Master QLMN
Experimental quantum computing

Pascal Scholl
pascal.scholl@pasqal.com

Lecture 8 - Introduction to fault-tolerant quantum computing

February 18th 2026

1 Introduction

In the previous lecture, we describe the methods for encoding redundantly the quantum information over many physical qubits into an unit called a *logical qubit* thanks to quantum error correcting (QEC) codes. This lecture describes how to use these logical qubits to perform computations, which is called fault-tolerant quantum computing (FTQC).

FTQC promises that, even though the physical qubit performances will never reach the requirements to perform large-scale algorithms such as Shor's, the computation is made in a way such that it *tolerates* the presence of these errors and therefore one obtains the correct output of the implemented computation.

A computation has two main components: qubits, and operations between these qubits. We will first focus on fault-tolerant qubits, and provide details on key aspects discussed in the previous lecture. We then detail the various methods for performing fault-tolerant operations between logical qubits.

We will ground this lecture on the example of the surface code. Note that all of the general conclusion we will reach here apply to most QEC codes.

2 Quiz: Implementation of the surface code below threshold

Let's begin by discussing the paper from Google Quantum AI team untitled 'Quantum error correction below the surface code threshold" *Nature* (2025). (https://www.nature.com/articles/s41586-024-08449-y). Here are 5 questions to check our understanding.

Q1: Code Implementation

What type of errors are corrected with the error-correcting codes used in this work?
A. Bit-flip errors.
B. Phase-flip errors.
C. Bit-flip and phase-flip errors.
D. Bit-flip, phase-flip and leakage errors.

Q2: Improvement of performances thanks to QEC

According to the experimental results, what is the improvement that was obtained using logical qubits with respect to the physical qubits?
A. An improvement of gate fidelities by a factor 2.
B. An improvement of qubit lifetime by a factor 2.
C. An improvement of cycle speed by a factor 2.
D. An improvement of the logical fidelity when increasing code distance by a factor 2.

Q3: Number of correction cycles

In the results shown in the paper, what is the maximum number of error correction cycles in a row that are implemented?
A. 250 cycles.
B. 1000 cycles.
C. 10^6 cycles.
D. 10^{11} cycles.

Q4: Error Budget

What physical process is limiting the most the obtained results?
A. Physical qubit lifetime.
B. Two qubit gate.
C. Single qubit gate.
D. Measurement.

Q5: Real-time decoding

Why is it required to decode the errors faster than the QEC cycle time?
A. Otherwise the classical computer cannot send the gates instructions because its memory is full.
B. Some gates require that the errors are known before applying the gate.
C. We need to correct for the errors in between each rounds to apply the adequate correction.
D. It is not required, the errors can be decoded after the computation.

3 Fault-tolerant qubits

Fault-tolerant qubits means that there are mechanisms in the quantum processor which mitigate the errors spontaneously arising on the qubits, such as the bit-flip errors (associated to the T_1 time) and the phase-flip errors (associated to the T_2 time). Fault-tolerant qubits are engineered using quantum error correcting codes, such as the surface code described in the previous lecture. These quantum error correcting codes are introducing a new object, called *logical qubit* (discussed in the previous lecture). The computational states of these logical qubits $|0\rangle_L$ and $|1\rangle_L$ are delocalized over many entangled physical qubits. This last point is what enables the fault-tolerance: even if an error occur on one of the physical qubit, it can be detected and corrected by the quantum error correcting code, and therefore the error does not propagate on the logical qubit.

In practice, achieving fault-tolerance at the qubit level means that the T_1 and T_2 times are better using the logical qubits rather than the physical ones. This has been experimentally demonstrated in the publication 'Quantum error correction below the surface code threshold" discussed above.

As a reminder, a quantum error correcting code is denoted as $[[n, k, d]]$ where:

- n is the number of physical qubits in the code.
- k is the number of logical qubits that the code contains. Its value for usual codes is 1, although intensive research is currently being conducted to discover *high-encoding rate* error-correcting codes which can carry more than one logical qubit.
- d is the distance of the code. As discussed in the previous lecture, d quantifies the protection that the code provides: a distance-d code can correct up to $(d-1)/2$ errors.

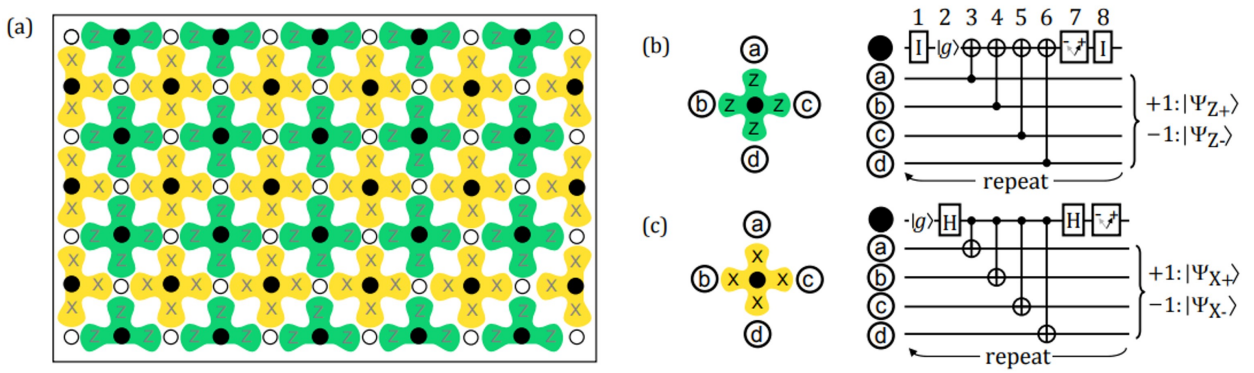

Figure 1: **Surface code implementation**: A two-dimensional array implementation of the surface code. Data qubits are open circles, measurement qubits are filled circles, with measure-Z ancillary qubits colored green (dark) and measure-X ancillary qubits colored orange (light). Away from the boundaries, each data qubit contacts four ancillary qubits, and each measure qubit contacts four ancillary qubits; the ancillary qubits perform four-terminal measurements. On the boundaries, the ancillary qubits contact only three data qubits and perform three-terminal measurements, and the data qubits contact either two or three ancillary qubits. (b) Geometric sequence of operations (left), and quantum circuit (right) for one surface code cycle for a measure-Z qubit, which stabilizes $Z^a Z^b Z^c Z^d$. (c) Geometry and quantum circuit for a measure-X qubit, which stabilizes $X^a X^b X^c X^d$. The two identity I operators for the measure-Z process, which are performed by simply waiting, ensure that the timing on the measure-X qubit matches that of the measure-Z qubit, the former undergoing two Hadamard H operations. The identity operators come at the beginning and end of the sequence, reducing the impact of any errors during these steps. Adapted from [2].

