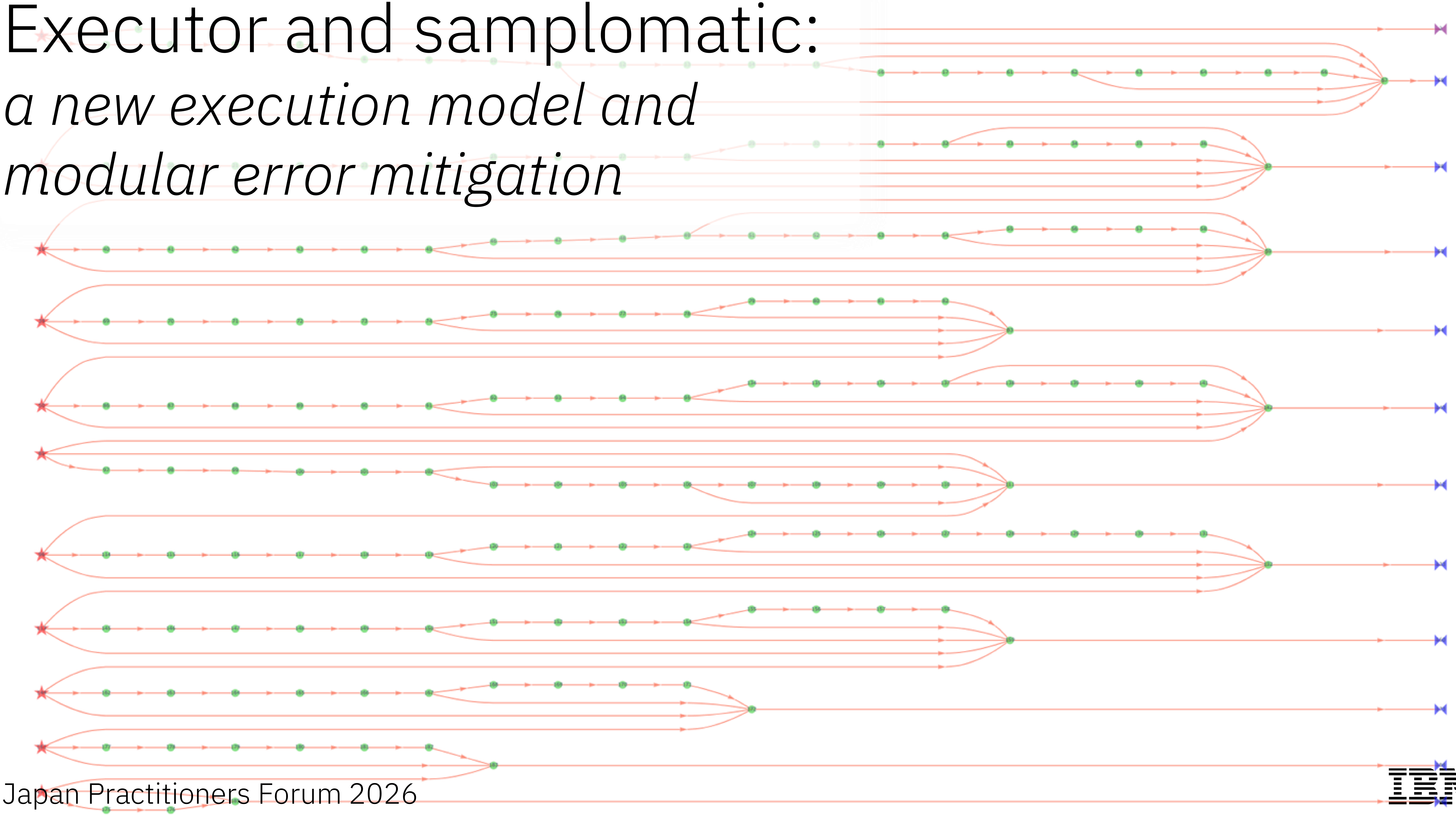
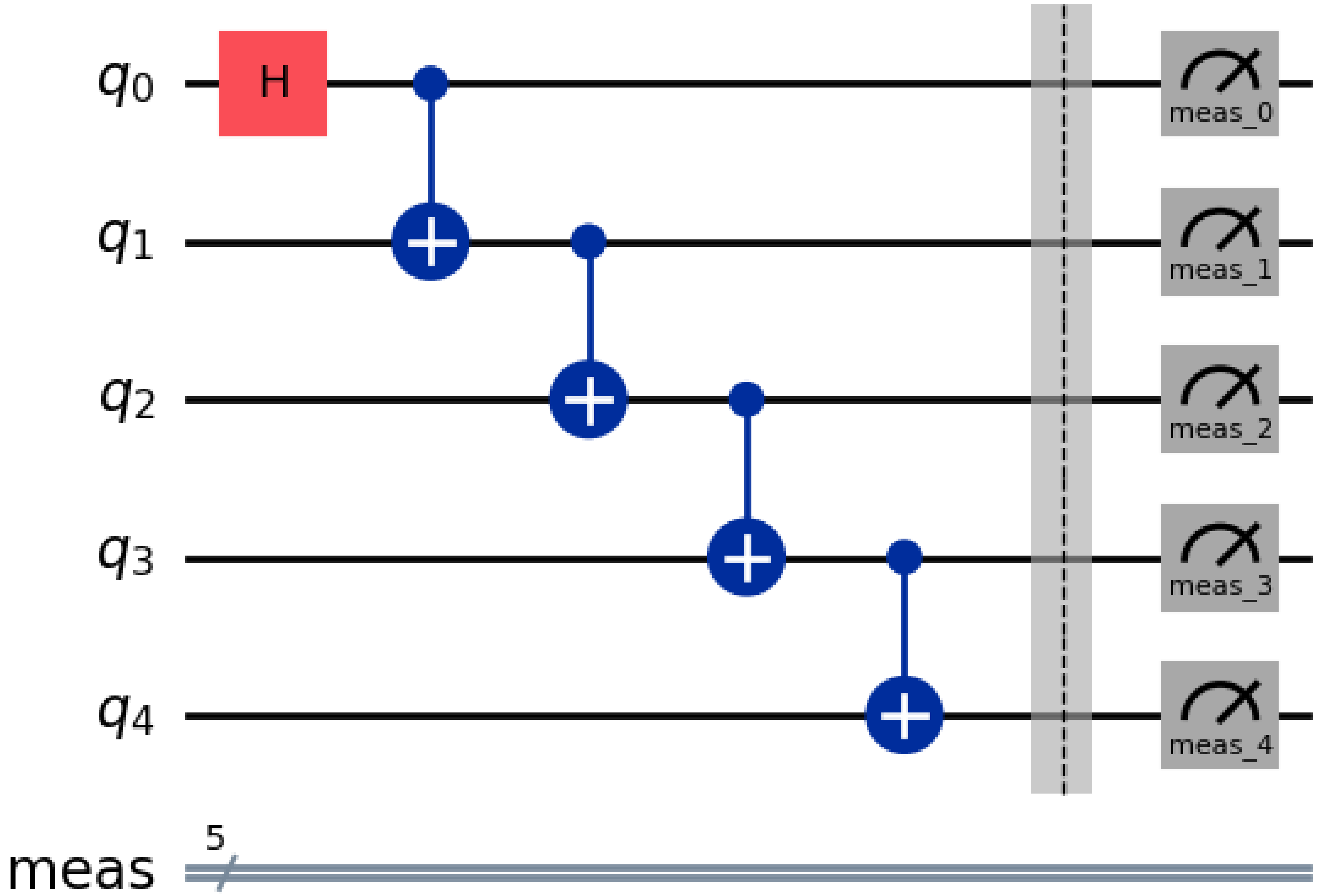
Executor and samplomatic:

