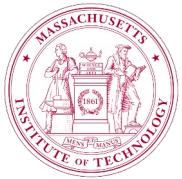
# Pauli-Based Computation on QLDPC Codes

## A tutorial on surgery and surgery-based architectures

Sunny Zhiyang He @ Benasque, August 2026



# Outline

I. Fault-tolerant quantum computation in 2026

II. Code surgery via graphs and chain maps

III. Surgery-based architectures

IV. Further techniques in surgery

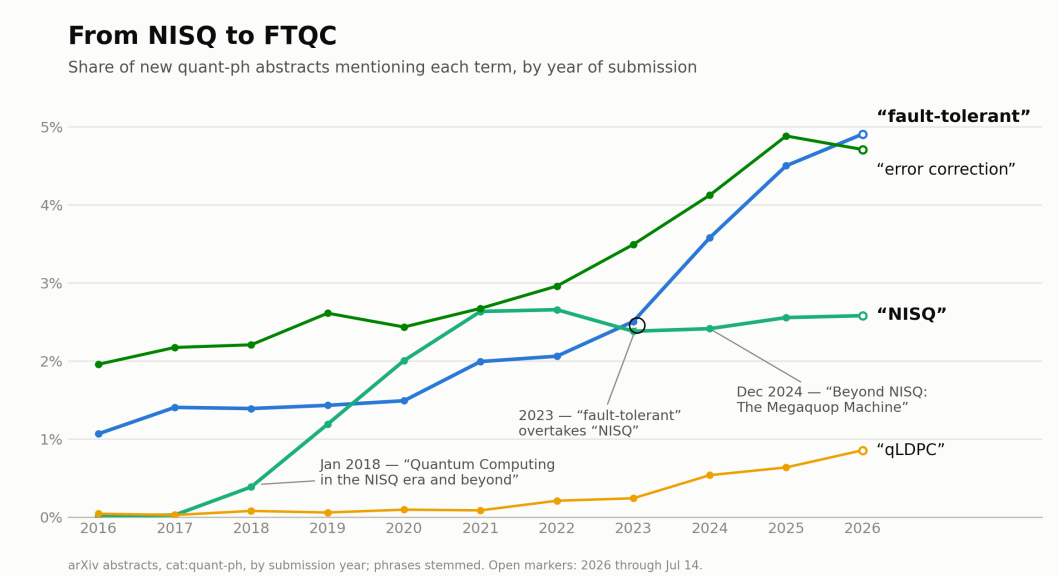
V. Future directions

# Fault-tolerant quantum computation in 2026

# Now is the time for FTQC

*2018:* “Because of the very hefty overhead cost of quantum error correction, the era of fault-tolerant quantum computing (FQTC) may still be rather distant. No one really knows how long it will take to get there.” — John Preskill, “Quantum Computing in the NISQ era and beyond”

*2026:* Industries converge to fault-tolerance, aim to deliver hundreds to thousands of logical qubits, from megaquop to teraquop, before 2030.



# Convergence of experiments and QEC theory

## QEC Theory

2019+: New theory for QLDPC codes, leading to asymptotically good codes in 2021

2019+: New decoding algorithms, focus on BP and ML

2020+: Development of simulation software, such as STIM and better decoders

2023+: Focus on finite-size QLDPC codes and their implementation on hardware

2023+: Computational methods for LDPC codes, including QLDPC surgery and transversal logic

## Experiments

2022+: Reconfigurable qubits enable non-local connectivity

2023+: Sub-threshold surface code memory, larger code gives more error suppression

2023+: Long-range connection that supports modular architectures

2024+: Magic state distillation and cultivation, gate teleportation

Consistently: Lower gate error rate, larger systems, better physical control

---

Finite-scale proposals to reduce space overhead, with focus on LDPC codes

New hardware capabilities, lower physical and logical error rates, larger scale systems

# Result of fusion: new proposals of finite-size FTQC

Gidney 2025 estimate [2505.15917]:

- Surface code architecture with lattice surgery, applied magic state cultivation and code concatenation.
- RSA-2048 in less than 1 million qubits, less than 1 week, on superconducting qubits.

IBM Bicycle Architecture [2506.03094]:

- Modular LDPC architecture, computation by LDPC surgery.
- Roughly 10 times less qubits and 10 times slower than surface codes.

Iceberg [2602.11457]:

- Modular LDPC architecture, computation by LDPC surgery.
- RSA-2048 in 100000 qubits, 30000 days, on neutral atoms.

Oratomic [2603.28627]:

- Modular LDPC architecture with high-rate codes, computation by LDPC surgery.
- ECC-256 in 10000 qubits, 1000 days, on neutral atoms. RSA-2048 in 11000 qubits, 43000 days.

Caution

These are proposals and estimates, not finalized reports. Some assumptions and methods are under discussion in the community.

More recent proposals: IonQ Walking Cat [2604.19481], Duke + UTAustin + Yale [2604.19735], QuEra's LDPC STAR [2606.25011], Google's modular hyperbolic architecture [presented at QEC26], and more.

# Result of fusion: new proposals of finite-size FTQC

Gidney 2025 estimate [2505.15917]:

- Surface code architecture with lattice surgery, applied magic state cultivation and code concatenation.
- RSA-2048 in less than 1 million qubits, less than 1 week, on superconducting qubits.

IBM Bicycle Architecture [2506.03094]:

- Modular LDPC architecture, computation by LDPC surgery.
- Roughly 10 times less qubits and 10 times slower than surface codes.

Iceberg [2602.11457]:

- Modular LDPC architecture, computation by LDPC surgery.
- RSA-2048 in 100000 qubits, 30000 days, on neutral atoms.

Oratomic [2603.28627]:

- Modular LDPC architecture with high-rate codes, computation by LDPC surgery.
- ECC-256 in 10000 qubits, 1000 days, on neutral atoms. RSA-2048 in 11000 qubits, 43000 days.

More recent proposals: IonQ Walking Cat [2604.19481], Duke + UTAustin + Yale [2604.19735], QuEra's LDPC STAR [2606.25011], Google's modular hyperbolic architecture [presented at QEC], and more.

**Observation 1**

All (except Gidney's) architectures are done with LDPC codes. We see dramatic reduction in space overhead, which has been the key bottleneck for FTQC.

# Result of fusion: new proposals of finite-size FTQC

Gidney 2025 estimate [2505.15917]:

- Surface code architecture with lattice surgery, applied magic state cultivation and code concatenation.
- RSA-2048 in less than 1 million qubits, less than 1 week, on superconducting qubits.

IBM Bicycle Architecture [2506.03094]:

- Modular LDPC architecture, computation by LDPC surgery.
- Roughly 10 times less qubits and 10 times slower than surface codes.

Iceberg [2602.11457]:

- Modular LDPC architecture, computation by LDPC surgery.
- RSA-2048 in 100000 qubits, 30000 days, on neutral atoms.

Oratomic [2603.28627]:

- Modular LDPC architecture with high-rate codes, computation by LDPC surgery.
- ECC-256 in 10000 qubits, 1000 days, on neutral atoms. RSA-2048 in 11000 qubits, 43000 days.

More recent proposals: IonQ Walking Cat [2604.19481], Duke + UTAustin + Yale [2604.19735],
QuEra's LDPC STAR [2606.25011], Google's modular hyperbolic architecture [presented at QEC], and more.

Observation 2

LDPC surgery serves as the computational backbone for half of these proposals.

# Result of fusion: new proposals of finite-size FTQC

Gidney 2025 estimate [2505.15917]:

- Surface code architecture with lattice surgery, applied magic state cultivation and code concatenation.
- RSA-2048 in less than 1 million qubits, less than 1 week, on superconducting qubits.

IBM Bicycle Architecture [2506.03094]:

- Modular LDPC architecture, computation by LDPC surgery.
- Roughly 10 times less qubits and 10 times slower than surface codes.

Iceberg [2602.11457]:

- Modular LDPC architecture, computation by LDPC surgery.
- RSA-2048 in 100000 qubits, 30000 days, on neutral atoms.

Oratomic [2603.28627]:

- Modular LDPC architecture with high-rate codes, computation by LDPC surgery.
- ECC-256 in 10000 qubits, 1000 days, on neutral atoms. RSA-2048 in 11000 qubits, 43000 days.

More recent proposals: IonQ Walking Cat [2604.19481], Duke + UTAustin + Yale [2604.19735], QuEra's LDPC STAR [2606.25011], Google's modular hyperbolic architecture [presented at QEC], and more.

Observation 3

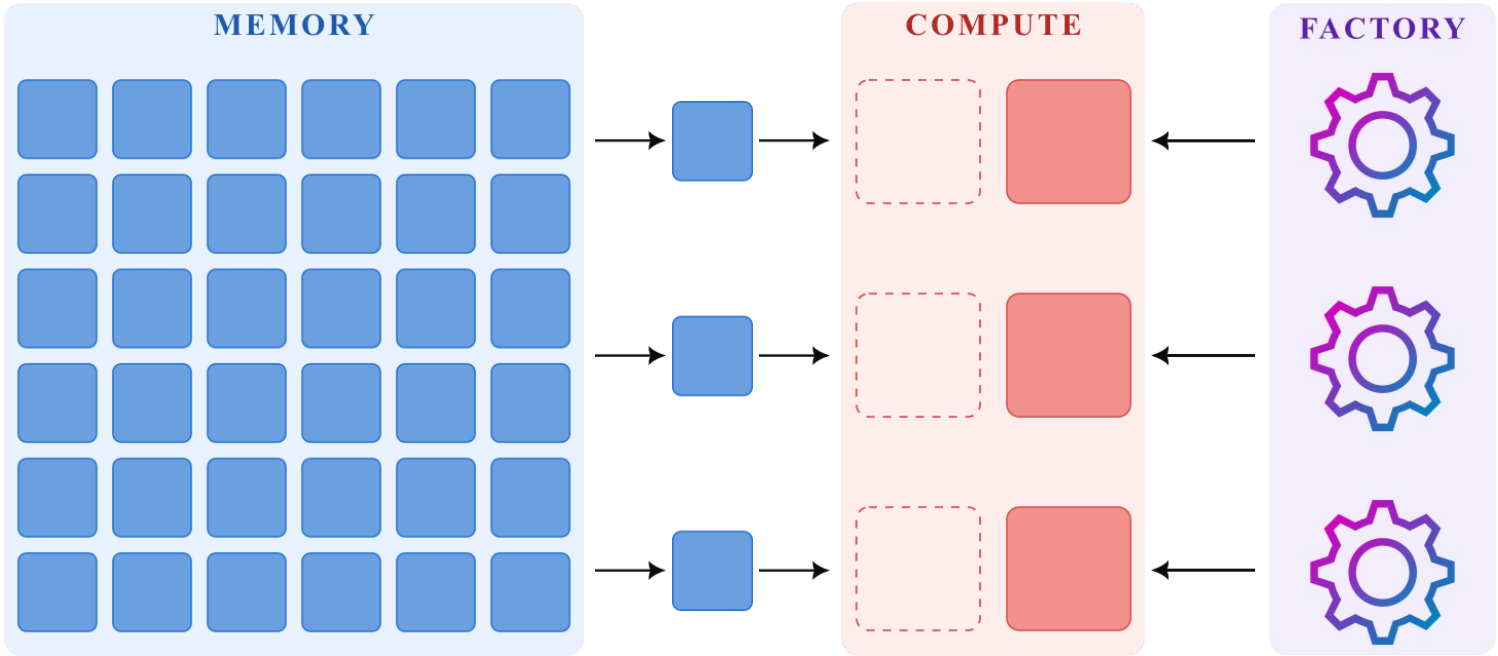
Significant increase in time cost, to the point of infeasible.

Is this fundamental?

- No, not yet.

# Modern QLDPC architectures

# Sketch of current architectures




Memory for compact storage of idling qubits; somethings called ‘cold storage’

Loading

Compute zone for active qubits

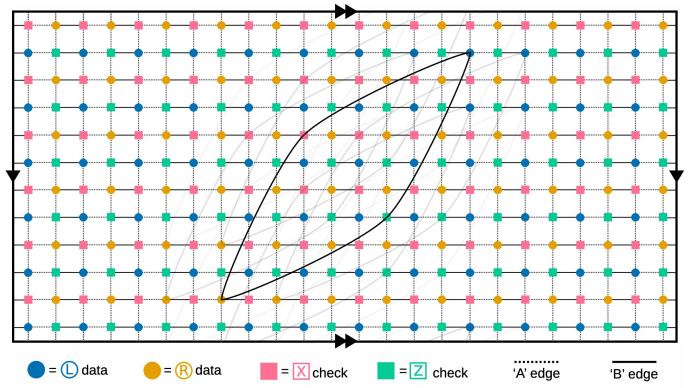
Resource state factories, often producing magic states

# 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

Common memory designs:

- Few big blocks
- Many small blocks
- Concatenated codes [4,5]

Tradeoff among modularity, space-efficiency, and access speed.

