Introduction to Low-density Parity-check Code (LDPC)

NGUYEN Trong Cuong

Computer Communication Lab

21st Dec., 2022

Outline

I. Error Correction Code

II. Low-density Parity-check Codes
   1. Introduction
   2. Hard-Decision Bit Flipping
   3. Messaging-Passing Algorithm

III. Future Direction

Outline

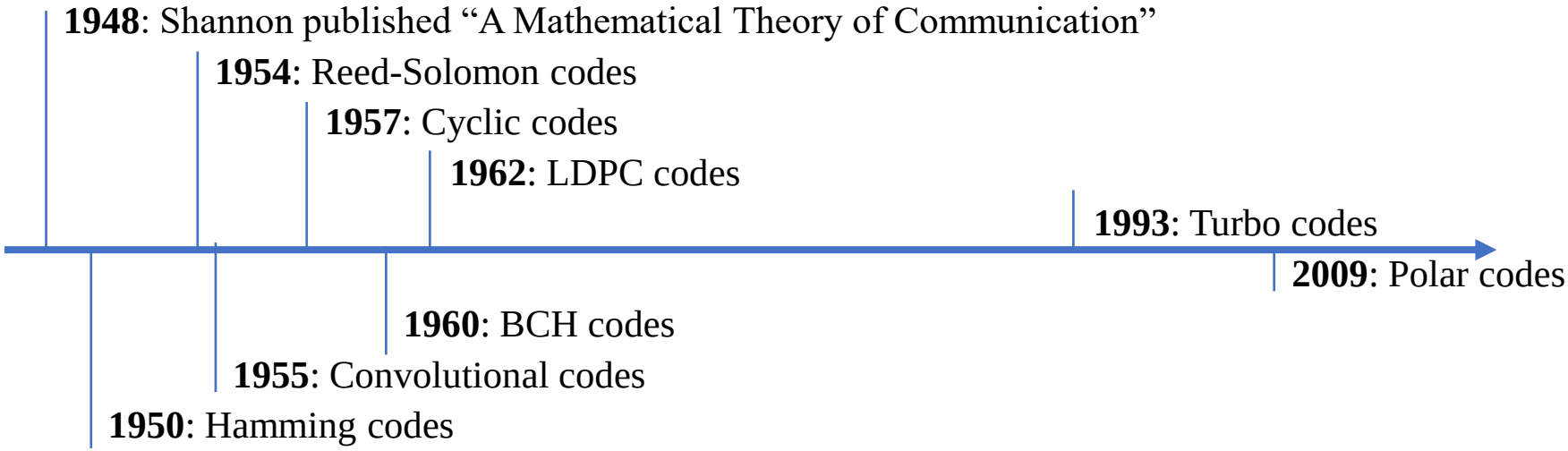
I. Error Correction Code

II. Low-density Parity-check Codes
   1. Introduction
   2. Hard-Decision Bit Flipping
   3. Messaging-Passing Algorithm

III. Future Direction

Error Correction Codes

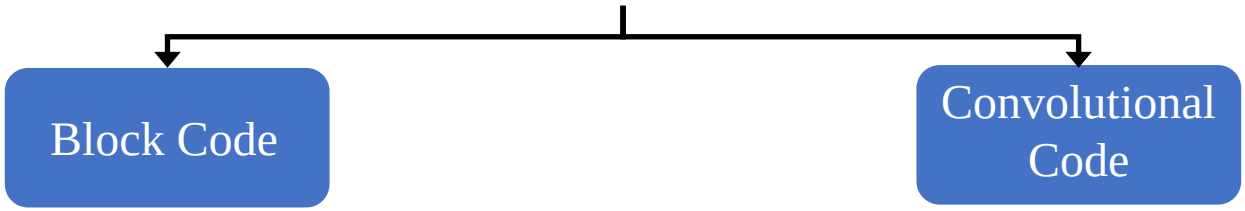
- **Error Correction Codes (ECC)** are error-control methods that add redundancy to the original message so that a certain number of errors can be corrected.
- **History of ECCs**



ECCs in the Past	ECCs in Recent Standards	Future Requirements of ECCs
• Hardware limitations • Hard-decision decoder • Convolutional codes are most efficient but have high decoding complexity. • Other common codes are Reed-Solomon, BCH,..	• **Turbo codes**: 3G UMTS, 4G LTE • **LDPC**: WiFi, WiMax, 10G Base-T Ethernet, 5G New Radio (data channel) • **Polar codes**: 5G New Radio (control channel)	To enhance user experience and support diverse applications, ECCs needs • Higher error correction capability • Lower computational complexity • Wide range support of different code rate with compatibility.

Classifications of ECC

- **Error Correction Codes (ECC)** can be classified into **block code** and **convolutional code.**



- Work on fixed-size blocks of bits or symbols

(+) Best for burst sources

(+) Decoding complexity is simpler than the convolutional code

- E.g., Reed–Solomon code, which is used in compact disc, DVD,…

- Work on bit streams of arbitrary length

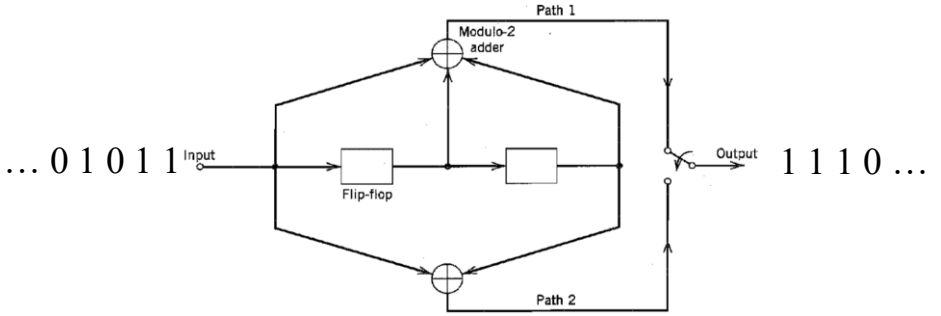
(+) Best for very large data streams

(-) Decoding complexity increases exponentially in the code length

- Digital video, radio, and satellite communication

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

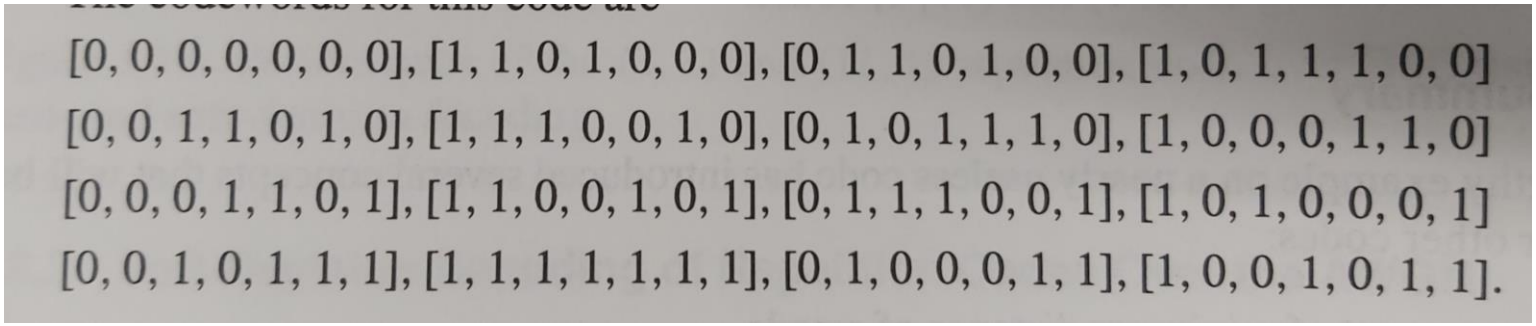
Message Modulo 2 Generator matrix Codeword



Linear Block Code

- A (N, K) **binary block code** $\mathcal{C}$ is a set of 2^K vectors of length N.
  - Each vector is called a **codeword.**
  - Each codewords has N bits.

Example: All the codewords of the Hamming block code (7,4)
=> There are $2^4 = 16$ codewords. Each of them has 7 bits.


[0, 0, 0, 0, 0, 0, 0], [1, 1, 0, 1, 0, 0, 0], [0, 1, 1, 0, 1, 0, 0], [1, 0, 1, 1, 1, 0, 0]
[0, 0, 1, 1, 0, 1, 0], [1, 1, 1, 0, 0, 1, 0], [0, 1, 0, 1, 1, 1, 0], [1, 0, 0, 0, 1, 1, 0]
[0, 0, 0, 1, 1, 0, 1], [1, 1, 0, 0, 1, 0, 1], [0, 1, 1, 1, 0, 0, 1], [1, 0, 1, 0, 0, 0, 1]
[0, 0, 1, 0, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1], [0, 1, 0, 0, 0, 1, 1], [1, 0, 0, 1, 0, 1, 1].

- An (N, K) block code $\mathcal{C}$ is **linear** if and only if <u>any linear combination of codewords is also a codeword.</u>

- A linear code can be described by a **generator matrix** G or a **parity check matrix** A.

Generator Matrix

❑ **Generator matrix** G of a (N, K) block code $\mathcal{C}$ is used to encode a K-bits message to a N-bits codeword $\mathbf{c} \in \mathcal{C}$.

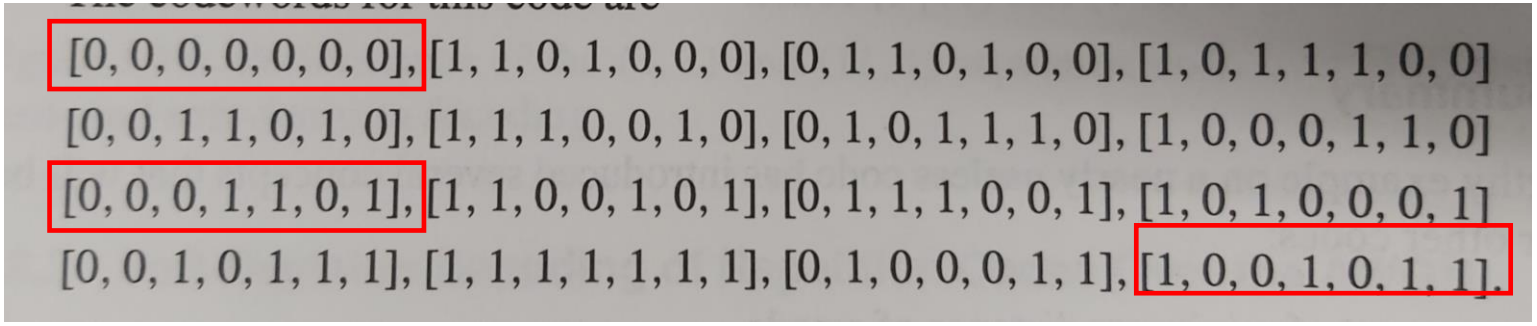
$$\mathbf{m}G = \mathbf{c}$$

where $\mathbf{m}$ is the $1 \times K$ message vector, G is a $K \times N$ matrix.

Example: The generator matrix G for block code $\mathcal{C}$ (7,4)
Every possible message $\mathbf{m}$ can be mapped into a codeword in $\mathcal{C}$

If $\mathbf{m} = [0,0,0,0]$ => $\mathbf{c} = [0,0,0,0,0,0,0]$
If $\mathbf{m} = [0,0,0,1]$ => $\mathbf{c} = [0,0,0,1,1,0,1]$
…
If $\mathbf{m} = [1,1,1,1]$ => $\mathbf{c} = [1,0,0,1,0,1,1]$

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


[0, 0, 0, 0, 0, 0, 0], [1, 1, 0, 1, 0, 0, 0], [0, 1, 1, 0, 1, 0, 0], [1, 0, 1, 1, 1, 0, 0]
[0, 0, 1, 1, 0, 1, 0], [1, 1, 1, 0, 0, 1, 0], [0, 1, 0, 1, 1, 1, 0], [1, 0, 0, 0, 1, 1, 0]
[0, 0, 0, 1, 1, 0, 1], [1, 1, 0, 0, 1, 0, 1], [0, 1, 1, 1, 0, 0, 1], [1, 0, 1, 0, 0, 0, 1]
[0, 0, 1, 0, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1], [0, 1, 0, 0, 0, 1, 1], [1, 0, 0, 1, 0, 1, 1].

Parity Check Matrix

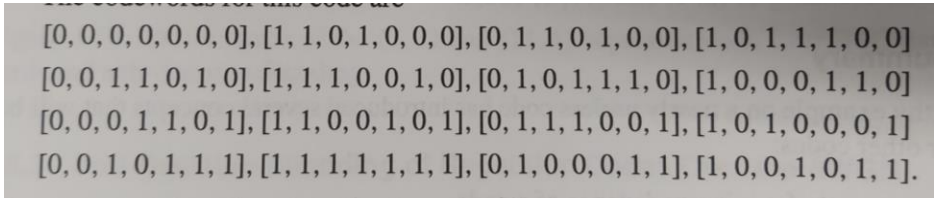
- For every codeword $\mathbf{c} \in \mathcal{C}$, the $(N-K) \times N$ matrix A is defined as the **parity check matrix** for $\mathcal{C}$ only if

$$A\mathbf{c}^T = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \cdots \\ \mathbf{a}_M \end{bmatrix} \mathbf{c}^T = \mathbf{0}^T$$

where $M = N - K$, $z_m = \mathbf{a}_m \mathbf{c}^T = 0$ is **a parity check** $(0 < m \leq M)$**.**

- **Example:** The parity check matrix A for block code $\mathcal{C}$ (7,4)

For every codeword $\mathbf{c} \in \mathcal{C}$, $A\mathbf{c}^T = \mathbf{0}$**.**


[0, 0, 0, 0, 0, 0, 0], [1, 1, 0, 1, 0, 0, 0], [0, 1, 1, 0, 1, 0, 0], [1, 0, 1, 1, 1, 0, 0]
[0, 0, 1, 1, 0, 1, 0], [1, 1, 1, 0, 0, 1, 0], [0, 1, 0, 1, 1, 1, 0], [1, 0, 0, 0, 1, 1, 0]
[0, 0, 0, 1, 1, 0, 1], [1, 1, 0, 0, 1, 0, 1], [0, 1, 1, 1, 0, 0, 1], [1, 0, 1, 0, 0, 0, 1]
[0, 0, 1, 0, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1], [0, 1, 0, 0, 0, 1, 1], [1, 0, 0, 1, 0, 1, 1].

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

Outline

I. Error Correction Code

II. Low-density Parity-check Codes
   1. Introduction
   2. Hard-Decision Bit Flipping
   3. Messaging-Passing Algorithm

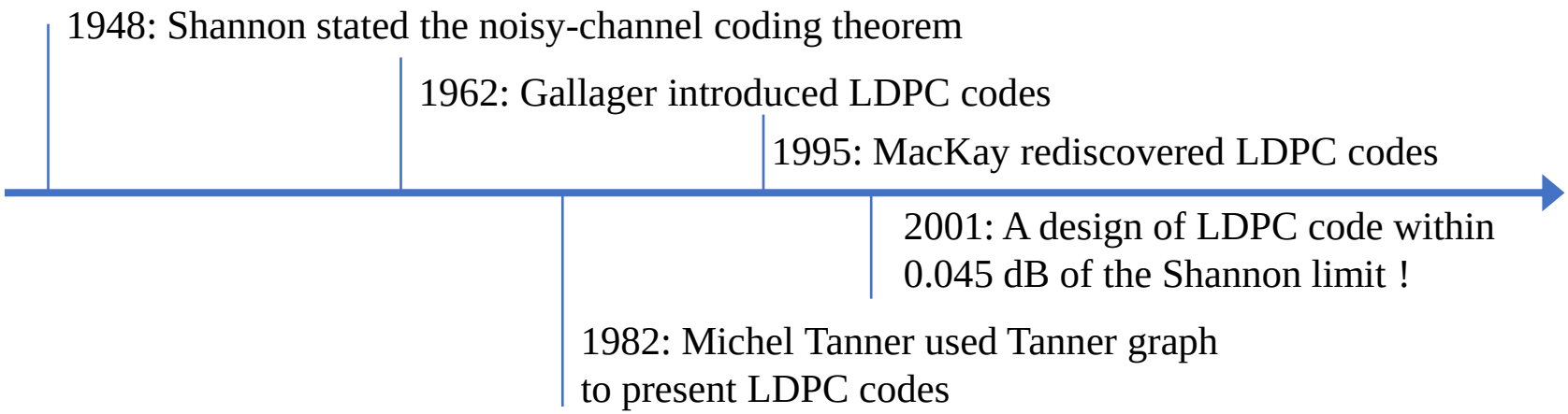
III. Future Direction

Low-density Parity-check Codes

❑ A **low-density parity check (LDPC)** code is a linear block code that *has a very sparse parity check matrix.*

- A matrix is said to be **sparse** if *more than half of elements are zero.*

❑History of LDPC codes



❑Applications

- Digital Video Broadcasting - Satellite - Second Generation (DVB-S2)
- 10G Base-T Ethernet
- 802.11ax (Wifi 6)
- 5G New Radio (data channel)

Classification of LDPC

- **A regular LDPC code** has a parity check matrix, whose every columns has the same number of ones and every row has the same number of ones.

Example:

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

The parity check matrix *A* has 4 ones each row and 2 ones each column => **regular**

- An **irregular** LDPC code is one whose number of ones in each row/column of the parity check matrix is not constant.

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

The parity check matrix *A* has different numbers of ones in each row => **irregular**

Regular LDPC

- **The column weight** w_c is the *number of 1's in each column.*
  - w_c represents the number of checks that one bits takes.
- **The row weight** w_r is the *number of 1's in each row.*
  - w_r represents the number of bits in one check.
- . The **number of ones** in $M \times N$ parity check matrix A is $Mw_r = Nw_c$

  => The code rate $R = 1 - \frac{w_c}{w_r}$

Example: A 5×10 parity check matrix A has $w_r = 4, w_c = 2$

$w_r = 4$ => Every checks involves 4 bits.
$w_c = 2$ => Every bits participates in 2 checks

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

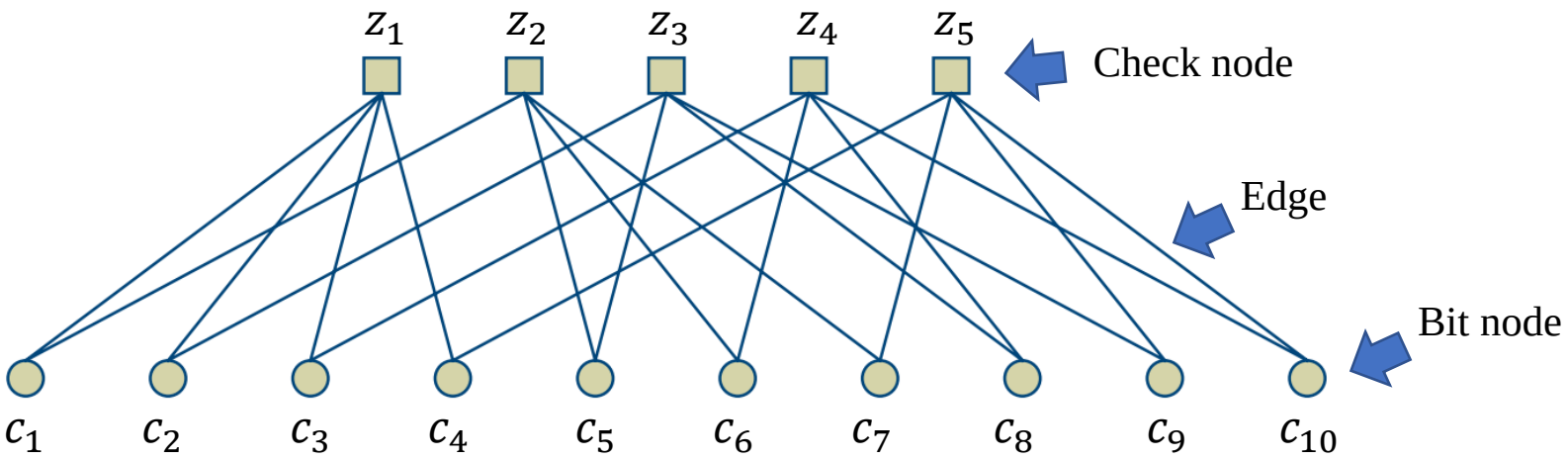
Check z_1 involves bit c_4, c_7, c_8, c_{10}

Bit c_3 participates in check z_2, z_5

Representation of LDPC codes (1)

- The parity check matrix can be presented as a **bipartite graph**.
  - **A bipartite graph** is a graph in which the *nodes can be classified into two classes*, and no edge connects two nodes from the same class.
  - **A Tanner graph** is *a bipartite graph* which represents the parity check matrix of an error correcting code.

