Spatially Coupled LDPC Codes: From Theory to Practice

Daniel J. Costello, Jr.

Dept. of Electrical Engineering,
University of Notre Dame

European School of Information Theory
Madrid, Spain, May 8, 2017

The author gratefully acknowledges David Mitchell for help with the preparation of this presentation

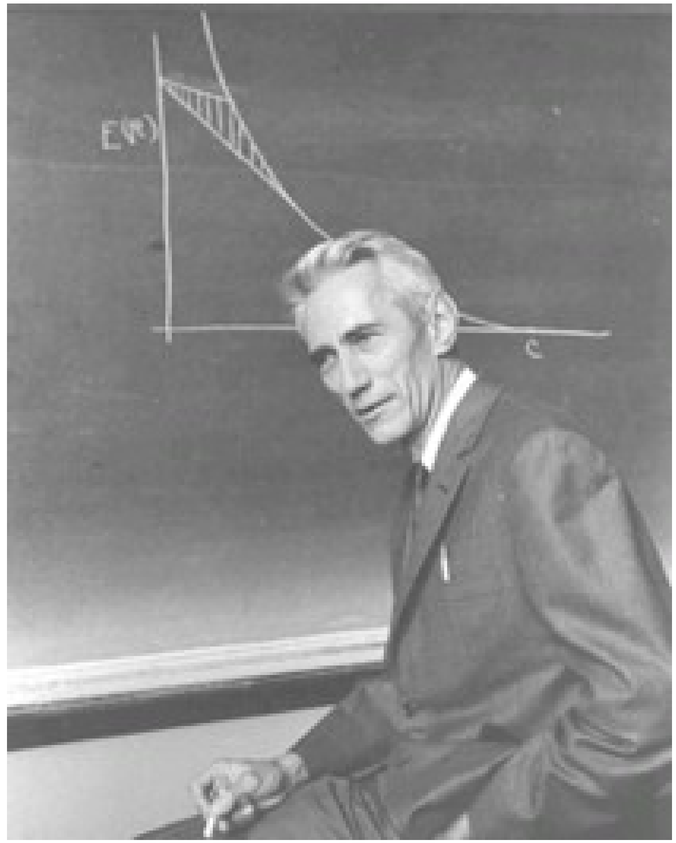
Outline

- **Introduction: From Shannon to Modern Coding Theory**
  - ➡ Channel capacity, structured codes, random codes, LDPC codes
- **LDPC Block Codes**
  - ➡ Parity-check matrix and Tanner graph representations, minimum distance bounds, iterative decoding thresholds, protograph-based constructions
- **Spatially Coupled LDPC Codes**
  - ➡ Protograph representation, edge-spreading construction, termination
  - ➡ Iterative decoding thresholds, threshold saturation, minimum distance
- **Practical Considerations**
  - ➡ Window decoding, performance, latency, and complexity comparisons to LDPC block codes, rate-compatibility, implementation aspects

Outline

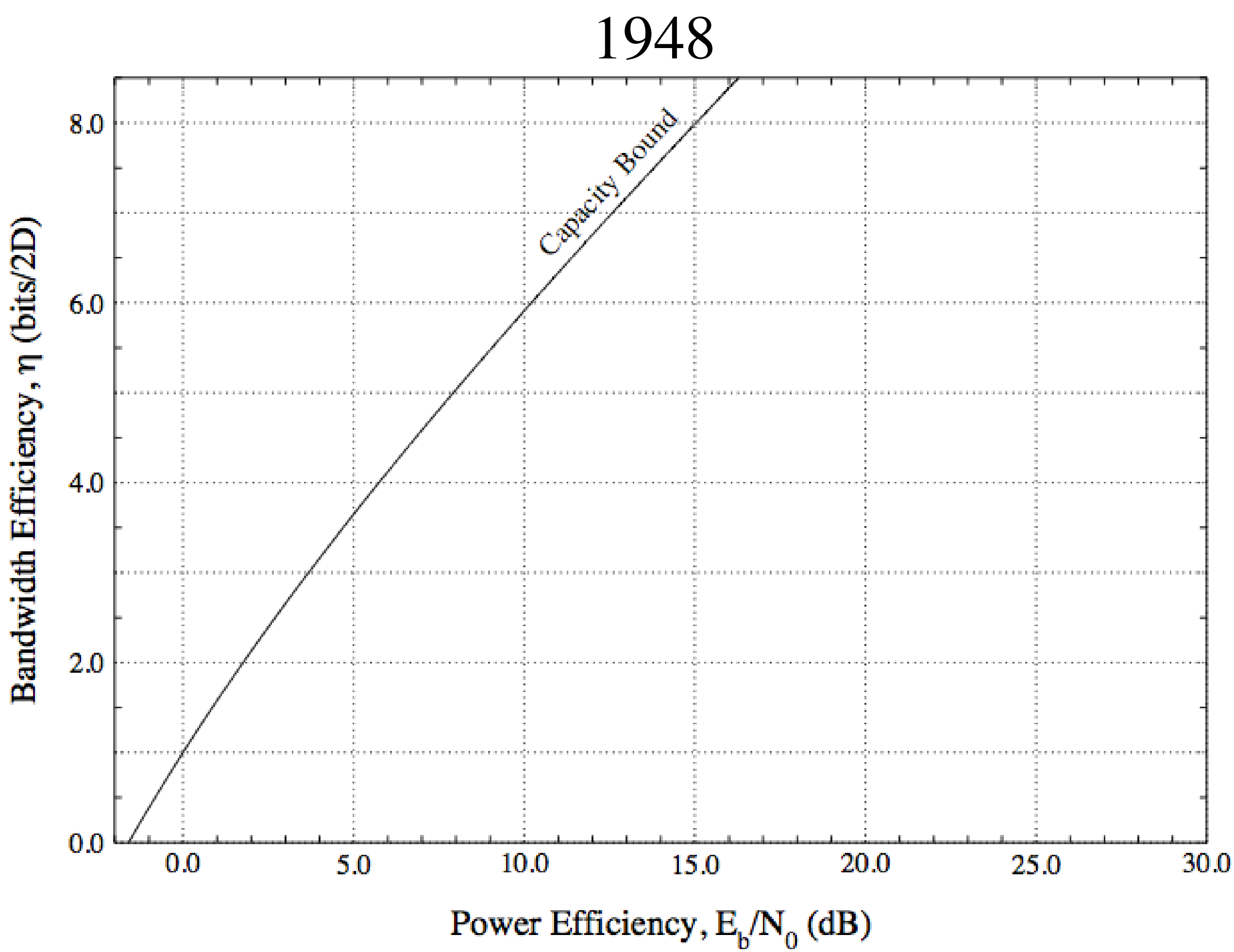
- **Introduction: From Shannon to Modern Coding Theory**
  - Channel capacity, structured codes, random codes, LDPC codes
- **LDPC Block Codes**
  - Parity-check matrix and Tanner graph representations, minimum distance bounds, iterative decoding thresholds, protograph-based constructions
- **Spatially Coupled LDPC Codes**
  - Protograph representation, edge-spreading construction, termination
  - Iterative decoding thresholds, threshold saturation, minimum distance
- **Practical Considerations**
  - Window decoding, performance, latency, and complexity comparisons to LDPC block codes, rate-compatibility, implementation aspects

Shannon's Legacy

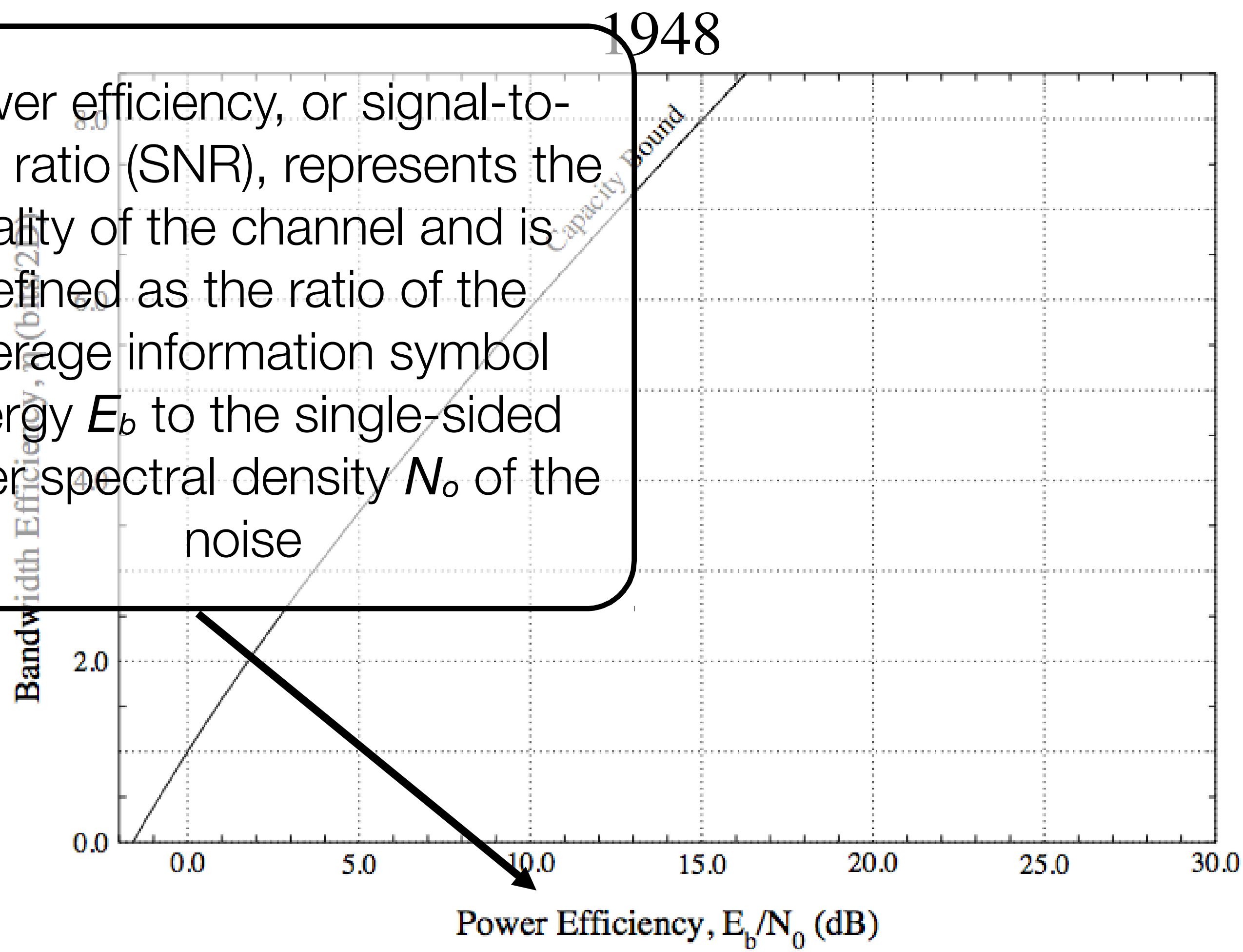


Claude Elwood Shannon
Apr. 30, 1916 – Feb. 24, 2001
Father of **Information Theory**

Channel Capacity (AWGNC)

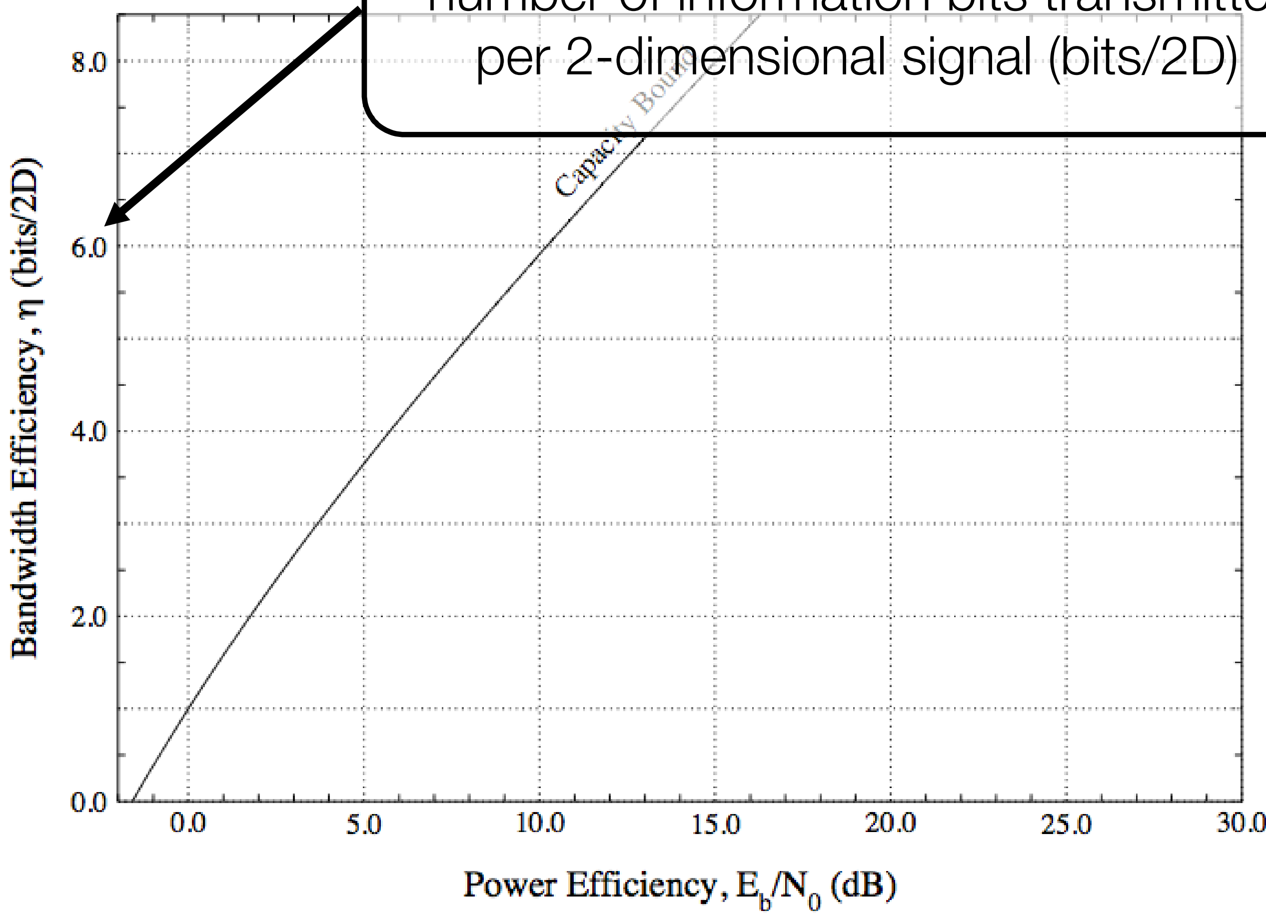


Channel Capacity (AWGNC)



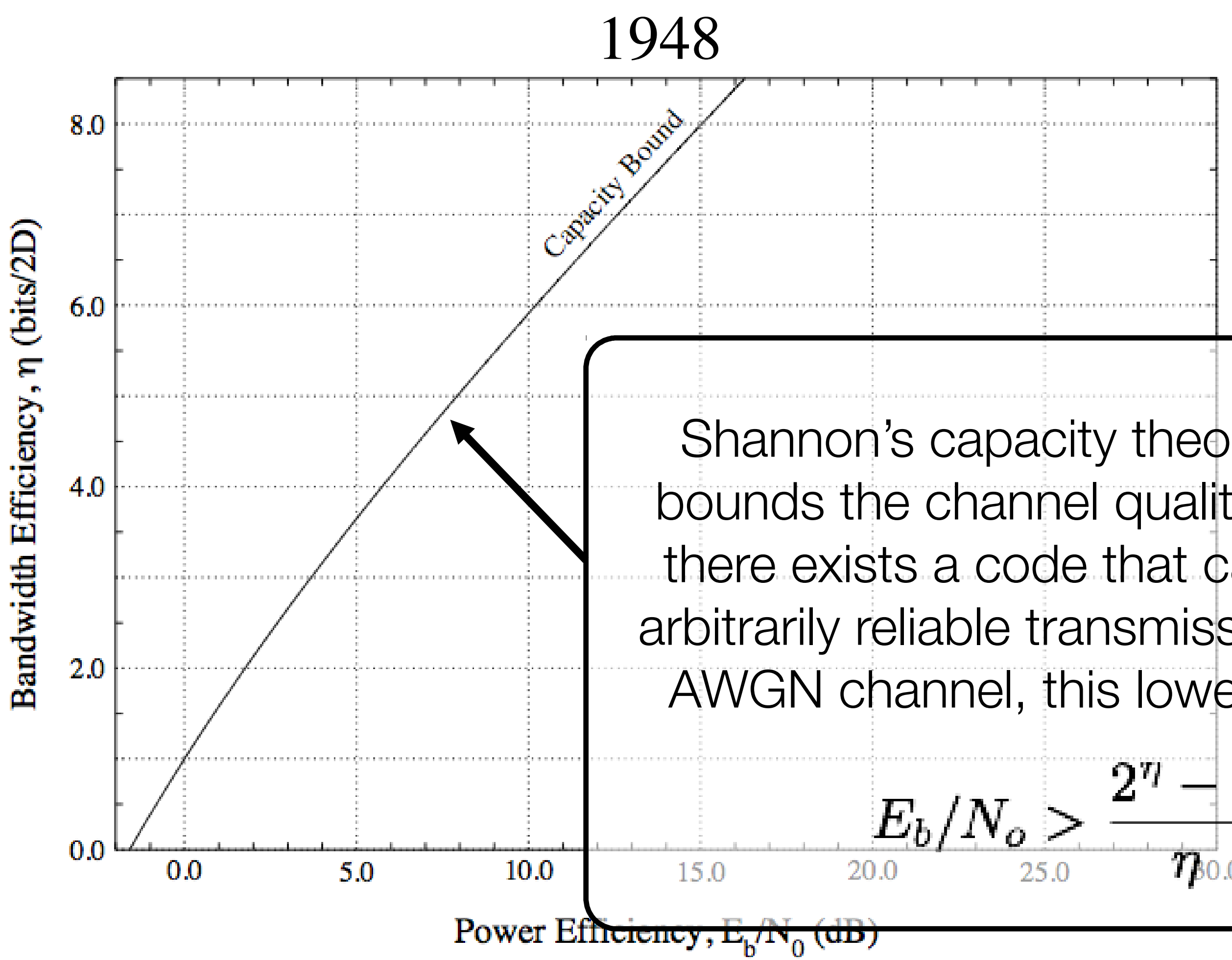
Power efficiency, or signal-to-noise ratio (SNR), represents the quality of the channel and is defined as the ratio of the average information symbol energy E_b to the single-sided power spectral density N_o of the noise

Channel Capacity (AWGNC)



Bandwidth efficiency is normalized to the number of information bits transmitted per 2-dimensional signal (bits/2D)

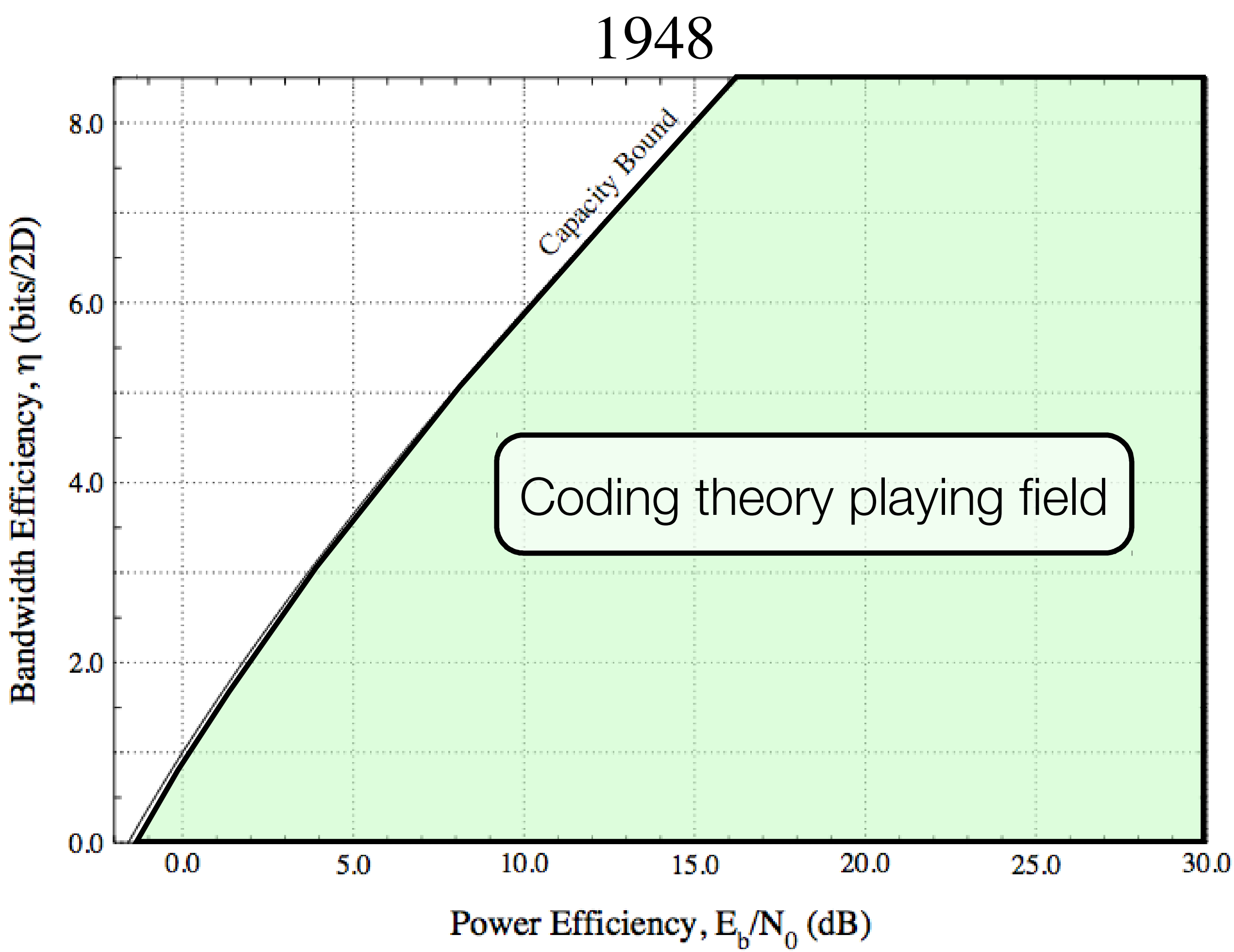
Channel Capacity (AWGNC)



Shannon's capacity theorem lower bounds the channel quality for which there exists a code that can achieve arbitrarily reliable transmission. For the AWGN channel, this lower bound is

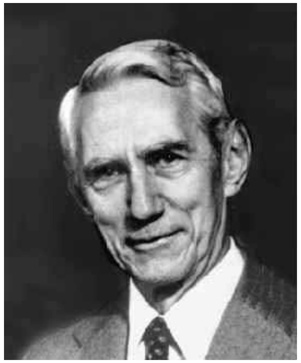
$$E_b/N_o > \frac{2^\eta - 1}{\eta}$$

Channel Capacity (AWGNC)

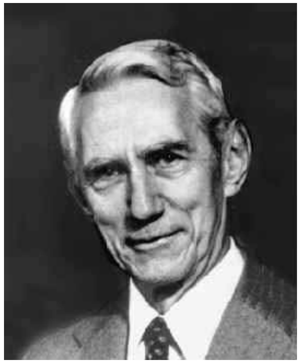


The coding dilemma

- Shannon showed that **random codes** with large block length can **achieve capacity**, but...



The coding dilemma



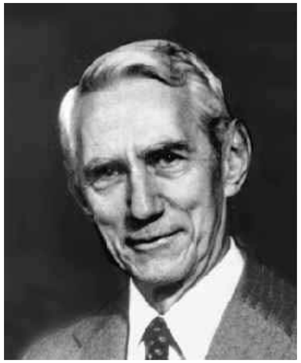
- Shannon showed that **random codes** with large block length can **achieve capacity**, but...

  … code **structure** (algebraic/topological) is required in order to permit decoding with reasonable complexity.

The coding dilemma

- Shannon showed that **random codes** with large block length can **achieve capacity**, but...

  … code **structure** (algebraic/topological) is required in order to permit decoding with reasonable complexity.

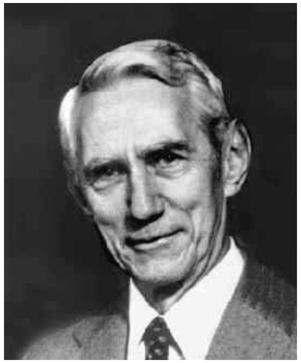


“Almost all codes are good...

The coding dilemma

- Shannon showed that **random codes** with large block length can **achieve capacity**, but...

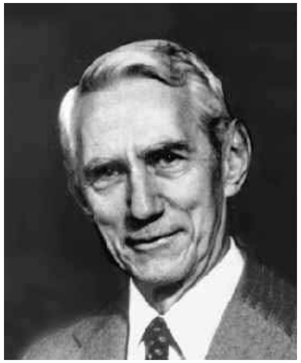
  … code **structure** (algebraic/topological) is required in order to permit decoding with reasonable complexity.



"Almost all codes are good... except those we can think of."

The coding dilemma

UNIVERSITY OF NOTRE DAME



- Shannon showed that **random codes** with large block length can **achieve capacity**, but...

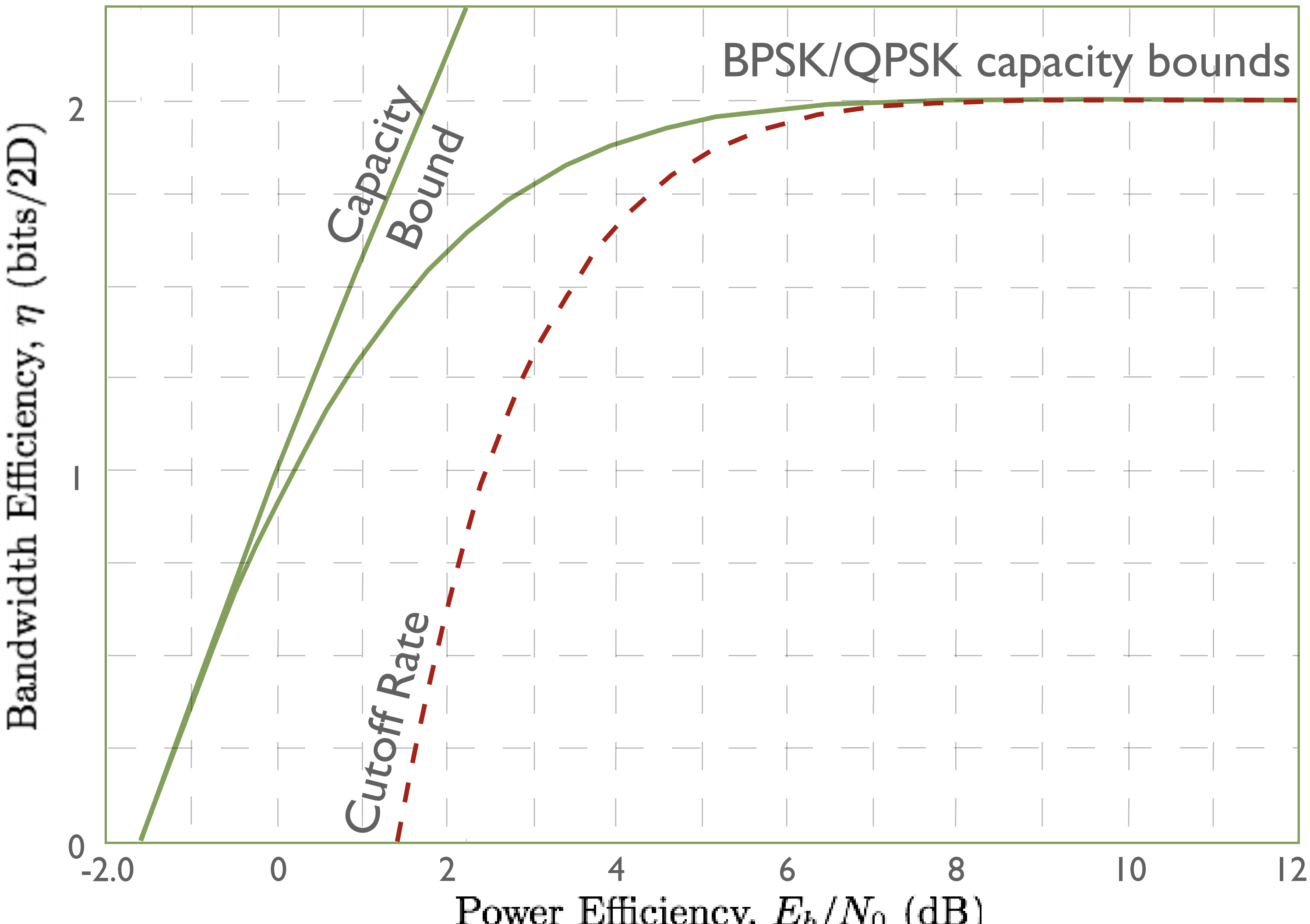
  … code **structure** (algebraic/topological) is required in order to permit decoding with reasonable complexity.

“Almost all codes are good... except those we can think of.”

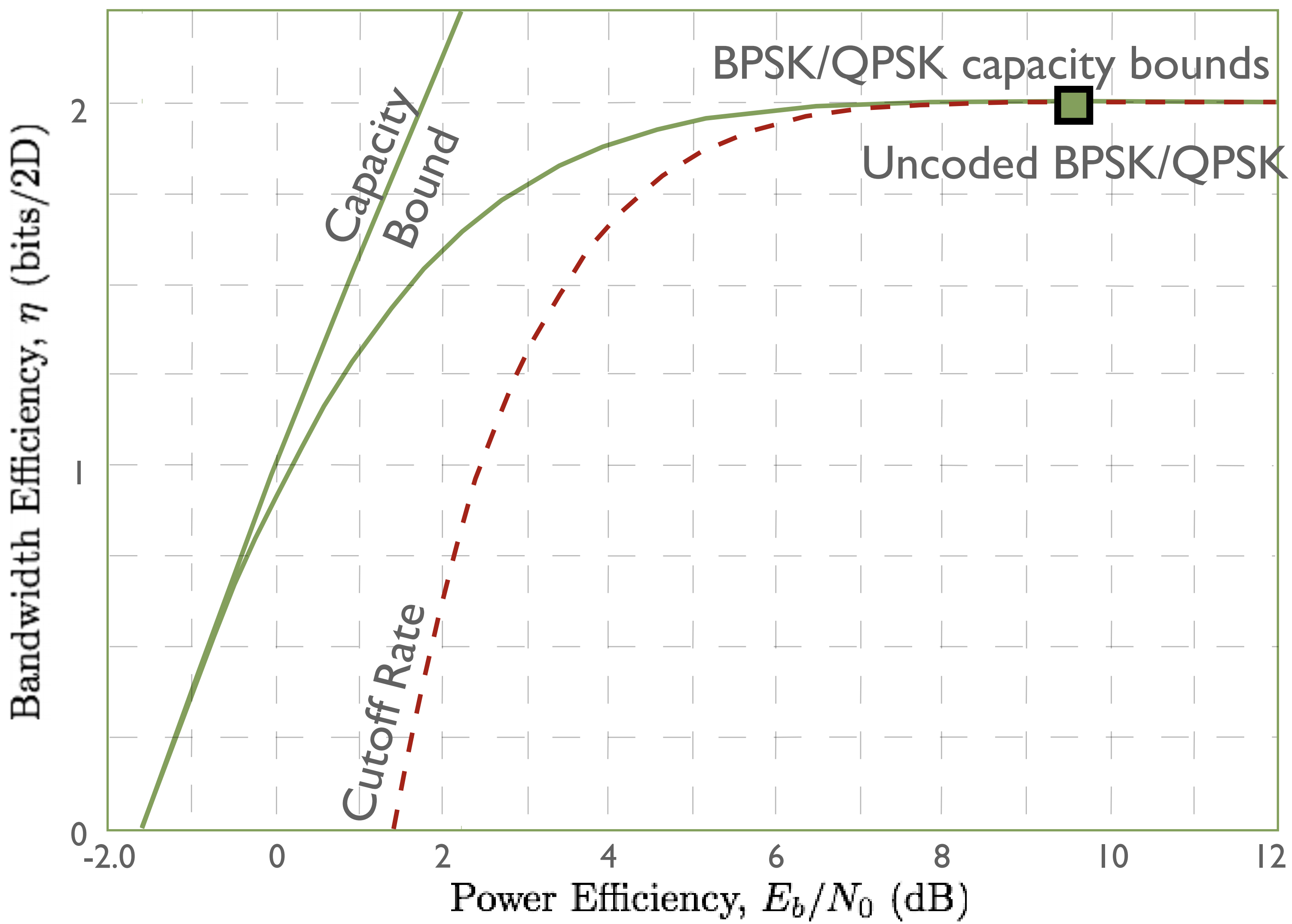
Solution: Construct random-like codes with just enough structure to allow efficient decoding

➡ Modern Coding Theory

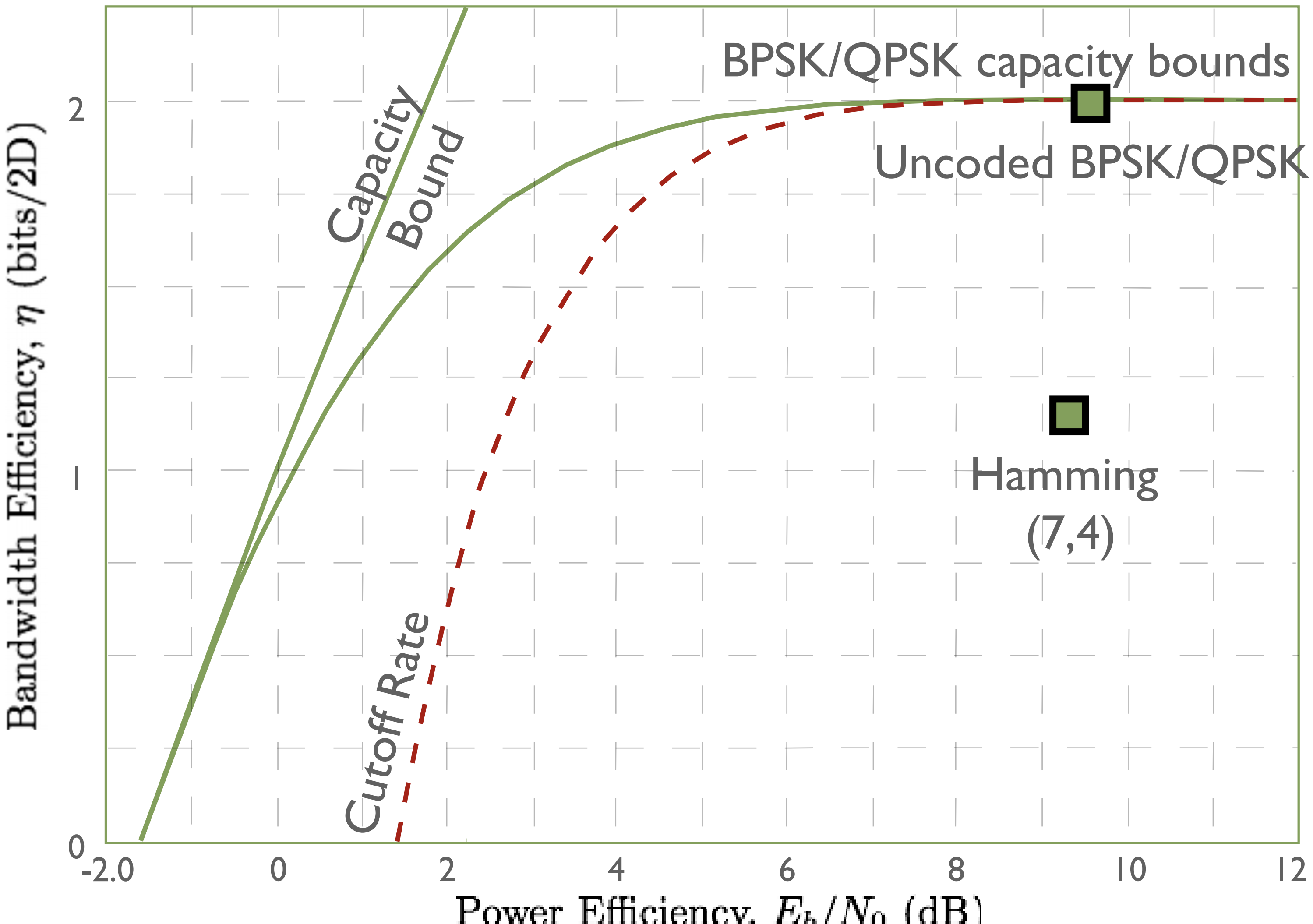
LDPC Codes: motivation (for a target BER 10^{-5})



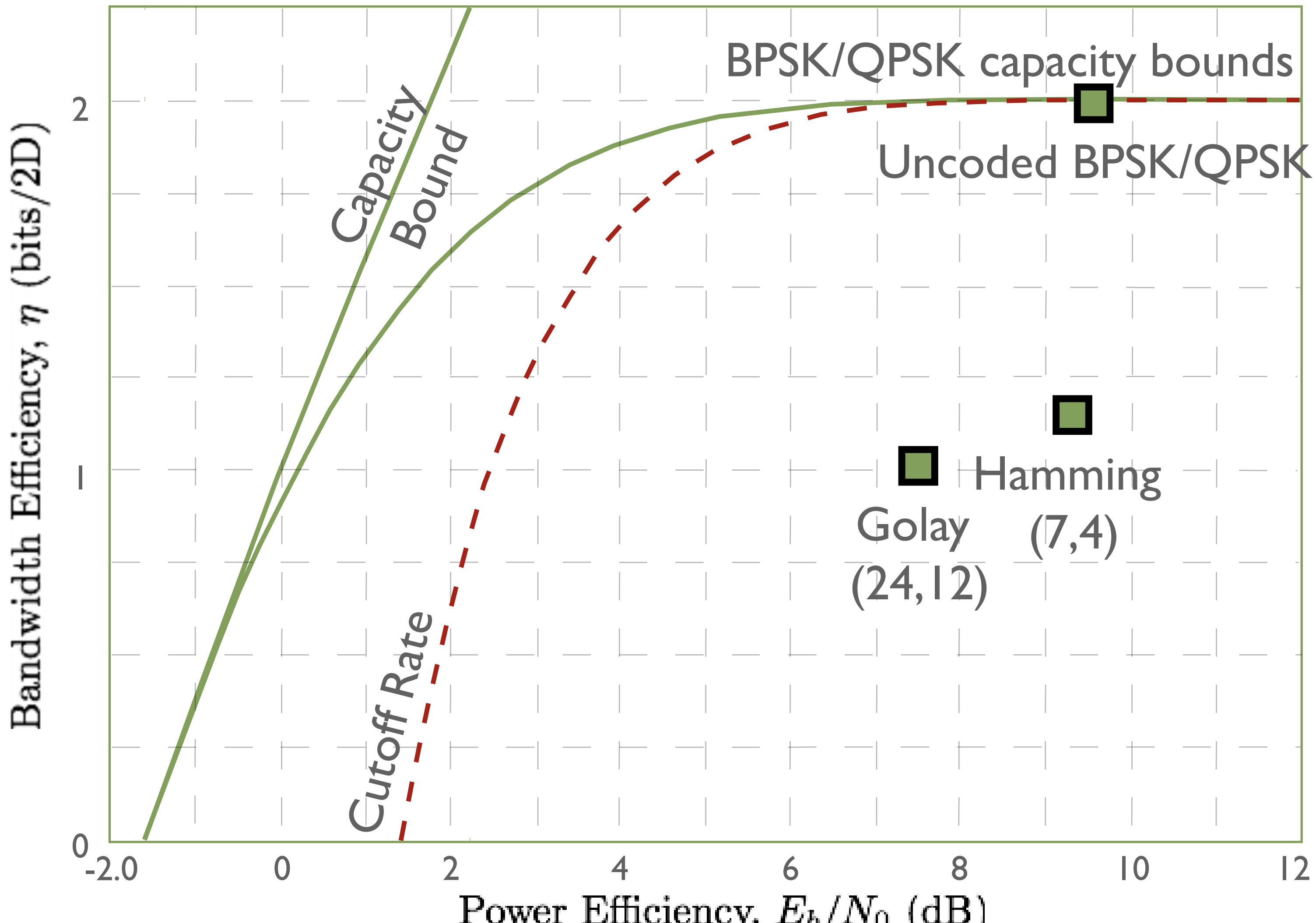
LDPC Codes: motivation (for a target BER 10^{-5})



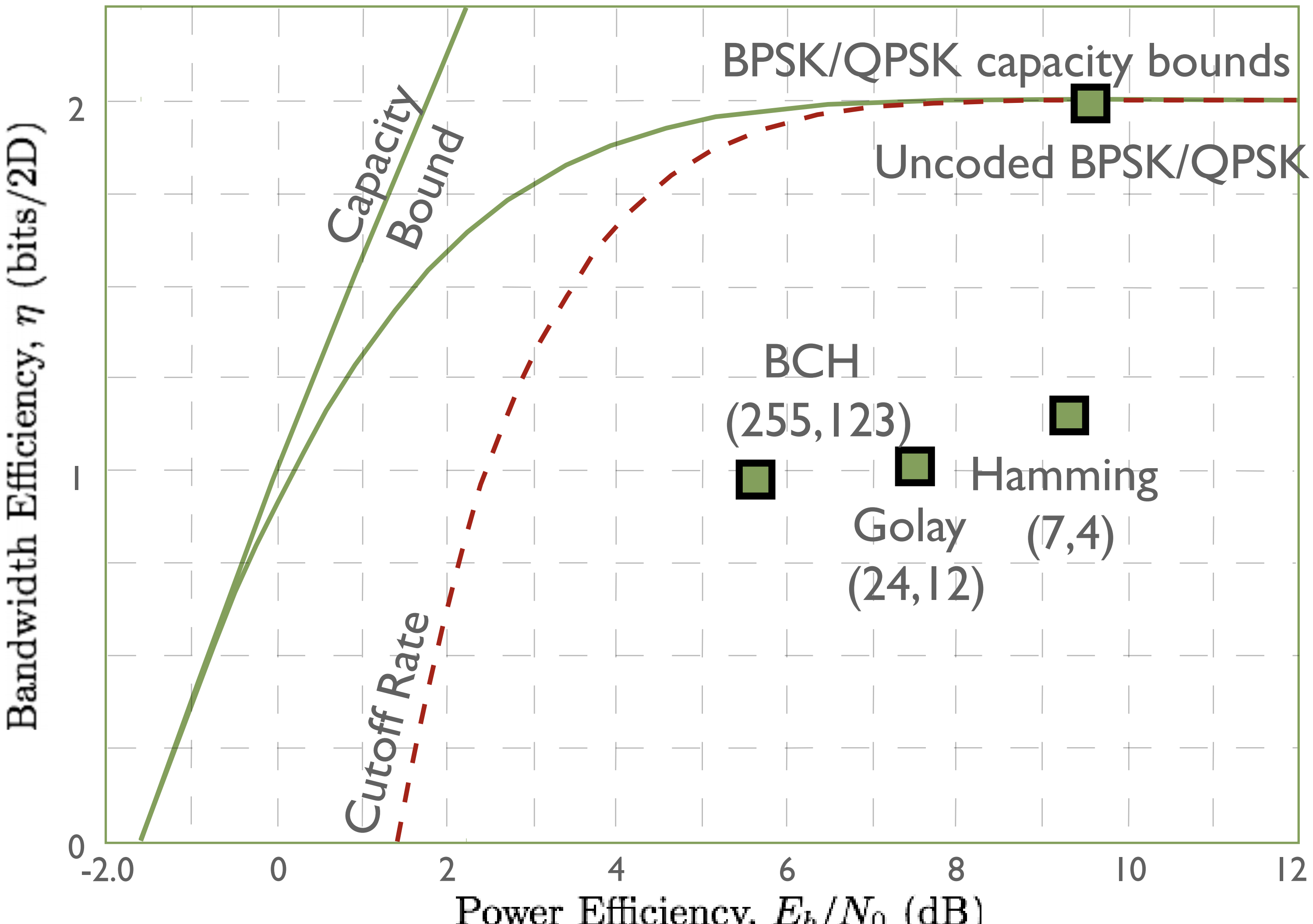
LDPC Codes: motivation (for a target BER 10^{-5})



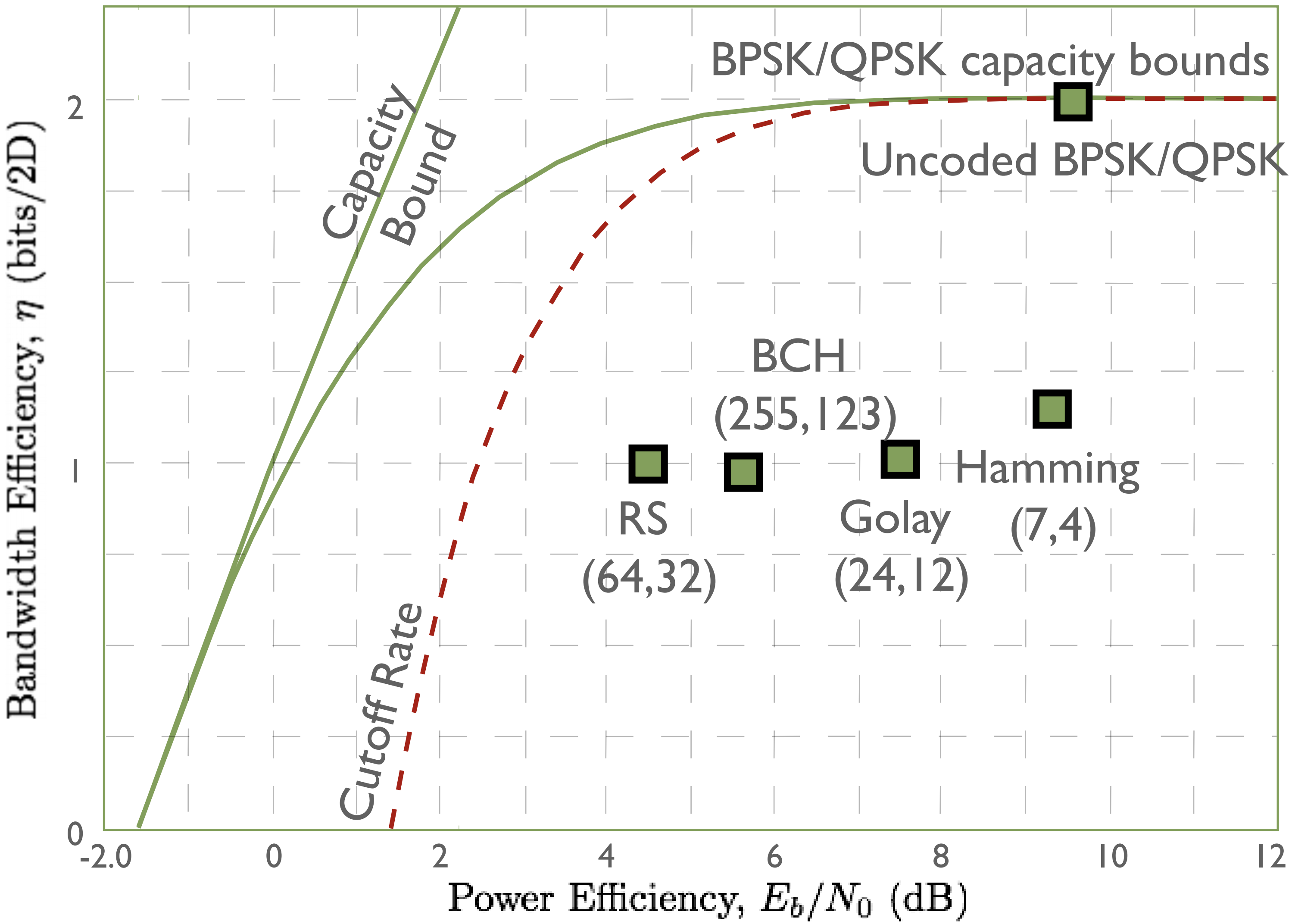
LDPC Codes: motivation (for a target BER 10^{-5})



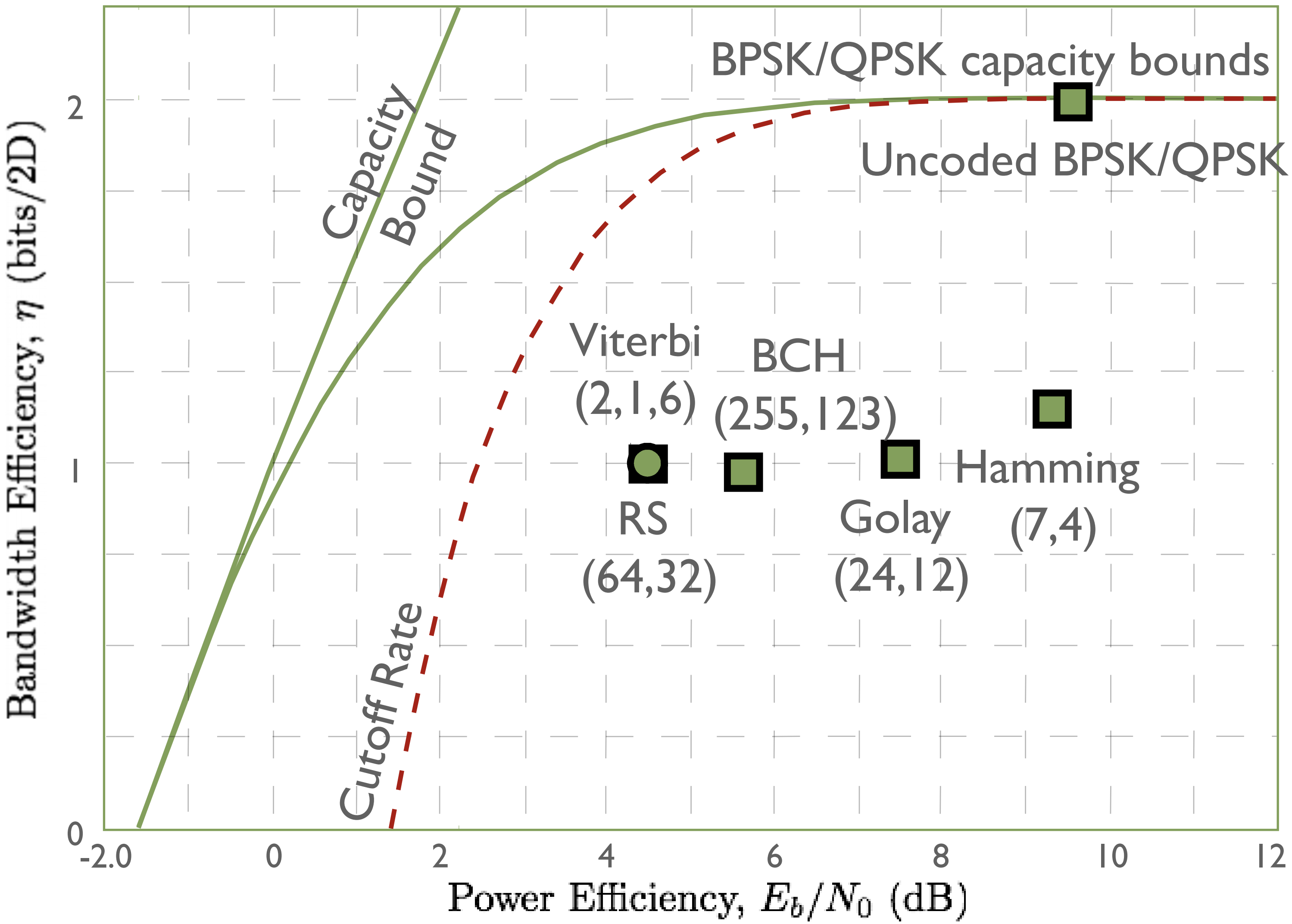
LDPC Codes: motivation (for a target BER 10^{-5})



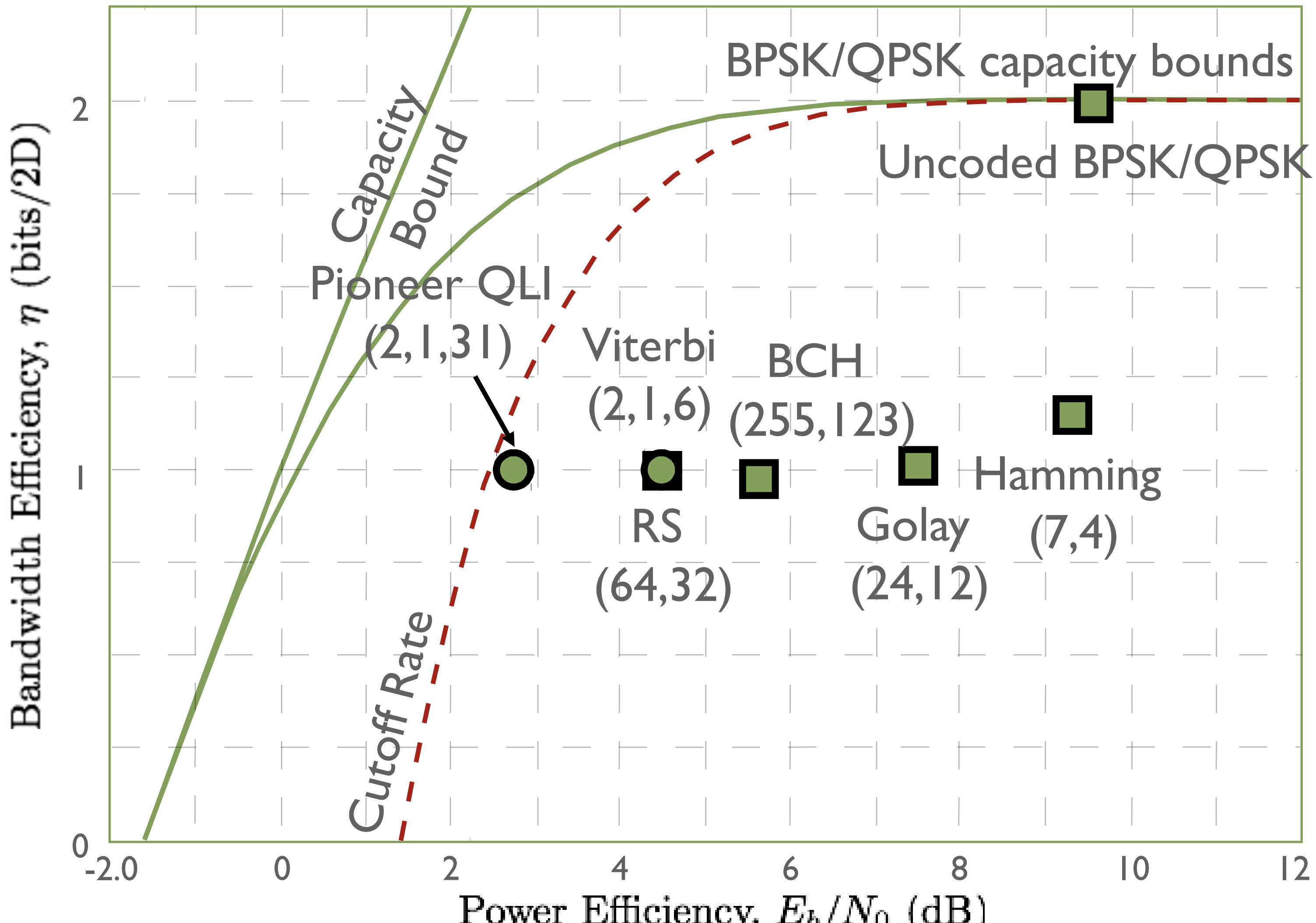
LDPC Codes: motivation (for a target BER 10^{-5})



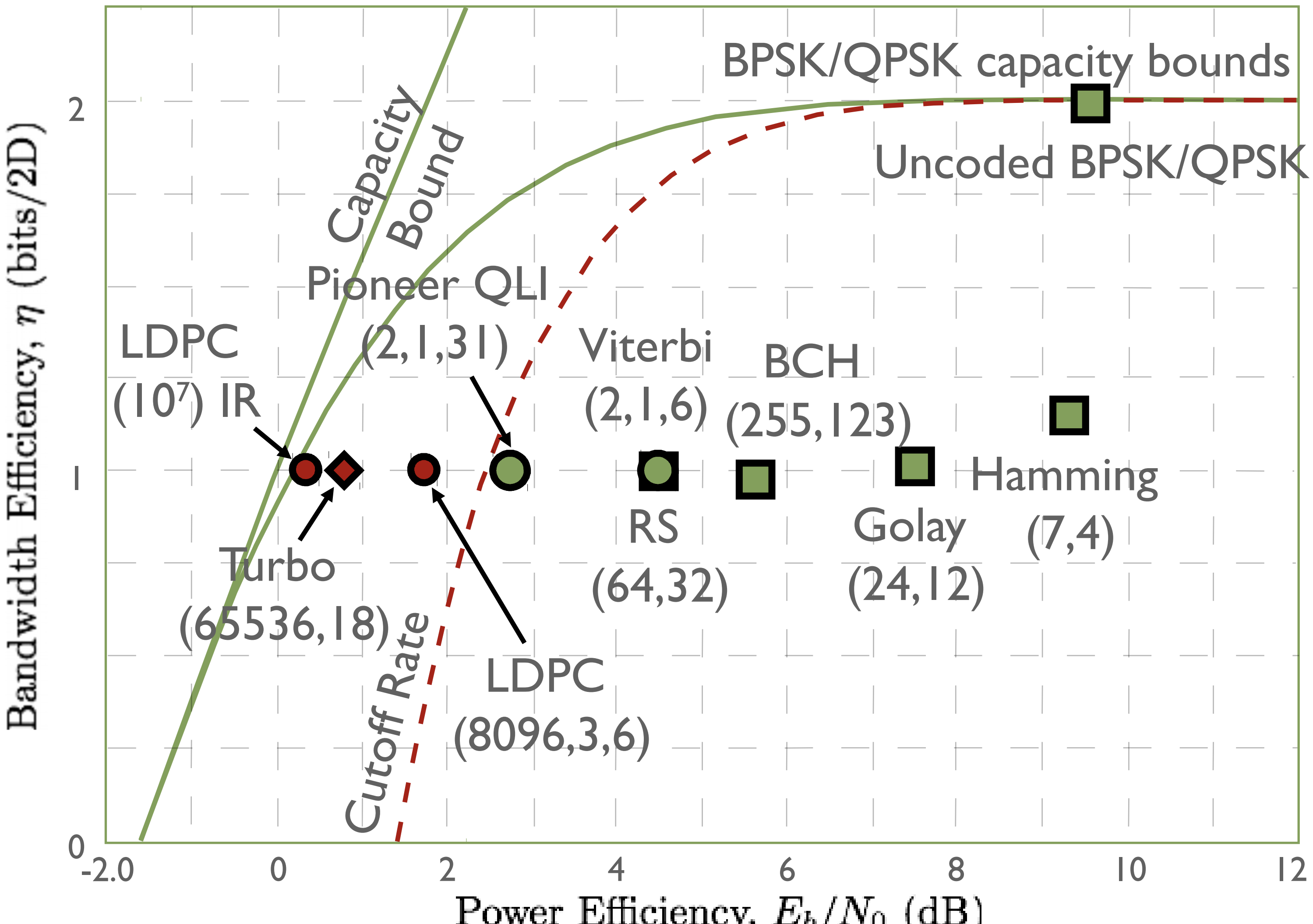
LDPC Codes: motivation (for a target BER 10^{-5})



LDPC Codes: motivation (for a target BER 10^{-5})



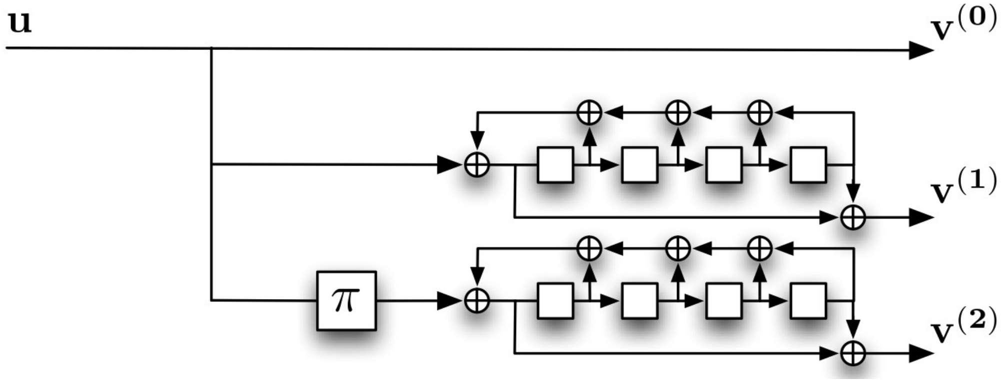
LDPC Codes: motivation (for a target BER 10^{-5})



Random-like codes (2000s - today)

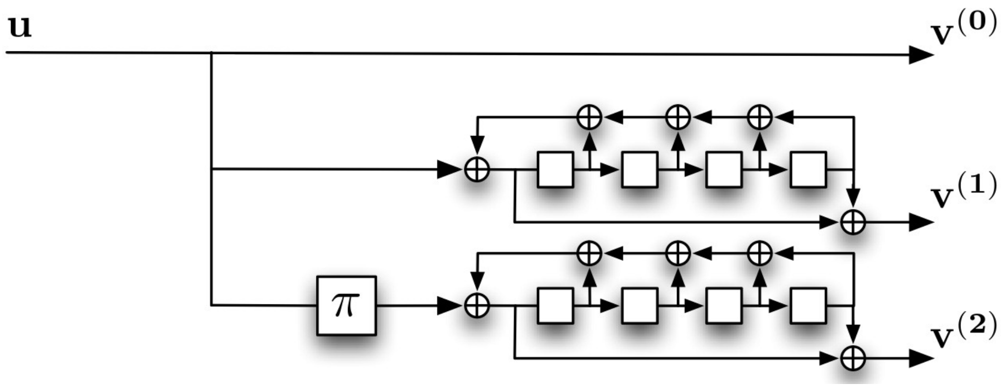
- Turbo codes use a long pseudorandom interleaver

➡ 3G and 4G telephony standards HSPA, EV-DO, LTE, satellite DVB-RCS, Mars Reconnaissance Rover, WiMAX, and so on.



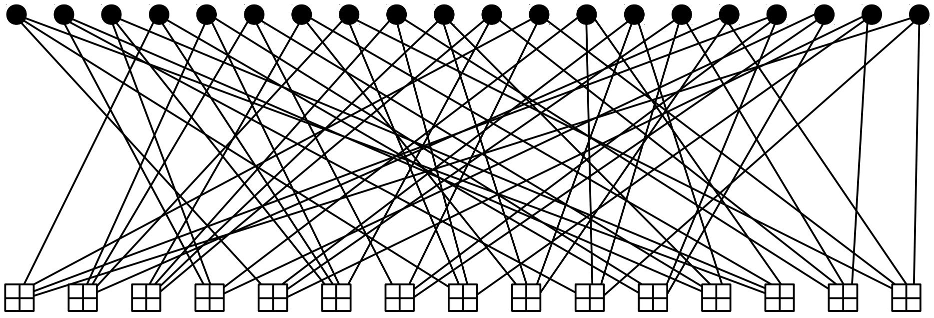
Random-like codes (2000s - today)

- Turbo codes use a long pseudorandom interleaver



➡ 3G and 4G telephony standards HSPA, EV-DO, LTE, satellite DVB-RCS, Mars Reconnaissance Rover, WiMAX, and so on.

- Low-density parity-check (LDPC) codes are defined on a large sparse graph



➡ DVB-S2, ITU-T G.hn standard (data networking over power lines, phone lines, and coaxial cables), 10GBase-T Ethernet, Wi-Fi standards 802.11, and so on.

Outline

- Introduction: From Shannon to Modern Coding Theory
  - Channel capacity, structured codes, random codes, LDPC codes
- **LDPC Block Codes**
  - Parity-check matrix and Tanner graph representations, minimum distance bounds, iterative decoding thresholds, protograph-based constructions
- Spatially Coupled LDPC Codes
  - Protograph representation, edge-spreading construction, termination
  - Iterative decoding thresholds, threshold saturation, minimum distance
- Practical Considerations
  - Window decoding, performance, latency, and complexity comparisons to LDPC block codes, rate-compatibility, implementation aspects

Definitions

- Recall, an (n,k) binary linear code C is a k-dimensional subspace of $\{0,1\}^n$
  - G is a generator matrix for C if its rows span C. (G is an $l \times n$ matrix where $l \geq k$.)
  - H is a parity-check matrix for C if its rows span $C^{\perp}$, i.e., $v \in C$ iff $vH^T = 0$. (H is an $m \times n$ matrix where $m \geq n-k$.)

Definitions

UNIVERSITY OF NOTRE DAME

- Recall, an (n,k) binary linear code C is a k-dimensional subspace of $\{0,1\}^n$
  - ➡ G is a generator matrix for C if its rows span C. (G is an $l \times n$ matrix where $l \geq k$.)
  - ➡ H is a parity-check matrix for C if its rows span $C^{\perp}$, i.e., $v \in C$ iff $vH^T = 0$. (H is an $m \times n$ matrix where $m \geq n-k$.)

Definition: A **low-density parity-check** (LDPC) code is a code for which the parity-check matrix of interest has a low density of ones.

- ➡ LDPC refers to the **representation** of a code, rather than the code itself.
- ➡ "low" is a vague term. (Complexity of decoding increases with the density of ones.)

Definitions

UNIVERSITY OF NOTRE DAME

- LDPC block code designs can be classified as either **regular** or **irregular**.

Definitions

- LDPC block code designs can be classified as either **regular** or **irregular**.

(1) Definition: A **(J,K)-regular** (n,k) LDPC code is a code for which the $m \times n$ parity check matrix has K ones in every row and J ones in every column.

Definitions

- LDPC block code designs can be classified as either **regular** or **irregular**.

(1) Definition: A **(J,K)-regular** (n,k) LDPC code is a code for which the $m \times n$ parity check matrix has K ones in every row and J ones in every column.

➡ Each parity-check contains K code bits and each code bit is involved in J parity-checks.

Definitions

- LDPC block code designs can be classified as either **regular** or **irregular**.

(1) Definition: A **(J,K)-regular** (n,k) LDPC code is a code for which the $m \times n$ parity check matrix has K ones in every row and J ones in every column.

➡ Each parity-check contains K code bits and each code bit is involved in J parity-checks.

➡ "low-density" means $J \ll m$ and $K \ll n$

Definitions

- LDPC block code designs can be classified as either **regular** or **irregular**.

(1) Definition: A **(J,K)-regular** (n,k) LDPC code is a code for which the $m \times n$ parity check matrix has K ones in every row and J ones in every column.

- ➡ Each parity-check contains K code bits and each code bit is involved in J parity-checks.
- ➡ "low-density" means $J \ll m$ and $K \ll n$
- ➡ $Jn = Km$ is the number of ones in H

Definitions

- LDPC block code designs can be classified as either **regular** or **irregular**.

(1) Definition: A **(J,K)-regular** (n,k) LDPC code is a code for which the $m \times n$ parity check matrix has K ones in every row and J ones in every column.

- ➡ Each parity-check contains K code bits and each code bit is involved in J parity-checks.
- ➡ "low-density" means $J \ll m$ and $K \ll n$
- ➡ $Jn = Km$ is the number of ones in H
- ➡ $m \geq n - k$ means $R = \frac{k}{n} \geq 1 - \frac{J}{K}$, and thus $J < K$.

Definitions

- LDPC block code designs can be classified as either **regular** or **irregular**.

(1) Definition: A **(J,K)-regular** (n,k) LDPC code is a code for which the $m \times n$ parity check matrix has K ones in every row and J ones in every column.

➡ Each parity-check contains K code bits and each code bit is involved in J parity-checks.

➡ "low-density" means $J \ll m$ and $K \ll n$

➡ $Jn = Km$ is the number of ones in H

➡ $m \geq n - k$ means $R = \frac{k}{n} \geq 1 - \frac{J}{K}$, and thus $J < K$.