a new execution model and modular error mitigation

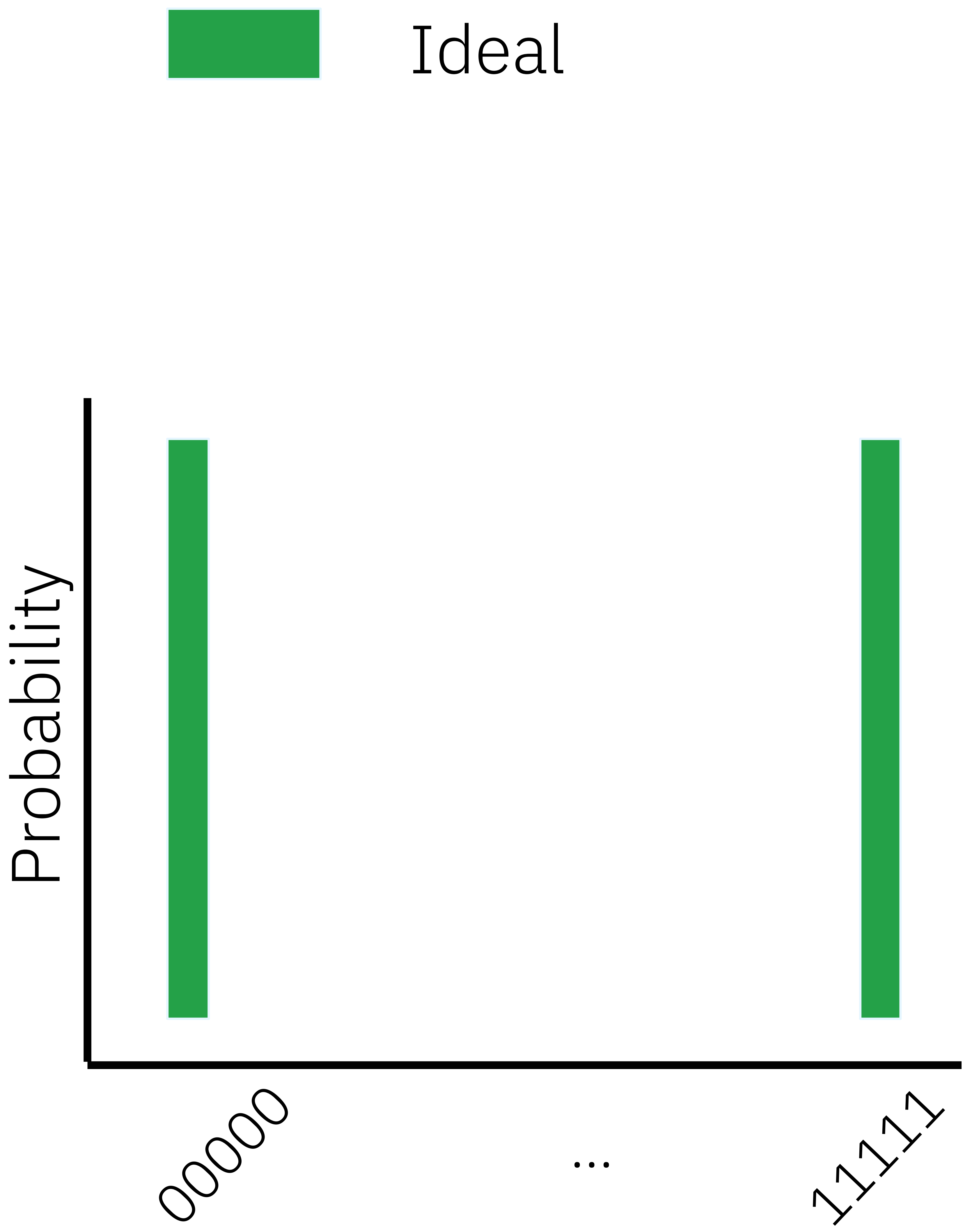


Japan Practitioners Forum 2026

Pre-fault tolerance devices are noisy, and noise can interact with circuits in complicated ways.

