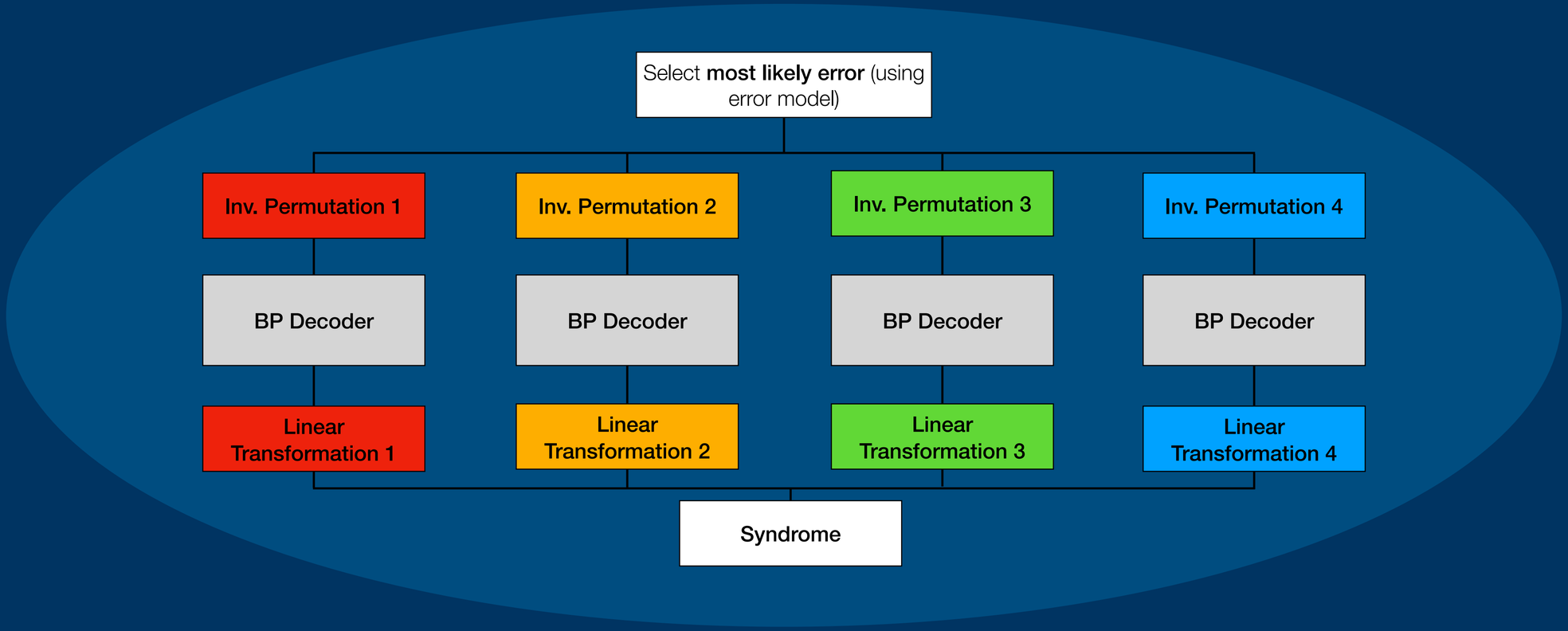
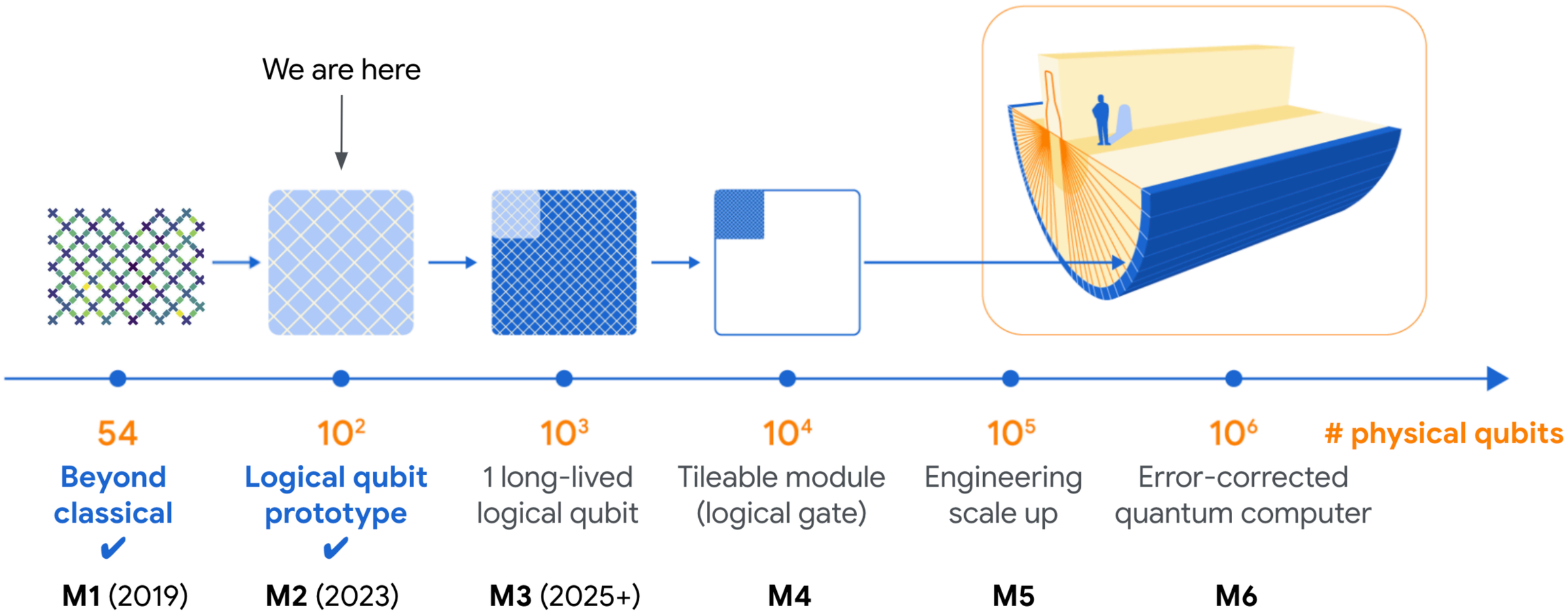
# The AutDEC Decoder

## Improving belief propagation by ensembling and code symmetry




**Dan Browne -** University College London
Joint work with **Stergios Koutsioumpas**, **Hasan Sayginel**, **Mark Webster**

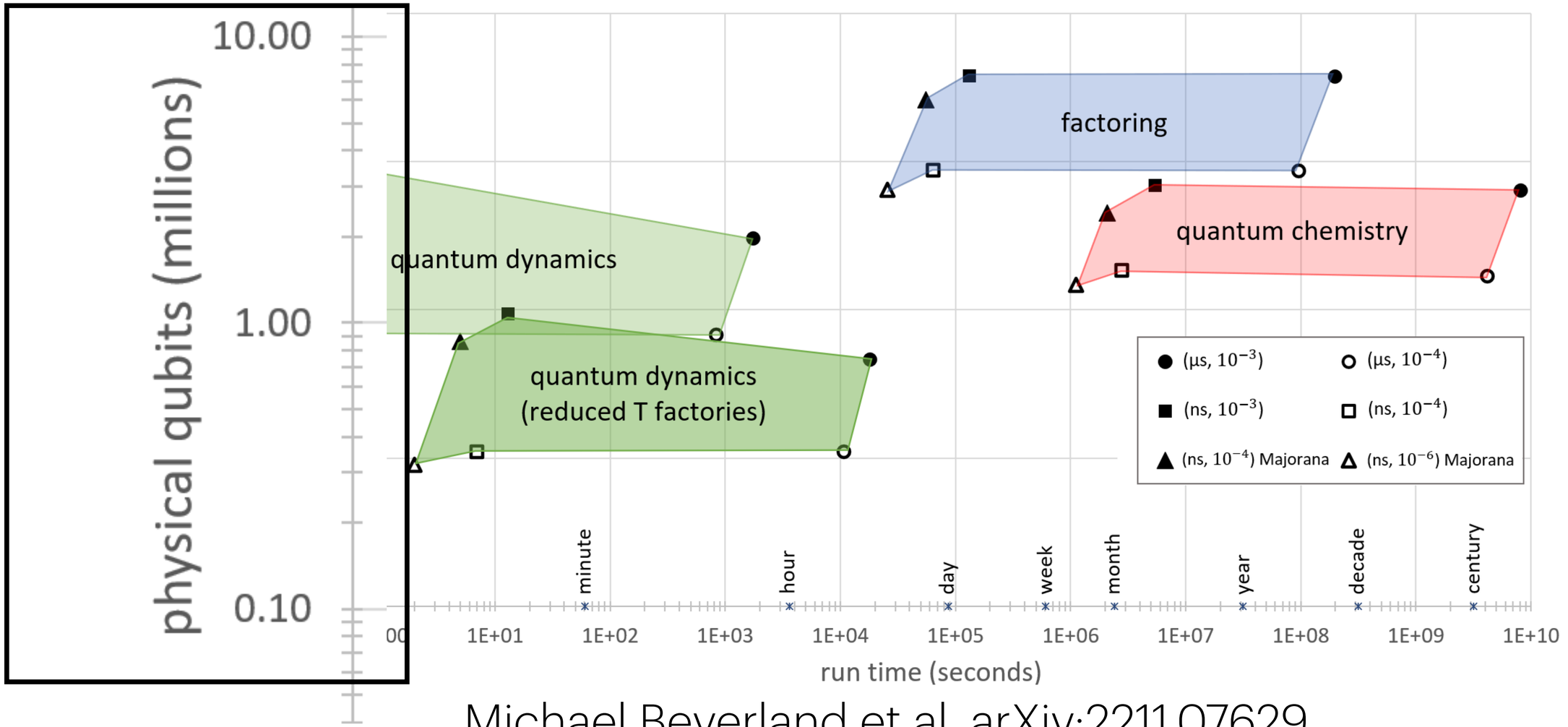
# Background - Google roadmap from 2023




https://quantumai.google/qecmilestone

# Surface Code resource estimates

- The most mature model of fault-tolerant quantum computing, but overheads can be extremely high.




Michael Beverland et al, arXiv:2211.07629

# Low-density Parity-Check Codes

- In the classical world, we use **high rate LDPC codes**.
- **Low-density:**
  - each bit involved in **few parity checks**
  - each check involves **few bits**
- High performance "**good codes**" exist:

$$k \propto n$$
$$d \propto n$$

- Efficient decoding via **belief propagation**.

# Quantum LDPC Codes

- **Low-density:**
  - each qubit involved in **few checks**
  - each check involves **few qubits**

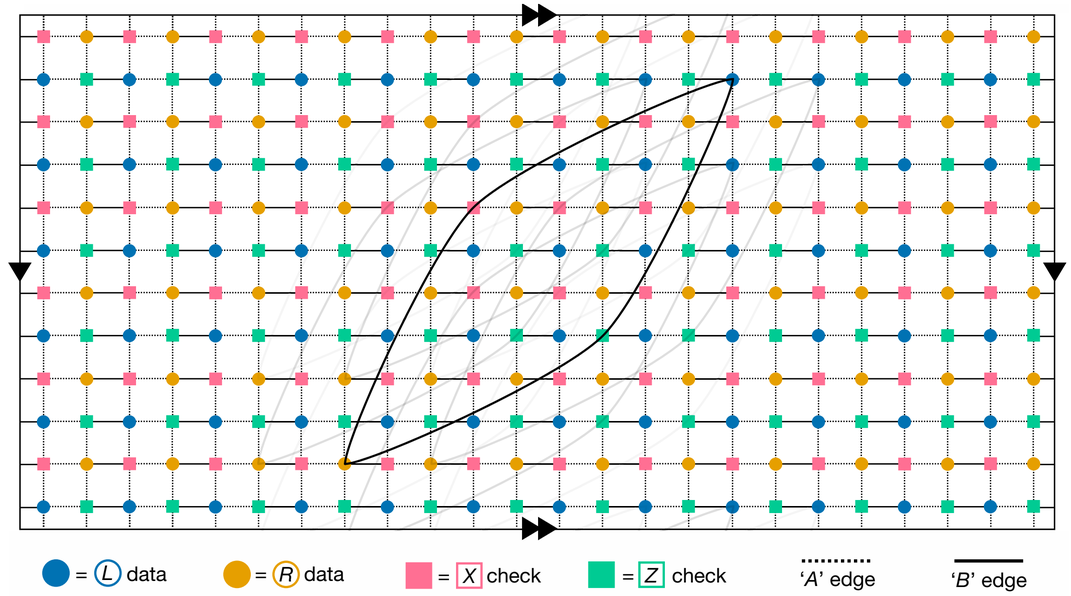
- High performance "**good codes**" discovered in 2022:

$$k \propto n$$
$$d \propto n$$

- **Hypergraph product**, **lifted product** and **balanced product** → many novel codes with interesting properties appearing every week on arXiv!