(2) Definition: An **irregular** (n,k) LDPC code is one in which the row and/or column weights of the parity-check matrix are not constant.

LDPC Block Codes

Definition by parity-check matrix: [Gallager, '62]

$$
H = \left[\begin{array}{cccccccccccccccccccc}
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&1\\
0&0&0&0&1&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&0\\
\hline
0&0&1&1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&1&0&0\\
1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&1&0&0&1&0&0&0&1&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&1\\
\hline
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
1&0&1&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1
\end{array}\right]_{15 \times 20}
$$

Code: $\{\mathbf{v} \mid \mathbf{v}\mathbf{H}^{\mathrm{T}} = \mathbf{0}\}$

(null space of a **sparse** parity-check matrix $\mathbf{H}$)

LDPC Block Codes

Definition by parity-check matrix: [Gallager, '62]

$$
H = \left[\begin{array}{cccccccccccccccccccc}
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&1\\
0&0&0&0&1&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&0\\
\hline
0&0&1&1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&1&0&0\\
1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&1&0&0&1&0&0&0&1&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&1\\
\hline
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
1&0&1&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1
\end{array}\right]
$$

15×20

Code: $\{\mathbf{v} \mid \mathbf{v}\mathbf{H}^{\mathrm{T}} = \mathbf{0}\}$

(null space of a **sparse** parity-check matrix $\mathbf{H}$)

Regular LDPC code:

Column weight: $J = 3$

Row weight: $K = 4$

$$R \geq 1 - \frac{J}{K}$$

LDPC Block Codes

UNIVERSITY OF NOTRE DAME

Definition by parity-check matrix: [Gallager, '62]

$$
\boldsymbol{H} = \left[\begin{array}{cccccccccccccccccccc}
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&1\\
0&0&0&0&1&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&0\\
\hline
0&0&1&1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&1&0&0\\
1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&1&0&0&1&0&0&0&1&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&1\\
\hline
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
1&0&1&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1
\end{array}\right]_{15 \times 20}
$$

Code: $\{\mathbf{v} \mid \mathbf{v}\mathbf{H}^{\mathrm{T}} = \mathbf{0}\}$

(null space of a **sparse** parity-check matrix $\mathbf{H}$)

Regular LDPC code:

Column weight: $J = 3$

Row weight: $K = 4$

$$R \geq 1 - \frac{J}{K}$$

- If the row and column weights J and K are not constant, then the LDPC code is **irregular** (more later)

Graphical representation of LDPC Codes

Definition: A **bipartite graph** is one in which the nodes can be partitioned into two classes, and no edge can connect two nodes from the same class

Graphical representation of LDPC Codes

Definition: A **bipartite graph** is one in which the nodes can be partitioned into two classes, and no edge can connect two nodes from the same class

Definition: A **Tanner graph** for an LDPC code is a bipartite graph such that:

- In the first class of nodes, there is one node for each of the n bits in the codeword (these are called the “bit nodes” or “variable nodes”).

Graphical representation of LDPC Codes

Definition: A **bipartite graph** is one in which the nodes can be partitioned into two classes, and no edge can connect two nodes from the same class

Definition: A **Tanner graph** for an LDPC code is a bipartite graph such that:

- In the first class of nodes, there is one node for each of the n bits in the codeword (these are called the "bit nodes" or "variable nodes").
- In the second class of nodes, there is one node for each of the m parity-checks (these are called the "check nodes", "constraint nodes", or "function nodes").

Graphical representation of LDPC Codes

UNIVERSITY OF NOTRE DAME

Definition: A **bipartite graph** is one in which the nodes can be partitioned into two classes, and no edge can connect two nodes from the same class

Definition: A **Tanner graph** for an LDPC code is a bipartite graph such that:

- In the first class of nodes, there is one node for each of the n bits in the codeword (these are called the “bit nodes” or “variable nodes”).
- In the second class of nodes, there is one node for each of the m parity-checks (these are called the “check nodes”, “constraint nodes”, or “function nodes”).
- An edge connects a variable node to a check node if and only if that bit is included in that parity-check, i.e., iff H has a one in the associated position.

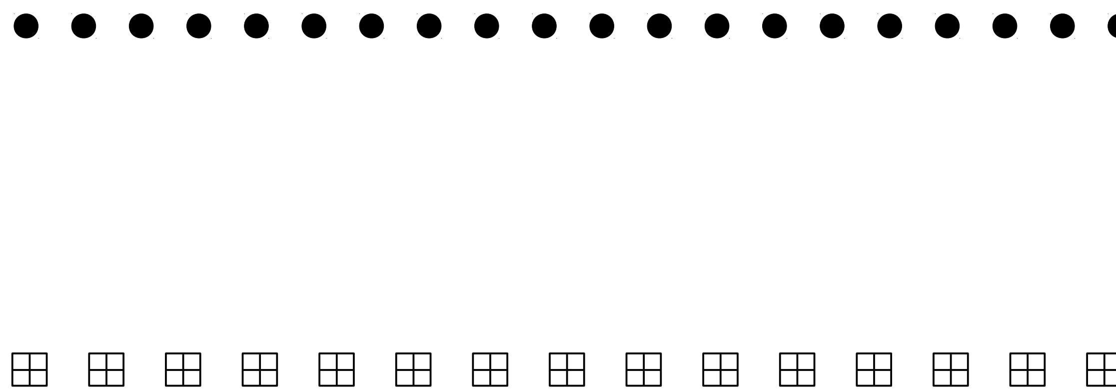
LDPC Block Codes

UNIVERSITY OF NOTRE DAME

Representation by bipartite graph: [Tanner, '81]

$$H = \left[\begin{array}{cccccccccccccccccccc}
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&1\\
0&0&0&0&1&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&0\\
\hline
0&0&1&1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&1&0&0\\
1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&1&0&0&1&0&0&0&1&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&1\\
\hline
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
1&0&1&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1
\end{array}\right]_{15 \times 20}$$

$n = 20$ **variable nodes** of degree $J = 3$



$l = 15$ **check nodes** of degree $K = 4$

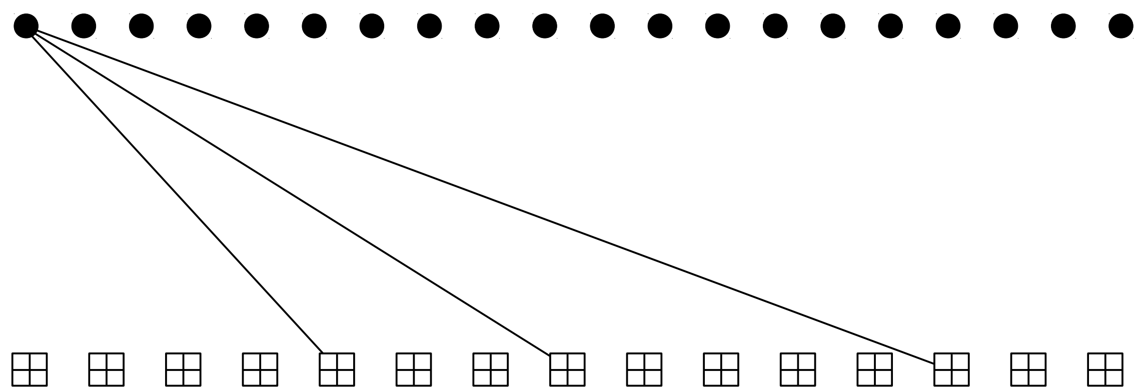
LDPC Block Codes

Representation by bipartite graph: [Tanner, '81]

$$H = \left[\begin{array}{cccccccccccccccccccc}
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&1\\
0&0&0&0&1&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&0\\
\hline
0&0&1&1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&1&0&0\\
1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&1&0&0&1&0&0&0&1&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&1\\
\hline
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
1&0&1&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1
\end{array}\right]$$

15×20

$n = 20$ **variable nodes** of degree $J = 3$



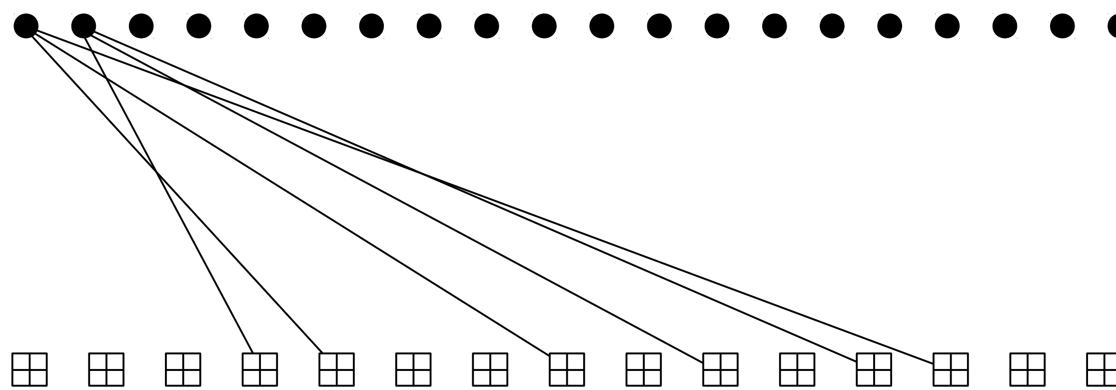
$l = 15$ **check nodes** of degree $K = 4$

LDPC Block Codes

Representation by bipartite graph: [Tanner, '81]

$$
H = \left[\begin{array}{cccccccccccccccccccc}
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&1\\
0&0&0&0&1&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&0\\
\hline
0&0&1&1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&1&0&0\\
1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&1&0&0&1&0&0&0&1&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&1\\
\hline
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
1&0&1&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1
\end{array}\right]_{15 \times 20}
$$

$n = 20$ **variable nodes** of degree $J = 3$



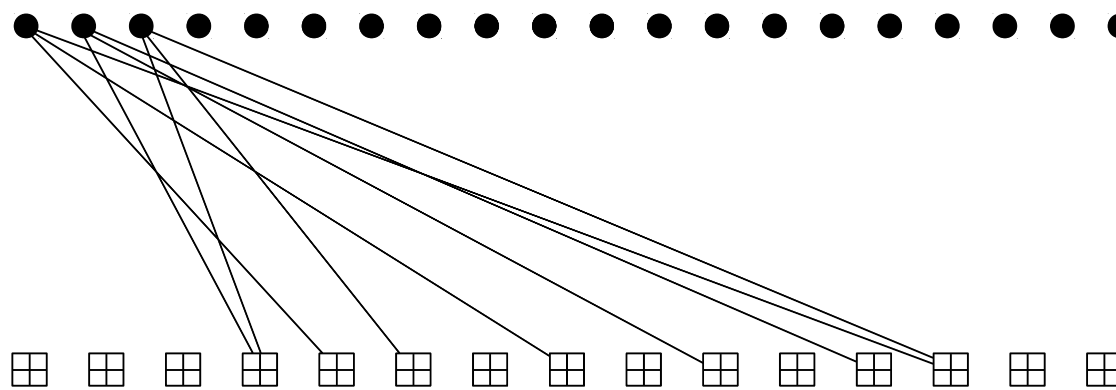
$l = 15$ **check nodes** of degree $K = 4$

LDPC Block Codes

Representation by bipartite graph: [Tanner, '81]

$$
H = \left[\begin{array}{cccccccccccccccccccc}
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&1\\
0&0&0&0&1&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&0\\
\hline
0&0&1&1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&1&0&0\\
1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&1&0&0&1&0&0&0&1&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&1\\
\hline
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
1&0&1&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1
\end{array}\right]_{15 \times 20}
$$

$n = 20$ **variable nodes** of degree $J = 3$



$l = 15$ **check nodes** of degree $K = 4$

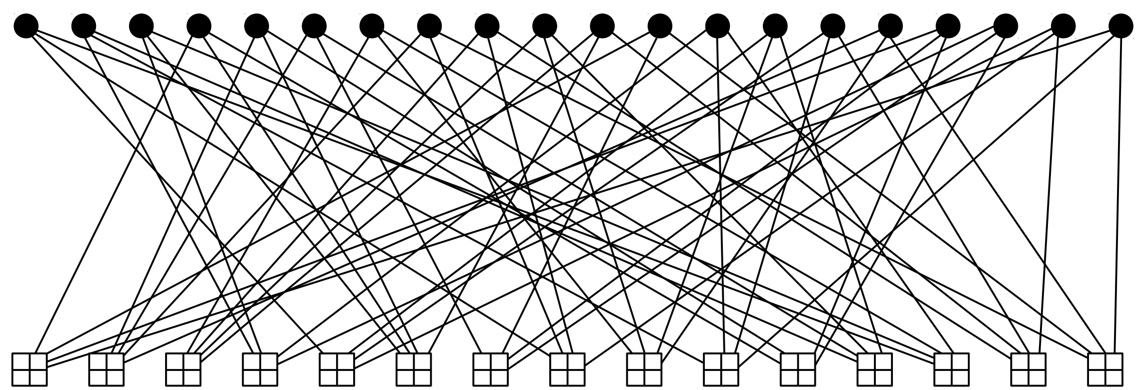
LDPC Block Codes

UNIVERSITY OF NOTRE DAME

Representation by bipartite graph: [Tanner, '81]

$$
H = \left[\begin{array}{cccccccccccccccccccc}
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&1\\
0&0&0&0&1&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&0\\
\hline
0&0&1&1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&1&0&0\\
1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&1&0&0&1&0&0&0&1&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&1\\
\hline
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
1&0&1&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1
\end{array}\right]_{15 \times 20}
$$

$n = 20$ **variable nodes** of degree $J = 3$



$l = 15$ **check nodes** of degree $K = 4$

Cycles in Tanner graphs

Definition: A **cycle** of length l in a Tanner graph is a path comprised of l edges from a node back to the same node.

Cycles in Tanner graphs

Definition: A **cycle** of length l in a Tanner graph is a path comprised of l edges from a node back to the same node.

Definition: The **girth** of a Tanner graph is the minimum length of any cycle in the graph.

Cycles in Tanner graphs

UNIVERSITY OF NOTRE DAME

Definition: A **cycle** of length l in a Tanner graph is a path comprised of l edges from a node back to the same node.

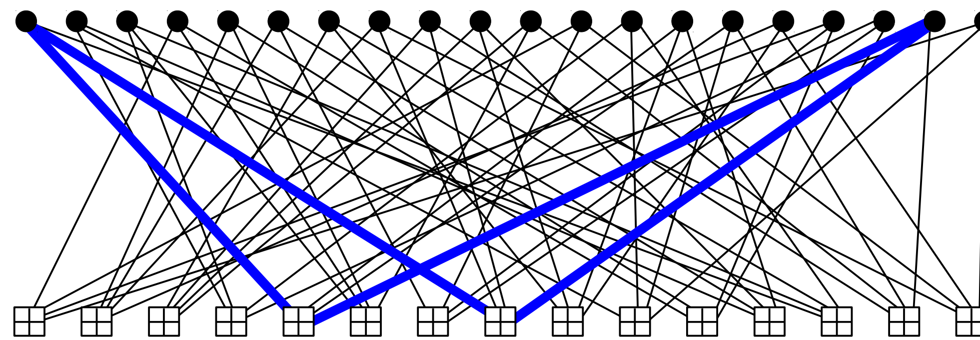
Definition: The **girth** of a Tanner graph is the minimum length of any cycle in the graph.

- The shortest possible cycle in any graph is 4 (indicated by a "rectangle" of four ones in $\boldsymbol{H}$).
- Short cycles are usually considered to be bad for message passing decoding (more later).

Cycles in Tanner graphs

- Tanner graphs typically contain **cycles**. The shortest cycle is called the **girth**.

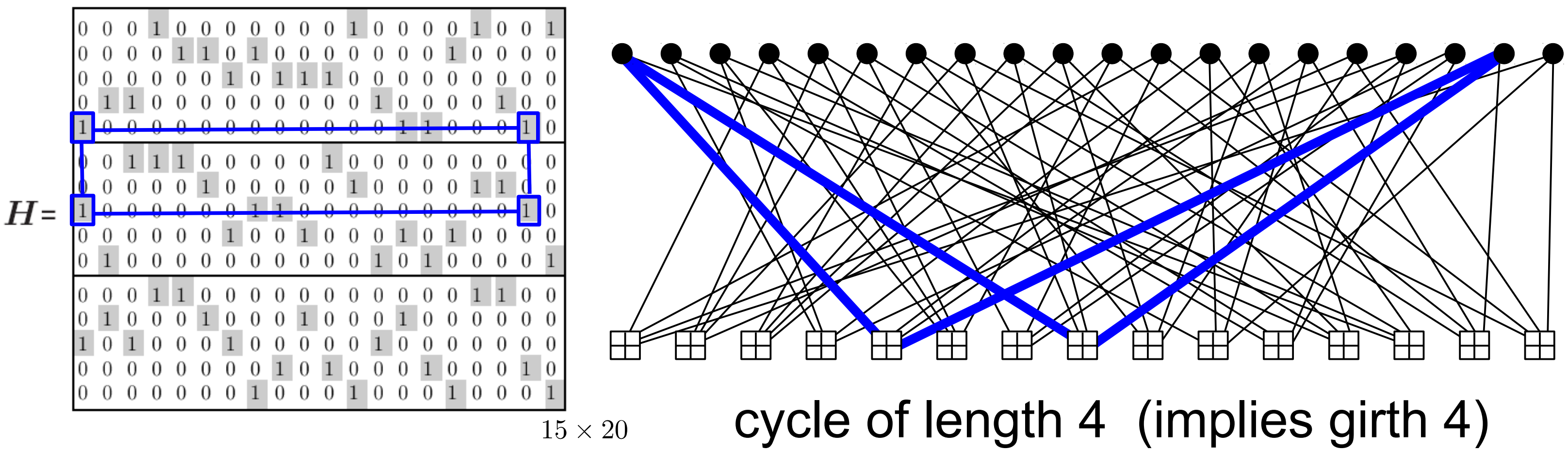
$$H = \begin{bmatrix}
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&1\\
0&0&0&0&1&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&0\\
\hline
0&0&1&1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&1&0&0\\
1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&1&0&0&1&0&0&0&1&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&1\\
\hline
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
1&0&1&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1
\end{bmatrix}_{15 \times 20}$$



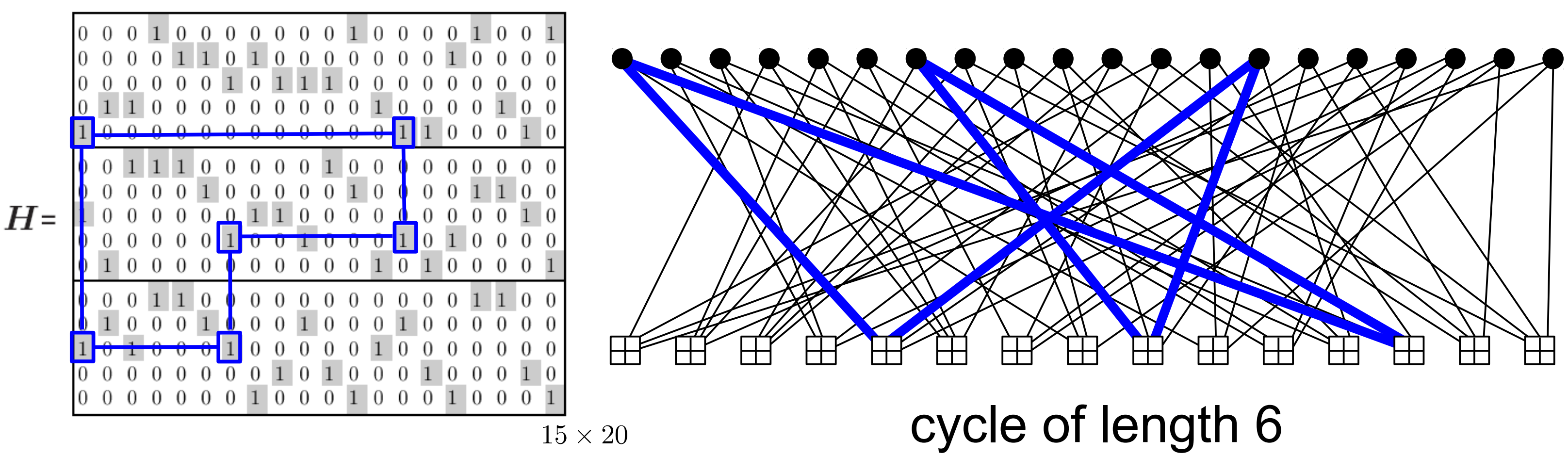
cycle of length 4 (implies girth 4)

Cycles in Tanner graphs

- Tanner graphs typically contain **cycles**. The shortest cycle is called the **girth**.



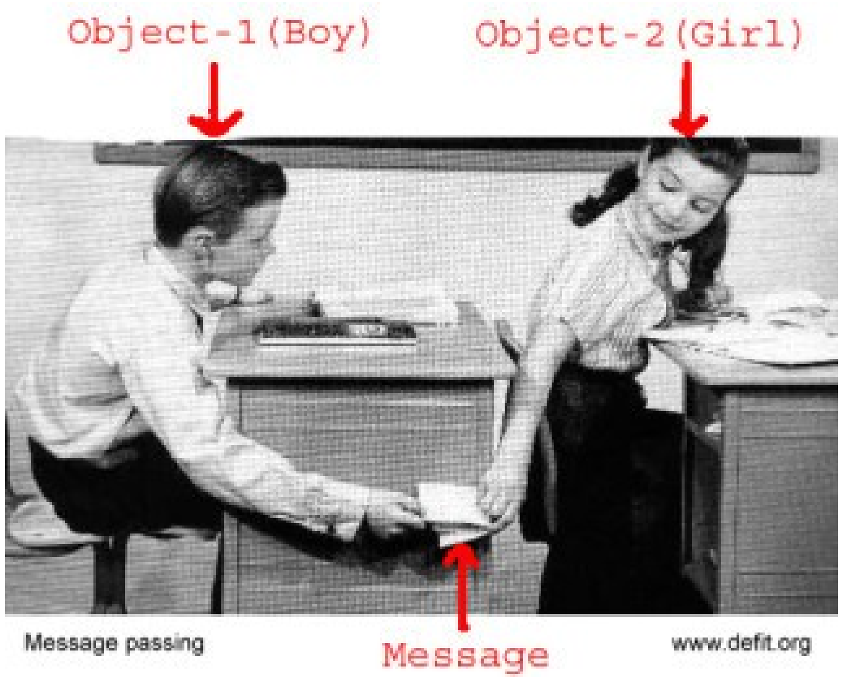
cycle of length 4 (implies girth 4)



cycle of length 6

Message passing decoding

- The class of decoding algorithms used to decode LDPC codes are collectively called **message passing (MP)** algorithms since their operation can be explained by the iterative passing of messages between nodes in the Tanner graph.

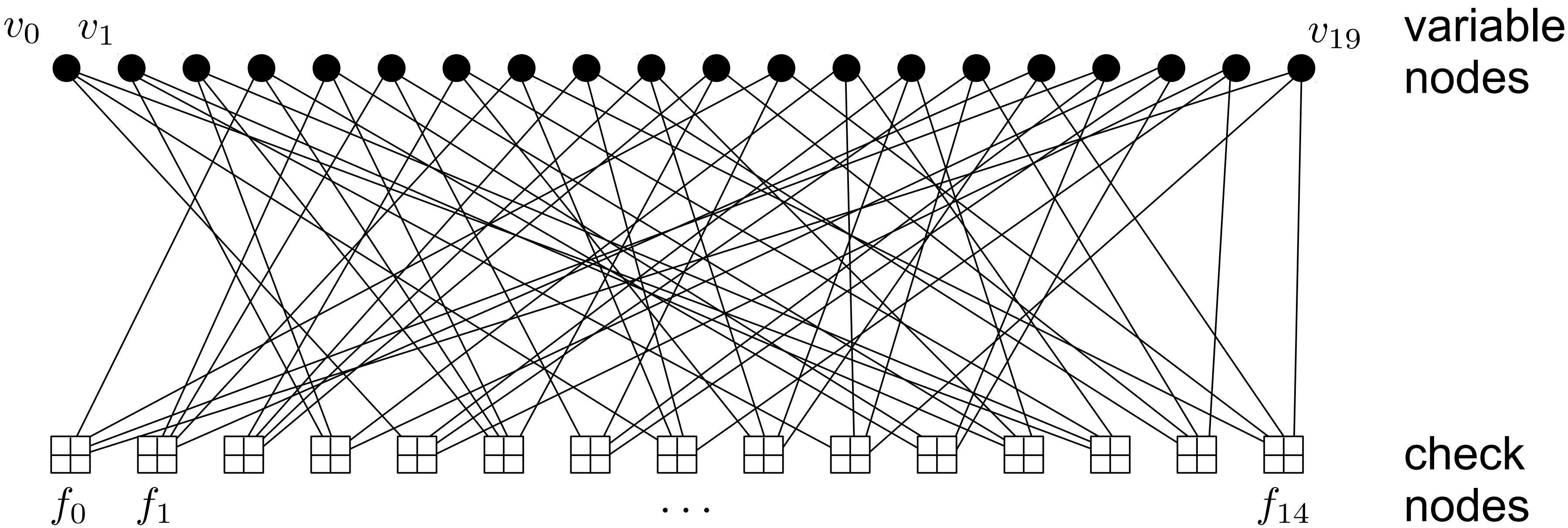


Message Passing Decoding

- Graph-based codes can be decoded **iteratively** with **low-complexity**

→ **Iterative decoding:** exchange of messages in Tanner graph

- We modify the graph as follows:



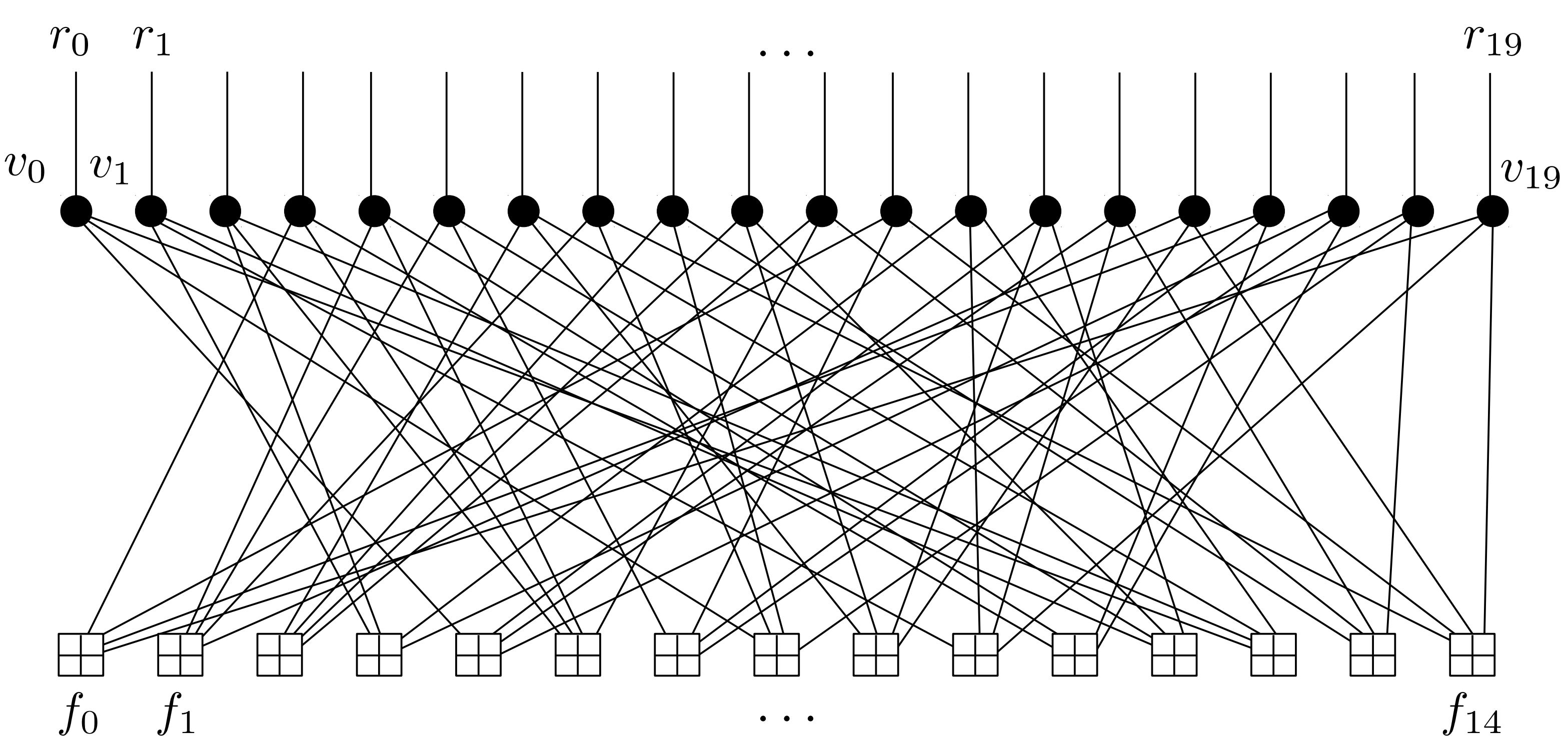
Message Passing Decoding

UNIVERSITY OF NOTRE DAME

- Graph-based codes can be decoded **iteratively** with **low-complexity**

➡ **Iterative decoding:** exchange of messages in Tanner graph

- We modify the graph as follows:



Belief Propagation (BP) decoding

- BP decoding is a **soft decision** algorithm, where the messages are **probabilities** which represent a level of belief about the value of the codeword bits

Belief Propagation (BP) decoding

UNIVERSITY OF NOTRE DAME

- BP decoding is a **soft decision** algorithm, where the messages are **probabilities** which represent a level of belief about the value of the codeword bits

- BP decoding can be carried out in the probability domain or the log domain, but it is often more convenient to work with **log likelihoods.**

  ➡ Costly multiplications become less costly additions

  ➡ The product of many probabilities can become numerically unstable.

Belief Propagation (BP) decoding

UNIVERSITY OF NOTRE DAME

- BP decoding is a **soft decision** algorithm, where the messages are **probabilities** which represent a level of belief about the value of the codeword bits
- BP decoding can be carried out in the probability domain or the log domain, but it is often more convenient to work with **log likelihoods.**

  → Costly multiplications become less costly additions

  → The product of many probabilities can become numerically unstable.

- In the log domain, the algorithm is also referred to as the **sum product algorithm (SPA).**

Belief Propagation (BP) decoding

- The **log-likelihood ratio (LLR)** of a binary variable is

$$L(x) = \log\left(\frac{\mathbb{P}(x=0)}{\mathbb{P}(x=1)}\right) = \log(\mathbb{P}(x=0)) - \log(\mathbb{P}(x=1))$$

Belief Propagation (BP) decoding

- The **log-likelihood ratio (LLR)** of a binary variable is

$$L(x) = \log\left(\frac{\mathbb{P}(x=0)}{\mathbb{P}(x=1)}\right) = \log(\mathbb{P}(x=0)) - \log(\mathbb{P}(x=1))$$

- If $\mathbb{P}(x=0) > \mathbb{P}(x=1)$, $L(x) > 0$ and a larger difference results in a larger LLR

Belief Propagation (BP) decoding

- The **log-likelihood ratio (LLR)** of a binary variable is

$$L(x) = \log\left(\frac{\mathbb{P}(x=0)}{\mathbb{P}(x=1)}\right) = \log(\mathbb{P}(x=0)) - \log(\mathbb{P}(x=1))$$

- If $\mathbb{P}(x=0) > \mathbb{P}(x=1)$, $L(x) > 0$ and a larger difference results in a larger LLR

- If $\mathbb{P}(x=1) > \mathbb{P}(x=0)$, $L(x) < 0$ and a larger difference results in a larger (negative) LLR

Belief Propagation (BP) decoding

- The **log-likelihood ratio (LLR)** of a binary variable is

$$L(x) = \log\left(\frac{\mathbb{P}(x=0)}{\mathbb{P}(x=1)}\right) = \log(\mathbb{P}(x=0)) - \log(\mathbb{P}(x=1))$$

- If $\mathbb{P}(x=0) > \mathbb{P}(x=1)$, $L(x) > 0$ and a larger difference results in a larger LLR

- If $\mathbb{P}(x=1) > \mathbb{P}(x=0)$, $L(x) < 0$ and a larger difference results in a larger (negative) LLR

➡ $\mathrm{sgn}(L(x))$ = hard decision

$|L(x)|$ determines the reliability

Belief Propagation (BP) decoding

Input: A priori probabilities on code bits, $\boldsymbol{H}$, I_{max}

Output: A posteriori probabilities (APPs) on code bits

Goal: Make the maximum a posteriori (MAP) probability decision for each codeword bit

Belief Propagation (BP) decoding

Input: A priori probabilities on code bits, $\boldsymbol{H}$, I_{max}

Output: A posteriori probabilities (APPs) on code bits

Goal: Make the maximum a posteriori (MAP) probability decision for each codeword bit

(1) Each variable node sends a message (“extrinsic information”) to each check node it is connected to.

➡ “extrinsic” means we do not pass information the receiving node already possesses (the message sent in the previous round).

Belief Propagation (BP) decoding

Input: A priori probabilities on code bits, $\boldsymbol{H}$, I_{max}

Output: A posteriori probabilities (APPs) on code bits

Goal: Make the maximum a posteriori (MAP) probability decision for each codeword bit

(1) Each variable node sends a message (“extrinsic information”) to each check node it is connected to.

→ “extrinsic” means we do not pass information the receiving node already possesses (the message sent in the previous round).

(2) Each check node sends a message (“extrinsic information”) to each variable node it is connected to.

Belief Propagation (BP) decoding

Input: A priori probabilities on code bits, $\boldsymbol{H}$, I_{max}

Output: A posteriori probabilities (APPs) on code bits

Goal: Make the maximum a posteriori (MAP) probability decision for each codeword bit

(1) Each variable node sends a message ("extrinsic information") to each check node it is connected to.

➡ "extrinsic" means we do not pass information the receiving node already possesses (the message sent in the previous round).

(2) Each check node sends a message ("extrinsic information") to each variable node it is connected to.

(3) We compute the APP for each codeword bit given the received data and given that all the parity-checks must be satisfied

Belief Propagation (BP) decoding

Input: A priori probabilities on code bits, $\boldsymbol{H}$, I_{max}

Output: A posteriori probabilities (APPs) on code bits

Goal: Make the maximum a posteriori (MAP) probability decision for each codeword bit

(1) Each variable node sends a message (“extrinsic information”) to each check node it is connected to.

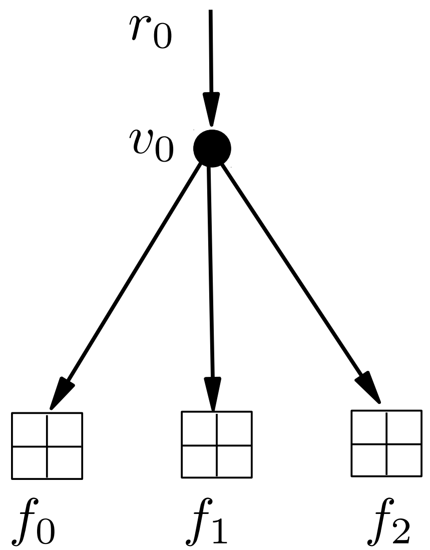
→ “extrinsic” means we do not pass information the receiving node already possesses (the message sent in the previous round).

(2) Each check node sends a message (“extrinsic information”) to each variable node it is connected to.

(3) We compute the APP for each codeword bit given the received data and given that all the parity-checks must be satisfied

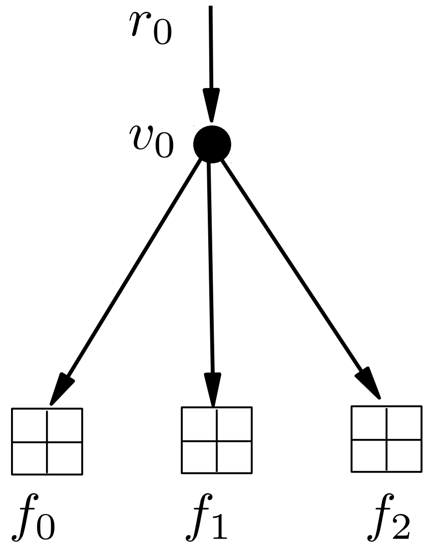
(4) We make hard decisions and terminate if $vH^T = 0$ or if I_{max}, the maximum number of iterations, is reached. Otherwise go to (1).

Belief Propagation (BP) decoding



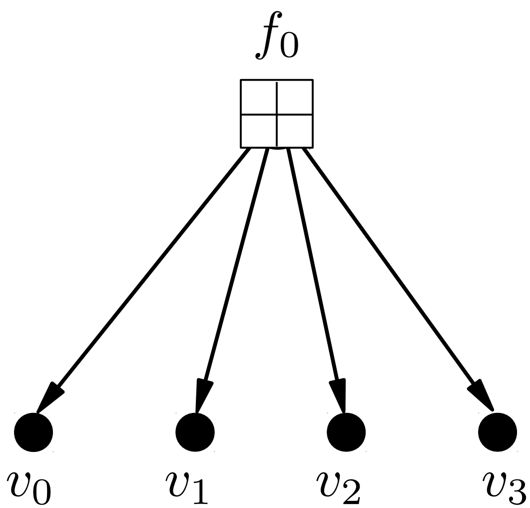
p_{ij} = message passed from variable node v_i to check node f_j
= probability v_i has a certain value given the observed value r_i and all the other messages passed to v_i in the last round from check nodes other than f_j

Belief Propagation (BP) decoding



$p_{ij} =$ message passed from variable node v_i to check node f_j

$=$ probability v_i has a certain value given the observed value r_i and all the other messages passed to v_i in the last round from check nodes other than f_j

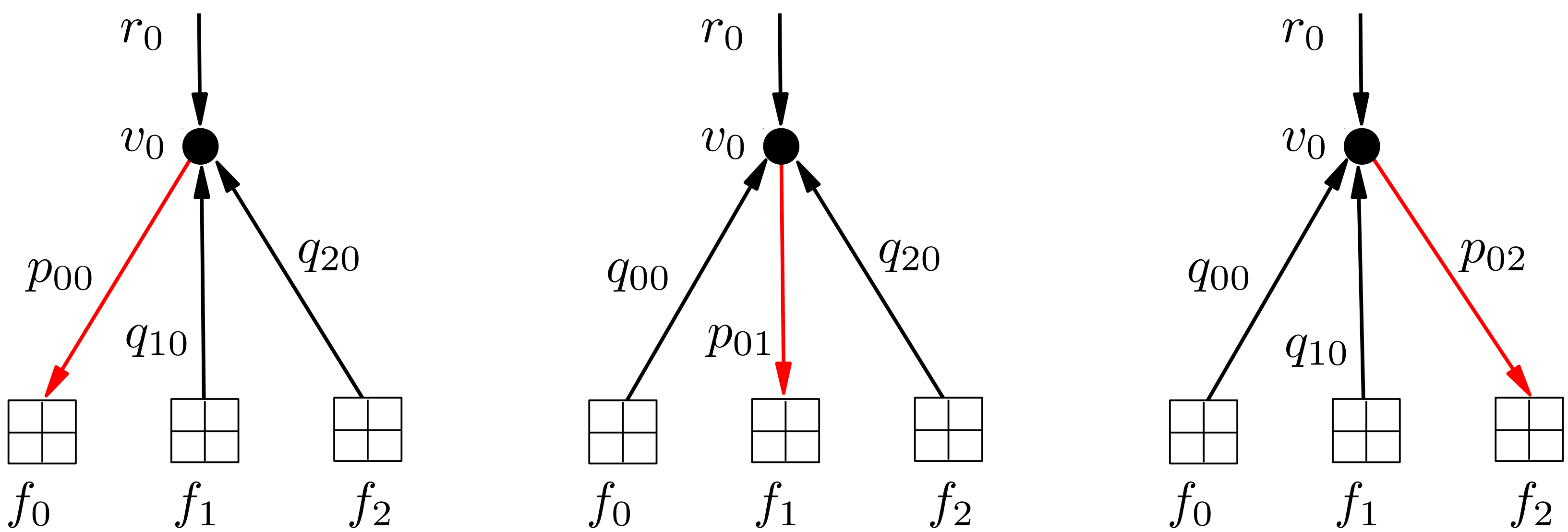


$q_{ji} =$ message passed from check node f_j to variable node v_i

$=$ probability v_i has a certain value given all the other messages passed to f_j in the last round from variable nodes other than v_i

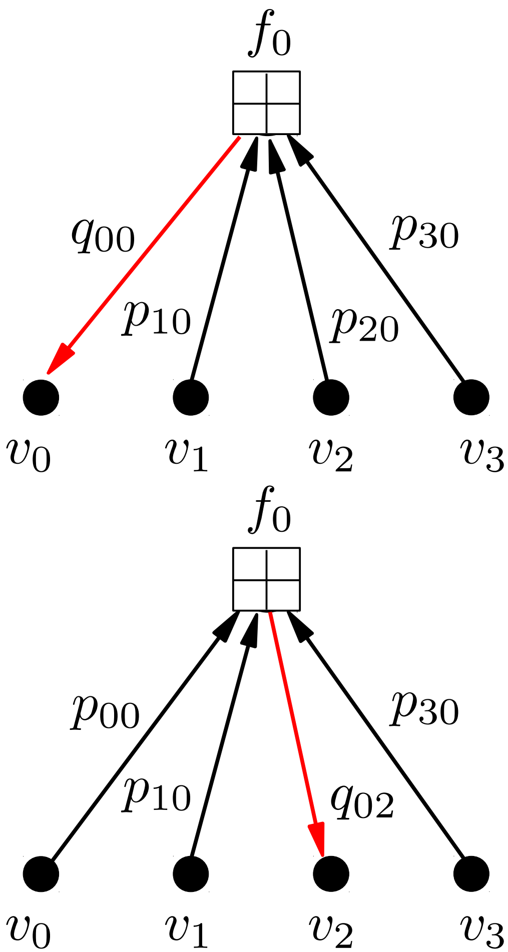
Message dependencies

- The message dependencies of the information flow into and out of a variable node look like this:



Message dependencies

- The message dependencies of the information flow into and out of a check node look like this:



Is this globally optimal?

UNIVERSITY OF NOTRE DAME

Problem: The extrinsic information from a parity-check is **independent of the a priori probability** in the first iteration only **until it is returned back via a cycle** in the graph.

➡ Cycles cause a correlation in extrinsic information.

Is this globally optimal?

Problem: The extrinsic information from a parity-check is **independent of the a priori probability** in the first iteration only **until it is returned back via a cycle** in the graph.

➡ Cycles cause a correlation in extrinsic information.

BP decoding is a MAP algorithm if no cycles exist in the graph; otherwise it is sub-optimal!

Message passing on the BEC

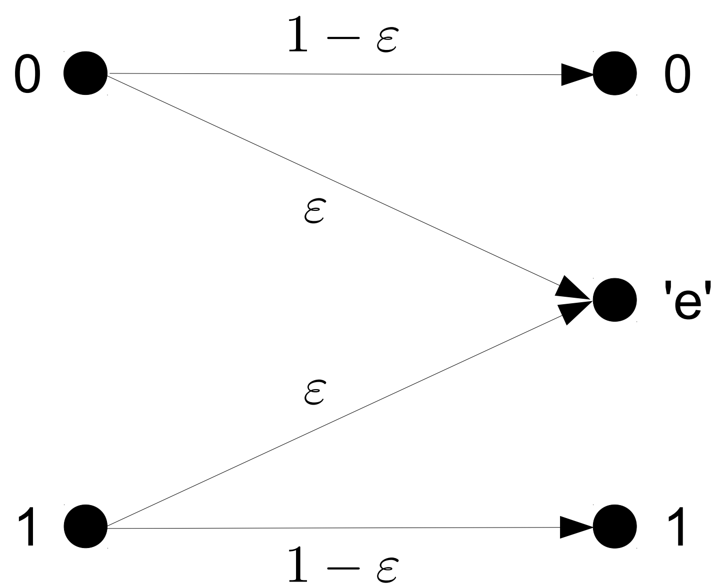
UNIVERSITY OF NOTRE DAME

Recall: On the binary erasure channel (BEC), a bit is either received correctly or erased with probability ε

Message passing on the BEC

Recall: On the binary erasure channel (BEC), a bit is either received correctly or erased with probability ε

➡ The BEC is **much simpler** to analyze than the BSC or AWGN channels.



Belief Propagation - BEC

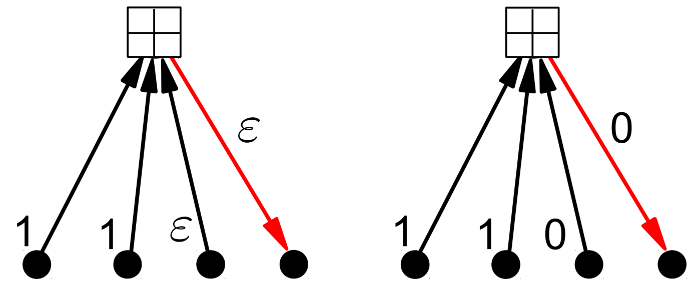
UNIVERSITY OF NOTRE DAME

- Iterative message passing decoding, where the messages represent **probabilities**, or **beliefs**, is known as **belief propagation (BP)** decoding

Belief Propagation - BEC

- Iterative message passing decoding, where the messages represent **probabilities**, or **beliefs**, is known as **belief propagation (BP)** decoding
- For the BEC, BP decoding simplifies to the following two rules:

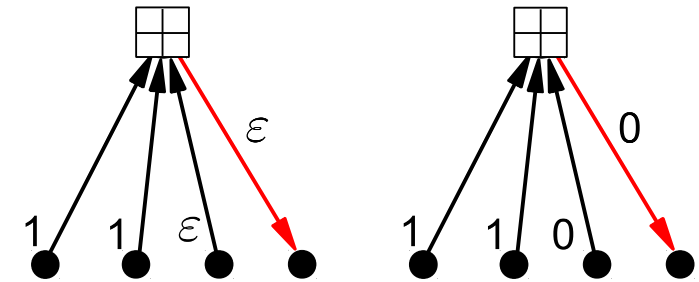
→ At a **check node**, the outgoing message is an erasure if **any** of the incoming messages are erasures; otherwise the message is the mod-2 sum of the incoming messages.



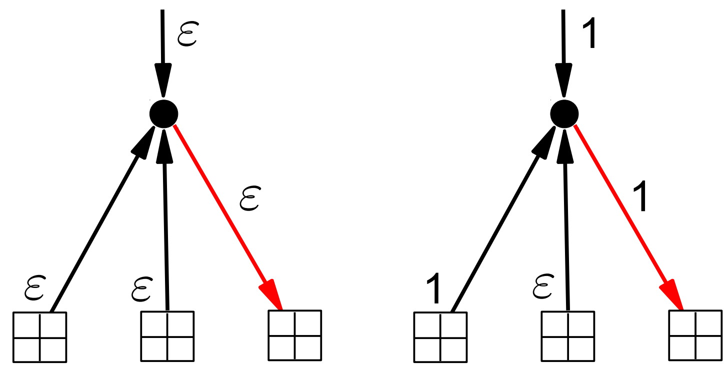
Belief Propagation - BEC

- Iterative message passing decoding, where the messages represent **probabilities**, or **beliefs**, is known as **belief propagation (BP)** decoding
- For the BEC, BP decoding simplifies to the following two rules:

➡ At a **check node**, the outgoing message is an erasure if **any** of the incoming messages are erasures; otherwise the message is the mod-2 sum of the incoming messages.



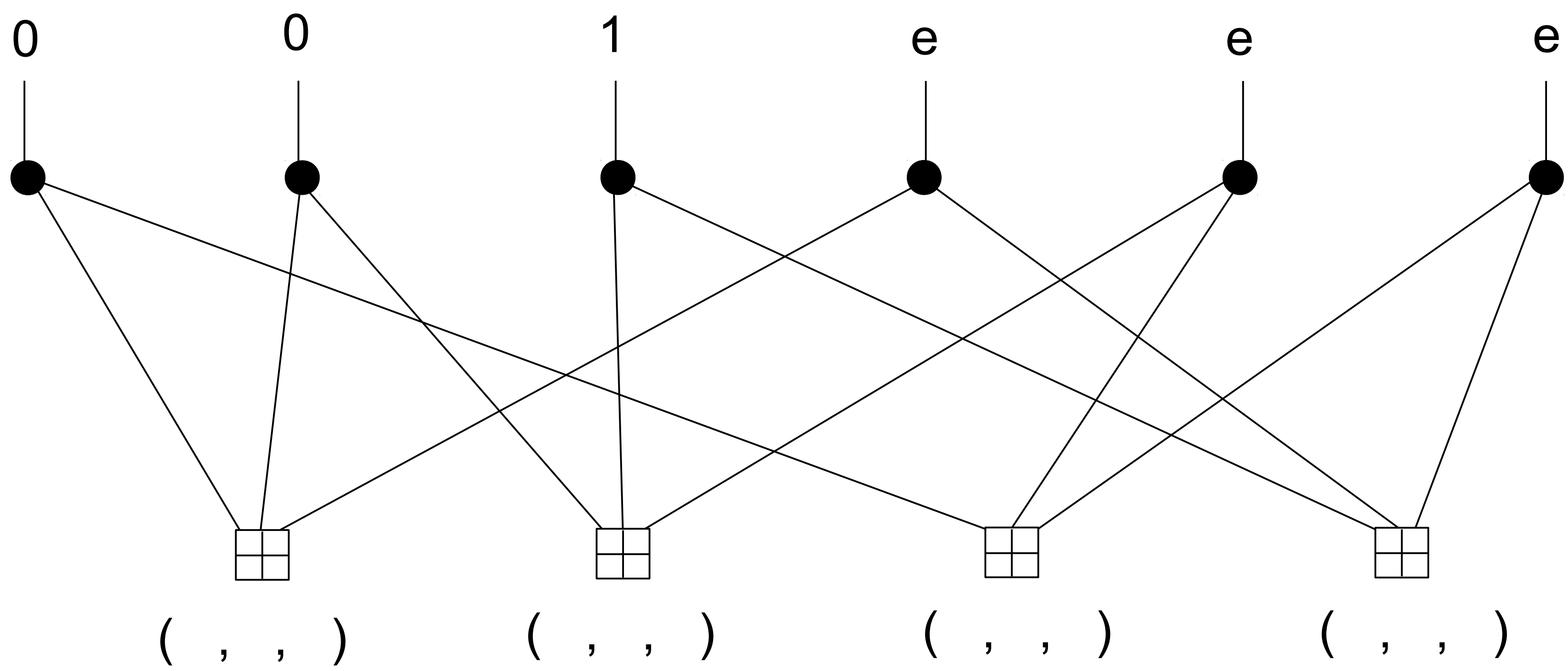
➡ At a **variable node**, the outgoing message is an erasure if **all** of the incoming messages are erasures; otherwise the message is the value of all non-erasures, which must agree.



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [\,0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\,] \rightarrow \mathbf{r} = [\,0\ \ 0\ \ 1\ \ e\ \ e\ \ e\,]$

received (corrupted) codeword

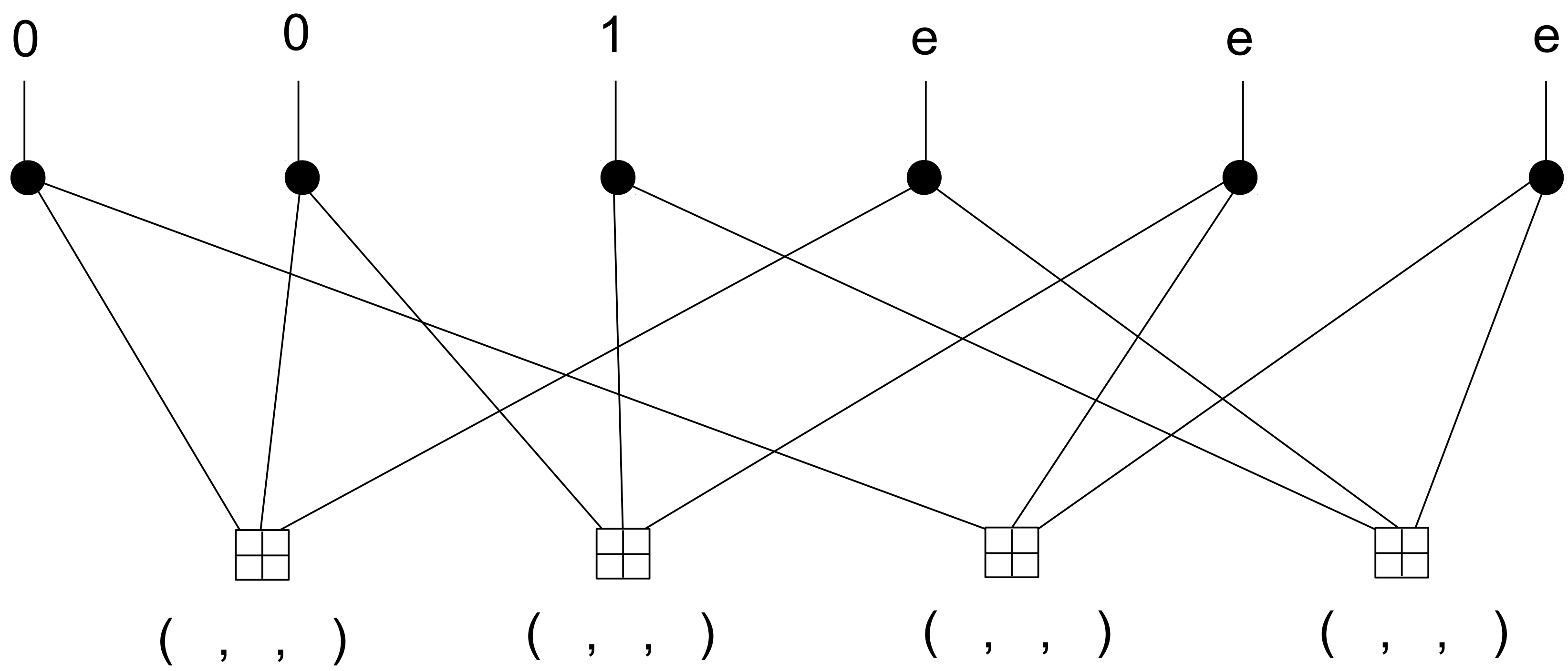


Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [\,0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\,] \rightarrow \mathbf{r} = [\,0\ \ 0\ \ 1\ \ e\ \ e\ \ e\,]$

Iteration 1: bit-to-check

received (corrupted) codeword

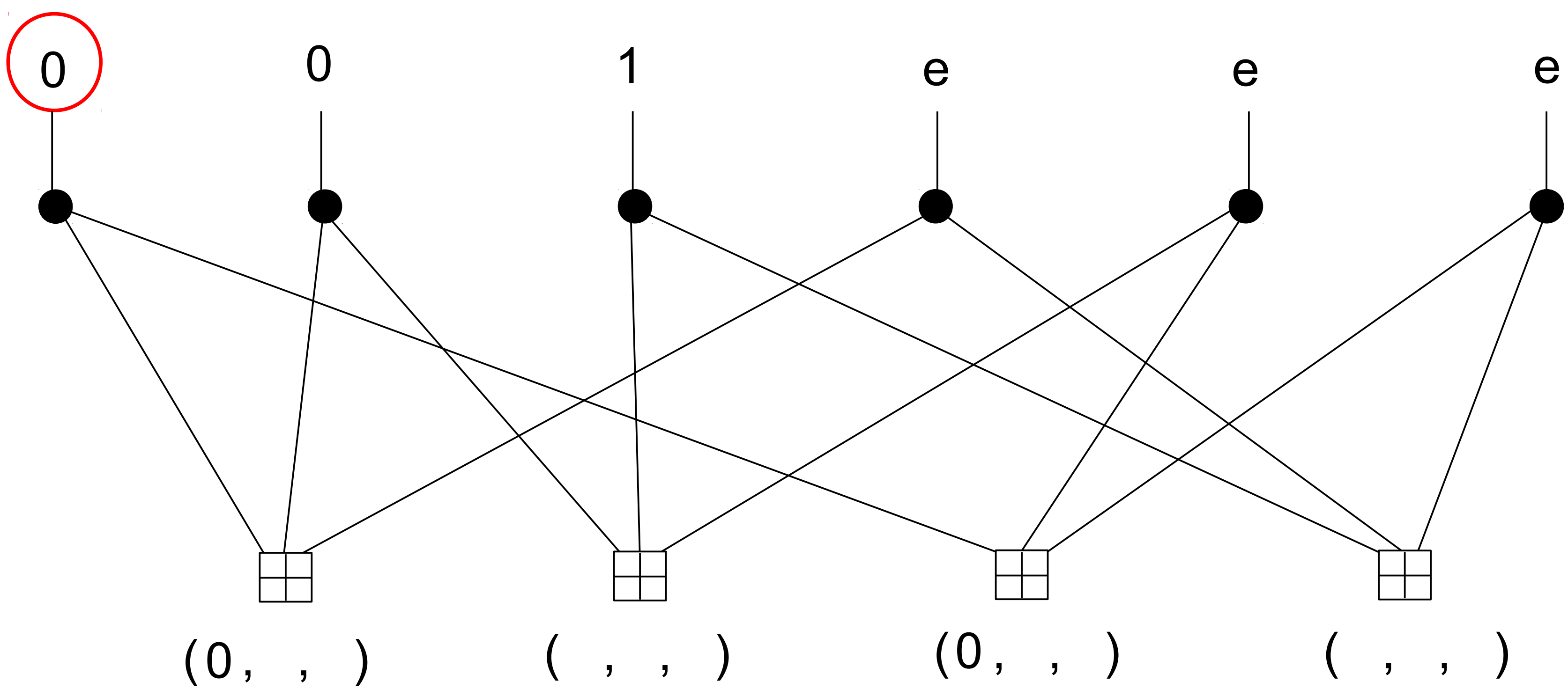


Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [\,0\;\;0\;\;1\;\;0\;\;1\;\;1\,] \rightarrow \mathbf{r} = [\,0\;\;0\;\;1\;\;e\;\;e\;\;e\,]$

Iteration 1: bit-to-check

received (corrupted) codeword



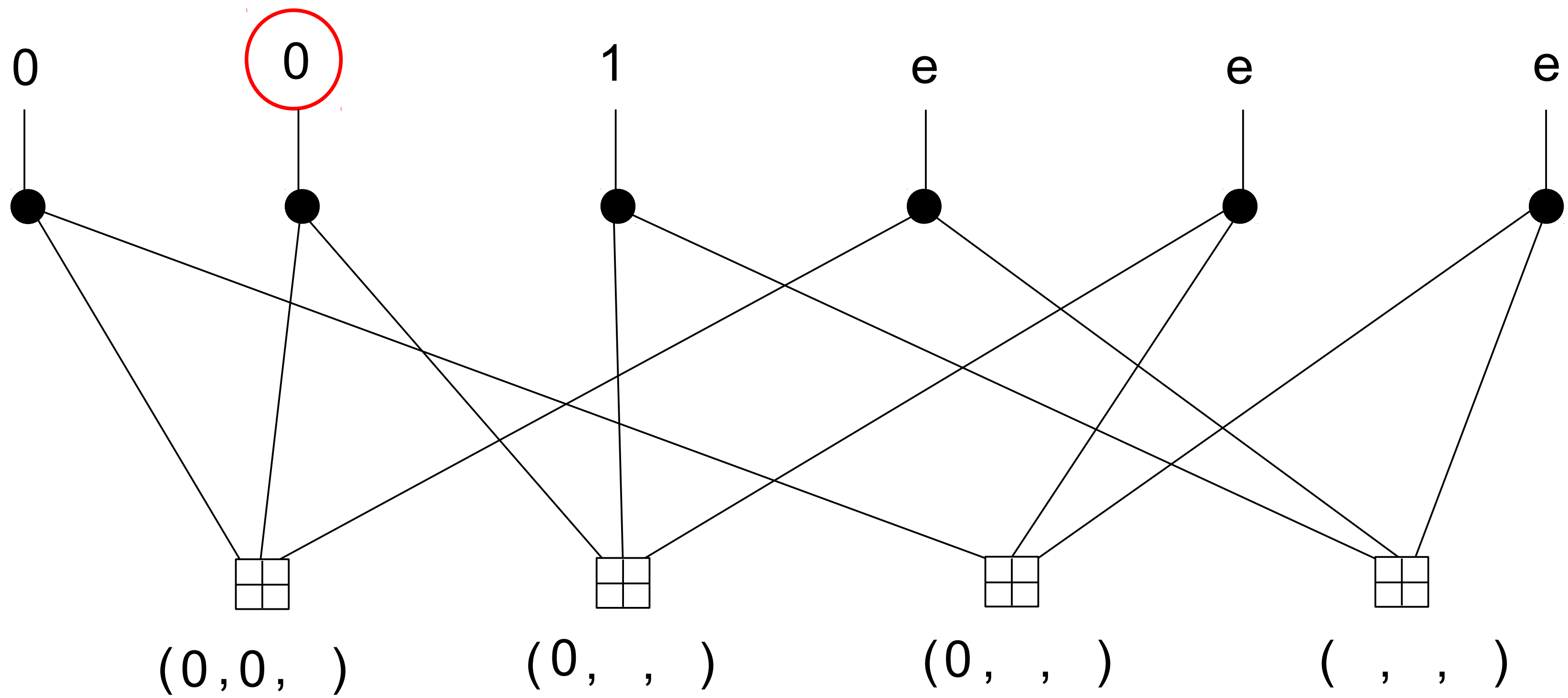
Iterative Decoding Example (BEC)

UNIVERSITY OF NOTRE DAME

Ex: $\mathbf{v} = [0 \ 0 \ 1 \ 0 \ 1 \ 1] \rightarrow \mathbf{r} = [0 \ 0 \ 1 \ e \ e \ e]$

Iteration 1: bit-to-check

received (corrupted) codeword

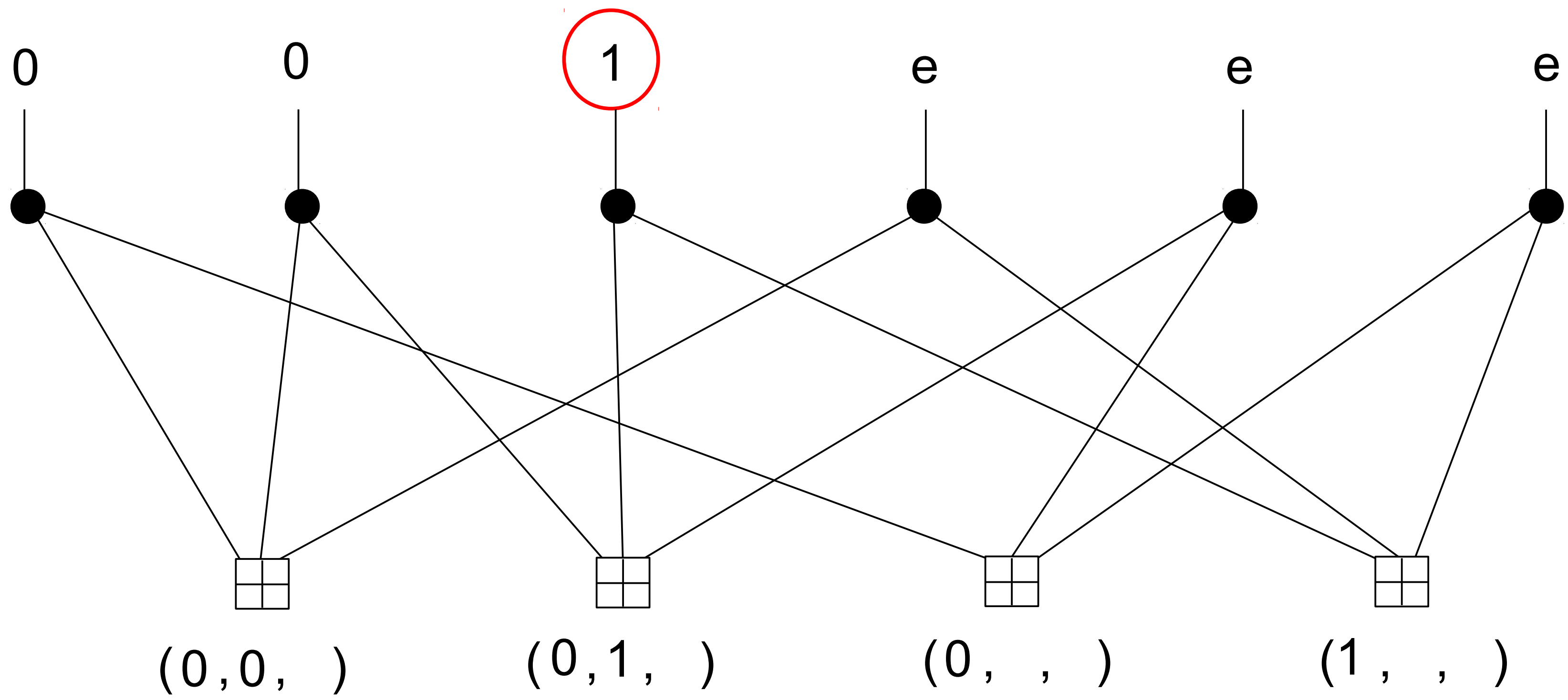


Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

Iteration 1: bit-to-check

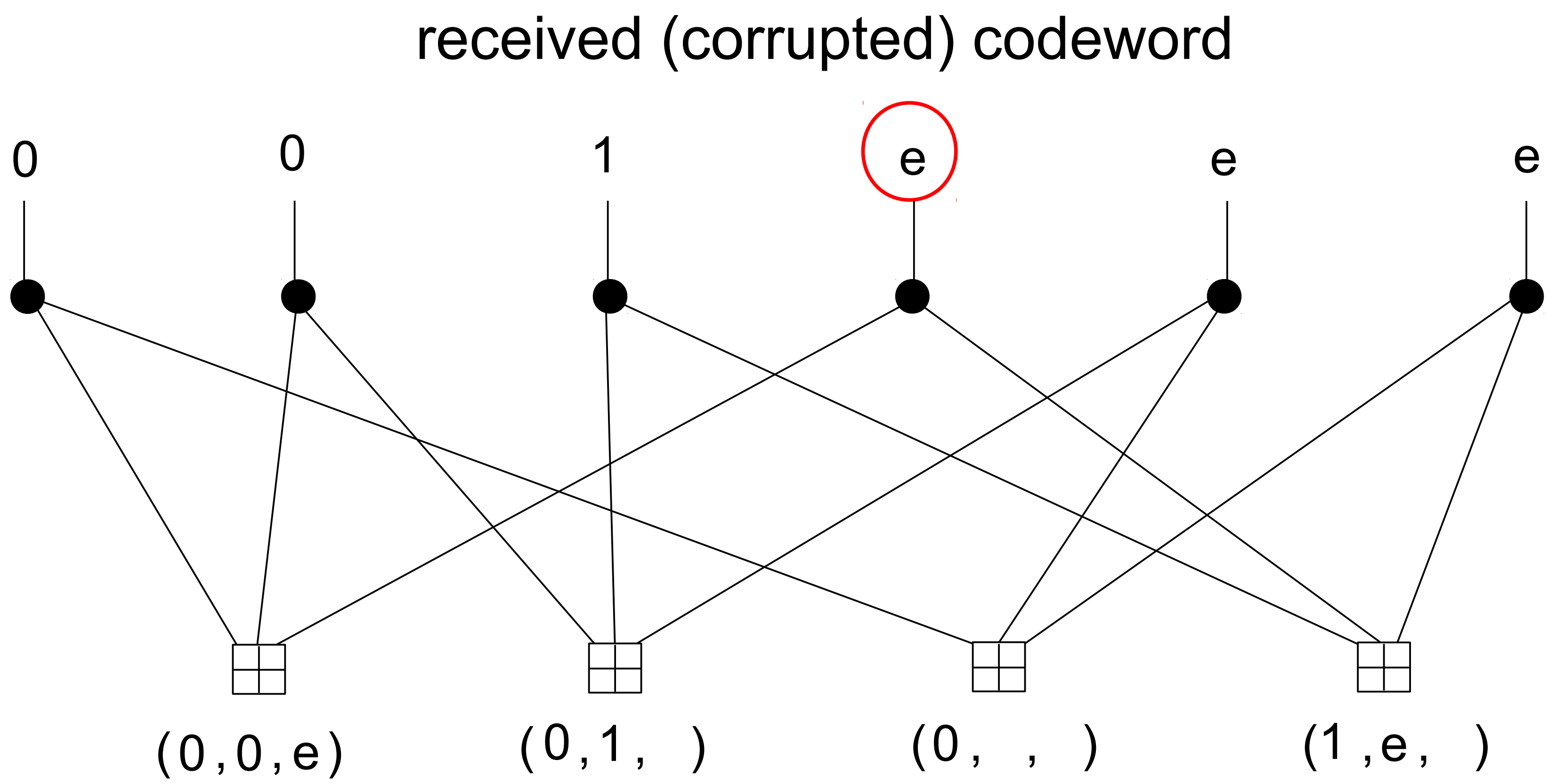
received (corrupted) codeword



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [\,0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\,] \rightarrow \mathbf{r} = [\,0\ \ 0\ \ 1\ \ e\ \ e\ \ e\,]$

