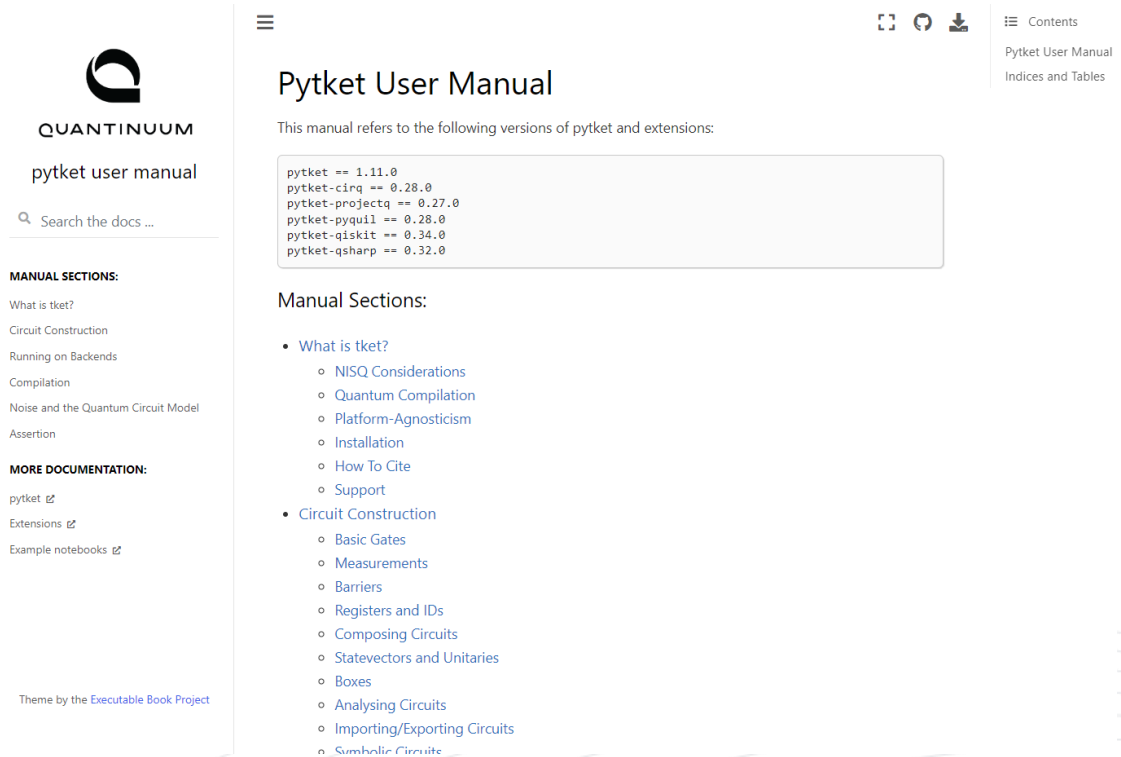
PyTKET

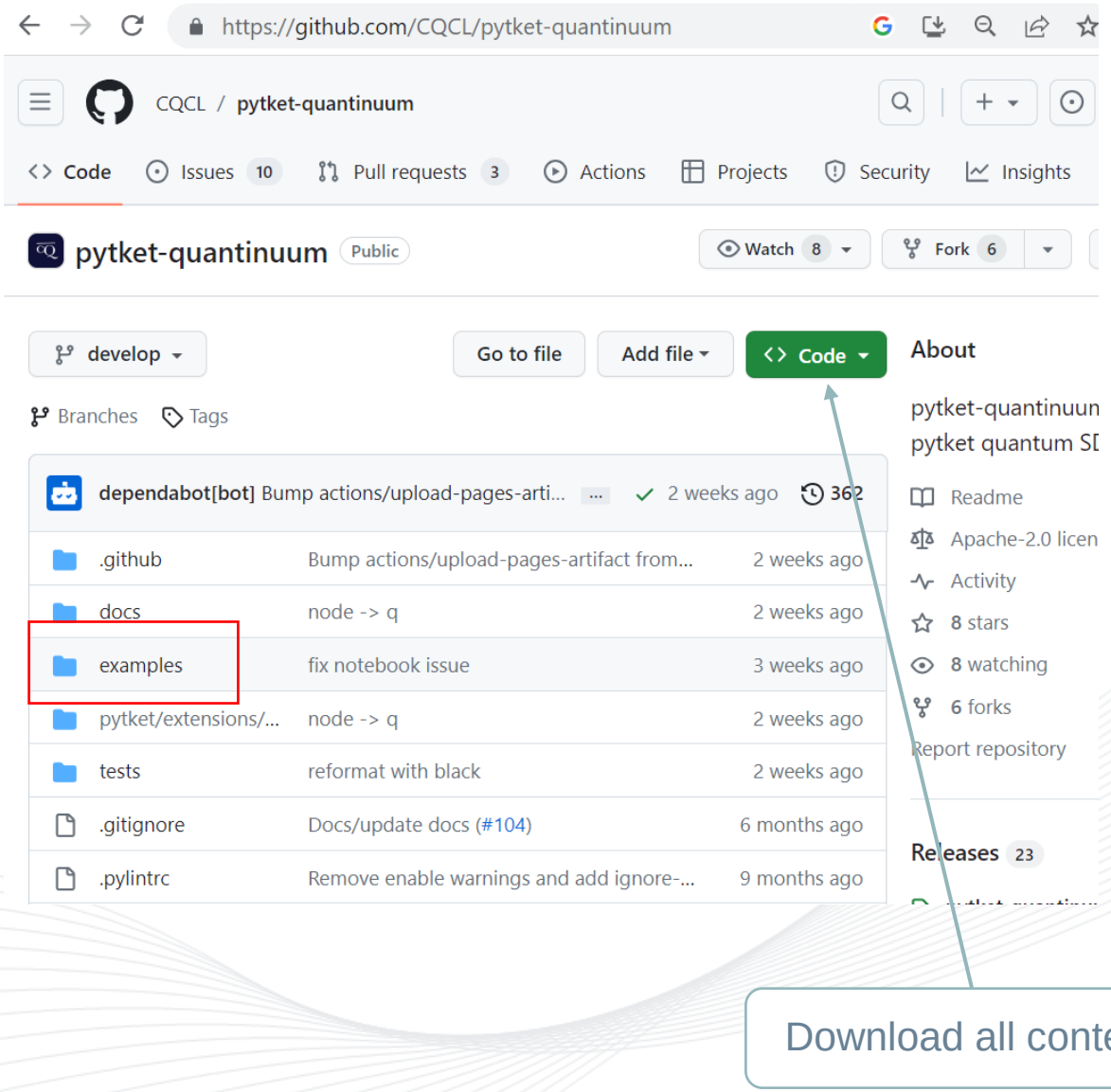


- PyTKET User Manual: https://cqcl.github.io/pytket/manual/index.html
- PyTKET API: https://cqcl.github.io/tket/pytket/api/index.html
- Github: https://github.com/CQCL/pytket-extensions
- Slack: https://tketusers.slack.com

Getting Started today

- Github: https://um.qapi.quantinuum.com
- Download the set of examples
- Set up Python Environment on your computer following the instructions (you can use qBraid)
- Open new jupyter notebook in this folder

```
pip install pytket
pip install pytket-quantinuum
```



