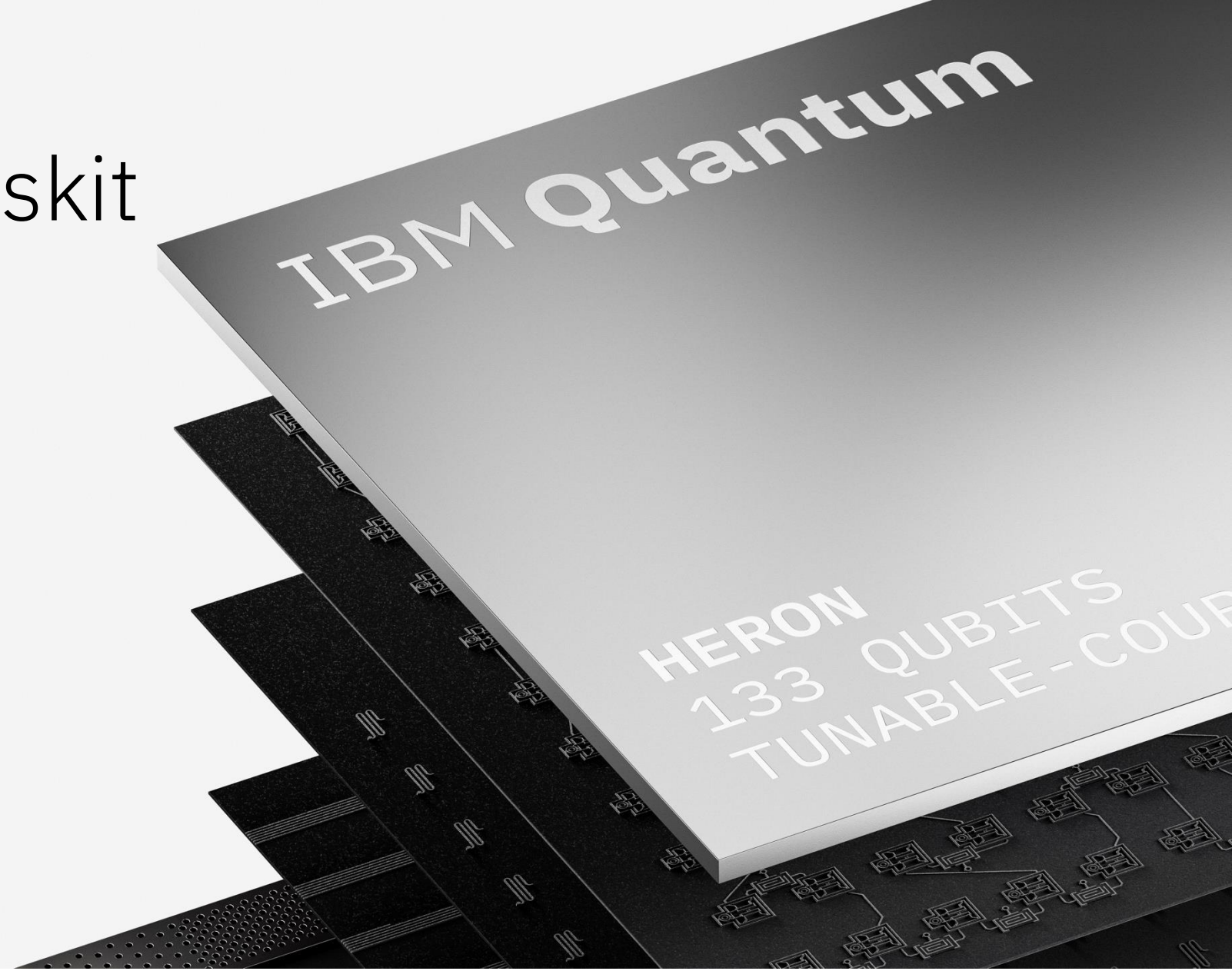
Error mitigation workflows with Qiskit



Nate Earnest-Noble
Quantum Algorithm Engineer
Nate@ibm.com

IBM

Agenda

- Overview of Hardware Noise
- Error Mitigation & Suppression Overview
- Overview of Qiskit implementation
- Code Demos
  - Estimator Demo – ZNE gate folding
  - Samplomatic Demo – SLC, PNA
- Q&A

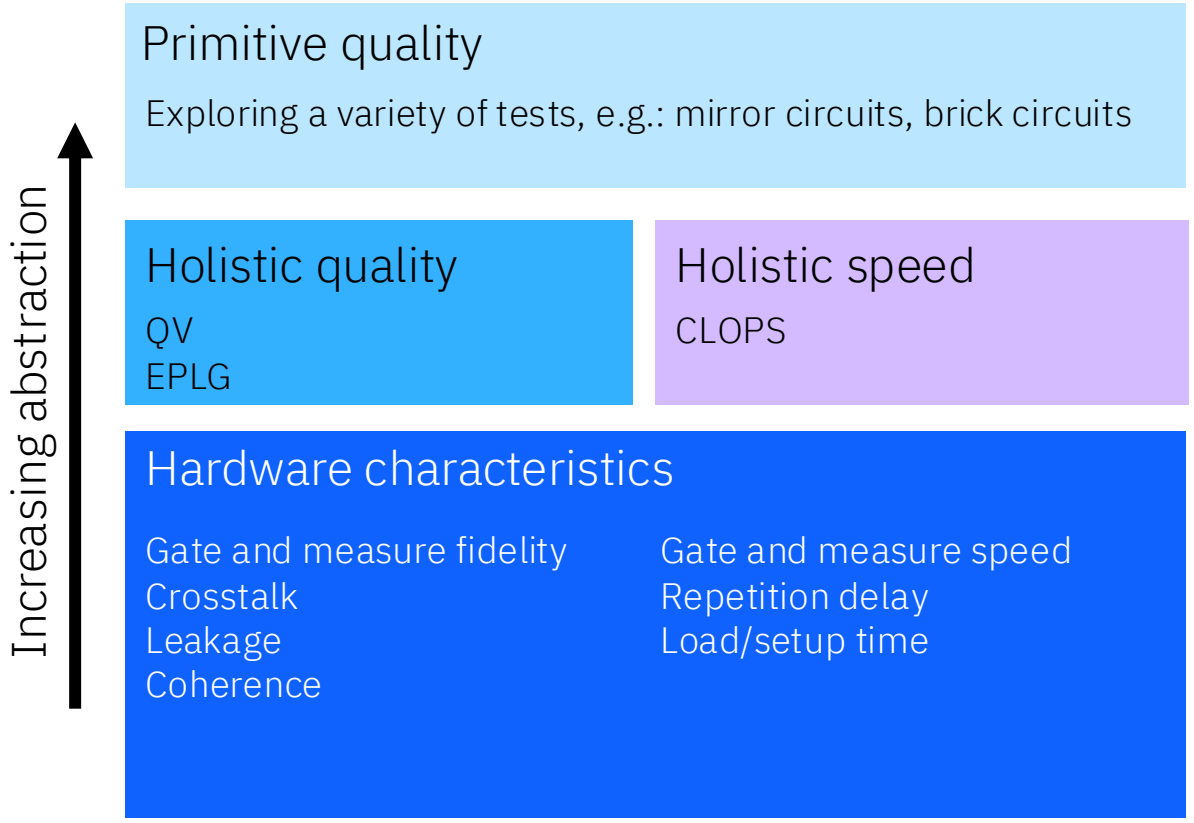
IBM

Agenda

- Overview of Hardware Noise
- Error Mitigation & Suppression Overview
- Overview of Qiskit implementation
- Code Demos
  - Estimator Demo – ZNE gate folding
  - Samplomatic Demo – SLC, PNA
- Q&A

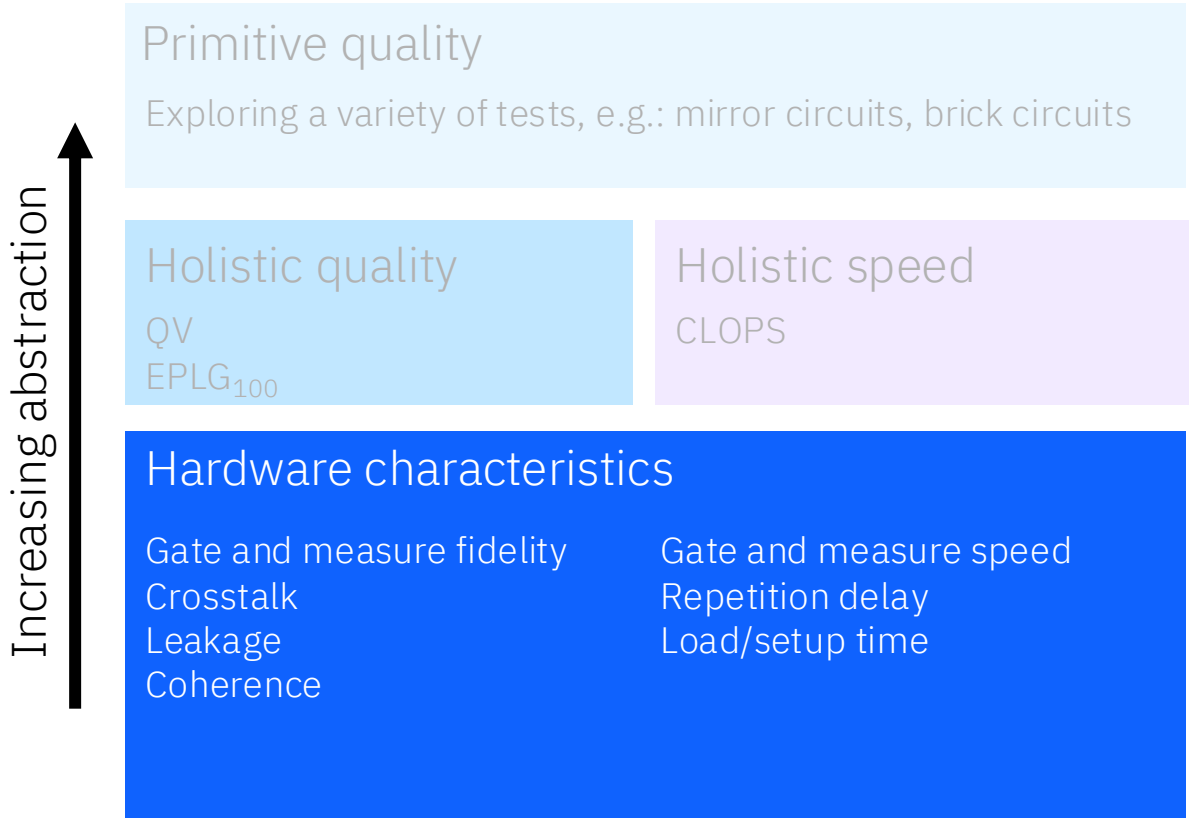
IBM

Execution benchmarks



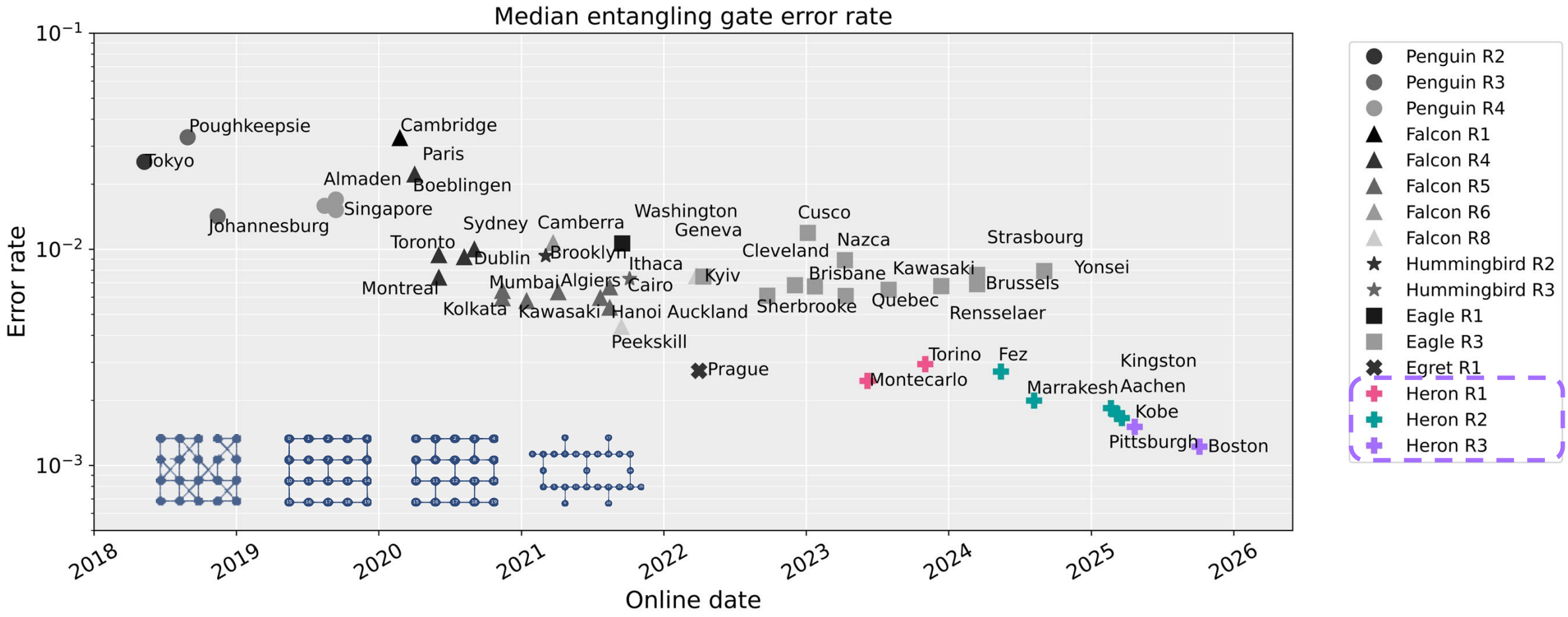
Strategies on defining benchmarks: M. Amico *et al.*, arXiv:2303.02108

