Quantum Computing
A Practical Perspective

Marco Venere

marco.venere@polimi.it

December 3rd, 2024
Politecnico di Milano

Agenda

- Lecture 1 → Theory Recap on Quantum Computing
- Lecture 2 → Initial Setup and First Experiments
- Lecture 3 → Grover's Algorithm
- Lecture 4 → Combinatorial Optimization
- Lecture 5 → VQE, QNN, QMC
- Lecture 6 → Quantum Error Correction & Mitigation – Projects Presentation

Error Correction

Computers make errors. It happens all the time.

Thermal noise and electromagnetic interference may induce bit flips in the memory of the devices, causing wrong outputs from the running algorithm.

To avoid such errors, it is necessary to create redundancy of information: in this way, bit flips can be detected and corrected, before the mistakes affect computation.

Such a principle applies to quantum computers as well.

Quantum Error Correction

QEC aims to identify errors in the amplitude (**bit flips**) or in the phase (**phase flips**) of the qubits.

Classical computing involves duplicating bytes to create redundancy. This cannot be performed on quantum computers because of the No-Cloning Theorem.

Furthermore, in classical computing you may have to read bytes in order to evaluate bit parity and detect errors. This cannot happen on QCs as well, because of the destroying effect of any measurement.

Quantum Errors

We can formalize the errors that occur in a quantum computer.

Given a qubit $|\psi\rangle$, a generic operation U may provoke errors.

We can describe $U|\psi\rangle$ as $U|\psi\rangle = \alpha_I I|\psi\rangle + \alpha_X X|\psi\rangle + \alpha_Z Z|\psi\rangle + \alpha_y Y|\psi\rangle$

Considering that Y=XZ: $U|\psi\rangle = \alpha_I I|\psi\rangle + \alpha_X X|\psi\rangle + \alpha_Z Z|\psi\rangle + \alpha_y XZ|\psi\rangle$

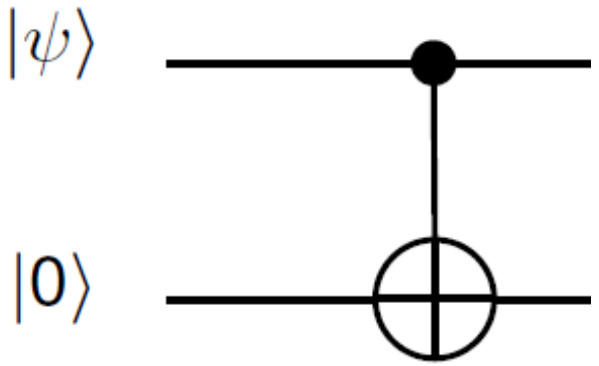
Therefore, a generic **error correction code** to solve errors should be able to correct errors happening with X and Z Pauli operations.

X operations produce bit flips, while Y operations produce phase flips.

Ways To Create Redundancy

Even though we cannot duplicate a generic quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, we can duplicate basis states!

$$|\psi\rangle|0\rangle = (\alpha|0\rangle + \beta|1\rangle)|0\rangle$$
$$= \alpha|0\rangle|0\rangle + \beta|1\rangle|0\rangle$$
$$\rightarrow \alpha|0\rangle|0\rangle + \beta|1\rangle|1\rangle$$



Even though we have not cloned the state, we have entangled it to another qubit. If we measure qubits and get a different output, we can understand that an error occurred.

This and the following slides for QEC are based on material from Quantum Information Processing 101

Stabilizers

We have encoded a **logical qubit**:

$$|\psi\rangle_L \in \mathcal{C} = \text{ span}\{|00\rangle, |11\rangle\} \subset \mathcal{H}_4$$

where $\mathcal{C}$ is called **codespace**.

Let's assume a **bitflip** happens on the first qubit:

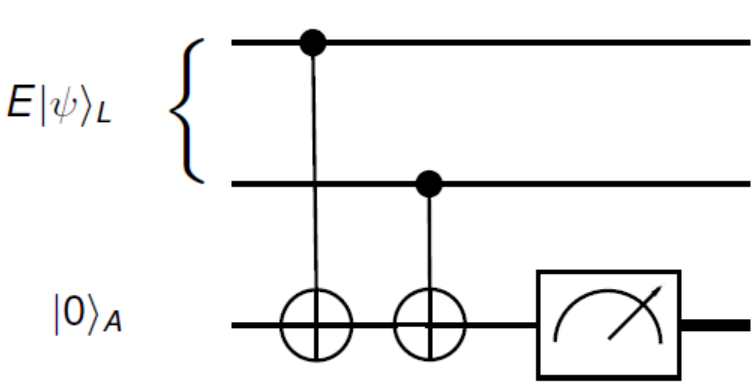
$$X_1|\psi\rangle_L = \alpha|10\rangle + \beta|01\rangle \in \mathcal{F} \perp \mathcal{C}$$

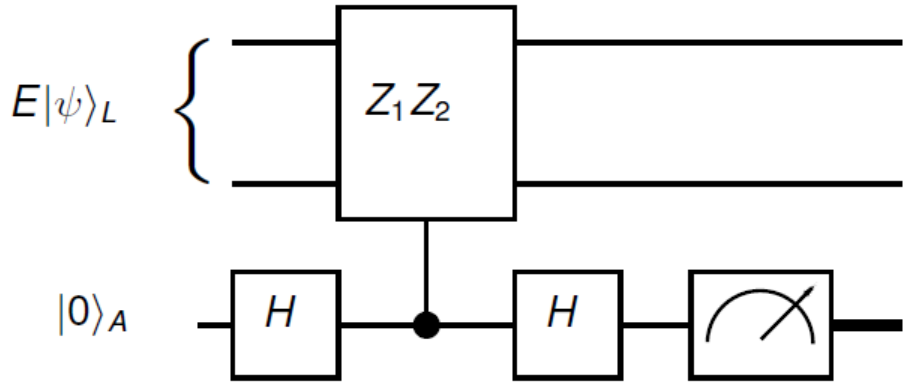
where $\mathcal{F}$ is called **error subspace**.