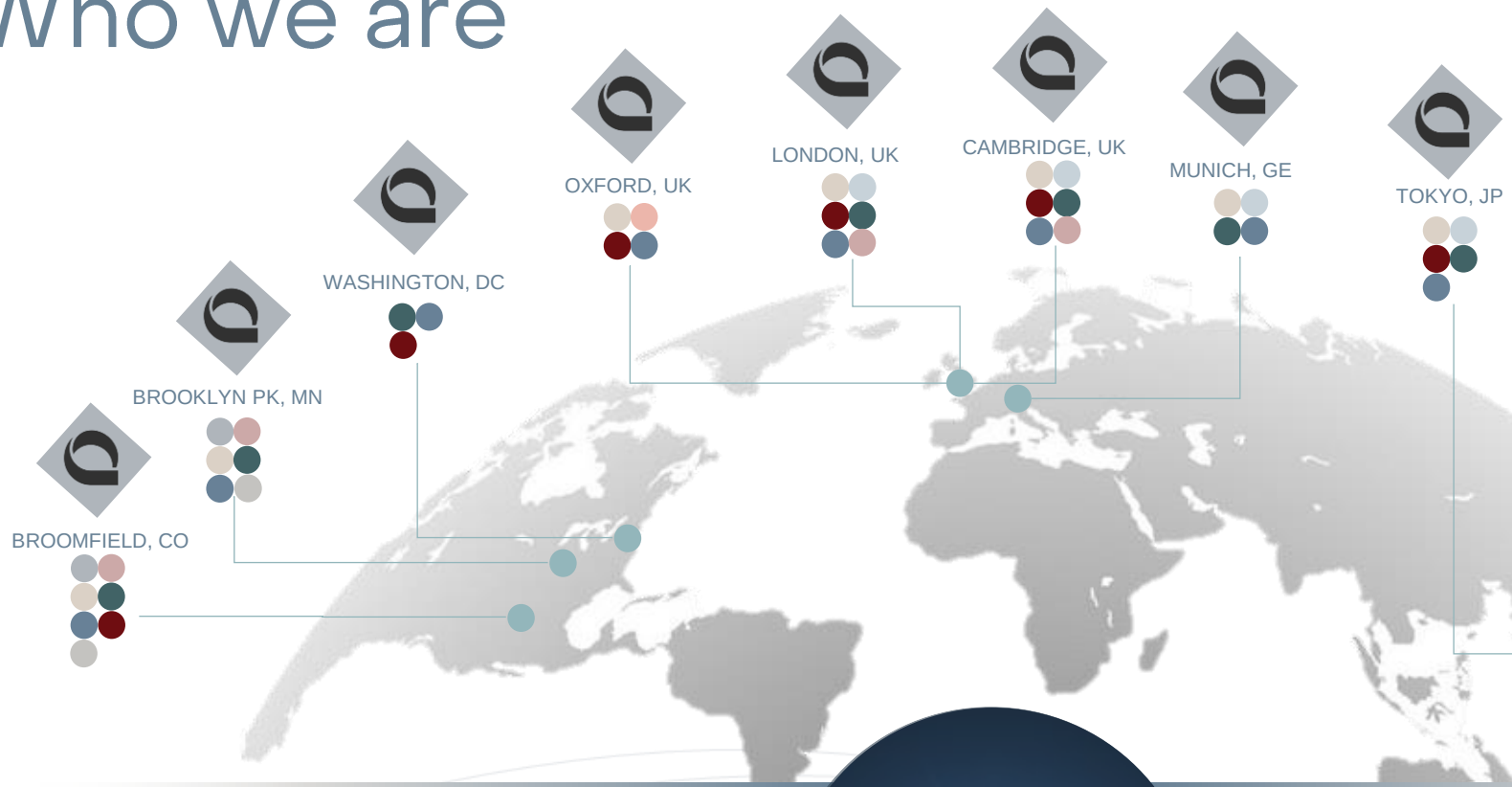
Improving quantum circuit performance with TKET on H-Series devices and other platforms

Presented by

Kathrin Spendier (Quantum Evangelist)
kathrin.spendier@quantinuum.com

Connect With Me!

Who we are



Our people are clustered by specialty and science domain

Ion trapping
Engineering
Quantum algorithms
Quantum foundations
Quantum chemistry
Quantum cryptography
Integrated Supply Chain
HR, finance, IT, legal
Biz dev, comms

Cambridge Quantum

Leader in Quantum Computing Software

Honeywell Quantum Solutions

Leading Quantum Computing Hardware

An integrated approach

Industrial Collaborators

Telcom, Finance, Pharma, Automotive, Manufacturing, Transport, Chemicals....

Cybersecurity

Quantum Origin: quantum computing-enabled cryptographic keys

Quantum Chemistry

InQuanto: State-of-the-art chemistry platform for quantum computers

AI & ML

Including open-source QNLP Toolkit and Library '**LAMBEQ**'

TKET

Open-source quantum software development platform

Third Party Platforms

Quantum Hardware, Simulators and Cloud Providers

H-Series

Quantum Computers

Powered by Honeywell

Quantinuum Products

Partner / Ecosystem Products

Agenda for this TKET workshop

1) Introduction to TKET and Quantinuum H-series unique features

2) Go over some pytket-quantinuum coding examples

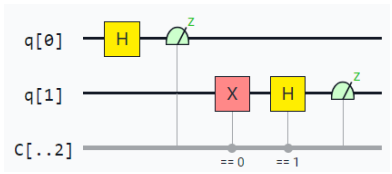
3) Finish with some pytket-qiskit examples if there is time

QUANTUM COMPUTING HARDWARE

TYPICAL QUANTUM ALGORITHM WORKFLOW ON A GATE-MODEL QUANTUM COMPUTER

e.g. Traveling Salesman

```
circuit = QuantumCircuit(q,c)
normal = NormalDistribution(num_target_qubits = 5, mu=0, sigma=1,
low=-1, high=1)
normal.build(circuit,q)
circuit.measure(q,c)

job = execute(circuit, backend, shots=8192)
job_monitor(job)
counts = job.result().get_counts()

print(counts)
sortedcounts = []
sortedkeys = sorted(counts)

for i in sortedkeys:
    for j in counts:
        if(i == j):
            sortedcounts.append(counts.get(j))
```



e.g. TKET

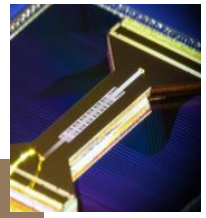
Problem Definition

Quantum Algorithm

Quantum Circuit

Quantum Compiler

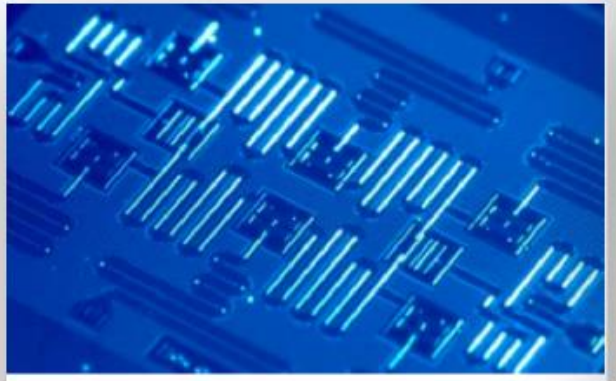
Quantum Processor



Quantum Simulator

Full-Stack

TYPES OF QUANTUM HARDWARE

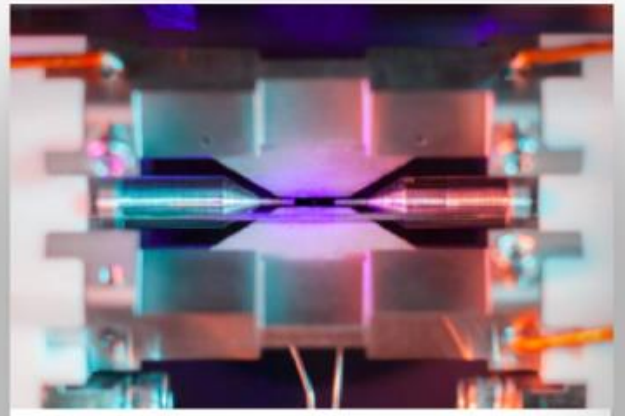


Superconducting

IBM

Rigetti

Google

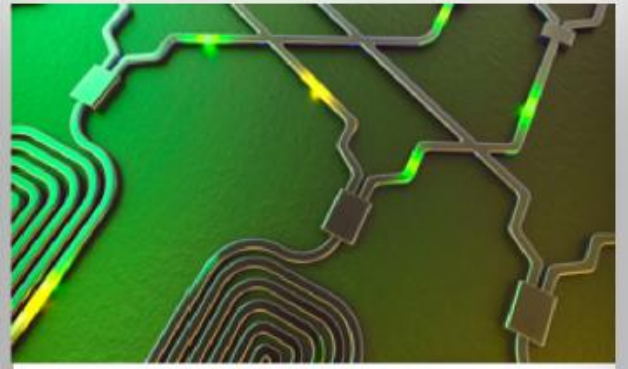


Ion Trap

IonQ

Quantinuum

AQT



Photonic

Xanadu

PsiQuantum

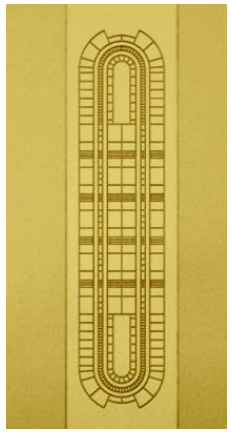
Other technologies

UNSW Sydney—atom spin quantum computer

Archer Materials – room temperature qubits

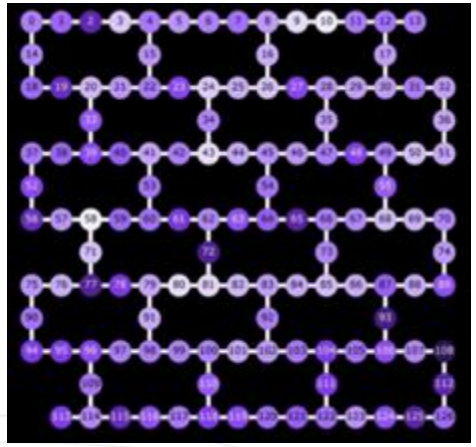
Quantum hardware architectures

H-2



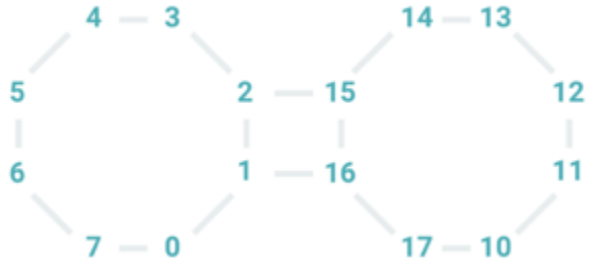
Basis Gates:
$U1q(\theta,\varphi)$, R_Z, ZZ, RZZ