Execution benchmarks

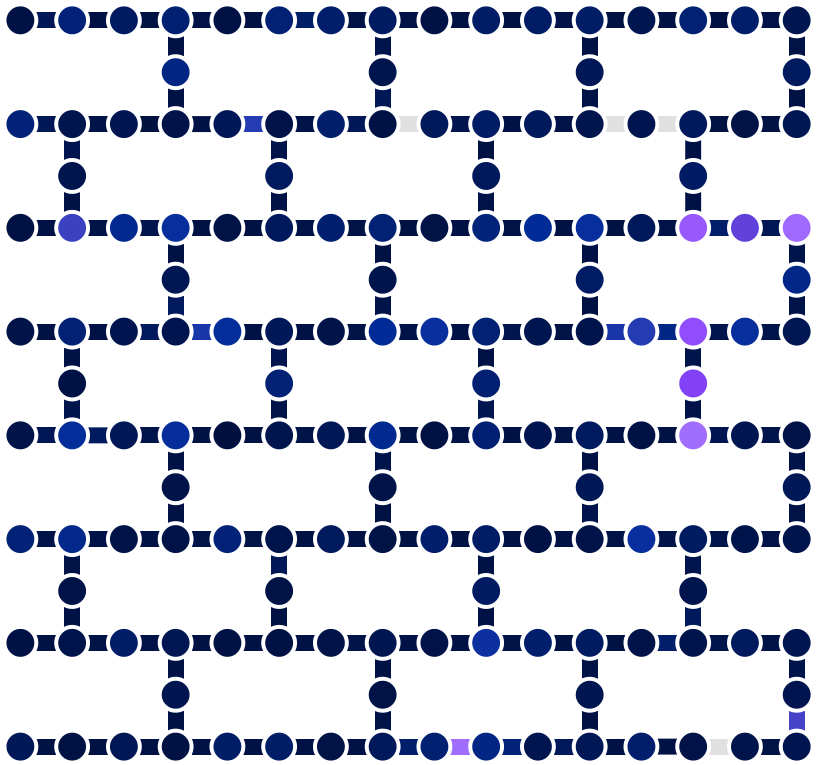


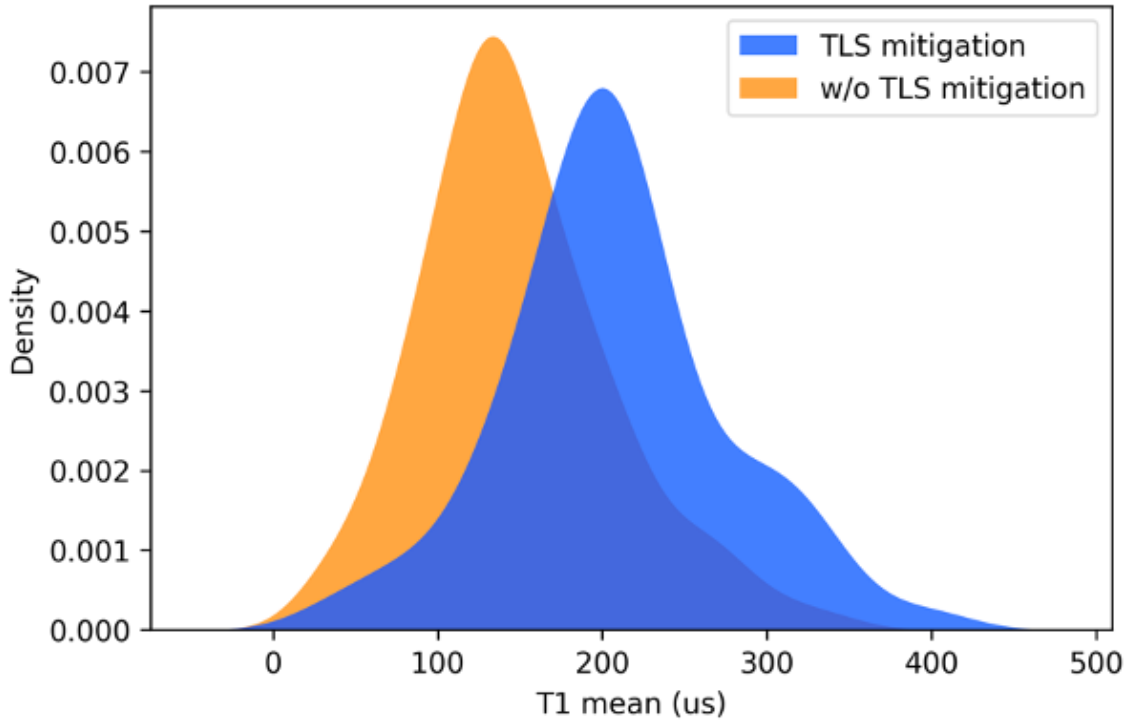
Strategies on defining benchmarks: M. Amico *et al.*, arXiv:2303.02108

Quality: The road to improved 2Q gate errors



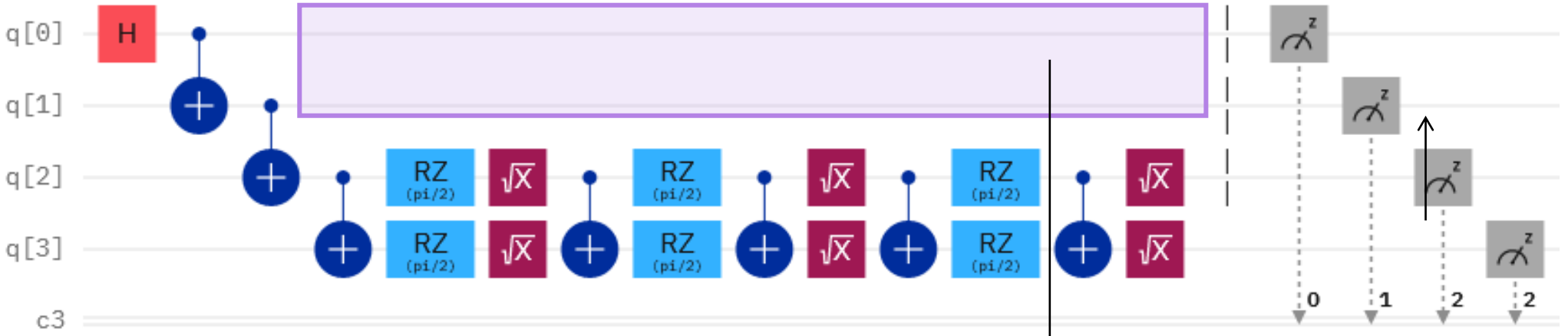
Two-level system mitigation with Heron 2.x (*ibm_fez*)





47% improvement in qubit coherence via active two-level system mitigation

Noise in quantum systems



ibm_torino

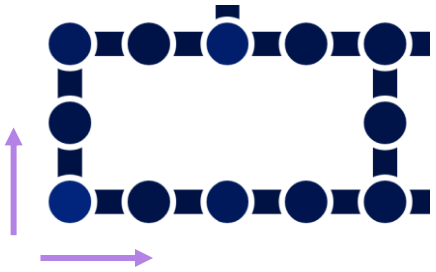
OpenQASM 3

Details

133 Qubits	Status:	Online	Median CZ error: 4.384e-3
0.8% EPLG	Total pending jobs:	43 jobs	Median SX error: 3.161e-4
3.8K CLOPS	Processor type ⓘ:	Heron r1	Median readout error: 1.770e-2
	Version:	1.0.16	Median T1: 175.89 us
	Basis gates:	CZ, ID, RZ, SX, X	Median T2: 134.83 us
	Your instance usage:	0 jobs	

Bath/system coupling

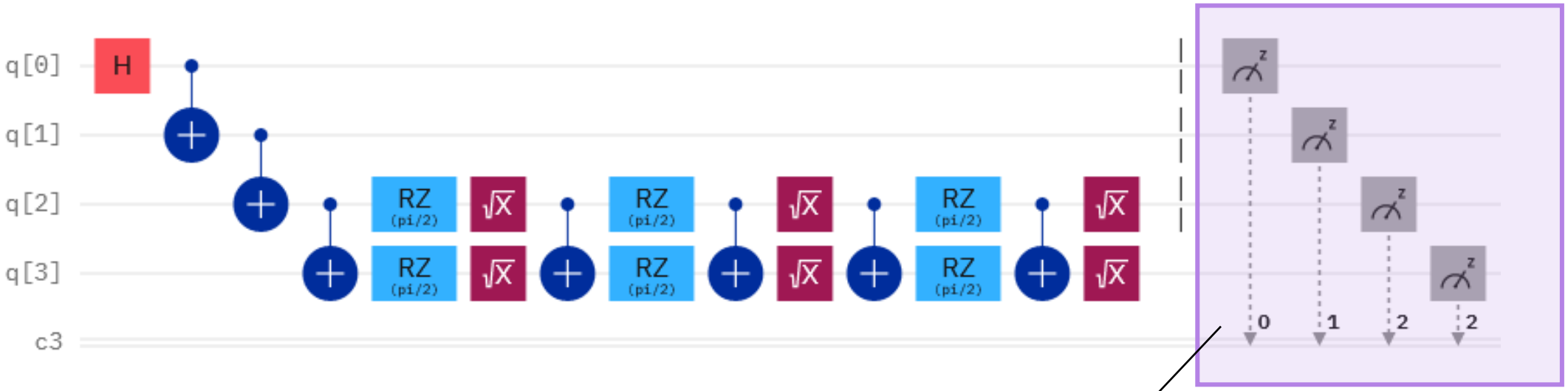
Environmental noise: relevant for deep sparse circuits



Decoherence
Information loss over time.

Crosstalk
Idle qubits interact with their neighbors. (Difficult to predict but less severe in heron devices!)

Noise in quantum systems



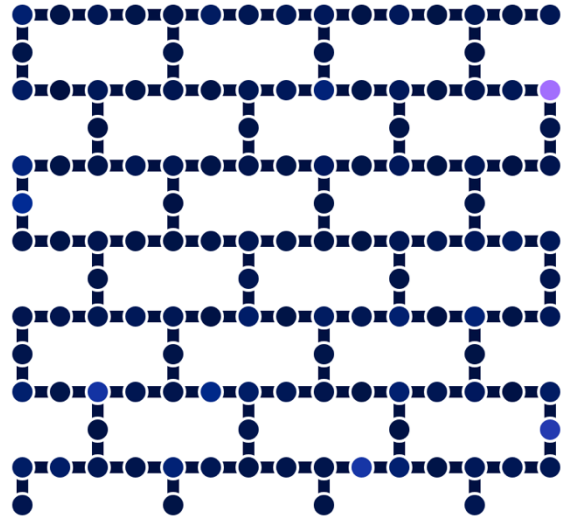
ibm_torino

OpenQASM 3

Details

133 Qubits	Status:	Online	Median CZ error: 4.384e-3
0.8% EPLG	Total pending jobs:	43 jobs	Median SX error: 3.161e-4
3.8K CLOPS	Processor type ⓘ:	Heron r1	Median readout error: 1.770e-2
	Version:	1.0.16	Median T1: 175.89 us
	Basis gates:	CZ, ID, RZ, SX, X	Median T2: 134.83 us
	Your instance usage:	0 jobs	

Readout errors because of imperfect measurements.



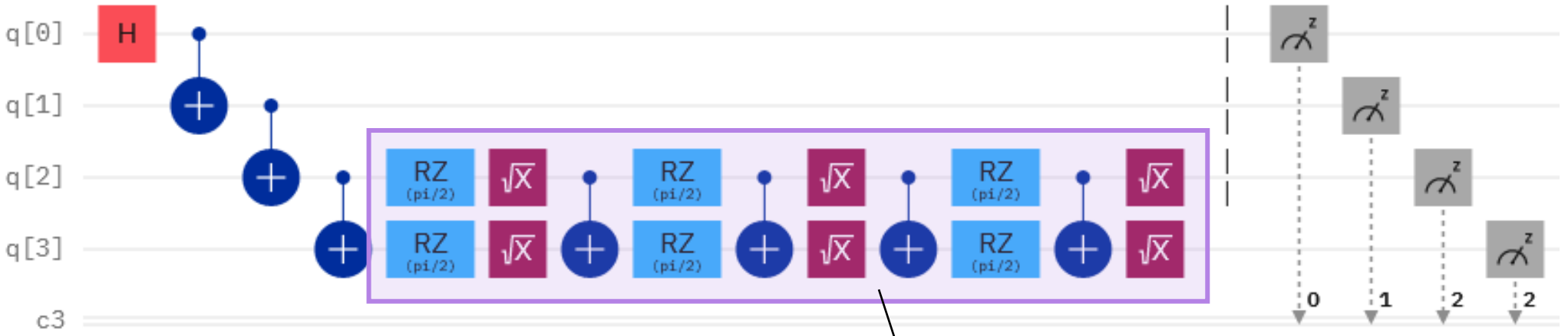
Qubit:

Readout assignment error

Median 1.940e-2

min 3.333e-3 max 4.333e-1

Noise in quantum systems



ibm_torino

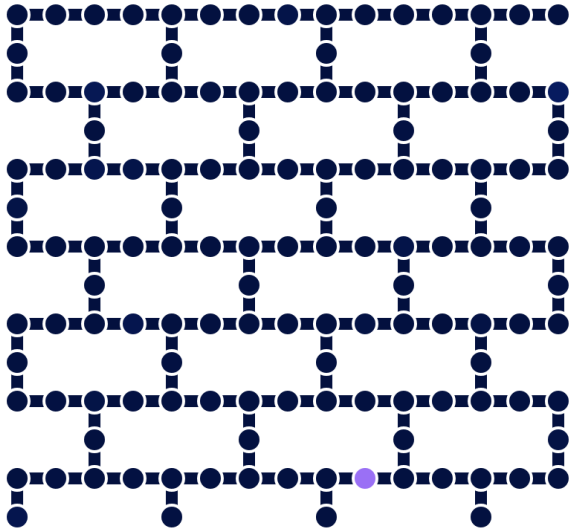
OpenQASM 3

Details

133 Qubits	Status:	Online	Median CZ error: 4.384e-3
0.8% EPLG	Total pending jobs:	43 jobs	Median SX error: 3.161e-4
3.8K CLOPS	Processor type ⓘ:	Heron r1	Median readout error: 1.770e-2
	Version:	1.0.16	Median T1: 175.89 us
	Basis gates:	CZ, ID, RZ, SX, X	Median T2: 134.83 us
	Your instance usage:	0 jobs	

Gate errors

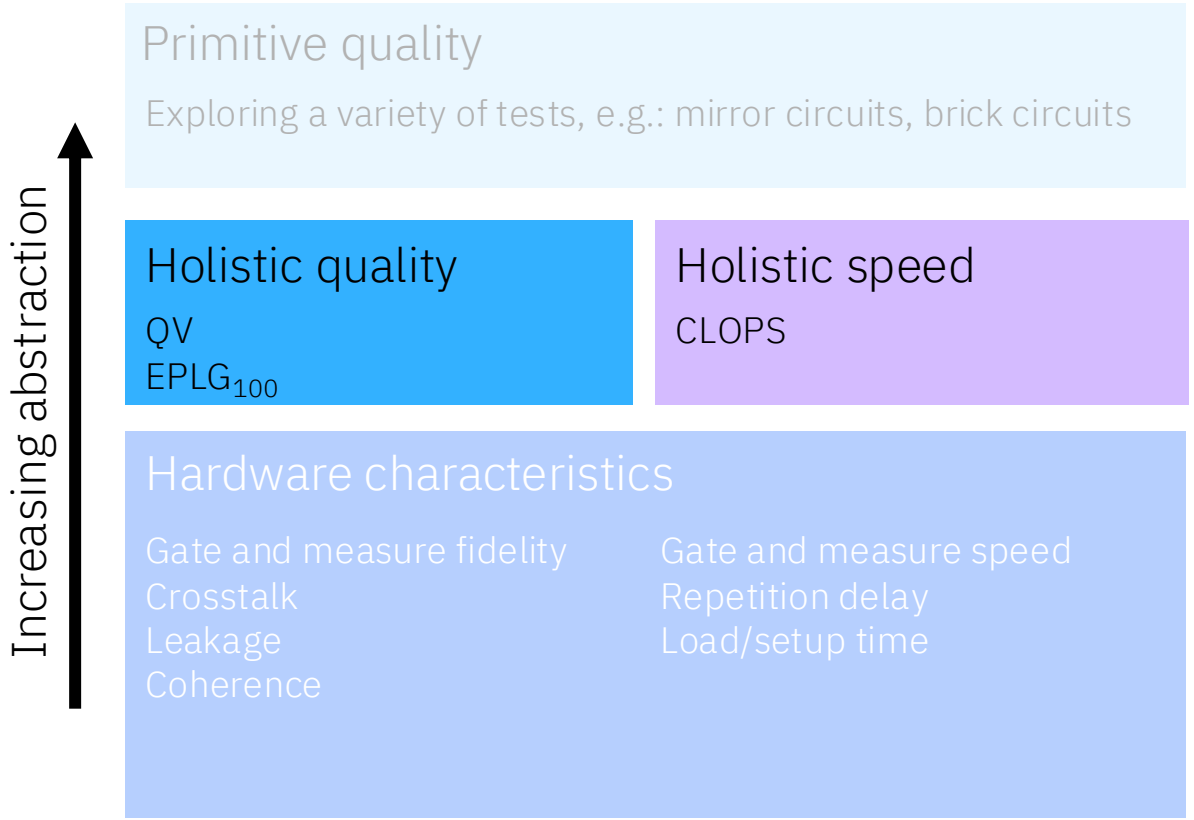
Because of imperfect operations on qubits.
(Higher for deep, dense circuits with many two-qubit gates).



