Ph/CS 219: Quantum Error Correction

Lecture Notes

John Preskill
Caltech

Spring 2026

These notes were transcribed by Claude from my handwritten notes.
I have not checked them in detail.

Contents

Codes from Circulant Matrices

Lecture 5 — 13 Apr 2026

Last time: limitations of local stabilizer codes. The BPT bound, $kd^2 = O(n)$ in 2D, saturated by toric. 2D local stabilizer codes have string operators $\Longrightarrow$ constant energy barrier $\Longrightarrow$ not self-correcting.

What about (CSS) codes that are *not* local? We now know such codes can be explicitly constructed with k, d much better than for local codes.

We'll develop some tools:

1. Chain complexes, boundary operators, homology.
2. Polynomial formalism for codes with translation symmetry = "quasi-cyclic" ("quasi" because circulant parity check for subblocks).

Rings, modules, ideals, group algebras, commutative algebra.

Examples: repetition code, toric code.

The repetition code and the group algebra

Rep code = 1D Ising (on a cycle) — PBC. There are n check ops, of which $n-1$ are independent:

$$Z_1Z_2,\ Z_2Z_3,\ \ldots\ ;\ Z_nZ_1.$$

E.g. $n = 4$: the cyclic shift is

$$x = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \qquad x^T = x^{-1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \qquad \begin{aligned} x &: \ 1 \to 2 \to 3 \to 4 \to 1 \\ x^{-1} &: \ 4 \to 3 \to 2 \to 1 \to 4 \\ x^4 &= \mathbb{1}. \end{aligned}$$

The rep code parity check is $H = 1 + x$,

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

or think of x as a dummy variable, $1 + x$ as a polynomial:

$$(1+x)\sum_{a=0}^{n-1} c_a x^a = \sum_a c_a\big(x^a + x^{a+1}\big) = \sum_a \big(c_a + c_{a-1}\big)x^a.$$

Now the $\{x^a\}$ coefficients in $f(x)$ represent a column vector (length n, $x^n = 1$, $c_a \in \{0,1\}$).

Think of x as the generator of $\mathbb{Z}_n$. The elements are $\{x^a,\ a = 0, 1, \ldots, n-1\}$. Then $\sum_a c_a x^a$ with $c_a \in \{0,1\}$ is an element of the group algebra over $\mathbb{F}_2$. $\mathbb{F}_2[x, x^{-1}]$ is the group algebra, $x^n = 1$: Laurent polynomials in x, x^{-1}.

For CSS codes we want to know whether the X and Z stabilizer generators commute:

$$X \text{ type:} \quad f(x) = \sum_a c_a x^a \qquad c_a = 1 \text{ where } X \text{ acts},$$

$$Z \text{ type:} \quad g(x) = \sum_b d_b x^b \qquad d_b = 1 \text{ where } Z \text{ acts},$$

$$f(x^{-1})\,g(x) = \sum_{a,b} c_a d_b\, x^{b-a} = \sum_c h_c x^c, \qquad h_c = \sum_{b-a=c} c_a d_b :$$

h_c counts the # of collisions (mod 2) when shifted by c.

Shifted checks and logical operators

Because of cyclic symmetry, if $f(x)$ is an X-type generator, so is $x^b f(x)$. We need all shifted X type to commute with all shifted Z type.

Example: the logical X operator of the rep code:

$$(1+x)\,f(x) = 0 \implies f(x) = 1 + x + x^2 + \cdots + x^{n-1}, \quad \text{i.e. } \overline{X} = (XX\cdots X).$$

What if $H_X = (1+x)$ and $H_Z = (1+x)$? Then

$$f(x^{-1})\,g(x) = (1+x^{-1})(1+x) = x + x^{-1}.$$

E.g.

$$\begin{matrix} Z & Z & I & \cdots & I \\ I & X & X & \cdots & I \end{matrix}$$

when shifted by one place left or right, ZZ and XX anticommute.

The toric code: two shifts, two registers

Toric code: now there are 2 cyclic symmetries: shift horizontally (x), shift vertically (y). And there are two registers (we can interpret them as horizontal and vertical edges).

Two registers: 2 qubits per unit cell. The unit cell is the minimal subgraph that generates a tiling of the plane when translated (applying $x^a y^b$). There are LM unit cells and $2LM$ qubits, where $x^L = 1 = y^M$.

The first register is the vertical edges, the 2nd is horizontal:

$$H_X = (1+y,\ 1+x), \qquad H_Z = (1+x^{-1},\ 1+y^{-1}) \implies H_Z^T = (1+x,\ 1+y).$$

Do they commute?

$$(1+y)(1+x) + (1+x)(1+y) = 0 \quad \checkmark$$

This is a "bivariate bicycle code" (?). Two cycles because two registers; bivariate because two shifts: x and y.

Generalize several ways:

$$H_X = \big(A(x,y) \,\big|\, B(x,y)\big), \qquad H_Z = \big(B^T(x,y) \,\big|\, A^T(x,y)\big),$$

$$H_X H_Z^T = AB + BA = 0$$

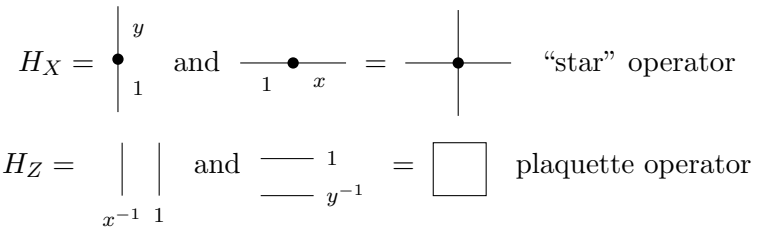
circulant matrices commute (because poly mult does!)

A special case: A, B are univariate:

$$H_X = \big(A(y) \,\big|\, B(x)\big), \qquad H_Z = \big(B^T(x) \,\big|\, A^T(y)\big)$$

these are hypergraph product codes that have cyclic symmetries.

Why is this the toric code?



Logical operators of the toric code

A Z logical is (f, g) where $(1 + y)f + (1 + x)g = 0$. Two solutions:

$$Z_H = \Big(0,\ \sum_a x^a\Big), \qquad Z_V = \Big(\sum_a y^a,\ 0\Big) :$$

vertical and horizontal string operators.

An X logical is (f, g) where $(1 + x^{-1})f + (1 + y^{-1})g = 0$. Two solutions:

$$X_H = \Big(\sum_a x^a,\ 0\Big), \qquad X_V = \Big(0,\ \sum_a y^a\Big).$$

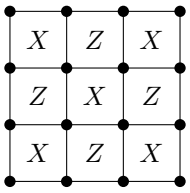
Do Z_H and X_V commute?

$$\sum_{a,b} x^a y^b.$$

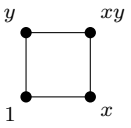
Same for Z_V and X_H: no matter how we shift them relative to one another, they <u>still</u> anticommute!

One register: the rotated surface code

An example with one register: the <u>rotated surface code</u> (checkerboard). Qubits are on sites; X, Z checks are on plaquettes:



$$H_X = H_Z = 1 + x + y + xy = (1 + x)(1 + y)$$

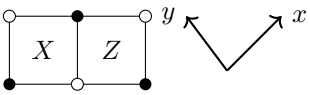


Both $\sum x^a$ and $\sum y^a$ are logical (annihilated by $1 + x$, $1 + y$ respectively). But do the Z and X checks commute?

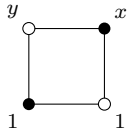
$$(1 + x^{-1})(1 + x)\,(1 + y^{-1})(1 + y) = (x + x^{-1})(y + y^{-1}) :$$

they <u>don't</u> commute when shifted both V and H.

Actually, the unit cell is: there are *two* plaquettes per unit cell. The two registers are two checkerboards of vertices; x and y are primitive lattice translations:



$$H_X = (1+y,\ 1+x), \qquad H_Z = (1+x^{-1},\ 1+y^{-1})$$

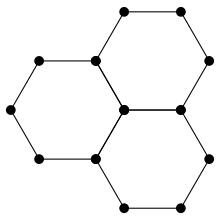


The codes have the *same* polynomial presentation, yet $n = d^2$ for rotated and $n = 2d^2$ for unrotated. Unrotated: 2 $d \times d$ registers, hence $n = 2d^2$. Rotated: now there are $d^2/2$ unit cells and two qubits per cell $\Longrightarrow n = d^2$.

Another way to say this: for the rotated code, x and y are 45° rotated. We have $x^d = y^d = 1$. The unit cell has area = 2 vs area = 1 for unrotated, yet the area of the torus is the same.

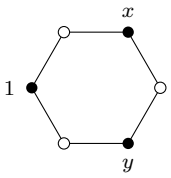
The color code on a honeycomb

Color code on a honeycomb: the checks are $X^{\otimes 6}$ and $Z^{\otimes 6}$ on each hexagon. We can formulate it using either 2 or 3 registers.



Two-coloring of the vertices $\Longrightarrow$ two registers:

$$H_X = H_Z = \big(1 + x + y,\ 1 + x^{-1} + y^{-1}\big)$$



Check commutation:

$$0 = \big(1 + x^{-1} + y^{-1}\big)\big(1 + x + y\big) + \big(1 + x + y\big)\big(1 + x^{-1} + y^{-1}\big).$$

We can map to 2 copies of the toric code on 4 registers (homework).

Beyond local codes: the Gross code

So far the codes are local: only small powers of x and y. Allowing larger powers can make the codes more efficient.

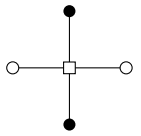
Example: the Gross code, $[[144, 12, 12]]$ — cf. the 12×12 toric $[[144, 2, 12]]$ (or just $k = 1$ for the surface code).

$$H_X = (A \,|\, B), \qquad H_Z = (B^T \,|\, A^T). \qquad \text{Now} \quad \begin{matrix} A(x,y) = x^3 + y + y^2 \\ B(x,y) = y^3 + x + x^2 \end{matrix}$$

Relabel:

$$A' = y^{-1}A = y^{-1}x^3 + 1 + y,$$
$$B' = x^{-1}B = x^{-1}y^3 + 1 + x.$$

$x^{12} = 1 = y^6 \Longrightarrow$ two registers, each w/ 72 qubits. Each check is weight 6:



4 neighbors
as for toric code

And two longer connections: either 3 steps horizontal and 1 vertical, or 3 steps vertical and 1 horizontal.

Aside: circulant matrices and polynomials

Circulant means all columns are obtained from the first by cyclic permutations:

$$C = \begin{pmatrix} c_0 & c_{n-1} & \cdots & c_1 \\ c_1 & c_0 & \cdots & c_2 \\ c_2 & c_1 & & \vdots \\ \vdots & \vdots & \ddots & \\ c_{n-1} & c_{n-2} & \cdots & c_0 \end{pmatrix} = c_0\mathbb{1} + c_1 x + c_2 x^2 + \cdots + c_{n-1}x^{n-1}.$$

Or — for the transpose:

$$C^T = \begin{pmatrix} c_0 & c_1 & c_2 & \cdots & c_{n-1} \\ c_{n-1} & c_0 & c_1 & \cdots & c_{n-2} \\ c_{n-2} & c_{n-1} & c_0 & \cdots & \vdots \\ \vdots & & & \ddots & \end{pmatrix} = \sum_a c_a \big(x^{-1}\big)^a.$$

Hence

$$C \cdot D = \Big(\sum_a c_a x^a\Big)\Big(\sum_b d_b x^b\Big) = \text{product of polynomials.}$$

Polynomials live in the ring $\mathbb{F}_2[x]/(x^n - 1)$. We consider

$$\mathbb{F}_2[x, y]/\big(x^L - 1,\ y^M - 1\big)$$

— its parity check is invariant under 2 independent shift symmetries. Again the matrices can be expanded in powers $\{x^a y^b\}$, and matrix multiplication is the same as polynomial multiplication.

Bivariate Bicycle Codes

Lecture 6 — 15 Apr 2026

Reference: Bravyi et al., arXiv: 2308.07915.

Circulant matrices and the polynomial ring

Shift symmetry: each row (or column) is obtained by shifting the previous one by one place,

$$C = \begin{pmatrix} c_0 & c_2 & c_1 \\ c_1 & c_0 & c_2 \\ c_2 & c_1 & c_0 \end{pmatrix}, \qquad C^T = \begin{pmatrix} c_0 & c_1 & c_2 \\ c_2 & c_0 & c_1 \\ c_1 & c_2 & c_0 \end{pmatrix},$$

$$x = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \qquad x^T = x^{-1} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix},$$

$$C = c_0 X^0 + c_1 x^1 + c_2 x^2, \qquad C^T = c_0 (x^{-1})^0 + c_1 (x^{-1})^1 + c_2 (x^{-1})^2.$$

The matrix is completely determined by its first row or column.

$x^a x^b = x^{a+b}$ rule for multiplying shifts or monomials. Commutative!

An $n \times n$ matrix $\Rightarrow x^n = 1$. Our check matrices have entries $\in \{0, 1\}$

$\implies$ ring of polynomials $\mathbb{F}_2[x] = \mathbb{F}_2[x, x^{-1}]$ where $x^n = 1$,

$\mathbb{F}_2[x]/(x^n - 1)$ Finite ring: we can multiply and add.

We can also represent the action of a circulant matrix on a vector as multiplication of polynomials. Now x is a "dummy variable" — not a shift:

$$v = \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix} = v_0 + v_1 x + v_2 x^2 \implies xv = v_2 + v_0 x + v_1 x^2 = \begin{pmatrix} v_2 \\ v_0 \\ v_1 \end{pmatrix},$$

$$x^{-1} v = v_1 + v_2 x + v_0 x^2 = \begin{pmatrix} v_1 \\ v_2 \\ v_0 \end{pmatrix}.$$

Parity checks with shift symmetry

I can write a parity check matrix with shift symmetry as $H = f(x)$,

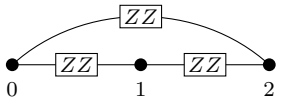
$$f(x) = c_0 + c_1 x + c_2 x^2 + \cdots$$

$(c_0, c_1, c_2, \ldots)$ has two interpretations: a single parity check (vector), or the full matrix (obtained by cyclic shifts).

$x^T = x^{-1} \implies f(x)^T = f(x^{-1})$ interpreting $f(x)$ as a matrix.

Example: repetition code.

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



The checks are redundant (only 2 are linearly independent).

If $H = f(x)$ and $v = g(x)$, then

$$Hv = 0 \iff f(x)\, g(x) = 0.$$

Here $(1 + x)\, g(x) = 0$; the solutions are

$$g(x) = 0 \Rightarrow v = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \qquad g(x) = 1 + x + x^2 \Rightarrow v = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

For CSS codes, the X and Z checks commute:

$$H_X H_Z^T = 0, \quad H_Z H_X^T = 0 \qquad \text{the rows of } H_X \perp \text{ to rows of } H_Z.$$

Take a single X and Z check (one row of H_X, H_Z); they should be orthogonal when shifted by any amount. $H_X = f(x)$, $H_Z = g(x) \Longrightarrow$

$$H_X H_Z^T = f(x)\, g(x)^T = f(x)\, g(x^{-1}) = 0,$$

$$f(x)\, g(x^{-1}) = \Big(\sum_a c_a x^a\Big)\Big(\sum_b d_b x^{-b}\Big) = \sum_c h_c x^c,$$

$$h_c = \sum_{a-b=c} c_a d_b = \#\text{ of collisions when shifted by } c.$$

More than one register: the toric code

We can have more than one cyclic symmetry and more than one qubit register. Toric code:

$$\sigma(x, y) = \begin{pmatrix} 1 + y & 1 + x \\ 1 + x^{-1} & 1 + y^{-1} \end{pmatrix} \begin{matrix} \Longleftarrow H_X \\ \Longleftarrow H_Z \end{matrix} \qquad \text{(columns: register 1, register 2).}$$

An $L \times M$ torus $\Rightarrow x^L = 1 = y^M$. The ring is

$$\mathbb{F}_2[x, y] / \big(x^L - 1,\ y^M - 1\big).$$

Register 1 is the vertical edges; register 2 is the horizontal edges.

Logical operator: think of an X-type Pauli as a vector — a string of I and X. It commutes with Z-type checks. This means the first row of H_Z and all shifts of it annihilate the X-type vector.

$$H = f(x) = \sum_a f_a x^a = \begin{pmatrix} f_0 & f_2 & f_1 \\ f_1 & f_0 & f_2 \\ f_2 & f_1 & f_0 \end{pmatrix} \implies H_{ij} = f_{j-i},$$

$$v(x) = \sum_b v_b x^b, \qquad (Hv)_i = \sum_j H_{ij}\, v_j = \sum_j f_{j-i}\, v_j$$

condition derived earlier: $h(x^{-1})\, v(x) = 0$

For the rep code, $(1 + x^{-1})\, v(x) = 0$:

$$v(x) = 1 + x + x^2 + \cdots + x^{n-1}, \qquad \overline{X} = X \cdots X.$$

Same for the toric code: a Z logical is (f, g) where

$$(1 + y^{-1})\, f + (1 + x^{-1})\, g = 0 : \qquad \Big(0,\ \sum_a x^a\Big) \ \text{ or } \ \Big(\sum_a y^a,\ 0\Big).$$

Bivariate bicycle codes

More general:

$$\begin{pmatrix} A(x,y) & B(x,y) \\ B(x,y)^T & A(x,y)^T \end{pmatrix} \begin{matrix} \Longleftarrow H_X \\ \Longleftarrow H_Z \end{matrix} \qquad \text{(columns: register 1, register 2).}$$

The checks commute:

$$H_X H_Z^T : \quad AB + BA = 0 \qquad \text{(circulant polynomials commute — poly in } x \text{ and } y\text{)}.$$

These are

$$\underbrace{\text{bivariate}}_{\text{2 shifts}}\ \underbrace{\text{bicycle}}_{\text{2 registers}}\ \text{codes.}$$

If the polynomials have finite order as n grows, a local code. If A is a function of x only and B of y only, a hypergraph product. Higher order polynomials $\to$ longer distance checks $\to$ opportunity to improve $[[n, k, d]]$.

Example: the Gross code, $[[144, 12, 12]]$. Cf. the rotated surface code: $[[144, 1, 12]]$.

The Gross code layout

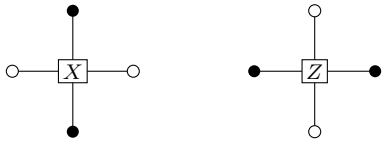
$$A(x,y) = x^3 + y + y^2, \qquad B(x,y) = y^3 + x + x^2, \qquad x^{12} = 1 = y^6.$$

We can lay it out on a torus, but with some long connections. Relabel qubits:

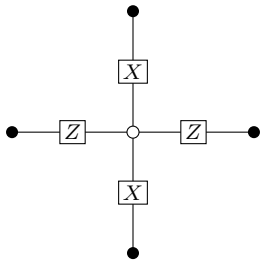
$$A' = y^{-1}A = y^{-1}x^3 + 1 + y,$$
$$B' = x^{-1}B = x^{-1}y^3 + 1 + x.$$

The two registers are 2 12×6 lattices. All checks are weight 6. Four Paulis are neighbors of the check, as for the toric code; two Paulis are farther from the check qubits.

For the toric code:



x, y shift each sublattice (colored and uncolored); uncolored is first register and colored is second.



In addition, 2 more qubits are in each check. The X check includes:

$$\begin{aligned} \text{1st register (uncolored):} &\quad 3 \text{ steps right} + 1 \text{ down}, \\ \text{2nd register (colored):} &\quad 3 \text{ steps up} + 1 \text{ left}. \end{aligned}$$

The Z check:

$$\begin{aligned} \text{1st:} &\quad 3 \text{ down} + 1 \text{ right}, \\ \text{2nd:} &\quad 3 \text{ left} + 1 \text{ up}. \end{aligned}$$

Counting logical qubits

How to determine $k = \#\, X$ logicals $= \#\, Z$ logicals? We have $n = 144$ qubits, 72 X checks, 72 Z checks. But the checks are not independent. The X logicals are

$$\mathcal{L}_X = \frac{\overbrace{\mathrm{Ker}(H_Z)}^{\text{satisfies checks}}}{\underbrace{\mathrm{Im}(H_X^T)}_{\text{in stab.}}}, \qquad k = \dim \mathrm{Ker}(H_Z) - \mathrm{rank}(H_X^T).$$

Redundancy will increase the kernel and diminish the rank.

Translation invariance means: find solutions to $A = B = 0$ using the discrete Fourier transform (reciprocal lattice). Now the shifts become phases:

$$x = e^{i\theta_x}, \quad \theta_x = \frac{2\pi}{L_x} k; \qquad y = e^{i\theta_y}, \quad \theta_y = \frac{2\pi}{L_y} \ell.$$

$$A = x^3 + y + y^2 = 0, \qquad B = y^3 + x + x^2 = 0,$$

$$x^{12} - 1 = 0 = (x^3 - 1)^4, \qquad y^6 - 1 = 0 = (y^3 - 1)^2, \qquad x^3 = y^3 = 1.$$

We can look for solutions with $x^3 = 1 = y^3$

$$\implies \quad 1 + y + y^2 = 0 = 1 + x + x^2.$$

The solutions are ω and ω^2:

$$\omega = e^{2\pi i/3} = -\tfrac{1}{2} + i\tfrac{\sqrt{3}}{2}, \qquad \omega^2 = e^{4\pi i/3} = -\tfrac{1}{2} - i\tfrac{\sqrt{3}}{2}.$$

4 values solve the equations:

$$(\omega, \omega), \quad (\omega, \omega^2), \quad (\omega^2, \omega), \quad (\omega^2, \omega^2).$$

But (ω, ω^2) and (ω^2, ω) are doubly degenerate, so 6 solutions and $k = 12$.

Another way to see 2 logicals per solution:

$$A = B = 0 \implies Af + Bg = 0 \text{ for } (f, 0) \text{ and } (0, g).$$

Homework: use the inverse F.T. to reconstruct the logicals as Paulis.

$d = 12 \implies$ need to show there is no nontrivial logical of lower weight.

From BB codes to the hypergraph product

The BB code is one way to generalize the toric code. The hypergraph product code is another (HGP code). Toric is the HGP of two rep codes. We can take the HGP of *any* two classical linear codes (they don't need to be shift invariant) — the two register "picture."

We use a code and its transpose code (distinct from its dual): C has parity check H and C^T has parity check H^T.

