Qiskit Global Summer School 2024

Execution on noisy quantum hardware

Fighting errors before fault tolerance

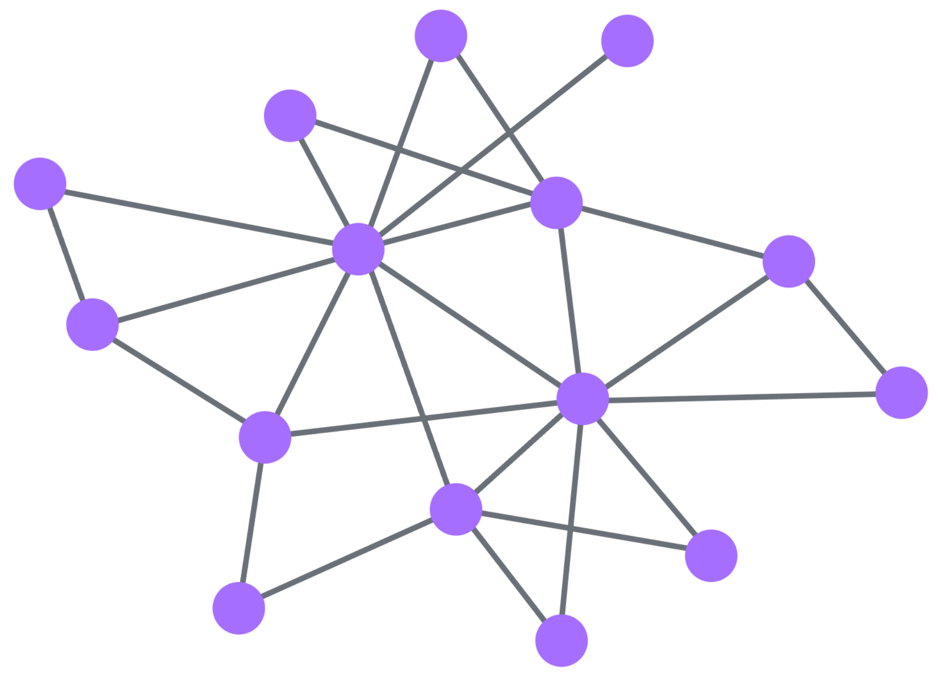
Pedro Rivero
Quantum Algorithm Engineering Technical Lead
IBM Quantum

Qiskit Patterns

The anatomy of a quantum algorithm

Step 1

Map classical inputs to a quantum problem



Step 2

Optimize problem for quantum execution.

```
PassManager([UnitarySynthesis(),
             BasisTranslator(),
             EnlargeWithAncilla(),
             AISwap(),
             Collect1qRuns(),
             Optimize1qGates(),
             Collect2qBlocks(),
             ConsolidateBlocks()])
```

Step 3

Execute using Qiskit Runtime Primitives.

Sampler

000101...,
110110...

circuit($\vec{\theta}$)

bit-strings

Estimator

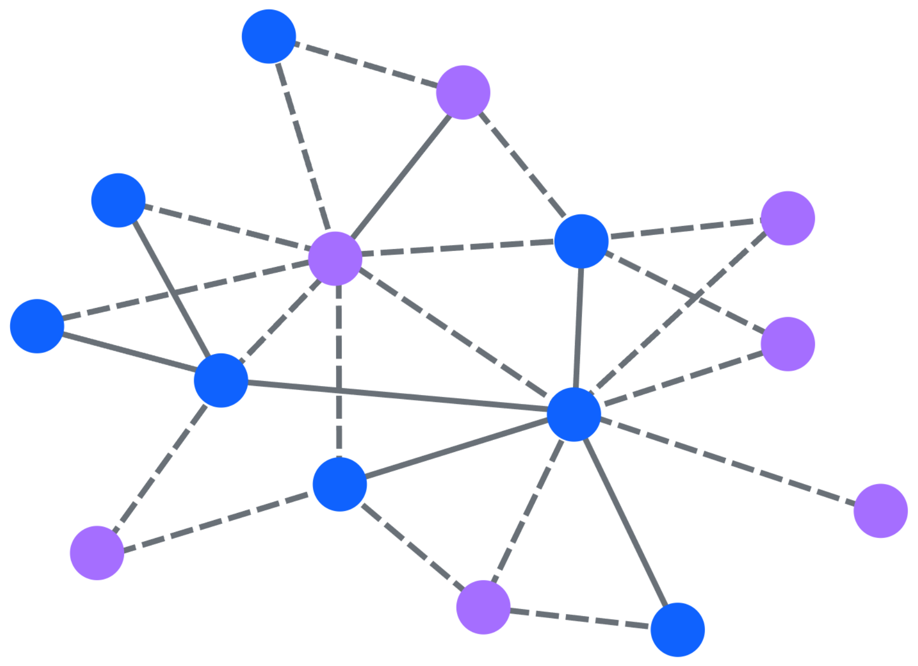
$\langle O \rangle$

circuit($\vec{\theta}$) + observable $\hat{O}$

expectation value

Step 4

Analyze result in classical format.

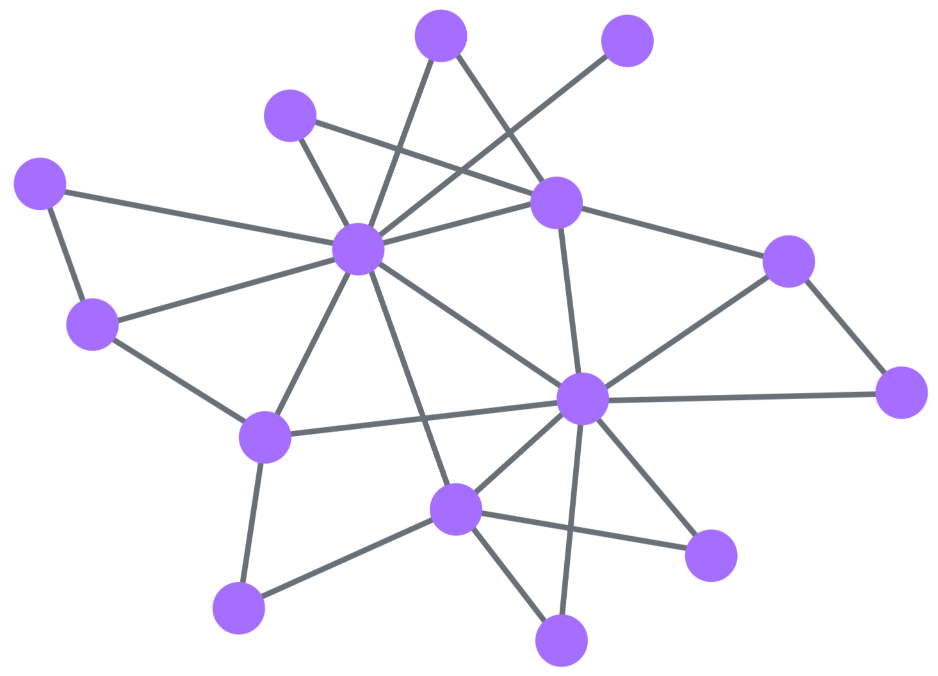


Qiskit Patterns

The anatomy of a quantum algorithm

Step 1

Map classical inputs to a quantum problem



Step 2

Optimize problem for quantum execution.

```
PassManager([UnitarySynthesis(),
             BasisTranslator(),
             EnlargeWithAncilla(),
             AISwap(),
             Collect1qRuns(),
             Optimize1qGates(),
             Collect2qBlocks(),
             ConsolidateBlocks()])
```

Step 3

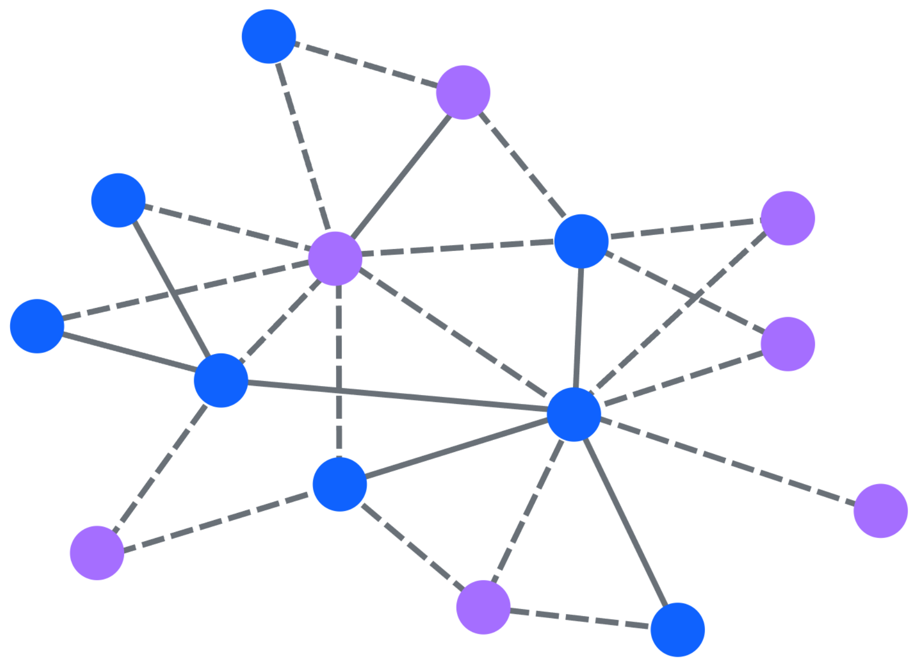
Execute using Qiskit Runtime Primitives.

circuit($\vec{\theta}$) → Sampler → 000101...,
110110...
bit-strings

circuit($\vec{\theta}$) + observable $\hat{O}$ → Estimator → $\langle O \rangle$
expectation value

Step 4

Analyze result in classical format.



Noise in quantum systems

Quantum computers are noisy *"in every way possible"*

Fault tolerance is still unfeasible today

We need interim ways to recover a better signal:

1. Limit the amount of noise
2. Clean the signal by filtering the noise out

This is accomplished by:

1. Run modified noisy quantum computations
2. Process collected outputs on a classical computer
3. Compute an improved result

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

Fighting noise in quantum systems

Suppression

- Reduce or avoid the impact of errors
- Before or during execution (typically)
- Requires additional *classical* resources

Mitigation

- Filter errors out after they occur
- After or during execution (typically)
- Requires additional *quantum* resources

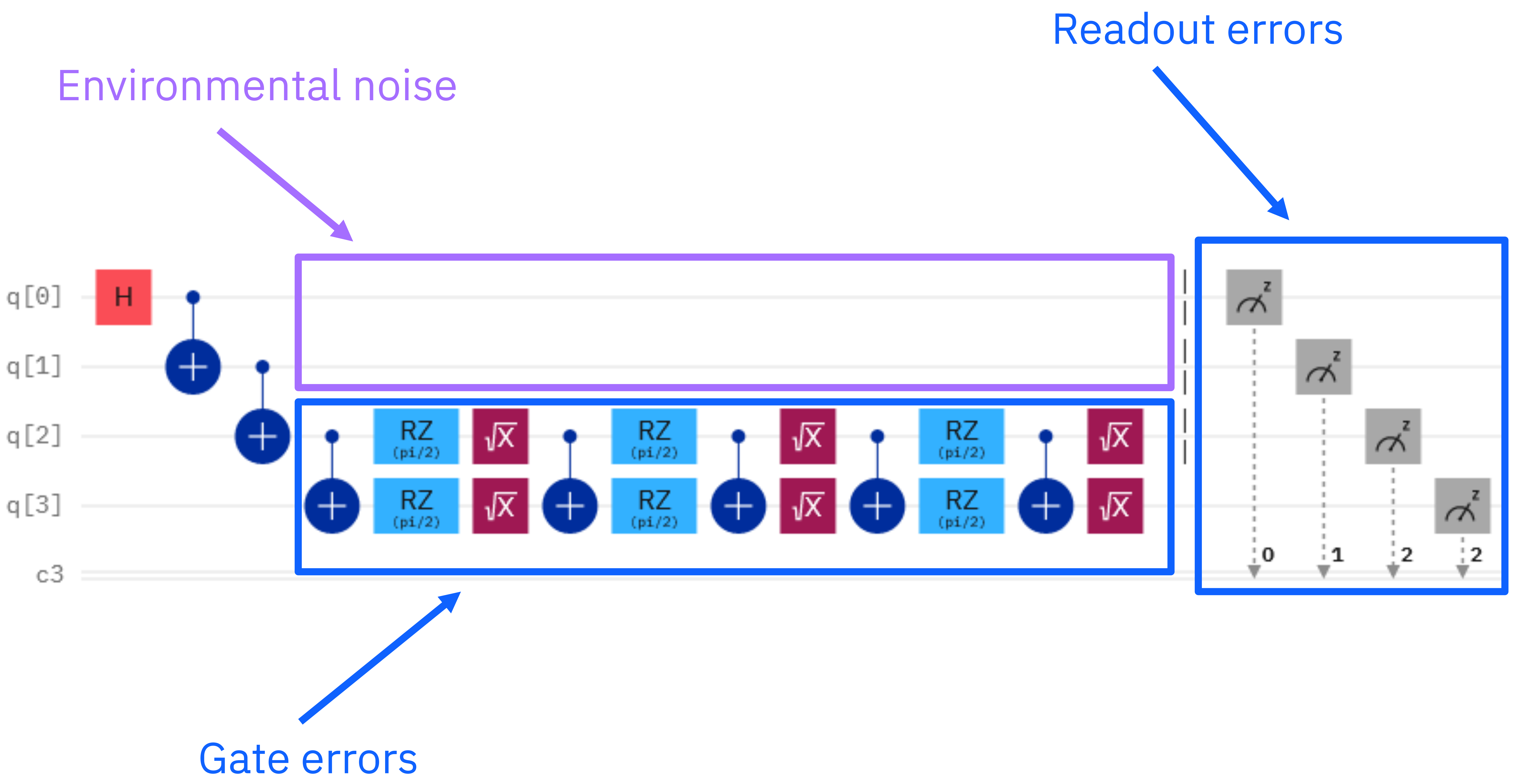
Correction

- Detect and fix errors as they occur
- During execution
- Requires additional *quantum and classical* resources

