Error-correction tutorial 3 : Surface code

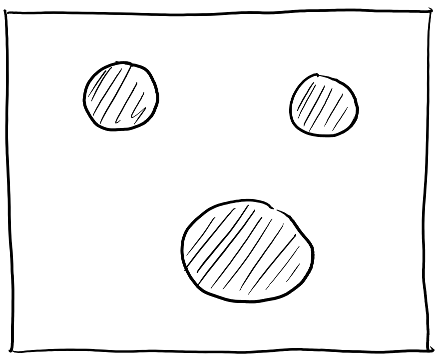
Monday, August 18, 2014 11:06 PM

THE SURFACE CODE

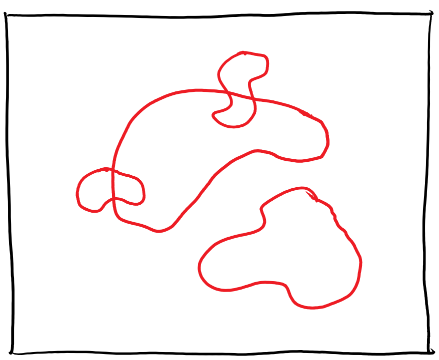
Ben Reichardt

Intuition: (scale-invariant superpositions of) String nets

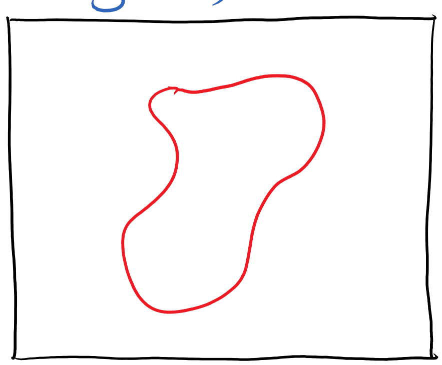
1. Start with a surface (2-manifold with boundary)



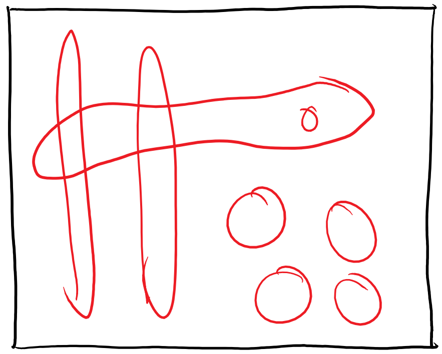
2. Draw a net on it (cycles or degree 4)