EAGLE



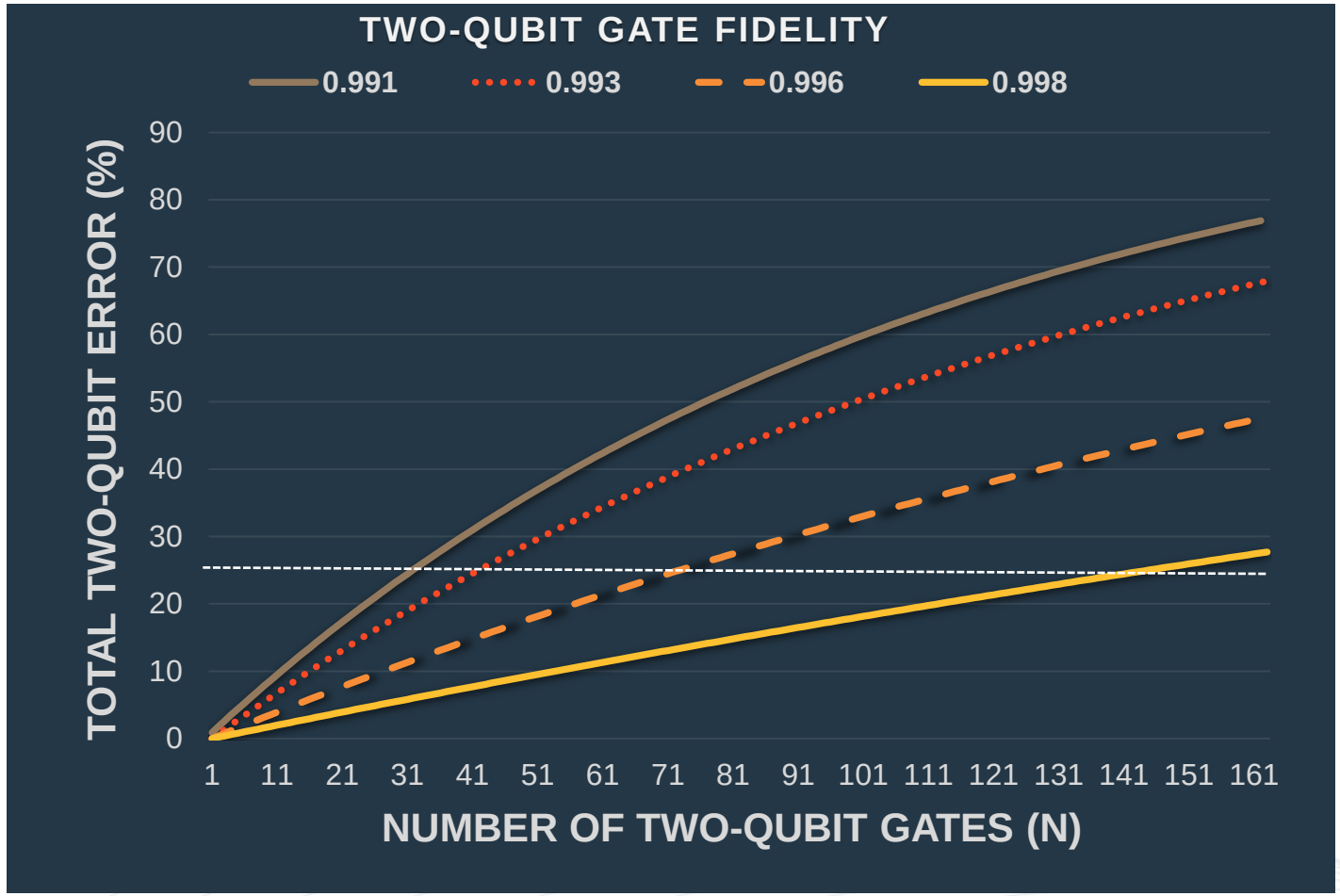
Basis Gates:
ECR, ID, R_Z, S_X, X

ASPEN



Basis Gates:
CZ, R_X, R_Z

ESTIMATING TWO-QUBIT GATE ERRORS



$$\%\, Error_{2QG} = \left(1 - Fidelity^{N}\right)(100\%)$$

Examples:

100 2QGs with a fidelity of 0.993
$(1 - 0.993^{100})\ (100\%) = 50\%$

100 2QGs with a fidelity of 0.998
$(1 - 0.998^{100})\ (100\%) = 18\%$

100 2QGs with a fidelity of 0.999
$(1 - 0.999^{100})\ (100\%) = 9.5\%$

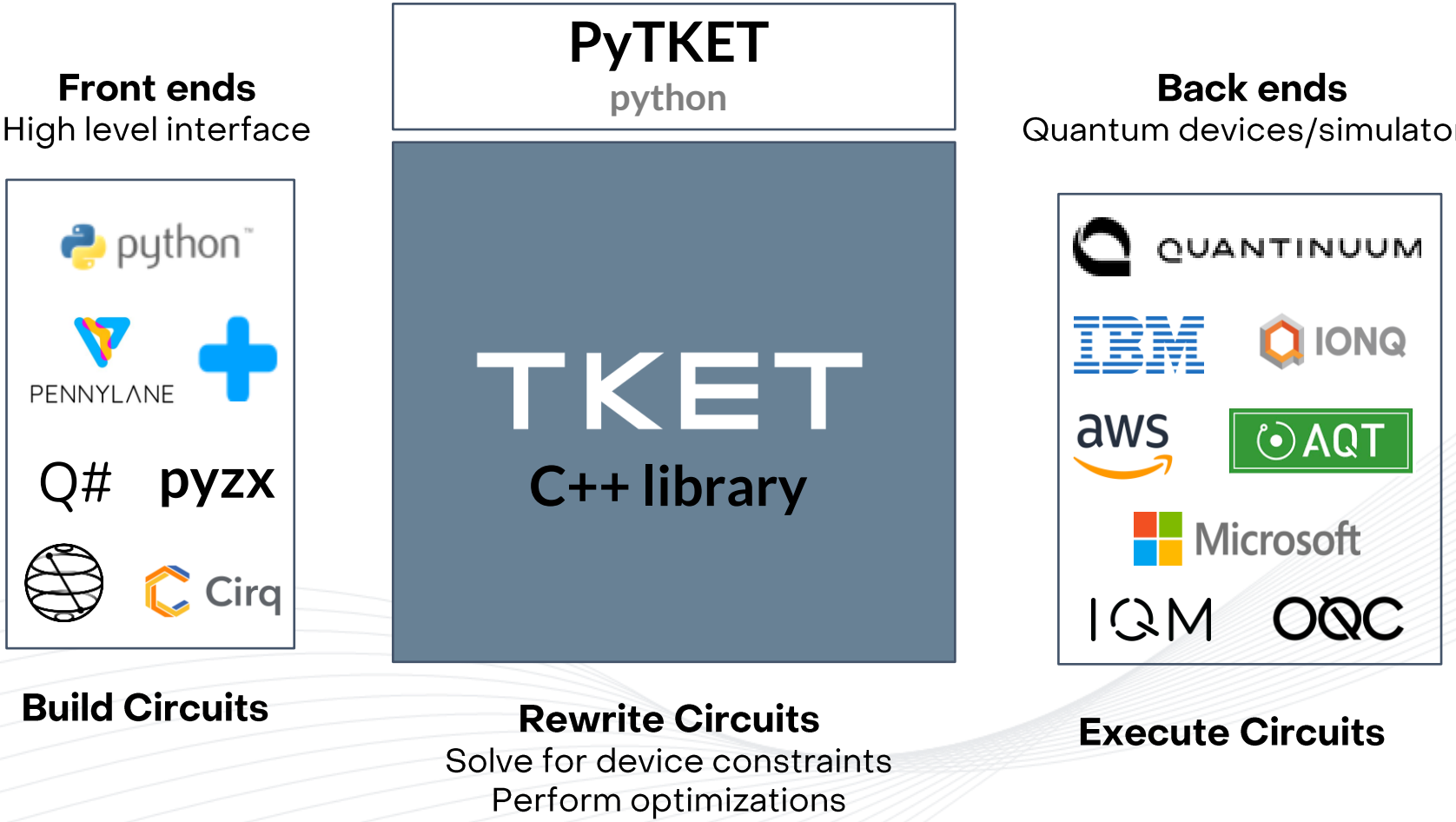
QUANTUM COMPUTING SOFTWARE

Quantum Information Scientist (End User)

- Use gate-level quantum computers to solve my problems
- Program a quantum computer in my favorite language
- Run my quantum algorithms efficiently
- Get the most accurate result while focusing on solving my problems

TKET as a universal SDK

TKET optimizes quantum circuits, reducing the number of required operations – essential for NISQ devices.



