

Modern Quantum Error Correction

Instructor Blackboard Companion: Six 60-Minute Sessions

ICTP–SAIFR Summer School · São Paulo · 2026

July 2026

Table of contents

Notation and conventions	2
1 Lecture 1 — Introduction to quantum error correction through the repetition code	3
1.1 Derivation script	3
1.2 Conceptual close	6
1.3 Exercises	6
2 Lecture 2 — Pauli and Clifford structure and efficient simulation	7
2.1 Derivation script	7
2.2 Exercises	10
3 Lecture 3 — Physical noise to Pauli channels	11
3.1 Derivation script	11
3.2 Exercises	16
4 Lecture 4 — Pauli errors and detecting regions in the repetition circuit	17
4.1 Insert Pauli errors into the circuit	17
4.2 Detectors and detecting regions	17
4.3 Overlap explains correlated detection events	17
4.4 From detecting regions to a detector error model	20
4.5 Exercises	21
5 Lecture 5 — Decoding as probabilistic inference	22
5.1 Errors, detectors, and the parity-check matrix	22
5.2 Enforcing the syndrome constraints	23
5.3 Add the noise model	23
5.4 Add logical observables	24
5.5 Most-likely error versus most-likely logical class	24
5.6 Logical error rate	24
5.7 Exercises	25
6 Lecture 6 — Hands-on QEC with Stim and PyMatching	26
6.1 Set up the notebook	26
6.2 Build an ideal repetition-code memory	26
6.3 Verify the ideal circuit	28
6.4 Add noise	28
6.5 Compile and inspect the detector error model	28
6.6 Decode with PyMatching	29
6.7 Student experiments	29
6.8 Lab report	30
6.9 Exercises	30
7 Compact misconception index	31
8 References	31

Notation and conventions

- Computational basis: $Z|0\rangle = +|0\rangle$, $Z|1\rangle = -|1\rangle$.
- Density matrix and Bloch vector:

$$\rho = \begin{pmatrix} \rho_{00} & \rho_{01} \\ \rho_{10} & \rho_{11} \end{pmatrix} = \frac{1}{2}(I + xX + yY + zZ).$$

- $\sigma_- = |0\rangle\langle 1|$, $\sigma_+ = |1\rangle\langle 0|$.
- T_1 is the population-relaxation time; T_ϕ is the pure-dephasing time; T_2 is the total coherence time.
- Products such as X_2Z_4 mean tensor products with identity elsewhere. The weight $\text{wt}(P)$ is the number of nonidentity factors.
- Paulis are normally considered **modulo global phase**, so a Pauli string is represented by its I, X, Y, Z letters and commutation sign.
- Stabilizer outcomes are signs ± 1 . A syndrome bit is $s_i = 0$ for $+1$ and $s_i = 1$ for -1 .
- $\oplus$ is XOR. Measurement records $m_{a,t}$, detector bits $d_{a,t}$, and observable flips ℓ_j are binary.
- Stim targets are written D_i for detector i and L_j for logical observable j . A detector is a parity of records, not a physical sensor.
- CNOT notation is $c \rightarrow t$, with the control listed first.
- Decoder output is a predicted observable-flip vector $\hat{\ell}$, not necessarily a microscopic correction.
- All probability models are declared models. Generated circuits are pedagogical baselines, not calibrated hardware descriptions.

1 Lecture 1 — Introduction to quantum error correction through the repetition code

1.1 Derivation script

1.1.1 Classical repetition and majority intuition

Suppose a classical bit $b \in \{0, 1\}$ is sent through three independent bit-flip locations. Encode

$$0 \mapsto 000, \quad 1 \mapsto 111.$$

If at most one bit flips, majority vote recovers b . Equivalently, compare neighboring bits through the parity checks

$$x_1 \oplus x_2, \quad x_2 \oplus x_3.$$

The two check bits locate any single flip without needing to know the encoded value:

Corruption	$x_1 \oplus x_2$	$x_2 \oplus x_3$
none	0	0
bit 1 flips	1	0
bit 2 flips	1	1
bit 3 flips	0	1

This separates **information** from **redundant consistency checks**. Quantum error correction keeps that separation but cannot inspect or copy an unknown quantum state directly.

1.1.2 Quantum encoding without cloning

Let an unknown qubit be $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. The repetition encoding is the isometry

$$|\psi\rangle \mapsto |\psi_L\rangle = \alpha|000\rangle + \beta|111\rangle.$$

This is not $|\psi\rangle^{\otimes 3}$ and therefore does not clone the state. For example, two CNOTs from the input qubit into ancillas initialized in $|00\rangle$ produce the entangled encoded state while preserving linearity.

The code space is

$$\mathcal{C} = \text{span}\{|0_L\rangle, |1_L\rangle\}, \quad |0_L\rangle = |000\rangle, \quad |1_L\rangle = |111\rangle.$$

An $[[n, k, d]]$ quantum code embeds a 2^k -dimensional logical space into n physical qubits and has quantum distance d . Here $n = 3$ and $k = 1$; the important qualification about d appears below.

1.1.3 Detection versus correction

The two-qubit repetition encoding

$$|\psi_L\rangle = \alpha|00\rangle + \beta|11\rangle$$

illustrates error **detection**. Its single check is $S = Z_1 Z_2$, with projectors

$$\Pi_{\pm} = \frac{1}{2}(I \pm S).$$

The codespace has eigenvalue $+1$, while either X_1 or X_2 moves the state to the -1 error subspace. Measuring the check therefore reveals that one bit flip occurred without revealing α or β . However, both errors produce the same syndrome:

$$X_1 \mathcal{C} = X_2 \mathcal{C}.$$

The measurement cannot determine which qubit should be corrected. Applying X_1 when the error was X_2 produces X_1X_2 , which acts as a logical bit flip. The two-qubit code detects one X error but cannot correct it.

The three-qubit code adds a second independent check. The pair Z_1Z_2, Z_2Z_3 assigns a distinct syndrome to each single-qubit X error, so the decoder can identify an appropriate recovery. Detection asks whether the state left the codespace; correction requires enough syndrome information to return it without changing its logical state (Roffe 2019).

1.1.4 Checks, syndromes, and logical information

The repetition-code checks are

$$S_1 = Z_1Z_2, \quad S_2 = Z_2Z_3.$$

Every state in $\mathcal{C}$ has eigenvalue $+1$ for both checks. Measuring them reveals whether neighboring computational-basis values agree, but it does not reveal α or β . Write outcome $+1$ as syndrome bit zero and outcome -1 as syndrome bit one. For an error E ,

$$S_i E = (-1)^{s_i} E S_i.$$

Thus an error that changes a check sign produces syndrome information:

Error	s_1	s_2	Single- X interpretation
I	0	0	no bit flip
X_1	1	0	qubit 1
X_2	1	1	qubit 2
X_3	0	1	qubit 3
X_1X_2	0	1	same syndrome as X_3

The last row is a warning: a syndrome constrains an error class; it does not uniquely identify the microscopic error.

Logical observables distinguish states *within* the code space. One choice is

$$\bar{Z} = Z_1, \quad \bar{X} = X_1X_2X_3.$$

Here $\bar{Z}|0_L\rangle = +|0_L\rangle$ and $\bar{Z}|1_L\rangle = -|1_L\rangle$, while $\bar{X}$ swaps the two logical basis states. Equivalent representatives, such as Z_2 or Z_3 for $\bar{Z}$, act identically on the code space because they differ by checks. A physical operator can therefore preserve every check outcome and still change the encoded information.

1.1.5 Distance, detection, and correction

For Pauli errors, distance is the minimum weight of a nontrivial logical operator: an operator that preserves the code space but acts nontrivially within it. A distance- d quantum code detects all errors of weight less than d and corrects arbitrary errors of weight at most

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor,$$

provided the stated error set satisfies the correction conditions.

The three-qubit repetition code has an asymmetric protection profile. Against the intended X -only noise, the shortest undetectable X -type logical is $X_1X_2X_3$, so

$$d_X = 3.$$

It detects any one or two X flips and corrects one X flip by majority logic. But $Z_1 = \bar{Z}$ is already a weight-one logical phase operation. Therefore, as a code against arbitrary quantum errors, its overall quantum distance is

$$\boxed{d = 1},$$

not three. Calling it an $[[3, 1, 3]]$ quantum code without the X -noise qualification is incorrect; its full quantum parameters are $[[3, 1, 1]]$, with $d_X = 3$ and $d_Z = 1$ under this convention.

1.1.6 Why the repetition code does not correct general quantum noise

The checks

$$S_1 = Z_1 Z_2, \quad S_2 = Z_2 Z_3$$

are designed to detect X errors. Each X_i anticommutes with at least one check and therefore changes the syndrome. A single Z_i , however, commutes with both checks:

$$[Z_i, S_1] = [Z_i, S_2] = 0.$$

Consequently, a phase error produces the same syndrome as no error. For example,

$$Z_1|\psi_L\rangle = Z_1(\alpha|000\rangle + \beta|111\rangle) = \alpha|000\rangle - \beta|111\rangle = \bar{Z}|\psi_L\rangle.$$

The encoded state remains inside the codespace, but its logical phase has changed. Because $\bar{Z} = Z_1$ has weight one, this logical error cannot be detected by the repetition checks.

