Master QLMN
Experimental quantum computing

François-Marie Le Régent
francois-marie.le-regent@pasqal.com

Lecture 7 - Foundations of Quantum Error Correction

February 11th 2026

1 Introduction

Over the past weeks, we have covered the two ends of the quantum computing stack. We started with the **physical platforms**: neutral atoms trapped, trapped ions, superconducting circuits, photons, and quantum dots. We then moved to **quantum algorithms**. We studied Shor's factoring algorithm, we explored Grover's search algorithm and its quadratic speedup through amplitude amplification. We also confronted the **resource estimation reality check**: factoring a 2048-bit RSA key requires millions of physical qubits and hours/days of coherent operation, far beyond what any current device can provide.

The gap between the error rates of today's physical qubits and the requirements of fault-tolerant algorithms is the central challenge of the field. This is not a nice-to-have problem: it is *the* bottleneck standing between quantum computing theory and reality.

Today's lecture addresses this gap with **Quantum Error Correction (QEC)**. We will see how to protect fragile quantum information from noise by encoding it redundantly across many physical qubits, creating robust **logical qubits**. We begin with classical error correction to build intuition, then confront the unique challenges of the quantum world (eg the No-Cloning Theorem, measurement back-action), and progressively build up from simple repetition codes, the most classical of QEC codes, to the Shor code, the stabilizer formalism, and finally the surface code. We will also discuss fault-tolerant circuit design, logical gates, and the threshold theorem that guarantees error correction works even with imperfect components.

These notes were written for the **Experimental Quantum Computing and Error Correction** course of M2 Master QLMN, see the course's website for more information. They rely on the standard way of introducting QEC typically found in [1]. Also, a really nice introduction to QEC is accessible on Coursera, entitled Hands-on quantum error correction with Google Quantum AI and presented by Austin Fowler.

For next time 2026-02-18

You are not expected to solve any exercice from today's lecture. Instead, read Acharya et al, *Quantum error correction below the surface code threshold*, 2025 (https://www.nature.com/articles/s41586-024-08449-y) [2]. We'll discuss it during next lecture.

2 Quiz: Exponential Suppression of Errors in QEC

Let's begin by discussing the paper from Google Quantum AI team untitled "Exponential suppression of bit or phase errors with cyclic error correction." *Nature* **595**, 383-387 (2021). DOI: 10.1038/s41586-021-03588-y [3]. Here are 5 questions to check our understanding.

Q1: Code Implementation

Which quantum error correction code structure was primarily implemented to demonstrate the exponential suppression of errors in this experiment?
A. A 2D surface code measuring both X and Z stabilizers simultaneously to correct all errors.
B. A 1D repetition code embedded in a 2D grid, configured separately for either bit-flip or phase-flip protection.

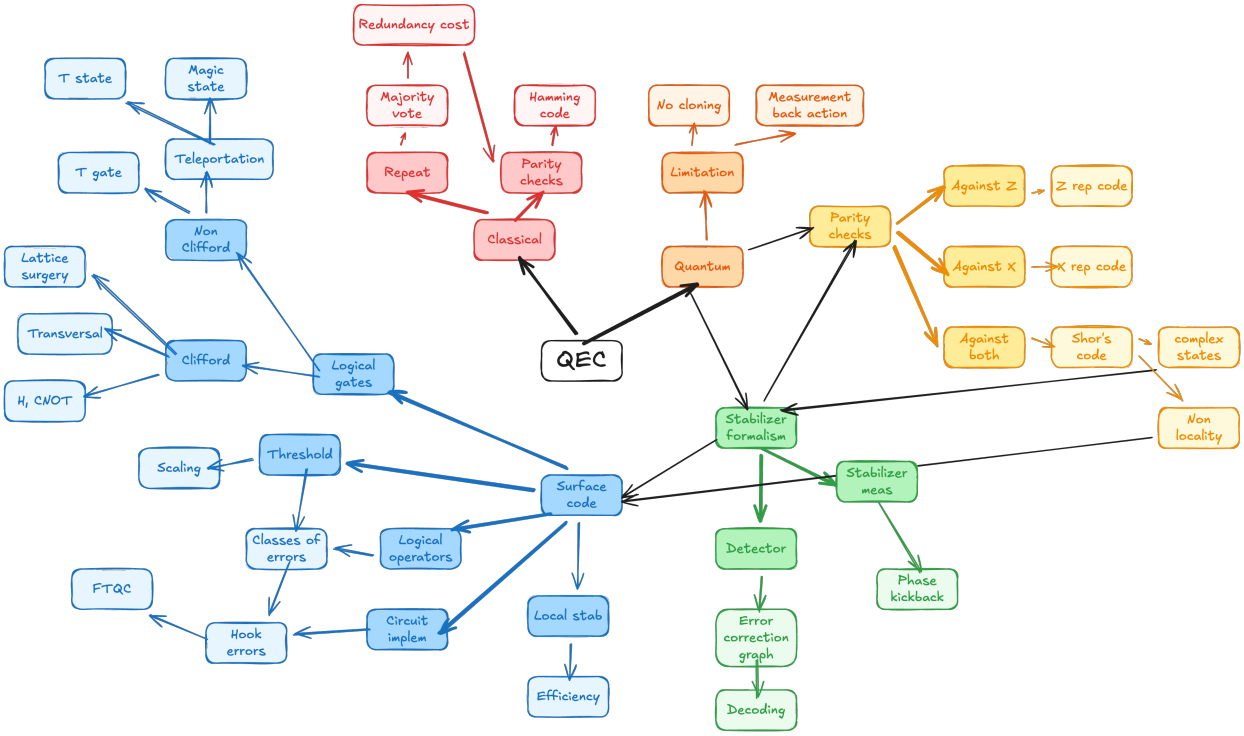

Figure 1: Mind map of the topics that we are going to discuss today.

C. A Shor code encoding 1 logical qubit into 9 physical qubits.
D. A bosonic cat code using superconducting cavities.

Answer

Correct Answer: B
The experiment implemented a 1D repetition code embedded in the 2D Sycamore chip grid. The code was configured separately for bit-flip protection (using Z_iZ_{i+1} stabilizers) or phase-flip protection (using X_iX_{i+1} stabilizers), but not both simultaneously.

Q2: Logical Error Scaling

According to the experimental results, how does the logical error probability per round (ϵ_L) scale with the code distance d?
A. Linearly: $\epsilon_L \propto 1/d$
B. Quadratically: $\epsilon_L \propto 1/d^2$
C. Exponentially: $\epsilon_L \propto \Lambda^{-d/2}$ where Λ is the error suppression factor.
D. The logical error rate remained constant as d increased due to correlated noise.

Answer

Correct Answer: C
The main result of the paper is the exponential suppression of logical errors with code distance. $\Lambda > 1$ means each additional pair of qubits suppresses the error by a constant factor, demonstrating

below-threshold operation.

Q3: Experimental Parameters

Which of the following correctly describe the scale of the repetition code experiments performed? *(Select all that apply)*
A. The code distances ranged from $d = 3$ to $d = 11$.
B. The largest code used 21 superconducting qubits (11 data qubits, 10 measure qubits).
C. The experiment was limited to $d = 3$ due to chip size constraints.
D. The logical error suppression was shown to be stable over 50 rounds of error correction.

Answer

Correct Answers: A, B, D
The experiment demonstrated codes from $d = 3$ to $d = 11$ using up to 21 qubits, and showed stable suppression over 50 rounds. The Sycamore chip had sufficient qubits to go well beyond $d = 3$.

Q4: Error Handling Configuration

How did the experiment address the distinction between bit-flip and phase-flip errors? *(Select all that apply)*
A. A single surface code patch corrected both error types concurrently.
B. The "Phase-Flip Code" used X_iX_{i+1} stabilizers to detect Z errors.
C. The "Bit-Flip Code" used Z_iZ_{i+1} stabilizers to detect X errors.
D. Phase-flip errors were found to be negligible compared to bit-flip errors.

Answer

Correct Answers: B, C
The experiment ran two separate configurations. The bit-flip code measured Z_iZ_{i+1} parity to catch X errors, while the phase-flip code measured X_iX_{i+1} parity to catch Z errors. Each configuration protects against only one error type at a time.

Q5: Physical Error Model

What conclusion did the authors reach regarding the physical error mechanisms in the device?
A. The errors were dominated by non-local cosmic ray events that destroyed the code immediately.
B. The device performance was well-described by a simple uncorrelated depolarizing error model.
C. T_1 relaxation was the sole limiting factor for both bit and phase codes.
D. Correlated errors were significant enough to prevent any error suppression as distance increased.

Answer

Correct Answer: B
The paper found that the device noise was well-modeled by simple uncorrelated Pauli errors, which is why the exponential suppression with distance was observed. Correlated errors were present but subdominant.

3 Classical Error Correction

Before diving into the quantum realm, let's build intuition from the classical world. To understand why we need quantum codes, we first need to appreciate how we protect classical information from noise. This section draws inspiration from the foundational concepts of information theory [4].

3.1 The Noisy Channel

Imagine we want to send a single bit of information, either a 0 or a 1, through a communication line. In a perfect world, what you send is what you get. But physical systems are rarely perfect. We face two primary challenges:

1. **Transmission Noise:** Sending a bit via a noisy line where there is a non-zero probability that the bit flips during transit.
2. **Storage Decay:** Storing a bit in memory where, over time, thermal fluctuations or cosmic rays might cause the bit to flip with a probability p.

The question is: **How can we increase the chance of successful transmission or storage despite this inherent noise?**

3.2 Redundancy and the Majority Vote

The most intuitive solution is **redundancy**. If a single bit is fragile, why not use many? This is the core idea behind the **Classical Repetition Code**. Instead of storing a single bit, we store multiple copies of that same bit and periodically perform a "sanity check" using a **Majority Vote** algorithm.

The Idealized Assumption

For this introductory derivation, we assume that the process of "reading" the bits to take a majority vote is itself perfect and instantaneous. We are only concerned with the errors occurring in the bits themselves during the "waiting" or "traveling" periods.

If we use 3 bits to represent a single logical 0 (encoded as 000), we can tolerate one error. If a single bit flips to 1 (e.g., 010), the majority is still 0, and we can "correct" the state back to 000. However, if two bits flip, the majority vote fails, and we incorrectly conclude the logical bit was a 1. This error suppression mechanism is illustrated in fig. 2.

3.3 Lexicon and Definitions

To *speak* the language of error correction, we must define two critical terms:

1. **Logical States:** These are the "effective" qubit states we care about. Remember that for a physical qubit also, we emphasised that qubit states do not exist, they are *encoded* in physical Hilbert space. For a repetition code, we thus define:
   - $0_L = 000\ldots0$
   - $1_L = 111\ldots1$
2. **Code Distance (d):** This is the minimal number of physical bit flips required to transform a valid 0_L into a valid 1_L.
   - For a 3-bit code ($d = 3$), you need 3 flips to go from 000 to 111.
   - For a 5-bit code ($d = 5$), you need 5 flips.

Intuition: Code Distance and Correction Power

A code with distance d can **correct** any t errors as long as those errors do not flip the **majority**. Mathematically, the code fails only if at least $\frac{d+1}{2}$ errors occur. So a distance-d code corrects up to

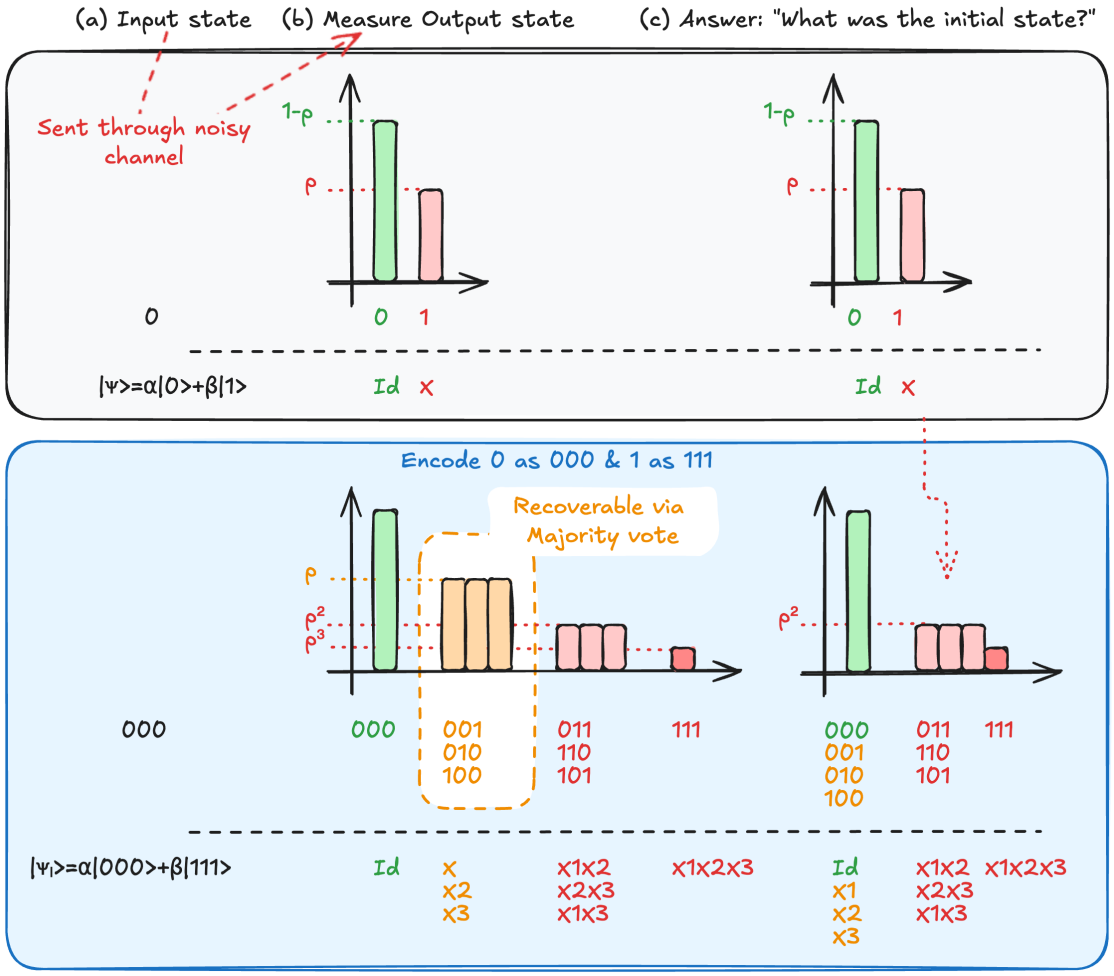

Figure 2: Error suppression in the 3-qubit repetition code: comparison of error probabilities between an unencoded single qubit (top) and a logical qubit encoded as $|0\rangle_L = |000\rangle$ (bottom). The repetition code allows recovery from all single-bit errors via majority vote decoding, suppressing the logical error rate to $O(p^2)$.

