Fast or Furious surgery: New primitives for QLDPC computation

FTQT 2026 @ Benasque, Spain

Katie Chang

Yale Quantum Institute, Yale University

Joint works with:

[2603.02157] KC, Zhiyang He, Guanyu Zhu, Theodore Yoder, Tomas Jochym-O'Connor

[in prep.] KC, Anasuya Lyons, Yuanjie Ren, Harald Putterman, Victor Albert, Ben Brown, Nathaniel Tantivasadakarn, Dominic Williamson

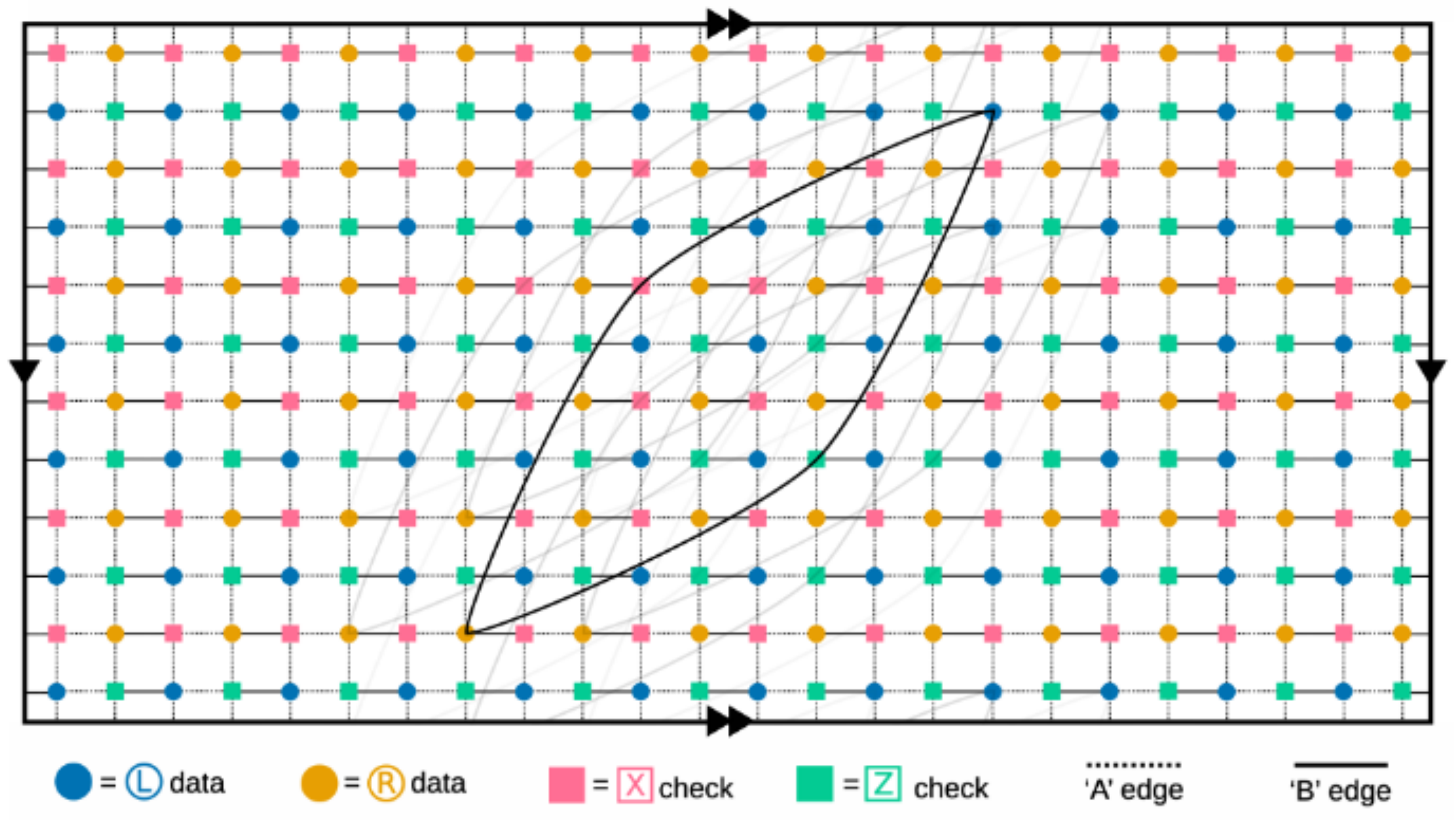
QLDPC Codes

QLDPC Codes

QLDPC codes enable fault-tolerant memories at low space overhead

QLDPC Codes

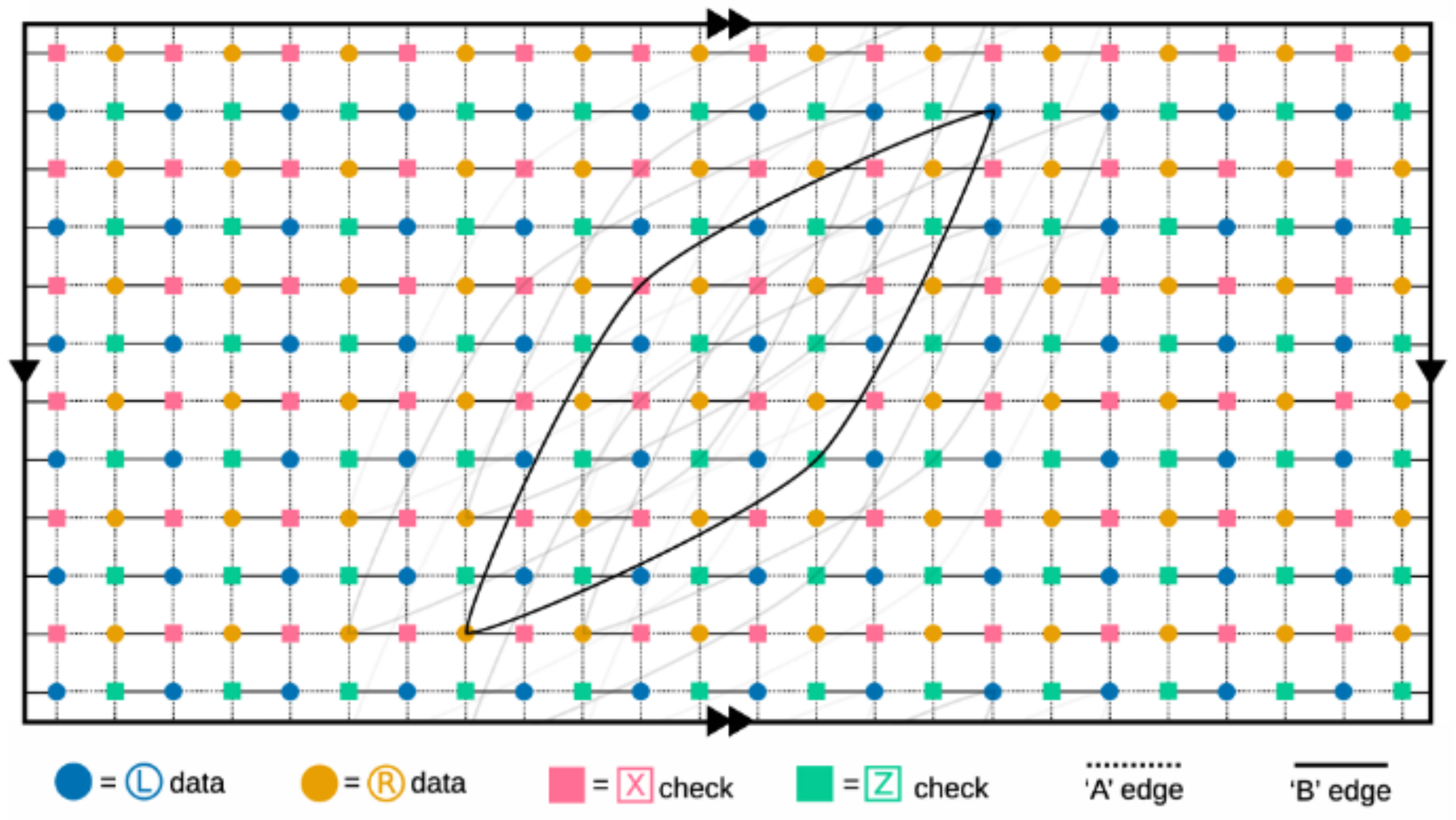
QLDPC codes enable fault-tolerant memories at low space overhead



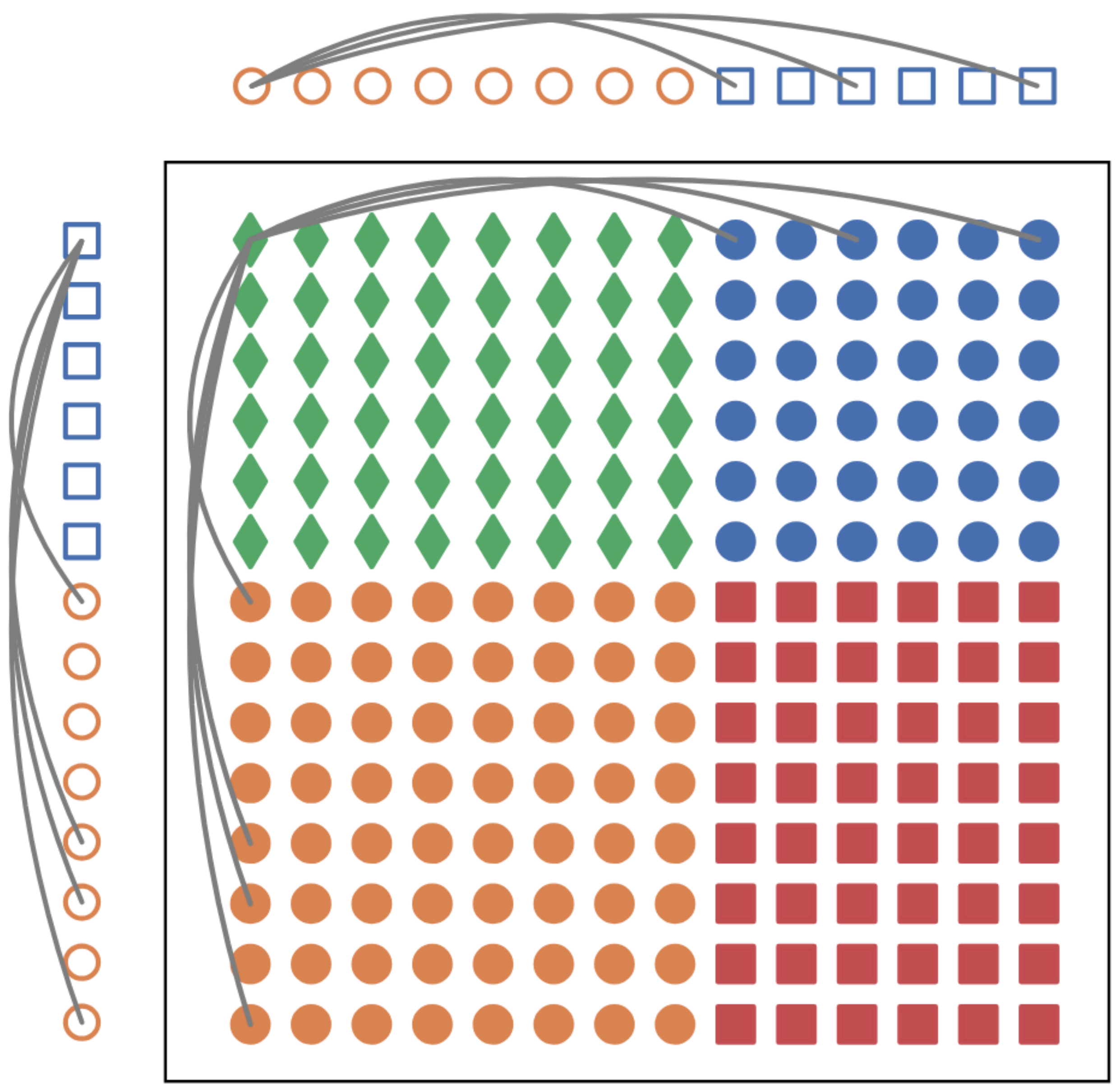
Bivariate Bicycle Code

QLDPC Codes

QLDPC codes enable fault-tolerant memories at low space overhead



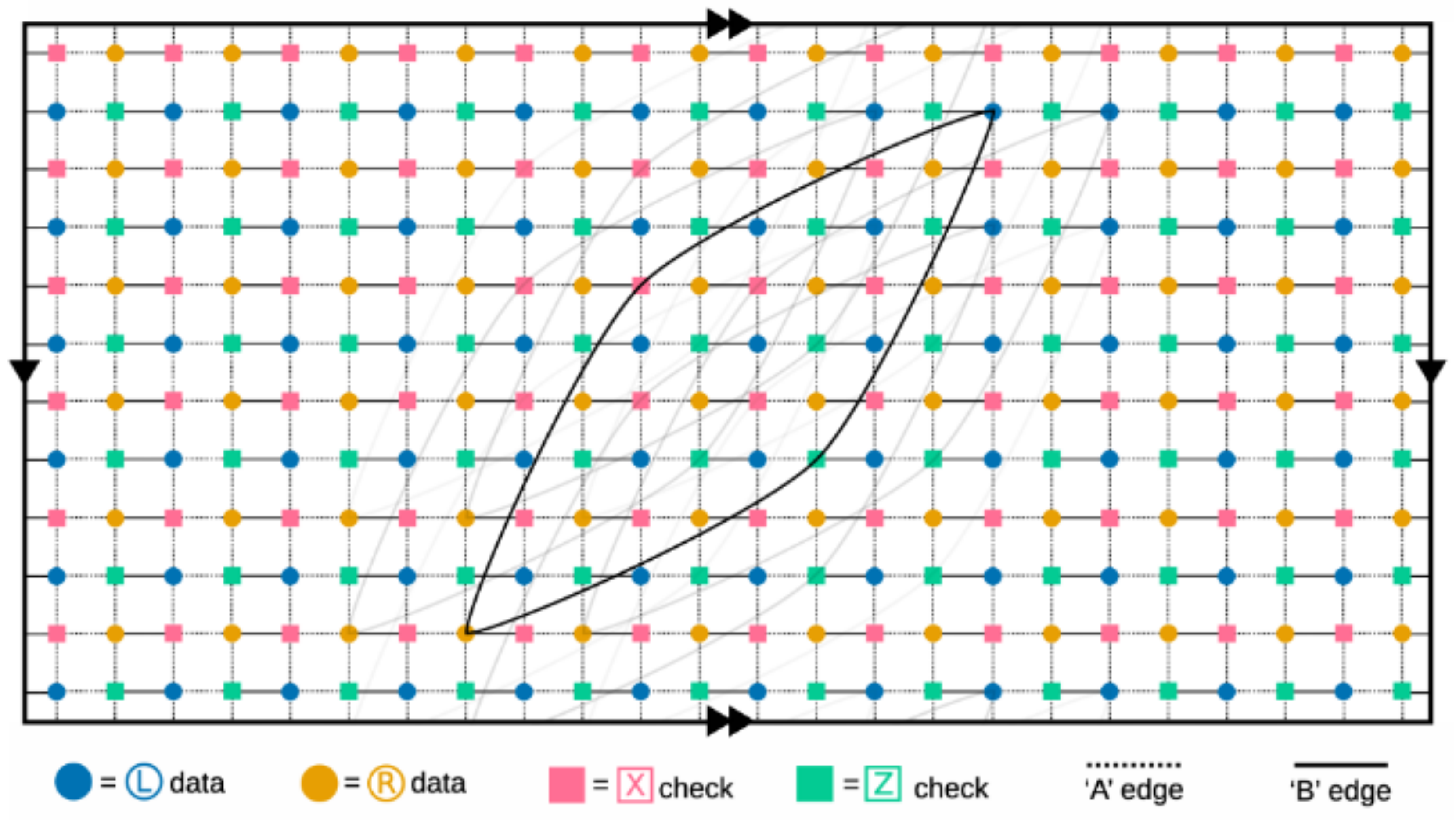
Bivariate Bicycle Code



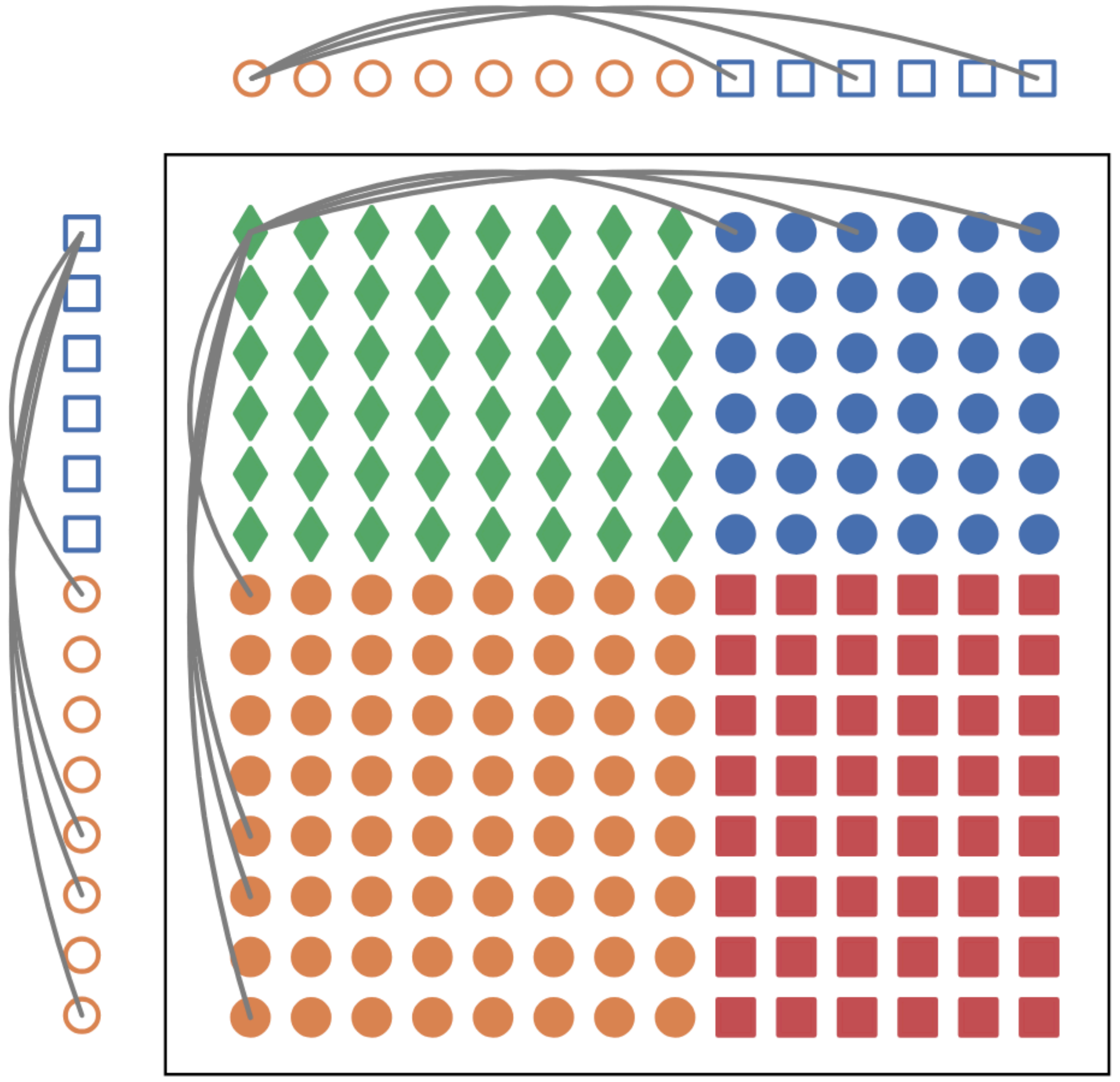
Hypergraph Product Code

QLDPC Codes

QLDPC codes enable fault-tolerant memories at low space overhead



Bivariate Bicycle Code



Hypergraph Product Code

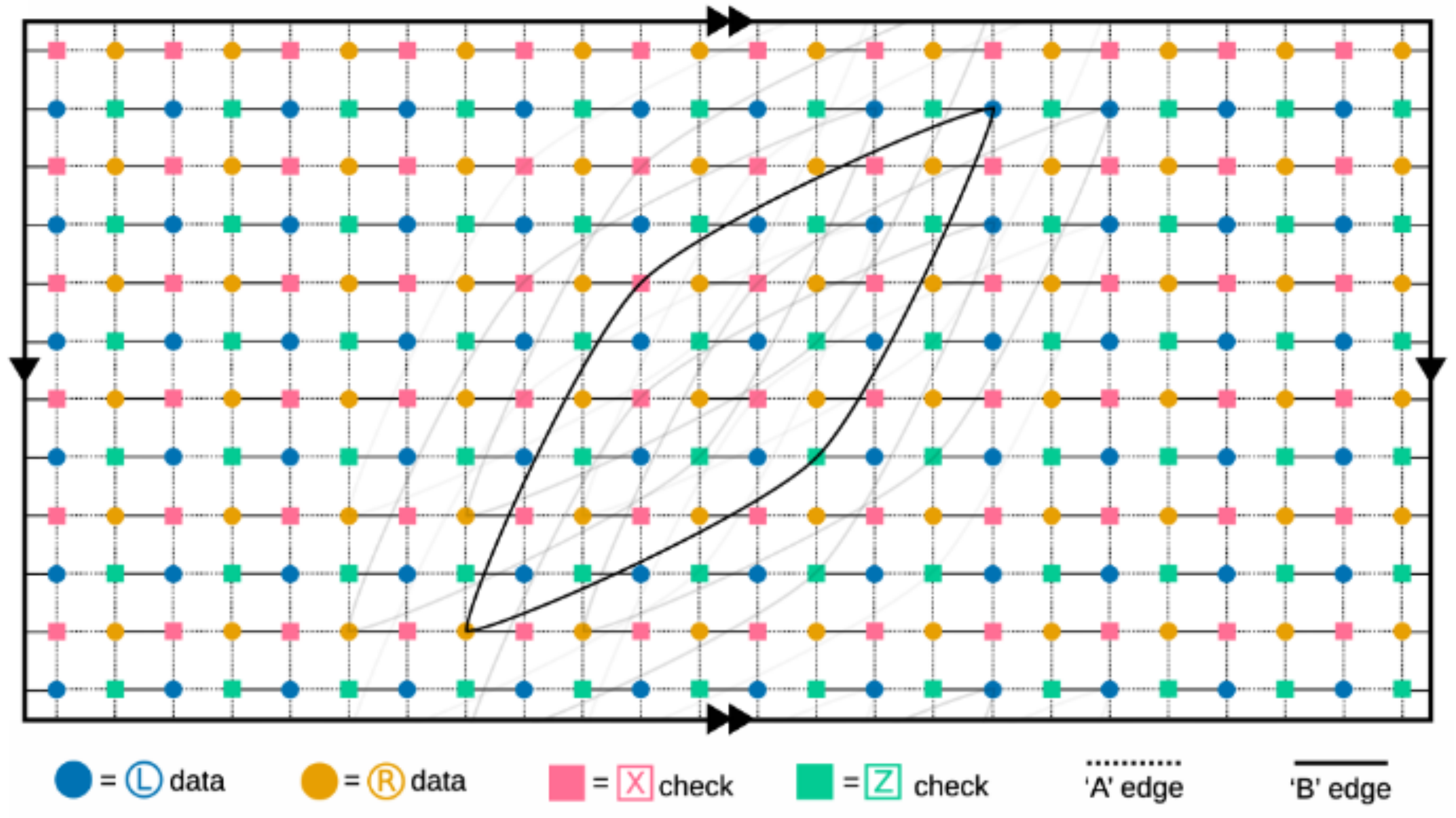
$$H_X = \left(\begin{array}{cccccc|cccccc} F_0 & F_1 & F_2 & F_3 & F_4 & F_5 & G_0 & G_1 & G_2 & G_3 & G_4 & G_5 \\ F_5 & F_0 & F_1 & F_2 & F_3 & F_4 & G_5 & G_0 & G_1 & G_2 & G_3 & G_4 \\ F_4 & F_5 & F_0 & F_1 & F_2 & F_3 & G_4 & G_5 & G_0 & G_1 & G_2 & G_3 \end{array}\right)$$

$$H_Z = \left(\begin{array}{cccccc|cccccc} G_0^T & G_5^T & G_4^T & G_3^T & G_2^T & G_1^T & F_0^T & F_5^T & F_4^T & F_3^T & F_2^T & F_1^T \\ G_1^T & G_0^T & G_5^T & G_4^T & G_3^T & G_2^T & F_1^T & F_0^T & F_5^T & F_4^T & F_3^T & F_2^T \\ G_2^T & G_1^T & G_0^T & G_5^T & G_4^T & G_3^T & F_2^T & F_1^T & F_0^T & F_5^T & F_4^T & F_3^T \end{array}\right)$$

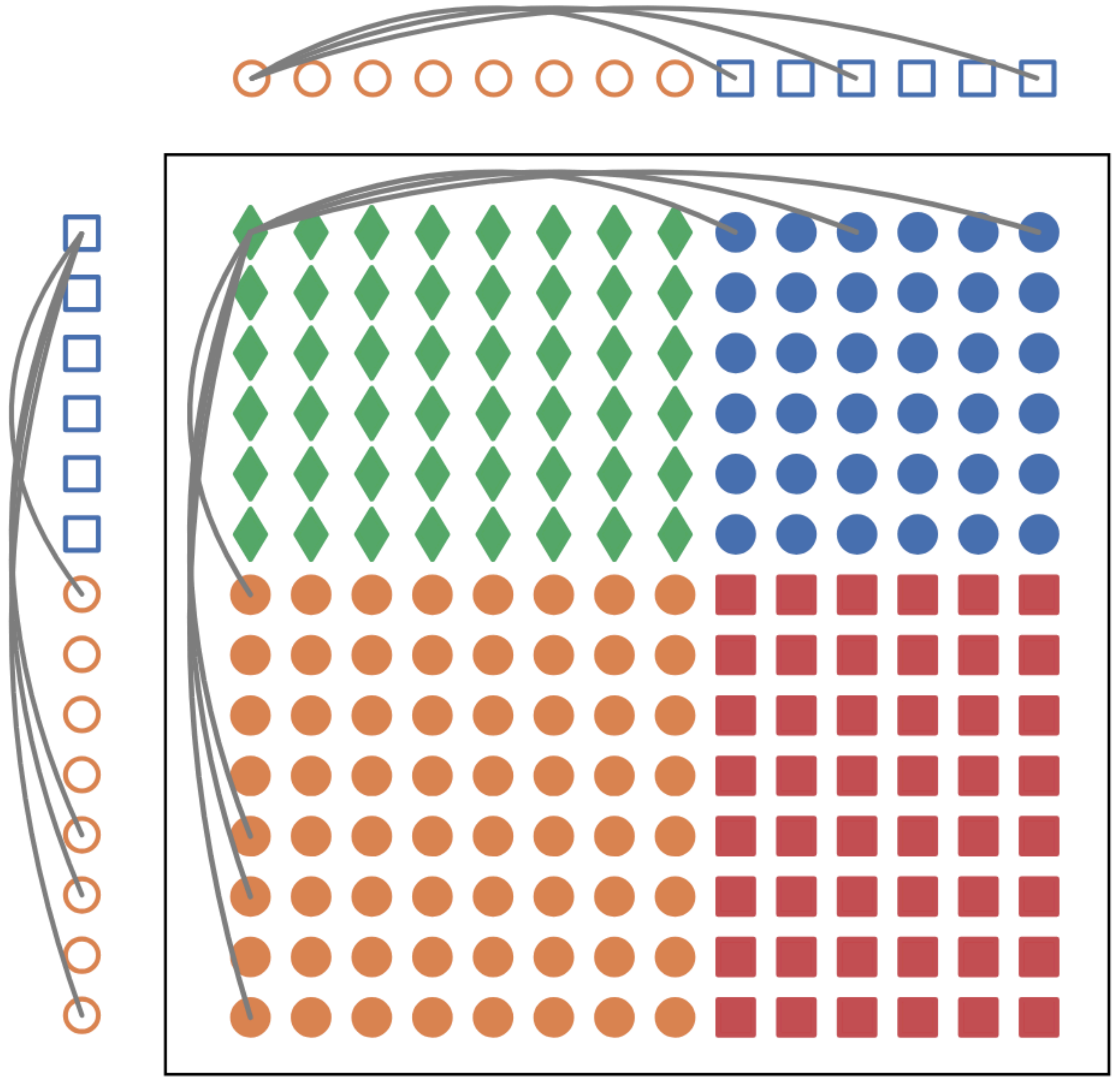
Affine Permutation Matrices (APM) Code

QLDPC Codes

QLDPC codes enable fault-tolerant memories at low space overhead



Bivariate Bicycle Code



Hypergraph Product Code

$$H_X = \left(\begin{array}{cccccc|cccccc} F_0 & F_1 & F_2 & F_3 & F_4 & F_5 & G_0 & G_1 & G_2 & G_3 & G_4 & G_5 \\ F_5 & F_0 & F_1 & F_2 & F_3 & F_4 & G_5 & G_0 & G_1 & G_2 & G_3 & G_4 \\ F_4 & F_5 & F_0 & F_1 & F_2 & F_3 & G_4 & G_5 & G_0 & G_1 & G_2 & G_3 \end{array}\right)$$

$$H_Z = \left(\begin{array}{cccccc|cccccc} G_0^T & G_5^T & G_4^T & G_3^T & G_2^T & G_1^T & F_0^T & F_5^T & F_4^T & F_3^T & F_2^T & F_1^T \\ G_1^T & G_0^T & G_5^T & G_4^T & G_3^T & G_2^T & F_1^T & F_0^T & F_5^T & F_4^T & F_3^T & F_2^T \\ G_2^T & G_1^T & G_0^T & G_5^T & G_4^T & G_3^T & F_2^T & F_1^T & F_0^T & F_5^T & F_4^T & F_3^T \end{array}\right)$$

Affine Permutation Matrices (APM) Code

Lots of work in the past two years that study logical computation on QLDPC codes.

Computation on QLDPC codes

Computation on QLDPC codes

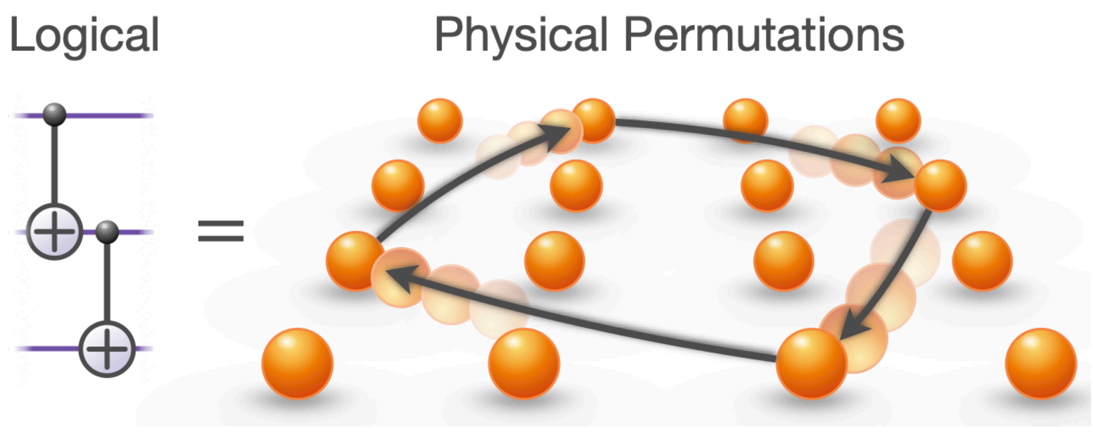
Clifford gates

Non-Clifford gates

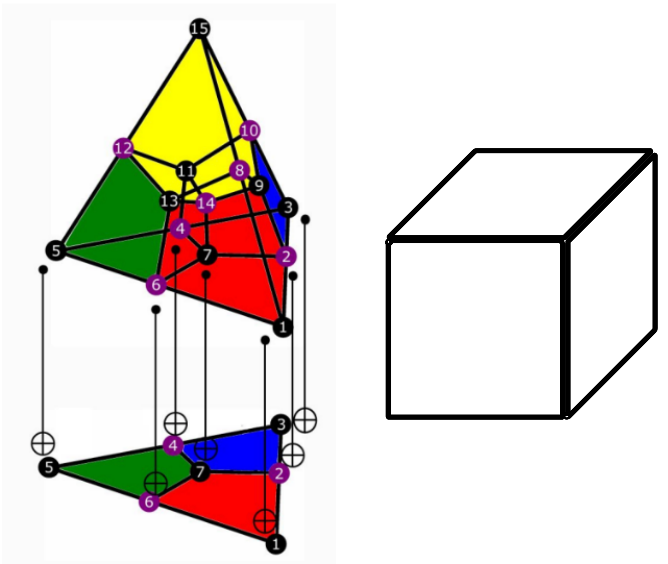
Computation on QLDPC codes

Clifford gates

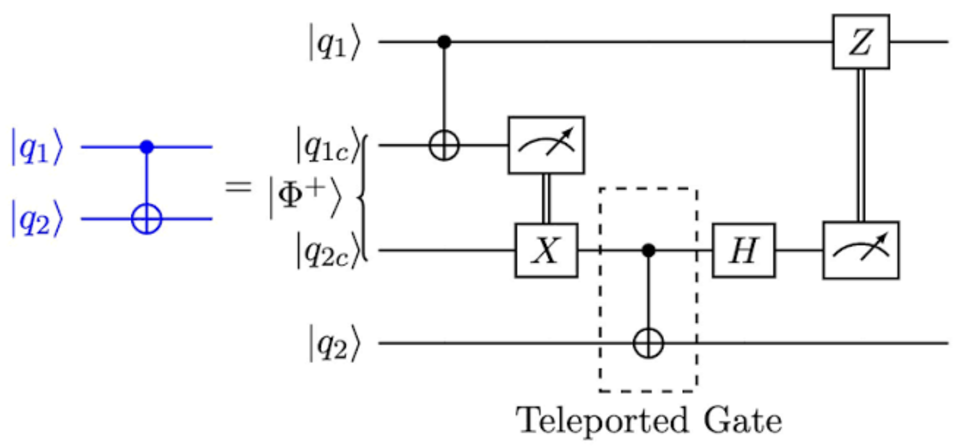
Non-Clifford gates



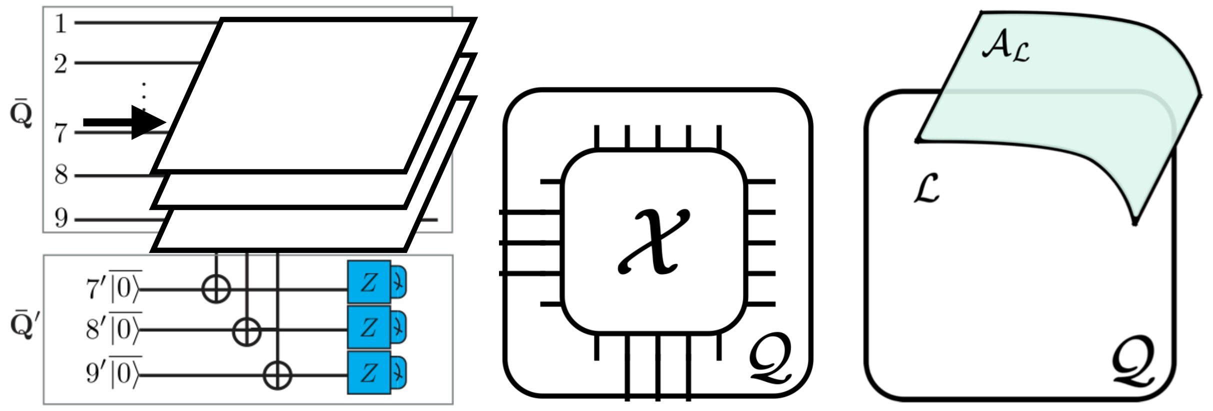
Automorphisms,
Low-depth unitaries



Code-switching



State prep. & gate teleportation

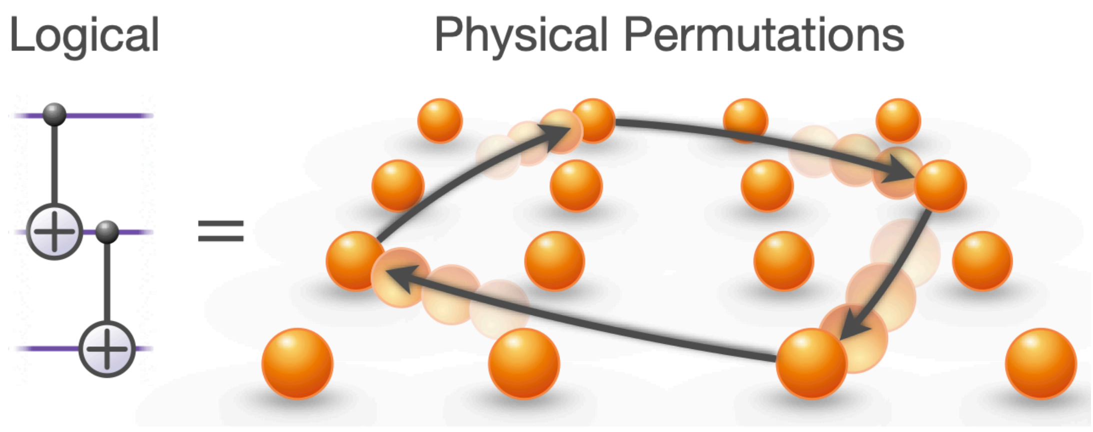


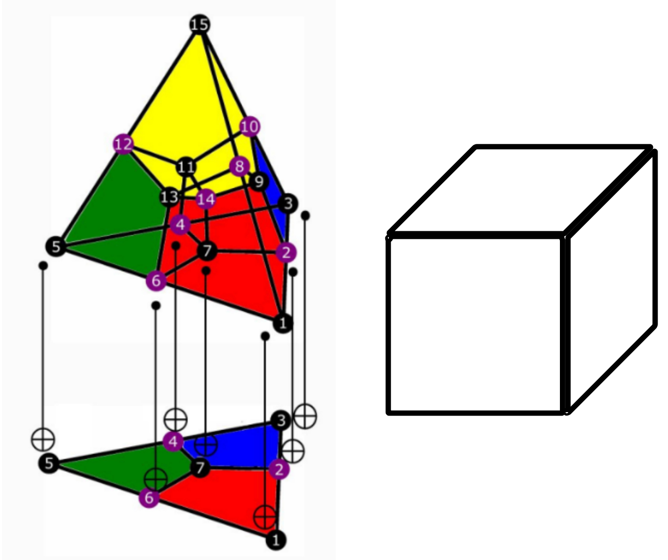
Measuring Pauli operators

Fig. Credits: Koh, *et al.* [2601.20927], Daguerre, Kim. PRR **7** (2025), Xu, *et al.* PRX **15** (2025), He, *et al.* [2503.10390],

Computation on QLDPC codes

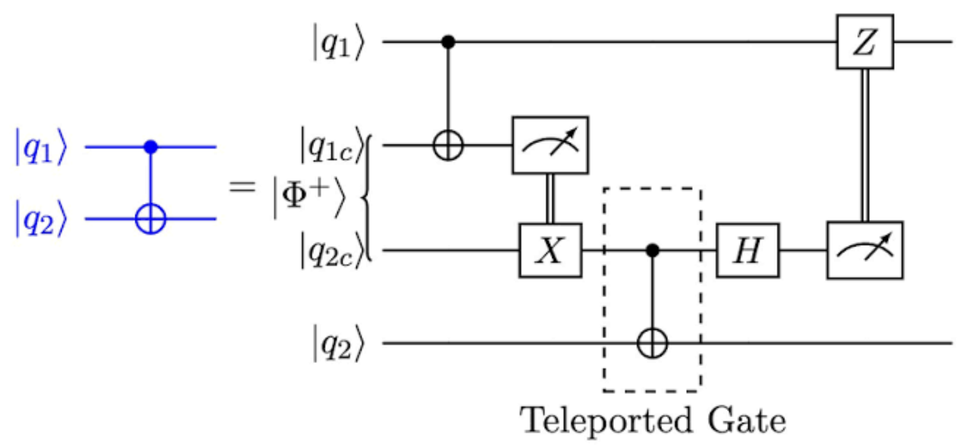
Clifford gates



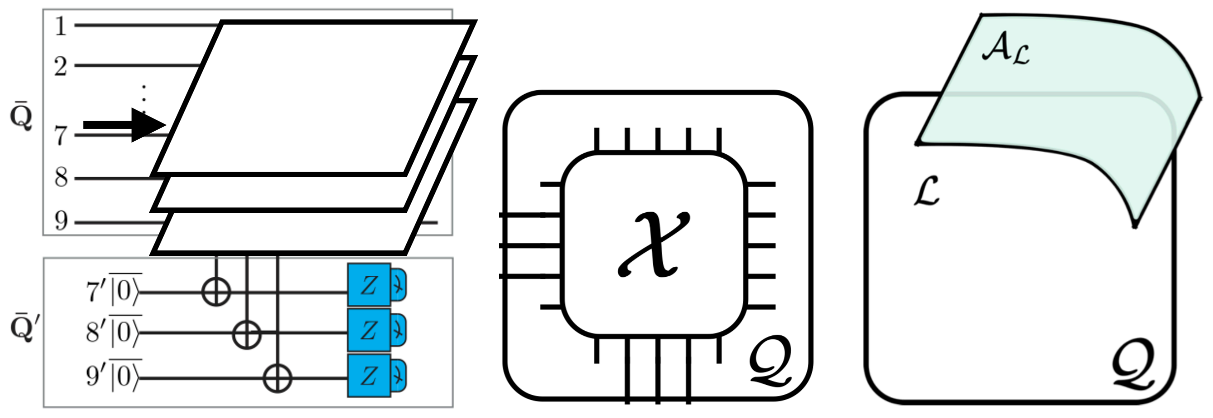


Automorphisms,
Low-depth unitaries

Code-switching

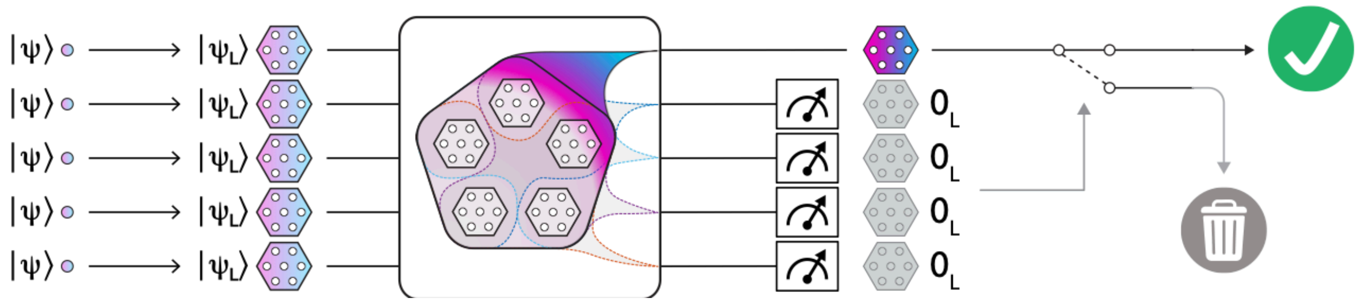


State prep. & gate teleportation

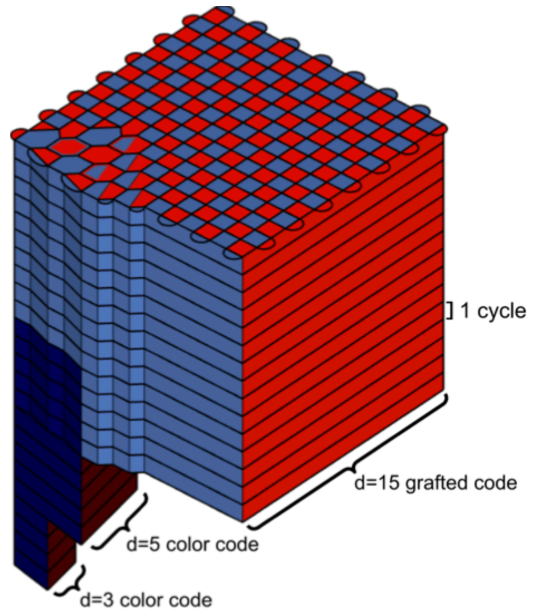


Measuring Pauli operators

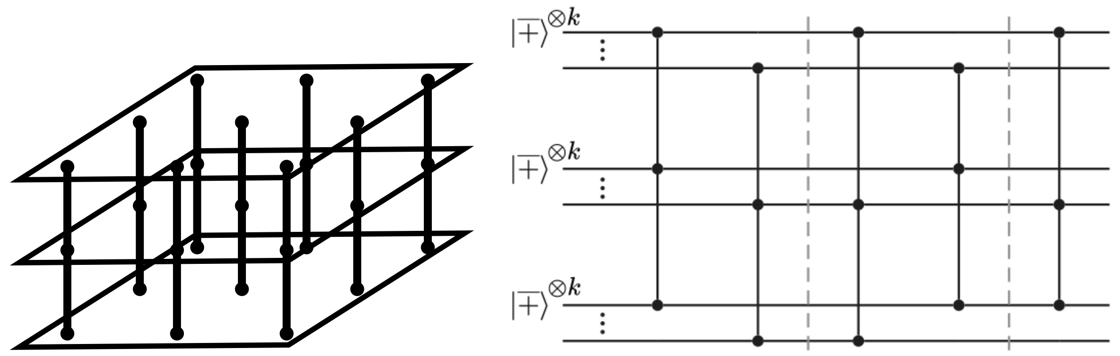
Non-Clifford gates



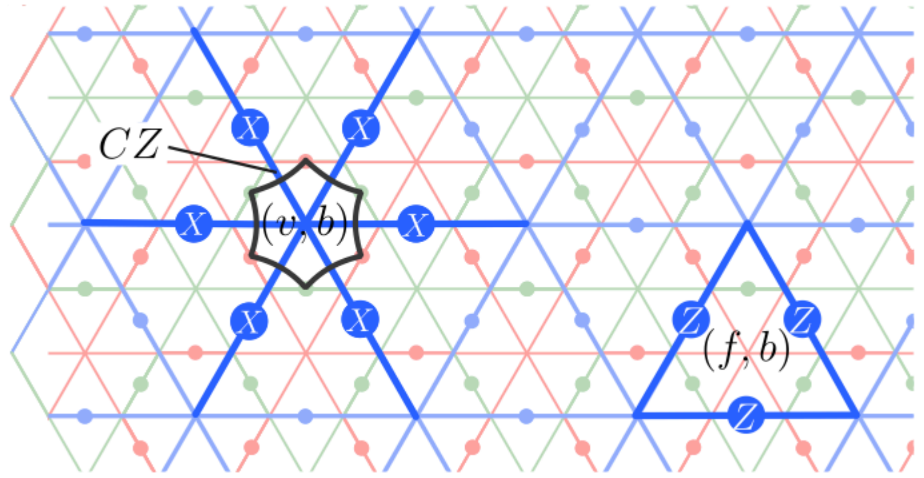
Magic state distillation



Magic state cultivation



Low-depth unitaries



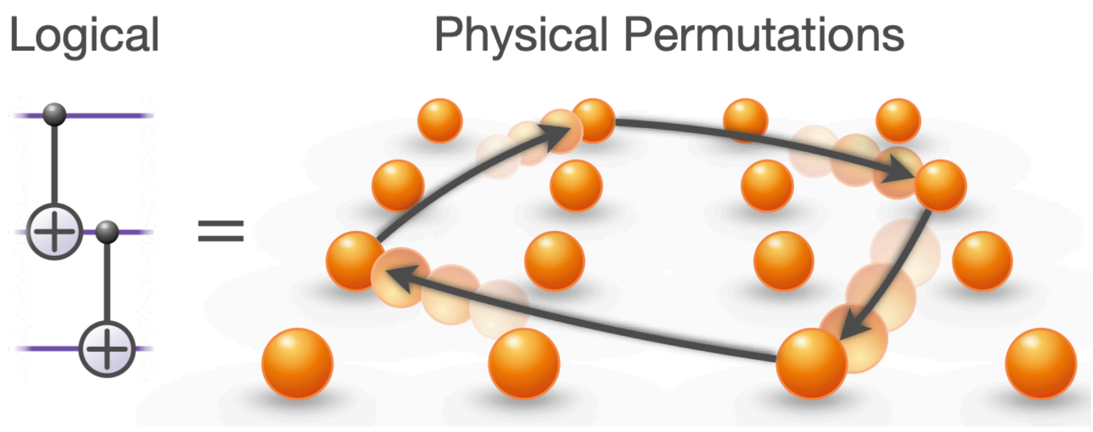
Measuring Clifford operators

Fig. Credits: Koh, *et al.* [2601.20927], Daguerre, Kim. PRR **7** (2025), Xu, *et al.* PRX **15** (2025), He, *et al.* [2503.10390], Sales Rodriguez, *et al.* Nature **645** (2025), Gidney, *et al.* [2409.17595], Menon, *et* al. PRX **16** (2026).Davydova, *et al.* PRX **16** (2026)

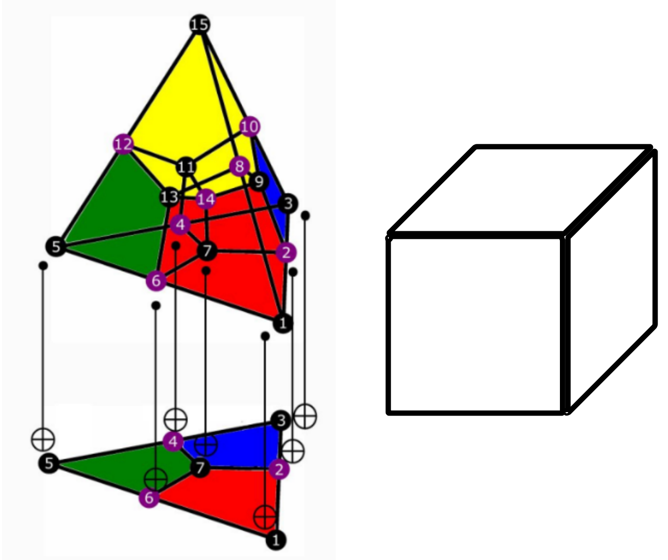
Computation on QLDPC codes

Clifford gates

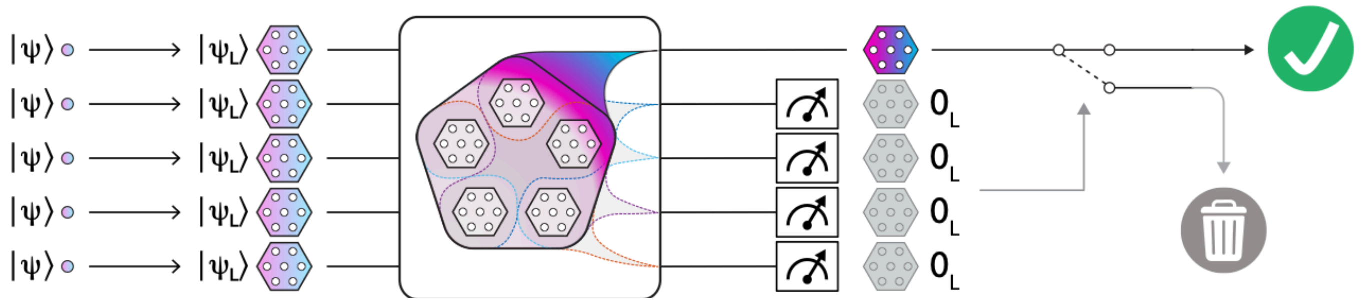
Non-Clifford gates



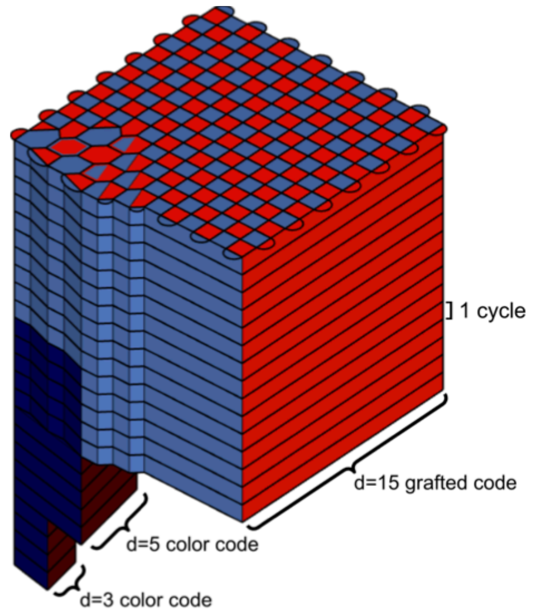
Automorphisms,
Low-depth unitaries



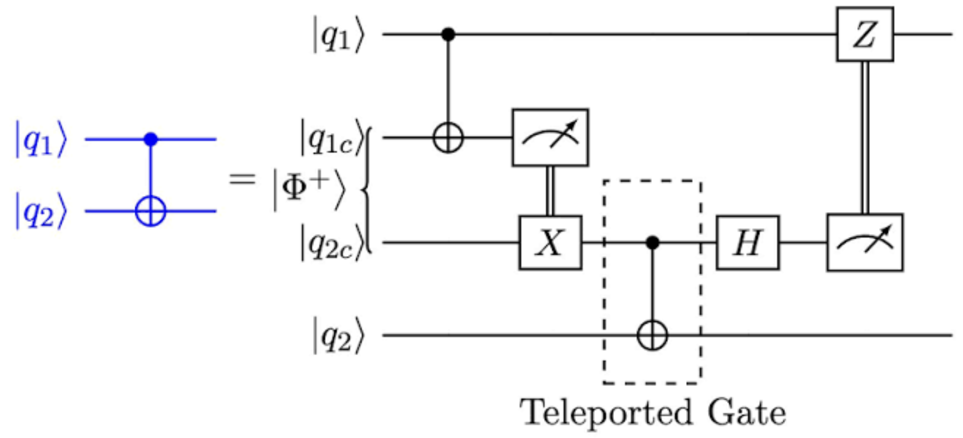
Code-switching



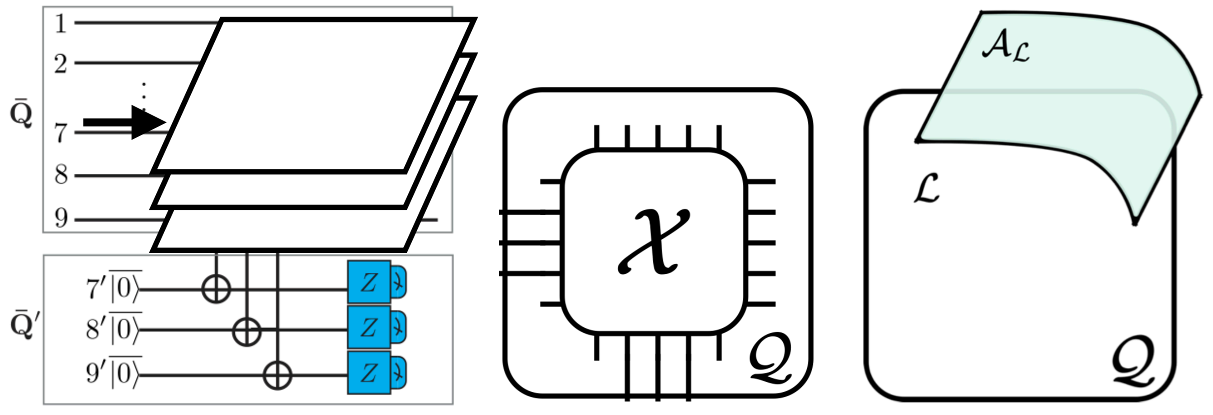
Magic state distillation



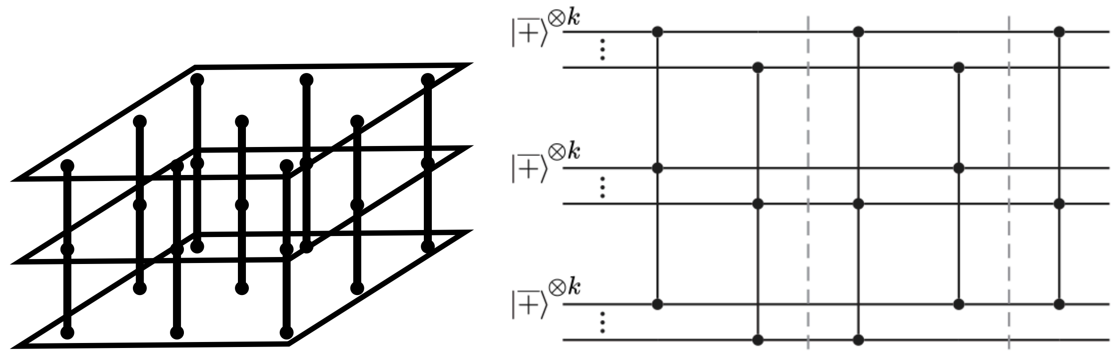
Magic state cultivation



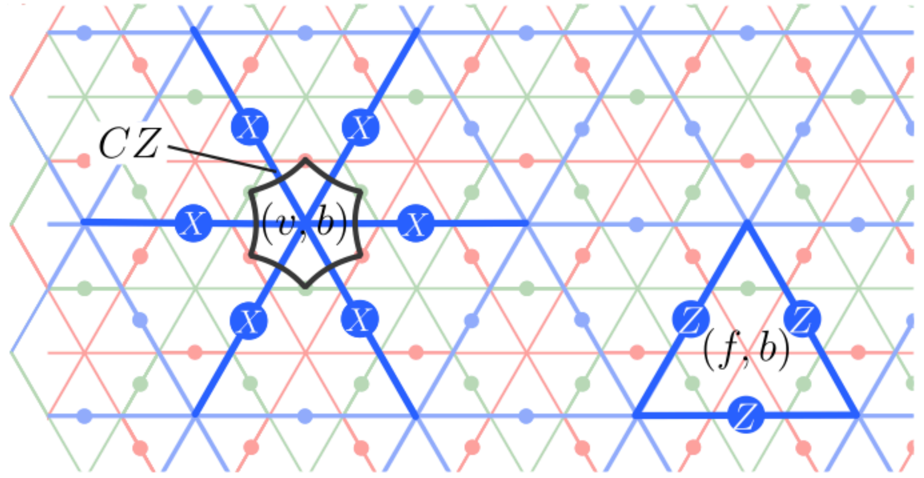
State prep. & gate teleportation



Measuring Pauli operators



Low-depth unitaries

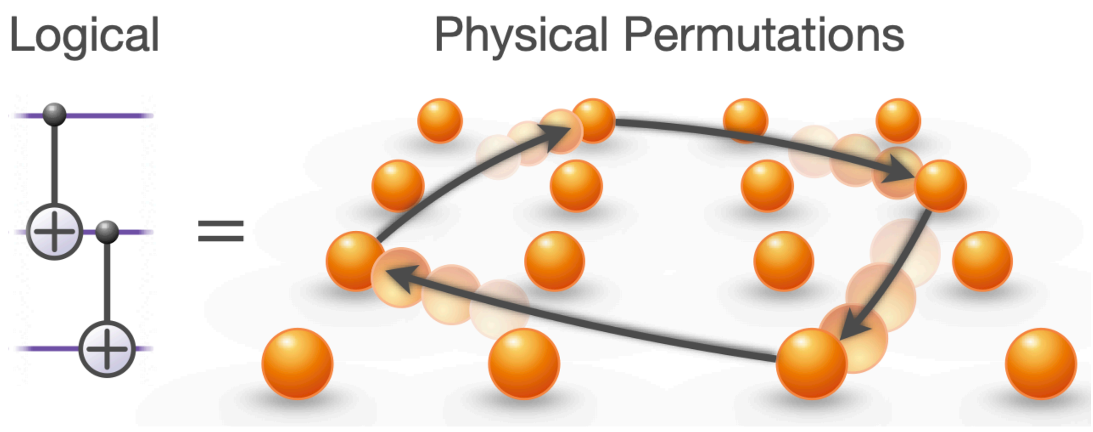


Measuring Clifford operators

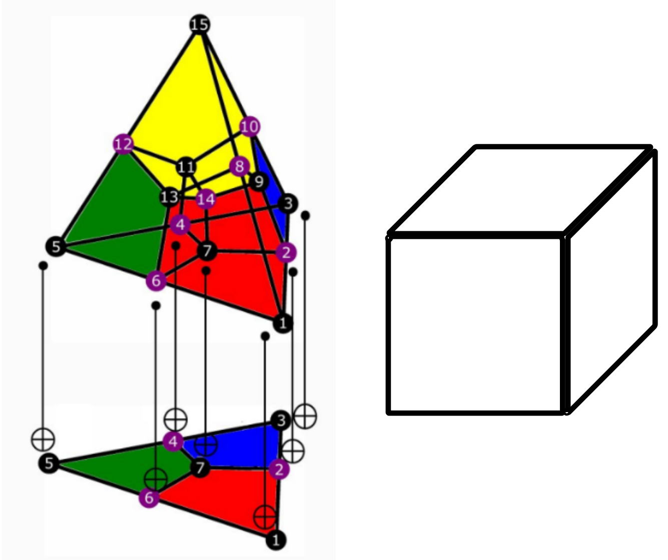
Fig. Credits: Koh, *et al.* [2601.20927], Daguerre, Kim. PRR **7** (2025), Xu, *et al.* PRX **15** (2025), He, *et al.* [2503.10390], Sales Rodriguez, *et al.* Nature **645** (2025), Gidney, *et al.* [2409.17595], Menon, *et* al. PRX **16** (2026).Davydova, *et al.* PRX **16** (2026)

Computation on QLDPC codes

Clifford gates

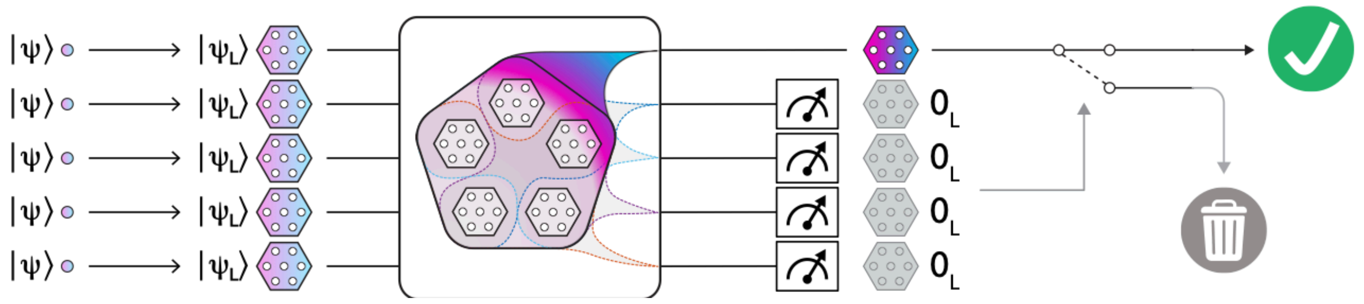


Automorphisms,
Low-depth unitaries

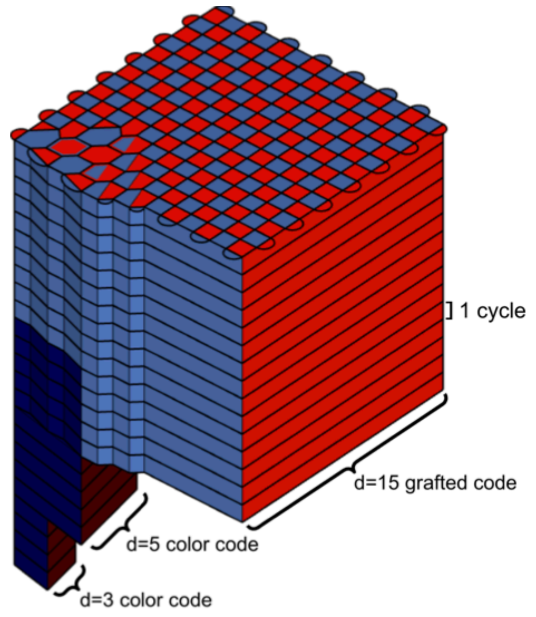


Code-switching

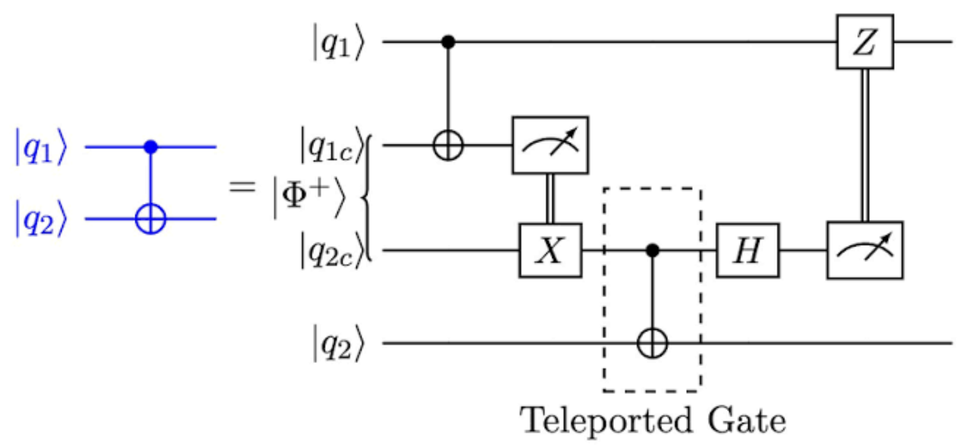
Non-Clifford gates



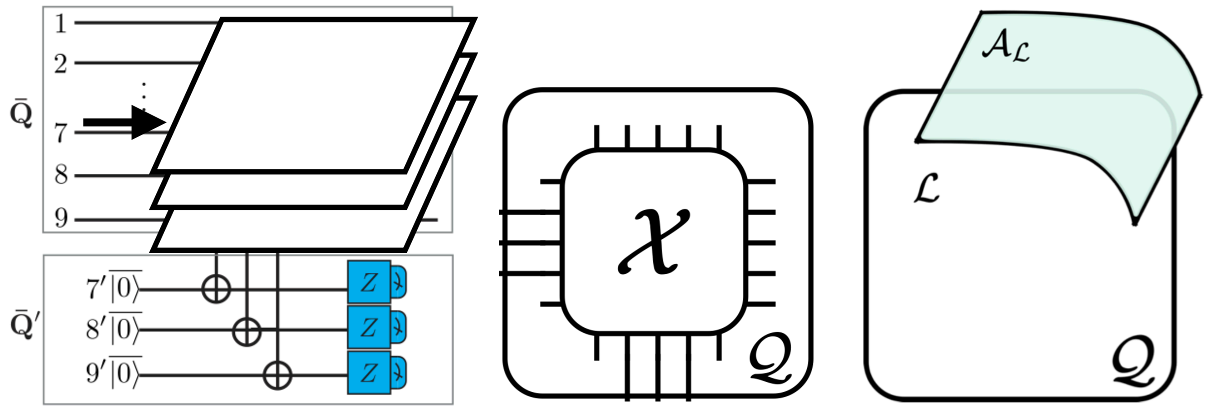
Magic state distillation



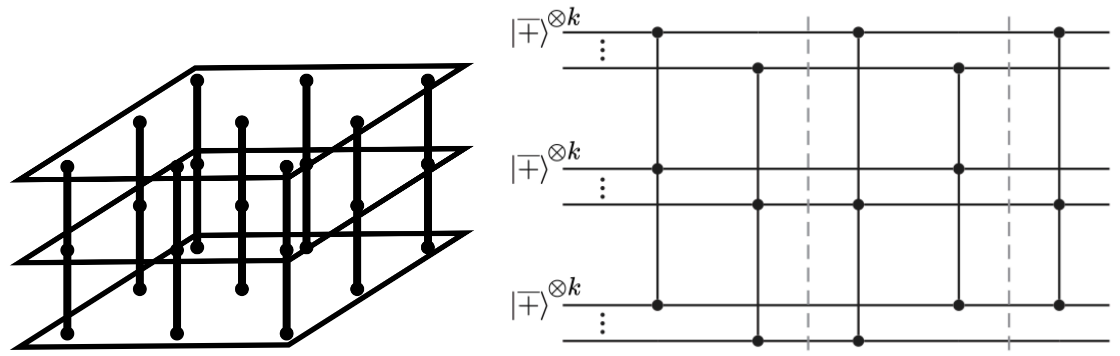
Magic state cultivation



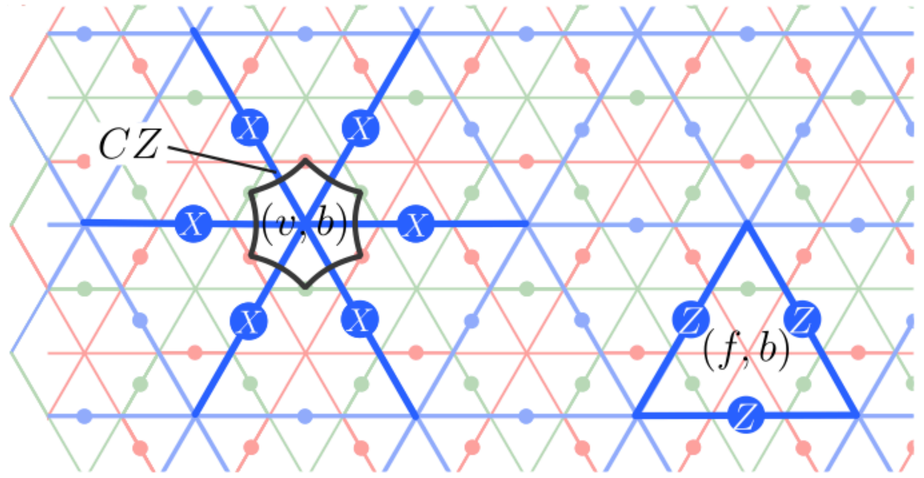
State prep. & gate teleportation



Measuring Pauli operators



Low-depth unitaries

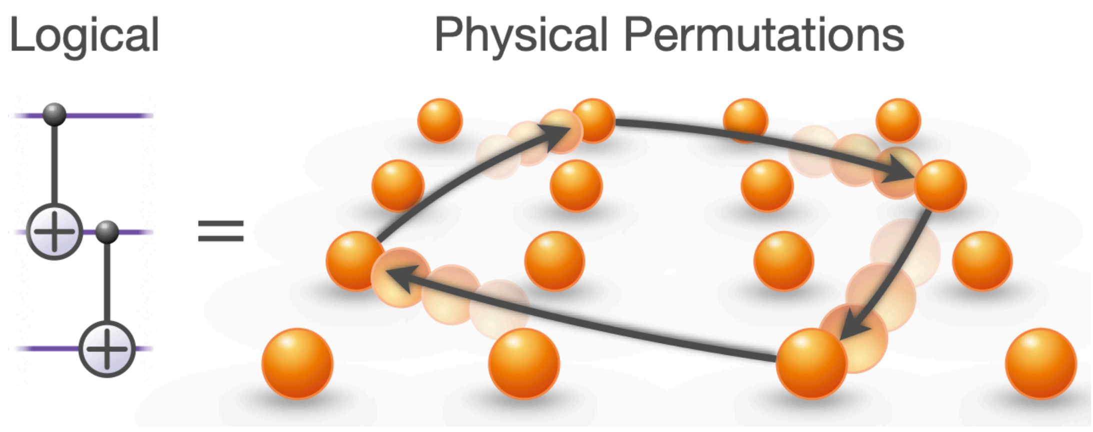


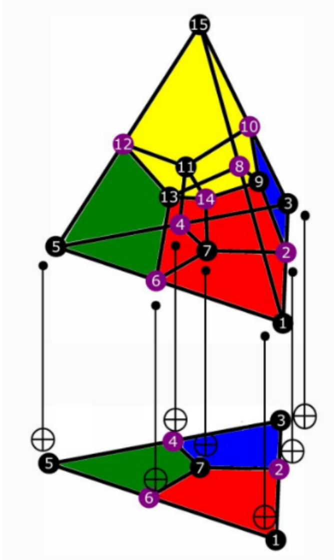
Measuring Clifford operators

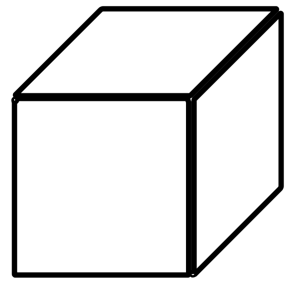
Fig. Credits: Koh, *et al.* [2601.20927], Daguerre, Kim. PRR **7** (2025), Xu, *et al.* PRX **15** (2025), He, *et al.* [2503.10390], Sales Rodriguez, *et al.* Nature **645** (2025), Gidney, *et al.* [2409.17595], Menon, *et* al. PRX **16** (2026).Davydova, *et al.* PRX **16** (2026)

Computation on QLDPC codes

Clifford gates

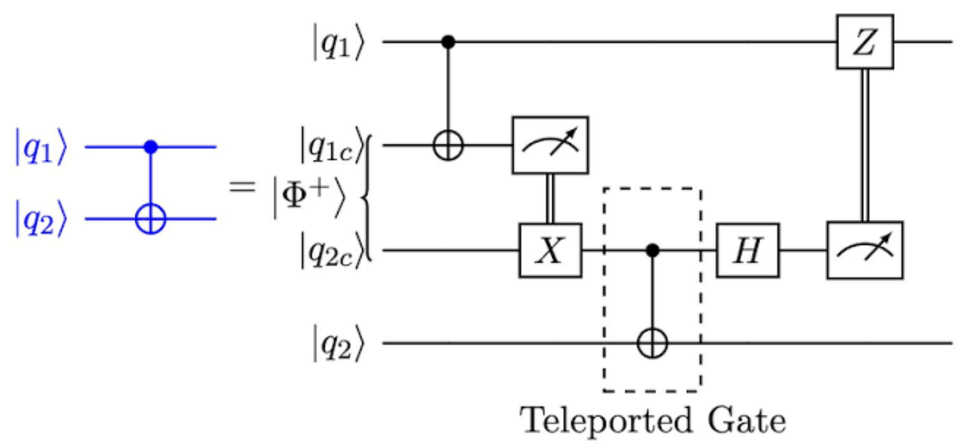




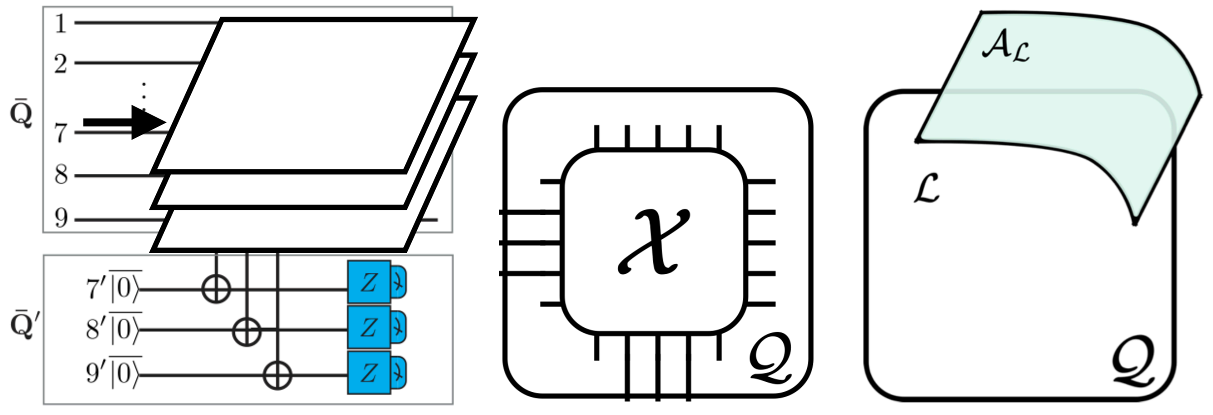


Automorphisms,
Low-depth unitaries

Code-switching



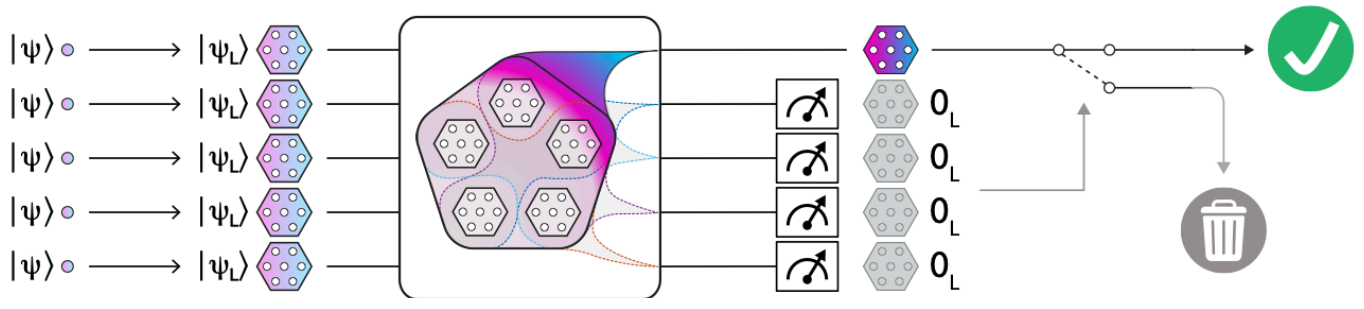
State prep. & gate teleportation

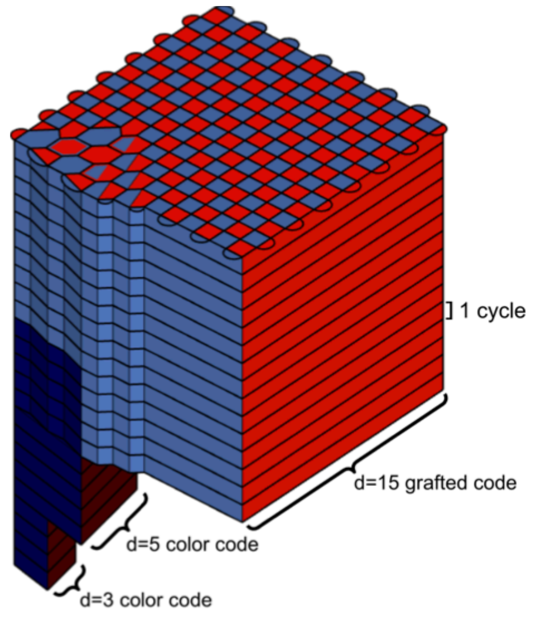


Measuring Pauli operators

Part I. *Fast* primitives for Pauli operator measurements

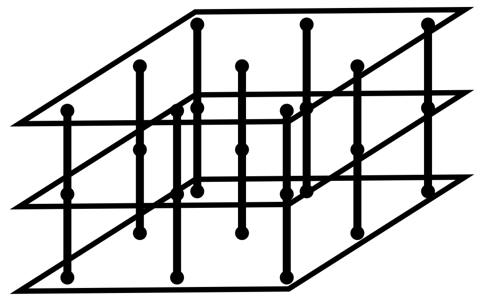
Non-Clifford gates

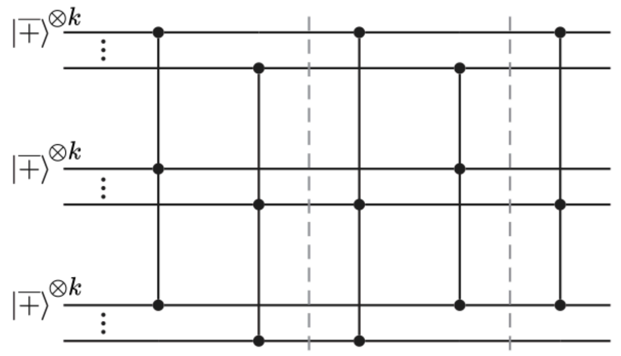




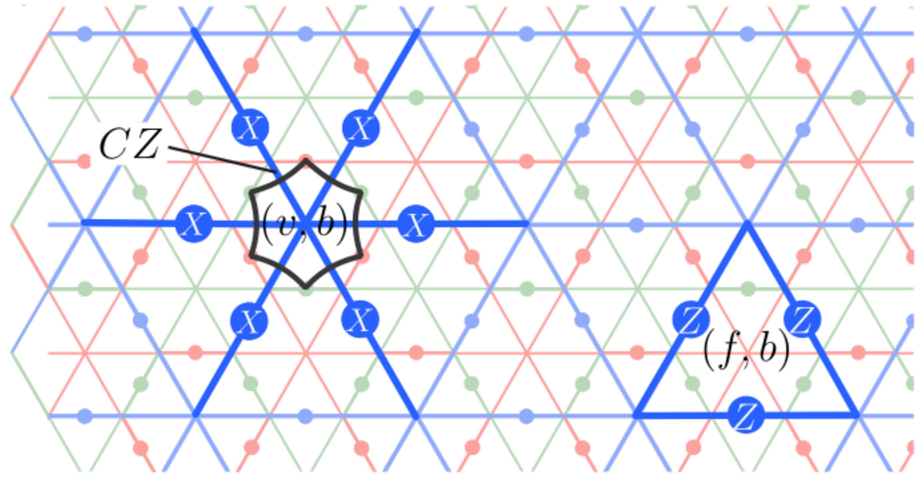
Magic state distillation

Magic state cultivation





Low-depth unitaries

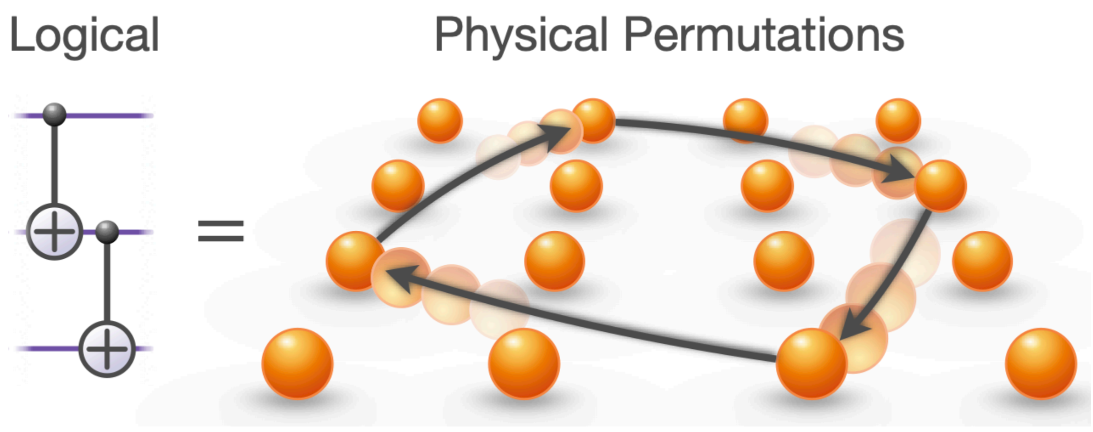


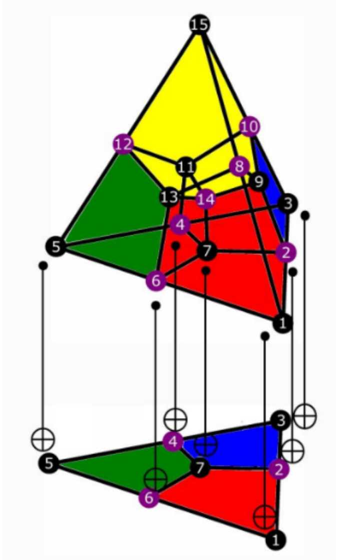
Measuring Clifford operators

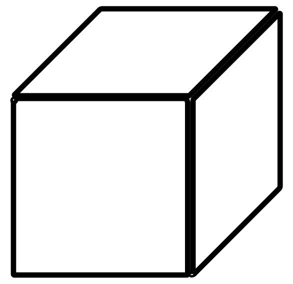
Fig. Credits: Koh, *et al.* [2601.20927], Daguerre, Kim. PRR **7** (2025), Xu, *et al.* PRX **15** (2025), He, *et al.* [2503.10390], Sales Rodriguez, *et al.* Nature **645** (2025), Gidney, *et al.* [2409.17595], Menon, *et* al. PRX **16** (2026).Davydova, *et al.* PRX **16** (2026)

Computation on QLDPC codes

Clifford gates

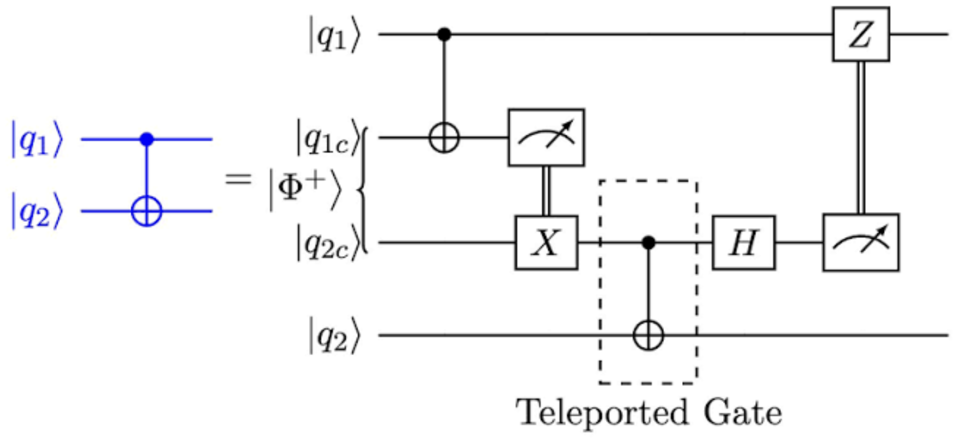




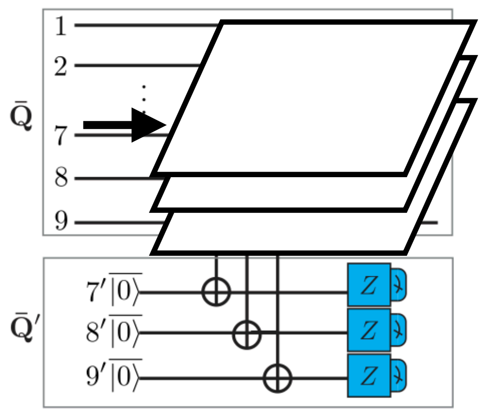


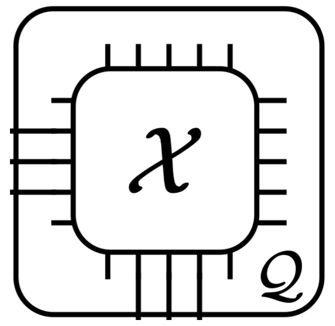
Automorphisms,
Low-depth unitaries

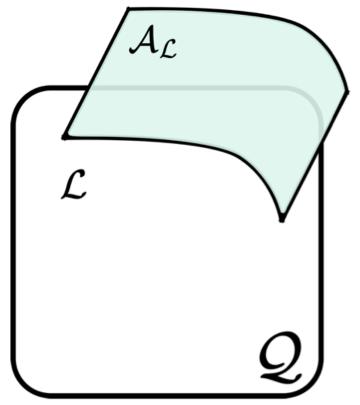
Code-switching



State prep. & gate teleportation

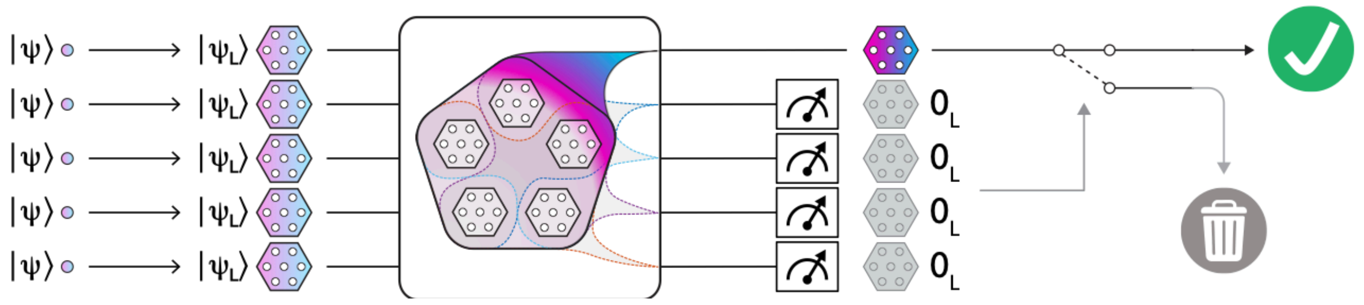


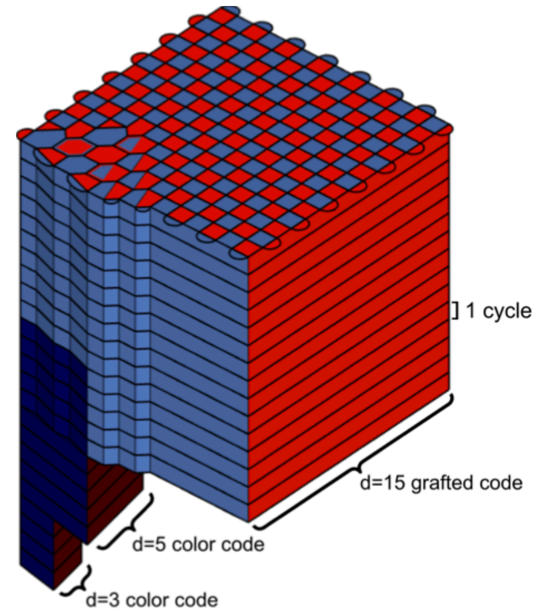




Measuring Pauli operators

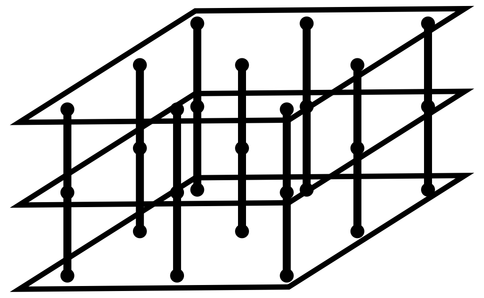
Non-Clifford gates

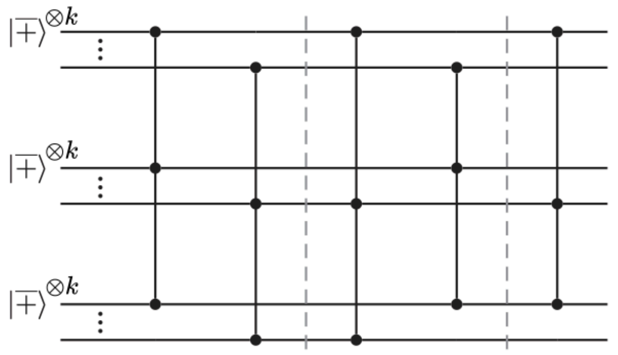


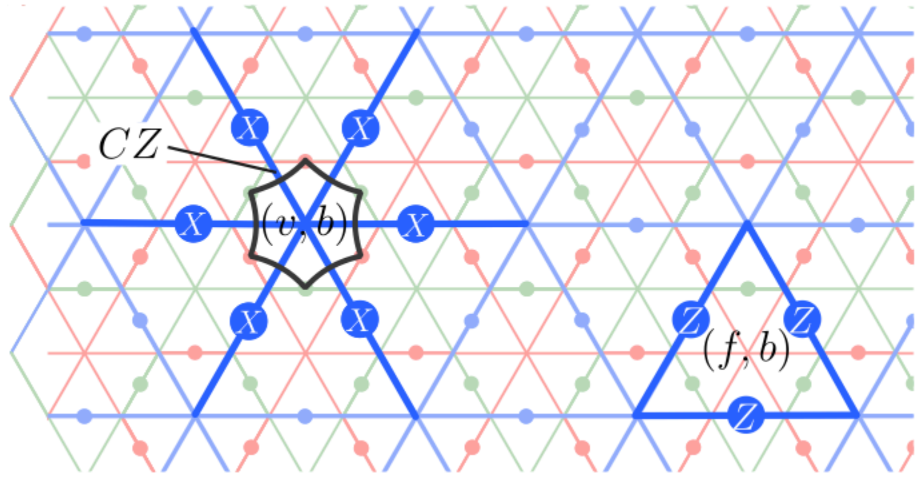


Magic state distillation

Magic state cultivation







Low-depth unitaries

Measuring Clifford operators

Part I. *Fast* primitives for Pauli operator measurements

Part II. New (*Furious*?) techniques for measuring Clifford operators on *QLDPC* codes

Fig. Credits: Koh, *et al.* [2601.20927], Daguerre, Kim. PRR **7** (2025), Xu, *et al.* PRX **15** (2025), He, *et al.* [2503.10390], Sales Rodriguez, *et al.* Nature **645** (2025), Gidney, *et al.* [2409.17595], Menon, *et* al. PRX **16** (2026).Davydova, *et al.* PRX **16** (2026)

Code Surgery for QLDPC Codes

Code Surgery for QLDPC Codes

Surgery is a method to perform logical Pauli measurement on QLDPC codes using ancilla systems.

Code Surgery for QLDPC Codes

Surgery is a method to perform logical Pauli measurement on QLDPC codes using ancilla systems.

Many desirable properties:

Code Surgery for QLDPC Codes

Surgery is a method to perform logical Pauli measurement on QLDPC codes using ancilla systems.

Many desirable properties:
- Provably fault-tolerant and low overhead.[1,2]

[1] Williamson, Yoder. [2410.02213] [2] Ide et al. [2410.02753]

Code Surgery for QLDPC Codes

Surgery is a method to perform logical Pauli measurement on QLDPC codes using ancilla systems.

Many desirable properties:

- Provably fault-tolerant and low overhead.[1,2]
- Good finite-size performance.[3 - 10]

[1] Williamson, Yoder. [2410.02213] [2] Ide et al. [2410.02753] [3] Cross et al. [2407.18393] [4] Zheng et al. [2510.08523] [5] Yoder et al. [2506.03094]
[6] Webster et al. [2602.11457] [7] Gu et al. [2603.05398] [8] Cain et al. [2603.28627] [9] Wills et al. [2605.21898] [10] Blue et al. [2606.03507]

Code Surgery for QLDPC Codes

Surgery is a method to perform logical Pauli measurement on QLDPC codes using ancilla systems.

Many desirable properties:

- Provably fault-tolerant and low overhead.[1,2]
- Good finite-size performance.[3 - 10]
- Highly versatile:

[1] Williamson, Yoder. [2410.02213] [2] Ide et al. [2410.02753] [3] Cross et al. [2407.18393] [4] Zheng et al. [2510.08523] [5] Yoder et al. [2506.03094]
[6] Webster et al. [2602.11457] [7] Gu et al. [2603.05398] [8] Cain et al. [2603.28627] [9] Wills et al. [2605.21898] [10] Blue et al. [2606.03507]

Code Surgery for QLDPC Codes

Surgery is a method to perform logical Pauli measurement on QLDPC codes using ancilla systems.

Many desirable properties:

- Provably fault-tolerant and low overhead.[1,2]
- Good finite-size performance.[3 - 10]
- Highly versatile:
  - Can build an extractor system to measure arbitrary logicals sequentially.[11]

[1] Williamson, Yoder. [2410.02213] [2] Ide et al. [2410.02753] [3] Cross et al. [2407.18393] [4] Zheng et al. [2510.08523] [5] Yoder et al. [2506.03094]
[6] Webster et al. [2602.11457] [7] Gu et al. [2603.05398] [8] Cain et al. [2603.28627] [9] Wills et al. [2605.21898] [10] Blue et al. [2606.03507]
[11] He et al. [2503.10390]

Code Surgery for QLDPC Codes

Surgery is a method to perform logical Pauli measurement on QLDPC codes using ancilla systems.

Many desirable properties:

- Provably fault-tolerant and low overhead.[1,2]
- Good finite-size performance.[3 - 10]
- Highly versatile:
  - Can build an extractor system to measure arbitrary logicals sequentially.[11]

[1] Williamson, Yoder. [2410.02213] [2] Ide et al. [2410.02753] [3] Cross et al. [2407.18393] [4] Zheng et al. [2510.08523] [5] Yoder et al. [2506.03094]
[6] Webster et al. [2602.11457] [7] Gu et al. [2603.05398] [8] Cain et al. [2603.28627] [9] Wills et al. [2605.21898] [10] Blue et al. [2606.03507]
[11] He et al. [2503.10390]

Code Surgery for QLDPC Codes

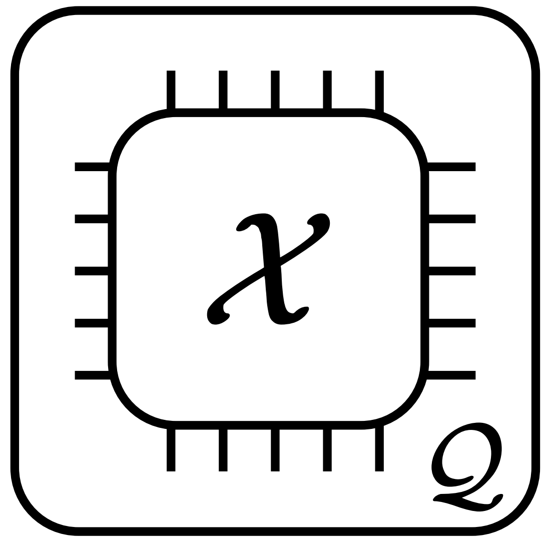
Surgery is a method to perform logical Pauli measurement on QLDPC codes using ancilla systems.

Many desirable properties:

- Provably fault-tolerant and low overhead.[1,2]
- Good finite-size performance.[3 - 10]
- Highly versatile:
  - Can build an extractor system to measure arbitrary logicals sequentially.[11]

[1] Williamson, Yoder. [2410.02213] [2] Ide et al. [2410.02753] [3] Cross et al. [2407.18393] [4] Zheng et al. [2510.08523] [5] Yoder et al. [2506.03094]
[6] Webster et al. [2602.11457] [7] Gu et al. [2603.05398] [8] Cain et al. [2603.28627] [9] Wills et al. [2605.21898] [10] Blue et al. [2606.03507]
[11] He et al. [2503.10390]

Code Surgery for QLDPC Codes



Surgery is a method to perform logical Pauli measurement on QLDPC codes using ancilla systems.

Many desirable properties:
- Provably fault-tolerant and low overhead.[1,2]
- Good finite-size performance.[3 - 10]
- Highly versatile:
  - Can build an extractor system to measure arbitrary logicals sequentially.[11]
  - Or in parallel with different designs.[3,12,13]

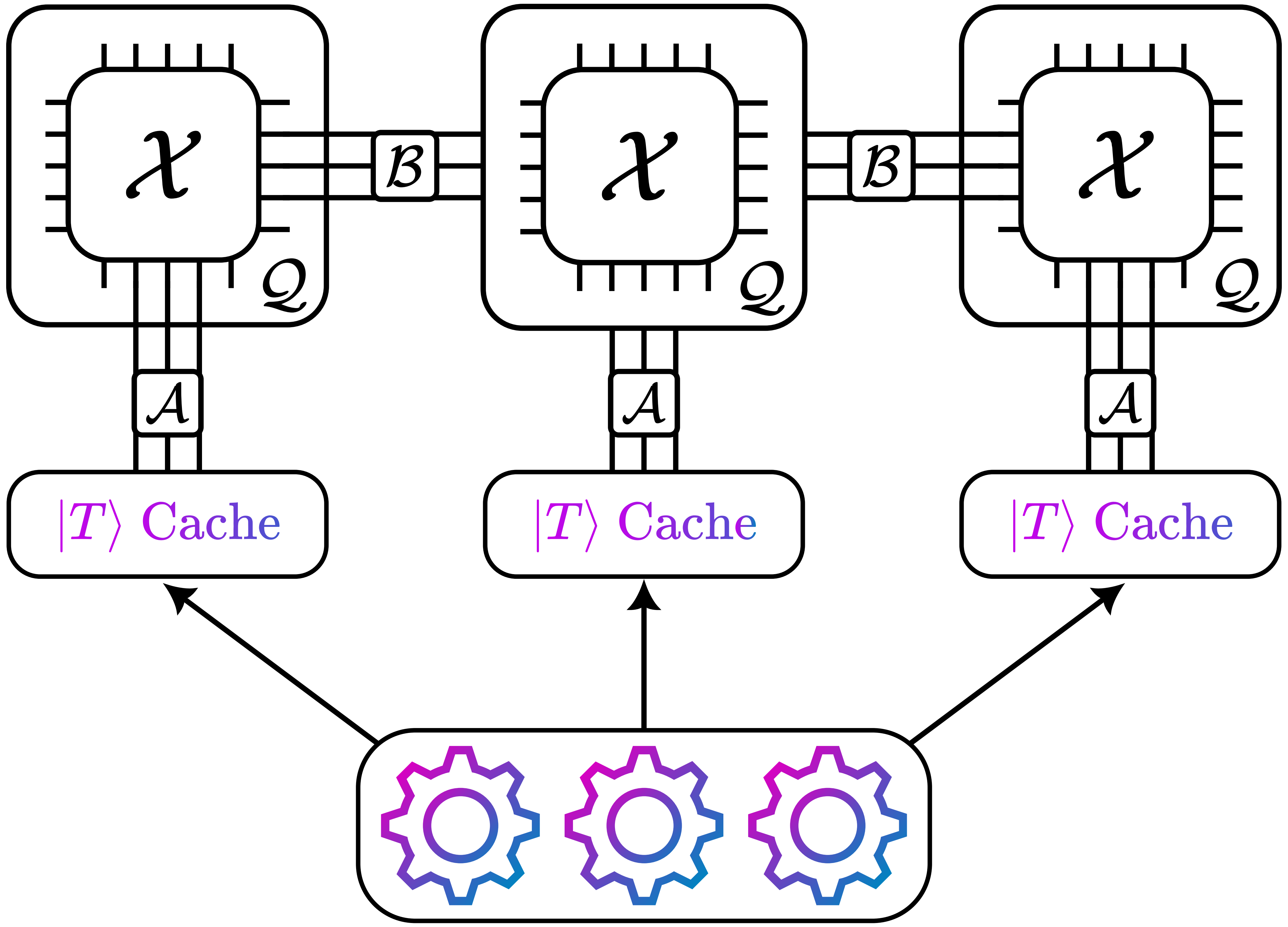
[1] Williamson, Yoder. [2410.02213] [2] Ide et al. [2410.02753] [3] Cross et al. [2407.18393] [4] Zheng et al. [2510.08523] [5] Yoder et al. [2506.03094]
[6] Webster et al. [2602.11457] [7] Gu et al. [2603.05398] [8] Cain et al. [2603.28627] [9] Wills et al. [2605.21898] [10] Blue et al. [2606.03507]
[11] He et al. [2503.10390] [12] Zhang, Li. [2408.01339] [13] Cowtan et al. [2503.05003]

Code Surgery for QLDPC Codes

Surgery is a method to perform logical Pauli measurement on QLDPC codes using ancilla systems.

Many desirable properties:

- Provably fault-tolerant and low overhead.[1,2]
- Good finite-size performance.[3 - 10]
- Highly versatile:
  - Can build an extractor system to measure arbitrary logicals sequentially.[11]
  - Or in parallel with different designs.[3,12,13]
- Can be adapted to many hardware platforms.[5,6,8,10,11]



Extractor Architectures,
a model for surgery architectures

[1] Williamson, Yoder. [2410.02213] [2] Ide et al. [2410.02753] [3] Cross et al. [2407.18393] [4] Zheng et al. [2510.08523] [5] Yoder et al. [2506.03094]
[6] Webster et al. [2602.11457] [7] Gu et al. [2603.05398] [8] Cain et al. [2603.28627] [9] Wills et al. [2605.21898] [10] Blue et al. [2606.03507]
[11] He et al. [2503.10390] [12] Zhang, Li. [2408.01339] [13] Cowtan et al. [2503.05003]

The Need for Speed

The Need for Speed

Aforementioned surgery systems all take O(d) time to implement one round of logical gates.

- This is a crucial bottleneck. After all, we cannot build more time.

The Need for Speed

Aforementioned surgery systems all take O(d) time to implement one round of logical gates.
◆ This is a crucial bottleneck. After all, we cannot build more time.

This talk: breaking the O(d) time barrier to implement surgery in O(1) time.

The Need for Speed

Aforementioned surgery systems all take O(d) time to implement one round of logical gates.
◆ This is a crucial bottleneck. After all, we cannot build more time.

This talk: breaking the O(d) time barrier to implement surgery in O(1) time.

Formulation of fast surgery as constant-time surgery from [1]

[1] Cowtan, He, Williamson, Yoder. [2510.14895]

The Need for Speed

Aforementioned surgery systems all take O(d) time to implement one round of logical gates.
- This is a crucial bottleneck. After all, we cannot build more time.

This talk: breaking the O(d) time barrier to implement surgery in O(1) time.

Formulation of fast surgery as constant-time surgery from [1]
- O(d) surgery operations in O(d) time.

[1] Cowtan, He, Williamson, Yoder. [2510.14895]

The Need for Speed

Aforementioned surgery systems all take O(d) time to implement one round of logical gates.
- This is a crucial bottleneck. After all, we cannot build more time.

This talk: breaking the O(d) time barrier to implement surgery in O(1) time.

Formulation of fast surgery as constant-time surgery from [1]
- O(d) surgery operations in O(d) time.

Construction on 2-dimensional hypergraph product codes

[1] Cowtan, He, Williamson, Yoder. [2510.14895]

The Need for Speed

Aforementioned surgery systems all take O(d) time to implement one round of logical gates.

- This is a crucial bottleneck. After all, we cannot build more time.

This talk: breaking the O(d) time barrier to implement surgery in O(1) time.

Formulation of fast surgery as constant-time surgery from [1]

- O(d) surgery operations in O(d) time.

Construction on 2-dimensional hypergraph product codes

- Fast, partially addressable and parallel measurements in almost constant space overhead, on any hypergraph product code.

[1] Cowtan, He, Williamson, Yoder. [2510.14895]

The Need for Speed

Aforementioned surgery systems all take O(d) time to implement one round of logical gates.

- This is a crucial bottleneck. After all, we cannot build more time.

This talk: breaking the O(d) time barrier to implement surgery in O(1) time.

Formulation of fast surgery as constant-time surgery from [1]

- O(d) surgery operations in O(d) time.

Construction on 2-dimensional hypergraph product codes

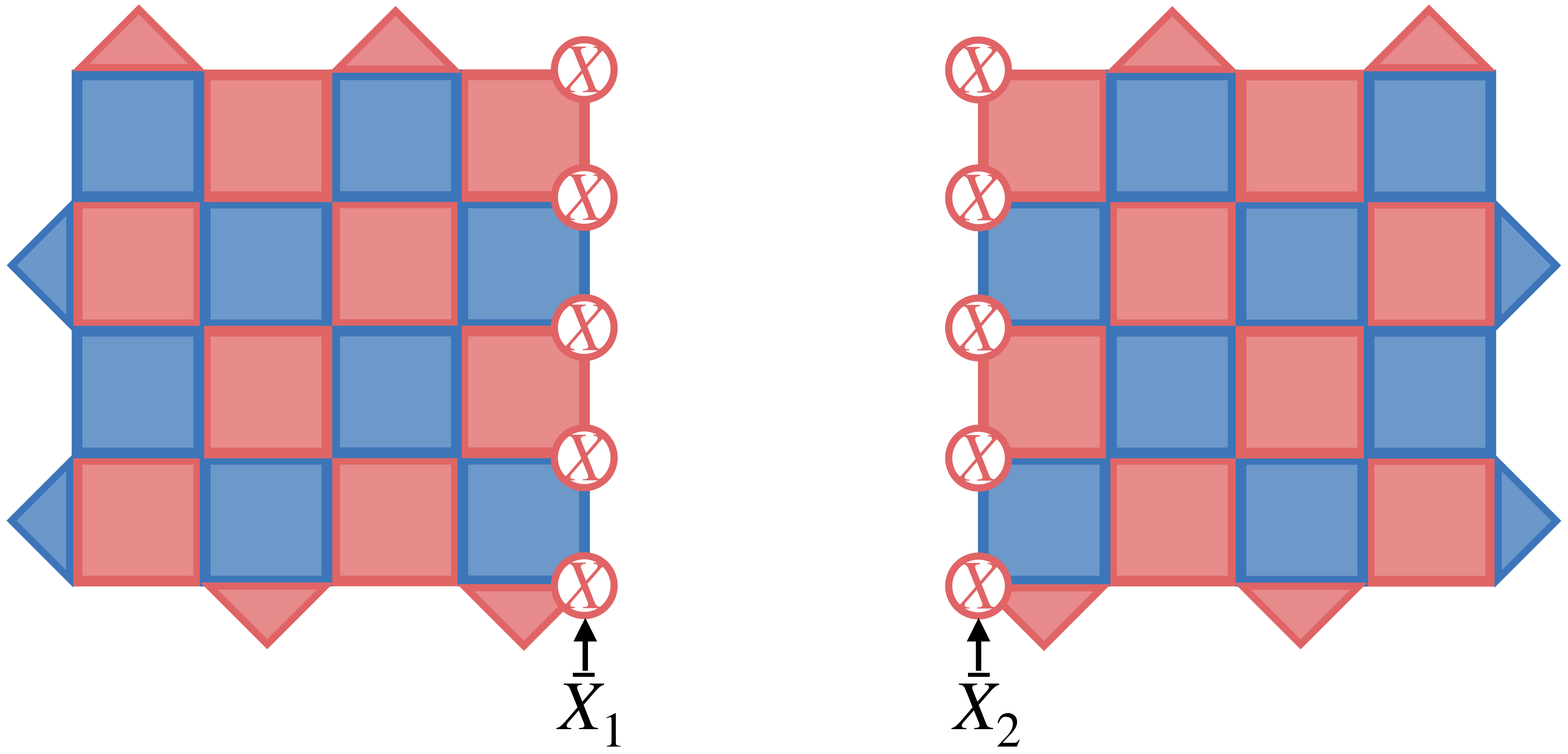
- Fast, partially addressable and parallel measurements in almost constant space overhead, on any hypergraph product code.

Next: why does surgery take O(d) time?

