

Automating Program Translation & Migration

From Circuit Level to Program Level

Xiaoyu Guo

Supervisor: Jianjun Zhao

Kyushu University



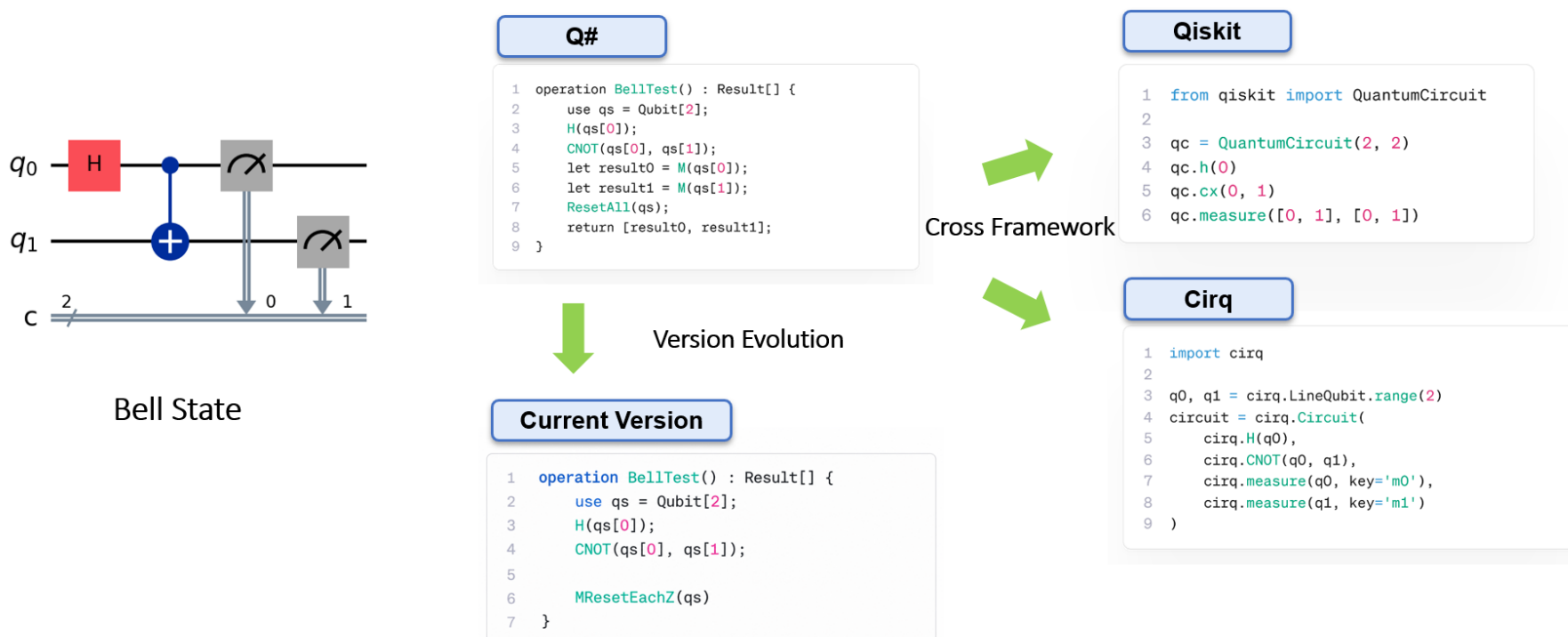
- Motivation
- Approach
- Conclusion

Motivation

Quantum Program Translation

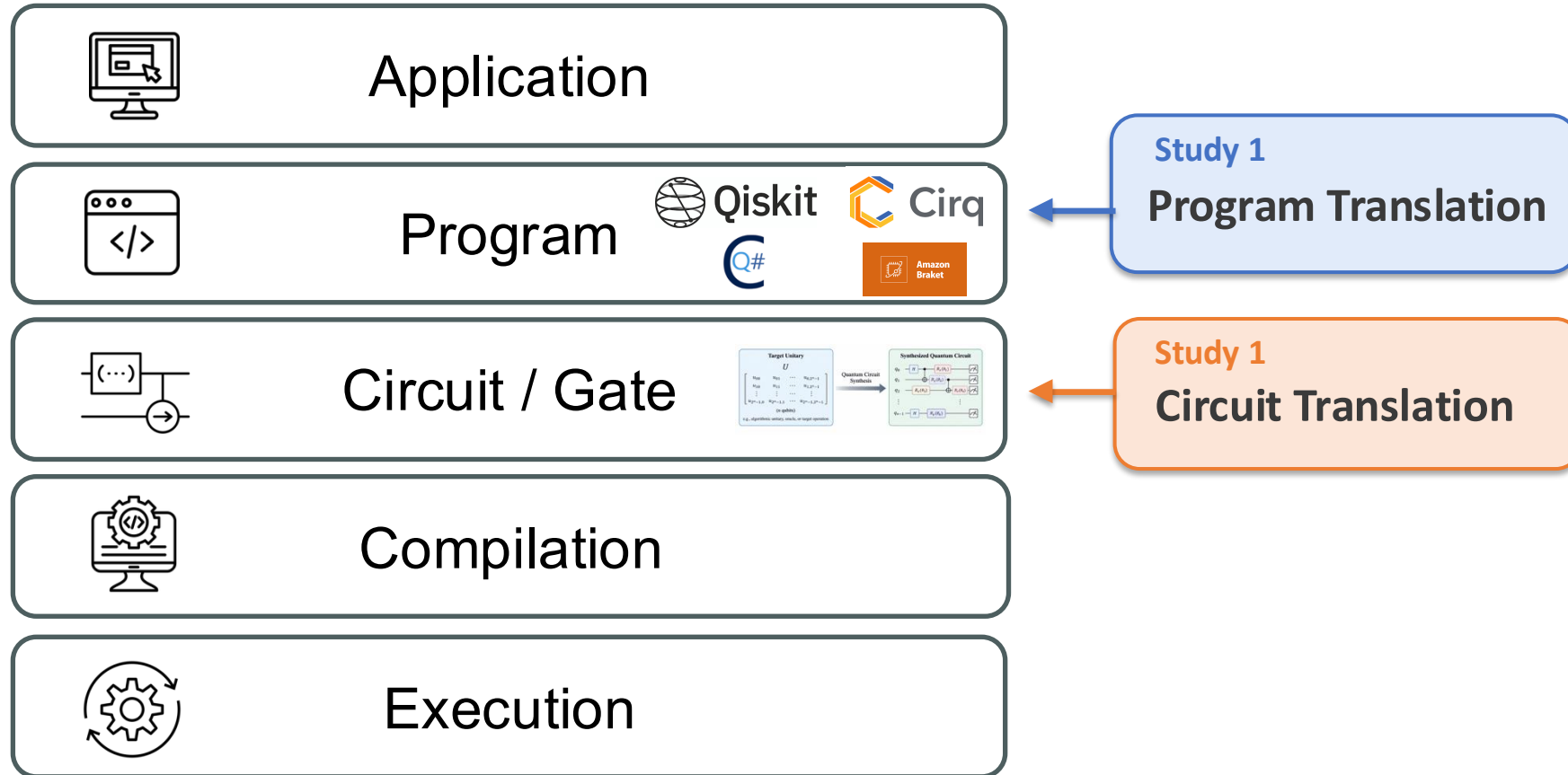
Definition

- *Code Translation* is the process of converting a program written in one programming language or framework into an equivalent program in another language or framework while preserving its intended behavior.
- *Code Migration* is the process of adapting existing program to a different version while preserving its functionality and compatibility.



● Quantum Software Abstraction Level

Different abstraction levels introduce different development challenges and require different forms of automation



■ Motivation Example

- How can we preserve program behavior across heterogeneous quantum frameworks?
 - Time Consuming
 - Consistency



Qiskit 0.46



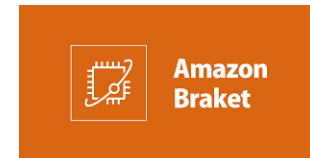
Migration



Qiskit 2.0



Translation



Braket



Q#



Cirq

Cirq

- Framework Translation
 - Q-Convert [1]
 - Quantum circuit simulator support import from OpenQASM, Quil, IONQ, and export to OpenQASM, pyQuil, Quil, Qiskit, Cirq, TensorFlow Quantum, Q#, and CudaQ
 - LinguaQuanta [2]
 - Quantum transpiler between Quipper language and OpenQASM
 - Quantum frameworks
 - Current frameworks support translation through OpenQASM, QIR to other frameworks.

Existing work has studied circuit translation. However, most translation mechanisms are predefined, circuit-centric, or version-specific. Meanwhile, empirical studies show that quantum frameworks and compiler transformations can contain subtle semantic defects [3].