Example of the rep code: $H = 1 + x$ and $H^T = 1 + x^{-1}$ (same code). Important that H is redundant:

$$H = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}, \qquad H^T = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \qquad \underline{\text{same}} \text{ checks.}$$

But

$$H = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \implies H^T = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \qquad \underline{\text{no solutions}}.$$

Again: 2 registers, A is $r_1 \times n_1$, B is $r_2 \times n_2$. A, B act on distinct registers in H_X and H_Z:

$$H_X = \big(A \otimes I_{n_2},\ I_{r_1} \otimes B^T\big), \qquad H_Z = \big(I_{n_1} \otimes B,\ A^T \otimes I_{r_2}\big)$$

Toric code:
a grid of vertical edges,
a grid of horizontal edges.

The number of qubits is $n_1 n_2 + r_1 r_2$ (the lengths of rows in X and Z checks). How many constraints (# of rows)? $r_1 n_2$ (H_X) and $n_1 r_2$ (H_Z):

$$k \ \geq\ n_1 n_2 + r_1 r_2 - r_1 n_2 - n_1 r_2 = (n_1 - r_1)(n_2 - r_2) \sim k_1 k_2.$$

The key point is

$$H_X H_Z^T = A \otimes B^T + A \otimes B^T = 0 :$$

we've arranged the # of collisions to always be even.

$$d = \min\big(d_A,\, d_B,\, d_{A^T},\, d_{B^T}\big).$$

E.g. a codeword annihilated by A is annihilated by $A \otimes I$, etc. We want the codes A, B, A^T, B^T to have high rate and distance:

$$k_1, d_1 = \Omega(n_1), \qquad k_2, d_2 = \Omega(n_2) \qquad \underline{\text{LDPC codes}}.$$

$$\text{Rate} \sim \frac{k_1 k_2}{n_1 n_2} = \Omega(1). \quad \text{But distance} \sim n_1, n_2 \sim \sqrt{n} \ = \text{the "square root barrier."}$$

Hypergraph Product Codes

Lecture 7 — 20 Apr 2026

Reference: Tillich & Zémor, arXiv: 0903.0566.

From any two classical codes, construct a CSS code. Why "hypergraph"? A graph $G = (V, E)$ is a set of vertices, and a set of edges = pairs of vertices. In a hypergraph, elements of E are not pairs of vertices, but arbitrary subsets of V.

Codes and hypergraphs

A parity check matrix H defines a hypergraph. If H is $r \times n$, each of the r rows defines a subset of the n bits (vertices), those where the matrix has 1 rather than 0 (r subsets).

Another hypergraph: each of the n columns defines a subset of the r checks (vertices), those where the matrix has 1.

One hypergraph: sets of bits participating in a check. Other hypergraph: sets of checks involving a bit.

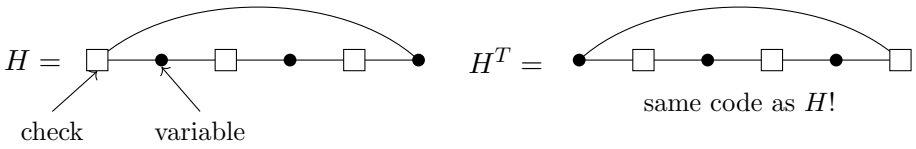
The hypergraph product puts together the hypergraphs of two classical codes to get a new code.

The transpose code and the Tanner graph

The construction of the HGP uses the concept of the *transpose* of a code. If C has parity check H, then C^T has parity check H^T. The roles of checks and bits are interchanged. C^T depends not just on C, but also on how C is described by H.

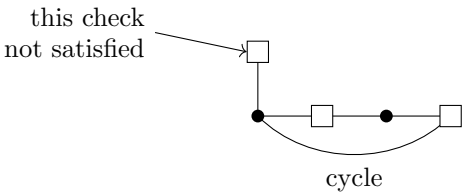
A nice way to visualize the transpose: the Tanner graph. This is a bipartite graph, with two types of vertices — variable nodes and check nodes. Each edge connects a variable node to a check node. Each check node connects to all bits in the check: a row of H. Each variable node connects to all checks it participates in: a column of H. Under $H \to H^T$, the variable nodes become checks, and the check nodes become variables.

Example: repetition code on a cycle.



What if not "periodic boundary condition"?

Remark: the Tanner graph might have short cycles, but these need not correspond to low-weight codewords.



all checks and variables have constant degree $\longrightarrow$ LDPC

The HGP construction

As for BB codes, there are two qubit registers, and the code is defined by two parity check matrices:

$$A \text{ is } r_1 \times n_1 \qquad \text{and} \qquad B \text{ is } r_2 \times n_2.$$

These act on the two registers in X and Z checks according to

$$H_X = \big(\underbrace{A \otimes I_{n_2}}_{\substack{\text{register 1:} \\ n_1 n_2 \text{ qubits}}} , \underbrace{I_{r_1} \otimes B^T}_{\substack{\text{register 2:} \\ r_1 r_2 \text{ qubits}}} \big), \qquad H_Z = \big(I_{n_1} \otimes B,\ A^T \otimes I_{r_2} \big).$$

Both registers are arranged in a grid, with "horizontal" and "vertical" directions. In H_X, the A check acts on qubits in the same "row" and B^T on qubits in the same "column" (this row and column characterize the grid, not the parity check matrix). Furthermore, A acts on the first register and B^T on the second register. The H_Z matrix is chosen so that for each collision between H_X and H_Z in register 1, there is a corresponding collision in register 2. Therefore the number of collisions is even, and the checks commute:

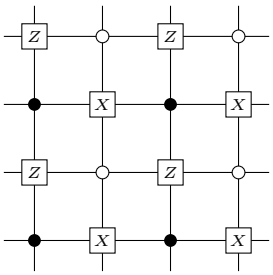
$$H_X H_Z^T = \underbrace{A \otimes B^T}_{\text{1st register}} + \underbrace{A \otimes B^T}_{\text{2nd register}} = 0.$$

Example: the 2×2 toric code

The 2×2 toric code is the HGP of two $n = 2$ repetition codes, with

$$H = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = H^T$$

(Tanner graph, periodically identified):



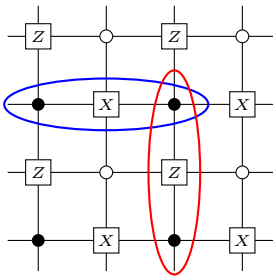
2×2 grid of black nodes
2×2 grid of white nodes

A : X checks couple black nodes in same row of grid

B^T : X checks couple white nodes in same column

B : Z checks couple black nodes in same column

A^T : Z checks couple white nodes in same row



By construction, if X and Z checks collide at a black node, there is also a corresponding collision on a white node.

Code parameters

In the HGP code, the length n is the number of columns in H_X (or H_Z). This is

$$n = n_1 n_2 + r_1 r_2 \qquad \begin{array}{l}\text{variable-variable register w/ } n_1 n_2 \\ \text{check-check register w/ } r_1 r_2\end{array}$$

How many X checks? The number of rows in H_X, $= r_1 n_2$. How many Z checks? The number of rows in H_Z, $= n_1 r_2$. Therefore, the number of logical qubits is

$$k \geq \underbrace{n_1 n_2 + r_1 r_2}_{n} - \underbrace{(n_2 r_1 + n_1 r_2)}_{\#\text{ of checks}} = (n_1 - r_1)(n_2 - r_2).$$

k can be larger if checks are redundant. E.g., for the toric code = HGP of rep codes with $n = r$ but 1 redundant check; we actually have $k = 2$.

If A and B both have full rank (linearly independent rows), they are parity checks for codes with $k_1 = (n_1 - r_1)$ and $k_2 = (n_2 - r_2)$, and we have $k = k_1 k_2 \implies$ the code rate is

$$R = \frac{k_1 k_2}{n_1 n_2 + r_1 r_2} \geq \frac{k_1 k_2}{2 n_1 n_2} = \tfrac{1}{2} R_1 R_2.$$

The HGP of two constant-rate (= "high-rate") codes is a high-rate quantum code.

What are the check weights?

$$\begin{array}{rl} X \text{ checks:} & \mathrm{wt}(A \text{ row}) + \mathrm{wt}(B \text{ column}) \\ Z \text{ checks:} & \mathrm{wt}(B \text{ row}) + \mathrm{wt}(A \text{ column}) \end{array}$$

Logical operators and distance

We can choose $k_1 k_2$ logical operators of X type, each acting on just the first register, such that each is $u \otimes v$ where $u \in \mathrm{Ker}\, A$ and $v \in \mathrm{Ker}\, B$. Here $v \in \mathrm{Ker}\, B$ is the condition to commute with H_Z, and $u \in \mathrm{Ker}\, A$ ensures that no two differ by an element of the X stabilizer. Likewise, there are $k_1^T k_2^T$ X logical operators supported on the second register, where each has the form $u \otimes v$ with $u \in \mathrm{Ker}\, A^T$ and $v \in \mathrm{Ker}\, B^T$. These are all the logical ops:

$$k = k_1 k_2 + k_1^T k_2^T .$$

But what about distance?

$$d = \min\big(d_1,\, d_2,\, d_1^T,\, d_2^T\big).$$

Codewords of this weight exist, of the form

$$\overline{U} = \begin{pmatrix} & u_1 & \\ 0 & \vdots & 0 \\ & u_{n_1} & \end{pmatrix}, \qquad \overline{V} = \begin{pmatrix} & 0 & \\ v_1 & \cdots & v_{n_2} \\ & 0 & \end{pmatrix},$$

with support on a single strip: a codeword of the classical code (or its transpose).

Double-sided expansion

We're interested in LDPC codes, which typically have highly redundant checks, so if we want distance $d = \Omega(\sqrt{n}\,)$ we'll need our codes and their transposes to have distance $\Omega(n_1, n_2)$. Hence A, B, A^T, B^T all are parity check matrices of good codes.

Idea: the Tanner graph is a double-sided expander. This means sets of variables (if not too large) trigger many checks for both H and H^T.

The Tanner graph is

$$G = \big(\underbrace{V}_{\text{variables}} \cup \underbrace{C}_{\text{checks}},\ E \big).$$

It is an (s, α) expander if any set S of variables with $|S| \le s$ has at least $\alpha|S|$ check neighbors.

Suppose variable nodes have $\le w_{\max}$ neighbors (degree), and suppose $\alpha > w_{\max}/2$. Then errors on bits in S with $|S| \le s$ have to trigger at least one check (hence not a codeword). Why? If all checks are satisfied, then each check has an even number of variable error neighbors. If all checks with error neighbors have at least 2 neighbors (true for a codeword), then the total number of neighbors is $\le \frac{w_{\max}}{2}|S|$. Therefore, since expansion $\implies > \frac{w_{\max}}{2}|S|$ neighbors, at least one check has a single neighbor.

If our (s, α) expander also has $s = \Omega(n)$, then codeword weight $> s \implies$ i.e., code distance $= \Omega(n)$. (Not possible in finite Euclidean dimensions.)

We also want high rate — more variables than checks. With $n =$ the number of variables and $m =$ the number of checks, the rate is

$$R \ \ge\ \frac{n-m}{n} = 1 - \frac{m}{n} \qquad \text{(at least } n-m \text{ unconstrained)}.$$

Suppose $D_v =$ ave. vertex degree and $D_c =$ ave. check degree. Then, because we can count edges two ways:

$$D_v\, n = D_c\, m \implies \frac{m}{n} = \frac{D_v}{D_c} \implies R \ \ge\ 1 - \frac{D_v}{D_c}.$$

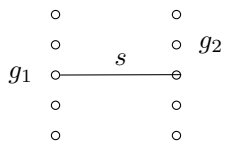
Tension: to get high expansion we want large enough D_v, which reduces rate.

Cayley graphs and coset graphs

Our bipartite graph is unbalanced, with a different number of checks and variables to get high rate, and we want expansion in both directions: sets of variables have many check neighbors and sets of checks have many variable neighbors (so A and A^T have large distance). Fortunately only A, and not A^T, needs to have high rate.

There are randomized constructions of unbalanced bipartite graphs that yield double-sided expansion with high probability, but for practical error correction we want explicit constructions with nice structure (for decoding and logical operations).

One way to get a balanced double-sided expander is with Cayley graphs. Nodes are elements of a nonabelian group, and we pick a set of generators. An edge connects g_1 on the left with g_2 on the right if $g_2 = s\, g_1$, and s is in the generating set.



The degree is the size of the generator set.

How to get double-sided expansion in an unbalanced bipartite graph? Now choose two subgroups K and H, and nodes are cosets. So e.g. the number of variables is $|G|/|H|$ and the number of checks is $|G|/|K|$, where $|K| > |H|$. Cosets are connected by an edge: connect gK to the H cosets such that gKs contains a representative of that coset, where s is a generator.

$$\text{Rate} \;\geq\; \frac{|G|/|H| - |G|/|K|}{|G|/|H|} = 1 - \frac{|H|}{|K|}.$$

Need to be careful how we choose G, H, K, and the generating set. A coset graph.

Toward lifted product codes

Another way to build on the HGP code construction to build codes of practical interest: lifted product codes. Replace entries in A and B by polynomials — i.e. circulant matrices. These could be multivariate polynomials, but we can construct useful codes where entries are univariate monomials:

$$x^p \quad \text{where } p \in \{0, 1, \ldots, L-1\} \;\text{ and }\; x^L = 1.$$

These are $L \times L$ matrices with a single 1 in each row and each column.

Chain Complexes and CSS Codes

Lecture 8 — 22 Apr 2026

Reference: §7.18 of the lecture notes, Chapter 7.

CSS codes are closely related to chain complexes. This relationship can be helpful for discovering codes, understanding their properties, thinking about decoding, and doing calculations. The chain complex viewpoint also illuminates the structure of hypergraph product codes.

We'll understand the connection by revisiting the toric code, and then generalizing.

Chain complexes and the toric code

A chain complex is a sequence of vector spaces and "boundary operators" mapping each space to the next one in the sequence,

$$\cdots \longrightarrow V_2 \xrightarrow{\partial_2} V_1 \xrightarrow{\partial_1} V_0 \longrightarrow \cdots$$

where e.g. $\partial_1 \circ \partial_2 = 0$ (the boundary of a boundary is zero). In the case of interest to us, the vector spaces are over $\mathbb{F}_2$. V_2 is associated with the elementary plaquettes on a lattice (a.k.a. faces, a.k.a. 2-cells): each 2-cell is assigned either 0 or 1. V_1 is likewise associated with edges (1-cells) and V_0 with vertices (0-cells).

We'll identify elements of V_2 (2-chains) with elements of the Z-type stabilizer of the toric code. Each check operator (S_Z generator) is Z_P, a product of $Z^{\otimes 4}$ on plaquette P. An element of S_Z is

$$\tilde{Z}(\omega) = \prod_P Z_P^{\omega_P},$$

where the 2-chain ω assigns 0 or 1 to each plaquette. Then multiplication of stabilizers becomes addition of vectors over $\mathbb{F}_2$:

$$\tilde{Z}(\omega)\,\tilde{Z}(\omega') = \tilde{Z}(\omega + \omega').$$

An arbitrary Z-type operator corresponds to a 1-chain, which assigns 0 or 1 to each lattice link:

$$Z(\varepsilon) = \bigotimes_e Z_e^{\varepsilon_e}$$

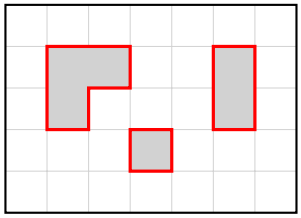
tensor product over all lattice links — i.e. all qubits in the code block,

$$Z(\varepsilon)\,Z(\varepsilon') = Z(\varepsilon + \varepsilon').$$

The boundary operator $\partial_2 : V_2 \to V_1$ acts as (sum mod 2)

$$(\partial_2\,\omega)_\ell = \sum_{P:\ \ell \in \partial P} \omega_P \qquad \Longrightarrow \qquad \tilde{Z}(\omega) = Z(\partial_2\,\omega),$$

the sum of boundaries of droplets of filled plaquettes — links inside a droplet cancel:

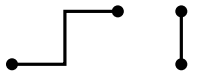


links inside
droplet cancel

Another boundary operator, $\partial_1 : V_1$ (edges) $\to V_0$ (vertices), assigns to site s

$$(\partial_1 \varepsilon)_s = \sum_{\ell:\ s\in\partial\ell} \varepsilon_\ell,$$

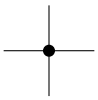
the sum of endpoints of connected paths:



sum of endpoints
of connected paths

Boundaries are defined so that $\partial_1 \circ \partial_2 = 0$ (boundaries of droplets are closed loops).

When does a Z-type operator commute with S_X (so it is contained in $S_X^\perp$)? At each site, $Z(\varepsilon)$ must contain an even number of Z's meeting at that site:



Hence $\partial_1 \varepsilon = 0$: "$\varepsilon$ is a 1-cycle." And $\partial_1 \circ \partial_2 = 0$ means elements of the Z-type stabilizer commute with the X-type stabilizer (as required for a CSS code).

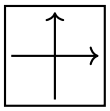
Z-type logical operators are $Z(\varepsilon)$ where ε is a 1-cycle. Trivial Z-type logical operators (in the stabilizer) are those in the image of ∂_2 (in the Z-type stabilizer):

$$\underset{Z\text{-stab: } \tilde{Z}(\omega)}{V_2} \xrightarrow{\partial_2} \underset{Z(\varepsilon)}{V_1} \xrightarrow{\partial_1} \underset{X\text{-type syndrome}}{V_0}$$

$$\mathcal{L}_Z = \frac{\mathrm{Ker}(\partial_1)}{\mathrm{Im}(\partial_2)} \equiv H_1(C), \qquad C = \text{chain complex over } \mathbb{F}_2.$$

For the torus,

$$H_1(C) = \underbrace{\mathbb{Z}_2}_{\substack{\text{horizontal}\\ \text{cycle}}} \oplus \underbrace{\mathbb{Z}_2}_{\substack{\text{vertical}\\ \text{cycle}}}$$



The dual complex and X-type logical operators

What about X-type logical operators? We could use the same construction on the dual lattice. Or — consider the *dual complex*, in which coboundary operators act in the opposite direction. These are transposes of the boundary operators:

$$V_2 \underset{\delta_1}{\overset{\partial_2}{\rightleftarrows}} V_1 \underset{\delta_0}{\overset{\partial_1}{\rightleftarrows}} V_0 \qquad \delta_1 = \partial_2^T, \quad \delta_0 = \partial_1^T.$$

δ_0 maps a site to all links meeting at the site. δ_1 maps a link to all plaquettes meeting at the link.

$$\delta_1 \circ \delta_0 = \partial_2^T \circ \partial_1^T = (\partial_1 \circ \partial_2)^T = 0,$$

$$\underset{Z\text{ stab}}{V_2} \xleftarrow{\delta_1} \underset{\text{qubits}}{V_1} \xleftarrow{\delta_0} \underset{X\text{ stab}}{V_0}$$

X-type logicals are in the kernel of δ_1 (they commute with Z-type stabilizers); trivial if in the image of δ_0 (contained in the stabilizer):

$$\mathcal{L}_X = \frac{\mathrm{Ker}(\delta_1)}{\mathrm{Im}(\delta_0)} = H^1(C^*) = \mathbb{Z}_2 \oplus \mathbb{Z}_2.$$

General CSS code

The Z-type check matrix H_Z is $m_Z \times n$; the X-type check matrix H_X is $m_X \times n$.

A general Z-type operator is $Z(\varepsilon)$, with ε a length-n 1-chain. A Z-type stabilizer element is a product of generators, $\tilde{Z}(\omega)$: ω is a length-m_Z 2-chain, indicating which generators (rows of H_Z) are multiplied together. The boundary $\partial_2 \omega$ of ω indicates which qubits support Z when the generators are multiplied. We sum the rows of H_Z that are tagged by ω (columns of H_Z^T):

$$\partial_2 \omega = H_Z^T \omega \qquad (\text{length } n).$$

For the 1-chain ε, we define its boundary as its length-m_X X-type syndrome:

$$\partial_1 \varepsilon = H_X \varepsilon.$$

Then $\partial_1 \circ \partial_2 = H_X H_Z^T = 0$ for a CSS code: Z stabilizers have trivial X-type syndrome.

$$\mathcal{L}_Z = \frac{\mathrm{Ker}(\partial_1)}{\mathrm{Im}(\partial_2)} = \frac{\mathrm{Ker}(H_X)}{\mathrm{Im}(H_Z^T)}.$$

For the dual complex,

$$\delta_1 = \partial_2^T = H_Z, \qquad \delta_0 = \partial_1^T = H_X^T, \qquad \delta_1 \circ \delta_0 = H_Z H_X^T = 0,$$

$$\mathcal{L}_X = \frac{\mathrm{Ker}(\delta_1)}{\mathrm{Im}(\delta_0)} = \frac{\mathrm{Ker}(H_Z)}{\mathrm{Im}(H_X^T)}.$$

Check: dimensions match for $\mathcal{L}_Z$ and $\mathcal{L}_X$:

$$\dim \mathrm{Ker}\, H_X - \dim \mathrm{Im}\, H_Z^T = \dim \mathrm{Ker}\, H_Z - \dim \mathrm{Im}\, H_X^T.$$

Recall: $\dim \mathrm{Ker}\, H + \dim \mathrm{Im}\, H = n$ if H is $r \times n$. This is the rank–nullity theorem: every dimension is either annihilated (kernel) or mapped to the output (image). Also: $\dim \mathrm{Im}\, H = \dim \mathrm{Im}\, H^T$ (the number of linearly independent rows = the number of linearly independent columns; e.g. show by Gaussian elimination). Hence

$$\dim \mathrm{Ker}\, H_X + \dim \mathrm{Im}\, H_X^T = n = \dim \mathrm{Ker}\, H_Z + \dim \mathrm{Im}\, H_Z^T$$

— it checks.

Homology/cohomology also provide a pairing of elements of $\mathcal{L}_Z$ and $\mathcal{L}_X$ (Poincaré duality), the intersection product, which defines a logical basis. Cycles and cocycles "pierce" one another. We saw how this works for the toric code, and it holds more generally.

Classical codes in homological language

We can understand the hypergraph product as a tensor product of chain complexes. First: a classical code and its transpose in the language of homology:

$$0 \xrightarrow{\partial_2} V_1 \xrightarrow{\partial_1} V_0 \xrightarrow{\partial_0} 0.$$

∂_2 has trivial kernel; the kernel of ∂_0 is all of V_0. $\partial_1 = H$ is the parity check matrix.

$$H_1(C) = \mathcal{L} = \frac{\mathrm{Ker}\, \partial_1}{\mathrm{Im}\, \partial_2} = \mathrm{Ker}\, H.$$

That is the code space. Also of interest:

$$H_0(C) = \mathcal{L}^T = \frac{\operatorname{Ker}\partial_0}{\operatorname{Im}\partial_1} = \frac{V_0}{\operatorname{Im} H}.$$

These are syndromes that don't occur, because H is redundant (more rows than needed). The vectors correspond to linear combinations of rows of H that sum to zero:

$$v^T H = 0 \iff H^T v = 0;$$

these are codewords of the transpose code. $H_1(C)$ is the code, $H_0(C)$ is the transpose code.