$t = \lfloor \frac{d-1}{2} \rfloor$ errors. Furthermore, there is a distinction between **detecting** and **correcting** errors. A code can detect up to $d-1$ errors (since the state is no longer a valid codeword), but it can only safely correct up to $\lfloor \frac{d-1}{2} \rfloor$.

Error Suppression Scaling

If the probability of a single bit flipping is p, then for a $d = 3$ code, the failure probability scales as $O(p^2)$. For a $d = 5$ code, it scales as $O(p^3)$. As long as $p < 0.5$, increasing the redundancy significantly suppresses the logical error rate. This exponential suppression with distance is the whole point of error correction.

Exercise 1: Repetition Code Failure Probability

Consider a classical $d = 5$ repetition code with single-bit error probability $p = 0.01$.

1. Calculate the probability that the majority vote fails (i.e., 3 or more out of 5 bits flip).
2. Compare this to the uncoded error probability $p = 0.01$.
3. What is the ratio of encoded to unencoded error probability? What does this tell you about the value of redundancy?

Answer

1. The majority vote fails when 3, 4, or 5 bits flip. Using the binomial distribution:

$$P_{\text{fail}} = \binom{5}{3} p^3 (1-p)^2 + \binom{5}{4} p^4 (1-p) + \binom{5}{5} p^5$$

$$= 10 \times (0.01)^3 \times (0.99)^2 + 5 \times (0.01)^4 \times 0.99 + (0.01)^5$$

$$\approx 10 \times 10^{-6} \times 0.98 + 5 \times 10^{-8} \times 0.99 + 10^{-10}$$

$$\approx 9.8 \times 10^{-6}$$

2. The unencoded error probability is $p = 0.01 = 10^{-2}$.
3. The ratio is approximately $10^{-6}/10^{-2} = 10^{-4}$. The 5-bit code suppresses the error rate by four orders of magnitude. Redundancy pays off when p is small.

3.4 The Cost of Redundancy

While redundancy grants us protection, it is not free. We must consider the **resource overhead**. If we want to encode k logical bits into n physical bits, we define the **Code Rate** as $R = k/n$.

Metric	Calculation	Value
Data (k)	The original information	4 bits
Encoded (n)	Total physical bits (4×3)	12 bits
Redundancy	$n - k$	8 bits
Rate (R)	k/n	1/3

The physical scaling is linear: if you have 1 Terabyte of data to protect, you would need 3 Terabytes of storage. The efficiency of the encoding becomes a challenge. We have seen that we can achieve high protection by increasing d, but this usually forces the Rate k/n to plummet. The question becomes how can we increase the

Rate (R) or decrease the redundancy required while maintaining the same level of protection? This question leads us directly into the world of Linear Codes and, eventually, Quantum Error Correcting Codes.

3.5 Hamming Code: Parity and the Price of Information

In the 1940s at Bell Labs, Richard Hamming was frustrated by the fragility of electromechanical computers, which frequently crashed due to bit flips. Here is an excerpt of the Hamming code Wikipédia page:

> *Hamming worked on weekends, and grew increasingly frustrated with having to restart his programs from scratch due to detected errors. In a taped interview, Hamming said, "And so I said, 'Damn it, if the machine can detect an error, why can't it locate the position of the error and correct it?' ".*

This led to the development of the **Hamming Code**, a method of checking the parity of data bits to increase the reliability (the rate) of information transfer. The idea behind Hamming codes is to add a specific number of redundant bits, which we often call **ancilla bits** (specifically in the context of our QEC circuits). These ancilla bits allow us to detect that an error has occurred *and* uniquely identify *where* (on which data qubit) it happened.

3.5.1 Geometric Intuition: The Hamming(7,4) Code

Consider $k = 4$ data bits, labeled d_1, d_2, d_3, d_4. To protect these, we add three parity bits: p_1, p_2, p_3. A good way to visualize this is through a Venn diagram where the four data bits sit in the intersections of three overlapping circles, see fig. 3. Each parity bit is responsible for maintaining an even parity within its respective circle:

- p_1 is the parity of d_1, d_2, and d_4.
- p_2 is the parity of d_1, d_3, and d_4.
- p_3 is the parity of d_2, d_3, and d_4.

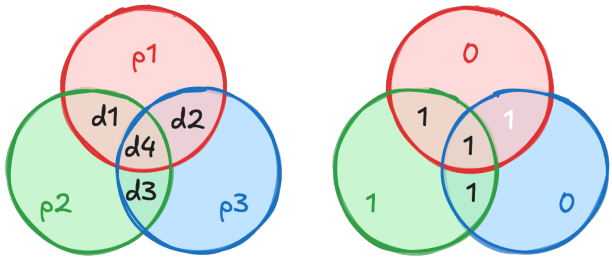

Figure 3: Venn diagram representation of the Hamming(7,4) code. Data bits sit at the intersections of three circles, with parity bits controlling each circle. When data 1011 has an error on d_2, parity checks p_1 and p_3 fail while p_2 remains correct, uniquely identifying d_2.

We can also represent this relationship using a **parity-check matrix** where each row shows which data bit is involved in a given parity bit:

Parity	d_1	d_2	d_3	d_4
p_1	x	x		x
p_2	x		x	x
p_3		x	x	x

Parity Logic

Mathematically, we define the parity such that the sum (modulo 2) of the bits in a circle is zero. For example, $p_1 \oplus d_1 \oplus d_2 \oplus d_4 = 0$.

3.5.2 Error Identification and the Syndrome

What makes this data+parity bits system work is **univocal identification**. If any given data bit flips, it will change the parity of the specific circles it belongs to such that by reading the set of parity violations, also called the **syndrome**, we obtain an "error location address" that tells us where the error occurred.

Suppose our data is 1011 and an error occurs on d_2. We would observe that the new values of the parity checks associated with p_1 and p_3 fail (they were initially 0, but should be 1 after the error), while p_2 remains correct. Looking at our map, d_2 is the only bit shared exclusively by p_1 and p_3, pointing us directly to the source of the error. From a linear algebra perspective, this is handled by the **Code Generator Matrix**.

Looking back at our redundancy table, while a simple repetition code (1 data bit with 2 "copy" ancilla bits) provides high redundancy but low efficiency, the Hamming(7,4) code strikes a balance. With $n = 7$ total bits and $k = 4$ data bits, we have a rate of $R = 4/7 \approx 0.57$ almost two as big as the previous encoding rate. This scaling is efficient: the redundancy grows only as $\log k$ (compared to linear).

Exercise 2: Hamming Code Syndrome Calculation

Consider a Hamming(7,4) code with data bits $d_1d_2d_3d_4 = 1101$ and parity bits computed as described above.

1. Compute the three parity bits p_1, p_2, p_3.
2. Suppose d_3 is flipped. What syndrome do you observe (which parity checks fail)?
3. Verify that the syndrome uniquely identifies d_3 as the corrupted bit.

Answer

1. With $d_1 = 1, d_2 = 1, d_3 = 0, d_4 = 1$:
   - $p_1 = d_1 \oplus d_2 \oplus d_4 = 1 \oplus 1 \oplus 1 = 1$
   - $p_2 = d_1 \oplus d_3 \oplus d_4 = 1 \oplus 0 \oplus 1 = 0$
   - $p_3 = d_2 \oplus d_3 \oplus d_4 = 1 \oplus 0 \oplus 1 = 0$
2. If d_3 flips ($0 \to 1$), recompute parities:
   - p_1: $d_1 \oplus d_2 \oplus d_4 = 1 \oplus 1 \oplus 1 = 1$ (unchanged, p_1 passes)
   - p_2: $d_1 \oplus d_3' \oplus d_4 = 1 \oplus 1 \oplus 1 = 1 \neq 0$ (p_2 fails)
   - p_3: $d_2 \oplus d_3' \oplus d_4 = 1 \oplus 1 \oplus 1 = 1 \neq 0$ (p_3 fails)
   - Syndrome: p_1 OK, p_2 FAIL, p_3 FAIL.
3. Looking at the parity-check matrix, d_3 participates in p_2 and p_3 but not p_1. This matches the observed syndrome, uniquely identifying d_3.

3.6 From Classical to Quantum: The Measurement Back-Action

As we move from the classical regime to the quantum one, the motivation for encoding via parity bits changes fundamentally. In the classical world, adding parity encoding ancilla bits is an **efficiency-driven** choice: we do it to avoid the cost of re-calculating or re-storing data.

In the quantum world, however, encoding data parity in parity storing ancilla qubits becomes a **physical necessity**. This is due to two things: the No-cloning theorem that says that we cannot simply clone aka copy quantum states (said in another way, there is no unitary operation U such that $U|\psi\rangle|0\rangle = |\psi\rangle|\psi\rangle$ for all $|\psi\rangle$) and the "measurement Back action" problem. This last one implies that if we were to measure our data qubits encoding an arbitrary quantum state directly to check for errors, we would collapse their superposition, destroying the very quantum information we are trying to protect. Instead, we must map the parity information onto ancilla qubits and measure *only those*, allowing us to extract the syndrome without "touching" the data itself. This insight is at the heart of the stabilizers, as established in the foundational work of [5] and that we will discuss after looking at simpler examples first.

Exercise 3: Code Rate Comparison

Compare the code rate $R = k/n$ for the following classical codes: 1. A 3-bit repetition code encoding 1 logical bit. 2. A Hamming(7,4) code encoding 4 logical bits. 3. A Hamming(15,11) code encoding 11 logical bits. 4. What general trend do you observe? Why is this relevant for quantum codes?

Answer

1. $R = 1/3 \approx 0.33$
2. $R = 4/7 \approx 0.57$
3. $R = 11/15 \approx 0.73$
4. The rate increases as we encode more data bits with the Hamming family. The number of parity bits grows as $\log_2(n+1)$, so the overhead becomes a smaller fraction. For quantum codes, this trade-off is even more critical because each physical qubit is expensive (in terms of hardware and control), making high-rate codes desirable.

Exercise 4: No-Cloning Theorem

Assume a unitary U exists such that $U|0\rangle|0\rangle = |0\rangle|0\rangle$ and $U|1\rangle|0\rangle = |1\rangle|1\rangle$. Apply U to the state $|+\rangle|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle$ and show that the result is *not* $|+\rangle|+\rangle$.

Answer

By linearity of U:

$$U|+\rangle|0\rangle = \frac{1}{\sqrt{2}}(U|0\rangle|0\rangle + U|1\rangle|0\rangle) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

This is a Bell state (maximally entangled), not the product state $|+\rangle|+\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$. A machine that correctly copies $|0\rangle$ and $|1\rangle$ cannot correctly copy $|+\rangle$. No single unitary can clone all states, so a universal quantum cloning machine is impossible. This is why quantum error correction must use entanglement rather than copying.

4 Quantum Repetition Code

To bypass the measurement problem, we use a **Repetition Code** that follows the key concepts of the classical one but tailored for quantum states. While this code only protects against one type of error at a time, specifically bit-flips (X errors) or phase-flips (Z errors), it introduces the concept of **Quantum Parity Measurements**.

Instead of checking the value of qubit 1 and then qubit 2, we ask a relational question, just like in Hamming codes: *"Are these two qubits the same or different?"* This question has a binary answer: 0 for same like in $|00\rangle$ and $|11\rangle$, 1 for different, as in $|01\rangle$ and $|10\rangle$. This tells us if an error occurred without revealing whether the qubits are in state $|0\rangle$ or $|1\rangle$. Said in another way, if we consider a general 2-qubit state:

$$|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$$

Measuring the qubits Z_1 and Z_2 individually, we would gain too much information. For instance, if we found $Z_1 = +1$ and $Z_2 = -1$, the state would collapse specifically to $|01\rangle$.

