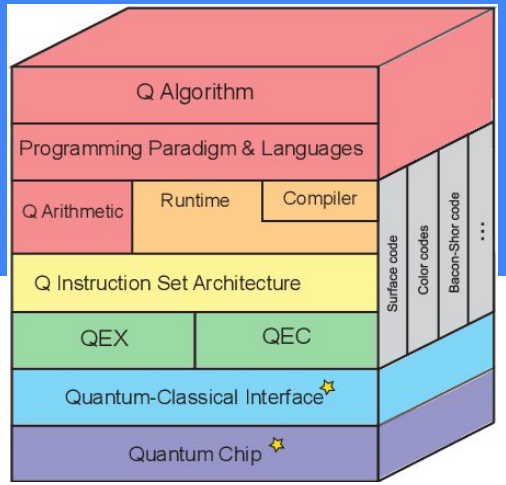
t|ket Compiler with evaluation results

Introduction

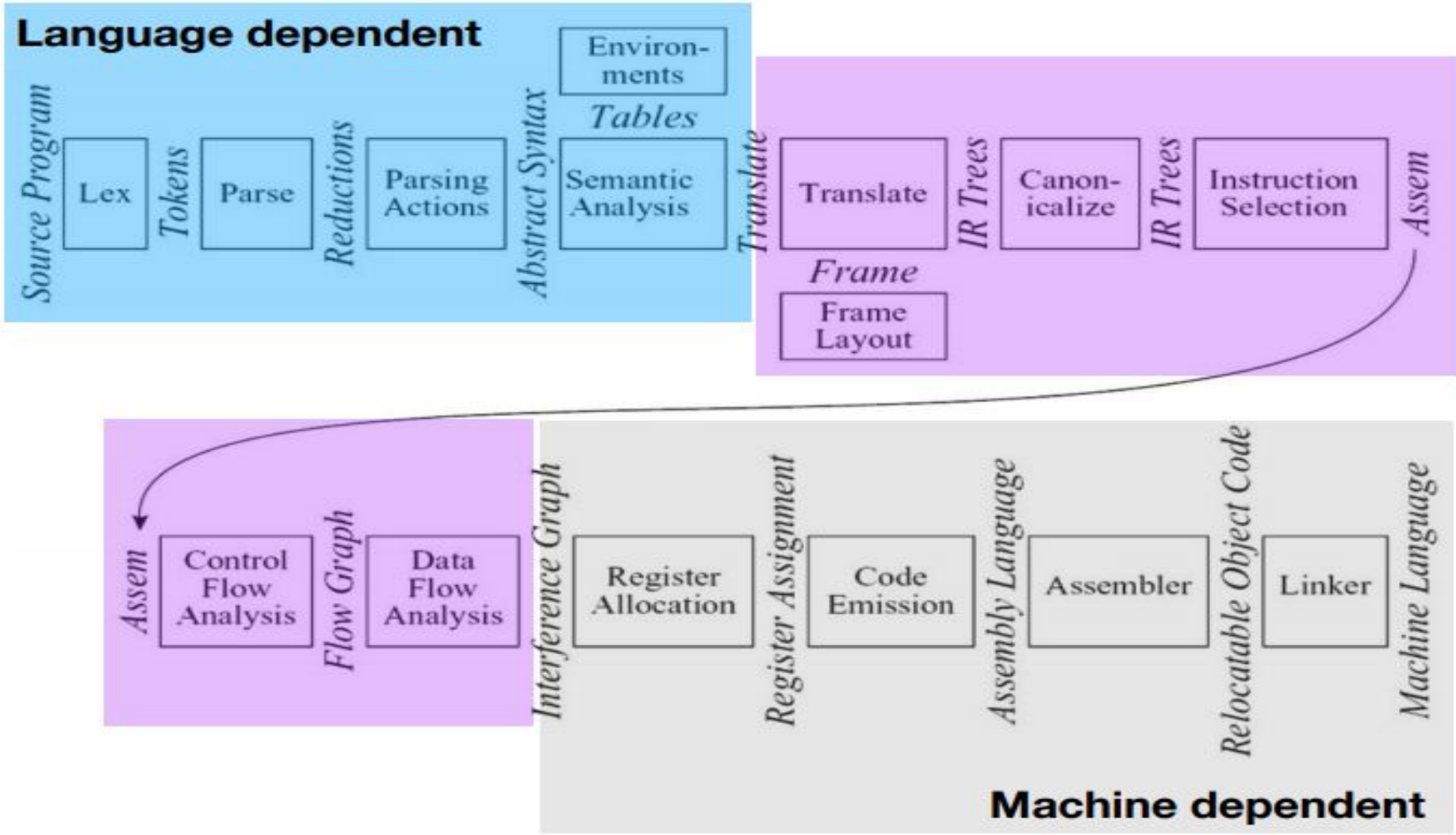
1. t|ket is a language and platform agnostic retargetable compiler for NISQ devices.
2. Generates optimized code for variety of NISQ devices.
3. Minimizes noise by optimising the circuit size.
4. Use whatever language you want to write the program and run the program on whatever device you want.
5. Device error rate used by the compiler to improve overall performance.

