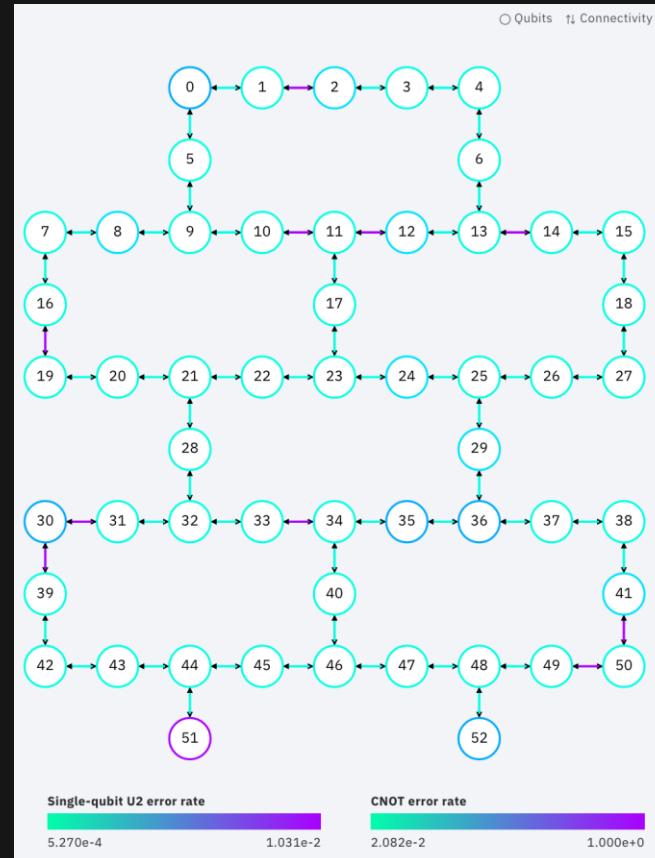


Introduction to Quantum Error Correction

James R. Wootton
Qubit Wrangler

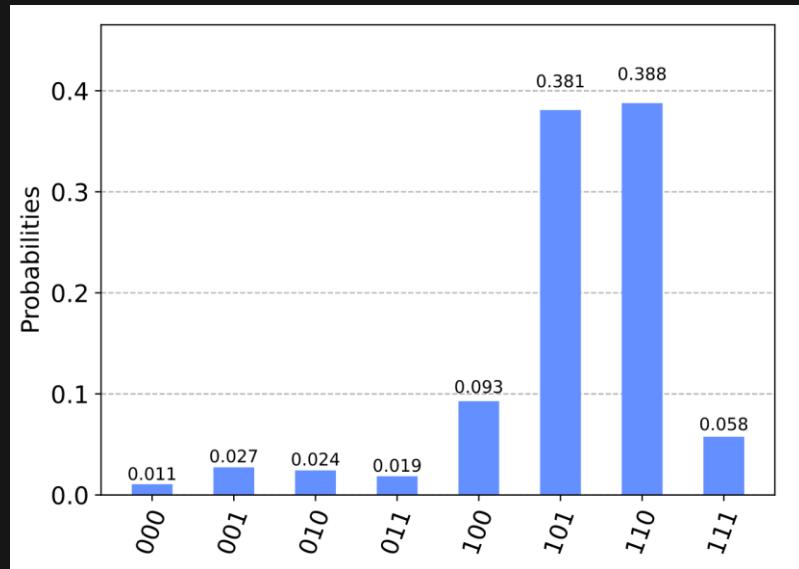
Why do we need QEC?

- Superconducting qubits don't work exactly the way they should
 - Each gate is slightly wrong
 - Qubits get poked by (small) external forces
 - Measurements sometimes lie
- The same is true for spin qubits, topological qubits, photonic qubits, ...
- **Any qubit made out of a real-life physical system will be at least a little bit rubbish**



Why do we need QEC?

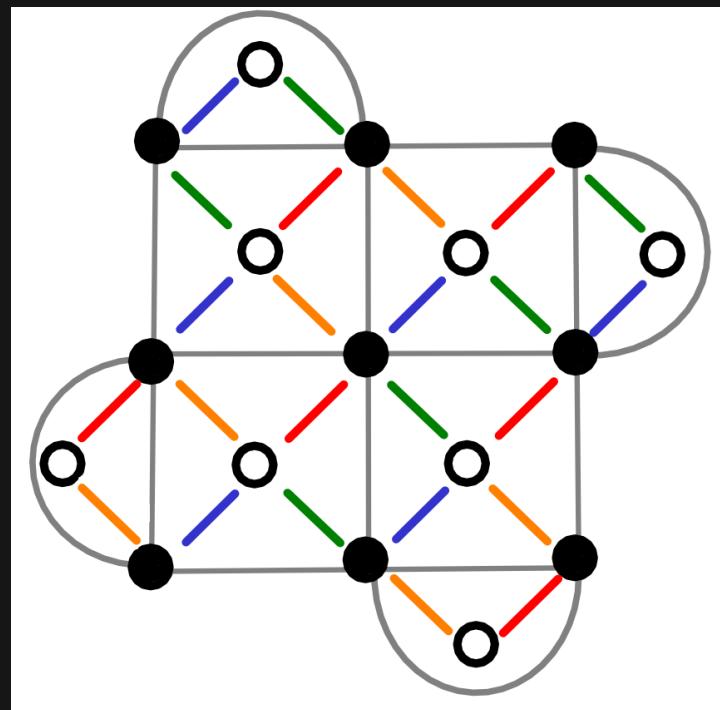
- But algorithms like Shor, Grover, etc assume that qubits are perfect
- Run them with imperfect *physical qubits*, and you'll get nonsense out
- Instead we need qubits that are flawless incarnations of the idea of quantum information
- Such ideal qubits are called *logical qubits*



- **We need to somehow make perfect logical qubits out of noisy physical qubits**

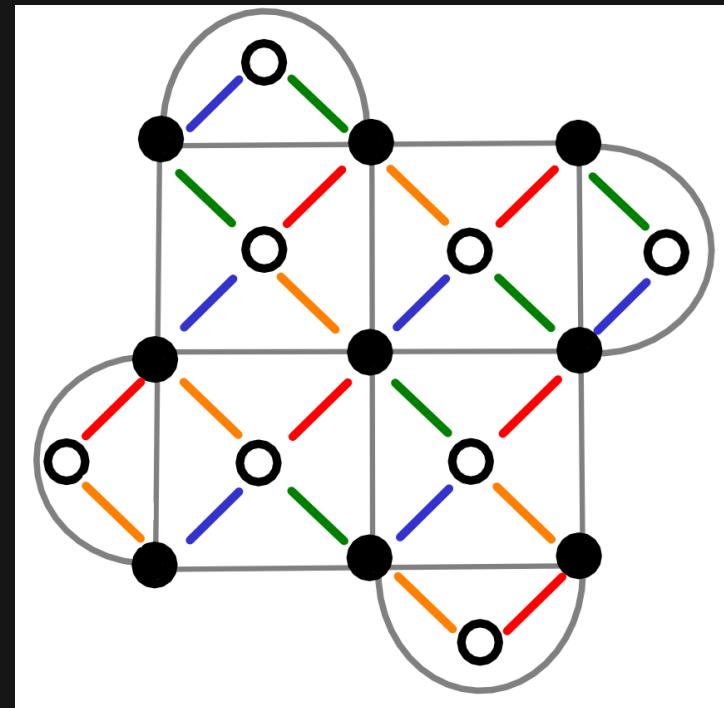
Why do we need QEC?

- QEC is how we do this!
- Typically, *many* physical qubits are needed per logical qubit
- Using QEC in algorithms is therefore a long-term goal
- As are the algorithms that require it
- In the near term, we'll try to make do and mitigate



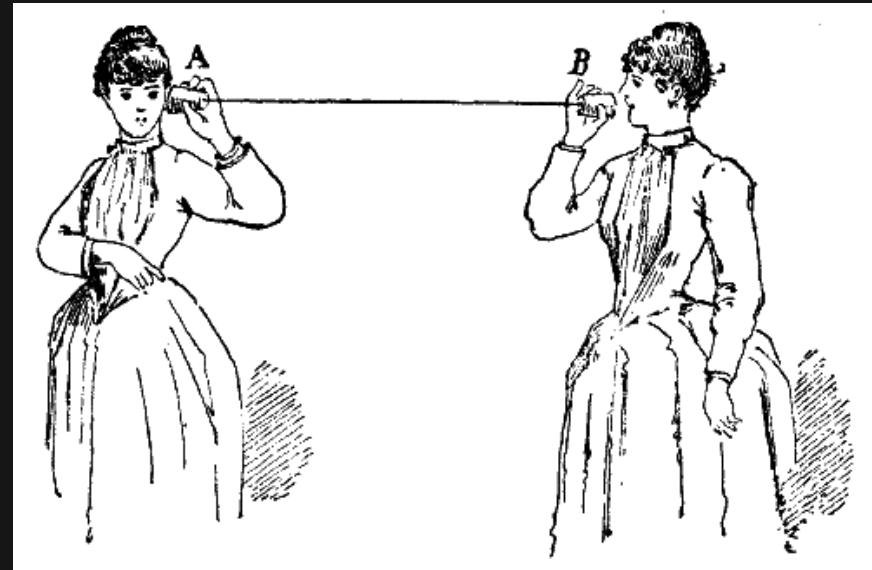
In this lecture

- We'll look at the ideas behind error correction (classical and quantum)
- We'll go through specific examples
 - Repetition code
 - Surface code
- We'll see the most important techniques
 - Syndrome measurements
 - Decoding
 - Logical operations



What is Error Correction?

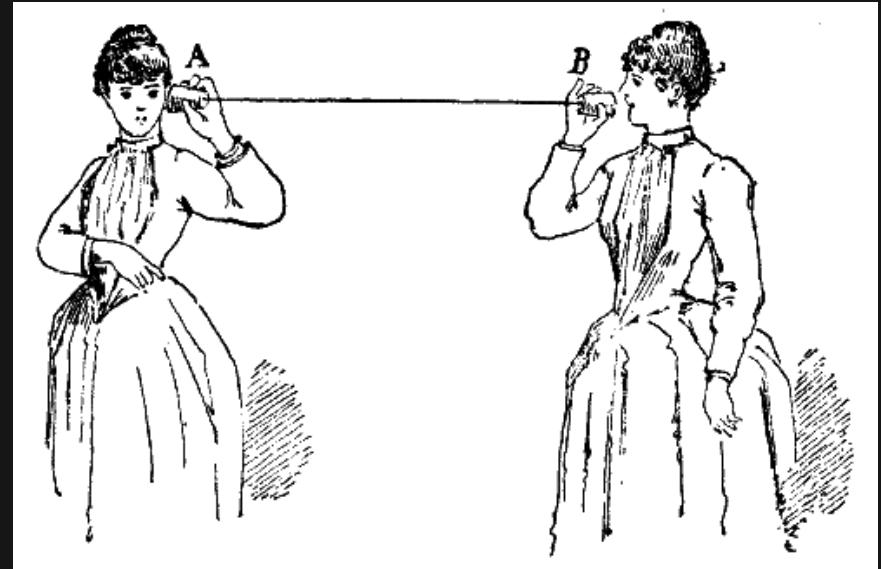
- Before we explain *quantum* error correction, we should think about the general idea
- A simple example: you are talking on the phone, and need to answer a question with ‘yes’ or ‘no’.
- Two important things to consider:
 - How likely is it that you will be misheard?
 p = probability that ‘no’ sounds like ‘yes’, etc
 - How much do you care about being misunderstood?



P_a = maximum acceptable error probability

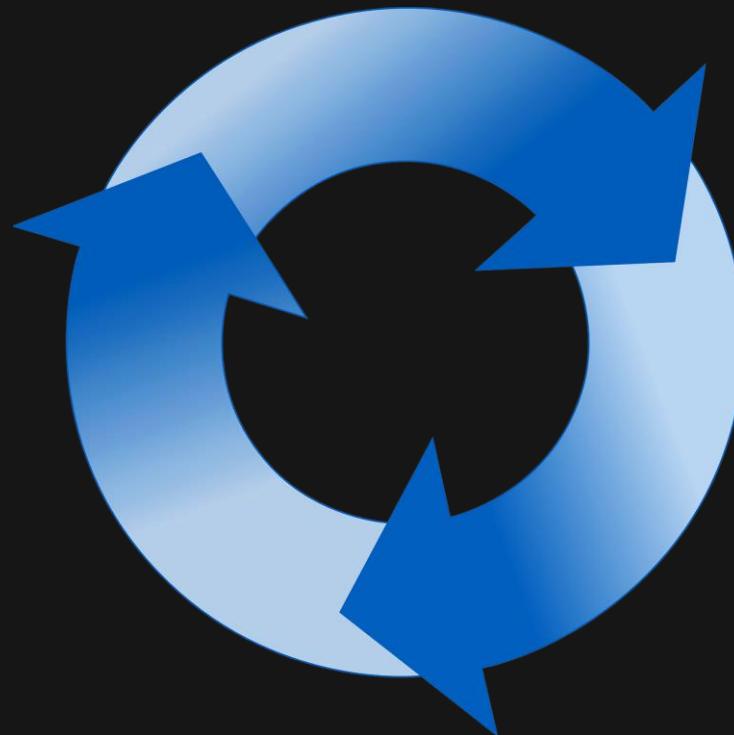
What is Error Correction?

- Usually $p \ll P_a$, so we don't need to worry.
- But what if we are being asked life-or-death questions over a noisy line?
- How can we make sure we are understood?