4.1 The 3-Qubit Bit-Flip Code

To build our intuition, let's look at the simplest possible instance: protecting against a single bit-flip error (X gate). Again, we borrow the classical idea of redundancy but adapt it to the quantum regime. We define our logical basis states by the mapping:

$$|0\rangle_L = |000\rangle$$
$$|1\rangle_L = |111\rangle$$

An arbitrary physical state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ is now a logical state $|\psi\rangle_L = \alpha|000\rangle + \beta|111\rangle$ spread across three physical qubits. If a bit flip occurs on the first qubit, the state becomes $\alpha|100\rangle + \beta|011\rangle$. Our parity measurements would reveal that the first and second qubits are different, but the second and third are the same. Just like in the classical setting, this "syndrome" tells us exactly which qubit flipped, allowing us to apply a corrective X gate. In the following, we will see what we mean by parity measurements in this context and how to implement it in practice. Before that we quickly mention how to switch from X to Z protection.

Exercise 5: Bit-Flip Code Stabilizer Action

Consider the logical state $|\psi\rangle_L = \alpha|000\rangle + \beta|111\rangle$.

1. Verify that $Z_1Z_2|\psi\rangle_L = +1|\psi\rangle_L$ (i.e., this state is in the $+1$ eigenspace of Z_1Z_2).
2. An X error occurs on qubit 2. Compute $Z_1Z_2(X_2|\psi\rangle_L)$ and $Z_2Z_3(X_2|\psi\rangle_L)$.
3. What is the syndrome? How do you correct the error?

Answer

1. $Z_1Z_2|000\rangle = (+1)(+1)|000\rangle = |000\rangle$ and $Z_1Z_2|111\rangle = (-1)(-1)|111\rangle = |111\rangle$. Therefore $Z_1Z_2|\psi\rangle_L = \alpha|000\rangle + \beta|111\rangle = +1|\psi\rangle_L$.
2. $X_2|\psi\rangle_L = \alpha|010\rangle + \beta|101\rangle$.
   - $Z_1Z_2(\alpha|010\rangle + \beta|101\rangle) = \alpha(+1)(-1)|010\rangle + \beta(-1)(+1)|101\rangle = -(\alpha|010\rangle + \beta|101\rangle)$ So Z_1Z_2 gives eigenvalue -1.
   - $Z_2Z_3(\alpha|010\rangle + \beta|101\rangle) = \alpha(-1)(+1)|010\rangle + \beta(+1)(-1)|101\rangle = -(\alpha|010\rangle + \beta|101\rangle)$. So Z_2Z_3 gives eigenvalue -1.
3. Syndrome: $(-1, -1)$. From the syndrome table, this uniquely identifies an error on qubit 2. We apply X_2 to correct.

4.2 The Phase-Flip Code

With quantum mechanics comes a new type of error: the phase-flip (Z gate), where $Z|1\rangle = -|1\rangle$, on the equator of the Bloch sphere. If we apply the same bit-flip code, it fails completely because a Z error on any qubit results in a logical phase-flip $|\psi\rangle_L = \alpha|000\rangle - \beta|111\rangle$, which our ZZ parity checks cannot detect.

To solve this, we rotate our basis. By working in the dual basis $\{|+\rangle, |-\rangle\}$ (accessible via Hadamard gate), a phase-flip in the computational basis acts as a bit-flip in the X-basis:

$$|+\rangle_L = |+++\rangle$$
$$|-\rangle_L = |---\rangle$$

By measuring the stabilizers X_1X_2 and X_2X_3, we can detect and correct a single Z error just as we did for X errors in the previous example. This symmetry between X and Z errors is discussed in [1].

Intuition: The X-Z Duality

Z errors in the computational basis look exactly like X errors in the Hadamard basis. The Hadamard gate H interchanges the two: $HZH = X$ and $HXH = Z$. If you can correct bit-flips, you can correct phase-flips simply by rotating your perspective by 90 degrees on the Bloch sphere.

Exercise 6: Phase-Flip Code

1. Show that a single Z_1 error on the state $|\psi\rangle_L = \alpha |+++\rangle + \beta |---\rangle$ is undetectable by the Z_1Z_2 stabilizer.
2. Show that the same Z_1 error *is* detected by the X_1X_2 stabilizer. *Hint:* Use the fact that $Z|+\rangle = |-\rangle$ and $Z|-\rangle = |+\rangle$.

Answer

1. $Z_1 |+++\rangle = |-++\rangle$ and $Z_1 |---\rangle = |+--\rangle$. The corrupted state is $\alpha |-++\rangle + \beta |+--\rangle$. In the computational basis, $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Computing Z_1Z_2 on this state: both $|-++\rangle$ and $|+--\rangle$ are superpositions of computational basis states where qubits 1 and 2 can be anything. The Z_1Z_2 check measures computational-basis parity, which does not correlate with phase information. Formally, $[Z_1Z_2, Z_1] = 0$, so Z_1 commutes with Z_1Z_2 and cannot be detected.
2. $X_1X_2 |+++\rangle = |+++\rangle$ (eigenvalue $+1$). $X_1X_2 |-++\rangle = |-++\rangle \cdot (-1)(+1) = -|-++\rangle$ So $X_1X_2(Z_1 |+++\rangle) = X_1X_2 |-++\rangle = (-1)(+1) |-++\rangle = -|-++\rangle$. The eigenvalue flipped to -1, detecting the error. This works because $\{X_1X_2, Z_1\} = 0$ (they anti-commute).

Exercise 7: Joint vs. Individual Measurement

Consider the state $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ (a Bell state).

1. What happens if you measure Z_1 individually? What is the post-measurement state for each outcome?
2. What happens if you measure Z_1Z_2 jointly? What is the post-measurement state?

Answer

1. Measuring Z_1:
   - Outcome $+1$ (prob. $1/2$): state collapses to $|00\rangle$.
   - Outcome -1 (prob. $1/2$): state collapses to $|11\rangle$. In both cases, the superposition is destroyed.
2. Measuring Z_1Z_2:
   - $Z_1Z_2 |00\rangle = (+1)(+1) |00\rangle = +|00\rangle$
   - $Z_1Z_2 |11\rangle = (-1)(-1) |11\rangle = +|11\rangle$ Both terms have eigenvalue $+1$, so the measurement returns $+1$ with certainty. The post-measurement state is still $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, completely unchanged.

4.3 Repetition Code Circuit Implementation

4.3.1 One Parity Measurement

How do we actually measure the parity Z_1Z_2 in practice? We want to check the joint parity of the 0/1 values on the Z-axis. Mathematically, this corresponds to the eigenvalue of the joint operator $Z_1 \otimes Z_2$. To implement parity measurement, we introduce an ancilla qubit initialized in $|0\rangle$. By applying CNOT gates controlled by d_1 and d_2 onto this ancilla, we map the parity Z_1Z_2 onto the ancilla's state. If the data qubits are in the same state (both 0 or both 1), the ancilla remains $|0\rangle$. If they differ, the ancilla flips to $|1\rangle$. This allows us to "monitor" the state without learning the specific values of α and β. The corresponding circuit is shown in fig. 4.

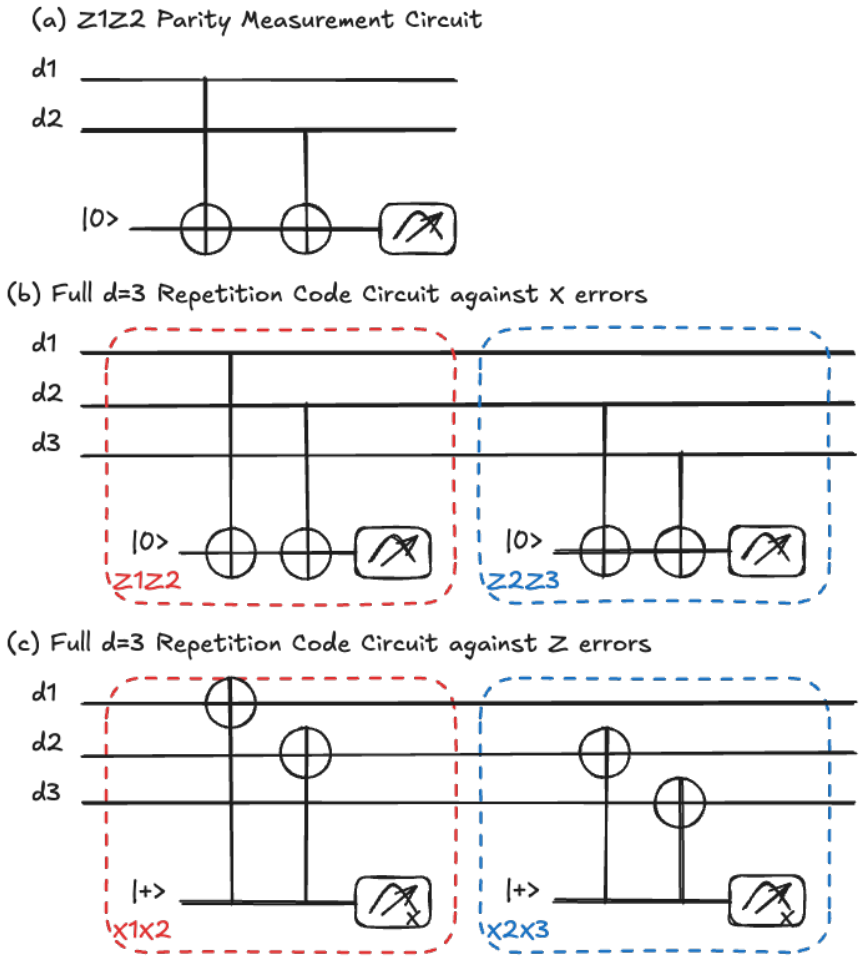

Figure 4: **a** Circuit for a single parity measurement: two CNOT gates controlled by data qubits target a single ancilla, which is then measured. **b** Full syndrome extraction circuit for the 3-qubit bit-flip code with two ancilla qubits measuring Z_1Z_2 and Z_2Z_3. **c** Full syndrome extraction circuit for the 3-qubit phase-flip code with two ancilla qubits measuring X_1X_2 and X_2X_3.

Exercise 8: Parity Measurement Circuit

Consider the state $|\psi\rangle = \alpha\,|00\rangle + \beta\,|11\rangle$ on data qubits, and an ancilla initialized to $|0\rangle$.

1. Write the full 3-qubit state before any gates: $|\psi\rangle\,|0\rangle_a$.
2. Apply $\text{CNOT}_{1\to a}$ (qubit 1 controls, ancilla is target). Write the resulting state.
3. Apply $\text{CNOT}_{2\to a}$. Write the resulting state.

4. What does measuring the ancilla tell you? Does it reveal α or β?

Answer

1. $(\alpha|00\rangle + \beta|11\rangle)|0\rangle = \alpha|000\rangle + \beta|110\rangle$
2. $\text{CNOT}_{1\to a}$: qubit 1 controls the ancilla flip. $\alpha|000\rangle + \beta|111\rangle$ (The first $|0\rangle$ doesn't flip the ancilla; the first $|1\rangle$ in $|110\rangle$ flips $|0\rangle \to |1\rangle$.)
3. $\text{CNOT}_{2\to a}$: qubit 2 controls the ancilla flip. $\alpha|000\rangle + \beta|110\rangle$ (In $|111\rangle$, qubit 2 is $|1\rangle$, so the ancilla flips back: $|1\rangle \to |0\rangle$. Result: $|110\rangle$.) Final state: $(\alpha|00\rangle + \beta|11\rangle)|0\rangle$.
4. The ancilla is $|0\rangle$, meaning the parity is even (qubits agree). The measurement reveals *only* the parity, not α or β. The data state is undisturbed.

4.3.2 Full $d = 3$ Bit Flips Repetition Code

If we want to protect a logical qubit more robustly, we come back to a 3-qubit system capable of monitoring states of the form $\alpha|000\rangle + \beta|111\rangle$. By using two ancilla qubits, we can perform twice the parity measurement describe above, once for each pair of qubits: Z_1Z_2 and Z_2Z_3. The measurement results of these two ancillas provide us with an "error syndrome." This syndrome tells us exactly which qubit (if any) suffered a bit-flip error (X), as can be summarized in the following table:

Error	Z_1Z_2	Z_2Z_3	**Action**
I (None)	$+1$	$+1$	Nothing
X_1	-1	$+1$	Flip d_1
X_2	-1	-1	Flip d_2
X_3	$+1$	-1	Flip d_3

The full syndrome extraction circuit is shown in fig. 4 (b).