As an example, the surface code can be written as $[[d^2, 1, d]]$, meaning that it encodes one logical qubit with distance d over d^2 physical qubits.

In this section, we describe the requirements to reach a regime for which using error-correction enables fault-tolerance. We will base our discussion on the surface code, but the general arguments remain valid for all types of QEC codes.

3.1 QEC code implementation - example of the surface code

As a reminder of the previous lecture, we here describe the surface code and its implementation. By arranging qubits on a 2D lattice, we can perform all parity checks using only nearest-neighbor interactions involving 4 data qubits. In the surface code, we define two types of plaquettes:

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

In nowadays quantum processors, the surface code is implemented as presented in Figure 1: the ancillary qubits are placed in between the data qubits on a 2D plane, and each ancillary qubit has 4 neighboring data qubits.

3.1.1 Logical codespace and logical states

The logical **codespace** ($\text{Span}\{|0\rangle_L, |1\rangle_L\}$) is defined as the $+1$ simultaneous eigenspace of **all** stabilizers. For the surface code, the error correction is performed using $d^2 - 1$ *ancillary qubits*, meaning that the total number of qubit to realize the surface code is $2d^2 - 1$. The ancillary qubits perform the stabilizer measurements described above.

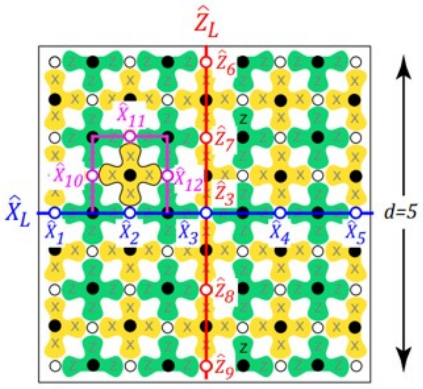

Figure 2: **Logical operators**: The logical operators X_L and Z_L are defined as the product of data physical qubits operators along given lines. The vertical lines provide X_L, while the horizontal lines provide Z_L. Adapted from [2].

Starting from a "vacuum" of physical $|0\rangle$ state, the system is intialized in the logical state $|0\rangle_L$ by performing stabilizer measurement. 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.

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.

3.1.2 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 $|1\rangle_L$ but from the logical qubit Pauli operators. The logical X_L and Z_L operators, introduced in the previous lecture, are strings of physical X (Z) operators stretching across the lattice, see fig. 2 for an example on a $d = 5$ surface code. In the following, we will often consider a $d = 3$ surface code. We will use:

$$X_L = X^1X^4X^7$$
$$Z_L = Z^1Z^2Z^3$$

but for example $X_L = X^2X^5X^8$ is also valid: any string of operators work along that corresponding direction work.

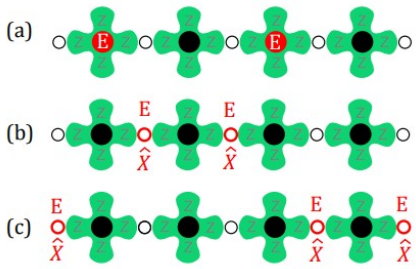

Figure 3: **Ambiguity in error detection leading to logical error**: An example where two measure-Z qubits report errors in a single row of a 2D array, marked by "E"s. This error report could be generated by (b) two X errors appearing in the same surface code cycle on the 2nd and 3rd data qubit from the left, or (c) three X errors appearing in the other three data qubits in the row. Adapted from [2].

3.2 Error correction cycle

The error correction cycle is the set of operations which are required in order to extract the stabilizer value (± 1) from the code. Even though we here focus on the surface code, most of the codes possess are built in the same way as what is described here. This set of operation is repeated many times during a computation in order to correct errors that build-up during the computation. In the case of the surface code, the error correction cycle is composed of 8 steps, which slightly differ between X and Z stabilizer measurements but can be decomposed as:

- Initialisation of the ancillary qubits into one defined state, such as $|0\rangle$.
- Single qubit gates (here H).
- Two qubit gates (here CNOT gates) between the ancillary qubit and the 4 neighboring data qubits.
- Measurement of the ancillary qubits.

The typical duration of an error correction cycle therefore depends on the speed at which the processor is able to do these operations. In nowadays platforms, the typical durations are:

- **Superconducting circuits:** In the range of $\sim 10\,\mu$s
- **Atomic qubits:** In the range of ~ 1 ms

This very large difference is currently seen as one important bottleneck of the atomic qubits platform, as longer correction cycle times means slower computations.

Since all of these operations are exposed to errors themselves, when performing the stabilizer measurement in order to check for errors, there is a probability that the error correction itself induces errors. In the following, we will assume for simplicity that each of these 8 steps has an error probability p, and that the total error probability over the full cycle is $p_{\text{cycle}} = 8p$. This per-step error p has a huge impact on the logical error P_L, which we describe next.

3.3 Logical error and threshold behavior

We here derive the logical error P_L as a function of p for a toy model on the surface code, which provides a correct intuition on the behavior of P_L. Assuming a surface code of distance d (with d an odd integer), a logical error happens when at least one logical operator X_L or Z_L flips due to physical errors. In practice, this happens because there is an inherent ambiguity in the error detection: if k stabilizers show errors, it is unclear if it comes from k errors, or from $d-k$ errors. An example is provided in Figure **??**. The code detector will always assume that there are k errors, because we assume that the error probability is small. This means that when more than $(d-1)/2$ errors happen along one line of qubits (there are $2d$ lines), a logical error happens.