TKET EXTENSIONS

Device & Simulators

- pytket-quantinuum
- pytket-qiskit (IBM)
- pytket-ionq
- pytket-aqt
- pytket-braket (AWS)
- pytket-qsharp (Azure)
- pytket-pyquil (Rigetti)
- pytket-iqm

Simulators

- pytket-qujax
- pytket-project
- pytket-pysimplex
- pytket-qulacs
- pytket-stim
- pytket-cutensornet*

* Under development

Transpilers

- pytket-pennylane
- pytket-pyzx
- pytket-cirq
- pytket-qir*

* Under development

Compilation tasks

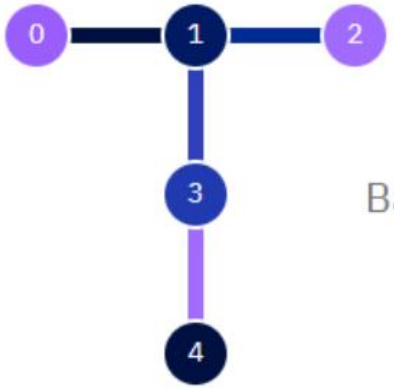
SOLVE CONSTRAINTS

Alter the form to meet the restrictions of the backend

+

CIRCUIT OPTIMIZATION

Preserve semantics and make smaller



IBMQ QUITO

Basis gates: CX, ID, RZ, SX, X

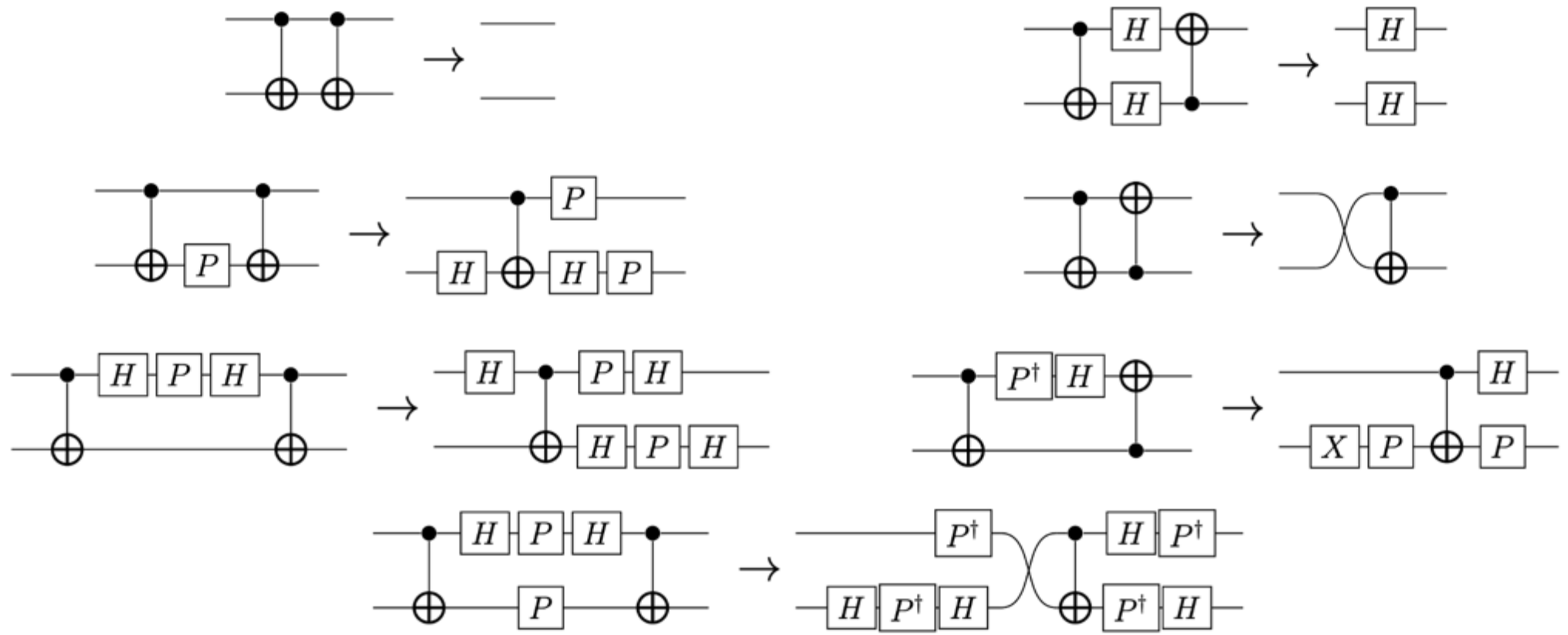
REBASE

```
1  from pytket.passes import auto_rebase_pass
2
3  ibm_rebase = auto_rebase_pass({OpType.X,OpType.SX,OpType.Rz,OpType.CX})
4  ibm_rebase.apply(circ);
```

PLACEMENT

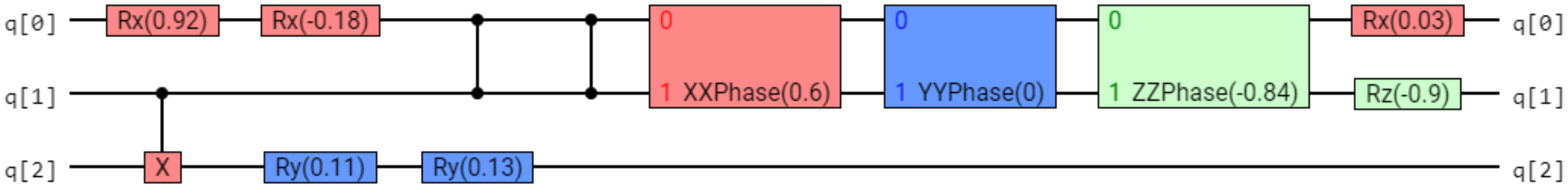
```
1  from pytket.passes import PlacementPass
2  from pytket.placement import GraphPlacement
3  place = PlacementPass(GraphPlacement(backend.backend_info.architecture))
4  place.apply(circ)
```

