# Reliable Quantum Computing

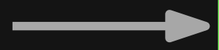
**Michael Pradel** (University of Stuttgart)

**Joint work with Matteo Paltenghi**

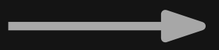


# Big Picture

**Quantum developer**
(software engineer or physicist)

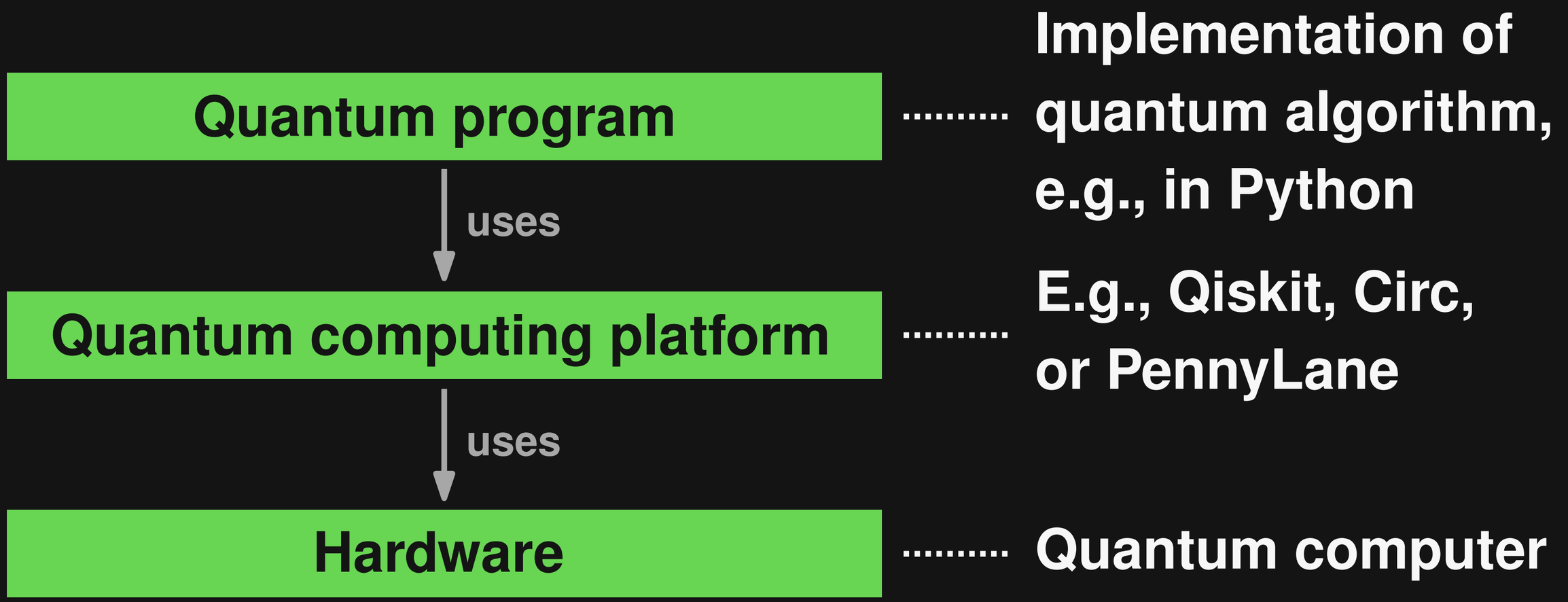


**Quantum computing**

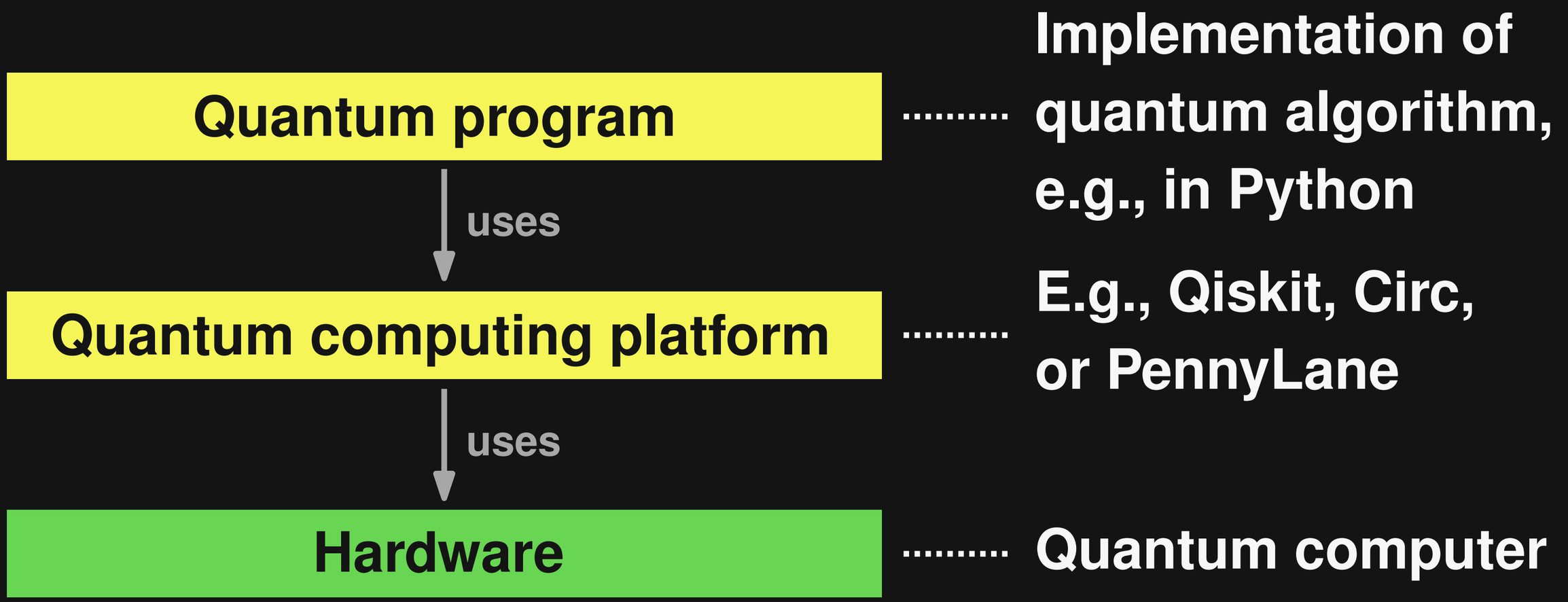