Compilers

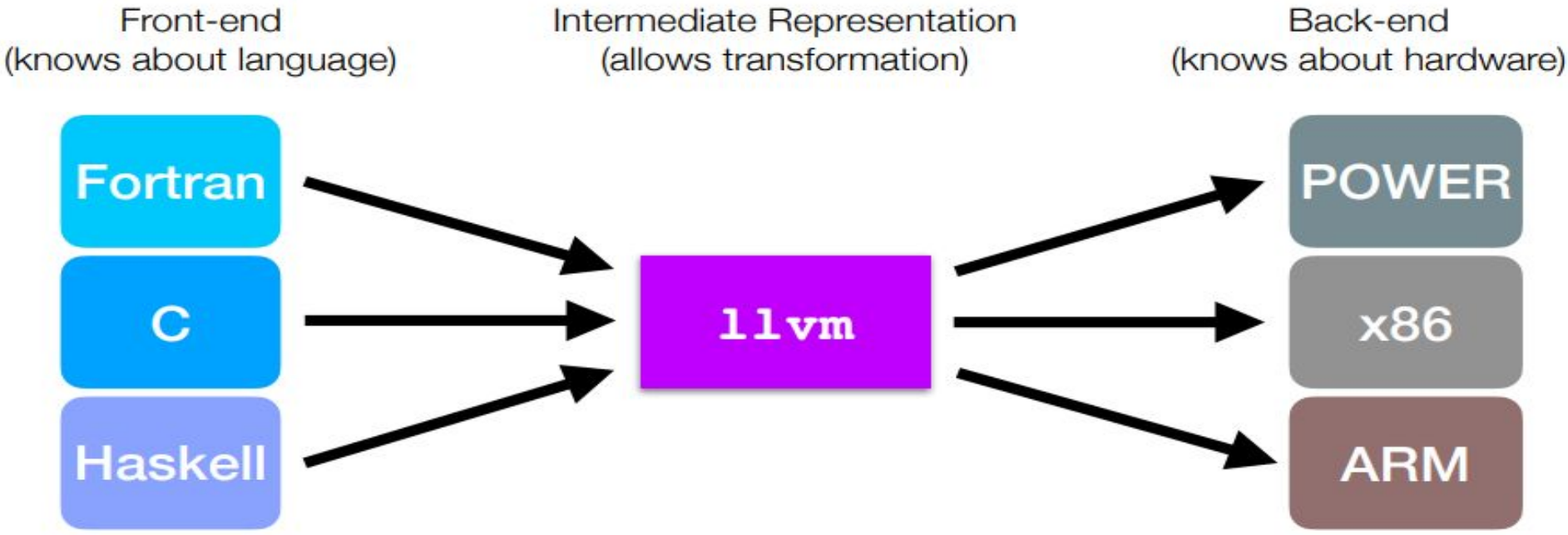


High-level language
(for humans)