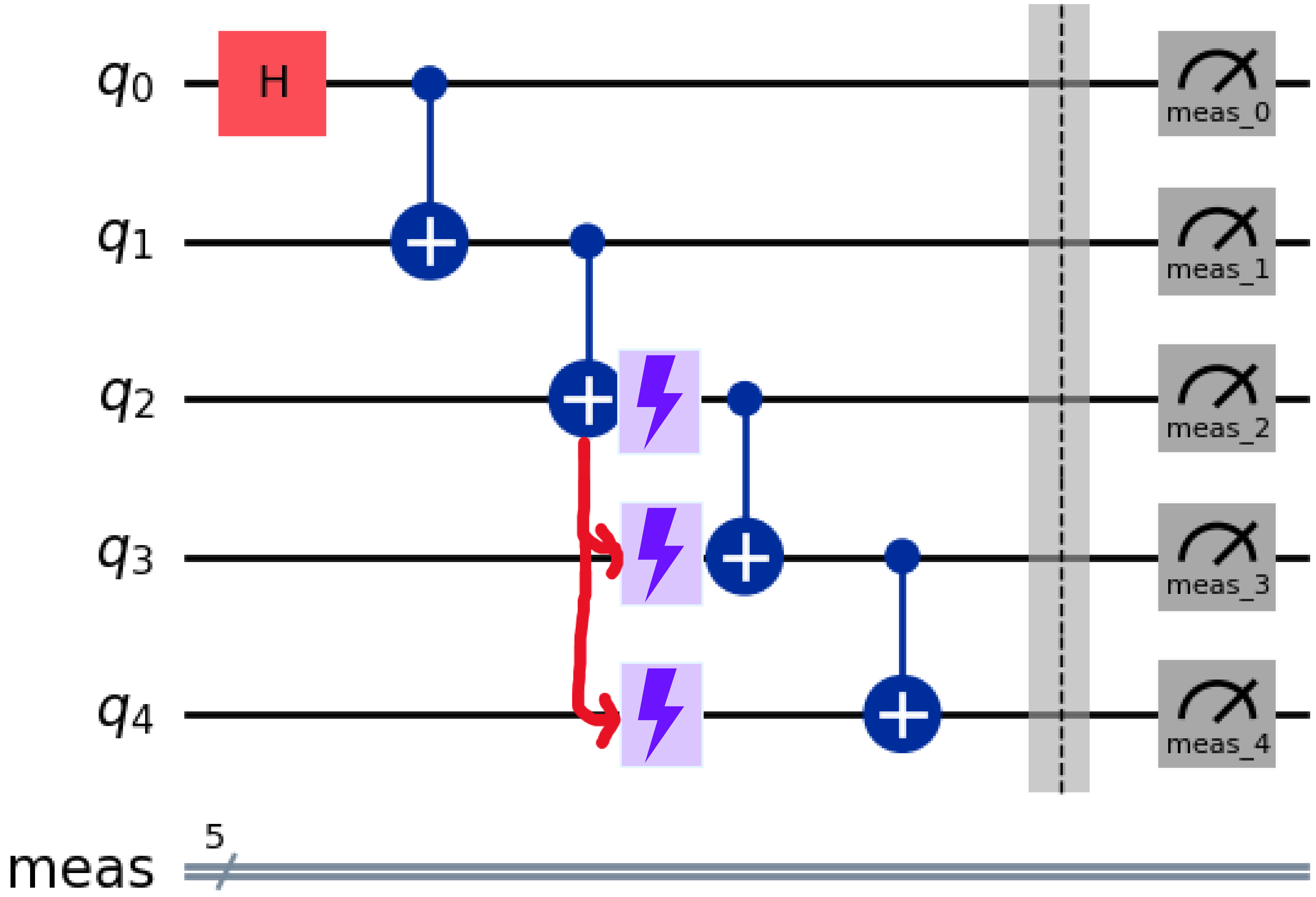


Application Circuit

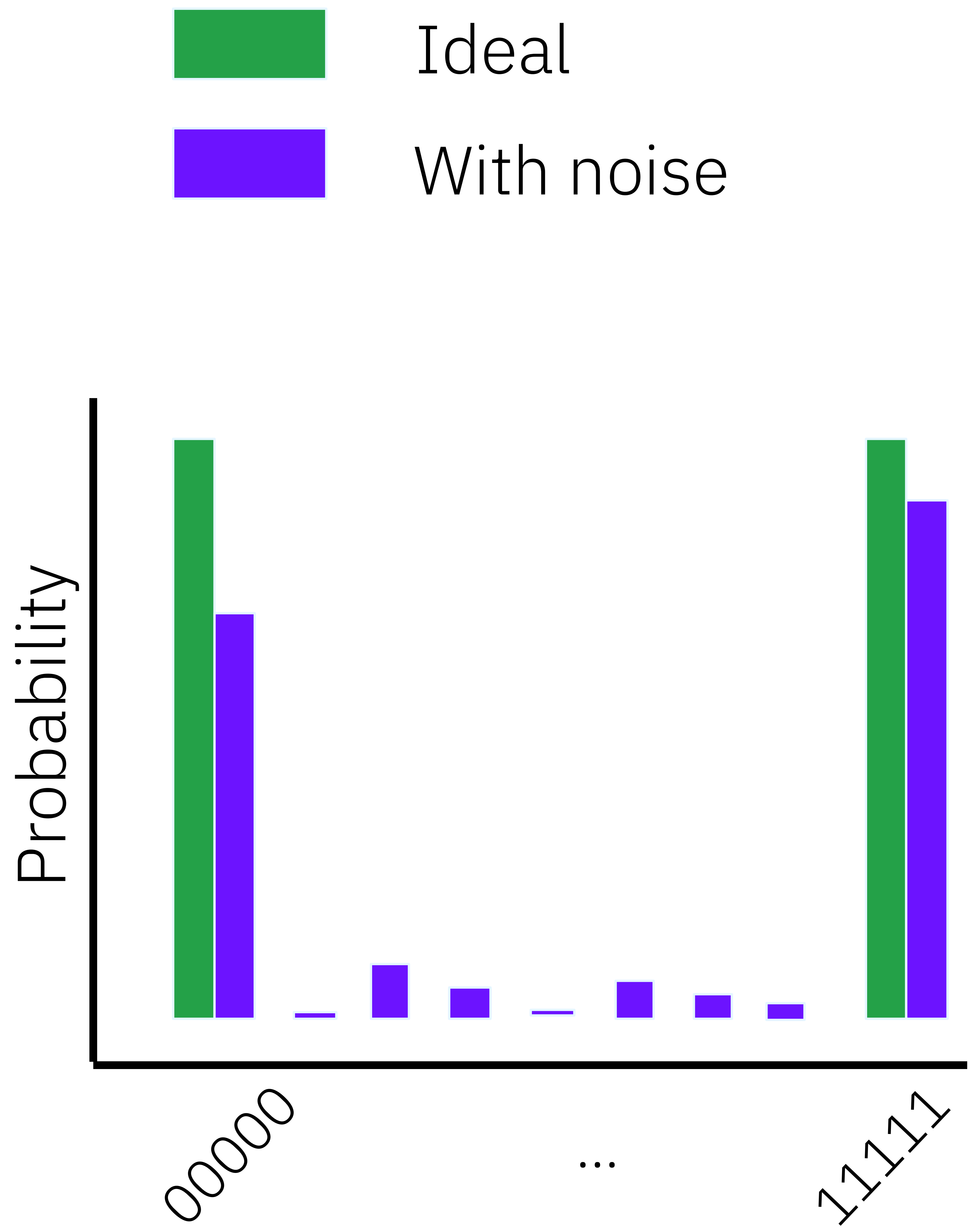


QPU Data

Pre-fault tolerance devices are noisy, and noise can interact with circuits in complicated ways.