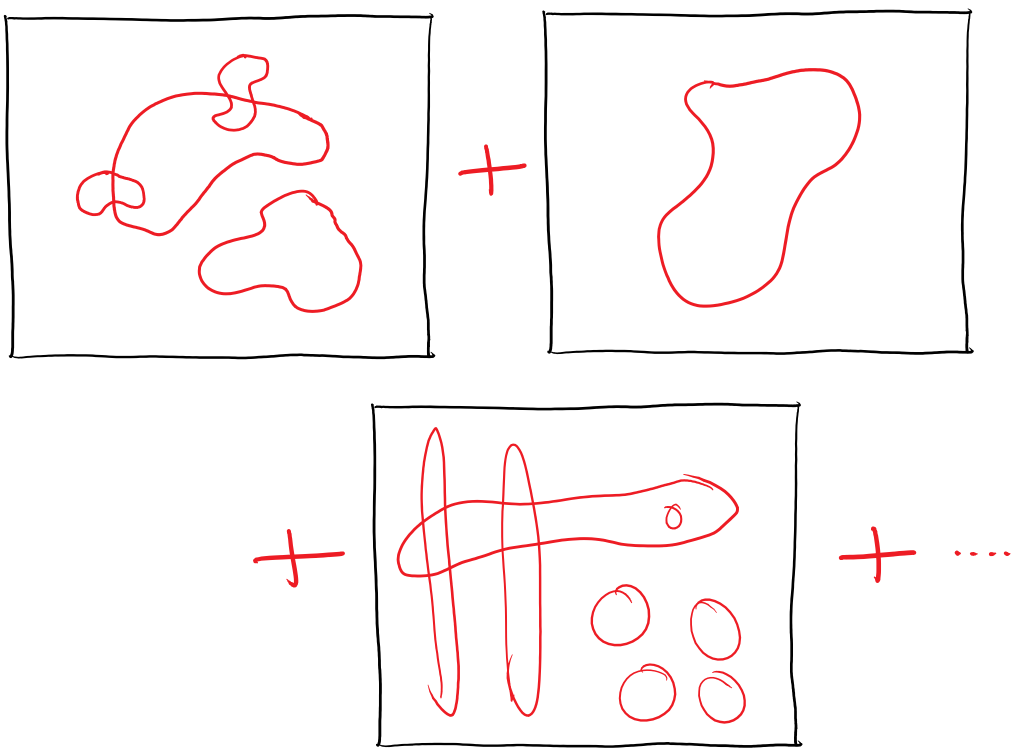
or



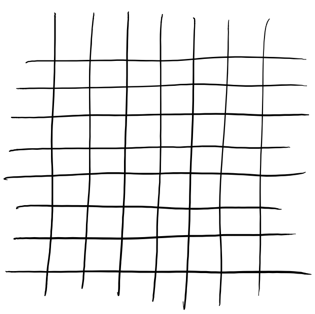
or



3. Codeword = uniform superposition over all such pictures



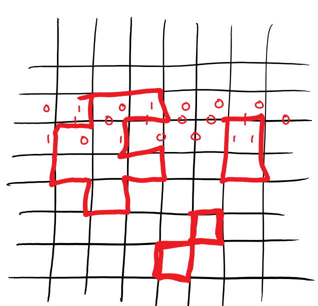
Discretization



qubit for every lattice edge

$|1\rangle$ = net edge

$|0\rangle$ = none



This gives a quantum code!

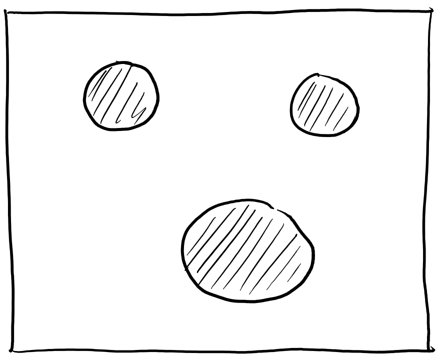
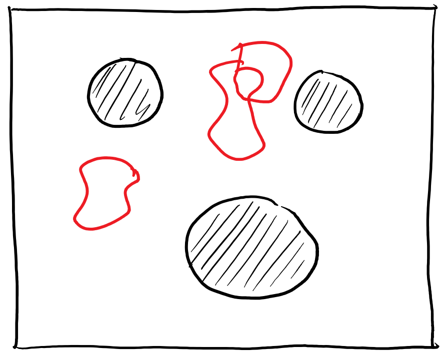
A. Codewords

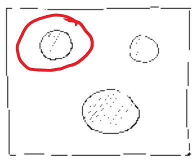
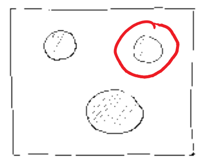
B. Protects against

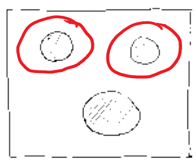
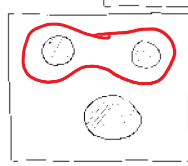
C. Stabilizers

This gives a quantum code:

A. Codewords   B. Protects against errors   C. Stabilizers

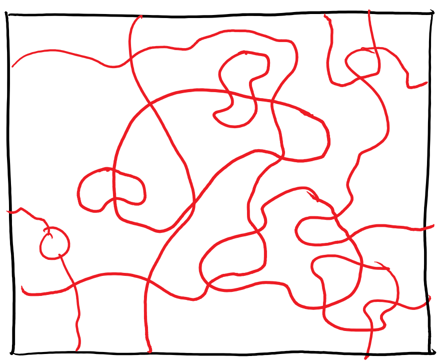
$|\text{encoded } 000\rangle =$  $+$  $+ \cdots$

$|\text{encoded } 100\rangle =$  $+ \cdots$ $|\text{encoded } 010\rangle =$  $+ \cdots$