The Repetition Code

- We could repeat ourselves.
- With a lot of ‘no’s, it’s obvious we mean ‘no’.
- Same for mostly ‘no’s with a few random ‘yes’s thrown in.
- With this *encoding* of our message, it has become tolerant to small faults



The Repetition Code

- The receiver will need to *decode* the message.
- A sensible option is majority voting.
- A misunderstanding only happens when the majority of copies are flipped
- For d repetitions

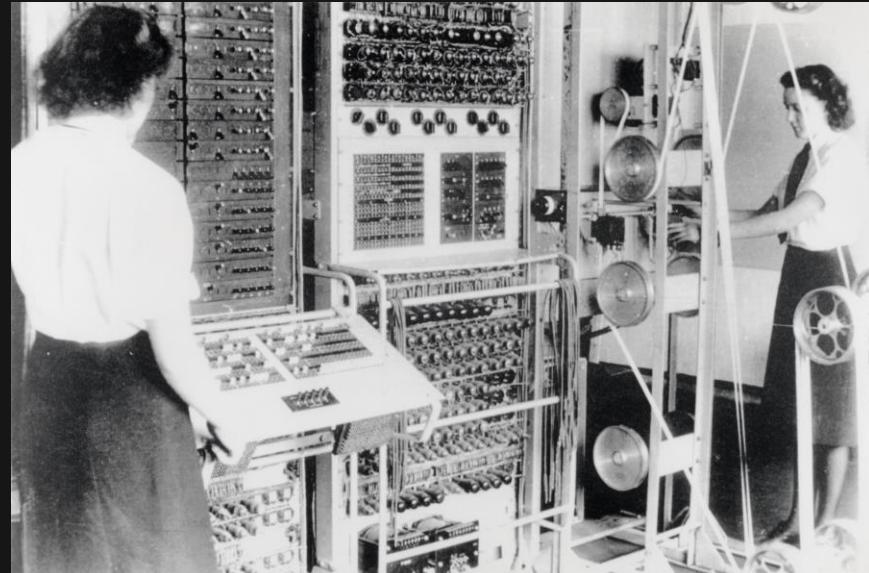
$$P = \sum_{n=0}^{\lfloor d/2 \rfloor} \binom{d}{n} p^n (1-p)^{d-n} \sim \left(\frac{p}{(1-p)} \right)^{\lfloor d/2 \rfloor}$$

- P decays exponentially with d
- With enough repetitions, we can make P as small as we like



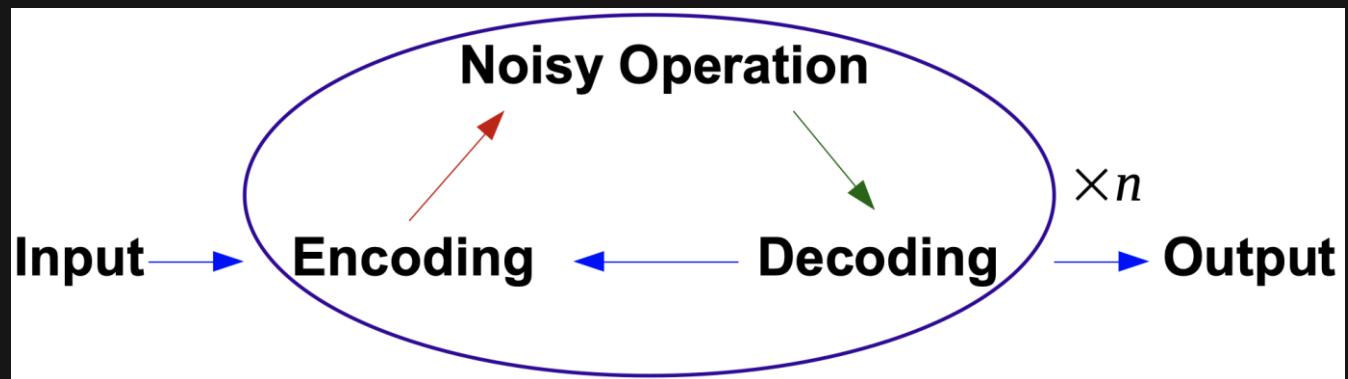
Encoding and Decoding

- This example contained all the basic features of any protocol for quantum error correction.
 - **Input:** Some information to protect.
 - **Encoding:** Transform the information to make it easier to protect.
 - **Errors:** Random perturbations of the encoded message.
 - **Decoding:** Trying to deduce the input from the perturbed message



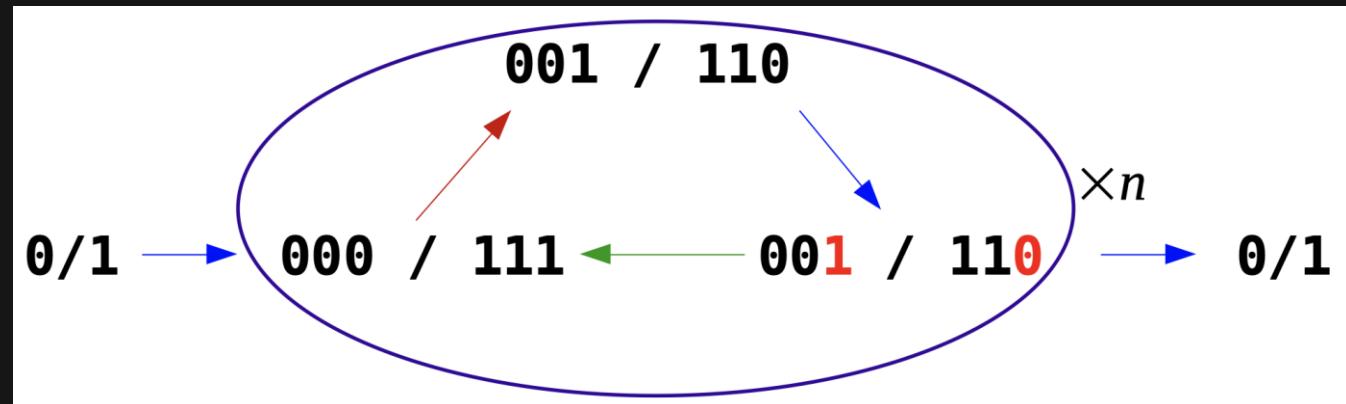
Computation

- The example we just considered was one of communication, with errors occurring during transmission.
- For computation, errors are introduced whenever we perform an operation.
- We need to keep corrected errors as they are introduced.
- Can be done by constantly decoding and re-encoding



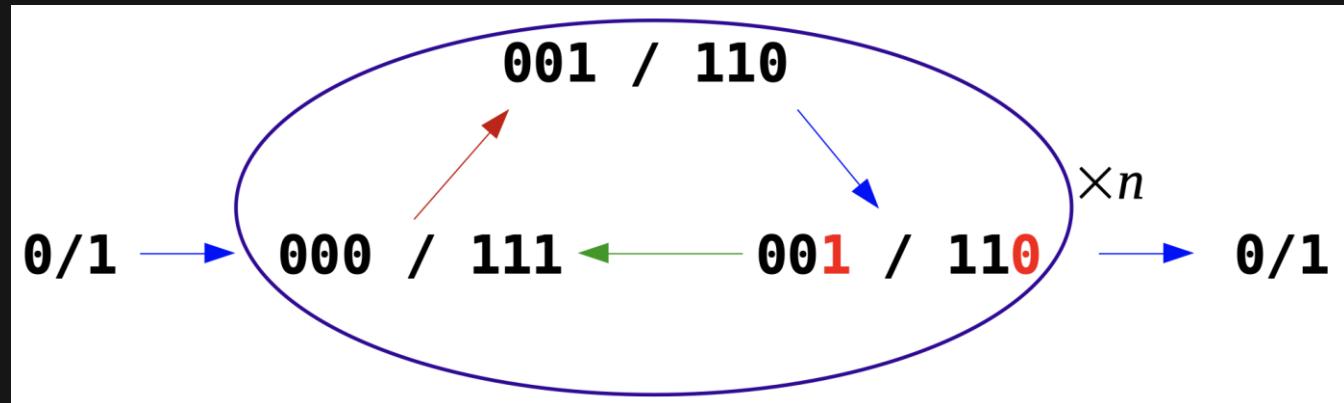
Computation

- For example, here's a single bit using the Repetition code
- The 'noisy operation' here is just doing nothing (with some errors)
- Here we don't completely decode and re-encode, but just do it enough to find and fix the errors.



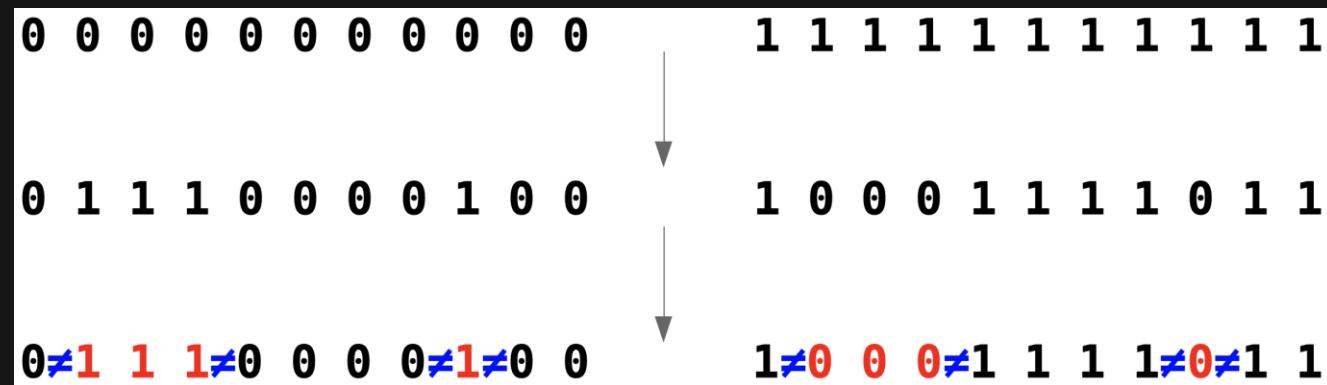
Quantum Computation

- This method works great for bits, but not for qubits
- Suppose we wanted to encode some superposition state
$$a|0\rangle + b|1\rangle \rightarrow a|000\rangle + b|111\rangle$$
- Decoding requires measurement, and that destroys the superposition.
- To protect against one bad thing, we caused another!



Quantum Computation

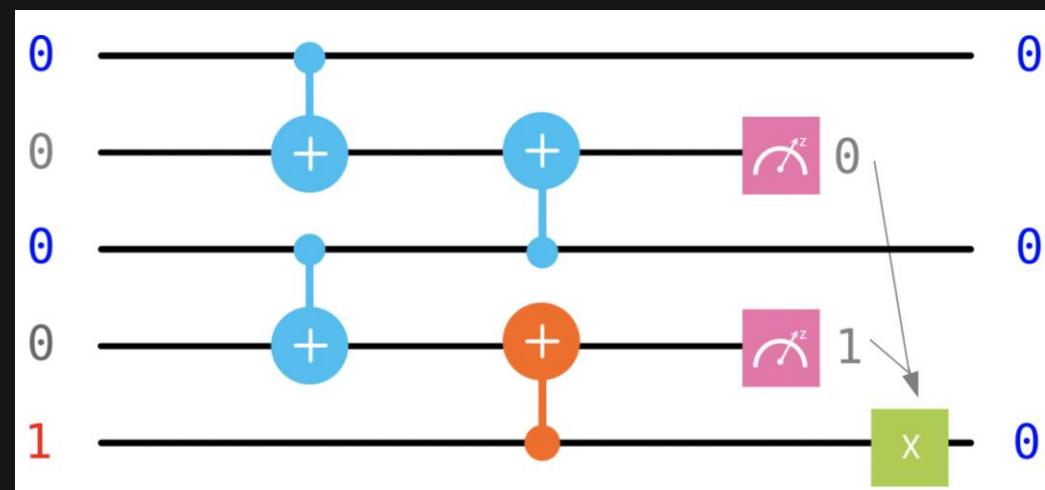
- To solve this, we need to be more careful with our measurements.
- We do need to measure, to get information about errors. But we must avoid learning about the encoded information
- For this, note that we don't actually need the bit values. We only need to know which ones have a different value to the rest.



Quantum Repetition Code

- Can be done with some extra qubits: one for each pair of code qubits.
- They are initialized in state $|0\rangle$, and are used as the target for two CX gates
- The net effect is to measure the observable $Z_j Z_{j+1}$: the Z basis parity of the two qubits
- In short: whether they are the same or different

$\text{cx } |00\rangle = |00\rangle$
 $\text{cx } |01\rangle = |00\rangle$
 $\text{cx } |10\rangle = |11\rangle$
 $\text{cx } |11\rangle = |10\rangle$



$$|00\ 0\rangle \rightarrow |00\ 0\rangle$$

$$\begin{matrix} \downarrow & \downarrow & \downarrow & \downarrow \\ a|01\ 0\rangle + b|\underline{10}\ \underline{0}\rangle \\ - & - & - & - \end{matrix}$$

$$\begin{matrix} \downarrow & \downarrow & \downarrow & \downarrow \\ a|01\ 0\rangle + b|\overset{\circ}{1}\overset{\circ}{0}\ 1\rangle \end{matrix}$$

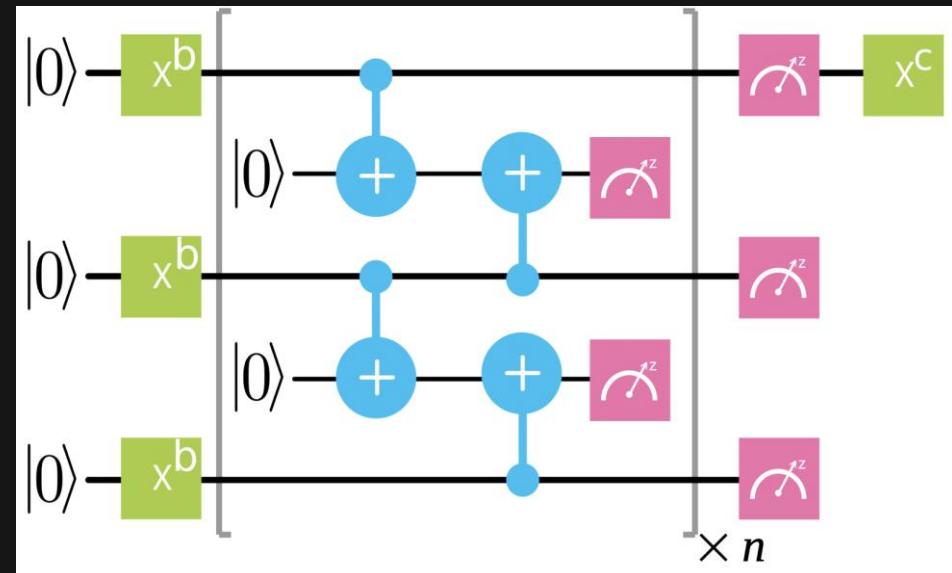
$$\begin{matrix} \downarrow & \downarrow & \downarrow & \downarrow \\ a|\overset{\circ}{0}\overset{\circ}{1}\ 1\rangle + b|\overset{\circ}{1}\overset{\circ}{0}\ 1\rangle = (a|01\rangle + b|10\rangle) @ 11 \end{matrix}$$

$$|00\rangle \rightarrow \sin\theta|00\rangle + \cos\theta|01\rangle$$


Quantum Repetition Code

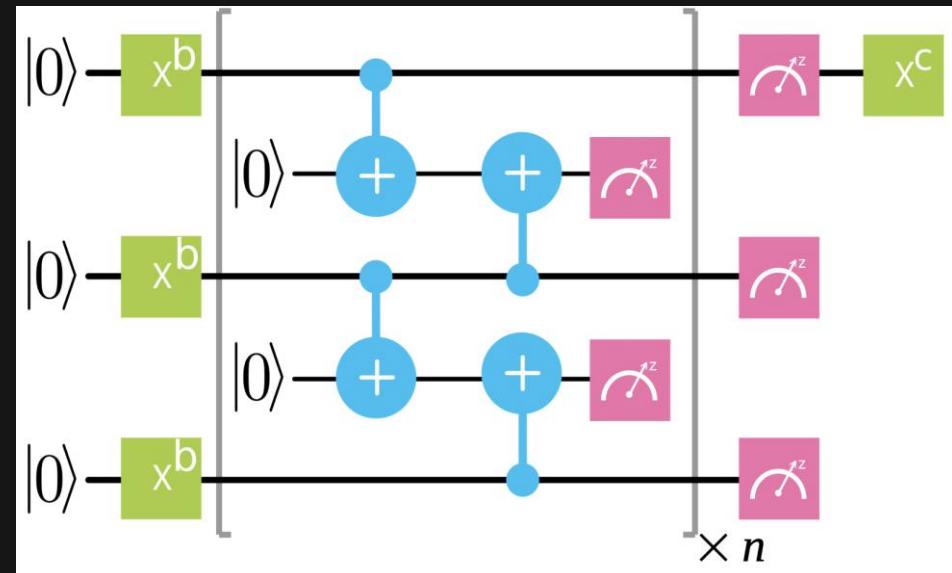
IBM Quantum

- The parity measurements are repeated over the course of the computation
- The results are used to identify where errors likely occurred and how to remove their effects
- This is done by means of a classical algorithm: a decoding algorithm
- With this we can protect against bit flip errors for an arbitrarily long time