Since $\mathcal{F} \perp \mathcal{C}$, we can project the qubits state in such a way that we can detect the error without compromising quantum information.

This operation is called **stabilizer measurement**. We define **distance** of the code the minimum number of error necessary to move from one valid codeword to another one (thus preventing error detection).

Stabilizers



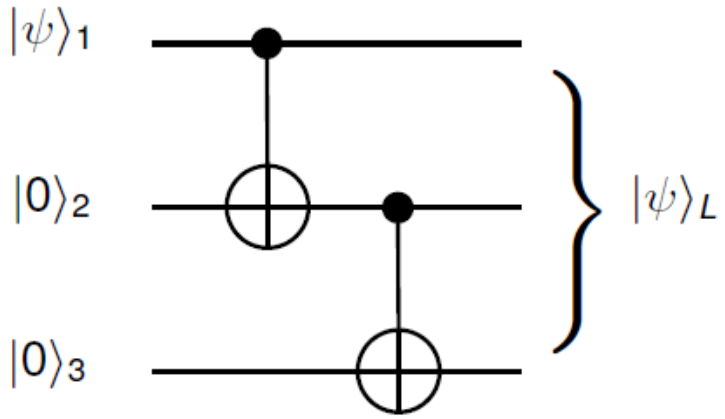


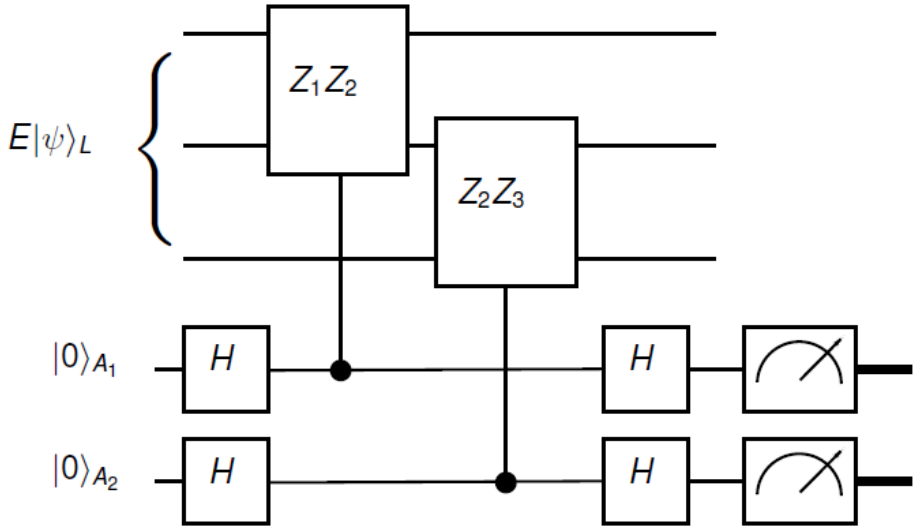
These two circuits display such a projection. The output of the measurement is called **syndrome**, and can give us a hint about what errors occurred:

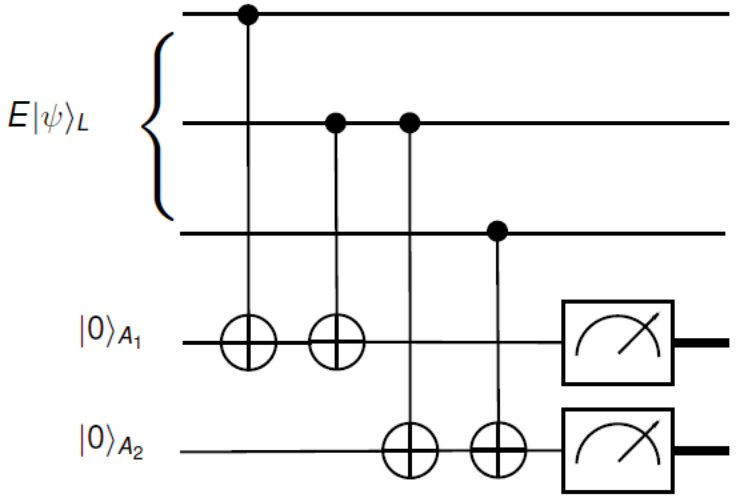
Error E	Syndrome
I_1I_2	0
X_1I_2	1
I_1X_2	1
X_1X_2	0

We can detect only 1 qubit error, and we cannot correct it.