$|\text{encoded } 110\rangle =$  $+$  $+ \cdots$

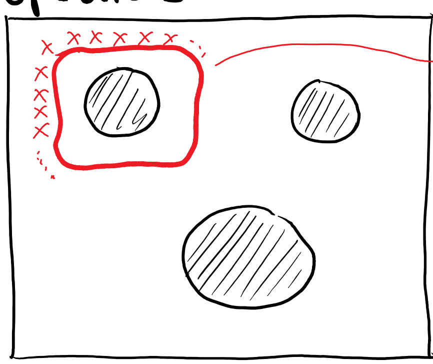
Protects against errors:

Any region just looks like



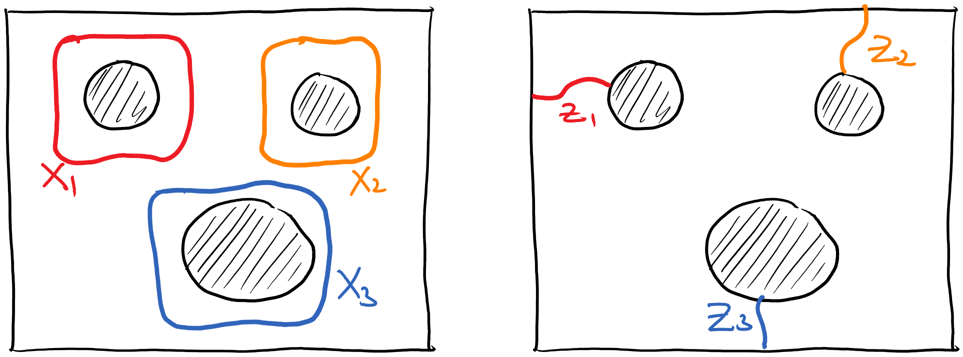
$\therefore$ you can't tell if there are an even ($|0\rangle$) or odd ($|1\rangle$) number of loops around a hole

Logical operators



logical X on first encoded qubit

(meaning $\otimes X \otimes I$ — $\otimes X$ along loop, $\otimes I$ elsewhere — switches first qubit $|0\rangle \longleftrightarrow |1\rangle$)

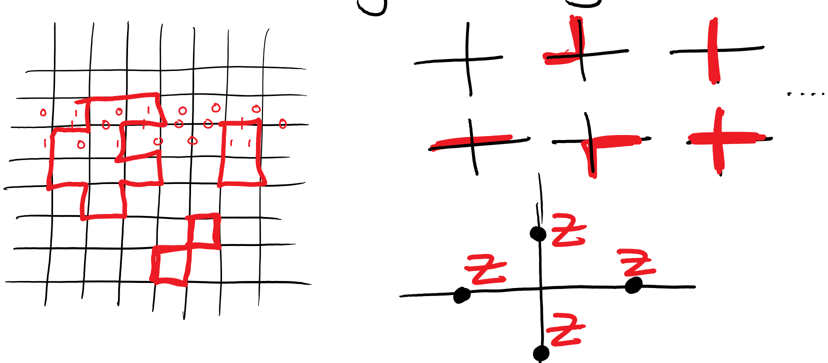


distance (minimum weight of an operator acting nontrivially on the codespace)

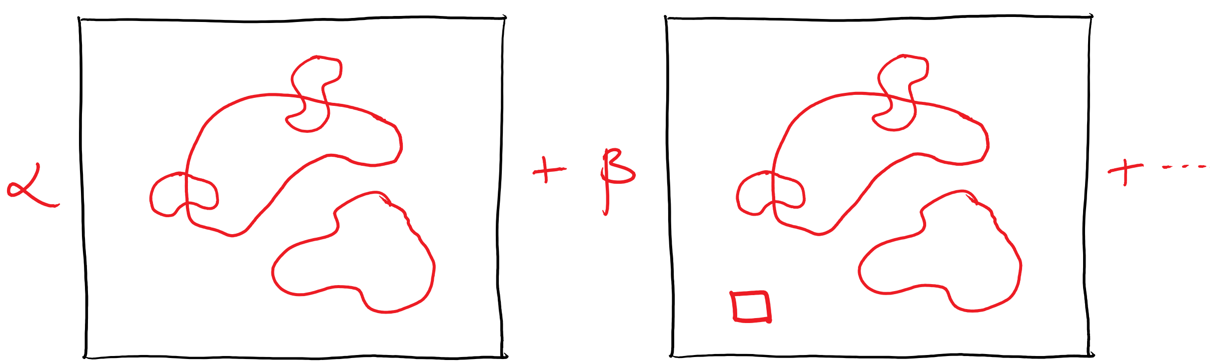
$= \min \left\{ \begin{array}{l} \text{circumference} \\ \text{of a hole} \end{array} , \begin{array}{l} \text{distance between} \\ \text{holes, or from hole} \\ \text{to boundary} \end{array} \right\}$