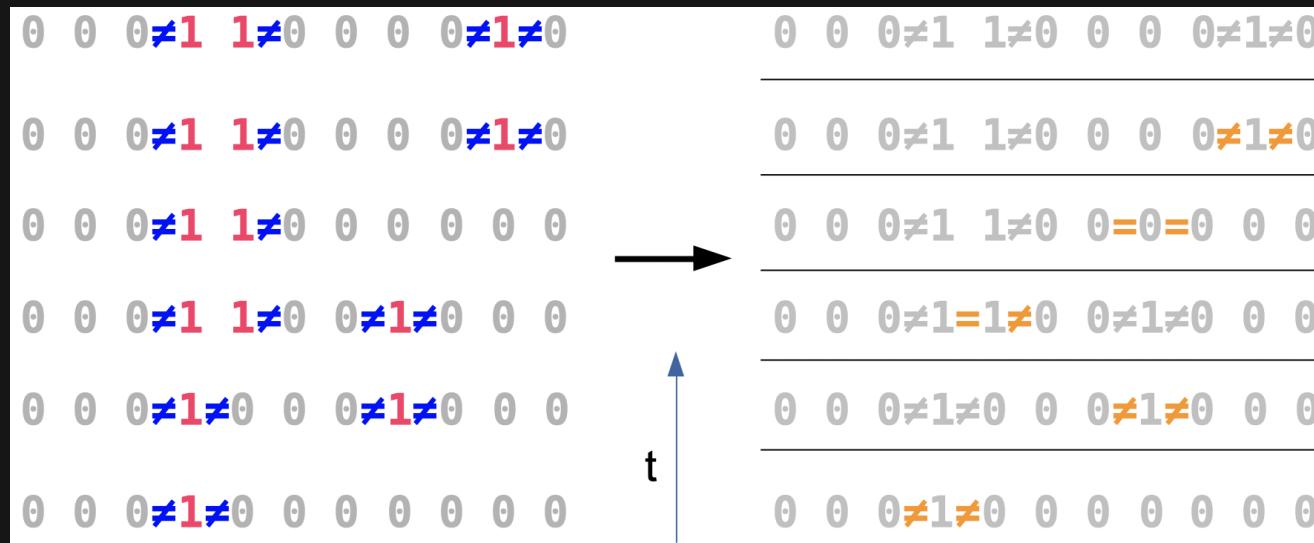
Syndrome Measurement

- This measurement is a *syndrome measurement*
- The result is known as the *syndrome*
- All codes have them
- They are ways to check that all the qubits are doing what they should be
- Their form changes from code to code, but their job is always the same
- They provide the clues that allow us to detect and correct errors



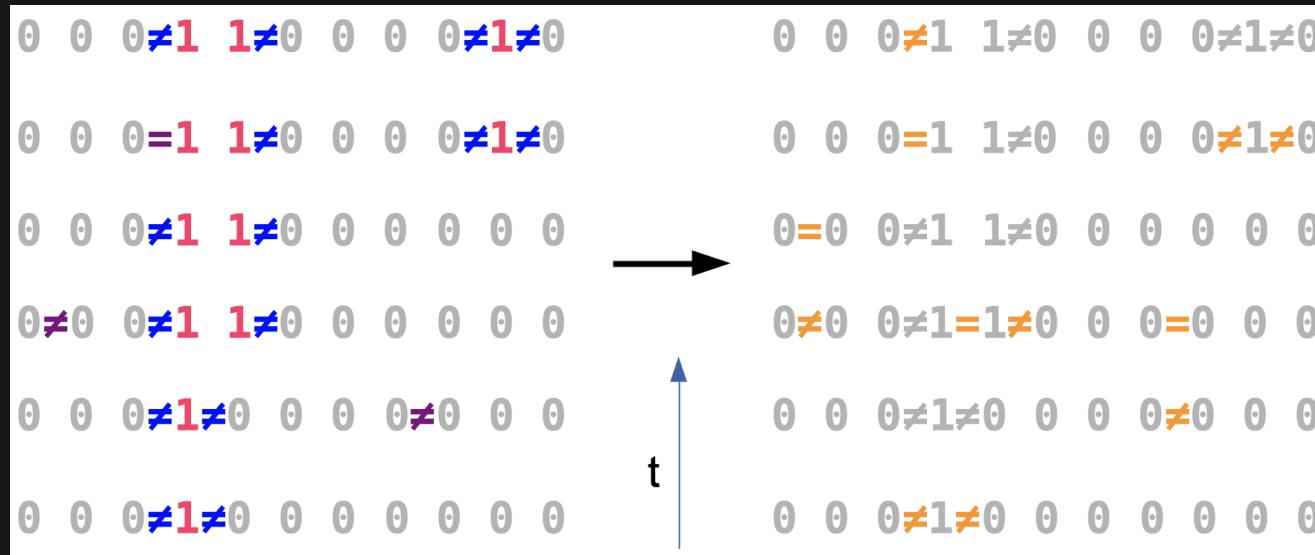
Decoding

- Start with an unrealistically simple case: errors between parity measurements only (not during)
- Focus on identifying errors for now (not correcting)
- Look for changes between rounds
- Errors create pairs of ‘defects’. Majority voting can be used to find a minimal pairing.



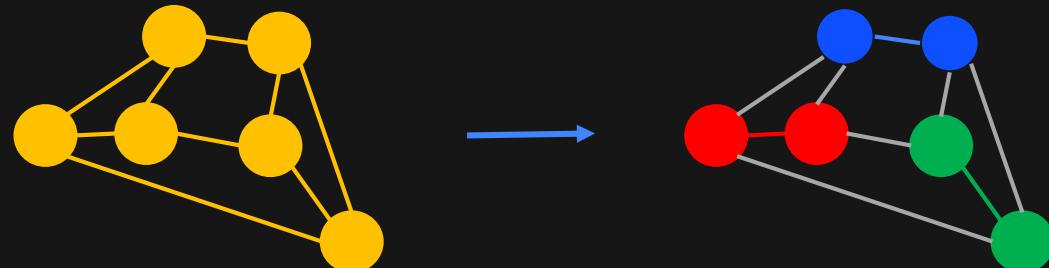
Decoding

- Next, a simple model of noise in the measurements: they randomly lie
- Again, look for changes between rounds
- Bit flips create defects with space-like separation, measurement errors with time-like
- Pairing is now a 2D problem, majority voting won't work



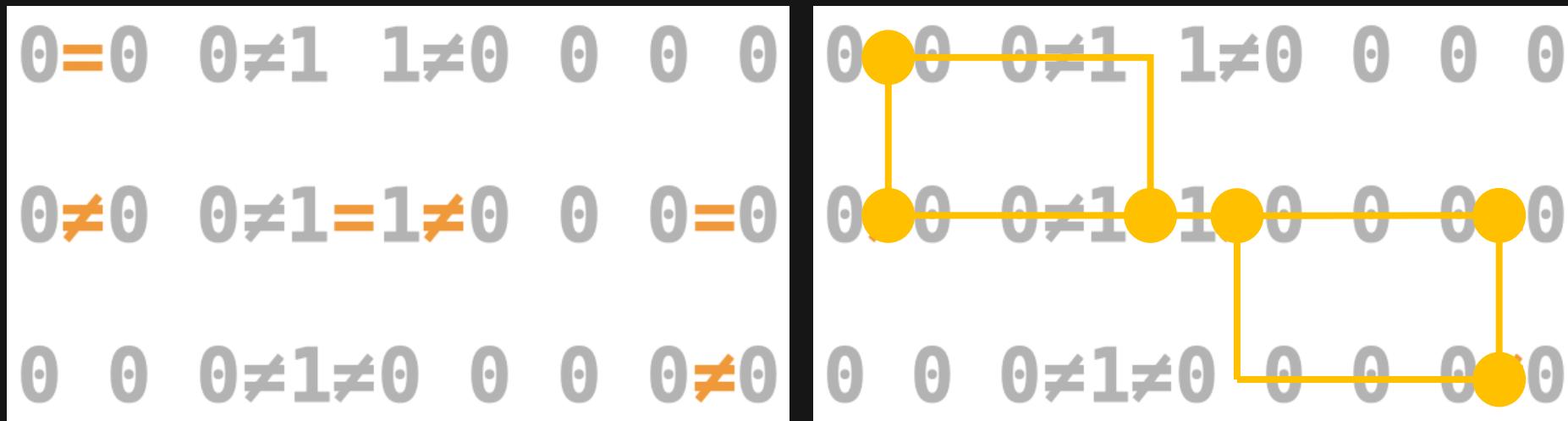
Decoding

- There are many ways to do this, all with pros and cons
- One of the best is to think of the problem as a graph
 - Defects are nodes
 - The number of errors required to link them are weighted edges
- A likely set of errors corresponds to the ‘minimum weight perfect matching’ of the graph
- Efficiently computable using the ‘Blossom’ algorithm



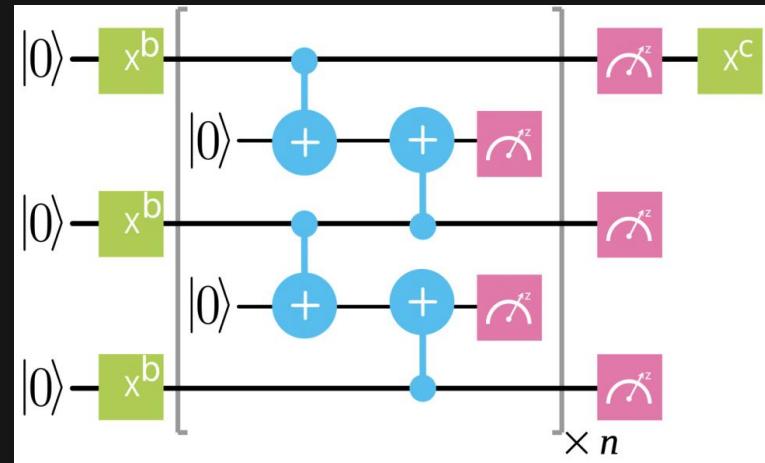
Decoding

- Here it is in action for a portion of the example from earlier



Logical Operations

- If we wanted to manipulate the logical qubit, how could we do it?
- This requires a *logical operation*, made up of many physical operations
- The *logical X* gate is easy for the repetition code
 - do *physical X* gates on code qubits
 - Code corrects imperfections
- An Rx gate essentially needs you to take the code apart and put it back together again
 - Not protected while unencoded



$$R_x(2\theta)|0\rangle = \cos \theta|0\rangle + \sin \theta|1\rangle$$

$$\rightarrow \cos \theta |000\rangle + \sin \theta |111\rangle$$

$$X|0\rangle = |1\rangle \quad X|1\rangle = |0\rangle$$

$$|000\rangle \rightarrow |111\rangle \quad |111\rangle \rightarrow |1000\rangle$$

$$X_L = X \otimes X \otimes X$$

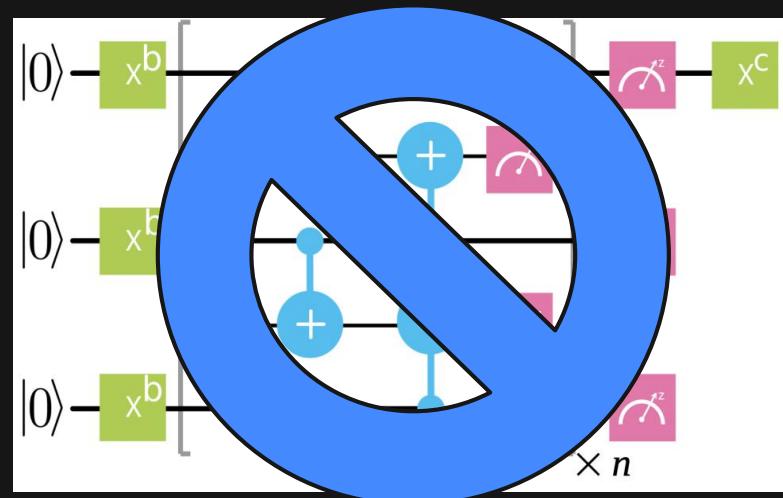
* $U|0\rangle = a|0\rangle + b|1\rangle$

$$|000\rangle \rightarrow \underline{a|000\rangle + b|111\rangle}$$

$$U_L = U \otimes U \otimes U$$

Quantum Repetition Code

- The limited set of fault-tolerant logical gates is a major problem with the repetition code
 - But it is not the only one!
 - The code only allows us to detect and correct bit flips, and only bit flips
 - Though we made sure that it doesn't cause superpositions to collapse, it doesn't protect them either
-
- **The repetition code is a good first example of quantum error correction, but it cannot give us fault-tolerant quantum computation**



$$P_X \quad X \quad a|000\rangle + b|111\rangle \quad \checkmark$$

$$P_Z \quad Z \quad a|000\rangle - b|111\rangle \quad \checkmark$$

$$P_Y \quad Y \sim XZ$$

Towards a good quantum code

- The problem with the repetition code is that it treats z basis states very different to x and y basis states
- Example 1: z basis states are product states, the others are entangled

$$\begin{aligned} |0\rangle &\rightarrow |000\rangle \\ |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) &\rightarrow \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle) \end{aligned}$$

- Example 2: Distinguishing encoded z basis requires a single qubit measurement, distinguishing x basis states requires d
- Example 3: Flipping between z basis states requires d gates, but the x basis only takes one

$$\begin{aligned} |0\rangle \rightarrow |1\rangle : \quad X_0 X_1 X_2 |000\rangle &= |111\rangle \\ |+\rangle \rightarrow |-\rangle : \quad Z_j \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle) &= \frac{1}{\sqrt{2}}(|000\rangle - |111\rangle) \end{aligned}$$

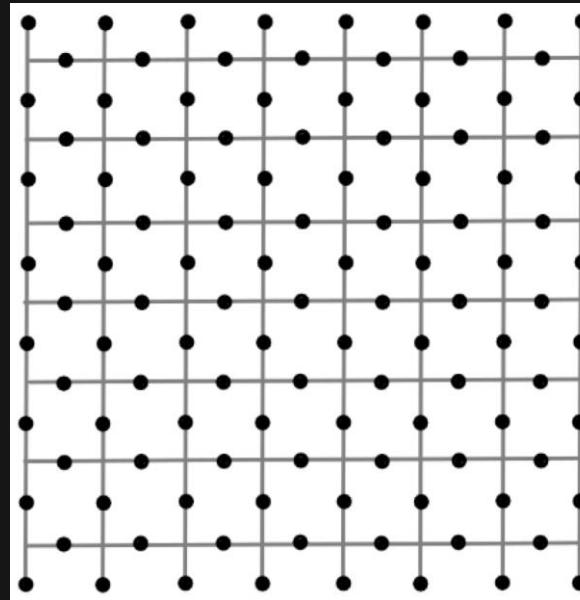
- It is not good when things are too easy, because they are easy for errors too!