Example circuit optimization

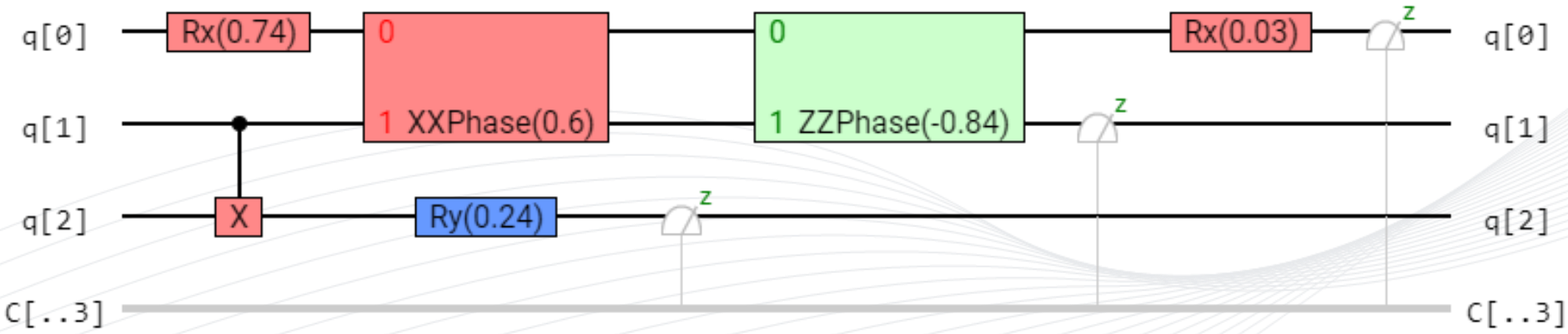


For more technical detail see arXiv:2003.10611

OPTIMIZATION: REMOVE REDUNDANCIES



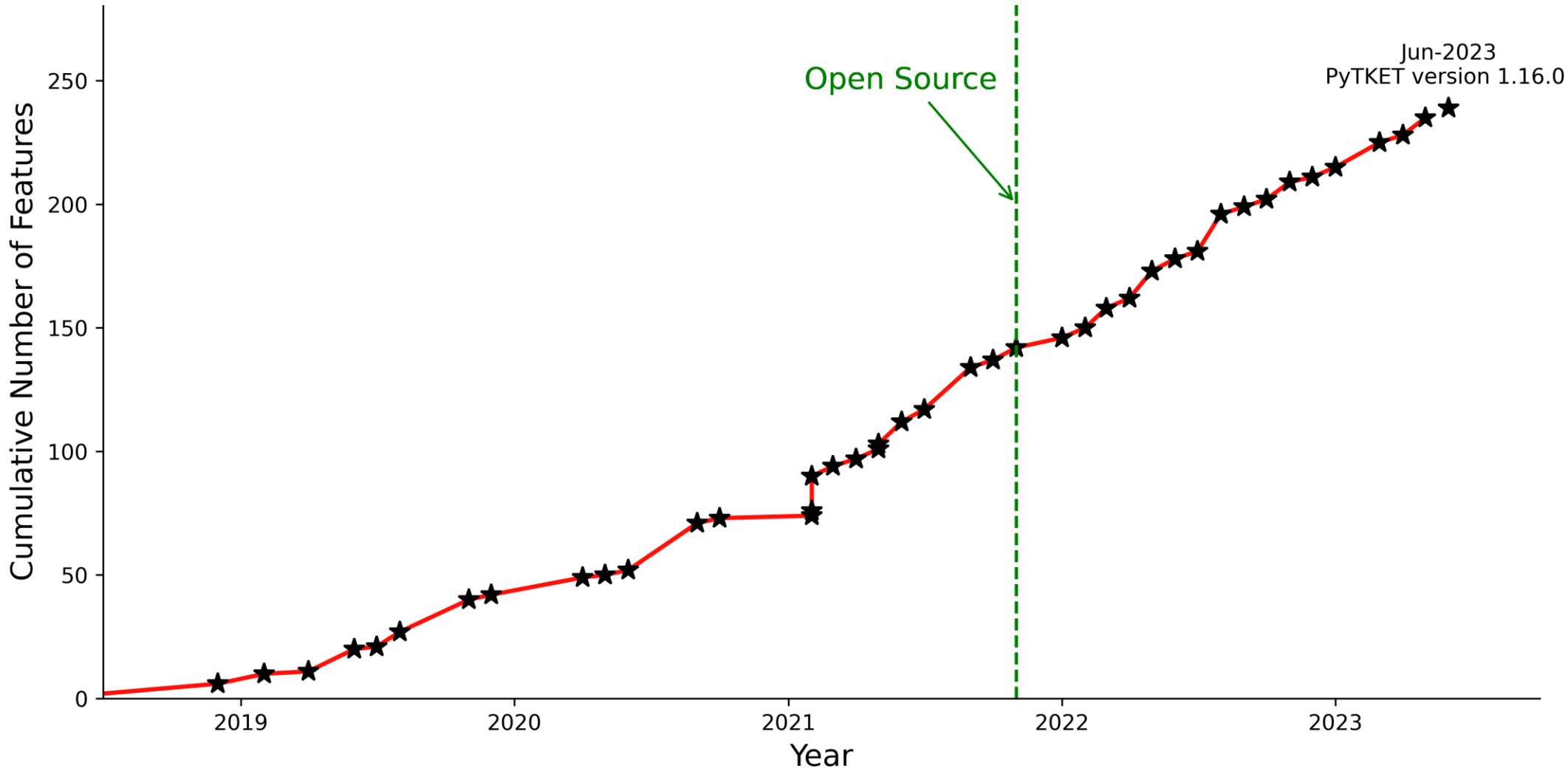

```
from pytket.passes import RemoveRedundancies
RemoveRedundancies().apply(circuit.measure_all())
```



TKET-SPECIFIC COMPILER PASSES

- Synthesize many qubit operations to 1-, 2-qubit gates
- Local graph rewrites, Pattern-replacement
- Resynthesize sub-circuits via special representations
  - ZX-terms, Clifford tableaux, Phase-polynomial / Phase gadget
- Architecture-aware synthesis
- Mapping to chosen gate basis
- Mapping and routing circuits to fixed architectures
- Symbolic expression optimization
- …. more!

PyTKET's growing number of features



TKET has a default pass manager for each backend

```
get_compiled_circuit(circuit,optimization_level)
```

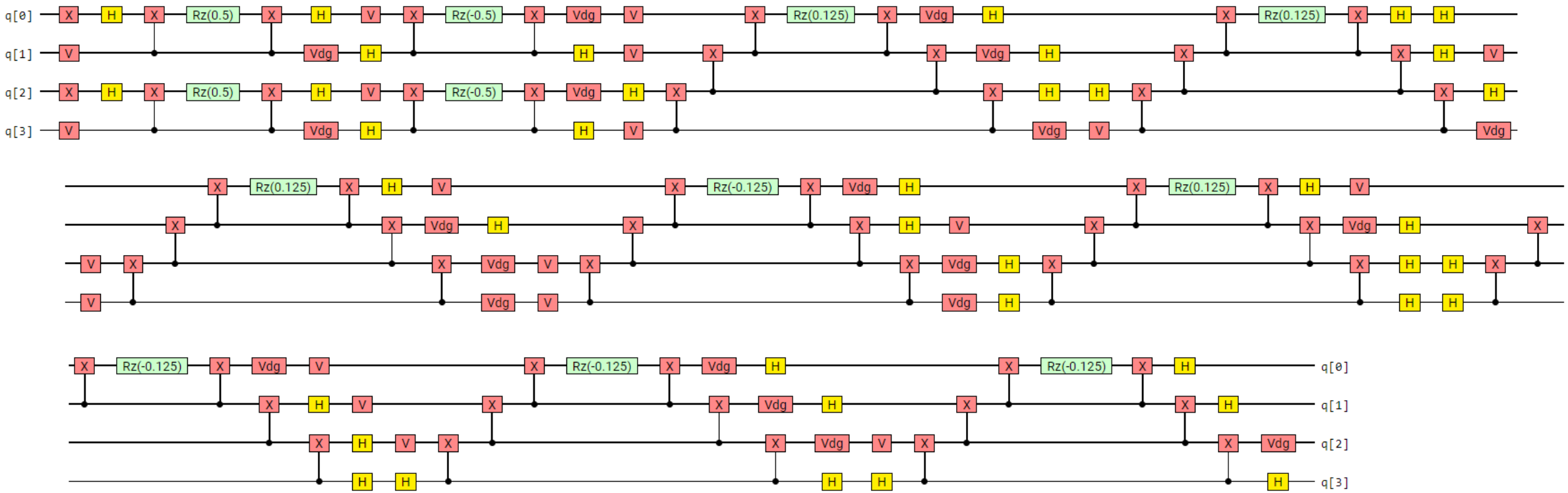
Level 0	Level 1	Level 2 (default)
Solves the device constraints without optimizing.	Additionally performs some light optimizations.	Adds more intensive optimizations that can increase compilation time for large circuits.

TARGET A DIFFERENT BACKEND EASILY

```python
from pytket.extensions.quantinuum import QuantinuumBackend
# Select the H1-2 emulation device
machine = 'H1-2E'
backend_emu = QuantinuumBackend(device_name=machine)
backend_emu.login()
```

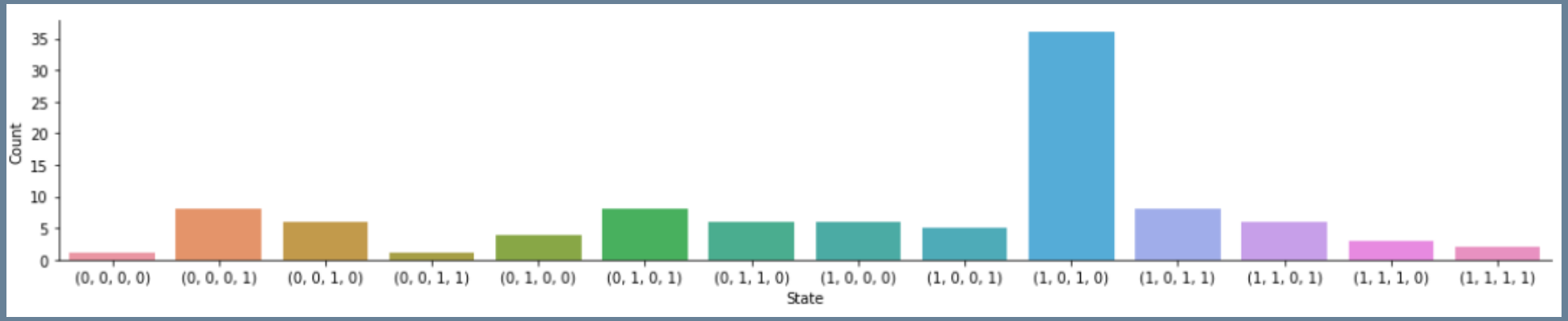
```python
from pytket.extensions.qiskit import IBMQBackend
from qiskit_ibm_provider import IBMProvider

my_instance=f"{hub}/{group}/{project}"
ibm_provider = IBMProvider(instance=my_instance)
backend = IBMQBackend("ibmq_belem") # Initialise backend for an IBM device
```