The hypergraph product as a tensor product of chain complexes

In HGP we put two chain complexes together to get a product complex that defines a CSS code:

$$\begin{aligned} C_A: &\quad 0 \xrightarrow{(\partial_2)_A} A_1 \xrightarrow{(\partial_1)_A} A_0 \xrightarrow{(\partial_0)_A} 0 \qquad\qquad (\partial_1)_A = A \text{ parity check,} \\ C_B: &\quad 0 \xrightarrow{(\partial_2)_B} B_1 \xrightarrow{(\partial_1)_B} B_0 \xrightarrow{(\partial_0)_B} 0. \end{aligned}$$

In the product complex,

$$C_A \otimes C_B: \quad A_1 \otimes B_1 \xrightarrow{\partial_2} \underbrace{A_1 \otimes B_0 \,\oplus\, A_0 \otimes B_1}_{\text{qubits in two registers}} \xrightarrow{\partial_1} A_0 \otimes B_0.$$

The boundary acts by the product rule:

$$\begin{aligned} \partial_2(A_1 \otimes B_1) &= (\partial_1)_A A_1 \otimes B_1 + A_1 \otimes (\partial_1)_B B_1, \\ \partial_1(A_1 \otimes B_0 \oplus A_0 \otimes B_1) &= \partial_1 A_1 \otimes B_0 + A_1 \otimes (\partial_0)_B B_0 + \partial_0 A_0 \otimes B_1 + A_0 \otimes \partial_1 B_1. \end{aligned}$$

Check $\partial_1 \circ \partial_2 = 0$:

$$\begin{aligned} \partial_1\big(\partial_2(A_1 \otimes B_1)\big) &= \partial_0\partial_1 A_1 \otimes B_1 + \partial_1 A_1 \otimes \partial_1 B_1 + \partial_1 A_1 \otimes \partial_1 B_1 + A_1 \otimes \partial_0\partial_1 B_1 \\ \partial_0\partial_1 = 0 \implies &= \partial_1 A_1 \otimes \partial_1 B_1 + \partial_1 A_1 \otimes \partial_1 B_1 = 0 \quad (\text{over } \mathbb{F}_2). \end{aligned}$$

This is the homological way to understand our observation that HGP arranges for collisions between H_X and H_Z to occur in matched pairs.

With this product complex, we construct the X-type (or Z-type) logical operators of a CSS code. This allows us to draw on the rich mathematical knowledge about homological algebra. Example: topologists know the Künneth formula for the homology of a product complex, in particular:

$$H_1(A \otimes B) = \underbrace{H_1(A)}_{A \text{ code}} \otimes \underbrace{H_0(B)}_{B^T \text{ code}} \,\oplus\, \underbrace{H_0(A)}_{A^T \text{ code}} \otimes \underbrace{H_1(B)}_{B \text{ code}}.$$

In particular,

$$k = k_A\, k_{B^T} + k_{A^T}\, k_B.$$

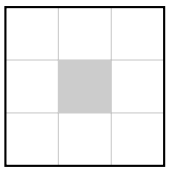
[To match this to what I claimed earlier, need to flip $B \to B^T$. Sorry about that!]

Higher-dimensional complexes: redundant syndromes

For a CSS code, what we need is $\partial_i \circ \partial_{i+1} = 0$, so we can draw on higher-dimensional chain complexes as well, and extending the complex further can be useful. Example: the *2D classical repetition code*.

Bits are on plaquettes, and checks on edges are triggered if two plaquettes disagree:

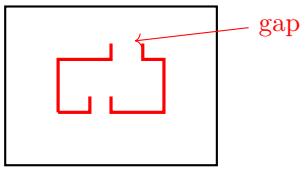
$$0 \xrightarrow{\partial_3} V_2 \xrightarrow{\partial_2} V_1 \xrightarrow{\partial_1} V_0 \xrightarrow{\partial_0} 0.$$



check on shared edge
is triggered

Now V_2 is a vector space assigning elements of $\mathbb{F}_2$ to each plaquette (1 indicates the location of an error). $\partial_2 = H$ is the parity check, assigning 1 to each edge where the bits disagree in $\partial_2 \omega$. $\partial_1 \varepsilon$ assigns 1 to each vertex where the number of edges in the support meeting at the vertex is odd.

$\partial_1 \circ \partial_2 = 0 \implies$ the syndrome of this classical code is redundant — some syndromes cannot occur. That there is a constraint on the syndrome can be useful for syndrome decoding if there are syndrome measurement errors. There might be gaps in the syndrome, and we can guess how to fill them in:



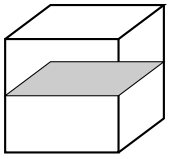
The 3D toric code and single-shot error correction

This idea can also apply to a CSS code. Example: the 3D toric code. Qubits live on plaquettes. Z stabilizers live on cubes (weight-6). X stabilizers live on edges:

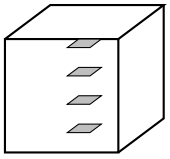
$$Z(c) = \prod_{P \in c} Z_P \qquad X(\ell) = \prod_{P \ni \ell} X_P$$

$$0 \xrightarrow{\partial_4} \text{cubes} \xrightarrow{\partial_3} \text{plaquettes} \xrightarrow{\partial_2} \text{links} \xrightarrow{\partial_1} \text{sites} \xrightarrow{\partial_0} 0.$$

Z logicals are closed-surface operators modulo trivial closed surfaces; X logicals are loop operators modulo trivial closed loops on the dual lattice:



logical Z



logical X

a stack of plaquettes
winding around the 3-torus

There is a pairing of X-type and Z-type logicals; $k = 3$ logical qubits. The X-type syndrome is a closed string at the boundary of a Z-type surface.

In the chain complex,

$$\mathcal{L}_Z = \frac{\overbrace{\operatorname{Ker} \partial_2}^{\text{commutes with } X \text{ stab}}}{\underbrace{\operatorname{Im} \partial_3}_{\text{not in } Z \text{ stab}}} = H_2(C).$$

In the dual complex, $\delta_1 = \partial_2^T$ fans out an edge to its plaquettes, and $\delta_2 = \partial_3^T$ maps a plaquette to its neighboring cubes:

$$\mathcal{L}_X = \frac{\overbrace{\operatorname{Ker} \delta_2}^{\text{commutes with } Z \text{ stab}}}{\underbrace{\operatorname{Im} \delta_1}_{\text{not in } X \text{ stab}}} = H^2(C),$$

$$\text{cubes} \xleftarrow{\delta_2} \text{plaquettes} \xleftarrow{\delta_1} \text{links} \xleftarrow{\delta_0} \text{sites}.$$

The X-type syndromes of Z errors are highly redundant: $\partial_1(\text{syndrome 1-chain}) = 0 \implies$ syndromes form closed loops. This enables "single-shot" error correction for the Z errors. If there are measurement errors we can "fill in the gaps" and have adequate confidence in the syndrome.

Unfortunately this is not true for the X errors. We need to *repeat* syndrome measurement multiple times to correct X errors if syndromes are noisy.

Lifted Product Codes

Lecture 9 — 27 Apr 2026

Reference: arXiv: 2603.28627.

We have established some tools for quantum code construction: polynomial rings, chain complexes, BB codes, HGP codes, ... The next step is *lifted product codes*, which combine the HGP construction with polynomial rings.

The idea is to follow the HGP/chain complex strategy to satisfy the CSS conditions (commuting X and Z stabilizer generators), but where the matrix entries are elements of a *ring* rather than a finite field like $\mathbb{F}_2$.

We'll specifically consider the ring of univariate polynomials over $\mathbb{F}_2$ ($R = \mathbb{F}_2[x]/(x^L - 1)$). But this can be generalized to multivariate polynomials, group algebras of abelian groups (or even nonabelian groups, if we are careful to preserve the CSS conditions).

Recall the HGP construction:

$$H_X = \big(\underbrace{A \otimes I_{n_B}}_{\substack{\text{Register 1:} \\ n_A n_B \text{ qubits}}} , \underbrace{I_{r_A} \otimes B^T}_{\substack{\text{Register 2:} \\ r_A r_B \text{ qubits}}} \big), \qquad H_Z = \big(I_{n_A} \otimes B,\ A^T \otimes I_{r_B} \big), \qquad \begin{matrix} A = r_A \times n_A \\ B = r_B \times n_B \end{matrix}$$

Number of X checks (rows of H_X): $r_A n_B$. Number of Z checks (rows of H_Z): $n_A r_B$.

$$\implies \quad n = n_A n_B + r_A r_B, \qquad k \geq n - r_A n_B - n_A r_B = (n_A - r_A)(n_B - r_B).$$

The lift

Now, instead of the entries in the A and B matrices being $0, 1$, we choose them to be elements of the ring. In principle, these could be polynomials of any order up to L, but monomials are preferred, because we don't want the check weight to grow. Each entry is x^p, where $p \in \{0, \ldots, L-1\}$, and we interpret x as a circulant matrix with a single 1 in each row — hence the same for x^p.

This construction is called a "lift" of the "seed matrices" A and B. In the seed code, entries are elements of R; in the lifted code, these elements of R are replaced by $L \times L$ matrices. These provide a compact and convenient way to describe very large check matrices. We may say that the lift turns the 2D structure of the HGP construction into a 3D structure, with a "fiber" (a cyclic 1D graph with L vertices) sitting above each vertex of a 2D "base." In this base, there are $n_A \times n_B$ sites in register 1 and $r_A \times r_B$ sites in register 2.

The lift increases the number of rows and columns in H_X and H_Z each by a factor of L, but without increasing the check weight. For example, one row of A has n_A entries and each column of B has r_B entries. Each entry expands under the lift to an $L \times L$ matrix with weight 1 in each row. So the check weights are $n_A + r_B$ for H_X and $n_B + r_A$ for H_Z.

The lifted code has

$$n = (n_A n_B + r_A r_B)\, L \quad \text{qubits},$$

and the number of checks is

$$(r_A n_B + n_A r_B)\, L.$$

Hence we have

$$k \geq (n_A - r_A)(n_B - r_B)\, L$$

(a lower bound, because checks might not be independent).

Logical operators and orbits

A Z-operator that commutes with the X checks can be represented as a column vector (length n) over R, where each entry is a polynomial. The nonzero coefficients in that polynomial indicate, at a particular site of the base graph, on which of the qubits in the fiber the Z's act.

Therefore, each Z operator that commutes with the X stabilizer corresponds to a solution to a set of polynomial equations of the form

$$v^T = (v_1, v_2, \ldots, v_n), \qquad x^{p_{i_1}}\, v_{i_1} + \cdots + x^{p_{i_w}}\, v_{i_w} = 0$$

— each term associated with a nonzero entry of a row of H_X, where w is the weight of the row. The weight of the solution is the sum from 1 to n of the number of nonzero terms in the polynomial v_i.

Note that if v satisfies all the checks, then so does xv, x^2v, etc. Each solution lies on an orbit of L solutions. If

$$H_X v = 0 \quad \text{then} \quad xH_X v = 0 = H_X(xv) \qquad \text{(distributive property in the ring)}.$$

Also, multiplication by x preserves the stabilizer, the image of H_Z^T. If v is in this image, then

$$v = H_Z^T u \quad \text{for some vector of polynomials } u,$$

and $xv = xH_Z^T u = H_Z^T(xu)$, so xv is in the image. Therefore, each nontrivial Z-type logical operator lies on an orbit:

$$\begin{aligned} &v \text{ commutes with } X\text{-stab} \implies \text{so does } xv, \\ &v \text{ not in stab} \implies \text{neither is } xv \text{ (or else } x^{-1}(xv) = v \text{ would be)}. \end{aligned}$$

To be avoided: in a given row of the seed with monomial entries

$$x^{p_1},\ x^{p_2},\ \ldots,\ x^{p_w}$$

we don't want L to have common factors with all differences $p_2 - p_1$, $p_3 - p_2$, $p_4 - p_3$, etc. That could result in shorter orbits, which might reduce the distance.

A distance upper bound

There is an upper bound on the distance d of this lifted product code, which arises from the $\mathbb{F}_2[x]$ variant of Cramer's rule in linear algebra. It says:

Consider a matrix which is $r \times (r+1)$. Because there are more columns than rows, its kernel is nontrivial. To construct an element of the kernel, each component is the determinant of an $r \times r$ submatrix (same as a *permanent* over $\mathbb{F}_2$, because $+1 \equiv -1$). The permanent is a sum of $r!$ terms.

In the case of our $r \times n$ seed matrix, suppose $n > r$, and look at a submatrix with $r+1$ columns. Each entry in the matrix is a monomial, so each term in the permanent is a product of monomials and hence a monomial. There are $r!$ terms, resulting in a polynomial with at most $r!$ terms. We construct a vector v with $r+1$ nonzero components, annihilated by the seed matrix, where each component has weight at most $r!$, resulting in a total weight of $(r+1)!$

In our lifted product construction with $n_A > r_A$ and $n_B > r_B$, this shows there are operators of weight no larger than $\min\big[(r_A+1)!,\ (r_B+1)!\big]$. But this does not by itself give an upper bound on the distance, unless we can show this operator does not lie in the stabilizer. (See below.)

Examples with $A = B$

An often-studied case is $A = B$:

$$H_X = \big(A \otimes I_n,\ I_r \otimes A^T\big), \qquad H_Z = \big(I_n \otimes A,\ A^T \otimes I_r\big), \qquad \text{where } A \text{ is } r \times n.$$

Some examples are discussed in Cain et al., arXiv: 2603.28627 (where it is noted that $d \le (r+1)!$). E.g., seed matrix 3×5 and $L = 33$:

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & x^{14} & x^{19} & x^{11} & x^{26} \\ 1 & x^{13} & x^2 & x^{15} & x^{21} \end{pmatrix}$$

Here

$$\begin{aligned} n &= (5^2 + 3^2)\, 33 = 34 \cdot 33 = 1122 \\ k &\ge (5-3)^2\, 33 = 132 \quad \text{(actually 148)} \\ d &\le (3+1)! = 24 \quad \text{(actually } d \le 20\text{)} \end{aligned}$$

Check weight is $3 + 5 = 8$.

Another example: A is 3×7 and $L = 91$:

$$\begin{aligned} n &= (7^2 + 3^2) \cdot 91 = 5278 \\ k &\ge (7-3)^2 \cdot 91 = 1456 \quad \text{(actually 1480)} \\ d &\le (3+1)! = 24 \quad \text{(not known to be lower)} \end{aligned}$$

Check weight $3 + 7 = 10$. Good rate:

$$k/n = \frac{1480}{5278} = .28\,.$$

Increasing L maintains the high rate, but does not increase the distance; we can't reach codes with $d = \Omega(n)$ with a fixed size of A. To get larger distance we can increase the size of the base, making it a double expander, but as we discussed, that alone is not enough to break the $d = O(\sqrt{n})$ barrier, and just adding a fiber does not help — a more sophisticated LP construction with *twisted* fibers forces logical operators to spread, increasing their weight to $\Omega(n)$.

Back to the distance bound $d \le (r+1)!$: We find a vector v of weight $(r+1)!$ in the kernel of A, $Av = 0$. So there is a vector $v \otimes e$ supported on the first register in the kernel of H_X. This can't be in the stabilizer, because $H_Z = I \otimes B$ on the first register.

Another way to say this: we constructed a representative of $H_1(A) \otimes H_0(B^T)$. $H_1(A)$ is just the code with A parity check, and $H_0(B^T)$ is the space of vectors modulo those generated by the rows of B^T (dual to the code defined by B). If B is $r \times n$ with $r < n$, there exists a weight-1 vector not in $\mathrm{Im}(B^T)$.

Syndrome measurement in atomic processors

The nice structure of our LP codes has benefits for syndrome extraction, syndrome decoding, and logical operations. Logical ops are a particular worry in high-rate codes, as there are many "overlapping" codewords distributed among the same physical qubits. We'll eventually

discuss a solution: "teleporting" from a memory code to a processor code via ancilla-assisted measurements.

Let's discuss syndrome measurement, first for the HGP code with

$$H_X = \big(A \otimes I_n,\ I_r \otimes A^T\big), \qquad H_Z = \big(I_n \otimes A,\ A^T \otimes I_r\big).$$

Arrange the 2 registers in $n \times n$ and $r \times r$ arrays. $A \otimes I_n$ acts nontrivially on qubits in the same row of Register 1; $I_r \otimes A^T$, on qubits in the same column of Register 2.

Suppose the data is fixed and the ancillas are mobile. For each row in A, an ancilla interacts, in each row of the array, with the qubit positions where that row has support. We flash the Rydberg laser as the ancilla passes a data qubit where A has support. Do this in all rows of the array in parallel.

We're not done: each ancilla qubit, after passing by n data qubits in a row of register 1, must then pass through r qubits in a column of register 2. For $r \approx n$, there is no bottleneck; we need more clever procedures for scheduling and routing.

Discussed by Xu et al., arXiv: 2308.08648. Two ideas help. ① *Color scheduling.* Color the edges of the Tanner graph so each check and variable connects to unlike colors; # of colors = degree of the bipartite graph. In a time step, perform interactions of one color. Each ancilla and data qubit interacts with one other qubit. ② *"Divide & conquer."* Minimize the number of moves with a recursive routing algorithm.

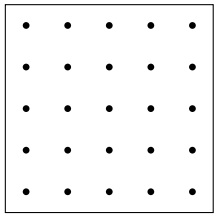
Syndrome Decoding

Lecture 10 — 29 Apr 2026

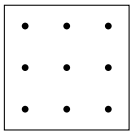
References: arXiv: 2308.08648, arXiv: quant-ph/0110143.

Dance of the tweezers

$$H_X = \big(A \otimes I_n,\ I_r \otimes A^T\big). \qquad \text{Suppose } A \text{ is } 3 \times 5.$$



Register 1: 5×5



Register 2: 3×3

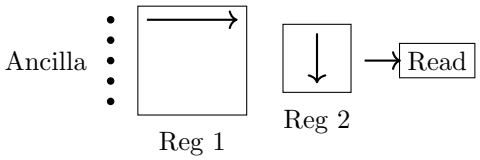
$$\big(A \otimes I_5\big)_{ik,\,j\ell} = A_{ij}\,\delta_{k\ell}, \qquad \begin{array}{l} i \in \{1,2,3\} \quad 3 \text{ rows of } A \\ j \in \{1,2,3,4,5\} \quad 5 \text{ columns of } A \end{array}$$

Three checks act on each row of the 5×5 array. Each check weight ≤ 5.

$$\big(I_3 \otimes A^T\big)_{ik,\,j\ell} = \delta_{ij}\,A^T_{k\ell}, \qquad \begin{array}{l} k \in \{1,2,\ldots,5\} \quad 5 \text{ columns of } A \\ \ell \in \{1,2,3\} \quad 3 \text{ rows of } A \end{array}$$

Five checks act on each column of the 3×3 array. Each check weight ≤ 3.

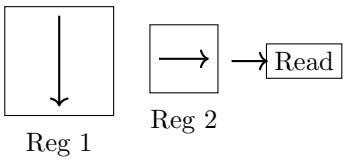
H_X has 15 checks, each supported on one row of the Register 1 array and one column of the Register 2 array.



In principle, a mobile ancilla qubit could interact with a row of Register 1 and a column of Register 2 before proceeding to the readout zone.

We also need to measure

$$H_Z = \big(I_5 \otimes A,\ A^T \otimes I_3\big).$$



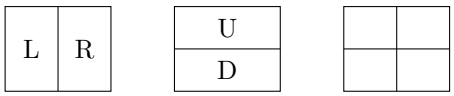
Each mobile ancilla interacts with a column of Reg. 1 and a row of Reg. 2.

There are further considerations in scheduling. We want ancillas to be next to target data qubits when the Rydberg laser is flashed. We want to extract syndrome data with high parallelism. We want to minimize movement times to the extent possible, since move time dominates the time for syndrome extraction.

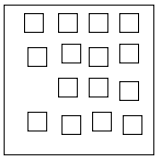
Some helpful ideas — Xu et al., arXiv: 2308.08648:

1. *Coloration scheduling.* Each data qubit and each ancilla qubit interacts with only one other qubit at a time. For both H_X and H_Z, we color the edges of the bipartite Tanner graph so that each qubit is connected to others via edges of different colors. The number of colors is the max degree of the bipartite graph. In each time step, gates are performed on pairs connected by edges of a specified color.

2. Reduce the number of moves via a recursive "divide and conquer" movement algorithm. E.g., given target locations, first move across the center line so each qubit is in its targeted L or R rectangle. Do the same for U and D. Then repeat at smaller and smaller scales. So $O(\log n)$ moves put all in the right position.



In LP codes, each vertex of the base graph becomes a fiber of L qubits, arranged e.g. in a square array, perhaps with vacancies.



The monomial x^p at each entry of A specifies which of the L qubits participates in each of the L checks arising from each row of H_X or H_Z. That needs to be included in the "tweezer dance." Optimizing the movement algorithm will be an important element of operating a neutral atom FTQC at scale.

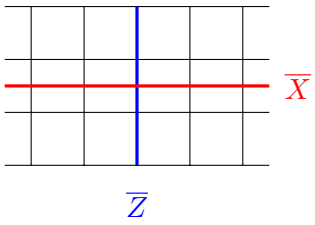
Syndrome decoding

An important part of FTQC is the classical algorithm for syndrome decoding, which must be robust against errors in syndrome bits. In particular, we wish to verify that there is an "accuracy threshold" such that when the error rate in 2Q gates is less than ε_0, the probability of a logical error decays exponentially with increasing block size and distance. This requires a circuit-level analysis of how errors propagate, including for gadgets that execute universal gates on logical qubits.

For concatenated codes, verifying the threshold is relatively simple, at least conceptually. We can analyze decoding performed "level by level." There is an effective error rate that decays double-exponentially with encoding level, while the code length expands exponentially.

For other codes like topological codes and high-rate codes, we need different methods. To get started, consider our favorite example: the 2D toric code. For now, consider only memory, not computation, and "phenomenological" noise rather than circuit-based noise.

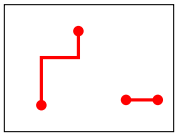
Consider the planar unrotated surface code, $L \times L$, with qubits on edges:



The distance is $d = L$. Min-weight logical ops are strings: $\overline{Z}$ running from top to bottom, $\overline{X}$ running from left to right.