- *The nodes in the Tanner graph* are partitioned into two groups, i.e., **bit nodes** and **check nodes**.
  - Each **bit node** presents *a bit in the codeword*.
  - Each **check node** presents *a check*.
  - Each **edge** presents a *1's in the parity check matrix.*

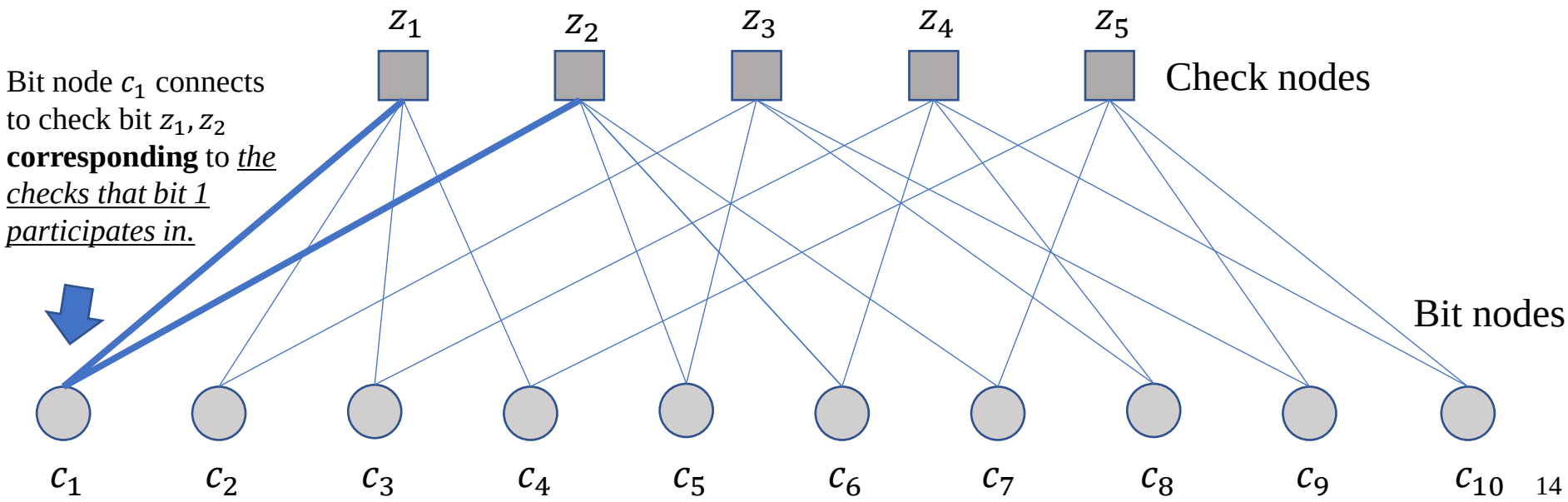


Representation of LDPC codes (2)

Example: A 5×10 parity check matrix A has $w_r = 4, w_c = 2$

- M = number of checks = 5 => **5 check nodes**
- N = number of bits = 10 => *10 bit nodes*
- Each **check node** connect to $w_r = 4$ *bit nodes.*
- Each *bit node* connects to $w_c = 2$ **check nodes.**

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

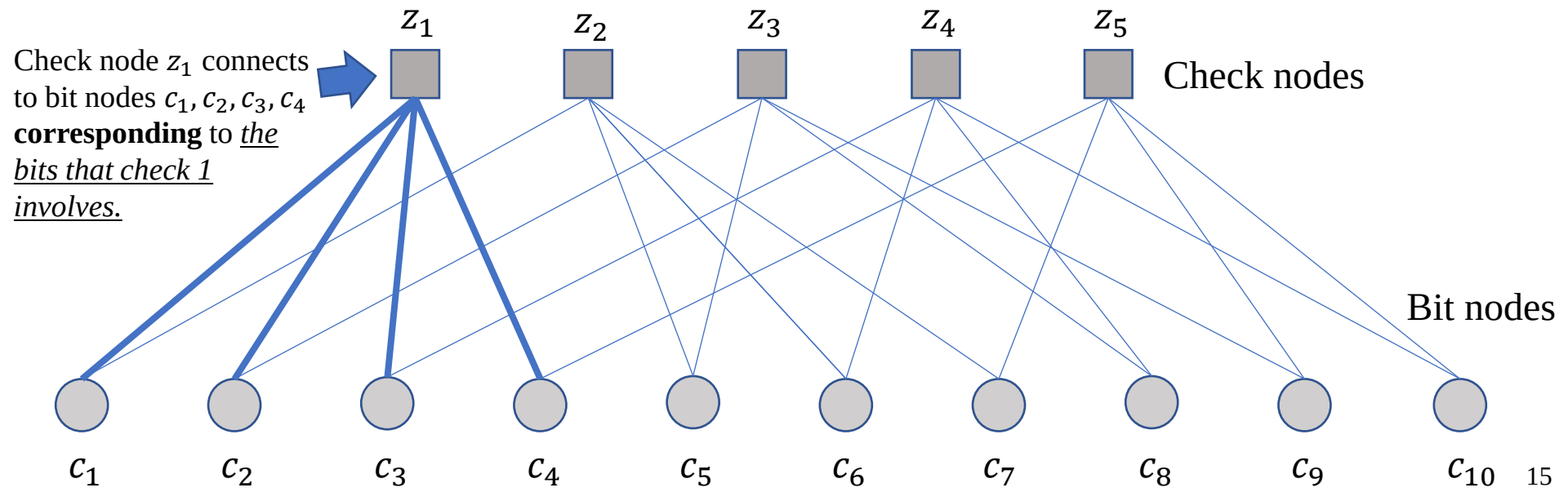


Representation of LDPC codes (3)

Example: A 5×10 parity check matrix A has $w_r = 4, w_c = 2$

- M = number of checks = 5 => **5 check nodes**
- N = number of bits = 10 => *10 bit nodes*
- Each **check node** connect to $w_r = 4$ *bit nodes.*
- Each *bit node* connects to $w_c = 2$ **check nodes.**

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



Outline

I. Error Correction Code

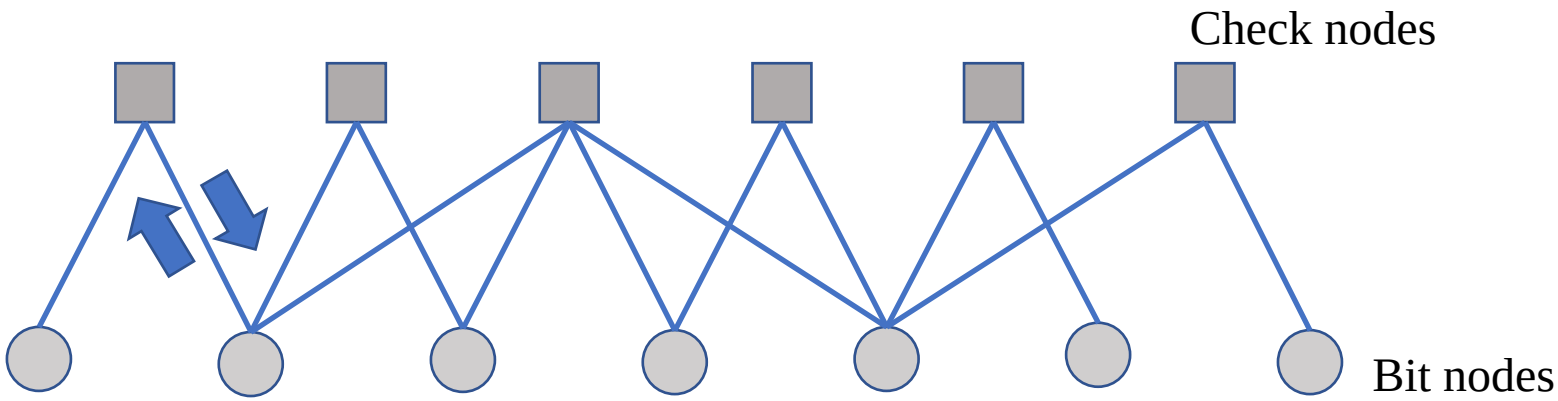
II. Low-density Parity-check Codes

1. Introduction
2. Hard-Decision Bit Flipping
3. Messaging-Passing Algorithm

III. Future Direction

Hard-Decision Bit Flipping

- The basic idea of **Hard-Decision Bit Flipping** is the information update between check and bit nodes in each iteration.
  - A check node uses the information of its bit nodes to compute the parity check.
  - A bit node uses the information of its check nodes to know how many parity checks it has failed.



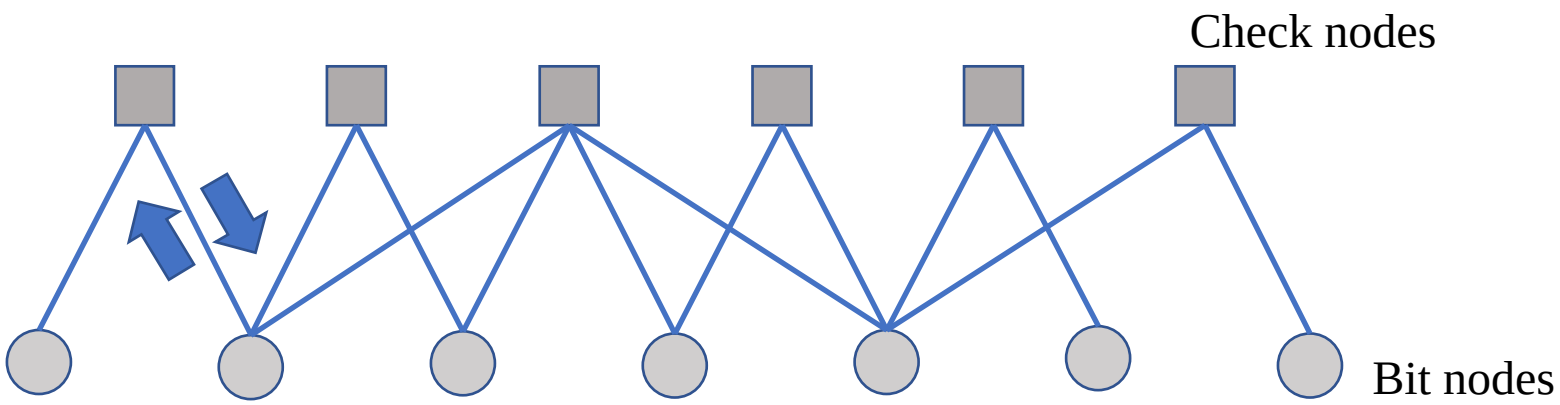
- The **more failed parity checks**, the more likely that the *bit node is wrong*.

Hard-Decision Bit Flipping: Algorithm

❑ The Hard-Decision Bit Flipping algorithm:

Initial: Set a maximum **number of iterations** L. For each iteration:

1. Compute each parity check. If all the parity checks are satisfied, the algorithm is stopped.
2. For each bit, count the number of failing parity checks.
3. For the bit(s) with the largest number of failed parity checks, flip the bits.
4. Increase the iteration counter. If the maximum number of iterations is reached, the algorithm is stopped.



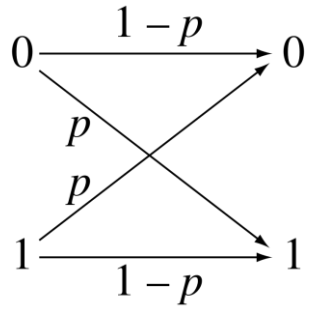
Hard-Decision Bit Flipping: An Example

- Consider an irregular parity check matrix A

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

Original bits

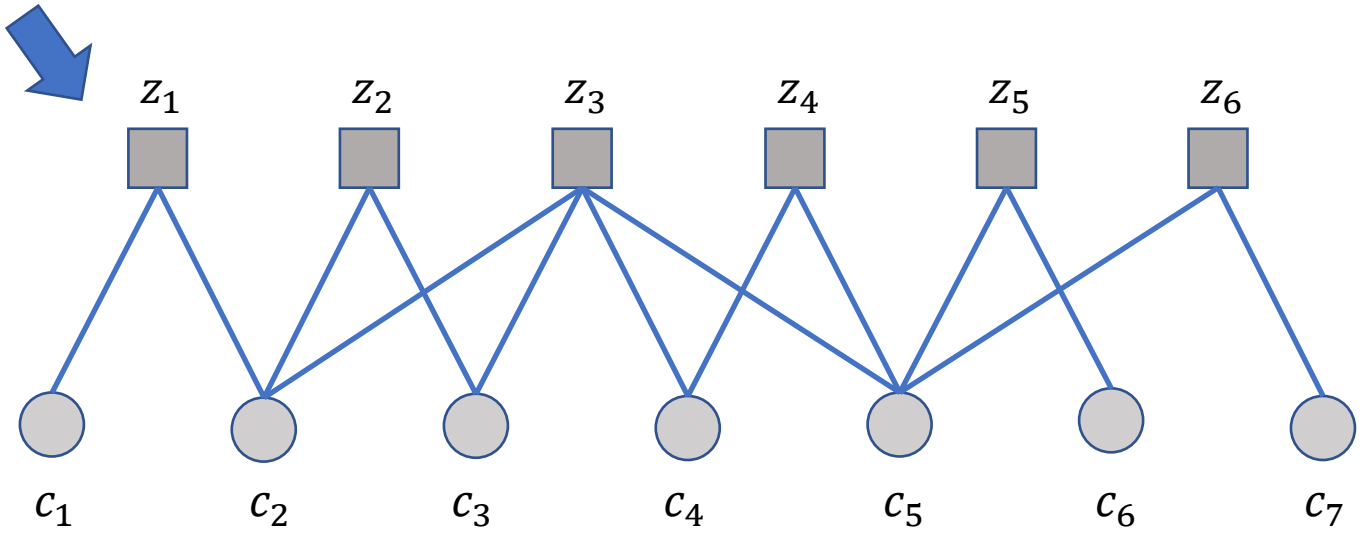
0 0 0 0 0 0 0



Binary Symmetric Channel (BSC)

Received bits

0 1 0 0 1 0 0



Hard-Decision Bit Flipping: An Example

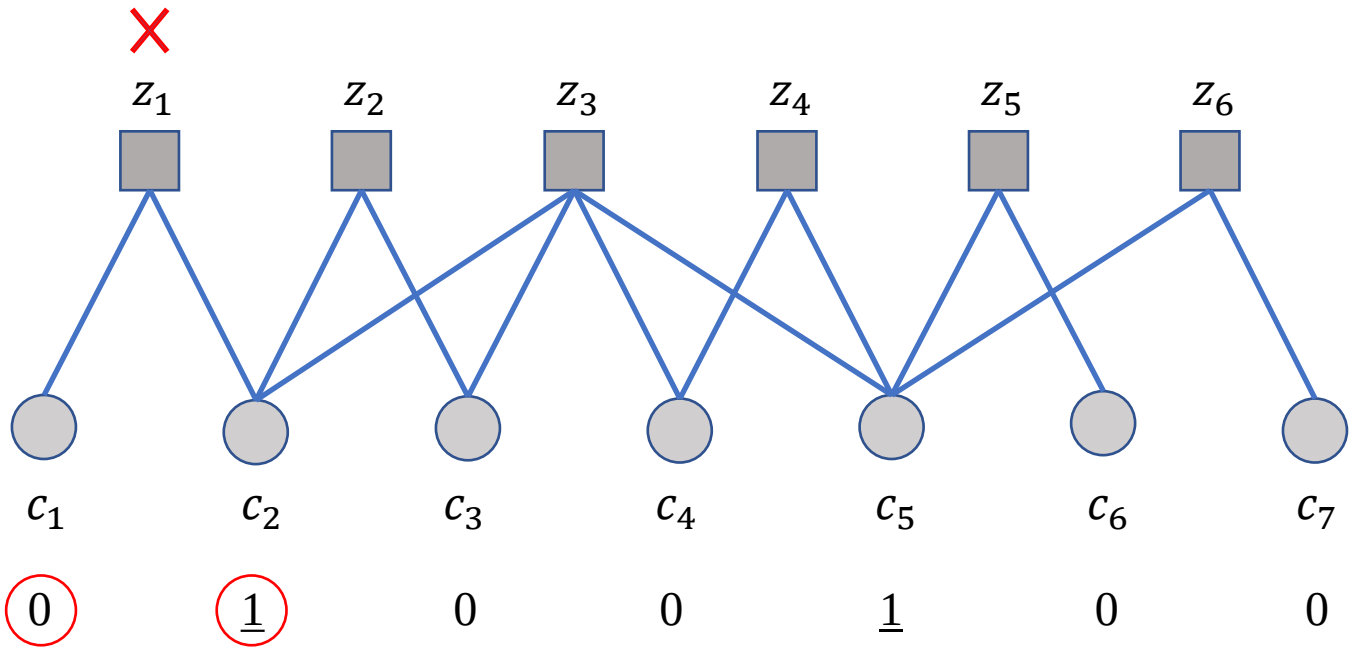
❑ Iteration 1:

1. Compute each parity check. If all the parity checks are satisfied, the algorithm is stopped.
2. For each bit, count the number of failing parity checks.
3. For the bit(s) with the largest number of failed parity checks, flip the bits.
4. Increase the iteration counter. If the maximum number of iterations is reached, the algorithm is stopped.

Failed parity check

Successful parity check

Original bits: 0 0 0 0 0 0 0



Hard-Decision Bit Flipping: An Example

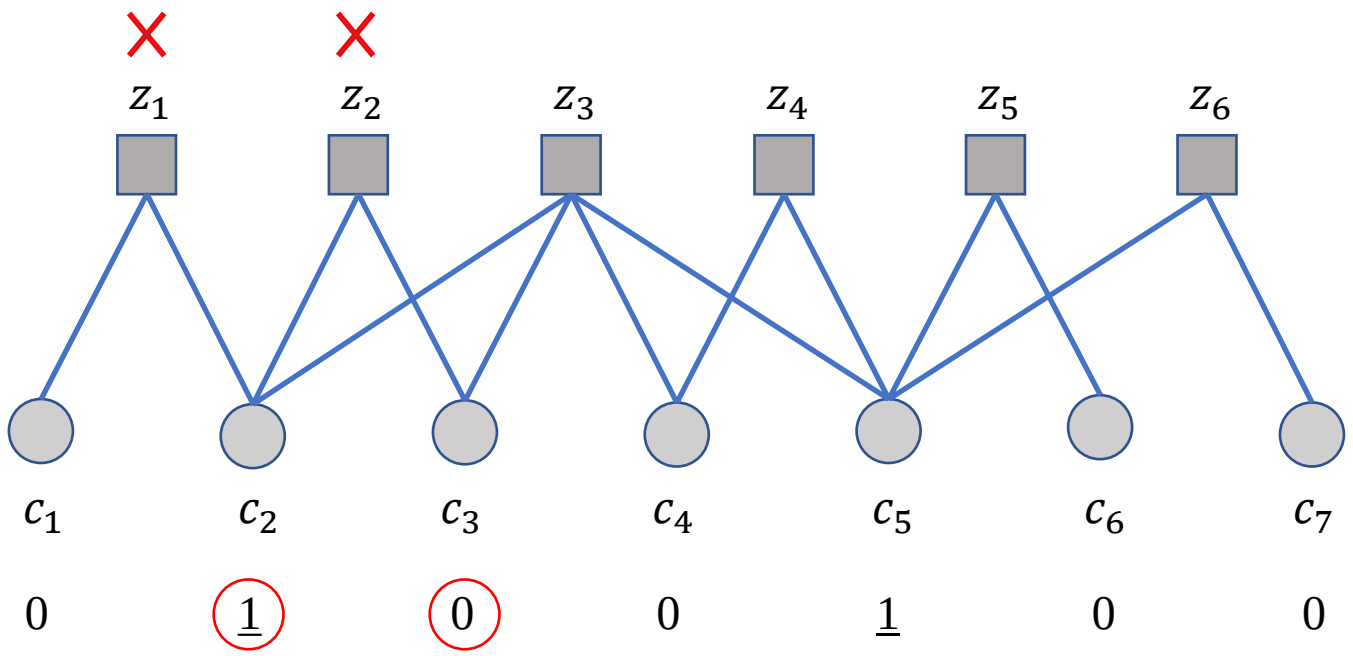
❑ Iteration 1:

1. Compute each parity check. If all the parity checks are satisfied, the algorithm is stopped.
2. For each bit, count the number of failing parity checks.
3. For the bit(s) with the largest number of failed parity checks, flip the bits.
4. Increase the iteration counter. If the maximum number of iterations is reached, the algorithm is stopped.

Failed parity check

Successful parity check

Original bits: 0 0 0 0 0 0 0



Hard-Decision Bit Flipping: An Example

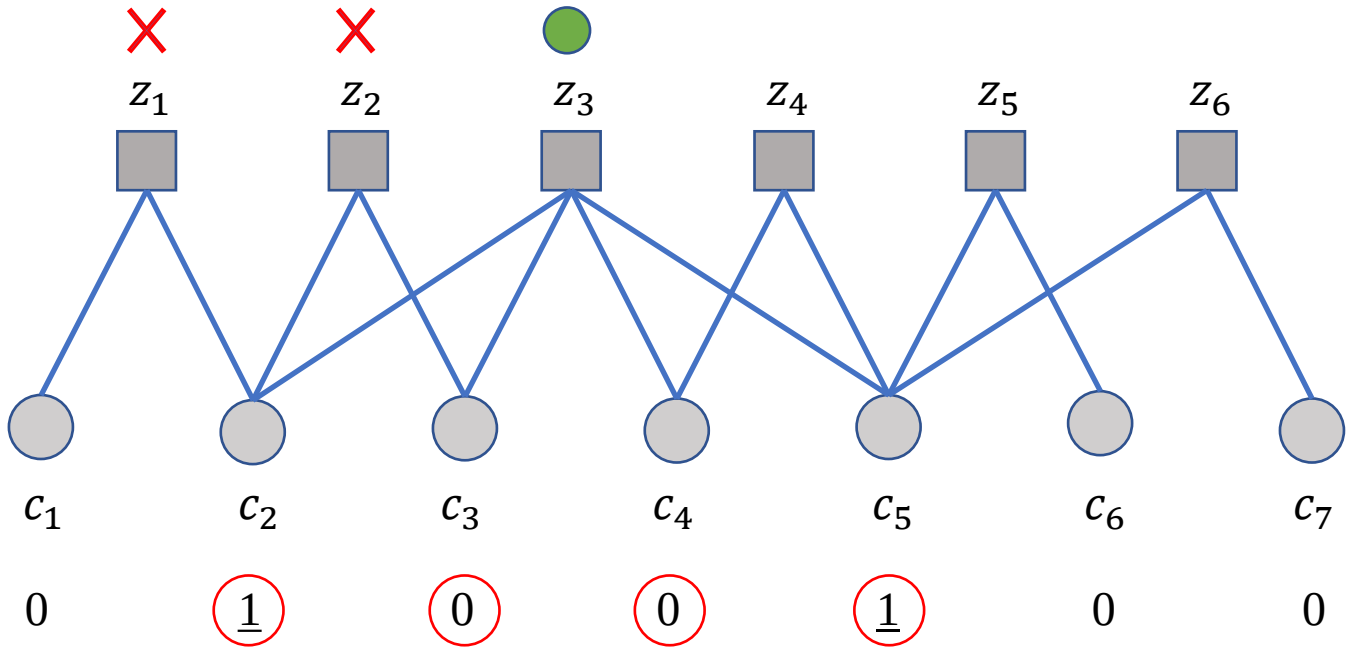
❑ Iteration 1:

1. Compute each parity check. If all the parity checks are satisfied, the algorithm is stopped.
2. For each bit, count the number of failing parity checks.
3. For the bit(s) with the largest number of failed parity checks, flip the bits.
4. Increase the iteration counter. If the maximum number of iterations is reached, the algorithm is stopped.