3-Qubit Error Correction Code





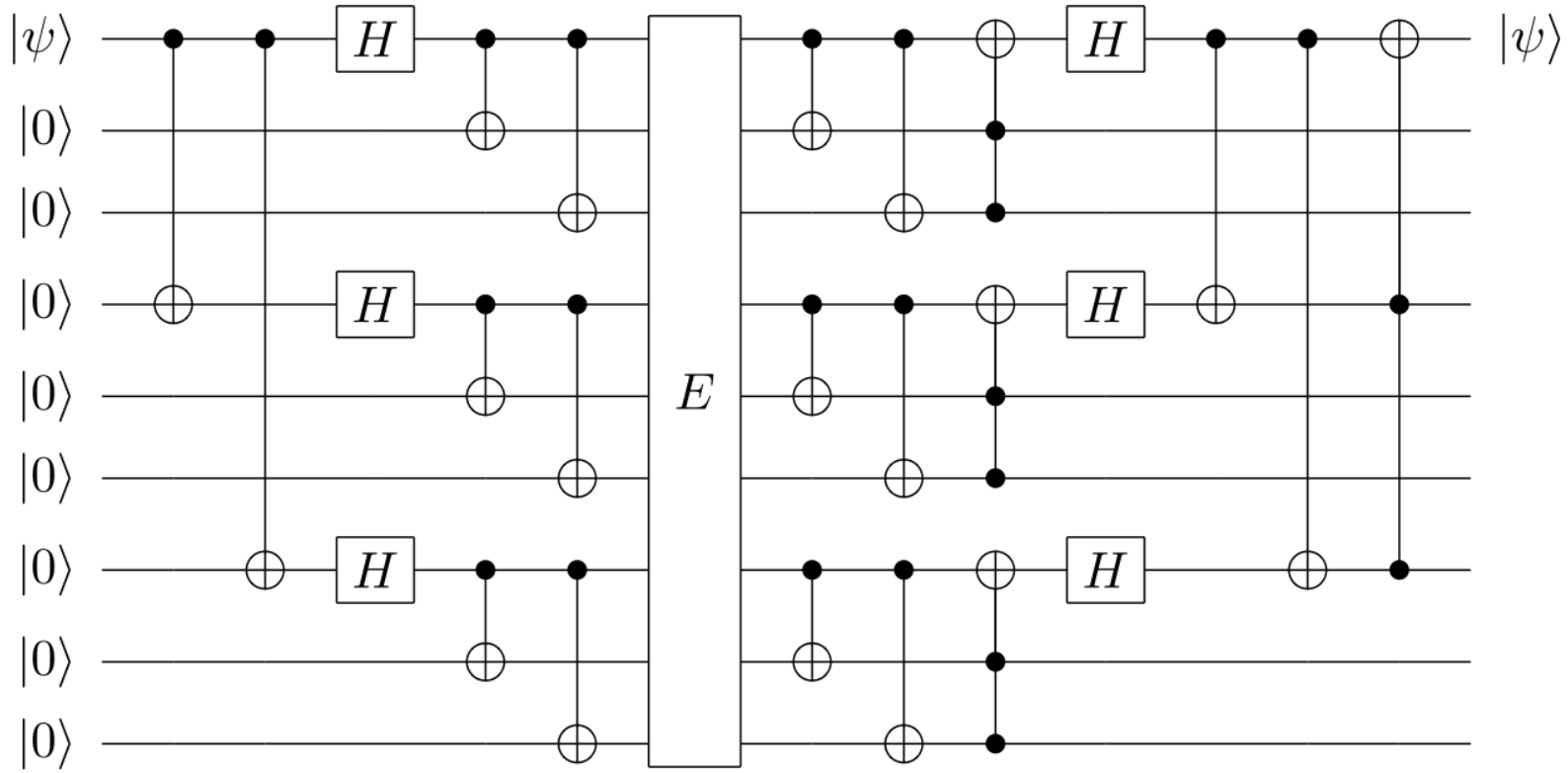


Error E	Syndrome	Error E	Syndrome
$I_1I_2I_3$	00	$X_1X_2X_3$	00
$X_1I_2I_3$	10	$I_1X_2X_3$	10
$I_1X_2I_3$	11	$X_1I_2X_3$	11
$I_1I_2X_3$	01	$X_1X_2I_3$	01

We can detect up to 2 qubit errors,
and we can correct only 1 qubit error.

9-Qubit Shor Code

Shor's Code is famous for historical reasons. It has distance = 3 and can correct all 1-qubit errors.



9-Qubit Shor Code

Shor's Code is famous for historical reasons. It has distance = 3 and can correct all 1-qubit errors.

Error E	Syndrome	Recovery	Error E	Syndrome	Recovery
X_1	10000000	X_1	Z_1	00000010	Z_1
X_2	11000000	X_2	Z_2	00000010	Z_2
X_3	01000000	X_3	Z_3	00000010	Z_3
X_4	00100000	X_4	Z_4	00000011	Z_4
X_5	00110000	X_5	Z_5	00000011	Z_5
X_6	00010000	X_6	Z_6	00000011	Z_6
X_7	00001000	X_7	Z_7	00000001	Z_7
X_8	00001100	X_8	Z_8	00000001	Z_8
X_9	00000100	X_9	Z_9	00000001	Z_9

Let’s Experiment With MATLAB

We can now use MATLAB to generate a circuit for 9-Qubit Shor Code.

Correcting Codes

Assume that we use a code which is able to identify the error.

We need to apply a specific correction to the error, in order to bring the state back to the ideal one.