[1] Cowtan, He, Williamson, Yoder. [2510.14895]

Lattice surgery on the surface code

Ex: Measuring $\bar{X}_1\bar{X}_2$

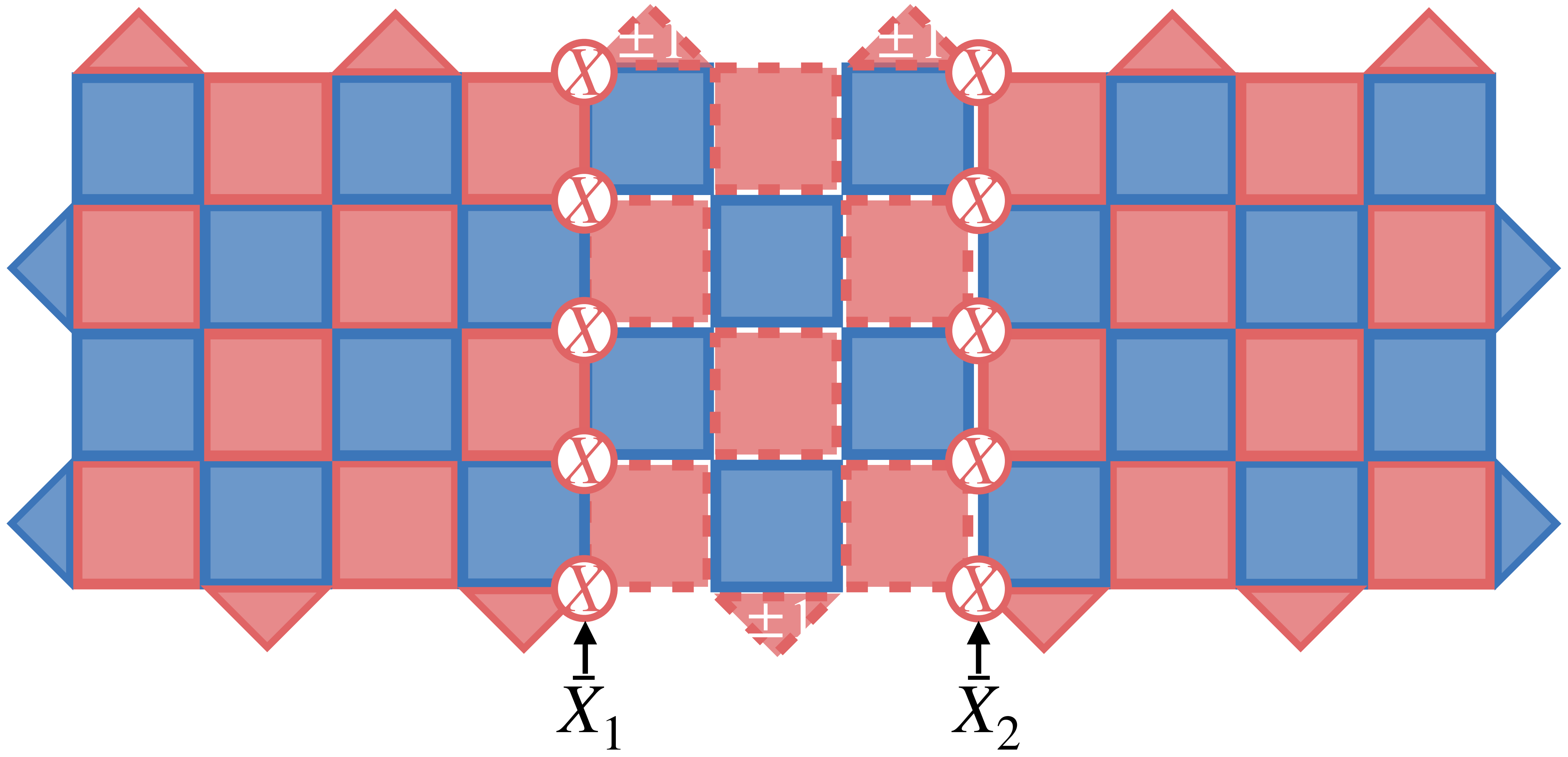


[1] Vuillot *et al.* NJP **21** (2019).

Lattice surgery on the surface code

Ex: Measuring $\bar{X}_1\bar{X}_2$

$\bar{X}_1\bar{X}_2 = \prod$



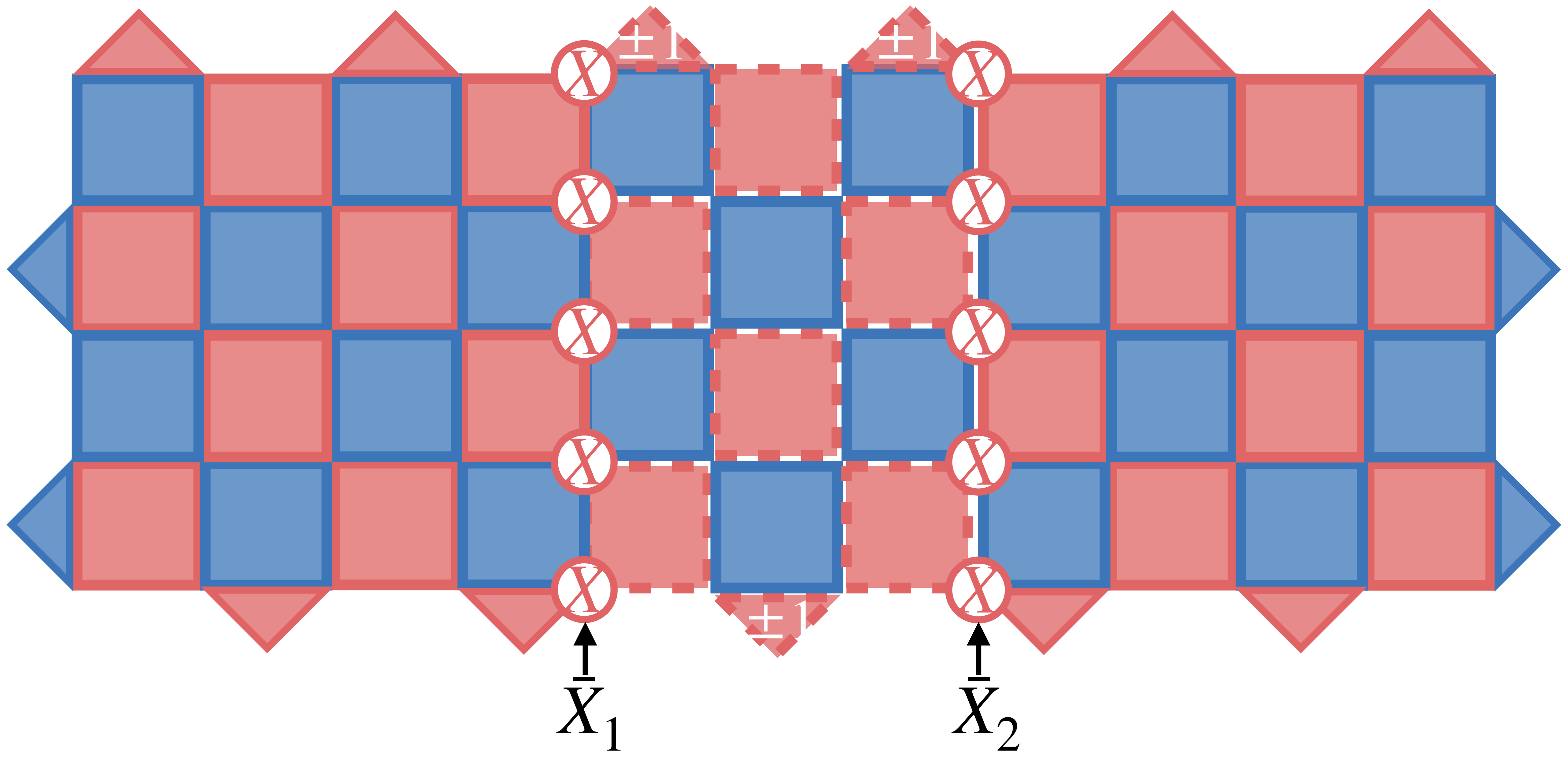
- Switch to deformed/merged code where logical operator to be measured = products of stabilizers[1]

[1] Vuillot *et al.* NJP **21** (2019).

Lattice surgery on the surface code

Ex: Measuring $\bar{X}_1\bar{X}_2$

$$\bar{X}_1\bar{X}_2 = \prod \square$$

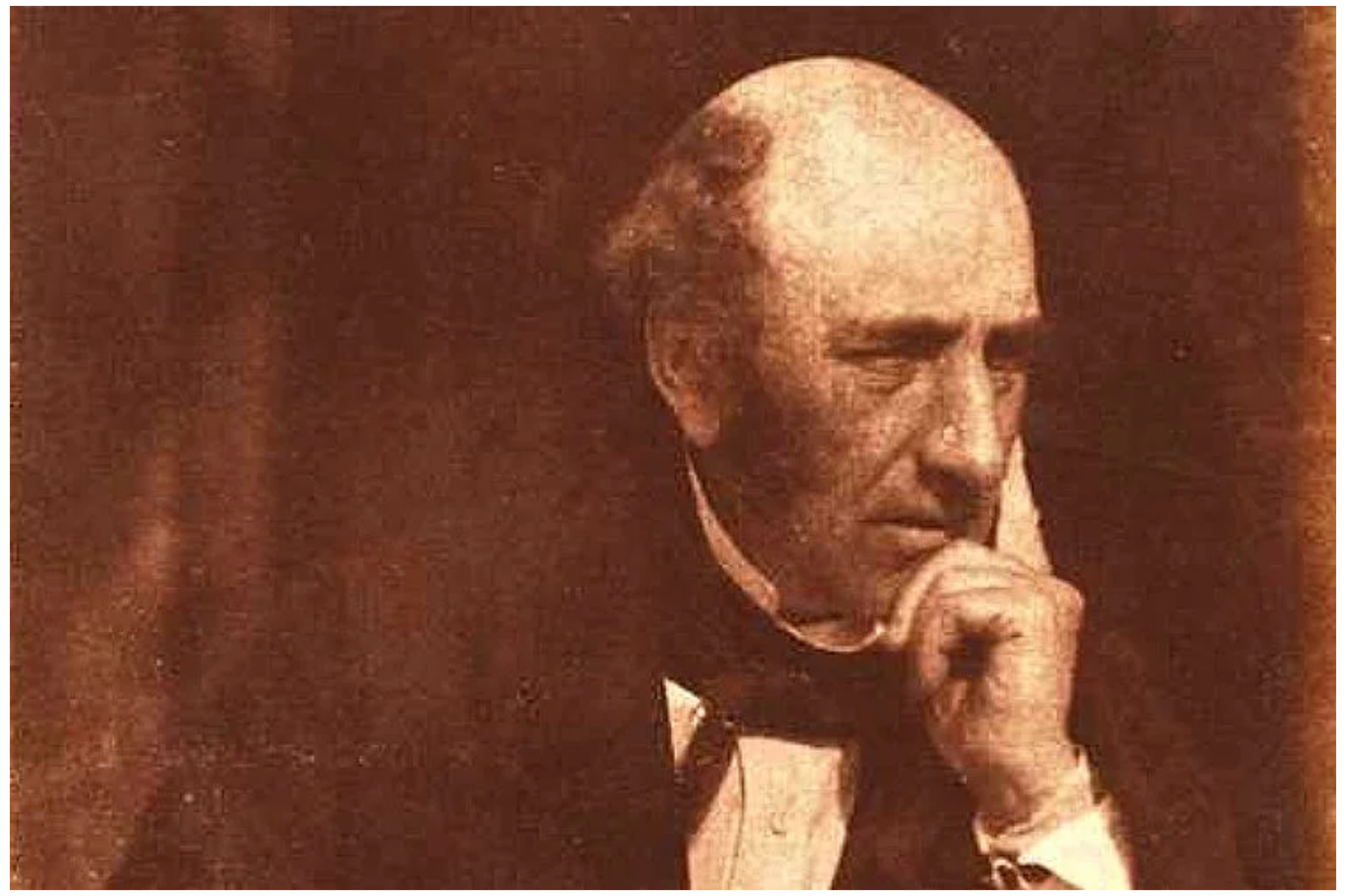


- Switch to deformed/merged code where logical operator to be measured = products of stabilizers[1]
- Measurement errors → need to repeat for $O(d)$ rounds.

[1] Vuillot *et al.* NJP **21** (2019).

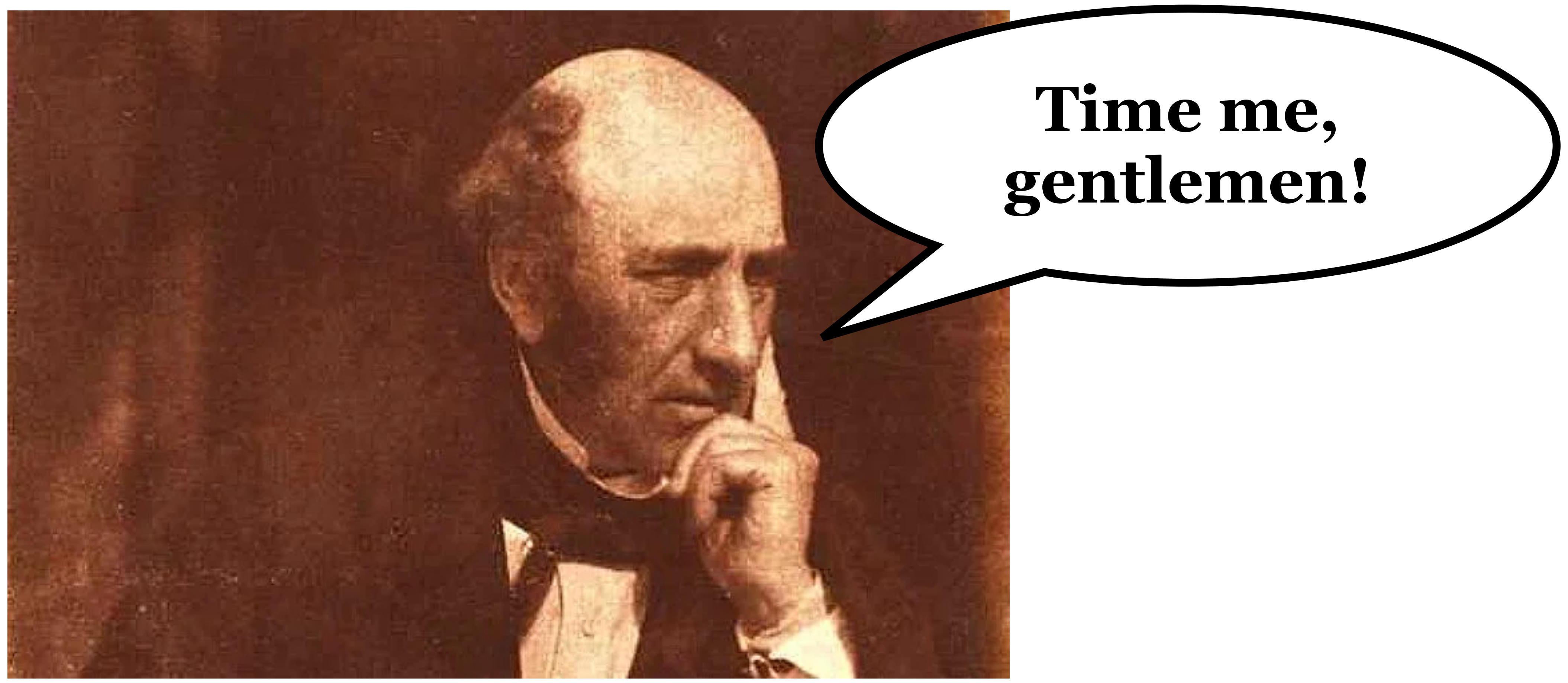
What is fast surgery?

What is fast surgery?



Robert Liston, infamous 17th century Scottish surgeon

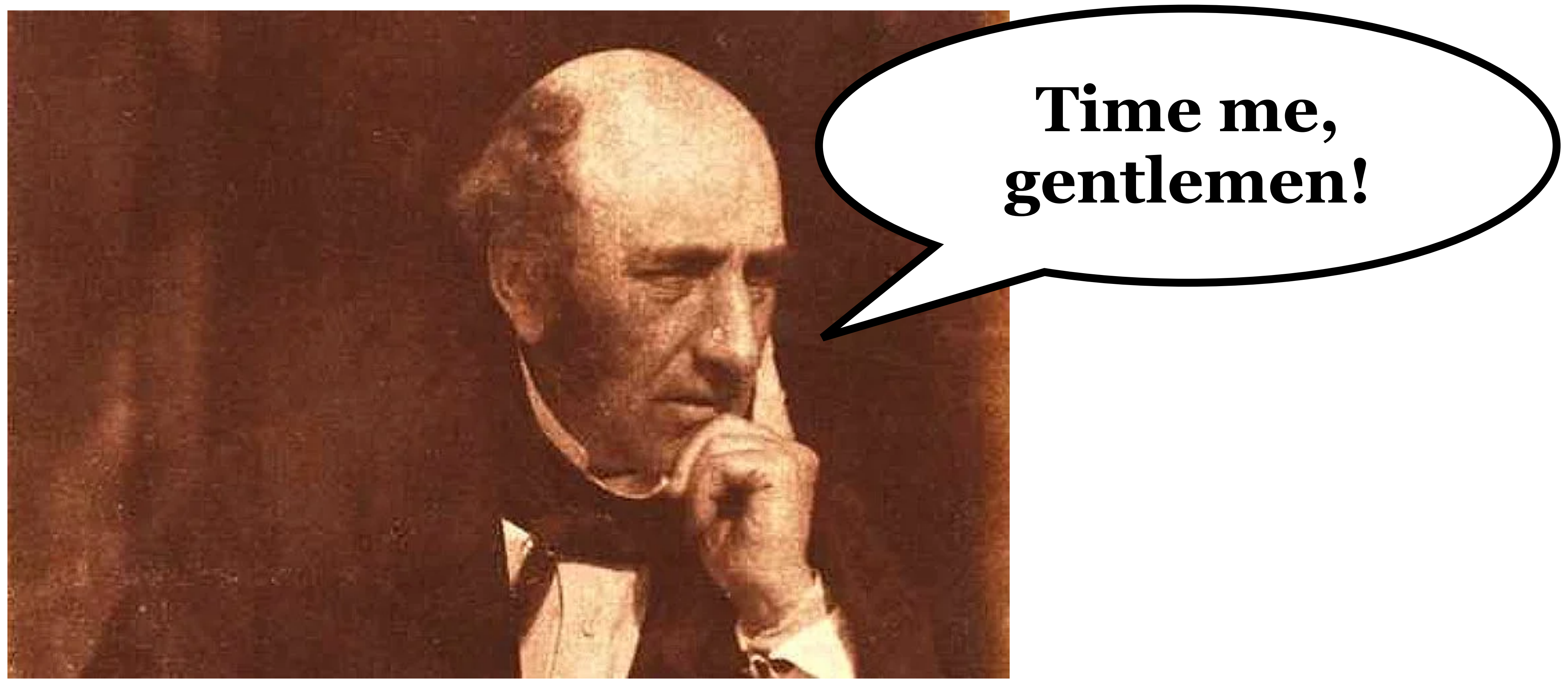
What is fast surgery?



Robert Liston, infamous 17th century Scottish surgeon

2.5 minute leg amputation

What is fast surgery?

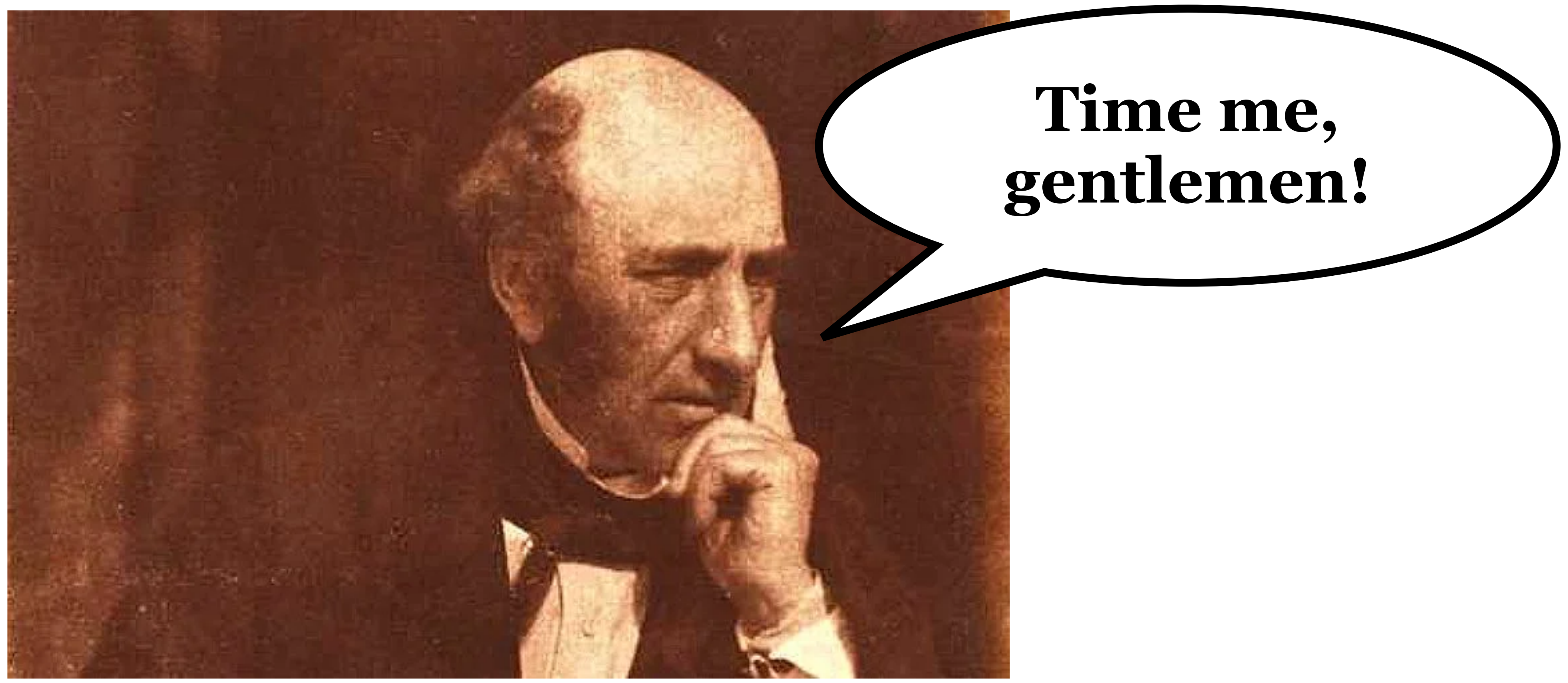


Robert Liston, infamous 17th century Scottish surgeon

2.5 minute leg amputation

The patient and two bystanders died

What is fast surgery?



Robert Liston, infamous 17th century Scottish surgeon

2.5 minute leg amputation

The patient and two bystanders died

300% mortality rate

Regular surgery → Fast surgery

Regular surgery → Fast surgery

3-term chain complex for **regular code surgery, Z measurement**

Regular surgery $\rightarrow$ Fast surgery

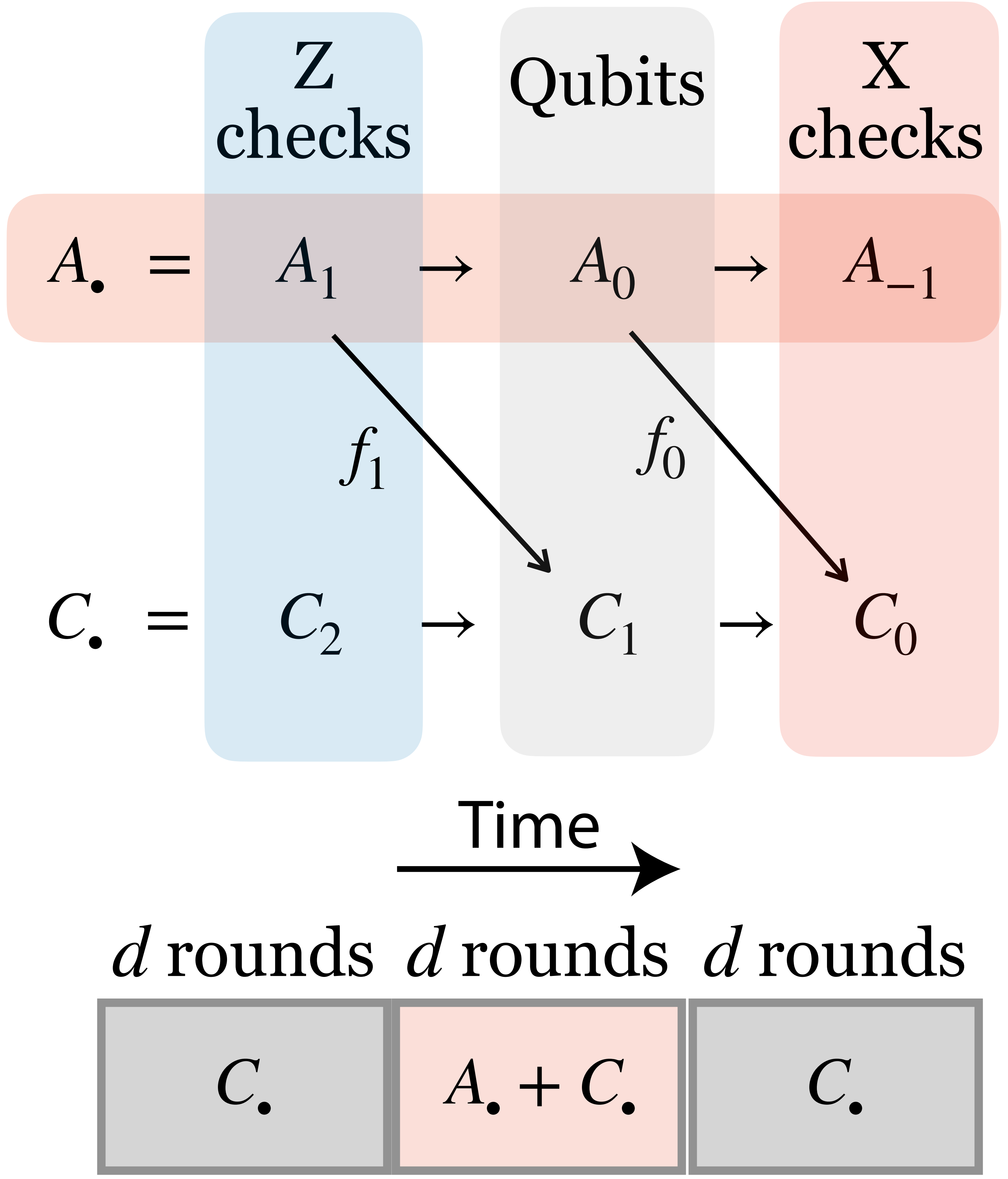
3-term chain complex for **regular code surgery, Z measurement**

	Z checks	Qubits	X checks

$$
\begin{tikzcd}
A_\bullet = & A_1 \arrow[r] \arrow[dr, "f_1"] & A_0 \arrow[r] \arrow[dr, "f_0"] & A_{-1} \\
C_\bullet = & C_2 \arrow[r] & C_1 \arrow[r] & C_0
\end{tikzcd}
$$

Regular surgery → Fast surgery

3-term chain complex for **regular code surgery, Z measurement**



Regular surgery → Fast surgery

3-term chain complex for **regular code surgery, Z measurement**

$$
\begin{tikzcd}
 & \text{Z checks} & \text{Qubits} & \text{X checks} \\
A_\bullet = & A_1 \arrow[r] \arrow[dr, "f_1"] & A_0 \arrow[r] \arrow[dr, "f_0"] & A_{-1} \\
C_\bullet = & C_2 \arrow[r] & C_1 \arrow[r] & C_0
\end{tikzcd}
$$

Time

d rounds	d rounds	d rounds
$C_\bullet$	$A_\bullet + C_\bullet$	$C_\bullet$

4-term chain complex for **fast surgery**

Take a ***4-term*** chain ancilla complex, $A_\bullet$, $f_\bullet : A_\bullet \to C_\bullet$, st:

Regular surgery $\rightarrow$ Fast surgery

3-term chain complex for **regular code surgery, Z measurement**

Z checks	Qubits	X checks

$$\begin{tikzcd} A_\bullet = & A_1 \arrow[r] \arrow[dr, "f_1"] & A_0 \arrow[r] \arrow[dr, "f_0"] & A_{-1} \\ C_\bullet = & C_2 \arrow[r] & C_1 \arrow[r] & C_0 \end{tikzcd}$$

Time $\longrightarrow$

d rounds	d rounds	d rounds
$C_\bullet$	$A_\bullet + C_\bullet$	$C_\bullet$

4-term chain complex for **fast surgery**

Take a ***4-term*** chain ancilla complex, $A_\bullet$, $f_\bullet : A_\bullet \rightarrow C_\bullet$, st:

$$\begin{tikzcd} A_\bullet = & A_2 \arrow[r] \arrow[dr, "f_2"] & A_1 \arrow[r] \arrow[dr, "f_1"] & A_0 \arrow[r] \arrow[dr, "f_0"] & A_{-1} \\ C_\bullet = & C_2 \arrow[r] & C_1 \arrow[r] & C_0 & \end{tikzcd}$$

Regular surgery $\rightarrow$ Fast surgery

3-term chain complex for **regular code surgery, Z measurement**

$$\begin{tikzcd}[row sep=large]
& \text{Z checks} & \text{Qubits} & \text{X checks} \\
A_\bullet = & A_1 \arrow[r] \arrow[dr, "f_1"] & A_0 \arrow[r] \arrow[dr, "f_0"] & A_{-1} \\
C_\bullet = & C_2 \arrow[r] & C_1 \arrow[r] & C_0
\end{tikzcd}$$

Time $\longrightarrow$

d rounds	d rounds	d rounds
$C_\bullet$	$A_\bullet + C_\bullet$	$C_\bullet$

4-term chain complex for **fast surgery**

Take a ***4-term*** chain ancilla complex, $A_\bullet$, $f_\bullet : A_\bullet \rightarrow C_\bullet$, st:

$$\begin{tikzcd}[row sep=large]
& \text{Meta-checks} & \text{Z checks} & \text{Qubits} & \text{X checks} \\
A_\bullet = & A_2 \arrow[r] \arrow[dr, "f_2"] & A_1 \arrow[r] \arrow[dr, "f_1"] & A_0 \arrow[r] \arrow[dr, "f_0"] & A_{-1} \\
& C_\bullet = & C_2 \arrow[r] & C_1 \arrow[r] & C_0
\end{tikzcd}$$

Regular surgery → Fast surgery

3-term chain complex for **regular code surgery, Z measurement**

Z checks | Qubits | X checks

$$\begin{tikzcd} A_\bullet = & A_1 \arrow[r] \arrow[dr, "f_1"] & A_0 \arrow[r] \arrow[dr, "f_0"] & A_{-1} \\ C_\bullet = & C_2 \arrow[r] & C_1 \arrow[r] & C_0 \end{tikzcd}$$

Time →

d rounds	d rounds	d rounds
$C_\bullet$	$A_\bullet + C_\bullet$	$C_\bullet$

4-term chain complex for **fast surgery**

Take a ***4-term*** chain ancilla complex, $A_\bullet$, $f_\bullet : A_\bullet \to C_\bullet$, st:

Meta-checks | Z checks | Qubits | X checks

$$\begin{tikzcd} A_\bullet = & A_2 \arrow[r] \arrow[dr, "f_2"] & A_1 \arrow[r] \arrow[dr, "f_1"] & A_0 \arrow[r] \arrow[dr, "f_0"] & A_{-1} \\ & C_\bullet = & C_2 \arrow[r] & C_1 \arrow[r] & C_0 \end{tikzcd}$$

Detects measurement errors

Regular surgery → Fast surgery

3-term chain complex for **regular code surgery, Z measurement**

Z checks	Qubits	X checks

$$\begin{tikzcd} A_\bullet = A_1 \arrow[r] \arrow[dr, "f_1"] & A_0 \arrow[r] \arrow[dr, "f_0"] & A_{-1} \\ C_\bullet = C_2 \arrow[r] & C_1 \arrow[r] & C_0 \end{tikzcd}$$

Time →

d rounds	d rounds	d rounds
$C_\bullet$	$A_\bullet + C_\bullet$	$C_\bullet$

4-term chain complex for **fast surgery**

Take a ***4-term*** chain ancilla complex, $A_\bullet$, $f_\bullet : A_\bullet \to C_\bullet$, st:

Meta-checks	Z checks	Qubits	X checks

$$\begin{tikzcd} A_\bullet = A_2 \arrow[r] \arrow[dr, "f_2"] & A_1 \arrow[r] \arrow[dr, "f_1"] & A_0 \arrow[r] \arrow[dr, "f_0"] & A_{-1} \\ C_\bullet = & C_2 \arrow[r] & C_1 \arrow[r] & C_0 \end{tikzcd}$$

Detects measurement errors

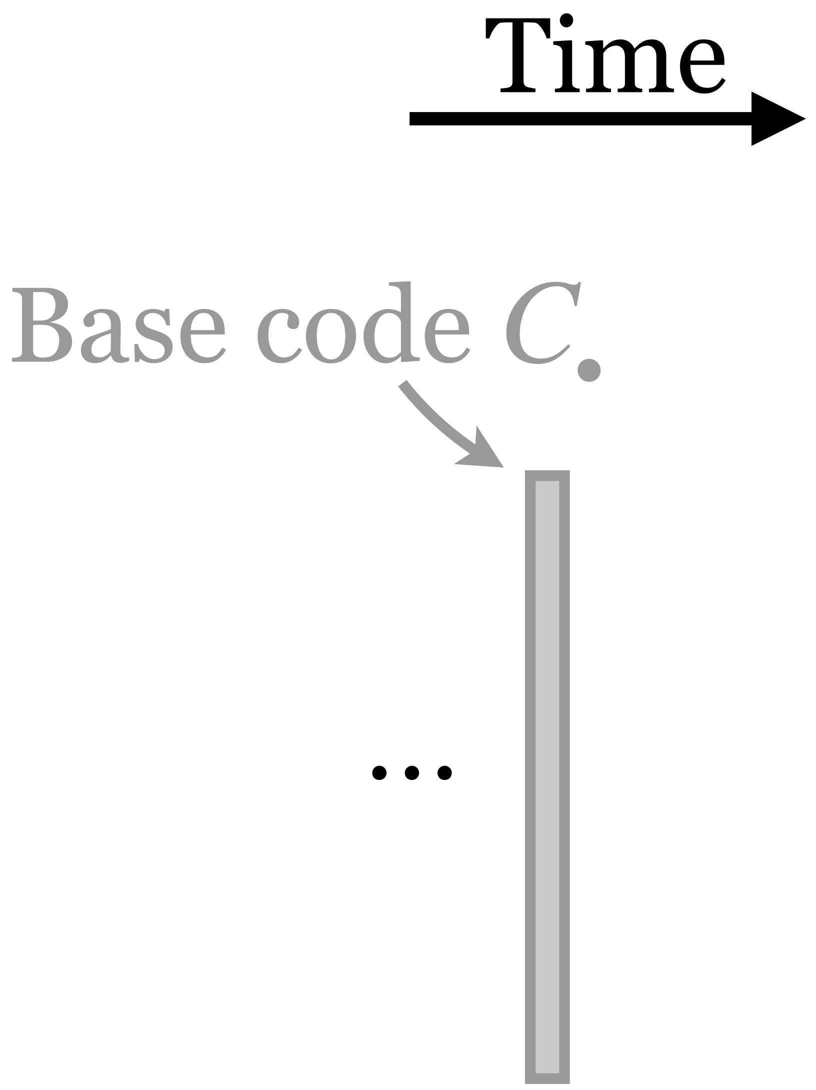
What is the surgery procedure now?

Want fault-distance d for data qubit & measurement errors

Is this fault-tolerant?

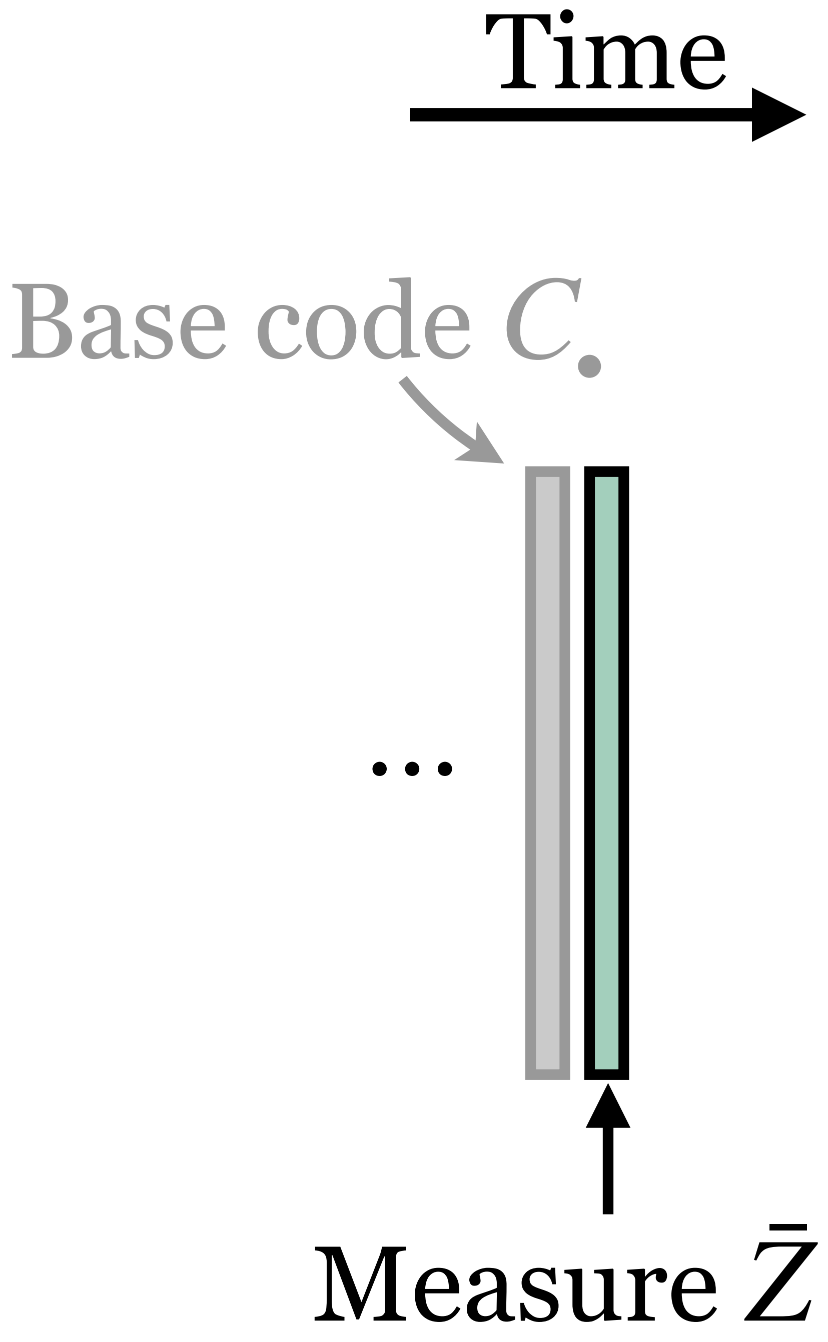
Is this fault-tolerant?

1. 1 round in the base code $C_\bullet$



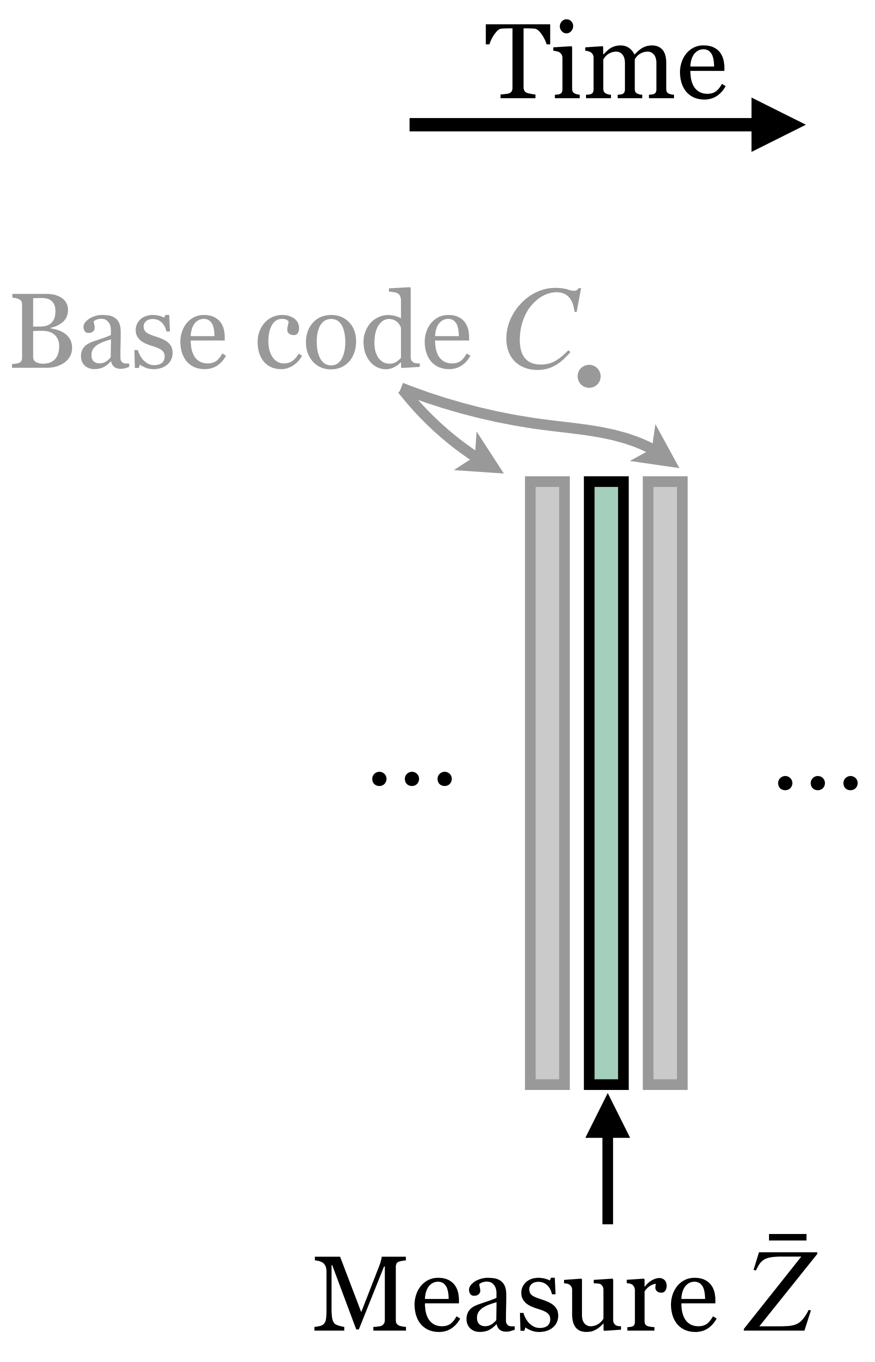
Is this fault-tolerant?

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



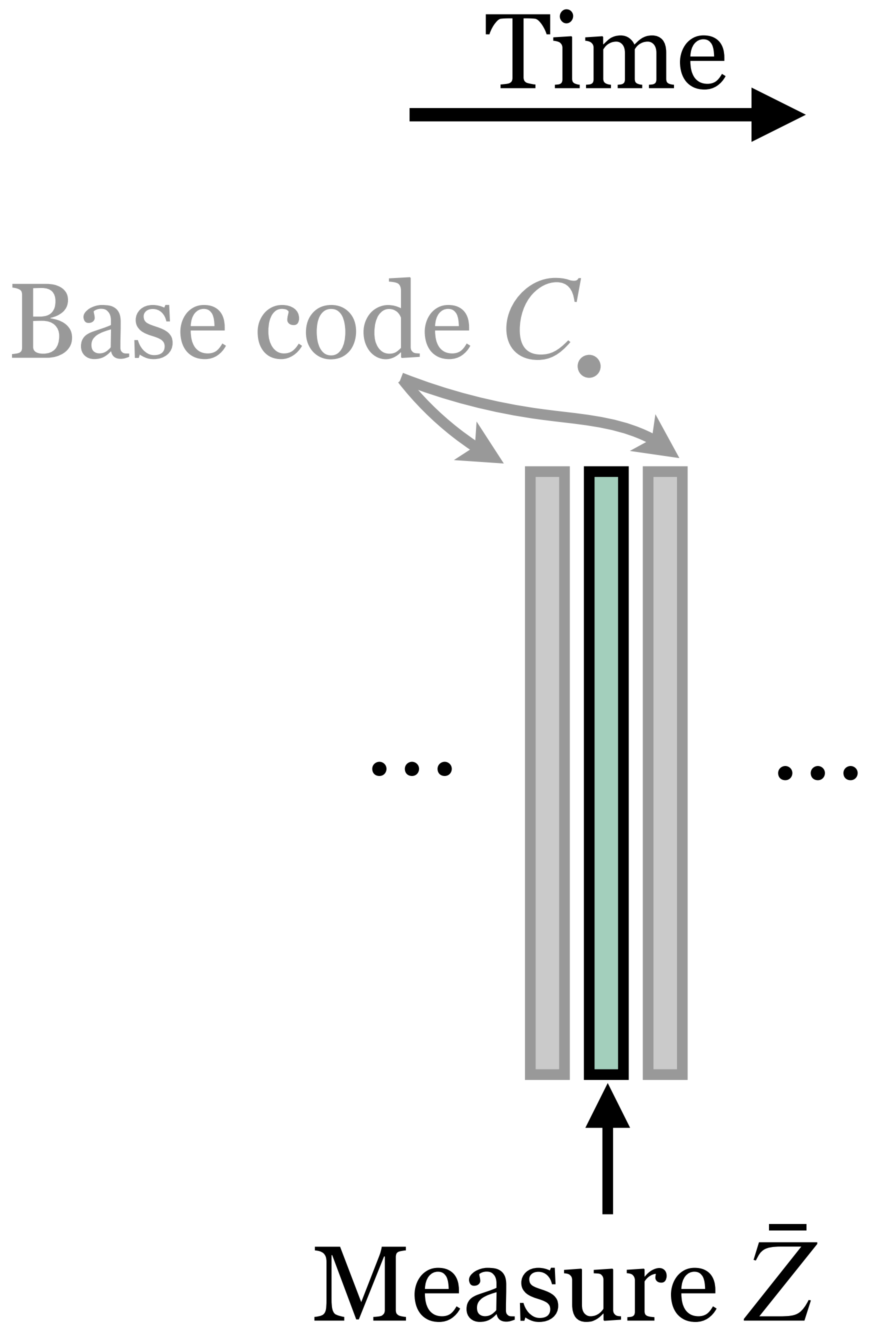
Is this fault-tolerant?

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 $C_\bullet$



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 $C_\bullet$



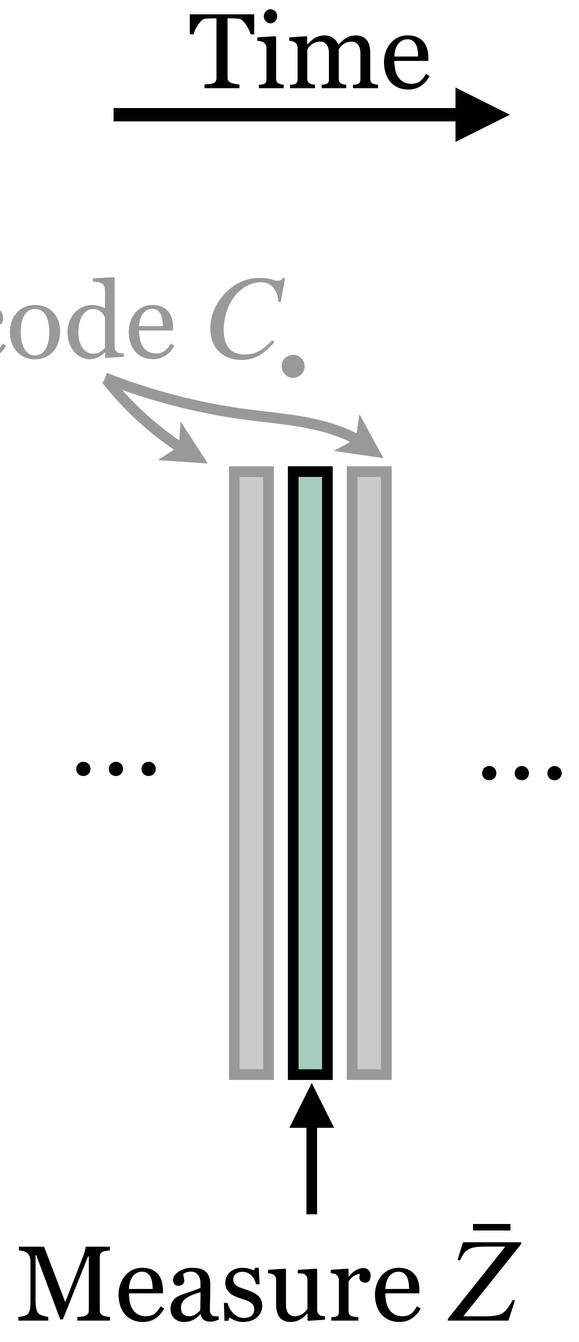
Measure $\bar{Z}$

Meta-checks | Z checks | Qubits | X checks

$$
\begin{tikzcd}
A_\bullet = & A_2 \arrow[r] \arrow[dr, "f_2"'] & A_1 \arrow[r] \arrow[dr, "f_1"'] & A_0 \arrow[r] \arrow[dr, "f_0"'] & A_{-1} \\
C_\bullet = & & C_2 \arrow[r] & C_1 \arrow[r] & C_0
\end{tikzcd}
$$

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 $|+\rangle$

Measure $\bar{Z}$

Meta-checks	Z checks	Qubits	X checks

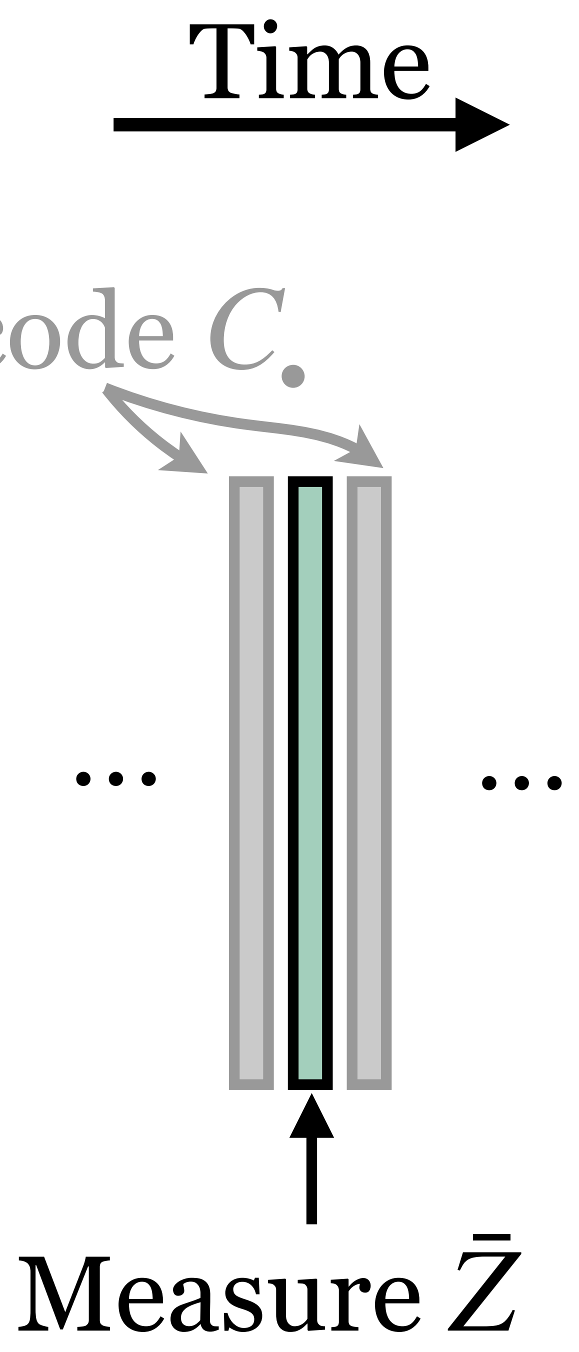
$$A_\bullet = A_2 \to A_1 \to A_0 \to A_{-1}$$

$$f_2,\ f_1,\ f_0$$

$$C_\bullet = C_2 \to C_1 \to C_0$$

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 $|+\rangle$

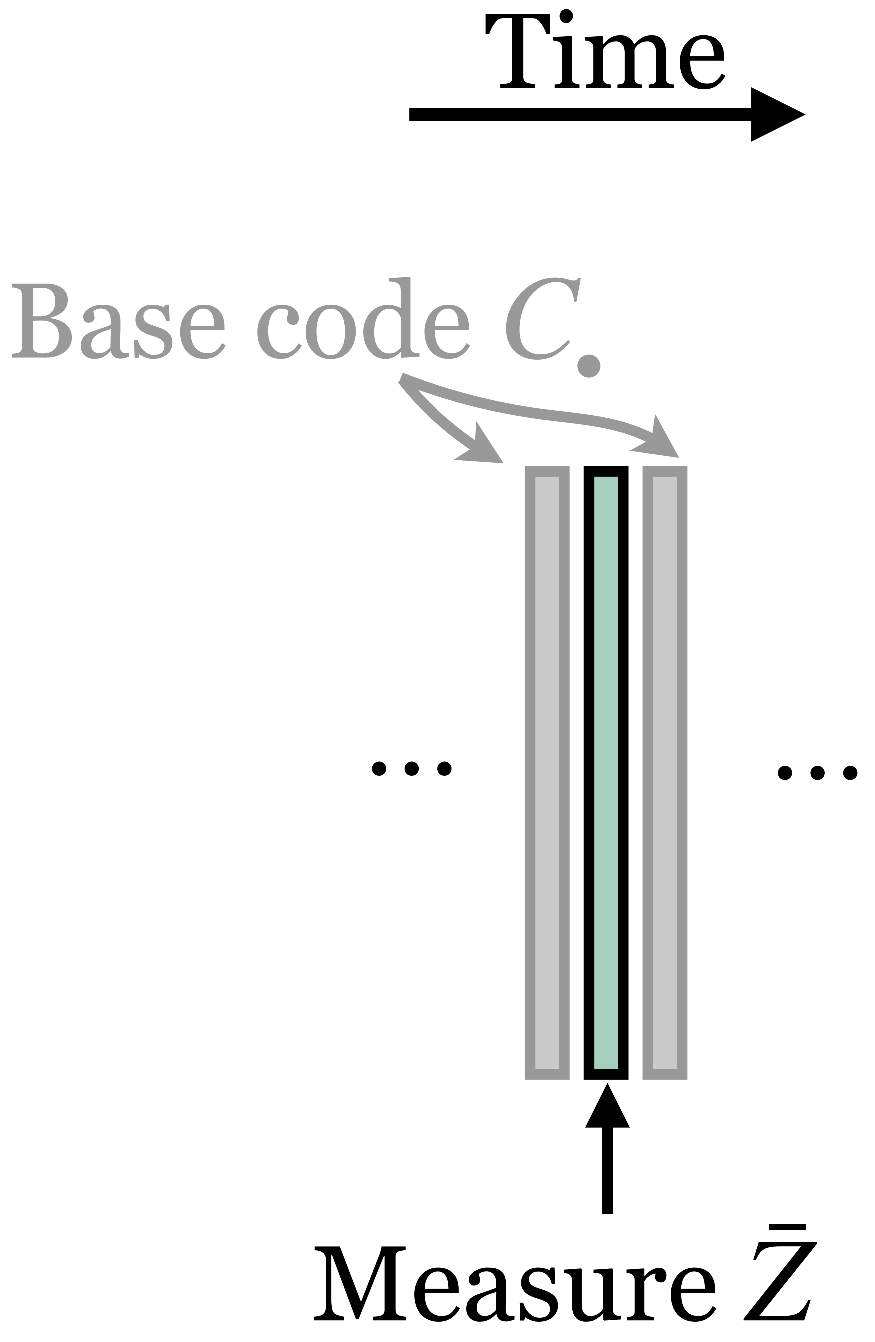
Finish: measure A_0 qubits in X

Measure $\bar{Z}$

$$\begin{tikzcd}
 & \text{Meta-checks} & \text{Z checks} & \text{Qubits} & \text{X checks} \\
A_\bullet = & A_2 \arrow[r] \arrow[dr, "f_2"] & A_1 \arrow[r] \arrow[dr, "f_1"] & A_0 \arrow[r] \arrow[dr, "f_0"] & A_{-1} \\
 & C_\bullet = & C_2 \arrow[r] & C_1 \arrow[r] & C_0
\end{tikzcd}$$

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 $\bar{Z}$

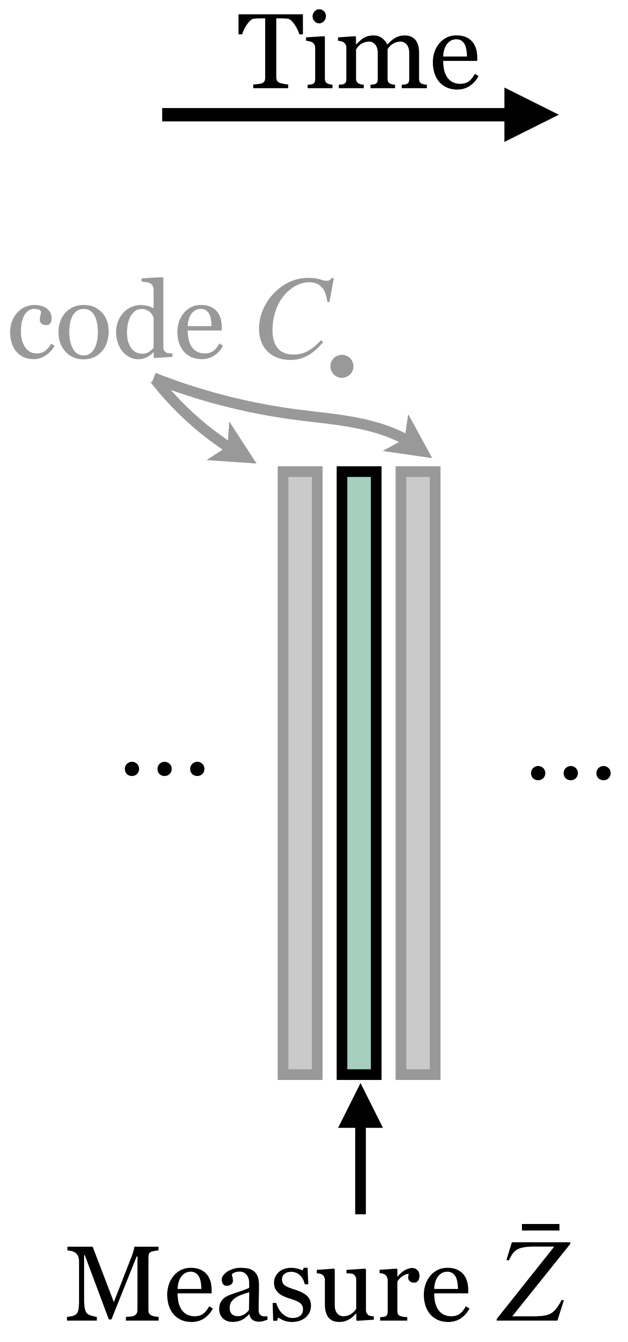
Meta-checks	Z checks	Qubits	X checks

$$\begin{tikzcd} A_\bullet = & A_2 \arrow[r] \arrow[dr, "f_2"'] & A_1 \arrow[r] \arrow[dr, "f_1"'] & A_0 \arrow[r] \arrow[dr, "f_0"'] & A_{-1} \\ C_\bullet = & & C_2 \arrow[r] & C_1 \arrow[r] & C_0 \end{tikzcd}$$

Updated after X measurement, *random*.

Is this fault-tolerant?

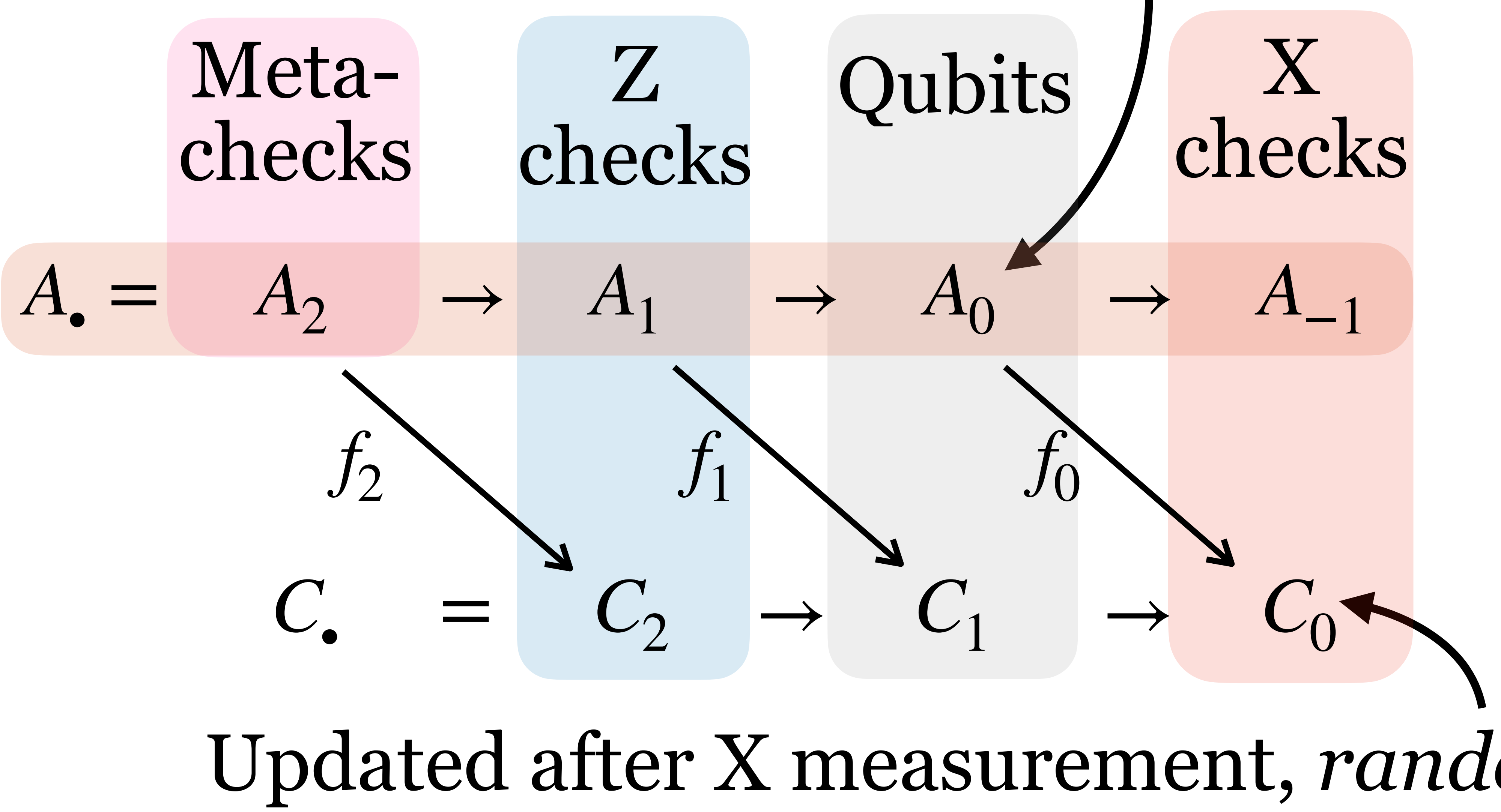
1. 1 round in the base code $C_\bullet$
2. 1 round in the deformed code (measure $\bar{Z}$)
3. 1 round in the base code $C_\bullet$



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

Finish: measure A_0 qubits in X

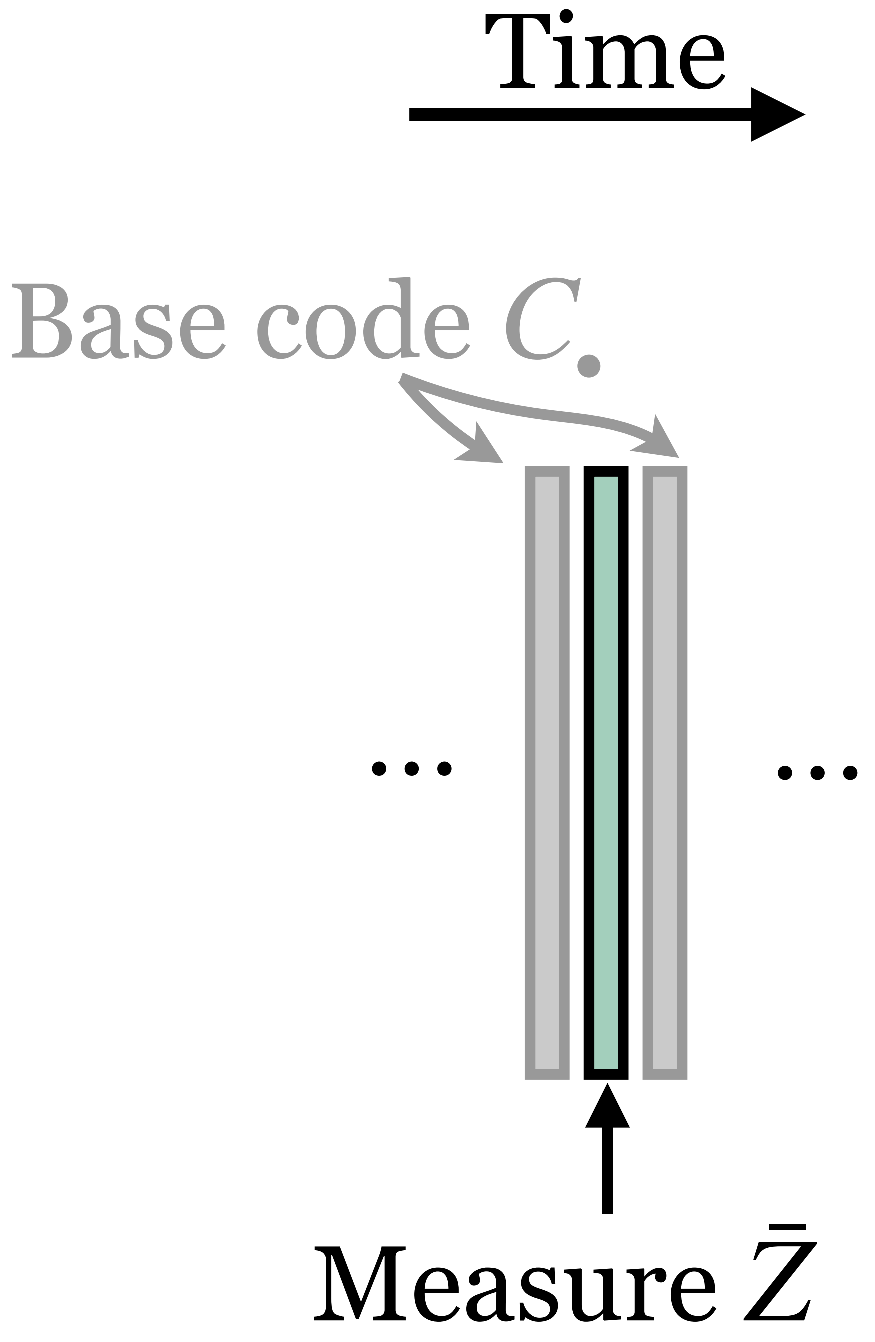
Measure $\bar{Z}$



- State preparation-like error. Pauli frame is random after surgery.

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 $\bar{Z}$

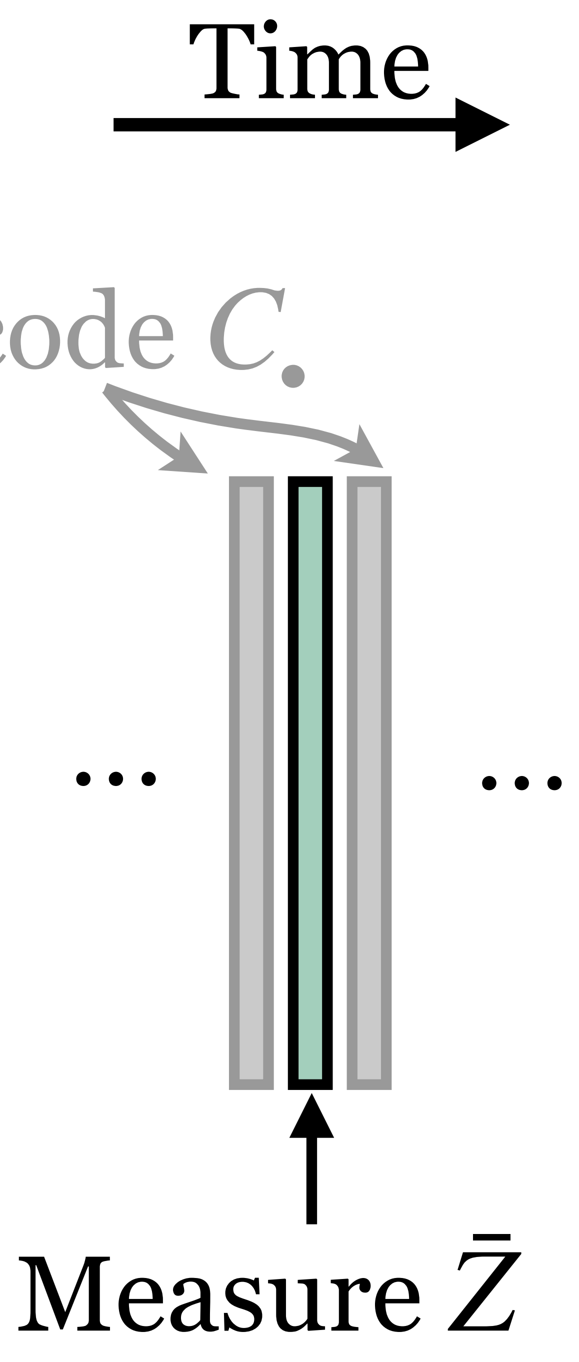
	Meta-checks		Z checks		Qubits		X checks
$A_\bullet =$	A_2	$\rightarrow$	A_1	$\rightarrow$	A_0	$\rightarrow$	A_{-1}

$$\begin{tikzcd} A_2 \arrow[r] \arrow[dr, "f_2"] & A_1 \arrow[r] \arrow[dr, "f_1"] & A_0 \arrow[r] \arrow[dr, "f_0"] & A_{-1} \\ C_\bullet = & C_2 \arrow[r] & C_1 \arrow[r] & C_0 \end{tikzcd}$$

Updated after X measurement, *random*.

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

Is this fault-tolerant?

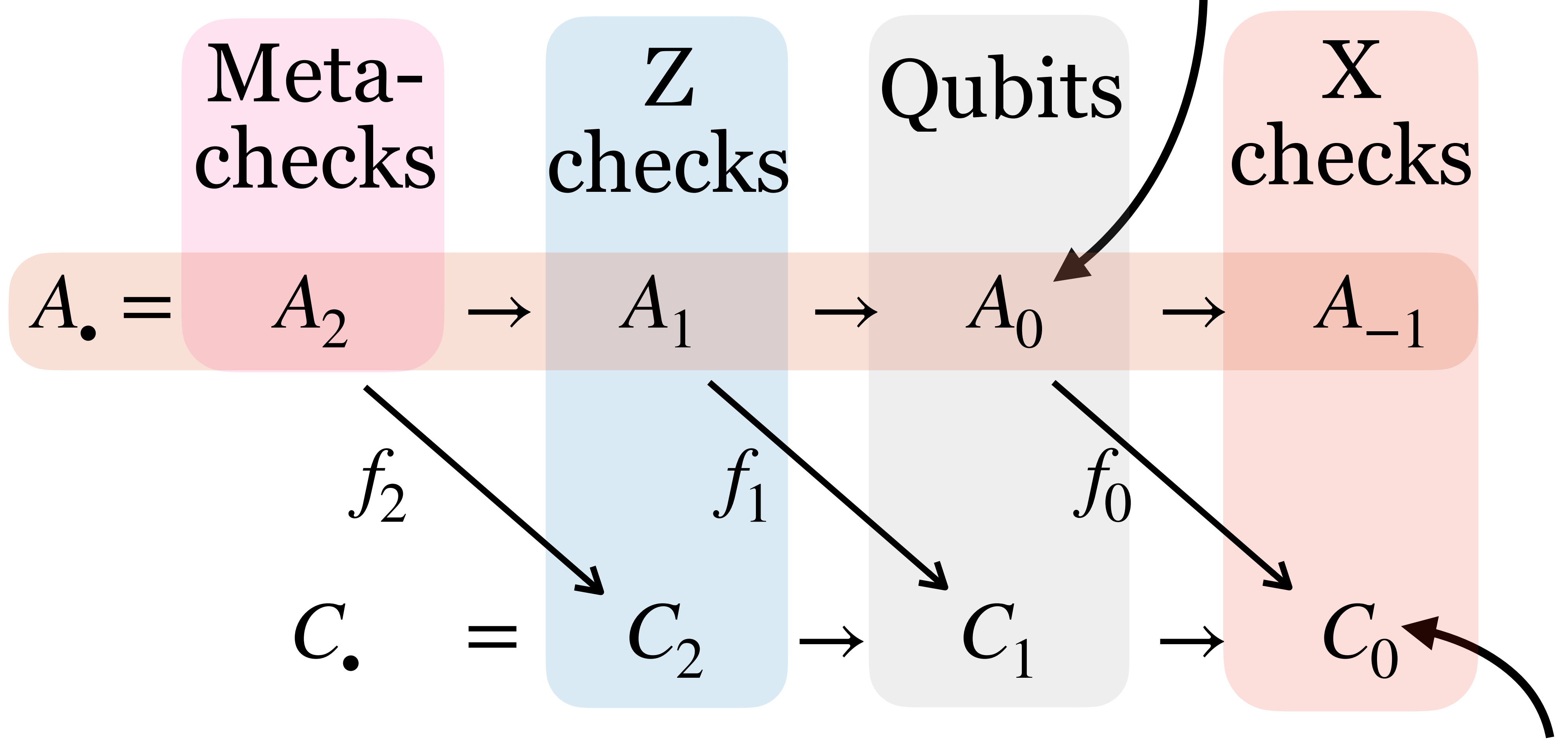


1. 1 round in the base code $C_\bullet$
2. 1 round in the deformed code (measure $\bar{Z}$)
3. 1 round in the base code $C_\bullet$

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

Finish: measure A_0 qubits in X

Measure $\bar{Z}$

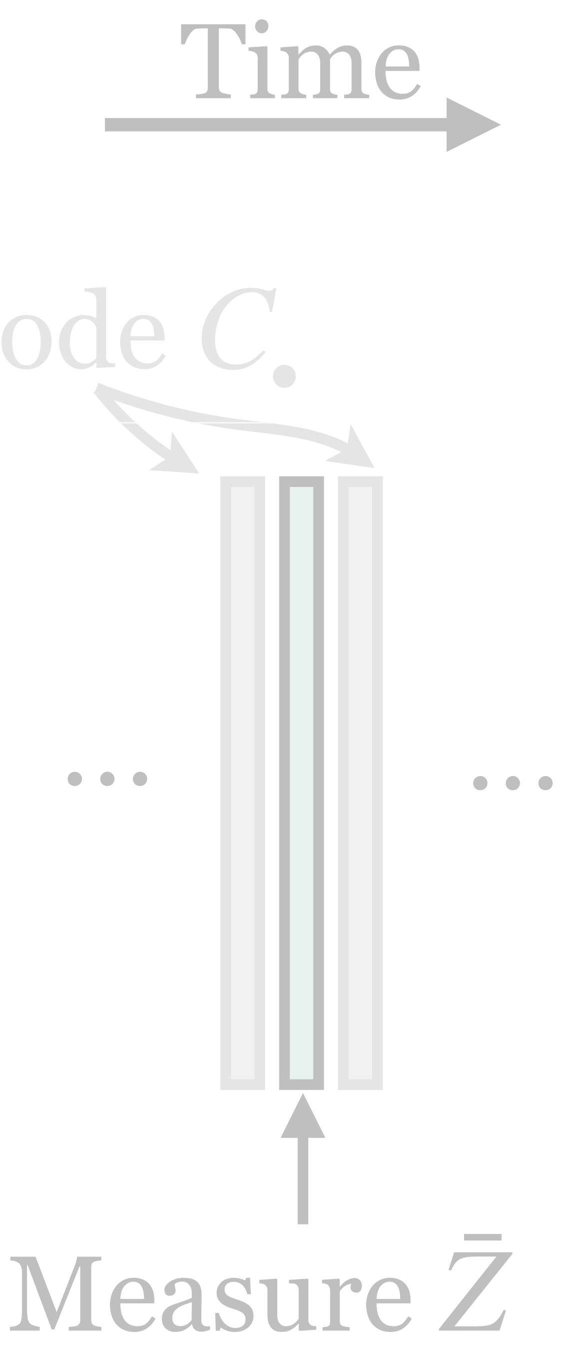


Updated after X measurement, *random*.

- 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

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 $C_\bullet$



Base code $C_\bullet$ must infer syndromes w/
1 round of syndrome extraction

Satisfied if single-shot state preparation

Start: init. A_0 qubits

Finish: measure A_0 qu

Measure $\bar{Z}$

Meta-checks

Z checks

$A_\bullet = A_2 \rightarrow A_1 \rightarrow A_0 \rightarrow A_{-1}$

f_2 f_1 f_0

$C_\bullet = C_2 \rightarrow C_1 \rightarrow C_0$

Updated after X measurement, *random*.

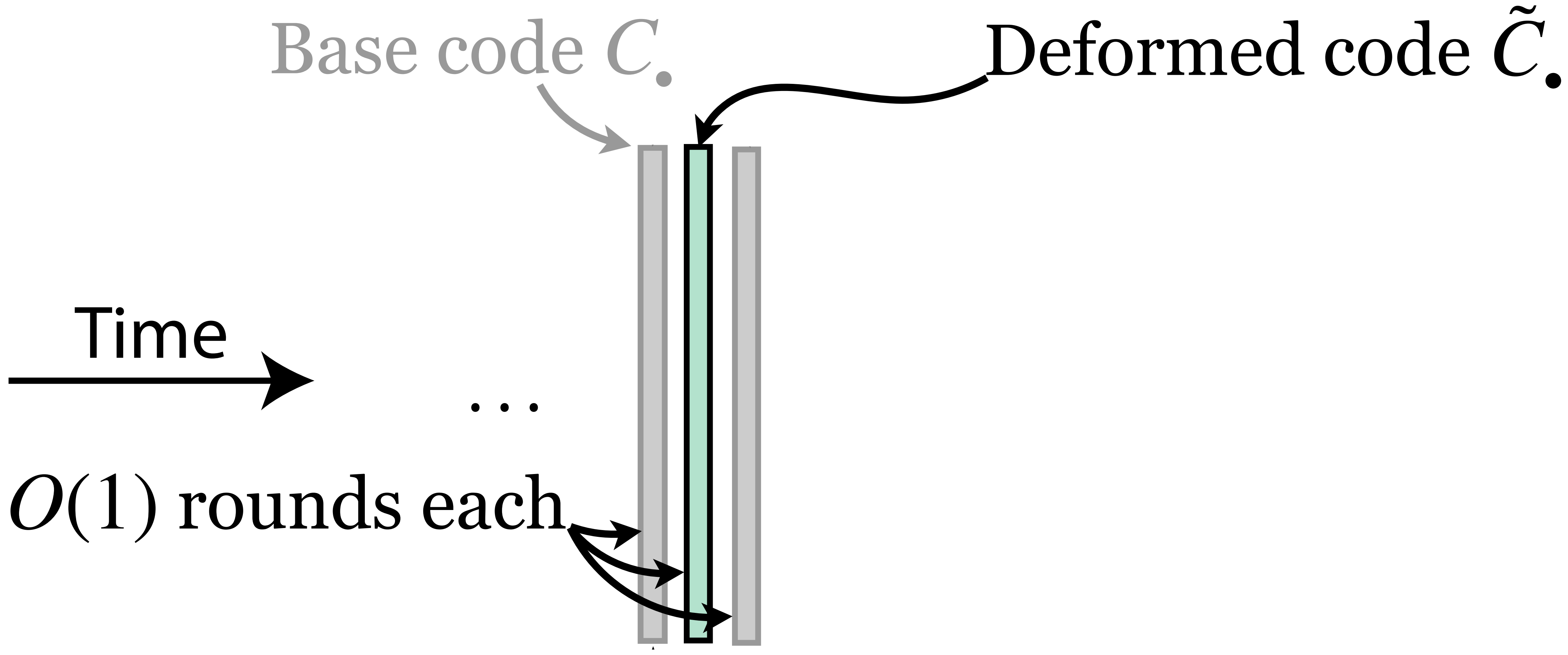
error. Pauli frame is random after surgery.

- Only 1 round in $C_\bullet$ to learn the new stabilizer values → incorrectly inferred Pauli frame
- Can affect future logical operations

Single-shot surgery [1-3]

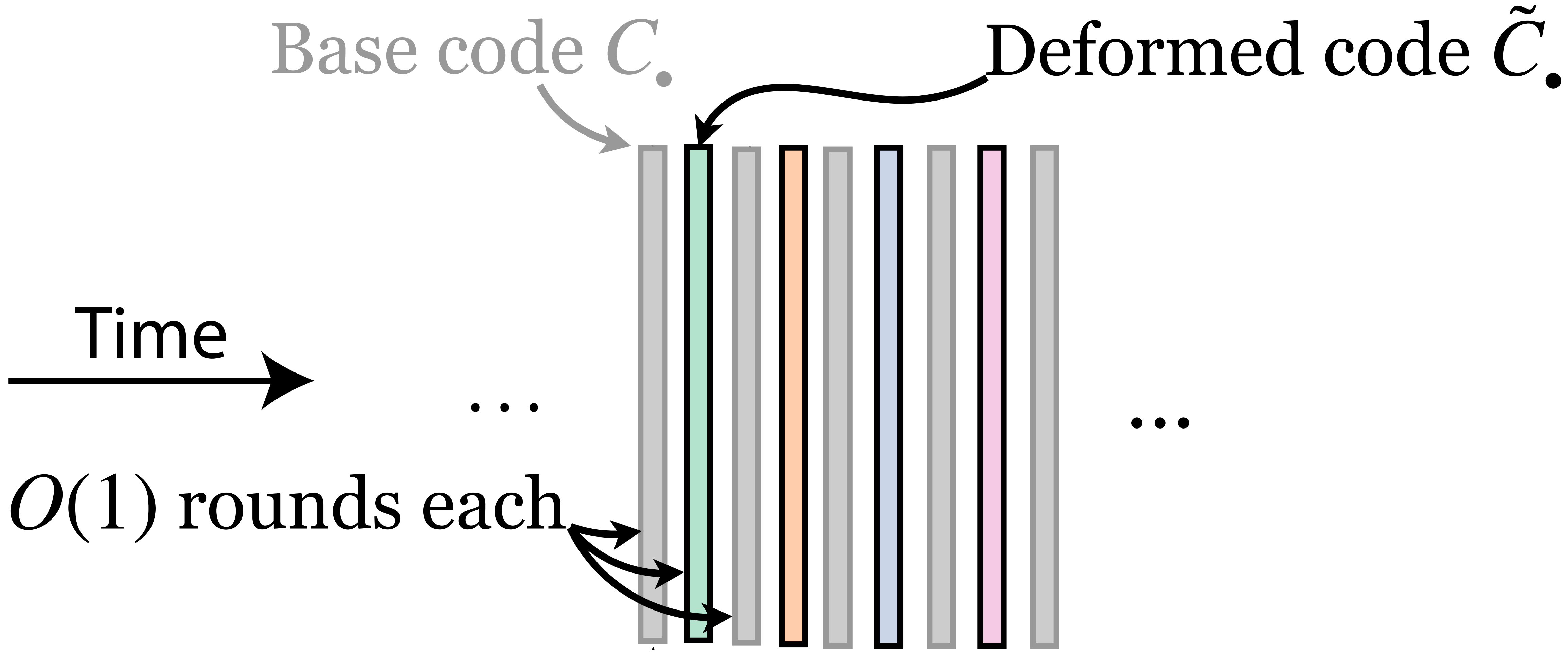
[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]

Single-shot surgery [1-3]



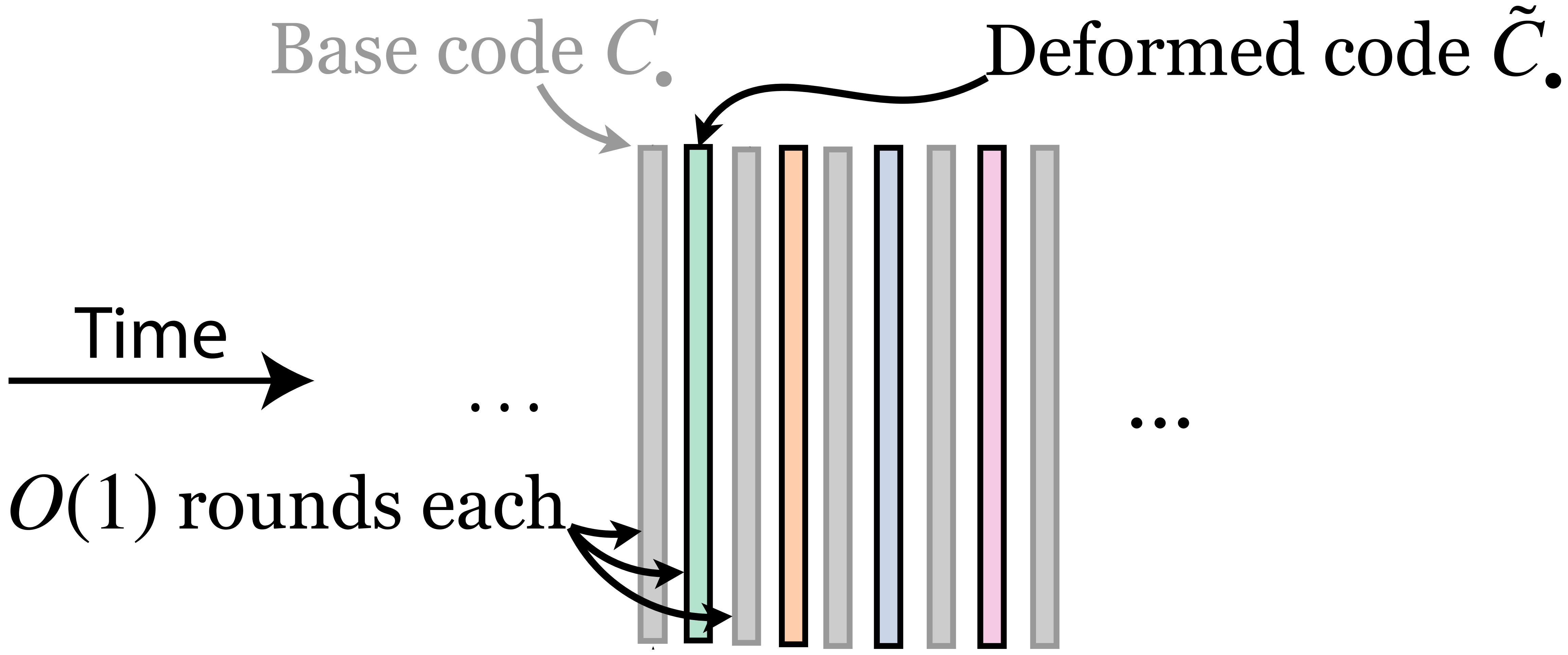
[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]

Single-shot surgery [1-3]



[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]

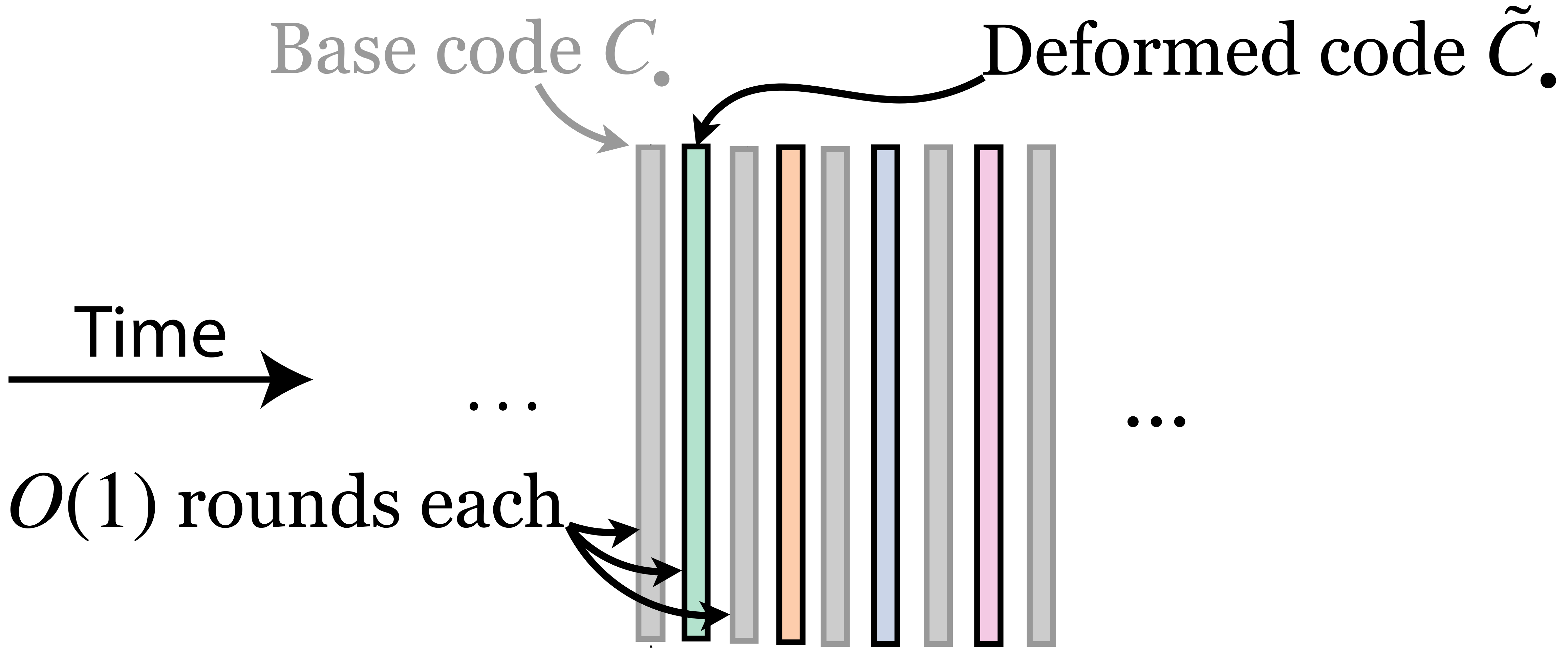
Single-shot surgery [1-3]



- Requires **single-shot preparation** of **deformed code**, and **base code** $C_\bullet$.

[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]

Single-shot surgery [1-3]

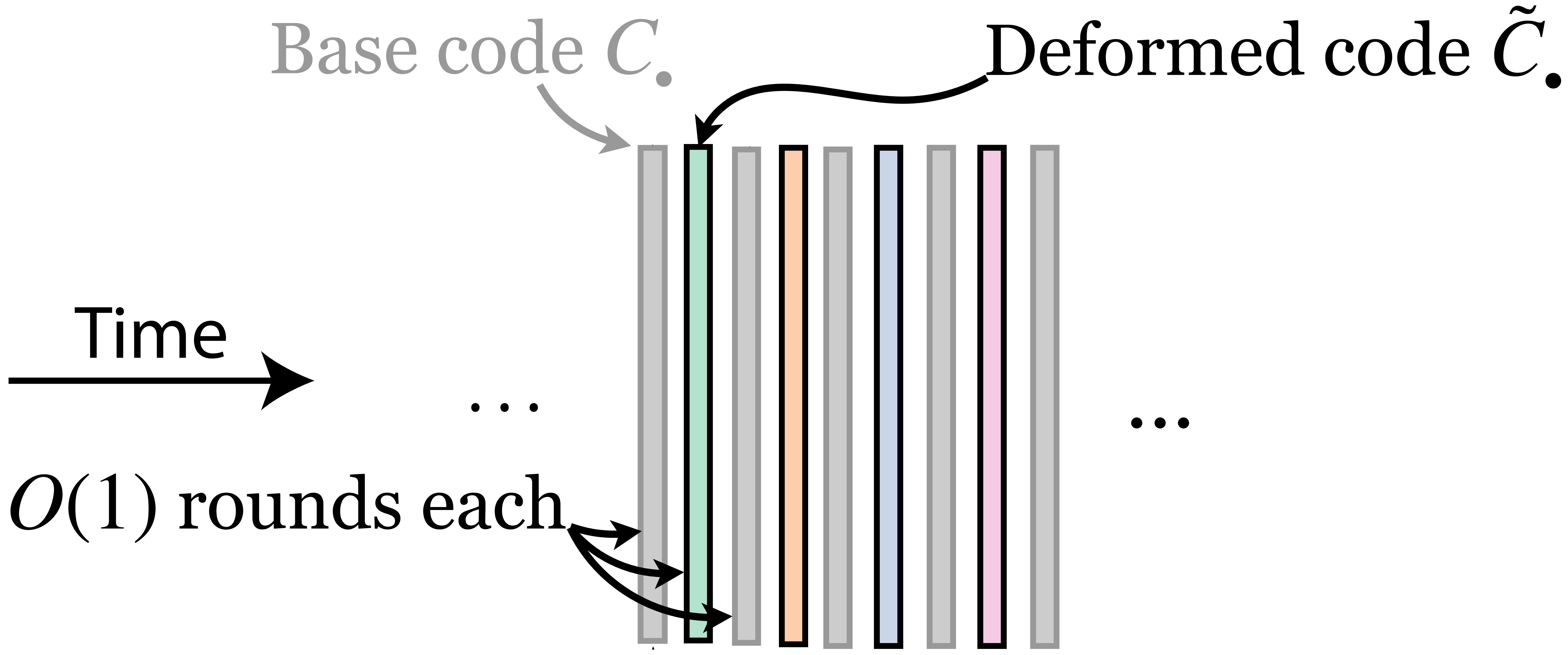


- Requires **single-shot preparation** of **deformed code**, and **base code** $C_\bullet$.