Stabilizers (parity checks satisfied by codewords)

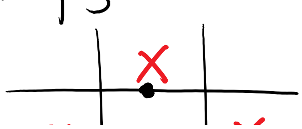
Rule 1: Even net degree at every vertex



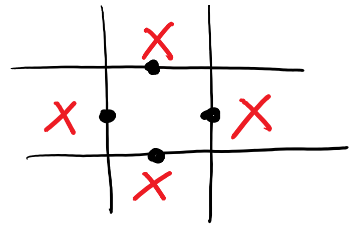
Rule 2: All cycles have equal amplitude



To force $\alpha = \beta$, use the stabilizer

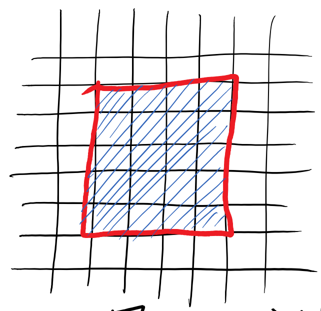


this either creates a cycle with the same

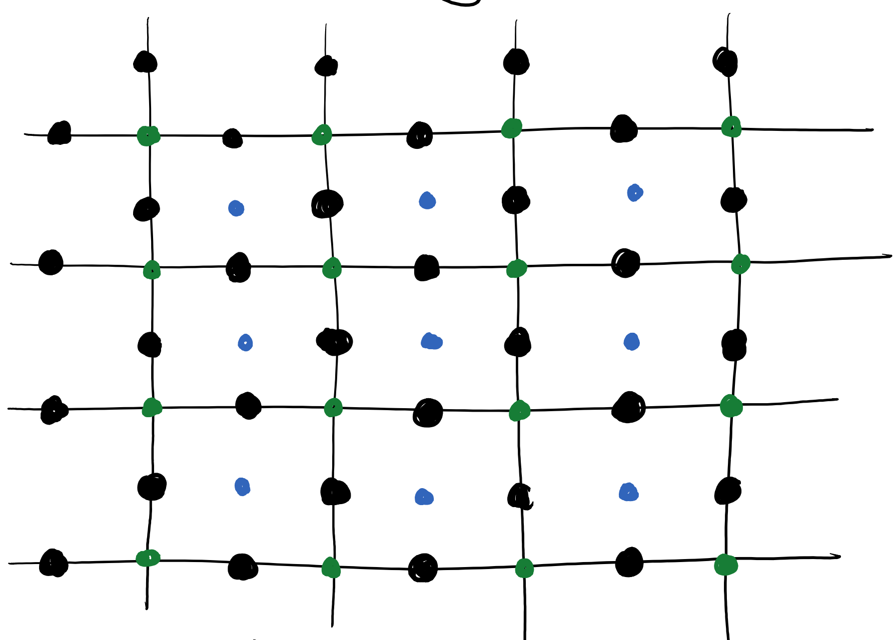


this either creates a cycle with the same amplitude, or it deforms a cycle already there

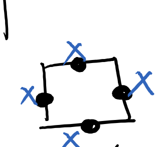
A big cycle is created by multiplying the stabilizers for the tiles it encloses.

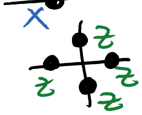


Observe: The stabilizers are local!
(codespace = ground space of local Hamiltonian)
— Important for physical implementation



● = code qubits

● = qubits used to measure  tile stabilizers

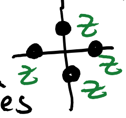
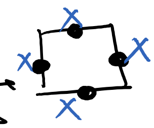
● = qubits used to measure  vertex stabilizers

(**Observe**: Mathematically, the logical operators commute with the stabilizers: this is why they leave the code unchanged.)

