Qiskit Updates

Matthew Treinish
Qiskit Architect – STSM
matthew.treinish@ibm.com

Qiskit Releases

- Fixed 3 month minor/major version release cadence
- Follows semver with 1 major version per year maximum frequency

0.x - < 2024

The releases prior to 1.0, that incrementally started using Rust in the 0.20.0 release. Went end-of-life in Sept. 2024 with 0.46.3

1.x - 2024

The first major release version for Qiskit. Introduced new stability policy and formally moved to semantic versioning.

This release series focused on improving the runtime performance and writing the core data model in Rust.

1.4.5 - Oct. 2025

The final release in 1.x. No longer supported except, for potential security fixes until March 2026.

2.x - >=2025

The second major version has some small Python API incompatibilities with 1.x. The current development release series which is getting new features.

Introduced C API as new public interface.

Qiskit Rust APIs in 1.x

- Qiskit **internally** contains a Rust library that is only exposed as a private Python API.
- The output of Qiskit's Rust library builds as a single shared library file that is built as an extension for Python.
- Most of Qiskit's core data model is built in Rust but still has data dependency on Python for cold code paths.
- For Qiskit's internal usage we could operate solely in the Rust domain without calling Python.
- The Rust internals enabled Qiskit to be the most performant quantum computing SDK available [1]

[1] Nation, P.D., Saki, A.A., Brandhofer, S., Bello, L., Garion, S., Treinish, M., Javadi-Abhari, A. Benchmarking the performance of quantum computing software for quantum circuit creation, manipulation and compilation. *Nat Comput Sci* **5**, 427–435 (2025). https://doi.org/10.1038/s43588-025-00792-y

Benchpress: Transpilation

Paper results:

Qiskit 1.2 vs Tket 1.31

24%
Mean reduction in 2Q-gate count

13x
Mean transpilation time improvement

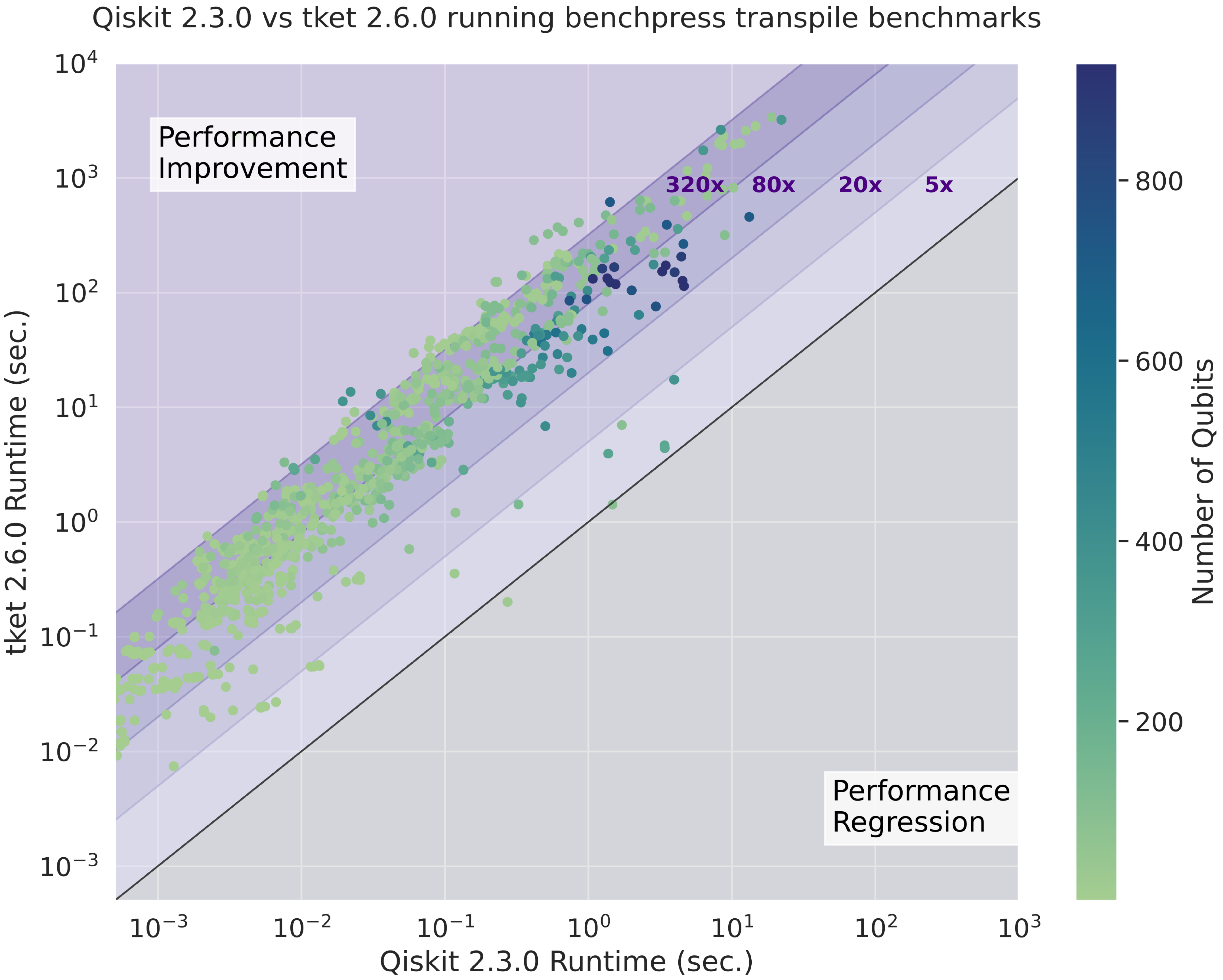
Latest results:

Qiskit 2.3.0 vs Tket 2.6.0:

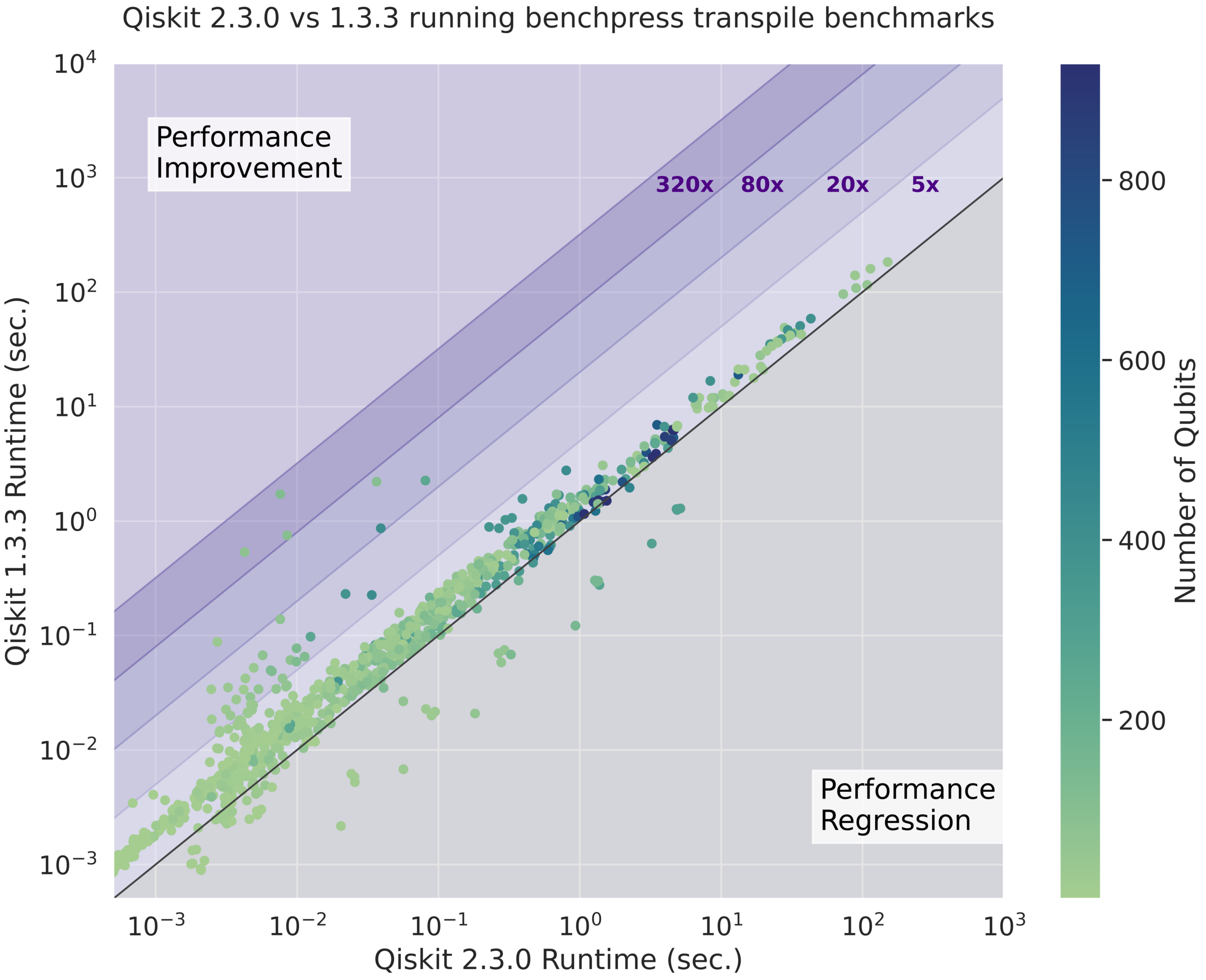
29%
Mean reduction in 2Q-gate count

93x
Mean transpilation time improvement

[Qiskit vs BQSKit: **18%** and **514x**]



Qiskit Transpiler Year on Year Performance



Qiskit Rust/Python Architecture in < 2.0

qiskit._accelerate
User interfaces
_accelerate.abi3.so
Backend
qiskit-pyext
qiskit-qasm2
qiskit-accelerate
qiskit-qasm3
qiskit-circuit

Qiskit Rust limitations before 2.x

- Operating in the Rust/compiled language domain is only available to Qiskit itself.
  - Prevents users from getting the highest performance paths.

- Rust doesn't provide a stable ABI,
  - Limits reusability and interoperability with other languages.

- Users are only able to consume Qiskit from Python.
  - If running in an environment without Python
  - Most of the Rust code in Qiskit has some runtime dependency on Python.

Qiskit C API for 2.x

- Introduce a standalone Rust crate that builds a C foreign function interface (FFI) that exposes an interface for consumption from compiled languages

- A C API can natively interface with almost every programming language (Fortran, C++, Rust, Python, Julia, etc.)

