QEC: Classical Errors to the Surface Code

Daniel Mandragona

November 3, 2025

Table of Contents

Introduction

- Quantum computers attempt to solve *certain* computational problems faster than classical computers.

Introduction

- Quantum computers attempt to solve *certain* computational problems faster than classical computers.
- However, QCs are incredibly fragile and susceptible to noise.

Introduction

- Quantum computers attempt to solve *certain* computational problems faster than classical computers.
- However, QCs are incredibly fragile and susceptible to noise.
    - The *basic* theory assumes a closed quantum system.

Introduction

- Quantum computers attempt to solve *certain* computational problems faster than classical computers.
- However, QCs are incredibly fragile and susceptible to noise.
  - The *basic* theory assumes a closed quantum system.
- This leads to a paradox:

How can a completely closed quantum system be manipulated by an observer?

What is Error Correction?

- The fundamental idea behind any error correction is to encode information in a **redundant** way.

What is Error Correction?

- The fundamental idea behind any error correction is to encode information in a **redundant** way.
- Redundancy allows us to detect and correct errors without losing information.

What is Error Correction?

- The fundamental idea behind any error correction is to encode information in a **redundant** way.
- Redundancy allows us to detect and correct errors without losing information.
- Unfortunately, this is not so straightforward in the quantum setting.

What is Error Correction?

- The fundamental idea behind any error correction is to encode information in a **redundant** way.
- Redundancy allows us to detect and correct errors without losing information.
- Unfortunately, this is not so straightforward in the quantum setting.
    - Non-classical types of errors.

What is Error Correction?

- The fundamental idea behind any error correction is to encode information in a **redundant** way.
- Redundancy allows us to detect and correct errors without losing information.
- Unfortunately, this is not so straightforward in the quantum setting.
    - Non-classical types of errors.
    - There's no copying a quantum state.

Table of Contents

- Bits are physical, not theoretical, and are therefore subject to noise from:

Classical Error Correction

- Bits are physical, not theoretical, and are therefore subject to noise from:
  - CPUs overheating

Classical Error Correction

- Bits are physical, not theoretical, and are therefore subject to noise from:
  - CPUs overheating
  - Manufacturing defects

Classical Error Correction

- Bits are physical, not theoretical, and are therefore subject to noise from:
  - CPUs overheating
  - Manufacturing defects
  - Cosmic rays...

Classical Error Correction

- Bits are physical, not theoretical, and are therefore subject to noise from:
  - CPUs overheating
  - Manufacturing defects
  - Cosmic rays...
- The situation is much better than in the quantum setting, however.

Classical Error Correction

- Bits are physical, not theoretical, and are therefore subject to noise from:
  - CPUs overheating
  - Manufacturing defects
  - Cosmic rays...
- The situation is much better than in the quantum setting, however.
  - Error rates are extremely low (e.g., one per billion operations).

Classical Error Correction

- Bits are physical, not theoretical, and are therefore subject to noise from:
  - CPUs overheating
  - Manufacturing defects
  - Cosmic rays...
- The situation is much better than in the quantum setting, however.
  - Error rates are extremely low (e.g., one per billion operations).
  - Strong error correction methods, like the Hamming code are still in use (e.g., in RAM) today.

The Repetition Code: Encoding

The simplest classical error-correcting code.

- Goal: Send a single bit (0 or 1) across a noisy channel with a bit-flip probability of p.

The Repetition Code: Encoding

The simplest classical error-correcting code.

- Goal: Send a single bit (0 or 1) across a noisy channel with a bit-flip probability of p.
- Encoding: Repeat the bit three times.

The Repetition Code: Encoding

The simplest classical error-correcting code.

- Goal: Send a single bit (0 or 1) across a noisy channel with a bit-flip probability of p.
- Encoding: Repeat the bit three times.
  - $0 \rightarrow 000$ (Logical 0)

The Repetition Code: Encoding

The simplest classical error-correcting code.

- Goal: Send a single bit (0 or 1) across a noisy channel with a bit-flip probability of p.
- Encoding: Repeat the bit three times.
  - $0 \rightarrow 000$ (Logical 0)
  - $1 \rightarrow 111$ (Logical 1)

The Repetition Code: Encoding

The simplest classical error-correcting code.

- Goal: Send a single bit (0 or 1) across a noisy channel with a bit-flip probability of p.
- Encoding: Repeat the bit three times.
  - $0 \to 000$ (Logical 0)
  - $1 \to 111$ (Logical 1)
- The encoded blocks (000, 111) are called **codewords**.

- Suppose a single bit-flip error occurs during transmission.

The Repetition Code: Decoding

- Suppose a single bit-flip error occurs during transmission.
- Example: We send logical 0 (encoded as 000) and receive 010.

The Repetition Code: Decoding

- Suppose a single bit-flip error occurs during transmission.
- Example: We send logical 0 (encoded as 000) and receive 010.
- Decoding: The receiver uses a **majority vote**.

The Repetition Code: Decoding

- Suppose a single bit-flip error occurs during transmission.
- Example: We send logical 0 (encoded as 000) and receive 010.
- Decoding: The receiver uses a **majority vote**.
- In the case of 010, the majority of bits are 0, so the receiver correctly deduces the original bit was 0.

The Repetition Code: Decoding

- Suppose a single bit-flip error occurs during transmission.
- Example: We send logical 0 (encoded as 000) and receive 010.
- Decoding: The receiver uses a **majority vote**.
- In the case of 010, the majority of bits are 0, so the receiver correctly deduces the original bit was 0.
- This simple code can correct any *single* bit-flip error.

The Repetition Code: Limitations

- The code fails if two or more bits are flipped.

The Repetition Code: Limitations

- The code fails if two or more bits are flipped.
- Example: 000 becomes 110. The majority vote would incorrectly decode this as 1.

The Repetition Code: Limitations

- The code fails if two or more bits are flipped.
- Example: 000 becomes 110. The majority vote would incorrectly decode this as 1.
- What is the probability of failing now?

The Repetition Code: Limitations

- The code fails if two or more bits are flipped.
- Example: 000 becomes 110. The majority vote would incorrectly decode this as 1.
- What is the probability of failing now?
- $\binom{3}{2} p^2(1-p) + \binom{3}{3} p^3 = 3p^2 - 2p^3$

The Repetition Code: Limitations

- The code fails if two or more bits are flipped.
- Example: 000 becomes 110. The majority vote would incorrectly decode this as 1.
- What is the probability of failing now?
- $\binom{3}{2}p^2(1-p) + \binom{3}{3}p^3 = 3p^2 - 2p^3$
- The code is only advantageous if this probability is less than p.

The Repetition Code: Limitations

- The code fails if two or more bits are flipped.
- Example: 000 becomes 110. The majority vote would incorrectly decode this as 1.
- What is the probability of failing now?
- $\binom{3}{2}p^2(1-p) + \binom{3}{3}p^3 = 3p^2 - 2p^3$
- The code is only advantageous if this probability is less than p.
- We can **improve** the code by using longer odd-length repetitions.

Table of Contents

Linear Codes

- Linear codes are a more sophisticated type of code. Ex. Hamming Code.

Linear Codes

- Linear codes are a more sophisticated type of code. Ex. Hamming Code.
- A code is **linear** if the sum of any two codewords is also a codeword.

Linear Codes

- Linear codes are a more sophisticated type of code. Ex. Hamming Code.
- A code is **linear** if the sum of any two codewords is also a codeword.
  - "Sum" here means bitwise XOR.

Linear Codes

- Linear codes are a more sophisticated type of code. Ex. Hamming Code.
- A code is **linear** if the sum of any two codewords is also a codeword.
  - "Sum" here means bitwise XOR.
  - This operation turns the set of bit strings into a vector space modulo 2.

Linear Codes

- Linear codes are a more sophisticated type of code. Ex. Hamming Code.
- A code is **linear** if the sum of any two codewords is also a codeword.
  - "Sum" here means bitwise XOR.
  - This operation turns the set of bit strings into a vector space modulo 2.
- Linear codes are described by two matrices:

Linear Codes

- Linear codes are a more sophisticated type of code. Ex. Hamming Code.
- A code is **linear** if the sum of any two codewords is also a codeword.
  - "Sum" here means bitwise XOR.
  - This operation turns the set of bit strings into a vector space modulo 2.
- Linear codes are described by two matrices:
  - **Generator Matrix (G):** Encodes the message.

Linear Codes

- Linear codes are a more sophisticated type of code. Ex. Hamming Code.
- A code is **linear** if the sum of any two codewords is also a codeword.
  - "Sum" here means bitwise XOR.
  - This operation turns the set of bit strings into a vector space modulo 2.
- Linear codes are described by two matrices:
  - **Generator Matrix (G):** Encodes the message.
  - **Parity Check Matrix (H):** Detects errors.

The [7,4] Hamming Code: Encoding

- Encodes 4 logical bits into a 7-bit codeword.

The [7,4] Hamming Code: Encoding

- Encodes 4 logical bits into a 7-bit codeword.
- The generator matrix G is:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

The [7,4] Hamming Code: Encoding

- Encodes 4 logical bits into a 7-bit codeword.
- The generator matrix G is:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

- A 4-bit message m is encoded by $c = mG$, what is the bitstring 1011 encoded as?

The [7,4] Hamming Code: Encoding

- Encodes 4 logical bits into a 7-bit codeword.
- The generator matrix G is:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

- A 4-bit message m is encoded by $c = mG$, what is the bitstring 1011 encoded as?
- Example: The message 1011 is encoded as:

$$(1, 0, 1, 1)G = (1, 0, 1, 1, 0, 1, 0)$$

The [7,4] Hamming Code: Parity Check

- The parity check matrix H has the property that for any valid codeword c, $Hc^T = 0$.

The [7,4] Hamming Code: Parity Check

- The parity check matrix H has the property that for any valid codeword c, $Hc^T = 0$.
- This means the codewords are in the **null space** of H.

The [7,4] Hamming Code: Parity Check

- The parity check matrix H has the property that for any valid codeword c, $Hc^T = 0$.
- This means the codewords are in the **null space** of H.
- The parity check matrix for the [7,4] Hamming code is:

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

The [7,4] Hamming Code: Parity Check

- The parity check matrix H has the property that for any valid codeword c, $Hc^T = 0$.
- This means the codewords are in the **null space** of H.
- The parity check matrix for the [7,4] Hamming code is:

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

- If the result of Hc^T is non-zero, an error has been detected.