4.3.3 Dealing with Phase Errors

Again, we quickly mention that phase-flip errors (Z) are just as deadly as bit-flip error in the quantum world. To monitor states of the form $\alpha|+++\rangle + \beta|---\rangle$, we use the fact that a phase error in the computational basis is just a bit-flip error in the Hadamard (diagonal) basis. By applying a basis change, we replace our $|0\rangle$ and $|1\rangle$ logic with $|+\rangle$ and $|-\rangle$. Consequently, the gates are swapped: X errors become Z errors, and our Z parity checks are replaced by X parity checks: X_1X_2 and X_2X_3, see fig. 4 (c).

4.3.4 Moving Beyond Idealized Assumptions

Here we only gave examples of robustness to two simple error channels: Pauli X and Pauli Z. However, in previous lessons, we mentioned that we do not live in a "Pauli-only" world: experimental reality is much messier.

The Reality of the Lab

In a real device, we face a cocktail of decoherence:

- **Correlated Pauli errors**: X and Z happening simultaneously (effectively a Y error).
- **Leakage**: The qubit leaves the computational subspace ($|0\rangle, |1\rangle$) and enters a higher state $|2\rangle$.
- **Atom Loss**: In neutral atom or trapped ion setups, the physical carrier of the information might literally disappear from the trap.

QEC should in theory take into account all these different noise channels to model as precisely as possible what is happening in our hardware. We are only going to focus on Pauli errors in this lecture, but adding more complex noise channels is the end goal of QEC and an active field of research. To achieve robust error correction, having physical redundancy is not enough. We will rely on two pillars of Fault Tolerance: **Spatial Redundancy** and also **Temporal Redundancy**.

1. **Spatial Redundancy** involves increasing the number of physical qubits as we just saw for the repetition code. As we increase the code distance d, we increase the number of errors required to create a logical one.
2. **Temporal Redundancy** will involve repeating the syndrome measurements roughly d times. This makes the system robust against data errors but also measurement (ancilla) noise, ensuring that a single "bad read" doesn't lead to a false correction.

4.4 Imperfect Parity Measurements

Indeed, in reality, our measurement hardware is not perfect and can affect data qubits but ancilla qubits as well. We must distinguish between a **transient measurement error** (where the ancilla gives the wrong result) and a **persistent data error** (where the qubit actually flipped). If we measure once and see a flip, we don't know if the data is corrupted or if the measurement device just "lied" to us. The solution is to **repeat** the measurement. By measuring the parity measurement circuits shown in fig. 4 multiple times, we can use a majority vote on the syndromes (or more sophisticated decoding) to decide if an error has truly occurred. By looking at the temporal pattern of the syndromes, we can isolate exactly where the fault occurred. This idea is at the heart of **Fault-Tolerant Quantum Computing** [6].

Modeling Errors

- **Absence of Error:** Each measurement consistently reports 0.
- **Data Error:** A flip at time t will cause **all future** parity measurements to report a flip.
- **Measurement Error:** This can be modeled as an X gate applied to the ancilla right before the measurement. It results in a **single flipped result**, while subsequent measurements return to "normal."

Exercise 9: Distinguishing Data Errors from Measurement Errors

An ancilla measuring Z_1Z_2 produces the following sequence of results over 5 rounds: $0, 0, 1, 1, 1$.

1. Is this pattern consistent with a data error? If so, when did it occur?
2. Another ancilla produces the sequence $0, 0, 1, 0, 0$. Is this consistent with a measurement error? When?
3. Why does repeating the measurement help distinguish between these two cases?

Answer

1. Yes. A data error between rounds 2 and 3 would flip the parity permanently. All subsequent measurements (rounds 3, 4, 5) report 1, consistent with a persistent data error occurring between rounds 2 and 3.
2. Yes. A measurement error in round 3 would produce a single incorrect result. The ancilla "lied" once, but the underlying parity didn't change, so rounds 4 and 5 return to 0.
3. A data error produces a *permanent* change in the syndrome (all future rounds flip). A measurement error produces a *transient* blip (only one round is affected). By repeating, we can distinguish the two: a sustained change signals a real data error, while an isolated flip signals a measurement error.

5 Shor Code: Nine Qubits for Arbitrary Error Protection

If we want to protect against *any* single-qubit error (bit-flip, phase-flip, or a combination of both), the repetition code we discuss is not enough. We must use both two approaches together, in a concatenated manner. Peter Shor proposed a 9-qubit code that nests the bit-flip code inside the phase-flip code [5].

5.1 Construction

The logical $|0\rangle_L$ and $|1\rangle_L$ are defined by first encoding the state into a phase-flip triplet:

$$|0\rangle \to |+\rangle\,|+\rangle\,|+\rangle \quad \text{and} \quad |1\rangle \to |-\rangle\,|-\rangle\,|-\rangle$$

Then, we expand each $|+\rangle$ and $|-\rangle$ using the 3-qubit bit-flip code:

$$|+\rangle \to \frac{|000\rangle + |111\rangle}{\sqrt{2}}$$

This results in a 9-qubit state where the logical codewords are:

$$|0\rangle_L = \frac{(|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle)}{2\sqrt{2}}$$

$$|1\rangle_L = \frac{(|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle)}{2\sqrt{2}}$$

Intuition: Hierarchical Protection

You can think of this as a two-level hierarchy. The inner "blocks" (sets of 3 qubits) protect against bit-flips within each block. The outer structure (the relative signs between blocks) protects against phase-flips. Labelling them from 1 to 9, if a bit-flip occurs on qubit 2, the parity checks within the first block (Z_1Z_2 and Z_2Z_3) will flag it. If a phase-flip occurs, it doesn't matter which specific qubit in the block was hit; the phase-flip affects the collective phase of the $|000\rangle \pm |111\rangle$ superposition, which we then detect by comparing the phases between the three blocks.

Digitization of Errors

Even though errors in nature are continuous (like a 1° rotation), the act of measuring the syndrome collapses the error into either "no error" or a "discrete error" (like a full X flip). We effectively digitize the noise. Because any single-qubit error E can be written as a linear combination of I, X, Y, Z, the ability to correct X and Z (and thus $Y = iXZ$) implies we can correct *any* arbitrary rotation of a single qubit.

The encoding circuit for the Shor code is shown in fig. 5.

5.2 Correction Protocol

The Shor code decouples the correction of different error types.

- **Correcting X errors:** We perform check operators Z_1Z_2 and Z_2Z_3 within each of the three blocks. This allows us to identify and flip back any single-qubit bit flip.

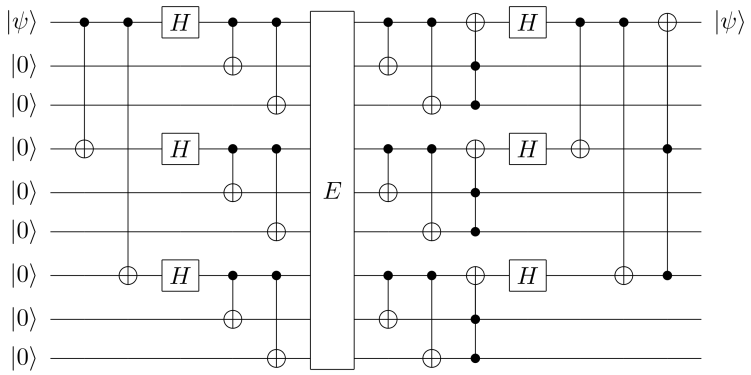

Figure 5: Circuit diagram for the 9-qubit Shor code encoding process.

- **Correcting Z errors:** A phase flip on any qubit in a block flips the sign of that entire block (e.g., changes $|+\rangle$ to $|-\rangle$). To detect this, we measure the "logical" X parity between blocks. Specifically, we compare the phase of Block 1 vs Block 2, and Block 2 vs Block 3.

To measure the phase of a block, we use the operator $X_{\text{block}} = X_1X_2X_3$. Therefore, the stabilizers for phase-flip detection are the products of these operators across blocks, such as $X_1X_2X_3X_4X_5X_6$.

Stabilizer	**Type**	**Function**
Z_1Z_2, Z_2Z_3	Local Z check	Detects X errors in Block 1
Z_4Z_5, Z_5Z_6	Local Z check	Detects X errors in Block 2
Z_7Z_8, Z_8Z_9	Local Z check	Detects X errors in Block 3
$X_1X_2X_3X_4X_5X_6$	Global X	Detects Z between Blocks 1 & 2
$X_4X_5X_6X_7X_8X_9$	Global X	Detects Z between Blocks 2 & 3

Exercise 10: Shor Code and the Y Error

A Y error is equivalent to $Y = iXZ$ (up to a global phase). Suppose a Y error occurs on qubit 5 (the middle qubit of Block 2).

1. What is the effect of X_5 on the encoded state? Which stabilizers detect it?
2. What is the effect of Z_5 on the encoded state? Which stabilizers detect it?
3. Explain why the ability to correct X and Z separately implies the ability to correct Y.

Answer

1. X_5 is a bit-flip in Block 2. The stabilizers Z_4Z_5 and Z_5Z_6 will both return -1 (since qubit 5 is shared by both). Syndrome: $(-1, -1)$ within Block 2, identifying qubit 5.
2. Z_5 is a phase-flip on a qubit in Block 2. It flips the sign of the entire Block 2 superposition ($|000\rangle + |111\rangle \to |000\rangle - |111\rangle$). The inter-block stabilizers $X_1X_2X_3X_4X_5X_6$ and $X_4X_5X_6X_7X_8X_9$ will both return -1.
3. When we measure the syndrome, we first detect and correct the X part (qubit 5 in Block 2), then detect and correct the Z part (Block 2 phase flip). Since $Y = iXZ$, correcting both X and Z on the same qubit also corrects Y. The global phase i is unobservable and doesn't affect the logical state. More formally, the syndrome measurement projects any continuous error onto the discrete Pauli basis, so only I, X, Z, or Y corrections are ever needed.

6 The Stabilizer Formalism

Manually tracking states with 9 qubits starts to be cumbersome, we can easily imagine how complex it becomes when dealing with tens of qubits. We come back to the two examples we already mentioned, repetition codes and Shor's code, but now using the **Stabilizer Formalism** [7]. Instead of describing the state vector, we describe the group of operators $\mathcal{S}$ that leave the state invariant, or said in another way, the operators for which the state is an eigenvector associated with the eigenvalue $+1$. The central idea is to define a quantum state (or a subspace of states) not by writing out its full vector of amplitudes, but by describing the set of operators that leave that state unchanged. This is an efficient way to handle quantum error-correcting codes.

A code is defined by a subgroup of the Pauli group, think $|0\rangle_L$ and $|1\rangle_L$ and their superpositions among all the available states or equivalently the physical representation of logical Pauli operators X_L and Z_L. For a code using n physical qubits to encode k logical qubits, the stabilizer group has $n-k$ independent generators, the operators to check. In terms of protection, we distinguish:

- **Detection:** An error E is detectable if it anti-commutes with at least one stabilizer $S \in \mathcal{S}$.
- **Correction:** We measure the generators of $\mathcal{S}$. The resulting eigenvalues (± 1) form the syndrome that identifies the error.

6.1 The Intuition of Stabilizers

To get a feel for this, let's start by looking at the basic Pauli operators and their eigenstates. We can characterize specific states by the operators they are "stabilized" by:

- The operator $+Z$ stabilizes the state $|0\rangle$, since $Z|0\rangle = +1|0\rangle$.
- The operator $-Z$ stabilizes the state $|1\rangle$, since $-Z|1\rangle = +1|1\rangle$.
- The operator $+X$ stabilizes the state $|+\rangle$, since $X|+\rangle = +1|+\rangle$.
- The operator $-X$ stabilizes the state $|-\rangle$, since $-X|-\rangle = +1|-\rangle$.

In each case, we are identifying an operator and an associated eigenstate corresponding to the eigenvalue $+1$.

Definition: The Codespace

The codespace is the simultaneous $+1$ eigenspace of a set of commuting operators called the **Stabilizers**. If $|\psi\rangle$ is a valid codeword, then $S_i|\psi\rangle = +1|\psi\rangle$ for all S_i in the stabilizer group.

Exercise 11: Stabilizers of Simple States

1. What is the stabilizer of the 2-qubit Bell state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$? *Hint:* Find all 2-qubit Pauli operators P such that $P|\Phi^+\rangle = +|\Phi^+\rangle$.
2. How many independent generators does this stabilizer group have? Is this consistent with the formula $n-k$ for $n=2$ qubits and $k=0$ logical qubits (since $|\Phi^+\rangle$ is a unique state)?

Answer

1. We check all 2-qubit Pauli operators:
   - $Z_1Z_2|\Phi^+\rangle = \frac{1}{\sqrt{2}}(Z_1Z_2|00\rangle + Z_1Z_2|11\rangle) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = +|\Phi^+\rangle$. So Z_1Z_2 is a stabilizer.
   - $X_1X_2|\Phi^+\rangle = \frac{1}{\sqrt{2}}(X_1X_2|00\rangle + X_1X_2|11\rangle) = \frac{1}{\sqrt{2}}(|11\rangle + |00\rangle) = +|\Phi^+\rangle$. So X_1X_2 is a stabilizer. The full stabilizer group is $\{I, X_1X_2, Z_1Z_2, Y_1Y_2\}$ (note $Y_1Y_2 = (X_1X_2)(Z_1Z_2)$ up to sign).
2. Two independent generators: X_1X_2 and Z_1Z_2. This gives $n-k = 2-0 = 2$ generators, consistent with the formula. The Bell state is a unique state (1D subspace $= 2^2/2^2 = 1$), which means $k=0$ encoded qubits.