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

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

APM codes of rate ½

[1] [IBM 2308.07915], [2] [Xu et al. 2308.08648], [3] [Zhao et al. 2604.16209], [Kasai 2601.08824] [4] [Gidney et al. 2312.04522], [5] [Wills et al. 2605.21898].

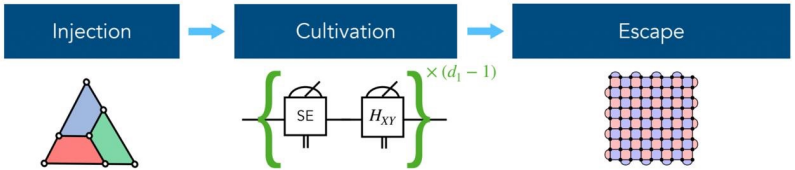
# Resource state factories

Almost all FTQC proposals use resource states, notably magic states for T and CCZ gates.

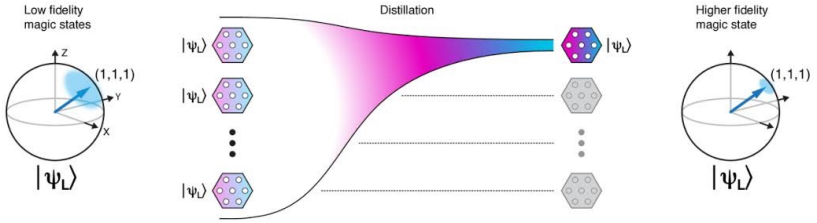
Recent architectures combine two primitives:

- Magic state cultivation: prepare a good ($10^{-5\sim9}$) magic state at low distance, rapidly escape into large distance surface code.
- Magic state distillation: take many magic states at large distance, distill them into fewer states of better quality ($10^{-12}$ and less).
- Typical factory: cultivate first in surface code, load into LDPC codes and distill.

Magic state production is one of the key bottlenecks in current QLDPC architectures! More on that later.




Magic state cultivation[1]




Magic state distillation[2]

[1] [Gidney et al. 2409.17595], figure from [Sahay et al. 2509.05212]. [2] [Bravyi & Kitaev quant-ph/0403025], experiment and figure from [QuEra, 2412.15165]

# Computation machinery

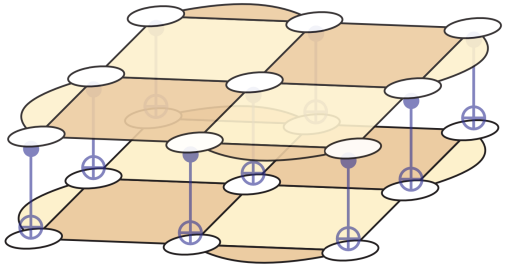
For LDPC codes, we have two main methods of computation:
low-depth unitaries and Pauli measurements.

Low-depth unitaries:

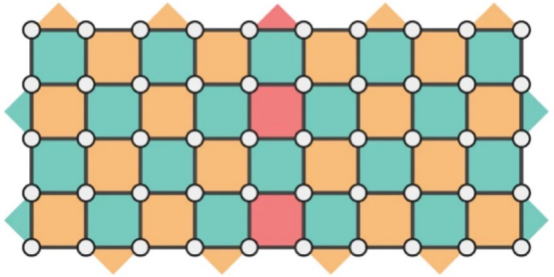
- ➢ Typical example: transversal CNOT
- ➢ Constant-depth physical circuits that implement a logical unitary
- \+ Parallel, fast, no space overhead
- \- Logical control is constrained and often limited
- \- Usually needs qubit reconfiguration

Pauli measurements:

- ➢ Usually implemented by code surgery, sometimes by CAT states
- ➢ Compute by extracting Pauli observables from logical qubits.
- \+ Complete logical control, can work with fixed connectivity
- \- Usually slower, logical control is sequential in many cases



Transversal CNOT on surface code[1]



Lattice surgery[2]

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

# Why surgery now?

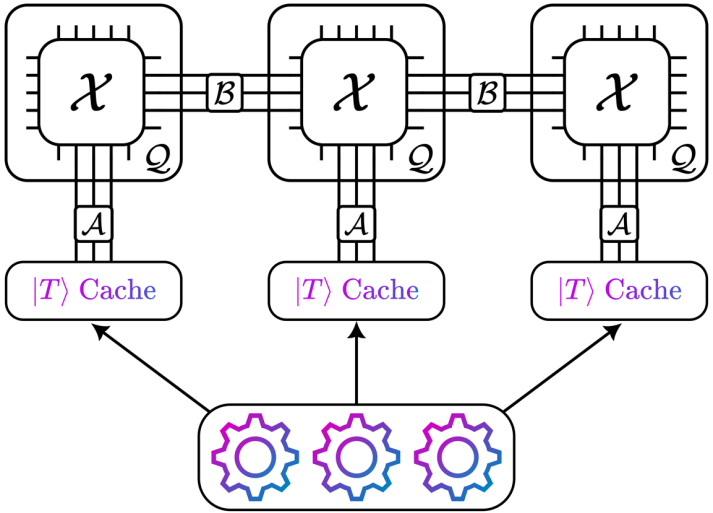
LDPC surgery [1-6] is a new technique that has advanced rapidly.

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]

# If FTQC is like printing…

Suppose you have two different printing presses:

- One that prints 5 words at a time, any words of your choice;
- Another that prints 100 copies of one chosen word at a time;

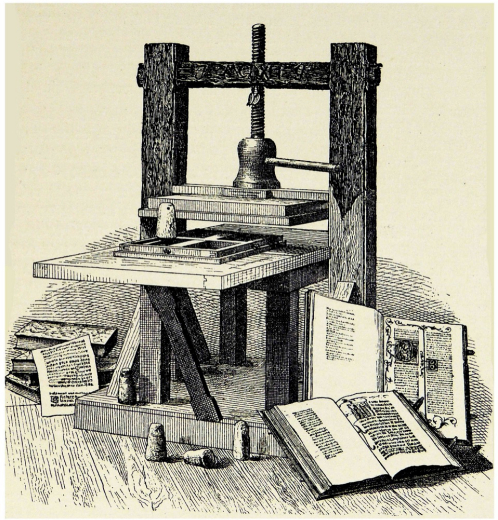
When resource is limited and you really want to print one specific essay, use the first press

- This is sequential surgery, which was studied in [1, 2] and adopted in recent QLDPC architectures;

When you want to print many copies of the same essay, use the second press

- This is batched, parallel computation via transversal gates[3]

* Surgery can give parallel, structured logical control[4] as well, more on that later.



[1] [Litinski 1808.02892], [2] [He et al. 2503.10390]. [3] [Xu et al. 2510.06159] [4] [Chang et al. 2603.02157]

# Code surgery via graphs and chain maps

# Universal computation via logical measurements

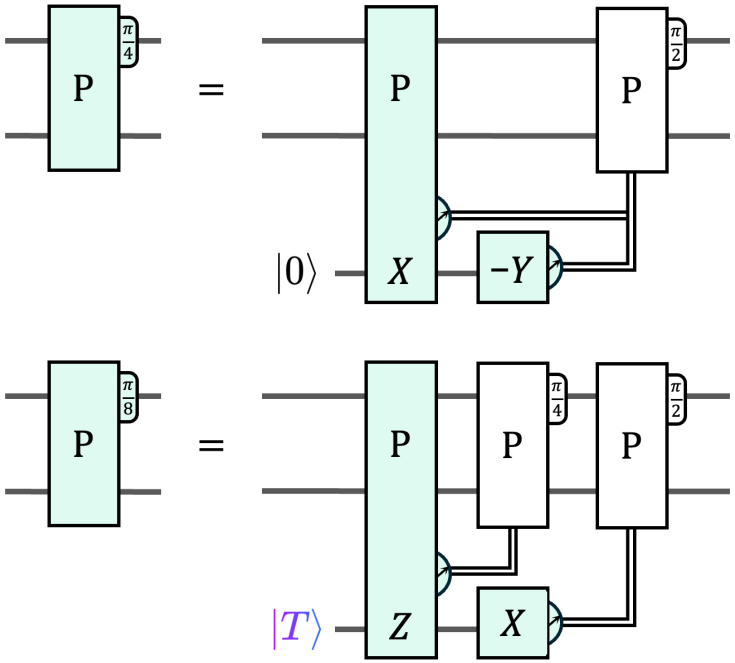
A Clifford + T circuit can be compiled into two primitives:

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

This is called Pauli-based computation.

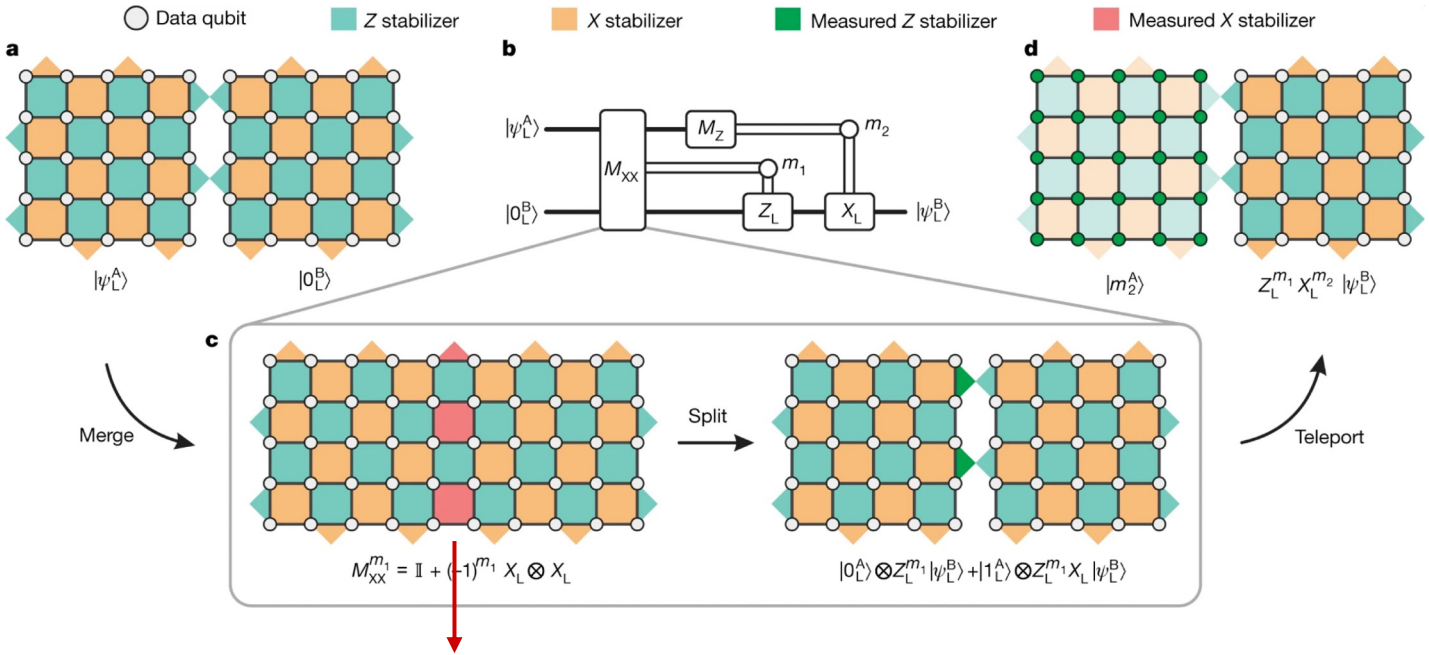
Code surgery is a way to measure logical Pauli observables fault-tolerantly.