**Computations that seem intractable otherwise**

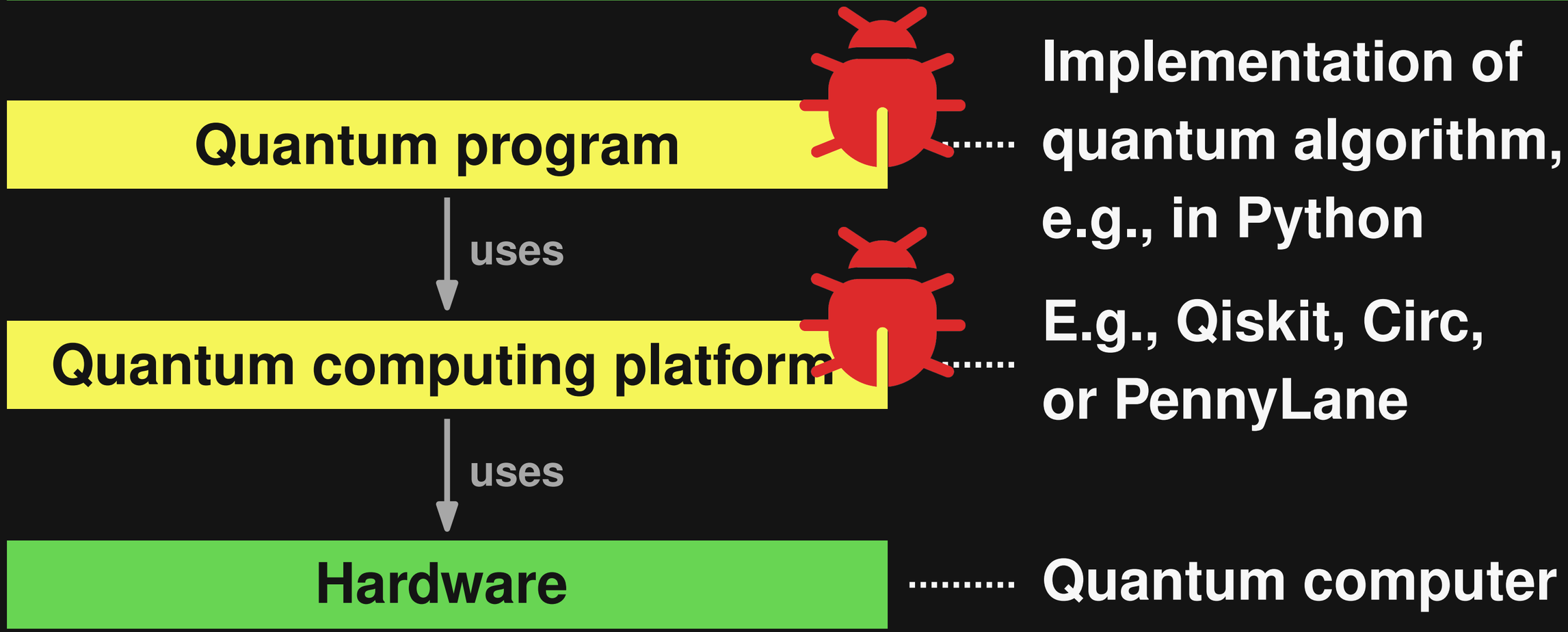
# Quantum Computing Stack



# Quantum Computing Stack



# Quantum Computing Stack




**Usefulness of quantum computing critically depends on the reliability of the software!**

# What Could Go Wrong?

## 1) Example of an application-level bug

```
qc = QuantumCircuit(2, 2)
qc.h(1)
qc.cx(1, 0)
qc.measure(0, 0)
qc.measure(1, 1)
qc.z(0)
qc.measure(0, 0)
```