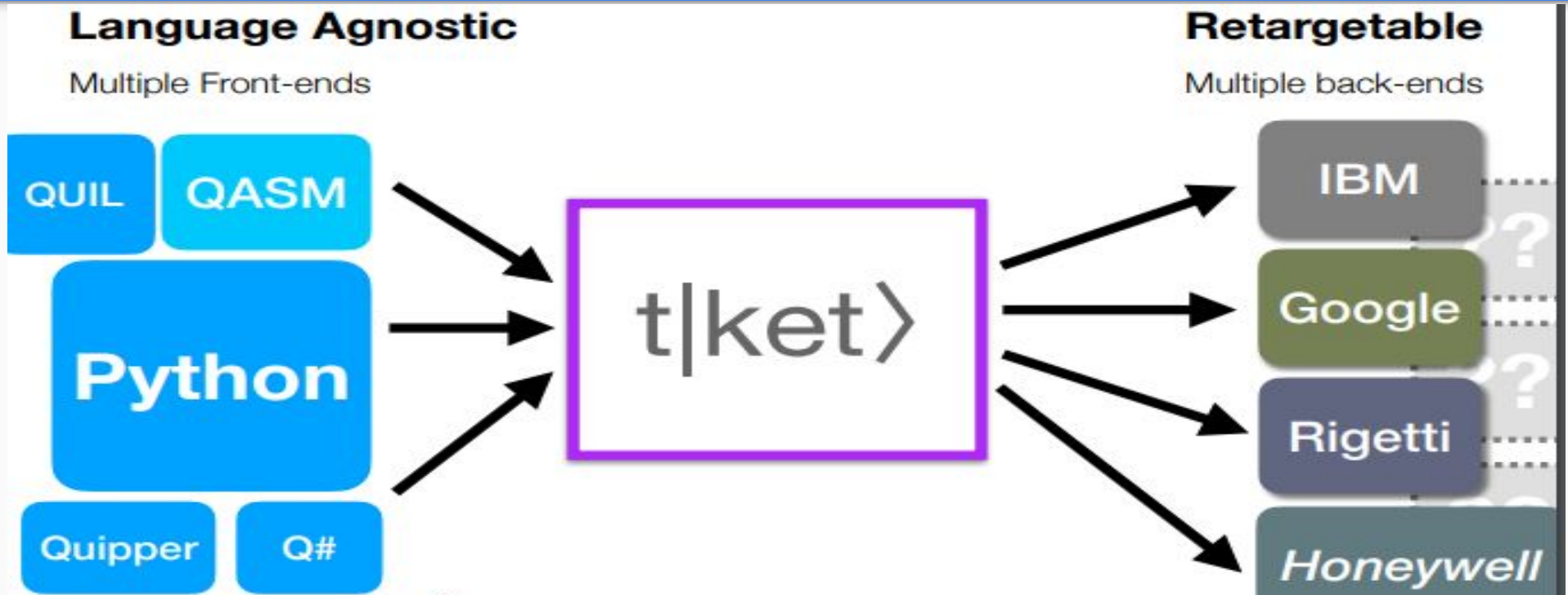
Machine language
(for machines)