A $Y_i = iX_iZ_i$ error does trigger an X -type syndrome, but correcting only the inferred X_i component leaves the undetected Z_i component behind. Thus the code does not correct a general single-qubit error

$$E_i = aI + bX_i + cY_i + dZ_i.$$

One can reverse the protection by encoding in the conjugate basis,

$$|0_L\rangle = |+++\rangle, \quad |1_L\rangle = |--\rangle,$$

and measuring X_1X_2 and X_2X_3 . This phase-flip repetition code detects Z errors but is then vulnerable to X errors. Protecting an arbitrary quantum state requires both X -type and Z -type checks, as in CSS, Shor, and surface codes (Roffe 2019).

1.1.7 Quantitative error suppression

Assume independent X errors with probability p on each of the three physical qubits, and assume perfect encoding, syndrome measurement, and recovery. Majority decoding succeeds when zero or one qubit flips. It fails when two or three qubits flip, so

$$\begin{aligned} p_L &= \binom{3}{2}p^2(1-p) + \binom{3}{3}p^3 \\ &= 3p^2(1-p) + p^3 \\ &= 3p^2 - 2p^3. \end{aligned}$$

For $p \ll 1$,

$$\boxed{p_L \simeq 3p^2}.$$

The unencoded error is first order in p , while the encoded logical error is second order. This is the simplest quantitative example of error suppression: all weight-one X errors are correctable, so failure requires at least two faults (Roffe 2019).

The conclusion is deliberately limited. It assumes X -only noise and error-free correction circuitry. The crossover at $p = 1/2$ for this ideal majority-vote model is not a realistic fault-tolerance threshold.

At high level, a code corrects a set of errors $\{E_a\}$ exactly when their effects remain distinguishable without revealing the logical state:

$$\Pi_{\mathcal{C}} E_a^\dagger E_b \Pi_{\mathcal{C}} = c_{ab} \Pi_{\mathcal{C}}.$$

This Knill–Laflamme condition says that within the code space, $E_a^\dagger E_b$ contains no logical-state information. For the intended set $\{I, X_1, X_2, X_3\}$, the repetition code satisfies this condition; adding arbitrary single-qubit Z errors does not.

1.1.8 Distance is not a threshold

Distance is a property of one code: it measures the smallest undetectable logical operation under the chosen error model. A fault-tolerance threshold is an asymptotic property of a **family** of implementations, together with a noise model, syndrome-extraction circuit, and decoder. Below that family’s threshold, increasing distance suppresses logical failure despite the extra physical locations. Above it, increasing distance need not help and can hurt.

This is not a numerical threshold derivation. A threshold value cannot be read from d , from one code instance, or from a small simulation. Establishing one requires controlled scaling across increasing distances under a fixed, documented model.

1.2 Conceptual close

The repetition code introduces the core architecture: encode logical information into a subspace, measure redundant checks rather than the logical amplitudes, infer an error class from a syndrome, and judge protection by logical operators and distance. The next lecture develops the Pauli and Clifford language that makes these calculations systematic.

1.3 Exercises

1.3.1 Easy

For the three-qubit repetition code with checks $S_1 = Z_1 Z_2$ and $S_2 = Z_2 Z_3$, compute the syndrome bits for

$$I, \quad X_1, \quad X_2, \quad X_3.$$

Explain why measuring these checks does not reveal the logical amplitudes α, β .

1.3.2 Medium

Assume independent X errors with probability $p = 0.02$ on each data qubit and perfect syndrome extraction.

1. Compute the exact majority-decoding failure probability $p_L = 3p^2(1-p) + p^3$.
2. Compare it with the unencoded error probability p .
3. Show explicitly that Z_1 produces no syndrome but changes $\alpha|000\rangle + \beta|111\rangle$.
4. Explain why the same code can suppress X errors while having overall quantum distance one.

2 Lecture 2 — Pauli and Clifford structure and efficient simulation

2.1 Derivation script

2.1.1 Start from the repetition-code circuit

Consider the complete repetition-code circuit in Figure 1. It first encodes one input qubit into three data qubits and then measures the two parity checks. The circuit was generated directly with Stim.

In a Stim diagram, R means **reset to $|0\rangle$** ; it is not a rotation gate. Qubit $q0$ is not reset because it carries the unknown input. Qubits $q1, q2$ are reset as redundancy qubits, while $q3, q4$ are reset as fresh syndrome-measurement ancillas.

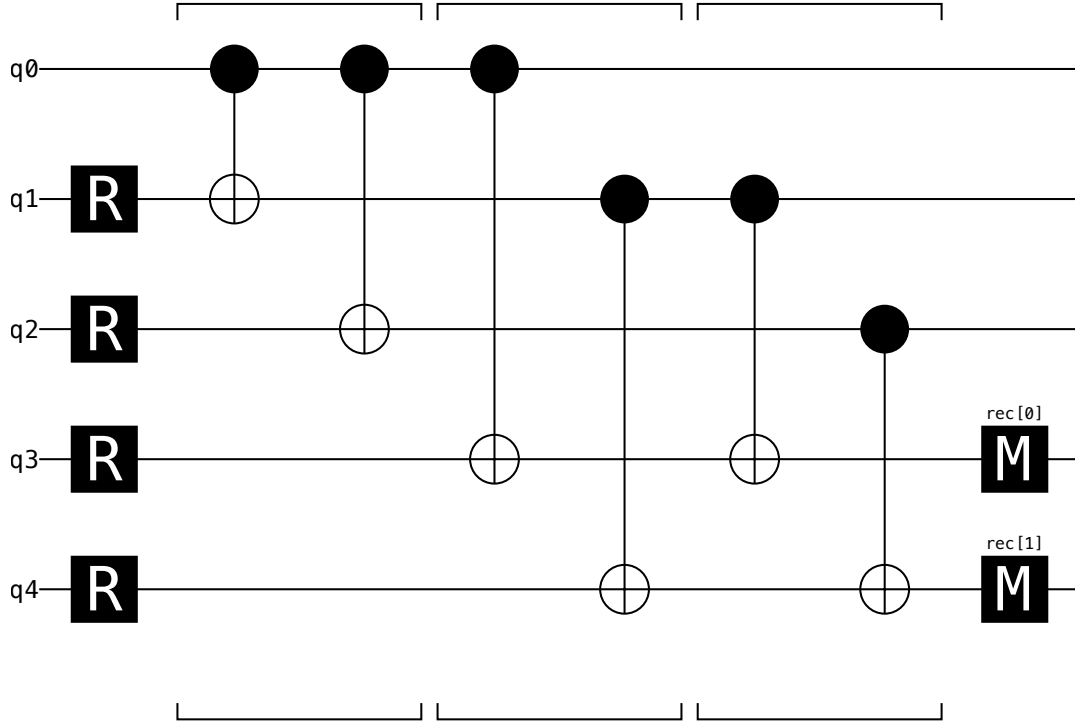


Figure 1: Three-qubit repetition-code encoding and one round of syndrome extraction. Qubit $q0$ carries the input state; $q1, q2$ are redundancy qubits; and $q3, q4$ are measurement ancillas for Z_0Z_1 and Z_1Z_2 , respectively. R prepares $|0\rangle$, and the final measurements produce the two syndrome bits.

Let the input on $q0$ be $\alpha|0\rangle + \beta|1\rangle$. The first two CNOTs encode

$$(\alpha|0\rangle + \beta|1\rangle)|00\rangle \mapsto \alpha|000\rangle + \beta|111\rangle.$$

The remaining CNOTs copy data parities onto the measurement ancillas. On a computational-basis component $|x_0x_1x_2\rangle$,

$$q3 = x_0 \oplus x_1, \quad q4 = x_1 \oplus x_2.$$

The two measured bits are therefore the eigenvalue bits of

$$S_1 = Z_0Z_1, \quad S_2 = Z_1Z_2.$$

For either encoded codeword, both parities are zero. A single data-bit flip changes one or both outcomes without revealing the amplitudes α, β .

Now inspect the circuit rather than the state amplitudes. Every preparation is a Pauli- Z eigenstate, every measurement is a Pauli- Z measurement, and every entangling gate is a CNOT. Propagating the measured observable Z_{q3} backward through its two CNOTs gives

$$Z_{q3} \leftarrow Z_1 Z_{q3} \leftarrow Z_0 Z_1 Z_{q3}.$$

Because $q3$ was prepared with $Z_{q3} = +1$, its final measurement is exactly a measurement of $Z_0 Z_1$. The same argument gives $Z_1 Z_2$ for $q4$.

This circuit exposes the useful algebraic closure:

- the code constraints and measured observables are Pauli strings;
- CNOT conjugation maps Pauli strings to Pauli strings;
- Pauli faults remain Pauli faults as they propagate through the CNOT network;
- syndrome bits are determined only by commutation between propagated faults and measured Pauli checks.

A unitary that maps every Pauli operator to another Pauli operator under conjugation is a **Clifford** operation. Thus the repetition-code circuit is not merely expressible using Pauli and Clifford operations: its encoding, checks, syndromes, and fault propagation are closed within that language. Stim can simulate it by tracking Pauli constraints and faults instead of storing all 2^n state amplitudes.

The input coefficients α, β may be arbitrary, so the encoded state is not necessarily a stabilizer state. What remains efficiently describable is the codespace, the Clifford circuit, the Pauli measurements, and the action of Pauli faults on logical observables. The next sections formalize the Pauli and Clifford groups underlying this circuit.