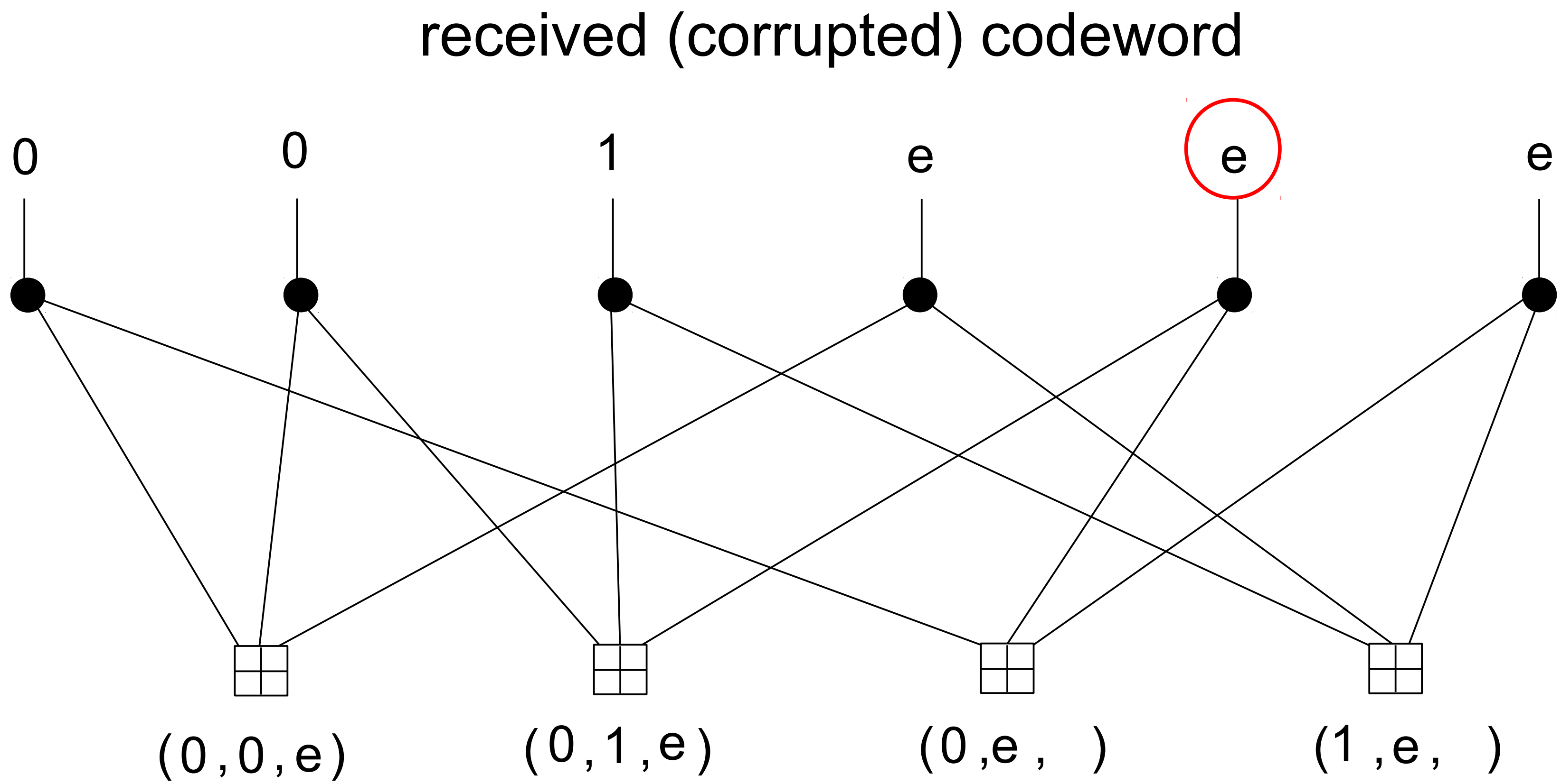
Iteration 1: bit-to-check



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

Iteration 1: bit-to-check



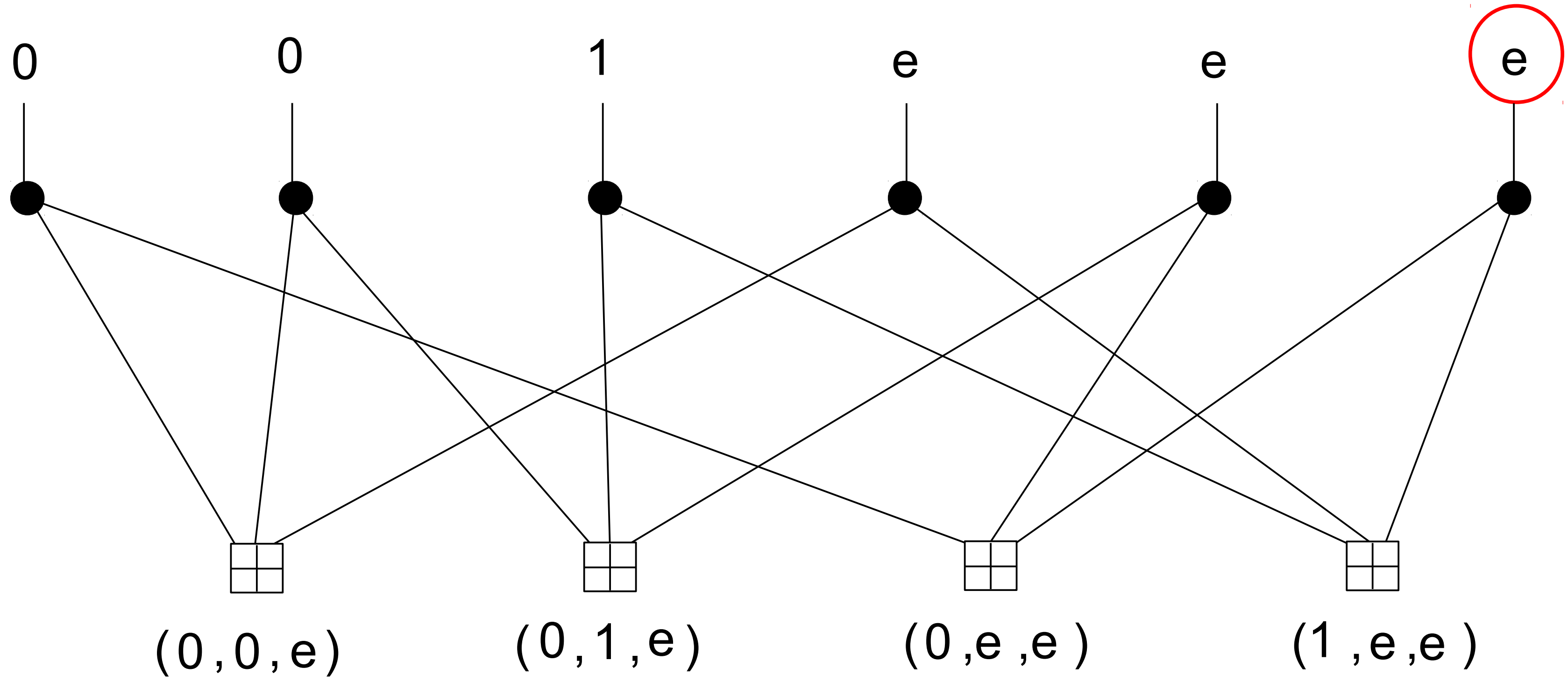
Iterative Decoding Example (BEC)

UNIVERSITY OF NOTRE DAME

Ex: $\mathbf{v} = [\,0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\,] \rightarrow \mathbf{r} = [\,0\ \ 0\ \ 1\ \ e\ \ e\ \ e\,]$

Iteration 1: bit-to-check

received (corrupted) codeword



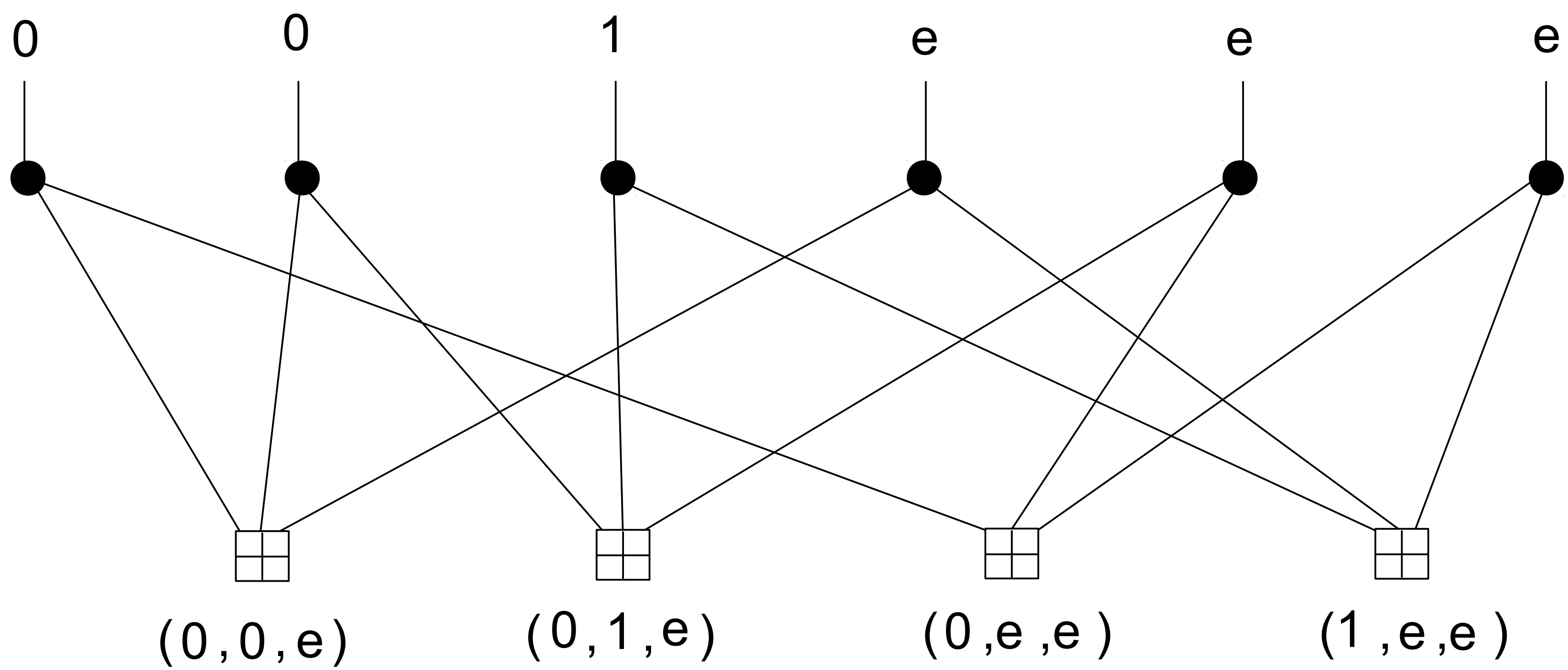
Iterative Decoding Example (BEC)

UNIVERSITY OF NOTRE DAME

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

Iteration 1: bit-to-check

received (corrupted) codeword

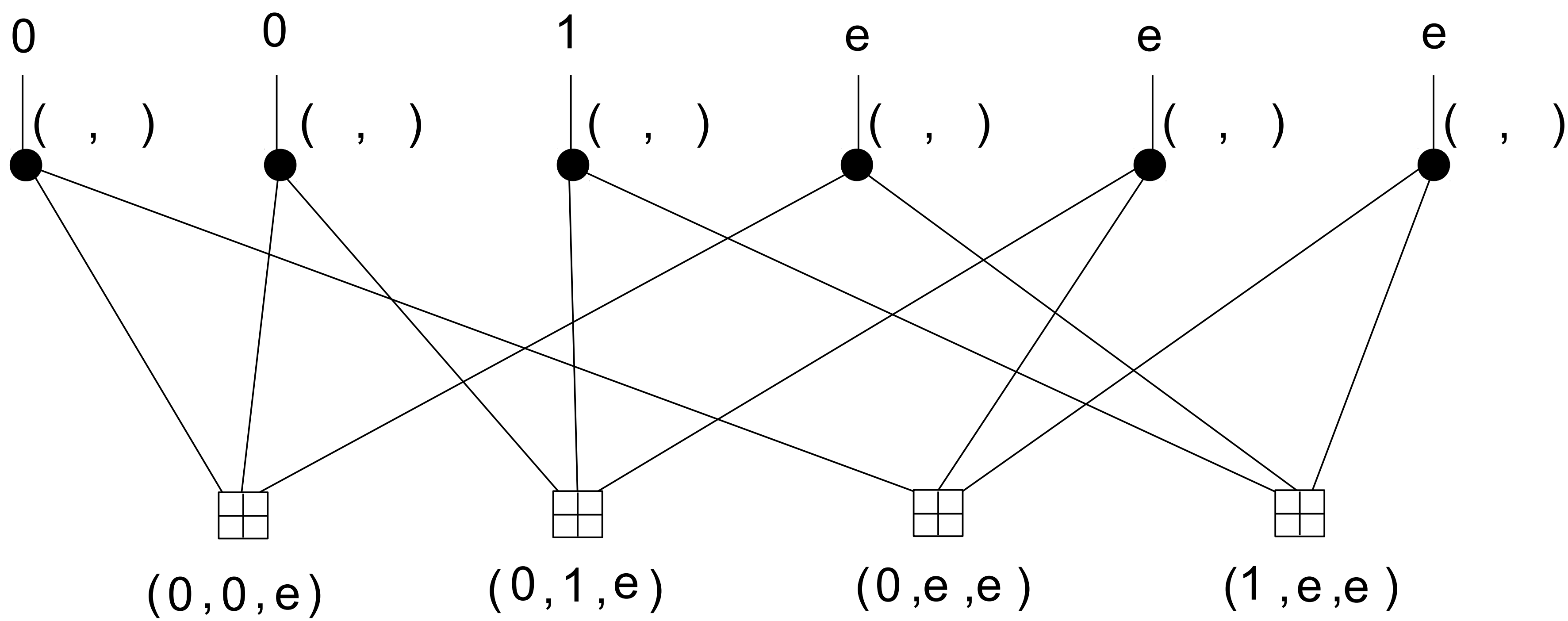


Iterative Decoding Example (BEC)

UNIVERSITY OF NOTRE DAME

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

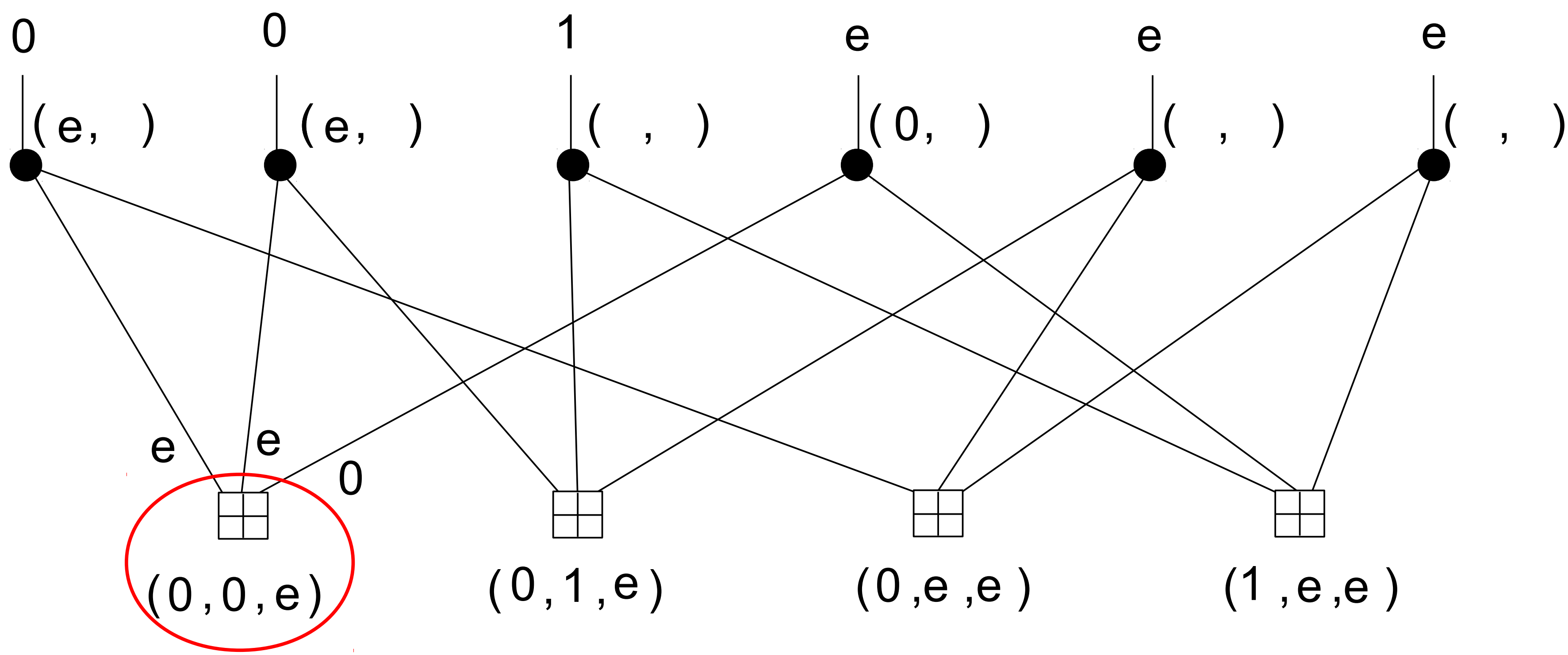
Iteration 1: check-to-bit



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

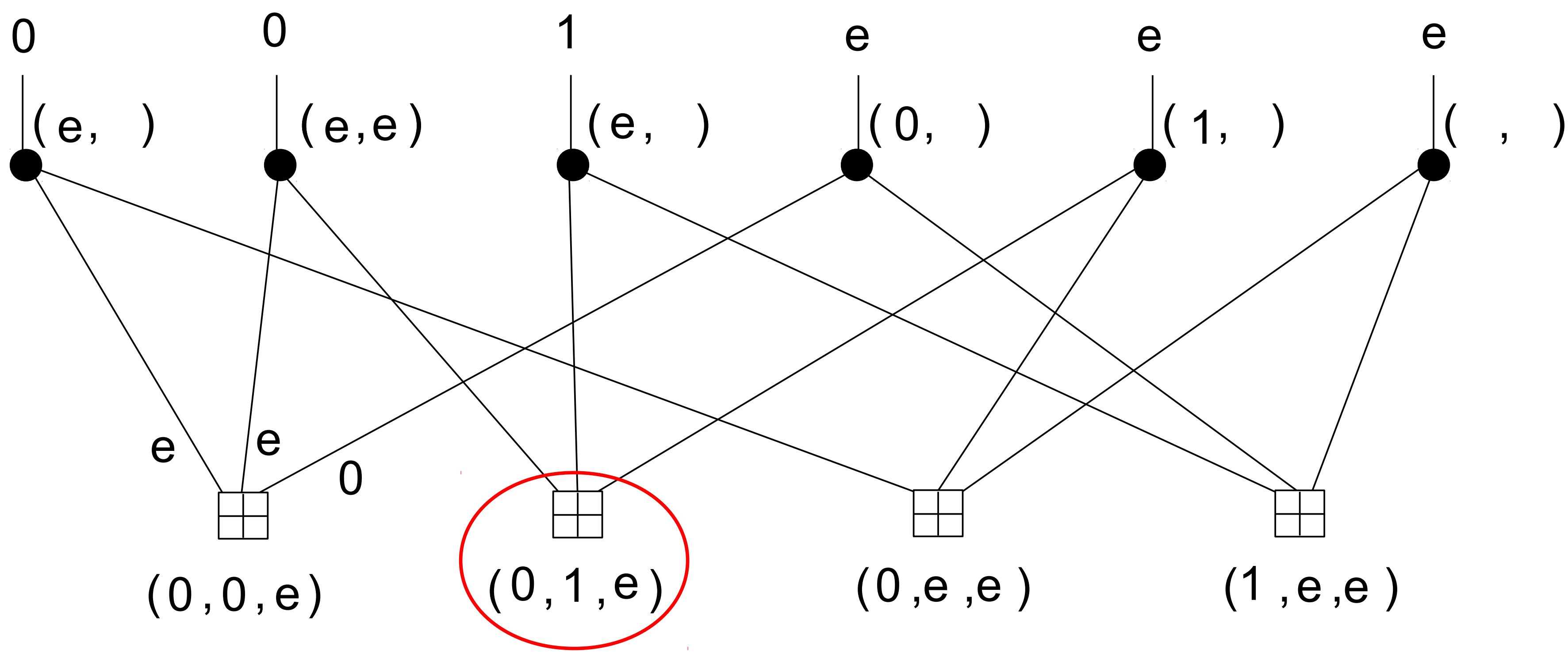
Iteration 1: check-to-bit



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

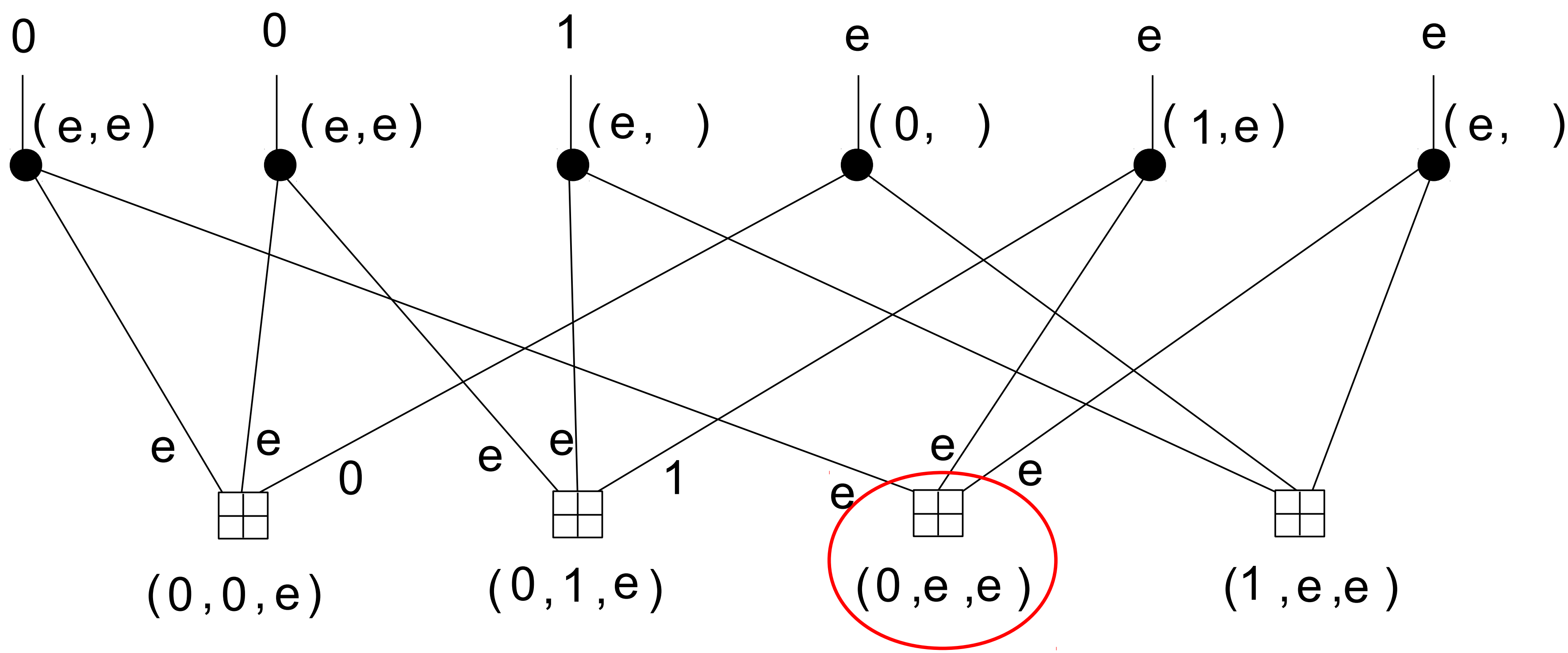
Iteration 1: check-to-bit



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

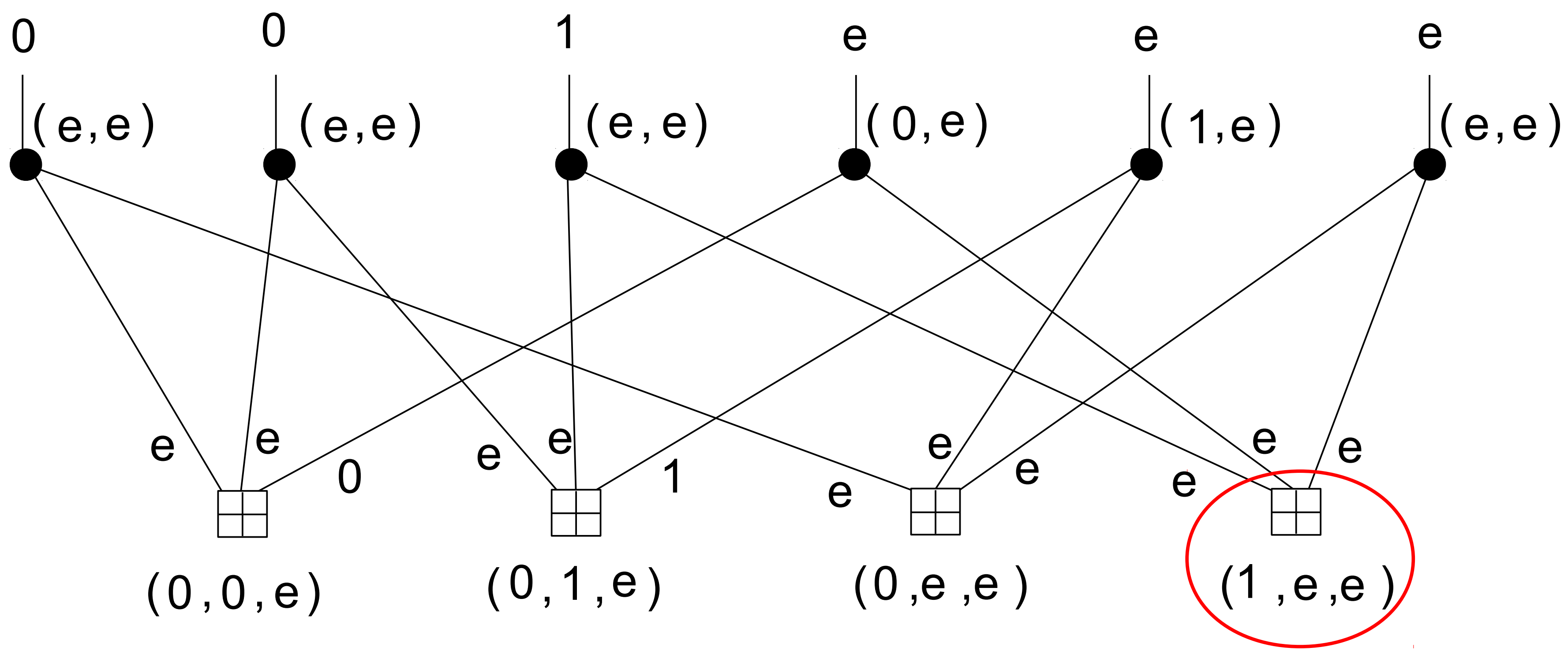
Iteration 1: check-to-bit



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

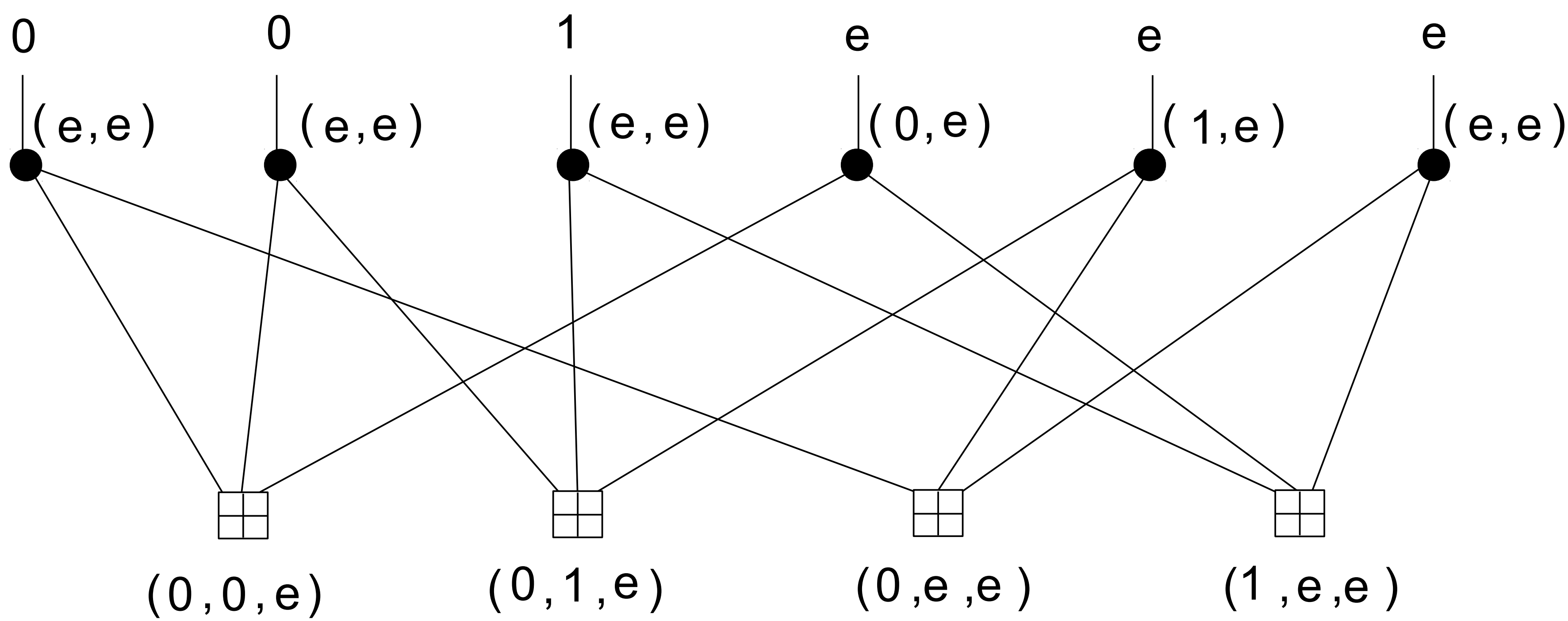
Iteration 1: check-to-bit



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

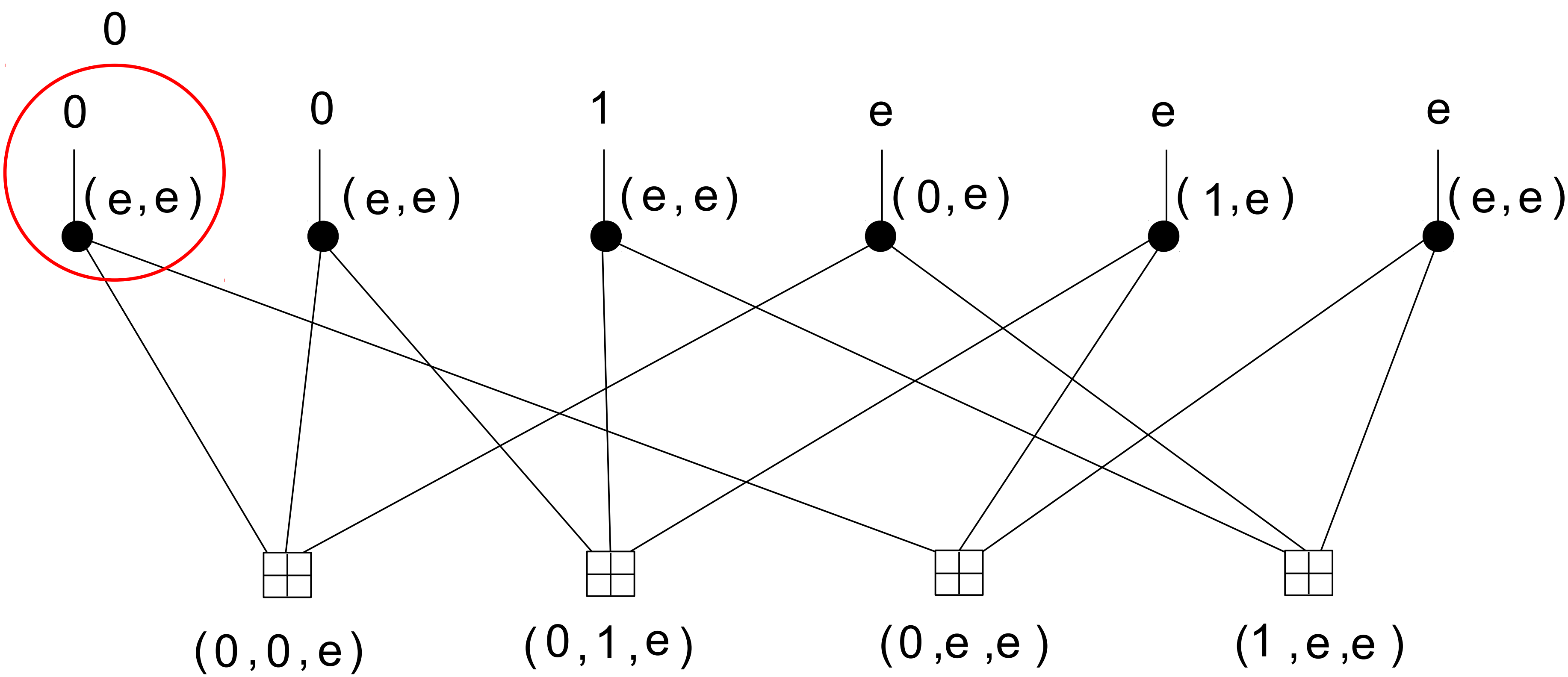
Iteration 1: bit processing



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

Iteration 1: bit processing

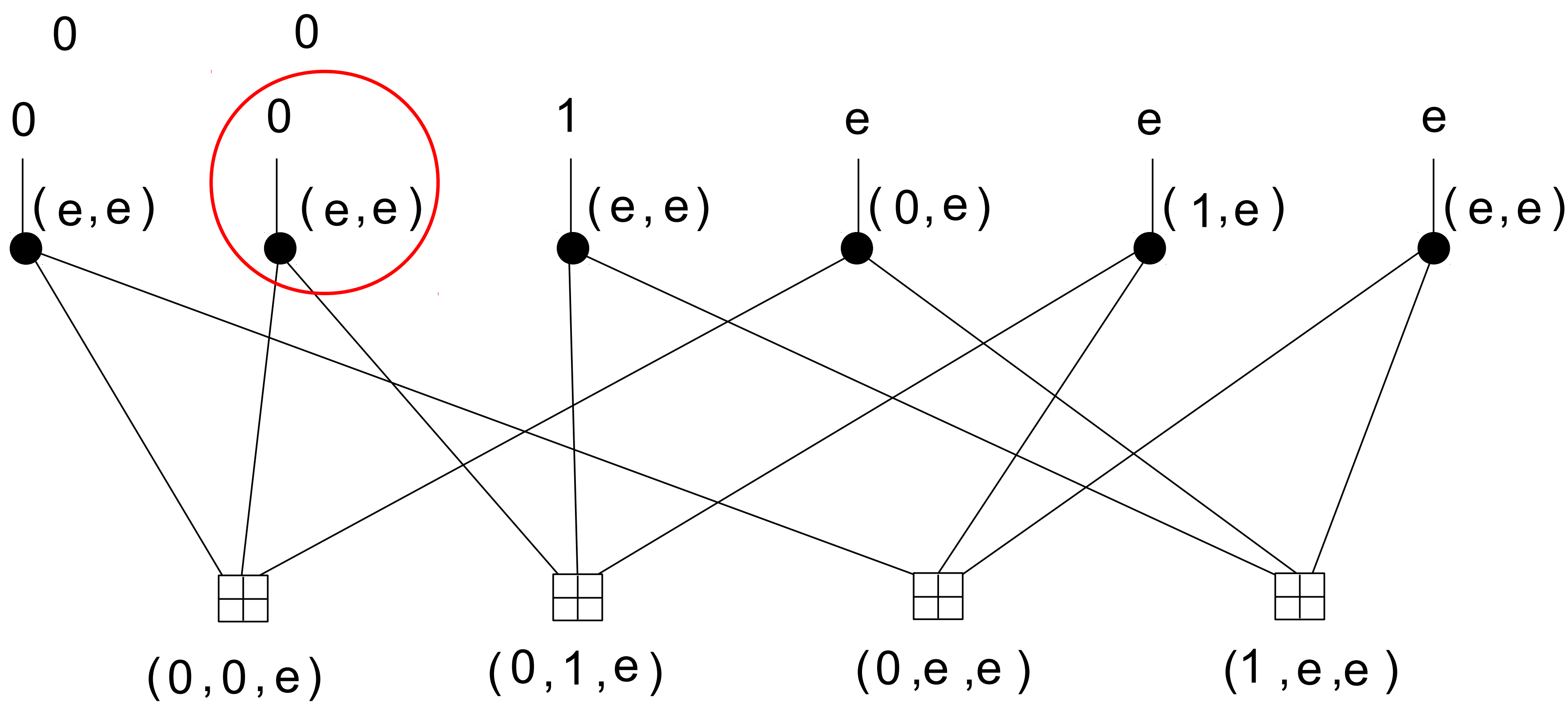


Iterative Decoding Example (BEC)

UNIVERSITY OF NOTRE DAME

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

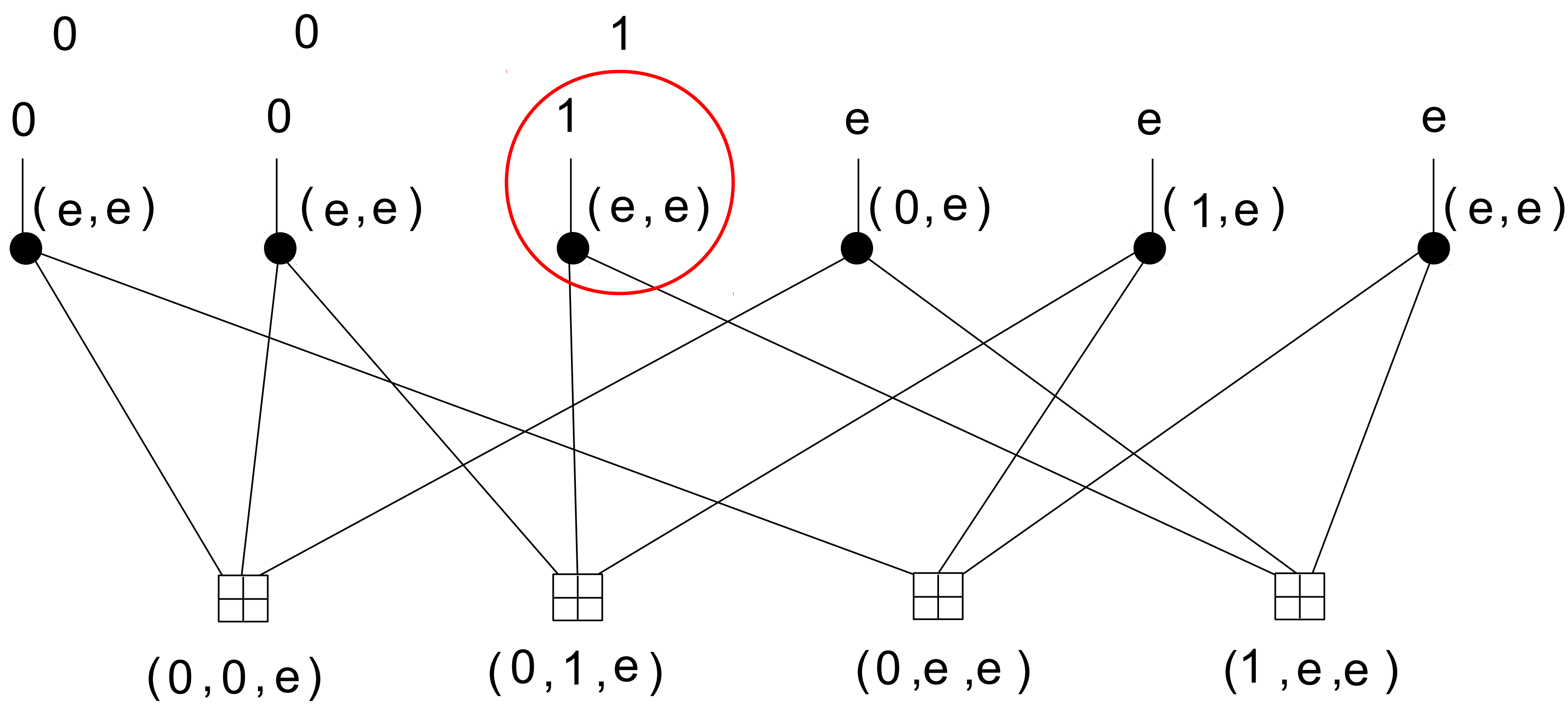
Iteration 1: bit processing



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

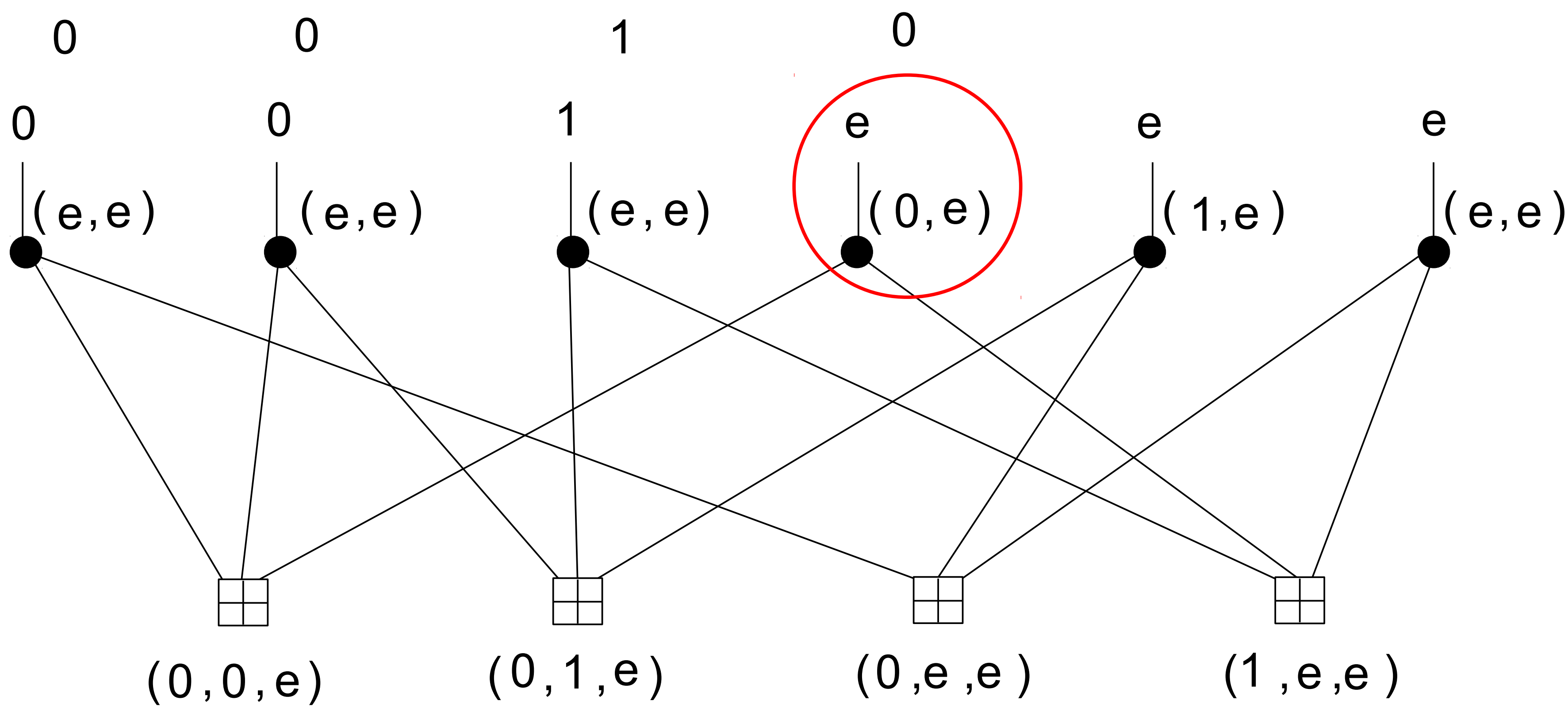
Iteration 1: bit processing



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

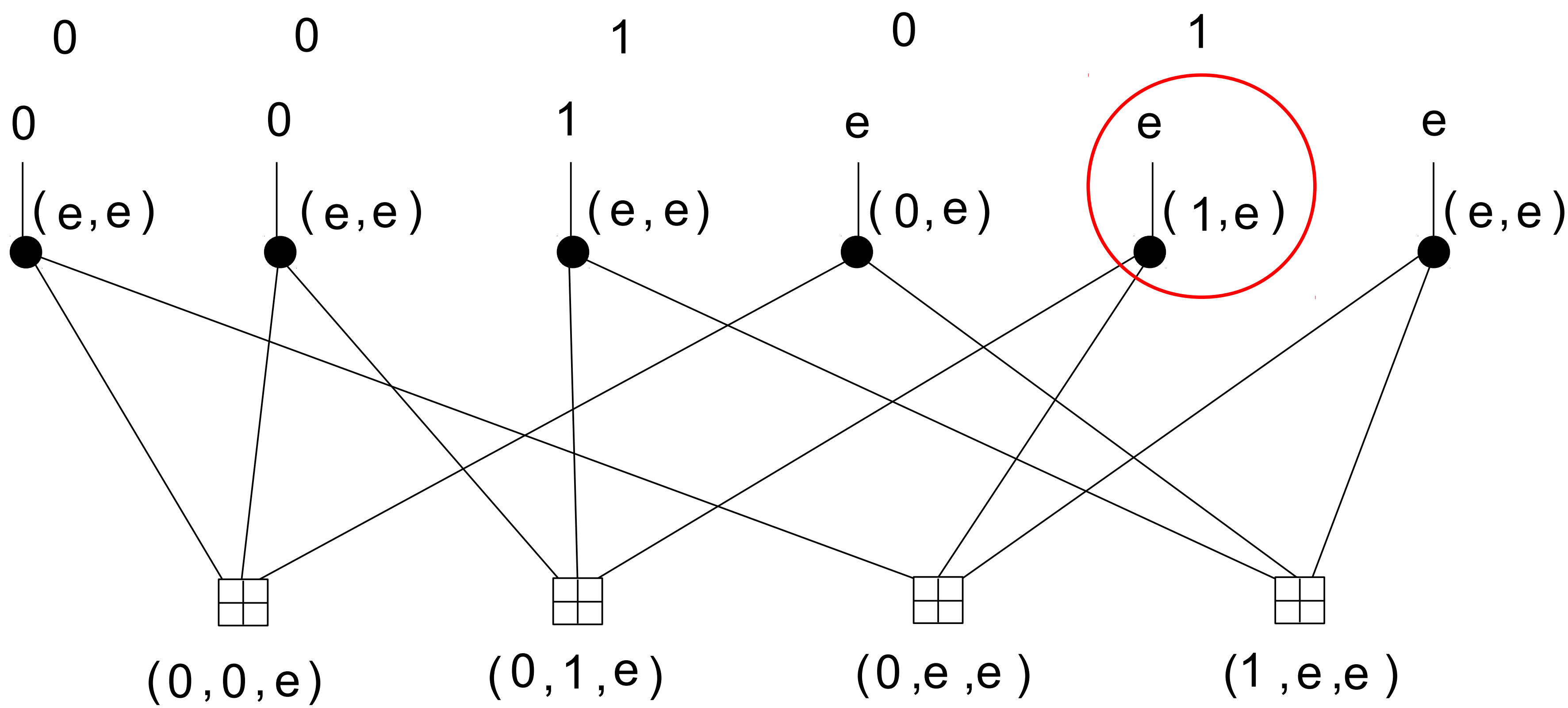
Iteration 1: bit processing



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

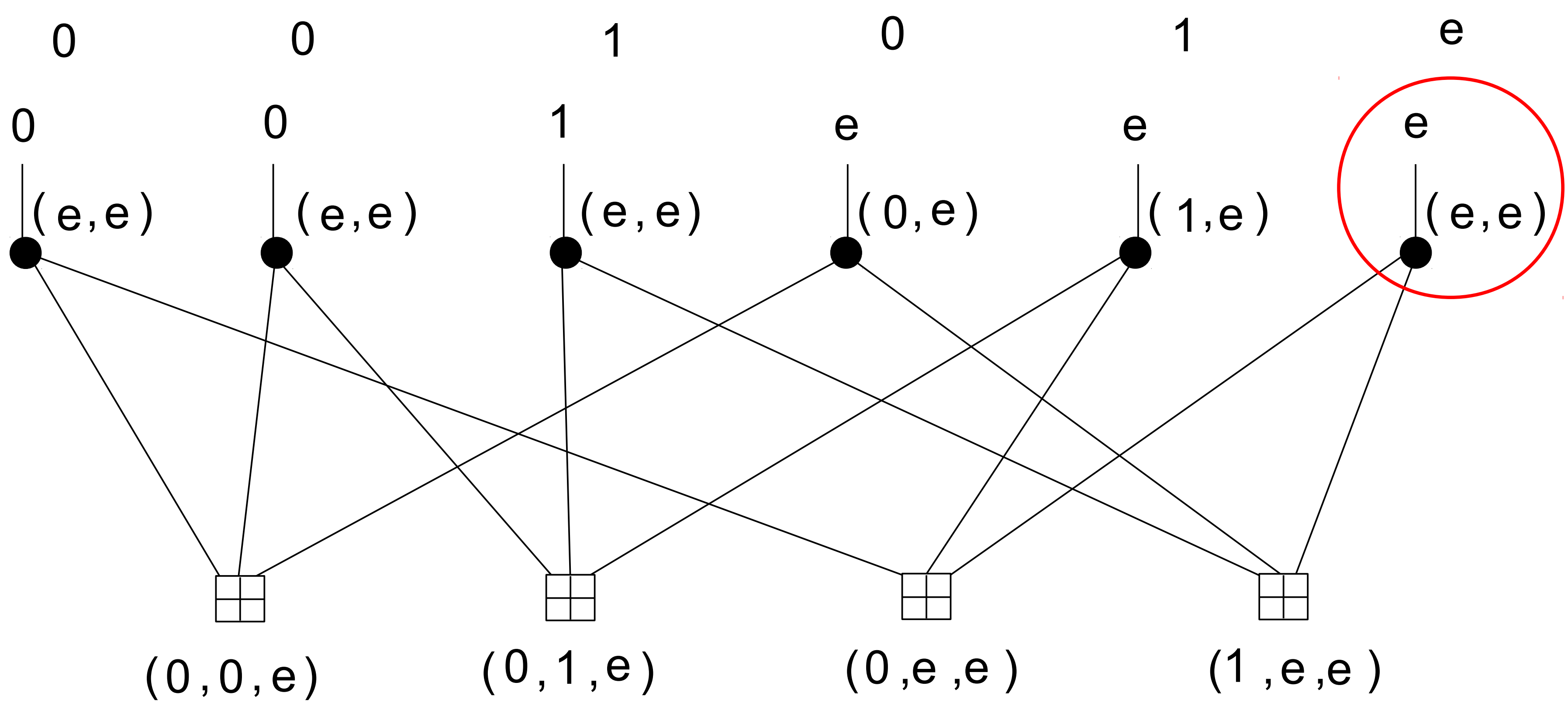
Iteration 1: bit processing



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

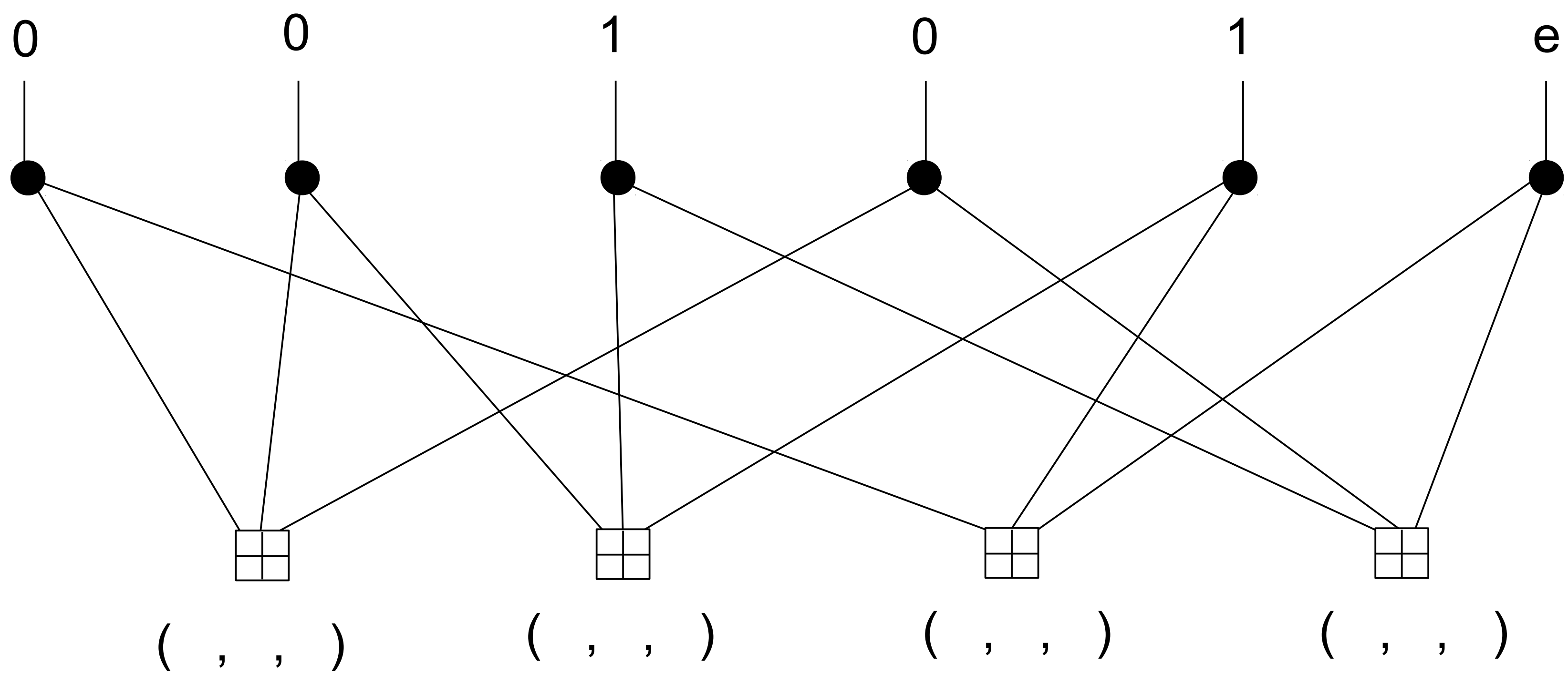
Iteration 1: bit processing



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [\,0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\,] \rightarrow \mathbf{r} = [\,0\ \ 0\ \ 1\ \ e\ \ e\ \ e\,]$

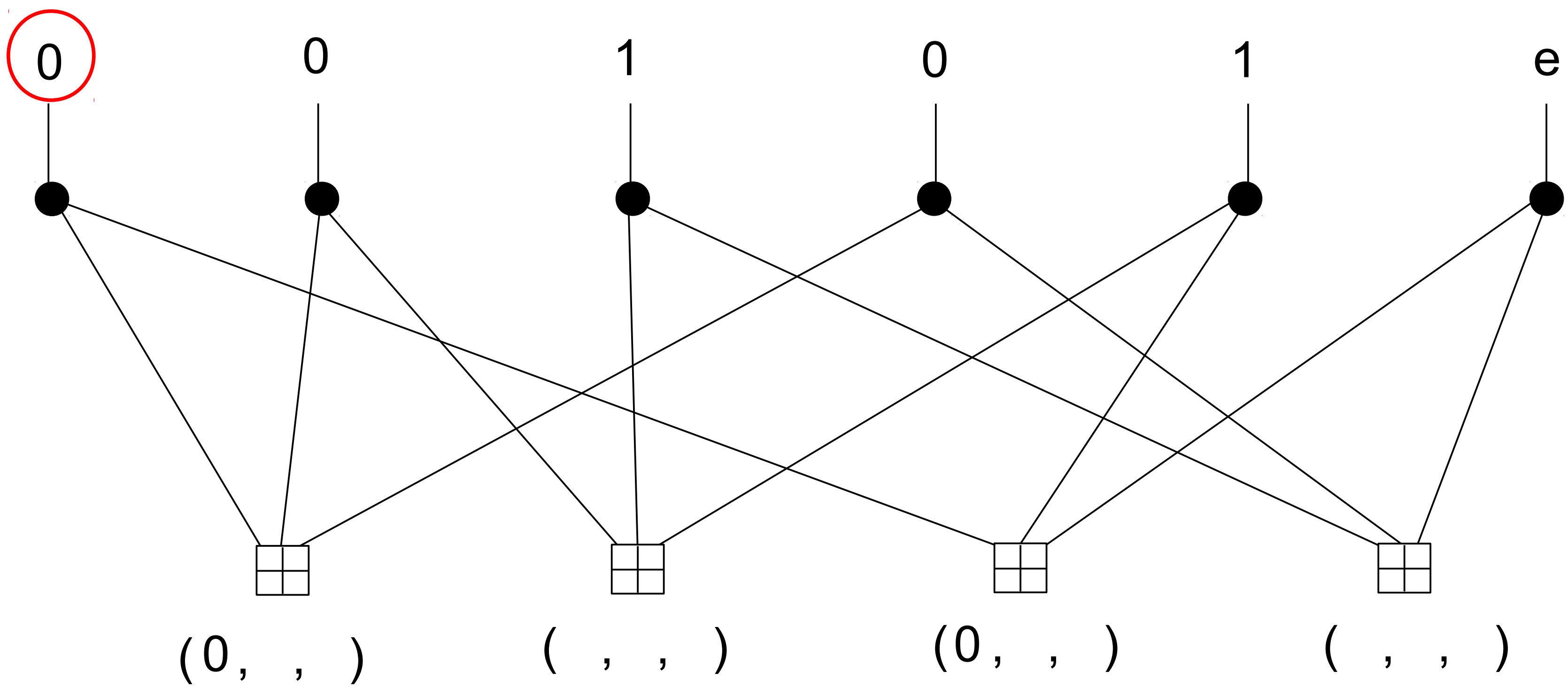
Iteration 2: bit-to-check



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [\,0\;\;0\;\;1\;\;0\;\;1\;\;1\,] \rightarrow \mathbf{r} = [\,0\;\;0\;\;1\;\;e\;\;e\;\;e\,]$

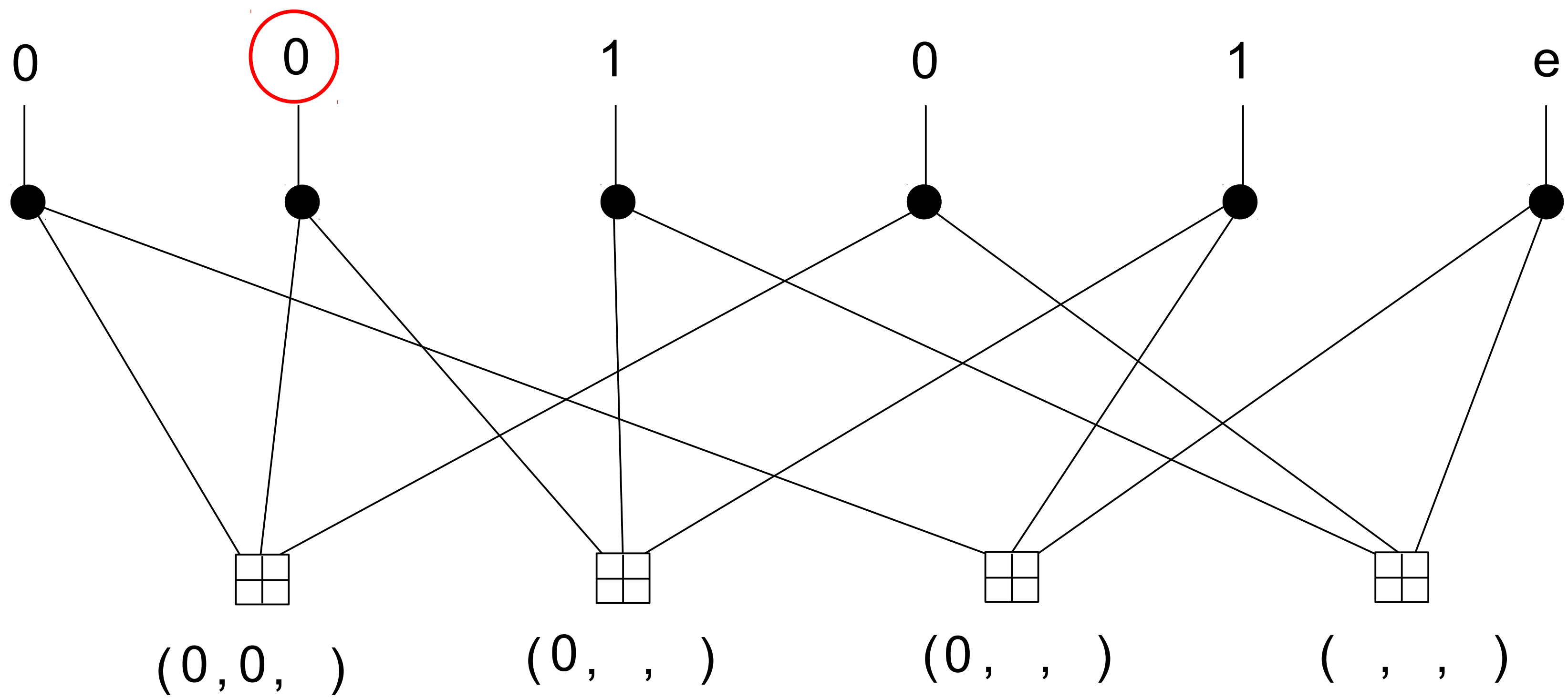
Iteration 2: bit-to-check



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [\,0\;\;0\;\;1\;\;0\;\;1\;\;1\,] \rightarrow \mathbf{r} = [\,0\;\;0\;\;1\;\;e\;\;e\;\;e\,]$