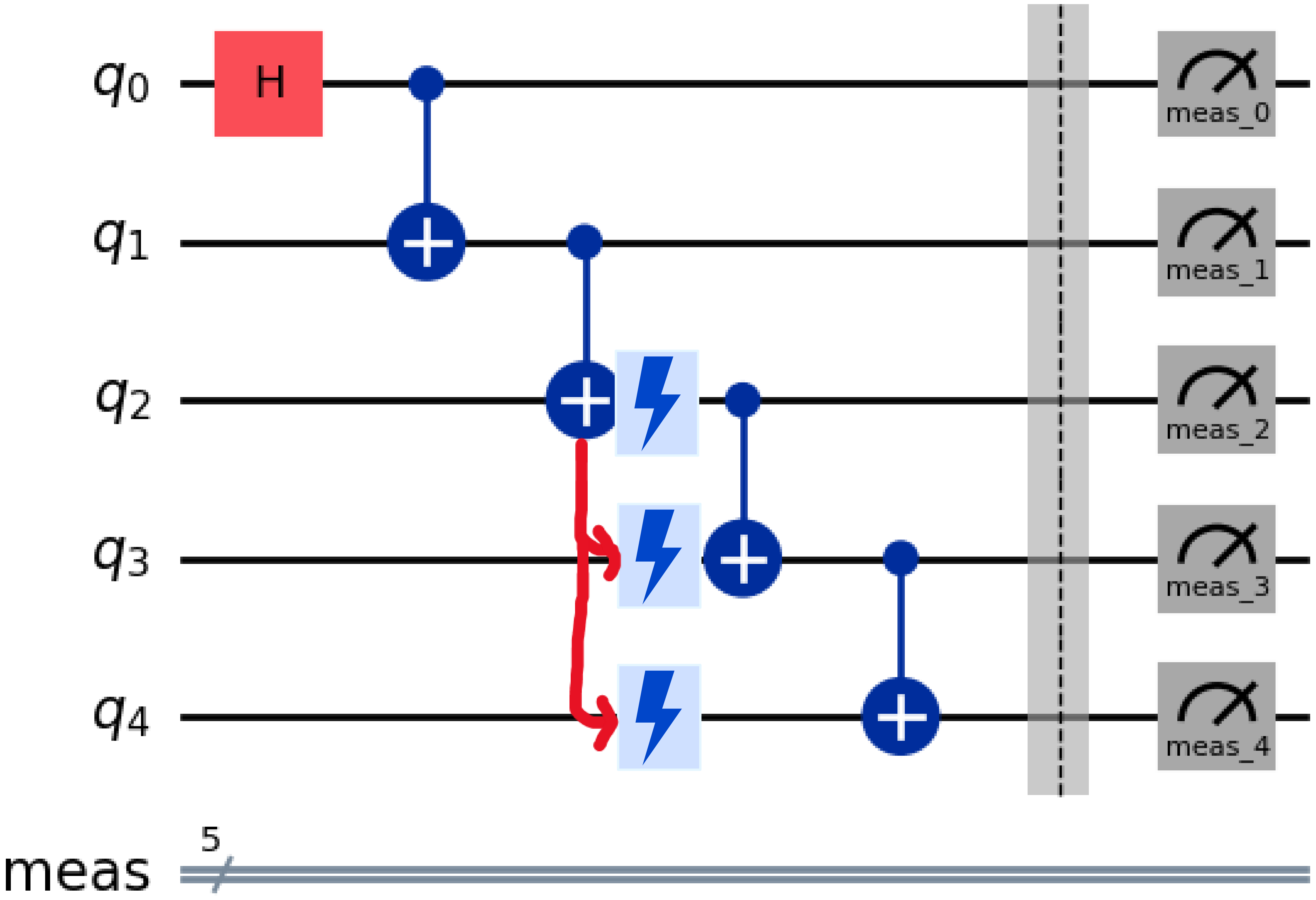
Application Circuit



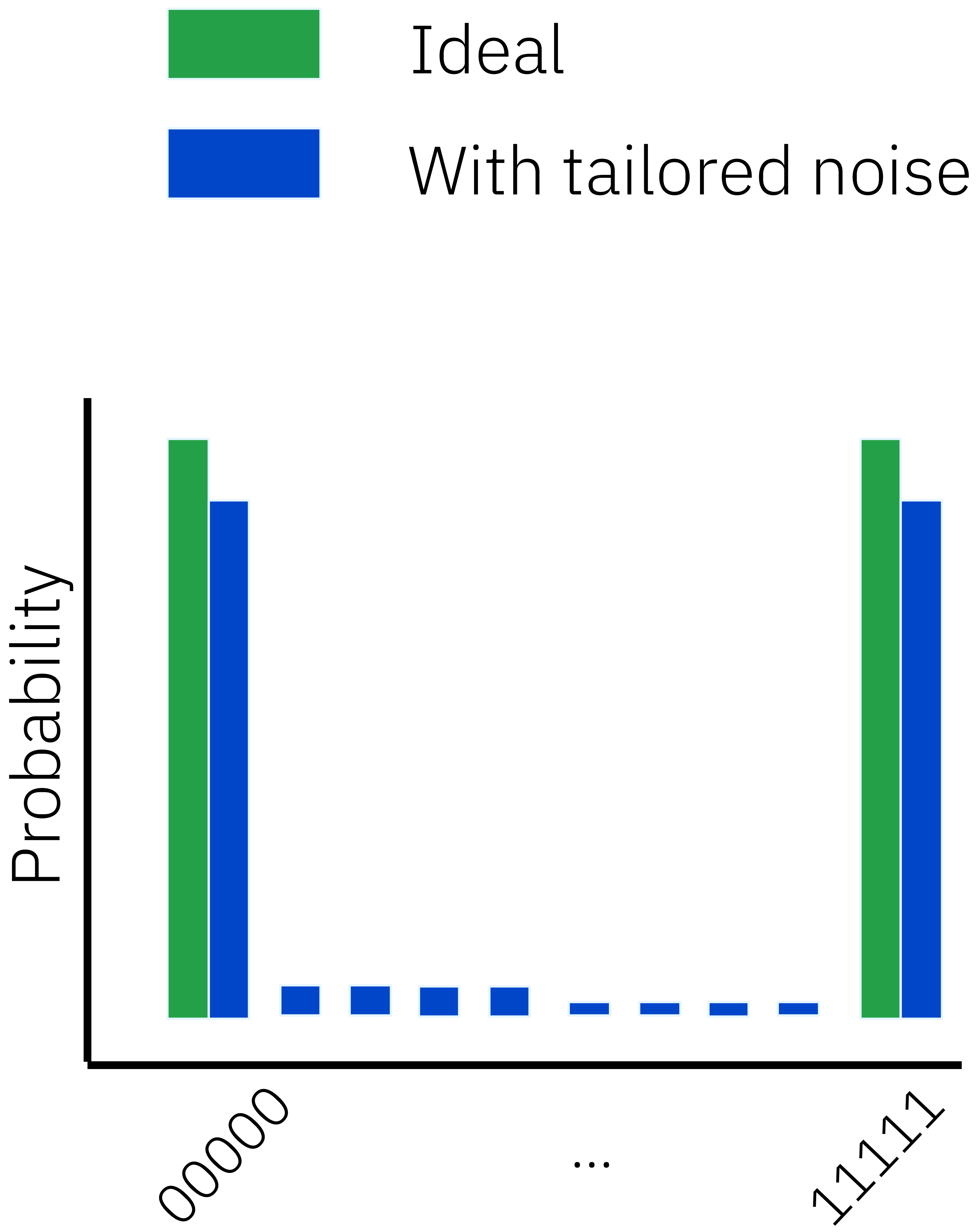
QPU Data

If we can’t eliminate the noise, can we simplify it?

- easier to learn and profile
- easier to mitigate



Application Circuit



QPU Data

Benefits of circuit randomization

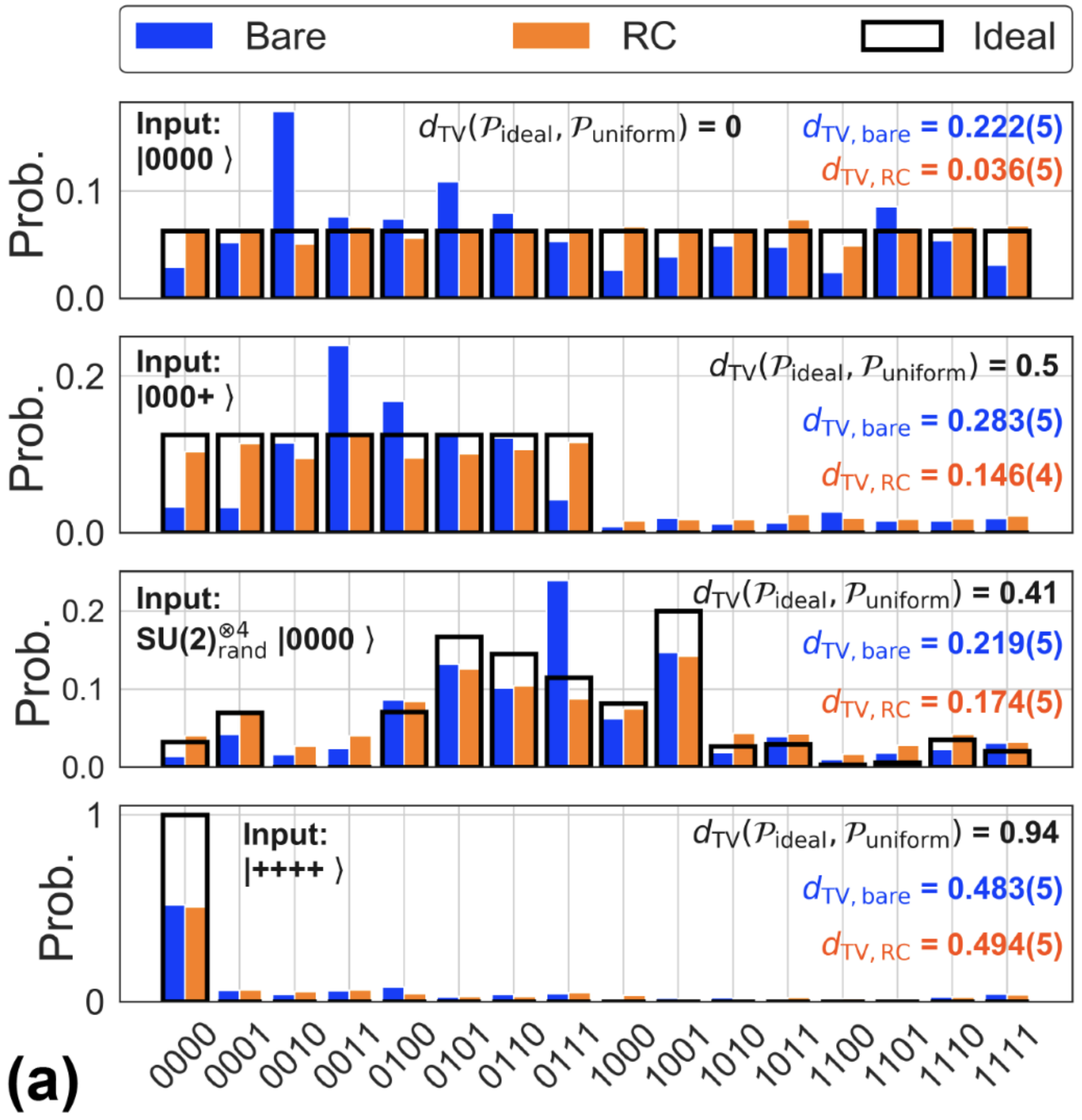
Pauli Twirling / Randomized Compiling

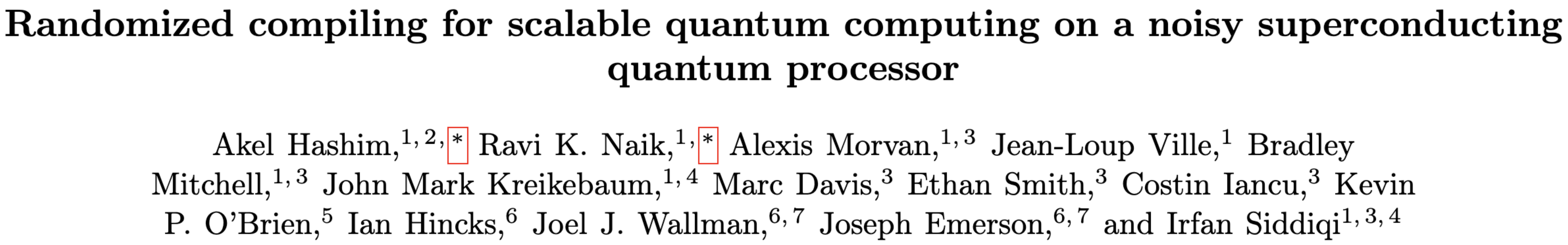
- Remove coherent parts of noise
    - Stochastic noise is more predictable
    - Coherent noise can build up faster
    - Twirled noise models are easier to learn and reason about (RB and friends)

Mitigation

- Noise injection from known distributions is a powerful mitigation technique
    - PEA -> amplify noise by known amounts to extrapolate back to 0-noise
    - PEC -> inject anti-noise channels

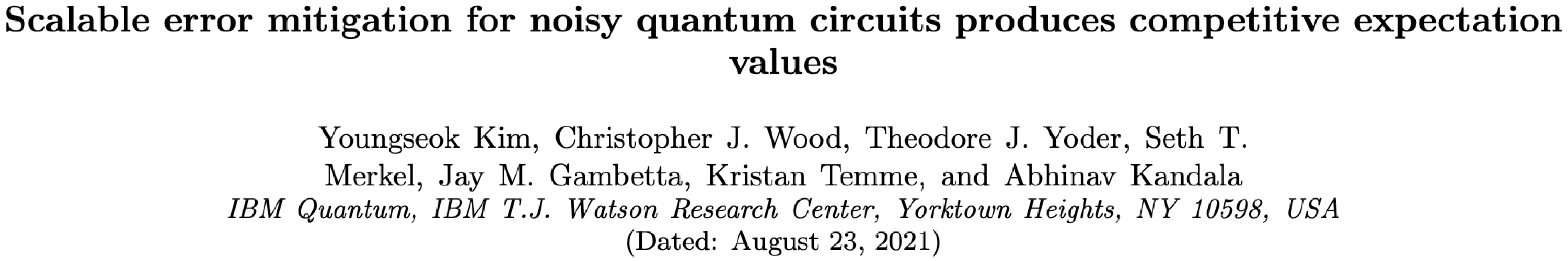
For example...