Logical errors therefore happen when the number of physical errors k is $k \geq (d+1)/2$ (remember that d is odd). Assuming that each qubit has an error probability p_{cycle}, the probability P_k that k qubits have an error is:

$$P_k = \binom{d}{k} p_{\text{cycle}}^k (1 - p_{\text{cycle}})^{d-k} \tag{1}$$

where $\binom{d}{k}$ stands for all the possible positions of the errors. From these considerations, the logical error is expressed as:

$$P_L = 2d \sum_{k=(d+1)/2}^{d} P_k = 2d \sum_{k=(d+1)/2}^{d} \binom{d}{k} p_{\text{cycle}}^k (1 - p_{\text{cycle}})^{d-k} \tag{2}$$

where the $2d$ factor here represents the possible qubits lines of size d which can trigger a logical error. We assume next that $p_{\text{cycle}} \ll 1$, meaning that (1) the first term of the above sum is the major contributor to P_L, and (2) $(1 - p_{\text{cycle}})^{d-k} \simeq 1$. This gives:

$$P_L \simeq 2d \binom{d}{\frac{d+1}{2}} p_{\text{cycle}}^{(d+1)/2} = 2d \frac{d!}{((d+1)/2)!((d-1)/2)!} p_{\text{cycle}}^{(d+1)/2} \tag{3}$$

We assume that $d \gg 1$, and simplify the above equation using Sterling's formula: $d! \simeq \sqrt{2\pi d}(d/e)^d$. We further assume that $((d-1)/2)! \simeq (d/2)!$ and $((d+1)/2)! \simeq (d/2)!$. We thus obtain:

$$\begin{aligned} P_L &\simeq 2d \frac{\sqrt{2\pi d}(d/e)^d}{2\pi(d/2)(d/2e)^d} p_{\text{cycle}}^{(d+1)/2} \\ &\simeq 2d \frac{2^{d+1}}{\sqrt{2\pi d}} p_{\text{cycle}}^{(d+1)/2} \\ P_L &\simeq \sqrt{\frac{2d}{\pi}} 2^{d+1} p_{\text{cycle}}^{(d+1)/2} \end{aligned}$$

We can rewrite the above equation with $2^{d+1} = 4^{(d+1)/2}$:

$$P_L = \sqrt{\frac{2d}{\pi}} (4 p_{\text{cycle}})^{(d+1)/2} \tag{4}$$

As discussed previously, we assume here that $p_{\text{cycle}} = 8p$ where p is the per-step physical error rate. By setting:

$$\begin{aligned} p_{\text{th}} &= 1/32 \\ \text{and } A &= \sqrt{\frac{2d}{\pi}}, \end{aligned}$$

we obtain the general expression that describes the logical error rate of most QEC codes:

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

From this formula, we can see that:

- **When $p > p_{\text{th}}$**: The logical error rate actually increases with d, eventually saturating at $P_L = 50\%$, which represents total loss of information (random noise). We are in the "noise-dominated" regime.
- **When $p < p_{\text{th}}$**, we are in the "protection" regime. Increasing the distance d of the code suppresses the logical error rate **exponentially**. This is experimentally demonstrated in Figure. The logical error rate decreases exponentially with d. This means that there exist a distance d for which $P_L < p$, and thus the logical qubit is better than the physical qubits.

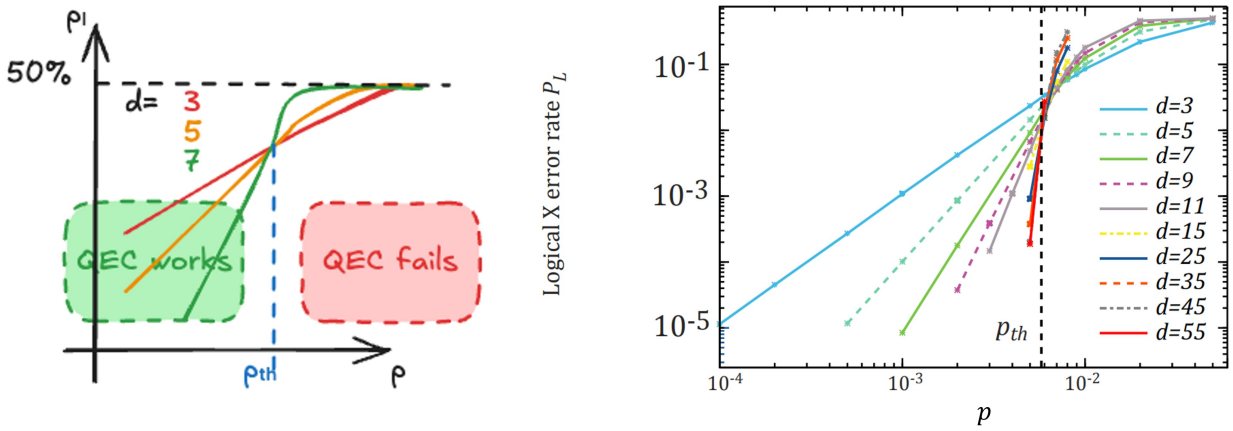

Figure 4: Threshold plot: logical error rate P_L vs physical error rate p for different code distances $d = 3$ (red), 5 (orange) and 7 (green). The curves cross at the threshold p_{th} (dashed blue line). Numerical simulations of surface code error rates assuming depolarizing errors, and how these error rates scale with the distance d. We obtain $p_{\text{th}} \simeq 0.57\%$. This threshold corresponds a cycle error rate of $p_{\text{cycle}} \simeq 4\%$. Adapted from [2].

The value of the threshold in practice depends on the exact physical noise structure, decoder, and QEC code. Its value is often in the range $p_{\text{th}} \sim 0.001 - 0.01$, a bit lower than the value obtained here of $\simeq 0.03$. This difference mainly comes from the fact that we did not take into account error propagation through time. For a given code, decoder and physical error model, the value of p_{th} can be assessed via simulations, by looking at the logical error as a function of p for various d, see Figure 4. The curves roughly cross for $p = p_{\text{th}}$.

This threshold value means that the physical error rates p must be below 0.01, meaning that the fidelity must be higher than 99%. The recent advances in the physical gate fidelities, well beyond 99%, enabled the implementation of QEC codes below threshold.

In our toy model, we obtained that the prefactor A also depends on d, which could look puzzling. We note that this is also true in practice, although the $\sqrt{d}$ scaling is not guaranteed for all types of QEC codes, decoders and physical error models. However, this depedence is always found to be polynomial in d, meaning that the *exponential suppression* of errors remains true. In practice, this means that for small values of d, deviations from pure exponential decrease of P_L as a function of d are observed.

3.4 Conclusion on fault-tolerance and experimental demonstrations

In this section, we have seen that even though the physical qubits are noisy with with an associated error probability p, by employing a quantum error correcting code we can get a logical error rate $P_L < p$. This requires that the physical error rate is below a certain threshold p_{th}, which arises from the fact that correcting errors is itself noisy. Obtaining $P_L < p$ means that the system is fault-tolerant: even though errors exist, the system is resilient to their presence.