2.1.2 Pauli strings modulo phase

The one-qubit Pauli group is

$$\mathcal{P}_1 = \{\omega P : \omega \in \{\pm 1, \pm i\}, P \in \{I, X, Y, Z\}\}.$$

For syndrome and propagation reasoning, global phase is irrelevant, so Pauli strings can be represented modulo phase by their I, X, Y, Z letters. An n -qubit string is $P = P_1 \otimes \cdots \otimes P_n$. Two strings obey

$$PQ = (-1)^{N_{\text{ac}}} QP,$$

where N_{ac} counts positions at which both factors are nonidentity and the local Paulis differ. They commute for even N_{ac} and anticommute for odd N_{ac} .

For example,

$$P = X_1 Z_2 I_3, \quad Q = Z_1 X_2 X_3$$

anticommute locally at sites 1 and 2, so the two signs cancel and $PQ = QP$. Identity-versus-Pauli positions do not contribute.

2.1.3 Stabilizers and logical Paulis

A stabilizer code is the simultaneous $+1$ eigenspace of a commuting group $\mathcal{S}$:

$$\mathcal{C} = \{|\psi\rangle : S|\psi\rangle = |\psi\rangle, \forall S \in \mathcal{S}\}.$$

The repetition checks from Lecture 1, $Z_1 Z_2$ and $Z_2 Z_3$, are stabilizer generators. If an error anticommutes with a generator, it flips that generator's measured sign. This turns the syndrome table into a commutation calculation.

The Pauli normalizer

$$N(\mathcal{S}) = \{P : PSP^\dagger = S\}$$

contains Paulis that preserve the code space. Stabilizers act trivially there, while elements of $N(\mathcal{S}) \setminus \mathcal{S}$ represent nontrivial logical Paulis. For the repetition code, $\bar{X} = X_1 X_2 X_3$ and $\bar{Z} = Z_1$ are examples. Multiplying a logical representative by a stabilizer changes its physical support but not its action on encoded states.

2.1.4 Clifford normalizer and conjugation rules

The Clifford group is the normalizer of the Pauli group:

$$\mathcal{C}_n^{\text{Cliff}} = \{U : U\mathcal{P}_n U^\dagger = \mathcal{P}_n\}.$$

Up to phase,

$$HXH = Z, \quad HZH = X, \quad HYH = -Y,$$

$$SXS^\dagger = Y, \quad SY S^\dagger = -X, \quad SZS^\dagger = Z.$$

For $\text{CNOT}_{c \rightarrow t}$,

$$X_c \mapsto X_c X_t, \quad Z_c \mapsto Z_c,$$

$$X_t \mapsto X_t, \quad Z_t \mapsto Z_c Z_t.$$

These rules propagate Pauli observables and Pauli faults through a circuit without expanding a 2^n -component state vector.

2.1.5 Measuring parity with an ancilla

To measure $Z_1 Z_2$, prepare ancilla a in $|0\rangle$, apply $\text{CNOT}_{1 \rightarrow a}$ and $\text{CNOT}_{2 \rightarrow a}$, then measure Z_a . On a computational-basis component,

$$|x_1 x_2\rangle |0\rangle_a \mapsto |x_1 x_2\rangle |x_1\rangle_a \mapsto |x_1 x_2\rangle |x_1 \oplus x_2\rangle_a.$$

The ancilla records parity while coherence is retained inside each parity subspace. Equivalently, propagate the measured observable backward:

$$Z_a \xleftarrow{\text{CNOT}_{2 \rightarrow a}} Z_2 Z_a \xleftarrow{\text{CNOT}_{1 \rightarrow a}} Z_1 Z_2 Z_a.$$

Because the initialized ancilla has $Z_a = +1$, the final measurement measures $Z_1 Z_2$. This circuit-level view will matter when faults can occur between the CNOTs.

2.1.6 Stabilizer updates and efficient simulation

Under a Clifford gate, conjugate every tracked stabilizer generator by the gate's Pauli rule. A Pauli preparation adds a known stabilizer; a Pauli measurement either reads a determined sign or updates the generating set to include the measured observable. A stochastic Pauli fault toggles signs according to commutation. These operations manipulate polynomially many Pauli strings rather than exponentially many amplitudes.

This is the Gottesman–Knill intuition: circuits composed of stabilizer-state preparations, Clifford gates, Pauli measurements, classical feed-forward, and stochastic Pauli noise can be simulated efficiently on a classical computer (Gottesman 1997). No tableau implementation details are needed here.

Stim exploits exactly this structure for fast stabilizer-circuit sampling and for propagating Pauli fault mechanisms into detector and logical-observable effects (Gidney 2021; Stim contributors 2026). This tractability has boundaries. Arbitrary non-Clifford gates, coherent or otherwise non-Pauli noise, and leakage outside the computational subspace are not represented exactly by the basic stabilizer/Pauli model; they require approximation, specialized extensions, or more general simulation methods.

Large codes use the same local-check principle. In a surface code, local X - and Z -type checks are repeatedly measured across a two-dimensional lattice. The next lecture adds the time direction: changes between repeated check outcomes become detectors.

2.2 Exercises

2.2.1 Easy

Propagate the following Pauli operators through $\text{CNOT}_{c \rightarrow t}$:

$$X_c, \quad Z_c, \quad X_t, \quad Z_t.$$

Use the results to explain why backward propagation of a measured ancilla Z_a through two data-to-ancilla CNOTs measures a two-data-qubit ZZ parity.

2.2.2 Medium

In the repetition-code circuit of Figure 1, insert an X_1 error on the middle data qubit immediately before syndrome extraction.

1. Propagate it through both CNOTs controlled by q_1 .
2. Predict the two measured syndrome bits.
3. Repeat the analysis for a Z_1 error.
4. Explain why a stabilizer simulator can perform both calculations without storing the amplitudes of the encoded state.

3 Lecture 3 — Physical noise to Pauli channels

3.1 Derivation script

3.1.1 Context: from open-system dynamics to a transmon model

A quantum processor is never perfectly isolated. Its state changes because of both intentional control and uncontrolled coupling to electromagnetic modes, materials defects, control electronics, and other environmental degrees of freedom. After tracing out those degrees of freedom, a standard Markovian model is the Lindblad master equation

$$\dot{\rho} = -i[H, \rho] + \sum_{\mu} \gamma_{\mu} \left(L_{\mu} \rho L_{\mu}^{\dagger} - \frac{1}{2} \{L_{\mu}^{\dagger} L_{\mu}, \rho\} \right).$$

At this stage, keep the interpretation high level:

- H generates coherent evolution;
- each L_{μ} identifies a physical noise process;
- γ_{μ} sets the rate of that process;
- the equation preserves trace and complete positivity;
- the Markov approximation assumes that the environment has no relevant memory on the timescale being modeled.

The example throughout this lecture is a superconducting transmon. Using the standard superconducting-circuit quantization framework described by Devoret ([Devoret 1997](#)), a transmon is a weakly anharmonic electrical oscillator formed from a Josephson junction and a capacitance. A common approximate Hamiltonian is

$$\frac{H_{\text{tr}}}{\hbar} \simeq \omega_q a^{\dagger} a - \frac{\alpha}{2} a^{\dagger 2} a^2.$$

The anharmonicity α allows the two lowest levels, $|0\rangle$ and $|1\rangle$, to be addressed as a qubit. Real quantum platforms exhibit many kinds of noise, including:

1. **Energy relaxation:** an excitation is lost and $|1\rangle \rightarrow |0\rangle$. Use $L_1 = \sigma_- = |0\rangle\langle 1|$ with rate $\gamma_1 = 1/T_1$.
2. **Pure dephasing:** fluctuations in the qubit frequency randomize the relative phase of $|0\rangle$ and $|1\rangle$ without changing their populations. Use $L_{\phi} = Z$ with coefficient $1/(2T_{\phi})$.
3. **Thermal excitation:** a finite-temperature environment can drive $|0\rangle \rightarrow |1\rangle$, producing an upward transition in addition to relaxation.
4. **Leakage:** population can leave the computational subspace, for example through unwanted occupation of the transmon level $|2\rangle$.
5. **Particle or photon loss:** a photon may leave a cavity or optical mode, while a neutral atom may be lost from its trap. Such events can sometimes be treated as detectable erasures rather than unknown Pauli errors.
6. **Spontaneous emission in atomic systems:** an excited atomic level can decay into another internal state while emitting a photon.
7. **Crosstalk and correlated noise:** driving or measuring one qubit can disturb nearby qubits, and a common environment can create spatially correlated errors.
8. **Coherent control errors:** pulse-amplitude, phase, timing, or calibration errors can produce systematic over-rotations that accumulate coherently.
9. **Non-Markovian noise:** slow drift, $1/f$ fluctuations, or a structured environment can retain memory, invalidating a time-local Lindblad description with constant rates.
10. **Preparation, measurement, and reset errors:** the state may be prepared, read out, or reinitialized incorrectly even when the intervening evolution is ideal.

Here we focus only on energy relaxation and dephasing, summarized experimentally by T_1 and T_2 . Restrict to the transmon's two-level subspace and consider an idle qubit in its rotating frame, so the coherent term can be omitted.