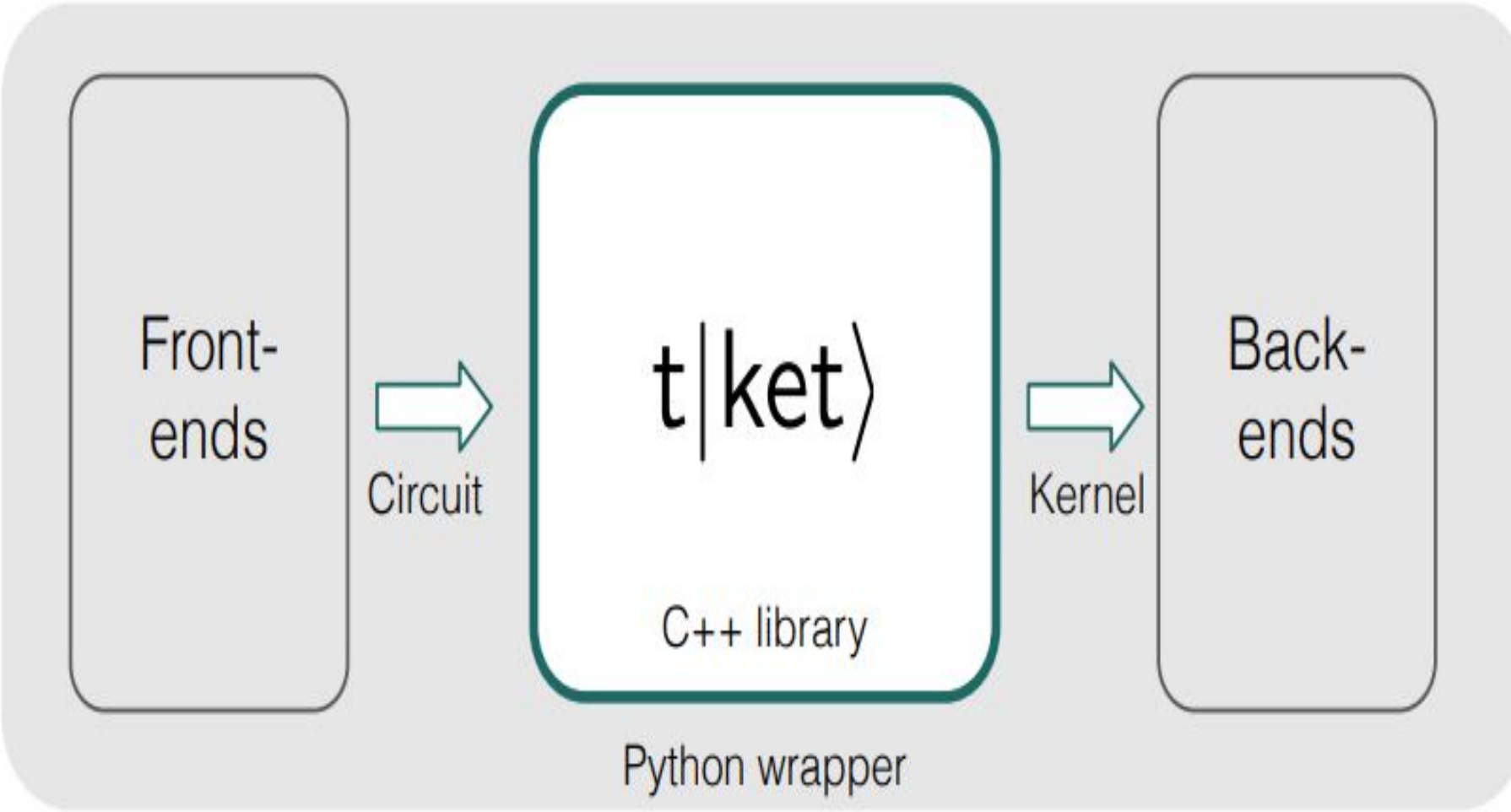

Language dependent
Source Program
Lex
Tokens
Parse
Reductions
Parsing Actions
Abstract Syntax
Semantic Analysis
Environ-ments
Tables
Translate
Translate
Frame
Frame Layout
IR Trees
Canon-icalize
IR Trees
Instruction Selection
Assem
Assem
Control Flow Analysis
Flow Graph
Data Flow Analysis
Interference Graph
Register Allocation
Register Assignment
Code Emission
Assembly Language
Assembler
Relocatable Object Code
Linker
Machine Language
Machine dependent