Such fault-tolerance on the surface code was demonstrated for the first time by the Google quantum AI team in 'Quantum error correction below the surface code threshold" *Nature* (2025). $P_L < p$ is demonstrated experimentally, see Figure 5(a). For a $d = 3$ surface code, the logical error is larger than the physical error. However for a distance $d = 7$ surface code, $P_L < p$. The below-threshold behavior of the system is demonstrated in Figure 5(b): the logical error rate decreases with increasing d, meaning that $p < p_{\text{th}}$. The same type of results were recently obtained in the group of M. Lukin at Harvard on the neutral atom platform (Figure 5(c)). To our knowledge, below-threshold demonstration on the surface code was only demonstrated

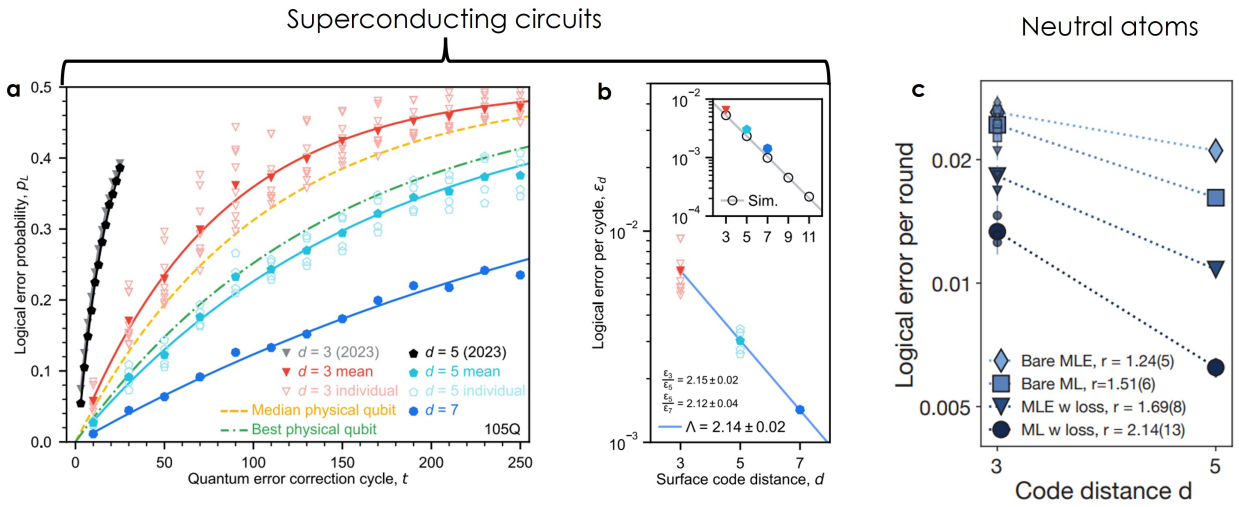

Figure 5: **Experimental demonstration of below-threshold on the surface code**: (a) Logical error as a function of the number of QEC cycles, for logical qubits encoded on the surface code of distances $d = 3, 5, 7$, and for physical qubits (best and medians). For $d = 5$, the logical qubit shows better performances than the best physical qubit, meaning that $P_L < p$. (b) Logical error as a function of the code distance. We see that the logical error decreases when increasing code distance, meaning that the system is below threshold $p < p_{\text{th}}$. (c) Same results on the neutral atom platform. Adapted from [1] and [3].

on these two platforms.

3.5 Resource estimates

We briefly comment on the requirements to be able to run large-scale algorithms such as Shor's. Such algorithms typically require fidelities in the range $P_L \sim 10^{-15}$. Assuming that $A = 0.1$, $p_{\text{th}} = 0.01$ and $p = 0.001$ (level of performance of the best current platforms), this means that we need a surface code with distance $d = 29$. As the total number of qubits is $\simeq 2d^2$, this means that one logical qubit is made of about 1700 qubits. Since large-scale algorithms typically require thousands of algorithmic qubits, we obtain that the required numbers of physical qubits is about 1 million.

Assuming that in the coming years the quantum processors becomes better and $p = 0.0001$, a $d = 15$ surface code would be enough, meaning that only 450 physical qubits per logical qubits would be required.

4 Fault-tolerant logical gates using transversal physical gates

Having described how we can engineer fault-tolerant qubits, we now turn to how to perform computations with these logical qubits, in a fault-tolerant manner. As we are now dealing with logical qubits rather than physical qubits, we need to understand how to compute with them. We will see that certain type of gates, which are very simple to implement on physical qubits, are non trivial to perform at the logical level. In order to perform universal computing on logical qubits, we will consider three type of operations:

1. Logical gates directly implemented by physical gates on the physical qubits, which we call *transversal operations*, and are the subject of this section
2. Logical gates induced by magic state teleportation, described in the next section.
3. Logical gates induced by lattice surgery (not described in this lecture)

Depending on the exact QEC code, other methods can allow implementing logical gates. Discovering all the ways to implement logical gates is an active field of research.

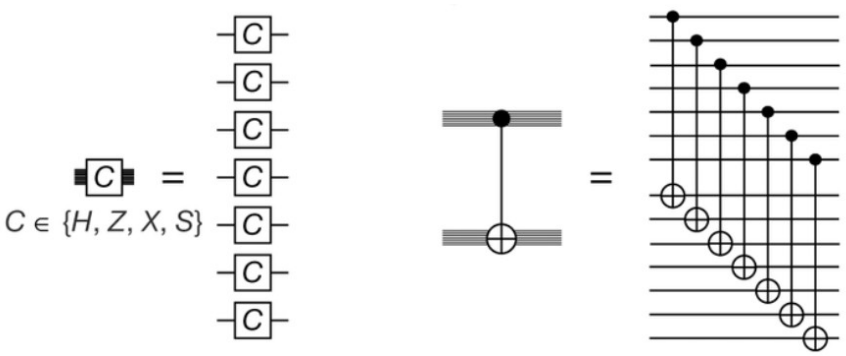

Figure 6: **Performing logical gates via the corresponding physical gates**: Logical single qubit gates such as H_L, Z_L, X_L, S_L can be implemented by performing these gates at the physical level on all the data qubits of the QEC code. Two-qubit logical gates such as the $CNOT_L$ gate can be performed between logical qubits by performing physical CNOTs between the data qubits of the two logical qubits.

4.1 Description

We here assume a logical gate U_L which acts on a single logical qubit in state $|\psi_L\rangle = \alpha\,|0_L\rangle + \beta\,|1_L\rangle$, with $U_L\,|\psi_L\rangle = |\psi'_L\rangle$. A transversal gate is defined as $U_L = \bigotimes_n U^n$ where U^n is a single qubit gate acting on the n^{th} physical qubit. This means that the logical gate is implemented by performing the physical gate on all the data qubits that encode the logical qubit. For example, in some QEC codes H, X or CNOT gates can be implemented in this way (see figure 6). This method is simple to execute and understand, but it has some constraints:

- The exact gates that can be implemented transversally depend on the considered QEC code.
- **not all gates can be implemented in this fashion**, meaning that universal computing with logical qubits cannot be achieved using only transversal operations.