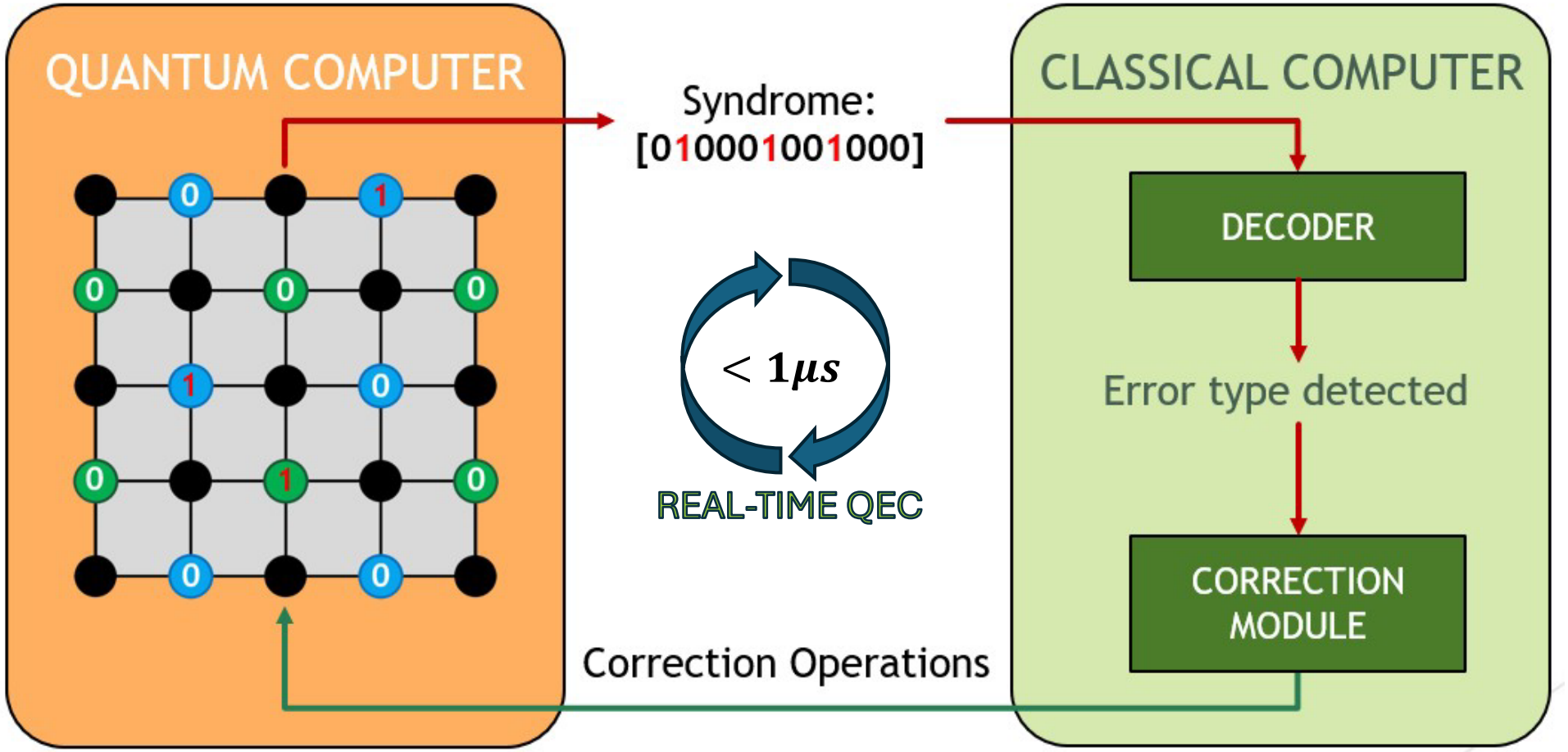
This implies that during computation, we need to continuously measure syndromes and apply their corresponding operations. In theory, this should be performed at a rate of $1\mu s$ per correction.

Hardware Acceleration

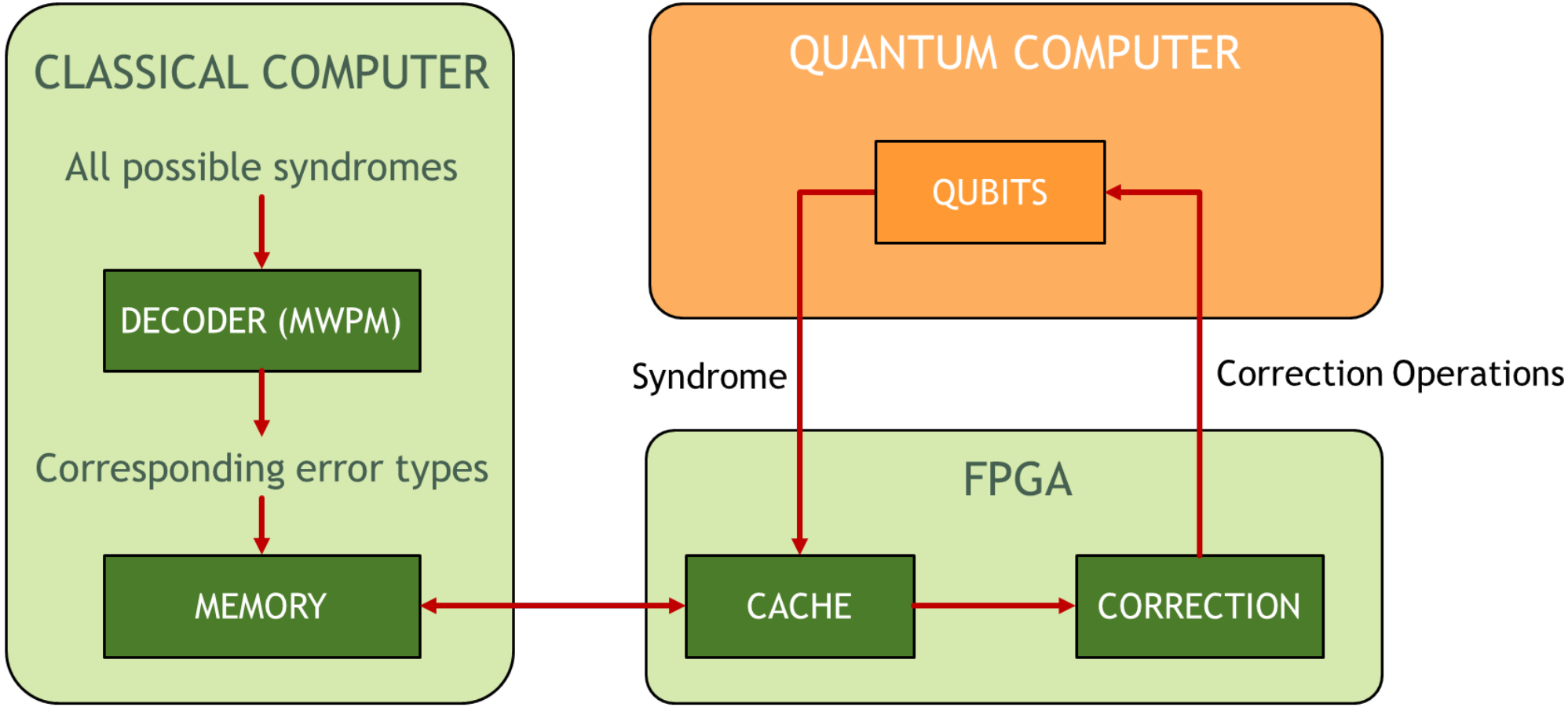
This kind of operation cannot be performed by the quantum computer. Indeed, we need an external device that retrieves the **syndrome**, **decodes** it to understand what errors have occurred, and required the QC to apply the corresponding correction.

