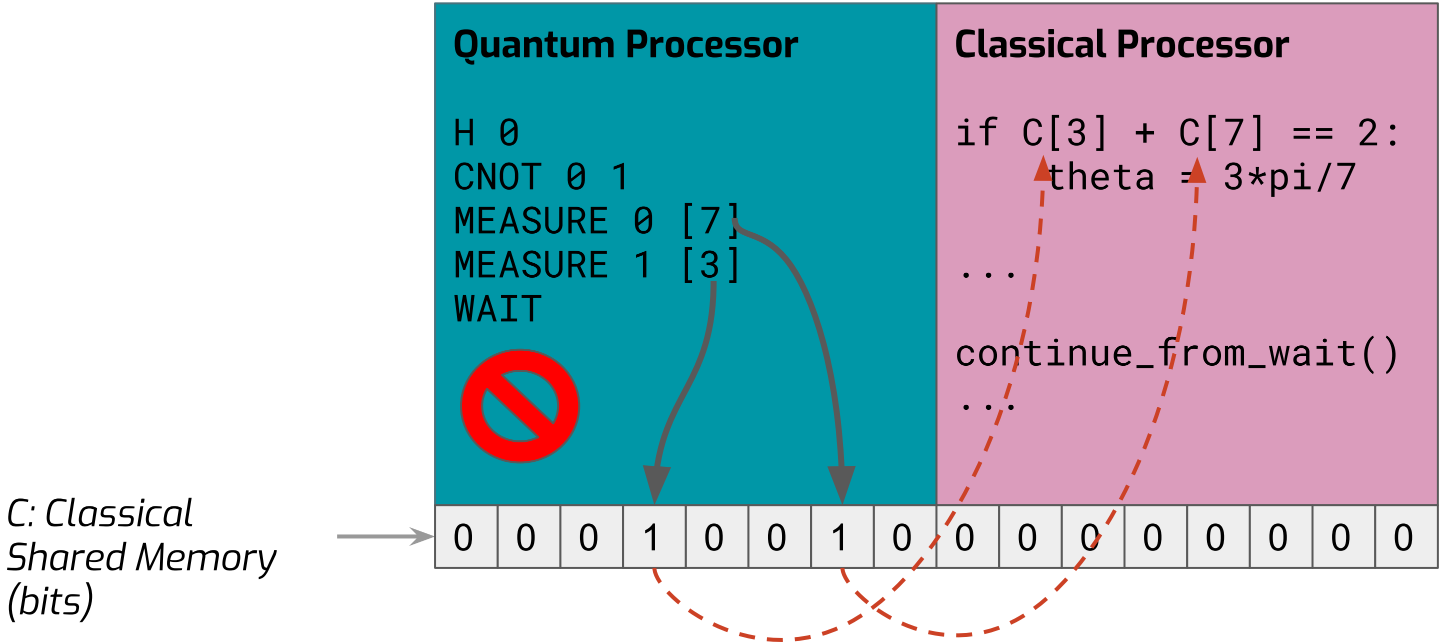
Ψ: Quantum state (qubits) → quantum instructions
C: Classical state (bits) → classical and measurement instructions
κ: Execution state (program)→ control instructions (e.g., jumps)

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



Interacting with a Classical Computer

> The Quantum Abstract Machine has a **shared classical state**.
> The QAM becomes a practical device with this shared state.
> Classical computers can take over with classical/quantum synchronization.



Formal Details: The Quil White Paper

arXiv:1608.03355

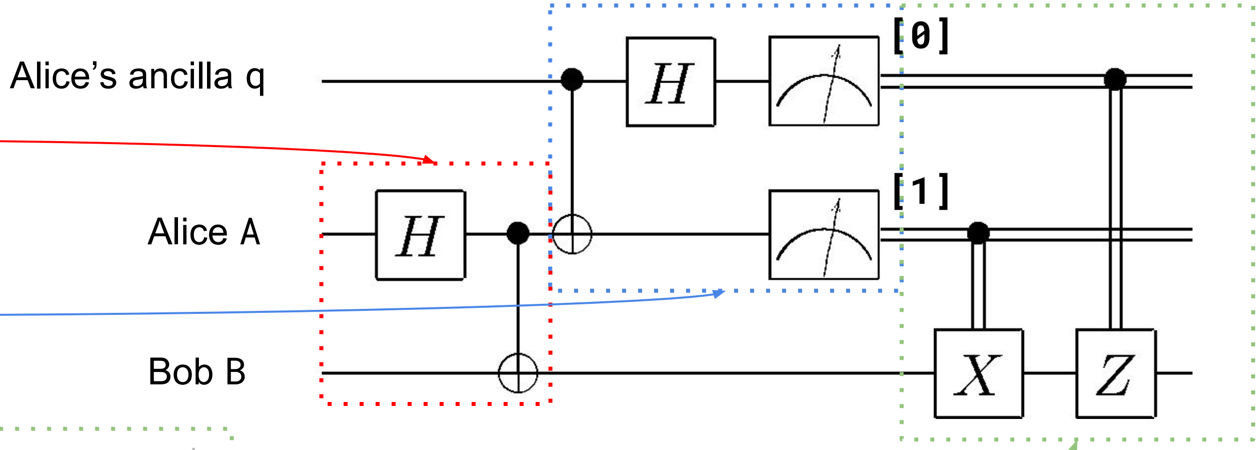
A Practical Quantum Instruction Set Architecture

Robert S. Smith, Michael J. Curtis, William J. Zeng
Rigetti Computing
775 Heinz Ave.
Berkeley, California 94710
Email: {robert, spike, will}@rigetti.com

***Abstract*—Quantum computing technology has advanced rapidly in the last few years. Physical systems—superconducting qubits in particular—promise scalable gate-based hardware. Alongside these advances, new algorithms have been discovered that are adapted to the relatively smaller, noisier hardware that will become available in the next few years. These tend to be hybrid classical/quantum algorithms, where the quantum hardware is used in a co-processor model. Here, we introduce an abstract machine architecture for describing these algorithms, along with a language for representing computations on this machine, and discuss a classically simulable implementation architecture.** ***Keywords*—quantum computing, software architecture**

Quantum Teleportation in Quil

```
DEFCIRCUIT TELEPORT A q B:
    # Bell pair
    H         A
    CNOT      A B

    # Teleport
    CNOT      q A
    H         q
    MEASURE   q [0]
    MEASURE   A [1]

    # Classically communicate measurements
    JUMP-UNLESS @SKIP [1]
    X B
    LABEL @SKIP
    JUMP-UNLESS @END [0]
    Z B
    LABEL @END

# If Alice's qubits are 0 and 1
# and Bob's is 5
TELEPORT 0 1 5
```



Teleportation Truth Table

- Bell State: $|00\rangle + |11\rangle$
- Ancilla: $\alpha|0\rangle + \beta|1\rangle$

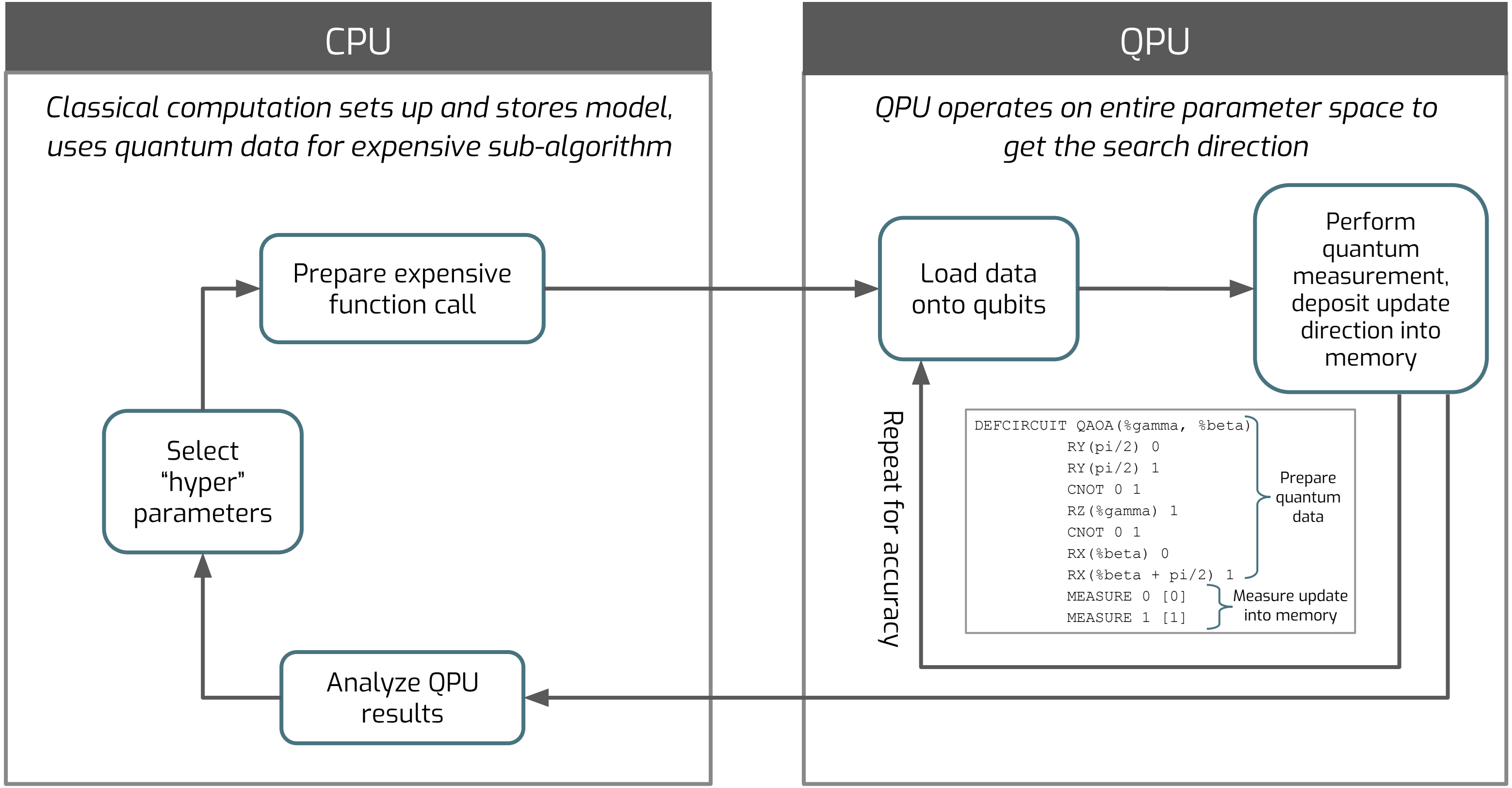
State $\|A B q\rangle$	CNOT q A	H q	Classic C-X A B	Classic C-Z q B
$\alpha\|000\rangle$	$\alpha\|000\rangle$	$\|00\rangle_{Aq}(\alpha\|0\rangle + \beta\|1\rangle)_B$	$(0, 0)_{Aq}; \alpha\|0\rangle + \beta\|1\rangle$	$\alpha\|0\rangle + \beta\|1\rangle$
$\beta\|001\rangle$	$\beta\|101\rangle$	$\|11\rangle_{Aq}(\alpha\|1\rangle - \beta\|0\rangle)_B$	$(1, 1)_{Aq}; \alpha\|0\rangle - \beta\|1\rangle$	$\alpha\|0\rangle + \beta\|1\rangle$
$\alpha\|110\rangle$	$\alpha\|110\rangle$	$\|10\rangle_{Aq}(\alpha\|1\rangle + \beta\|0\rangle)_B$	$(1, 0)_{Aq}; \alpha\|0\rangle + \beta\|1\rangle$	$\alpha\|0\rangle + \beta\|1\rangle$
$\beta\|111\rangle$	$\beta\|011\rangle$	$\|01\rangle_{Aq}(\alpha\|0\rangle - \beta\|1\rangle)_B$	$(0, 1)_{Aq}; \alpha\|0\rangle - \beta\|1\rangle$	$\alpha\|0\rangle + \beta\|1\rangle$

pyQuil generates Quil

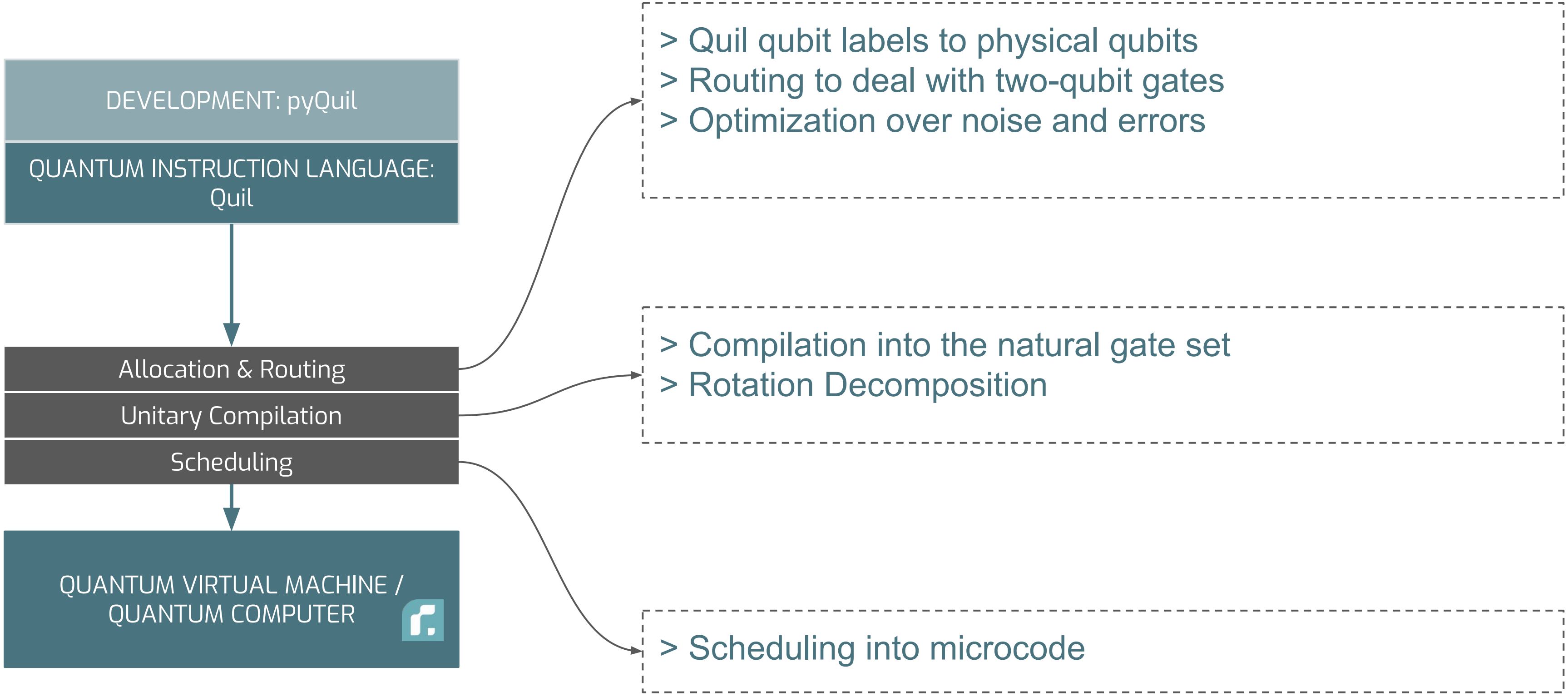
```python
from pyquil.gates import  X, CNOT, H, Z, RX, I
from pyquil.api import QVMConnection
from pyquil.quil import Program
import numpy as np

qvm = QVMConnection()

alice_register = 0
ancilla_register = 1

flip_correction_branch = Program(X(1))
phase_correction_branch = Program(Z(1))

prog = (Program()
        .inst(H(0))
        .inst(CNOT(0, 1))
        .inst(RX(0.2 * np.pi, 2))
        .inst(CNOT(2, 0))
        .inst(H(2))
        .measure(0, alice_register)
        .measure(2, ancilla_register)
        .if_then(alice_register, flip_correction_branch)
        .if_then(ancilla_register, phase_correction_branch))

qvm.run_and_measure(prog, list(prog.get_qubits()), trials=10)
```

```
H 0
CNOT 0 1
RX(pi/5) 2
CNOT 2 0
H 2
MEASURE 0 [0]
MEASURE 2 [1]
JUMP-WHEN @THEN1 [0]
JUMP @END2
LABEL @THEN1
X 1
LABEL @END2
JUMP-WHEN @THEN5 [1]
JUMP @END6
LABEL @THEN5
Z 1
LABEL @END6
```

Hybrid Quantum Computing



Compilation



Some useful tools

- QVM with up to 37 qubits (on AWS)
- Can simulate arbitrary 1 & 2 qubit noise by definition of Kraus maps

```
def damping_channel(p=.01):
    damping_op = np.array([[0, sqrt(p)],
                           [0,   0    ]])
    residual_kraus = np.diag([1, sqrt(1-p)])
    return [residual_kraus, damping_op]

# overload identity gate I on qc 0 with p=1% damping probability
p.define_noisy_gate("I", [0], damping_channel(0.01))
```

- Compilation layer
  - Specify arbitrary circuit and compile to natural gate set
  - Compression optimizations build in
  - Simple layout optimization: Quantum Circuit to Chip Topology
  - E.g. compiled RB circuit to identity