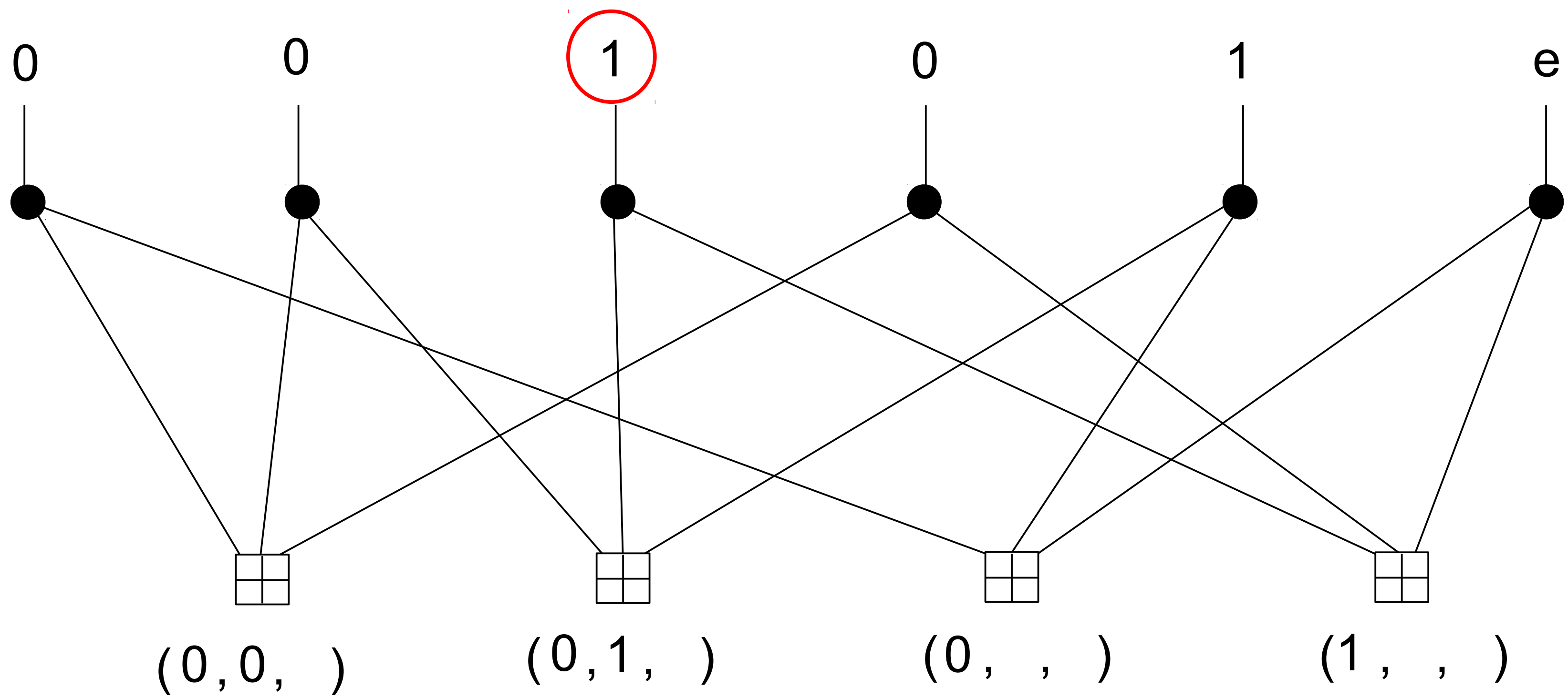
Iteration 2: bit-to-check



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

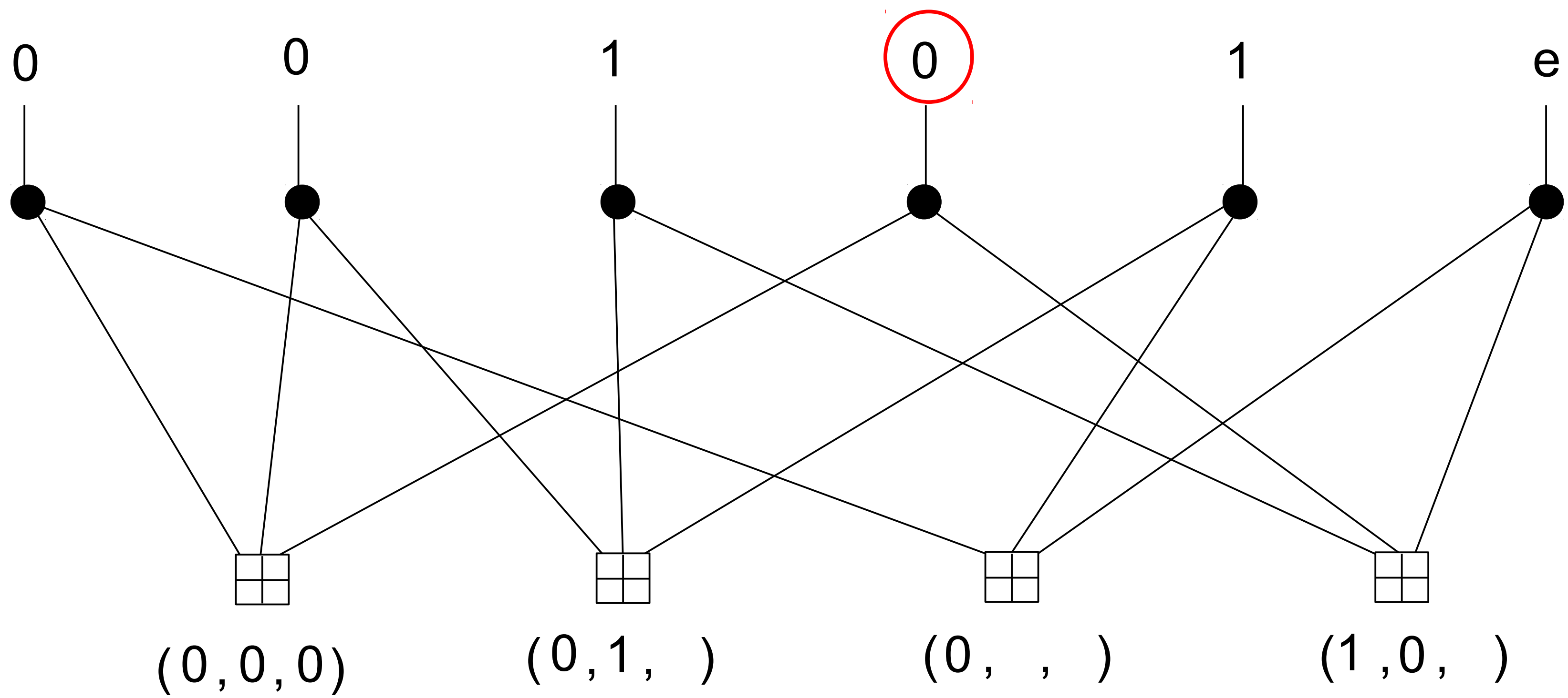
Iteration 2: bit-to-check



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

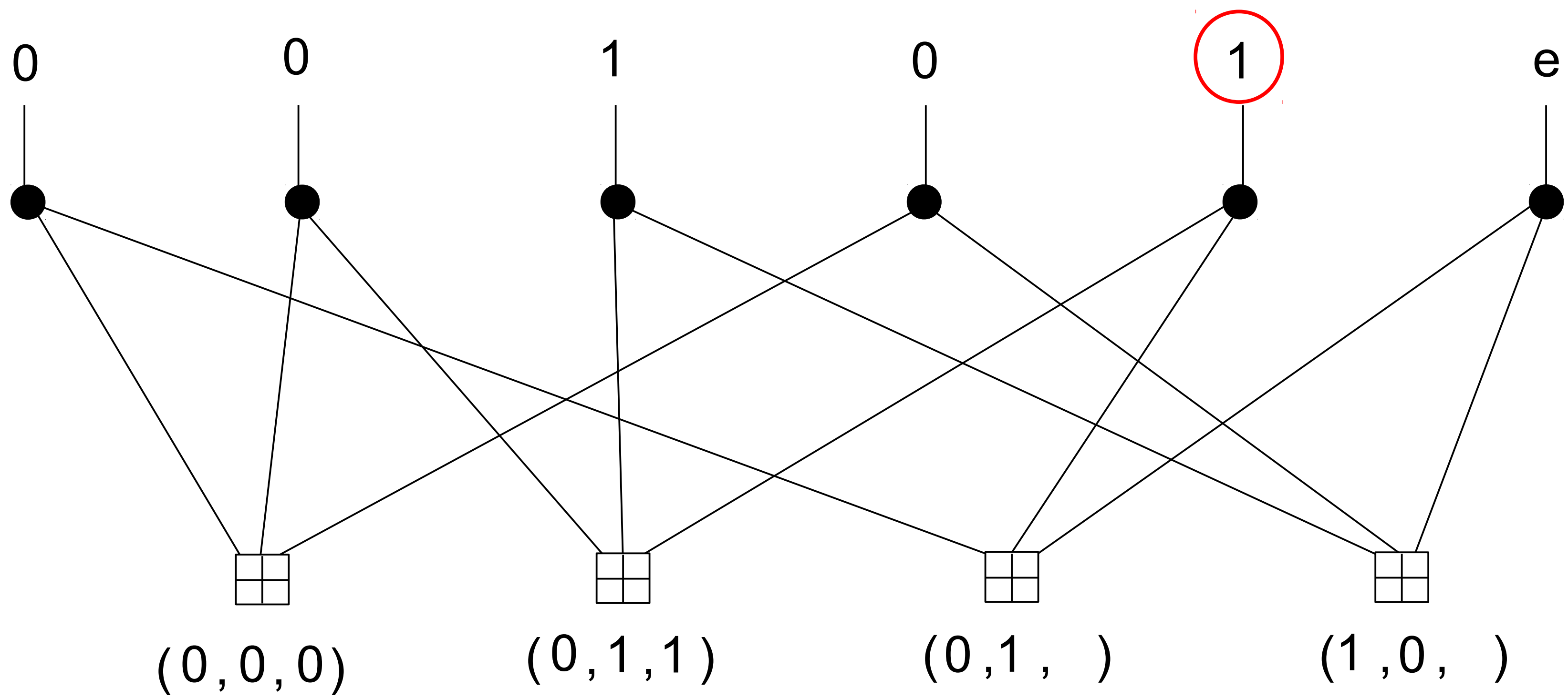
Iteration 2: bit-to-check



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

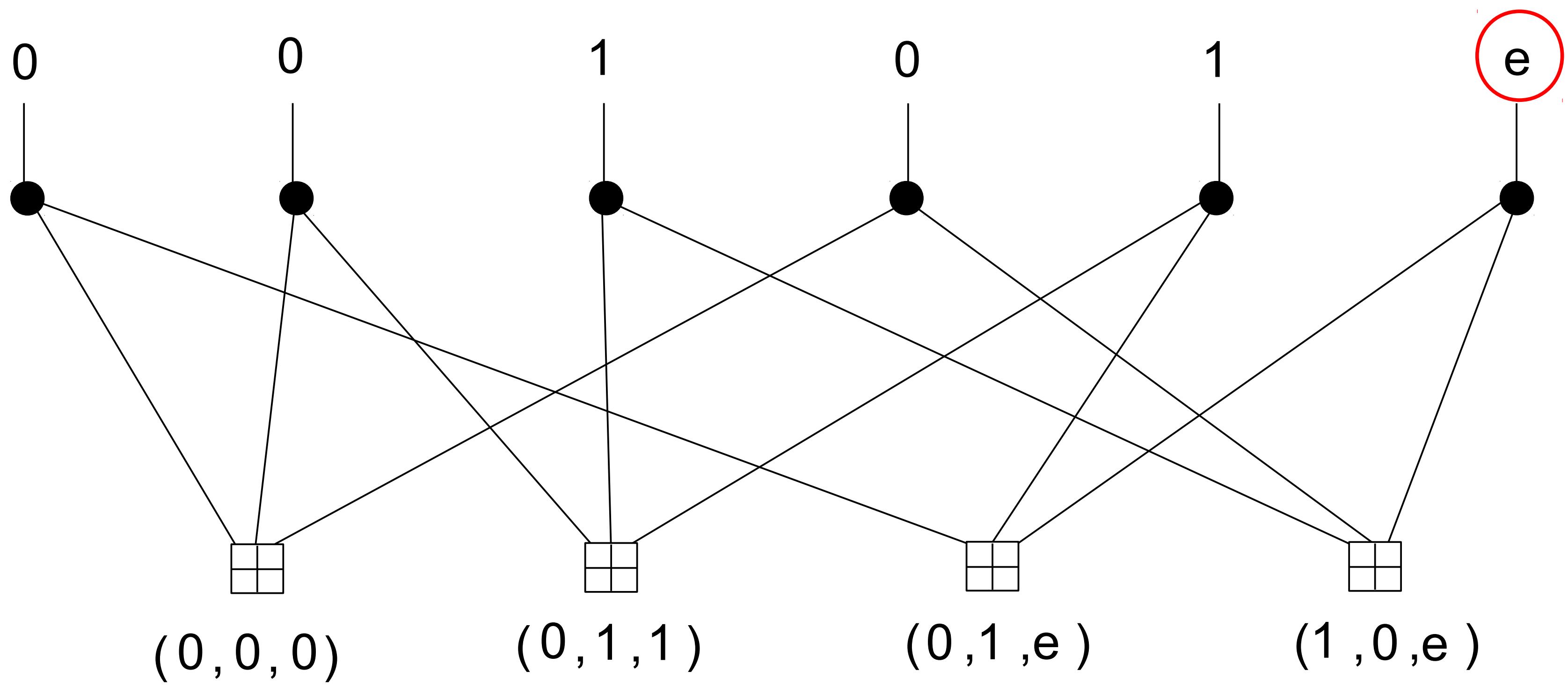
Iteration 2: bit-to-check



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [\,0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\,] \rightarrow \mathbf{r} = [\,0\ \ 0\ \ 1\ \ e\ \ e\ \ e\,]$

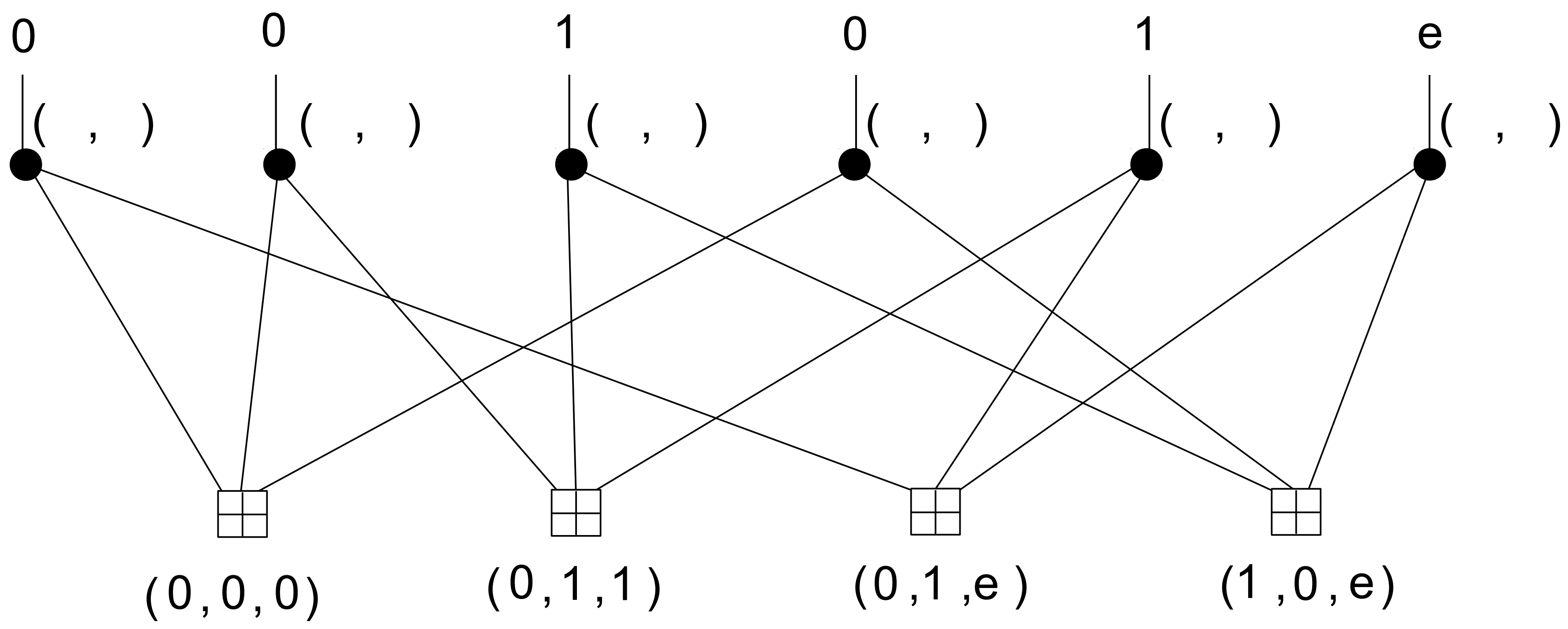
Iteration 2: bit-to-check



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

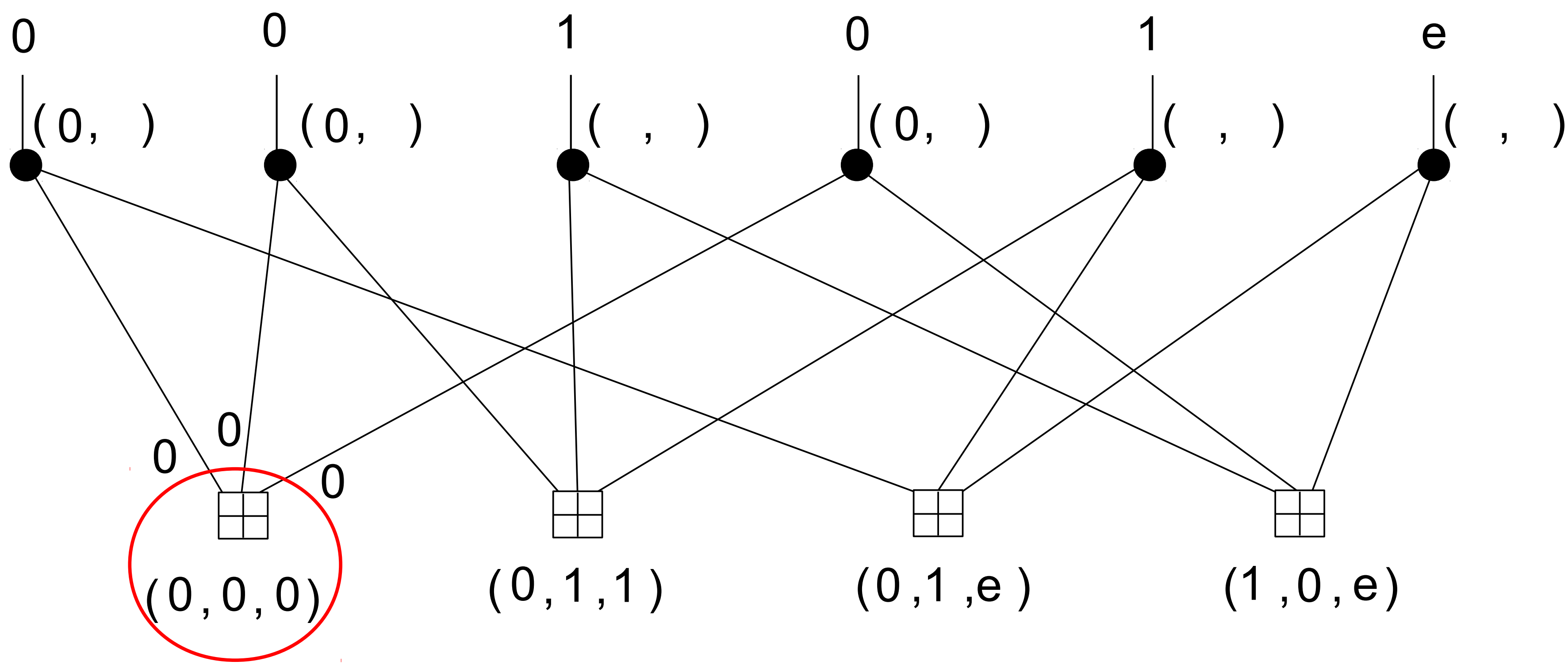
Iteration 2: check-to-bit



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

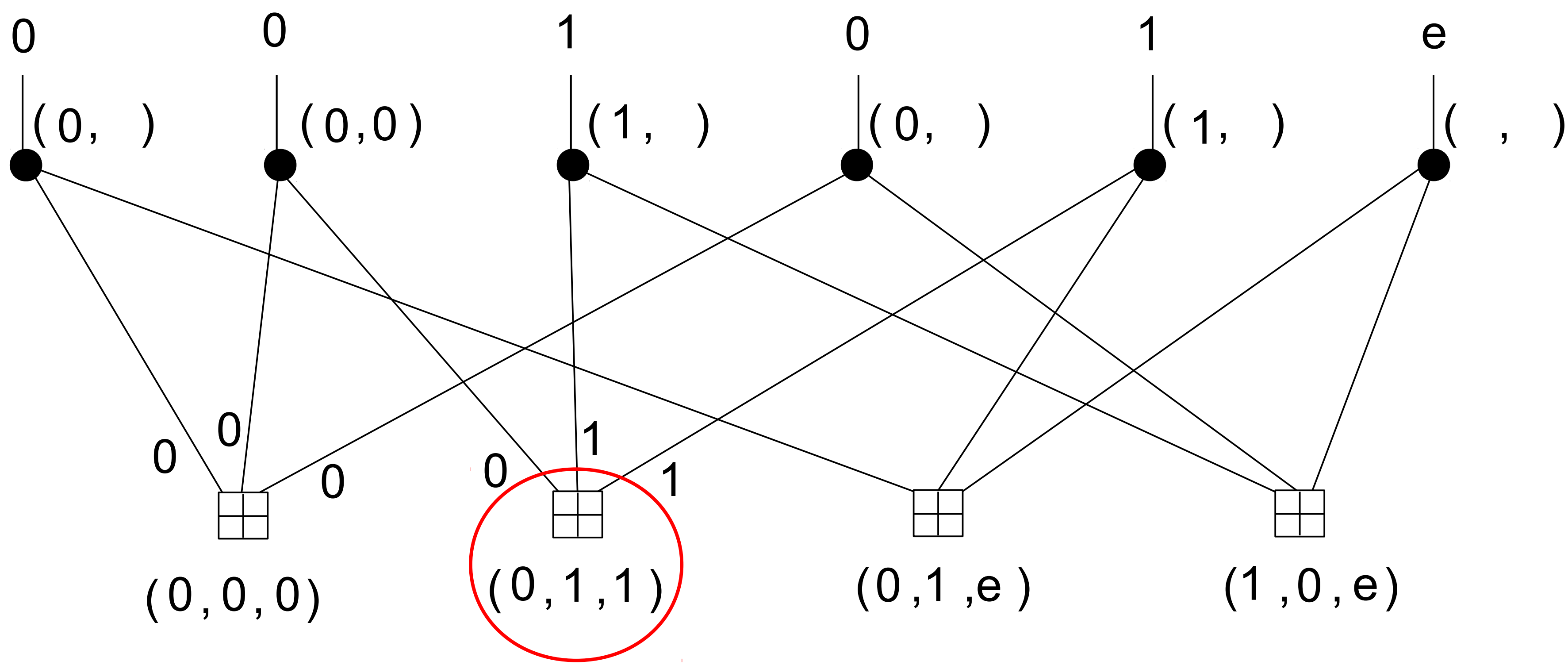
Iteration 2: check-to-bit



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

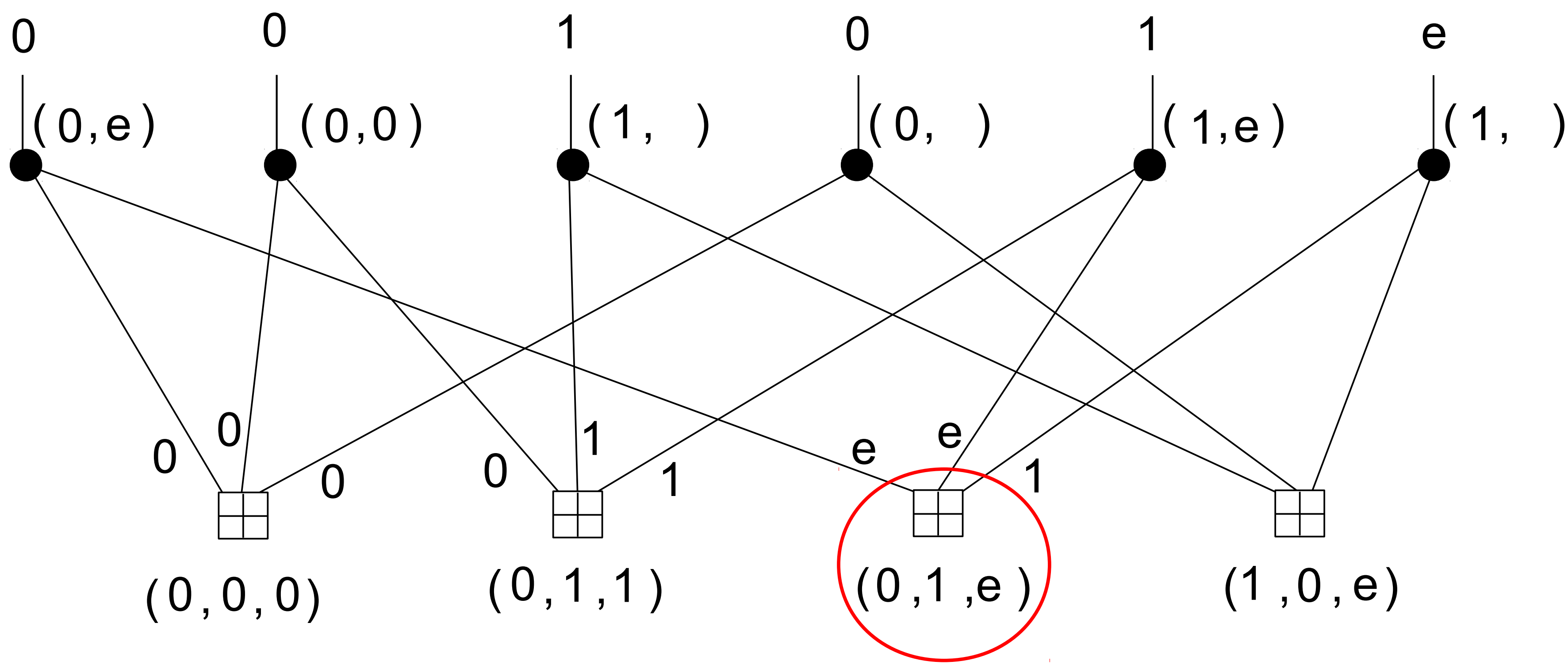
Iteration 2: check-to-bit



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

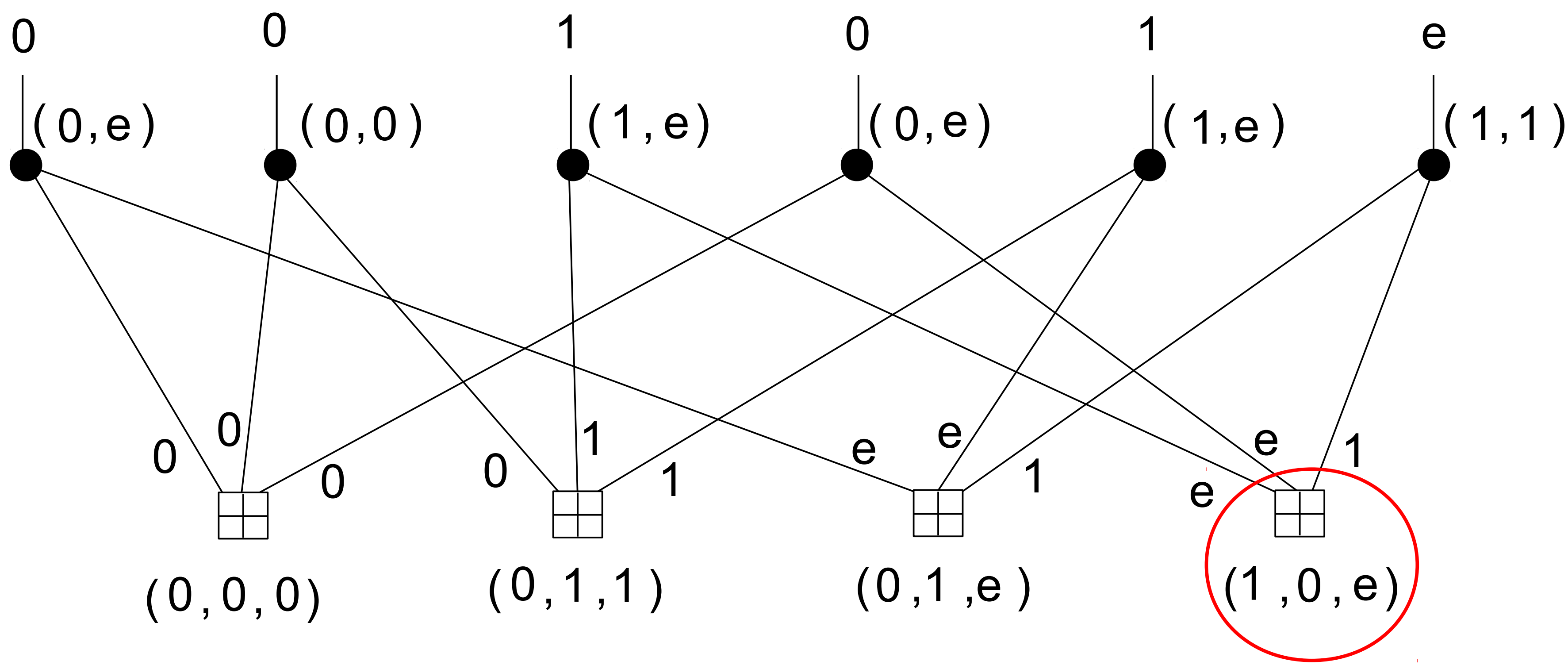
Iteration 2: check-to-bit



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

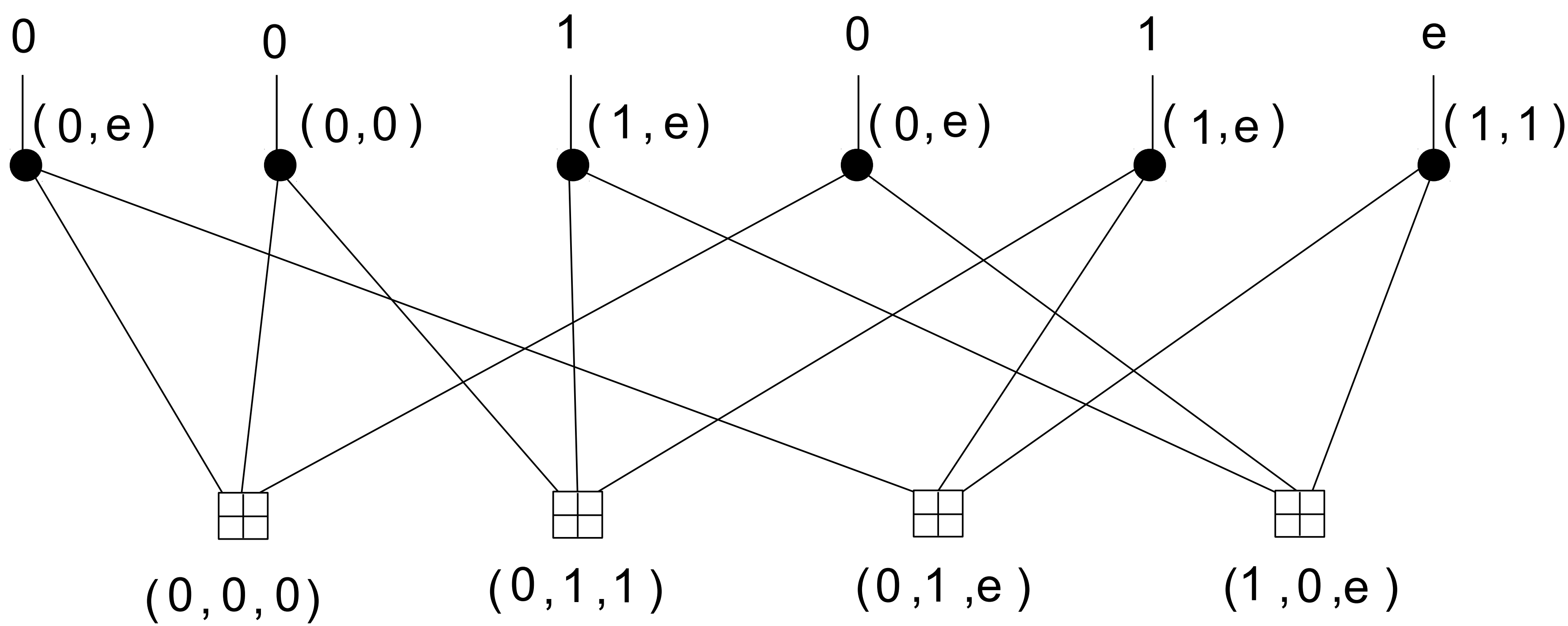
Iteration 2: check-to-bit



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

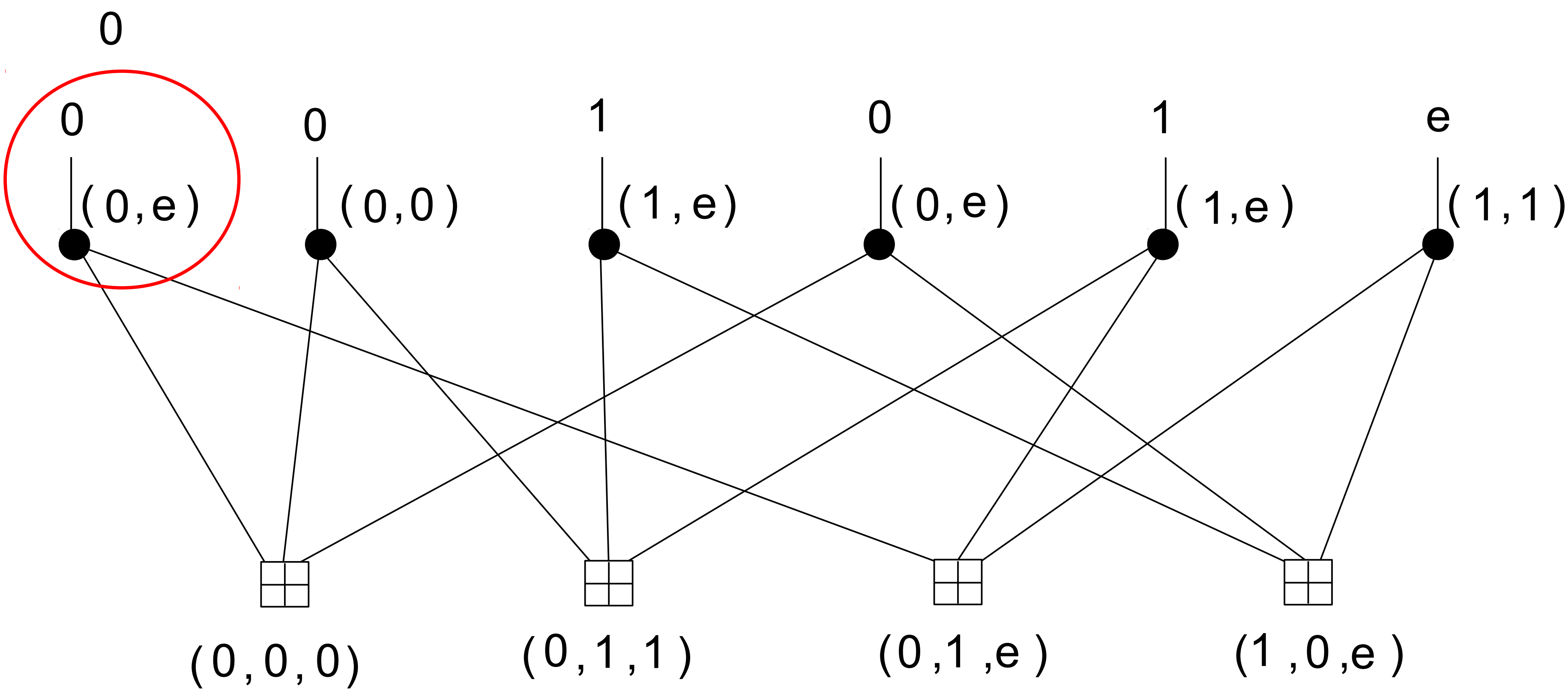
Iteration 2: bit processing



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

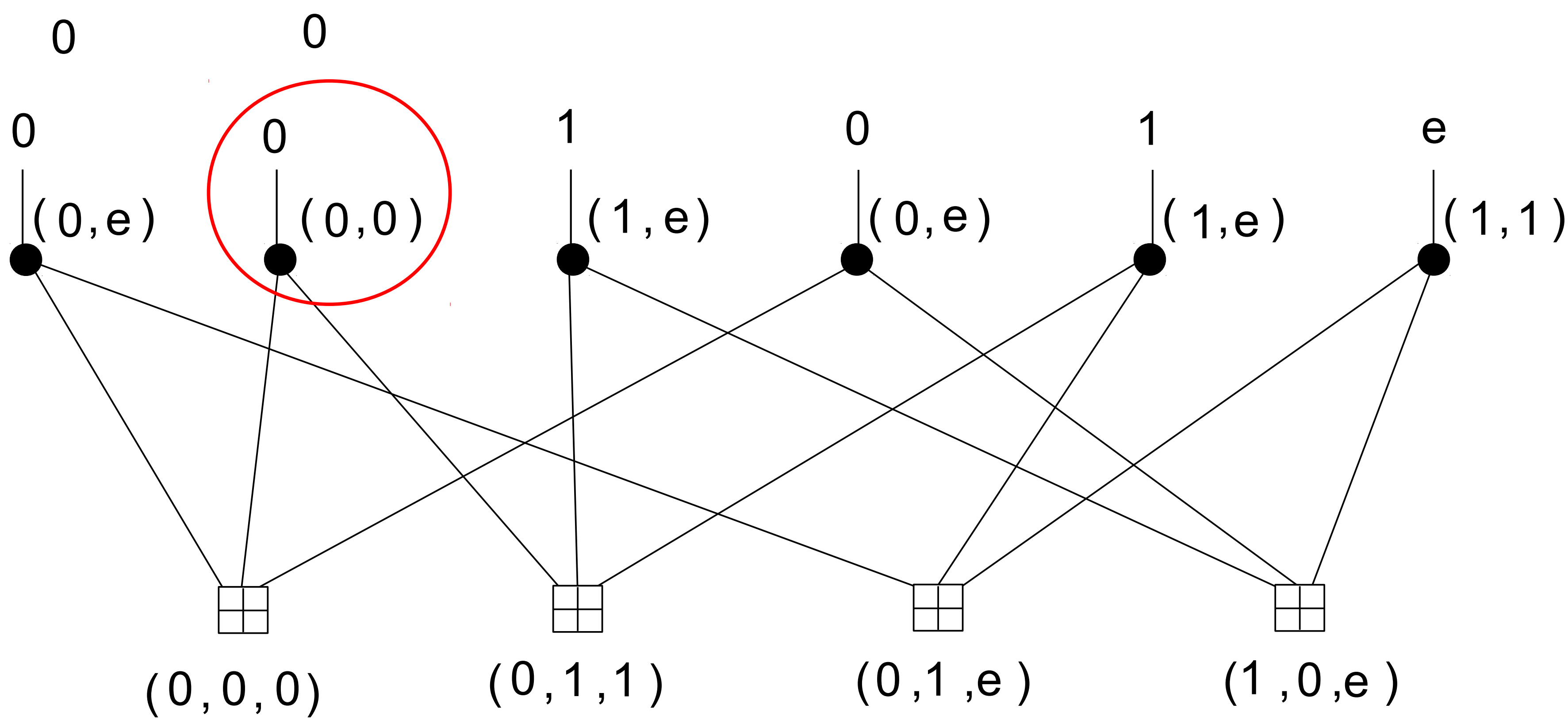
Iteration 2: bit processing



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

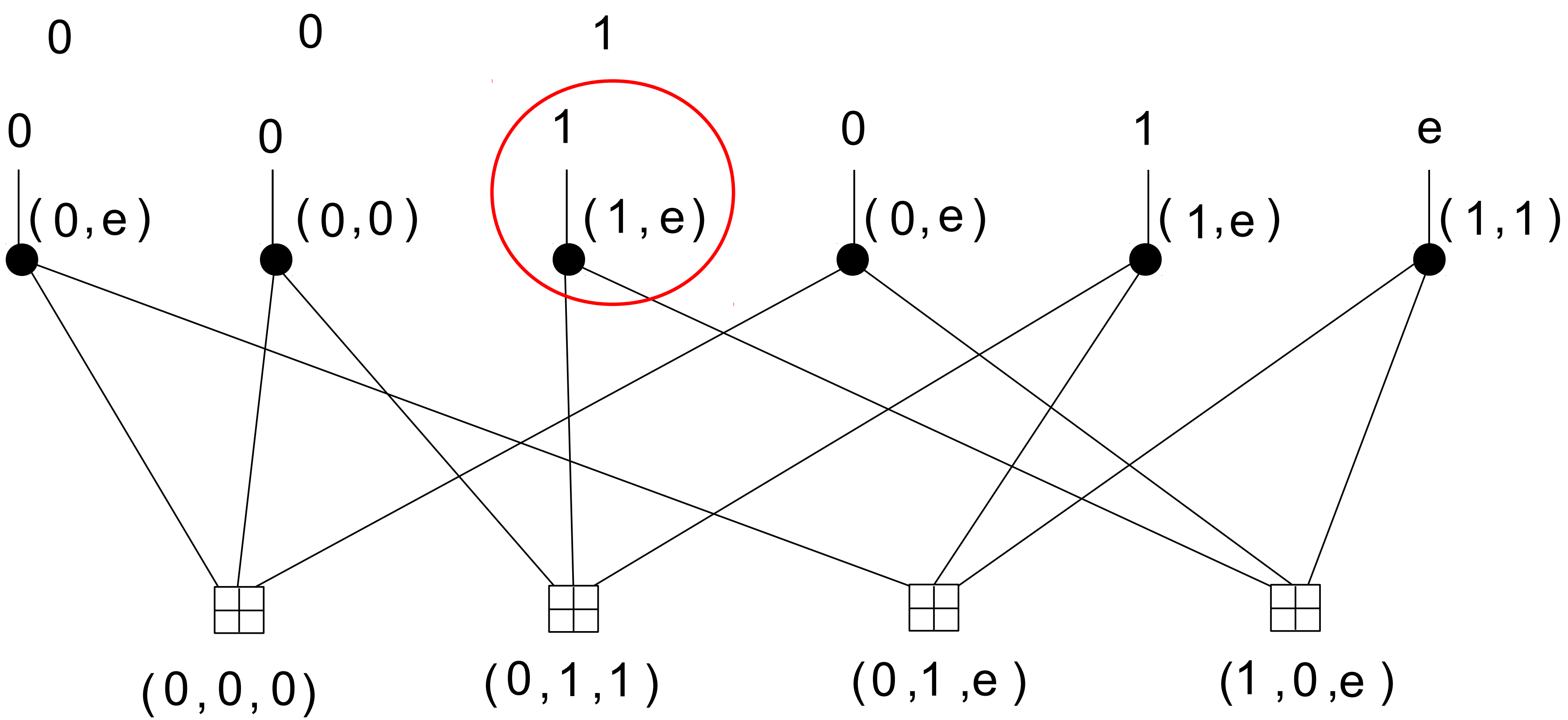
Iteration 2: bit processing



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

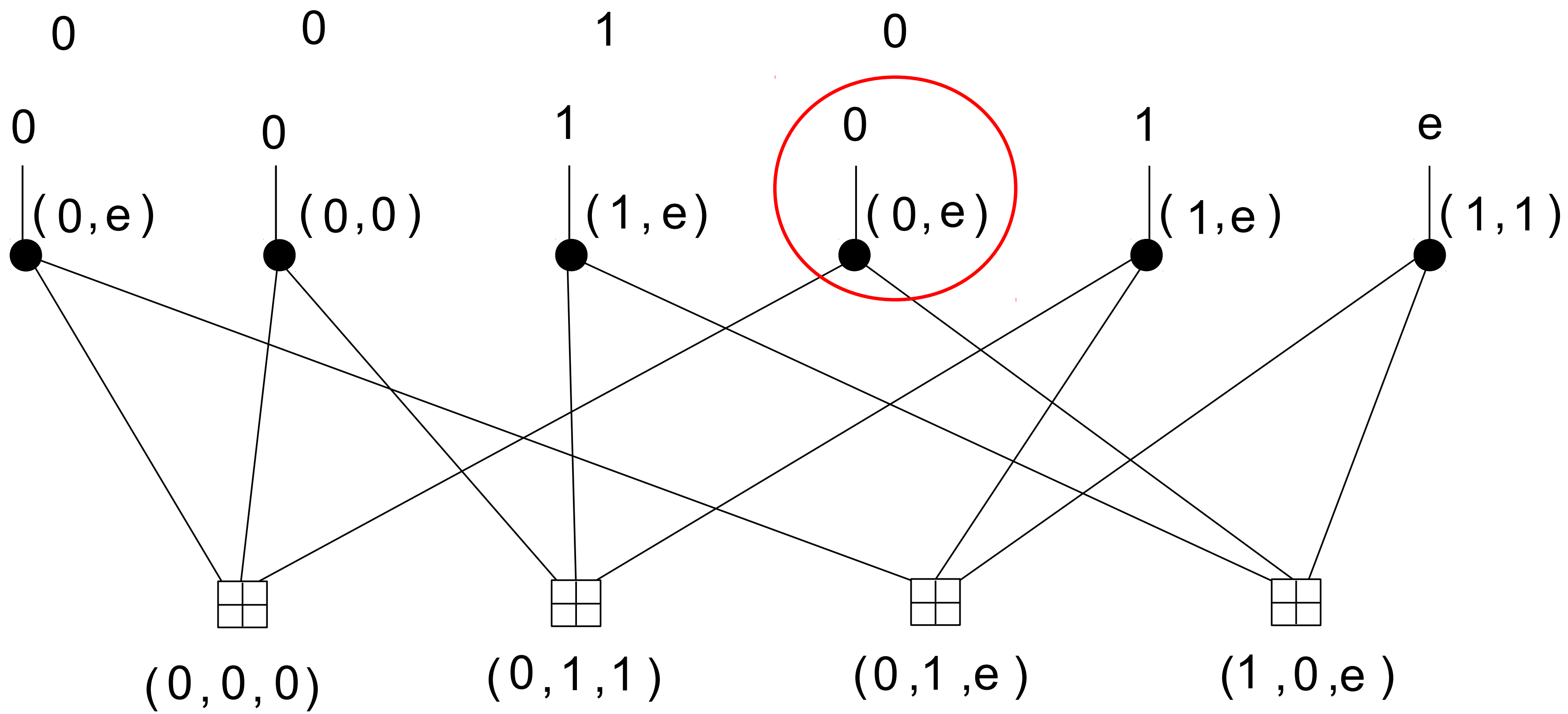
Iteration 2: bit processing



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

Iteration 2: bit processing

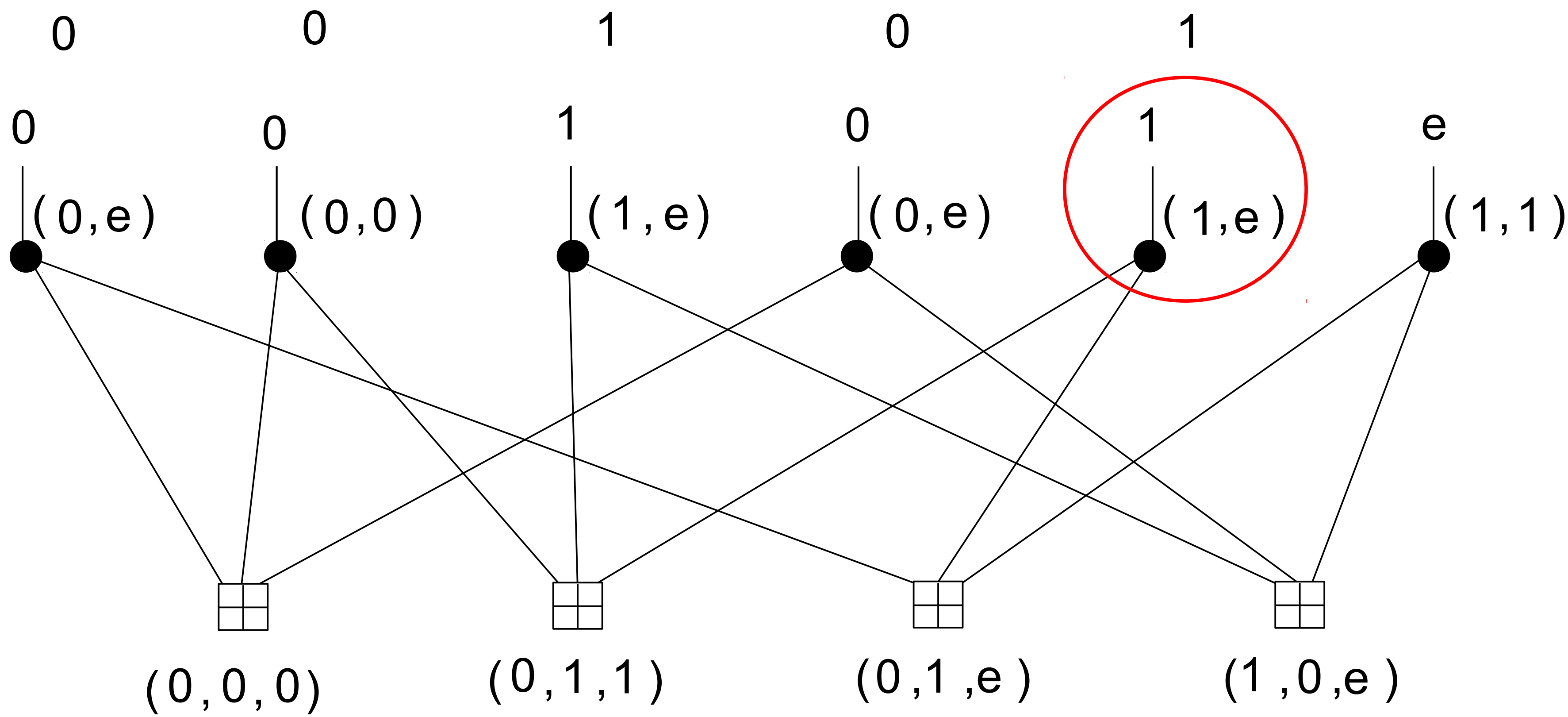


Iterative Decoding Example (BEC)

UNIVERSITY OF NOTRE DAME

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

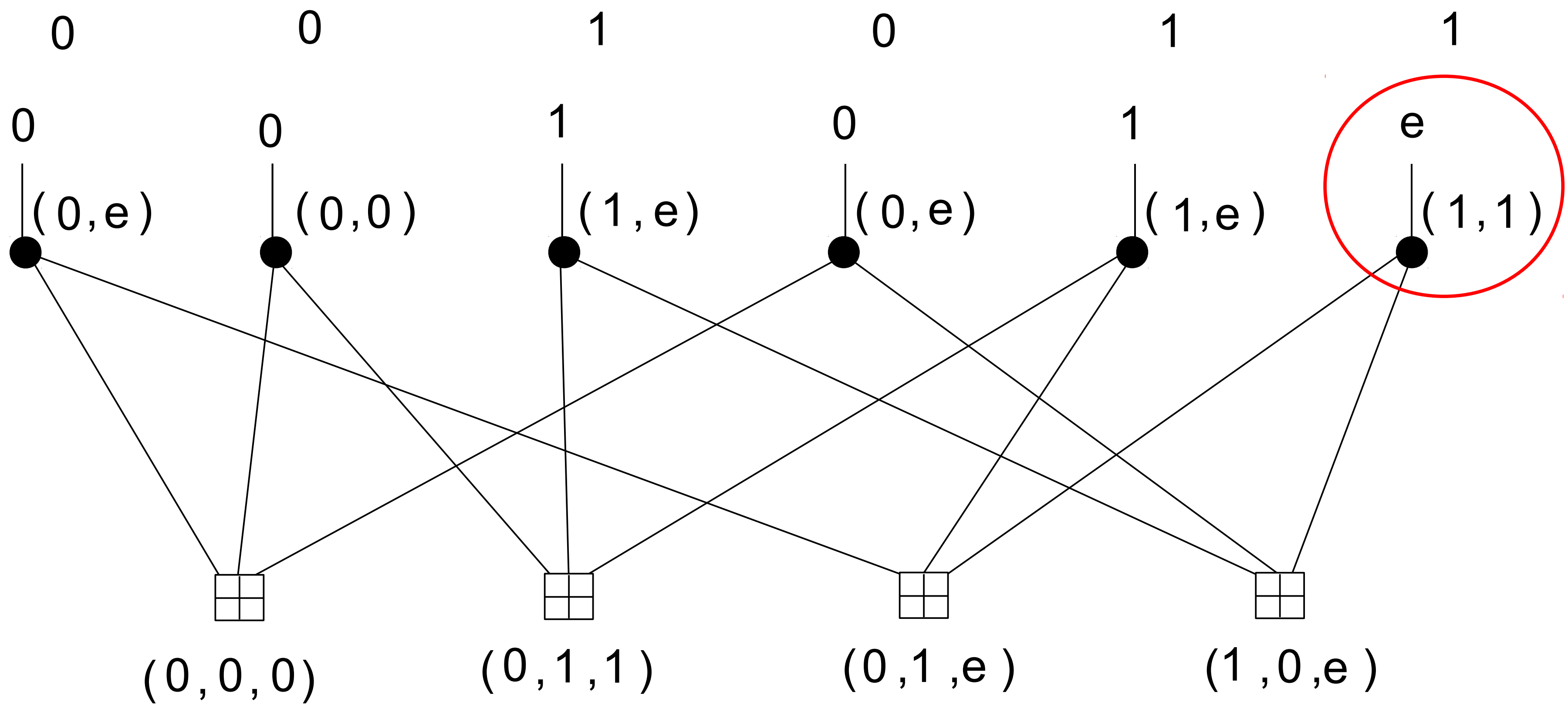
Iteration 2: bit processing



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

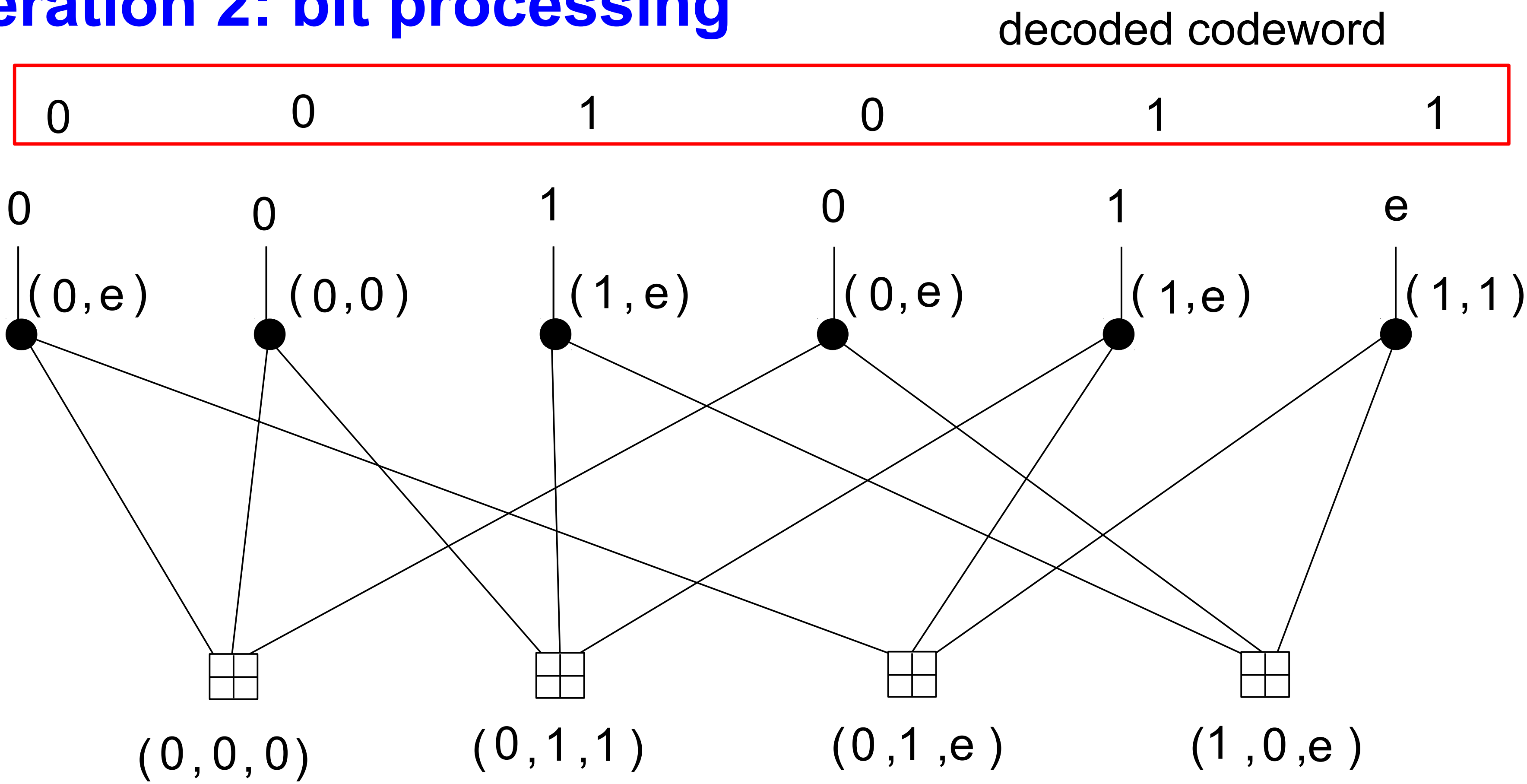
Iteration 2: bit processing



Iterative Decoding Example (BEC)

Ex: $\mathbf{v} = [0\ 0\ 1\ 0\ 1\ 1] \rightarrow \mathbf{r} = [0\ 0\ 1\ e\ e\ e]$

Iteration 2: bit processing



Code Ensembles

- We often analyze a set or **ensemble** of all possible codes with certain parameters rather than a particular code.

Code Ensembles

- We often analyze a set or **ensemble** of all possible codes with certain parameters rather than a particular code.

Gallagers LDPC code construction

- Gallager constructed a (J,K)-regular LDPC code with block length n in the following way

Code Ensembles

- We often analyze a set or **ensemble** of all possible codes with certain parameters rather than a particular code.

Gallagers LDPC code construction

- Gallager constructed a (J,K)-regular LDPC code with block length n in the following way

  (1) The parity-check matrix H is split into J submatrices with m/J rows per submatrix

Code Ensembles

- We often analyze a set or **ensemble** of all possible codes with certain parameters rather than a particular code.

Gallagers LDPC code construction

- Gallager constructed a (J,K)-regular LDPC code with block length n in the following way

  (1) The parity-check matrix H is split into J submatrices with m/J rows per submatrix

  (2) The first submatrix contains diagonally descending m/J x K blocks of K 1's

Code Ensembles

- We often analyze a set or **ensemble** of all possible codes with certain parameters rather than a particular code.

Gallagers LDPC code construction

- Gallager constructed a (J,K)-regular LDPC code with block length n in the following way

  (1) The parity-check matrix H is split into J submatrices with m/J rows per submatrix

  (2) The first submatrix contains diagonally descending m/J x K blocks of K 1's

  (3) The other $J-1$ submatrices are random column permutations of the first submatrix

Code Ensembles

- We often analyze a set or **ensemble** of all possible codes with certain parameters rather than a particular code.

Gallagers LDPC code construction

- Gallager constructed a (J,K)-regular LDPC code with block length n in the following way

  (1) The parity-check matrix H is split into J submatrices with m/J rows per submatrix

  (2) The first submatrix contains diagonally descending m/J x K blocks of K 1's

  (3) The other $J-1$ submatrices are random column permutations of the first submatrix

- The corresponding (J,K)-regular LDPC code ensemble is the set of all such parity-check matrices for a fixed J,K, and n.

Code Ensembles

- Sets of codes that share certain characteristics or properties are called **code ensembles**.

Code Ensembles

UNIVERSITY OF NOTRE DAME

- Sets of codes that share certain characteristics or properties are called **code ensembles**.
- Coding theorists often analyze **ensemble average** performance. Typically, one aims to show that **'good' codes occur with high probability in the ensemble**, i.e., almost all codes are good.

Code Ensembles

- Sets of codes that share certain characteristics or properties are called **code ensembles**.
- Coding theorists often analyze **ensemble average** performance. Typically, one aims to show that **'good' codes occur with high probability in the ensemble**, i.e., almost all codes are good.

Example: Gallager's (J, K)-regular LDPC code ensemble

$$n = 20, J = 3, K = 4$$

$$\left[\begin{array}{cccccccccccccccccccc}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
\hline
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\
\hline
1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0\\
0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1
\end{array}\right]$$

Code Ensembles

- Sets of codes that share certain characteristics or properties are called **code ensembles**.
- Coding theorists often analyze **ensemble average** performance. Typically, one aims to show that **'good' codes occur with high probability in the ensemble**, i.e., almost all codes are good.

Example: Gallager's (J, K)-regular LDPC code ensemble

$$\left[\begin{array}{cccccccccccccccccccc}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
\hline
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\
\hline
1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0\\
0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1
\end{array}\right]$$

$n = 20, J = 3, K = 4$

➡ n=20 cols., J=3 submatrices

Code Ensembles

- Sets of codes that share certain characteristics or properties are called **code ensembles**.
- Coding theorists often analyze **ensemble average** performance. Typically, one aims to show that **'good' codes occur with high probability in the ensemble**, i.e., almost all codes are good.

Example: Gallager's (J, K)-regular LDPC code ensemble

$$n = 20, J = 3, K = 4$$

$$\left[\begin{array}{cccccccccccccccccccc}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
\hline
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\
\hline
1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0\\
0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1
\end{array}\right]$$

➡ n=20 cols., J=3 submatrices

➡ First submatrix contains descending blocks of four 1's

Code Ensembles

- Sets of codes that share certain characteristics or properties are called **code ensembles**.
- Coding theorists often analyze **ensemble average** performance. Typically, one aims to show that **'good' codes occur with high probability in the ensemble**, i.e., almost all codes are good.

Example: Gallager's (J, K)-regular LDPC code ensemble

$$\left[\begin{array}{cccccccccccccccccccc}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
\hline
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\
\hline
1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0\\
0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1
\end{array}\right]$$

$n = 20, J = 3, K = 4$

- ➡ n=20 cols., J=3 submatrices
- ➡ First submatrix contains descending blocks of four 1's
- ➡ Other J-1=2 submatrices are random column permutations of the first submatrix

Code Ensembles

UNIVERSITY OF NOTRE DAME

- Sets of codes that share certain characteristics or properties are called **code ensembles**.
- Coding theorists often analyze **ensemble average** performance. Typically, one aims to show that **'good' codes occur with high probability in the ensemble**, i.e., almost all codes are good.

Example: Gallager's (J, K)-regular LDPC code ensemble

$$\left[\begin{array}{cccccccccccccccccccc}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
\hline
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\
\hline
1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0\\
0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1
\end{array}\right]$$

$n = 20, J = 3, K = 4$

- ➡ n=20 cols., J=3 submatrices
- ➡ First submatrix contains descending blocks of four 1's
- ➡ Other J-1=2 submatrices are random column permutations of the first submatrix
- ➡ The code ensemble contains all such parity-check matrices

Code Ensembles

- Sets of codes that share certain characteristics or properties are called **code ensembles**.
- Coding theorists often analyze **ensemble average** performance. Typically, one aims to show that **'good' codes occur with high probability in the ensemble**, i.e., almost all codes are good.

Example: Gallager's (J, K)-regular LDPC code ensemble

$$n = 20, J = 3, K = 4$$

$$
\left[\begin{array}{cccccccccccccccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\hline
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0
\end{array}\right]
$$

- ➡ n=20 cols., J=3 submatrices
- ➡ First submatrix contains descending blocks of four 1's
- ➡ Other J-1=2 submatrices are random column permutations of the first submatrix
- ➡ The code ensemble contains all such parity-check matrices

Code Ensembles

- Sets of codes that share certain characteristics or properties are called **code ensembles**.
- Coding theorists often analyze **ensemble average** performance. Typically, one aims to show that **'good' codes occur with high probability in the ensemble**, i.e., almost all codes are good.

Example: Gallager's (J, K)-regular LDPC code ensemble

$$n = 20, J = 3, K = 4$$

$$\left[\begin{array}{cccccccccccccccccccc}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
\hline
0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&1&0&0&1&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
1&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&0&0&1&0&1&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&1&0&0&0&1&0&0&0&0\\
\hline
0&0&0&0&1&0&1&0&0&0&1&0&0&1&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&0&0&1&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0\\
1&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1
\end{array}\right]$$

- ➡ n=20 cols., J=3 submatrices
- ➡ First submatrix contains descending blocks of four 1's
- ➡ Other J-1=2 submatrices are random column permutations of the first submatrix
- ➡ The code ensemble contains all such parity-check matrices

Minimum distance of (J,K)-regular LDPC codes

- Gallager proved that for $J > 2$ and large block length n, the minimum distance typical of most members of a (J,K)-regular LDPC code ensemble is bounded below as $\delta_{min} n$, where $\delta_{min} > 0$ is a constant that depends on J and K (not n), i.e., $d_{min} \geq \delta_{min} n$.

Minimum distance of (J,K)-regular LDPC codes

- Gallager proved that for $J > 2$ and large block length n, the minimum distance typical of most members of a (J,K)-regular LDPC code ensemble is bounded below as $\delta_{min} n$, where $\delta_{min} > 0$ is a constant that depends on J and K (not n), i.e., $d_{min} \geq \delta_{min} n$.

- δ_{min} is called the **minimum distance growth rate**. For (J,K)-regular code ensembles, we write $\delta_{min} = \delta_{JK}$.

Minimum distance of (J,K)-regular LDPC codes

- Gallager proved that for $J > 2$ and large block length n, the minimum distance typical of most members of a (J,K)-regular LDPC code ensemble is bounded below as $\delta_{min} n$, where $\delta_{min} > 0$ is a constant that depends on J and K (not n), i.e., $d_{min} \geq \delta_{min} n$.

- δ_{min} is called the **minimum distance growth rate**. For (J,K)-regular code ensembles, we write $\delta_{min} = \delta_{JK}$.

Example: For (3,6)-regular codes, $\delta_{min} = \delta_{36} = 0.023$. Then with high probability:

$$n = 1000 \implies d_{\min} \geq 0.023 \times 1000 = 23$$
$$n = 10000 \implies d_{\min} \geq 0.023 \times 10000 = 230$$

Minimum distance of (J,K)-regular LDPC codes

- Gallager proved that for $J > 2$ and large block length n, the minimum distance typical of most members of a (J,K)-regular LDPC code ensemble is bounded below as $\delta_{min} n$, where $\delta_{min} > 0$ is a constant that depends on J and K (not n), i.e., $d_{min} \geq \delta_{min} n$.

- δ_{min} is called the **minimum distance growth rate**. For (J,K)-regular code ensembles, we write $\delta_{min} = \delta_{JK}$.

Example: For (3,6)-regular codes, $\delta_{min} = \delta_{36} = 0.023$. Then with high probability:

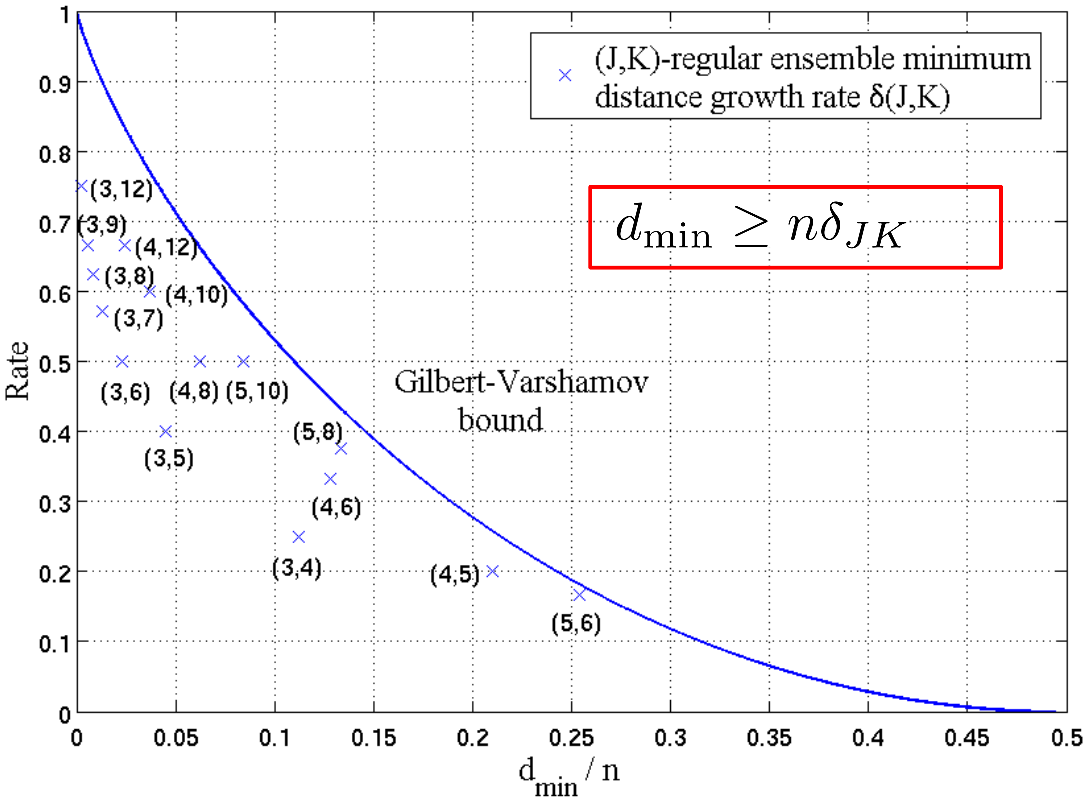
$$n = 1000 \implies d_{\min} \geq 0.023 \times 1000 = 23$$
$$n = 10000 \implies d_{\min} \geq 0.023 \times 10000 = 230$$

- Such code ensembles are called **asymptotically good**

→ Under ML decoding, we will achieve excellent performance

Code Ensembles – Minimum Distance Growth Rates

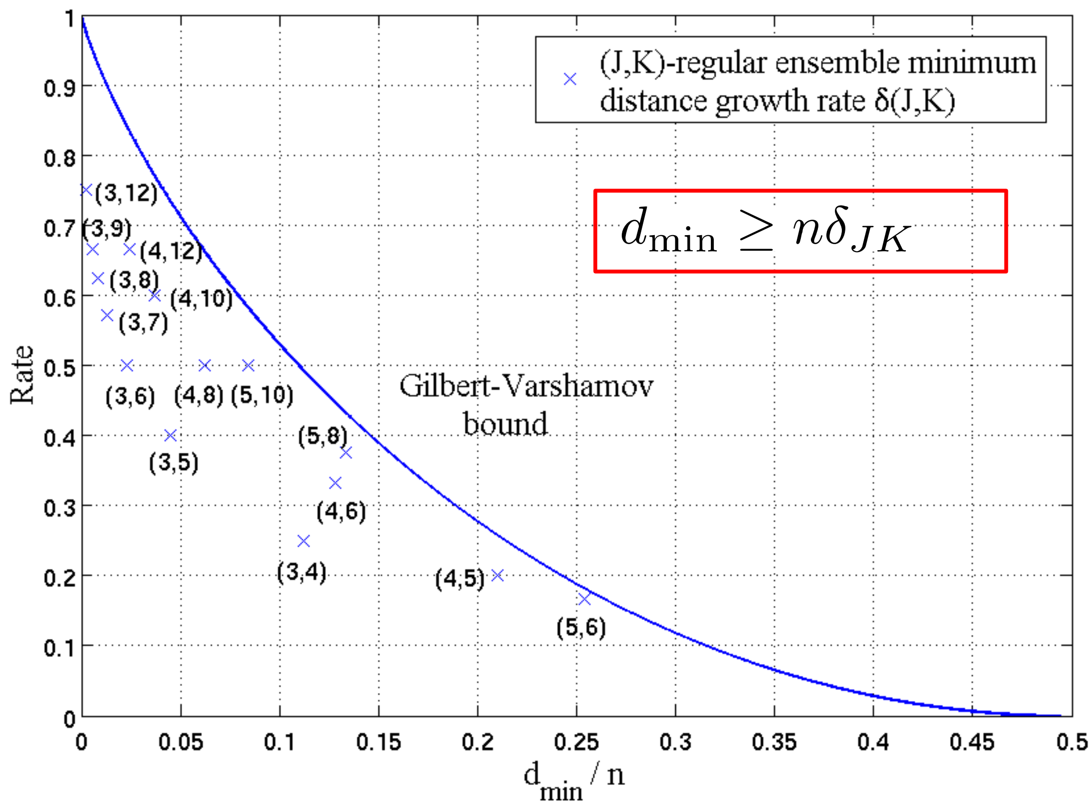
- δ_{JK} is called the **typical minimum distance ratio**, or **minimum distance growth rate**, of a (J,K)-regular code ensemble $(R \geq 1 - J/K)$.



$$d_{\min} \geq n\delta_{JK}$$

Code Ensembles – Minimum Distance Growth Rates

- δ_{JK} is called the **typical minimum distance ratio**, or **minimum distance growth rate**, of a (J,K)-regular code ensemble $(R \geq 1 - J/K)$.