RUNNING QUANTUM CIRCUITS WITH DEFAULT PASSES

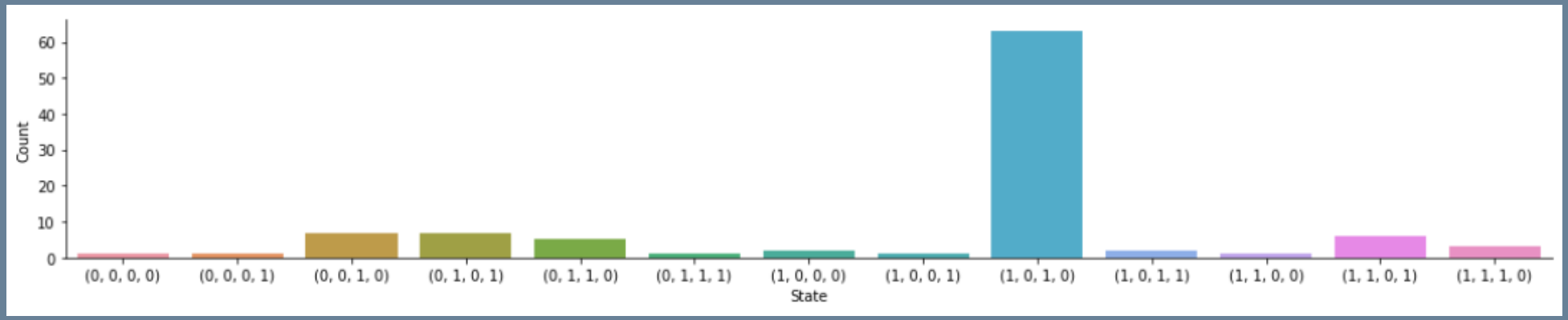


EXAMPLE RESULTS: IBMQ QUITO

Before optimization (routing and placement only, 56 two-qubit gates)



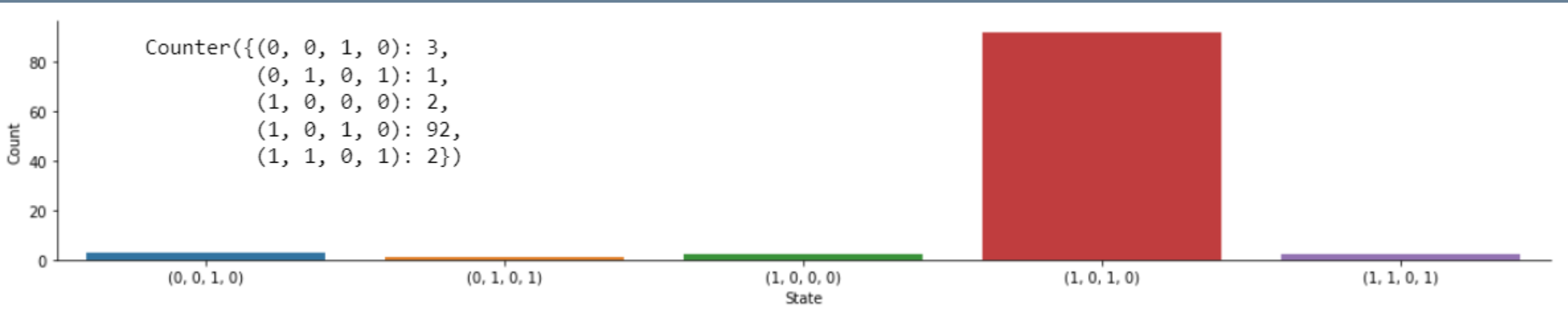
After optimizing at level 2 (19 two-qubit gates)



EXAMPLE RESULTS: H1 SYSTEM EMULATOR

Avg. single-qubit gate fidelity	**99.99(1)%**
Avg. two-qubit gate fidelity	**99.72(6)%**
Measurement fidelity	**99.70(5)%**

After optimizing at level 2 (18 two-qubit gates)

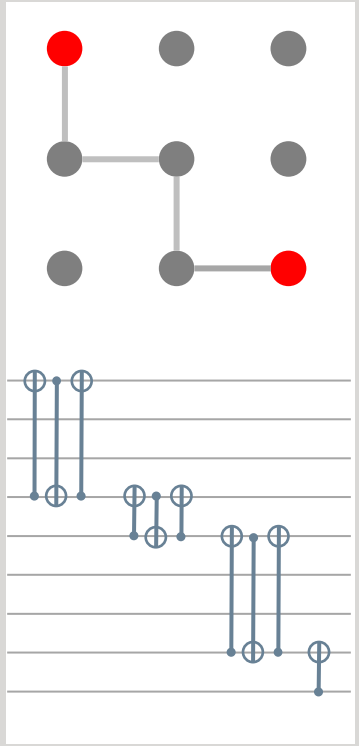


QUANTINUUM H-SERIES UNIQUE FEATURES

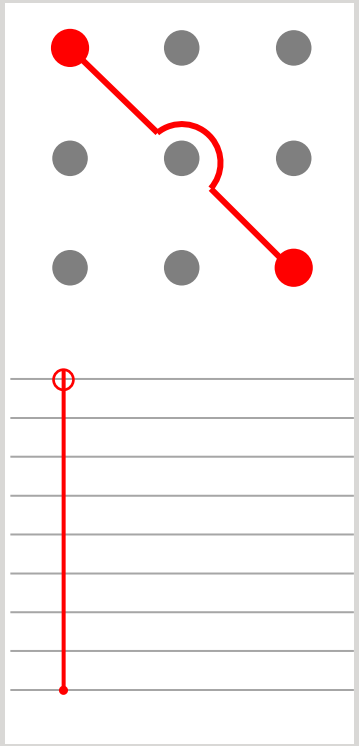
QCCD ARCHITECTURE DIFFERENTIATING FEATURES

All-to-All Connectivity

Nearest Neighbor



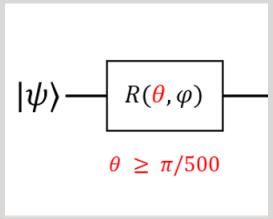
All-to-All



High-Fidelity Gates

1Q fidelity	> 99.996%
2Q fidelity	> 99.8%
State preparation and measurement	> 99.7%
Measurement cross-talk error	< 0.005%
Memory error per qubit at average depth-1 circuit	< 0.06%

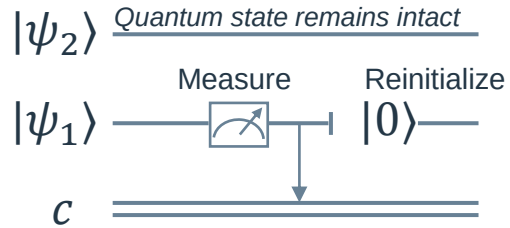
Arbitrary Angle 1-qubit gates and 2-qubit gates



$$RZZ(\theta) = e^{-\frac{i\theta}{2}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{i\theta} & 0 & 0 \\ 0 & 0 & e^{i\theta} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

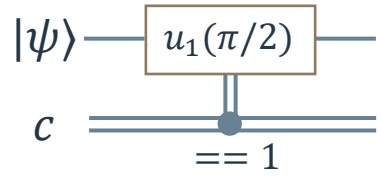
Qubit Measurement and Reuse

Measurement and reuse



Conditional logic

If c==1, perform gate
If c==0, do not



ARBITRARY ANGLE TWO-QUBIT GATE

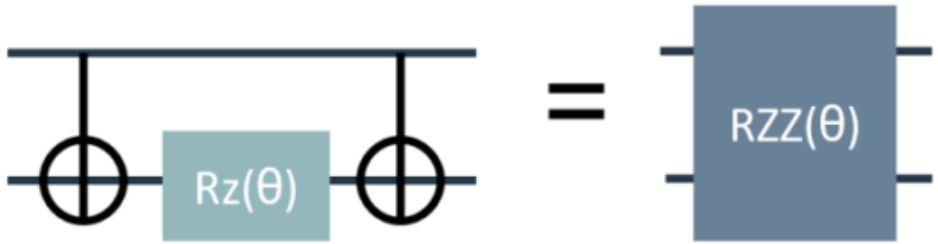
Fully entangling two-qubit gate

$$ZZ() = e^{-i\frac{\pi}{4}\hat{Z}\otimes\hat{Z}} = e^{-\frac{i\pi}{4}}\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & i & 0 & 0\\ 0 & 0 & i & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

Arbitrary angle two-qubit gate

$$RZZ(\theta) = e^{-i\frac{\theta}{2}\hat{Z}\otimes\hat{Z}} = e^{-\frac{i\theta}{2}}\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & e^{i\theta} & 0 & 0\\ 0 & 0 & e^{i\theta} & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

Quantum circuits that use the gate sequence CNOT, RZ, CNOT can be replaced with the arbitrary angle ZZ gate. Enables a lower number of two-qubit gates in a quantum circuit, improving performance by decreasing gate errors.