The resulting idle model is

$$\dot{\rho} = \frac{1}{T_1} \mathcal{D}[\sigma_-](\rho) + \frac{1}{2T_{\phi}} \mathcal{D}[Z](\rho), \quad \mathcal{D}[L](\rho) = L\rho L^{\dagger} - \frac{1}{2} \{L^{\dagger} L, \rho\}.$$

Bloch-sphere view of the isolated T_1 and T_2 effects

Longitudinal relaxation: T_1
 $z(t) = 1 - 2e^{-t/T_1}$ for initial $|1\rangle$

Transverse coherence decay: T_2
 $x(t) = e^{-t/T_2}$ for initial $|+\rangle$



Figure 2: Bloch-sphere view of the isolated relaxation processes. Longitudinal T_1 relaxation drives an initially excited qubit from $|1\rangle$ toward the ground state $|0\rangle$. Transverse T_2 decay shrinks the coherence of an initial $|+\rangle$ state toward the center.

Figure 2 shows the two effects separately. In a physical transmon they generally act simultaneously: an equatorial state moves inward while also drifting toward $|0\rangle$.

Long-time limit. For $t \gg T_1$, zero-temperature relaxation removes the excited-state population and $z(t) \rightarrow +1$. For $t \gg T_2$, the transverse components satisfy $x(t), y(t) \rightarrow 0$. With both processes present, the steady state of this model is therefore

$$\rho_{ss} = |0\rangle\langle 0|.$$

Pure dephasing by itself has no unique steady state: it removes the off-diagonal coherences but leaves the initial populations unchanged, so every density matrix diagonal in the energy basis is stationary. A real transmon at finite temperature also experiences upward transitions; its long-time state is then a mixed thermal state rather than exactly $|0\rangle$.

This deliberately narrow model is simple enough to derive explicitly while retaining the two experimentally familiar quantities T_1 and T_2 .

We now compute the matrix elements in detail. This will show why relaxation contributes only half its population-decay rate to coherence decay and lead to

$$\frac{1}{T_2} = \frac{1}{2T_1} + \frac{1}{T_\phi}.$$

3.1.2 Relaxation and dephasing

[**Essential**] Write the GKSL form for one jump $L = \sqrt{\kappa}\sigma_-$:

$$\dot{\rho} = L\rho L^\dagger - \frac{1}{2}\{L^\dagger L, \rho\} = \kappa \left(\sigma_- \rho \sigma_+ - \frac{1}{2}\{|1\rangle\langle 1|, \rho\} \right).$$

Compute each piece without skipping the intermediate matrices:

$$\sigma_- \rho \sigma_+ = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \rho_{00} & \rho_{01} \\ \rho_{10} & \rho_{11} \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} \rho_{11} & 0 \\ 0 & 0 \end{pmatrix},$$

$$|1\rangle\langle 1|\rho = \begin{pmatrix} 0 & 0 \\ \rho_{10} & \rho_{11} \end{pmatrix}, \quad \rho|1\rangle\langle 1| = \begin{pmatrix} 0 & \rho_{01} \\ 0 & \rho_{11} \end{pmatrix}.$$

Therefore

$$\dot{\rho} = \kappa \begin{pmatrix} \rho_{11} & -\rho_{01}/2 \\ -\rho_{10}/2 & -\rho_{11} \end{pmatrix},$$

so

$$\dot{\rho}_{11} = -\kappa\rho_{11}, \quad \dot{\rho}_{00} = +\kappa\rho_{11}, \quad \dot{\rho}_{01} = -\frac{\kappa}{2}\rho_{01}, \quad \dot{\rho}_{10} = -\frac{\kappa}{2}\rho_{10}.$$

With $T_1 = 1/\kappa$, $\eta = e^{-t/T_1}$, and trace one:

$$\begin{aligned} \rho_{11}(t) &= \eta\rho_{11}(0), & \rho_{00}(t) &= \rho_{00}(0) + (1-\eta)\rho_{11}(0), \\ \rho_{01}(t) &= \sqrt{\eta}\rho_{01}(0), & \rho_{10}(t) &= \sqrt{\eta}\rho_{10}(0). \end{aligned}$$

For pure dephasing choose $L_\phi = \sqrt{\kappa_\phi/2}Z$. Since $L_\phi^\dagger L_\phi = (\kappa_\phi/2)I$,

$$\dot{\rho}|_\phi = \frac{\kappa_\phi}{2}(Z\rho Z - \rho) = \frac{\kappa_\phi}{2} \left[\begin{pmatrix} \rho_{00} & -\rho_{01} \\ -\rho_{10} & \rho_{11} \end{pmatrix} - \begin{pmatrix} \rho_{00} & \rho_{01} \\ \rho_{10} & \rho_{11} \end{pmatrix} \right],$$

hence populations are fixed and $\dot{\rho}_{01}|_\phi = -\kappa_\phi\rho_{01}$. Combining generators,

$$\rho_{01}(t) = e^{-t/(2T_1)}e^{-t/T_\phi}\rho_{01}(0) = e^{-t/T_2}\rho_{01}(0), \quad \boxed{\frac{1}{T_2} = \frac{1}{2T_1} + \frac{1}{T_\phi}}.$$

Question. “If pure dephasing vanished, what is the largest Markovian T_2 allowed by relaxation alone?”

Expected answer. $T_2 = 2T_1$.

Common wrong turn. Writing $1/T_2 = 1/T_1 + 1/T_\phi$. Return to the off-diagonal ODE: amplitude damping contributes $\kappa/2$, not κ .

From analog physics to digital quantum operations. The underlying device is analog: control fields vary continuously in time, the Hamiltonian acts continuously, and coupling to the environment never switches off. Quantum programs, however, are expressed as circuits—a sequence of discrete gates, preparations, and measurements. We therefore coarse-grain the continuous evolution over each control interval into a quantum operation that can be composed with the other operations in the circuit.

The coherent part is already present in the first Lindblad equation through the Hamiltonian term $-i[H(t), \rho]$. For the transmon, the Hamiltonian written above describes the uncontrolled device, while externally applied microwave or flux pulses make its parameters time dependent. By choosing the pulse amplitude, phase, frequency, and duration, one can generate coherent gates such as X and Y rotations, while frequency shifts or frame changes implement Z rotations. The circuit model records the net effect of each continuous control pulse as a discrete gate. Imperfect calibration produces coherent gate errors such as systematic over-rotations.

For the **open-system part**, the finite-time evolution must be described more generally by a completely positive trace-preserving map,

$$\mathcal{E}(\rho) = \sum_a K_a \rho K_a^\dagger, \quad \sum_a K_a^\dagger K_a = I.$$

The Kraus operators are not unique and should not automatically be interpreted as directly observed microscopic events. Together they encode the finite-time effect of the unresolved environment. Formally, a Lindblad generator $\mathcal{L}$ over a gate interval Δt produces the channel $\mathcal{E}_{\Delta t} = e^{\Delta t \mathcal{L}}$. In a circuit noise model one often writes a noisy gate approximately as an ideal gate followed by a noise channel, $\mathcal{E}_{\text{gate}} \simeq \mathcal{N} \circ \mathcal{U}_G$, although in hardware the coherent drive and dissipation occur simultaneously and the noise can depend on the gate.

The next calculation performs this analog-to-discrete conversion explicitly for relaxation during a time interval t .

3.1.3 Example: an X gate with T_1 decay

Consider a resonant drive that implements an X gate while the transmon relaxes. In the rotating frame and with $\hbar = 1$, use

$$H_X = \frac{\Omega}{2}X, \quad \tau = \frac{\pi}{\Omega}.$$

Without noise,

$$U_X = e^{-iH_X\tau} = e^{-i\pi X/2} = -iX,$$

which is the X gate up to an irrelevant global phase. Including zero-temperature T_1 decay gives the driven Lindblad equation

$$\dot{\rho} = -i \left[\frac{\Omega}{2}X, \rho \right] + \frac{1}{T_1} \mathcal{D}[\sigma_-](\rho).$$

3.1.3.1 Exact finite-time channel in the Pauli basis

The master equation defines a linear generator $\mathcal{L}_X$. Evolution for one gate time is the exact channel

$$\mathcal{E}_X = e^{\tau \mathcal{L}_X}.$$

Expand every operator in the Pauli basis

$$P_0 = I, \quad P_1 = X, \quad P_2 = Y, \quad P_3 = Z,$$

and write

$$\rho = \frac{1}{2} \sum_{j=0}^3 r_j P_j, \quad \mathbf{r} = (1, x, y, z)^T.$$

The matrix representing the Lindblad generator is

$$G_{ij} = \frac{1}{2} \text{Tr}[P_i \mathcal{L}_X(P_j)].$$

For the driven X gate with $\kappa = 1/T_1$, act on each basis operator:

$$\mathcal{L}_X(I) = \kappa Z,$$

$$\mathcal{L}_X(X) = -\frac{\kappa}{2}X,$$

$$\mathcal{L}_X(Y) = -\frac{\kappa}{2}Y + \Omega Z,$$

$$\mathcal{L}_X(Z) = -\Omega Y - \kappa Z.$$

Therefore

$$\frac{d}{dt} \begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & -\kappa/2 & 0 & 0 \\ 0 & 0 & -\kappa/2 & -\Omega \\ \kappa & 0 & \Omega & -\kappa \end{pmatrix}}_G \begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix}.$$

The Pauli-transfer matrix of the complete noisy gate is simply

$$\boxed{R_X = e^{\tau G}}, \quad \tau = \frac{\pi}{\Omega}.$$

It acts on Bloch coordinates as $\mathbf{r}(\tau) = R_X \mathbf{r}(0)$. When $\kappa = 0$, exponentiating the Y - Z rotation block through an angle $\Omega\tau = \pi$ gives

$$R_X|_{\kappa=0} = \text{diag}(1, 1, -1, -1),$$

which is exactly conjugation by the ideal X gate. For $\kappa > 0$, the first column contains the non-unital drift toward $|0\rangle$, and the Y - Z block contains the simultaneous drive and decay. Thus the exact driven channel is not generally a Pauli channel.