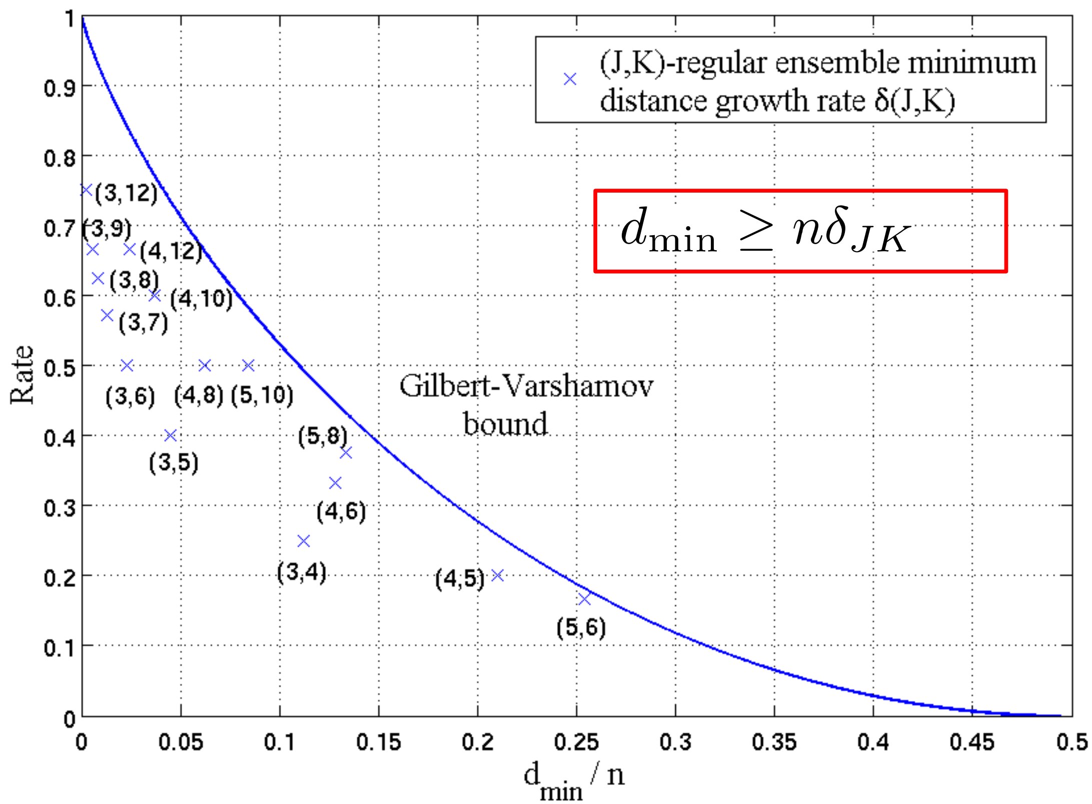


- An ensemble with minimum distance growth rate strictly greater than zero is called **asymptotically good**, in the sense that, as the block length tends to infinity, the minimum distance grows linearly with block length.

Code Ensembles – Minimum Distance Growth Rates

UNIVERSITY OF NOTRE DAME

- δ_{JK} is called the **typical minimum distance ratio**, or **minimum distance growth rate**, of a (J,K)-regular code ensemble $(R \geq 1 - J/K)$.



- An ensemble with minimum distance growth rate strictly greater than zero is called **asymptotically good**, in the sense that, as the block length tends to infinity, the minimum distance grows linearly with block length.
- As the **density** (J and K) of (J,K)-regular ensembles **increases**, the minimum distance growth rate **approaches the Gilbert-Varshamov bound** for the ensemble of all block codes.

Density Evolution

UNIVERSITY OF NOTRE DAME

- We use **density evolution** (DE) to determine for which channel noise levels the message passing decoder can correct errors and for which levels it cannot.

Density Evolution

- We use **density evolution** (DE) to determine for which channel noise levels the message passing decoder can correct errors and for which levels it cannot.
- This is achieved by tracking probability density functions over iterations.

Density Evolution

- We use **density evolution** (DE) to determine for which channel noise levels the message passing decoder can correct errors and for which levels it cannot.
- This is achieved by tracking probability density functions over iterations.

We assume:

(1) An ensemble of Tanner graphs

(2) The channel is memoryless

(3) The graphs are cycle free

Density Evolution

- We use **density evolution** (DE) to determine for which channel noise levels the message passing decoder can correct errors and for which levels it cannot.
- This is achieved by tracking probability density functions over iterations.

We assume:

(1) An ensemble of Tanner graphs

(2) The channel is memoryless

(3) The graphs are cycle free

- For a given code ensemble we determine an **iterative decoding threshold** – the maximum level of noise that can possibly be corrected by a **particular code ensemble**

DE on the BEC

UNIVERSITY OF NOTRE DAME

DE for (J,K)-regular ensembles:

- Messages are '1', '0', or 'e'
- Let $q^{(l)}$ be the probability that a check-to-bit message is an 'e' at iteration l
- Let $p^{(l)}$ be the probability that a bit-to-check message is an 'e' at iteration l

Check-to-bit probabilities

Recall: At iteration l, a message from check-to-bit is an 'e' iff any of the $K-1$ other incoming messages are an 'e'

Check-to-bit probabilities

Recall: At iteration l, a message from check-to-bit is an 'e' iff any of the $K-1$ other incoming messages are an 'e'

- Assuming all incoming messages are independent (memoryless channel and no cycles of length $2l$ or less), the probability that no incoming message is an 'e' is

$$(1 - p^{(l)})^{K-1}$$

Check-to-bit probabilities

Recall: At iteration l, a message from check-to-bit is an 'e' iff any of the $K-1$ other incoming messages are an 'e'

- Assuming all incoming messages are independent (memoryless channel and no cycles of length $2l$ or less), the probability that no incoming message is an 'e' is

$$(1-p^{(l)})^{K-1}$$

- This implies that the probability that **at least one** incoming message is and 'e' is

$$q^{(l)} = 1-(1-p^{(l)})^{K-1}$$

Bit-to-check probabilities

- At iteration l, a message from bit-to-check is an 'e' if
  (1) The original message from the channel is an 'e', **and**
  (2) all the incoming messages at iteration $l-1$ are an 'e'.

Bit-to-check probabilities

UNIVERSITY OF NOTRE DAME

- At iteration l, a message from bit-to-check is an 'e' if
  (1) The original message from the channel is an 'e', **and**
  (2) all the incoming messages at iteration $l-1$ are an 'e'.
- Assuming independent iterations

$$p^{(l)} = \varepsilon \left(q^{(l-1)}\right)^{J-1} = \varepsilon \left(1 - \left(1 - p^{(l-1)}\right)^{K-1}\right)^{J-1}$$

Bit-to-check probabilities

- At iteration l, a message from bit-to-check is an 'e' if
  (1) The original message from the channel is an 'e', **and**
  (2) all the incoming messages at iteration $l-1$ are an 'e'.
- Assuming independent iterations

$$p^{(l)} = \varepsilon \left(q^{(l-1)}\right)^{J-1} = \varepsilon \left(1 - \left(1 - p^{(l-1)}\right)^{K-1}\right)^{J-1}$$

$\Longrightarrow$ for a (J,K)-regular LDPC code ensemble

$$p^{(l)} = \begin{cases} \varepsilon, & l = 0 \\ \varepsilon \left(1 - \left(1 - p^{(l-1)}\right)^{K-1}\right)^{J-1}, & l > 0 \end{cases}$$

Bit-to-check probabilities

- At iteration l, a message from bit-to-check is an 'e' if
  (1) The original message from the channel is an 'e', **and**
  (2) all the incoming messages at iteration $l-1$ are an 'e'.
- Assuming independent iterations

$$p^{(l)} = \varepsilon\left(q^{(l-1)}\right)^{J-1} = \varepsilon\left(1-\left(1-p^{(l-1)}\right)^{K-1}\right)^{J-1}$$

$\Longrightarrow$ for a (J,K)-regular LDPC code ensemble

$$p^{(l)} = \begin{cases} \varepsilon, & l = 0 \\ \varepsilon\left(1-\left(1-p^{(l-1)}\right)^{K-1}\right)^{J-1}, & l > 0 \end{cases}$$

- We can use this method to determine for what values of ε the message passing decoder is likely to correct erasures and how many iterations are necessary!

Density Evolution: Example

- Suppose a code from the (3,6)-regular LDPC code ensemble is transmitted on a BEC with erasure probability $\varepsilon = 0.3$

Density Evolution: Example

- Suppose a code from the (3,6)-regular LDPC code ensemble is transmitted on a BEC with erasure probability $\varepsilon = 0.3$

- The probability that a codeword bit will remain erased after l iterations (if the code graph is cycle free) is given by

$$p^{(0)} = 0.3, \qquad p^{(l)} = 0.3\left(1 - \left(1 - p^{(l-1)}\right)^5\right)^2$$

Density Evolution: Example

- Suppose a code from the (3,6)-regular LDPC code ensemble is transmitted on a BEC with erasure probability $\varepsilon = 0.3$

- The probability that a codeword bit will remain erased after l iterations (if the code graph is cycle free) is given by

$$p^{(0)} = 0.3, \qquad p^{(l)} = 0.3\left(1 - \left(1 - p^{(l-1)}\right)^5\right)^2$$

- Then:

$$\begin{aligned} p^{(0)} &= 0.3000 & p^{(5)} &= 0.0098 \\ p^{(1)} &= 0.2076 & p^{(6)} &= 0.0007 \\ p^{(2)} &= 0.1419 & p^{(7)} &= 0.0000 \\ p^{(3)} &= 0.0858 \\ p^{(4)} &= 0.0392 \end{aligned}$$

Density Evolution: Example

- Suppose a code from the (3,6)-regular LDPC code ensemble is transmitted on a BEC with erasure probability $\varepsilon = 0.3$

- The probability that a codeword bit will remain erased after l iterations (if the code graph is cycle free) is given by

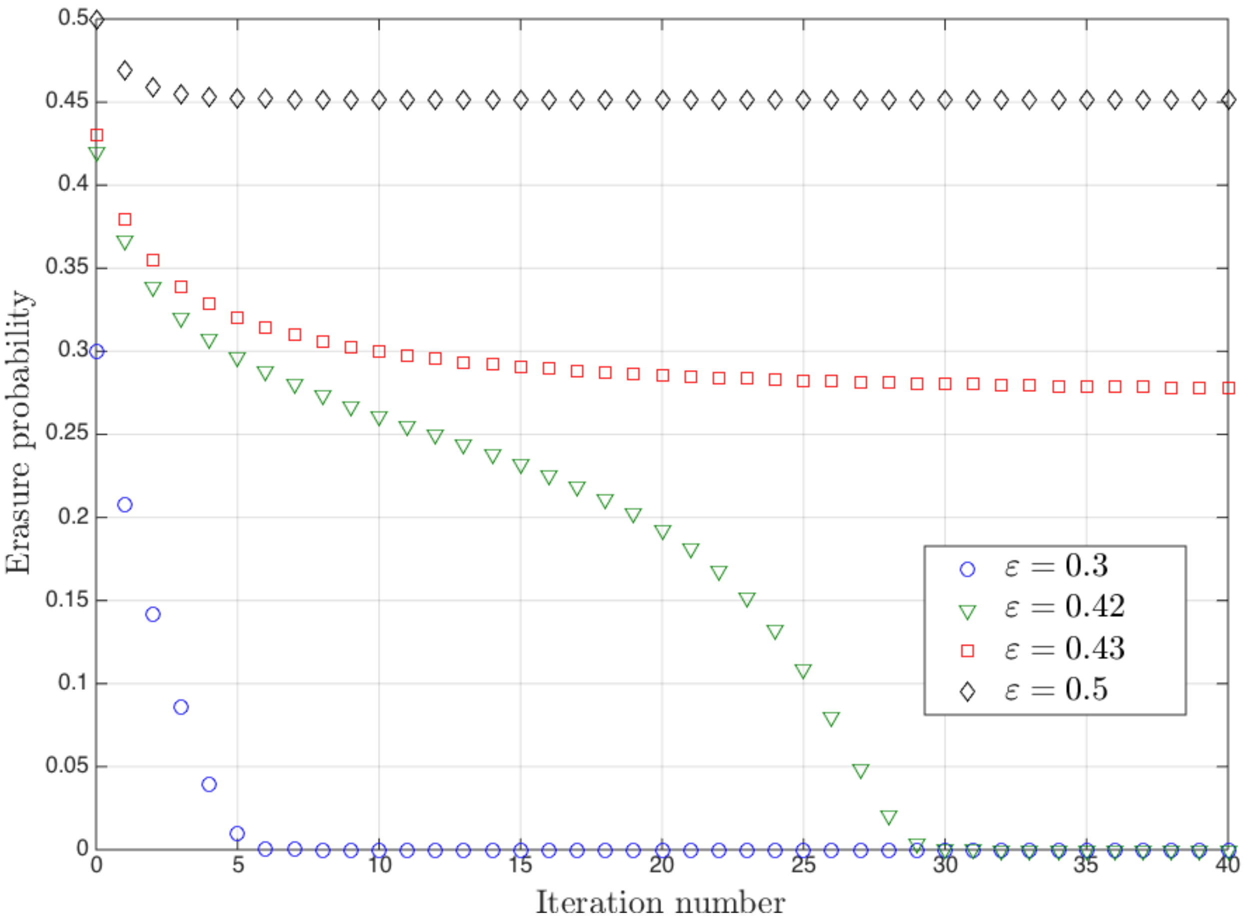
$$p^{(0)} = 0.3, \qquad p^{(l)} = 0.3\left(1-\left(1-p^{(l-1)}\right)^5\right)^2$$

- Then:

$$p^{(0)} = 0.3000 \qquad p^{(5)} = 0.0098$$
$$p^{(1)} = 0.2076 \qquad p^{(6)} = 0.0007$$
$$p^{(2)} = 0.1419 \qquad p^{(7)} = 0.0000$$
$$p^{(3)} = 0.0858$$
$$p^{(4)} = 0.0392$$

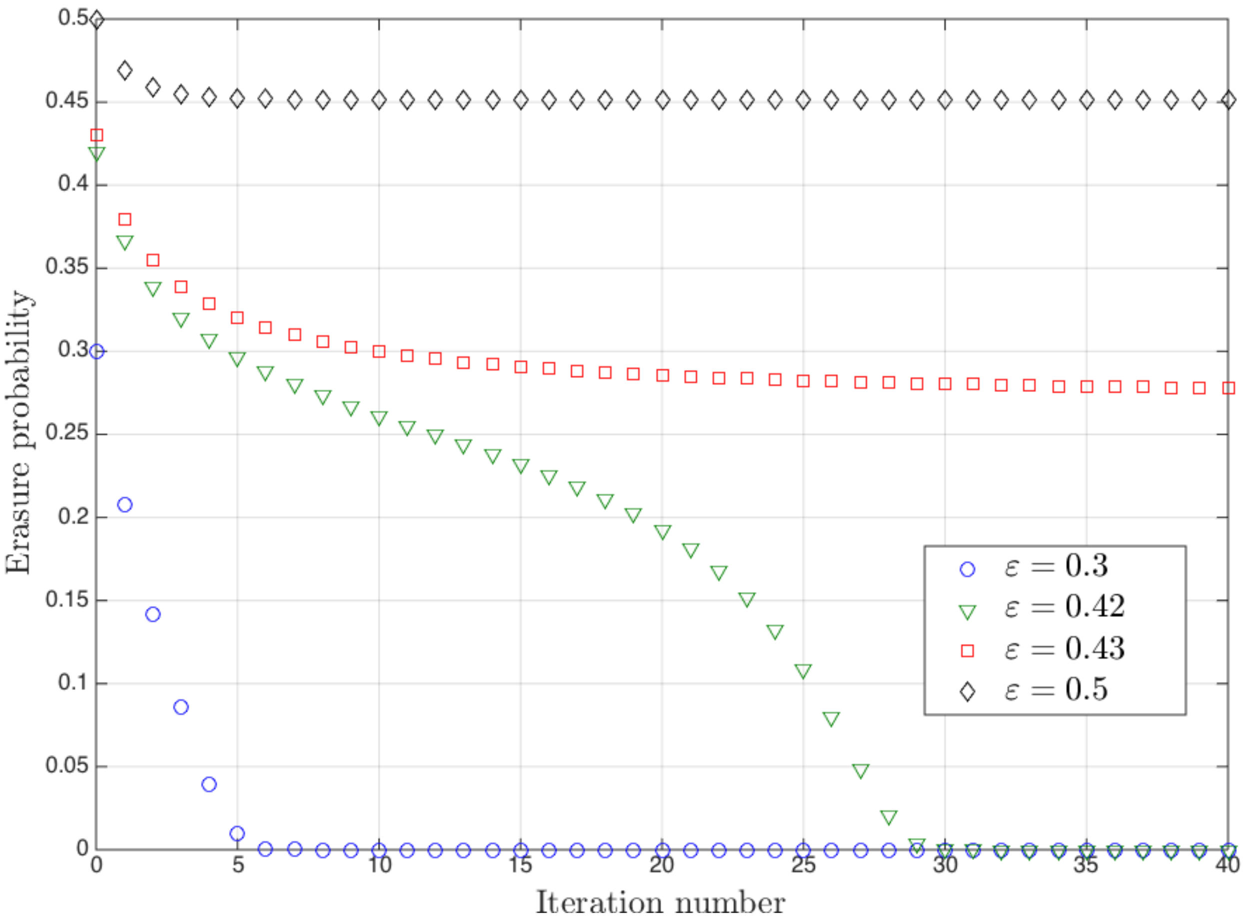
→ After **7 iterations**, the erasure probability in a codeword will go to 0 for a (3,6)-regular LDPC code (with no 14-cycles or less) transmitted over a BEC with $\varepsilon = 0.3$

Example (cont.)



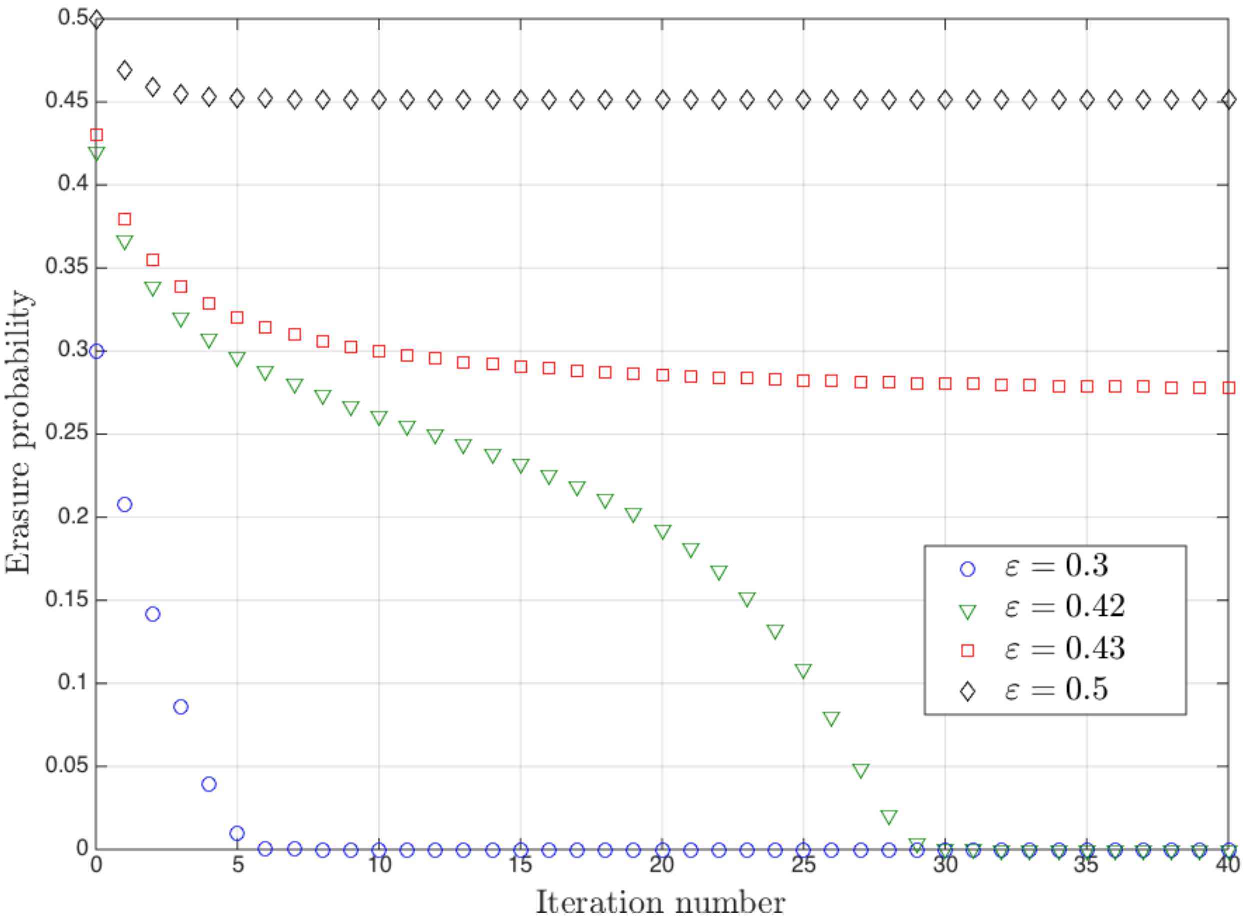
- As ε increases, it **takes more iterations** for the erasure probability to converge to 0.

Example (cont.)



- As ε increases, it **takes more iterations** for the erasure probability to converge to 0.
- For a large enough ε (a bad channel) the erasure probability **does not go to 0.**

Example (cont.)



- As ε increases, it **takes more iterations** for the erasure probability to converge to 0.
- For a large enough ε (a bad channel) the erasure probability **does not go to 0.**
- For (3,6)-regular LDPC codes, this value is $\varepsilon^* = 0.429$. ε^* is called the **iterative decoding threshold**.

DE on General Memoryless Channels

- DE can be carried out on general memoryless channels, but it is more complicated.

DE on General Memoryless Channels

- DE can be carried out on general memoryless channels, but it is more complicated.

- Recall that LLR values are continuous

$$L(x) = \log\left(\frac{\mathbb{P}(x=0)}{\mathbb{P}(x=1)}\right)$$

so the probability that a message is a particular LLR value is described by a probability density function (PDF).

DE on General Memoryless Channels

- DE can be carried out on general memoryless channels, but it is more complicated.
- Recall that LLR values are continuous

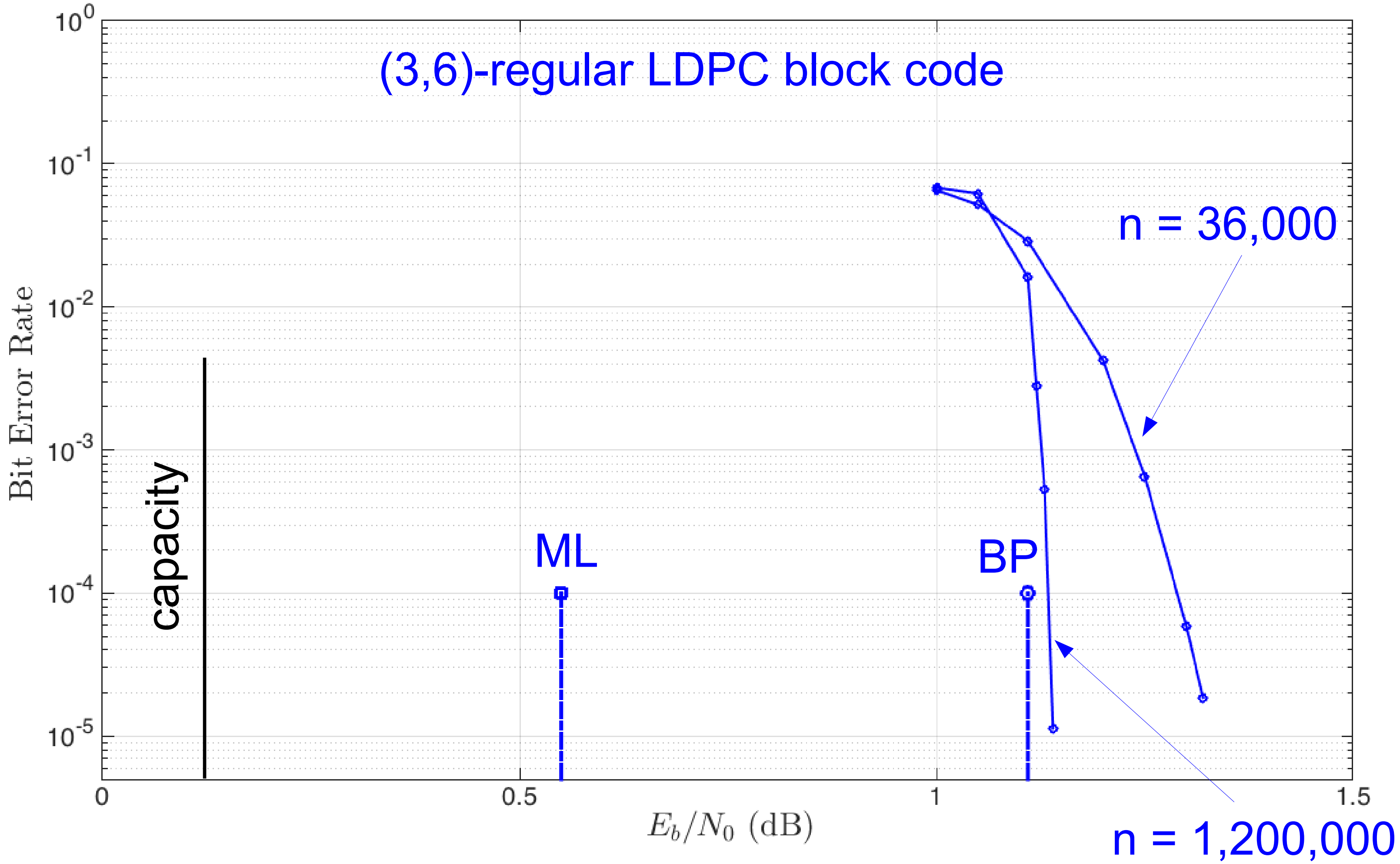
$$L(x) = \log\left(\frac{\mathbb{P}(x=0)}{\mathbb{P}(x=1)}\right)$$

so the probability that a message is a particular LLR value is described by a probability density function (PDF).

➡ We track the PDFs over iterations and determine if the probability of error goes to zero as the number of iterations goes to infinity for a particular channel SNR.

Example: AWGNC

BP = iterative (suboptimal) decoding threshold
ML = maximum likelihood (optimal) decoding threshold



Thresholds of (J,K)-regular LDPC Code Ensembles

UNIVERSITY OF NOTRE DAME

- The iterative decoding thresholds can be calculated for a variety of (J,K)-regular LDPC code ensembles using DE.

BEC thresholds

J	K	Rate	ε^*	ε_{Sh}
3	6	0.5	0.429	0.5
4	8	0.5	0.383	0.5
5	10	0.5	0.341	0.5
3	5	0.4	0.517	0.6
4	6	0.333	0.506	0.667
3	4	0.25	0.647	0.75

AWGNC thresholds

J	K	Rate	$(E_b/N_0)^*$	$(E_b/N_0)_{\text{Sh}}$
3	6	0.5	1.11	0.184
4	8	0.5	1.61	0.184
5	10	0.5	2.04	0.184
3	5	0.4	0.96	-0.229
4	6	0.333	1.67	-0.480
3	4	0.25	1.00	-0.790

[RU01] T. J. Richardson, and R. Urbanke, "The capacity of low-density parity-check codes under message passing decoding", *IEEE Transactions on Information Theory*, vol. 47 no. 2, Feb. 2001.

Thresholds of (J,K)-regular LDPC Code Ensembles

- The iterative decoding thresholds can be calculated for a variety of (J,K)-regular LDPC code ensembles using DE.

BEC thresholds

J	K	Rate	ε^*	ε_{Sh}
3	6	0.5	0.429	0.5
4	8	0.5	0.383	0.5
5	10	0.5	0.341	0.5
3	5	0.4	0.517	0.6
4	6	0.333	0.506	0.667
3	4	0.25	0.647	0.75

AWGNC thresholds

J	K	Rate	$(E_b/N_0)^*$	$(E_b/N_0)_{\text{Sh}}$
3	6	0.5	1.11	0.184
4	8	0.5	1.61	0.184
5	10	0.5	2.04	0.184
3	5	0.4	0.96	-0.229
4	6	0.333	1.67	-0.480
3	4	0.25	1.00	-0.790

- There exists a relatively **large gap to capacity**.

[RU01] T. J. Richardson, and R. Urbanke, "The capacity of low-density parity-check codes under message passing decoding", *IEEE Transactions on Information Theory*, vol. 47 no. 2, Feb. 2001.

Thresholds of (J,K)-regular LDPC Code Ensembles

- The iterative decoding thresholds can be calculated for a variety of (J,K)-regular LDPC code ensembles using DE.

BEC thresholds

J	K	Rate	ε^*	ε_{Sh}
3	6	0.5	0.429	0.5
4	8	0.5	0.383	0.5
5	10	0.5	0.341	0.5
3	5	0.4	0.517	0.6
4	6	0.333	0.506	0.667
3	4	0.25	0.647	0.75

AWGNC thresholds

J	K	Rate	$(E_b/N_0)^*$	$(E_b/N_0)_{\text{Sh}}$
3	6	0.5	1.11	0.184
4	8	0.5	1.61	0.184
5	10	0.5	2.04	0.184
3	5	0.4	0.96	−0.229
4	6	0.333	1.67	−0.480
3	4	0.25	1.00	−0.790

- There exists a relatively **large gap to capacity**.
- Iterative decoding thresholds get **further from capacity** as the graph **density increases**. (Minimum distance growth rates improve.)

[RU01] T. J. Richardson, and R. Urbanke, "The capacity of low-density parity-check codes under message passing decoding", *IEEE Transactions on Information Theory*, vol. 47 no. 2, Feb. 2001.

Choosing the degree distribution

- Generally, better iterative decoding thresholds are obtained by more irregular degree distributions.

Choosing the degree distribution

- Generally, better iterative decoding thresholds are obtained by more irregular degree distributions.
- High degree variable nodes quickly converge to their solution and will pass correct messages to many connected check nodes.

Choosing the degree distribution

- Generally, better iterative decoding thresholds are obtained by more irregular degree distributions.
- High degree variable nodes quickly converge to their solution and will pass correct messages to many connected check nodes.
- To keep the density low, many variable nodes of degree 2 are required (but no more than allowed by the stability constraint). This typically has a negative effect on other code properties.

Choosing the degree distribution

- Generally, better iterative decoding thresholds are obtained by more irregular degree distributions.
- High degree variable nodes quickly converge to their solution and will pass correct messages to many connected check nodes.
- To keep the density low, many variable nodes of degree 2 are required (but no more than allowed by the stability constraint). This typically has a negative effect on other code properties.
- Check node irregularity is not essential to improve thresholds. Typically good code designs have one or two different check node degrees.

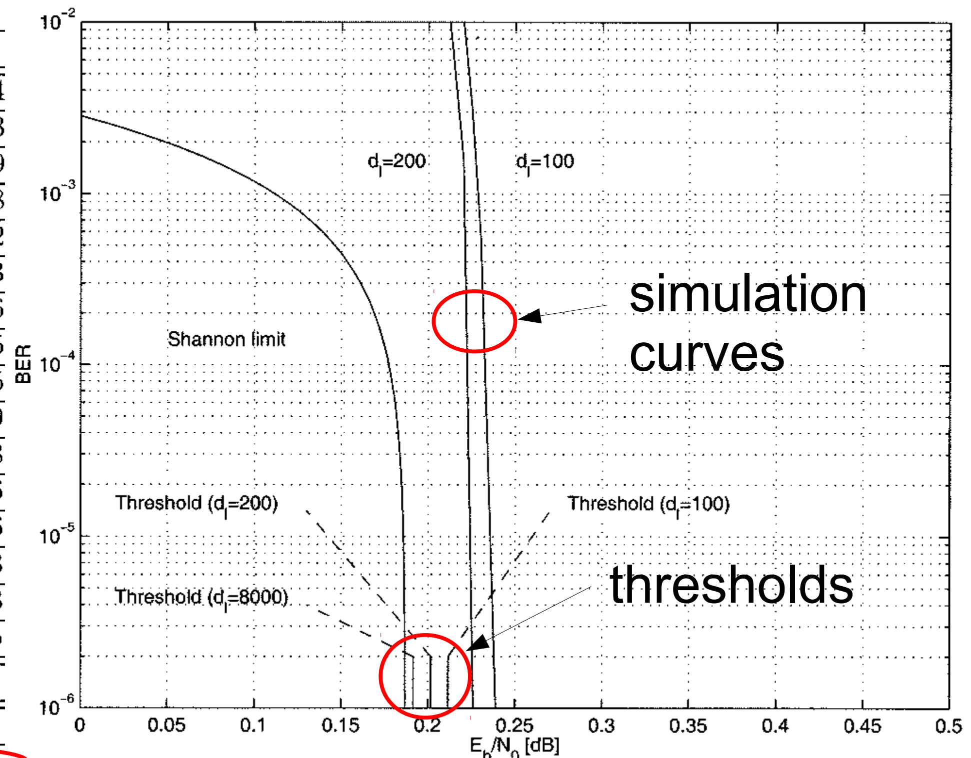
Choosing the degree distribution

- Generally, better iterative decoding thresholds are obtained by more irregular degree distributions.
- High degree variable nodes quickly converge to their solution and will pass correct messages to many connected check nodes.
- To keep the density low, many variable nodes of degree 2 are required (but no more than allowed by the stability constraint). This typically has a negative effect on other code properties.
- Check node irregularity is not essential to improve thresholds. Typically good code designs have one or two different check node degrees.
- Trying all possible degree distributions is not practical. Optimization techniques such as iterative linear programming and differential evolution have been applied to find good degree distributions.

Random Irregular LDPC Codes

- Good rate $R = 1/2$ irregular LDPC codes: [CFRU01]

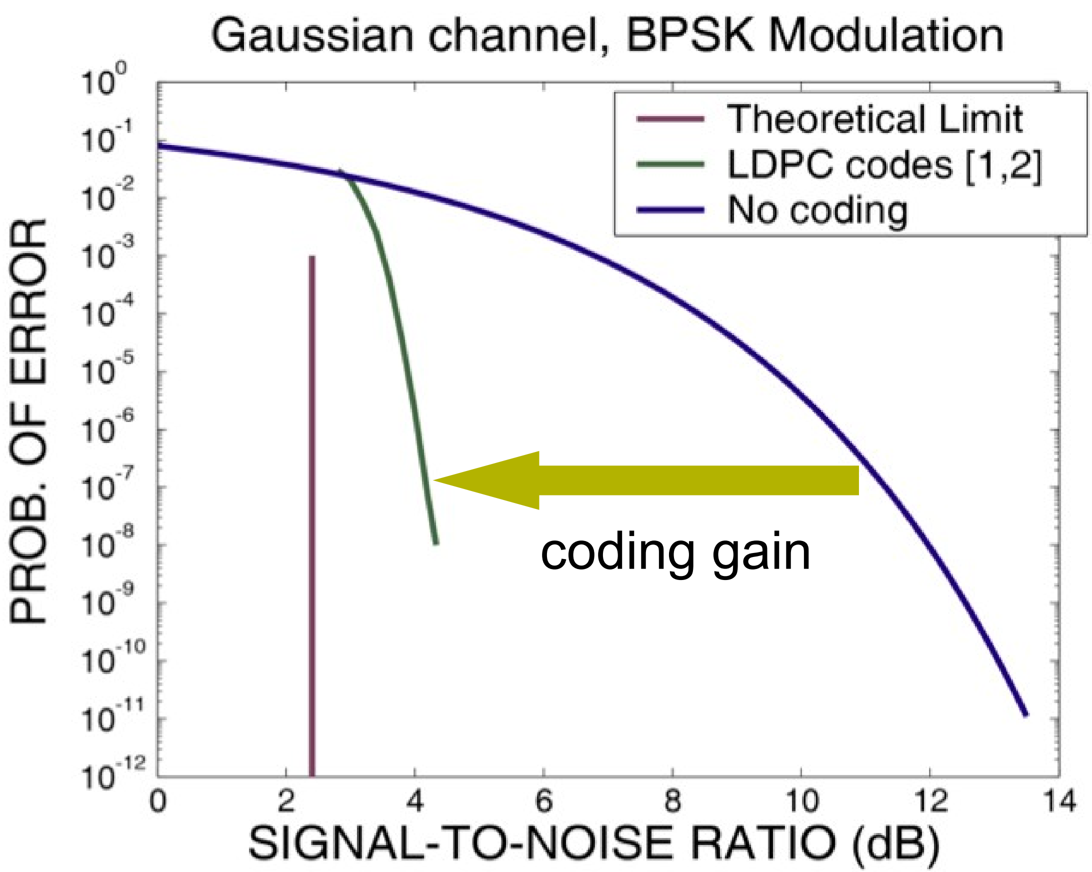
d_l	100		200		8000	
	x	λ_x	x	λ_x	x	λ_x
	2	0.170031	2	0.153425	2	0.096294
	3	0.160460	3	0.147526	3	0.095393
	6	0.112837	6	0.041539	6	0.033599
	7	0.047489	7	0.147551	7	0.091918
	10	0.011481	18	0.047938	15	0.031642
	11	0.091537	19	0.119555	20	0.086563
	26	0.152978	55	0.036379	50	0.093896
	27	0.036131	56	0.126714	70	0.006035
	100	0.217056	200	0.179373	100	0.018375
					150	0.086919
					400	0.089018
					900	0.057176
					2000	0.085816
					3000	0.006163
					6000	0.003028
					8000	0.118165
ρ_{av}	10.9375		12.0000		18.5000	
σ	0.97592		0.97704		0.9781869	
SNR_{norm}	0.0247		0.0147		0.00450	



[CFRU01] S.-Y. Chung, G. D. Forney, Jr., T. J. Richardson, and R. Urbanke, “On the Design of Low-Density Parity-Check Codes within 0.0045 dB of the Shannon Limit”, *IEEE Comm. Letters*, vol. 5 no. 2, Feb. 2001.

Error Floors of LDPC Codes

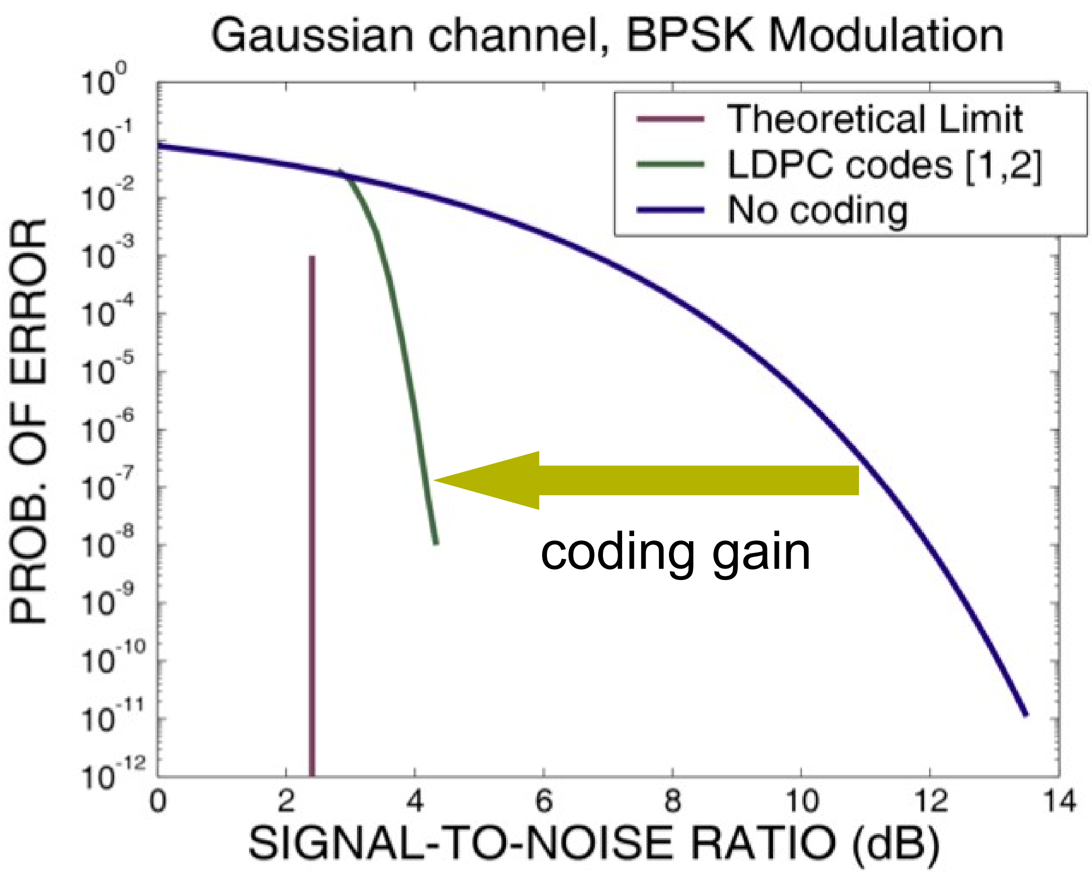
- Graph-based LDPC codes:



[1] Djurdevic, et al., 2003
[2] IEEE 802.3an

Error Floors of LDPC Codes

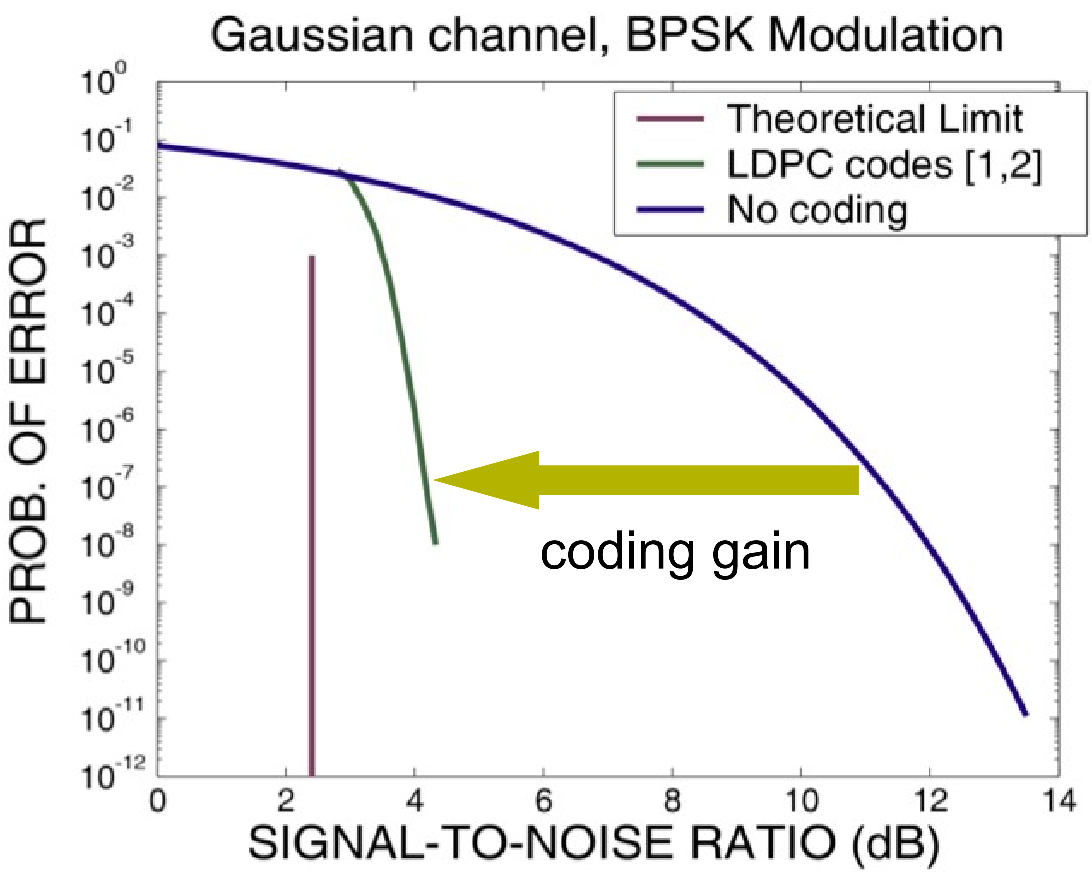
- Graph-based LDPC codes:
  - ✓ can approach capacity (at moderate error rates) asymptotically in codeword length



[1] Djurdevic, et al., 2003
[2] IEEE 802.3an

Error Floors of LDPC Codes

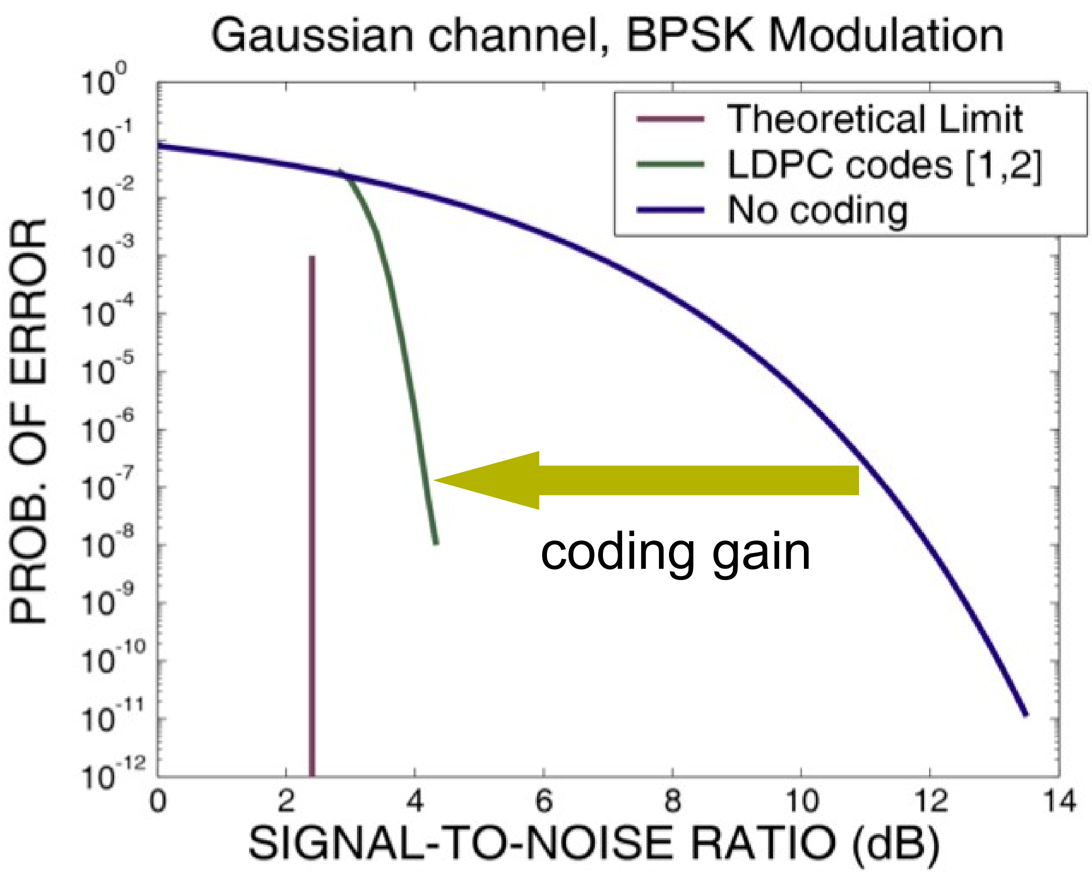
- Graph-based LDPC codes:
  - ✓ can approach capacity (at moderate error rates) asymptotically in codeword length
  - ✓ iterative decoding has low complexity



[1] Djurdevic, et al., 2003
[2] IEEE 802.3an

Error Floors of LDPC Codes

- Graph-based LDPC codes:
  - ✓ can approach capacity (at moderate error rates) asymptotically in codeword length
  - ✓ iterative decoding has low complexity
- Not known:

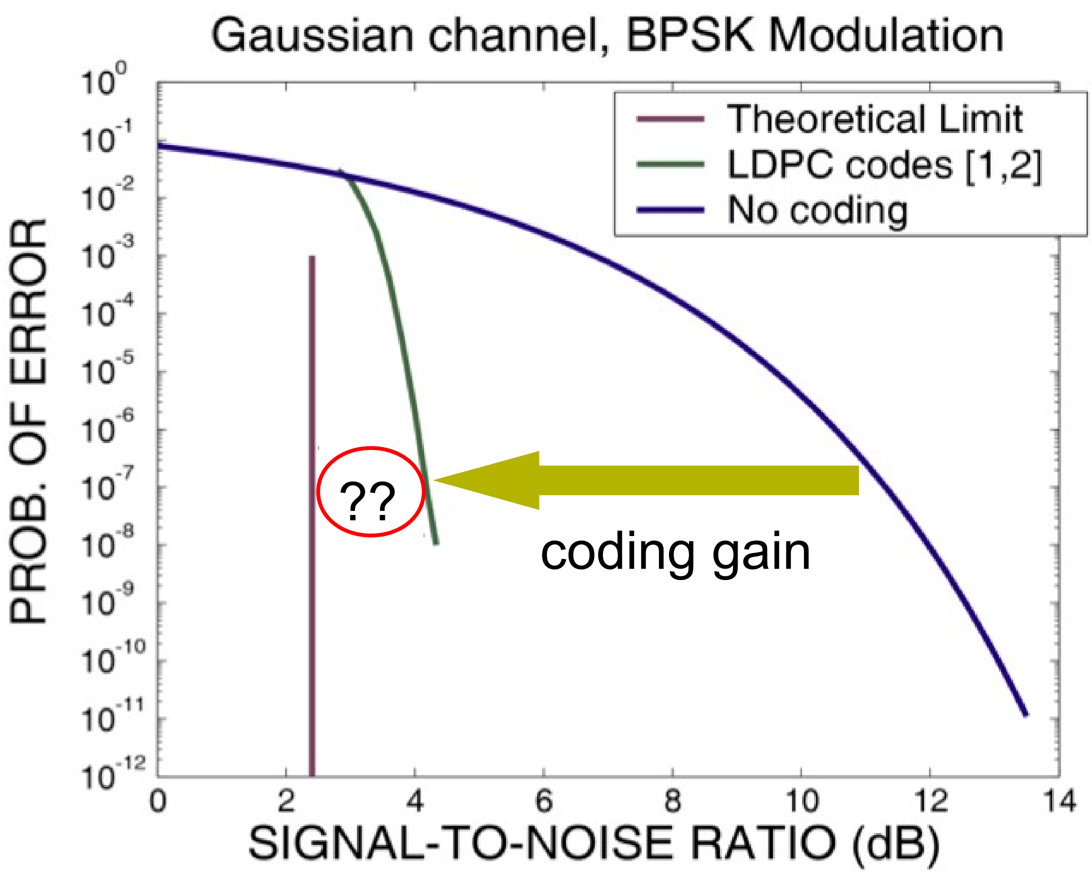


[1] Djurdevic, et al., 2003
[2] IEEE 802.3an

Error Floors of LDPC Codes

UNIVERSITY OF NOTRE DAME

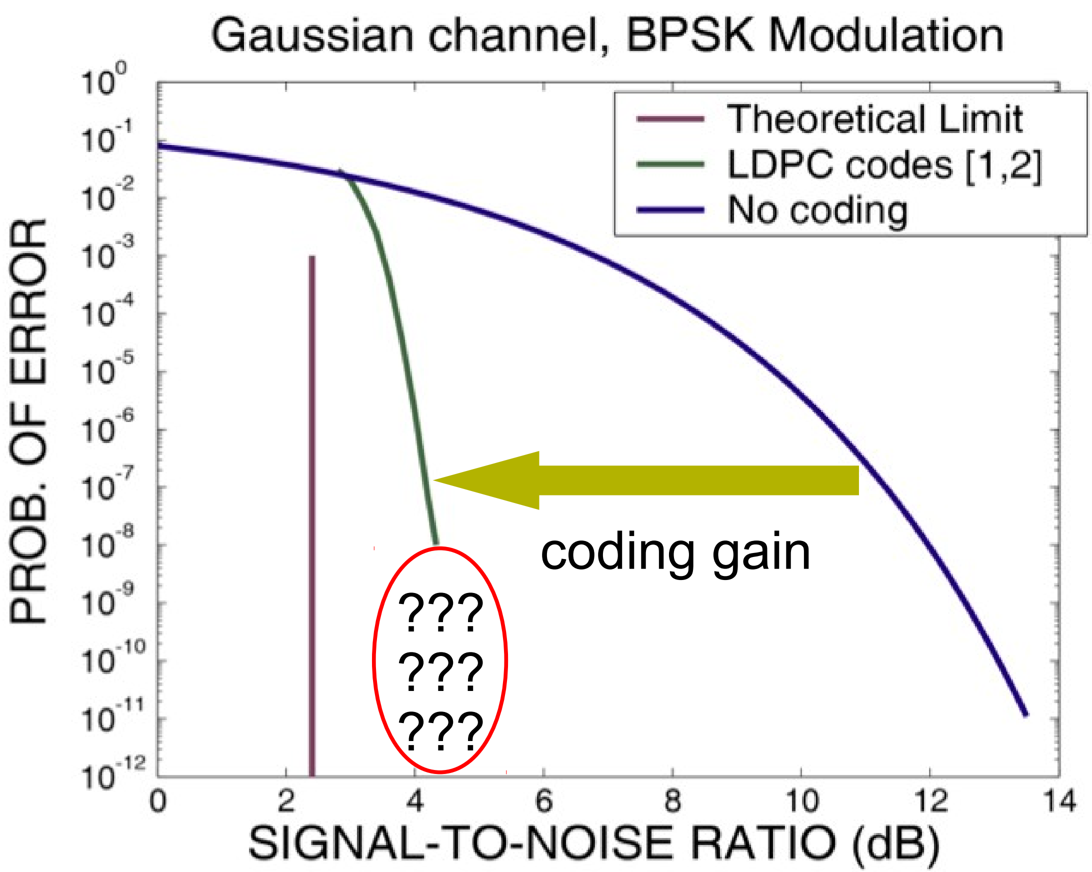
- Graph-based LDPC codes:
  - ✓ can approach capacity (at moderate error rates) asymptotically in codeword length
  - ✓ iterative decoding has low complexity
- Not known:
  - X how to approach capacity with acceptable implementation complexity and latency



[1] Djurdevic, et al., 2003
[2] IEEE 802.3an

Error Floors of LDPC Codes

- Graph-based LDPC codes:
  - ✓ can approach capacity (at moderate error rates) asymptotically in codeword length
  - ✓ iterative decoding has low complexity
- Not known:
  - X how to approach capacity with acceptable implementation complexity and latency
  - X finite length code performance for very low error rates

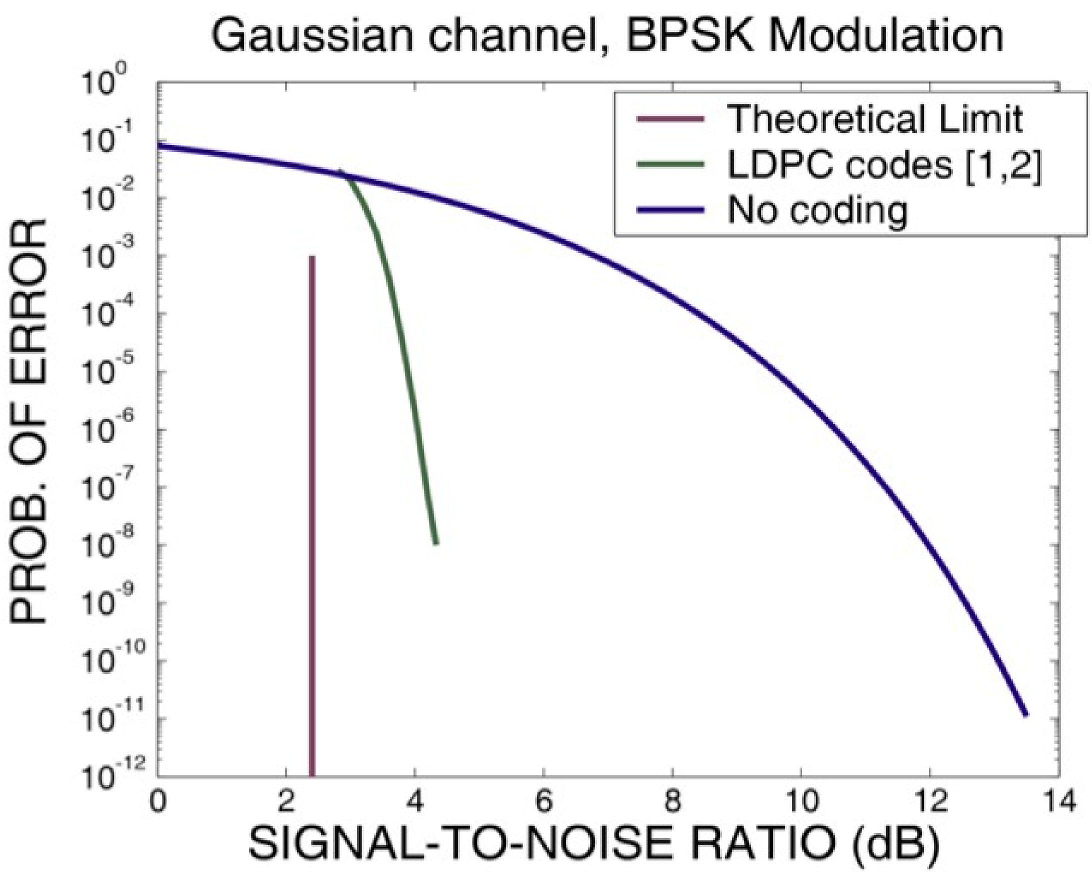


[1] Djurdevic, et al., 2003
[2] IEEE 802.3an

What is hard about LDPC codes?

UNIVERSITY OF NOTRE DAME

- For finite length codes:
  - ➡ understanding the iterative algorithm is not easy (asymptotic performance is known)

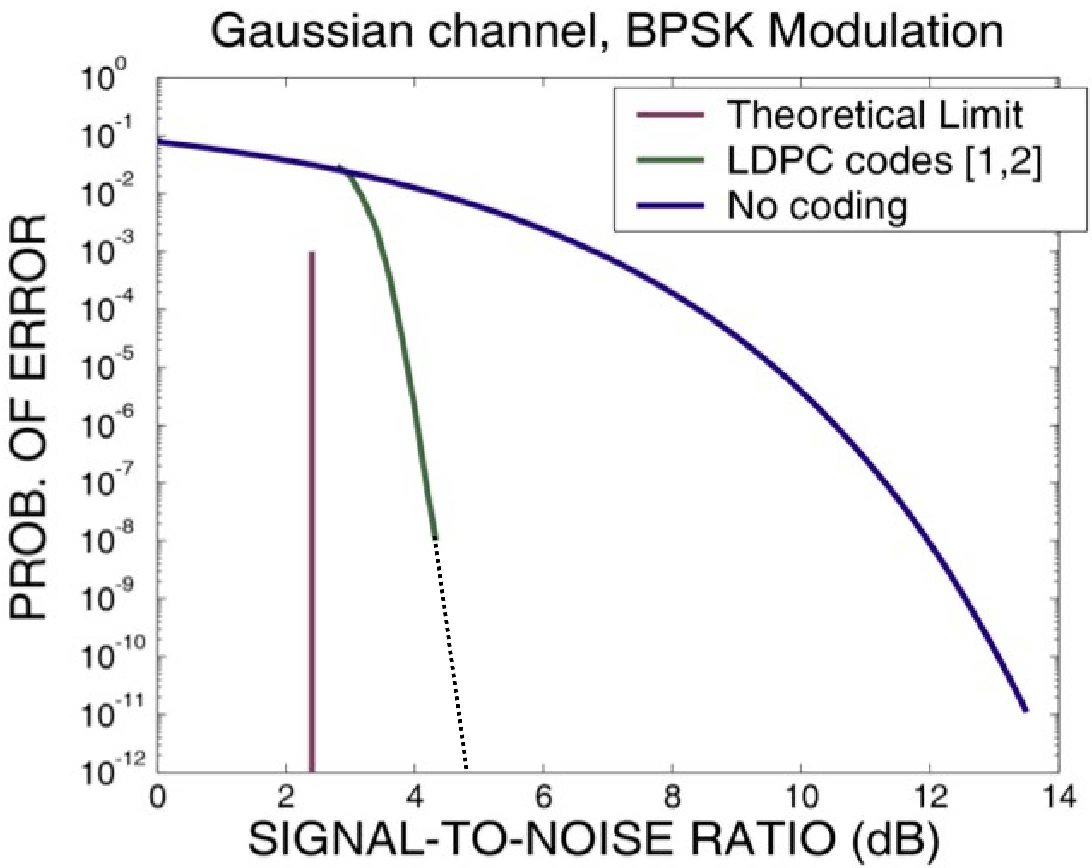


[1] Djurdevic, et al., 2003
[2] IEEE 802.3an

What is hard about LDPC codes?

UNIVERSITY OF NOTRE DAME

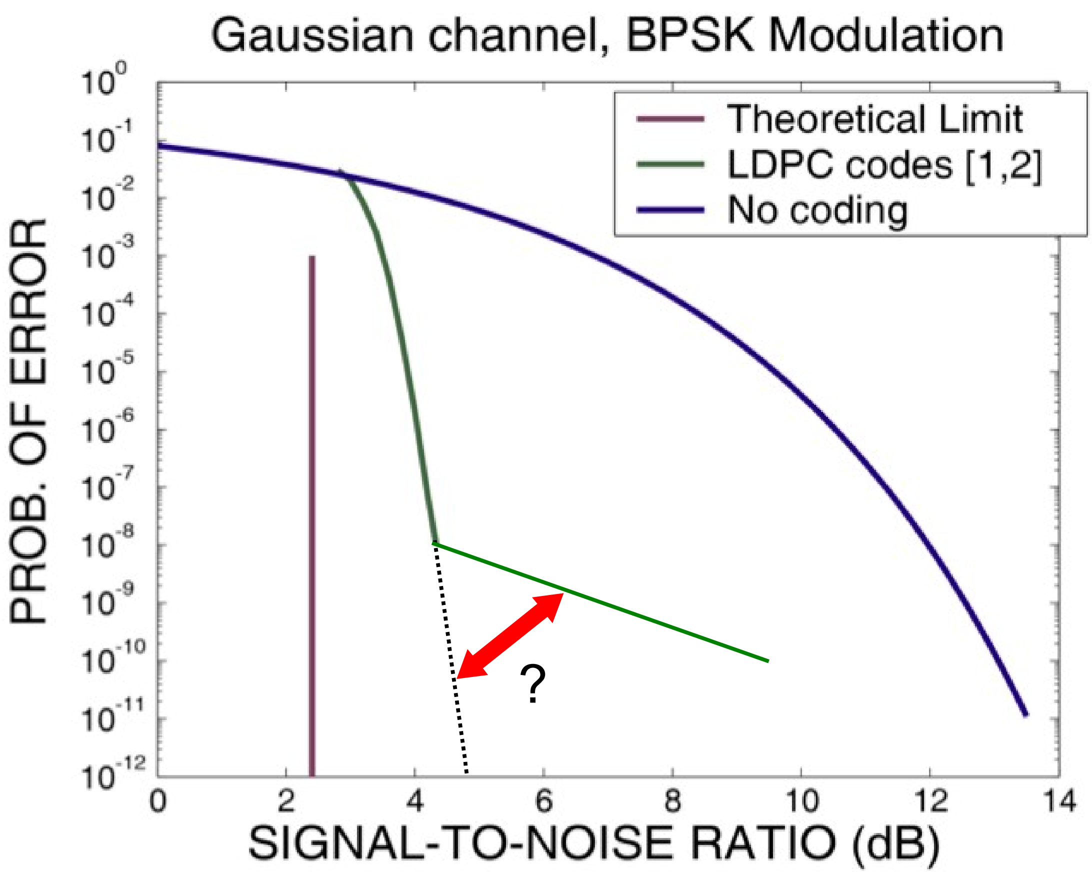
- For finite length codes:
  - ➡ understanding the iterative algorithm is not easy (asymptotic performance is known)



[1] Djurdevic, et al., 2003
[2] IEEE 802.3an

What is hard about LDPC codes?

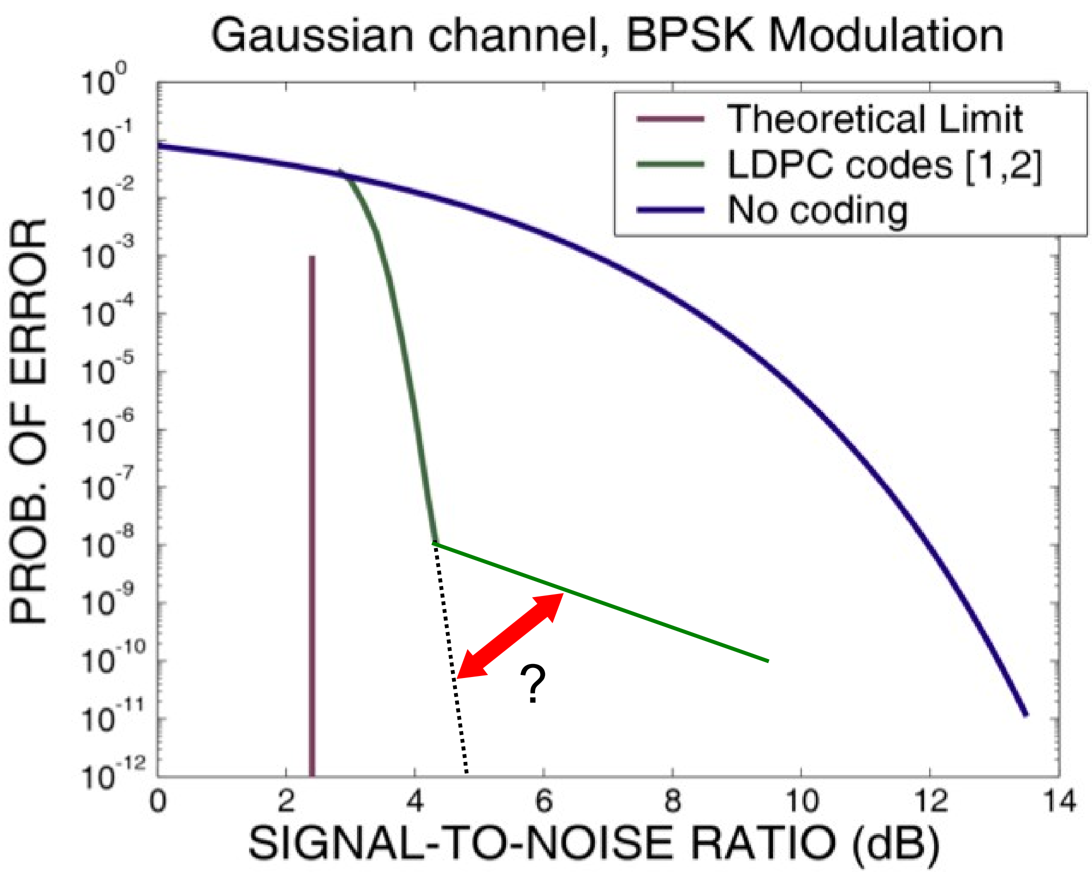
- For finite length codes:
  - ➡ understanding the iterative algorithm is not easy (asymptotic performance is known)



[1] Djurdevic, et al., 2003
[2] IEEE 802.3an

What is hard about LDPC codes?