- 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

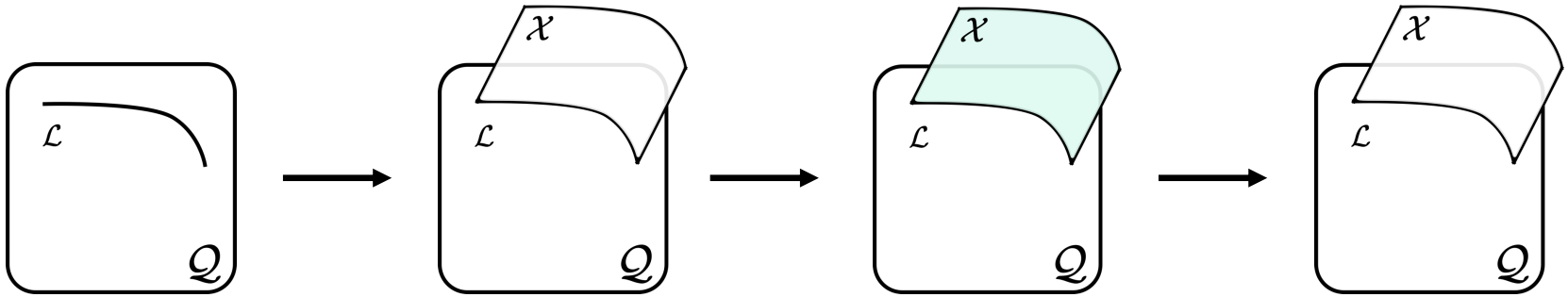
Logical measurements on surface codes: lattice surgery, [Horsman et al, 1111.4022].




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 $\mathcal{Q}$.

# Code surgery via graphs and chain maps

## Graph formalism

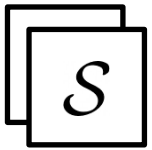
# Scalable Tanner Graphs

# Scalable Tanner Graphs


Code $\mathcal{Q}$

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

# Scalable Tanner Graphs

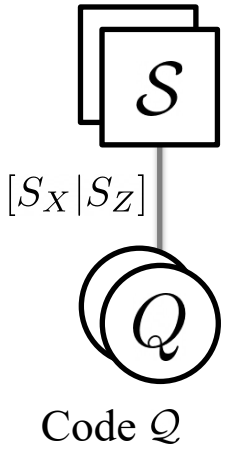



Stabilizers of the code $\mathcal{Q}$


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

Code $\mathcal{Q}$

# Scalable Tanner Graphs

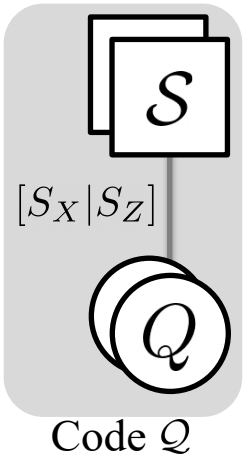



Stabilizers of the code $\mathcal{Q}$

Symplectic check matrix

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

# Scalable Tanner Graphs

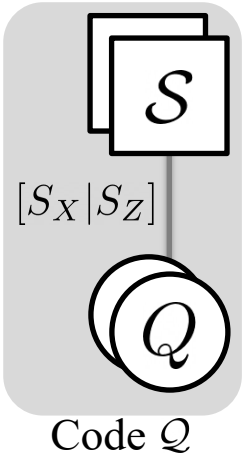



Stabilizers of the code $\mathcal{Q}$

Symplectic check matrix

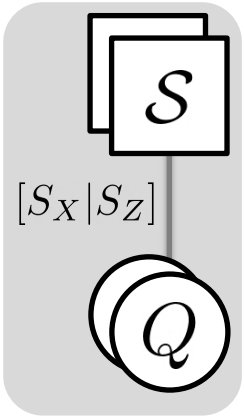
Physical qubits of the code $\mathcal{Q}$

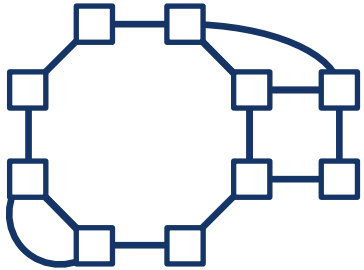
# Building Ancilla Systems from (Hyper)Graphs



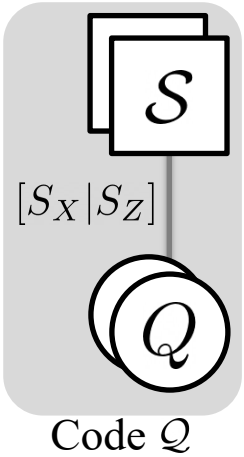
# Building Ancilla Systems from (Hyper)Graphs

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

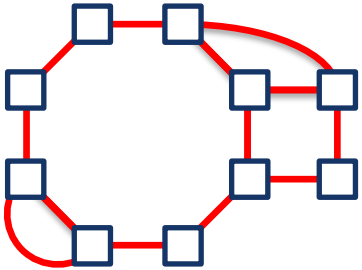


# Building Ancilla Systems from (Hyper)Graphs



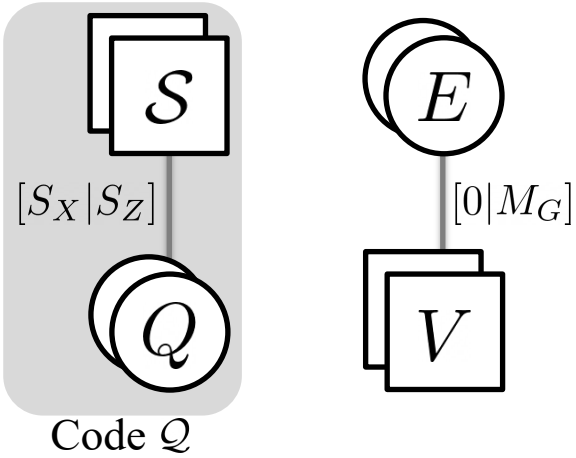
Code $\mathcal{Q}$

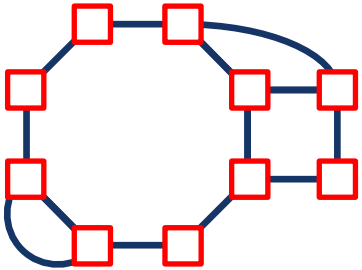



Let G = (V, E) be a (hyper)graph.

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

# Building Ancilla Systems from (Hyper)Graphs



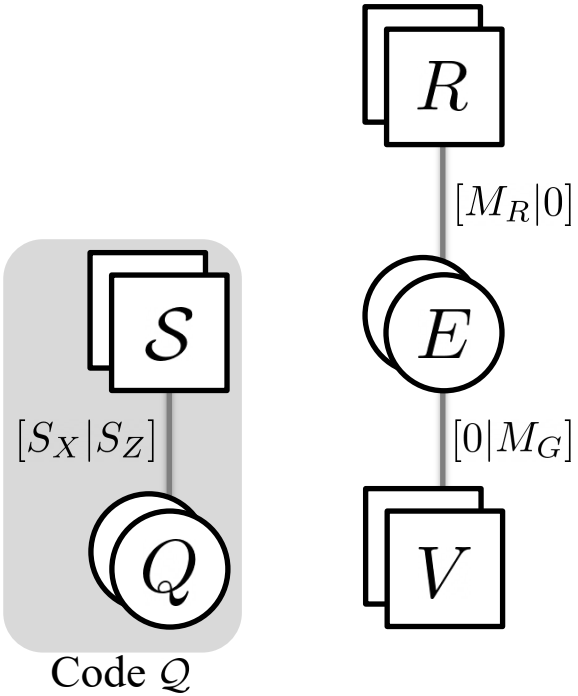



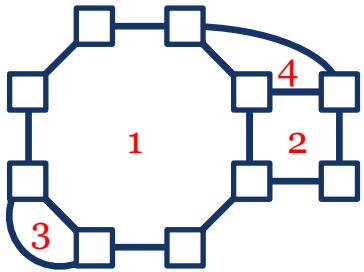
Let G = (V, E) be a (hyper)graph.

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

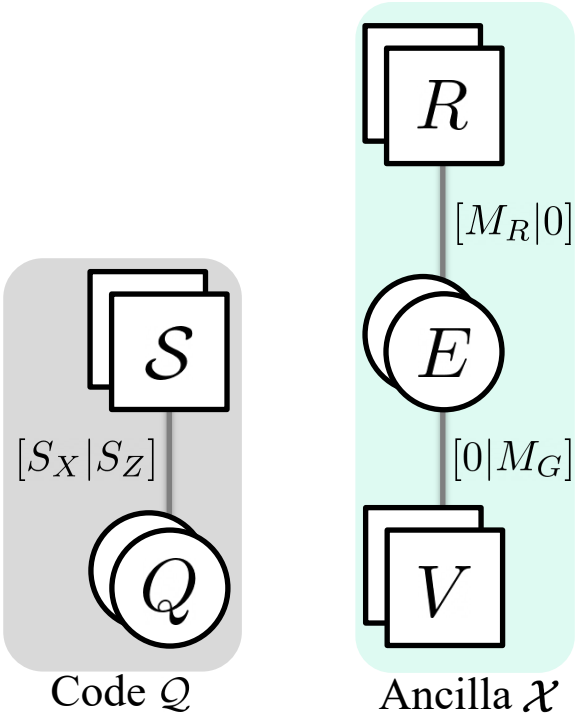


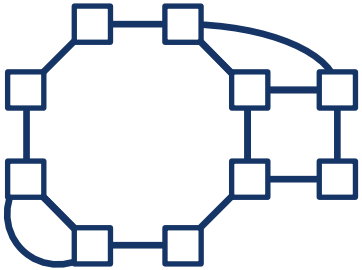




Let G = (V, E) be a (hyper)graph.

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

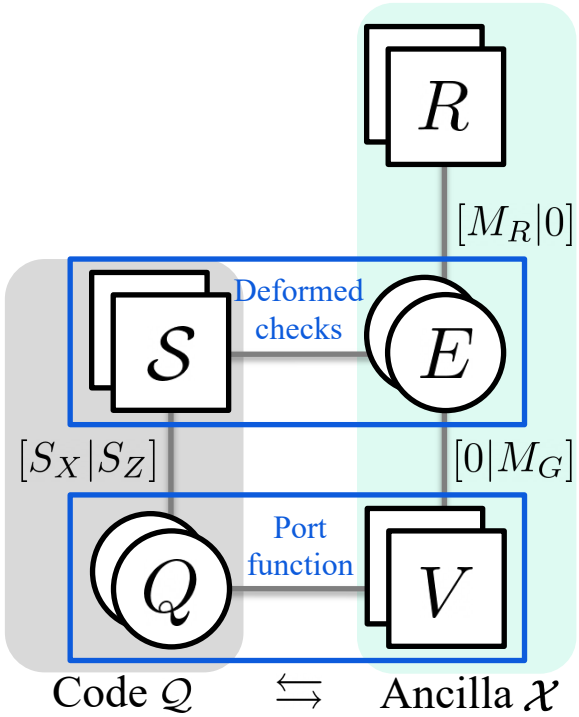


Let G = (V, E) be a (hyper)graph.

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:

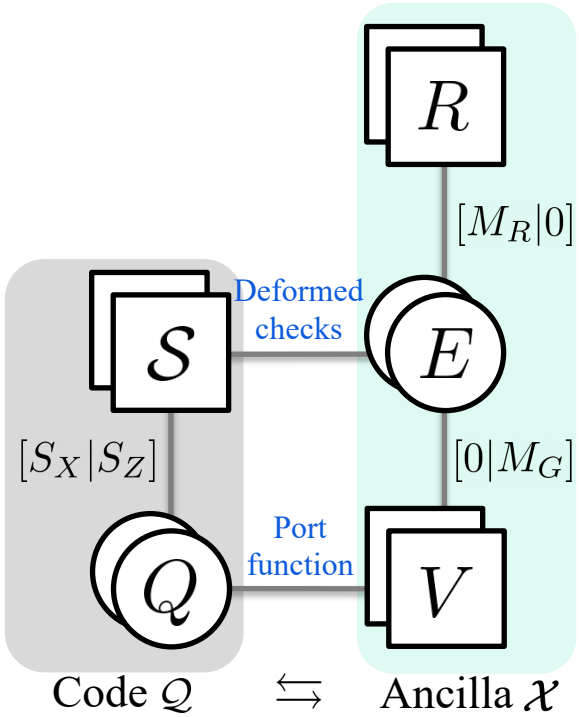
1. Measurement (hyper)graph G;
2. Port function: Pauli actions of vertex checks V on qubits of $\mathcal{Q}$; Defined so that product of vertex checks in a connected component equals $\mathcal{L}$.
3. Deformed checks: Pauli actions of stabilizers S on ancilla edge qubits E.