Qubit: √x (sx) error — Median 3.299e-4 (min 1.335e-4, max 8.469e-2)

Connection: CZ error — Median 1.000e+0 (min 1.000e+0, max 1.000e+0)

Execution benchmarks

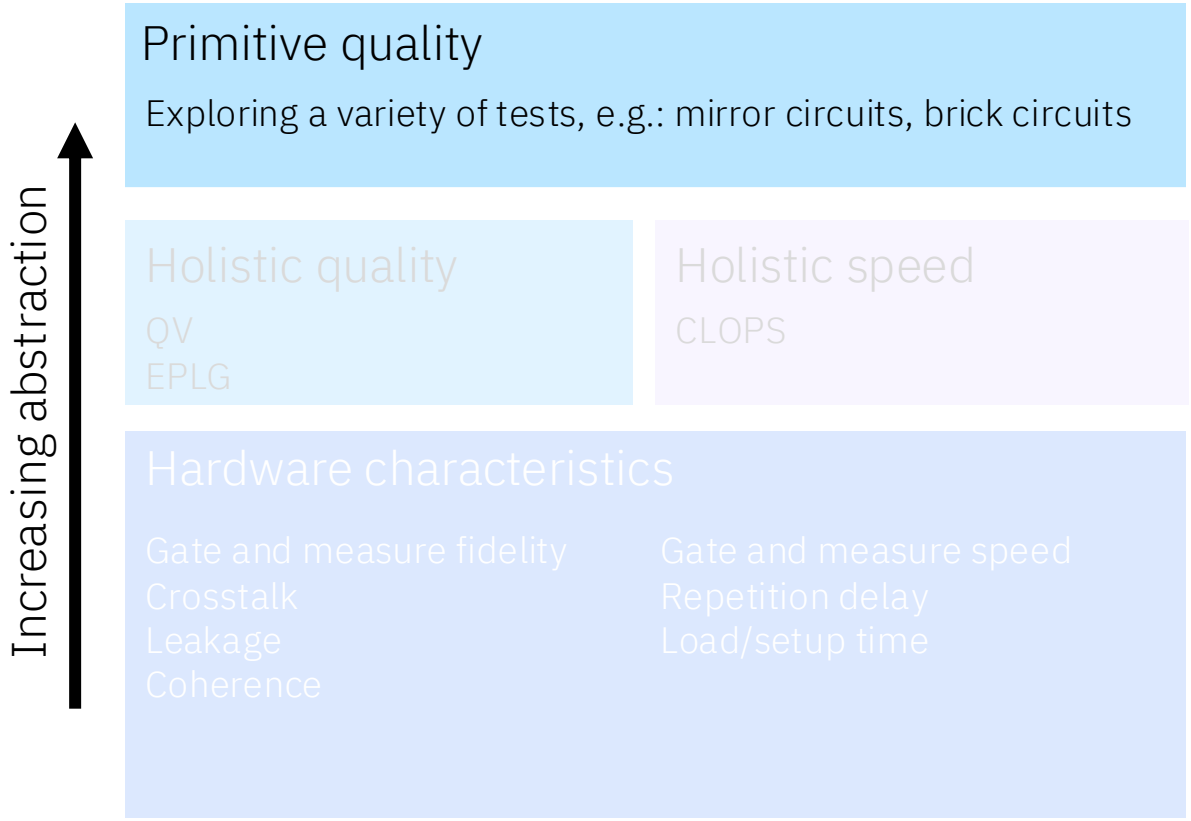


Performance = Scale + Quality + Speed

Performance =	Scale	+	Quality	+	Speed
~Two years ago	133 Qubits		$8.4x10^{-3}$ 2Q error ($EPLG_{100}$)		3.8K CLOPS

Performance =	Scale	+	Quality	+	Speed
Today	156 Qubits		2.2×10^{-3} 2Q error ($EPLG_{100}$) *Best 2Q gate error 6.53×10^{-4} as measured by RB		340K CLOPS

Execution benchmarks



Strategies on defining benchmarks: M. Amico *et al.*, arXiv:2303.02108

Agenda

- Overview of Hardware Noise
- Error Mitigation & Suppression Overview
- Overview of Qiskit implementation
- Code Demos
  - Estimator Demo – ZNE gate folding
  - Samplomatic Demo – SLC, PNA
- Q&A

IBM

Fighting noise in quantum systems

Before fault tolerance…

Error suppression techniques **minimize the impact of noise by either preventing errors from happening or modifying the noise structure:**

- Dynamical decoupling (DD)
- Pauli Twirling (PT)

Before or during execution (typically)
Additional classical resources dominate

Error mitigation techniques **allow errors to occur and reduce their effect by modeling the device noise that was present at the time of execution:**

- Twirled Readout Error eXtinction (TREX), MThree
- Zero Noise Extrapolation (ZNE)
- PEC, PEC-SLC
- PNA
- …

After or during execution (typically)
Additional quantum resources dominate

Fault tolerance

Error correction builds up redundancies that allow us to detect and correct errors when they occur, leading to essentially error-free logical qubits!

"Well, your quantum computer is broken in every way possible simultaneously."

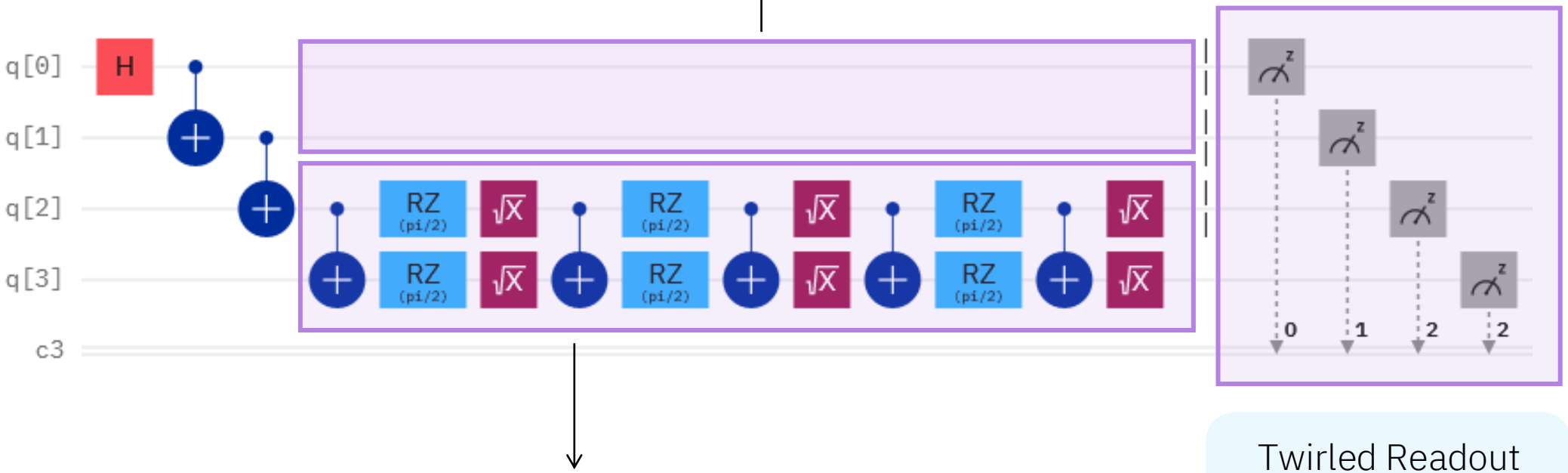
https://www.ibm.com/quantum/blog/quantum-error-suppression-mitigation-correction

Fighting noise in quantum systems

Dynamical decoupling (DD)

Readout errors

Environmental noise



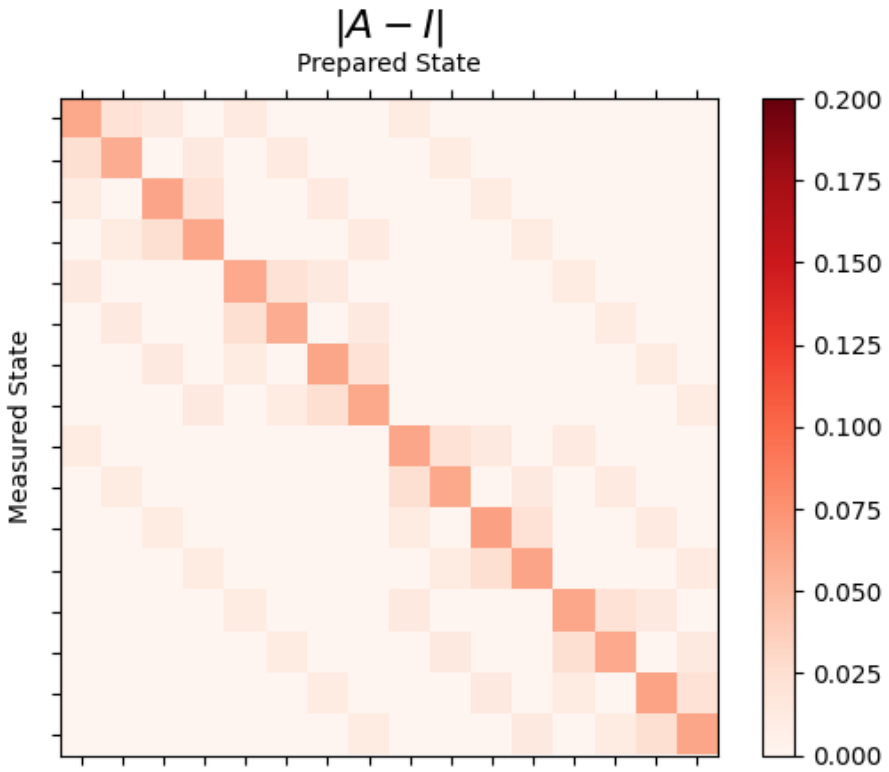
Gate errors

Pauli Twirling (PT)

Zero Noise Extrapolation (ZNE), PEC-SLC, PNA, ...

Twirled Readout Error eXtinction (TREX) or MThree

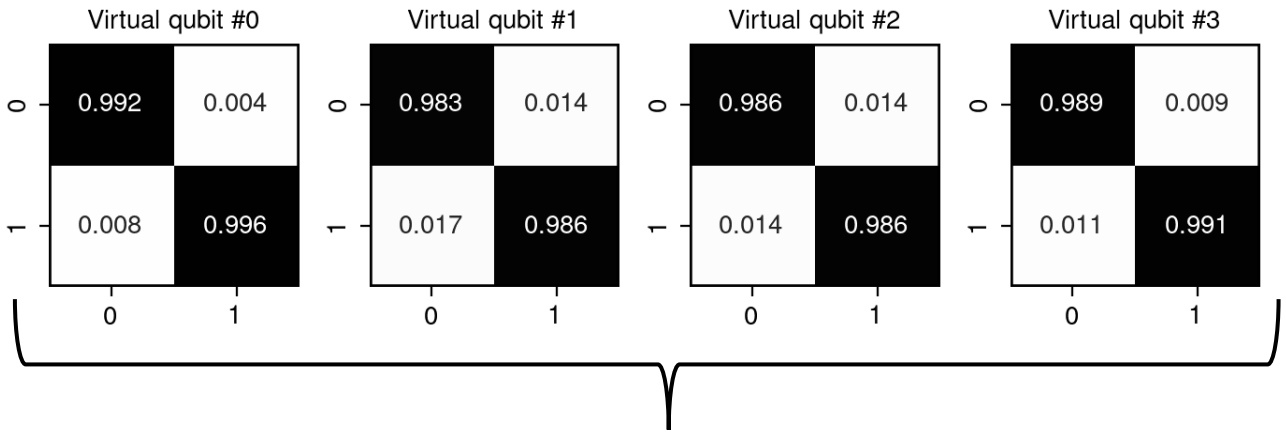
Twirled Readout Error eXtinction (TREX)



Readouts errors cause the wrong states to be measured.

This is depicted in the readout- error transfer matrix.

In scenarios when the noise is not correlated between qubits, measurement errors can be measured per qubit and the full transfer matrix is then reconstructed as a tensor product.



$$2^N \times 2^N$$

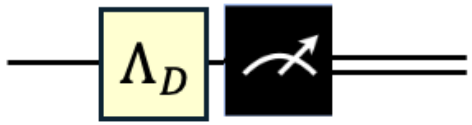
The inverse of the transfer matrix can be used for error mitigation, but it cannot be obtained efficiently in general.

