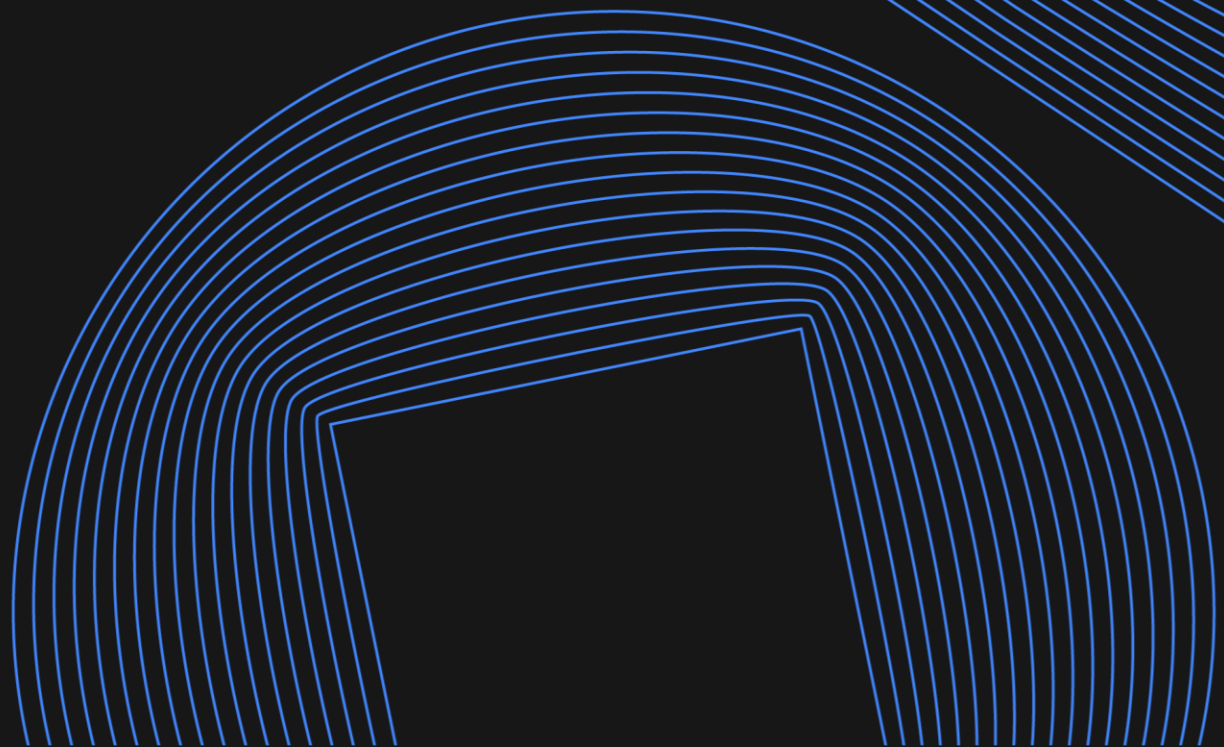
IBM Quantum
Qiskit Components

François Varchon, Ph.D
IBM Quantum Support Lead
francois.varchon@fr.ibm.com



IBM

Qiskit | Terra Aer Aqua Ignis IBM Q Account Community Tutorials Documentation

Welcome to Quantum

Qiskit is an open-source quantum computing software development framework for leveraging today's quantum processors in research, education, and business.

GET STARTED!

qiskit.org

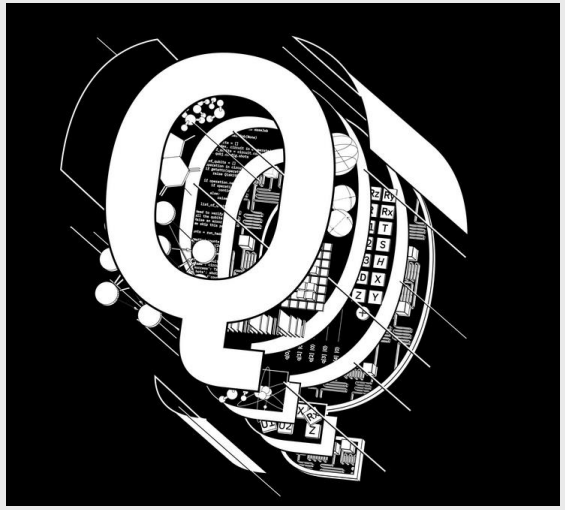
github.com/Qiskit

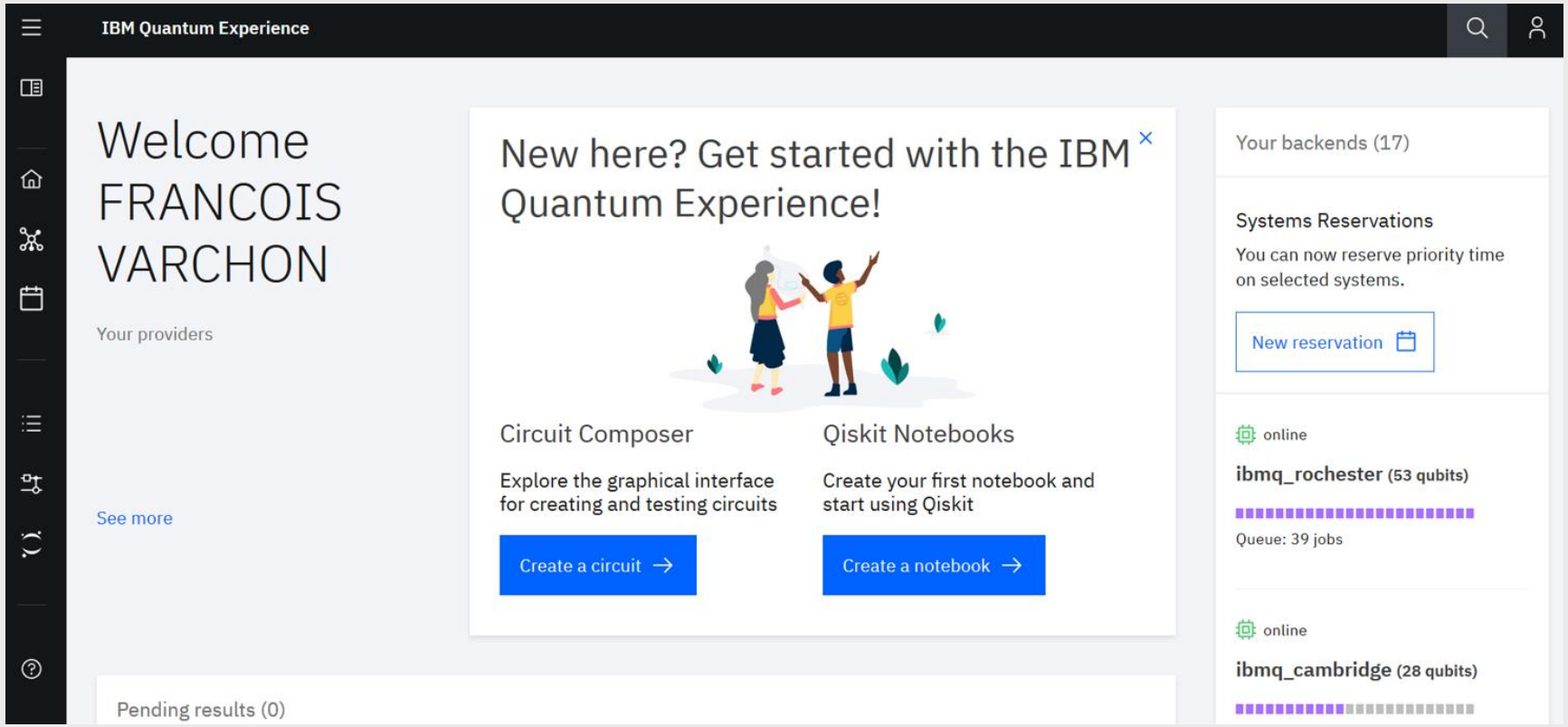
qiskit.slack.com

Qiskit in the cloud

IBM Quantum Experience

https://quantum-computing.ibm.com/





Installing Qiskit

Make sure Python >= 3.5 is available. Highly recommend installing Python using Anaconda (includes `jupyter notebook` and other Python packages).

```
$ pip install qiskit
```

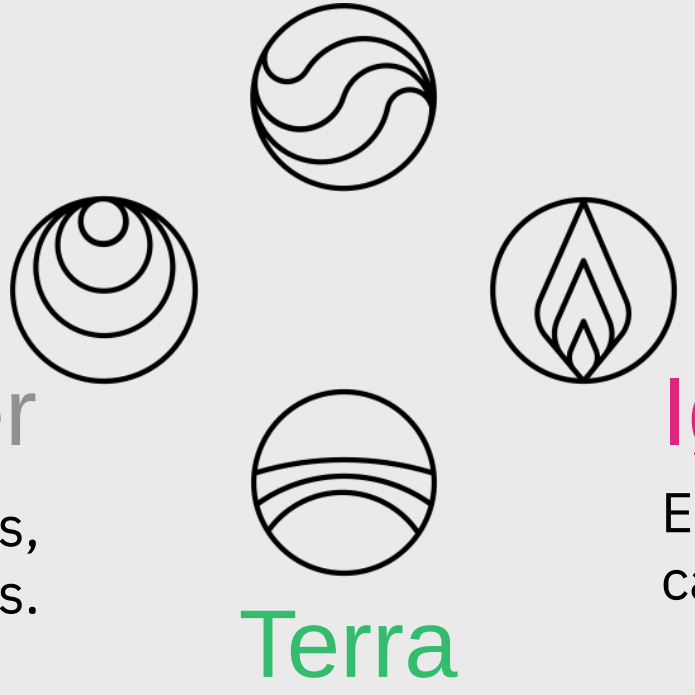
Check Qiskit

```
In [1]: import qiskit
        qiskit.__qiskit_version__

Out[1]: {'qiskit-terra': '0.14.1',
         'qiskit-aer': '0.5.2',
         'qiskit-ignis': '0.3.0',
         'qiskit-ibmq-provider': '0.7.2',
         'qiskit-aqua': '0.7.1',
         'qiskit': '0.19.3'}
```

The elements:

Algorithms and applications for noisy, near-term quantum hardware.

Aqua



Aer

Classical simulators, emulators, and debuggers.

Ignis

Error characterization, validation, calibration, and mitigation.

Terra

Circuits, pulses, and optimizations for running on quantum devices.

Foundation on which the rest of Qiskit lies

Qiskit Terra contains tools that **define**, **compile** and **execute** quantum circuits on arbitrary **backends.** It is organized in several modules**:**

Circuits: fundamental element of quantum computing, coherent quantum operations on quantum data, such as qubits

Pulse: lower level than circuits, pulses are sent to a quantum experiment that are applied to a channel (experimental input line). At this level the experiments can be designed to reduce errors.

Transpiler: how to run a quantum circuits on real devices. Explore optimization and find better quantum circuits

Quantum Information: advanced methods to estimate metrics and generate quantum states, operations, and channels

Visualization: tools to visualize and to plot the results from a quantum circuit

Foundation on which the rest of Qiskit lies

Qiskit Terra contains tools that **define**, **compile** and **execute** quantum circuits on arbitrary **backends.** It is organized in several modules**:**

Circuits: fundamental element of quantum computing, coherent quantum operations on quantum data, such as qubits

Pulse: lower level than circuits, pulses are sent to a quantum experiment that are applied to a channel (experimental input line). At this level the experiments can be designed to reduce errors.

Transpiler: how to run a quantum circuits on real devices. Explore optimization and find better quantum circuits

Quantum Information: advanced methods to estimate metrics and generate quantum states, operations, and channels

Visualization: tools to visualize and to plot the results from a quantum circuit

Hello world with Qiskit Terra

Bell State $\frac{(|00\rangle + |11\rangle)}{\sqrt{2}}$

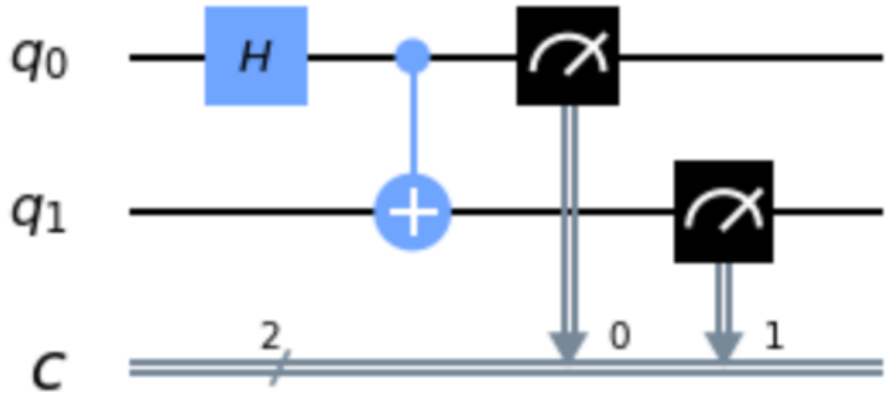
In [1]:
```
from qiskit import QuantumCircuit
```

In [2]:
```
circuit = QuantumCircuit(2, 2)
circuit.h(0)
circuit.cx(0,1)
circuit.measure([0,1],[0,1])
```

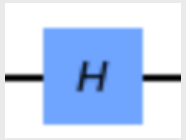
Out[2]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x23857a21790>
```