(**Observe**: Mathematically, the logical operators commute with the stabilizers; this is why they leave the code unchanged.

$$P\overline{X}|\psi\rangle = \overline{X}(P|\psi\rangle) = \overline{X}|\psi\rangle$$
)

Codespace = ground-space of Hamiltonian

$\mathcal{H} = -\sum_{\text{vertices}}$  $-\sum_{\text{tiles}}$ 

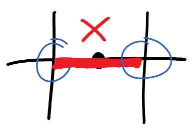
- all terms commute
- 4-local, and geometrically local in 2D

How to use this code?

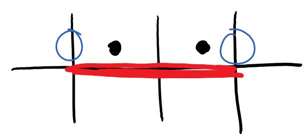
1. How to correct errors
2. How to correct errors fault tolerantly?
3. How to compute on the encoded data fault tolerantly?

Errors and error correction

X error

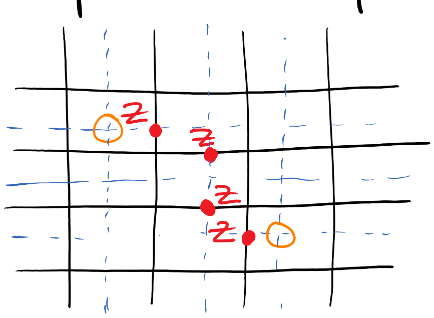


2 X errors



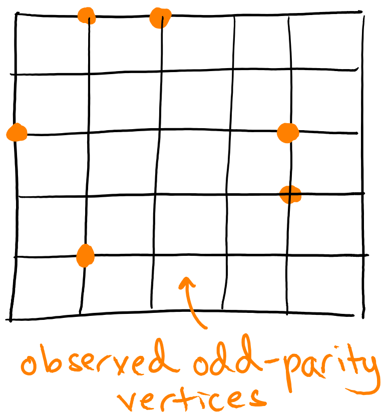
X errors create strings — undetectable in the interior, but detectable at the endpoints

Z errors can't be drawn as nets, but are completely symmetrical: chains of Z errors (on the dual lattice) show up at their endpoints

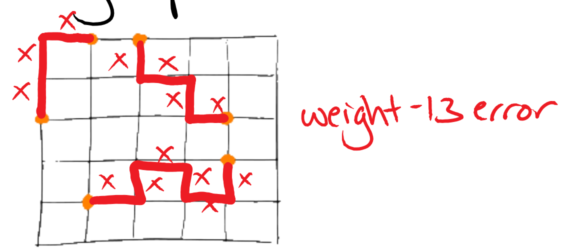


⇒ Error correction uses **minimum-weight matching**

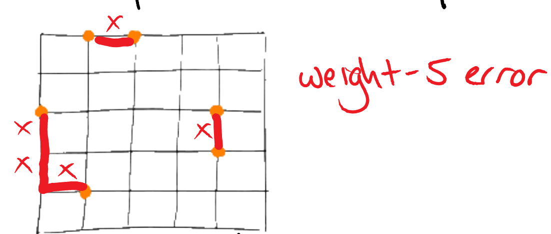
For every vertex, consider its parity (should be even)



many explanations:



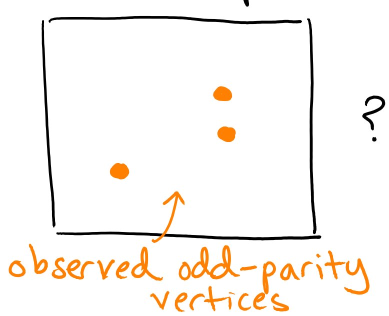
best explanation (not unique):



Can correct X and Z errors separately.
(I am ignoring some subtleties)

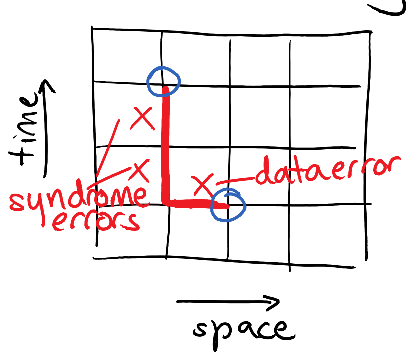
Fault-tolerant error correction

How do we explain



⇒ There must have been an error measuring the parity of a vertex

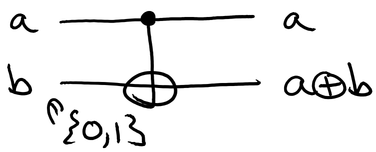
Solution: Repeat "syndrome extraction," and run matching algorithm also in time!