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

Sources of noise

- SPAM errors:
  related to state preparation and measurement/readout
- Gate errors:
  imperfect operations on qubits
- Environmental noise:
  even if there are no operations on qubits, these are exposed to errors coming from interaction with the environment



Qiskit Runtime

```python
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2, EstimatorV2

service = QiskitRuntimeService()
backend = service.least_busy()

sampler = SamplerV2(backend, options=None)
estimator = EstimatorV2(backend, options=None)

## Baseline configuration
estimator.options.default_shots = sampler.options.default_shots = 1024
estimator.options.optimization_level = 0  # Deactivate circuit optimization
estimator.options.resilience_level = 0  # Deactivate error mitigation
```

✓ 25.0s Python

- Sampler options: https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.SamplerOptions
- Estimator options: https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.EstimatorOptions

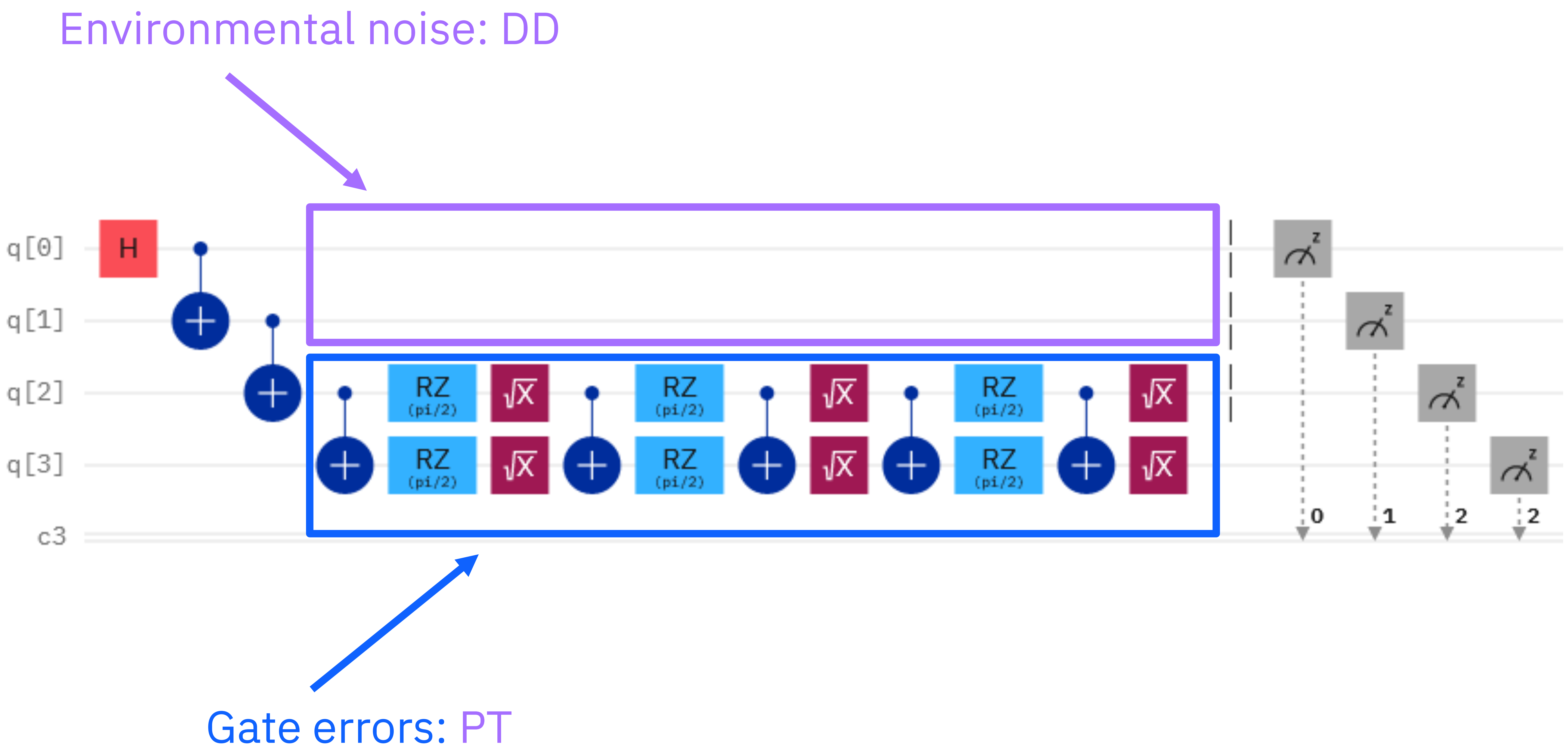
Error suppression

Dynamical decoupling (DD)

Inserts sequences of gates in idling qubits to avoid the effects of cross-talk

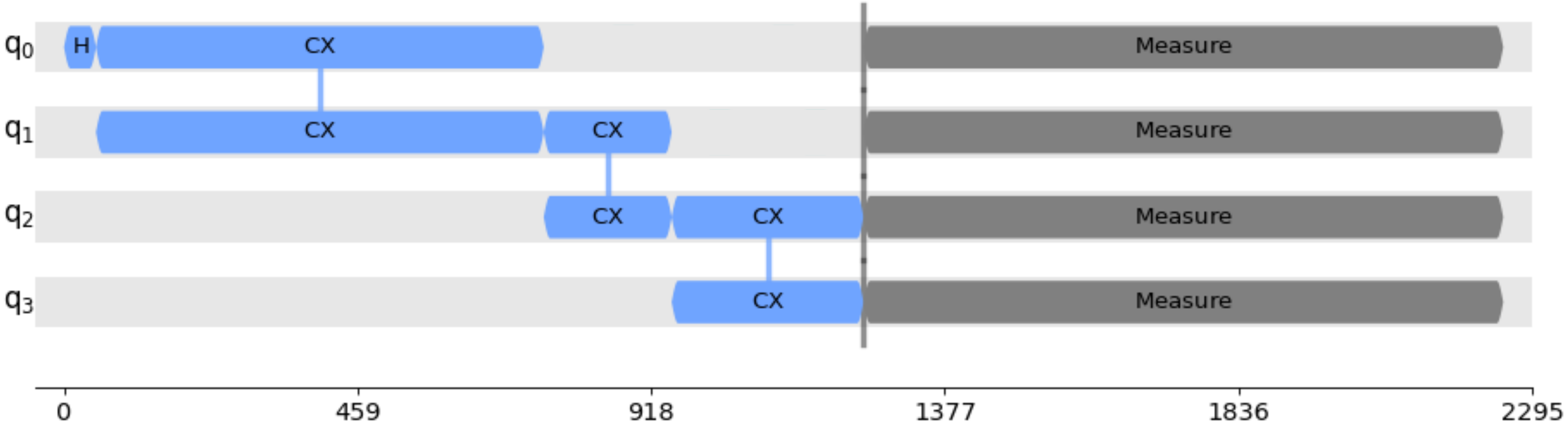
Pauli twirling (PT)

Executes an ensemble of equivalent quantum circuits to alter the structure of the observed noise



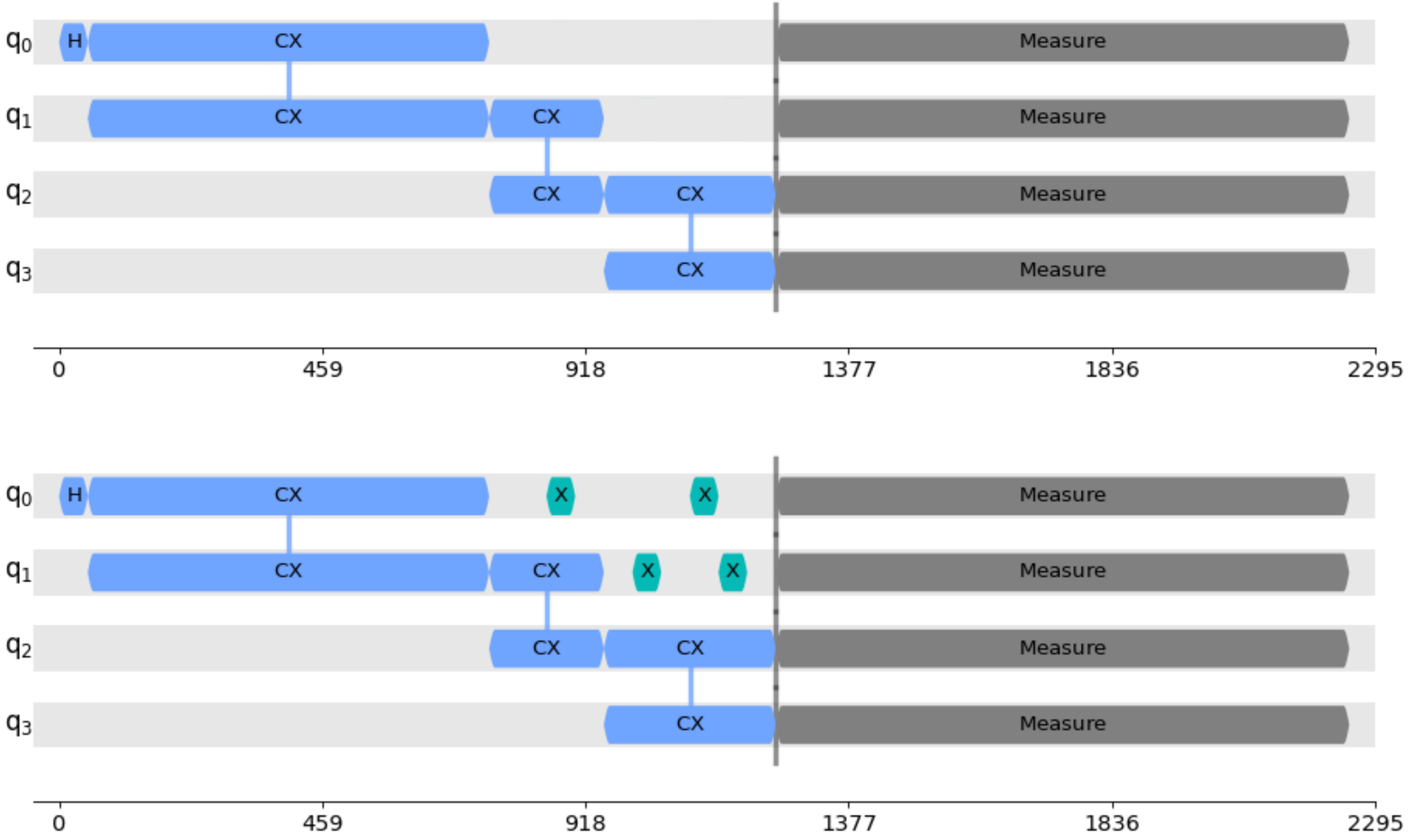
Dynamical decoupling (DD)

- Activity on neighboring qubits can induce noise while idling (i.e. cross-talk)
- Having gates applied to qubits can help suppress this effect
- The introduced gates need to add up to the identity to preserve the underlaying unitary
- These gates will also introduce errors, so there is a balance to be found



Dynamical decoupling (DD)

- Activity on neighboring qubits can induce noise while idling (i.e. cross-talk)
- Having gates applied to qubits can help suppress this effect
- The introduced gates need to add up to the identity to preserve the underlaying unitary
- These gates will also introduce errors, so there is a balance to be found



Dynamical decoupling (DD)

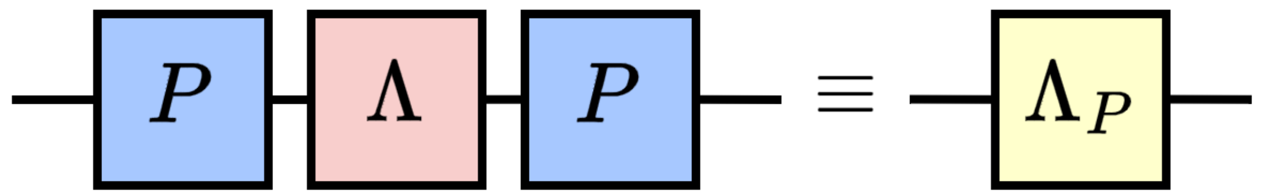
```python
from qiskit_ibm_runtime import SamplerOptions, EstimatorOptions

options = SamplerOptions(default_shots=1024)  # or...
options = EstimatorOptions(default_shots=1024, optimization_level=0, resilience_level=0)

## Configure Dynamical Decoupling
options.dynamical_decoupling.enable = True
options.dynamical_decoupling.sequence_type = 'XX'
options.dynamical_decoupling.extra_slack_distribution = 'middle'
options.dynamical_decoupling.scheduling_method = 'alap'
```

✓ 0.0s Python

Dynamical decoupling options: https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.DynamicalDecouplingOptions