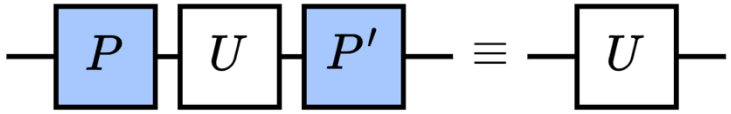
Twirled Readout Error eXtinction (TREX)

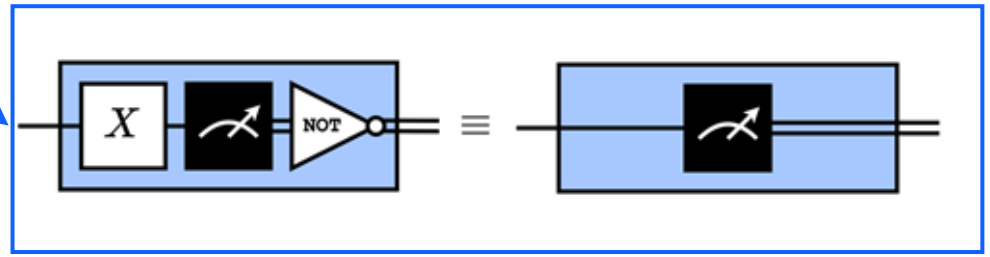
Via **measurement twirling we can diagonalize the** readout-error transfer matrix.

Arbitrary transfer matrix

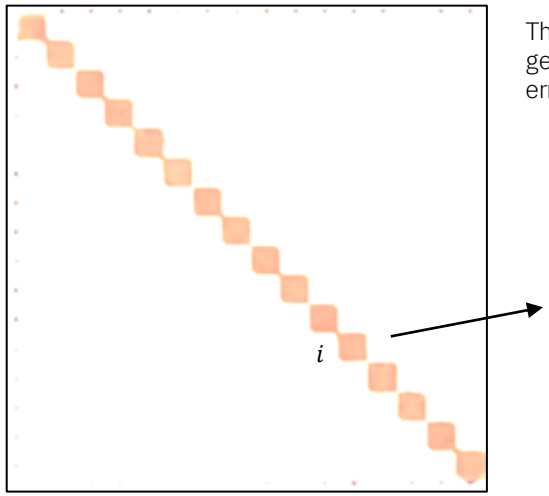
Diagonal transfer matrix







Average over random bitflips that are undone in classical postprocessing.

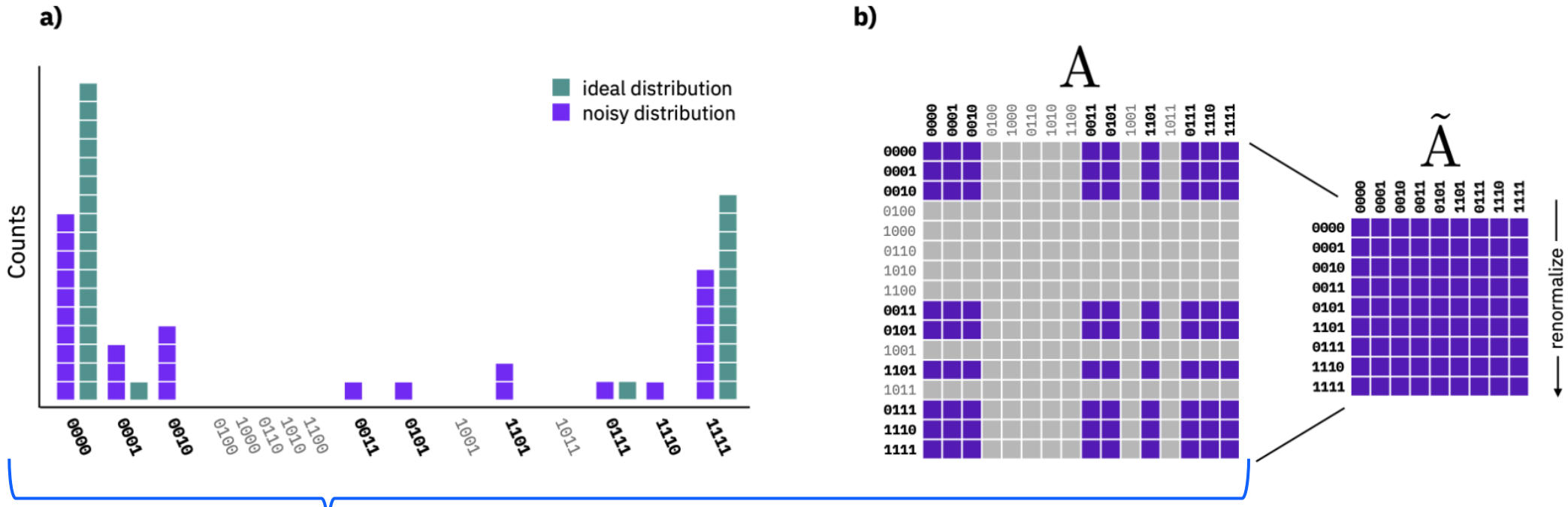


Then, we need to run calibration circuits to get the inverse of the (diagonal) readout error transfer matrix.

$$< O_i > = \frac{< \widetilde{O_i} >}{E_i}$$

Only valid for expectation values!

Matrix-free Measurement Mitigation (M3)



The full A-matrix can provide unphysical values and has scaling which is prohibitive to use are larger scale

Reduce matrix to only the noisy bit strings observed and invert this

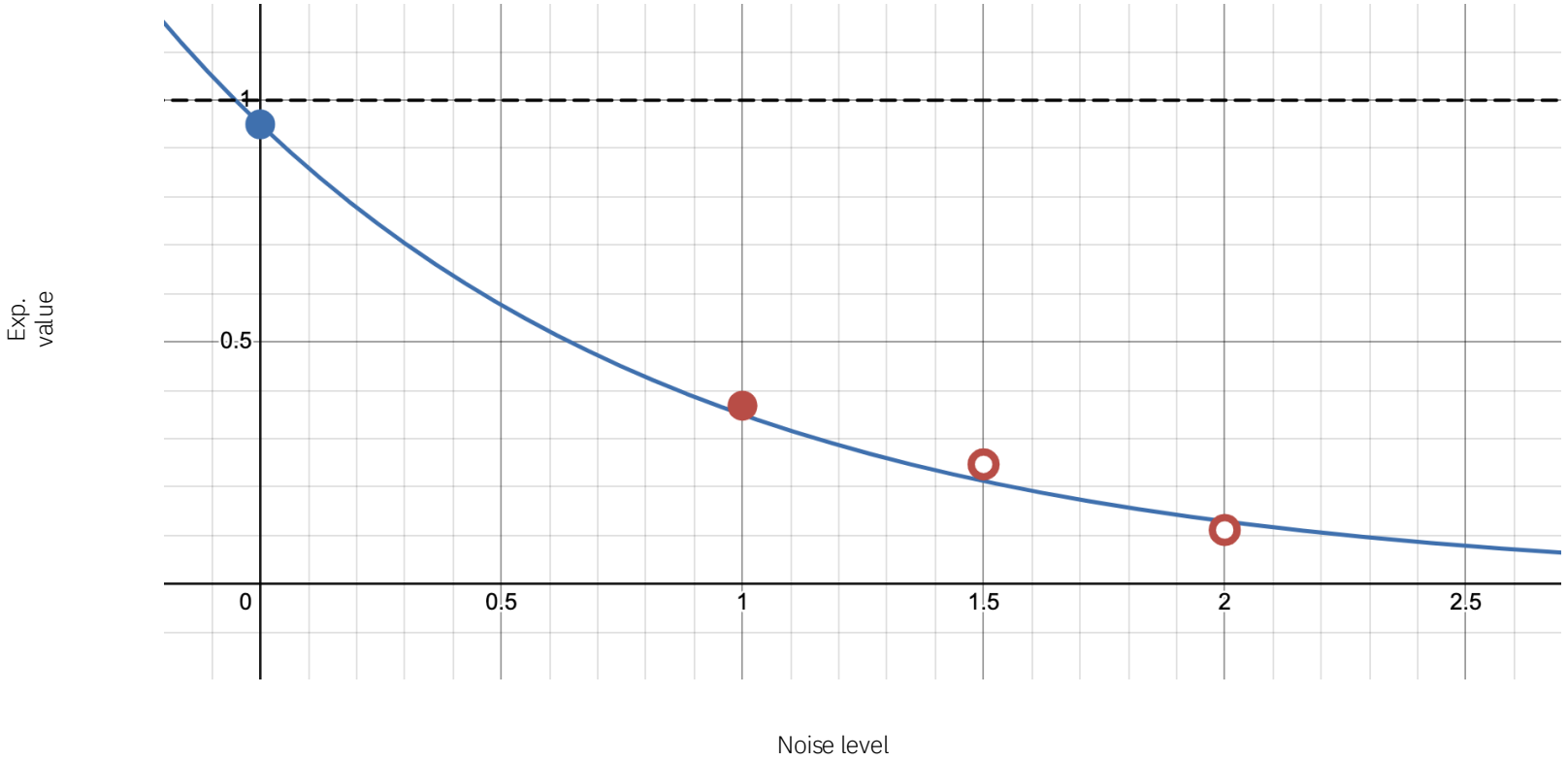
Gives Quasi-probability values. Can be projected onto near probability distribution, but guarantees in errors bars are lost

Zero Noise Extrapolation (ZNE)

Measures the effects of amplified noise to infer what the results would look like in the absence of noise.

1. Noise amplification:
the original circuit unitary is executed at different levels of noise.

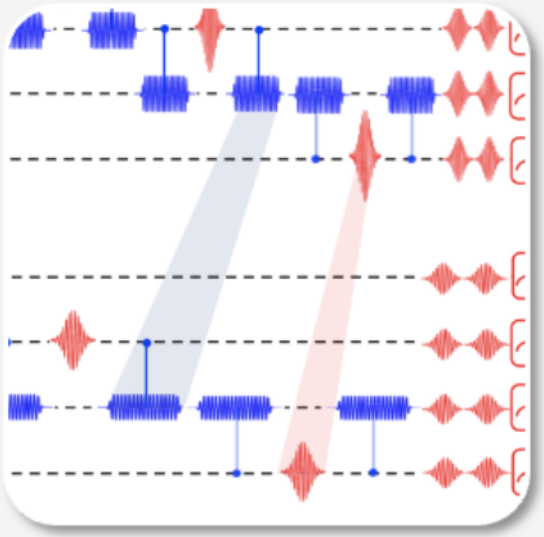
2. Extrapolation:
the zero-noise limit is inferred from the noisy expectation-value results.



Zero Noise Extrapolation (ZNE)

Step 1: Noise amplification

Pulse stretching	Gate folding	Probabilistic error amplification
Scale pulse duration via calibration	Repeat gates in identity cycles $U \mapsto U(U^{-1}U)^{\lambda-1}/2$	Add noise via sampling Pauli channels



Kandala et al. Nature (2019)

It's an analogue technique.

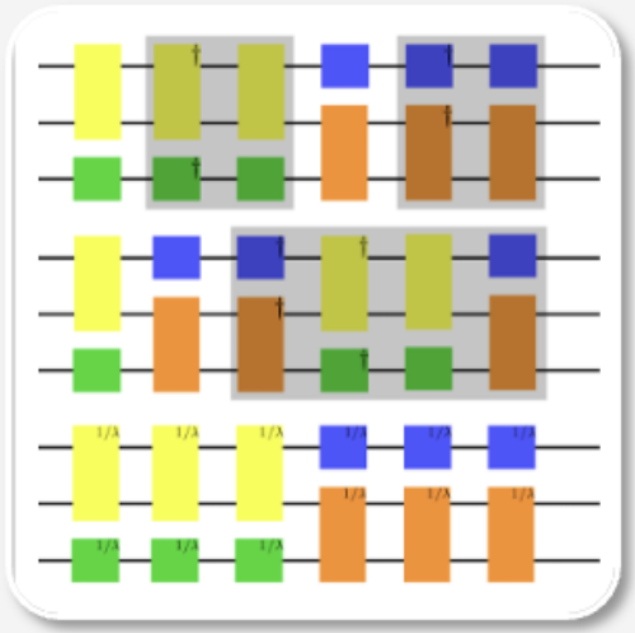
It assumes noise is proportional to pulse duration.

It requires costly pulse level calibration of the hardware.

Zero Noise Extrapolation (ZNE)

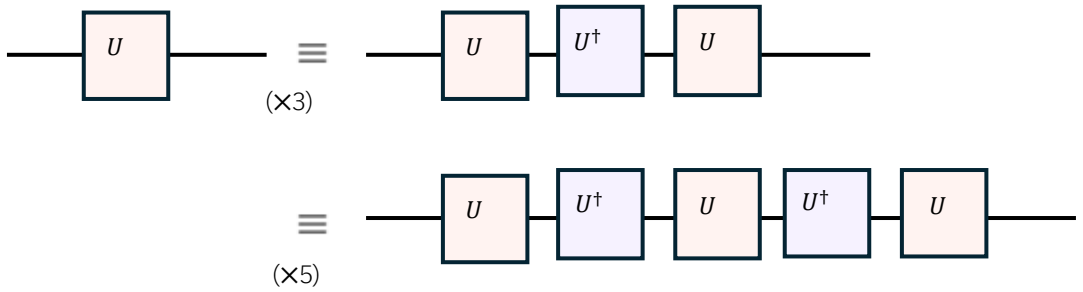
Step 1: Noise amplification

Pulse stretching	Gate folding	Probabilistic error amplification
Scale pulse duration via calibration	Repeat gates in identity cycles $U \mapsto U(U^{-1}U)^{\lambda-1}/2$	Add noise via sampling Pauli channels



Shultz et al. PRA (2022)

It is a heuristic approach but offers a good trade-off between result quality and resource requirements.

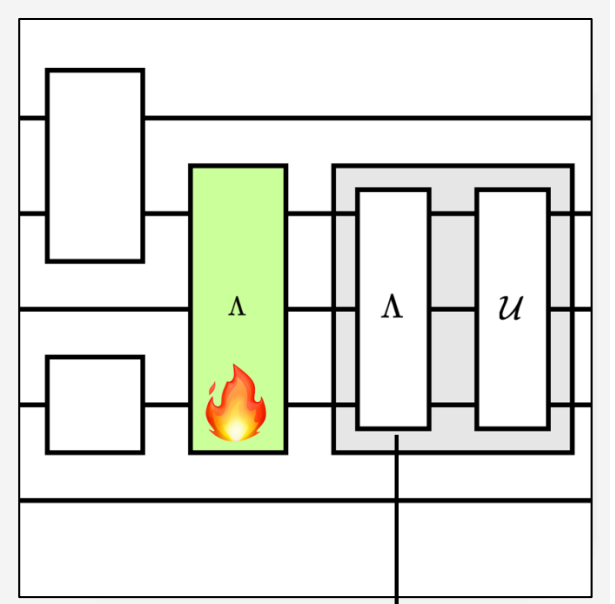


But it is limited by circuit depth!