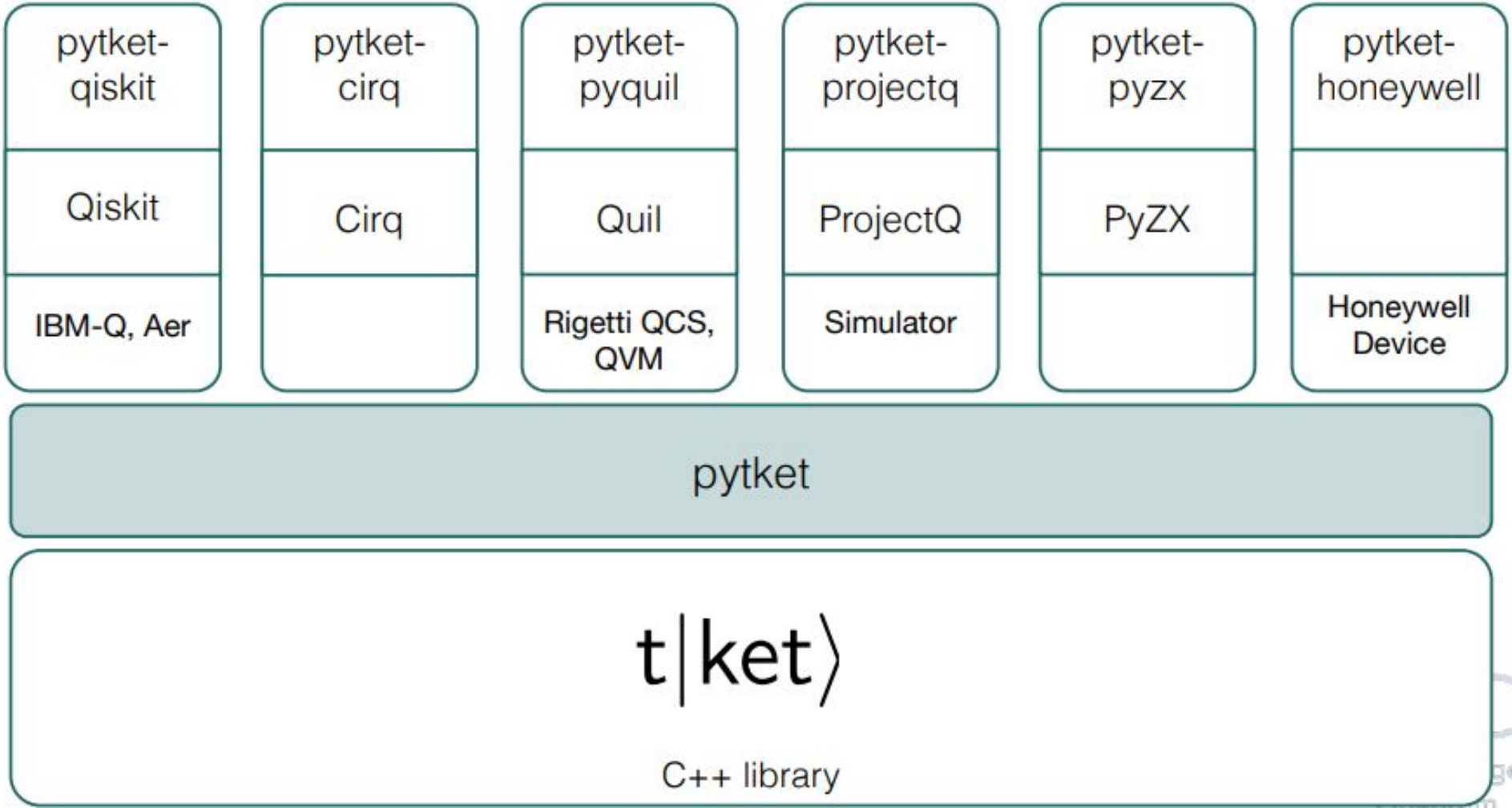
Modern Compiler



t|ket compiler

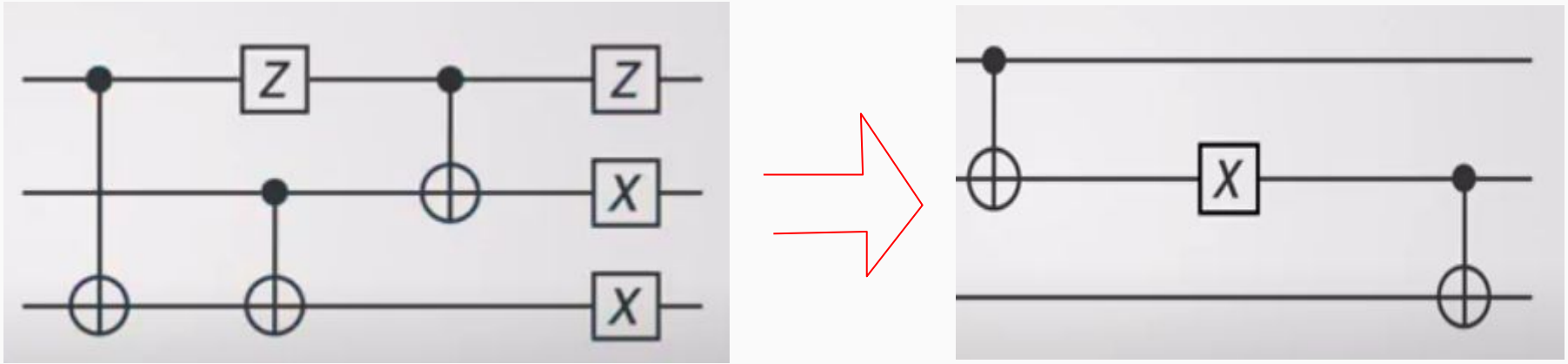



Front-ends
Circuit
t|ket⟩
C++ library
Kernel
Back-ends
Python wrapper


pytket-qiskit
Qiskit
IBM-Q, Aer
pytket-cirq
Cirq
pytket-pyquil
Quil
Rigetti QCS, QVM
pytket-projectq
ProjectQ
Simulator
pytket-pyzx
PyZX
pytket-honeywell
Honeywell Device
pytket
t|ket⟩
C++ library

Improvements, we can achieve using T|ket>

- It allows us to write our solution in variety of input formats and languages and Tket takes care of making it compatible for range of Quantum computer and simulator.
- It will also rewrite our circuit to make it smaller and reduce gate count.



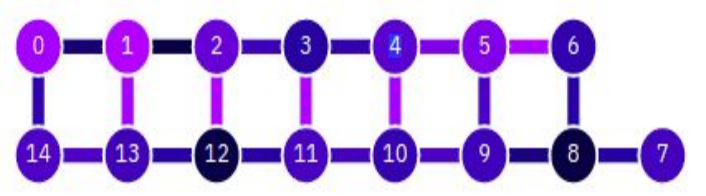
Why should we be concerned about this??

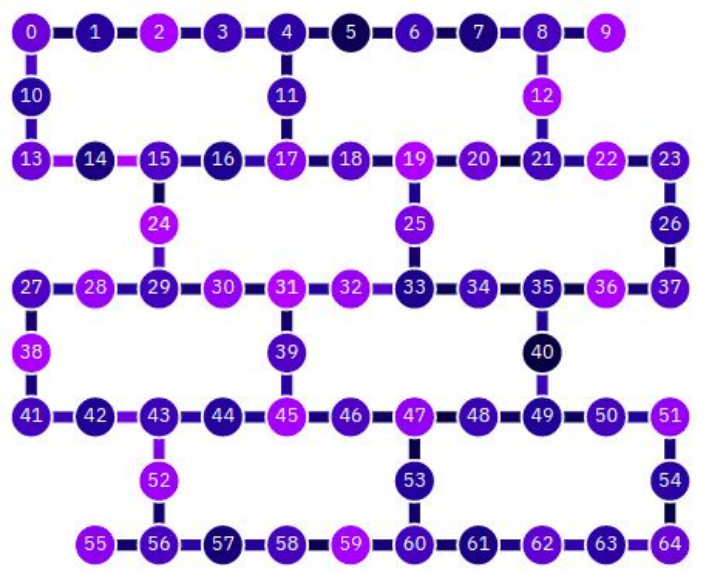
- **Rewriting circuit creates a room for improvement in terms of circuit depth, which will allow us to use CVQLS even for large matrices and circuit will be executed within T2 time.**