- Simple prototyping before requesting QPU

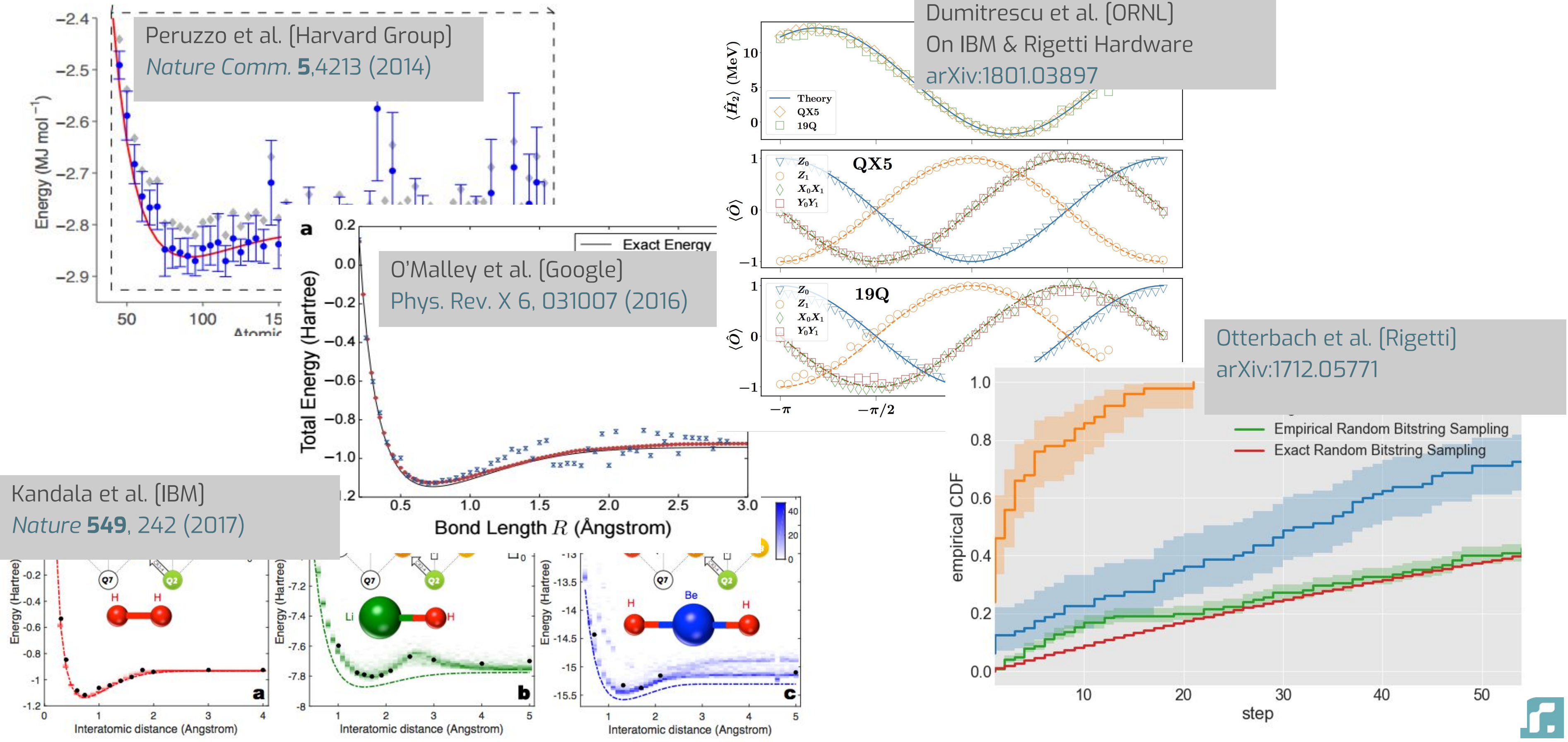
```
from pyquil.api import QVMConnection

cxn = QVMConnection()
cxn.run_and_measure(prog, ...)
```

```
from pyquil.api import QPUConnection

cxn = QPUConnection("19Q-Acorn")
cxn.run_and_measure(prog, ...)
```

NISQ Applications

NISQ - Near-term Intermediate Scale Quantum

Preskill, arXiv:[1801.00862]



Peruzzo et al. (Harvard Group)
Nature Comm. **5**,4213 (2014)

O'Malley et al. (Google)
Phys. Rev. X 6, 031007 (2016)

Dumitrescu et al. (ORNL)
On IBM & Rigetti Hardware
arXiv:1801.03897

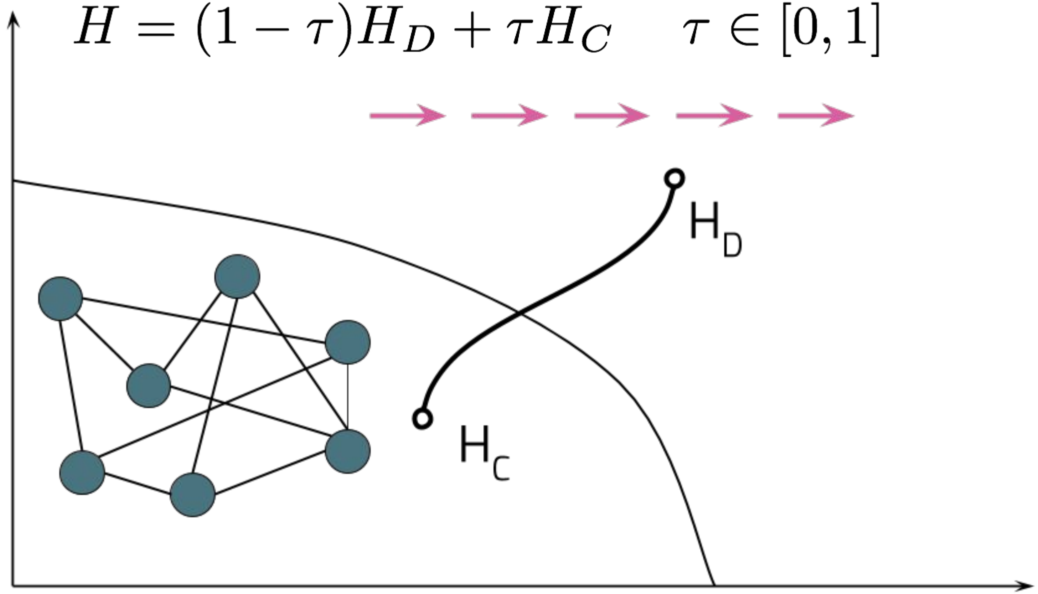
Otterbach et al. (Rigetti)
arXiv:1712.05771

Kandala et al. (IBM)
Nature **549**, 242 (2017)

QAOA - Quantum Approximate Optimization Algorithm

- Motivated through Adiabatic Quantum Computing
- Encode solution to hard NP-complete problem in ground-state of H_C
- Start at easy to prepare initial state with Hamiltonian H_D

$$H = (1-\tau)H_D + \tau H_C \quad \tau \in [0,1]$$



- Typical examples are graph problems
  - Traveling Salesman
  - Portfolio Optimization (Knapsack)
  - N-SAT
  - Sudoku (Exact Cover)

$$H_D = \sum_i \sigma_i^x$$

$$H_C = \sum_{i,j} w_{ij}\sigma_i^z \sigma_j^z + \sum_i h_i \sigma_i^z$$

Trotterization

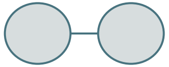
- Gate model of AQC (Farhi, Goldstone, Gutman, arxiv:1411.4028)

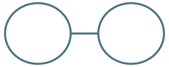
$$
\begin{aligned}
U &= \mathrm{e}^{-i((1-\tau)H_D+\tau H_C)t} \\
&= \lim_{p\to\infty} [\mathrm{e}^{-i(1-\tau)H_D t/p}\mathrm{e}^{-i\tau H_C t/p}]^p \\
&\to \prod_{p=1}^{\infty} \mathrm{e}^{-i\beta_p H_D}\mathrm{e}^{-i\gamma_p H_C}
\end{aligned}
$$

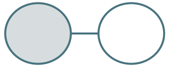
- Angles β_p, γ_p need not be small
- Theory for $p\to\infty$ is exact; in practice small p is already good
- No specification on how to find optimal β_p, γ_p

Example: Maxcut

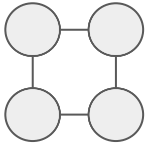
"Maximize disagreement on a colored graph"

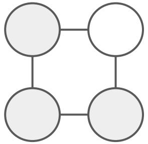

Score 0

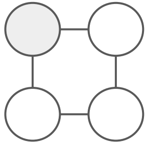

Score 0

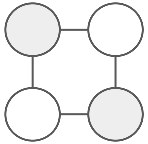

Score+1

4-node "ring of disagrees"


Score 0


Score 2


Score 2

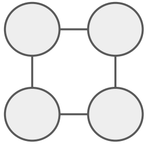
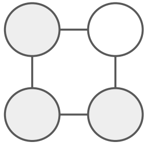
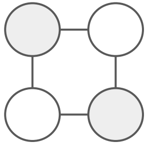

Score 4 (max)

Example: Maxcut

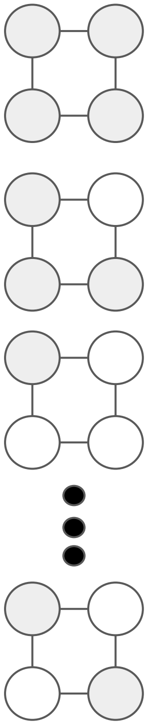
- Initial state is ground state of H_D : $$| \rightarrow \rangle = H^{\otimes n} |0\rangle$$
- Run the QAOA prescription:

$$|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle = \prod_{p=1}^{\infty} U_p V_p \; H^{\otimes n} |0\rangle$$

- Intuitively: Superposition of bitstring configurations

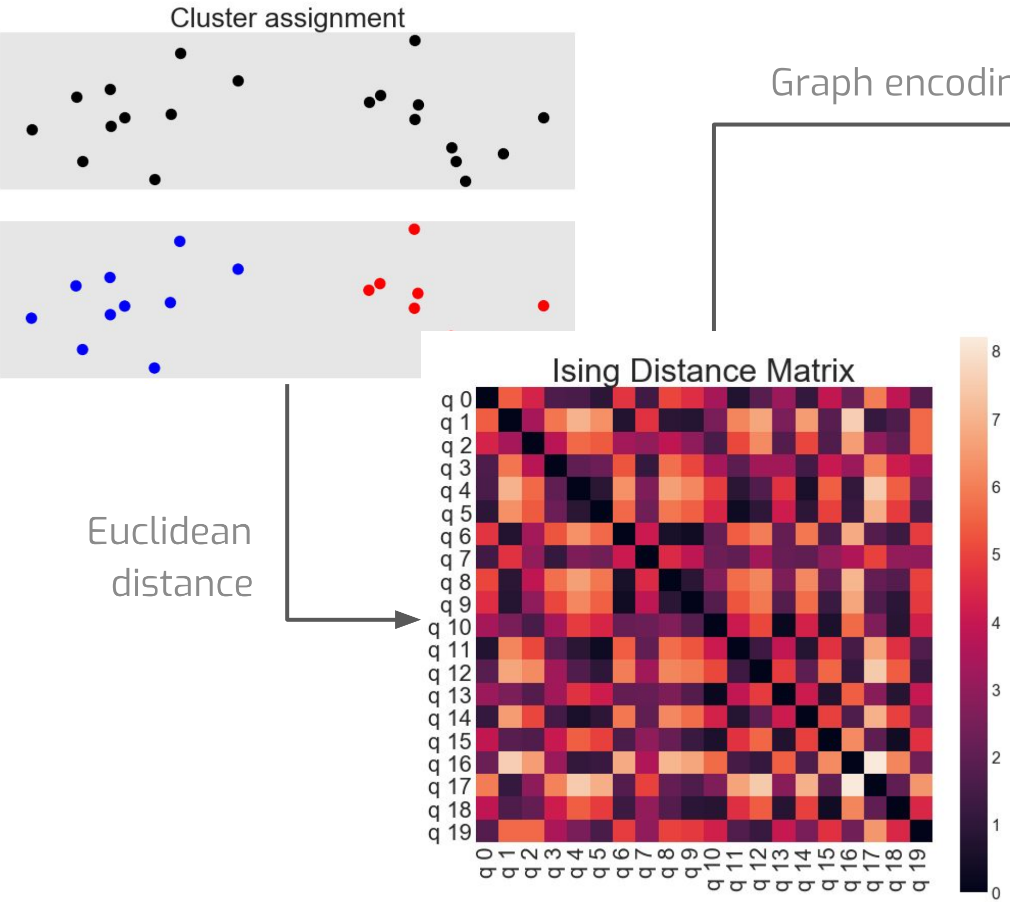
$| \beta, \gamma \rangle = \sqrt{p_1}$  $+ \sqrt{p_2}$  $+ \ldots + \sqrt{p_{16}}$ 

Probability distributions over bit strings



Cost	Choice $\vert \beta_1, \gamma_1 \rangle$	Choice $\vert \beta_2, \gamma_2 \rangle$	Choice $\vert \beta_3, \gamma_3 \rangle$
0	p = 0.1	p = 0.01	p = 0.2
2	p = 0.3	p = 0.15	p = 0.01
2	p = 0.05	p = 0.2	p = 0.0
4	p = 0.01	p = 0.51	p = 0.25

(Weighted) Maxcut as Clustering



Graph encoding

Hamiltonian

$$H_C = \frac{1}{2}\sum_{ij} w_{ij}(1 - \sigma_i^z \sigma_j^z)$$

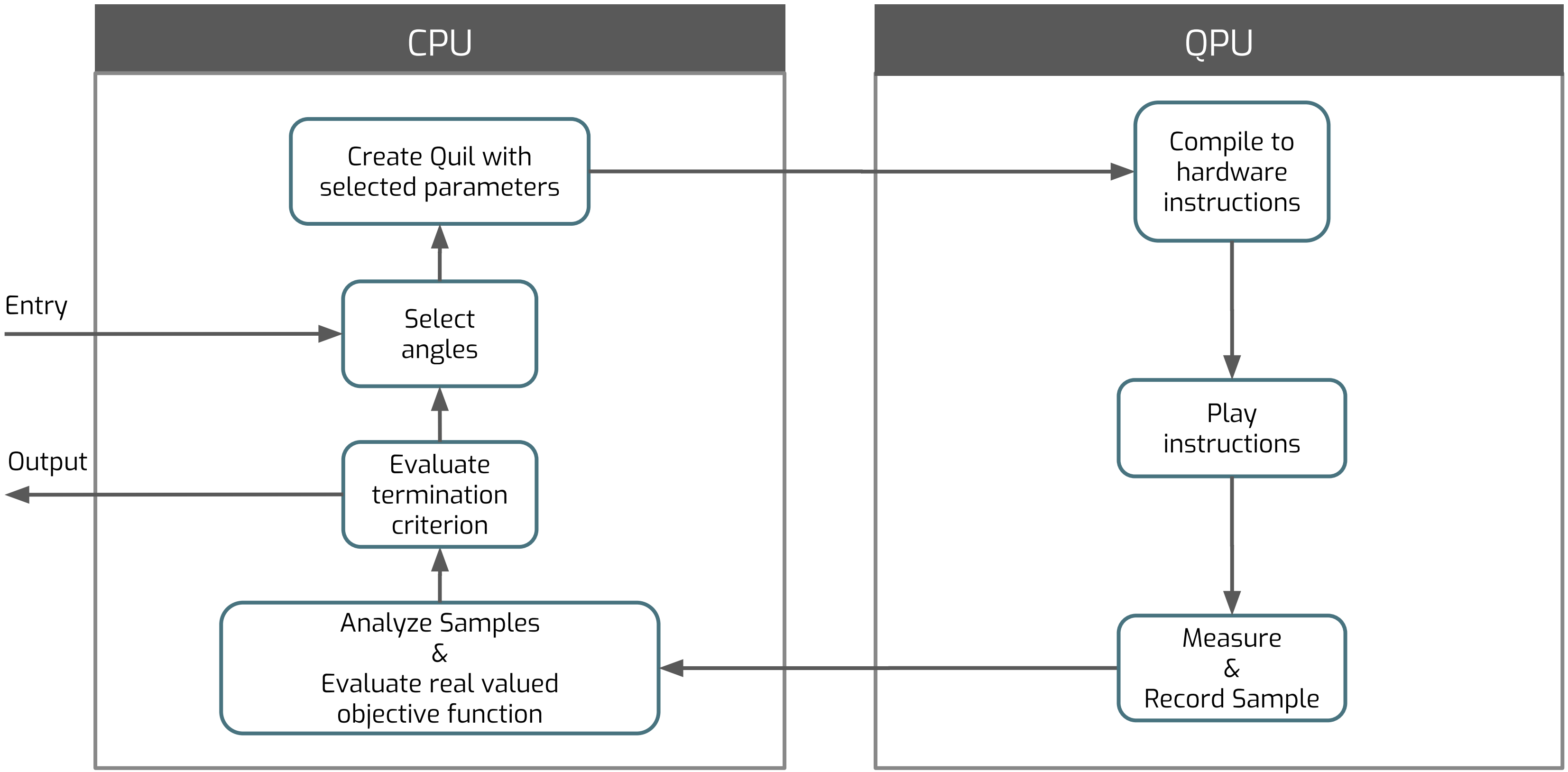
QAOA

$$|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle = \prod_{p=1}^{\infty} U_p V_p \, H^{\otimes n}|0\rangle$$

Measurement in computational basis

$$[0, 1, 0, 0, 0, 1, \ldots, 1]$$

When are we done?



Objective Function

- Loss/Reward Function:

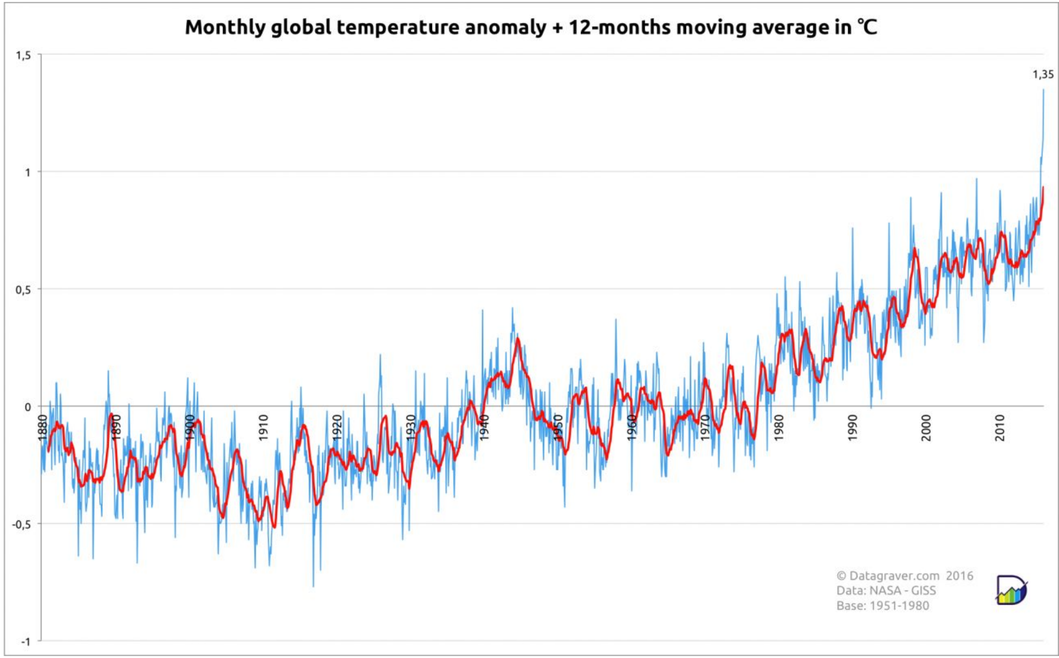
$$c_{\beta,\gamma} : \{0,1\}^n \mapsto \mathbb{R}$$

“quality of a sampled bit-string”

- Objective Function

$$f : X_{\beta,\gamma} \mapsto \mathbb{R},$$
$$(\beta,\gamma) \rightarrow \mathtt{STAT}_c(c;\beta,\gamma)$$

  - Extreme values
  - Mean
  - ...