```
In [3]: circuit.draw(output='mpl')
```

Out[3]:

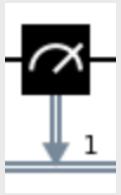


- Hadamard Gate (superposition)



- CNOT Gate (entanglement)

- Measure



Set up IBMQ account and provider (copy your token from IBM Quantum Experience)

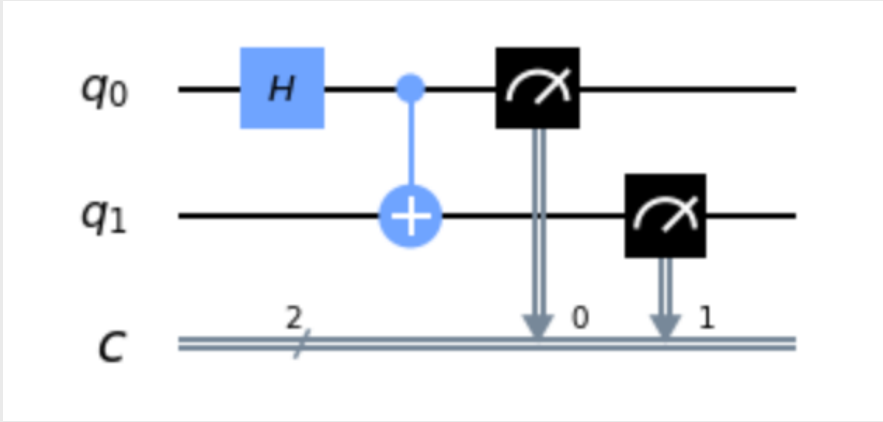
```
IBMQ.save_account('<Your Token>')
IBMQ.load_account()
provider=IBMQ.get_provider(hub='ibm-q-fraunhofer', group='yourgroup', project='yourproject')
```

Choose a backend

```
backend=provider.backends.ibmq_singapore
```

Execute

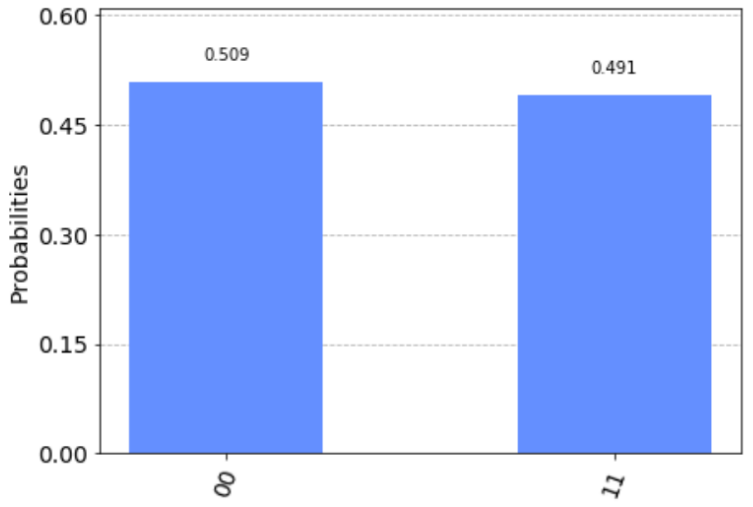
```
job=execute(circuit,backend,shots=8192)
```



Simulator

```
In [54]: from qiskit.visualization import plot_histogram

In [53]: plot_histogram(job.result().get_counts())
```

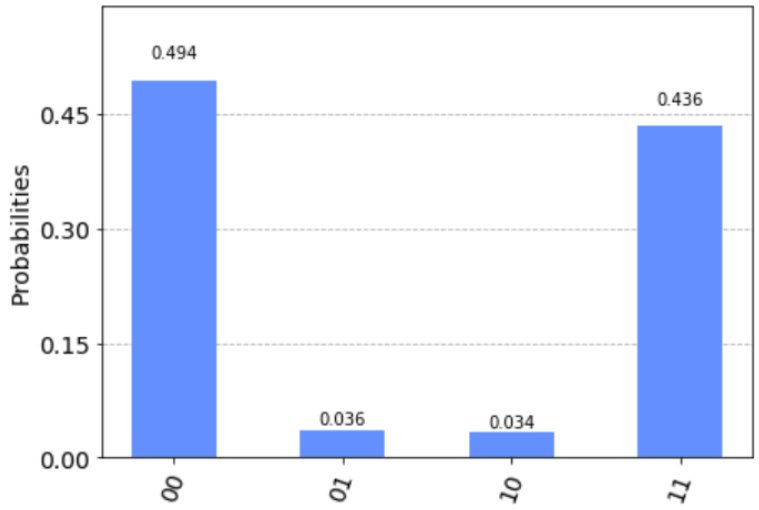
Out[53]:



Quantum Hardware

```
In [45]: from qiskit.visualization import plot_histogram

In [48]: plot_histogram(job.result().get_counts())
```

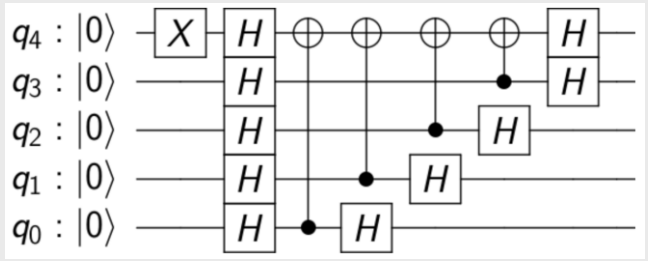
Out[48]:

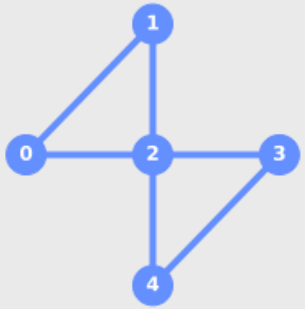


Real Backend

Running on real hardware devices:

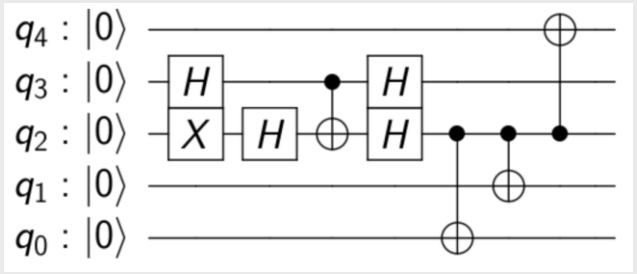
- **Ample amounts of noise, limited connectivity**.
- Must rewrite circuits for topology, and to minimize gate count / depth

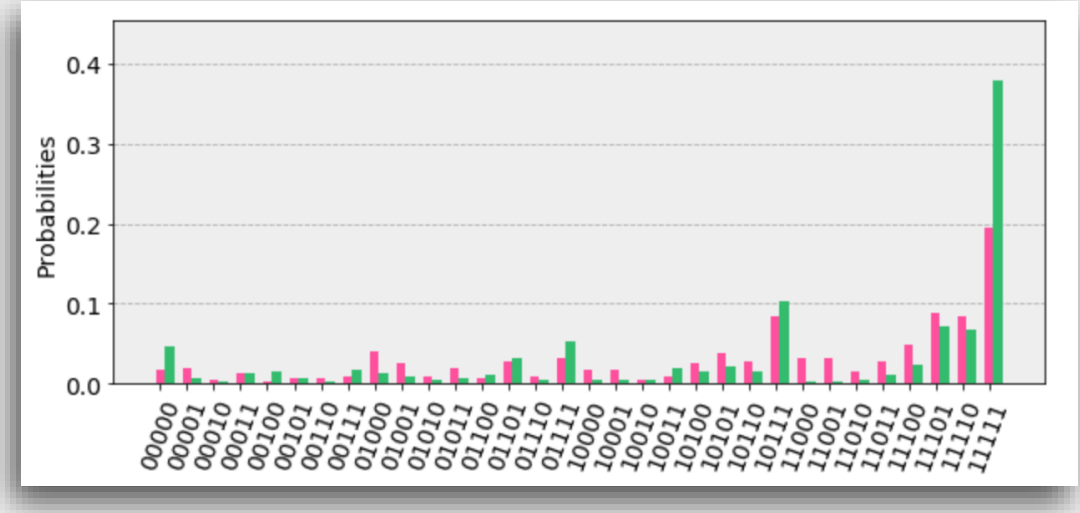




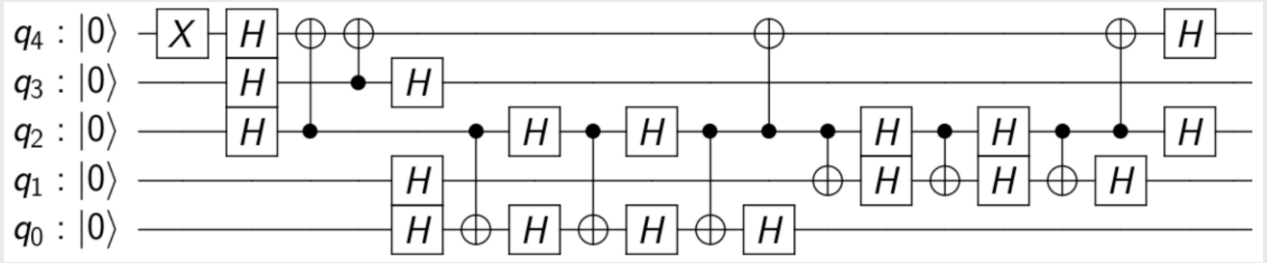
ibmq_5_tenerife

Good





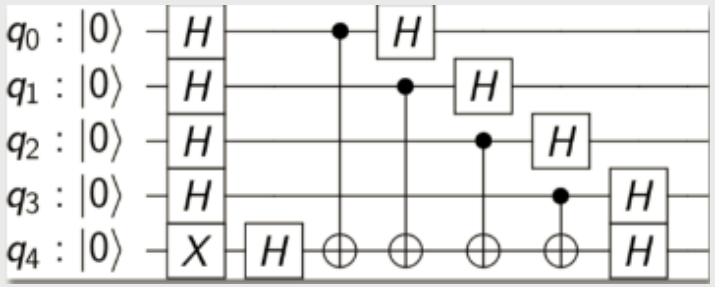
Bad



Qiskit Terra - Transpiler

Terra

Each pass does one small, well-defined task.



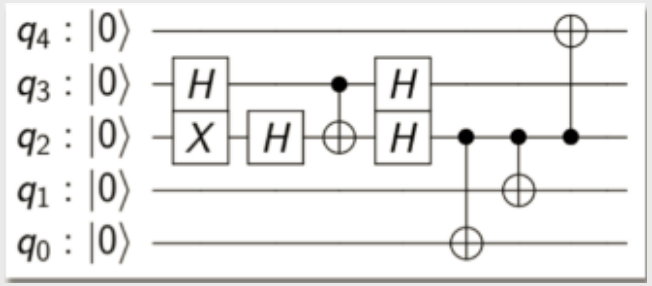
Pass 1 → Pass 2 → Pass 3 → Pass 4

Unroll to Basis

Gate cancellation

Topology mapping

Gate cancellation



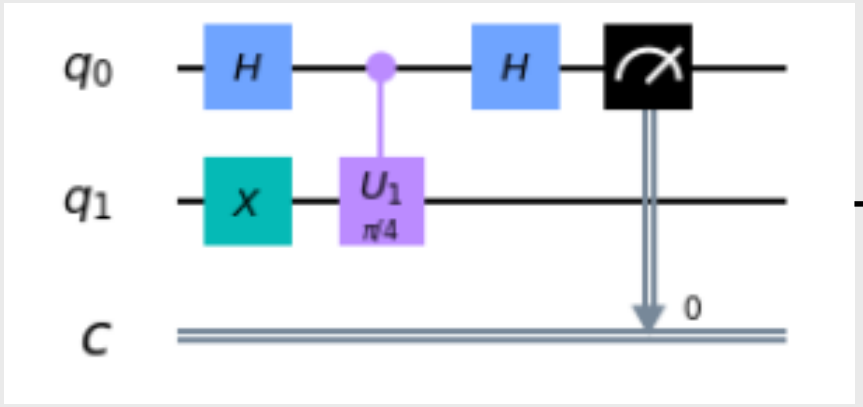
Pass Manager

- Depth
- Layout
- CX Directions
- Fixed point conditions

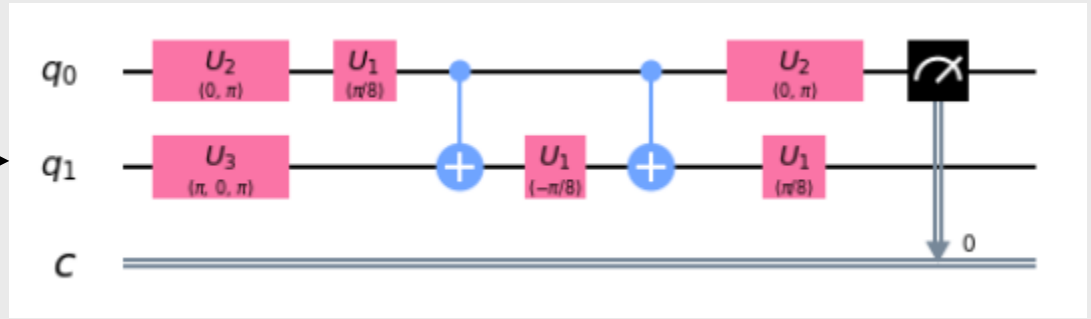
Unroll to basis

A quantum computing device can only natively support a handful of quantum gates and non-gate operations. Taken care for us in the execute function.

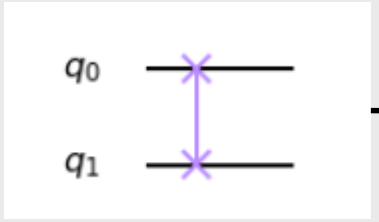
```
In [4]: from qiskit.transpiler import PassManager
        from qiskit.transpiler.passes import Unroller
        pass_ = Unroller(['u1', 'u2', 'u3', 'cx'])
        pm = PassManager(pass_)
        new_circ = pm.run(circ)
        new_circ.draw(output='mpl')
```

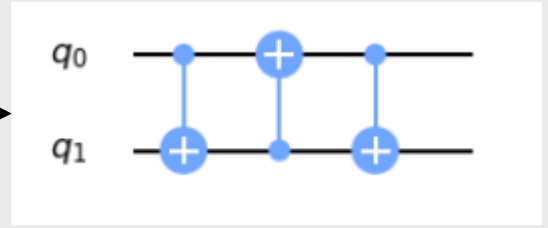


decomposition

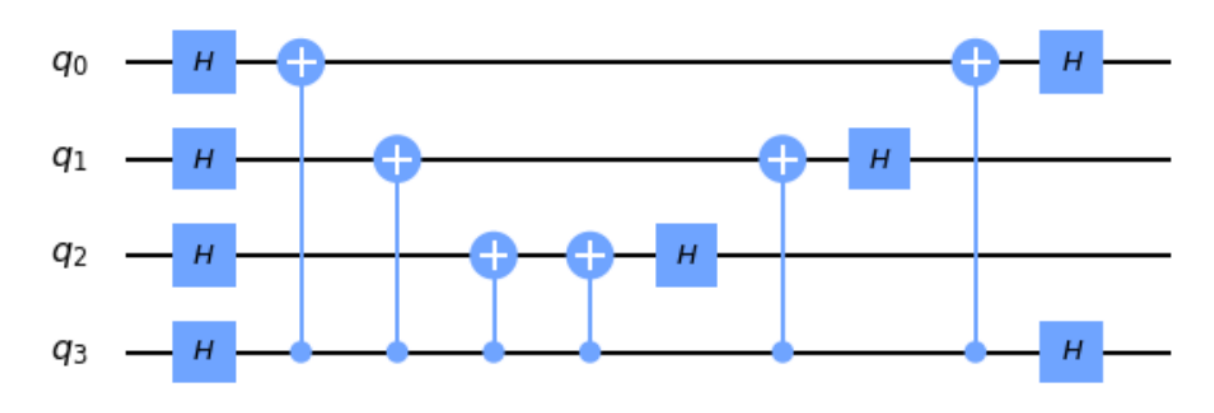


Swap decomposition

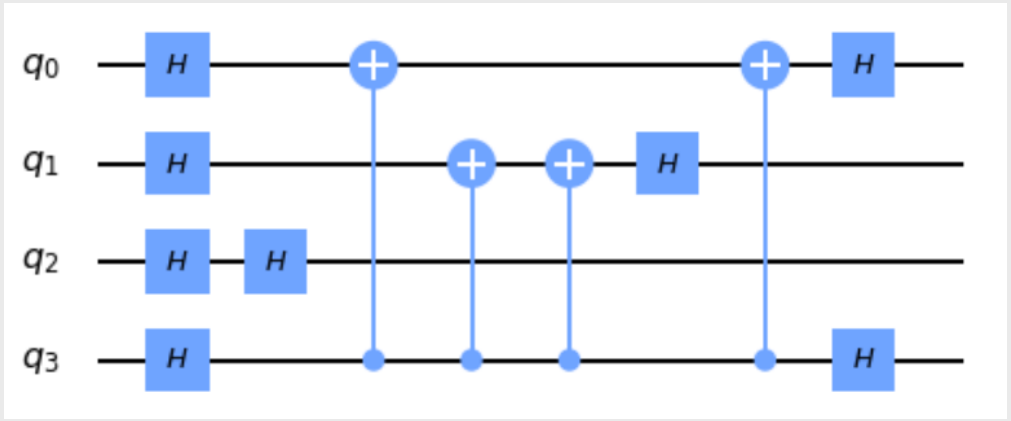




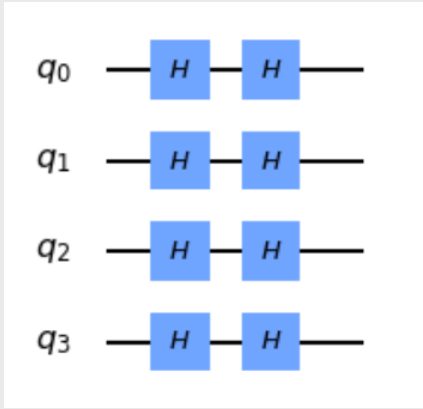
Gate Cancellation