- **Reducing gate count will also reduce the error upto an extent.**

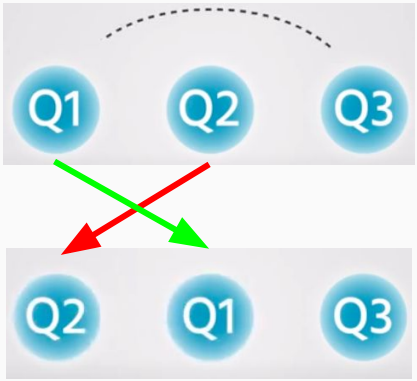
Qubit Routing

- In reality all qubits are not able to talk to each other, interaction is limited to specific pairs.





- So the circuit goes through “swapping” before being fed to a real chip



- Any real transmon qubit Quantum computer currently available must go through this swapping before feeding the circuit design to the chip but **TKET claims to do it with minimum number of swaps.**

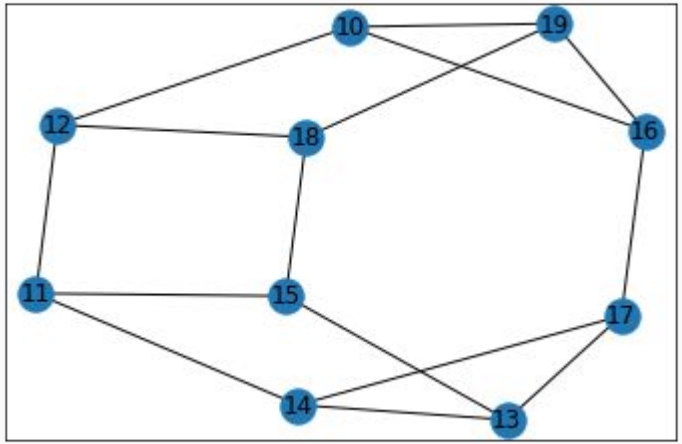
LiH simulation using T|ket>

- Claims to achieve 60% reduction in **Gate Count**, **Circuit Depth** and **Two-qubit Gates** over Qiskit and PyZX

