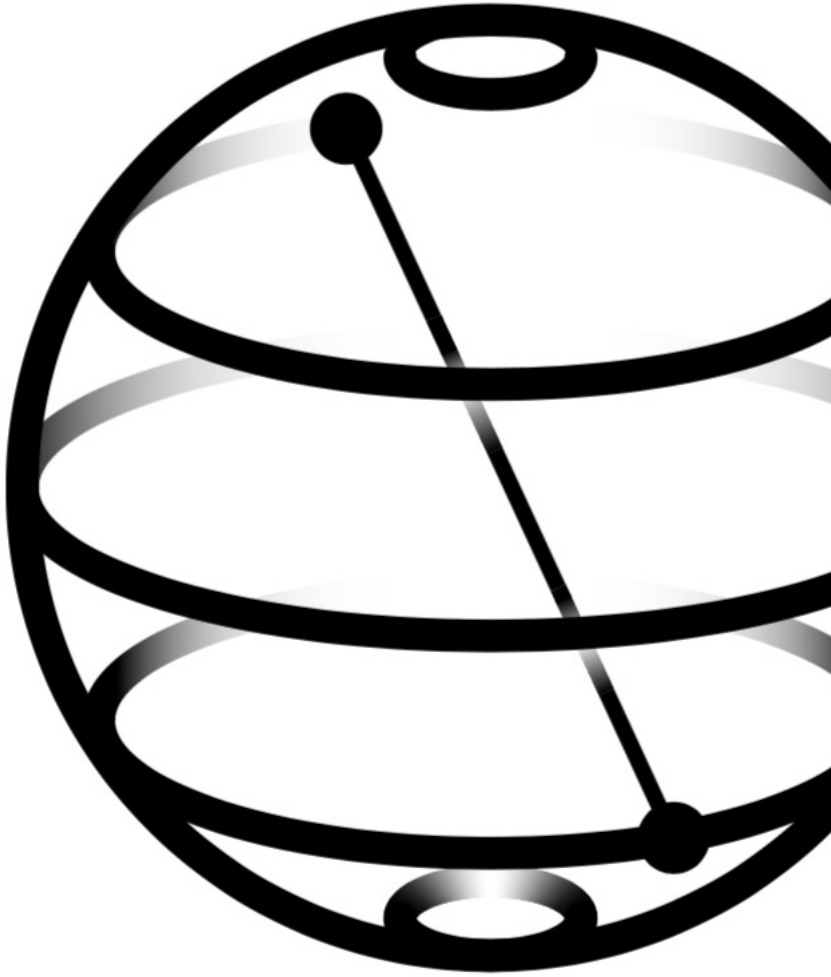
Qiskit Pulse

quantum computing with microwave pulses

Lauren Capelluto

IBM Quantum

19 November 2021



Overview

- Advantages of real-time control
- How to manipulate qubits using pulses
- The Pulse programming model
- Demo: calibrate an X180 gate

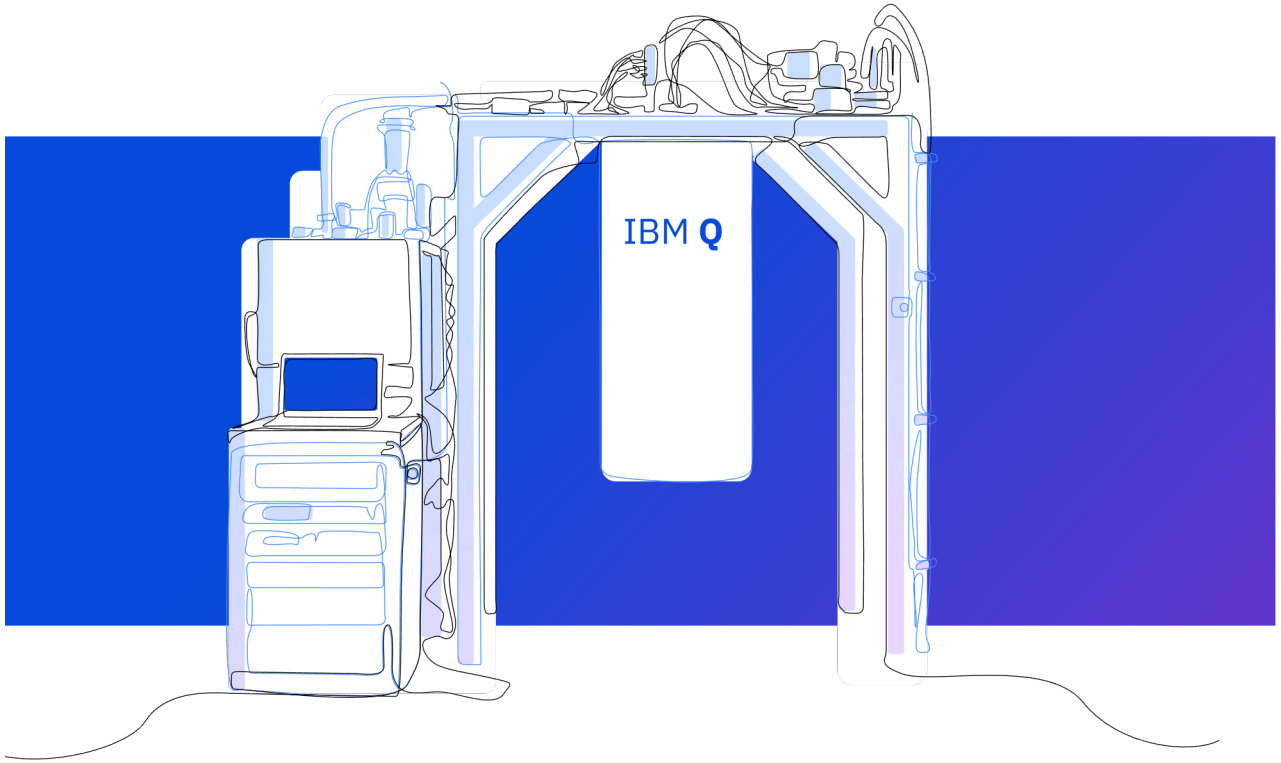
Near-term devices and error mitigation

Hardware

- system and device design
- device fabrication
- control hardware

Software

- sophisticated compilation techniques
  - front-end (gate based)
  - **back-end (pulse based)**



lauren@ibm.com

Quantum Circuits

Higher level programming

- traditionally timing agnostic
- fixed performance of individual operations
- transpiler (circuit optimizer)
  - can reduce the size of the input circuit with rewriting rules
  - optimize qubit mapping to utilize performant qubits and layouts

Pulse Schedules

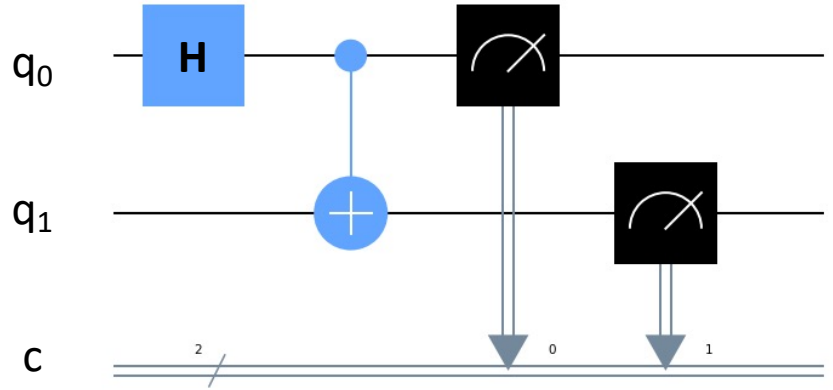
Low level hardware control

- real-time, hard deadline waveform scheduling
  - control over gate definitions
  - control over readout pipeline
- greater complexity
- requires more domain knowledge