- For finite length codes:
  - ➡ understanding the iterative algorithm is not easy (asymptotic performance is known)
- What to do?
  - ➡ use Monte Carlo simulation

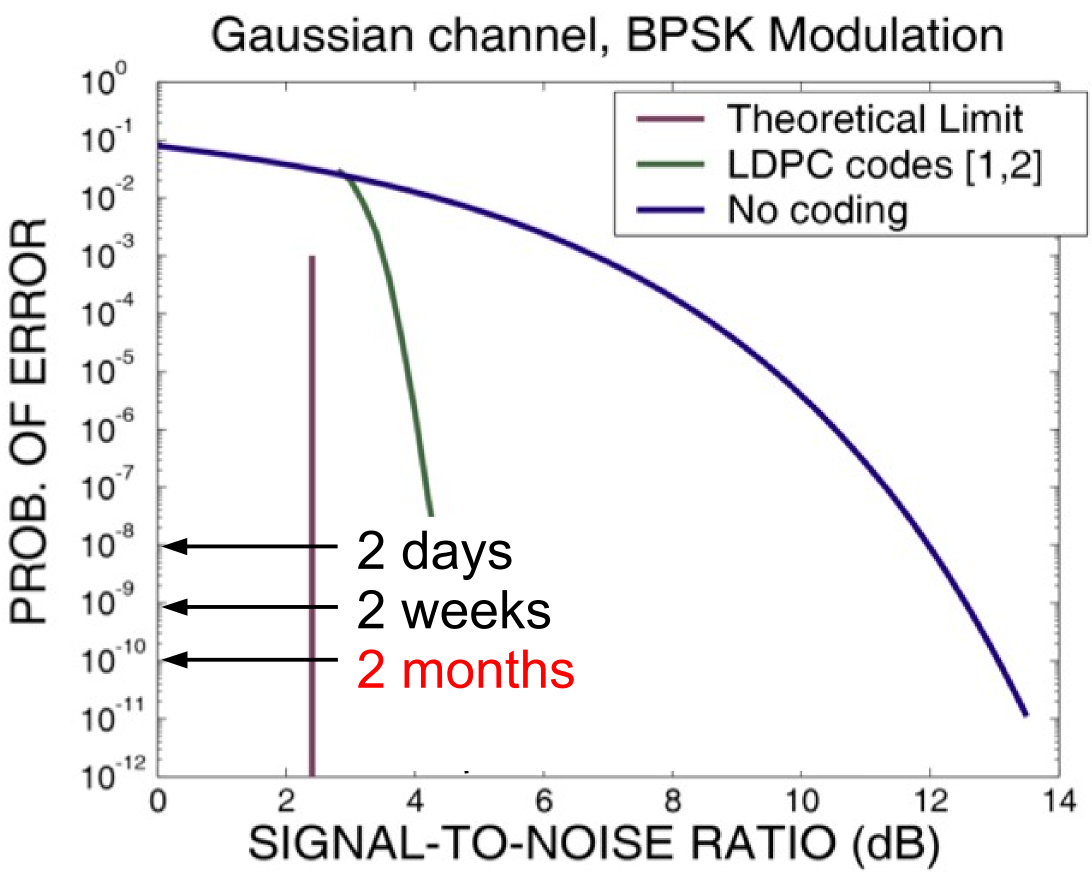


[1] Djurdevic, et al., 2003
[2] IEEE 802.3an

What is hard about LDPC codes?

UNIVERSITY OF NOTRE DAME

- For finite length codes:
  - ➡ understanding the iterative algorithm is not easy (asymptotic performance is known)
- What to do?
  - ➡ use Monte Carlo simulation
  - X computational cutoff

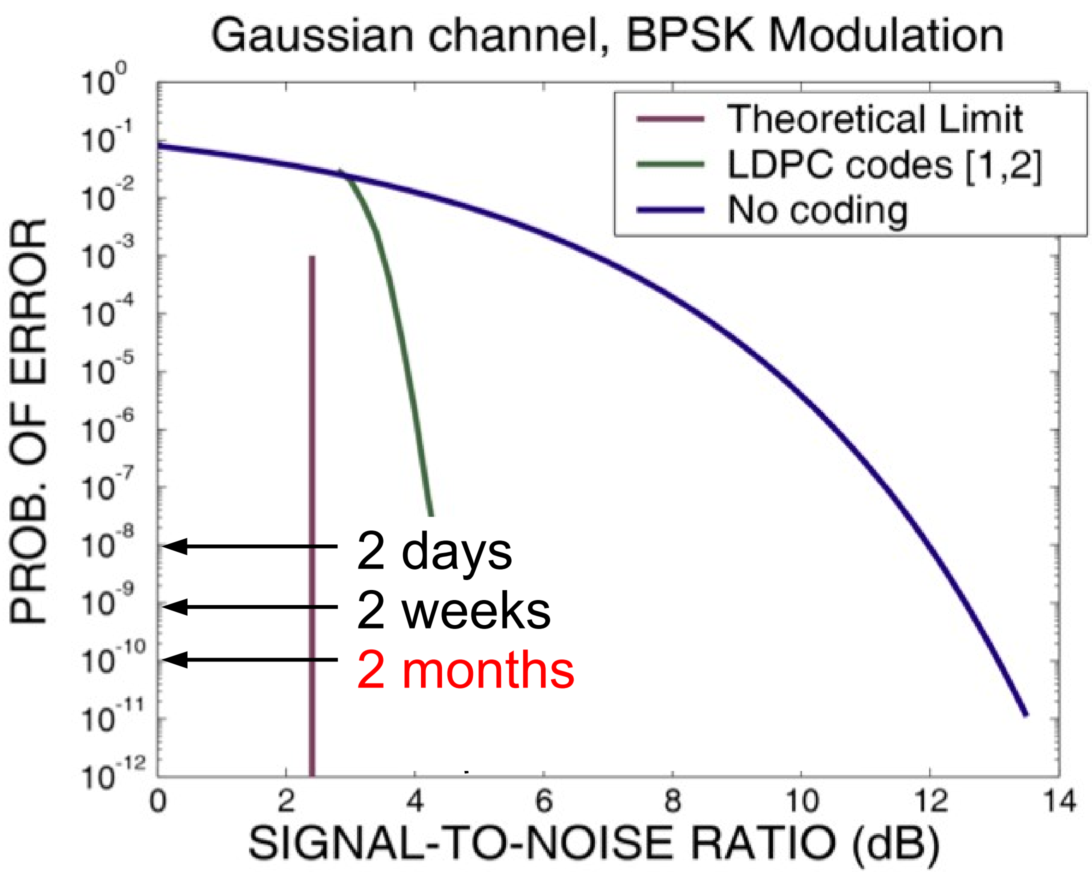


[1] Djurdevic, et al., 2003
[2] IEEE 802.3an

What is hard about LDPC codes?

UNIVERSITY OF NOTRE DAME

- For finite length codes:
  - ➡ understanding the iterative algorithm is not easy (asymptotic performance is known)
- What to do?
  - ➡ use Monte Carlo simulation
    - X computational cutoff
  - ➡ hardware emulation

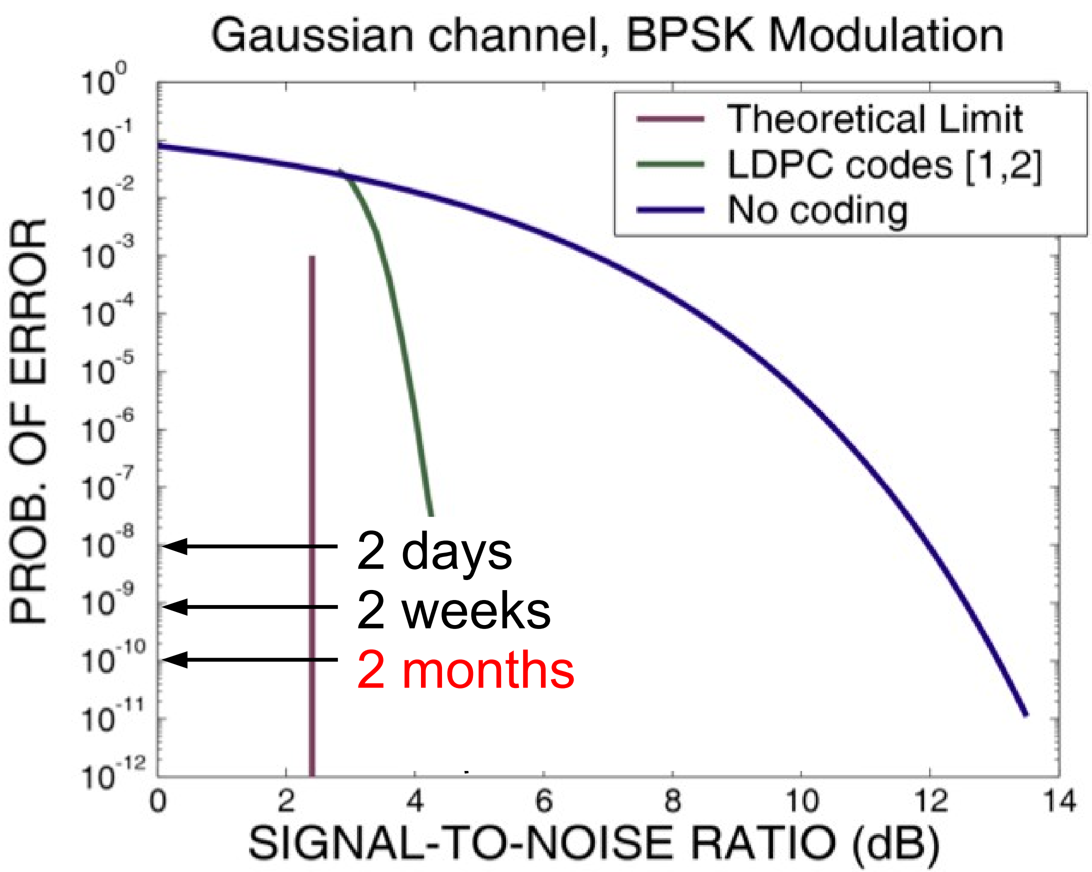


[1] Djurdevic, et al., 2003
[2] IEEE 802.3an

What is hard about LDPC codes?

UNIVERSITY OF NOTRE DAME

- For finite length codes:
  - ➡ understanding the iterative algorithm is not easy (asymptotic performance is known)
- What to do?
  - ➡ use Monte Carlo simulation
    - X computational cutoff
  - ➡ hardware emulation
    - X expensive

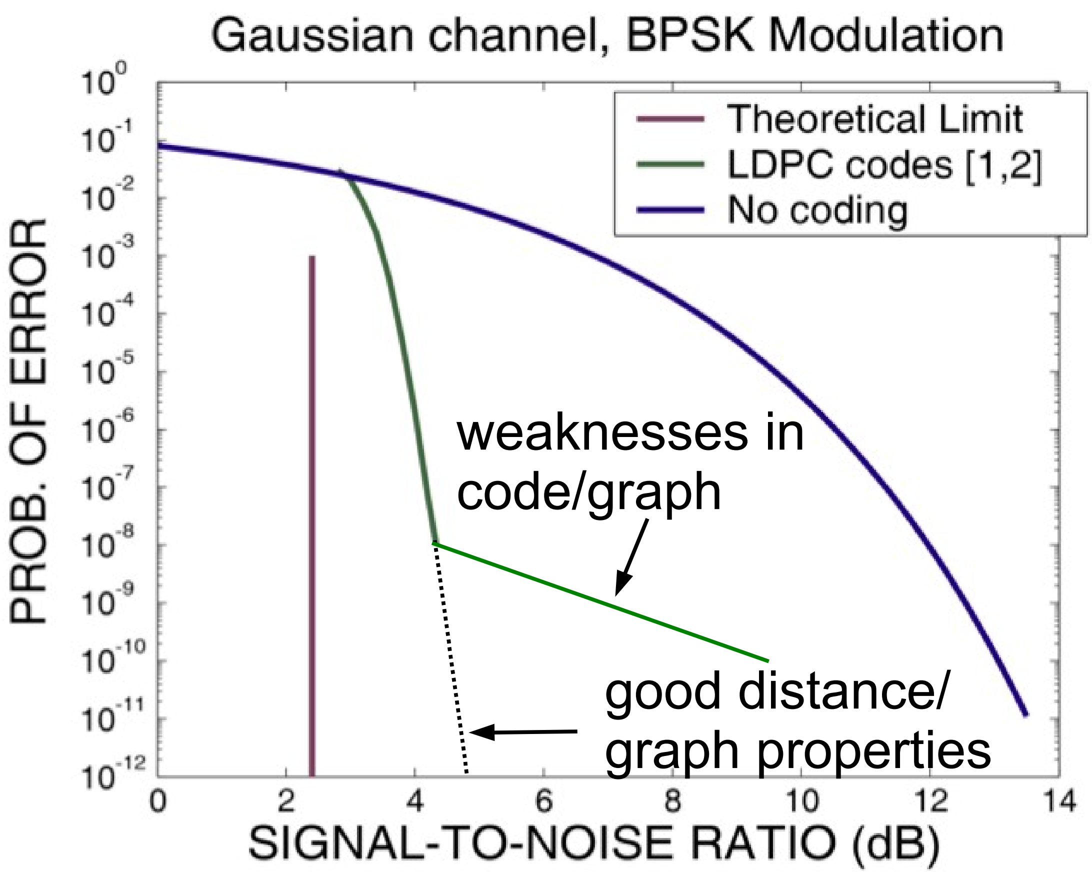


[1] Djurdevic, et al., 2003
[2] IEEE 802.3an

What is hard about LDPC codes?

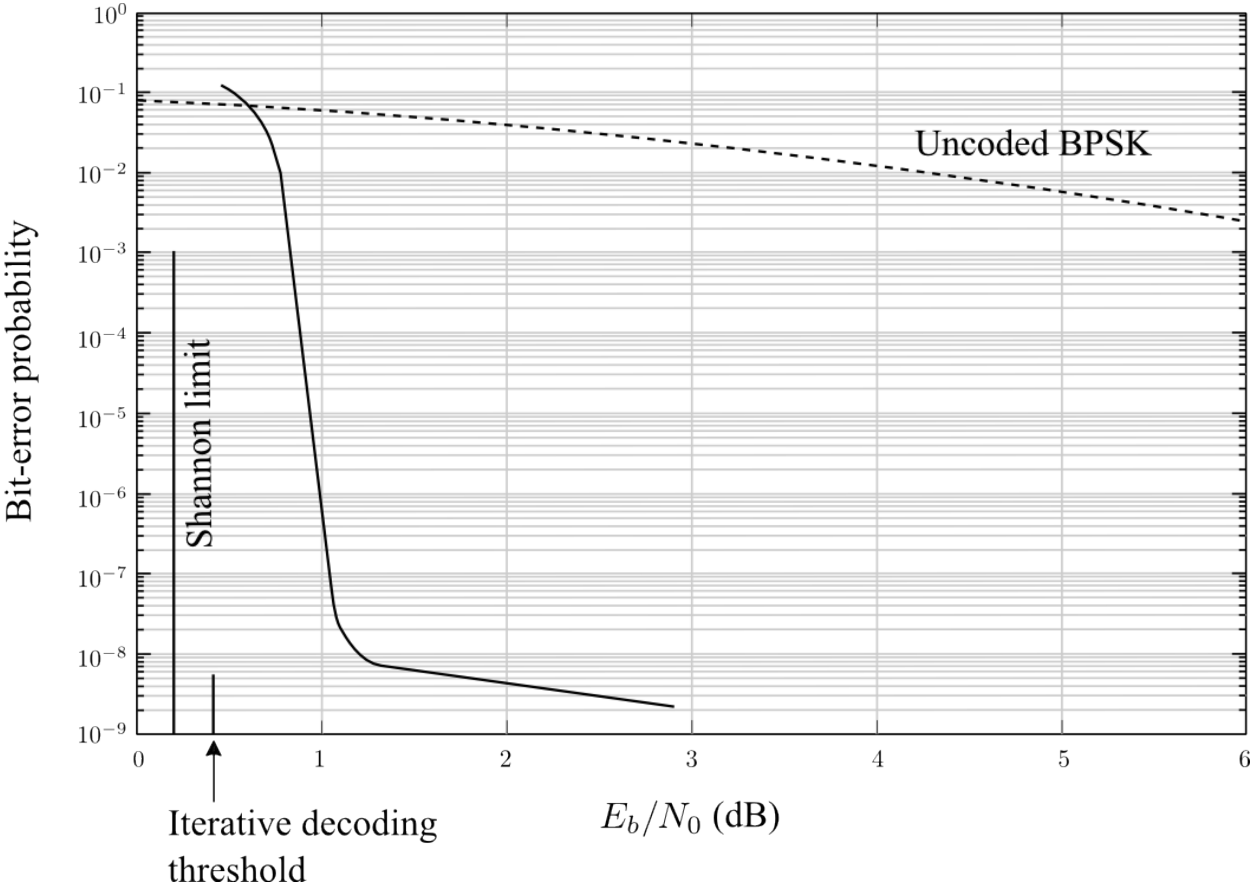
UNIVERSITY OF NOTRE DAME

- For finite length codes:
  - ➡ understanding the iterative algorithm is not easy (asymptotic performance is known)
- What to do?
  - ➡ use Monte Carlo simulation
    - X computational cutoff
  - ➡ hardware emulation
    - X expensive
  - ➡ analysis (performance guarantees)



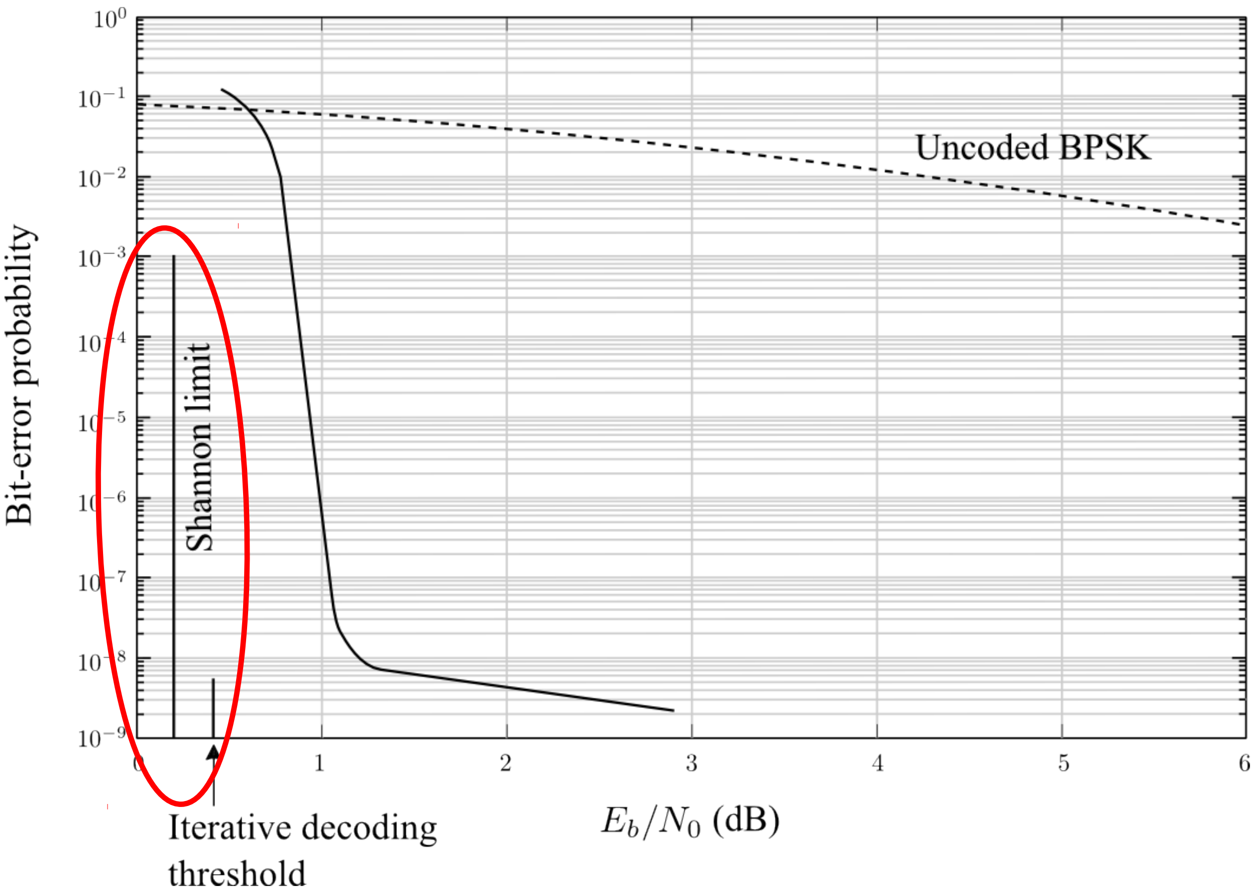
[1] Djurdevic, et al., 2003
[2] IEEE 802.3an

Typical LDPC Code Behavior



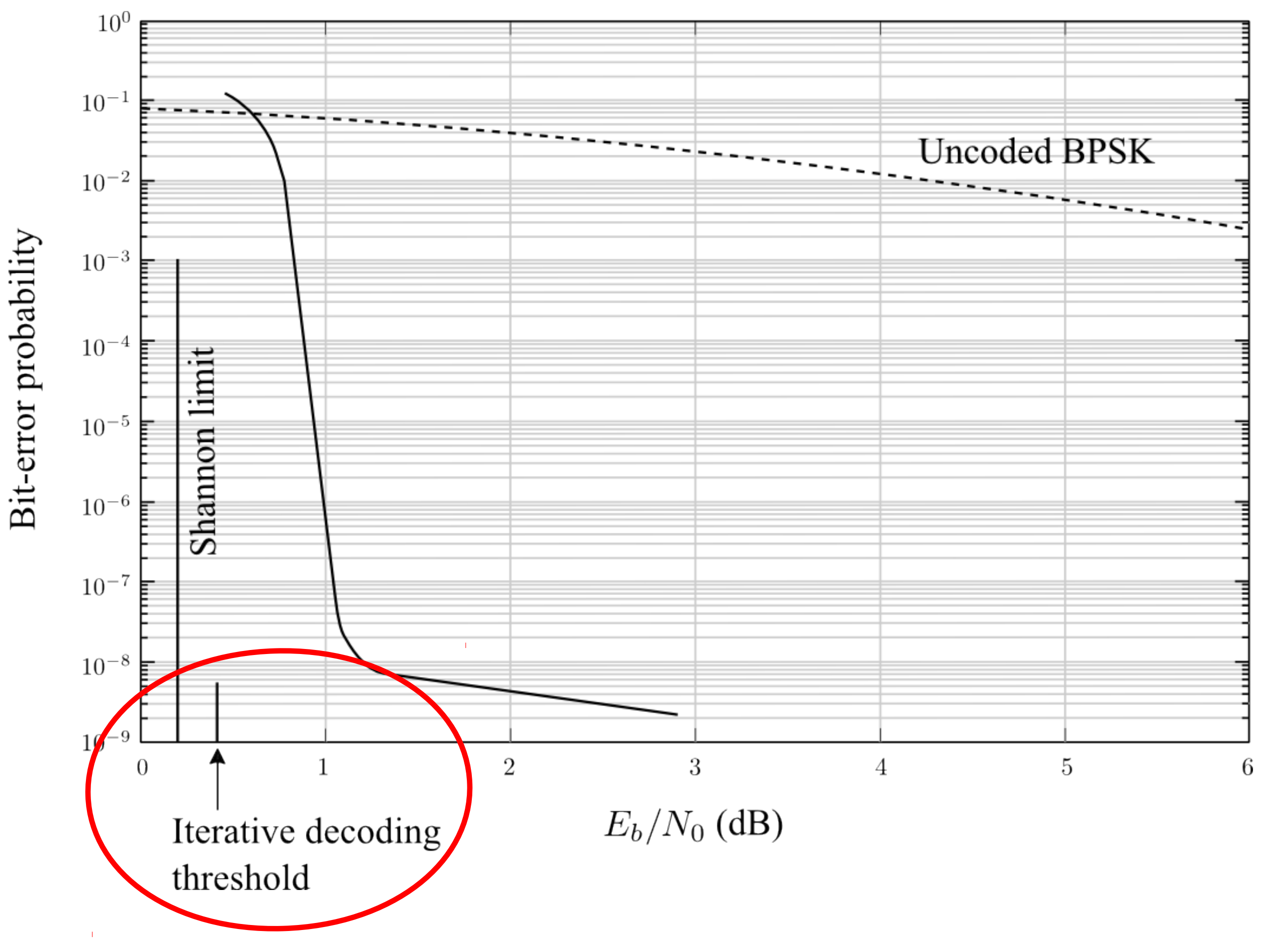
Typical LDPC Code Behavior

- The **Shannon limit** (capacity) is a property of the physical channel.



Typical LDPC Code Behavior

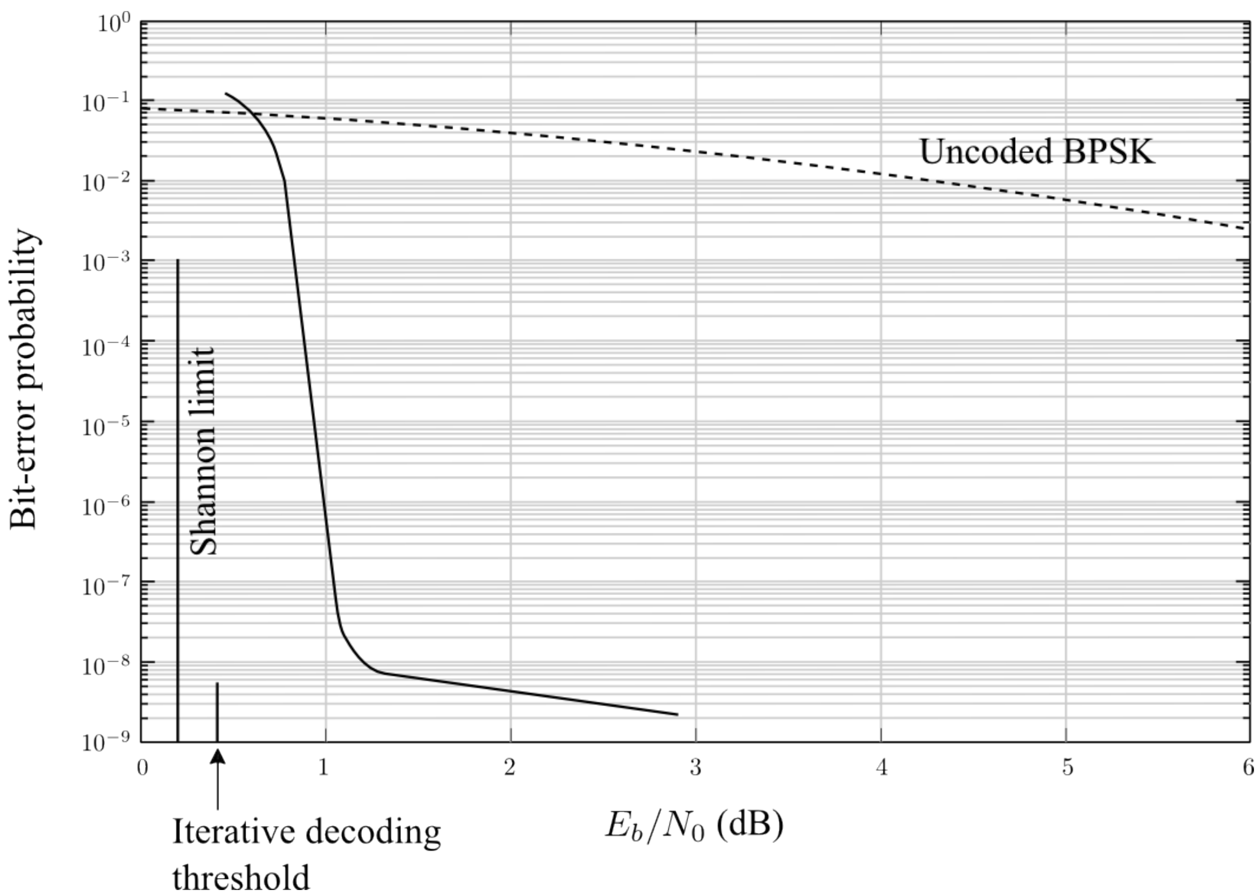
- The **Shannon limit** (capacity) is a property of the physical channel.



- The **iterative decoding threshold** depends on:
  1) the code ensemble
  2) the decoding algorithm in use

Typical LDPC Code Behavior

- The **Shannon limit** (capacity) is a property of the physical channel.

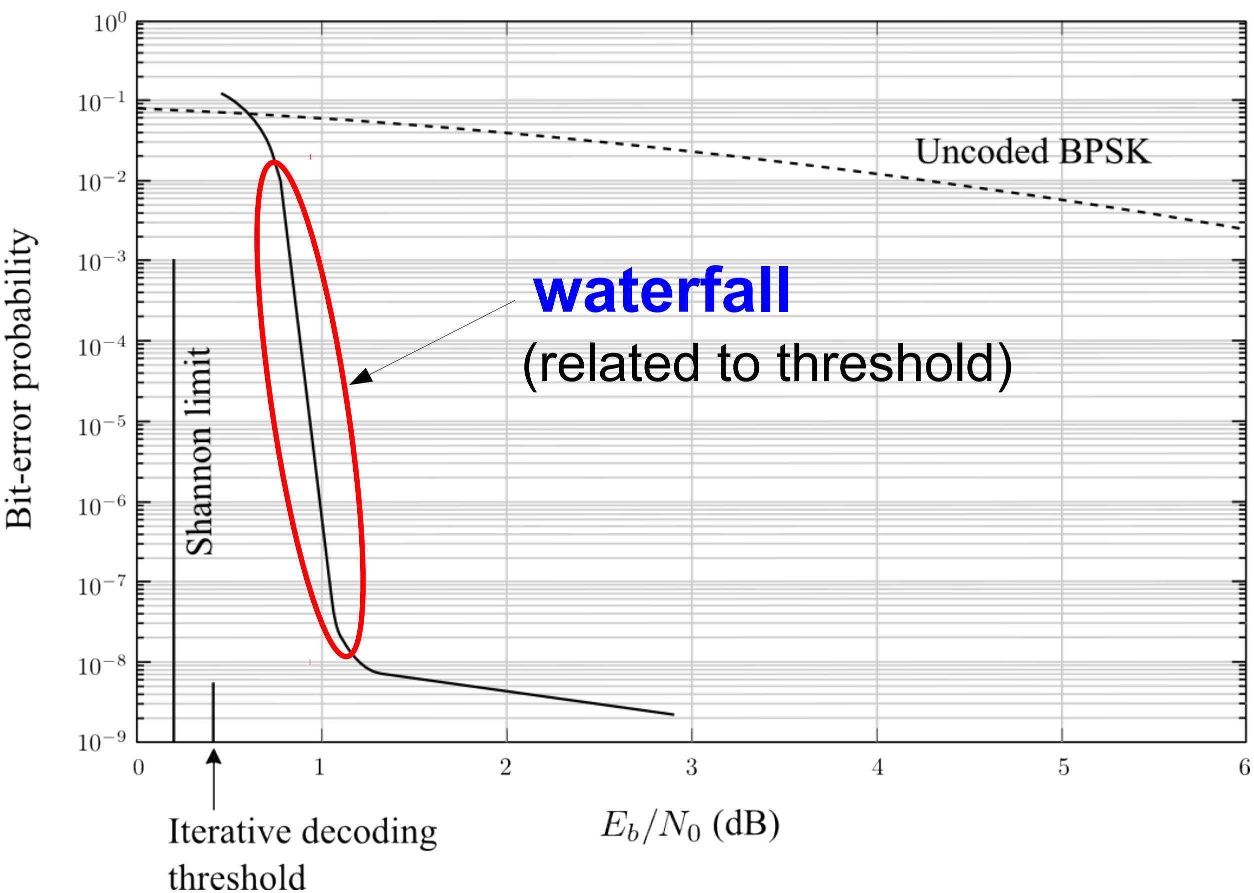


- The **iterative decoding threshold** depends on:
  1) the code ensemble
  2) the decoding algorithm in use

- With iterative decoding, codes typically display:

Typical LDPC Code Behavior

- The **Shannon limit** (capacity) is a property of the physical channel.



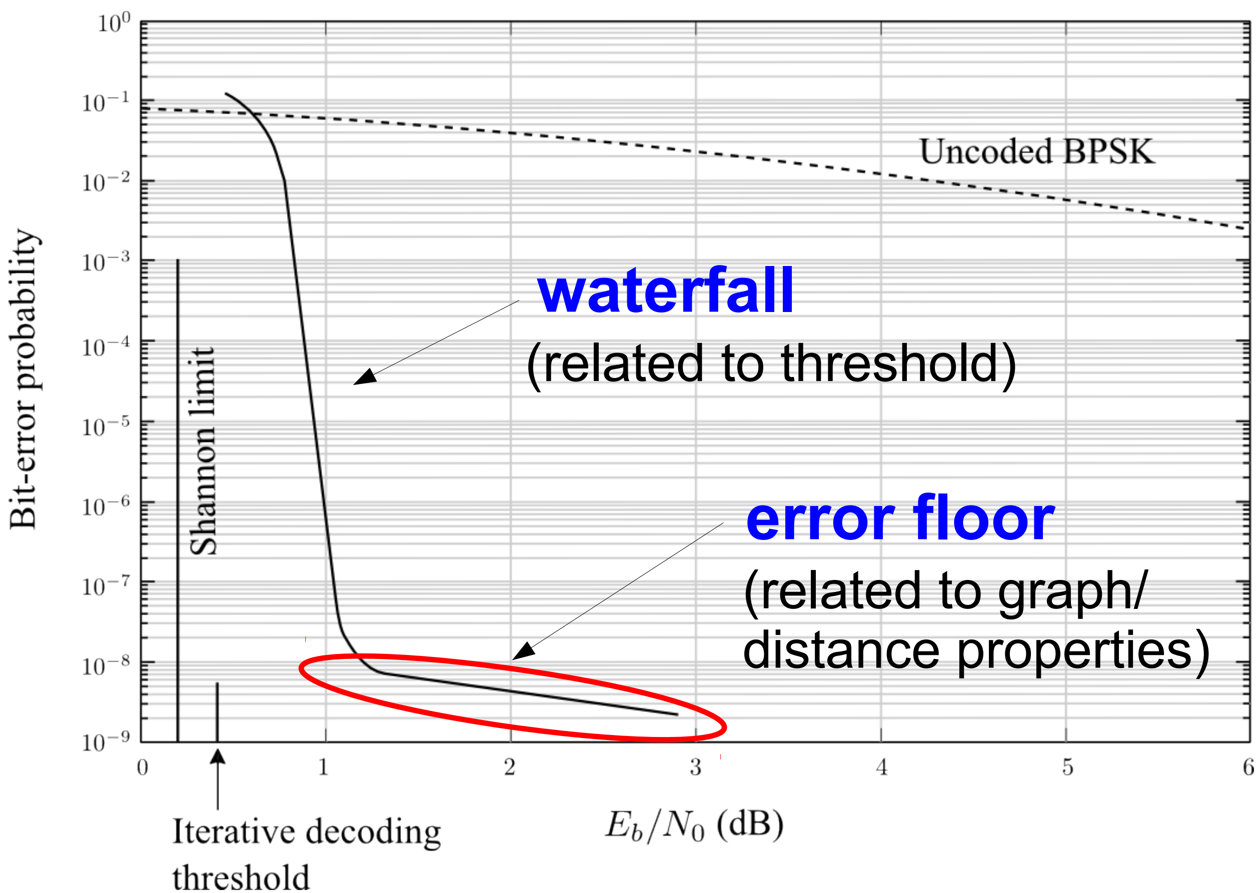
- The **iterative decoding threshold** depends on:
  1) the code ensemble
  2) the decoding algorithm in use

- With iterative decoding, codes typically display:

  a **waterfall**

Typical LDPC Code Behavior

- The **Shannon limit** (capacity) is a property of the physical channel.



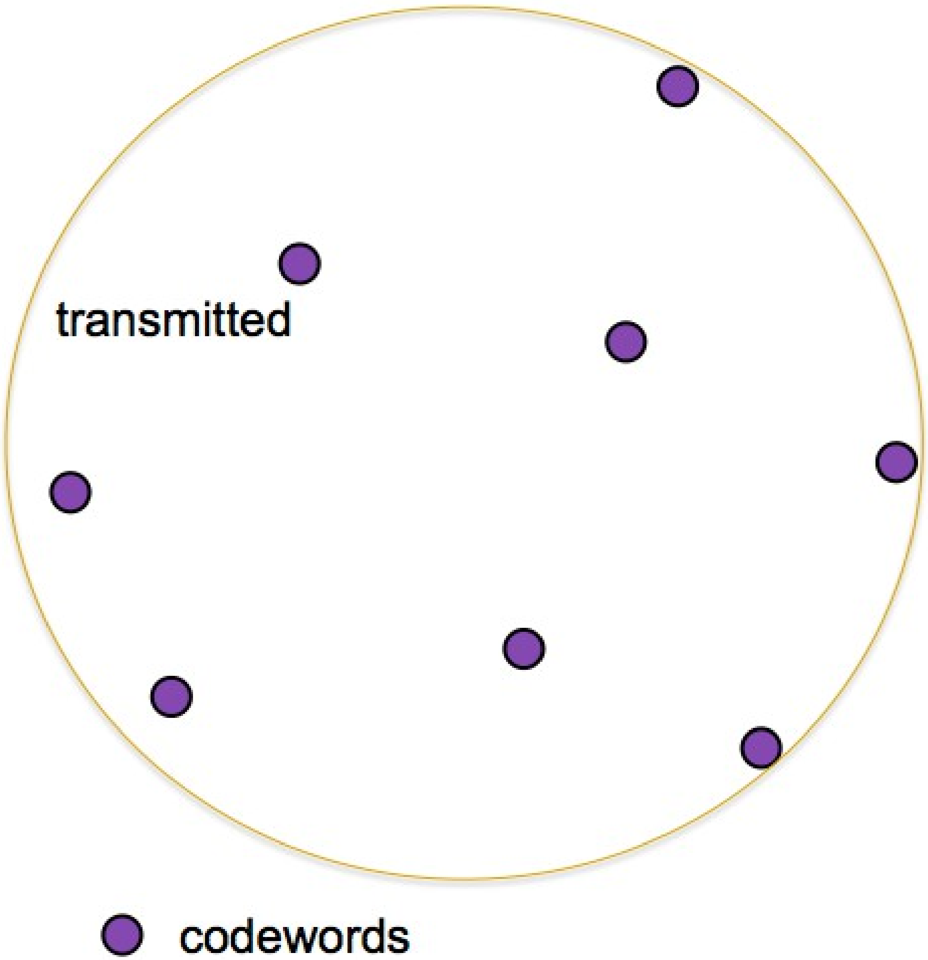
- The **iterative decoding threshold** depends on:
  1) the code ensemble
  2) the decoding algorithm in use

- With iterative decoding, codes typically display:

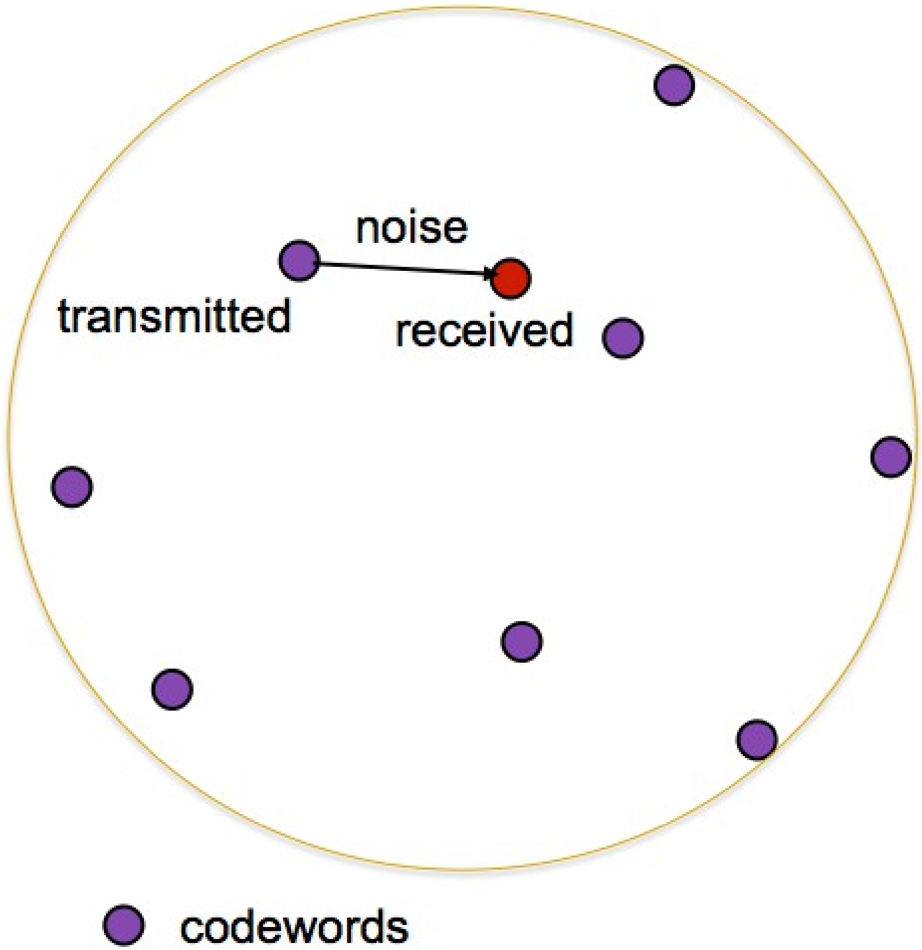
a **waterfall**

an **error floor**

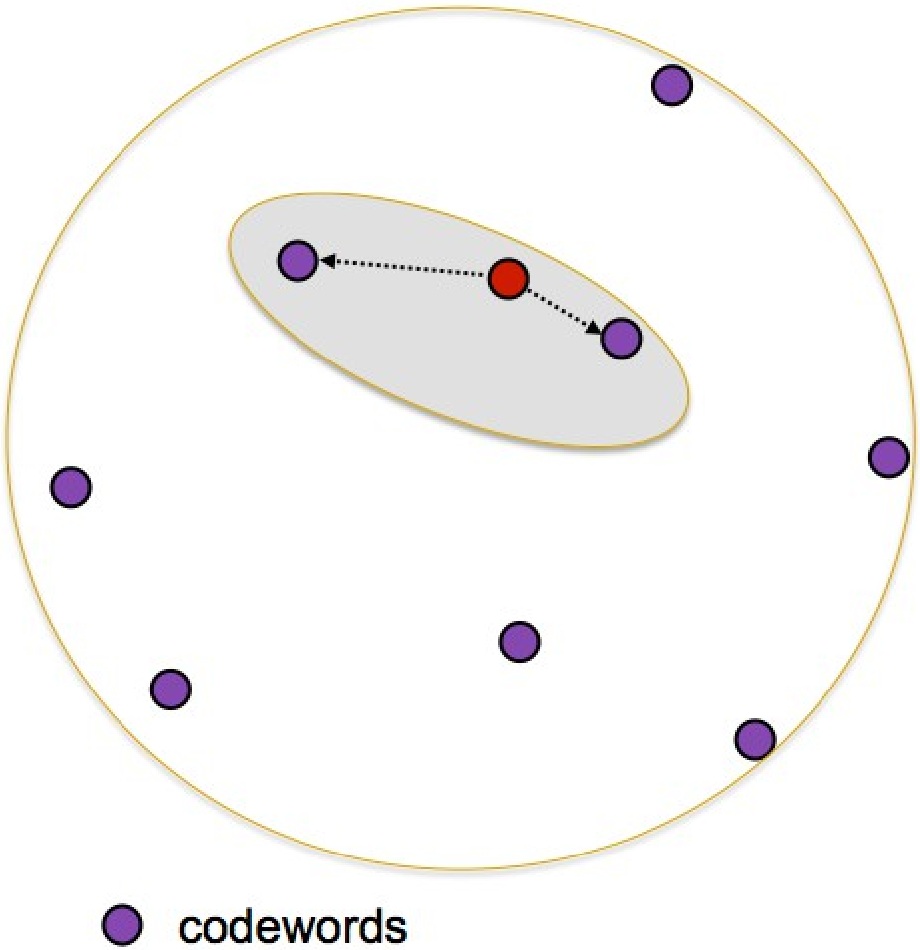
What causes decoding errors?



What causes decoding errors?



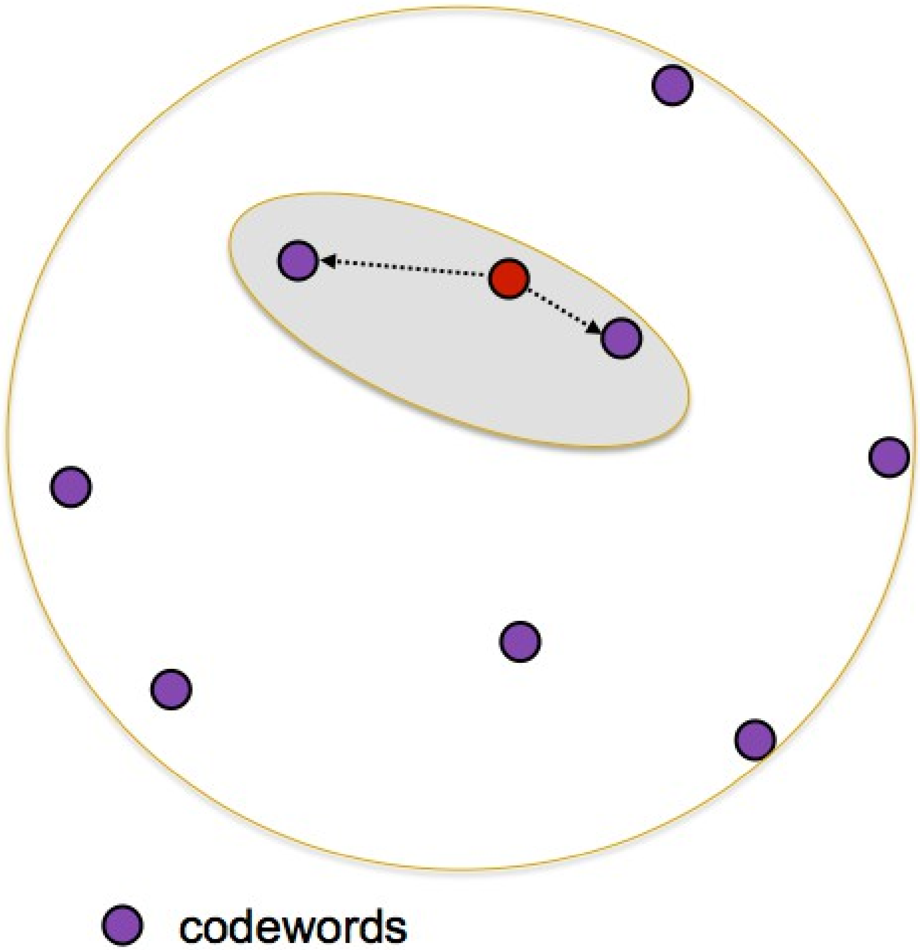
What causes decoding errors?



- Optimal choice:
- What causes errors?
- What errors are dominant?

What causes decoding errors?

UNIVERSITY OF NOTRE DAME



- Optimal choice:
  - ➡ Nearest neighbor (maximum likelihood)

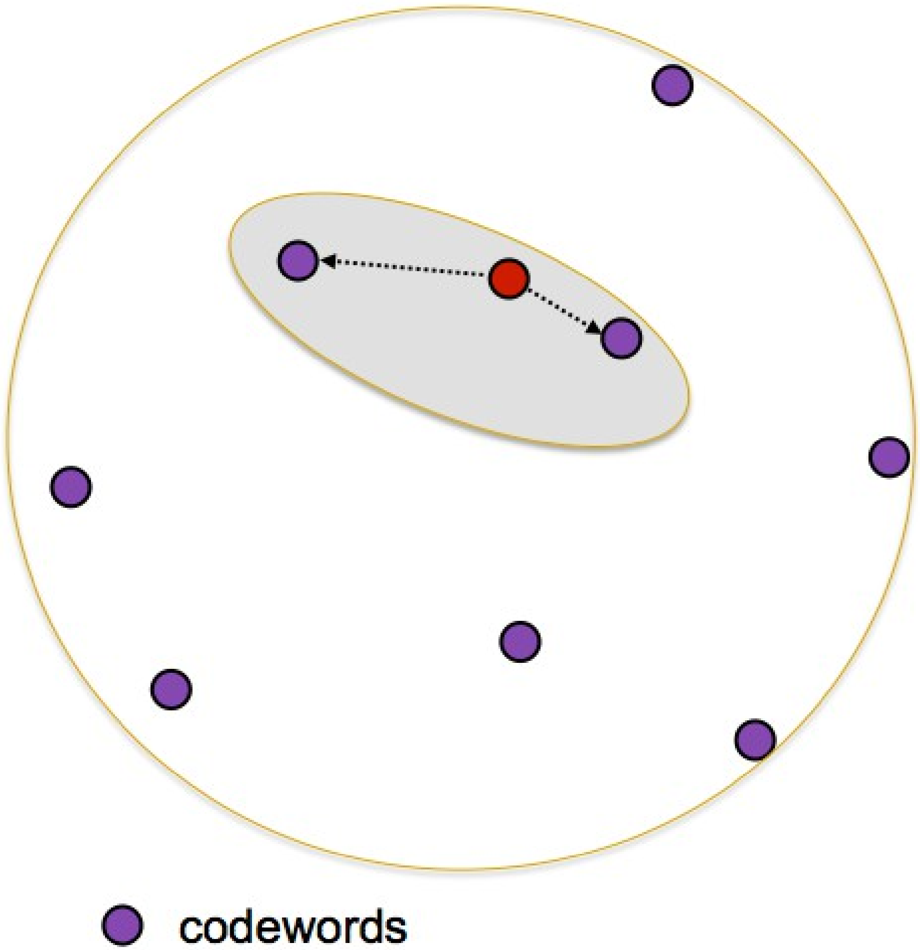
  high complexity

- What causes errors?

- What errors are dominant?

What causes decoding errors?

UNIVERSITY OF NOTRE DAME

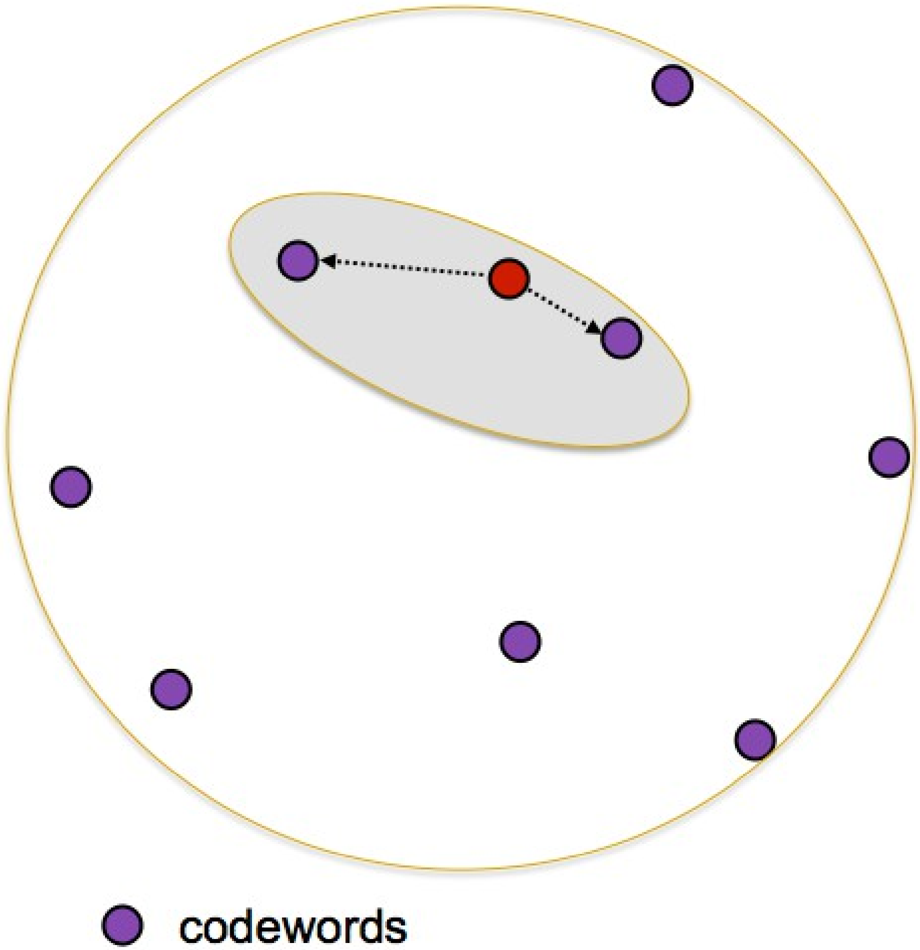


- Optimal choice:
  - ➡ Nearest neighbor (maximum likelihood)

    high complexity

- What causes errors?
  - ➡ Bit flips that cause the received word to be closer to another codeword
- What errors are dominant?

What causes decoding errors?



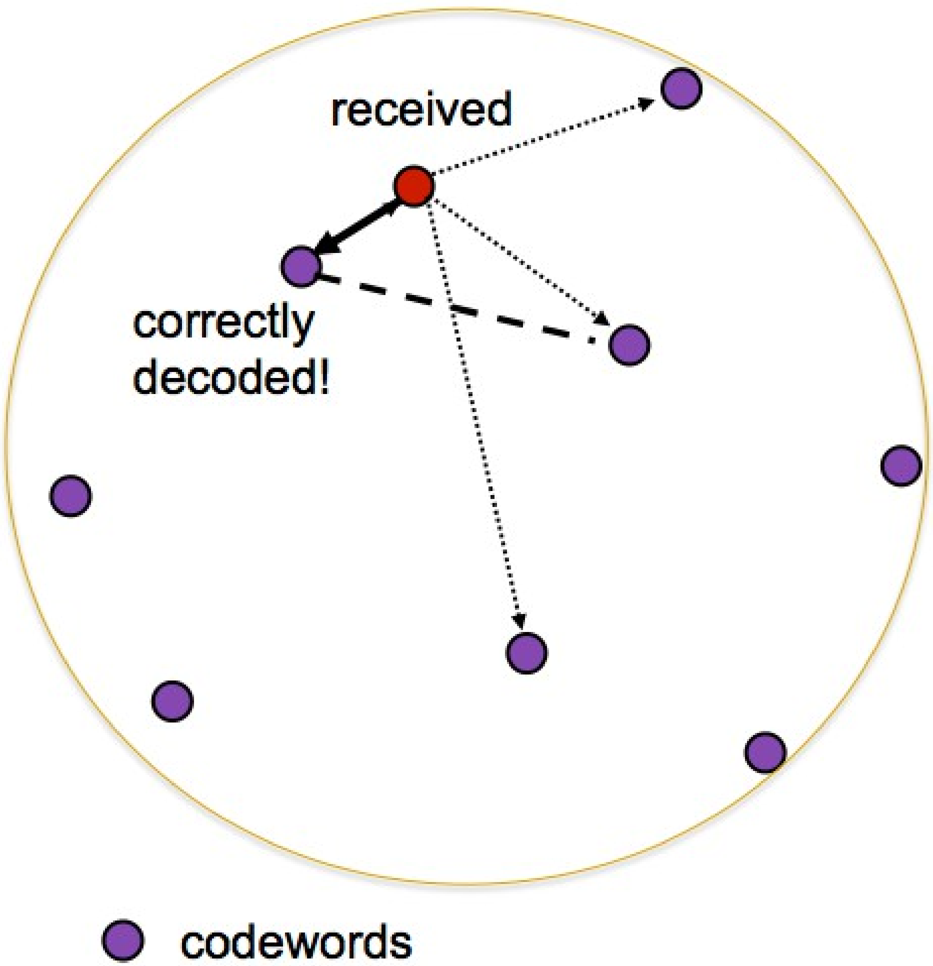
- Optimal choice:
  - ➡ Nearest neighbor (maximum likelihood)

  high complexity 

- What causes errors?
  - ➡ Bit flips that cause the received word to be closer to another codeword
- What errors are dominant?
  - ➡ Fewest bit flips

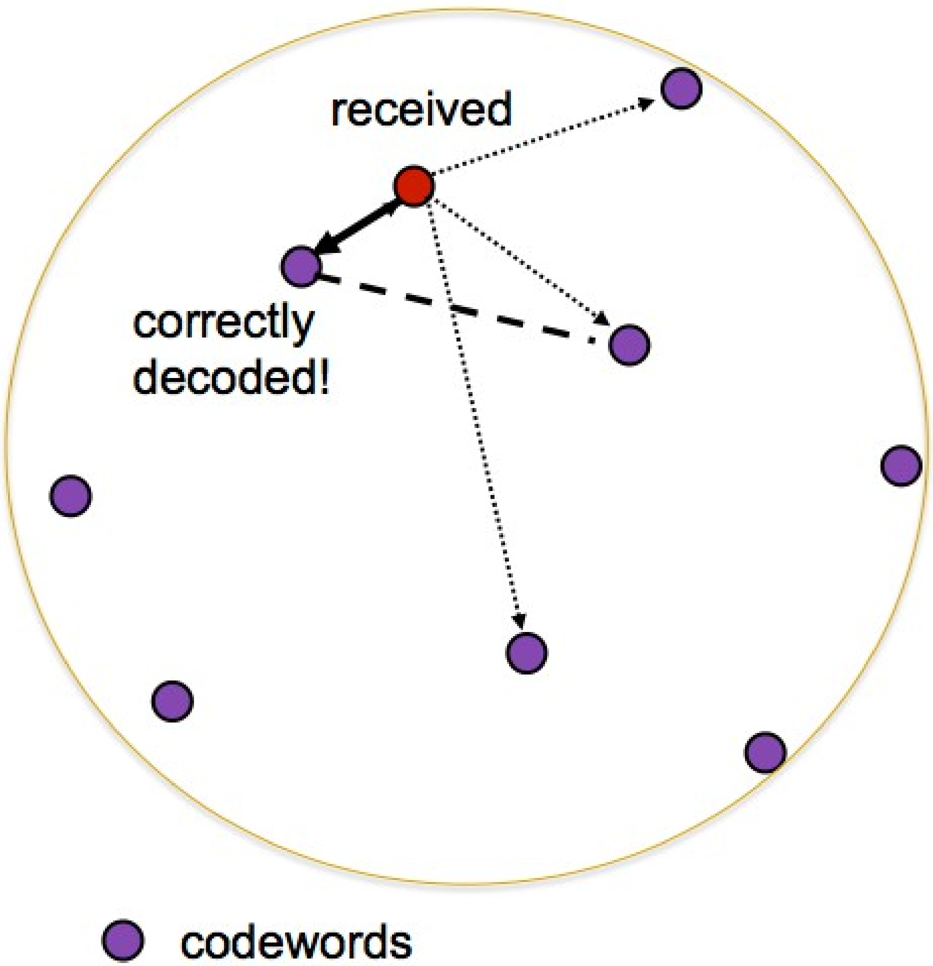
What causes decoding errors?

UNIVERSITY OF NOTRE DAME



- The minimum distance is the minimum separation between any two codewords

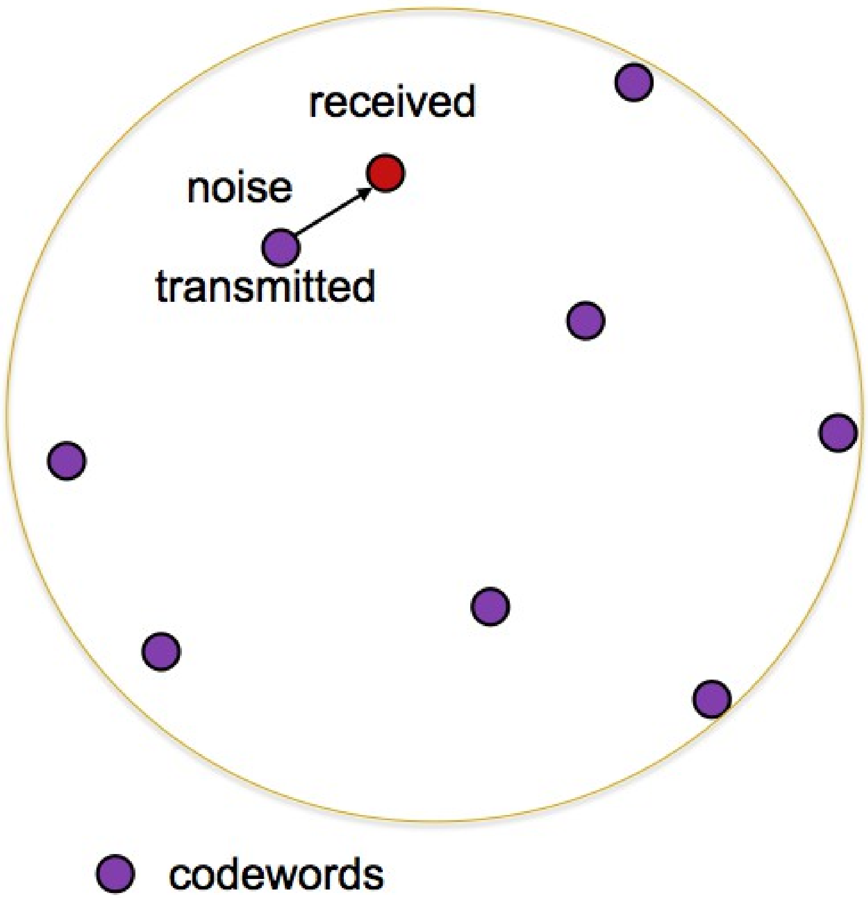
What causes decoding errors?



- The minimum distance is the minimum separation between any two codewords

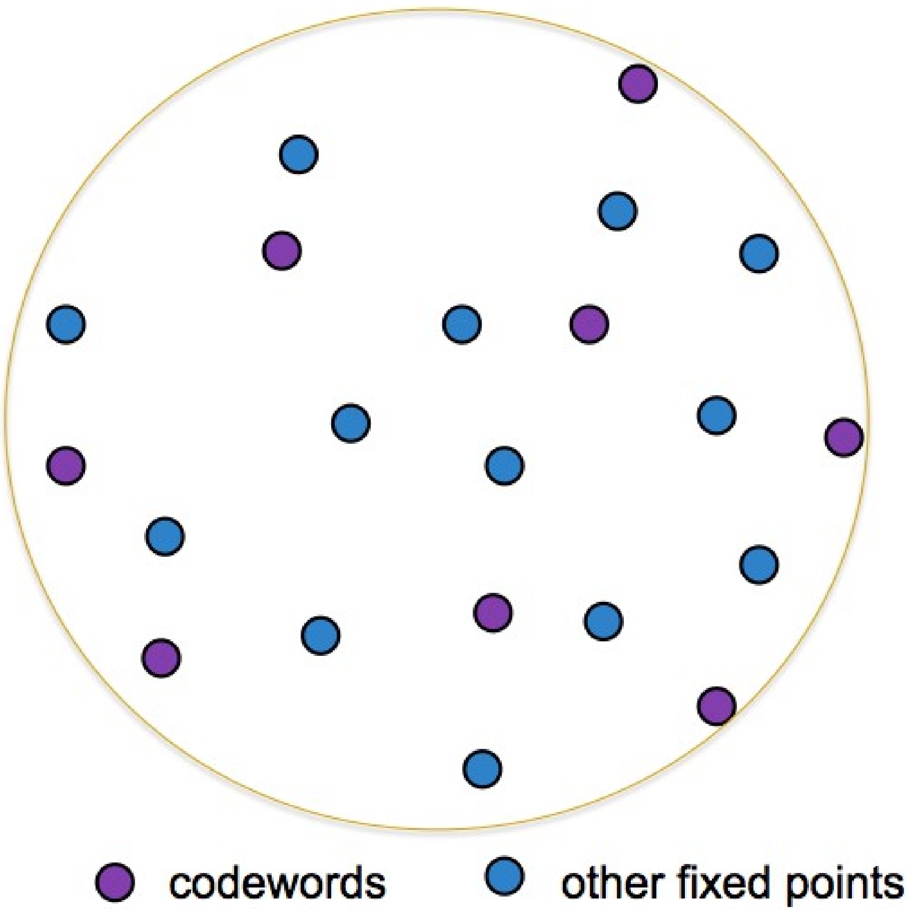
→ Large minimum distance implies high probability of successful decoding!

Back to iterative decoding...



- Codewords are fixed points, but are there other fixed points?

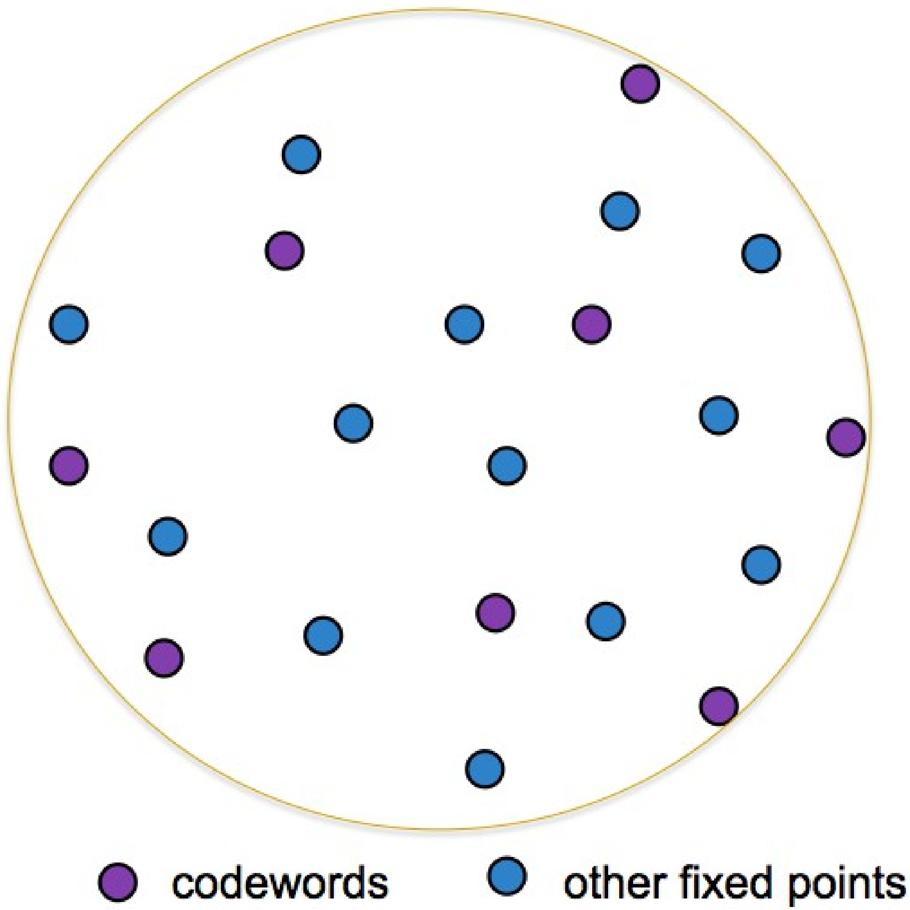
Back to iterative decoding...



- Codewords are fixed points, but are there other fixed points?

→ Yes - the suboptimal iterative decoder has other fixed points

Back to iterative decoding...



- Codewords are fixed points, but are there other fixed points?