Example of how |Tket> compiles

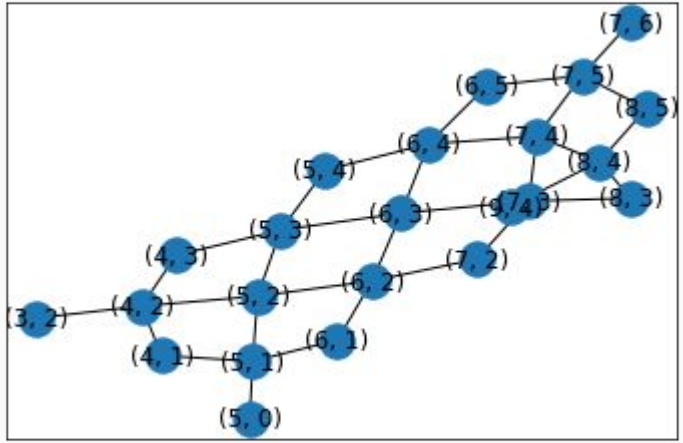
- QAOA algorithm on Sycamore23 chip

- The circuit shown here has no knowledge of qubit connectivity on the chip

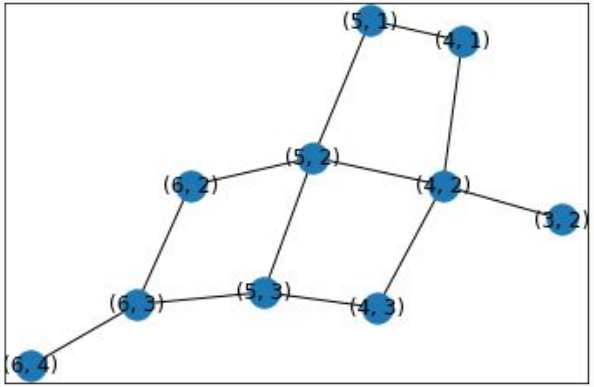


A 10 qubit circuit implementation for QAOA algorithm

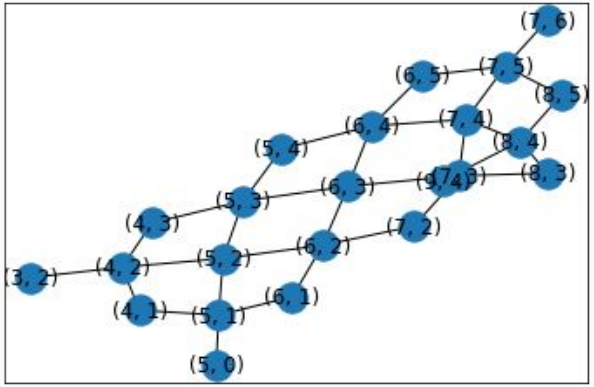
Sycamore-23 architecture



After being compiled by pytket



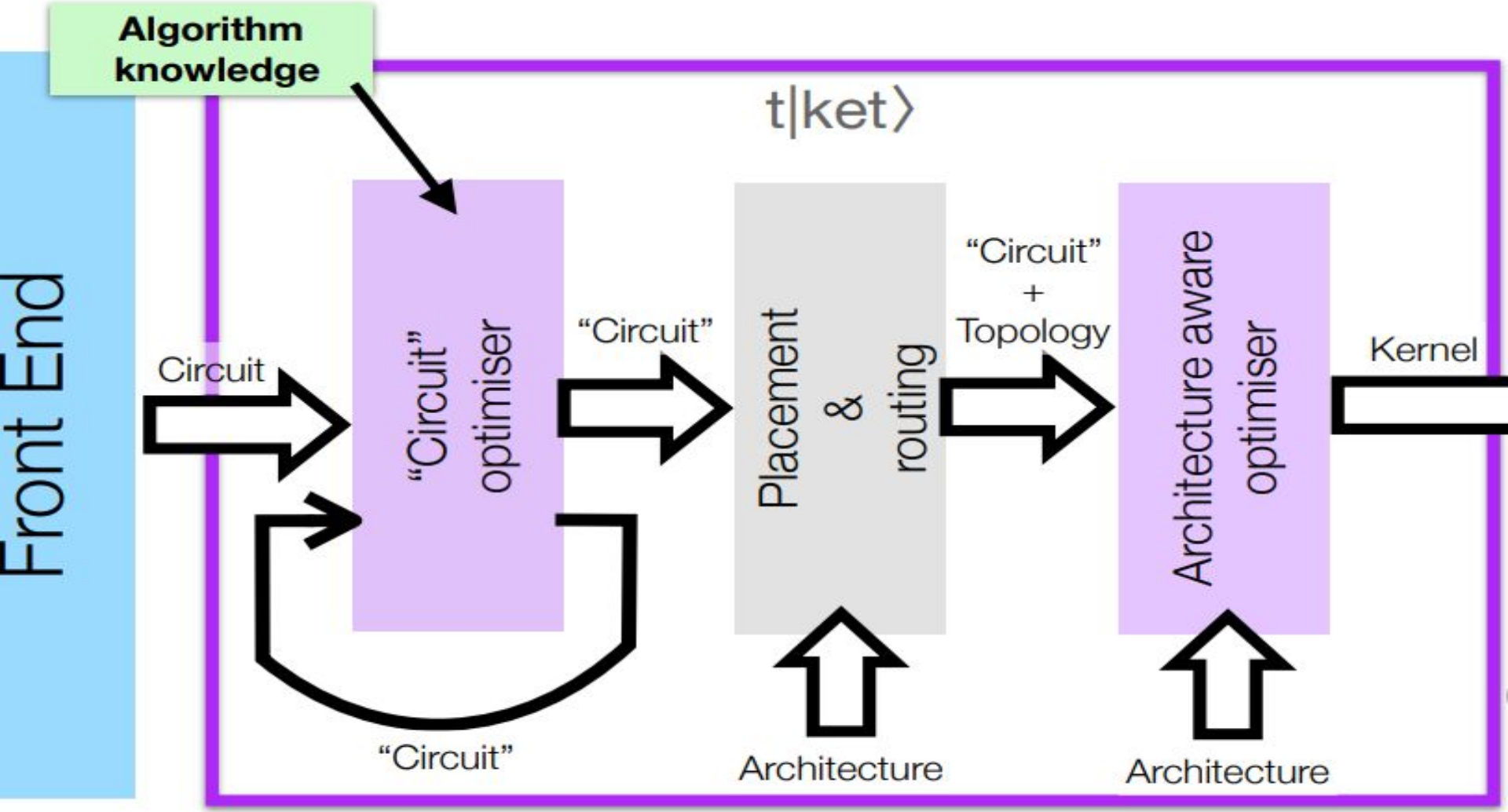
After getting compiled by TKET



Sycamore-23 architecture

pytket example

```
1  circuit = circuit_from_qasm('input.qasm')
2  q0, q1 = circuit.qubits[:2]
3  circuit = circuit.H(q0).CX(q0, q1).measure_all()
4
5  backend = IBMQBackend('ibmq_ourense')
6  if not backend.valid_circuit(circuit):
7      circuit = backend.compile_circuit(circuit)
8      assert backend.valid_circuit(circuit)
9
10 backend.process_circuits([circuit], n_shots=100)
11 results = backend.get_shots(circuit)
```