- Near term goal is to enable HPC use cases with Quantum, longer term is to enable broader usage from any software domain

- Rust has native support for building C FFIs, so adding a C API to pure Rust libraries is straightforward.

- Directly expose the Rust data model and functionality to provide a direct low-level, high-performance interface.

- Complements the existing Python high level interface.

- Two modes of operation:
  - Standalone build: `libqiskit.so`
  - Embedded in the Python extension: `_accelerate.abi3.so`

Qiskit Standalone user interfaces

Python scripts
imported in
qiskit._accelerate
exposed as
User interfaces
_accelerate.abi3.so
Backend
compiled into
qiskit-pyext
used by
C program
linked to
included in
qiskit.h
generate
libqiskit.so
compiled into
qiskit-cext
used by
Rust crates

@ 2025 IBM Corporation

Qiskit C API Embedded in the Python extension

Python scripts

imported in

qiskit._accelerate

exposed as

_accelerate.abi3.so

compiled into

qiskit-pyext

expose

C program

linked to

linked to

included in

qiskit.h

generate

libqiskit.so

compiled into

re-export

qiskit-cext

used by

used by

Rust crates

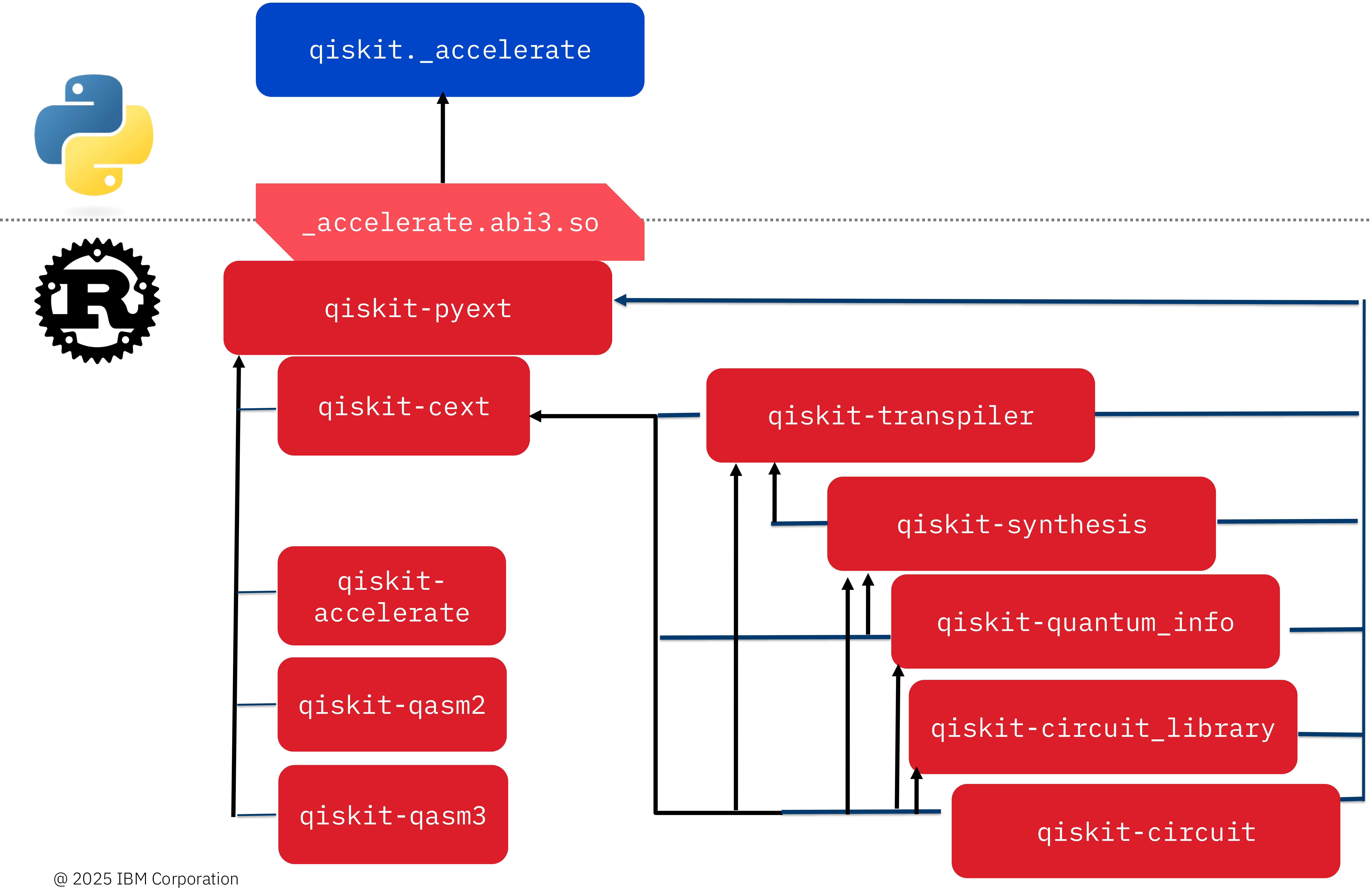
User interface

Backend

Qiskit C API Standalone mode in 2.3

C
qiskit.h
libqiskit.so
qiskit-cext
qiskit-transpiler
qiskit-circuit_library
qiskit-synthesis
qiskit-quantum_info
qiskit-circuit

Qiskit Rust/Python Architecture in 2.3



Qiskit C API in 2.3

- Majority of Qiskit's core data model is in pure Rust, Qiskit is now about 43% Rust by lines of code
- Support for transpilation, circuit construction and interaction, sparse observable construction
- Early prototype of Qiskit C Client for IBM Quantum Platform
  - https://github.com/Qiskit/qiskit-ibm-runtime-c/
  - Provides interfaces to query backends, build compilation targets for them, and to execute quantum circuits from C, without Python.
- C API still has limitations, currently doesn't have:
  - Parameterized Gates
  - DAGCircuit (compilation IR)
  - Custom transpiler passes
  - Custom gates
  - Control flow
  - Anything not defined in Rust
- C API is marked as experimental and potentially changes between minor version releases.