[1] <https://github.com/quantastica/quantum-circuit>

[2] Wesley S. LinguaQuanta: towards a quantum Transpiler between OpenQASM and Quipper[C]//International Conference on Reversible Computation. Cham: Springer Nature Switzerland, 2024: 142-160.

[3] Wang J, Zhang Q, Xu G H, et al. QDiff: Differential testing of quantum software stacks[C]//2021 36th IEEE/ACM international conference on automated software engineering (ASE). IEEE, 2021: 692-704.

- Version Migration
 - Automatic Qiskit code Refactoring[1, 2]
 - Refactoring Qiskit code using LLMs guided by a domain specific taxonomy
 - Migrating QAOA from Qiskit 1.x to 2.x [3]
 - Hidden quantum-classical interface behaviors can still lead to significant variations in result.
 - Quantum frameworks [4]
 - Qiskit propose migration tool flake8-qiskit-migration.

Existing work has begun to automate version migration, particularly for Qiskit, but current approaches still rely heavily on manually curated migration knowledge from documentation and release notes.

[1] Suárez J M, Bibbó L M, Bogado J, et al. Automatic qiskit code refactoring using large language models[C]//2025 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON). IEEE, 2025: 1-8.

[2] Suárez J M, Bibbó L M, Bogado J, et al. Taxonomy of migration scenarios for Qiskit refactoring using LLMs[J]. arXiv preprint arXiv:2506.07135, 2025.

[3] Cardinal J, Benzarti I, Boussaidi G E, et al. Migrating QAOA from Qiskit 1. x to 2. x: An experience report[C]//Proceedings of the 7th IEEE/ACM International Workshop on Quantum Software Engineering. 2026: 25-32.

[4] <https://github.com/qiskit-community/flake8-qiskit-migration>

- Existing translation support is primarily circuit centric
Lack program level translation and migration, quantum software is no longer just a circuit

Qiskit

```
1 from qiskit import QuantumCircuit
2
3 qc = QuantumCircuit(2, 2)
4 qc.h(0)
5 qc.cx(0, 1)
6 qc.measure([0, 1], [0, 1])
```

Circuit



Cirq

```
1 import cirq
2
3 q0, q1 = cirq.LineQubit.range(2)
4 circuit = cirq.Circuit(
5     cirq.H(q0),
6     cirq.CNOT(q0, q1),
7     cirq.measure(q0, key='m0'),
8     cirq.measure(q1, key='m1')
9 )
```

Program



```
1 from qiskit import execute
2 from qiskit.providers.aer import AerSimulator
3
4 backend = AerSimulator()
5 job = execute(qc, backend, shots=1024)
6 result = job.result()
7 counts = result.get_counts()
8 print(counts)
```

```
1 import cirq
2 import numpy as np
3
4 simulator = cirq.Simulator()
5 result = simulator.run(circuit, repetitions=1024)
6 m0 = result.measurements['m0']
7 m1 = result.measurements['m1']
8 counts = {
9     '00': int(np.sum((m0 == 0) & (m1 == 0))),
10    '01': int(np.sum((m0 == 0) & (m1 == 1))),
11    '10': int(np.sum((m0 == 1) & (m1 == 0))),
12    '11': int(np.sum((m0 == 1) & (m1 == 1)))
13 }
14 print(counts)
```

➤ Rule based circuit translation is not always reliable

Existing framework testing studies show that quantum software platforms themselves can contain defects, including issues in transformations, compilation, and execution.

ID	Trigger	Phenomenon	Reason
1	QASM round-trip + null-effect injection	Gate or opaque call to 'subcircuit'	exporter generated illegal QASM for subcircuit with classical register
2	QASM round-trip	Duplicate declaration for gate 'rzx'	Gate definition is repeated during QASM export/import process
3	optimization + QASM round-trip + null-effect	Cannot find gate definition for unitary	unitary/custom gate is not preserved correctly in QASM
4	QASM round-trip + null-effect	Cannot find gate definition for 'rcccX'	custom/compound gate definition is missing

Translation bug examples founded by MorphQ [1]

➤ Unreliable LLMs

LLMs often generate outdated or version mismatched APIs, leading to non-executable quantum programs [1].

