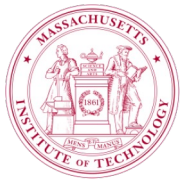
Modern Surgery Techniques

Sunny Zhiyang He, November 2026



Outline

I. QLDPC Computation and Surgery

II. Code surgery via graphs and chain maps

III. Extractor Architectures

IV. Modern Resource Estimations

V. Further topics in surgery

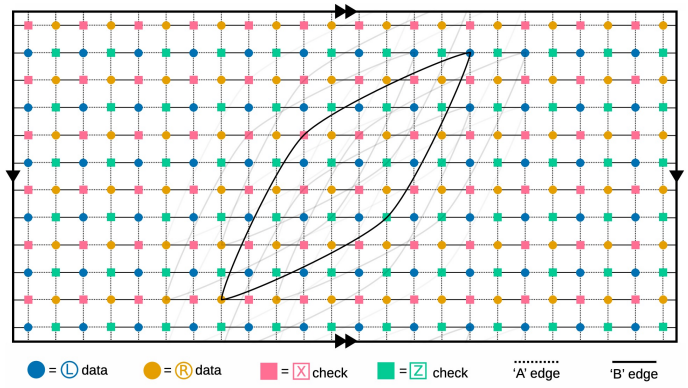
QLDPC Computation and Surgery

Memory with QLDPC codes

Quantum low-density parity-check (LDPC) codes have high encoding rate, at the cost of higher connectivity requirement. Examples:

- Bivariate Bicycle code [144, 12, 12] [1]
- Lifted/balanced product code [544, 80, ≤ 12] [2]
- APM Codes [1152, 580, ≤ 12] [3]

Surface code: [144, 1, 12]. Space overhead of encoding can be reduced by more than 100x!



IBM's bivariate bicycle code

$$H_X = \begin{pmatrix} F_0 & F_1 & F_2 & F_3 & F_4 & F_5 & G_0 & G_1 & G_2 & G_3 & G_4 & G_5 \\ F_5 & F_0 & F_1 & F_2 & F_3 & F_4 & G_5 & G_0 & G_1 & G_2 & G_3 & G_4 \\ F_4 & F_5 & F_0 & F_1 & F_2 & F_3 & G_4 & G_5 & G_0 & G_1 & G_2 & G_3 \end{pmatrix}$$

$$H_Z = \begin{pmatrix} G_0^T & G_5^T & G_4^T & G_3^T & G_2^T & G_1^T & F_0^T & F_5^T & F_4^T & F_3^T & F_2^T & F_1^T \\ G_1^T & G_0^T & G_5^T & G_4^T & G_3^T & G_2^T & F_1^T & F_0^T & F_5^T & F_4^T & F_3^T & F_2^T \\ G_2^T & G_1^T & G_0^T & G_5^T & G_4^T & G_3^T & F_2^T & F_1^T & F_0^T & F_5^T & F_4^T & F_3^T \end{pmatrix}$$

APM codes of rate ½

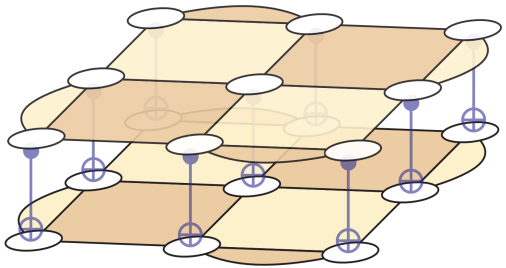
[1] [IBM 2308.07915], [2] [Xu et al. 2308.08648], [3] [Zhao et al. 2604.16209], [Kasai 2601.08824]

A Dichotomy in QLDPC Computation

For LDPC codes, the computational methods we know falls into two categories:

Parallel and structured logical control

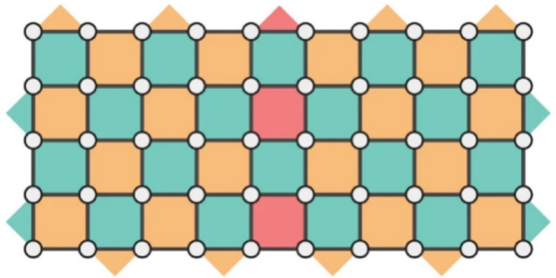
- ➢ Low-depth unitaries, such as transversal CNOT
- ➢ Batched syndrome extraction or code-switching
- ➢ Some forms of parallel surgery
- \+ Parallel, usually have O(1) time overhead
- \- Logical control is constrained and often limited
- \- Usually needs qubit reconfiguration



Transversal CNOT on surface code[1]

Sequential and complex control

- ➢ Pauli measurements with CAT states
- ➢ Many forms of code surgery
- ➢ Very few addressable and transversal gates
- \+ Complex logical control
- \- Often uses higher space-time overhead
- \- Logical control is sequential in many cases



Lattice surgery[2]

[1] Figure from [Sahay et al. 2408.01393]. [2] [Horsman et al. 1111.4022], figure from [Erhard et al. 2006.03071]

The Case for Code Surgery

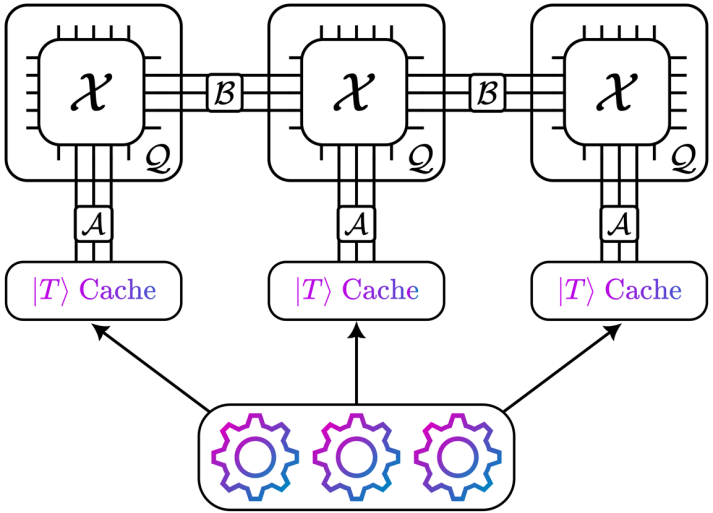
LDPC surgery [1-6] is a new technique that has advanced rapidly.

Modern surgery techniques has many desirable properties:

- Provably fault-tolerant and low-overhead [3,4]
- Complete logical control (often sequential) [6]
- Effective starting at small scale: <400 physical qubits enable control of 11 logical qubits at distance 12 [2,7]
- Implementable on different qubit platforms, including fixed-connectivity hardware [4,6]

Surgery and low-depth unitaries are similar to CPU and GPU: sequential and complex control, vs. parallel and structured control.

- In early FTQC, as we want to implement specific tasks ASAP, surgery is a very natural choice.



Extractor architecture for LDPC surgery-based FTQC[6]

[1] [Cohen et al. 2110.10794], [2] [Cross et al. 2407.18393], [3] [Williamson & Yoder 2410.02213], [4] [Ide et al. 2410.02753], [5] [Swaroop et al. 2410.03628], [6] [He et al. 2503.10390]. [7] [IBM 2506.03094]

Code surgery via graphs and chain maps

Universal computation via logical measurements

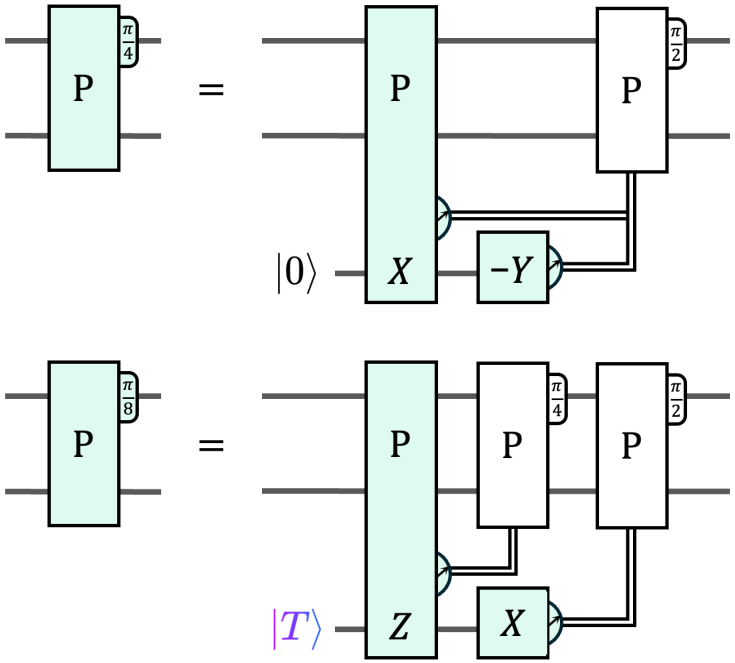
A Clifford + T circuit can be compiled into two primitives:

1. measurement of Pauli observables (Pauli-product measurements, PPMs)
2. magic states production

This is called Pauli-based computation.

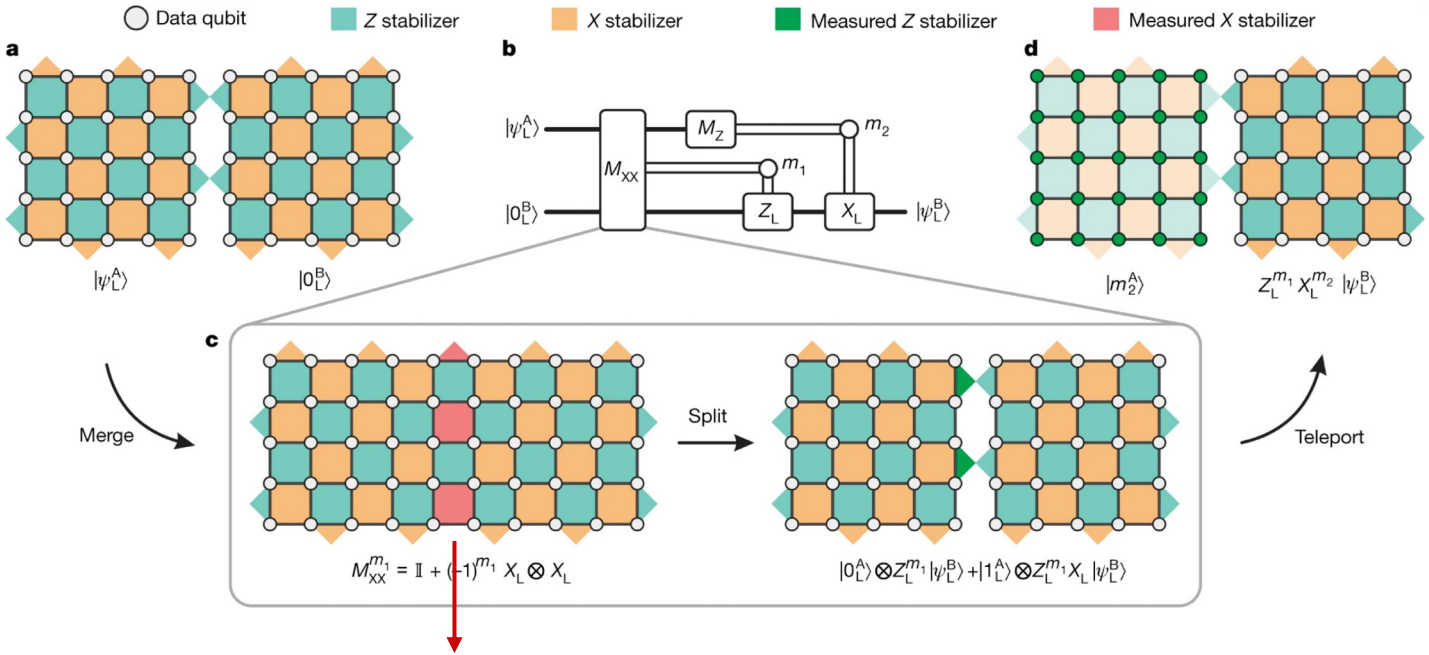
Code surgery is a way to measure logical Pauli observables fault-tolerantly.

- Starting at a quantum code (a stabilizer group), we measure a new stabilizer group, where the (large) logical Pauli become a product of (small) new stabilizers.
- Weight reduction → fault-tolerance



Surface Code Lattice Surgery

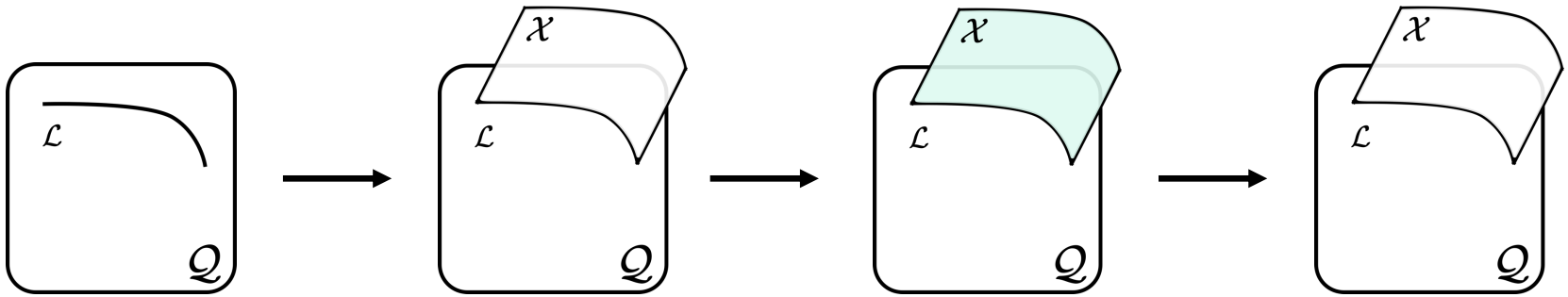
Logical measurements on surface codes: lattice surgery, [Horsman et al, 1111.4022].



Product of red X-checks = $X_L \otimes X_L$ – obtain logical measurement result by measuring new stabilizers.

Generalized Code Surgery

High level description: for a quantum code $\mathcal{Q}$ and a logical operator $\mathcal{L}$, define a new stabilizer group supported on $\mathcal{Q}$ and an ancilla system $\mathcal{X}$. Measure the new stabilizer group to measure $\mathcal{L}$.



(a) Start: Code $\mathcal{Q}$ and operator $\mathcal{L}$.

(b) Init: Initialize ancilla qubits in product state.

(c) Merge: Code deformation by measuring new stabilizers

(d) Split: Measure out ancilla and return to $\mathcal{Q}$.

Usual protocol: O(d) rounds in base code, O(d) rounds in merged code, then O(d) rounds in base code. Continuous chain of LDPC detectors ensure fault-tolerance.

Code surgery via graphs and chain maps

Graph formalism

Scalable Tanner Graphs

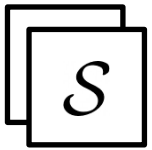
Scalable Tanner Graphs



Code $\mathcal{Q}$

Physical qubits of the code $\mathcal{Q}$

Scalable Tanner Graphs



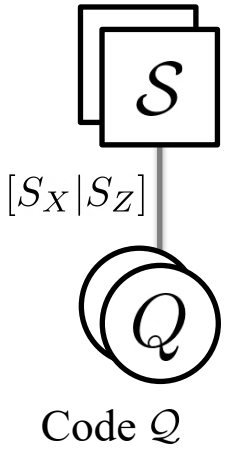
Stabilizers of the code $\mathcal{Q}$



Physical qubits of the code $\mathcal{Q}$

Code $\mathcal{Q}$

Scalable Tanner Graphs

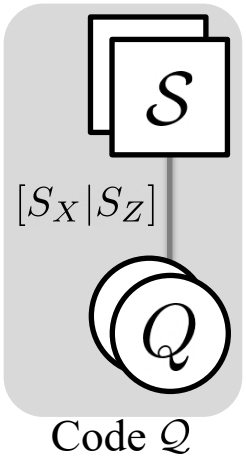


Stabilizers of the code $\mathcal{Q}$

Symplectic check matrix

Physical qubits of the code $\mathcal{Q}$

Scalable Tanner Graphs

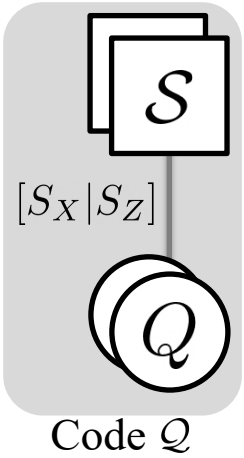


Stabilizers of the code $\mathcal{Q}$

Symplectic check matrix

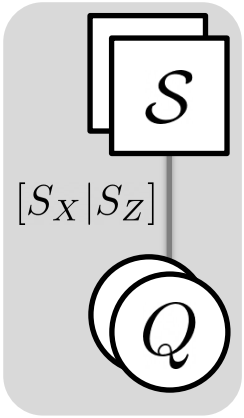
Physical qubits of the code $\mathcal{Q}$

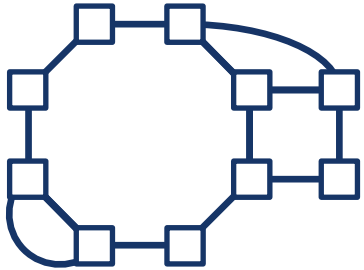
Building Ancilla Systems from (Hyper)Graphs



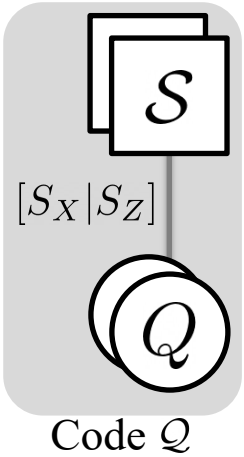
Building Ancilla Systems from (Hyper)Graphs

Let G = (V, E) be a (hyper)graph.