The [7,4] Hamming Code: Error Correction

- If a codeword c is sent and an error e occurs, the received vector is $r = c + e$.

The [7,4] Hamming Code: Error Correction

- If a codeword c is sent and an error e occurs, the received vector is $r = c + e$.
- To find the error, we compute the **syndrome**:

$$s = Hr^T = H(c+e)^T = Hc^T + He^T = He^T$$

The [7,4] Hamming Code: Error Correction

- If a codeword c is sent and an error e occurs, the received vector is $r = c + e$.
- To find the error, we compute the **syndrome**:

$$s = Hr^T = H(c+e)^T = Hc^T + He^T = He^T$$

- The syndrome depends only on the error, not the original codeword!

The [7,4] Hamming Code: Error Correction

- If a codeword c is sent and an error e occurs, the received vector is $r = c + e$.
- To find the error, we compute the **syndrome**:

$$s = Hr^T = H(c+e)^T = Hc^T + He^T = He^T$$

- The syndrome depends only on the error, not the original codeword!
- Now, assume e is a single-bit error, so its a vector with 0's everywhere except one place.

The [7,4] Hamming Code: Error Correction

- If a codeword c is sent and an error e occurs, the received vector is $r = c + e$.
- To find the error, we compute the **syndrome**:

$$s = Hr^T = H(c+e)^T = Hc^T + He^T = He^T$$

- The syndrome depends only on the error, not the original codeword!
- Now, assume e is a single-bit error, so its a vector with 0's everywhere except one place.
- What does the syndrome end up being then?

The [7,4] Hamming Code: Error Correction

- If a codeword c is sent and an error e occurs, the received vector is $r = c + e$.
- To find the error, we compute the **syndrome**:

$$s = Hr^T = H(c + e)^T = Hc^T + He^T = He^T$$

- The syndrome depends only on the error, not the original codeword!
- Now, assume e is a single-bit error, so its a vector with 0's everywhere except one place.
- What does the syndrome end up being then?
- For a single-bit error at position i, the syndrome s will be equal to the i-th column of H.

The [7,4] Hamming Code: Error Correction

- If a codeword c is sent and an error e occurs, the received vector is $r = c + e$.
- To find the error, we compute the **syndrome**:

$$s = Hr^T = H(c+e)^T = Hc^T + He^T = He^T$$

- The syndrome depends only on the error, not the original codeword!
- Now, assume e is a single-bit error, so its a vector with 0's everywhere except one place.
- What does the syndrome end up being then?
- For a single-bit error at position i, the syndrome s will be equal to the i-th column of H.
- This allows us to identify and correct the single-bit error.

Hamming Distance and Limitations

- The **Hamming distance** is the number of bits in which two codewords differ.

Hamming Distance and Limitations

- The **Hamming distance** is the number of bits in which two codewords differ.
- For the repetition code (000, 111), the distance is 3.

Hamming Distance and Limitations

- The **Hamming distance** is the number of bits in which two codewords differ.
- For the repetition code (000, 111), the distance is 3.
- For the [7,4] Hamming code, the distance is also 3.

Hamming Distance and Limitations

- The **Hamming distance** is the number of bits in which two codewords differ.
- For the repetition code (000, 111), the distance is 3.
- For the [7,4] Hamming code, the distance is also 3.
- A code with distance d can:

Hamming Distance and Limitations

- The **Hamming distance** is the number of bits in which two codewords differ.
- For the repetition code (000, 111), the distance is 3.
- For the [7,4] Hamming code, the distance is also 3.
- A code with distance d can:
  - Detect up to $d-1$ errors.

Hamming Distance and Limitations

- The **Hamming distance** is the number of bits in which two codewords differ.
- For the repetition code (000, 111), the distance is 3.
- For the [7,4] Hamming code, the distance is also 3.
- A code with distance d can:
    - Detect up to $d - 1$ errors.
    - Correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors.

Hamming Distance and Limitations

- The **Hamming distance** is the number of bits in which two codewords differ.
- For the repetition code (000, 111), the distance is 3.
- For the [7,4] Hamming code, the distance is also 3.
- A code with distance d can:
  - Detect up to $d-1$ errors.
  - Correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors.
- **Benefit:** More efficient than the repetition code (7 bits for 4 logical bits vs. 3 for 1).

Table of Contents

Why Quantum Error Correction is Harder

QEC is more complex than classical error correction for several reasons:

1. **More error types:** Beyond bit-flips, qubits can have phase-flips, and a continuous range of other errors.
2. **No-Cloning Theorem:** We cannot simply copy a qubit to create redundancy.
3. **Measurement is destructive:** Measuring a qubit to check for errors collapses its state, destroying the information we want to protect.

Seems difficult...

The 3-Qubit Bit-Flip Code: Encoding

A quantum version of the repetition code to correct bit-flip errors.

- Goal: Encode a single logical qubit:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

The 3-Qubit Bit-Flip Code: Encoding

A quantum version of the repetition code to correct bit-flip errors.

- Goal: Encode a single logical qubit:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

- Encoding: Create an entangled three-qubit state:

$$|\psi_L\rangle = \alpha|000\rangle + \beta|111\rangle$$

The 3-Qubit Bit-Flip Code: Encoding

A quantum version of the repetition code to correct bit-flip errors.

- Goal: Encode a single logical qubit:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

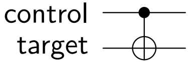
- Encoding: Create an entangled three-qubit state:

$$|\psi_L\rangle = \alpha|000\rangle + \beta|111\rangle$$

- This is NOT cloning. The No-Cloning Theorem forbids making independent copies like:

$$|\psi_{\text{copied}}\rangle = (\alpha|0\rangle + \beta|1\rangle)^{\otimes 3}$$

- The Controlled-NOT (CNOT) gate is a two-qubit gate that flips the target qubit if the control qubit is $|1\rangle$.

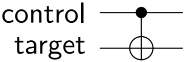


The CNOT Gate

- The Controlled-NOT (CNOT) gate is a two-qubit gate that flips the target qubit if the control qubit is $|1\rangle$.
- Matrix representation:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

acting on bases: $|00\rangle, |01\rangle, |10\rangle, |11\rangle$.



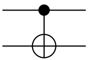
The CNOT Gate

- The Controlled-NOT (CNOT) gate is a two-qubit gate that flips the target qubit if the control qubit is $|1\rangle$.
- Matrix representation:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

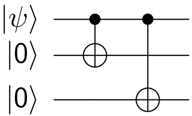
  acting on bases: $|00\rangle, |01\rangle, |10\rangle, |11\rangle$.
- Circuit diagram:

control
target



Encoding Circuit for 3 Qubit Bit-Flip Code

We use two CNOT gates to create the logical state $|\psi_L\rangle$:



$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ 0 \\ \beta \\ 0 \end{pmatrix}$$

Example (Creating $|\psi_L\rangle$)

The circuit transforms the initial state $|\psi\rangle|0\rangle|0\rangle$:

$$\begin{aligned} |\psi_{initial}\rangle &= (\alpha|0\rangle + \beta|1\rangle)|00\rangle = \alpha|000\rangle + \beta|100\rangle \\ &\xrightarrow{CNOT_{12}} \alpha|000\rangle + \beta|110\rangle \\ &\xrightarrow{CNOT_{13}} \alpha|000\rangle + \beta|111\rangle = |\psi_L\rangle \end{aligned}$$

Quantum Errors: Pauli Operators

- A **bit-flip error** is represented by the Pauli X operator:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

It flips the state: $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$.

Quantum Errors: Pauli Operators

- A **bit-flip error** is represented by the Pauli X operator:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

  It flips the state: $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$.

- A **phase-flip error** is represented by the Pauli Z operator:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

  It flips the phase: $Z|0\rangle = |0\rangle$ and $Z|1\rangle = -|1\rangle$.

Quantum Errors: Pauli Operators

- A **bit-flip error** is represented by the Pauli X operator:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

  It flips the state: $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$.

- A **phase-flip error** is represented by the Pauli Z operator:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

  It flips the phase: $Z|0\rangle = |0\rangle$ and $Z|1\rangle = -|1\rangle$.

- We even have a third error represented by the Pauli Y operator:

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

Quantum Errors: Pauli Operators

- A **bit-flip error** is represented by the Pauli X operator:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

It flips the state: $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$.

- A **phase-flip error** is represented by the Pauli Z operator:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

It flips the phase: $Z|0\rangle = |0\rangle$ and $Z|1\rangle = -|1\rangle$.

- We even have a third error represented by the Pauli Y operator:

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

- These operators satisfy the property: $XYZ = i$, so $Y = iXZ$, and are *their own inverses*.

Detecting Errors Without Measurement

How do we detect an error like X_1 on $|\psi_L\rangle$ without collapsing the state?

- We can't measure the qubits directly.

Detecting Errors Without Measurement

How do we detect an error like X_1 on $|\psi_L\rangle$ without collapsing the state?

- We can't measure the qubits directly.
- Solution: Use **stabilizer** measurements. These are like the parity checks from classic error correction.

Detecting Errors Without Measurement

How do we detect an error like X_1 on $|\psi_L\rangle$ without collapsing the state?

- We can't measure the qubits directly.
- Solution: Use **stabilizer** measurements. These are like the parity checks from classic error correction.
- For the bit-flip code, the stabilizers are $Z_1 Z_2$ and $Z_2 Z_3$.

Detecting Errors Without Measurement

How do we detect an error like X_1 on $|\psi_L\rangle$ without collapsing the state?

- We can't measure the qubits directly.
- Solution: Use **stabilizer** measurements. These are like the parity checks from classic error correction.
- For the bit-flip code, the stabilizers are $Z_1 Z_2$ and $Z_2 Z_3$.
- We measure a *joint* property of these operators, not the individual qubits. We collapse the *correlation*, not the individual states.

Measurement Intuition

- Quantum measurement is about defining an observable property of your system.

Measurement Intuition

- Quantum measurement is about defining an observable property of your system.
- The property is *observable* meaning that the values this property can hold are real-valued.

Measurement Intuition

- Quantum measurement is about defining an observable property of your system.
- The property is *observable* meaning that the values this property can hold are real-valued.
- Measuring an observable is then about projecting your system onto the system states associated to each of these values.