Randomized compiling for scalable quantum computing on a noisy superconducting quantum processor

Akel Hashim,[1,2,*] Ravi K. Naik,[1,*] Alexis Morvan,[1,3] Jean-Loup Ville,[1] Bradley Mitchell,[1,3] John Mark Kreikebaum,[1,4] Marc Davis,[3] Ethan Smith,[3] Costin Iancu,[3] Kevin P. O'Brien,[5] Ian Hincks,[6] Joel J. Wallman,[6,7] Joseph Emerson,[6,7] and Irfan Siddiqi[1,3,4]



Scalable error mitigation for noisy quantum circuits produces competitive expectation values

Youngseok Kim, Christopher J. Wood, Theodore J. Yoder, Seth T. Merkel, Jay M. Gambetta, Kristan Temme, and Abhinav Kandala
IBM Quantum, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
(Dated: August 23, 2021)

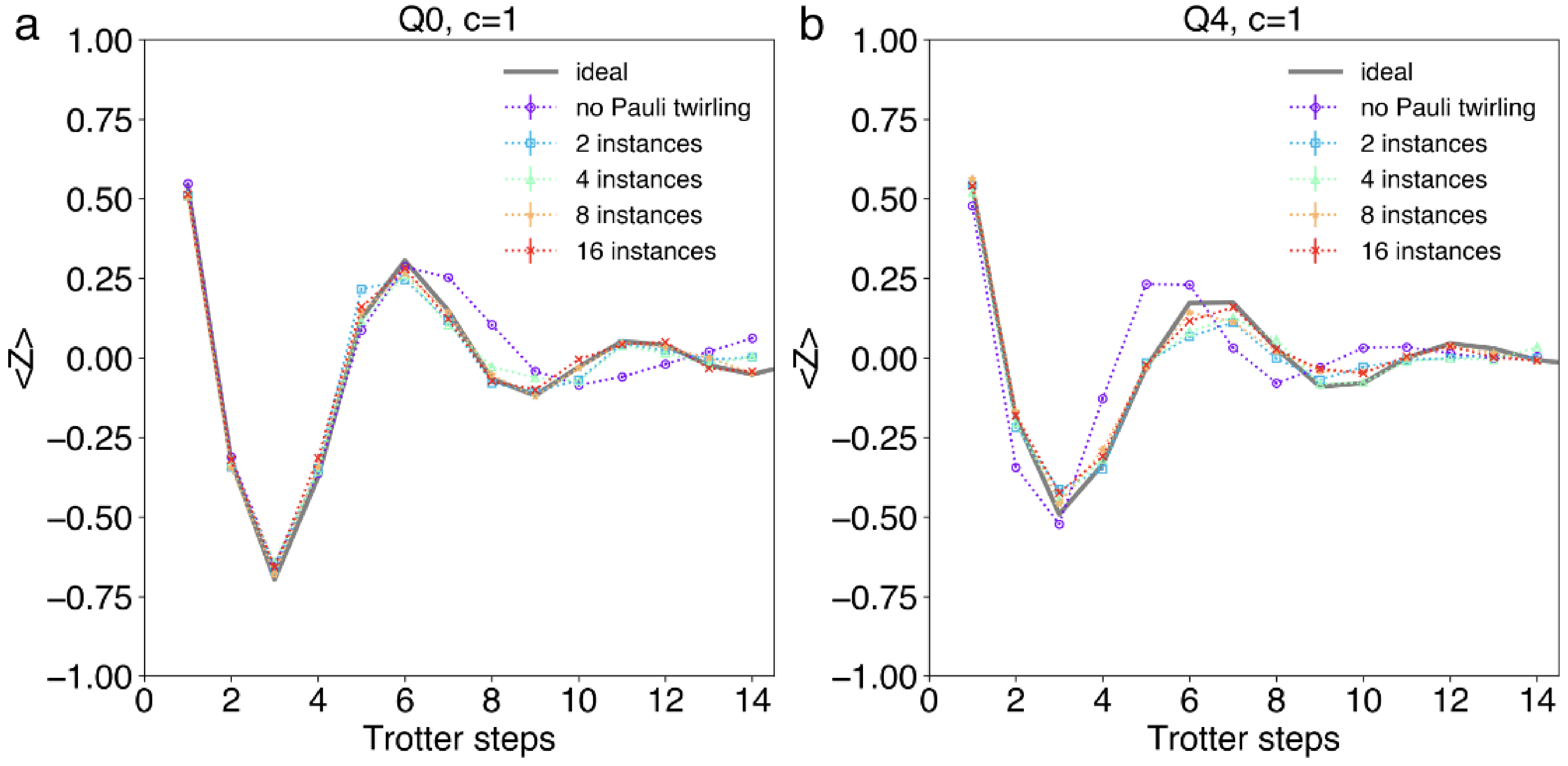
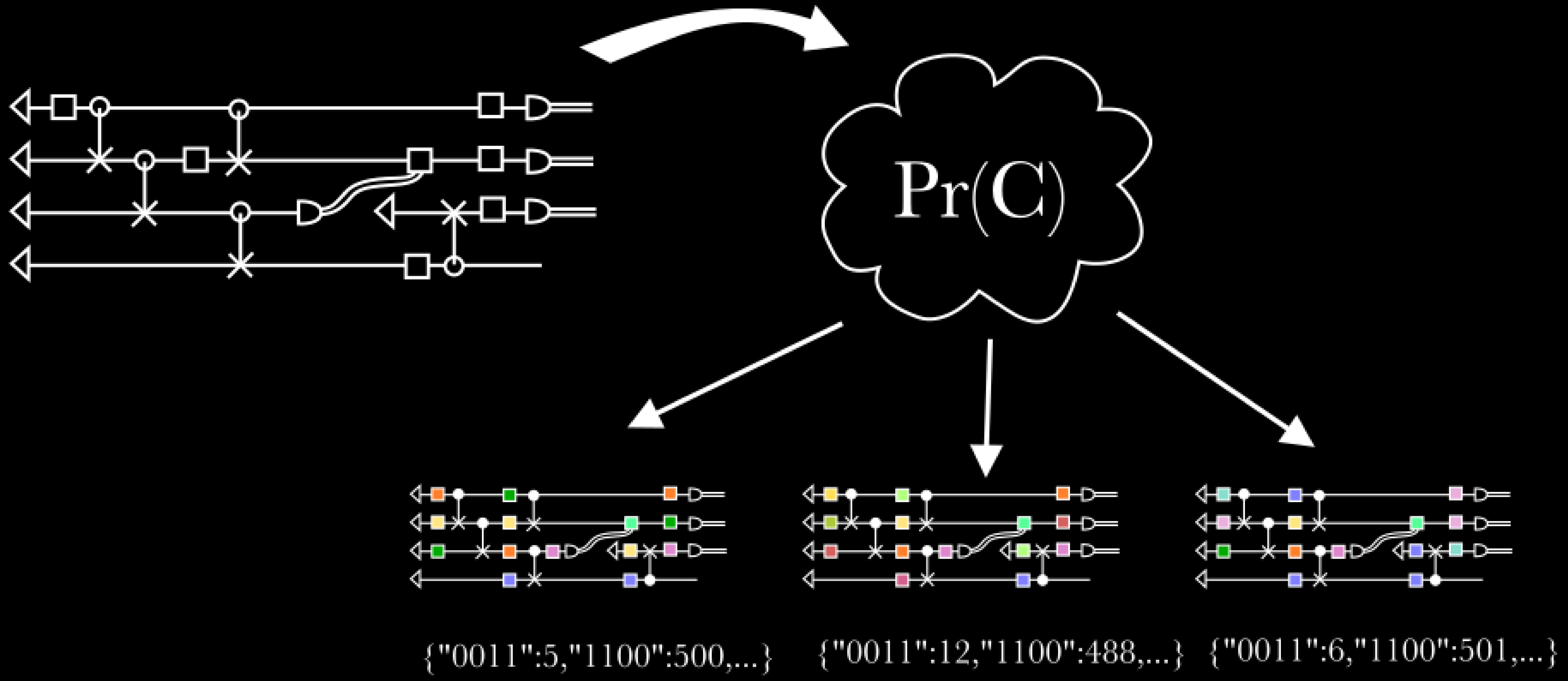


FIG. S4. **Impact of Pauli twirling for quench circuits with native gate d**
panels depict data from the quench dynamics with 26 qubits, for J = 0.5236, wi

Circuit randomization: big picture

Tailoring a more desirable noise profile by pooling the results from structurally similar random circuits.