The Surface Code

- Quantum error correcting codes are defined by the measurements we make
- Let's move beyond the simple $Z_j Z_{j+1}$ of the repetition code
- In the surface code we use a 2D lattice of code qubits, and define observables for plaquettes and vertices

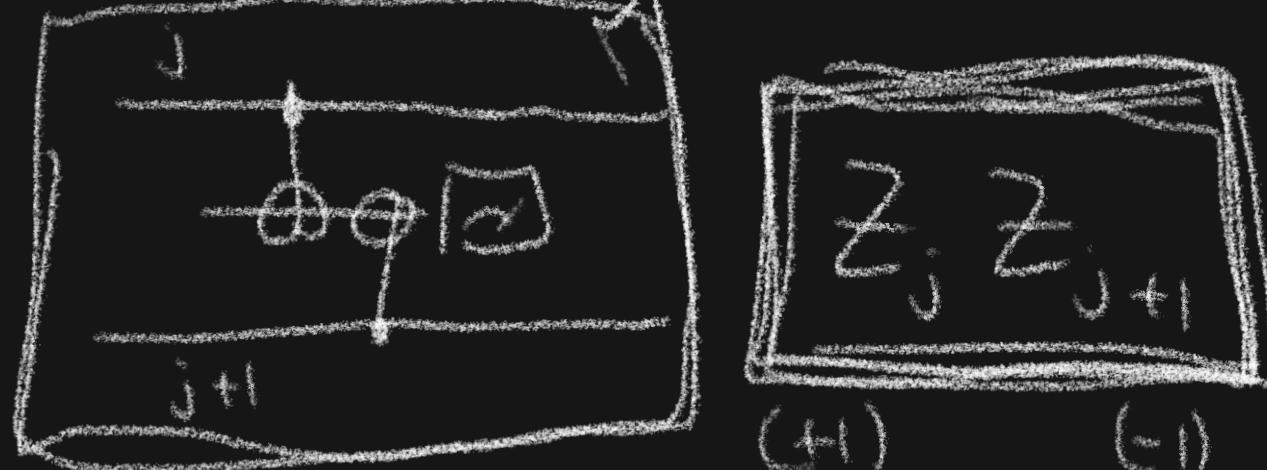
$$A_v = \sigma_x^i \sigma_x^j \sigma_x^k$$

$$A_v = \sigma_x^i \sigma_x^j \sigma_x^k$$



$$B_p = \sigma_z^l \sigma_z^m \sigma_z^n$$

$$B_p = \sigma_z^l \sigma_z^m \sigma_z^n$$



$(+1)$ (-1)

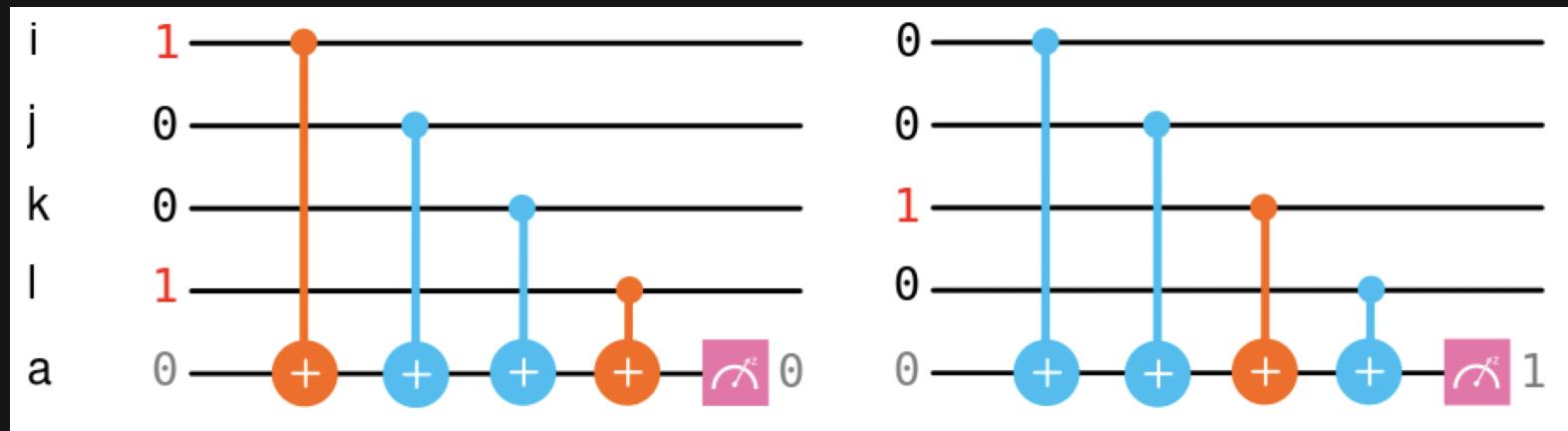
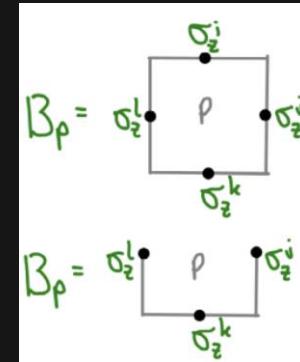
$$Z = |0 \times 0| - |1 \times 1|$$

\nearrow \searrow

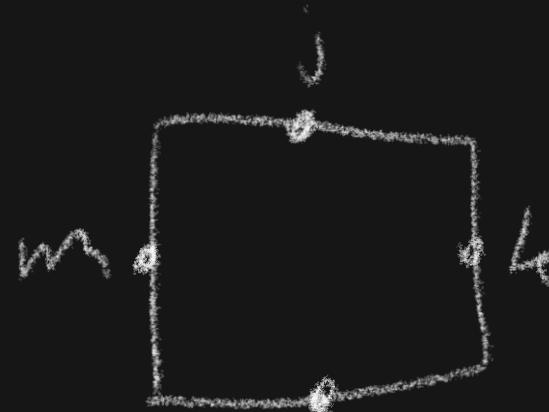
$$\begin{aligned} Z_j Z_{j+1} &= ((00 \times 00) + (11 \times 11)) \quad \leftarrow + \\ &- ((10 \times 01) + (11 \times 10)) \leftarrow \leftarrow - \end{aligned}$$

Plaquette Syndrome

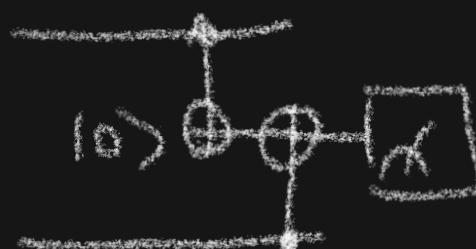
- First let's focus on the plaquette syndrome
- These are similar to the two qubit measurements in the repetition code
- Instead we measure the parity around plaquettes in the lattice
- Can again be done with CX gates and an extra qubit



$z_j z_k z_l z_m$



$z_j z_{j+1}$



0 0
1 1 → 0

0 0 0 0
0 1 1 0 → 0

0 1
1 1 → 1

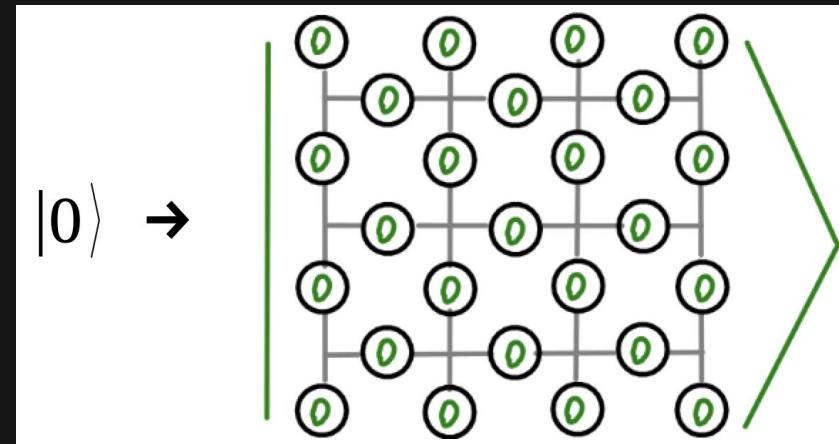
1 1 1 1

1 0
0 0 → 1

0 1 0 1
1 1 1 0 → 1

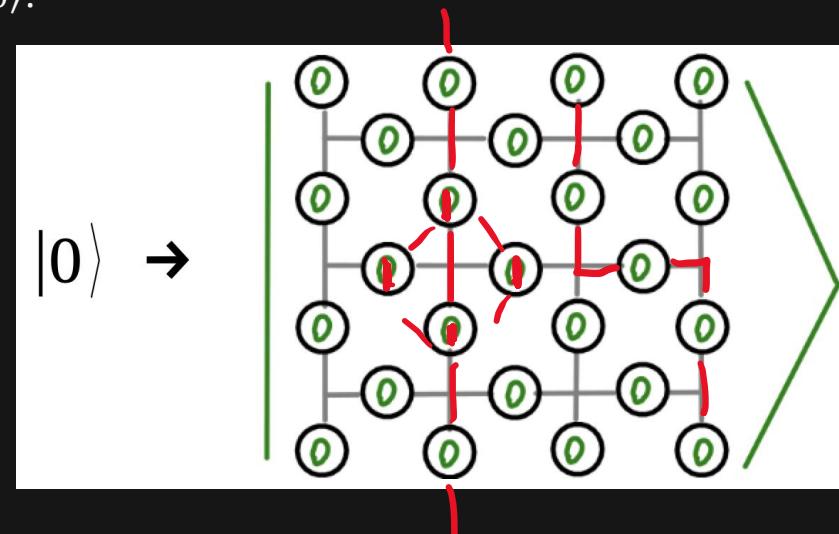
Plaquette Syndrome

- We can define a classical code (storing only a bit) based on the plaquette syndrome alone
- Valid states are those with trivial outcome for all plaquette syndrome measurements:
Even parity on all plaquettes
- How to store a 0 in this?
- How about the state where every code qubit is $|0\rangle$?



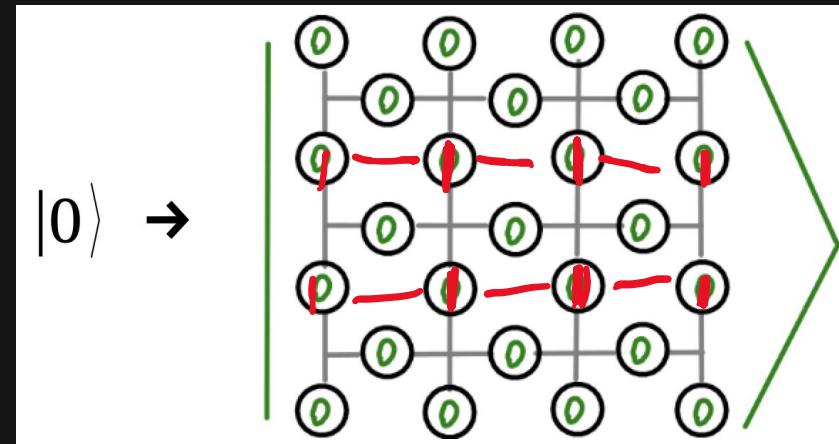
Plaquette Syndrome

- We can define a classical code (storing only a bit) based on the plaquette syndrome alone
- Valid states are those with trivial outcome for all plaquette syndrome measurements:
Even parity on all plaquettes
- How to store a 0 in this?
- How about the state where every code qubit is $|0\rangle$?



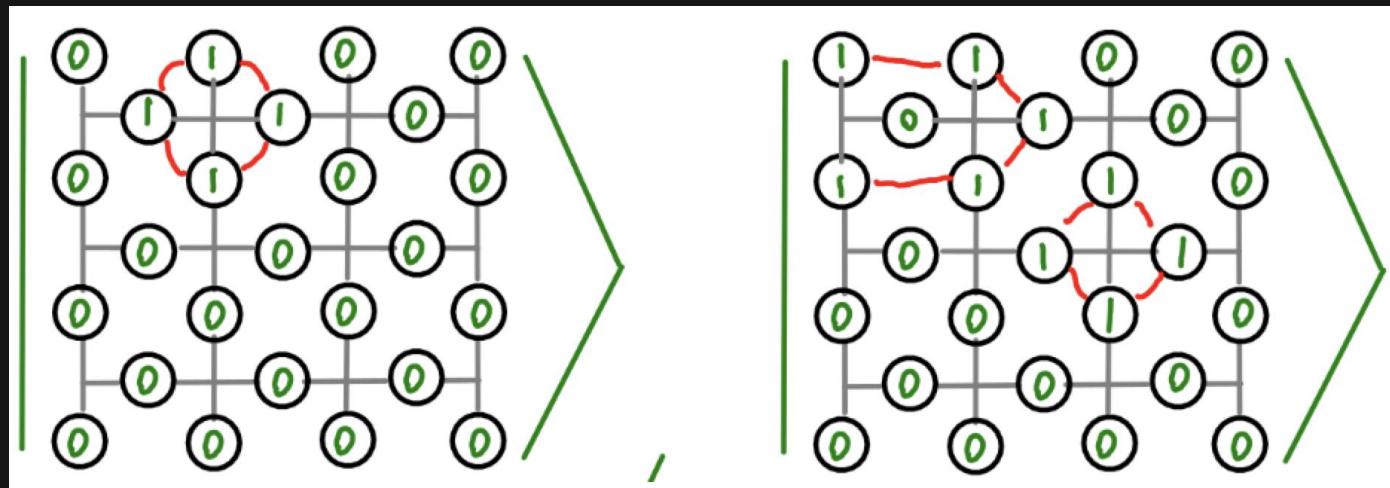
Plaquette Syndrome

- We can define a classical code (storing only a bit) based on the plaquette syndrome alone
- Valid states are those with trivial outcome for all plaquette syndrome measurements:
Even parity on all plaquettes
- How to store a 0 in this?
- How about the state where every code qubit is $|0\rangle$?