Errors define a *1-chain*, E: the set of edges where errors occur. Suppose at first that syndrome information is perfect. Consider Z errors and X-type stabilizer generators. The syndrome is a *zero-chain*: a set of sites S where $X_{\text{site}} = -1$. Hence

$$S = \partial E \qquad \text{the syndrome 0-chain is the boundary of the error 1-chain.}$$



The decoding algorithm specifies a way to return to the code space: a *recovery chain* E' such that $\partial E' = S$. Then $E + E'$ is a cycle, with trivial syndrome. A noise model assigns a probability to each possible error 1-chain E, denoted $\text{Prob}(E)$.

Homology classes and optimal recovery

Once the syndrome is known, choose a canonical E_0 such that $\partial E_0 = S$. The general 1-chain with boundary S is

$$E_0 + C, \quad \text{where } C \text{ is a cycle, } \partial C = 0 \qquad \text{(cycle is defined relative to the boundary).}$$

All cycles in the same homology class are equivalent. If C and C' are both in class h, then $C + C'$ is in the trivial class — that is, it can be "tiled by plaquettes," and hence is in the stabilizer.

Given the syndrome and the noise model, we can assign a probability to each homology class:

$$\text{Prob}(h \,|\, S) = \frac{\sum_{C \in h} \text{Prob}(E_0 + C)}{\sum_{C} \text{Prob}(E_0 + C)}.$$

Denominator = probability of syndrome $S = \partial E_0$. Numerator = ditto, and $E_0 + E' \in h$, where $E' = E_0 + C$.

Optimal recovery: choose h to be the most likely homology class (and the recovery chain to be any representative of that class). Then the probability of a decoding failure is

$$\text{Failure Probability} = \sum_{E} \text{Prob}(E)\ \text{Prob}\big(E + E_0 \notin h_{\max} \,\big|\, E\big).$$

$\text{Prob}(E)$ is determined by the noise model, and the other factor is the probability that E is not in the most likely class given syndrome $S = \partial E$.

If the noise is i.i.d. with error probability ε, then

$$\text{Prob}(E) = \varepsilon^{|E|}\,(1-\varepsilon)^{n-|E|} = (1-\varepsilon)^n \left(\frac{\varepsilon}{1-\varepsilon}\right)^{|E|}.$$

Computing the most likely homology class can be expensive — it is more feasible to find the most likely error *1-chain*, which for the i.i.d. noise model is a 1-chain E with minimal error

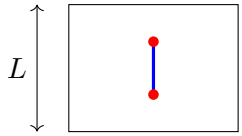
weight $|E|$ given the observed syndrome. This is done by the minimal-weight-perfect-matching algorithm (relative to the boundary). This works pretty well, but might not be optimal, because the most likely error might not belong to the most likely class. (Stat mech analogy: this is like minimizing the energy, ignoring entropy — we should minimize free energy instead. Dennis et al., arXiv: quant-ph/0110143.)

MWPM and the accuracy threshold

For the i.i.d. model, the probability of a decoding failure is the probability that the actual error E and the most likely error E' satisfy

$$E + E' = C \in \text{nontrivial homology class},$$

where the most likely error is determined by Min-Weight Perfect Matching (MWPM). For a Z error in the unrotated planar surface code, this is the probability that $C = E + E'$ contains an odd number of "strings" connecting the bottom and top boundaries.



There is at least one such path in $E + E'$, which has length (# of edges) $\ell \geq L$. Furthermore, E must contain at least half of the edges in this path. Otherwise, we could reduce the weight of E' by choosing E instead, which has the same boundary. Because MWPM found E', that can't be the case.

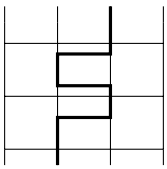
Denote this length-ℓ path by C_ℓ. The number of errors in E on this path is at least $\ell/2$, and the number of ways to choose which edges in C_ℓ are contained in E is at most 2^ℓ (each of ℓ edges either contained or not). Hence the probability that C_ℓ is contained in $E + E'$ is

$$\text{Prob}(C_\ell) \leq 2^\ell \, \varepsilon^{\ell/2} = (4\varepsilon)^{\ell/2}.$$

Let N_ℓ = # of connected paths of length ℓ connecting the top and bottom boundaries. Union bound $\Longrightarrow$

$$\text{Prob(failure)} \leq \sum_{\ell=L}^{n} N_\ell \, (4\varepsilon)^{\ell/2}.$$

Now we upper bound N_ℓ.



The path starts at the bottom on one of L edges. Each step is in one of 3 directions: forward, turn left, turn right. Whether or not the path reaches the upper boundary:

$$N_\ell \leq L \, 3^\ell \quad \Longrightarrow \quad \text{Prob(failure)} \leq L \sum_{\ell=L}^{n} (36 \, \varepsilon)^{\ell/2}.$$

If $\varepsilon < \frac{1}{36} \cong .028$, the first ($\ell = L$) term is the largest

$$\Longrightarrow \quad \text{Prob(failure)} \leq O(L^3) \left(\frac{\varepsilon}{\varepsilon_0}\right)^{L/2}, \qquad \varepsilon_0 = \frac{1}{36}.$$

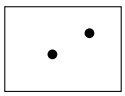
Hence there is an accuracy threshold $\varepsilon_{\text{th}} > \varepsilon_0$, and the error rate decays with code distance.

MC simulation $\Longrightarrow$

$$\varepsilon_0 \cong .103 \quad \text{for MWPM}$$
$$\varepsilon_0 \cong .109 \quad \text{for optimal recovery}$$

Syndrome errors

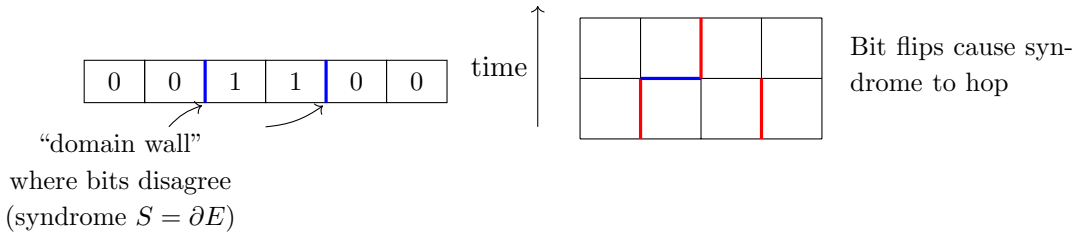
But — what if the syndrome has errors? Then it is dangerous to apply MWPM. There are apparent elements of S that are not near others. Connecting those "site defects" with min-weight 1-chains can cause E' to differ markedly from E.



To obtain trustworthy syndrome info, we should repeat the syndrome measurement multiple times. How, then, to recover?

Syndrome history in spacetime

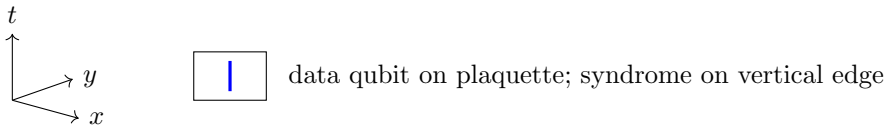
To visualize the syndrome history in spacetime, add a dimension representing time. Consider the easier case of the 1D repetition code. Bits are on plaquettes, checks are on vertical edges. Errors that occur between syndrome measurement rounds (where a bit flips) are indicated as horizontal edges.



Now, if the syndrome is perfect (no measurement errors), $S + E$ has no boundary.

If the syndrome has errors, let's consider E to contain syndrome errors as well as data bit-flip errors, so that E becomes a 1-chain containing both horizontal and vertical edges, and let S be the observed syndrome 1-chain (vertical edges only). The observed syndrome + syndrome errors is the same as the ideal syndrome. Therefore it is still true that $S + E$ has no boundary when E contains syndrome errors and S is the observed syndrome.

For the 2D toric code, the syndrome history is 3D. Again consider qubits on plaquettes, in xt and yt planes; the syndrome is on vertical edges where 4 such plaquettes meet.



Again, errors between rounds are indicated on horizontal edges, causing the ideal syndrome to hop. Again $\partial(S + E) = 0$, where S is the observed syndrome history and E includes syndrome measurement errors.

A phenomenological noise model assigns a probability to each error 1-chain (including data flips and syndrome flips). Although paths now travel in both space + time directions, they can be classified homologically according to whether they connect boundaries on opposite sides.

For simplicity, consider again an i.i.d. noise model. In principle, error probabilities on vertical (syndrome) edges and horizontal (data) edges could differ, but in practice they are comparable, so suppose ε for both.

We can apply MWPM to the spacetime syndrome S, and bound the probability that $E + S$ contains a path of length L or greater. The analysis is as before, except for two changes:

- The number of paths of length ℓ scales linearly with the elapsed time T, so we are estimating the probability of logical error per unit time (per syndrome measurement round).
- The path has 4 ways to turn, so the # of ways to add an edge is 5 instead of 3 $\Longrightarrow$

$$5^\ell \, (4\varepsilon)^{\ell/2} = (100\,\varepsilon)^{\ell/2} \quad \text{and} \quad \varepsilon_0 = .01.$$

  MC simulations show $\varepsilon_0 \cong .0293$.

- We'll want $T \gtrsim L$. Otherwise too many defects get matched to the future boundary, and we don't remove them.

Belief Propagation

Lecture 11 — 4 May 2026

Reminder: no lecture on May 6.

Belief propagation is a widely used heuristic for decoding high-rate classical LDPC codes. It is not so useful for the surface code, so a few years ago I would not have considered including it in lectures on QEC. But with the advent of high-rate qLDPC CSS codes, BP is frequently employed in QEC now, so it is worthwhile to discuss. We'll consider a classical linear code, but you can think of it as the X or Z syndrome decoding of a quantum CSS code.

Maximum likelihood decoding

H is the $(n-k)\times n$ parity check matrix. The error e is an n-component binary vector, indicating which bits have flipped. $He = s$, where the syndrome s is $(n-k)$ components. We observe s and want to infer e, but there are 2^k possibilities. Among these, we wish to find the most likely e, as determined by our noise model, which assigns probability $P(e)$ to error e.

According to Bayes:

$$P(e|s) = \frac{P(e)\,P(s|e)}{P(s)}.$$

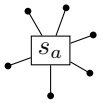
In Maximum Likelihood Decoding (MLD) we seek the Maximum Likelihood Estimate (MLE) of e given s:

$$e^* = \operatorname{argmax} P(e|s).$$

We need $P(s)$ in the denominator for normalization, but when s is known, all the dependence on e is in the numerator, so we are looking for the e that maximizes $P(e)\,P(s|e)$. Here $P(e)$ comes from the noise model, but s is a deterministic function of e — that is, for each e, one particular s occurs with probability 1.

Consider the code's Tanner graph. Check nodes are labeled $a, b, c, \ldots$, and variable nodes are labeled $i, j, k, \ldots$. We write $P(s|e)$ as a product of deterministic factors, one for each of the $n-k$ checks:

$$P(s|e) = \prod_{a\in C} \phi_a\big(e_{\partial a},\, s_a\big).$$



Here $e_{\partial a}$ denotes the components of e that are neighbors of a in the Tanner graph (its "boundary"), and $\phi_a = 1$ if the parity of these neighbors matches the observed syndrome bit s_a. Otherwise, it is zero. So $\prod_a \phi_a = 1$ if $He = s$, and is zero otherwise.

Let's suppose independent noise, so $P(e)$ factorizes, but is not necessarily identical on all bits:

$$P(e) = \prod_{i\in V} \psi_i(e_i).$$

Hence $\psi_i(e_i)$ is the prior distribution, which should be updated after we observe s.

The stat-mech analogy; marginals

We're looking for the error e that maximizes

$$P(e|s) = \frac{1}{Z(s)} \prod_{i\in V} \psi_i(e_i) \prod_{a\in C} \phi_a\big(e_{\partial a},\, s_a\big)$$

(where we write $P(s) = Z(s)$ to emphasize the stat mech analogy). We are trying to minimize the free energy of a spin-glass model, in the Gibbs distribution. The $\{e_i\}$ are the binary spins — they are warm, with $\psi_i(e_i)$ the Boltzmann factor for a spin in an external field. The $\{\phi_a\}$ are the interaction terms — they are cold, i.e., the interaction energy is $\gg$ temperature. The trouble is that for a general Tanner graph and a typical s, the free energy $-\log P(e|s)$ is a nonconvex function which is NP-hard to minimize.

Given s, the distribution $P(e|s)$, viewed globally, may have complex correlations among the bits that are difficult to characterize. In BP we set the more modest goal of estimating marginal distributions for each variable:

$$P_i(e_i|s) = \sum_{e \setminus e_i} P(e|s).$$

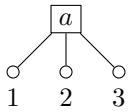
Then we'll guess no error occurred if

$$\frac{P_i(e_i = 0 \,|\, s)}{P_i(e_i = 1 \,|\, s)} > 1 \qquad \text{or log-likelihood ratio} \quad \text{LLR} = \ln\left(\frac{P_i(0)}{P_i(1)}\right) > 0.$$

Message passing

In BP, each variable has a belief about its own marginal; each check has beliefs about the marginals of all its neighboring variables, and in addition the knowledge that the check is satisfied. It does not claim to know anything else. BP is an iterative message-passing algorithm on the Tanner graph, in which in one round variables send messages about their beliefs to their neighboring checks, and in the following round checks send messages about their beliefs to their neighboring variables. Beliefs are updated in each round using the information just obtained, and the procedure is iterated many times until beliefs converge and no new updates are occurring. The goal is to reach an equilibrium in which the beliefs of checks and variables are consistent. We initialize the procedure by having variables believe their marginals are given by the prior $\psi_i(e_i)$.

What should a check tell a variable? Suppose, for example, that check a has degree 3, i.e. computes the parity of 3 variables (it's a row of H with weight 3).



In a previous round, a received estimates of $P_i(e_i)$ for each variable. It passes what it knows to, say, variable 1, assuming (perhaps incorrectly) that variables 2 and 3 are independent. Check a estimates, assuming $s_a = 0$:

$$P_1^{\text{new}}(0) = P_2^{\text{old}}(0)\, P_3^{\text{old}}(0) + P_2^{\text{old}}(1)\, P_3^{\text{old}}(1)$$
$$P_1^{\text{new}}(1) = P_2^{\text{old}}(0)\, P_3^{\text{old}}(1) + P_2^{\text{old}}(1)\, P_3^{\text{old}}(0)$$

If on the other hand $s_a = 1$, then the estimate is flipped: $P_1^{\text{new}}(0) \leftrightarrow P_1^{\text{new}}(1)$.

It is convenient to reformulate this "sum-product" formula for updated probability in terms of the "bias" $P(0) - P(1)$:

$$\underbrace{P_1^{\text{new}}(0) - P_1^{\text{new}}(1)}_{\text{new bias}} = (-1)^{s_a} \underbrace{\big(P_2^{\text{old}}(0) - P_2^{\text{old}}(1)\big)}_{\text{old bias}} \underbrace{\big(P_3^{\text{old}}(0) - P_3^{\text{old}}(1)\big)}_{\text{old bias}}$$

Expanding the product of binomials recovers the previous formula, and $(-1)^{s_a}$ flips the bias if $s_a = 1$.

A further convenience is using "log-likelihood ratios" (LLRs), which frees us from explicitly renormalizing probabilities, and transforms multiplication into addition:

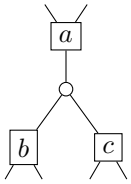
$$L = \ln\left(\frac{P(0)}{P(1)}\right) \implies P(0) - P(1) = \frac{P(0)/P(1) - 1}{P(0)/P(1) + 1} = \frac{e^L - 1}{e^L + 1} = \frac{e^{L/2} - e^{-L/2}}{e^{L/2} + e^{-L/2}} = \tanh\left(\frac{L}{2}\right).$$

So the bias update rule becomes

$$\tanh\left(\frac{L_1^{\text{new}}}{2}\right) = (-1)^{s_a} \tanh\left(\frac{L_2^{\text{old}}}{2}\right) \tanh\left(\frac{L_3^{\text{old}}}{2}\right).$$

This L_1^{new} is reported by the check to variable 1. Similarly, the check computes L_2^{new} and L_3^{new} and reports to the corresponding variable nodes.

How does the variable use the information it receives from the checks? It assumes (perhaps incorrectly) that the neighboring checks are independent witnesses, and uses the new information to update its prior.



It further assumes that each witness is independent of the prior distribution $\psi_i(e_i)$ for variable i. Under this *local independence assumption*, the variable multiplies probabilities (up to normalization) to update the prior:

$$P_i^{\text{new}}(e_i) \;\propto\; P_i^{\text{prior}}(e_i)\, P_i^{\text{old},a}(e_i)\, P_i^{\text{old},b}(e_i)\, P_i^{\text{old},c}(e_i).$$

In terms of the log-likelihood ratio:

$$\begin{aligned} L_i^{\text{new}} = \ln \frac{P_i^{\text{new}}(0)}{P_i^{\text{new}}(1)} &= \ln\left(\frac{P_i^{\text{prior}}(0)\, P_i^{\text{old},a}(0)\, P_i^{\text{old},b}(0)\, P_i^{\text{old},c}(0)}{\text{same } (0 \leftrightarrow 1)}\right) \\ &= L_i^{\text{prior}} + L_i^{\text{old},a} + L_i^{\text{old},b} + L_i^{\text{old},c}. \end{aligned}$$

This is variable i's updated belief about its own LLR.

In the message-passing protocol, we denote by $L_{a\to i}$ the message check a sends to variable i regarding check a's belief about variable i, updated using the sum-product rule. As above, this message satisfies

$$\tanh\left(\frac{L_{a\to i}^{\text{new}}}{2}\right) = (-1)^{s_a} \prod_{j \in \partial a \setminus i} \tanh\left(\frac{L_{j\to a}^{\text{old}}}{2}\right).$$

Importantly, using the local independence assumption, this update is based on information check a received in the previous round from all of its neighbors except for variable i.

Variable i's updated belief about itself is

$$L_i^{\text{belief,new}} = L_i^{\text{prior}} + \sum_{b \in \partial i} L_{b\to i}^{\text{old}}.$$

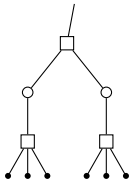
Importantly, vertex i does *not* report this belief to its neighboring check a. This belief is based in part on information i received from a, and so would violate the local independence assumption. Instead i leaves this information out of its message to a, and instead sends

$$L_{i\to a}^{\text{new}} = L_i^{\text{prior}} + \sum_{b\in\partial i\setminus a} L_{b\to i}^{\text{old}}.$$

These rules define the BP message-passing protocol on the Tanner graph, which we iterate multiple times. If we are lucky, messages tilt toward $\tanh(L/2) \to 1$ (strong bias favoring 0) or $\tanh(L/2) \to -1$ (strong bias favoring 1), so we are confident in inferring e from these estimated marginal distributions.

Trees and loops

If the Tanner graph is a tree graph, then we can apply the BP rules starting with prior beliefs about the variables on the leaves. Information propagates only inward, away from the leaves. (Variables don't send info received from a check back to the same check.)



Also the local independence assumption is true, as information is aggregated from disjoint subtrees. Hence BP converges to the correct marginals in time scaling like the graph's diameter (the maximum distance between leaves).

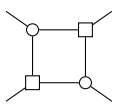
Also true, but less obvious: the marginals descended from the global joint distribution $P(e|s)$ yield a *fixed point* of BP (updates don't change beliefs), even though local independence does not hold. It's because the rule a check uses to update marginals is just a local version of computing marginals starting from the global distribution. This gives us hope that if we iterate enough times, marginals converge to the correct value.

The trouble is that, on graphs with loops (especially short loops), local independence can be badly violated, because information that flows away from a node can propagate back to the same node after traveling around a loop. As a result, there can be fixed points of BP other than the correct marginals, or BP may fail to converge at all.

Ordered statistics decoding (OSD)

Therefore, BP is typically augmented by another protocol, such as "ordered statistics decoding" (OSD). For example, if BP does converge, we can verify whether the fixed point e^* we found really satisfies $He^* = s$. If not, order the bits e_i^* according to how close the LLR L_i^* is to $\tanh(L_i^*/2) = \pm 1$. We accept the e_i^* in which we have the most confidence, and use Gaussian elimination to solve for the rest.

Other decoders outperform BP for, e.g., the surface code, because its Tanner graph contains short loops (length 4):



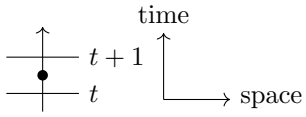
But for Tanner graphs of high-rate codes like HGP codes from double-expanders, loops are rather large (the graph has a large "girth"), so BP + OSD is pretty effective.

Syndrome errors and single shot

We assumed here that the syndrome s is accurate, but what if it's not because there are syndrome measurement errors? We might have to repeat syndrome measurements to improve reliability. But LDPC codes defined by Tanner graphs with sufficient expansion have the "single-shot" property, meaning that a syndrome measured just once can be decoded reliably.

The idea is that one can decode a syndrome using a greedy algorithm, a sequence of "small-set flips" that reduce the weight of the syndrome step-by-step. For an ideal syndrome, this eventually reduces the weight to zero, identifying a candidate e with a small probability of a logical error. Even if the syndrome has noise, because each decoding step is small and uses only local information on the Tanner graph, steps based on faulty information don't cause a large deviation between the true and apparent error, and with high probability we find an approximate codeword that is close to the correct codeword.

If we do need to measure the syndrome multiple times, we can apply BP to the syndrome history graph. As for our discussion of the surface code, bit errors occur on "horizontal" or "spacelike" edges of this graph, and measurement errors occur on "vertical" or "timelike" edges.



The check nodes identify where a change occurs in the syndrome between rounds t and $t+1$. The change can occur because a data bit flipped, or a syndrome error occurred. In BP, messages are passed in both timelike and spacelike directions, seeking locally consistent marginals for data + syndrome errors.

Quantum Computing by Measurement

Lecture 12 — 11 May 2026

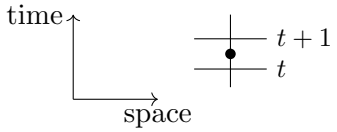
Last time: belief propagation

(See Lecture 11 notes.)

- Message passing on the Tanner graph.
- Ideally, want to know $e^* = \operatorname{argmax} P(e|s)$. Too hard.
- Instead, estimate the log-likelihood ratio of marginals:

$$L_i = \ln\left(\frac{P_i(0)}{P_i(1)}\right) \qquad \begin{array}{l}\text{Assume no error for } L_i > 0.\\ \text{Assume error for } L_i < 0.\end{array}$$

- Updates assume local independence of messages.
- Justified on a tree; violated on graphs with loops.
- Bad performance for the surface code. Better for expander graphs with "large girth" (not any short loops).
- If $He \neq s$ or BP fails to converge, augment with OSD = ordered statistic decoding: accept e_i with large $|L_i|$, solve for the rest by Gaussian elimination.
- Syndrome errors? Apply BP to the syndrome history. Check nodes are triggered where the syndrome changes. BP passes messages in both spacelike and timelike directions, seeking locally consistent marginals for both data and syndrome errors.