Using the C API

Accelerating Python with the C API

```c
#define PY_SSIZE_T_CLEAN
#include <Python.h> // include Python header for PyObject
#define QISKIT_C_PYTHON_INTERFACE // enable C->Python conversions
#include <qiskit.h>


PyObject *build_ghz(uint32_t num_bits) {
 QkCircuit *circuit = qk_circuit_new(num_bits, num_bits);
 // Add Hadamard on qubit 0
 uint32_t q0[1] = {0};
 qk_circuit_gate(circuit, QkGate_H, q0, NULL);

 // Add the CX gates
 uint32_t inter[2] = {0, 0};
 for (uint32_t i = 2; i < num_bits; i++) {
  inter[1] = i;
  qk_circuit_gate(circuit, QkGate_CX, inter, NULL);
 }

 // Measure the qubits
 for (uint32_t i = 0; i < num_bits; i++) {
  qk_circuit_measure(circuit, i, i);
 }

 return qk_circuit_to_python(circuit);
}
```

```python
def build_ghz_fast(n: int) -> QuantumCircuit:
 """Build a GHZ state skipping checks where possible."""

 circuit = QuantumCircuit(n, n)

 # use the internal Rust classes directly
 h = clib.HGate._standard_gate
 cx = clib.CXGate._standard_gate

 circuit._append(
  CircuitInstruction.from_standard(h, [circuit.qubits[0]], [])
 )

 for i in range(1, n):
  circuit._append(
  CircuitInstruction.from_standard(cx, [circuit.qubits[0], circuit.qubits[i]], [])
  )

 for i in range(n):
  circuit._append(
  CircuitInstruction(clib.Measure(), [circuit.qubits[i]], [circuit.clbits[i]])
  )

  return circuit
```

Time to create a 1000 qubit GHZ state Circuit:

C and Python: `342 µs ± 100 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)`

Python: `4.64 ms ± 330 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)`

Native Language bindings to Qiskit C API

```python
from qiskit import QuantumCircuit

def main():
  # Build a Bell state
  qc = QuantumCircuit(2, 2)
  qc.h(0)
  qc.cx(0, 1)
  qc.measure(0, 0)
  qc.measure(1, 1)
```

```c
#include <qiskit.h>

int main(int argc, char *argv[]) {
  // Build a Bell state
  QkCircuit *qc = qk_circuit_new(2, 2);

  uint32_t h_qargs[1] = {0};
  qk_circuit_gate(qc, QkGate_H, h_qargs, NULL);
  uint32_t cx_qargs[2] = {0, 1};
  qk_circuit_gate(qc, QkGate_CX, cx_qargs, NULL);
  for (uint32_t i = 0; i < 2; i++) {
    qk_circuit_measure(qc, i, i);
  }
  qk_circuit_free(qc);
  return 0;
}
```

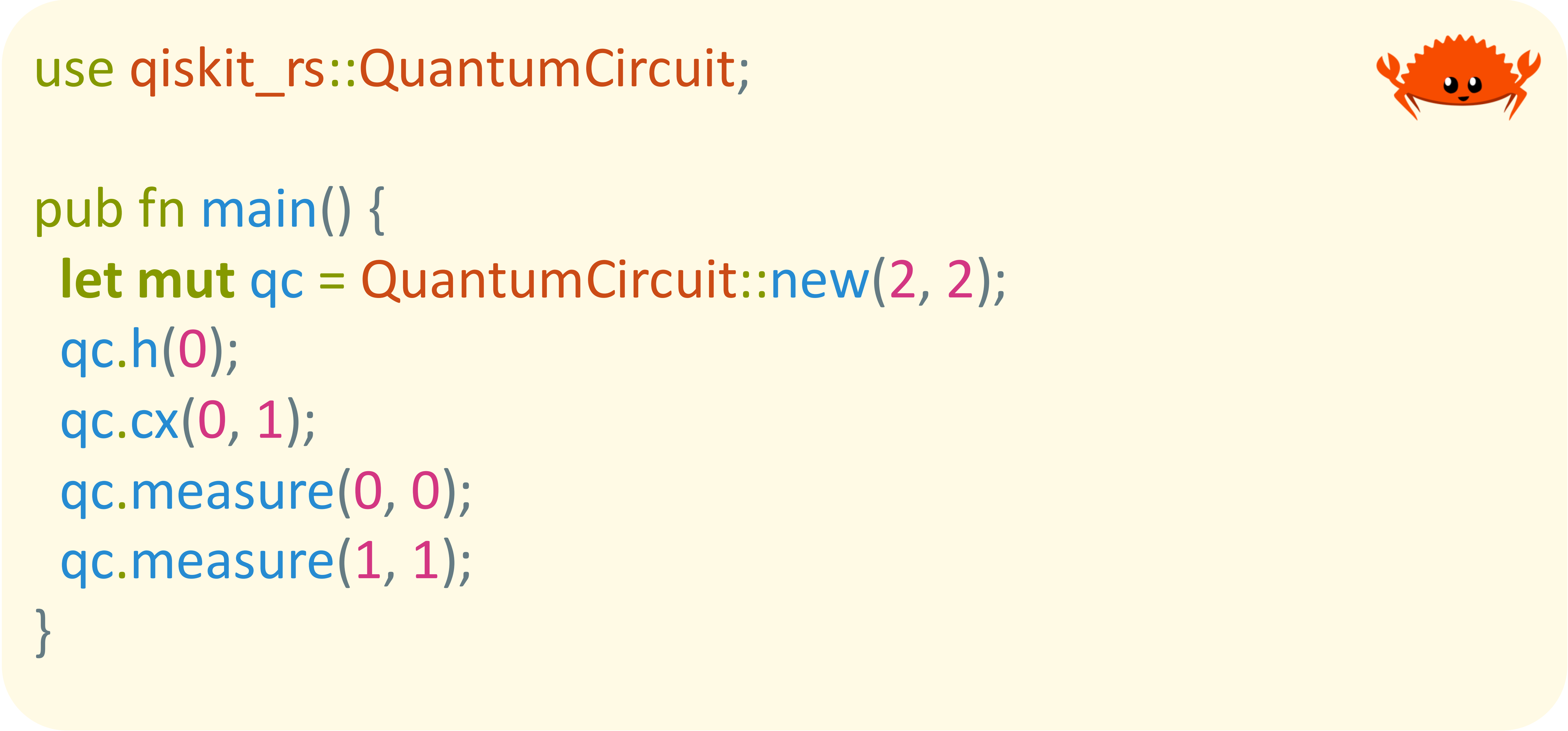
Native Language bindings to Qiskit C API