```
from qiskit.transpiler.passes import CXCancellation
pass_ = CXCancellation()
pm = PassManager(pass_)
new_circ = pm.run(circ)
new_circ.draw(output='mpl')
```

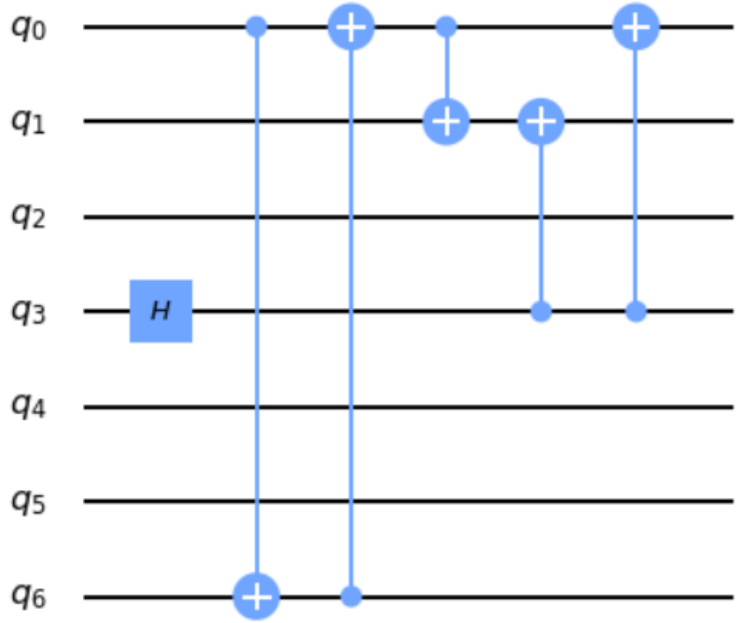



```
new_circ = pm.run(new_circ)
new_circ.draw(output='mpl')
```

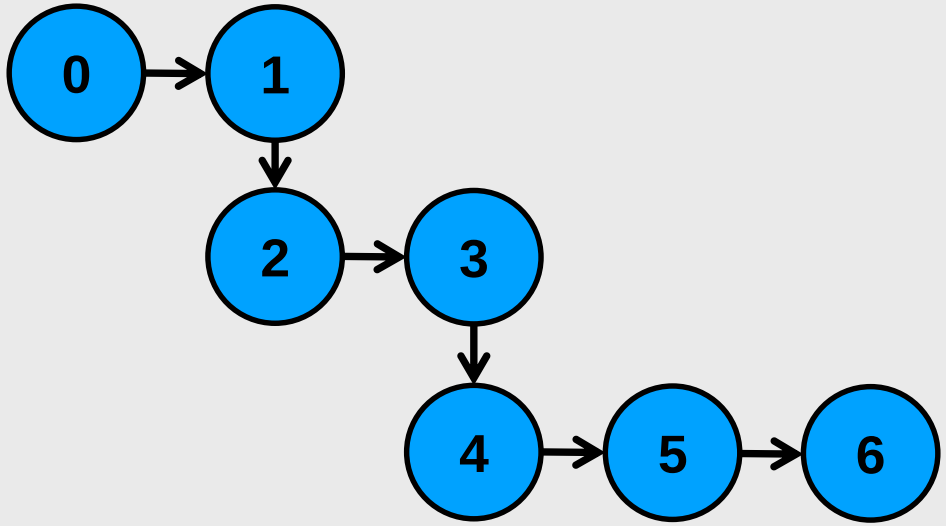


Topology mapping

```
circuit = QuantumCircuit(7)
circuit.h(3)
circuit.cx(0, 6)
circuit.cx(6, 0)
circuit.cx(0, 1)
circuit.cx(3, 1)
circuit.cx(3, 0)
```



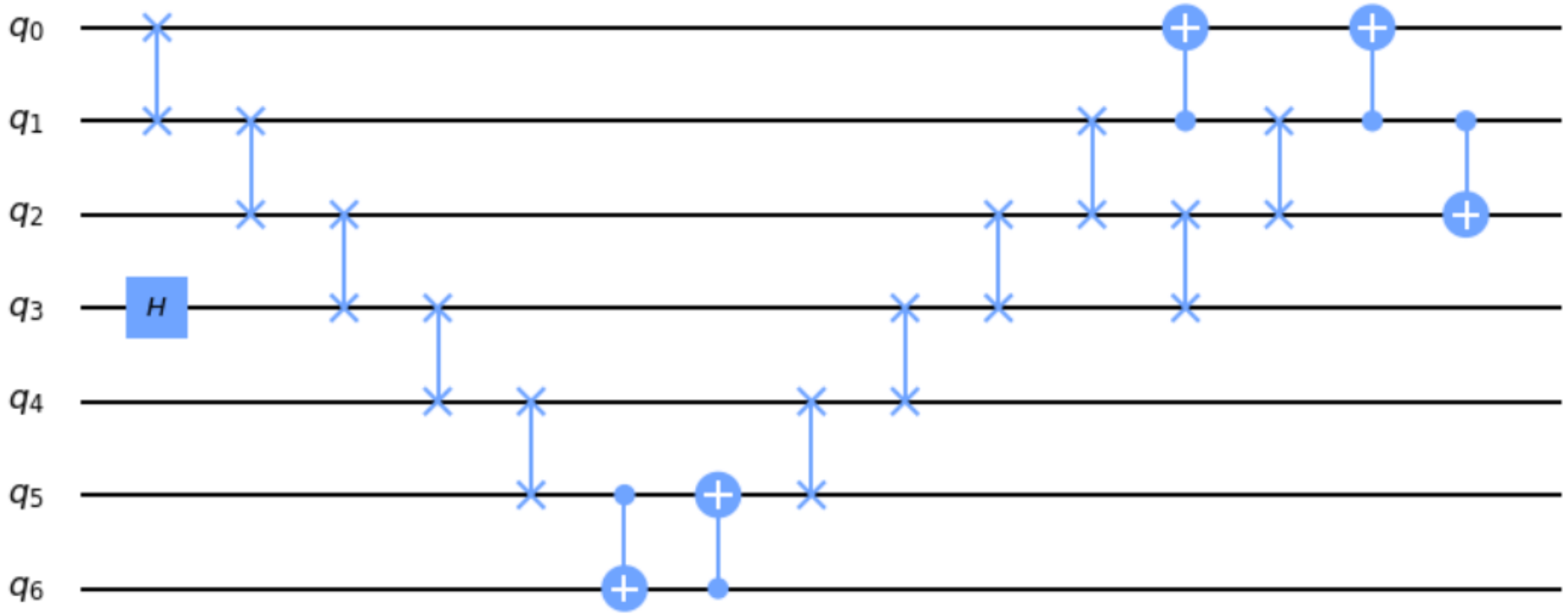

```
from qiskit.transpiler import CouplingMap

coupling = [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 6]]
coupling_map = CouplingMap(couplinglist=coupling)
```



Transpiler passes

```
from qiskit.transpiler.passes import BasicSwap, LookaheadSwap, StochasticSwap
```

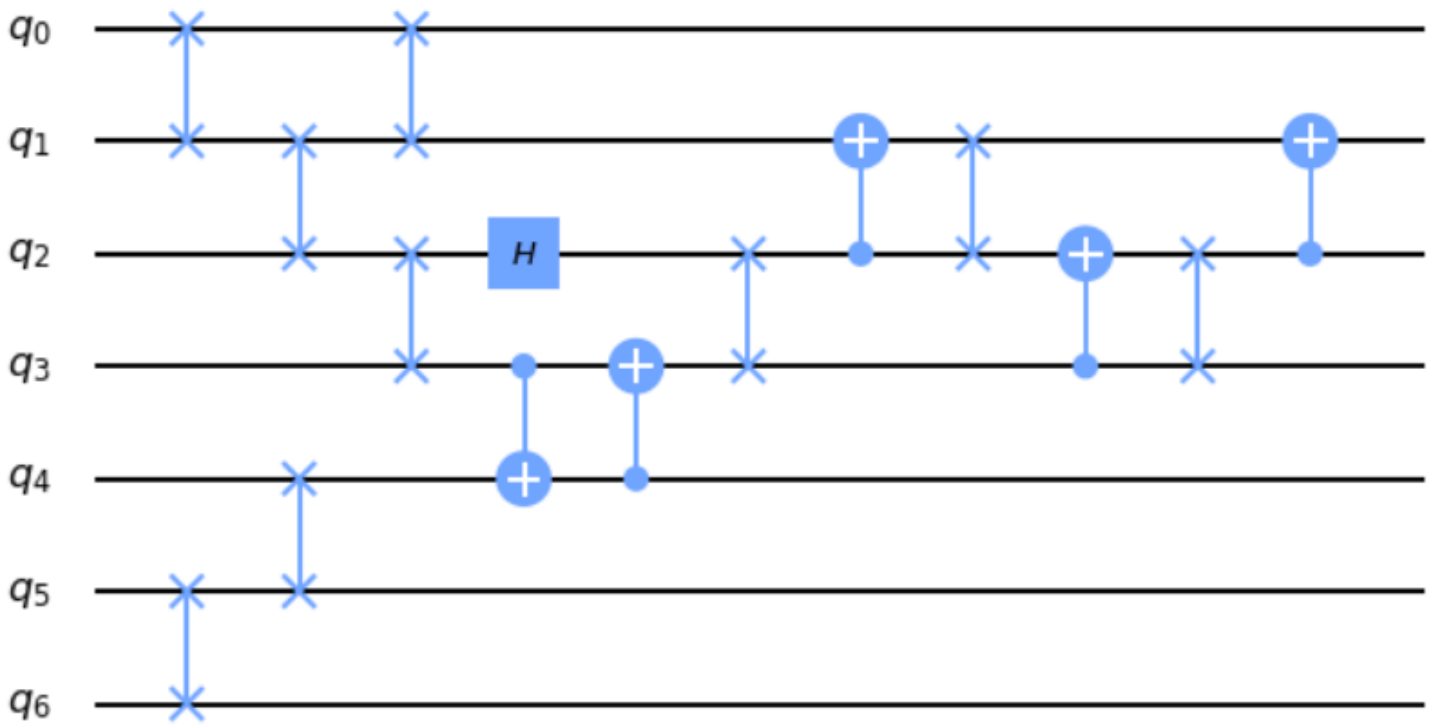
```
bs = BasicSwap(coupling_map=coupling_map)
pass_manager = PassManager(bs)
basic_circ = pass_manager.run(circuit)
basic_circ.draw(output='mpl')
```



Transpiler passes