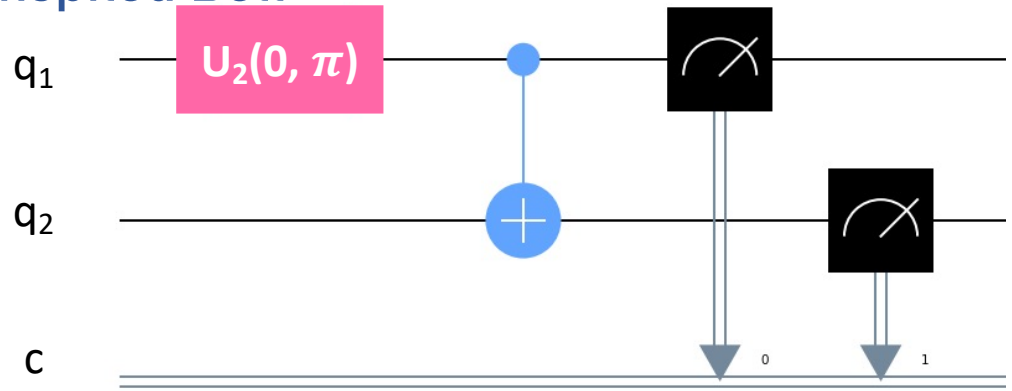
lauren@ibm.com

Quantum compilation: front-end

Bell state preparation and measurement



transpiled Bell



topology

performance

OPERATION	ERROR RATE
$CNOT_{01}$	0.05
$CNOT_{12}$	0.01

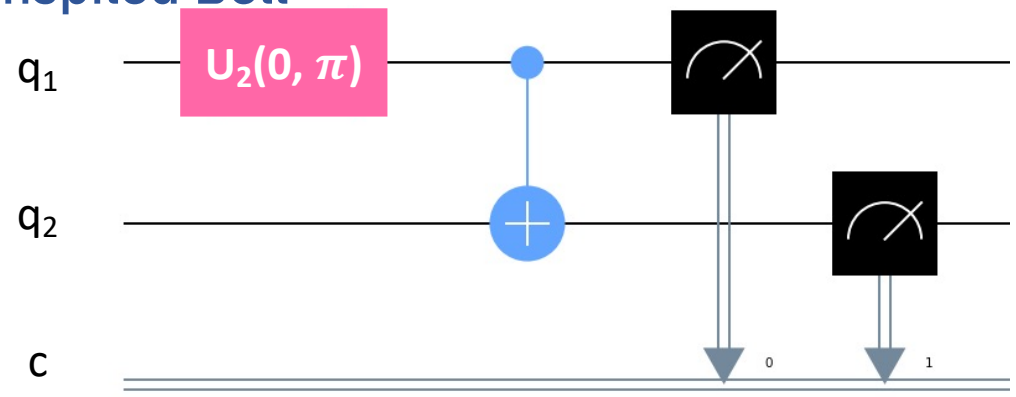
native gates

CNOT	$U_1(\lambda)$	$U_2(\phi, \lambda)$

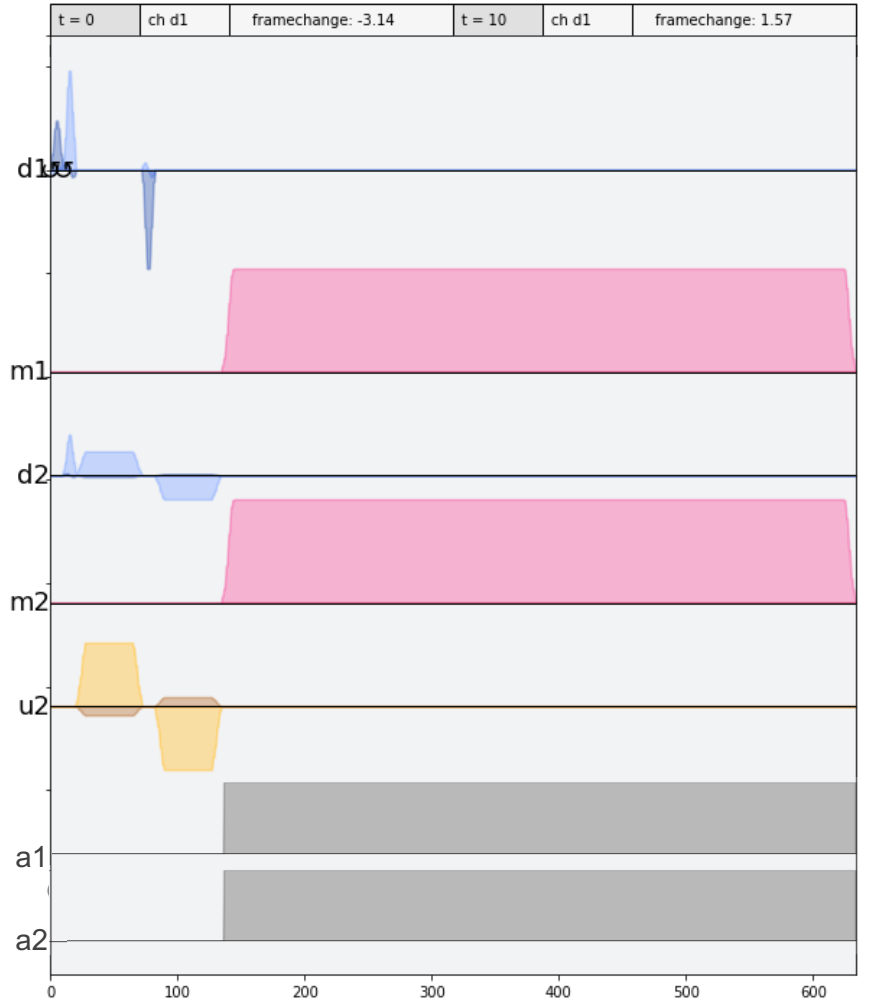
Quantum compilation: back-end

- back-end compilation is called scheduling
- schedules operate on channels (physical signal lines) rather than logical qubits

transpiled Bell



scheduled Bell



Quantum Circuits

Higher level programming

- traditionally timing agnostic
- fixed performance of individual operations

Pulse Schedules

Low level hardware control

- real-time, hard deadline waveform scheduling
- greater complexity

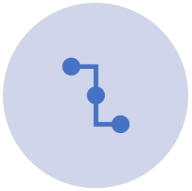
lauren@ibm.com

GATE SCHEDULING

MEASUREMENT DATA

ADVANCED DEVICE CHARACTERIZATION

GATE DISCOVERY

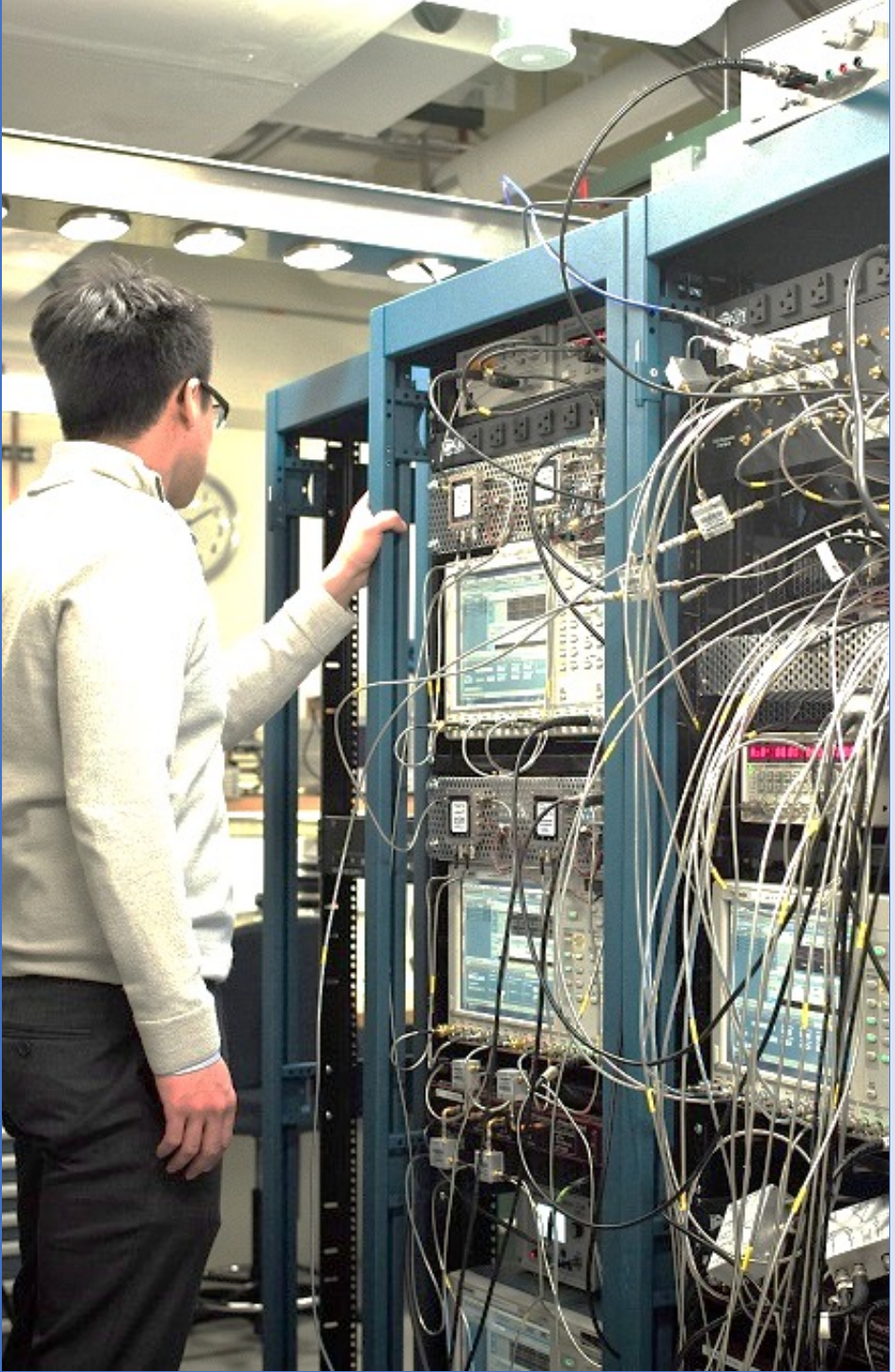


DYNAMICAL DECOUPLING

OPTIMAL CONTROL THEORY

Real-time programming

- Given hardware constraints
  - increase fidelity
  - increase robustness
  - decrease duration



Advantages of real-time control

How to manipulate qubits using pulses

The Pulse programming model

Demo: calibrate an X180 gate

Manipulating qubits

- universal gate set: a set of gate operations which can be used to generate any other gate
- native gate set: gates which are defined on a system