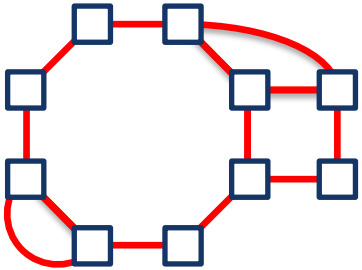


Building Ancilla Systems from (Hyper)Graphs



Code $\mathcal{Q}$

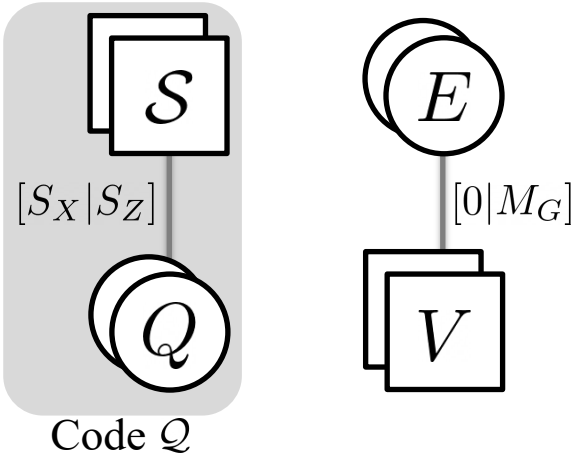


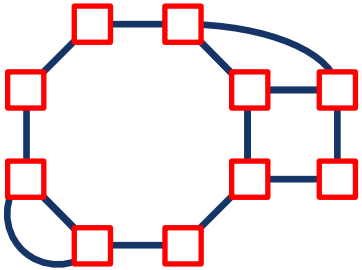


Let G = (V, E) be a (hyper)graph.

1. For every (hyper)edge in E, create an ancilla edge qubit.

Building Ancilla Systems from (Hyper)Graphs



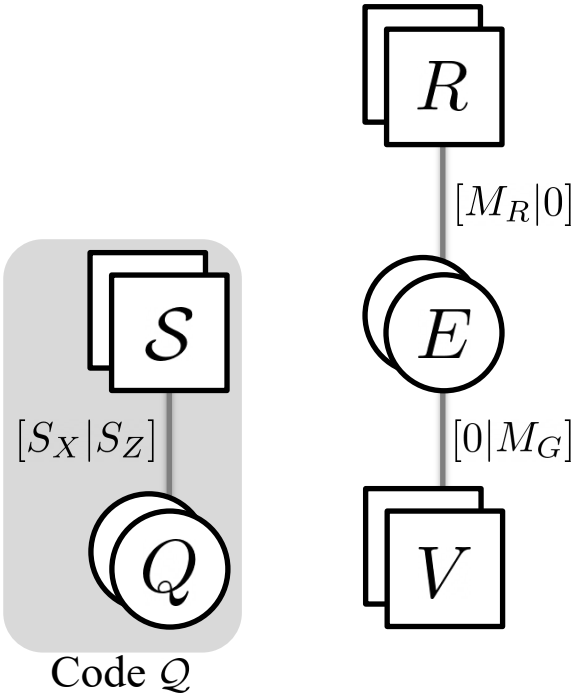


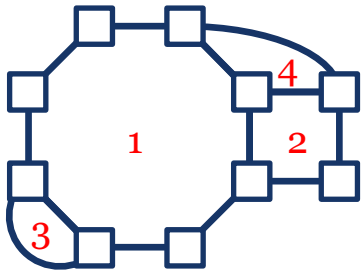
Let G = (V, E) be a (hyper)graph.

1. For every (hyper)edge in E, create an ancilla edge qubit.

2. For every vertex in V, create an vertex check, which act on adjacent edge qubits by Pauli Z.

Building Ancilla Systems from (Hyper)Graphs

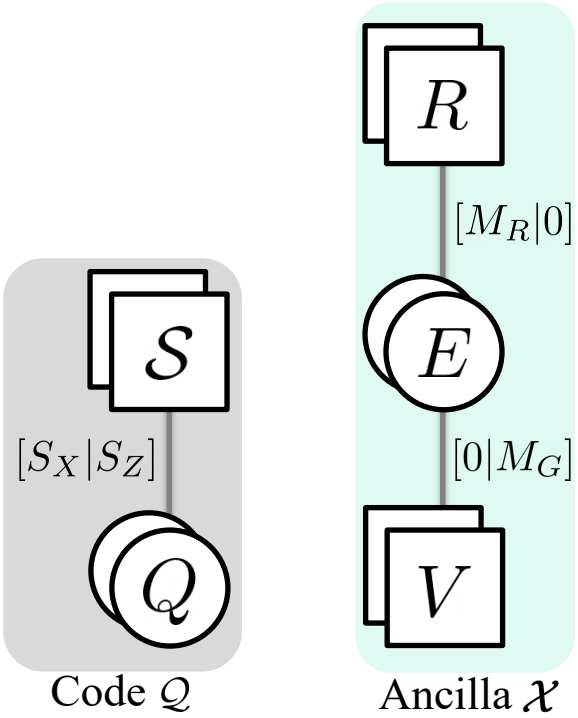


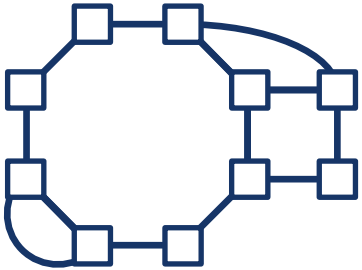


Let G = (V, E) be a (hyper)graph.

1. For every (hyper)edge in E, create an ancilla edge qubit.
2. For every vertex in V, create a vertex check, which act on adjacent edge qubits by Pauli Z.
3. Pick a cycle basis R of G. For every cycle C in R, create a cycle check, which act on edges in C by Pauli X.

Building Ancilla Systems from (Hyper)Graphs

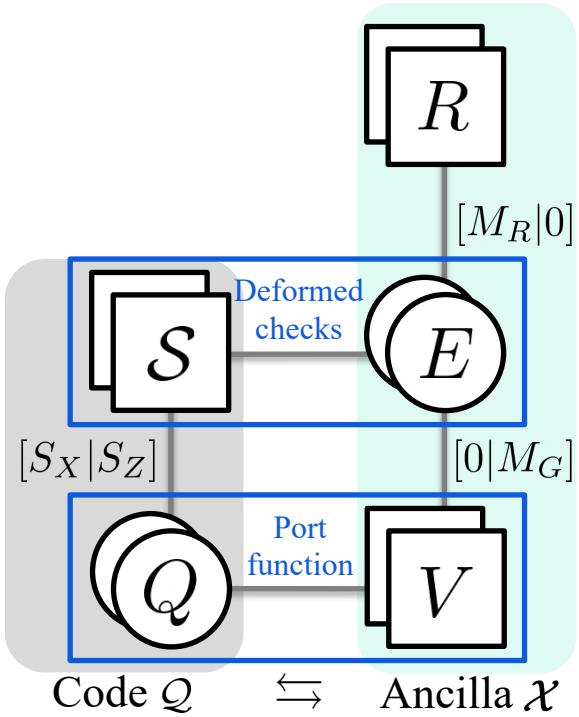




Let G = (V, E) be a (hyper)graph.

1. For every (hyper)edge in E, create an ancilla edge qubit.
2. For every vertex in V, create a vertex check, which act on adjacent edge qubits by Pauli Z.
3. Pick a cycle basis R of G. For every cycle C in R, create a cycle check, which act on edges in C by Pauli X.

Extracting Logical Pauli Observables

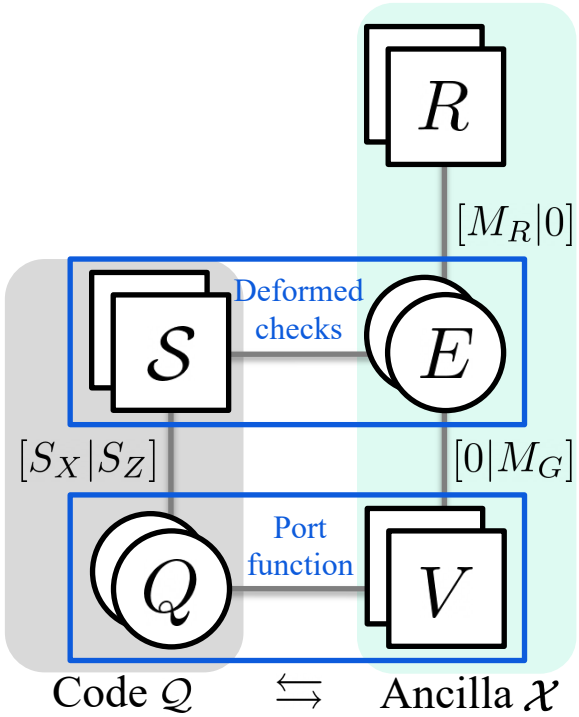


To measure a logical operator $\mathcal{L}$, we construct:

1. Measurement (hyper)graph G;
2. Port function: Pauli actions of vertex checks V on qubits of $\mathcal{Q}$; Defined so that product of vertex checks in a connected component equals $\mathcal{L}$.
3. Deformed checks: Pauli actions of stabilizers S on ancilla edge qubits E.

These data defines a subsystem code, which we call the measurement code $\mathcal{Q}_{\mathcal{L}}$.

Graph Desiderata



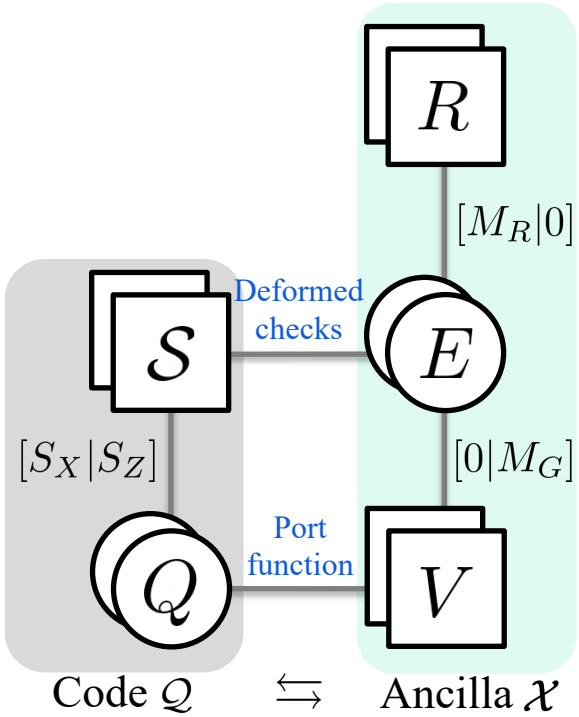
We want LDPC and code distance to ensure fault-tolerance. The following graph conditions are sufficient:

1. G has constant maximum degree
2. G has a sparse cycle basis
3. The port function is sparse
4. The deformed checks are low-weight
5. G is expanding

This formalism is developed gradually in [1-5]. See Section 3 of [5] for a short technical review.

[1] [Cross et al. 2407.18393], [2] [Williamson & Yoder 2410.02213], [3] [Ide et al. 2410.02753], [4] [Swaroop et al. 2410.03628], [5] [He et al. 2503.10390].

Fault Tolerance



In phenomenological noise, we need to protect again measurement (time-like) error and data (space-like) error.

Time-like distance:

- d rounds of measurement in deformed code

Space-like distance:

- The graph is expanding, and
- The ancilla system introduce no new logical qubits (measure cycle checks = gauge-fix new logicals)

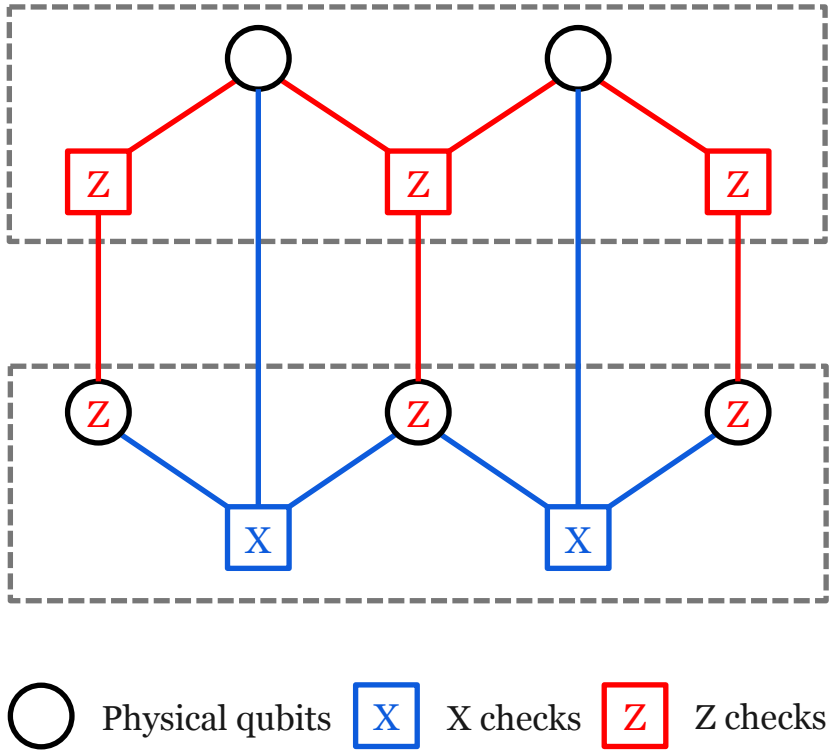
Example

Ancilla system

Port function (1-to-1)

Check deformation:
done with ‘path matching’

Local code structure, with
Z logical operator to be measured



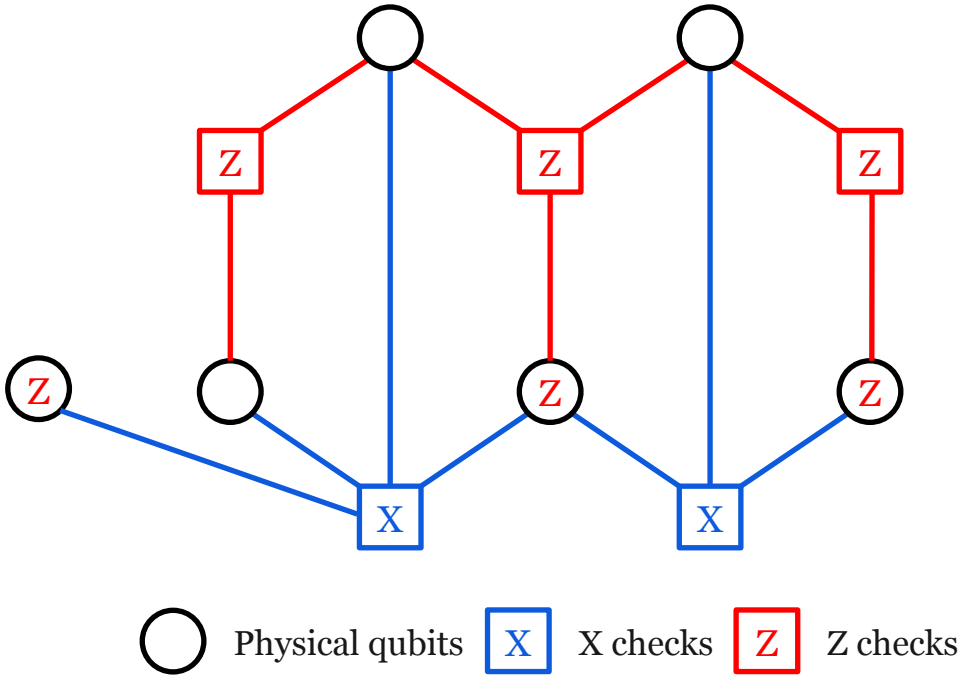
What can go wrong?

Consider another logical partly supported on the physical qubits.