Pr(C)

{"0011":5,"1100":500,...}

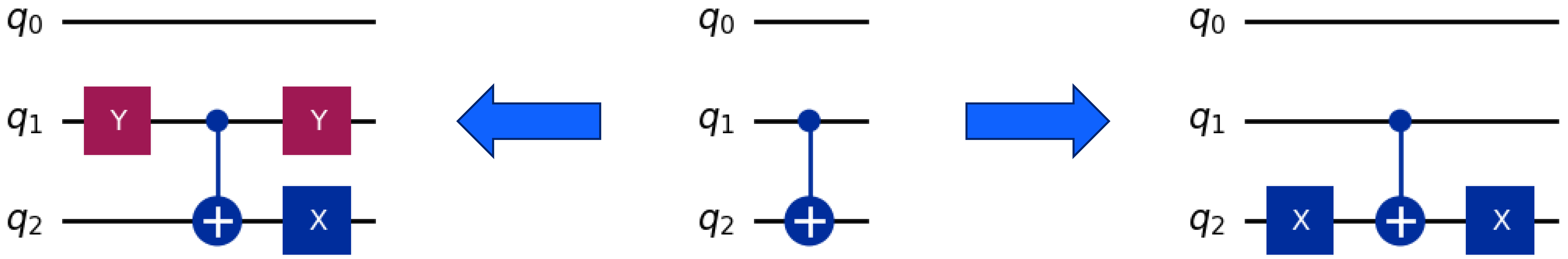
{"0011":12,"1100":488,...}

{"0011":6,"1100":501,...}

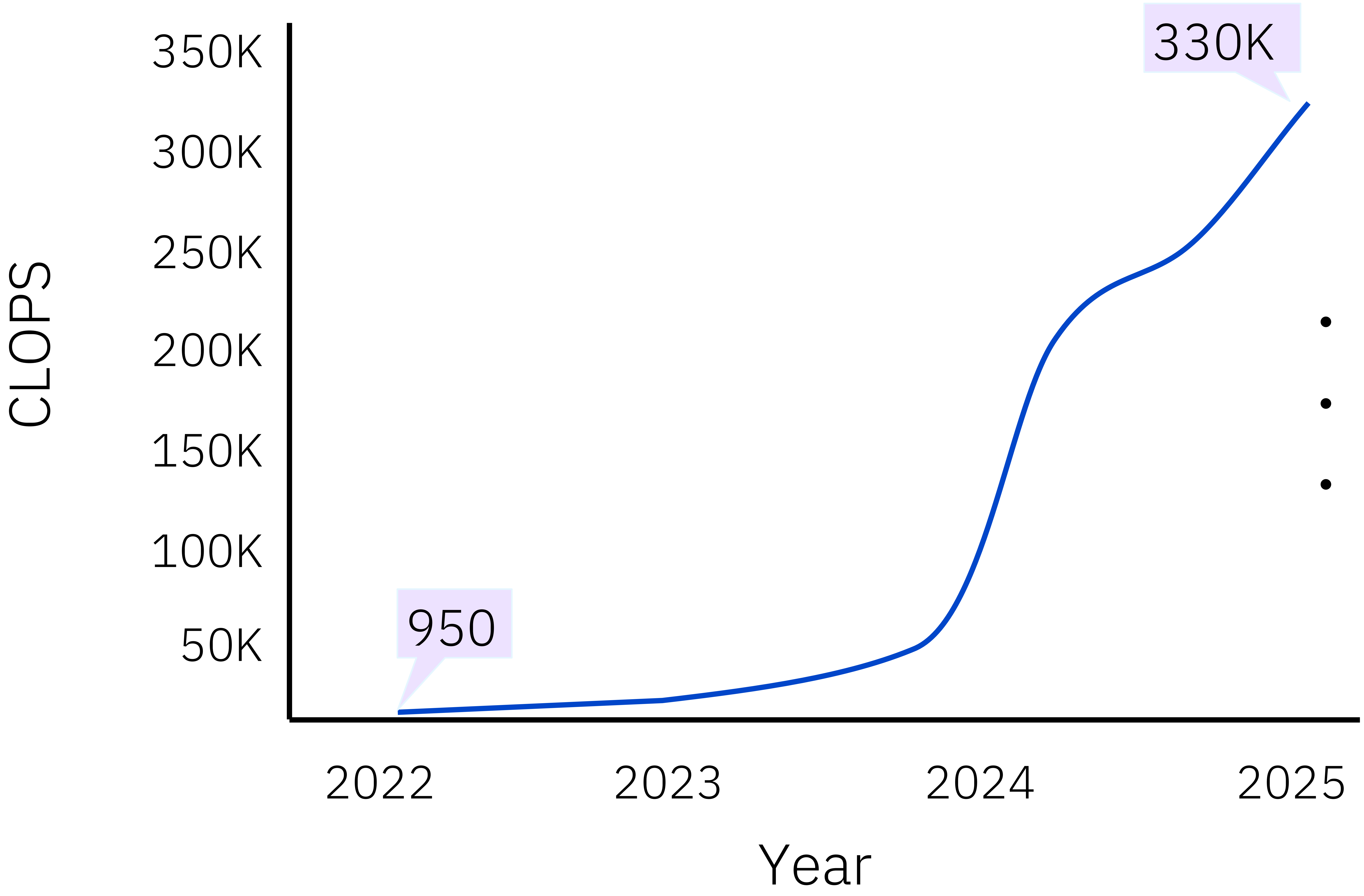
Pauli Twirling

Strategically add random gates which don’t alter the circuit logic in a structured way.
Canonical examples - **Pauli Twirling**

Average the experimental results over many randomizations



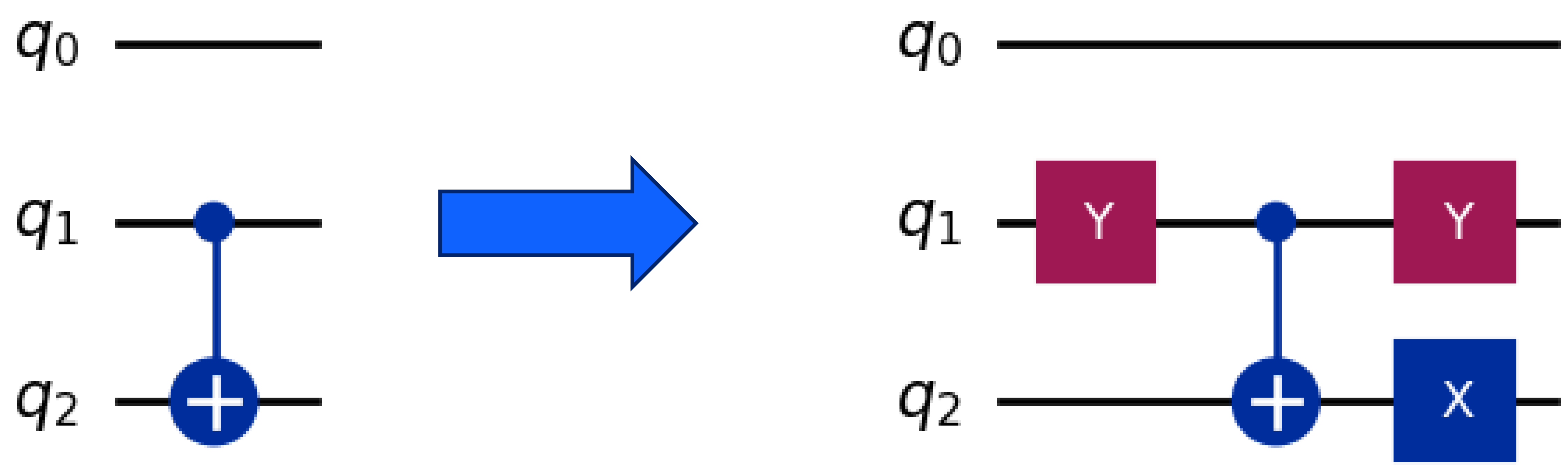
IBM Runtime Performance



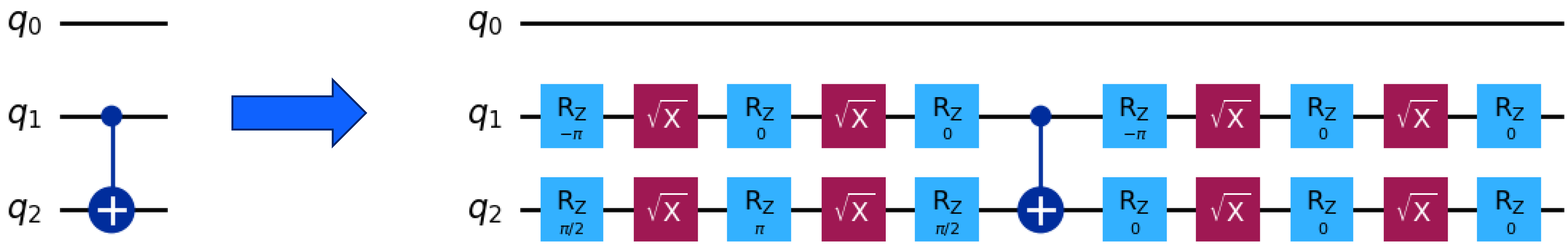
- **Parametric** circuit compilation
- Oxidation of low-level stack
- Optimized data movement