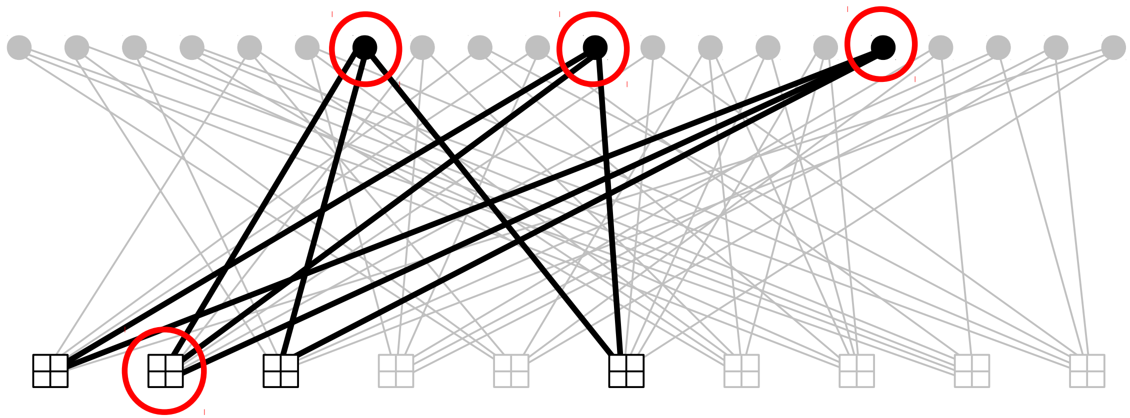
  ➡ Yes - the suboptimal iterative decoder has other fixed points

- What are these other fixed points? How many are there? How can we avoid them?

Error Floor Events - AWGNC

- On the AWGNC, failures are attributed to **trapping sets** [Richardson '03].

Definition: An (a,b) general trapping set $\tau_{a,b}$ is a set of a variable nodes which induce a subgraph with exactly b odd-degree check nodes.

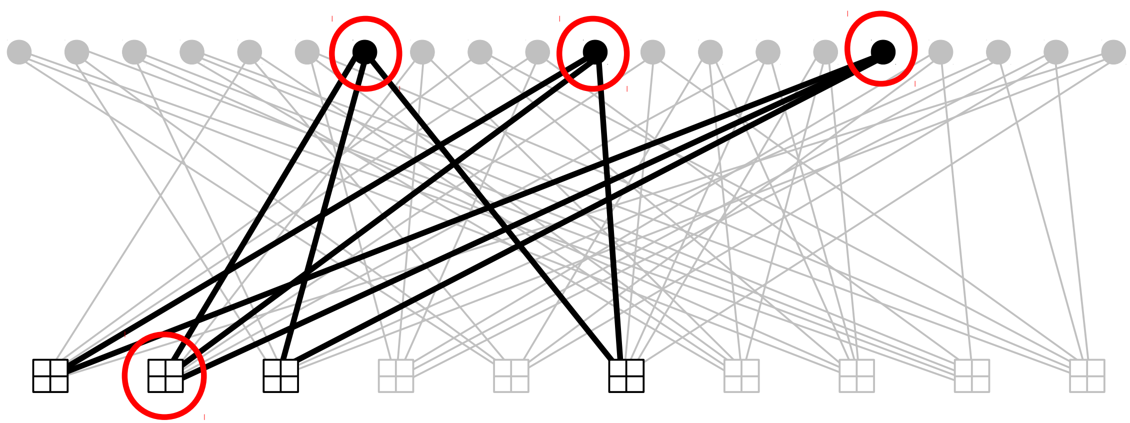


A $(3,1)$ trapping set in a $(3,6)$-regular Tanner graph

Error Floor Events - AWGNC

- On the AWGNC, failures are attributed to **trapping sets** [Richardson '03].

Definition: An (a,b) general trapping set $\tau_{a,b}$ is a set of a variable nodes which induce a subgraph with exactly b odd-degree check nodes.



A $(3,1)$ trapping set in a $(3,6)$-regular Tanner graph

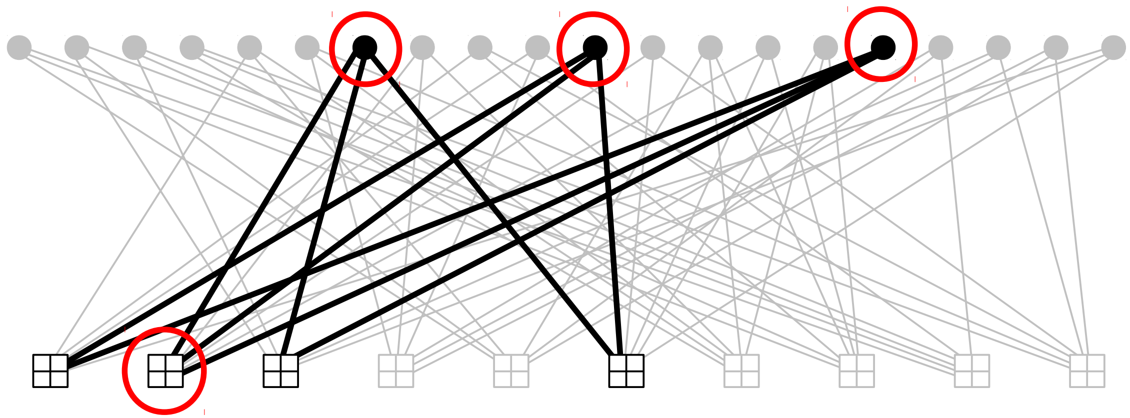
- **Low connectivity** outside the set can cause the iterative decoder to become trapped and fail to correct the symbols in the set.

Error Floor Events - AWGNC

UNIVERSITY OF NOTRE DAME

- On the AWGNC, failures are attributed to **trapping sets** [Richardson '03].

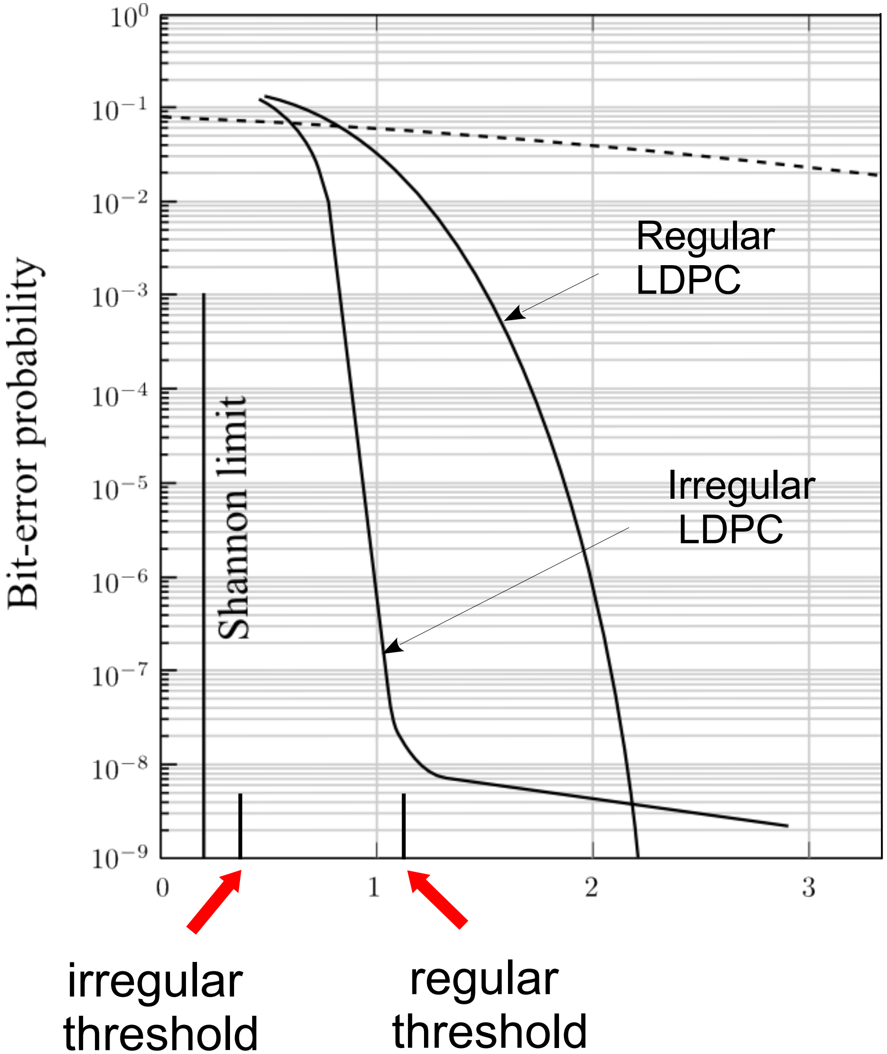
Definition: An (a,b) general trapping set $\tau_{a,b}$ is a set of a variable nodes which induce a subgraph with exactly b odd-degree check nodes.



A $(3,1)$ trapping set in a $(3,6)$-regular Tanner graph

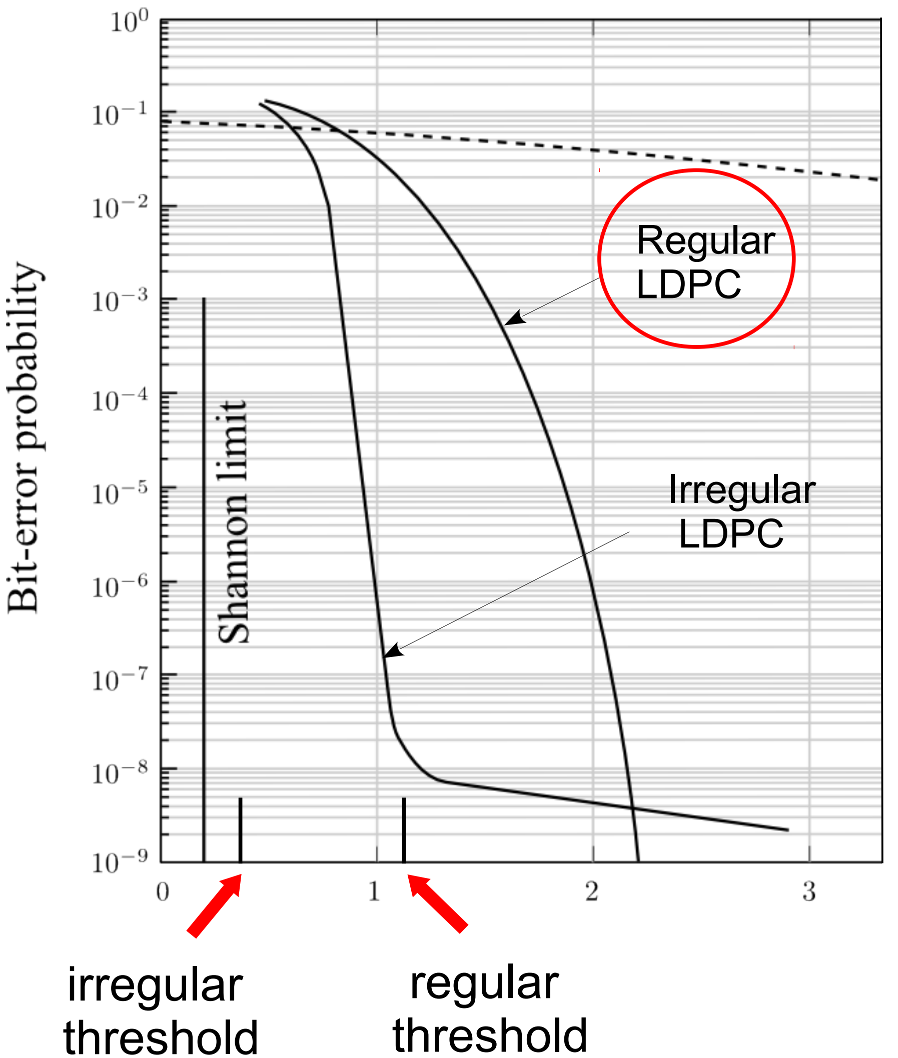
- **Low connectivity** outside the set can cause the iterative decoder to become trapped and fail to correct the symbols in the set.
- Certain types of trapping sets with small a and b, such as **elementary trapping sets** and **absorbing sets**, are known to be particularly harmful.

Regular vs. Irregular LDPC codes



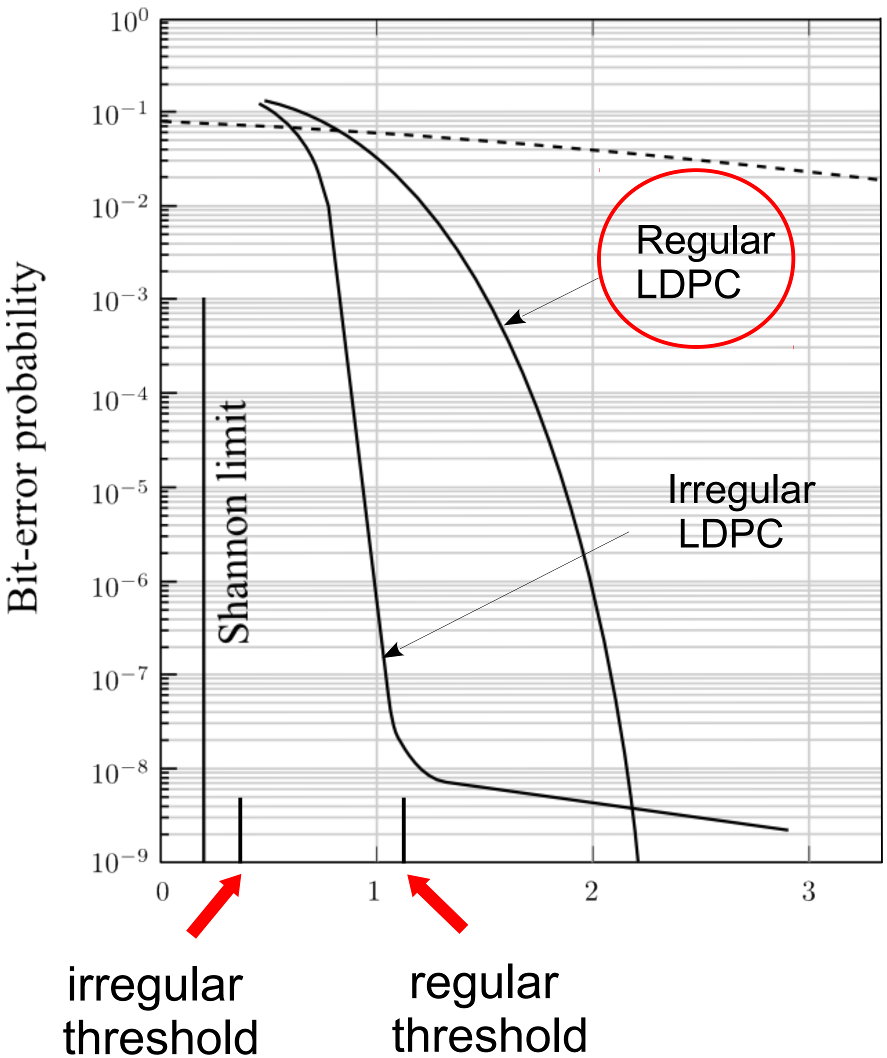
Regular vs. Irregular LDPC codes

UNIVERSITY OF NOTRE DAME



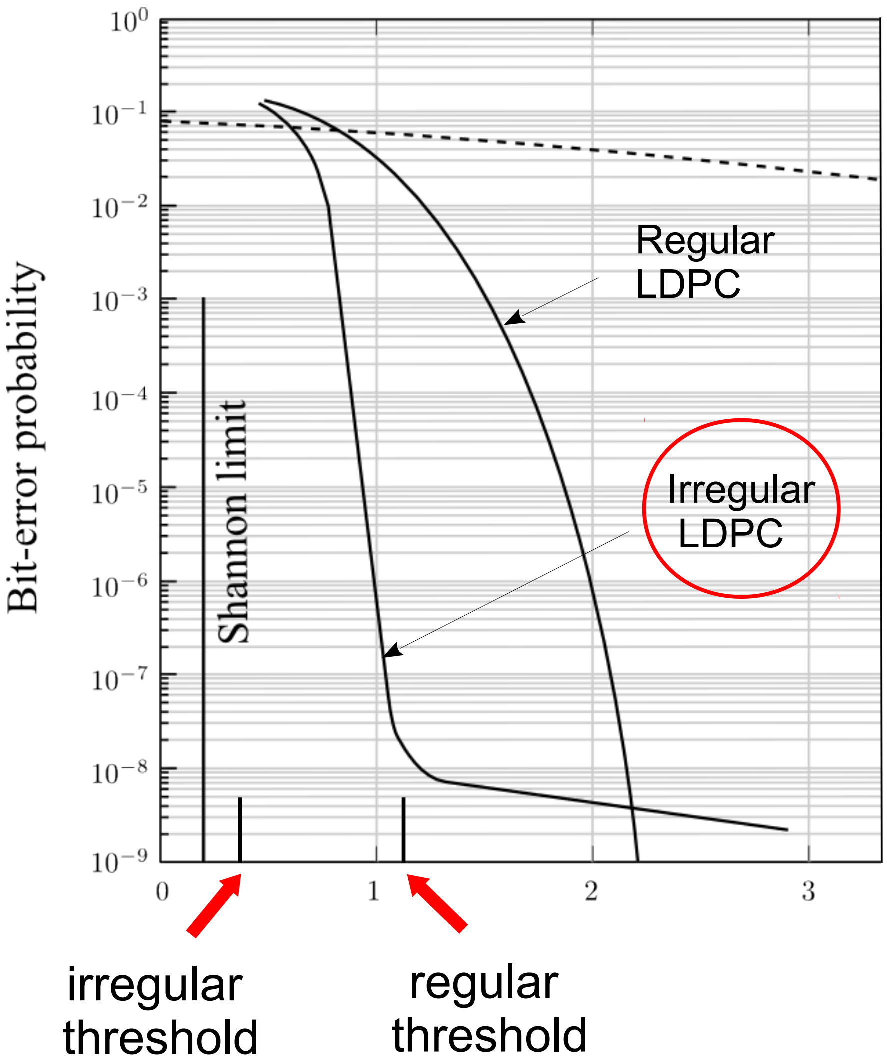
- "Regular" LDPC codes:
  - ✓ structure **aids implementation**
  - ✓ low **error floors**
  - ✗ thresholds **far from capacity**

Regular vs. Irregular LDPC codes



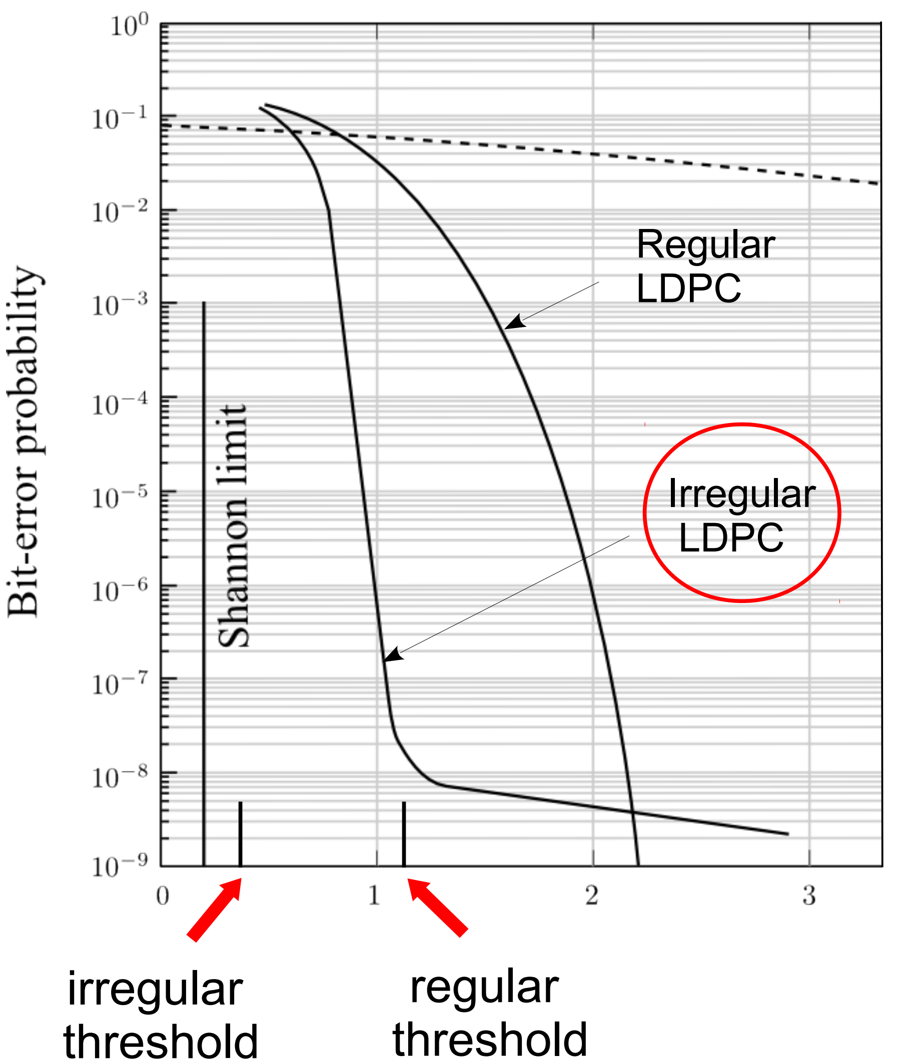
- "Regular" LDPC codes:
  - ✓ structure **aids implementation**
  - ✓ low **error floors**
  - ✗ thresholds **far from capacity**
  - ➡ not suitable for **severely power constrained** applications

Regular vs. Irregular LDPC codes



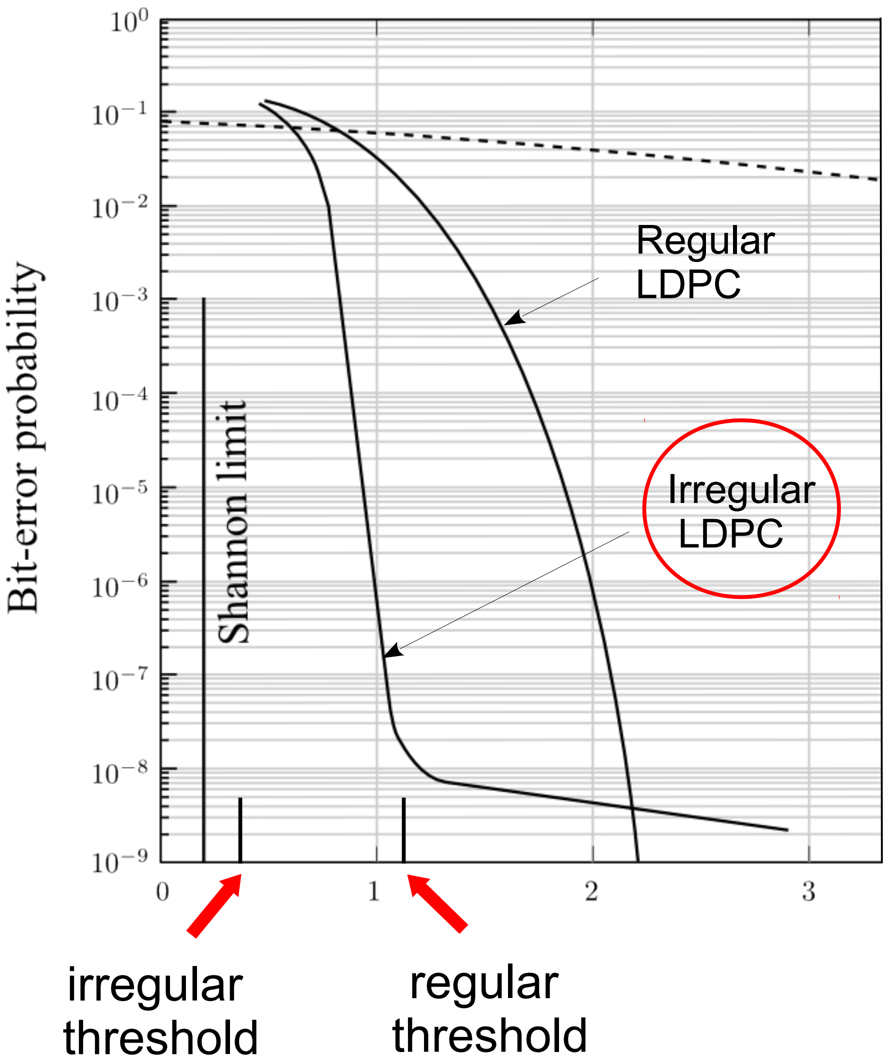
- "Regular" LDPC codes:
  - ✓ structure **aids implementation**
  - ✓ low **error floors**
  - x thresholds **far from capacity**
  - ➡ not suitable for **severely power constrained** applications
- "Irregular" LDPC codes:
  - x Less desirable structure
  - ✓ thresholds **close to capacity**
  - x visible **error floors**

Regular vs. Irregular LDPC codes



- "Regular" LDPC codes:
  - ✓ structure **aids implementation**
  - ✓ low **error floors**
  - x thresholds **far from capacity**
  - ➡ not suitable for **severely power constrained** applications
- "Irregular" LDPC codes:
  - x Less desirable structure
  - ✓ thresholds **close to capacity**
  - x visible **error floors**
  - ➡ not suitable for applications that require **very low error rates**

Regular vs. Irregular LDPC codes



- "Regular" LDPC codes:
  - ✓ structure **aids implementation**
  - ✓ low **error floors**
  - x thresholds **far from capacity**
  - ➡ not suitable for **severely power constrained** applications
- "Irregular" LDPC codes:
  - x Less desirable structure
  - ✓ thresholds **close to capacity**
  - x visible **error floors**
  - ➡ not suitable for applications that require **very low error rates**
- **Spatially coupled** LDPC codes combine all of the positive features!

Structured LDPC Codes

- Let $\mathbf{H}$ be composed of **permutation matrices** of size M:

$$\mathbf{H} = \begin{bmatrix} \mathbf{\Pi}_{1,1} & \mathbf{\Pi}_{1,2} & \cdots & \mathbf{\Pi}_{1,K} \\ \mathbf{\Pi}_{2,1} & \mathbf{\Pi}_{2,2} & \cdots & \mathbf{\Pi}_{2,K} \\ \vdots & \vdots & & \vdots \\ \mathbf{\Pi}_{J,1} & \mathbf{\Pi}_{J,2} & \cdots & \mathbf{\Pi}_{J,K} \end{bmatrix}_{J \times K}$$

(J,K)-regular codes:

- Arbitrarily large girth [Gallager, '63]
- Same distance growth rates as general regular codes [Sridharan, et al., '05]

Structured LDPC Codes

- Let $\mathbf{H}$ be composed of **permutation matrices** of size M:

$$\mathbf{H} = \begin{bmatrix} \mathbf{\Pi}_{1,1} & \mathbf{\Pi}_{1,2} & \cdots & \mathbf{\Pi}_{1,K} \\ \mathbf{\Pi}_{2,1} & \mathbf{\Pi}_{2,2} & \cdots & \mathbf{\Pi}_{2,K} \\ \vdots & \vdots & & \vdots \\ \mathbf{\Pi}_{J,1} & \mathbf{\Pi}_{J,2} & \cdots & \mathbf{\Pi}_{J,K} \end{bmatrix}_{J \times K}$$

(J,K)-regular codes:

- Arbitrarily large girth [Gallager, '63]
- Same distance growth rates as general regular codes [Sridharan, et al., '05]

Regular example:

$J = 3, K = 6, n = 24,$

$M = 4$ ($\mathbf{\Pi}_{i,j}$ is 4×4)

$$\left[\begin{array}{cccc|cccc|cccc|cccc|cccc|cccc}
0&1&0&0&0&0&0&1&0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&1\\
0&0&1&0&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&1&1&0&0&0\\
1&0&0&0&0&0&1&0&0&0&1&0&0&0&0&1&1&0&0&0&0&1&0&0\\
0&0&0&1&1&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0\\
\hline
0&0&0&1&1&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&0&1&0\\
1&0&0&0&0&0&1&0&0&1&0&0&1&0&0&0&0&1&0&0&0&0&0&1\\
0&1&0&0&0&1&0&0&0&0&0&1&0&0&1&0&1&0&0&0&0&1&0&0\\
0&0&1&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0&1&0&1&0&0&0\\
\hline
0&1&0&0&0&1&0&0&0&0&1&0&1&0&0&0&0&0&0&1&1&0&0&0\\
0&0&1&0&1&0&0&0&1&0&0&0&0&0&1&0&0&1&0&0&0&0&1&0\\
1&0&0&0&0&0&1&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0&1\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&1&0&0&0&0&1&0&0
\end{array}\right]$$

Structured LDPC Codes

UNIVERSITY OF NOTRE DAME

- Let $\mathbf{H}$ be composed of **permutation matrices** of size M:

$$\mathbf{H} = \begin{bmatrix} \mathbf{\Pi}_{1,1} & \mathbf{\Pi}_{1,2} & \cdots & \mathbf{\Pi}_{1,K} \\ \mathbf{\Pi}_{2,1} & \mathbf{\Pi}_{2,2} & \cdots & \mathbf{\Pi}_{2,K} \\ \vdots & \vdots & & \vdots \\ \mathbf{\Pi}_{J,1} & \mathbf{\Pi}_{J,2} & \cdots & \mathbf{\Pi}_{J,K} \end{bmatrix}_{J \times K}$$

(J,K)-regular codes:

- Arbitrarily large girth [Gallager, '63]
- Same distance growth rates as general regular codes [Sridharan, et al., '05]

Regular example:

$J = 3, K = 6, n = 24,$

$M = 4$ ($\mathbf{\Pi}_{i,j}$ is 4×4)

Irregular example:

Replace some permutation matrices by **0** matrix of size M

$J_{\max} = 3, K_{\max} = 6$

$$\left[\begin{array}{cccc|cccc|cccc|cccc|cccc|cccc}
0&0&1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0\\
1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0\\
0&0&0&1&0&0&1&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0\\ \hline
1&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&1&0&0&0&0&1&0\\
0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&1\\
0&0&0&1&0&0&0&0&0&0&0&1&0&1&0&0&1&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0\\ \hline
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0
\end{array}\right]$$

Protograph Representation

- **Compact representation** of a structured LDPC code ensemble based on permutation matrices of size $M \times M$ by a base matrix of size $b_c \times b_v$:

$$\mathbf{H} = \begin{bmatrix} \mathbf{\Pi}_{1,1} & \mathbf{\Pi}_{1,2} & \mathbf{\Pi}_{1,3} & \mathbf{\Pi}_{1,4} & \mathbf{\Pi}_{1,5} & \mathbf{0} \\ \mathbf{\Pi}_{2,1} & \mathbf{0} & \mathbf{\Pi}_{2,3} & \mathbf{\Pi}_{2,4} & \mathbf{\Pi}_{2,5} & \mathbf{\Pi}_{2,6} \\ \mathbf{0} & \mathbf{\Pi}_{3,2} & \mathbf{\Pi}_{3,3} & \mathbf{0} & \mathbf{0} & \mathbf{\Pi}_{3,6} \end{bmatrix}_{b_c M \times b_v M}$$

[Tho05] J. Thorpe, "Low-Density Parity-Check (LDPC) codes constructed from protographs", *Jet Propulsion Laboratory INP Progress Report*, Vol. 42-154 Aug. 2003.

Protograph Representation

UNIVERSITY OF NOTRE DAME

- **Compact representation** of a structured LDPC code ensemble based on permutation matrices of size $M \times M$ by a base matrix of size $b_c \times b_v$:

$$\mathbf{H} = \begin{bmatrix} \mathbf{\Pi}_{1,1} & \mathbf{\Pi}_{1,2} & \mathbf{\Pi}_{1,3} & \mathbf{\Pi}_{1,4} & \mathbf{\Pi}_{1,5} & \mathbf{0} \\ \mathbf{\Pi}_{2,1} & \mathbf{0} & \mathbf{\Pi}_{2,3} & \mathbf{\Pi}_{2,4} & \mathbf{\Pi}_{2,5} & \mathbf{\Pi}_{2,6} \\ \mathbf{0} & \mathbf{\Pi}_{3,2} & \mathbf{\Pi}_{3,3} & \mathbf{0} & \mathbf{0} & \mathbf{\Pi}_{3,6} \end{bmatrix}_{b_c M \times b_v M}$$

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}_{b_c \times b_v}$$

base matrix

[Tho05] J. Thorpe, “Low-Density Parity-Check (LDPC) codes constructed from protographs”, *Jet Propulsion Laboratory INP Progress Report*, Vol. 42-154 Aug. 2003.

Protograph Representation

UNIVERSITY OF NOTRE DAME

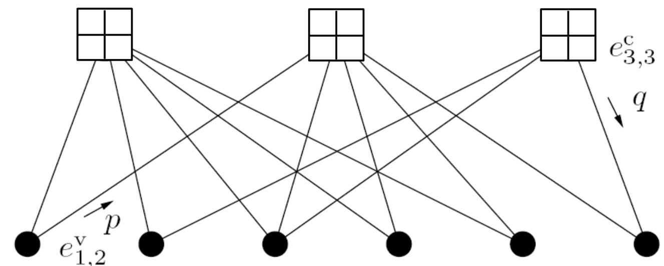
- **Compact representation** of a structured LDPC code ensemble based on permutation matrices of size $M \times M$ by a base matrix of size $b_c \times b_v$:

$$\mathbf{H} = \begin{bmatrix} \mathbf{\Pi}_{1,1} & \mathbf{\Pi}_{1,2} & \mathbf{\Pi}_{1,3} & \mathbf{\Pi}_{1,4} & \mathbf{\Pi}_{1,5} & \mathbf{0} \\ \mathbf{\Pi}_{2,1} & \mathbf{0} & \mathbf{\Pi}_{2,3} & \mathbf{\Pi}_{2,4} & \mathbf{\Pi}_{2,5} & \mathbf{\Pi}_{2,6} \\ \mathbf{0} & \mathbf{\Pi}_{3,2} & \mathbf{\Pi}_{3,3} & \mathbf{0} & \mathbf{0} & \mathbf{\Pi}_{3,6} \end{bmatrix}_{b_c M \times b_v M}$$

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}_{b_c \times b_v}$$

base matrix

b_c check nodes



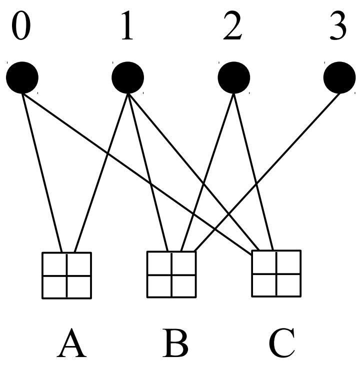
b_v variable nodes

protograph

[Tho05] J. Thorpe, "Low-Density Parity-Check (LDPC) codes constructed from protographs", *Jet Propulsion Laboratory INP Progress Report*, Vol. 42-154 Aug. 2003.

Protograph Construction

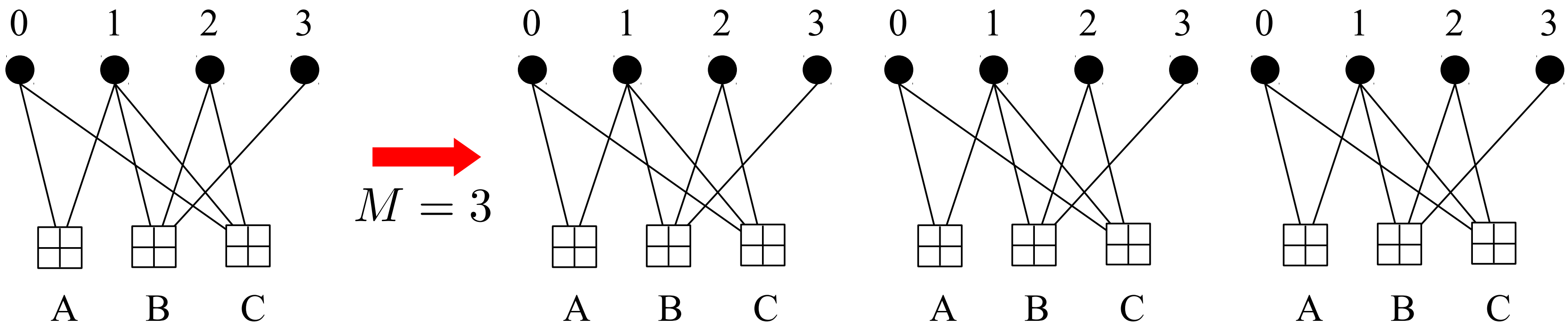
- Structured codes of varying lengths can be constructed from a **protograph** using a **copy-and-permute** operation, or ***M*-fold graph lifting**,



[Tho05] J. Thorpe, “Low-Density Parity-Check (LDPC) codes constructed from protographs”, *Jet Propulsion Laboratory INP Progress Report*, Vol. 42-154 Aug. 2003.

Protograph Construction

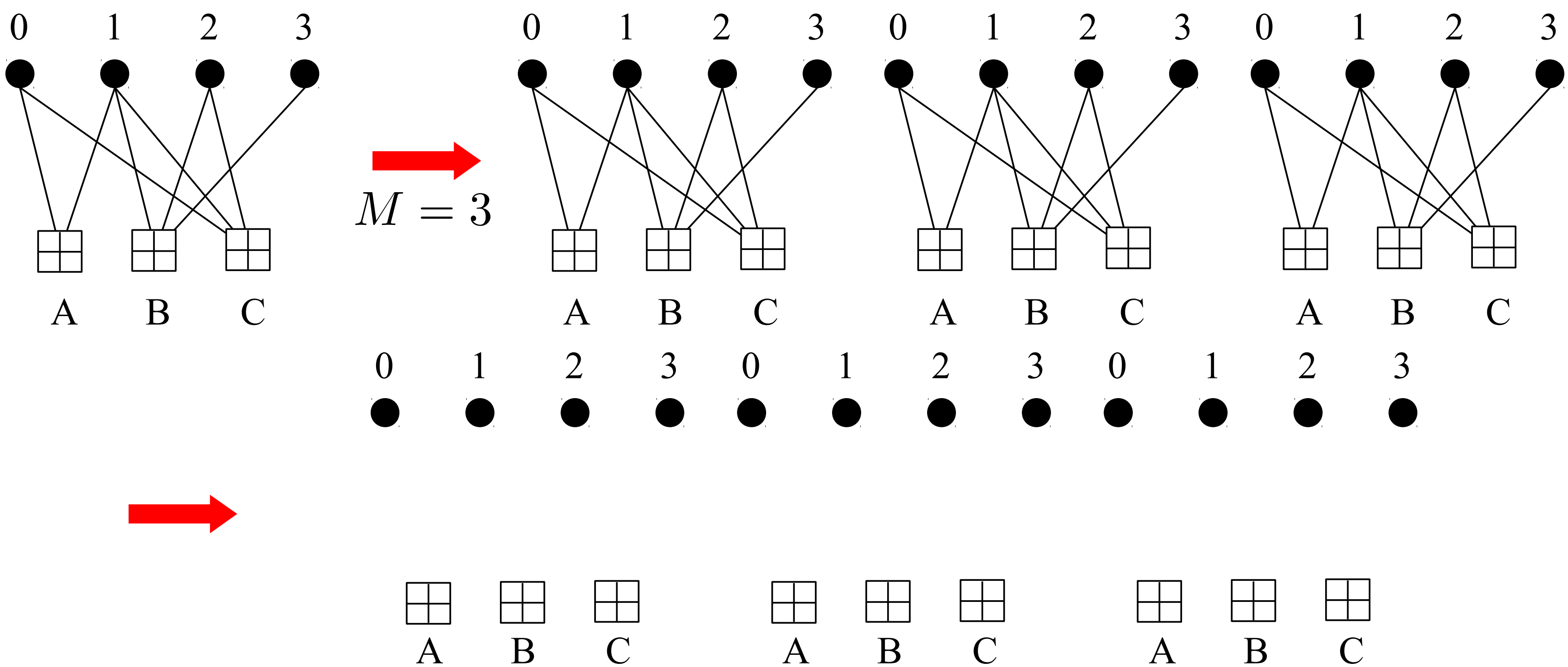
- Structured codes of varying lengths can be constructed from a **protograph** using a **copy-and-permute** operation, or ***M*-fold graph lifting**,



[Tho05] J. Thorpe, "Low-Density Parity-Check (LDPC) codes constructed from protographs", *Jet Propulsion Laboratory INP Progress Report*, Vol. 42-154 Aug. 2003.

Protograph Construction

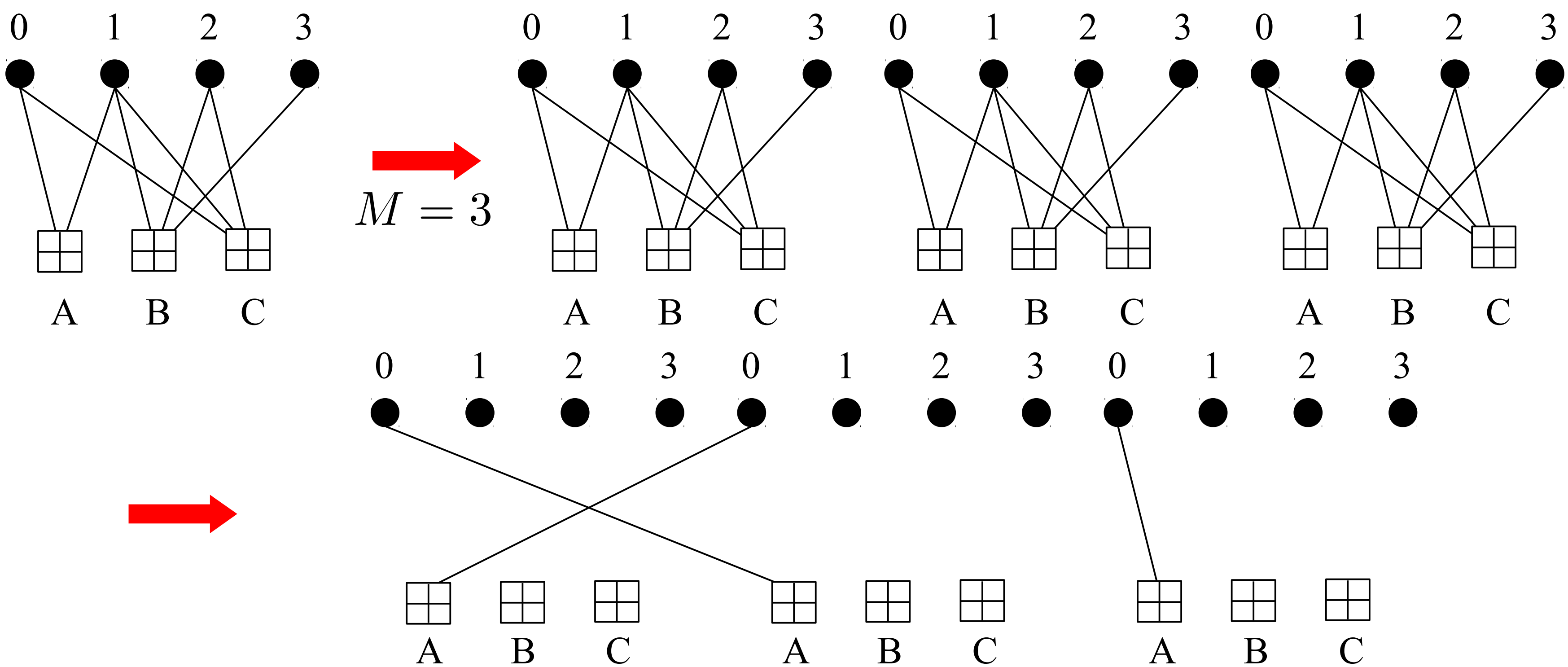
- Structured codes of varying lengths can be constructed from a **protograph** using a **copy-and-permute** operation, or ***M*-fold graph lifting**,



[Tho05] J. Thorpe, "Low-Density Parity-Check (LDPC) codes constructed from protographs", *Jet Propulsion Laboratory INP Progress Report*, Vol. 42-154 Aug. 2003.

Protograph Construction

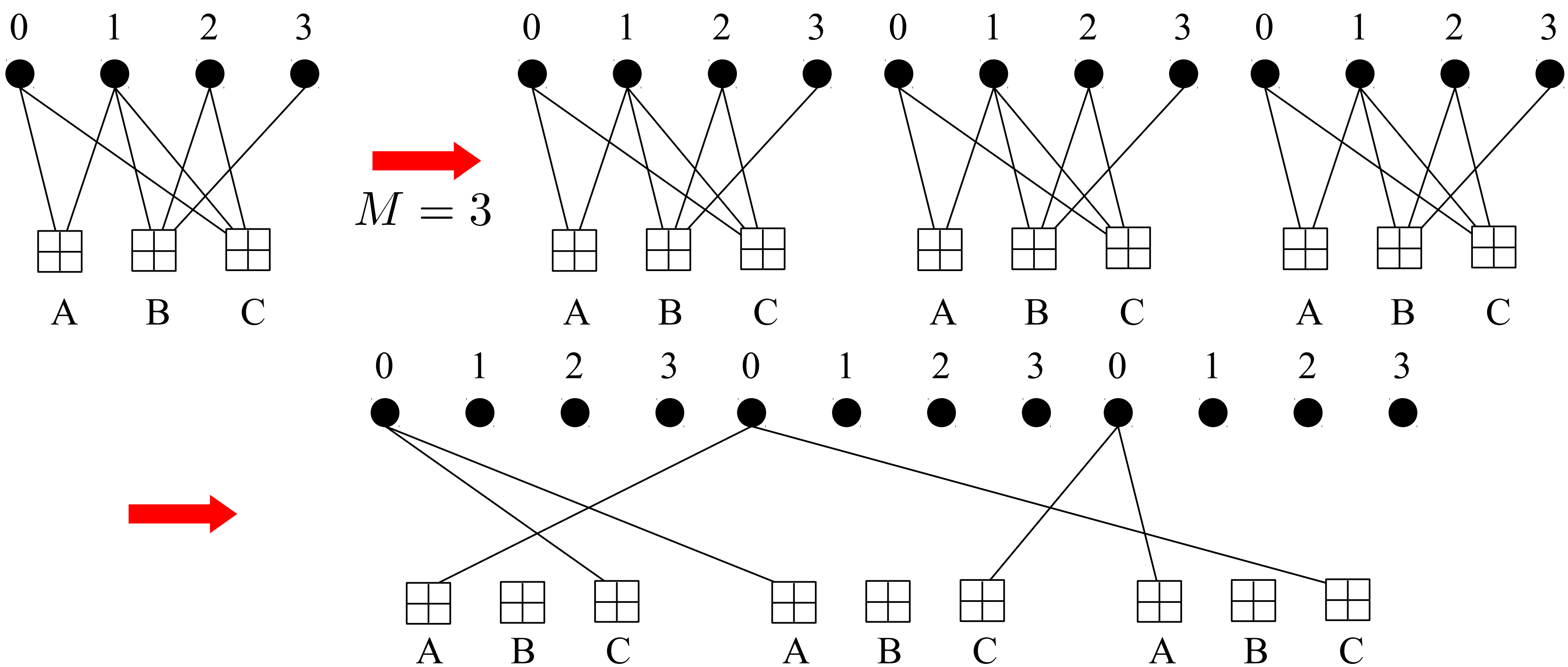
- Structured codes of varying lengths can be constructed from a **protograph** using a **copy-and-permute** operation, or ***M*-fold graph lifting**,



[Tho05] J. Thorpe, “Low-Density Parity-Check (LDPC) codes constructed from protographs”, *Jet Propulsion Laboratory INP Progress Report*, Vol. 42-154 Aug. 2003.

Protograph Construction

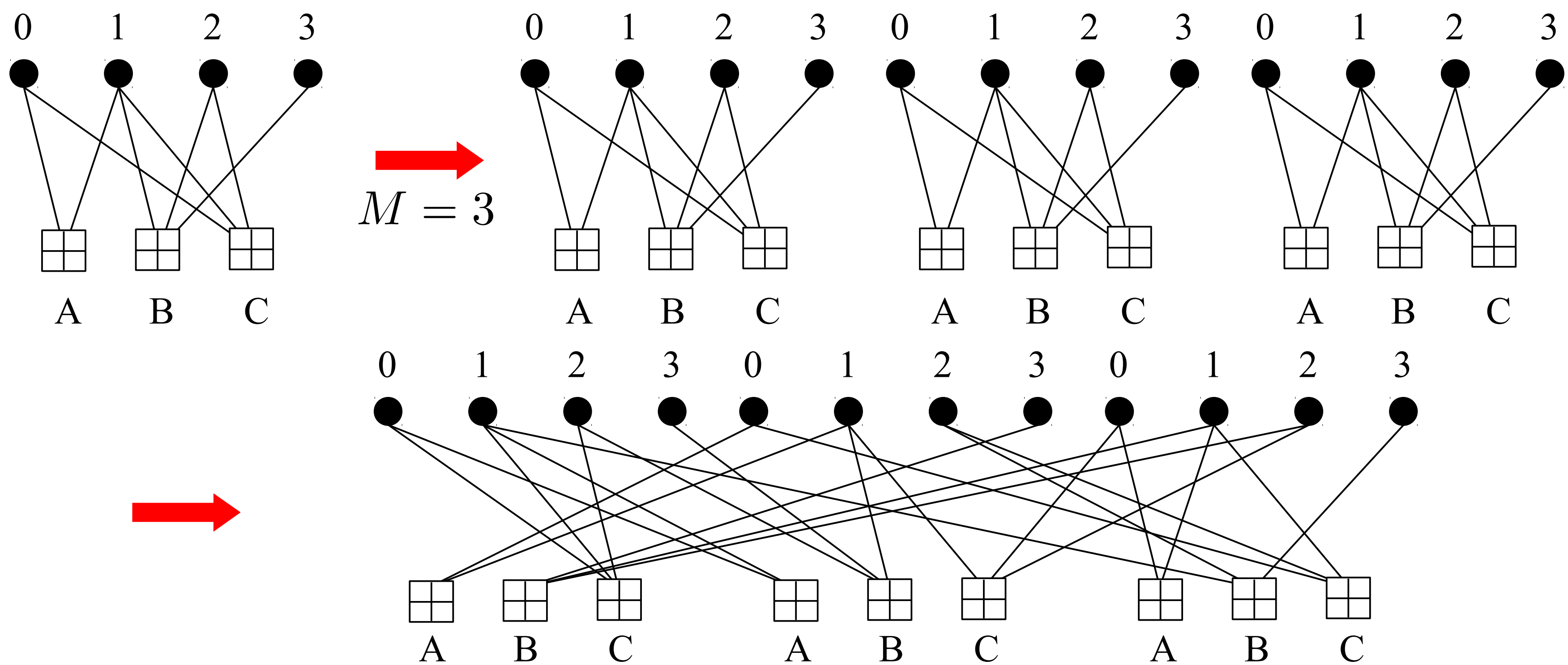
- Structured codes of varying lengths can be constructed from a **protograph** using a **copy-and-permute** operation, or **M-fold graph lifting**,



[Tho05] J. Thorpe, "Low-Density Parity-Check (LDPC) codes constructed from protographs", *Jet Propulsion Laboratory INP Progress Report*, Vol. 42-154 Aug. 2003.

Protograph Construction

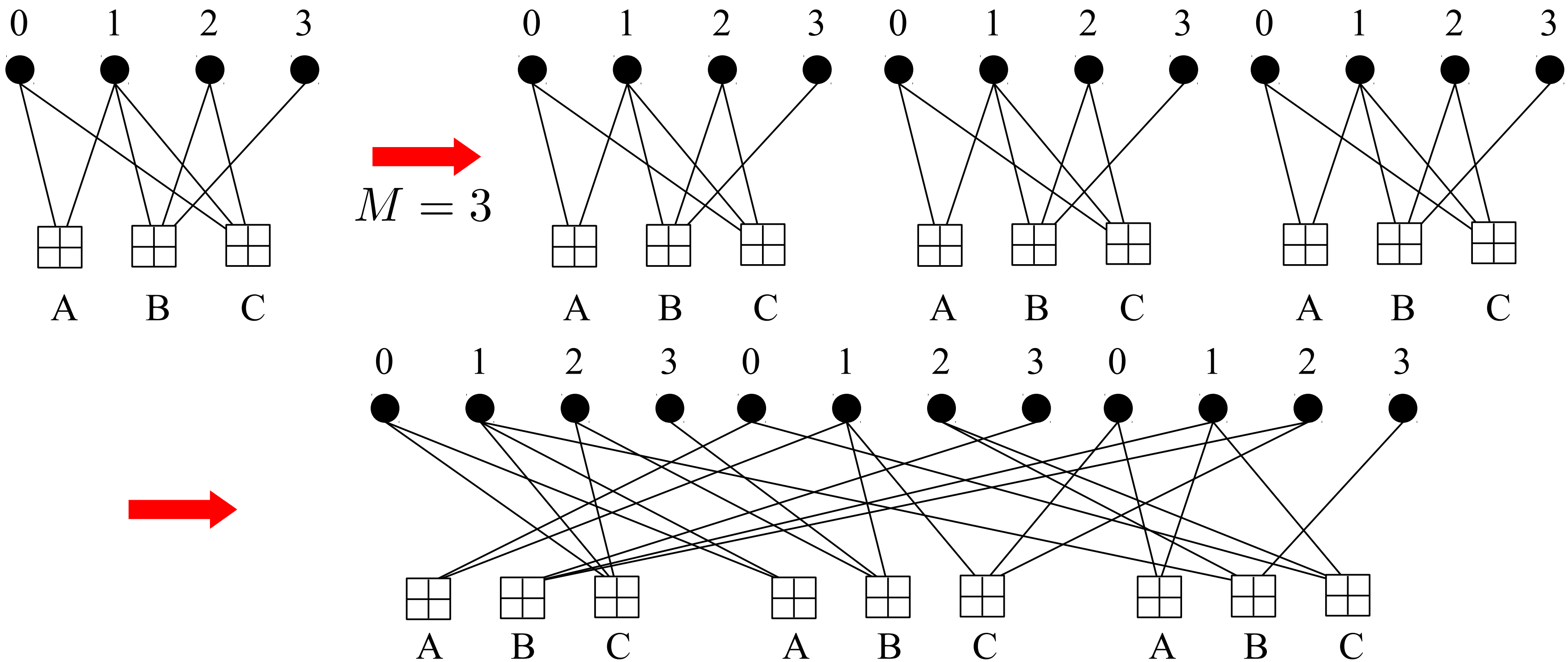
- Structured codes of varying lengths can be constructed from a **protograph** using a **copy-and-permute** operation, or ***M*-fold graph lifting**,



[Tho05] J. Thorpe, "Low-Density Parity-Check (LDPC) codes constructed from protographs", *Jet Propulsion Laboratory INP Progress Report*, Vol. 42-154 Aug. 2003.

Protograph Construction

- Structured codes of varying lengths can be constructed from a **protograph** using a **copy-and-permute** operation, or ***M*-fold graph lifting**,



- Code rate $R \geq (b_v - b_c)/b_v$, code length $n = Mb_v$

[Tho05] J. Thorpe, “Low-Density Parity-Check (LDPC) codes constructed from protographs”, *Jet Propulsion Laboratory INP Progress Report*, Vol. 42-154 Aug. 2003.

Quasi-Cyclic LDPC Codes

- **Quasi-cyclic** (QC) LDPC codes are of great interest to code designers, since they can be **encoded with low complexity** using simple feedback shift-registers and lend themselves to **efficient decoder implementation.**

Quasi-Cyclic LDPC Codes

- **Quasi-cyclic** (QC) LDPC codes are of great interest to code designers, since they can be encoded with low complexity using simple feedback shift-registers and lend themselves to efficient decoder implementation.

Example: protograph construction of a $(2,3)$-regular QC-LDPC block code

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Quasi-Cyclic LDPC Codes

- **Quasi-cyclic** (QC) LDPC codes are of great interest to code designers, since they can be **encoded with low complexity** using simple feedback shift-registers and lend themselves to **efficient decoder implementation.**

Example: protograph construction of a $(2{,}3)$-regular QC-LDPC block code

> For QC codes, the permutation matrices are **circulant**

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \rightarrow \mathbf{H} = \left[\begin{array}{ccccccc|ccccccc|ccccccc}
0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0\\
0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0\\
\hline
0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0\\
0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1\\
0&0&0&0&1&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&0&0&0&0\\
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0
\end{array}\right]$$

$$M = 7$$
$$n = Mb_v = 21$$

Quasi-Cyclic LDPC Codes

UNIVERSITY OF NOTRE DAME

- **Quasi-cyclic** (QC) LDPC codes are of great interest to code designers, since they can be **encoded with low complexity** using simple feedback shift-registers and lend themselves to **efficient decoder implementation.**

Example: protograph construction of a $(2,3)$-regular QC-LDPC block code

For QC codes, the permutation matrices are **circulant**

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \rightarrow \mathbf{H} = \left[\begin{array}{ccccccc|ccccccc|ccccccc}
0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0\\
0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0\\
\hline
0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0\\
0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1\\
0&0&0&0&1&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&0&0&0&0\\
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0
\end{array}\right]$$

$$M = 7$$

$$n = Mb_v = 21$$

- The QC sub-ensemble contains **atypical** codes. Thus ensemble average results **do not apply** to these highly structured codes.

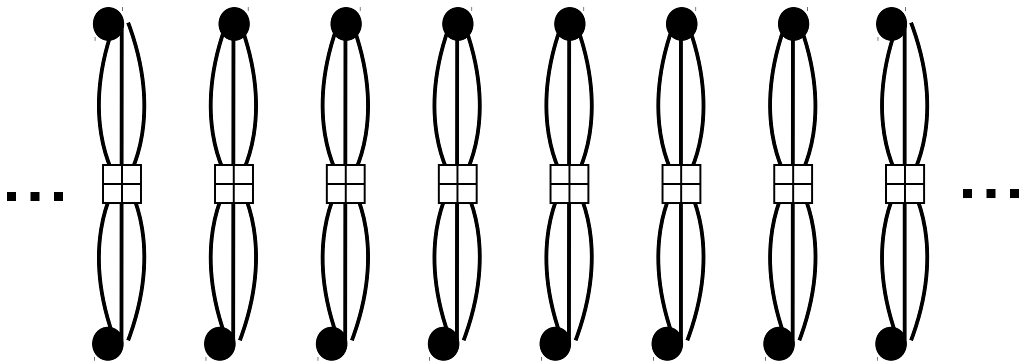
Outline

UNIVERSITY OF NOTRE DAME

- Introduction: From Shannon to Modern Coding Theory
  - ➡ Channel capacity, structured codes, random codes, LDPC codes
- LDPC Block Codes
  - ➡ Parity-check matrix and Tanner graph representations, minimum distance bounds, iterative decoding thresholds, protograph-based constructions
- **Spatially Coupled LDPC Codes**
  - ➡ Protograph representation, edge-spreading construction, termination
  - ➡ Iterative decoding thresholds, threshold saturation, minimum distance
- Practical Considerations
  - ➡ Window decoding, performance, latency, and complexity comparisons to LDPC block codes, rate-compatibility, implementation aspects

Spatially Coupled Protographs

- Consider transmission of consecutive blocks (protograph representation):



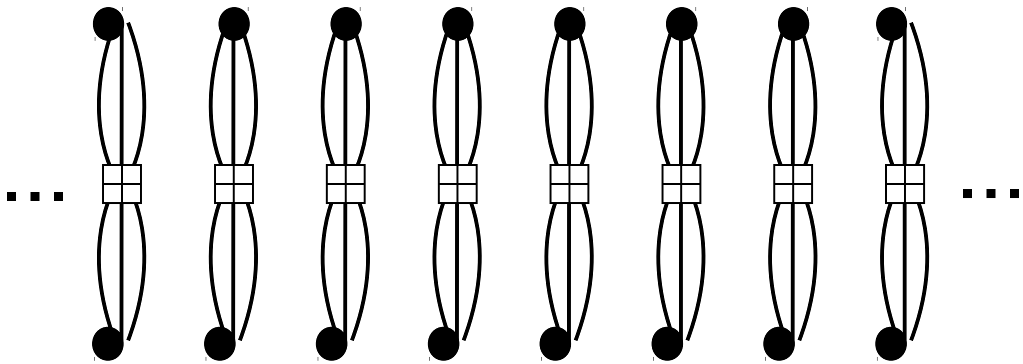
$$\mathbf{B} = \begin{bmatrix} 3 & 3 \end{bmatrix}_{b_c \times b_v}$$

$(3,6)$-regular
LDPC-BC
base matrix

Spatially Coupled Protographs

UNIVERSITY OF NOTRE DAME

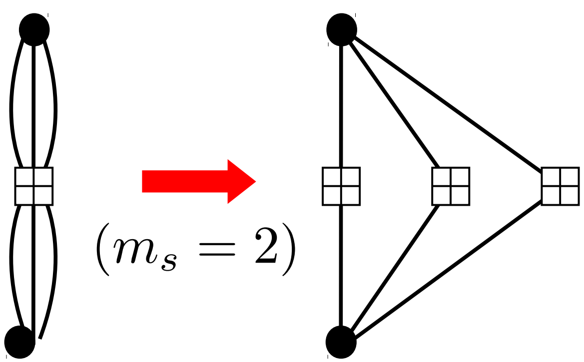
- Consider transmission of consecutive blocks (protograph representation):



$$\mathbf{B} = \begin{bmatrix} 3 & 3 \end{bmatrix}_{b_c \times b_v}$$

(3,6)-regular LDPC-BC base matrix

- Blocks are **spatially coupled** (introducing **memory**) by **spreading edges** over time:

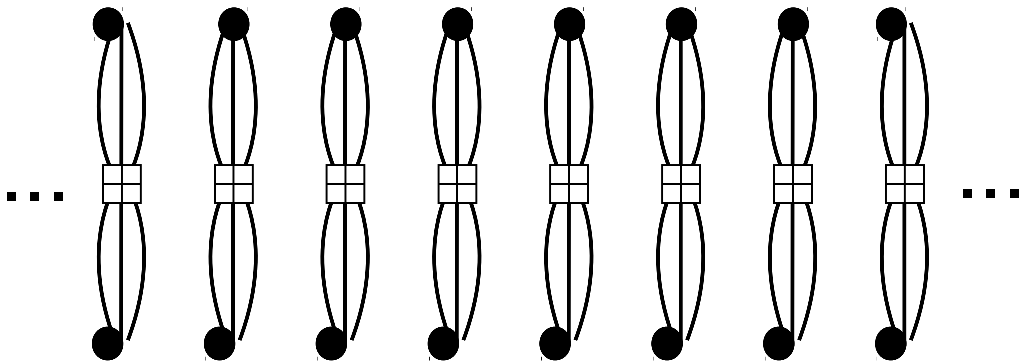


$$\mathbf{B} = \begin{bmatrix} 3 & 3 \end{bmatrix} \xrightarrow{(m_s = 2)} \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \\ \mathbf{B}_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Spatially Coupled Protographs

UNIVERSITY OF NOTRE DAME

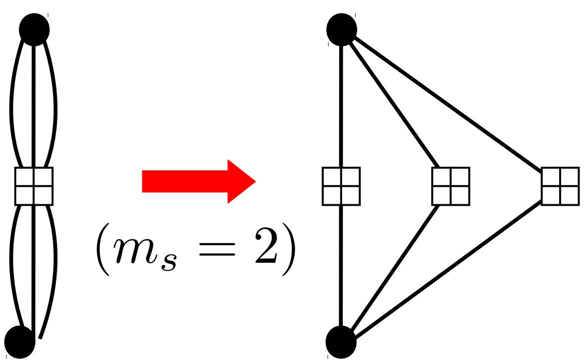
- Consider transmission of consecutive blocks (protograph representation):



$$\mathbf{B} = \begin{bmatrix} 3 & 3 \end{bmatrix}_{b_c \times b_v}$$

(3,6)-regular LDPC-BC base matrix

- Blocks are **spatially coupled** (introducing **memory**) by **spreading edges** over time:



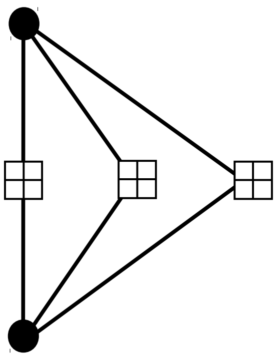
$(m_s = 2)$

$$\mathbf{B} = \begin{bmatrix} 3 & 3 \end{bmatrix} \xrightarrow{(m_s = 2)} \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \\ \mathbf{B}_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

- **Spreading constraint:** $\sum_{i=0}^{m_s} \mathbf{B}_i = \mathbf{B}$ ($\mathbf{B}_i$ has size $b_c \times b_v$)

Spatially Coupled Protographs

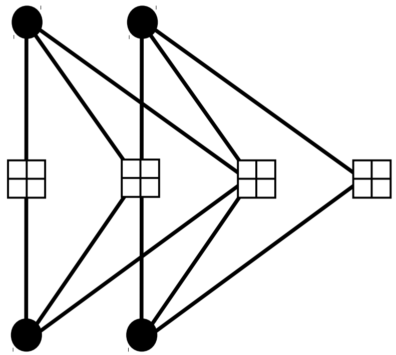
- Transmission of consecutive spatially coupled (SC) blocks results in a **convolutional protograph**:



Spatially Coupled Protographs

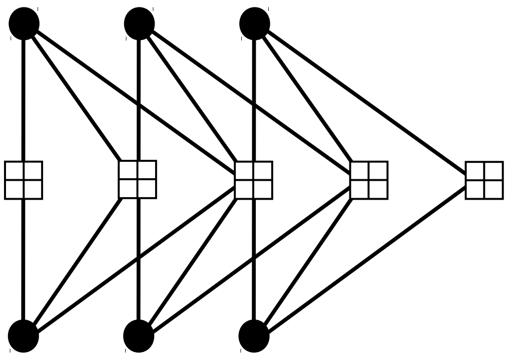
UNIVERSITY OF NOTRE DAME

- Transmission of consecutive spatially coupled (SC) blocks results in a **convolutional protograph**:



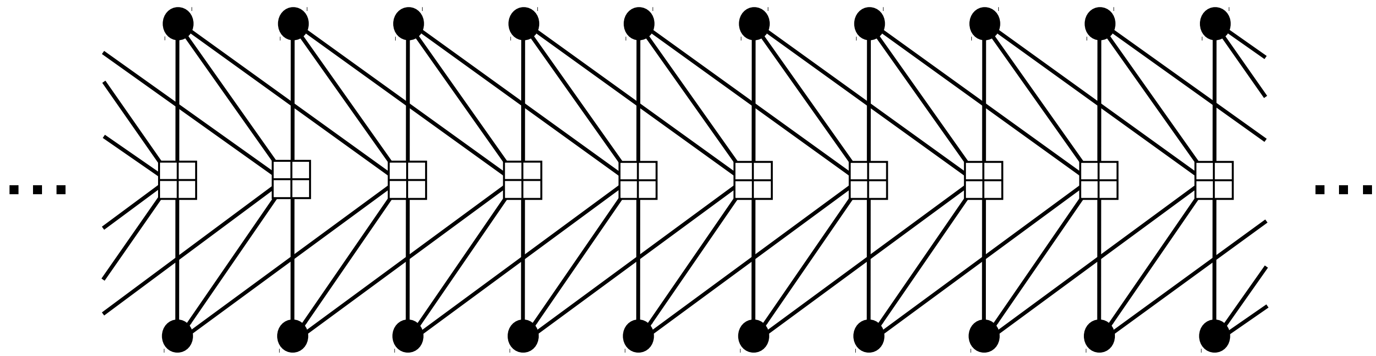
Spatially Coupled Protographs

- Transmission of consecutive spatially coupled (SC) blocks results in a **convolutional protograph**:



Spatially Coupled Protographs

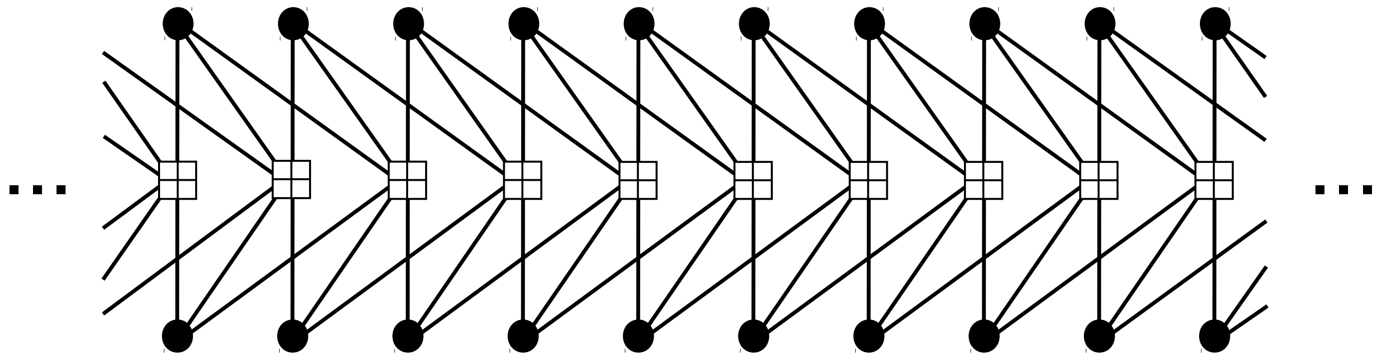
- Transmission of consecutive spatially coupled (SC) blocks results in a **convolutional protograph**:



Spatially Coupled Protographs

UNIVERSITY OF NOTRE DAME

- Transmission of consecutive spatially coupled (SC) blocks results in a **convolutional protograph**:



- The bi-infinite convolutional protograph corresponds to a bi-infinite **convolutional base matrix**:

$$\mathbf{B}_{[-\infty,\infty]} = \begin{bmatrix} \ddots & & \ddots & \ddots & & & \\ & \mathbf{B}_{m_s} & \cdots & \mathbf{B}_1 & \mathbf{B}_0 & & \\ & & \ddots & & \ddots & \ddots & \\ & & & \mathbf{B}_{m_s} & \cdots & \mathbf{B}_1 & \mathbf{B}_0 \\ & & & & \ddots & & \ddots & \ddots \end{bmatrix}$$

Rate:

$$R = \frac{b_v - b_c}{b_v}$$

Constraint length:

$$\nu_s = b_v(m_s + 1)$$

SC-LDPC Code Ensembles

- An ensemble of (3,6)-regular SC-LDPC codes can be created from the **convolutional protograph** by the graph lifting operation

$$\mathbf{B}_{[-\infty,\infty]} = \begin{bmatrix} \ddots & \ddots & \ddots & & & \\ \mathbf{B}_2 & \mathbf{B}_1 & \mathbf{B}_0 & & & \\ & \mathbf{B}_2 & \mathbf{B}_1 & \mathbf{B}_0 & & \\ & & \mathbf{B}_2 & \mathbf{B}_1 & \mathbf{B}_0 & \\ & & & \ddots & \ddots & \ddots \end{bmatrix}$$

SC-LDPC Code Ensembles

- An ensemble of (3,6)-regular SC-LDPC codes can be created from the **convolutional protograph** by the graph lifting operation

$$\mathbf{B}_{[-\infty,\infty]} = \begin{bmatrix} \ddots & & \ddots & & \ddots & & & & & & & \\ 1 & 1 & 1 & 1 & 1 & 1 & & & & & \\ & & 1 & 1 & 1 & 1 & 1 & 1 & & & \\ & & & & 1 & 1 & 1 & 1 & 1 & 1 \\ & & & & & & \ddots & & \ddots & & \ddots \end{bmatrix}$$

$$\mathbf{B}_i = [\,1 \quad 1\,]$$
$$b_c = 1$$
$$b_v = 2$$
$$m_s = 2$$

SC-LDPC Code Ensembles

- An ensemble of (3,6)-regular SC-LDPC codes can be created from the **convolutional protograph** by the graph lifting operation

$$\mathbf{B}_{[-\infty,\infty]} = \begin{bmatrix} \ddots & & \ddots & & \ddots & & & & & \\ 1 & 1 & 1 & 1 & 1 & 1 & & & & \\ & & 1 & 1 & 1 & 1 & 1 & 1 & & \\ & & & & 1 & 1 & 1 & 1 & 1 & 1 \\ & & & & & \ddots & & \ddots & & \ddots \end{bmatrix}$$

$$\mathbf{B}_i = [1 \quad 1] \qquad b_c = 1 \qquad b_v = 2 \qquad m_s = 2$$

Graph lifting: $\Pi_{i,j}$ is an $M \times M$ permutation matrix

$$\nu_s = M b_v(m_s + 1) = 6M$$

$$\mathbf{H}_{\text{cc}} = \begin{bmatrix} \ddots & & \ddots & & \ddots & & & & & \\ \Pi_{5,t} & \Pi_{4,t} & \Pi_{3,t} & \Pi_{2,t} & \Pi_{1,t} & \Pi_{0,t} & & & & \\ & & \Pi_{5,t+1} & \Pi_{4,t+1} & \Pi_{3,t+1} & \Pi_{2,t+1} & \Pi_{1,t+1} & \Pi_{0,t+1} & & \\ & & & & \Pi_{5,t+2} & \Pi_{4,t+2} & \Pi_{3,t+2} & \Pi_{2,t+2} & \Pi_{1,t+2} & \Pi_{0,t+2} \\ & & & & & \ddots & & \ddots & & \ddots \end{bmatrix}$$

SC-LDPC Code Ensembles

UNIVERSITY OF NOTRE DAME

- An ensemble of (3,6)-regular SC-LDPC codes can be created from the **convolutional protograph** by the graph lifting operation

$$\mathbf{B}_{[-\infty,\infty]} = \begin{bmatrix} \ddots & \ddots & \ddots & & & \\ 1\ \ 1 & 1\ \ 1 & 1\ \ 1 & & & \\ & 1\ \ 1 & 1\ \ 1 & 1\ \ 1 & & \\ & & 1\ \ 1 & 1\ \ 1 & 1\ \ 1 & \\ & & & \ddots & \ddots & \ddots \end{bmatrix}$$

$$\mathbf{B}_i = [1 \quad 1], \quad b_c = 1, \quad b_v = 2, \quad m_s = 2$$

Graph lifting: $\Pi_{i,j}$ is an $M \times M$ permutation matrix

$$\nu_s = M b_v (m_s + 1) = 6M$$

$$\mathbf{H}_{\mathrm{cc}} = \begin{bmatrix} \ddots & \ddots & \ddots & & & \\ \Pi_{5,t}\ \ \Pi_{4,t} & \Pi_{3,t}\ \ \Pi_{2,t} & \Pi_{1,t}\ \ \Pi_{0,t} & & & \\ & \Pi_{5,t+1}\ \ \Pi_{4,t+1} & \Pi_{3,t+1}\ \ \Pi_{2,t+1} & \Pi_{1,t+1}\ \ \Pi_{0,t+1} & & \\ & & \Pi_{5,t+2}\ \ \Pi_{4,t+2} & \Pi_{3,t+2}\ \ \Pi_{2,t+2} & \Pi_{1,t+2}\ \ \Pi_{0,t+2} & \\ & & & \ddots & \ddots & \ddots \end{bmatrix}$$

- If each permutation matrix $\Pi_{i,j}$ is **circulant**, the codes are **quasi-cyclic**

Terminated Spatially Coupled Codes

- Consider **terminating** $\mathbf{B}_{[-\infty,\infty]}$ to a (block code) **base matrix** of length Lb_v:

$$\mathbf{B}_{[0,L-1]} = \begin{bmatrix} \mathbf{B}_0 & & \\ \vdots & \ddots & \\ \mathbf{B}_{m_s} & & \mathbf{B}_0 \\ & \ddots & \vdots \\ & & \mathbf{B}_{m_s} \end{bmatrix}_{(L+m_s)b_c \times Lb_v}$$

Code rate:

$$R_L = \frac{Lb_v - (L+m_s)b_c}{Lb_v}.$$

Terminated Spatially Coupled Codes

- Consider **terminating** $\mathbf{B}_{[-\infty,\infty]}$ to a (block code) **base matrix** of length Lb_v:

$$\mathbf{B}_{[0,L-1]} = \begin{bmatrix} \mathbf{B}_0 & & \\ \vdots & \ddots & \\ \mathbf{B}_{m_s} & & \mathbf{B}_0 \\ & \ddots & \vdots \\ & & \mathbf{B}_{m_s} \end{bmatrix}_{(L+m_s)b_c \times Lb_v}$$

Code rate:

$$R_L = \frac{Lb_v - (L+m_s)b_c}{Lb_v}.$$

- For large L, R_L approaches the **unterminated** code rate $R = (b_v - b_c)/b_v$.