```
from qiskit.transpiler.passes import BasicSwap, LookaheadSwap, StochasticSwap
```

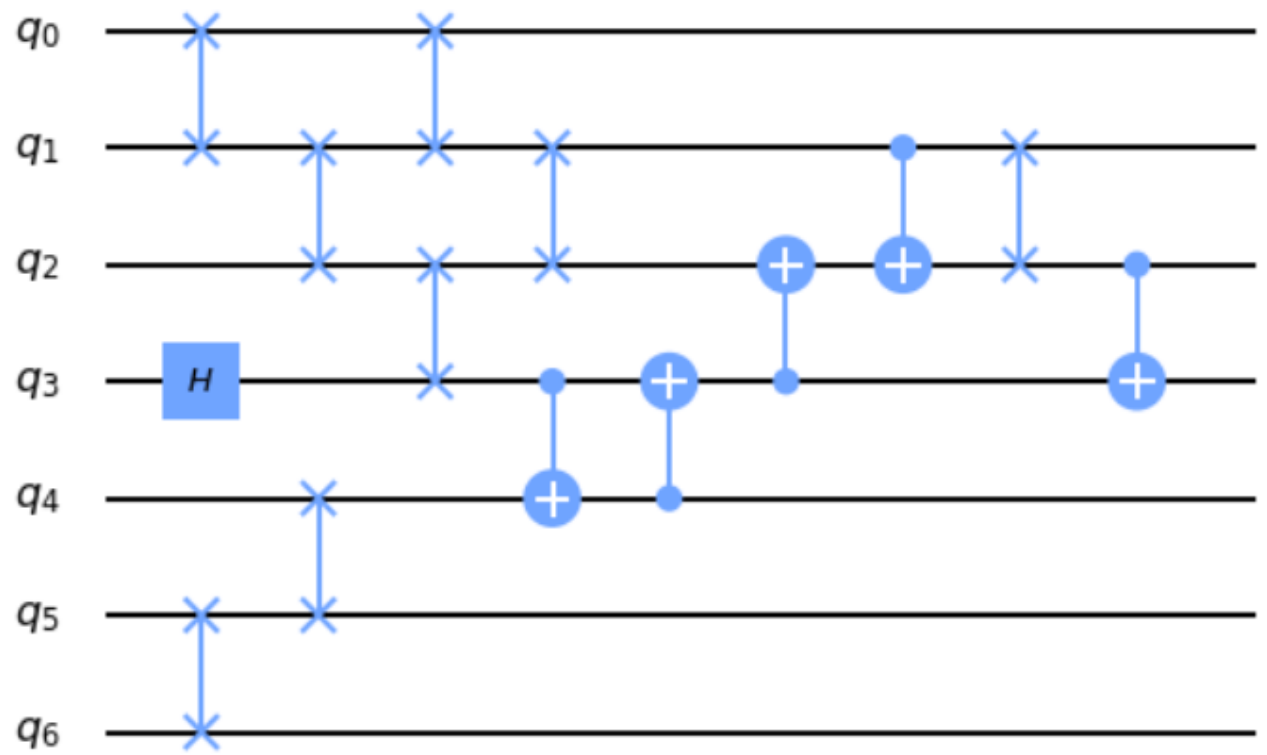
```
ss = StochasticSwap(coupling_map=coupling_map)
pass_manager = PassManager(ss)
stochastic_circ = pass_manager.run(circuit)
stochastic_circ.draw(output='mpl')
```



Transpiler passes

```
from qiskit.transpiler.passes import BasicSwap, LookaheadSwap, StochasticSwap
```

```
ls = LookaheadSwap(coupling_map=coupling_map)
pass_manager = PassManager(ls)
lookahead_circ = pass_manager.run(circuit)
lookahead_circ.draw(output='mpl')
```



Preset Pass Managers

Qiskit comes with several pre-defined pass managers, corresponding to various levels of optimization achieved through different pipelines of passes (each level contains all optimizations of earlier levels).

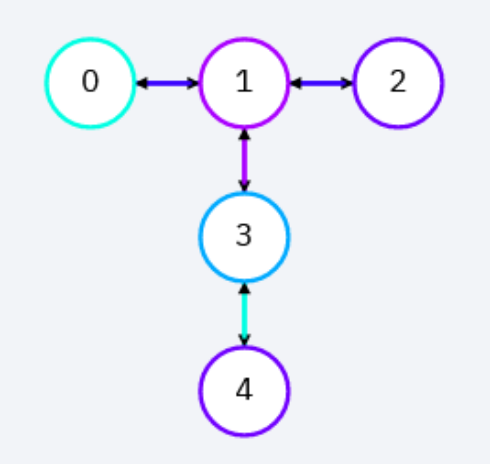
- `optimization_level=0`: **No optimization**. Just maps the circuit to the backend.
- `optimization_level=1`: (Default.) **Light optimization**. Noise-aware, dense layout search. Single qubit gate optimization.
- `optimization_level=2`: **Medium optimization**. CSP layout search, commutation analysis.
- `optimization_level=3`: **Heavy optimization**. Block collection and optimal re-synthesis. Redundant gate removal.

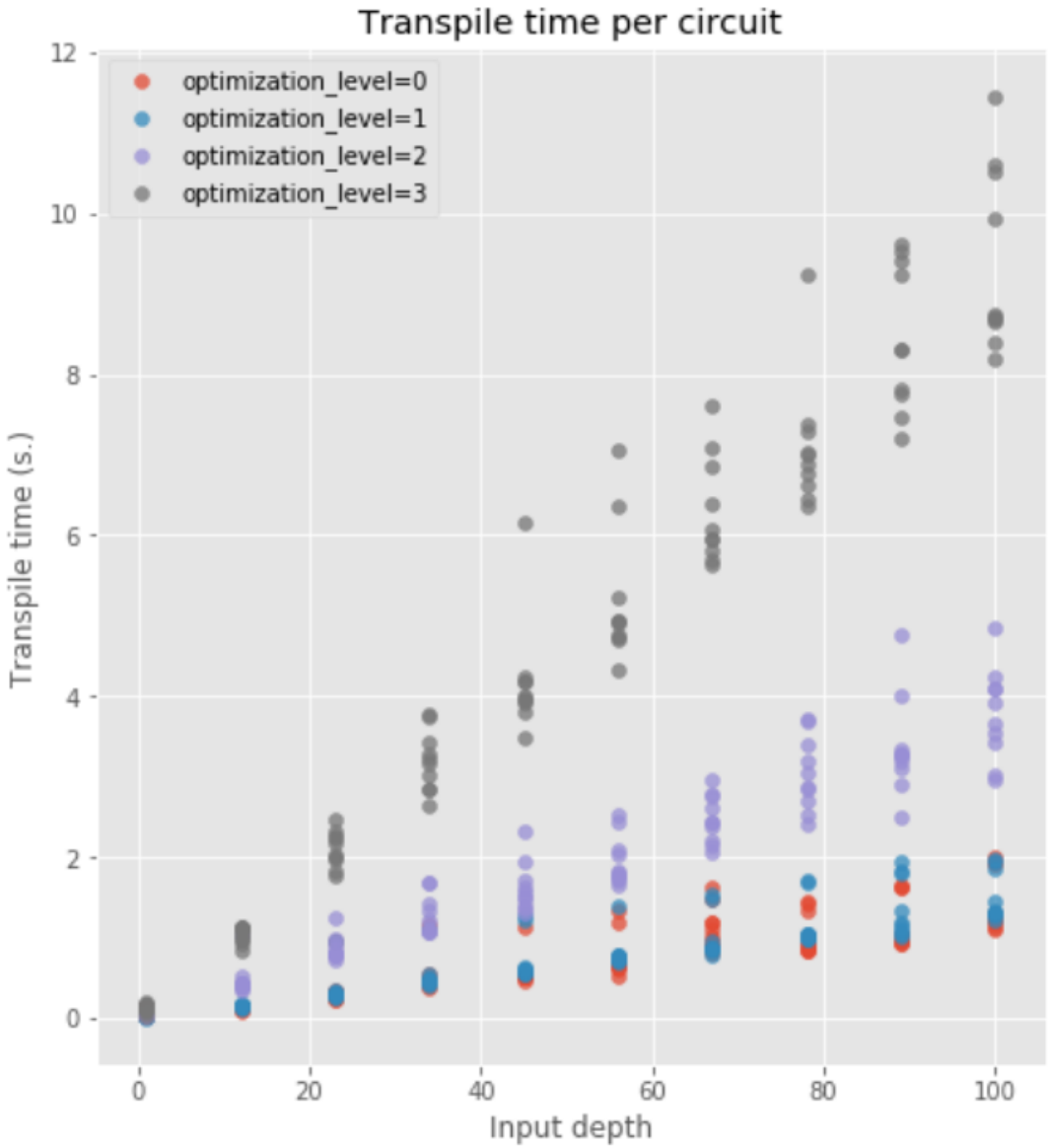
Optimization level

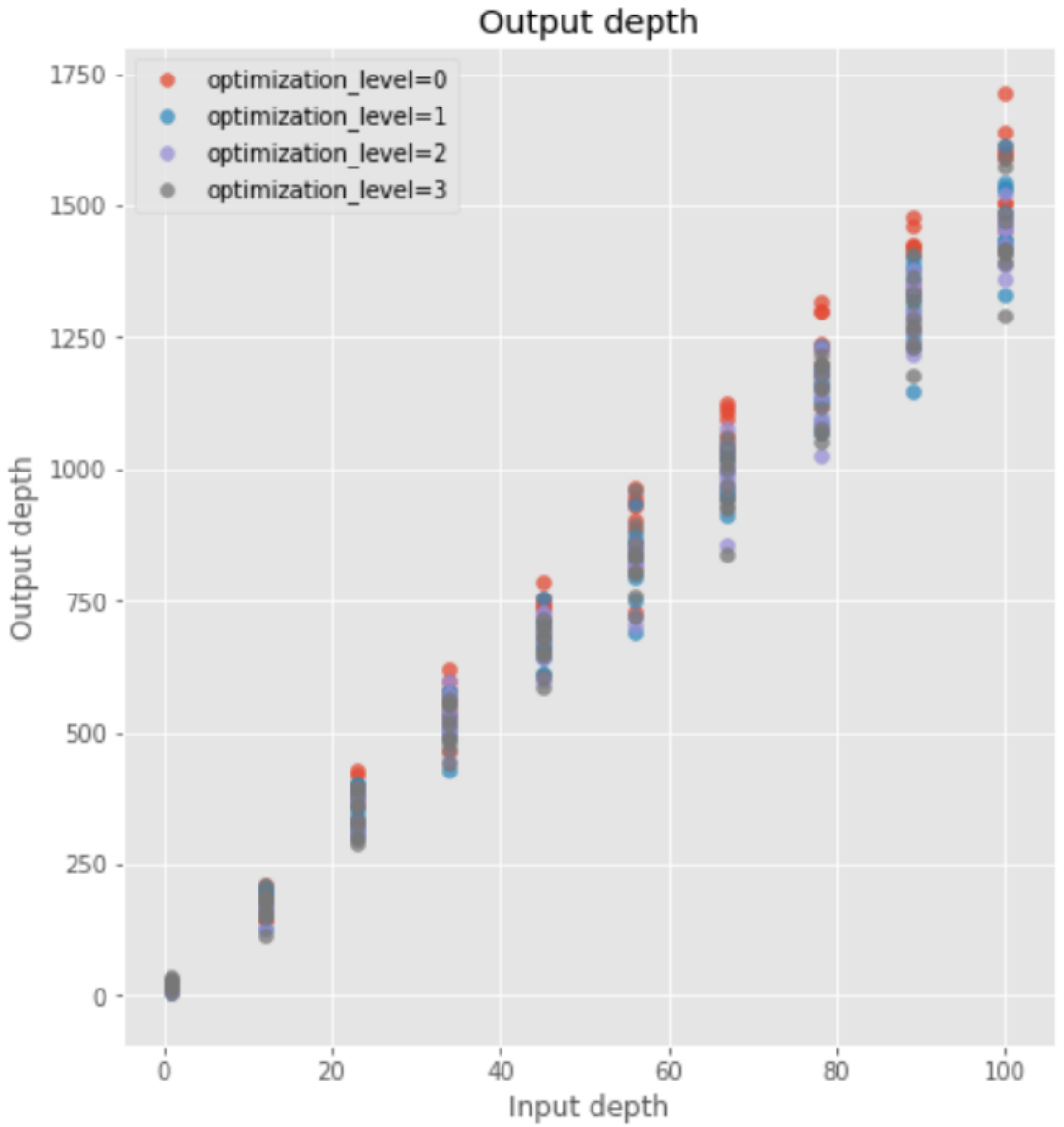
Random circuits (5 qubits – depth from 1 to 100)