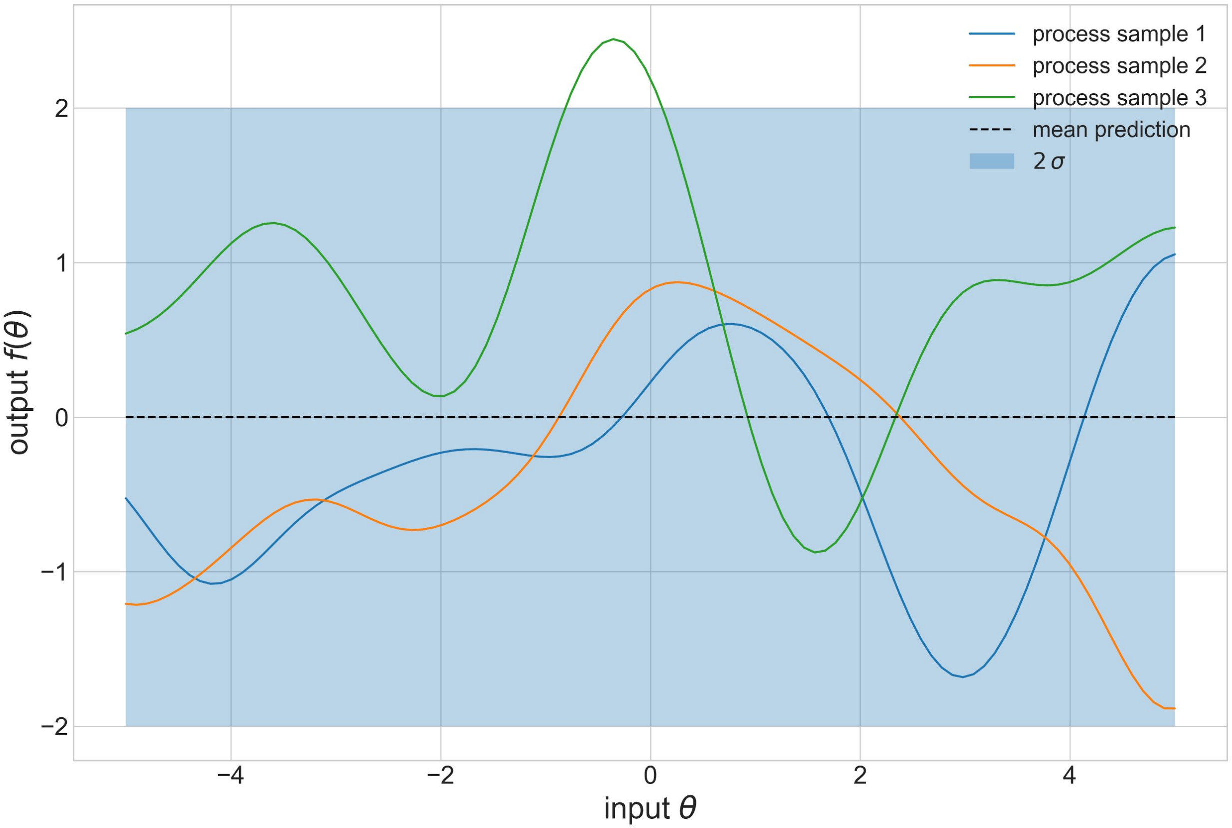


- Find the optimum of objective:
  - No easy access to gradients, need derivative free methods
  - Nelder-Mead
  - Bayesian Methods
  - ...

Image from https://datagraver.com/case/world-temperature-anomalies-for-februari-2016

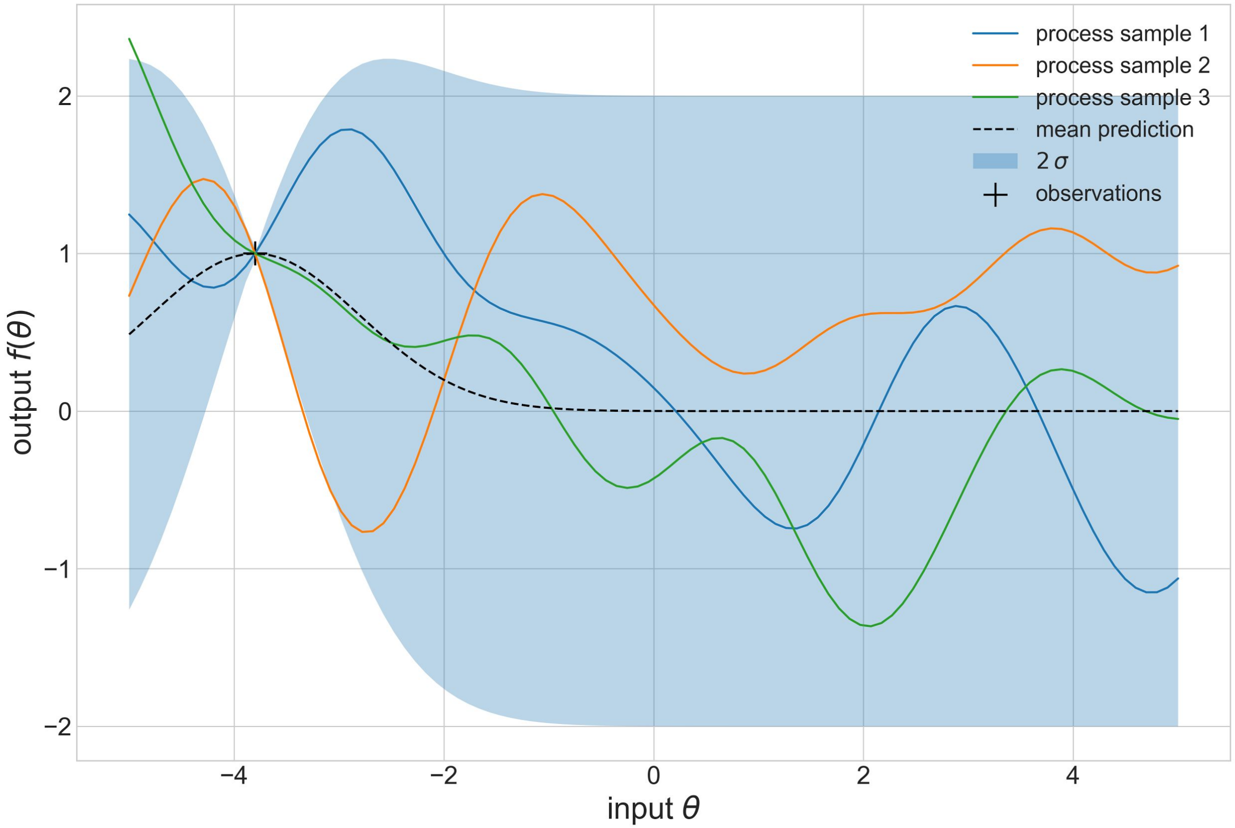
Bayesian Optimization

- Gaussian Process Prior of objective function



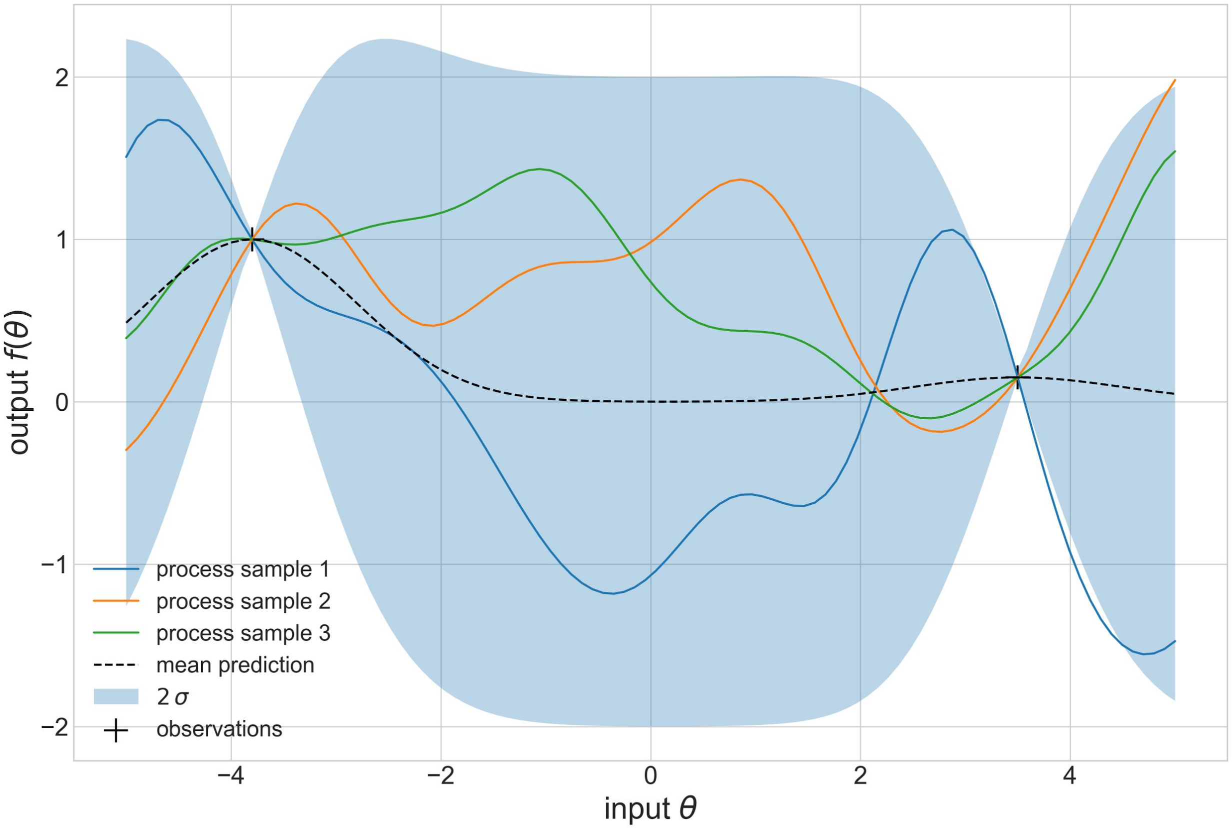
Bayesian Optimization

- Gaussian Process Prior of objective function
- Measure and update Prior



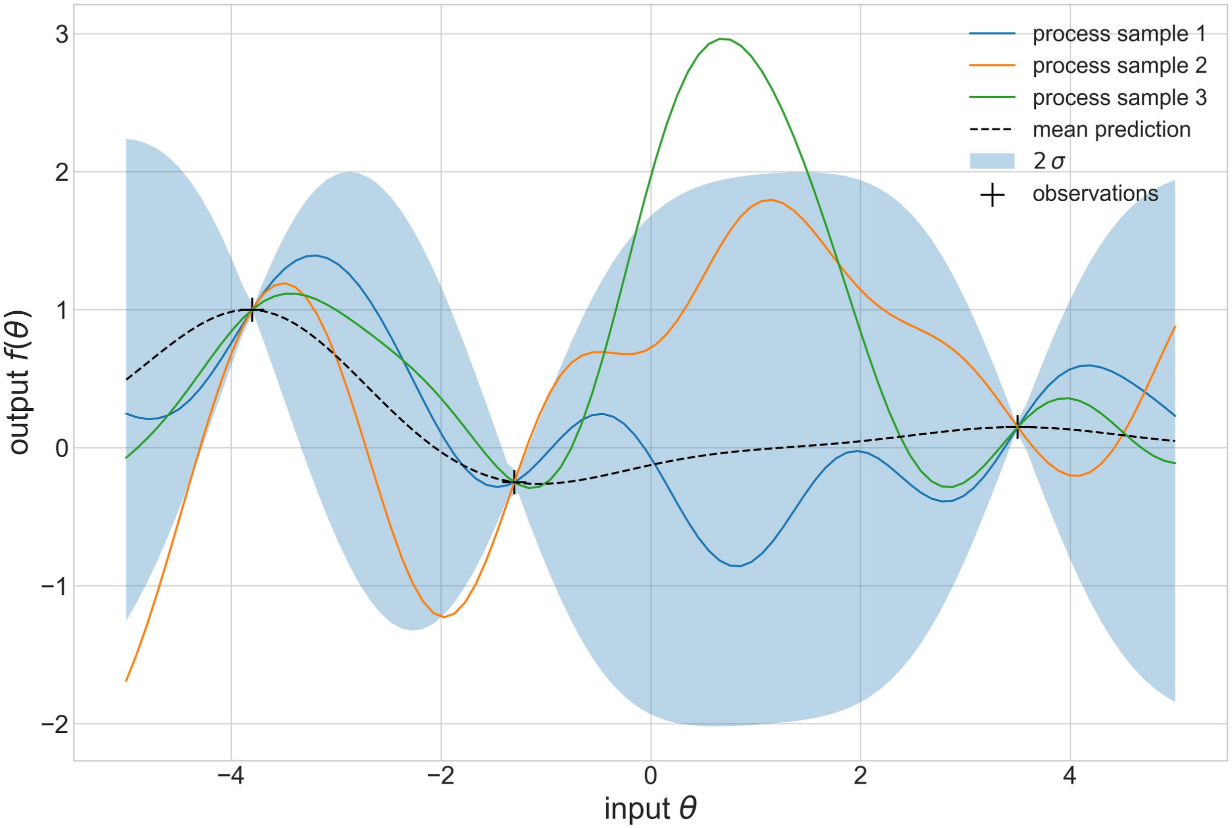
Bayesian Optimization

- Gaussian Process Prior of objective function
- Measure and update Prior
- Choose next point to measure and update



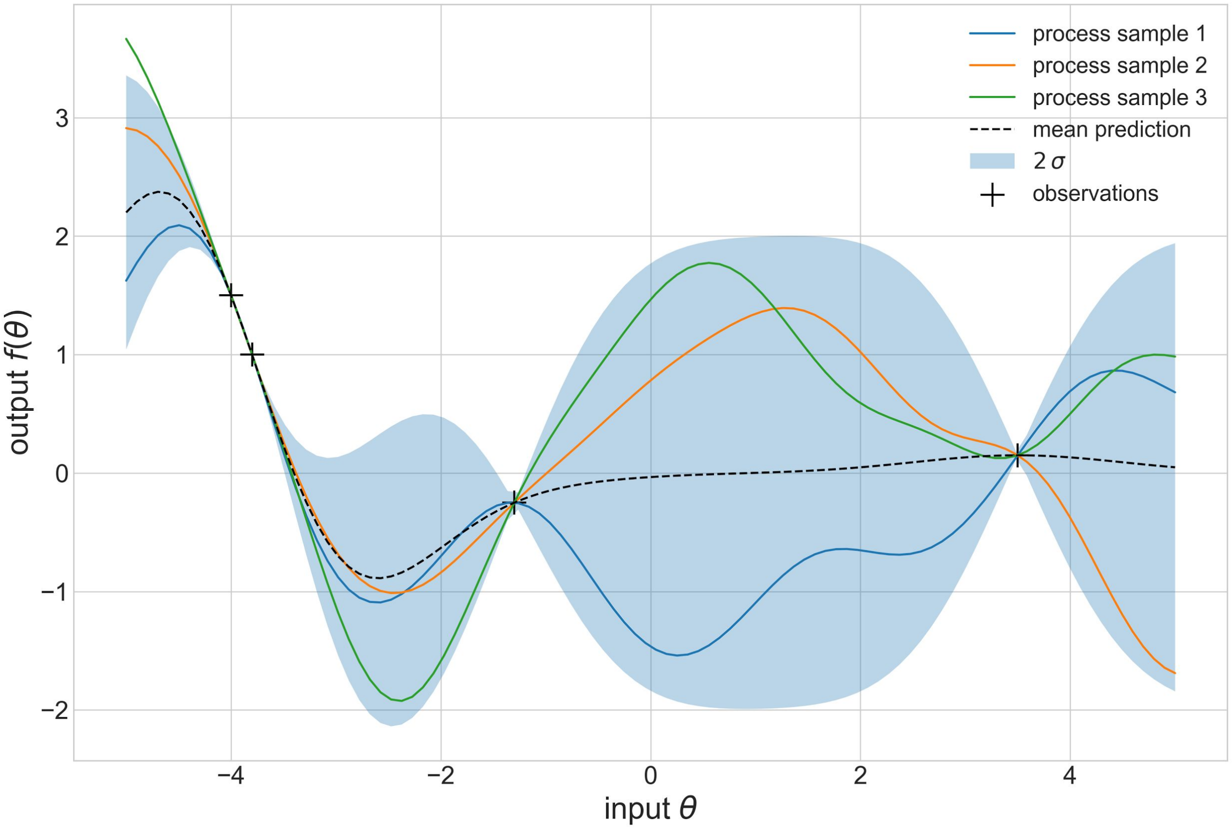
Bayesian Optimization

- Gaussian Process Prior of objective function
- Measure and update Prior
- Choose next point to measure and update
- Again



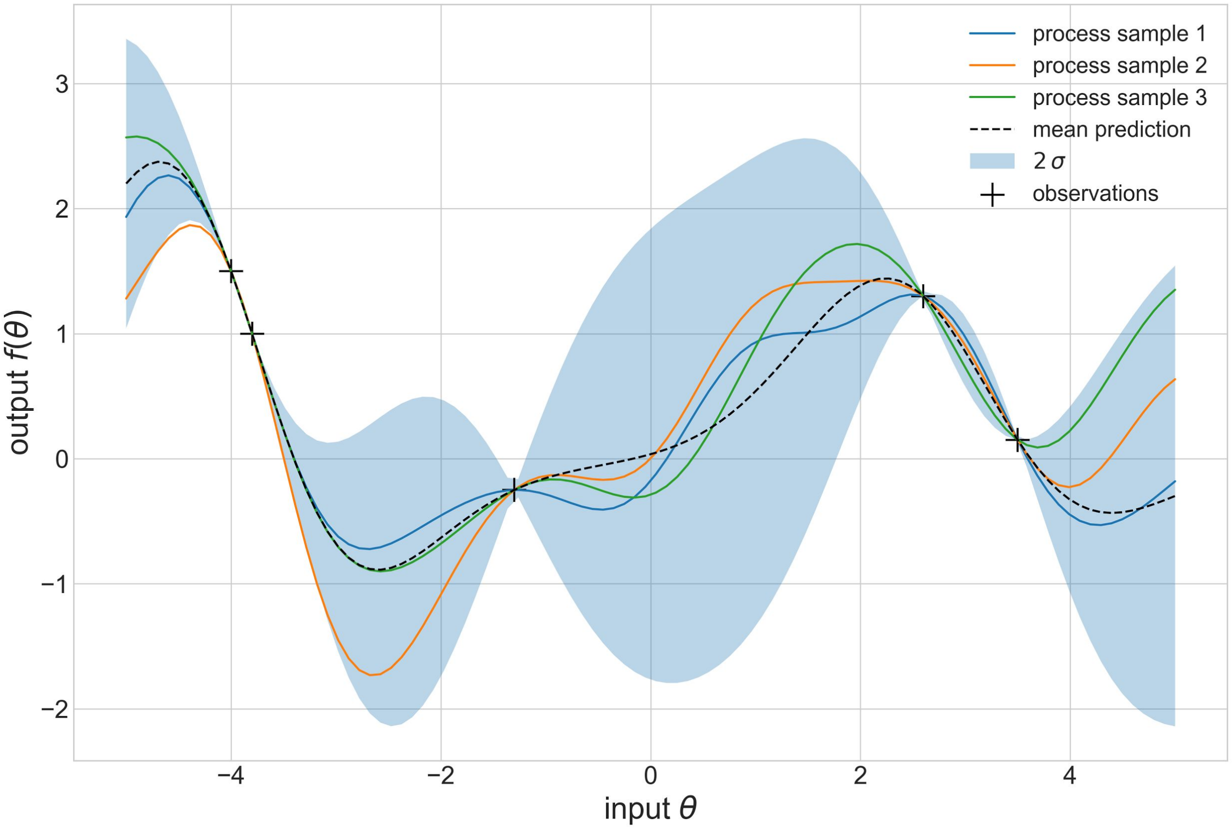
Bayesian Optimization

- Gaussian Process Prior of objective function
- Measure and update Prior
- Choose next point to measure and update
- Again
- ...