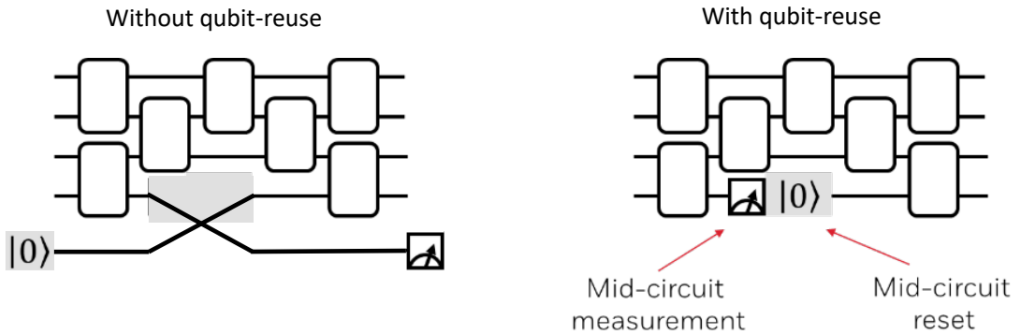
Byron Drury and Peter Love, Constructive quantum Shannon decomposition from Cartan involutions. Journal of Physics A: Mathematical and Theoretical, 41, 395305 (2008).
C. Baldwin, et al., Re-examining the quantum volume test: Ideal distributions, compiler optimizations, confidence intervals, and scalable resource estimations, Quantum 6, 707 (2022).

Mid-Circuit Measurement and Qubit Re-use + Conditional Logic

Examples:

- Quantum Teleportation
- Quantum Error Correction
- Feed Forward for quantum state preparation

Qubit-reuse



Example:

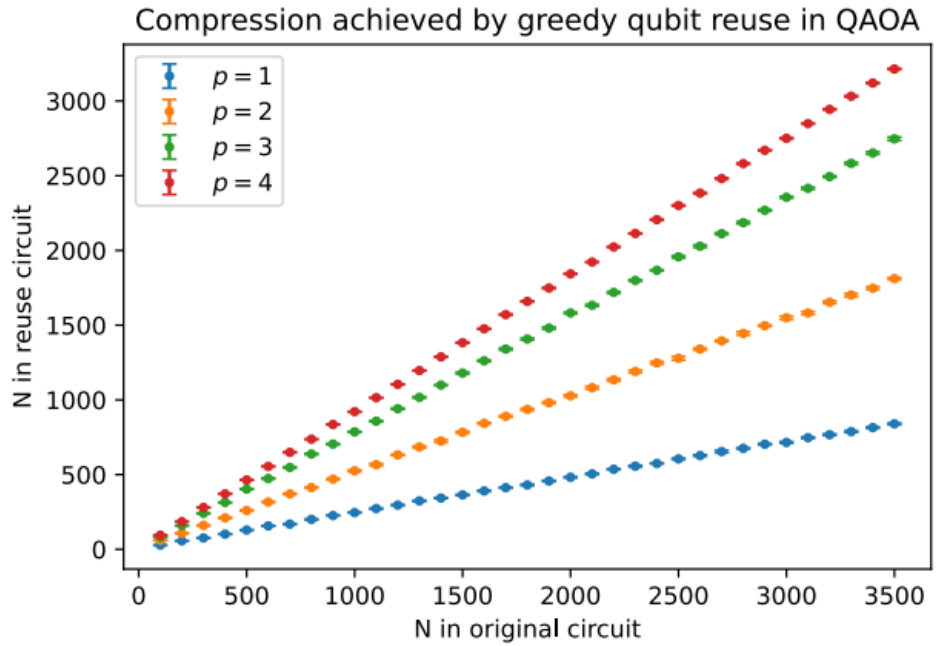
Quantum Approximate Optimization Algorithm (QAOA) MaxCut circuit

80-qubit circuit → 20 qubits (H1)

Mathew DeCross, Eli Chertkov, Megan Kohagen, Michael Foss-Feig, arxiv:2210.08039

130-qubit circuit → 32 qubits (H2)

Moses, S. A., et. al., A Race Track Trapped-Ion Quantum Processor. arXiv:2305.03828



Quantinuum Systems

Syntax Checker

- Ensure that your quantum circuit will run on Quantinuum hardware before submitting jobs
- Checks the quantum circuit syntax against a device's compiler
- Free to use, does not require H-System Quantum Credits (HQCs)

Emulator

- Classical emulation of the H-Series quantum computers
- Realistic physical and noise models of the devices
- Requires HQCs

Hardware

- Trapped ion quantum computers
- Requires HQCs

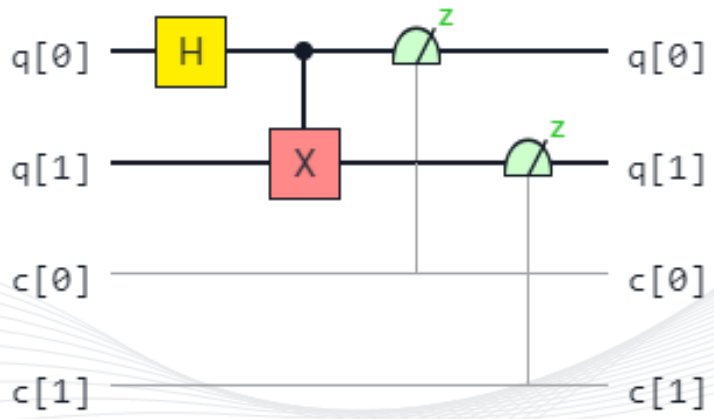
Estimating H-System Quantum Credits (HQCs)

$$HQC = 5 + \frac{N_{1q} + 10N_{2q} + 5N_m}{5000} C$$

- N_{1q}: number of single-qubit gates
- N_{2q}: number of two-qubit gates
- N_m: number of state preparation and measurement (SPAM) operations

Example: Bell State with 100 shots

$$HQC = 5 + \frac{1+10(1)+5(4)}{5000}(100) = 5.62$$



SYSTEM MODEL H-SERIES

Quantum Computers

Three trapped ion quantum computers

H2-1 (machine target: `H2-1`) ==> 32 qubits
H1-1 (machine target: `H1-1`) ==> 20 qubits
H1-2 (machine target: `H1-2`) ==> 20 qubits

Emulators

Emulators are available that model each machine's specific ion transport and error rates.

H2-1 Emulator (machine target: `H2-1E`)
H1-1 Emulator (machine target: `H1-1E`)
H1-2 Emulator (machine target: `H1-2E`)

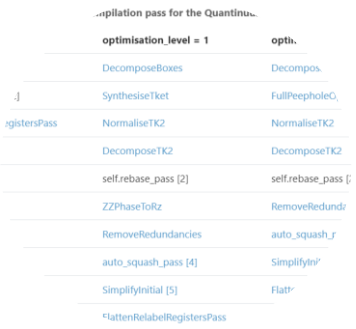
Syntax Checkers

The syntax checkers are provided to check program syntax and are specific to each device.

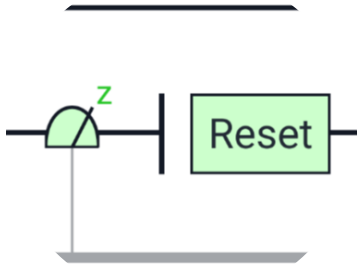
H2-1 Syntax Checker (machine target: `H2-1SC`)
H1-1 Syntax Checker (machine target: `H1-1SC`)
H1-2 Syntax Checker (machine target: `H1-2SC`)

pytket-quantinuum

Provides access to various H-series QPUs, emulators, and syntax checkers. Emulators and syntax checkers are available for specific devices, and each device has its own specifications.



Offers a default compilation pass that optimizes circuits based on different levels of optimization. The optimization levels range from 0 to 2, with level 2 being the default, applying more intensive optimizations.



Provides predicates that circuits must satisfy to run on H-series devices. It Supports mid-circuit measurements, fast classical feedforward, cost calculation, and partial results retrieval.

Supports batching of jobs (circuits), allowing submission of multiple circuits together as a batch, which will be executed one after another on the QPU.

Allows control of the language used for circuit submission such as QASM and QIR*.