- Restricts what $C_\bullet$ can be.
  - Base code must have **meta-checks**

[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]

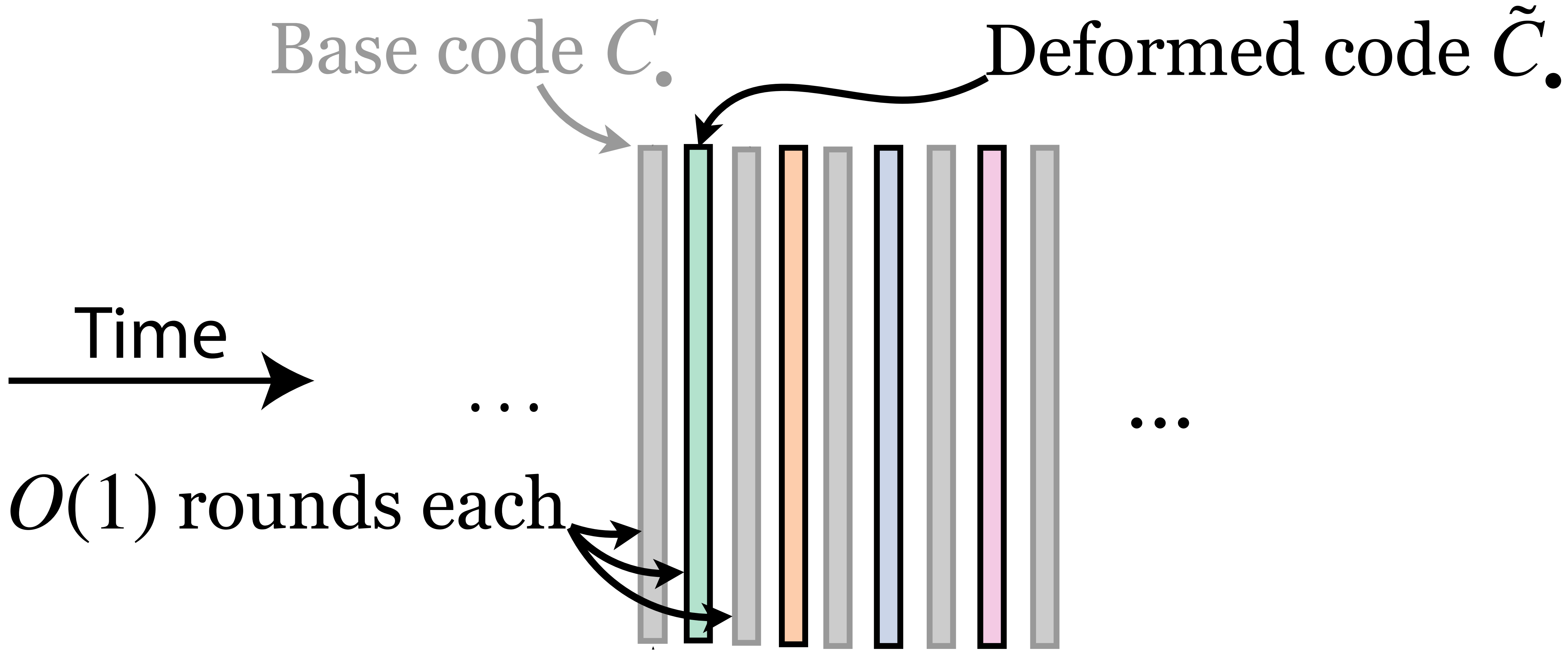
Single-shot surgery [1-3]



- 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: $\tilde{O}(nkd)$

[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]

Single-shot surgery [1-3]



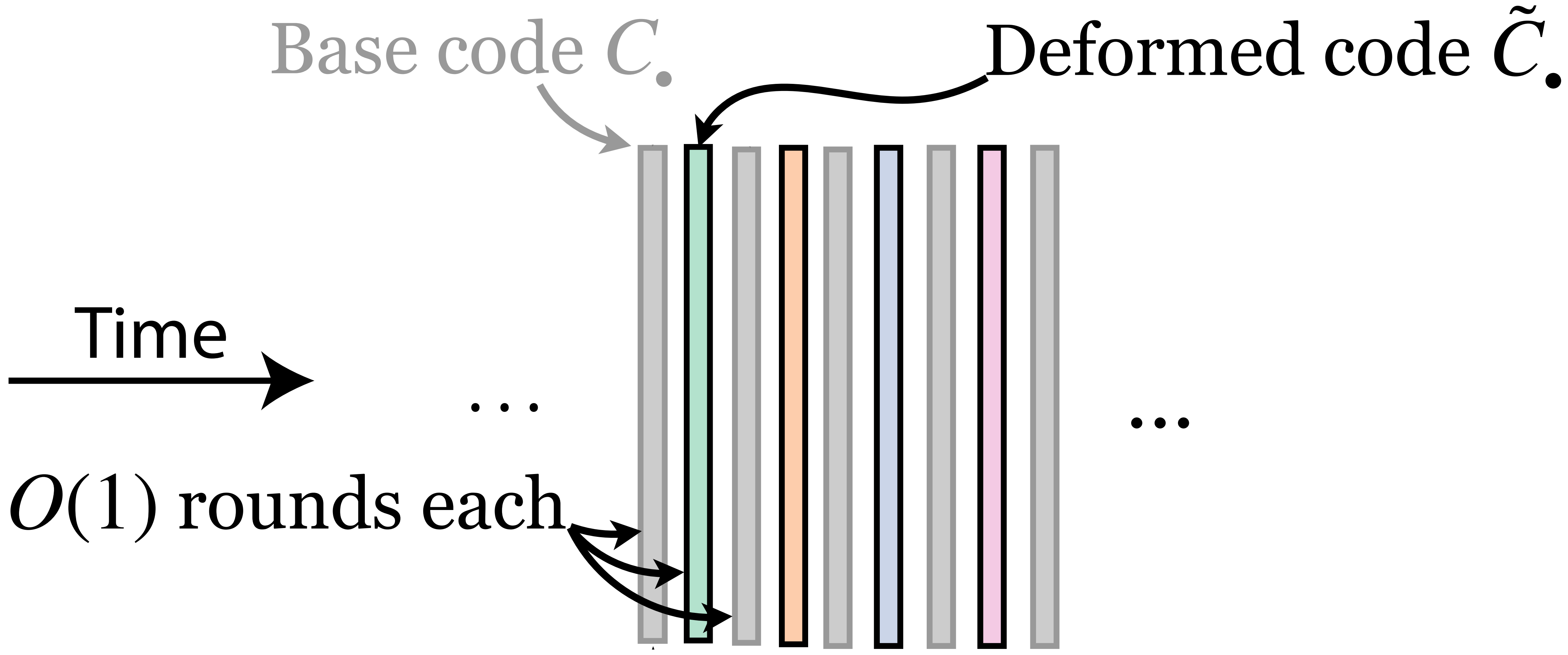
- Requires **single-shot preparation** of **deformed code**, and **base code** $C_\bullet$.
- Restricts what $C_\bullet$ can be.
  - Base code must have **meta-checks**
- Concurrent work[3]: deformed $\tilde{C}_\bullet$ may introduce large space overhead: $\tilde{O}(nkd)$

Constant-time surgery [4, 5]

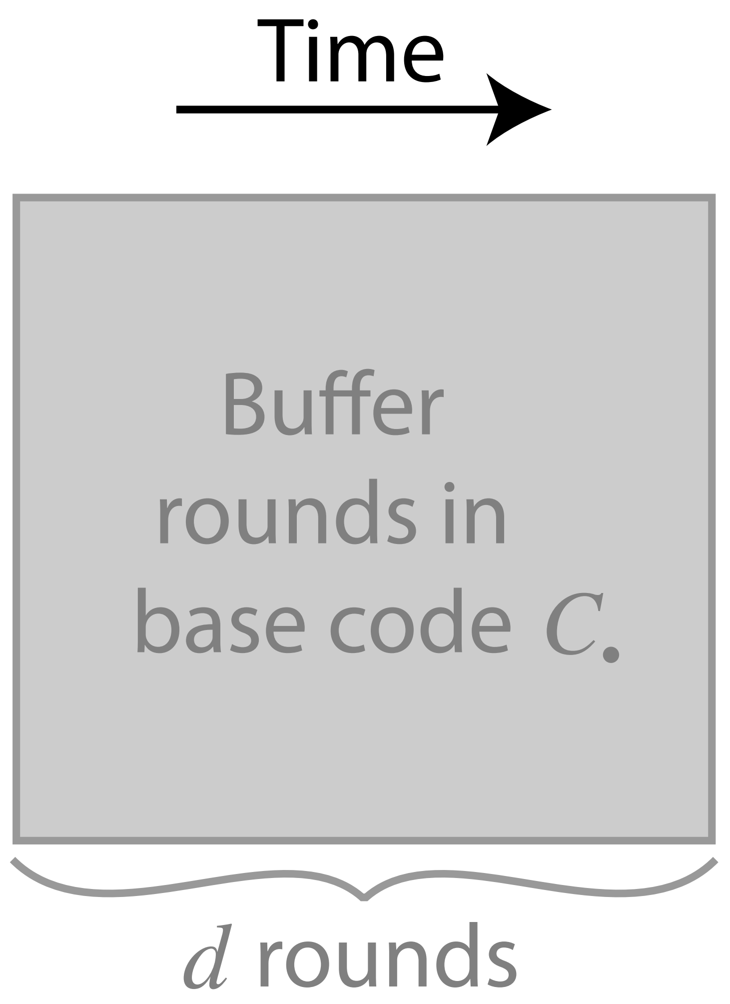
[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]
[4] Cowtan, *et al.* [2510.14895]
[5] **KC**, He, *et al.* [2603.02157]

Single-shot surgery [1-3]

Constant-time surgery [4, 5]

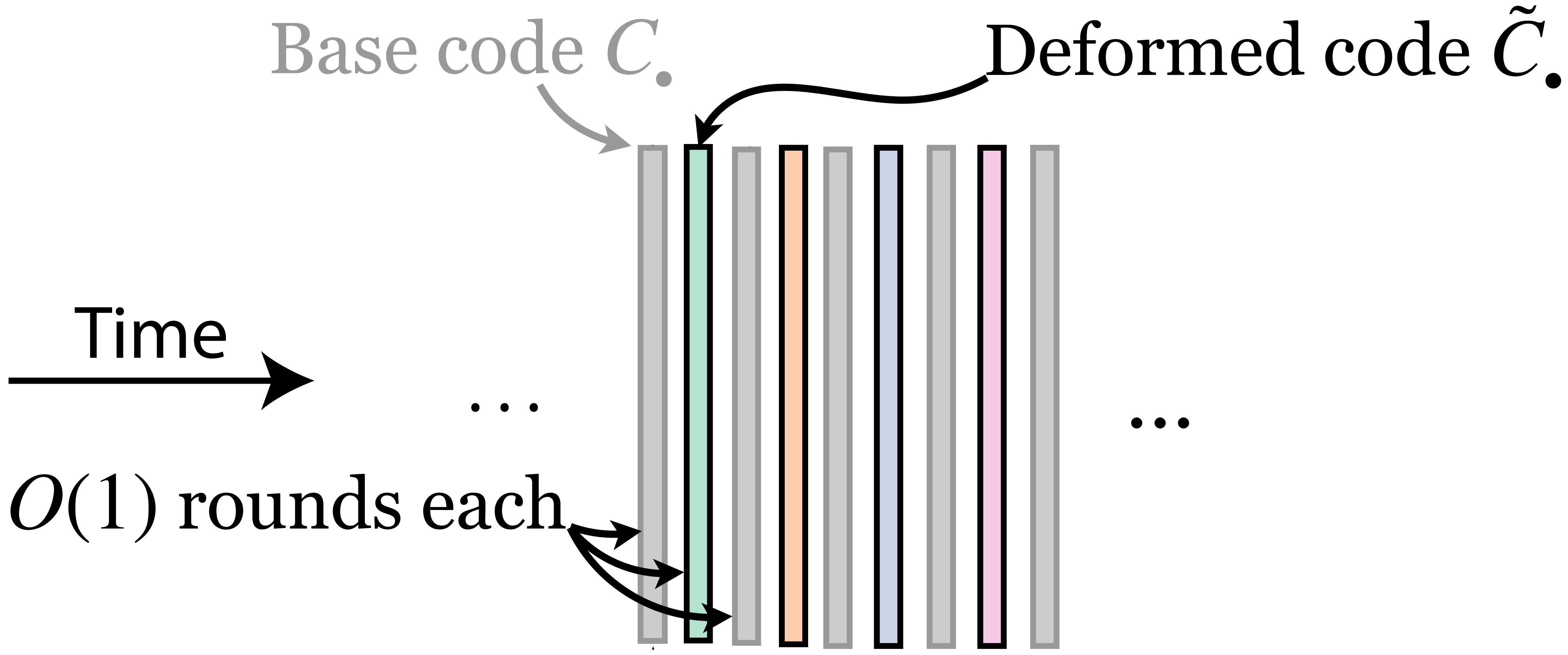


- Requires **single-shot preparation** of **deformed code**, and **base code $C_\bullet$**.
- Restricts what $C_\bullet$ can be.
  - Base code must have **meta-checks**
- Concurrent work[3]: deformed $\tilde{C}_\bullet$ may introduce large space overhead: $\tilde{O}(nkd)$



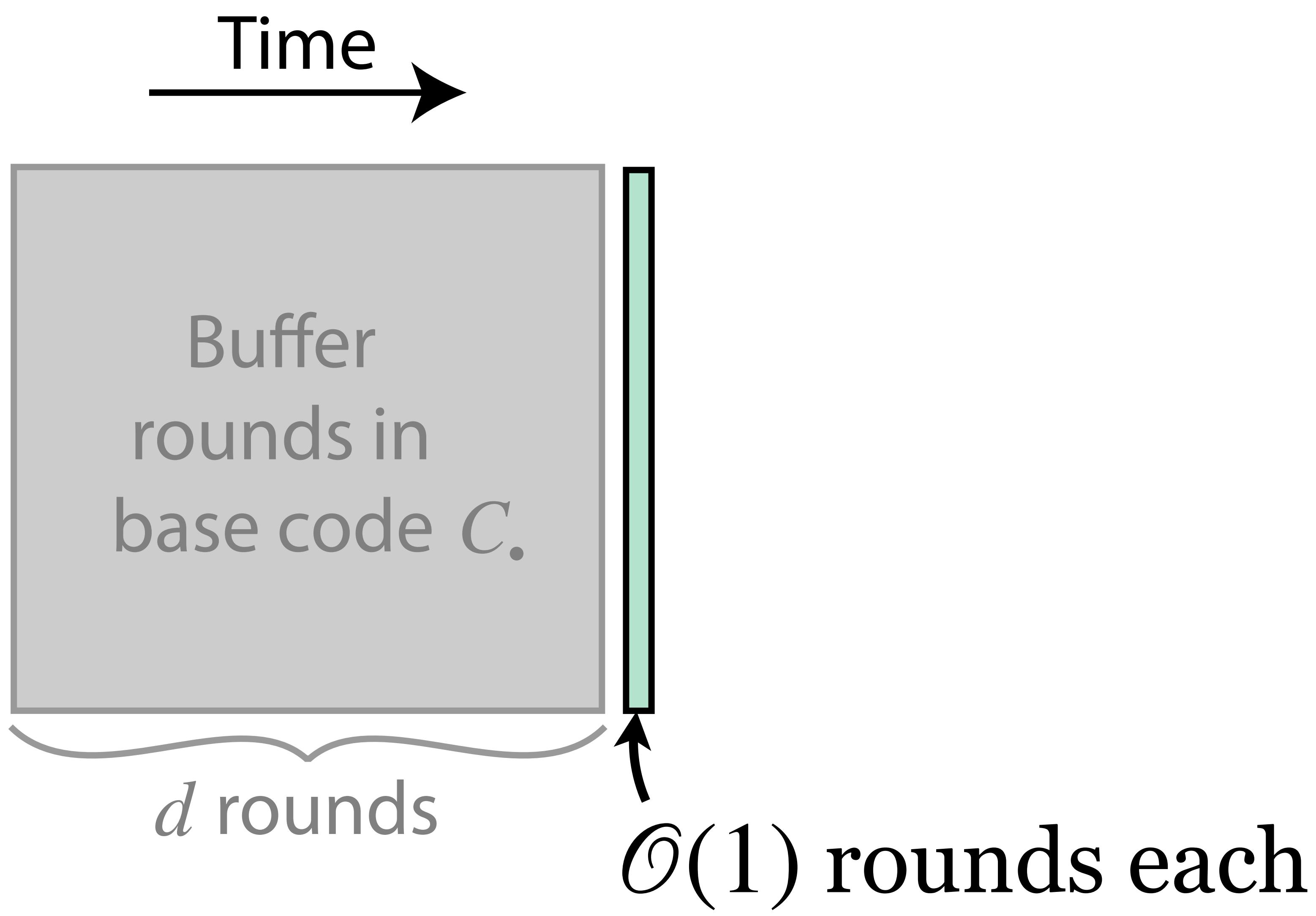
[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]
[4] Cowtan, *et al.* [2510.14895]
[5] **KC**, He, *et al.* [2603.02157]

Single-shot surgery [1-3]



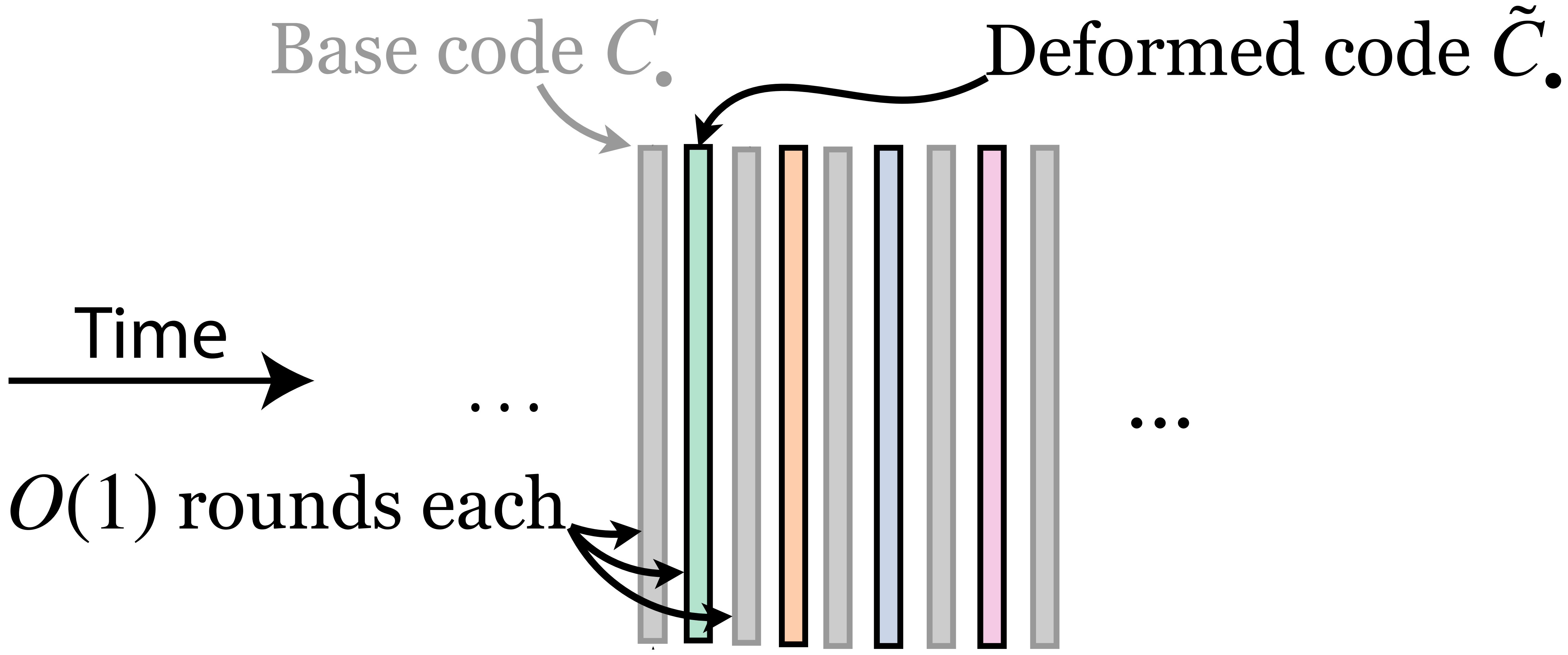
- Requires **single-shot preparation** of **deformed code**, and **base code** $C_\bullet$.
- Restricts what $C_\bullet$ can be.
  - Base code must have **meta-checks**
- Concurrent work[3]: deformed $\tilde{C}_\bullet$ may introduce large space overhead: $\tilde{O}(nkd)$

Constant-time surgery [4, 5]



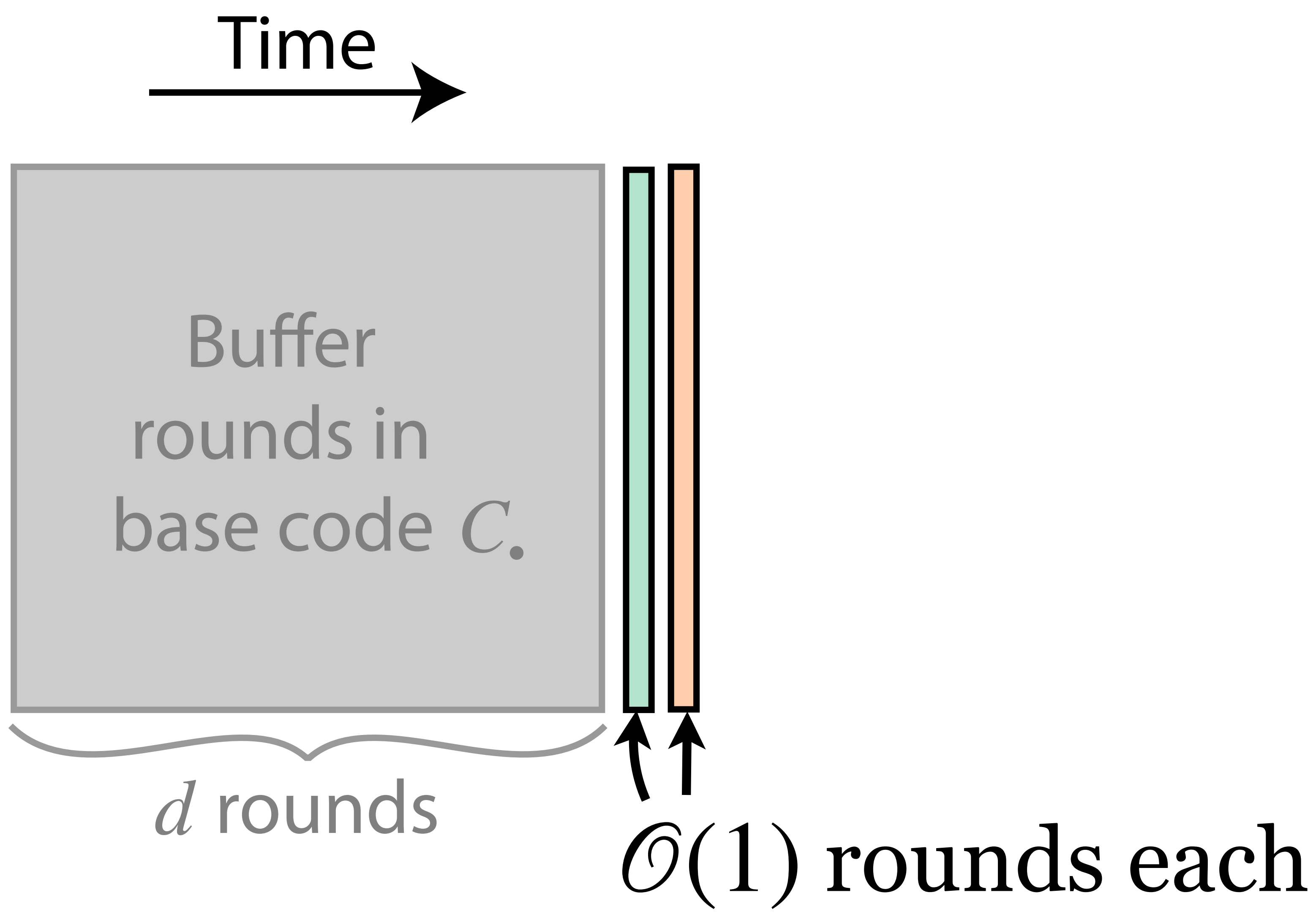
[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]
[4] Cowtan, *et al.* [2510.14895]
[5] **KC**, He, *et al.* [2603.02157]

Single-shot surgery [1-3]



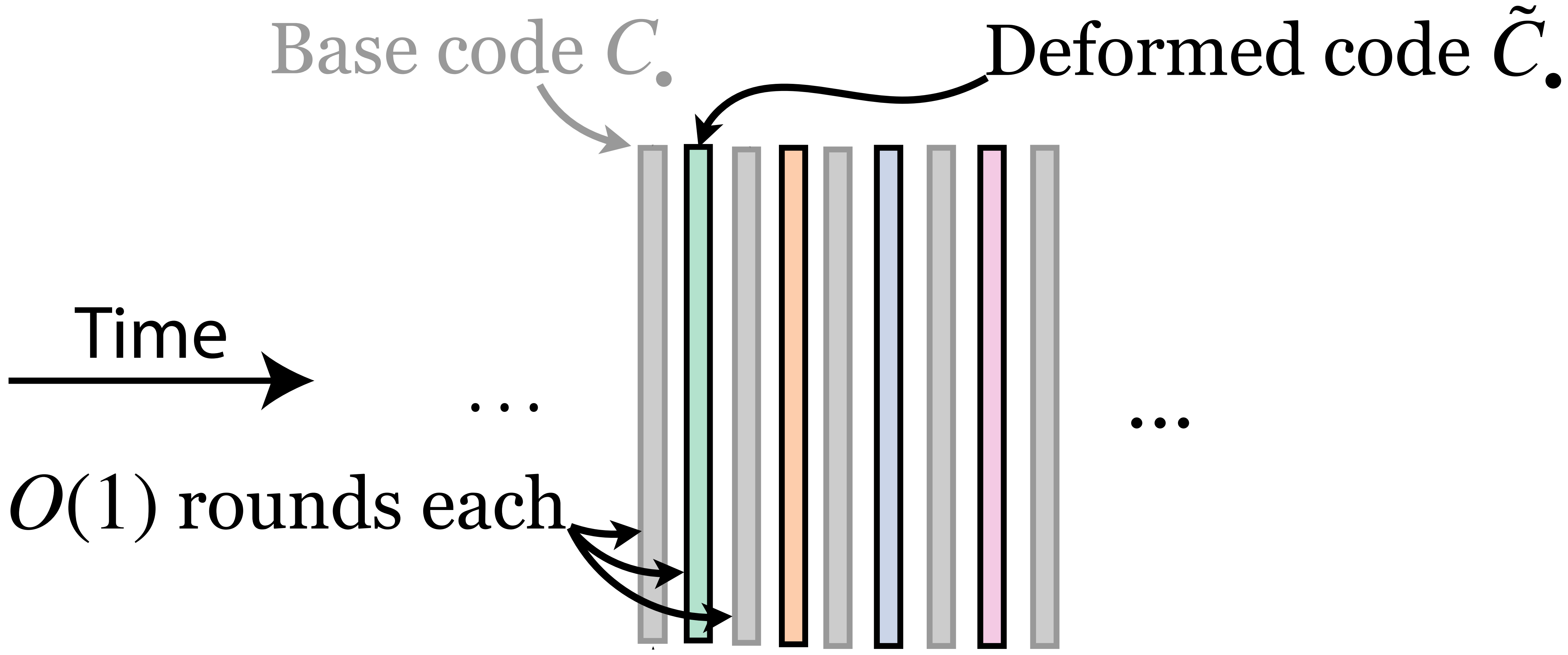
- Requires **single-shot preparation** of **deformed code**, and **base code** $C_\bullet$.
- Restricts what $C_\bullet$ can be.
    - Base code must have **meta-checks**
- Concurrent work[3]: deformed $\tilde{C}_\bullet$ may introduce large space overhead: $\tilde{O}(nkd)$

Constant-time surgery [4, 5]



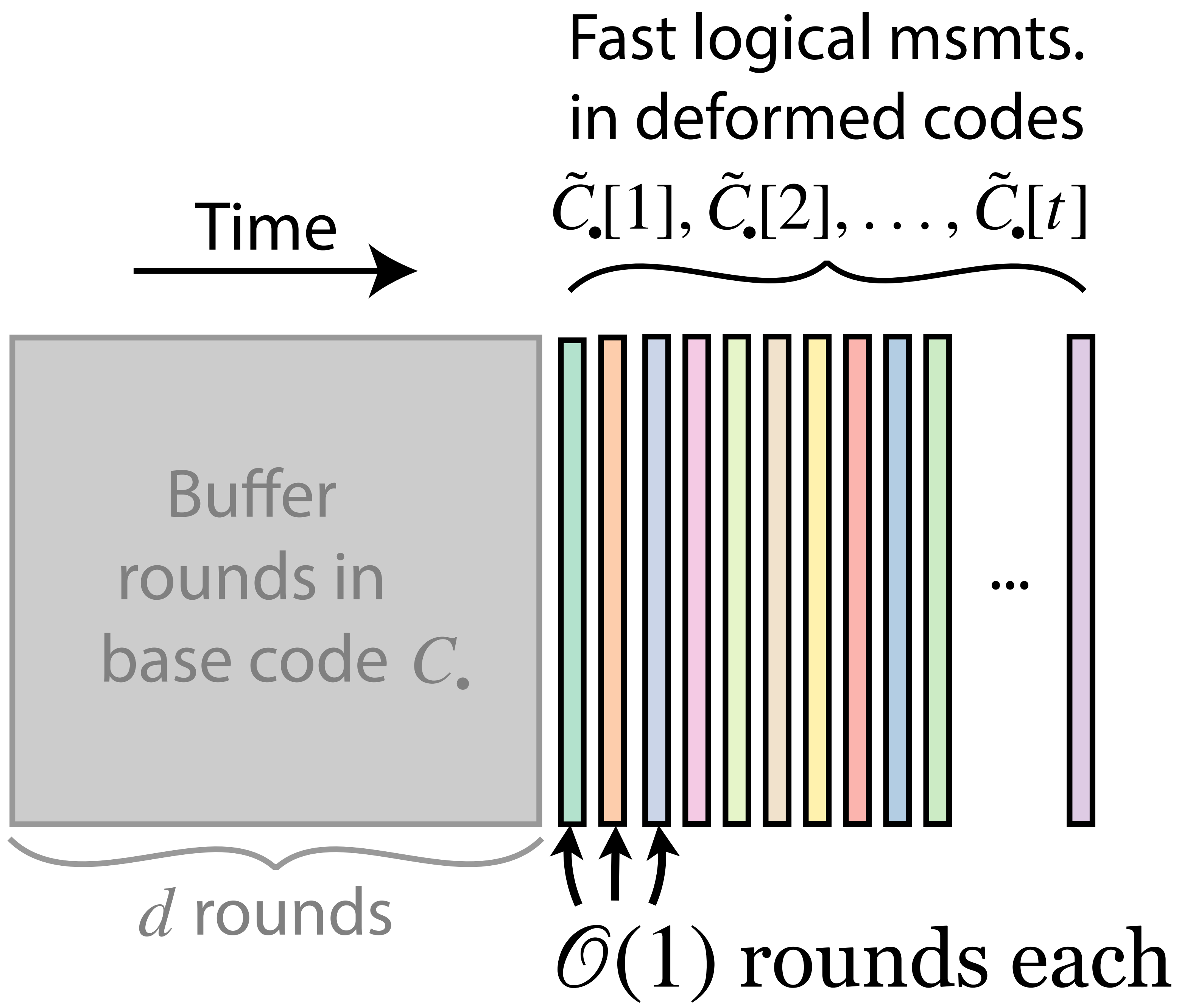
[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]
[4] Cowtan, *et al.* [2510.14895]
[5] **KC,** He, *et al.* [2603.02157]

Single-shot surgery [1-3]



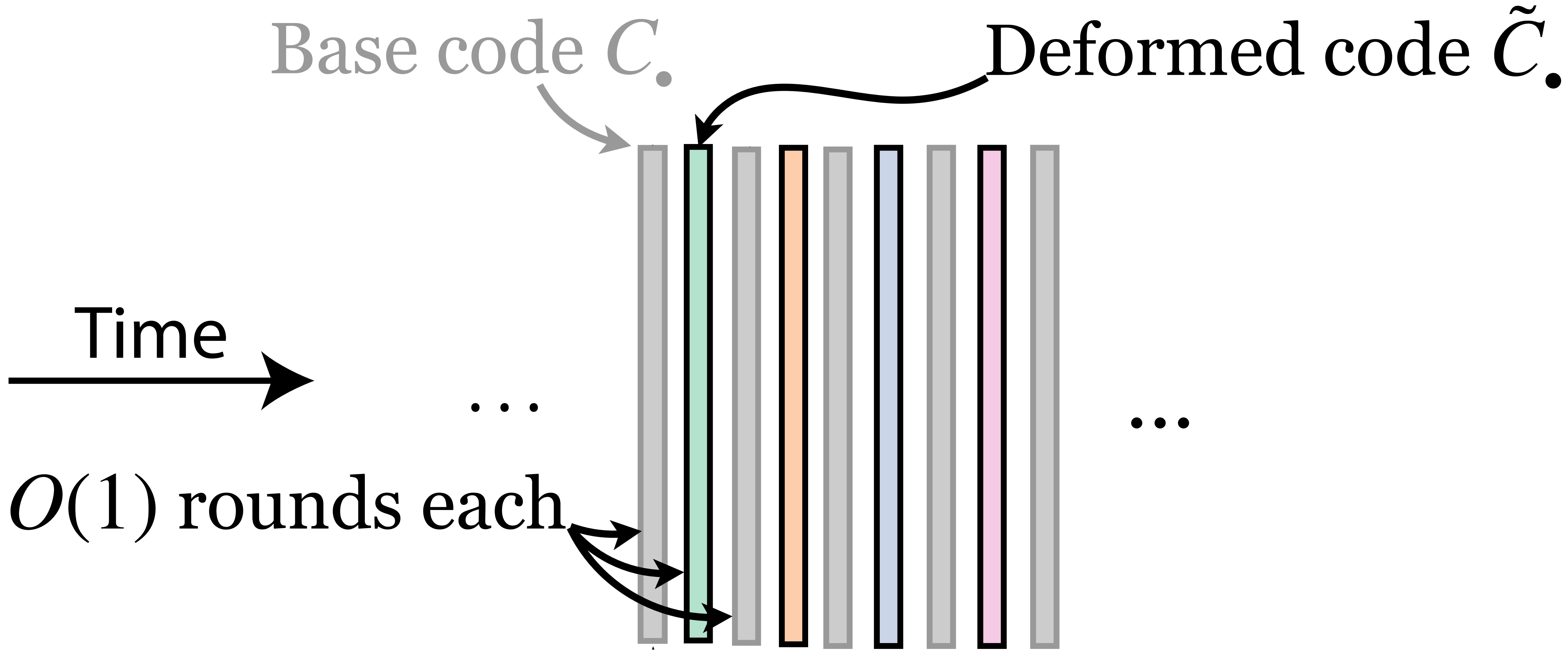
- Requires **single-shot preparation** of **deformed code**, and **base code** $C_\bullet$.
- Restricts what $C_\bullet$ can be.
  - Base code must have **meta-checks**
- Concurrent work[3]: deformed $\tilde{C}_\bullet$ may introduce large space overhead: $\tilde{O}(nkd)$

Constant-time surgery [4, 5]



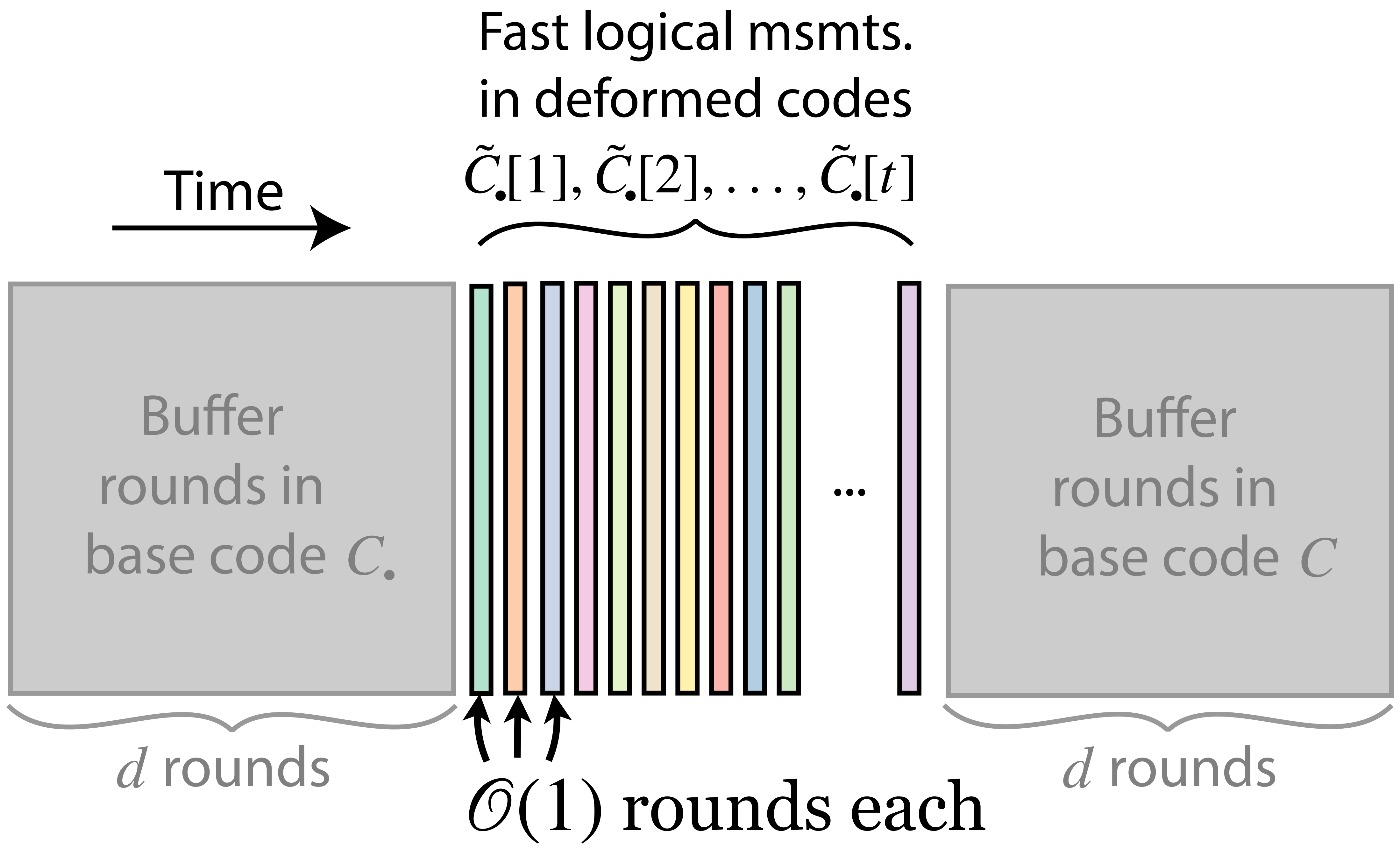
[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]
[4] Cowtan, *et al.* [2510.14895]
[5] **KC,** He, *et al.* [2603.02157]

Single-shot surgery [1-3]



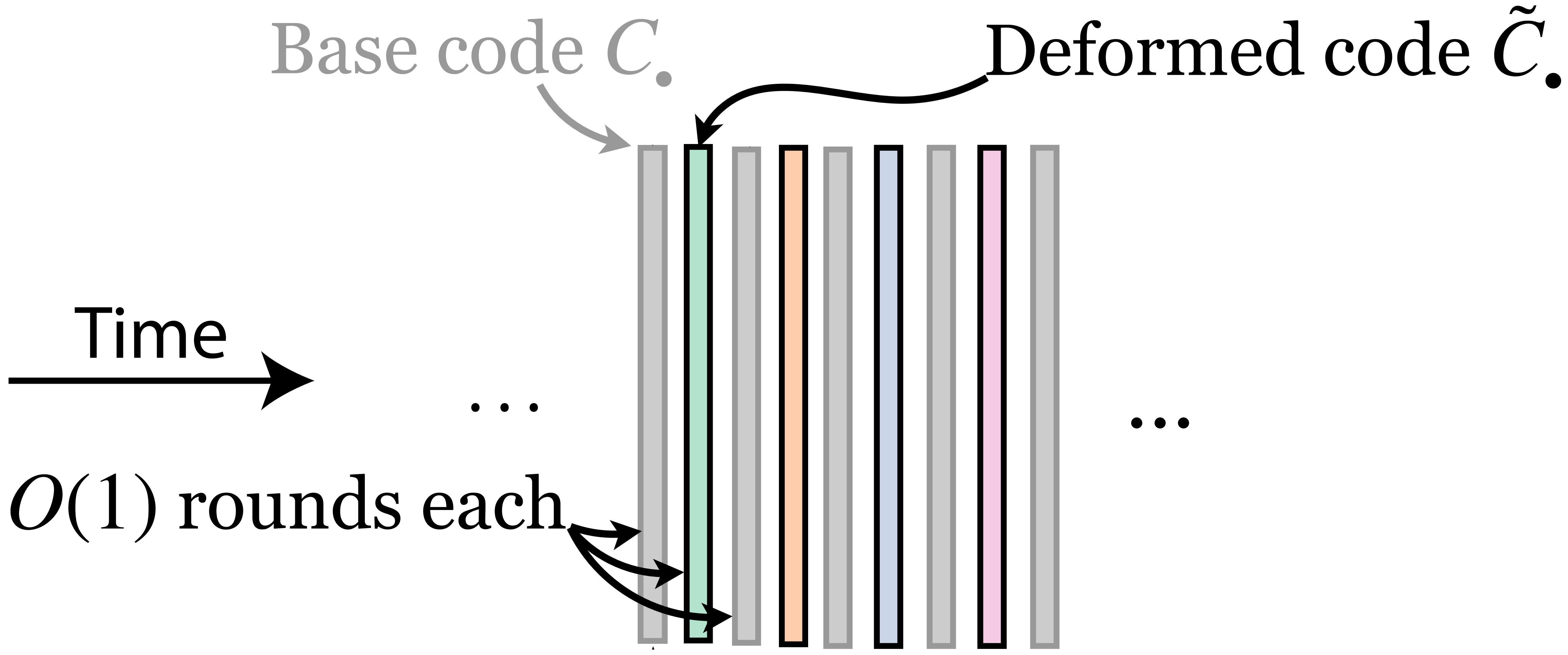
- Requires **single-shot preparation** of **deformed code**, and **base code** $C_\bullet$.
- Restricts what $C_\bullet$ can be.
  - Base code must have **meta-checks**
- Concurrent work[3]: deformed $\tilde{C}_\bullet$ may introduce large space overhead: $\tilde{O}(nkd)$

Constant-time surgery [4, 5]



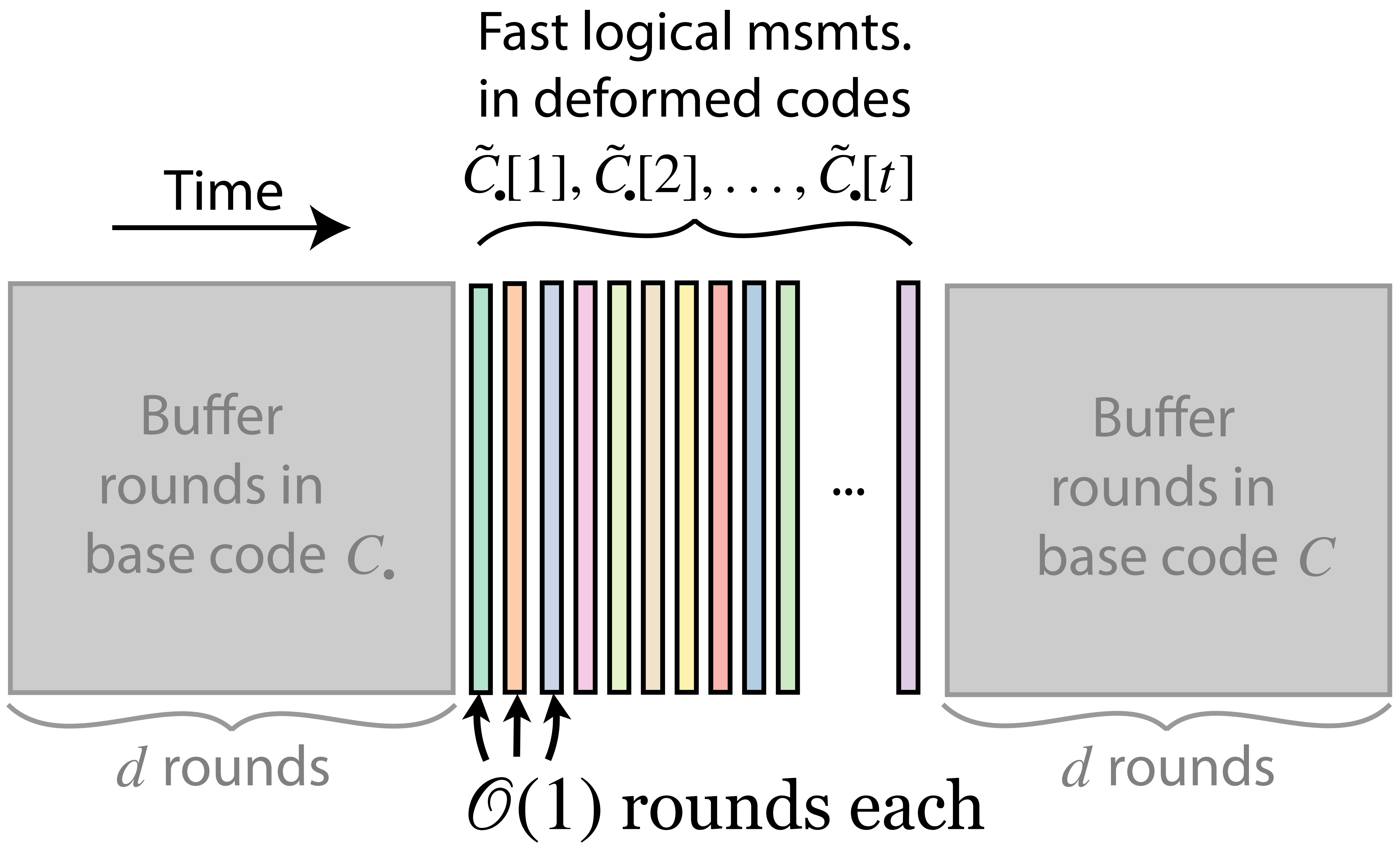
[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]
[4] Cowtan, *et al.* [2510.14895]
[5] **KC,** He, *et al.* [2603.02157]

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: $\tilde{O}(nkd)$

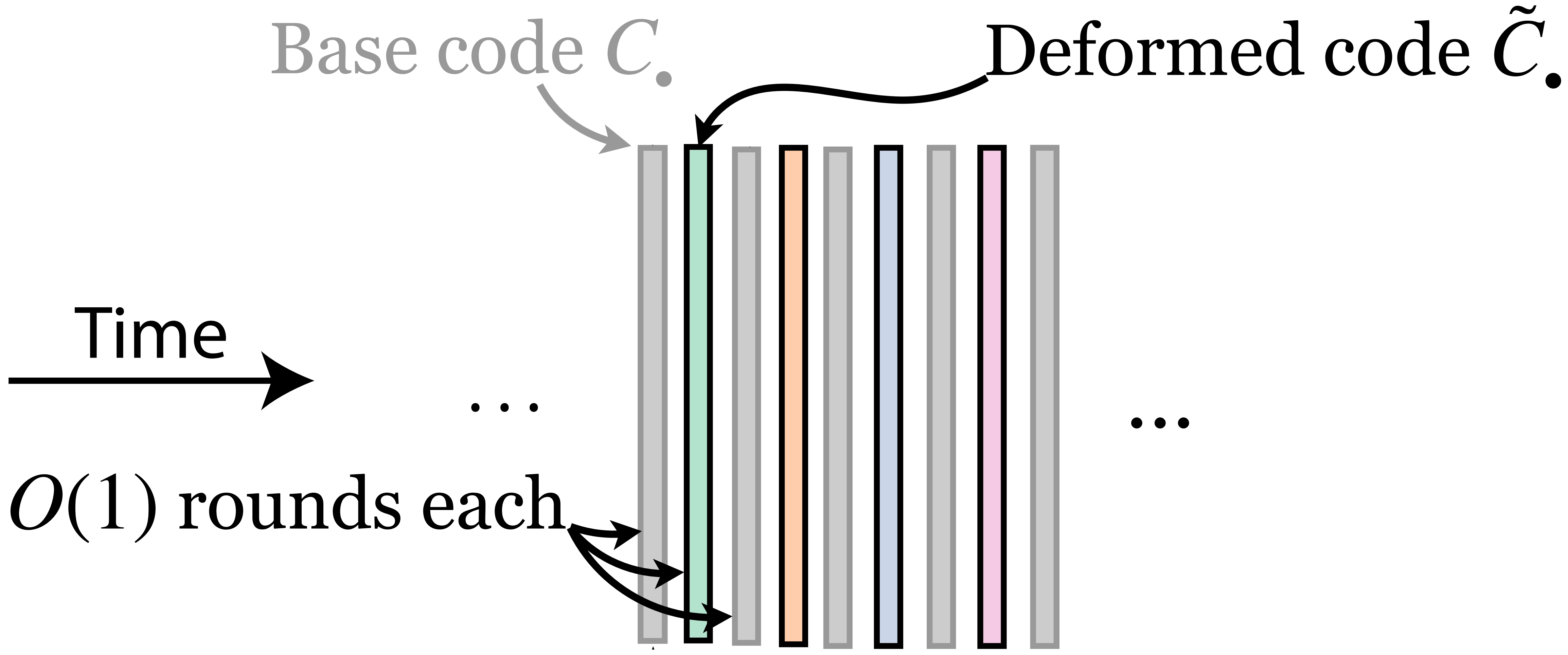
Constant-time surgery [4, 5]



- C need not be single-shot state preparable

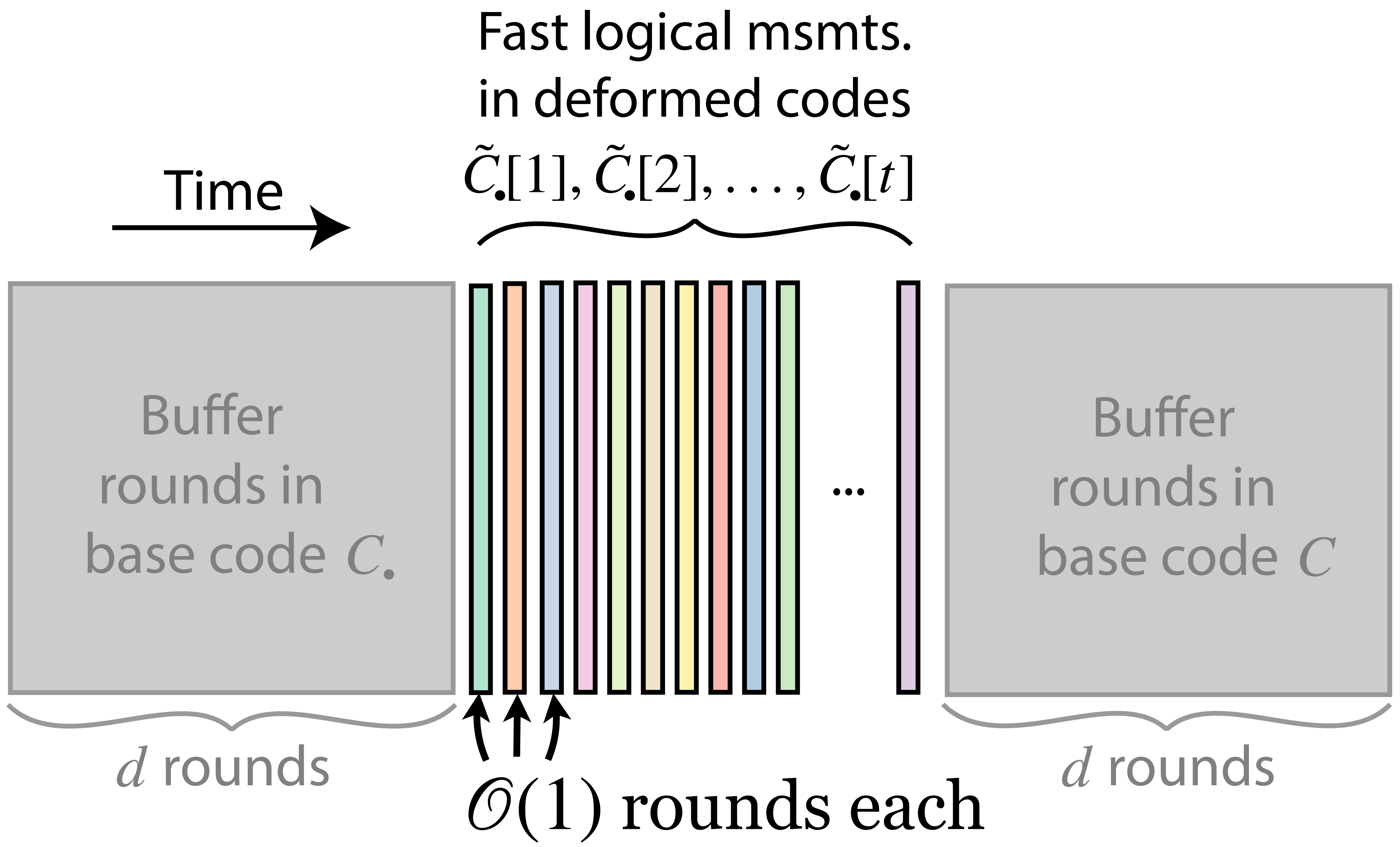
[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]
[4] Cowtan, *et al.* [2510.14895]
[5] **KC**, He, *et al.* [2603.02157]

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: $\tilde{O}(nkd)$

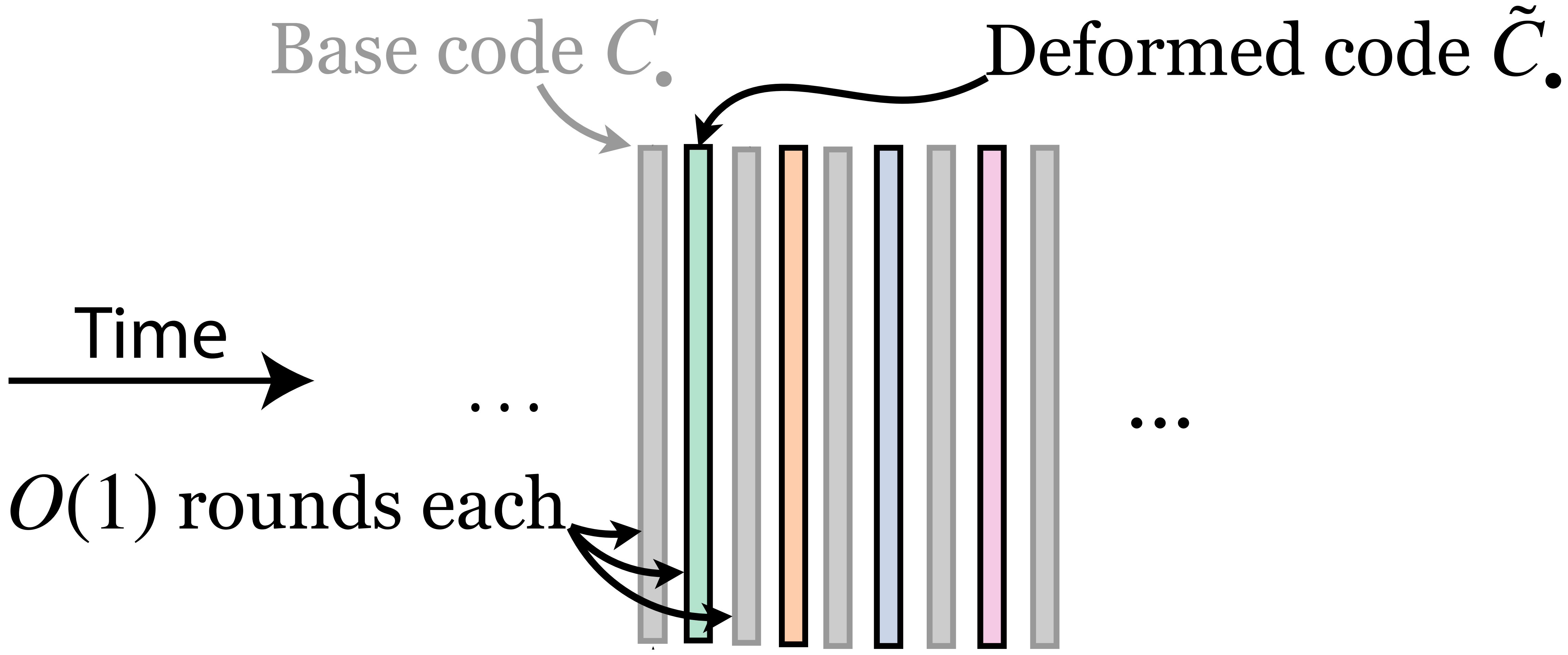
Constant-time surgery [4, 5]



- C need not be single-shot state preparable
- Sufficient detectors to infer the Pauli frame after each msmt

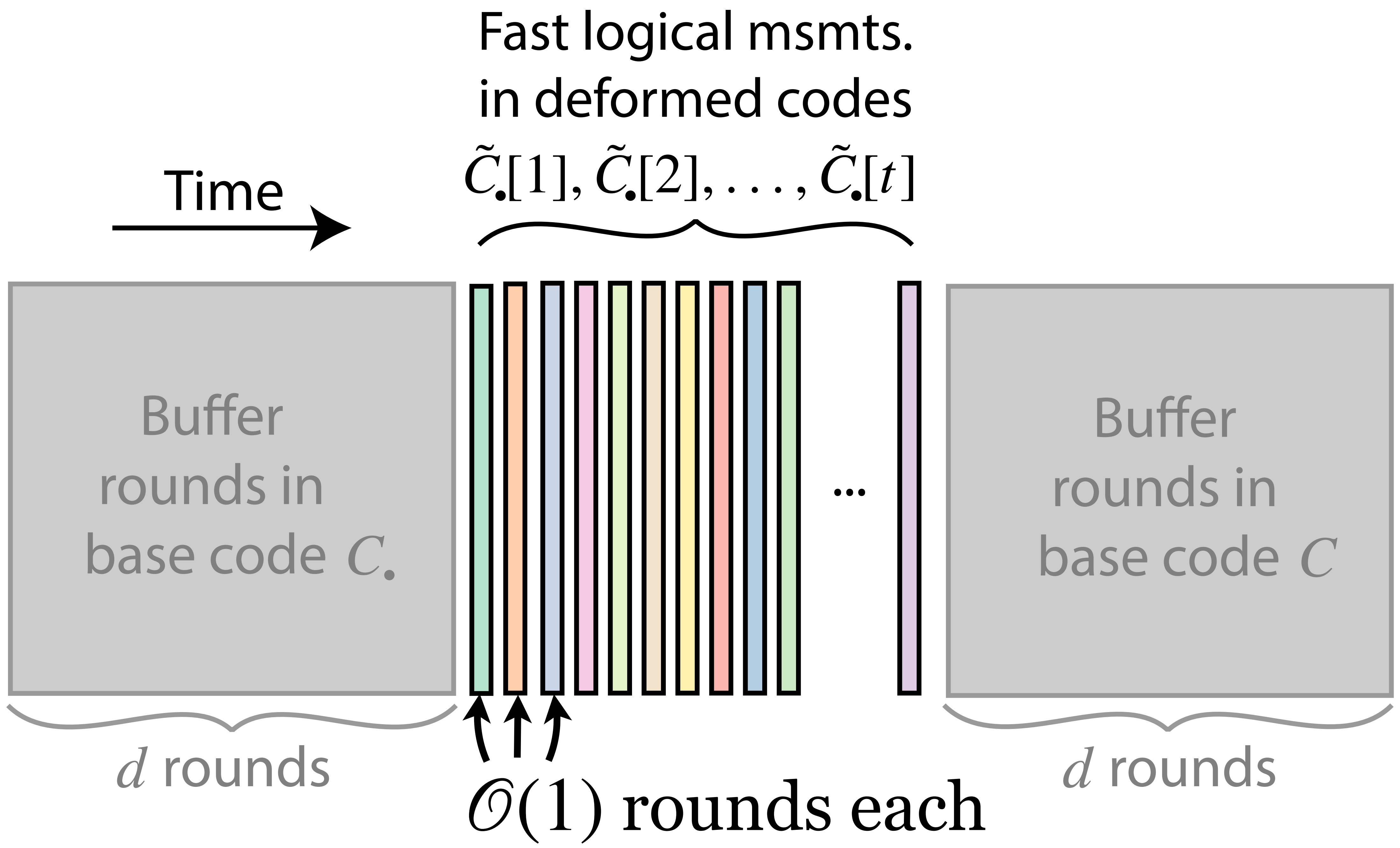
[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]
[4] Cowtan, *et al.* [2510.14895]
[5] **KC**, He, *et al.* [2603.02157]

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]

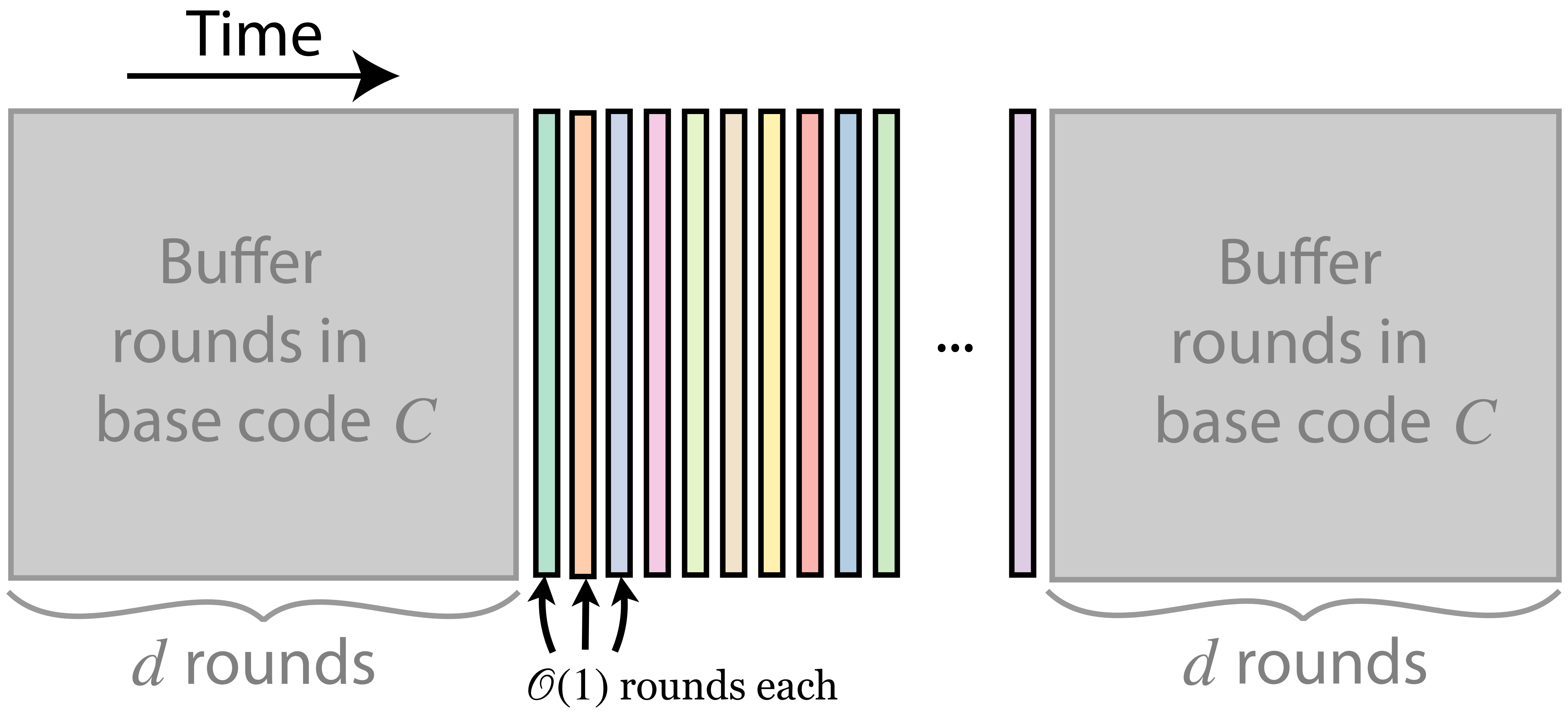


- C need not be single-shot state preparable
- Sufficient detectors to infer the Pauli frame after each msmt
- Each $\tilde{C}[i]$ needs to satisfy certain properties to ensure fault-tolerance

[1] Hillmann *et al.* [2410.12963]
[2] Tan, Hong, Lin *et al.* [2510.08552]
[3] Baspin *et al.* [2510.04521]
[4] Cowtan, *et al.* [2510.14895]
[5] **KC**, He, *et al.* [2603.02157]

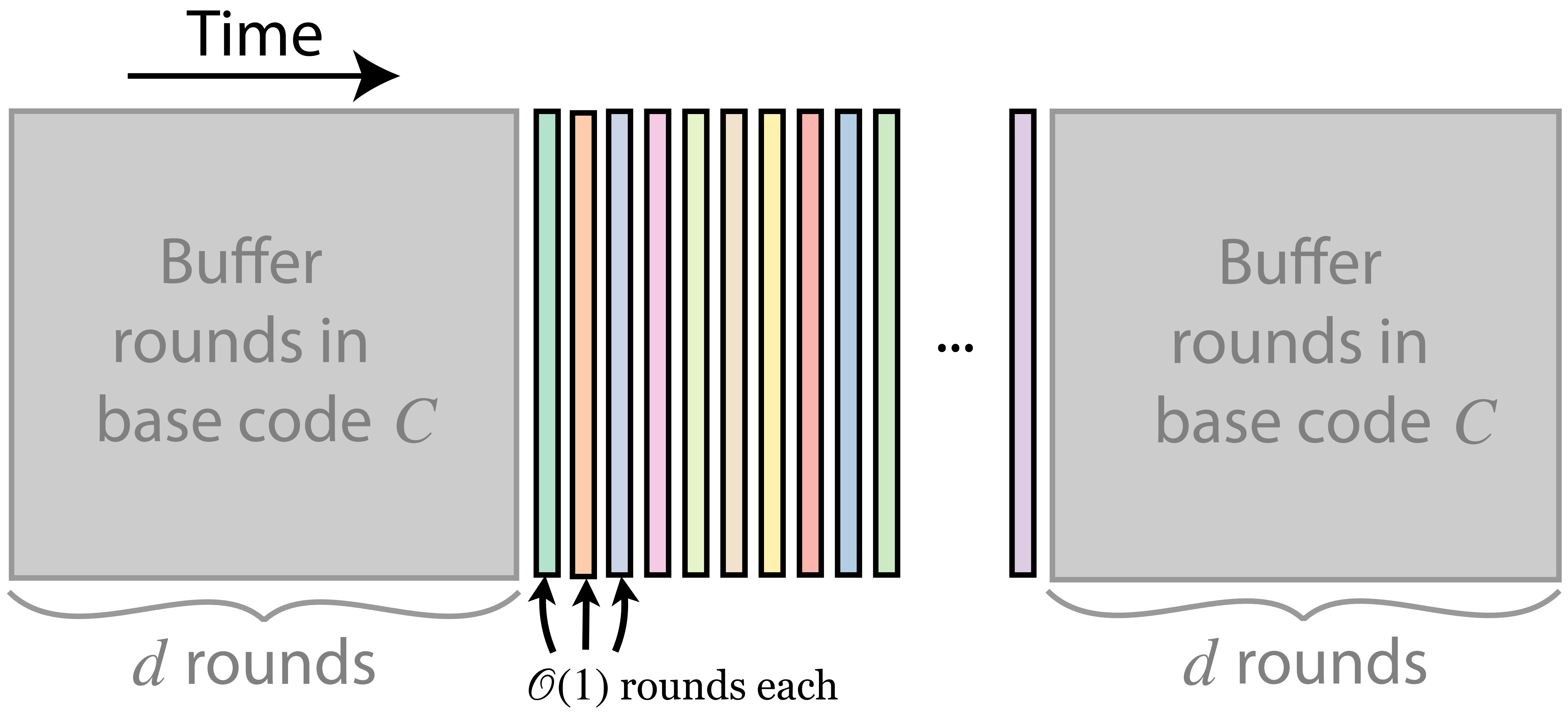
Fault-tolerance conditions for constant time surgery: Cowtan, *et al*. [2510.14895]

Fault-tolerance conditions for constant time surgery: Cowtan, *et al.* [2510.14895]



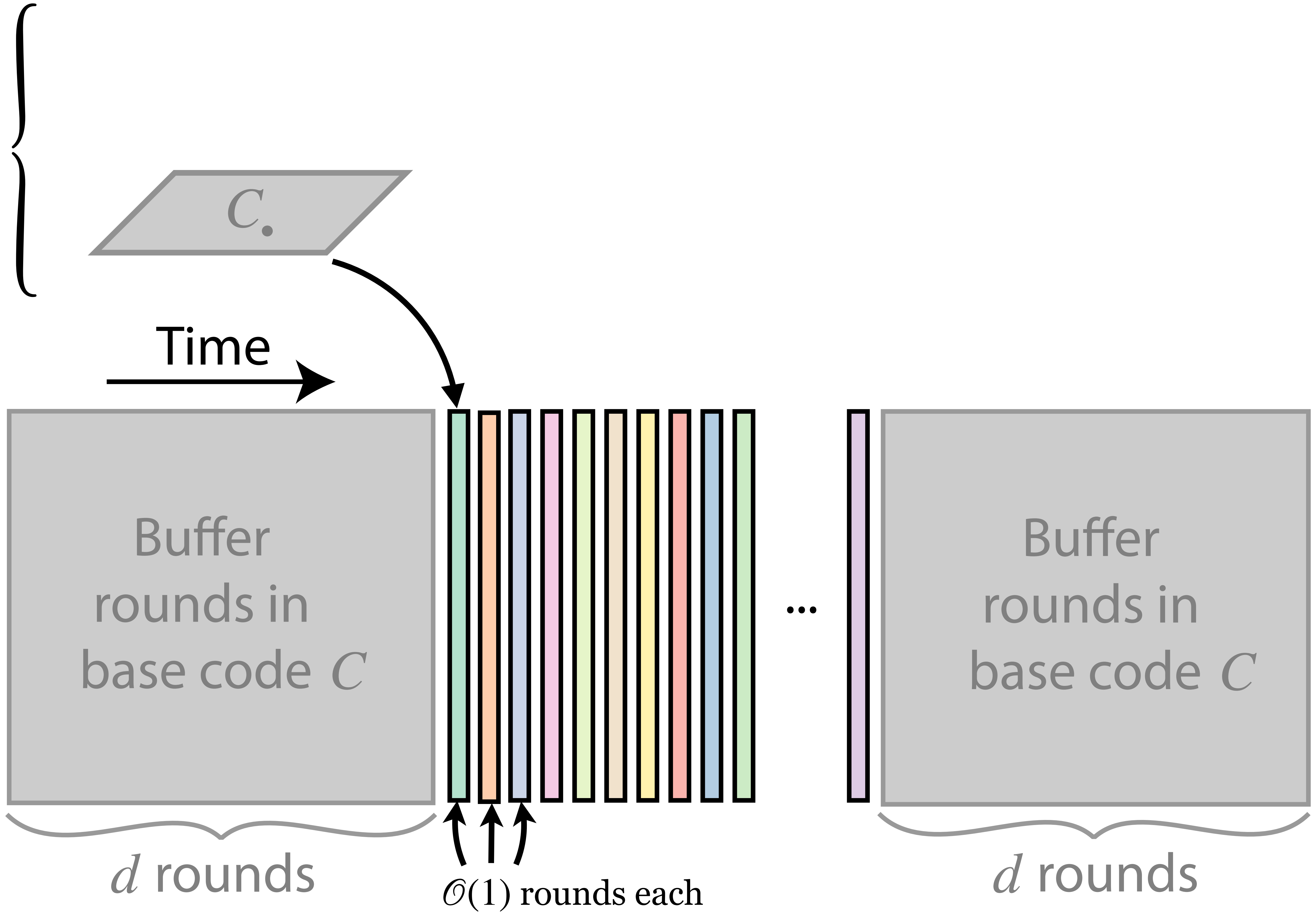
Fault-tolerance conditions for constant time surgery: Cowtan, *et al.* [2510.14895]

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 $f[1] : A[1]_\bullet \to C_\bullet, \ldots, f[t] : A[t]_\bullet \to C_\bullet$.



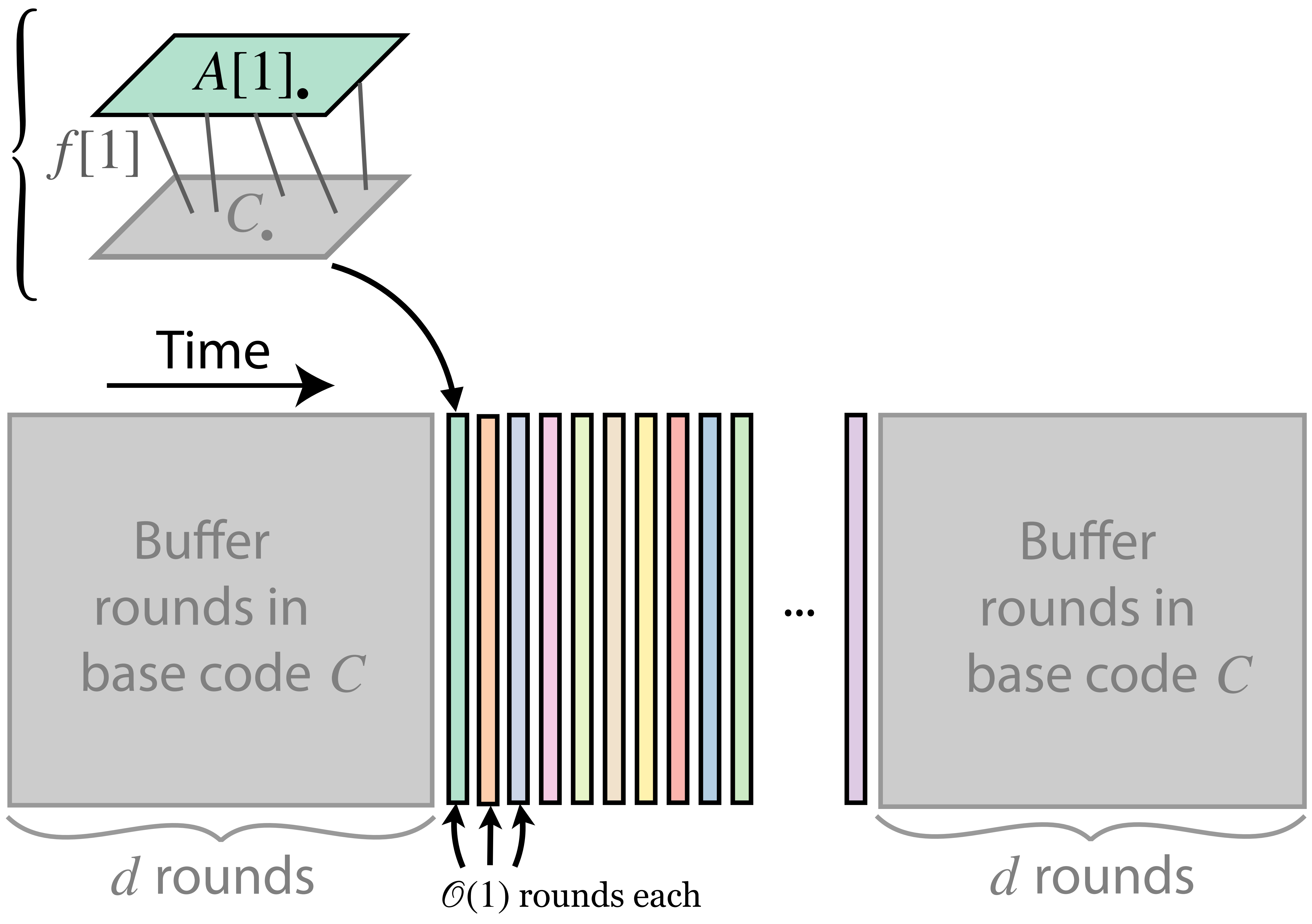
Fault-tolerance conditions for constant time surgery: Cowtan, *et al.* [2510.14895]

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



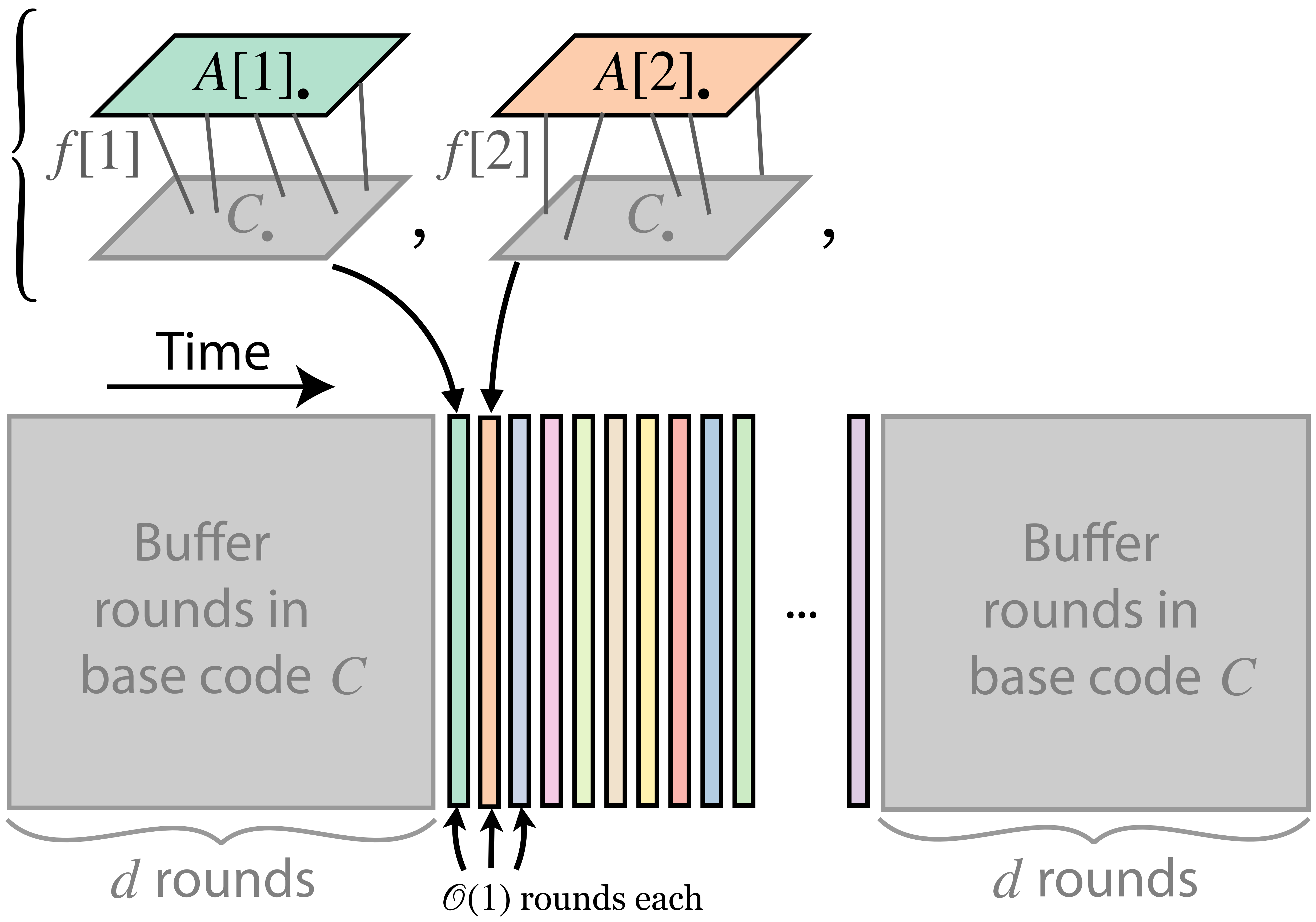
Fault-tolerance conditions for constant time surgery: Cowtan, *et al.* [2510.14895]

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 $f[1] : A[1]_\bullet \to C_\bullet, \ldots, f[t] : A[t]_\bullet \to C_\bullet$.



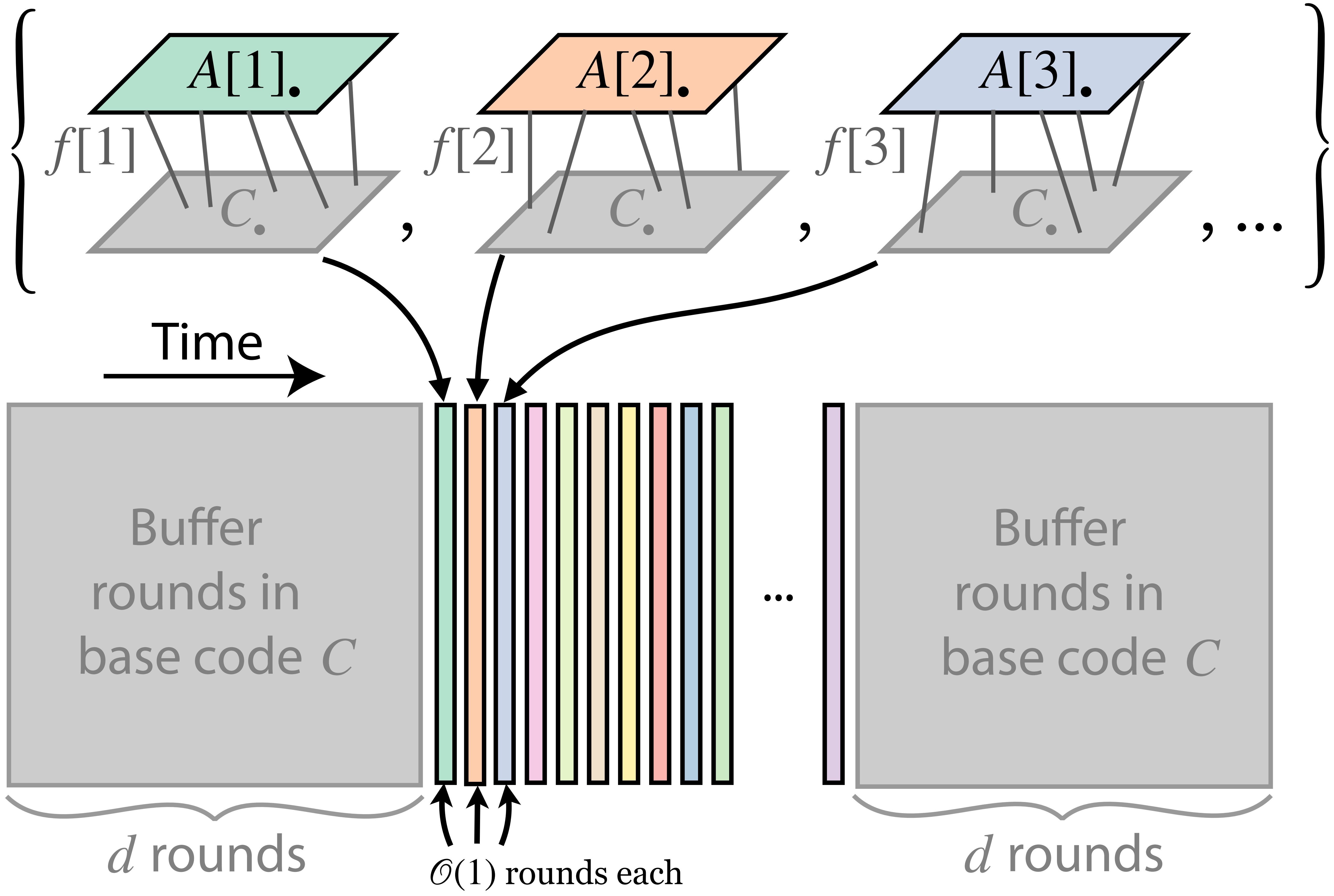
Fault-tolerance conditions for constant time surgery: Cowtan, *et al.* [2510.14895]

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



Fault-tolerance conditions for constant time surgery: Cowtan, *et al.* [2510.14895]

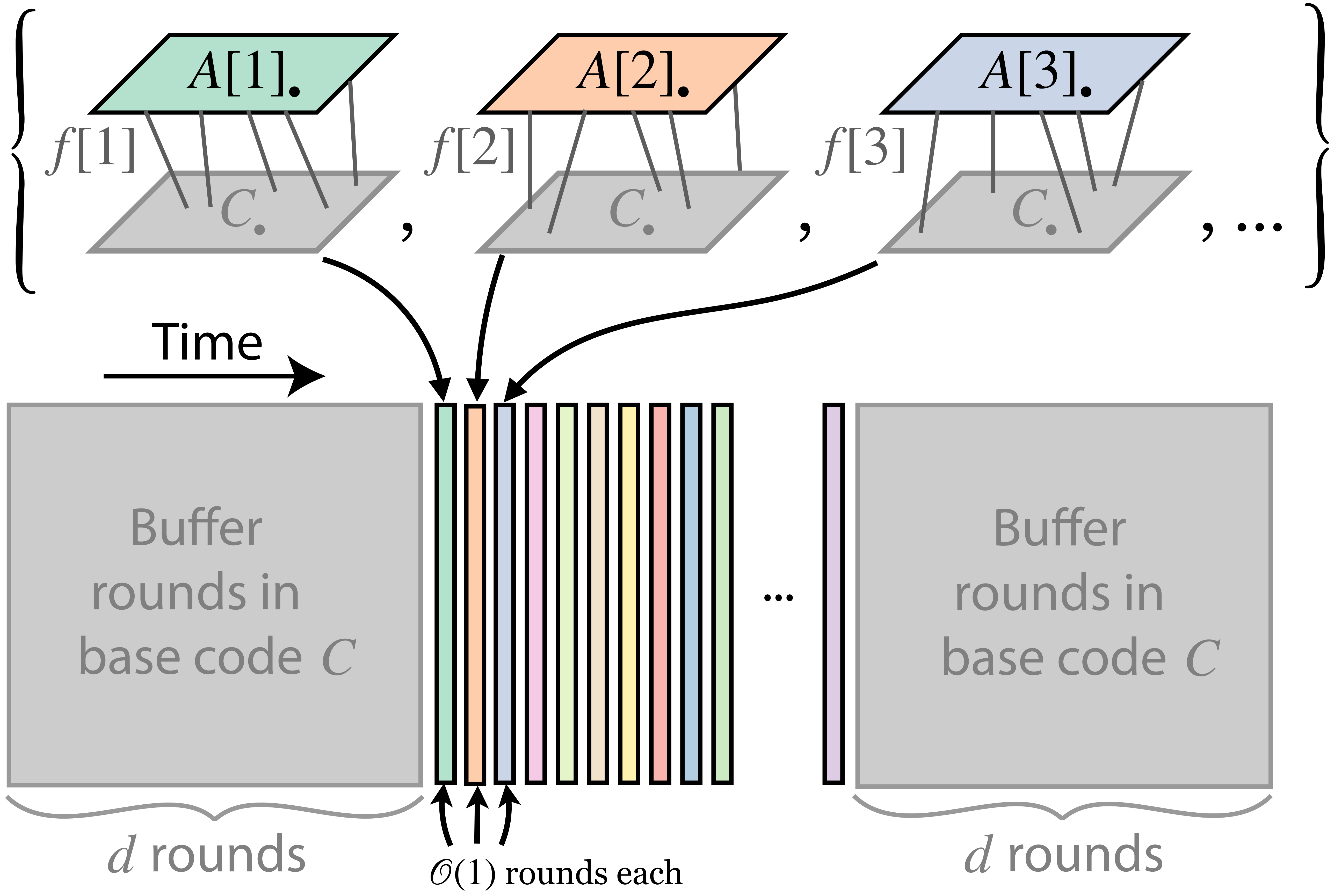
Let $C_\bullet$ be a CSS code of distance d, and consider a **sequence of surgery operations** defined by ancilla complexes $A[1]_\bullet, \dots, A[t]_\bullet$ and chain maps $f[1] : A[1]_\bullet \to C_\bullet, \dots, f[t] : A[t]_\bullet \to C_\bullet$.



Fault-tolerance conditions for constant time surgery: Cowtan, *et al.* [2510.14895]

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:**

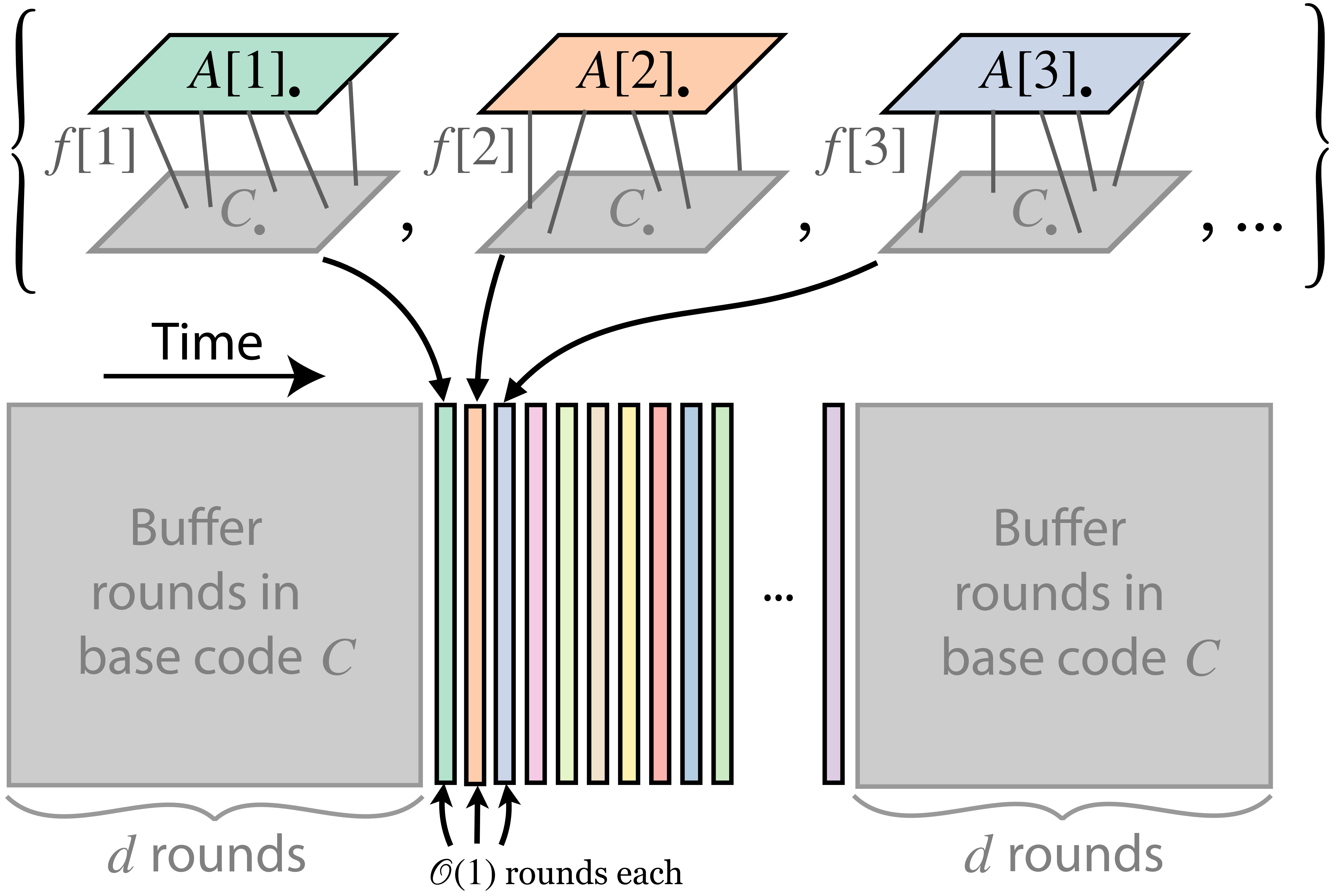


Fault-tolerance conditions for constant time surgery: Cowtan, *et al.* [2510.14895]

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.



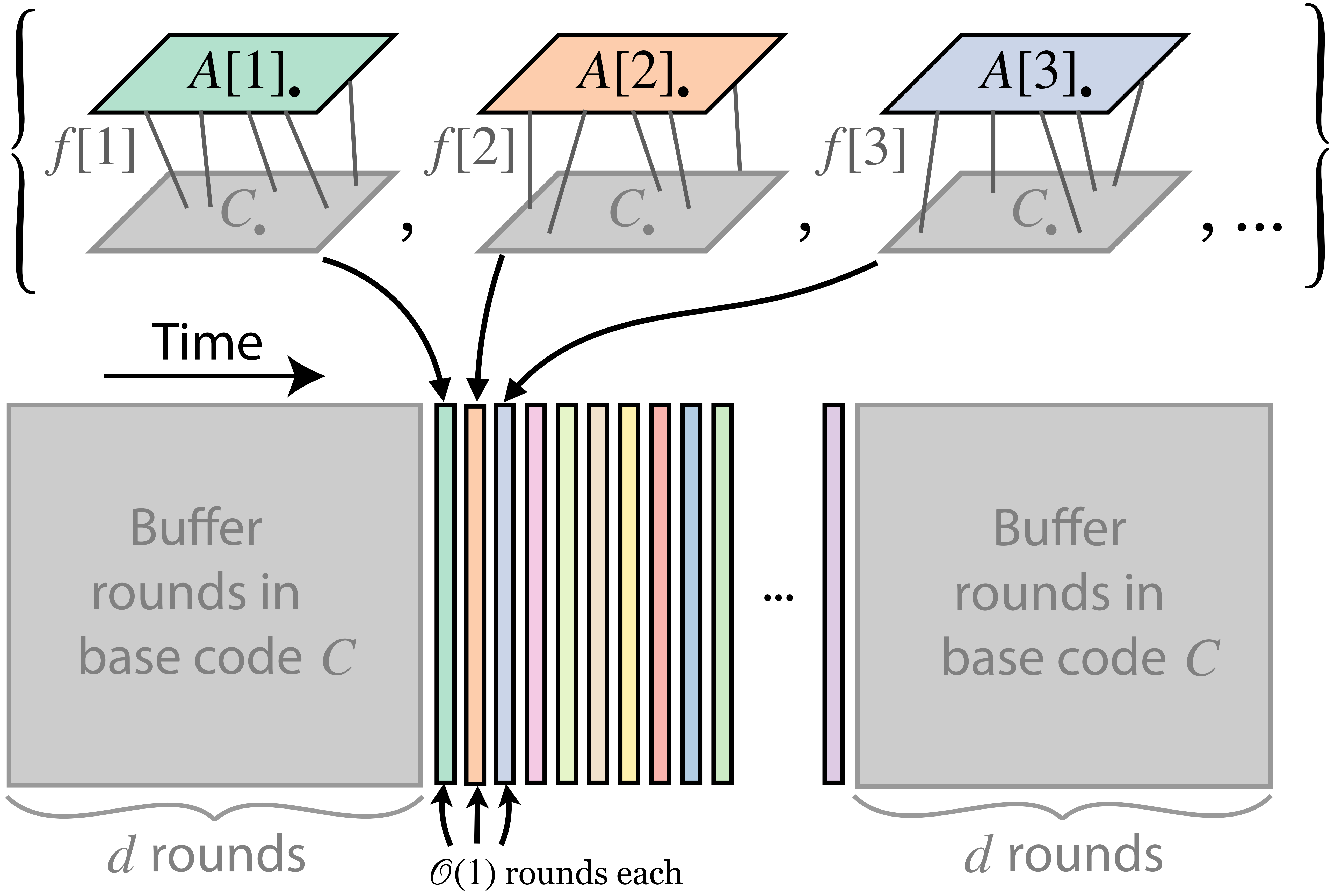
Fault-tolerance conditions for constant time surgery: Cowtan, *et al.* [2510.14895]

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



Fault-tolerance conditions for constant time surgery: Cowtan, *et al.* [2510.14895]

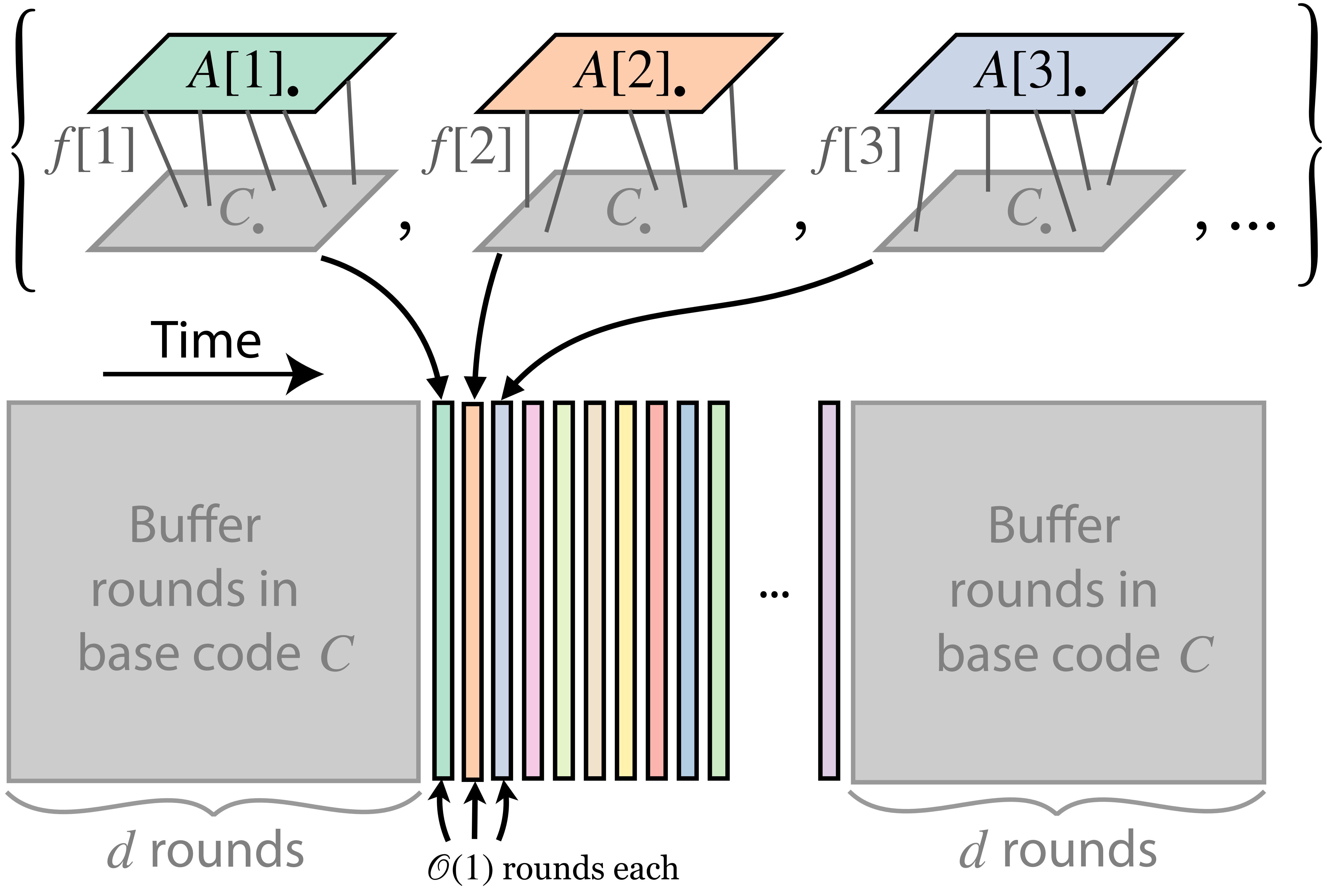
Let $C_\bullet$ be a CSS code of distance d, and consider a **sequence of surgery operations** defined by ancilla complexes $A[1]_\bullet, \ldots, A[t]_\bullet$ and chain maps $f[1] : A[1]_\bullet \to C_\bullet, \ldots, f[t] : A[t]_\bullet \to C_\bullet$.

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 $\mathrm{CC}_\bullet$* has distance $\geq d$.



a CSS code defined by t surgery operations

Fault-tolerance conditions for constant time surgery: Cowtan, *et al.* [2510.14895]

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 $f[1] : A[1]_\bullet \to C_\bullet, \ldots, f[t] : A[t]_\bullet \to C_\bullet$.

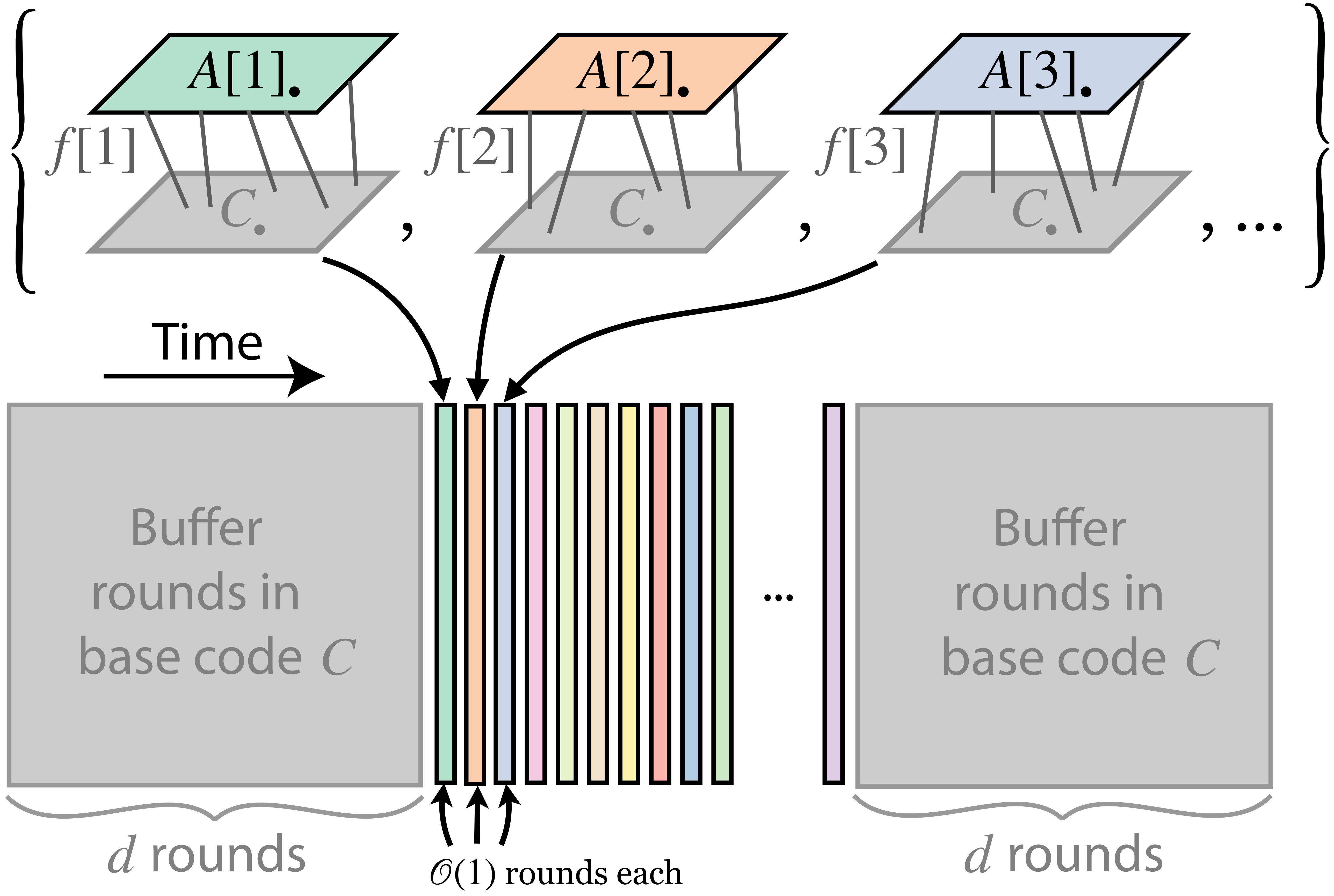
Then the constant-time surgery protocol has **fault distance** d **if:**

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

Can protect up to d ancilla measurement errors

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

Unmeasured logical qubits have high distance



**a CSS code defined by t surgery operations*

Fault-tolerance conditions for constant time surgery: Cowtan, *et al.* [2510.14895]

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

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

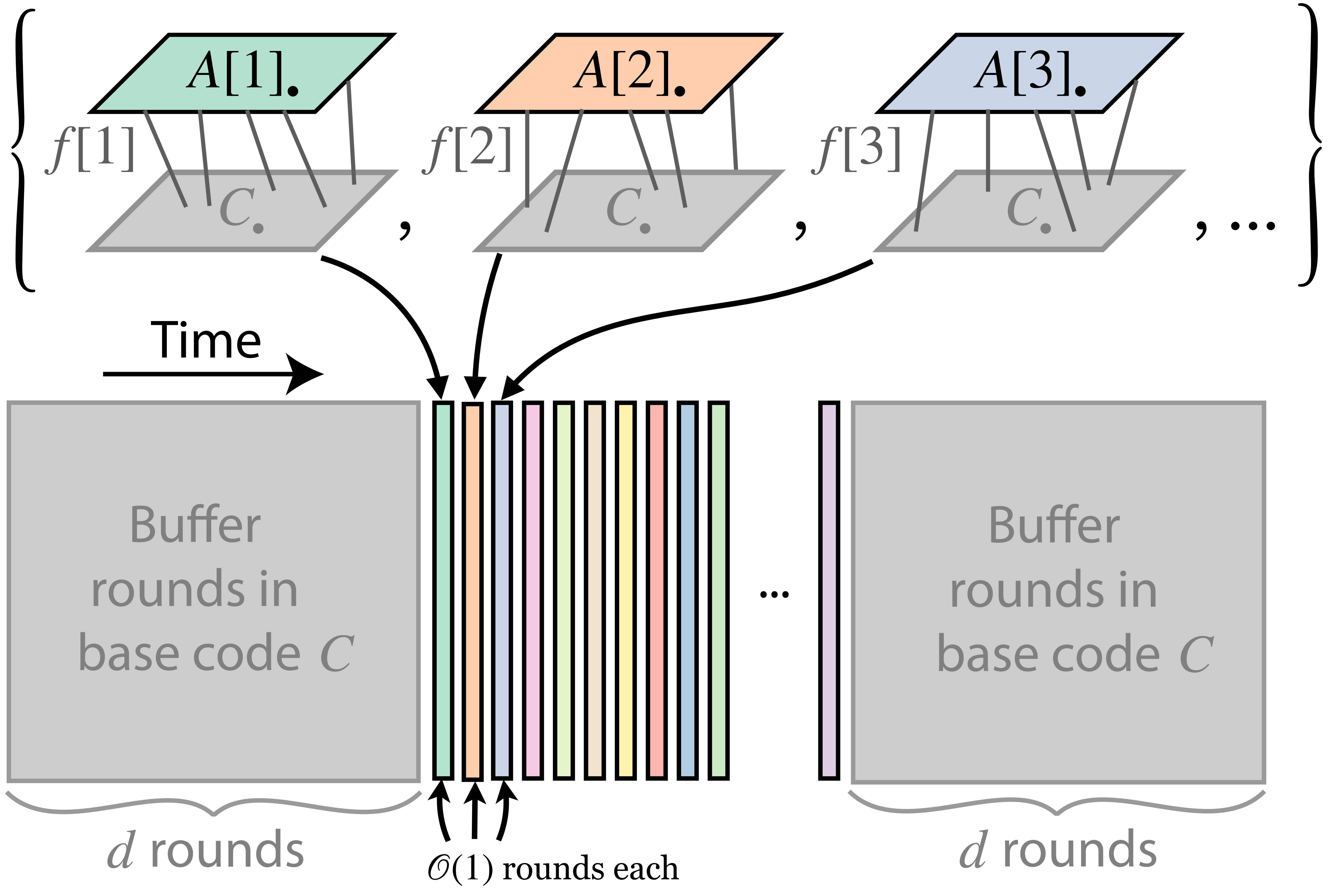
- Each ancilla complex $A[i]_\bullet$ has meta-check distance at least d.

Can protect up to d ancilla measurement errors

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

Unmeasured logical qubits have high distance

- Chain maps are partially injective.



*a CSS code defined by t surgery operations

Fault-tolerance conditions for constant time surgery: Cowtan, *et al.* [2510.14895]

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.

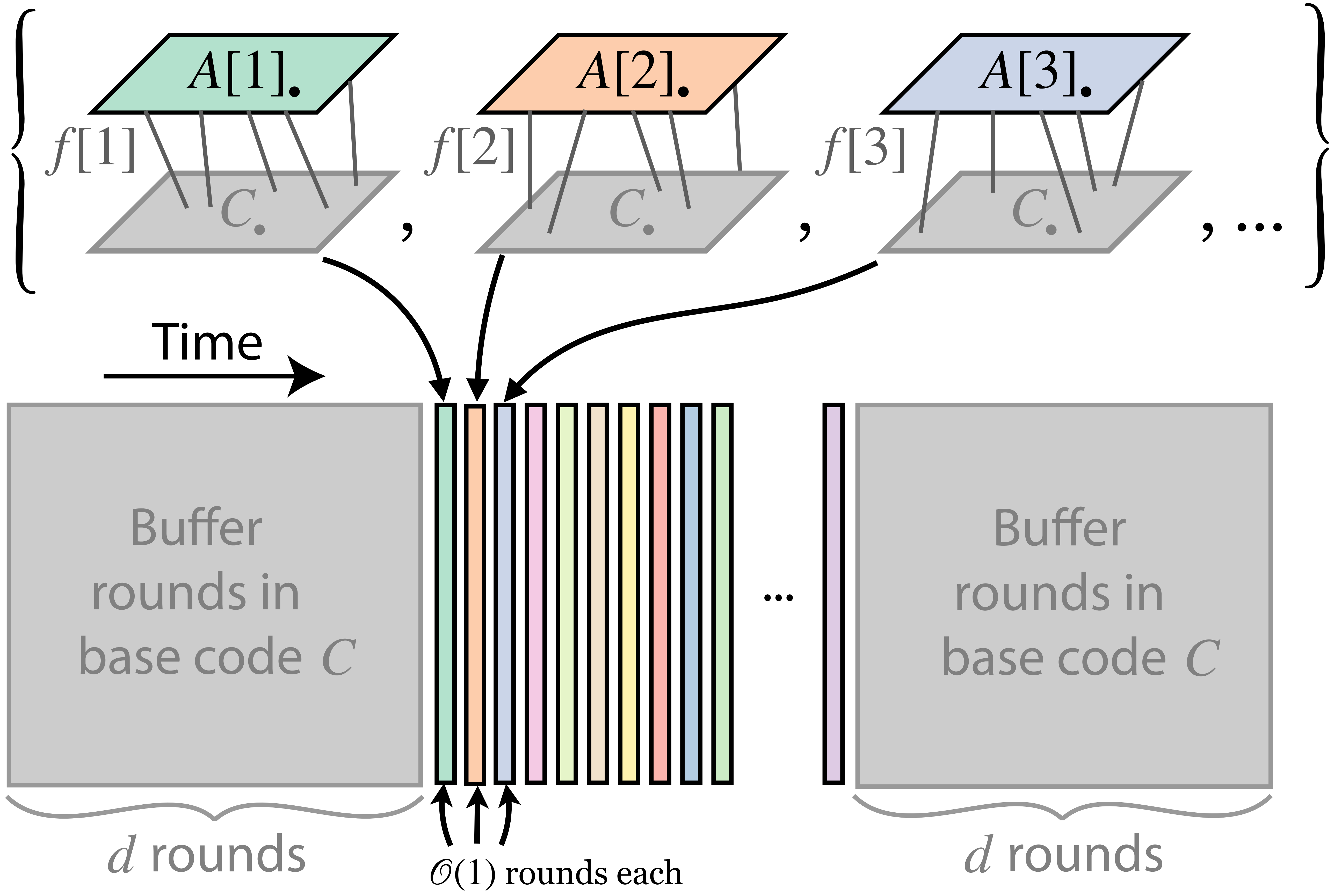
Can protect up to d ancilla measurement errors

- The compacted code $\mathrm{CC}_\bullet$* 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

Fault-tolerance conditions for constant time surgery: Cowtan, *et al.* [2510.14895]

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

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

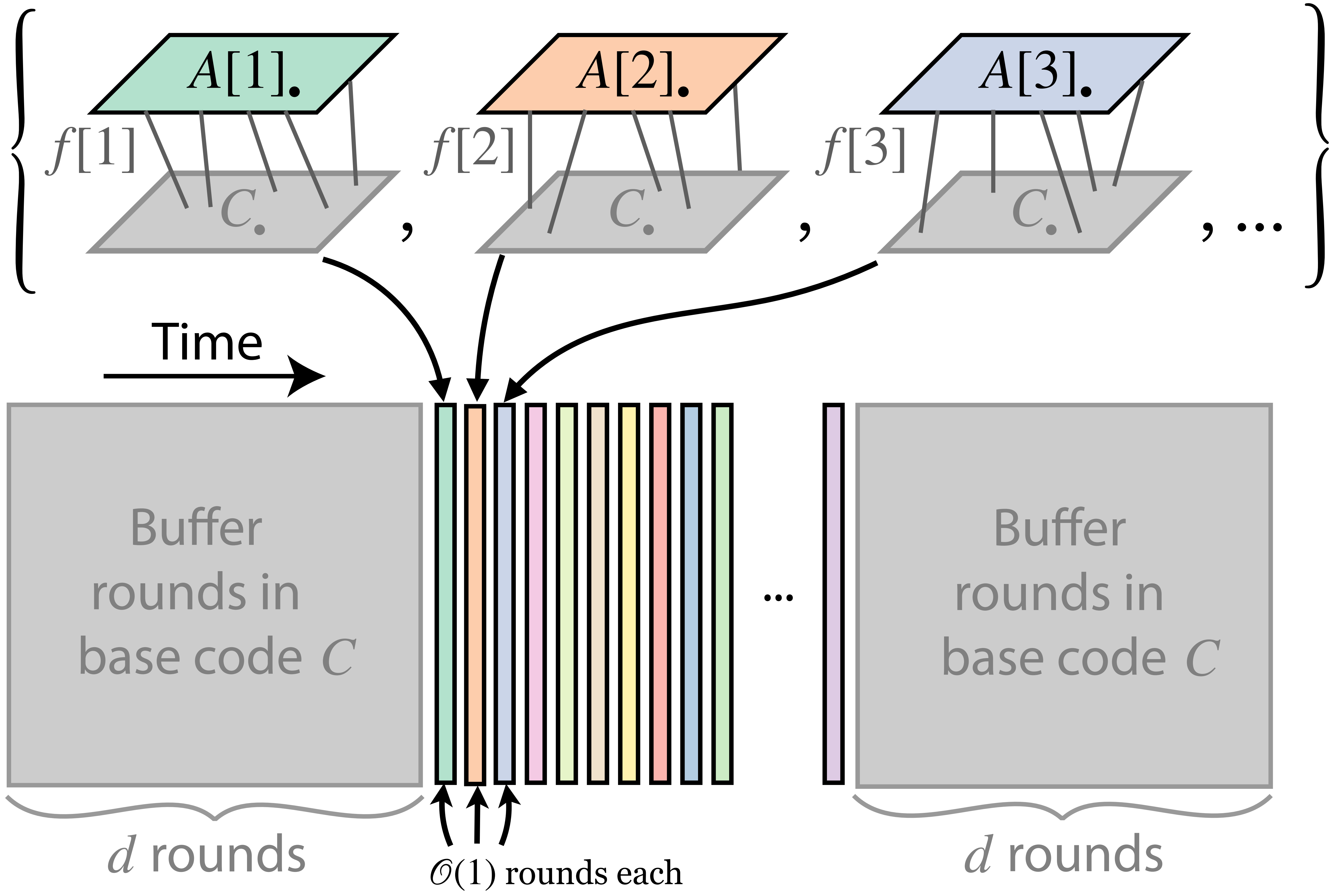
- The compacted code $\mathrm{CC}_\bullet$* has distance $\geq d$.

Unmeasured logical qubits have high distance

- Chain maps are partially injective.

Error propagation is bounded

*a CSS code defined by *t* surgery operations



Compacted code

Compacted code

- Theoretical tool to show that surgery does not reduce the code distance

Compacted code

- Theoretical tool to show that surgery does not reduce the code distance

For $i \in [t]$ surgeries, the compacted code is:

Compacted code

- Theoretical tool to show that surgery does not reduce the code distance

For $i \in [t]$ surgeries, the compacted code is:

$$
\begin{tikzcd}
 & \text{Meta-checks} & \text{Z checks} & \text{Qubits} & \text{X checks} \\
A[i]_\bullet = & \left[\bigoplus_{i\in[t]} A[i]_2\right] \arrow[r, "\partial_{A,2}"] \arrow[dr, "\bar{f}_2"'] & \left[\bigoplus_{i\in[t]} A[i]_1\right] \arrow[r, "\partial_{A,1}"] \arrow[dr, "\bar{f}_1"'] & \left[\bigoplus_{i\in[t]} A[i]_0\right] \arrow[r, "\partial_{A,0}"] \arrow[dr, "\bar{f}_0"'] & \left[\bigoplus_{i\in[t]} A[i]_{-1}\right] \\
C_\bullet = & & C_2 \arrow[r, "\partial_{C,2}"] & C_1 \arrow[r, "\partial_{C,1}"] & C_0
\end{tikzcd}
$$

Compacted code

- Theoretical tool to show that surgery does not reduce the code distance
- Not necessarily LDPC

For $i \in [t]$ surgeries, the compacted code is:

$$\begin{tikzcd}
 & \text{Meta-checks} & \text{Z checks} & \text{Qubits} & \text{X checks} \\
A[i]_\bullet = & {\left[\bigoplus_{i\in[t]} A[i]_2\right]} \arrow[r, "\partial_{A,2}"] \arrow[dr, "\bar{f}_2"'] & {\left[\bigoplus_{i\in[t]} A[i]_1\right]} \arrow[r, "\partial_{A,1}"] \arrow[dr, "\bar{f}_1"'] & {\left[\bigoplus_{i\in[t]} A[i]_0\right]} \arrow[r, "\partial_{A,0}"] \arrow[dr, "\bar{f}_0"'] & {\left[\bigoplus_{i\in[t]} A[i]_{-1}\right]} \\
C_\bullet = & & C_2 \arrow[r, "\partial_{C,2}"] & C_1 \arrow[r, "\partial_{C,1}"] & C_0
\end{tikzcd}$$

Compacted code

- Theoretical tool to show that surgery does not reduce the code distance
- Not necessarily LDPC
- Need to consider all surgery systems together — logicals may be weight reduced across different surgery systems.

For $i \in [t]$ surgeries, the compacted code is:

Meta-checks	Z checks	Qubits	X checks

$$
\begin{tikzcd}
A[i]_\bullet = & \left[\bigoplus_{i\in[t]} A[i]_2\right] \arrow[r, "\partial_{A,2}"] \arrow[dr, "\bar{f}_2"'] & \left[\bigoplus_{i\in[t]} A[i]_1\right] \arrow[r, "\partial_{A,1}"] \arrow[dr, "\bar{f}_1"'] & \left[\bigoplus_{i\in[t]} A[i]_0\right] \arrow[r, "\partial_{A,0}"] \arrow[dr, "\bar{f}_0"'] & \left[\bigoplus_{i\in[t]} A[i]_{-1}\right] \\
& C_\bullet = & C_2 \arrow[r, "\partial_{C,2}"] & C_1 \arrow[r, "\partial_{C,1}"] & C_0
\end{tikzcd}
$$

Low-overhead construction for the HGP code

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:

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.

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.

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-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 $[[n, O(n), O(\sqrt{n})]]$

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

No single-shot state preparation.

2D hypergraph product codes

Family of codes that can achieve $[[n, O(n), O(\sqrt{n})]]$

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

No single-shot state preparation.

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

2D hypergraph product codes

Family of codes that can achieve $[[n, O(n), O(\sqrt{n})]]$

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

No single-shot state preparation.

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

2D hypergraph product codes

Family of codes that can achieve $[[n, O(n), O(\sqrt{n})]]$

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

No single-shot state preparation.

$$\begin{array}{ccc} & \text{Bits} & \text{Checks} \\ \text{Classical code 1: } C_\bullet = & C_1 & \xrightarrow{\partial_C} C_0 \end{array}$$

2D hypergraph product codes

Family of codes that can achieve $[[n, O(n), O(\sqrt{n})]]$

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

No single-shot state preparation.

$$\begin{array}{ll} & \text{Bits} \quad \text{Checks} \\ \text{Classical code 1: } & C_\bullet = C_1 \xrightarrow{\partial_C} C_0 \end{array}$$

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

2D hypergraph product codes

Family of codes that can achieve $[[n, O(n), O(\sqrt{n})]]$

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

No single-shot state preparation.

Bits $\quad$ Checks

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

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

$$(C \otimes D)_\bullet = \begin{tikzcd} & C_1 \otimes D_0 \arrow[dr, "\partial_C \oplus I_D"] & \\ C_1 \otimes D_1 \arrow[ur, "I_C \oplus \partial_D"] \arrow[dr, "\partial_C \oplus I_D"'] & & C_0 \otimes D_0 \\ & C_0 \otimes D_1 \arrow[ur, "I_C \oplus \partial_D"'] & \end{tikzcd}$$

2D hypergraph product codes

Family of codes that can achieve $[[n, O(n), O(\sqrt{n})]]$

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

No single-shot state preparation.

Bits $\quad$ 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

$$(C \otimes D)_\bullet = \begin{tikzcd} & C_1 \otimes D_0 \arrow[dr, "\partial_C \oplus I_D"] & \\ C_1 \otimes D_1 \arrow[ur, "I_C \oplus \partial_D"] \arrow[dr, "\partial_C \oplus I_D"'] & & C_0 \otimes D_0 \\ & C_0 \otimes D_1 \arrow[ur, "I_C \oplus \partial_D"'] & \end{tikzcd}$$

2D hypergraph product codes

Family of codes that can achieve $[[n, O(n), O(\sqrt{n})]]$

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

No single-shot state preparation.

Bits $\quad$ 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

Z checks

$$
(C \otimes D)_\bullet =
\begin{tikzcd}
 & C_1 \otimes D_0 \arrow[dr, "\partial_C \oplus I_D"] & \\
\boxed{C_1 \otimes D_1} \arrow[ur, "I_C \oplus \partial_D"] \arrow[dr, "\partial_C \oplus I_D"'] & & C_0 \otimes D_0 \\
 & C_0 \otimes D_1 \arrow[ur, "I_C \oplus \partial_D"'] &
\end{tikzcd}
$$

2D hypergraph product codes

Family of codes that can achieve $[[n, O(n), O(\sqrt{n})]]$

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

No single-shot state preparation.

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

Z checks

Qubits

$$(C \otimes D)_\bullet = \begin{tikzcd} & C_1 \otimes D_0 \arrow[dr, "\partial_C \oplus I_D"] & \\ C_1 \otimes D_1 \arrow[ur, "I_C \oplus \partial_D"] \arrow[dr, "\partial_C \oplus I_D"'] & & C_0 \otimes D_0 \\ & C_0 \otimes D_1 \arrow[ur, "I_C \oplus \partial_D"'] & \end{tikzcd}$$

2D hypergraph product codes

Family of codes that can achieve $[[n, O(n), O(\sqrt{n})]]$

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

No single-shot state preparation.