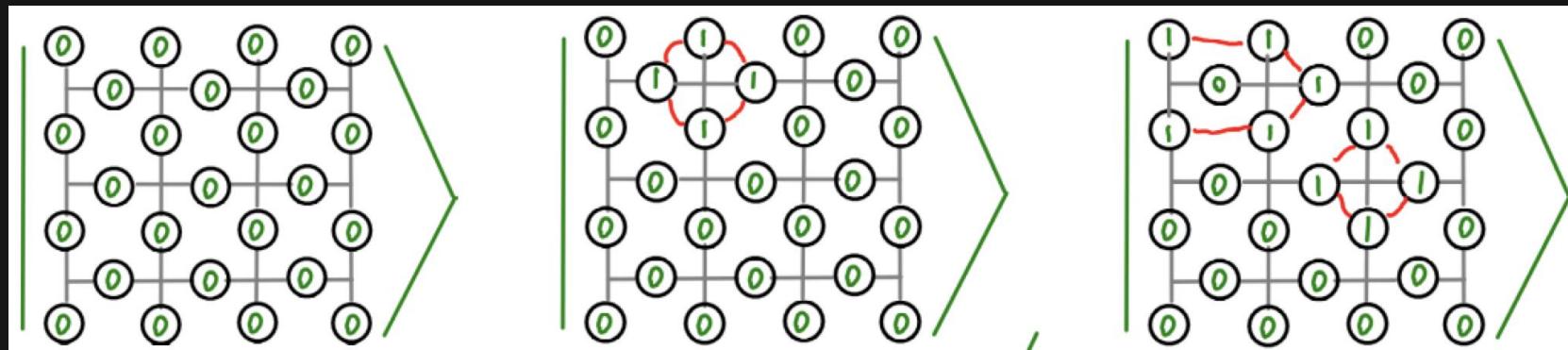
Plaquette Syndrome

- There are ‘nearby’ states that also have even parity on all plaquettes
- These can’t be a different encoded state: they are only a few bit flips away from our encoded 0 state
- We’ll treat them as alternative ways to store a 0



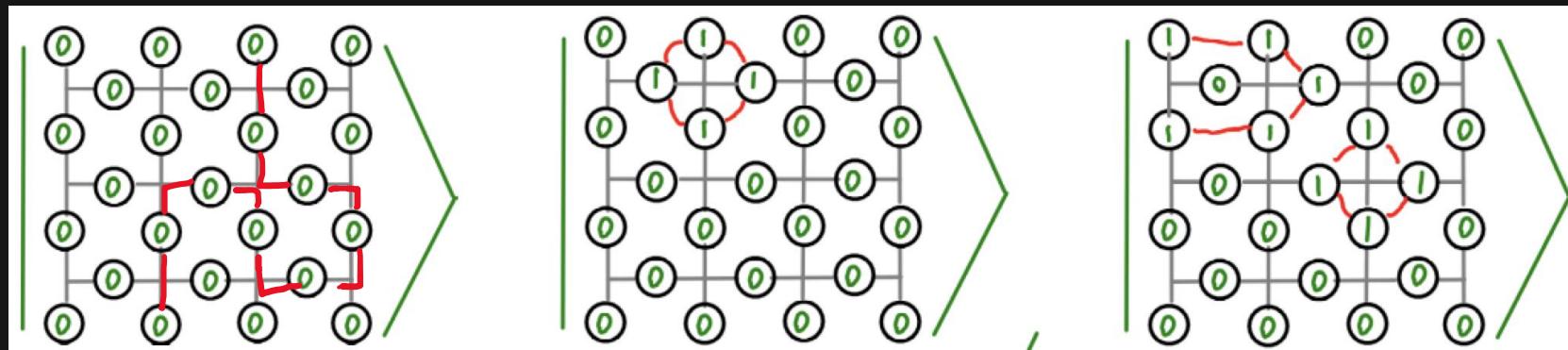
Plaquette Syndrome

- Given any state for an encoded 0
 - Pick a vertex
 - Apply bit flips around that vertex
- Now you have another valid state for 0
- This generates an exponentially large family



Plaquette Syndrome

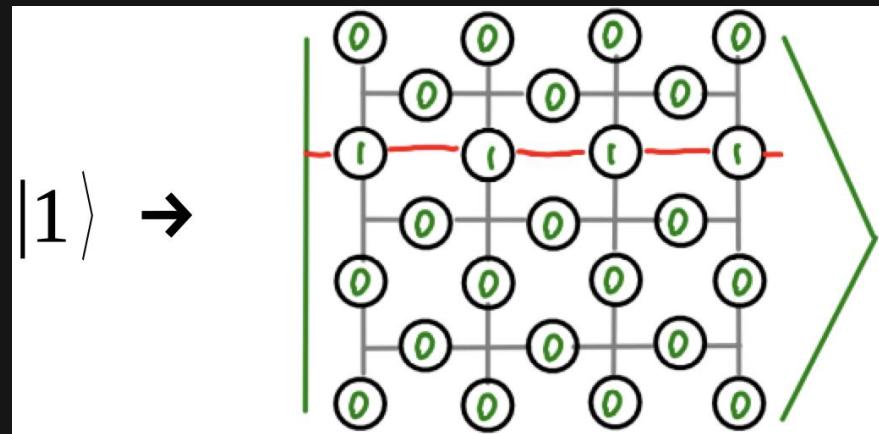
- The states in this family can be very different
- But they all share a common feature
 - Any line from top to bottom (passing along edges) has even parity
- This is how we can identify an encoded 0
- And it gives us a clue about how to encode a 1



Plaquette Syndrome

IBM Quantum

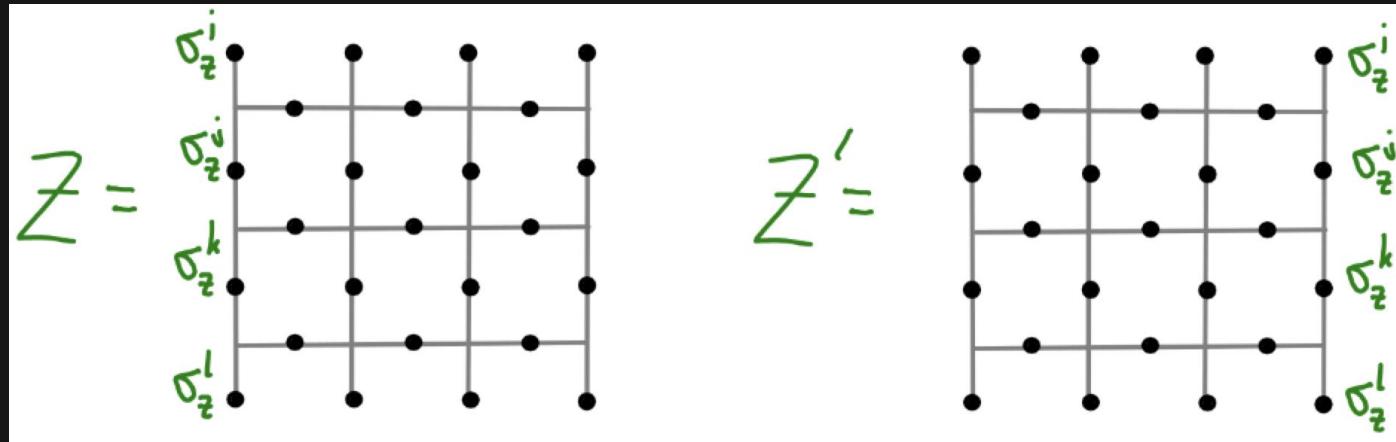
- For our basic encoded 1, we use a bunch of 0s with a line from left to right (passing through plaquettes)



- This also spawns an exponentially large family
- All have *odd* parity for a line from top to bottom
- Unlike the repetition code, distinguishing encoded 0 and 1 requires some effort (which is good!)

Logical X and Z

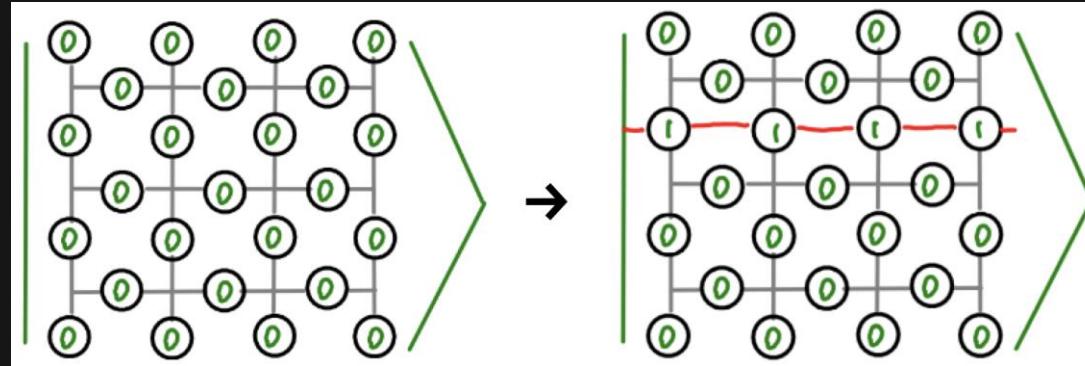
- Distinguishing 0 and 1 corresponds to measuring Z on the physical qubit
- The following observables detect what we need



- Or the same on any line from top to bottom
- Uses the edges has a nice advantage: we can think of them as large (unenforced) plaquettes

Logical X and Z

- To flip between 0 and 1, we can flip a line of qubits

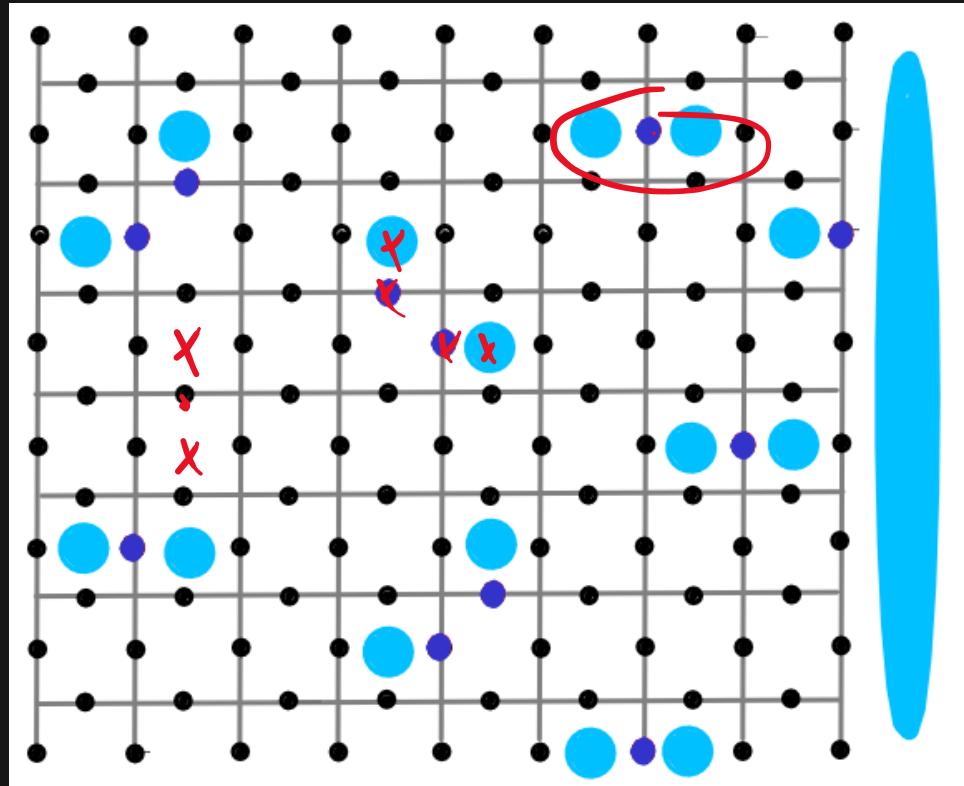


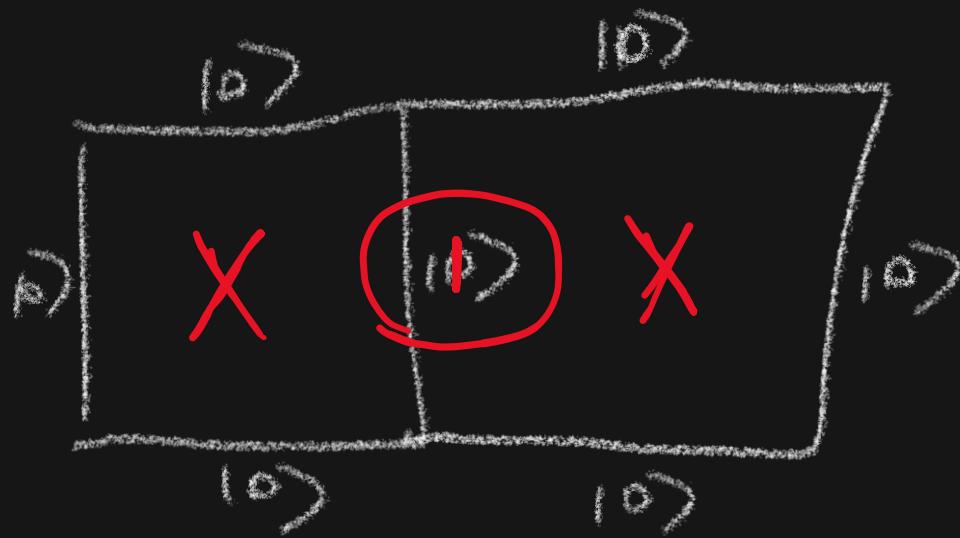
- Such lines of flips act as an X on the logical qubit

$$\text{X} = \begin{array}{c} \sigma_x^i \quad \sigma_x^j \quad \sigma_x^k \quad \sigma_x^l \\ \bullet - \bullet - \bullet - \bullet \\ | \quad | \quad | \quad | \\ \bullet - \bullet - \bullet - \bullet \\ | \quad | \quad | \quad | \\ \bullet - \bullet - \bullet - \bullet \\ | \quad | \quad | \quad | \\ \bullet - \bullet - \bullet - \bullet \end{array}$$

Effects of Errors

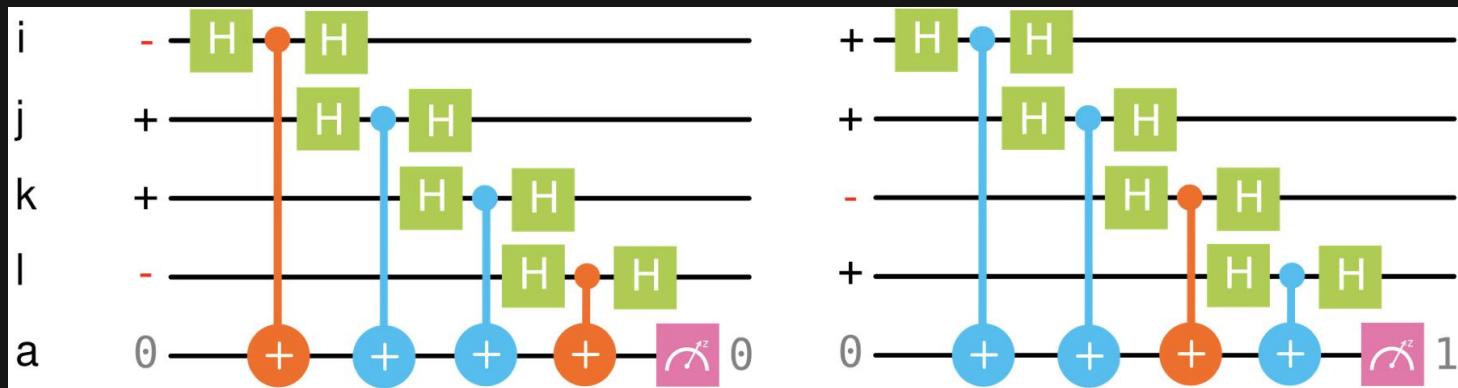
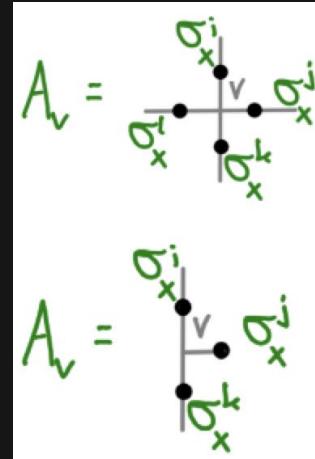
- Applying an X to any code qubit changes the parity of its two plaquettes
- An isolated X creates a pair of defects
- Further Xs can move a defect, or annihilate pairs of them
- A logical X requires many errors to stretch across the lattice
- With the plaquette operators, we can encode and protect a *bit*