Measurement Intuition

- Quantum measurement is about defining an observable property of your system.
- The property is *observable* meaning that the values this property can hold are real-valued.
- Measuring an observable is then about projecting your system onto the system states associated to each of these values.
- These observables in QC are matrices which can be decomposed into a set of projection matrices:

$$\mathcal{O} = \lambda_1 P_1 + \cdots + \lambda_k P_k$$

Measurement Intuition

- Quantum measurement is about defining an observable property of your system.
- The property is *observable* meaning that the values this property can hold are real-valued.
- Measuring an observable is then about projecting your system onto the system states associated to each of these values.
- These observables in QC are matrices which can be decomposed into a set of projection matrices:

$$\mathcal{O} = \lambda_1 P_1 + \cdots + \lambda_k P_k$$

- Your quantum state can then be *written* w.r.t. the bases of each one of the Projection Matrices.

Measurement Intuition

- Quantum measurement is about defining an observable property of your system.
- The property is *observable* meaning that the values this property can hold are real-valued.
- Measuring an observable is then about projecting your system onto the system states associated to each of these values.
- These observables in QC are matrices which can be decomposed into a set of projection matrices:

$$\mathcal{O} = \lambda_1 P_1 + \cdots + \lambda_k P_k$$

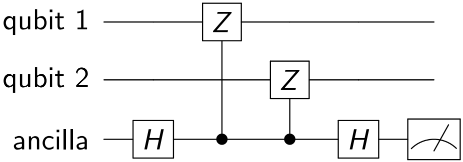
- Your quantum state can then be *written* w.r.t. the bases of each one of the Projection Matrices.
- Measurement collapses the state onto the eigenspace associated with the measured eigenvalue. The probability of collapsing to a specific eigenspace k is $p(k) = \|P_k|\psi\rangle\|^2$.".

Stabilizer Measurement Circuit

- So how would we *measure* a $Z_1 Z_2$ observable?

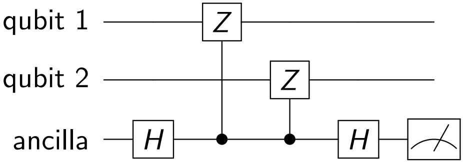
Stabilizer Measurement Circuit

- So how would we *measure* a $Z_1 Z_2$ observable?
- This circuit measures the $Z_1 Z_2$ stabilizer using an ancilla qubit.



Stabilizer Measurement Circuit

- So how would we *measure* a $Z_1 Z_2$ observable?
- This circuit measures the $Z_1 Z_2$ stabilizer using an ancilla qubit.



- The measurement outcome of the ancilla tells us the eigenvalue of $Z_1 Z_2$.

State Evolution: No Error

- Note: $|\psi_L\rangle = \alpha|000\rangle + \beta|111\rangle$ is a $+1$ eigenstate of $Z_1 Z_2$.

State Evolution: No Error

- Note: $|\psi_L\rangle = \alpha|000\rangle + \beta|111\rangle$ is a $+1$ eigenstate of $Z_1 Z_2$.
- Initial state: $|\psi_L\rangle|0\rangle_a = (\alpha|000\rangle + \beta|111\rangle)|0\rangle_a$.

State Evolution: No Error

- Note: $|\psi_L\rangle = \alpha|000\rangle + \beta|111\rangle$ is a $+1$ eigenstate of $Z_1 Z_2$.
- Initial state: $|\psi_L\rangle|0\rangle_a = (\alpha|000\rangle + \beta|111\rangle)|0\rangle_a$.
- **1. Hadamard on ancilla:**

$$|\psi_L\rangle \frac{1}{\sqrt{2}}(|0\rangle_a + |1\rangle_a) = |\psi_L\rangle \otimes |+\rangle$$

State Evolution: No Error

- Note: $|\psi_L\rangle = \alpha|000\rangle + \beta|111\rangle$ is a $+1$ eigenstate of Z_1Z_2.
- Initial state: $|\psi_L\rangle|0\rangle_a = (\alpha|000\rangle + \beta|111\rangle)|0\rangle_a$.
- **1. Hadamard on ancilla:**

$$|\psi_L\rangle\frac{1}{\sqrt{2}}(|0\rangle_a + |1\rangle_a) = |\psi_L\rangle \otimes |+\rangle$$

- **2. Controlled-Z gates:** Since $Z_1Z_2|\psi_L\rangle = |\psi_L\rangle$, the state is unchanged.

$$|\psi_L\rangle \otimes |+\rangle_a$$

State Evolution: No Error

- Note: $|\psi_L\rangle = \alpha|000\rangle + \beta|111\rangle$ is a $+1$ eigenstate of $Z_1 Z_2$.
- Initial state: $|\psi_L\rangle|0\rangle_a = (\alpha|000\rangle + \beta|111\rangle)|0\rangle_a$.
- **1. Hadamard on ancilla:**

$$|\psi_L\rangle \frac{1}{\sqrt{2}}(|0\rangle_a + |1\rangle_a) = |\psi_L\rangle \otimes |+\rangle$$

- **2. Controlled-Z gates:** Since $Z_1 Z_2|\psi_L\rangle = |\psi_L\rangle$, the state is unchanged.

$$|\psi_L\rangle \otimes |+\rangle_a$$

- **3. Hadamard on ancilla:**

$$|\psi_L\rangle \otimes H|+\rangle_a = |\psi_L\rangle|0\rangle_a$$

State Evolution: No Error

- Note: $|\psi_L\rangle = \alpha|000\rangle + \beta|111\rangle$ is a $+1$ eigenstate of $Z_1 Z_2$.
- Initial state: $|\psi_L\rangle|0\rangle_a = (\alpha|000\rangle + \beta|111\rangle)|0\rangle_a$.
- **1. Hadamard on ancilla:**

$$|\psi_L\rangle \frac{1}{\sqrt{2}}(|0\rangle_a + |1\rangle_a) = |\psi_L\rangle \otimes |+\rangle$$

- **2. Controlled-Z gates:** Since $Z_1 Z_2|\psi_L\rangle = |\psi_L\rangle$, the state is unchanged.

$$|\psi_L\rangle \otimes |+\rangle_a$$

- **3. Hadamard on ancilla:**

$$|\psi_L\rangle \otimes H|+\rangle_a = |\psi_L\rangle|0\rangle_a$$

- **4. Measure ancilla:** Result is **0** with certainty. No error detected.

State Evolution: With an X_1 Error

- Note: $|\psi_L\rangle = \alpha|100\rangle + \beta|011\rangle$ is a -1 eigenstate of $Z_1 Z_2$.

State Evolution: With an X_1 Error

- Note: $|\psi_L\rangle = \alpha|100\rangle + \beta|011\rangle$ is a -1 eigenstate of $Z_1 Z_2$.
- Initial state: $|\psi_{err}\rangle = X_1|\psi_L\rangle = \alpha|100\rangle + \beta|011\rangle$.

State Evolution: With an X_1 Error

- Note: $|\psi_L\rangle = \alpha|100\rangle + \beta|011\rangle$ is a -1 eigenstate of $Z_1 Z_2$.
- Initial state: $|\psi_{err}\rangle = X_1|\psi_L\rangle = \alpha|100\rangle + \beta|011\rangle$.
- **1. Hadamard on ancilla:**

$$|\psi_{err}\rangle \frac{1}{\sqrt{2}}(|0\rangle_a + |1\rangle_a) = |\psi_{err}\rangle \otimes |+\rangle$$

State Evolution: With an X_1 Error

- Note: $|\psi_L\rangle = \alpha|100\rangle + \beta|011\rangle$ is a -1 eigenstate of $Z_1 Z_2$.
- Initial state: $|\psi_{err}\rangle = X_1|\psi_L\rangle = \alpha|100\rangle + \beta|011\rangle$.
- **1. Hadamard on ancilla:**

$$|\psi_{err}\rangle \frac{1}{\sqrt{2}}(|0\rangle_a + |1\rangle_a) = |\psi_{err}\rangle \otimes |+\rangle$$

- **2. Controlled-Z gates:** Since $Z_1 Z_2|\psi_{err}\rangle = -|\psi_{err}\rangle$, the state becomes:

$$\begin{aligned} C_a\text{-}Z_1 Z_2\left(|\psi_{err}\rangle \frac{1}{\sqrt{2}}(|0\rangle_a + |1\rangle_a)\right) &= \frac{1}{\sqrt{2}}(|\psi_{err}\rangle|0\rangle_a - |\psi_{err}\rangle|1\rangle_a) \\ &= |\psi_{err}\rangle \otimes |-\rangle_a \end{aligned}$$

State Evolution: With an X_1 Error

- Note: $|\psi_L\rangle = \alpha|100\rangle + \beta|011\rangle$ is a -1 eigenstate of $Z_1 Z_2$.
- Initial state: $|\psi_{err}\rangle = X_1|\psi_L\rangle = \alpha|100\rangle + \beta|011\rangle$.
- **1. Hadamard on ancilla:**

$$|\psi_{err}\rangle \frac{1}{\sqrt{2}}(|0\rangle_a + |1\rangle_a) = |\psi_{err}\rangle \otimes |+\rangle$$

- **2. Controlled-Z gates:** Since $Z_1 Z_2|\psi_{err}\rangle = -|\psi_{err}\rangle$, the state becomes:

$$\begin{aligned} C_a\text{-}Z_1 Z_2\left(|\psi_{err}\rangle \frac{1}{\sqrt{2}}(|0\rangle_a + |1\rangle_a)\right) &= \frac{1}{\sqrt{2}}(|\psi_{err}\rangle|0\rangle_a - |\psi_{err}\rangle|1\rangle_a) \\ &= |\psi_{err}\rangle \otimes |-\rangle_a \end{aligned}$$

- **3. Hadamard on ancilla:**

$$|\psi_{err}\rangle \otimes H|-\rangle_a = |\psi_{err}\rangle|1\rangle_a$$

State Evolution: With an X_1 Error

- Note: $|\psi_L\rangle = \alpha|100\rangle + \beta|011\rangle$ is a -1 eigenstate of $Z_1 Z_2$.
- Initial state: $|\psi_{err}\rangle = X_1|\psi_L\rangle = \alpha|100\rangle + \beta|011\rangle$.
- **1. Hadamard on ancilla:**

$$|\psi_{err}\rangle \frac{1}{\sqrt{2}}(|0\rangle_a + |1\rangle_a) = |\psi_{err}\rangle \otimes |+\rangle$$

- **2. Controlled-Z gates:** Since $Z_1 Z_2|\psi_{err}\rangle = -|\psi_{err}\rangle$, the state becomes:

$$\begin{aligned} C_a\text{-}Z_1 Z_2\left(|\psi_{err}\rangle \frac{1}{\sqrt{2}}(|0\rangle_a + |1\rangle_a)\right) &= \frac{1}{\sqrt{2}}(|\psi_{err}\rangle|0\rangle_a - |\psi_{err}\rangle|1\rangle_a) \\ &= |\psi_{err}\rangle \otimes |-\rangle_a \end{aligned}$$

- **3. Hadamard on ancilla:**

$$|\psi_{err}\rangle \otimes H|-\rangle_a = |\psi_{err}\rangle|1\rangle_a$$

- **4. Measure ancilla:** Result is **1** with certainty. Error detected!