The transfer matrix already specifies the channel completely. To convert it to the Pauli process matrix, write the same finite-time map as

$$\mathcal{E}_X(\rho) = \sum_{P, Q \in \{I, X, Y, Z\}} \chi_{PQ} P \rho Q.$$

A generic channel requires all sixteen superoperators $P \rho Q$, not only the four Pauli conjugations $P \rho P$. To derive the change of basis, apply this channel to an input Pauli P_j and project the output onto P_i :

$$\begin{aligned} (R_X)_{ij} &= \frac{1}{2} \text{Tr}[P_i \mathcal{E}_X(P_j)] \\ &= \frac{1}{2} \sum_{P, Q} \chi_{PQ} \text{Tr}(P_i P P_j Q). \end{aligned}$$

For each pair (P, Q) , define the 4×4 matrix

$$(M^{PQ})_{ij} := \frac{1}{2} \text{Tr}(P_i P P_j Q).$$

The process-matrix representation and the Pauli-transfer representation are then related directly by the matrix expansion

$$\boxed{R_X = \sum_{P, Q} \chi_{PQ} M^{PQ}}.$$

For the unnormalized Pauli convention $\text{Tr}(P_i P_j) = 2\delta_{ij}$ used here, these sixteen matrices are orthogonal:

$$\text{Tr}_4[(M^{PQ})^\dagger M^{RS}] = 4\delta_{PR}\delta_{QS},$$

where Tr_4 is the ordinary trace of the resulting 4×4 matrix. Projecting R_X onto this matrix basis gives

$$\boxed{\chi_{PQ} = \frac{1}{4} \text{Tr}_4[(M^{PQ})^\dagger R_X]}.$$

The full route from the continuous-time model is consequently

$$\mathcal{L}_X \longrightarrow G_{ij} = \frac{1}{2} \text{Tr}[P_i \mathcal{L}_X(P_j)] \longrightarrow R_X = e^{\tau G},$$

followed by

$$\boxed{\chi_{PQ} = \frac{1}{4} \text{Tr}_4[(M^{PQ})^\dagger e^{\tau G}]}.$$

The exponentiation acts on G , because G represents composition of the generator on Pauli coefficient vectors. In particular, $\chi \neq e^{\tau G}$: χ uses the different left-right operator basis $P\rho Q$. As a check, for an ideal X gate, $R_X = \text{diag}(1, 1, -1, -1)$ and the conversion gives $\chi_{XX} = 1$ with all other entries zero.

Complete positivity implies $\chi \succeq 0$. If Kraus operators are desired, factor

$$\chi = V\Lambda V^\dagger.$$

Each eigenpair gives a Kraus operator

$$K_r = \sqrt{\lambda_r} \sum_{P \in \{I, X, Y, Z\}} V_{Pr} P,$$

and therefore

$$\mathcal{E}_X(\rho) = \sum_r K_r \rho K_r^\dagger.$$

3.1.3.2 Pauli twirling

Pauli twirling projects the channel onto the Pauli-diagonal subspace:

$$\Pi_P(\mathcal{E}) = \frac{1}{4} \sum_{R \in \{I, X, Y, Z\}} \mathcal{R}^\dagger \circ \mathcal{E} \circ \mathcal{R}, \quad \mathcal{R}(\rho) = R\rho R.$$

In the process matrix this projection is simply

$$\chi_{PQ} \mapsto \delta_{PQ} \chi_{PP}, \quad \Pi_P^2 = \Pi_P.$$

The twirled channel is therefore

$$\mathcal{E}_{PT}(\rho) = \sum_{P \in \{I, X, Y, Z\}} p_P P\rho P, \quad p_P = \chi_{PP}.$$

The diagonal entries of χ now have a direct physical interpretation: p_I is the probability of no Pauli change, while p_X, p_Y, p_Z are the effective probabilities of the corresponding Pauli operations. These are the probabilities—not the diagonal entries of the transfer matrix R_X , which are Bloch-axis contraction factors. For the complete noisy gate, the dominant Pauli may be the intended X operation itself; to interpret error probabilities, first express the channel in the ideal-gate frame. Twirling discards the off-diagonal interference terms, coherent accumulation, and non-unital information.

3.2 Exercises

3.2.1 Easy

A transmon has $T_1 = 80 \mu\text{s}$ and $T_2 = 60 \mu\text{s}$.

1. Compute T_ϕ from $T_2^{-1} = (2T_1)^{-1} + T_\phi^{-1}$.
2. Check that $T_2 \leq 2T_1$.
3. State the long-time density matrix for the zero-temperature model containing both relaxation and dephasing.

3.2.2 Medium

Consider an X gate of duration $\tau = 40 \text{ ns}$ on a transmon with $T_1 = 80 \mu\text{s}$.

1. Set $\Omega = \pi/\tau$ and $\kappa = 1/T_1$, and construct the Pauli-basis generator G from Section 1.1.3.
2. Use a matrix exponential to compute $R_X = e^{\tau G}$.
3. Verify that setting $\kappa = 0$ gives $\text{diag}(1, 1, -1, -1)$.
4. Identify the matrix element encoding non-unital drift and explain which information Pauli twirling removes.

4 Lecture 4 — Pauli errors and detecting regions in the repetition circuit

4.1 Insert Pauli errors into the circuit

Return to the repetition-code circuit from Lecture 2. Focus first on the Z_0Z_1 check and repeat its extraction circuit in consecutive cycles. Each cycle resets a check ancilla, couples both data qubits into it with CNOTs, and measures the ancilla in the Z basis.

A circuit-level Pauli error is inserted at a **location**: a qubit at a particular time between operations. Its effect depends on every gate that comes after that location. For a CNOT with control c and target t ,

$$X_c \mapsto X_c X_t, \quad Z_c \mapsto Z_c,$$

$$X_t \mapsto X_t, \quad Z_t \mapsto Z_c Z_t.$$

For example, an X error on a data qubit acting as the CNOT control is copied to the check ancilla. The data error persists, while the copied ancilla error can flip the subsequent Z -basis measurement. Moving the same error to a location after that CNOT changes its downstream effects. Error propagation is therefore a property of the whole space-time circuit, not only of the static code checks.

4.2 Detectors and detecting regions

Let m_t and m_{t+1} be consecutive measurements of the same parity check. In noiseless execution they agree, so

$$d_t = m_t \oplus m_{t+1}$$

is a detector. A nonzero value is a **detection event**.

A **detecting region** answers the inverse question: at which circuit locations could an error change this detector? It is a set of circuit locations, each labeled by a Pauli type. An inserted Pauli error flips the detector exactly when it anticommutes with the region's Pauli label at that location. This representation follows the detecting-region language introduced by McEwen, Bacon, and Gidney (McEwen et al. 2023, especially Figs. 2–3).

Focus first on panel (d) of Figure 3. The blue region is Z -type throughout and spans two consecutive measurements of a bit-flip repetition-code check. Therefore:

- an inserted X or Y error crossing the blue region flips the detector;
- an inserted Z error commutes with the region and does not flip this bit-flip-code detector;
- reset operations start parts of the region;
- included measurements terminate parts of the region;
- CNOT propagation expands and contracts the region across the data and check wires.

The region is another way to draw the stabilizer flow. Instead of propagating every possible error forward, propagate the detector's Pauli sensitivity backward from its measurements. The resulting colored region shows all error locations to which the detector is sensitive.

4.3 Overlap explains correlated detection events

A useful set of detectors has overlapping detecting regions. In the three-qubit repetition code, neighboring checks Z_0Z_1 and Z_1Z_2 overlap on the middle data qubit. An X_1 error at a covered location anticommutes with both Z -type regions and flips both detectors.

Figure 4 gives the direct route from circuit propagation to an error graph:

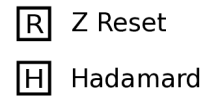
- each detecting region becomes a detector node;
- a Pauli fault covered by two regions becomes an edge between their nodes;
- a fault covered by one region becomes an edge from that detector to a boundary;
- a fault covered by several regions produces a correlated multi-detector signature.

This is why a single physical fault often creates a pair of detection events. The pair is not an assumption added by a decoder; it follows from the overlap of detector sensitivities in the circuit.