Circuit-based estimates of logical error rate

- Schedule to minimize error propagation during syndrome measurement.
- Consider memory errors (more about logical processing later).
- Surface code:

$$P_{\text{logical}} \approx 0.1\left(\frac{p}{p_{\text{th}}}\right)^{(d+1)/2} \quad (d \text{ odd}), \qquad n = d^2, \quad p_{\text{th}} \sim 0.01.$$

Example: factoring for RSA 2048 (Gidney 2025). Keep ∼1500 logicals alive for 12 hours (then do multiple shots for runtime ∼ few days):

$$P_{\text{logical}}^{-1} = (\text{logical error rate per round})^{-1} \sim 1500 \times 12 \times 3600 \times 10^6 \sim 10^{14},$$

assuming a round of syndrome measurement takes ∼1 microsecond. Suppose $p \sim 10^{-3} \Rightarrow$ want $(d+1)/2 \sim 13 \Rightarrow d = 25$, i.e. $\sim 2(25)^2 \sim 1250$ physical qubits per logical

$$\implies \quad (1500)(1250) \approx 1.9\,\text{M physical qubits.}$$

(Gidney reduces this to ∼1M via the "yoked surface code" for cold storage.)

Compare Cain et al. 2026: for a $[[5278, 1480, d \leq 24]]$ lifted product code ($k \sim 1500$), the block error rate (probability per cycle that any logical qubit fails) is 10^{-11} for $p = 10^{-3}$ physical error rate, with much smaller physical qubit count.

Quantum computing by measurement

For logical processing in both the surface code and high-rate codes, performing computation by measurement is a key concept, because we know how to measure logical operators (Paulis) fault-tolerantly. We typically use the Clifford + T universal gate set, and track how Pauli operators propagate through Clifford gates (Heisenberg picture — Gottesman–Knill). We'll assume we can do both destructive measurements (which destroy encoded blocks) and nondestructive measurements (revealing a Pauli eigenvalue, preserving the encoding).

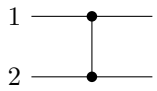
A Clifford unitary acting by conjugation takes Pauli P to Pauli P':

$$UPU^\dagger = P' \implies UP = P'U.$$

U maps a P eigenstate to a P' eigenstate with the same eigenvalue:

$$P|\psi\rangle = \lambda|\psi\rangle \implies P'\,(U|\psi\rangle) = UP|\psi\rangle = \lambda\,(U|\psi\rangle).$$

Example: CZ gate.



(1 ↔ 2 symmetry)

$$Z_1 \to Z_1, \qquad X_1 \to X_1 Z_2,$$
$$Z_2 \to Z_2, \qquad X_2 \to Z_1 X_2.$$

$|\pm\rangle|\psi\rangle \;\to\; |0\rangle|\psi\rangle \pm |1\rangle\, Z|\psi\rangle$. This is a $\pm$ eigenstate of $X_1 Z_2$.

Other examples:

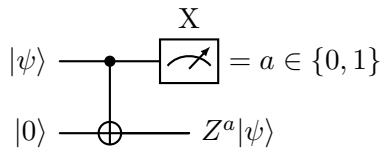
$$X:\ X \to X,\ Z \to -Z; \qquad H:\ X \to Z,\ Z \to X;$$

$$S = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix} \implies SXS^\dagger = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix} X \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} = \begin{pmatrix} 0 & i \\ -i & 0 \end{pmatrix} = -Y,$$
$$S:\ X \to -Y,\ Z \to Z.$$

Since $\{X_i, Z_i\}$ generate all Paulis, it is enough to know the action of U on these. [For CNOT, see the end of this lecture.]

One-bit teleportation

An important primitive: "one-bit teleportation."

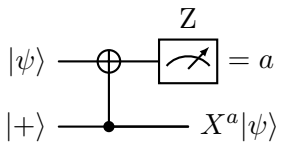


How it works. Stabilizer: $IZ \to ZZ$.

$$XI \to XX \to (-1)^a X \quad \longleftarrow \text{apply } Z^a \text{ to remove phase}$$
$$ZI \to ZI \equiv IZ \to Z$$

($ZI \equiv IZ$ because $ZZ \in$ Stab.)

The dual version:

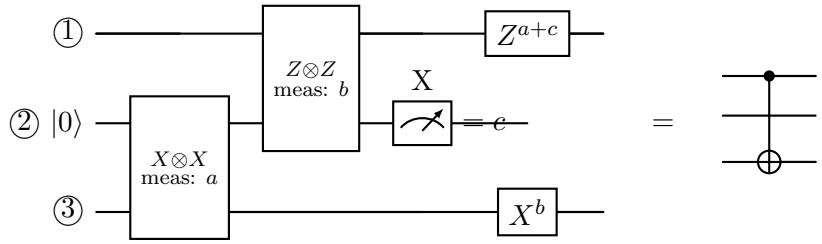


Stabilizer: $IX \to XX$.

$$XI \to XI \equiv IX \to X$$
$$ZI \to ZZ \to (-1)^a Z \quad \Longleftarrow \text{apply } X^a \text{ to remove phase}$$

One-bit teleportation is useful for moving logical info from one block to another. (We don't need to apply the Pauli correction; it suffices to keep track of it in software.)

Measurement gadget for CNOT



Note: the XX and ZZ measurements are nondestructive.

How does it work? The initial stabilizer is $M = IZI$. We are to measure IXX. We want to propagate

$$X_1 = XII, \quad X_3 = IIX, \quad Z_1 = ZII, \quad Z_3 = IIZ.$$

Z_3 does not commute with the IXX measurement. But $MZ_3 = IZZ$ does commute.

Next: measure ZZI; the new stabilizer is $M = (-1)^a IXX$. $X_1 = XII$ does not commute with ZZI, but $X_1 M = (-1)^a XXX$ does.

Last step: measure IXI; the new stabilizer is $M = (-1)^b ZZI$. Now $Z_3 \to IZZ$ does not commute, but $MZ_3 = (-1)^b ZIZ$ does.

Summary:

$$X_1 \to (-1)^a\, XXX \to (-1)^{a+c}\, XIX$$
$$Z_1 \to Z_1$$
$$X_3 \to X_3$$
$$Z_3 \to (-1)^b\, ZIZ$$

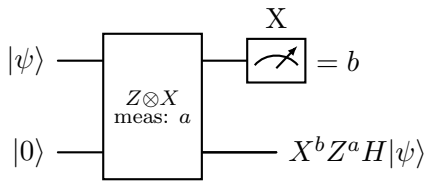
We remove the phases with Z_1^{a+c} and $X_3^b \Longrightarrow$

$$X_1 \to X_1X_3, \quad Z_1 \to Z_1, \quad X_3 \to X_3, \quad Z_3 \to Z_1Z_3. \quad \text{This is CNOT.}$$

We need the measurement outcomes to get the proper Pauli corrections.

H and S from measurement

Clifford is generated by CNOT, H, S. How to get H and S from measurement:



where H: $X \leftrightarrow Z$

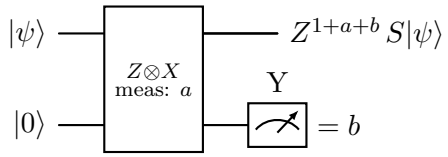
The initial stabilizer is $M = IZ$. $X_1 = XI \equiv XZ$ commutes with the ZX measurement. Then the X measurement gives $\to (-1)^bZ$.

$Z_1 = ZI$ commutes with the ZX measurement. After that measurement, the new stabilizer is $(-1)^aZX$. So now $Z_1 \equiv (-1)^aIX$, not affected by the XI measurement.

Summary: $X \to (-1)^bZ$, $Z \to (-1)^aX$. Hence H, after X^bZ^a removes the phases.

Note: to toggle the basis $X \leftrightarrow Z$, a Z-type or X-type measurement does not suffice — we need the mixed $Z \otimes X$ measurement.

One more Clifford generator: S: $Z \to Z$, $X \to -Y$.

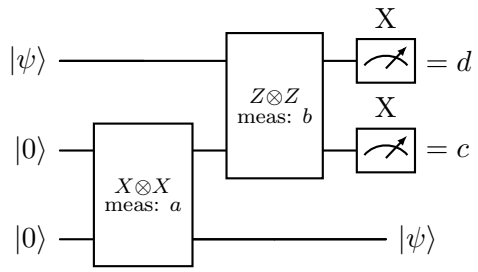


The initial stabilizer is $M = IZ$. $Z_1 \to Z_1$ commutes with the ZX measurement. $X_1 = XI \equiv XZ$, which commutes with the ZX measurement. The new stabilizer is $M = (-1)^aZX$

$$\Longrightarrow \quad XZ \equiv (-1)^a\,(ZX)(XZ) = (-1)^a\, YY.$$

Under the Y measurement, this becomes $(-1)^{a+b}\, Y$; we want $-Y$, so Z^{1+a+b} corrects the phase. Note: to toggle the basis $X \to -Y$, we need a Y measurement.

Remark: By combining the CNOT gadget with one-bit teleportation, we obtain teleportation via only measurements:



(Pauli corrections not shown.) We can move quantum states between code blocks if we can measure XX, ZZ across blocks.

The T gate by measurement

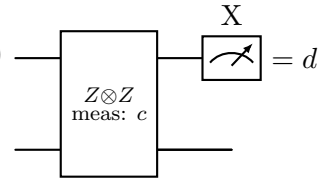
We've seen that $|0\rangle$ prep and Pauli measurement suffice for Clifford computation. But for universality, we need a non-Clifford gate, e.g. the T gate:

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \quad \text{or} \quad T(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} \text{ with } \theta = \pi/4.$$

$$T(\theta)\, X\, T(\theta)^\dagger = \begin{pmatrix} 0 & e^{-i\theta} \\ e^{i\theta} & 0 \end{pmatrix}. \qquad \text{For } \theta = \pi/4 \implies \tfrac{1}{\sqrt{2}} \begin{pmatrix} 0 & 1-i \\ 1+i & 0 \end{pmatrix} = \tfrac{1}{\sqrt{2}}(X+Y).$$

$$TX = \tfrac{1}{\sqrt{2}}(X+Y)\,T \implies T|\pm\rangle \text{ are } \pm \text{ eigenstates of } \tfrac{1}{\sqrt{2}}(X+Y) = \tfrac{1}{\sqrt{2}}\big(|0\rangle + e^{i\pi/4}|1\rangle\big).$$

Do a T gate with prep of $|T(\theta)\rangle = T(\theta)|+\rangle$ and Pauli measurement:



$$\tfrac{1}{\sqrt{2}}\big(|0\rangle + e^{i\theta}|1\rangle\big)\big(a|0\rangle + b|1\rangle\big) = \tfrac{1}{\sqrt{2}}\underbrace{\big(a|00\rangle + e^{i\theta}b|11\rangle\big)}_{ZZ=+1} + \tfrac{1}{\sqrt{2}}\underbrace{\big(b|01\rangle + ae^{i\theta}|10\rangle\big)}_{ZZ=-1}$$

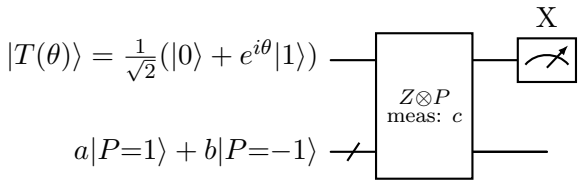
Measure $IX = (-1)^d$:

$$\propto\ a|0\rangle + (-1)^d e^{i\theta}\, b|1\rangle = Z^d\, T(\theta)|\psi\rangle \qquad (ZZ = +1)$$
$$\propto\ b|1\rangle + (-1)^d e^{i\theta}\, a|0\rangle = (-1)^d e^{i\theta}\, Z^d\, T(\theta)^{-1}|\psi\rangle \qquad (ZZ = -1)$$

Depending on the outcome of the $Z\otimes Z$ measurement, we have (up to a Pauli correction) $T(\theta)|\psi\rangle$ or $T(\theta)^{-1}|\psi\rangle$.

For $ZZ = -1$, we need a Clifford correction: $T(\theta)^2 = S^{-1}$ for $\theta = \pi/4$.

Up to a phase, $T(\theta) = e^{i\frac{\theta}{2}Z}$. We can expand the target state $|\psi\rangle$ in terms of eigenstates of any Pauli operator P:



For $c = 0$, this applies $\exp\big(i\frac{\theta}{2}P\big)$, up to Pauli. For $c = 1$, this applies $\exp\big(-i\frac{\theta}{2}P\big)$, up to Pauli.

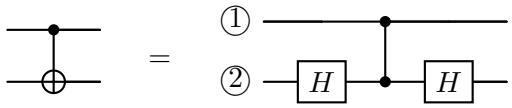
The correction for $c = 1$, $e^{i\theta P}$, is Clifford for $\theta = \pi/4$:

$$e^{i\theta P} = \cos\theta\,\mathbb{1} + i\sin\theta\,P = \tfrac{1}{\sqrt{2}}\big(\mathbb{1} + iP\big) \quad \text{for } \theta = \pi/4$$

$$\begin{aligned} \implies e^{i\theta P}\,Q\,e^{-i\theta P} &= \tfrac{1}{2}\big(\mathbb{1} + iP\big)Q\big(\mathbb{1} - iP\big) = Q \quad \text{if } Q,\ P \text{ commute} \\ \text{or } &= \tfrac{1}{2}\big(Q + iPQ - iQP + PQP\big) = iPQ \quad \text{if they anticommute} \end{aligned}$$

(and a product of Paulis is Pauli).

Pauli propagation by CNOT



$$\begin{aligned} X_1 &\to X_1 Z_2 \to X_1 X_2 \\ Z_1 &\to Z_1 \\ X_2 &\to Z_2 \to Z_2 \to X_2 \\ Z_2 &\to X_2 \to Z_1 X_2 \to Z_1 Z_2 \end{aligned}$$

Pauli-Based Computation

Lecture 13 — 13 May 2026

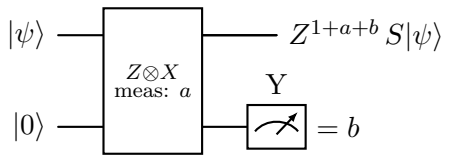
Reference: arXiv: 1506.01396.

We'd like to leverage the ability to perform fault-tolerant measurement of logical Pauli operators in FTQC.

Last time: execution of Clifford generators (CNOT, H, S) via $|0\rangle$ prep and nondestructive Pauli measurement. (And Pauli frame update.)

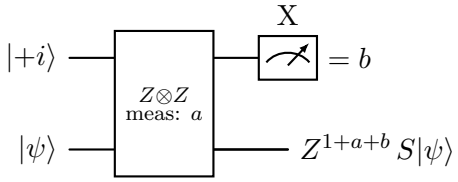
Realizing S by measurement

Example: S: $Z \to Z$, $X \to -Y$ (Heisenberg).



In CSS coding, we might prefer Z-type and X-type measurements. But we need to introduce phases: the $Y = +1$ eigenstate.

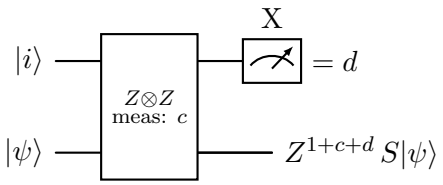
Another way: suppose we can prepare $|+i\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle + i|1\rangle\big)$. (Maybe by distillation.)



The initial stabilizer is YI. How do X, Z propagate?

$$IX \equiv YX \text{ (commutes w/ } ZZ) \equiv (-1)^a\,(ZY)(ZX) = (-1)^a\,XY \;\to\; (-1)^{a+b}\,Y,$$
$$IZ \to IZ \text{ (commutes w/ } ZZ).$$

Let's verify in the Schrödinger picture:



$$S = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}$$

$$\tfrac{1}{\sqrt{2}}\big(|0\rangle + i|1\rangle\big)\big(a|0\rangle + b|1\rangle\big) \begin{cases} \xrightarrow{ZZ=1} & a|00\rangle + ib|11\rangle \\ \xrightarrow{ZZ=-1} & ia|10\rangle + b|01\rangle \quad \text{(equiprobable)} \end{cases}$$

$$\begin{aligned} ZZ = +1,\ X = \pm 1: &\quad \to\ a|0\rangle \pm ib|1\rangle \quad \text{(equiprob.)} \\ ZZ = -1,\ X = \pm 1: &\quad \to\ \pm ia|0\rangle + b|1\rangle \quad \text{(equiprob.)} \\ &\quad =\ \pm i\big(a|0\rangle \mp ib|1\rangle\big) \end{aligned}$$

Hence the output is

$$a|0\rangle + (-1)^{c+d}\, ib|1\rangle = Z^{1+c+d}\big(a|0\rangle - ib|1\rangle\big).$$

Note: If we realize S via measurement as above, using input $|i\rangle$, we can measure Y using ZZ and X measurement, since $S : Y \to X$. Measuring X on the second qubit, with outcome $X = (-1)^e$, has

$$\text{Prob}\,\big|\, a + (-1)^{c+d+e}\, ib\,\big|^2.$$

This is equivalent to Y measurement, up to an outcome flip.

But — for Clifford computation, we can efficiently simulate classically by tracking how Paulis evolve in the Heisenberg picture. To give power to QC we need non-Cliffords ("magic"). How to do this via measurement?

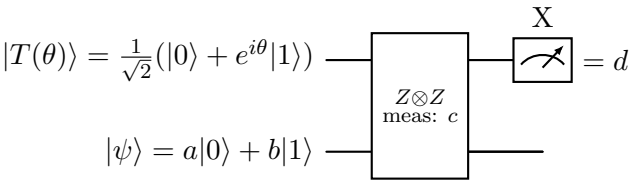
The T gate and $T(\theta)$

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \text{ is non-Clifford:} \qquad TXT^\dagger = \tfrac{1}{\sqrt{2}}(X+Y), \qquad TYT^\dagger = \tfrac{1}{\sqrt{2}}(-X+Y).$$

And Clifford + T is universal.

We can generalize the above Schrödinger picture description:

$$T(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}, \qquad |T(\theta)\rangle = T(\theta)|+\rangle = \tfrac{1}{\sqrt{2}}\big(|0\rangle + e^{i\theta}|1\rangle\big).$$



$$\tfrac{1}{\sqrt{2}}\big(|0\rangle + e^{i\theta}|1\rangle\big)\big(a|0\rangle + b|1\rangle\big) \begin{cases} \xrightarrow{ZZ=1} & a|00\rangle + e^{i\theta}b|11\rangle \\ \xrightarrow{ZZ=-1} & e^{i\theta}a|10\rangle + b|01\rangle \end{cases}$$

$$\begin{aligned} ZZ = +1,\ X = (-1)^d: &\quad a|0\rangle + (-1)^d e^{i\theta}\, b|1\rangle = Z^d\, T(\theta)|\psi\rangle \\ ZZ = -1,\ X = (-1)^d: &\quad (-1)^d e^{i\theta}\, a|0\rangle + b|1\rangle \\ &= (-1)^d e^{i\theta}\big(a|0\rangle + (-1)^d e^{-i\theta}\, b|1\rangle\big) \\ &= \text{phase} \times Z^d\, T(\theta)^{-1}|\psi\rangle \\ &= \text{phase} \times T(-2\theta)\, Z^d\, T(\theta)|\psi\rangle. \end{aligned}$$

To get the desired gate, in addition to the Pauli correction Z^d, we must apply $T(2\theta)$ for $c = 1$.

For $\theta = \pi/4$: $T(\theta) = T$ and $T(2\theta) = S^{-1}$. The correction $Z^d S^{-c}$ is Clifford but not Pauli. We can efficiently propagate it through the subsequent Clifford circuit classically.

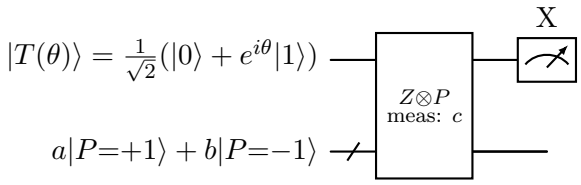
Except: because S toggles between the X and Y bases, when qubits are measured downstream from an S gate, we need to know the outcome $ZZ = (-1)^c$. Hence measurement bases must be adaptive (conditioned on prior measurements), even if S is tracked in software rather than applied. (Contrast with the Pauli frame update — we need the Pauli frame to interpret an outcome as 0 or 1, but not for the choice of measurement basis.)

Generalization to arbitrary Pauli P

A further generalization: if we can prepare $|T(\theta)\rangle$, we apply (up to Pauli frame)

$$U = \exp\bigl(\pm i\,\tfrac{\theta}{2}\,P\bigr)$$

where P is any Pauli operator (of arbitrary weight), where now we measure $Z \otimes P$ rather than $Z \otimes Z$ (notice that $T(\theta) = e^{i\frac{\theta}{2}Z}$):



It's the same analysis as before! To correct the $c = 1$ outcome, apply

$$e^{i\theta P} = \cos\theta\,\mathbb{1} + i\sin\theta\,P = \tfrac{1}{\sqrt{2}}\bigl(\mathbb{1} + iP\bigr) \quad \text{for } \theta = \pi/4.$$

This correction is Clifford for $\theta = \pi/4$. Need to show $\frac{1}{\sqrt{2}}(\mathbb{1} + iP)$ maps Pauli $\to$ Pauli. Of course $Q \to Q$ if Q commutes with P, so suppose P and Q anticommute:

$$\tfrac{1}{\sqrt{2}}\bigl(\mathbb{1} + iP\bigr)\,Q\,\tfrac{1}{\sqrt{2}}\bigl(\mathbb{1} - iP\bigr) = \tfrac{1}{2}\bigl(Q - Q + iPQ - iQP\bigr) = iPQ,$$

which is Pauli if P and Q are.

Pauli-Based Computation (Bravyi–Smith–Smolin 2016)

Now we've seen that universal computation can be executed replacing Clifford + T by measurement gadgets (including nondestructive Pauli measurement); assume inputs $|0\rangle$ and $|T\rangle$ are available. But the Clifford computation can be classically efficiently simulated — only the $|T\rangle$ states seem intrinsically quantum.

In fact, further simplification is possible:

> "$|T\rangle$ states and (nondestructive) Pauli measurements are all you need!"

- Measurements are adaptive.
- Pauli + Clifford frame tracked in software.
- Polynomial-time classical computation needed.
- Measurements can be chosen to be mutually commuting!