Cross resonant (CR) $R_Z(\lambda)$ $R_X(\pi/2)$

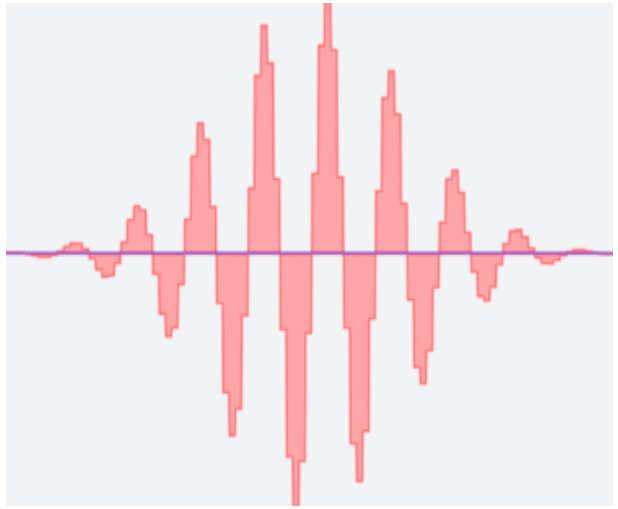
Manipulating qubits

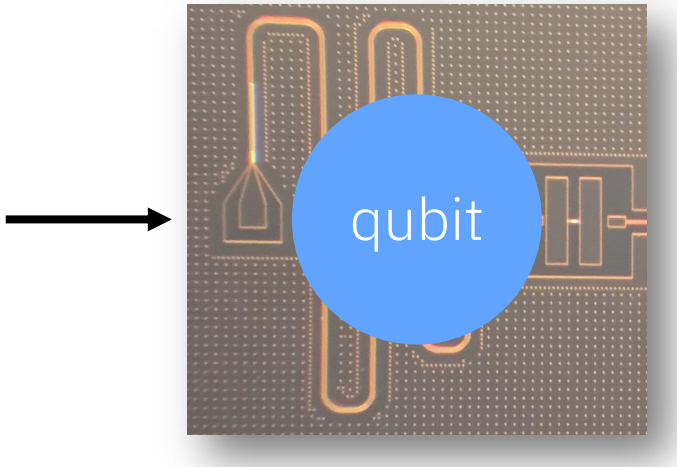
- universal gate set: a set of gate operations which can be used to generate any other gate
- native gate set: gates which are defined on a system

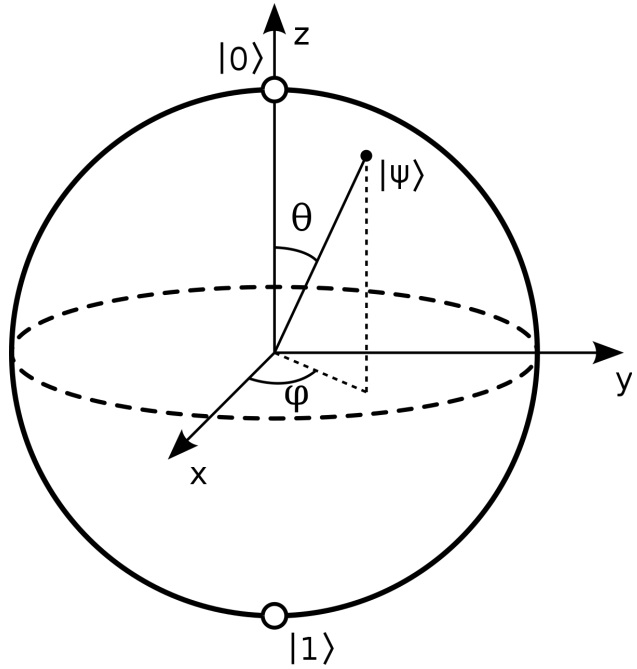
Cross resonant (CR) $R_Z(\lambda)$ $R_X(\theta)$

Single qubit control

- The qubit state naturally oscillates around the z-axis
  - The phase, φ, can be set arbitrarily by shifting the reference frame
- Microwave pulses driving the qubit at the frequency of its $|0\rangle \rightarrow |1\rangle$ transition will rotate the qubit state
  - Visualized as a rotation by the polar angle θ