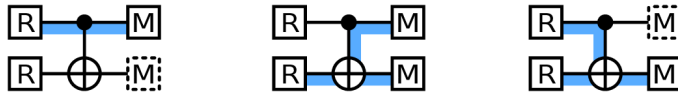
a) Trivial detecting regions



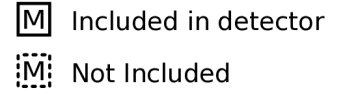
Legend



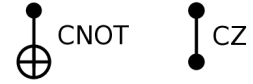
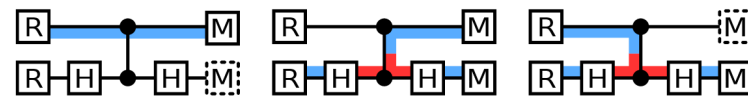
b) Simple detecting regions with CNOT



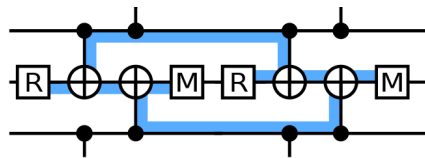
Z Measurement



c) Simple detecting regions with CZ



d) Bit-flip code circuit using CNOT



e) Phase-flip code circuit using CZ & H

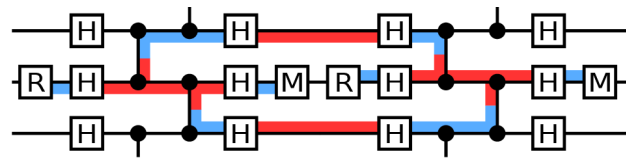
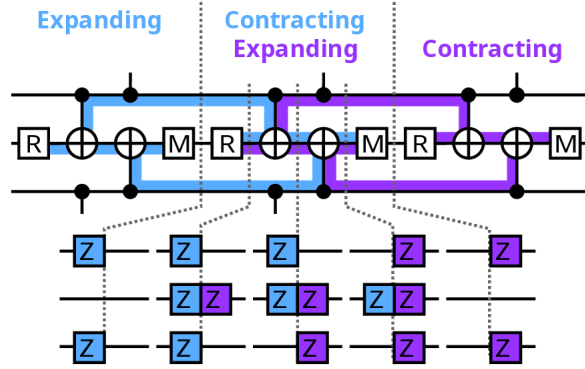


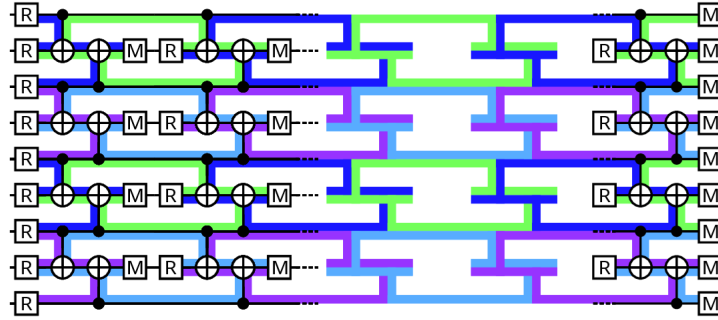
Figure 3: Examples of detecting regions. Blue segments are Z-type locations and red segments are X-type locations. Panels (d) and (e) show the bit-flip and phase-flip repetition-code circuits. Reproduced from Figure 2 of McEwen et al. (2023) under CC BY 4.0.

Bit-flip Repetition Code

a) Detecting regions during a cycle



b) All detecting regions for a distance-5 code run for 4 cycles



c) Error graph under an independent bit-flip error model

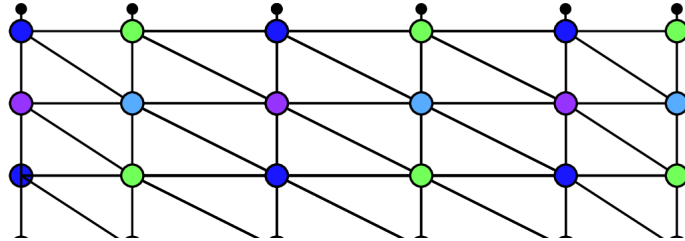


Figure 4: Overlapping detecting regions in a bit-flip repetition code. Panel (a) shows neighboring regions expanding and contracting through repeated syndrome-extraction cycles; panel (b) shows all regions for a distance-five code; panel (c) gives the corresponding error graph. Reproduced from Figure 3 of McEwen et al. (2023) under CC BY 4.0.

4.4 From detecting regions to a detector error model

A detector error model (DEM) is a classical list of the detector and logical effects of every elementary fault mechanism in the noisy circuit. Detecting regions provide a direct way to construct this list.

Start with a circuit noise model that assigns a probability p_e to each elementary Pauli fault E_e at circuit location ℓ_e . For every fault:

1. Find all detecting regions that cover ℓ_e .
2. At that location, compare E_e with the Pauli label carried by each region.
3. Include detector D_i exactly when the fault anticommutes with region i .
4. Propagate the fault to the end of the circuit and determine whether it flips any tracked logical observable L_j .

Define the detector and logical signatures

$$\partial e = \{D_i : E_e \text{ anticommutes with detecting region } i \text{ at } \ell_e\},$$

$$\lambda e = \{L_j : E_e \text{ flips logical observable } j\}.$$

The complete elementary mechanism is therefore summarized by

$$e \mapsto (p_e, \partial e, \lambda e).$$

Stim writes this information as

```
error(p_e) D... L...
```

For example, a fault lying in the overlap of regions D_0 and D_1 , with no logical effect, becomes

```
error(p_e) D0 D1
```

A fault covered only by D_0 becomes

```
error(p_e) D0
```

which is represented as an edge from D_0 to a boundary. If the first fault also crosses a logical observable, its instruction is

```
error(p_e) D0 D1 L0
```

The detector targets are toggles, so simultaneous mechanisms combine by XOR. If occurrence bit x_e indicates whether mechanism e happened, then

$$d_i = \bigoplus_{e: D_i \in \partial e} x_e, \quad \ell_j = \bigoplus_{e: L_j \in \lambda e} x_e.$$

This explains the error graph in panel (c) of Figure 4. A fault covered by two regions creates an edge between two detector nodes. A fault covered by one region creates a boundary edge. A mechanism affecting more than two detectors is a hyperedge rather than an ordinary graph edge.

The essential conversion is

$$\begin{aligned} \text{Pauli fault at a circuit location} &\longrightarrow \text{anticommuting detecting regions} \\ &\longrightarrow (\partial e, \lambda e) \\ &\longrightarrow \text{error}(p_e) D \dots L \dots \end{aligned}$$

Stim performs this compilation automatically for a declared noisy stabilizer circuit (Gidney 2021; Stim contributors 2026). The resulting DEM contains the information a detector-based decoder needs, but not the microscopic circuit trajectory. Correlations, leakage, or non-Pauli noise absent from the circuit model cannot appear in the DEM.

The next lecture uses this detector error model as the input to a decoder.

4.5 Exercises

4.5.1 Easy

At one circuit location, detecting regions D_0 and D_1 both carry a Z-type label. For each inserted error X , Y , and Z , determine which detectors flip. Explain your answer using commutation or anticommutation.

4.5.2 Medium

A noisy circuit has three elementary mechanisms:

- e_1 occurs with probability 0.01, is covered by D_0, D_1 , and has no logical effect;
- e_2 occurs with probability 0.02, is covered only by D_1 , and flips L_0 ;
- e_3 occurs with probability 0.005 and is covered by D_0, D_1, D_2 .

Write the three DEM `error(...)` instructions. Classify each as an ordinary edge, boundary edge, logical-labeled mechanism, or hyperedge, as applicable.

5 Lecture 5 — Decoding as probabilistic inference

This lecture adapts the general decoding framework of Pancotti and Svore (2026), Sec. 2, to the detector-error-model language developed in Lecture 4.

5.1 Errors, detectors, and the parity-check matrix

Suppose the detector error model contains n_E elementary fault mechanisms and n_D detectors. Represent a possible fault configuration by a binary vector

$$\mathbf{e} = (e_1, \dots, e_{n_E})^T \in \text{GF}(2)^{n_E},$$

where $e_i = 1$ means that fault mechanism i occurred. Represent the observed detection events by

$$\mathbf{s} = (s_1, \dots, s_{n_D})^T \in \text{GF}(2)^{n_D}.$$

Define the parity-check matrix

$$H_{ai} = \begin{cases} 1, & \text{fault } e_i \text{ flips detector } D_a, \\ 0, & \text{otherwise.} \end{cases}$$

Because detector effects combine by XOR, compatible fault configurations obey

$$\boxed{H\mathbf{e} = \mathbf{s} \pmod{2}}.$$

For example, consider

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

The rows are detector constraints and the columns are elementary fault mechanisms. The same matrix is the adjacency matrix of a bipartite Tanner graph, shown in the left panel of Figure 5.