```
import numpy as np
from qiskit.circuit.random import random_circuit

random_circuits = {input_depth: [random_circuit(5, int(input_depth), measure=True, seed=seed)
                   for seed in range(10)]
                   for input_depth in np.linspace(1, 100, num=10)}
```

ibmq_vigo







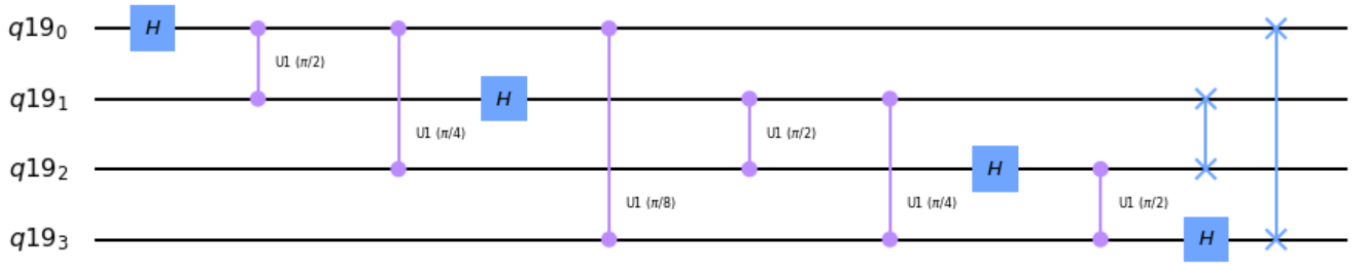
Thanks to Kevin Krsulich - https://www.youtube.com/watch?v=2T3163VjvWQ&

Circuit Library (`qiskit.circuit.library`)

Terra

Qiskit >=0.19.0

```
from qiskit.circuit.library import QFT

qft_circuit =QFT(4)
qft_circuit.draw(output='mpl',scale=0.7)
```



The circuit library gives users access:

- Rich set of well-studied circuit families,
- Building blocks in building more complex circuits
- Tool to explore quantum computational advantage over classical.
- **Contents of this library will continue to grow**

Library overview

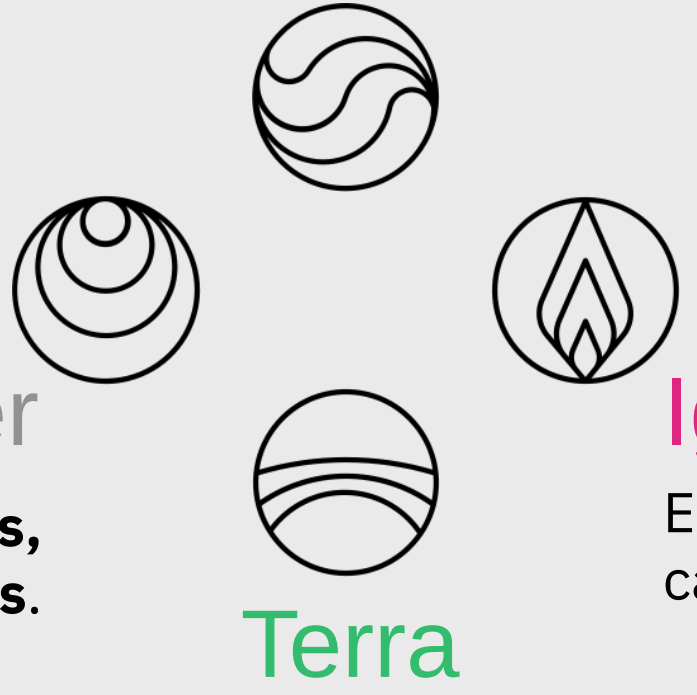
Circuit library

Boolean circuits	Generalized gates	Arithmetic	Others	N-local	Data preparation
AND	Global MS	Comparator	QFT	TwoLocal	ZFeatureMap
OR	Diagonal	LinearRot	QuantumVolume	RealAmplitudes	ZZFeatureMap
XOR	MCMT	PolynomialRot	IQP	EfficientSU2	Pauli FeatureMap
	Permutation	Piecewise-linearRot	HiddenLinear Function	Excitation Preserving	
		Adder	Fourier Checking		
			GraphState		

The elements:

Algorithms and applications for noisy, near-term quantum hardware.

Aqua



Aer

Classical simulators, emulators, and debuggers.

Ignis

Error characterization, validation, calibration, and mitigation.

Terra

Circuits, pulses, and optimizations for running on quantum devices.

Qiskit Aer: Simulator

Qiskit Aer includes four high performance simulator backends:

- `qasm_simulator`: Allows ideal and noisy multi-shot execution of qiskit circuits and returns counts or memory
- `statevector_simulator`: Allows ideal single-shot execution of qiskit circuits and returns the final statevector of the simulator after application.
- `unitary_simulator`: Allows ideal single-shot execution of qiskit circuits and returns the final unitary matrix of the circuit itself.
- `pulse_simulator`: Pulse schedule simulator backend.

Qiskit Aer: Advanced

Advanced methods to run circuits efficiently for `qasm_simulator`

```
backend = QasmSimulator()
backend_options = {"method": "statevector"}

# Circuit execution
job = execute(circuits, backend, backend_options=backend_options)

# Qobj execution
job = backend.run(qobj, backend_options=backend_options)
```

backend_options https://qiskit.org/documentation/stubs/qiskit.providers.aer.QasmSimulator.html

- ***statevector*** - Uses a dense statevector simulation.
- ***stabilizer*** - Uses a Clifford stabilizer state simulator that is only valid for Clifford circuits and noise models.
- ***extended_stabilizer*** - Uses an approximate simulator that decomposes circuits into stabilizer state terms, the number of which grows with the number of non-Clifford gates.
- ***matrix_product_state*** - Uses a Matrix Product State (MPS) simulator.

Qiskit Aer: Advanced

Run simulation with noise

```
from qiskit import Aer,execute
from qiskit.providers.aer.noise import NoiseModel
# Choose a real device to build the noise model.
device = provider.get_backend('ibmq_16_melbourne')

# Retrieve the device properties and the coupling map
properties = device.properties()
coupling_map = device.configuration().coupling_map

# Generate the basic noise model from device
noise_model = NoiseModel.from_backend(device)

# Retrieve the basis gates from the device
basis_gates = noise_model.basis_gates

# Submit a job for the circuit on the simulator without noise.
backend = Aer.get_backend('qasm_simulator')

# Perform noisy simulation
job_simulation_with_noise = execute(circuit, backend,
                            coupling_map=coupling_map,
                            noise_model=noise_model,
                            basis_gates=basis_gates)
```

Compatible with GPU*