P. Panteleev, G. Kalachev, STOC '22

- **E.g. Bivariate Bicycle codes**
  - (144,12,12) "Gross code"




S. Bravyi et al, Nature 2024

# Belief propagation + Ordered Statistics Decoding (BP+OSD)

- **BP** can be readily adapted for quantum codes but often **fails to converge** to a valid correction.
- **OSD** adds a **second decoder stage** using BP's output to focus decoding problem on most likely errors.

## BP+OSD: A decoder for quantum LDPC codes

A Python library implementing belief propagation with ordered statistics post-processing for decoding sparse quantum LDPC codes as described in arXiv:2005.07016. Note, this library has recently been completly rewritten using Python and Cython. The bulk of the code now resides in the LDPC repository. The original C++ version can be found in the `cpp_version` branch of this repository.

- **BP** can be parallelised to time **linear** in **n.**
- **OSD** uses Gaussian elimination and is often the **bottleneck**.
- Improvements to OSD have been made* with more parallel implementations and reduced runtime (**LSD**, **Ambiguity Clustering)**.
- But it would be advantageous if quantum LDPC codes could, like classical codes, be decoded reliably with **BP** alone.

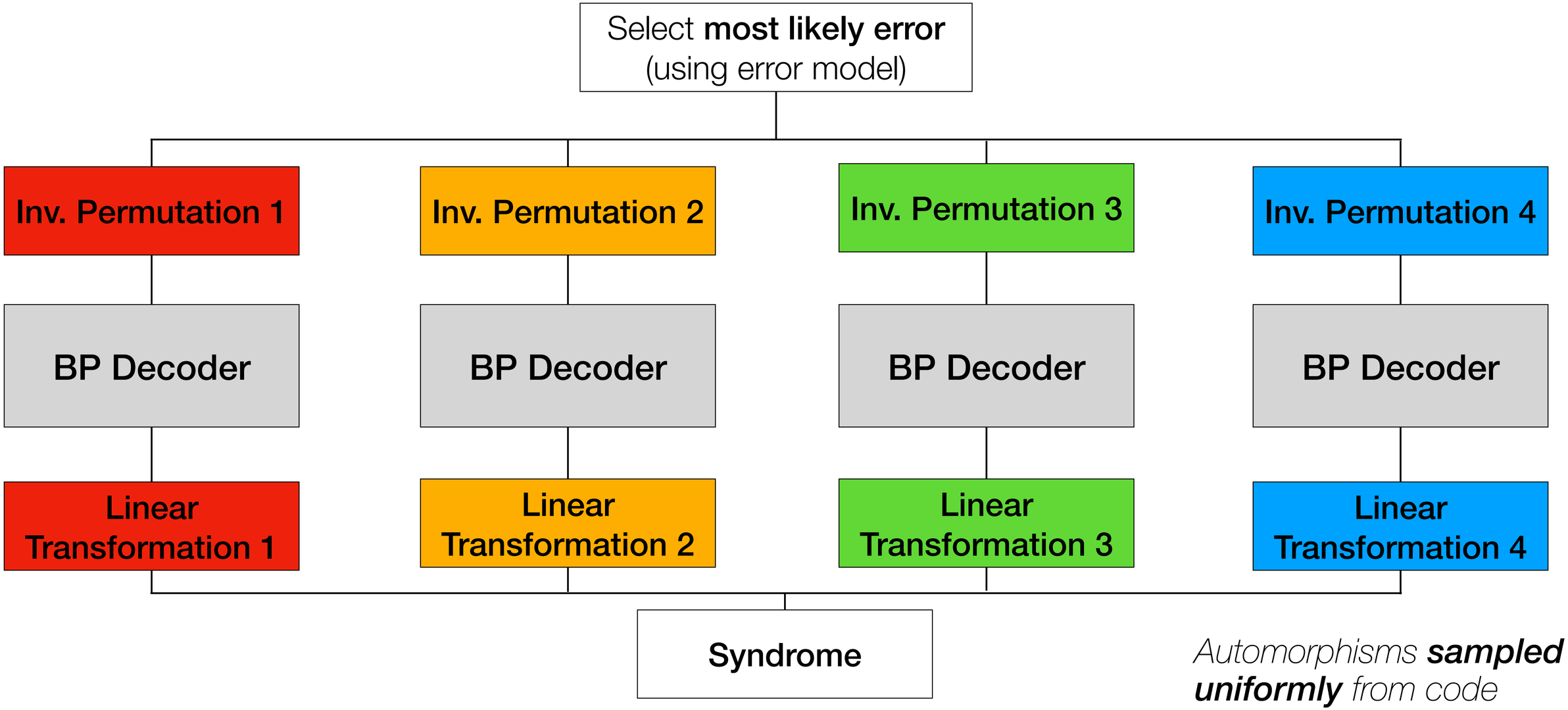
P. Panteleev, G. Kalachev, 2019

J. Roffe, https://pypi.org/project/ldpc/

*Wolanski et al, arxiv.org/2406.14527

*Hillman et al, arxiv.org/2406.18655

# The AutDec Decoder




*Automorphisms **sampled uniformly** from code automorphism group*

Koutsioumpas et al, quant-ph/2503.01738

# Talk structure

- Decoding Classical Codes and **Belief Propagation**
- Belief Propagation on **Quantum Codes**
- **Ensemble** Decoders
- Code **Automorphisms**
- The **AutDec** Decoder
- Numerical results for **small codes** and circuit level noise on **qLDPC codes**
- **Summary** and **Outlook**

# Decoding (classical) linear codes

- Example: [7,4,3] Hamming code
- Code-words:

$$c_1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$c_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$
$$c_3 = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$
$$c_4 = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

- Message:

$$m = \sum_j a_j c_j$$

*Bit-wise addition mod 2*

- Errors detected via parity checks

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

*Parity-check matrix*

- No errors:

$$Hc^{\mathsf{T}} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

# Decoding (classical) linear codes

- Error represented as bit string:

$$e = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- Received message:

$$r = m + e$$

- Syndrome:

$$\sigma = Hr^{\mathsf{T}} = He^{\mathsf{T}} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

- Aim of the **decoder**:
  - Given the syndrome $\sigma$ compute the error $e$
- We then can correct the error.

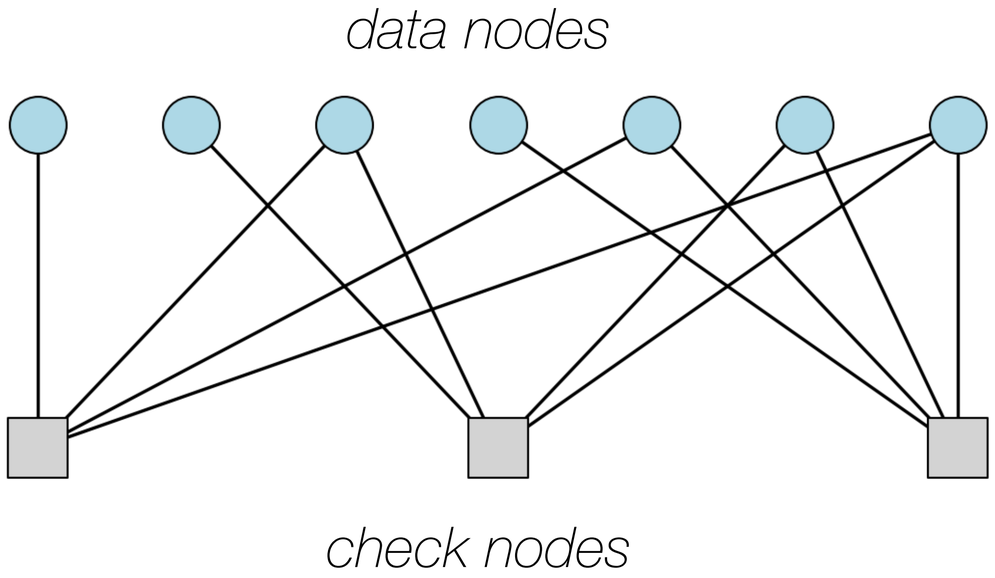
# Belief Propagation

- A message-passing decoder on the Tanner graph.

  E.g. 7-bit Hamming code:

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Tanner graph

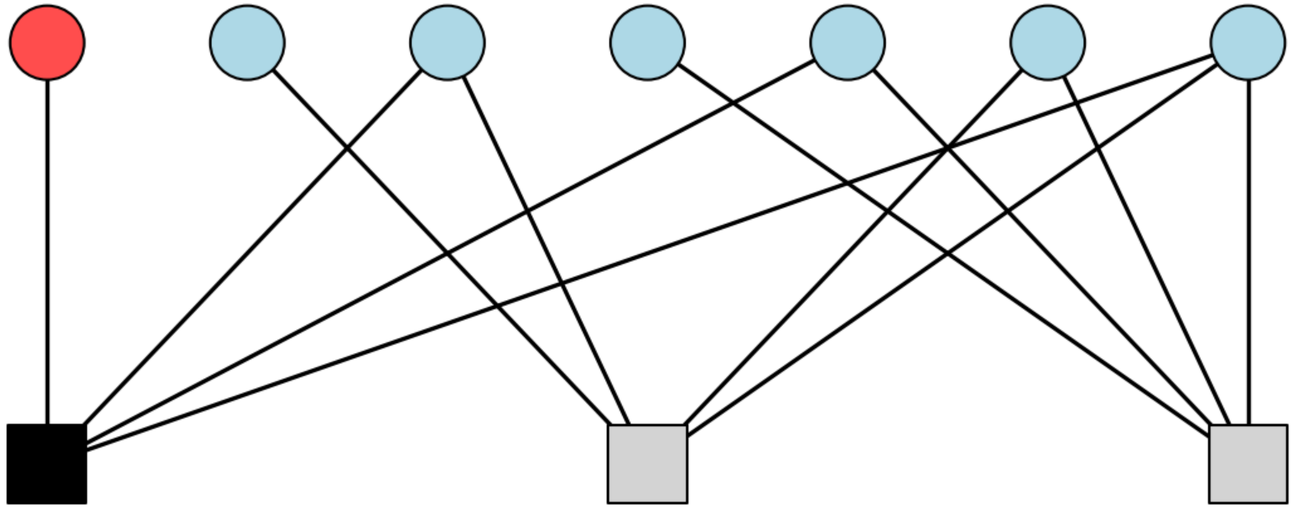



- Nodes **pass messages** back and forth, to establish marginal prob. of error on each bit.
- Finally, these probabilities are converted to a “**hard decision**” - a specific error.