Zero Noise Extrapolation (ZNE)

Step 1: Noise amplification

Pulse stretching	Gate folding	Probabilistic error amplification
Scale pulse duration via calibration	Repeat gates in identity cycles $U \mapsto U(U^{-1}U)^{\lambda-1}/2$	Add noise via sampling Pauli channels



Li & Benjamin PRX (2017)

It has general applicability and strong theoretical backing.

Used in the utility paper
Y. Kim et al. Nature (2023).

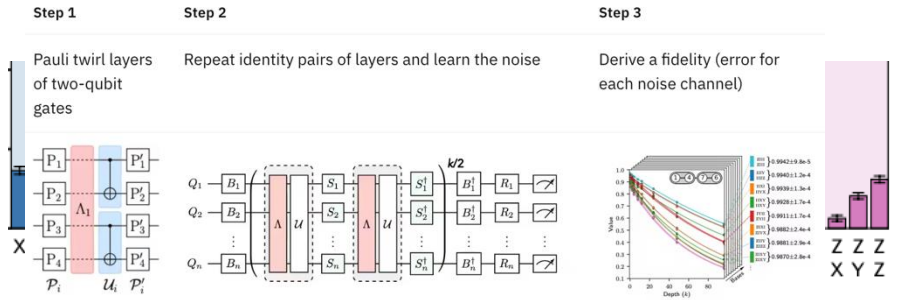
But it requires learning circuit-specific noise.

Idea: we want to learn the noise channel of certain layers of gates and reproduce it.

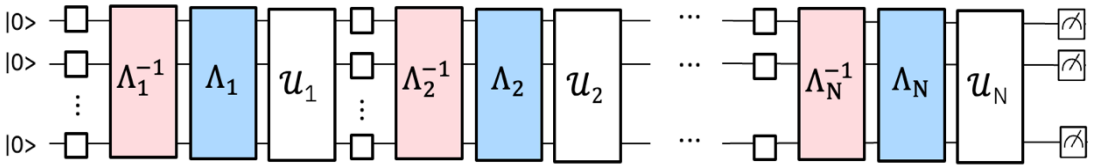
What is probabilistic error cancellation (PEC)?

A technique for mitigating gate noise by randomly sampling from an ensemble of circuits implementing the inverse noise channel.

Learn the noise model



Implement the inverse noise channel and sample



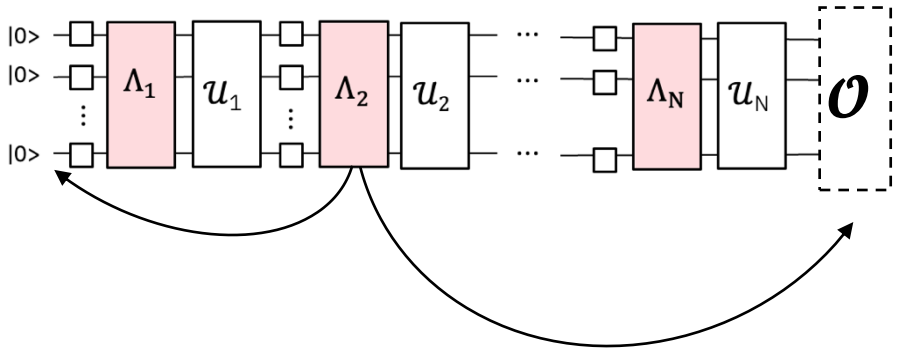
Reconstruct mitigated expectation value

[0100, ...]. ± 1
[0101, ...]. ± 1
[1001, ...]. ± 1

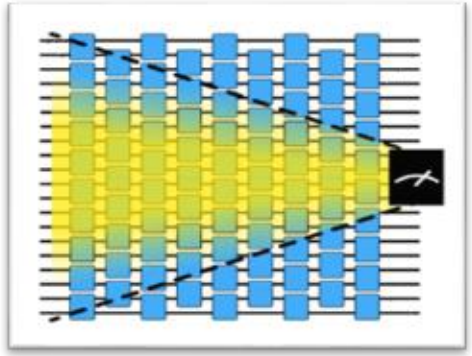
→ $\langle O \rangle$

What are shaded lightcones?

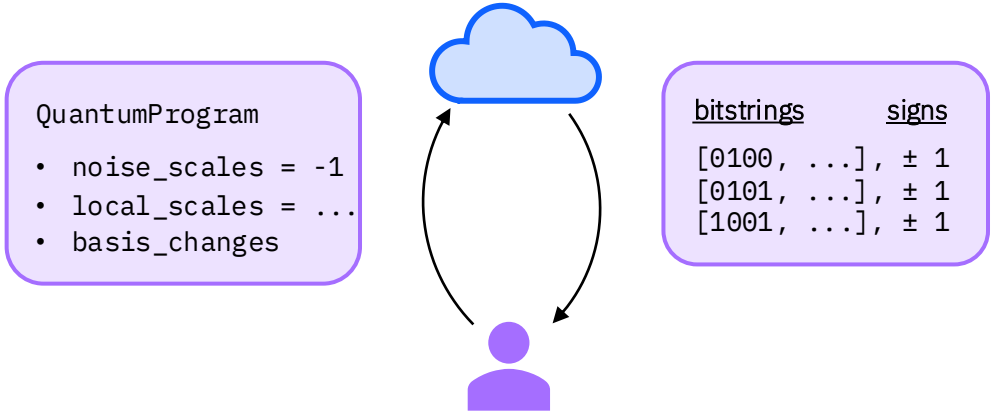
Determine impact of noise generators on expectation value.



Prune low-impact noise generators from the model, reducing the number of shots required for noise learning and error mitigation.



Perform PEC sampling with reduced overhead.



Computing the SLC bounds

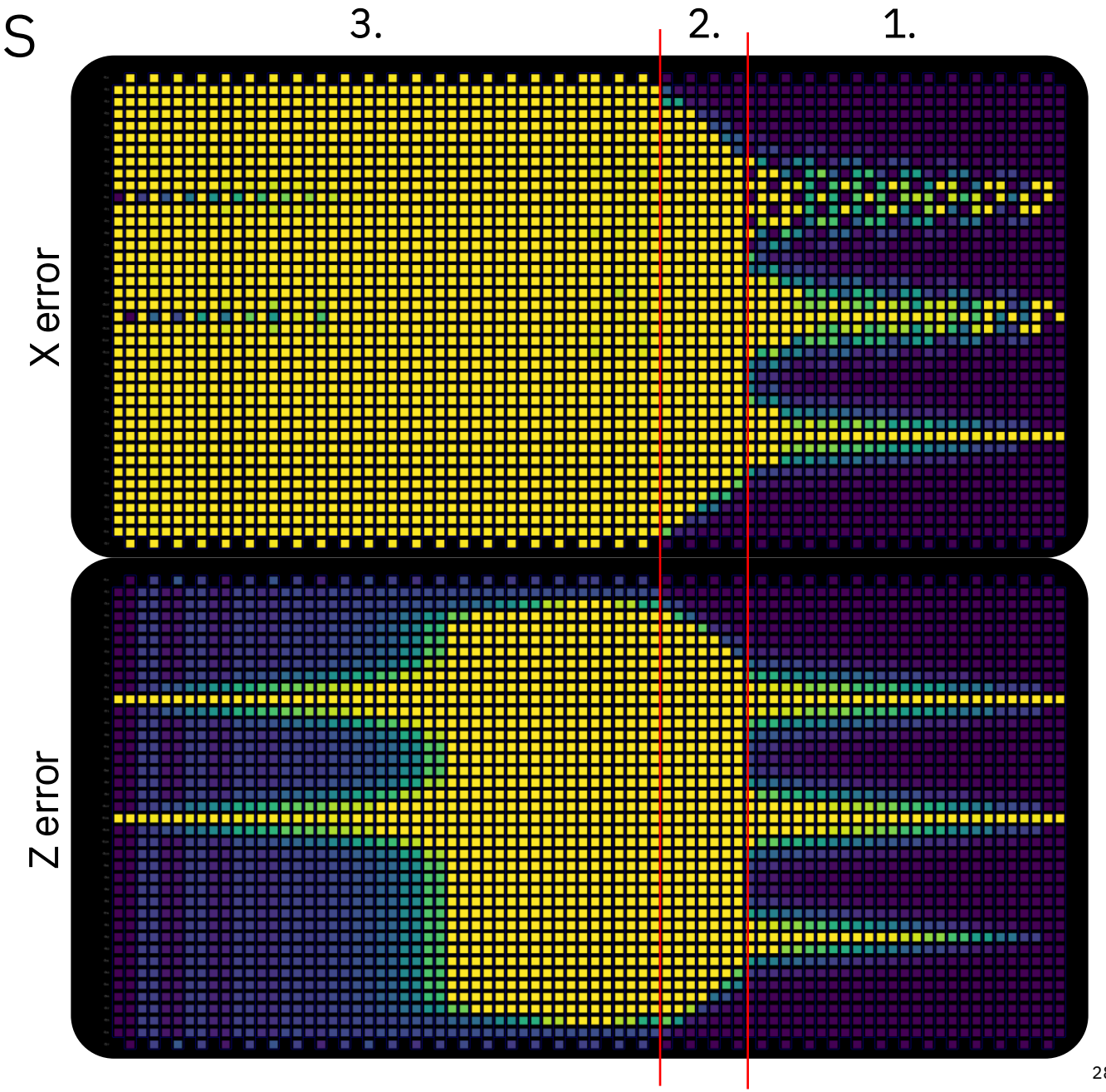
SLC bounds are computed in 3 steps:

1. Forward unequal time commutator bounds
2. Speed-limit tightening of forward bounds
3. Backward unequal time commutator bounds

Step 1 is the bottleneck due to the computationally intensive Pauli propagation of the non-Clifford circuit components.

We can scale the classical resources to our needs:

- Starting with 8 parallel threads and 30 minutes



Computing the SLC bounds

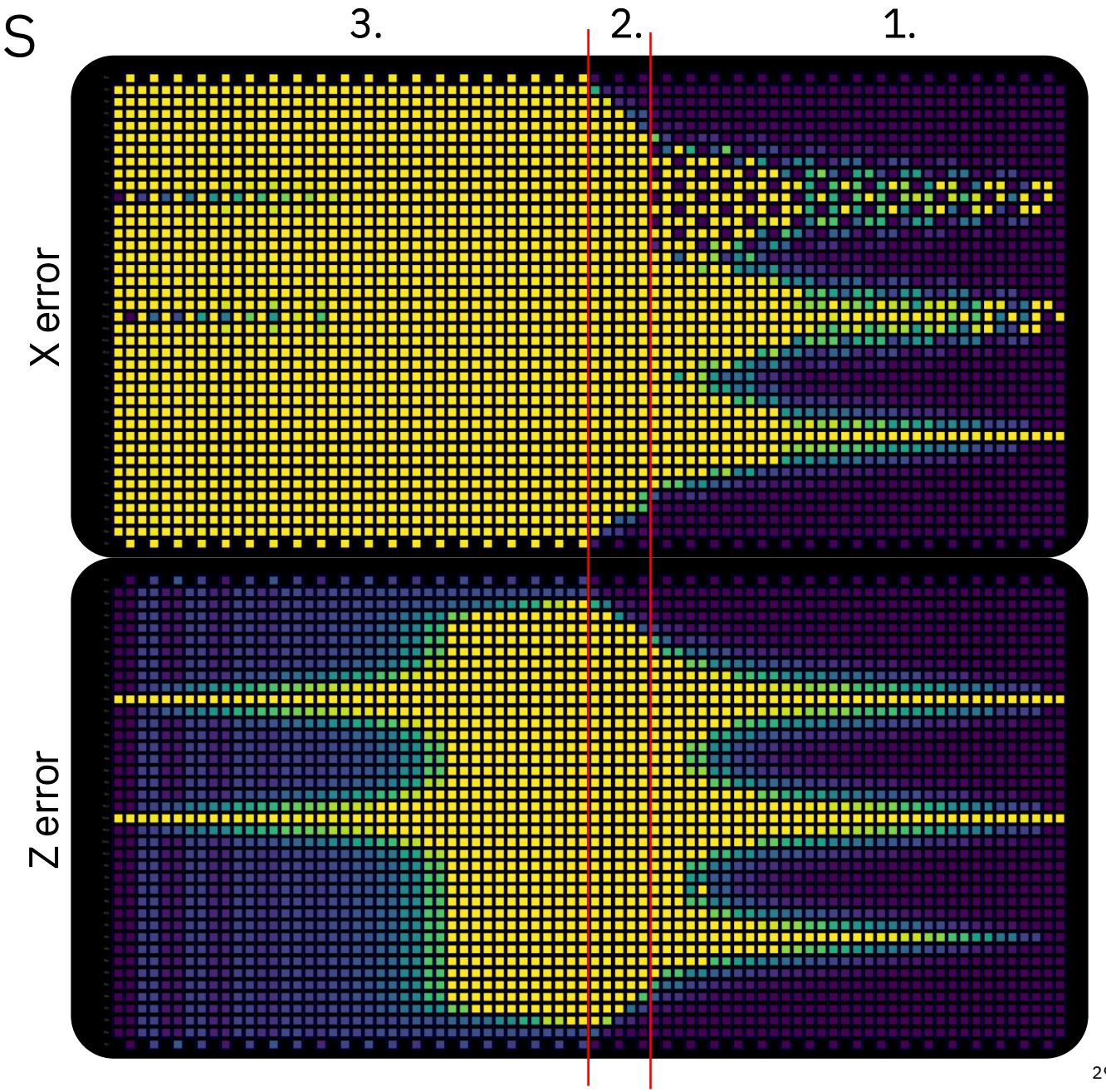
SLC bounds are computed in 3 steps:

1. Forward unequal time commutator bounds
2. Speed-limit tightening of forward bounds
3. Backward unequal time commutator bounds

Step 1 is the bottleneck due to the computationally intensive Pauli propagation of the non-Clifford circuit components.