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

Z checks | Qubits | X checks

$$
(C \otimes D)_\bullet = \begin{tikzcd} & C_1 \otimes D_0 \arrow[dr, "\partial_C \oplus I_D"] & \\ C_1 \otimes D_1 \arrow[ur, "I_C \oplus \partial_D"] \arrow[dr, "\partial_C \oplus I_D"'] & & C_0 \otimes D_0 \\ & C_0 \otimes D_1 \arrow[ur, "I_C \oplus \partial_D"'] & \end{tikzcd}
$$

2D hypergraph product codes

Family of codes that can achieve $[[n, O(n), O(\sqrt{n})]]$

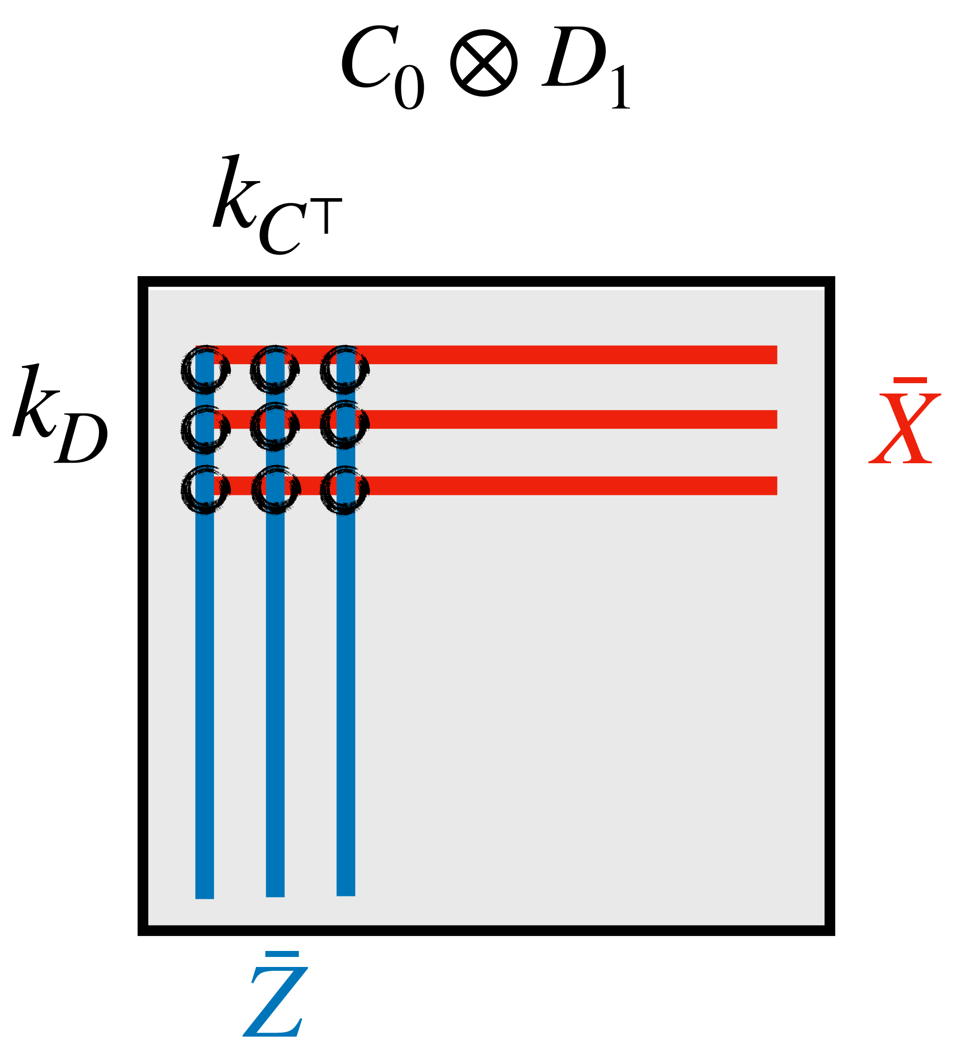
"2D" refers to the product of *two* classical codes

No single-shot state preparation.

Bits Checks

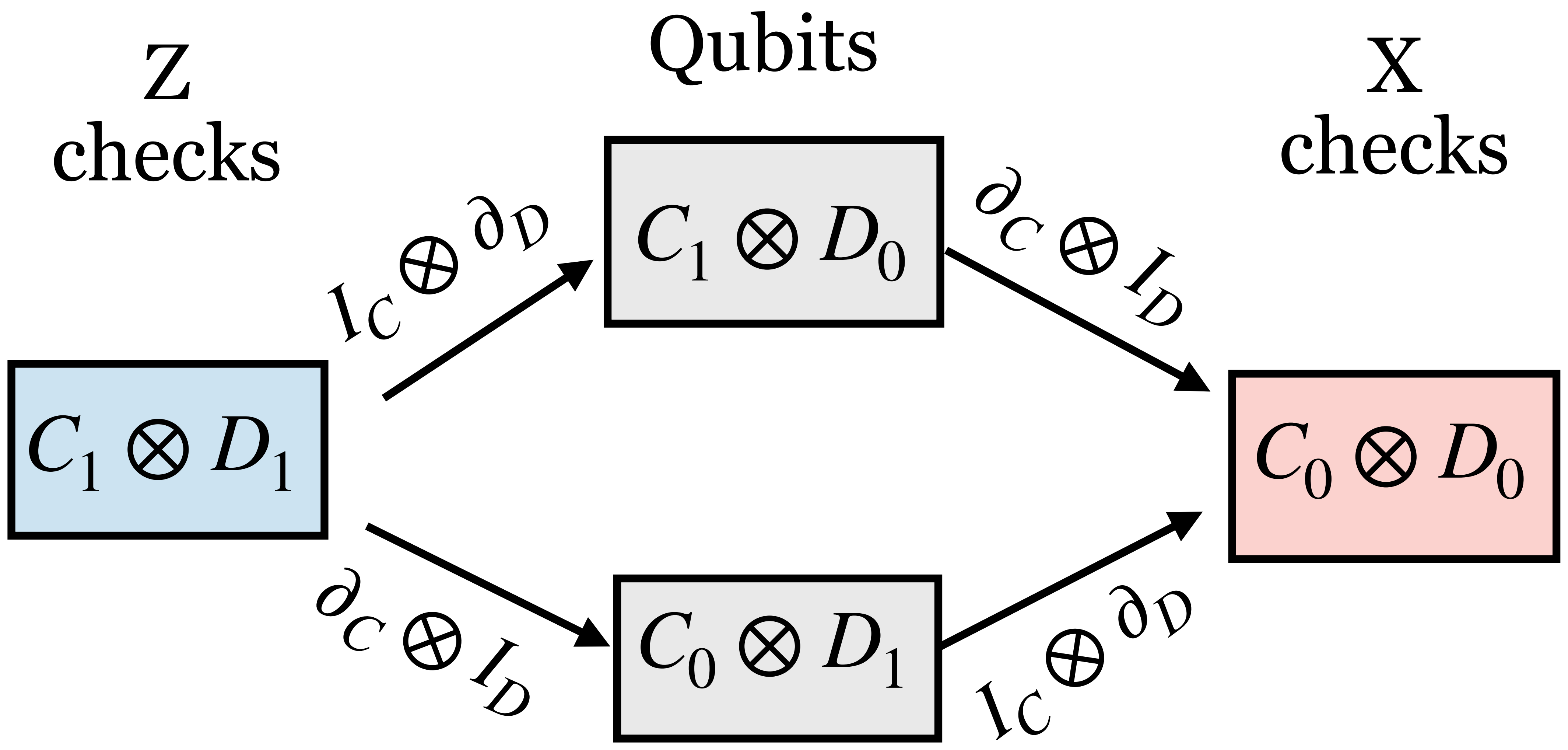
Classical code 1: $C_\bullet = C_1 \xrightarrow{\partial_C} C_0$

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



The hypergraph product of these codes is

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



2D hypergraph product codes

Family of codes that can achieve $[[n, O(n), O(\sqrt{n})]]$

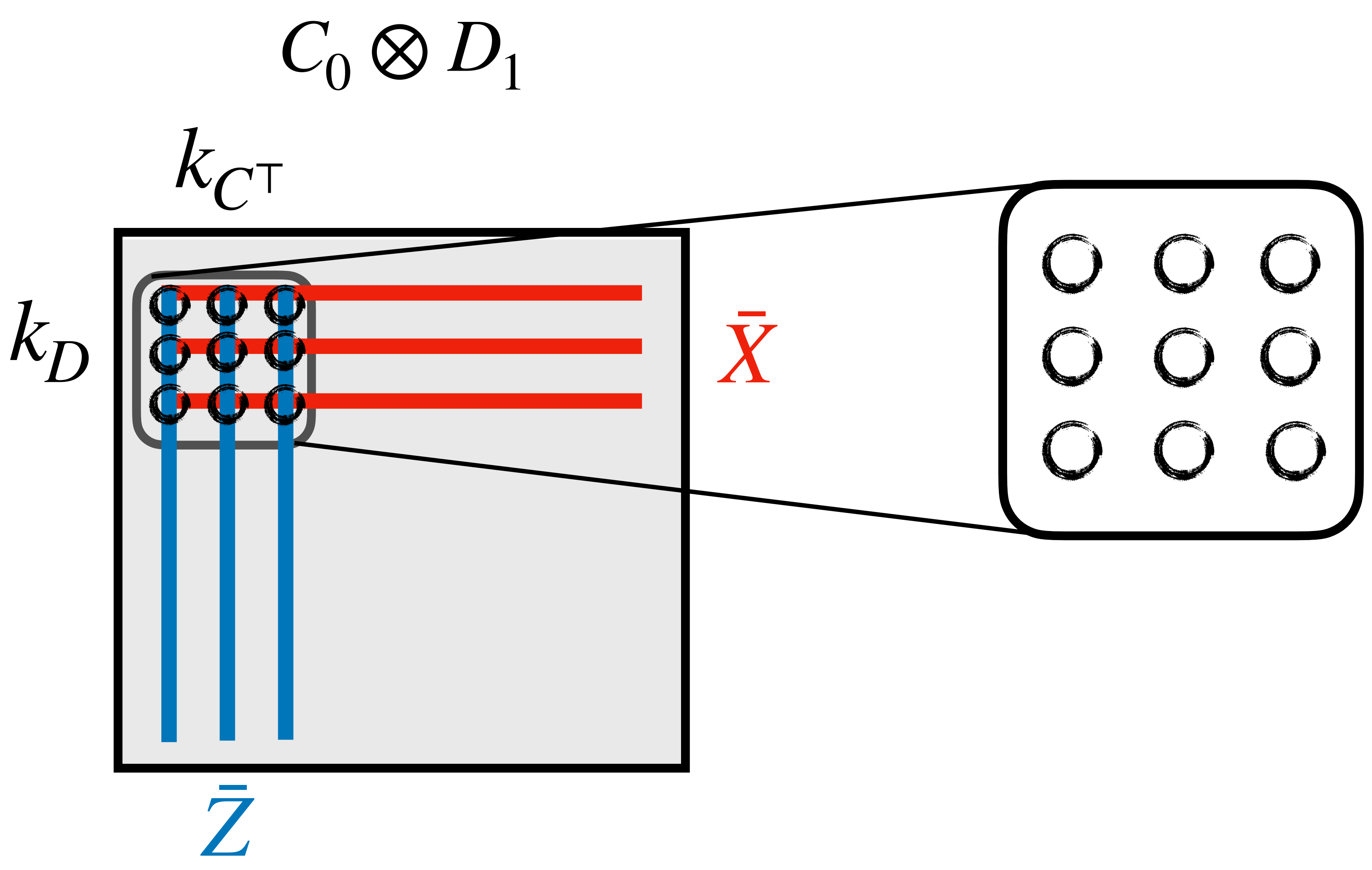
"2D" refers to the product of *two* classical codes

No single-shot state preparation.

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

Z checks

Qubits

X checks

$$(C \otimes D)_\bullet = \begin{tikzcd} & C_1 \otimes D_0 \arrow[dr, "\partial_C \oplus I_D"] & \\ C_1 \otimes D_1 \arrow[ur, "I_C \oplus \partial_D"] \arrow[dr, "\partial_C \oplus I_D"'] & & C_0 \otimes D_0 \\ & C_0 \otimes D_1 \arrow[ur, "I_C \oplus \partial_D"'] & \end{tikzcd}$$

2D hypergraph product codes

Family of codes that can achieve $[[n, O(n), O(\sqrt{n})]]$

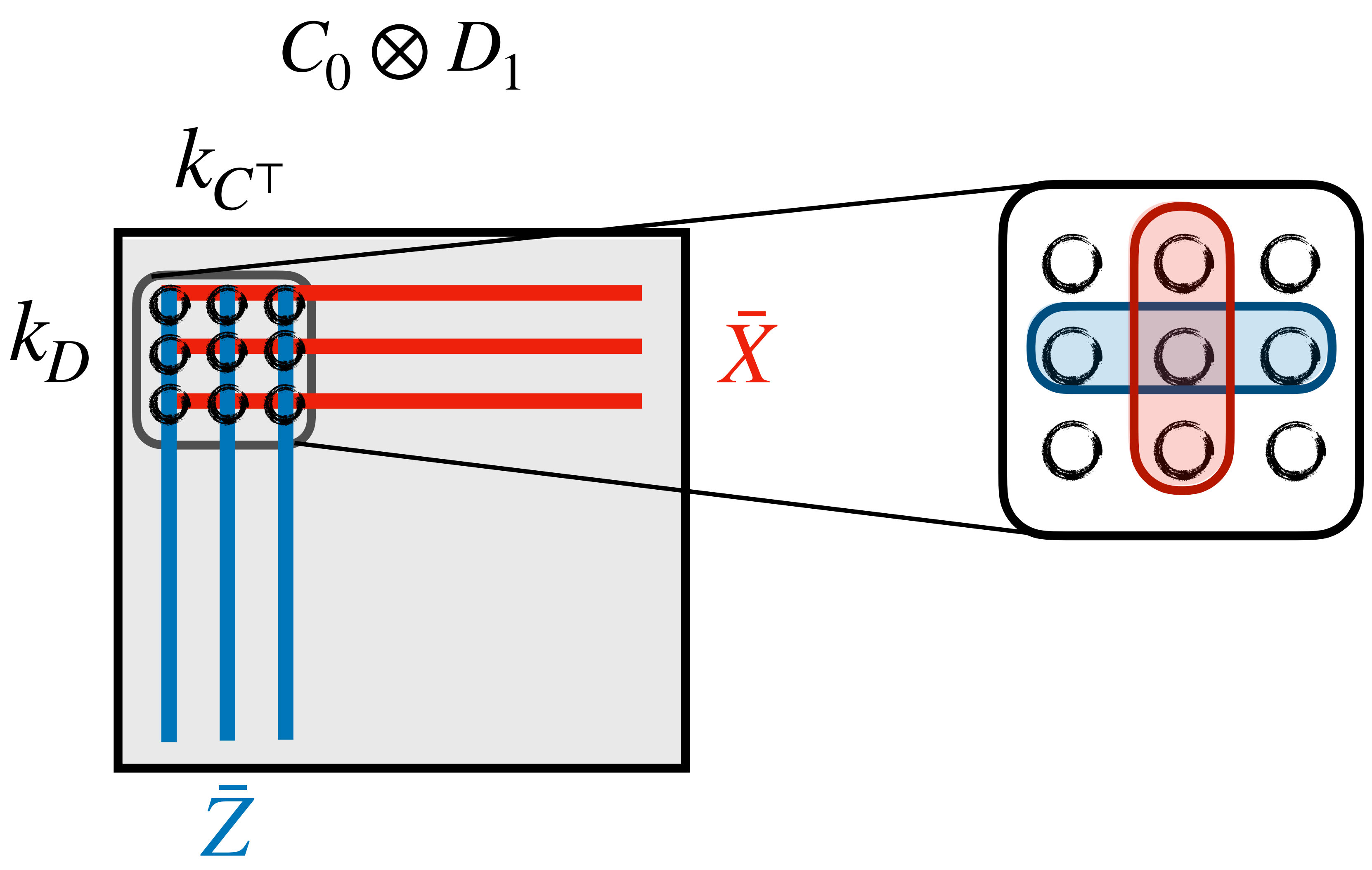
"2D" refers to the product of *two* classical codes

No single-shot state preparation.

Bits $\quad$ Checks

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

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



Each row (col) corresponds to a codeword of $D_\bullet$ ($C_\bullet^\perp$)

The hypergraph product of these codes is

Z checks $\quad$ Qubits $\quad$ X checks

$$(C \otimes D)_\bullet = \begin{tikzcd} & C_1 \otimes D_0 \arrow[dr, "\partial_C \oplus 1_D"] & \\ C_1 \otimes D_1 \arrow[ur, "1_C \oplus \partial_D"] \arrow[dr, "\partial_C \oplus 1_D"'] & & C_0 \otimes D_0 \\ & C_0 \otimes D_1 \arrow[ur, "1_C \oplus \partial_D"'] & \end{tikzcd}$$

Limited addressability, parallel measurements

We can measure rows of $\bar{Z}$ logicals (or columns of $\bar{X}$)

Constructing constant-time surgery gadgets

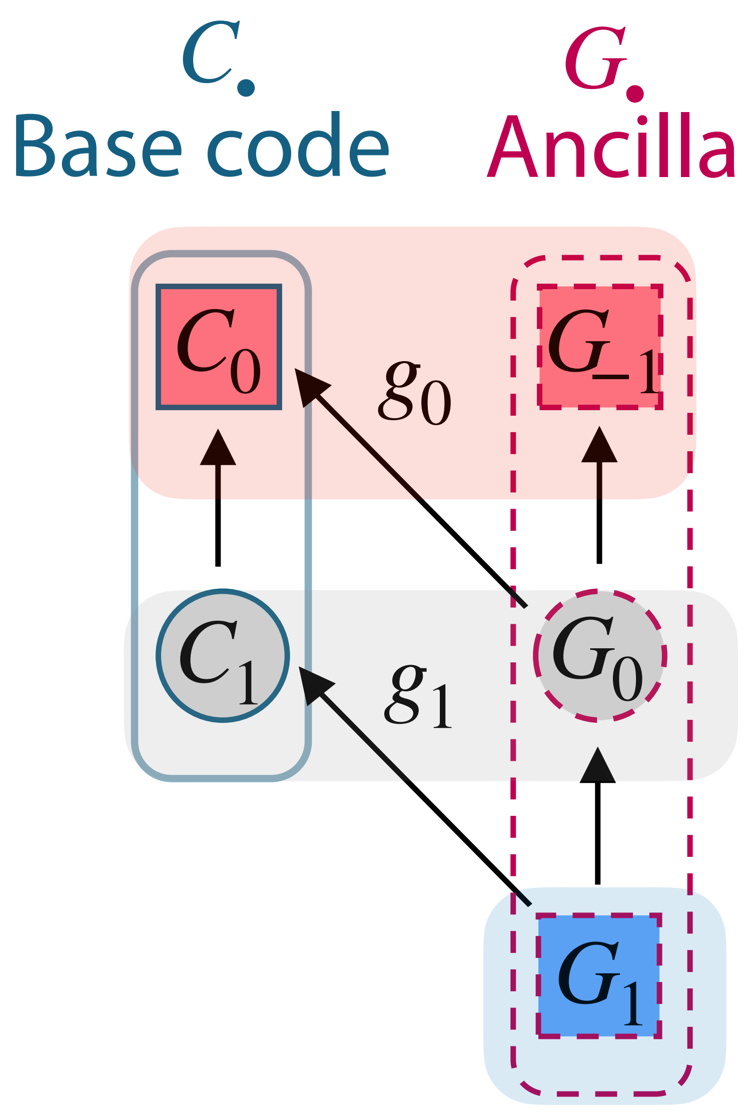
Constructing constant-time surgery gadgets

1. Apply surgery gadget
   on the classical code C.
   The ancilla = G.

Constructing constant-time surgery gadgets

1. Apply surgery gadget on the classical code $C_\bullet$. The ancilla = $G_\bullet$

Surgery on classical code
(3-term chain complex)



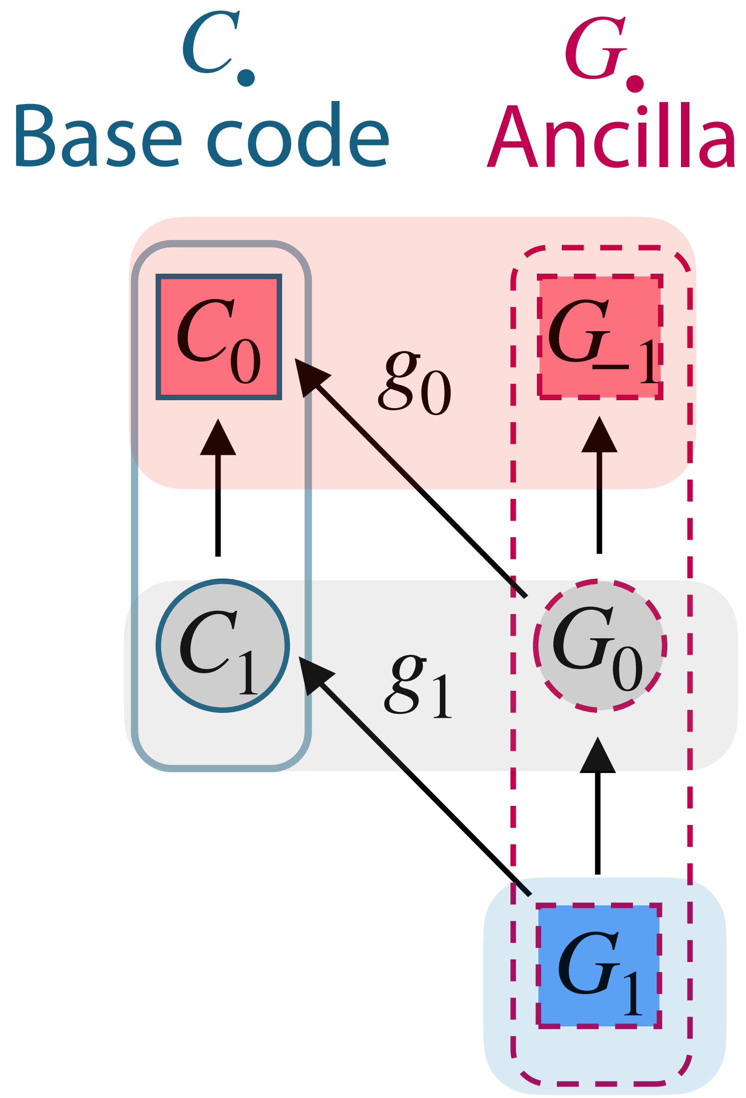
Constructing constant-time surgery gadgets

1. Apply surgery gadget on the classical code $C_\bullet$. The ancilla = $G_\bullet$
2. Take the tensor product with $D_\bullet$

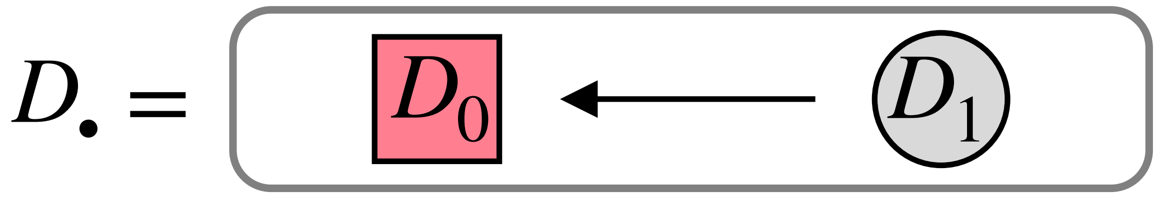
Surgery on classical code
(3-term chain complex)

$\otimes$

Classical code
(2-term chain complex)



$\otimes$



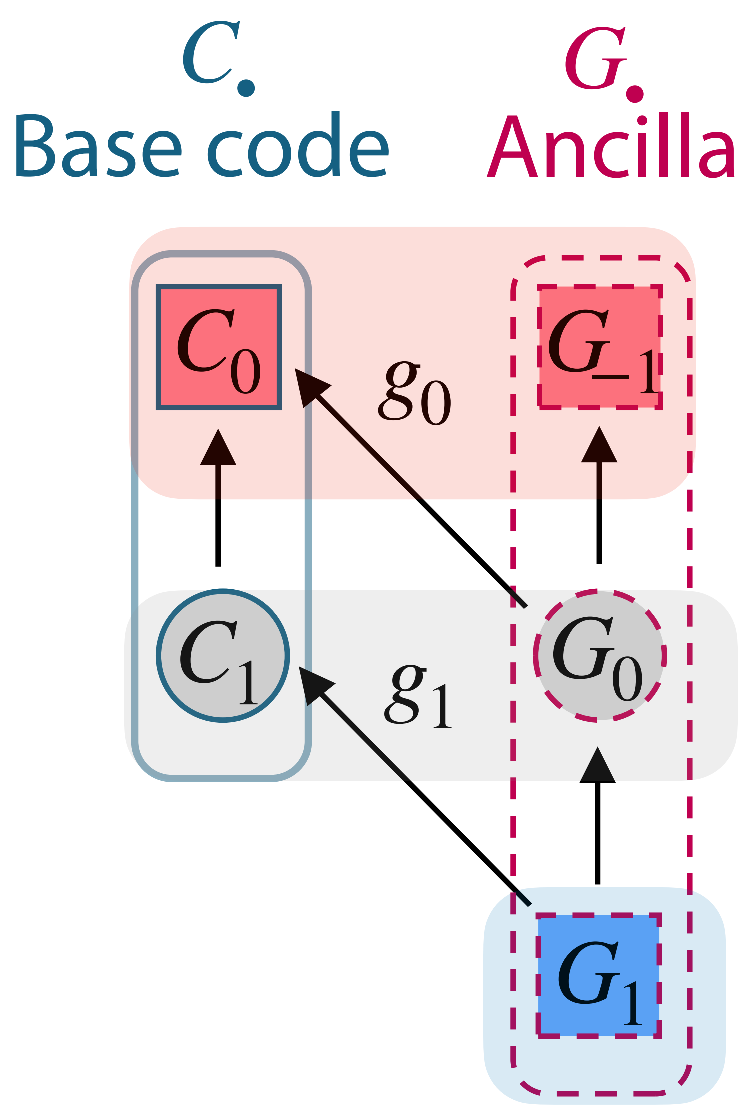
Constructing constant-time surgery gadgets

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)*

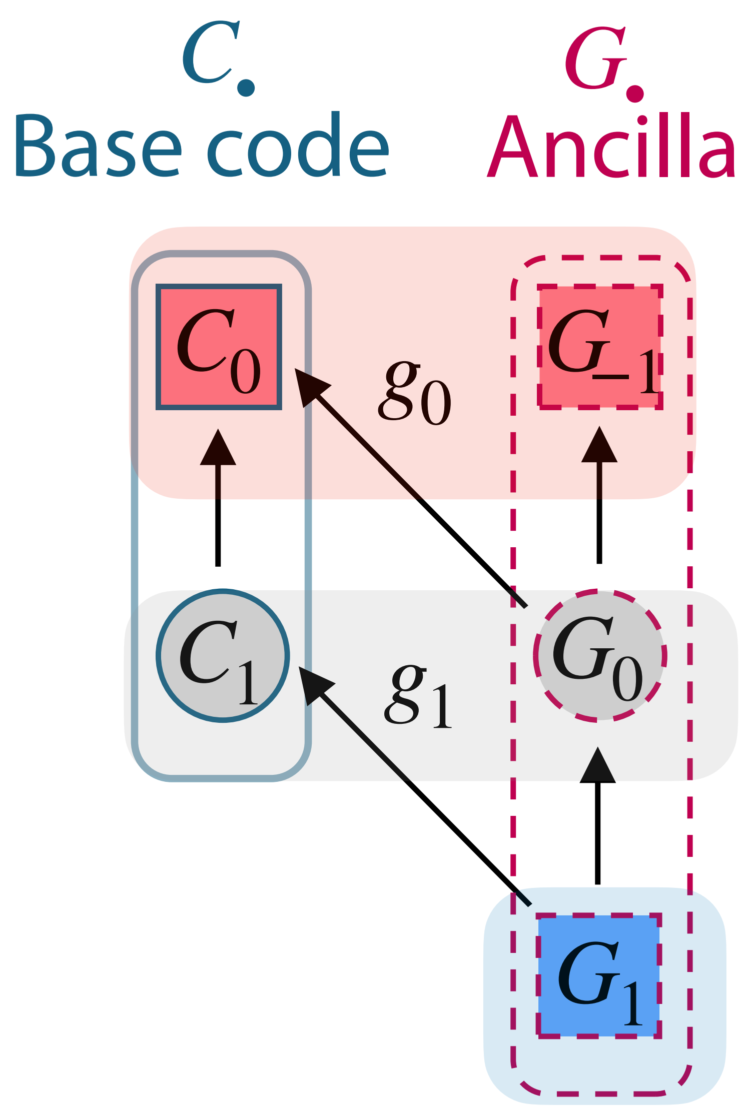


$\otimes$ $D_\bullet = \quad D_0 \leftarrow D_1$

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)*



$\otimes$ $D_\bullet = \boxed{D_0 \leftarrow D_1}$

Let $G_i^j = G_i \otimes D_j$

$C_i^j = C_i \otimes D_j$

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, "$(G + C) \otimes D$"

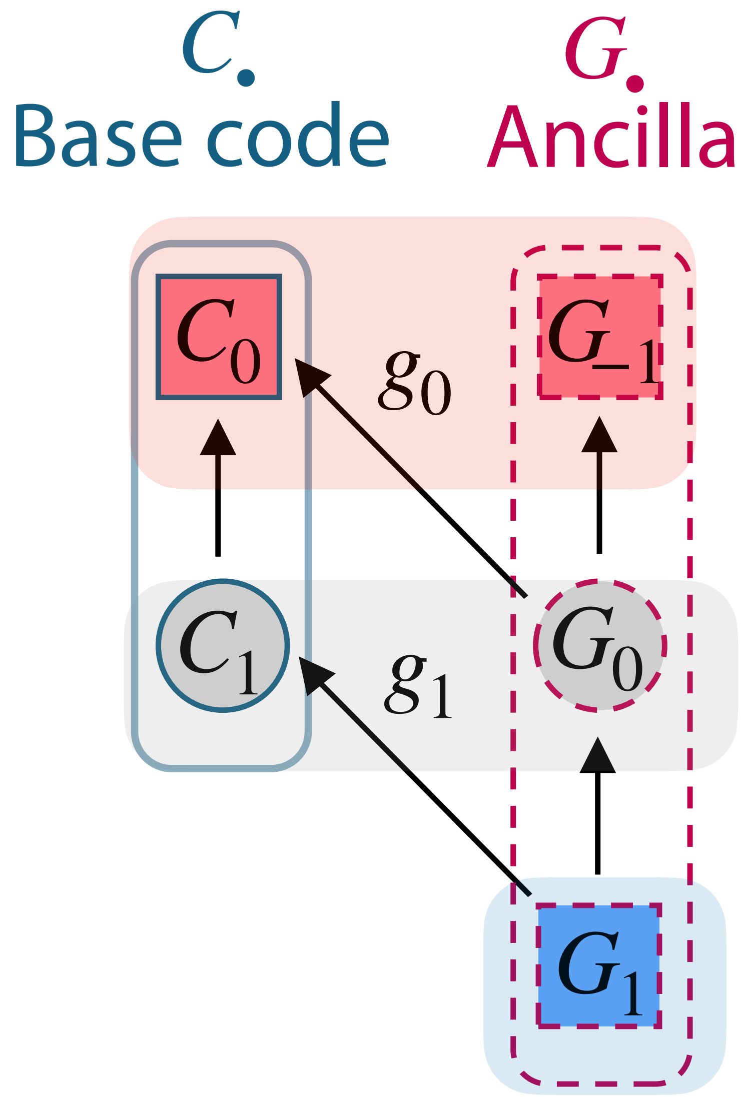
Surgery on classical code
(3-term chain complex)

$\otimes$

Classical code
(2-term chain complex)

$=$

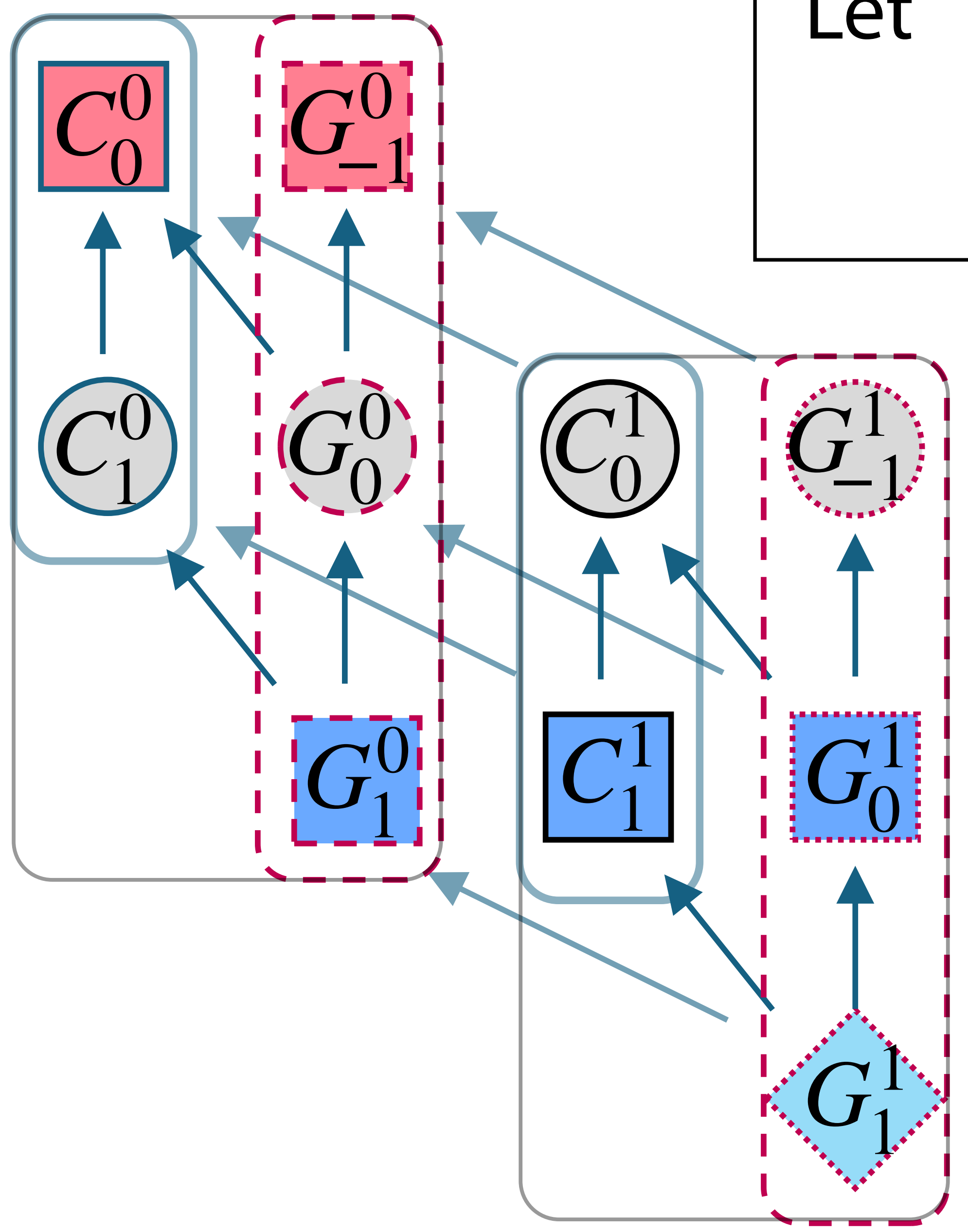
Quantum code
(4-term. chain complex)



$\otimes$

$$D_\bullet = \boxed{D_0 \longleftarrow D_1}$$

Let $G_i^j = G_i \otimes D_j$
$C_i^j = C_i \otimes D_j$



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, "$(G + C) \otimes D$"

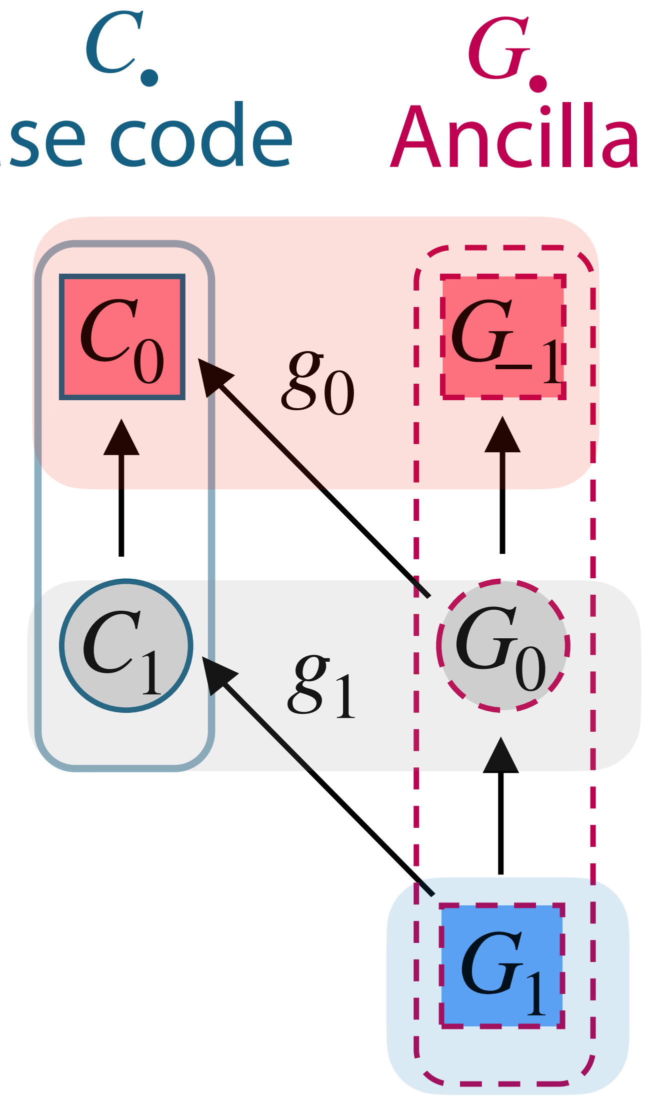
Surgery on classical code
(3-term chain complex)

$\otimes$

Classical code
(2-term chain complex)

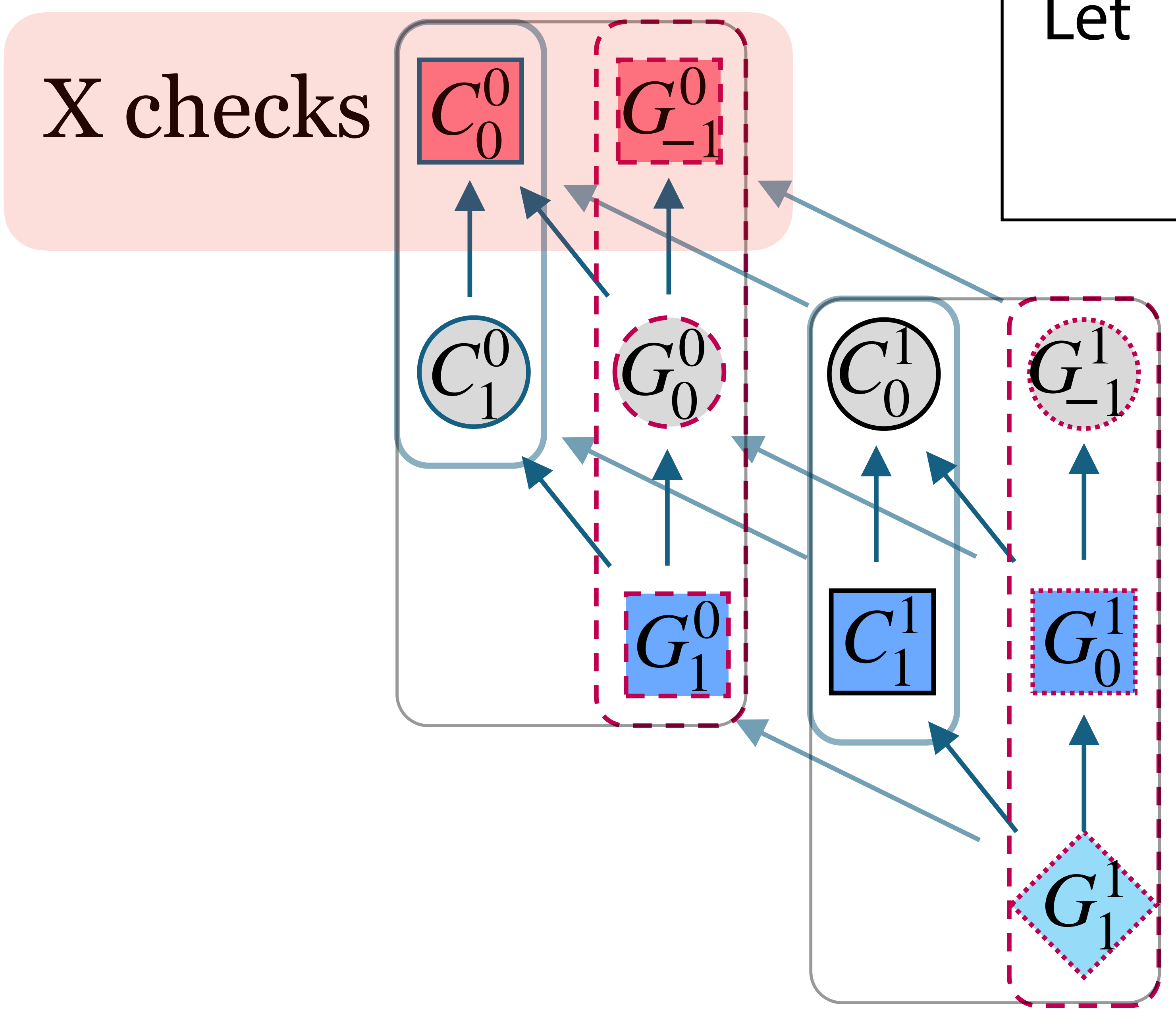
$=$

Quantum code
(4-term. chain complex)



$\otimes$

$D_\bullet = \boxed{D_0 \longleftarrow D_1}$



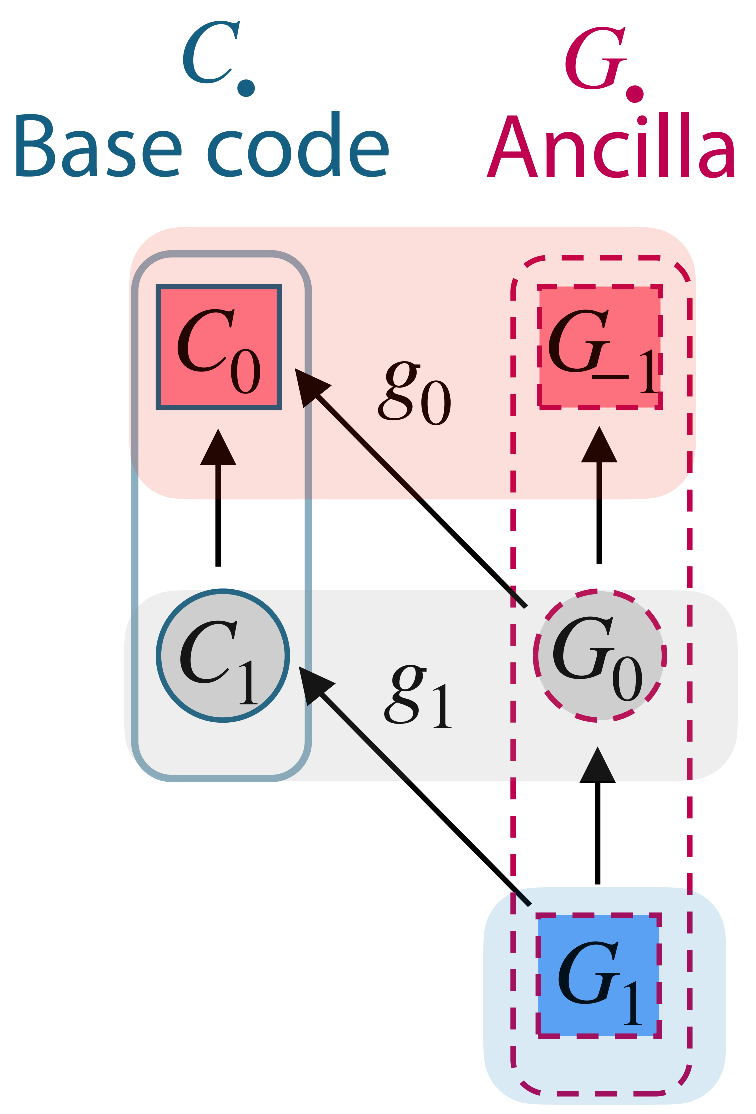
Let $G_i^j = G_i \otimes D_j$

$C_i^j = C_i \otimes D_j$

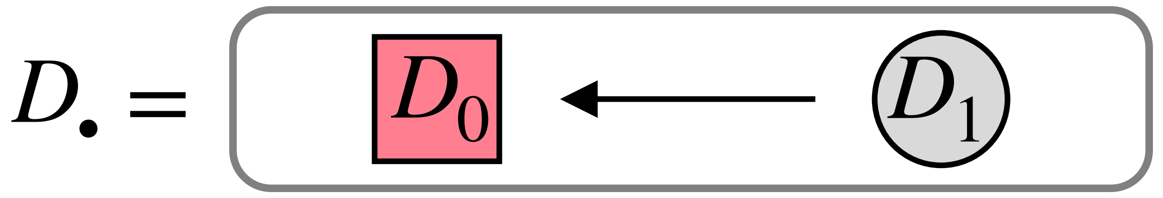
Constructing constant-time surgery gadgets

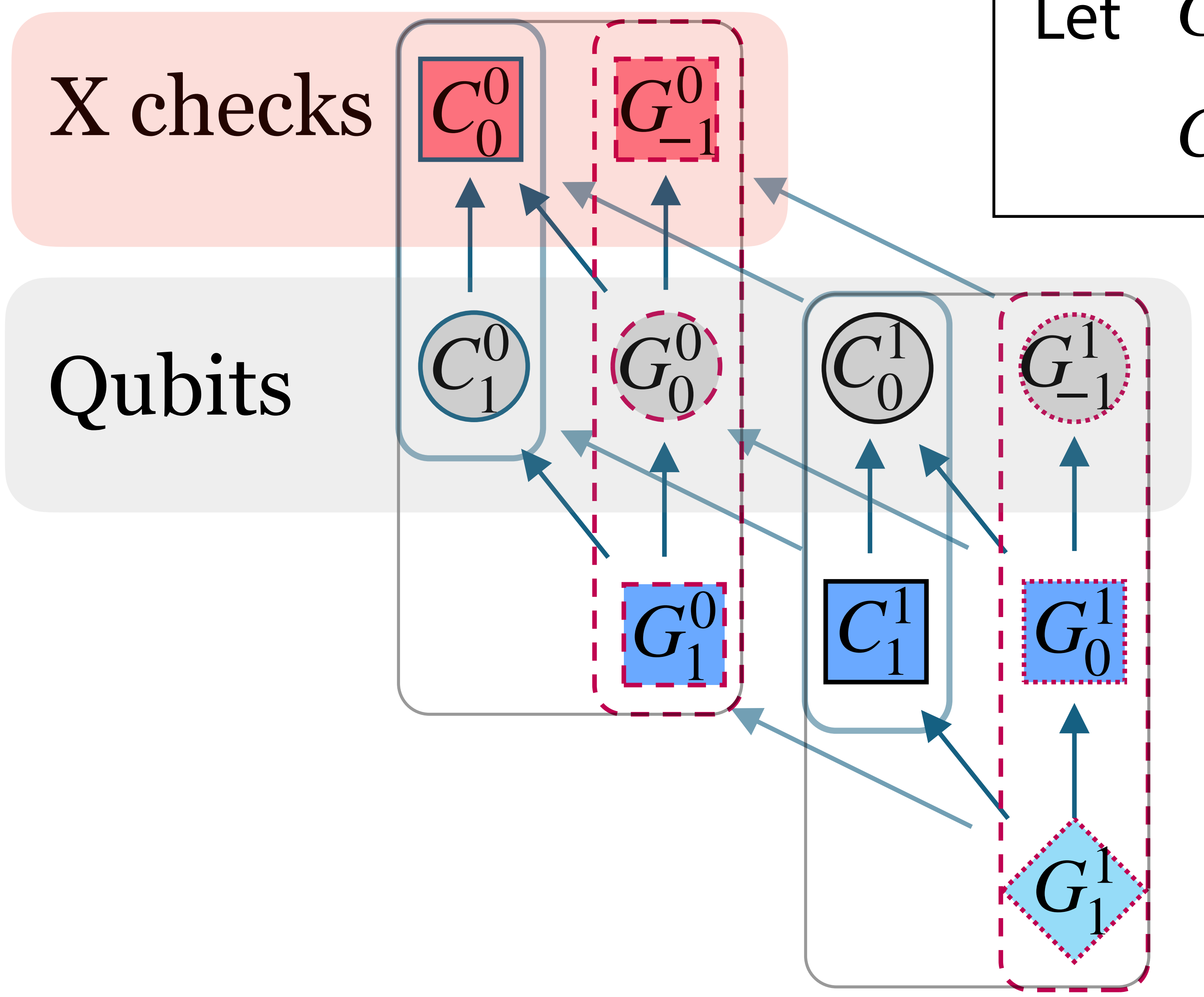
1. Apply surgery gadget on the classical code $C_\bullet$. The ancilla = $G_\bullet$
2. Take the tensor product with $D_\bullet$
3. Arrive at deformed code, “$(G + C) \otimes D$”

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



$\otimes$





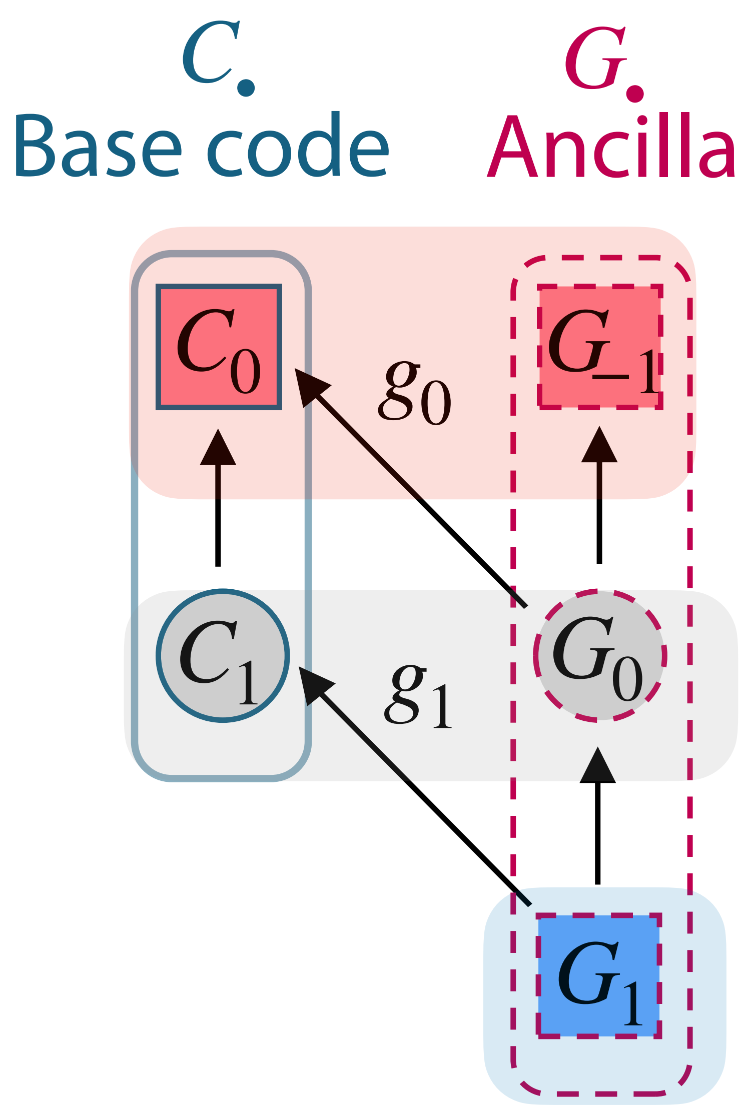
Let $G_i^j = G_i \otimes D_j$

$C_i^j = C_i \otimes D_j$

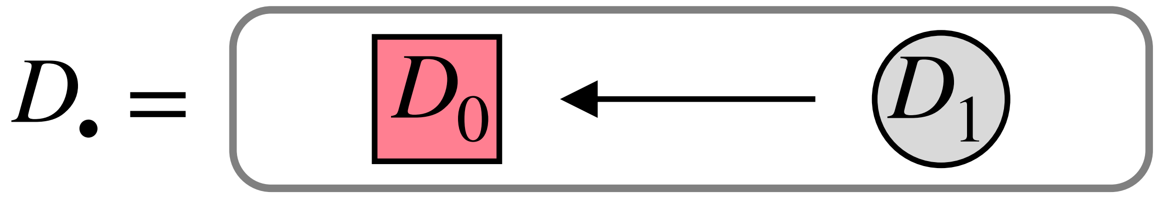
Constructing constant-time surgery gadgets

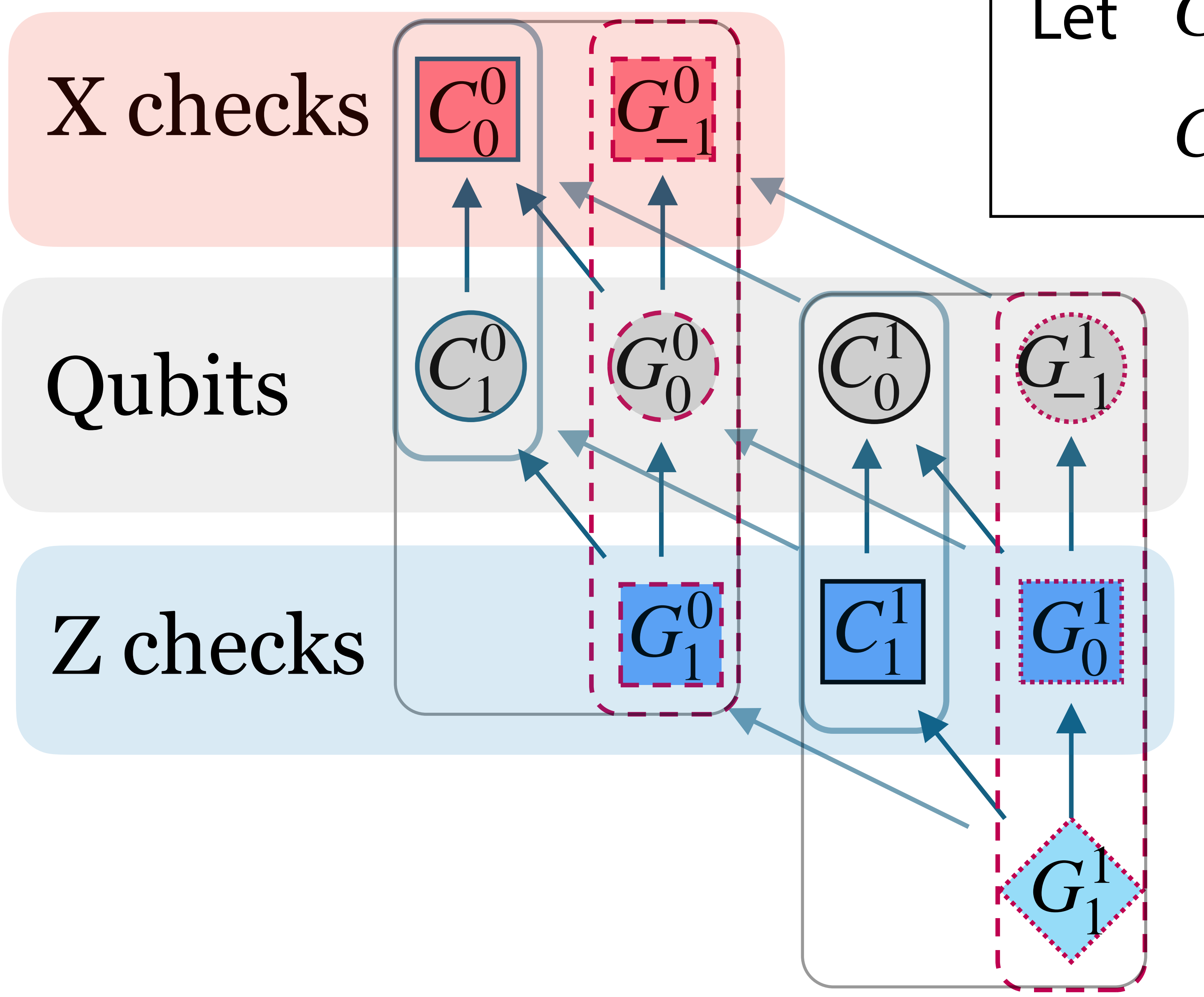
1. Apply surgery gadget on the classical code $C_\bullet$. The ancilla = $G_\bullet$
2. Take the tensor product with $D_\bullet$
3. Arrive at deformed code, "$(G + C) \otimes D$"

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



$\otimes$





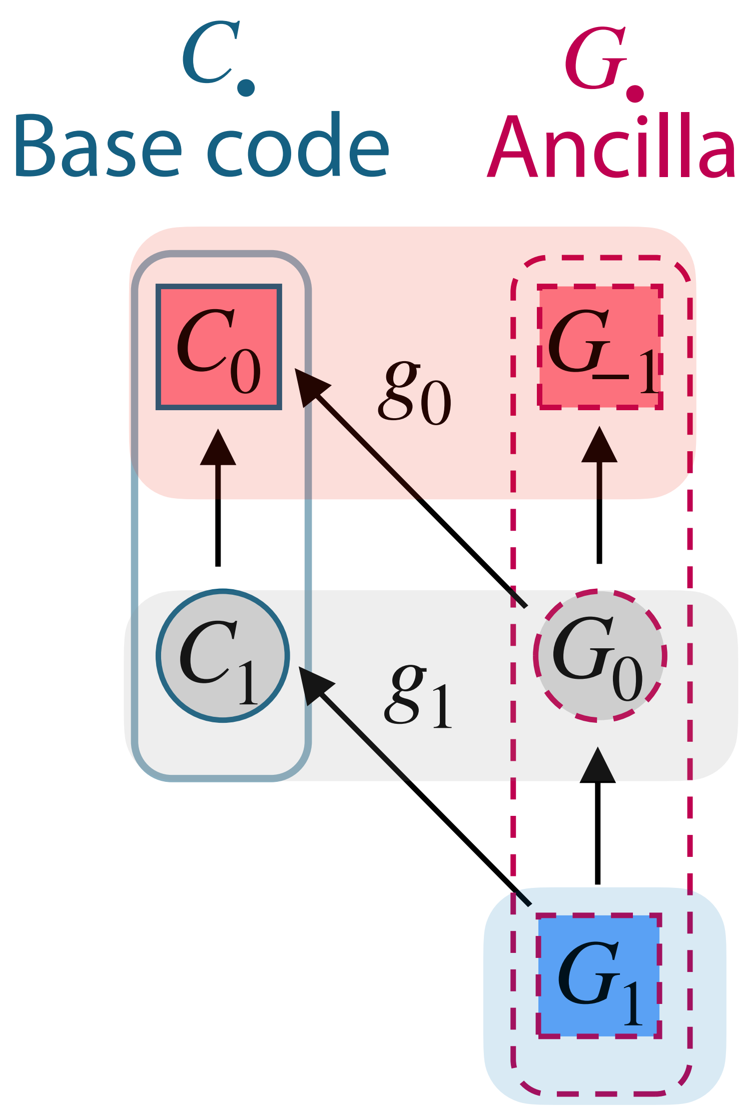
Let $G_i^j = G_i \otimes D_j$

$C_i^j = C_i \otimes D_j$

Constructing constant-time surgery gadgets

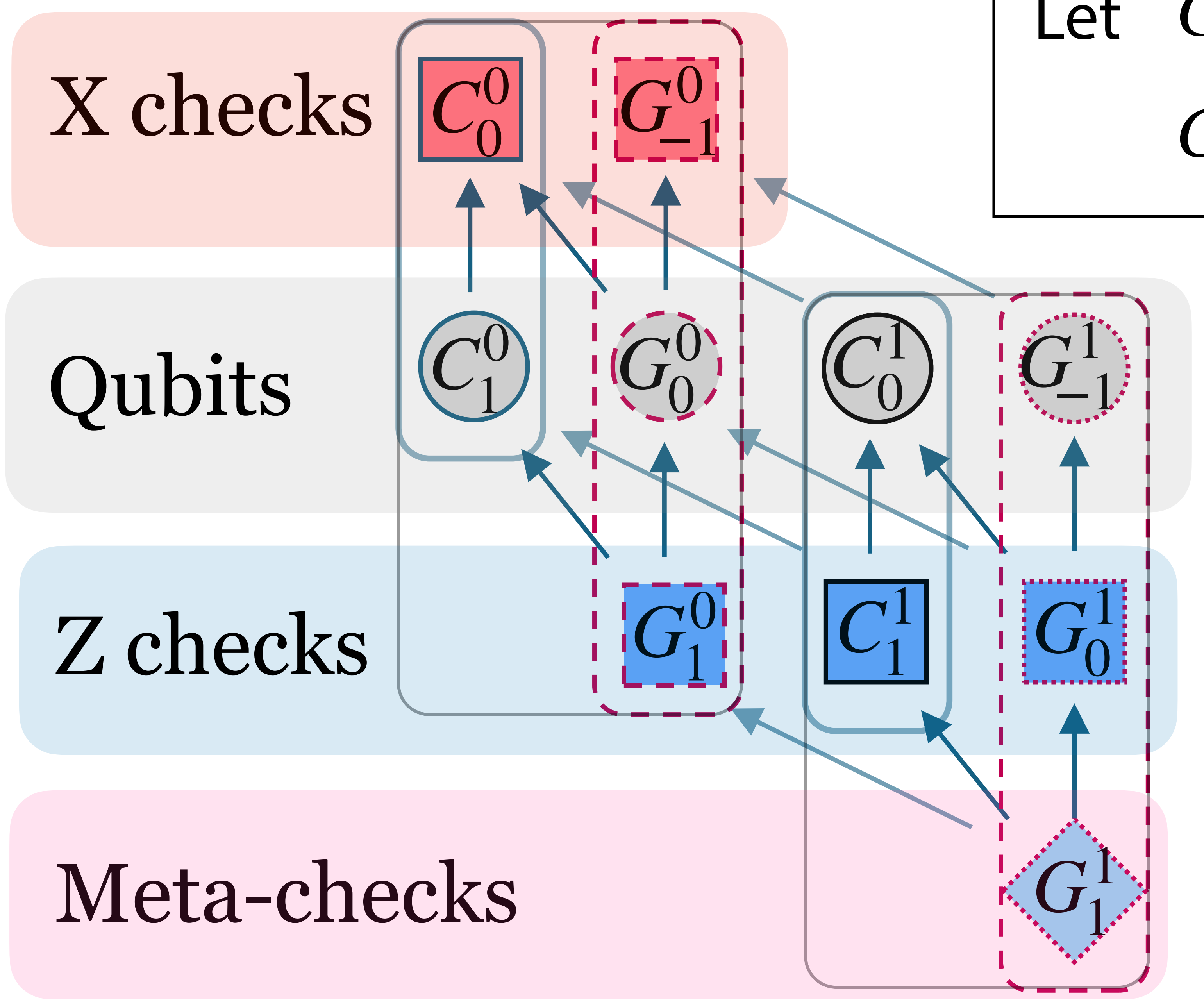
1. Apply surgery gadget on the classical code $C_\bullet$. The ancilla = $G_\bullet$
2. Take the tensor product with $D_\bullet$
3. Arrive at deformed code, "$(G + C) \otimes D$"

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



$\otimes$

$$D_\bullet = \boxed{D_0 \longleftarrow D_1}$$



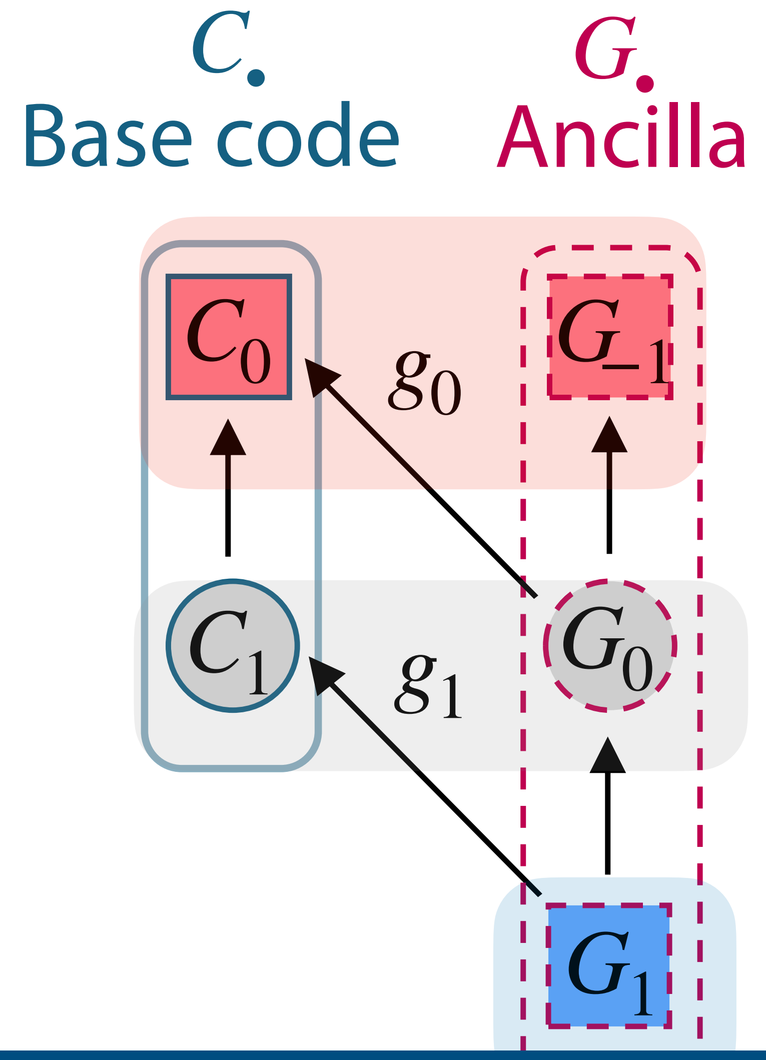
Let $G_i^j = G_i \otimes D_j$

$C_i^j = C_i \otimes D_j$

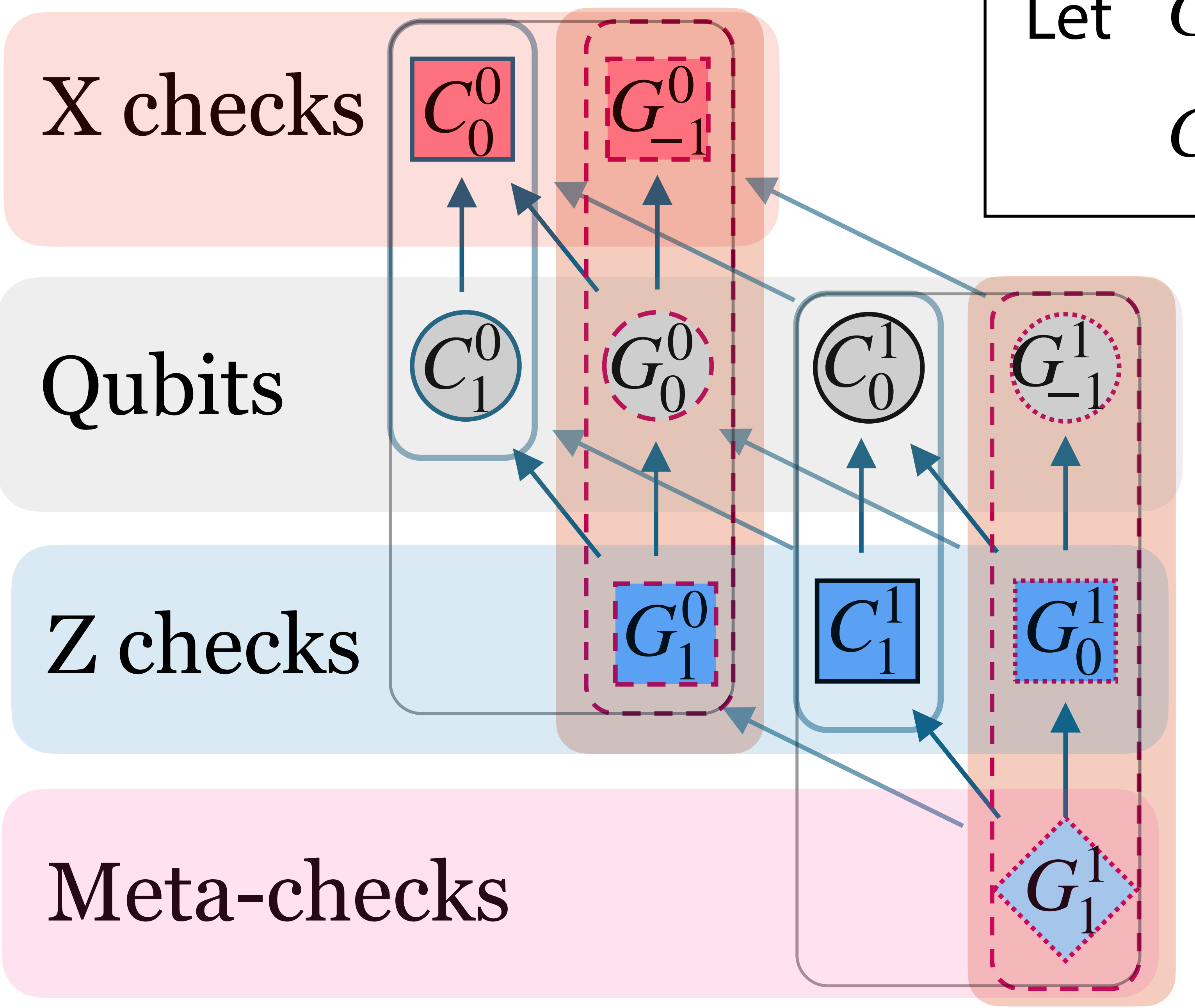
Constructing constant-time surgery gadgets

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)*



$\otimes$ $D_\bullet = \boxed{D_0 \longleftarrow D_1}$



Let $G_i^j = G_i \otimes D_j$

$C_i^j = C_i \otimes D_j$

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

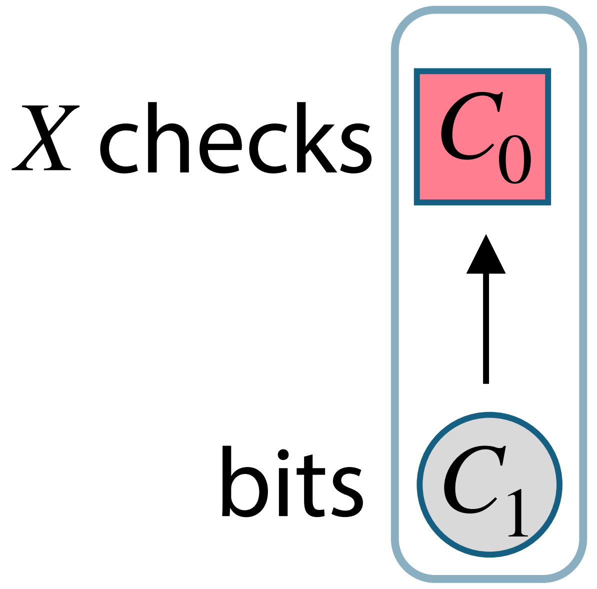
Ex. Constructing the deformed code

1. Take the base classical code C.

Ex. Constructing the deformed code

1. Take the base classical code $C_\bullet$

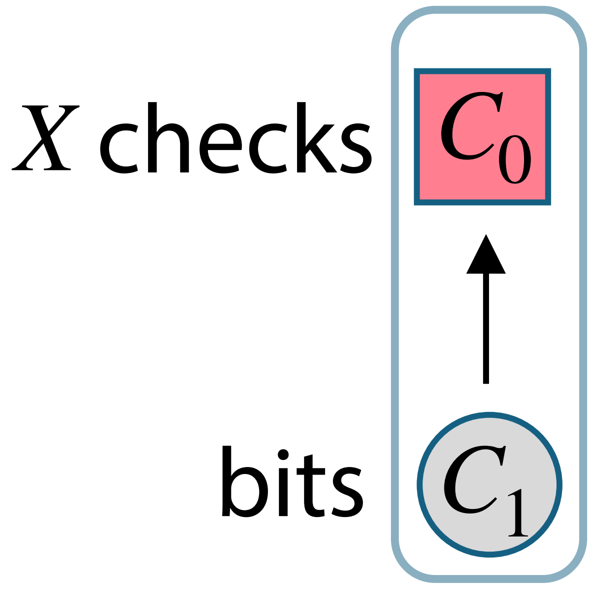
Chain complex



Ex. Constructing the deformed code

1. Take the base classical code $C_\bullet$

Chain complex



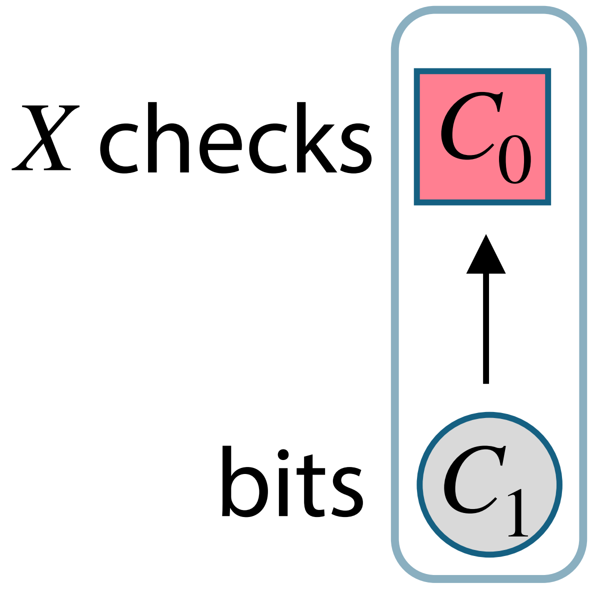
Tanner graph Ex.

[7,4,3]

Ex. Constructing the deformed code

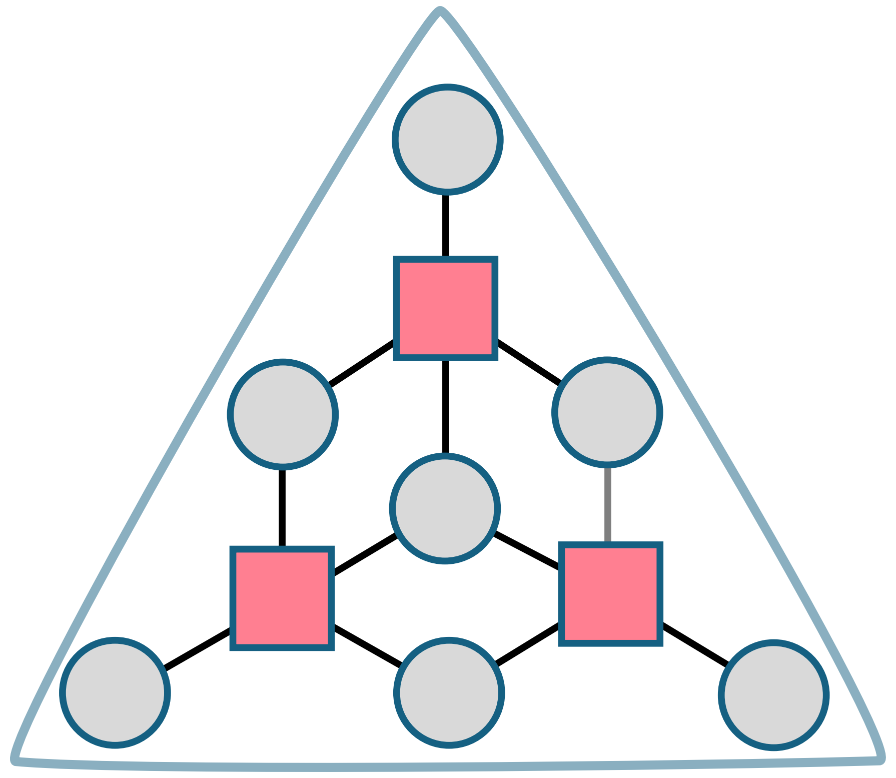
1. Take the base classical code $C_\bullet$

Chain complex



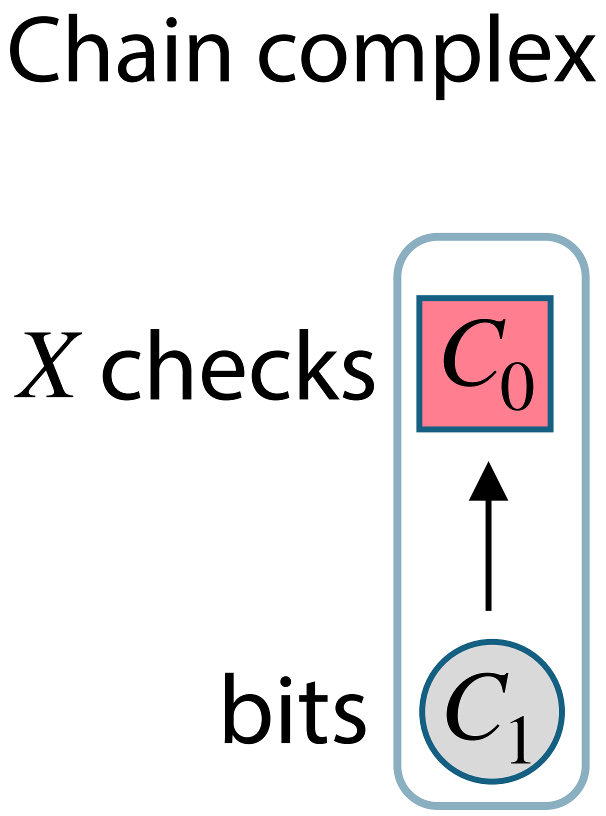
Tanner graph Ex.

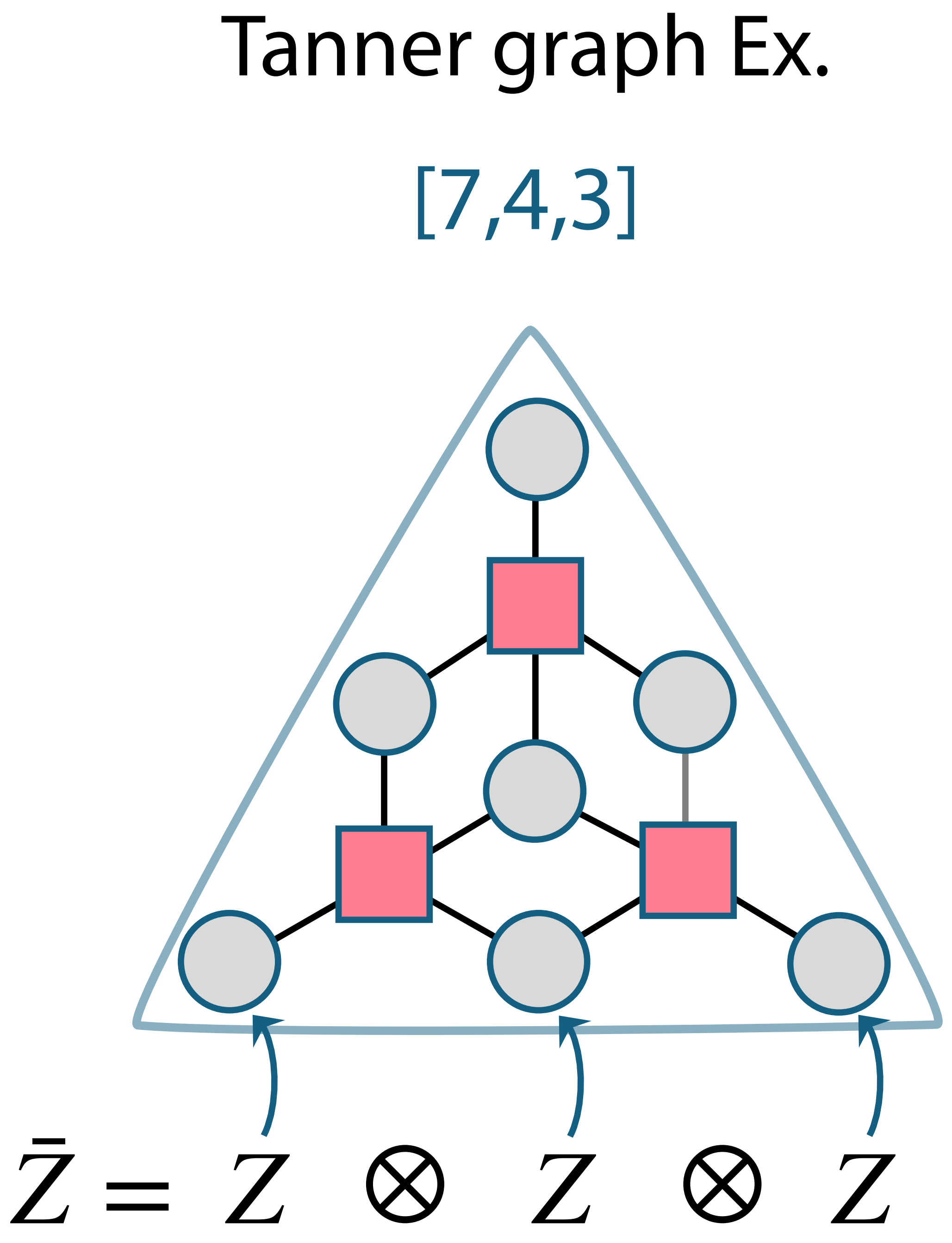
[7,4,3]



Ex. Constructing the deformed code

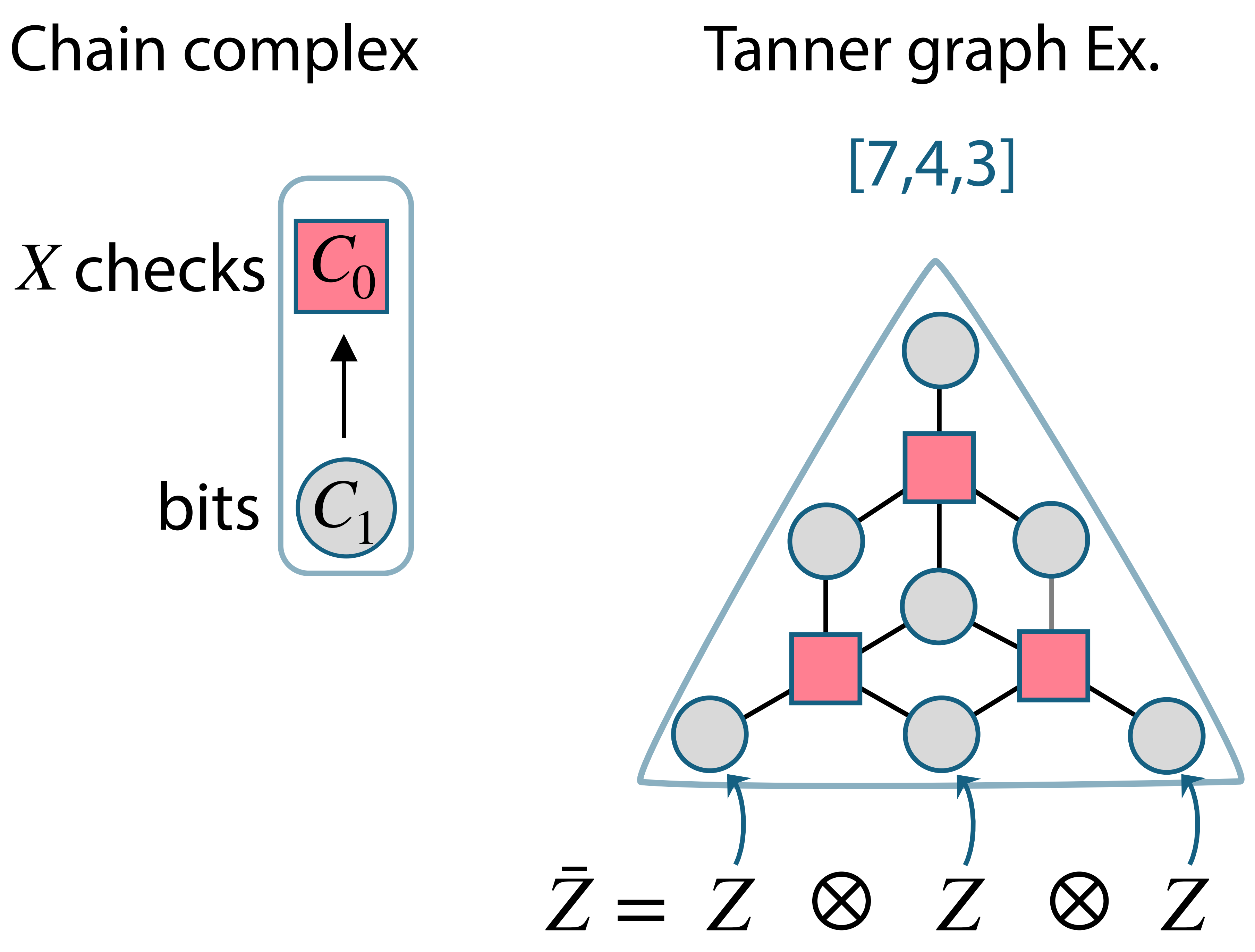
1. Take the base classical code $C_\bullet$





Ex. Constructing the deformed code

1. Take the base classical code C.



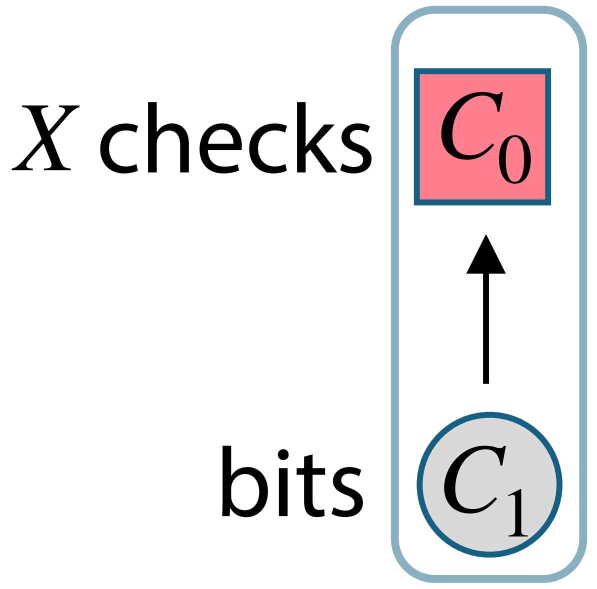
2. Augment the classical code with known surgery techniques[1,2], measure the classical codeword.

[1] Williamson, Yoder. [2410.02213] [2] Ide *et al.* PRX **15** (2025).

Ex. Constructing the deformed code

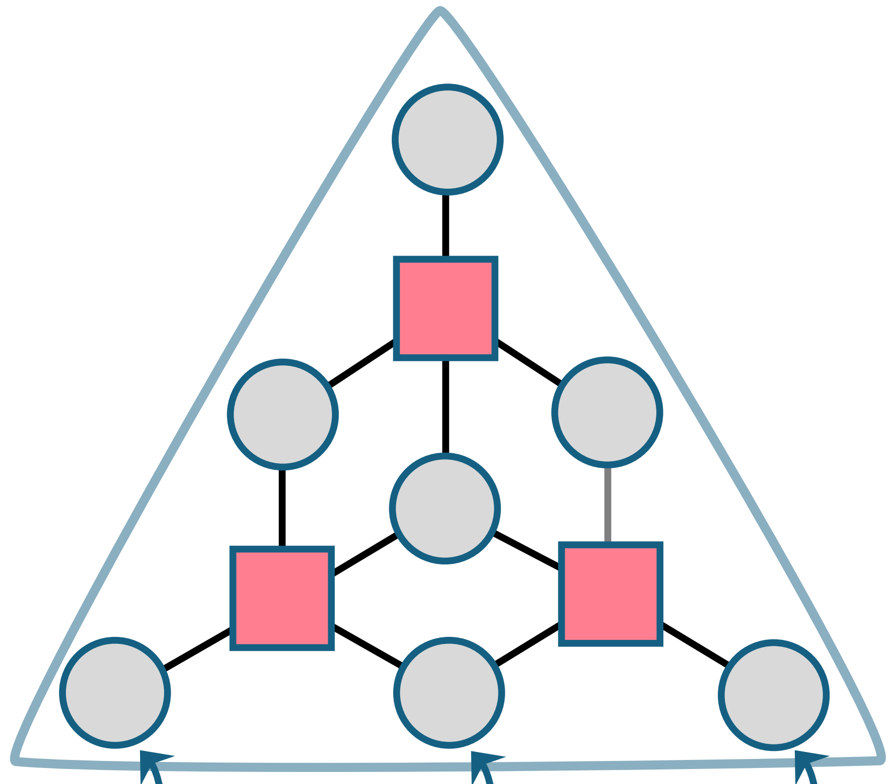
1. Take the base classical code $C_\bullet$

Chain complex



Tanner graph Ex.

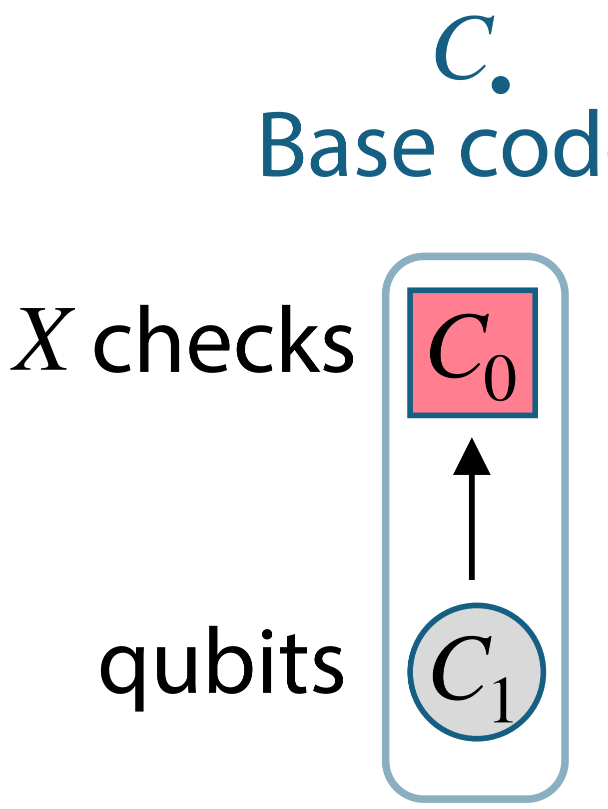
[7,4,3]



$$\bar{Z} = Z \otimes Z \otimes Z$$

2. Augment the classical code with known surgery techniques[1,2], measure the classical codeword.

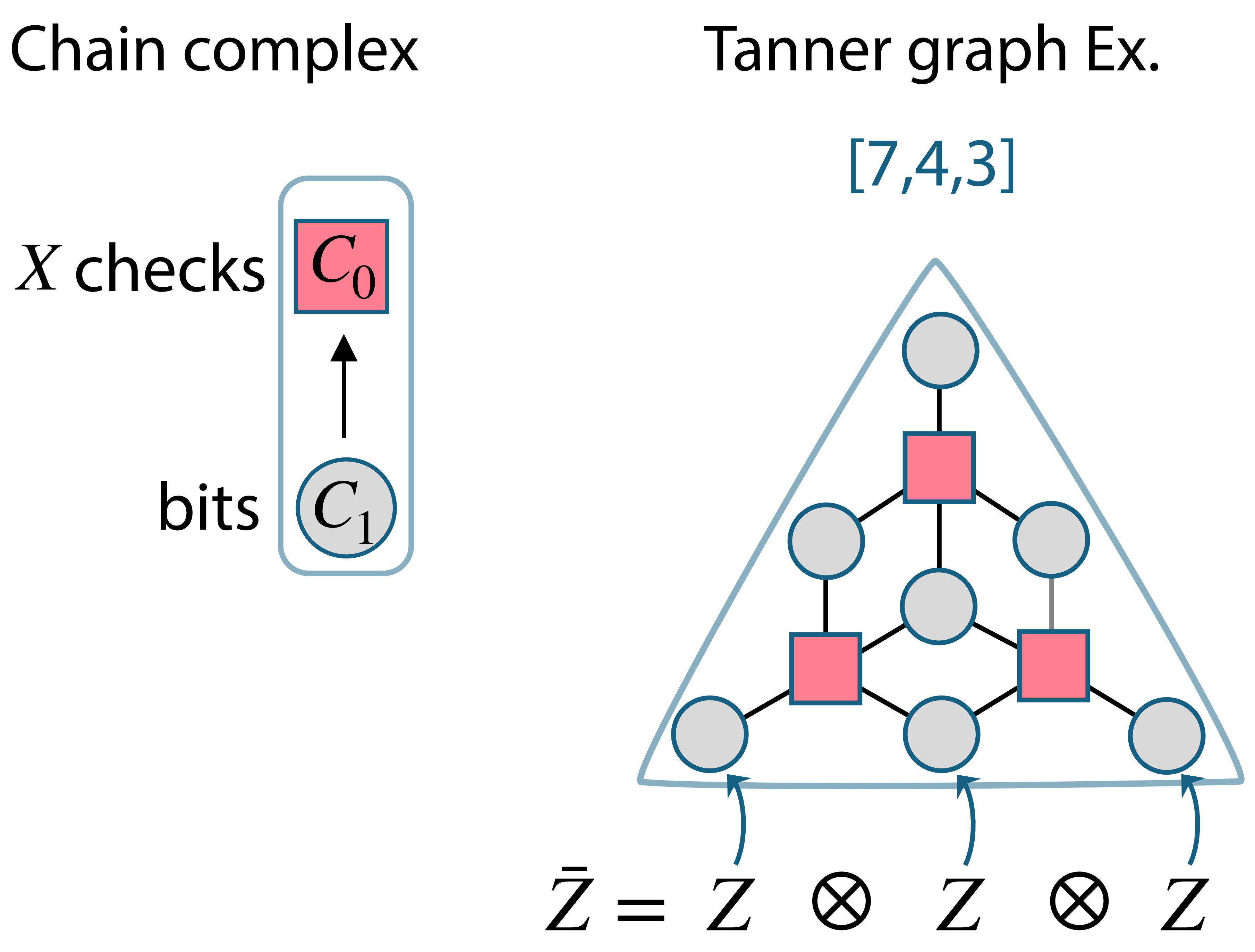
Chain complex



[1] Williamson, Yoder. [2410.02213] [2] Ide *et al.* PRX **15** (2025).

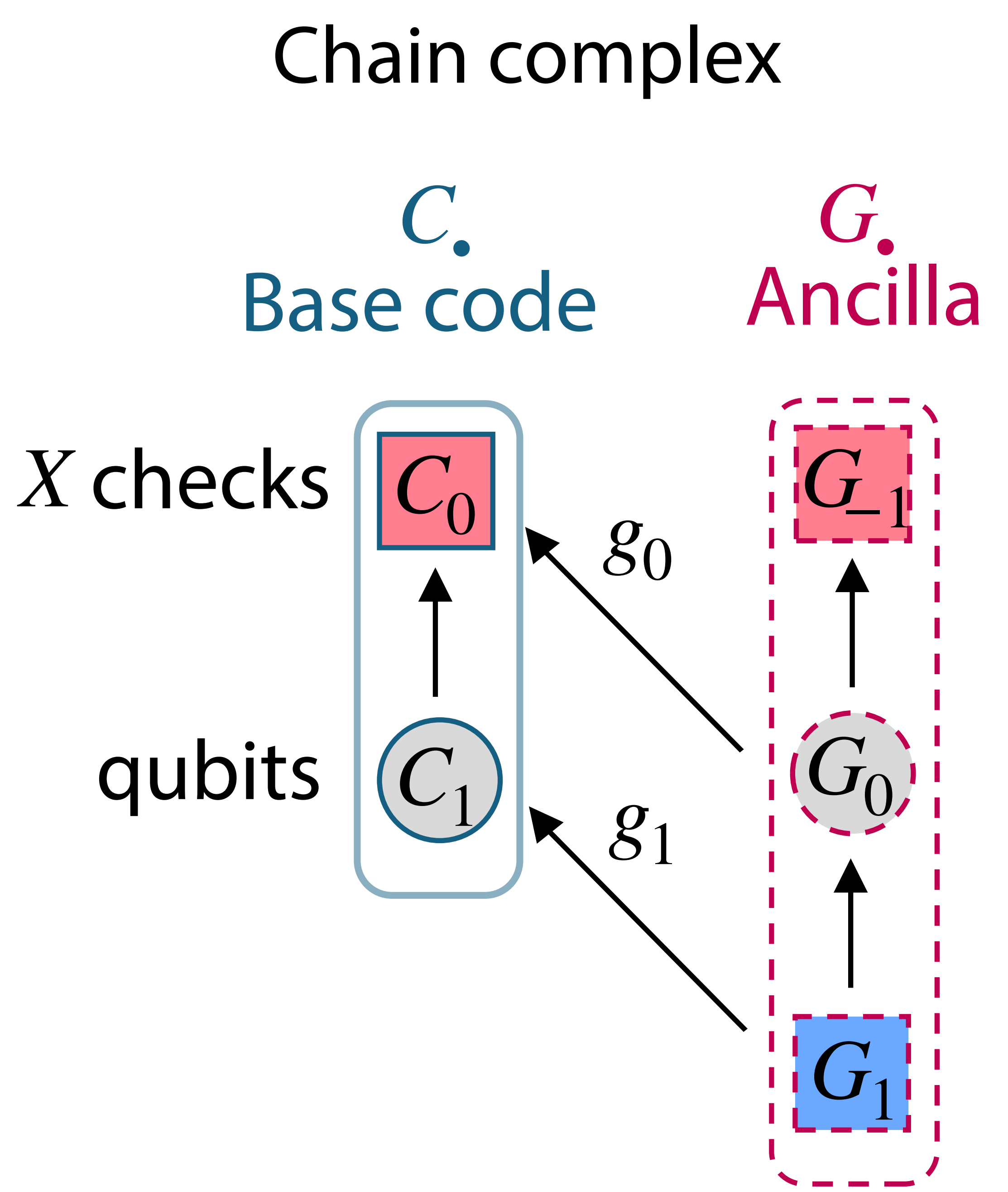
Ex. Constructing the deformed code

1. Take the base classical code $C_{\bullet}$



2. Augment the classical code with known surgery techniques[1,2], measure the classical codeword.

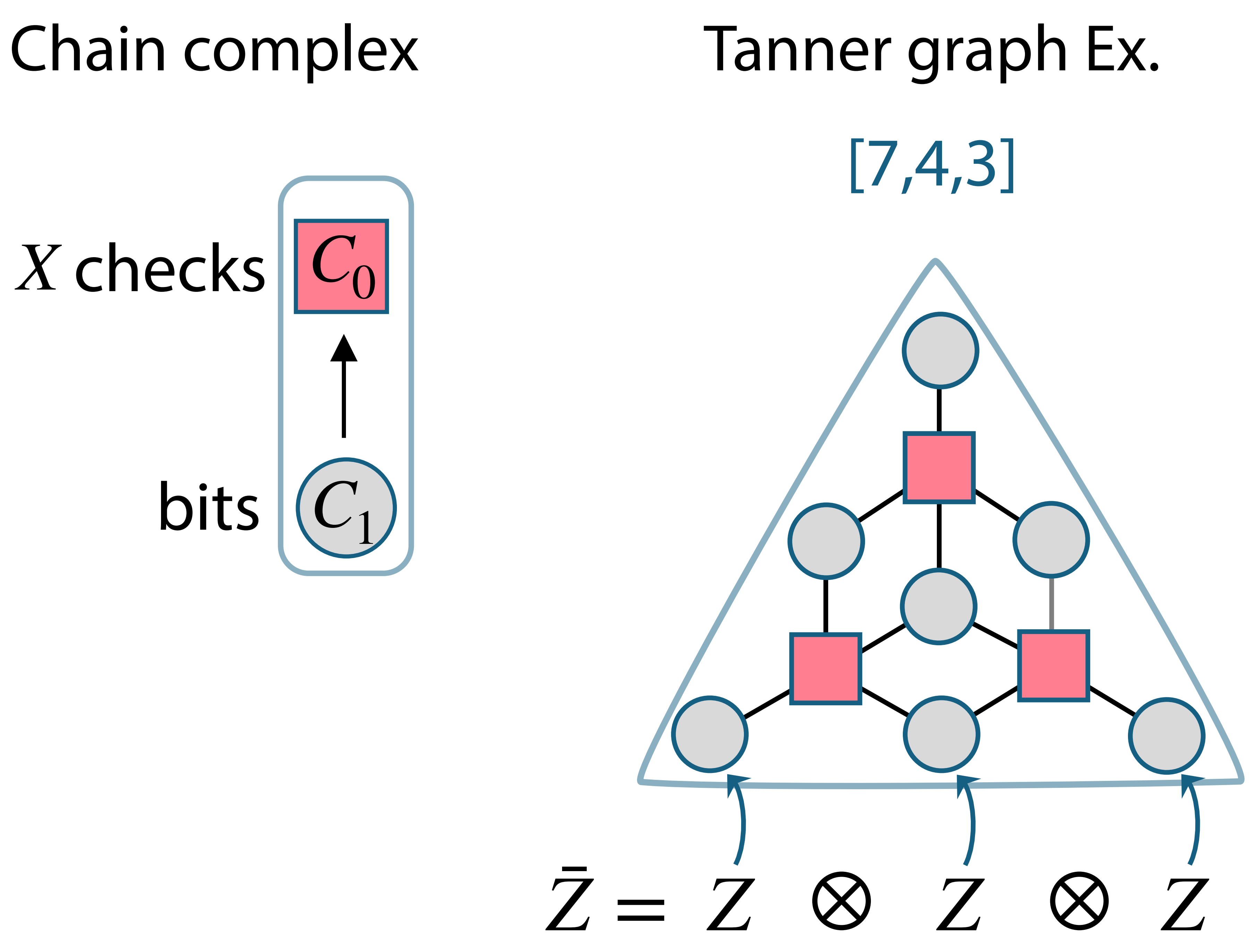
$$(g : G_{\bullet} \to C_{\bullet})$$



[1] Williamson, Yoder. [2410.02213] [2] Ide *et al.* PRX **15** (2025).

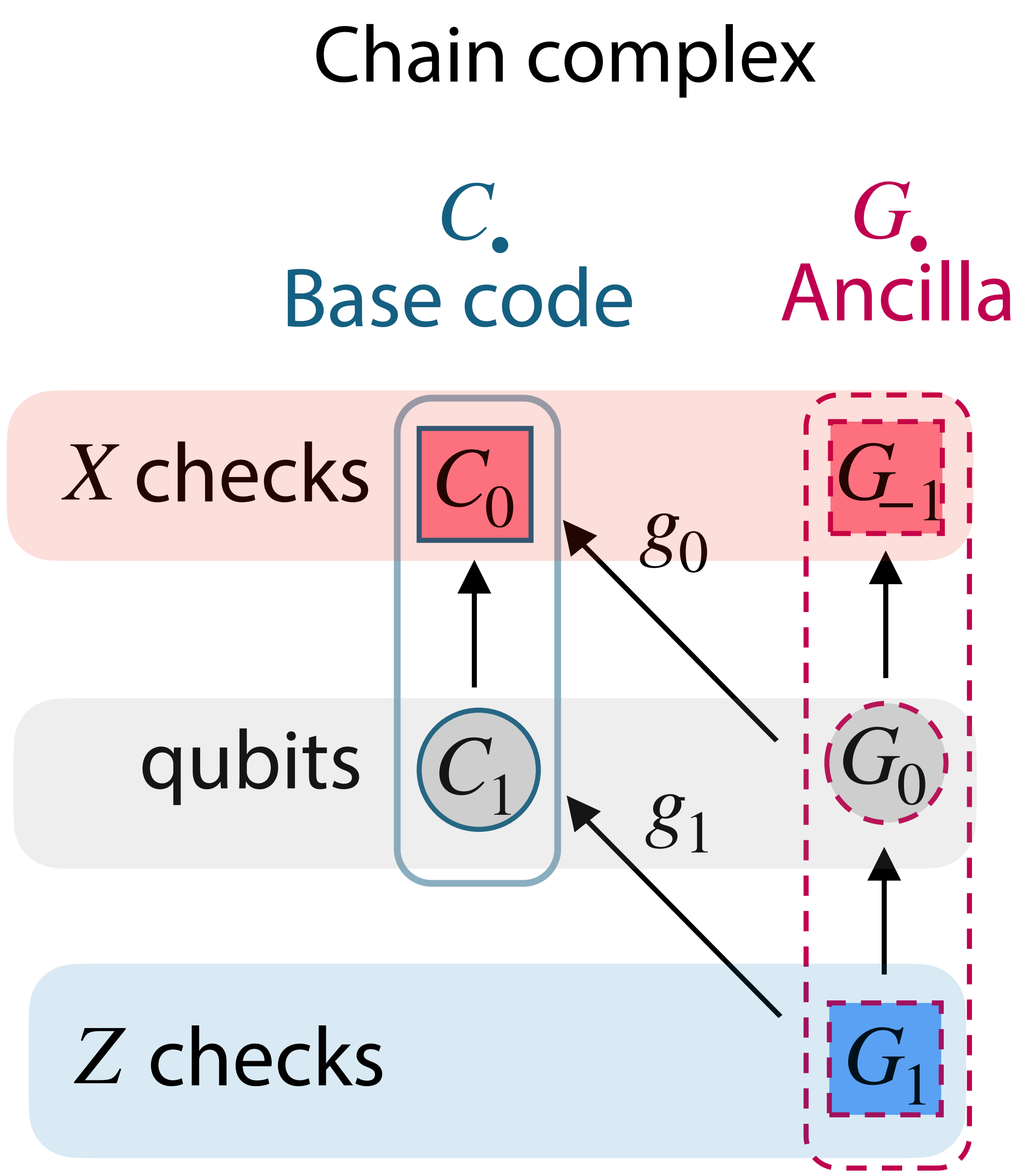
Ex. Constructing the deformed code

1. Take the base classical code $C_\bullet$



2. Augment the classical code with known surgery techniques[1,2], measure the classical codeword.

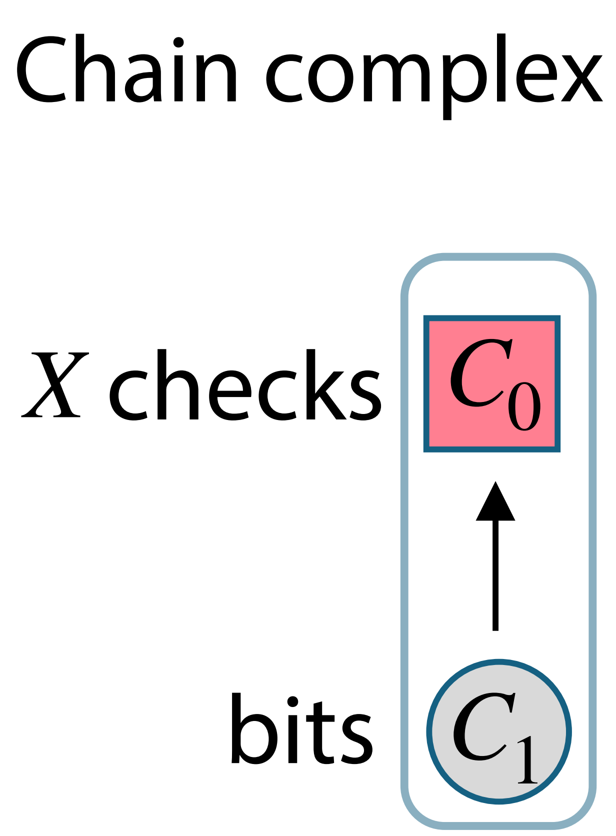
$$(g : G_\bullet \to C_\bullet)$$

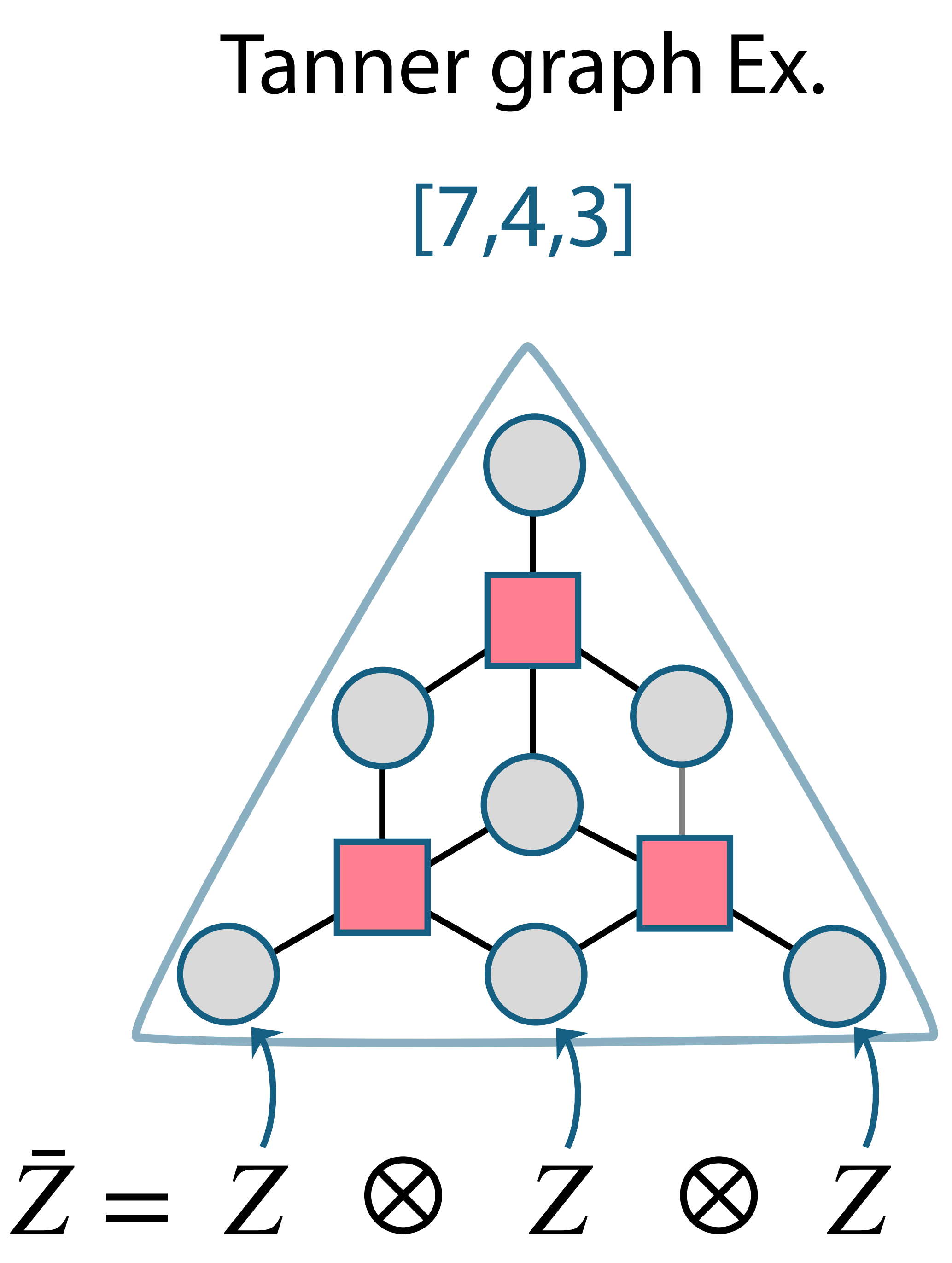


[1] Williamson, Yoder. [2410.02213] [2] Ide *et al.* PRX **15** (2025).