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

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].

# Parallel Surgery via Hypergraphs




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

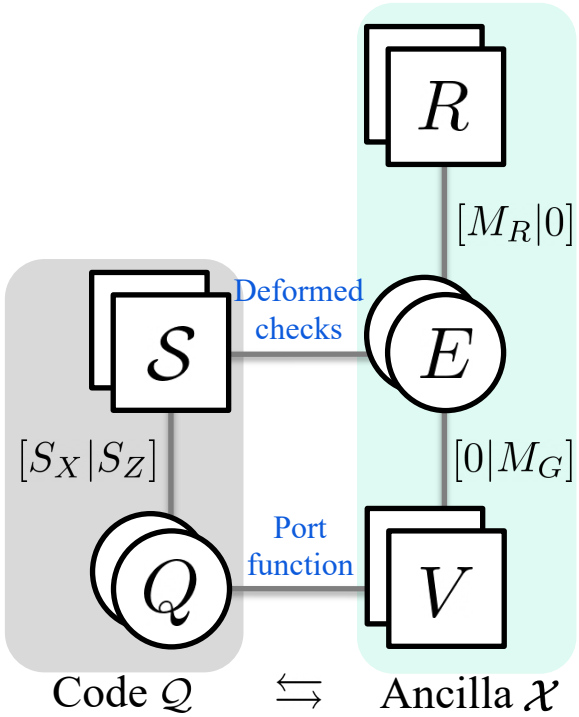
If G is a connected simple graph:

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

If G is a hypergraph:

- Multiple sets, implements parallel measurements.

# 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)

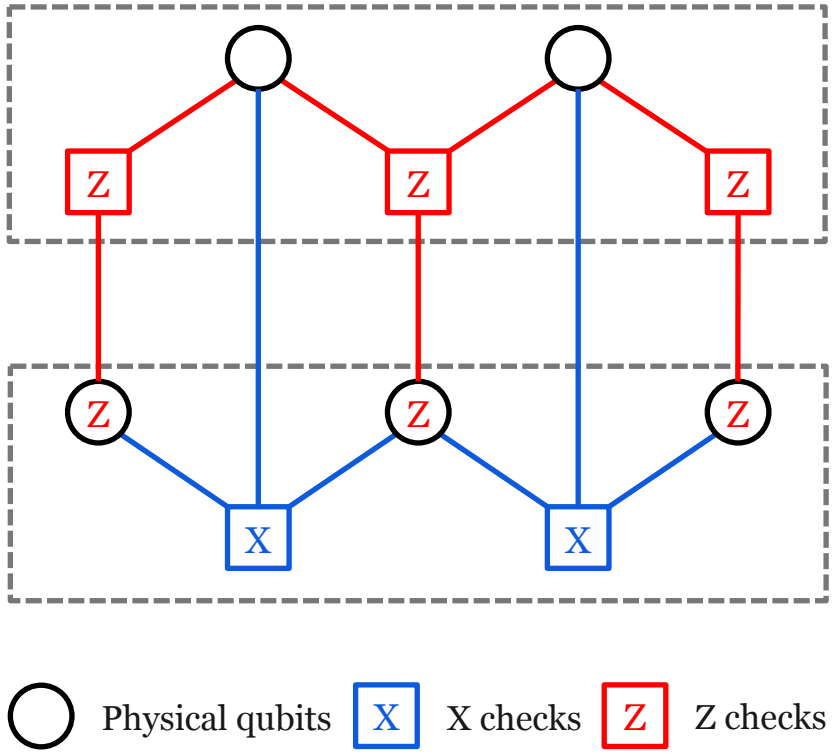
# Example

Ancilla system

Port function (1-to-1)

Check deformation:
done with ‘path matching’

Local code structure, with
Z logical operator to be measured



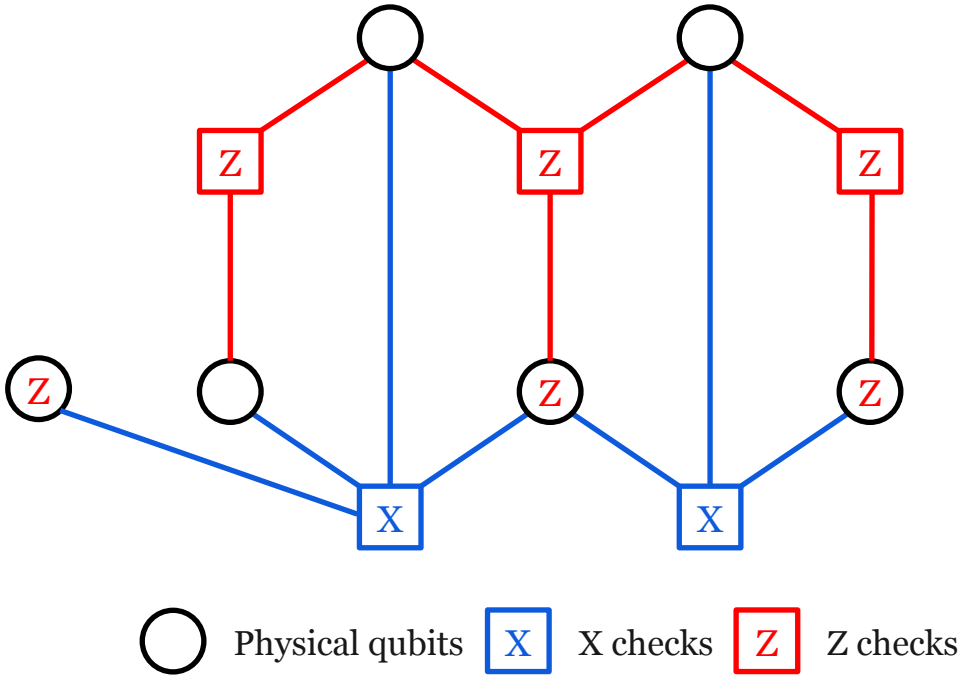
# What can go wrong?

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



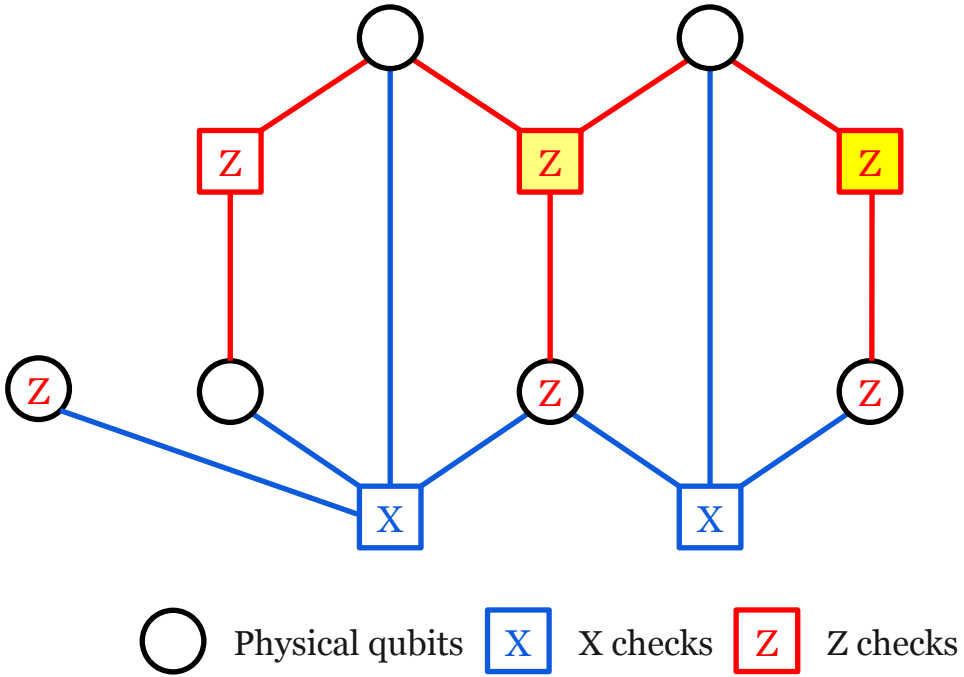
# What can go wrong?

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.

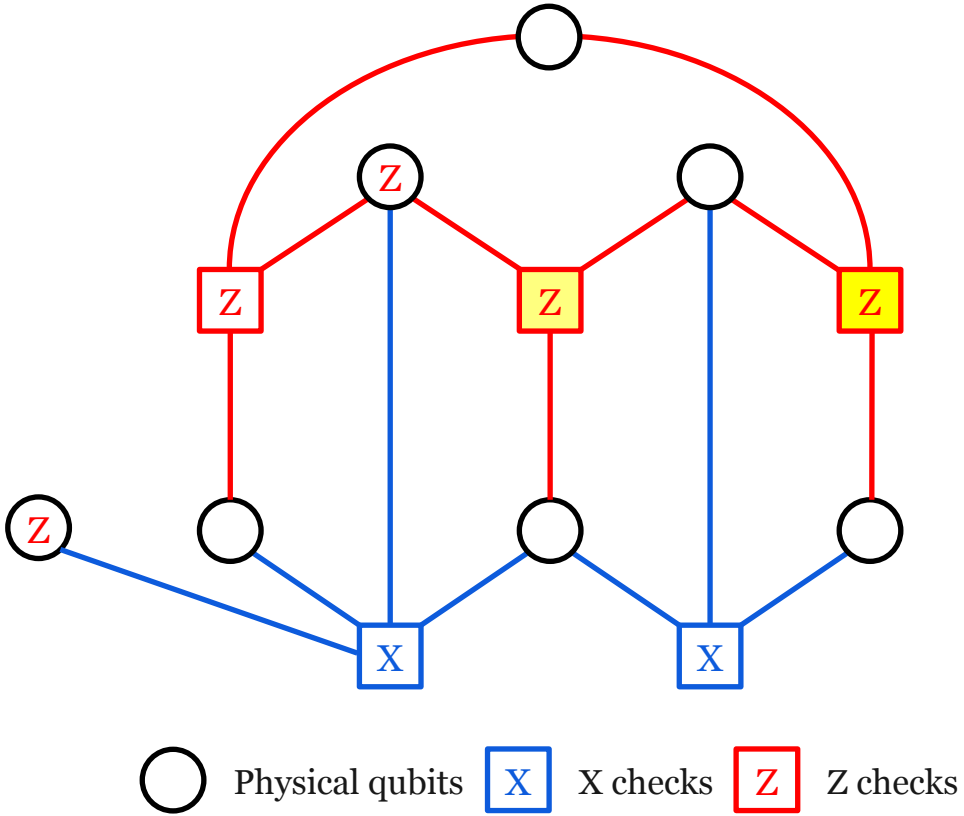
Its distance can be reduced in two ways:

- 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:

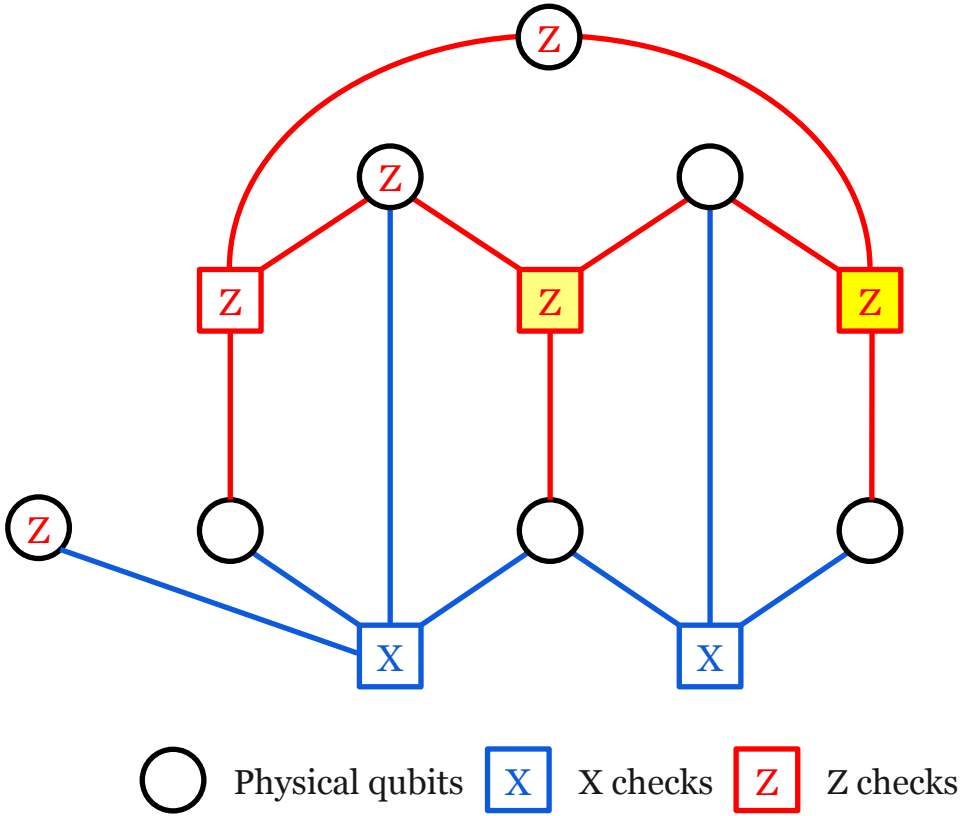
- Cleaning: multiply by new stabilizers
- Dressing: multiply by new logicals