The $1\mu s$ rate is a very high threshold, and we cannot simply exploit a generic CPU to perform this operation. We need specialized hardware that implements the error decoder. An example of such hardware is the Field Programmable Gate Array (FPGA).

General Workflow

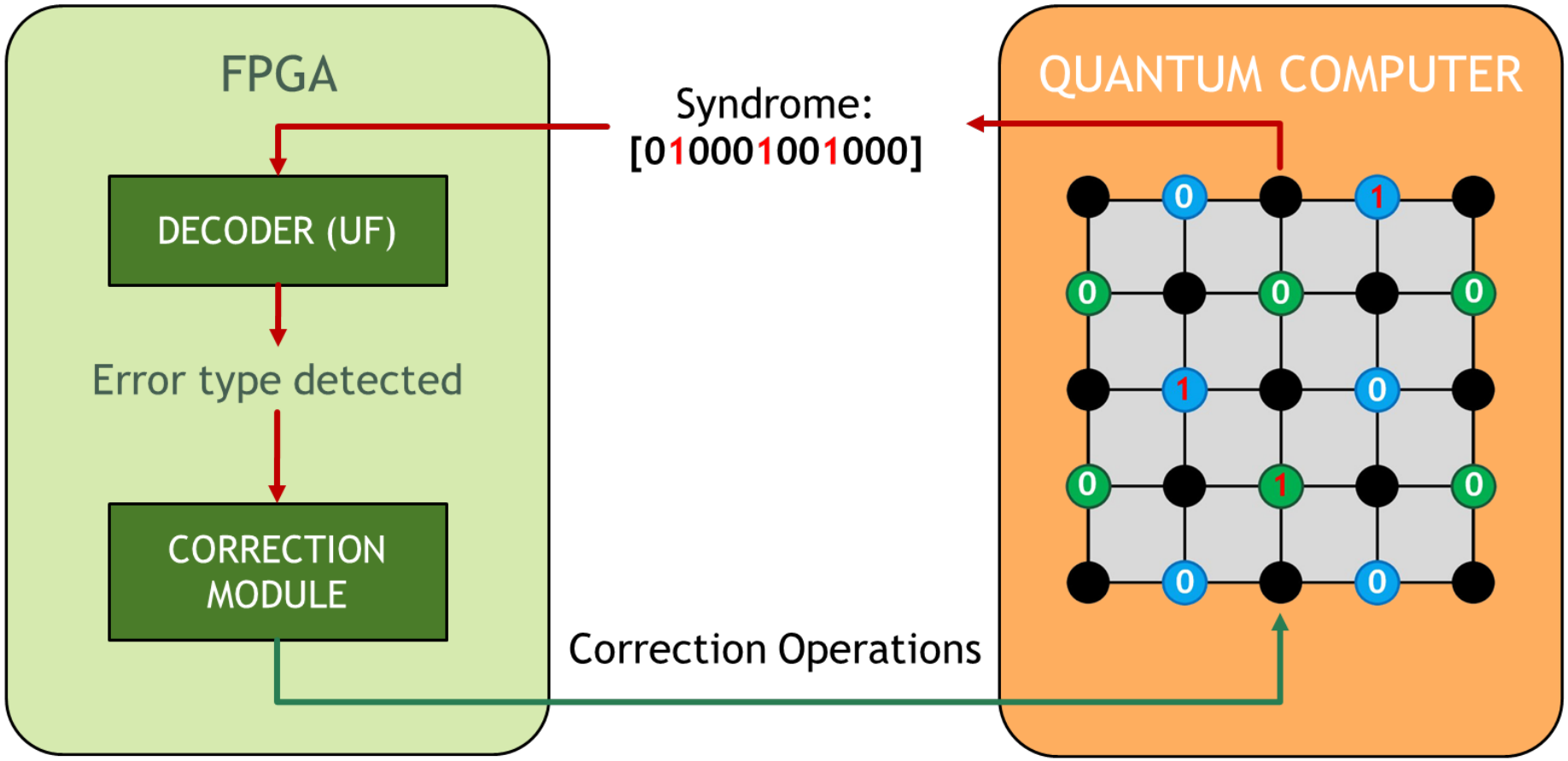


QEC: State of the Art – Lilliput (Perfect Matching)