Ex. Constructing the deformed code

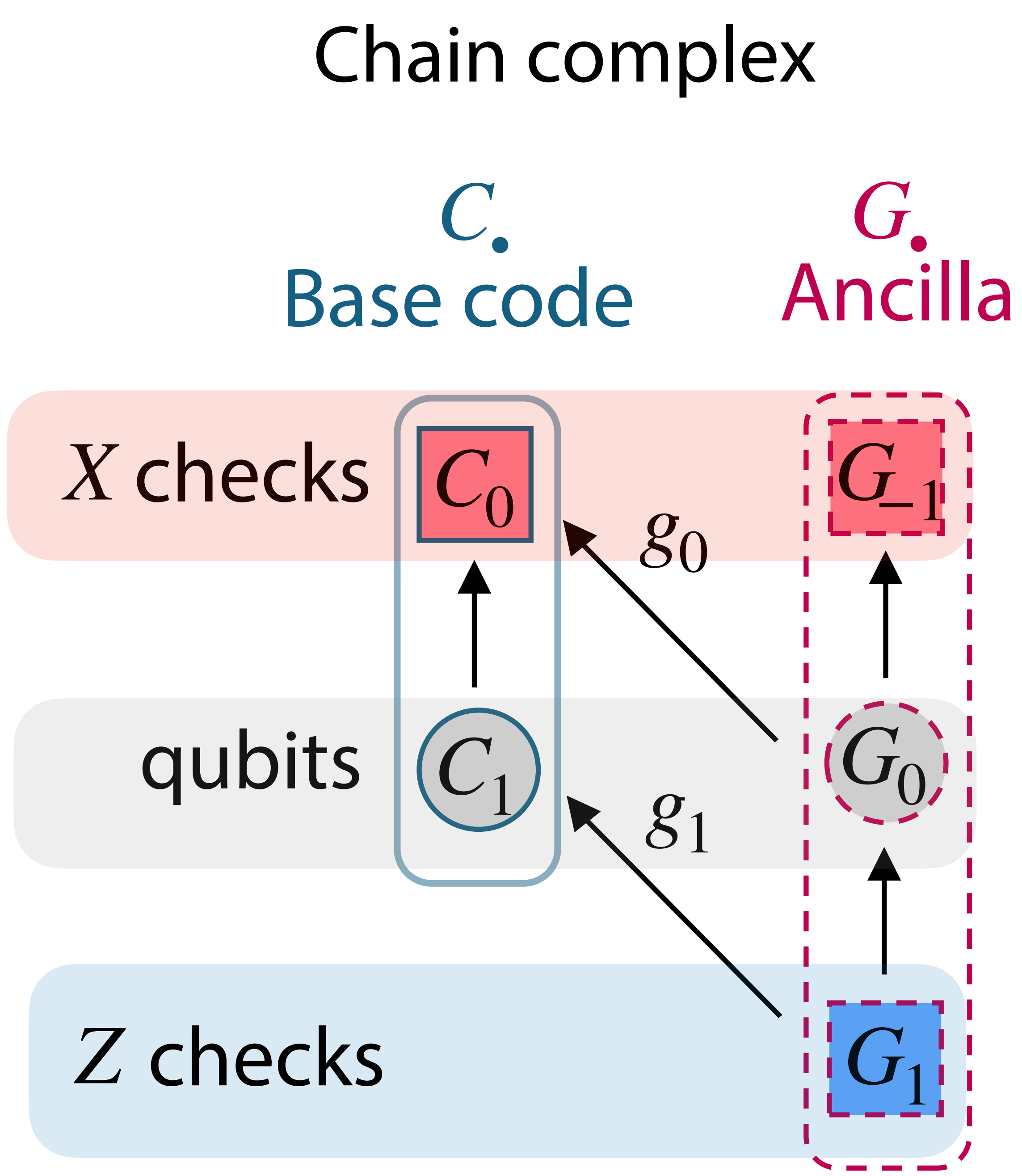
1. Take the base classical code $C_\bullet$

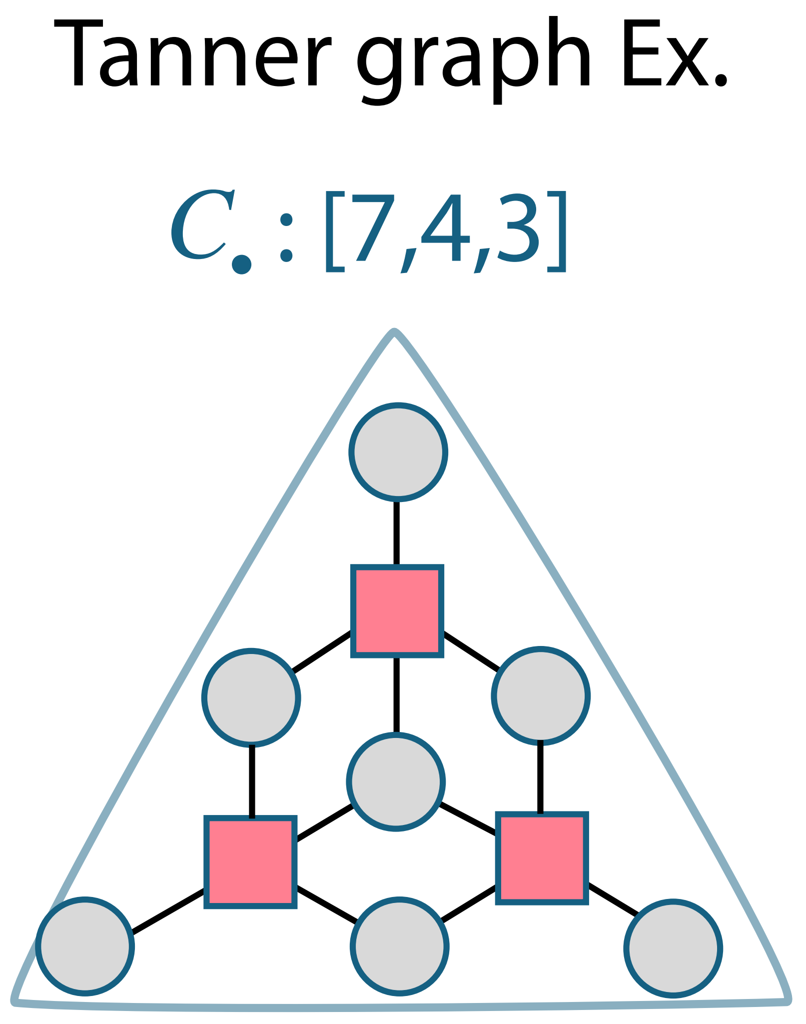




2. Augment the classical code with known surgery techniques[1,2], measure the classical codeword.

$$(g : G_\bullet \to C_\bullet)$$



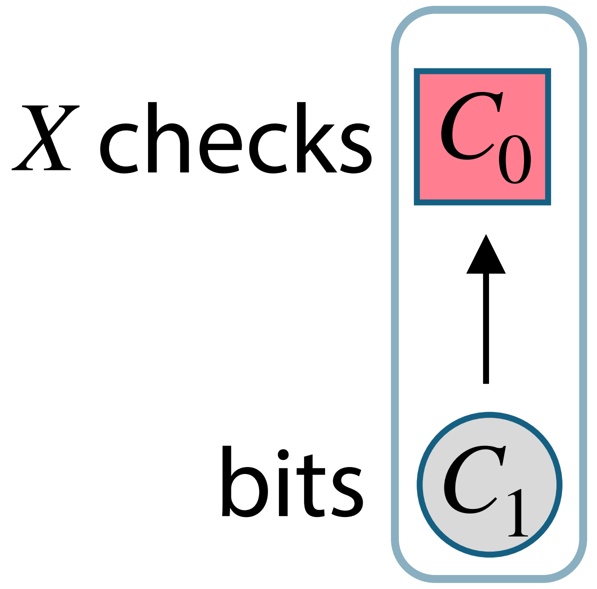


[1] Williamson, Yoder. [2410.02213]
[2] Ide *et al.* PRX **15** (2025).

Ex. Constructing the deformed code

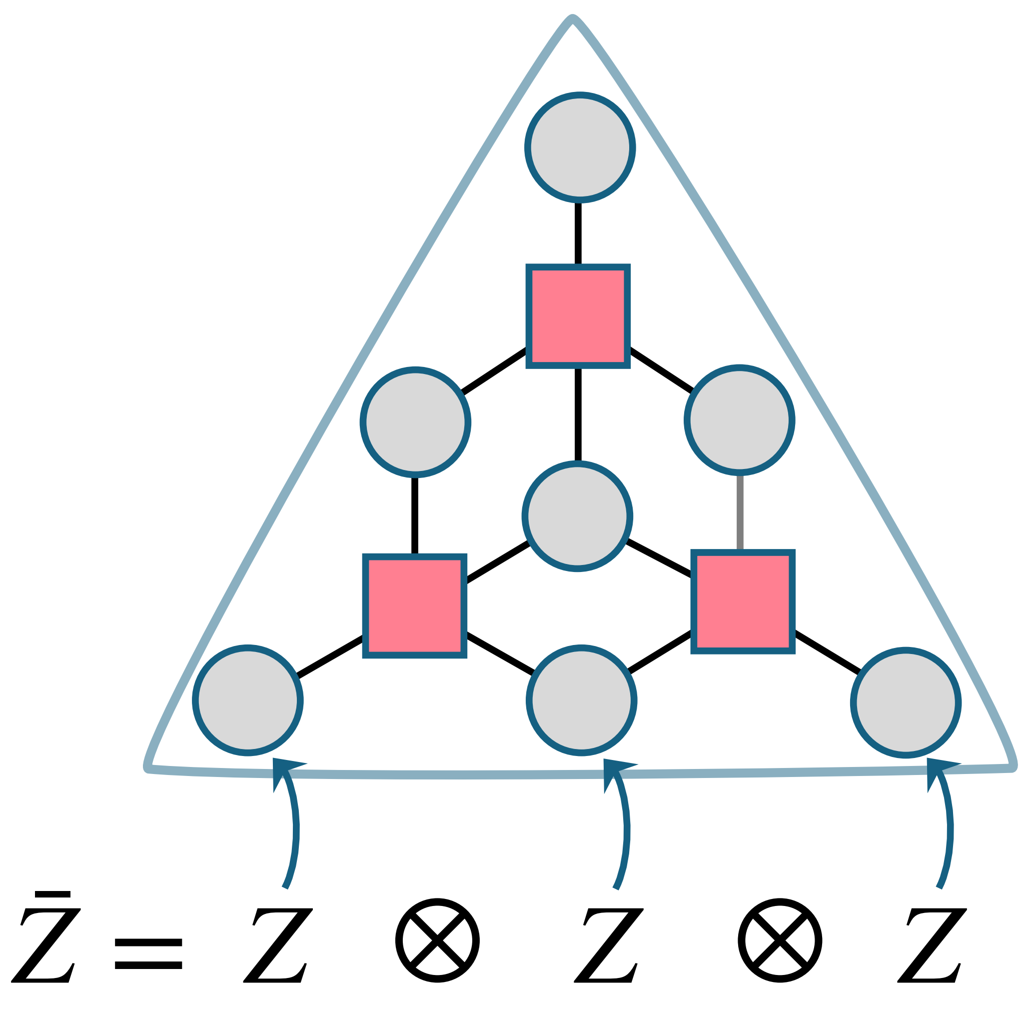
1. Take the base classical code $C_\bullet$

Chain complex



Tanner graph Ex.

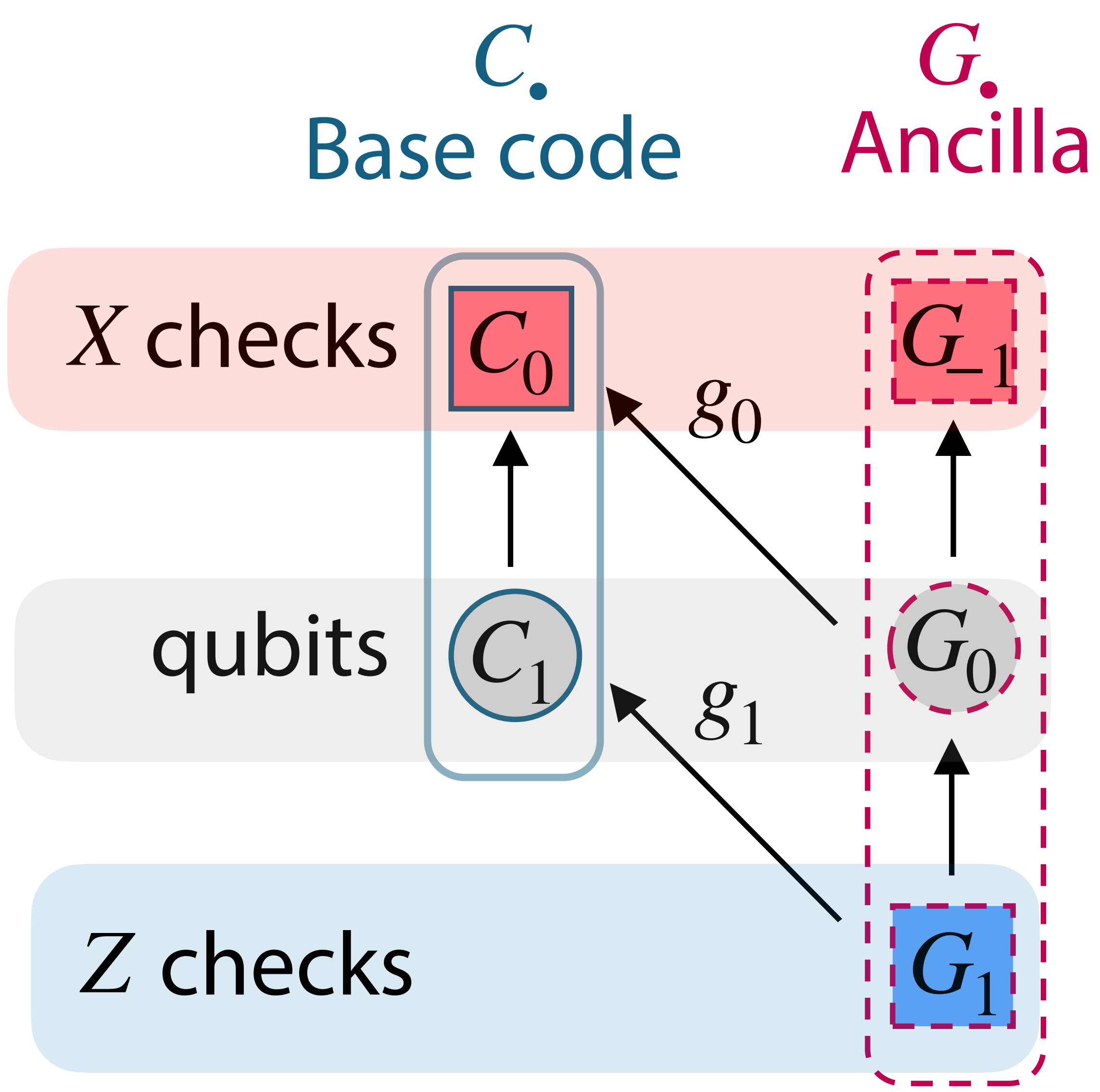
[7,4,3]



2. Augment the classical code with known surgery techniques[1,2], measure the classical codeword.

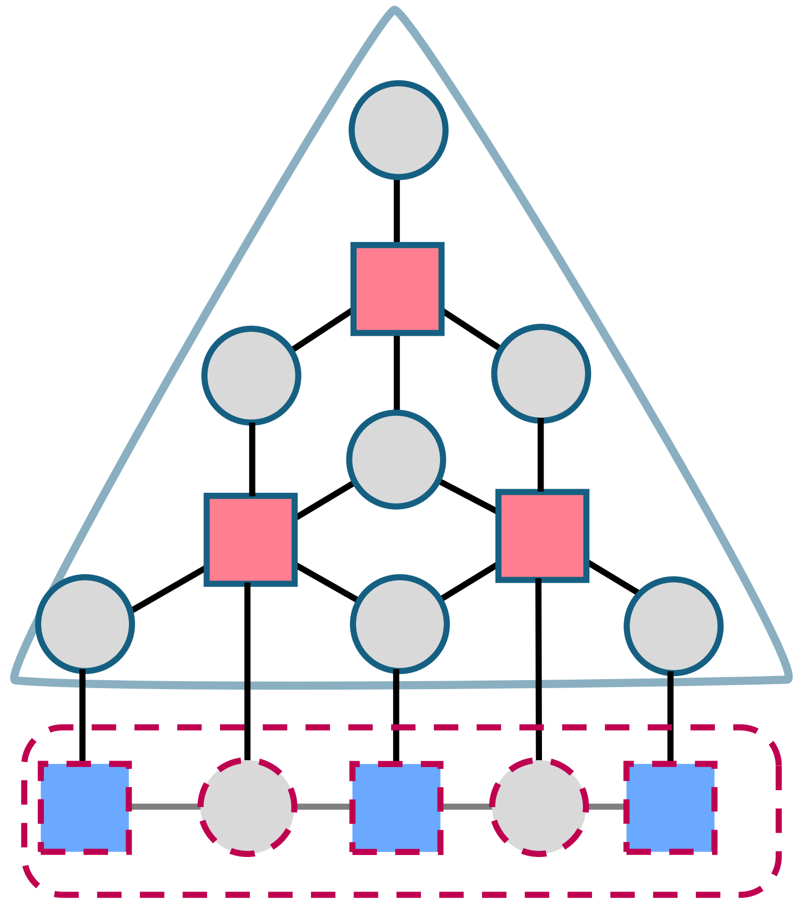
$$(g : G_\bullet \to C_\bullet)$$

Chain complex



Tanner graph Ex.

$C_\bullet$: [7,4,3]

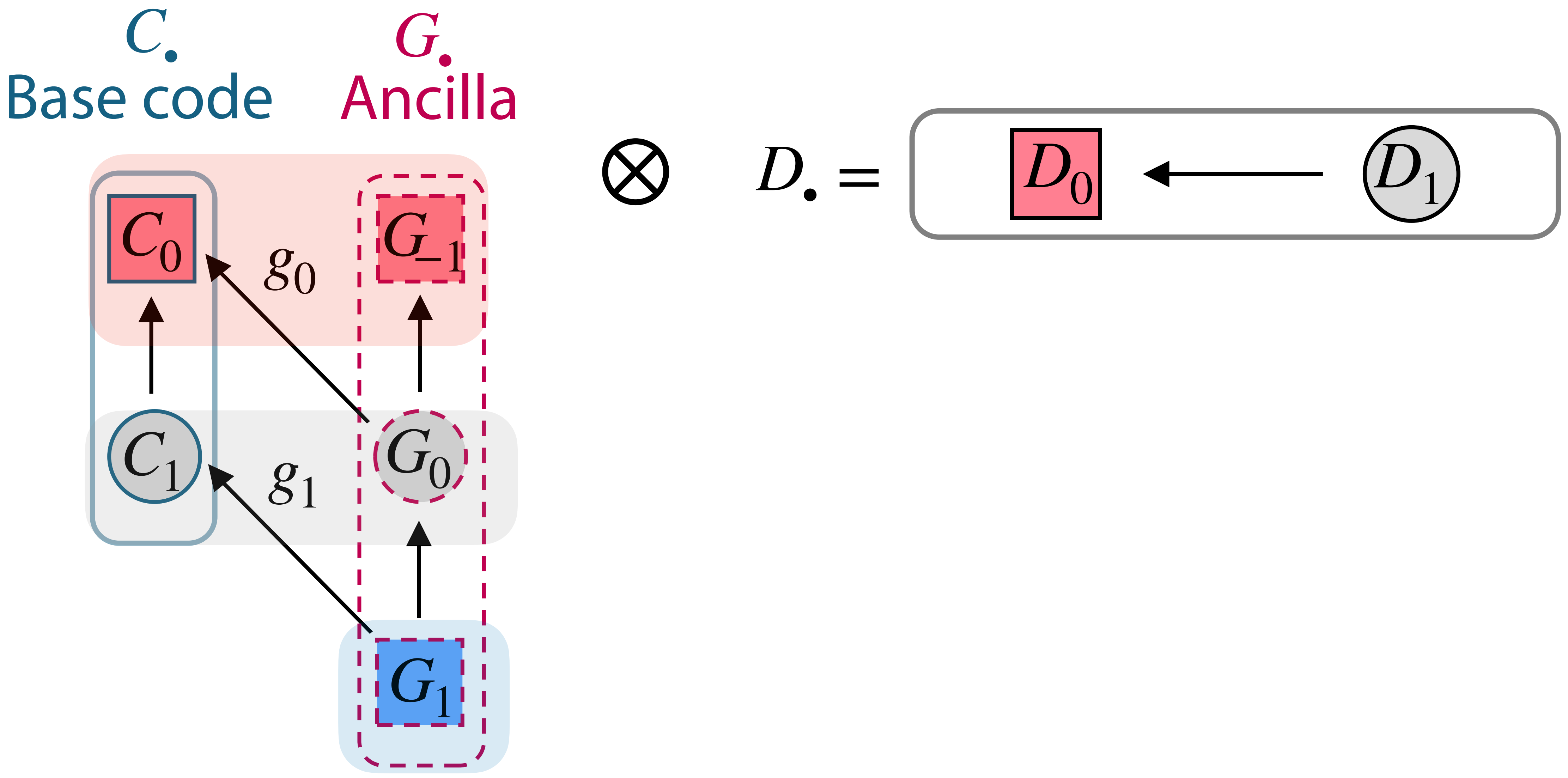


$G_\bullet$: $[3,1,3]^\top$

[1] Williamson, Yoder. [2410.02213] [2] Ide *et al.* PRX **15** (2025).

Ex. Constructing the deformed code

3. Take the hypergraph product with the other base classical code, $D_\bullet$.

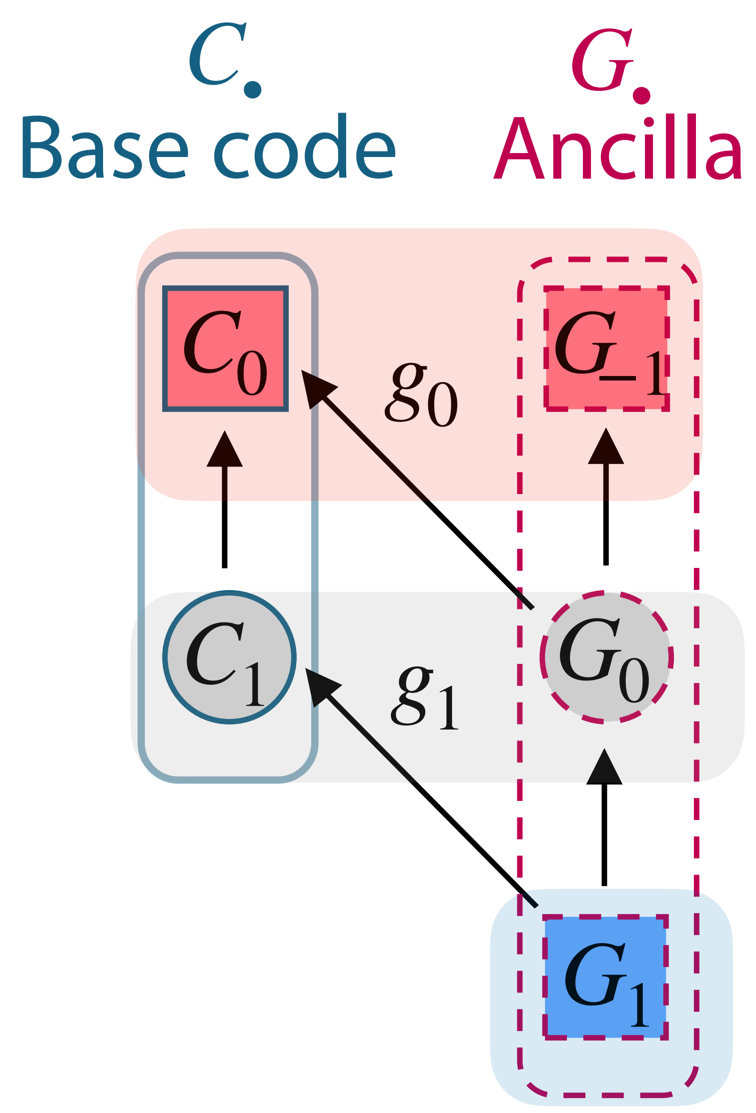


Ex. Constructing the deformed code

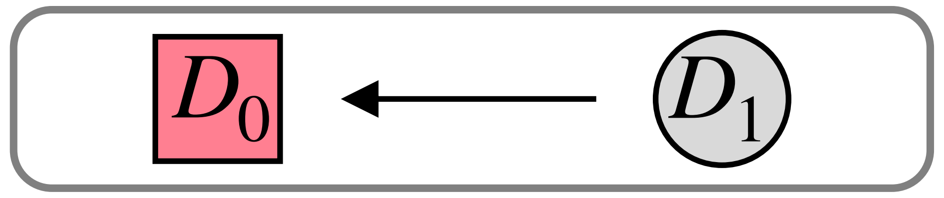
3. Take the hypergraph product with the other base classical code, $D_\bullet$.

Let $G_i^j = G_i \otimes D_j$

$C_i^j = C_i \otimes D_j$



$\otimes$ $D_\bullet =$



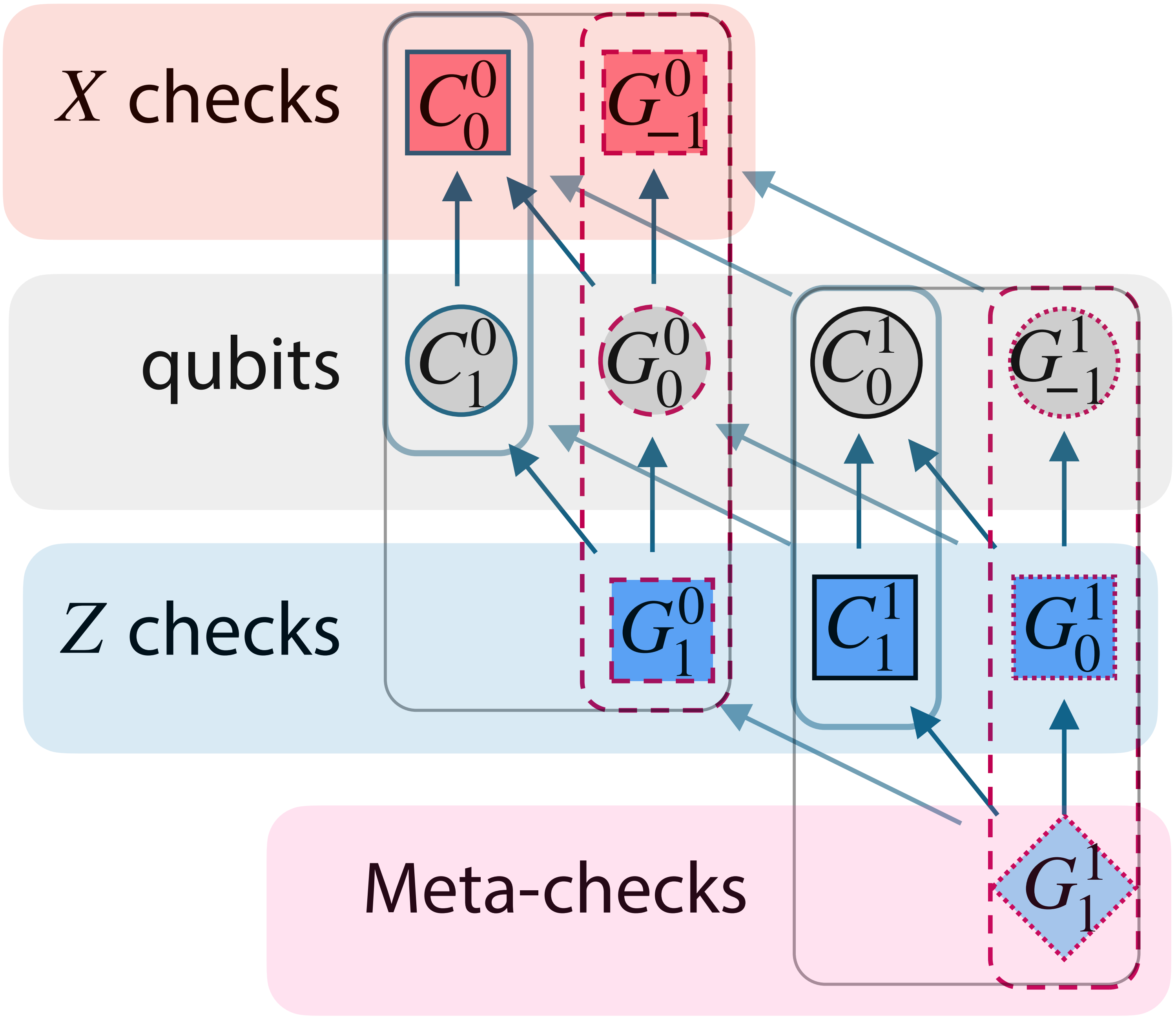
Ex. Constructing the deformed code

3. Take the hypergraph product with the other base classical code, $D_\bullet$.

Let $G_i^j = G_i \otimes D_j$

$C_i^j = C_i \otimes D_j$

Chain complex



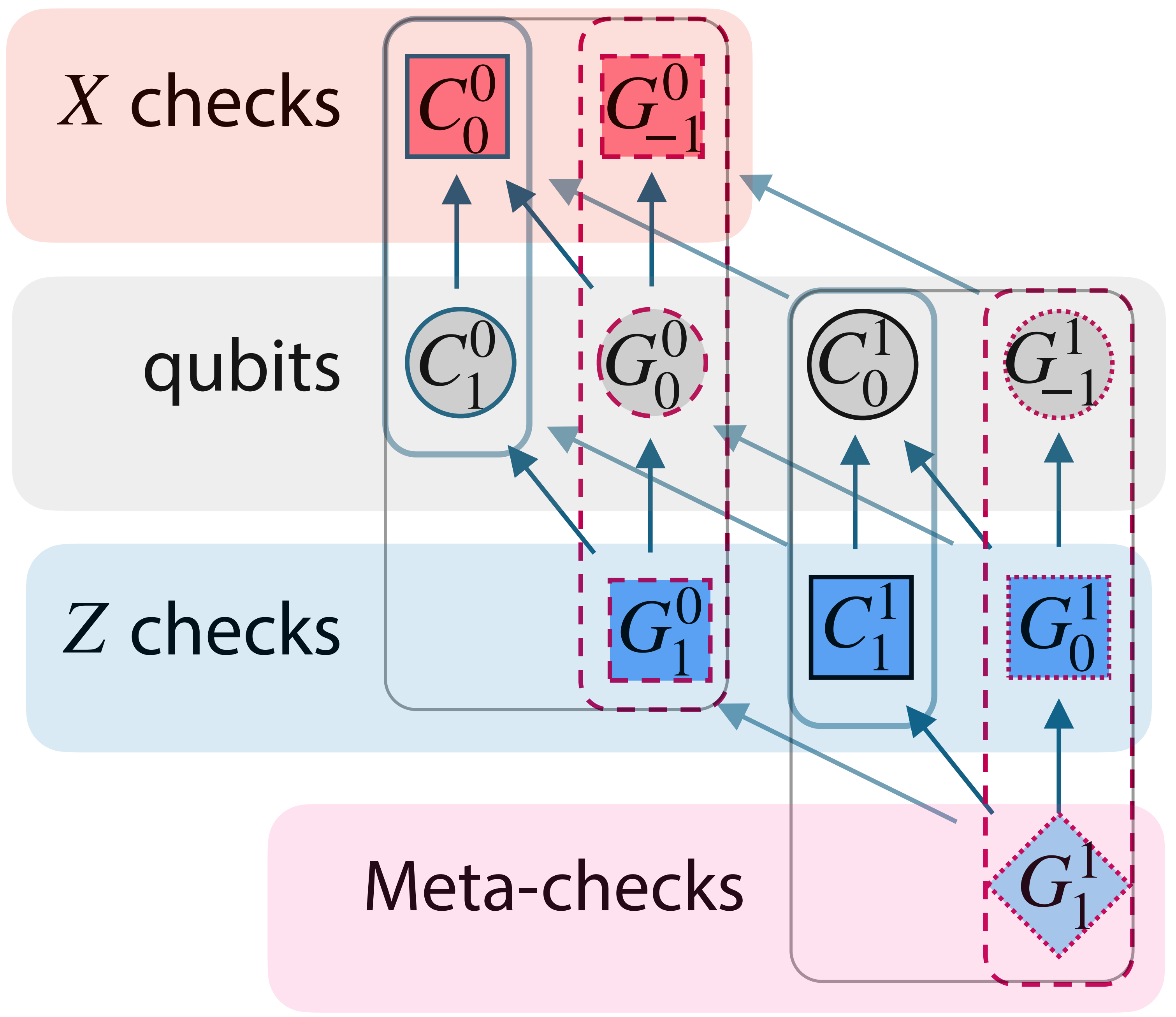
Ex. Constructing the deformed code

3. Take the hypergraph product with the other base classical code, $D_\bullet$.

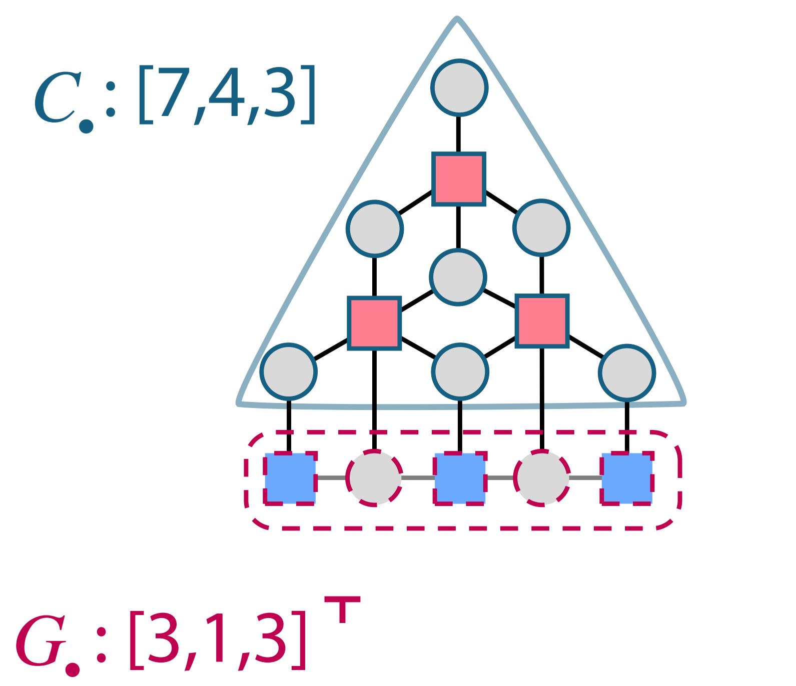
Let $G_i^j = G_i \otimes D_j$

$C_i^j = C_i \otimes D_j$

Chain complex



Tanner graph Ex.



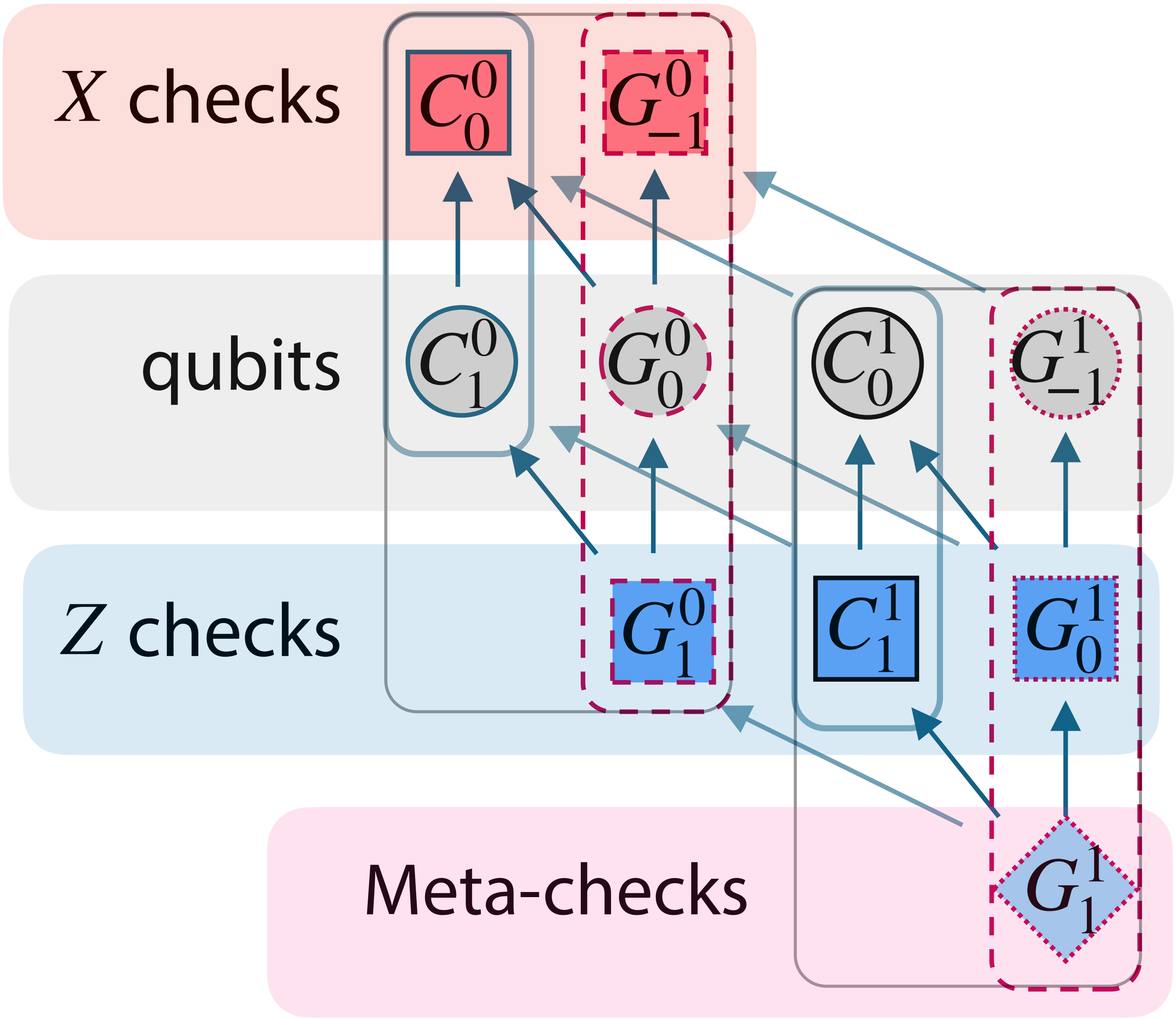
Ex. Constructing the deformed code

3. Take the hypergraph product with the other base classical code, $D_\bullet$.

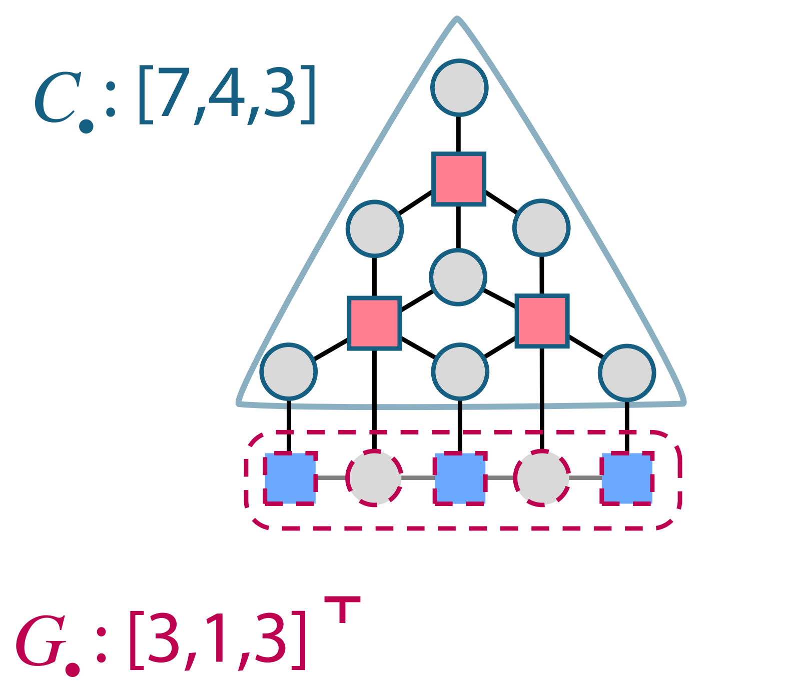
Let $G_i^j = G_i \otimes D_j$

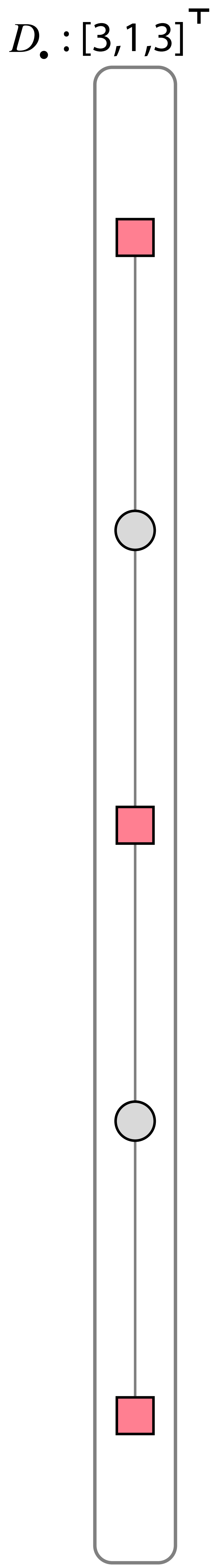
$C_i^j = C_i \otimes D_j$

Chain complex



Tanner graph Ex.

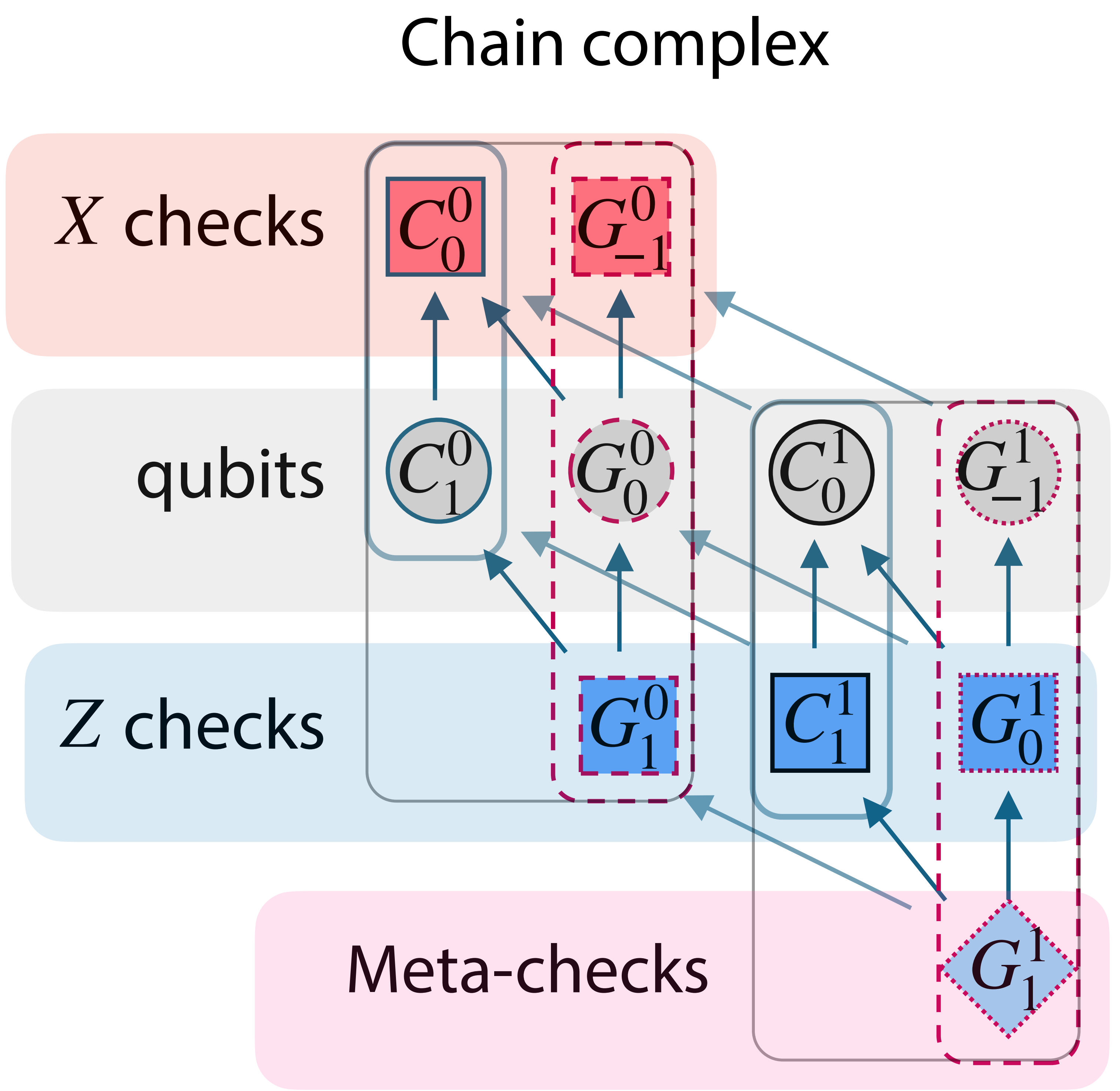


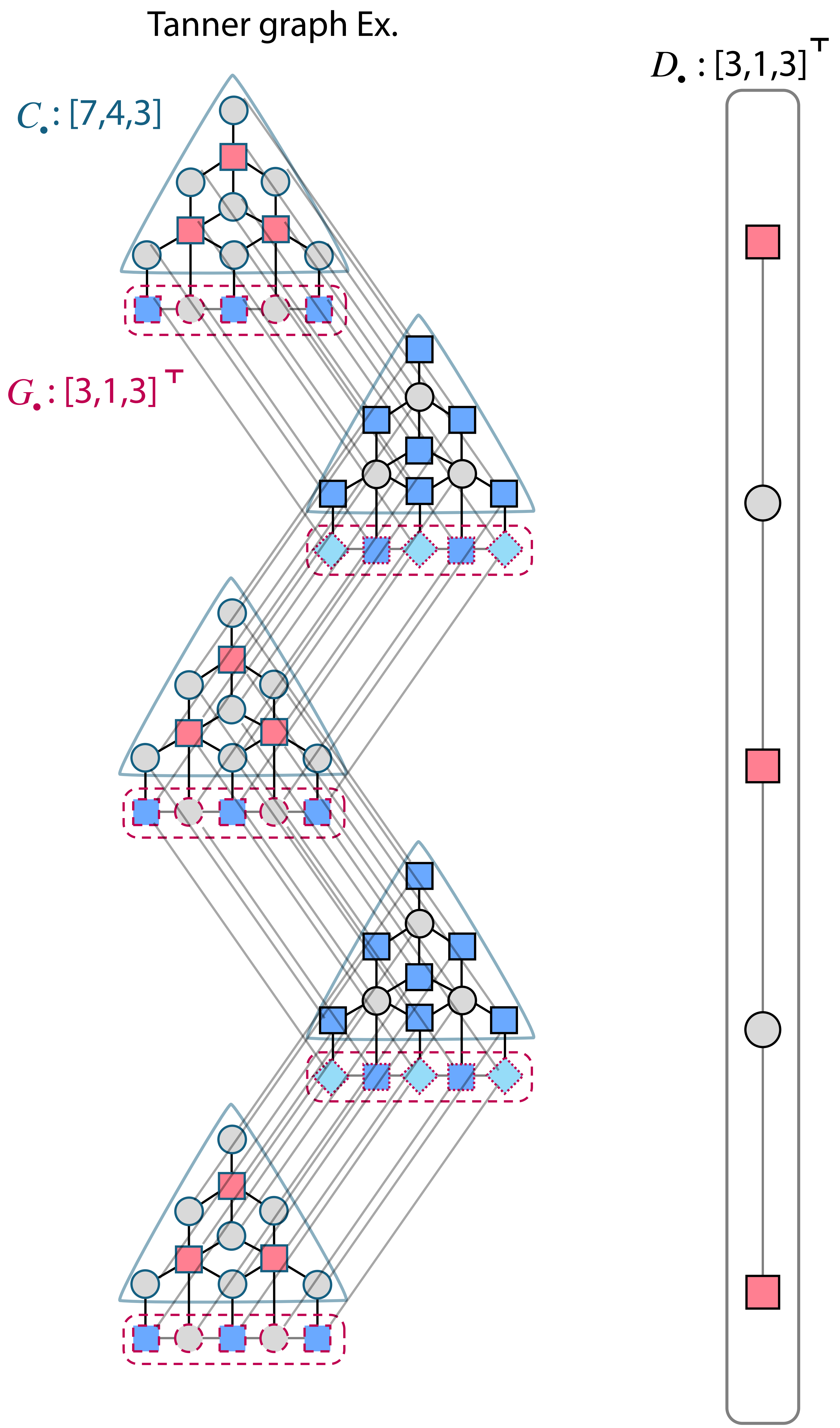


Ex. Constructing the deformed code

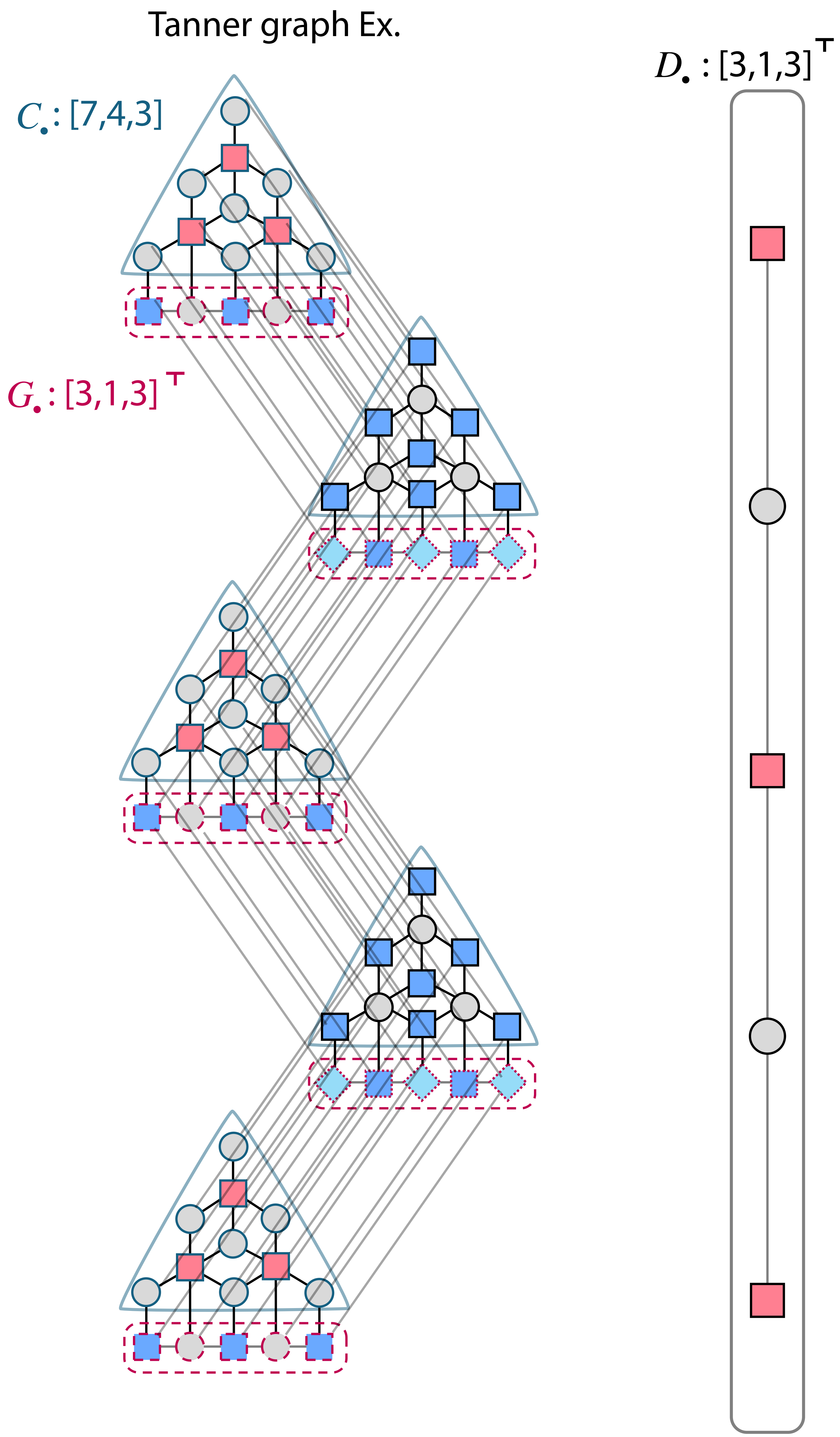
3. Take the hypergraph product with the other base classical code, $D_\bullet$.

Let $G_i^j = G_i \otimes D_j$
$C_i^j = C_i \otimes D_j$



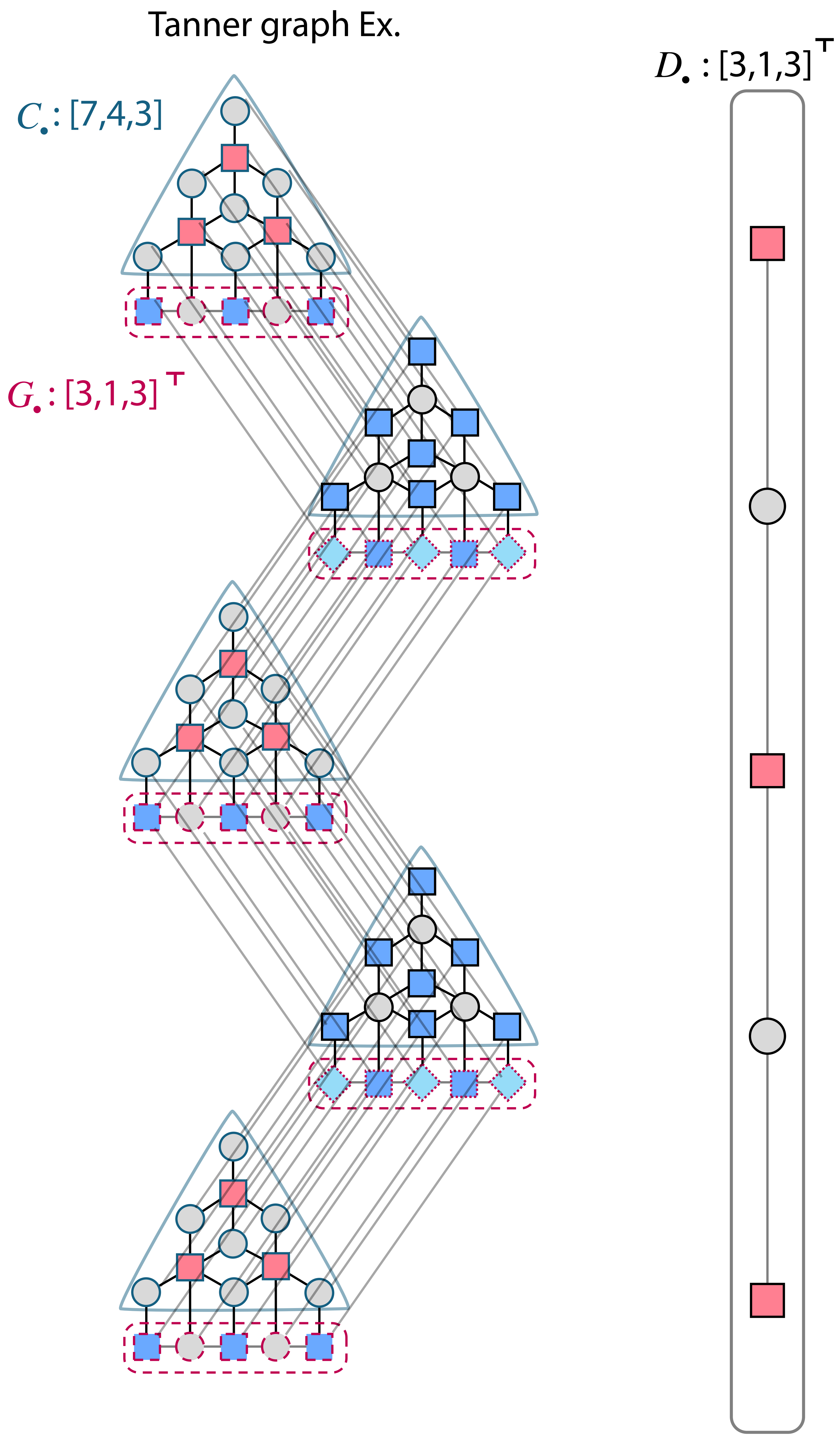


Ex. Constructing the deformed code



Ex. Constructing the deformed code

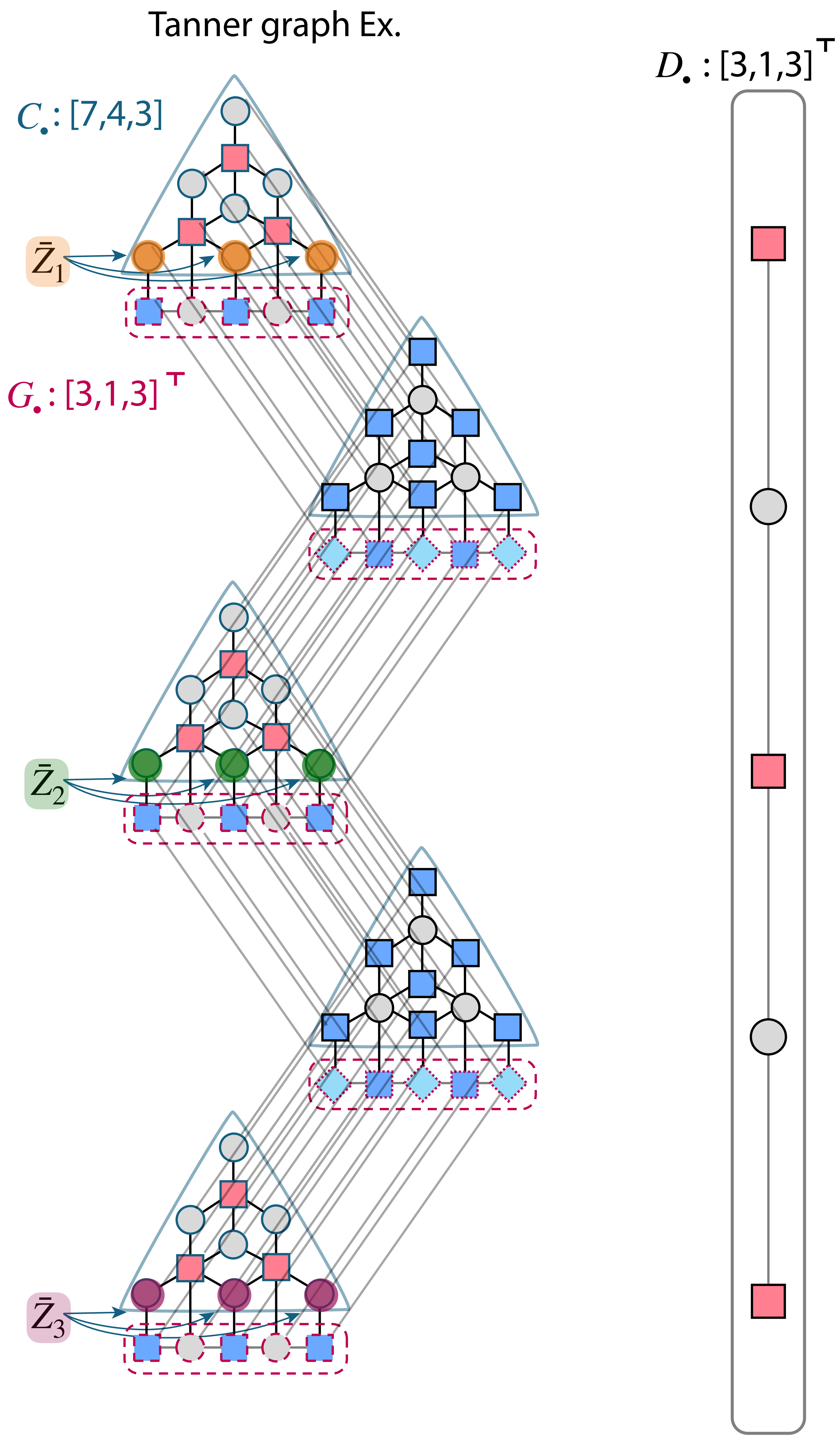
What does measuring the stabilizers of this deformed code accomplish?



Ex. Constructing the deformed code

What does measuring the stabilizers of this deformed code accomplish?

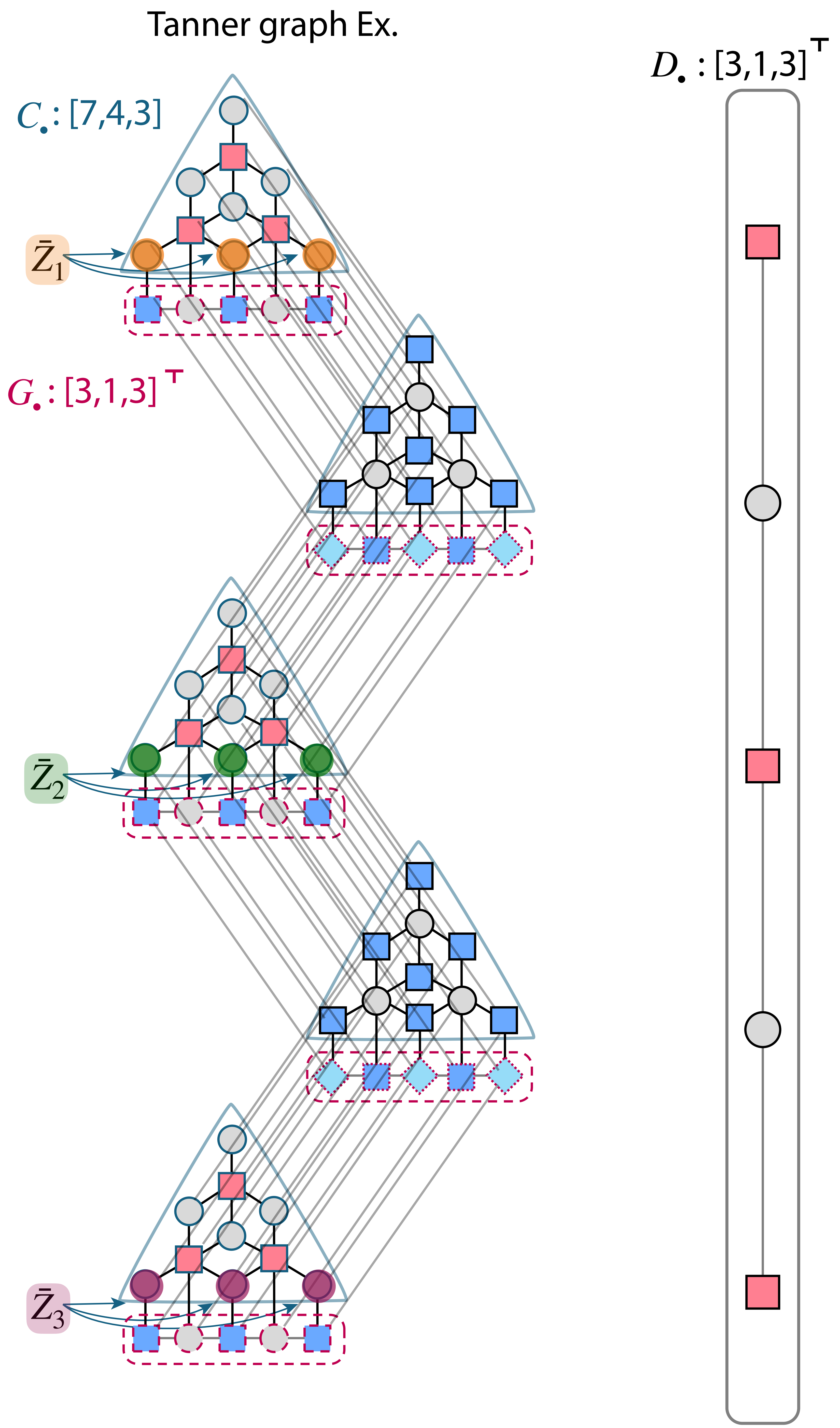
- $\bar{Z}_1$, $\bar{Z}_2$, $\bar{Z}_3$ are logically equivalent representatives of the same logical operator.



Ex. Constructing the deformed code

What does measuring the stabilizers of this deformed code accomplish?

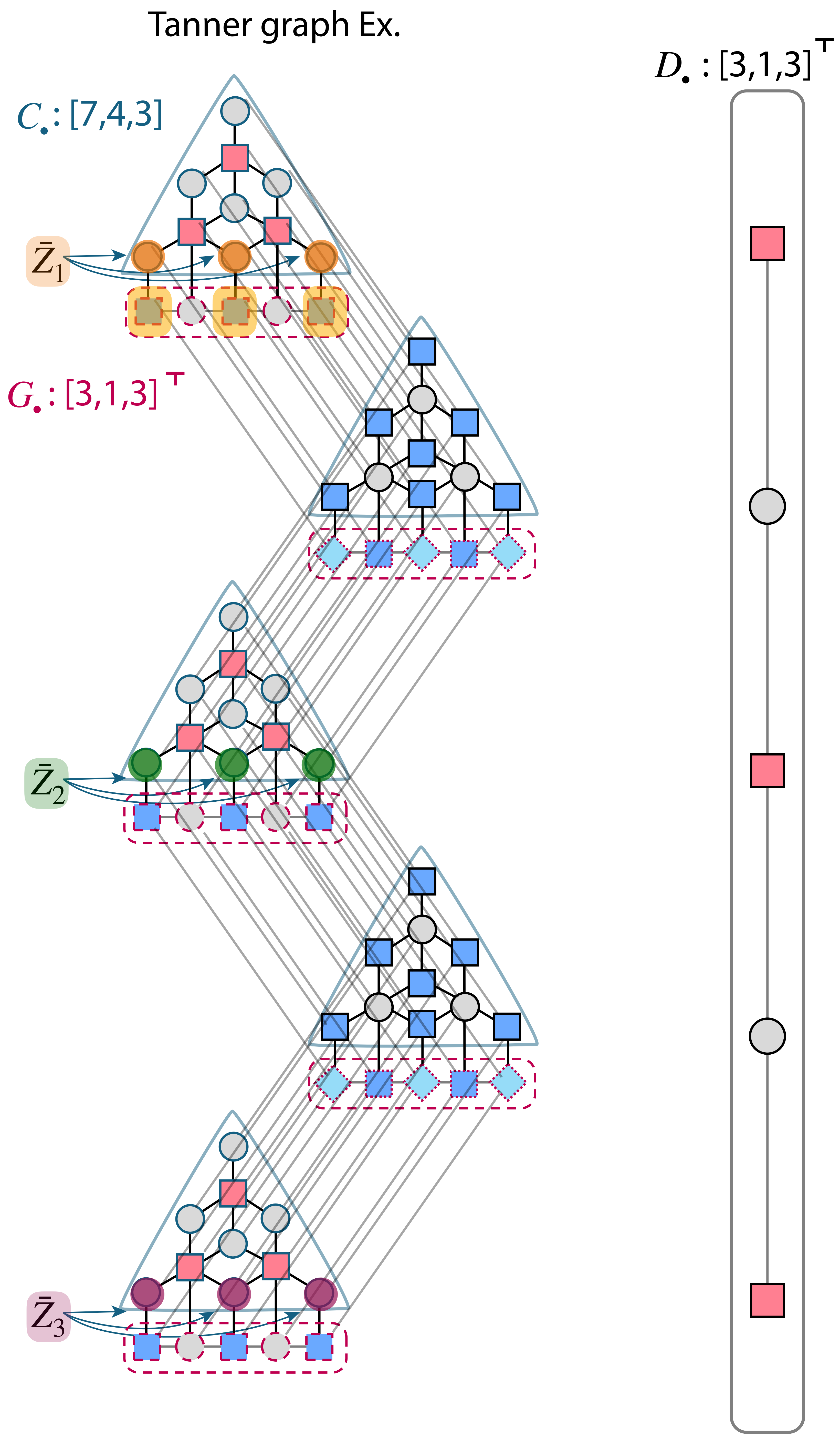
- $\bar{Z}_1$, $\bar{Z}_2$, $\bar{Z}_3$ are logically equivalent representatives of the same logical operator.
- $\bar{Z}_1$, $\bar{Z}_2$, $\bar{Z}_3$ logical representatives are measured simultaneously — each equal a product of ancilla Z checks



Ex. Constructing the deformed code

What does measuring the stabilizers of this deformed code accomplish?

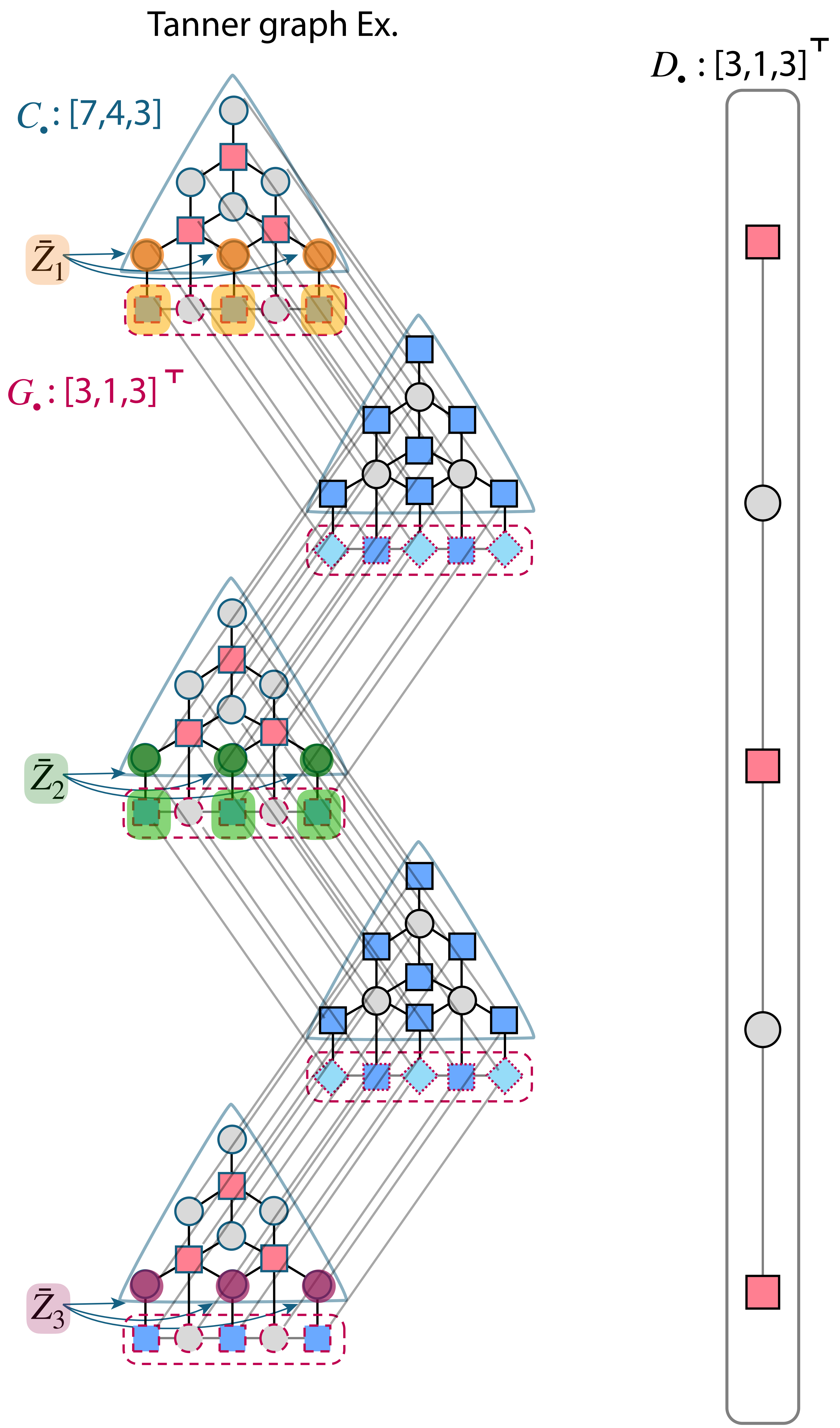
- $\bar{Z}_1$, $\bar{Z}_2$, $\bar{Z}_3$ are logically equivalent representatives of the same logical operator.
- $\bar{Z}_1$, $\bar{Z}_2$, $\bar{Z}_3$ logical representatives are measured simultaneously — each equal a product of ancilla Z checks



Ex. Constructing the deformed code

What does measuring the stabilizers of this deformed code accomplish?

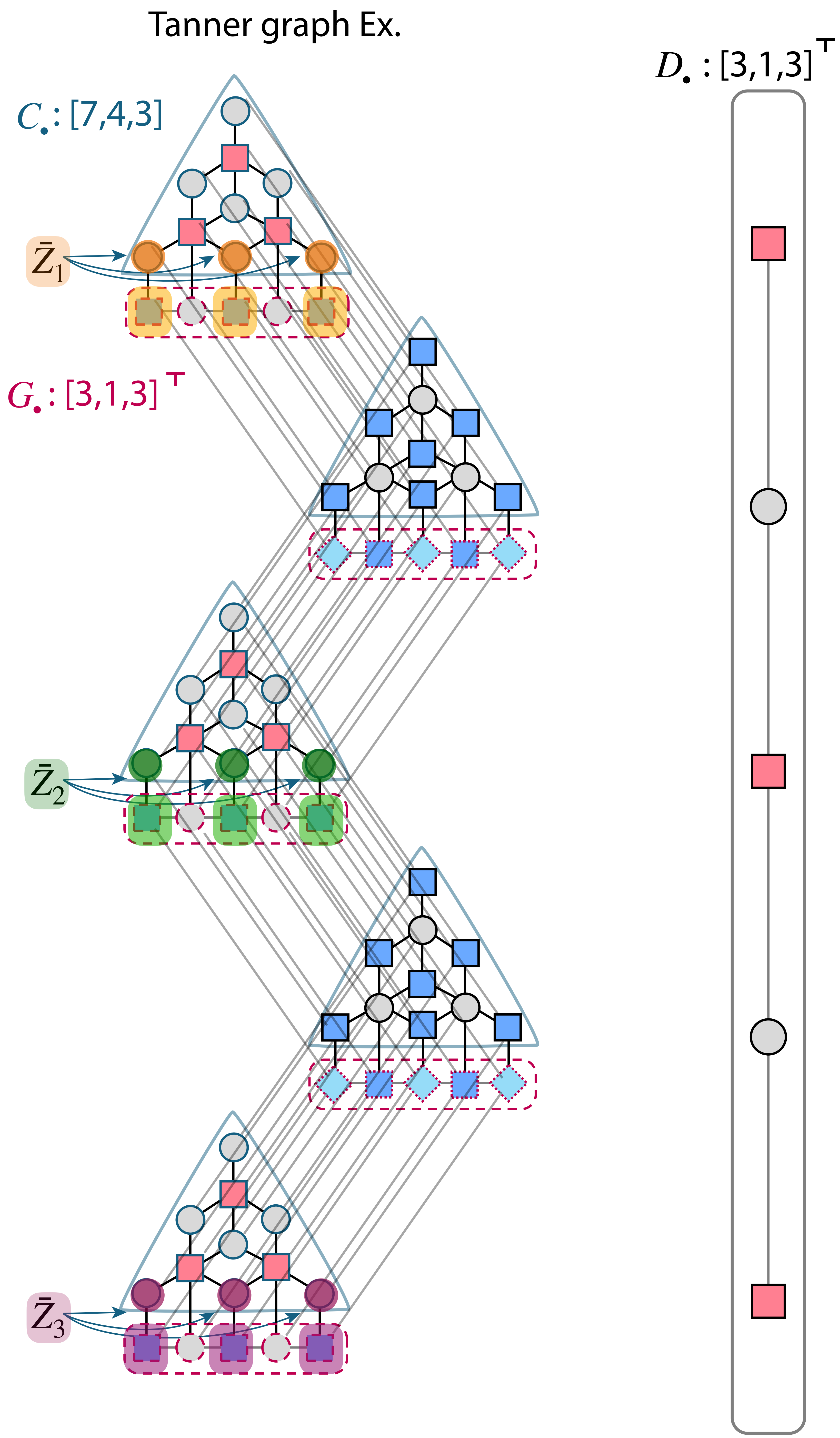
- $\bar{Z}_1$, $\bar{Z}_2$, $\bar{Z}_3$ are logically equivalent representatives of the same logical operator.
- $\bar{Z}_1$, $\bar{Z}_2$, $\bar{Z}_3$ logical representatives are measured simultaneously — each equal a product of ancilla Z checks



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



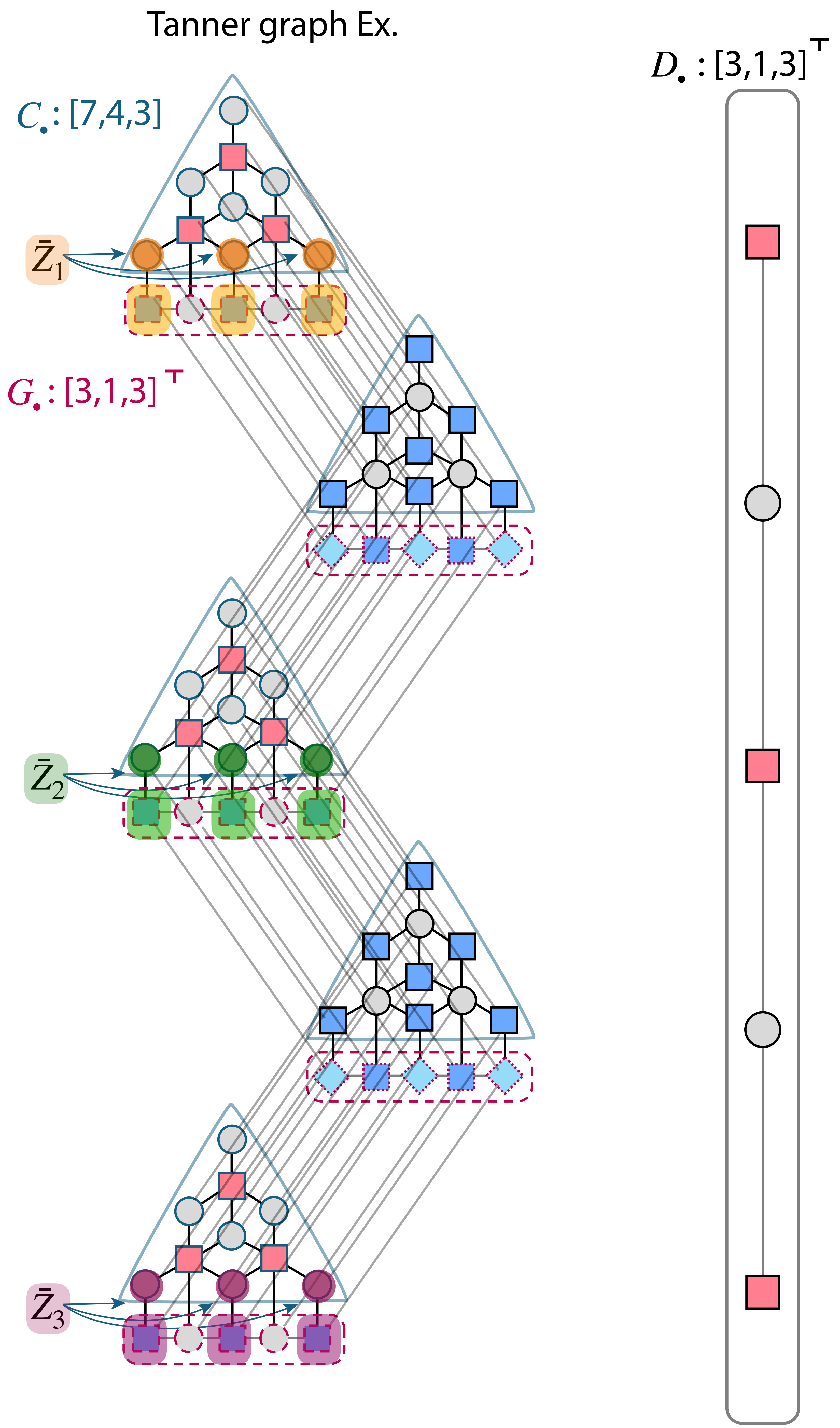
Ex. Constructing the deformed code

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.



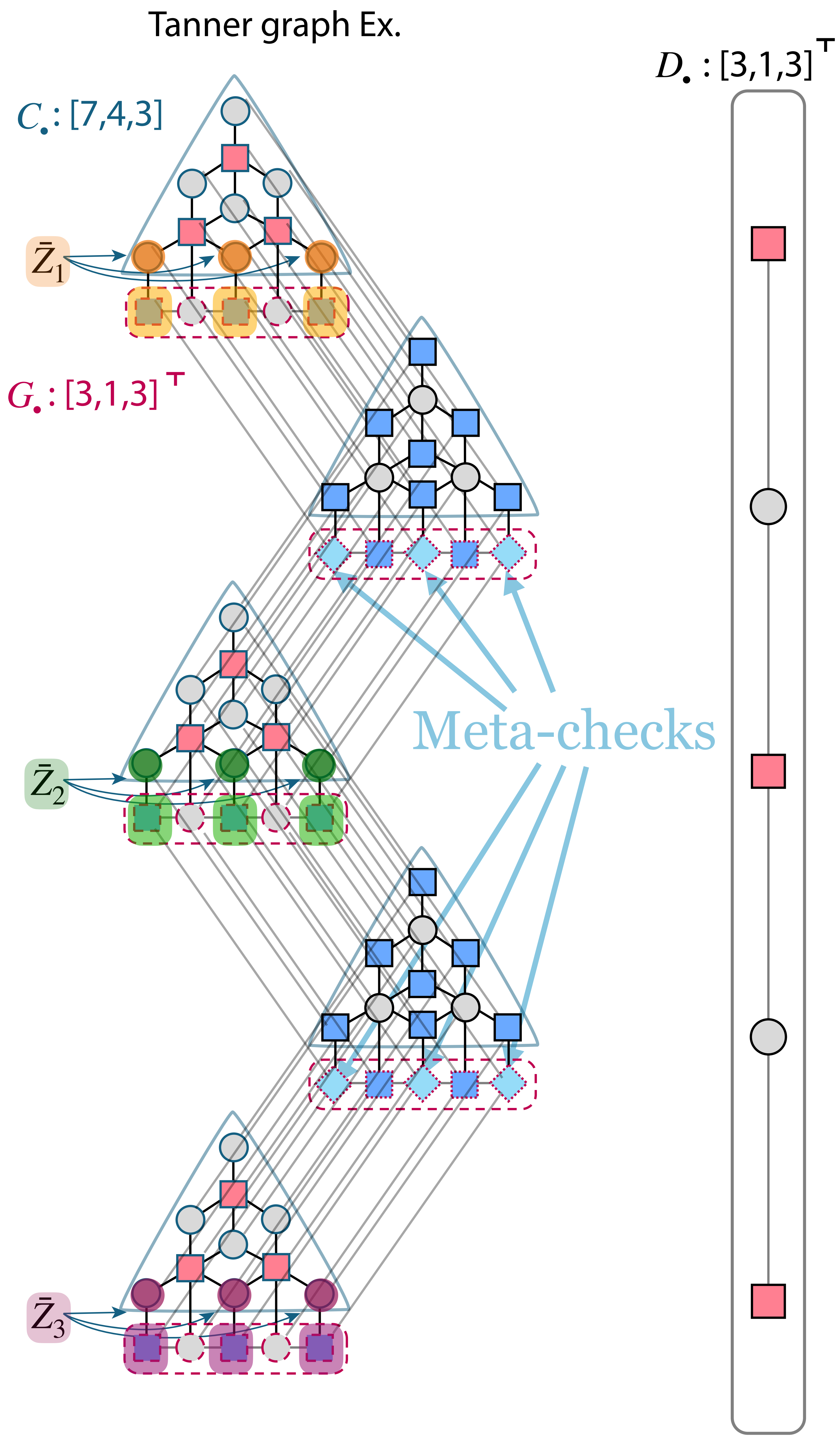
Ex. Constructing the deformed code

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.



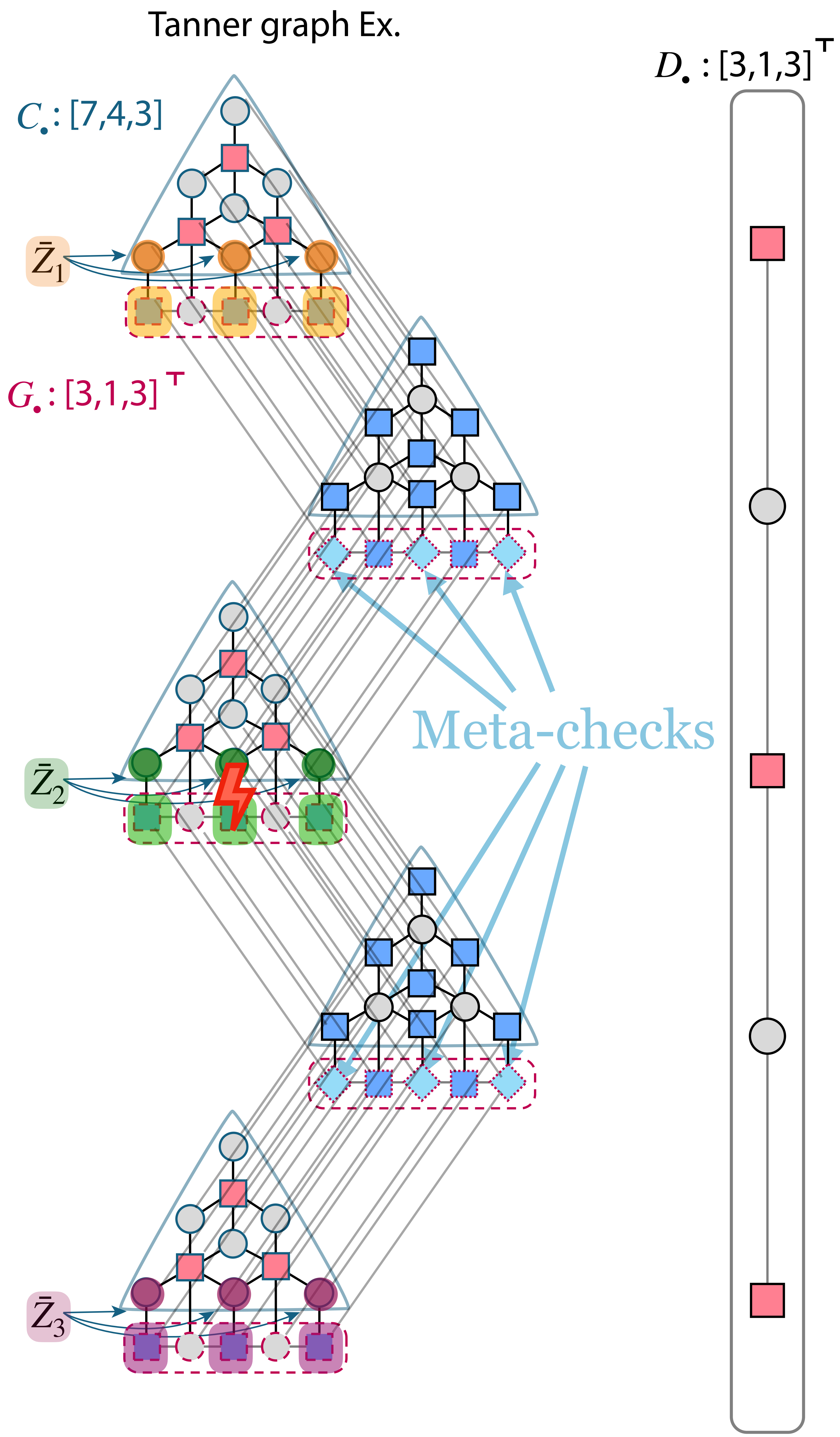
Ex. Constructing the deformed code

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.



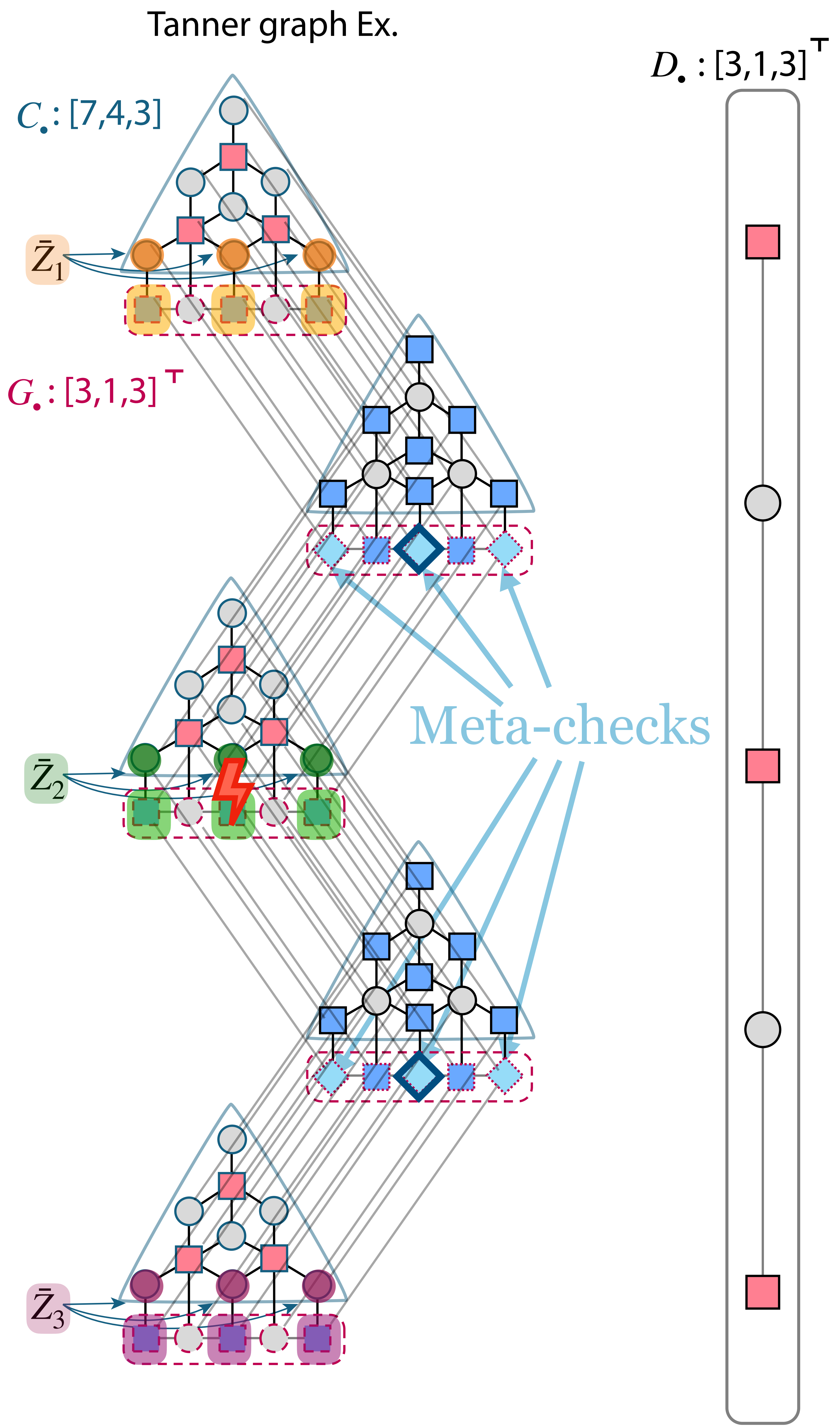
Ex. Constructing the deformed code

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.



Summary

Summary

- Formulation of constant-time surgery and fault-tolerance conditions

Summary

- Formulation of constant-time surgery and fault-tolerance conditions
- Given an $[[n, k, d]]$ hypergraph product code, we can construct a sequence of ***LDPC*** ancillary complexes that:

Summary

- Formulation of constant-time surgery and fault-tolerance conditions

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

☑ Satisfy ***fault-tolerance conditions*** for constant-time surgery

Summary

- Formulation of constant-time surgery and fault-tolerance conditions
- Given an $[[n, k, d]]$ hypergraph product code, we can construct a sequence of ***LDPC*** ancillary complexes that:

☑ Satisfy ***fault-tolerance conditions*** for constant-time surgery

☑ **Some addressibility:** Parallelized measurement of **rows (and columns) of Z (X)** logical operators.

Summary

- Formulation of constant-time surgery and fault-tolerance conditions
- Given an $[[n, k, d]]$ hypergraph product code, we can construct a sequence of ***LDPC*** ancillary complexes that:

☑ Satisfy ***fault-tolerance conditions*** for constant-time surgery

☑ **Some addressibility:** Parallelized measurement of **rows (and columns) of Z (X)** logical operators.

☑ **Constant time:** Each surgery operation takes $\mathcal{O}(1)$ **time** in amortization.

Summary

- Formulation of constant-time surgery and fault-tolerance conditions

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

☑ Satisfy ***fault-tolerance conditions*** for constant-time surgery

☑ **Some addressibility:** Parallelized measurement of **rows (and columns) of Z (X)** logical operators.

☑ **Constant time:** Each surgery operation takes $\mathcal{O}(1)$ **time** in amortization.

☐ **Low space overhead:** Each complex **uses** $\mathcal{O}(n \log(n))$ **physical ancilla qubits**.

Summary

- Formulation of constant-time surgery and fault-tolerance conditions
- Given an $[[n, k, d]]$ hypergraph product code, we can construct a sequence of ***LDPC*** ancillary complexes that:

☑ Satisfy ***fault-tolerance conditions*** for constant-time surgery

☑ **Some addressibility:** Parallelized measurement of **rows (and columns) of Z (X)** logical operators.

☑ **Constant time:** Each surgery operation takes $\mathcal{O}(1)$ **time** in amortization.

☐ **Low space overhead:** Each complex **uses** $\mathcal{O}(n \log(n))$ **physical ancilla qubits**.

Num. qubits in $(G \otimes D)_\bullet$**?**

Summary

- Formulation of constant-time surgery and fault-tolerance conditions

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

☑ Satisfy ***fault-tolerance conditions*** for constant-time surgery

☑ **Some addressibility:** Parallelized measurement of **rows (and columns) of Z (X)** logical operators.

☑ **Constant time:** Each surgery operation takes $\mathcal{O}(1)$ **time** in amortization.

☐ **Low space overhead:** Each complex **uses** $\mathcal{O}(n\log(n))$ **physical ancilla qubits**.

Num. qubits in $(G \otimes D)_\bullet$**?**

1. A classical codeword of $C_\bullet$ and each vector space of $D_\bullet$ have size at most $O(\sqrt{n})$.

Summary

- Formulation of constant-time surgery and fault-tolerance conditions

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

☑ Satisfy ***fault-tolerance conditions*** for constant-time surgery

☑ **Some addressibility:** Parallelized measurement of **rows (and columns) of Z (X)** logical operators.

☑ **Constant time:** Each surgery operation takes $\mathcal{O}(1)$ **time** in amortization.

☐ **Low space overhead:** Each complex **uses** $\mathcal{O}(n\log(n))$ **physical ancilla qubits**.

Num. qubits in $(G \otimes D)_\bullet$**?**

1. A classical codeword of $C_\bullet$ and each vector space of $D_\bullet$ have size at most $O(\sqrt{n})$.
2. Surgery of the classical codeword uses at most $\tilde{O}(\sqrt{n})$[1] qubits.

Summary

- Formulation of constant-time surgery and fault-tolerance conditions

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

☑ Satisfy ***fault-tolerance conditions*** for constant-time surgery

☑ **Some addressibility:** Parallelized measurement of **rows (and columns) of Z (X)** logical operators.

☑ **Constant time:** Each surgery operation takes $\mathcal{O}(1)$ **time** in amortization.

☑ **Low space overhead:** Each complex **uses** $\mathcal{O}(n\log(n))$ **physical ancilla qubits**.

Num. qubits in $(G \otimes D)_\bullet$**?**

1. A classical codeword of $C_\bullet$ and each vector space of $D_\bullet$ have size at most $O(\sqrt{n})$.
2. Surgery of the classical codeword uses at most $\tilde{O}(\sqrt{n})$[1] qubits.
3. Taking the tensor product with D yields a space overhead $\tilde{O}(\sqrt{n} \circ \sqrt{n})$

Summary

- Formulation of constant-time surgery and fault-tolerance conditions

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

☑ Satisfy ***fault-tolerance conditions*** for constant-time surgery

☑ **Some addressibility:** Parallelized measurement of **rows (and columns) of Z (X)** logical operators.

☑ **Constant time:** Each surgery operation takes $\mathcal{O}(1)$ **time** in amortization.

☑ **Low space overhead:** Each complex **uses** $\mathcal{O}(n\log(n))$ **physical ancilla qubits**.