- C++
  - https://github.com/Qiskit/qiskit-cpp

```
#include "circuit/quantumcircuit.hpp"

using namespace Qiskit::circuit;

int main() {
 QuantumRegister qr(2);
 ClassicalRegister cr(2);
 QuantumCircuit circ(qr, cr);
 circ.h(0);
 circ.cx(0, 1);
 circ.measure(0, 0);
 circ.measure(1, 1);

 return 0;
}
```

Native Language bindings to Qiskit C API

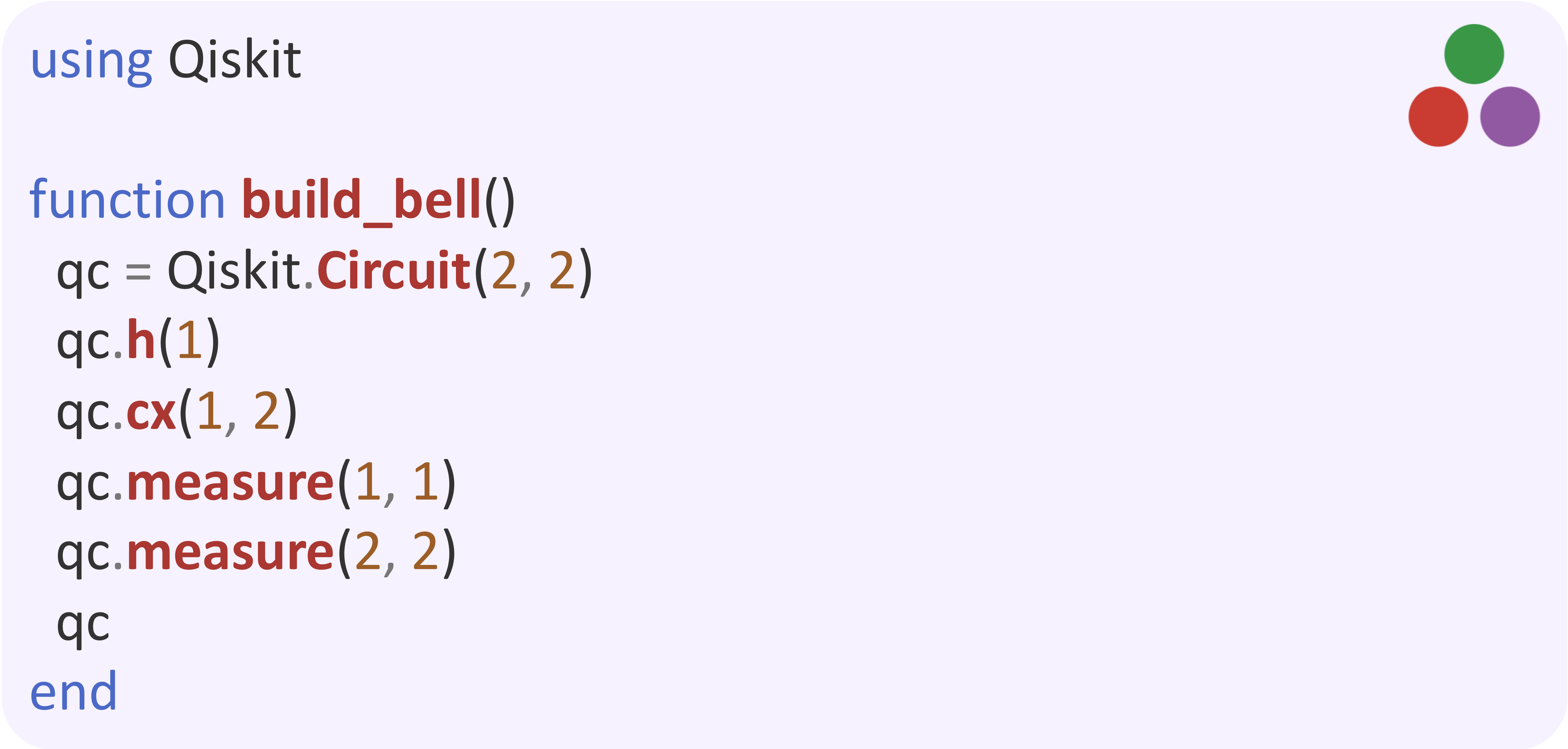
- C++
  - https://github.com/Qiskit/qiskit-cpp
- Rust
  - https://github.com/Qiskit/qiskit-rs



```rust
use qiskit_rs::QuantumCircuit;

pub fn main() {
  let mut qc = QuantumCircuit::new(2, 2);
  qc.h(0);
  qc.cx(0, 1);
  qc.measure(0, 0);
  qc.measure(1, 1);
}
```