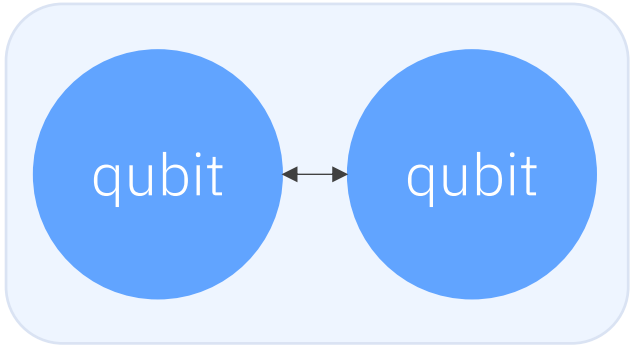






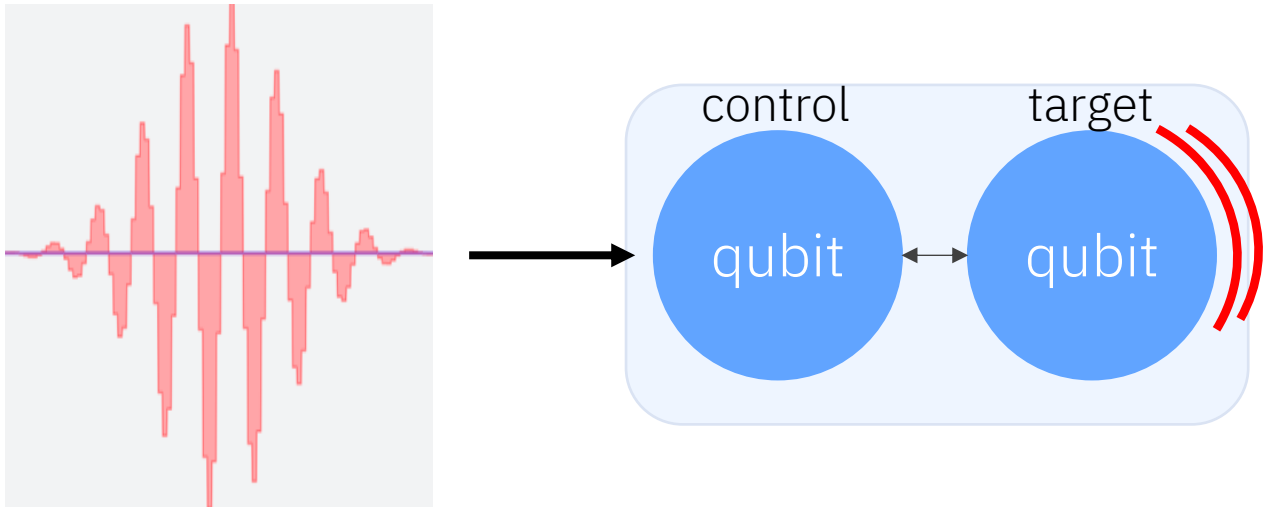
Two qubit control

- 2 qubits, coupled to one another, with detuned resonant frequencies

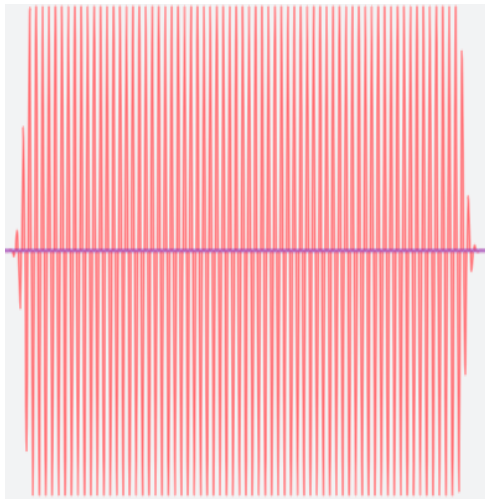


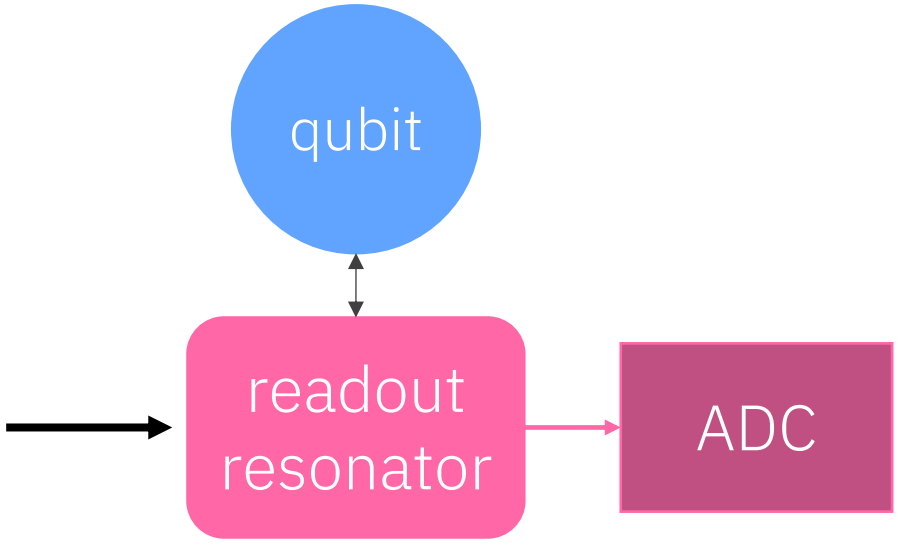
Two qubit control

- Cross-Resonance: Apply pulse to control qubit at frequency of target qubit

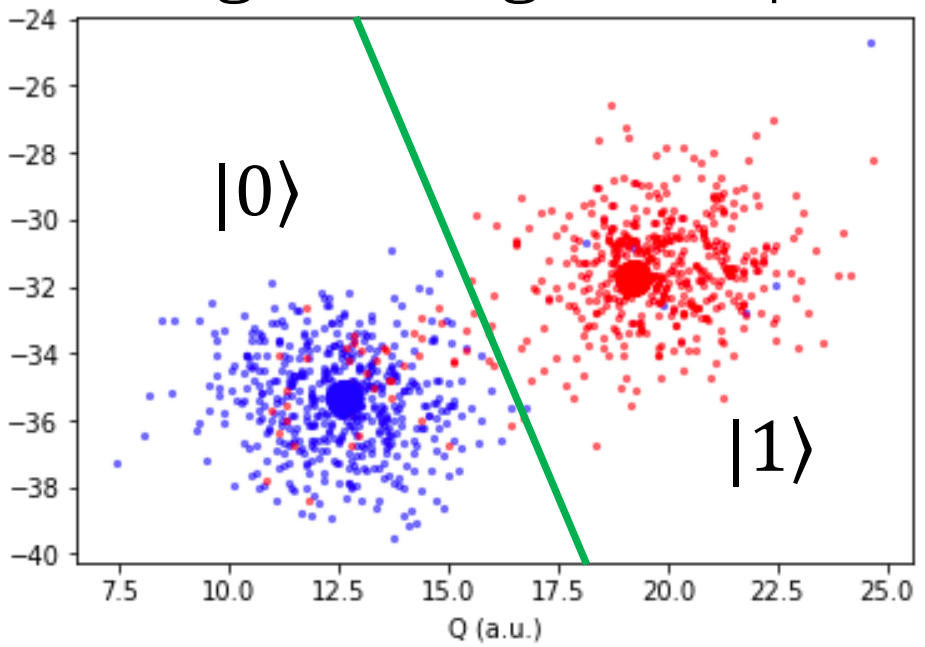


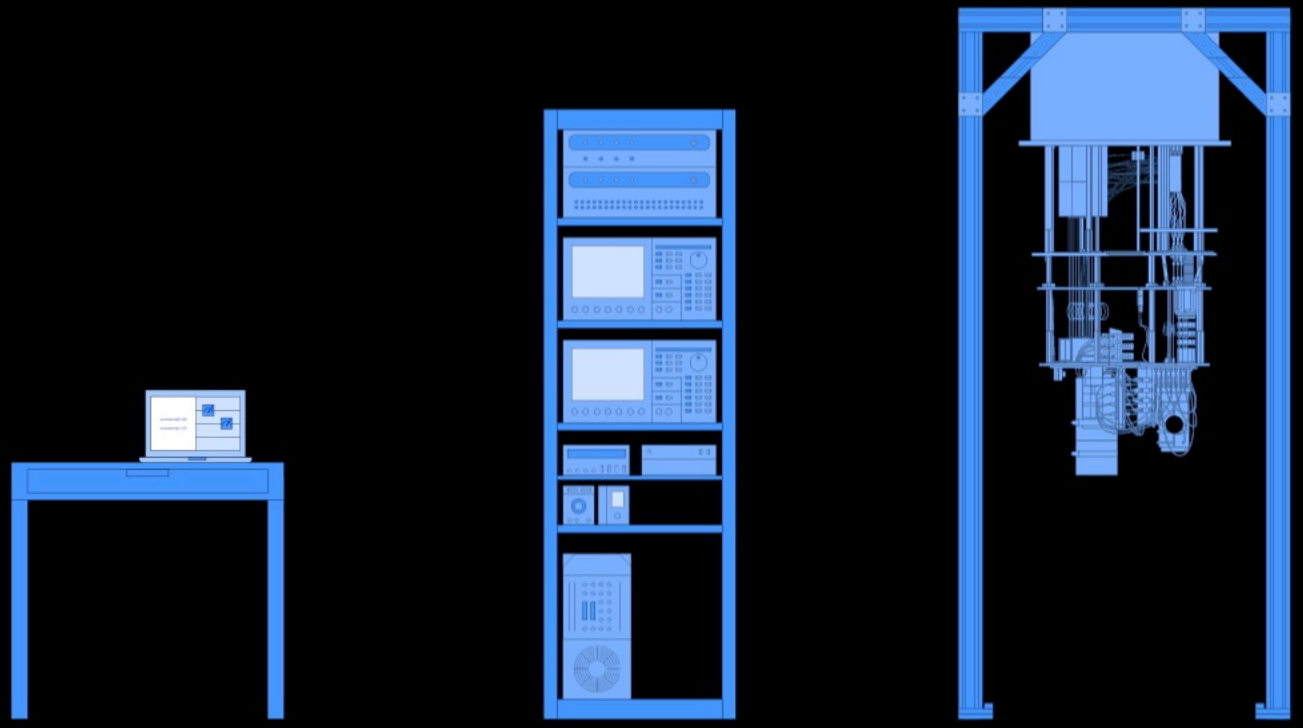
Measurement readout



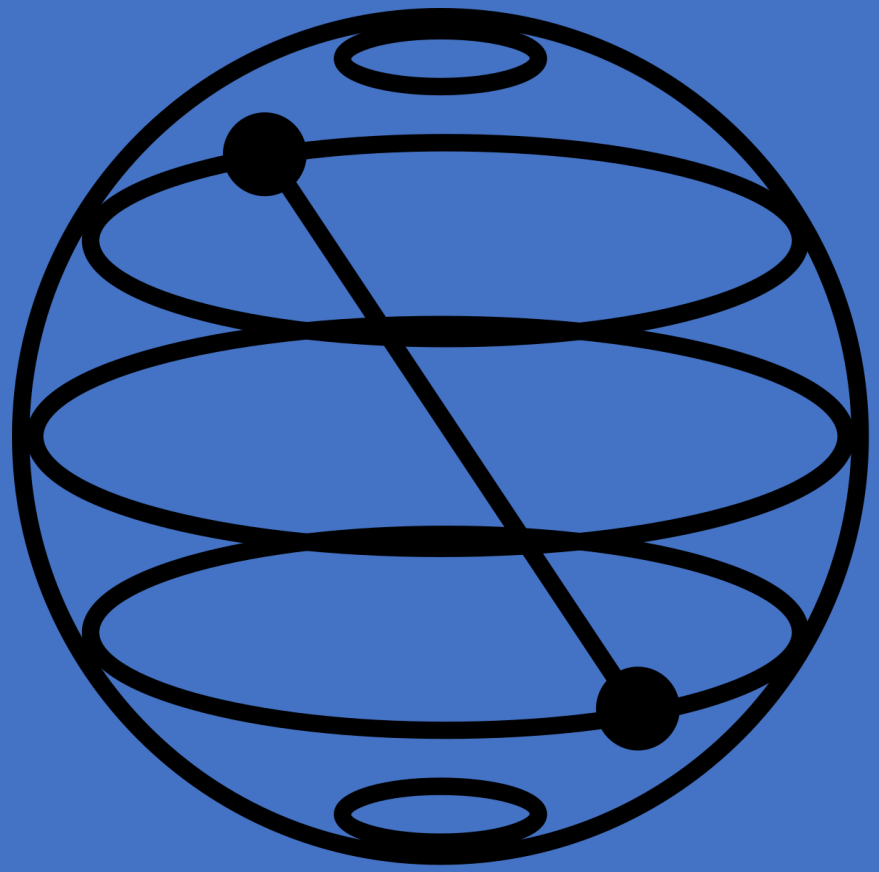


Integrated signal output





IBM Q



Advantages of real-time control

How to manipulate qubits using pulses

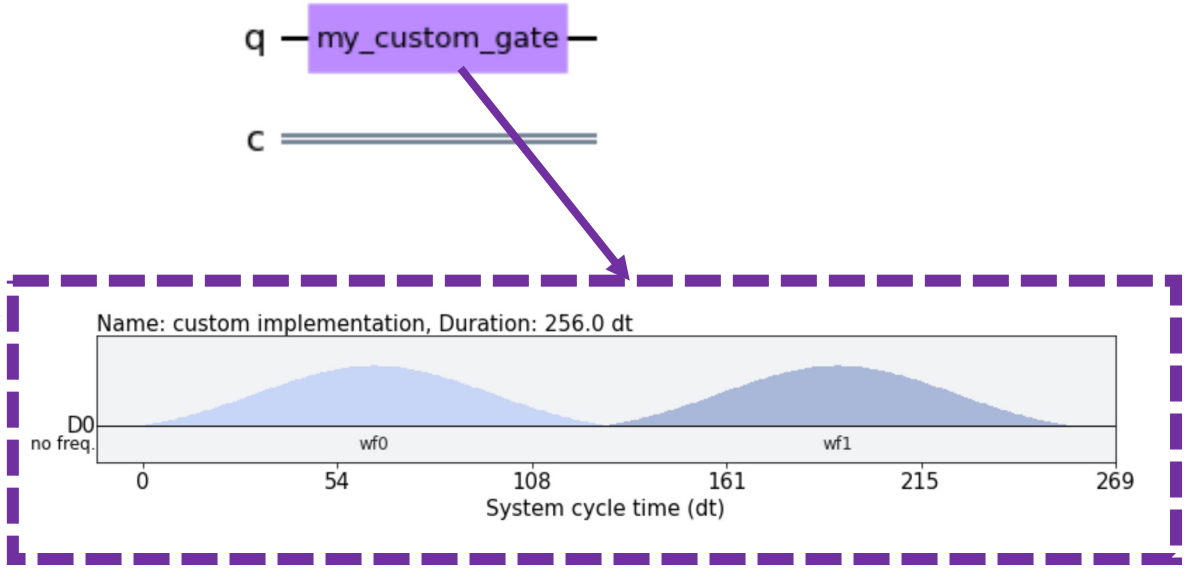
The Pulse programming model

Demo: calibrate an X180 gate

Embedding pulse programming

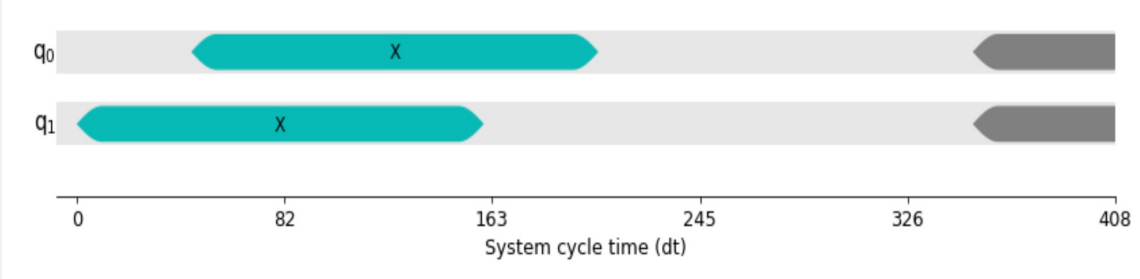
Pulse gates

- pulse programs embedded into a circuit