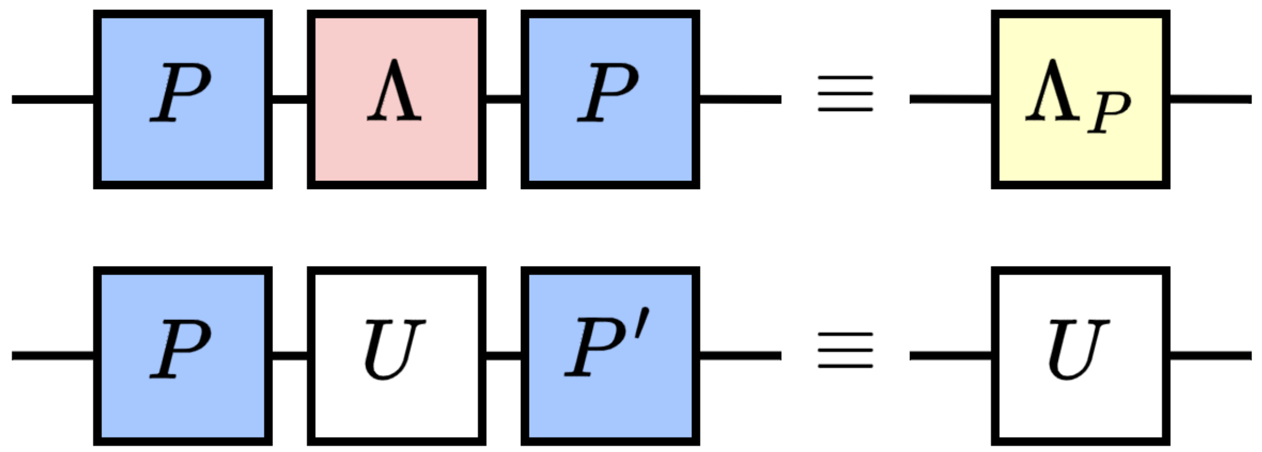
Randomized compiling (twirling)

- Used to convert arbitrary noise channels into other forms of noise
- Executing statistical ensembles of unitarily equivalent circuits
- Pauli Twirling (PT) converts any quantum channel into a Pauli channel.
- Suppresses the impact of coherent noise.
- Noisy results degrade in more predictable ways (useful for ZNE).



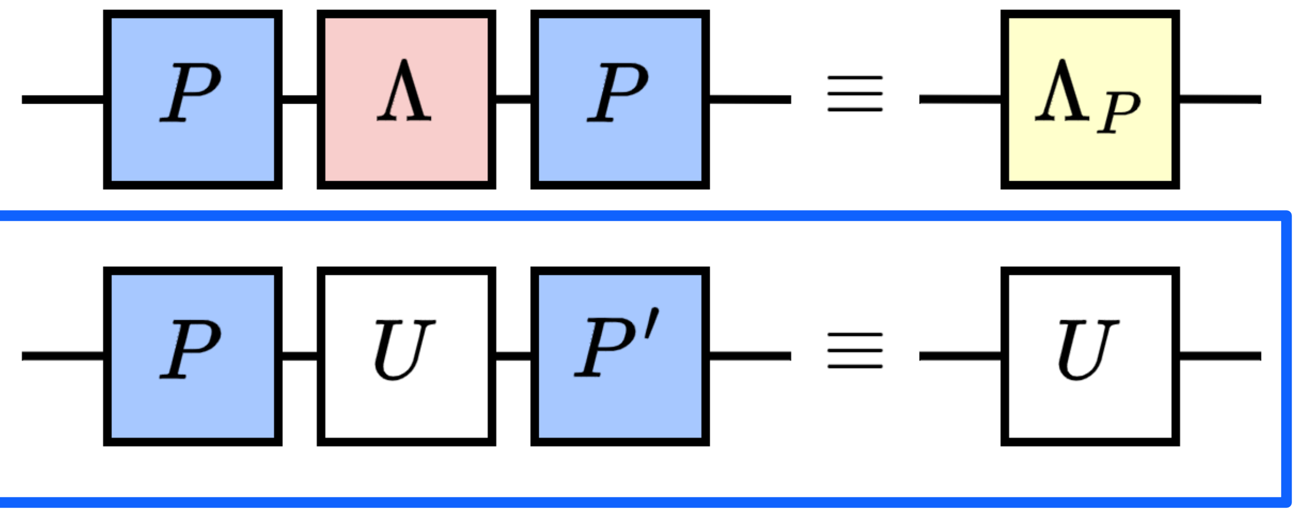
Randomized compiling (twirling)

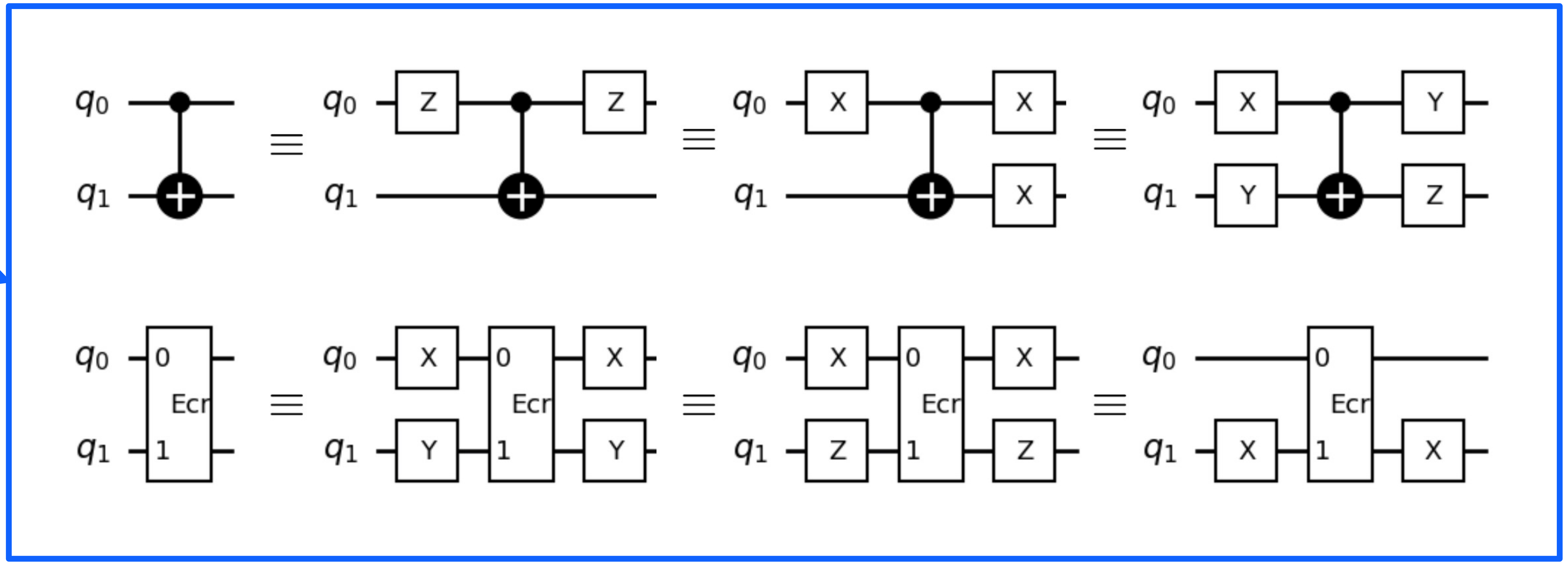
- Used to convert arbitrary noise channels into other forms of noise
- Executing statistical ensembles of unitarily equivalent circuits
- Pauli Twirling (PT) converts any quantum channel into a Pauli channel.
- Suppresses the impact of coherent noise.
- Noisy results degrade in more predictable ways (useful for ZNE).



Randomized compiling (twirling)

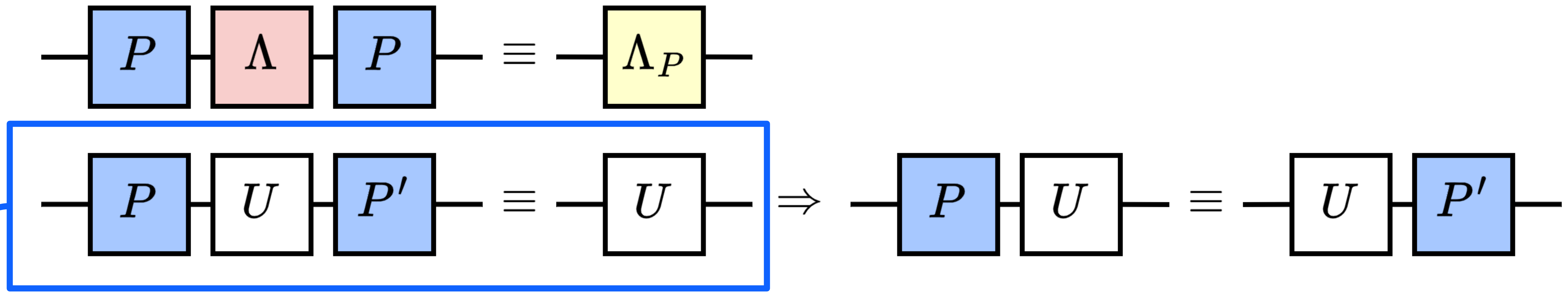
- Used to convert arbitrary noise channels into other forms of noise
- Executing statistical ensembles of unitarily equivalent circuits
- Pauli Twirling (PT) converts any quantum channel into a Pauli channel.
- Suppresses the impact of coherent noise.
- Noisy results degrade in more predictable ways (useful for ZNE).

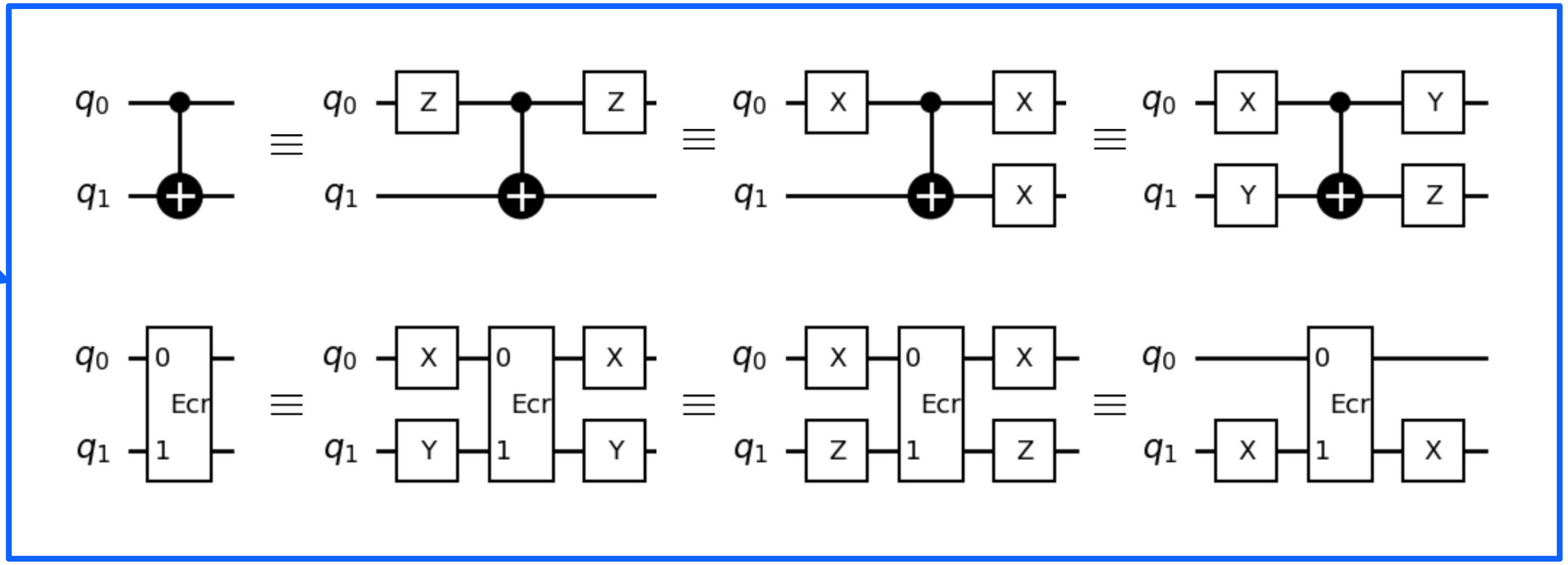




Randomized compiling (twirling)