Failed parity check

Successful parity check

Original bits: 0 0 0 0 0 0 0



Hard-Decision Bit Flipping: An Example

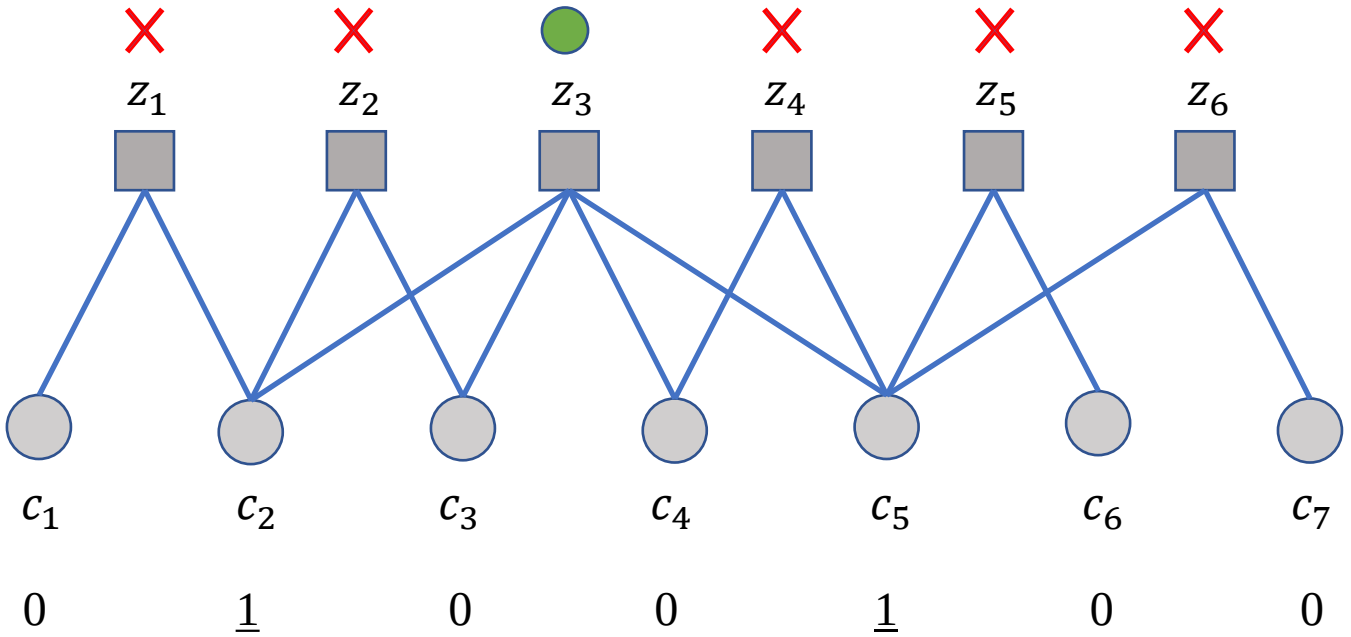
❑ Iteration 1:

1. Compute each parity check. If all the parity checks are satisfied, the algorithm is stopped.
2. For each bit, count the number of failing parity checks.
3. For the bit(s) with the largest number of failed parity checks, flip the bits.
4. Increase the iteration counter. If the maximum number of iterations is reached, the algorithm is stopped.

Failed parity check

Successful parity check

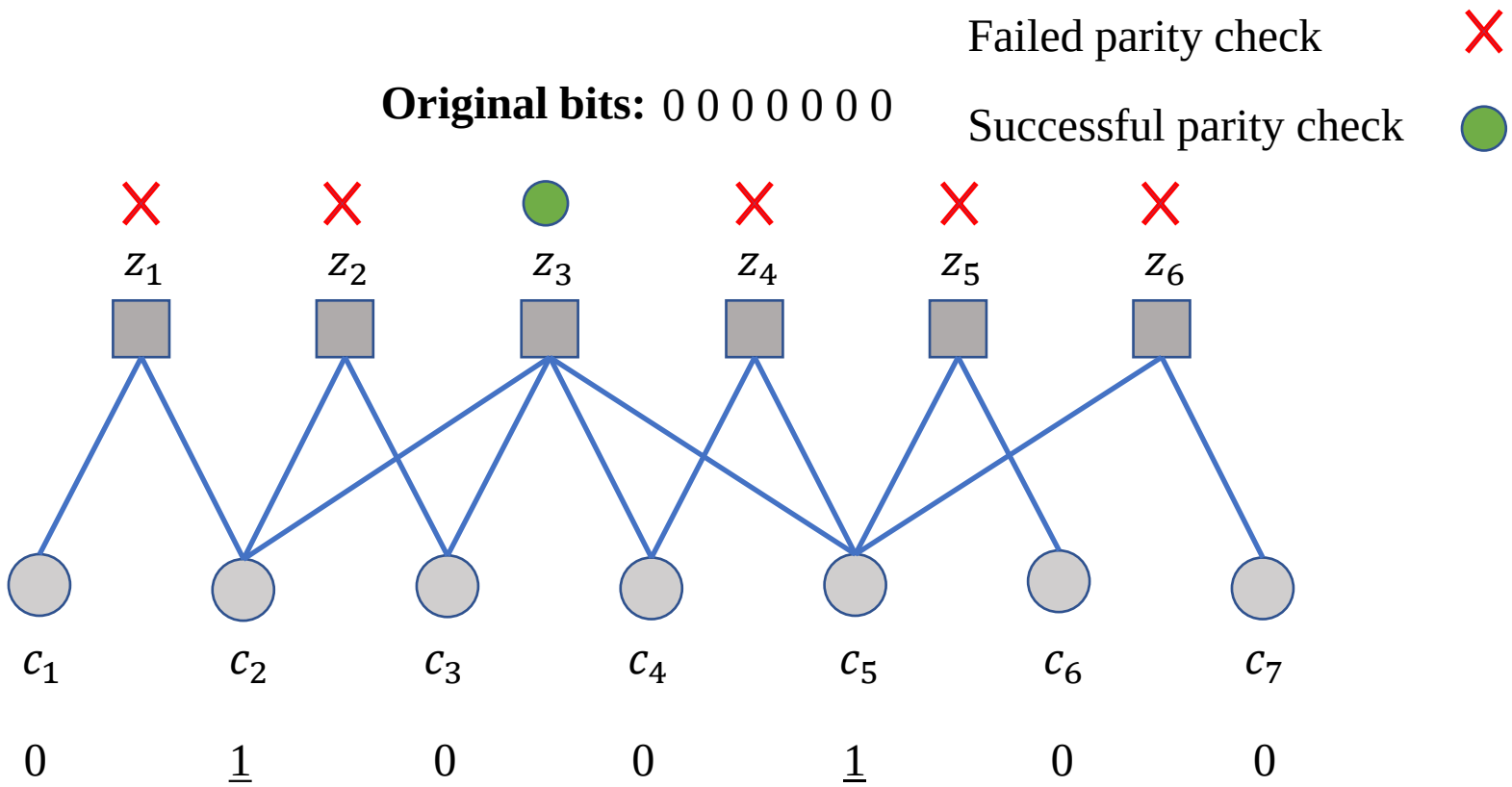
Original bits: 0 0 0 0 0 0 0



Hard-Decision Bit Flipping: An Example

❑ Iteration 1:

1. Compute each parity check. If all the parity checks are satisfied, the algorithm is stopped.
2. For each bit, count the number of failing parity checks.
3. For the bit(s) with the largest number of failed parity checks, flip the bits.
4. Increase the iteration counter. If the maximum number of iterations is reached, the algorithm is stopped.

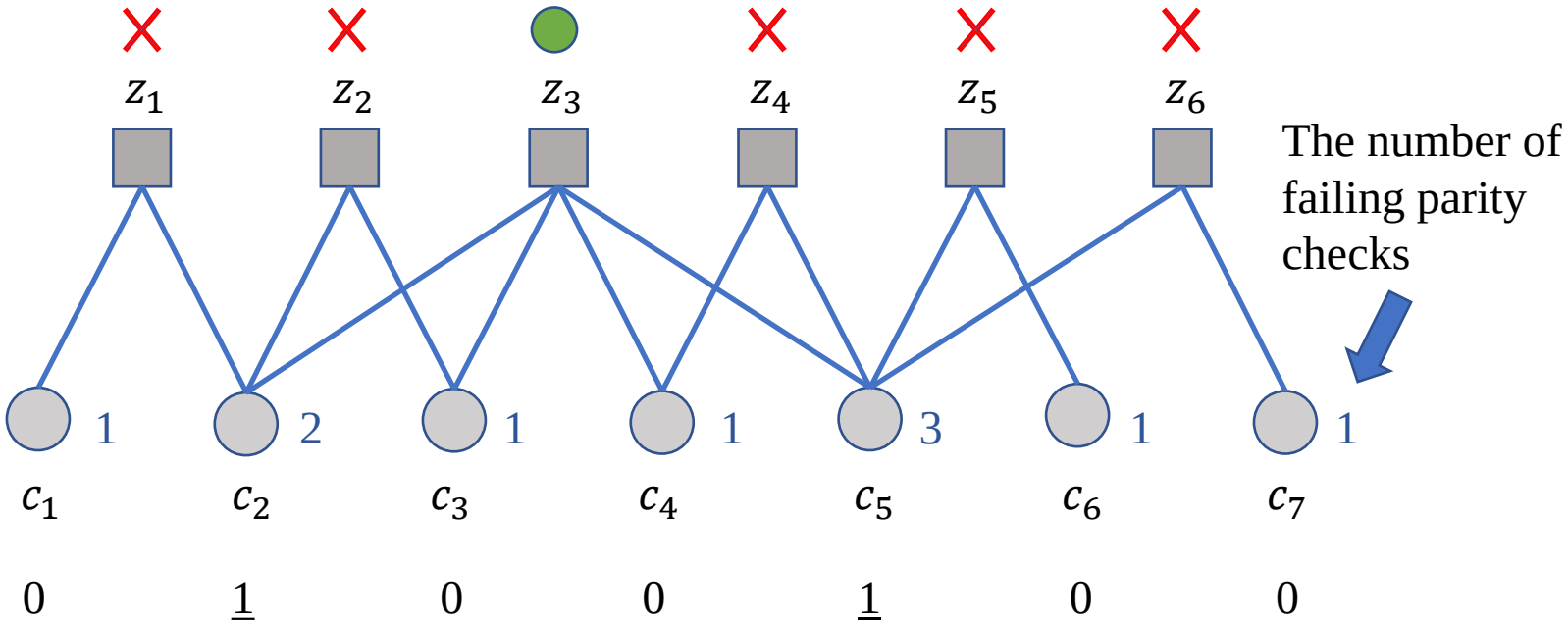


Hard-Decision Bit Flipping: An Example

- Iteration 1:

1. Compute each parity check. If all the parity checks are satisfied, the algorithm is stopped.
2. For each bit, count the number of failing parity checks.
3. For the bit(s) with the largest number of failed parity checks, flip the bits.
4. Increase the iteration counter. If the maximum number of iterations is reached, the algorithm is stopped.

Original bits: 0 0 0 0 0 0 0

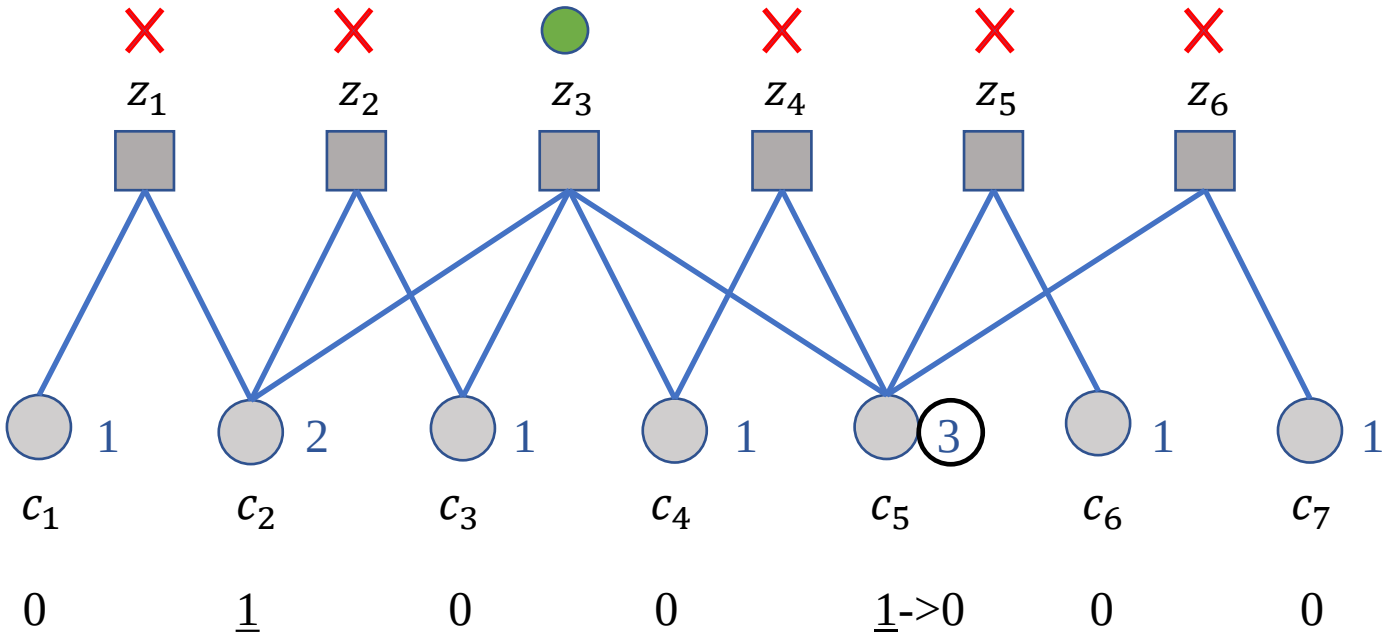


Hard-Decision Bit Flipping: An Example

- Iteration 1:

1. Compute each parity check. If all the parity checks are satisfied, the algorithm is stopped.
2. For each bit, count the number of failing parity checks.
3. For the bit(s) with the largest number of failed parity checks, flip the bits.
4. Increase the iteration counter. If the maximum number of iterations is reached, the algorithm is stopped.

Original bits: 0 0 0 0 0 0 0

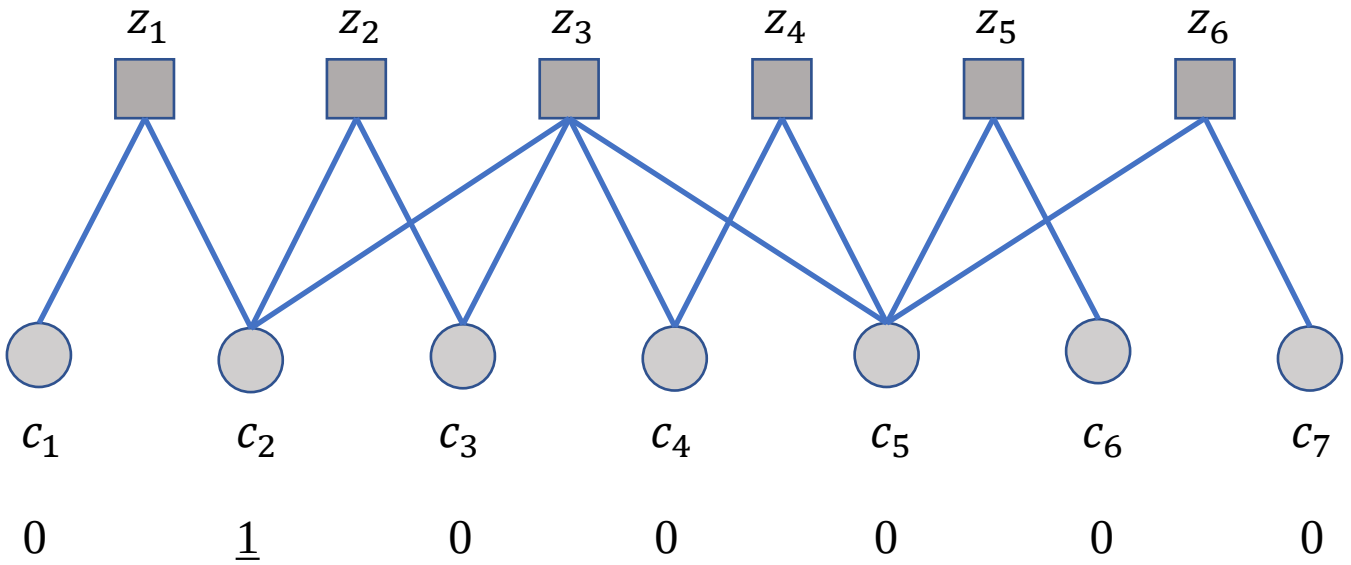


Hard-Decision Bit Flipping: An Example

- Iteration 1:

1. Compute each parity check. If all the parity checks are satisfied, the algorithm is stopped.
2. For each bit, count the number of failing parity checks.
3. For the bit(s) with the largest number of failed parity checks, flip the bits.
4. Increase the iteration counter. If the maximum number of iterations is reached, the algorithm is stopped.

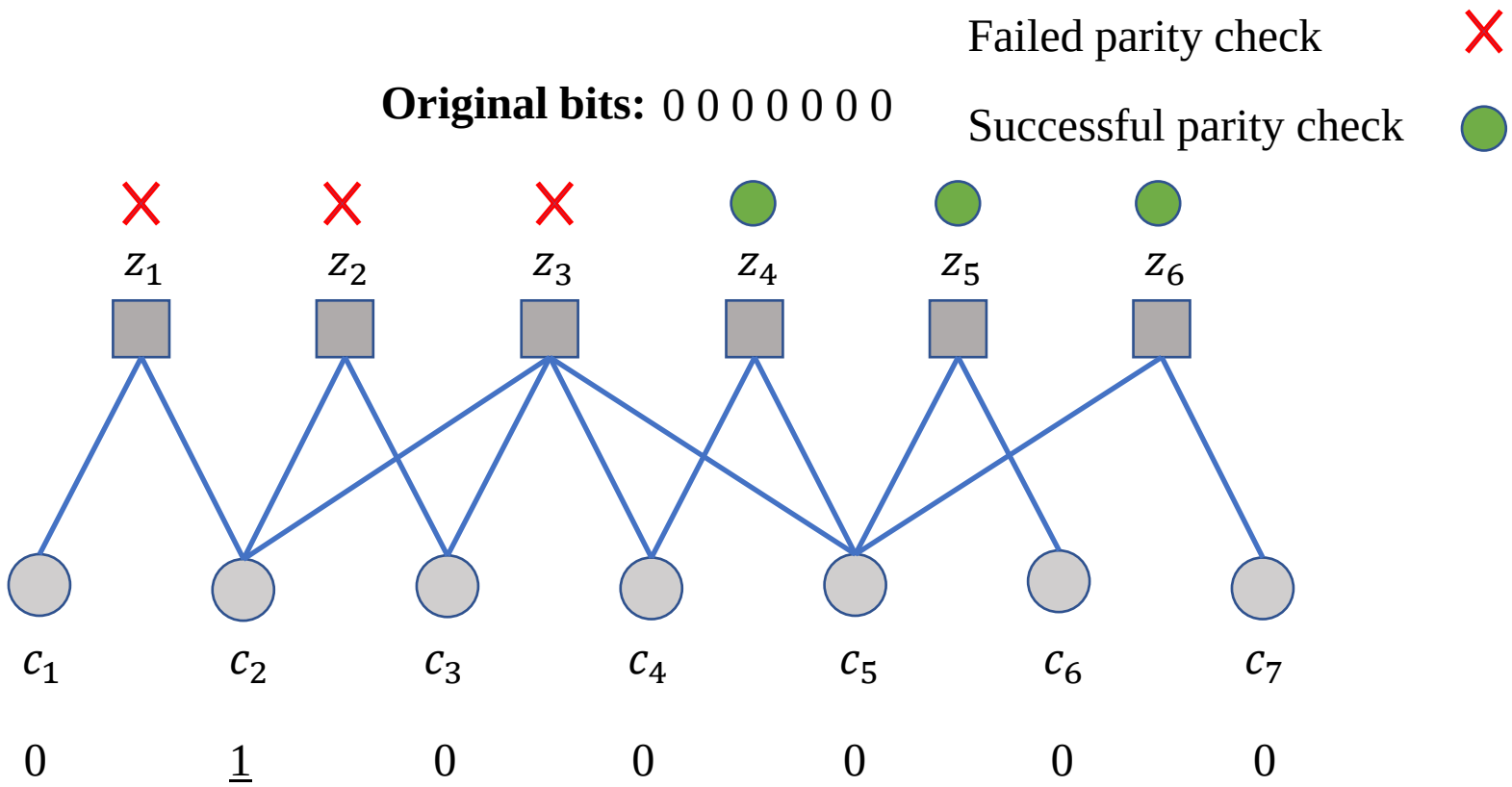
Original bits: 0 0 0 0 0 0 0



Hard-Decision Bit Flipping: An Example

❑ Iteration 2:

1. Compute each parity check. If all the parity checks are satisfied, the algorithm is stopped.
2. For each bit, count the number of failing parity checks.
3. For the bit(s) with the largest number of failed parity checks, flip the bits.
4. Increase the iteration counter. If the maximum number of iterations is reached, the algorithm is stopped.