Let's clean with two highlighted checks

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

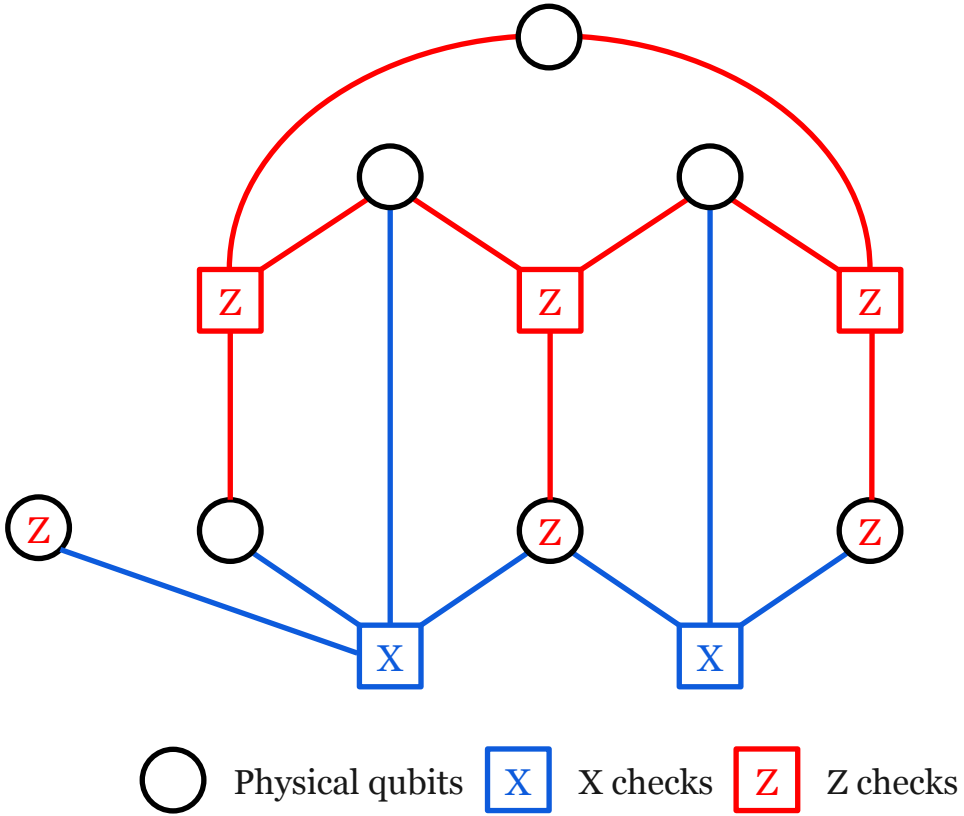
Solution: Expansion.

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



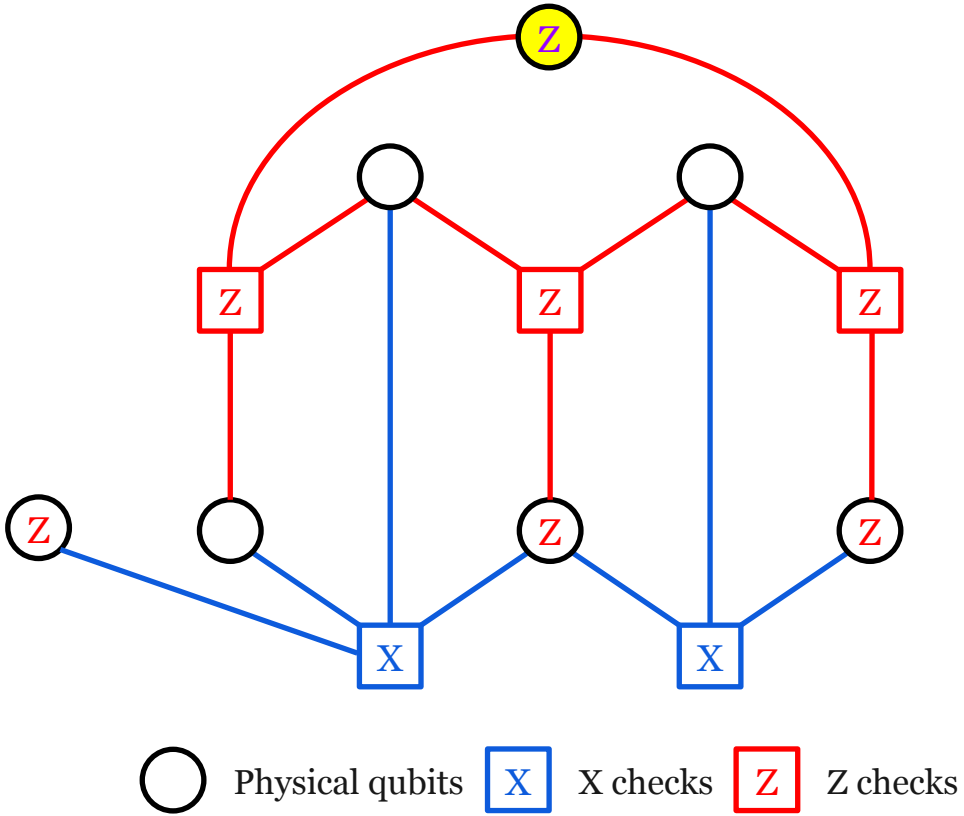
# What can go wrong (again)?

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



# What can go wrong (again)?

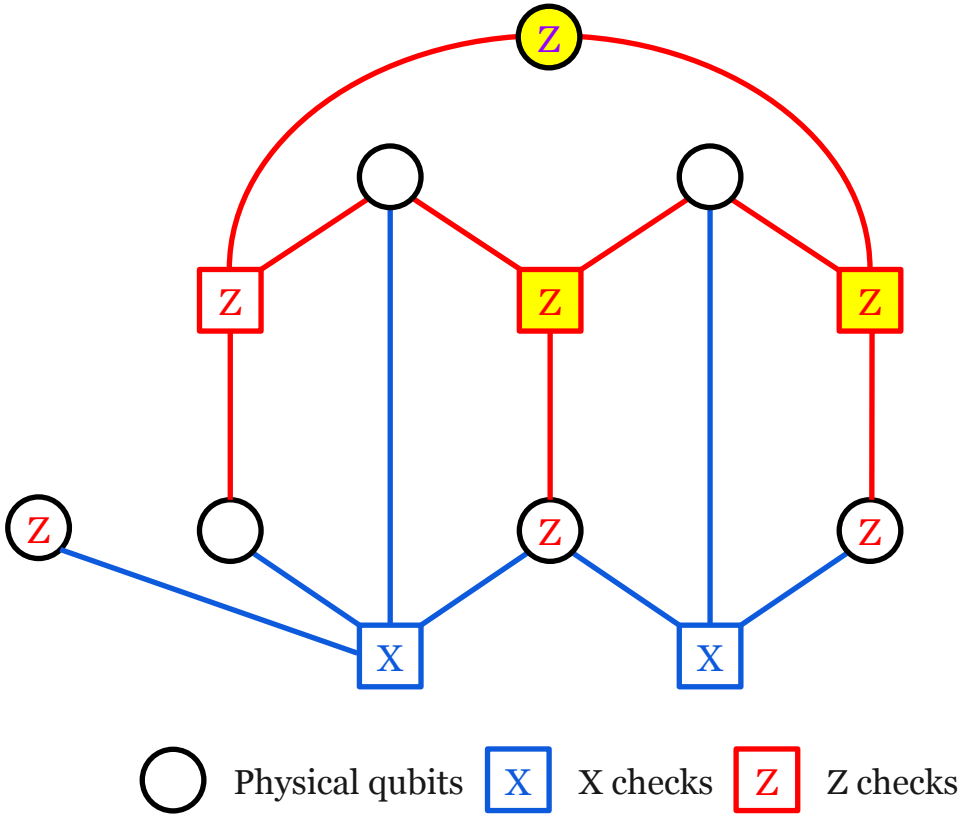
Now consider dressing. When we introduce more ancilla qubits than ancilla checks, we have new ‘gauge’ operators.



# 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)?

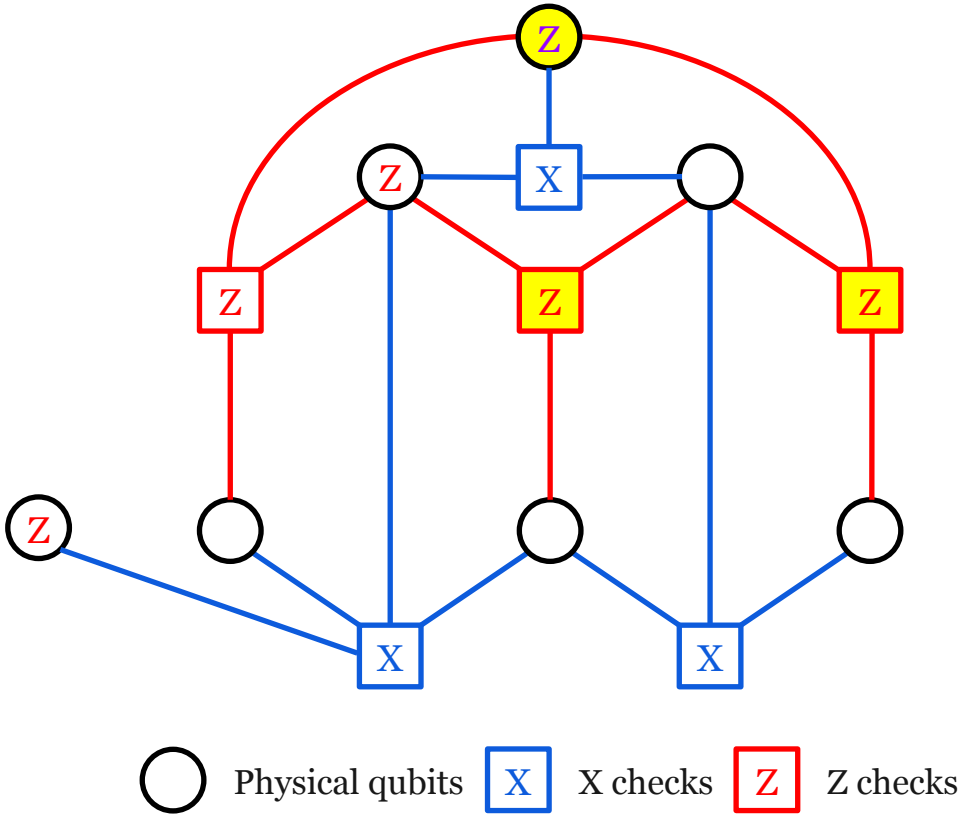
Now consider dressing. When we introduce more ancilla qubits than ancilla checks, we have new 'gauge' operators.

Multiply by the highlighted checks & logicals:

- Operator weight reduced 3 → 2 again!

Solution: measure all cycle as checks in opposite basis.

- No more new logicals → space distance preserved.