Error chains can grow in space and time, with syndrome flips observed at the endpoints.

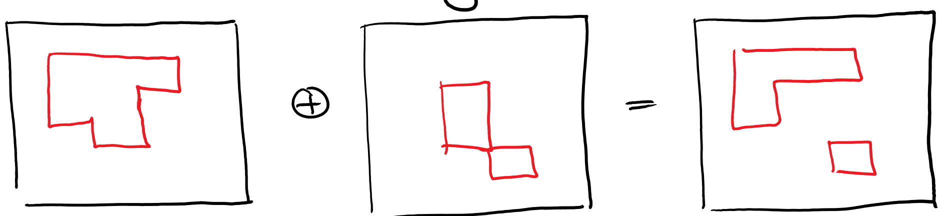
Fault-tolerant computation

Most interesting gate: CNOT (entangling)

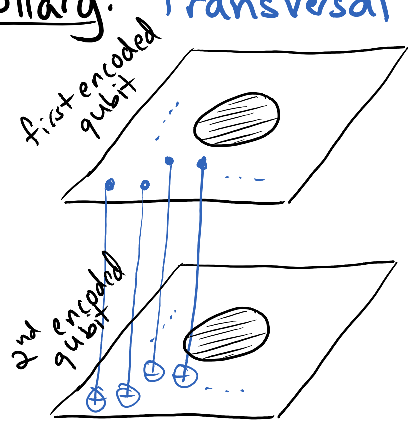


Method 1: Transversal gates

Observe: Sum mod 2 of two valid net diagrams is another valid diagram.



Corollary: Transversal CNOTs implement encoded CNOT.

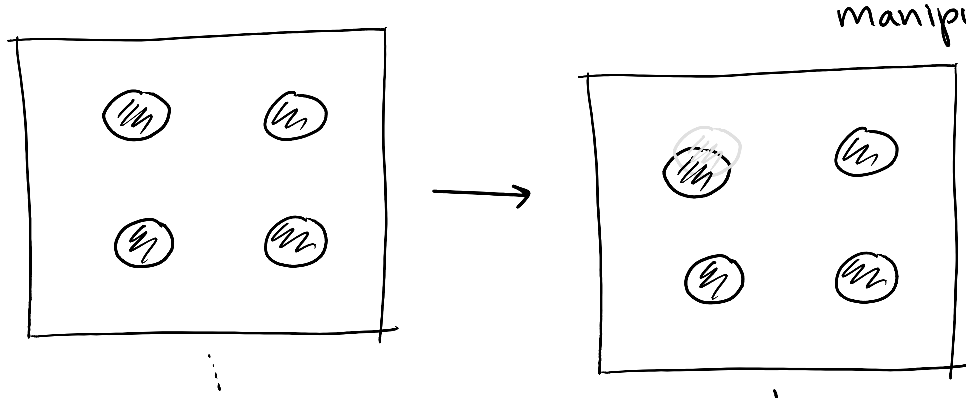


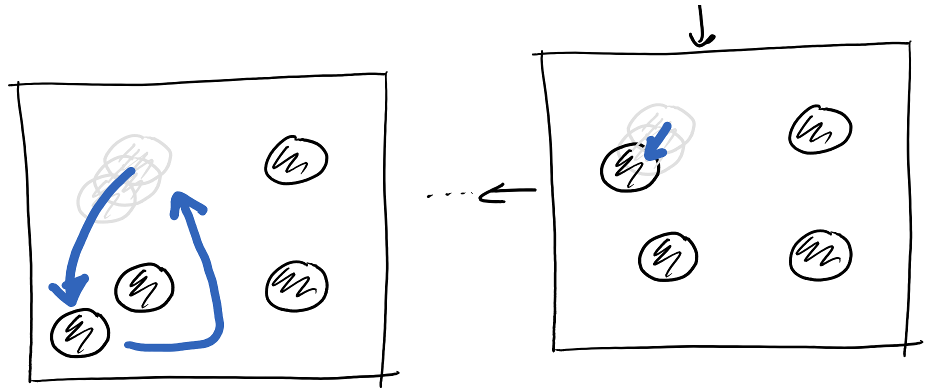
(a loop around the top hole will be copied around the bottom hole)