Consider a poly(n)-size quantum circuit on n qubits with m T gates + Clifford gates. It can be efficiently simulated (including classical processing) by computation with m T-state inputs (and no other inputs) and a sequence of Pauli product measurements on the m qubits. Furthermore, these (nondestructive, adaptive) measurements can be chosen to be mutually commuting. Because no more than m commuting measurements on m qubits are independent, no more than m measurement rounds are needed.

Let's suppose that, in the circuit to be simulated, qubits are initialized as $|0\rangle$ and measured at the end in the Z basis. We've seen how to simulate with adaptive measurement if we add an input $|T\rangle$ for each T gate, and an input $|0\rangle$ for each Clifford gate. But we don't really need the input $|0\rangle$ for each Clifford, because we can simulate Clifford classically by propagating Paulis through Pauli measurements until the first T gate is encountered. And beyond, because only Pauli measurements are encountered by stabilizer states, even though there are also input T states. We use the info to update the Clifford frame.

Choosing commuting PPMs

We can choose the PPMs (Pauli product measurements) to commute. Here is why. Suppose in round t, P_t is the first Pauli measurement that anticommutes with a Pauli P_s measured in round $s < t$. The outcome of the P_s measurement was $\sigma_s \in \{1, -1\}$. Hence the input state $|\psi\rangle$ to round t satisfies $P_s|\psi\rangle = \sigma_s|\psi\rangle$. Therefore,

$$\langle\psi|P_t|\psi\rangle = \langle\psi|P_s^\dagger P_t P_s|\psi\rangle = -\langle\psi|P_t|\psi\rangle \implies \langle\psi|P_t|\psi\rangle = 0 \implies \text{outcome } \sigma_t \text{ is 50-50 random.}$$

After the measurement, the state has been projected onto

$$|\psi'\rangle = \tfrac{1}{2}\big(P_t + \sigma_t \mathbb{1}\big)|\psi\rangle \qquad \text{(up to a } \sqrt{2} \text{ norm factor).}$$

Hence

$$|\psi'\rangle = \tfrac{1}{\sqrt{2}}\big(P_t + \sigma_s\sigma_t P_s\big)|\psi\rangle.$$

If P_t and P_s anticommute, then $P_t + \sigma P_s$ is Clifford (for $\sigma = \pm 1$). This means we don't need to do the measurement. We can simulate it perfectly by picking $\sigma_t \in \{\pm 1\}$ uniformly at random and inserting the known Clifford $\frac{1}{\sqrt{2}}(P_t + \sigma P_s)$. This Clifford can be propagated forward to the end of the circuit, with a Clifford frame update accordingly.

Inputs and outputs

What about the n input states $|0\rangle^{\otimes n}$ to the circuit to be simulated? Think of those as $\sigma = 1$ outcomes of Z measurements on those n qubits. By the above procedure we choose all subsequent PPMs to commute with these — i.e. they act trivially, and the qubits themselves are never needed.

What about the measurement of Z_i on each of the simulated circuit's output qubits? We can propagate Z_i backward through the circuit. It becomes the measurement of a Pauli operator acting on the m input T states, up to known Pauli frame updates.

If m independent commuting Paulis have been measured in m rounds, that provides the complete stabilizer of an m-qubit state determined by the measurement outcomes. If the back-propagated Z_i is in this stabilizer, it is determined by the PPM outcomes. Else it anticommutes with some element of the stabilizer, has a uniformly random outcome, and I simulate it by flipping a coin.

What if fewer than m independent measurements are performed, and our output observable of interest commutes with the stabilizer but is not in the stabilizer (hence not determined by the PPM outcomes)? Measure additional commuting operators!

Cluster States and the One-Way Quantum Computer

Lecture 14 — 18 May 2026

Last time: Pauli-based computation. Universal QC via nondestructive measurements (chosen adaptively) of high-weight Pauli product operators, where all inputs are $|T\rangle$ states. This is useful for FTQC: we can do reliable universal QC if we can prepare clean $|T\rangle$ states and perform FT measurements. We'll discuss how to build FT architectures based on this idea, but not until the next lecture.

First, though, we'll discuss a related type of measurement-based QC (MBQC), in which the computation proceeds via adaptive *single-qubit measurements* acting on an *entangled many-qubit resource state*. This is conceptually interesting, and also has applications to FT — for example in photonic QC, where each photon is measured shortly after entanglement with other photons is established. As we'll discuss, it is also an instructive example of a *symmetry-protected topological phase* (SPT phase), an important idea in quantum condensed matter physics.

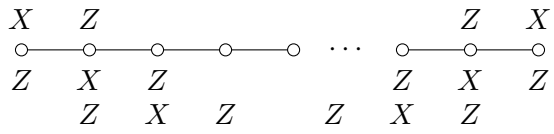
This scheme is sometimes called the "one-way quantum computer" because the entangled resource is irreversibly consumed by destructive measurements as the computation proceeds.

For universal QC we'll need a 2D resource state. But first consider 1D as a warm-up example.

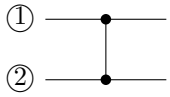
The 1D cluster state

The 1D cluster state (also called a graph state) is a stabilizer state: the simultaneous +1 eigenstate of the Pauli operators

$$X_1 Z_2, \qquad Z_{i-1} X_i Z_{i+1} \quad (i = 2, 3, \ldots, n-1), \qquad Z_{n-1} X_n.$$



These commute: 2 or 0 collisions of X with Z. The state is easy to prepare: start with $|+\rangle^{\otimes n}$ and apply CZ to all neighboring pairs.



CZ is symmetric under ① ↔ ②.
It commutes with Z_1, Z_2.

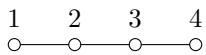
CZ takes $X_1 = \pm 1$ eigenstates to $X_1 Z_2 = \pm 1$ eigenstates:

$$|\pm\rangle|\psi\rangle \;\longrightarrow\; \tfrac{1}{\sqrt{2}}\big(|0\rangle|\psi\rangle \pm |1\rangle\, Z|\psi\rangle\big).$$

Hence

$$\text{CZ}: \quad X_1 \to X_1 Z_2, \quad X_2 \to Z_1 X_2, \quad Z_1 \to Z_1, \quad Z_2 \to Z_2.$$

For a 4-site chain,



$$\mathrm{CZ}(1,2)\,\mathrm{CZ}(2,3)\,\mathrm{CZ}(3,4): \quad \begin{aligned} X_1 &\to X_1 Z_2 \\ X_2 &\to Z_1 X_2 Z_3 \\ X_3 &\to Z_2 X_3 Z_4 \\ X_4 &\to Z_3 X_4 \end{aligned}$$

Hence $|+\rangle^{\otimes 4} \to$ cluster state. All CZ gates commute and can be done simultaneously in depth = 1. And this works for any graph:

$$X_i \xrightarrow{\text{CZs}} X_i \bigotimes_{j\in\partial i} Z_j, \qquad \text{where } \partial i \text{ means the set of vertices connected to } i \text{ by edges.}$$

The cluster state is a unique stabilizer state for any graph (number of independent stabilizer generators = number of qubits). Obtain encoded information by removing one or more stabilizers. E.g., in 1D remove the stabilizer generator at the left boundary, $X_1 Z_2$. Now there is an encoded qubit with logical operators

$$\overline{Z} = Z_1, \qquad \overline{X} = X_1 Z_2.$$

These anticommute with each other, and commute with the stabilizer.

Measuring the qubits one at a time

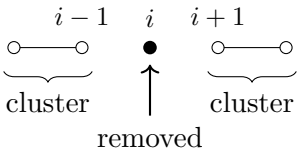
Consider measuring the qubits one at a time, starting at the left boundary.

Measure $Z_1 = \overline{Z}$. This is a Z measurement of a logical qubit.

Measure Z_i in the chain interior. Outcome $Z_i = (-1)^a$. Then:

$$\begin{aligned} Z_{i-2}\,X_{i-1}\,Z_i &\to (-1)^a\,Z_{i-2}\,X_{i-1}. && \text{New stab.} \\ Z_i\,X_{i+1}\,Z_{i+2} &\to (-1)^a\,X_{i+1}\,Z_{i+2}. && \text{New stab.} \\ Z_{i-1}\,X_i\,Z_{i+1} &\to Z_i = (-1)^a. && \text{New stab.} \end{aligned}$$

This removes the ith vertex, and cuts the chain into two disjoint cluster states (up to Pauli correction Z_{i-1}^a, Z_{i+1}^a):

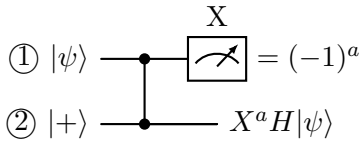


Measure $X_1 = (-1)^a$. This removes the left boundary site:

$$\begin{aligned} \overline{X} = X_1 Z_2 &\to (-1)^a Z_2 = (-1)^a\,\overline{Z}'. && \text{The logical } Z \text{ of the shortened chain.} \\ \overline{Z} = Z_1 \equiv X_2 Z_3 &= \overline{X}'. && \text{The logical } X \text{ of the shortened chain.} \end{aligned}$$

This is a logical Hadamard, up to Pauli correction X^a.

We can imagine preparing the cluster state one CZ at a time, in each step followed by an X measurement:



Stabilizer: $X_2 \xrightarrow{\text{CZ}} Z_1 X_2$, a new stabilizer. Then $Z_1 \equiv X_2$, and

$$X_1 \to X_1 Z_2 \to (-1)^a Z_2.$$

This is H up to Pauli:

$$H:\ X \to Z,\ Z \to X, \qquad X^a:\ Z \to (-1)^a Z.$$

Now apply CZ$(2,3)$ to $X^a H|\psi\rangle_2 \otimes |+\rangle_3$ and measure $X_2 = (-1)^b$. Propagation:

$$|\psi\rangle \ \to\ X^b H X^a H|\psi\rangle = X^b Z^a |\psi\rangle.$$

Measuring the two sites at the left boundary preserves $|\psi\rangle$ up to a Pauli frame update determined by the X measurements.

Single-qubit Cliffords and the T gate

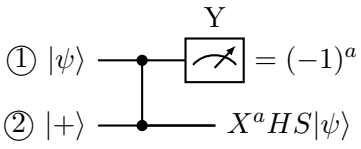
To generate the single-qubit Clifford group, we need S in addition to H:

$$S = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}: \qquad Z \to Z, \quad X \to -Y.$$

Alternatively, HS:

$$X \xrightarrow{S} -Y \xrightarrow{H} Y, \qquad Z \xrightarrow{S} Z \xrightarrow{H} X.$$

Obtain this by measuring Y:

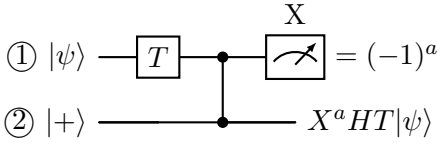


Stab: $X_2 \to Z_1 X_2$.

Then $Z_1 \equiv X_2$, and

$$X_1 \ \to\ X_1 Z_2 \equiv Y_1 Y_2 \ \to\ (-1)^a Y_2 \xRightarrow{X_2^a} Y_2.$$

For universal 1-qubit computation, we also need the T gate. From the above, applying CZ$(1,2)$ and M_{X_1} to the input state $T|\psi\rangle_1$, we have:



And the T gate commutes through CZ.

The combination T followed by M_X is measurement in the basis

$$T|\pm\rangle = \tfrac{1}{\sqrt{2}}\big(|0\rangle \pm e^{i\pi/4}|1\rangle\big), \qquad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}.$$

We apply the above circuit to $X^b|\psi\rangle$, where b is determined by how many times we have obtained outcome (-1) in preceding X and Y measurements (whether that number is even or odd). The outcome is $X^a HTX^b|\psi\rangle$, and we want to pull X^b out in front to update the Pauli frame. But (up to a phase) $TX = XT^{-1}$:

$$TX = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ e^{i\pi/4} & 0 \end{pmatrix},$$
$$XT^{-1} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{pmatrix} = \begin{pmatrix} 0 & e^{-i\pi/4} \\ 1 & 0 \end{pmatrix} = e^{-i\pi/4}\,TX.$$

Hence

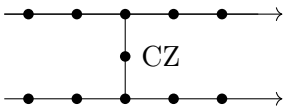
$$\begin{aligned} X^a HTX^b|\psi\rangle &= X^a Z^b HT^{-1}|\psi\rangle \quad \text{for } b = 1 \\ &= X^a HT|\psi\rangle \quad \text{for } b = 0. \end{aligned}$$

If we want to apply T, then measure in basis $T|\pm\rangle$ for $b = 0$, and in basis $T^{-1}|\pm\rangle$ for $b = 1$. The measurement basis is chosen adaptively.

Universality: 2D cluster states

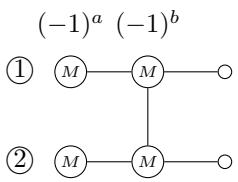
We have generators for the 1-qubit Clifford group. For universality we need CNOT or CZ. For this:

1. Build a 2D cluster state on the square lattice.
2. Carve out 1D *wires* by removing sites from the lattice via Z measurement (or we could have created a graph state with the desired topology to begin with).
3. X measurement propagates qubits along wires. CZ gate (up to conjugation by H) as shown:



4. Perform 1Q Clifford + T via adaptive measurement as described above.

We need to understand how interaction between wires results in a CZ gate. We follow how logical operators propagate when X is measured:



Multiply by elements of the stabilizer so that the support of the logicals on measured sites is X or I.

Writing operators as two rows (wire ① over wire ②), with columns labeling the sites along each wire:

$$\overline{Z}_1 = \begin{matrix} Z & I & I \\ I & I & I \end{matrix} \equiv \begin{matrix} I & X & Z \\ I & Z & I \end{matrix} \equiv \underbrace{\boxed{\begin{matrix} I & X \\ I & I \end{matrix}}}_{\text{measure } X} \begin{matrix} Z & I \\ X & Z \end{matrix} \longrightarrow (-1)^b \begin{matrix} Z & I \\ X & Z \end{matrix} = (-1)^b \overline{Z}_1 \overline{X}_2,$$

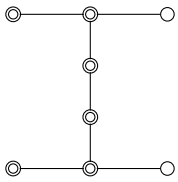
$$\overline{X}_1 = \begin{matrix} X & Z & I \\ I & I & I \end{matrix} \equiv \underbrace{\boxed{\begin{matrix} X & I \\ I & I \end{matrix}}}_{\text{measure } X} \begin{matrix} X & Z \\ I & I \end{matrix} \longrightarrow (-1)^a \begin{matrix} X & Z \\ I & I \end{matrix} = (-1)^a \overline{X}_1.$$

Remove the phases with $\overline{Z}_1^a \overline{Z}_2^b$. By symmetry: $\overline{Z}_2 \to \overline{Z}_2 \overline{X}_1$, $\overline{X}_2 \to \overline{X}_2$, up to Pauli.

This is the Hadamard-rotated CZ gate:

$$\left.\begin{aligned} X_1 &\xrightarrow{H} Z_1 \to Z_1 X_2 \xrightarrow{H} X_1 Z_2 \\ Z_1 &\xrightarrow{H} X_1 \to X_1 \xrightarrow{H} Z_1 \end{aligned}\right\} \quad \text{and similarly,} \quad X_2 \to Z_1 X_2, \quad Z_2 \to Z_2.$$

The same works when the coupler between the wires passes through intermediate sites. Measure the colored sites:

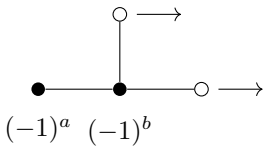


Writing rows for wire ①, the two coupler qubits, and wire ②:

$$\overline{Z}_1 = \begin{matrix} Z & I & I \\ & I & \\ I & I & I \end{matrix} \equiv \begin{matrix} I & X & Z \\ & Z & \\ I & I & I \end{matrix} \equiv \begin{matrix} I & X & Z \\ & I & \\ I & Z & I \end{matrix} \equiv \underbrace{\boxed{\begin{matrix} I & X \\ I & \\ X & \\ I & I \end{matrix}}}_{\text{measure } X} \begin{matrix} Z & I \\ & \\ & \\ X & Z \end{matrix} \longrightarrow \overline{Z}_1 \overline{X}_2 \text{ up to Pauli.}$$

Alternative formulation: CNOT

Consider the elementary graph with one qubit attached above the second site of a 3-site chain. The two open sites (with arrows) are the outputs; the two filled sites are measured in the X basis, with outcomes $(-1)^a$ and $(-1)^b$:



Writing operators as two rows (top qubit over the 3-site chain), the stabilizer is

$$\begin{matrix} Z & & I \\ Z\,X\,Z & & I\,Z\,X \end{matrix}$$

and the logicals are

$$\overline{Z}_1 = \begin{matrix} Z \\ I\,I\,I \end{matrix} \qquad \overline{Z}_2 = \begin{matrix} I \\ Z\,I\,I \end{matrix} \qquad \overline{X}_1 = \begin{matrix} X \\ I\,Z\,I \end{matrix} \qquad \overline{X}_2 = \begin{matrix} I \\ X\,Z\,I \end{matrix}$$

Now multiply the logicals by stabilizers to obtain equivalent logicals:

$$\overline{Z}_2 \equiv \begin{matrix} Z \\ I\,X\,Z \end{matrix} \qquad \overline{X}_1 \equiv \begin{matrix} X \\ I\,I\,X \end{matrix} \qquad \overline{X}_2 \equiv \begin{matrix} I \\ X\,I\,X \end{matrix}$$

Measuring the two filled sites in the X basis then gives

$$\overline{Z}_2 \to (-1)^b \begin{matrix} Z \\ \circ\ \circ\ Z \end{matrix} \qquad \overline{X}_1 \to \begin{matrix} X \\ \circ\ \circ\ X \end{matrix} \qquad \overline{X}_2 \to (-1)^a \begin{matrix} I \\ \circ\ \circ\ X \end{matrix}$$

Thus

$$\begin{aligned} Z_1 &\to Z_1, & X_1 &\to X_1 X_2, \\ Z_2 &\to (-1)^b\, Z_1 Z_2, & X_2 &\to (-1)^a\, X_2. \end{aligned}$$

This is CNOT; the phase can be removed by $X_2^b\, Z_1^a\, Z_2^a$.

We have seen how to perform universal QC with adaptive single-qubit measurements acting on a 2D cluster state. Furthermore, we can prepare the cluster state "on the fly" using only CZ gates.

Lattice Surgery

Lecture 15 — 20 May 2026

References: arXiv: 1808.06709, arXiv: 1808.02892.

So far, we have described high-rate codes and also how universal quantum computing can be executed as a sequence of nondestructive Pauli-product measurements. Next we want to consider fault-tolerant implementation of Pauli-based computation using high-rate codes. But before we get to that, we'll discuss FTQC in the surface code.

In PBC, inputs are $|T\rangle$ states. For now, we'll suppose high-quality $|T\rangle$ states are provided — in practice they'll be prepared by a method such as magic state distillation or cultivation. To compute reliably, then, it will suffice to perform FT nondestructive PPMs, chosen adaptively.

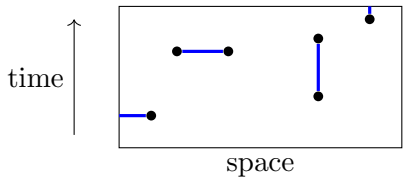
We'll consider two different methods, suited for different hardware modalities. The first is the *surgery* method, which is applicable to a platform in which qubits have fixed positions in a 2D layout with nearest-neighbor connectivity (e.g., a superconducting processor). The second is the *transversal* method, which is applicable if Clifford operations such as CNOT and CZ are implemented transversally by moving encoded surface code blocks, e.g., in an atomic processor.

An advantage of the transversal method is that it substantially reduces the time blowup due to FT — $O(1)$ rounds of syndrome extraction in between logical operations instead of $O(d)$ rounds for surgery (where d is the code distance) — at the cost of somewhat increasing the physical qubit count, and increasing the complexity of (real-time) syndrome decoding, as well as requiring qubit mobility.

Syndrome decoding

We discussed syndrome decoding in the surface code when there are both qubit errors and syndrome errors, for the purpose of maintaining a quantum memory. If we want to compute, how many SE rounds should be performed in between logical gates (which can spread errors to other code blocks)? The standard view (to be revisited later) is that for a distance-d code we should do d SE rounds between operations.

Even for the case of quantum memory, we may use a "sliding-window" decoder that takes a finite number of SE rounds as input. Recall we may apply MWPM to the syndrome history in spacetime:

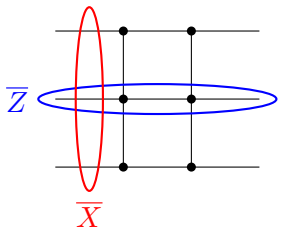


We match defects (endpoints of syndrome strings) to each other, or to spatial boundaries, or to past/future boundaries. Match to the future boundary when we lack confidence in the syndrome's authenticity. If the window is too short in the time direction, we'll fail to correct too many authentic errors.

To enjoy the full benefit of distance d, the window's time extent should roughly match its spatial extent $\implies O(d)$ SE rounds inside the window. We need to repeat this many times to gain sufficient confidence in the syndrome, before allowing a block to contact other blocks.

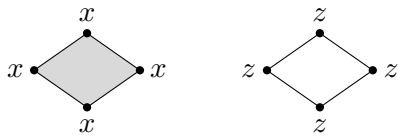
Surface code: rotated and unrotated

Consider the $d = 3$ "unrotated" $[[13, 1, 3]]$ surface code with qubits on edges:

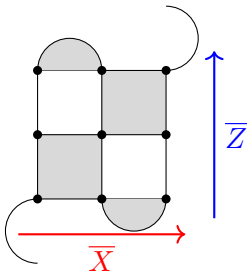


13 edges
6 plaquettes: 2 wt. 4 and 4 wt. 3
6 sites: 2 wt. 4 and 4 wt. 3

There is an alternative depiction with qubits on sites, and X and Z stabilizers on plaquettes:



The "rotated" $[[9, 1, 3]]$ surface code is more efficient ($n = 9$ instead of $n = 13$):



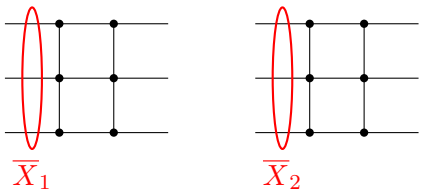
X and Z check ops collide on 0 or 2 sites.

Now 4 X-type stab: 2 wt 4 and 2 wt 2;
4 Z-type stab: 2 wt 4 and 2 wt 2.

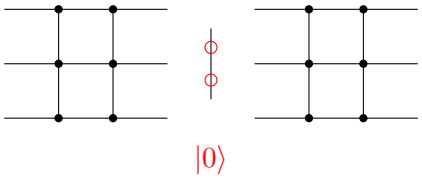
(Shaded plaquettes and lobes are X-type checks; white are Z-type.) The boundary check ops determine how $\overline{X}$ and $\overline{Z}$ are oriented.