Discussion and Open directions

[1] Swaroop *et al.* [2410.03628] [2] Xu *et al.* [2510.06159]

Discussion and Open directions

- **Entangling measurements** across **two or more hypergraph product codes** that share at least one classical code, $(C \otimes D)_\bullet$, $(C' \otimes D)_\bullet$, $(C'' \otimes D)_\bullet$, $\ldots$ using adapters[1]

[1] Swaroop *et al.* [2410.03628] [2] Xu *et al.* [2510.06159]

Discussion and Open directions

- **Entangling measurements** across **two or more hypergraph product codes** that share at least one classical code, $(C \otimes D)_\bullet$, $(C' \otimes D)_\bullet$, $(C'' \otimes D)_\bullet$, $\ldots$ using adapters[1]

- Can use extractors on classical code to build **fast, parallel, and partially addressable extractors**

[1] Swaroop *et al.* [2410.03628] [2] Xu *et al.* [2510.06159]

Discussion and Open directions

- **Entangling measurements** across **two or more hypergraph product codes** that share at least one classical code, $(C \otimes D)_\bullet$, $(C' \otimes D)_\bullet$, $(C'' \otimes D)_\bullet$, $\ldots$ using adapters[1]

- Can use extractors on classical code to build **fast, parallel, and partially addressable extractors**

- **Batched logical measurements**[2] through parallel measurements on 2D hypergraph product codes

[1] Swaroop *et al.* [2410.03628] [2] Xu *et al.* [2510.06159]

Discussion and Open directions

- **Entangling measurements** across **two or more hypergraph product codes** that share at least one classical code, $(C \otimes D)_\bullet$, $(C' \otimes D)_\bullet$, $(C'' \otimes D)_\bullet$, $\ldots$ using adapters[1]

- Can use extractors on classical code to build **fast, parallel, and partially addressable extractors**

- **Batched logical measurements**[2] through parallel measurements on 2D hypergraph product codes

---

- Practical performance and decoding

---

[1] Swaroop *et al.* [2410.03628]   [2] Xu *et al.* [2510.06159]

Discussion and Open directions

- **Entangling measurements** across **two or more hypergraph product codes** that share at least one classical code, $(C \otimes D)_\bullet$, $(C' \otimes D)_\bullet$, $(C'' \otimes D)_\bullet$, $\ldots$ using adapters[1]

- Can use extractors on classical code to build **fast, parallel, and partially addressable extractors**

- **Batched logical measurements**[2] through parallel measurements on 2D hypergraph product codes

---

- Practical performance and decoding

- Addressability beyond rows or columns, compilation

[1] Swaroop *et al.* [2410.03628] [2] Xu *et al.* [2510.06159]

Discussion and Open directions

- **Entangling measurements** across **two or more hypergraph product codes** that share at least one classical code, $(C \otimes D)_\bullet$, $(C' \otimes D)_\bullet$, $(C'' \otimes D)_\bullet$, $\ldots$ using adapters[1]

- Can use extractors on classical code to build **fast, parallel, and partially addressable extractors**

- **Batched logical measurements**[2] through parallel measurements on 2D hypergraph product codes

---

- Practical performance and decoding

- Addressability beyond rows or columns, compilation

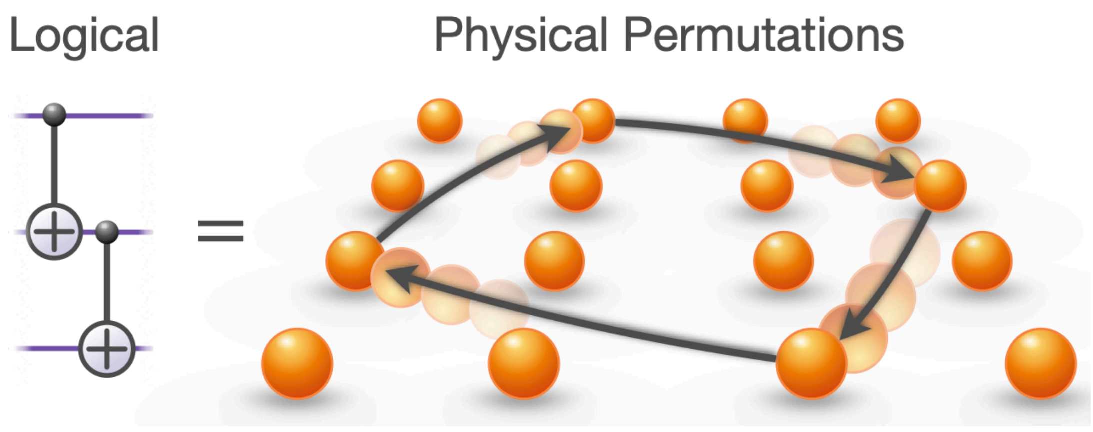
- Other codes

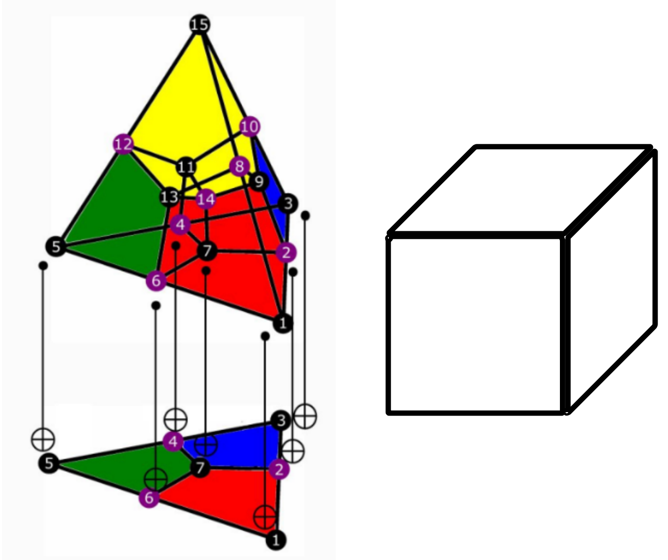
---

[1] Swaroop *et al.* [2410.03628] [2] Xu *et al.* [2510.06159]

Computation on QLDPC codes

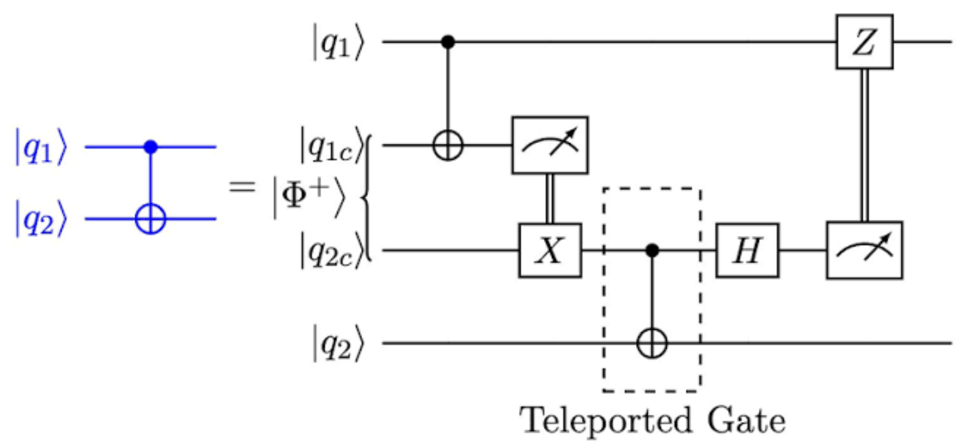
Clifford gates



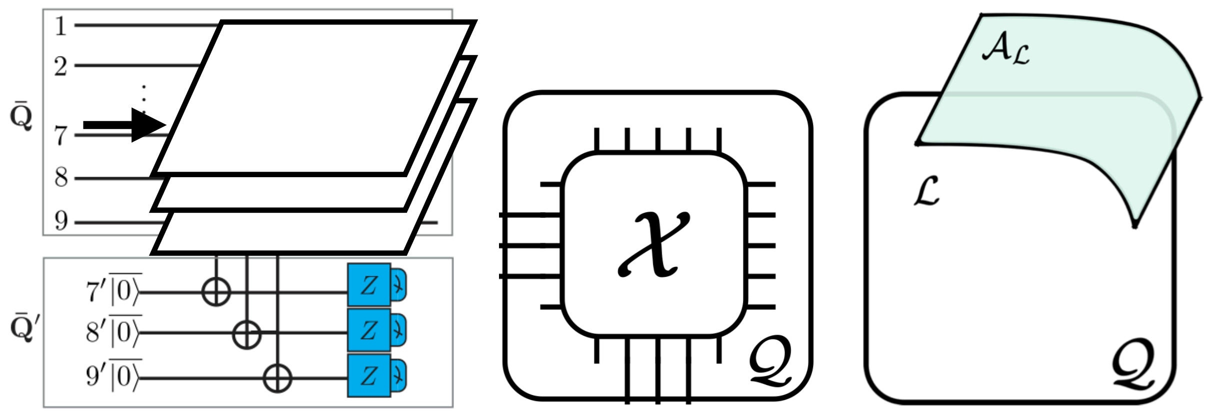


Automorphisms,
Low-depth unitaries

Code-switching



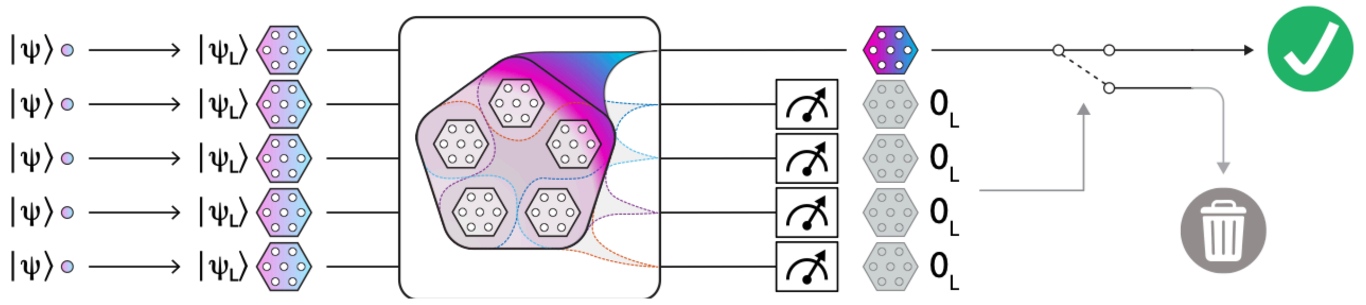
State prep. & gate teleportation

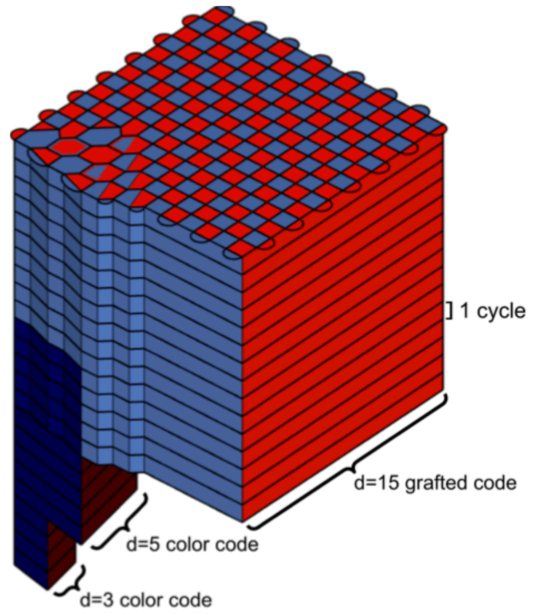


Measuring Pauli operators

Part I. *Fast* primitives for Pauli operator measurements

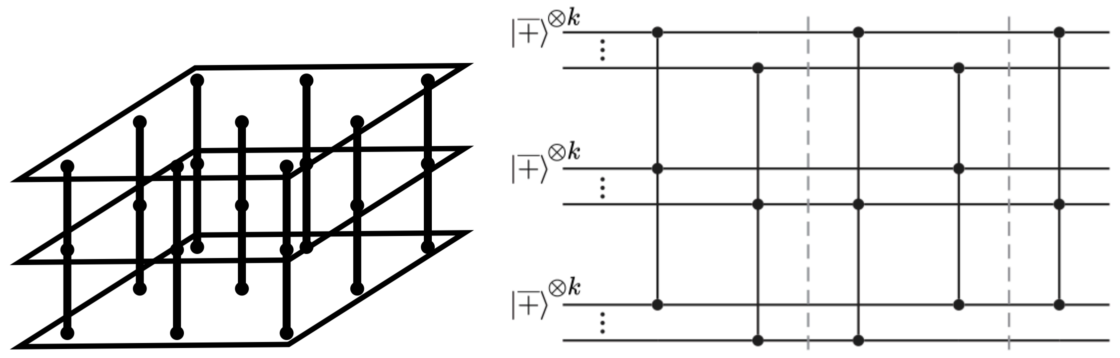
Non-Clifford gates



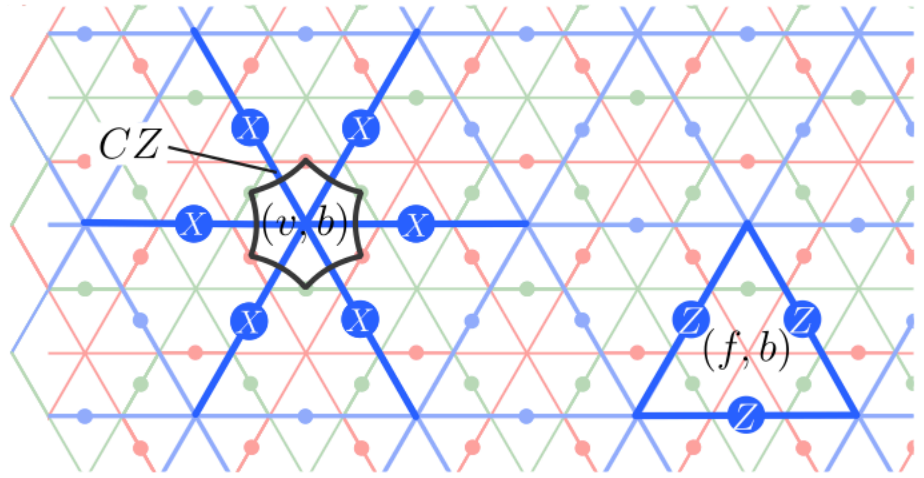


Magic state distillation

Magic state cultivation



Low-depth unitaries

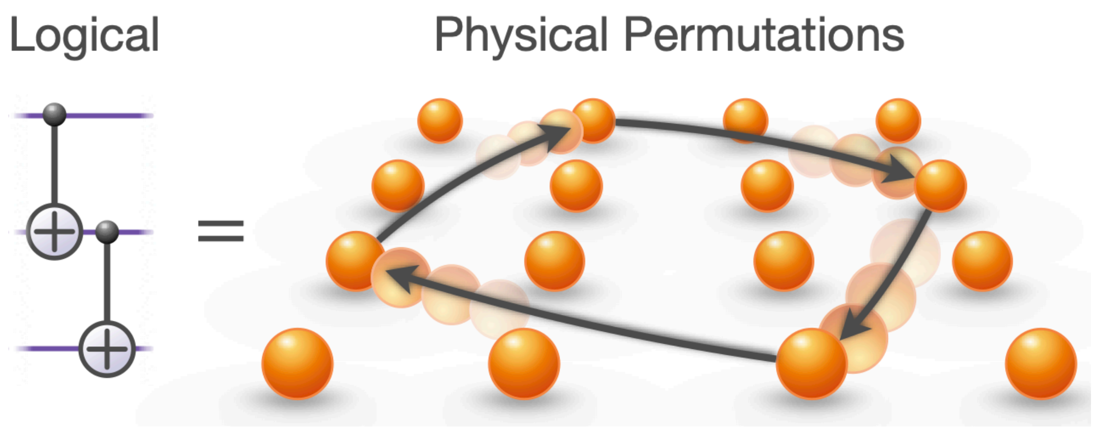


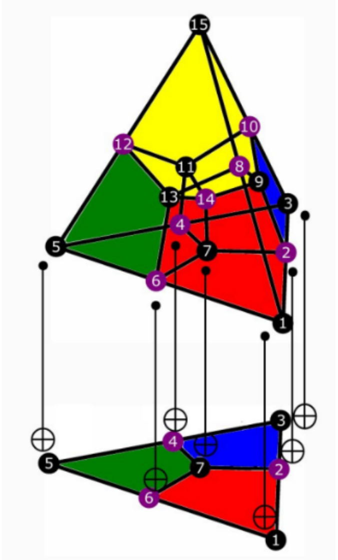
Measuring Clifford operators

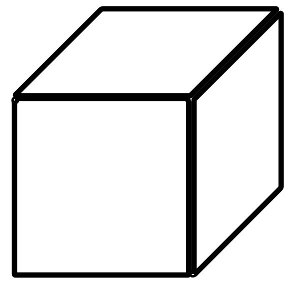
Fig. Credits: Koh, *et al.* [2601.20927], Daguerre, Kim. PRR **7** (2025), Xu, *et al.* PRX **15** (2025), He, *et al.* [2503.10390], Sales Rodriguez, *et al.* Nature **645** (2025), Gidney, *et al.* [2409.17595], Menon, *et* al. PRX **16** (2026).Davydova, *et al.* PRX **16** (2026)

Computation on QLDPC codes

Clifford gates

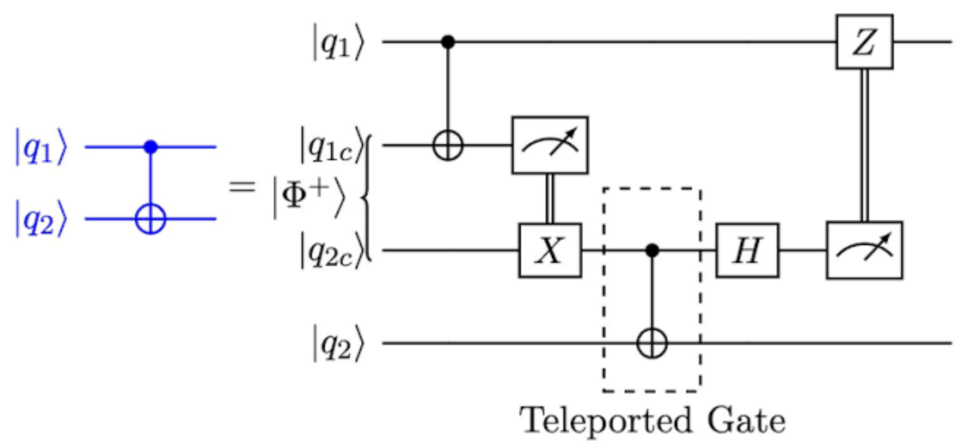




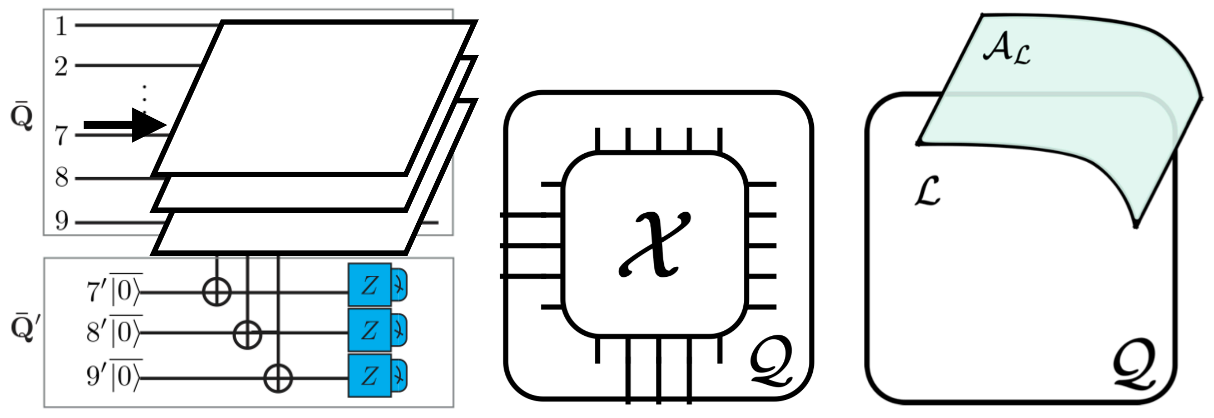


Automorphisms,
Low-depth unitaries

Code-switching

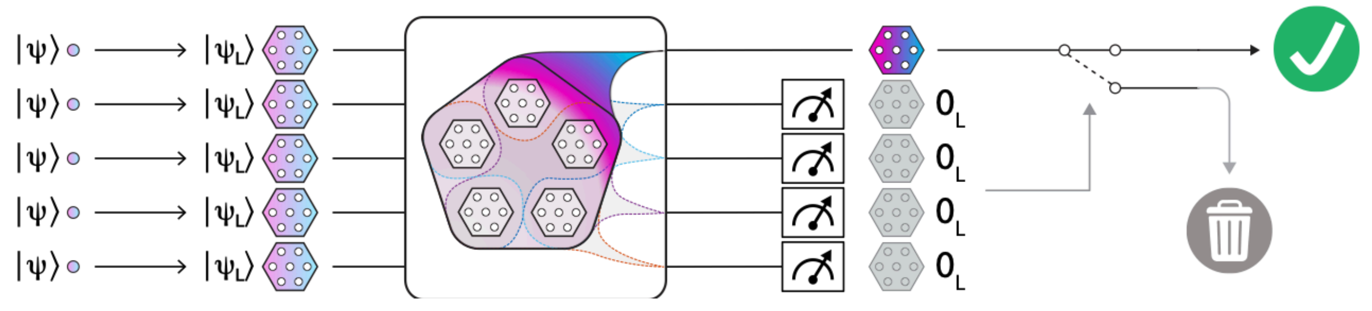


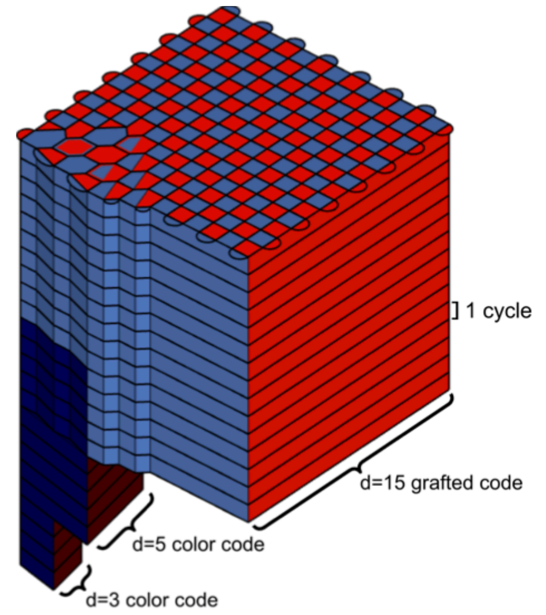
State prep. & gate teleportation



Measuring Pauli operators

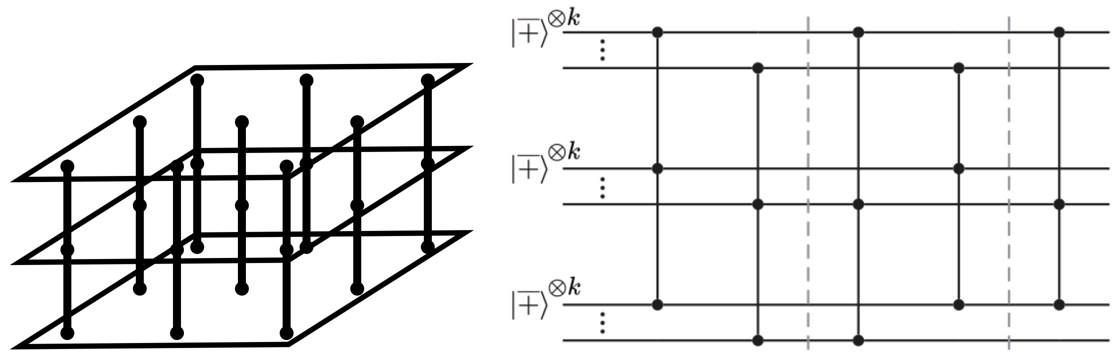
Non-Clifford gates

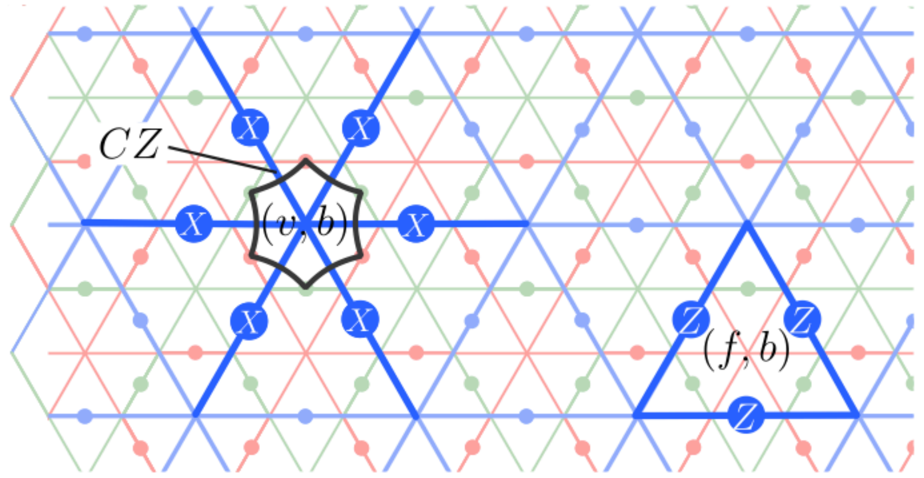




Magic state distillation

Magic state cultivation





Low-depth unitaries

Measuring Clifford operators

Part I. *Fast* primitives for Pauli operator measurements

Part II. New (*Furious*?) techniques for measuring Clifford operators on *QLDPC* codes

Fig. Credits: Koh, *et al.* [2601.20927], Daguerre, Kim. PRR **7** (2025), Xu, *et al.* PRX **15** (2025), He, *et al.* [2503.10390], Sales Rodriguez, *et al.* Nature **645** (2025), Gidney, *et al.* [2409.17595], Menon, *et* al. PRX **16** (2026).Davydova, *et al.* PRX **16** (2026)

Motivation: Measuring Clifford logicals

Motivation: Measuring Clifford logicals

We can implement **Clifford** gates on QLDPC codes, by **measuring** logical **Paulis**.

Motivation: Measuring Clifford logicals

We can implement **Clifford** gates on QLDPC codes, by **measuring** logical **Paulis**.

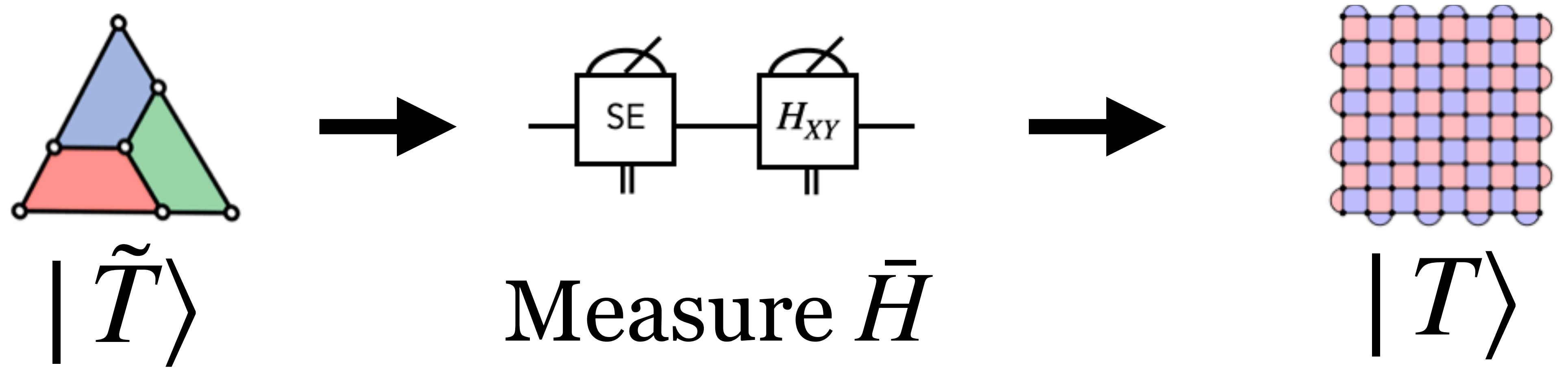
To implement **non-Clifford gates**, we can **measure** logical **Cliffords.**

Motivation: Measuring Clifford logicals

We can implement **Clifford** gates on QLDPC codes, by **measuring** logical **Paulis**.

To implement **non-Clifford gates**, we can **measure** logical **Cliffords.**

Example: Magic state cultivation on topological codes:

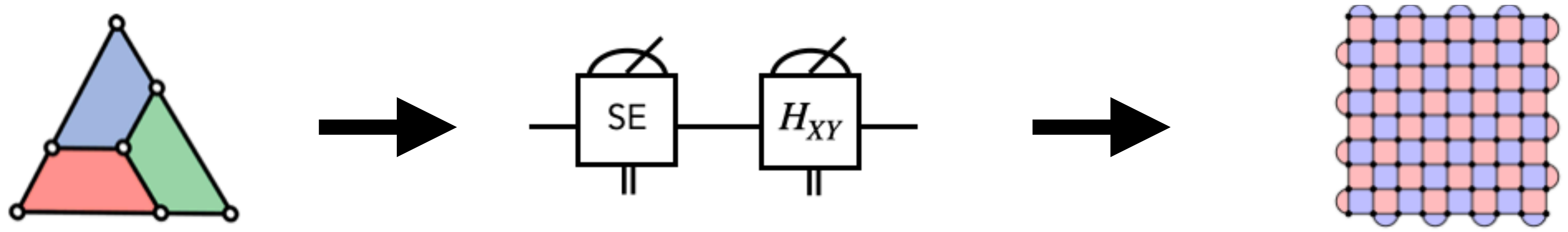


Motivation: Measuring Clifford logicals

We can implement **Clifford** gates on QLDPC codes, by **measuring** logical **Paulis**.

To implement **non-Clifford gates**, we can **measure** logical **Cliffords.**

Example: Magic state cultivation on topological codes:



$|\tilde{T}\rangle$ Measure $\bar{H}$ $|T\rangle$

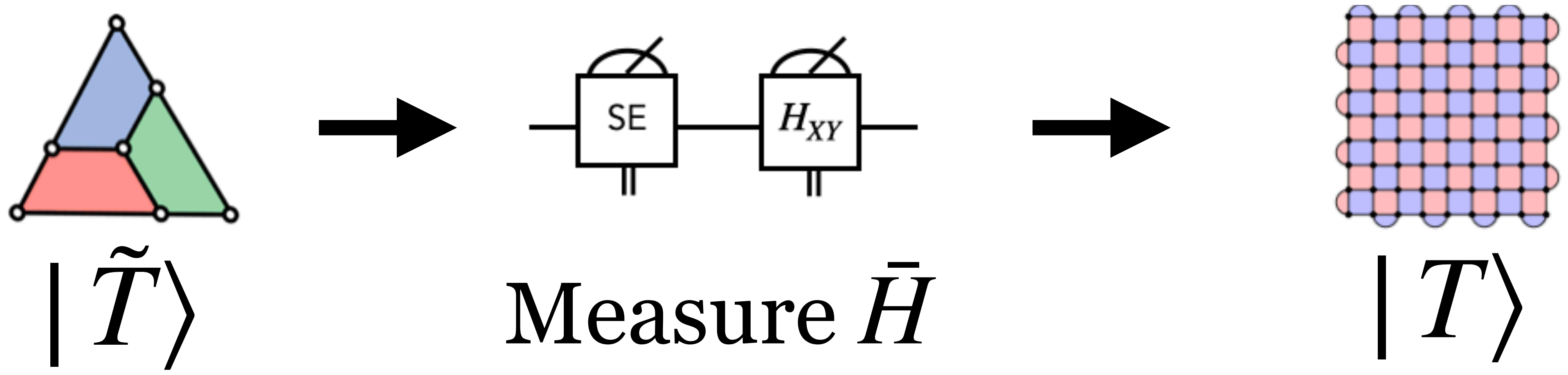
- *Uses post-selection on flagged cat state measurements*

Motivation: Measuring Clifford logicals

We can implement **Clifford** gates on QLDPC codes, by **measuring** logical **Paulis**.

To implement **non-Clifford gates**, we can **measure** logical **Cliffords.**

Example: Magic state cultivation on topological codes:



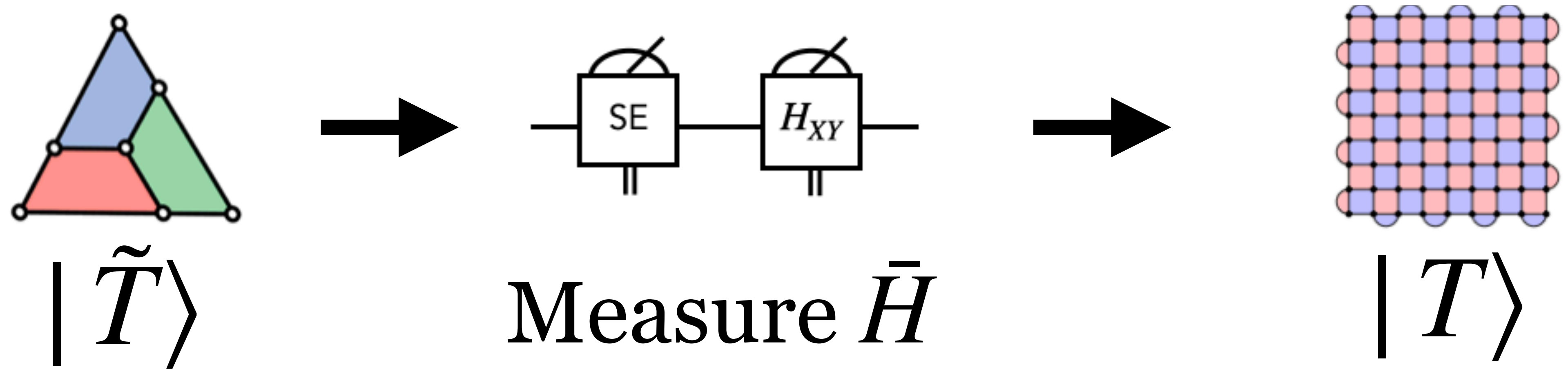
- *Uses post-selection on flagged cat state measurements*
- *Magic state in a low-rate code*

Motivation: Measuring Clifford logicals

We can implement **Clifford** gates on QLDPC codes, by **measuring** logical **Paulis**.

To implement **non-Clifford gates**, we can **measure** logical **Cliffords.**

Example: Magic state cultivation on topological codes:



- *Uses post-selection on flagged cat state measurements*
- *Magic state in a low-rate code*

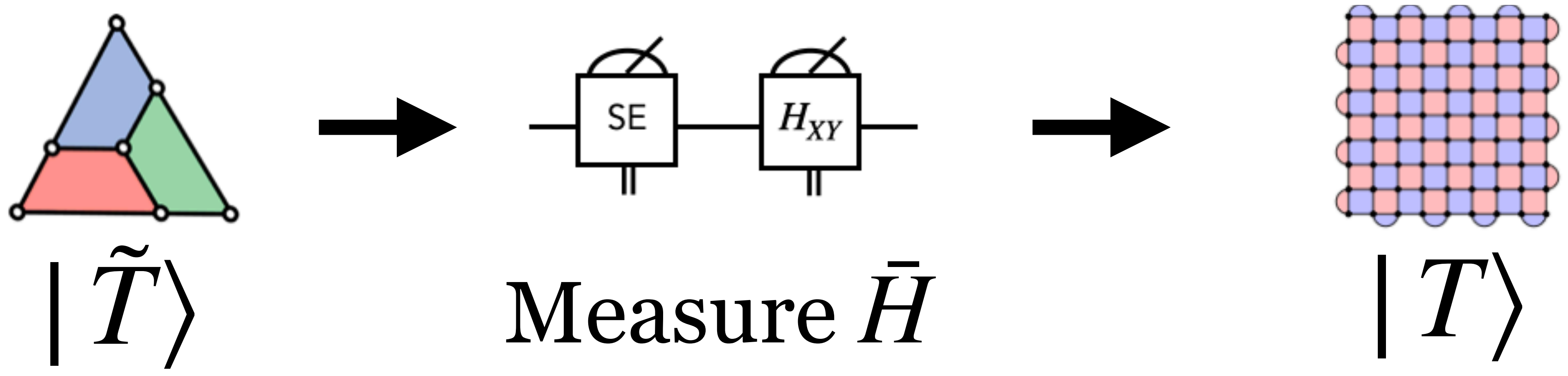
This talk:

Motivation: Measuring Clifford logicals

We can implement **Clifford** gates on QLDPC codes, by **measuring** logical **Paulis**.

To implement **non-Clifford gates**, we can **measure** logical **Cliffords.**

Example: Magic state cultivation on topological codes:



- *Uses post-selection on flagged cat state measurements*
- *Magic state in a low-rate code*

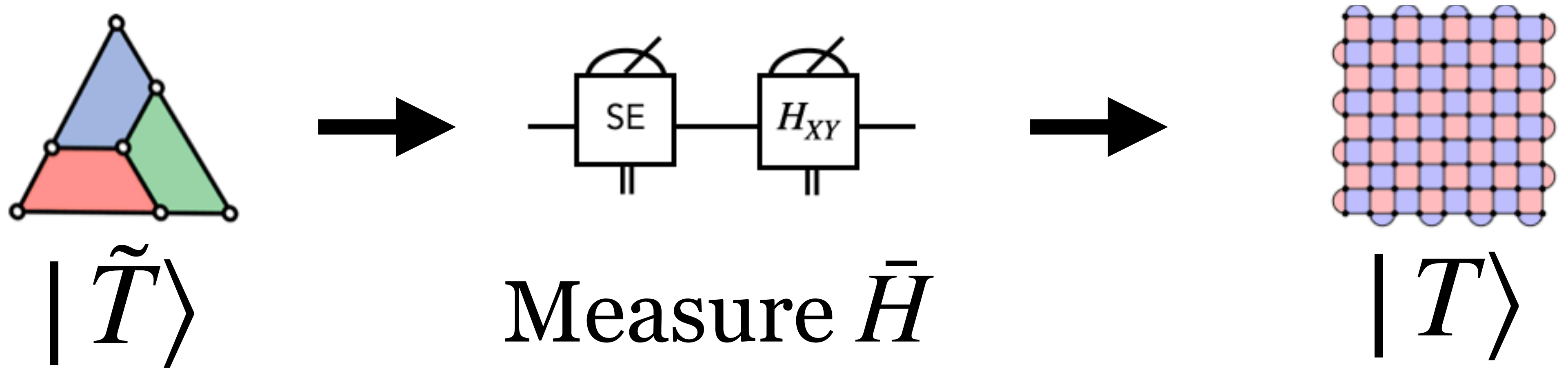
This talk:
- *Aim to have an error-corrected code surgery primitive*

Motivation: Measuring Clifford logicals

We can implement **Clifford** gates on QLDPC codes, by **measuring** logical **Paulis**.

To implement **non-Clifford gates**, we can **measure** logical **Cliffords.**

Example: Magic state cultivation on topological codes:



- *Uses post-selection on flagged cat state measurements*
- *Magic state in a low-rate code*

This talk:

- *Aim to have an error-corrected code surgery primitive*
- *Magic states/non-Clifford gates in high-rate codes*

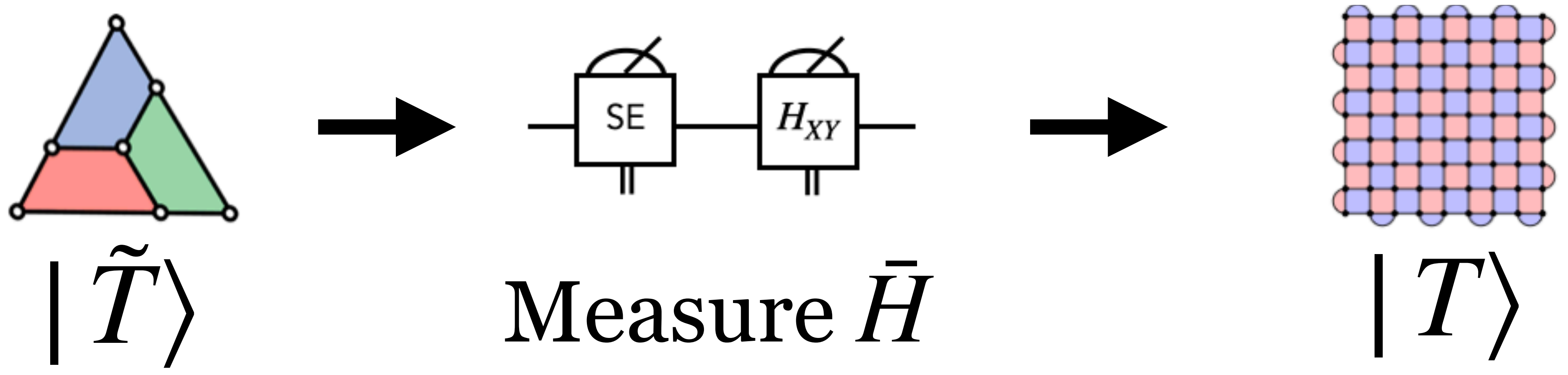
On topological codes, see related work by Brown. Sci. Adv. **6** (2024), Davydova, et al. PRX **16.3** (2026), Bauer, Magdalena de la Fuente PRXQ **7.2** (2026).
Practical implementation: Scruby, et al. Quantum **6** (2022), Magdalena de la Fuente, et al. [2604.02033]

Motivation: Measuring Clifford logicals

We can implement **Clifford** gates on QLDPC codes, by **measuring** logical **Paulis**.

To implement **non-Clifford gates**, we can **measure** logical **Cliffords.**

Example: Magic state cultivation on topological codes:



- *Uses post-selection on flagged cat state measurements*
- *Magic state in a low-rate code*

This talk:
- *Aim to have an error-corrected code surgery primitive*
- *Magic states/non-Clifford gates in high-rate codes*

We can measure logical Cliffords in QLDPC codes with large fault-distance.

On topological codes, see related work by Brown. Sci. Adv. **6** (2024), Davydova, et al. PRX **16.3** (2026), Bauer, Magdalena de la Fuente PRXQ **7.2** (2026).
Practical implementation: Scruby, et al. Quantum **6** (2022), Magdalena de la Fuente, et al. [2604.02033]

See also, Christos et al. [2602.12228]. Williamson [2601.22939]. Zhu, et al. [2601.06736].

Result: Fault-tolerant protocol for measuring a Clifford logical operator

Result: Fault-tolerant protocol for measuring a Clifford logical operator

Consider a CSS code C with an order-2 Clifford logical operator T. ($\mathsf{T}^2 = I$)

Result: Fault-tolerant protocol for measuring a Clifford logical operator

Consider a CSS code C with an order-2 Clifford logical operator T. ($\mathsf{T}^2 = I$)

Let T be decomposable into a tensor product of constant-sized operators $\{\mathsf{T}_v\}_v$

Result: Fault-tolerant protocol for measuring a Clifford logical operator

Consider a CSS code C with an order-2 Clifford logical operator T. ($\mathsf{T}^2 = I$)

Let T be decomposable into a tensor product of constant-sized operators $\{\mathsf{T}_v\}_v$

$$\mathsf{T} = \bigotimes_v \mathsf{T}_v$$

Result: Fault-tolerant protocol for measuring a Clifford logical operator

Consider a CSS code C with an order-2 Clifford logical operator T. ($\mathsf{T}^2 = I$)

Let T be decomposable into a tensor product of constant-sized operators $\{\mathsf{T}_v\}_v$

$$\mathsf{T} = \otimes_v \mathsf{T}_v \qquad\qquad |\mathsf{T}_v| = O(1)$$

Result: Fault-tolerant protocol for measuring a Clifford logical operator

Consider a CSS code C with an order-2 Clifford logical operator T. ($\mathsf{T}^2 = I$)
Let T be decomposable into a tensor product of constant-sized operators $\{\mathsf{T}_v\}_v$

$$\mathsf{T} = \otimes_v \mathsf{T}_v \qquad\qquad |\mathsf{T}_v| = O(1)$$

Then there is a protocol that switches to a gauged code, C_{g}, that measures T.
The whole protocol has an adversarial fault-distance to data qubit and measurement errors that is

Result: Fault-tolerant protocol for measuring a Clifford logical operator

Consider a CSS code C with an order-2 Clifford logical operator T. ($\mathsf{T}^2 = I$)
Let T be decomposable into a tensor product of constant-sized operators $\{\mathsf{T}_v\}_v$

$$\mathsf{T} = \bigotimes_v \mathsf{T}_v \qquad\qquad |\mathsf{T}_v| = O(1)$$

Then there is a protocol that switches to a gauged code, C_g, that measures T.
The whole protocol has an adversarial fault-distance to data qubit and measurement errors that is

$$d_\mathrm{F} \geq d/\gamma \qquad\qquad \gamma \geq 1, \text{ and } \gamma = O(1).$$

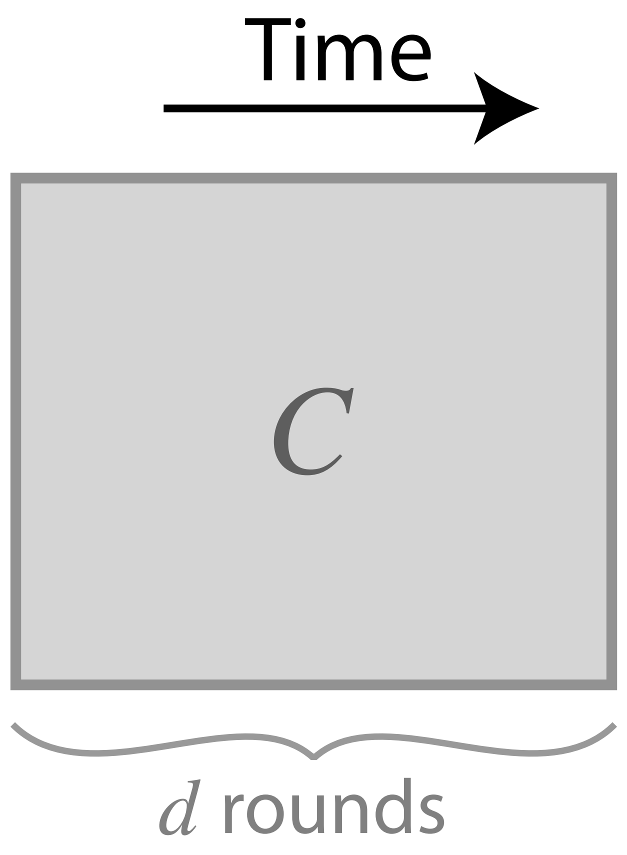
Result: Fault-tolerant protocol for measuring a Clifford logical operator

Consider a CSS code C with an order-2 Clifford logical operator T. ($\mathsf{T}^2 = I$)
Let T be decomposable into a tensor product of constant-sized operators $\{\mathsf{T}_v\}_v$

$$\mathsf{T} = \otimes_v \mathsf{T}_v \qquad\qquad |\mathsf{T}_v| = O(1)$$

Then there is a protocol that switches to a gauged code, C_{g}, that measures T.
The whole protocol has an adversarial fault-distance to data qubit and measurement errors that is

$$d_{\mathrm{F}} \geq d/\gamma \qquad\qquad \gamma \geq 1, \text{ and } \gamma = O(1).$$



1. Start in a CSS code C with a logical Clifford T

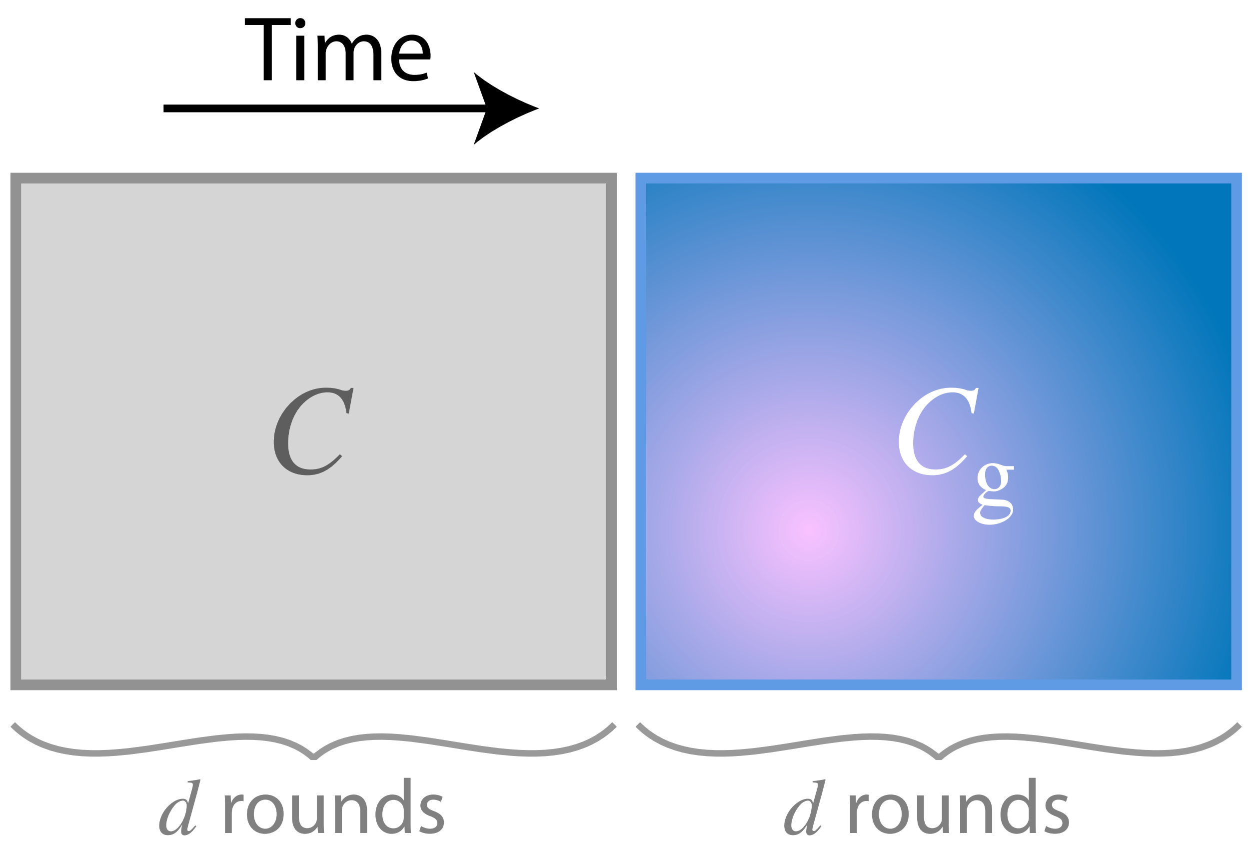
Result: Fault-tolerant protocol for measuring a Clifford logical operator

Consider a CSS code C with an order-2 Clifford logical operator T. ($\mathsf{T}^2 = I$)
Let T be decomposable into a tensor product of constant-sized operators $\{\mathsf{T}_v\}_v$

$$\mathsf{T} = \otimes_v \mathsf{T}_v \qquad\qquad |\mathsf{T}_v| = O(1)$$

Then there is a protocol that switches to a gauged code, C_{g}, that measures T.
The whole protocol has an adversarial fault-distance to data qubit and measurement errors that is

$$d_{\mathrm{F}} \geq d/\gamma \qquad\qquad \gamma \geq 1, \text{ and } \gamma = O(1).$$



1. Start in a CSS code C with a logical Clifford T
2. Add ancilla qubits, measure stabilizers of C_{g}

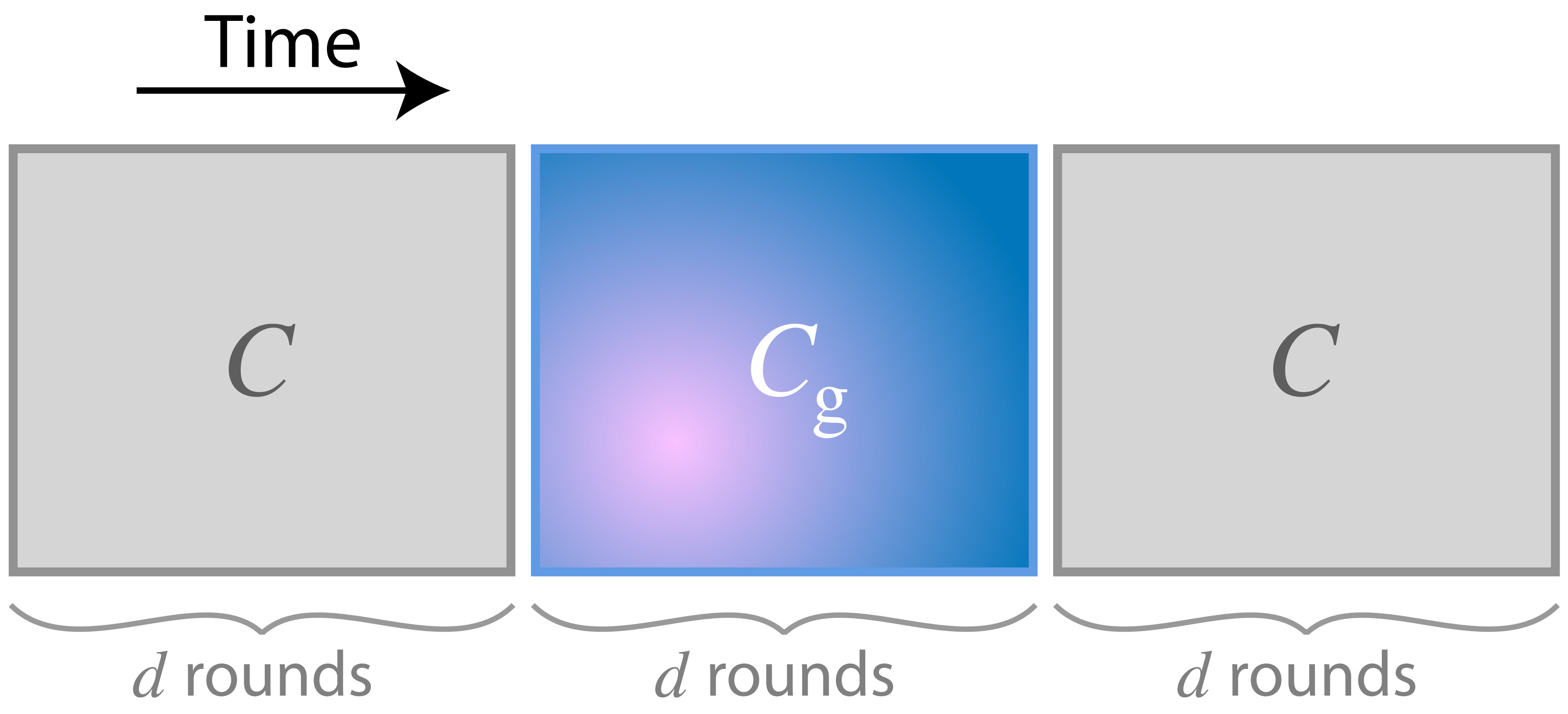
Result: Fault-tolerant protocol for measuring a Clifford logical operator

Consider a CSS code C with an order-2 Clifford logical operator T. ($\mathsf{T}^2 = I$)
Let T be decomposable into a tensor product of constant-sized operators $\{\mathsf{T}_v\}_v$

$$\mathsf{T} = \otimes_v \mathsf{T}_v \qquad\qquad |\mathsf{T}_v| = O(1)$$

Then there is a protocol that switches to a gauged code, C_g, that measures T.
The whole protocol has an adversarial fault-distance to data qubit and measurement errors that is

$$d_F \geq d/\gamma \qquad\qquad \gamma \geq 1, \text{ and } \gamma = O(1).$$



1. Start in a CSS code C with a logical Clifford T
2. Add ancilla qubits, measure stabilizers of C_g
3. Return to C.

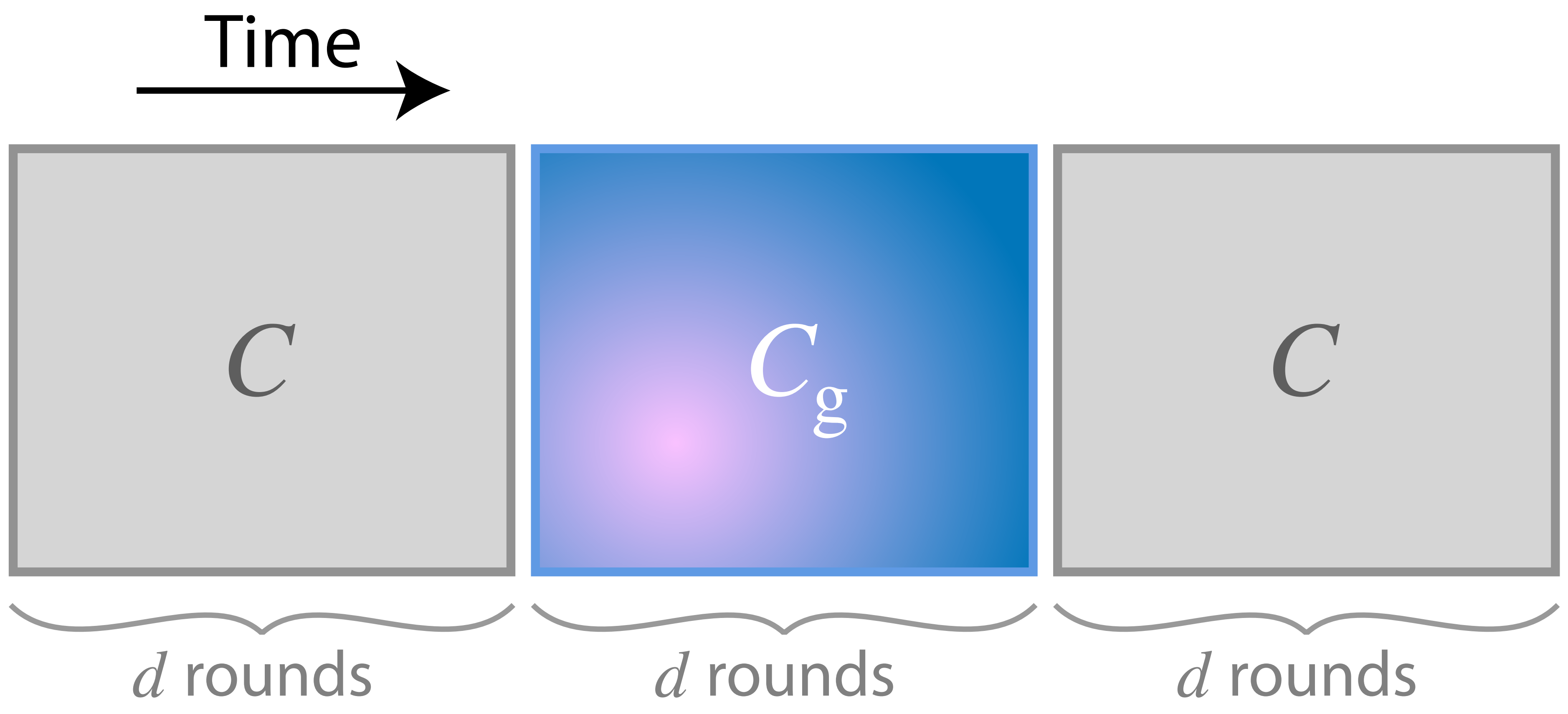
Result: Fault-tolerant protocol for measuring a Clifford logical operator

Consider a CSS code C with an order-2 Clifford logical operator T. ($\mathsf{T}^2 = I$)
Let T be decomposable into a tensor product of constant-sized operators $\{\mathsf{T}_v\}_v$

$$\mathsf{T} = \otimes_v \mathsf{T}_v \qquad\qquad |\mathsf{T}_v| = O(1)$$

Then there is a protocol that switches to a gauged code, C_g, that measures T.
The whole protocol has an adversarial fault-distance to data qubit and measurement errors that is

$$d_F \geq d/\gamma \qquad\qquad \gamma \geq 1, \text{ and } \gamma = O(1).$$



1. Start in a CSS code C with a logical Clifford T
2. Add ancilla qubits, measure stabilizers of C_g
3. Return to C.

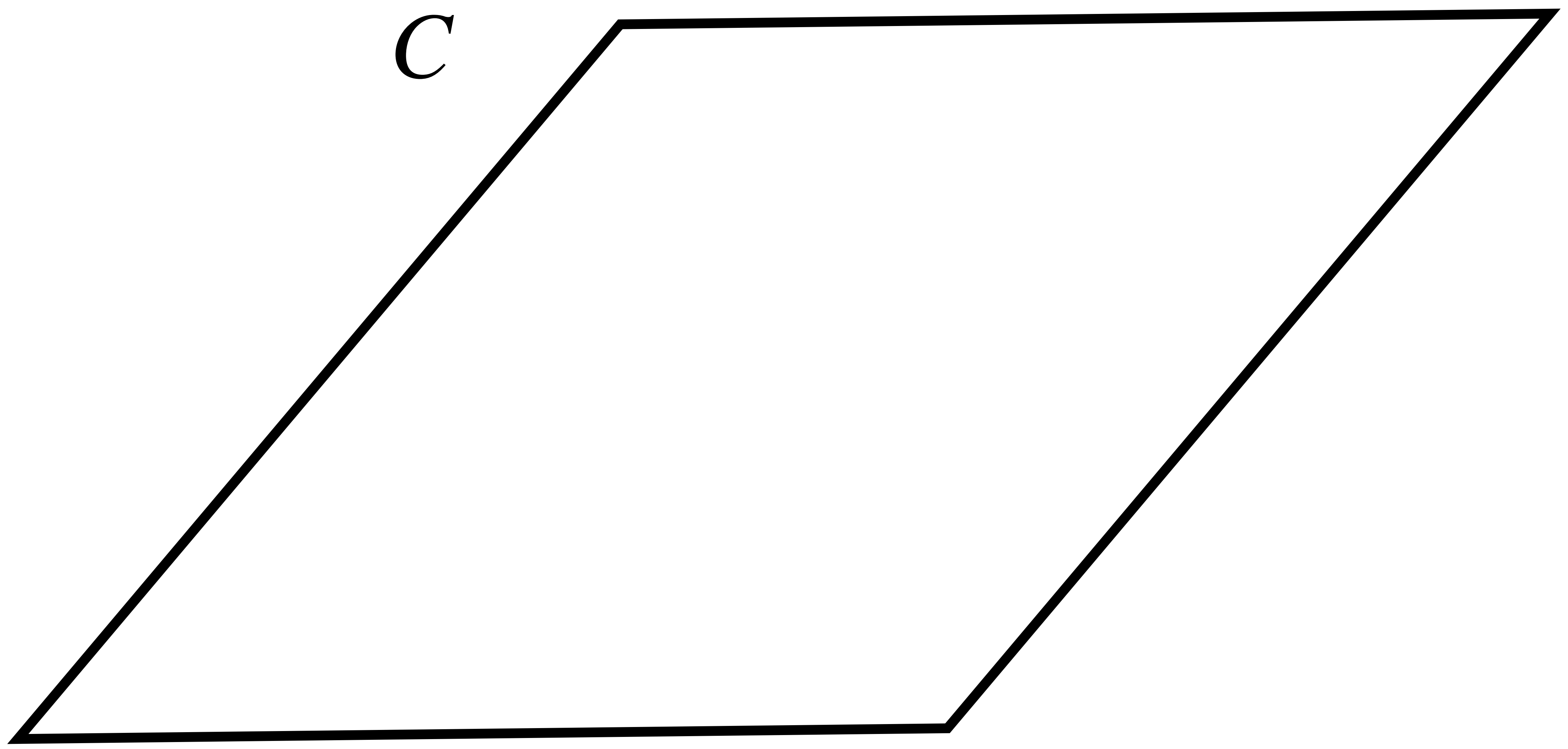
Next: what is C_g?

Constructing the gauged code

Tantivasadakarn, et al. PRX **14** (2024)
Christos, et al. [2602.12228]

Constructing the gauged code

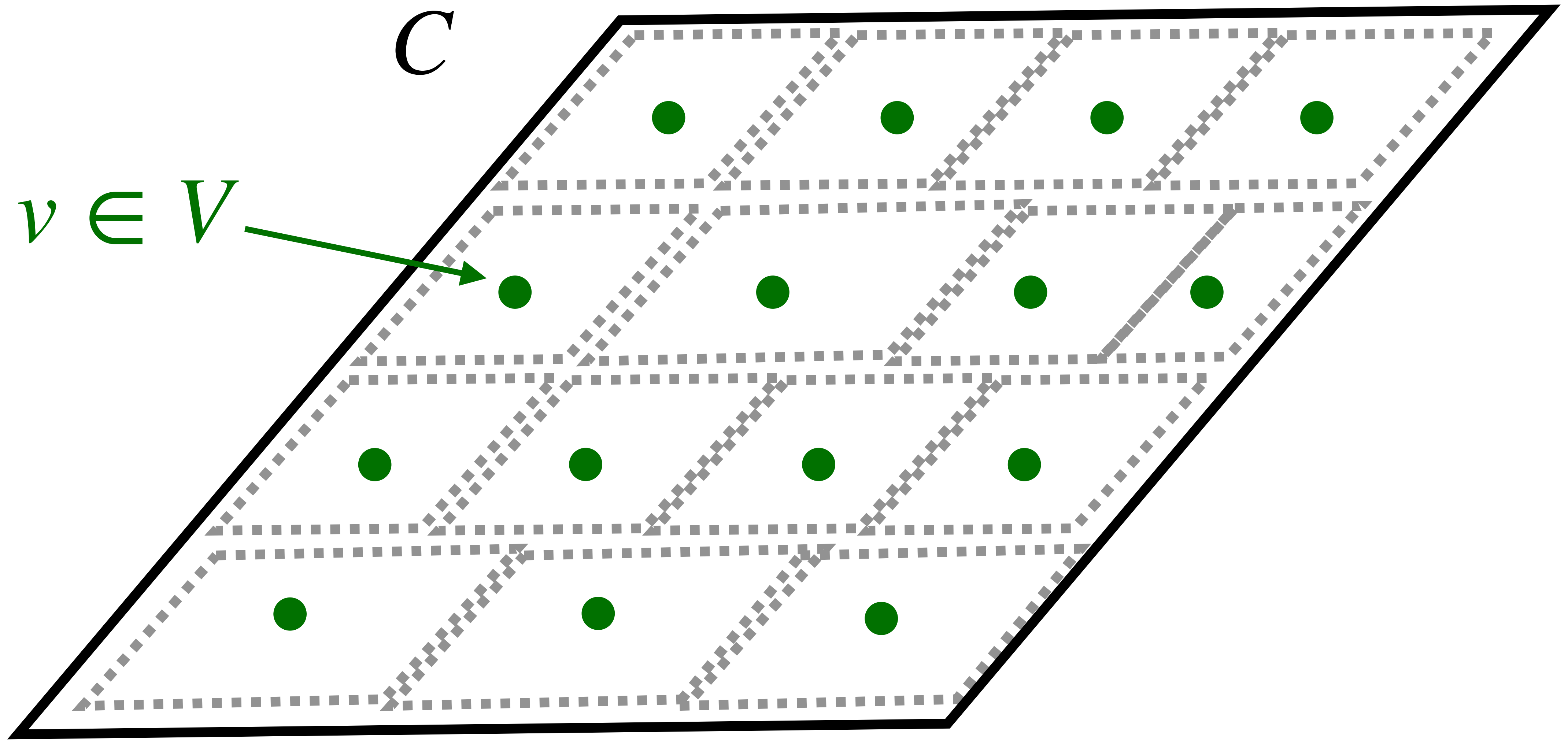
$$\mathsf{T} = \bigotimes_v \mathsf{T}_v \qquad |\mathsf{T}_v| = O(1)$$



Tantivasadakarn, et al. PRX **14** (2024)
Christos, et al. [2602.12228]

Constructing the gauged code

$$\mathsf{T} = \otimes_v \mathsf{T}_v \qquad |\mathsf{T}_v| = O(1)$$

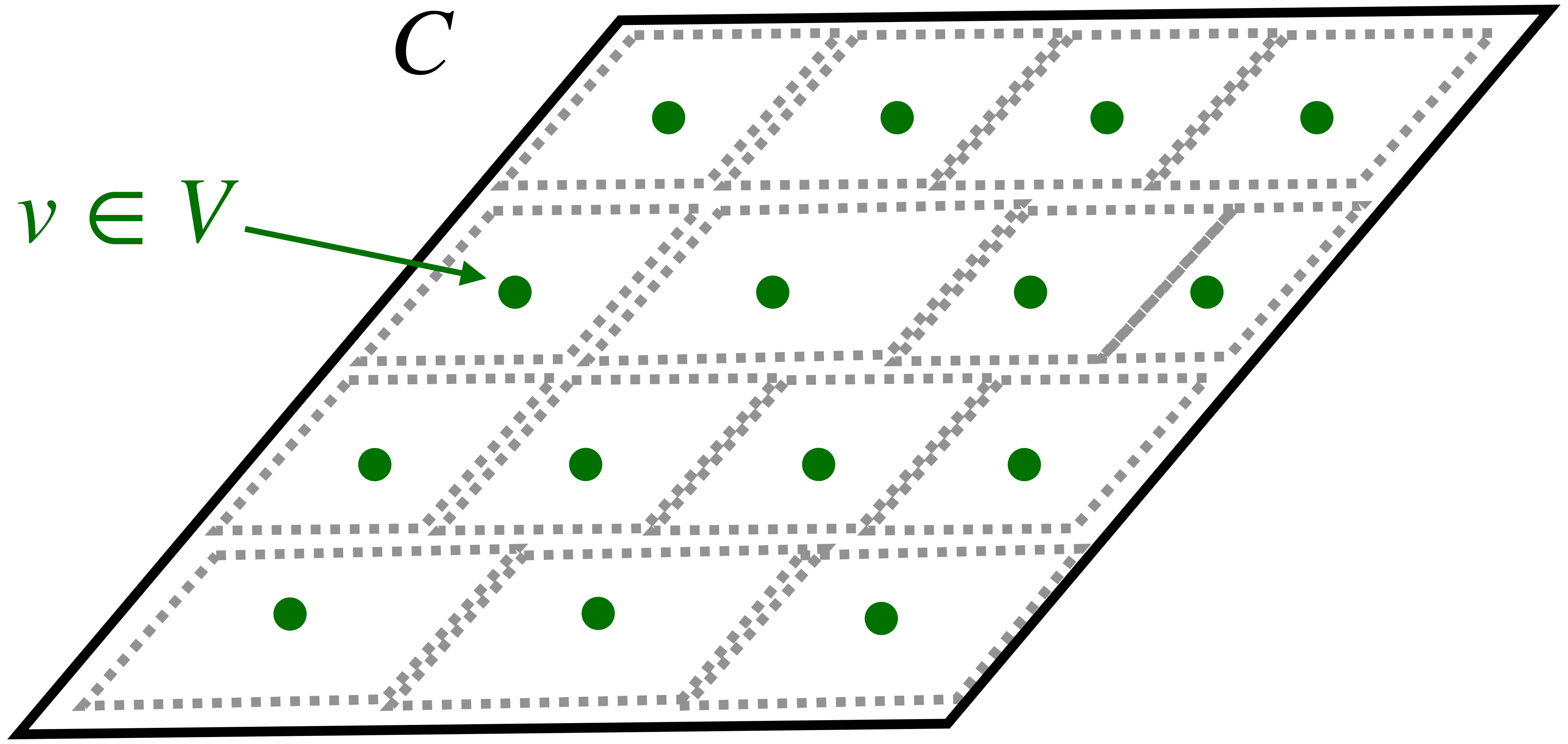


1. Partition data qubits into constant-sized vertex sets.

Tantivasadakarn, et al. PRX **14** (2024)
Christos, et al. [2602.12228]

Constructing the gauged code

$$\mathsf{T} = \otimes_v \mathsf{T}_v \qquad |\mathsf{T}_v| = O(1)$$



Ex. $\overline{H}_{\mathrm{XY}} = H_{\mathrm{XY}}^{\otimes n}$ Single-qubit vertex sets

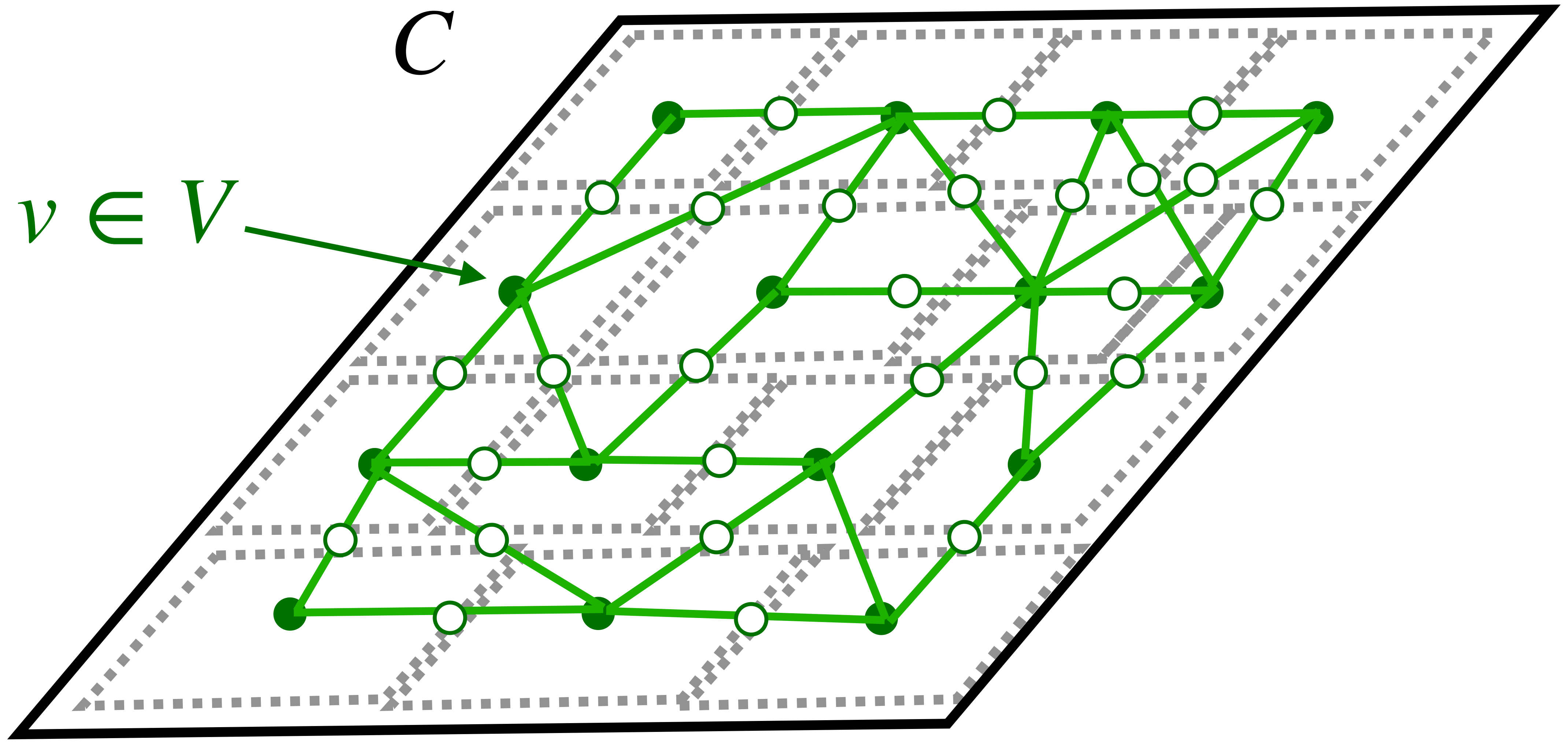
Ex. $\overline{CX}^{\otimes k}_{\mathrm{ctrl}\to\mathrm{tar}} = CX^{\otimes n}_{\mathrm{ctrl}\to\mathrm{tar}}$ Two-qubit vertex sets: (ctrl, tar) pair

1. Partition data qubits into constant-sized vertex sets.

Tantivasadakarn, et al. PRX **14** (2024)
Christos, et al. [2602.12228]

Constructing the gauged code

$\mathsf{T} = \bigotimes_v \mathsf{T}_v \qquad |\mathsf{T}_v| = O(1)$



Ex. $\overline{H}_{\mathrm{XY}} = H_{\mathrm{XY}}^{\otimes n}$ Single-qubit vertex sets

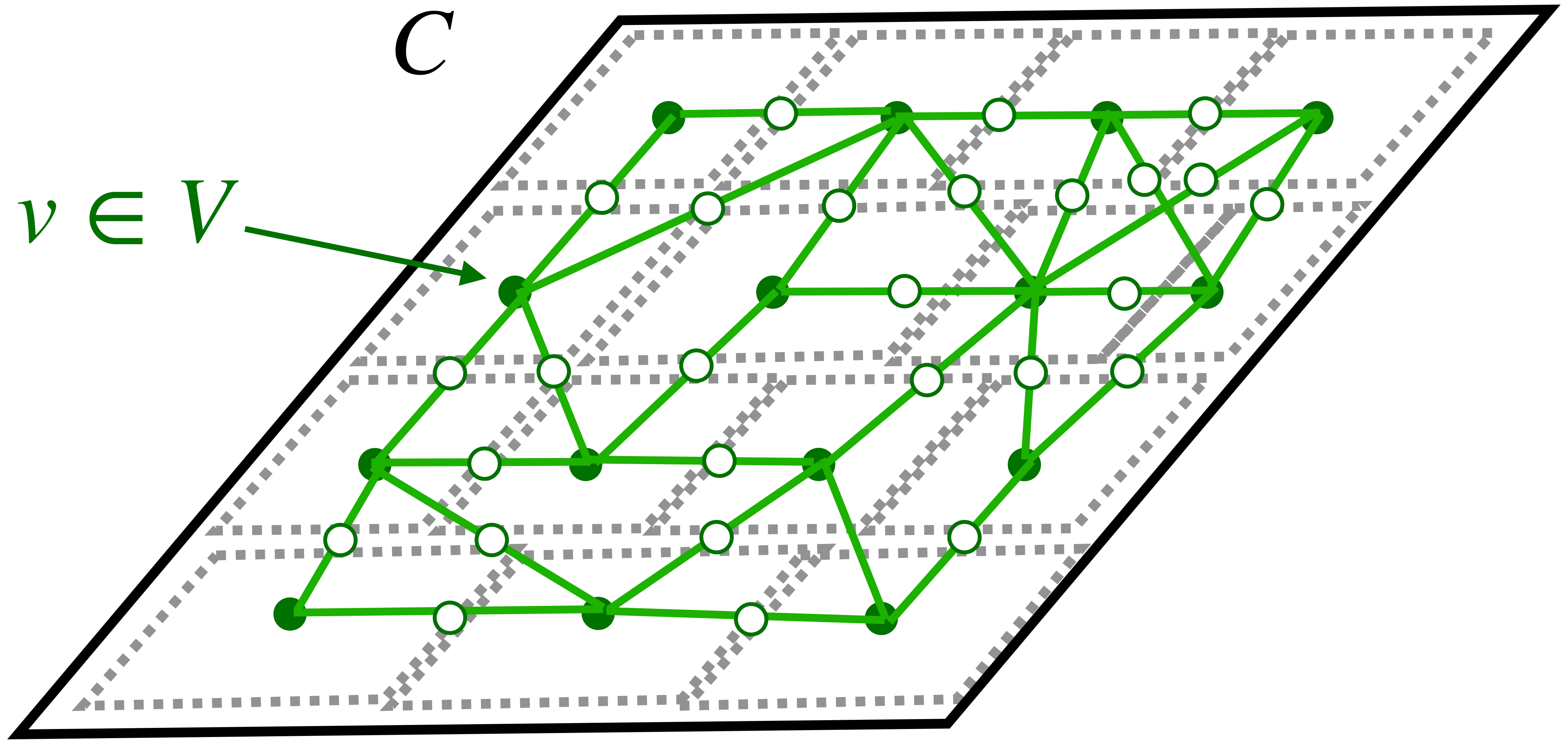
Ex. $\overline{CX}^{\otimes k}_{\mathrm{ctrl}\to\mathrm{tar}} = CX^{\otimes n}_{\mathrm{ctrl}\to\mathrm{tar}}$ Two-qubit vertex sets: (ctrl, tar) pair

1. Partition data qubits into constant-sized vertex sets.
2. Construct a sufficiently expanding graph with |V| vertices and a sparse cycle basis [WY24].

Tantivasadakarn, et al. PRX **14** (2024)
Christos, et al. [2602.12228]

Constructing the gauged code

$$\mathsf{T} = \otimes_v \mathsf{T}_v \qquad |\mathsf{T}_v| = O(1)$$



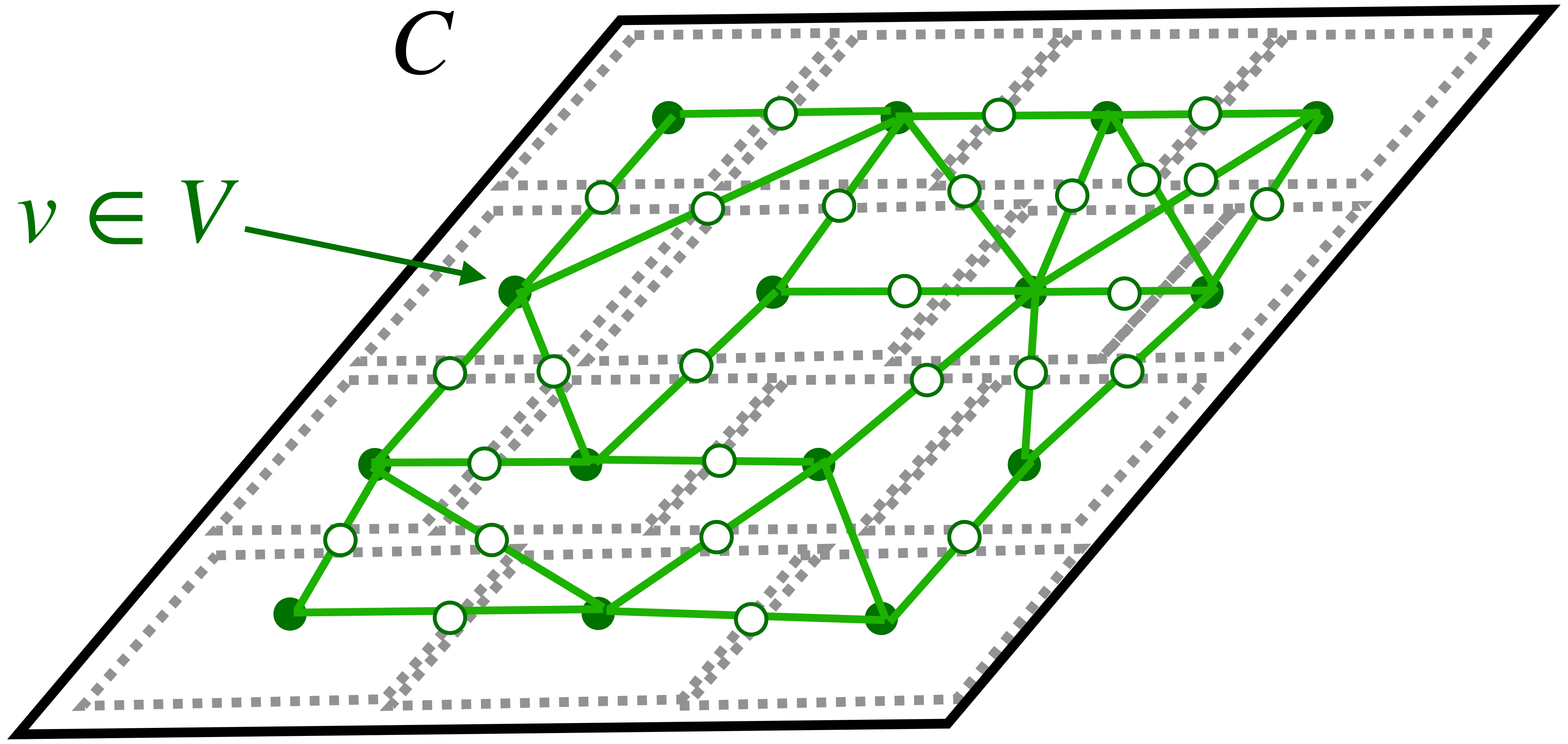
Ex. $\overline{H}_{XY} = H_{XY}^{\otimes n}$ Single-qubit vertex sets

Ex. $\overline{CX}^{\otimes k}_{\text{ctrl}\to\text{tar}} = CX^{\otimes n}_{\text{ctrl}\to\text{tar}}$ Two-qubit vertex sets: (ctrl, tar) pair

1. Partition data qubits into constant-sized vertex sets.
2. Construct a sufficiently expanding graph with |V| vertices and a sparse cycle basis [WY24].
3. For each vertex, initialize $|+\rangle$ & for each edge, initialize $|0\rangle$.

Tantivasadakarn, et al. PRX **14** (2024)
Christos, et al. [2602.12228]

Constructing the gauged code



$$\mathsf{T} = \otimes_v \mathsf{T}_v \qquad |\mathsf{T}_v| = O(1)$$

Ex. $\overline{H}_{\mathrm{XY}} = H_{\mathrm{XY}}^{\otimes n}$ Single-qubit vertex sets

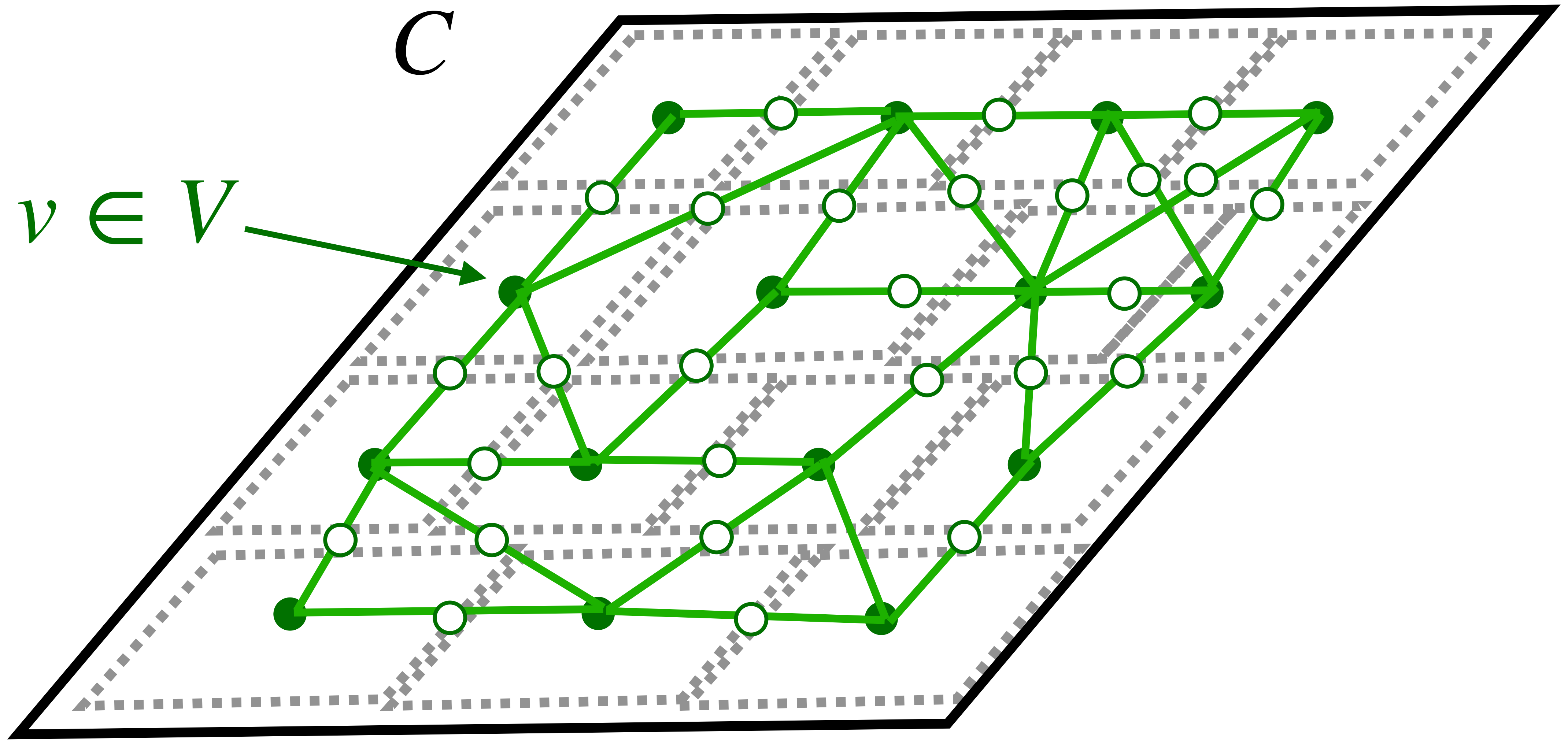
Ex. $\overline{CX}^{\otimes k}_{\mathrm{ctrl}\to\mathrm{tar}} = CX^{\otimes n}_{\mathrm{ctrl}\to\mathrm{tar}}$ Two-qubit vertex sets: (ctrl, tar) pair

1. Partition data qubits into constant-sized vertex sets.
2. Construct a sufficiently expanding graph with |V| vertices and a sparse cycle basis [WY24].
3. For each vertex, initialize $|+\rangle$ & for each edge, initialize $|0\rangle$.
4. Apply $C_v \mathsf{T}_v$ for each v.

Tantivasadakarn, et al. PRX **14** (2024)
Christos, et al. [2602.12228]

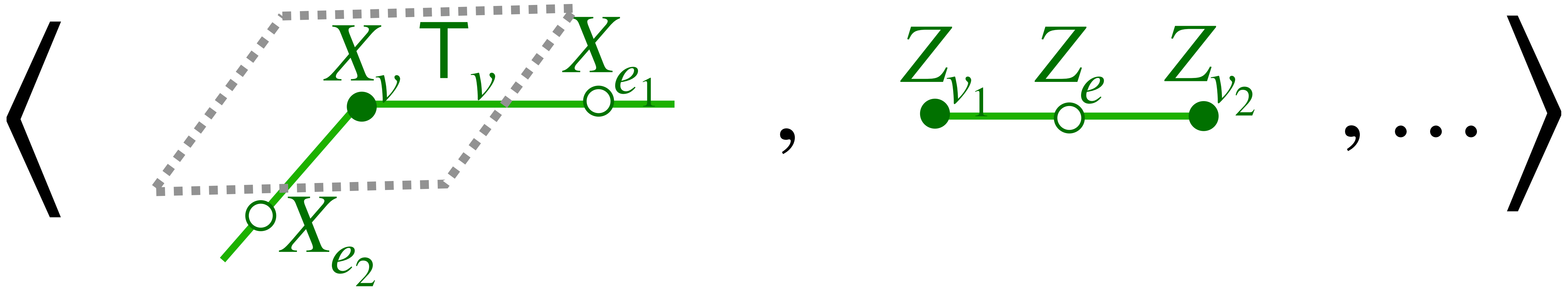
Constructing the gauged code

$$\mathsf{T} = \bigotimes_v \mathsf{T}_v \qquad |\mathsf{T}_v| = O(1)$$



Ex. $\overline{H}_{\mathrm{XY}} = H_{\mathrm{XY}}^{\otimes n}$ Single-qubit vertex sets

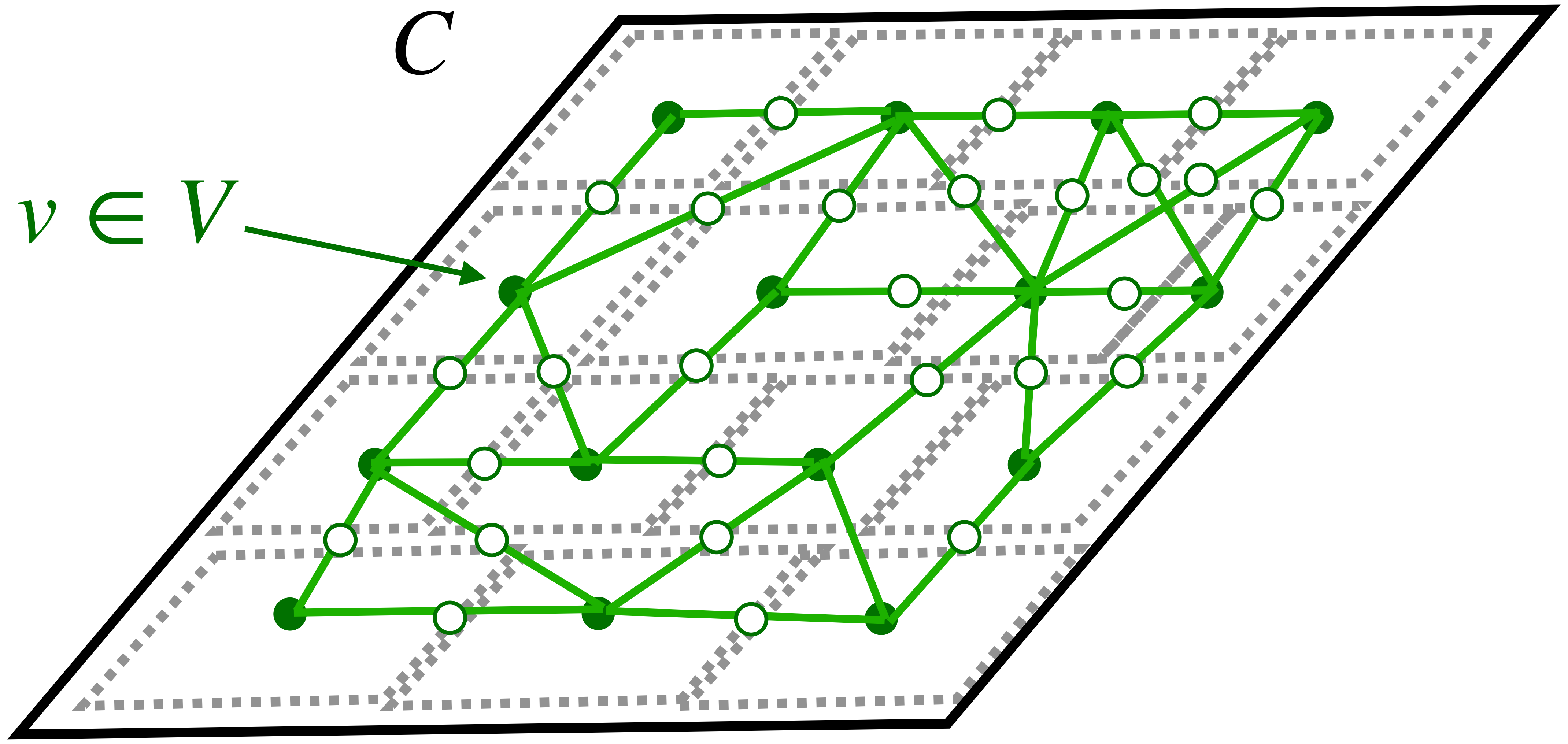
Ex. $\overline{CX}^{\otimes k}_{\mathrm{ctrl}\to\mathrm{tar}} = CX^{\otimes n}_{\mathrm{ctrl}\to\mathrm{tar}}$ Two-qubit vertex sets: (ctrl, tar) pair



1. Partition data qubits into constant-sized vertex sets.
2. Construct a sufficiently expanding graph with |V| vertices and a sparse cycle basis [WY24].
3. For each vertex, initialize $|+\rangle$ & for each edge, initialize $|0\rangle$.
4. Apply $C_v\mathsf{T}_v$ for each v.
5. Apply C_vX_e for each $e \in \delta(v)$.

Tantivasadakarn, et al. PRX **14** (2024)
Christos, et al. [2602.12228]

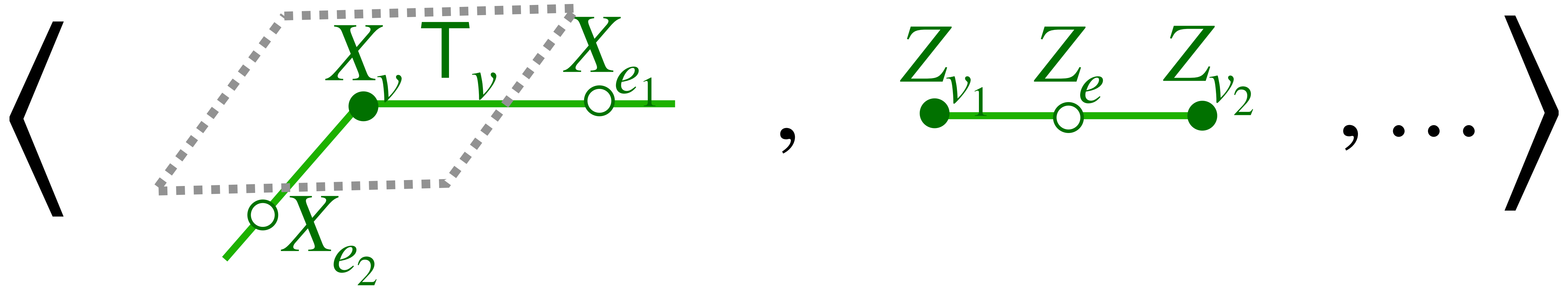
Constructing the gauged code



$$\mathsf{T} = \bigotimes_v \mathsf{T}_v \qquad |\mathsf{T}_v| = O(1)$$

Ex. $\overline{H}_{\mathrm{XY}} = H_{\mathrm{XY}}^{\otimes n}$ Single-qubit vertex sets

Ex. $\overline{CX}^{\otimes k}_{\mathrm{ctrl}\to\mathrm{tar}} = CX^{\otimes n}_{\mathrm{ctrl}\to\mathrm{tar}}$ Two-qubit vertex sets: (ctrl, tar) pair



1. Partition data qubits into constant-sized vertex sets.
2. Construct a sufficiently expanding graph with |V| vertices and a sparse cycle basis [WY24].
3. For each vertex, initialize $|+\rangle$ & for each edge, initialize $|0\rangle$.
4. Apply $C_v\mathsf{T}_v$ for each v.
5. Apply C_vX_e for each $e \in \delta(v)$.
6. Measure all v in X.

$$\prod_v x_v = \mathsf{T}$$

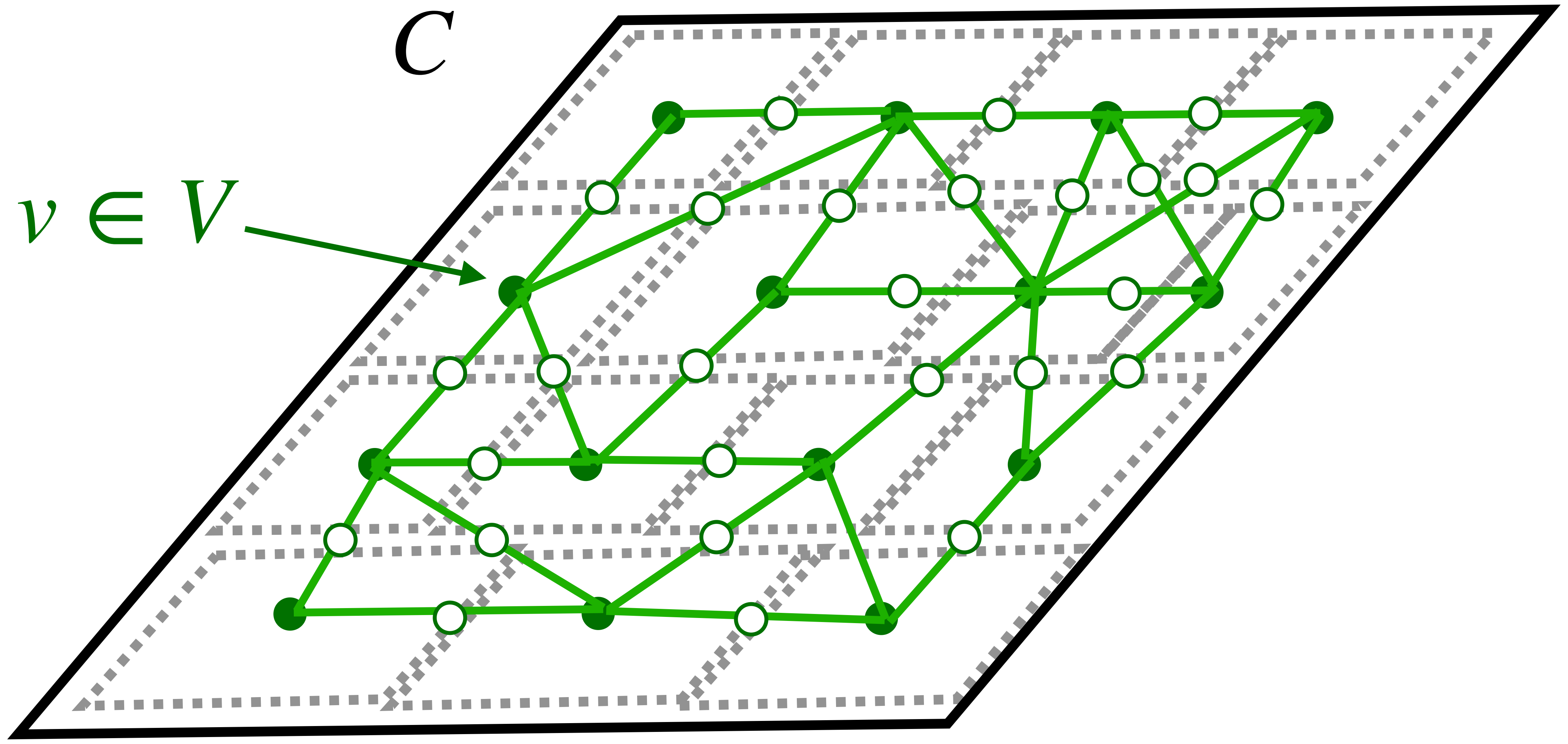
Tantivasadakarn, et al. PRX **14** (2024)
Christos, et al. [2602.12228]

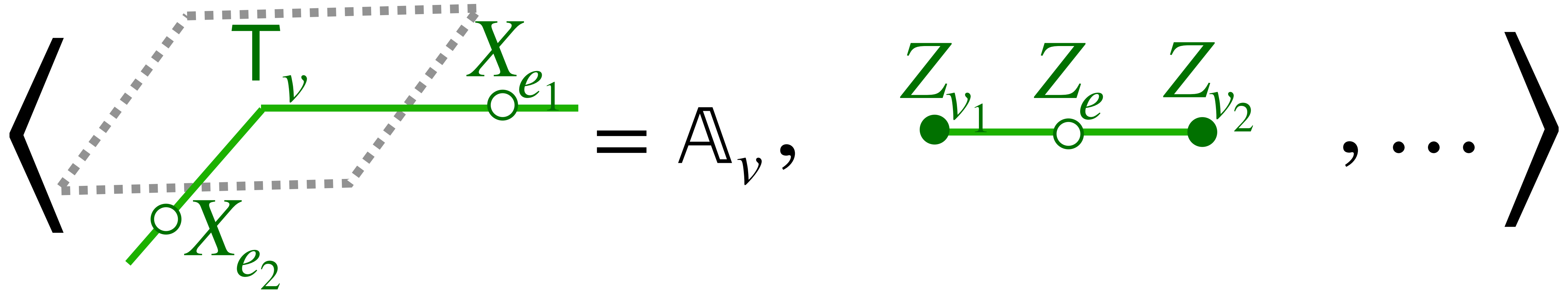
Constructing the gauged code

$$\mathsf{T} = \bigotimes_v \mathsf{T}_v \qquad |\mathsf{T}_v| = O(1)$$

Ex. $\overline{H}_{\mathrm{XY}} = H_{\mathrm{XY}}^{\otimes n}$ Single-qubit vertex sets

Ex. $\overline{CX}^{\otimes k}_{\mathrm{ctrl}\to\mathrm{tar}} = CX^{\otimes n}_{\mathrm{ctrl}\to\mathrm{tar}}$ Two-qubit vertex sets: (ctrl, tar) pair



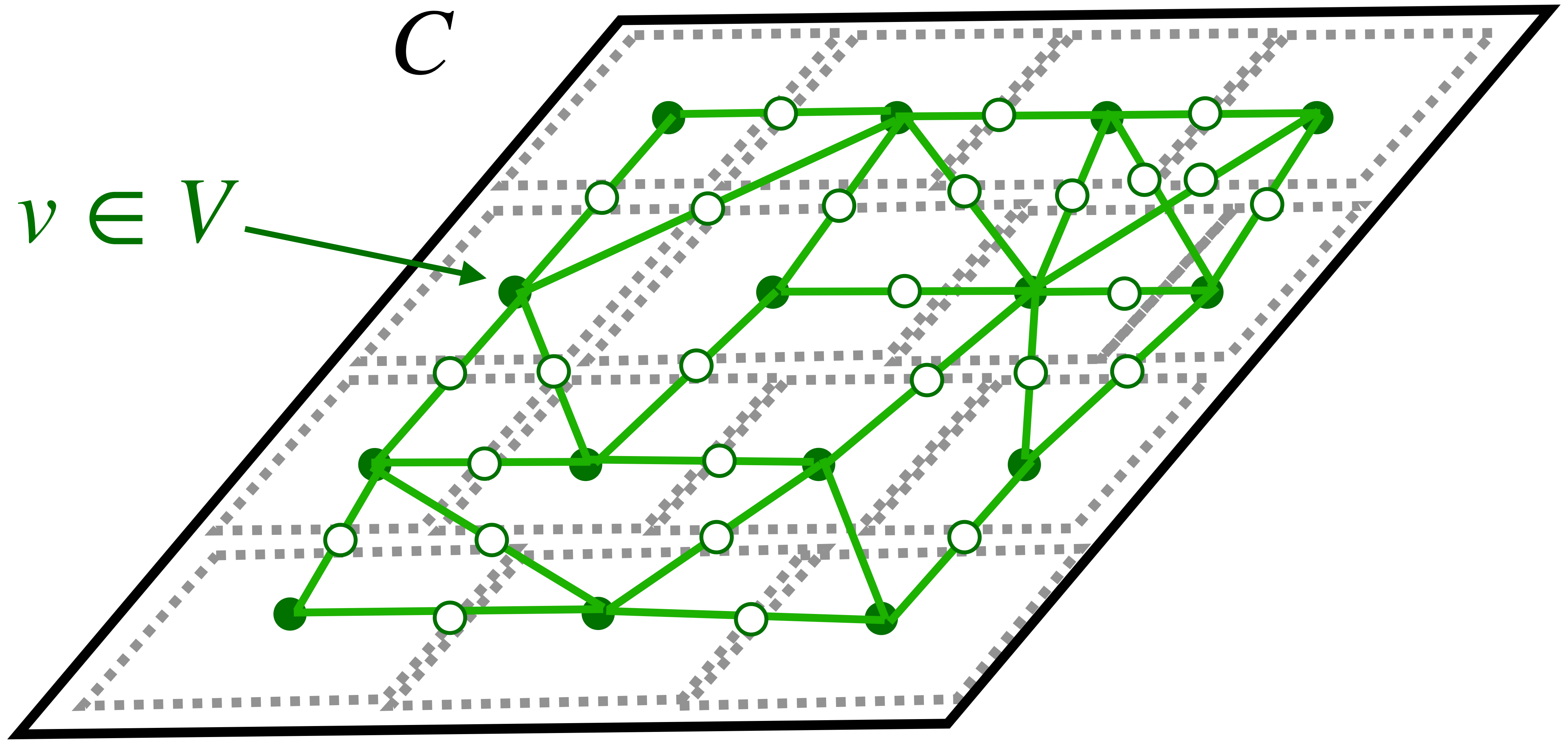


1. Partition data qubits into constant-sized vertex sets.
2. Construct a sufficiently expanding graph with |V| vertices and a sparse cycle basis [WY24].
3. For each vertex, initialize $|+\rangle$ & for each edge, initialize $|0\rangle$.
4. Apply $C_v\mathsf{T}_v$ for each v.
5. Apply C_vX_e for each $e \in \delta(v)$.
6. Measure all v in X.

$$\prod_v x_v = \mathsf{T}$$

Tantivasadakarn, et al. PRX **14** (2024)
Christos, et al. [2602.12228]

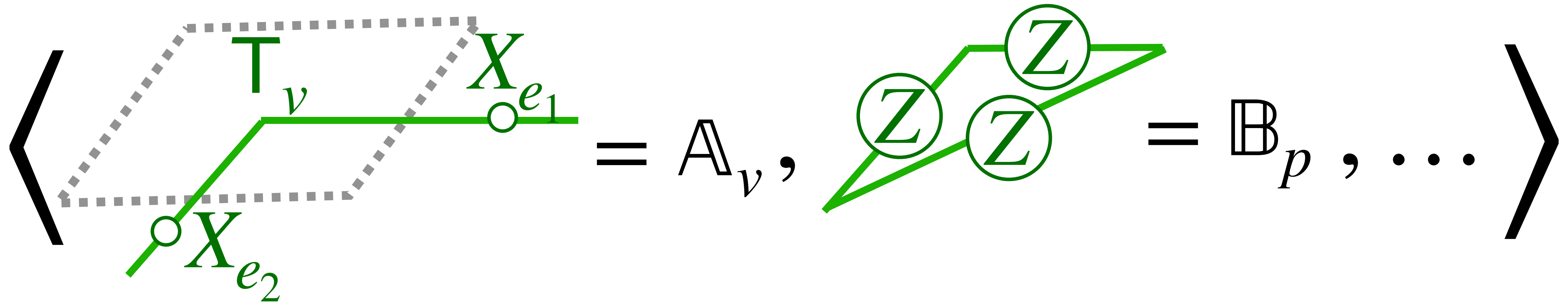
Constructing the gauged code



$$\mathsf{T} = \otimes_v \mathsf{T}_v \qquad |\mathsf{T}_v| = O(1)$$

Ex. $\overline{H}_{\mathrm{XY}} = H_{\mathrm{XY}}^{\otimes n}$ Single-qubit vertex sets

Ex. $\overline{CX}^{\otimes k}_{\mathrm{ctrl}\to\mathrm{tar}} = CX^{\otimes n}_{\mathrm{ctrl}\to\mathrm{tar}}$ Two-qubit vertex sets: (ctrl, tar) pair



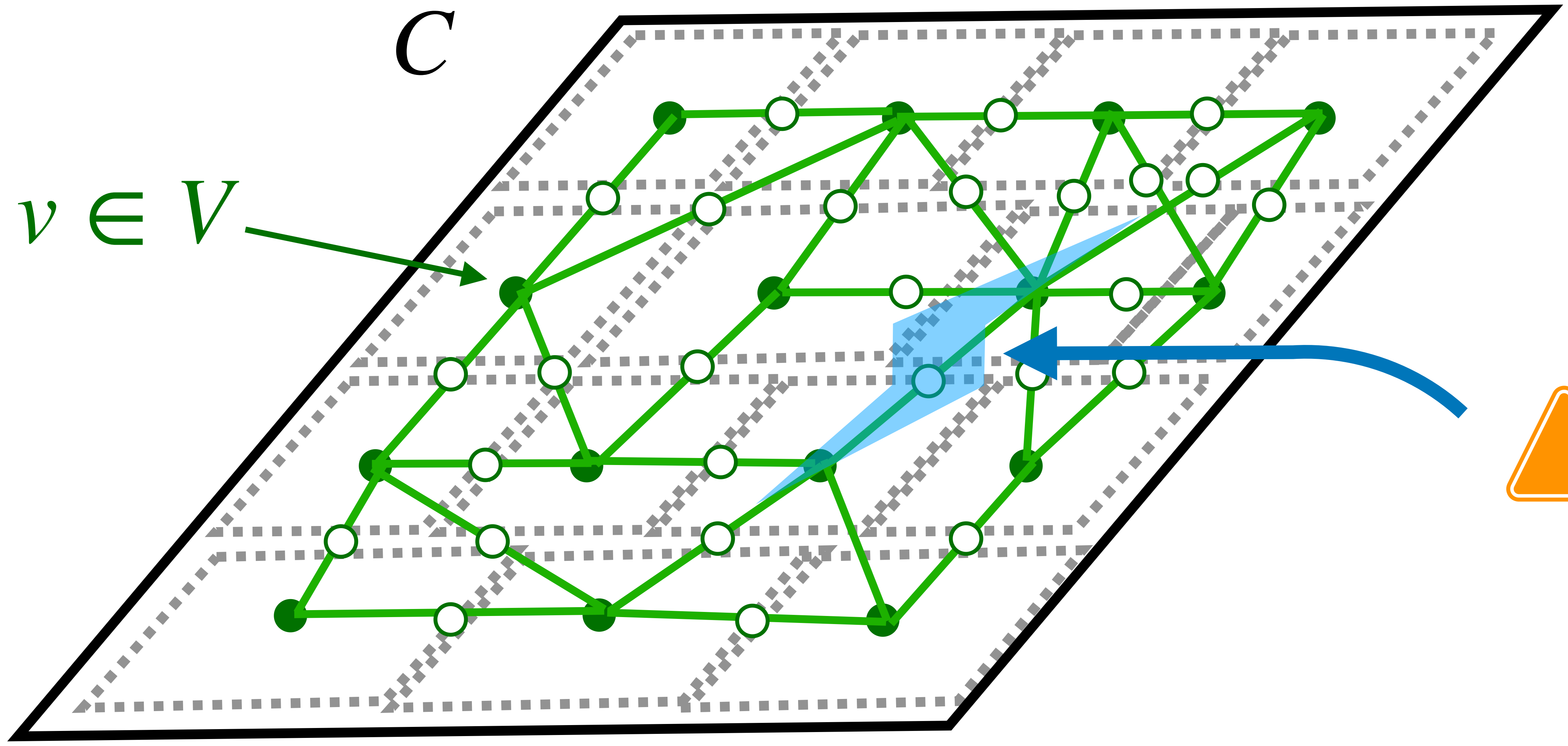
1. Partition data qubits into constant-sized vertex sets.
2. Construct a sufficiently expanding graph with |V| vertices and a sparse cycle basis [WY24].
3. For each vertex, initialize $|+\rangle$ & for each edge, initialize $|0\rangle$.
4. Apply $C_v\mathsf{T}_v$ for each v.
5. Apply C_vX_e for each $e \in \delta(v)$.
6. Measure all v in X.

$$\prod_v x_v = \mathsf{T}$$

Tantivasadakarn, et al. PRX **14** (2024)
Christos, et al. [2602.12228]

Constructing the gauged code

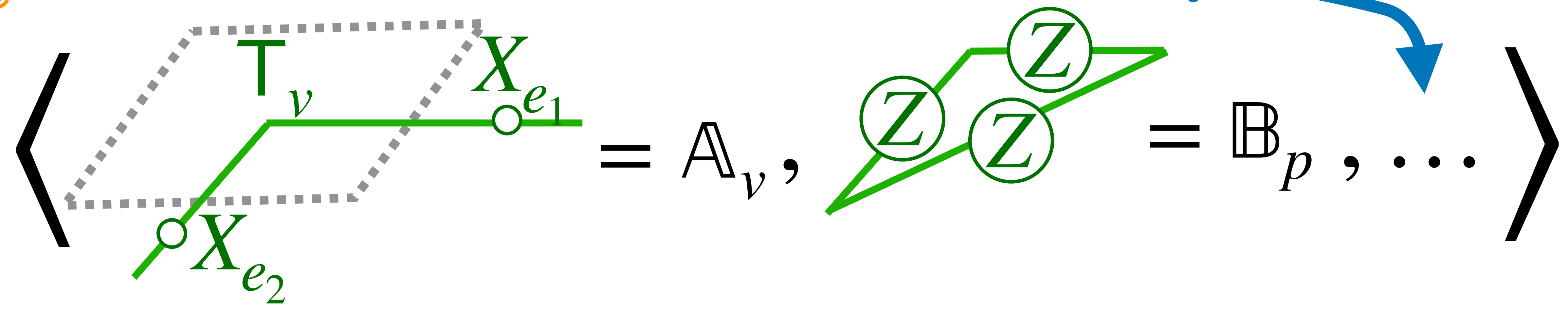
$\mathsf{T} = \otimes_v \mathsf{T}_v \qquad |\mathsf{T}_v| = O(1)$



Ex. $\overline{H}_{\mathrm{XY}} = H_{\mathrm{XY}}^{\otimes n}$ Single-qubit vertex sets

Ex. $\overline{CX}^{\otimes k}_{\mathrm{ctrl}\to\mathrm{tar}} = CX^{\otimes n}_{\mathrm{ctrl}\to\mathrm{tar}}$ Two-qubit vertex sets: (ctrl, tar) pair

A stabilizer of C deforms non-trivially.