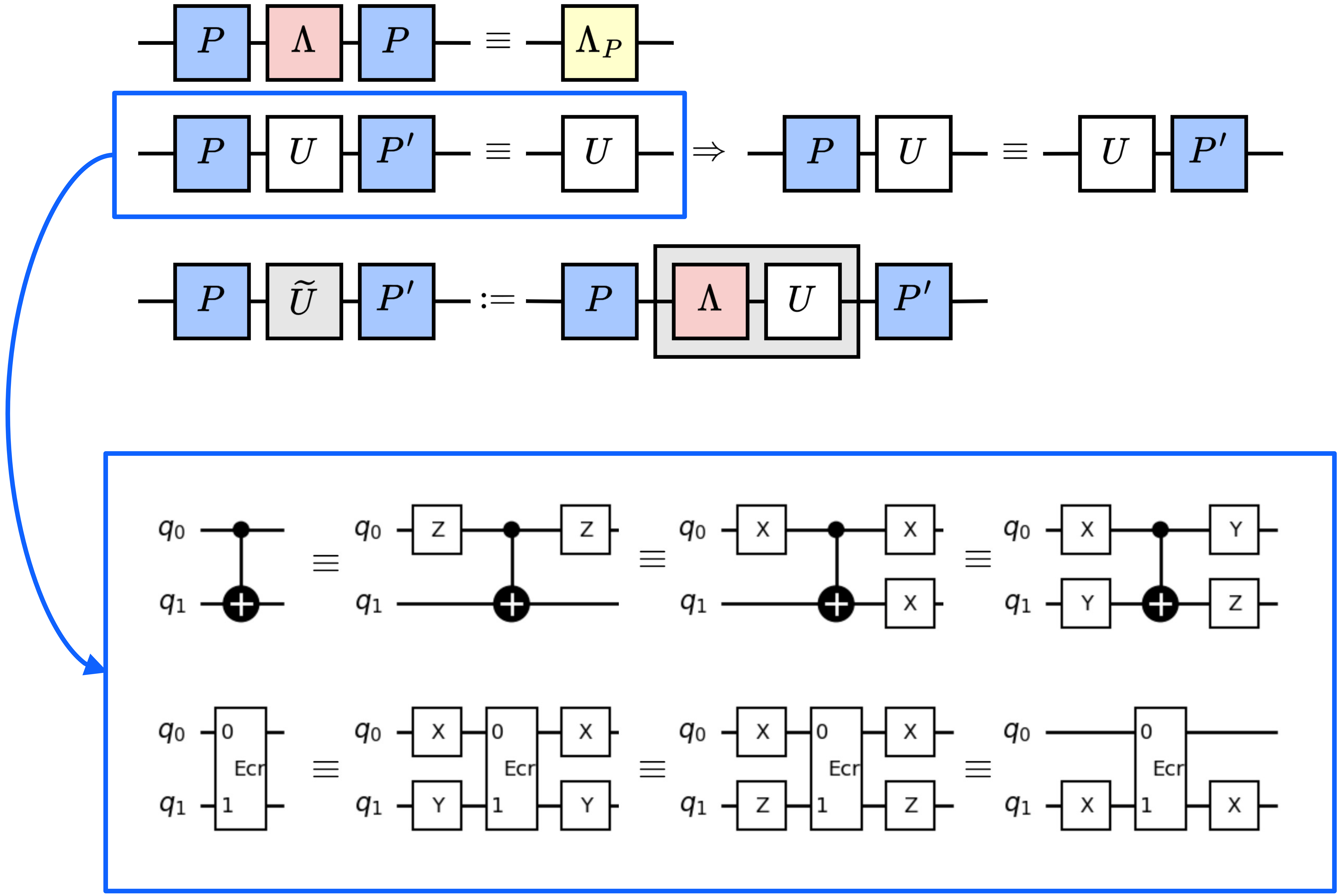
- Used to convert arbitrary noise channels into other forms of noise
- Executing statistical ensembles of unitarily equivalent circuits
- Pauli Twirling (PT) converts any quantum channel into a Pauli channel.
- Suppresses the impact of coherent noise.
- Noisy results degrade in more predictable ways (useful for ZNE).





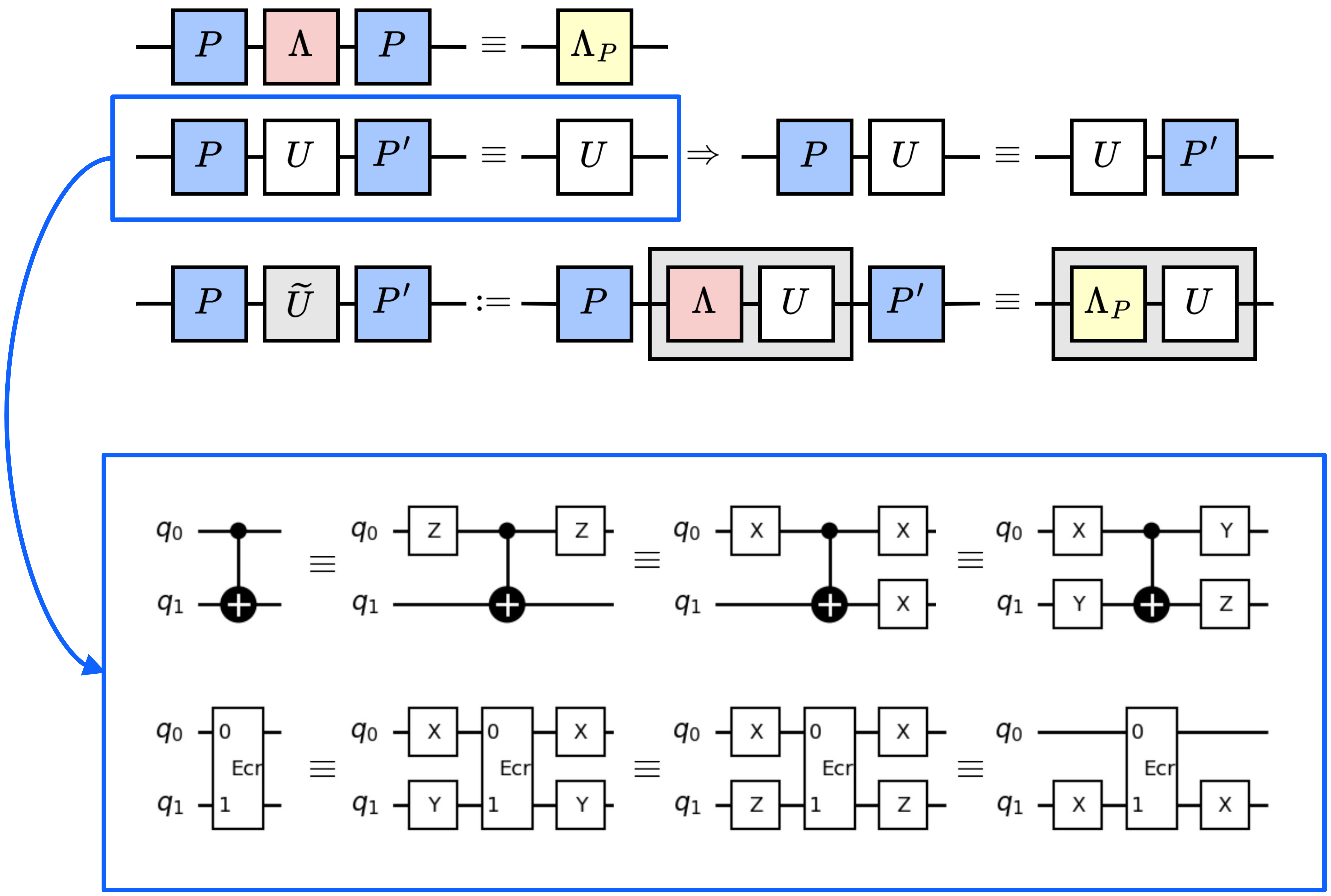
Randomized compiling (twirling)

- Used to convert arbitrary noise channels into other forms of noise
- Executing statistical ensembles of unitarily equivalent circuits
- Pauli Twirling (PT) converts any quantum channel into a Pauli channel.
- Suppresses the impact of coherent noise.
- Noisy results degrade in more predictable ways (useful for ZNE).



Randomized compiling (twirling)

- Used to convert arbitrary noise channels into other forms of noise
- Executing statistical ensembles of unitarily equivalent circuits
- Pauli Twirling (PT) converts any quantum channel into a Pauli channel.
- Suppresses the impact of coherent noise.
- Noisy results degrade in more predictable ways (useful for ZNE).



Pauli twirling (PT)

```python
from qiskit_ibm_runtime import SamplerOptions, EstimatorOptions

options = SamplerOptions(default_shots=1024)  # or...
options = EstimatorOptions(default_shots=1024, optimization_level=0, resilience_level=0)

## Configure Twirling
options.twirling.enable_gates = True
options.twirling.enable_measure = False
options.twirling.num_randomizations = 'auto'
options.twirling.shots_per_randomization = 'auto'
options.twirling.strategy = 'active-accum'
```

✓ 0.0s Python

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

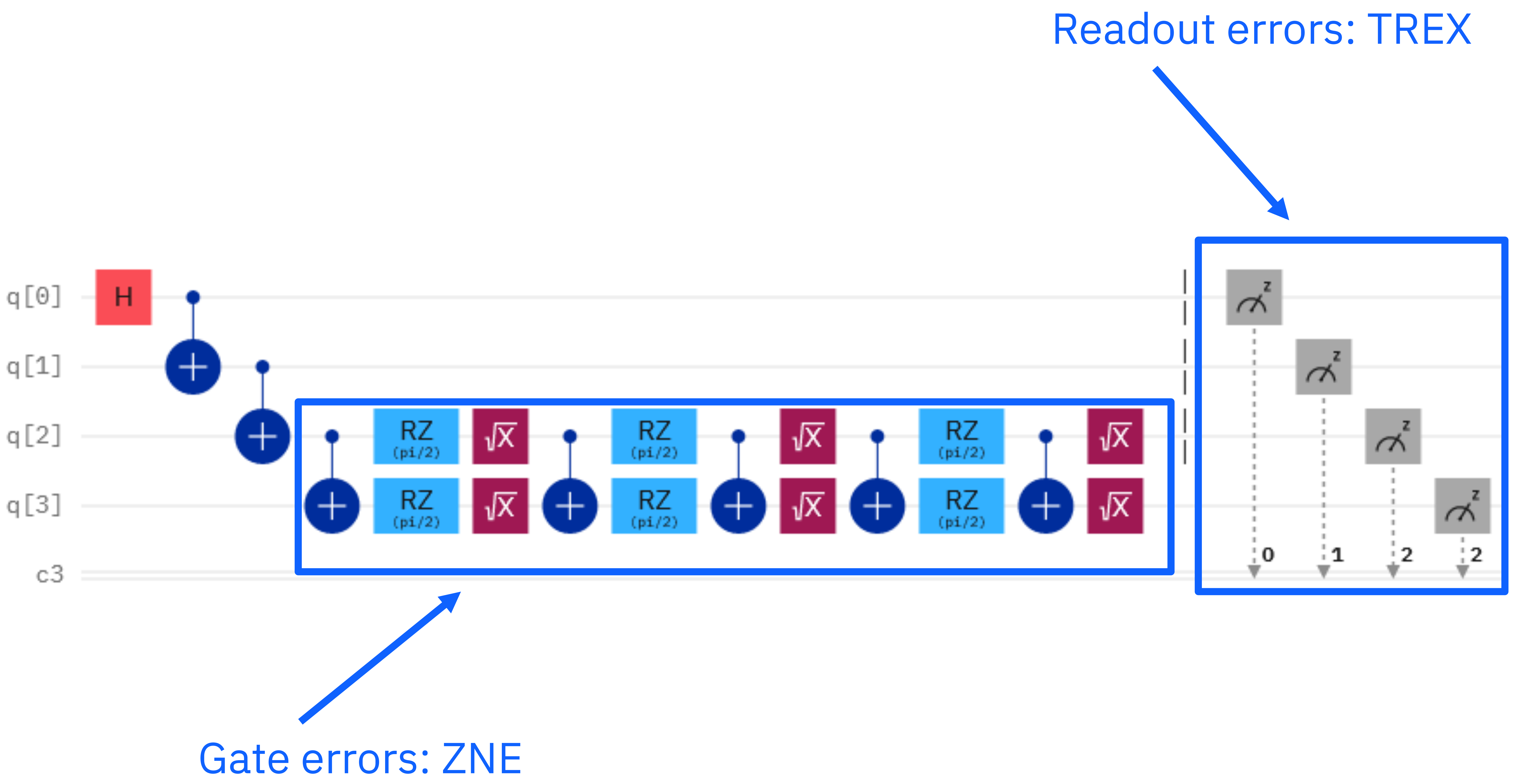
Error mitigation

Twirled readout error extinction (TREX)

Diagonalizes the readout error transfer matrix, calibrates it, and applies its inverse in post-processing