Its distance can be reduced in two ways:

- Cleaning: multiply by new stabilizers
- Dressing: multiply by new logicals

Let's clean with two highlighted checks



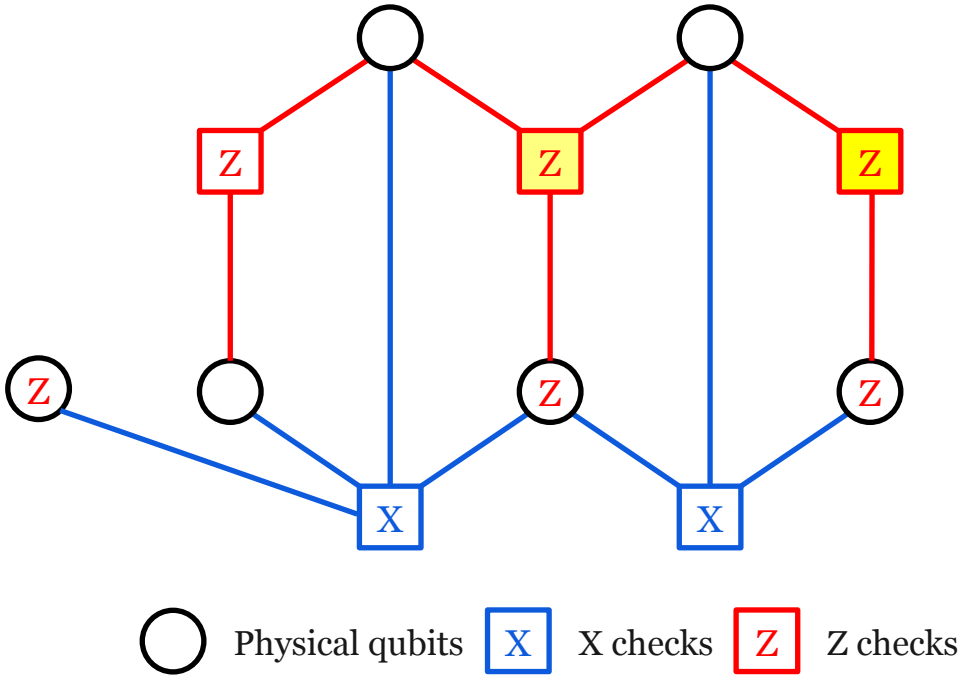
What can go wrong?

Consider another logical partly supported on the physical qubits.

Its distance can be reduced in two ways:

- Cleaning: multiply by new stabilizers
- Dressing: multiply by new logicals

Let's clean with two highlighted checks



What can go wrong?

Consider another logical partly supported on the physical qubits.

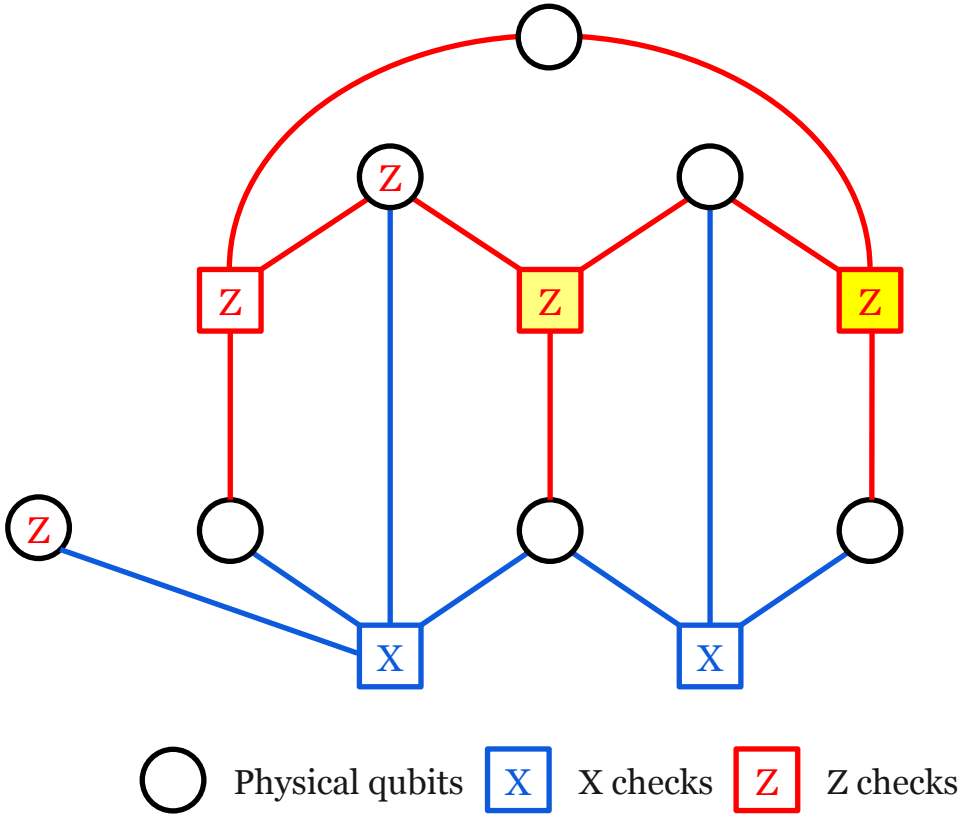
Its distance can be reduced in two ways:

- Cleaning: multiply by new stabilizers
- Dressing: multiply by new logicals

Let's clean with two highlighted checks

- Operator weight reduced 3 → 2
- Not fault-tolerant!

Solution: Expansion.



What can go wrong?

Consider another logical partly supported on the physical qubits.

Its distance can be reduced in two ways:

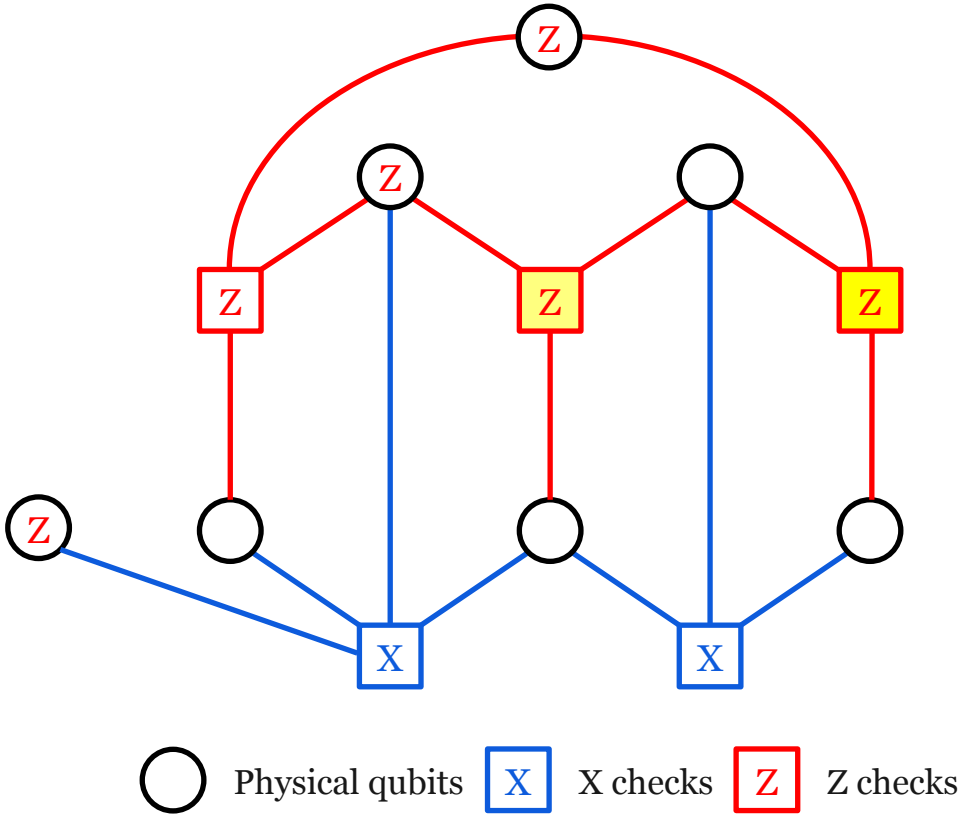
- Cleaning: multiply by new stabilizers
- Dressing: multiply by new logicals

Let's clean with two highlighted checks

- Operator weight reduced 3 → 2
- Not fault-tolerant!

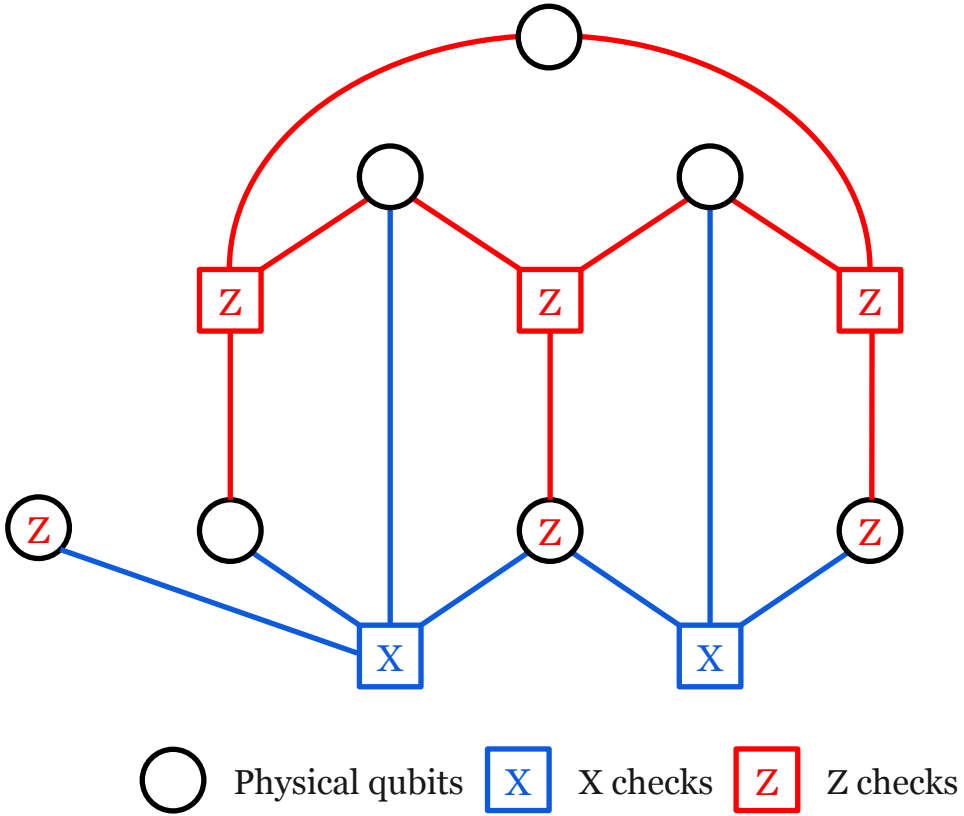
Solution: Expansion.

- Every two vertices now have two outgoing edges.
- Operator weight preserved by cleaning.



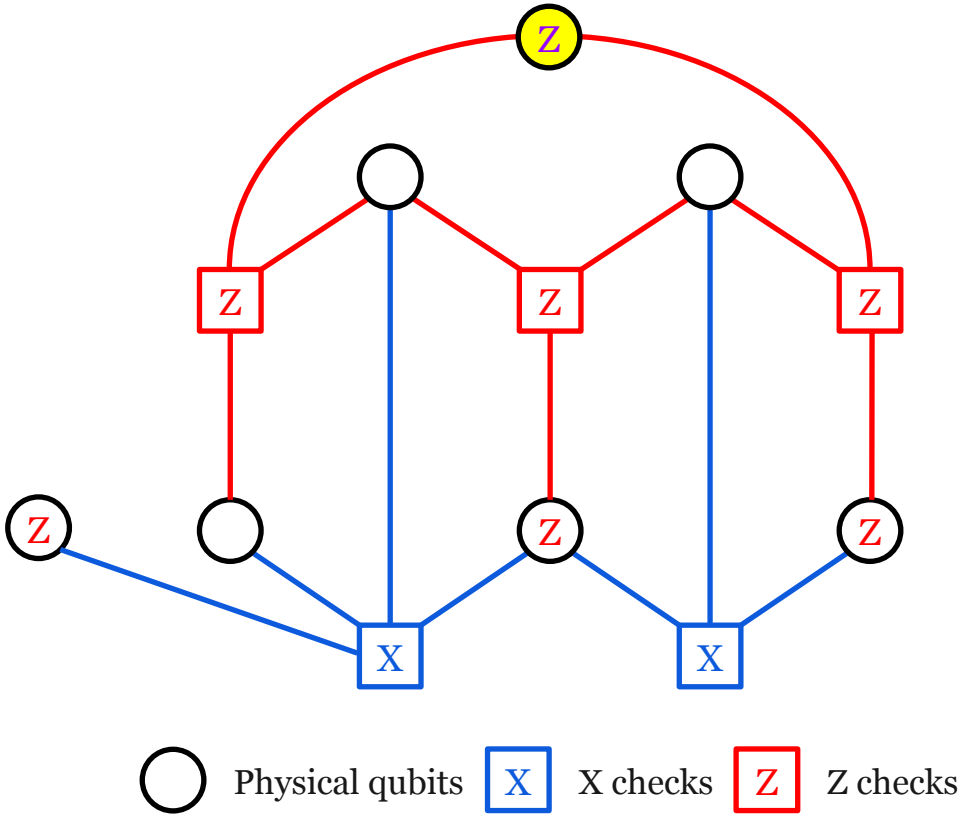
What can go wrong (again)?

Now consider dressing. When we introduce more ancilla qubits than ancilla checks, we have new ‘gauge’ operators.



What can go wrong (again)?

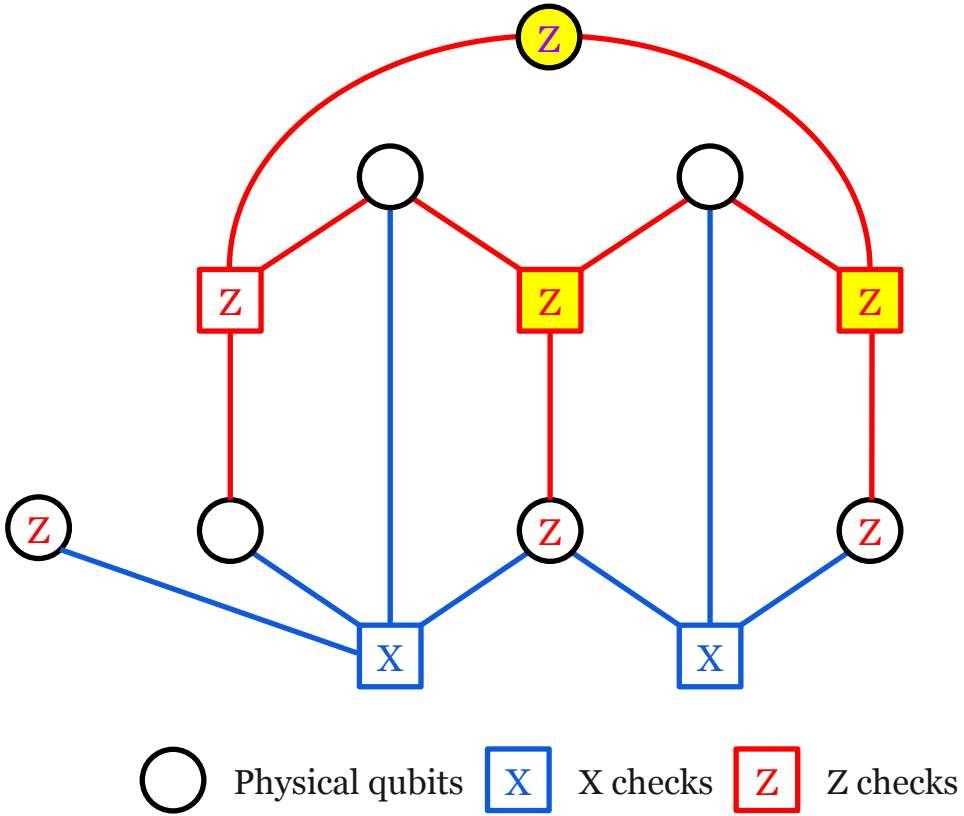
Now consider dressing. When we introduce more ancilla qubits than ancilla checks, we have new ‘gauge’ operators.



What can go wrong (again)?

Now consider dressing. When we introduce more ancilla qubits than ancilla checks, we have new ‘gauge’ operators.

Multiply by the highlighted checks & logicals:



What can go wrong (again)?