# What Could Go Wrong?

## 1) Example of an application-level bug

```
qc = QuantumCircuit(2, 2)
qc.h(1)
qc.cx(1, 0)
qc.measure(0, 0)
qc.measure(1, 1)
qc.z(0)
qc.measure(0, 0)
```

**Problem: Qubit 0 has collapsed due to measurement**

# What Could Go Wrong?

## 2) Example of a platform-level bug

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

subcircuit = QuantumCircuit(qr, cr)
subcircuit.x(qr[0])

qc.append(subcircuit, qargs=qr, cargs=cr)

qasm_str = qc.qasm()
qc = QuantumCircuit.from_qasm_str(qasm_str)
```

# What Could Go Wrong?

## 2) Example of a platform-level bug

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

subcircuit = QuantumCircuit(qr, cr)
subcircuit.x(qr[0])

qc.append(subcircuit, qargs=qr, cargs=cr)

qasm_str = qc.qasm()
qc = QuantumCircuit.from_qasm_str(qasm_str)
```

Create a circuit

# What Could Go Wrong?

## 2) Example of a platform-level bug

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

subcircuit = QuantumCircuit(qr, cr)
subcircuit.x(qr[0])

qc.append(subcircuit, qargs=qr, cargs=cr)

qasm_str = qc.qasm()
qc = QuantumCircuit.from_qasm_str(qasm_str)
```

Create a circuit

Create a subcircuit and add to main circuit

# What Could Go Wrong?

## 2) Example of a platform-level bug

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

subcircuit = QuantumCircuit(qr, cr)
subcircuit.x(qr[0])

qc.append(subcircuit, qargs=qr, cargs=cr)

qasm_str = qc.qasm()
qc = QuantumCircuit.from_qasm_str(qasm_str)
```

Create a circuit

Create a subcircuit and add to main circuit

Export to QASM and import it again

# What Could Go Wrong?

## 2) Example of a platform-level bug

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

subcircuit = QuantumCircuit(qr, cr)
subcircuit.x(qr[0])

qc.append(subcircuit, qargs=qr, cargs=cr)