# Belief Propagation

- Example:
  - error on first qubit
  - error prob. 0.05

*Data nodes initialised with the prior probabilities*

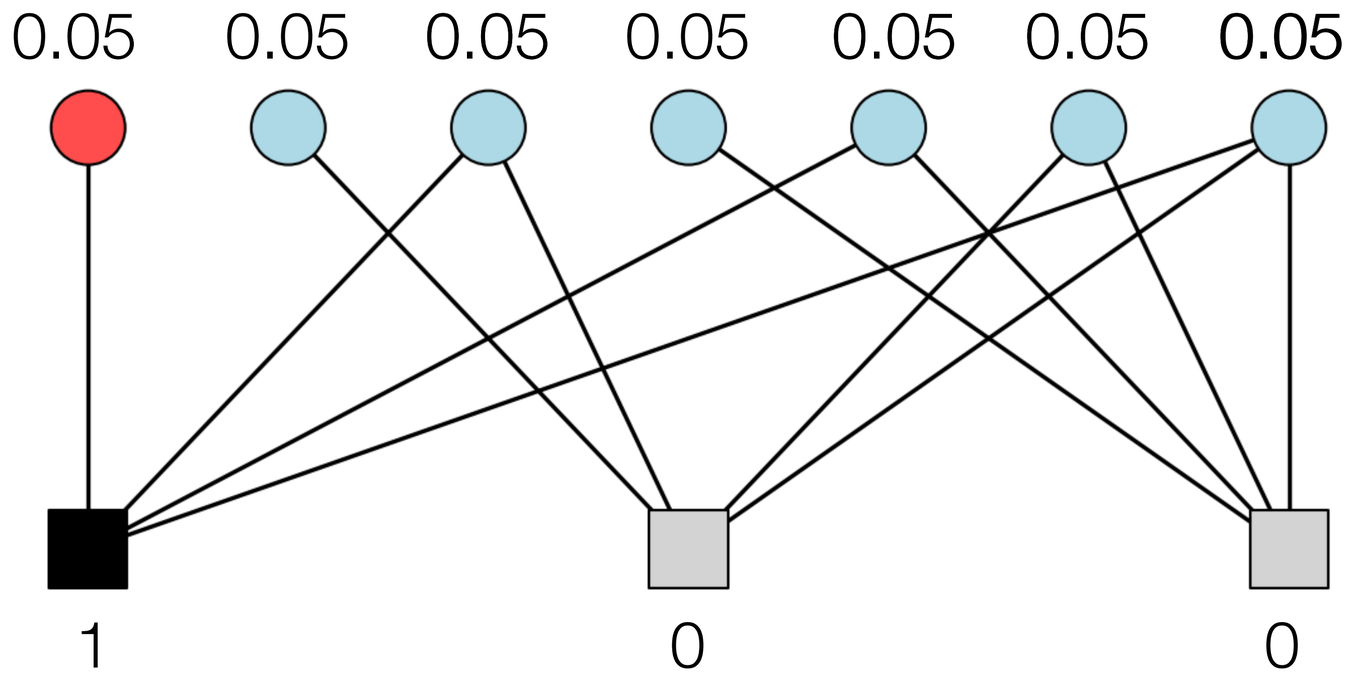


*Check nodes know their respective syndrome bit*

# Belief Propagation

- Example:
  - error on first qubit
  - error prob. 0.05

*Data nodes initialised with the prior probabilities*

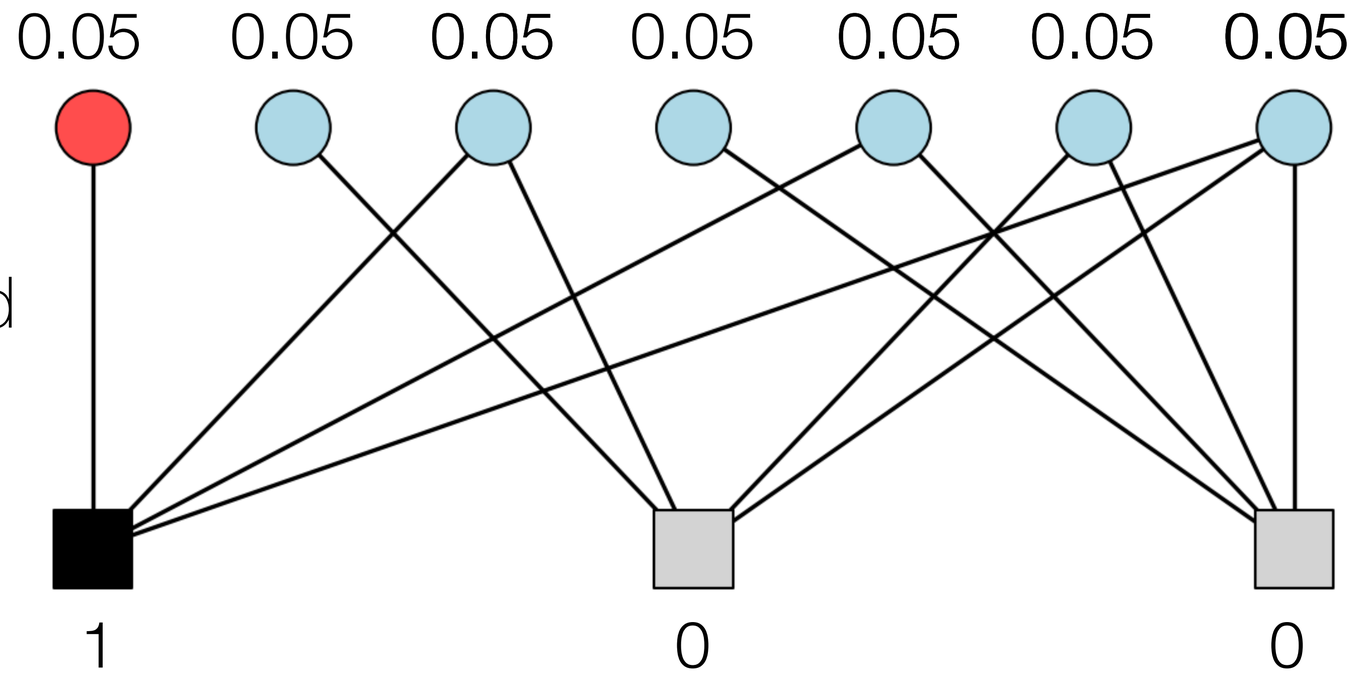



*Check nodes know their respective syndrome bit*

# Belief Propagation

*Data nodes initialised with the prior probabilities*

Date nodes **send** their prior probabilities to their connected check nodes.



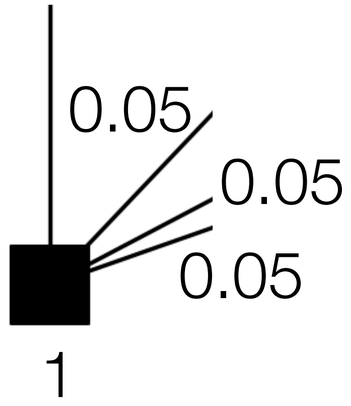

*Check nodes know their respective syndrome bit*

# Belief Propagation

Check nodes use

the **received probabilities** and their **syndrome bit**

to estimate a **new error probability** for each data bit.




E.g. Given that :

the overall parity is odd

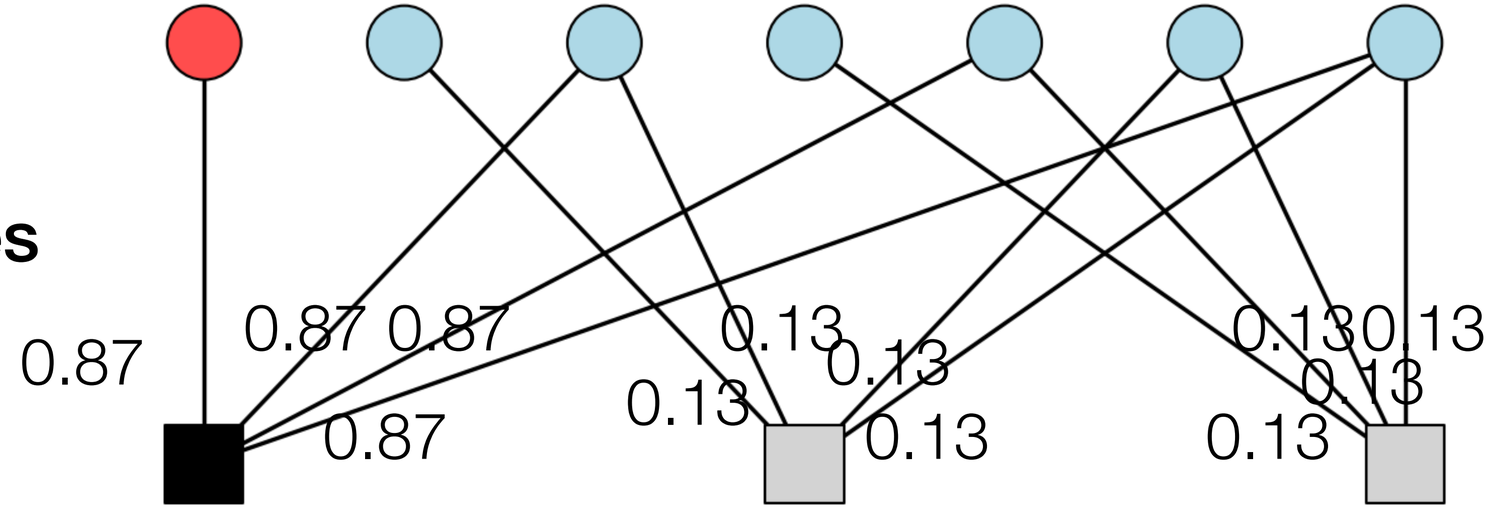
the prior probs of other bits are all 0.05

Prob of an error on the first bit is:

$$\frac{1+(1-2\times 0.05)^3}{2}=0.87$$

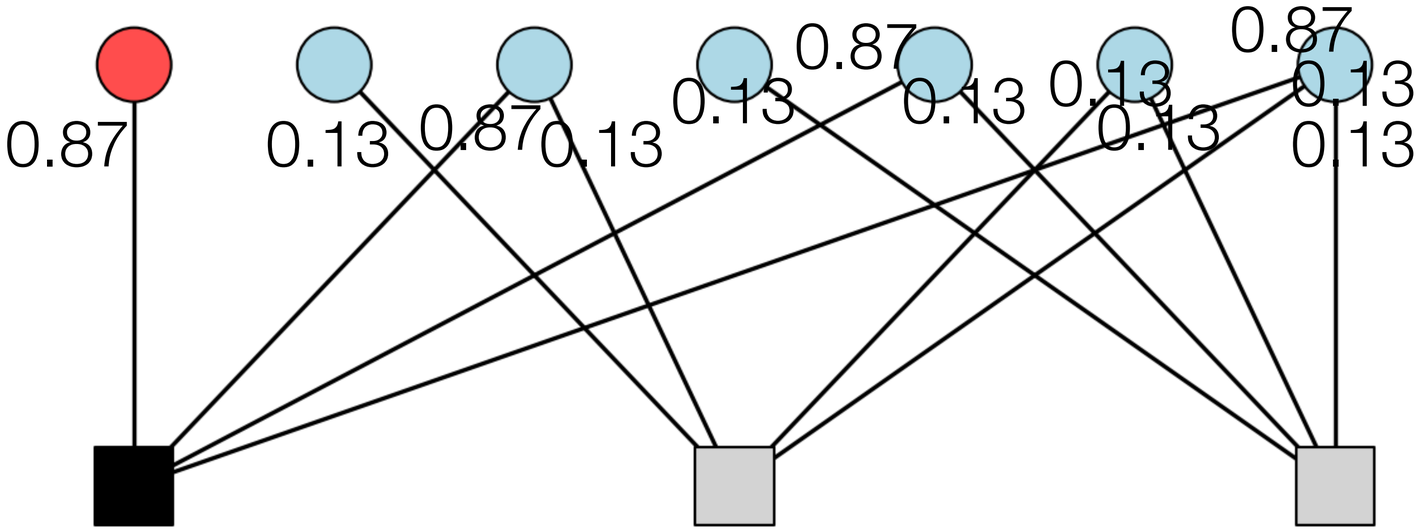
# Belief Propagation

Check nodes send the **computed probabilities** to the data nodes.



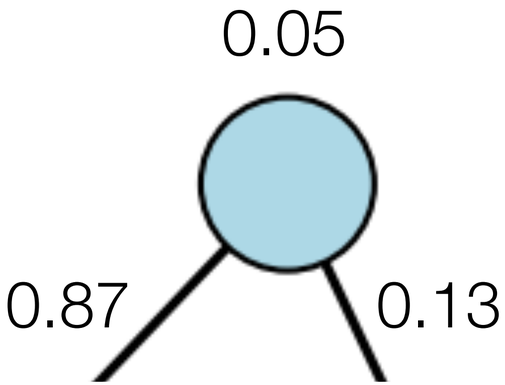
# Belief Propagation

Check nodes send the **computed probabilities** to the data nodes.