Vertex Syndrome

- Now forget the plaquettes and focus on vertices
- These observables can also be measured using CX gates an an ancilla
- In this case they look at the $|+\rangle$ and $|-\rangle$ states, and count the parity of the number of $|-\rangle$ s



$|0\rangle \rightarrow 0$  $|1\rangle \rightarrow 1$

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

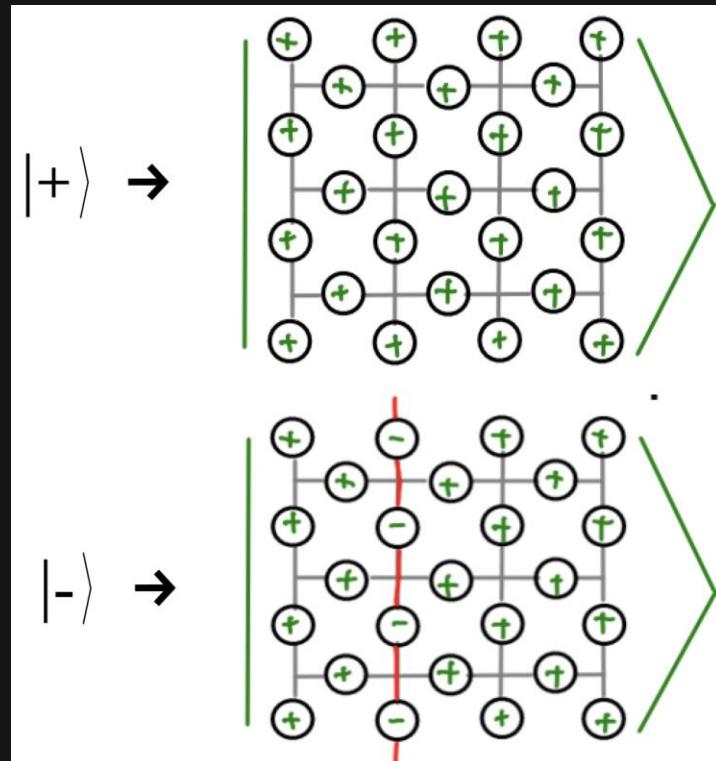
 $|+\rangle \rightarrow 0$  $|-\rangle \rightarrow 1$

$$H|0\rangle = |+\rangle$$

$$H|+\rangle = |0\rangle$$

Vertex Syndrome

- These operators also allow us to encode and protect a bit value
- In this case, let's use + and - to label the two states
- They are encoded using suitable patterns of $|+\rangle$ and $|-\rangle$ states for the code qubits
- As with the plaquettes, these also correspond to exponentially large families of states

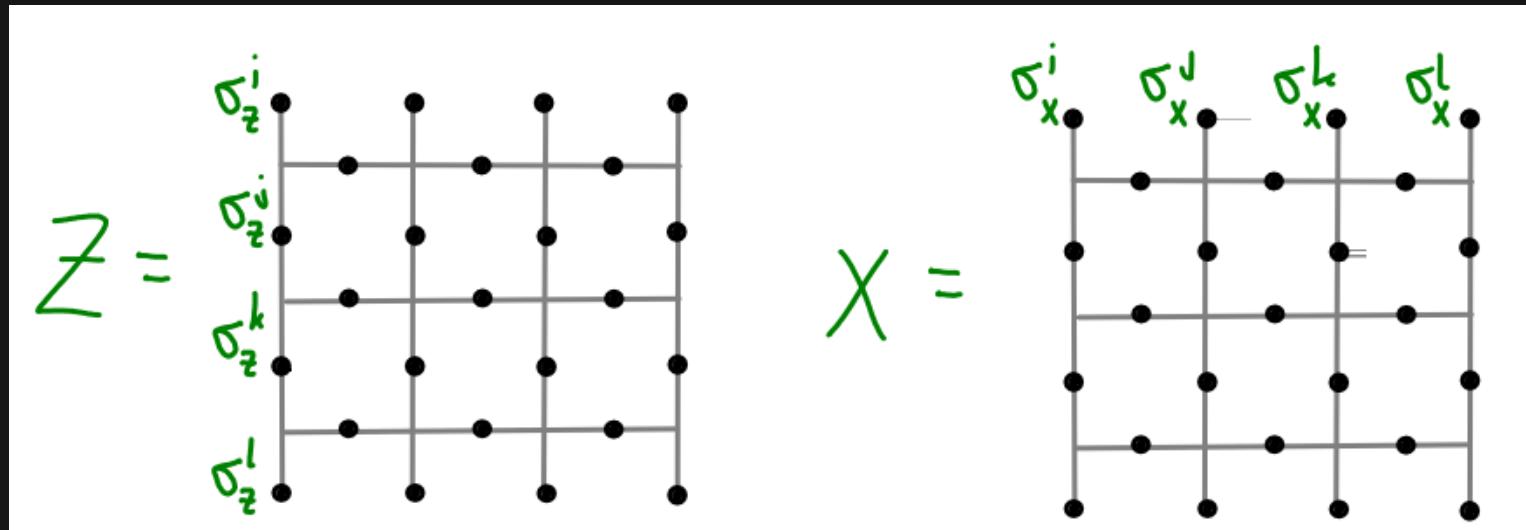


$$X|0\rangle = |1\rangle \quad X|1\rangle = |0\rangle$$

$$Z|+\rangle = |+\rangle \quad Z|-\rangle = |-\rangle$$

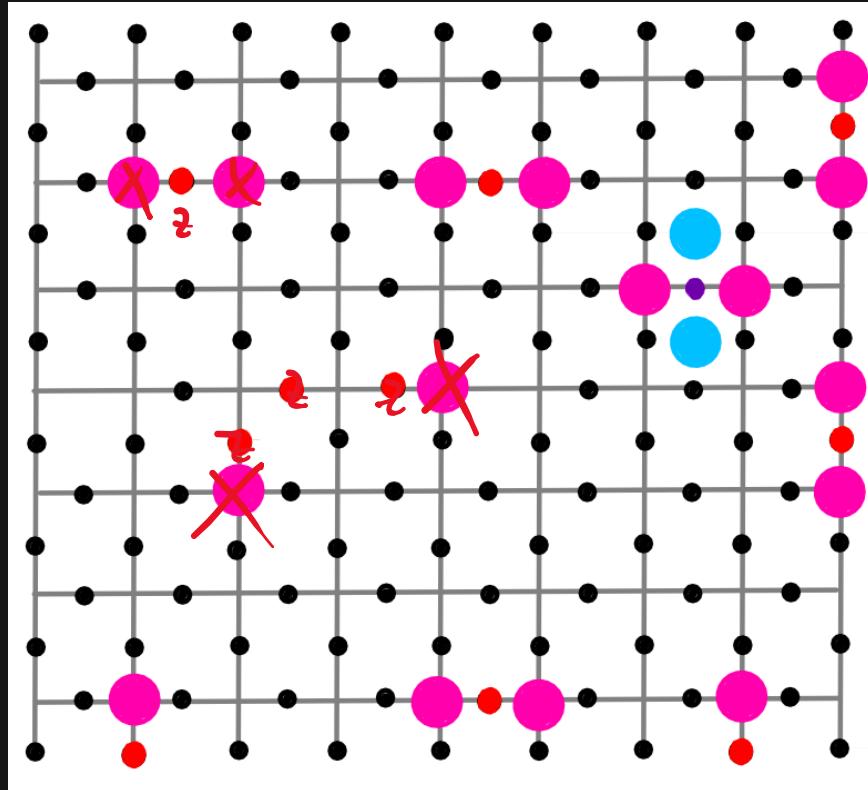
Logical X and Z

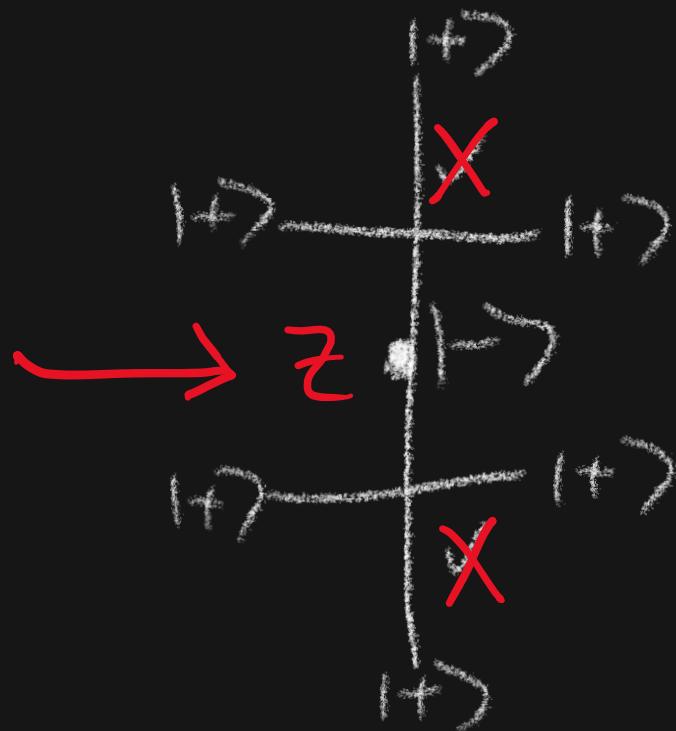
- What is the X operator (distinguish between $|+\rangle$ and $|-\rangle$)?
- What is the Z operator (flip between $|+\rangle$ and $|-\rangle$)?
- Turns out they are exactly the same as before!



Effects of Errors

- Applying a Z to any code qubit changes the X parity of its two vertices
- An isolated Z creates a pair of defects
- Further Zs can move a defect, or annihilate pairs of them
- A logical Z requires many errors to stretch across the lattice
- With the vertex operators, we can encode and protect a *bit*



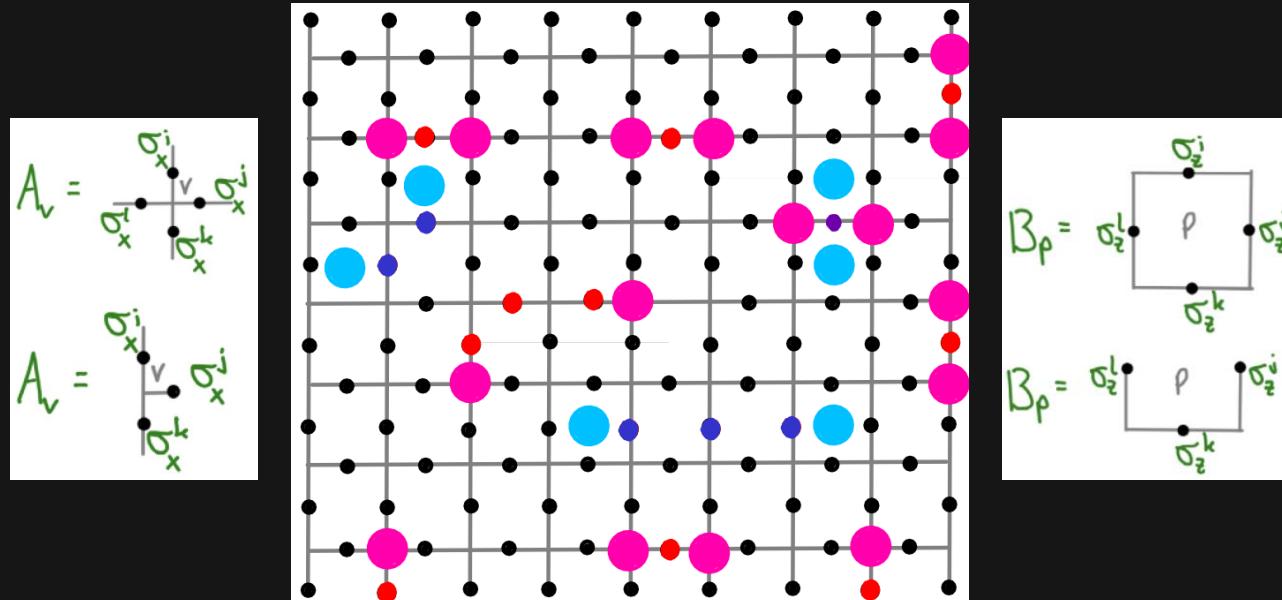


$$\cancel{X|+\rangle = |+\rangle}$$

$$Z|+\rangle = |-\rangle$$

Putting it all Together

- The plaquette and vertex operators commute
- This allows us to detect both X and Z errors
- Since $Y \sim XZ$, we can detect Y errors too