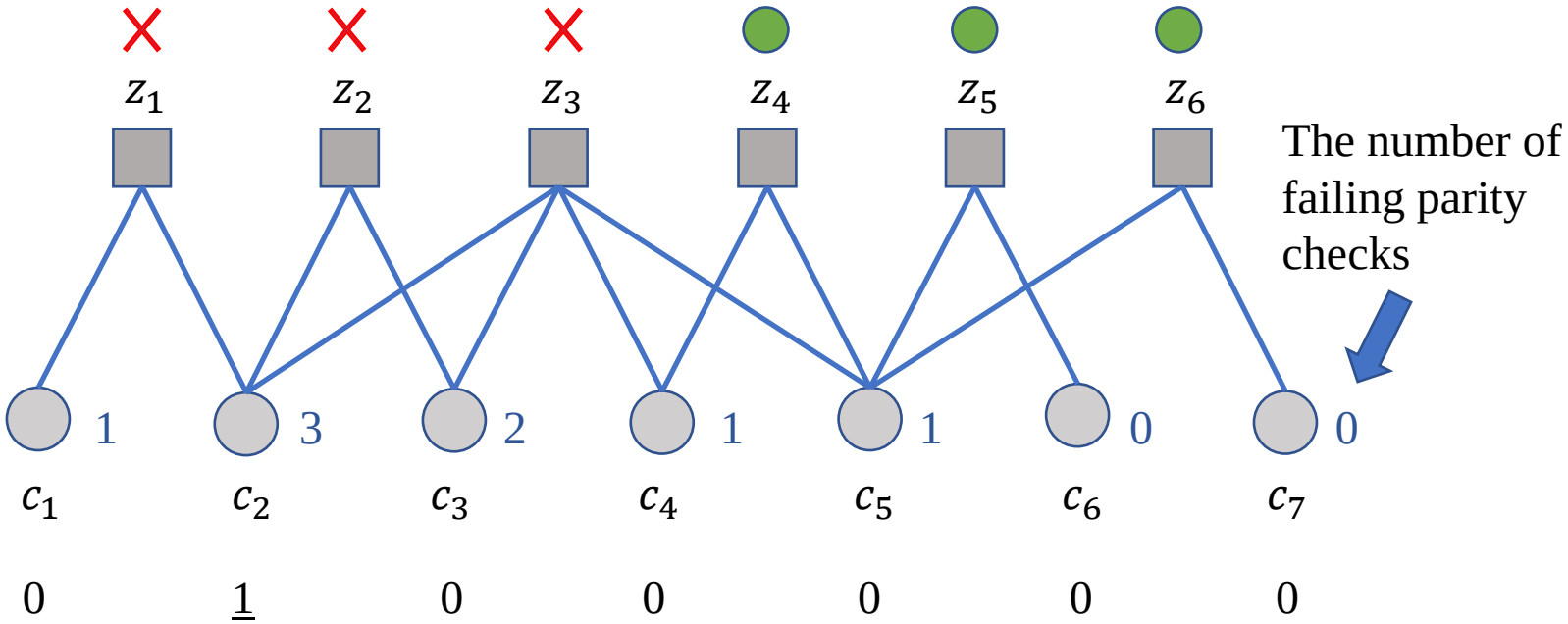


Hard-Decision Bit Flipping: An Example

- Iteration 2:

1. Compute each parity check. If all the parity checks are satisfied, the algorithm is stopped.
2. For each bit, count the number of failing parity checks.
3. For the bit(s) with the largest number of failed parity checks, flip the bits.
4. Increase the iteration counter. If the maximum number of iterations is reached, the algorithm is stopped.

Original bits: 0 0 0 0 0 0 0

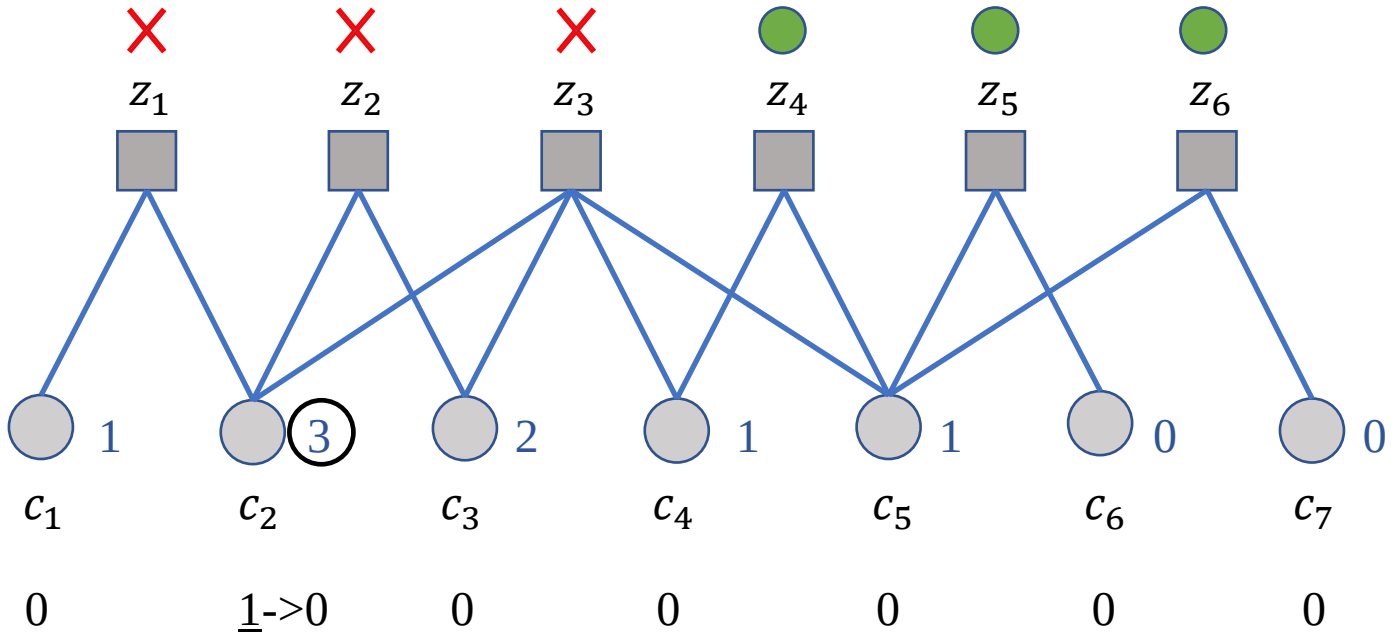


Hard-Decision Bit Flipping: An Example

- Iteration 2:

1. Compute each parity check. If all the parity checks are satisfied, the algorithm is stopped.
2. For each bit, count the number of failing parity checks.
3. For the bit(s) with the largest number of failed parity checks, flip the bits.
4. Increase the iteration counter. If the maximum number of iterations is reached, the algorithm is stopped.

Original bits: 0 0 0 0 0 0 0

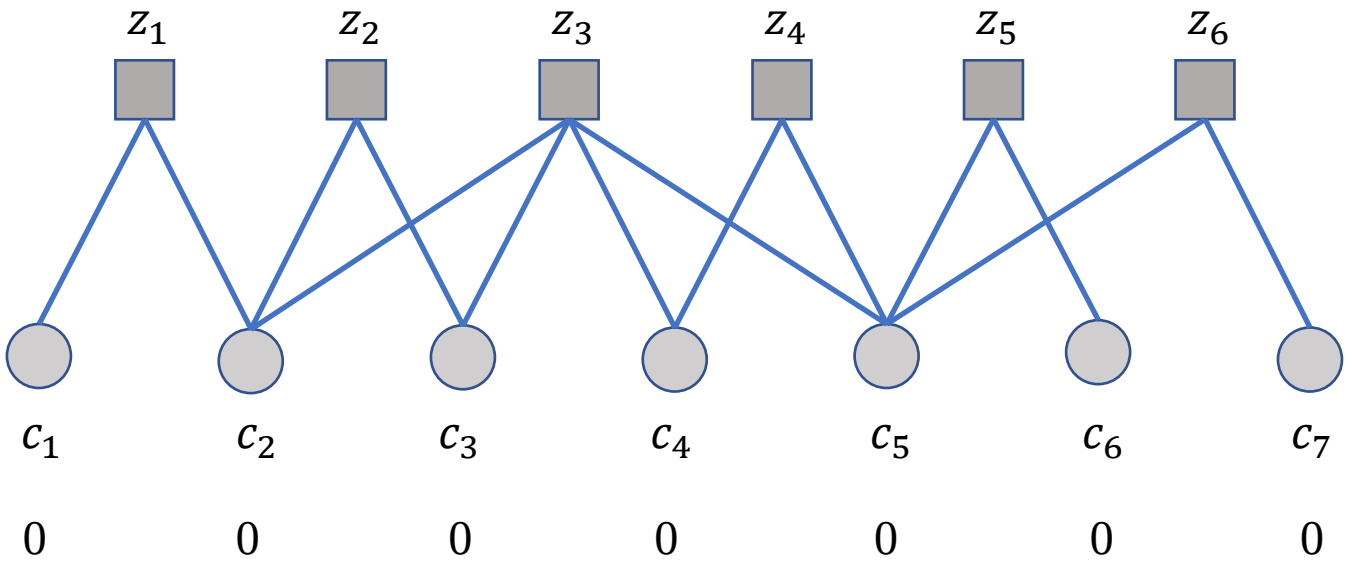


Hard-Decision Bit Flipping

- Iteration 2:

1. Compute each parity check. If all the parity checks are satisfied, the algorithm is stopped.
2. For each bit, count the number of failing parity checks.
3. For the bit(s) with the largest number of failed parity checks, flip the bits.
4. Increase the iteration counter. If the maximum number of iterations is reached, the algorithm is stopped.

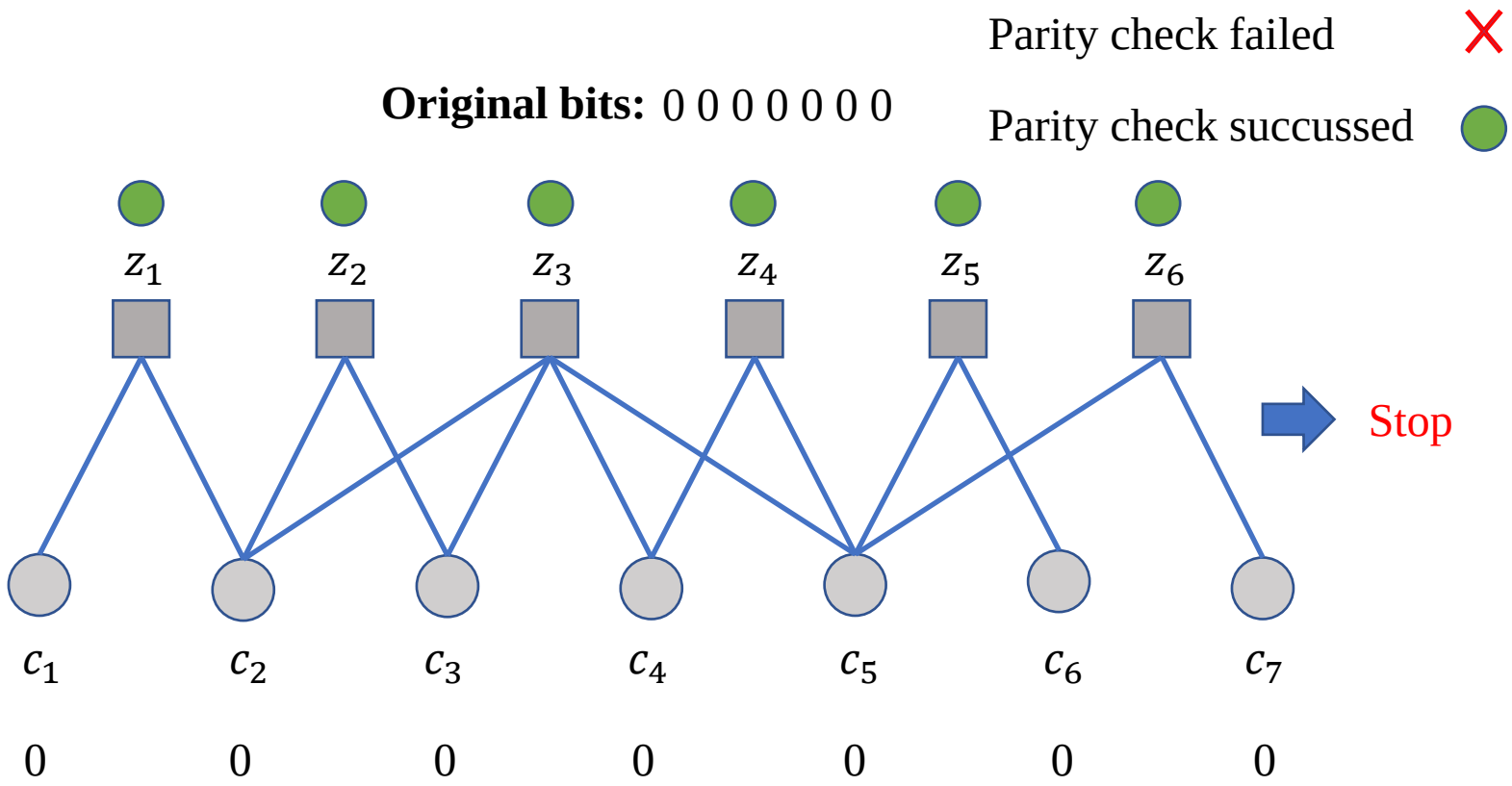
Original bits: 0 0 0 0 0 0 0



Hard-Decision Bit Flipping: An Example

❑ Iteration 3:

1. Compute each parity check. If all the parity checks are satisfied, the algorithm is stopped.
2. For each bit, count the number of failing parity checks.
3. For the bit(s) with the largest number of failed parity checks, flip the bits.
4. Increase the iteration counter. If the maximum number of iterations is reached, the algorithm is stopped.



Hard-Decision Bit Flipping: An Example

The received codeword is recovered successfully

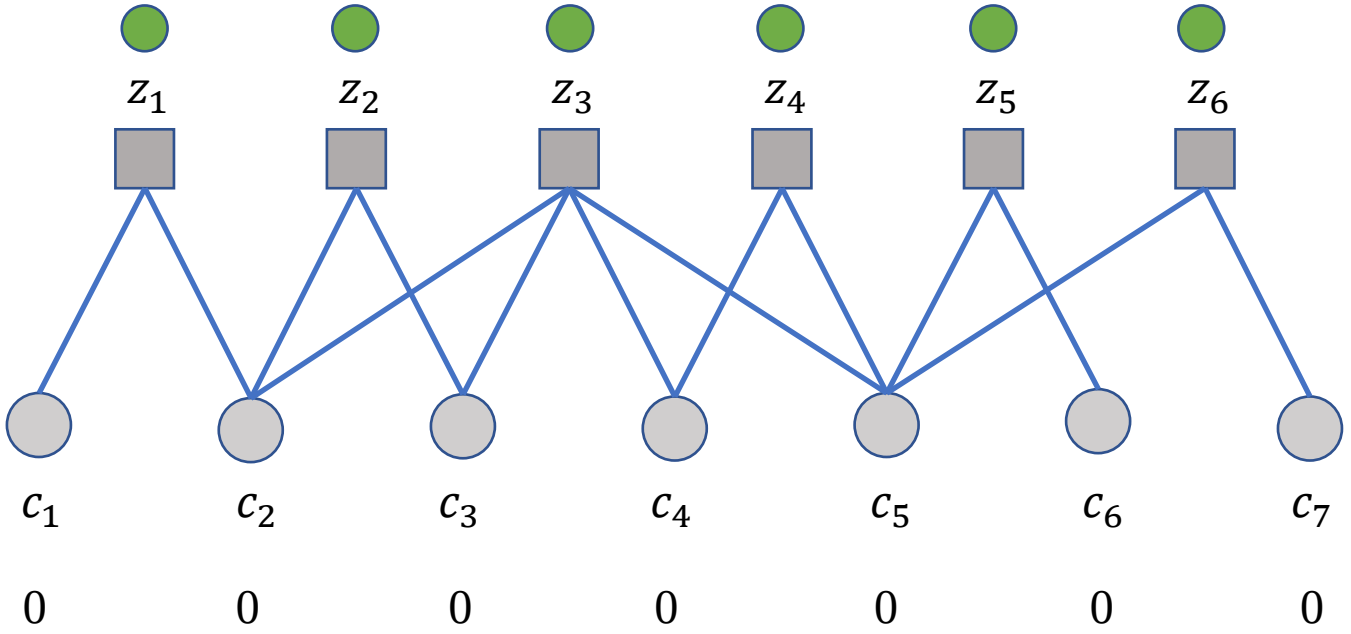
Original bits: 0 0 0 0 0 0 0

Received bits: 0 1 0 0 1 0 0

Recovered bits: 0 0 0 0 0 0 0

Failed parity check

Successful parity check



Outline

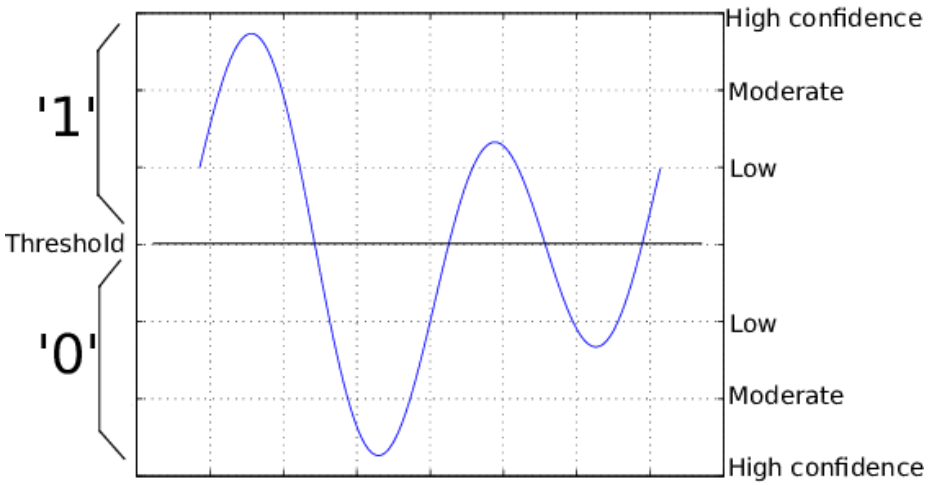
I. Error Correction Code

II. Low-density Parity-check Codes
   1. Introduction
   2. Hard-Decision Bit Flipping
   3. Messaging-Passing Algorithm

III. Future Direction

Soft-decision Decoder (1)

- The basic idea of **Hard-Decision Bit Flipping** is *the information update between check and bit nodes in each iteration.*
- **Message-passing Algorithm** is based on this idea but for *soft-decision decoding.*
- **Hard-decision decoder** operates on data that take on a *fixed set of possible values (commonly 0 and 1).*
  - In real life, the received signal is fluctuating
  - Treats the different signal levels as the same value
- **Soft-decision decoder** takes on *a whole range of values in between.*
  - Uses the information from the physical layer to "helps" the decoder make the decision better.

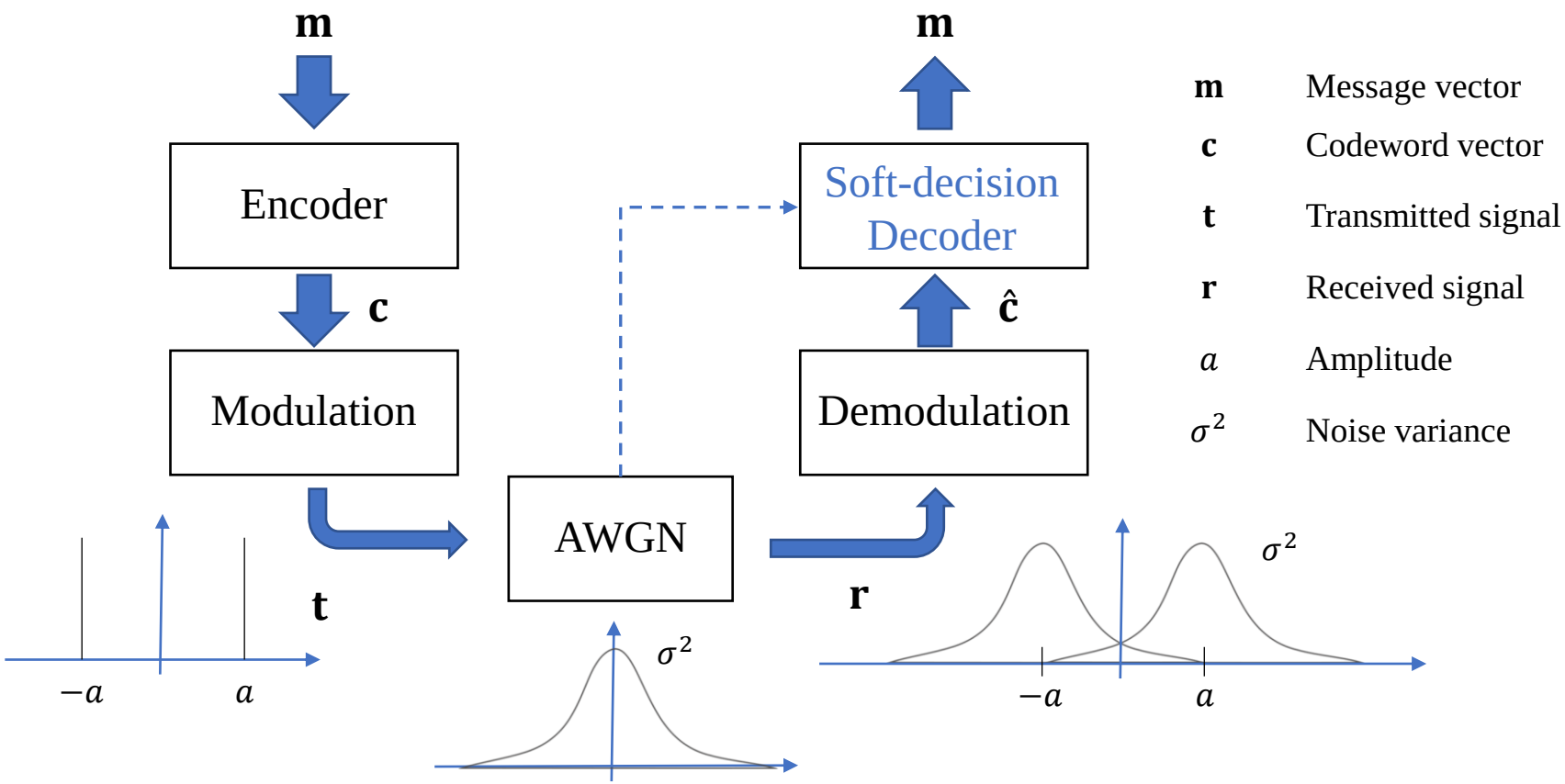


Soft-decision Decoder (2)