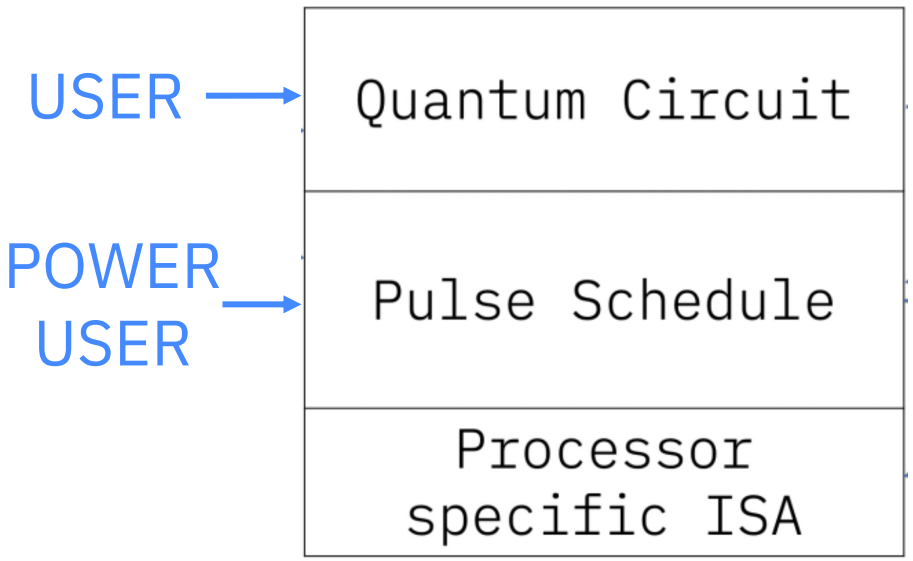


Scheduled circuits

- timing-aware circuits
- generally multiple scheduling methods
  - greedy “as-soon-as-possible”
  - lazy “as-late-as-possible”



Programming model



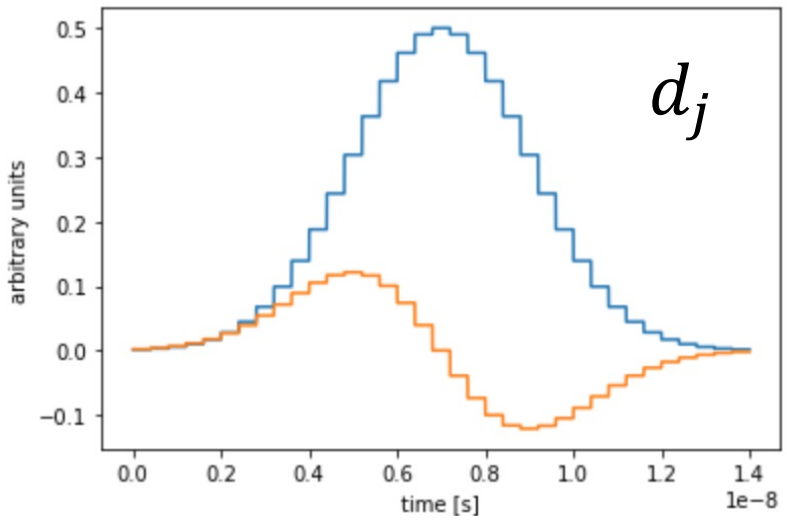
- Qiskit `pulse`: open source framework for low-level pulse control
- Exposes an abstract interface between circuit programs and the control electronics
  - agnostic of the backend implementation
- OpenPulse enabled backends contain extra information such as default frequency settings

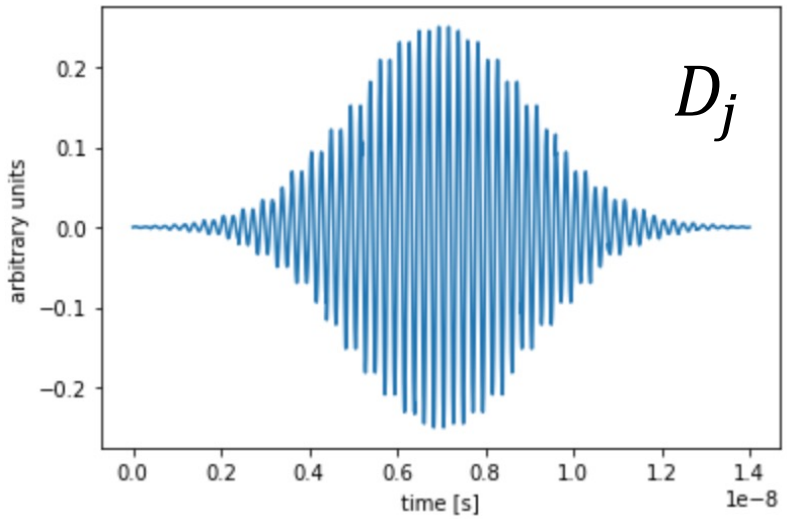
lauren@ibm.com

Programming model pulses

- pulses: complex-valued amplitudes $[d_0, \ldots, d_{n-1}]$.
- phase ϕ and frequency f are controlled by dedicated instructions

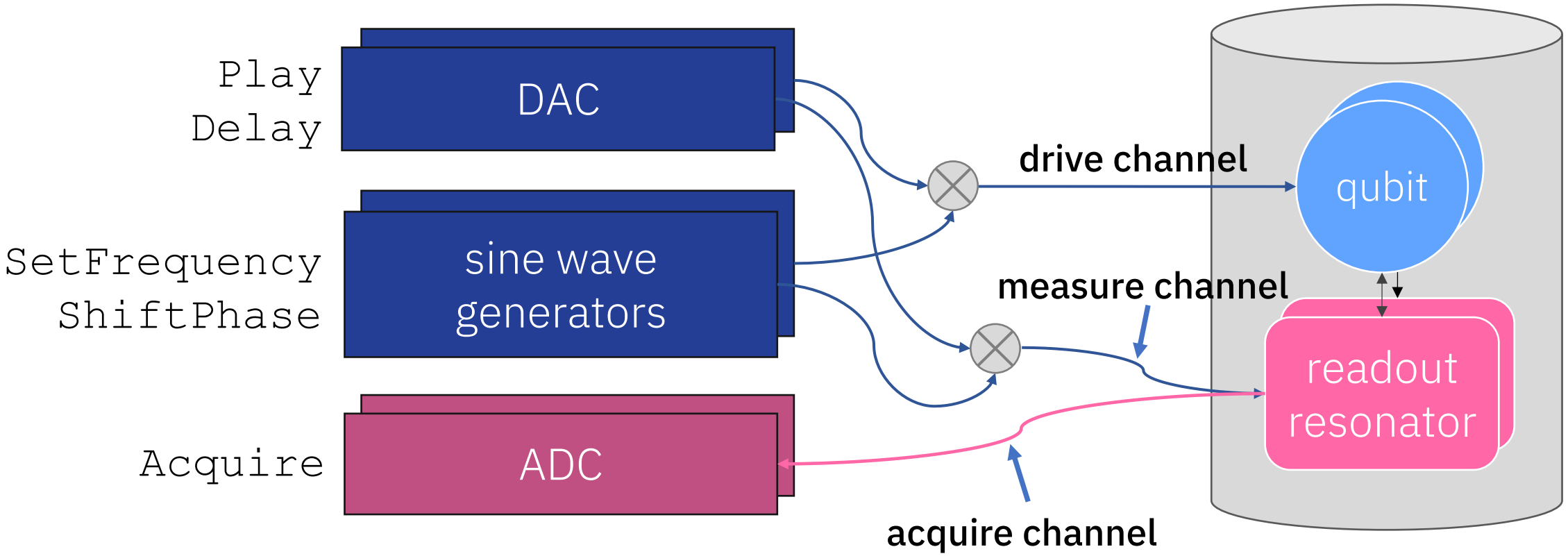
$$D_j = \mathrm{Re}\left[e^{i2\pi f j \mathbf{dt} + \phi} d_j\right]$$