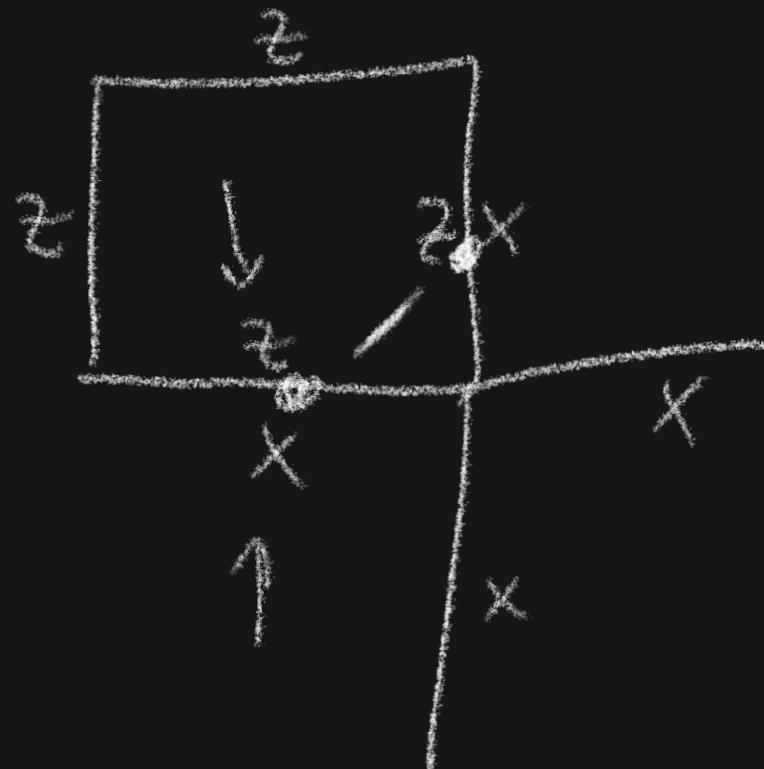


$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

$$\frac{1}{\sqrt{2}}(|++\rangle + |--\rangle)$$

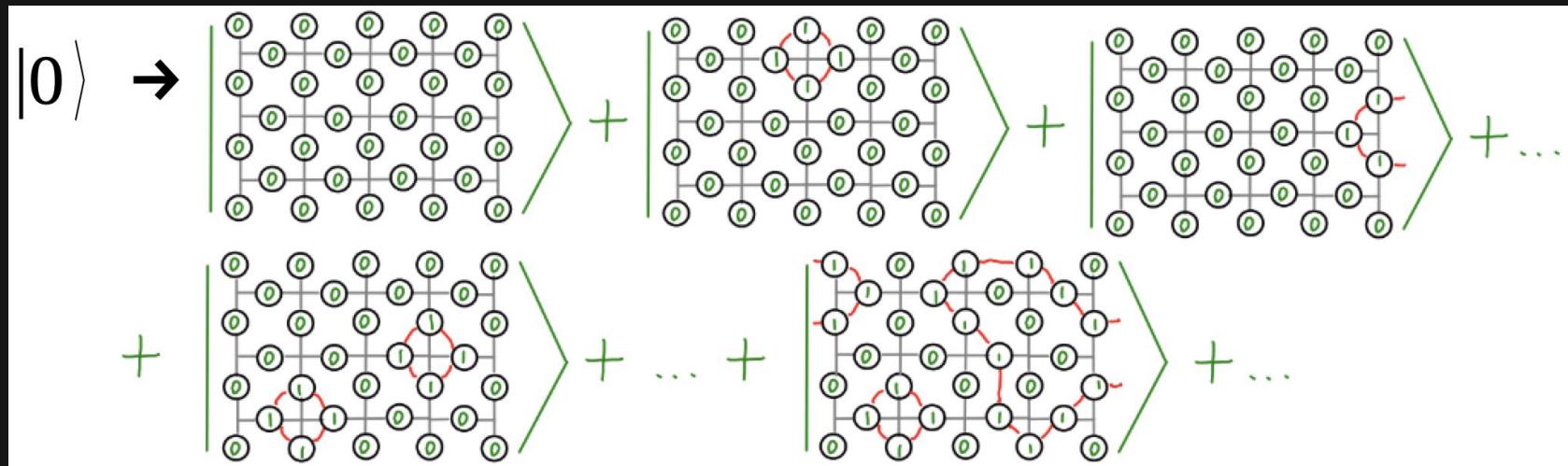
$$\underline{|0\rangle = \frac{|+\rangle + |- \rangle}{\sqrt{2}}}$$

$$|0\rangle$$



Putting it all Together

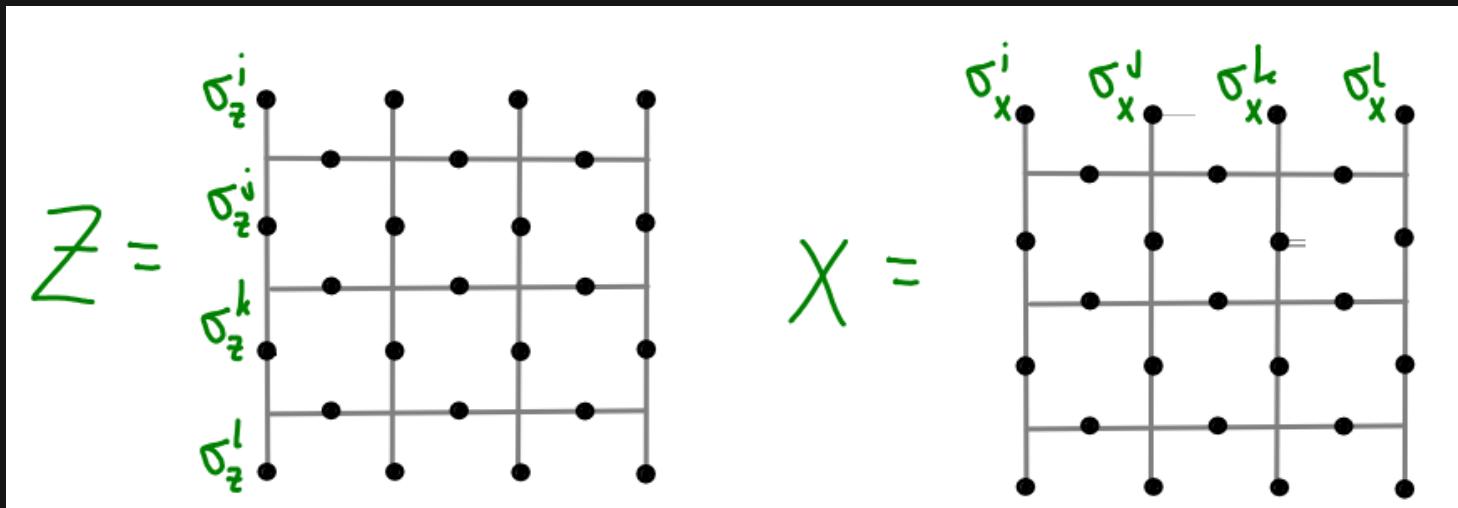
- Encoded states now unique: superposition of all previous solutions
- For example, the encoded 0



- Satisfies $A_v|\psi\rangle = |\psi\rangle$ and $B_v|\psi\rangle = |\psi\rangle$, so will give the 0 outcome for all stabilizer measurements

Putting it all Together

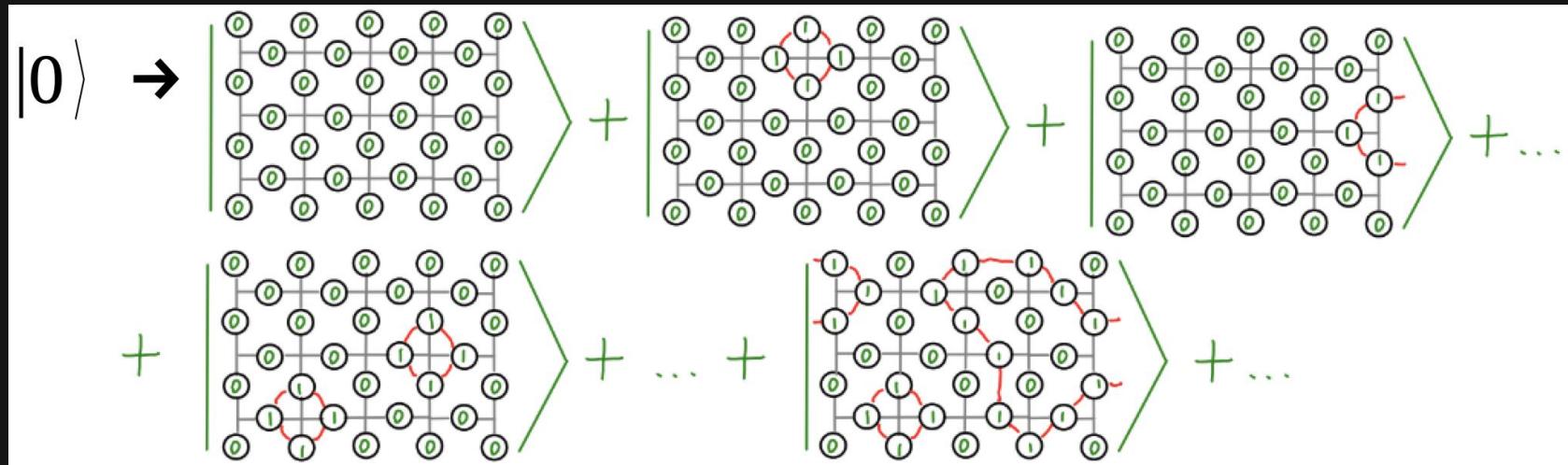
- The Z and X operators on the encoded qubit are exactly the same as before



- These, and the Hadamard, can be performed fault-tolerantly

Putting it all Together

- The states we need are highly entangled quantum states
- They are examples of topologically ordered states



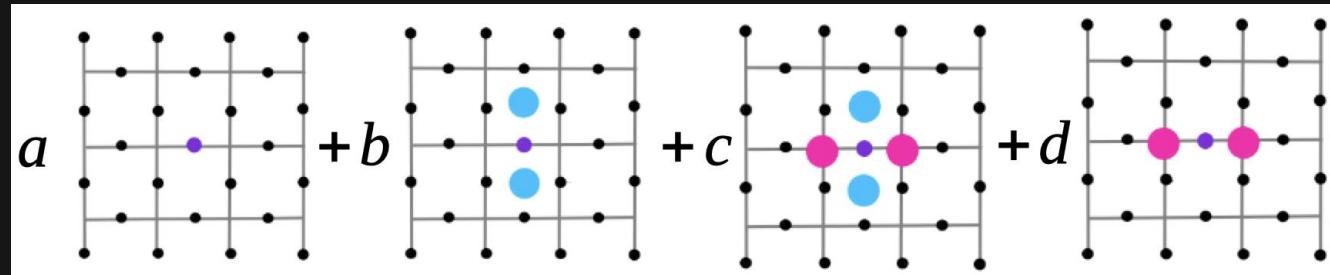
- Though such things can be hard to make, we create and protect them with the syndrome measurements

Putting it all Together

- We are not just protected against X and Z, but all local errors
- As mentioned earlier, $Y \sim XZ$
- Everything else can be expressed

$$E = a I + b X + c Y + d Z$$

- This creates a superposition of different types of error on the surface code



- Measuring the stabilizers collapses this to a simple X, Y or Z

$P_x \quad X$

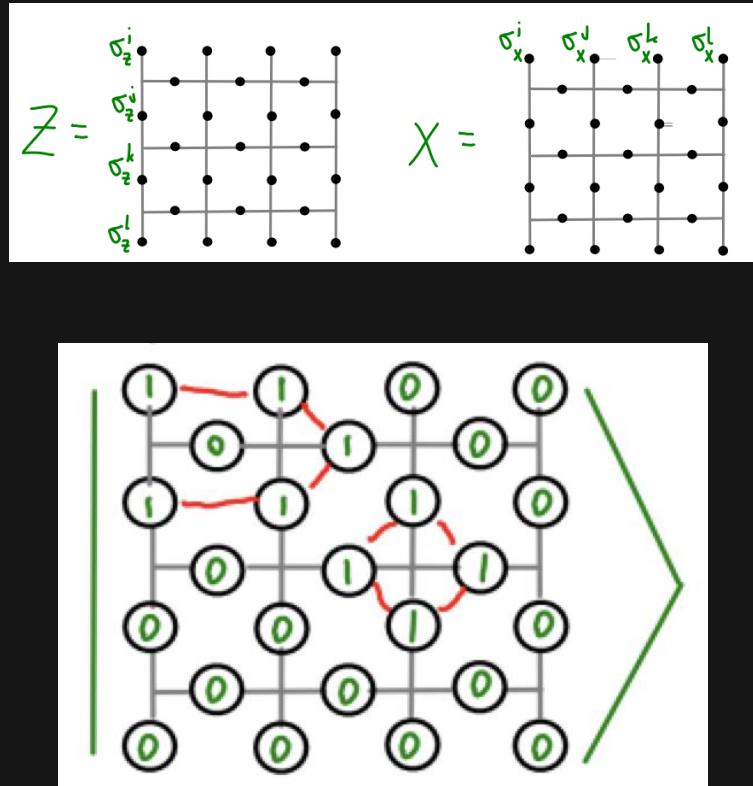
$P_y \quad Y$

$P_z \quad Z$

$$E = a\mathbb{1} + bX \\ + cY + Z$$

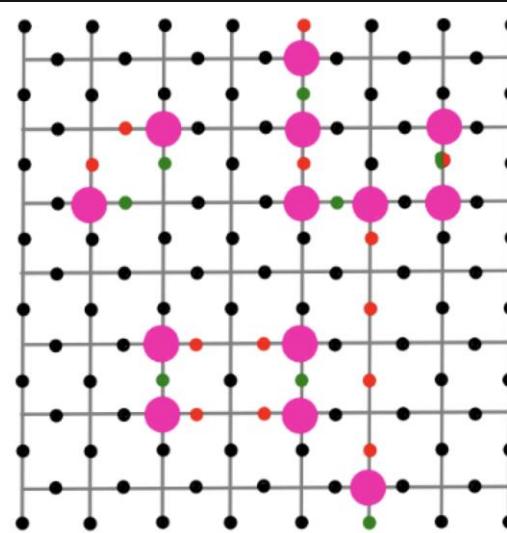
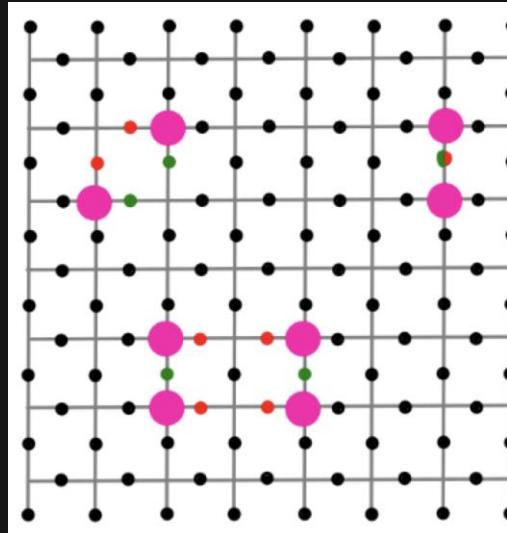
Final Readout

- The logical operators are many-body observables
- So how do we read them out fault-tolerantly
- When you decide on a basis for final measurement, you stop caring about some errors
- You can then measure in a product basis
- Final readout and final stabilizer measurement can be constructed from the result
- Measurement errors are effectively the same as errors before measurement



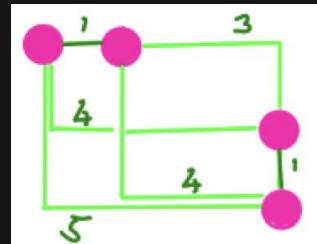
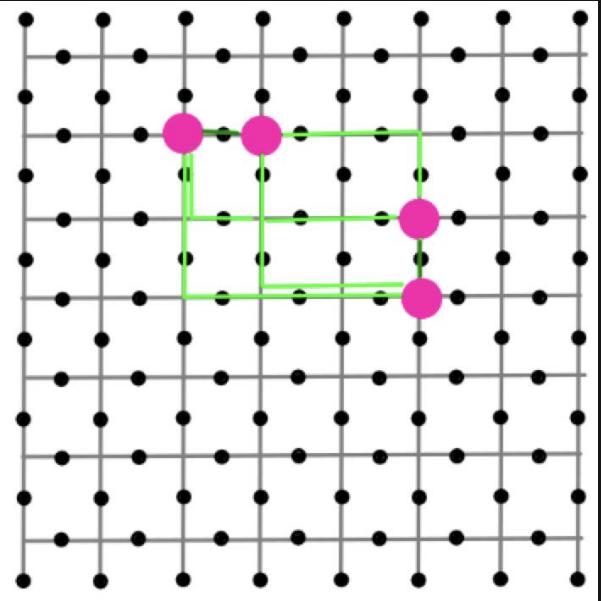
Decoding