Measuring $\overline{X}_1\overline{X}_2$ by merging and splitting

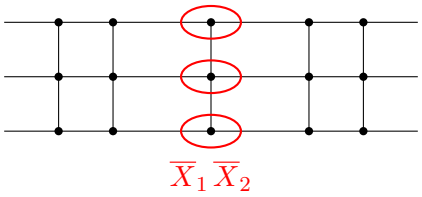
Suppose we want to measure XX or ZZ for two neighboring blocks using only geometrically local measurements. First, consider two blocks of unrotated surface code. Consider XX for two $d = 3$ blocks (a weight-6 operator). We do the measurement by first merging and then splitting the two blocks.



Initialize 2 qubits in the $|0\rangle$ state between the blocks:

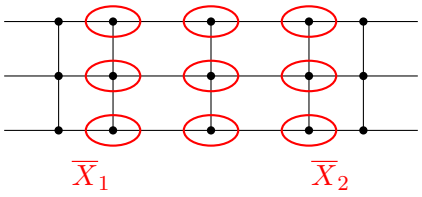


and measure the X stabilizers of the merged code (Z stabilizers are satisfied automatically). The three site outcomes X_s along the seam are random. Their product is $\overline{X}_1\overline{X}_2$!



A measurement error in any of the site operators X_s will flip our value of $\overline{X}_1\overline{X}_2$. We need to measure the syndrome $O(d)$ times to have confidence in the measurement outcomes. Finally, we split the blocks by measuring the two middle qubits in the Z basis, obtaining random outcomes $|0\rangle$ or $|1\rangle$. The $|1\rangle$ outcome flips the values of the boundary Z check ops in the blocks, which we can correct or absorb into the Pauli frame.

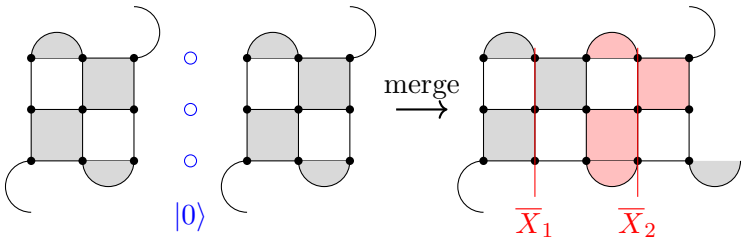
Note that we can also obtain $\overline{X}_1\overline{X}_2$ from the parity of a larger set of X_s outcomes, which are bounded on left and right by X-string logical operators:



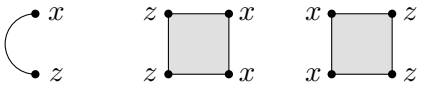
Measuring $\overline{Z}_1\overline{Z}_2$ is the same procedure, but in the dual basis. Now we merge by introducing 2 qubits in the $|+\rangle$ state and measure three Z_p operators. Their parity is $\overline{Z}_1\overline{Z}_2$. Then split by measuring in the X basis.

Mixed-type operators: leaving the CSS world

What if we want to measure mixed-type operators like $\overline{X}_1\overline{Z}_2$? To do that we need to leave the CSS world — that is, measure stabilizers containing both X and Z. Let's consider this in the rotated surface code. First consider $\overline{X}_1\overline{X}_2$ measurement. We initialize 3 qubits as $|0\rangle$ and measure 4 X-type stabilizers:



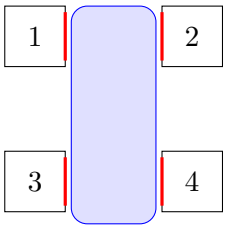
Now $\overline{X}_1\overline{X}_2$ is the parity of the 4 check ops colored red. Can split by Z measurement. For $\overline{X}_1\overline{Z}_2$, we'll measure stabilizers like XZ and $XXZZ$:



We merge $\overline{X}_1$ (an X-type boundary of block 1) with $\overline{Z}_2$ (a Z-type boundary of block 2) through a seam of ancillas, which can be initialized as $|0\rangle$ and split with Z measurement. Again, a product of 4 checks, 2 of which are mixed, gives $\overline{X}_1\overline{Z}_2$.

High-weight Pauli products

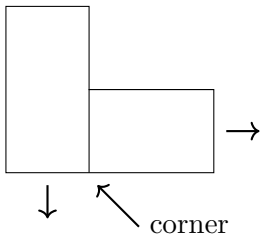
Using this trick of mixed stabilizers, we can measure high-weight logical Paulis with a mix of Xs and Zs. We initialize ancillas which form a *sea* that can be connected to the X or Z boundaries of various blocks, so that the desired logical operator is the parity of check operator outcomes; then split the blocks by measuring all the ancillas.



Measures: $Z_1X_2X_3Z_4$

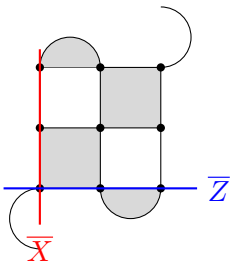
(The red segments indicate where the ancilla sea contacts each block's boundary, of X type or Z type as appropriate.)

Note that a block can turn around at a corner, where there is a weight-3 check:



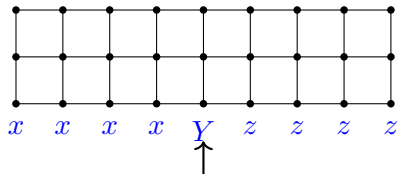
To measure a Pauli product that contains Y, though, we need a further trick: a "twist" defect.

The logical $\overline{Y}$ is the product of $\overline{X}$ and $\overline{Z}$ (up to a phase). On a $d \times d$ block, this is weight $2d-1$, with $d-1$ Xs, $d-1$ Zs, and a Y at the corner:

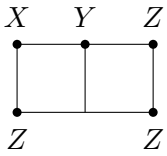


To obtain this operator as a product of stabilizers, we need a stabilizer containing Y.

We can "straighten out" the corner, with Z and X edges meeting where Y is inserted. It looks like this:



We want to glue this to a Z boundary. To get the edges to match, we'll need a weight-5 stabilizer (the "twist"):



When we merge the blocks, the deformed code contains a strip of checks along the seam: black (shaded) checks are X-type, white are Z-type, mixed checks appear near the twist, and red indicates which checks are multiplied to yield the target logical operator. Take the product of the checks colored red: this is $\overline{Y}_1\overline{Z}_2$! You can verify that all checks commute.

Gauging Logical Operators

Lecture 16 — 27 May 2026

Reference: arXiv: 2410.02213.

We have seen that universal quantum computation can be reduced to (1) preparation of (high-quality) non-Clifford states, such as $|T\rangle$ states; (2) measurement (fault tolerantly) of (possibly high-weight) logical Pauli product operators. This is called *Pauli-based computation* (PBC).

Taking magic-state preparation for granted, we discussed fault-tolerant Pauli product measurement in the surface code. We would like to generalize the *lattice surgery* method that works for the surface code to other qLDPC codes, including high-rate codes.

A crucial feature of the surface code is that syndrome measurement requires only geometrically local connectivity in a 2D layout, and lattice surgery is formulated to be compatible with that geometric constraint. In high-rate codes we already allow geometric nonlocality in syndrome measurement, so there is no reason to insist on local connectivity in logical processing. But other features of lattice surgery provide guidance.

Viewed broadly, lattice surgery involves these steps/features:

- Introduce additional ancilla qubits initialized in a product state.
- Define a "deformed code" whose checks act on both the original data qubits and the ancilla (including some checks that act on both).
- The deformed code is designed so that the target logical operator to be measured is a product of the (low-weight) check operators of the deformed code.
- The deformed code should be qLDPC (have low-weight checks) and should have a distance comparable to the distance of the original data code.
- We measure the checks of the deformed code multiple times, to have high confidence in their value.
- We infer the value of the target logical by taking the parity of the outcomes of the appropriately chosen check operators.
- We measure the ancilla qubits to remove them from the deformed code and return to the original code.

(Before, I spoke of a "merged code." Now I'm saying "deformed code" because in some cases we'll consider PPMs within a single code block rather than a product of Paulis in different blocks.)

There are two (related) ways of looking at lattice surgery, each suggesting a method for generalizing it, described in these references:

Williamson & Yoder, arXiv: 2410.02213	"Low-overhead fault-tolerant quantum computation by gauging logical operators" (W-Y)
Ide, Gowda, Nadkarni, Dauphinais, arXiv: 2410.02753	"Fault-tolerant logical measurements via homological measurements" (Xanadu)

Gauging logical operators

A quantum *subsystem code* is defined by a set of (typically low weight) generators which are not necessarily commuting, and the code stabilizer is the center of the gauge group (hence stabilizer generators are expressible as products of gauge generators). There are logical operators that commute with the gauge group but are not contained in the gauge group ("bare" logical operators). We may also consider "dressed" logical operators, obtained by multiplying bare logical operators by elements of the gauge group. We don't try to protect the gauge qubits; rather, the code protects the (bare) logical operators, which may have much higher weight than the gauge generators. What is useful is that we can determine the (high-weight) stabilizers by measuring the (low-weight) gauge operators.

When we speak of "gauging logical operators," we mean that the deformed code may be regarded as a subsystem code, such that the high-weight target logical operator can be expressed as a product of low-weight gauge generators.

Example of subsystem code: the Bacon–Shor code. The code is $[[9, 1, 3]]$.

$$\overline{X} = \begin{pmatrix} X & X & X \\ I & I & I \\ I & I & I \end{pmatrix} \qquad \overline{Z} = \begin{pmatrix} Z & I & I \\ Z & I & I \\ Z & I & I \end{pmatrix} \qquad \text{are logical Paulis.}$$

Stabilizer:

$$S_{X1} = \begin{pmatrix} X & X & X \\ X & X & X \\ I & I & I \end{pmatrix} \quad S_{X2} = \begin{pmatrix} I & I & I \\ X & X & X \\ X & X & X \end{pmatrix} \quad S_{Z1} = \begin{pmatrix} Z & Z & I \\ Z & Z & I \\ Z & Z & I \end{pmatrix} \quad S_{Z2} = \begin{pmatrix} I & Z & Z \\ I & Z & Z \\ I & Z & Z \end{pmatrix}$$

Each weight-6 stabilizer generator is a product of 3 weight-2 gauge generators, e.g.

$$S_{X1} = \begin{pmatrix} X & I & I \\ X & I & I \\ I & I & I \end{pmatrix} \begin{pmatrix} I & X & I \\ I & X & I \\ I & I & I \end{pmatrix} \begin{pmatrix} I & I & X \\ I & I & X \\ I & I & I \end{pmatrix}.$$

Each gauge generator commutes with the stabilizer and the (bare) logicals $\Longrightarrow$ we can determine stabilizer generators by measuring low-weight gauge generators. "Fix the gauge" $\rightarrow$ Shor code.

Homological measurement

Recall from Lecture 8 the connection between CSS codes and chain complexes:

$$V_2 \underset{\delta_1}{\overset{\partial_2}{\rightleftarrows}} V_1 \underset{\delta_0}{\overset{\partial_1}{\rightleftarrows}} V_0$$

$$\partial_2 = H_Z^T, \qquad \partial_1 = H_X, \qquad \delta_1 = \partial_2^T = H_Z, \qquad \delta_0 = \partial_1^T = H_X^T.$$

Hence

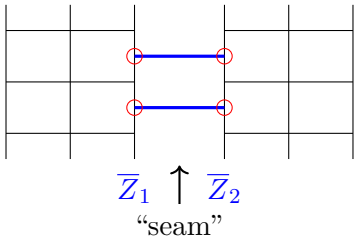
$$\partial_1 \circ \partial_2 = H_X H_Z^T = 0,$$
$$\delta_1 \circ \delta_0 = H_Z H_X^T = 0.$$

$$\mathcal{L}_Z = \frac{\mathrm{Ker}(\partial_1)}{\mathrm{Im}(\partial_2)} = \frac{\mathrm{Ker}(H_X)}{\mathrm{Im}(H_Z^T)} \qquad \text{commutes with } X\text{-stab and not in } Z\text{-stab,}$$

$$\mathcal{L}_X = \frac{\mathrm{Ker}(\delta_1)}{\mathrm{Im}(\delta_0)} = \frac{\mathrm{Ker}(H_Z)}{\mathrm{Im}(H_X^T)} \qquad \text{commutes with } Z\text{-stab and not in } X\text{-stab.}$$

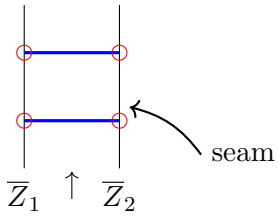
When we speak of "homological measurement" of a logical operator, we mean that the target logical (let's say a Z-type operator), which is closed but not exact in the original code (in $\mathrm{Ker}(\partial_1)$ but not in $\mathrm{Im}(\partial_2)$), becomes *exact* in the deformed code (in $\mathrm{Im}\,\partial_2'$, the image of the deformed boundary operator). If the deformed code is qLDPC, this means the logical becomes expressible as a product of the low-weight checks of the deformed code.

Consider lattice surgery in the surface code from the perspective of homological measurement, e.g. the measurement of $\overline{Z}_1\overline{Z}_2$ on two blocks of the unrotated surface code:



We add the two ancilla qubits, shown as horizontal blue edges, and sew the two blocks together at the seam.

In the deformed code, 4 weight-3 site operators are promoted to weight-4 site operators, and 3 new plaquette operators are added. The target logical $\overline{Z}_1\overline{Z}_2$ is the product of these 3 plaquette operators (the blue horizontal edges across the seam cancel out).



In this case, the logical really is the boundary of the set of plaquettes at the seam.

How to interpret this procedure as "gauging" the logical operator? We can say that we have added 3 gauge generators, the 3 new plaquettes, all of which fail to commute with checks of the original code, but their product yields $\overline{Z}_1\overline{Z}_2$.

How to "gauge logical operators"

We follow W-Y. Suppose without loss of generality that the target logical operator is of X-type. We'll introduce ancillas and deform the original code to a subsystem code which has X-type gauge generators, acting on the original qubits and the ancilla qubits, with low weight, as well as low-weight Z-type stabilizers acting only on ancillas. We want the target X-type logical to be a product of low-weight X-type gauge operators, and we want the distance of the deformed subsystem code (lowest weight of a dressed logical operator) to be no less than the distance of the original code. It is also important that each qubit participates in $O(1)$ gauge generators.

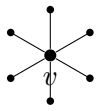
In the protocol for PPM, we measure the gauge generators $O(d)$ times. Because the code is a subsystem code, the X-type and Z-type generators do not commute, and hence fluctuate during

the d rounds. But the logical we decode is a gauge operator of the subsystem code (commutes with all gauge generators) and hence can be determined from the syndrome history (as in the Bacon–Shor code).

Measurement of the X-type logical operator L is achieved by a subsystem code associated with a graph $G(V, E)$ which is required to have certain specified properties. The vertices $v \in V$ of the graph correspond to qubits in the support of L. The edges in E have the property that all cycles in the graph are generated by a set P of cycles of constant length (here P stands for "plaquettes," not for Pauli — these short cycles are analogous to the plaquettes of the surface code). We associate an ancilla qubit with each edge $e \in E$.

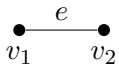
To construct the deformed subsystem code, we add gauge generators associated with each $v \in V$ (each qubit in the support of L):

$$A_v = X_v \bigotimes_{e \ni v} X_e$$



A product of X acting on v and X acting on each of its neighboring edges. In physics terminology this may be called a "Gauss law operator." Note that

$$L = \bigotimes_{v} X_v = \bigotimes_{v} A_v$$

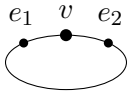


because each edge has two neighboring vertices, so two factors of X_e on edge e cancel out.

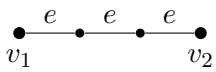
In addition, there are gauge generators of Z-type, "flux operators" associated with all the short generating cycles on G:

$$B_p = \bigotimes_{e \in p} Z_e.$$

These are actually in the code stabilizer; they commute with all A_v because a cycle has support on an even number of edges neighboring v:



The Z-type stabilizers of the original code may fail to commute with some of the A_v gauge generators. We know L commutes with the original stabilizer, so the Z-type stabilizer generator has support on an even number of L's qubits — to be concrete, suppose the number is 2.



We will assume that for any pair of vertices both in the support of any Z-type generator of the original code, there is an edge path on the graph, of constant length, that connects them. Then we apply Z_e to each of these edges to obtain a new stabilizer that commutes with the gauge generators of the deformed code.

It is important that each qubit participates in $O(1)$ gauge generators, in that we want to be able to measure all the generators in constant depth.

Distance of the deformed code

For fault tolerance, we would like the deformed code to have large distance. We introduced a lot of ancilla qubits, and we worry that there could be logical operators in the deformed code that have lower weight than the distance of the original code.

We don't need to worry about Z logicals supported only on ancillas. To commute with all A_v, a Z operator needs to be a cycle of $G(V, E)$, and by construction all cycles are in the stabilizer, hence can't be nontrivial logicals.

More concerning are X operators with support on vertices. Can these be partially cleaned with the gauge generators $\{A_v\}$, resulting in a "dressed" logical operator with low weight?

Cleaning a product of X_v, using an A_v to remove each X_v, results in an operator with support on the edges in the boundary of the set where the original operator had support:

$$\bigotimes_{v\in S} X_v \quad \overset{\text{gauge}}{\underset{\text{equivalent}}{\sim}} \quad \bigotimes_{e\in\partial S} X_e.$$

We'll suppose the graph $G(V, E)$ is an *expander* with the property

$$|\partial S| \geq |S| \quad \text{for } |S| \leq |V|/2.$$

In that case, the dressing cannot reduce the weight. A graph with this property is said to have Cheeger constant $h(G) \geq 1$.

To recap, we want our graph to have these properties:

- The graph has constant degree.
- All cycles on G are generated by a sparse set of constant-length cycles.
- Pairs of vertices that are in the support of the same Z-type stabilizer generator of the original code are connected in G by a path of edges with constant length.
- The Cheeger constant $h(G)$ is at least 1.

W-Y show that such a graph exists (we might need to add some additional "dummy vertices" beyond the support of L), and that it is not much larger than W, the weight of the target logical operator L. Specifically, the sum of the number of vertices + the number of edges is

$$|E| + |V| = O(W \log W),$$

and so in particular the number of ancilla qubits used is $O(W \log W)$ — this is the qubit overhead of the measurement protocol. The time overhead is $O(d)$, the number of times we need to repeat the gauge generator measurements to achieve fault tolerance.

This W-Y protocol, based on gauging, is analyzed graph-theoretically. What we'll consider next, the Xanadu protocol, is analyzed homologically and based on a different idea: the chain map and its mapping cone.

Algorithmic Fault Tolerance

Lecture 17 — 1 June 2026

References: arXiv: 2406.17653, arXiv: 2505.13587.

Review: PBC and PPM via surgery

We've discussed Pauli-based computation (PBC): universal quantum computation via $|T\rangle$ prep and Pauli-product measurement (PPM). And we've discussed how to do PPM via surgery on either many surface code blocks or blocks of high-rate codes (the former with 2D connectivity, the latter with nonlocal connectivity). In surgery, we introduce ancillas and deform the code. For fault tolerance, we consider qLDPC codes with distance d, and formulate the deformed code so that it, too, is qLDPC with $O(d)$ distance, and in addition the target Pauli product is in the stabilizer of the deformed code. Therefore we can perform PPM by measuring the low-weight checks of the deformed code, to determine the PPM as the parity of many check measurement outcomes.

We want this procedure to be fault tolerant (FT), meaning that $\Omega(d)$ faults (qubit errors and measurement errors combined) are necessary for the PPM to fail. To gain sufficient confidence in the check measurement outcomes, we repeat the measurement of the deformed code $O(d)$ times. Then the probability of error in the PPM is $\exp(-O(d))$.

Avoiding the $O(d)$ slowdown

Now we'll discuss a way to avoid this $O(d)$ blowup in the cycle time for FTQC. For this purpose, we'll make a distinction between the *online* part of the computation and the $|T\rangle$ *factory*. Preparing $|T\rangle$'s with $\exp(-O(d))$ probability of a logical error, we will need $O(d)$ syndrome extraction (SE) rounds. What we'll aim to speed up is the Clifford part of the computation, assuming that clean input $|T\rangle$ states are available. So our goal is really a space/time tradeoff. The Clifford computation is *fast* ($O(1)$ cycle time) and the $|T\rangle$ factory is *slow* ($O(d)$ preparation time). This means we'll need to *pipeline* the $|T\rangle$ prep, so there are always $|T\rangle$'s available when needed; that will increase the processor's spatial footprint. We'll describe how it works for the surface code, though similar ideas might apply to other CSS qLDPC codes that allow transversal Clifford group gates. In the surface code, $|T\rangle$ prep has spacetime cost $O(d^3) = O(d^2)$ qubits $\times$ $O(d)$ time. Conventionally we would say that a transversal CNOT gate also has spacetime cost $O(d^3)$, but we wish to reduce that to $O(d^2) = O(d^2)$ qubits $\times$ $O(1)$ time. This means we'll need fresh $|T\rangle$'s more often and must therefore enlarge the $|T\rangle$ factory.

For this to work, we'll need a paradigm shift concerning FTQC. We often formulate the goal of QEC to be protection of a quantum *state*, so that *any* decoded (Pauli) measurements sample from the same distribution of outcomes as if we had measured the ideal state instead. But that is more than we need to run one *particular* computation. In that case, all we care about is sampling from the ideal distribution for that one computation, which might work even if the protected state is far from ideal in other respects. This can make FTQC easier to achieve. This perspective is called "Algorithmic Fault Tolerance" (protecting a *computation* rather than a *state*). See:

Zhou et al. 2024	arXiv: 2406.17653
Cain et al. 2025	arXiv: 2505.13587

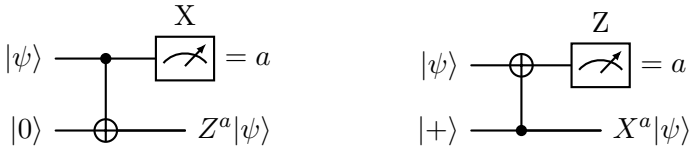
In the surface code with 2D connectivity we are stuck with the $O(d)$ slowdown. In lattice surgery in particular, only $O(1)$ measurement errors can cause a logical error if we measure the syndrome for only $O(1)$ rounds. But we'll allow nonlocal connectivity, as in neutral atom tweezer arrays, in which case gates that generate the Clifford group can be executed in depth 1.