Error Syndromes

The measurement outcomes of the stabilizers form the **error syndrome**.

- The logical state $|\psi_L\rangle$ is a $+1$ eigenstate of both Z_1Z_2 and Z_2Z_3.

Error Syndromes

The measurement outcomes of the stabilizers form the **error syndrome**.

- The logical state $|\psi_L\rangle$ is a $+1$ eigenstate of both Z_1Z_2 and Z_2Z_3.
- An error like X_1 changes the eigenvalues:

$$Z_1Z_2(X_1|\psi_L\rangle) = -1(X_1|\psi_L\rangle)$$
$$Z_2Z_3(X_1|\psi_L\rangle) = +1(X_1|\psi_L\rangle)$$

Error Syndromes

The measurement outcomes of the stabilizers form the **error syndrome**.

- The logical state $|\psi_L\rangle$ is a $+1$ eigenstate of both Z_1Z_2 and Z_2Z_3.
- An error like X_1 changes the eigenvalues:

$$
\begin{aligned}
Z_1Z_2(X_1|\psi_L\rangle) &= -1(X_1|\psi_L\rangle) \\
Z_2Z_3(X_1|\psi_L\rangle) &= +1(X_1|\psi_L\rangle)
\end{aligned}
$$

- The syndrome identifies the error:

Error Syndromes

The measurement outcomes of the stabilizers form the **error syndrome**.

- The logical state $|\psi_L\rangle$ is a $+1$ eigenstate of both Z_1Z_2 and Z_2Z_3.
- An error like X_1 changes the eigenvalues:

$$Z_1Z_2(X_1|\psi_L\rangle) = -1(X_1|\psi_L\rangle)$$
$$Z_2Z_3(X_1|\psi_L\rangle) = +1(X_1|\psi_L\rangle)$$

- The syndrome identifies the error:
  - Syndrome (-1, +1) $\implies$ X_1 error

Error Syndromes

The measurement outcomes of the stabilizers form the **error syndrome**.

- The logical state $|\psi_L\rangle$ is a $+1$ eigenstate of both Z_1Z_2 and Z_2Z_3.
- An error like X_1 changes the eigenvalues:

$$Z_1Z_2(X_1|\psi_L\rangle) = -1(X_1|\psi_L\rangle)$$
$$Z_2Z_3(X_1|\psi_L\rangle) = +1(X_1|\psi_L\rangle)$$

- The syndrome identifies the error:
  - Syndrome (-1, +1) $\implies$ X_1 error
  - Syndrome (+1, -1) $\implies$ X_3 error

Error Syndromes

The measurement outcomes of the stabilizers form the **error syndrome**.

- The logical state $|\psi_L\rangle$ is a $+1$ eigenstate of both Z_1Z_2 and Z_2Z_3.
- An error like X_1 changes the eigenvalues:

$$Z_1Z_2(X_1|\psi_L\rangle) = -1(X_1|\psi_L\rangle)$$
$$Z_2Z_3(X_1|\psi_L\rangle) = +1(X_1|\psi_L\rangle)$$

- The syndrome identifies the error:
  - Syndrome (-1, +1) $\implies$ X_1 error
  - Syndrome (+1, -1) $\implies$ X_3 error
  - Syndrome (-1, -1) $\implies$ X_2 error

Error Syndromes

The measurement outcomes of the stabilizers form the **error syndrome**.

- The logical state $|\psi_L\rangle$ is a $+1$ eigenstate of both Z_1Z_2 and Z_2Z_3.
- An error like X_1 changes the eigenvalues:

$$Z_1Z_2(X_1|\psi_L\rangle) = -1(X_1|\psi_L\rangle)$$
$$Z_2Z_3(X_1|\psi_L\rangle) = +1(X_1|\psi_L\rangle)$$

- The syndrome identifies the error:
  - Syndrome (-1, +1) $\implies$ X_1 error
  - Syndrome (+1, -1) $\implies$ X_3 error
  - Syndrome (-1, -1) $\implies$ X_2 error
  - Syndrome (+1, +1) $\implies$ No error

Error Syndromes

The measurement outcomes of the stabilizers form the **error syndrome**.

- The logical state $|\psi_L\rangle$ is a $+1$ eigenstate of both Z_1Z_2 and Z_2Z_3.
- An error like X_1 changes the eigenvalues:

$$Z_1Z_2(X_1|\psi_L\rangle) = -1(X_1|\psi_L\rangle)$$
$$Z_2Z_3(X_1|\psi_L\rangle) = +1(X_1|\psi_L\rangle)$$

- The syndrome identifies the error:
  - Syndrome (-1, +1) $\implies$ X_1 error
  - Syndrome (+1, -1) $\implies$ X_3 error
  - Syndrome (-1, -1) $\implies$ X_2 error
  - Syndrome (+1, +1) $\implies$ No error
- Once identified, we apply the same operator again to correct it.

Table of Contents

The 3-Qubit Phase-Flip Code

- Now that we can correct bit-flip errors, we must handle **phase-flips** (Z errors).

$$\alpha|0\rangle + \beta|1\rangle \Leftrightarrow \alpha|0\rangle - \beta|1\rangle$$

The 3-Qubit Phase-Flip Code

- Now that we can correct bit-flip errors, we must handle **phase-flips** (Z errors).

$$\alpha|0\rangle + \beta|1\rangle \Leftrightarrow \alpha|0\rangle - \beta|1\rangle$$

- A phase-flip becomes a "bit-flip" in the Hadamard basis $|+\rangle, |-\rangle$.

The 3-Qubit Phase-Flip Code

- Now that we can correct bit-flip errors, we must handle **phase-flips** (Z errors).

$$\alpha|0\rangle + \beta|1\rangle \Leftrightarrow \alpha|0\rangle - \beta|1\rangle$$

- A phase-flip becomes a "bit-flip" in the Hadamard basis $|+\rangle, |-\rangle$.
  - $Z|+\rangle = |-\rangle$ and $Z|-\rangle = |+\rangle$.

The 3-Qubit Phase-Flip Code

- Now that we can correct bit-flip errors, we must handle **phase-flips** (Z errors).

$$\alpha|0\rangle + \beta|1\rangle \Leftrightarrow \alpha|0\rangle - \beta|1\rangle$$

- A phase-flip becomes a "bit-flip" in the Hadamard basis $|+\rangle, |-\rangle$.
  - $Z|+\rangle = |-\rangle$ and $Z|-\rangle = |+\rangle$.
- We use a similar code to the bit-flip code, but in a different basis.

The 3-Qubit Phase-Flip Code

- Now that we can correct bit-flip errors, we must handle **phase-flips** (Z errors).

$$\alpha|0\rangle + \beta|1\rangle \Leftrightarrow \alpha|0\rangle - \beta|1\rangle$$

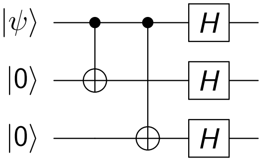
- A phase-flip becomes a "bit-flip" in the Hadamard basis $|+\rangle, |-\rangle$.
  - $Z|+\rangle = |-\rangle$ and $Z|-\rangle = |+\rangle$.
- We use a similar code to the bit-flip code, but in a different basis.
- **Encoding**:

$$|\psi_L\rangle = \alpha|+++\rangle + \beta|---\rangle$$

where $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

Phase-Flip Code: Encoding Circuit

The encoding circuit uses Hadamards to change basis.



- The stabilizers for this code are X_1X_2 and X_2X_3.
- A Z error on one qubit will flip the sign of one or both stabilizer measurements, revealing the error.

Phase-Flip Code: Syndrome Measurement

The syndrome measurement circuit is analogous to the bit-flip code's, but with controlled-X gates.

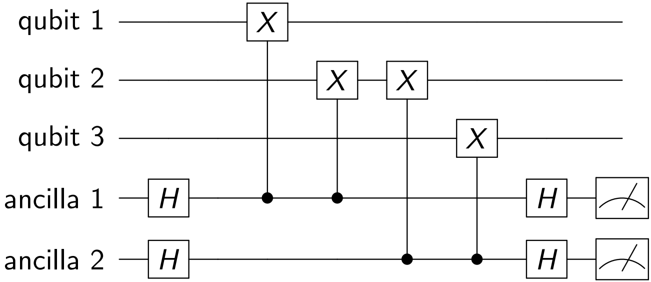


Figure: Circuit for measuring errors in the phase-flip code.

The Shor Code: The First QEC Code

- The first quantum error correcting code, created by Peter Shor in 1995.

The Shor Code: The First QEC Code

- The first quantum error correcting code, created by Peter Shor in 1995.
- It combines the bit-flip and phase-flip codes to correct any arbitrary single-qubit error.

The Shor Code: The First QEC Code

- The first quantum error correcting code, created by Peter Shor in 1995.
- It combines the bit-flip and phase-flip codes to correct any arbitrary single-qubit error.
- It encodes 1 logical qubit into 9 physical qubits.