qasm_str = qc.qasm()
qc = QuantumCircuit.from_qasm_str(qasm_str)
```

**Problem: Qiskit crashes here due to bug in QASM exporter**

# Goal

- **Make quantum computing more reliable**
- **Automatic techniques for finding bugs**
  - in quantum applications
  - in quantum computing platforms

# This Talk

1) **LintQ**

- Static analysis framework for quantum programs

2) **MorphQ**

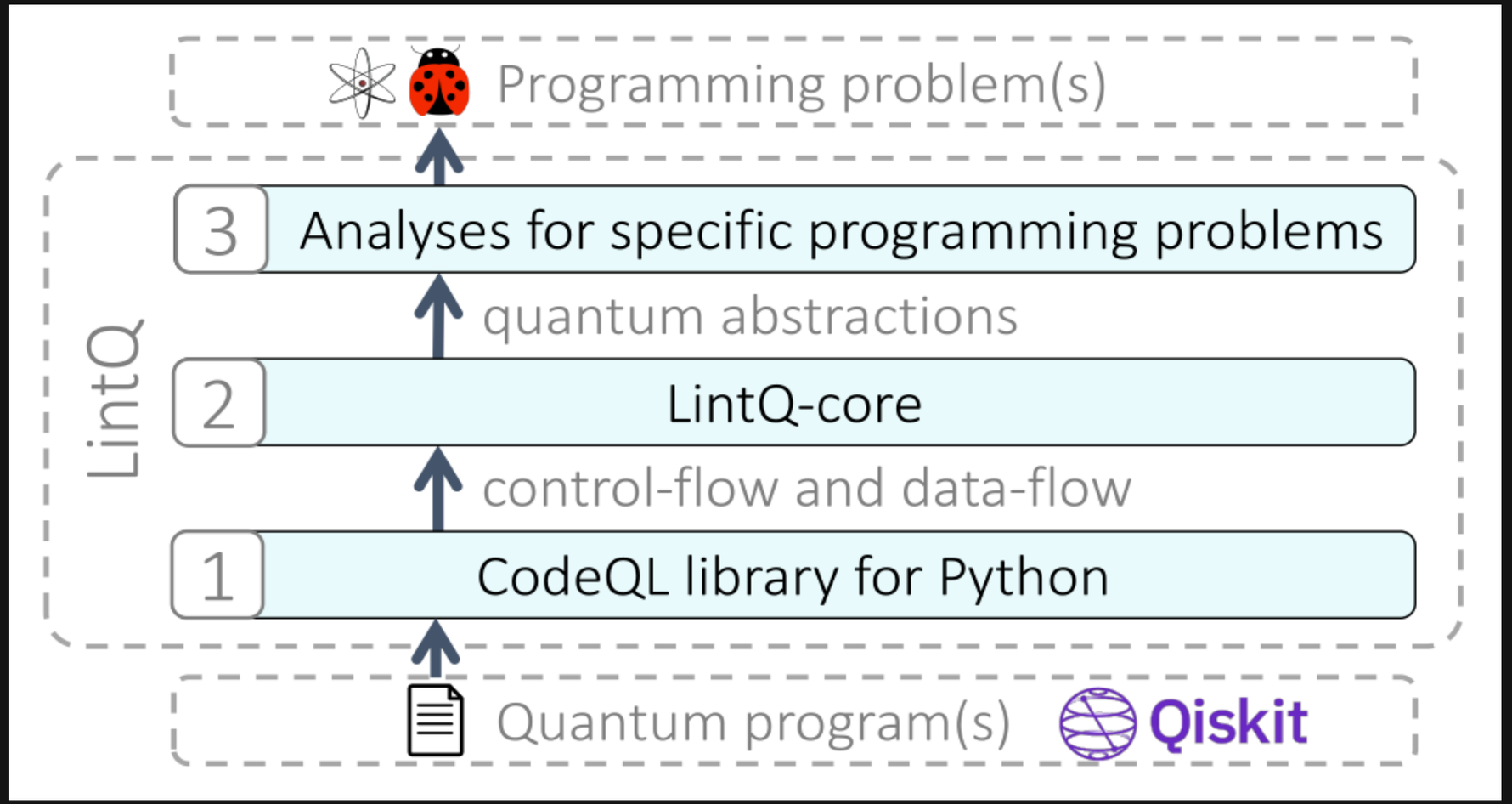
- Metamorphic testing of quantum computing platforms

# This Talk

1) **LintQ**
   - Static analysis framework for quantum programs

2) **MorphQ**
   - Metamorphic testing of quantum computing platforms

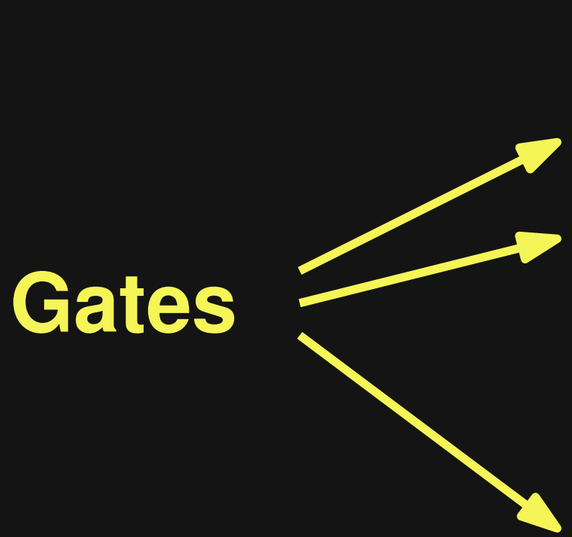
# Overview of LintQ



# Example: Program and Its Abstraction

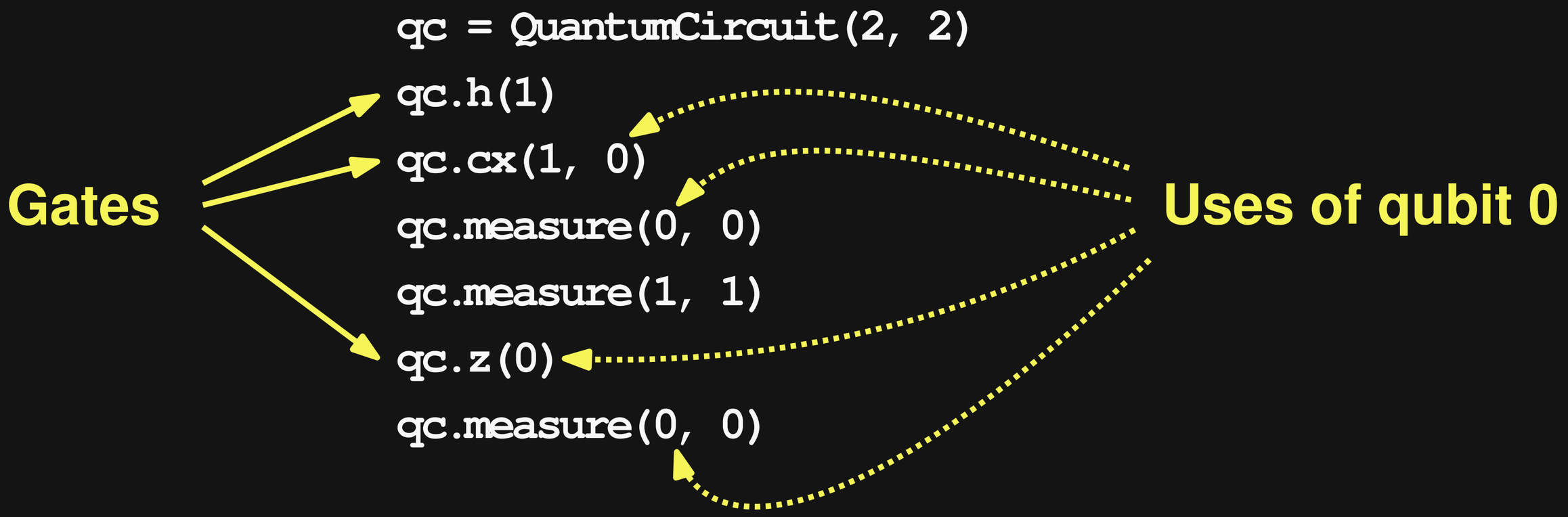
```
qc = QuantumCircuit(2, 2)
qc.h(1)
qc.cx(1, 0)
qc.measure(0, 0)
qc.measure(1, 1)
qc.z(0)
qc.measure(0, 0)
```

# Example: Program and Its Abstraction