# Belief Propagation

Data nodes use the received information to estimate their **new error probability**




*multiplying and normalising*

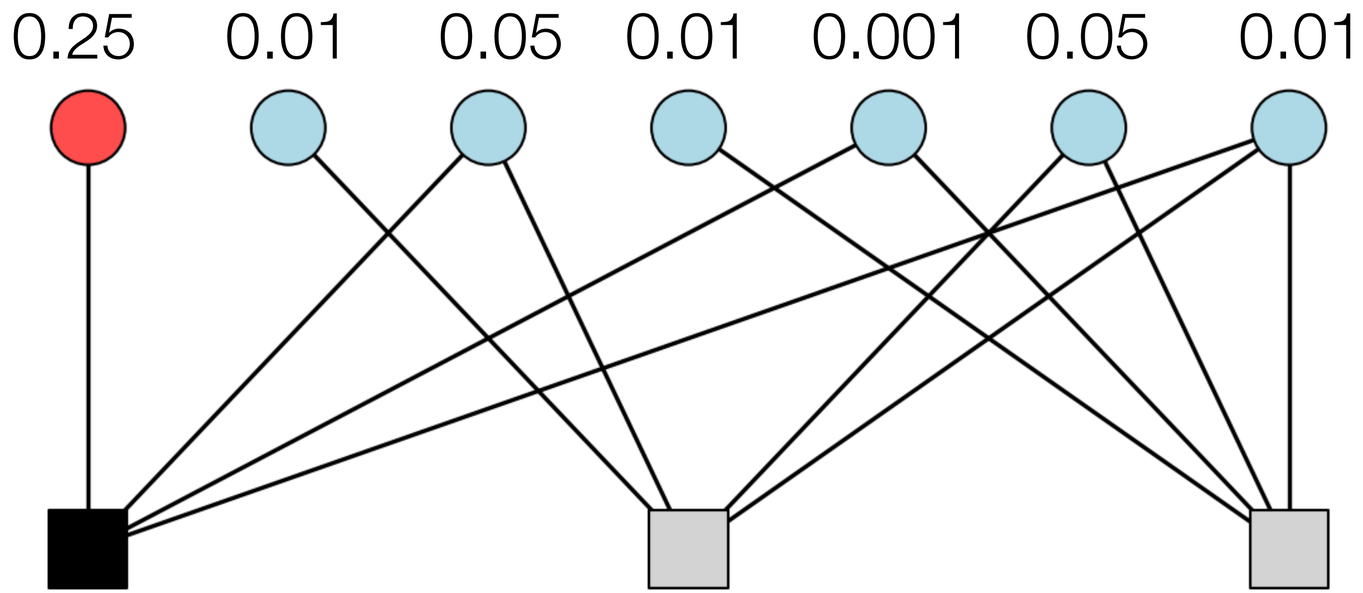
$$\text{Belief}(\text{error}) = 0.05 \times 0.87 \times 0.13 = 0.006$$

$$\text{Belief}(\text{no error}) = (1 - 0.05) \times (1 - 0.87) \times (1 - 0.13) = 0.112$$

$$\text{Prob}(\text{error}) = \frac{0.006}{0.112 + 0.006} = 0.05$$

# Belief Propagation

Data nodes use the received information to estimate their **new error probability.**

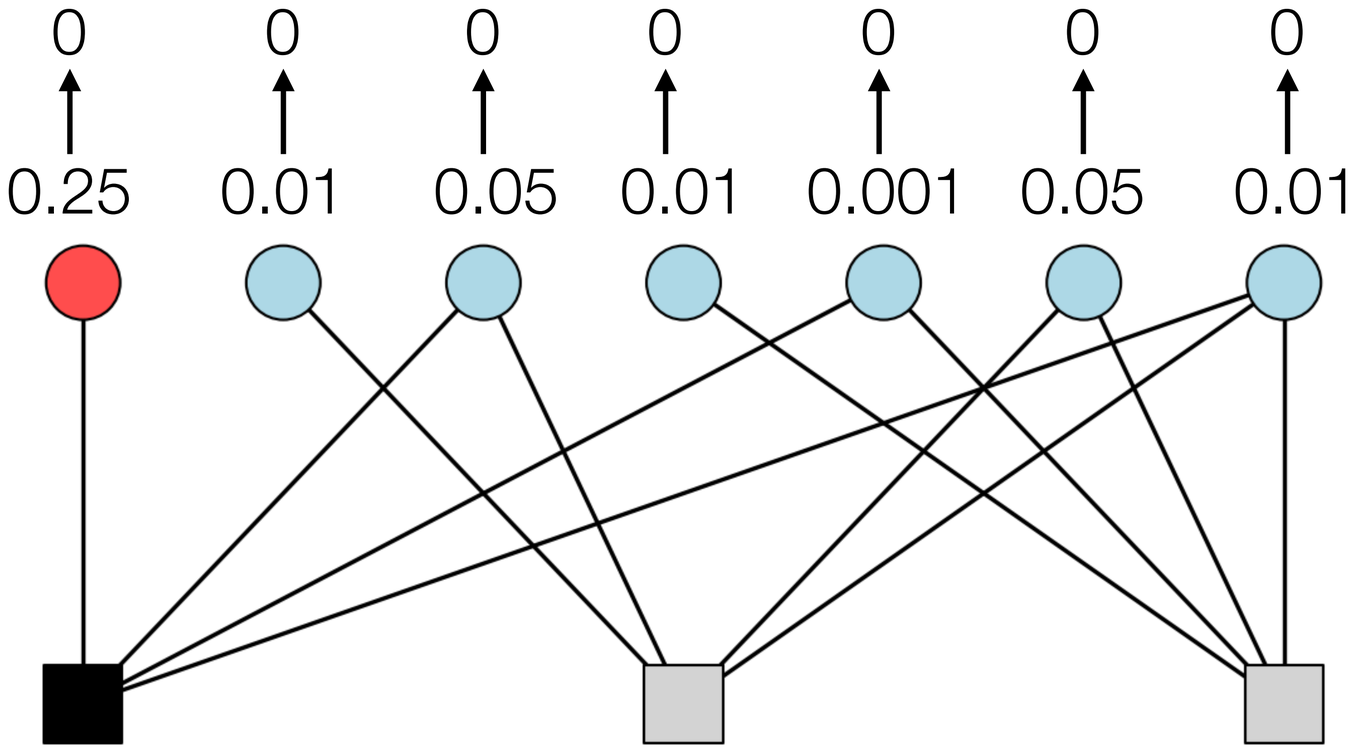


# Belief Propagation

Data nodes convert probabilities to a **hard decision**.

$$\text{Prob}(\text{error}) \leq 0.5 \rightarrow 0$$

$$\text{Prob}(\text{error}) > 0.5 \rightarrow 1$$



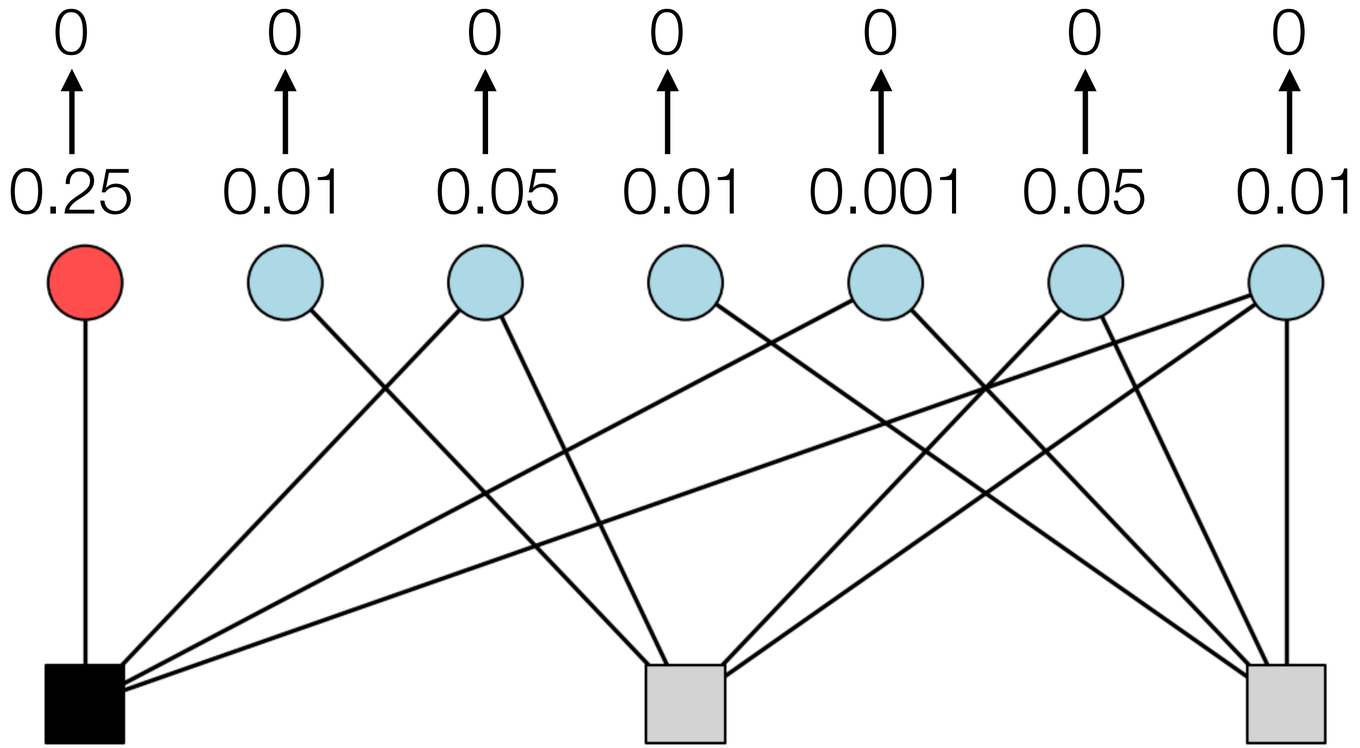
# Belief Propagation

Is hard decision compatible with **syndrome**?

$$\sigma = He^{\mathsf{T}}$$

If **yes**, output the error.

If **no**, **repeat** using the new priors.

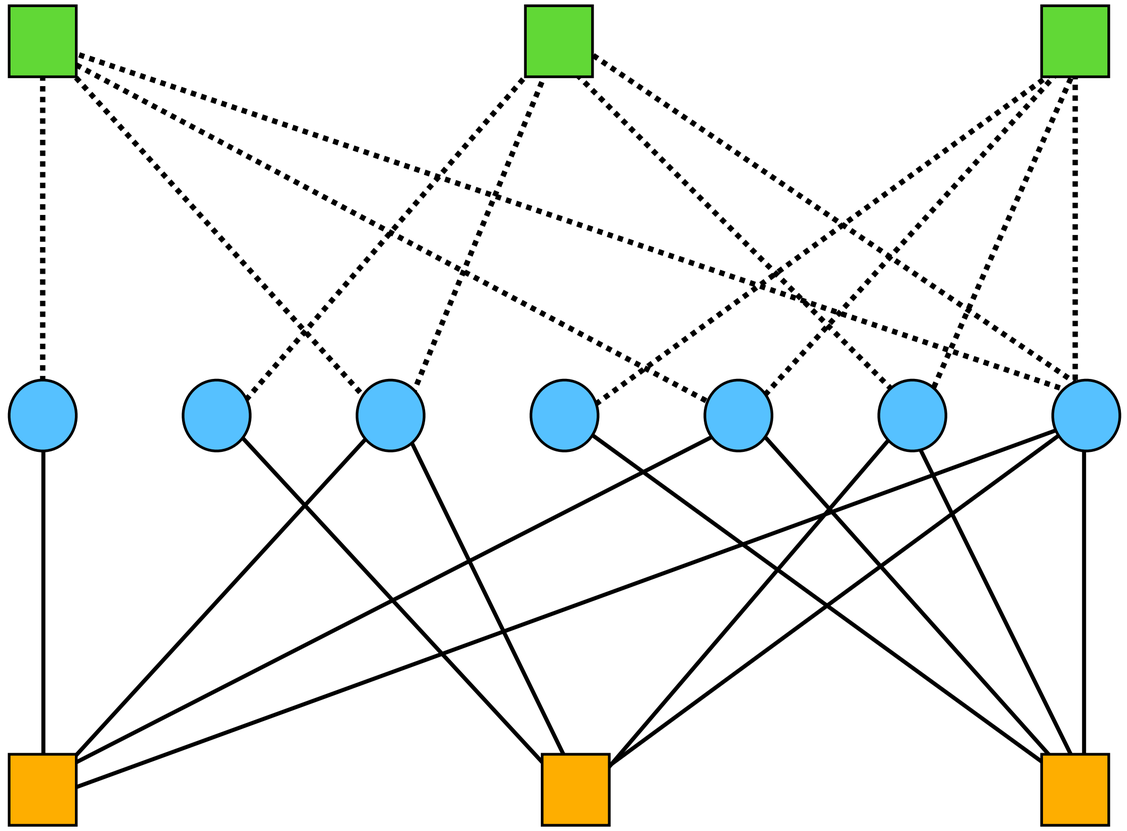


# Belief Propagation on quantum codes

- Quantum CSS code can be treated as two classical codes.

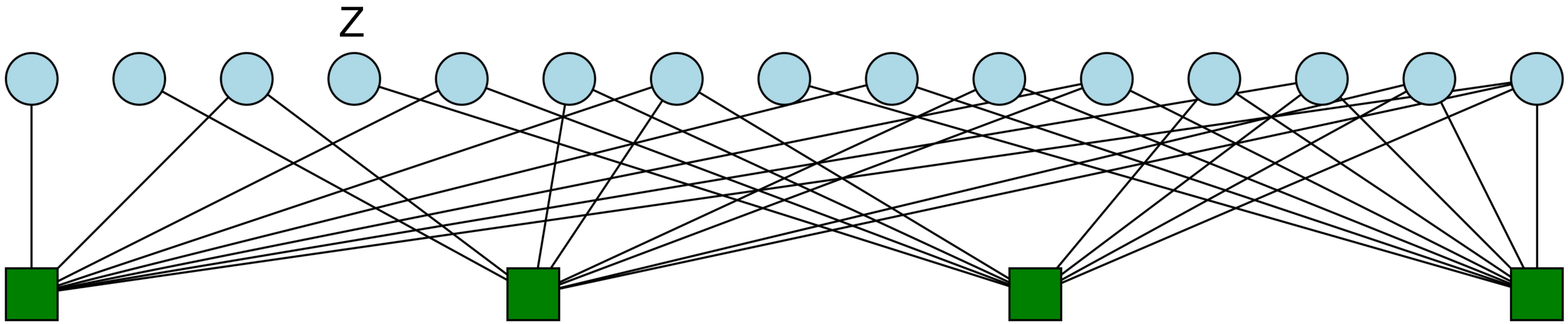
$$H = \begin{bmatrix} H_X & 0 \\ 0 & H_Z \end{bmatrix}$$

- Very similar to a classical decoding problem in structure.
- BP decoders can be used.
- But **degeneracy** and **small cycles** can lead to convergence issues and bad outputs.