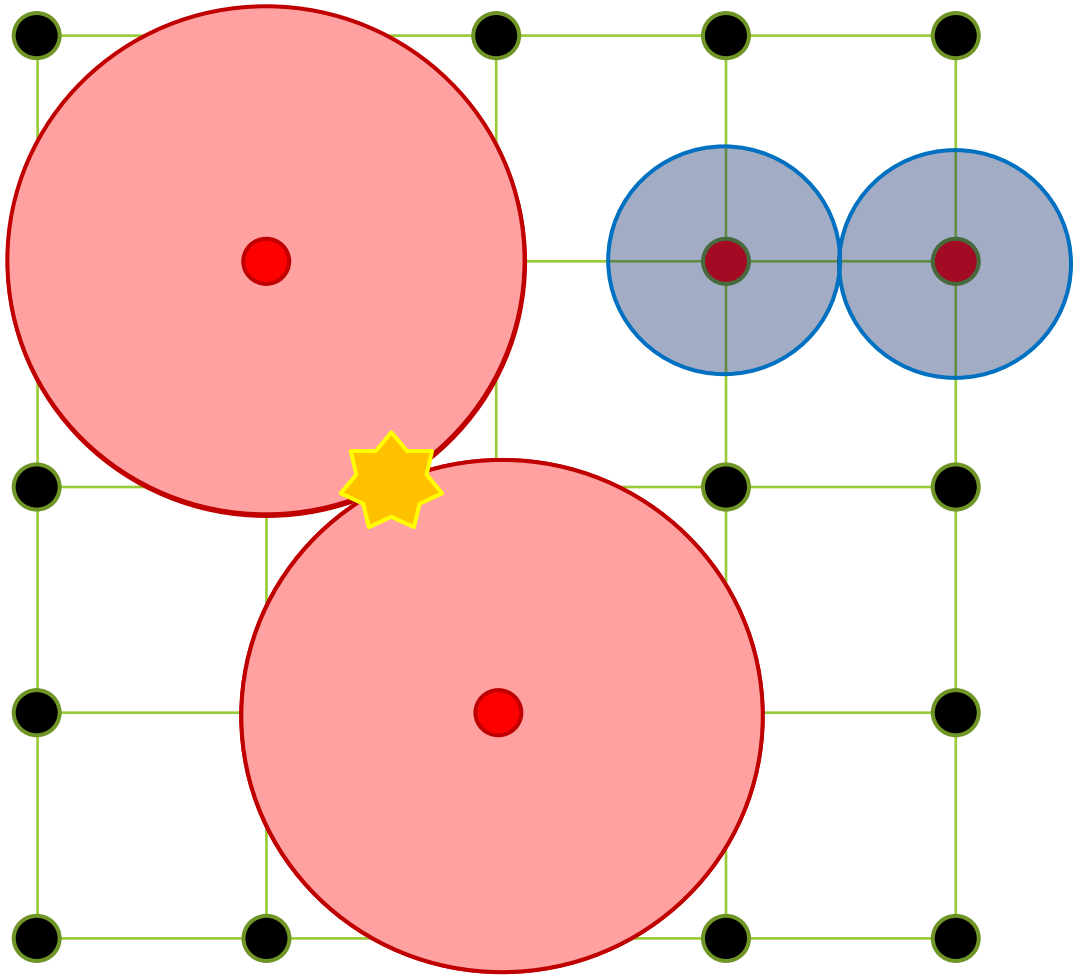
Das, Poulami, Aditya Locharla, and Cody Jones. "Lilliput: A lightweight low-latency lookup-table based decoder for near-term quantum error correction." *arXiv preprint arXiv:2108.06569* (2021).

QEC: State of the Art – Helios (Union Find)



Liyanage, Namitha, et al. "Scalable Quantum Error Correction for Surface Codes using FPGA." *arXiv preprint arXiv:2301.08419* (2023).

Sparse Blossom Algorithm



- Sparse Blossom algorithm for fast **Minimum-Weight Perfect Matching.**
- Algorithm based on **regions** that **expand** and **collide.**
- A **match** between two triggered detectors is an **edge** that represents the error.

QEC: Comparison

Algorithm	Execution Time	Accuracy
Perfect Matching	Very slow ✗	Very accurate ✓
Union Find	Very fast ✓	Less accurate ✗
Sparse Blossom*	Fast ✓	Accurate ✓

Trade-off execution time – accuracy!

* Higgott, Oscar, and Craig Gidney. "Sparse Blossom: correcting a million errors per core second with minimum-weight matching." *arXiv preprint arXiv:2303.15933* (2023)

Quantum Error Mitigation

Correcting errors is a direct solution to the problem, but sometimes it cannot be performed. Indeed, it requires the usage of more qubits, and dedicated hardware interfacing with the QC.

An alternative approach is to use error mitigation. Its objective is not to detect what errors happened, but mitigate their effects on the final output from the QC.