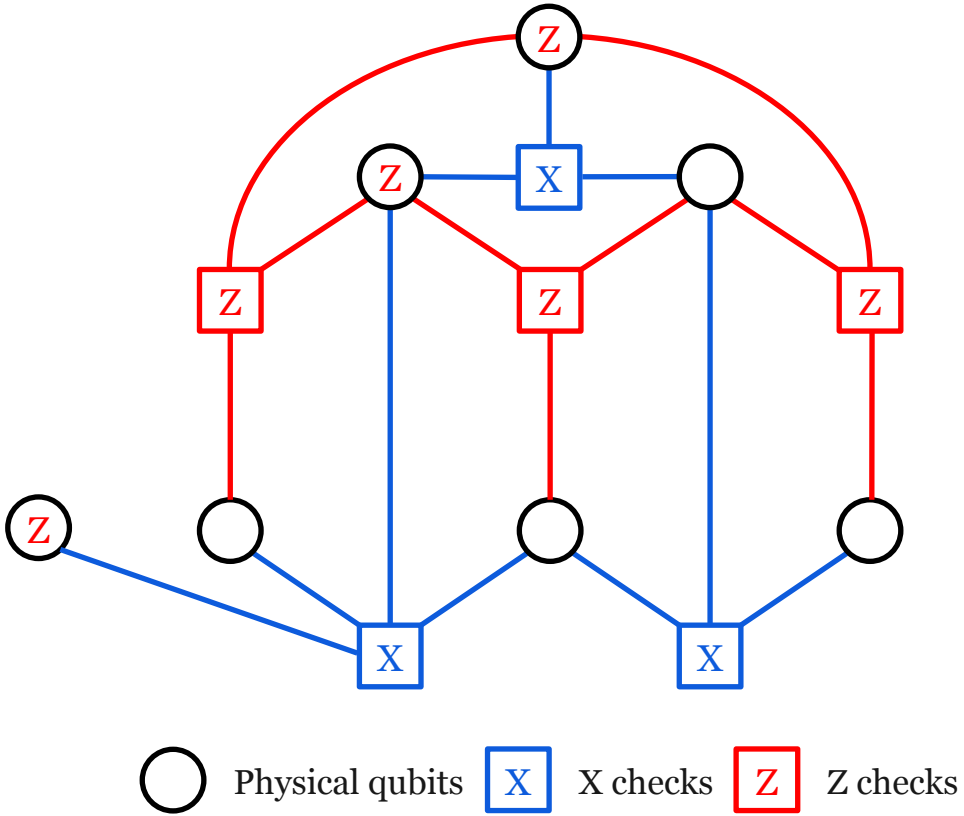
Now consider dressing. When we introduce more ancilla qubits than ancilla checks, we have new ‘gauge’ operators.

Multiply by the highlighted checks & logicals:

- Operator weight reduced 3 → 2 again!

Solution: measure all cycle as checks in opposite basis.

- No new logical qubits → space distance preserved.

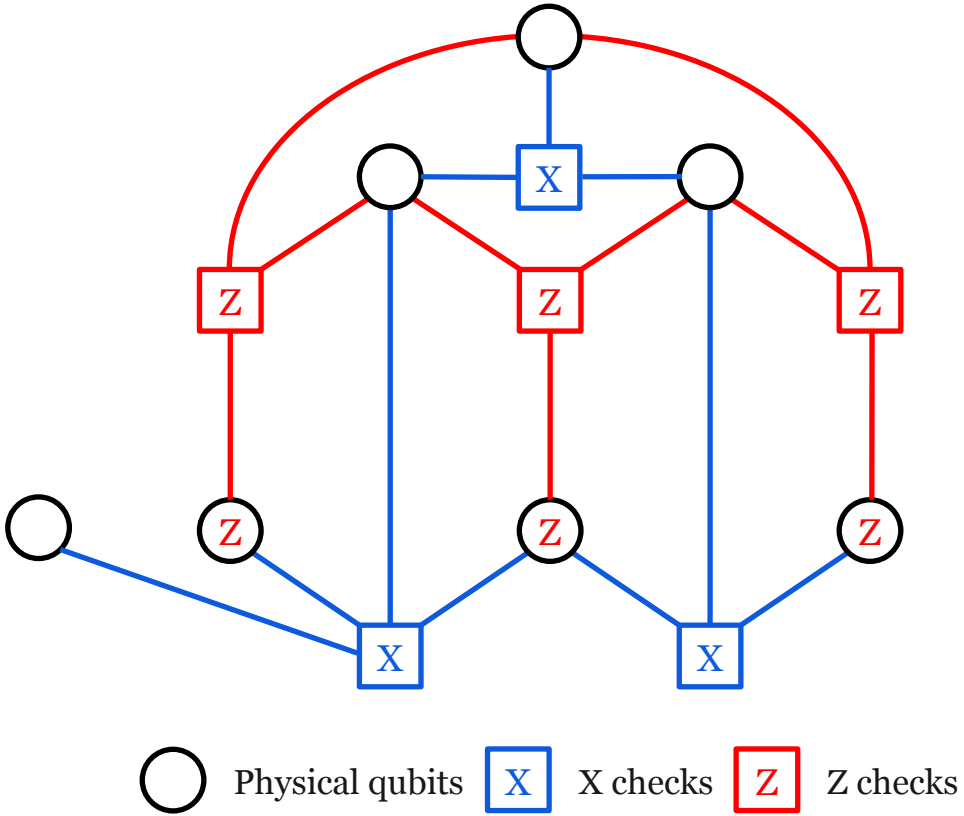


Summary: how to build graphs for surgery

One common way to build a surgery graph for a logical operator L:

1. Create one vertex for each physical qubit
2. Port function: 1-to-1
3. Add edges (‘path matchings’) to ensure commutation
4. Add edges to boost expansion
5. Sparsify the cycle basis (‘decongestion’ + ‘thickening’), measure all cycles

Satisfy all our desiderata!



Backup, in case you ask me about sparsifying cycles

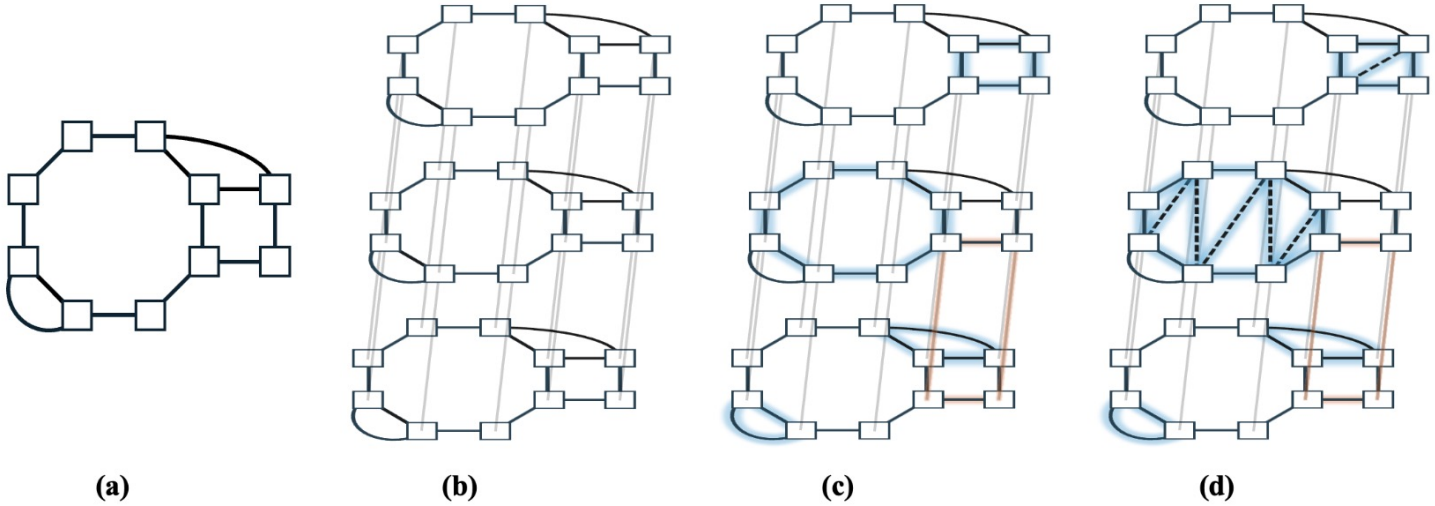


FIG. 4. Depicting of thickening, decongestion, and cellulation. **(a)** A generic graph G_1. **(b)** G_1 thickened by a line graph J_3, $G = G_1 \square J_3$. **(c)** A cycle basis in G, where the blue cycles are a cycle basis of G_1 spread into distinct levels and therefore do not overlap, and the red cycle is one of the new cycles created by thickening (Fact 17). **(d)** Cellulating cycles into triangles. We did not cellulate the red cycle(s) as they always have weight 4.

Code surgery via graphs and chain maps

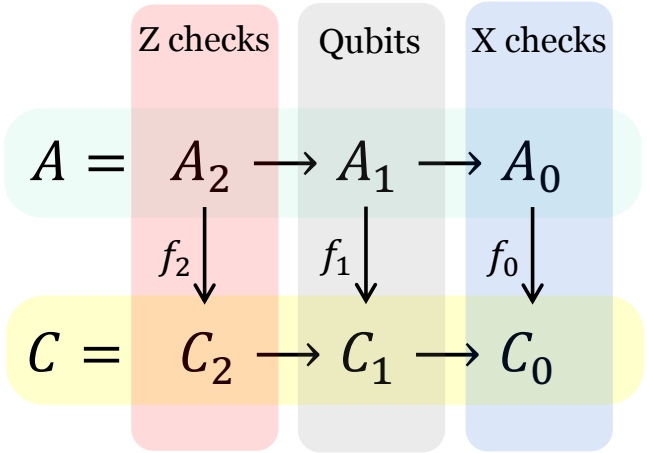
Chain map formalism

Chain Complex and Maps

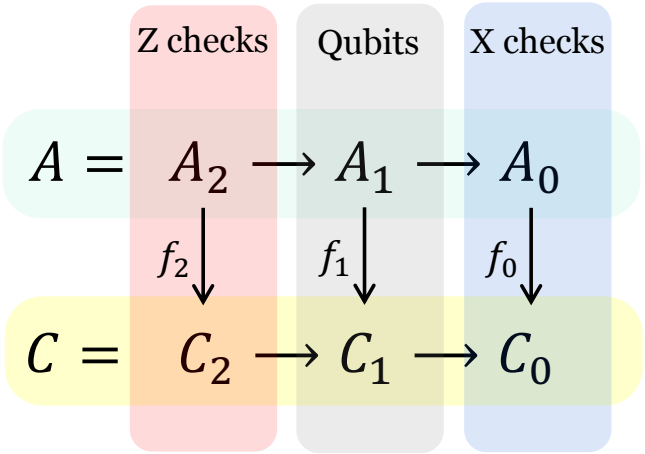
A CSS code is a chain complex. We can consider an ancilla code/complex, and maps between two complexs.

The diagram commutes → valid chain map.

- Can be used to describe transversal CNOTs, homomorphic measurements (Steane EC).
- These maps can be trivial (0).

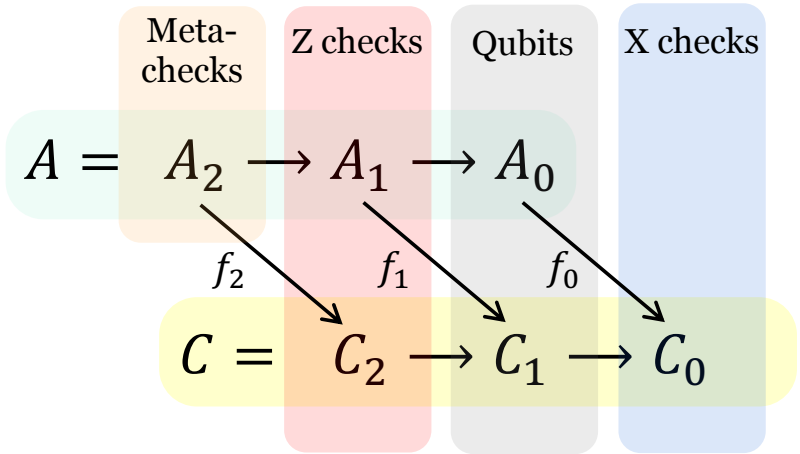


Mapping Cone



Chain map

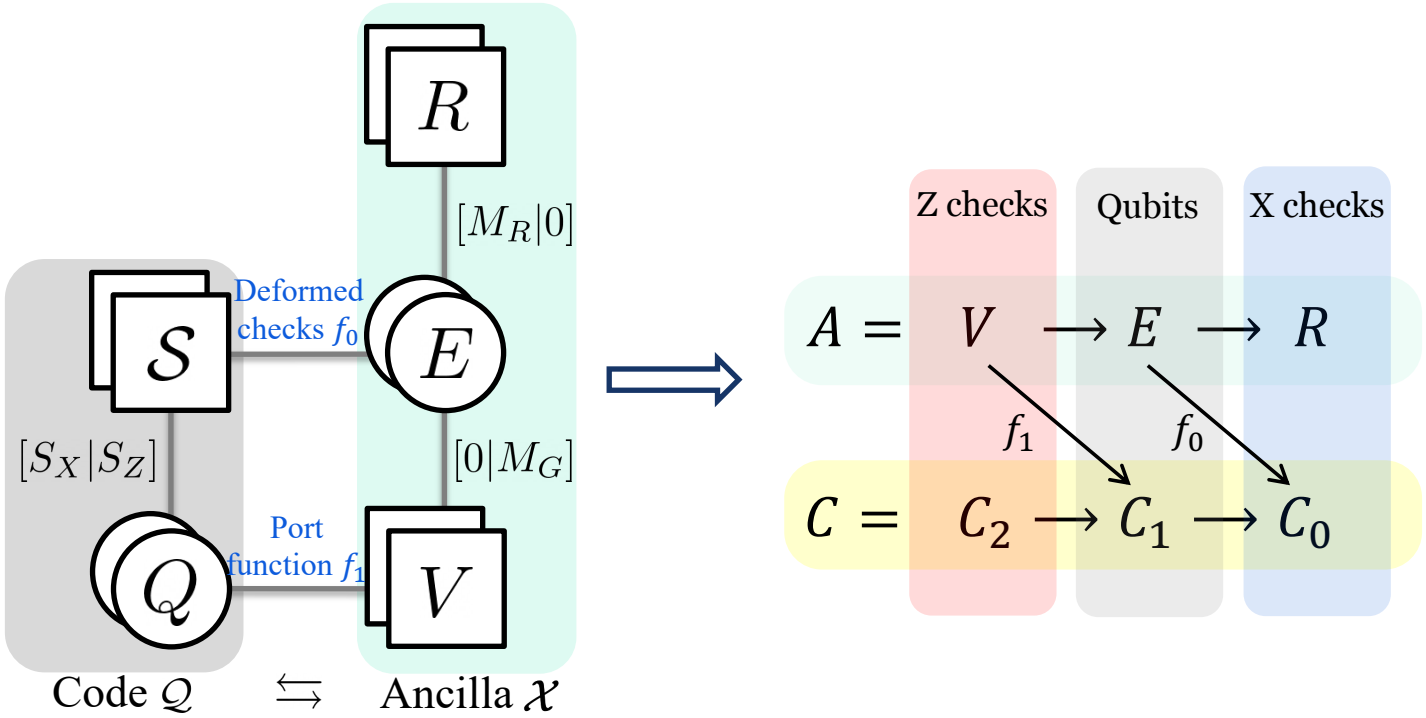




Mapping cone

Shift one complex by one degree. Changes operational meaning!

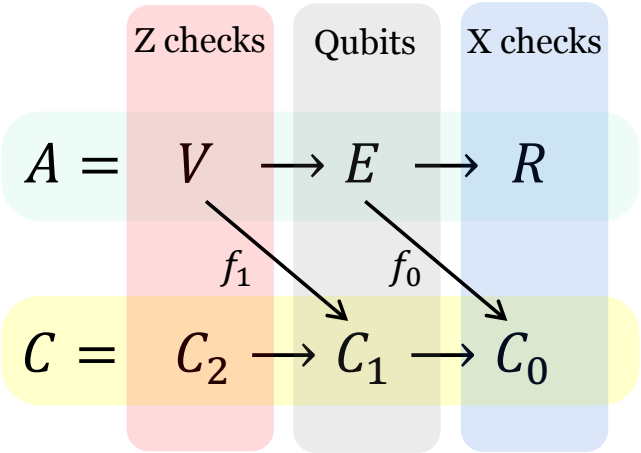
Surgery as Mapping Cones



Pros and Cons

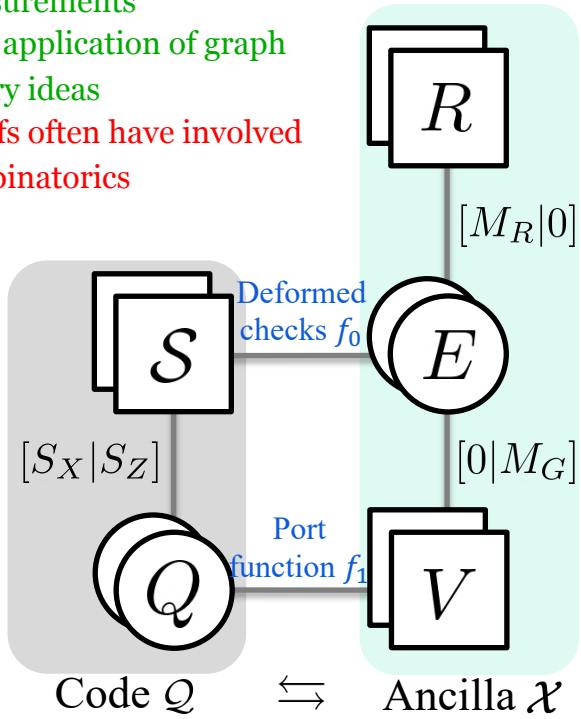
Chain map formalism:

+ More algebraic constructions and proofs
+ Easy tracking of logical qubits
- Have trouble handling non-CSS measurements and codes



Graph formalism:

+ Can easily handle non-CSS measurements
+ Easy application of graph theory ideas
- Proofs often have involved combinatorics



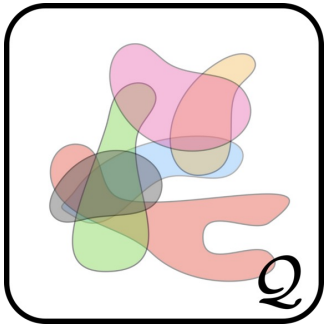
Extractor Architectures

Changing perspective

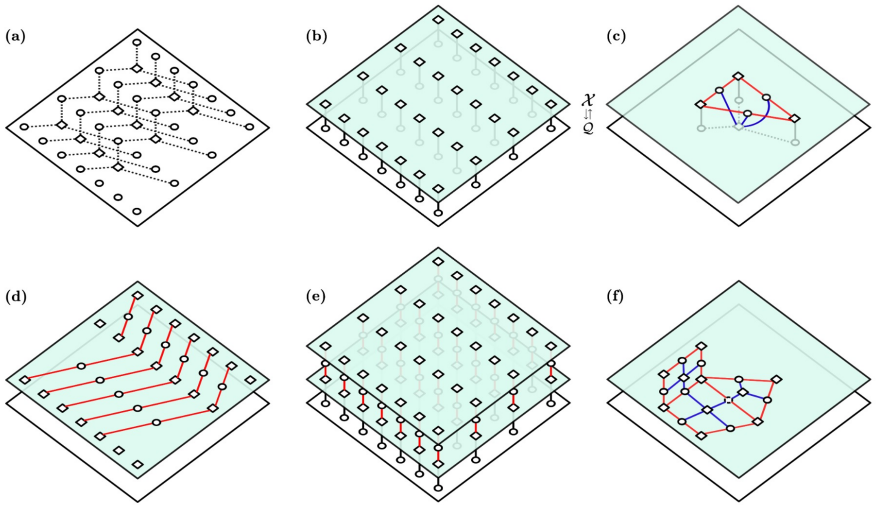
We have just figured out how to perform one gate. However, what we really want is an architecture that can perform universal computation.