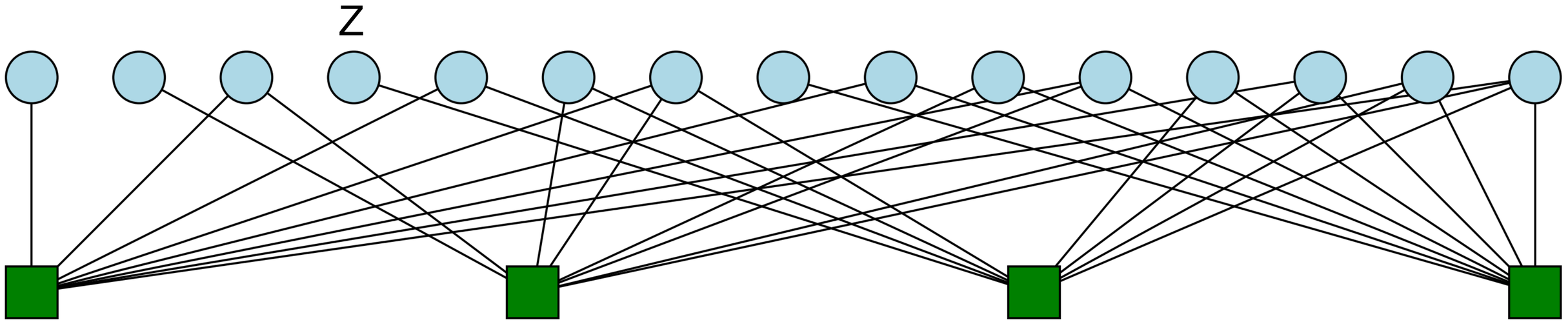
# Belief Propagation on quantum codes

- For example, 15 qubit Reed Muller code:
- Error:



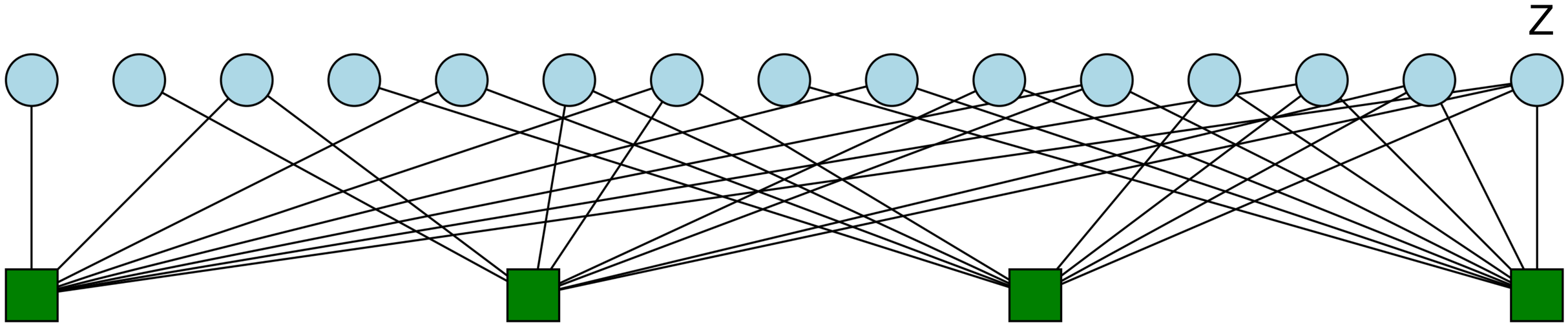
# Belief Propagation on quantum codes

- For example, 15 qubit Reed Muller code:
- BP output:



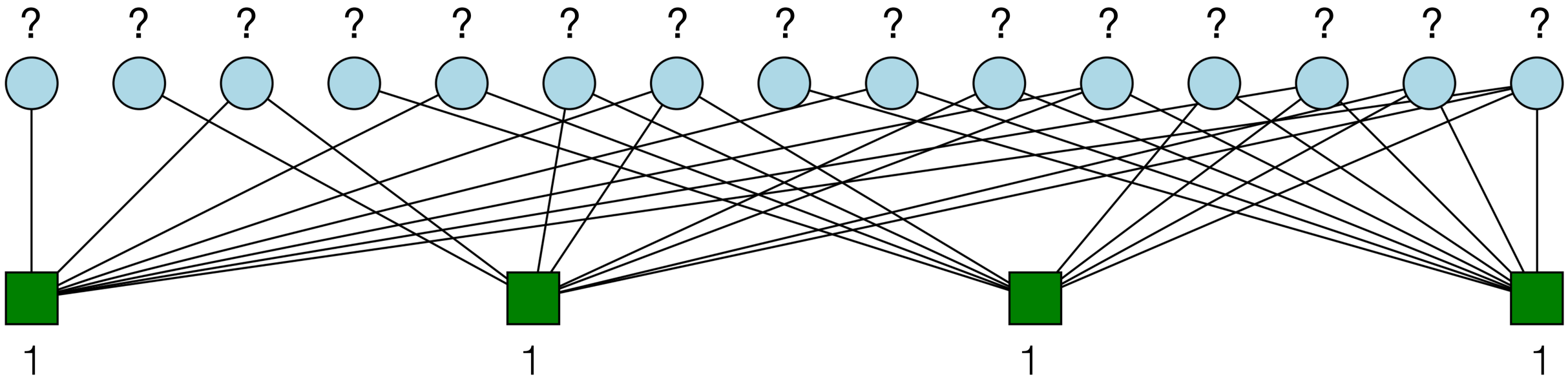
# Belief Propagation on quantum codes

- For example, 15 qubit Reed Muller code:
- Error:



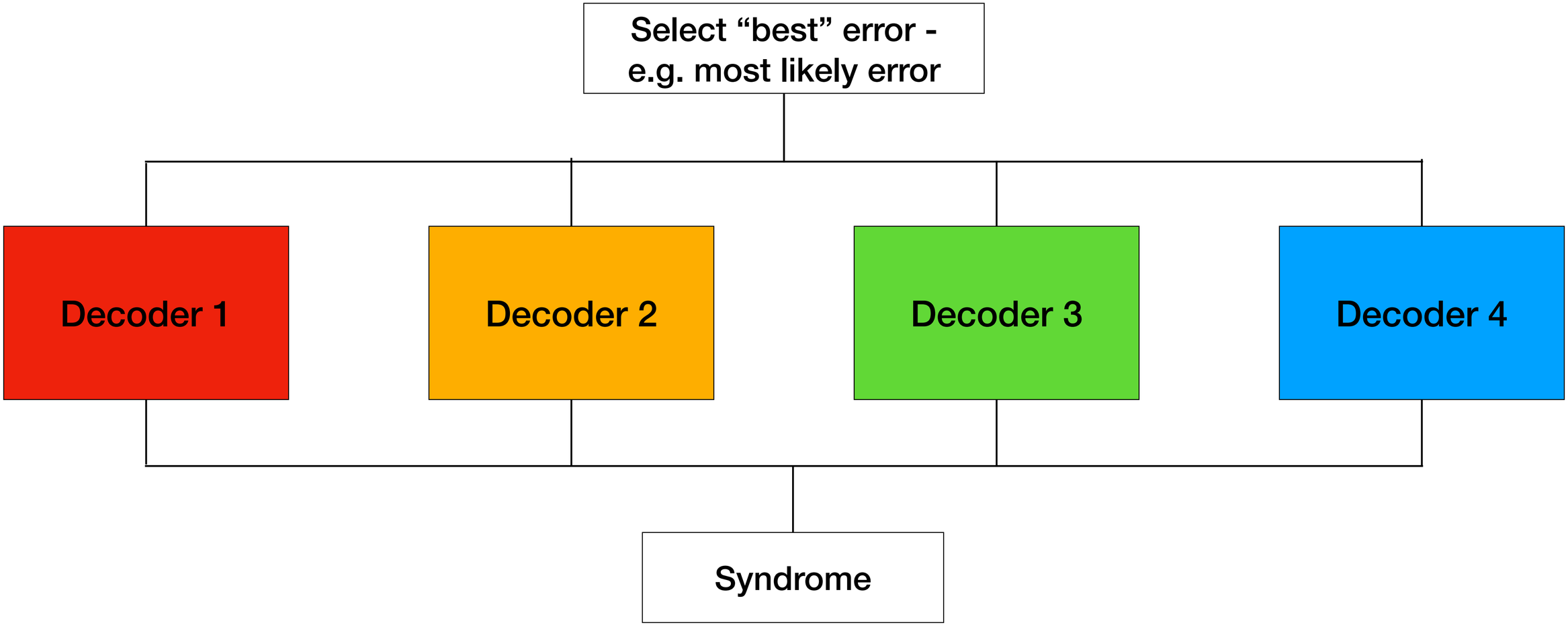
# Belief Propagation on quantum codes

- For example, 15 qubit Reed Muller code:
- BP Output:




BP “sees errors everywhere” and **fails to converge**.

# Ensemble Decoders




“Harmonisation” - N. Shutty, et al,  arXiv:2401.12434

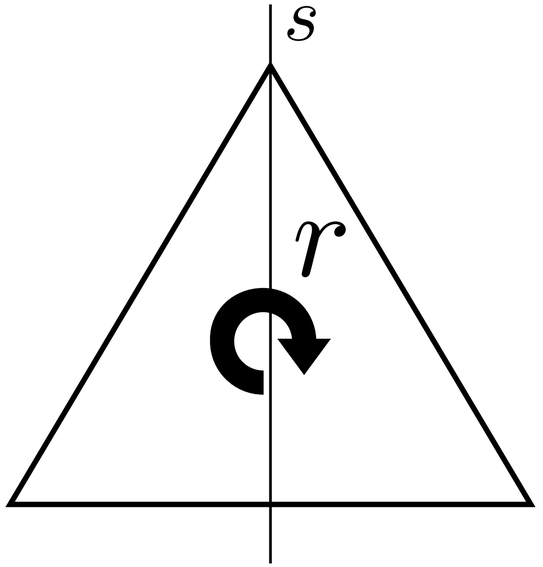
Automorphism Ensemble Decoding - Geiselhart, et al. , arXiv:2012.07635

# Code automorphisms

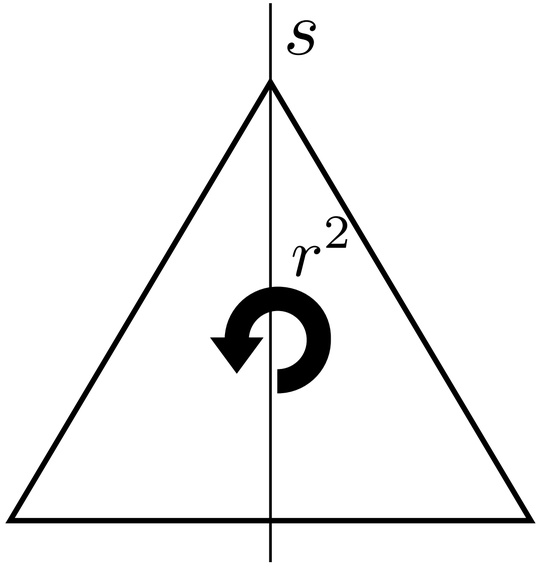
- Origin in group theory:
- An **automorphism** is a permutation of group elements which preserves the group structure

$$g, h \in G$$
$$\pi(g)\pi(h) = \pi(gh)$$

- Automorphisms form their own group, the ***automorphism group***.




$$r \leftrightarrow r^2$$



# Code automorphisms

- A code (permutation) **automorphism** is a **permutation** of bits which maps **codewords to codewords**.