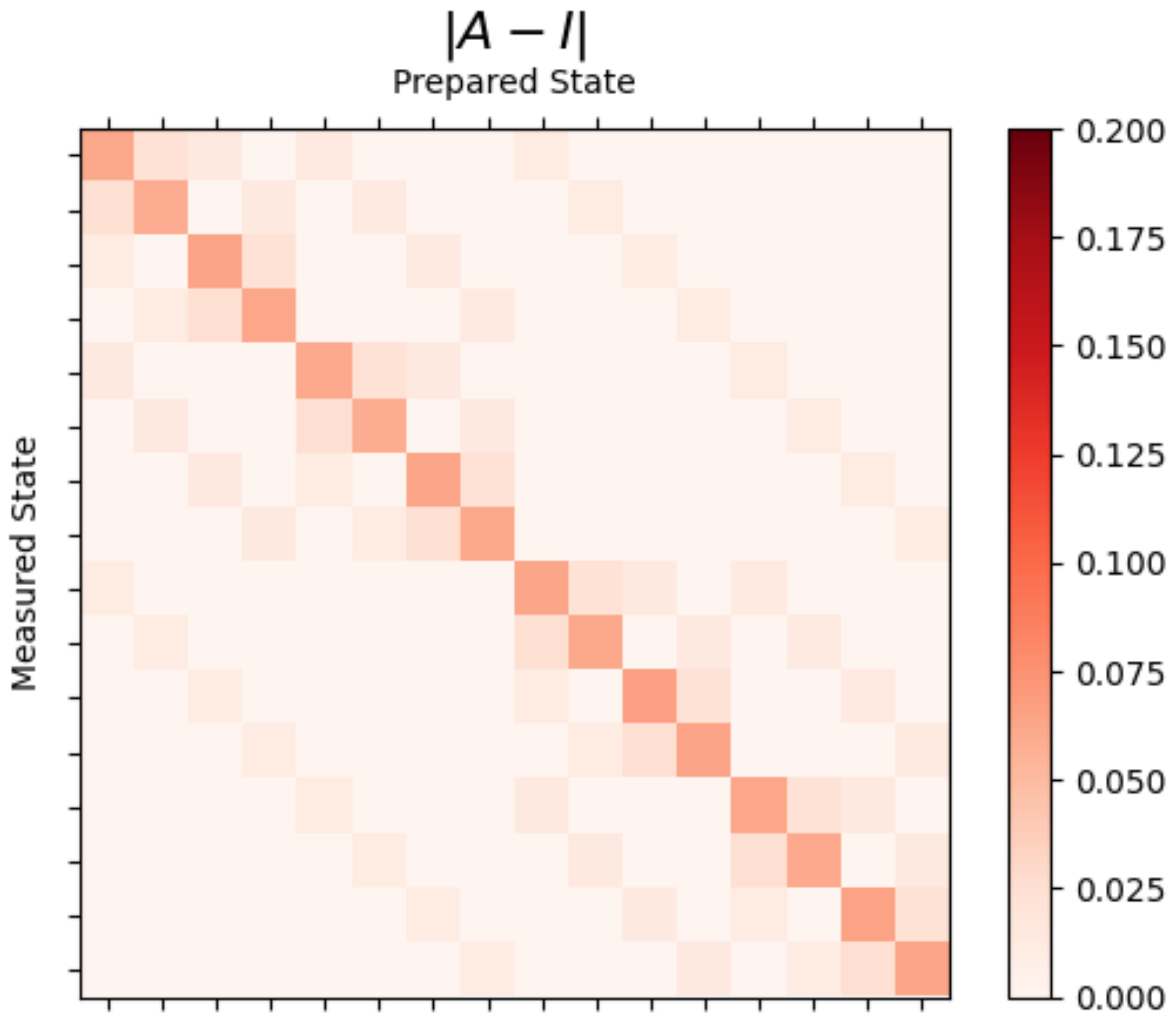
Zero noise extrapolation (ZNE)

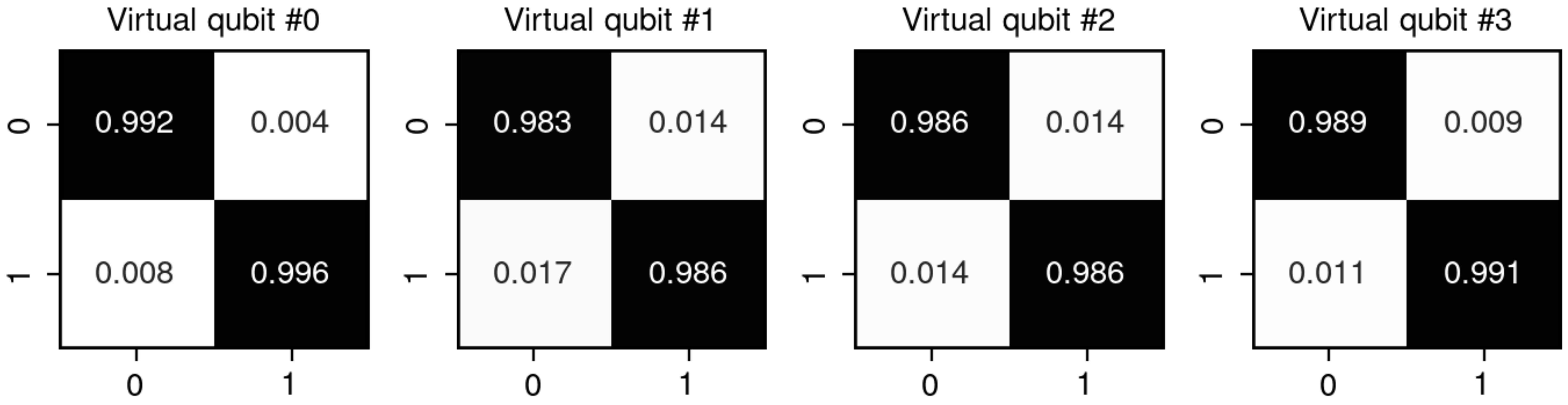
Measures the effects of increased noise to infer what the results would look like in the absence of noise



Readout errors

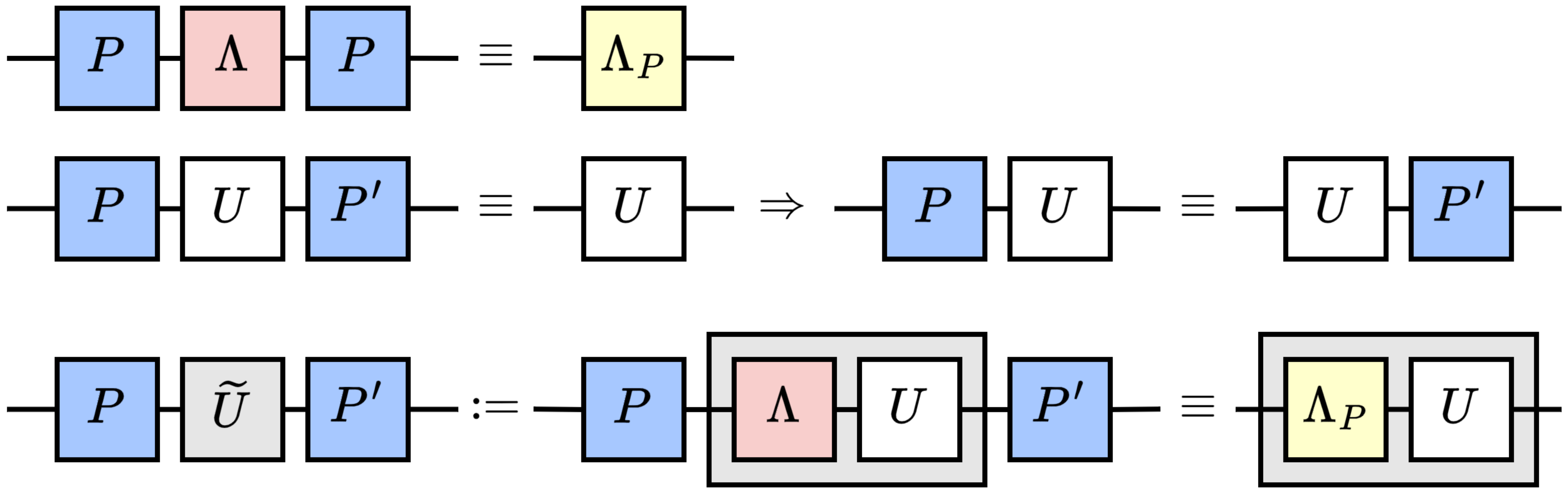
- Readouts errors cause the wrong states to be measured
- This can be modeled as a classical noise channel
- Readout error can be measured per qubit and the full error matrix reconstructed as a tensor product
- The inverse matrix can be used for error mitigation when efficiently calculable





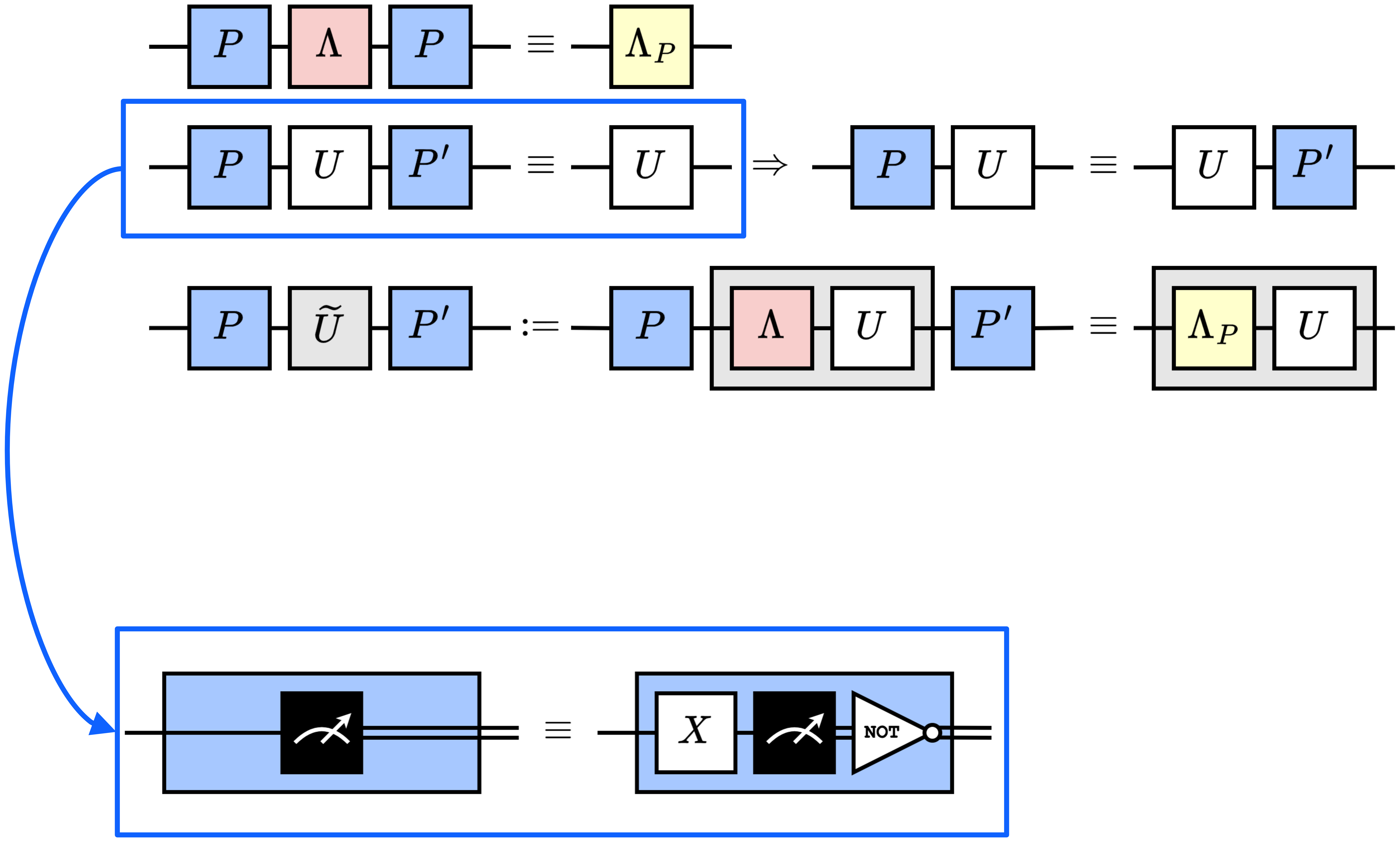
Twirled readout error extinction (TREX)

- Diagonalizes the readout-error transfer matrix via measurement twirling
- Such diagonal matrix is learned by running identity calibration circuits
- Finally, one can trivially invert the diagonal matrix and apply it to the target results
- Only valid for expectation-value problems



Twirled readout error extinction (TREX)

- Diagonalizes the readout-error transfer matrix via measurement twirling
- Such diagonal matrix is learned by running identity calibration circuits
- Finally, one can trivially invert the diagonal matrix and apply it to the target results
- Only valid for expectation-value problems



Twirled readout error extinction (TREX)