Qiskit < 1.0

Execute

```
from qiskit import execute
from qiskit_aer import AerSimulator

backend = AerSimulator()
job = execute(qc, backend, shots=1024)
result = job.result()
counts = result.get_counts()
```



Qiskit > 1.0

transpile

```
from qiskit import transpile
from qiskit_aer import AerSimulator

backend = AerSimulator()
qc_new = transpile(qc, backend)

job = backend.run(qc_new, shots=1024)
result = job.result()
counts = result.get_counts()
```

➤ Current translation & migration is mostly handcrafted and version-specific

Existing translation mechanisms are largely predefined, framework-specific, or circuit-oriented. They require manually maintained mappings and therefore struggle to adapt to rapidly evolving APIs and program-level semantics.

Name	Framework	Version
Automatic Qiskit Code Refactoring [2]	Qiskit	from 0.41-0.45 to 0.46
Q-Convert [1]	e.g. Qiskit, pyQuil	0.7, 1.9-2.1

Support framework & version

[1] <https://github.com/quantastica/quantum-circuit>

[2] Suárez J M, Bibbó L M, Bogado J, et al. Automatic qiskit code refactoring using large language models[C]//2025 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON). IEEE, 2025: 1-8.

➤ Framework evolution makes manual migration increasingly unsustainable

Frequent API deprecations, replacements, semantic changes, and default-behavior changes make manually maintained migration mappings expensive and quickly outdated.

Framework	Frequency
Qiskit	3~4 minor release/ year
Cirq	Quarterly
PennyLane	Every 2~3 month
Amazon Braket	Monthly