6.2 The Bit-Flip Repetition Code in Stabilizer Language

We have already encountered the 3-qubit repetition code for bit-flips. In this code, our logical states are $|\psi\rangle_L = \alpha|000\rangle + \beta|111\rangle$. Instead of checking the qubits directly (which would collapse our superposition), we monitor the state using what we called "parity operators" used above, which happen to be the stabilizers. For this specific code, the stabilizers are:

$$S_1 = Z_1Z_2 \quad \text{and} \quad S_2 = Z_2Z_3$$

These operators allow us to monitor the states without destroying the encoded information. By using two stabilizers, we define a 2-dimensional subspace (a logical qubit) within a larger 8-dimensional Hilbert space (2^3 states).

Counting Dimensions

If we have n physical qubits, the total Hilbert space dimension is 2^n. Each independent stabilizer divides the available dimension by 2. For 3 qubits ($2^3 = 8$ dimensions):

- 1 stabilizer (S_1) restricts us to a $2^3/2 = 4$ dimensional subspace.
- 2 stabilizers (S_1, S_2) restrict us to a $2^3/2^2 = 2$ dimensional subspace.

This leaves us with a 2-dimensional subspace, which is exactly what we need to encode a single logical qubit.

Exercise 12: Stabilizer Dimension Counting

1. The Shor code uses $n = 9$ physical qubits to encode $k = 1$ logical qubit. How many independent stabilizer generators does it have?
2. List all stabilizer generators of the Shor code.

Answer

1. $n - k = 9 - 1 = 8$ independent stabilizer generators.
2. The 8 generators are: Z_1Z_2, Z_2Z_3, Z_4Z_5, Z_5Z_6, Z_7Z_8, Z_8Z_9 (6 local checks), $X_1X_2X_3X_4X_5X_6$, $X_4X_5X_6X_7X_8X_9$ (2 global checks).

6.3 Detecting Errors with Stabilizers

Detecting errors boils down to checking if the **stabilizer sign** has flipped from $+1$ to -1. An error that anti-commutes with our stabilizer will flip its eigenvalue. We mention again the concept of error syndrome, now within the scope of the stabilizer formalism. Consider the state $|\psi\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$. If we measure the stabilizer Z_1Z_2, we find $Z_1Z_2|\psi\rangle = 1|\psi\rangle$. However, if a bit-flip error occurs on the second qubit, represented by the operator X_2, our state becomes:

$$|\psi'\rangle = X_2|\psi\rangle = \frac{1}{\sqrt{2}}(|010\rangle + |101\rangle)$$

Now, if we check our stabilizer Z_1Z_2 on this corrupted state:

$$Z_1Z_2(X_2|\psi\rangle) = -X_2(Z_1Z_2|\psi\rangle) = -X_2|\psi\rangle$$

The eigenvalue has changed to -1. This negative sign is our **error syndrome**, telling us precisely that something has gone wrong between qubits 1 and 2.

Intuition: Commutation as Detection

The detection rule is simple: if error E **anti-commutes** with stabilizer S (i.e., $ES = -SE$), then S detects E. If they **commute** ($ES = SE$), then S is blind to E. This is why we need multiple stabilizers: each one is a "detector" sensitive to a different subset of errors.

Exercise 13: Anti-Commutation and Error Detection

For the 3-qubit bit-flip code with stabilizers $S_1 = Z_1Z_2$ and $S_2 = Z_2Z_3$:

1. Compute the commutator $[S_1, X_1]$. Does S_1 detect X_1?
2. Compute the commutator $[S_1, Z_1]$. Does S_1 detect Z_1?
3. Explain why the bit-flip code cannot detect Z errors.

Answer

1. $S_1X_1 = Z_1Z_2X_1 = (Z_1X_1)Z_2 = (-X_1Z_1)Z_2 = -X_1(Z_1Z_2) = -X_1S_1$. So $\{S_1, X_1\} = 0$ (they anti-commute). Yes, S_1 detects X_1.
2. $S_1Z_1 = Z_1Z_2Z_1 = (Z_1Z_1)Z_2 = I \cdot Z_2 = Z_2$. $Z_1S_1 = Z_1Z_1Z_2 = Z_2$. So $S_1Z_1 = Z_1S_1$ (they commute). No, S_1 does not detect Z_1.
3. Both $S_1 = Z_1Z_2$ and $S_2 = Z_2Z_3$ commute with any single Z_i error, since Z_i commutes with all Z operators. Therefore, no Z error produces a syndrome flip, and the bit-flip code is completely blind to phase errors.

6.4 How to Measure a Stabilizer

We already saw the stabilizer measurement circuit without knowing it! For the repetition code, it is exactly the circuits of fig. 4, using controlled-unitary operations.

More generally, if we want to measure any stabilizer A, we use the following circuit construction:

1. Initialize an ancilla in $|0\rangle$ and apply a Hadamard gate to put it in $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.
2. Perform a controlled-A operation where the ancilla is the control and the codespace is the target.
3. Apply another Hadamard to the ancilla and measure it.

This should remind you of phase kickback mechanisms, that we discussed in the context of Grover's algorithm and phase estimation! This construction is depicted in fig. 6.

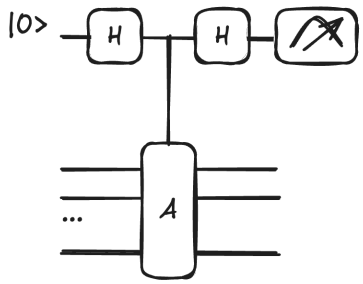

Figure 6: General circuit for measuring a stabilizer A using an ancilla qubit with Hadamard gates and a controlled-A operation.

Derivation

The joint state of the ancilla and the data $|\psi\rangle$ evolves as follows:

$$|0\rangle \otimes |\psi\rangle \xrightarrow{H} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |\psi\rangle \xrightarrow{\text{Ctrl-}A} \frac{1}{\sqrt{2}}(|0\rangle |\psi\rangle + |1\rangle A |\psi\rangle)$$

Applying the second Hadamard yields:

$$\frac{1}{\sqrt{2}}\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} |\psi\rangle + \frac{|0\rangle - |1\rangle}{\sqrt{2}} A |\psi\rangle\right) = |0\rangle \otimes \frac{(I + A) |\psi\rangle}{2} + |1\rangle \otimes \frac{(I - A) |\psi\rangle}{2}$$

Measuring the ancilla in the $|0\rangle$ state effectively applies the projection operator $P_+ = \frac{1}{2}(I + A)$, which projects the data onto the $+1$ eigenspace of A. If the state was already a $+1$ eigenstate, we simply get $|\psi\rangle$ back.

Exercise 14: Stabilizer Measurement Circuit

1. Suppose $|\psi\rangle$ is a $+1$ eigenstate of A: $A |\psi\rangle = + |\psi\rangle$. Trace through the circuit above and show that the ancilla measurement always returns 0.
2. Now suppose an error has occurred and $|\psi'\rangle$ is a -1 eigenstate: $A |\psi'\rangle = - |\psi'\rangle$. Show that the ancilla measurement always returns 1.
3. In the case of the 3-qubit code with $A = Z_1 Z_2$, what specific gates replace the "controlled-A" box?

Answer

1. If $A |\psi\rangle = + |\psi\rangle$, then after the controlled-A: $\frac{1}{\sqrt{2}}(|0\rangle |\psi\rangle + |1\rangle |\psi\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) |\psi\rangle$ After the second Hadamard: $|0\rangle |\psi\rangle$. Measurement always gives 0.
2. If $A |\psi'\rangle = - |\psi'\rangle$, then after the controlled-A: $\frac{1}{\sqrt{2}}(|0\rangle |\psi'\rangle + |1\rangle (- |\psi'\rangle)) = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) |\psi'\rangle$ After the second Hadamard: $|1\rangle |\psi'\rangle$. Measurement always gives 1.
3. The controlled-$Z_1 Z_2$ can be decomposed as two controlled-Z gates (or equivalently, two CNOT gates from data qubits to the ancilla, since CNOT implements a controlled-X which, combined with Hadamards on the ancilla, effectively measures Z). In practice, for Z-type stabilizers, we skip the Hadamards on the ancilla and just use two CNOTs from data qubits 1 and 2 to the ancilla.

6.5 Detectors and Decoding

Now that we have seen the full circuit of stabilizer measurements via repeated syndrome extraction, what should we do with all the ancilla bitstring? How to recover the potentially corrupted encoded data state? Before taking the example of the surface code, we discuss the decoding part by revisiting the repetition code with a focus on **detectors** and **decoding**.

6.5.1 The Logic of Sequential Measurements

In the bit-flip repetition code, we are interested in repeated parity ZZ measurements. As we already mentioned, we don't measure the state of the data qubits directly, as this would collapse the superposition we are trying to protect, but rather the relative parity between neighbors.

When we look at the results of these measurements, we are looking for consistency. If we have a sequence of measurements, any change in the parity result over time or space signals that an error has occurred. We define a **Detector** as a specific set of measurements that, in the absence of any errors, has a deterministic expected parity (usually even).

Detector Event

A **Detector Event** occurs when a detector returns an "unexpected parity." It signals that an error has entered the system. Crucially, a single measurement result in isolation tells us nothing; it is only the relationship between sequential measurements that allows us to track errors.

The intuition is that every sequential pair of measurements forms a detector. It isn't the specific value of the individual measurement that carries the information, but rather the change between them. For instance, if a measurement m_i at time t gives 0 and the measurement at $t+1$ gives 1, we know that an error must have occurred in the interval between those two measurements, because of this parity change. In practice, here is an example a raw ancilla measurement sequence and its associated detection signal:

Ancilla Measurement Sequence	**Detection Signal**
$0 \to 0 \to 1 \to 1 \to 0$	$0 \to 1 \to 0 \to 1$
$0 \to 0 \to 0 \to 1 \to 1$	$0 \to 0 \to 1 \to 0$

The detection signal is essentially the XOR of consecutive bits. A "1" in the detection signal indicates a detector event, a change in parity. This is the first step toward building a "syndrome" that we can use for decoding.

6.5.2 The Error Detection Graph

To make sense of these detector events across many qubits (space) and many rounds of measurement (time), we map the problem onto a **Spacetime Graph**. In this representation, we can visualize how physical errors (the "causes") manifest as detector events (the "symptoms").

- **Nodes** represent potential detector events (the comparison between two measurements).
- **Edges** represent the physical errors that "link" two detector events together.

An edge connects two nodes if a single physical error triggers exactly those two detector events. For example, a bit-flip on a data qubit between two measurement rounds will only trigger the detectors that share that qubit, since after this bit-flip, future detectors will not see any new change. A measurement error on an ancilla, however, might only trigger a "timelike" edge between two consecutive rounds of measurement at the same location. An example of such a detection graphs is shown in fig. 7.

6.5.3 Inferring Errors through Matching

When we actually run an experiment, we obtain a pattern of measurement results. We then identify the nodes where the parity was unexpected. Our goal is to find the most likely set of physical errors (edges) that could have produced this specific pattern of nodes.

This is fundamentally a graph problem. If we see only two detector events, we can "match" them easily with an edge. If we have a complex web of events, we use algorithms like **Minimum Weight Perfect Matching (MWPM)** to infer the most probable error chain. This mapping of quantum errors to graphs is what makes error correction scalable [8].

Exercise 15: Detection Graph Analysis

Consider a distance-3 repetition code (3 data qubits, 2 ancillas) measured for 4 rounds. The detection signal for ancilla 1 (measuring Z_1Z_2) is: $0, 1, 1, 0$. The detection signal for ancilla 2 (measuring Z_2Z_3) is: $0, 1, 1, 0$.

1. At which time step did the detector events occur for each ancilla?
2. Is this pattern consistent with a single data error? If so, on which qubit and at what time?
3. Is this pattern also consistent with two measurement errors? Explain.