4.2 Constraints on logical gates

In order to preserve the code integrity, the logical gates should only act within the code, meaning:

- $|\psi'_L\rangle$ should be a valid state of the QEC code.
- The logical operators after the application of U_L should remain valid.

We now detail these two constraints.

4.2.1 Codespace preservation

In order to preserve the structure of the underlying QEC code, the operation U_L should be built such that $|\psi'_L\rangle$ is a valid state of the code. This fact is called **preserving the codespace**. This means that $|\psi'_L\rangle$ must be an eigenstate of all the stabilizers S_i:

$$S_i\,|\psi'_L\rangle = |\psi'_L\rangle\,,\ \forall S_i \in \mathcal{S} \tag{5}$$

where $\mathcal{S}$ is the ensemble of stabilizers (for example $Z^1Z^2Z^4Z^5$ for the $d = 3$ surface code). We can rewrite (5) as:

$$S_iU_L\,|\psi_L\rangle = U_L\,|\psi_L\rangle$$
$$U_L^\dagger S_iU_L\,|\psi_L\rangle = |\psi_L\rangle$$

meaning that **the operator $U_L^\dagger S_iU_L\,\forall\,S_i$ must be a stabilizer of** $|\psi_L\rangle$. Note that $U_L^\dagger S_iU_L$ does not need to be equal to a given stabilizer, but instead can be a product of all stabilizers. This means that $U_L^\dagger S_iU_L$

takes the general form:

$$U_L^\dagger S_i U_L = \prod_j S_j^{a_j} = S_i' \tag{6}$$

where $a_j \in \{0, 1\}$.

Said in other words for our considered surface code, the condition is that U_L must map:

- X-checks $\rightarrow$ products of X-checks
- Z-checks $\rightarrow$ products of Z-checks

This condition is shared among most of the QEC codes, as most of them uses Pauli operators as stabilizers (for the surface code, it is X and Z).

4.2.2 Logical operators preservation

Now that we have established the constraint on preserving the codespace, another important point is that U_L also **preserves the logical operators** L, for example X_L and Z_L in the surface code. Under the action of U_L, the logical operators are modified as:

$$L' = U_L L U_L^\dagger \tag{7}$$

These new logical operators L' should still act correctly on the logical qubit, meaning that for any stabilizer S_i:

$$S_i L' \left|\psi_L'\right\rangle = \left|\psi_L'\right\rangle \ \forall\, S_i \tag{8}$$

We demonstrate that this is automatically the case if U_L preserves the codespace, adding no further constraints than the one described above. We rewrite $S_i L' \left|\psi_L'\right\rangle$ as:

$$\begin{aligned} S_i L' \left|\psi_L'\right\rangle &= S_i (U_L L U_L^\dagger)(U_L \left|\psi_L\right\rangle) \\ &= S_i U_L L \left|\psi_L\right\rangle \\ &= U_L (U_L^\dagger S U_L) L \left|\psi_L\right\rangle \end{aligned}$$

From equation (6), we know that $U_L^\dagger S_i U_L = \prod_j S_j^{a_j} = S_i'$ to fulfill the codespace preservation condition. Further, L commutes with S', meaning that $S'L = LS'$. We therefore obtain that:

$$\begin{aligned} S_i L' \left|\psi_L'\right\rangle &= U_L S_i' L \left|\psi_L\right\rangle \\ &= U_L L S_i' \left|\psi_L\right\rangle \\ &= U_L L \left|\psi_L\right\rangle \\ &= U_L L U_L^\dagger U_L \left|\psi_L\right\rangle \\ &= L' \left|\psi_L'\right\rangle \ \forall\, S_i \end{aligned}$$

We therefore obtain that L' is a valid logical operator.

4.3 Available gates using transversal operations

We have seen that U_L must fulfill the condition $U_L^\dagger S_i U_L = \prod_j S_j^{a_j} \ \forall\, S_i$. Since S_j are Pauli operators, $\prod_j S_j^{a_j}$ is a Pauli operator (the product of Pauli operators is a Pauli operator), meaning that $U_L^\dagger S_i U_L$ is a Pauli operator. Calling this Pauli operator $\mathcal{P}_i$, we have:

$$U_L^\dagger S_i U_L = \mathcal{P}_i, \ \mathcal{P}_i \in \{\pm X, \pm Y, \pm Z\} \tag{9}$$

which is exactly the definition of the **Clifford group**. The Clifford group possesses 24 operators, among which the four Pauli operators (I, X, Y, Z). The 24 gates can be generated by multiplying these Pauli operators with two other gates (which are called the generators of the Clifford group), H and S which are:

$$H_L = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad S_L = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

This consideration means that not all possible gates can be implemented (in particular non-Clifford gates), meaning that a universal gate set is not accessible via logical gates performed by physical gates. Note that this statement is valid for any QEC code. In order to perform non-Clifford gate, another method is required: magic state injection (described later).

The above considerations also apply to multi-qubit gates. A CNOT gate maps Pauli operators to Pauli operators, as well as a CCNOT gate. However, CCNOT and higher-order gates cannot be implemented *fault-tolerantly* in this way, because the error propagation between logical qubits is too large.

4.4 Example of logical gates engineering on the surface code

Having described the requirements for performing logical gates originating from physical gates, we turn to discuss how to implement them in practice. For single logical qubit operations, the logical gate U_L can be decomposed as the logical Pauli operators:

$$U_L = a_0 I_L + a_1 X_L + a_2 Y_L + a_3 Z_L \tag{10}$$

where the a_i coefficients are complex numbers with $|a_0|^2 + |a_1|^2 + |a_2|^2 + |a_3|^2 = 1$, and $Y_L = iX_L Z_L$. To illustrate how logical gates can be applied, we will consider three examples on the surface code: the X_L gate, the H_L gate, and the $CNOT_L$ gate.

4.4.1 X_L gate engineering

We start with the simple example of engineering an X_L gate. This gate is directly one of the logical operators, defined for example as $X_L = X^1 X^4 X^7$ for the $d = 3$ surface code. By construction, applying X_L preserves the codespace, so it is a valid gate. In order to perform such operation in practice, one can apply a physical X gate on qubits $1, 4$ and 7. This operation can be applied to any line of qubits (for example qubits 2, 5 and 8), and it will apply X_L. The arbitrary choice that the user has here nicely shows the strength of delocalizing a logical qubit over many physical qubits: applying physical gates on a subset of the physical qubits perform a logical operation on the full logical qubit.