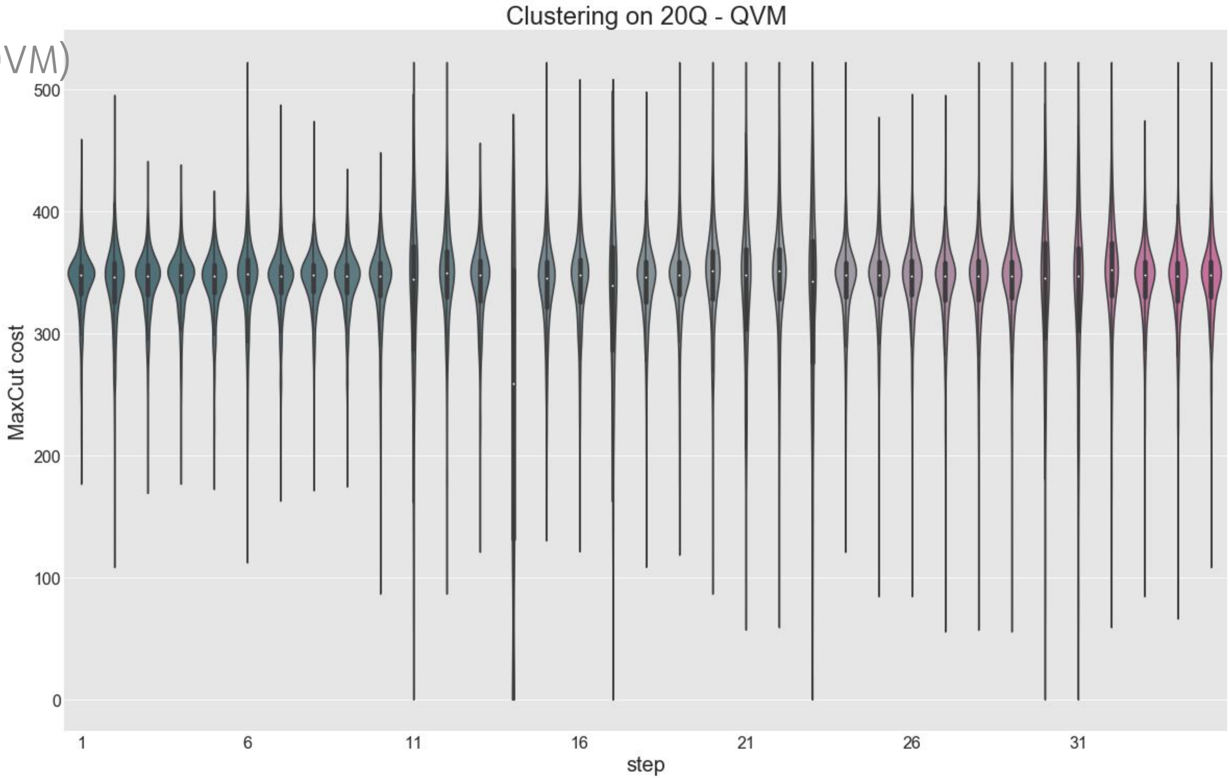
Bayesian Optimization

- Gaussian Process Prior of objective function
- Measure and update Prior
- Choose next point to measure and update
- Again
- ...
- ...



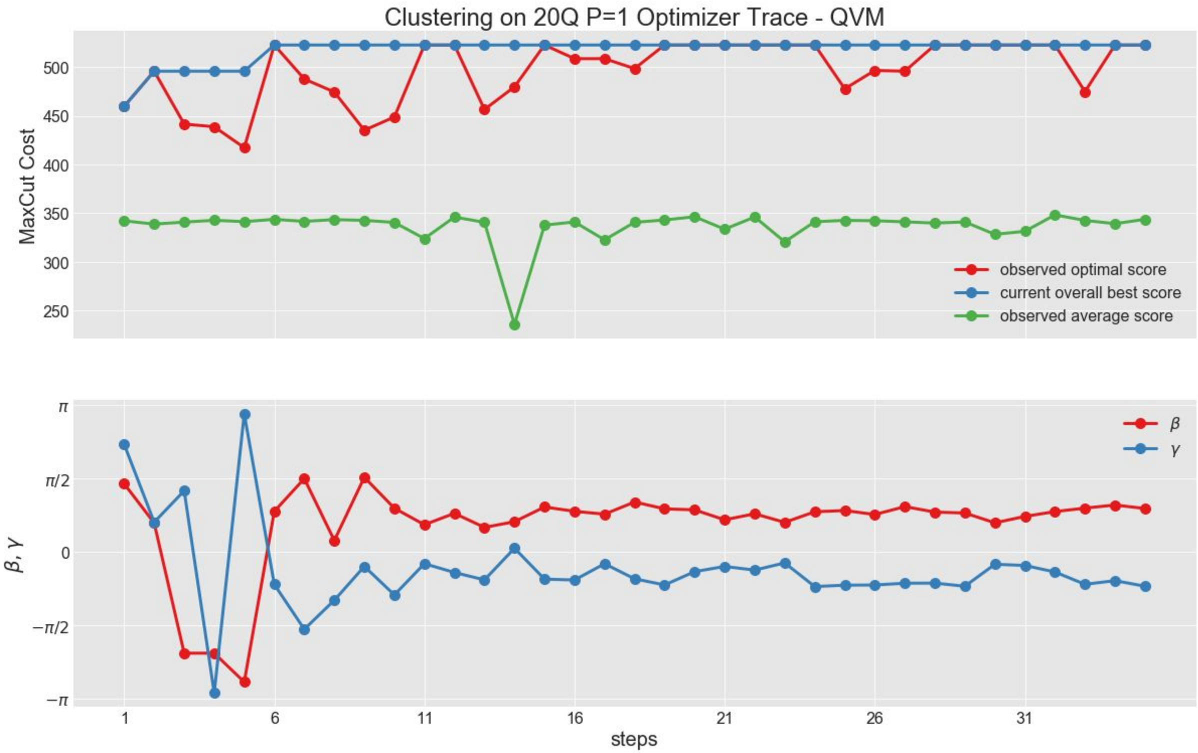
That's how it looks in practice

- Fully Connected Graph for clustering
- Noiseless Simulator (Rigetti QVM)
- p=1 QAOA



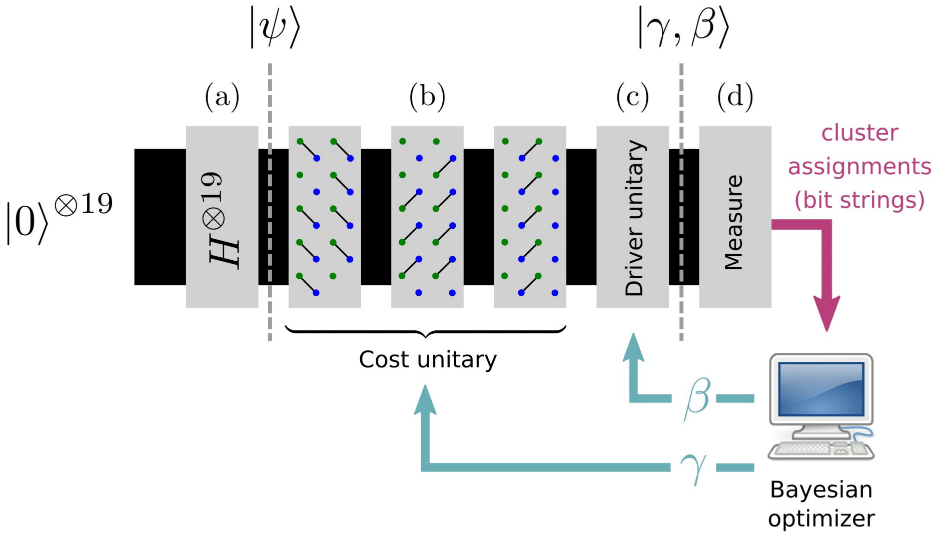
That's how it looks in practice

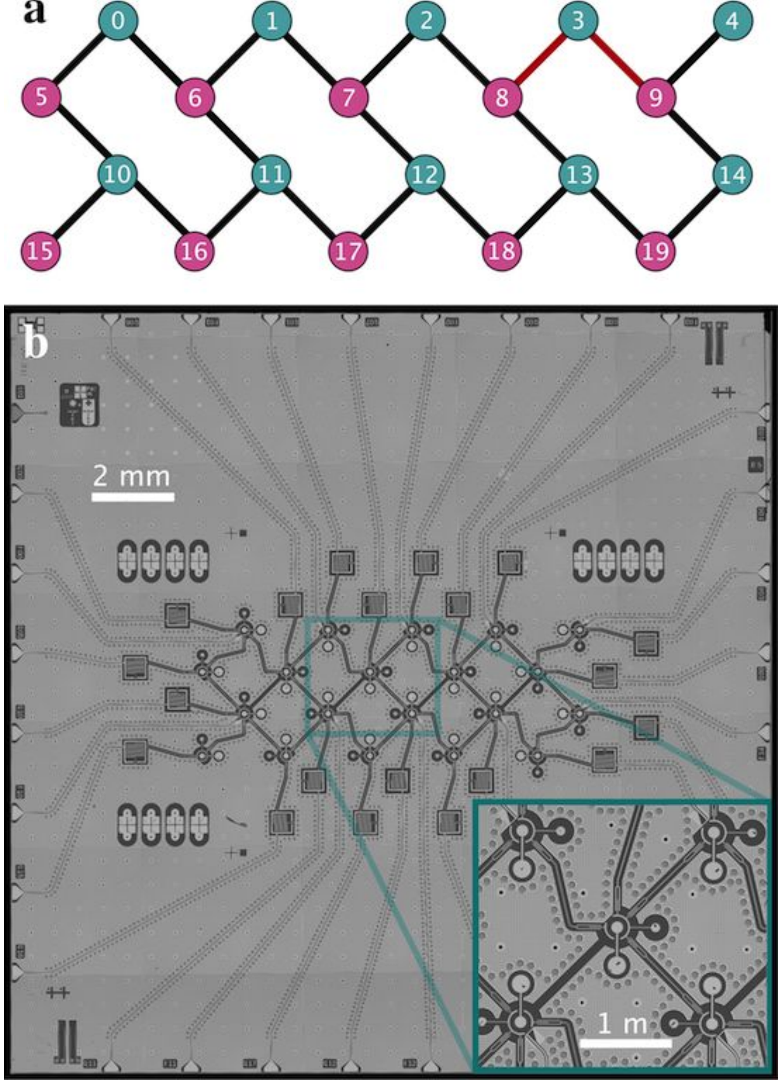
- Fully Connected Graph for clustering
- Noiseless Simulator (Rigetti QVM)
- p=1 QAOA



Clustering on a 19-Q Chip

- Chip topology requires smart gate sequence for QAOA to execute all gates on a vertex
- Staggering gate applications according to edge-coloring



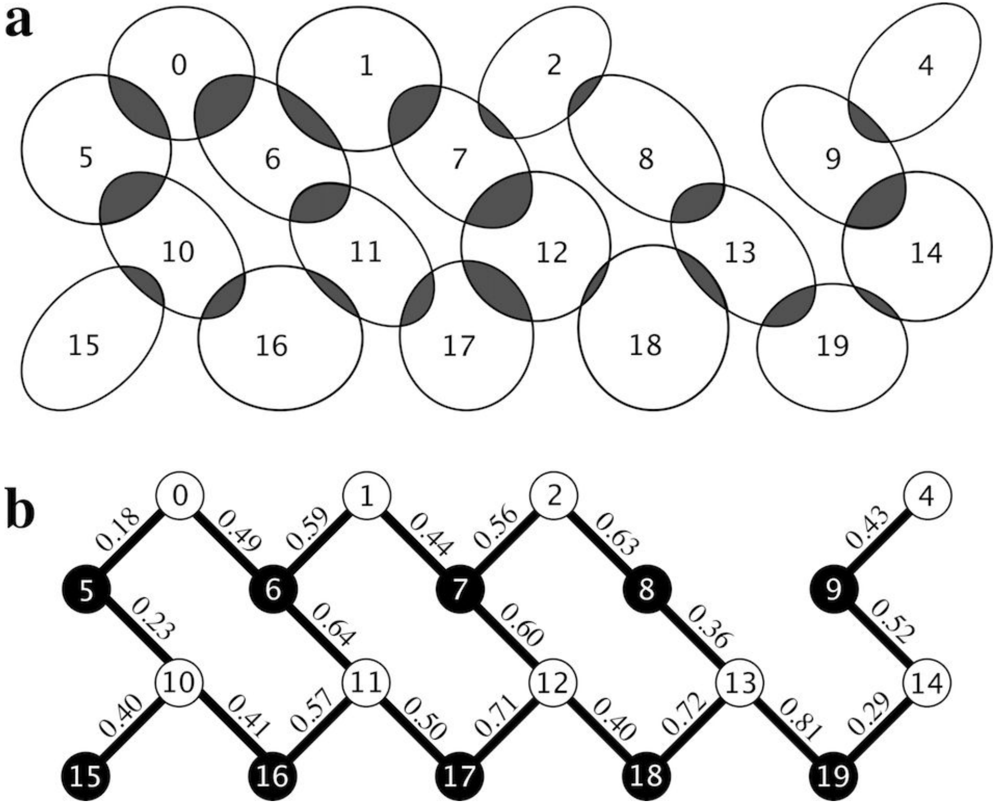


Clustering on a 19-Q Chip

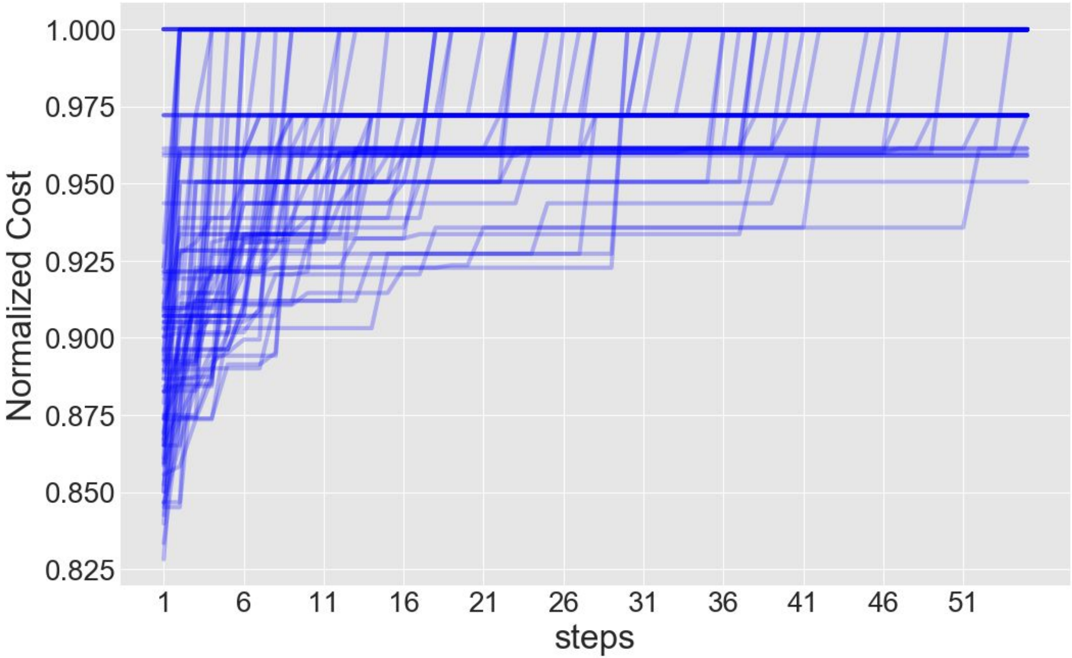
- Moderate coherence times
- Moderate 2Q fidelities
- Moderate readout fidelities