Shor Code: Encoding

Encoding is a two-step concatenation:

1. **Outer Code (Phase-Flip):** The logical qubit is first encoded to 3 qubits.
   - $\alpha|0\rangle + \beta|1\rangle \to \alpha|+++\rangle + \beta|---\rangle$
2. **Inner Code (Bit-Flip):** Each of these 3 qubits is then encoded into 3 more qubits.
   - $|+\rangle \to \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$
   - $|-\rangle \to \frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$

This results in a 9-qubit state:

$$|0_L\rangle \to \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle)$$
$$|1_L\rangle \to \frac{1}{2\sqrt{2}}(|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle)$$

How Shor Code Corrects Bit-Flips

The 9 qubits are grouped into three blocks of three. Each block is a bit-flip code.

- If a single bit-flip error occurs on any qubit (e.g., q_1), it happens within one block.

How Shor Code Corrects Bit-Flips

The 9 qubits are grouped into three blocks of three. Each block is a bit-flip code.

- If a single bit-flip error occurs on any qubit (e.g., q_1), it happens within one block.
- The inner code's stabilizers for that block ($Z_1 Z_2, Z_2 Z_3$) will detect the error.

How Shor Code Corrects Bit-Flips

The 9 qubits are grouped into three blocks of three. Each block is a bit-flip code.

- If a single bit-flip error occurs on any qubit (e.g., q_1), it happens within one block.
- The inner code's stabilizers for that block (Z_1Z_2, Z_2Z_3) will detect the error.
- The syndrome identifies the errored qubit.

How Shor Code Corrects Bit-Flips

The 9 qubits are grouped into three blocks of three. Each block is a bit-flip code.

- If a single bit-flip error occurs on any qubit (e.g., q_1), it happens within one block.
- The inner code's stabilizers for that block (Z_1Z_2, Z_2Z_3) will detect the error.
- The syndrome identifies the errored qubit.
- We apply another X gate to that qubit to correct it.

How Shor Code Corrects Bit-Flips

The 9 qubits are grouped into three blocks of three. Each block is a bit-flip code.

- If a single bit-flip error occurs on any qubit (e.g., q_1), it happens within one block.
- The inner code's stabilizers for that block (Z_1Z_2, Z_2Z_3) will detect the error.
- The syndrome identifies the errored qubit.
- We apply another X gate to that qubit to correct it.
- This works for any single bit-flip on any of the 9 qubits.

How Shor Code Corrects Phase-Flips

The outer code is a phase-flip code, where each "qubit" is one of the 3-qubit blocks.

- A single phase-flip error (Z) on one physical qubit is equivalent to a logical phase-flip on its block.

How Shor Code Corrects Phase-Flips

The outer code is a phase-flip code, where each "qubit" is one of the 3-qubit blocks.

- A single phase-flip error (Z) on one physical qubit is equivalent to a logical phase-flip on its block.
- Example: Z_1 on $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$ gives $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$.

How Shor Code Corrects Phase-Flips

The outer code is a phase-flip code, where each "qubit" is one of the 3-qubit blocks.

- A single phase-flip error (Z) on one physical qubit is equivalent to a logical phase-flip on its block.
- Example: Z_1 on $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$ gives $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$.
- The outer code's stabilizers detect which block has the phase error.

How Shor Code Corrects Phase-Flips

The outer code is a phase-flip code, where each "qubit" is one of the 3-qubit blocks.

- A single phase-flip error (Z) on one physical qubit is equivalent to a logical phase-flip on its block.
- Example: Z_1 on $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$ gives $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$.
- The outer code's stabilizers detect which block has the phase error.
- These stabilizers are logical X operators of the inner codes, e.g., $(X_1X_2X_3)(X_4X_5X_6)$.

How Shor Code Corrects Phase-Flips

The outer code is a phase-flip code, where each "qubit" is one of the 3-qubit blocks.

- A single phase-flip error (Z) on one physical qubit is equivalent to a logical phase-flip on its block.
- Example: Z_1 on $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$ gives $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$.
- The outer code's stabilizers detect which block has the phase error.
- These stabilizers are logical X operators of the inner codes, e.g., $(X_1X_2X_3)(X_4X_5X_6)$.
- Once the block is identified, we apply a Z gate to one of its qubits to correct the error.

Shor Code: Y Errors

- The Shor code can correct both bit-flip (X) and phase-flip (Z) errors on any single qubit.

Shor Code: Y Errors

- The Shor code can correct both bit-flip (X) and phase-flip (Z) errors on any single qubit.
- A Y error ($= iXZ$) is a rotated combination of X and Z.

Shor Code: Y Errors

- The Shor code can correct both bit-flip (X) and phase-flip (Z) errors on any single qubit.
- A Y error ($= iXZ$) is a rotated combination of X and Z.
- How can we deal with this i factor?

Shor Code: Y Errors

- The Shor code can correct both bit-flip (X) and phase-flip (Z) errors on any single qubit.
- A Y error ($= iXZ$) is a rotated combination of X and Z.
- How can we deal with this i factor?
- In fact, errors are always a linear combination of the pauli bases.

Shor Code: Y Errors

- The Shor code can correct both bit-flip (X) and phase-flip (Z) errors on any single qubit.
- A Y error ($= iXZ$) is a rotated combination of X and Z.
- How can we deal with this i factor?
- In fact, errors are always a linear combination of the pauli bases.
- How can we deal with a *continuous* range of errors?

Table of Contents

Errors as a Superposition

- Any arbitrary single-qubit error, represented by a matrix E, can be expressed as a linear combination of Pauli matrices:

$$E = c_i I + c_x X + c_y Y + c_z Z$$

Errors as a Superposition

- Any arbitrary single-qubit error, represented by a matrix E, can be expressed as a linear combination of Pauli matrices:

$$E = c_i I + c_x X + c_y Y + c_z Z$$

- What happens when an error E acts on a logical state $|\psi_L\rangle$?

Errors as a Superposition

- Any arbitrary single-qubit error, represented by a matrix E, can be expressed as a linear combination of Pauli matrices:

$$E = c_i I + c_x X + c_y Y + c_z Z$$

- What happens when an error E acts on a logical state $|\psi_L\rangle$?
- The result is a superposition of the original state and the three Pauli errors acting on it:

$$E|\psi_L\rangle = c_i I|\psi_L\rangle + c_x X|\psi_L\rangle + c_y Y|\psi_L\rangle + c_z Z|\psi_L\rangle$$

Error Discretization via Measurement

- When we measure the stabilizers of the code, the state collapses.

Error Discretization via Measurement

- When we measure the stabilizers of the code, the state collapses.
  - Collapses $\implies$ The c goes away.

Error Discretization via Measurement

- When we measure the stabilizers of the code, the state collapses.
  - Collapses $\implies$ The c goes away.
  - Ex: Collapses to X with probability $|c_x|^2$.

Error Discretization via Measurement

- When we measure the stabilizers of the code, the state collapses.
  - Collapses $\implies$ The c goes away.
  - Ex: Collapses to X with probability $|c_x|^2$.
- This has the effect of **discretizing** the error. The state snaps to one of four possibilities:

Error Discretization via Measurement

- When we measure the stabilizers of the code, the state collapses.
  - Collapses $\implies$ The c goes away.
  - Ex: Collapses to X with probability $|c_x|^2$.
- This has the effect of **discretizing** the error. The state snaps to one of four possibilities:
  - No error (I)

Error Discretization via Measurement

- When we measure the stabilizers of the code, the state collapses.
  - Collapses $\implies$ The c goes away.
  - Ex: Collapses to X with probability $|c_x|^2$.
- This has the effect of **discretizing** the error. The state snaps to one of four possibilities:
  - No error (I)
  - An X error

Error Discretization via Measurement

- When we measure the stabilizers of the code, the state collapses.
  - Collapses $\implies$ The c goes away.
  - Ex: Collapses to X with probability $|c_x|^2$.
- This has the effect of **discretizing** the error. The state snaps to one of four possibilities:
  - No error (I)
  - An X error
  - A Z error

Error Discretization via Measurement

- When we measure the stabilizers of the code, the state collapses.
  - Collapses $\implies$ The c goes away.
  - Ex: Collapses to X with probability $|c_x|^2$.
- This has the effect of **discretizing** the error. The state snaps to one of four possibilities:
  - No error (I)
  - An X error
  - A Z error
  - A Y error

Error Discretization via Measurement

- When we measure the stabilizers of the code, the state collapses.
    - Collapses $\implies$ The c goes away.
    - Ex: Collapses to X with probability $|c_x|^2$.
- This has the effect of **discretizing** the error. The state snaps to one of four possibilities:
    - No error (I)
    - An X error
    - A Z error
    - A Y error

Error Discretization via Measurement

- When we measure the stabilizers of the code, the state collapses.
  - Collapses $\implies$ The c goes away.
  - Ex: Collapses to X with probability $|c_x|^2$.
- This has the effect of **discretizing** the error. The state snaps to one of four possibilities:
  - No error (I)
  - An X error
  - A Z error
  - A Y error

Requirements for Error Discretization

- Stabilizers must be able to actually detect the Pauli error basis.
- *Simultaneous* stabilizer measurements require their stabilizers to **commute**.

Summarizing Discretization

- The syndrome measurements tells us *which* discrete Pauli error occurred and *where*.

Summarizing Discretization

- The syndrome measurements tells us *which* discrete Pauli error occurred and *where*.
- Shor stabilizers commute with each other (so that we can measure them simultaneously).

Summarizing Discretization

- The syndrome measurements tells us *which* discrete Pauli error occurred and *where*.
- Shor stabilizers commute with each other (so that we can measure them simultaneously).
  - Ex: Outer stabilizers like: $(X_1X_2X_3)(X_4X_5X_6)$ commute with inner stabilizers Z_1Z_2 and Z_2Z_3.

Summarizing Discretization

- The syndrome measurements tells us *which* discrete Pauli error occurred and *where*.
- Shor stabilizers commute with each other (so that we can measure them simultaneously).
  - Ex: Outer stabilizers like: $(X_1X_2X_3)(X_4X_5X_6)$ commute with inner stabilizers Z_1Z_2 and Z_2Z_3.
- Once the error is identified, we simply apply the same operator again to reverse it (since Pauli operators are their own inverse).

Summarizing Discretization

- The syndrome measurements tells us *which* discrete Pauli error occurred and *where*.
- Shor stabilizers commute with each other (so that we can measure them simultaneously).
  - Ex: Outer stabilizers like: $(X_1X_2X_3)(X_4X_5X_6)$ commute with inner stabilizers Z_1Z_2 and Z_2Z_3.
- Once the error is identified, we simply apply the same operator again to reverse it (since Pauli operators are their own inverse).
- **Conclusion:** By correcting only the discrete Pauli errors, we can correct any arbitrary single-qubit error.