Framework release frequency

Approach

Method and Implementation

Challenge

Program Level Semantics

Unreliable method

Framework Heterogeneity

Rapidly API Evolution

Limited scalability



Design Goal

API Semantics Preserving

Validation

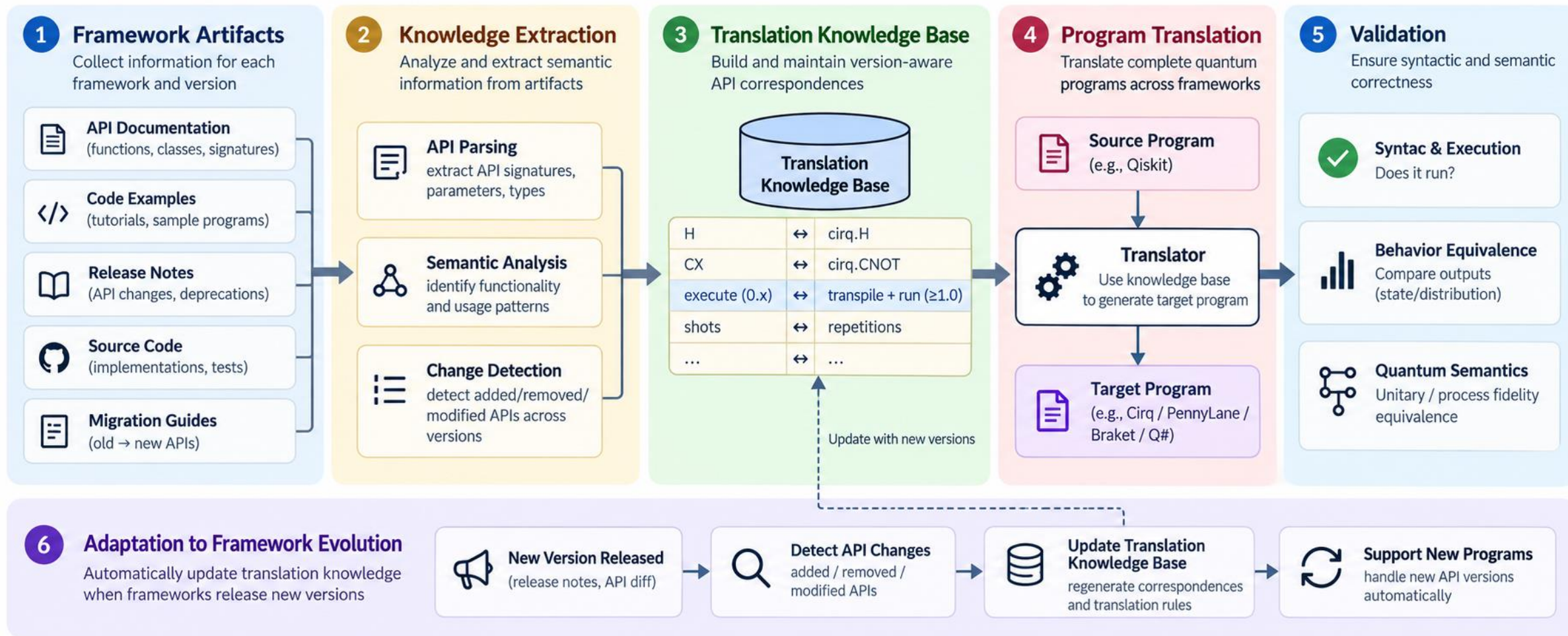
Version Aware Translation

Continuously Adaptable

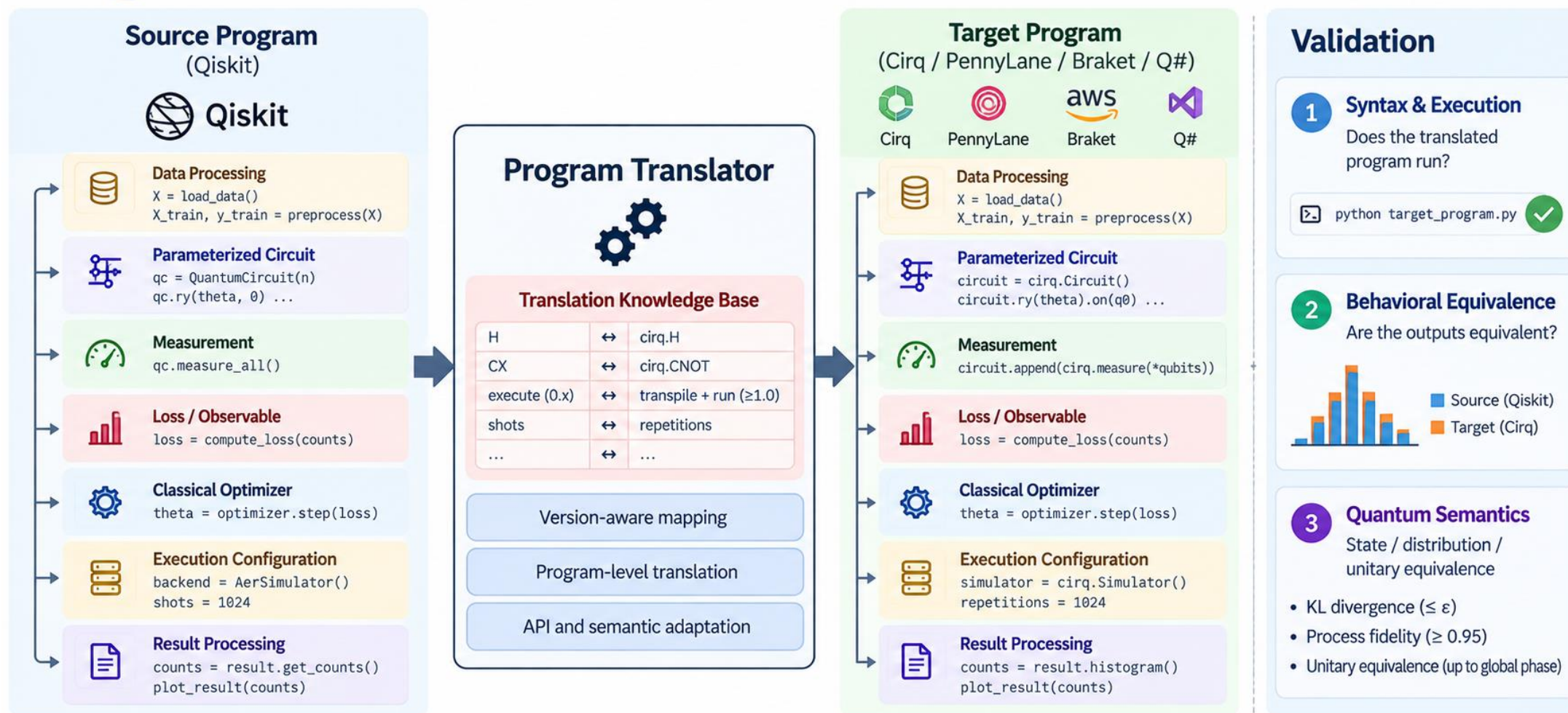
Automated

From static migration rules to continuously evolving translation knowledge

Automated and Version-Aware Program Translation



Program-Level Translation and Validation



Conclusion

Conclusion and Future Work

■ Conclusion

- Quantum software automation requires addressing challenges across multiple abstraction levels.
- Reusing API semantics and migration knowledge can reduce translation effort and support evolving quantum frameworks.
- Across abstraction levels, reusable knowledge provides a common foundation for reducing development effort and enabling more scalable quantum software automation.

Thank for your listening!