1. Partition data qubits into constant-sized vertex sets.
2. Construct a sufficiently expanding graph with |V| vertices and a sparse cycle basis [WY24].
3. For each vertex, initialize $|+\rangle$ & for each edge, initialize $|0\rangle$.
4. Apply $C_v\mathsf{T}_v$ for each v.
5. Apply C_vX_e for each $e \in \delta(v)$.
6. Measure all v in X.

$$\prod_v x_v = \mathsf{T}$$

Tantivasadakarn, et al. PRX **14** (2024)
Christos, et al. [2602.12228]

Properties of the gauged code

Properties of the gauged code

- [x] Some product of stabilizers of C_g should equal the Clifford logical, T. $\left(\prod_v \mathbb{A}_v = \mathsf{T}\right)$

Properties of the gauged code

- ☑ Some product of stabilizers of C_{g} should equal the Clifford logical, T. $(\prod_{v} \mathbb{A}_v = \mathsf{T})$
- ☑ LDPC

Properties of the gauged code

- ☑ Some product of stabilizers of C_{g} should equal the Clifford logical, T. *($\prod_{v} \mathbb{A}_v = \mathsf{T}$)*
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*

Properties of the gauged code

- ☑ Some product of stabilizers of C_{g} should equal the Clifford logical, T. *($\prod_v \mathbb{A}_v = \mathsf{T}$)*
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*

Properties of the gauged code

- ☑ Some product of stabilizers of C_{g} should equal the Clifford logical, T. *($\prod_{v} \mathbb{A}_v = \mathsf{T}$)*
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*
- ☑ Any logical representative of C_{g} is supported on at least d qubits *(good spatial distance)*

Properties of the gauged code

- ☑ Some product of stabilizers of C_{g} should equal the Clifford logical, T. *($\prod_v \mathbb{A}_v = \mathsf{T}$)*
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*
- ☑ Any logical representative of C_{g} is supported on at least d qubits *(good spatial distance)*

Stabilizers commute, but only in the codespace

Properties of the gauged code

- ☑ Some product of stabilizers of C_g should equal the Clifford logical, T. $(\prod_v \mathbb{A}_v = \mathsf{T})$
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*
- ☑ Any logical representative of C_g is supported on at least d qubits *(good spatial distance)*

Stabilizers commute, but only in the codespace

Ex. Stabilizers of BB code when gauging transversal $\overline{CX}$

Properties of the gauged code

- ☑ Some product of stabilizers of C_{g} should equal the Clifford logical, T. $\left(\prod_{v} \mathbb{A}_v = \mathsf{T}\right)$
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*
- ☑ Any logical representative of C_{g} is supported on at least d qubits *(good spatial distance)*

Stabilizers commute, but only in the codespace

Ex. Stabilizers of BB code when gauging transversal $\overline{CX}$

A decorated X-stabilizer of the control BB code

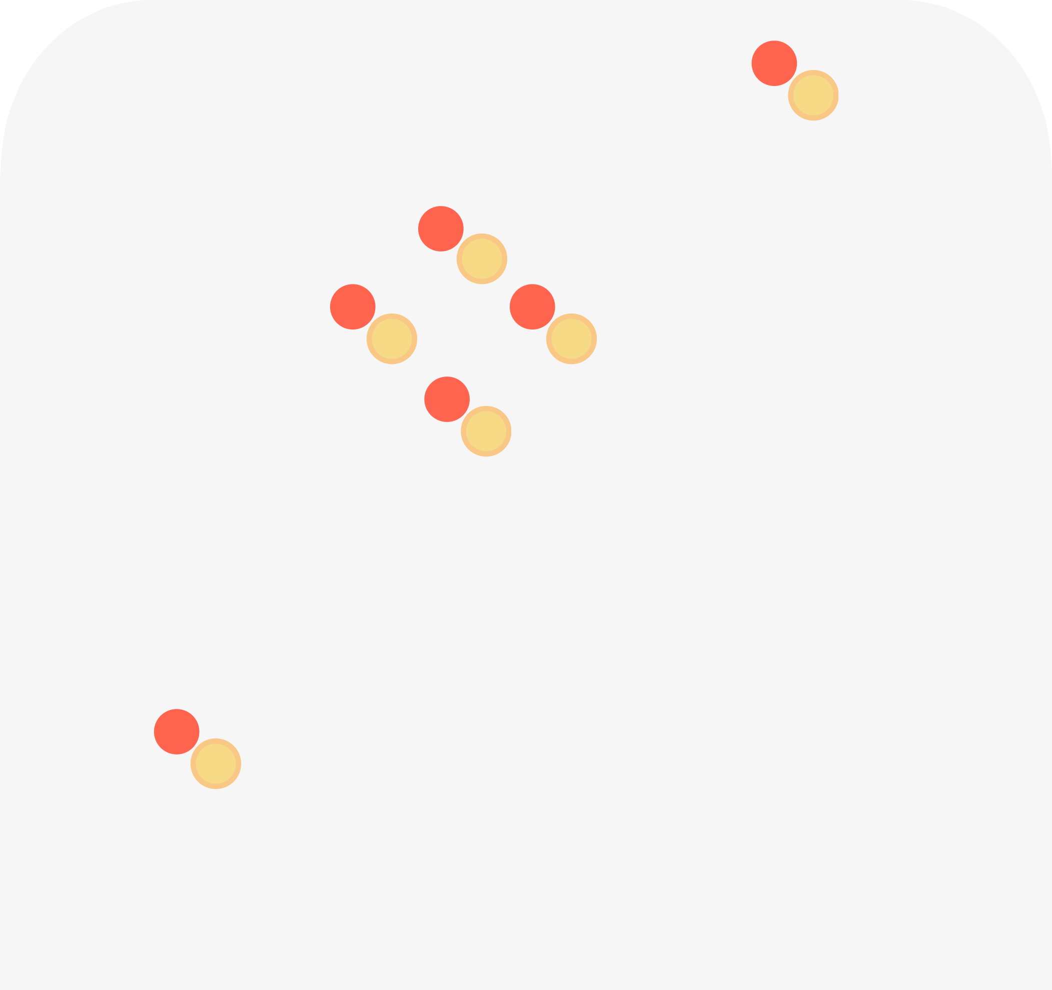
Properties of the gauged code

- [x] Some product of stabilizers of C_g should equal the Clifford logical, T. *($\prod_v \mathbb{A}_v = \mathsf{T}$)*
- [x] LDPC
- [x] Stabilizers are unitary *(potentially non-Pauli)*
- [x] Stabilizers commute *(in some eigenspace)*
- [x] Any logical representative of C_g is supported on at least d qubits *(good spatial distance)*

Stabilizers commute, but only in the codespace

Ex. Stabilizers of BB code when gauging transversal $\overline{CX}$

- Qubits of control BB codeblock
- Qubits of target BB codeblock



A decorated X-stabilizer of the control BB code

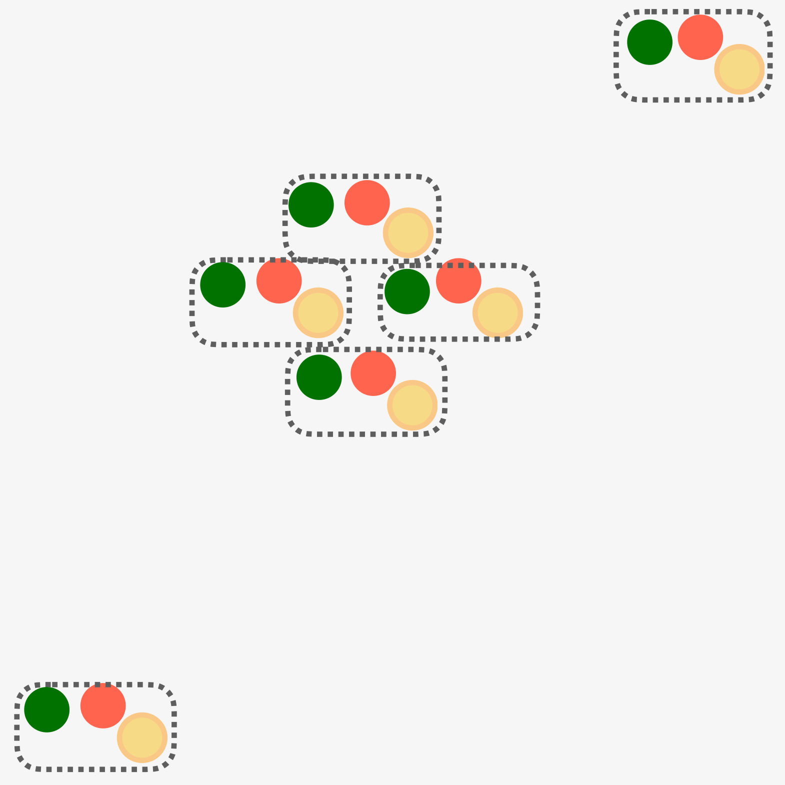
Properties of the gauged code

- ☑ Some product of stabilizers of C_g should equal the Clifford logical, T. $\left(\prod_v \mathbb{A}_v = \mathsf{T}\right)$
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*
- ☑ Any logical representative of C_g is supported on at least d qubits *(good spatial distance)*

Stabilizers commute, but only in the codespace

Ex. Stabilizers of BB code when gauging transversal $\overline{CX}$

- Qubits of control BB codeblock
- Qubits of target BB codeblock
- Gauging graph vertices



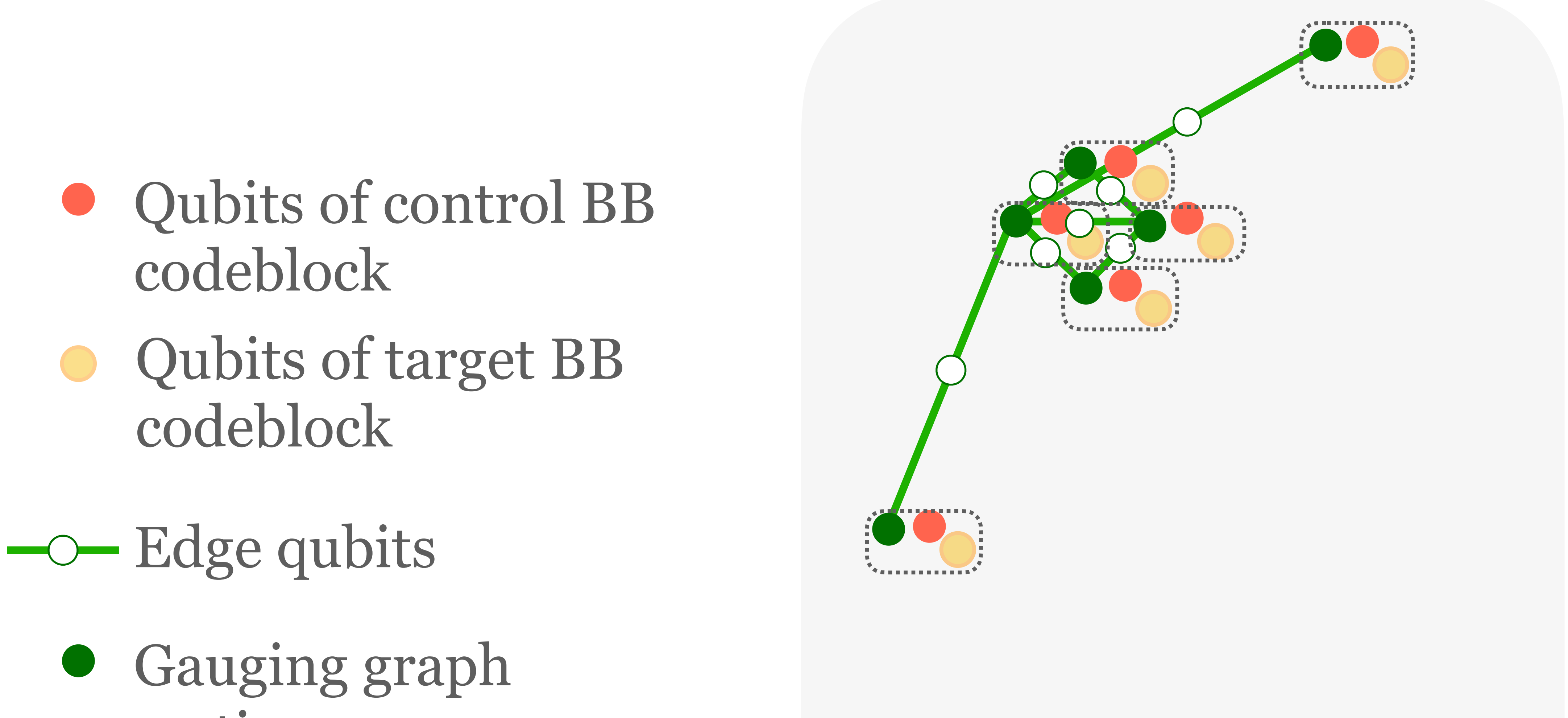
A decorated X-stabilizer of the control BB code

Properties of the gauged code

- ☑ Some product of stabilizers of C_{g} should equal the Clifford logical, T. $(\prod_v \mathbb{A}_v = \mathsf{T})$
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*
- ☑ Any logical representative of C_{g} is supported on at least d qubits *(good spatial distance)*

Stabilizers commute, but only in the codespace

Ex. Stabilizers of BB code when gauging transversal $\overline{CX}$



A decorated X-stabilizer of the control BB code

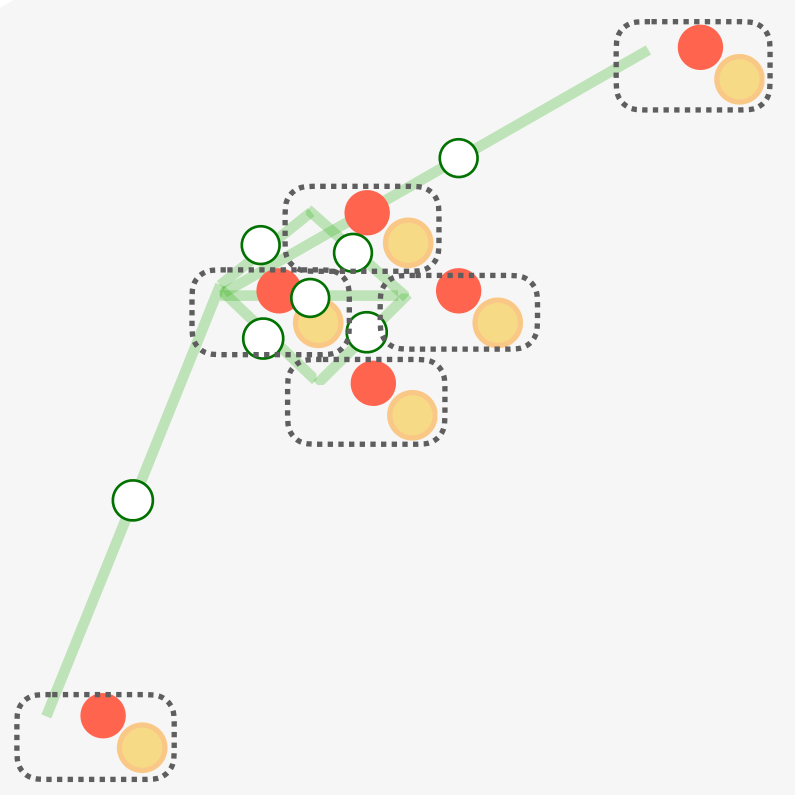
Properties of the gauged code

- ☑ Some product of stabilizers of C_g should equal the Clifford logical, T. $(\prod_v \mathbb{A}_v = \mathsf{T})$
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*
- ☑ Any logical representative of C_g is supported on at least d qubits *(good spatial distance)*

Stabilizers commute, but only in the codespace

Ex. Stabilizers of BB code when gauging transversal $\overline{CX}$

- Qubits of control BB codeblock
- Qubits of target BB codeblock
- Edge qubits
- Gauging graph vertices



A decorated X-stabilizer of the control BB code

Properties of the gauged code

- ☑ Some product of stabilizers of C_g should equal the Clifford logical, T. $(\prod_v \mathbb{A}_v = \mathsf{T})$
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*
- ☑ Any logical representative of C_g is supported on at least d qubits *(good spatial distance)*

Stabilizers commute, but only in the codespace

Ex. Stabilizers of BB code when gauging transversal $\overline{CX}$

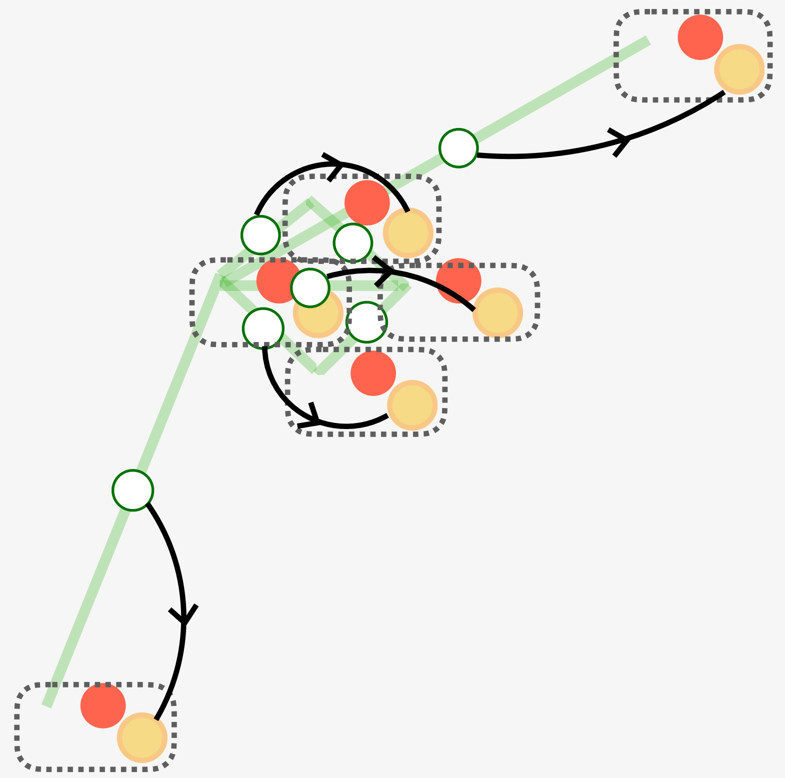
→ Physical CX

● Qubits of control BB codeblock

● Qubits of target BB codeblock

—○— Edge qubits

● Gauging graph vertices



$$\mathbb{S} = \prod_{\bullet} X_{\bullet} \prod_{\to} C_{\circ} X_{\bullet}$$

A decorated X-stabilizer of the control BB code

Properties of the gauged code

- ☑ Some product of stabilizers of C_g should equal the Clifford logical, T. $(\prod_v \mathbb{A}_v = \mathsf{T})$
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*
- ☑ Any logical representative of C_g is supported on at least d qubits *(good spatial distance)*

Stabilizers commute, but only in the codespace

Ex. Stabilizers of BB code when gauging transversal $\overline{CX}$

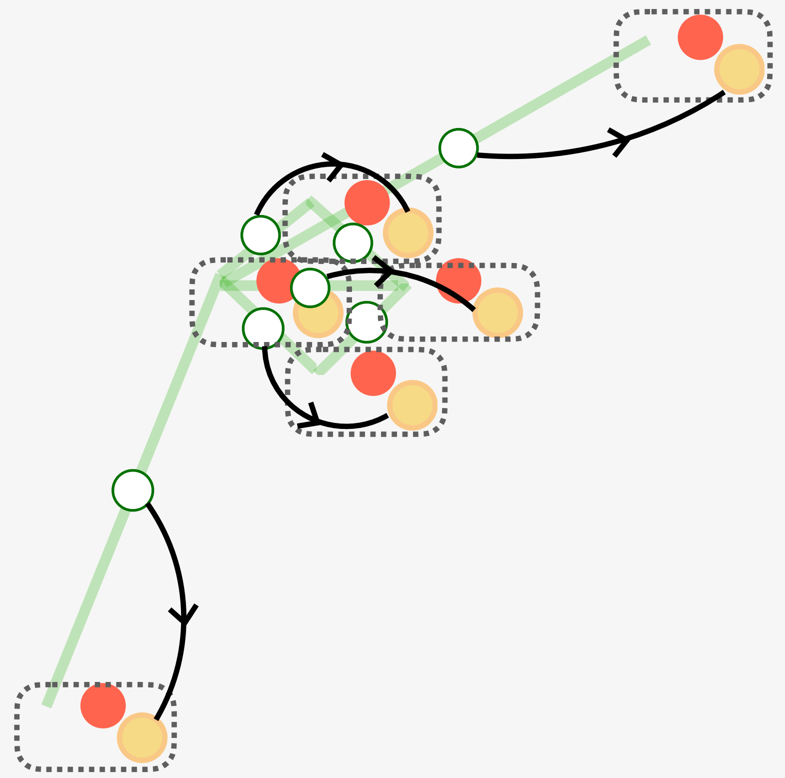
→ Physical CX

● Qubits of control BB codeblock

● Qubits of target BB codeblock

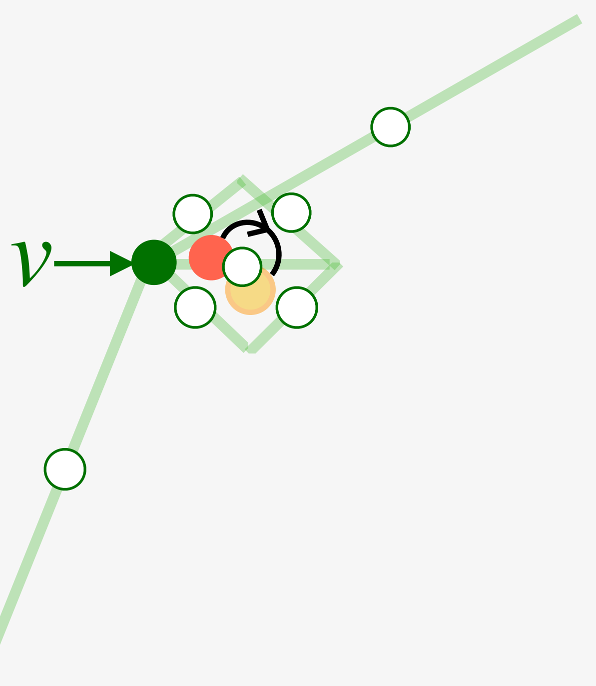
—○— Edge qubits

● Gauging graph vertices



$$\mathbb{S} = \prod_{\bullet} X_{\bullet} \prod_{\rightarrow} C_{\circ} X_{\bullet}$$

A decorated X-stabilizer of the control BB code



$$\mathbb{A}_v = C_{\bullet} X_{\bullet} \prod_{\circ \in \delta(v)} X_{\circ}$$

A vertex stabilizer to measure logical CX

Properties of the gauged code

- ☑ Some product of stabilizers of C_{g} should equal the Clifford logical, T. $\left(\prod_{v} \mathbb{A}_v = \mathsf{T}\right)$
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*
- ☑ Any logical representative of C_{g} is supported on at least d qubits *(good spatial distance)*

Stabilizers commute, but only in the codespace

Ex. Stabilizers of BB code when gauging transversal $\overline{CX}$

$$[\![A, B]\!] := ABA^{-1}B^{-1} = I \text{ if } A, B \text{ commute}$$

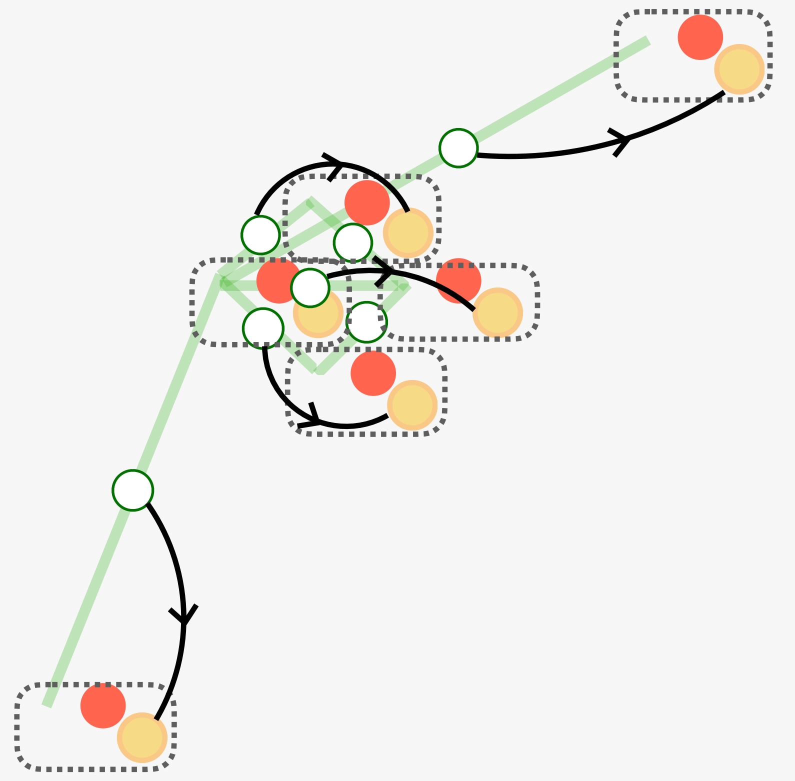
$\rightarrow$ Physical CX

● Qubits of control BB codeblock

● Qubits of target BB codeblock

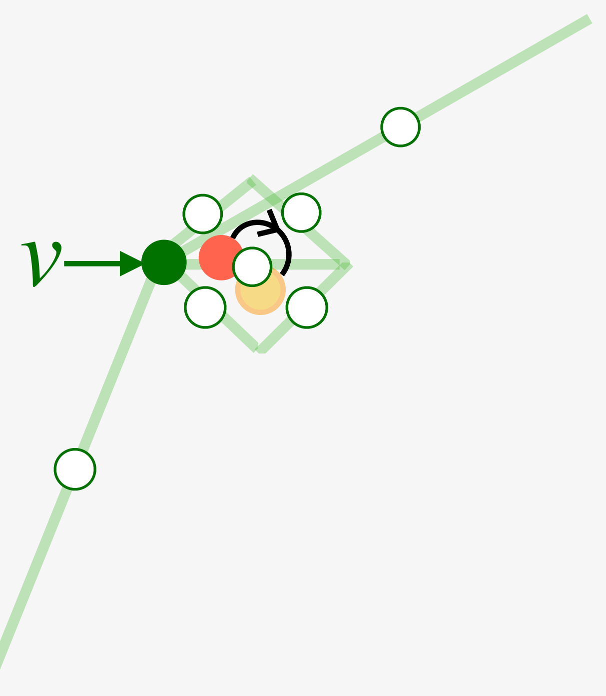
—○— Edge qubits

● Gauging graph vertices



$$\mathbb{S} = \prod_{\bullet} X_{\bullet} \prod_{\rightarrow} C_{\circ} X_{\bullet}$$

A decorated X-stabilizer of the control BB code



$$\mathbb{A}_v = C_{\bullet} X_{\bullet} \prod_{\circ \in \delta(v)} X_{\circ}$$

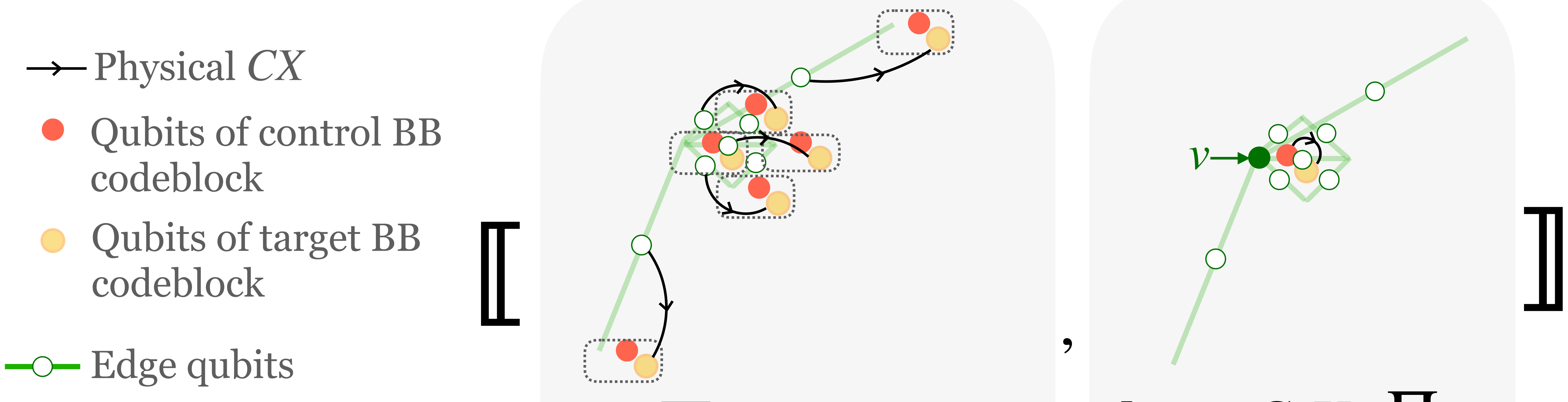
A vertex stabilizer to measure logical CX

Properties of the gauged code

- ☑ Some product of stabilizers of C_g should equal the Clifford logical, T. $(\prod_v \mathbb{A}_v = \mathsf{T})$
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*
- ☑ Any logical representative of C_g is supported on at least d qubits *(good spatial distance)*

Stabilizers commute, but only in the codespace

Ex. Stabilizers of BB code when gauging transversal $\overline{CX}$ $\quad [\![A, B]\!] := ABA^{-1}B^{-1} = I$ if A, B commute



$$[\![\ \mathbb{S} = \prod_{\color{red}\bullet} X_{\color{red}\bullet} \prod_{\to} C_{\circ} X_{\color{orange}\bullet}\ ,\ \mathbb{A}_v = C_{\color{red}\bullet} X_{\color{orange}\bullet} \prod_{\circ \in \delta(v)} X_{\circ}\ ]\!]$$

A decorated X-stabilizer of the control BB code

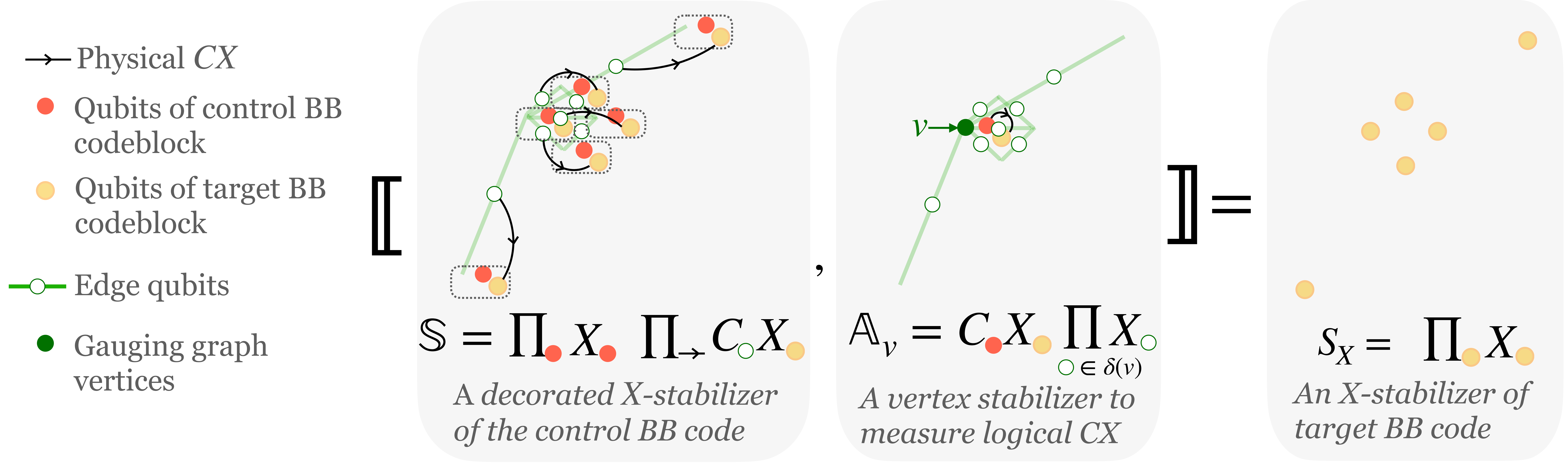
A vertex stabilizer to measure logical CX

Properties of the gauged code

- ☑ Some product of stabilizers of C_g should equal the Clifford logical, T. *($\prod_v \mathbb{A}_v = \mathsf{T}$)*
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*
- ☑ Any logical representative of C_g is supported on at least d qubits *(good spatial distance)*

Stabilizers commute, but only in the codespace

Ex. Stabilizers of BB code when gauging transversal $\overline{CX}$ $[\![A, B]\!] := ABA^{-1}B^{-1}$ $= I$ if A, B commute

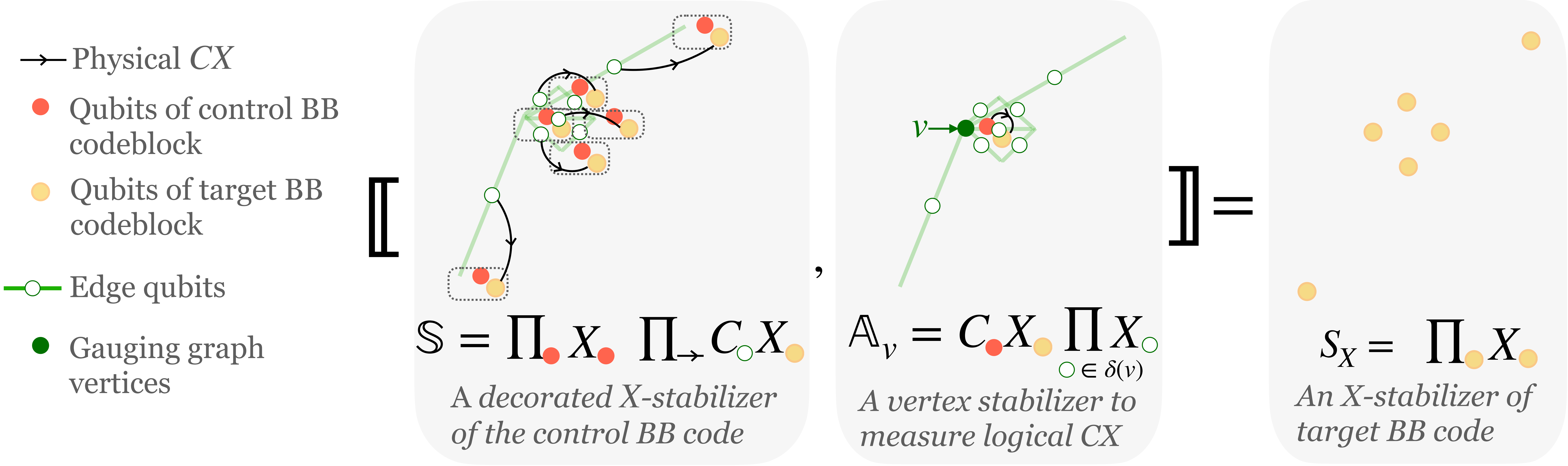


Properties of the gauged code

- ☑ Some product of stabilizers of C_g should equal the Clifford logical, T. ($\prod_v \mathbb{A}_v = \mathsf{T}$)
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*
- ☑ Any logical representative of C_g is supported on at least d qubits *(good spatial distance)*

Stabilizers commute, but only in the codespace

Ex. Stabilizers of BB code when gauging transversal $\overline{CX}$ $[\![A, B]\!] := ABA^{-1}B^{-1}$ = I if A, B commute



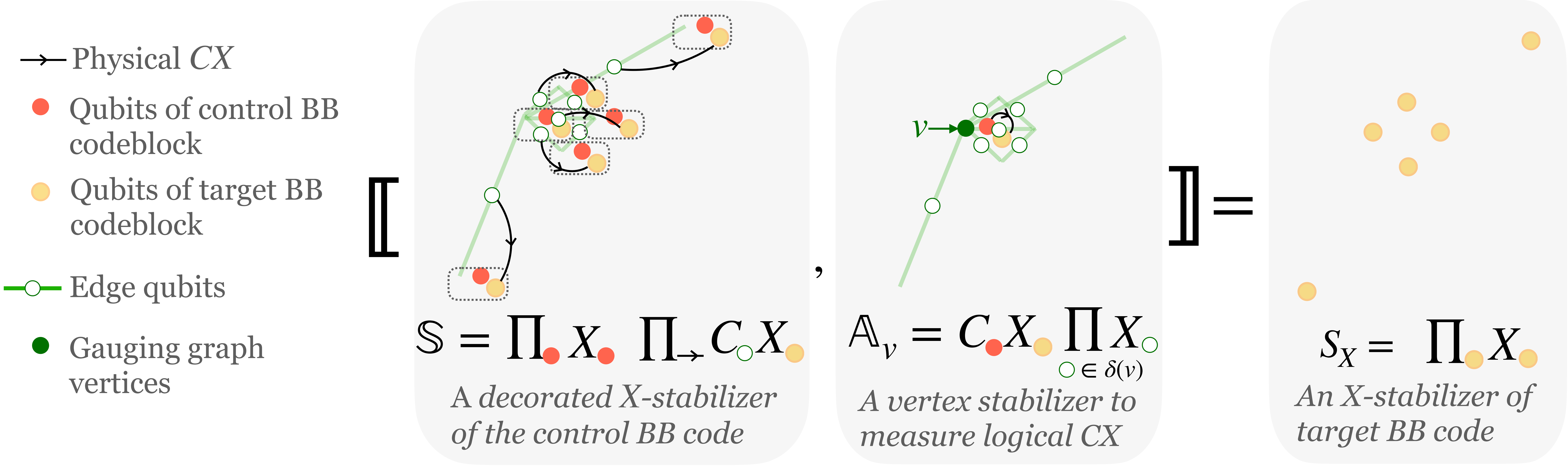
Error → -1 eigenspace of $S_X \to \mathbb{S}, \mathbb{A}_v$ stabilizers do not commute

Properties of the gauged code

- ☑ Some product of stabilizers of C_{g} should equal the Clifford logical, T. ($\prod_v \mathbb{A}_v = \mathsf{T}$)
- ☑ LDPC
- ☑ Stabilizers are unitary *(potentially non-Pauli)*
- ☑ Stabilizers commute *(in some eigenspace)*
- ☑ Any logical representative of C_{g} is supported on at least d qubits *(good spatial distance)*

Stabilizers commute, but only in the codespace

Ex. Stabilizers of BB code when gauging transversal $\overline{CX}$ $[\![A, B]\!] := ABA^{-1}B^{-1}$ $= I$ if A, B commute



Error → -1 eigenspace of $S_X \to \mathbb{S}, \mathbb{A}_v$ stabilizers do not commute

$n_{C_{\mathrm{g}}} - n_C = \tilde{O}(n_C)$

Changes to fault-tolerance in the gauged code

Changes to fault-tolerance in the gauged code

In contrast to normal, Pauli stabilized codes, **detectors** can now be **randomized from errors**!

Ex. $[\![\mathbb{S}, \mathbb{A}_v]\!] = S_x$

Changes to fault-tolerance in the gauged code

In contrast to normal, Pauli stabilized codes, **detectors** can now be **randomized from errors**!

Ex. $[\![\mathbb{S}, \ \mathbb{A}_v]\!] = S_x$

Regular detectors : flip near anticommuting Pauli/measurement error.

Dropped detectors : irregardless of faults, returns random outcomes.

Changes to fault-tolerance in the gauged code

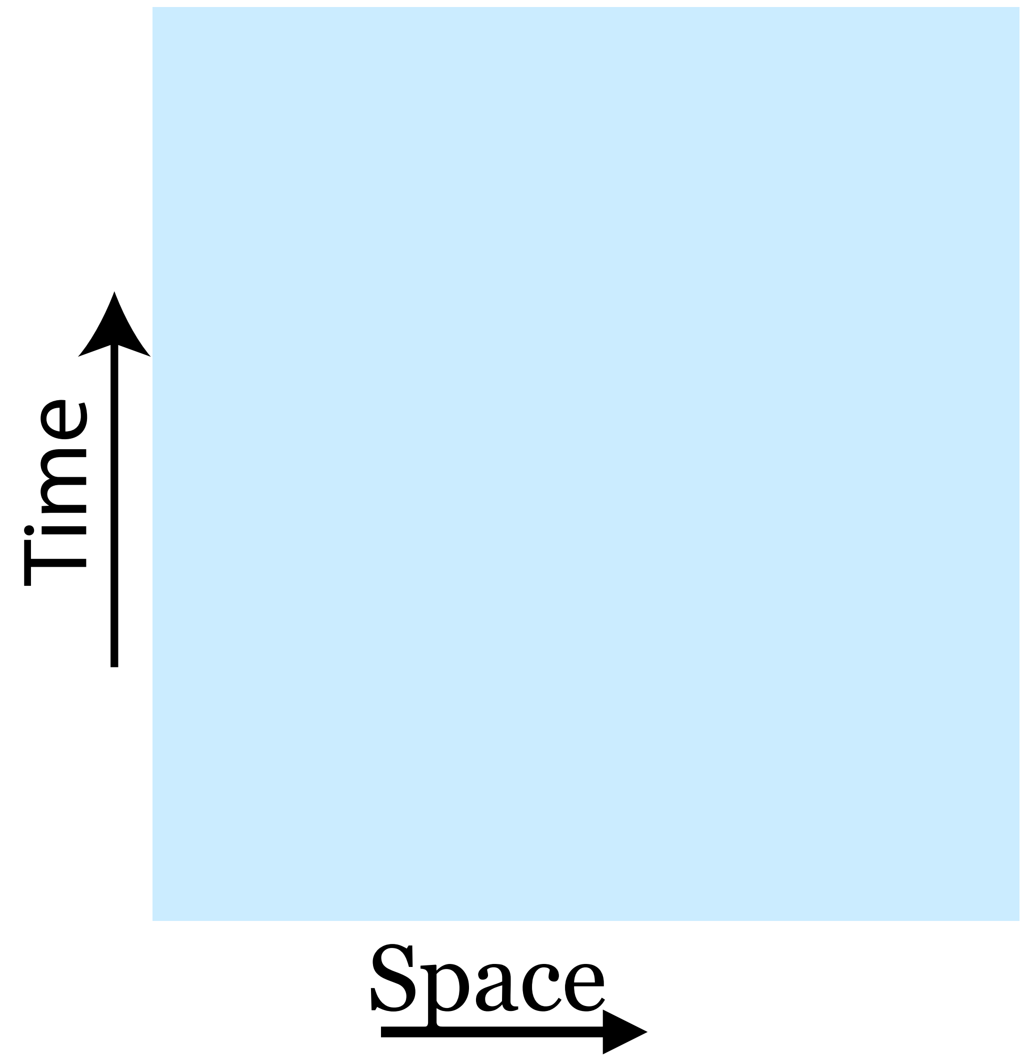
In contrast to normal, Pauli stabilized codes, **detectors** can now be **randomized from errors**!

Regular detectors : flip near anticommuting Pauli/measurement error.

Dropped detectors : irregardless of faults, returns random outcomes.

Ex. $[\![\mathbb{S}, \mathbb{A}_v]\!] = S_x$

Spacetime volume



Changes to fault-tolerance in the gauged code

In contrast to normal, Pauli stabilized codes, **detectors** can now be **randomized from errors**!

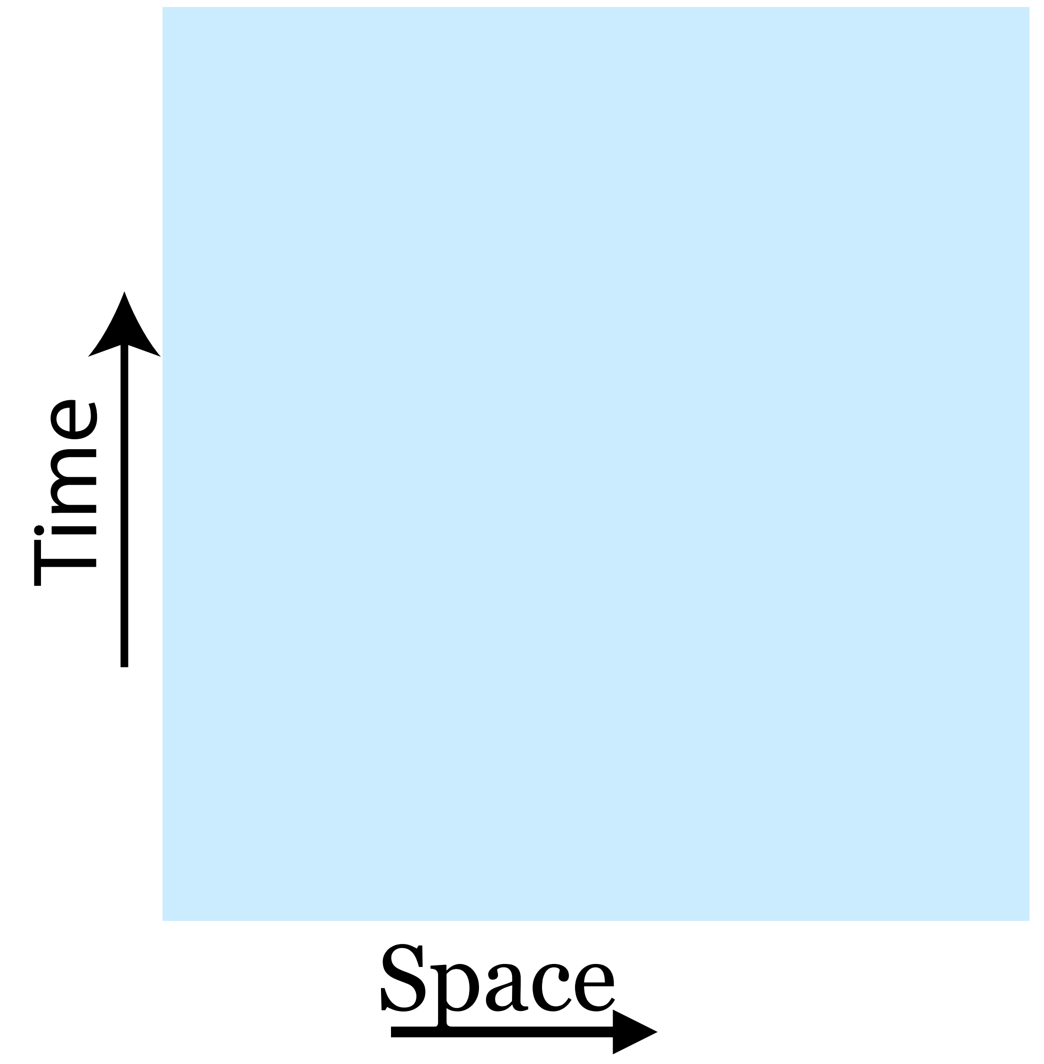
Regular detectors : flip near anticommuting Pauli/measurement error.

Dropped detectors : irregardless of faults, returns random outcomes.

To show **large distance to Pauli and measurement errors**, we use the following convenient observations:

Ex. $[\![\mathbb{S}, \mathbb{A}_v]\!] = S_x$

Spacetime volume



Changes to fault-tolerance in the gauged code

In contrast to normal, Pauli stabilized codes, **detectors** can now be **randomized from errors**!

Regular detectors : flip near anticommuting Pauli/measurement error.

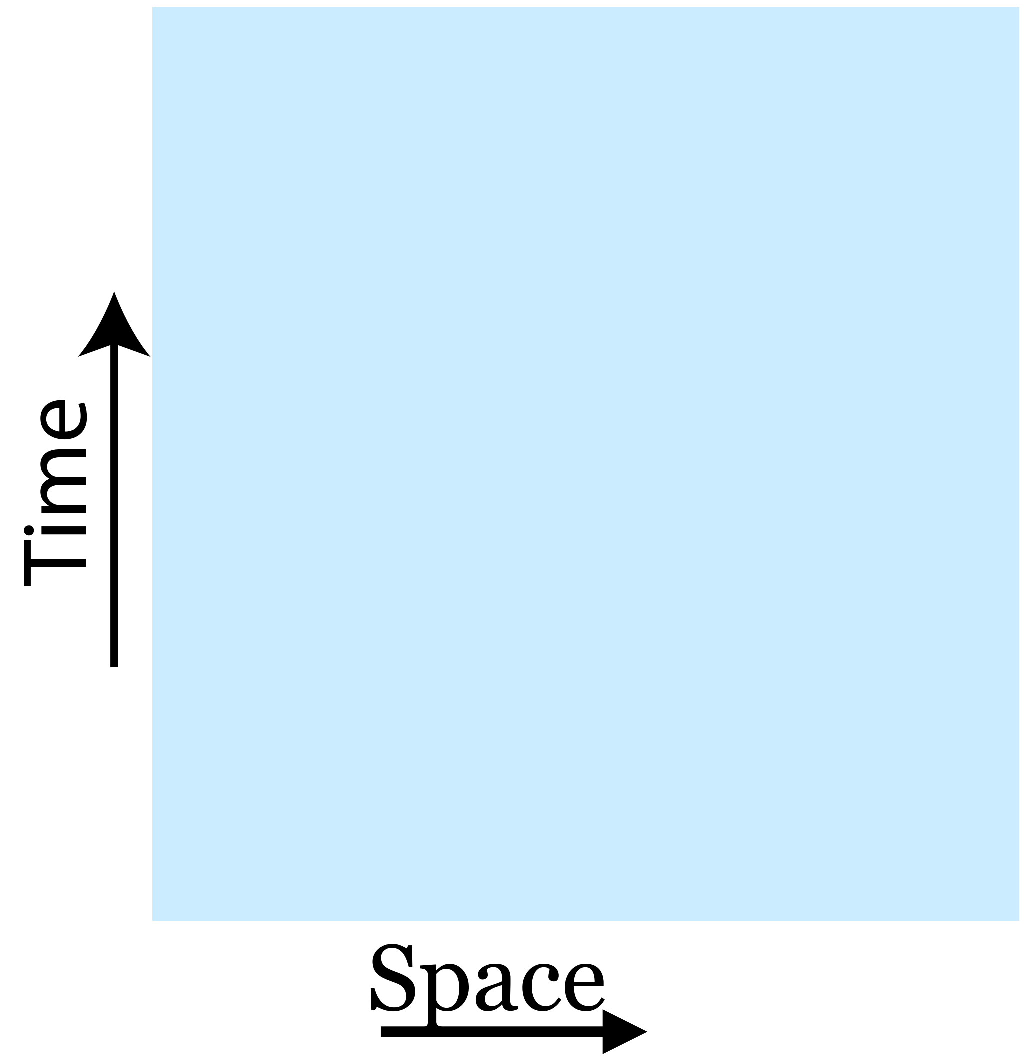
Dropped detectors : irregardless of faults, returns random outcomes.

To show **large distance to Pauli and measurement errors**, we use the following convenient observations:

When Clifford T is **order-2, stabilizer-randomizing errors** are **detected by stabilizers** that are **never random.**

Ex. $[\![\mathbb{S}, \mathbb{A}_v]\!] = S_x$

Spacetime volume



Changes to fault-tolerance in the gauged code

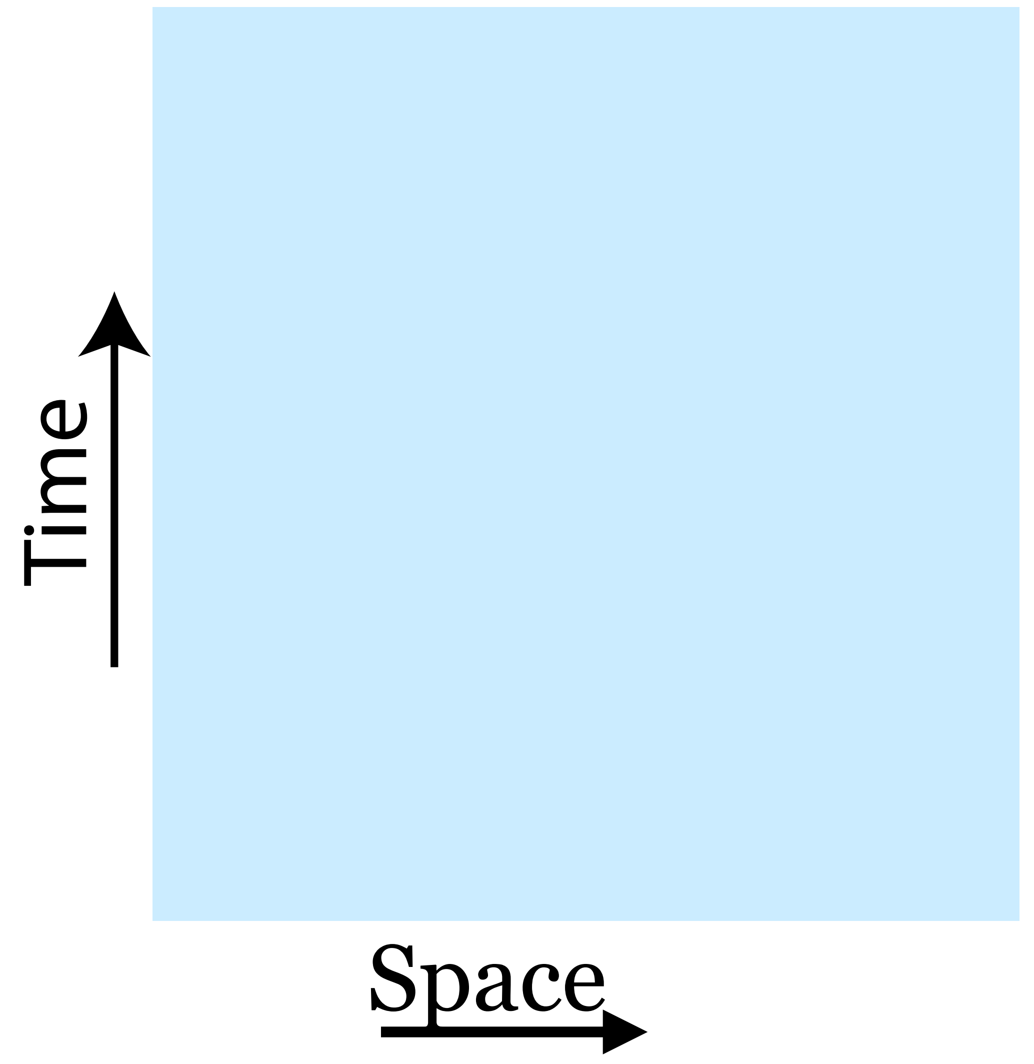
In contrast to normal, Pauli stabilized codes, **detectors** can now be **randomized from errors**!

Regular detectors : flip near anticommuting Pauli/measurement error.

Dropped detectors : irregardless of faults, returns random outcomes.

Ex. $[\![\mathbb{S}, \mathbb{A}_v]\!] = S_x$

Spacetime volume



To show **large distance to Pauli and measurement errors**, we use the following convenient observations:

When Clifford T is **order-2, stabilizer-randomizing errors** are **detected by stabilizers** that are **never random.**

+ LDPC-ness

Changes to fault-tolerance in the gauged code

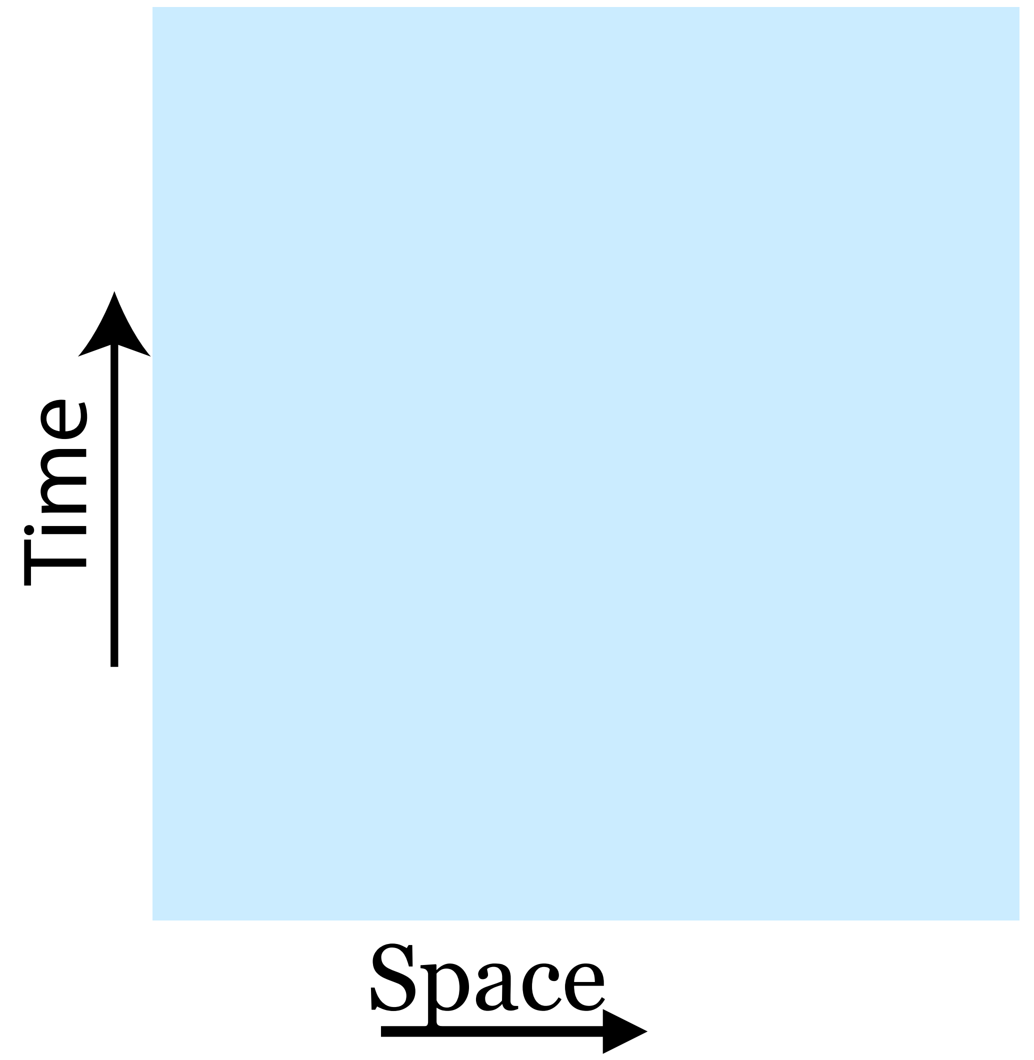
In contrast to normal, Pauli stabilized codes, **detectors** can now be **randomized from errors**!

Regular detectors : flip near anticommuting Pauli/measurement error.

Dropped detectors : irregardless of faults, returns random outcomes.

Ex. $[\![\mathbb{S}, \mathbb{A}_v]\!] = S_x$

Spacetime volume



To show **large distance to Pauli and measurement errors**,
we use the following convenient observations:

When Clifford T is **order-2, stabilizer-randomizing errors** are **detected by stabilizers** that are **never random.**

\+ **LDPC**-ness

Randomized stabilizers are in some **bounded spacetime radius** around an *undetectable* fault.

Changes to fault-tolerance in the gauged code

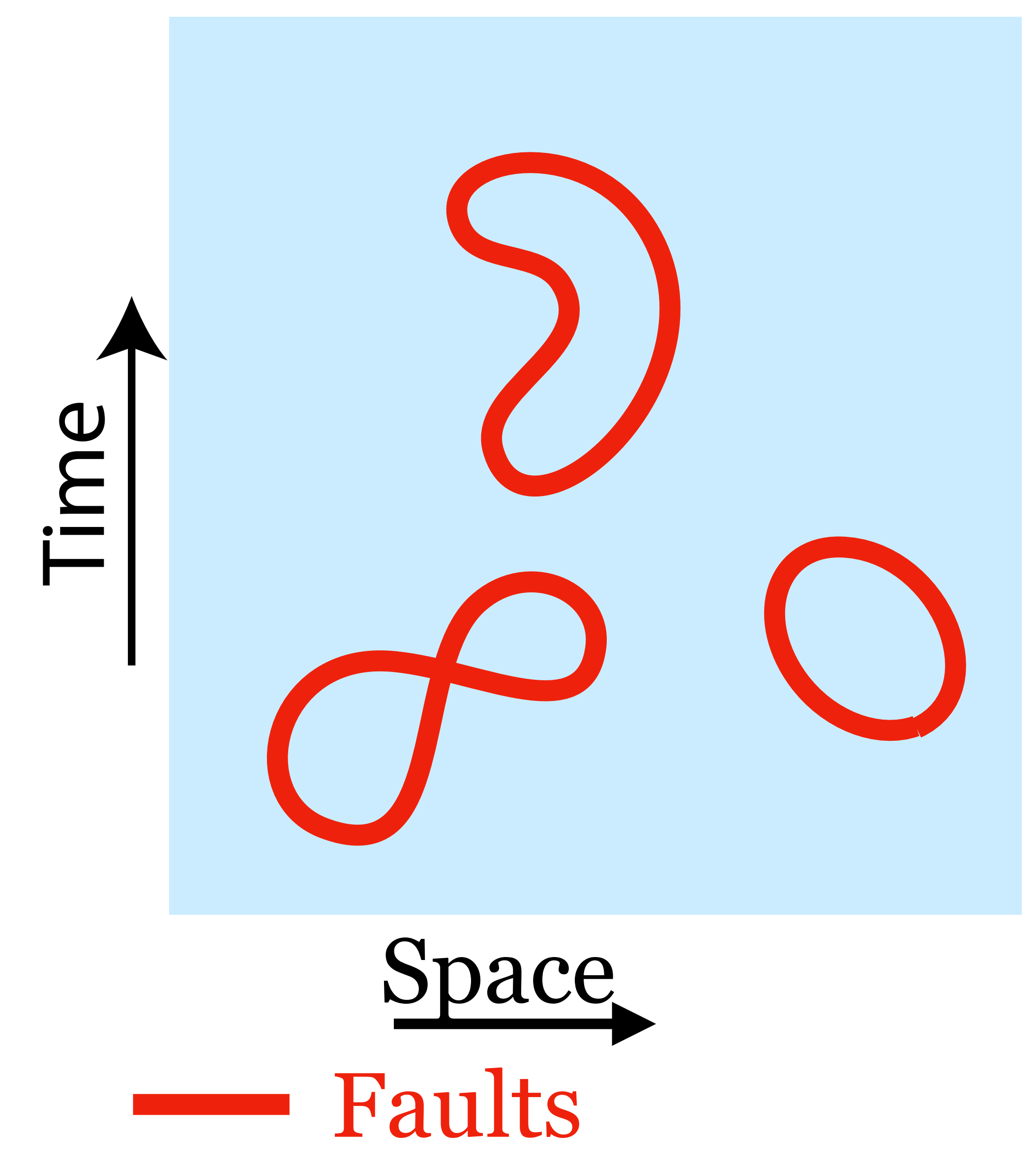
In contrast to normal, Pauli stabilized codes, **detectors** can now be **randomized from errors**!

Regular detectors : flip near anticommuting Pauli/measurement error.

Dropped detectors : irregardless of faults, returns random outcomes.

Ex. $[\![\mathbb{S}, \mathbb{A}_v]\!] = S_x$

Spacetime volume



To show **large distance to Pauli and measurement errors**, we use the following convenient observations:

When Clifford T is **order-2, stabilizer-randomizing errors** are **detected by stabilizers** that are **never random.**

\+ **LDPC**-ness

Randomized stabilizers are in some **bounded spacetime radius** around an *undetectable* fault.

Changes to fault-tolerance in the gauged code

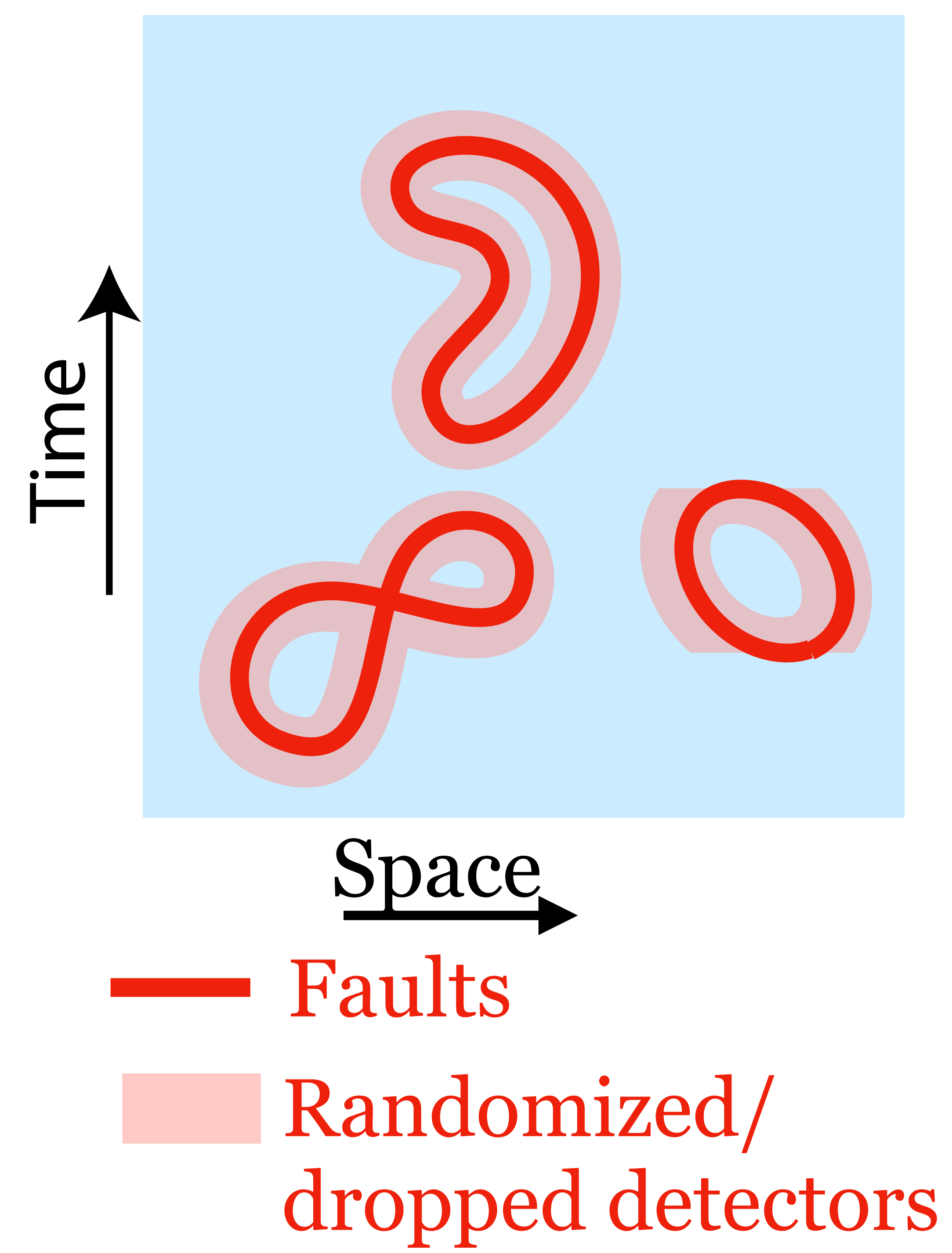
In contrast to normal, Pauli stabilized codes, **detectors** can now be **randomized from errors**!

Regular detectors : flip near anticommuting Pauli/measurement error.

Dropped detectors : irregardless of faults, returns random outcomes.

Ex. $[\![\mathbb{S}, \mathbb{A}_v]\!] = S_x$

Spacetime volume



To show **large distance to Pauli and measurement errors**, we use the following convenient observations:

When Clifford T is **order-2, stabilizer-randomizing errors** are **detected by stabilizers** that are **never random.**

+ LDPC-ness

Randomized stabilizers are in some **bounded spacetime radius** around an *undetectable* fault.

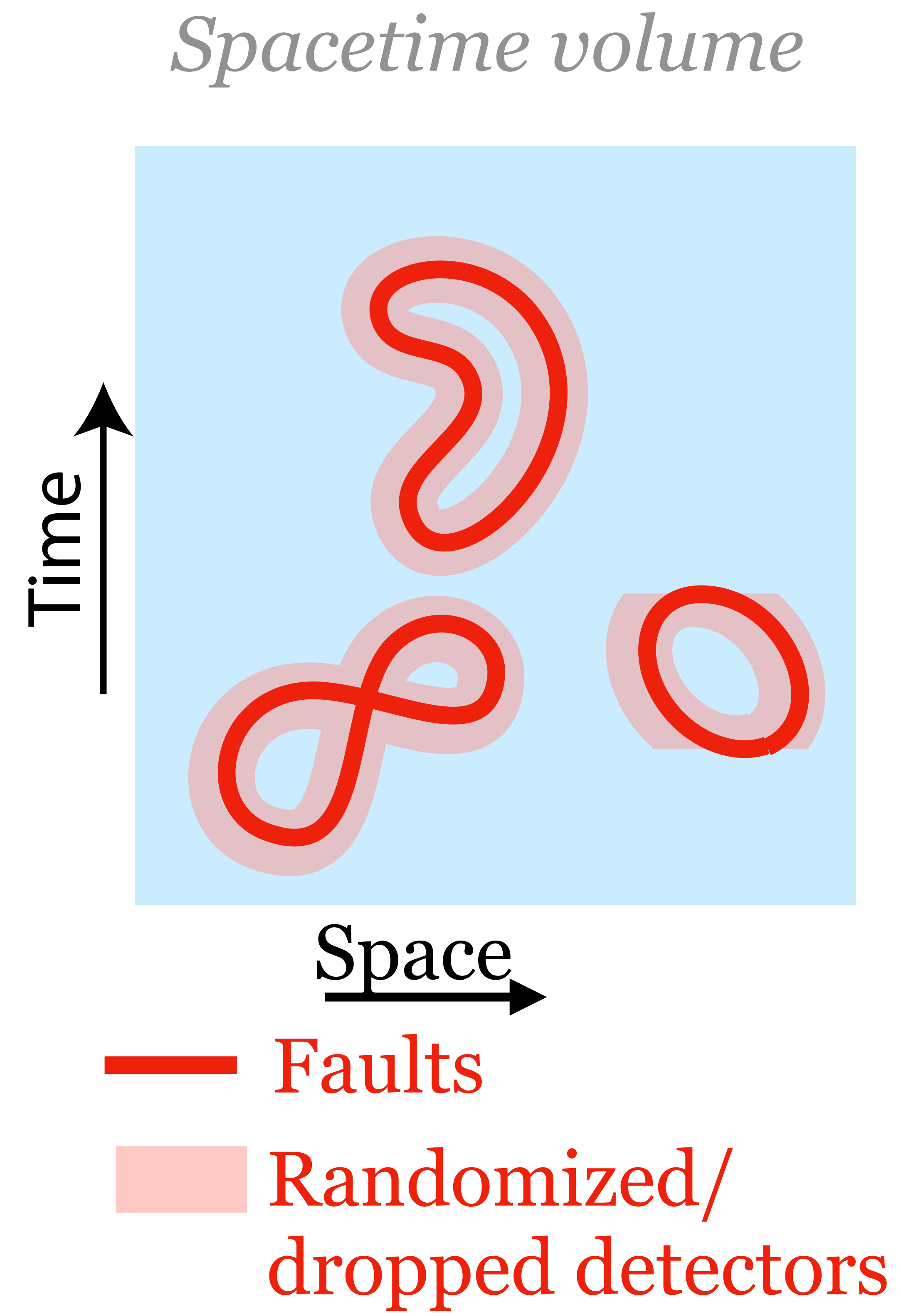
Changes to fault-tolerance in the gauged code

In contrast to normal, Pauli stabilized codes, **detectors** can now be **randomized from errors**!

Regular detectors : flip near anticommuting Pauli/measurement error.

Dropped detectors : irregardless of faults, returns random outcomes.

Ex. $[\![\mathbb{S}, \mathbb{A}_v]\!] = S_x$



To show **large distance to Pauli and measurement errors**, we use the following convenient observations:

When Clifford T is **order-2, stabilizer-randomizing errors** are **detected by stabilizers** that are **never random.**

\+ **LDPC**-ness

Randomized stabilizers are in some **bounded spacetime radius** around an *undetectable* fault.

The fault distance to Pauli and measurement errors is d/γ
(distance reduced by some constant depending on the connectivity of C_g)

Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

Two subsets of checks:

- $\mathbb{N}$ -type checks
- $\mathbb{C}$ -type checks

Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

Two subsets of checks:

- $\mathbb{N}$ -type checks
- $\mathbb{C}$ -type checks

$$[\![\mathbb{N}_a, \mathbb{N}_b]\!] = I$$
$$[\![\mathbb{N}_a, \mathbb{C}_i]\!] = I$$
$$[\![\mathbb{C}_i, \mathbb{C}_j]\!] \in \langle \mathbb{N} \rangle$$

Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

Two subsets of checks:

- $\mathbb{N}$ -type checks
- $\mathbb{C}$ -type checks
- $\mathbb{N}$-type checks always commute.

$$[\![\mathbb{N}_a, \mathbb{N}_b]\!] = I$$

$$[\![\mathbb{N}_a, \mathbb{C}_i]\!] = I$$

$$[\![\mathbb{C}_i, \mathbb{C}_j]\!] \in \langle \mathbb{N} \rangle$$

Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

Two subsets of checks:

$$[\![\mathbb{N}_a, \mathbb{N}_b]\!] = I$$
$$[\![\mathbb{N}_a, \mathbb{C}_i]\!] = I$$
$$[\![\mathbb{C}_i, \mathbb{C}_j]\!] \in \langle \mathbb{N} \rangle$$

- $\mathbb{N}$ -type checks
- $\mathbb{C}$ -type checks
- $\mathbb{N}$-type checks always commute.
- Faults that flip $\mathbb{N}$ are the only ones that cause ($\mathbb{C}$-type) stabilizers to not commute.

Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

- $\mathbb{N}$-type checks always commute.
- Faults that flip $\mathbb{N}$ are the only ones that cause ($\mathbb{C}$-type) stabilizers to not commute.

Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

- $\mathbb{N}$-type checks always commute.
- Faults that flip $\mathbb{N}$ are the only ones that cause ($\mathbb{C}$-type) stabilizers to not commute.

Randomized stabilizers are in some **bounded spacetime radius** around an *undetectable* fault.

Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

- $\mathbb{N}$-type checks always commute.
- Faults that flip $\mathbb{N}$ are the only ones that cause ($\mathbb{C}$-type) stabilizers to not commute.

Randomized stabilizers are in some **bounded spacetime radius** around an *undetectable* fault.

LDPC-ness:

Intuition for the adversarial setting

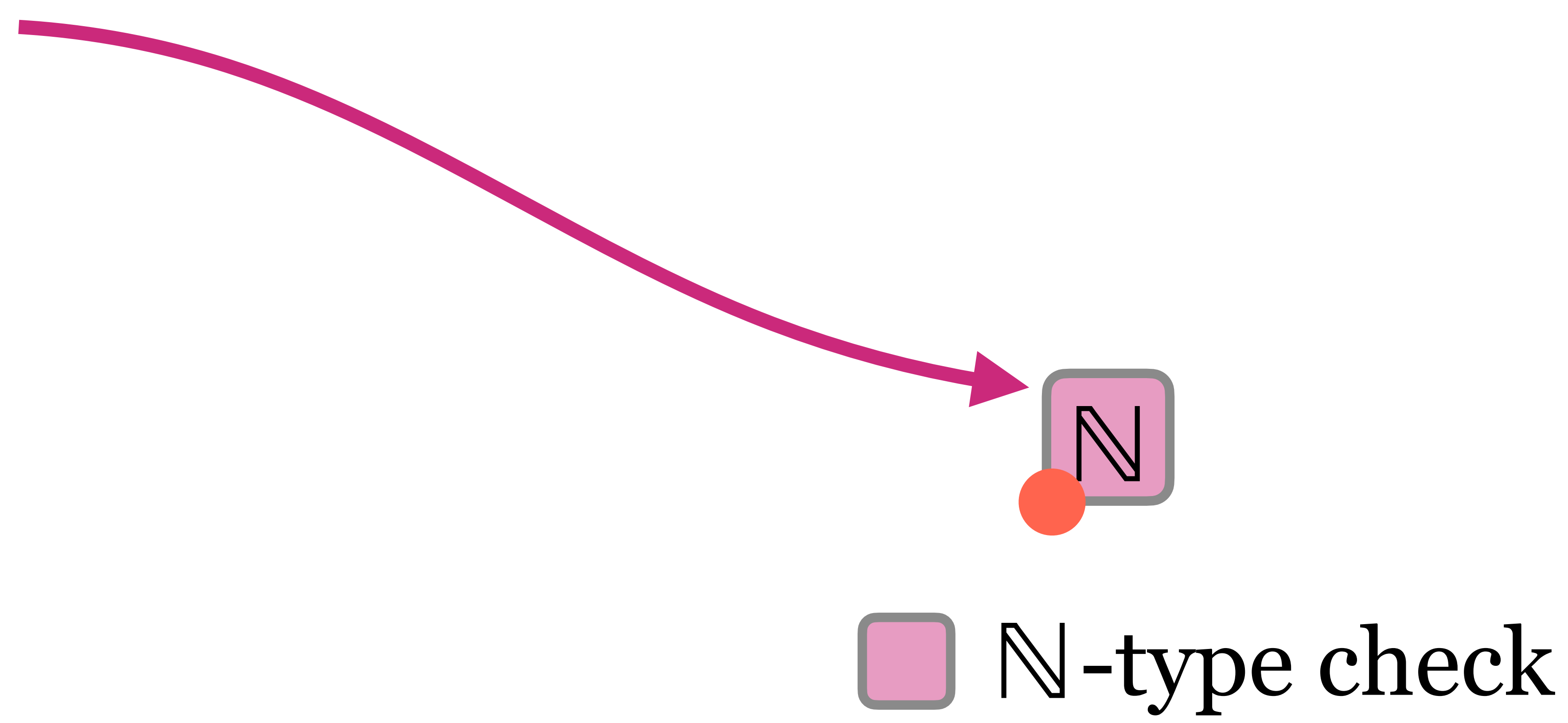
When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

- $\mathbb{N}$-type checks always commute.
- Faults that flip $\mathbb{N}$ are the only ones that cause ($\mathbb{C}$-type) stabilizers to not commute.

Randomized stabilizers are in some **bounded spacetime radius** around an *undetectable* fault.

LDPC-ness:

- Error ● can only flip $O(1)$ stabilizers around it. Say it violates $\mathbb{N}$



Intuition for the adversarial setting

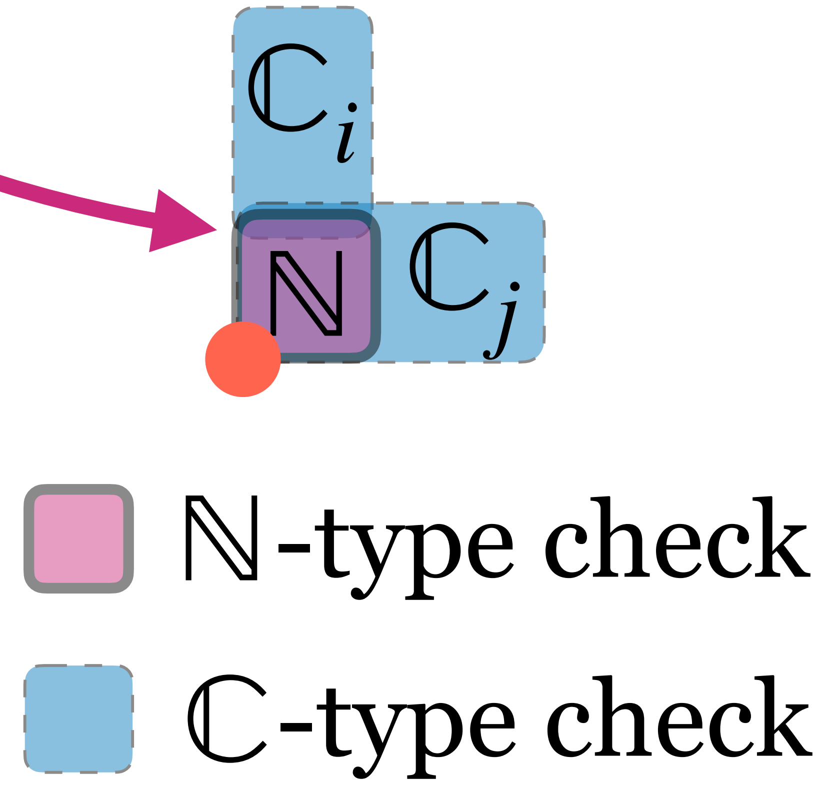
When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

- $\mathbb{N}$-type checks always commute.
- Faults that flip $\mathbb{N}$ are the only ones that cause ($\mathbb{C}$-type) stabilizers to not commute.

Randomized stabilizers are in some **bounded spacetime radius** around an *undetectable* fault.

LDPC-ness:

- Error ● can only flip $O(1)$ stabilizers around it. Say it violates $\mathbb{N}$
- If $\mathbb{N} = [\![\mathbb{C}_i, \mathbb{C}_j]\!]$ it must be that $\mathrm{Supp}(\mathbb{N}) \subseteq \mathrm{Supp}(\mathbb{C}_i) \cup \mathrm{Supp}(\mathbb{C}_j)$



Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

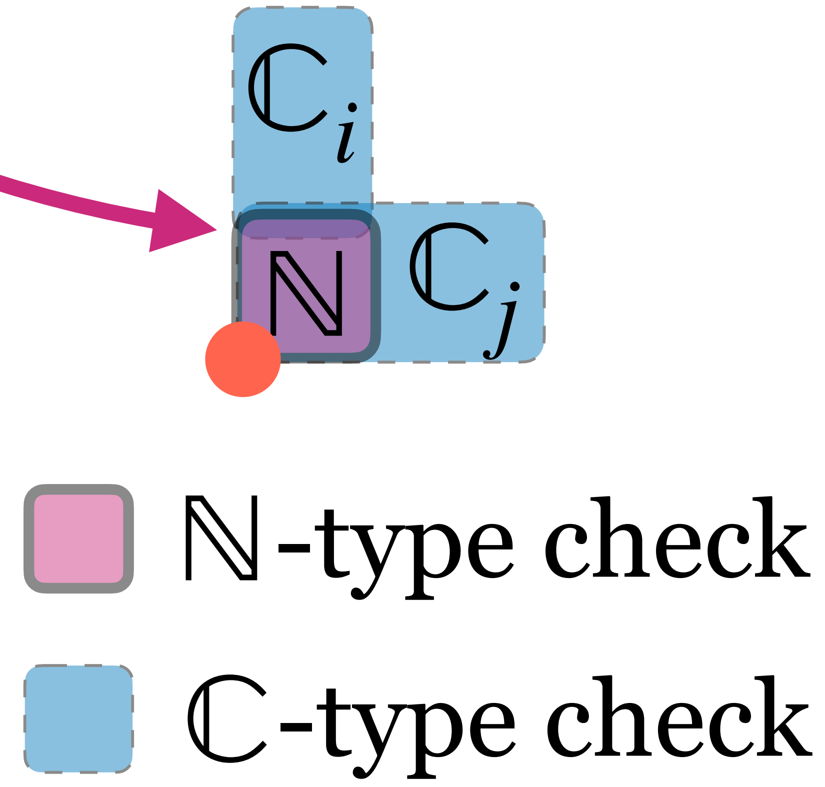
- $\mathbb{N}$-type checks always commute.
- Faults that flip $\mathbb{N}$ are the only ones that cause ($\mathbb{C}$-type) stabilizers to not commute.

Randomized stabilizers are in some **bounded spacetime radius** around an *undetectable* fault.

LDPC-ness:

- Error ● can only flip $O(1)$ stabilizers around it. Say it violates $\mathbb{N}$
- If $\mathbb{N} = [\![\mathbb{C}_i, \mathbb{C}_j]\!]$ it must be that $\mathrm{Supp}(\mathbb{N}) \subseteq \mathrm{Supp}(\mathbb{C}_i) \cup \mathrm{Supp}(\mathbb{C}_j)$

A space error can cause stabilizers to not commute for unbounded time



Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

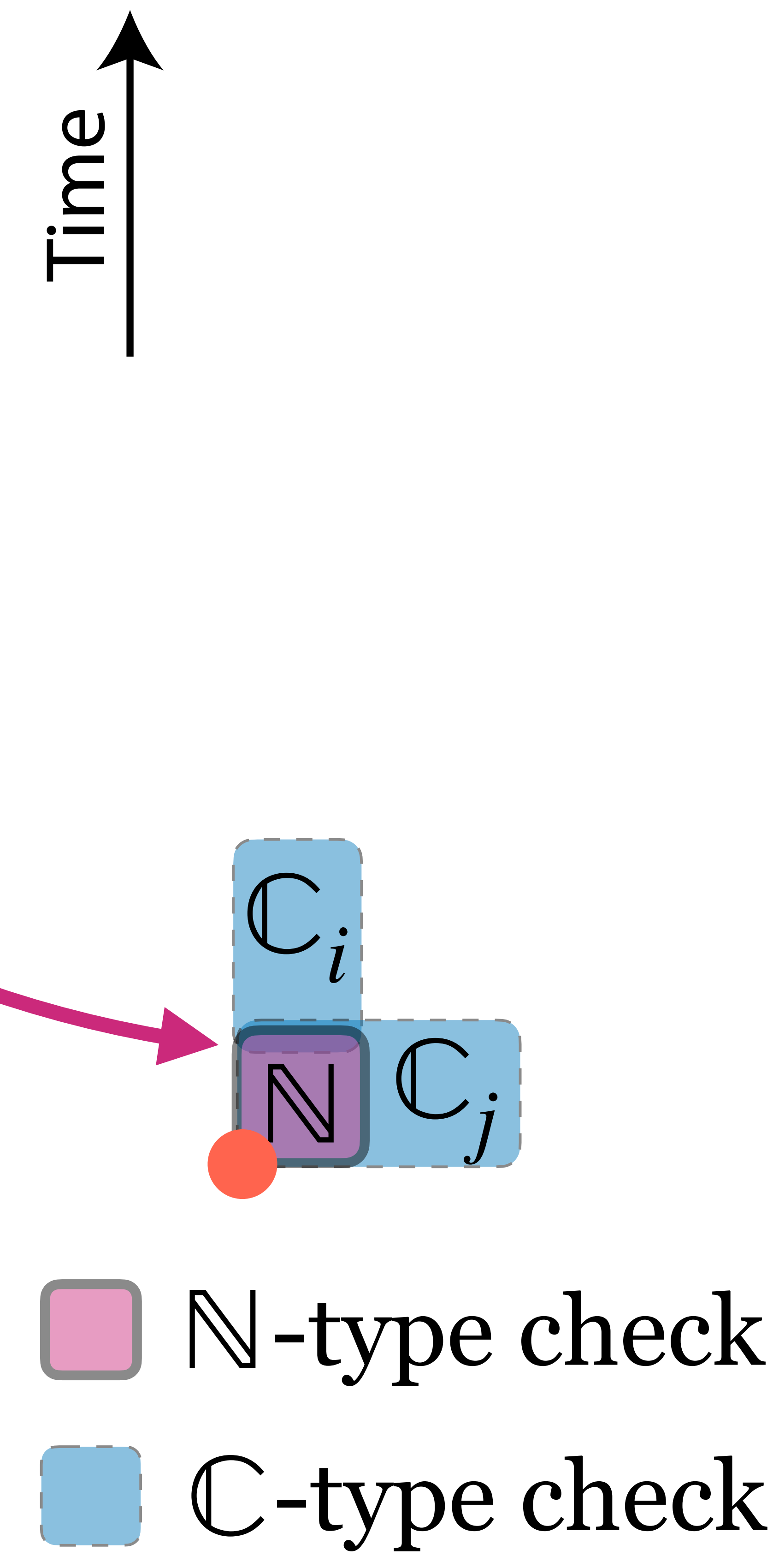
- $\mathbb{N}$-type checks always commute.
- Faults that flip $\mathbb{N}$ are the only ones that cause ($\mathbb{C}$-type) stabilizers to not commute.

Randomized stabilizers are in some **bounded spacetime radius** around an *undetectable* fault.

LDPC-ness:

- Error ● can only flip $O(1)$ stabilizers around it. Say it violates $\mathbb{N}$
- If $\mathbb{N} = [\![\mathbb{C}_i, \mathbb{C}_j]\!]$ it must be that $\mathrm{Supp}(\mathbb{N}) \subseteq \mathrm{Supp}(\mathbb{C}_i) \cup \mathrm{Supp}(\mathbb{C}_j)$

A space error can cause stabilizers to not commute for unbounded time



Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

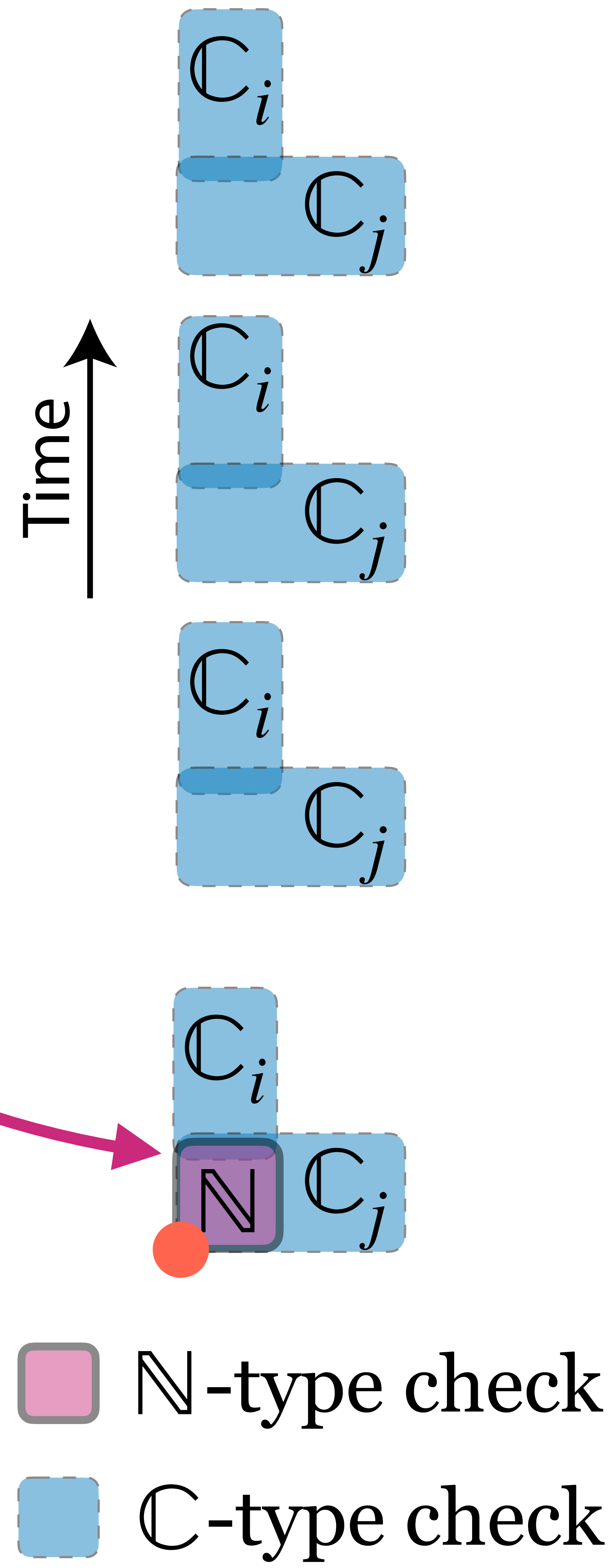
- $\mathbb{N}$-type checks always commute.
- Faults that flip $\mathbb{N}$ are the only ones that cause ($\mathbb{C}$-type) stabilizers to not commute.

Randomized stabilizers are in some bounded spacetime radius around an *undetectable* fault.

LDPC-ness:

- Error ● can only flip $O(1)$ stabilizers around it. Say it violates $\mathbb{N}$
- If $\mathbb{N} = [\![\mathbb{C}_i, \mathbb{C}_j]\!]$ it must be that $\mathrm{Supp}(\mathbb{N}) \subseteq \mathrm{Supp}(\mathbb{C}_i) \cup \mathrm{Supp}(\mathbb{C}_j)$

A space error can cause stabilizers to not commute for unbounded time



Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

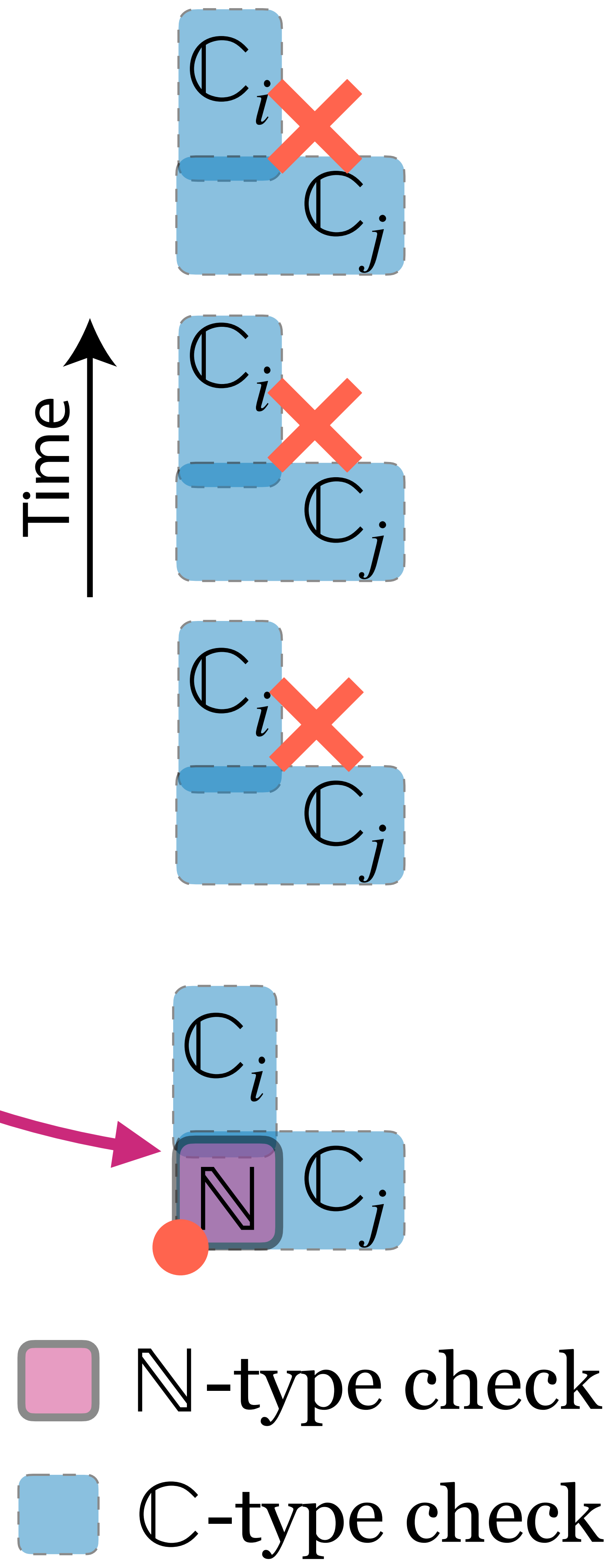
- $\mathbb{N}$-type checks always commute.
- Faults that flip $\mathbb{N}$ are the only ones that cause ($\mathbb{C}$-type) stabilizers to not commute.

Randomized stabilizers are in some **bounded spacetime radius** around an *undetectable* fault.

LDPC-ness:

- Error can only flip $O(1)$ stabilizers around it. Say it violates $\mathbb{N}$
- If $\mathbb{N} = [\![\mathbb{C}_i, \mathbb{C}_j]\!]$ it must be that $\mathrm{Supp}(\mathbb{N}) \subseteq \mathrm{Supp}(\mathbb{C}_i) \cup \mathrm{Supp}(\mathbb{C}_j)$

A space error can cause stabilizers to not commute for unbounded time



Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

- $\mathbb{N}$-type checks always commute.
- Faults that flip $\mathbb{N}$ are the only ones that cause ($\mathbb{C}$-type) stabilizers to not commute.

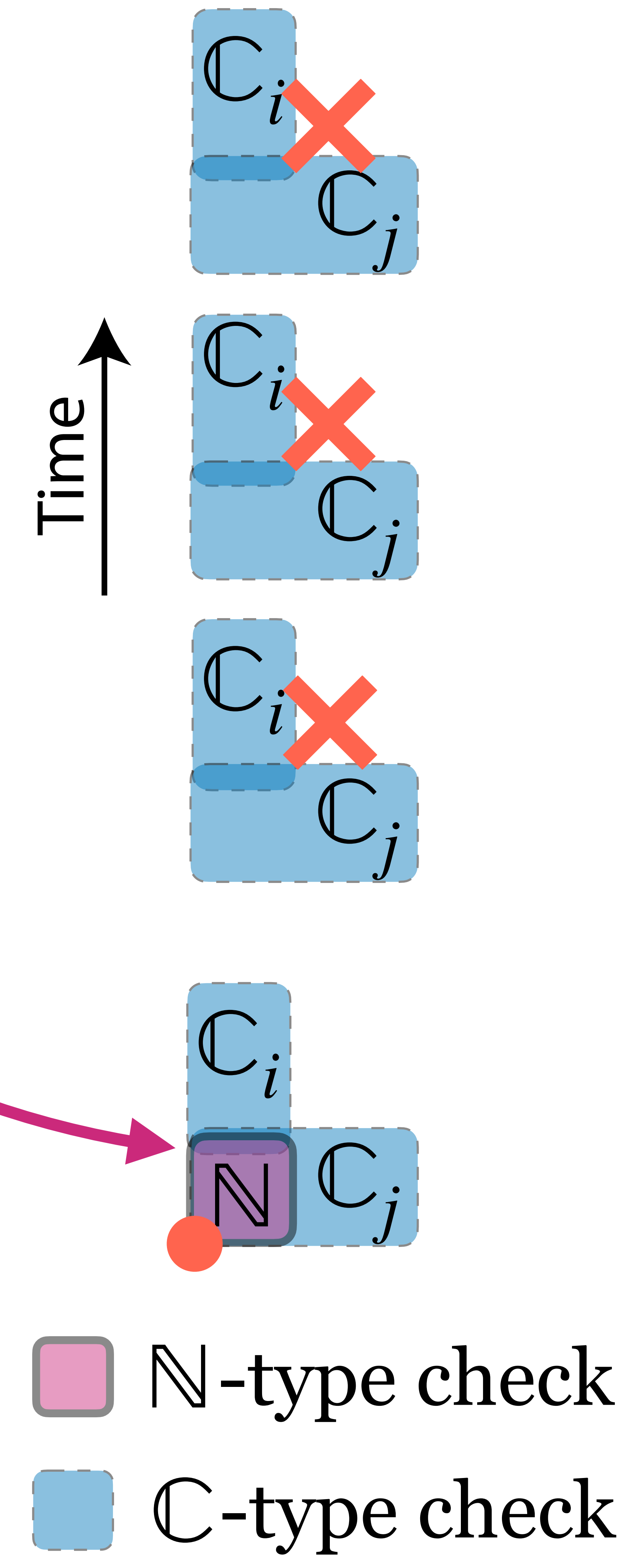
Randomized stabilizers are in some bounded spacetime radius around an *undetectable* fault.

LDPC-ness:

- Error ● can only flip $O(1)$ stabilizers around it. Say it violates $\mathbb{N}$
- If $\mathbb{N} = [\![\mathbb{C}_i, \mathbb{C}_j]\!]$ it must be that $\mathrm{Supp}(\mathbb{N}) \subseteq \mathrm{Supp}(\mathbb{C}_i) \cup \mathrm{Supp}(\mathbb{C}_j)$

A space error can cause stabilizers to not commute for unbounded time

But, errors that flip $\mathbb{N}$ can always be detected



Intuition for the adversarial setting

When Clifford T is order-2, stabilizer-randomizing errors are detected by stabilizers that are never random.

- $\mathbb{N}$-type checks always commute.
- Faults that flip $\mathbb{N}$ are the only ones that cause ($\mathbb{C}$-type) stabilizers to not commute.

Randomized stabilizers are in some **bounded spacetime radius** around an *undetectable* fault.

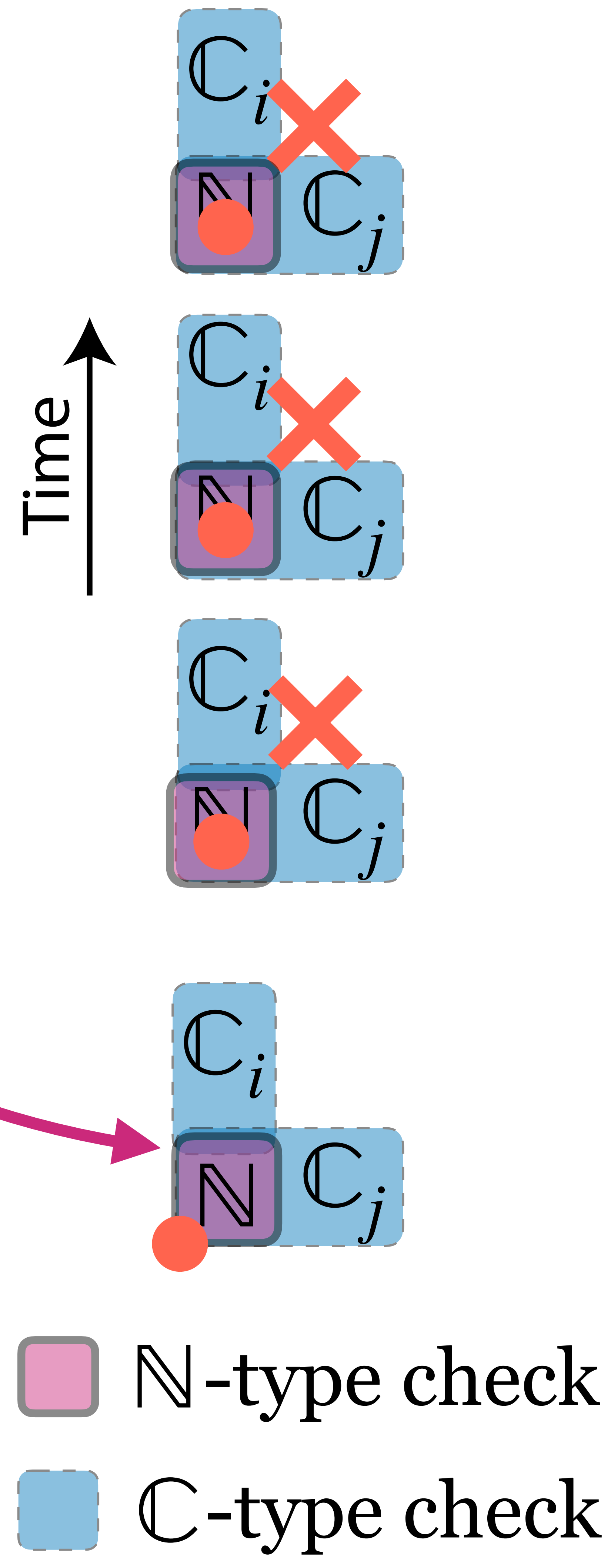
LDPC-ness:

- Error ● can only flip $O(1)$ stabilizers around it. Say it violates $\mathbb{N}$
- If $\mathbb{N} = [\![\mathbb{C}_i, \mathbb{C}_j]\!]$ it must be that $\mathrm{Supp}(\mathbb{N}) \subseteq \mathrm{Supp}(\mathbb{C}_i) \cup \mathrm{Supp}(\mathbb{C}_j)$

A space error can cause stabilizers to not commute for unbounded time

But, errors that flip $\mathbb{N}$ can always be detected

- For this space error to be undetected, a measurement error needed to have occurred on $\mathbb{N}$ for each round that $\mathbb{C}_i$ and $\mathbb{C}_j$ do not commute



More work to be done!

I live my life a quarter mile at a time - Vin Diesel

More work to be done!

I live my life a quarter mile at a time - Vin Diesel

Sometimes adversarial fault distance is not enough

More work to be done!

I live my life a quarter mile at a time - Vin Diesel

Sometimes adversarial fault distance is not enough

Typically one waits d rounds before inferring a correction

More work to be done!

I live my life a quarter mile at a time - Vin Diesel

Sometimes adversarial fault distance is not enough

Typically one waits d rounds before inferring a correction

Non-abelian codes are trickier! Waiting d rounds before correction leaves too many dropped detectors.

More work to be done!

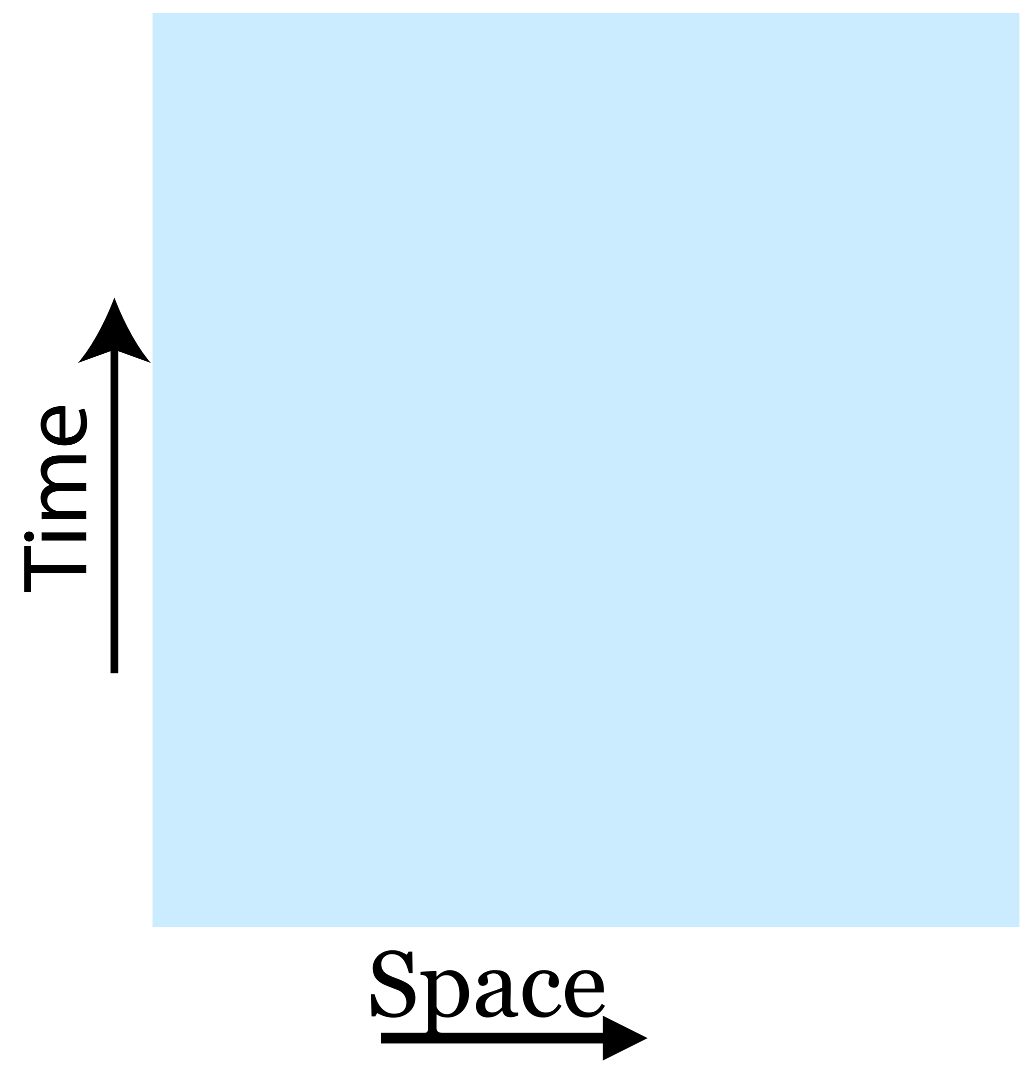
I live my life a quarter mile at a time - Vin Diesel

Sometimes adversarial fault distance is not enough

Typically one waits d rounds before inferring a correction

Non-abelian codes are trickier! Waiting d rounds before correction leaves too many dropped detectors.

Spacetime volume



More work to be done!

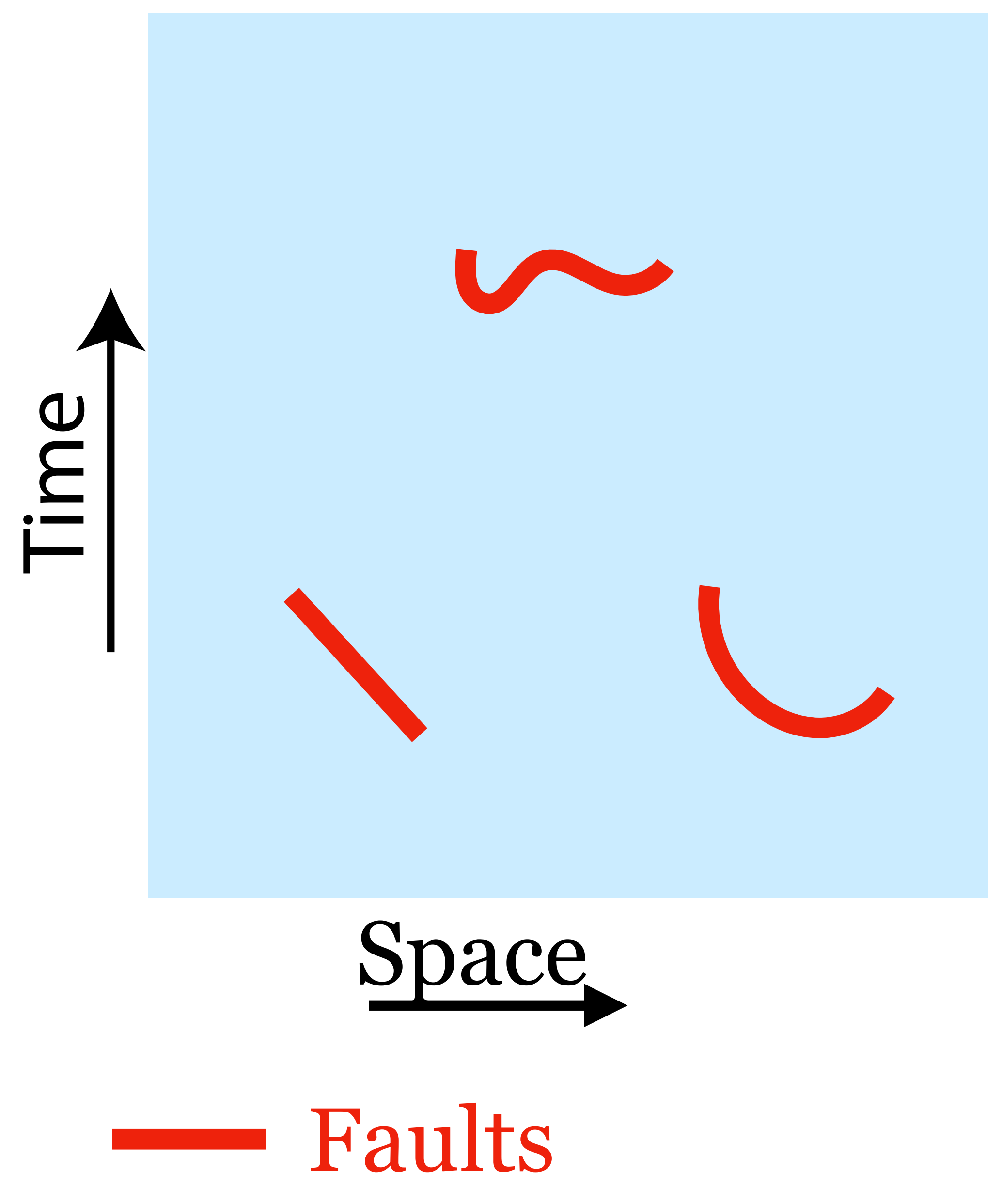
I live my life a quarter mile at a time - Vin Diesel

Sometimes adversarial fault distance is not enough

Typically one waits d rounds before inferring a correction

Non-abelian codes are trickier! Waiting d rounds before correction leaves too many dropped detectors.

Spacetime volume



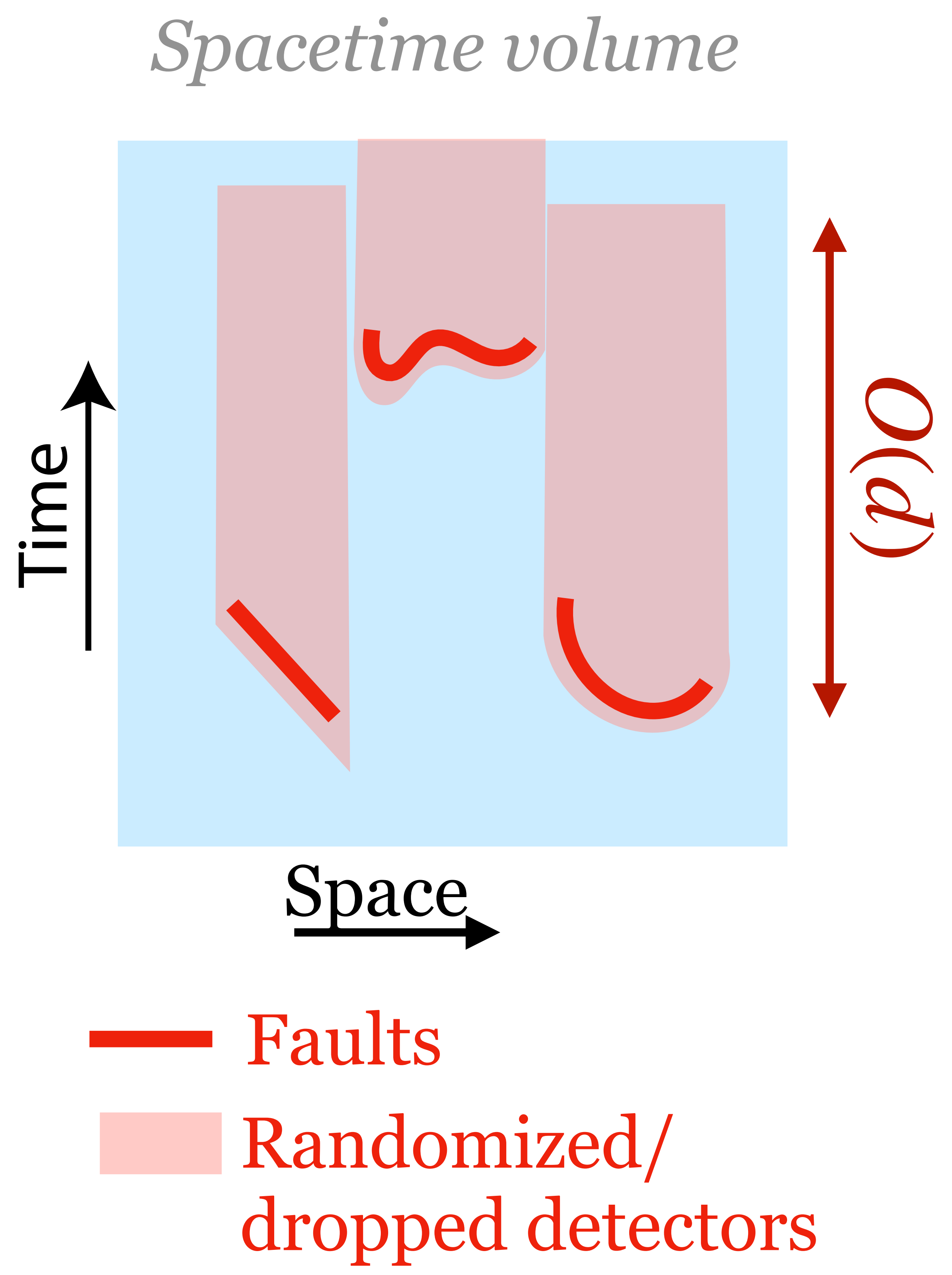
More work to be done!