Decoding combines code constraints, a noise model, and logical observables

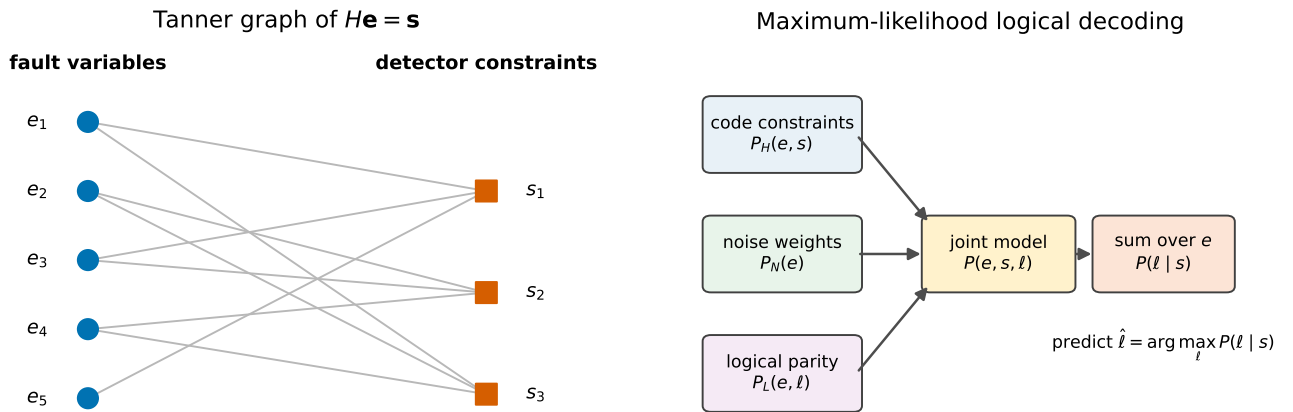


Figure 5: Left: Tanner-graph representation of the parity-check matrix. Right: maximum-likelihood logical decoding combines code constraints, noise weights, and logical parity before summing over compatible fault configurations. Adapted from the framework of Pancotti and Svore (2026), Sec. 2.

A syndrome generally does not identify a unique fault. For the example above,

$$\mathbf{s} = (1, 0, 1)^T$$

is produced by both

$$\mathbf{e}^{(A)} = (1, 0, 0, 0, 0)^T, \quad \mathbf{e}^{(B)} = (0, 1, 1, 0, 0)^T.$$

Decoding must therefore rank or sum over multiple compatible explanations.

5.2 Enforcing the syndrome constraints

For detector row a , define an indicator function

$$I_a(\mathbf{e}, s_a) = \begin{cases} 1, & \bigoplus_{i: H_{ai}=1} e_i = s_a, \\ 0, & \text{otherwise.} \end{cases}$$

The product

$$P_H(\mathbf{e}, \mathbf{s}) = \prod_{a=1}^{n_D} I_a(\mathbf{e}, s_a)$$

is one exactly when $H\mathbf{e} = \mathbf{s}$ and zero otherwise. It is an indicator of compatibility, not yet a normalized probability distribution. For the example matrix,

$$P_H = I_1(e_1, e_3, e_5, s_1) I_2(e_2, e_3, e_4, s_2) I_3(e_1, e_2, e_4, s_3).$$

5.3 Add the noise model

The parity constraints say which faults are possible, but not which are likely. That information is supplied by the noise model

$$P_N(\mathbf{e}).$$

For independent elementary faults with probabilities p_i ,

$$P_N(\mathbf{e}) = \prod_{i=1}^{n_E} p_i^{e_i} (1 - p_i)^{1-e_i}.$$

Correlated models use a more general joint distribution. Combining code constraints and noise gives the unnormalized weight

$$W(\mathbf{e}; \mathbf{s}) = P_H(\mathbf{e}, \mathbf{s}) P_N(\mathbf{e}).$$

The conditional probability of a fault configuration is

$$P(\mathbf{e} \mid \mathbf{s}) = \frac{W(\mathbf{e}; \mathbf{s})}{\sum_{\mathbf{e}'} W(\mathbf{e}'; \mathbf{s})}.$$

A most-likely-error decoder selects

$$\mathbf{e}_{\text{MLE}}(\mathbf{s}) = \arg \max_{\mathbf{e}: H\mathbf{e}=\mathbf{s}} P_N(\mathbf{e}).$$

5.4 Add logical observables

The decoder does not need to identify the microscopic fault. It needs to infer its effect on the encoded information. Let L be a binary matrix whose rows specify the tracked logical observables. The logical-flip vector is

$$\ell = Le \pmod{2}.$$

As for the detectors, define a logical indicator

$$P_L(\mathbf{e}, \ell) = \prod_b \mathbb{1} \left[\bigoplus_{i: L_{bi}=1} e_i = \ell_b \right].$$

The three ingredients combine into

$$W(\mathbf{e}, \mathbf{s}, \ell) = P_H(\mathbf{e}, \mathbf{s}) P_N(\mathbf{e}) P_L(\mathbf{e}, \ell).$$

To obtain the probability of a logical class, sum over every compatible fault configuration in that class:

$$Z(\mathbf{s}, \ell) = \sum_{\mathbf{e}} W(\mathbf{e}, \mathbf{s}, \ell),$$

$$P(\ell \mid \mathbf{s}) = \frac{Z(\mathbf{s}, \ell)}{\sum_{\ell'} Z(\mathbf{s}, \ell')}.$$

Maximum-likelihood logical decoding predicts

$$\hat{\ell}(\mathbf{s}) = \arg \max_{\ell} P(\ell \mid \mathbf{s}).$$

For one logical observable, this reduces to comparing $P(\ell = 1 \mid \mathbf{s})$ with $P(\ell = 0 \mid \mathbf{s})$.

5.5 Most-likely error versus most-likely logical class

These are different inference tasks:

- **Most-likely error:** choose one configuration $\mathbf{e}_{\text{MLE}}$ with the largest individual probability.
- **Maximum-likelihood logical decoding:** sum the probabilities of **all** compatible configurations in each logical class and choose the most probable class.

A logical class containing many moderately likely errors can outweigh a class containing the single most likely error. Degenerate quantum codes make this distinction especially important. Recovering the exact physical fault is unnecessary; predicting the correct logical class is sufficient.

5.6 Logical error rate

Given a dataset of N memory-experiment shots

$$\mathcal{D} = \{(\mathbf{s}^{(r)}, \mathbf{y}^{(r)})\}_{r=1}^N,$$

where $\mathbf{y}^{(r)}$ is the observed logical-flip vector, define

$$\text{LER} = \frac{1}{N} \sum_{r=1}^N \mathbb{1} [\hat{\ell}(\mathbf{s}^{(r)}) \neq \mathbf{y}^{(r)}].$$

For a single binary logical observable, the indicator is equivalently the squared difference between predicted and observed bits. Even the exact maximum-likelihood decoder can have a nonzero logical error rate because the same syndrome may be compatible with different logical classes. The noise model determines their relative posterior probabilities.

Summary: decoding combines the code constraints H , noise model P_N , and logical map L to compute $P(\ell \mid \mathbf{s})$.

5.7 Exercises

5.7.1 Easy

For

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

compute $\mathbf{s} = H\mathbf{e}$ for $\mathbf{e} = (1, 0, 0, 0, 0)^T$. Find a second fault vector with the same syndrome and verify it explicitly.

5.7.2 Medium

For one observed syndrome, suppose the compatible faults have relative weights

fault	logical class	weight
e_1	0	0.30
e_2	1	0.24
e_3	1	0.23
e_4	0	0.08

1. Which fault is selected by a most-likely-error decoder?
2. Which logical class is selected by maximum-likelihood logical decoding?
3. Explain why the two answers differ and why summing over a logical class is the correct objective for logical-observable prediction.

6 Lecture 6 — Hands-on QEC with Stim and PyMatching

This lecture is a guided coding lab. Students will build the repetition-code memory circuit used throughout the course, add circuit-level noise, compile its detector error model, sample detection events, and decode them.

6.1 Set up the notebook

Install and import the required CPU-only packages:

```
python -m pip install stim pymatching numpy
```

```
import numpy as np
import pymatching
import stim
```

Check that the imports work:

```
print("Stim:", stim.__version__)
print("PyMatching:", pymatching.__version__)
```

6.2 Build an ideal repetition-code memory

Use data qubits q_0, q_1, q_2 and check ancillas q_3, q_4 . The two ancillas measure Z_0Z_1 and Z_1Z_2 . The memory starts in the logical state $|0_L\rangle = |000\rangle$.

Enter the following function one block at a time and inspect the circuit after each stage:

```
def build_repetition_memory(
    rounds: int,
    p_data: float = 0.0,
    p_meas: float = 0.0,
    p_gate: float = 0.0,
) -> stim.Circuit:
    circuit = stim.Circuit()
    data = [0, 1, 2]
    checks = [3, 4]

    # Prepare  $|0_L\rangle = |000\rangle$ .
    circuit.append("R", data)

    for round_index in range(rounds):
        circuit.append("TICK")
        circuit.append("R", checks)

        # Data faults occurring before this syndrome-extraction round.
        if p_data:
            circuit.append("X_ERROR", data, p_data)

        # Measure  $Z_0Z_1$  and  $Z_1Z_2$  using ancilla targets.
        circuit.append("CX", [0, 3, 1, 3, 1, 4, 2, 4])

        # Optional two-qubit gate noise.
        if p_gate:
            circuit.append(
                "DEPOLARIZE2",
                [0, 3, 1, 3, 1, 4, 2, 4],
                p_gate,
            )

        # Flip an ancilla immediately before measurement.
        if p_meas:
            circuit.append("X_ERROR", checks, p_meas)
        circuit.append("M", checks)
```

```

# The first round is compared with the known initial syndrome 00.
if round_index == 0:
    circuit.append("DETECTOR", [stim.target_rec(-2)])
    circuit.append("DETECTOR", [stim.target_rec(-1)])
else:
    # Current measurements are rec[-2], rec[-1].
    # Previous measurements are rec[-4], rec[-3].
    circuit.append(
        "DETECTOR",
        [stim.target_rec(-2), stim.target_rec(-4)],
    )
    circuit.append(
        "DETECTOR",
        [stim.target_rec(-1), stim.target_rec(-3)],
    )

# Destructive final data measurement closes the temporal boundaries.
circuit.append("TICK")
if p_meas:
    circuit.append("X_ERROR", data, p_meas)
circuit.append("M", data)

# Final Z0 Z1 parity compared with the last q3 result.
circuit.append(
    "DETECTOR",
    [
        stim.target_rec(-3),
        stim.target_rec(-2),
        stim.target_rec(-5),
    ],
)

# Final Z1 Z2 parity compared with the last q4 result.
circuit.append(
    "DETECTOR",
    [
        stim.target_rec(-2),
        stim.target_rec(-1),
        stim.target_rec(-4),
    ],
)

# Use final q0 as the logical-Z memory observable.
circuit.append(
    "OBSERVABLE_INCLUDE",
    [stim.target_rec(-3)],
    0,
)
return circuit