Algorithm knowledge
t|ket⟩
Front End
Circuit
"Circuit" optimiser
"Circuit"
Placement & routing
"Circuit" + Topology
Architecture aware optimiser
Kernel
"Circuit"
Architecture
Architecture

Designing circuit directly on pytket

```
from pytket import Circuit
c = Circuit(2,2) # define a circuit with 2 qubits and 2 bits
c.H(0)           # add a Hadamard gate to qubit 0
c.Rz(0.25, 0)    # add an Rz gate of angle 0.25*pi to qubit 0
c.CX(1,0)        # add a CX gate with control qubit 1 and target qubit 0
c.measure_all()  # measure qubits 0 and 1, recording the results in bits 0 and 1
```

Importing circuit from other frameworks

```
from pytket.qasm import circuit_from_qasm
c = circuit_from_qasm("my_qasm_file.qasm")
```

Or, if an extension module like `pytket-qiskit` is installed,

```
from qiskit import QuantumCircuit
qc = QuantumCircuit()
# ...
from pytket.extensions.qiskit import qiskit_to_tk
c = qiskit_to_tk(qc)
```

Backend options

```
from pytket.backends.ibm import IBMQBackend
b = IBMQBackend("ibmq_london")
# ...
from pytket.backends.forest import ForestBackend
b = ForestBackend("Aspen-8")
# ...
from pytket.backends.aqt import AQTBackend
b = AQTBackend(access_token, "sim")
# ...
b.compile_circuit(c)        # performs the minimal compilation to satisfy the device/simula
handle = b.process_circuit(c, 10)   # run the circuit 10 times
shots = b.get_shots(handle) # retrieve and return the readouts
print(shots)
```

Qiskit Vs Pytket comparison results

1.In the next page we show the results of running the VQLS algorithm in the Qiskit transpiler and pytket compiler.

2. We have used the COBYLA optimizer and compared the results of gate counts in terms of unitary gates and CX gates.

3. Also the number of iterations are also taken into account for it.

	Qiskit with COBYLA optimizer	Qiskit-TKET with COBYLA optimizer
U3 gate count	53	38
U1 gate count	0	15
CX gate count	55	55
Cost After **40 iteration**	0.0003693821224256544	0.00500301944965231 3
Optimization time for **40 iterations**	1 min 54 sec	2 min 42 sec
Accuracy after **40 iterations**	82.22297491439735 %	72.30226203179629 %
Distance b/w classical and Quantum value ,**40 iterations**	0.1777702508560265	0.276977 3796820372 3
Cost After **100 iteration**	0.0004544855125172553 3	0.0003258426945879078
Optimization time for **100 iterations**	4 min 47 sec	6 min 46 sec
Accuracy after **100 iterations**	75.61563170907158 %	72.20211269941248 %
Distance b/w classical and Quantum value ,**100 iterations**	0.2438436829092842 7	0.2779788730058751 5

Pytket passes results on VQLS algorithm

1. In the next page we show the results of applying various pytket passes on VQLS.
2. We use passes separately and also as combination of passes in Sequence passes form
3. The metrics we get are the time required when we apply the passes and the accuracy we get and the get count reduction.

***All results are after 40 iterations**

	Gate count	Time taken	Optimized Cost	Accuracy	Distance
qiskit - only	U3 - 53 CX - 55	1 min 53 sec	0.000525007373278541	83.73 %	0.1626927283501346
Normal Pytket compilation with ticket wrapped Aerbackend	U3 - 38 U1 - 15 CX - 55	2 min 39 sec	0.0010305353715852617	81.36 %	0.18634833946606766
DecomposeMultiQubitsIBM	U3 - 38 U1 - 15 CX - 55	2 min 39 sec	0.0012613972072137436	82.21 %	0.17784009022116523
FullPeepholeOptimise		2 min 39 sec	0.001007969162470368	86.25 %	0.1374374796152781
RemoveRedundancies		2 min 39 sec	0.000679152987779652	85.62 %	0.14373037466842226
USquashIBM		2 min 39 sec	0.0006889237815229654	80.56 %	0.19433904903381855
SequencePass, with all passes shown above		2 min 39 sec	0.0008735010873487381	85.89 %	0.1410256931712874