Method 2: Code deformation

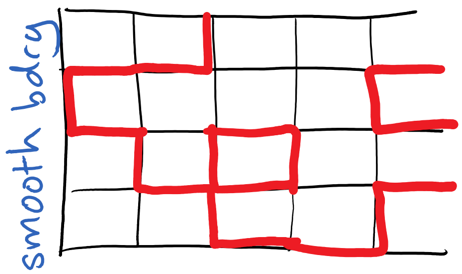
Idea:

Code → slightly different code (different surface) → Original code — but with manipulated codespace

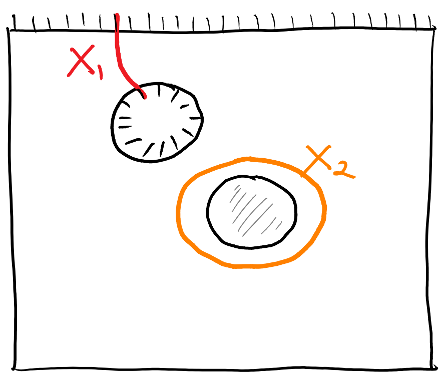


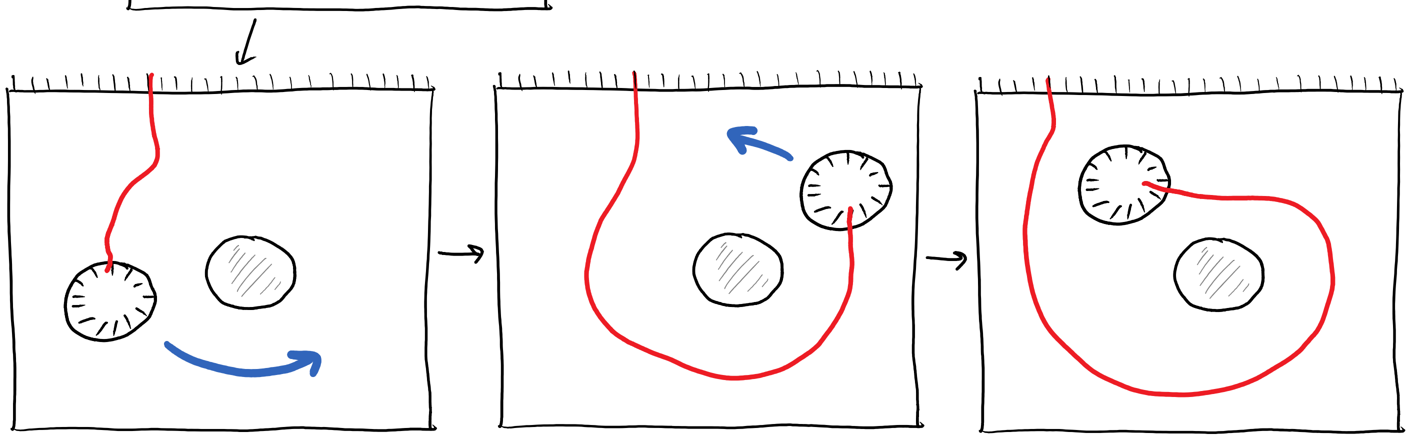


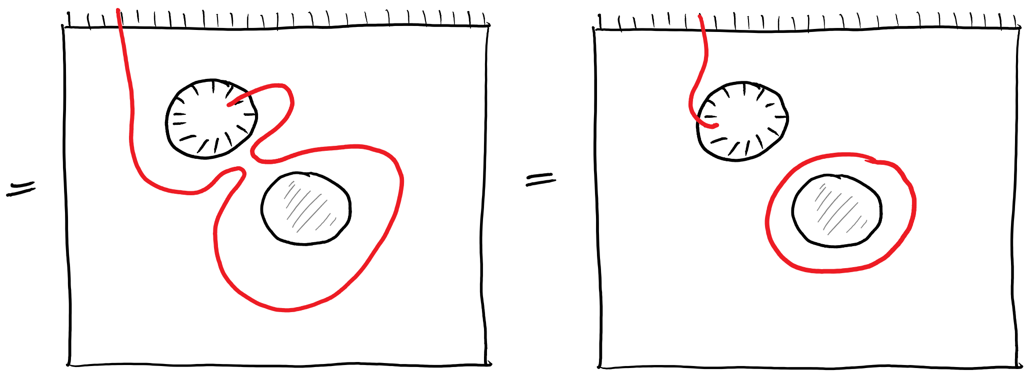
Smooth and rough boundary conditions



rough boundary
= dual of smooth bdry
— allows net lines to terminate





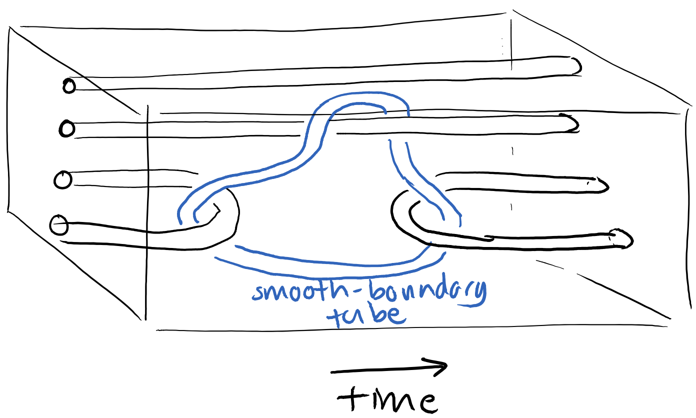


(mod 2)