Table of Contents

The Stabilizer Formalism

- A powerful framework for constructing and understanding QEC codes.

The Stabilizer Formalism

- A powerful framework for constructing and understanding QEC codes.
- The bit-flip, phase-flip, and Shor codes are all examples of **stabilizer codes**.

The Stabilizer Formalism

- A powerful framework for constructing and understanding QEC codes.
- The bit-flip, phase-flip, and Shor codes are all examples of **stabilizer codes**.
- A stabilizer code is defined by a set of commuting Pauli operators $\{S_i\}$ called the **stabilizers**.

The Stabilizer Formalism

- A powerful framework for constructing and understanding QEC codes.
- The bit-flip, phase-flip, and Shor codes are all examples of **stabilizer codes**.
- A stabilizer code is defined by a set of commuting Pauli operators $\{S_i\}$ called the **stabilizers**.
- The full set, $\mathcal{S}$, is a commutative group that is *generated* by a smaller set of *independent* stabilizers.

The Stabilizer Formalism

- A powerful framework for constructing and understanding QEC codes.
- The bit-flip, phase-flip, and Shor codes are all examples of **stabilizer codes**.
- A stabilizer code is defined by a set of commuting Pauli operators $\{S_i\}$ called the **stabilizers**.
- The full set, $\mathcal{S}$, is a commutative group that is *generated* by a smaller set of *independent* stabilizers.
- Start with $G = \{g_1, g_2, \ldots, g_n\}$ then:

$$\mathcal{S} = \{S_i | S_i = \prod_{g_i \in A} g_i, A \subset G\}$$

The Stabilizer Formalism

- A powerful framework for constructing and understanding QEC codes.
- The bit-flip, phase-flip, and Shor codes are all examples of **stabilizer codes**.
- A stabilizer code is defined by a set of commuting Pauli operators $\{S_i\}$ called the **stabilizers**.
- The full set, $\mathcal{S}$, is a commutative group that is *generated* by a smaller set of *independent* stabilizers.
- Start with $G = \{g_1, g_2, \ldots, g_n\}$ then:

$$\mathcal{S} = \{S_i | S_i = \prod_{g_i \in A} g_i, A \subset G\}$$

- Caution: We can't include $-I$ in $\mathcal{S}$ (next slide will explain why).

How Stabilizer Codes Work

- The **codespace** is the subspace of states $|\psi\rangle$ that are left unchanged ($+1$ eigenvalue) by all stabilizers:

$$S_i|\psi\rangle = +1|\psi\rangle \quad \text{for all } S_i \in \mathcal{S}$$

How Stabilizer Codes Work

- The **codespace** is the subspace of states $|\psi\rangle$ that are left unchanged ($+1$ eigenvalue) by all stabilizers:

$$S_i|\psi\rangle = +1|\psi\rangle \quad \text{for all } S_i \in \mathcal{S}$$

- Made from Pauli operators so the values that each property can be is in $\{+1, -1\}$.

How Stabilizer Codes Work

- The **codespace** is the subspace of states $|\psi\rangle$ that are left unchanged ($+1$ eigenvalue) by all stabilizers:

$$S_i|\psi\rangle = +1|\psi\rangle \quad \text{for all } S_i \in \mathcal{S}$$

- Made from Pauli operators so the values that each property can be is in $\{+1, -1\}$.
- Each stabilizer generator splits the quantum space in half.

How Stabilizer Codes Work

- The **codespace** is the subspace of states $|\psi\rangle$ that are left unchanged ($+1$ eigenvalue) by all stabilizers:

$$S_i|\psi\rangle = +1|\psi\rangle \quad \text{for all } S_i \in \mathcal{S}$$

- Made from Pauli operators so the values that each property can be is in $\{+1, -1\}$.
- Each stabilizer generator splits the quantum space in half.
  - Trace of Pauli operators is 0. Hint: Trace is the sum of eigenvalues (including repetition).

How Stabilizer Codes Work

- The **codespace** is the subspace of states $|\psi\rangle$ that are left unchanged ($+1$ eigenvalue) by all stabilizers:

$$S_i|\psi\rangle = +1|\psi\rangle \quad \text{for all } S_i \in \mathcal{S}$$

- Made from Pauli operators so the values that each property can be is in $\{+1, -1\}$.
- Each stabilizer generator splits the quantum space in half.
  - Trace of Pauli operators is 0. Hint: Trace is the sum of eigenvalues (including repetition).
- An error E occurs. The state $E|\psi\rangle$ may no longer be a $+1$ eigenstate of some stabilizers.

How Stabilizer Codes Work

- The **codespace** is the subspace of states $|\psi\rangle$ that are left unchanged ($+1$ eigenvalue) by all stabilizers:

$$S_i|\psi\rangle = +1|\psi\rangle \quad \text{for all } S_i \in \mathcal{S}$$

- Made from Pauli operators so the values that each property can be is in $\{+1, -1\}$.
- Each stabilizer generator splits the quantum space in half.
    - Trace of Pauli operators is 0. Hint: Trace is the sum of eigenvalues (including repetition).
- An error E occurs. The state $E|\psi\rangle$ may no longer be a $+1$ eigenstate of some stabilizers.
- We measure all stabilizer generators to get the **error syndromes**.

Table of Contents

The 5-Qubit Code: The Smallest Perfect Code

- The most *efficient* code possible for protecting 1 logical qubit from any single-qubit error.

The 5-Qubit Code: The Smallest Perfect Code

- The most *efficient* code possible for protecting 1 logical qubit from any single-qubit error.
- It encodes 1 logical qubit into 5 physical qubits.

The 5-Qubit Code: The Smallest Perfect Code

- The most *efficient* code possible for protecting 1 logical qubit from any single-qubit error.
- It encodes 1 logical qubit into 5 physical qubits.
- It satisfies the **Quantum Hamming Bound** where n is physical qubits and k is logical qubits:

$$\# \text{ of syndromes} \geq \# \text{ of important configurations}$$
$$2^{n-k} \geq 1 + 3n$$

The 5-Qubit Code: The Smallest Perfect Code

- The most *efficient* code possible for protecting 1 logical qubit from any single-qubit error.
- It encodes 1 logical qubit into 5 physical qubits.
- It satisfies the **Quantum Hamming Bound** where n is physical qubits and k is logical qubits:

$$\# \text{ of syndromes} \geq \# \text{ of important configurations}$$
$$2^{n-k} \geq 1 + 3n$$

- For $n = 5, k = 1$:

$$2^{5-1} = 16 \geq 1 + 3(5) = 16$$

The 5-Qubit Code: The Smallest Perfect Code

- The most *efficient* code possible for protecting 1 logical qubit from any single-qubit error.
- It encodes 1 logical qubit into 5 physical qubits.
- It satisfies the **Quantum Hamming Bound** where n is physical qubits and k is logical qubits:

$$\# \text{ of syndromes} \geq \# \text{ of important configurations}$$
$$2^{n-k} \geq 1 + 3n$$

- For $n = 5, k = 1$:

$$2^{5-1} = 16 \geq 1 + 3(5) = 16$$

- Caveat: This requires a non-degenerate quantum code, meaning that every correctable error has a unique syndrome.

5-Qubit Code: Stabilizers

- We need $n - k = 4$ independent and commuting stabilizers to define a two dimensional codespace for a logical qubit.

5-Qubit Code: Stabilizers

- We need $n - k = 4$ independent and commuting stabilizers to define a two dimensional codespace for a logical qubit.
- The four commuting stabilizer generators are:

$$S_1 = X \otimes Z \otimes Z \otimes X \otimes I$$
$$S_2 = I \otimes X \otimes Z \otimes Z \otimes X$$
$$S_3 = X \otimes I \otimes X \otimes Z \otimes Z$$
$$S_4 = Z \otimes X \otimes I \otimes X \otimes Z$$

5-Qubit Code: Logical Operators

- Logical operators act on the encoded qubit without leaving the codespace.

5-Qubit Code: Logical Operators

- Logical operators act on the encoded qubit without leaving the codespace.
- They must commute with all stabilizers but not be in the stabilizer group themselves.

5-Qubit Code: Logical Operators

- Logical operators act on the encoded qubit without leaving the codespace.
- They must commute with all stabilizers but not be in the stabilizer group themselves.
- They must anti-commute with each other ($\bar{X}\bar{Z} = -\bar{Z}\bar{X}$), forming a valid logical qubit.

5-Qubit Code: Logical Operators

- Logical operators act on the encoded qubit without leaving the codespace.
- They must commute with all stabilizers but not be in the stabilizer group themselves.
- They must anti-commute with each other ($\bar{X}\bar{Z} = -\bar{Z}\bar{X}$), forming a valid logical qubit.
    - This gives us the uncertainty principle from within this protected subspace, and enables us to rebuild the Pauli algebra/bloch sphere.

5-Qubit Code: Logical Operators

- Logical operators act on the encoded qubit without leaving the codespace.
- They must commute with all stabilizers but not be in the stabilizer group themselves.
- They must anti-commute with each other ($\bar{X}\bar{Z} = -\bar{Z}\bar{X}$), forming a valid logical qubit.
    - This gives us the uncertainty principle from within this protected subspace, and enables us to rebuild the Pauli algebra/bloch sphere.
- The logical operators for the 5-qubit code are:

$$\bar{X} = X \otimes X \otimes X \otimes X \otimes X$$
$$\bar{Z} = Z \otimes Z \otimes Z \otimes Z \otimes Z$$

5-Qubit Code Summary

- A code is often described by $[[n, k, d]]$:

5-Qubit Code Summary

- A code is often described by $[[n, k, d]]$:
  - n: number of physical qubits

5-Qubit Code Summary

- A code is often described by $[[n, k, d]]$:
  - n: number of physical qubits
  - k: number of logical qubits

5-Qubit Code Summary

- A code is often described by $[[n, k, d]]$:
  - n: number of physical qubits
  - k: number of logical qubits
  - d: **code distance**

5-Qubit Code Summary

- A code is often described by $[[n, k, d]]$:
  - n: number of physical qubits
  - k: number of logical qubits
  - d: **code distance**
- The code distance d is the minimum weight (single qubit Pauli operations) of a non-trivial logical operator.