# What can go wrong (again)?

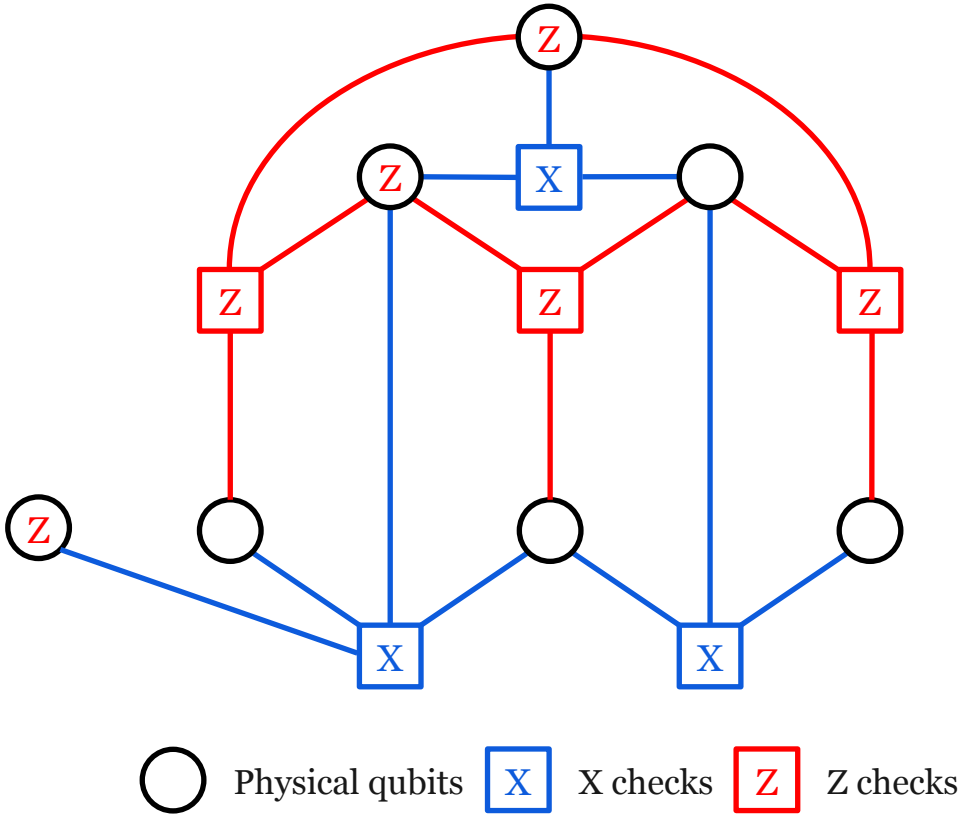
Now consider dressing. When we introduce more ancilla qubits than ancilla checks, we have new ‘gauge’ operators.

Multiply by the highlighted checks & logicals:

- Operator weight reduced 3 → 2 again!

Solution: measure all cycle as checks in opposite basis.

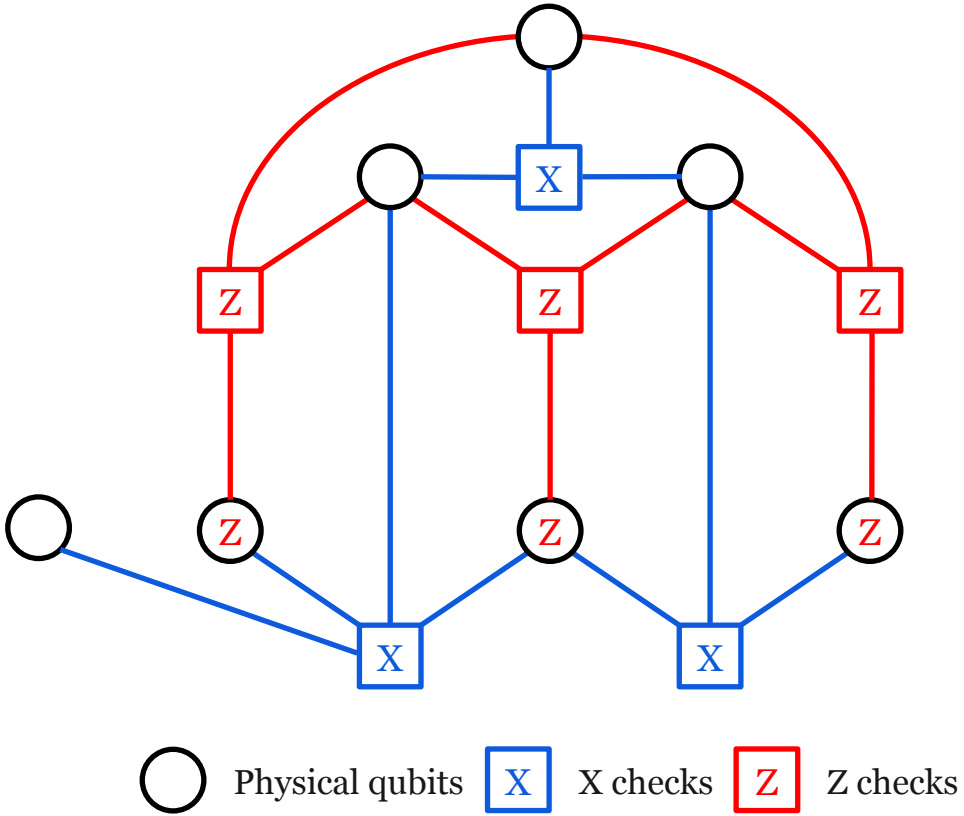
- No more new logical qubits → space distance preserved.



# Summary: how to build graphs for surgery

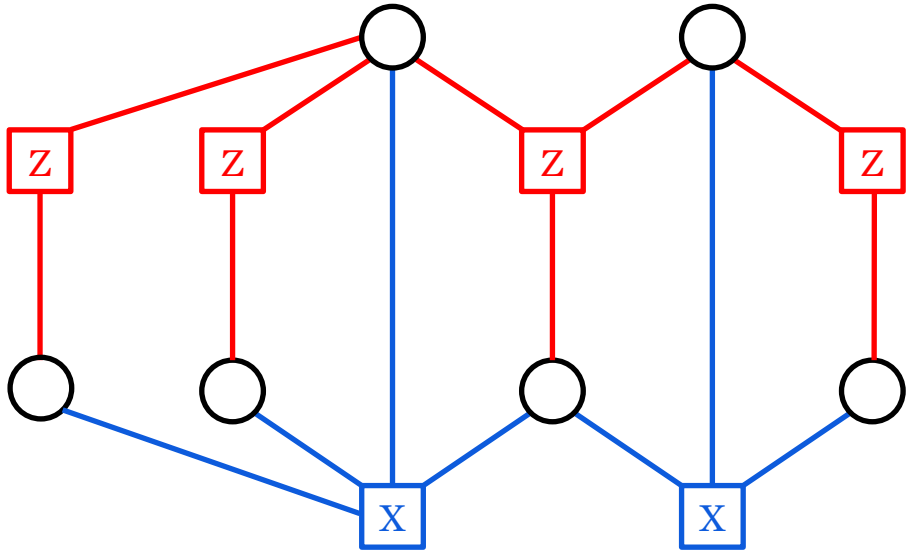
One common way to build a surgery graph for a logical operator L:

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’), measure all cycles



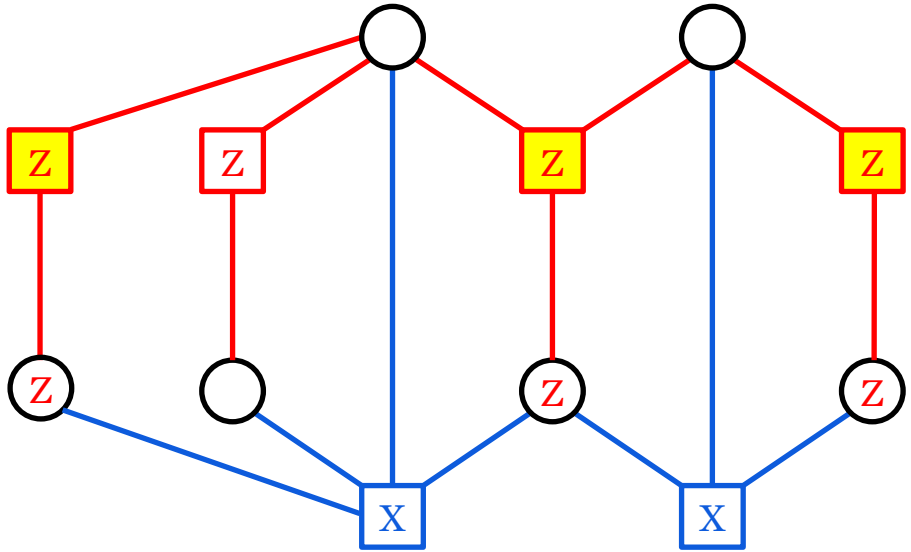
# Parallel Measurements via Hypergraphs

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



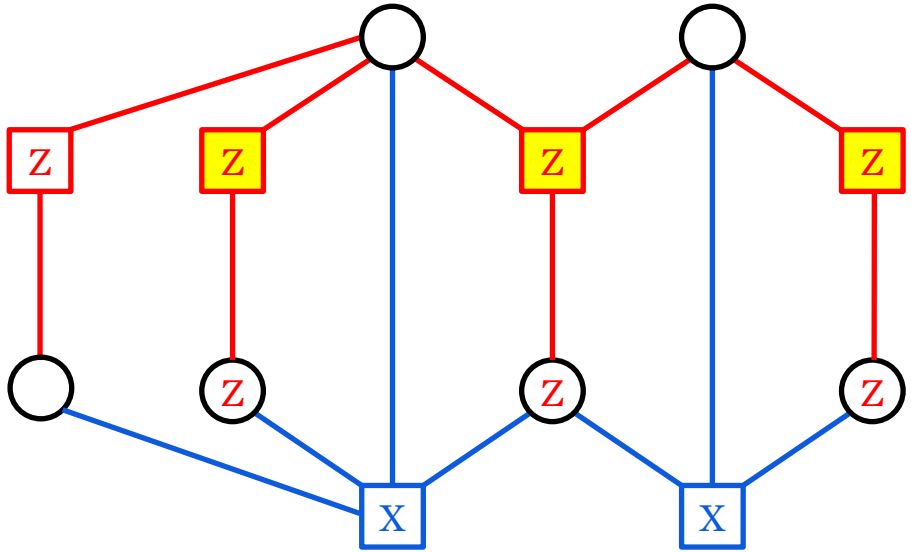
# Parallel Measurements via Hypergraphs

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.



# Code surgery via graphs and chain maps

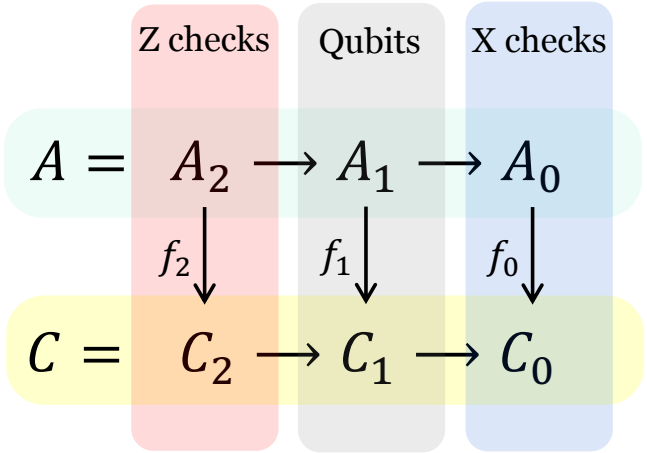
## Chain map formalism

# Chain Complex and Maps

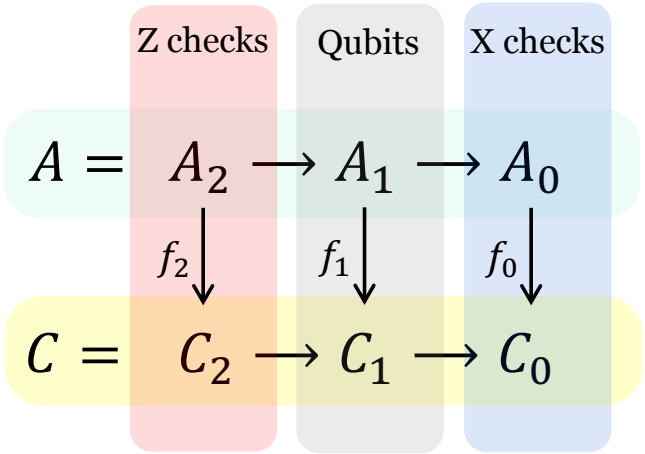
A CSS code is a chain complex. We can consider an ancilla code/complex, and maps between two complexs.

The diagram commutes → valid chain map.