Native Language bindings to Qiskit C API

- C++
  - https://github.com/Qiskit/qiskit-cpp
- Rust
  - https://github.com/Qiskit/qiskit-rs
- Julia
  - https://github.com/Qiskit/Qiskit.jl/

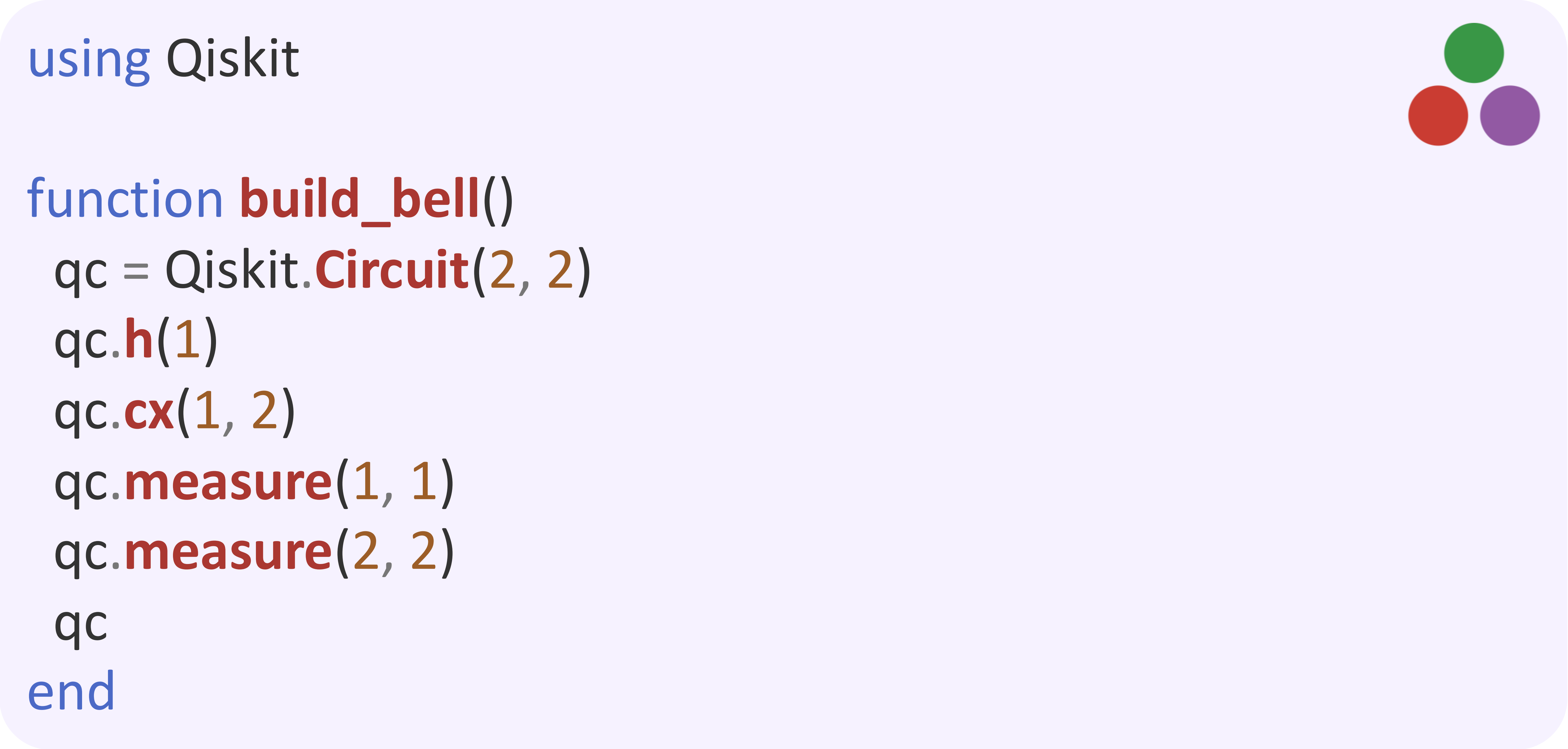


```
using Qiskit

function build_bell()
  qc = Qiskit.Circuit(2, 2)
  qc.h(1)
  qc.cx(1, 2)
  qc.measure(1, 1)
  qc.measure(2, 2)
  qc
end
```