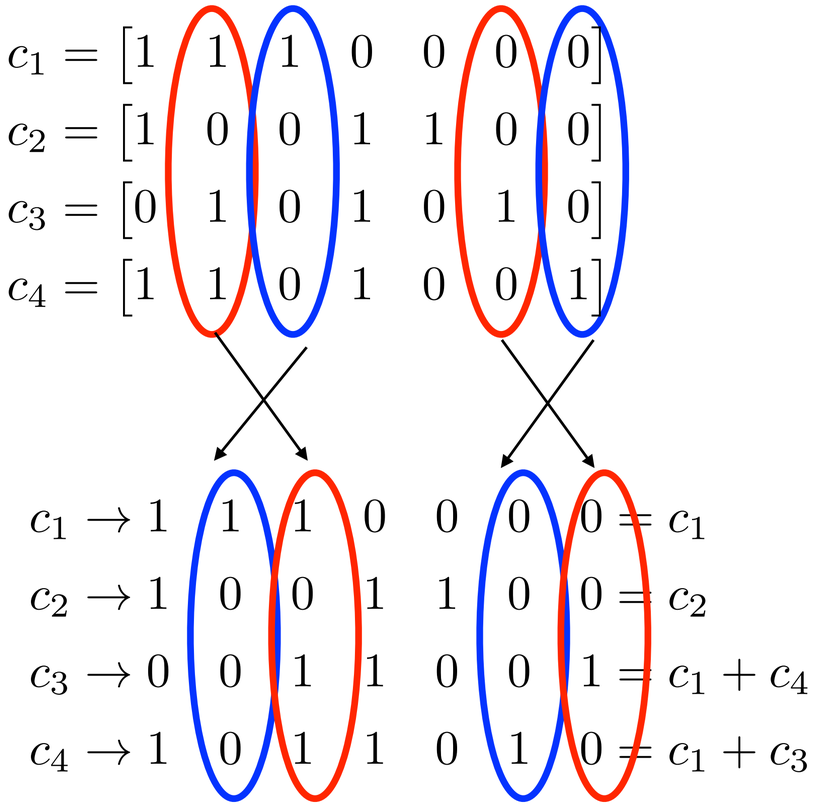
$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- E.g. (7,4) Hamming code
  - The permutation:

$$(2 \leftrightarrow 3)(6 \leftrightarrow 7)$$

    Maps codewords to codewords, leaving the codespace invariant

  - Code automorphisms can be computed using **brute force search** or **Leon's algorithm** (e.g. in MAGMA).



# Code automorphisms

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- After the permutation, the **original parity check matrix** is **still valid**.
- So the permutation changes the relationship between **errors** and **checks**.

- E.g. Error on bit 2

$$e = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$s = He^{\mathsf{T}} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

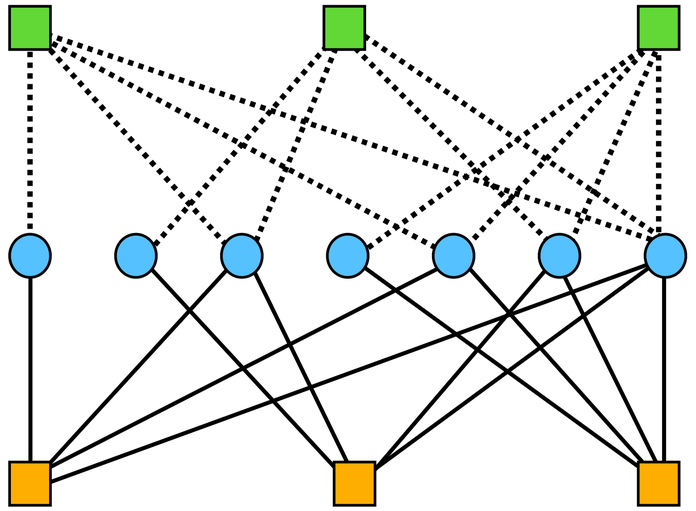
- Applying the permutation (without permuting the parity checks)

$$e' = \pi(e) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$s' = He'^{\mathsf{T}} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

# Quantum code automorphisms

- Code automorphisms can also be defined for **quantum codes**.
- For example, **permutation automorphisms**:
  - Permuting qubits mapping code-space to code-space.
- Automorphisms of quantum codes are a ready source of **fault-tolerant quantum logic gates**.
- Automorphism groups can be computed via a mapping to **classical codes** (e.g. X-check and Z-checks of a CSS code).

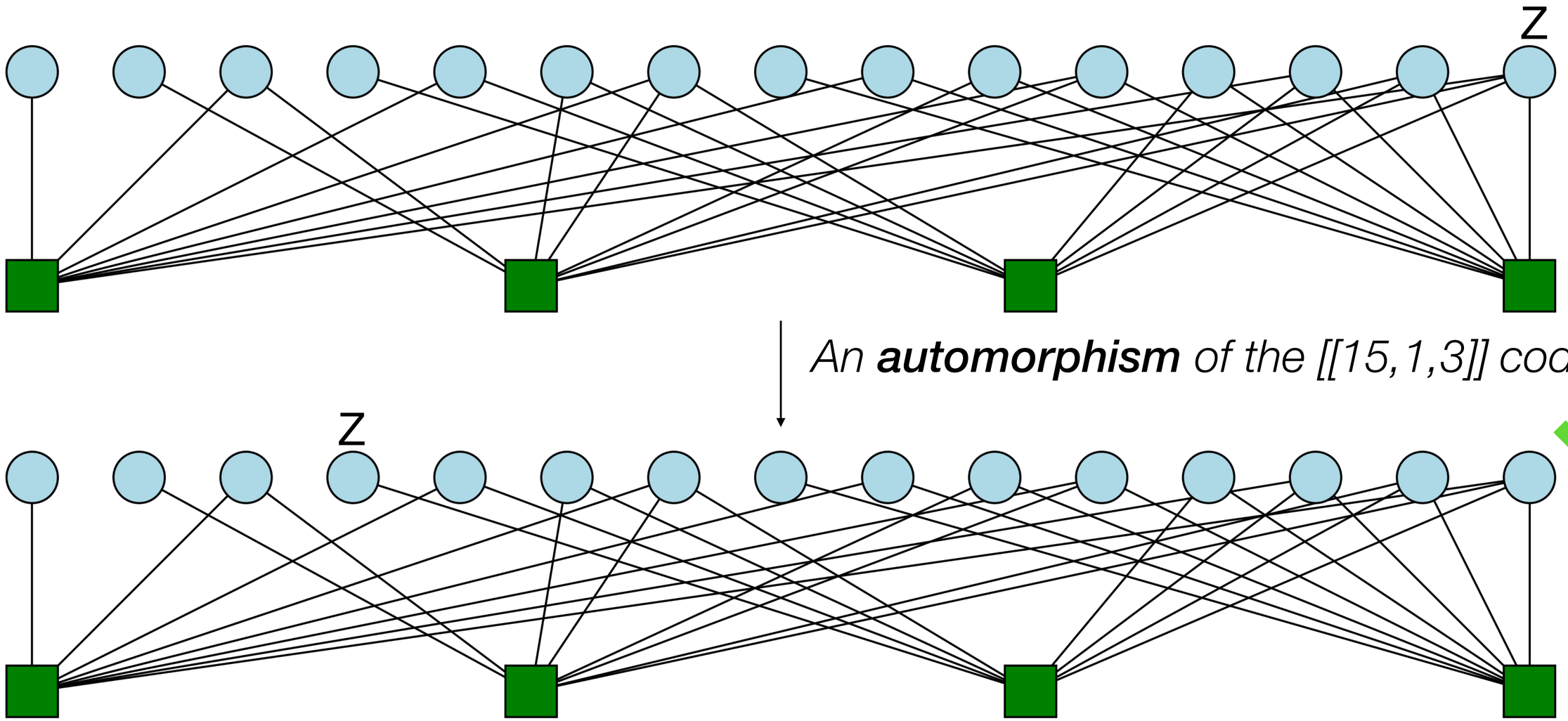


A. Calderbank et al, Proc IEEE (2007)

S. Bravyi et al, Nature 2024

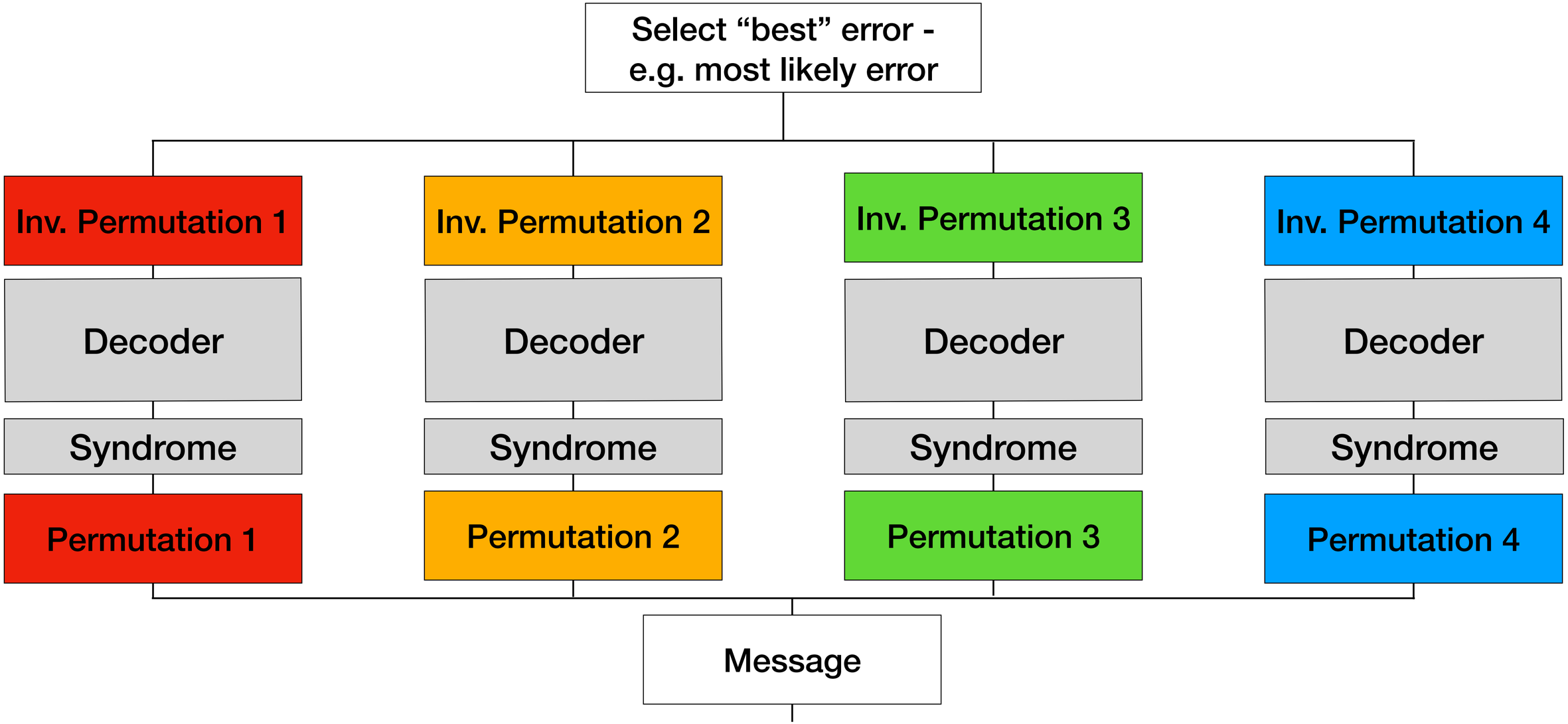
H. Sayginel et al, arXiv:2409.18175

# A quantum automorphism ensemble decoder?




- Automorphisms can **improve BP performance!**

# Classical Automorphism Ensemble Decoders




Automorphism Ensemble Decoding - Geiselhart, et al. , arXiv:2012.07635

# The AutDec Decoder

- Applying **physical permutations** to qubits would **degrade** code performance.
- Fortunately,
  - Permuting **(qu)bits** while keeping the **parity checks** unchanged
- is equivalent to
  - Permuting the **parity checks** while keeping **(qu)bit positions** unchanged.

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\begin{aligned} c_1 &= [1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0] \\ c_2 &= [1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0] \\ c_3 &= [0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0] \\ c_4 &= [1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1] \end{aligned}$$

# The AutDec Decoder

- Since the permutation is an automorphism:
  - the new parity checks must be parity **checks of the code** (e.g. still in the stabiliser group)
- Thus the permutation is equivalent to a **linear transformation** on the **parity checks**.
- This equivalent to a **linear transformation** on the **syndrome**.