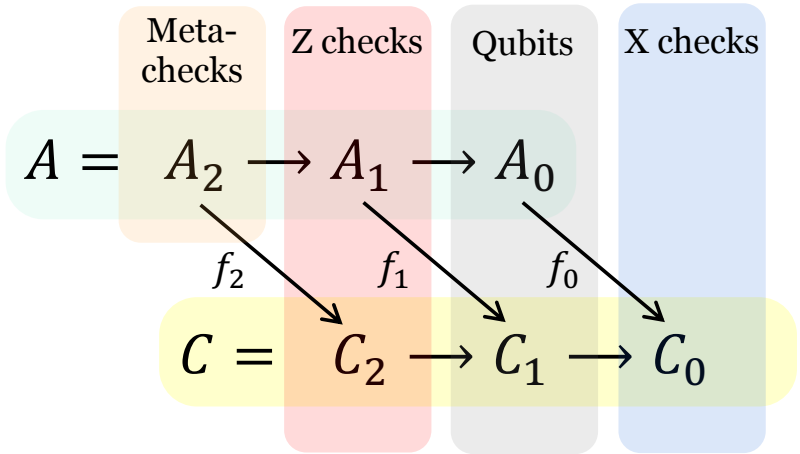
- Can be used to describe transversal CNOTs, homomorphic measurements (Steane EC).
- These maps can be trivial (0).



# Mapping Cone



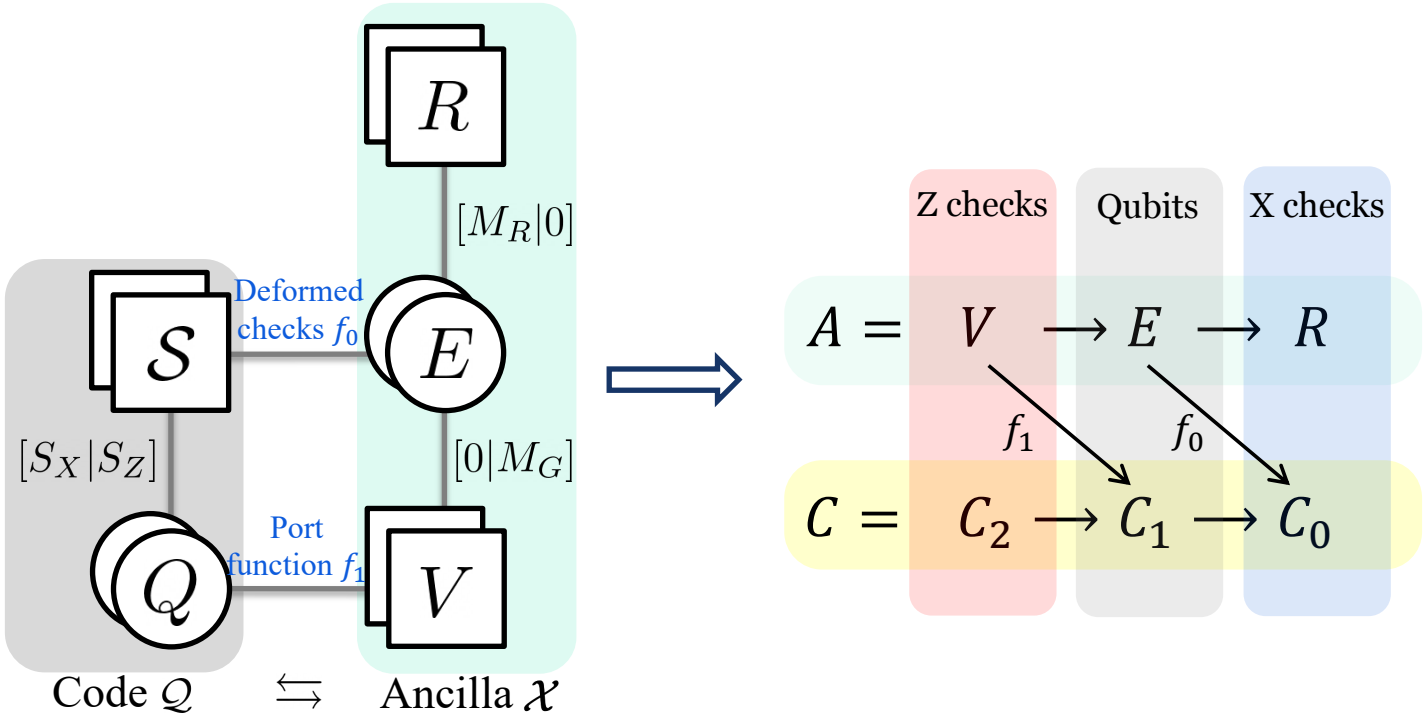

Chain map




Mapping cone

Shift one complex by one degree. Changes operational meaning!

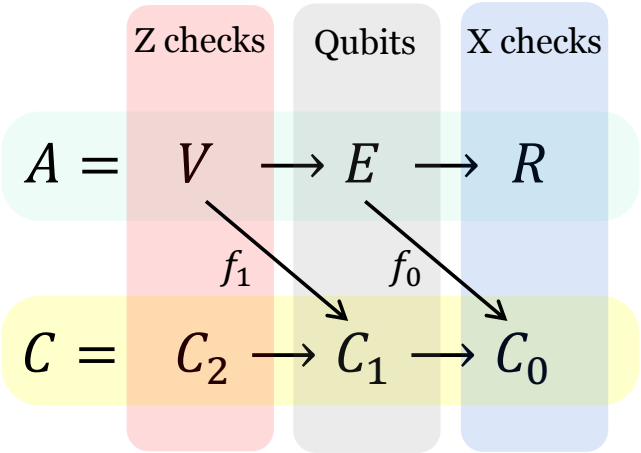
# Surgery as Mapping Cones



# Pros and Cons

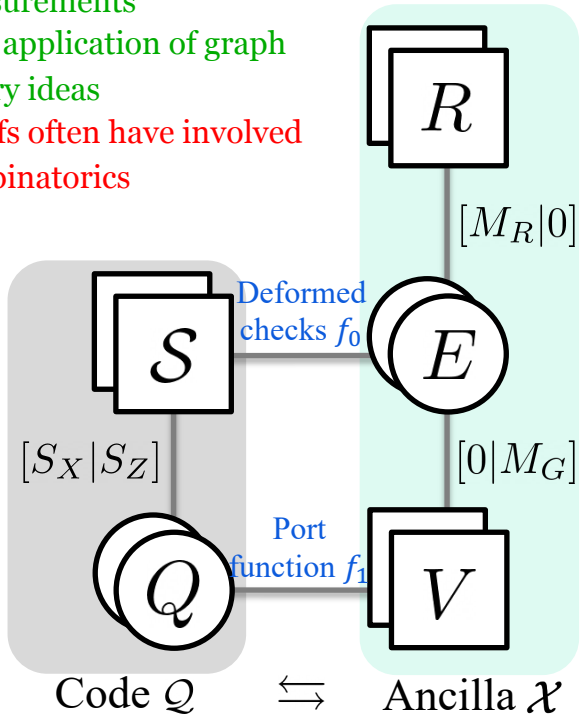
Chain map formalism:

+ 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



# Surgery-based architectures

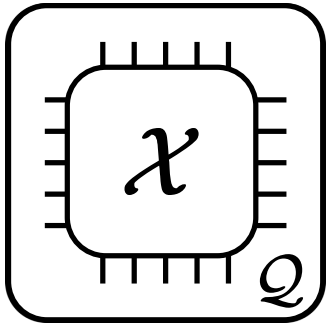
# LDPC surgery via Extractors

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.

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*.

- Extractor size ranges between $\tilde{O}(d) \sim \tilde{O}(n)$.

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], [Oratomic 2603.28627]

**8~16 Operators**
[Cross et al. 2407.18393], [IBM 2506.03094], reaches hundreds by automorphisms

**16 Operators**
[Iceberg 2602.11457], reaches $4^k$ by qubit movement

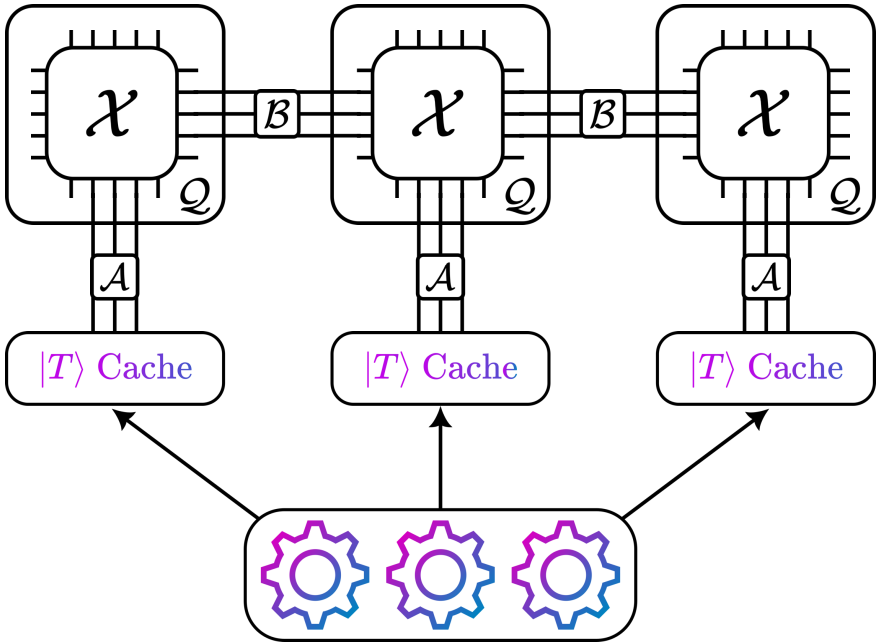
**$4^k$ Operators**
[He et al. 2503.10390], [Blue et al. 2606.03507] , [Han et al. 2607.28605], [Bhardwaj et al. 2607.28795]

# of operators covered by extractors

# 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[1,2]

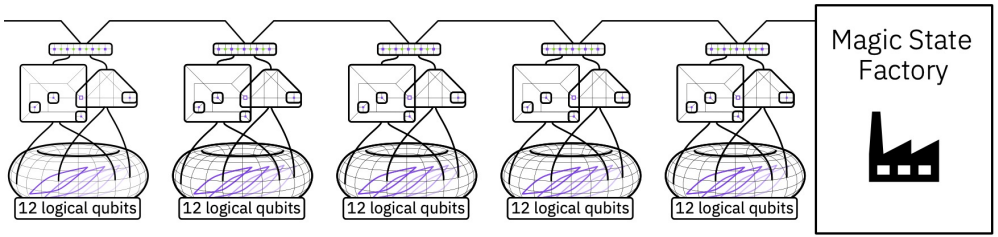



[1] [Swaroop et al. 2410.03628], [2] [Cross et al. 2407.18393]

# Design choices of recent architectures

Similarity: sequential logic

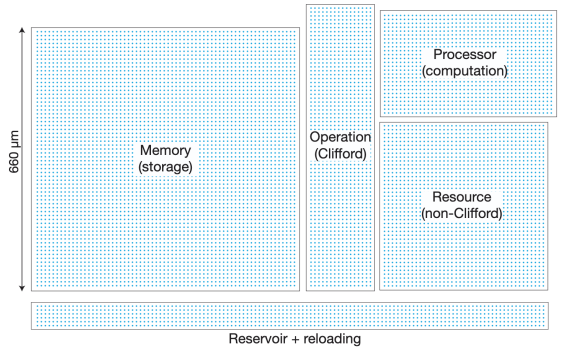
- Every CCZ (or T) gate in the circuit is compiled into one (large) PPM. Many (or all) Clifford gates are compiled away.
- Magic state throughput is one/logical cycle.



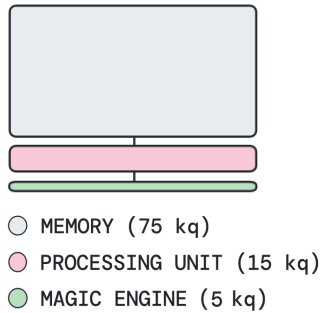

IBM's bicycle architecture[1]

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.