5-Qubit Code Summary

- A code is often described by $[[n, k, d]]$:
  - n: number of physical qubits
  - k: number of logical qubits
  - d: **code distance**
- The code distance d is the minimum weight (single qubit Pauli operations) of a non-trivial logical operator.
- For the 5-qubit code, $d = 3$. This means it can correct $t = \lfloor \frac{3-1}{2} \rfloor = 1$ error.

Table of Contents

The Surface Code: Overview

- A QEC code that is a realistic prospect for building a fault-tolerant quantum computer.

- A QEC code that is a realistic prospect for building a fault-tolerant quantum computer.
- **Key Features**:

The Surface Code: Overview

- A QEC code that is a realistic prospect for building a fault-tolerant quantum computer.
- **Key Features**:
  - Only requires nearest-neighbor interactions between qubits on a 2D grid.

The Surface Code: Overview

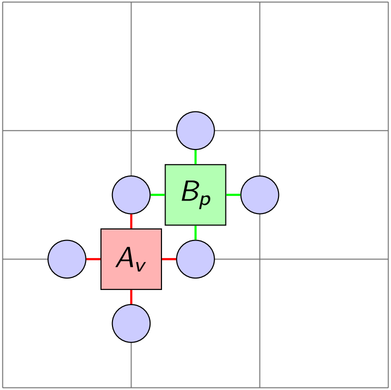
- A QEC code that is a realistic prospect for building a fault-tolerant quantum computer.
- **Key Features**:
  - Only requires nearest-neighbor interactions between qubits on a 2D grid.
  - Has a high **threshold error rate**, tolerating more noise.

The Surface Code: Overview

- A QEC code that is a realistic prospect for building a fault-tolerant quantum computer.
- **Key Features**:
  - Only requires nearest-neighbor interactions between qubits on a 2D grid.
  - Has a high **threshold error rate**, tolerating more noise.
- It is a **topological code**: its properties are protected by the global structure of the system.

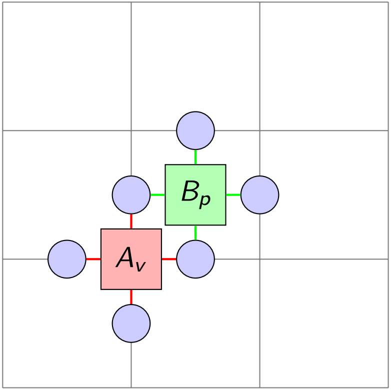
Surface Code: The Lattice

- Data qubits reside on the **edges** of a 2D lattice.



Surface Code: The Lattice

- Data qubits reside on the **edges** of a 2D lattice.
- Stabilizers are defined based on the vertices and faces (plaquettes) of the lattice.



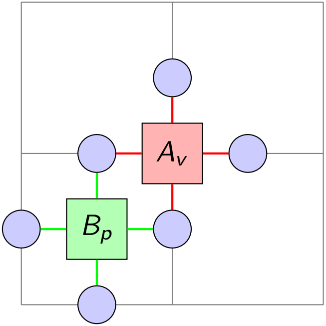
Surface Code: The Stabilizers

- **Star operators (A_v):** Product of X on qubits meeting at a vertex v.

$$A_v = \bigotimes_{i \in \mathrm{star}(v)} X_i$$

- **Plaquette operators (B_p):** Product of Z on qubits bounding a face p.

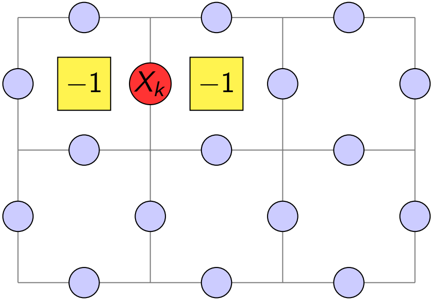
$$B_p = \bigotimes_{i \in \mathrm{boundary}(p)} Z_i$$



Surface Code: Error Detection

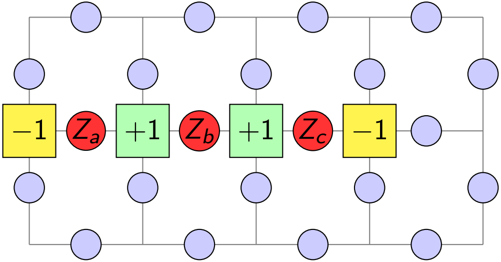
Errors create pairs of **defects** at the ends of error chains.

- X **errors:** Anti-commutes with two adjacent **plaquette** operators, creating a pair of Z-defects.
- Z **errors:** Anti-commutes with two adjacent **star** operators, creating a pair of X-defects.



Error Chains

- A string of errors of the same type creates defects only at the **endpoints** of the chain.
- Example: A chain of three Z errors (Z_a, Z_b, Z_c).

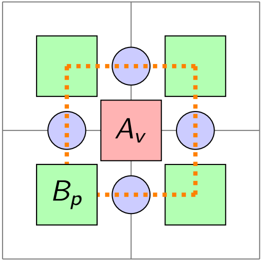


Do X Errors Form Chains?

- Motivation: Want to show[1] that X errors also form chains.

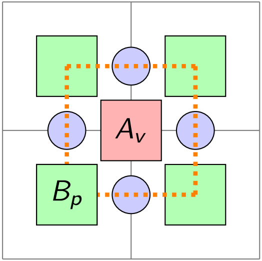
[1] Thank you PanQEC (Eric Huang & Arthur Pesah)!

- The **Dual Lattice** is created by rotating all of the qubit edges by 90 degrees (orange dotted lines).



The Dual Lattice

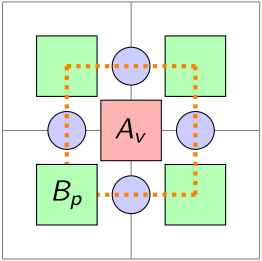
- The **Dual Lattice** is created by rotating all of the qubit edges by 90 degrees (orange dotted lines).



- **Result**: Star Operators $\Leftrightarrow$ Plaquette Operators.

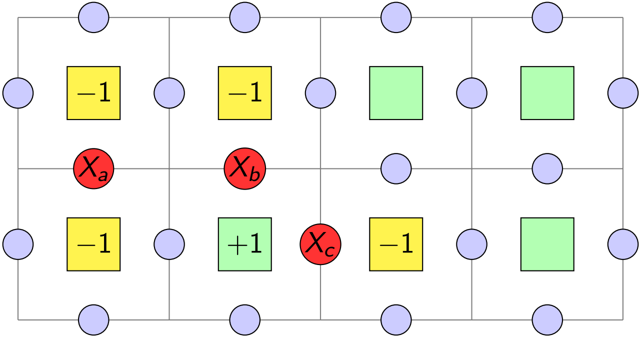
The Dual Lattice

- The **Dual Lattice** is created by rotating all of the qubit edges by 90 degrees (orange dotted lines).

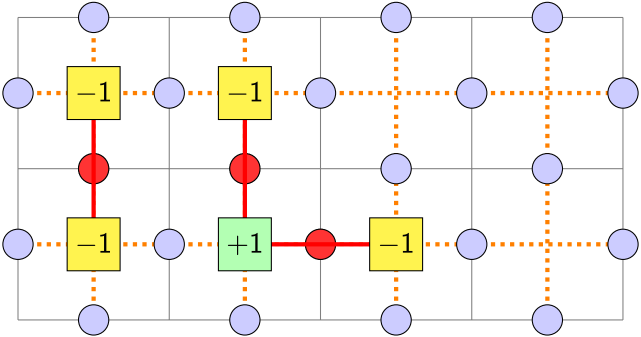


- **Result**: Star Operators $\Leftrightarrow$ Plaquette Operators.
- An X error chain on the dual lattice behaves like a Z error chain on the primal lattice.

X Error Chain on the Primal Lattice



X Error Chain on the Dual Lattice



Correcting Errors

- The syndrome tells us the endpoints of an error chain.

Correcting Errors

- The syndrome tells us the endpoints of an error chain.
- But which path did the error take? There are many possibilities.

Correcting Errors

- The syndrome tells us the endpoints of an error chain.
- But which path did the error take? There are many possibilities.
- Let E_{true} be the actual error. We make a guess, E_{guess}.

Correcting Errors

- The syndrome tells us the endpoints of an error chain.
- But which path did the error take? There are many possibilities.
- Let E_{true} be the actual error. We make a guess, E_{guess}.
- We apply the correction for E_{guess}. The remaining error is $E_{guess}E_{true}$.

Correcting Errors

- The syndrome tells us the endpoints of an error chain.
- But which path did the error take? There are many possibilities.
- Let E_{true} be the actual error. We make a guess, E_{guess}.
- We apply the correction for E_{guess}. The remaining error is $E_{guess}E_{true}$.
- This combined operation is a **loop** in the lattice.

Correcting Errors

- The syndrome tells us the endpoints of an error chain.
- But which path did the error take? There are many possibilities.
- Let E_{true} be the actual error. We make a guess, E_{guess}.
- We apply the correction for E_{guess}. The remaining error is $E_{guess}E_{true}$.
- This combined operation is a **loop** in the lattice.
  - If the loop is a stabilizer, then $E_{guess}E_{true}|\psi_L\rangle = |\psi_L\rangle$.

Correcting Errors

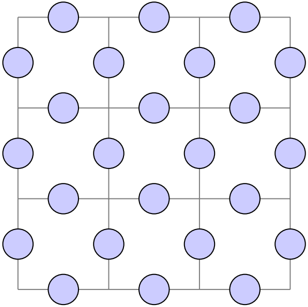
- The syndrome tells us the endpoints of an error chain.
- But which path did the error take? There are many possibilities.
- Let E_{true} be the actual error. We make a guess, E_{guess}.
- We apply the correction for E_{guess}. The remaining error is $E_{guess}E_{true}$.
- This combined operation is a **loop** in the lattice.
    - If the loop is a stabilizer, then $E_{guess}E_{true}|\psi_L\rangle = |\psi_L\rangle$.
    - The correction works!

Logical Operators are Undetectable

- What makes a loop a logical operator vs. a stabilizer?