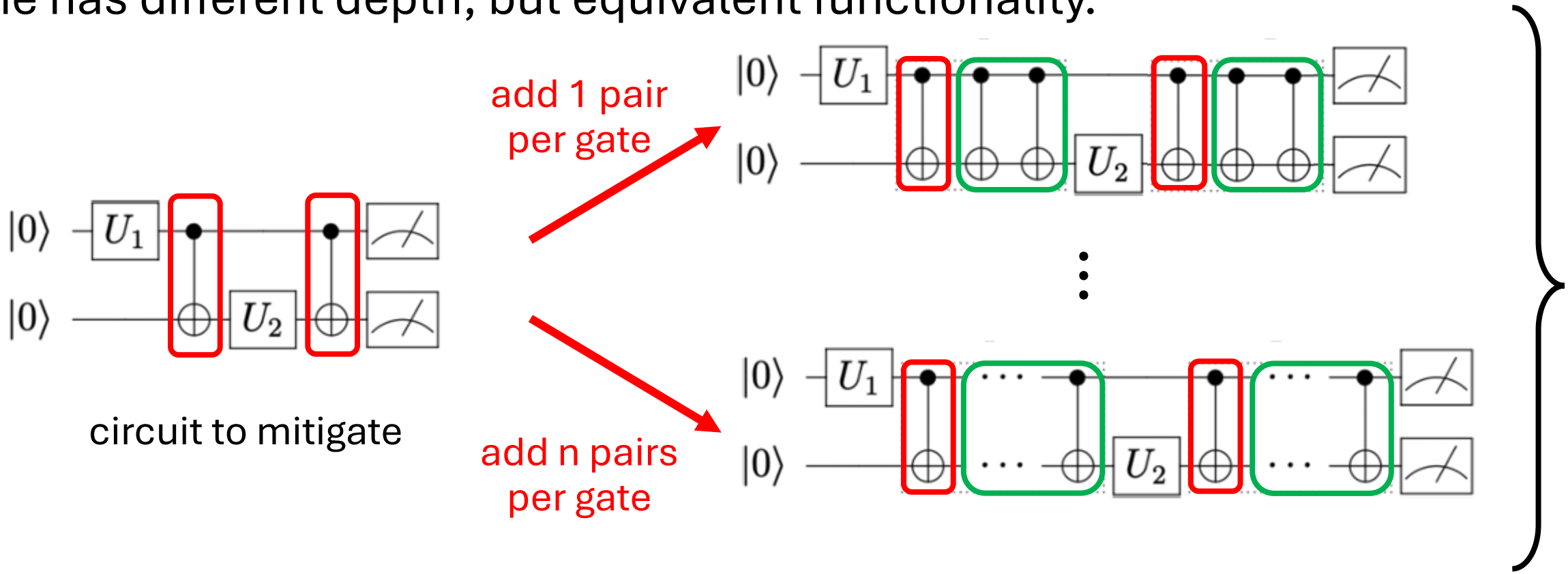
QEM: State of the Art

There are multiple algorithms that can be used to perform QEM. They are mainly based on machine learning and statistical inference.

- Zero-Noise Extrapolation
- Probabilistic Error Cancellation

QEM: Zero-Noise Extrapolation

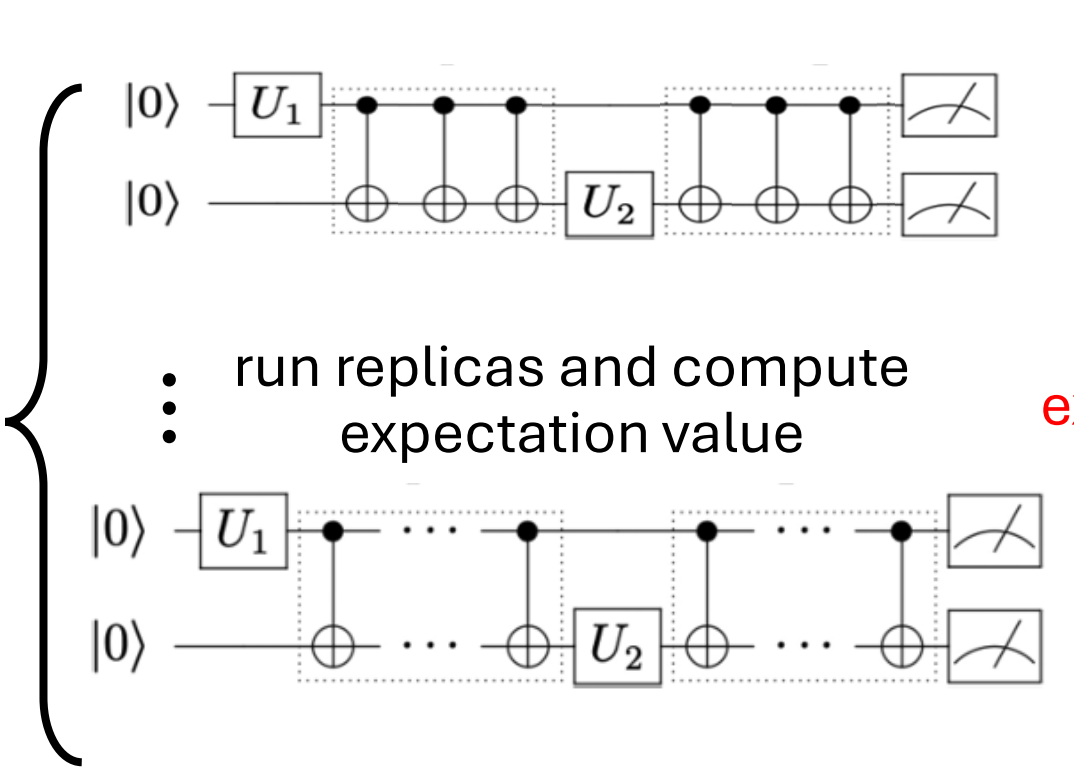
ZNE is based on creating multiple copies of the circuit to run. Each one has different depth, but equivalent functionality.