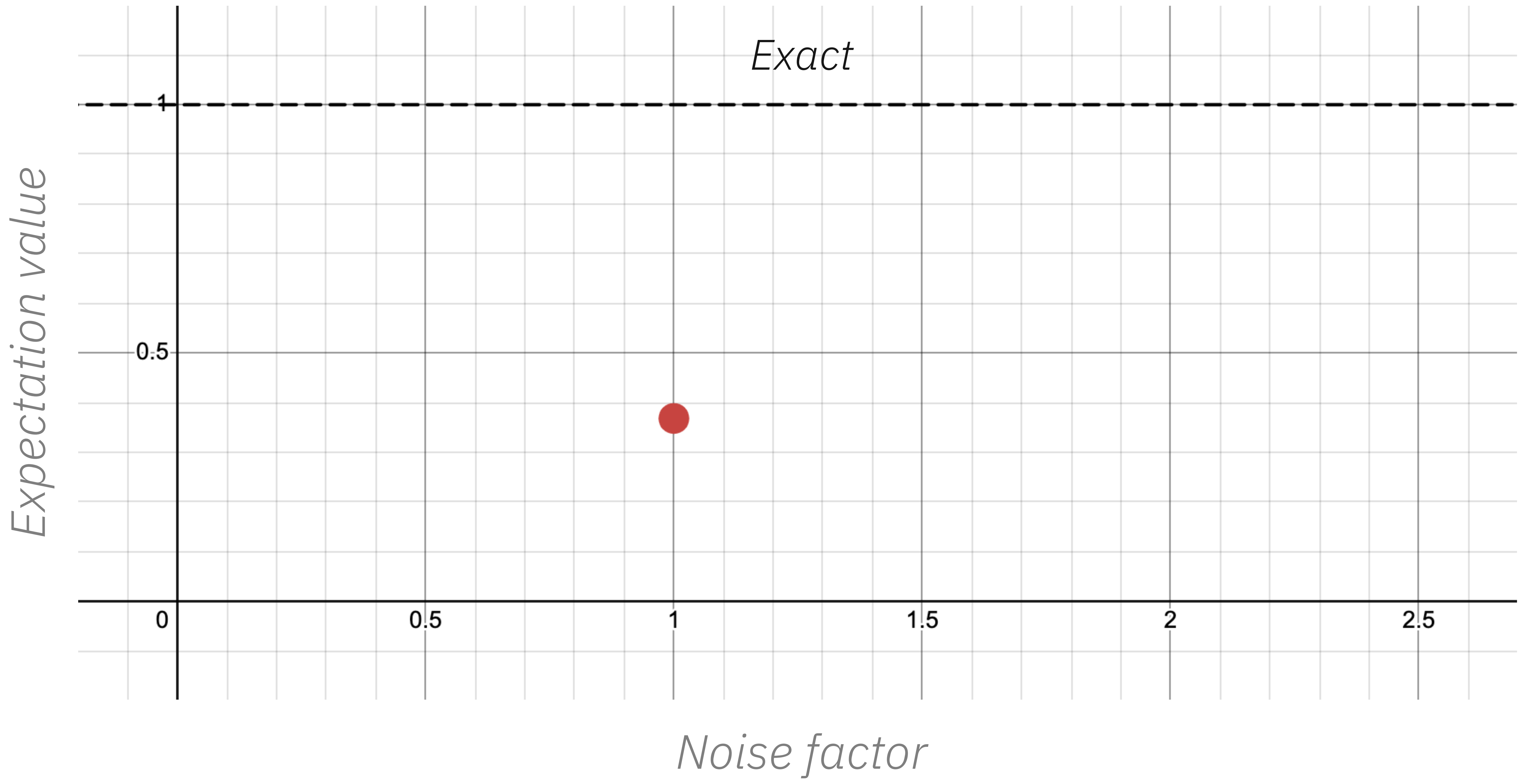
```python
from qiskit_ibm_runtime import EstimatorOptions

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
```

✓ 0.0s Python

Resilience options (V2): https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.ResilienceOptionsV2

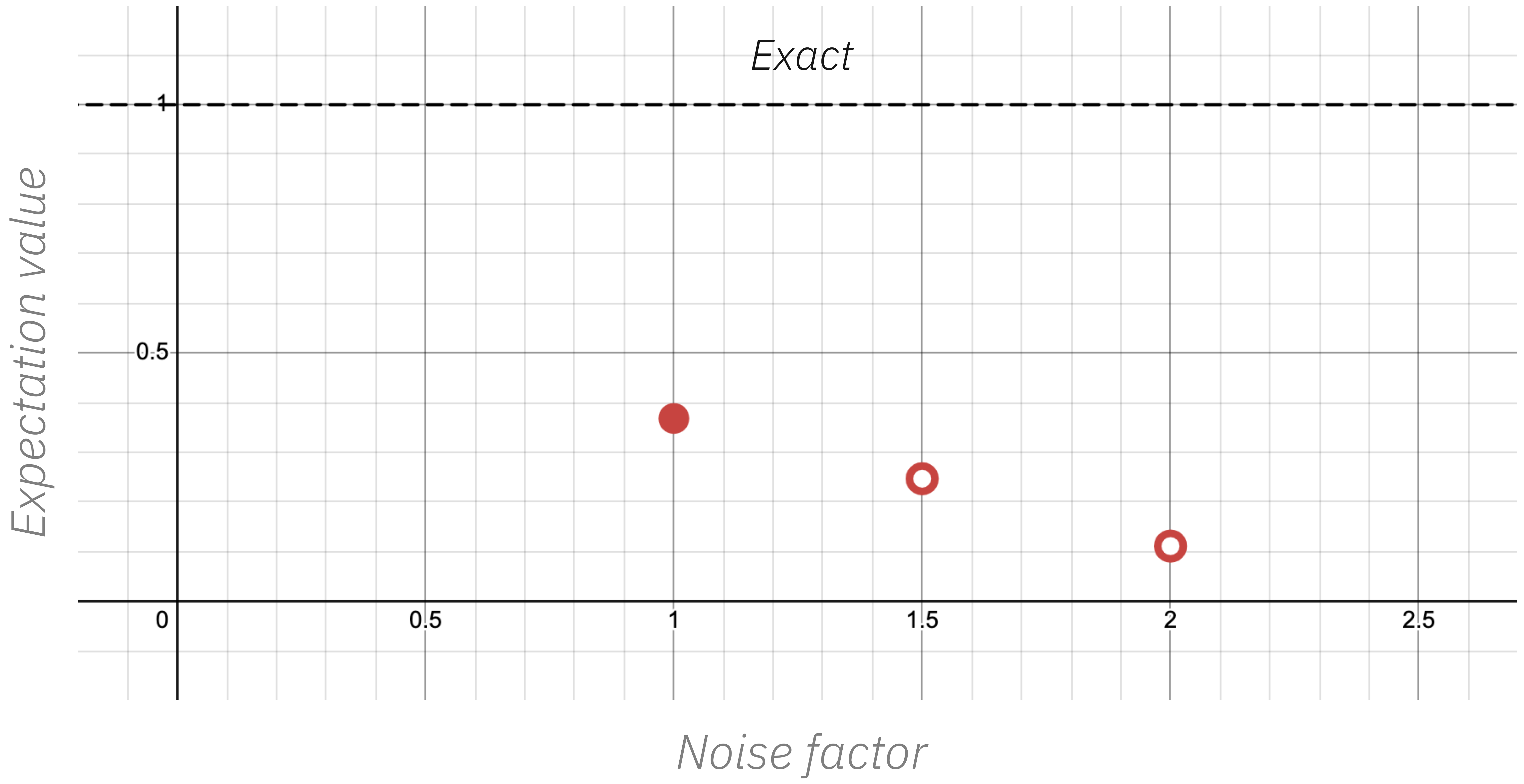
Zero noise extrapolation (ZNE)

- Divided in two phases
  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
- Needs careful attention but exhibits great potential
- Only valid for expectation-value problems



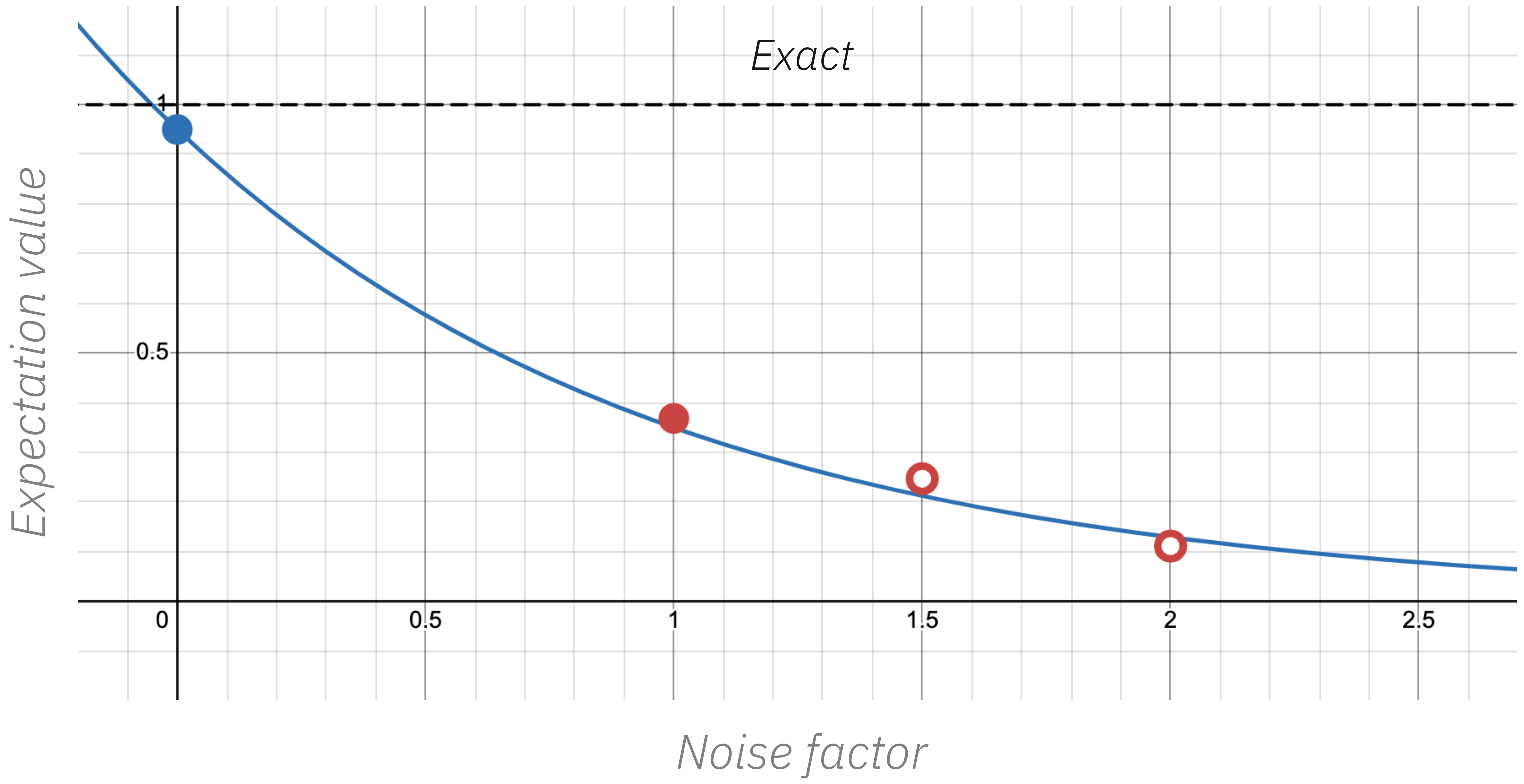
Zero noise extrapolation (ZNE)

- Divided in two phases
  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
- Needs careful attention but exhibits great potential
- Only valid for expectation-value problems