lauren@ibm.com

Programming model
instructions and channels



lauren@ibm.com

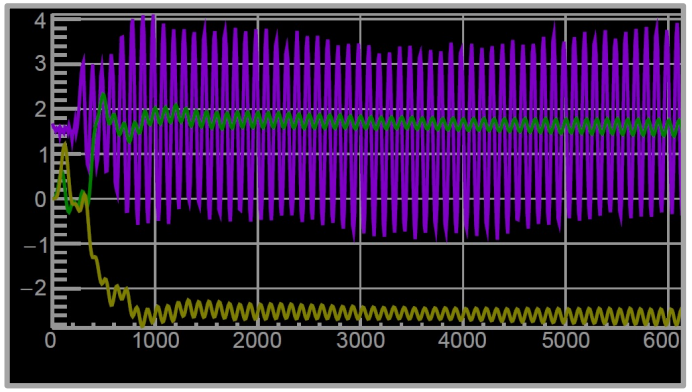
Programming model measurement levels

RAW: Level 0

Output from digitizer

Output: Time dependent signal

User input: none

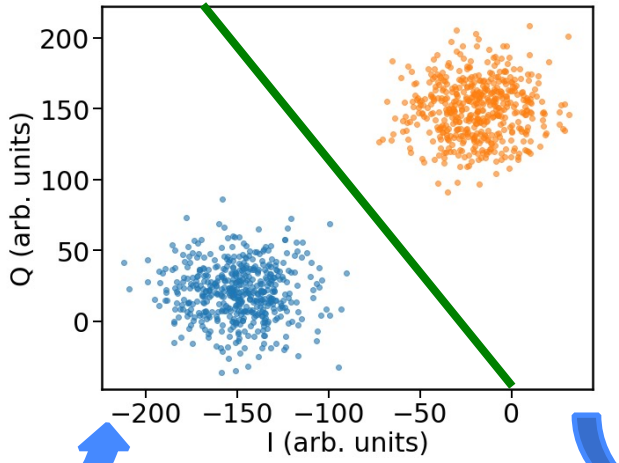


KERNELED: Level 1

Converts level 0 to IQ point

Output: IQ point

User input: Kernel



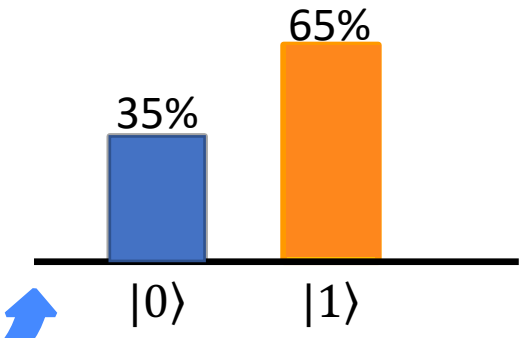
Kernel

DISCRIMINATED: Level 2

Converts level 1 to qubit 1/0 state

Output: Qubit state

User input: Kernel & discriminator



Discriminator

lauren@ibm.com

gate discovery example

Advantages of real-time control

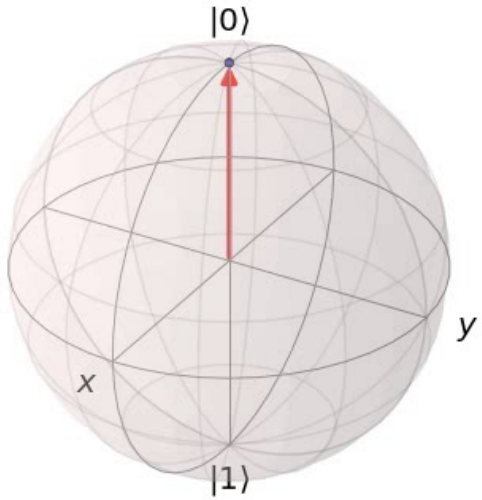
How to manipulate qubits using pulses

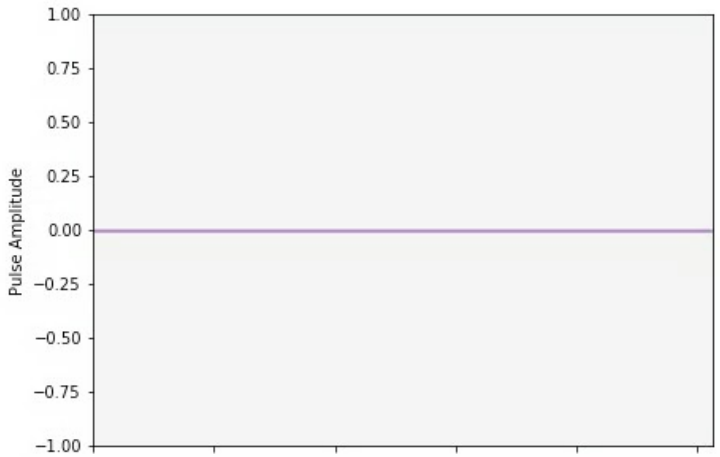
The Pulse programming model

Demo: calibrate an X180 gate

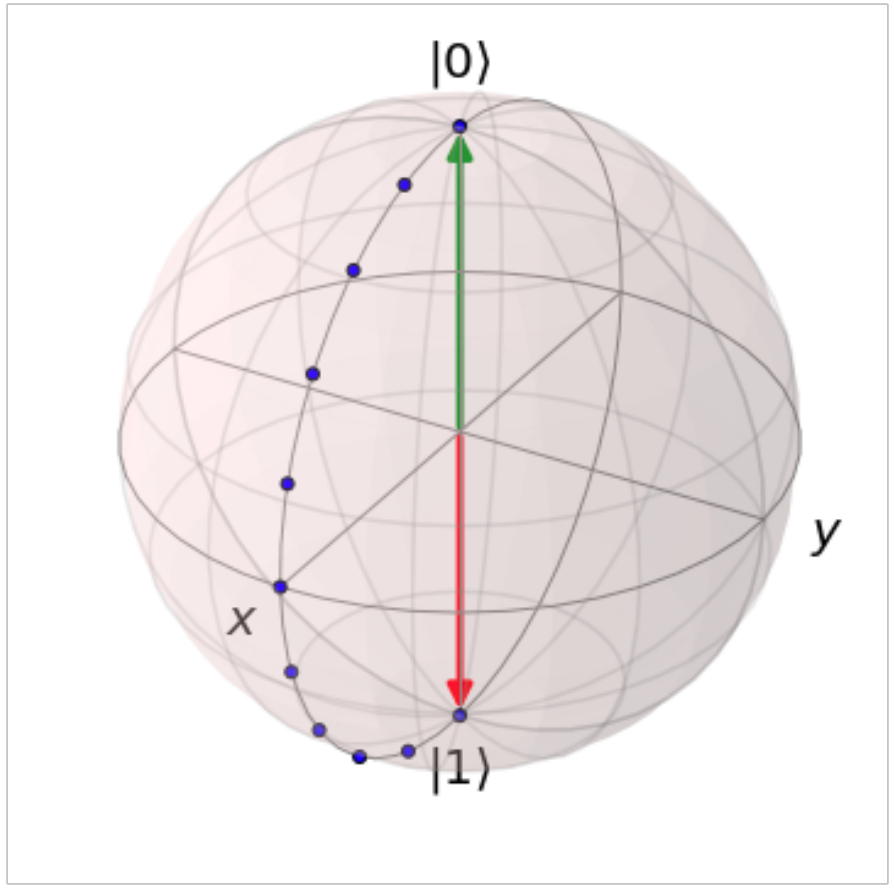
X pulse calibration via power Rabi Experiment

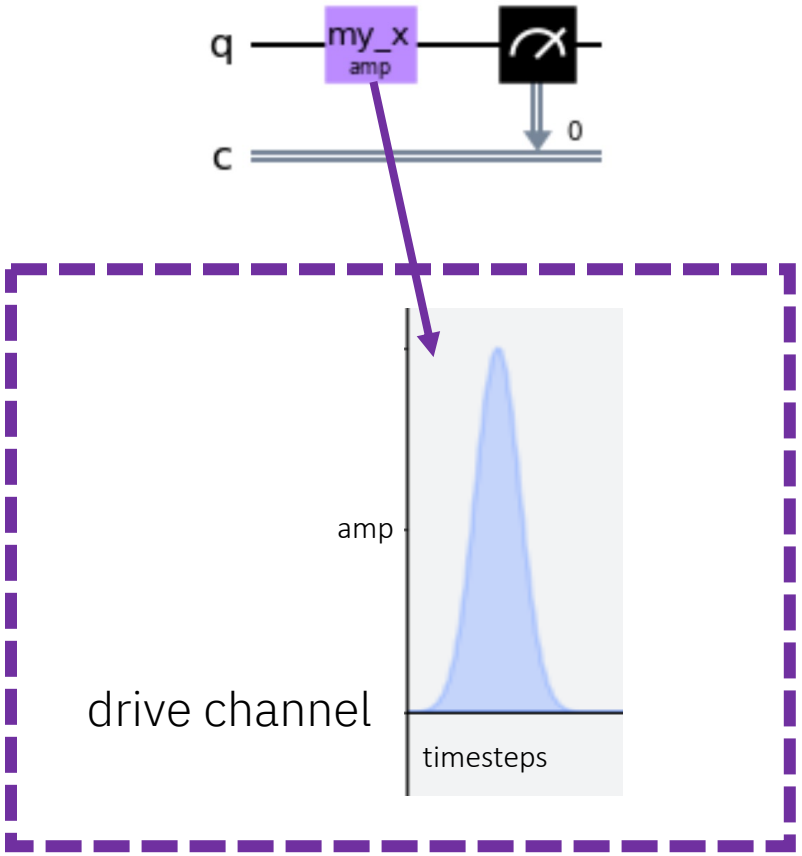
- given:
  - the qubit frequency
  - a gaussian drive pulse envelope
- determine:
  - the *X* pulse amplitude (X180, or π-pulse)
- this amplitude can be used to drive $RX(\theta)$ by linearly varying the amplitude of the pulse