```
qc = QuantumCircuit(2, 2)
qc.h(1)
qc.cx(1, 0)
qc.measure(0, 0)
qc.measure(1, 1)
qc.z(0)
qc.measure(0, 0)
```

# Example: Program and Its Abstraction



# Analyses to Find Common Bugs

| Analysis name | Description |
|---|---|
| *Measurement-related and gate-related problems:* | |
| DoubleMeas | Two measurements measure the same qubit state one after the other. |
| OpAfterMeas | A gate operates on a qubit after it has been measured. |
| MeasAllAbuse | Measurement results are stored in an implicitly created new register, even though another classical register already exists. |
| CondWoMeas | Conditional gate without measurement of the associated register. |
| ConstClasBit | A qubit is measured but has not been transformed. |
| *Resource allocation problems:* | |
| InsuffClasReg | Classical bits do not suffice to measure all qubits. |
| OversizedCircuit | The quantum register contains unused qubits. |
| *Implicit API constraints:* | |
| GhostCompose | Composing two circuits without using the resulting composed circuit. |
| OpAfterOpt | A gate is added after transpilation. |
| OldIdenGate | Using a now-removed API to create an identity gate. |

# Analyses to Find Common Bugs

| Analysis name | Description |
|---|---|
| *Measurement-related and gate-related problems:* | |
| DoubleMeas | Two measurements measure the same qubit state one after the other. |
| OpAfterMeas | A gate operates on a qubit after it has been measured. |
| MeasAllAbuse | Measurement results are stored in an implicitly created new register, even though another classical register already exists. |
| CondWoMeas | Conditional gate without measurement of the associated register. |
| ConstClasBit | A qubit is measured but has not been transformed. |
| *Resource allocation problems:* | |
| InsuffClasReg | Classical bits do not suffice to measure all qubits. |
| OversizedCircuit | The quantum register contains unused qubits. |
| *Implicit API constraints:* | |
| GhostCompose | Composing two circuits without using the resulting composed circuit. |
| OpAfterOpt | A gate is added after transpilation. |
| OldIdenGate | Using a now-removed API to create an identity gate. |

# Example of Analysis

Check for operations after measurement:

```
from Measurement m, Gate g, int q
where
  mayFollowDirectly(m, g, q)
  and not g.isConditional()
select gate , "Gate after measurement on qubit" + q
```

# Example of Analysis

Check for operations after measurement:

Specifies the involved abstractions

```
from Measurement m, Gate g, int q
where
  mayFollowDirectly(m, g, q)
  and not g.isConditional()
select gate , "Gate after measurement on qubit" + q
```

# Example of Analysis

Check for operations after measurement:

```
from Measurement m, Gate g, int q
where
  mayFollowDirectly(m, g, q)
  and not g.isConditional()
select gate , "Gate after measurement on qubit" + q
```

Specifies the involved abstractions

Specifies how the abstractions are related

# Example of Analysis

Check for operations after measurement:

```
from Measurement m, Gate g, int q
where
  mayFollowDirectly(m, g, q)
  and not g.isConditional()
select gate , "Gate after measurement on qubit" + q
```

Specifies the involved abstractions

Specifies how the abstractions are related

Produces a warning message

# Example of Analysis

Check for operations after measurement:

```
from Measurement m, Gate g, int q
where
  mayFollowDirectly(m, g, q)
  and not g.isConditional()
select gate , "Gate after measurement on qubit" + q
```

Specifies the involved abstractions

Specifies how the abstractions are related

Produces a warning message

Nb. of lines to specify an analysis (avg.): 10

# Results

- Applied to 7,568 quantum programs gathered from GitHub
- LintQ reports 4,699 warnings
- Manually inspected representative sample of 345: 216 true positives, i.e., bugs that should be fixed
- Time to analyze a program: 1.3 seconds (avg.)

# This Talk

1) **LintQ**
   - Static analysis framework for quantum programs

2) **MorphQ**
   - Metamorphic testing of quantum computing platforms

# This Talk

1) **LintQ**

- Static analysis framework for quantum programs

2) **MorphQ**

- Metamorphic testing of quantum computing platforms

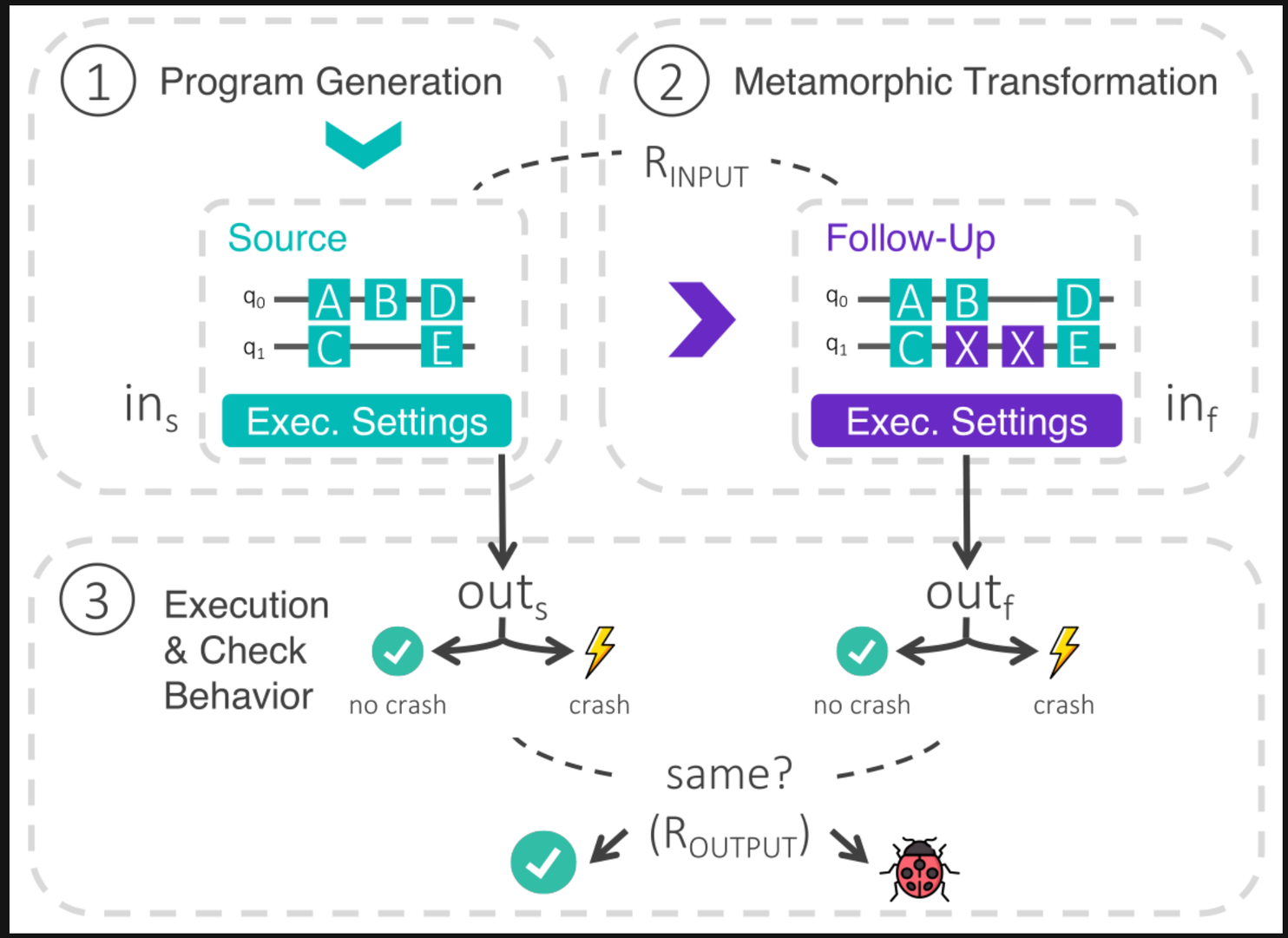
# Key Challenges

Challenges for automatically finding bugs in quantum computing platforms:

1) **Few quantum programs**

2) **Oracle problem**

# Overview of MorphQ

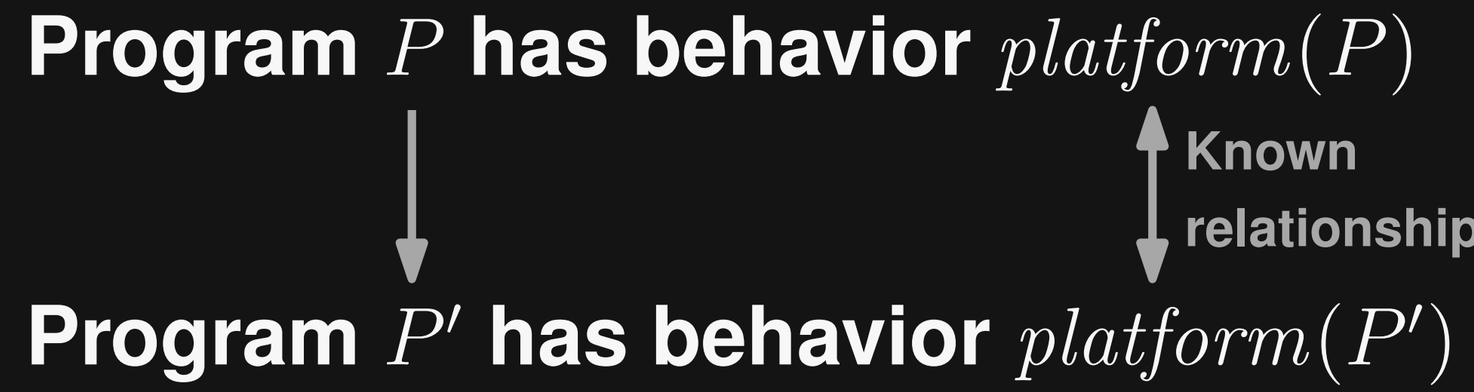


# Program Generation

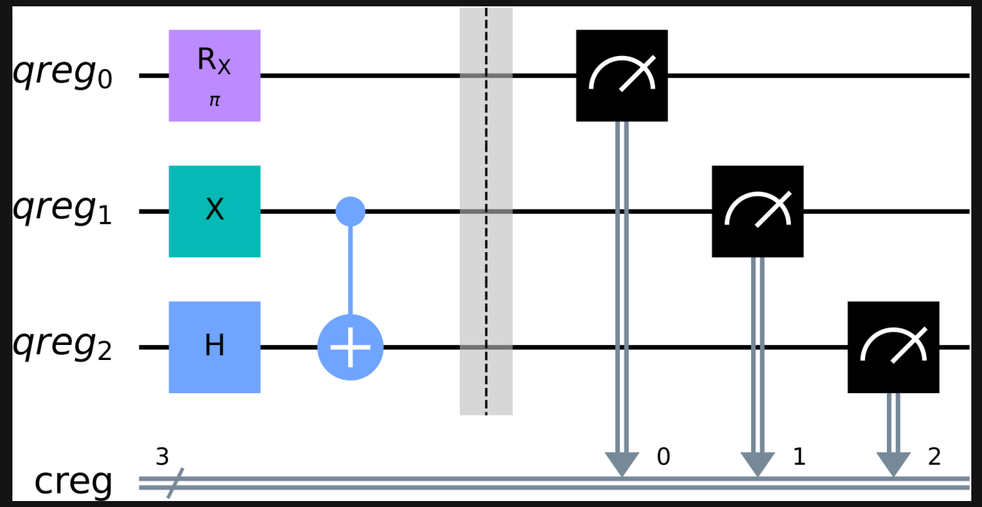
- **Naive approach: Randomly combine calls of Qiskit APIs**
  - Problem: Mostly invalid programs (crashes, no deep testing)
- **Instead: Combine template-based and grammar-based generation**
  - Generates non-crashing programs by design

# Metamorphic Testing

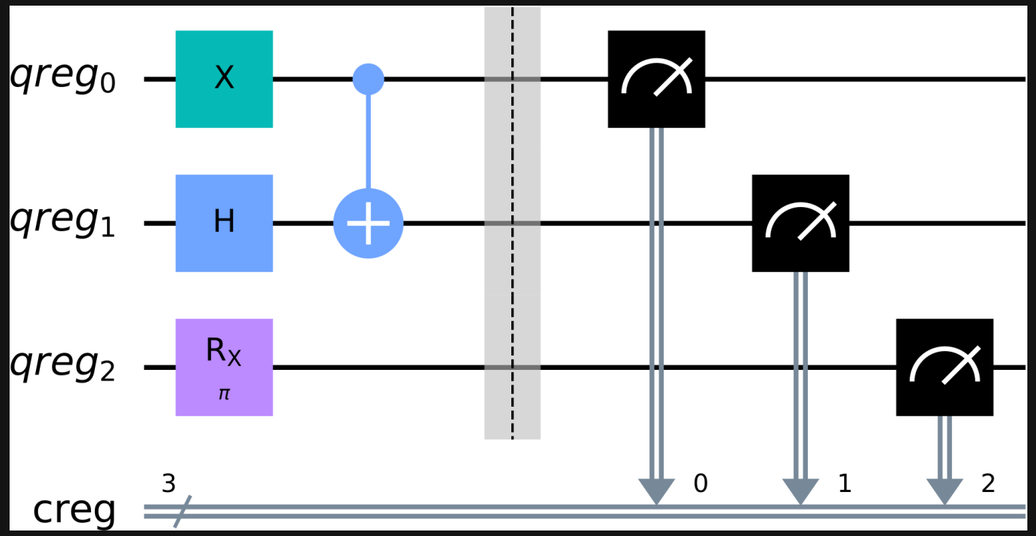
- **How to know whether the platform executes a program correctly?**
- **Metamorphic transformations**
  - Modify program in a way where we know how its behavior should change



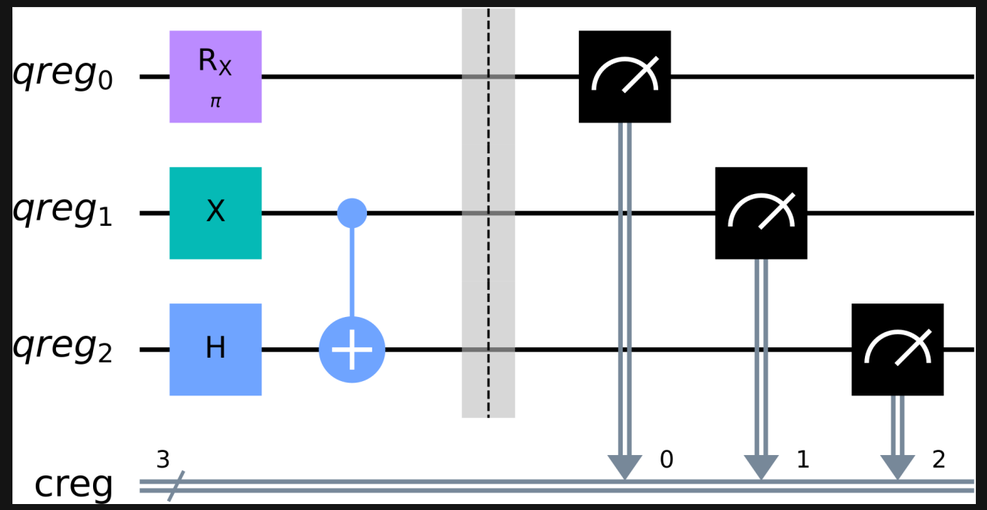
# Example: Change Qubit Order



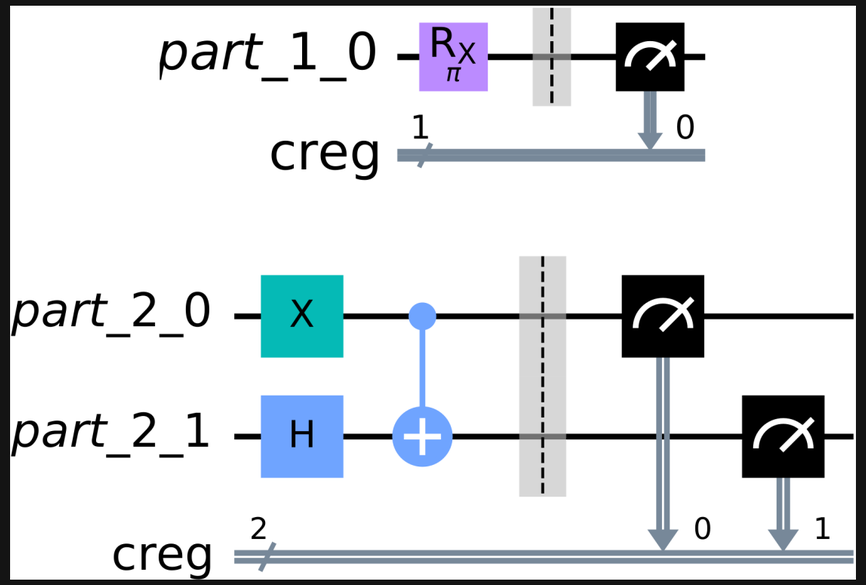

**Reorder qubits, and then adjust gates and measurements**



# Example: Partitioning the Computation




**Separate into subcircuits and combine output distributions**



# Results

- **48 hours of automated testing of Qiskit**
  - 8,360 tested program pairs
  - 1,943 crashes in follow-up program
  - 56 differences in output distributions

**Automated clustering, de-duplication, and minimization**

- **13 bugs reported to Qiskit developers**
  - All 13 confirmed and/or fixed

# Conclusion

## Reliable software:
## Crucial for success of quantum computing

MorphQ: Metamorphic Testing of the Qiskit Quantum Computing Platform

Matteo Paltenghi
*Department of Computer Science*
*University of Stuttgart*
Stuttgart, Germany
mattepalte@live.it

Michael Pradel
*Department of Computer Science*
*University of Stuttgart*
Stuttgart, Germany
michael@binaervarianz.de

IEEE/ACM International Conference on Software Engineering (ICSE) 2023

Checkout our newest idea:
Poster by Matteo

**Analyzing Quantum Programs with LintQ: A Static Analysis Framework for Qiskit**

MATTEO PALTENGHI, University of Stuttgart, Germany
MICHAEL PRADEL, University of Stuttgart, Germany

ACM International Conference on the Foundations of Software Engineering (FSE) 2024