- Given the measurement results, we need to work out what errors happened
- More specifically, the ‘equivalence class’ of errors
- This is the job of the decoding algorithm



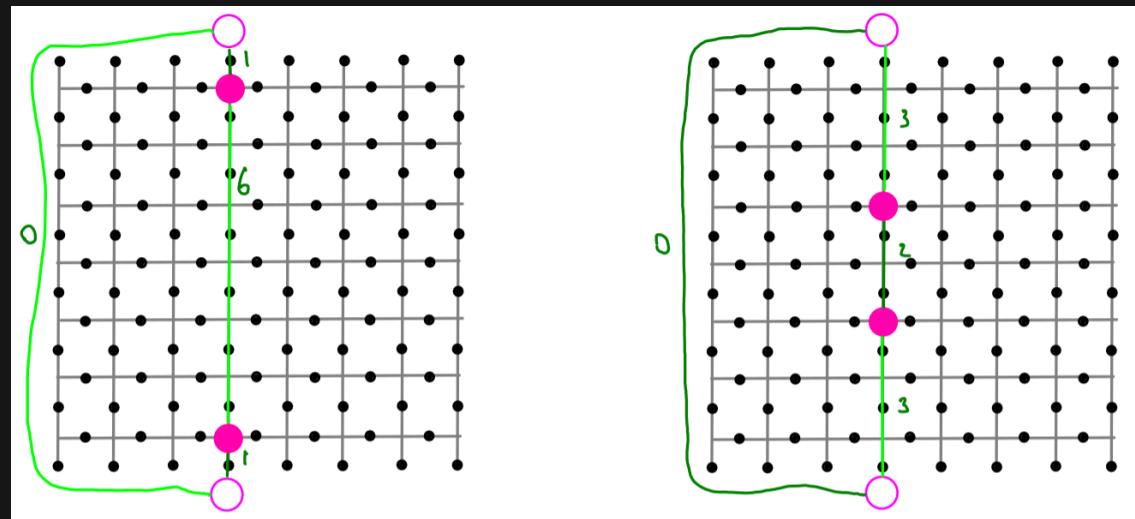
Decoding with MWPM

- A good option is MWPM
- Just as we considered for the repetition code
- Again we start with the simple and unrealistic case: errors only between measurement
- Each round can be decoded separately, corresponding to MWPM on a 2D graph
- Decoding for X and Z errors can also be done independently



Decoding

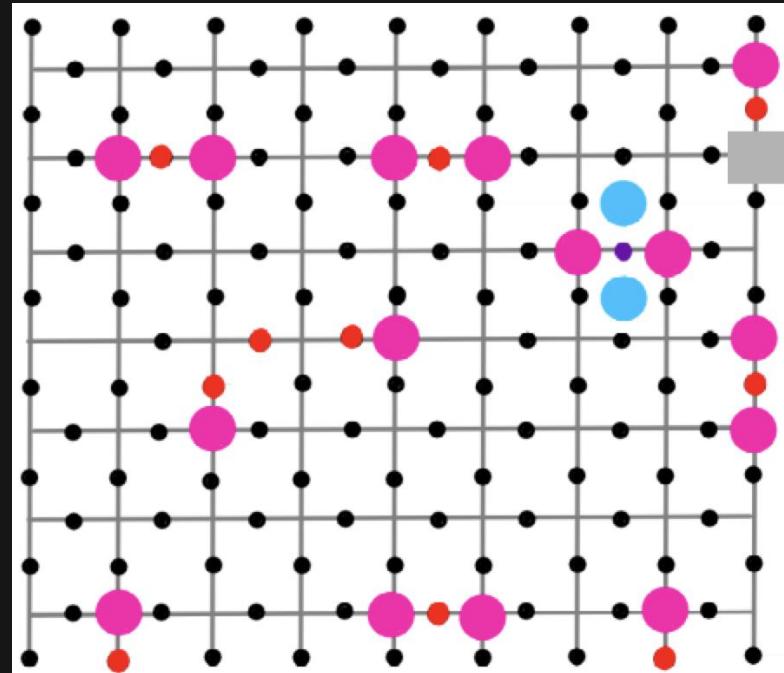
- We need to be careful to account for the effects of the edges
- This is done by introducing extra ‘virtual nodes’
(also required for the repetition code, but we ignored it earlier)



Imperfect Measurements

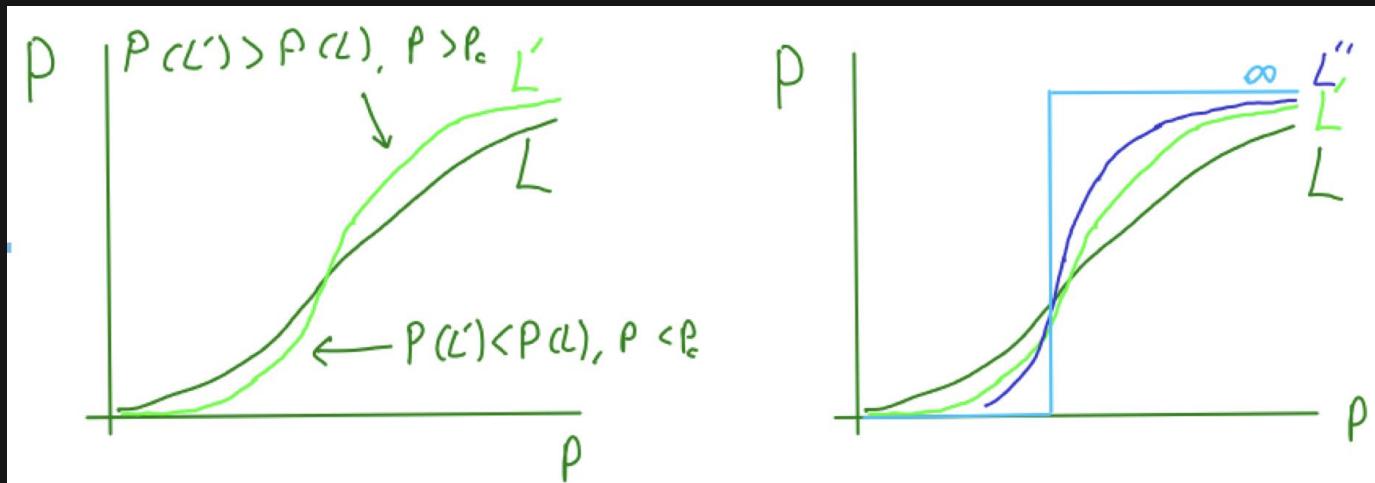
IBM Quantum

- Again, we have the problem of imperfect measurements
 - The measurements might lie
 - Errors on the additional qubit
 - Errors in the CX gates
- We base the decoding using syndrome changes
- This leads to a 3D MWPM problem (2D space + time)



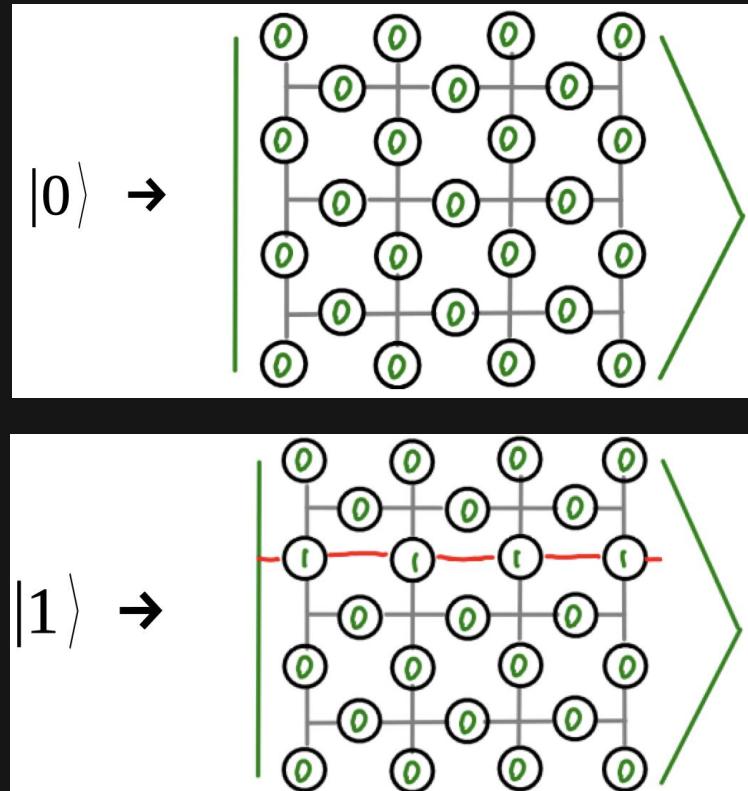
Threshold

- Correcting according to the right class removes the effects of errors
- Correcting according to the wrong class causes an operation on the encoded qubit (without our knowing)
- What is the probability of such an error, P , given the probability on the qubits of the code, p ?
- We find a phase transition as L is increased (for an $L \times L$ grid)



More Logical Gates

- We've seen how to do logical X and Z
- A logical CX can be done without much trouble
- A logical H requires the lattice to be rotated, but that can be done
- Other logical Clifford gates can be done with some crazy tricks
- But that's all! No other logical operations can be done fault-tolerantly.
- A solution is *magic state distillation*, using the logical gates we have to clean up the one we don't
- But that's beyond today's lecture...



Anyons in the Surface Code

- There are many variants on how qubits can be encoded and manipulated in the surface code
- They all depend on the unique topological nature
- The ‘defects’ created by errors in the surface code behave like particles
- All particles in our universe are either bosons or fermions
- This due to topological restrictions in a 3D universe

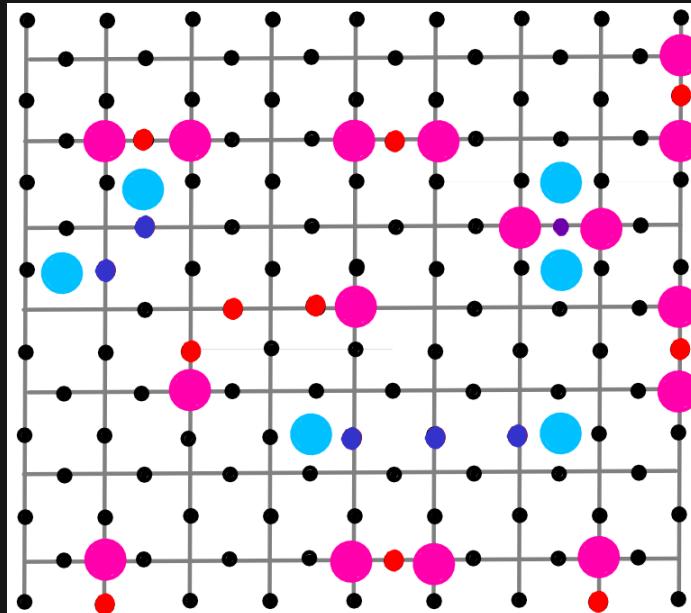
$$|\text{---} \circlearrowleft \text{---}\rangle = |\text{---} \text{---}\rangle$$
$$|\text{---} \circlearrowleft \text{---}\rangle = -|\text{---} \text{---}\rangle$$

$$|\text{---} \circlearrowright \text{---}\rangle = |\text{---} \text{---}\rangle$$

Anyons in the Surface Code

IBM Quantum

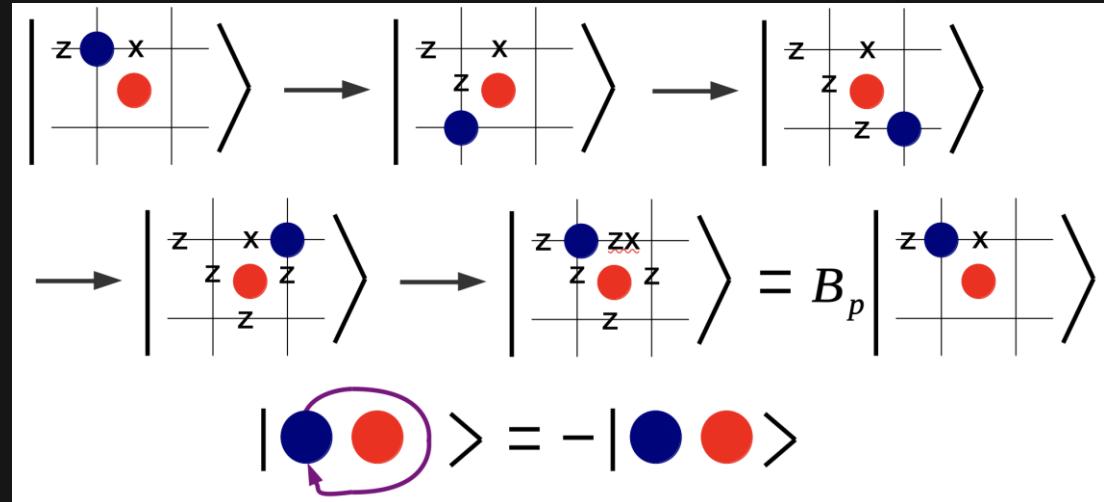
- The surface code is only a 2D ‘universe’, so doesn’t have these restrictions
- How do these particles behave?



$$| \text{blue circle} \text{ (top)} \text{ red circle (bottom)} \rangle = e^{i\theta} | \text{red circle} \text{ (top)} \text{ blue circle (bottom)} \rangle$$

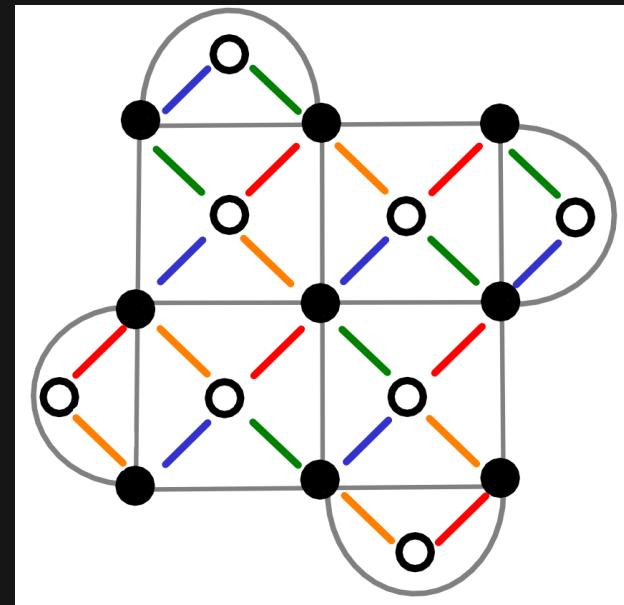
Anyons in the Surface Code

- Braiding a particle corresponds to applying a stabilizer
- Their eigenstates defines the braiding phase
- Neither bosons nor fermions, but anyons!



Minimal Implementation

- The first surface code will probably be a small one
- The smallest that allows correction is $d=3$
- This requires ~17 qubits
- Who will be among the first to run and study this code?
- **Maybe you!**



Thanks for
listening!