X pulse calibration via power Rabi Experiment

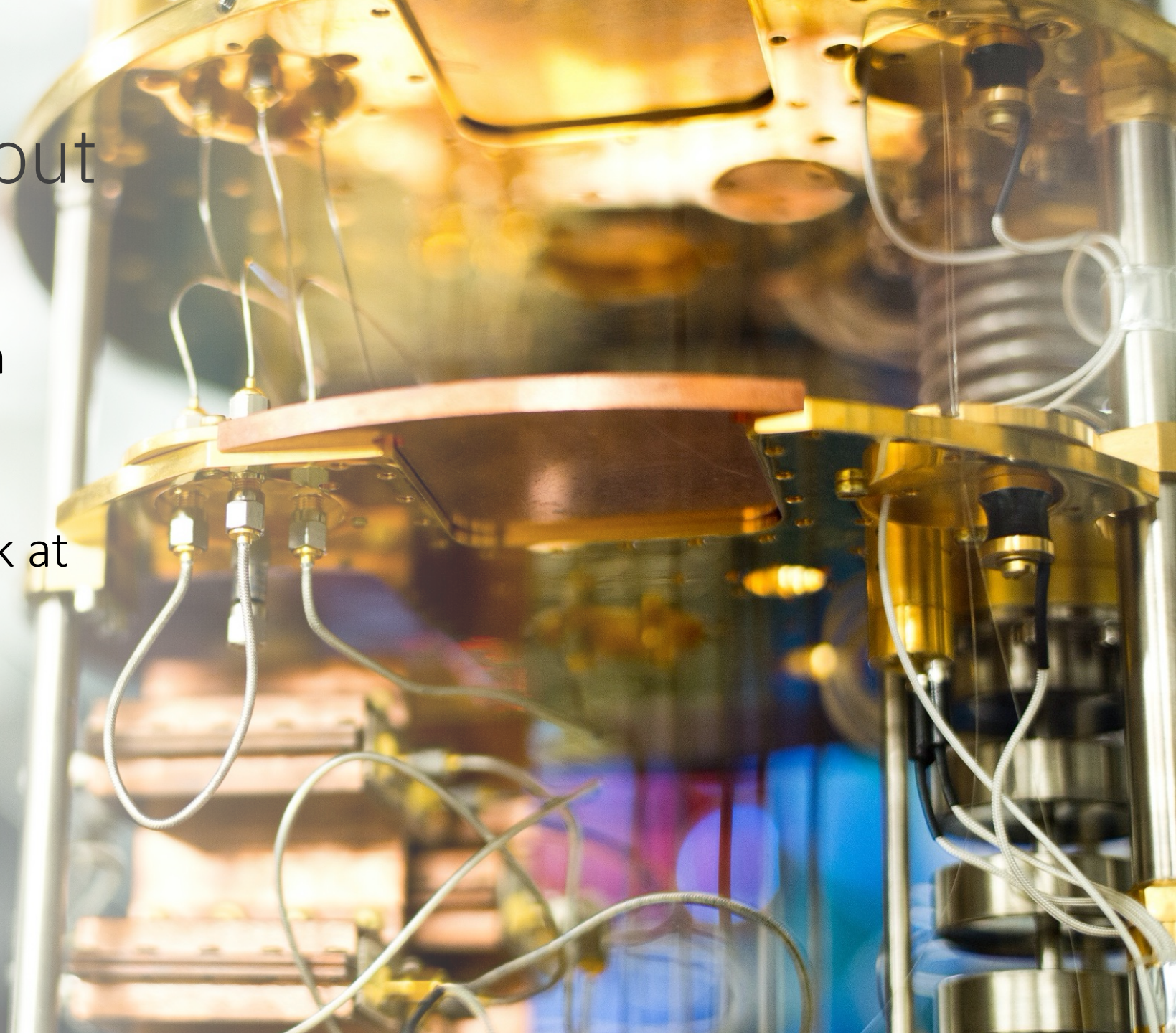





demo

summary: all about pulses

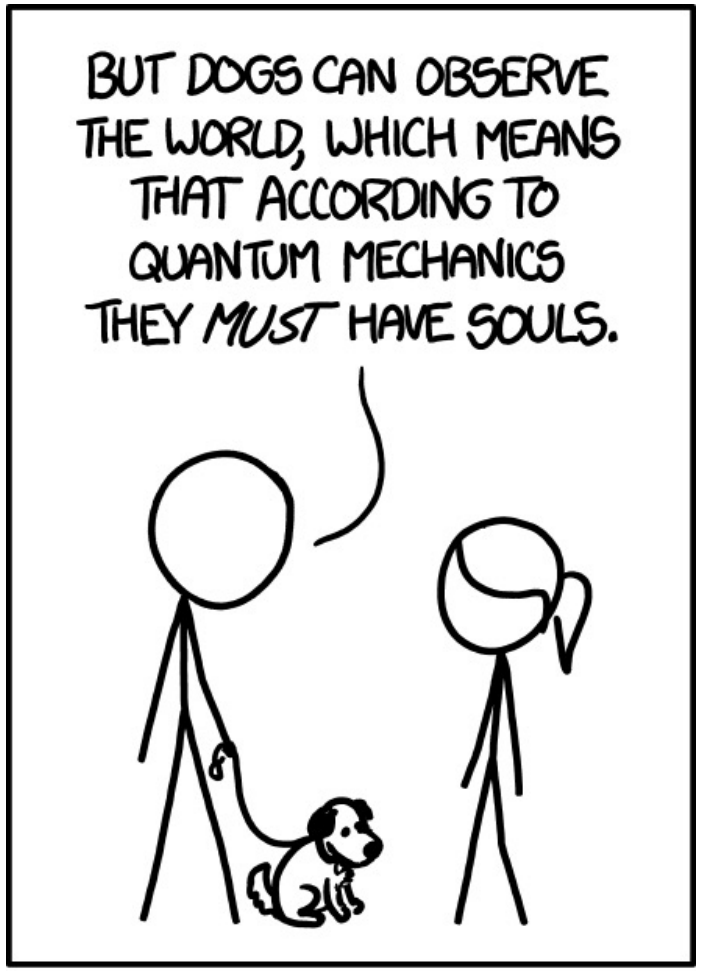
- Pulse programming is an avenue for driving down error rates
- learned how qubits work at the hardware level
- presented the Qiskit `pulse` programming model, pulse gates and scheduled circuits
- calibrated an *X* gate




Thank you!

questions?

extra slides



PROTIP: YOU CAN SAFELY IGNORE ANY SENTENCE THAT INCLUDES THE PHRASE "ACCORDING TO QUANTUM MECHANICS"

more features of `pulse`

- scheduling
- backend `configuration`, `properties`, and `defaults`

Scheduling `QuantumCircuits`

Build a circuit

```
from qiskit import QuantumCircuit

circ = QuantumCircuit(5, 5)
circ.h(0)
circ.cx(0, 1)
circ.measure([0, 1], [0, 1])
```

import scheduler

```
from qiskit.pulse.scheduler import schedule as build_schedule
```

get backend

```
IBMQ.load_account()
provider = IBMQ.get_provider(<hub, group, project>)
backend = provider.get_backend(<backend_name>)
```

schedule!

```
schedule = build_schedule(transpile(circ, backend), backend)
```

Custom scheduling

add a schedule to the inst_map

```
inst_map = backend.defaults().instruction_schedule_map
inst_map.add('h', 0, custom_q0_h_schedule)
```

schedule with options

```
schedule = build_schedule(circ,
                          backend,
                          inst_map=inst_map)
```

backend.configuration()

- static system setup

```
config = backend.configuration()

# Basic backend features    # Example values
#-------------------------------------------------------------#
config.backend_name         # 'Fake20QSystem'
config.backend_version      # '0.0.0'
config.n_qubits             # 20
config.meas_levels          # [0, 1, 2]
                            # or RAW, KERNELED and DISCRIMINATED respectively
config.basis_gates          # ['u1', 'u2', 'u3', 'cx', 'id']
config.open_pulse           # Is open pulse supported? E.g. True
config.max_shots            # 65536
config.coupling_map         # [[0, 1], [1, 0], [1, 2], [1, 6],
                            #  [2, 1], [2, 3], ...]
config.n_uchannels          # 46
config.u_channel_lo         # [[UchannelLO(q=1, scale=(1+0j))],
                            #   ...]
config.dt                   # 3.5555*1e-9 [s]
config.dtm                  # 10.5*1e-9 [s]
config.meas_map             # [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                            #   11, 12, 13, 14, 15, 16, 17, 18, 19]]