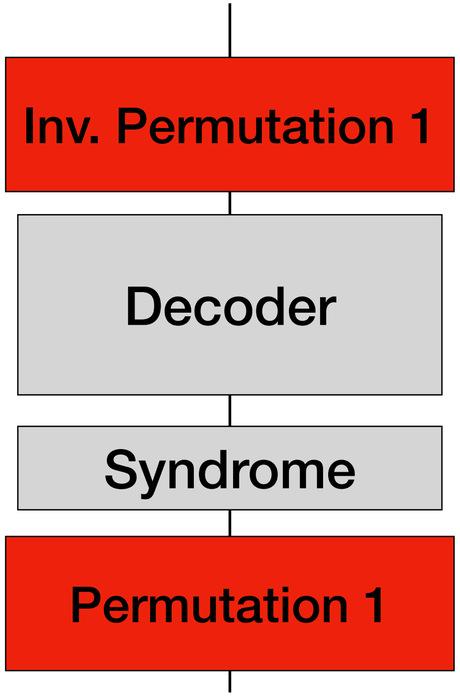
$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\pi(H) = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

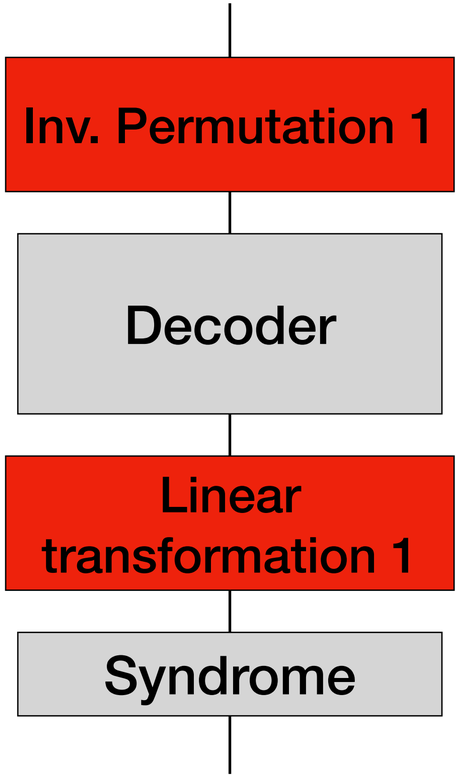
# The AutDec Decoder

- Hence, instead of permuting the qubits:
  - We can apply the automorphism as a **linear transformation** to the **syndrome**.
- Thus syndrome extraction circuit can be used **unchanged**.
- **Different permutations** can be applied to the same **syndrome measurement**.

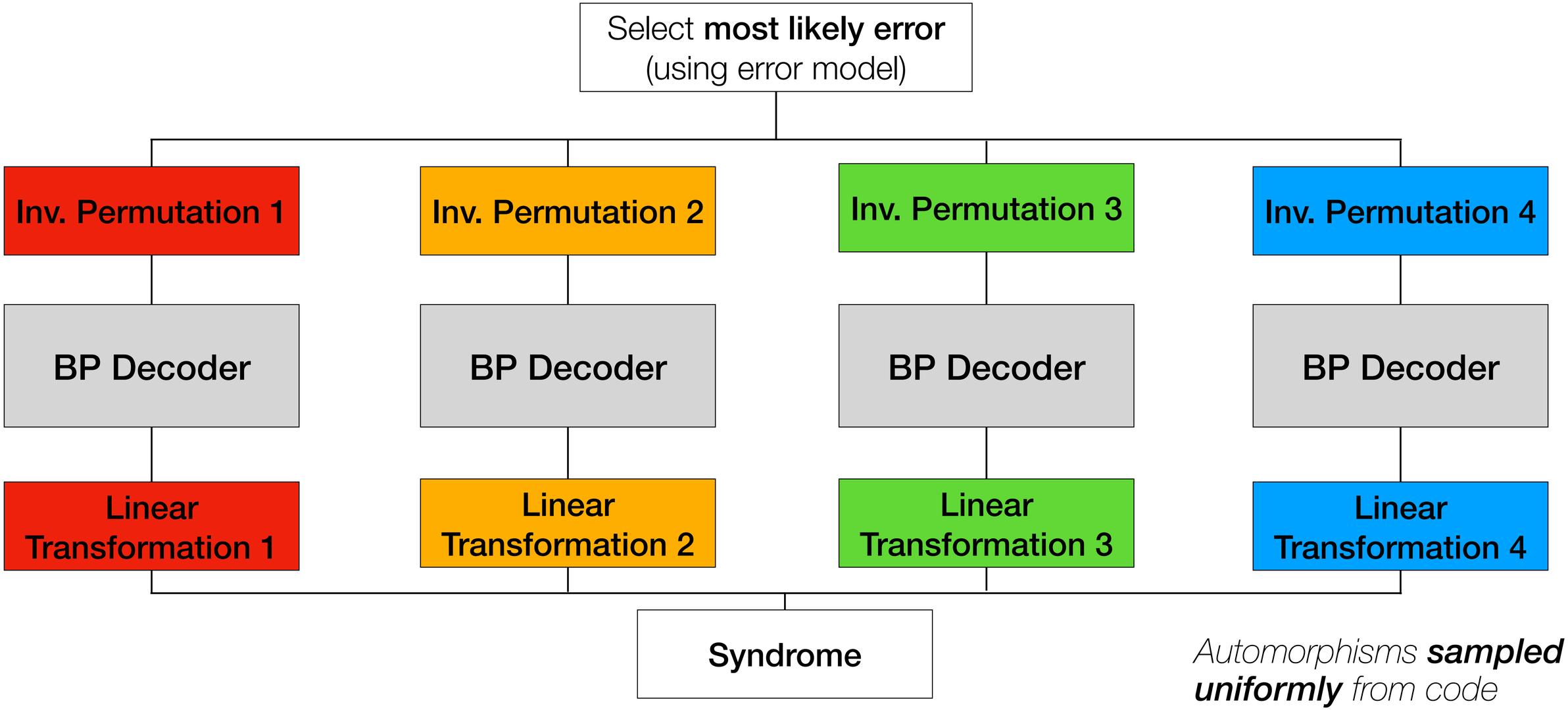


# The AutDec Decoder

- Hence, instead of permuting the qubits:
  - We can apply the automorphism as a **linear transformation** to the **syndrome**.
- Thus syndrome extraction circuit can be used **unchanged**.
- **Different permutations** can be applied to the same **syndrome measurement**.



# The AutDec Decoder

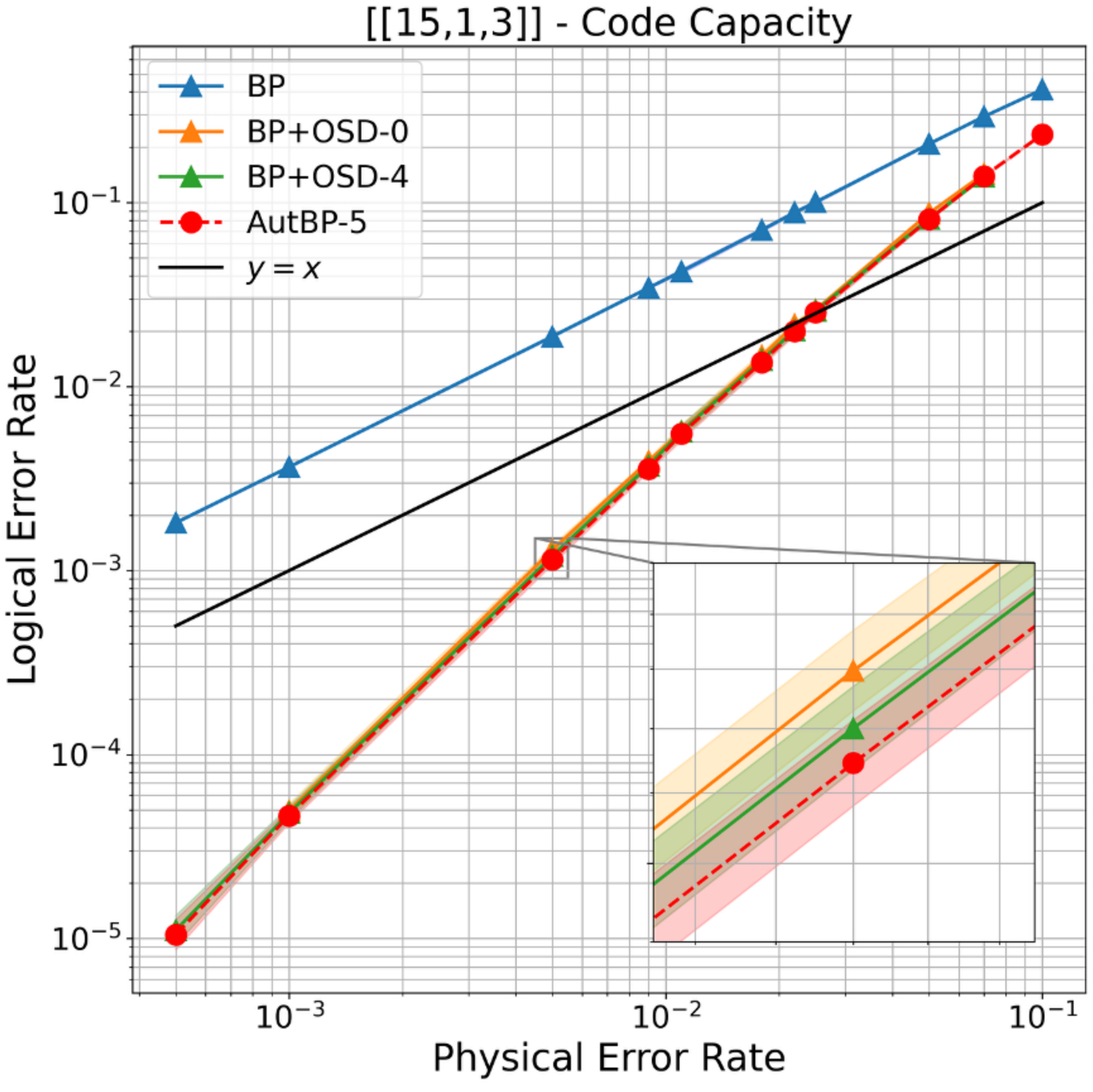



*Automorphisms **sampled uniformly** from code automorphism group*

Koutsioumpas et al, quant-ph/2503.01738

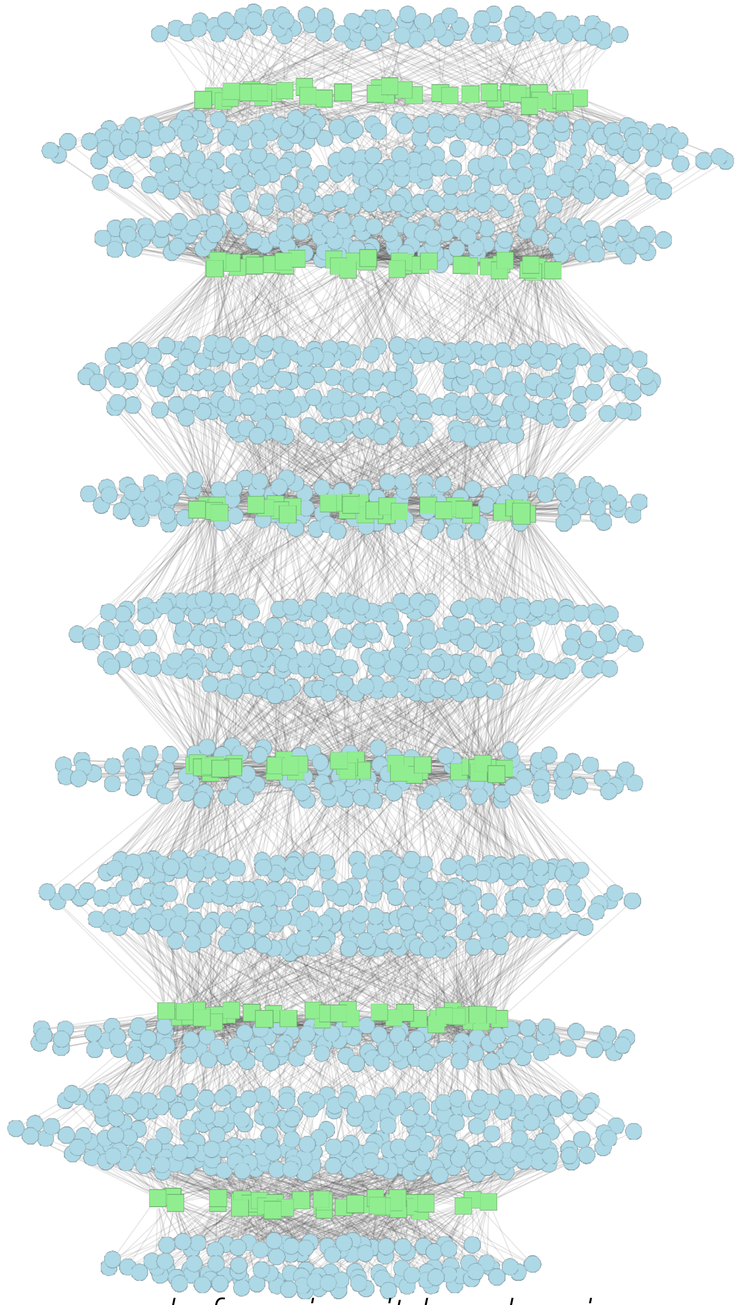
# Code capacity decoder

- 15-qubit **Reed Muller** code
  - Automorphism group computed analytically, order **20160**.
- **Code capacity** noise model:
  - Depolarising noise on data qubits
  - No measurement errors
- AutDec coder with **5 parallel BP instances** (15 iterations)



# Larger codes and Detector Error Models

- **Circuit level noise** on a large LDPC code is usually represented as a **Detector Error Model**.
- Decoding problem essentially a large classical code with a **very large parity check matrix**.
- Automorphism group size and calculation time, blow up with size of the code. The group becomes **too expensive** to compute for even a relatively modest code (e.g. 72 qubits).
- However, **graph automorphisms** of the Tanner graph form a much smaller sub-group of the automorphism group.