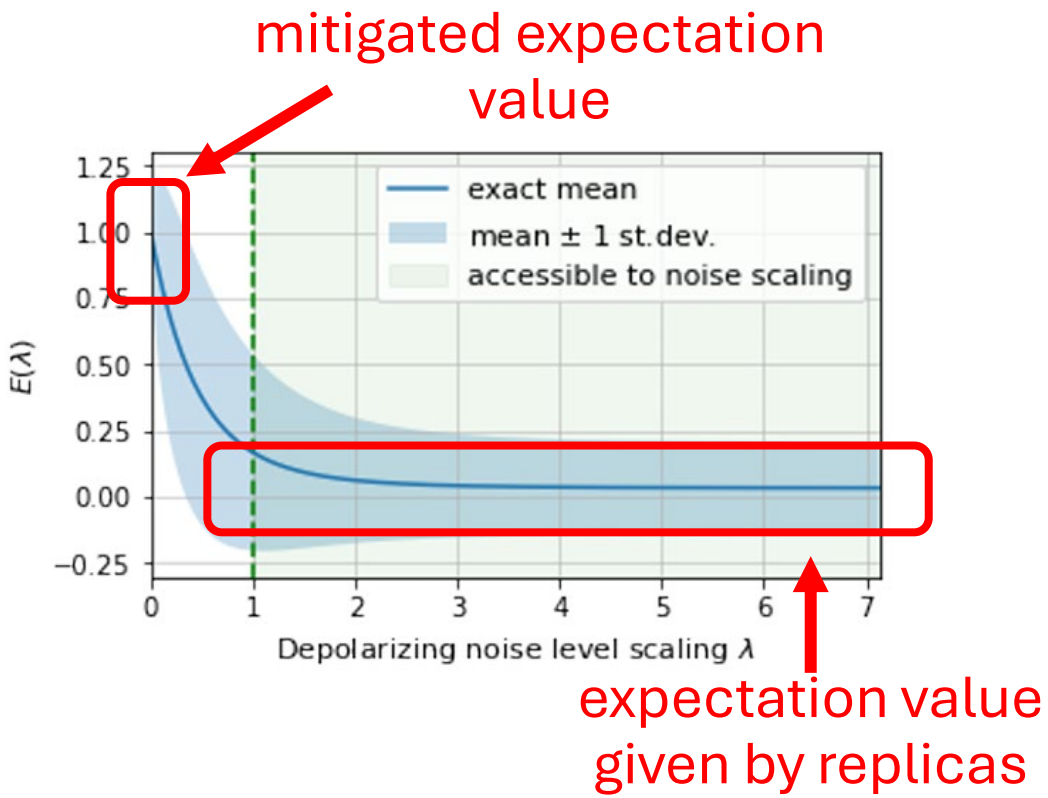
Pascuzzi, Vincent R., et al. "Computationally efficient zero-noise extrapolation for quantum-gate-error mitigation." Physical Review A 105.4

QEM: Zero-Noise Extrapolation

Different depth implies different degree of noise. By extrapolation, we can predict the zero-noise measurement probability.



extrapolation



Giurgica-Tiron, Tudor, et al. "Digital zero noise extrapolation for quantum error mitigation." 2020 IEEE International Conference on Quantum Computing and Engineering (QCE). IEEE, 2020.

QEM: Probabilistic Error Cancellation

This methodology is based on the principle that the noise.free expectation value can be written as a linear combination of expectation values from a set of noisy quantum circuits.

In practice, we start by defining our ideal operation $\mathcal{U}$ as a linear combination of noisy hardware operators: $\mathcal{U} = \sum_n \alpha_n \mathcal{B}_n$.

Then, we perform a Monte Carlo Sampling, where we select the noisy circuit for $\mathcal{B}_n$ with probability $\frac{|\alpha_n|}{\sum_n \alpha_n}$. The fina expectation value for $\mathcal{U}$ will be derived from the expectation values of components $\mathcal{B}_n$.

Mitiq: A Framework For QEM

Mitiq is a Python Framework that allows for applying multiple error mitigation techniques, in order to reduce the effects of noise.

Both techniques described here can be used.

Each technique has a number of parameters to explore, in order to improve performance. For example, ZNE can specify the number of times we duplicate gates, the criteria for duplication, the mathematical tool to perform extrapolation and its hyperparameters.

https://github.com/unitaryfund/mitiq

Mitiq: A Framework For QEM

```
mitiq.zne.zne.execute_with_zne(circuit, executor, observable=None, *,
factory=None, scale_noise=<function fold_gates_at_random>, num_to_average=1)
```

Estimates the error-mitigated expectation value associated to the input circuit, via the application of zero-noise extrapolation (ZNE). [source]

Parameters:

- **circuit** (Union[Circuit, Program, QuantumCircuit, Circuit, QuantumTape, Circuit]) – The input circuit to execute with ZNE.
- **executor** (Union[Executor, Callable[[Union[Circuit, Program, QuantumCircuit, Circuit, QuantumTape, Circuit]], Union[float, MeasurementResult, ndarray]]]) – A Mitiq executor that executes a circuit and returns the unmitigated QuantumResult (e.g. an expectation value).
- **observable** (Optional[Observable]) – Observable to compute the expectation value of. If None, the executor must return an expectation value. Otherwise, the QuantumResult returned by executor is used to compute the expectation of the observable.
- **factory** (Optional[Factory]) – Factory object that determines the zero-noise extrapolation method.
- **scale_noise** (Callable[[Union[Circuit, Program, QuantumCircuit, Circuit, QuantumTape, Circuit], float], Union[Circuit, Program, QuantumCircuit, Circuit, QuantumTape, Circuit]]) – The function for scaling the noise of a quantum circuit. A list of built-in functions can be found in mitiq.zne.scaling.

https://mitiq.readthedocs.io/en/latest/apidoc.html#mitiq.zne.zne.execute_with_zne

Projects

Some ideas for possible projects (you can also suggest something else): PDF file in the dropbox folder.

You must fill in your name in this spreadsheet: here.

Deadlines: January 31st, but it is not strict. Please, reach out to us periodically and give us updates and feedback on the project status.

Thank you so much!

Thank you so much for attending Quantum Computing: A Practical Perspective.

I would really appreciate if you could fill in this very small survey: survey.

This was the first time we had this course. Every feedback is important!

POLITECNICO MILANO 1863
NECST laboratory

Thank you for your attention!

Quantum Computing
A Practical Perspective

Marco Venere

marco.venere@polimi.it

December 3rd, 2024
Politecnico di Milano