We note that for most QEC codes, implementing a gate which consists of the logical operator themselves is performed in the same way as described here.

4.4.2 H_L gate engineering

We now detail how to implement a H_L gate. We have $H_L = (X_L + Z_L)/\sqrt{2}$. In order to find the physical operations we need to implement in order to perform H_L, we observe that $H_L^\dagger X_L H_L = Z_L$, and $H_L^\dagger Z_L H_L = X_L$. This means that, on our example of the $d = 3$ surface code, we want to implement a physical operation which transforms $X^1 X^4 X^7$ into $Z^1 Z^4 Z^7$, which corresponds to applying physical H gates. In a similar fashion as for X_L described previously, the idea would thus be to apply physical H gates on qubits 1, 4, 7 for X_L, and qubits 2 and 3 for Z_L (as qubit 1 is already considered for X_L).

However, if we apply the five H gates indicated above, for example the $Z^1 Z^2 Z^4 Z^5$ stabilizer measurement before-H then becomes a $X^1 X^2 X^4 Z^5$ stabilizer measurement after-H, which stabilizes the logical state outside of the codespace, and is thus not fault-tolerant. The solution to solve this issue is to *perform the H gate on all the physical data qubits*: that way, all the stabilizers are of the type $XXXX$ or $ZZZZ$, and since d is an odd number, X_L and Z_L are impacted as intended.

The impact of the H gates is to change the role of the stabilizers: X-stabilizers become Z-stabilizer. In order to solve this issue, one can perform for example qubit re-ordering: ancillary qubits that were performing X-stabilizers before-H can exchange their position with ancillary qubits that were performing Z-stabilizers. Such qubit repositioning is a feature which is currently unavailable in nowadays quantum processor for superconducting circuits. For atomic qubits, since the qubits are trapped, this repositioning is possible,

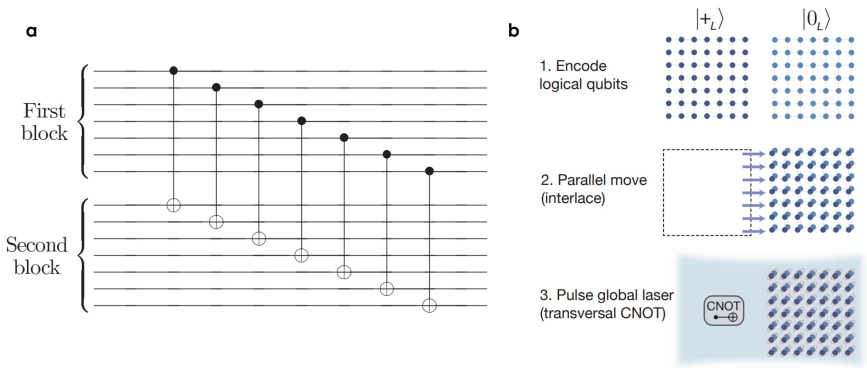

Figure 7: **Logical CNOT via transversal CNOTs**: (a) A logical CNOT can be implemented by performing physical CNOT between each physical qubits of the two logical qubits. (b) Implementation on the neutral atom processor. This operation is highly efficient thanks to parallel qubit mouvement and parallel CNOT gate implementation. Adapted from [4].

enabling the implementation of H_L in the described way.

This action of performing a logical gate (such as H_L here) by performing the equivalent operation at the physical level on all the data qubits (H gate on all the qubits) is called a **transversal gate**. Such transversal gates exist in most of the QEC codes.

4.4.3 CNOT gate engineering

We now consider two logical qubits, and engineer a logical CNOT gate $CNOT_L$, with a control logical qubit with logical operators X_C, Z_C and a target logical qubit with logical operators X_T, Z_T. In order to know what transformations at the physical level are required, we decompose $CNOT_L$ as

$$CNOT_L = \frac{I + Z_C}{2} \otimes I_T + \frac{I - Z_C}{2} \otimes X_T \tag{11}$$

by replacing $Z_C = Z_C^1 Z_C^2 Z_C^3$ and $X_T = X_T^1 X_T^4 X_T^7$, we obtain:

$$CNOT_L = \frac{I + Z_C^1 Z_C^2 Z_C^3}{2} \otimes I_T + \frac{I - Z_C^1 Z_C^2 Z_C^3}{2} \otimes X_T^1 X_T^4 X_T^7 \tag{12}$$

We simplify this expression by exploiting the property of Pauli operators:

$$\frac{I - Z_C^1 Z_C^2 Z_C^3}{2} \otimes X_T^1 X_T^4 X_T^7 = \prod_{i=1}^{3} \left(\frac{I - Z_C^i}{2} \otimes X_T^i \right) = \prod_{i=1}^{3} CNOT^i \tag{13}$$

Therefore, performing the $CNOT$ gates at the physical level performs a logical CNOT. In order to ensure that all stabilizers are impacted the same way by $CNOT_L$, the physical CNOT gates are performed on all the data qubits. A visualization is provided in Figure 7.

In practice, performing such CNOT gate requires the ability to entangle physical qubits from two logical qubits, which can be hard to perform. The neutral atom platform, thanks to the all-to-all connectivity of the platform. To our knowledge, it is the only platform who has demonstrated such entangling operation on a surface code. For platforms with lower connectivity (such as superconducting circuits), transversal CNOTs are not possible, and therefore one needs to perform a more complex operation (called lattice surgery) to implement CNOT gates.

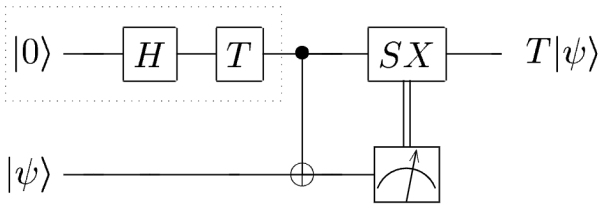

Figure 8: Circuit diagram for T-gate teleportation using a magic state and a corrective S gate.

5 Performing logical operations via teleportation

Another method for performing logical operations is to perform **Gate Teleportation**. This method allows implementing fault-tolerantly any gate, but has a much higher cost than the operations described above.