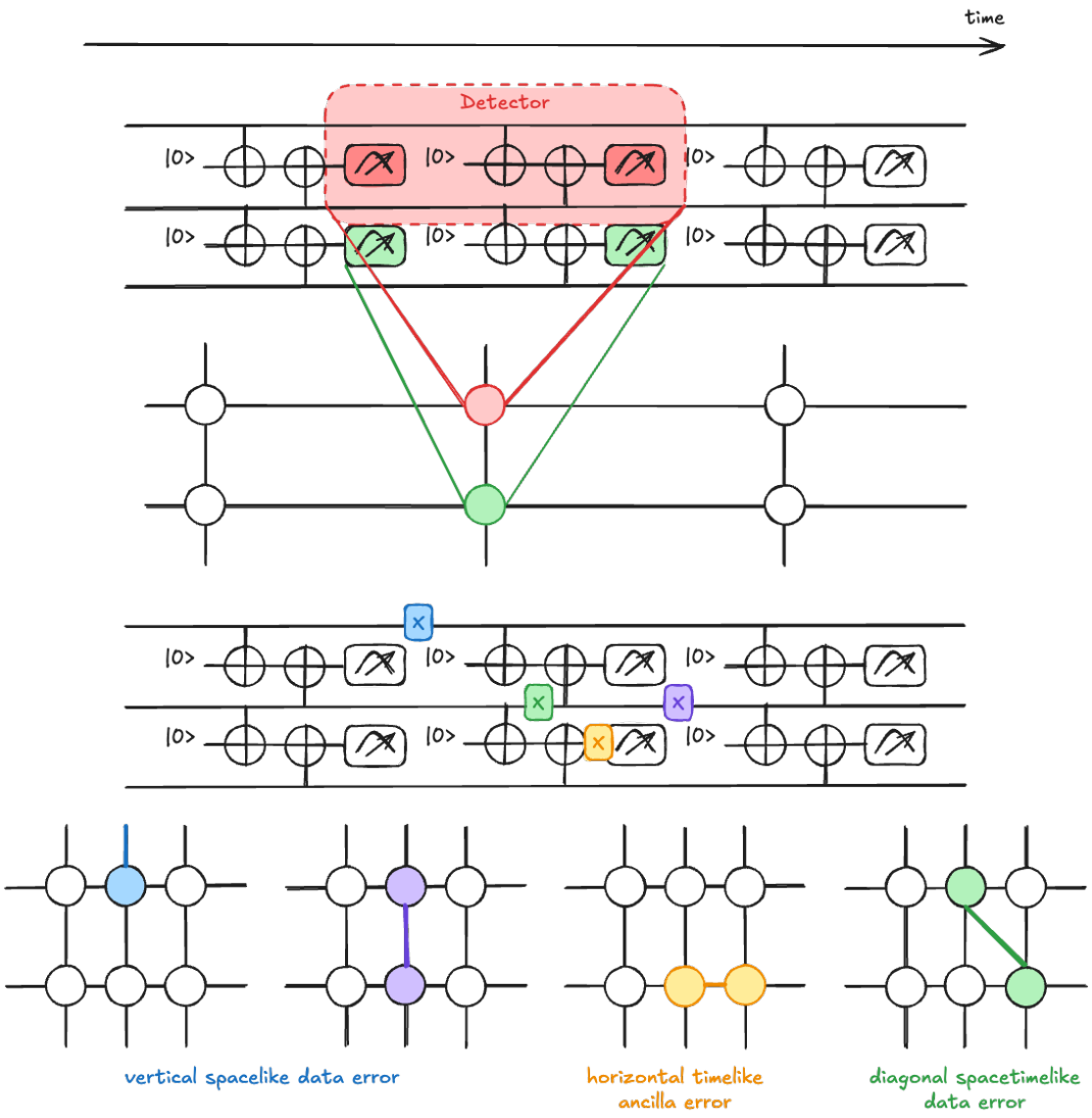

Figure 7: **Spacetime detection graph for the repetition code.** Nodes are detector events, horizontal edges are data errors, vertical edges are measurement errors, diagonal edges are data errors between CNOTs.

Answer

1. Detector events (where the detection signal = 1):
   - Ancilla 1: events at times $t = 2$ and $t = 3$ (XOR flipped between rounds 1-2 and 2-3).
   - Ancilla 2: events at times $t = 2$ and $t = 3$.
2. Detection signal $0, 1, 1, 0$ means: no change between rounds 0-1, change between 1-2, change between 2-3, no change between 3-4. This is consistent with a data error on qubit 2 occurring between rounds 1 and 2 (first detection at round 2), followed by another event between rounds 2 and 3.
3. Yes. Two measurement errors, one at round 2 and one at round 3 for both ancillas, could produce the same pattern. This ambiguity is why MWPM decoding is needed: it finds the most probable explanation given the error rates.

Now that we are familiar with the stabilizer formalism and the decoding problem and having built intuition from the repetition code, we discuss the surface code, probably the most well studied QEC code.

7 Surface Code

7.1 Limitations of the Shor Code

The Shor Code showed we could protect against both X and Z errors by nesting codes. However, the Shor code has a distinct hierarchy: we have local Z checks to catch bit-flips, but the X checks are "global" across blocks.

The disadvantage is the "non-locality" of the X stabilizers. In a physical architecture, requiring a 6-qubit joint measurement ($X_1X_2X_3X_4X_5X_6$) is hardware-intensive and prone to introducing more errors than it fixes.

7.2 The Surface Code

The Surface Code takes a different approach. Instead of mixing local and global checks, we make **everything local**. By arranging qubits on a 2D lattice, we can perform all parity checks using only nearest-neighbor interactions involving 4 data qubits.

Key Concept: Local Parity

In the surface code, we define two types of plaquettes:

1. **Z-plaquettes** (darker regions) which measure the parity of Z operators on 4 surrounding data qubits to detect X errors.
2. **X-plaquettes** (lighter regions) which measure the parity of X operators on 4 surrounding data qubits to detect Z errors.

The logical **codespace** ($\text{Span}\{|0\rangle_L, |1\rangle_L\}$) is defined as the $+1$ simultaneous eigenspace of **all** stabilizers.

The lattice structure is illustrated in fig. 8.

Intuition: Why Locality Matters

Locality means each stabilizer measurement only involves nearest-neighbor qubits on a 2D layout. Long-range interactions are either impossible (superconducting chip) or error-prone (atom movement) on most platforms, so this matters a lot in practice. The surface code gives the same protection as the Shor code but with only local operations, which is why it is the leading candidate for fault-tolerant quantum computing as the numerous experimental implementations show.

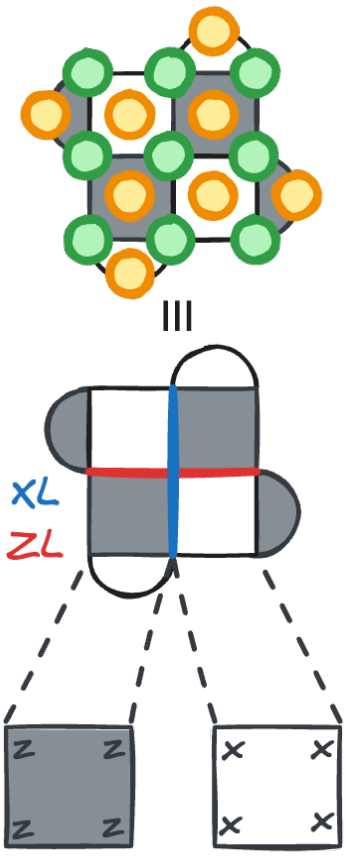

Figure 8: **The surface code lattice**: data qubits (green) sit on vertices, ancilla qubits (orange) are in the centers of Z (dark) and X (light) tiles of the surface. Boundary-to-boundary product of physical operators define the logical operators Z_L (red) and $|X\rangle_L$ blue.

Surface code features

The 2D layout and the small weight of the stabilizers are the reasons why the surface code has been well studied. Another reason that we will discuss soon is its high accuracy threshold: we can tolerate a lot of physical error probability per physical gate before the QEC protection stops working.

7.3 Constructing the Logical State

How to write the $|0\rangle_L$ of this code? We will see that it is not as straightforward as the $|0\rangle_L = |000\rangle$ of the repetition code. To construct the logical $|0\rangle_L$, we start from a "vacuum" of physical $|0\rangle$ states and "project" them consecutively into the stabilizer subspace. If we start from $|0\dots0\rangle$ and apply a stabilizer projector $P = \frac{1}{2}(\mathbb{I} + X_a)$ for each X-type stabilizer, we create the necessary entanglement. Each projector creates a superposition:

$$|0\dots0\rangle + (-1)^a |0\dots1111\dots0\rangle$$

where a represents the measurement outcome.

Let's verify this

$SP|\psi\rangle = S\left(\frac{1}{2}(\mathbb{I} + X_a)\right)|\psi\rangle = \frac{1}{2}(S + S^2)|\psi\rangle = \frac{1}{2}(S + \mathbb{I}) = P|\psi\rangle$

[!Note] What happens if we measured -1?

To return to the $+1$ eigenspace, you would apply a Pauli operator that **anticommutes** with Xa.

- *Example:* If $S_a = X_1X_2X_3X_4$ and you measure -1
- You could apply Z_1.
- Since Z_1 anticommutes with S_a, it flips the eigenvalue from -1 back to $+1$.

If the error was actually on qubit 2 (and you fixed qubit 1), you have now created a logical error (a chain of errors). This is why we wait for the **Decoder** to tell us *which* correction is the smartest one to apply. In practice, we update our "mental model" of the state. We say: "The current state is the *correct* logical state, but with a known Pauli error E sitting on top of it."

As we apply successive stabilizer projectors, the state branches further (twice for each stabilizer), spreading entanglement across the surface. The resulting state is a highly entangled superposition:

$$\begin{aligned}
|0_L\rangle \quad &= |000000000\rangle + (-1)^d|000000110\rangle \\
&+(-1)^a|011000000\rangle + (-1)^{a+d}|011000110\rangle \\
&+(-1)^b|110110000\rangle + (-1)^{b+d}|110110110\rangle \\
&+(-1)^{a+b}|101110000\rangle + (-1)^{a+b+d}|101110110\rangle \\
&+(-1)^c|000011011\rangle + (-1)^{c+d}|000011101\rangle \\
&+(-1)^{a+c}|011011011\rangle + (-1)^{a+c+d}|011011101\rangle \\
&+(-1)^{b+c}|110101011\rangle + (-1)^{b+c+d}|110101101\rangle \\
&+(-1)^{a+b+c}|101101011\rangle + (-1)^{a+b+c+d}|101101101\rangle
\end{aligned}$$

Where $a, \dots, d$ represent the values of each X plaquette measurement.

7.4 The Efficiency of the Stabilizer Formalism

When we look at a lattice of N plaquettes, describing the state of the system by tracking every individual term in the Hilbert space becomes quickly unreasonable, as we are dealing with 2^N terms. The stabilizer formalism avoids this entirely. Instead of an exponential description of the state, we rely on a linear description via the

stabilizers. This is what makes such codes tractable despite the exponential size of the many-body Hilbert space.

7.5 Defining the Logical Operators

To actually use our code for computation, we need to define our logical space. Since we are dealing with a stabilizer code, we will not define the code from its logical states $|0\rangle_L$ and $|1\rangle_L$ (we tried) but from the logical qubit Pauli operators. Specifically, we want to find a logical X_L operator. From a physical perspective, X_L is a string of physical X operators stretching across the lattice, see fig. 8. It cannot be just any string; it must commute with all the stabilizers in our stabilizer group $\mathcal{S}$.

Why Commutation Matters

Stabilizers are "blind" to the full codespace. If a stabilizer measurement could distinguish between states within the codespace, it would necessarily collapse the logical state. Therefore, any logical operator, which by definition moves us from one point in the codespace to another, must commute with **all** stabilizers to ensure it doesn't leave the "protected" subspace or trigger an error syndrome.

Any chain of physical X operators connecting the top boundary to the bottom boundary is a valid logical X_L, provided it commutes with the stabilizers. These different "versions" of X_L are actually equivalent up to a stabilizer (they are in the same *homology* class).

7.6 Logical Z and the Geometry of Errors

Similarly, we define the logical Z_L as a string of physical Z operators connecting the left boundary to the right boundary. For example, in a small lattice:

$$Z_L |1\rangle_L = Z_3 Z_4 Z_5 |1\rangle_L$$

Geometric Intuition

Just as physical X and Z anti-commute when they act on the same qubit, logical X_L and Z_L anti-commute because their paths on the lattice cross an odd number of times. This topological "intersection number" is the deep reason why the code works as a single logical unit. The code distance d equals the minimum length of such a logical operator string, which is why a larger lattice provides more protection.

In this picture, quantum information is not stored at specific locations but in the global, non-local properties of the entire lattice [9].

Exercise 16: Surface Code Logical Operators

Consider a distance-3 surface code on a 3×3 lattice of data qubits (with appropriate boundary conditions). 1. How many data qubits, X-stabilizers, and Z-stabilizers does the code have? 2. Verify the $[[n, k, d]]$ parameters using $k = n -$ (number of stabilizers). 3. Give an example of a weight-3 logical X_L operator (a string of 3 X operators crossing the lattice). 4. Why can't a weight-2 string of X operators be a logical operator?

Answer

1. A distance-3 surface code has $n = d^2 = 9$ data qubits (on a rotated lattice), 4 X-stabilizers, and 4 Z-stabilizers, for a total of 8 stabilizers.
2. $k = n - (\text{stabilizers}) = 9 - 8 = 1$ logical qubit. $d = 3$. So the code is $[[9, 1, 3]]$.
3. A horizontal string of $X_1X_2X_3$ along one column of the lattice (connecting top boundary to bottom boundary if X_L runs vertically in your convention).
4. A weight-2 string would not span the full lattice. It would either be equivalent to a stabilizer (if it forms a closed loop) or anti-commute with some stabilizer (if it's a partial string). Neither case gives a valid logical operator.

7.7 The Threshold Theorem and Scaling

We briefly mentioned the high accuracy threshold of the surface code as being one of the nice features of this code. Now it's time to be more precise on this specific point.