How can we achieve that? Logical control on high-rate (k = O(n)) codes is fundamentally different from that on constant-dimension (k = O(1)) codes. For PPMs, there are 4^k many logical Pauli operators. We can't build 4^k many surgery systems!

Can we build one system that enables *all* measurements? A one-shot solution to computing on all QLDPC codes?



One system, all measurements



Let's build one graph and one set of connections, such that for any operator, there is a subgraph and a subset of connections that satisfy the desiderata.

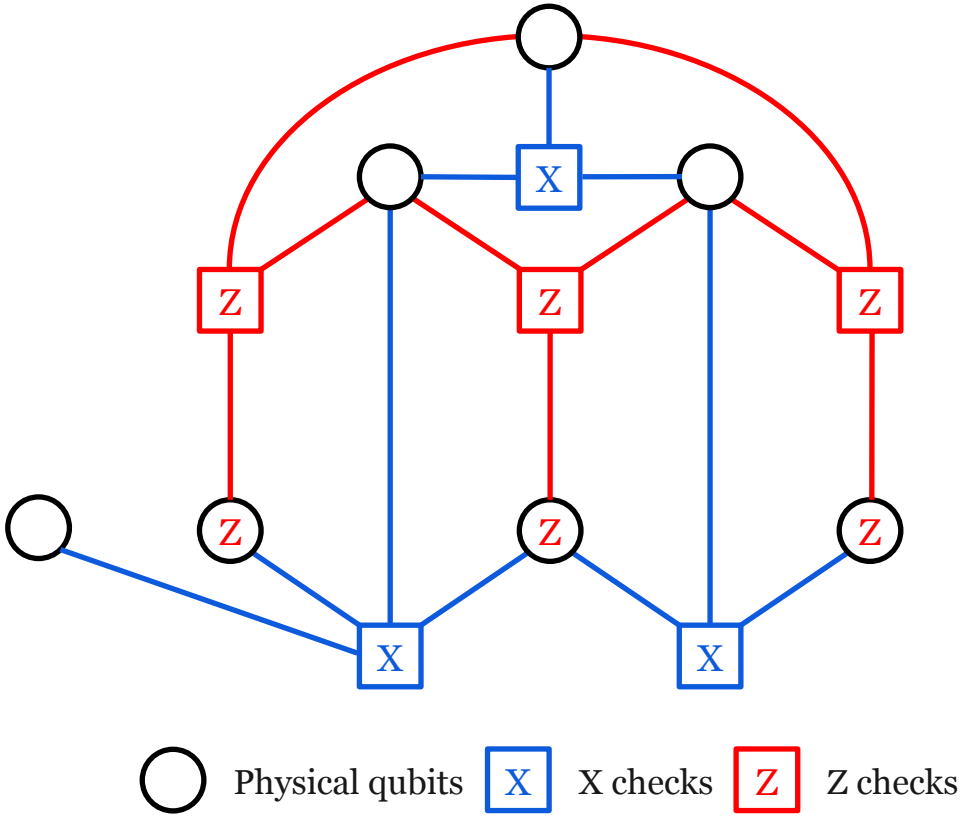
1. Create one vertex for each physical qubit
2. Port function: 1-to-1
3. For every stabilizer check, add a cycle of edges connecting the vertices corresponding to qubits.
4. Add edges to boost expansion
5. Sparsify the cycle basis ('decongestion' + 'thickening'), measure all cycles

Summary: how to build graphs for surgery

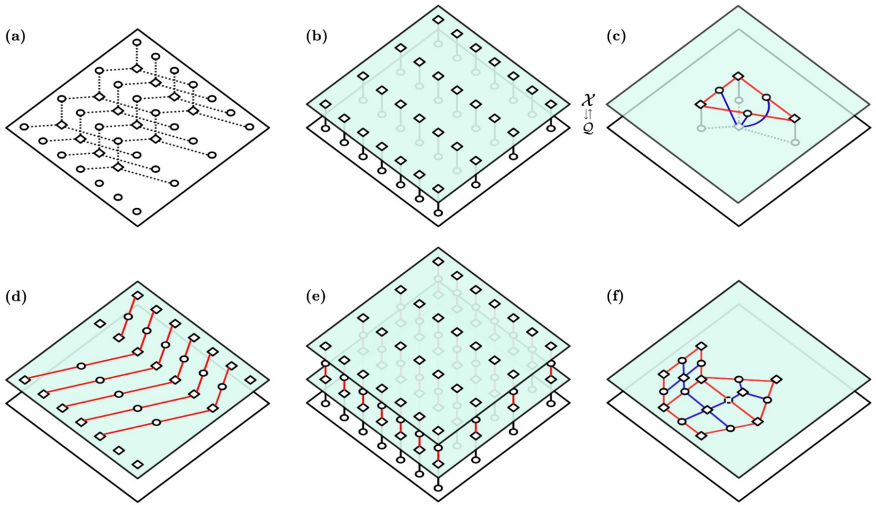
One common way to build a surgery graph for a logical operator L:

1. Create one vertex for each physical qubit
2. Port function: 1-to-1
3. Add edges (‘path matchings’) to ensure commutation
4. Add edges to boost expansion
5. Sparsify the cycle basis (‘decongestion’ + ‘thickening’), measure all cycles

Satisfy all our desiderata!



One system, all measurements



This system now enables all measurements. We call it an extractor, as it extracts logical observables from a base memory.

When you want to measure a specific operator:

1. Use the entire graph
2. Use the 1-to-1 port function
3. For every stabilizer check that needs to deform, use a subset of the cycle

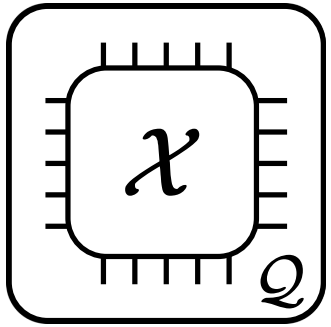
4^k fault-tolerant surgery operations using one LDPC extractor. In principle, can be built with fixed-connectivity hardware.

A Processing Unit

Current model of sequential surgery:
For *any* code, given *any* set of logical Paulis, can build an extractor system that enable measurement of *any* operator from the set, *one at a time*.

- This is the current definition of the word 'extractor'.
- Extractor size ranges between $\tilde{O}(\mathrm{d}) \sim \tilde{O}(\mathrm{n})$.

Key idea: extractors augment QLDPC memories into processors.



An extractor-augmented computational (EAC) block.

Single Operator
[Cohen et al. 2110.10794], [Williamson & Yoder 2410.02213], [Ide et al. 2410.02753], [Yuan et al. 2603.05082], [Oratomic 2603.28627]

8~16 Operators
[Cross et al. 2407.18393], [IBM 2506.03094], reaches hundreds by automorphisms [Iceberg 2602.11457], reaches 4^k by qubit movement

4^k Operators
[He et al. 2503.10390], [Blue et al. 2606.03507] , [Han et al. 2607.28605], [Bhardwaj et al. 2607.28795], [Webster et al. 2609.21249], [Lu et al. 2610.19073]

# of operators covered by extractors

Putting pieces together

Now we have a bunch of processing blocks. We need to:

- Connect them together, and
- Feed them magic states.

Can we connect two extractors together?

Connecting two graphs is very easy – just add edges.

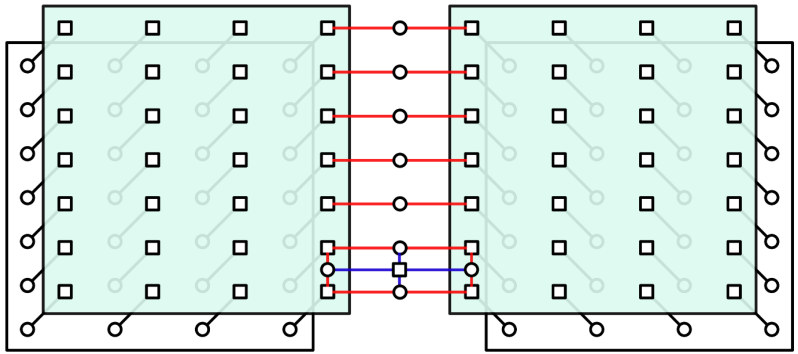
- Add d edges to ensure expansion
- d edges create d-1 cycles, need to sparsify them

For two copies of the same graph, very easy! This is what's called a bridge system[1].

How about for two arbitrary graphs?

- Still possible! This is called an adapter system[2].

This is wonderful – we can now connect arbitrary codes, factories, or fault-tolerant gadgets together with surgery!

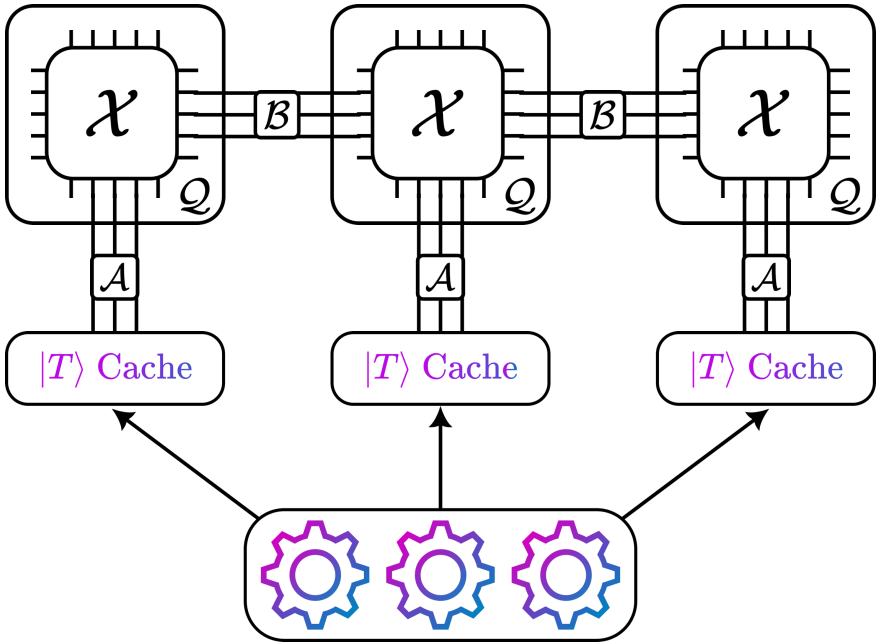


[1] [Cross et al. 2407.18393], [2] [Swaroop et al. 2410.03628]

Extractor architectures for Pauli-based FTQC

Building an architecture with extractors is like putting together Lego pieces:

1. Choose your QLDPC codes/memory
2. Choose your logical control (PPMs) and extractor systems
   - ➢ This choice deeply impacts/depends on compilation method and resource budget!
3. Choose your magic state factories
4. Connect them up with adapters

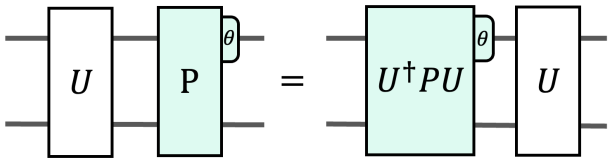


Compilation for an Extractor Architecture

Compilation similar to Game of Surface Code. Given a logical circuit of Pauli rotations, we consider three types of gates:

1. Pauli π/8 rotations,
2. Pauli π/4 rotations supported within one EAC block (in-block Cliffords),
3. Pauli π/4 rotations supported on two EAC blocks (cross-block Cliffords).

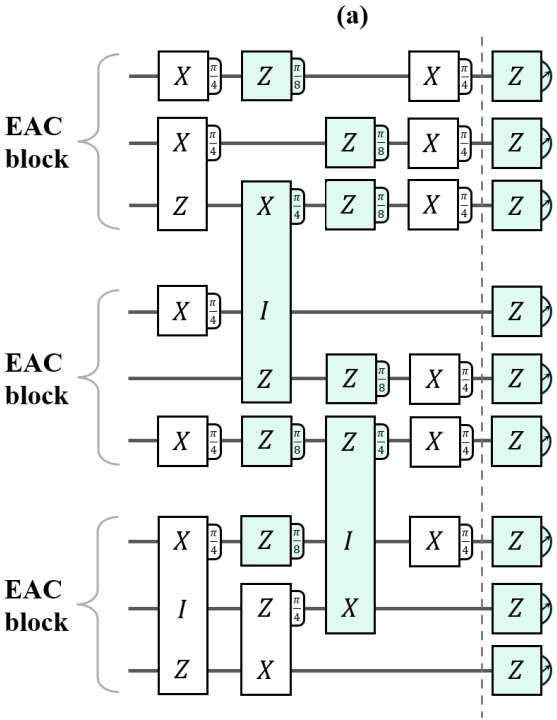
We conjugate all in-block Cliffords to the end of the circuit. This conjugation changes the Pauli axis of type 1 and 3 rotations, but do not expand their support to other EAC blocks.



1. We implement type 1 & 3 rotations via logical measurements;
2. In-block Cliffords are absorbed by standard basis measurements at the end of the computation.

Circuit Example

1. Conjugate all in-block Cliffords to the end of the circuit.

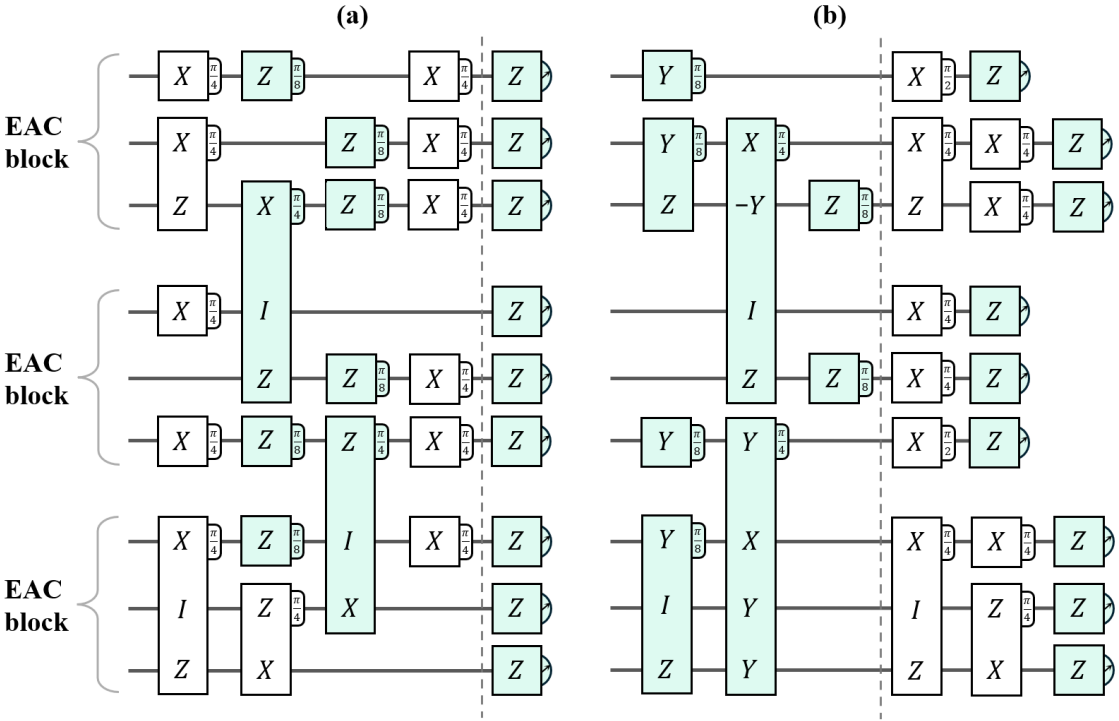


Green operations are what we compile and implement. White operations are in-block Clifford that are compiled away.