```
pip install qiskit-aer-gpu
```

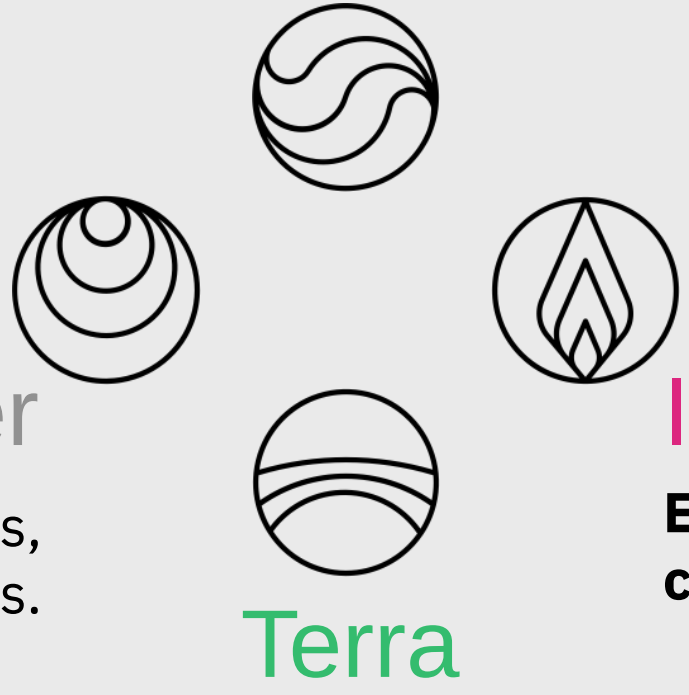
This will overwrite your current qiskit-aer package installation giving you the same functionality found in the canonical `qiskit-aer` package, plus the ability to run the GPU supported simulators

*In order to install and run the GPU supported simulators, you need CUDA® 10.1 or newer previously installed.

The elements:

Algorithms and applications for noisy, near-term quantum hardware.

Aqua



Aer

Classical simulators, emulators, and debuggers.

Ignis

Error characterization, validation, calibration, and mitigation.

Terra

Circuits, pulses, and optimizations for running on quantum devices.

Qiskit Ignis

Ignis

The toolkit is divided into topics based on common functionality of the circuits

Characterization

Experiments to isolate individual noise parameters

Incoherent Noise
- T1
- T2/CPMG - Hahn Echo experiment
- T2* - Ramsey experiment

Coherent Noise
- ZZ

Verification

Experiments to measure gate errors, reconstruct states and process maps

Randomized Benchmarking
- Sequences of random Clifford gates that equal the identity
- Ground state population vs sequence length is a simple measure of gate fidelity
- 1+2Q RB, simultaneous

Tomography
- State and Process
- Different fitting procedures

Mitigation

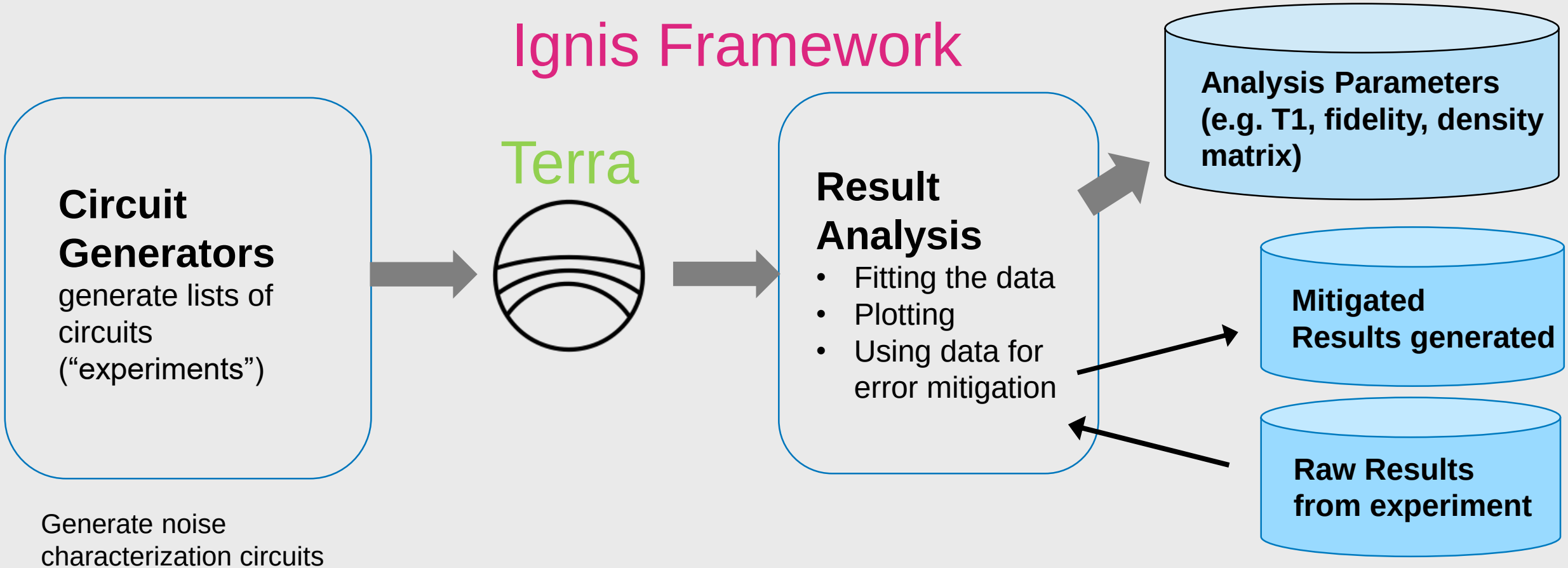
Experiments to mitigate the role of noise in circuits

Measurement
- Measure outcomes of circuits that prepare a single state
- Use to construct a calibration that can be applied to other circuits to correct measurement errors *on average*

Measurement Error Mitigation

Ignis

Ignis is a toolkit for understanding
and mitigating noise in Quantum circuits and devices



Measurement Error Mitigation

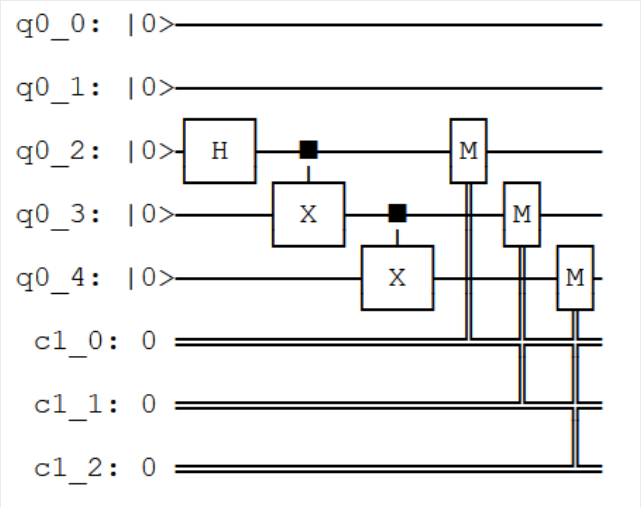
1/ Calibration

In [8]:
```
# Execute the calibration
backend = IBMQ.get_backend('ibmq_poughkeepsie')
job = qiskit.execute(meas_calibs, backend=backend, shots=1024)
cal_results = job.result()
```

2/ Test 3Q GHZ State

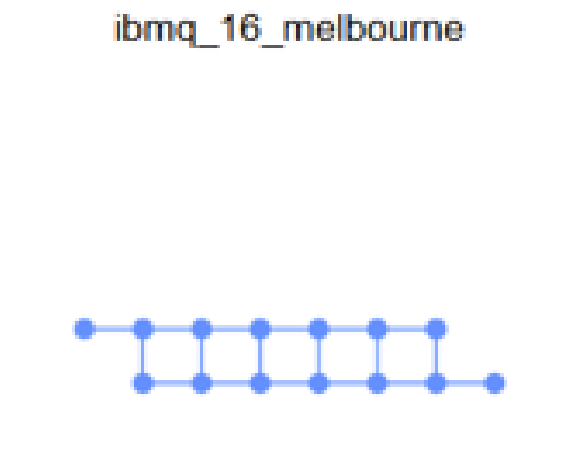
As an example, we start with the 3-qubit GHZ state on the qubits Q2,Q3,Q4:

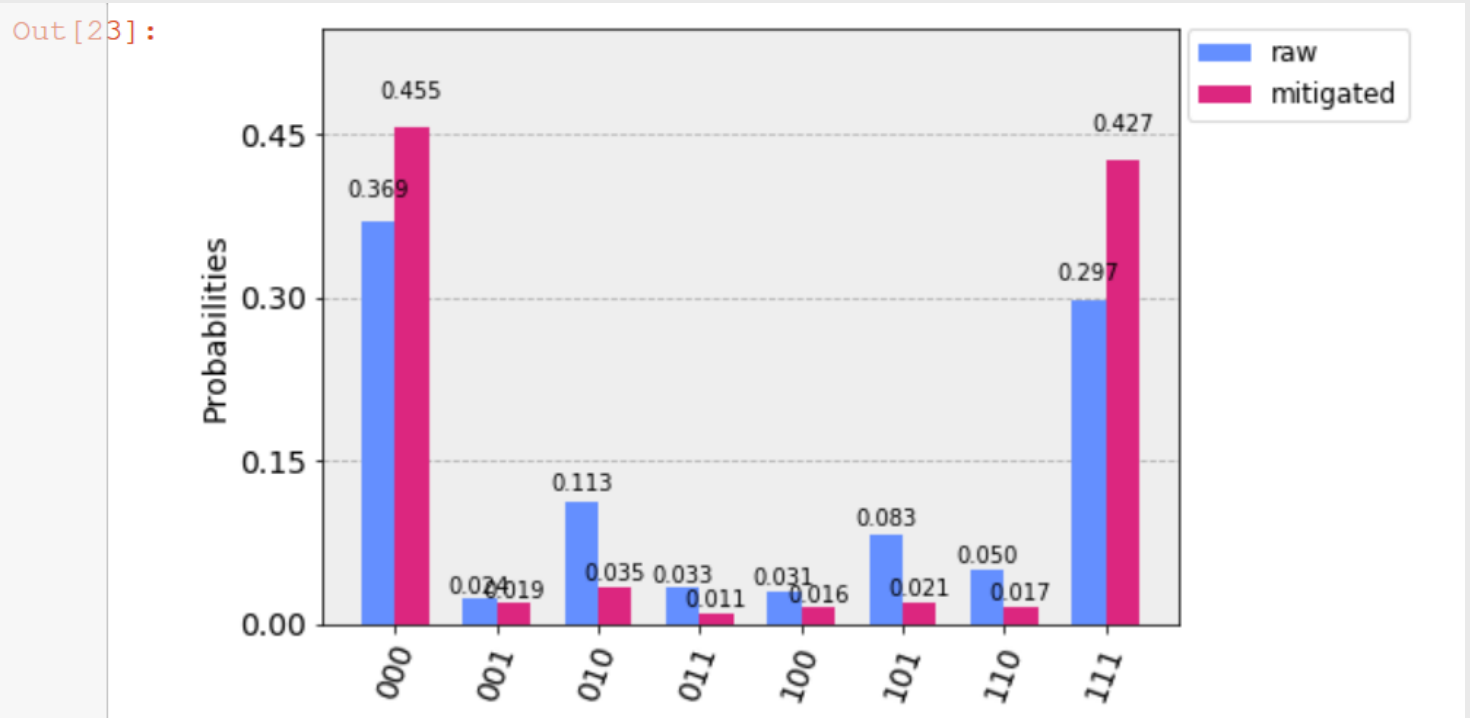
$$|GHZ\rangle = \frac{|000\rangle + |111\rangle}{\sqrt{2}}$$



Measurement Error Mitigation

3/ Correct measurement Noise

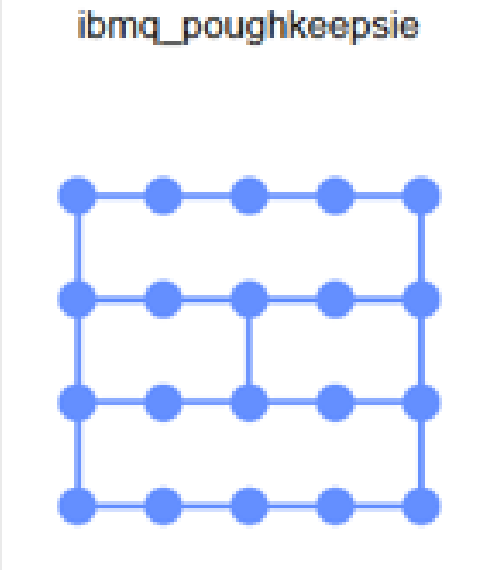


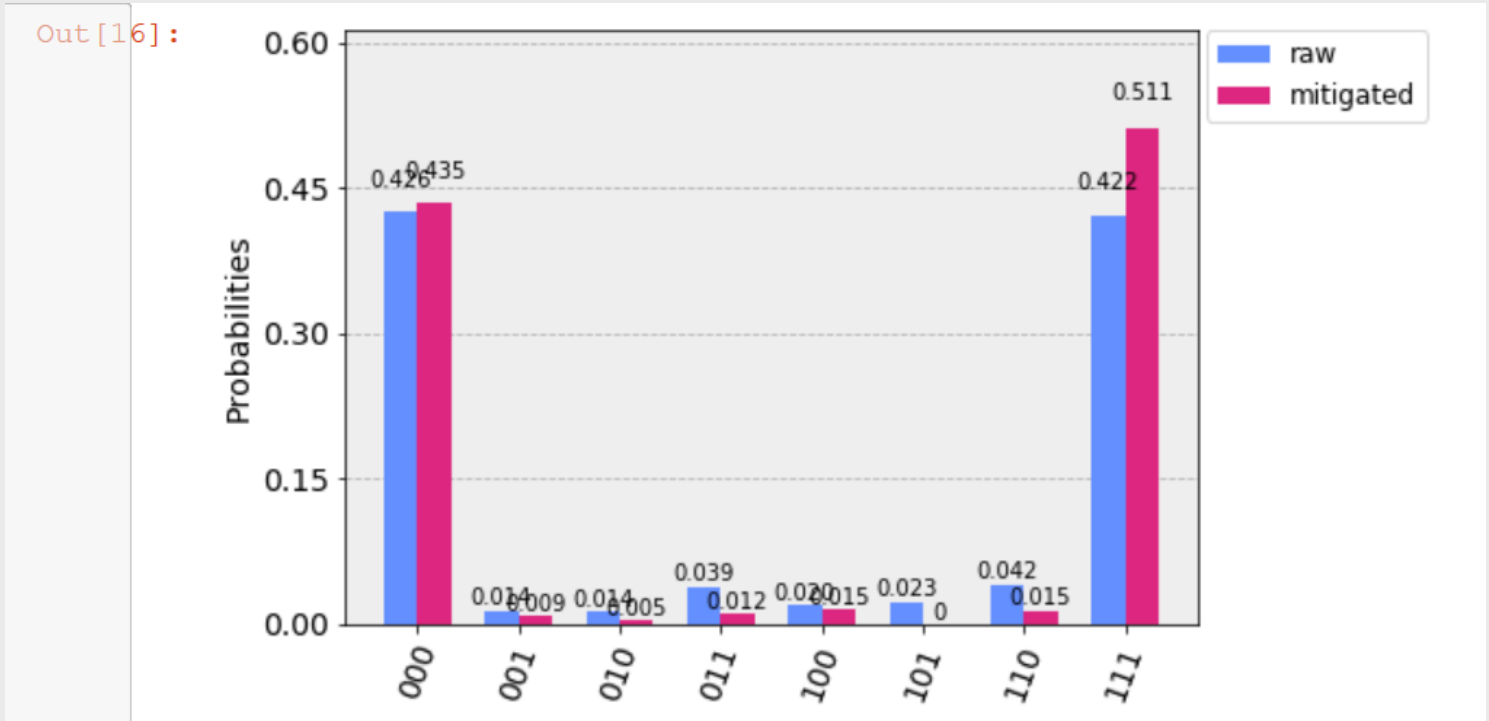


Mitigating Noise on Real Quantum Computers
https://www.youtube.com/watch?v=yuDxHJOKsVA

Measurement Error Mitigation

3/ Correct measurement Noise

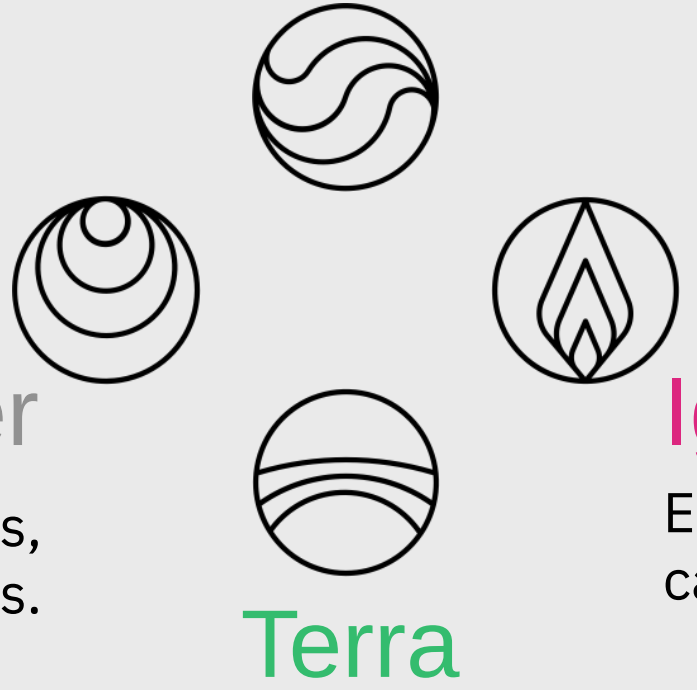




Mitigating Noise on Real Quantum Computers
https://www.youtube.com/watch?v=yuDxHJOKsVA

The elements:

Algorithms and applications for noisy, near-term quantum hardware.



Classical simulators, emulators, and debuggers.

Error characterization, validation, calibration, and mitigation.

Circuits, pulses, and optimizations for running on quantum devices.

Qiskit Aqua

Qiskit >=0.19.0

Qiskit Aqua contains a library of cross-domain quantum algorithms upon which applications for near-term quantum computing can be built

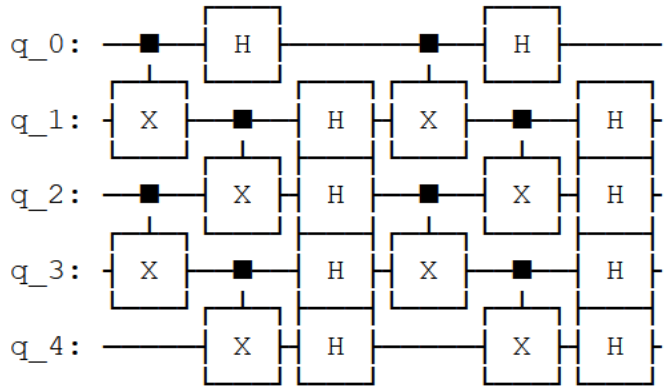
Library for Quantum Algorithms & Applications