- Demonstration with chip specific problem taylored to the topology

- *Overlap* problem similar to VLSI design

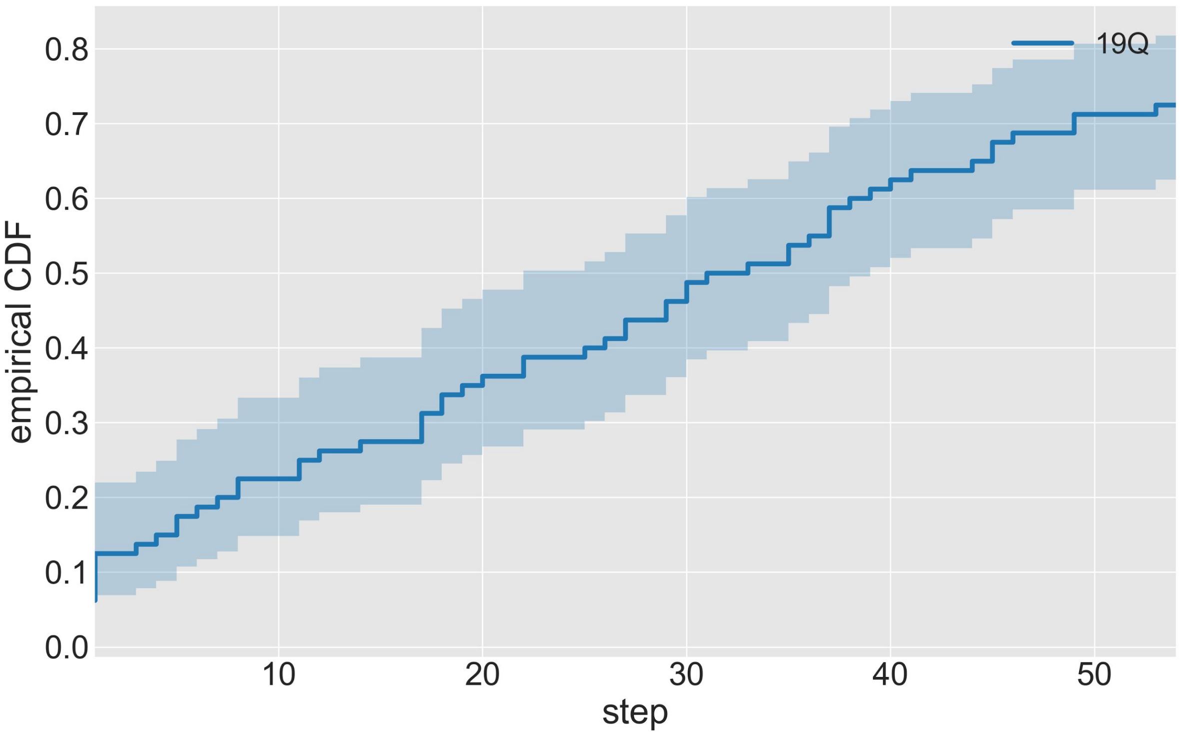


Putting it all together



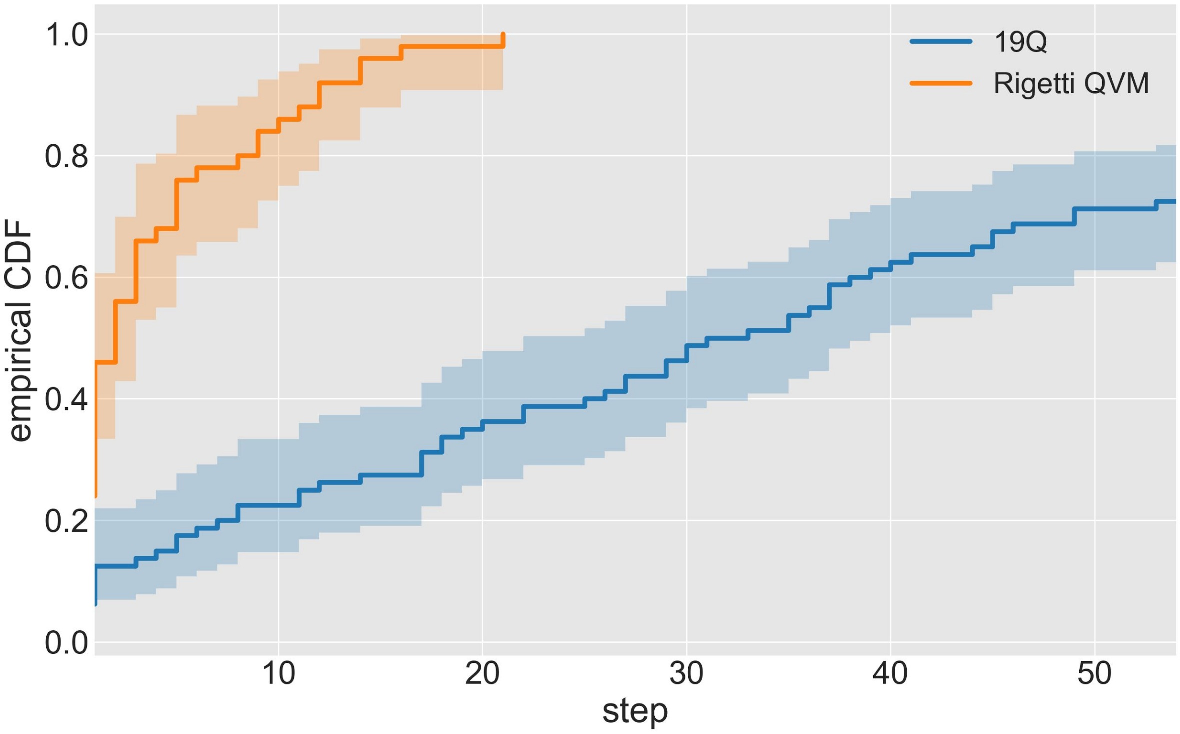
- 83 trial runs on the QPU
- p=1 QAOA, i.e. single application of U and V
- Algorithm finds the optimum most of the time
- Calculate eCDF form the traces

Empirical performance



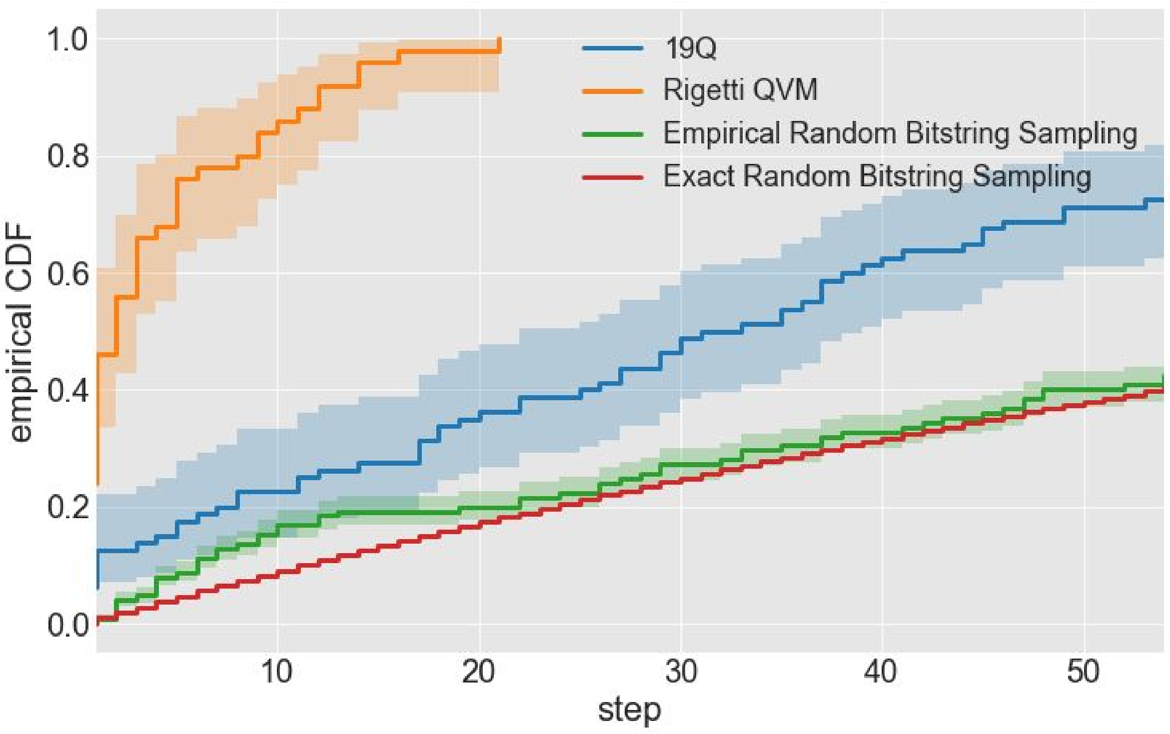
- Success probability monotonically increases with number of steps.

Empirical performance



- Success probability monotonically increases with number of steps.
- Noise in 19Q has a significant impact on performance.

Empirical performance



- Success probability monotonically increases with number of steps.
- Noise in 19Q has a significant impact on performance.
- Approach clearly outperforms random sampling.

Forest

Join our community Slack:

slack.rigetti.com

Find us on Github:

github.com/rigetticomputing

Sign-Up @ rigetti.com/forest

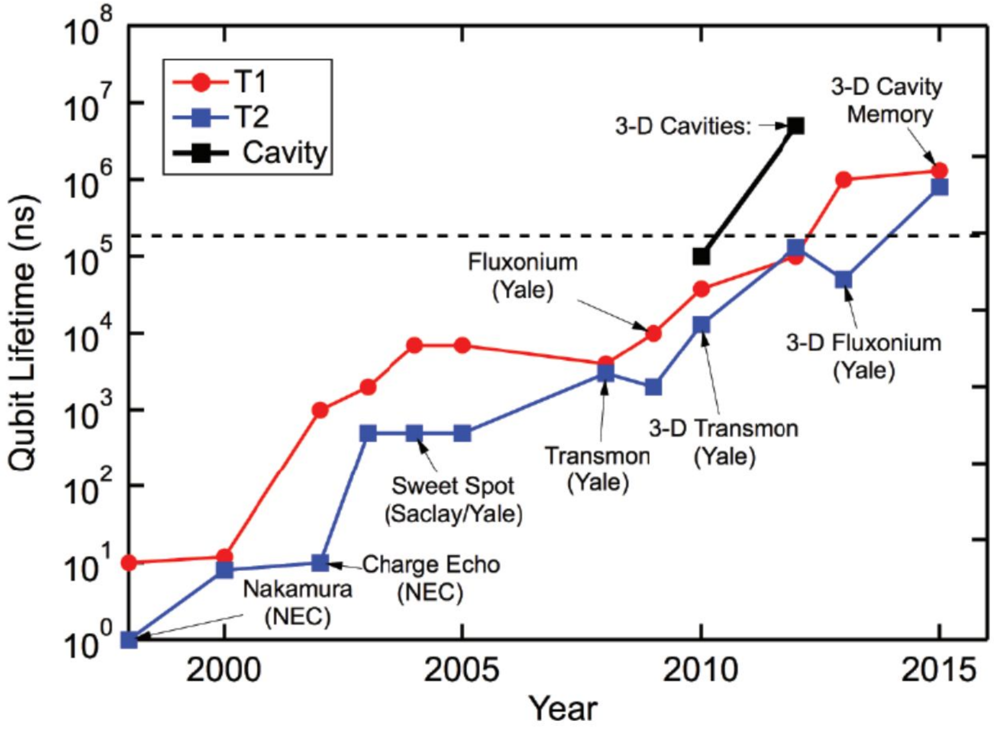
QPU access @ rigetti.com/qpu-request

Thank you

More details in our pre-print arXiv: 1712.05771

Spare slides

15 years of exponential performance improvement



M. Reagor thesis, 2015

Trends in “modern” superconducting qubits:

> All (or mostly) RF control
> Dispersive readout
> 3D cavity resonators

Quantum Fourier Transform (QFT)

Discrete Fourier Transform (DFT)

Fourier conjugates $\mathbf{q}, \mathbf{p}$ (vectors)

Vector: $\mathbf{q} = (q_0, q_1, ..., q_{N-1})$

N is a power of two

$$\text{DFT}[\mathbf{q}] = \mathbf{p}$$
$$\text{DFT}[\mathbf{p}] = \mathbf{q}$$

Quantum Fourier Transform

Fourier conjugates $|q\rangle, |p\rangle$ (state vectors)

State: $|q\rangle = q_0|0\rangle + q_1|1\rangle + ... + q_{N-1}|N-1\rangle$

(Basis vectors explicit)

N is a power of two:

- $N=2^n$ with *n* qubits

$$\text{QFT}|q\rangle = |p\rangle$$
$$\text{QFT}|q\rangle = |p\rangle$$

Both:

$$p_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} q_j e^{2\pi i jk/N}$$

Quantum Fourier Transform (QFT)

Quantum Fourier Transform

- Is unitary ✔
- Is faster than DFT
- Is an important subroutine of other algorithms

$$p_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} q_j e^{2\pi i j k / N}$$

Quantum Fourier Transform (QFT)

Quantum Fourier Transform

- Is unitary ✔
- Is faster than DFT
- Is an important subroutine of other algorithms

$$p_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} q_j e^{2\pi i jk/N}$$

$$|q\rangle = q_0|0\rangle + q_1|1\rangle + ... + q_N|N-1\rangle$$ Decimal labels

$$= q_{0...00}|0...00\rangle + q_{0...01}|0...01\rangle + ... + q_{11...1}|11...1\rangle$$ Binary labels

Quantum Fourier Transform (QFT)

Quantum Fourier Transform

- Is unitary ✔
- Is faster than DFT
- Is an important subroutine of other algorithms

$$p_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} q_j e^{2\pi ijk/N}$$

$$|q\rangle = q_0|0\rangle + q_1|1\rangle + \ldots + q_N|N-1\rangle$$ Decimal labels

$$= q_{0\ldots00}|0\ldots00\rangle + q_{0\ldots01}|0\ldots01\rangle + \ldots + q_{11\ldots1}|11\ldots1\rangle$$ Binary labels

Fact:

$$\mathrm{QFT}|j_n, \ldots, j_1\rangle = \frac{1}{\sqrt{2^n}}\Big(|0\rangle + e^{i\varphi_n}|1\rangle\Big) \otimes \ldots \otimes \Big(|0\rangle + e^{i\varphi_1}|1\rangle\Big)$$

$$\varphi_n \equiv 2\pi(j_1/2^n + j_2/2^{n-1} + \ldots + j_n/2)$$

Quantum Fourier Transform (QFT)

Concrete example: $|q\rangle = |j_3 j_2 j_1\rangle = |101\rangle$

$$\varphi_n \equiv 2\pi(j_1/2^n + j_2/2^{n-1} + ... + j_n/2)$$

Quantum Fourier Transform (QFT)

Concrete example: $|q\rangle = |j_3 j_2 j_1\rangle = |101\rangle$

$$\varphi_n \equiv 2\pi(j_1/2^n + j_2/2^{n-1} + ... + j_n/2)$$

$$\mathrm{QFT}|101\rangle = \frac{1}{\sqrt{2^n}}\Big(|0\rangle + e^{i2\pi(5/8)}|1\rangle\Big) \otimes \Big(|0\rangle + e^{i2\pi(1/4)}|1\rangle\Big) \otimes \Big(|0\rangle + e^{i2\pi(1/2)}|1\rangle\Big)$$

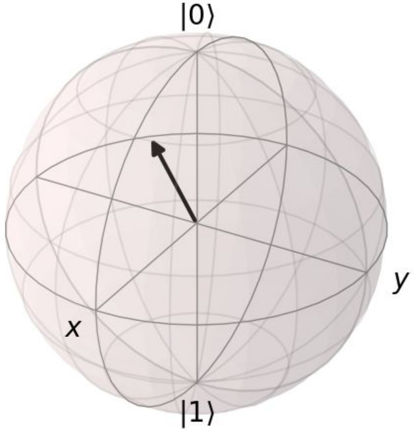
Quantum Fourier Transform (QFT)

Concrete example: $|q\rangle = |j_3 j_2 j_1\rangle = |101\rangle$

$$\varphi_n \equiv 2\pi(j_1/2^n + j_2/2^{n-1} + ... + j_n/2)$$

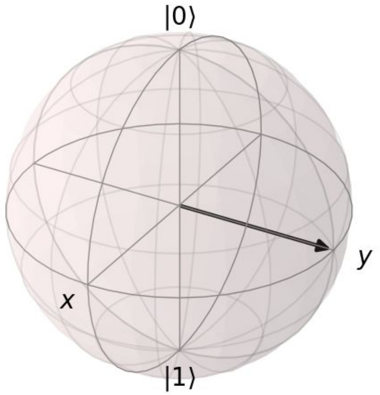
$$\text{QFT}|101\rangle = \frac{1}{\sqrt{2^n}}\Big(|0\rangle + e^{i2\pi(5/8)}|1\rangle\Big) \otimes \Big(|0\rangle + e^{i2\pi(1/4)}|1\rangle\Big) \otimes \Big(|0\rangle + e^{i2\pi(1/2)}|1\rangle\Big)$$

$$|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\varphi}\sin(\theta/2)|1\rangle$$