5.1 Overview of the protocol

Assuming we want to implement a unitary U_L on a target logical qubit such that $U_L |\psi_L\rangle = |\psi'_L\rangle$, the protocol utilizes another logical qubit, which we will call the control logical qubit. The protocol has two main steps:

1. **Magic state preparation**: On the control logical qubit, prepare a state, called *magic state*, which is $|\phi_L\rangle = U_L |+_L\rangle$
2. **Magic state teleportation**: Perform a logical CNOT gate between the two logical qubits, then readout the target logical qubit. Retro-act on the control logical qubit conditioned on the measured logical state.

The logical circuit associated to this protocol for the specific case of $U = T$ is described in figure 8. We below detail these two operations, starting with the last step as it is the easiest to understand.

5.2 Magic state teleportation

Assuming we are able to prepare the magic state $|\phi_L\rangle = U_L |+_L\rangle$ in the control logical qubit, the aim is to teleport this state to obtain $U_L |\psi\rangle$. $|\phi_L\rangle$ can be written as:

$$|\phi_L\rangle = U_L \frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{U_L |0\rangle + U_L |1\rangle}{\sqrt{2}} \tag{14}$$

After the CNOT operation, we obtain the entangled state $|\Psi_L\rangle$:

$$|\Psi_L\rangle = \frac{U_L |0_L\rangle \otimes |\psi_L\rangle + U_L |1\rangle \otimes X_L |\psi_L\rangle}{\sqrt{2}} \tag{15}$$

where X_L is the logical X operator acting on the target logical qubit in state $|\psi_L\rangle$. We can rewrite (15) considering that $|\psi_L\rangle = a |0_L\rangle + b |1_L\rangle$ as:

$$\begin{aligned} |\Psi_L\rangle &= \frac{U_L |0_L\rangle \otimes (a |0_L\rangle + b |1_L\rangle) + U_L |1_L\rangle \otimes (b |0_L\rangle + a |1_L\rangle)}{\sqrt{2}} \\ &= \frac{(aU_L |0_L\rangle + bU_L |1_L\rangle) \otimes |0_L\rangle + (bU_L |0_L\rangle + aU_L |1_L\rangle) \otimes |1_L\rangle}{\sqrt{2}} \end{aligned}$$

We then perform the measurement of the target logical qubit in the Z basis (the second qubit in the expression of $|\Psi_L\rangle$ above. There are two possible outputs

- The logical qubit is measured in $|0_L\rangle$, projecting the state into $|\Psi_L\rangle = (aU_L |0_L\rangle + bU_L |1_L\rangle) \otimes |0_L\rangle$

- The logical qubit is measured in $|1_L\rangle$, projecting the state into $|\Psi_L\rangle = (bU_L|0_L\rangle + aU_L|1_L\rangle) \otimes |1_L\rangle$.

We treat these two cases next.

5.2.1 Measurement outcome $|0_L\rangle$

If we assume that the measurement outcome was $|0_L\rangle$, we obtain

$$|\Psi_L\rangle = U_L(a|0_L\rangle + b|1_L\rangle) \otimes |0_L\rangle = U_L|\psi_L\rangle \otimes |0_L\rangle \tag{16}$$

which is what we wanted to do in the first place: the logical operation U_L is now applied to $|\psi_L\rangle$. The magic state is therefore correctly teleported.

5.2.2 Measurement outcome $|1_L\rangle$ and conditional operations

If the measurement outcome is $|1_L\rangle$, we obtain:

$$|\Psi_L\rangle = U_L(b|0_L\rangle + a|1_L\rangle) \otimes |1_L\rangle \tag{17}$$

In order to obtain the desired operation, we need to engineer an operation V_L which transform $(bU_L|0_L\rangle + aU_L|1_L\rangle)$ into $(aU_L|0_L\rangle + bU_L|1_L\rangle)$. This means we need to engineer V_L such that:

$$V_L U_L|0_L\rangle = U_L|1_L\rangle = U_L X_L|0_L\rangle$$
$$V_L U_L|1_L\rangle = U_L|0_L\rangle = U_L X_L|1_L\rangle$$

meaning that V_L must verify:

$$V_L U_L = U_L X_L \implies V_L = U_L X_L U_L^\dagger \tag{18}$$

As an example, assuming we want to teleport a T_L state, we obtain that the conditional operation to be performed after measurement is:

$$V_L = T_L X_L T_L^\dagger = e^{-i\pi/4} SX, \tag{19}$$

hence the gate in figure 8.

5.2.3 Fault-tolerance condition

Having detailed the magic state teleportation procedure, we now need to wonder if it is fault-tolerant. In order for it to be fault-tolerant, every involved operation must be fault-tolerant. The involved operations are:

- A CNOT gate, which can be performed in a fault-tolerant fashion via transversal CNOT (described earlier), or via lattice surgery (not described in this lecture).
- A logical measurement, which is fault-tolerant if the logical qubit is implemented in a fault-tolerant fashion.
- The conditioned operation V_L if the measurement outcome is $|1_L\rangle$, which depends on the implemented U_L.

In order to verify fault-tolerance, we therefore need to consider if V_L can be implemented in a fault-tolerant fashion. The simplest way is to perform this operation via transversal gates. However as discussed earlier, the fault-tolerant gates that can be performed transversally are the gates of the Clifford group, which verify equation (9). Since $U_L X_L U_L^\dagger = V_L =$, we have $U_L(\text{Pauli})U_L = \text{Clifford}$, meaning that U_L maps a Pauli operators into Clifford gates. Among the U_L which verify this relationship, we find non-Clifford operations such as $CCNOT$ and T.

With this study on fault-tolerance, we conclude that magic state teleportation do allow to perform more gates than what was possible using only transversal operations. However to preserve fault-tolerance, the easily achievable gates are T and CCZ, meaning that we cannot perform any U_L using this method.

5.3 Magic state preparation

We quickly comment on how to prepare the magic state $|T_L\rangle = T|+_L\rangle$, focusing on the example of a T state generation (since this is one of the gates that are possible to fault-tolerantly teleport eventually). Preparing such magic state is not possible fault-tolerantly, since it requires non-Clifford operations. However, here this is not a problem: the prepared $|T_L\rangle$ is not part of the computation directly: it is only used as an "auxiliary gadget" to be teleported onto the real logical qubit. As long as the $|T_L\rangle$ state preparation fidelity is high enough and the teleportation protocol is fault-tolerant, the whole procedure remains fault-tolerant.

The condition is thus to produce $|T_L\rangle$ with high enough fidelity. In order to do so, two main protocols exist:

- Magic state distillation
- Magic state cultivation

We note that magic state cultivation was invented in 2024, showing how active this field of research is. We here briefly describe magic state distillation, which is also described in figure 9.