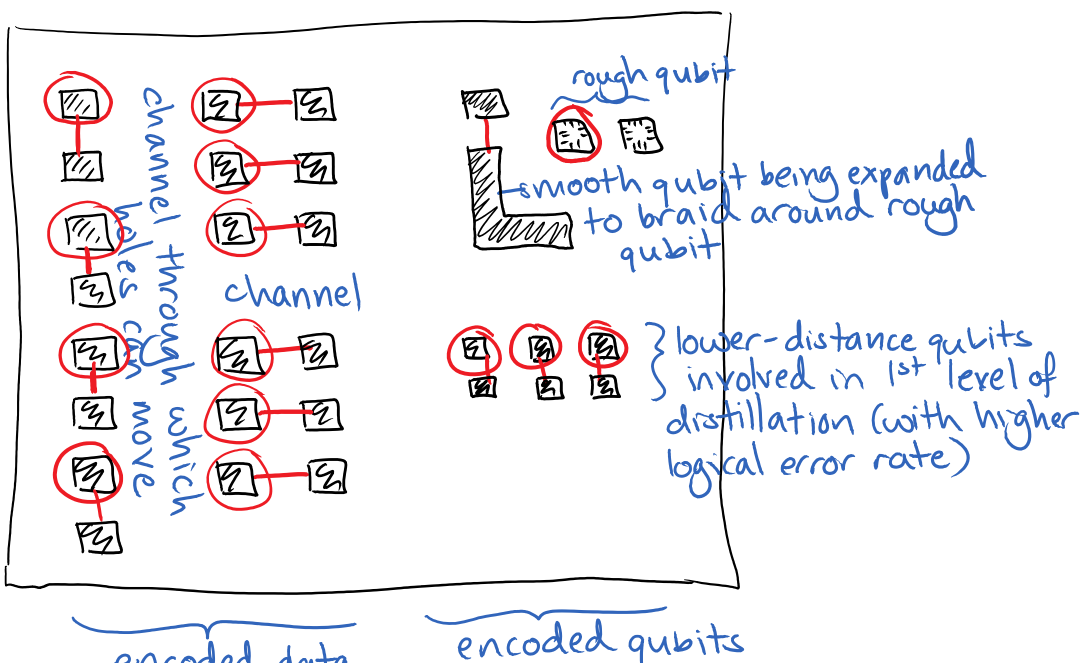
⇒ first bit (0 or 1) is XORed into 2^{nd} bit

⇒ CNOT gate ✓

Typical braiding for CNOT gate, in time



2D Architecture for a quantum computer



encoded data qubits

for distilling magic states

Elaborations:

- Surface code on other lattices
- Complexity of min-wt matching: $O(n^3)$ by Edmond
- (non-fault tolerant) Syndrome extraction
- Codewords $|\bar{0}\rangle$ and $|\bar{1}\rangle$ in the computational basis
- Obtaining a universal gate set