7.7.1 Classes of Errors

First, when we analyze how errors propagate and are caught in a QEC code like the surface code, we saw that we categorize them based on their orientation in the (2+1)D spacetime volume (the "+1" standing for time), just as we did above in the Decoding part:

- **Horizontal matching** corresponds to **time-like errors**: syndrome measurement failures. A flip in the measurement outcome appears as two syndrome changes separated in time.
- **Vertical matching** represents **space-like errors**: the "classic" data qubit errors (Pauli X or Z) that occur on the physical qubits between measurement cycles.
- **Diagonal matching** represents **spacetime errors**: often arising from errors during the entangling gates (like a CNOT) of the stabilizer measurement circuit itself. A single failure in a CNOT can propagate to both the data qubit and the ancilla, creating a correlated error that is diagonal in our decoding graph.

7.7.2 The Threshold

To visualize if our error correction is actually helping, we use a **Threshold Plot**. This plot compares the physical error rate p against the logical error rate P_L. As we increase the code distance d, we see a distinct behavior change at a specific point called the **threshold** p_{th}. We can model the logical error rate as:

$$P_L \approx C \cdot \left(\frac{p}{p_{\text{th}}}\right)^{\frac{d+1}{2}}$$

- **When $p < p_{\text{th}}$**: We are in the "protection" regime. **More is better.** Increasing the distance d suppresses the logical error rate **exponentially**.
- **When $p > p_{\text{th}}$**: We are in the "noise-dominated" regime. **More is worse.** Adding more qubits just adds more opportunities for errors to occur, and the complex decoding can no longer keep up. The logical error rate actually increases with d, eventually saturating at $P_L = 50\%$, which represents total loss of information (random noise).

This behavior is shown in the threshold plot fig. 9.

Key Takeaway

The abscisse of the "crossing point" on the graph is the threshold p_{th}. For the surface code under circuit-level noise, this is often cited around 1% [8]. But be aware that this value varies significantly depending on the noise model of the gates, the circuit implementation and the decoder used. Current

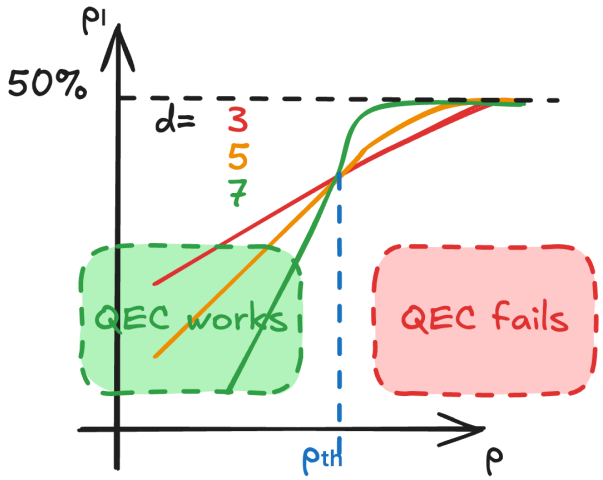

Figure 9: Threshold plot: logical error rate P_L vs physical error rate p for different code distances $d = 3$ (red), 5 (orange) and 7 (green). The curves cross at the threshold p_{th}, (dashed blue line).

superconducting and neutral atom platforms are operating near or just below this threshold.

Exercise 17: Threshold Calculation

Using the approximate formula $P_L \approx 0.1 \cdot (p/p_{\text{th}})^{(d+1)/2}$ with $p_{\text{th}} = 1\%$:

1. For $p = 0.1\%$ and $d = 3$, compute P_L.
2. For $p = 0.1\%$ and $d = 5$, compute P_L.
3. For $p = 0.1\%$ and $d = 7$, compute P_L.
4. By what factor does P_L decrease when going from $d = 3$ to $d = 5$? What about $d = 5$ to $d = 7$?

Answer

With $p/p_{\text{th}} = 0.001/0.01 = 0.1$:

1. $P_L = 0.1 \cdot (0.1)^{(3+1)/2} = 0.1 \cdot (0.1)^2 = 0.1 \cdot 0.01 = 10^{-3}$
2. $P_L = 0.1 \cdot (0.1)^{(5+1)/2} = 0.1 \cdot (0.1)^3 = 0.1 \cdot 10^{-3} = 10^{-4}$
3. $P_L = 0.1 \cdot (0.1)^{(7+1)/2} = 0.1 \cdot (0.1)^4 = 0.1 \cdot 10^{-4} = 10^{-5}$
4. From $d = 3$ to $d = 5$: factor of $10^{-3}/10^{-4} = 10$ improvement. From $d = 5$ to $d = 7$: factor of $10^{-4}/10^{-5} = 10$ improvement. Each increment of 2 in distance gives another order of magnitude in error suppression (when $p/p_{\text{th}} = 0.1$).

7.8 Circuit Implementation and Fault-Tolerant Design

When we move from the theoretical framework of the surface code to its physical implementation, we must consider the specific gates that realize our stabilizers. A Z-plaquette corresponds to a $ZZZZ$ operator acting on four data qubits, which detects X errors. An X-stabilizer corresponds to an $XXXX$ operator designed to detect Z errors.

To measure these stabilizers without destroying the quantum information in the data qubits, we introduce an **ancilla qubit**, just like for the repetition code. The circuit involves a sequence of four CNOT gates (compared to 2 for the repetition code) between the ancilla and the data qubits. But we cannot apply these

gates in any arbitrary order. The sequence of CNOTs matters for **fault tolerance** because of how errors propagate through the circuit.

7.8.1 The Intuition of Error Propagation

Again, our goal is to detect errors in the data by periodically checking the stabilizers via the ancilla. But the ancilla itself is a physical qubit prone to noise. If a single error occurs on the ancilla during the measurement sequence and propagates to multiple data qubits, we risk creating a high-weight error from a single fault.

Consider the "Hook error." If a single X error occurs on the ancilla qubit halfway through the measurement of a Z-stabilizer, it can propagate to the remaining data qubits in the sequence. In a four-qubit plaquette, an error after the second CNOT could result in a ZZ error on the data.

Hook Errors

A single fault on an ancilla that propagates to two or more data qubits is often called a **hook error**. In the surface code, if these errors align poorly, they can effectively reduce the code distance.

7.8.2 Gate Ordering

The danger arises when propagated errors align with our logical operators. For a distance-3 surface code, one hook error could lead to 2 physical data errors, hence a logical one. On a distance-5 code, if two hook errors occur due to unlucky CNOT ordering, they could lead to 4 data errors that form a chain across the lattice, potentially also resulting in a **logical error**.

To prevent this, we follow a specific design rule: the last two qubits touched by a Z-stabilizer measurement must propagate errors **perpendicular** to the logical Z_L operator. By orienting the "stretch" of the hook error so it doesn't bridge the gap between boundaries, we ensure that the error weight remains manageable (aka does not increase) and doesn't lead to a premature logical failure.

One solution is to use a specific CNOT ordering, often a "Z-shape" or "N-shape" on qubits in the plaquette. Specific orderings ensure that the "last 2 qubits touched" rule is preserved, as shown in fig. 10.

Note

This specific geometric solution for gate ordering is not unique, but it is mandatory for maintaining the threshold of the surface code. The specific mapping differs for architectures like Neutral Atoms due to different connectivity and constraints.

Exercise 18: Hook Error Analysis

Consider a X-plaquette measuring $X_1X_2X_3X_4$ with the CNOT sequence: ancilla controls qubits in a circle order.

1. If an X error occurs on the ancilla between the 2nd and 3rd CNOT, which data qubits are affected? What is the resulting data error?
2. If instead the CNOT order is 1, 3, 2, 4, and the same X error occurs between the 2nd and 3rd CNOT, which data qubits are affected?
3. Explain why the second ordering is better for fault tolerance in the surface code.

Answer

1. With ordering 1, 2, 3, 4: the X error after the 2nd CNOT propagates via the 3rd and 4th CNOTs. Each CNOT (ancilla as control for Z measurement) propagates X errors from ancilla to data. The data qubits 3 and 4 receive Z errors (via the CNOT propagation rule). Resulting

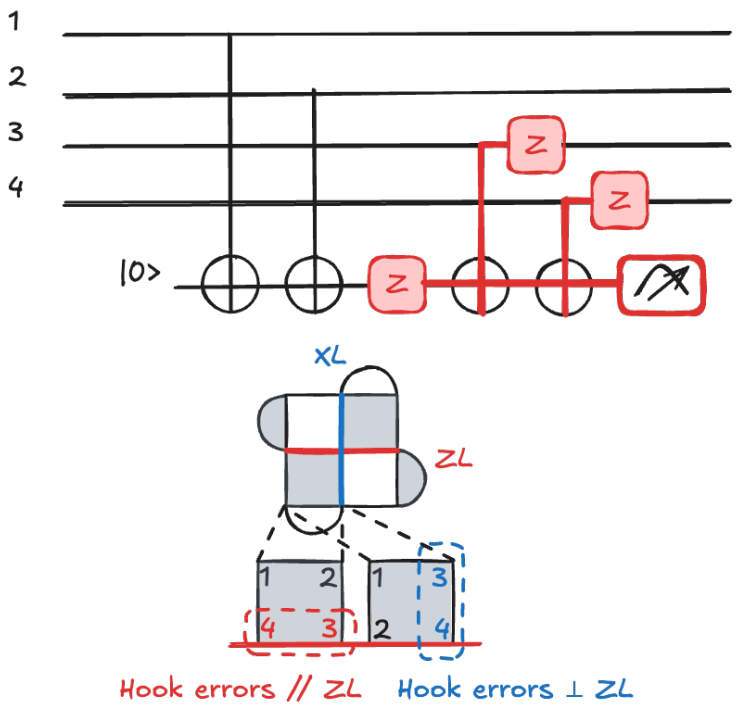

Figure 10: Comparison of CNOT ordering for a Z stabilizer to ensure fault tolerance.

error: Z_3Z_4 on the data, a weight-2 error.

2. With ordering 1, 3, 2, 4: after the 2nd CNOT (which targets qubit 3), the X error propagates via the 3rd CNOT (targeting qubit 2) and 4th CNOT (targeting qubit 4). Resulting error: Z_2Z_4 on the data.
3. In the surface code, qubits 3 and 4 in ordering (a) are adjacent and their Z_3Z_4 error aligns with the logical Z_L string direction, potentially bridging boundaries. In ordering (b), Z_2Z_4 are not adjacent along the logical string direction; they stretch perpendicular to Z_L. A perpendicular error is less dangerous because it doesn't contribute to shortening the effective code distance along the logical operator direction.

7.8.3 The Surface Code Detection Graph

To decode errors on the Surface code, we move from a 2D physical layout to a 3D **Detection Graph**. While a 1D repetition code allows us to visualize errors on a line, the surface code requires a 2D plane of qubits. When we add the dimension of time (the repeated measurement rounds), we get a 3D spacetime volume.

Nothing differs from the repetition code: in this graph, "nodes" still represent stabilizer measurement events, and "edges" represent potential errors (either physical qubit flips or measurement bit-flips). If a stabilizer changes its value between round t and $t+1$, it "triggers," creating a defect in the graph. The job of the decoder is then to pair these defects in a way that most likely explains the underlying physical noise.

7.9 Implementing Logical Gates

7.9.1 Logical Initialization and Measurement

We briefly mention two important logical operations: initialization and measurement. We begin with **logical initialization**, where we prepare the code in a specific state. For the surface code, we typically initialize the

logical state $|0\rangle_L$ by preparing all physical data qubits in the $|0\rangle$ state and measuring the X stabilizers once. Indeed, projecting (via measuring) the X-type stabilizers is what "adds" the missing constraints and creates the entanglement of the encoded $|0\rangle_L$. Conversely, to prepare the logical $|+\rangle_L$ state, we initialize all physical qubits in the $|+\rangle$ basis and measure the Z stabilizers.

Logical measurement in the Z_L basis is performed by measuring all physical data qubits in the Z basis (M_z). We then aggregate these results to determine the parity of the logical operator. Similarly, measurement in the X_L basis involves measuring all physical data qubits in the X basis (M_x).

7.9.2 The Surface Code Memory Experiment

The "Memory Experiment" is the basic test of whether a quantum error-correcting code actually works, relying simply on logical initialization, repeated stabilizer measurement and logical measurement. The goal is simple: can we keep a logical qubit alive longer than its constituent physical parts?

The procedure follows a specific cadence. First, we initialize the logical state $|0\rangle_L$. Then, we enter a "syndrome extraction" phase consisting of repeated rounds of stabilizer measurements. In each round, we measure the parity of X-type and Z-type stabilizers to detect if any Pauli errors have occurred.

Stabilization Rounds

We don't just measure once. Because our measurements themselves are noisy, we must repeat these rounds d times (where d is the code distance) to build up enough temporal confidence to distinguish between a real qubit error and a faulty measurement outcome.

Finally, we perform a logical Z_L measurement. By comparing the initial state, the history of measurement syndromes, and the final readout, we can determine if the logical information remained intact.

Storing a logical qubit via a memory experiment is not enough though, we also need to manipulate it. Some manipulations are easier than others: for the surface code, the logical CNOT and the Hadamard are **transversal** (modulo some hidden details), it means that a logical gate is applied by simply performing the same gate on every physical qubit. However, the Eastin-Knill theorem tells us that no single code can implement a universal gate set transversally. How to get a non-Clifford gate then? We rely on quantum teleportation.