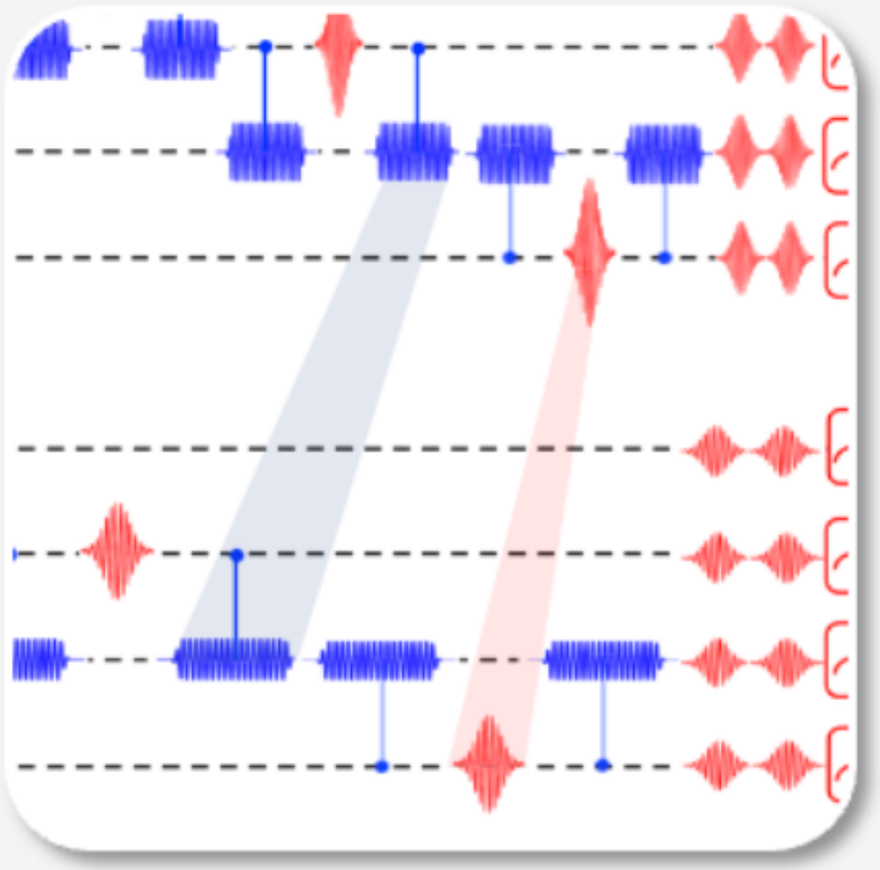
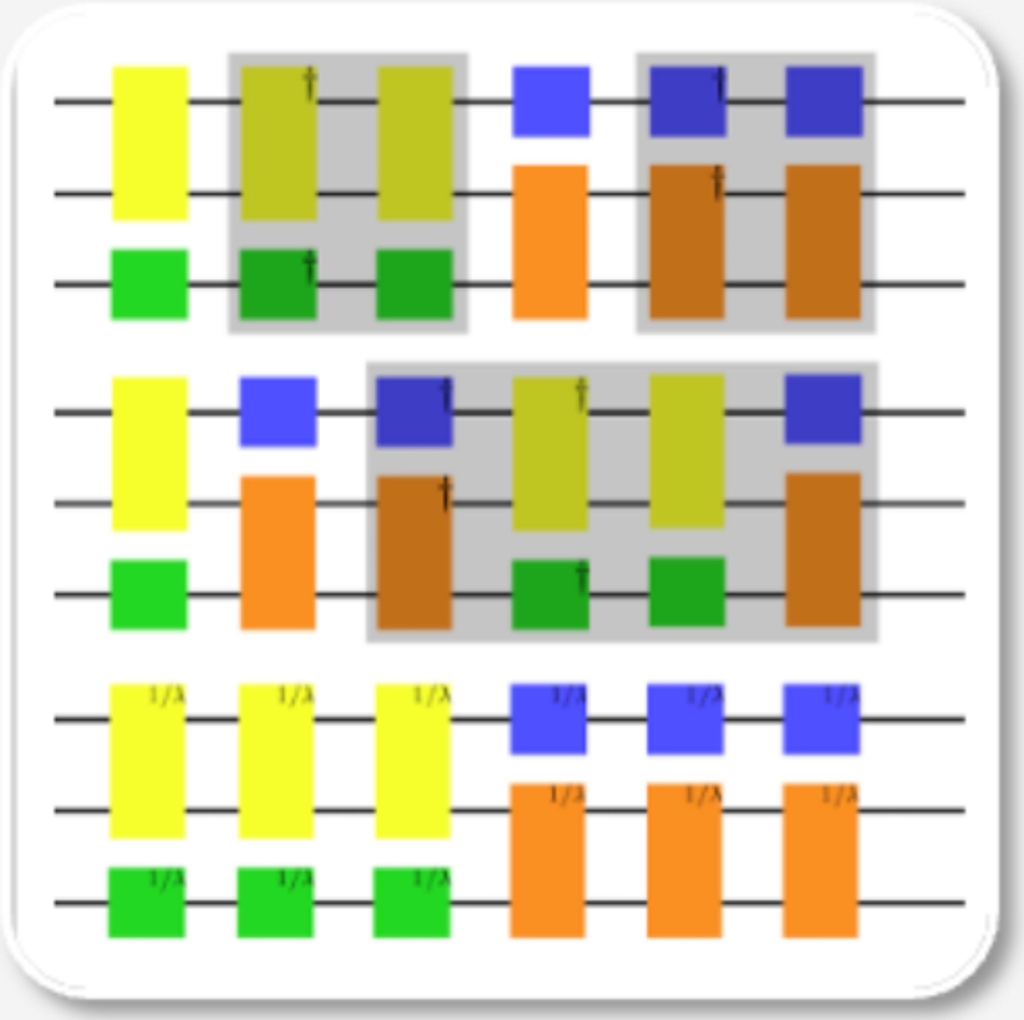
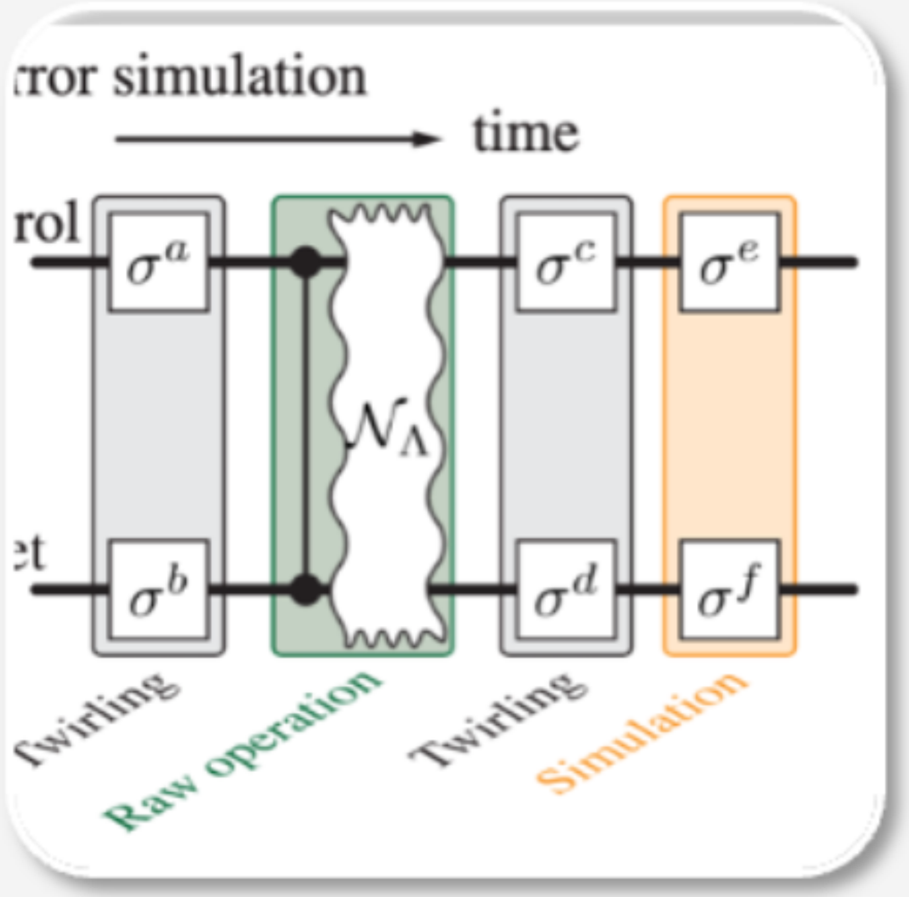
Zero noise extrapolation (ZNE)

- Divided in two phases
  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
- Exhibits great potential but needs careful attention
- Only valid for expectation-value problems



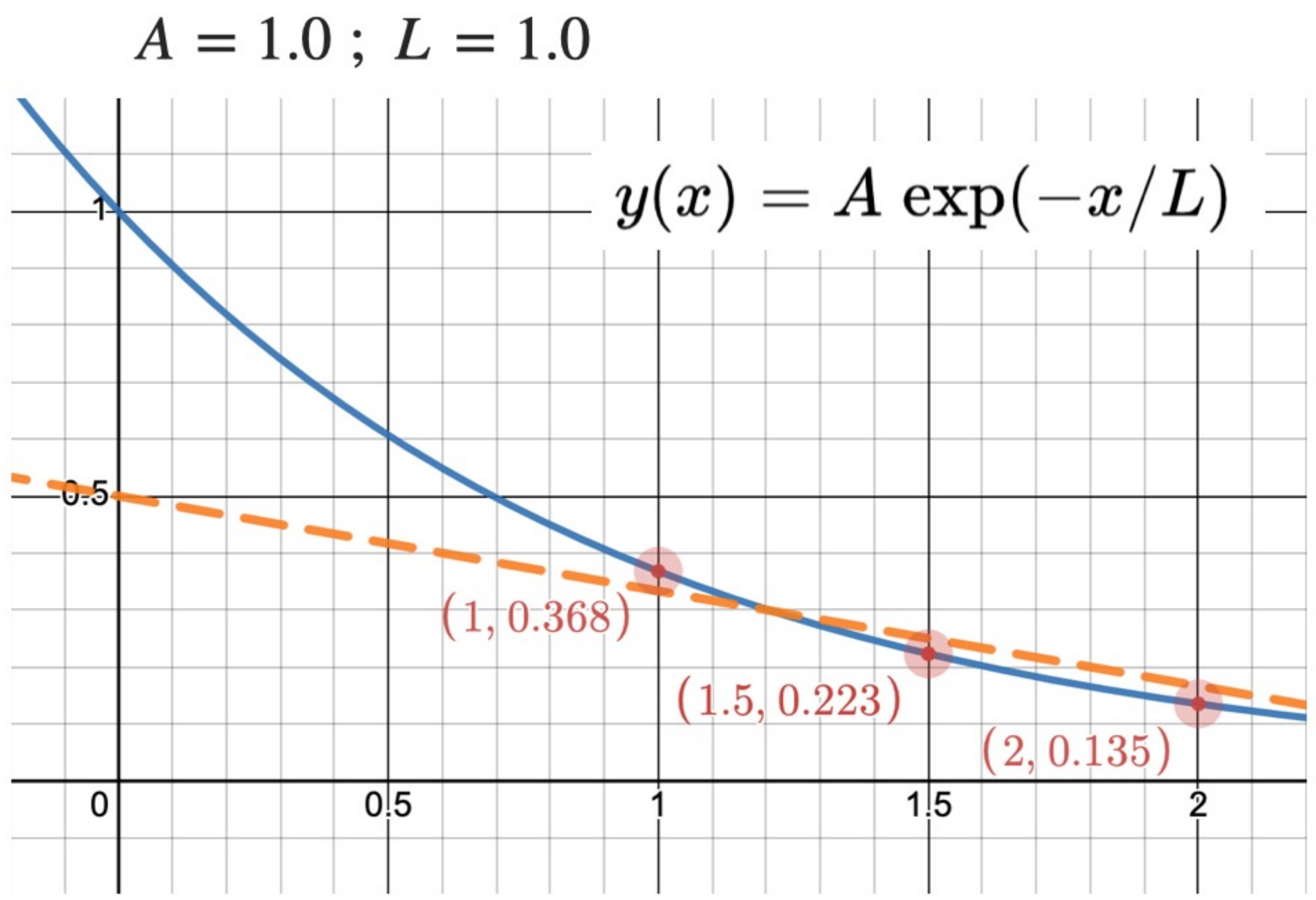
Noise amplification (ZNE)

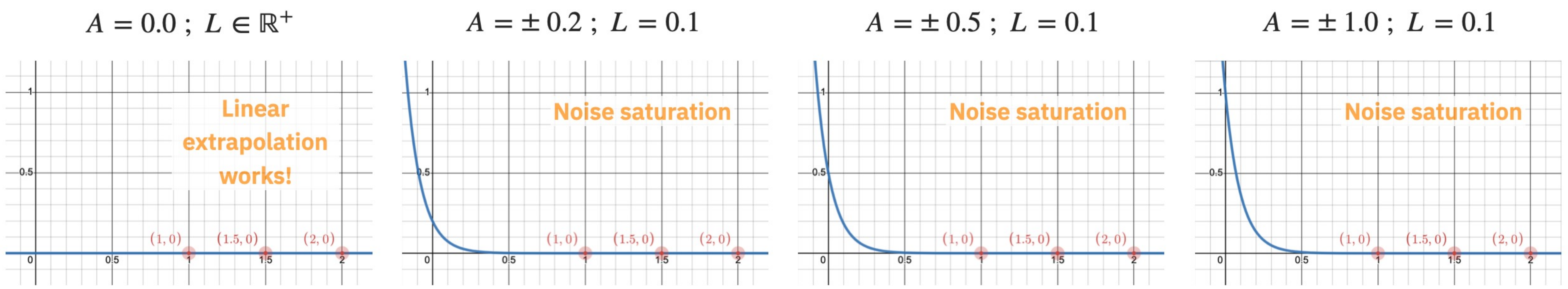
- Pulse stretching commonly requires costly pulse level calibration of the hardware
- Gate folding is largely a heuristic approach but offers a good trade-off between result quality and resource requirements
- Probabilistic error amplification (PEA) requires learning circuit-specific noise but has general applicability and strong theoretical backing

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)	Shultz et al. PRA (2022)	Li & Benjamin PRX (2017)

Extrapolation (ZNE)

- Theoretical/experimental results predict exponential decay in observed expectation values
- Exponential extrapolation mitigates aggressively but is unstable, since the *scale* is unknown
- Polynomial extrapolation is stable but mitigates worse, since it retains the *scale* of the noisy data
- Needs careful attention but exhibits great potential





Zero noise extrapolation (ZNE)

```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)
options.resilience.zne.extrapolator = ('exponential', 'linear')
```