- The message-passing algorithm considers **the channel posterior probability**.
  - Measure the "certainty" of bit c_n *given the observation of the received signal*, **r.**

- **Example:** For binary phase-shift Keying (BPSK) modulation, the channel posterior probability is given as

$$p_n(x) = P(c_n = x|\mathbf{r}) = \frac{1}{1 + e^{-2ar_n/\sigma^2}}$$



Message-passing Algorithms: $q_n(x)$ (1)

Target of the algorithm: The decoder tries to evaluate the probabilities

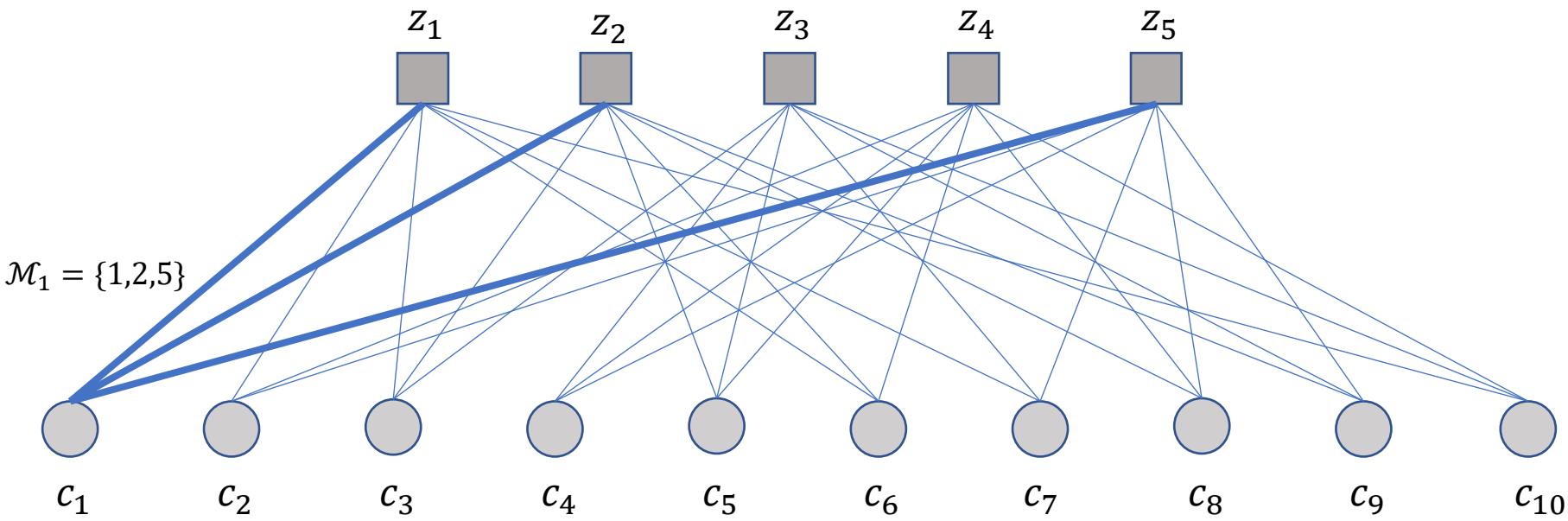
$$q_n(x) = P(c_n = x | \mathbf{r}, \{z_m = 0, \mathrm{m} \in \mathcal{M}_n\})$$

c_n: n^{th} bit
z_m: m^{th} check

where $\mathcal{M}_n$ is the set of checks in which bit c_n participates

$$\mathcal{M}_n = \{m: A_{mn} = 1\}$$

- $q_n(0) + q_n(1) = 1$
- Presents the probability that bit $c_n = x$ given the *observation of received signal* and <u>all checks involving c_n are satisfied.</u>
- The higher $q_n(x)$, the more likely that $c_n = x$.



Message-passing Algorithms: $q_n(x)$ (2)

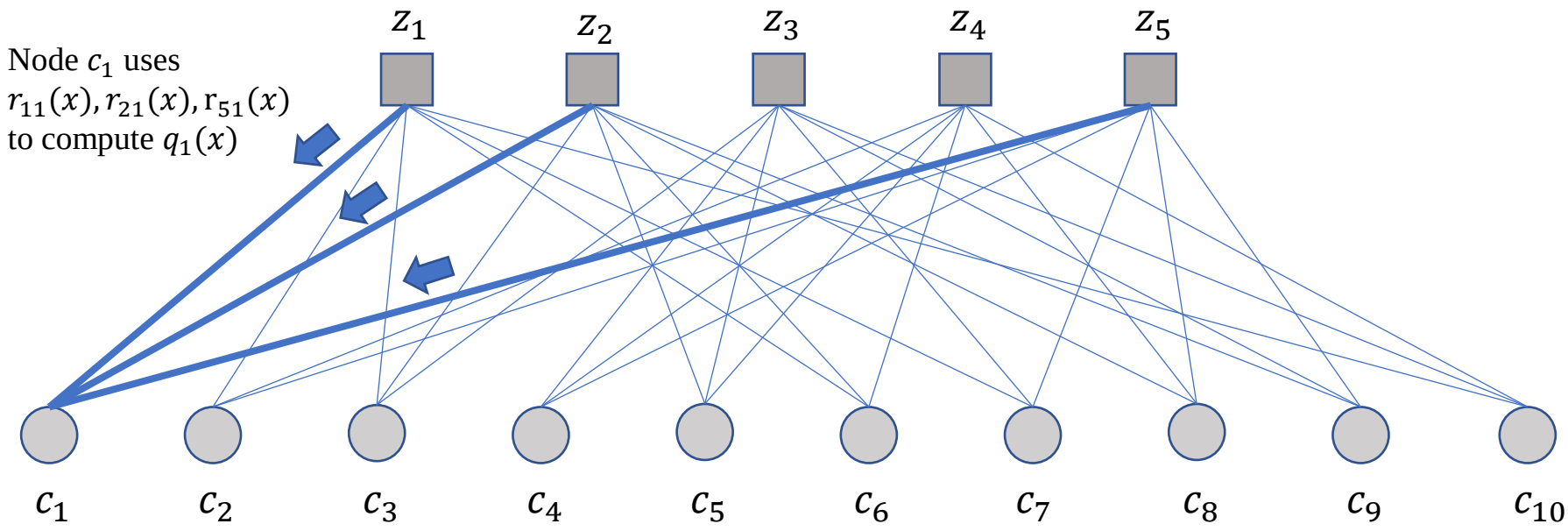
How to compute $q_n(x)$?

$$q_n(x) = \frac{p_n(x) \prod_{m \in \mathcal{M}_n} P(z_m = 0 | c_n = x, \mathbf{r})}{\sum_{x' \in \{0,1\}} p_n(x') \prod_{m \in \mathcal{M}_n} P(z_m = 0 | c_n = x', \mathbf{r})}$$

$p_n(x)$: the posterior probability

- $r_{mn}(x) = P(z_m = 0 | c_n = x, \mathbf{r})$ presents *the probability that check node* z_m *is satisfied given* $c_n = x$.

=> To compute the probability $q_n(x)$, bit node c_n will use $r_{mn}(x)$ from its involving check nodes.



Message-passing Algorithms : r_{mn} (1)

How to compute r_{mn}?

$$r_{mn}(x) = \sum_{\{c_{n'}\}} P(z_m = 0 | c_n = x, \{c_{n'}\}) \prod_{n'} q_{mn'}(c_{n'}) \qquad n' \in \mathcal{N}(m) \backslash n$$

where

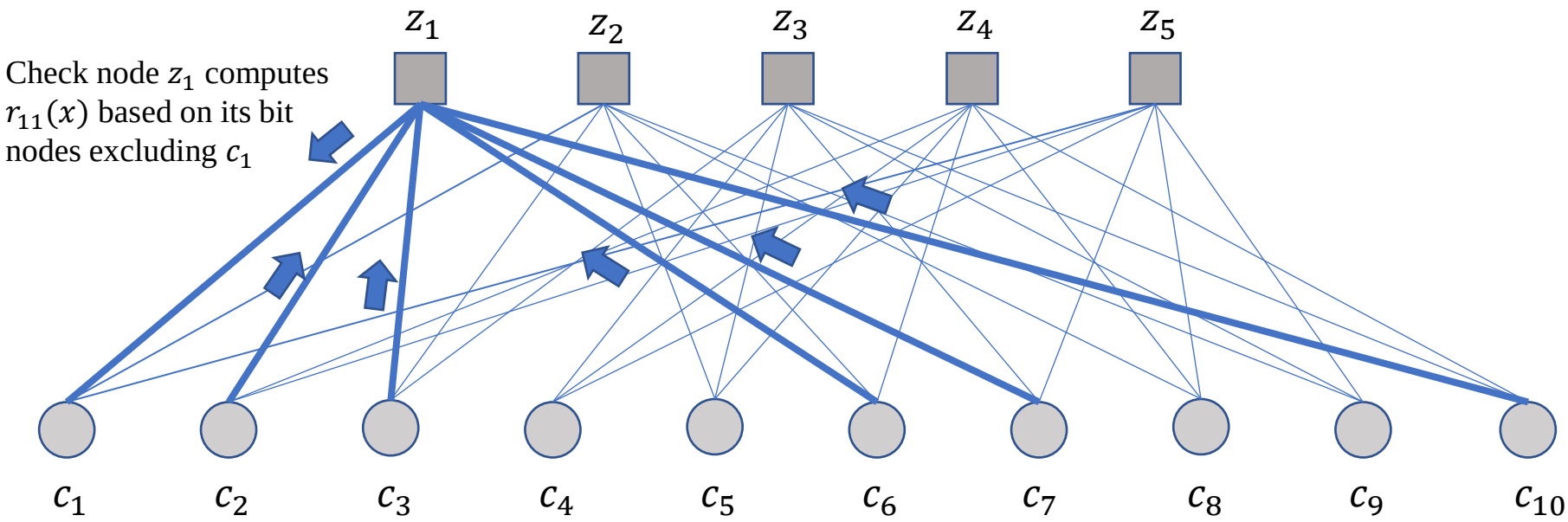
- $\mathcal{N}_m$ is the set of bits in which check z_m participates

$$\mathcal{N}_m = \{n : A_{mn} = 1\}$$

- $\mathcal{N}_m \backslash n$ is the set of bits in which check z_m participates, excluding bit c_n

E.g., $\mathcal{N}_1 = \{1,2,3,6,7,10\}$
$\mathcal{N}_1 \backslash 1 = \{2,3,6,7,10\}$

=> $r_{mn}(x)$ is computed based on bit nodes that z_m participates excluding c_n.

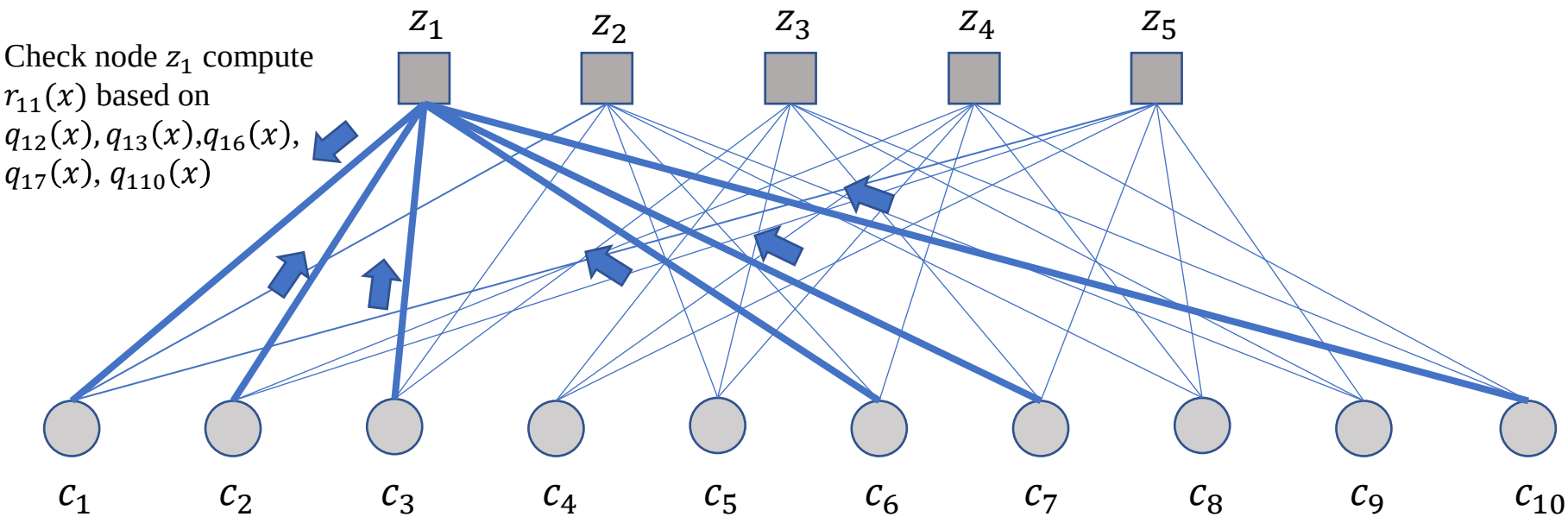


Message-passing Algorithms : r_{mn} (2)

How to compute r_{mn}?

$$r_{mn}(x) = \sum_{\{c_{n'}\}} P(z_m = 0 | c_n = x, \{c_{n'}\}) \prod_{n'} q_{mn'}(c_{n'}) \qquad n' \in \mathcal{N}(m) \backslash n$$

- $\{c_{n'}\}$ is set of bit node $c_{n'}$ with $n' \in \mathcal{N}_m \backslash n$
  - **E.g.**, $n = 1$, $\{c_{n'}\} = \{c_2, c_3, c_6, c_7, c_{10}\}$
  - This set ranges from {0,0,0,0,0} to {1,1,1,1,1}
- The probability $z_m = 0$ given the value of c_n and the $\{c_{n'}\}$ is **either 0 or 1.**



Message-passing Algorithms : q_{mn} (1)

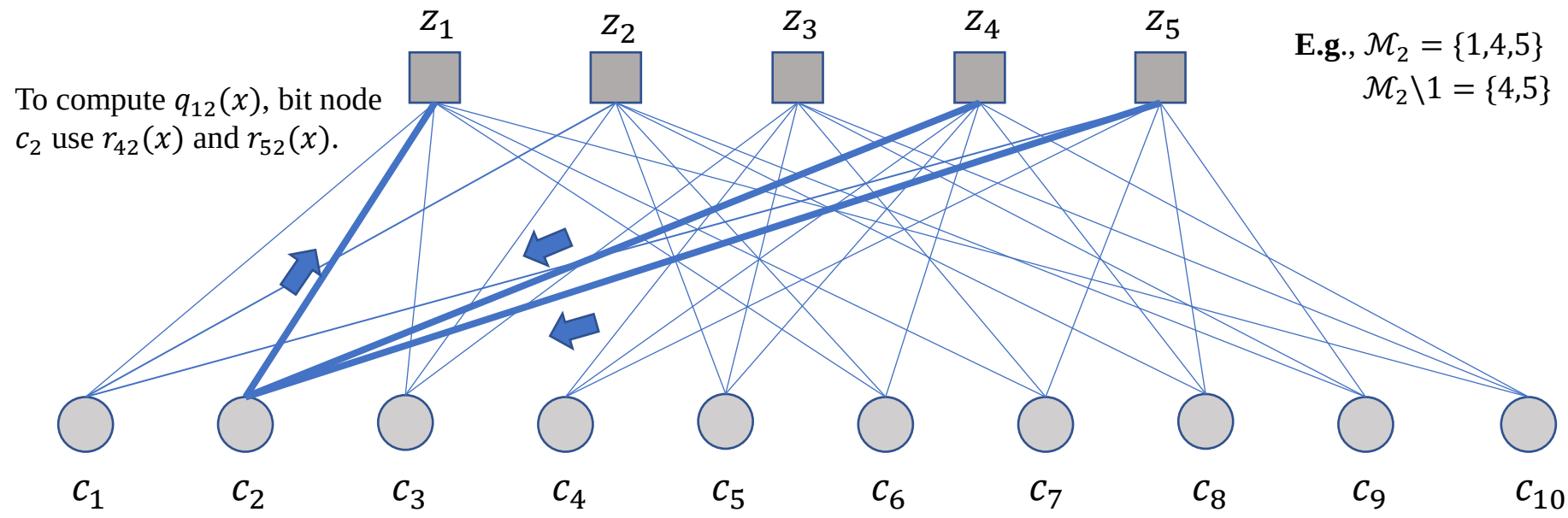
How to compute q_{mn}?

- $q_{mn}(x)$ presents the probability that *bit $c_n = x$ given all checks involving c_n, excluding z_m, are satisfied.*

  => Similar to $q_n(x)$, $q_{mn}(x)$ can be expressed as

$$q_{mn}(x) = \frac{p_n(x) \prod_{m' \in \mathcal{M}_n \backslash m} r_{m'n}}{\sum_{x' \in \{0,1\}} p_n(x') \prod_{m' \in \mathcal{M}_n \backslash m} r_{m'n}}$$

where $\mathcal{M}_n \backslash m$ is the set of checks in which bit c_n participates, excluding z_m.



The Relationship Between Probabilities (1)

- *Target of the algorithm: The decoder tries to evaluate the probabilities*

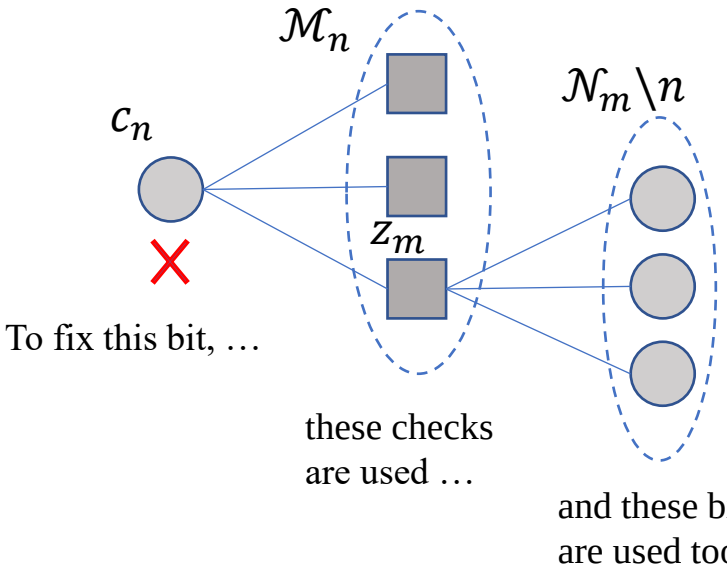
$$q_n(x) = P(c_n = x|\ \mathbf{r}, \text{all checks involving } c_n \text{ are satisfied})$$

- $q_n(x)$ is computed based on $r_{mn}(x)$, $m \in \mathcal{M}_n$

$$r_{mn}(x) = P(z_m = 0|\ c_n = x, \mathbf{r})$$

- $r_{mn}(x)$is computed based on $q_{mn'}(x)$, $n' \in \mathcal{N}_m \backslash n$

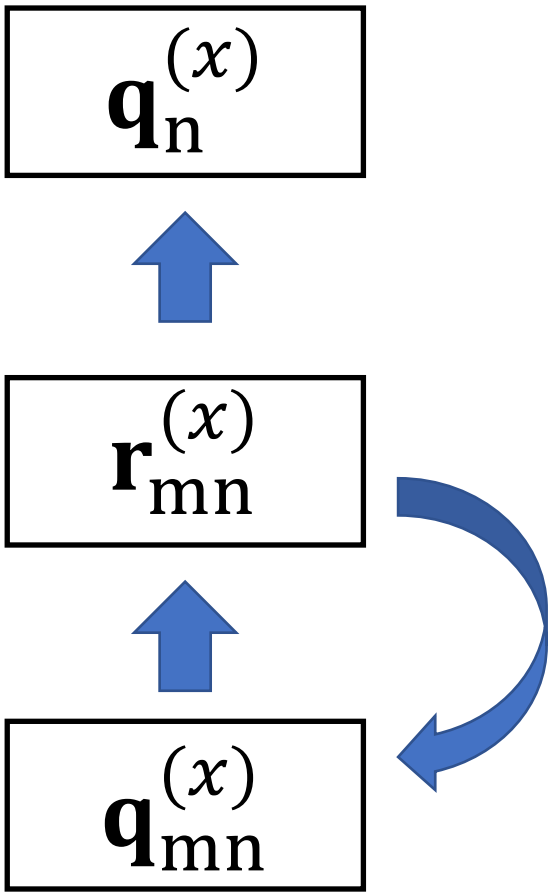
$$q_{mn}(x) = P(c_n = x|\ \mathbf{r}, \text{all checks, except } z_m, \text{involving } c_n \text{ are satisfied})$$



- Each bit is consider separately.
- Its probabilities is then used to compute other bits.

➡ By iteratively updating, all errors can be detected and corrected.

The Relationship Between Probabilities (2)



- $\mathbf{q}_{\mathrm{n}}^{(x)}$ represents a $N \times 1$ vector, in which the element in the position n is the probability $q_n(x)$.
- $\mathbf{r}_{\mathrm{mn}}^{(x)}$ represents a $M \times N$ matrix, in which the element in the position (m, n) is the probability $r_{mn}(x)$.
  - $0 < m < M, 0 < n < N$
- $\mathbf{q}_{\mathrm{mn}}^{(x)}$ represents a $M \times N$ matrix, in which the element in the position (m, n) is the probability $q_{mn}(x)$.
  - $0 < m < M, 0 < n < N$

Pseudocode of The Decoding Algorithm

Input: The parity check matrix A**,** the maximum number of iterations L and the channel posterior probabilities $p_n(x)$.
Initialization: Set $q_{mn}(x) = p_n(x)$ for all (m, n) with $A(m, n) = 1$.
For each iteration:

 # Check node update:

Update $\mathbf{r}_{\mathrm{mn}}^{(x)}$ by the value of $\mathbf{q}_{\mathrm{mn}}^{(x)}$.

 # Bit node update:

Update $\mathbf{q}_{\mathrm{mn}}^{(x)}$ by the value of $\mathbf{r}_{\mathrm{mn}}^{(x)}$.