Native Language bindings to Qiskit C API

- C++
  - https://github.com/Qiskit/qiskit-cpp
- Rust
  - https://github.com/Qiskit/qiskit-rs
- Julia
  - https://github.com/Qiskit/Qiskit.jl/

Insert Your Language of choice here



```
using Qiskit

function build_bell()
  qc = Qiskit.Circuit(2, 2)
  qc.h(1)
  qc.cx(1, 2)
  qc.measure(1, 1)
  qc.measure(2, 2)
  qc
end
```

Qiskit C API demo for 2.2

- Demonstrate SQD workflow described in:

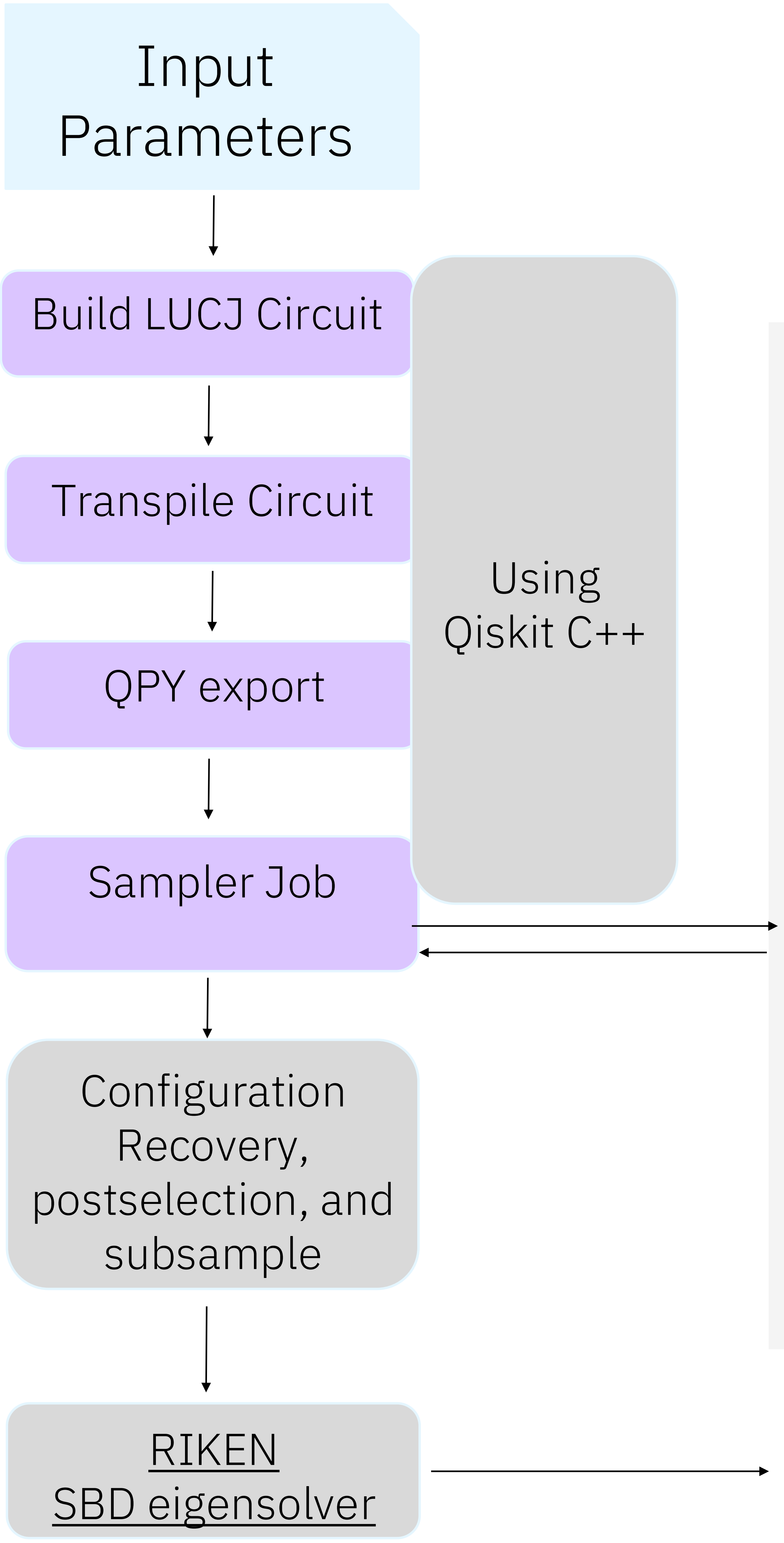
Javier Robledo-Moreno *et al.*, Chemistry beyond the scale of exact diagonalization on a quantum-centric supercomputer.*Sci. Adv.***11**,eadu9991(2025).DOI:10.1126/sciadv.adu9991