```

Build a three-round ideal memory:

```

ideal_circuit = build_repetition_memory(rounds=3)
print(ideal_circuit)
print("detectors:", ideal_circuit.num_detectors)
print("observables:", ideal_circuit.num_observables)

```

Expected checks:

```

assert ideal_circuit.num_detectors == 8
assert ideal_circuit.num_observables == 1

```

In a Jupyter notebook, display the circuit with

```
ideal_circuit.diagram("timeline-svg")
```

Identify the logical-state preparation, repeated check measurements, bulk detectors, final-boundary detectors, and logical observable.

6.3 Verify the ideal circuit

Sample the ideal circuit:

```
ideal_detector_samples, ideal_observables = (  
    ideal_circuit.compile_detector_sampler(seed=1).sample(  
        shots=100,  
        separate_observables=True,  
    )  
)  
  
assert not ideal_detector_samples.any()  
assert not ideal_observables.any()  
print("Ideal circuit check passed.")
```

If this fails, inspect the `rec[...]` offsets before adding noise.

6.4 Add noise

Build a noisy circuit with independent data and measurement faults:

```
noisy_circuit = build_repetition_memory(  
    rounds=3,  
    p_data=0.01,  
    p_meas=0.01,  
)
```

Sample a few shots and inspect the arrays:

```
detector_samples, actual_observables = (  
    noisy_circuit.compile_detector_sampler(seed=2).sample(  
        shots=10,  
        separate_observables=True,  
    )  
)  
  
print("detector sample shape:", detector_samples.shape)  
print("observable shape:", actual_observables.shape)  
print(detector_samples.astype(int))
```

Check that the shapes agree with the circuit metadata:

```
assert detector_samples.shape == (10, noisy_circuit.num_detectors)  
assert actual_observables.shape == (10, noisy_circuit.num_observables)
```

For one nonzero row, identify whether its event pattern looks space-like, time-like, or boundary-like.

6.5 Compile and inspect the detector error model

Compile the noisy circuit into a DEM:

```
dem = noisy_circuit.detector_error_model(decompose_errors=True)  
print(dem)
```

For several `error(...)` instructions, record:

1. the mechanism probability;
2. the detector IDs it flips;
3. whether it carries logical observable L0;
4. whether it is an ordinary edge, boundary edge, or correlated mechanism.

Check the interface dimensions:

```
assert dem.num_detectors == noisy_circuit.num_detectors
assert dem.num_observables == noisy_circuit.num_observables
```

6.6 Decode with PyMatching

Construct a matching decoder directly from the DEM:

```
matching = pymatching.Matching.from_detector_error_model(dem)
```

Generate a larger dataset and decode it in a batch:

```
shots = 10_000
detector_samples, actual_observables = (
    noisy_circuit.compile_detector_sampler(seed=3).sample(
        shots=shots,
        separate_observables=True,
    )
)

predicted_observables = matching.decode_batch(detector_samples)
```

A shot fails when any predicted logical-observable bit disagrees with the sampled logical flip:

```
failed = np.any(predicted_observables != actual_observables, axis=1)
failures = np.count_nonzero(failed)
p_logical = failures / shots

print("failures:", failures)
print("shots:", shots)
print("logical failure fraction:", p_logical)
```

Verify the final interface:

```
assert predicted_observables.shape == actual_observables.shape
assert 0.0 <= p_logical <= 1.0
```

6.7 Student experiments

Complete at least two of the following.

6.7.1 A. Sweep the physical error probability

Run

```
for p in [0.001, 0.003, 0.01, 0.03]:
    # Build, compile, sample, decode, and record p_logical.
    ...
```

Plot or tabulate physical error probability versus logical failure fraction. Use identical shot counts and explicit random seeds.

6.7.2 B. Separate data and measurement noise

Compare:

1. $p_{\text{data}} > 0, p_{\text{meas}} = 0$;
2. $p_{\text{data}} = 0, p_{\text{meas}} > 0$;
3. both nonzero.

Explain how the detector-event geometry changes.

6.7.3 C. Add noisy two-qubit gates

Set $p_{\text{gate}} > 0$ and inspect how the DEM changes. Determine whether new correlated or logical-labeled mechanisms appear. Remember that depolarizing noise includes phase errors that the bit-flip repetition code does not

protect against.

6.7.4 D. Introduce decoder-model mismatch

Sample from a circuit built with one error probability, but construct the PyMatching decoder from a DEM built with a different assumed probability. Keep the circuit structure and detector ordering identical. Measure how the logical failure fraction changes.

6.8 Lab report

For each experiment, record:

- the complete circuit/noise parameters;
- number of rounds and shots;
- random seed;
- detector and observable array shapes;
- representative DEM instructions;
- number of logical failures and logical failure fraction;
- one claim supported by the experiment;
- one conclusion that the experiment does **not** establish.

A small finite-shot sweep does not establish a threshold or validate the noise model against hardware. The goal is to understand and verify the complete pipeline:

Stim circuit $\longrightarrow$ noisy samples $\longrightarrow$ DEM $\longrightarrow$ PyMatching $\longrightarrow$ logical score

6.9 Exercises

6.9.1 Easy

Run the lab circuit with `rounds=3` in two cases:

1. `p_data=p_meas=p_gate=0`;
2. `p_data=p_meas=0.01, p_gate=0`.

For each case, report the detector-array shape, observable-array shape, mean detector-event fraction, and logical failure fraction over 1,000 shots. Verify that the ideal case has no events or logical failures.

6.9.2 Medium

Create decoder-model mismatch:

1. Sample 20,000 shots from a circuit with `p_data=p_meas=0.02`.
2. Decode the same samples using DEMs constructed with assumed probabilities 0.005, 0.02, and 0.05.
3. Keep the circuit structure, detector order, and random seed fixed.
4. Compare logical failure fractions and explain why the correctly weighted DEM should perform best, while noting the role of finite-shot fluctuations.

7 Compact misconception index

1. T_2 is a time constant, not a Z -error probability.
2. Trace preserving does not imply unital; amplitude damping is non-unital.
3. Pauli twirling is a mathematical modeling operation; randomized compiling is a physical protocol.
4. The repetition encoding does not clone an unknown state.
5. A syndrome constrains an equivalence class; it need not identify a fault.
6. A detector is a parity of records, not a hardware sensor or raw syndrome.
7. Detector coordinates are metadata, not circuit operations.
8. A DEM is neither a quantum circuit nor a correction list.
9. `decompose_errors=True` does not make physical mechanisms independent.
10. An ancilla fault propagates according to CNOT role and remaining schedule; fault type alone is insufficient.
11. A decoder need not recover the actual physical fault.
12. A lower detector-event rate does not automatically imply a lower logical failure rate.
13. Zero observed failures do not prove zero failure probability.
14. More shots reduce statistical uncertainty, not modeling bias.
15. A small distance/noise sweep is not a threshold calculation.

8 References

- Devoret, Michel H. 1997. “Quantum Fluctuations in Electrical Circuits.” In *Quantum Fluctuations*, edited by S. Reynaud, E. Giacobino, and J. Zinn-Justin. Elsevier. https://boulderschool.yale.edu/sites/default/files/files/devoret_quantum_fluct_les_houches.pdf.
- Gidney, Craig. 2021. “Stim: A Fast Stabilizer Circuit Simulator.” *Quantum* 5: 497. <https://doi.org/10.22331/q-2021-07-06-497>.
- Gottesman, Daniel. 1997. “Stabilizer Codes and Quantum Error Correction.” *arXiv:quant-Ph/9705052*, ahead of print. <https://doi.org/10.48550/arXiv.quant-ph/9705052>.
- McEwen, Matthew, Dave Bacon, and Craig Gidney. 2023. “Relaxing Hardware Requirements for Surface Code Circuits Using Time-Dynamics.” *Quantum* 7: 1172. <https://doi.org/10.22331/q-2023-11-07-1172>.
- Pancotti, Nicola, and Krysta Svore. 2026. “Exact Learning of Quantum Noise with Tensor Networks.” Unpublished manuscript.
- Roffe, Joschka. 2019. “Quantum Error Correction: An Introductory Guide.” *Contemporary Physics* 60 (3): 226–45. <https://doi.org/10.1080/00107514.2019.1667078>.
- Stim contributors. 2026. *Stim Documentation and Gate Reference*. <https://github.com/quantumlib/Stim/wiki>.