I live my life a quarter mile at a time - Vin Diesel

Sometimes adversarial fault distance is not enough

Typically one waits d rounds before inferring a correction

Non-abelian codes are trickier! Waiting d rounds before correction leaves too many dropped detectors.



More work to be done!

I live my life a quarter mile at a time - Vin Diesel

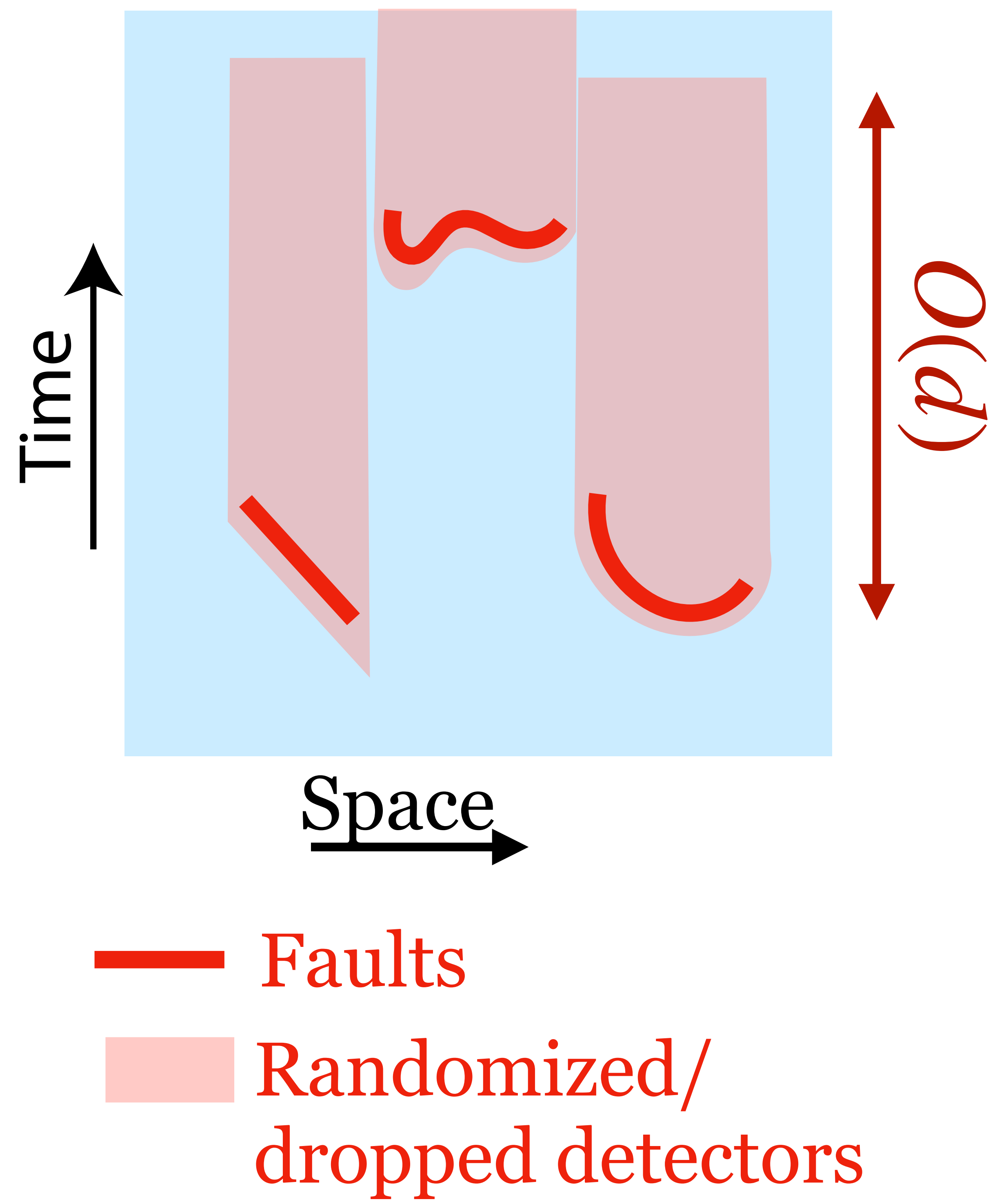
Sometimes adversarial fault distance is not enough

Typically one waits d rounds before inferring a correction

Non-abelian codes are trickier! Waiting d rounds before correction leaves too many dropped detectors.

In practice, need single-shot EC or a Just-in-Time decoder to infer corrections quickly enough for a finite threshold[1,2]
(WIP for QLDPC codes)

Spacetime volume



[1] Bombín. [1810.09571] [2] Brown. Sci. Adv. **6** (2024)

More work to be done!

I live my life a quarter mile at a time - Vin Diesel

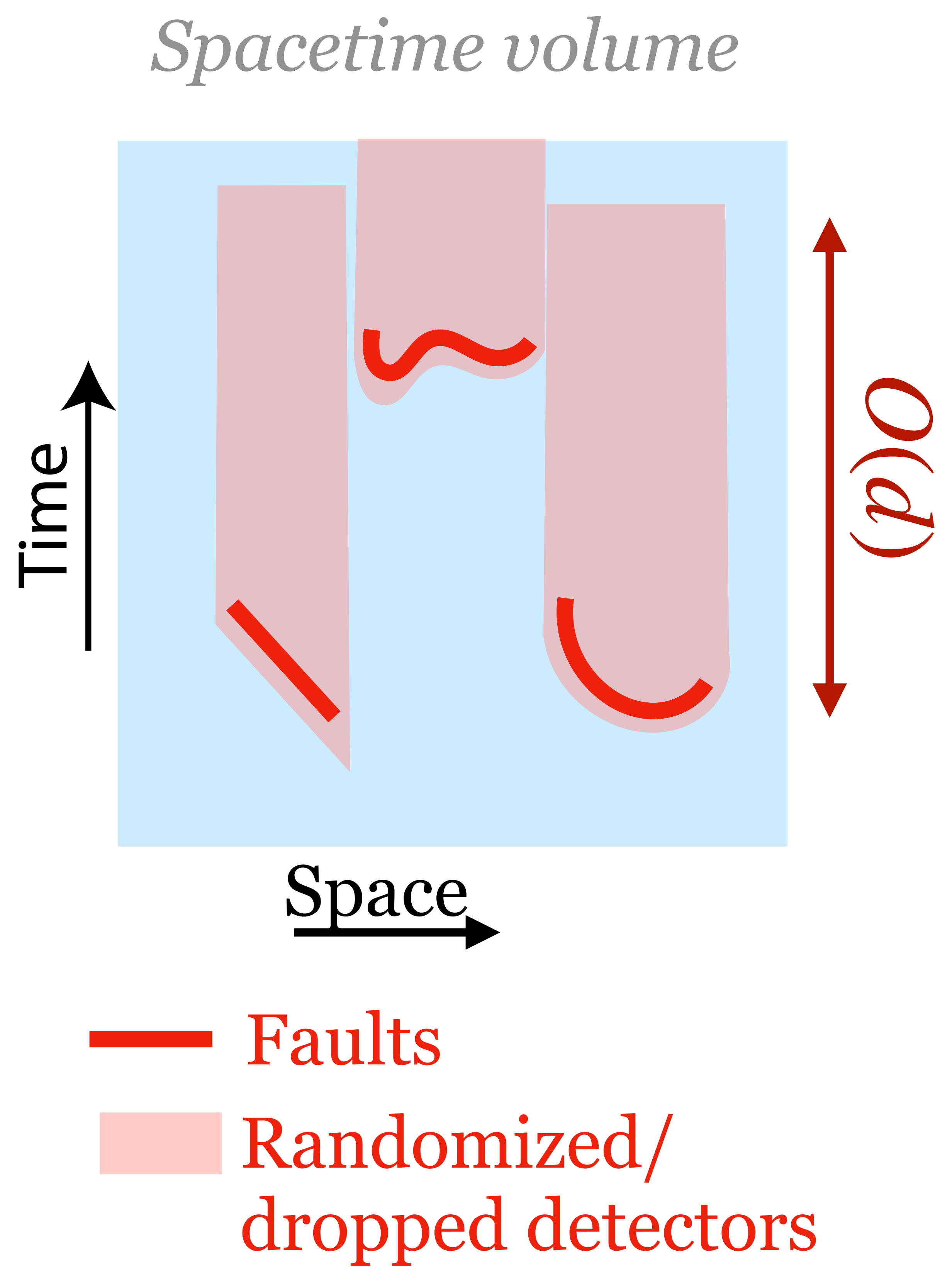
Sometimes adversarial fault distance is not enough

Typically one waits d rounds before inferring a correction

Non-abelian codes are trickier! Waiting d rounds before correction leaves too many dropped detectors.

In practice, need single-shot EC or a Just-in-Time decoder to infer corrections quickly enough for a finite threshold[1,2]
(WIP for QLDPC codes)

Heuristic decoders?[3,4]



[1] Bombín. [1810.09571] [2] Brown. Sci. Adv. **6** (2024) [3] Scruby, et al. Quantum **6** (2022). [4] Magdalena de la Fuente, et al. [2604.02033]

Toffoli gate resource states on any CSS code

Toffoli gate resource states on any CSS code

How do we use this to make magic states?

Toffoli gate resource states on any CSS code

How do we use this to make magic states? → what transversal Cliffords can we gauge?

$T = \ ?$

Toffoli gate resource states on any CSS code

How do we use this to make magic states? $\rightarrow$ **what transversal Cliffords can we gauge?**

Problem: Many transversal Cliffords in high rate codes are not addressable. e.g. $H^{\otimes k}$, $CX^{\otimes k}$

$\rightarrow$ **Won't get** $|T\rangle^{\otimes k}$

T = ?

Toffoli gate resource states on any CSS code

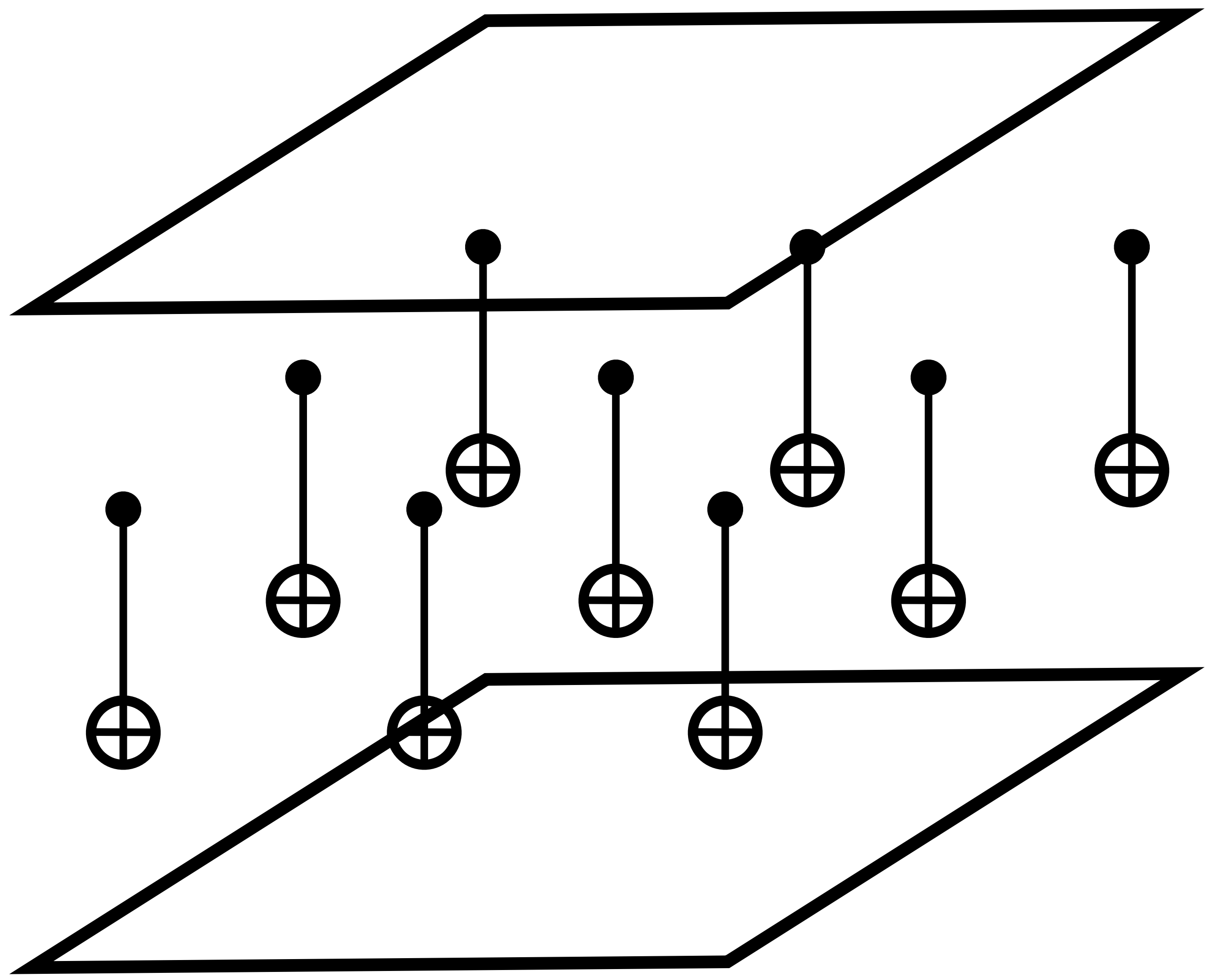
How do we use this to make magic states? → what transversal Cliffords can we gauge?

Problem: Many transversal Cliffords in high rate codes are not addressable. e.g. $H^{\otimes k}$, $CX^{\otimes k}$

→Won't get $|T\rangle^{\otimes k}$

Solution: Instead of looking for addressable Cliffords, pay some time cost to **make it work with a transversal $CX^{\otimes k}$**

T =



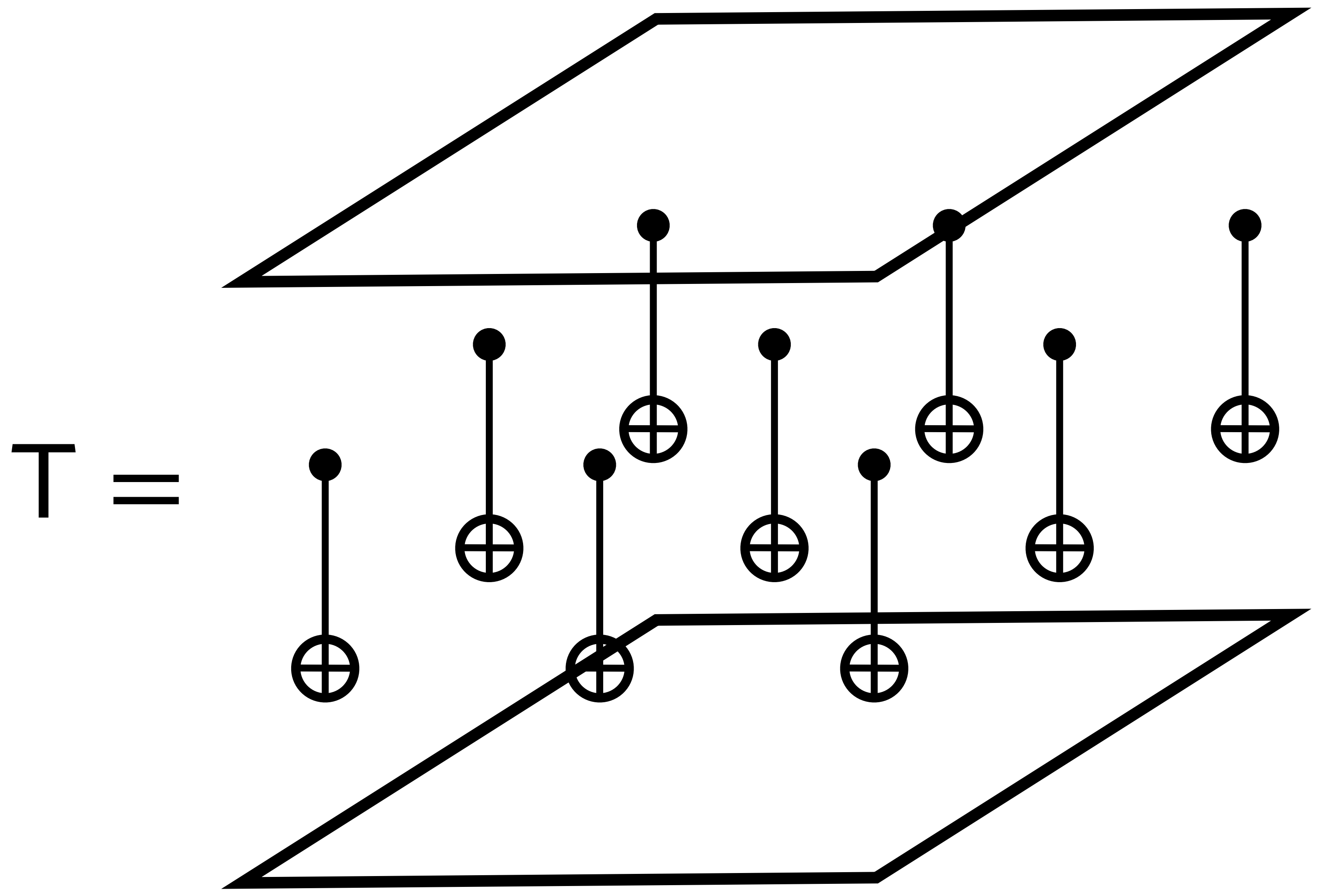
Toffoli gate resource states on any CSS code

How do we use this to make magic states? $\rightarrow$ **what transversal Cliffords can we gauge?**

Problem: Many transversal Cliffords in high rate codes are not addressable. e.g. $H^{\otimes k}$, $CX^{\otimes k}$

$\rightarrow$**Won't get** $|T\rangle^{\otimes k}$

Solution: Instead of looking for addressable Cliffords, pay some time cost to **make it work with a transversal** $CX^{\otimes k}$



$|+\mathrm{CX}\rangle^{\otimes 2}$ **can be used to create** $|\mathrm{TOF}\rangle$ with a few Pauli measurements.

Toffoli gate resource states on any CSS code

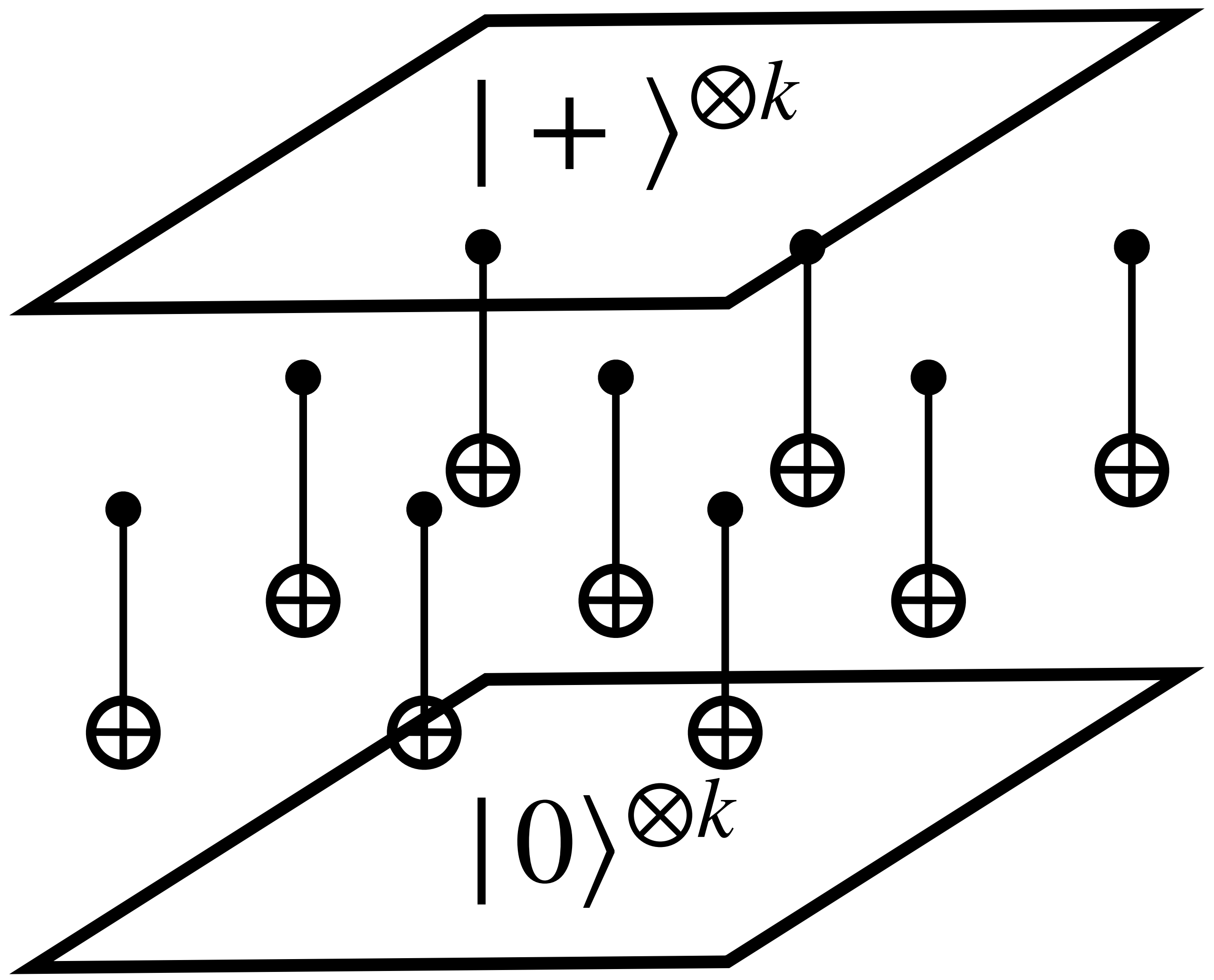
How do we use this to make magic states? $\rightarrow$ **what transversal Cliffords can we gauge?**

Problem: Many transversal Cliffords in high rate codes are not addressable. e.g. $H^{\otimes k}$, $CX^{\otimes k}$

$\rightarrow$**Won't get** $|T\rangle^{\otimes k}$

Solution: Instead of looking for addressable Cliffords, pay some time cost to **make it work with a transversal** $CX^{\otimes k}$

T =



$|+\mathrm{CX}\rangle^{\otimes 2}$ **can be used to create** $|\mathrm{TOF}\rangle$ with a few Pauli measurements.

- Initialize control logicals in $|+\rangle$ and target logicals in $|0\rangle$

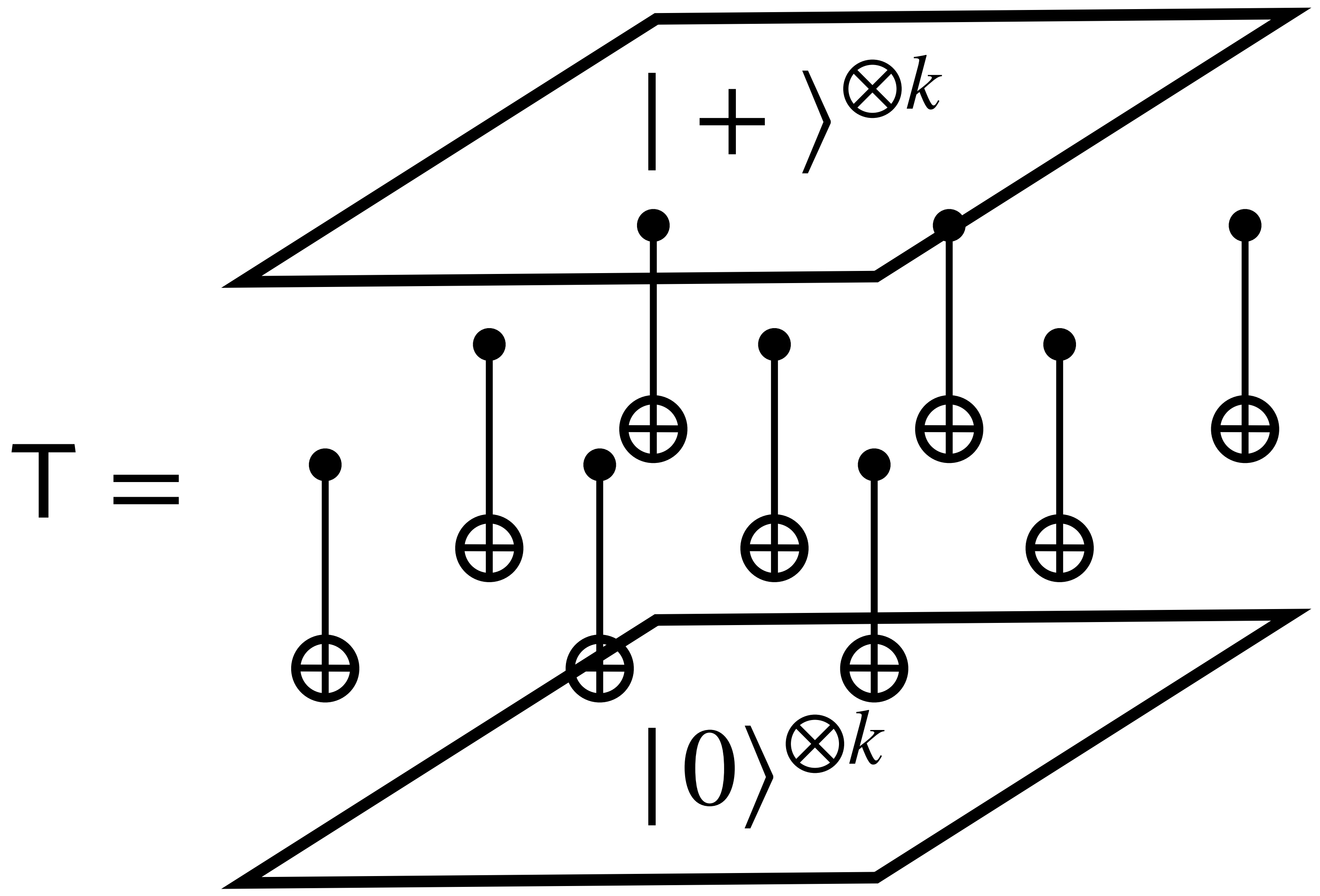
Toffoli gate resource states on any CSS code

How do we use this to make magic states? $\rightarrow$ **what transversal Cliffords can we gauge?**

Problem: Many transversal Cliffords in high rate codes are not addressable. e.g. $H^{\otimes k}$, $CX^{\otimes k}$

$\rightarrow$**Won't get** $|T\rangle^{\otimes k}$

Solution: Instead of looking for addressable Cliffords, pay some time cost to **make it work with a transversal** $CX^{\otimes k}$



$|+\mathrm{CX}\rangle^{\otimes 2}$ **can be used to create** $|\mathrm{TOF}\rangle$ with a few Pauli measurements.

- Initialize control logicals in $|+\rangle$ and target logicals in $|0\rangle$
- Repeatedly gauge the global T, using Pauli measurements with feedforward control.

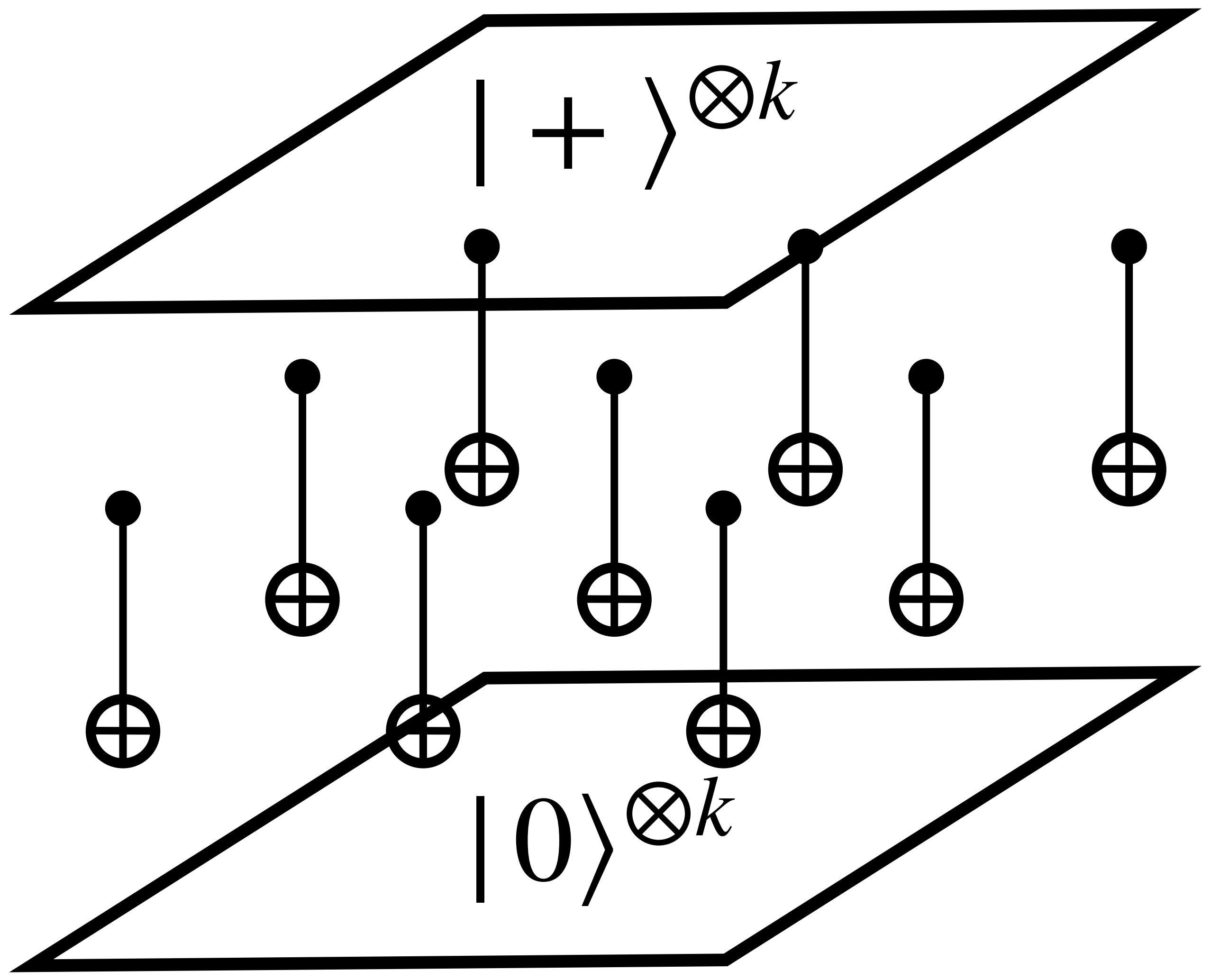
Toffoli gate resource states on any CSS code

How do we use this to make magic states? $\rightarrow$ **what transversal Cliffords can we gauge?**

Problem: Many transversal Cliffords in high rate codes are not addressable. e.g. $H^{\otimes k}$, $CX^{\otimes k}$

$\rightarrow$ **Won't get** $|T\rangle^{\otimes k}$

Solution: Instead of looking for addressable Cliffords, pay some time cost to **make it work with a transversal** $CX^{\otimes k}$

T = 

$|+\mathrm{CX}\rangle^{\otimes 2}$ **can be used to create** $|\mathrm{TOF}\rangle$ with a few Pauli measurements.

- Initialize control logicals in $|+\rangle$ and target logicals in $|0\rangle$
- Repeatedly gauge the global T, using Pauli measurements with feedforward control.
- $|+\mathrm{CX}\rangle^{\otimes k}$ after $\Theta(k)$ measurements of T

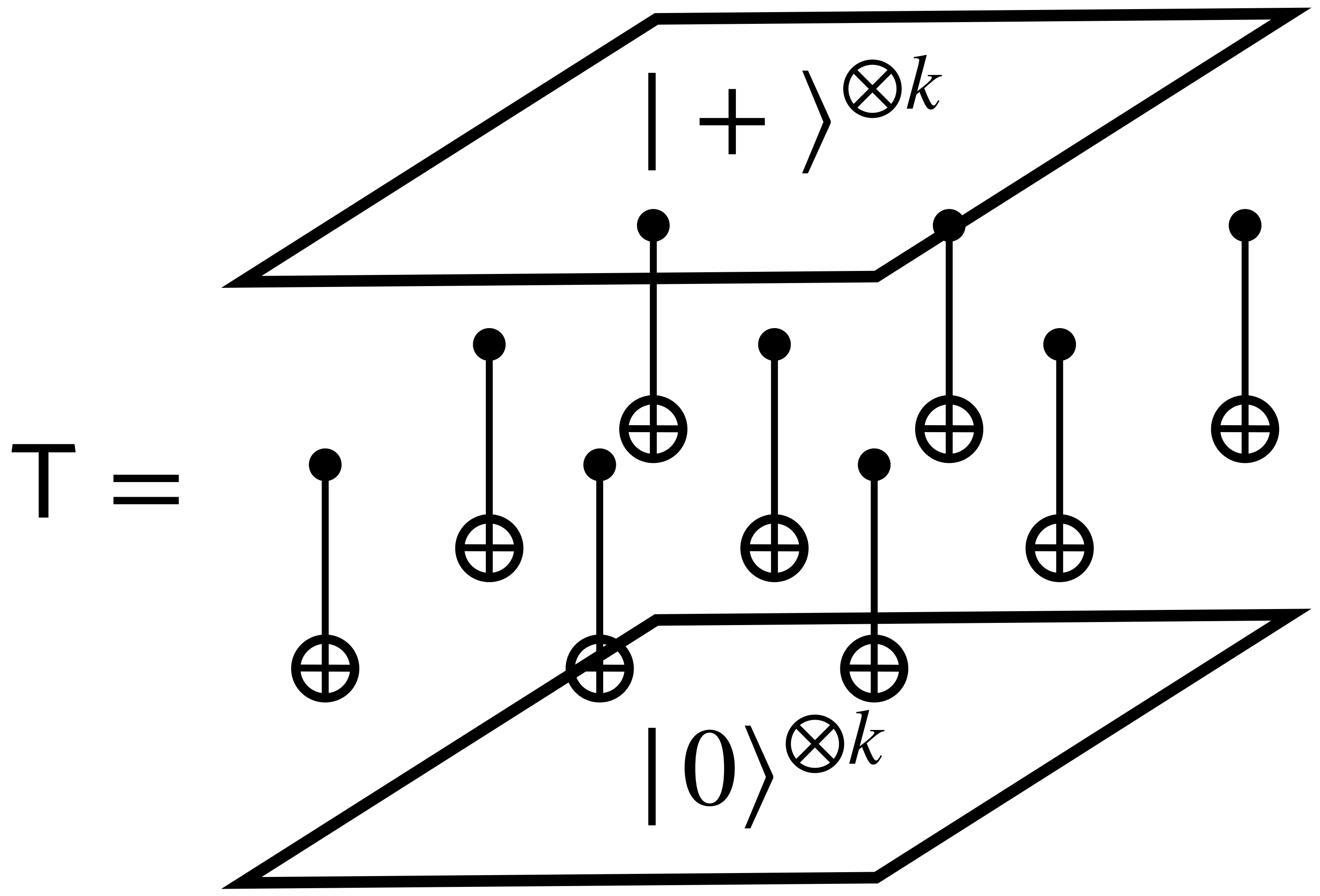
Toffoli gate resource states on any CSS code

How do we use this to make magic states? → **what transversal Cliffords can we gauge?**

Problem: Many transversal Cliffords in high rate codes are not addressable. e.g. $H^{\otimes k}$, $CX^{\otimes k}$

→**Won't get** $|T\rangle^{\otimes k}$

Solution: Instead of looking for addressable Cliffords, pay some time cost to **make it work with a transversal** $CX^{\otimes k}$



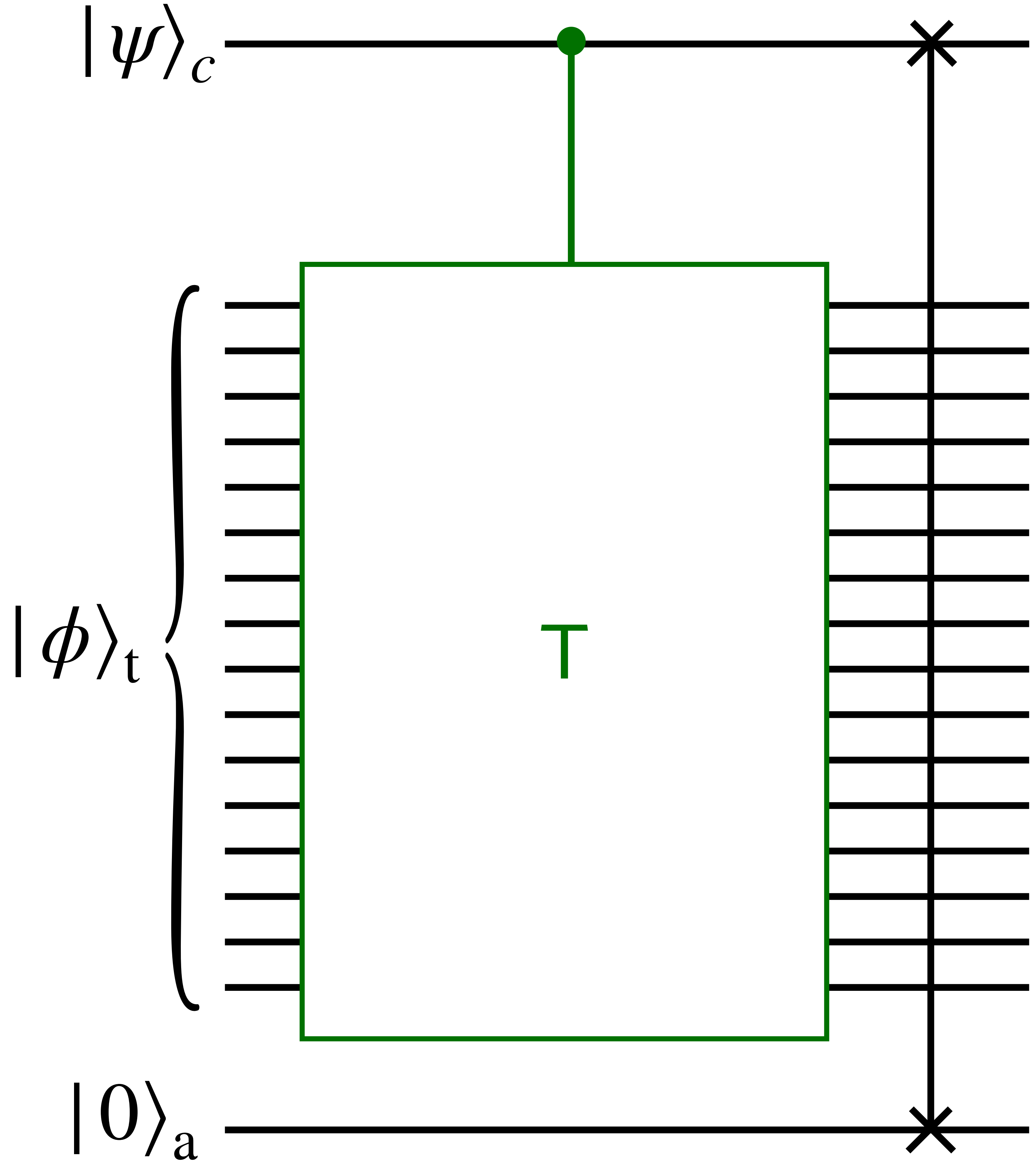
$|+\mathrm{CX}\rangle^{\otimes 2}$ **can be used to create** $|\mathrm{TOF}\rangle$ with a few Pauli measurements.

- Initialize control logicals in $|+\rangle$ and target logicals in $|0\rangle$
- Repeatedly gauge the global T, using Pauli measurements with feedforward control.
- $|+\mathrm{CX}\rangle^{\otimes k}$ after $\Theta(k)$ measurements of T

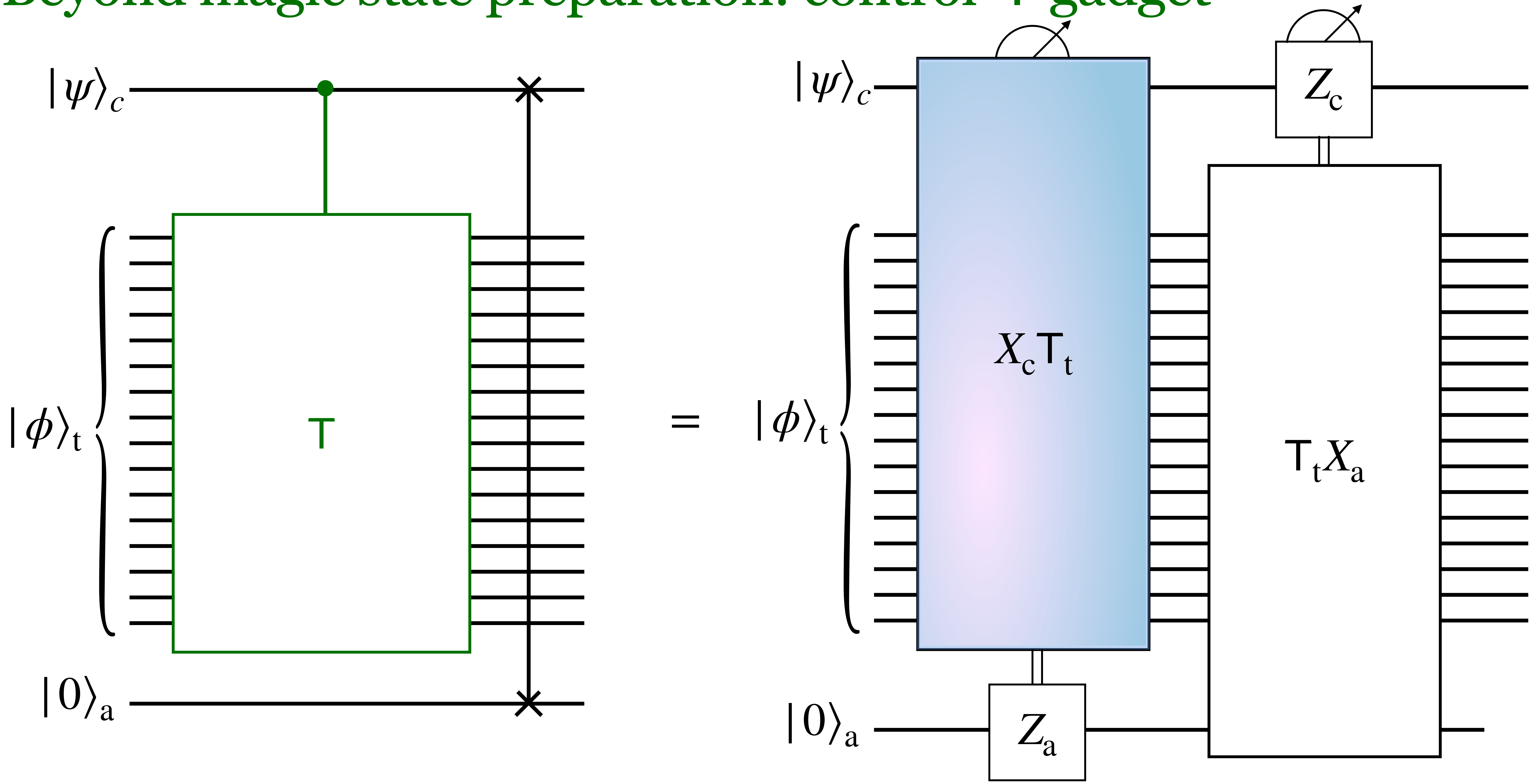
Very mild requirement on the code (transversal CNOT) to generate magic states in situ.
In principle, any CSS QLDPC code could be turned into a magic state factory.

Beyond magic state preparation: control-T gadget

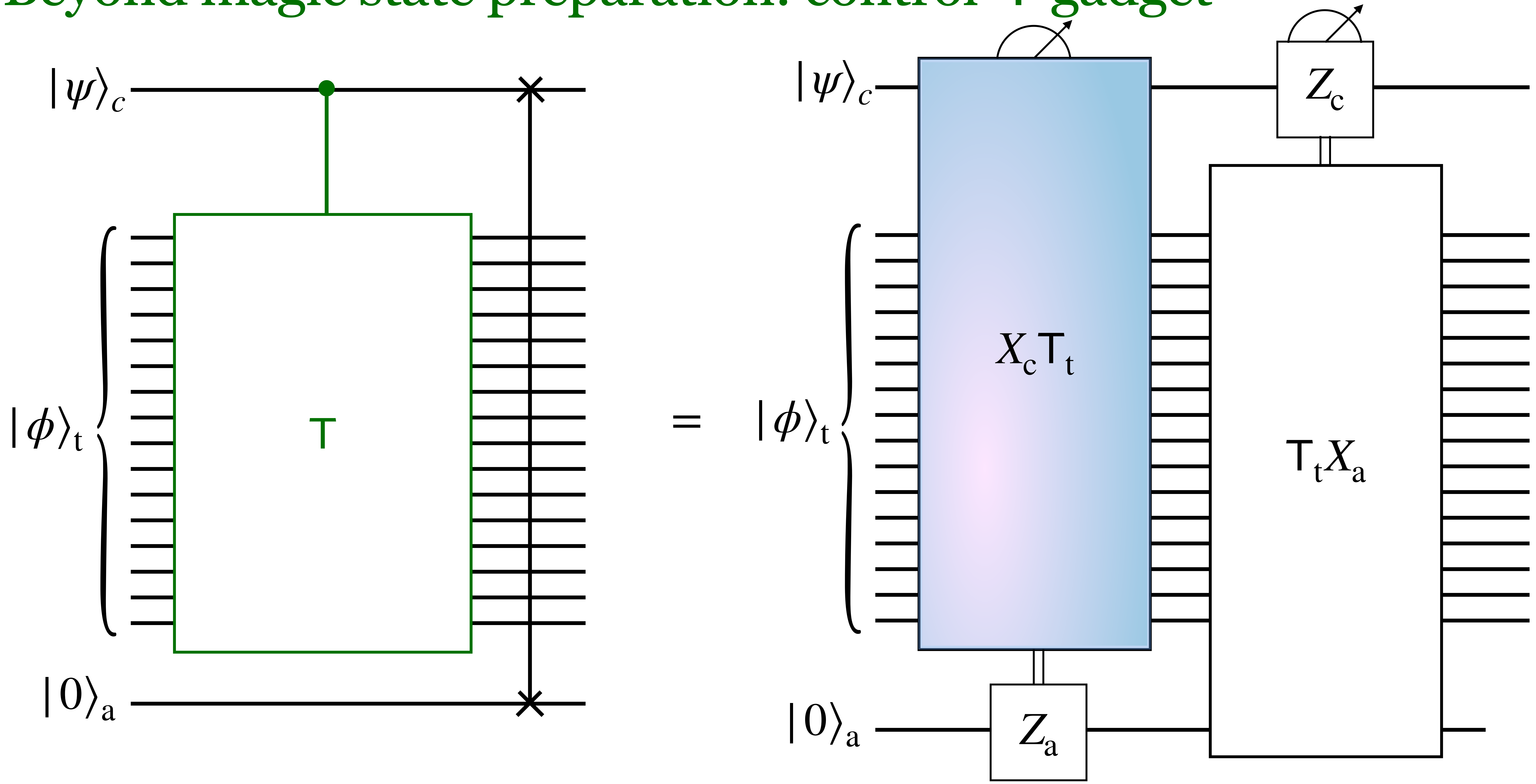
Beyond magic state preparation: control-T gadget



Beyond magic state preparation: control-T gadget



Beyond magic state preparation: control-T gadget



One Clifford measurement + feedforward Paulis and Cliffords *deterministically* performs the control-T

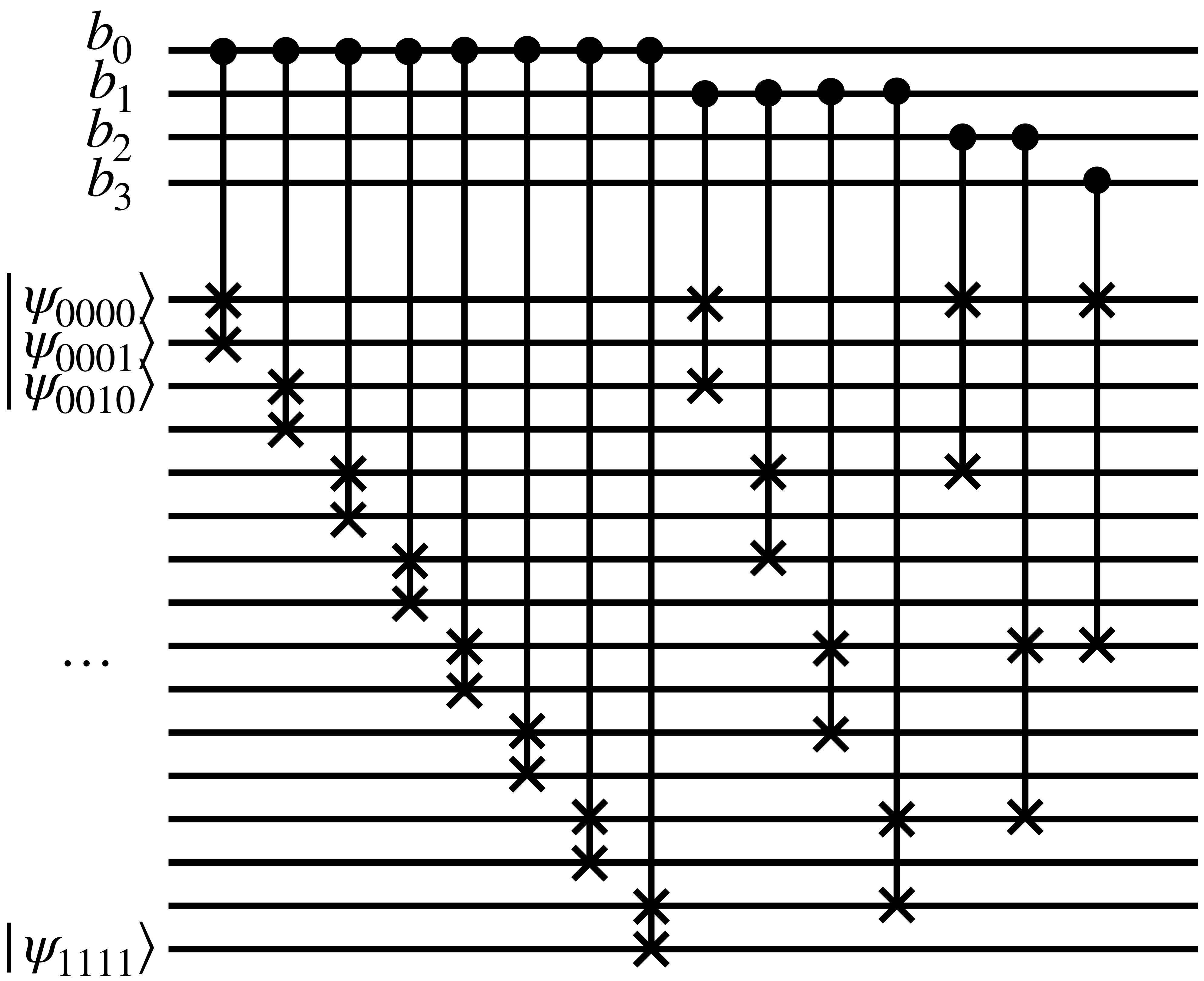
Data lookup in superposition

Data lookup in superposition

- controlled-SWAP networks are used in data look-up oracles, or quantum read only memory[1,2].

[1] Low, et al. Quantum **8** (2024). [2] Motlagh, Pocrnic. [2605.20334]

Data lookup in superposition

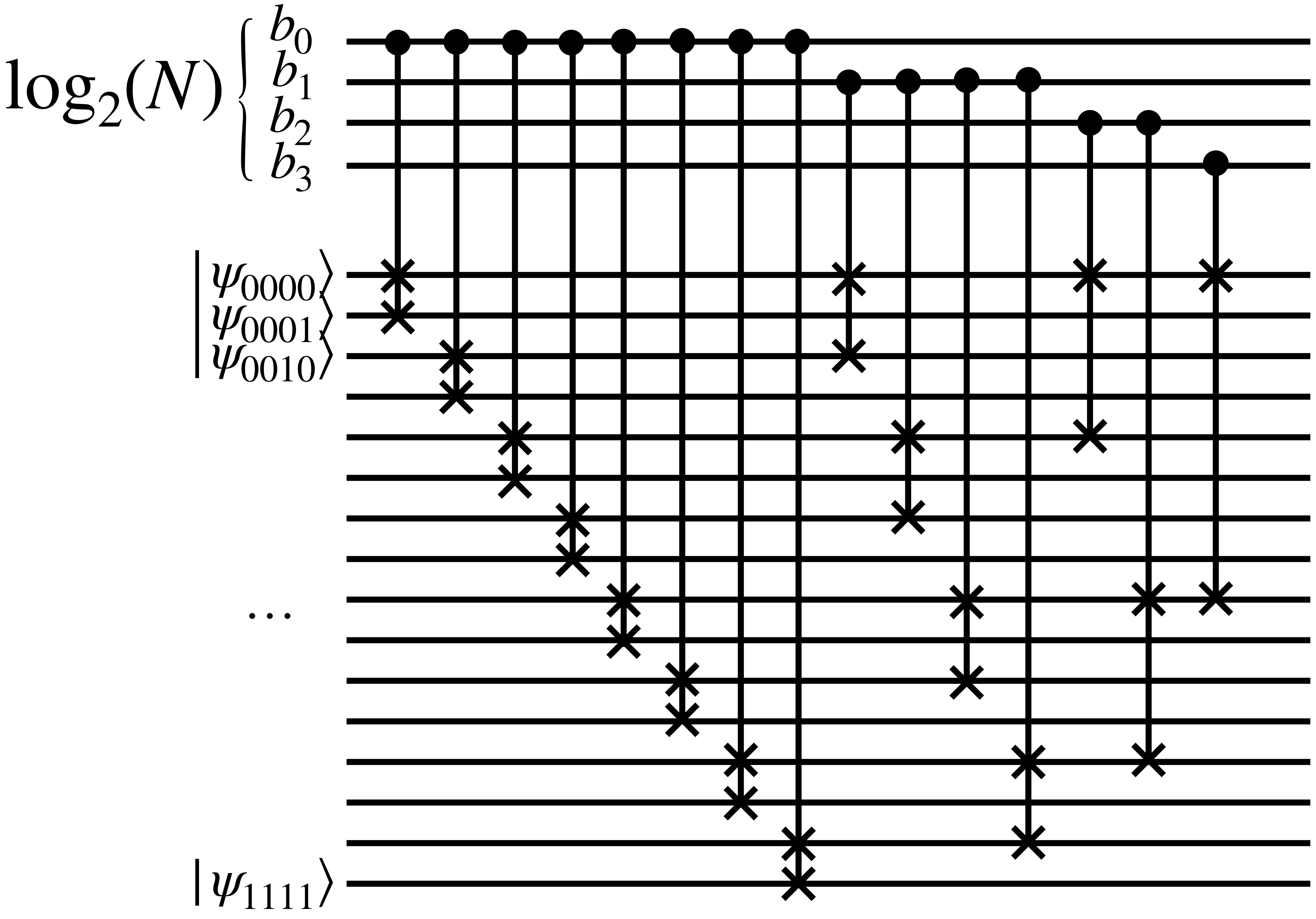


- controlled-SWAP networks are used in data look-up oracles, or quantum read only memory[1,2].
- Operationally,

$$|\vec{b}\rangle \otimes \left(|\psi_0\rangle \otimes |\psi_1\rangle \otimes \ldots \otimes |\psi_N\rangle \right)$$

[1] Low, et al. Quantum **8** (2024). [2] Motlagh, Pocrnic. [2605.20334]

Data lookup in superposition

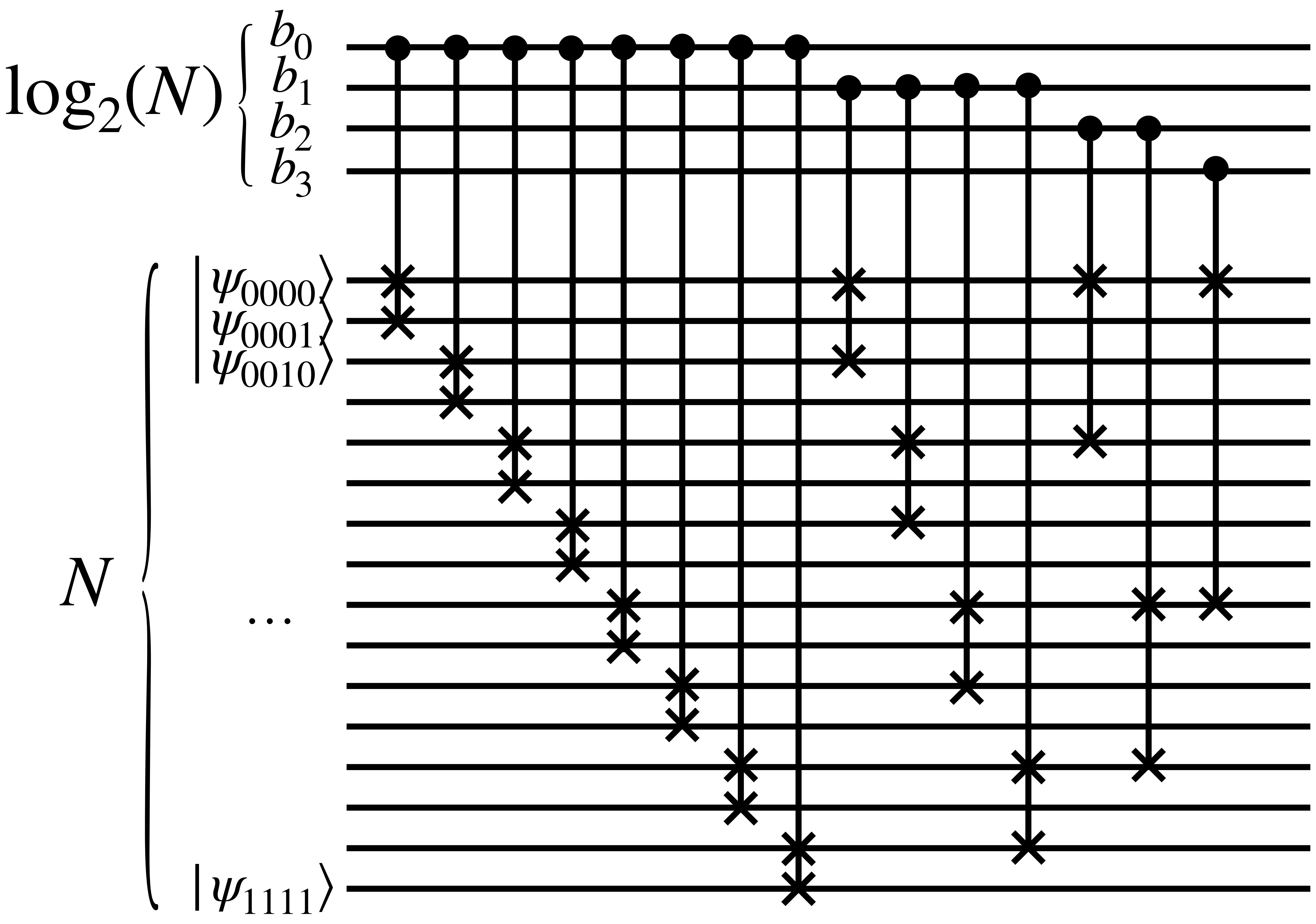


- controlled-SWAP networks are used in data look-up oracles, or quantum read only memory[1,2].
- Operationally,

$$|\vec{b}\rangle \otimes \left(|\psi_0\rangle \otimes |\psi_1\rangle \otimes \ldots \otimes |\psi_N\rangle \right)$$

[1] Low, et al. Quantum **8** (2024). [2] Motlagh, Pocrnic. [2605.20334]

Data lookup in superposition



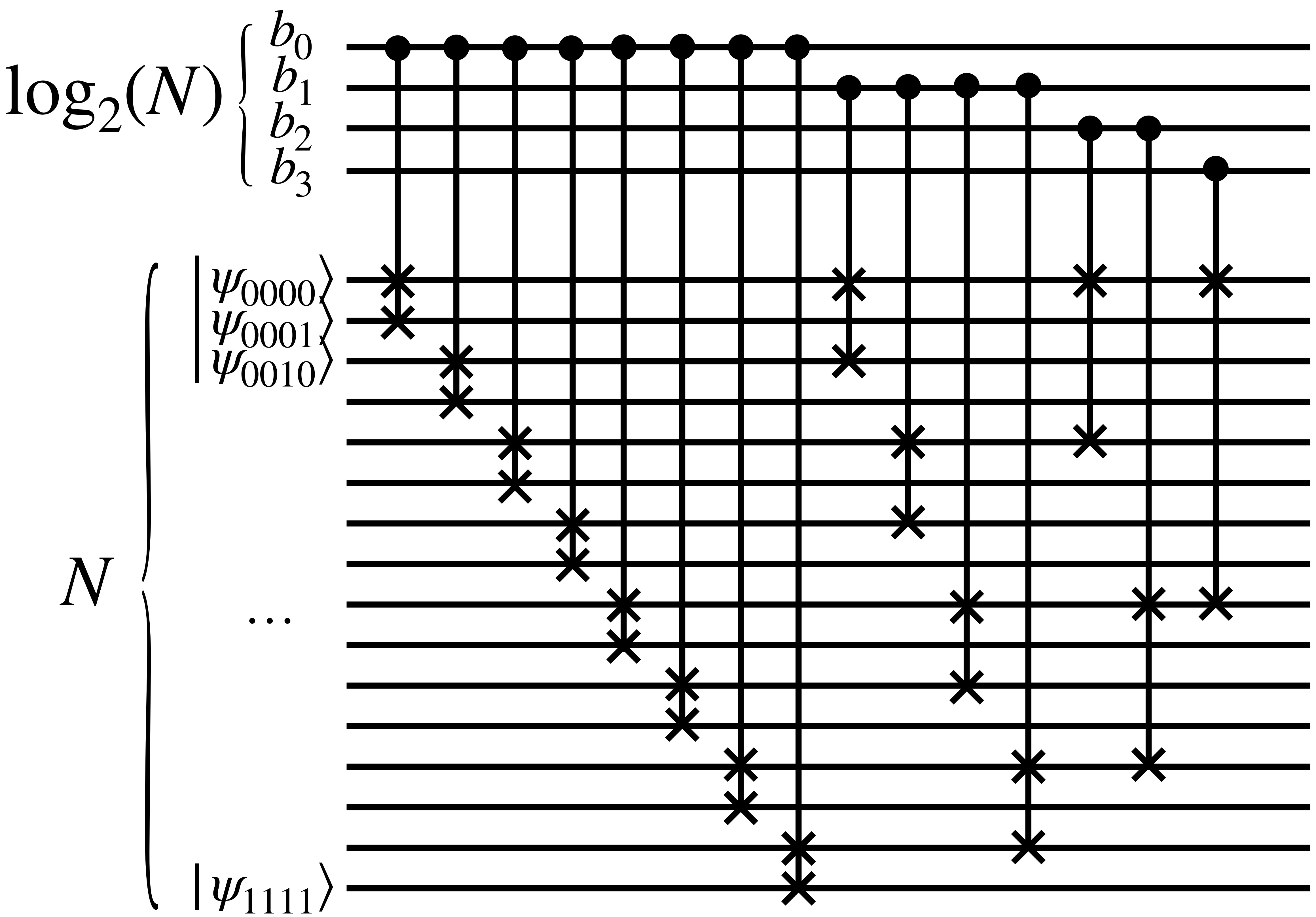
- controlled-SWAP networks are used in data lookup oracles, or quantum read only memory[1,2].
- Operationally,

$$|\vec{b}\rangle \otimes \left(|\psi_0\rangle \otimes |\psi_1\rangle \otimes \ldots \otimes |\psi_N\rangle \right)$$

---

[1] Low, et al. Quantum **8** (2024). [2] Motlagh, Pocrnic. [2605.20334]

Data lookup in superposition

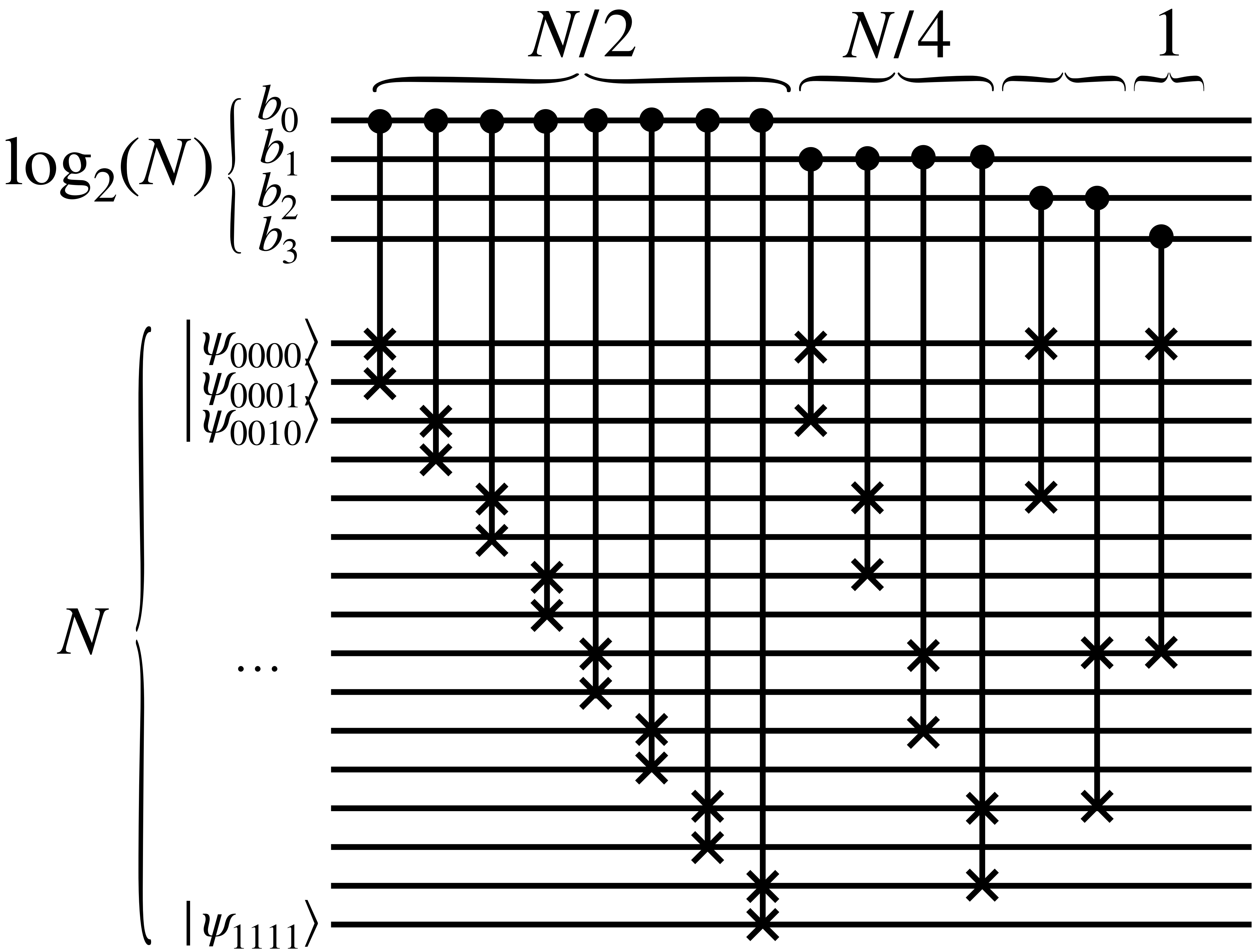


- controlled-SWAP networks are used in data look-up oracles, or quantum read only memory[1,2].
- Operationally,

$$|\vec{b}\rangle \otimes \left(|\psi_0\rangle \otimes |\psi_1\rangle \otimes \ldots \otimes |\psi_N\rangle \right)$$
$$\to |\vec{b}\rangle \otimes |\psi_{\vec{b}}\rangle \otimes \ldots$$

[1] Low, et al. Quantum **8** (2024). [2] Motlagh, Pocrnic. [2605.20334]

Data lookup in superposition

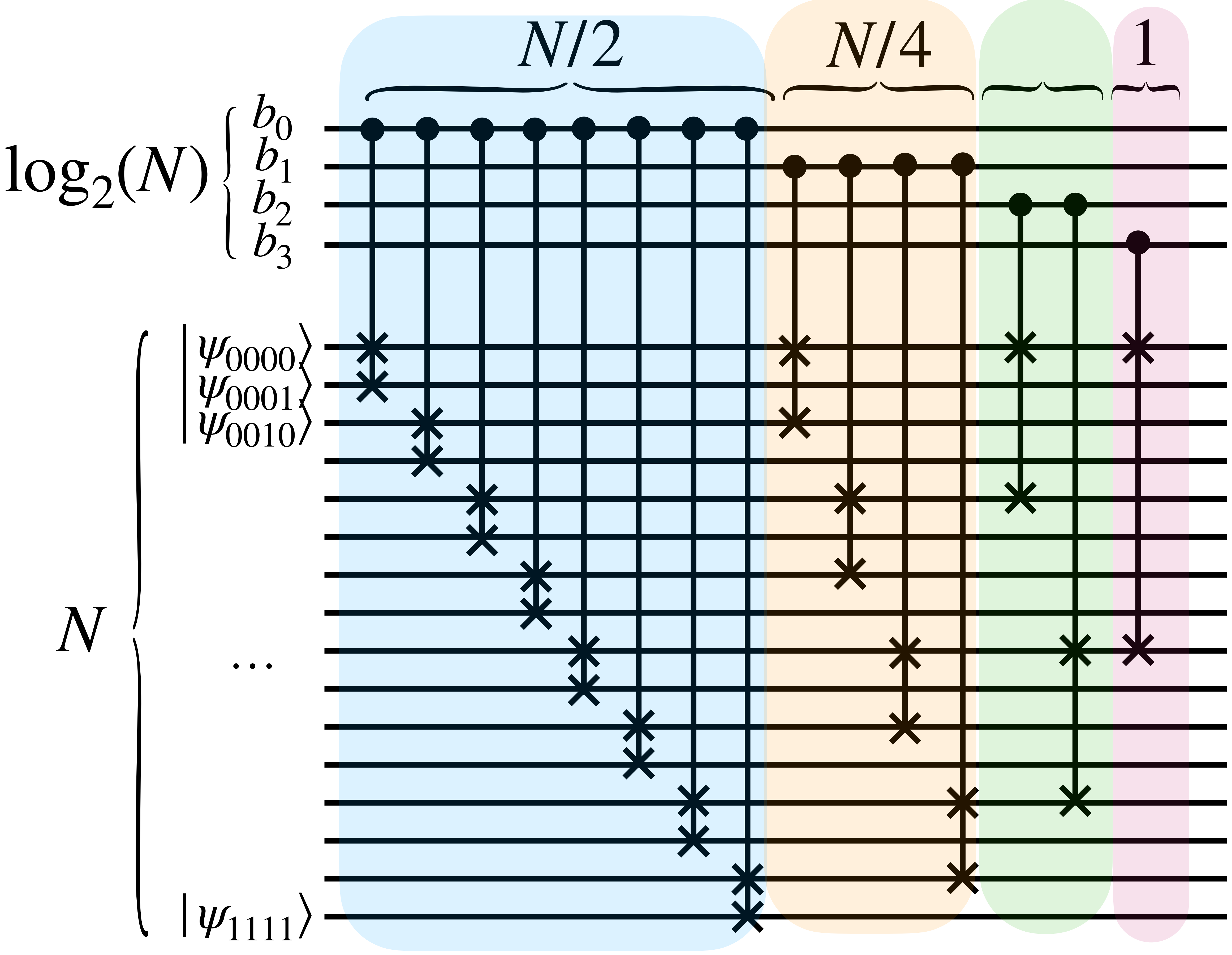


- controlled-SWAP networks are used in data look-up oracles, or quantum read only memory[1,2].
- Operationally,

$$|\vec{b}\rangle \otimes \left(|\psi_0\rangle \otimes |\psi_1\rangle \otimes \ldots \otimes |\psi_N\rangle \right)$$
$$\rightarrow |\vec{b}\rangle \otimes |\psi_{\vec{b}}\rangle \otimes \ldots$$

[1] Low, et al. Quantum **8** (2024). [2] Motlagh, Pocrnic. [2605.20334]

Data lookup in superposition

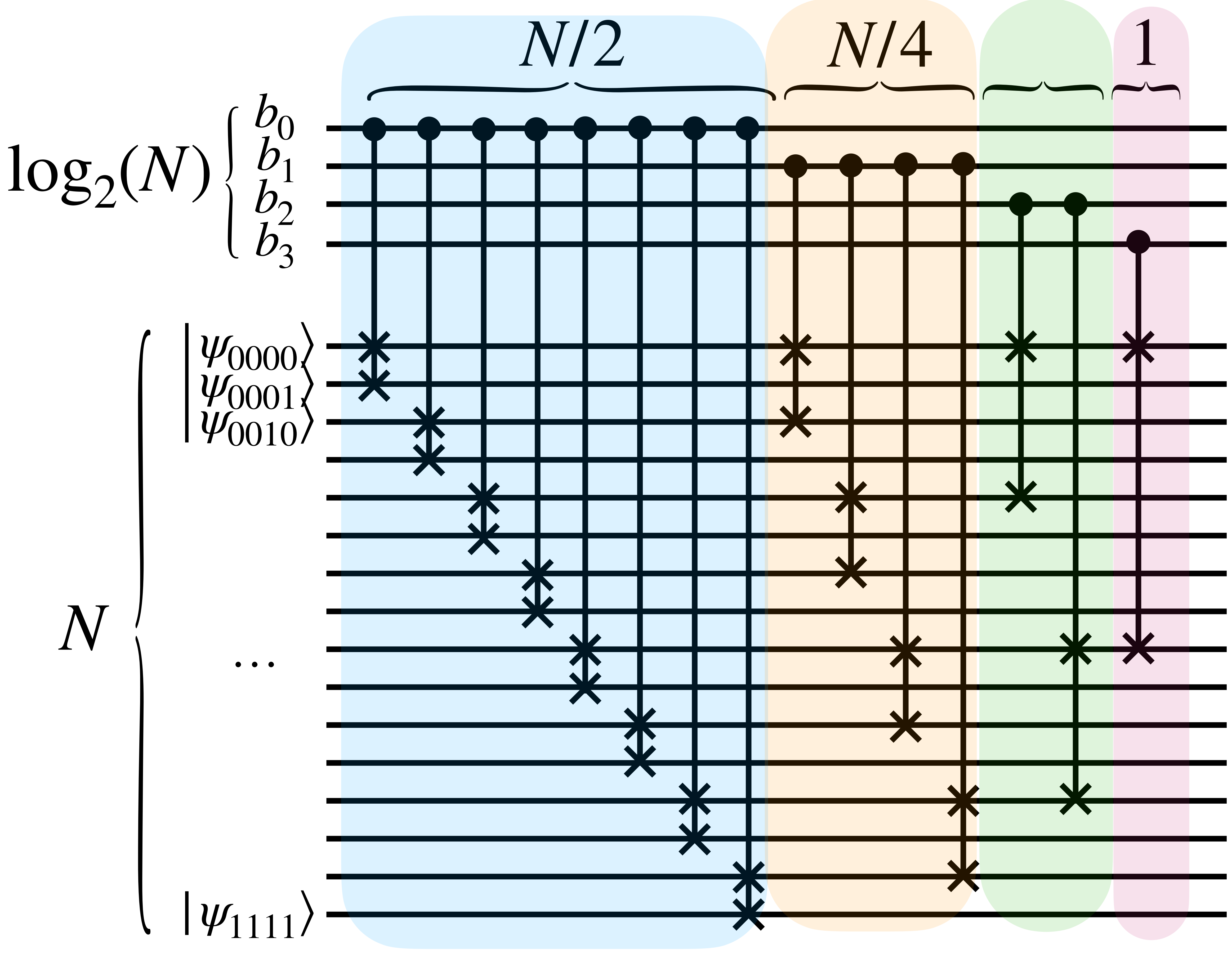


- controlled-SWAP networks are used in data look-up oracles, or quantum read only memory[1,2].
- Operationally,

$$|\vec{b}\rangle \otimes \big(|\psi_0\rangle \otimes |\psi_1\rangle \otimes \ldots \otimes |\psi_N\rangle\big)$$
$$\to |\vec{b}\rangle \otimes |\psi_{\vec{b}}\rangle \otimes \ldots$$

[1] Low, et al. Quantum **8** (2024). [2] Motlagh, Pocrnic. [2605.20334]

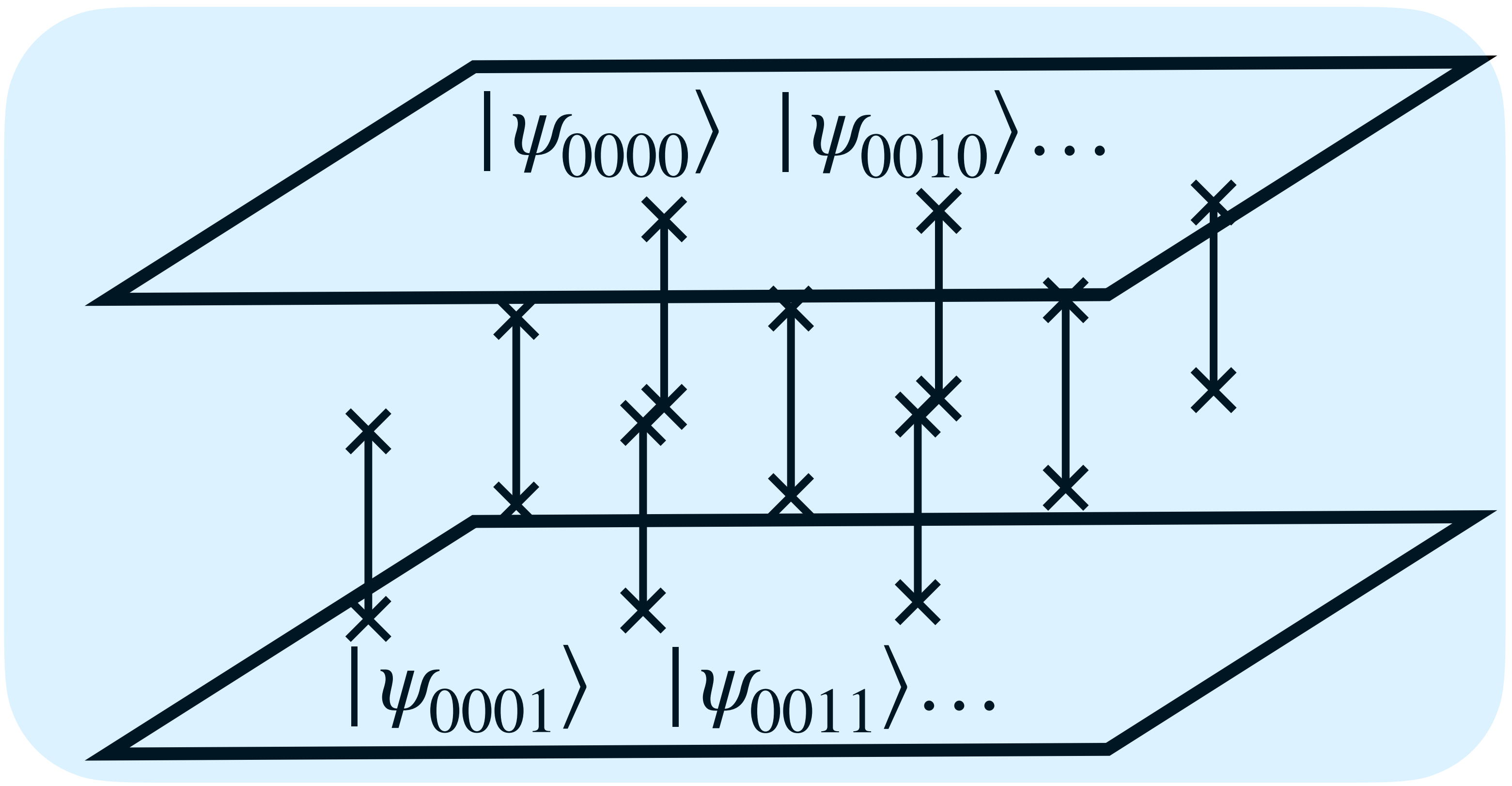
Data lookup in superposition



- controlled-SWAP networks are used in data look-up oracles, or quantum read only memory[1,2].
- Operationally,

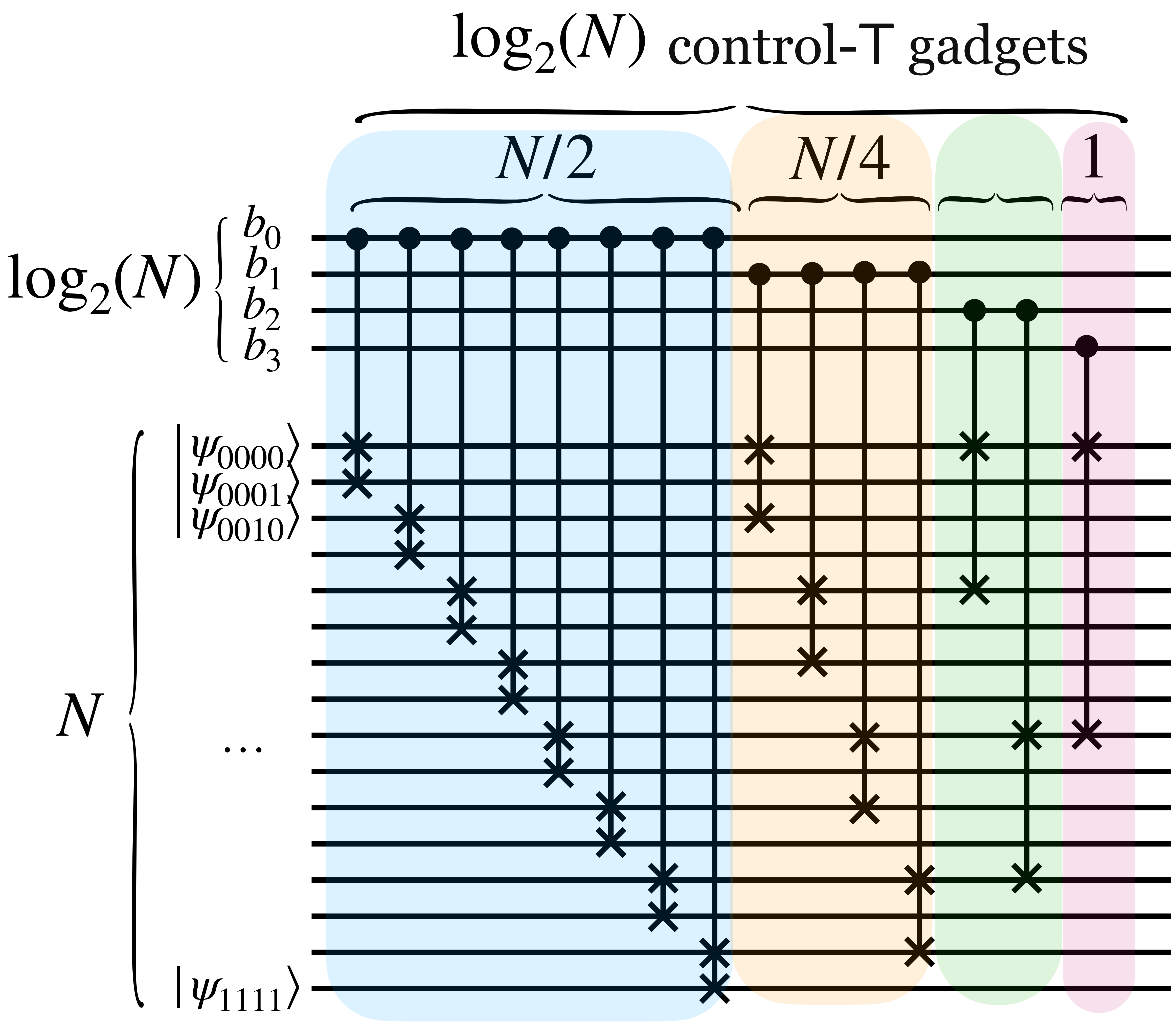
$$|\vec{b}\rangle \otimes \left(|\psi_0\rangle \otimes |\psi_1\rangle \otimes \ldots \otimes |\psi_N\rangle \right)$$
$$\to |\vec{b}\rangle \otimes |\psi_{\vec{b}}\rangle \otimes \ldots$$

- Using our Ctrl-T gadget, gauging transversal $SWAP^{\otimes k}$ on two copies of the same code



[1] Low, et al. Quantum **8** (2024). [2] Motlagh, Pocrnic. [2605.20334]

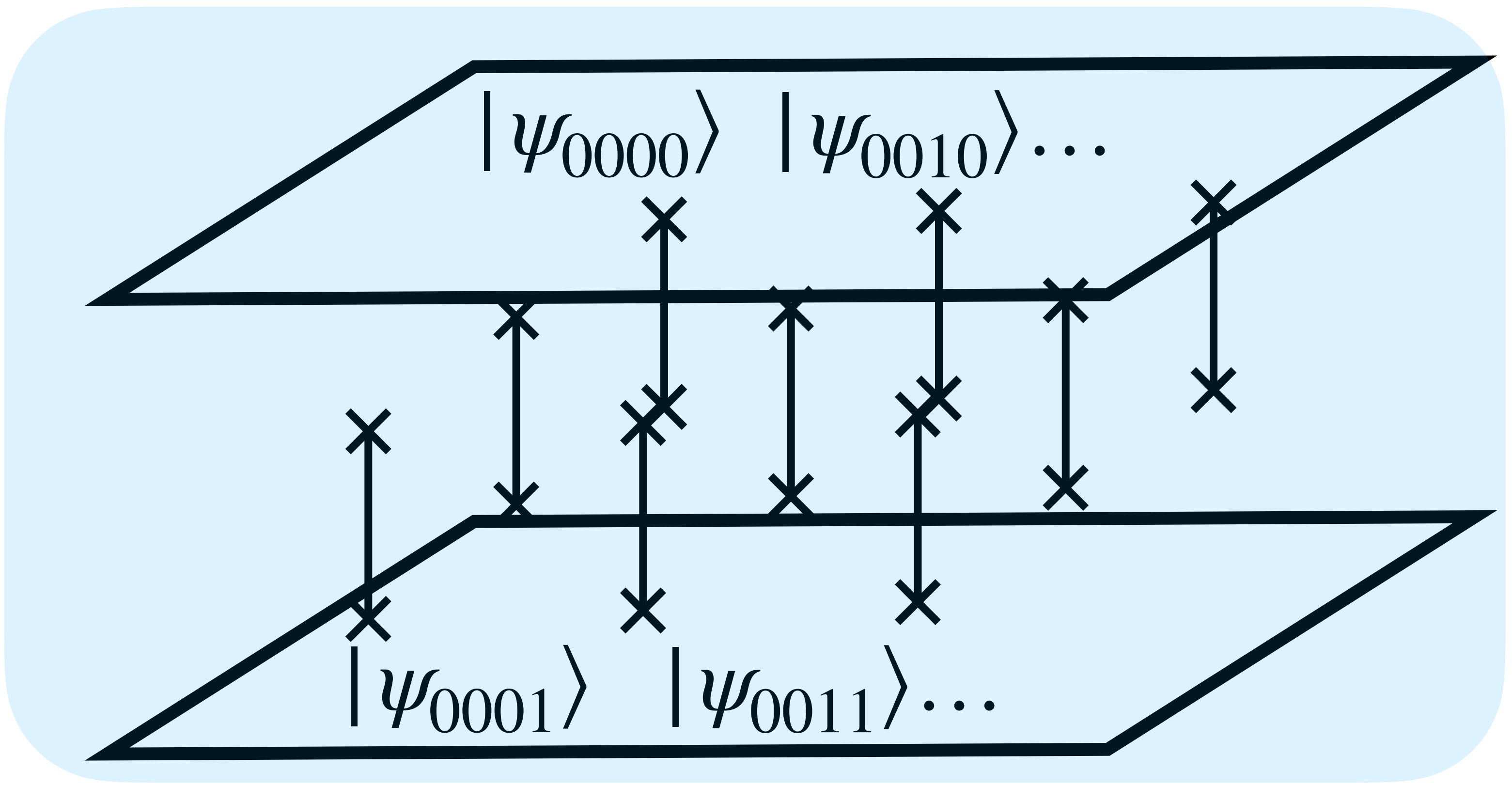
Data lookup in superposition



- controlled-SWAP networks are used in data look-up oracles, or quantum read only memory[1,2].
- Operationally,

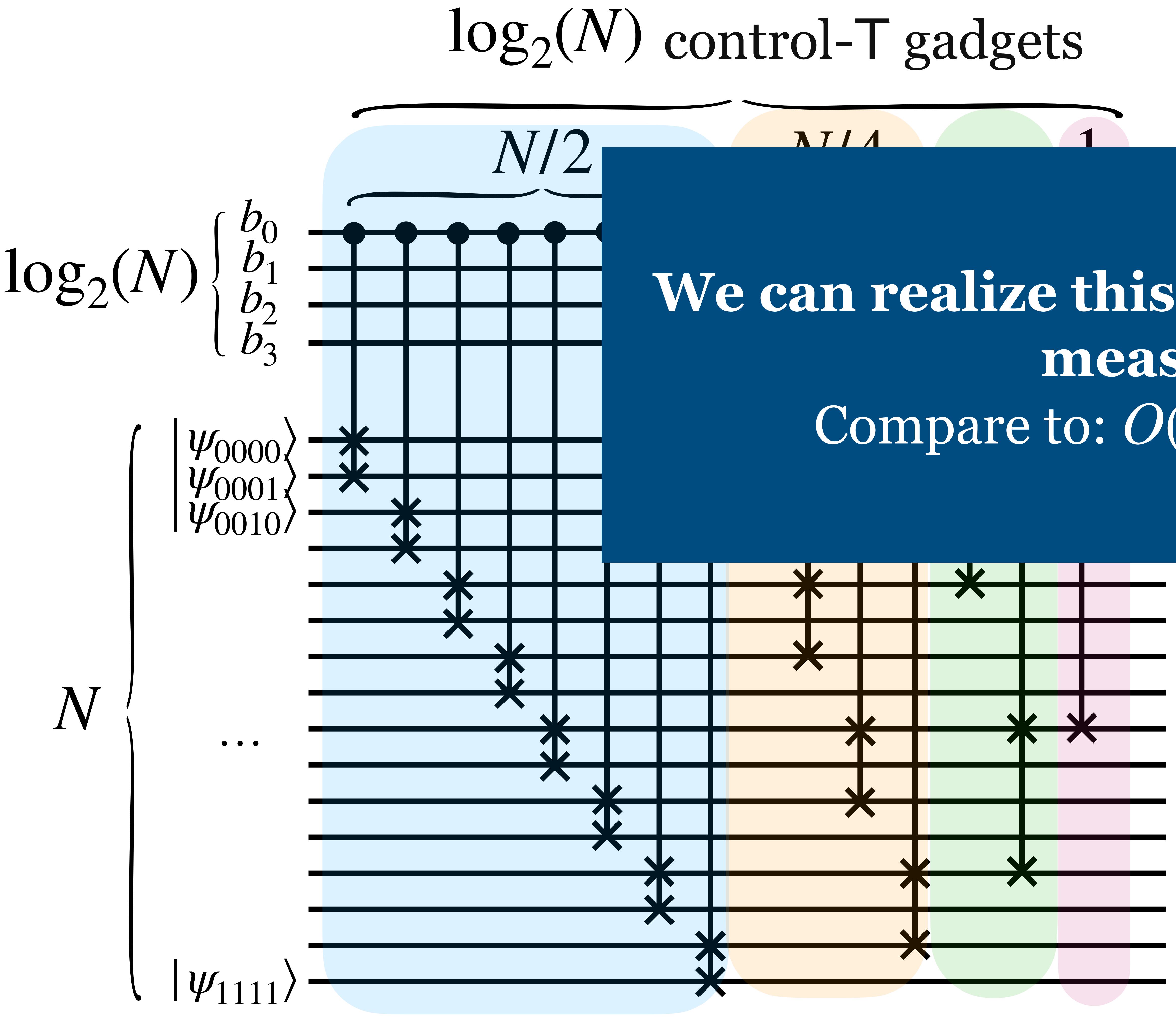
$$|\vec{b}\rangle \otimes \left(|\psi_0\rangle \otimes |\psi_1\rangle \otimes \ldots \otimes |\psi_N\rangle \right)$$
$$\rightarrow |\vec{b}\rangle \otimes |\psi_{\vec{b}}\rangle \otimes \ldots$$

- Using our Ctrl-T gadget, gauging transversal $SWAP^{\otimes k}$ on two copies of the same code



[1] Low, et al. Quantum **8** (2024). [2] Motlagh, Pocrnic. [2605.20334]

Data lookup in superposition

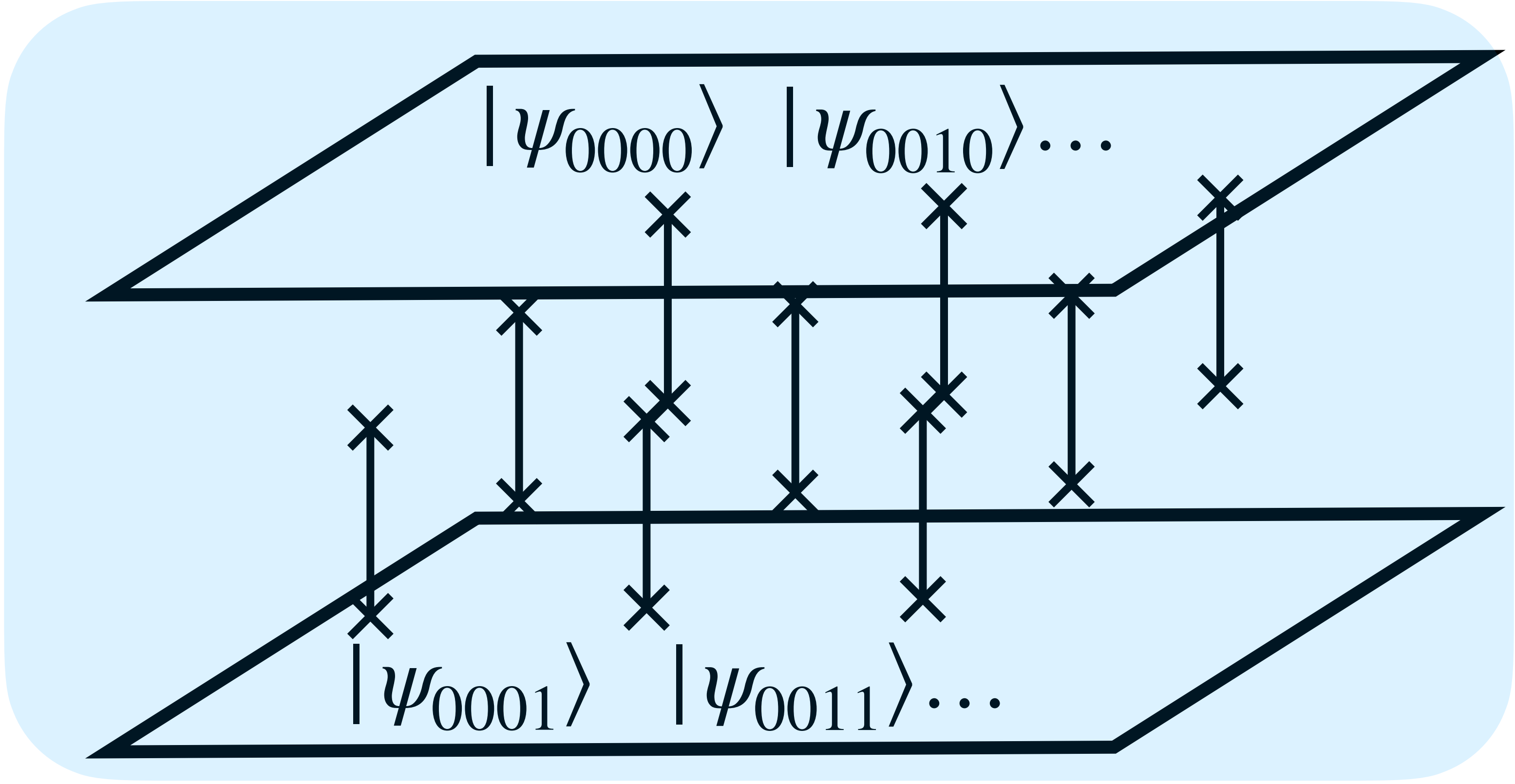


- controlled-SWAP networks are used in data look-up oracles, or quantum read only memory[1,2].

$\ldots_1\rangle \otimes \ldots \otimes |\psi_N\rangle\big)$

We can realize this with just $\log_2(N)$ Clifford measurements!
Compare to: $O(N)$ Toffolis/T gates[2].

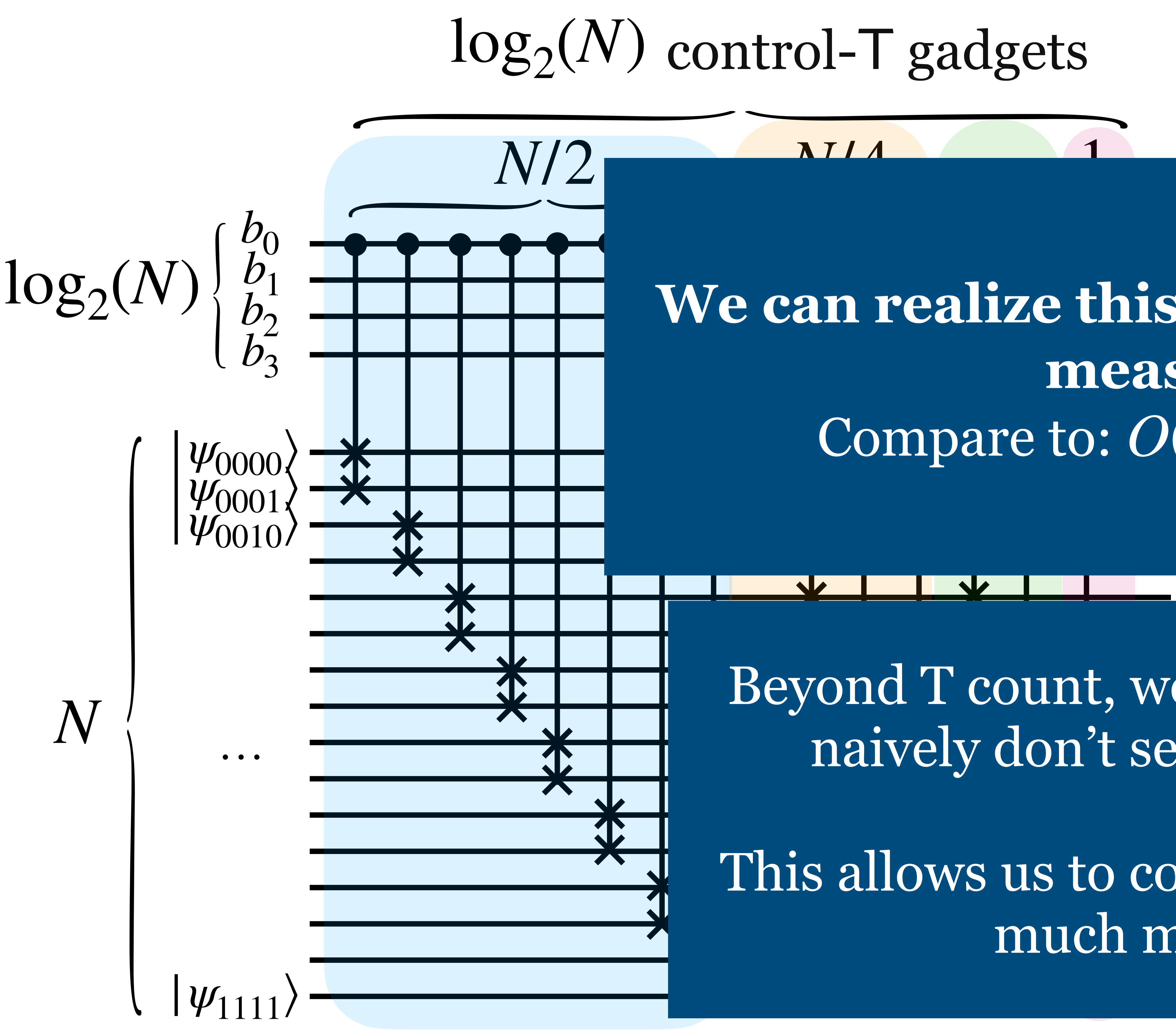
... gauging transversal ... of the same code



- $O(N)$ CSWAPs. Each CSWAP = 1 Toffoli. Each Toffoli has a decomposition of 4 T gates[3]

[1] Low, et al. Quantum **8** (2024). [2] Motlagh, Pocrnic. [2605.20334] [3] Jones, PRA **87** (2013)

Data lookup in superposition



- controlled-SWAP networks are used in data look-up oracles, or quantum read only memory[1,2].

$_1\rangle \otimes \ldots \otimes |\psi_N\rangle)$

gauging transversal

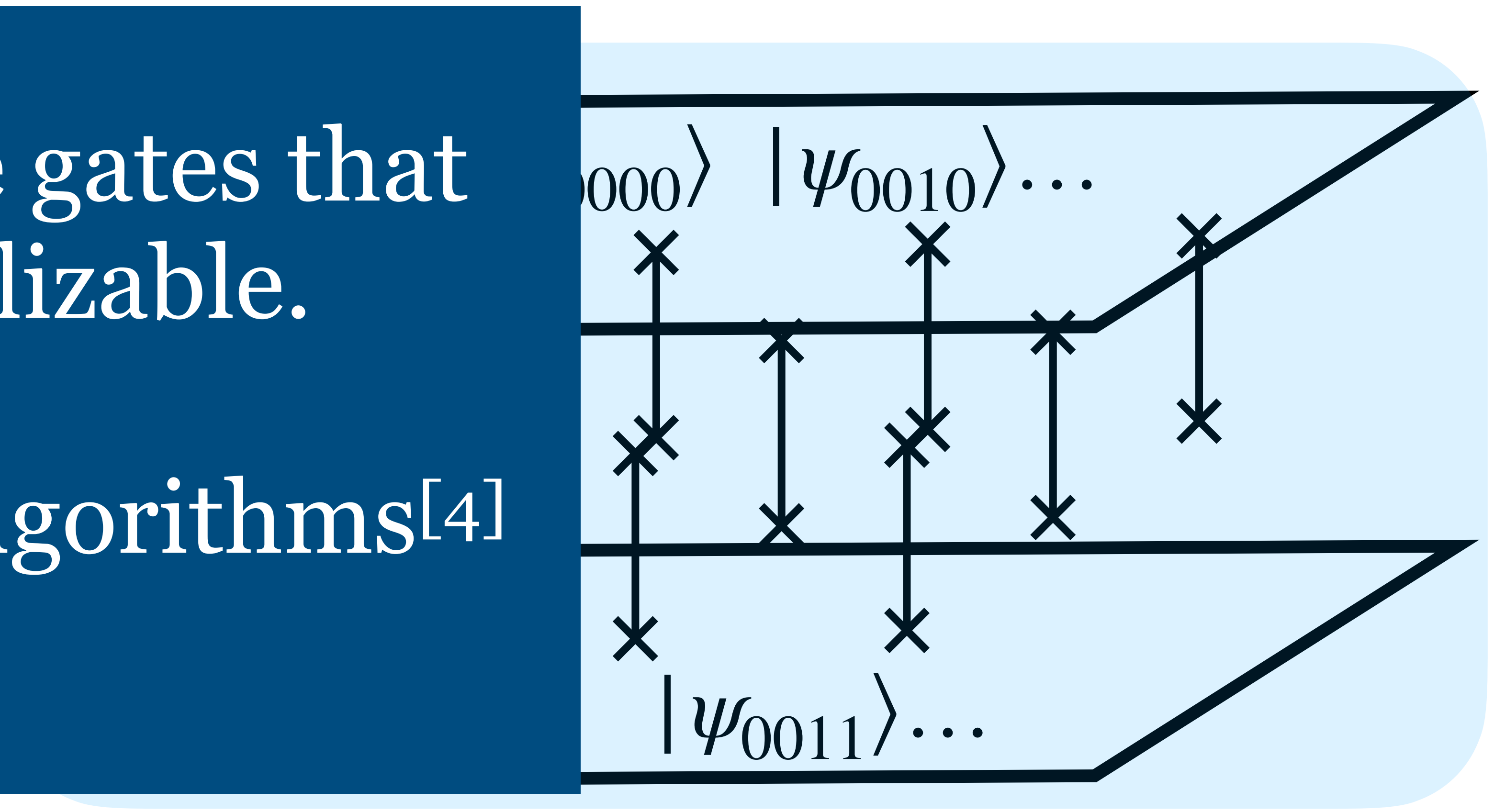
of the same code

We can realize this with just $\log_2(N)$ Clifford measurements!

Compare to: $O(N)$ Toffolis/T gates[2].

Beyond T count, we can parallelize gates that naively don't seem very parallelizable.

This allows us to compile certain algorithms[4] much more efficiently.



- $O(N)$ CSWAPs. Each CSWAP = 1 Toffoli. Each Toffoli has a decomposition of 4 T gates[3]

[1] Low, et al. Quantum **8** (2024). [2] Motlagh, Pocrnic. [2605.20334] [3] Jones, PRA **87** (2013) [4] Bouland, et al. ACM Symposium on the Theory of Computing (2025).

Summary and future work

Summary and future work

- We show a generalized gauging Clifford measurement gadget* for LDPC codes

*The gauging procedure and proof of space distance more generally applies to qudit codes with higher-order Clifford symmetries $\mathsf{T}^p = I, p > 2$

Summary and future work

- We show a generalized gauging Clifford measurement gadget* for LDPC codes
- Present a simple procedure to derive unitary, LDPC stabilizers that commute in the codespace

*The gauging procedure and proof of space distance more generally applies to qudit codes with higher-order Clifford symmetries $\mathsf{T}^p = I, p > 2$

Summary and future work

- We show a generalized gauging Clifford measurement gadget* for LDPC codes
- Present a simple procedure to derive unitary, LDPC stabilizers that commute in the codespace
- Show adversarial fault distance to measurement and data errors

*The gauging procedure and proof of space distance more generally applies to qudit codes with higher-order Clifford symmetries $\mathsf{T}^p = I, p > 2$

Summary and future work

- We show a generalized gauging Clifford measurement gadget* for LDPC codes
- Present a simple procedure to derive unitary, LDPC stabilizers that commute in the codespace
- Show adversarial fault distance to measurement and data errors
- Examples of utility in algorithmic primitives

*The gauging procedure and proof of space distance more generally applies to qudit codes with higher-order Clifford symmetries $\mathsf{T}^p = I, p > 2$

Summary and future work

- We show a generalized gauging Clifford measurement gadget* for LDPC codes
- Present a simple procedure to derive unitary, LDPC stabilizers that commute in the codespace
- Show adversarial fault distance to measurement and data errors
- Examples of utility in algorithmic primitives

- Extend **fault tolerance** to Pauli and measurement errors where $\mathsf{T}^p = I, p > 2$

---

*The gauging procedure and proof of space distance more generally applies to qudit codes with higher-order Clifford symmetries $\mathsf{T}^p = I, p > 2$

Summary and future work

- We show a generalized gauging Clifford measurement gadget* for LDPC codes
- Present a simple procedure to derive unitary, LDPC stabilizers that commute in the codespace
- Show adversarial fault distance to measurement and data errors
- Examples of utility in algorithmic primitives

- Extend **fault tolerance** to Pauli and measurement errors where $\mathsf{T}^p = I, p > 2$
- Just-in-time decoder with **provable finite threshold**

*The gauging procedure and proof of space distance more generally applies to qudit codes with higher-order Clifford symmetries $\mathsf{T}^p = I, p > 2$

Summary and future work

- We show a generalized gauging Clifford measurement gadget* for LDPC codes
- Present a simple procedure to derive unitary, LDPC stabilizers that commute in the codespace
- Show adversarial fault distance to measurement and data errors
- Examples of utility in algorithmic primitives

- Extend **fault tolerance** to Pauli and measurement errors where $\mathsf{T}^p = I, p > 2$
- Just-in-time decoder with **provable finite threshold**
- **Practical decoders** and numerics: in what regimes is this primitive practically competitive?[1,2]

[1] Scruby, et al. Quantum **6** (2022).
[2] Magdalena de la Fuente, et al. [2604.02033]

*The gauging procedure and proof of space distance more generally applies to qudit codes with higher-order Clifford symmetries $\mathsf{T}^p = I, p > 2$

Summary and future work

- We show a generalized gauging Clifford measurement gadget* for LDPC codes
- Present a simple procedure to derive unitary, LDPC stabilizers that commute in the codespace
- Show adversarial fault distance to measurement and data errors
- Examples of utility in algorithmic primitives

- Extend **fault tolerance** to Pauli and measurement errors where $\mathsf{T}^p = I, p > 2$
- Just-in-time decoder with **provable finite threshold**
- **Practical decoders** and numerics: in what regimes is this primitive practically competitive?[1,2]
- Compiling into other important **algorithmic primitives** besides data lookup?

[1] Scruby, et al. Quantum **6** (2022).
[2] Magdalena de la Fuente, et al. [2604.02033]

*The gauging procedure and proof of space distance more generally applies to qudit codes with higher-order Clifford symmetries $\mathsf{T}^p = I, p > 2$