7.9.3 Non-Clifford Gates: T via Teleportation

The most difficult gate to implement fault-tolerantly is the T gate ($T = \mathrm{diag}(1, e^{i\pi/4})$), which is essential for universality. Since we cannot perform it transversally on the surface code without destroying the code's protection, we use **Gate Teleportation**.

The idea is to "pre-bake" the non-Cliffordness into a special resource state, called a **Magic State** $|\phi\rangle = T|+\rangle$. We then use only Clifford operations (which are "easy" and fault-tolerant) to consume this magic state and teleport the T gate onto our target logical state $|\psi\rangle$.

The circuit logic backpropagate a T gate into a classic teleportation circuit, following the relation $TXT^\dagger = e^{-i\pi/4}SX$. If we measure the resource qubit and get a "1" outcome, we must apply a corrective S gate (an "Easy" Clifford) to finish the operation.

$$|\phi\rangle = T|+\rangle = \frac{|0\rangle + e^{i\pi/4}|1\rangle}{\sqrt{2}}$$

Then, the process of **Magic State Distillation** allows us to convert many of such noisy non-Clifford states into one highly pure state, which we then inject into our computation using the circuit shown in fig. 11.

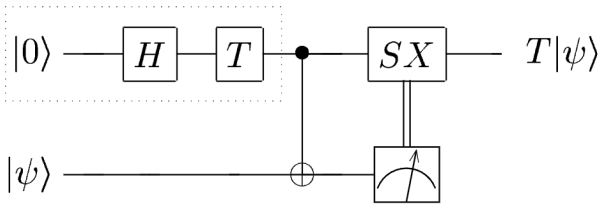

Figure 11: Circuit diagram for T-gate teleportation using a magic state and a corrective S gate.

Exercise 19: Magic State Properties

The magic state is $|T\rangle = T|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{i\pi/4}|1\rangle)$. 1. Show that $|T\rangle$ is *not* a stabilizer state (i.e., it cannot be the $+1$ eigenstate of any single Pauli operator). 2. Compute $\langle T|X|T\rangle$, $\langle T|Y|T\rangle$, and $\langle T|Z|T\rangle$. Where does $|T\rangle$ sit on the Bloch sphere? 3. Why is it significant that the T gate cannot be implemented transversally? What does the Eastin-Knill theorem say?

Answer

1. A stabilizer state must lie on an axis of the Bloch sphere ($\pm x$, $\pm y$, or $\pm z$). $|T\rangle$ has a phase of $\pi/4$ on the $|1\rangle$ component, placing it at a 45° angle between the x and y axes on the equator. It is not an eigenstate of X, Y, or Z, so it is not a stabilizer state.
2. $\langle T|X|T\rangle = \frac{1}{2}(\langle 0| + e^{-i\pi/4}\langle 1|)(X)(|0\rangle + e^{i\pi/4}|1\rangle) = \frac{1}{2}(\langle 0| + e^{-i\pi/4}\langle 1|)(|1\rangle + e^{i\pi/4}|0\rangle) = \frac{1}{2}(e^{i\pi/4} + e^{-i\pi/4}) = \cos(\pi/4) = 1/\sqrt{2}$. Similarly $\langle T|Y|T\rangle = 1/\sqrt{2}$ and $\langle T|Z|T\rangle = 0$. The state sits on the equator of the Bloch sphere at angle $\pi/4$ between the $+x$ and $+y$ axes.
3. The Eastin-Knill theorem states that no quantum error-correcting code can implement a universal gate set entirely through transversal gates. Since Clifford gates alone are not universal (they can be efficiently classically simulated by the Gottesman-Knill theorem), we need at least one non-Clifford gate (T) for universality. The T gate must therefore be implemented via non-transversal methods like magic state injection and distillation.

Exercise 20: Resource Overhead for the Surface Code

A distance-d surface code uses approximately $2d^2 - 1$ physical qubits (data + ancilla) to encode 1 logical qubit.

1. How many physical qubits are needed for $d = 3, 5, 7, 9, 11$?
2. If we need a logical error rate of $P_L = 10^{-15}$ (required for useful quantum computation), and the physical error rate is $p = 10^{-3}$ with threshold $p_{\text{th}} = 10^{-2}$, estimate the required distance d using $P_L \approx 0.1 \cdot (p/p_{\text{th}})^{(d+1)/2}$.
3. How many physical qubits does this require for a single logical qubit? For 100 logical qubits?

Answer

1. Physical qubits
   - $d = 3$: $2(9) - 1 = 17$ qubits
   - $d = 5$: $2(25) - 1 = 49$ qubits
   - $d = 7$: $2(49) - 1 = 97$ qubits
   - $d = 9$: $2(81) - 1 = 161$ qubits
   - $d = 11$: $2(121) - 1 = 241$ qubits
2. We need $0.1 \cdot (0.1)^{(d+1)/2} \leq 10^{-15}$. $(0.1)^{(d+1)/2} \leq 10^{-14}$, so $(d+1)/2 \geq 14$, giving $d \geq 27$. We

need $d = 27$ (rounding to the nearest odd number).

3. For $d = 27$: $2(27^2) - 1 = 2(729) - 1 = 1457$ physical qubits per logical qubit. For 100 logical qubits: $100 \times 1457 = 145{,}700$ physical qubits. This does not include the overhead for magic state distillation factories, which can increase the total by a lot.

8 Experimental Implementations of QEC

We end these lecture notes by some "preview material" that we will be covered in detail during next week's lecture, connecting the theoretical framework above to the state of experimental QEC. It is based on the review [10] and the associated database accessible at github.com/francois-marie/awesome-quantum-computing-experiments.

8.1 Platforms and Codes

Experimental efforts have explored a variety of QEC codes across multiple hardware platforms. The platforms considered include:

- **Superconducting Circuits** (Transmons, Fluxoniums) [11]
- **Trapped Ions** [12]
- **Neutral Atoms** (in optical tweezers) [13]
- **Photonic Systems** (linear optics, integrated photonics)
- **NV Centers** in Diamond [14]
- **Nuclear Magnetic Resonance** (NMR, for historical reasons)

The codes implemented experimentally range from simple repetition codes to topological codes:

- **Repetition Codes** (distance d, $[[d, 1, d]]$ for bit-flip or phase-flip) [2, 3, 15]
- **Four-qubit codes** ($[[4, 1, 2]]$ or $[[4, 2, 2]]$) [16, 17]
- $[[5, 1, 3]]$ **Perfect Code** [18]
- **Steane** $[[7, 1, 3]]$ **Code**
- **Surface Codes** (various distances) [2, 19, 20]
- **Color Codes** [21, 22]
- **Bacon-Shor Codes** [23]

8.2 Timeline

The field has progressed through several milestones:

1. **First QEC demonstrations (1998-2011):** Early NMR and trapped ion experiments demonstrated basic 3-qubit codes [15, 24, 25].
2. **Scaling to larger codes (2014-2021):** Superconducting circuits began demonstrating repeated error correction and exponential error suppression with code distance [3, 26].
3. **Below-threshold operation (2022-2024):** Google demonstrated that increasing code distance from $d = 3$ to $d = 5$ to $d = 7$ reduces logical error rates, crossing the break-even point [2, 19].
4. **Logical processors (2024-present):** Neutral atom platforms demonstrated 48 logical qubits with color codes [20], and superconducting platforms demonstrated logical computation [17].

Intuition: Where We Stand

Multiple platforms have now demonstrated below-threshold operation for the repetition code and small surface codes. The next step is below-threshold operation for the full surface code at increasing distances, and performing useful logical computations with encoded qubits. This requires improvements in qubit quality, qubit count, and real-time decoding speed.

We will discuss these experimental results in depth during next week's lecture.

9 Conclusion

We started this lecture by seeing that classical error correction protects information through redundancy and majority voting. The Hamming code does this efficiently, with overhead growing only logarithmically.

In the quantum setting, two constraints change everything: the No-Cloning Theorem forbids copying unknown states, and measurement back-action means we cannot inspect qubits without disturbing them. These constraints force us to use parity measurements and entanglement instead.

The 3-qubit repetition codes showed the central trick: measure *parity* rather than individual qubits just like for the Hamming code, and you can extract error information without collapsing the logical state. The Shor code extended this to protect against all single-qubit errors simultaneously, and showed that continuous errors are effectively digitized by syndrome measurement.

The stabilizer formalism gave us a compact language to describe codes: instead of tracking exponentially many amplitudes, we track a linear number of operators. This led to the surface code, where all stabilizers are local.

We discussed the threshold theorem (error correction works as long as physical error rates are below a critical value) and fault-tolerant circuit design (careful gate ordering prevents single faults from cascading into logical errors). We also saw how logical gates are implemented, with Clifford gates being straightforward and the T gate requiring magic state injection.

In essence, without quantum error correction, the algorithms from our previous lectures (Shor, Grover, QPE) remain theoretical. QEC is what bridges the gap and connects the noisy physical qubits from our platform lectures to the logical qubits that algorithms need. Next week, we look at how different platforms implement these codes in practice.

References

1. Nielsen, M. A. & Chuang, I. L. *Quantum Computation and Quantum Information 10th Anniversary Edition*. (Cambridge University Press, 2000). doi:10.1017/CBO9780511976667.
2. Acharya, R. *et al.* Quantum error correction below the surface code threshold. *Nature* **638**, 920–926 (2025).
3. Chen, Z. *et al.* Exponential suppression of bit or phase errors with cyclic error correction. *Nature* **595**, 383–387 (2021).
4. Shannon, C. E. A mathematical theory of communication. *The Bell System Technical Journal* **27**, 379–423 (1948).
5. Shor, P. W. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A* **52**, R2493–R2496 (1995).
6. Terhal, B. M. Quantum Error Correction for Quantum Memories. *Rev. Mod. Phys.* **87**, 307–346 (2015).
7. Gottesman, D. E. Stabilizer Codes and Quantum Error Correction. (California Institute of Technology, 1997). doi:10.7907/rzr7-dt72.
8. Fowler, A. G., Mariantoni, M., Martinis, J. M. & Cleland, A. N. Surface codes: Towards practical large-scale quantum computation. Preprint at https://doi.org/10.48550/arXiv.1208.0928 (2012).
9. Kitaev, A. Y. Fault-tolerant quantum computation by anyons. *Annals of Physics* **303**, 2–30 (2003).
10. Le Régent, F.-M. Awesome Quantum Computing Experiments: Benchmarking Experimental Progress Towards Fault-Tolerant Quantum Computation. (2025) doi:10.48550/arXiv.2507.03678.
11. Kjaergaard, M. *et al.* Superconducting Qubits: Current State of Play. *Annu. Rev. Condens. Matter Phys.* **11**, 369–395 (2020).
12. Bruzewicz, C. D., Chiaverini, J., McConnell, R. & Sage, J. M. Trapped-Ion Quantum Computing: Progress and Challenges. *Applied Physics Reviews* **6**, 021314 (2019).

13. Wintersperger, K. *et al.* Neutral Atom Quantum Computing Hardware: Performance and End-User Perspective. *EPJ Quantum Technol.* **10**, 32 (2023).
14. Pezzagna, S. & Meijer, J. Quantum computer based on color centers in diamond. *Applied Physics Reviews* **8**, 011308 (2021).
15. Cory, D. G. *et al.* Experimental Quantum Error Correction. *Phys. Rev. Lett.* **81**, 2152–2155 (1998).
16. Linke, N. M. *et al.* Fault-tolerant quantum error detection. *Sci. Adv.* **3**, e1701074 (2017).
17. Reichardt, B. W. *et al.* Logical computation demonstrated with a neutral atom quantum processor. (2024) doi:10.48550/arXiv.2411.11822.
18. Knill, E., Laflamme, R., Martinez, R. & Negrevergne, C. Implementation of the Five Qubit Error Correction Benchmark. *Phys. Rev. Lett.* **86**, 5811–5814 (2001).
19. Acharya, R. *et al.* Suppressing quantum errors by scaling a surface code logical qubit. *Nature* **614**, 676–681 (2023).
20. Bluvstein, D. *et al.* Logical quantum processor based on reconfigurable atom arrays. *Nature* **626**, 58–65 (2024).
21. Bluvstein, D. *et al.* A quantum processor based on coherent transport of entangled atom arrays. *Nature* **604**, 451–456 (2022).
22. Lacroix, N. *et al.* Scaling and logic in the color code on a superconducting quantum processor. (2024) doi:10.48550/arXiv.2412.14256.
23. Egan, L. *et al.* Fault-tolerant control of an error-corrected qubit. *Nature* **598**, 281–286 (2021).
24. Chiaverini, J. *et al.* Realization of quantum error correction. *Nature* **432**, 602–605 (2004).
25. Schindler, P. *et al.* Experimental Repetitive Quantum Error Correction. *Science* **332**, 1059–1061 (2011).
26. Kelly, J. *et al.* State preservation by repetitive error detection in a superconducting quantum circuit. *Nature* **519**, 66–69 (2015).