Circuit Example

1. Conjugate all in-block Cliffords to the end of the circuit.

2. Absorb in-block Cliffords by the final measure-out.

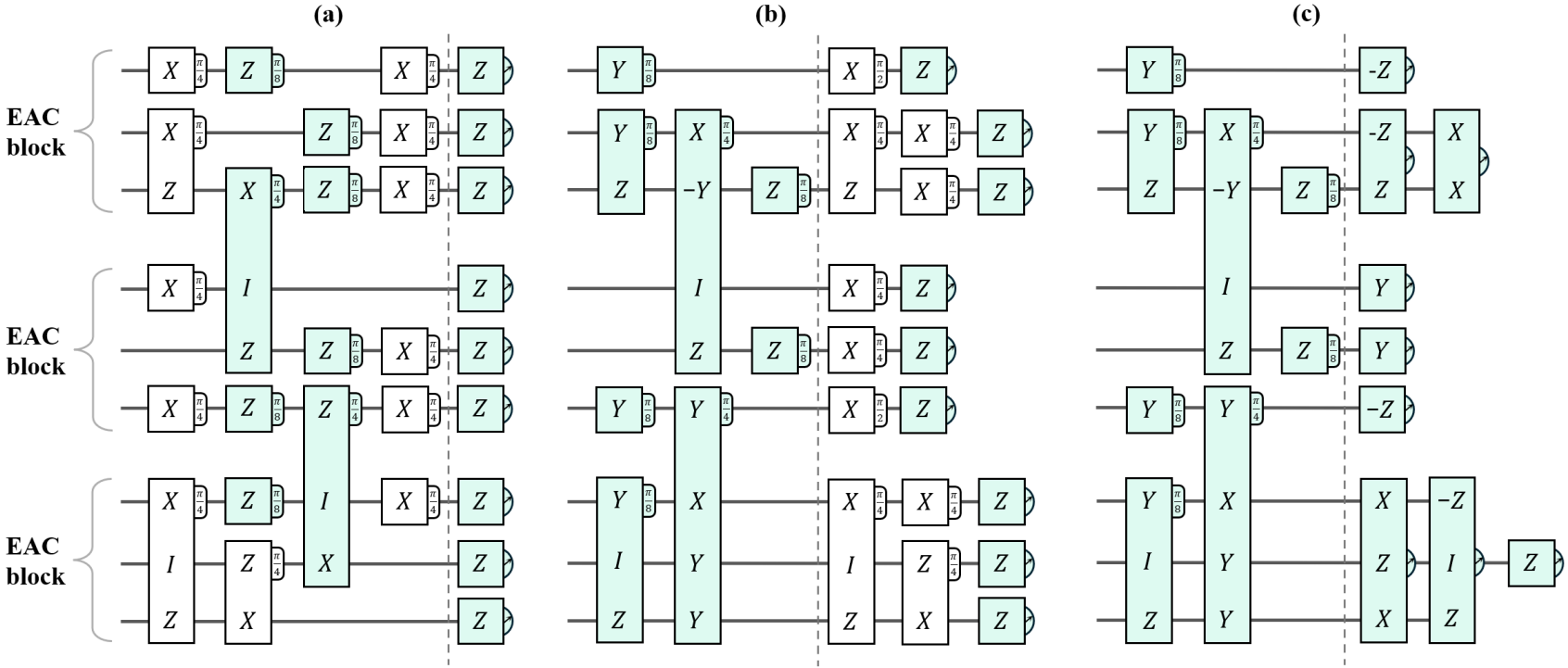


Green operations are what we compile and implement. White operations are in-block Clifford that are compiled away.

Circuit Example

1. Conjugate all in-block Cliffords to the end of the circuit.

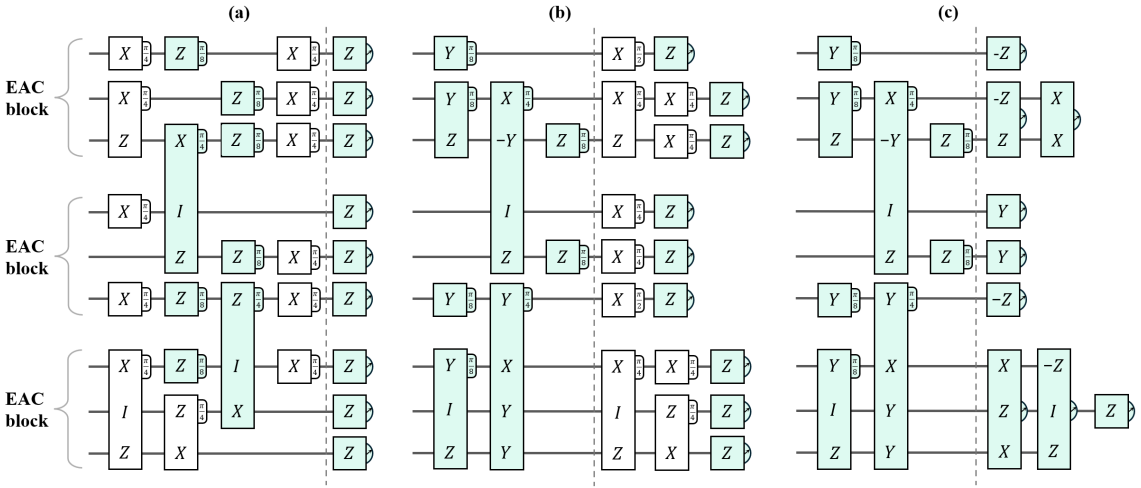
2. Absorb in-block Cliffords by the final measure-out.



Green operations are what we compile and implement. White operations are in-block Clifford that are compiled away.

Remarks

- In-block Clifford gates are essentially free.
- This compilation heavily relies on the fact extractors can measure any logical Pauli.
- Bottleneck: magic state supply speed and number of cross-block gates.
- Highly optimizable for specific applications.

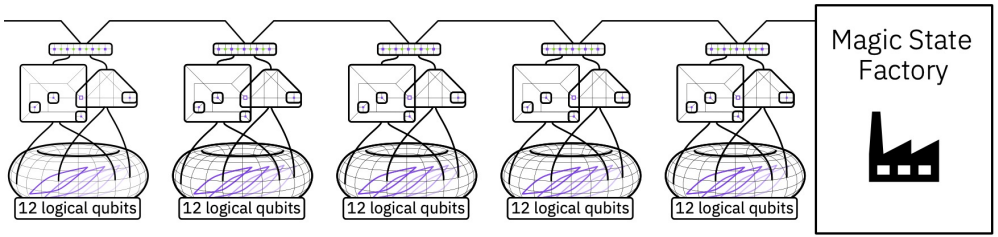


Modern Resource Estimations

Design choices of recent architectures

Similarity: sequential logic

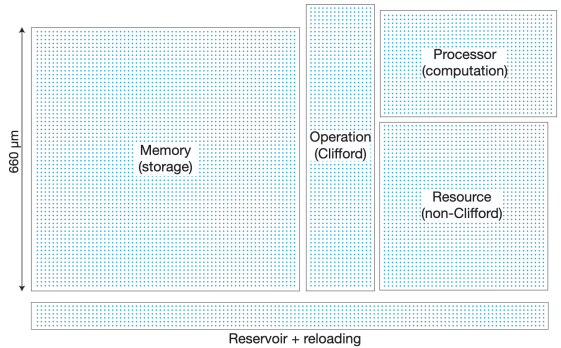
- Every CCZ (or T) gate in the circuit is compiled into one (large) PPM. Many (or all) Clifford gates are compiled away.
- Magic state throughput is one/logical cycle.



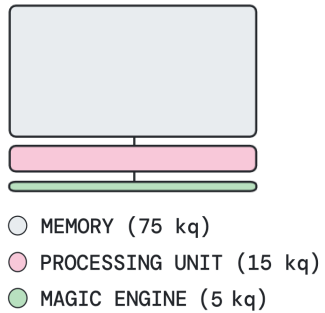
IBM's bicycle architecture[1]

Differences:

- Choice of codes – heavily impact space overhead
- Choice of extractors, including loading – heavily impacts time overhead
- Choice of magic state factories – cost of factory limits magic throughput
- Physical assumptions on qubits
- Compilation and resource estimation target
- Many more details



Oratomic's architecture [3]



Iceberg's architecture[2]

[1] [IBM 2506.03094], [2] [Iceberg 2602.11457], [3] [Oratomic 2603.28627].

Demystifying the numbers

Back-of-envelop space estimates for leading factors

Memory: Use high-rate codes[1-3], ~1500 logical qubits in 5000~9000 physical qubits.

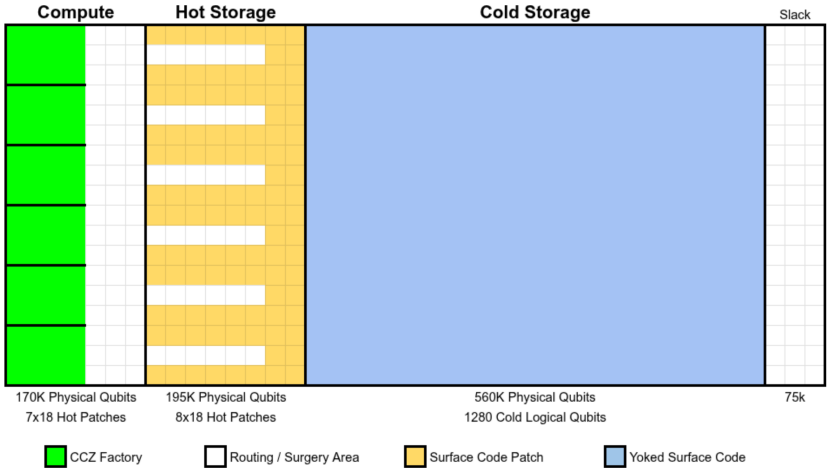
Compute: Can use smaller processor codes, or compute directly on memory.

- For simplicity, we can use single-operator extractors to execute large PPMs directly on memory.
- Rule of thumb: roughly the same size as memory.

Magic states: cultivate to d~15 surface codes, load into n~500 QLDPC codes using extractors, distill.

- Many designs ranging from 5000~35000 qubits.
- Need detailed simulation to pin down exact costs.

Total: 15k ~ 55k physical qubits.



Baseline: Gidney 2025 estimate [2505.15917]

- Surface code architecture with lattice surgery
- RSA-2048 in less than 1 million qubits, less than 1 week, on superconducting qubits
- One CCZ state per logical cycle, some parallelism in Clifford operations.

[1] [Kasai, 2601.08824], [2] [Chen et al. 2604.16209], [3] [Oratomic 2603.28627].

Demystifying the numbers

Back-of-envelop time estimates for leading factors

Hardware: neutral atom code cycle 1 *ms* vs. superconducting code cycle 1 μs. Factor of 1000!

- Takes Gidney's 5 days to 5000 days.

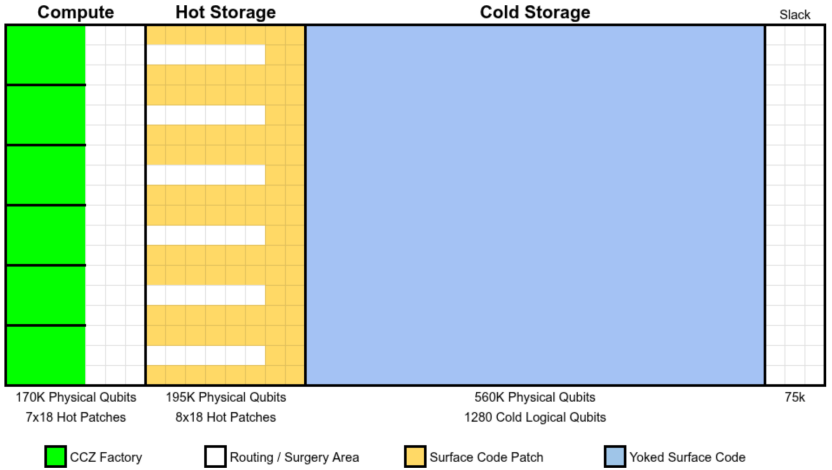
Compilation:

- Load in/out between memory and compute;
- Compilation of PPMs;
- Choice of magic states (T vs. CCZ);
- Many more factors which depends on architectural and algorithmic choices.

Logical clock speed:

- In current estimates, 1 logical cycle = O(d) ~ 20 physical cycles. Varies with assumptions.

Total: on the order of 10k days.



Baseline: Gidney 2025 estimate [2505.15917]

- Surface code architecture with lattice surgery
- RSA-2048 in less than 1 million qubits, less than 1 week, on superconducting qubits
- One CCZ state per logical cycle, some parallelism in Clifford operations.

Time overhead is not fundamental (yet)

Magic state throughput:
- Can we do more than 1 CCZ per logical cycle?
- Can we consume more than 1 CCZ per logical cycle?

⇒ Magic state production is the key bottleneck right now!
- Can we design more efficient factories?
- Parallel extractors can be built for consumption.[1-3]

Hardware: can reconfigurable qubits move faster?

⇒ Need to ask our experimental overlords. However, we can implement surgery on superconducting qubits, which are a lot faster!

Logical cycle time:
- In current estimates, 1 logical cycle = O(d) ~ 20 physical cycles
- Can this be reduced to O(1)?

⇒ Yes! There are ways to implement surgery with constant time overhead.[4]

Compilation:
- If the extractor is limited, a large PPM would need to be compiled into supported 'native instructions'
- Leads to the 10x slow down in bicycle architecture

⇒ Can be avoided by building a bigger extractor.

---

[1] [Zhang & Li 2408.01339], [2] [Cowtan et al. 2503.05003], [3] [Zheng et al. 2510.08523] [4] See Introduction of [Chang et al. 2603.02157].

Challenges

Building a LDPC quantum computer will not be easy! Many challenges remain, and we need lots of solid work to resolve them.

Decoding:

- LDPC decoding is *much harder* than surface code decoding.
- Amazing progress in recent years, but we still needs lots of work to improve speed *and* accuracy.
- Good vs. bad decoding can lead to order-of-magnitudes difference in logical performance.

Hardware complexity:

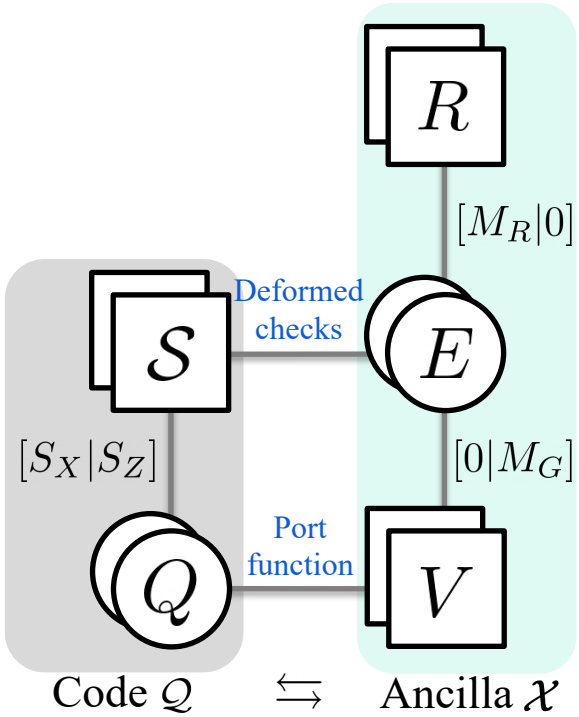
- Implementing QLDPC codes & surgery won't be easy, even for reconfigurable qubits.
- For superconducting qubits, extractors can enable universal FTQC. However, the connectivity needed for ultra-high-rate codes (rate 1/2 or 1/3) is more than current assumptions.

More QEC theory:

- To enable large-scale FTQC, we need further reduction in space-time overheads.
- Many concrete problems here.

Further topics in surgery: Parallel Surgery

Parallel Surgery via Hypergraphs



Measurements = kernel of adjacency matrix of G = sets of vertices that have even overlap with every hyperedge.