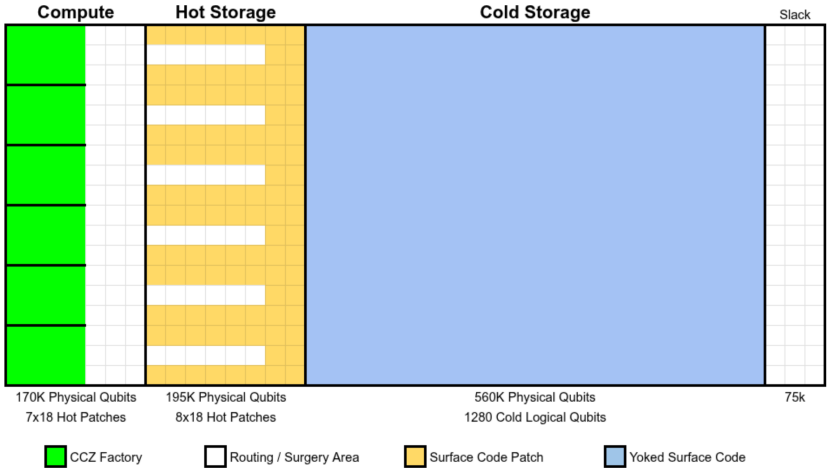
Compute: Can use smaller processor codes, or compute directly on memory.

- 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 $\mu s$. Factor of 1000!

- Takes Gidney's 5 days to 5000 days.

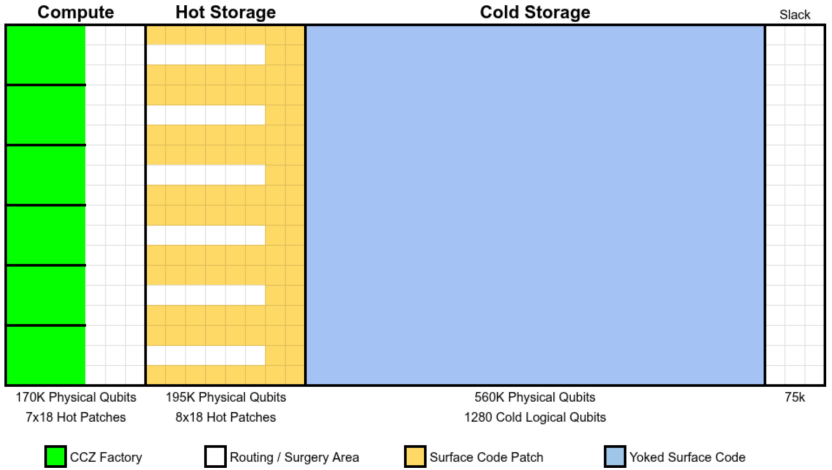
Compilation:

- 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.

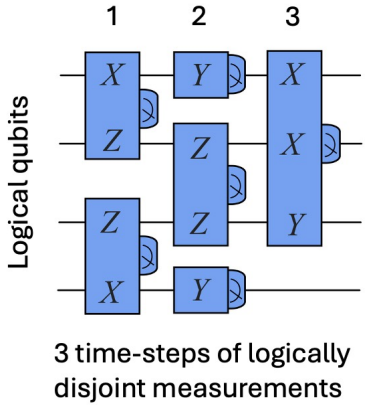
# Further techniques in surgery

# Parallel Surgery

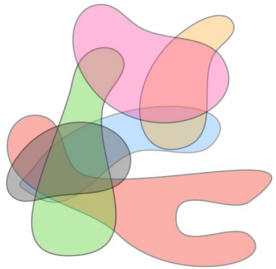
Goal: measure a collection of commuting logical Pauli operators simultaneously.

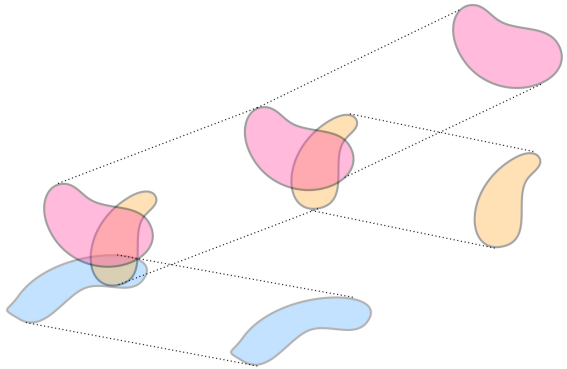
Existing techniques:

- 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]




3 time-steps of logically disjoint measurements





[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].

# Parallel Surgery

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

Existing 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].

# Fast (and Parallel) Surgery

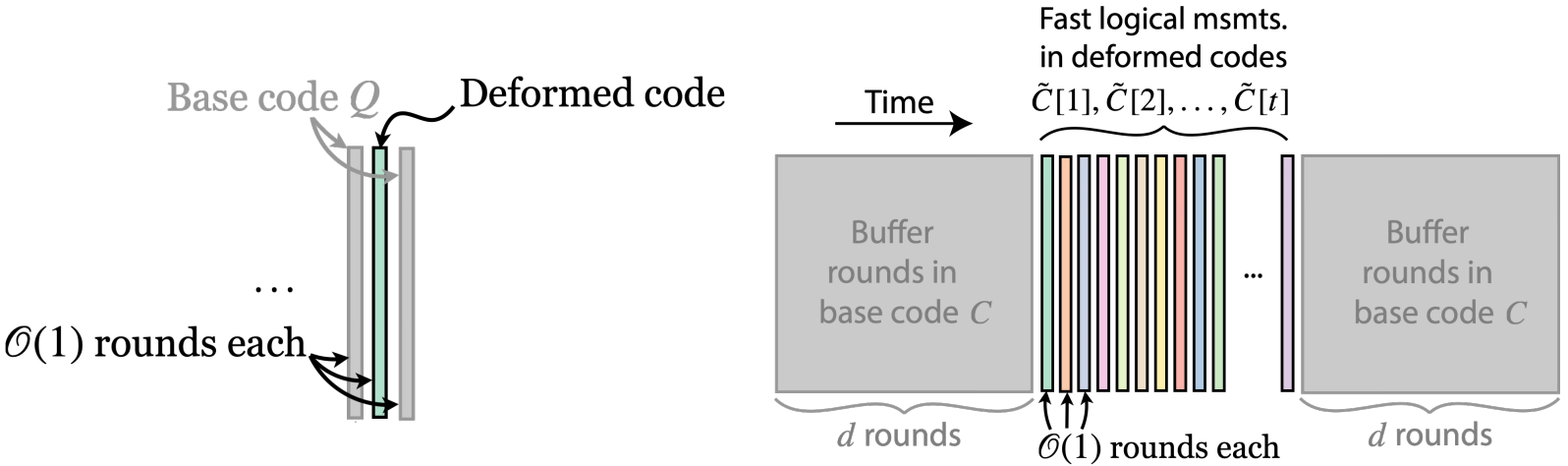
Two formulations: single-shot surgery and constant-time surgery.

Single-shot surgery [1-3]:

- Requires code to admit single-shot state preparation, such as 3D HGP codes.

Constant-time surgery [4,5]:

- Does not require single-shot state preparation. Akin to algorithmic fault tolerance.

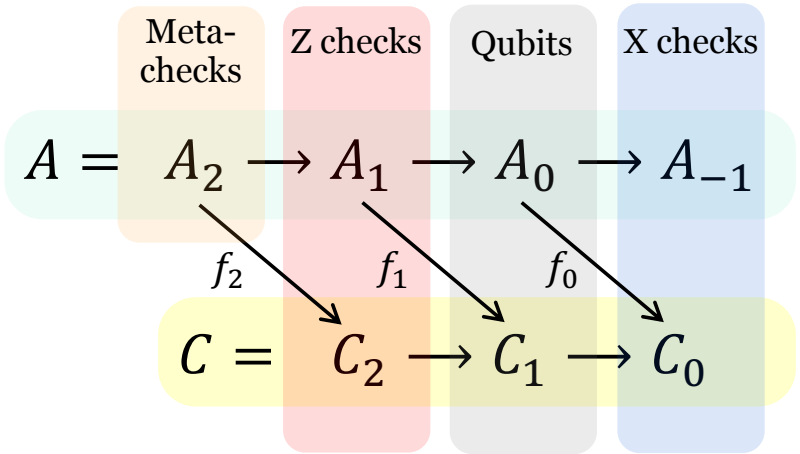



[1] [Hillman et al. 2410.12963], [2] [Tan et al. 2510.08552], [3] [Baspin et al. 2510.04521]. [4] [Cowtan et al. 2510.14895], [5] [Chang et al. 2603.02157].

# Fast (and Parallel) Surgery

Both single-shot surgery and constant-time surgery requires meta-checks.

- Meta-checks protect against measurement errors in vertex checks
- 4-term complex is a hypergraph → naturally gives parallel measurements.



# Chronicles of QLDPC Surgery

Early weight reduction/surgery
- [H 2102.10030]
- [CKBB 2110.10794]

Modern graph surgery formulation:
- [C**H**RY 2407.18393]
- [WY 2410.02213]
- [IGND 2410.02753]
- [SJY 2410.03628]

Parallel surgery via hypergraphs:
- [ZL 2408.01339]
- [C**H**WY 2503.05003]
- [ZJX 2510.08523]

More works which we didn't include here. The field is growing rapidly.

Extractor Architectures:
- [**H**CWY 2503.10390]
- [IBM 2506.03094]
- [Iceberg 2602.11457]
- [Oratomic 2603.28627]

Fast Surgery (single-shot)
- [HDTV 2410.12963]
- [THLGH 2510.08552]
- [BBC 2510.04521]

Fast Surgery (constant-time)
- [C**H**WY 2510.14895]
- [C**H**YZJ 2603.02157]

New weight reduction:
- [HLL 2510.09601]
- [YC**H**LW 2603.05082]

Code-tailored Surgery/Extractors:
- [GLQXER 2603.05398]
- [B**H**ZC 2606.03507]
- [ZZJX 2607.28605]
- [BM+ 2607.28795]

Clifford measurements:
- [DBMWWB 2503.15751]
- [CLKS 2602.12228]
- More work upcoming.

Applications:
- [ZZYL 2505.06981]
- [YZL 2506.18061]
- [ZZL 2510.19442]
- [XLR**H**D 2602.20546]

# 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 extractors:

- Can we design constant-space-overhead full extractors on some family of codes? (The space-overhead is log(n) right now)
- Can we co-design code and extractors?

Clifford measurements:

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

Hypergraph sparsification:

- Can we sparsify a hypergraph (cycle basis and degree)?

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

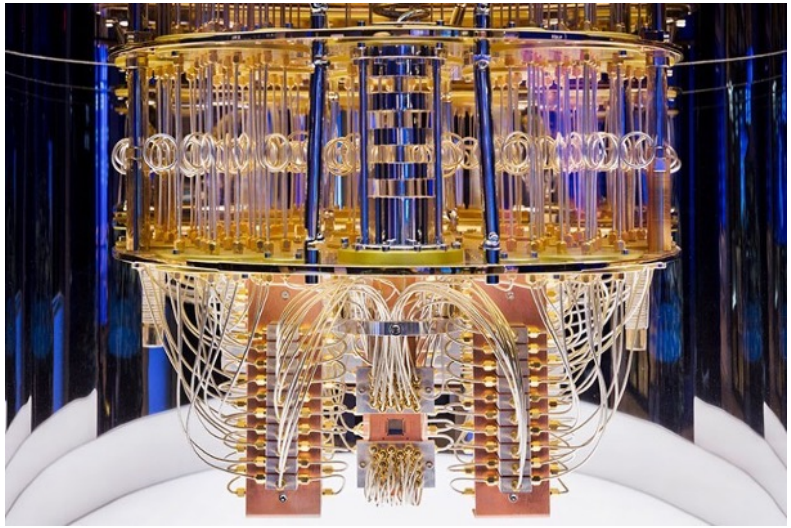
- Algorithms and their subroutines often structured. Can we design the logical control (surgery or otherwise) with respect to their structure?[3]

### Practical design of (parallel/fast/full) extractors

- Surgery is very versatile, lots of opportunity for practical/finite-scale studies.

### 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]

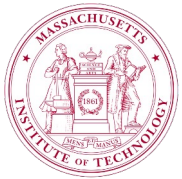


[1] [Breuckmann et al. 2410.16250], [2] [Manon et al. 2508.10714] [3] [QuEra 2606.25011] [4] [Pattison & Nguyen 2411.03632]

# Pauli-Based Computation on QLDPC Codes

## A tutorial on surgery and surgery-based architectures

Sunny Zhiyang He @ Benasque, August 2026




Slides