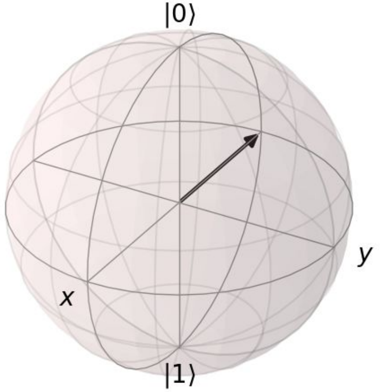
$R_z(\varphi_3 = 5\pi/4)$

$\otimes$



$R_z(\varphi_2 = \pi/2)$

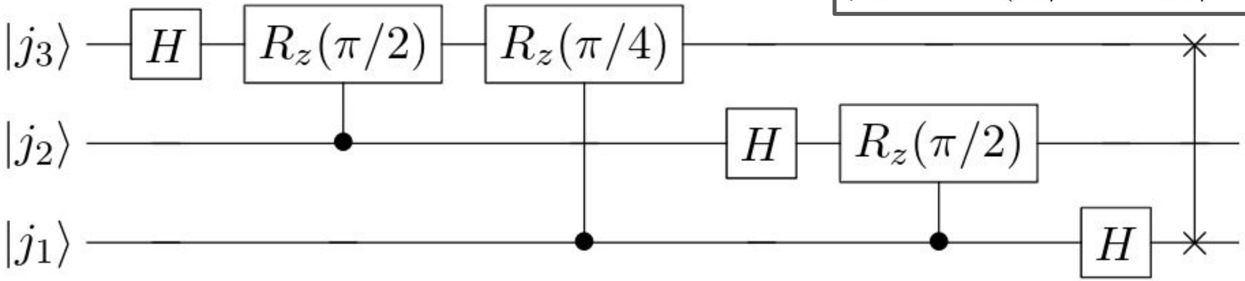
$\otimes$



$R_z(\varphi_1 = \pi)$

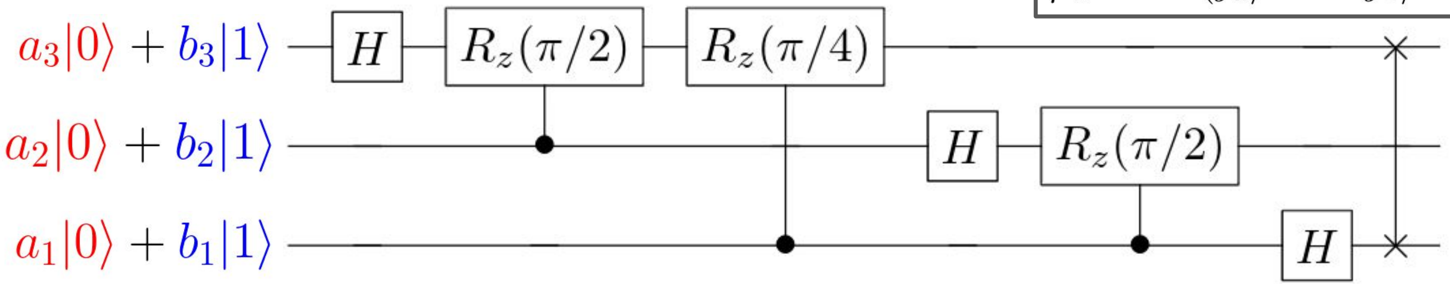
Quantum Fourier Transform (QFT)

$$\varphi_n \equiv 2\pi(j_1/2^n + j_2/2^{n-1} + \ldots + j_n/2)$$

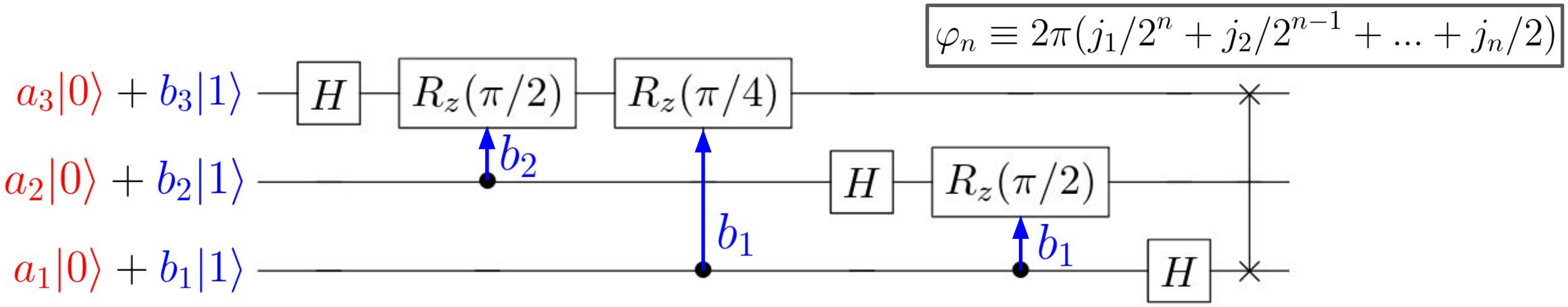


Quantum Fourier Transform (QFT)

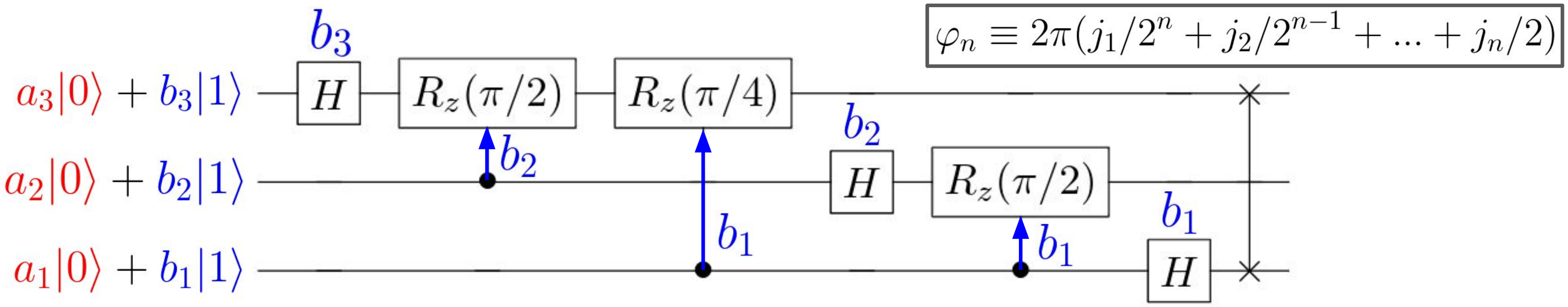
$$\varphi_n \equiv 2\pi(j_1/2^n + j_2/2^{n-1} + ... + j_n/2)$$



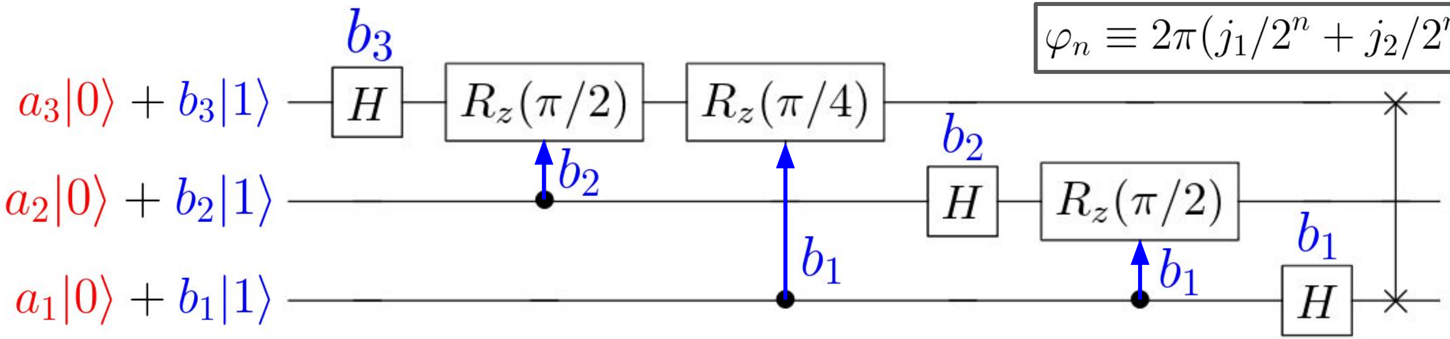
Quantum Fourier Transform (QFT)



Quantum Fourier Transform (QFT)



Quantum Fourier Transform (QFT)



$$\varphi_n \equiv 2\pi(j_1/2^n + j_2/2^{n-1} + ... + j_n/2)$$

$$p_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} q_j e^{2\pi i jk/N}$$

Fast Fourier Transform: $\Theta(n2^n)$

Quantum Fourier Transform: $\Theta(n^2)$

Quantum Fourier Transform (QFT)

$$p_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} q_j e^{2\pi i j k / N}$$

Quantum Fourier Transform

- Is unitary ✔
- Is faster than Fast Fourier Transform
- Is an important subroutine of other algorithms

Fast Fourier Transform: $\Theta(n2^n)$

Quantum Fourier Transform: $\Theta(n^2)$

Caveats:

- Can't directly read out p_k
  → use as subroutine
- State preparation of $|q\rangle$ is inefficient
  → restricted to simple initial states

Quantum Fourier Transform (QFT)

$$p_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} q_j e^{2\pi i jk/N}$$

Quantum Fourier Transform

- Is unitary ✔
- Is faster than Fast Fourier Transform
- Is an important subroutine of other algorithms

Fast Fourier Transform: $\Theta(n2^n)$

Quantum Fourier Transform: $\Theta(n^2)$

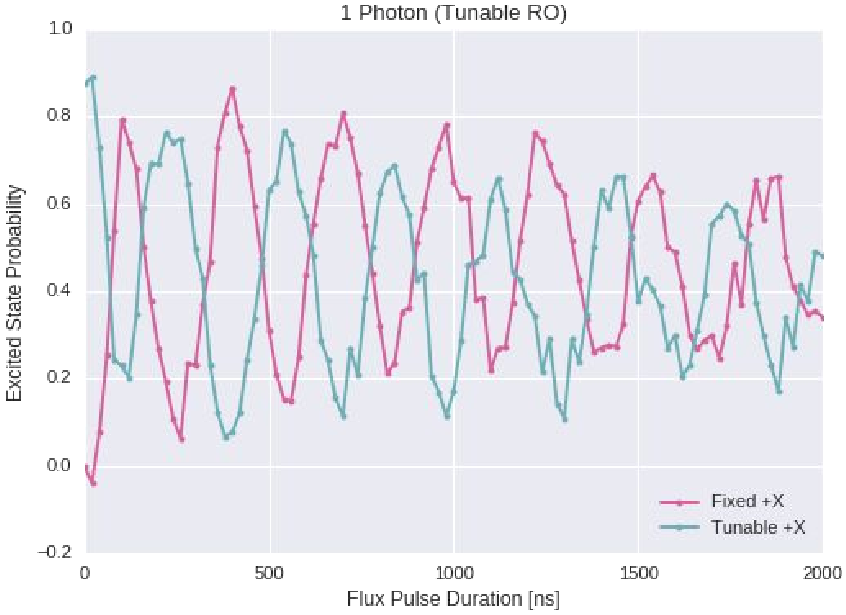
Caveats:

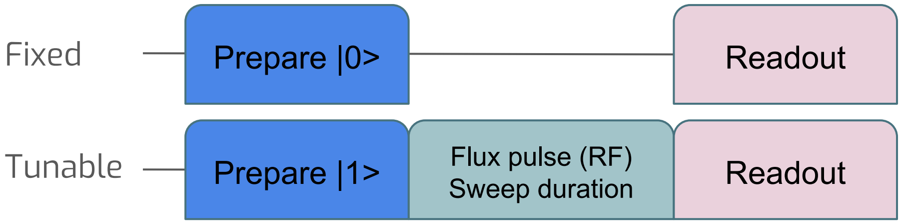
- Can't directly read out p_k
  → use as subroutine
- State preparation of $|q\rangle$ is inefficient
  → restricted to simple initial states

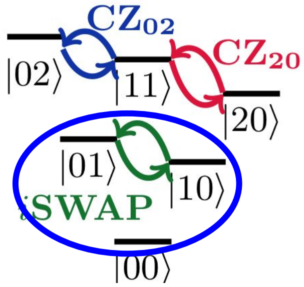
QFT descendants:

- Phase estimation
- Order finding
- Prime number factorization

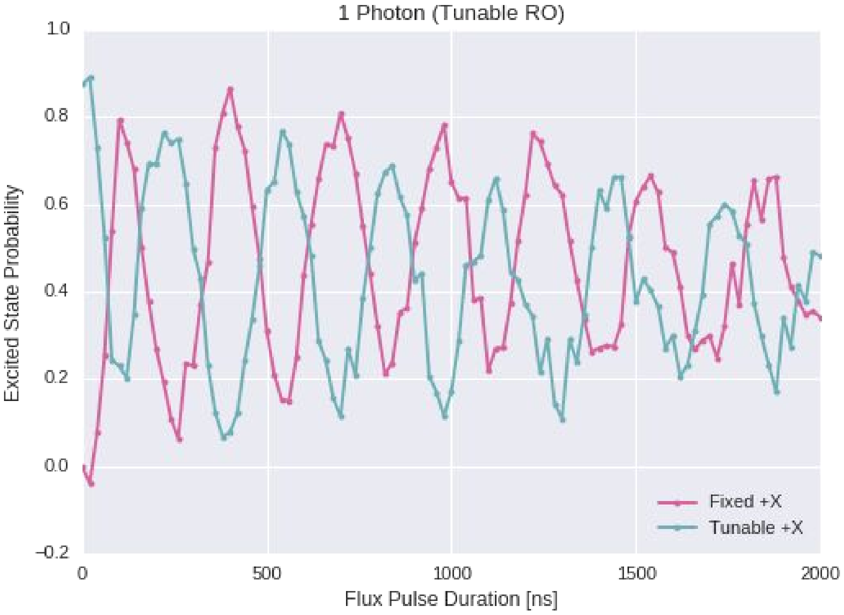
Proof of principle

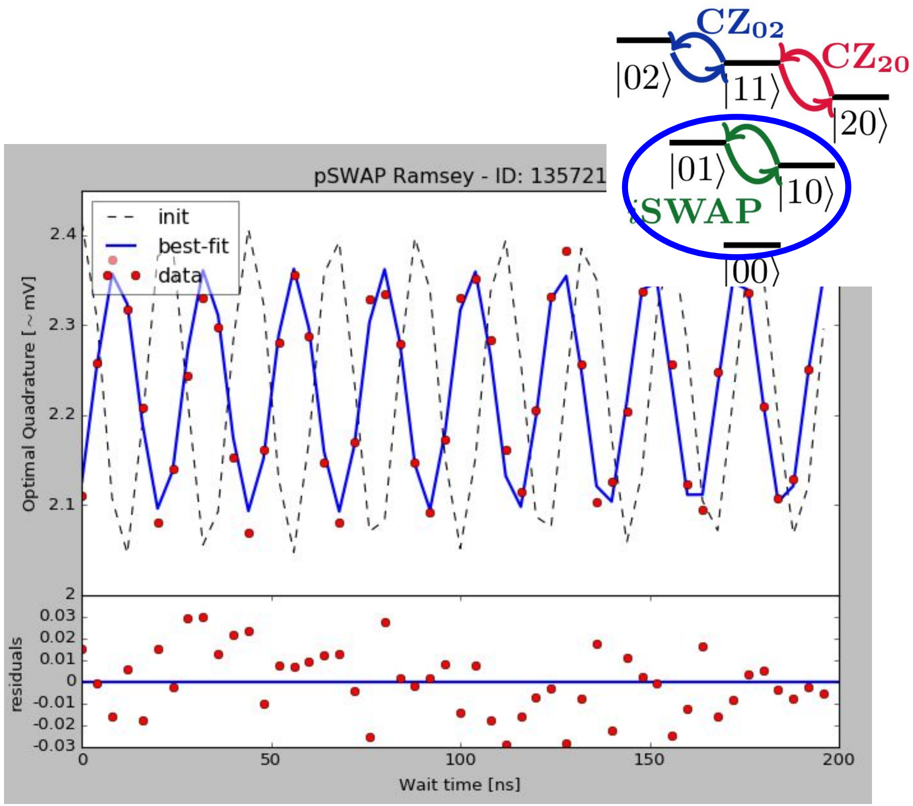


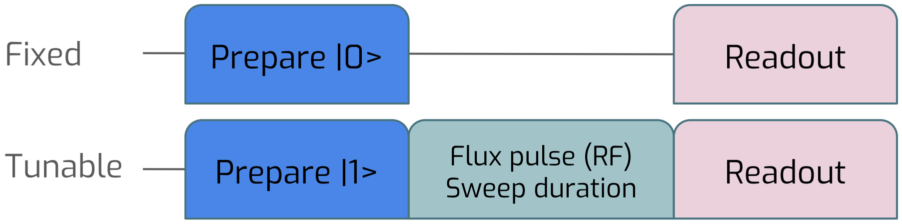


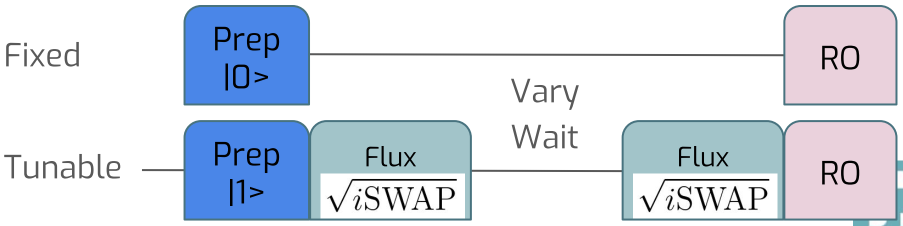


Proof of principle



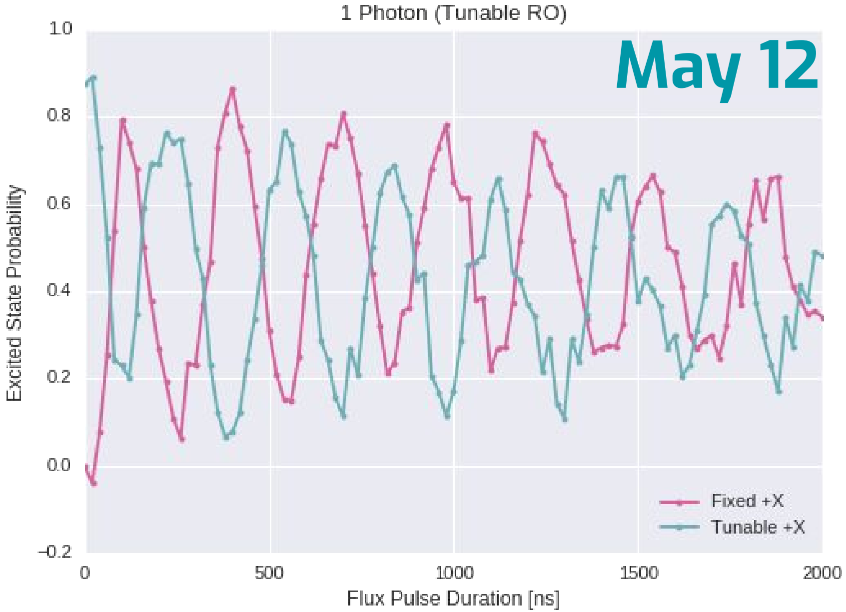


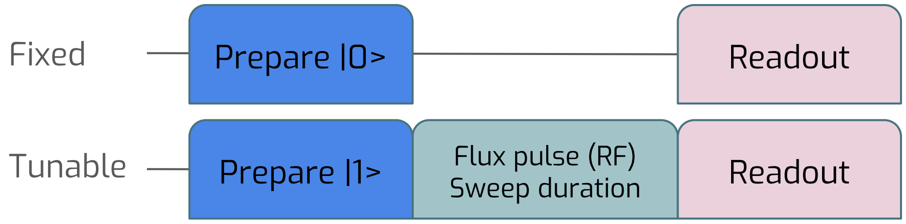




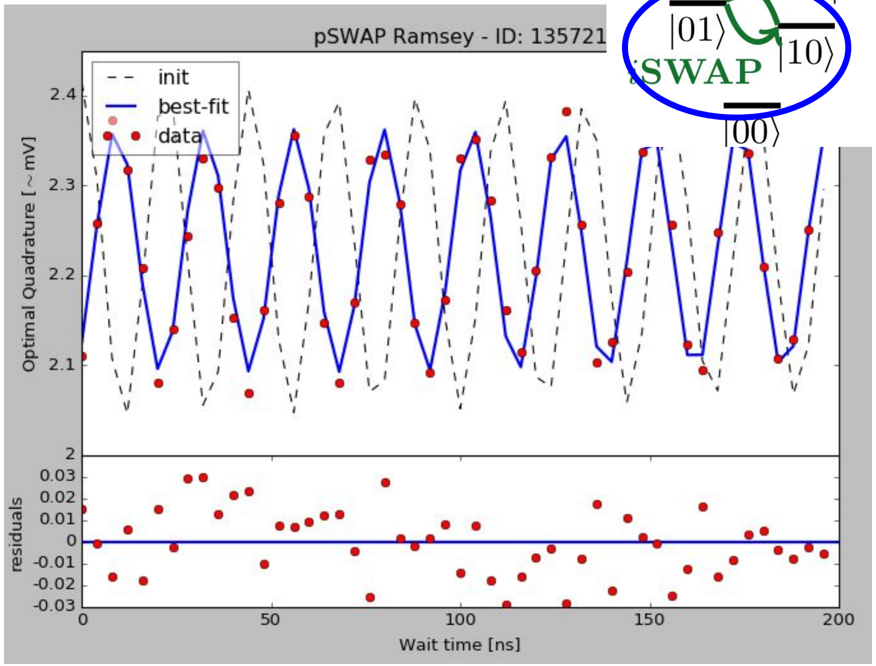
Proof of principle

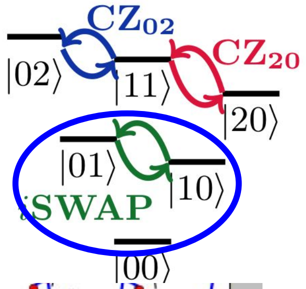
May 12

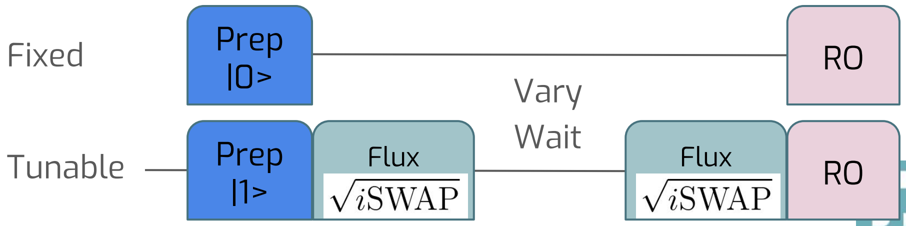




May 16



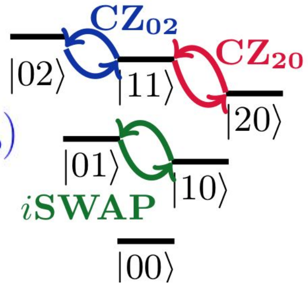




Theory

$$\Phi(t) = \overline{\Phi} + \widetilde{\Phi}\cos(\omega_p t + \theta_p)$$

$$\omega_T(t) \approx \overline{\omega_T}(\widetilde{\Phi}) + \widetilde{\omega_T}(\widetilde{\Phi})\cos(2\omega_p t + 2\theta_p)$$

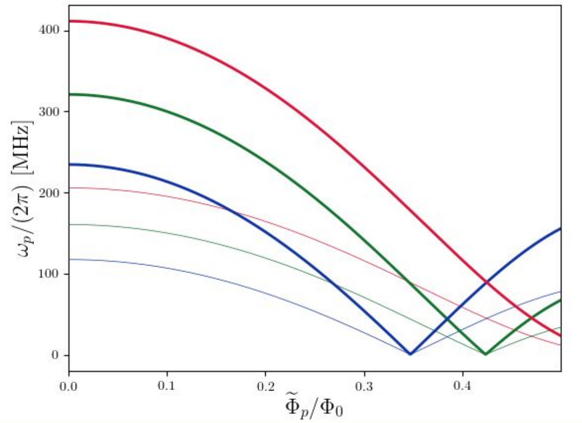


Resonant frequencies

$$f[\cos\phi_{\rm ext}(t)] \simeq \bar{f} + \tilde{f}\cos[2(\omega_p t + \theta_p)], \quad (34)$$

$$\bar{f} = f^{(0)}[\mathrm{J}_0(\widetilde{\phi}_p)] + \mathrm{J}_2^2(\widetilde{\phi}_p) f^{(2)}[\mathrm{J}_0(\widetilde{\phi}_p)] + \mathrm{J}_2^2(\widetilde{\phi}_p)\mathrm{J}_4(\widetilde{\phi}_p) f^{(3)}[\mathrm{J}_0(\widetilde{\phi}_p)] + \tfrac{1}{4}\mathrm{J}_2^4(\widetilde{\phi}_p) f^{(4)}[\mathrm{J}_0(\widetilde{\phi}_p)] \quad (35)$$

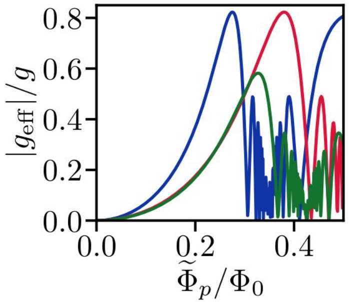
$$\tilde{f} = -2\mathrm{J}_2(\widetilde{\phi}_p)\{ f^{(1)}[\mathrm{J}_0(\widetilde{\phi}_p)] + \mathrm{J}_4(\widetilde{\phi}_p) f^{(2)}[\mathrm{J}_0(\widetilde{\phi}_p)] + \tfrac{1}{2}\mathrm{J}_2^2(\widetilde{\phi}_p) f^{(3)}[\mathrm{J}_0(\widetilde{\phi}_p)] + \tfrac{2}{3}\mathrm{J}_2^2(\widetilde{\phi}_p)\mathrm{J}_4(\widetilde{\phi}_p) f^{(4)}[\mathrm{J}_0(\widetilde{\phi}_p)]\}, \quad (36)$$



Effective couplings

$$g_{11}^{(n)} = \bar{g}_{11}\mathrm{J}_n\left(\frac{\widetilde{\omega}_{T_{01}}}{2\omega_p}\right) - \tfrac{1}{2}\widetilde{g}_{11}\left[\mathrm{J}_{n-1}\left(\frac{\widetilde{\omega}_{T_{01}}}{2\omega_p}\right) + \mathrm{J}_{n+1}\left(\frac{\widetilde{\omega}_{T_{01}}}{2\omega_p}\right)\right], \quad (41)$$

$$g_{21}^{(n)} = \bar{g}_{21}\mathrm{J}_n\left(\frac{\widetilde{\omega}_{T_{01}}}{2\omega_p}\right) - \tfrac{1}{2}\widetilde{g}_{21}\left[\mathrm{J}_{n-1}\left(\frac{\widetilde{\omega}_{T_{01}}}{2\omega_p}\right) + \mathrm{J}_{n+1}\left(\frac{\widetilde{\omega}_{T_{01}}}{2\omega_p}\right)\right], \quad (42)$$



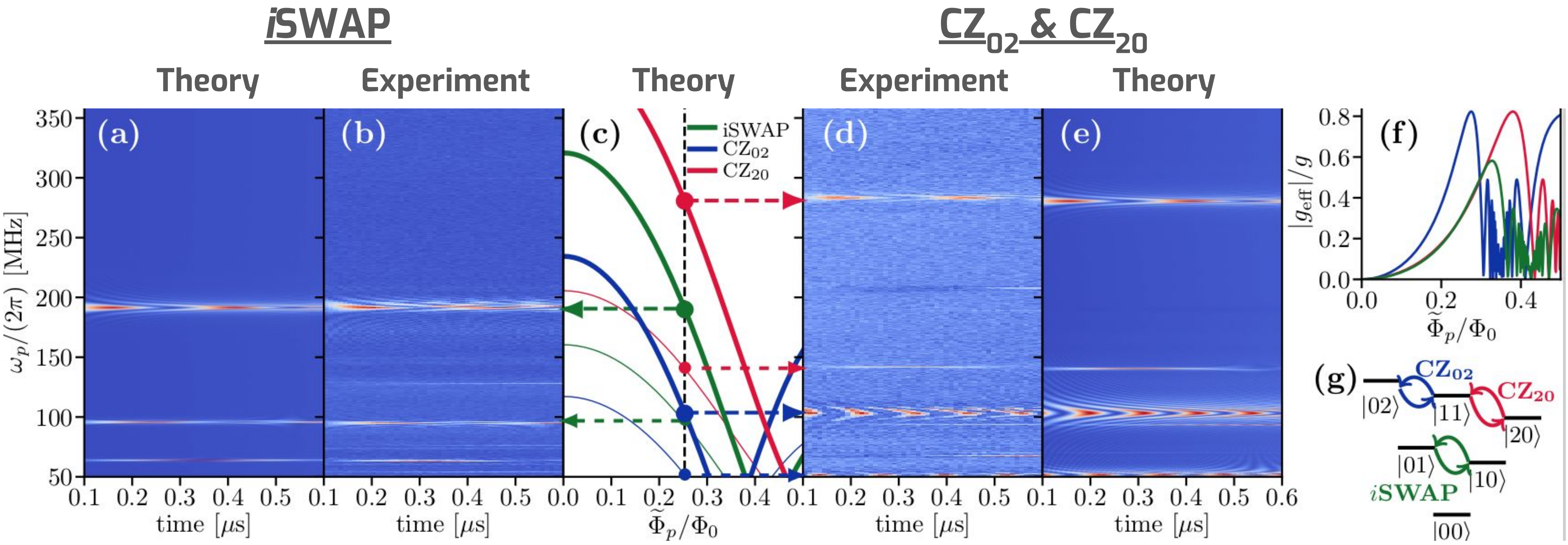
Didier & Rigetti arXiv:1706.06566

Why do we need to schedule?

- Quil has **no** notion of time or synchronization.
- But time and synchronization are very important.
- What are our options?

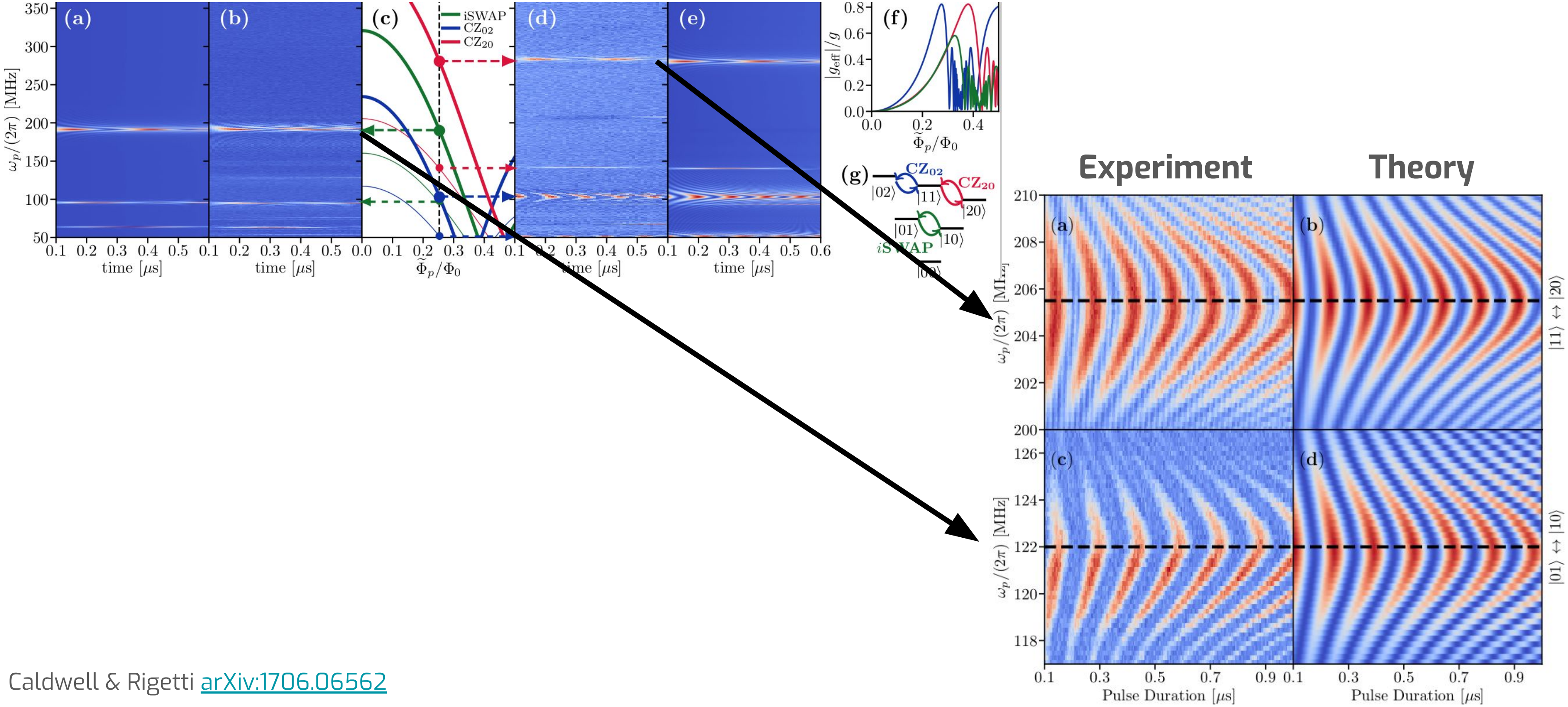
Give up; Admit the physicists are better "Program" with buttons and wires.	**Include *ad hoc* synchronization instructions** Extend Quil to "know" about time.	**Compile Quil into some temporal representation** Add machine-specific directives.
Pros: • Maximal control Cons: • Difficult to reason about • Nixes the idea of an abstraction • Difficult to automate • Have to think about hardware	Pros: • Directly addresses the issue • Still an abstract framework Cons: • Extremely complicated! • Difficult to reason about • Not easily extensible • Hard to implement • Loses the "essence"	Pros: • Remains abstract • Adds control as necessary • Extensible! • Keeps Quil "clean" Cons: • Compilation is more difficult • Performance characterization is machine-specific

Parametric entangling gates



Caldwell & Rigetti arXiv:1706.06562

Parametric entangling gates



Caldwell & Rigetti arXiv:1706.06562

Parametric entangling gates

TABLE II. Characteristics of the two-qubit CZ gates performed between neighboring qubit pairs (Q_0, Q_1), (Q_1, Q_2), and (Q_2, Q_3). g represents the qubit-qubit coupling, $\Delta_{11\leftrightarrow02}$ the detuning between $|11\rangle$ and $|02\rangle$, $\Delta_{11\leftrightarrow20}$ the detuning between $|11\rangle$ and $|20\rangle$, ω_m the modulation frequency, $\delta\omega$ the effective detuning of the tunable qubit under modulation, $T^*_{2,\mathrm{eff}}$ the effective coherence time of the tunable qubit under modulation, τ the duration of the CZ gate, and $\mathcal{F}_{QPT}$ the two-qubit gate fidelity measured by quantum process tomography. The symbol † denotes the transitions used for the gate.

Qubit pair index	$g/2\pi$ (MHz)	$\Delta_{11\leftrightarrow02}/2\pi$ (MHz)	$\Delta_{11\leftrightarrow20}/2\pi$ (MHz)	$\omega_m/2\pi$ (MHz)	$\delta\omega/2\pi$ (MHz)	$T^*_{2,\mathrm{eff}}$ (μs)	τ (ns)	$\mathcal{F}_{QPT}$ %
$Q_0 - Q_1$	3.8	69.2†	315.0	83.3	281	3.8	278	95
$Q_1 - Q_2$	4.2	187.3†	180.1	82.9	338	3.0	353	93
$Q_2 - Q_3$	4.2	855.1	1240.3†	199.9	257	5.2	395	91