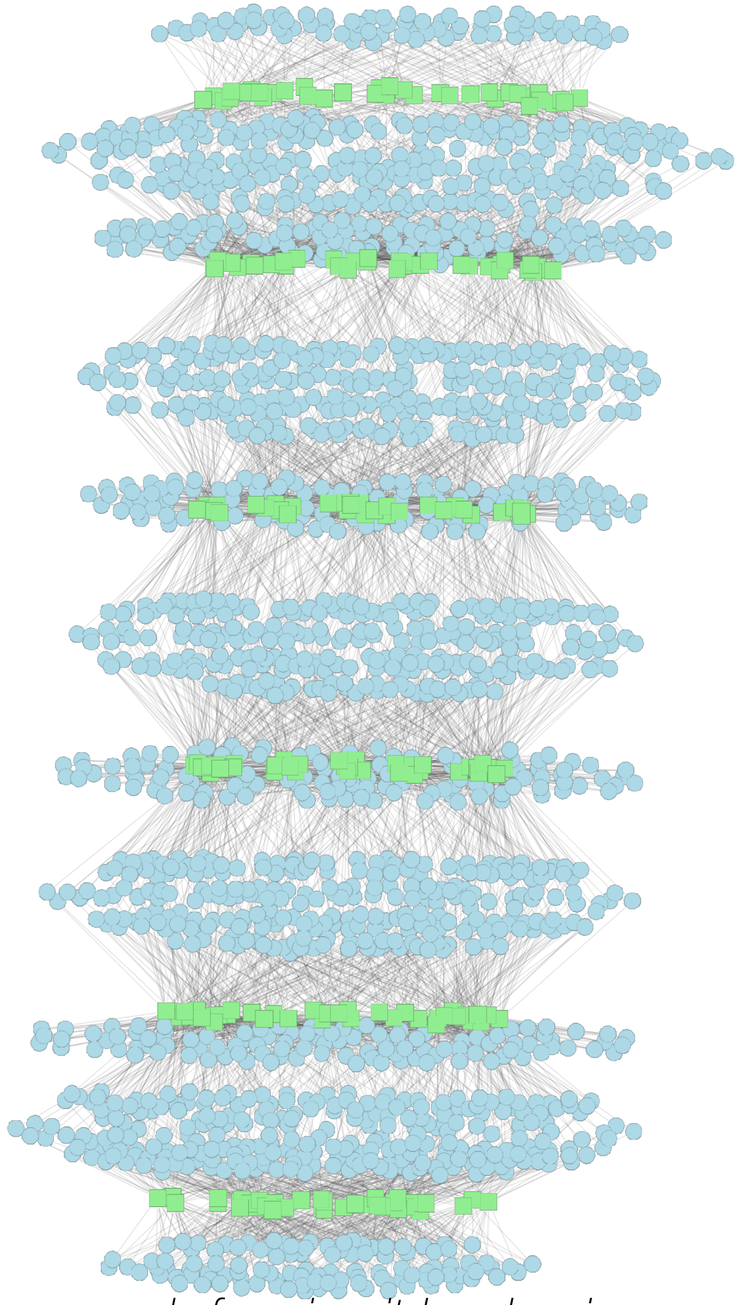


*Tanner graph for circuit level noise on [[72,12,6]] BB code*

# Larger codes and Detector Error Models

- **Graph automorphism** algorithms are fast and open-sourced.

| Code | DEM pcm shape | Tanner graph automorphism group order | time (sec) |
|---|---|---|---|
| $[[72, 12, 6]]$ | $(252, 2232)$ | 36 | 0.002 |
| $[[90, 8, 10]]$ | $(495, 4590)$ | 45 | 0.004 |
| $[[108, 8, 10]]$ | $(594, 5508)$ | 54 | 0.005 |
| $[[144, 12, 12]]$ | $(936, 8784)$ | 72 | 0.009 |
| $[[288, 12, 18]]$ | $(2736, 26208)$ | 144 | 0.024 |
| $[[360, 12, <= 24]]$ | $(4500, 43560)$ | 180 | 0.048 |
| $[[756, 16, <= 34]]$ | $(13230, 129276)$ | 378 | 0.139 |



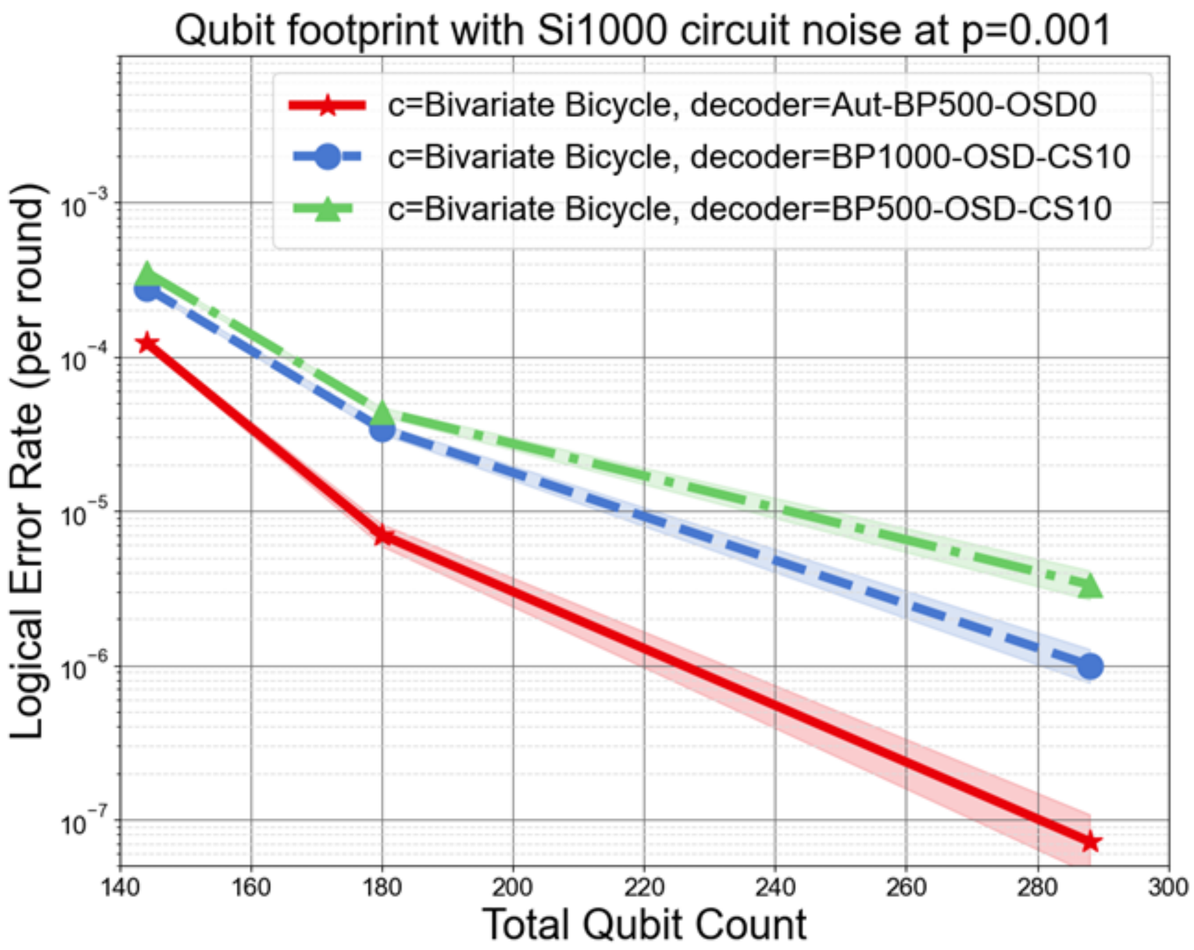
*Tanner graph for circuit level noise on [[72,12,6]] BB code*

# Larger codes and DEMS


***Bivariate bicycle codes*** *with* ***circuit level noise***

# Combined with OSD

- To further improve accuracy, AutDec can be used **together with OSD**.
- It can then significantly **outperform** BP-OSD.
- Figure shows the results for large **bivariate bicycle codes** under **circuit level noise**.

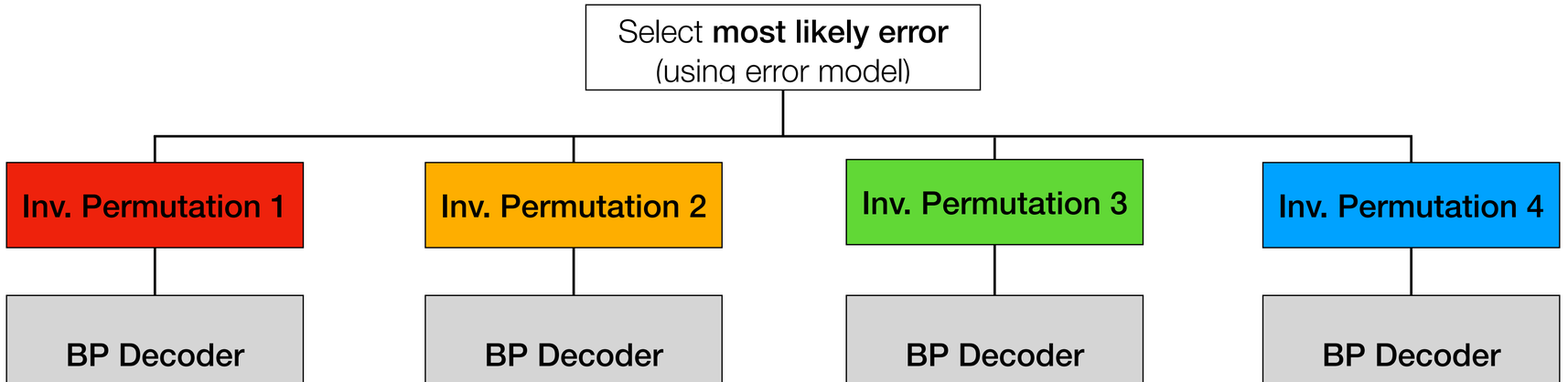



*GPU implementation in collaboration with Roffe group at Edinburgh, Supermicro, INVIDIA*
*https://qec.codes/blog/autdec.htm*

# Summary and Outlook

- **AutDec** parallelises **equivalent but different** versions of the decoding problem to overcome BP “dead-ends”.
- It works well for codes with rich automorphism groups, in particular **Reed Muller** and **Bivariate Bicycle** codes.
- In many cases it matches BP+OSD error rates with a **significantly faster runtime** (linear time BP).
- **Graph automorphisms** let us target **large codes** and **complicated DEMs**.

- Numerical evidence suggests that ensemble needs to grow in the size of the code. Can we find **analytic bounds** for the **parallel resource scaling**?
- Can we derandomise? Can insights from AutDec help us design **better deterministic decoders** for specific codes?
- What **other ways** can we help BP avoid getting stuck decoding quantum codes?
- Can **ensembling** be used to improve other decoder approaches? (Eg. “Vibe decoding” arXiv:2508.15743, Relay-BP arXiv:2506.01779)



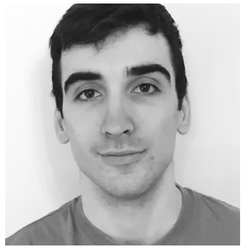
Engineering and Physical Sciences Research Council

UCL

# Thanks for listening

**Joint work with:**

Stegios Koutsioumpas, Hasan Sayginel and Mark Webster
(Thanks also to Joschka Roffe)



**arXiv:2503.01738**
Automorphism Ensemble Decoding for Quantum LDPC Codes

GitHub: https://github.com/hsayginel/autdec

pip install autdec