We can scale the classical resources to our needs:

- Starting with 8 parallel threads and 30 minutes
- Increasing the timeout to 1 hour



Computing the SLC bounds

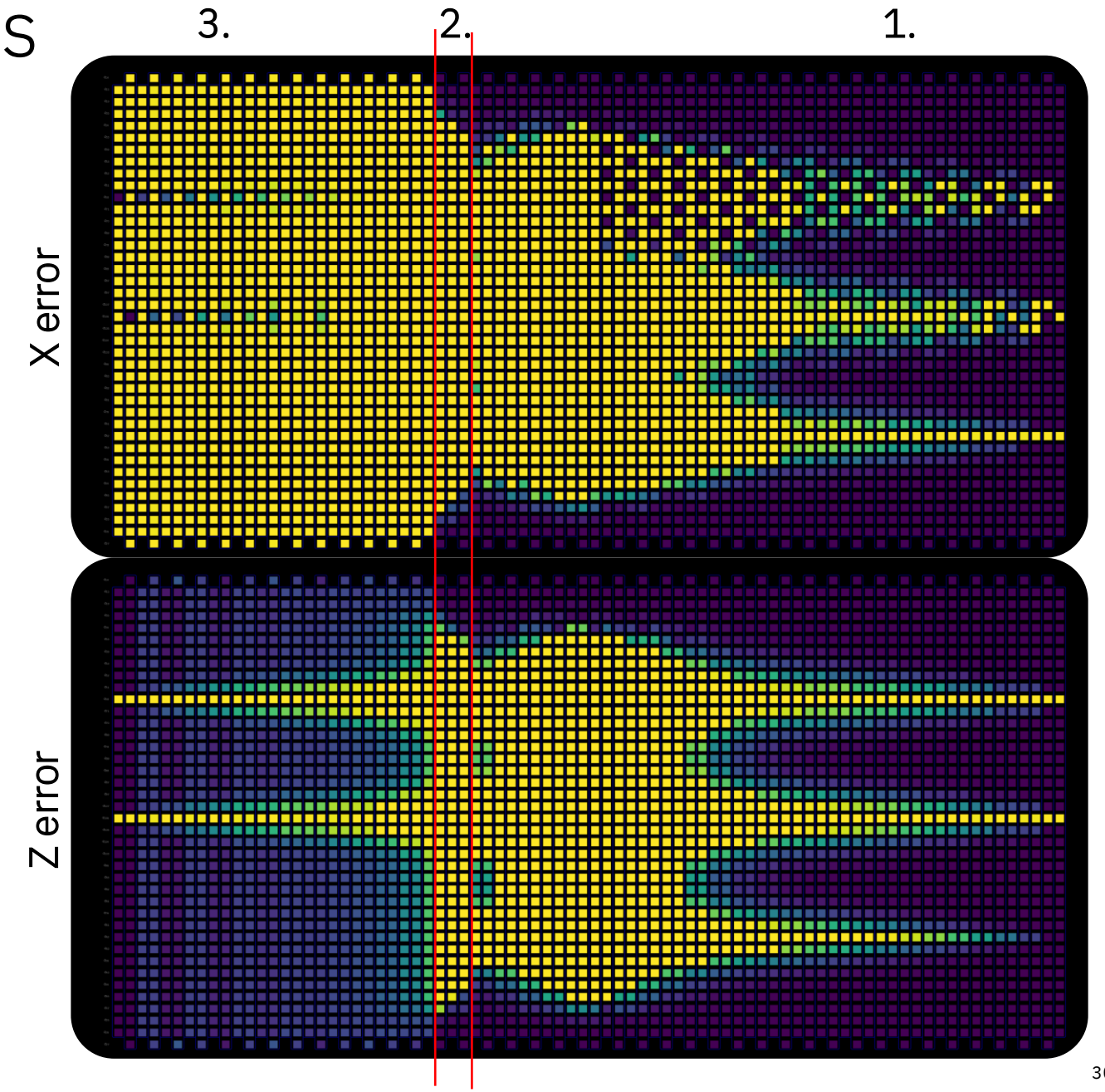
SLC bounds are computed in 3 steps:

1. Forward unequal time commutator bounds
2. Speed-limit tightening of forward bounds
3. Backward unequal time commutator bounds

Step 1 is the bottleneck due to the computationally intensive Pauli propagation of the non-Clifford circuit components.

We can scale the classical resources to our needs:

- Starting with 8 parallel threads and 30 minutes
- Increasing the timeout to 1 hour
- Using 128 parallel threads but 30 minutes again



Computing the SLC bounds

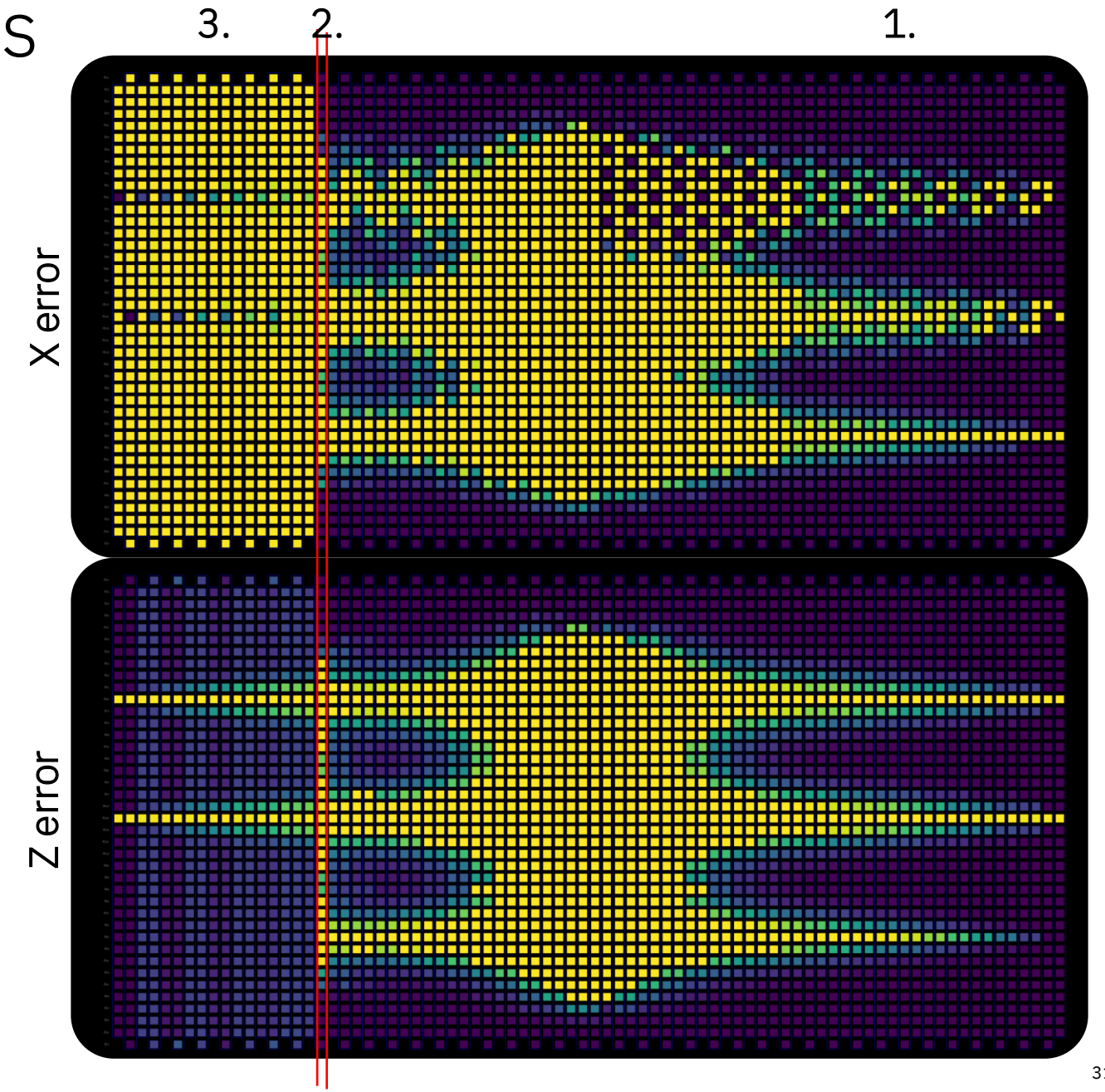
SLC bounds are computed in 3 steps:

1. Forward unequal time commutator bounds
2. Speed-limit tightening of forward bounds
3. Backward unequal time commutator bounds

Step 1 is the bottleneck due to the computationally intensive Pauli propagation of the non-Clifford circuit components.

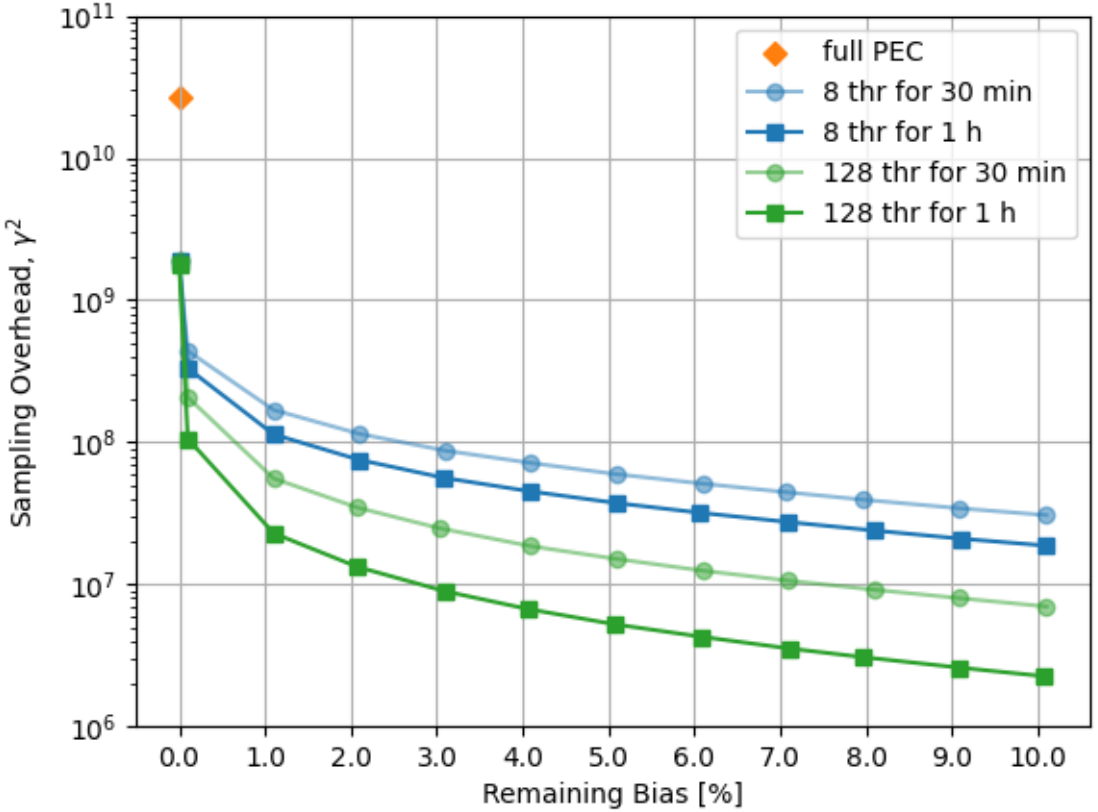
We can scale the classical resources to our needs:

- Starting with 8 parallel threads and 30 minutes
- Increasing the timeout to 1 hour
- Using 128 parallel threads but 30 minutes again
- Increasing the timeout to 1 hour again



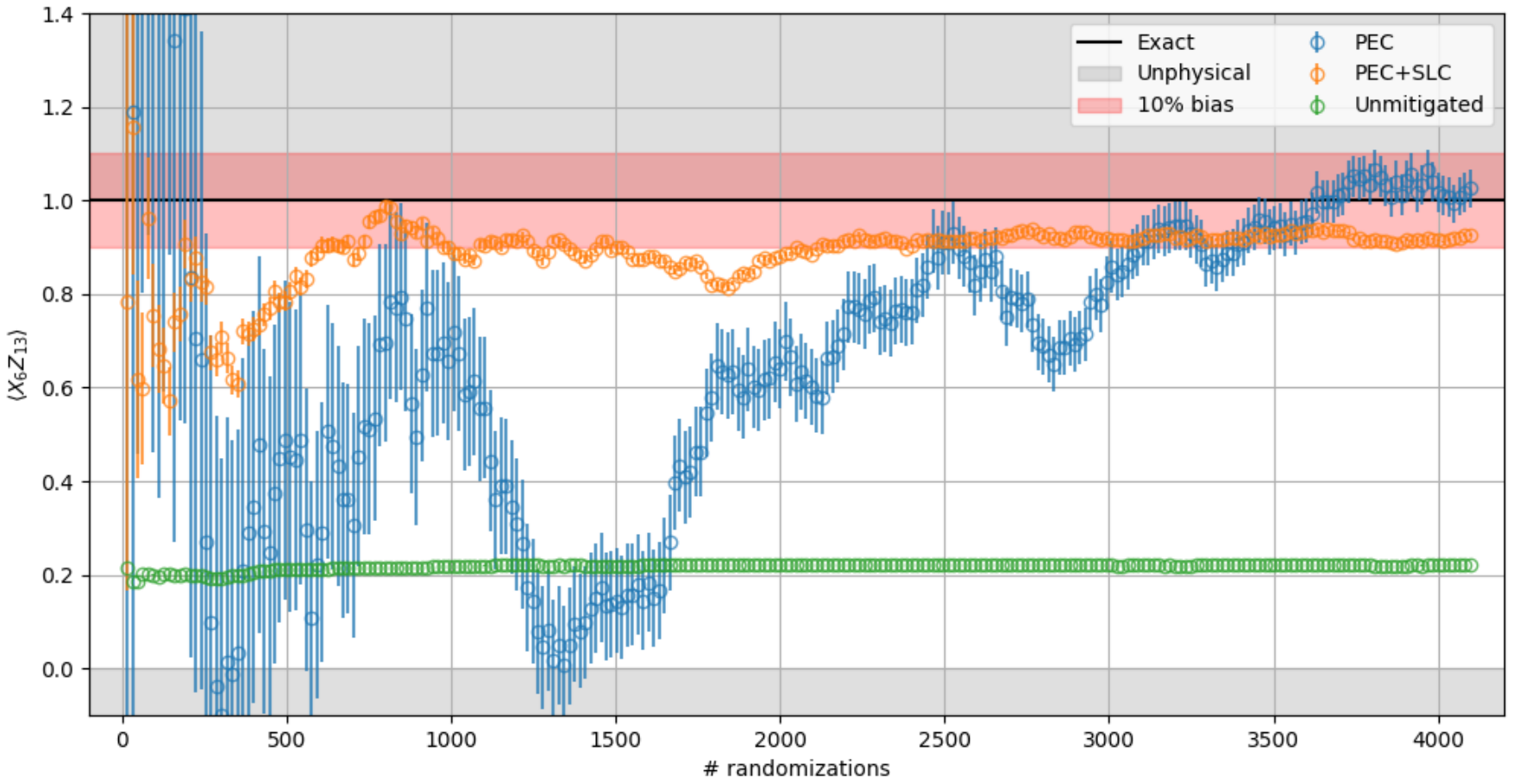
Sampling Overhead Reduction

- SLCs provide systematic means to reduce the sampling overhead of PEC.
- Even with just the computational resources equivalent to a laptop, this reduction can amount to multiple orders of magnitude.
- With more significant classical computational resources and the willingness to accept a residual bias in the expectation value, the sampling overhead can be brought within reasonable range.