 # Parity check:

Update $\mathbf{q}_{\mathrm{n}}^{(x)}$ by the value of $\mathbf{r}_{\mathrm{mn}}^{(x)}$.

For each elements in $\mathbf{q}_{\mathrm{n}}^{(x)}$:

If $q_n(x) > 0.5$: set $\hat{\boldsymbol{c}}_n = x$.

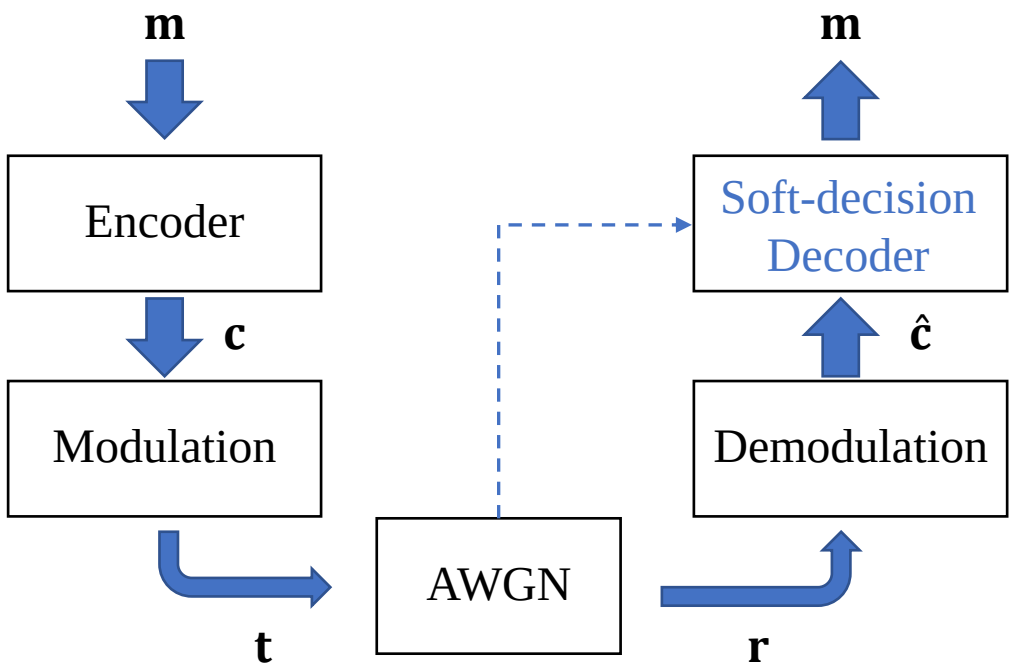
If $A\hat{\boldsymbol{c}} = 0$: break;

Message-passing Algorithms: An Example

- Example: Consider the system with the original message

$$\mathbf{m} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

- The BPSK is used with the amplitude $a = 2$, the noise variance $\sigma^2 = 2$.



$$\mathbf{c} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$\mathbf{t} = \begin{pmatrix} -2 & -2 & -2 & 2 & -2 & 2 & -2 & 2 & -2 & 2 \end{pmatrix}$$

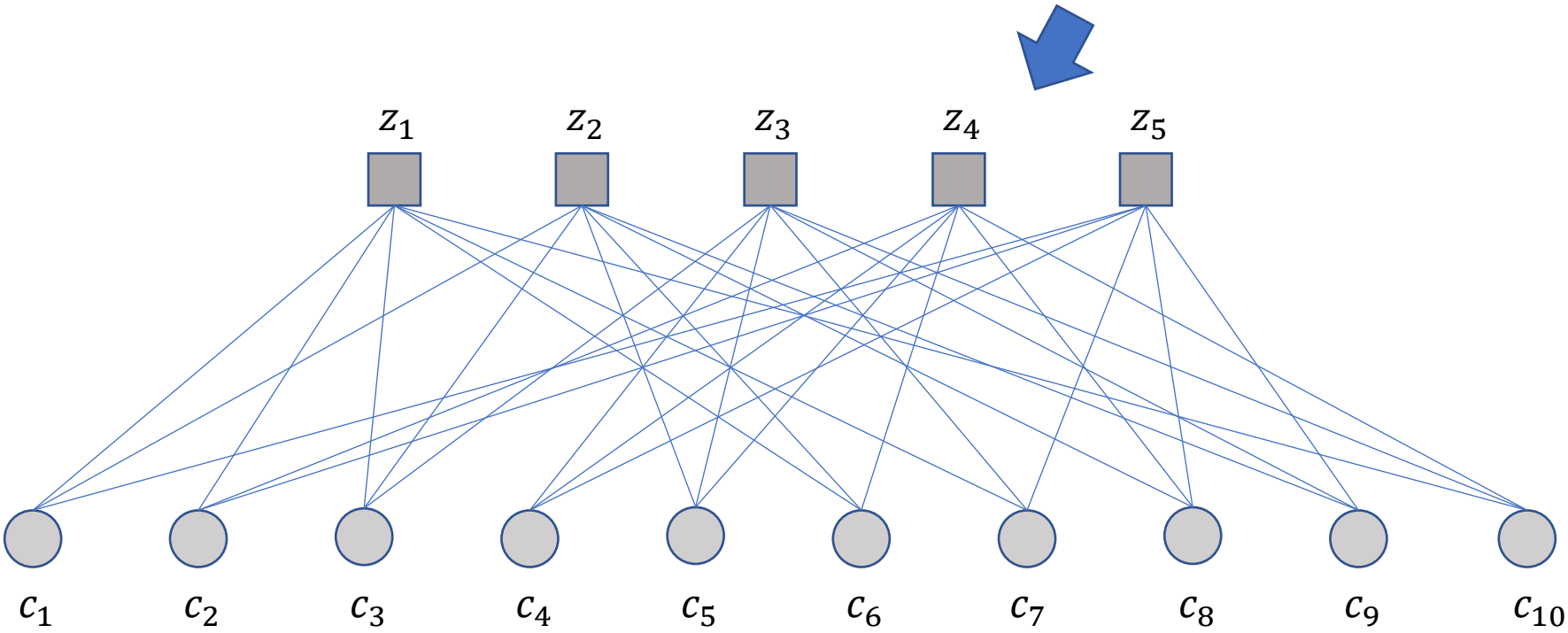
$$\mathbf{r} = \begin{pmatrix} -0.63 & -0.83 & -0.73 & -0.04 & 0.1 & 0.95 & -0.76 & 0.66 & -0.55 & 0.58 \end{pmatrix}$$

$$\hat{\mathbf{c}} = \begin{pmatrix} 0 & 0 & 0 & \underline{0} & \underline{1} & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Example: Parity Check Matrix

- The parity check matrix A is given as follows.
- It should be noted that the matrix A is not sparse. However, it is used in the example for illustrative purposes.

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



Initialization

- The channel posterior probabilities are

$$p_n(x) = \begin{bmatrix} 0.22 & 0.16 & 0.19 & 0.48 & 0.55 & 0.87 & 0.18 & 0.79 & 0.25 & 0.76 \end{bmatrix}$$

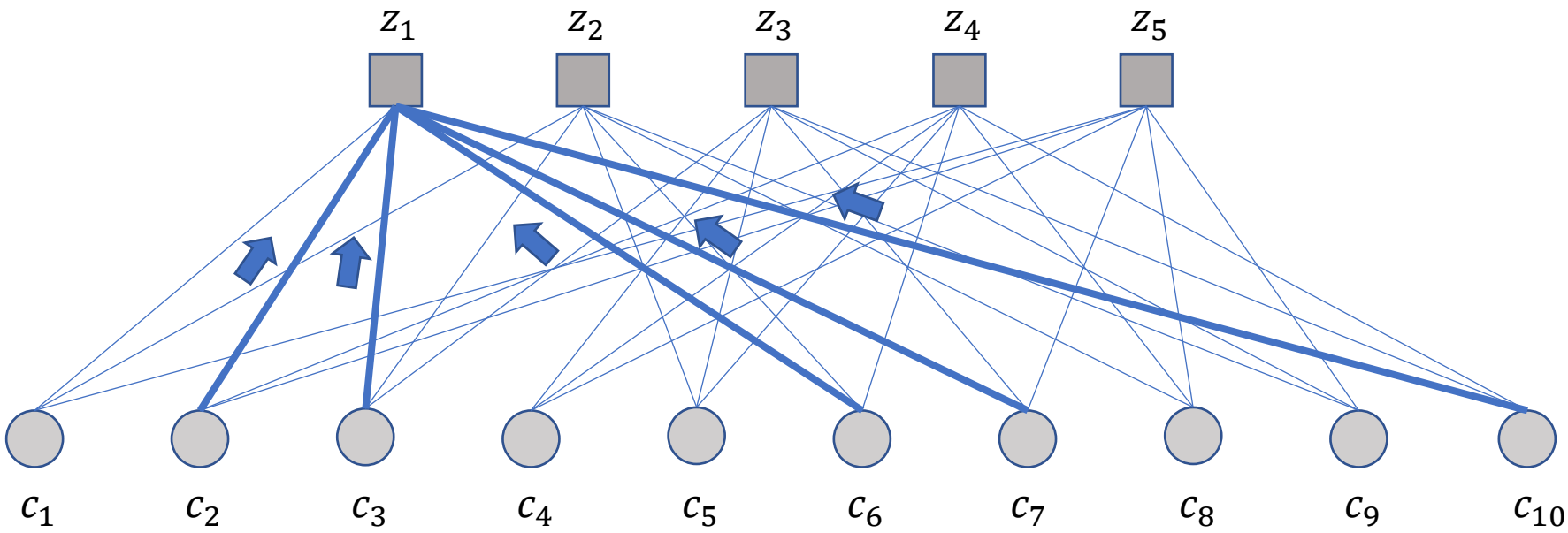
$$\mathbf{q}_{mn}^{(1)} = \begin{pmatrix} 0.22 & 0.16 & 0.19 & 0 & 0 & 0.87 & 0.18 & 0 & 0 & 0.76 \\ 0.22 & 0 & 0.19 & 0 & 0.55 & 0.87 & 0 & 0.79 & 0.25 & 0 \\ 0 & 0 & 0.19 & 0.48 & 0.55 & 0 & 0.18 & 0 & 0.25 & 0.76 \\ 0 & 0.16 & 0 & 0.48 & 0.55 & 0.87 & 0 & 0.79 & 0 & 0.76 \\ 0.22 & 0.16 & 0 & 0.48 & 0 & 0 & 0.18 & 0.79 & 0.25 & 0 \end{pmatrix}$$

$$\mathbf{r}_{mn}^{(1)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Iteration 1 – Check Node Update

$$\mathbf{r}_{mn}^{(1)} = \begin{pmatrix} 0.45 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

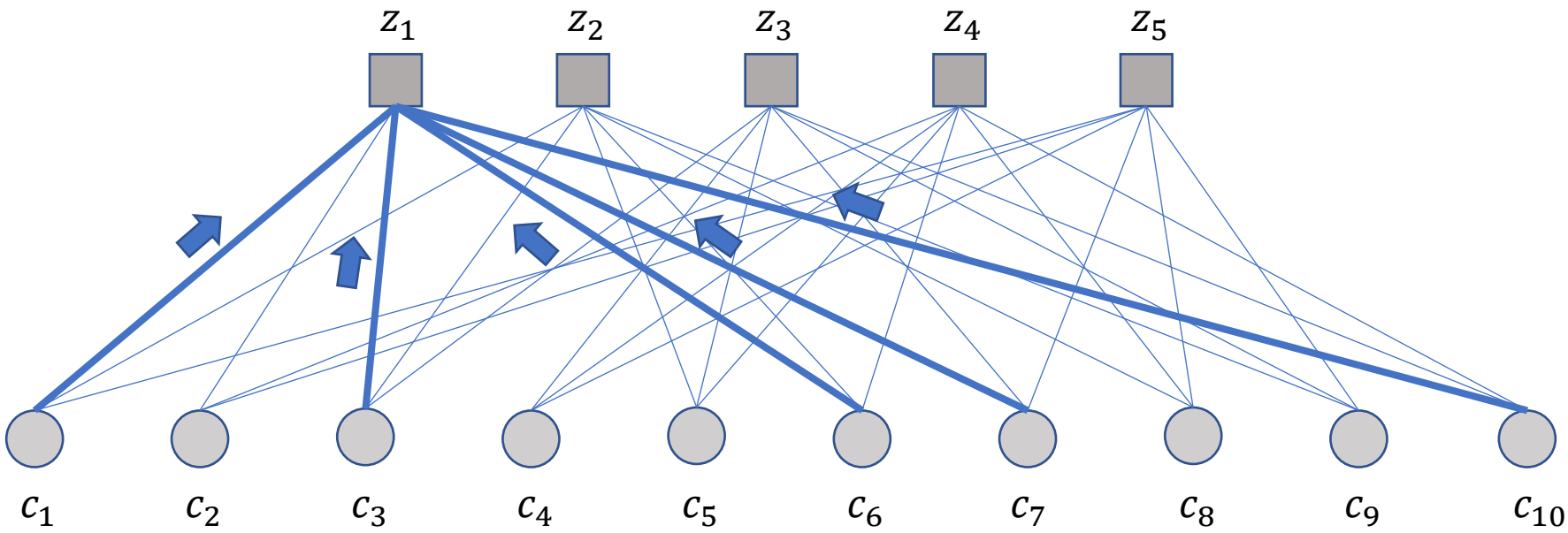
Update $r_{11}(1)$ based on $q_{12}(1), q_{13}(1), q_{16}(1), q_{17}(1)$, and $q_{1,10}(1)$



Iteration 1 – Check Node Update

$$\mathbf{r}_{mn}^{(1)} = \begin{pmatrix} 0.45 & 0.46 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

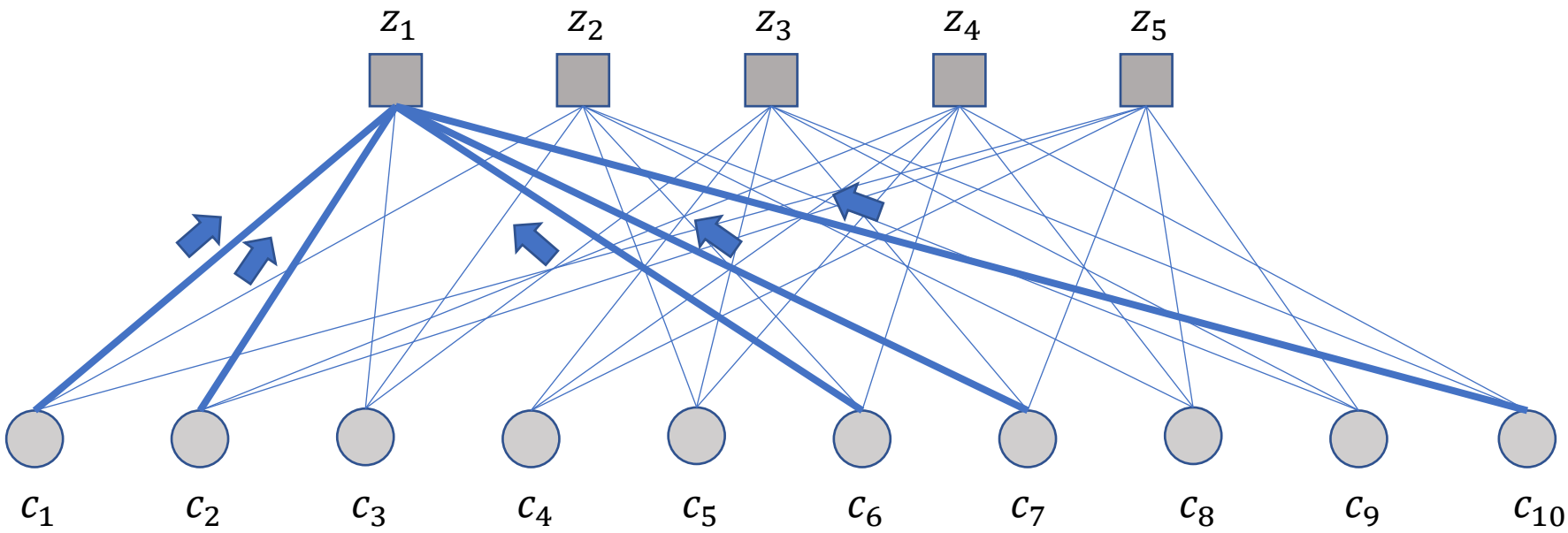
Update $r_{12}(1)$ based on $q_{11}(1), q_{13}(1), q_{16}(1), q_{17}(1)$, and $q_{1,10}(1)$



Iteration 1 – Check Node Update

$$\mathbf{r}_{mn}^{(1)} = \begin{pmatrix} 0.45 & 0.46 & 0.45 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

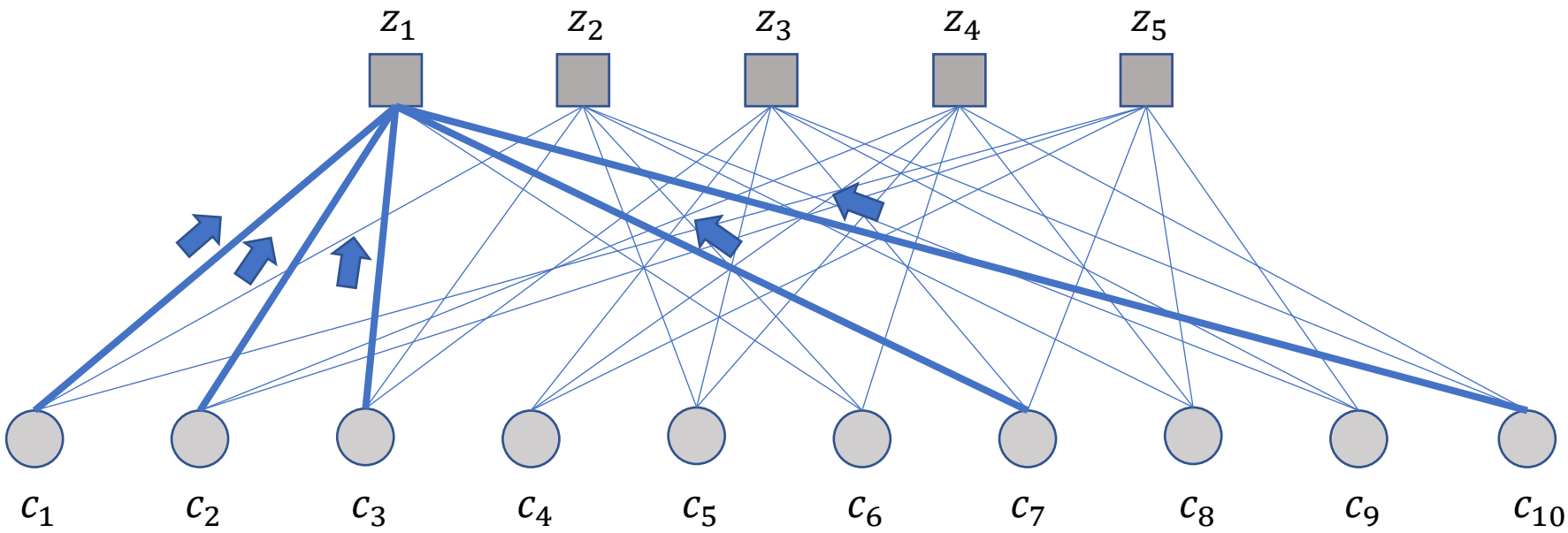
Update $r_{13}(1)$ based on $q_{11}(1), q_{12}(1), q_{16}(1), q_{17}(1)$, and $q_{1,10}(1)$



Iteration 1 – Check Node Update

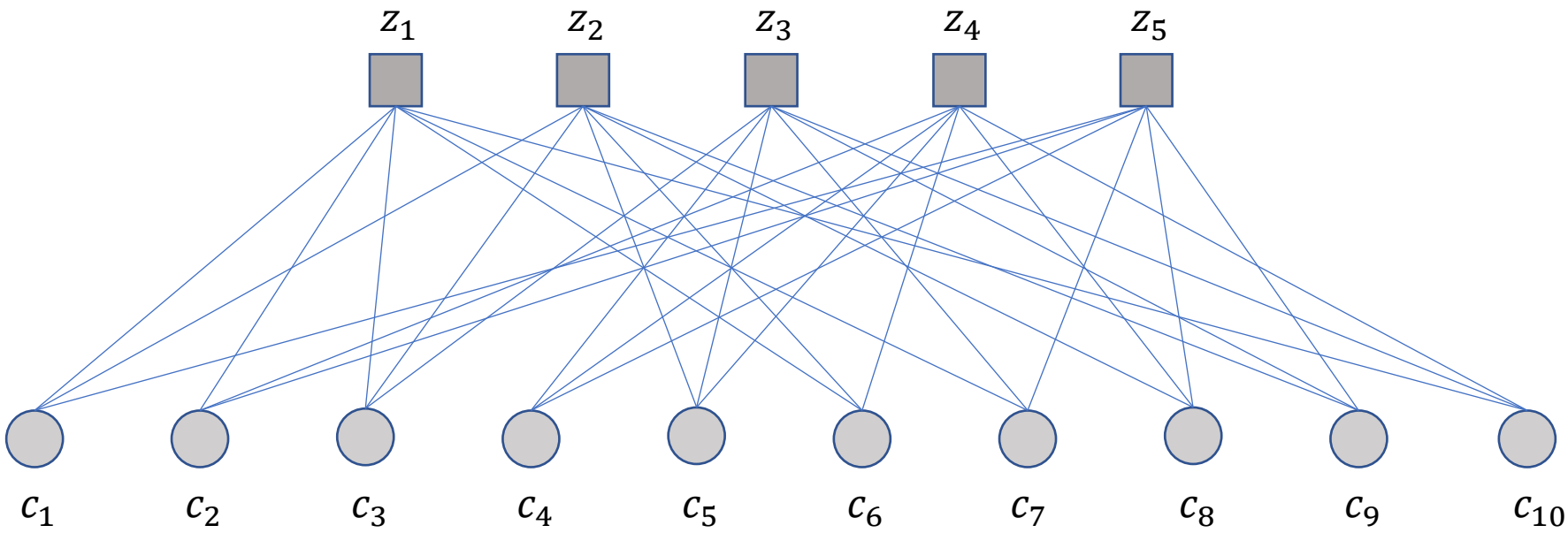
$$\mathbf{r}_{mn}^{(1)} = \begin{pmatrix} 0.45 & 0.46 & 0.45 & 0 & 0 & 0.54 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The procedure continues until all the check nodes are updated.