* Under development

ACCESS THE H-SERIES QUANTUM COMPUTER

Start with ORNL

Microsoft Azure

Start with Azure Quantum

INQUANTO™

Start with InQuanto

Agenda for this TKET workshop

1) Introduction to TKET and Quantinuum H-series unique features

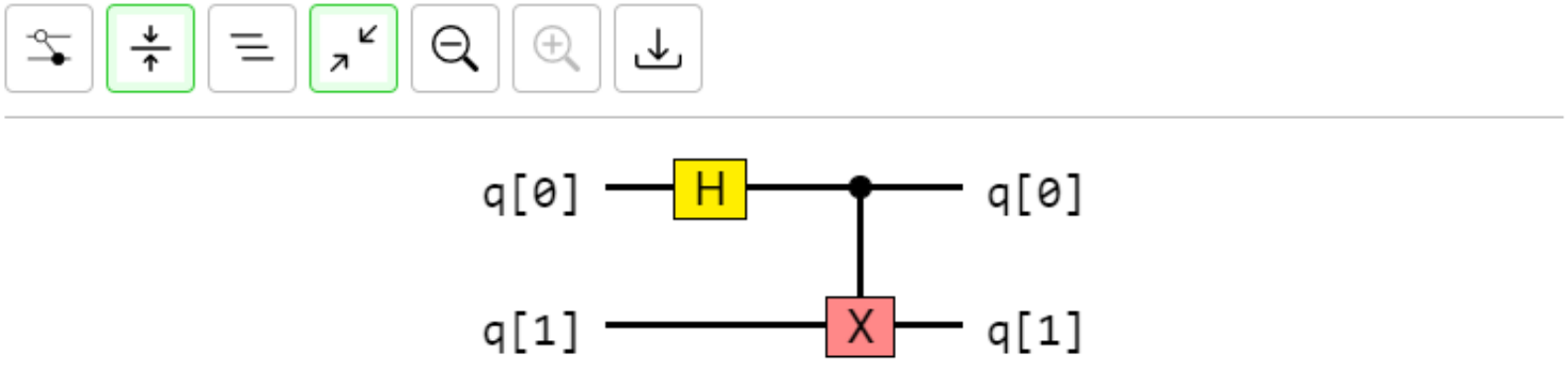
2) Go over some pytket-quantinuum coding examples

3) Finish with some pytket-qiskit examples if there is time

QUANTINUUM

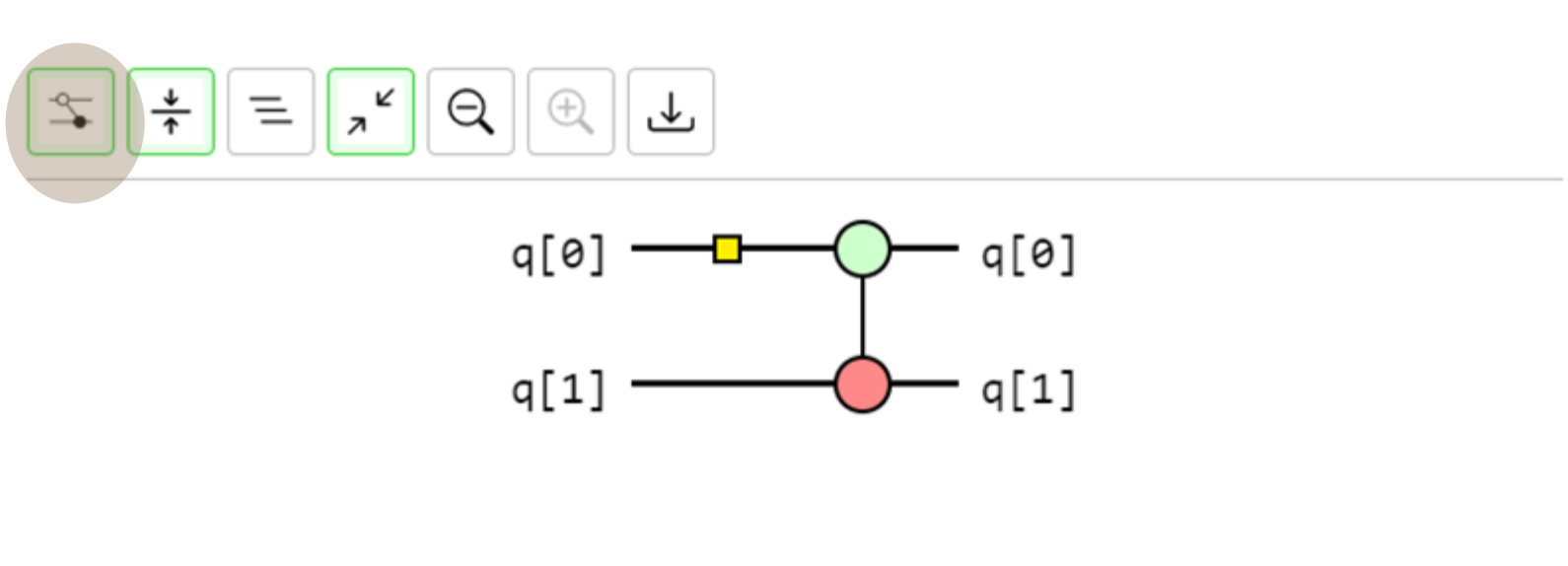
CIRCUIT CLASS

```python
from pytket import Circuit
from pytket.circuit.display import render_circuit_jupyter

circ = Circuit(2)
circ.H(0).CX(0,1)

render_circuit_jupyter(circ)
```

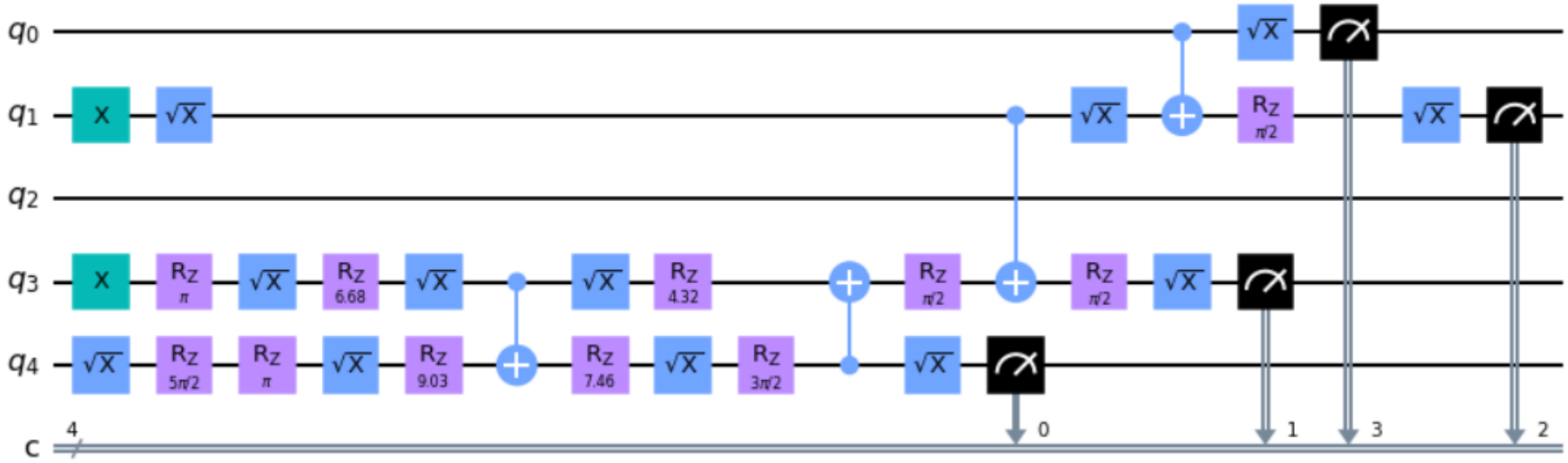


ZX CALCULUS REPRESENTATION

```
1 from pytket import Circuit
2 from pytket.circuit.display import render_circuit_jupyter
3
4 circ = Circuit(2)
5 circ.H(0).CX(0,1)
6
7 render_circuit_jupyter(circ)
```



Qiskit Quantum circuit



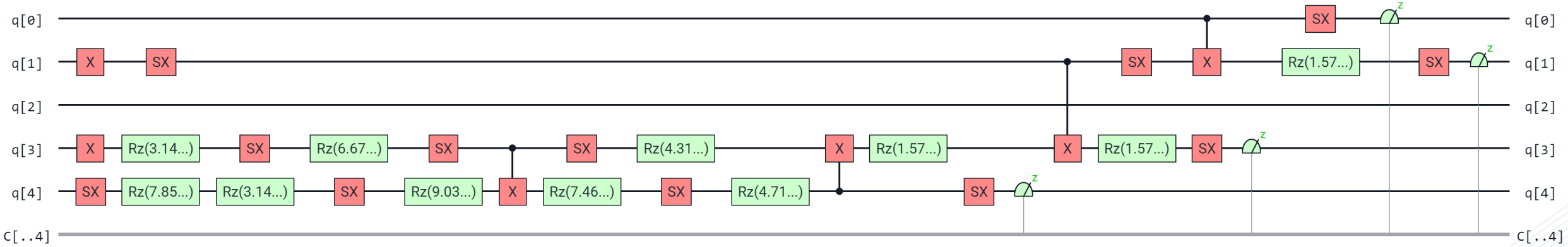
INTEGRATING OTHER QUANTUM TOOLKITS IS EASY

qiskit_to_tk(circuit)

tk_to_qiskit(circuit)

Other bidirectional conversions exist: Cirq, Bracket etc.

TKET Quantum circuit



RESOURCES

GitHub

https://github.com/CQCL/tket

User Manual

https://tket.quantinuum.com/user-manual/

Slack

https://tketusers.slack.com

H1-series

https://www.quantinuum.com/hardware#access

Other tools based on TKET: Qermit, Lambeq, InQuanto