If G is a connected simple graph:

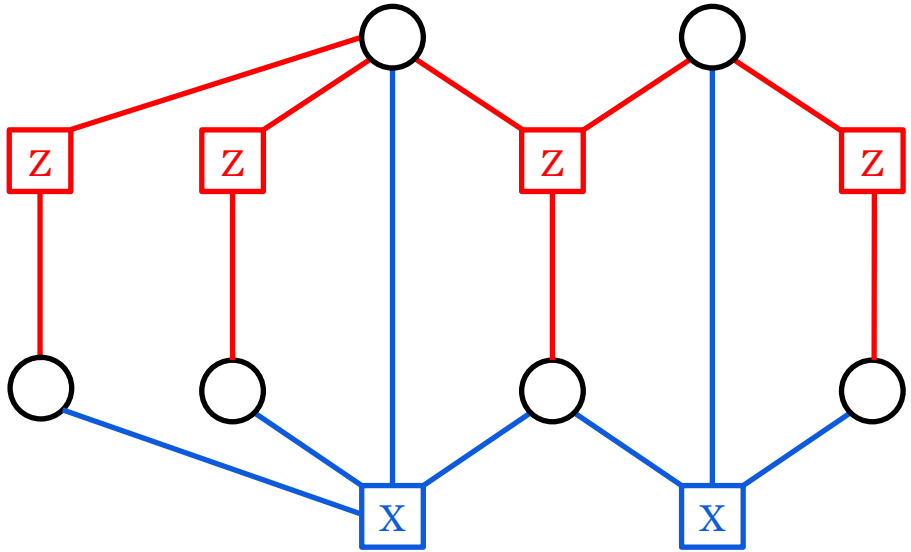
- Only 1 set: all the vertices. Implements one measurement.

If G is a hypergraph:

- Multiple sets, implements parallel measurements.

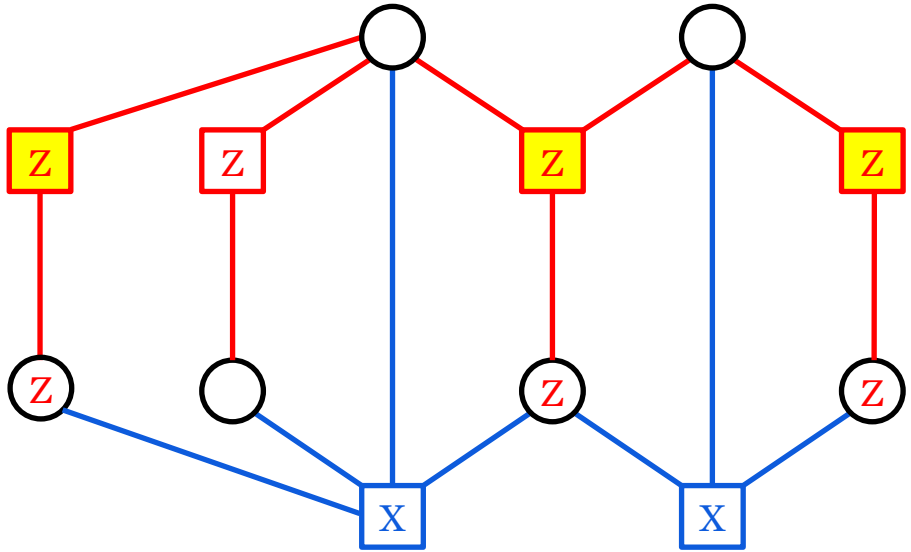
Parallel Measurements via Hypergraphs

With a hypergraph, we can measure multiple operators at the same time.



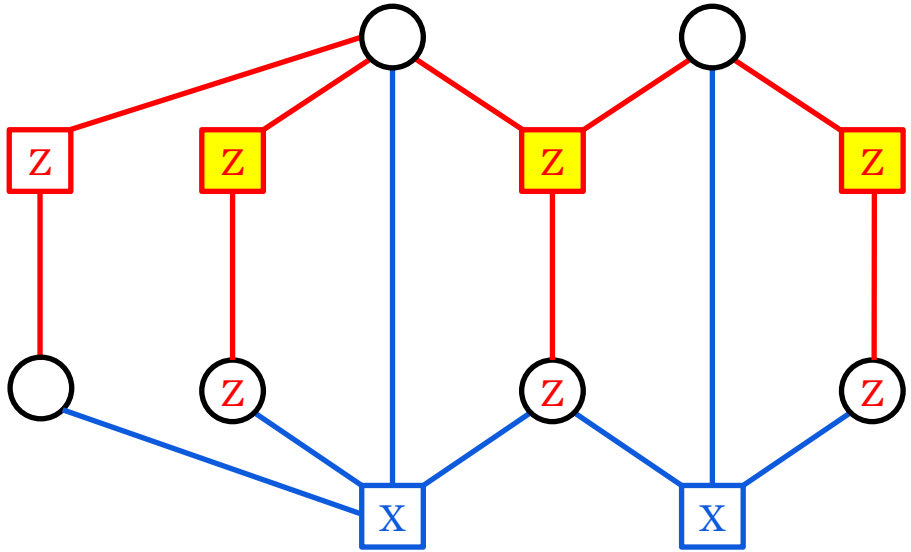
Parallel Measurements via Hypergraphs

With a hypergraph, we can measure multiple operators at the same time.



Parallel Measurements via Hypergraphs

With a hypergraph, we can measure multiple operators at the same time.



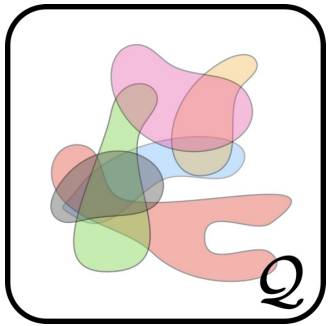
Parallel Surgery is Tricky

Goal: measure a collection of commuting logical Pauli operators simultaneously.

Problems: these operators overlap a lot. How do you build the hypergraph to:

- Measure all the target logical operators;
- Not measure the other operators that are in the same support;
- Ensure every other operator is protected;
- Ensure LDPC?

Hypergraphs are much harder to work with!



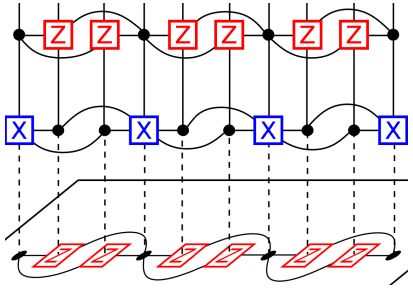
Branch and Measure

Divide and Conquer.

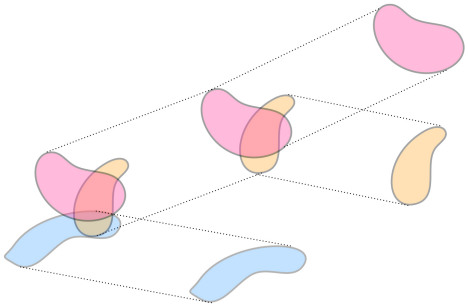
- Use “branching” to create new copies of target logicals on an ancilla system,
- O(log k) many branching steps can separate every logical into its own support
- measure them with graph surgery.[1,2]

Overhead:

- O(n log k) from branching
- O(n log d) for each graph-surgery measurement
- Total to O(kn log n).
- Best known up to logarithmic factors.



A 4-qubit operator on the code gets ‘branched’ into the ancilla system.



[1] [Zhang & Li 2408.01339], [2] [Cowtan et al. 2503.05003].

Recent Progress

Goal: measure a collection of commuting logical Pauli operators simultaneously.

A range of techniques:

- Use "branching" to create new copies of target logicals on an ancilla system, measure them with graph surgery.[1,2]
- Start with a surgery gadget that measures all single-logical X, puncture and/or augment it.[3]
- Heuristic construction of hypergraph surgery.[4]
- Special designs tailored to product codes.[3-7]

Caution: what operators are you measuring? How complicated are they?

- Parallelism alone is easy to achieve: use transversal CNOT. What's hard is parallel *and* addressable.
- Measuring complicated group of (non-CSS) Pauli products[1,2] costs more space than measuring many constant-logical-weight products in the same basis.[3,4,5,7]
- Both are interesting primitives, make sure to distinguish them when reading/writing a parallel surgery paper.

---

[1] [Zhang & Li 2408.01339], [2] [Cowtan et al. 2503.05003]. [3] [Han et al. 2607.28605]. [4] [Zheng et al. 2510.08523].
[5] [Gu et al. 2603.05398], [6] [Chang et al. 2603.02157], [7] [Bhardwaj et al. 2607.28795].

Further topics in surgery: Fast Surgery

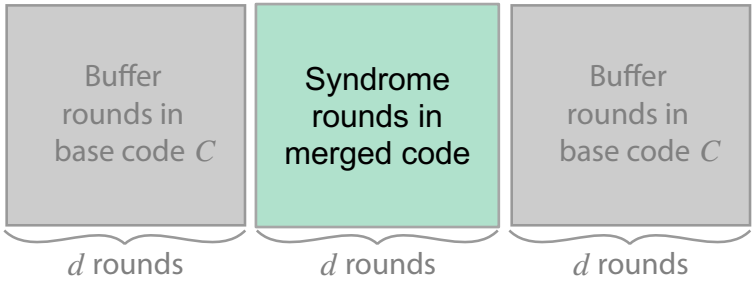
What do you mean fast?

Standard surgery takes 3d rounds of syndrome extraction.

- Fault-tolerance comes from a continuous, O(d)-long chain of LDPC detectors.
- Time distance is given by repetition, space distance is given by distance of merged code.

If you want to speed things up, you need:

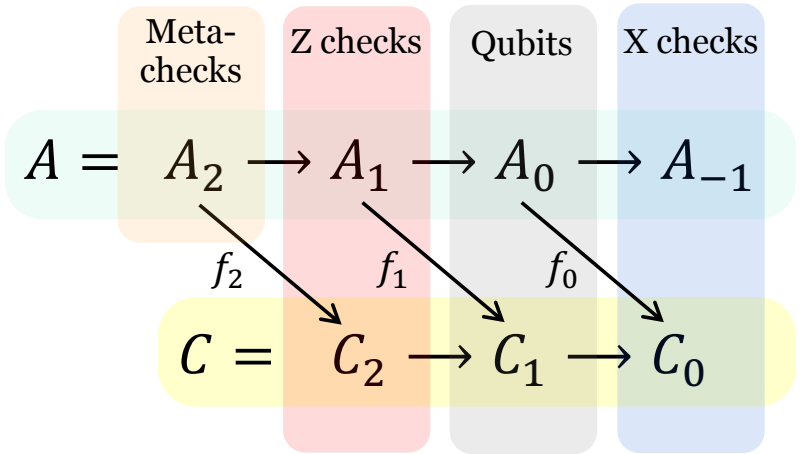
- New protocol that produces sufficient LDPC detectors for time distance in shorter time
- New analysis of space-time distance translation.



Fast Surgery Uses Metachecks

Meta-checks protect against measurement errors in vertex checks

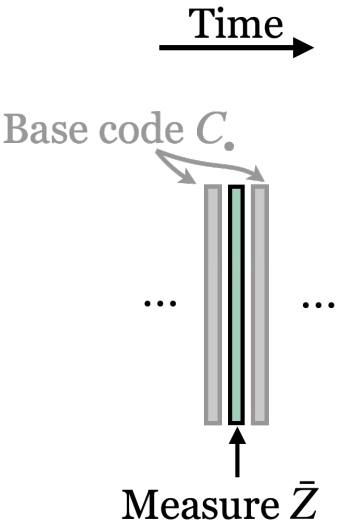
- Ensures time-like distance is high
- 4-term complex is a hypergraph → naturally gives parallel measurements.



Is this fault-tolerant?



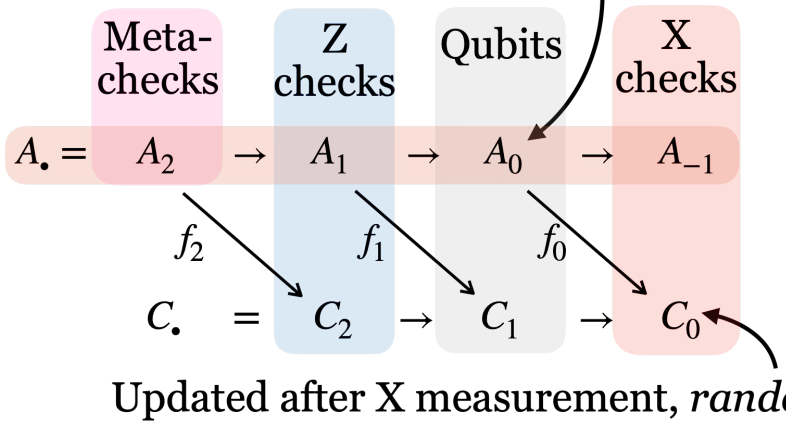
1. 1 round in the base code $C_\bullet$
2. 1 round in the deformed code (measure $\bar{Z}$)
3. 1 round in the base code $C_\bullet$



Start: init. A_0 qubits in $|+\rangle$

Finish: measure A_0 qubits in X

Measure $\bar{Z}$

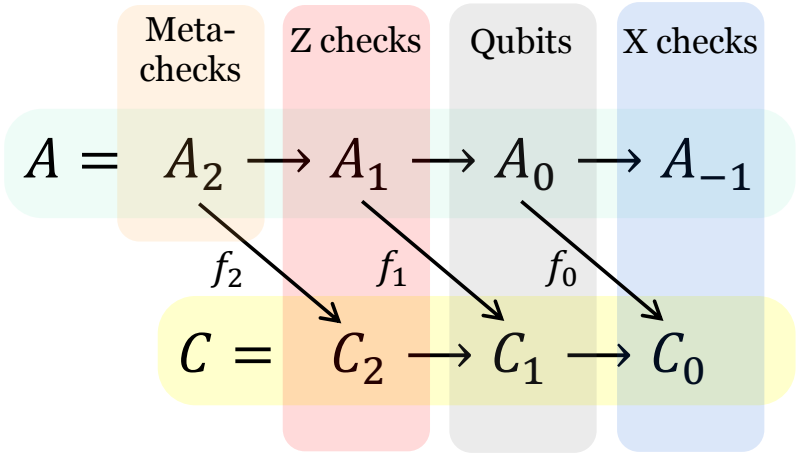


- State preparation-like error. Pauli frame is random after surgery.
- Only 1 round in $C_\bullet$ to learn the new stabilizer values → incorrectly inferred Pauli frame
- Can affect future logical operations

Single-shot Surgery Example

Consider 3D surface codes, which have metachecks in one basis.

- Lattice surgery between two 3D surface codes in the correct basis can be single-shot.



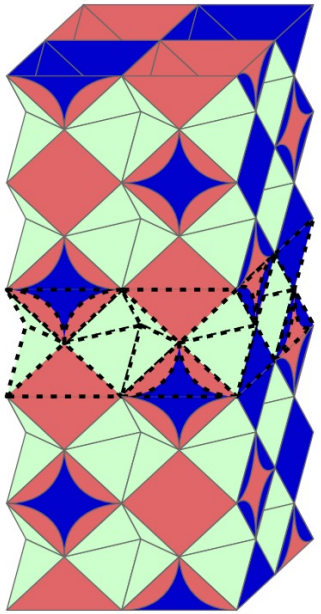
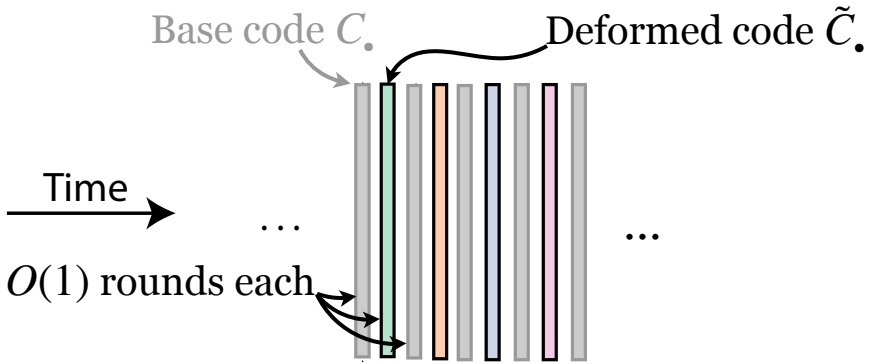


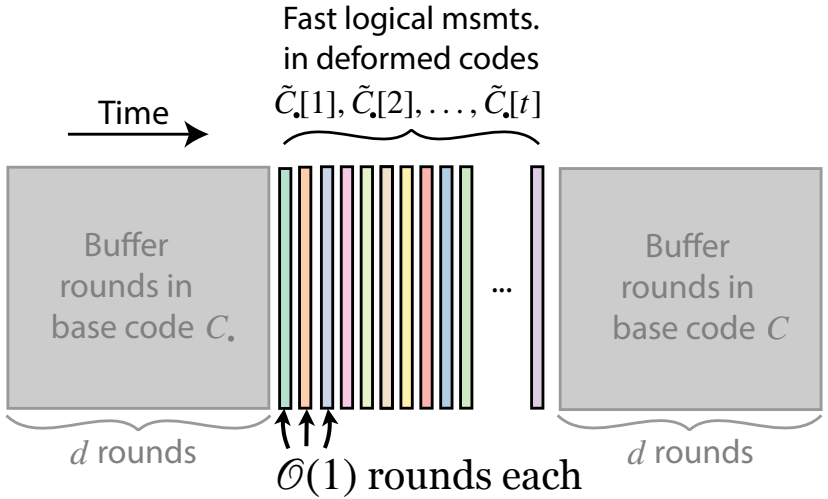
Figure from arxiv 1801.04255.