5.3.1 Magic state distillation

In order to produce high fidelity $|T_L\rangle$ state, the idea is as follows:

- Prepare physical $|T\rangle$ states, with an error level close to the physical error p, typically $p \sim 10^{-3}$ error.
- Encode these states into a set of distance d error-correcting code, obtaining noisy $|T_L\rangle$.
- Apply a QEC code **on top** of the set of QEC codes, leading to one super-QEC code in which data qubits are themselves logical qubits.
- Perform stabilizer measurements on the super-QEC code. If an error is detected, restart the process from scratch. If not, the obtained $|T_L\rangle$ has a higher fidelity. The logical error p_L^{dis} probability, is given by the probability that an error is not detected. Assuming a distance d_s QEC code, up to $d_s - 1$ errors can be detected, meaning that $p_L^{\text{dis}} \propto p^{d_s}$.

The key consideration here is the fact that this process is trial-and-error. This technique enables to reach low error rates, at the cost of performing many operations (during the trial and error process) which are very expensive in practice.

The name "distillation" comes from the fact that, out of the many physical $|T\rangle$ states with error p we start with, eventually on "one" logical $|T_L\rangle$ state is kept, with a much lower error. This can be seen as "removing" the impurities in the $|T\rangle$ state preparation, as distillation in chemistry does.

5.3.2 Resource estimates

We compute the resource estimates for magic state preparation for large-scale fault-tolerant computing. In typical algorithms such as Shor's, typically $\sim 10^{12}$ T gates with $\sim 10^{-15}$ errors are required. Each T gate requires the distillation process. In order to reach this level of performance with $p = 0.001$, one can do this using a super-QEC code with distance $d_s = 3$ and two successive rounds of distillation, as one round provides $\sim 10^-9$ error. This mean ~ 100 QEC codes of distance d, where each QEC code possesses $\sim 2d^2$ qubits. The required value of d to maintain a T state with 10^{-15} error is what we computed previously ($d = 29$). We therefore here see that the required number of physical qubits to produce these T states is **much larger** than the requirements for storing the quantum information:

- For storing quantum information, we found that we need ~ 1 million physical qubits
- To produce the T states, we need ~ 100 million physical qubits.

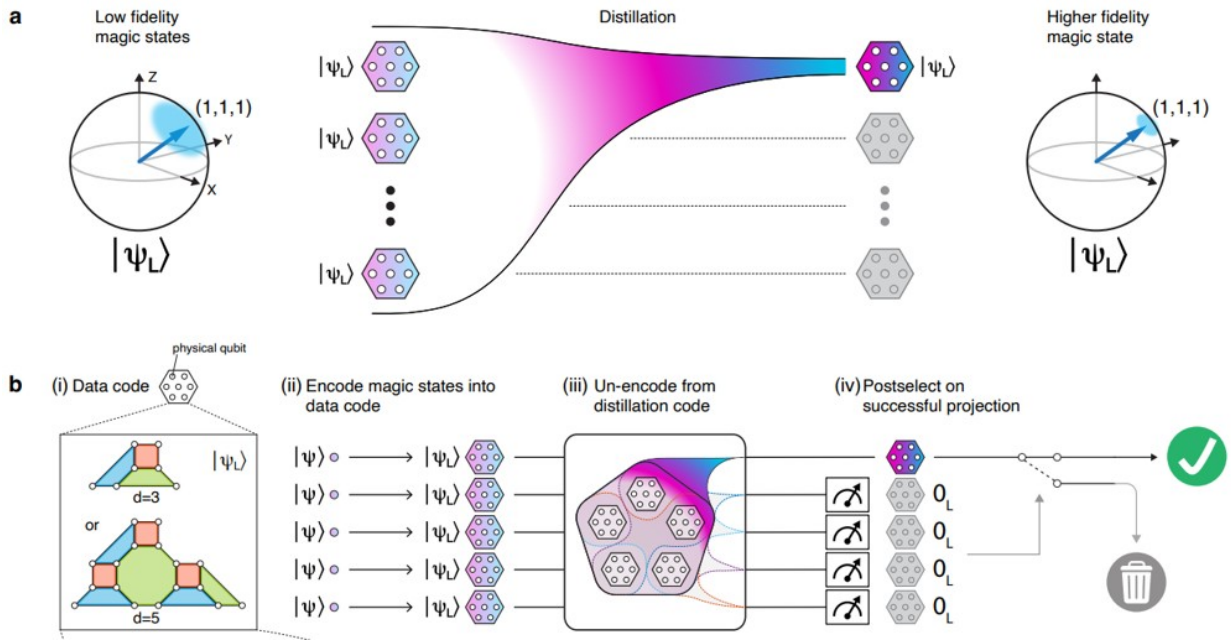

Figure 9: Magic state distillation. Adapted from [5]

Therefore, in FTQC, the main challenge nowadays is performing certain gates. We note that FTQC is a very active field of research, and it has been demonstrated in 2024 that using magic state cultivation, one can reduce by one to two orders of magnitude the requirements in the number of qubits.

6 Conclusion

In this lecture, we described the key elements that are required for fault-tolerant quantum computing: error-corrected logical qubits operating below threshold, and fault-tolerant gates.

We have seen that obtaining useful error-corrected logical qubits require low physical errors. Experimental demonstrations of below-threshold performance is therefore a great achievement on its, own, and this explains why such demonstrations were obtained very recently, and not for all the platforms discussed in the previous lectures.

We have also seen that performing fault-tolerant logical gates is highly non-trivial. Unlike for physical qubits, at the logical level the number of gates that can be performed is finite, and the number of such gates is relatively small. Thankfully, it is possible to perform universal computing with the gates that can be engineered, even though some of these gates come at a large cost.

We emphasize that the experimental demonstrations of fault-tolerant quantum computing are extremely recent, and the whole field in general is an extremely active field of research. In the coming years, new QEC codes will be discovered, and new methods for performing logical gates as well.

References

[1] Google Quantum AI, Quantum error correction below the surface code threshold, Nature (2024).

[2] Fowler, Mariantoni, Martinis, Cleland, Surface codes: Towards practical large-scale quantum computation (2012).

[3] Dolev Blustein *et al.*, Architectural mechanisms of a universal fault-tolerant quantum computer (2025).

[4] Dolev Blustein *et al.*, Logical quantum processor based on reconfigurable atom arrays, Nature (2024).

[5] Pedro Rodriguez *et al.*, Experimental demonstration of logical magic state distillation, Nature (2025).