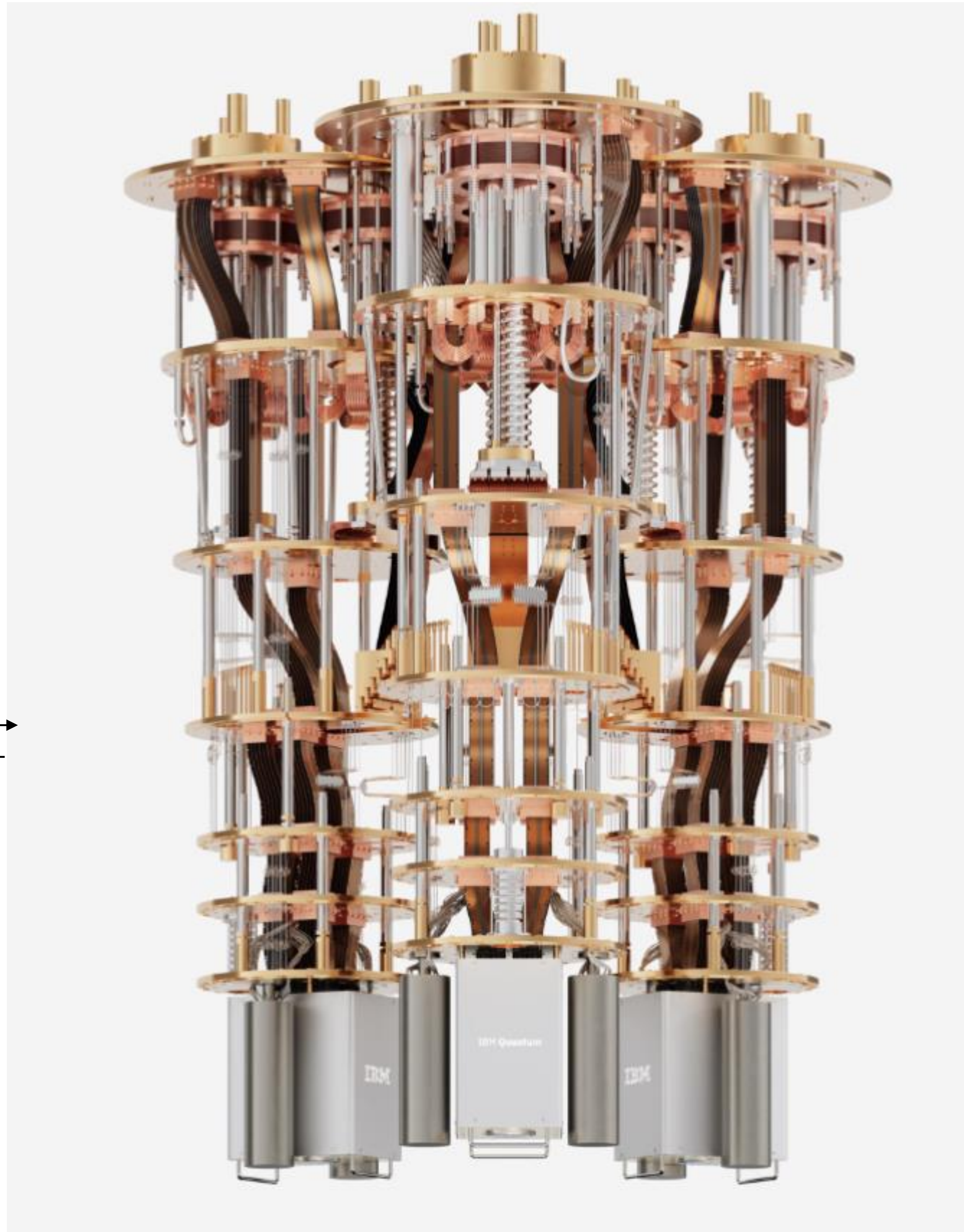
- A single binary application that runs classical side as an MPI workload on a cluster and submits jobs to quantum system
- Leverage Qiskit C++ interface to use one language for entire Quantum + HPC workflow
- Demo code can be found here: https://github.com/qiskit-community/qiskit-c-api-demo

I/O

C API (Qiskit or IBM Quantum Platform C client)

C++ (either Qiskit-C++, SQD HPC addon, or an external lib)





Ground state energy

Qiskit 2.0

March 31, 2025

Release notes:
https://docs.quantum.ibm.com/api/qiskit/release-notes/2.0

- **Introduce C FFI as second public interface to Qiskit**
- **Build infrastructure and packaging for exposing C API**
- **Expose the `SparseObservable` class**

Qiskit 2.1

June 19, 2025

Release notes:
https://docs.quantum.ibm.com/api/qiskit/release-notes/2.1

- **Expand C FFI functionality to include:**
  - **Circuit construction with standard gates**
  - **Target representation**

Qiskit 2.2

September 18, 2025

Release notes:
https://docs.quantum.ibm.com/api/qiskit/release-notes/2.2

- **Expand C FFI to include:**
  - **Transpiler function (to compile circuits for target)**
  - **Standalone transpiler pass functions for most of the passes needed in the above transpile() function**

Qiskit 2.3

December 18, 2025

- **Expand C FFI**
  - **DAGCircuit (compilation IR)**
  - **Building custom transpiler passes from C**

Qiskit 2.4

March 26, 2026

- **Expand C FFI**
  - **Custom gates**
  - **PPM and Pauli Evolution Gate**
  - **Parameterized Gates**

Qiskit 2.5

June, 2026

- **Your important feature**

Backup Slides

LightSabre

- Qiskit has significantly changed and improved version of the algorithm from the original paper: https://arxiv.org/abs/2409.08368
- Original SABRE paper: https://arxiv.org/pdf/1809.02573.pdf
- Introduces multiple trials to improve the quality of the results
- Improves runtime scaling through algorithmic optimization

Docs:
https://docs.quantum.ibm.com/api/qiskit/qiskit.transpiler.passes.SabreLayout