```
from qiskit.aqua.algorithms import VQE,QAOA,Grover,HHL
```

Operator flow

- **Tools to make writing algorithms simple and easy.**
- Layer of modules between the circuits and algorithms, providing the language and computational primitives for Quantum algorithms research
- Work natively with `qiskit.circuit.library`

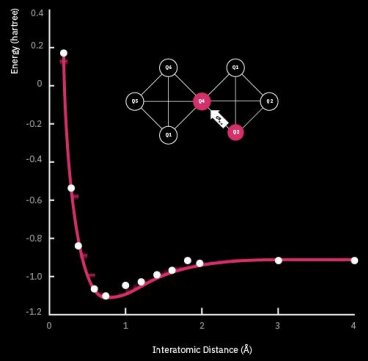
```
from qiskit.aqua.operators import X, Y, Z, I, CX, T, H
# We can build complicated Operators very easily
print(((H^5) @ ((CX^2)^I) @ (I^(CX^2)))**2)
```



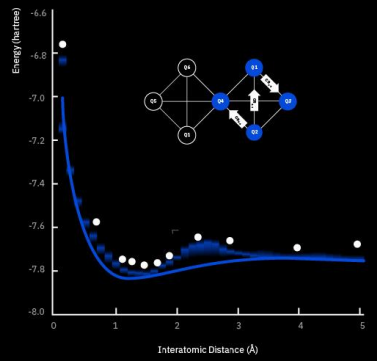
Defined use cases that address key industry imperatives will accelerate business investment decisions

	Chemicals and Petroleum	Distribution and Logistics	Financial Services	Health Care and Life Sciences	Manufacturing
Chemical Simulation	Chemical product design Surfactants, Catalysts			Drug Discovery Protein Structure Predictions	Materials Discovery Quantum Chemistry
Scenario Simulation		Disruption Management	Derivatives Pricing Investment Risk Analysis	Disease Risk Predictions	
Optimization	Feedstock To Product Oil Shipping / Trucking Refining Processes	Distribution Supply Chain Network Optimization Vehicle Routing	Portfolio Management Transaction Settlement	Medical/Drug Supply Chain	Fabrication Optimization Manufacturing Supply Chain Process Planning
AI/ML	Drilling Locations Seismic imaging	Consumer Offer Recommender Freight Forecasting Irregular Behaviors (ops)	Finance Offer Recommender Credit/Asset Scoring Irregular Behaviors (fraud)	Accelerated Diagnosis Genomic Analysis Clinical Trial Enhancements	Quality Control Structural Design & Fluid Dynamics

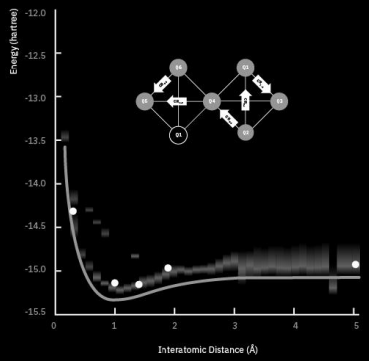
Hydrogen (H_2): 2 Qubits



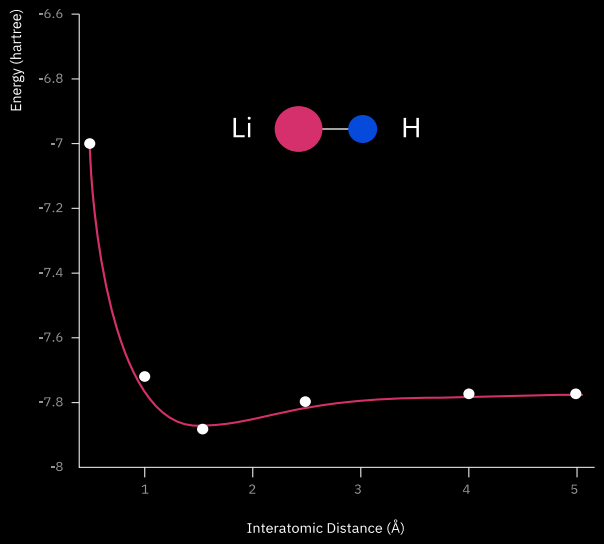
Lithium Hybride (LiH): 4 Qubits



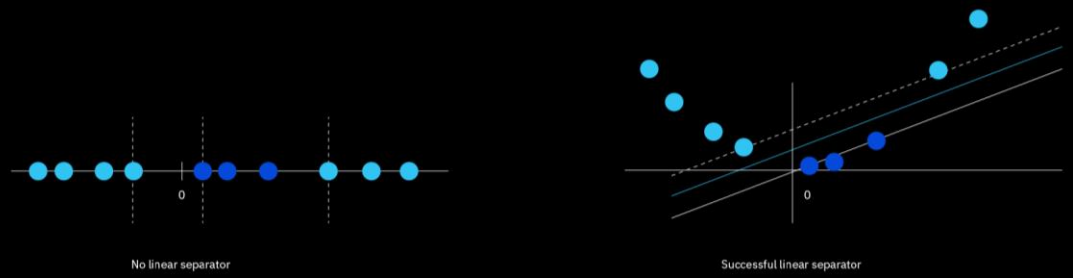
Beryllium hydride (BeH_2): 8 Qubits



IBM 2018

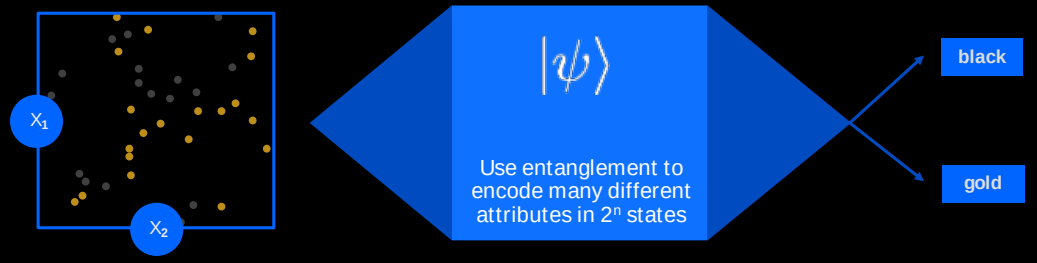


SVM



Quantum SVM

Quantum shallow depth circuits as classifiers

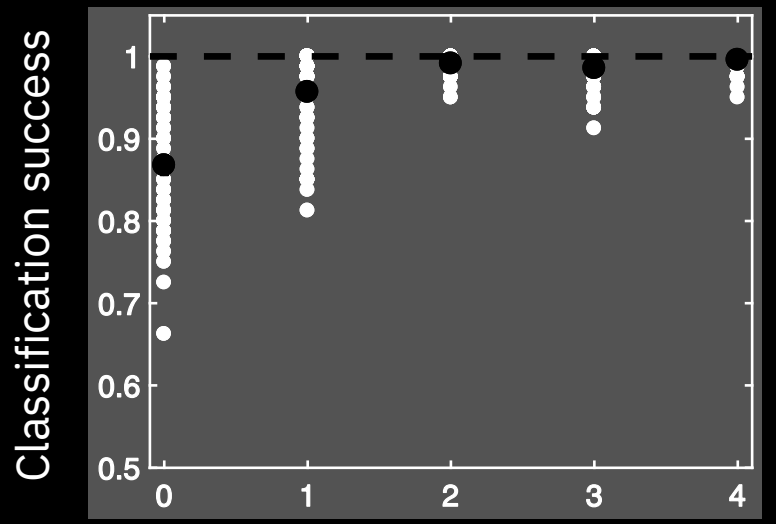


Function we are learning

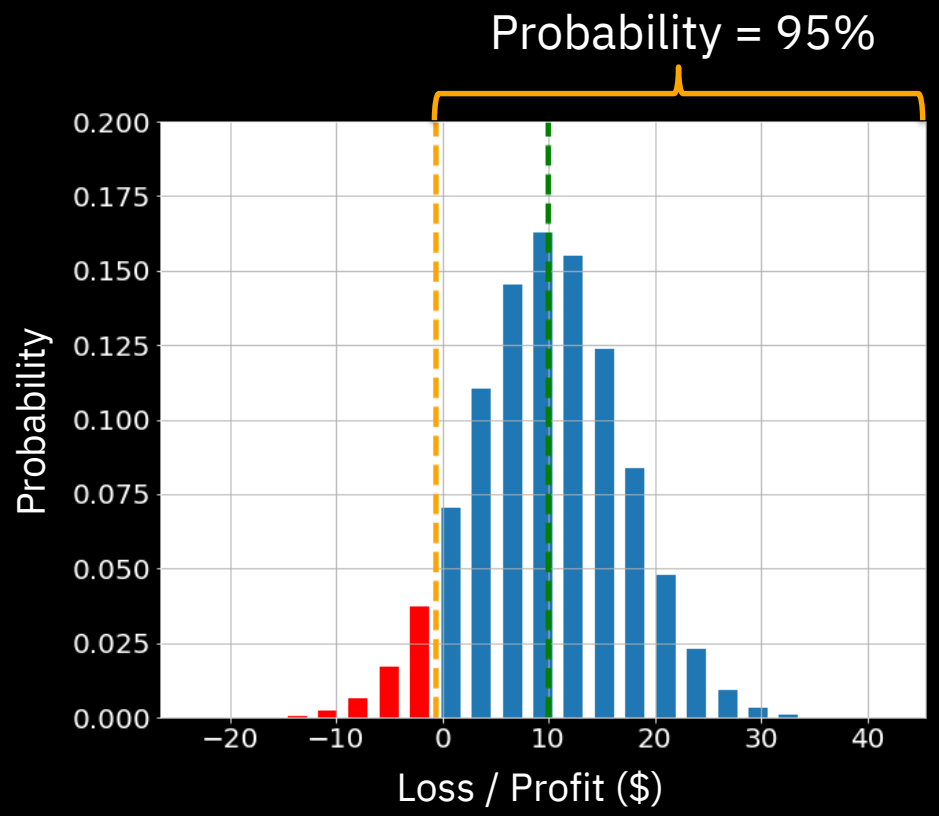
Generate a quantum feature map

Process the quantum data

Experimental results



Quantum Finance



Goal:
Estimate
- **Expected Value**
- **Value at Risk (e.g. 95%)**
- **Conditional Value at Risk (e.g. 95%)**

npj | Quantum Information

ARTICLE OPEN

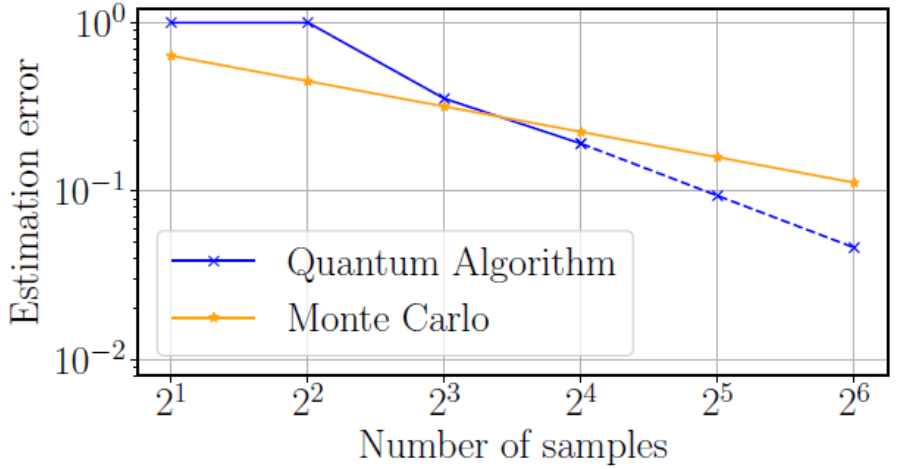
Quantum risk analysis

Stefan Woerner[1] and Daniel J. Egger[1]

www.nature.com/npjqi

Quantum algorithm for **pricing & risk analysis**

Quadratic speed-up compared to Monte Carlo:



Demonstrated on **real quantum device**

Qiskit Aqua

Aqua currently allows the user to experiment on chemistry, AI, optimization and finance applications for near-term quantum computers.

Chemistry

Ground State
Energy
Dipole Moment
Excited States

Finance

Portfolio
Optimization
Risk Analysis
Pricing

ML

Training
Classification
SVM
QGAN

Optimization

3SAT
MaxCut
TSP
Graph Partition
Stableset
Clique
Exact Cover
Set Packing
Vertex Cover

Quantum Algorithms

“VQE” “QAOA” “Dynamics” “QPE/IQPE” “Amplitude Estimation”
“Grover” “SVM Q Kernel” “SVM Variational” “Simon” “Deutsch-Josza”
“Bernstein-Varizani” “HHL”

Qiskit Aqua - Modules

Qiskit >=0.19.0

Aqua includes domain application support for:

- Chemistry (**refactor ongoing**)
- Finance (in preparation)
- Machine Learning (in preparation)
- Optimization (**released May 2020**)
- Physics (in preparation)

The `qiskit.optimization` package covers the whole range from high-level modeling of optimization problems (compatible with docplex / CPLEX models), with automatic conversion of problems to different required representations, to a suite of easy-to-use quantum optimization algorithms that are ready to run on classical simulators, as well as on real quantum devices via Qiskit.