Single-shot surgery [1-3]



- Requires **single-shot preparation** of **deformed code**, and **base code** $C_\bullet$.
- Restricts what $C_\bullet$ can be.
  - Base code must have **meta-checks**
- Concurrent work[3]: deformed $\tilde{C}_\bullet$ may introduce large space overhead: $\tilde{O}(nkd)$

Constant-time surgery [4, 5]



- C need not be single-shot state preparable
- Sufficient detectors to infer the Pauli frame after each msmt
- Each $\tilde{C}[i]$ needs to satisfy certain properties to ensure fault-tolerance

[1] Hillman *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]
[4] Cowtan, He, Williamson, Yoder. [2510.14895]
[5] Chang, He, Yoder, Zhu, Jochym-O'Connor. [2603.02157]

Fault-tolerance conditions for constant time surgery

Let $C_\bullet$ be a CSS code of distance d, and consider a **sequence of surgery operations** defined by ancilla complexes $A[1]_\bullet, \ldots, A[t]_\bullet$ and chain maps $f[1] : A[1]_\bullet \to C_\bullet, \ldots, f[t] : A[t]_\bullet \to C_\bullet$.

Then the constant-time surgery protocol has **fault distance** d **if:**

- Each ancilla complex $A[i]_\bullet$ has meta-check distance at least d.

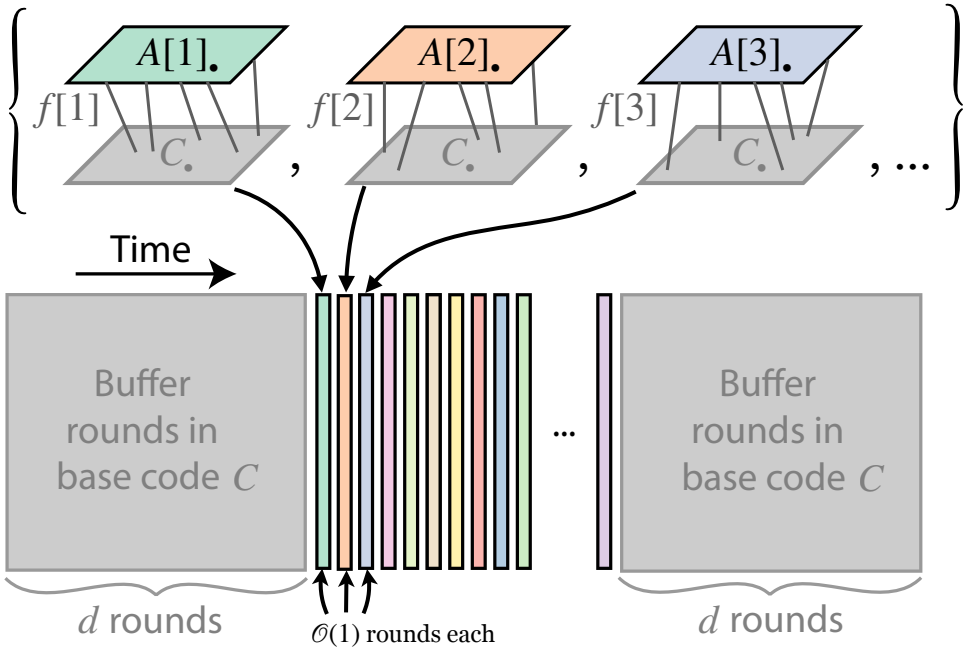
Can protect up to d ancilla measurement errors

- The compacted code $\mathsf{CC}_\bullet$* has distance $\geq d$.

Unmeasured logical qubits have high distance

- Chain maps are partially injective.

Error propagation is bounded



*a CSS code defined by t surgery operations

Explicit low-overhead construction for the HGP code

Given an $[[n, k, d]]$ hypergraph product code, we can construct a sequence of ***LDPC*** ancillary complexes that:

- **Large fault distance:** Satisfies fault tolerance conditions for constant time surgery**.**
- **Parallel, partially addressable measurements:** Parallelized measurement of rows (and columns) of Z (X) logical operators.
- **Constant time per measurement:** Each surgery operation takes $\mathcal{O}(1)$ **time** in amortization.
- **Low space overhead:** Each complex **uses** $\mathcal{O}(n \log(n))$ **physical ancilla qubits**.

2D hypergraph product codes

Family of codes that can achieve constant encoding rate and sublinear distance $[[n, O(n), O(\sqrt{n})]]$

"2D" refers to the product of *two* classical codes

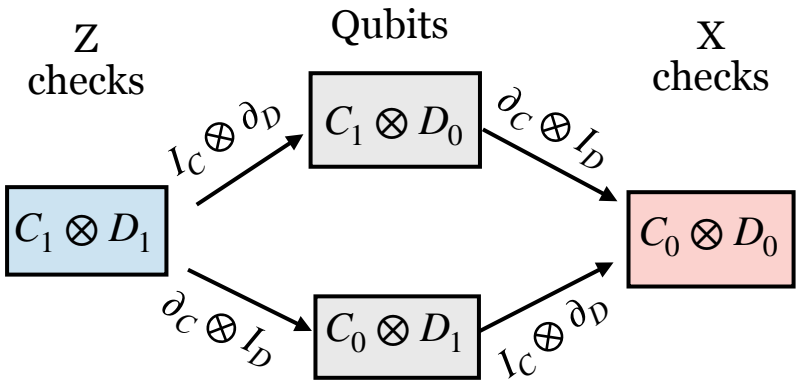
Bits Checks

Classical code 1: $C_\bullet = C_1 \xrightarrow{\partial_C} C_0$

Classical code 2: $D_\bullet = D_1 \xrightarrow{\partial_D} D_0$

The hypergraph product of these codes is

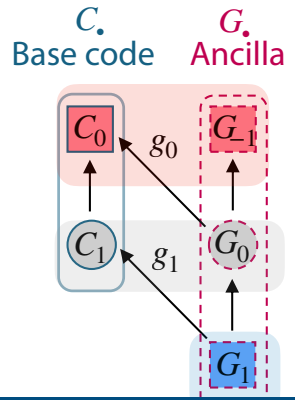
$(C \otimes D)_\bullet =$

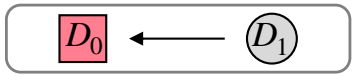


Constructing constant-time surgery gadgets

1. Apply surgery gadget on the classical code $C_\bullet$. The ancilla = $G_\bullet$.
2. Take the tensor product with $D_\bullet$.
3. Arrive at deformed code, "$(G + C) \otimes D$"

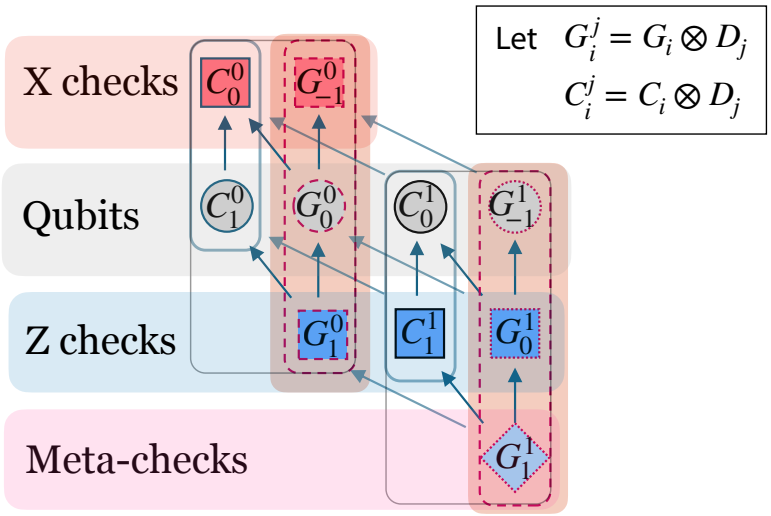
Surgery on classical code *(3-term chain complex)* $\otimes$ Classical code *(2-term chain complex)* $=$ Quantum code *(4-term. chain complex)*



$\otimes$ $D_\bullet =$ 

The ancilla complex $A_\bullet$ is $(G \otimes D)_\bullet$.

The chain map is $f : (G \otimes D)_\bullet \xrightarrow{g \otimes \mathrm{Id}_D} (C \otimes D)_\bullet$.

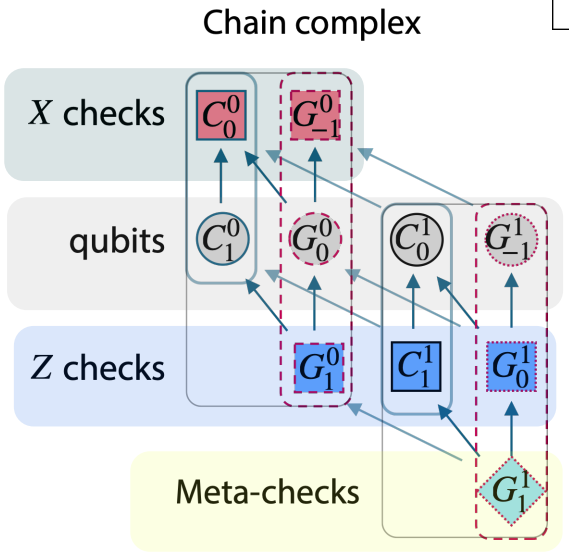


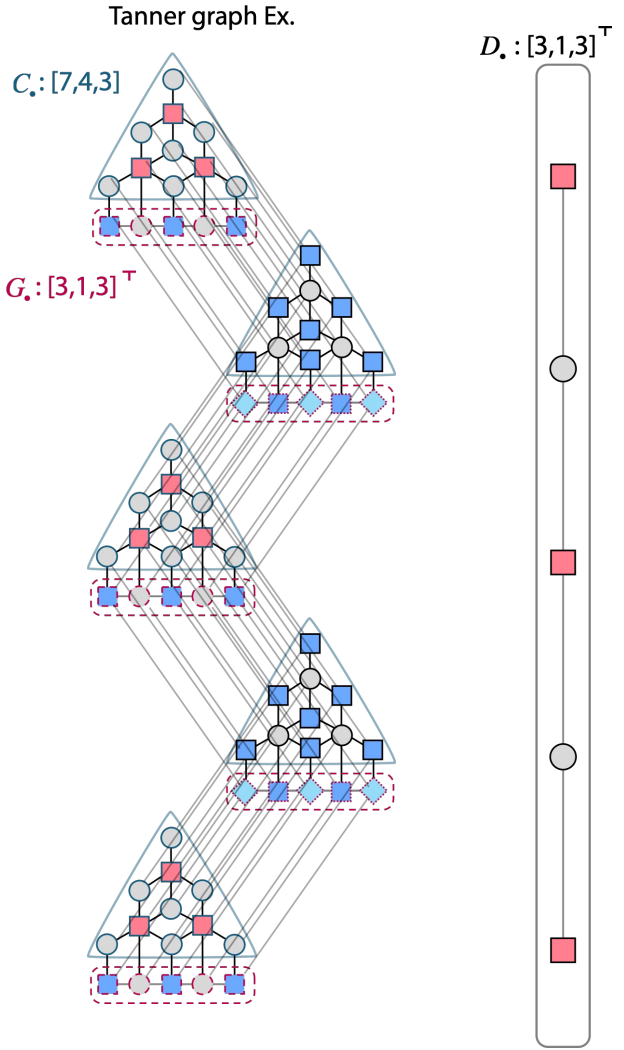
Ex. Constructing the deformed code

Take the hypergraph product with the other base classical code, D.

Let $G_i^j = G_i \otimes D_j$

$C_i^j = C_i \otimes D_j$





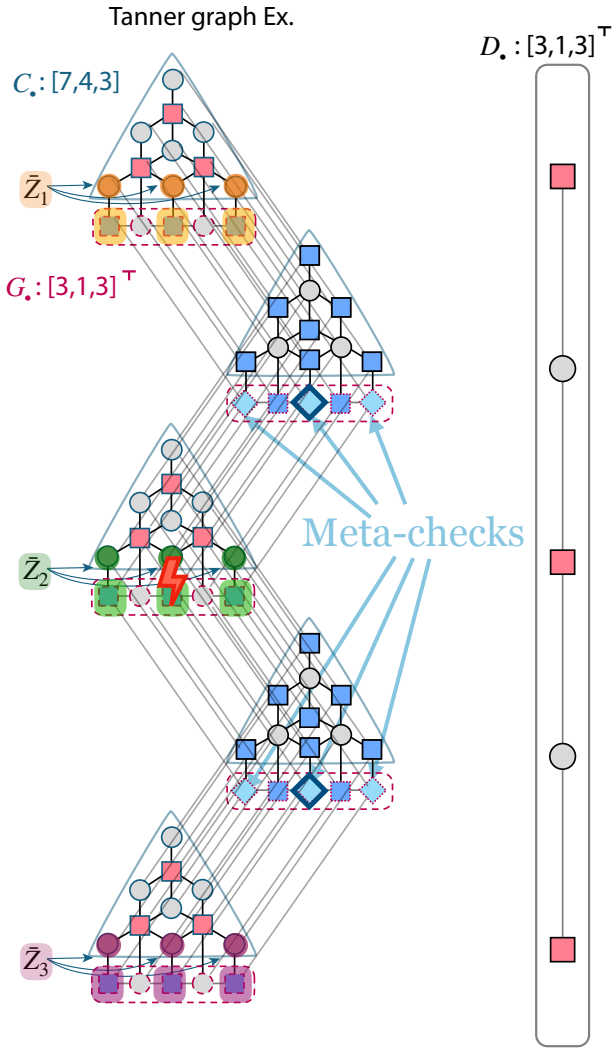
Ex. Constructing the deformed code

What does measuring the stabilizers of this deformed code accomplish?

- $\bar{Z}_1$, $\bar{Z}_2$, $\bar{Z}_3$ are logically equivalent representatives of the same logical operator.
- $\bar{Z}_1$, $\bar{Z}_2$, $\bar{Z}_3$ logical representatives are measured simultaneously — each equal a product of ancilla Z checks

Redundancy **by measuring different** ***representatives*** **of the same logical operator.**

Ensure fault tolerance against measurement errors.



Further topics in surgery: Magic Surgery

Weight Reducing *Clifford* Operators

Every Pauli stabilizer code has many Pauli operators that can be measured

- Measure Pauli implements Clifford operations
- Measure Clifford operators implement non-Clifford operations

Clifford measurement is much harder! Detectors are hard to build and decode.

Ask these folks

Magic Quantum Code Surgery

Kathleen (Katie) Chang,[1,2,*] Anasuya Lyons,[3,*] Yuanjie Ren,[4,*]
Harald Putterman,[3] Nathanan Tantivasadakarn,[5] Victor V. Albert,[6]
Benjamin J. Brown,[7,8] and Dominic J. Williamson[9]

Future directions

Future problems for QLDPC surgery

Decoding:

- Surgery decoding has not been deeply studied. Opportunity for impactful work!
- More simulations, such as with magic state loading.

Design of surgery gadgets:

- Given specific application, how can we best design the code and surgery gadgets to perform the algorithm?
- How can we design better parallel/fast surgery gadgets?

Clifford measurements:

- Clifford measurement with surgery is a very new topic – lots to understand!

Overall, lots to be discovered!

Future directions for QLDPC architectures

Design more efficient magic state factories

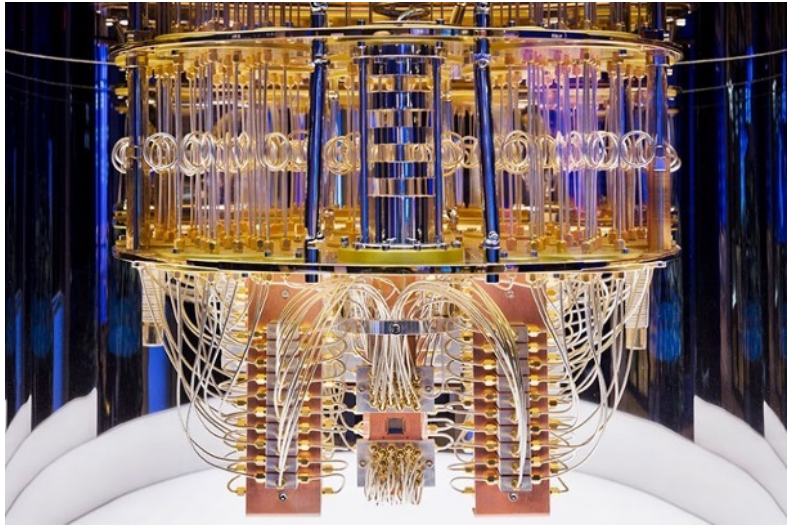
- Can we find practical codes with low-depth CCZ (or other non-Clifford) gates?[1,2]

Structured logical control co-designed with algorithm

- Algorithms and their subroutines often structured. Can we design the logical control (surgery or otherwise) with respect to their structure?[3]

Is time-overhead fundamental?

- Does more efficient encoding of logical information necessarily lead to slowdown in logical computation (due to addressability or routing costs)?[4]



[1] [Breuckmann et al. 2410.16250], [2] [Manon et al. 2508.10714] [3] [QuEra 2606.25011] [4] [Pattison & Nguyen 2411.03632]

Modern Surgery Techniques

Sunny Zhiyang He, November 2026