So, always try to express circuit variations parametrically!

✗

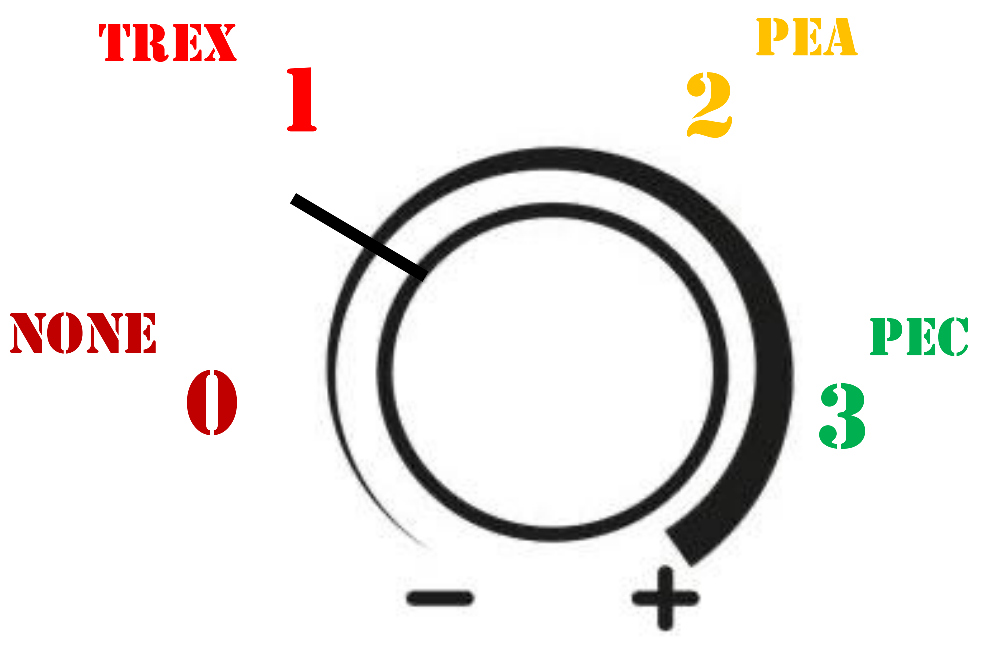


✓



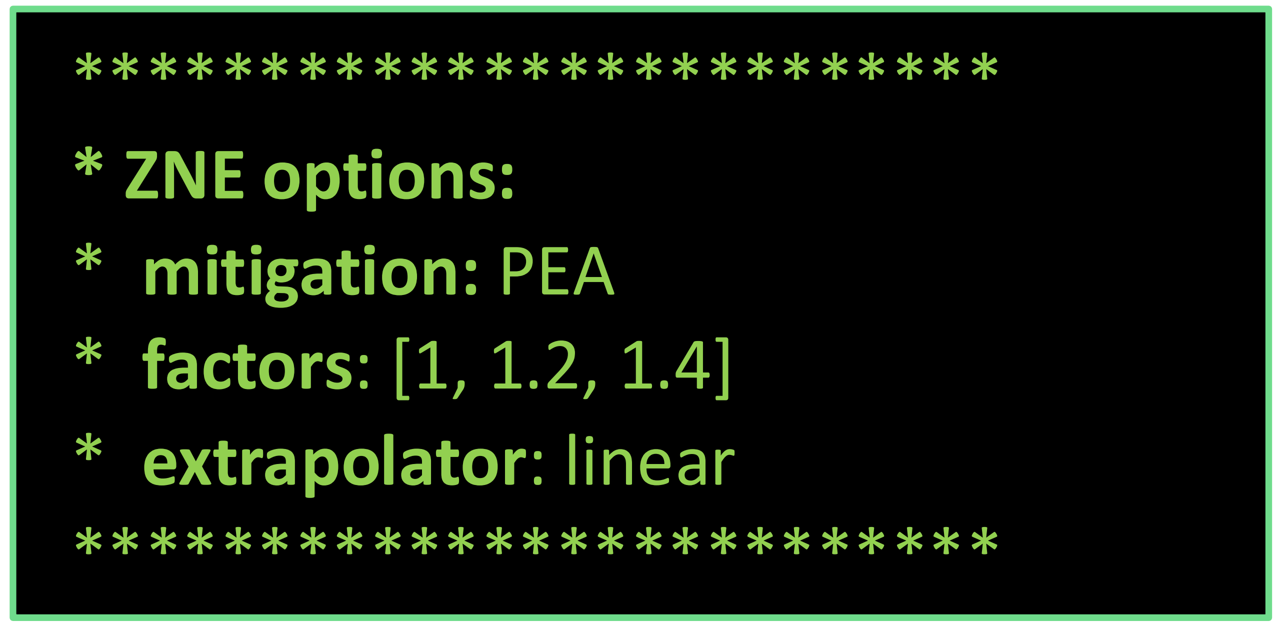
How can I introduce circuit randomization into my application and build mitigation workflows?

The evolution of the IBM primitives.



Phase 1. One knob.
Where choices are made server-side.

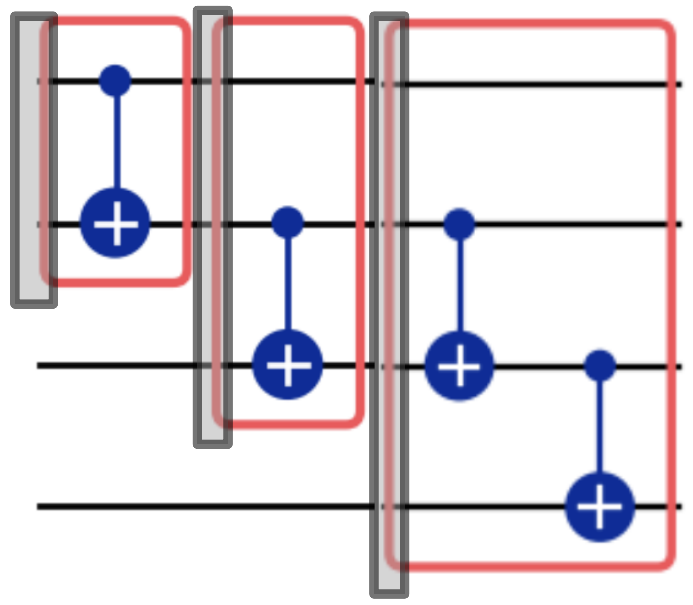
- Users select resilience levels.
- Runtime makes choices and does the heavy lifting.



```
************************
* ZNE options:
*  mitigation: PEA
*  factors: [1, 1.2, 1.4]
*  extrapolator: linear
************************
```

Phase 2. Guided control.
Where you can tweak some parameters.

- Resilience levels still supported.
- Additionally, users can define custom options to meet their needs.



Phase 3. Modular error mitigation.
Where you get near-complete control on the client-side.