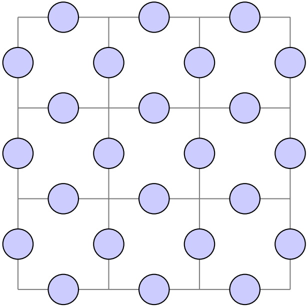
Logical Operators are Undetectable

- What makes a loop a logical operator vs. a stabilizer?
- On a simple plane, all loops are stabilizers.



Logical Operators are Undetectable

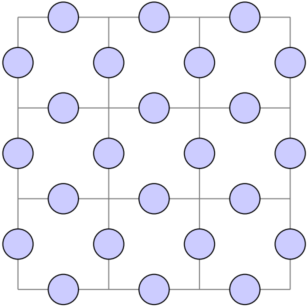
- What makes a loop a logical operator vs. a stabilizer?
- On a simple plane, all loops are stabilizers.



- To get a logical operator we need to create a non-trivial non-detectable operation.

Logical Operators are Undetectable

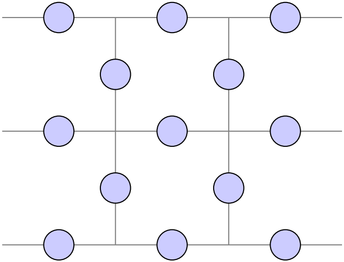
- What makes a loop a logical operator vs. a stabilizer?
- On a simple plane, all loops are stabilizers.



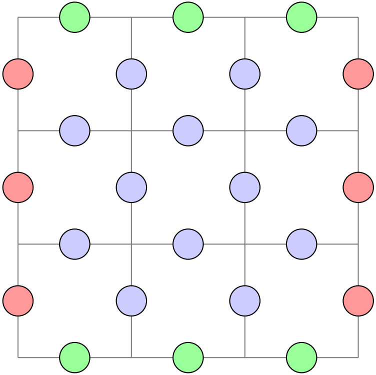
- To get a logical operator we need to create a non-trivial non-detectable operation.
- We can move the endpoints into each other, but this just creates stabilizer given the topology:

Different Topologies

- To have logical operations we need a different topology:



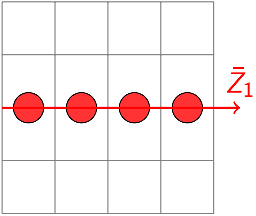
3x3 Planar Code

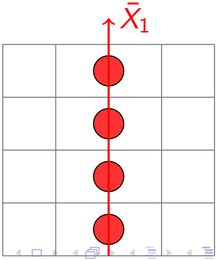


$d = 3$ Toric Code

Logical Operators on the Torus

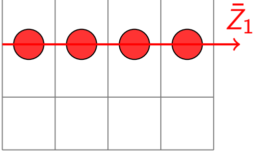
- On a torus, there are non-trivial loops that cannot be shrunk to a point.

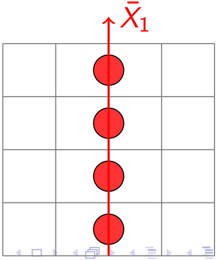




Logical Operators on the Torus

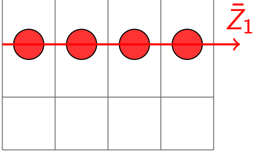
- On a torus, there are non-trivial loops that cannot be shrunk to a point.
- These correspond to logical operators, the Toric Code has four of them.

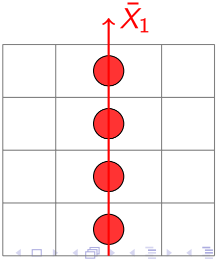




Logical Operators on the Torus

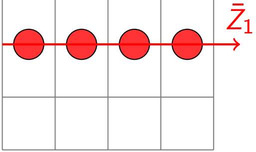
- On a torus, there are non-trivial loops that cannot be shrunk to a point.
- These correspond to logical operators, the Toric Code has four of them.
- **Logical $\overline{Z}$:** A string of Z operators wrapping around the torus (both directions).

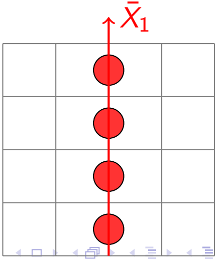




Logical Operators on the Torus

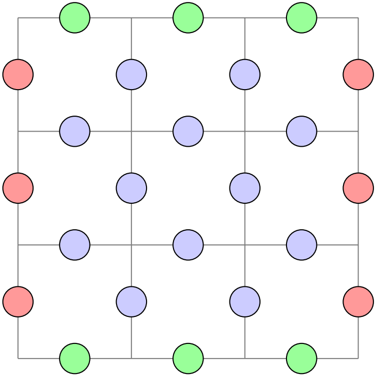
- On a torus, there are non-trivial loops that cannot be shrunk to a point.
- These correspond to logical operators, the Toric Code has four of them.
- **Logical $\overline{Z}$:** A string of Z operators wrapping around the torus (both directions).
- **Logical $\overline{X}$:** A string of X operators wrapping around the torus (both directions).





Toric Code Properties

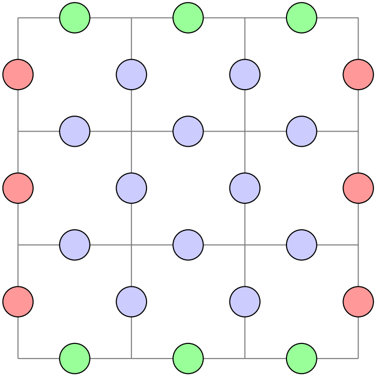
- For an $L \times L$ toric code, we have $2L^2$ data qubits.



3×3 Toric Code

Toric Code Properties

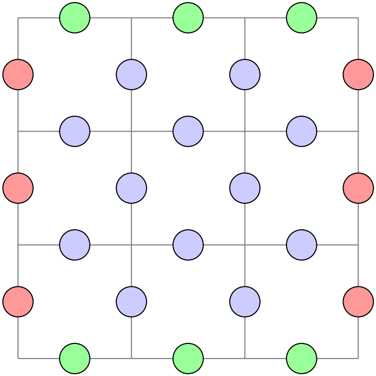
- For an $L \times L$ toric code, we have $2L^2$ data qubits.
- There are L^2 star generators and L^2 plaquette generators.



3×3 Toric Code

Toric Code Properties

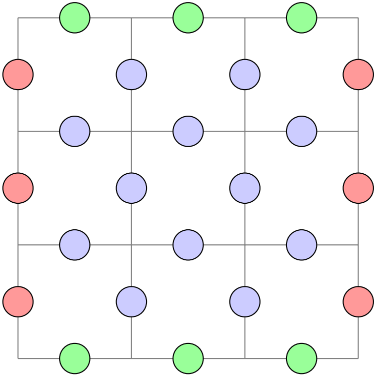
- For an $L \times L$ toric code, we have $2L^2$ data qubits.
- There are L^2 star generators and L^2 plaquette generators.
- Total **independent** stabilizers: $2L^2 - 2$.



3×3 Toric Code

Toric Code Properties

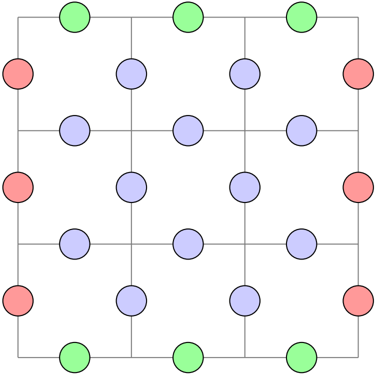
- For an $L \times L$ toric code, we have $2L^2$ data qubits.
- There are L^2 star generators and L^2 plaquette generators.
- Total **independent** stabilizers: $2L^2 - 2$.
- Number of logical qubits $k = n - (\#\text{ of generators}) = 2$.



3×3 Toric Code

Toric Code Properties

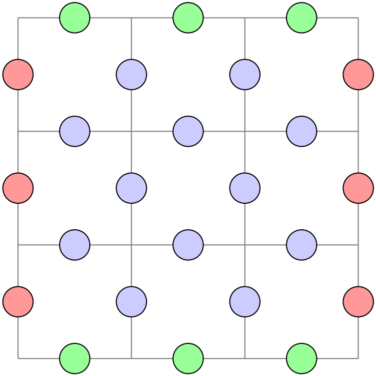
- For an $L \times L$ toric code, we have $2L^2$ data qubits.
- There are L^2 star generators and L^2 plaquette generators.
- Total **independent** stabilizers: $2L^2 - 2$.
- Number of logical qubits $k = n - (\# \text{ of generators}) = 2$.
- The **code distance** d is the length of the shortest logical operator, which is L.



3×3 Toric Code

Toric Code Properties

- For an $L \times L$ toric code, we have $2L^2$ data qubits.
- There are L^2 star generators and L^2 plaquette generators.
- Total **independent** stabilizers: $2L^2 - 2$.
- Number of logical qubits $k = n - (\#\text{ of generators}) = 2$.
- The **code distance** d is the length of the shortest logical operator, which is L.
- So, the toric code is a $[[2L^2, 2, L]]$ code; correcting up to $t = \lfloor (L-1)/2 \rfloor$ errors.



3×3 Toric Code

Toric Code: Decoding

- Decode surface code errors by finding *most likely* error path to explain observed detectors.

Toric Code: Decoding

- Decode surface code errors by finding *most likely* error path to explain observed detectors.
- Many approaches:

Toric Code: Decoding

- Decode surface code errors by finding *most likely* error path to explain observed detectors.
- Many approaches:
  - Minimum Weight Perfect Matching.

Toric Code: Decoding

- Decode surface code errors by finding *most likely* error path to explain observed detectors.
- Many approaches:
  - Minimum Weight Perfect Matching.
  - Belief Propagation + Ordered Statistics.

Toric Code: Decoding

- Decode surface code errors by finding *most likely* error path to explain observed detectors.
- Many approaches:
  - Minimum Weight Perfect Matching.
  - Belief Propagation + Ordered Statistics.
- Tradeoffs between correctness and runtime.

Toric Code: Decoding

- Decode surface code errors by finding *most likely* error path to explain observed detectors.
- Many approaches:
  - Minimum Weight Perfect Matching.
  - Belief Propagation + Ordered Statistics.
- Tradeoffs between correctness and runtime.
- Tesseract Decoder