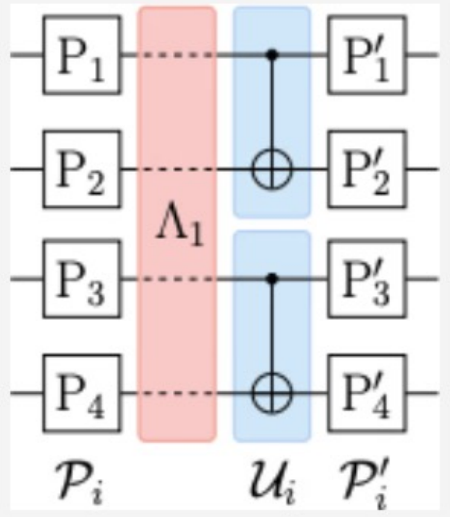
✓ 0.0s Python

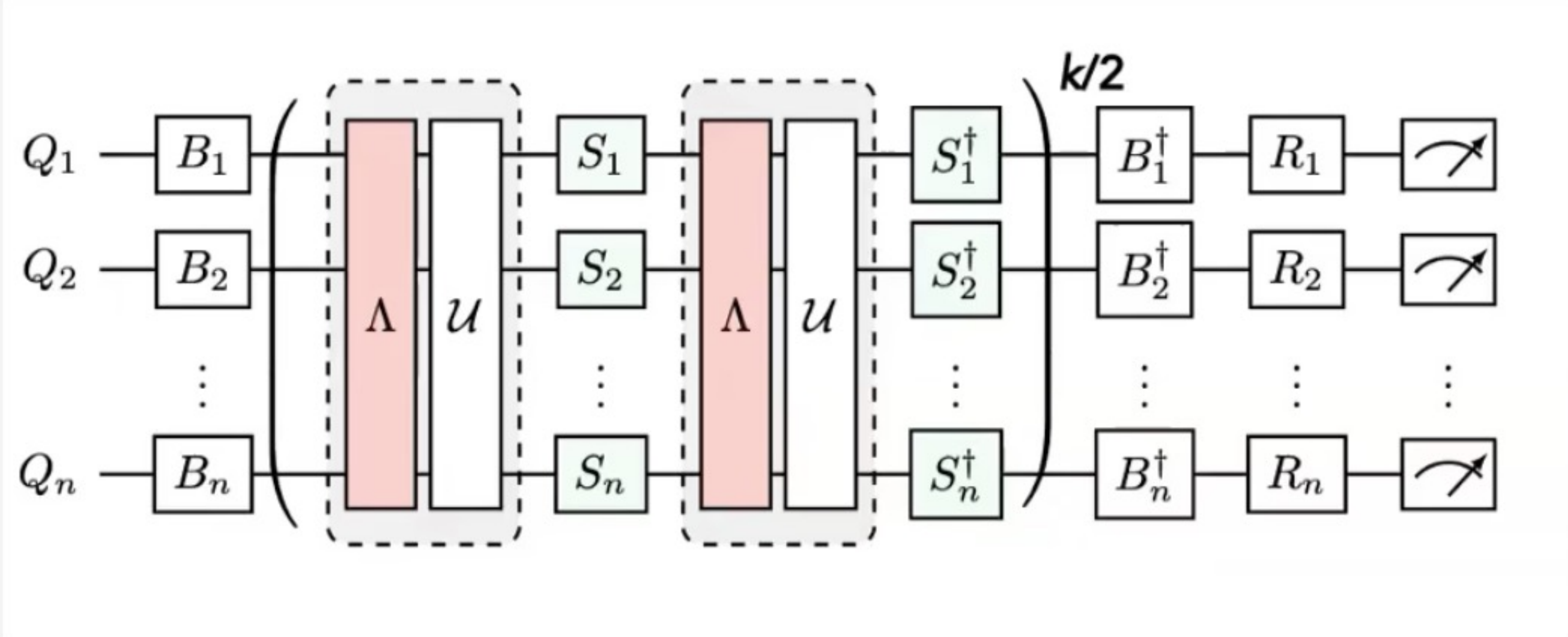
Resilience options (V2): https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.ResilienceOptionsV2

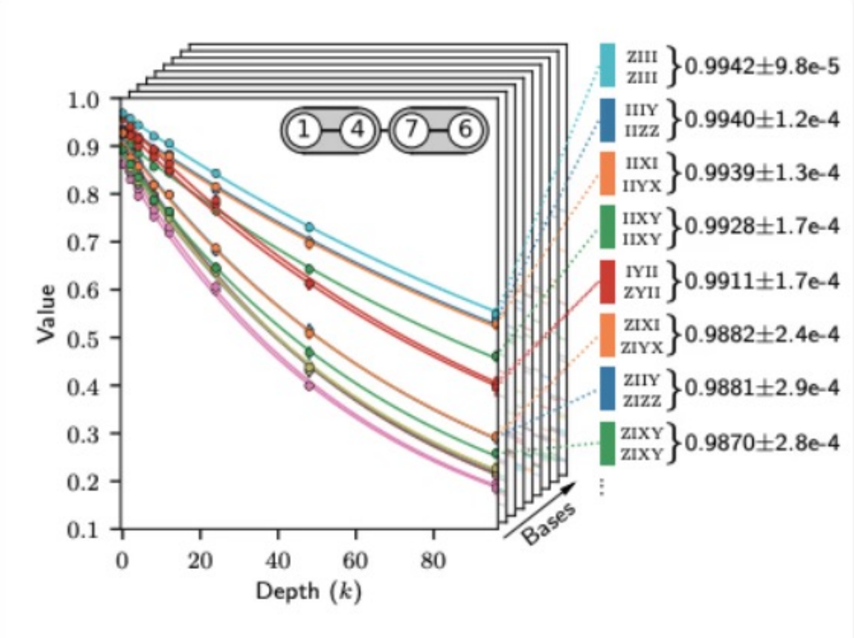
Probabilistic error amplification (PEA)

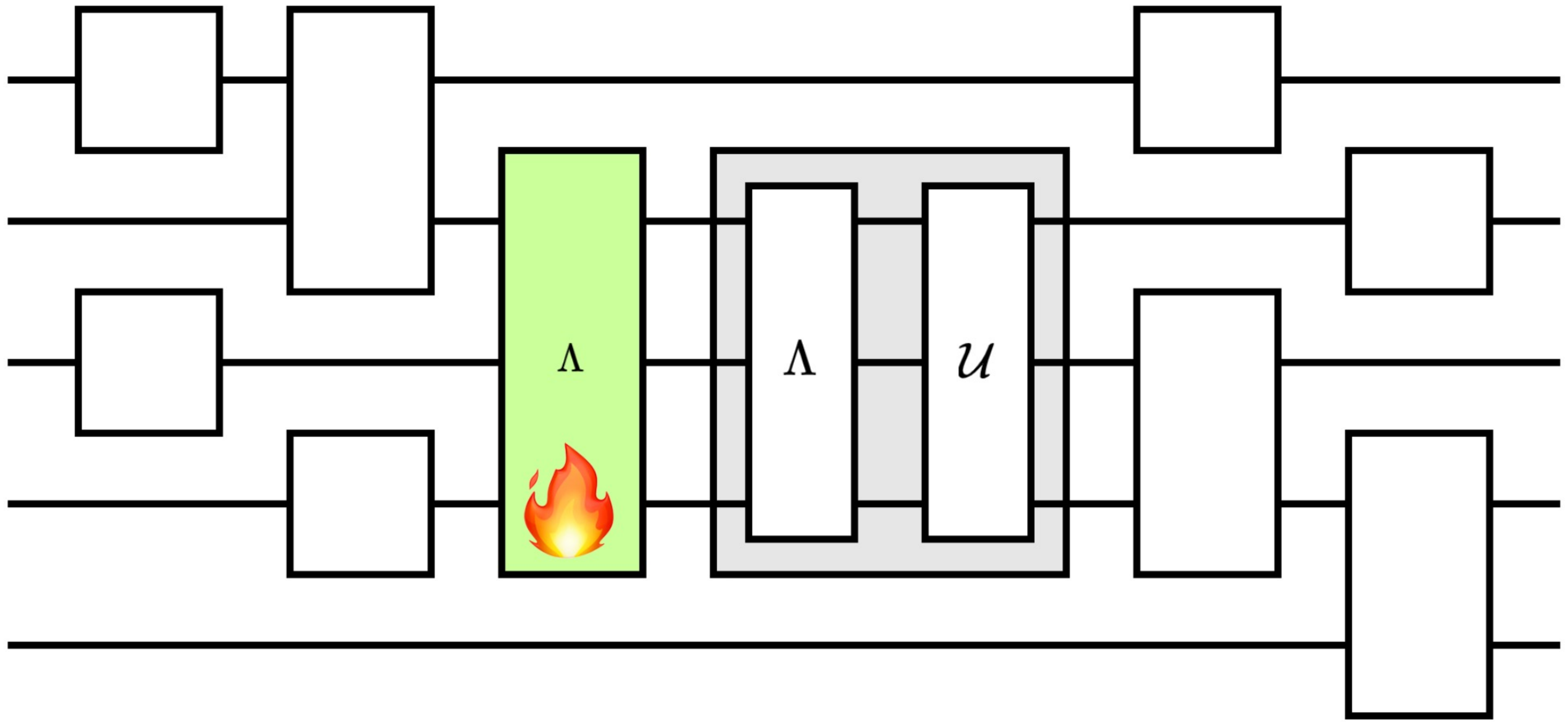
- Noise amplification technique for ZNE
- Executing statistical ensembles of circuits
- Two tasks per layer:
  1. Noise learning
  2. Noise injection
- General applicability and strong theoretical backing

Step 1	Step 2	Step 3
Pauli twirl layers of 2-qubit gates	Repeat identity pairs of layers and learn the noise	Derive a fidelity (error for each noise channel)









Probabilistic error amplification (PEA)

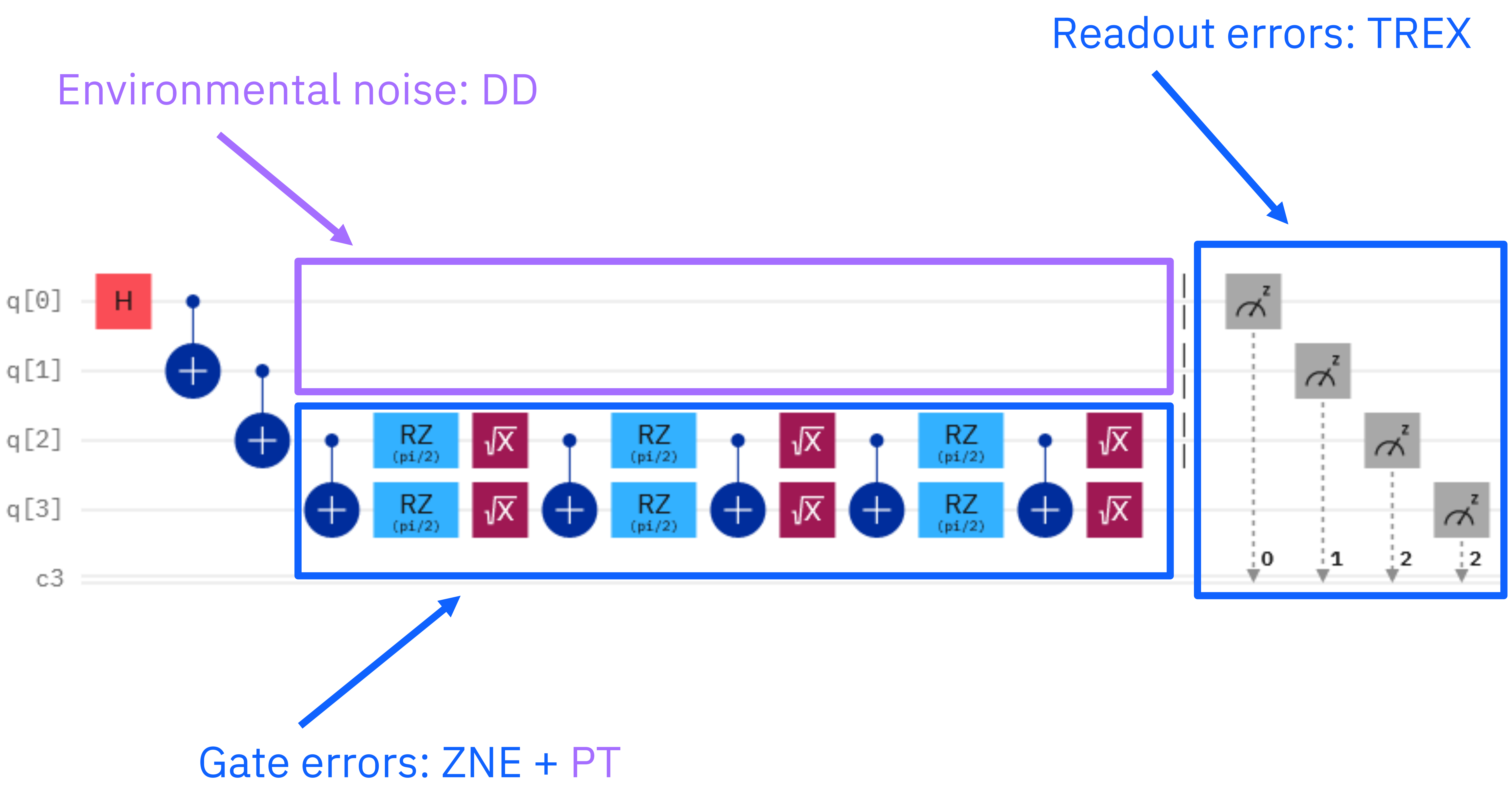
```python
from qiskit_ibm_runtime import EstimatorOptions

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

## Configure ZNE with PEA
options.resilience.zne_mitigation = True
options.resilience.zne.noise_factors = (1, 3, 5)
options.resilience.zne.extrapolator = 'exponential'

options.experimental = {'resilience': {'zne': {'amplifier': 'pea'}}}

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)
```

✓ 0.0s Python

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

Fighting noise before error correction

- Different types of noise need different suppression and mitigation techniques
- Different techniques can be combined



Combining techniques

```python
from qiskit_ibm_runtime import QiskitRuntimeService, EstimatorV2, EstimatorOptions

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

## Configure Dynamical Decoupling
options.dynamical_decoupling.enable = True
options.dynamical_decoupling.sequence_type = 'XX'
options.dynamical_decoupling.extra_slack_distribution = 'middle'
options.dynamical_decoupling.scheduling_method = 'alap'

## Configure Twirling
options.twirling.enable_gates = True
options.twirling.enable_measure = True  # Needed for TREX
options.twirling.num_randomizations = 'auto'
options.twirling.shots_per_randomization = 'auto'
options.twirling.strategy = 'active-accum'

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

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

service = QiskitRuntimeService()
backend = service.least_busy()
estimator = EstimatorV2(backend, options=options)
```

✓ 25.4s Python

Resilience levels

Resilience Level	Definition	Technique
0	No mitigation	None
1 [Default]	Minimal mitigation costs: Mitigate error associated with readout errors	Twirled Readout Error eXtinction (TREX) measurement twirling
2	Medium mitigation costs. Typically reduces bias in estimators, but is not guaranteed to be zero-bias.	Level 1 + Zero Noise Extrapolation (ZNE) and gate twirling

IBM