Phase 3 Tools

samplomatic

pip install samplomatic

- Python library for circuit randomization
- Vendor-agnostic
- Tools to implement error mitigation protocols
- Describes the process of randomizing a circuit as a graph, “samplex”

Executor

qiskit-ibm-runtime

- New, low-level primitive
- Sampler-esque
- Optionally accepts samplex graphs for server-side randomization

Noise learner

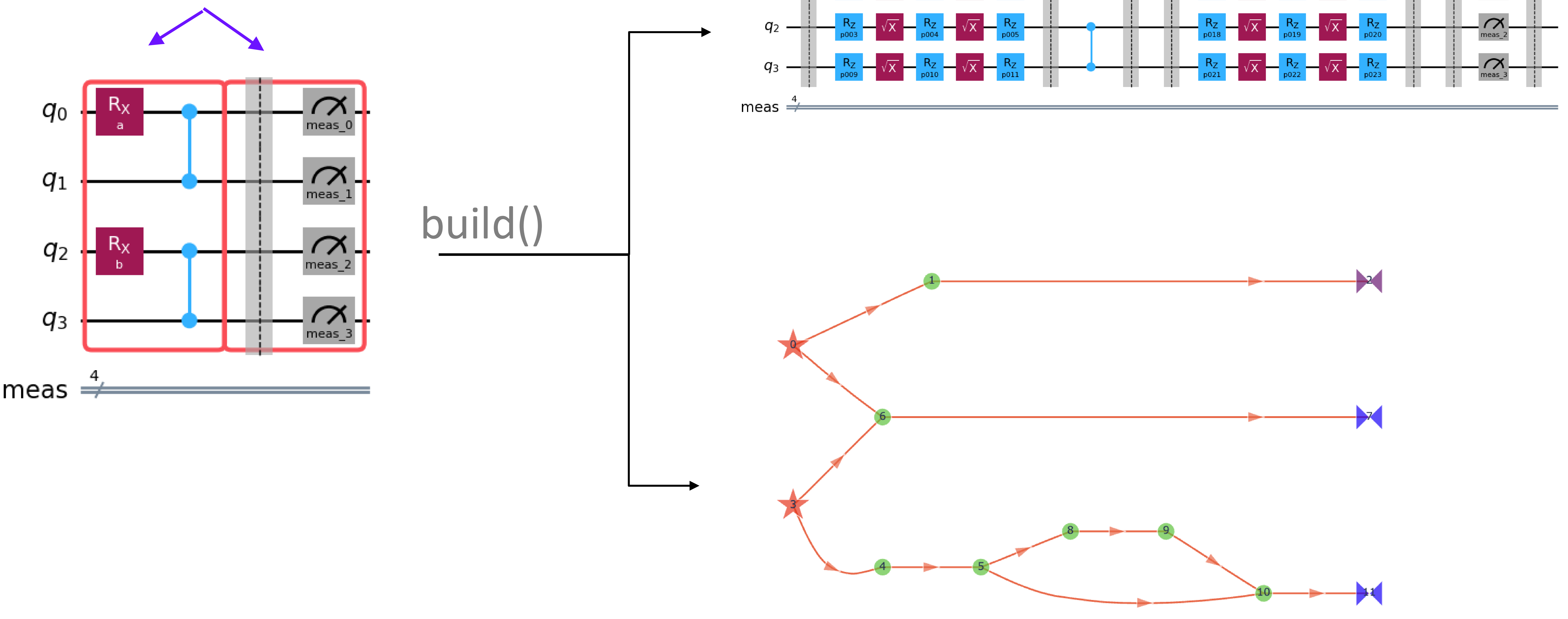
qiskit-ibm-runtime

- Accepts circuit layers, returns noise models
- Refresh of existing service

Samplomatic: main idea

```
from samplomatic import build
template, samplex = build(circuit)
```

Annotated with Twirl / InjectNoise / ChangeBasis



build()

template

samplex

delarative

procedural

Executor: main idea

```
from qiskit-ibm-runtime import Executor, QuantumProgram

job = Executor(backend).run(quantum_program)
```

User

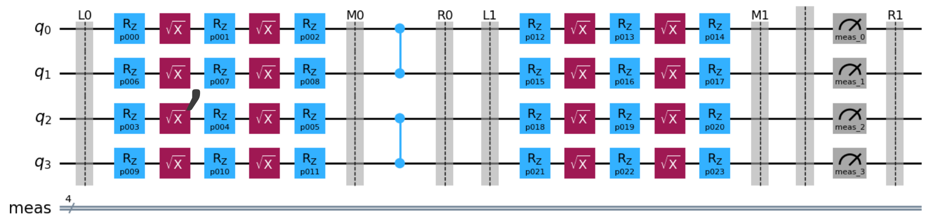
client-side

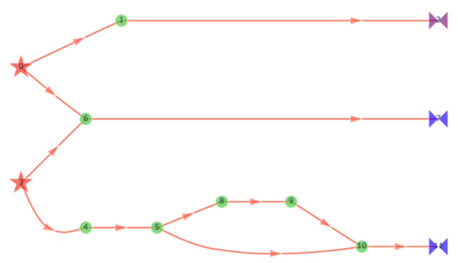
server-side

Runtime

QPU

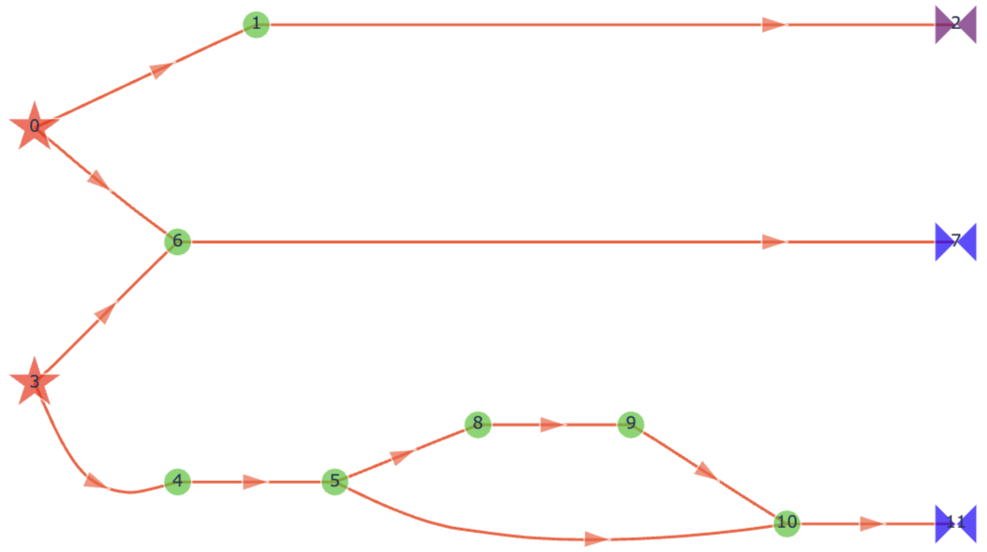
```
program = QuantumProgram(shots=1000)
program.append(
```

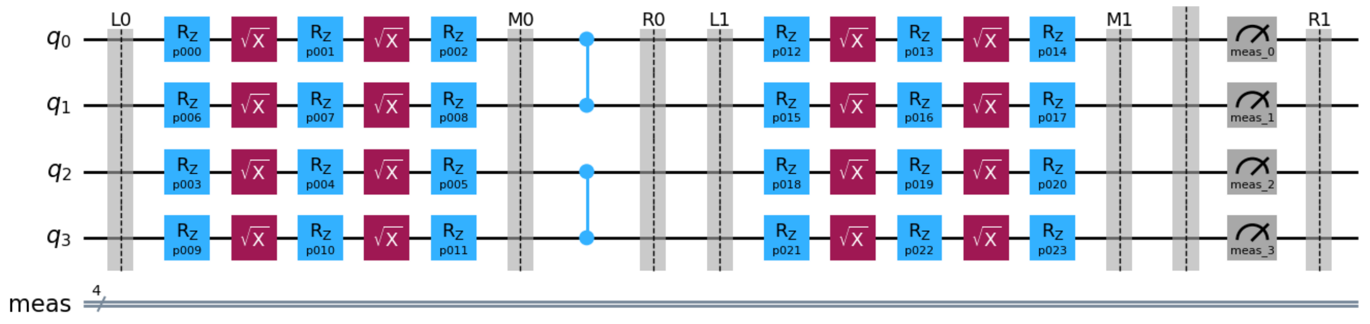




```
)

job = Executor(backend).run(program)
```





Job input

Job results

parameter values

bit data

Noise Learner: main idea

```
from qiskit-ibm-runtime import NoiseLearnerV3

job = NoiseLearnerV3(backend).run(boxes)
```

Submit a job:

- List of boxes whose twirled noise models you want to know

Runtime+QPU:

- A noise learning protocol with 100s of circuits is run server-side

Get results:

- One noise model object for every box

IBM