```
from qiskit.optimization.algorithms import grover_optimizer
```

https://github.com/Qiskit/qiskit-tutorials/tree/master/tutorials/optimization

Community, Ressources

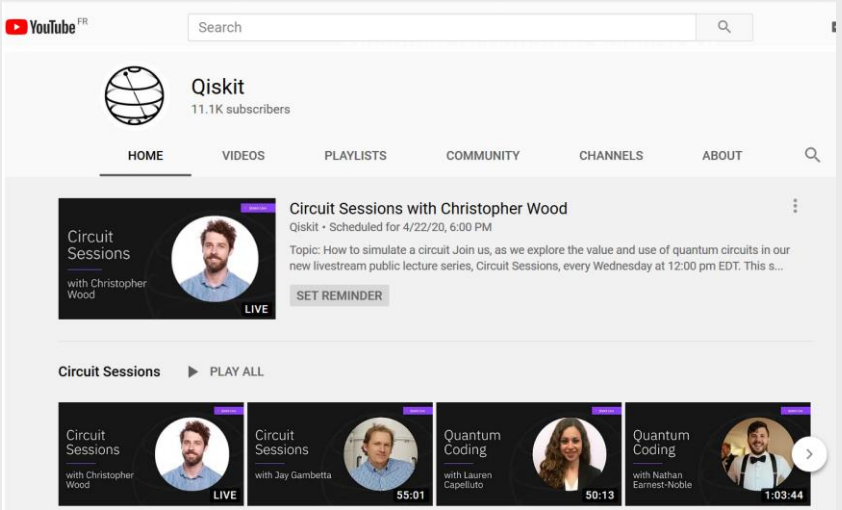
qiskit.org

github.com/Qiskit

qiskit.slack.com

New live series are available

https://www.youtube.com/Qiskit



Tutorials:

Qiskit Tutorial
https://github.com/Qiskit/qiskit-tutorials/tree/master/tutorials

Legacy
https://github.com/Qiskit/qiskit-tutorials/tree/master/legacy_tutorials
https://github.com/qiskit-community/qiskit-community-tutorials

Qiskit Pulse Module

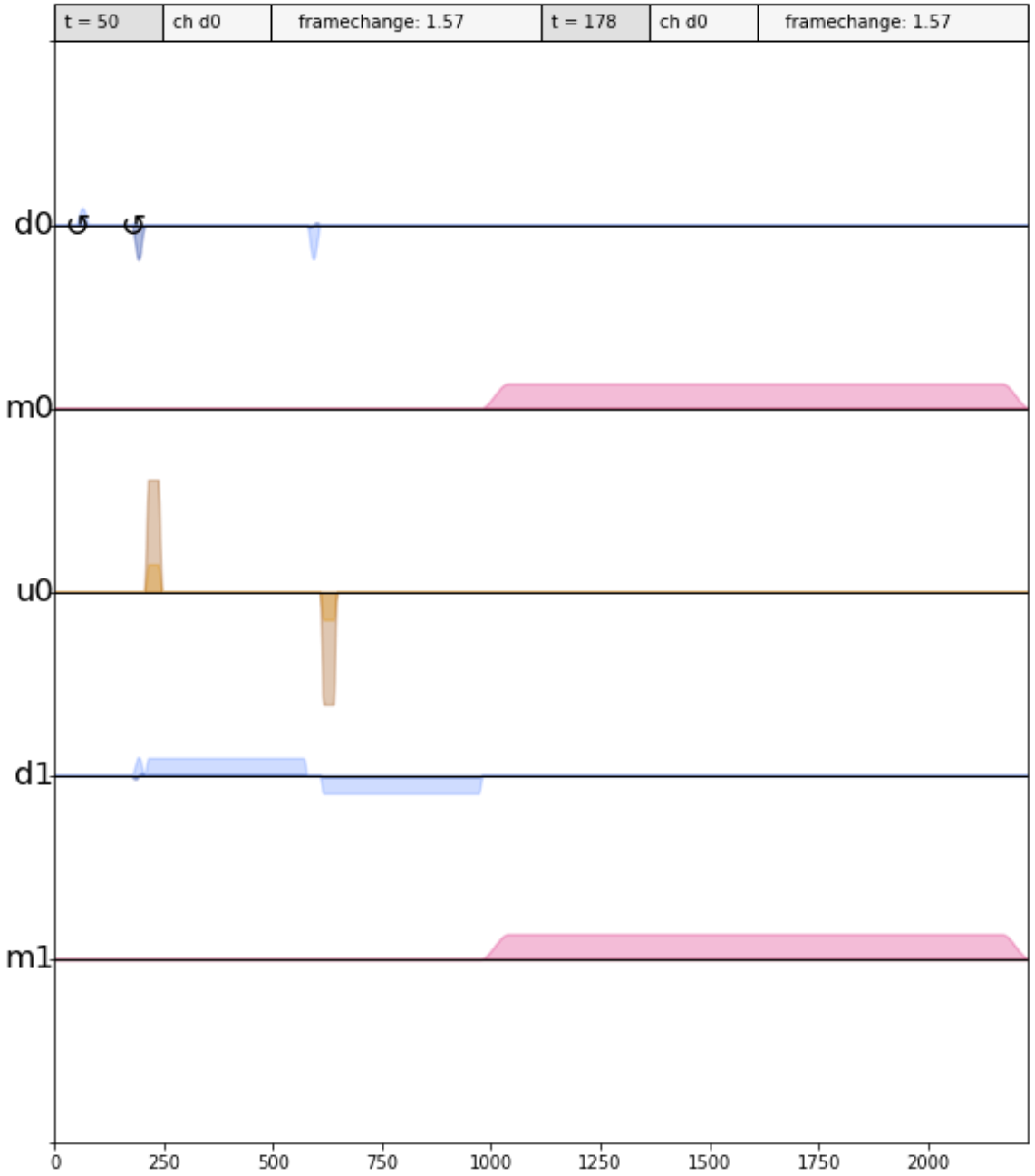
Experiment hardware is abstracted as set of channels on which pulse instructions are scheduled:

Pulse channels: Pulses are played on microwave generators. These manipulate and measure qubits.

- d: Channels are individual qubit drive channels.
- m: Channels are individual qubit measurement stimulus channels
- u: Channels are arbitrary stimulus lines. For example these may be used for two-body terms.

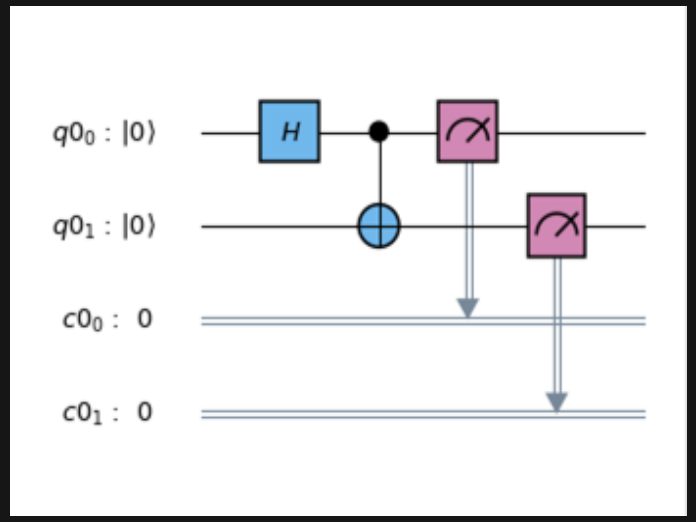
Acquire channels: Acquisitions are scheduled on digitizers.

Memoryslots: Measurement results are stored in memory slots



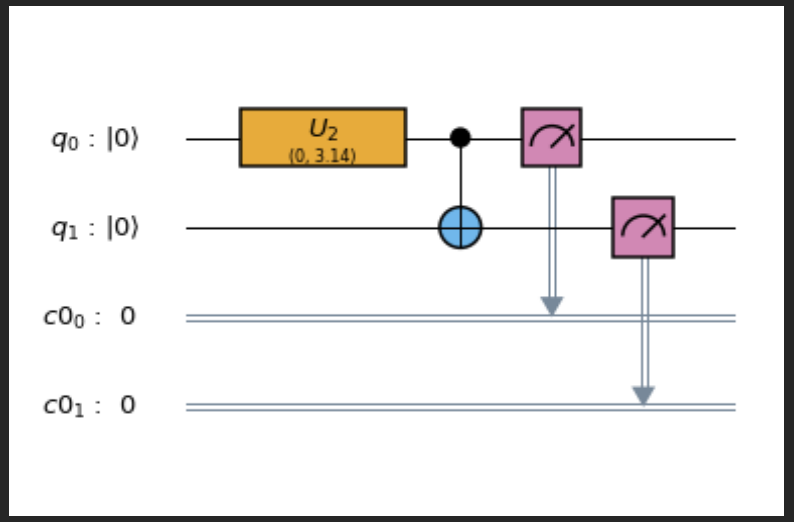
Circuits

```
qr = QuantumRegister(2)
cr = ClassicalRegister(2)
qc = QuantumCircuit(qr, cr)

qc.h(qr[0])
qc.cx(qr[0], qr[1])
qc.measure(qr, cr)
qc.draw(output='mpl')
```



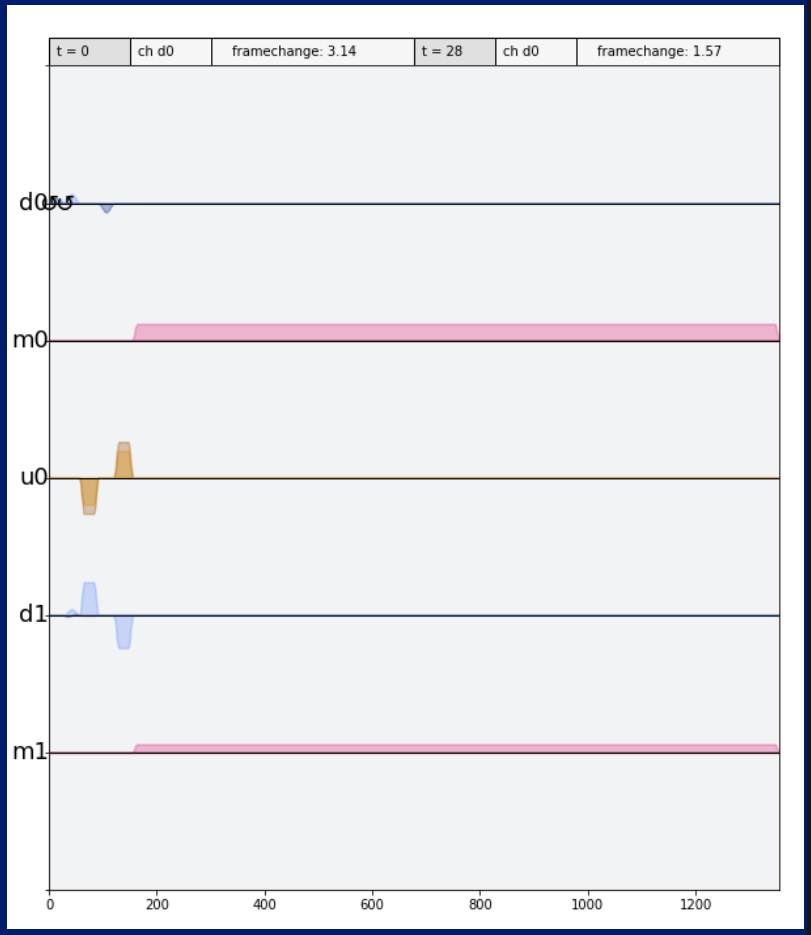
Circuits to low level

```
qc_device = transpile(qc, backend)
qc.draw(output='mpl')
```



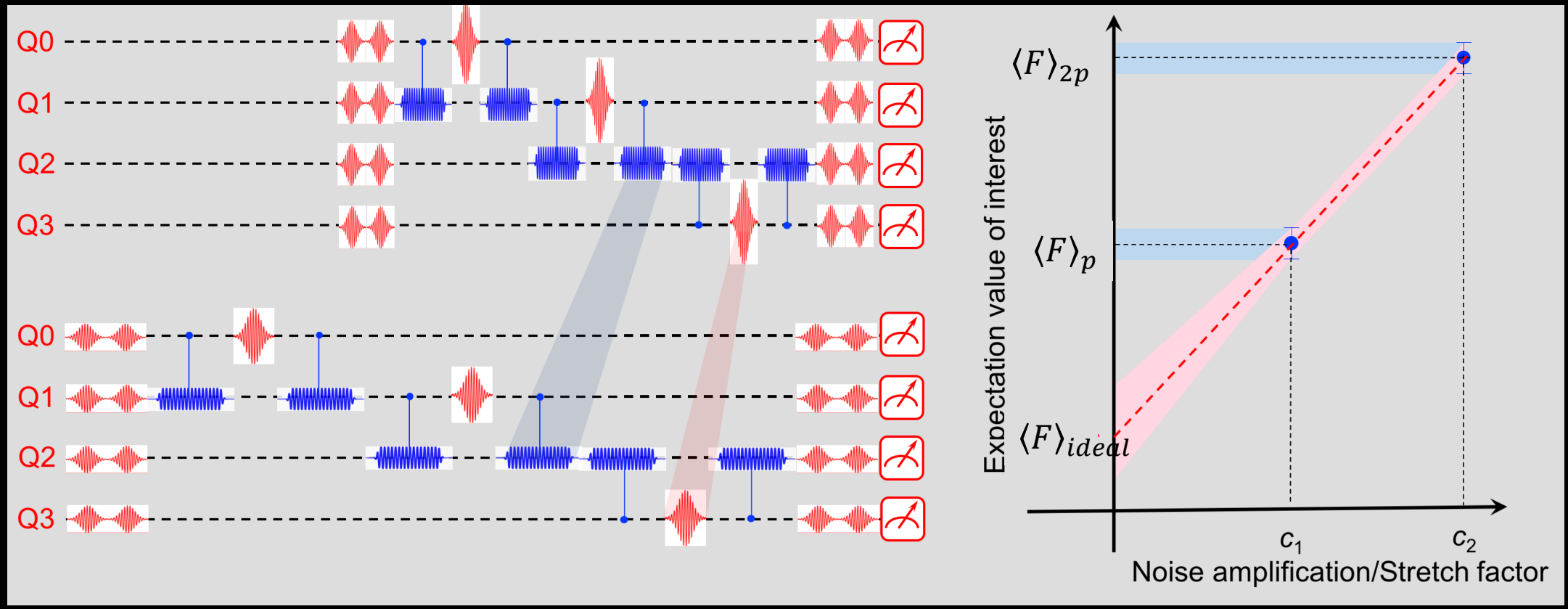
Schedules

```
sched_circ = schedule(qc_device, backend)
sched_circ.draw()
```

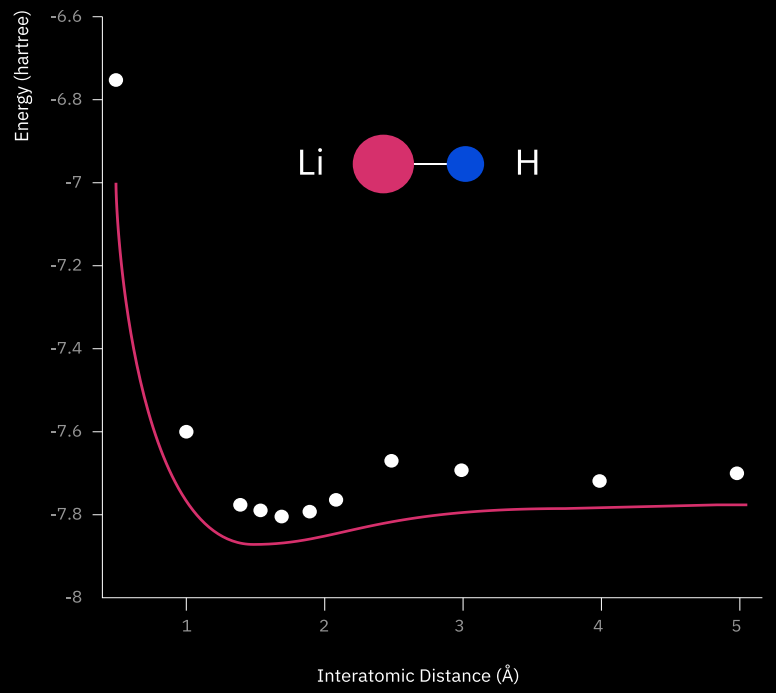


Error mitigation with pulse

Amplifying error rate is equivalent to to rescaling classical control dynamics
(under reasonable assumptions about the nature of noise)



IBM 2017



IBM 2018