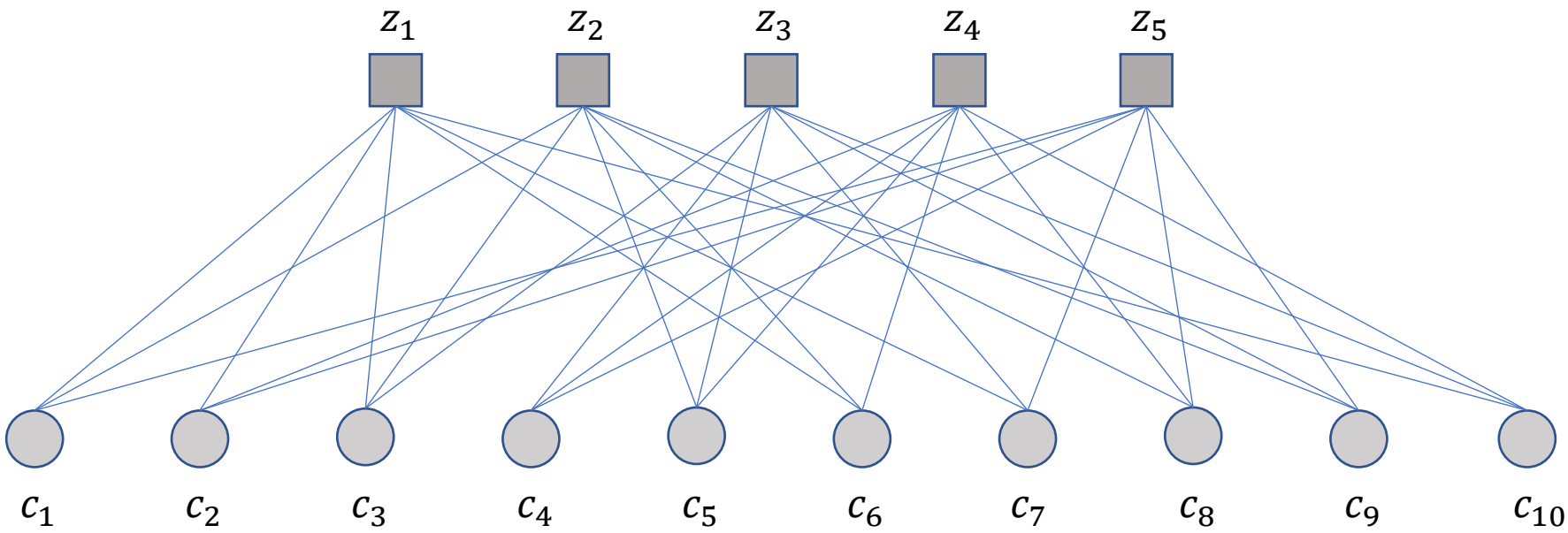
Iteration 1 – Check Node Update Finishes

$$\mathbf{r}_{mn}^{(1)} = \begin{pmatrix} 0.45 & 0.46 & 0.45 & 0 & 0 & 0.54 & 0.45 & 0 & 0 & 0.56 \\ 0.51 & 0 & 0.51 & 0 & 0.46 & 0.49 & 0 & 0.49 & 0.51 & 0 \\ 0 & 0 & 0.5 & 0.49 & 0.5 & 0 & 0.5 & 0 & 0.5 & 0.5 \\ 0 & 0.5 & 0 & 0.49 & 0.5 & 0.5 & 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0.5 & 0 & 0.54 & 0 & 0 & 0.5 & 0.5 & 0.5 & 0 \end{pmatrix}$$



Iteration 1 – Bit Node Update

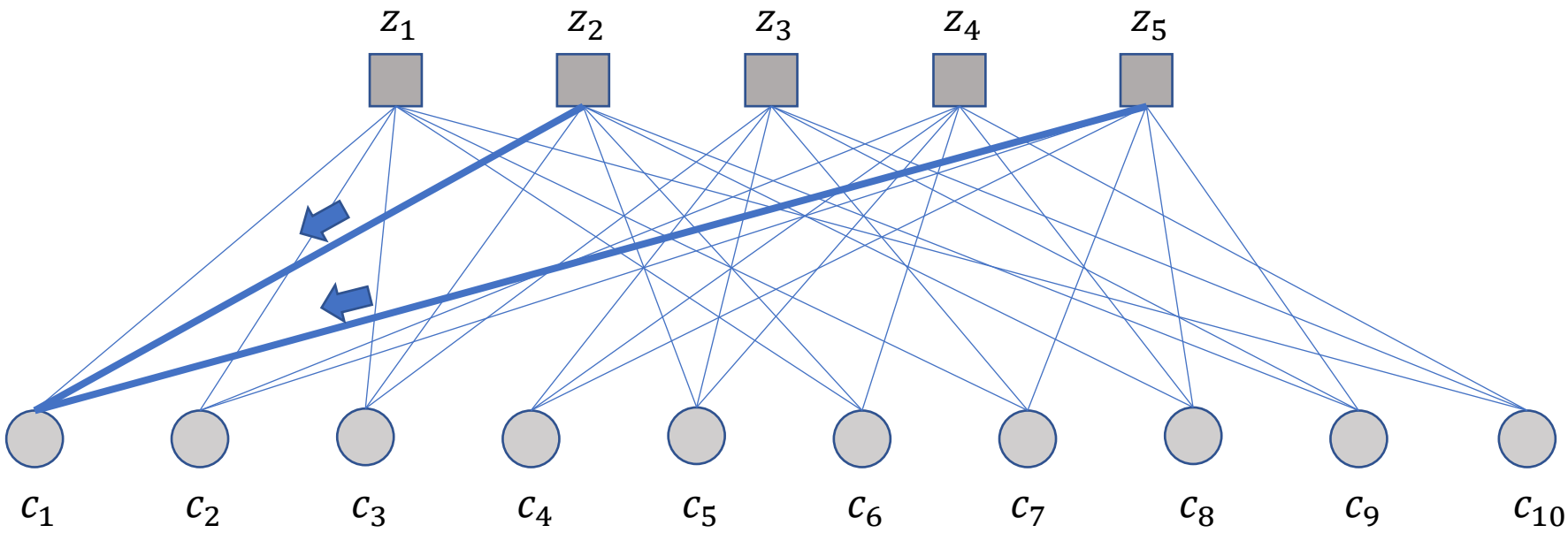
$$\mathbf{q}_{mn}^{(1)} = \begin{pmatrix} 0.22 & 0.16 & 0.19 & 0 & 0 & 0.87 & 0.18 & 0 & 0 & 0.76 \\ 0.22 & 0 & 0.19 & 0 & 0.55 & 0.87 & 0 & 0.79 & 0.25 & 0 \\ 0 & 0 & 0.19 & 0.48 & 0.55 & 0 & 0.18 & 0 & 0.25 & 0.76 \\ 0 & 0.16 & 0 & 0.48 & 0.55 & 0.87 & 0 & 0.79 & 0 & 0.76 \\ 0.22 & 0.16 & 0 & 0.48 & 0 & 0 & 0.18 & 0.79 & 0.25 & 0 \end{pmatrix}$$



Iteration 1 – Bit Node Update

$$\mathbf{q}_{mn}^{(1)} = \begin{pmatrix} 0.23 & 0.16 & 0.19 & 0 & 0 & 0.87 & 0.18 & 0 & 0 & 0.76 \\ 0.22 & 0 & 0.19 & 0 & 0.55 & 0.87 & 0 & 0.79 & 0.25 & 0 \\ 0 & 0 & 0.19 & 0.48 & 0.55 & 0 & 0.18 & 0 & 0.25 & 0.76 \\ 0 & 0.16 & 0 & 0.48 & 0.55 & 0.87 & 0 & 0.79 & 0 & 0.76 \\ 0.22 & 0.16 & 0 & 0.48 & 0 & 0 & 0.18 & 0.79 & 0.25 & 0 \end{pmatrix}$$

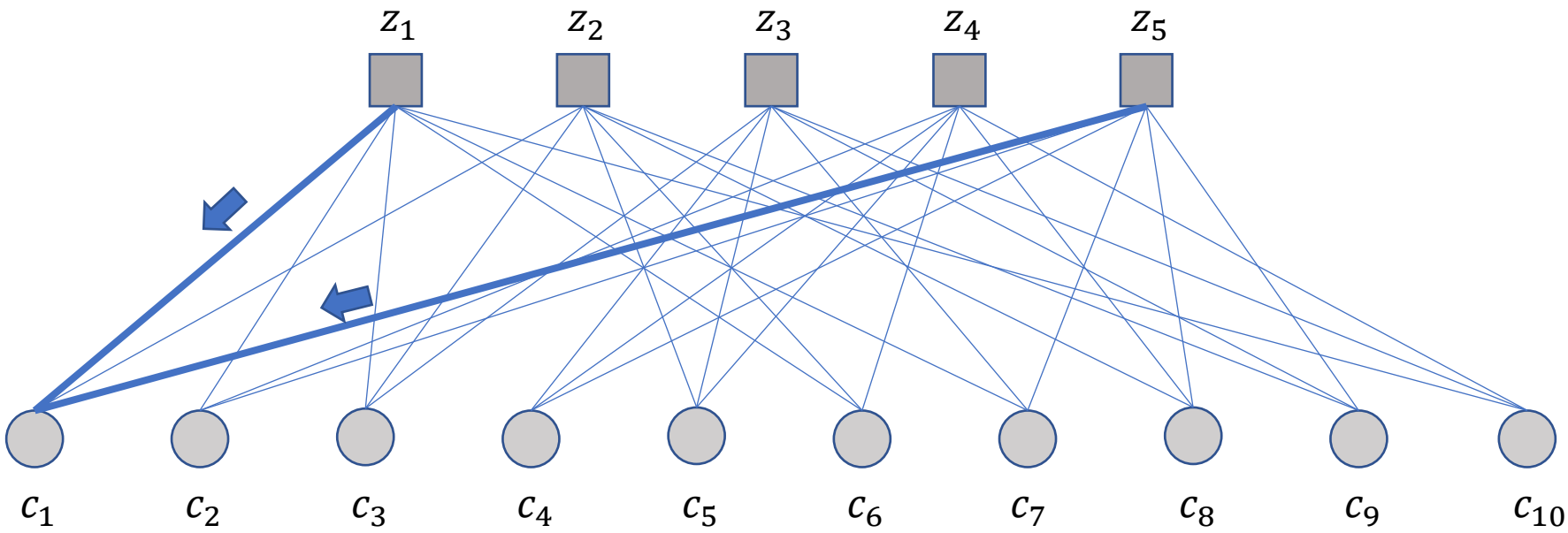
Update $q_{11}(1)$ based on $r_{21}(1), r_{51}(1)$.



Iteration 1 – Bit Node Update

$$\mathbf{q}_{mn}^{(1)} = \begin{pmatrix} 0.23 & 0.16 & 0.19 & 0 & 0 & 0.87 & 0.18 & 0 & 0 & 0.76 \\ 0.19 & 0 & 0.19 & 0 & 0.55 & 0.87 & 0 & 0.79 & 0.25 & 0 \\ 0 & 0 & 0.19 & 0.48 & 0.55 & 0 & 0.18 & 0 & 0.25 & 0.76 \\ 0 & 0.16 & 0 & 0.48 & 0.55 & 0.87 & 0 & 0.79 & 0 & 0.76 \\ 0.22 & 0.16 & 0 & 0.48 & 0 & 0 & 0.18 & 0.79 & 0.25 & 0 \end{pmatrix}$$

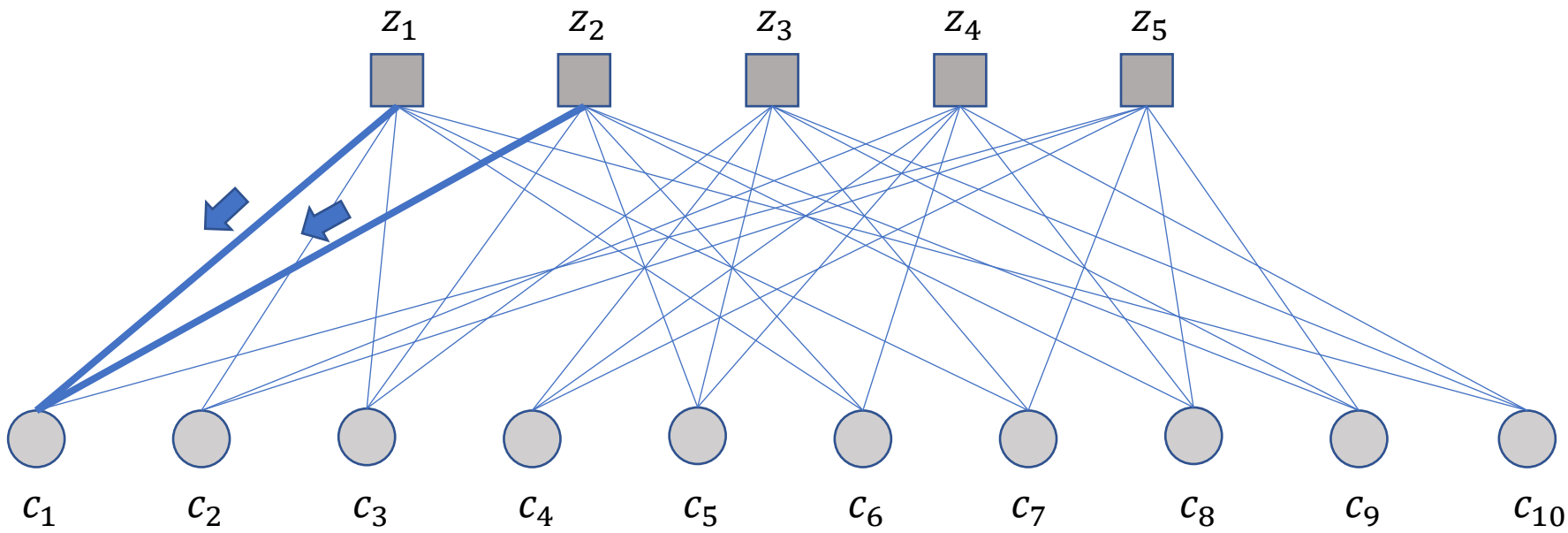
Update $q_{21}(1)$ based on $r_{11}(1), r_{51}(1)$.



Iteration 1 – Bit Node Update

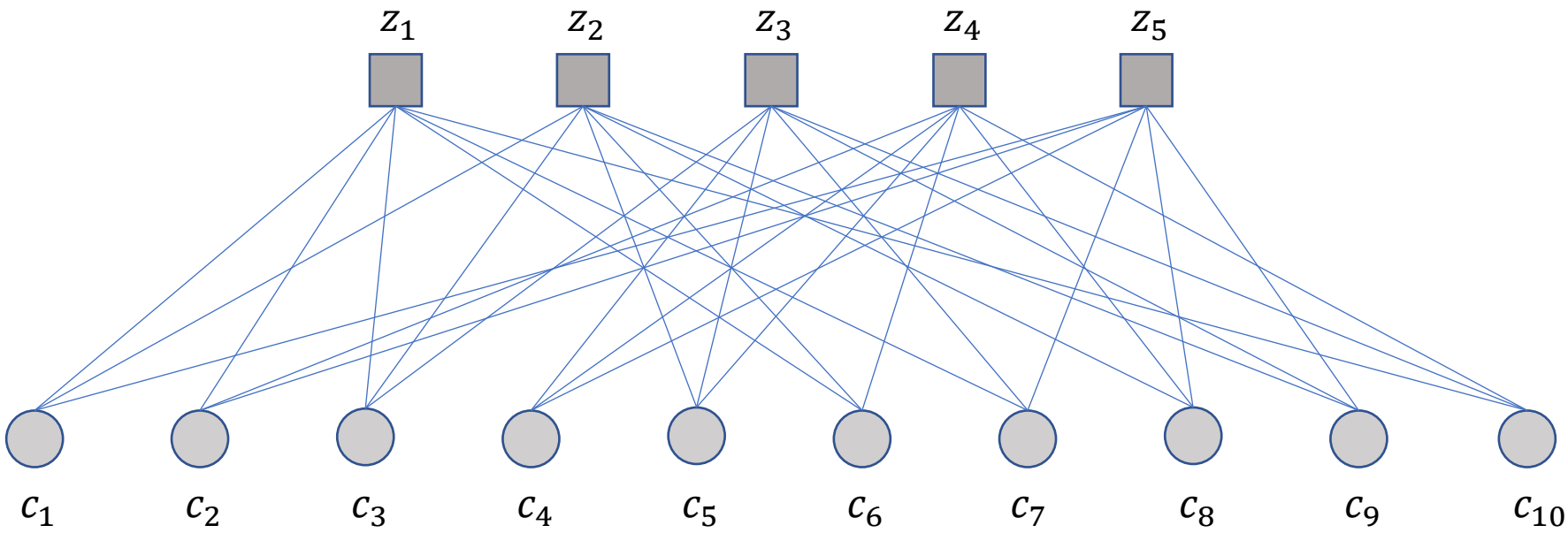
$$\mathbf{q}_{mn}^{(1)} = \begin{pmatrix} 0.23 & 0.16 & 0.19 & 0 & 0 & 0.87 & 0.18 & 0 & 0 & 0.76 \\ 0.19 & 0 & 0.19 & 0 & 0.55 & 0.87 & 0 & 0.79 & 0.25 & 0 \\ 0 & 0 & 0.19 & 0.48 & 0.55 & 0 & 0.18 & 0 & 0.25 & 0.76 \\ 0 & 0.16 & 0 & 0.48 & 0.55 & 0.87 & 0 & 0.79 & 0 & 0.76 \\ 0.19 & 0.16 & 0 & 0.48 & 0 & 0 & 0.18 & 0.79 & 0.25 & 0 \end{pmatrix}$$

Update $q_{51}(1)$ based on $r_{11}(1), r_{21}(1)$. The procedures finished until all the bit nodes are updated.



Iteration 1 – Bit Node Update Finished

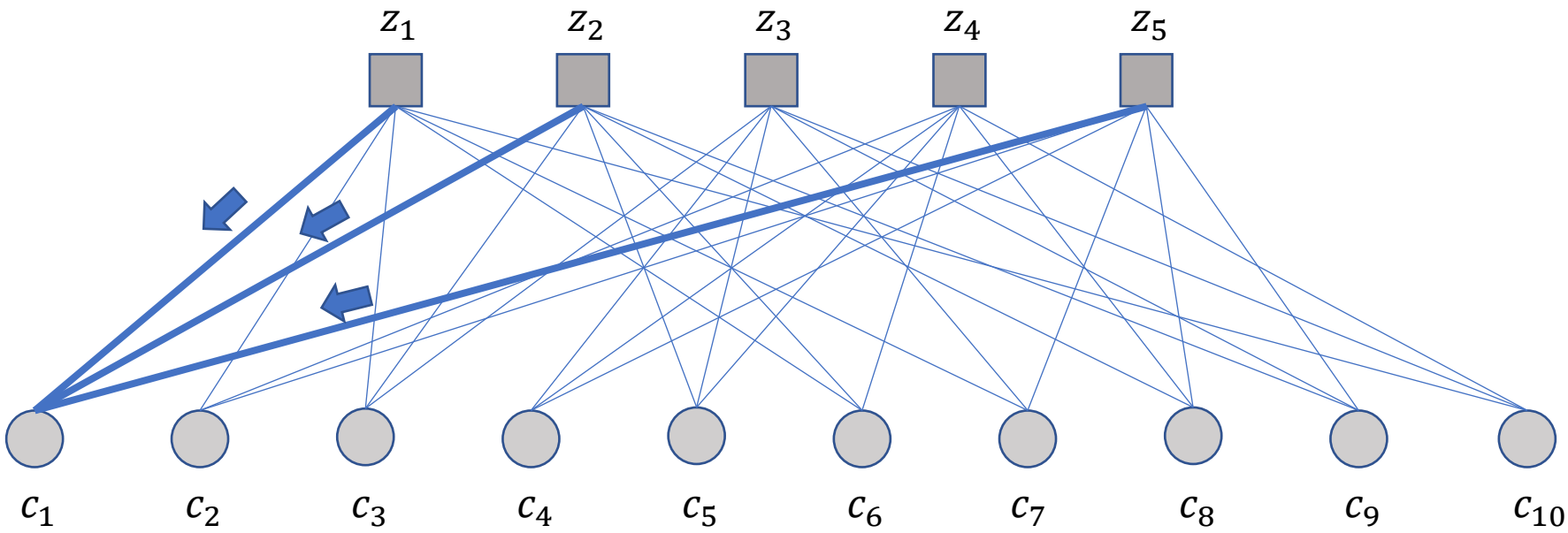
$$\mathbf{q}_{mn}^{(1)} = \begin{pmatrix} 0.23 & 0.16 & 0.19 & 0 & 0 & 0.87 & 0.18 & 0 & 0 & 0.76 \\ 0.19 & 0 & 0.16 & 0 & 0.56 & 0.89 & 0 & 0.79 & 0.25 & 0 \\ 0 & 0 & 0.17 & 0.51 & 0.52 & 0 & 0.16 & 0 & 0.26 & 0.8 \\ 0 & 0.14 & 0 & 0.51 & 0.51 & 0.88 & 0 & 0.78 & 0 & 0.8 \\ 0.19 & 0.14 & 0 & 0.47 & 0 & 0 & 0.15 & 0.79 & 0.26 & 0 \end{pmatrix}$$



Iteration 1 – Compute the $\mathbf{q}_n^{(x)}$

$$\mathbf{r}_{mn}^{(1)} = \begin{pmatrix} 0.45 & 0.46 & 0.45 & 0 & 0 & 0.54 & 0.45 & 0 & 0 & 0.56 \\ 0.51 & 0 & 0.51 & 0 & 0.46 & 0.49 & 0 & 0.49 & 0.51 & 0 \\ 0 & 0 & 0.5 & 0.49 & 0.5 & 0 & 0.5 & 0 & 0.5 & 0.5 \\ 0 & 0.5 & 0 & 0.49 & 0.5 & 0.5 & 0 & 0.5 & 0 & 0.5 \\ 0.5 & 0.5 & 0 & 0.54 & 0 & 0 & 0.5 & 0.5 & 0.5 & 0 \end{pmatrix}$$

Compute $q_1(1)$ based on $r_{11}(1), r_{21}(1)$ and $r_{51}(1)$. The procedures finished until all the bit nodes are updated.