SLC Results

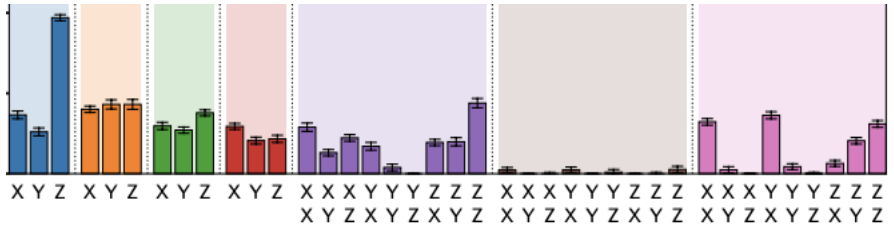
- Equivalent 20q circuit, $\theta_X = \frac{\pi}{4}$
- Observable: $X_6 Z_{13}$
- Full PEC γ^2: 2352.4
- Tolerated bias: 10%
- PEC+SLC γ^2: 145.3
- 64 shots per randomization
- TREX readout-error mitigation
- Qubit-based post-selection



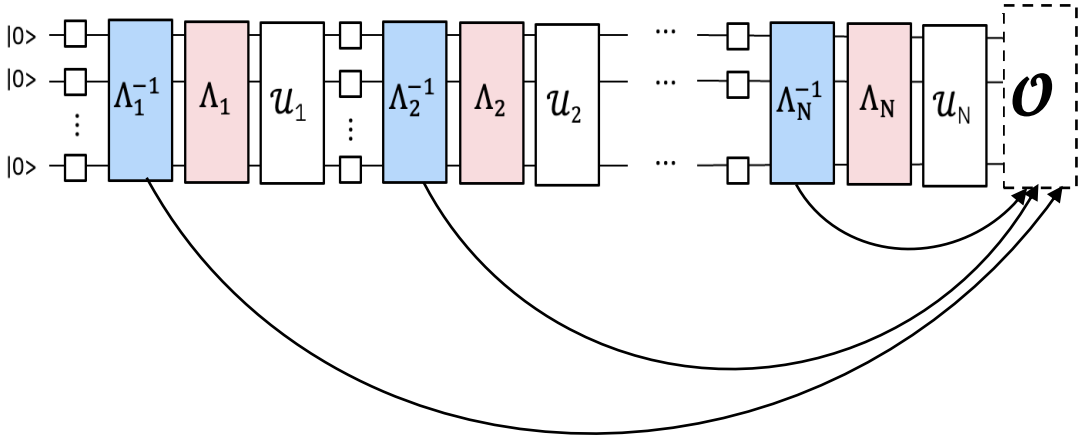
What is propagated noise absorption?

A technique for mitigating gate noise by propagating noise terms forward through your circuit and updating the observable you measure

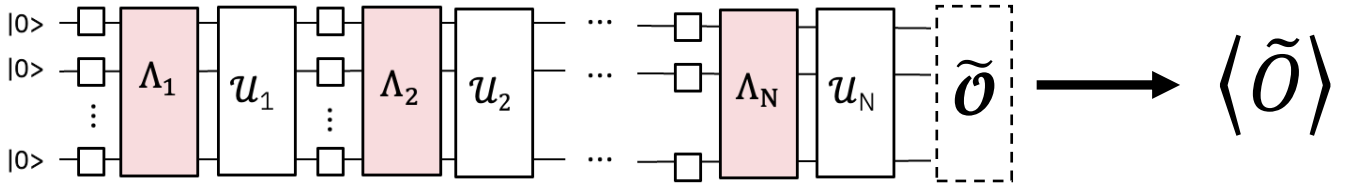
Learn the noise model



Propagate the observable through the anti-noise channel.

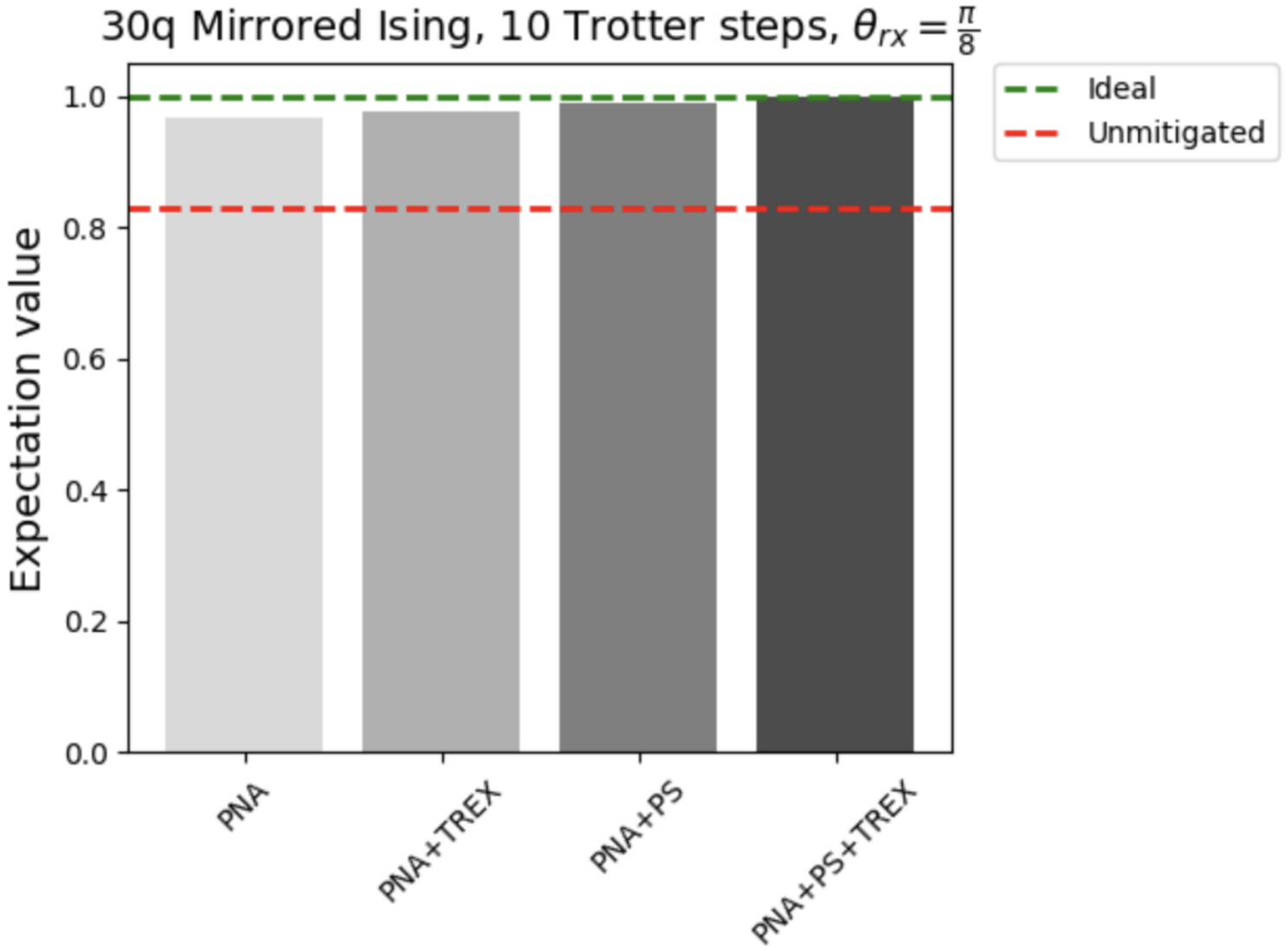


Measure the noise-mitigating observable on noisy hardware



PNA Results

- Equivalent 30q circuit, $\theta_X = \frac{\pi}{8}$
- Observable: $\frac{1}{N}\sum_{i=1}^{N} Z_i$
- 64 shots per randomization
- 10k Max Observables Terms
- 10k Max Err Terms
- TREX readout-error mitigation
- Qubit-based post-selection



Agenda

- Overview of Hardware Noise
- Error Mitigation & Suppression Overview
- Overview of Qiskit implementation
- Code Demos
  - Estimator Demo – ZNE gate folding
  - Samplomatic Demo – SLC, PNA
- Q&A

Quick Overview of Different Capabilities

Sampler Primitive	Estimator Primitive	Samplomatic
Readout Error Mitigation:	Readout Error Mitigation:	Readout Error Mitigation:
M3 addon	TREX	TREX
Circuit Error Mitigation:	Circuit Error Mitigation:	Circuit Error Mitigation:
None	ZNE (gate folding + PEA)	SLC-PEC, PEC, PNA, (ZNE)

*Dynamical Decoupling and Pauli Twirling Possible Across All

Matrix-free Measurement Mitigation (M3) with Sampler

Readout error mitigation for the Sampler primitive using M3

- Qiskit SDK v2.1 or later, with visualization support
- Qiskit Runtime v0.41 or later (`pip install qiskit-ibm-runtime`)
- M3 Qiskit addon v3.0 (`pip install mthree`)

```
job = run_sampler(backend, isa_circuit, NUM_SHOTS)
result = job.result()
pub_result = result[0]
counts = pub_result.data.meas.get_counts()
```

Run your job and get counts

```
# retrieve the final qubit mapping so mthree knows which qubits to calibrate
qubit_mapping = mthree.utils.final_measurement_mapping(isa_circuit)

# submit jobs for readout error calibration
mit = mthree.M3Mitigation(backend)
mit.cals_from_system(qubit_mapping, rep_delay=None)
```

Generate your mitigation matrix

```
# mitigate readout error
quasis = mit.apply_correction(counts, qubit_mapping)
```

Apply readout error mitigation

https://quantum.cloud.ibm.com/docs/en/tutorials/readout-error-mitigation-sampler

Twirled Readout Error eXtinction (TREX) with Estimator

Options	Sub-options	Sub-sub-options	Choices	Default
resilience	measure_mitigation		`True`/`False`	`True`
	measure_noise_learning	num_randomizations		`32`
		shots_per_randomization		`'auto'`

```
from qiskit_ibm_runtime import EstimatorOptions

options = EstimatorOptions(default_shots=1024, optimization_level=0, resilience_level=0)

## Configure TREX
options.resilience.measure_mitigation = True
options.resilience.measure_noise_learning.num_randomizations = 32
options.resilience.measure_noise_learning.shots_per_randomization = 'auto'

options.twirling.enable_measure = True  # Automatically set by TREX
```

https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.ResilienceOptionsV2

Zero Noise Extrapolation (ZNE) with Estimator

Options	Sub-options	Sub-sub-options	Choices	Default
resilience	zne_mitigation		`True`/`False`	`False`
	zne	noise_factors		`(1, 1.5, 2) for PEA, and (1, 3, 5) otherwise`
		amplifier	`'gate_folding'`/ `'gate_folding_front'`/ `'gate_folding_back'`/ `'pea'`	`gate_folding`
		extrapolator	`'exponential'`/ `'linear'`/ `'double_exponential'`/ `'polynomial_degree_(1 <= k <= 7)'`	(`'exponential'`, `'linear'`)

```python
from qiskit_ibm_runtime import EstimatorOptions

options = EstimatorOptions(default_shots=1024, optimization_level=0, resilience_level=0)

## Configure ZNE
options.resilience.zne_mitigation = True
options.resilience.zne.noise_factors = (1, 3, 5)   [Can have fractional noise factors, e.g. (1, 1.5, 2)]
options.resilience.zne.amplifier = 'gate_folding'
options.resilience.zne.extrapolator = ('exponential', 'linear')
```

Choosing the right noise factors and extrapolator is tricky!

https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.ResilienceOptionsV2

Zero Noise Extrapolation (ZNE) with Estimator

PEA – Noise learner

```
from qiskit_ibm_runtime import EstimatorOptions

options = EstimatorOptions(default_shots=1024, optimization_level=0, resilience_level=0)

## Configure ZNE
options.resilience.zne_mitigation = True
options.resilience.zne.noise_factors = (1, 3, 5)
options.resilience.zne.amplifier = 'pea'
options.resilience.zne.extrapolator = ('exponential', 'linear')
```

```
options.resilience.layer_noise_learning.max_layers_to_learn = 4
options.resilience.layer_noise_learning.num_randomizations = 32
options.resilience.layer_noise_learning.shots_per_randomization = 128
options.resilience.layer_noise_learning.layer_pair_depths = (0, 1, 2, 4, 16, 32)
```

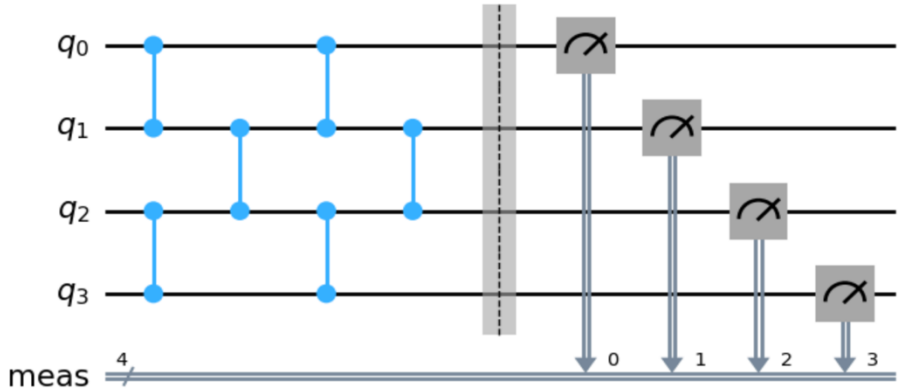
https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.LayerNoiseLearningOptions

Building a PEC workflow with Samplomatic

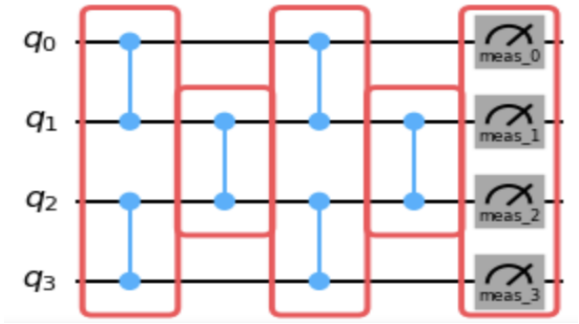
Step 1

Define a target circuit and box it up with annotations

Target circuit



Boxed circuit



qiskit

```
target_circuit = QuantumCircuit(...)
target_obs = SparseObservable(...)

pm = generate_preset_pass_manager(...)
```

samplomatic

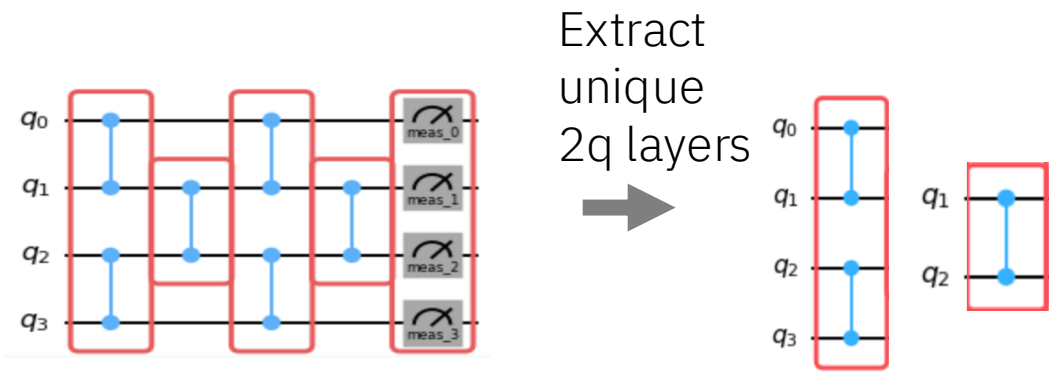
```
boxing_pm = generate_boxing_pass_manager(
    enable_gates=True,
    enable_measures=True,
    twirling_strategy="active",
    inject_noise_targets="gates",
    inject_noise_strategy=
        "individual_modification",
    measure_annotations="change_basis",
)
```

```
pm.post_scheduling = boxing_pm

boxed_isa_circuit = pm.run(target_circuit)
```

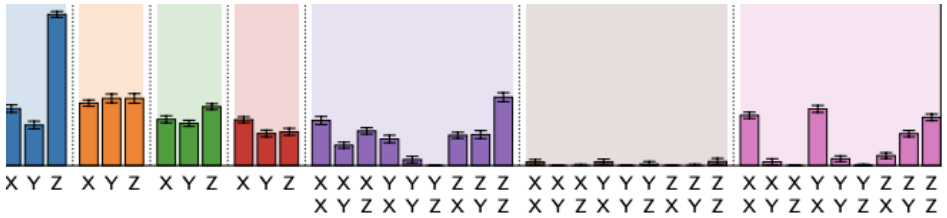
Building a PEC workflow with Samplomatic

Step 2

Learn the noise models on desired boxes.