```

backend.properties()

- measured properties of the system, optionally reported by the backend provider

```
# Conversions from standard SI
us = 1e6
GHz = 1e-9
props = backend.properties()
# Qubit 0 T1 time in microsec                          # Example values
q0_t1_us = props.t1(0) * us                            # 27.5647 [us]
# Qubit 0 T2 time in microsec
q0_t2_us = props.t2(0) * us                            # 46.1596 [us]
# Gate error for CNOT on qubits 0, 1
cx_error = props.gate_error('cx', (0, 1))              # 0.0175
# Resonant frequency of qubit 2
q2_freq_ghz = props.frequency(2) * GHz                 # 4.6854 [GHz]
# Duration of the X gate on qubit 1
q1_x_duration_us = props.gate_length('u2', 1) * us     # 0.0355 [us]

```

`backend.defaults()` (OpenPulse systems only)

- default settings for operation
  - qubit frequencies
  - measurement readout frequencies
  - `Schedule` definitions for each of the backend's native gates

```
defs = backend.defaults()

q0_freq = defs.qubit_freq_est[0]          # 4.676678814180502 5 [GHz]
q0_meas_freq = defs.meas_freq_est[0]      # 7.285777689 [GHz]

inst_map = defs.instruction_schedule_map

```

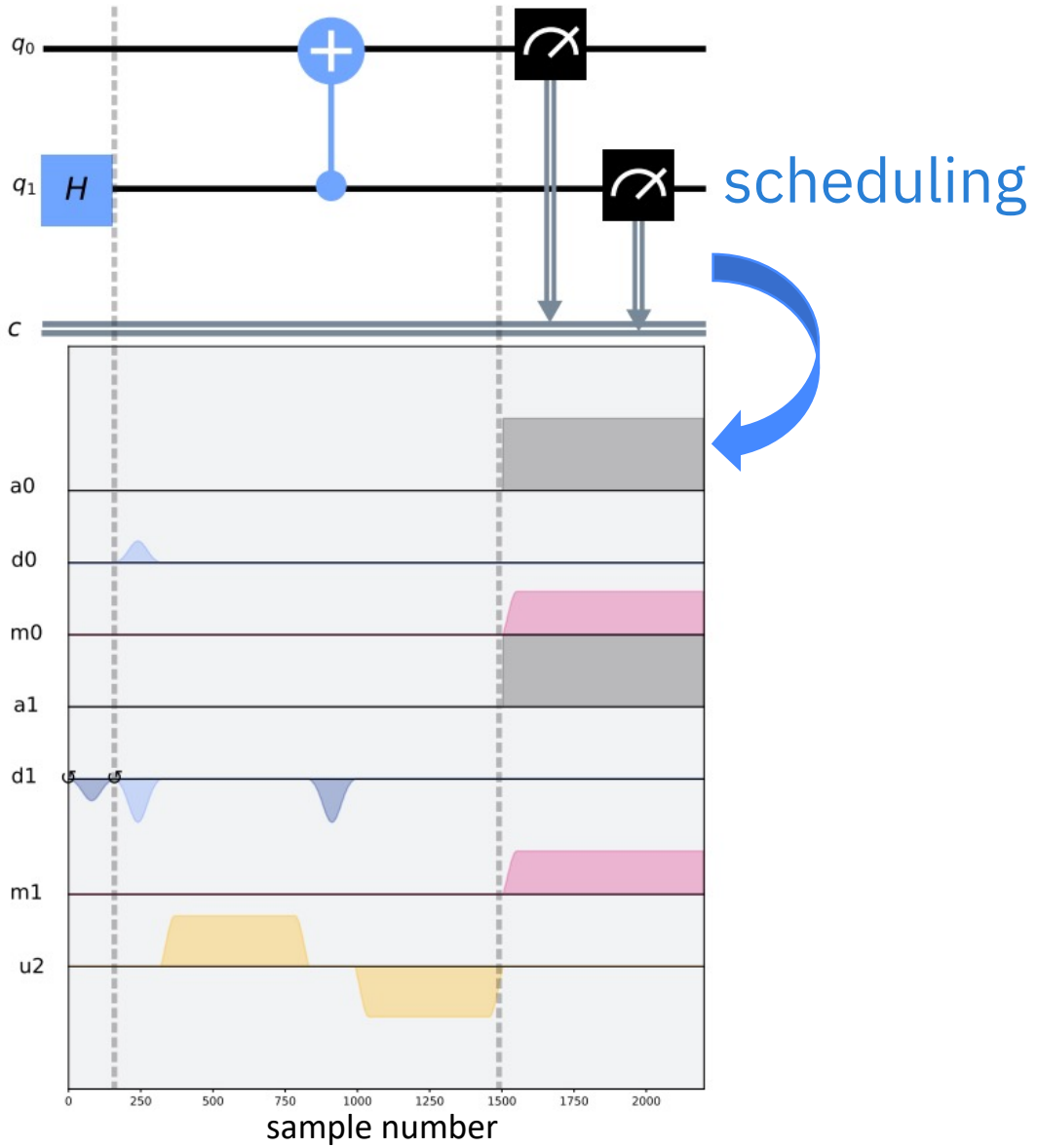
backend.defaults()
instruction_schedule_map

```
inst_map = backend.defaults().instruction_schedule_map

inst_map.get('measure',
             (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19))

inst_map.get('u1', 0, P0=3.1415)

inst_map.has('x', 3)  # Does qubit 3 have an x gate defined on it?


```

Circuit scheduling

- *scheduler*: $f(\text{circuit}) \rightarrow \text{schedule}$

```
from qiskit import schedule as build_schedule

qc = QuantumCircuit(2, 2)
qc.h(1)
qc.cx(1, 0)
qc.measure([0, 1], [0, 1])
transpiled_qc = transpile(qc, backend)
sched = build_schedule(transpiled_qc, backend)
```

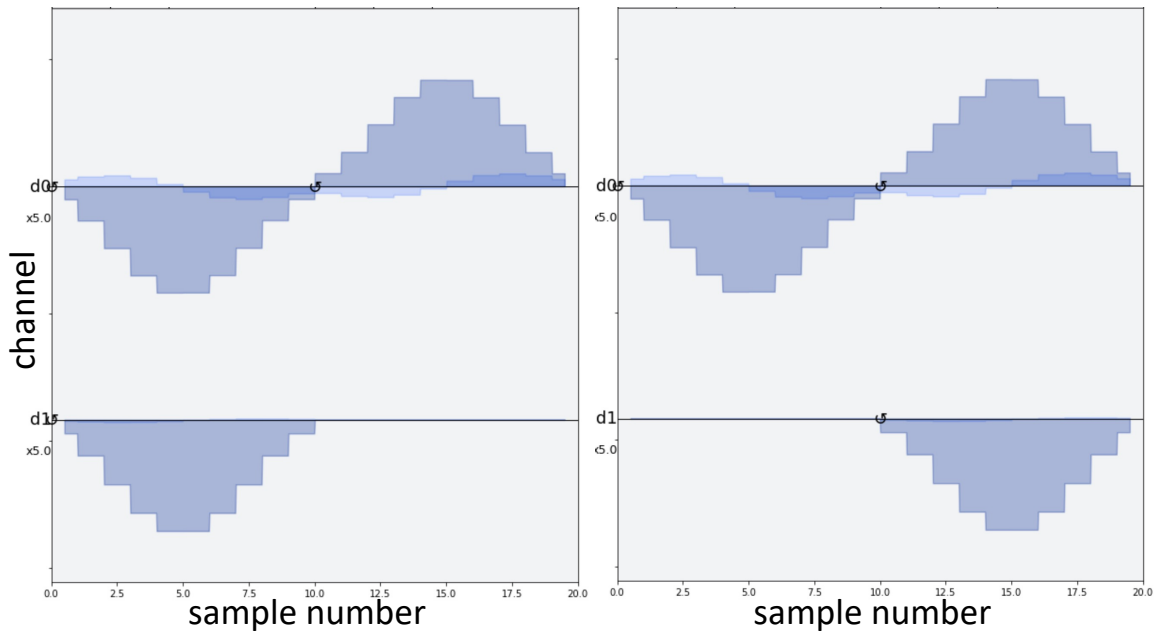


lauren@ibm.com

Circuit scheduling basic methods

- $DAG(V, E)$:
  - V: Pulse calibrations with *duration* and *resource* data
  - $\exists \mathrm{E}(V_i, V_j) \Rightarrow V_j$ must occur after V_i (e.g. operate on a shared resource)
- *minimal* schedule: Given the path $V_1 \ldots V_n$, in the DAG with the longest total duration, 1) the start time of V_{i+1} is the end time of V_i and 2) the start time of V_0 is 0.
- "as_soon_as_possible" (ASAP): $O(|V|)$
  - The start time of V_i is the maximum end time of its predecessors, or 0 if $\nexists E(V_k, V_i) \forall k$
- "as_late_as_possible" (ALAP): $O(|V|)$
  - The schedule is *minimal* and the end time of V_i is the minimum start time of its successors.

```
sched = build_schedule(transpiled_qc, backend,
                       method="as_soon_as_possible")
```



ASAP ALAP