Transversal gates; the model

How to perform logical H and S gates will be considered in a homework problem. What is especially valuable is the transversal CNOT gate (which works for any $k = 1$ CSS code): $X_C \to X_C X_T$ and $Z_T \to Z_C Z_T$, performed at the physical level on pairs drawn from two code blocks, realizes $\overline{X}_C \to \overline{X}_C \overline{X}_T$ and $\overline{Z}_T \to \overline{Z}_C \overline{Z}_T$ at the logical level. We can bring the two blocks into "contact," perform the transversal operation, and then separate them again.

Strictly speaking, we won't be in the PBC model here. The goal will *not* be to measure a high-weight logical Pauli product in a window with $O(1)$ temporal duration. We'll be doing Clifford + $|T\rangle$ prep. Though they are not really needed in the ideal computation, it will be helpful to introduce $|0\rangle$ and $|+\rangle$ prep as well as $|T\rangle$ prep. But the $|0\rangle$ and $|+\rangle$ prep (as well as X and Z measurement) will be part of our Clifford circuit — we do the $|0\rangle$ and $|+\rangle$ prep ourselves rather than taking it as given.

We might use 1-bit teleportation (1BT) to move logical information to a fresh block. Recall:

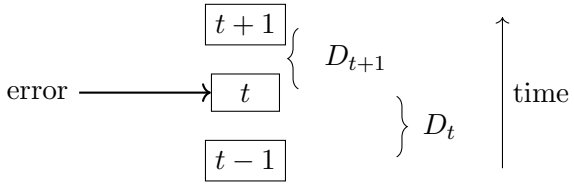


The measurement provides no info about $|\psi\rangle$. It is 50/50 random because the measured observable anticommutes with the stabilizer (XI with ZZ, ZI with XX).

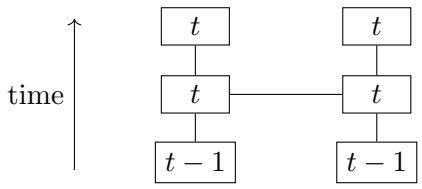
Decoding in spacetime

Recall that to protect a state in quantum memory, our decoder takes as input $O(d)$ rounds of SE. Why is that important?

There is a *decoding graph in spacetime*. Vertices are *detectors*. A detector fires if a particular syndrome bit changes from one time step to the next. The edges are errors (i.e. faults), which can be either qubit errors or measurement errors.

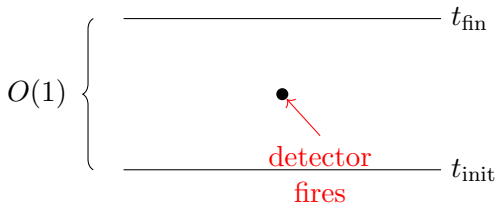


A measurement error in SE round t causes *two* detectors to fire. The syndrome bit flips twice: between rounds $t-1$ and t, then again between syndrome bits t and $t+1$.



A qubit error prior to SE round t also causes 2 detectors to fire — because it flips 2 syndrome bits in that one round.

We decode by performing minimum-weight perfect matching (MWPM) on this decoding graph.



A chain of $O(1)$ measurement errors can result in an isolated detector at an intermediate time, if we measure only $O(1)$ times. This detector gets paired with the boundary of the block or another detector far away, which can cause a logical error. To avoid this: $O(d)$ SE rounds, to ensure that $O(d)$ faults (measurement and qubit errors together) are needed for a logical error.

One way to say what we are trying to achieve is this: by focusing on protecting a *computation* instead of a *state*, we'll make such timelike chains of errors harmless. That way it will require a chain that has length $O(d)$ in the *spacelike* direction to cause failure. So the "fault distance" can be $O(d)$ even if vertical chains stretch across the temporal extent of our decoding window.

Transversal CNOTs complicate decoding

Our Clifford computation will include transversal CNOT gates that propagate errors from one block to another. This complicates decoding. In fact, if we perform $O(1)$ SE rounds between CNOT layers, a single fault could spread to $2^{O(d)}$ blocks if we waited for $O(d)$ rounds before decoding, which seems unmanageable. For this reason, the conventional view has been to allow $O(d)$ SE rounds between CNOT layers, so we collect reliable syndrome data before allowing the damage to spread.

Another way to say why CNOT gates are problematic is that, if we allow blocks to interact, the decoding graph becomes a *hypergraph*; now a single fault can cause 3 or 4 detectors to fire. (Edges are pairs of vertices; hyperedges are sets of vertices that can contain more than two elements.) Matching on a hypergraph (finding an adequate hypothesis for the faults that caused detectors to fire) is a much harder computational problem than matching on a graph.

For example, between SE rounds $t-1$ and t an X error occurs, which is then propagated by a CNOT from control to target block. In round t, 4 detectors are triggered in Z-type check operators, two in each block $\rightarrow$ a 4-site hyperedge.

A more subtle example involves a measurement error, which causes detections in two consecutive time steps. An X measurement error in step t causes detectors to fire twice in the control block:

$$Z_C^{(t-1)} \oplus Z_C^{(t)} \qquad \text{and} \qquad Z_C^{(t)} \oplus Z_C^{(t+1)}$$

are both nontrivial. In addition, the CNOT between steps t and $t+1$ modifies the detector on the target block, because $Z_T^{(t)} \to Z_C^{(t)} Z_T^{(t)}$, and hence the detector becomes

$$Z_C^{(t)}\, Z_T^{(t)}\, Z_T^{(t+1)},$$

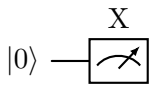
which also fires because of the $Z_C^{(t)}$ errors. This is a 3-site hyperedge.

We'll want to argue that, for the purpose of decoding a *computation*, the decoding hypergraph reduces to a graph, amenable to efficient MWPM decoding.

Which measurements need decoding?

Now imagine decoding a particular Clifford + T computation, with input $|T\rangle$ (given), $|+\rangle$, $|0\rangle$ (prepared by us), transversal Clifford gates, and X, Z measurements. We want our measurements to sample accurately from the distribution of outcomes in the ideal computation.

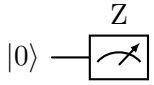
In fact, though, there are some measured Pauli operators we don't need to simulate. Here is an example:



The outcome is 50/50 random; we might as well flip a coin.

This happens because the "backpropagated" measured observable X anticommutes with the stabilizer Z of the initial state. (If, say, we are doing 1BT, we might care about the outcome for updating the Pauli frame. But that matters only if that Pauli frame update is needed to interpret a *later* measurement.)

Another example:



Here the outcome is deterministic, so we don't want to get it wrong. But suppose the measurement is *destructive*: all qubits measured in the Z basis; then those measurements are decoded to determine $\overline{Z}$. In that case, $|0\rangle$ does not need to be accurately encoded because Z errors don't matter. If we do the prep starting with $|0\rangle^{\otimes n}$ and then do $O(1)$ rounds of SE, we'll have lots of uncertainty about outcomes of $X^{\otimes 4}$ syndrome measurements, but that is okay because Z errors won't change how $\overline{Z}$ is decoded. The blocks are hot (might not even be entangled inside the block), but the many $X^{\otimes 4}$ detections don't cause trouble.

Now consider, for example, $\overline{X}$ measurement on some logical block in the Clifford circuit, and backpropagate through the circuit until obtaining a Pauli operator acting on all preparations. If we obtain X acting on any $|0\rangle$ or Z acting on any $|+\rangle$, then the backpropagated Pauli anticommutes with a stabilizer of the initial state, which means the outcome is 50/50 random. So we just assign a random bit to the outcome. No decoding is needed.

On the other hand, the backpropagated Pauli might have only Z or I acting on $|0\rangle$, or X or I acting on $|+\rangle$. In that case, though, we only need the $|0\rangle$ to give a reliable outcome for $\overline{Z}$ measurement. And for that purpose it is fine to prepare the state with only $O(1)$ rounds of syndrome extraction.

Now, if the outcome of the measurement of a particular block is 50/50 random, that does *not* mean we don't have to measure that block — even if its marginal distribution is uniform, it might have correlations with other blocks that we need to get right. So for each higher-weight

Pauli that is measured, we should consider whether or not, when backpropagated, it commutes or anticommutes with a stabilizer of $|0\rangle$ and $|+\rangle$ (that is, whether we find X acting on $|0\rangle$ or Z acting on $|+\rangle$). If it anticommutes, then no need to decode; just choose a random outcome. If it commutes, then states prepped in $O(1)$ rounds are okay. (The backpropagation may have X or Y or Z acting on $|T\rangle$, but that is okay because $|T\rangle$ has been prepared with reliable stabilizers.)

So our task is to decode the Pauli measurements we care about (called "reliable" in Cain 2025).

Reducing the hypergraph to a graph

We notice that, for this purpose, the decoding hypergraph becomes a graph to which MWPM can be applied. To build the decoding graph for a particular Pauli product, we choose a standard logical representative for the Pauli product (e.g., a canonical choice of string operator for each $\overline{X}$ or $\overline{Z}$ in a surface code block), and then backpropagate that operator through the Clifford circuit to find a Pauli $P(t)$ at each earlier time step. Now we consider the faults that can flip $P(t)$ (errors that anticommute with it) and include *only* those fault locations and the corresponding detectors triggered by those faults.

For example, consider the X error described earlier that was propagated by a CNOT from control to target, and that fired 4 Z-check detectors at time t. An X error might hit a Z logical string operator, flipping its value. If, acting on these two blocks, $P(t)$ is $Z_C \otimes I_T$ and the error flips the string, only the error on the control matters — hence two detectors. Same if $P(t) \sim I \otimes Z_T$. If the backpropagated $P(t)$ contains $Z_C \times Z_T$, then the error commutes with $P(t)$, and there are no detectors at all. In all three cases the detector becomes either an edge (2 vertices) or nothing.

Now consider the measurement error, which generated a 3-vertex hyperedge. Again we enumerate cases:

- $P(t+1) = Z_C \otimes Z_T \xrightarrow{\text{backprop}} P(t) = I \otimes Z_T$.
  No $(t-1, t)$ hyperedge, because the control block is not in $P(t)$.
  No $(t, t+1)$ hyperedge, because the XOR of the two detectors is 0.

- $P(t+1) = I_C \otimes Z_T \xrightarrow{\text{backprop}} P(t) = Z_C \otimes Z_T$.
  No $(t, t+1)$ detector on the control block — it is not in $P(t+1) \rightarrow$ 2 detectors $\rightarrow$ edge.

- $P(t+1) = Z_C \otimes I_T \xrightarrow{\text{backprop}} P(t) = Z_C \otimes I_T$.
  The target block is not in $P(t)$ or $P(t+1) \rightarrow$ 2 detectors $\rightarrow$ edge.

It always works, because Pauli products and errors propagate similarly.

Constant Depth and the Clifford Hierarchy

Lecture 18 — 3 June 2026

Reference: arXiv: 1206.1609.

Motivation

The Eastin–Knill theorem tells us that we can't do a universal set of logical gates on a QECC where each gate in the set is executed transversally. Hence we need other tools to achieve universal fault tolerance, like magic state distillation or code switching. But what principles dictate which gates have simple fault-tolerant realizations for a given code?

One useful guideline comes from the Bravyi–König theorem:

> Consider a family of topological stabilizer codes in D spatial dimensions (geometrically local checks and distance increasing without bound). Suppose a geometrically local, constant-depth unitary circuit executes a logical operation (preserves the code space). Then this logical unitary is contained in the Dth level of the Clifford hierarchy.

Constant depth is desirable in this code family because it limits the number of faults and the number of errors spread by the gates to a constant, below the growing distance of the code family.

This statement applies to local codes and circuits. But there is also a corresponding statement that holds for high-rate nonlocal codes, obtained from a product of D chain complexes.

The Clifford hierarchy

What is the Clifford hierarchy? Each level is a discrete set of unitaries, defined recursively up to a phase:

$$\mathcal{P}_0 = \{\mathbb{1}\}, \qquad \mathcal{P}_1 = \text{Pauli}, \qquad \mathcal{P}_2 = \text{Clifford},$$
$$\mathcal{P}_\ell \;=\; \big\{\, U \;:\; UPU^{-1} \in \mathcal{P}_{\ell-1} \;\; \forall\, P \in \mathcal{P}_1 \,\big\}.$$

This definition of $\mathcal{P}_\ell$ for $\ell \geq 3$ extends the notion of the Clifford group (unitaries whose action by conjugation takes Pauli to Pauli). $\mathcal{P}_2$ is a group because if U_1, U_2 map $\mathcal{P}_1$ to $\mathcal{P}_1$, then so does $U_1 U_2$. But $\mathcal{P}_\ell$ for $\ell \geq 3$ is *not* a group.

Examples of $\mathcal{P}_3$ gates are T, CCZ, and Tof. We exploit the $\mathcal{P}_3$ property in fault tolerance; it tells us, for example, that we can "clean up" a measurement-based T gadget with $|T\rangle$ state input using a (fault-tolerant) Clifford gate correction conditioned on the measurement outcome.

When we pull a Pauli frame update through a T gate, it becomes a Clifford frame update:

$$TX|\psi\rangle = \underbrace{\left(TXT^{-1}\right)}_{\text{Clifford}} T|\psi\rangle = \tfrac{1}{\sqrt{2}}\,(X+Y)\,T|\psi\rangle.$$

Examples of higher-level gates:

$$\mathcal{P}_m \text{ gates:} \quad \text{diag}\big(1,\, e^{i\pi/2^{m-1}}\big),$$
$$m=1:\; Z, \qquad m=2:\; S, \qquad m=3:\; T, \qquad m=4:\; \sqrt{T}.$$

Going to the next higher level $\sim$ taking a square root.

Key idea: the Cleaning Lemma (for stabilizer codes)

Recall — a set of qubits is *correctable* if erasure of the set can be corrected (size of set $<$ distance of code is sufficient but not necessary for correctability).

Suppose a set R is correctable and P is a logical Pauli. Then there is a stabilizer element S such that

$$P' = PS = \mathbb{1}_R \otimes \widetilde{P}'_{R^c}.$$

P' is unsupported on R (we can *clean* P on R by applying the code's check operators).

Note that if a logical unitary is supported on a correctable set, then it must be the logical identity up to a phase. (Else: we could apply U and erase R, an uncorrectable erasure error.)

From the perspective of the cleaning lemma: suppose

$$U = U_R \otimes \mathbb{1}_{R^c} \qquad \text{(supported on } R\text{)},$$

and let P be *any* logical Pauli, which can be cleaned on R:

$$P' = SP = \mathbb{1}_R \otimes \widetilde{P}'_{R^c}.$$

Therefore

$$\begin{aligned} UP'U^{-1} &= \big(U_R \otimes \mathbb{1}_{R^c}\big)\big(\mathbb{1}_R \otimes \widetilde{P}'_{R^c}\big)\big(U_R^{-1} \otimes \mathbb{1}_{R^c}\big) \\ &= \big(\mathbb{1}_R \otimes \widetilde{P}'_{R^c}\big) = P'. \end{aligned}$$

But a unitary that commutes with any logical Pauli must be a multiple of the identity,

$$U = \mathbb{1} \times \text{phase}$$

(because Paulis are a complete operator basis).

Transversal unitaries and the hierarchy

Using the cleaning lemma, we can show the following for *any* stabilizer code. Suppose the n qubits in the code block decompose into m nonoverlapping correctable sets, plus one more set that need not be correctable,

$$\text{code block } = \bigcup_{i=1}^{m} R_i \ \cup \ \widetilde{R},$$

and let unitary U act transversally on the correctable sets:

$$U = \bigotimes_{i=1}^{m} U_i \otimes \mathbb{1}_{\widetilde{R}}.$$

Then $U \in \mathcal{P}_{m-1}$ (is in level $m-1$ of the hierarchy). Here we make *no* further assumptions about code distance or geometric locality of the checks.

The proof is by induction. The $m = 1$ case was described above. Now consider $m = 2$: the code block is $R_1 \cup R_2 \cup \widetilde{R}$ where R_1 and R_2 are both correctable, and $U = U_1 \otimes U_2 \otimes \mathbb{1}_{\widetilde{R}}$.

Then consider any Pauli operator and clean it on R_2, obtaining $P' = P_1 \otimes \mathbb{1}_2 \otimes \widetilde{P}$. We find

$$\begin{aligned} UP'U^{-1}P'^{-1} &= \big(U_1 \otimes U_2 \otimes \mathbb{1}\big)\big(P_1 \otimes \mathbb{1} \otimes \widetilde{P}\big)\big(U_1^{-1} \otimes U_2^{-1} \otimes \mathbb{1}\big)\big(P_1^{-1} \otimes \mathbb{1} \otimes \widetilde{P}^{-1}\big) \\ &= U_1 P_1 U_1^{-1} P_1^{-1} \otimes \mathbb{1}_2 \otimes \mathbb{1}. \end{aligned}$$

This is a unitary supported on the correctable set R_1. That is the $m = 1$ case, for which we know the unitary must be a multiple of the identity:

$$\implies \quad UP'U^{-1} = \text{phase} \times P'.$$

Thus U commutes with every logical Pauli operator, up to a phase. The only such unitary is a Pauli; hence $U \in \mathcal{P}_1$. If we expand U in the Pauli basis,

$$U = \sum_a \alpha_a P_a, \qquad \text{then} \quad P'^{-1} U P' = \eta\, U \quad (\eta = \text{phase})$$

says that $P'^{-1} P_a P' = \eta\, P_a$ for all a such that $\alpha_a \neq 0$. Either all such P_a commute with P' or all anticommute. If there are P_a and P_b that (say) both commute with P', then there is another Pauli P'' that commutes with P_a and anticommutes with P_b; hence P'' does not commute with U up to a phase. We conclude that only one α_a is nonzero.

At the next level, the code is $R_1 \cup R_2 \cup R_3 \cup \widetilde{R}$ and $U = U_1 \otimes U_2 \otimes U_3 \otimes \mathbb{1}$, and the Pauli after cleaning is $P' = P_1 \otimes P_2 \otimes \mathbb{1} \otimes \mathbb{1} \implies UP'U^{-1}P'^{-1}$ is supported on $R_1 \cup R_2$ and hence is contained in $\mathcal{P}_1 \implies$

$$UP'U^{-1} = P' \times \text{Pauli} = \text{Pauli}.$$

This means $U \in \mathcal{P}_2$.

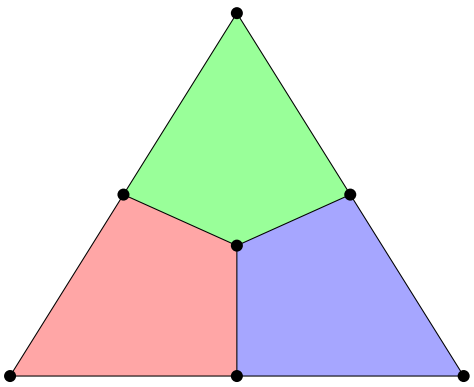
Likewise, after cleaning the Pauli on one of the m regions,

$$UP'U^{-1} \text{ supported on } m-1 \text{ regions}$$

and hence is $\mathcal{P}_{m-1}$.

Example: the $[[7, 1, 3]]$ Steane code (a color code)

For each 2-cell, checks include $XXXX$ and $ZZZZ$ acting on the four vertices.



vertices = qubits
2-cells = checks
faces are 3-colorable
and vertices are
trivalent ⇒ faces meet
on edges (or not at all)

We can choose logical X and Z to be supported on, e.g., the 3 bottom vertices (commutes with all checks). This means that the top 4 vertices must be a correctable set: no possible nontrivial logical operator is supported there, even though the size of the set (4) is larger than the distance ($d = 3$). We can decompose the 7 qubits into correctable sets

$$4 + 2 + 1 \qquad (2 \text{ and } 1 \text{ are } < d).$$

Therefore, a transversal unitary must be Clifford. (And in fact the entire Clifford group is transversal.)

In the 3D $[[15, 1, 3]]$ color code, the lattice is 4-valent and 3-cells are 4-colorable. X checks on 3-cells and Z checks on faces. Now the decomposition into correctable regions is

$$8 + 4 + 2 + 1 \quad \Longrightarrow \quad \text{transversal gates are } \mathcal{P}_3.$$

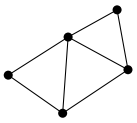
In fact, the code admits a transversal T (and then Eastin–Knill implies some Clifford gate is not transversal; there is not a transversal H. In fact Z logicals are strings and X logicals are membranes).

Higher-dimension Reed–Muller codes are $[[2^m - 1,\, 1,\, 3]]$. Decomposition into m regions is

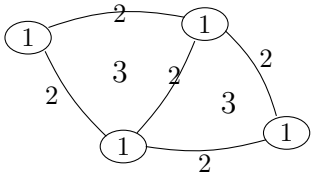
$$2^{m-1} + 2^{m-2} + \cdots + 2 + 1 \quad \Longrightarrow \quad \mathcal{P}_{m-1} \text{ gates can be transversal, like } \operatorname{diag}\big(1,\, e^{i\pi/2^{m-2}}\big).$$

Local codes with growing distance

Now let's consider local codes with growing distance, e.g. in $D = 2$ dimensions. We triangulate the plane (or a non-Euclidean surface), and then "fatten" the vertices, edges, and faces:



and then "fatten" the vertices, edges, and faces



The checks have finite range and the distance is large. Choose connected components of regions $1, 2, 3$ to be $<$ distance (hence correctable) and separation between components $>$ range.

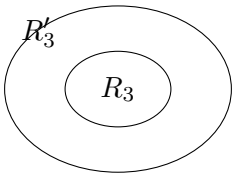
Recall the lemma: a union of separated correctable sets is correctable. We have decomposed the surface into 3 correctable sets. Hence a unitary transversal w.r.t. this decomposition must be in $\mathcal{P}_2$.

Similarly, in D dimensions, a fattened triangulation decomposes space into $D+1$ correctable regions. Therefore a transversal unitary is in $\mathcal{P}_D$.

Extension to constant-depth circuits

How does the argument extend to constant-depth unitary circuits? Consider the $D = 2$ case.

If gates have range r and the circuit for U has depth h, then the support of UPU^{-1} reaches beyond the support of P by $\leq rh$ in all directions. Suppose P is Pauli. If the distance is large enough, the connected components of R_3 can be chosen to be sufficiently separated that regions extending beyond R_3 by $> rh$ are still correctable and separated, so we can clean P on the larger region R_3', ensuring that $UP'U^{-1}$ is unsupported on R_3. Hence $UP'U^{-1}P'^{-1}$ is supported on the union of two correctable sets $R_1 \cup R_2$. This is what we need for the inductive step.



The argument by Bravyi & König is a little more sophisticated, and applies even in the case of a *code deformation* — where U does not preserve the code space but instead maps it to a different topological stabilizer code.