Learn noise model



samplomatic

```
unique_layers = find_unique_box_instructions(
    boxed_isa_circuit
)
```

qiskit_ibm_runtime

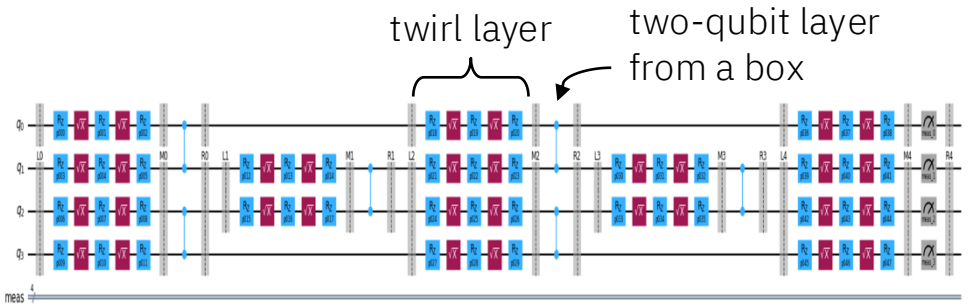
```
noise_learner = NoiseLearnerV3(
    session,
    NoiseLearnerV3Options(
        num_randomizations=64,
        shots_per_randomization=128,
        layer_pair_depths=[1, 2, ...],
        post_selection={...}
    )
)

noise_job = noise_learner.run(unique_layers)
result = noise_job.result()
```

Building a PEC workflow with Samplomatic

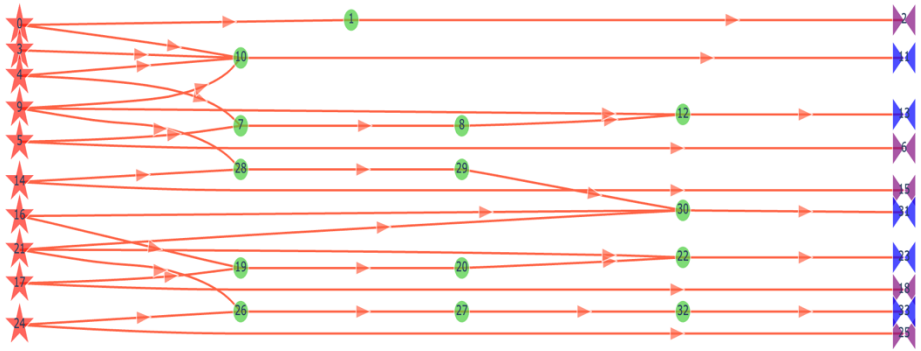
Step 3

Build the template circuit and samplex from the boxed circuit and prepare the final measurement bases.

template circuit



samplex



samplomatic

```
template, samplex = samplomatic.build(
    boxed_isa_circuit
)
```

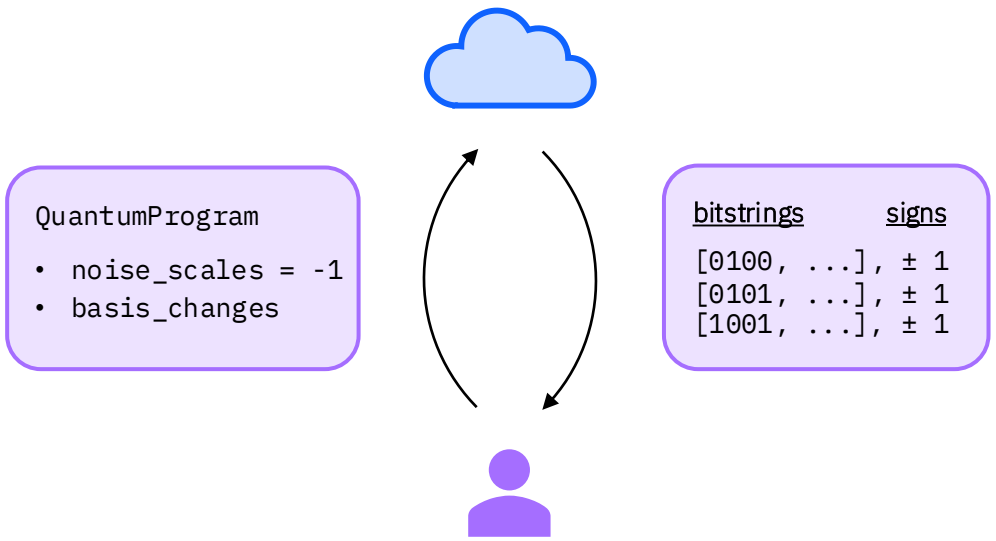
qiskit_addon_utils

```
post_selection_pm = PassManager(
    [
        AddSpectatorMeasures(...),
        AddPostSelectionMeasures(...),
    ]
)

template = post_selection_pm.run(template)


bases, reverser = get_measurement_bases(
    target_obs
)
```

Building a PEC workflow with Samplomatic

Step 4

Execute QuantumProgram defined by template circuit, samplex, and samplex inputs.



samplomatic

```
inputs = samplex.inputs().bind(
    noise_scales=-1,
    basis_changes={“basis”: bases[0]},
)
```

qiskit_ibm_runtime

```
program = QuantumProgram(
    shots=64, noise_maps={...}
)

program.append(
    circuit=template,
    samplex=samplex,
    samplex_arguments=inputs,
    shape=(1024,),
    chunk_size=128,
)

exec_job = Executor(session).run(program)
```

Building a PEC workflow with Samplomatic

Step 5

Postprocess results of execution to reconstruct an expectation value.

bitstrings	signs
[0100, ...]	± 1
[0101, ...]	± 1
[1001, ...]	± 1

→ $\langle \mathcal{O} \rangle$

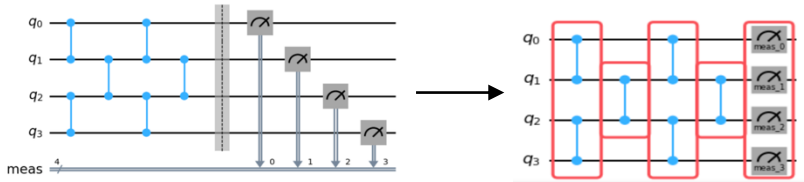
qiskit_ibm_runtime

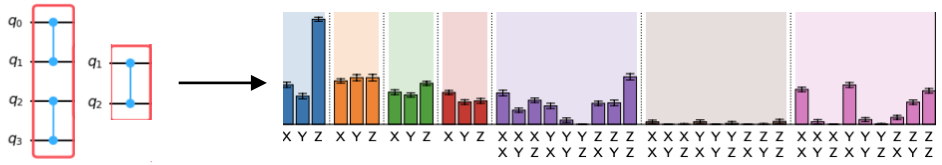
```
result = exec_job.result()[0]
```

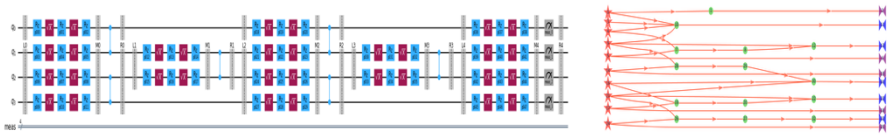
qiskit_addon_utils

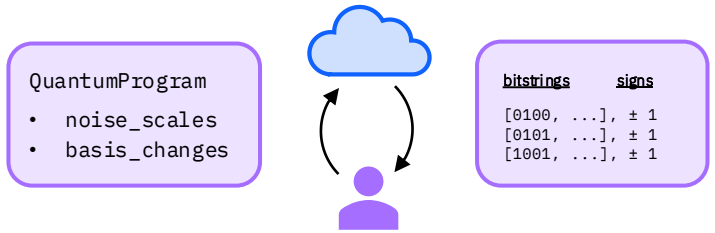
```
gamma = gamma_from_noisy_boxes(...)
mask = PostSelector(...).compute_mask(result)
trex = trex_factors(...)

exp_val, var = executor_expectation_values(
    result[“meas”],
    reverser,
    meas_basis_axis=...,
    avg_axis=...,
    meas_flips=result[“measurement_flips”],
    pec_signs=result[“pauli_signs”],
    pec_gamma=gamma,
    postselect_mask=mask,
    rescale_factors=trex,
)
```

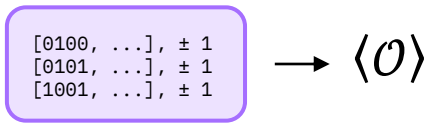
Building a PEC workflow with Samplomatic





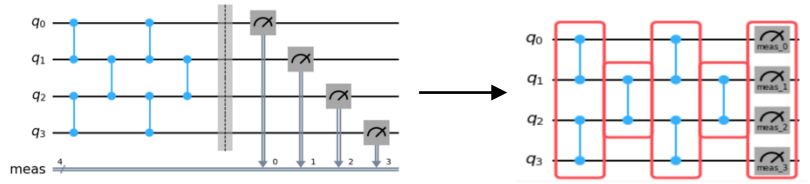


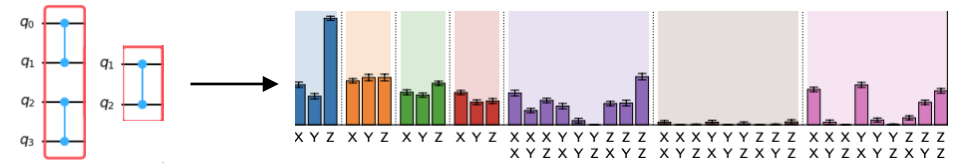


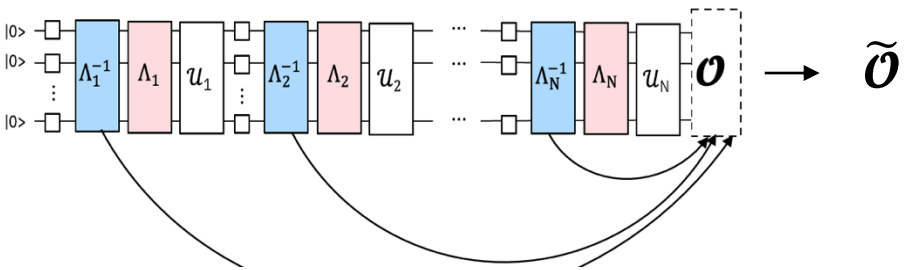


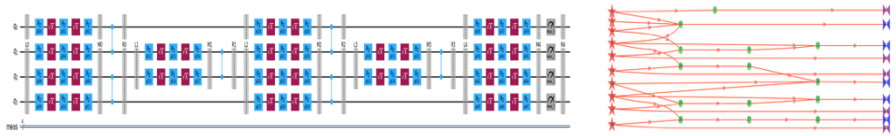

```
boxing_pm = generate_boxing_pass_manager(...)
boxed_circ = boxing_pm.run(target_circ)

layers = find_unique_box_instructions(boxed_circ)
noise = NoiseLearnerV3(...).run(layers).result()

template, samplex = samplomatic.build(boxed_circ)

program = QuantumProgram(..., noise)
program.append(template, samplex, ...)
results = Executor(...).run(program).result()

exp_val, var = executor_expectation_values(results)
```

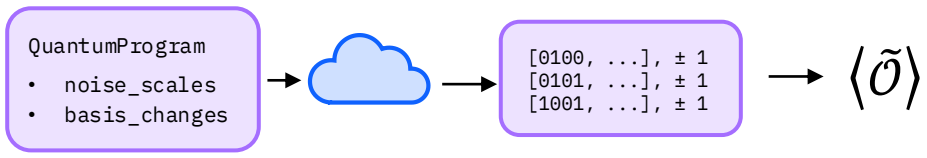
Leveraging Samplomatic framework to implement PNA












```
boxing_pm = generate_boxing_pass_manager(...)
boxed_circ = boxing_pm.run(target_circ)


layers = find_unique_box_instructions(boxed_circ)
noise = NoiseLearnerV3(...).run(layers).result()
```

qiskit_addon_pna

```
pna_obs = generate_noise_mitigating_observable(
    boxed_circ,
    obs,
    box_to_noise_map,
)
bases, reverser = get_measurement_bases(pna_obs)

template, samplex = samplomatic.build(boxed_circ)
inputs = sample.inputs().bind(...,
    basis_changes=bases
)

program = QuantumProgram(..., noise)
program.append(template, samplex, inputs, ...)
results = Executor(...).run(program).result()
exp_val, var = expectation_values(results, reverser)
```

Agenda

- Overview of Hardware Noise
- Error Mitigation & Suppression Overview
- Overview of Qiskit implementation
- Code Demos
  - Estimator Demo - ZNE gate folding
  - Samplomatic Demo - SLC, PNA
- Q&A

IBM


IBM