Iteration 1 – Parity Check

- We have

$$\mathbf{q}_n^{(x)} = \begin{pmatrix} 0.19 & 0.14 & 0.17 & 0.5 & 0.52 & 0.88 & 0.16 & 0.78 & 0.26 & 0.8 \end{pmatrix}$$

- The estimated codeword is

$$\hat{\mathbf{c}} = \begin{pmatrix} 0 & 0 & 0 & 1 & \underline{1} & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

- The estimated codeword is **failed** the parity check. The procedure is repeated in the next iteration.
- After two more iterations, the estimated codeword is corrected.

Iteration 1:

$$\hat{\mathbf{c}} = \begin{pmatrix} 0 & 0 & 0 & 1 & \underline{1} & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Iteration 2:

$$\hat{\mathbf{c}} = \begin{pmatrix} 0 & 0 & 0 & 1 & \underline{1} & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

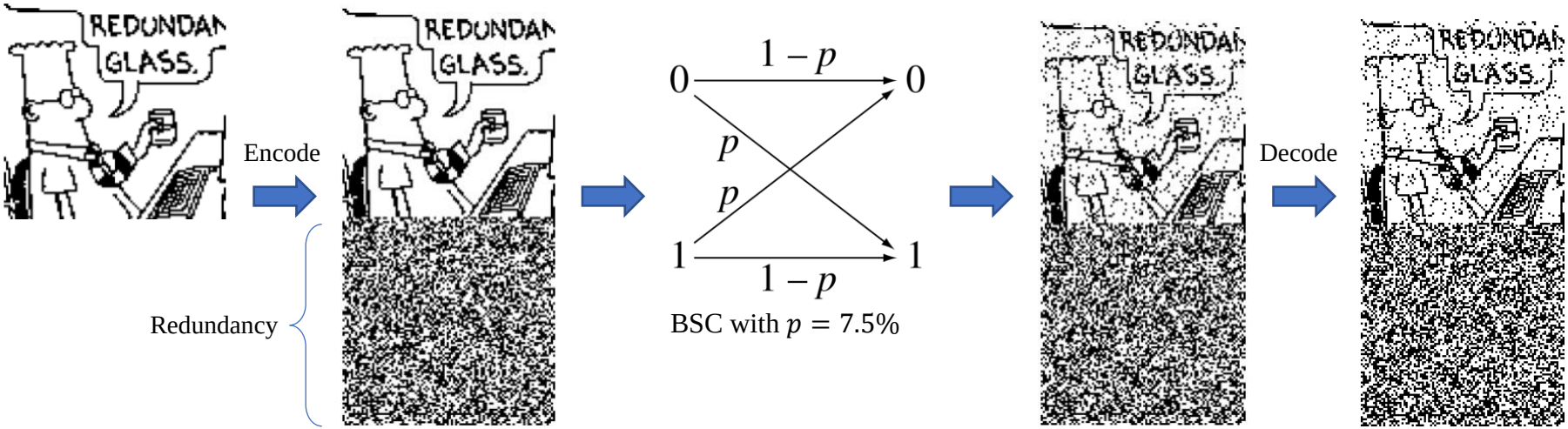
Iteration 3:

$$\hat{\mathbf{c}} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

=> Decoding successfully

Iterative Decoding Algorithms

- **Example**: Consider the decoding process of an LDPC code with $R = 1/2$, 10000×20000 parity check matrix, column weight $w_c = 3$, row weight $w_r = 6$ (*).



(1) Original picture

(1) The picture after encoding with LDPC code

(3) Corrupted picture after the transmission through a noisy channel

(4) Message-passing decode through 13 iterations

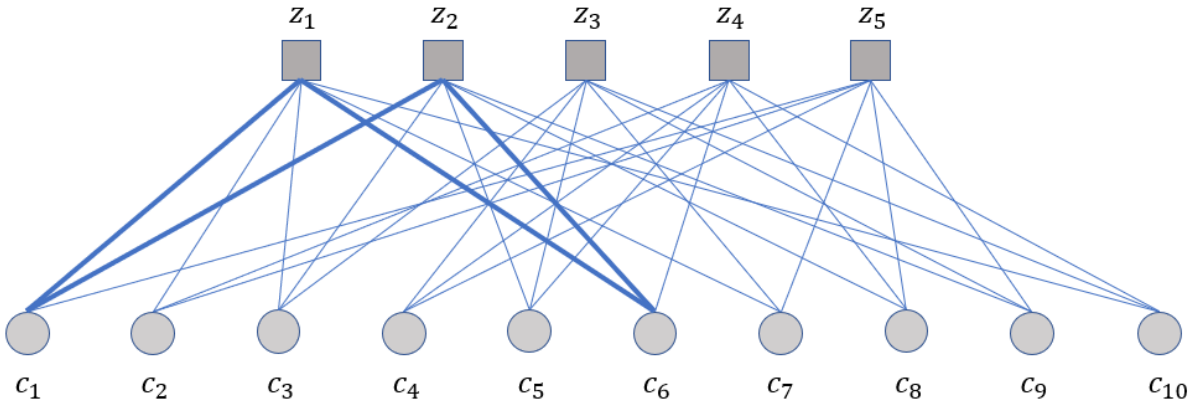
The codeword is gradually corrected after each iteration.

(*) http://www.inference.org.uk/mackay/codes/gifs/

The Limitation of Iterative Decoding

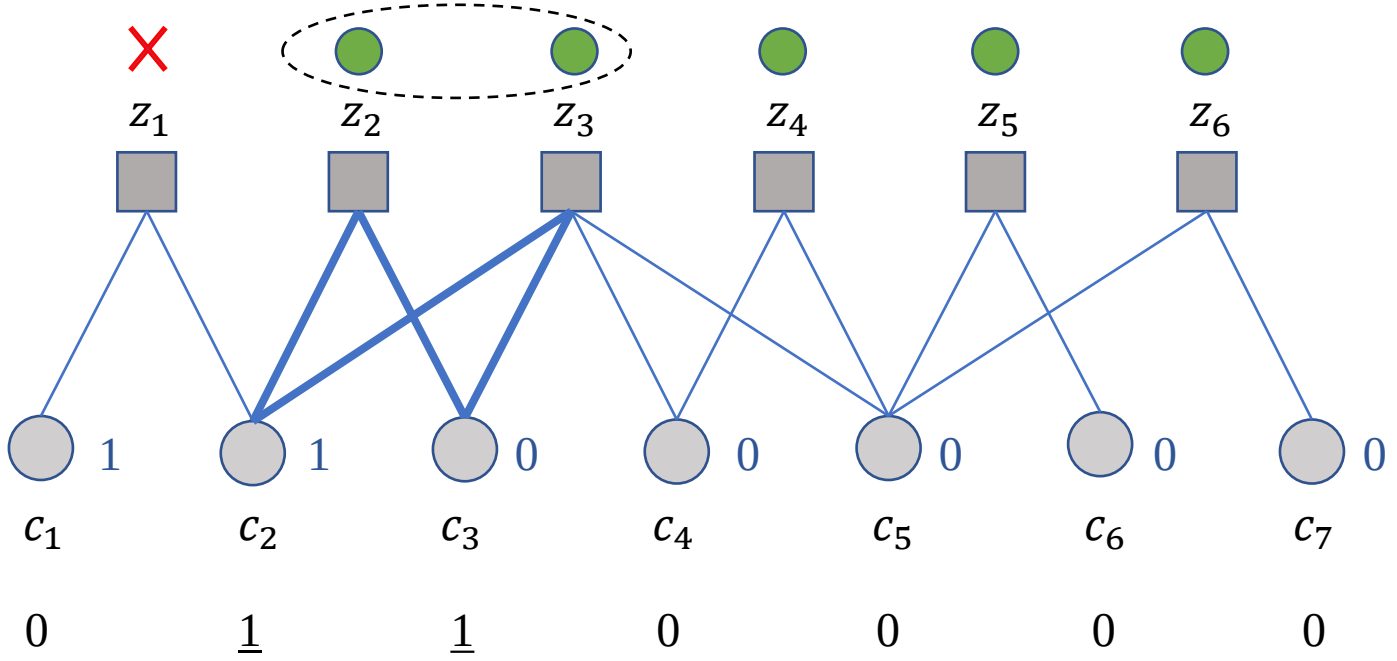
- The performance of iterative decoding algorithms is affected by **the presence of cycles** in the Tanner graph.
  - A **cycle** of length L in a Tanner graph is a path of L edges that closes back on itself.
  - **Girth** is the shortest cycle in the bipartite graph.
- Cycles increase the **dependence of information** being received at each node during message passing.
- Due to the presence of cycles, the results are only **approximate.**
  - The algorithm may take more iterations to decode successfully
  - Or the codeword *can not even be decoded successfully*

It is important to reduce the number of cycles in the parity check matrix.



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

The Limitation of Iterative Decoding: Example



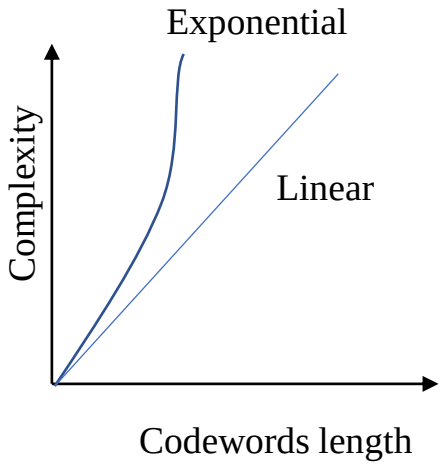
Why LDPC is good?

- According to **Shannon's theorem of noise-channel coding,** for a given rate $R \leq C$, there exists an ECC of length N such that the reliable transmission over a noisy channel can be obtained as $\boldsymbol{N}$ **increases.**
  - E.g., Random codes proposed by Shannon, convolutional codes
- However, the decoding complexity of these codes grows **exponentially** with the codewords length N.
- For LDPC codes, as fixed value of w_c and w_r, the complexity of iterative decoding grows **linearly** when N increases.

The spareness of LDPC codes contributes to the **good performance** and **linear decoding complexity**.

The comparison of Turbo, LDPC and polar decoders implemented on the application-specific integrated circuit (ASIC) drawn from over 100 papers [1]

Parameter	Turbo decoder	LDPC Decoder	Polar Decoder
Computational complexity	Higher for most coding rates. Low at low coding rates	Lower for most coding rates. High at low coding rates	Lower for most coding rates
Information throughput	Low. Enhanced by fully-parallel structure	High for high coding rates and parallel structure	High for pipeline structure
Area efficiency	Low for most coding rate. High at low coding rates.	High for most coding rates. Low at low coding rates.	High for most coding rates.
Energy efficiency	Low for most coding rates. High at low coding rates.	High for most coding rate. Low at low coding rates.	High for most coding rates.
Error correction performance	Similar at long block lengths.	Similar at long block lengths.	Superior for short block lengths.
Flexibility	High flexibility	Partial flexibility	Very high potential



[1] S. Shao et al., "Survey of turbo, LDPC, and polar decoder ASIC implementations," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2309–2333, 3rd Quart., 2019.

Outline

I. Error Correction Code

II. Low-density Parity-check Codes
   1. Introduction
   2. Hard-Decision Bit Flipping
   3. Messaging-Passing Algorithm

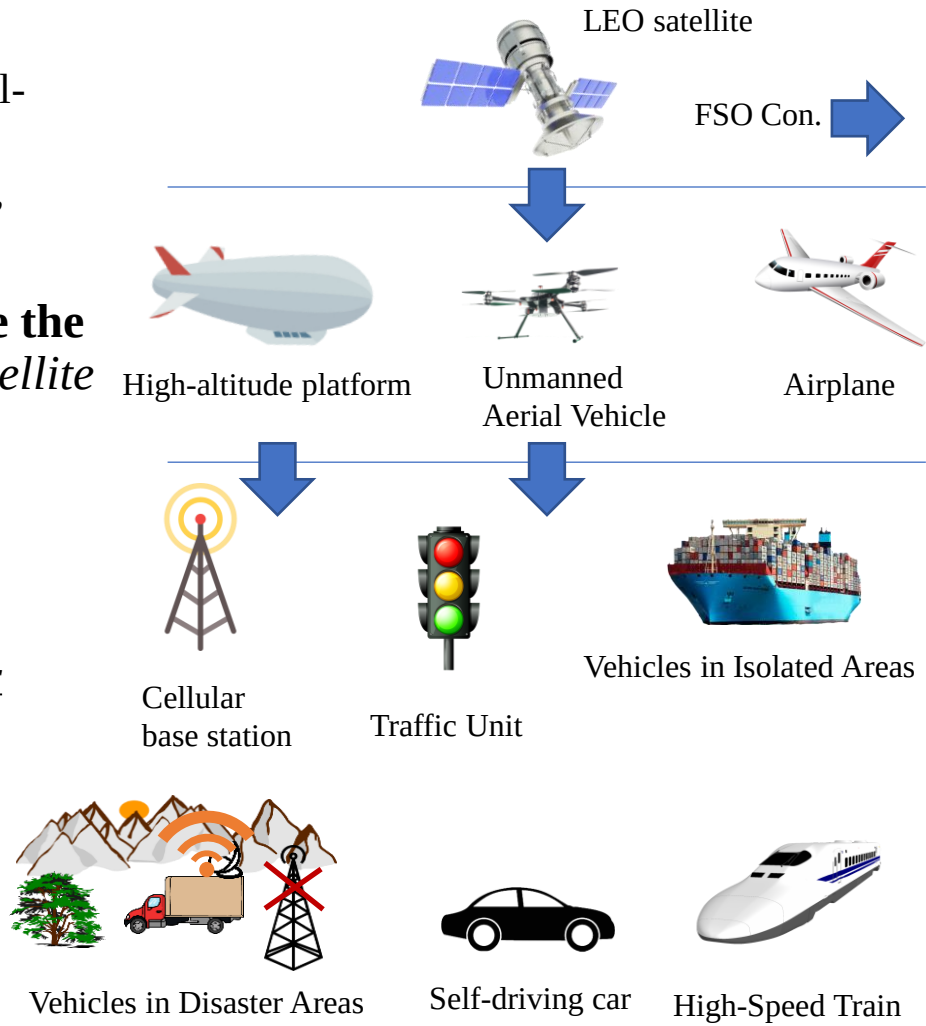
III. Future Direction

Optical Satellite-Assisted Internet of Vehicles

- **Internet of Vehicles (IoVs)** is defined as a network of users, vehicles, and network infrastructure to connect and exchange data over the Internet
- **Applications**
  - **Safety:** emergency call, speed control,…
  - **Navigation:** traffic congestion control, real-time information, parking helper,…
  - **Business:** high-speed Internet for vehicles, entertainment,…
- To **enable more applications and increase the quality of service for the IoVs,** *optical satellite systems are proposed*.
- **Optical Satellite-Assisted IoVs**
  - Uses infrared frequency bands (180-400 Thz) to transmit data in free space

  ⇒ *Offer extremely high data rate (~ Gbps or even Tbps)*
  - Use low-earth orbit (LEO) satellites

  ⇒ *Offer global coverage*
- **Challenging issue:** ***Uncertainty channel***



Research Direction (1)

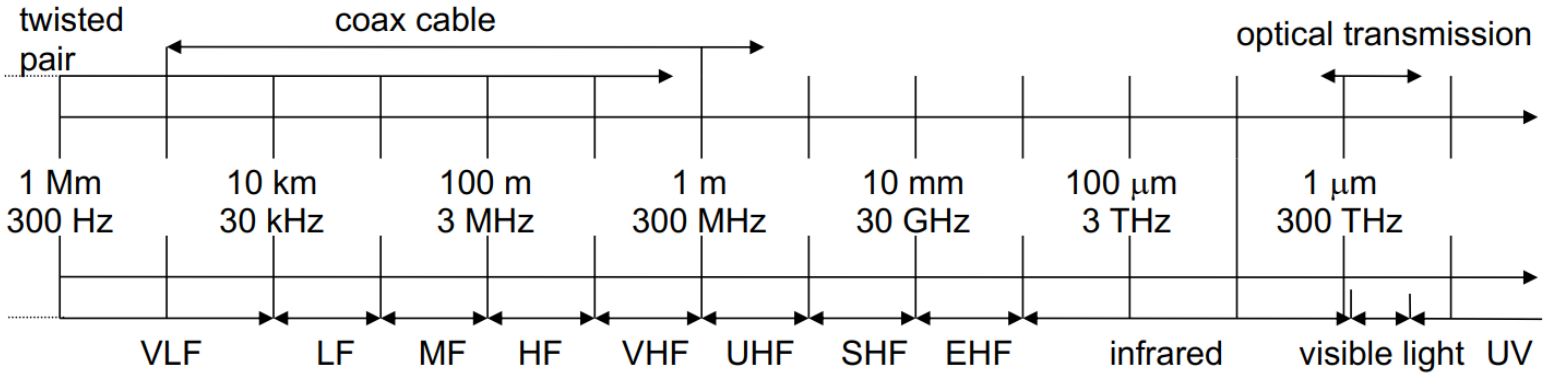
❑ *Why should we have new designs of error-control methods for optical satellite system?*

Due to the different frequency bands, the optical satellite communication is *different from the radio frequency (RF)-based satellite communication.*

- **E.g.,** the data rate and coherent time of the optical channel are much higher => The usage of burst transmission is much *preferable* => The ability to encode and correct long block codes in a low time manner.
- The optical link is deteriorated significantly by the atmosphere => The methods to cope with the new degrading factor.

⇒ The design of error-control methods for RF-based satellite systems **may not inefficient** for *the optical satellite system.*

It is worth *to reconsidering the new design of error-control methods for optical satellite systems.*



Research Direction (2)

❑ *What is the proposed design for optical satellite-assisted IoVs?*

1. Hybrid ARQ (HARQ) Incremental Redundancy (IR) is taken into account

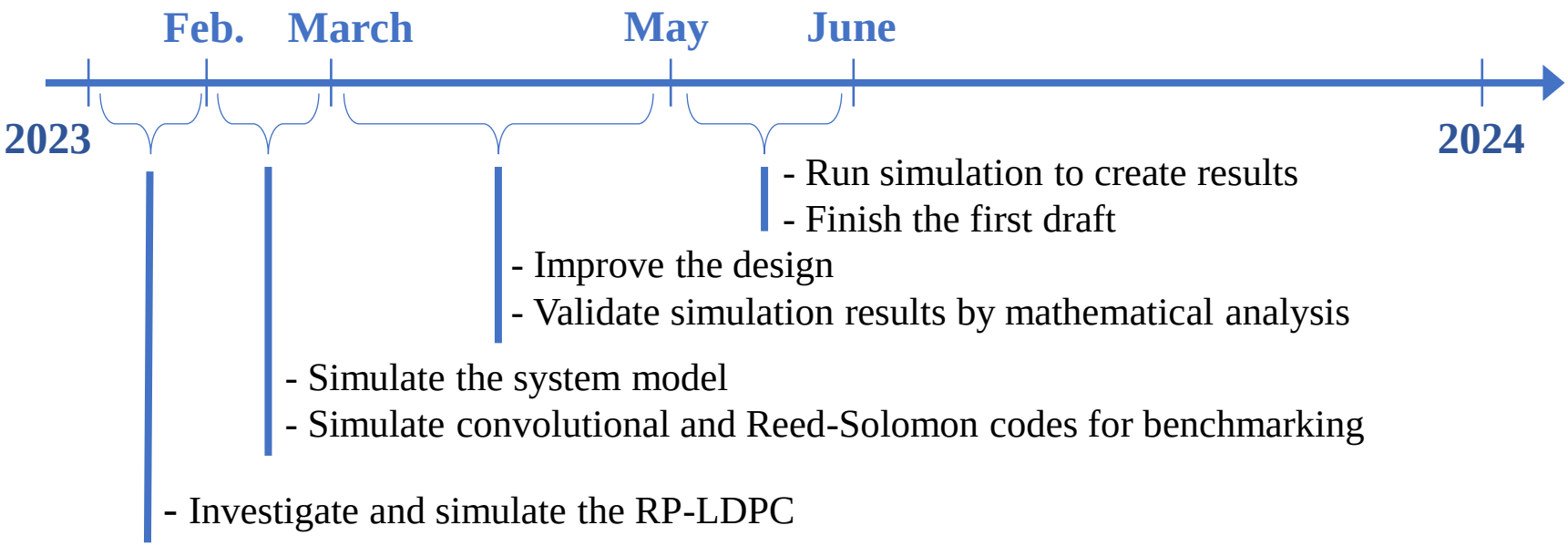
- In **HARQ-IR**, redundancy is transmitted whenever it is necessary and increases after each retransmission.
- Compared to standalone ARQ, **HARQ-IR** reduces the frequency of retransmissions, *especially in long-distance communication.*
- Compared to standalone ECCs, **HARQ-IR** increases the throughput efficiency as *redundancy is transmitted whenever it is necessary.*

2. Rate-compatible Puncturing (RP) LDPC codes are taken into account

- **LDPC codes** have **high performance** and **linear decoding complexity** for very long block code. => *Suitable for high-speed transmission over noisy channels*
- **LDPC decoders implemented on ASICs** show outstanding performance in terms of throughput, size and energy consumption => *Suitable for communications in IoVs*
- **Rate-compatible Puncturing** is the process of creating higher code rates from a lower mother code rate without changing their structure. => *Work with HARQ-IR to increase the performance*

Research Direction: Propose a design of RP-LDPC-based HARQ-IR for the optical satellite-assisted IoVs systems.

Research Plan



Thank you for your attention

The material and examples in this slide are adopted from
[1] *"Error Correction Coding: Mathematical Methods and Algorithms"* by Todd K. Moon
[2] *"Information Theory, Inference and Learning Algorithms"* by David J. C. MacKay
[3] "*Introduction to Low-Density Parity Check Codes"* slide by Brian Kurkoski