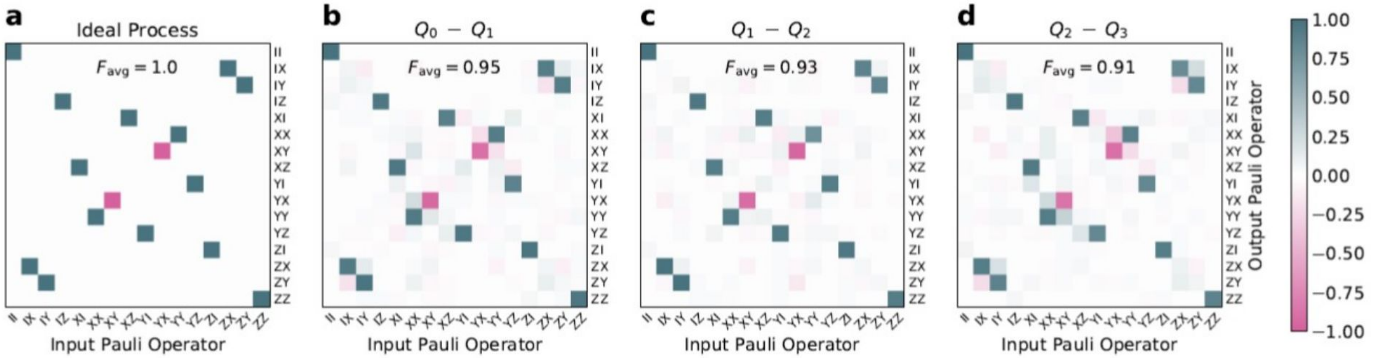
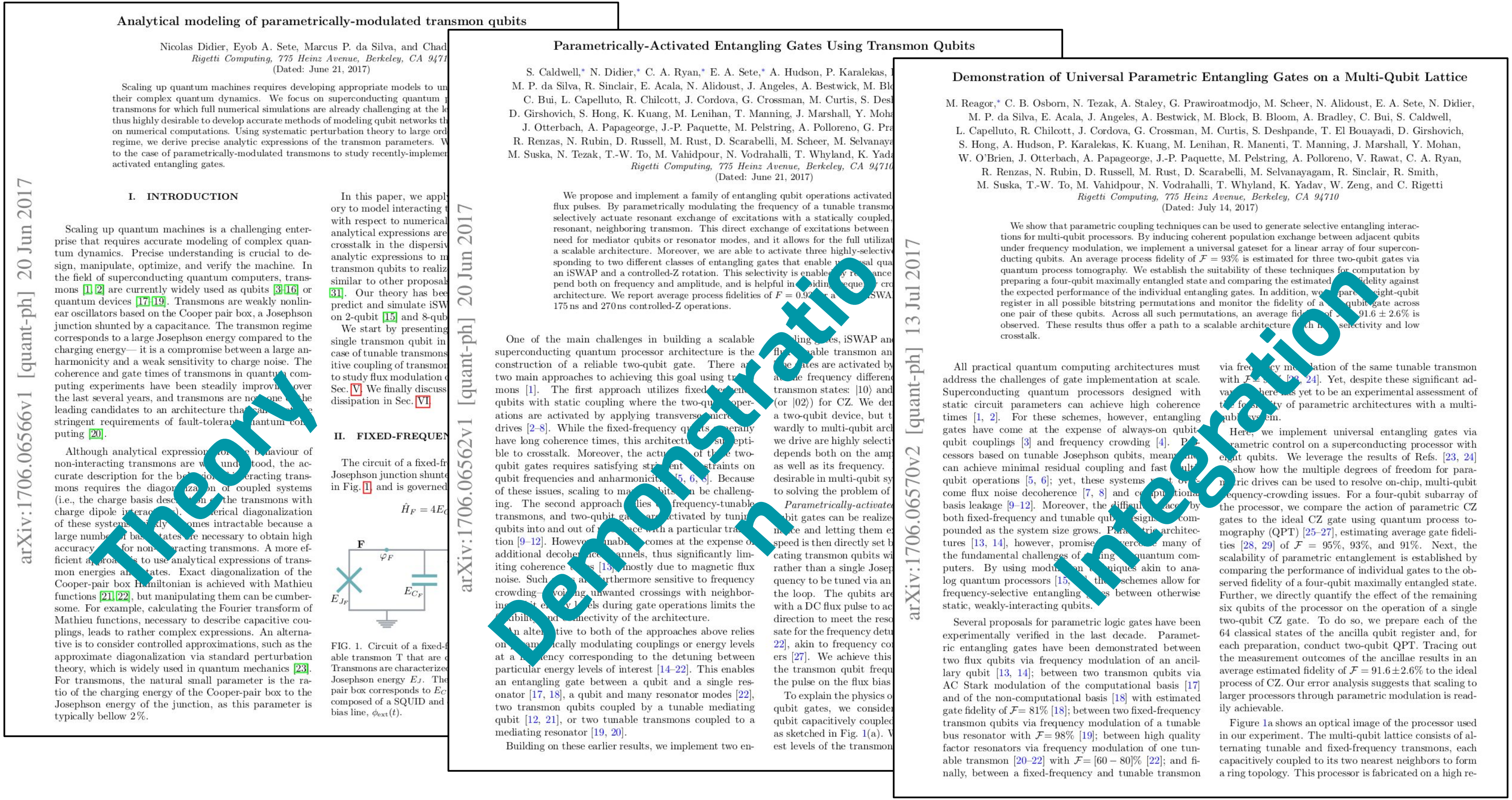


FIG. 3. **Quantum process tomography.** Process matrices of **a,** the ideal process, and CZ gates between **b,** $Q_0 - Q_1$, **c,** $Q_1 - Q_2$, and **d,** $Q_2 - Q_3$. The achieved average fidelities are measured to be 95%, 93%, and 91%, respectively.

Reagor & Rigetti arXiv:1706.06570

Forest 1.0

June 20



What is a quantum computer?

Machine that natively executes unitary operations on quantum systems

- Generalizes universal classical computer
- Benefits from inherent size of Hilbert spaces
- Better performance on notable hard problems

Classical state	Quantum state
011	$a_{000}\|000\rangle + a_{001}\|001\rangle + a_{010}\|010\rangle + a_{011}\|011\rangle + a_{100}\|100\rangle + a_{101}\|101\rangle + a_{110}\|110\rangle + a_{111}\|111\rangle$

$$|j_3 j_2 j_1\rangle \equiv |j_3\rangle \otimes |j_2\rangle \otimes |j_1\rangle$$

What is a quantum computer?

Machine that natively executes unitary operations on quantum systems

- Generalizes universal classical computer
- Benefits from inherent size of Hilbert spaces
- Better performance on notable hard problems

+1 qubit = 2x compute or memory

- Addressable problem size
- Energy efficiency

Classical state	Quantum state
011	$a_{000}\|000\rangle +$ $a_{001}\|001\rangle +$ $a_{010}\|010\rangle +$ $a_{011}\|011\rangle +$ $a_{100}\|100\rangle +$ $a_{101}\|101\rangle +$ $a_{110}\|110\rangle +$ $a_{111}\|111\rangle$

$$|j_3 j_2 j_1\rangle \equiv |j_3\rangle \otimes |j_2\rangle \otimes |j_1\rangle$$

What is a quantum computer?

Machine that natively executes unitary operations on quantum systems

- Generalizes universal classical computer
- Benefits from inherent size of Hilbert spaces
- Better performance on notable hard problems

+1 qubit = 2x compute or memory

- Addressable problem size
- Energy efficiency

Interesting properties

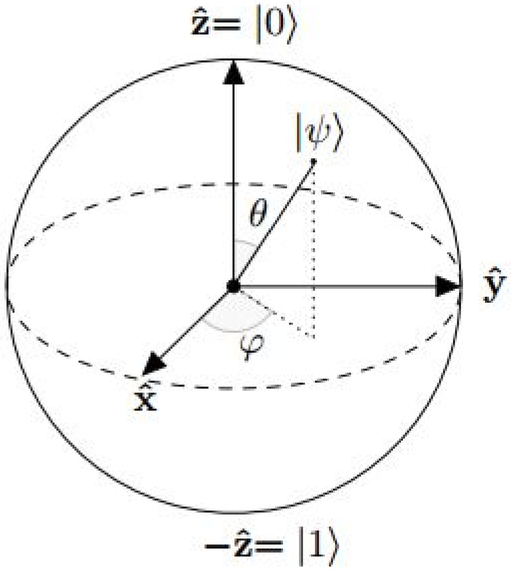
- Fully reversible
- No copying an arbitrary state
- Non-deterministic state readout

Classical state	Quantum state
011	$a_{000}\|000\rangle + a_{001}\|001\rangle + a_{010}\|010\rangle + a_{011}\|011\rangle + a_{100}\|100\rangle + a_{101}\|101\rangle + a_{110}\|110\rangle + a_{111}\|111\rangle$

$$|j_3 j_2 j_1\rangle \equiv |j_3\rangle \otimes |j_2\rangle \otimes |j_1\rangle$$

One-qubit quantum state

Lives on the surface of the Bloch sphere

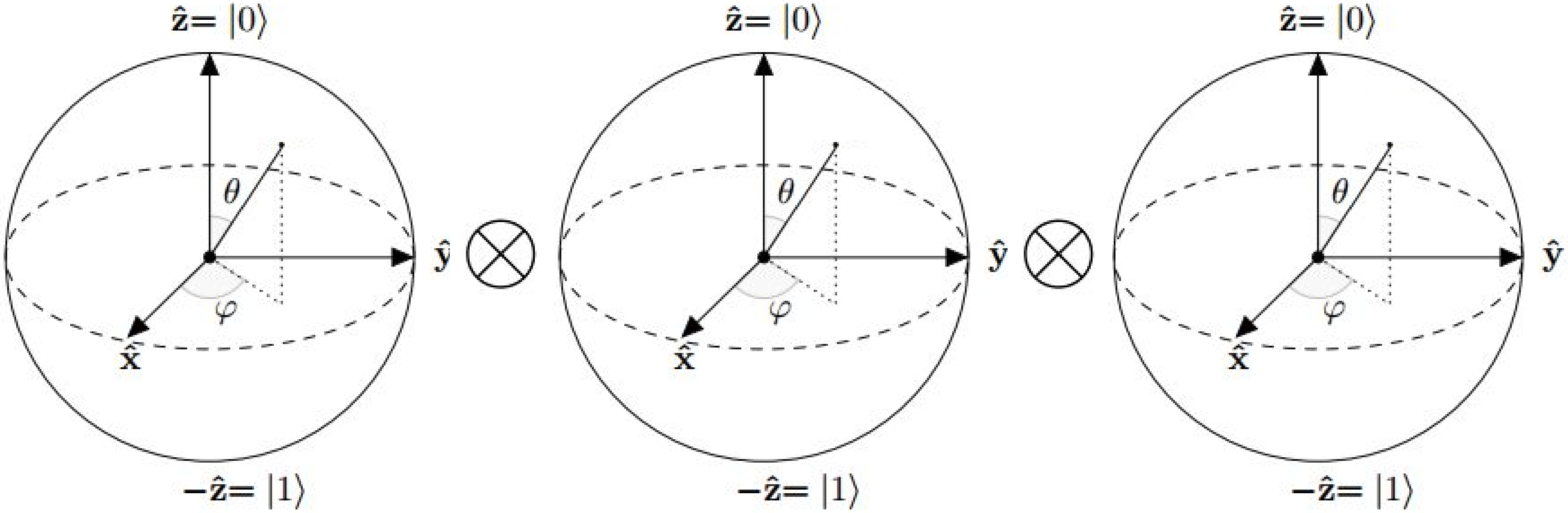


$$|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\varphi}\sin(\theta/2)|1\rangle$$

Multi-qubit state

$$|\psi\rangle = \begin{array}{l} a_{000}|000\rangle + \\ a_{001}|001\rangle + \\ a_{010}|010\rangle + \\ a_{011}|011\rangle + \\ a_{100}|100\rangle + \\ a_{101}|101\rangle + \\ a_{110}|110\rangle + \\ a_{111}|111\rangle \end{array}$$

Arbitrary state $|\psi\rangle$

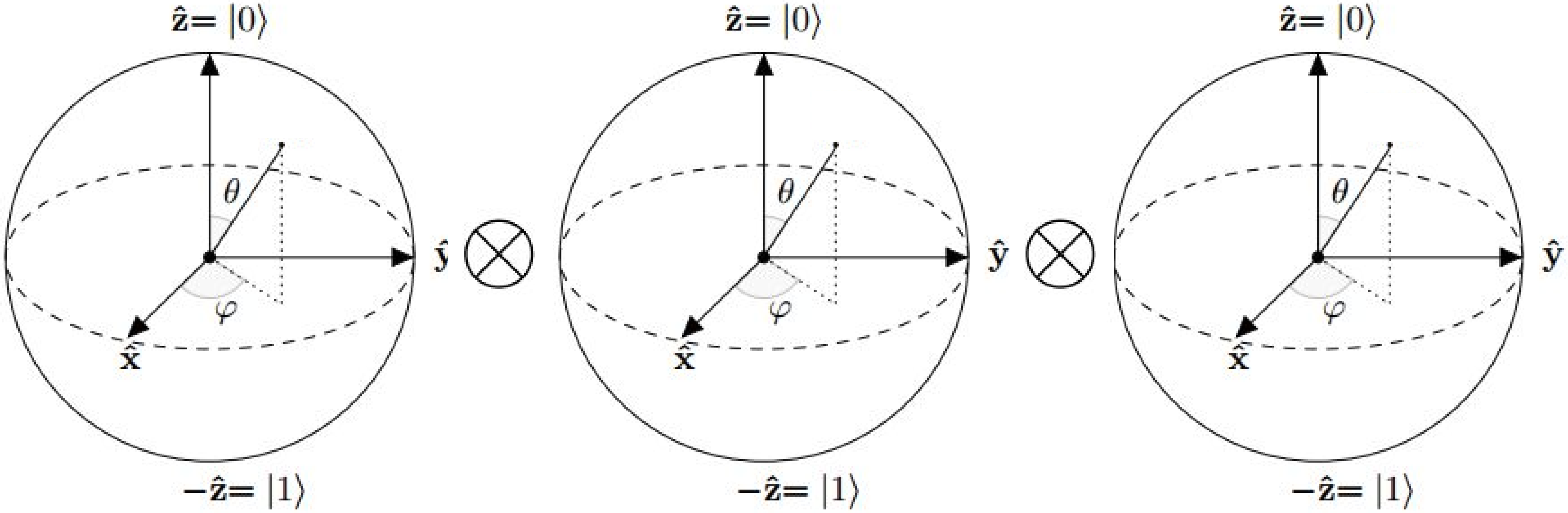


Larger Hilbert space is tensor product of smaller ones

Multi-qubit state

Arbitrary state $|\psi\rangle$

$$|\psi\rangle = \begin{array}{l} a_{000}|000\rangle + \\ a_{001}|001\rangle + \\ a_{010}|010\rangle + \\ a_{011}|011\rangle + \\ a_{100}|100\rangle + \\ a_{101}|101\rangle + \\ a_{110}|110\rangle + \\ a_{111}|111\rangle \end{array}$$

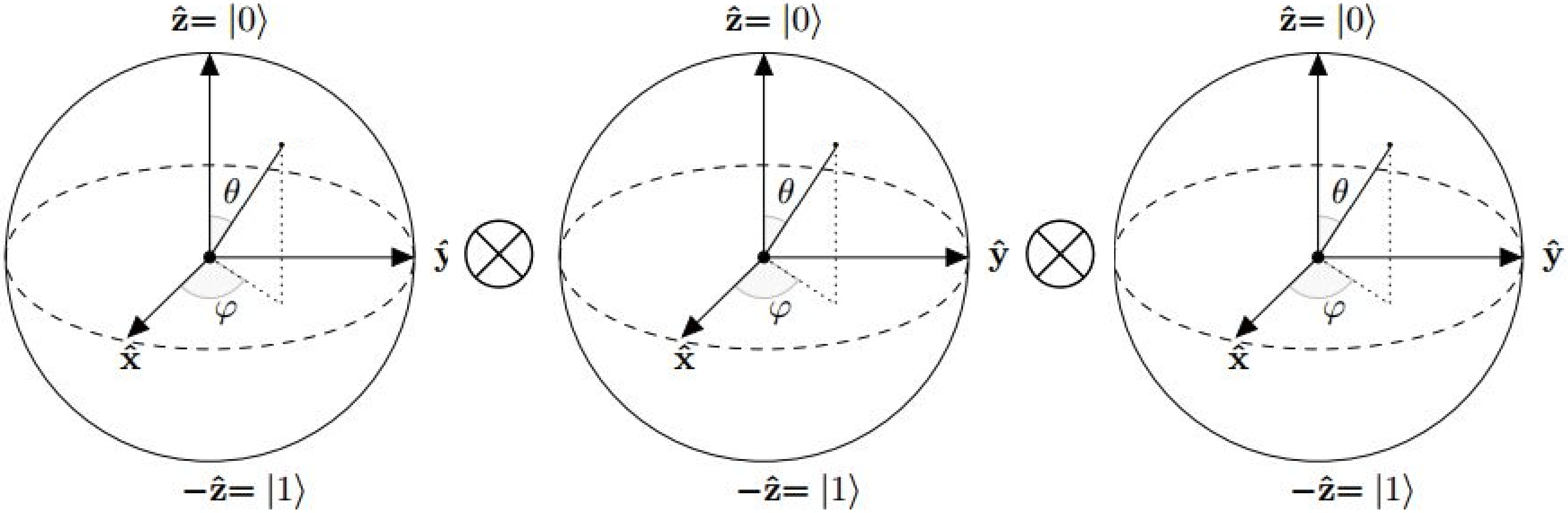


Basis states: $|j_3 j_2 j_1\rangle \equiv |j_3\rangle \otimes |j_2\rangle \otimes |j_1\rangle$

Multi-qubit state

$$|\psi\rangle = \begin{matrix} a_{000}|000\rangle + \\ a_{001}|001\rangle + \\ a_{010}|010\rangle + \\ a_{011}|011\rangle + \\ a_{100}|100\rangle + \\ a_{101}|101\rangle + \\ a_{110}|110\rangle + \\ a_{111}|111\rangle \end{matrix}$$

Arbitrary state $|\psi\rangle$



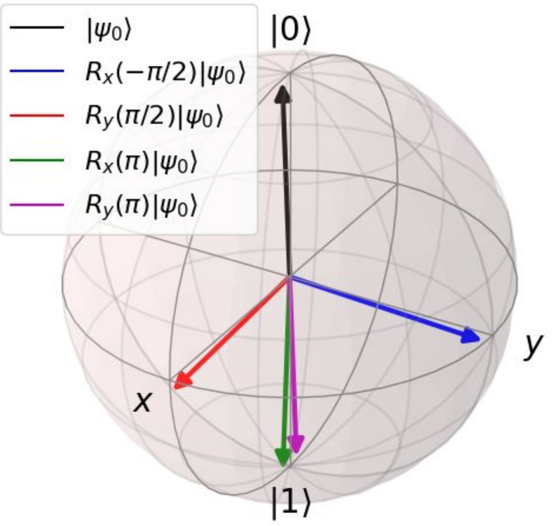
Basis states: $|j_3 j_2 j_1\rangle \equiv |j_3\rangle \otimes |j_2\rangle \otimes |j_1\rangle$

General states: $|\psi\rangle \not\equiv |\psi_3\rangle \otimes |\psi_2\rangle \otimes |\psi_1\rangle$ (entanglement)

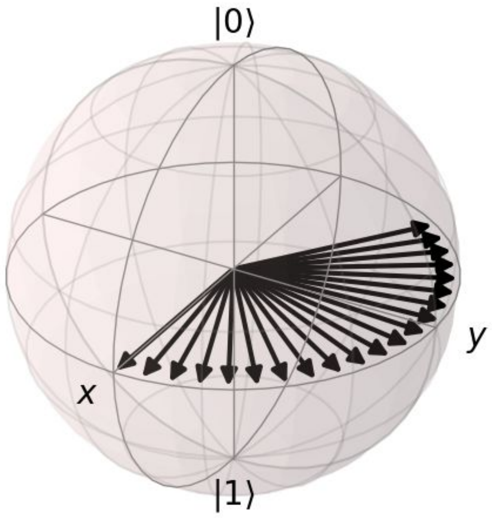
Controlling a quantum state

"Machine that natively executes unitary operations on quantum systems"

Unitaries are rotations



X and Y rotations by driving



Z rotations by waiting

- Qubit frequency f_q
- Qubit precesses in xy (complex) plane at f_q
- X and Y rotations driven with external fields