Terminated Spatially Coupled Codes

- Consider **terminating** $\mathbf{B}_{[-\infty,\infty]}$ to a (block code) **base matrix** of length Lb_v:

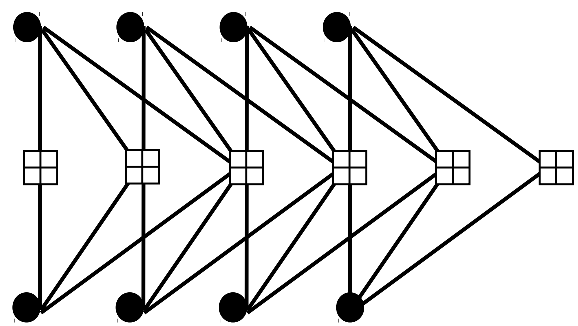
$$\mathbf{B}_{[0,L-1]} = \begin{bmatrix} \mathbf{B}_0 & & \\ \vdots & \ddots & \\ \mathbf{B}_{m_s} & & \mathbf{B}_0 \\ & \ddots & \vdots \\ & & \mathbf{B}_{m_s} \end{bmatrix}_{(L+m_s)b_c \times Lb_v}$$

Code rate:

$$R_L = \frac{Lb_v - (L+m_s)b_c}{Lb_v}.$$

- For large L, R_L approaches the **unterminated** code rate $R = (b_v - b_c)/b_v$.
- **Example:** $(3,6)$-regular base matrix $\mathbf{B} = [3 \;\; 3]$, $m_s = 2$, $L = 4$, $R_4 = 1/4$

$$\mathbf{B}_{[0,3]} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$



(**check node degrees lower at the ends**)

Terminated Spatially Coupled Codes

- Consider **terminating** $\mathbf{B}_{[-\infty,\infty]}$ to a (block code) **base matrix** of length Lb_v:

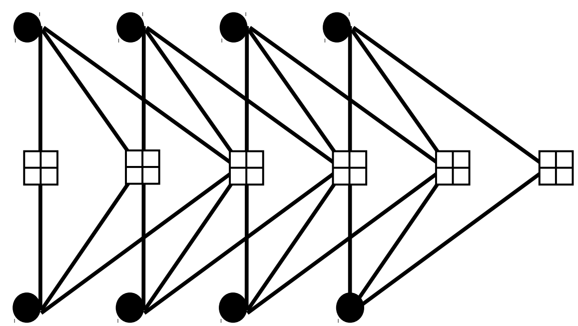
$$\mathbf{B}_{[0,L-1]} = \begin{bmatrix} \mathbf{B}_0 & & \\ \vdots & \ddots & \\ \mathbf{B}_{m_s} & & \mathbf{B}_0 \\ & \ddots & \vdots \\ & & \mathbf{B}_{m_s} \end{bmatrix}_{(L+m_s)b_c \times Lb_v}$$

Code rate:

$$R_L = \frac{Lb_v - (L+m_s)b_c}{Lb_v}.$$

- For large L, R_L approaches the **unterminated** code rate $R = (b_v - b_c)/b_v$.
- **Example:** $(3{,}6)$-regular base matrix $\mathbf{B} = [3 \;\; 3]$, $m_s = 2$, $L = 4$, $R_4 = 1/4$

$$\mathbf{B}_{[0,3]} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

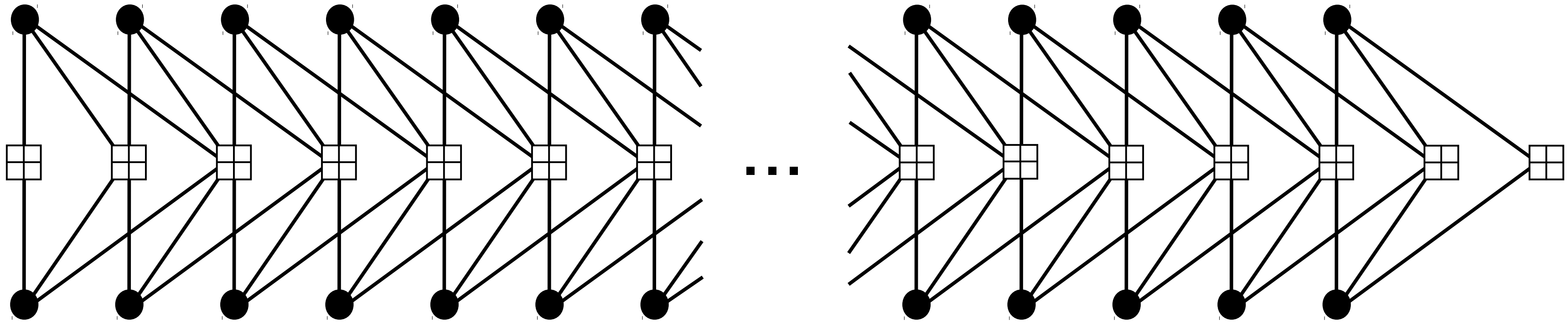


(**check node degrees lower at the ends**)

- Codes can be **lifted** to **different lengths and rates** by varying M and L.

Thresholds of SC-LDPC Codes

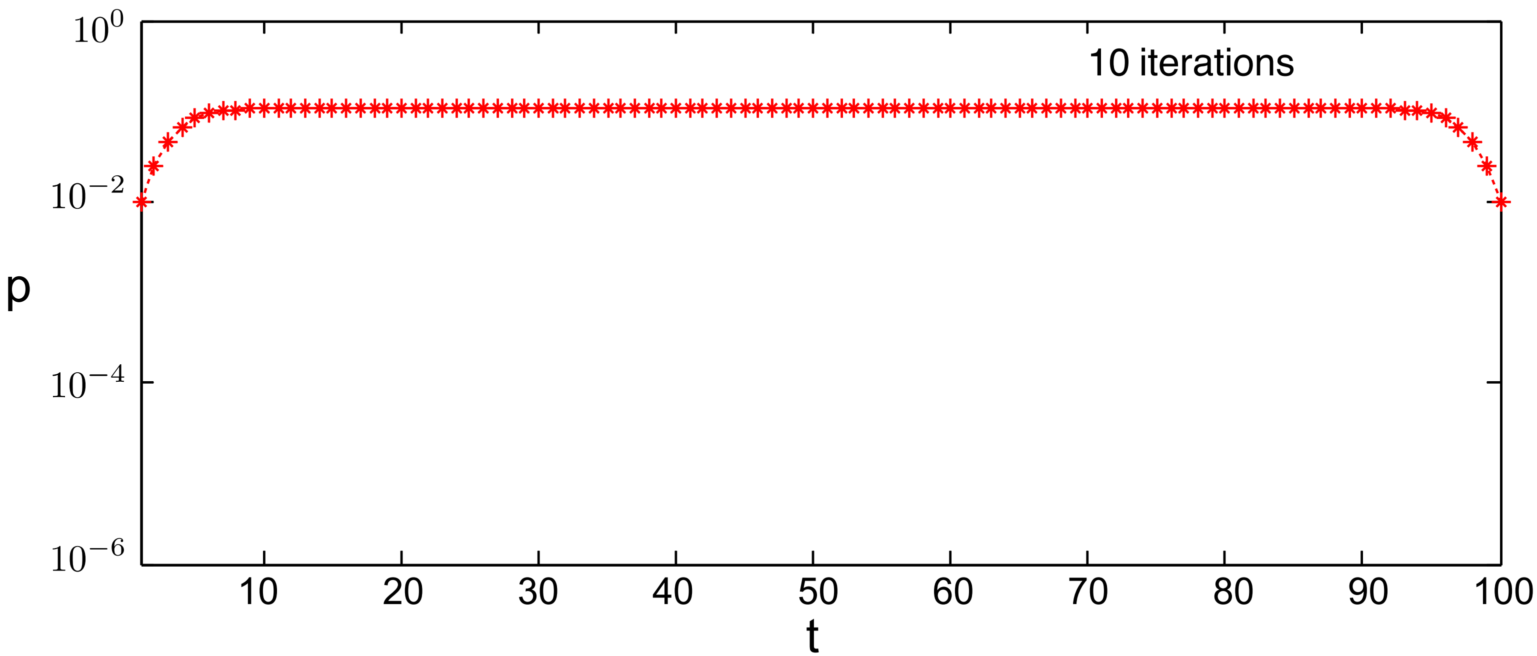
- Variable nodes all have the **same degree** as the block code.
- Check nodes with **lower degrees** (at the ends) improve the BP decoder.



Thresholds of SC-LDPC Codes

- Variable nodes all have the **same degree** as the block code.
- Check nodes with **lower degrees** (at the ends) improve the BP decoder.

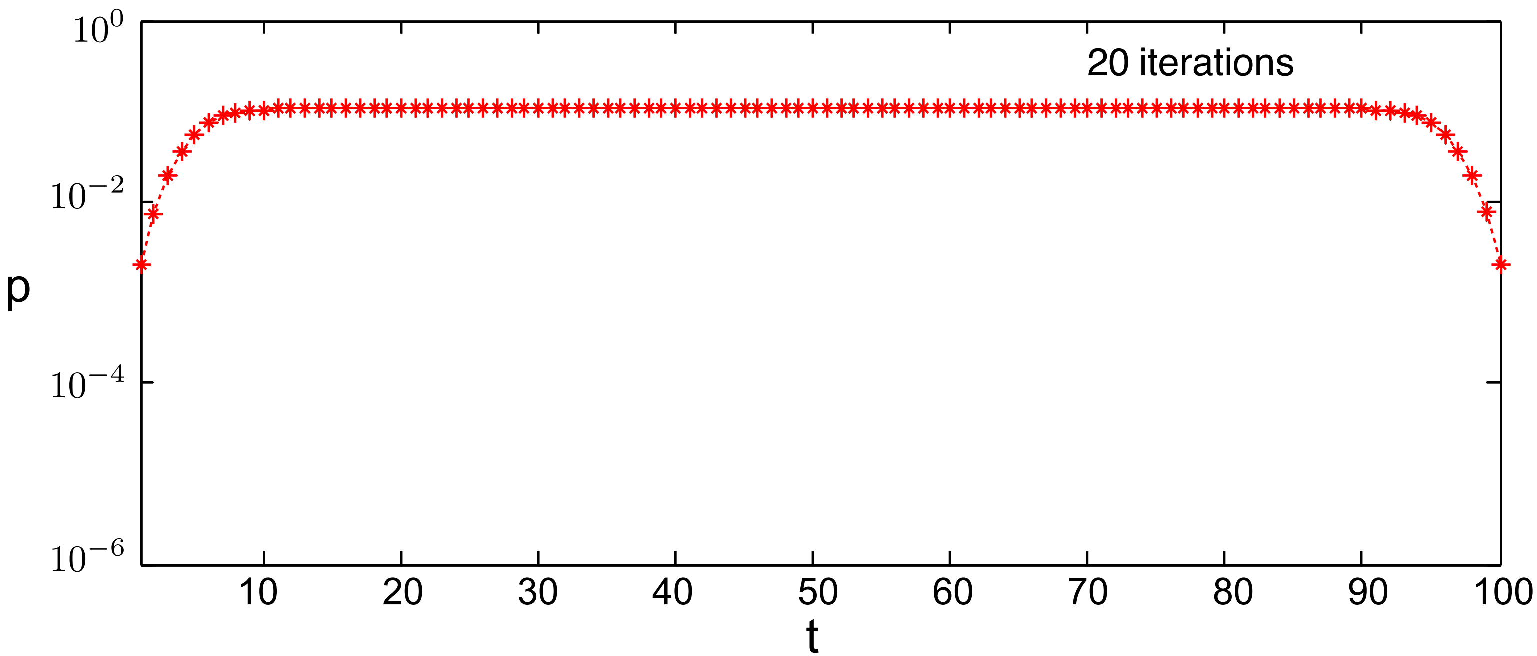
Evolution of message probabilities $(L = 100)$:



Thresholds of SC-LDPC Codes

- Variable nodes all have the **same degree** as the block code.
- Check nodes with **lower degrees** (at the ends) improve the BP decoder.

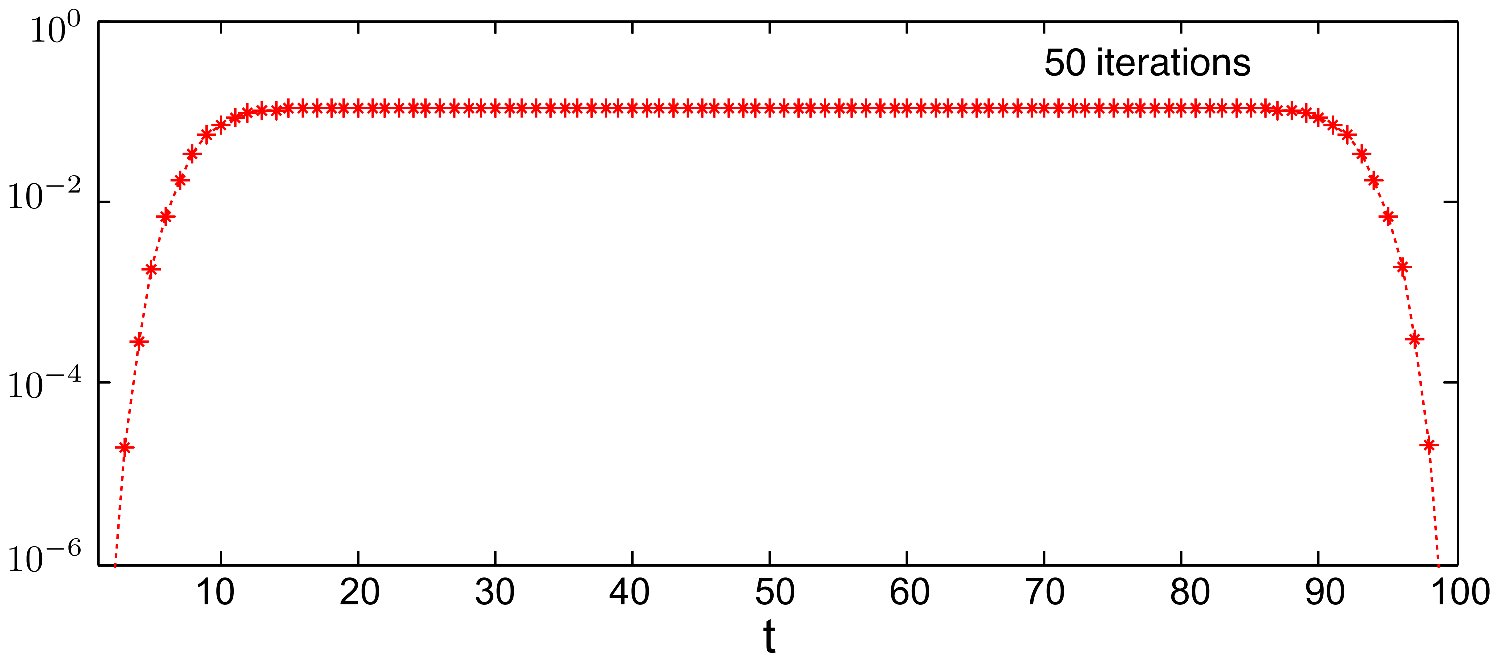
Evolution of message probabilities $(L = 100)$:



Thresholds of SC-LDPC Codes

- Variable nodes all have the **same degree** as the block code.
- Check nodes with **lower degrees** (at the ends) improve the BP decoder.

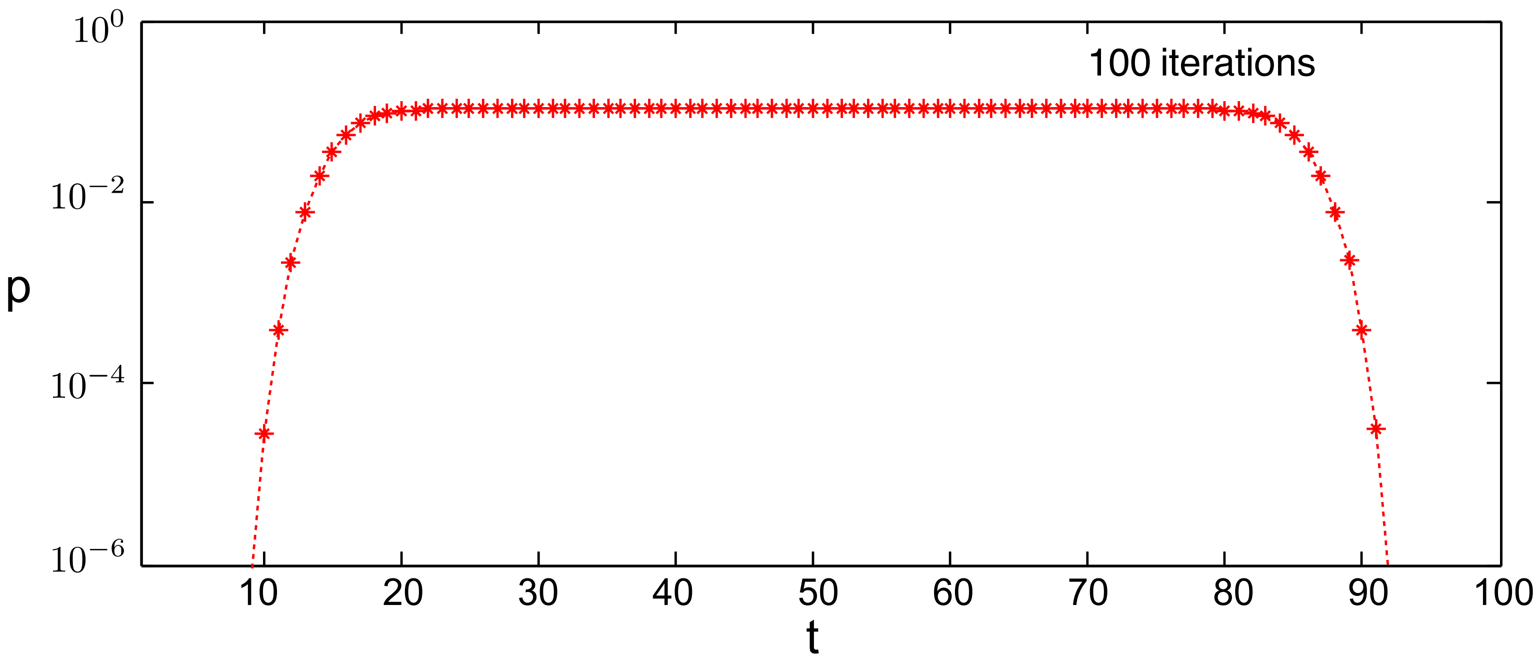
Evolution of message probabilities $(L = 100)$:



Thresholds of SC-LDPC Codes

- Variable nodes all have the **same degree** as the block code.
- Check nodes with **lower degrees** (at the ends) improve the BP decoder.

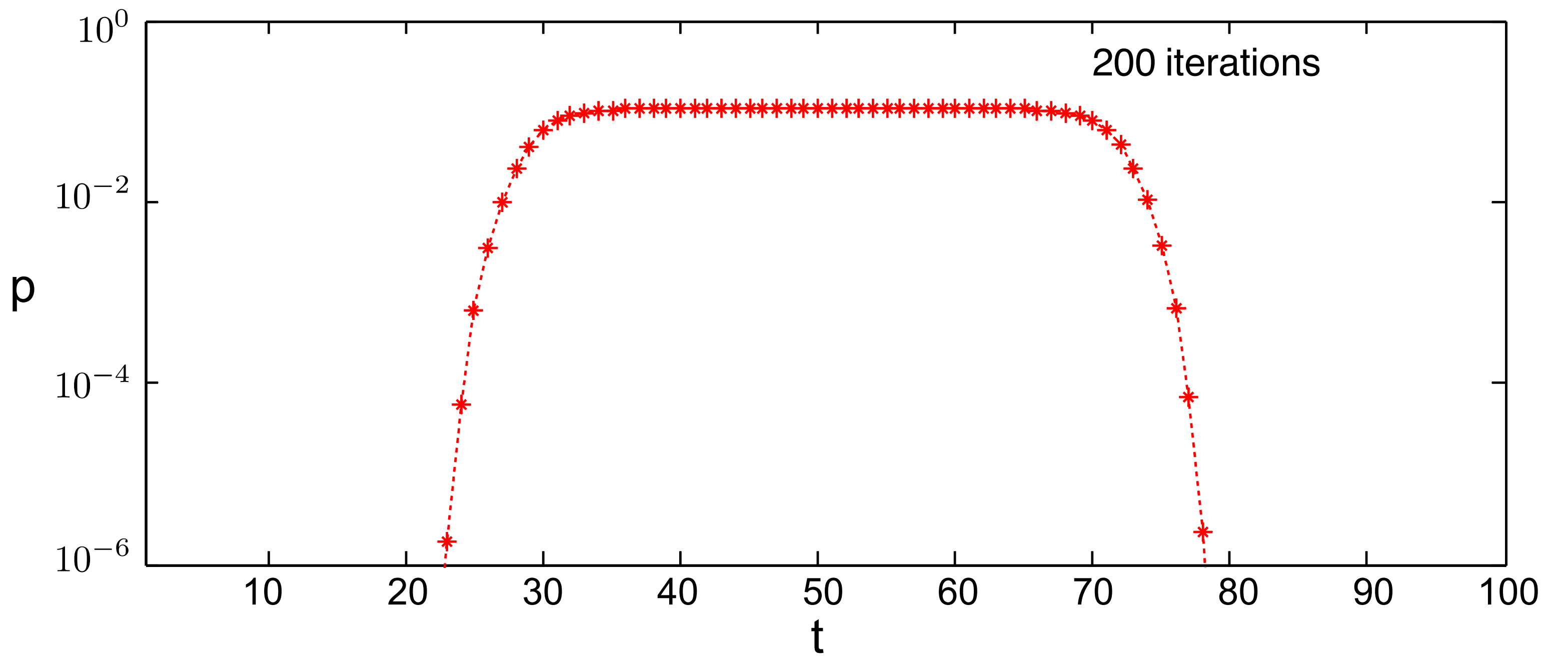
Evolution of message probabilities $(L = 100)$:



Thresholds of SC-LDPC Codes

- Variable nodes all have the **same degree** as the block code.
- Check nodes with **lower degrees** (at the ends) improve the BP decoder.

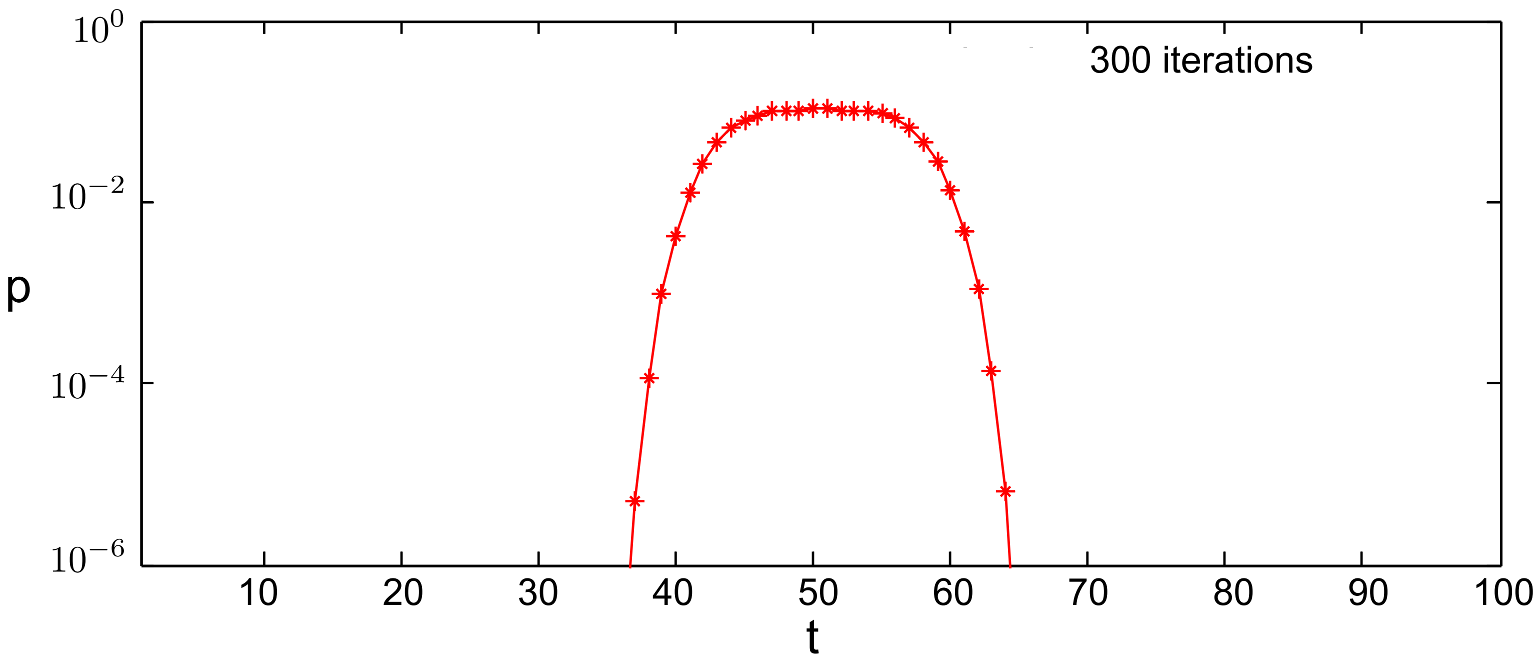
Evolution of message probabilities $(L = 100)$:



Thresholds of SC-LDPC Codes

- Variable nodes all have the **same degree** as the block code.
- Check nodes with **lower degrees** (at the ends) improve the BP decoder.

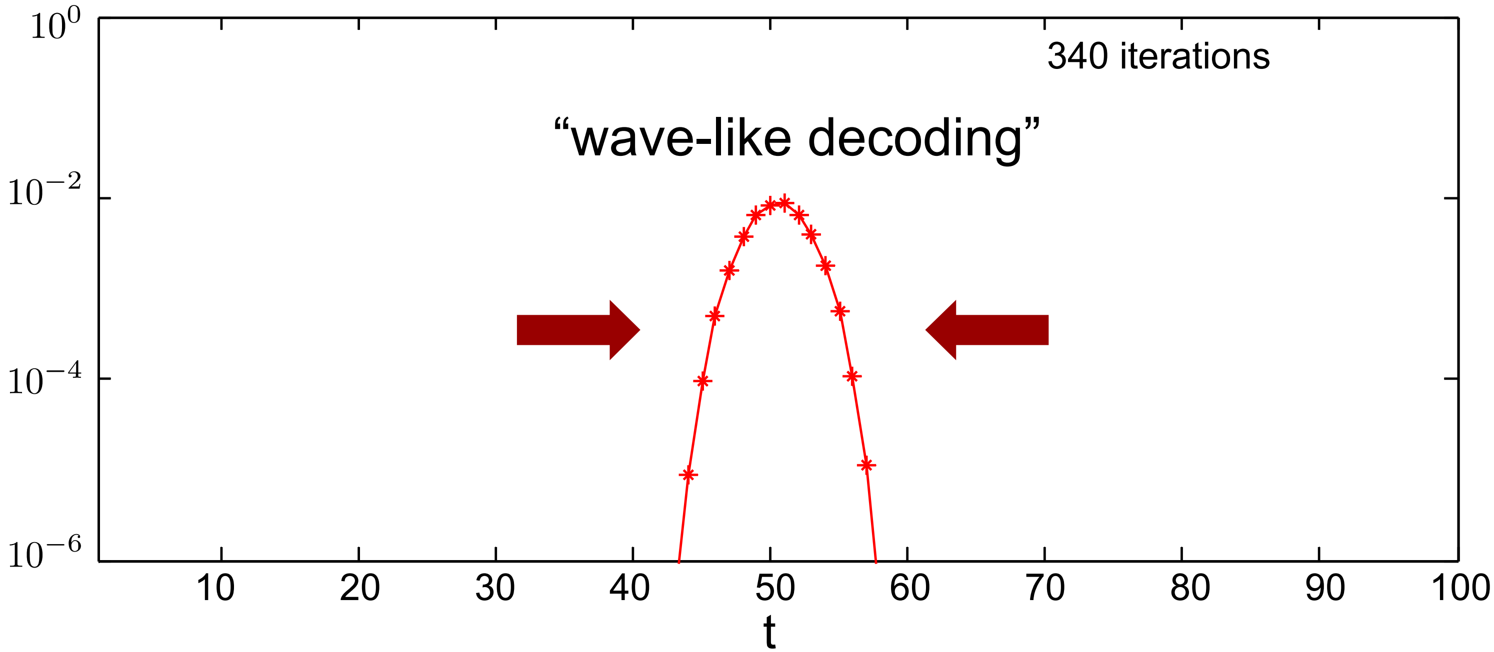
Evolution of message probabilities $(L = 100)$:



Thresholds of SC-LDPC Codes

- Variable nodes all have the **same degree** as the block code.
- Check nodes with **lower degrees** (at the ends) improve the BP decoder.

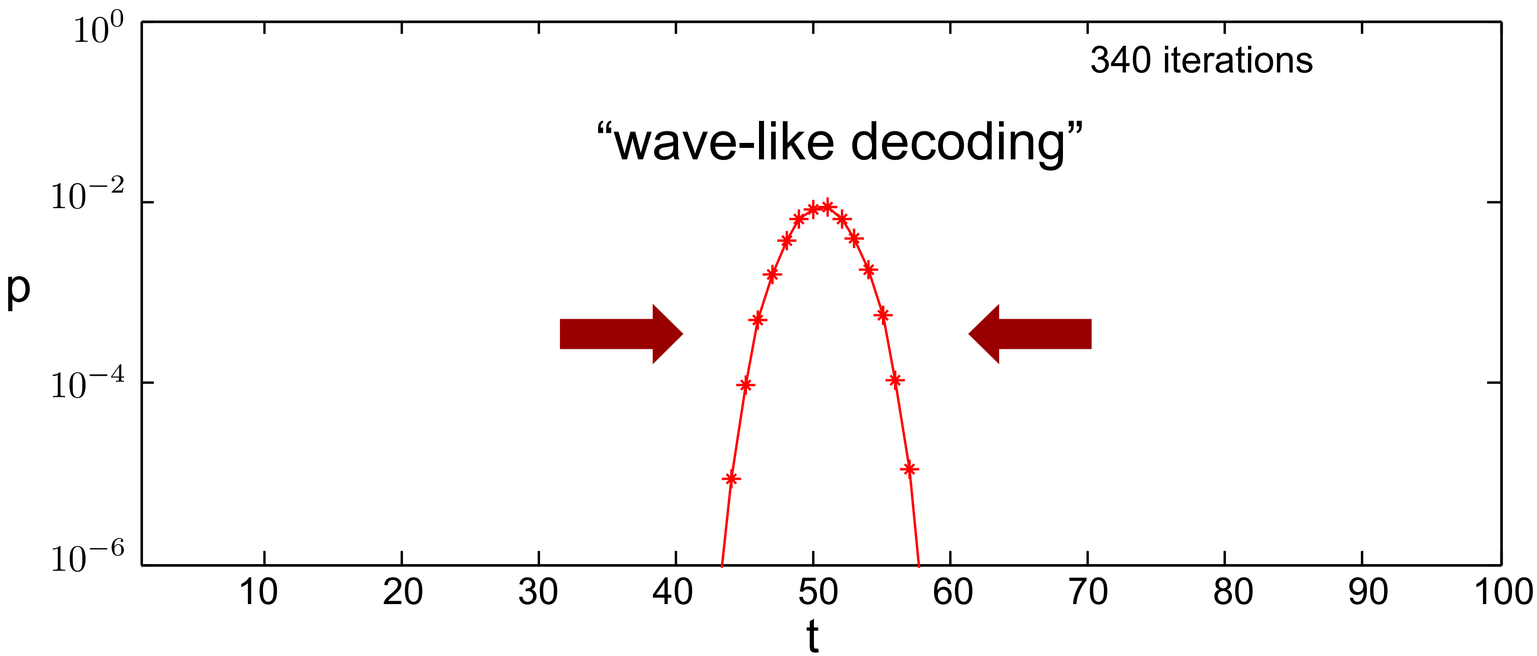
Evolution of message probabilities $(L = 100)$:



Thresholds of SC-LDPC Codes

- Variable nodes all have the **same degree** as the block code.
- Check nodes with **lower degrees** (at the ends) improve the BP decoder.

Evolution of message probabilities $(L = 100)$:

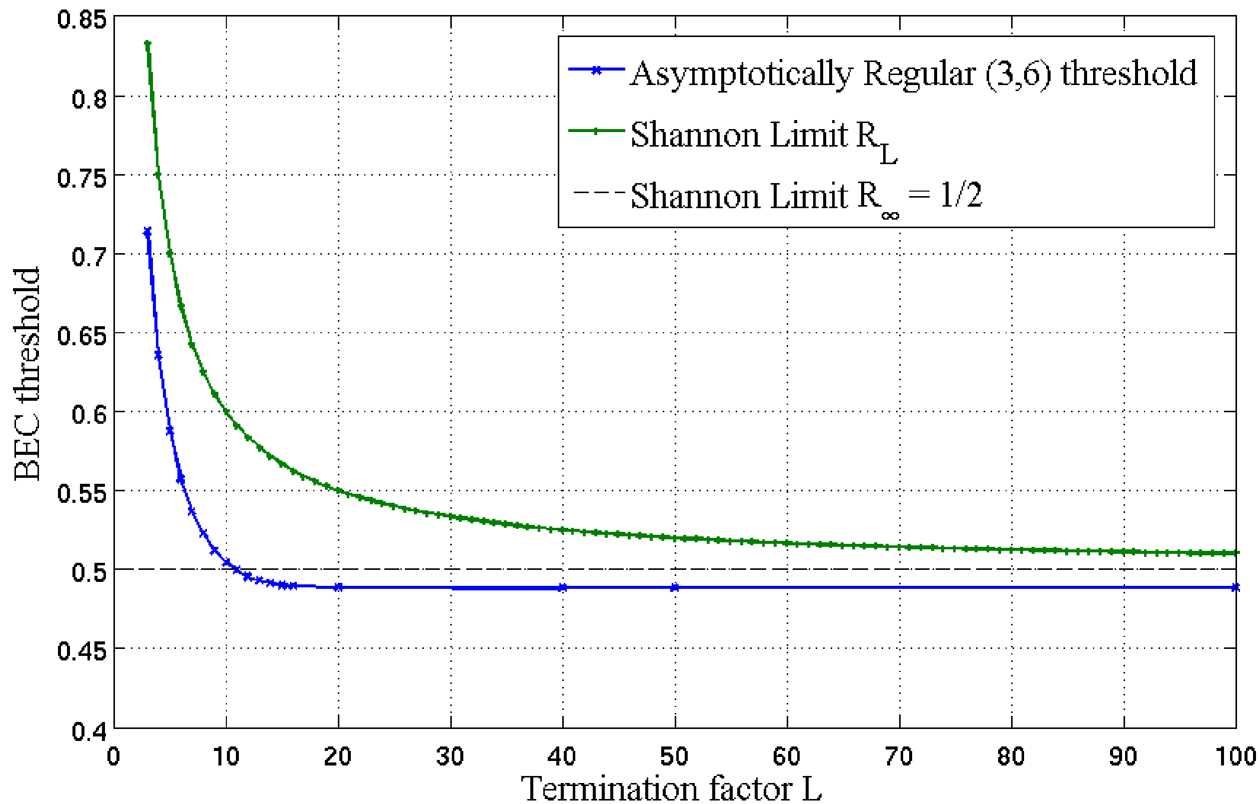


- Note: the **fraction** of **lower degree** nodes tends to zero as $L \to \infty$, i.e., the codes are **asymptotically regular**.

Thresholds of SC-LDPC Codes

- **Density evolution** can be applied to the protograph-based ensembles with $M \to \infty$ [Sridharan et al. '04]:

Example: BEC



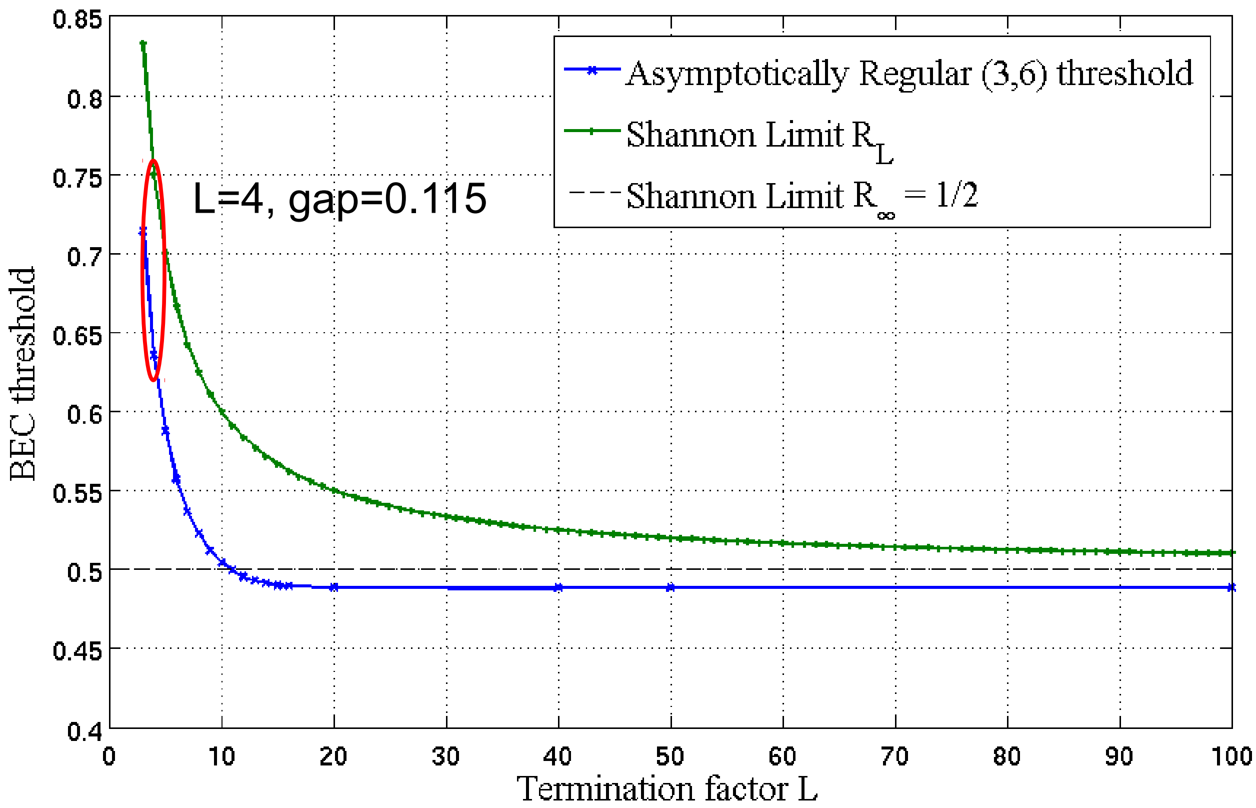
Thresholds of SC-LDPC Codes

- **Density evolution** can be applied to the protograph-based ensembles with $M \to \infty$ [Sridharan et al. '04]:

Example: BEC

$L = 4$, $R = 1/4$

$\varepsilon^* = 0.635$, $\varepsilon_{\text{Sh}} = 0.75$



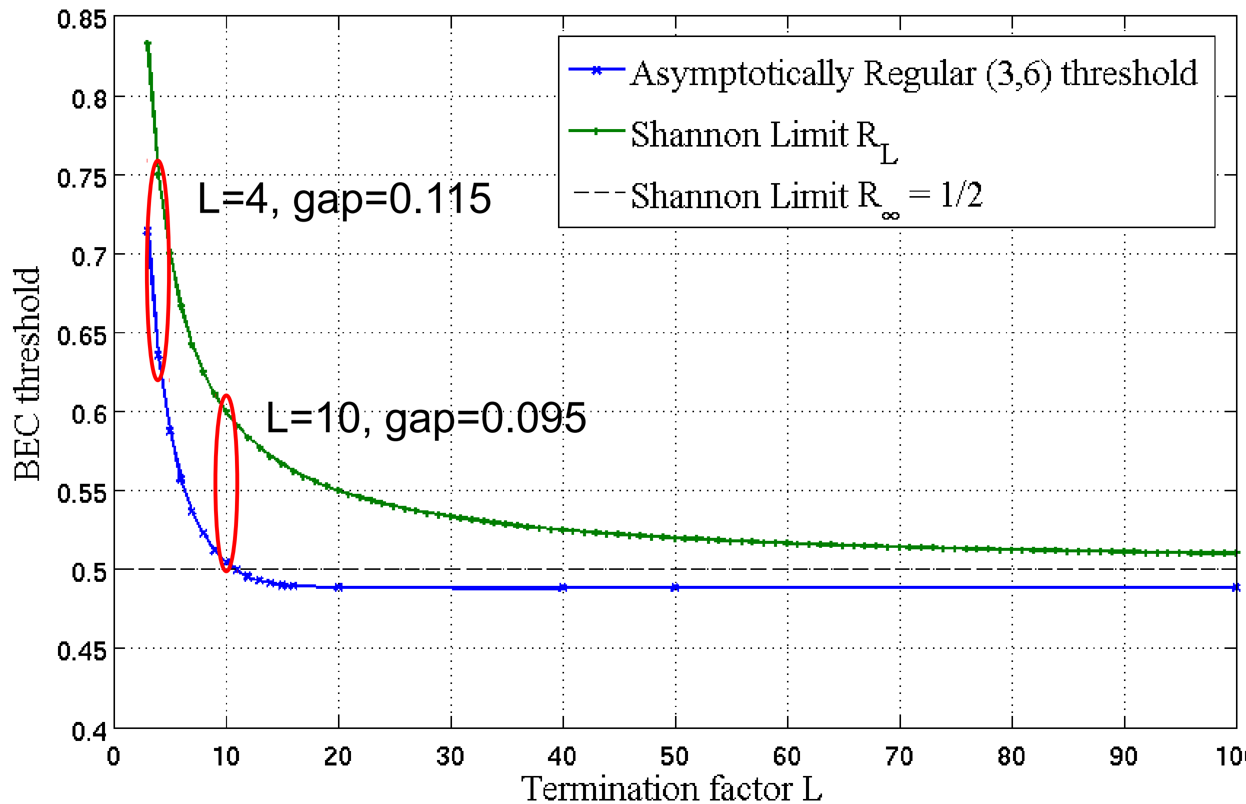
Thresholds of SC-LDPC Codes

- **Density evolution** can be applied to the protograph-based ensembles with $M \to \infty$ [Sridharan et al. '04]:

Example: BEC

$L = 4,\ R = 1/4$
$\varepsilon^* = 0.635,\ \varepsilon_{\text{Sh}} = 0.75$

$L = 10,\ R = 2/5$
$\varepsilon^* = 0.505,\ \varepsilon_{\text{Sh}} = 0.6$



Thresholds of SC-LDPC Codes

- **Density evolution** can be applied to the protograph-based ensembles with $M \to \infty$ [Sridharan et al. '04]:

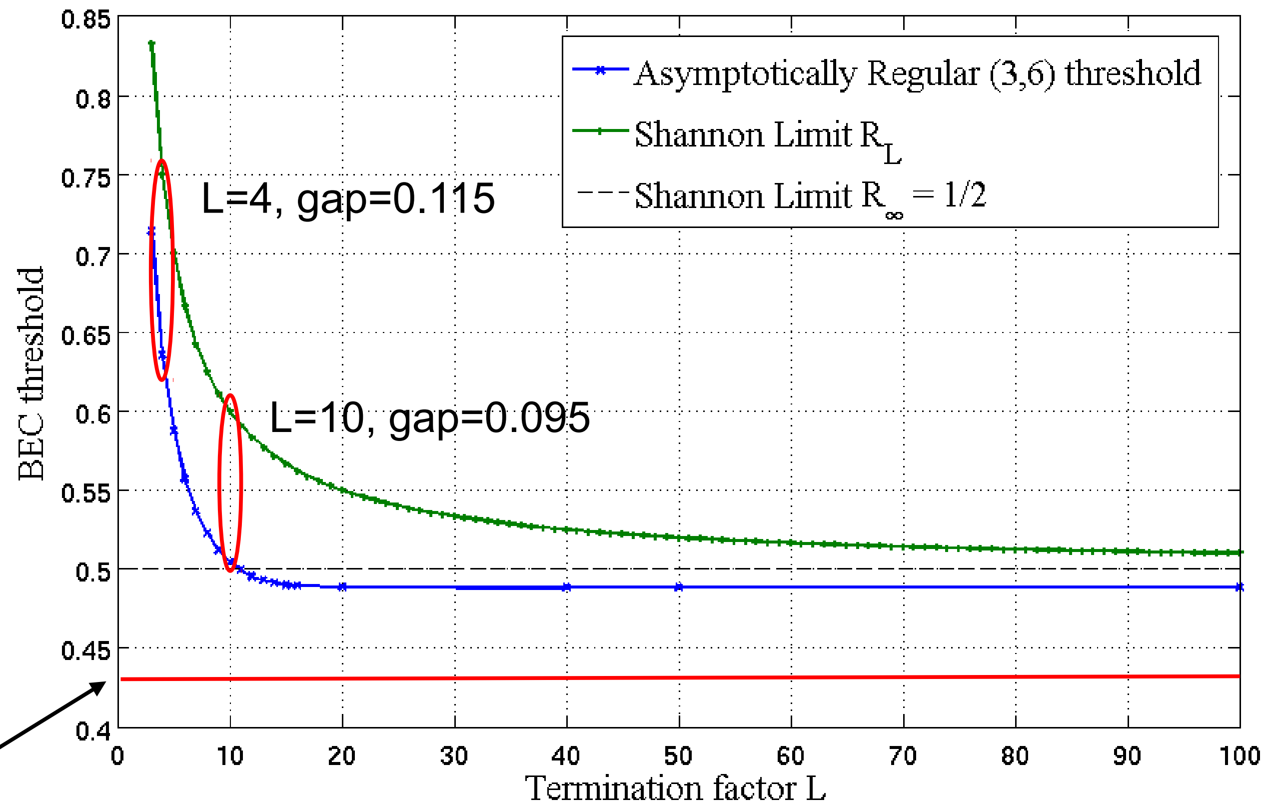
Example: BEC

$L = 4$, $R = 1/4$
$\varepsilon^* = 0.635$, $\varepsilon_{\mathrm{Sh}} = 0.75$

$L = 10$, $R = 2/5$
$\varepsilon^* = 0.505$, $\varepsilon_{\mathrm{Sh}} = 0.6$

⋮

$L \to \infty$, $R \to 1/2$
$\varepsilon^* = 0.488$, $\varepsilon_{\mathrm{Sh}} = 0.5$



(3,6)-regular block code:

$\varepsilon^* = 0.429$

Thresholds of SC-LDPC Codes

Iterative decoding thresholds (protograph-based ensembles)

BEC

(J, K)	ϵ^*_{SC}	$\epsilon^*_{\mathrm{blk}}$
(3,6)	0.488	0.429
(4,8)	0.497	0.383
(5,10)	0.499	0.341

AWGN

(J, K)	$E_b/N_{o\,\mathrm{SC}}$	$E_b/N_{o\,\mathrm{blk}}$
(3,6)	0.46 dB	1.11 dB
(4,8)	0.26 dB	1.61 dB
(5,10)	0.21 dB	2.04 dB

- We observe a **significant improvement** in the thresholds of SC-LDPC codes compared to the associated LDPC block codes (LDPC-BCs) due to the lower degree check nodes at the ends of the graph and the wave-like decoding.

[LSCZ10] M. Lentmaier, A. Sridharan, D. J. Costello, Jr., and K.Sh. Zigangirov, “Iterative decoding threshold analysis for LDPC convolutional codes,” *IEEE Trans. Inf. Theory*, 56:10, Oct. 2010.

Thresholds of SC-LDPC Codes

UNIVERSITY OF NOTRE DAME

Iterative decoding thresholds (protograph-based ensembles)

BEC

(J, K)	ϵ^*_{SC}	$\epsilon^*_{\mathrm{blk}}$
(3,6)	0.488	0.429
(4,8)	0.497	0.383
(5,10)	0.499	0.341

AWGN

(J, K)	E_b/N_o SC	E_b/N_o blk
(3,6)	0.46 dB	1.11 dB
(4,8)	0.26 dB	1.61 dB
(5,10)	0.21 dB	2.04 dB

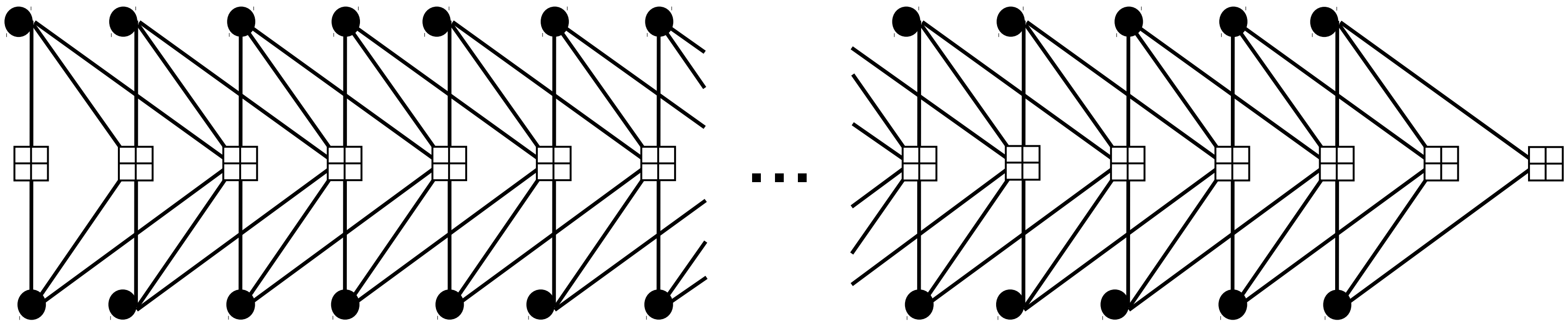
- We observe a **significant improvement** in the thresholds of SC-LDPC codes compared to the associated LDPC block codes (LDPC-BCs) due to the lower degree check nodes at the ends of the graph and the wave-like decoding.
- In contrast to LDPC-BCs, the iterative decoding thresholds of SC-LDPC codes **improve** as the graph density increases.

[LSCZ10] M. Lentmaier, A. Sridharan, D. J. Costello, Jr., and K.Sh. Zigangirov, "Iterative decoding threshold analysis for LDPC convolutional codes," *IEEE Trans. Inf. Theory*, 56:10, Oct. 2010.

Why are SC-LDPC Codes Better?

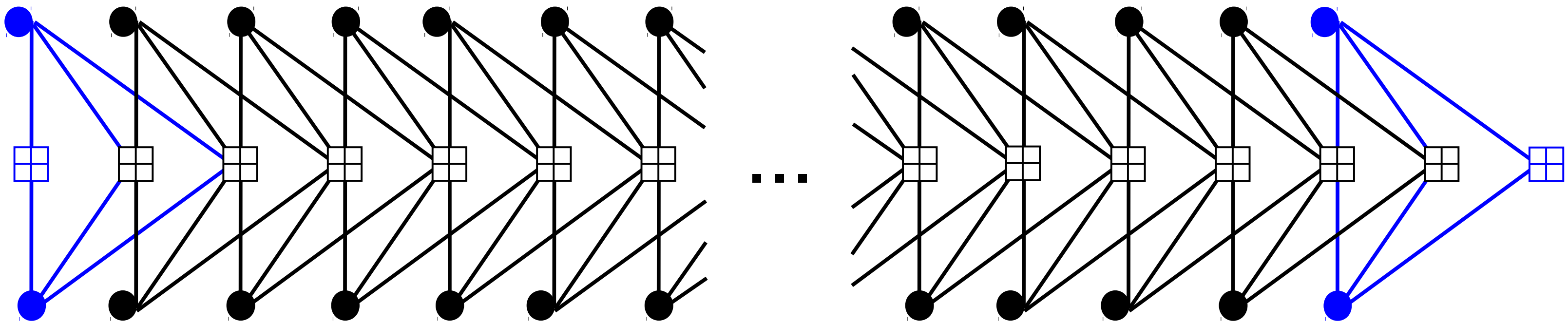
UNIVERSITY OF NOTRE DAME

- When symbols are perfectly known (BEC), all adjacent edges can be removed from the Tanner graph.



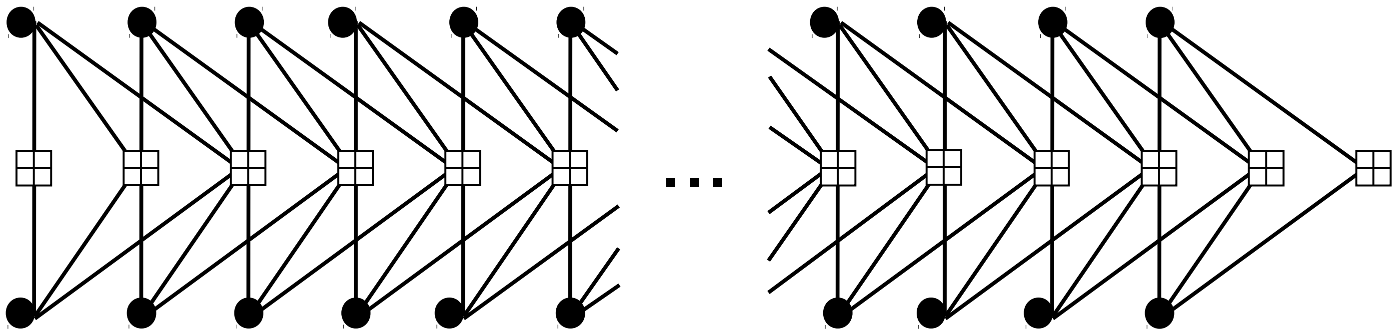
Why are SC-LDPC Codes Better?

- When symbols are perfectly known (BEC), all adjacent edges can be removed from the Tanner graph.



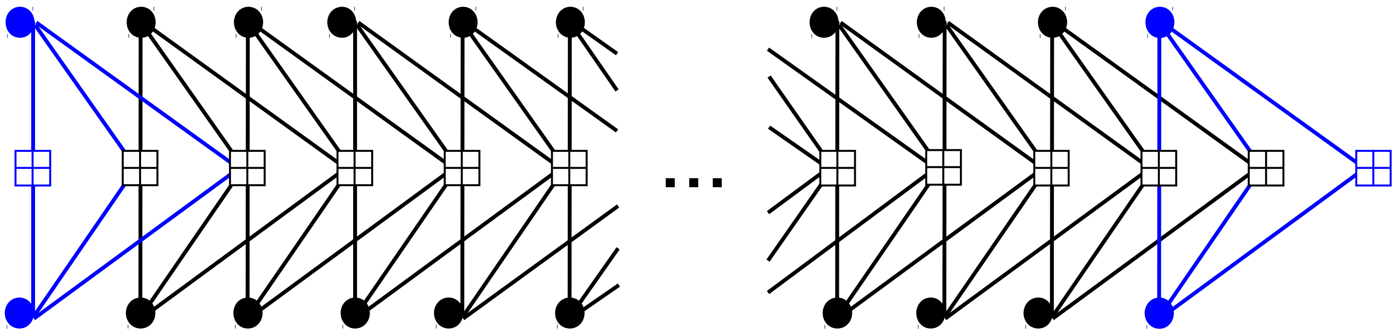
Why are SC-LDPC Codes Better?

- When symbols are perfectly known (BEC), all adjacent edges can be removed from the Tanner graph.



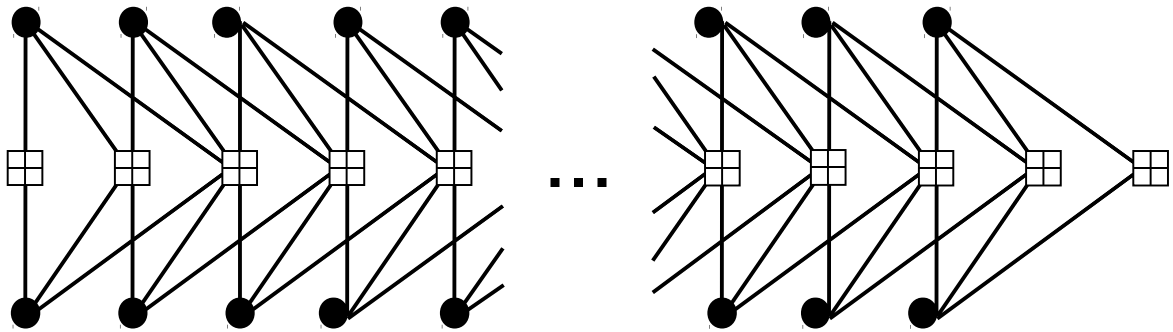
Why are SC-LDPC Codes Better?

- When symbols are perfectly known (BEC), all adjacent edges can be removed from the Tanner graph.



Why are SC-LDPC Codes Better?

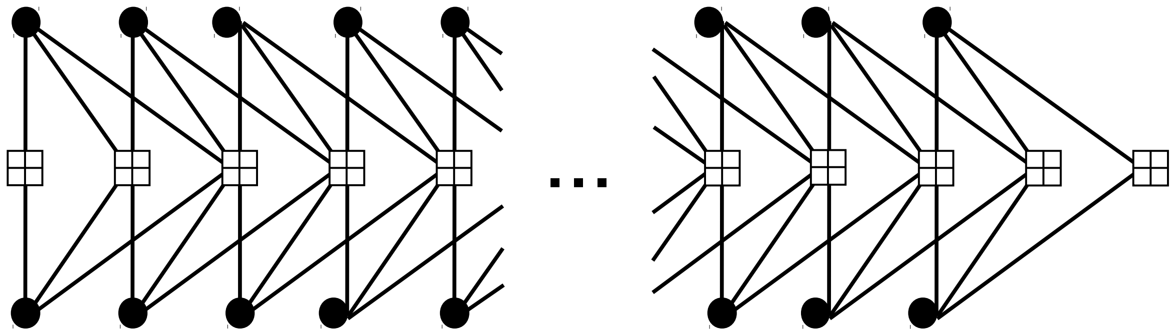
- When symbols are perfectly known (BEC), all adjacent edges can be removed from the Tanner graph.



Why are SC-LDPC Codes Better?

UNIVERSITY OF NOTRE DAME

- When symbols are perfectly known (BEC), all adjacent edges can be removed from the Tanner graph.



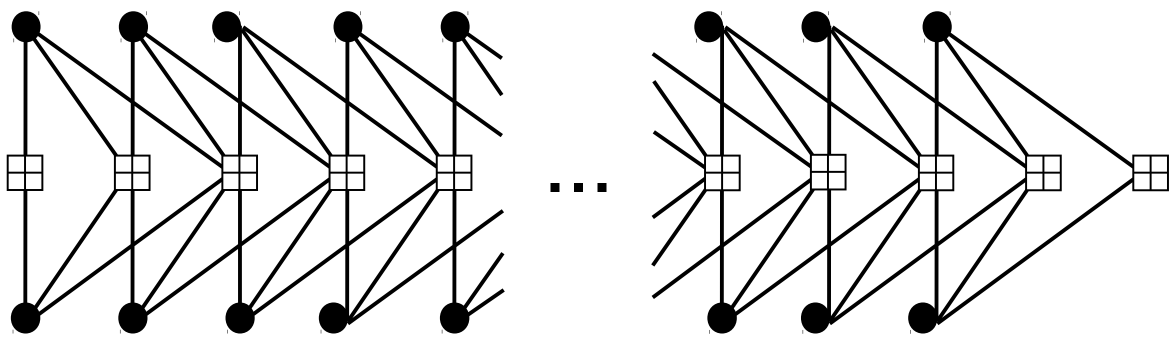
- The **threshold saturates** (converges) to a fixed value numerically indistinguishable from the **maximum a posteriori** (MAP) threshold of the (J, K)-regular LDPC-BC ensemble as $L \to \infty$ [LSCZ10].

[LSCZ10] M. Lentmaier, A. Sridharan, D. J. Costello, Jr., and K.Sh. Zigangirov, "Iterative decoding threshold analysis for LDPC convolutional codes," *IEEE Trans. Inf. Theory*, 56:10, Oct. 2010.

Why are SC-LDPC Codes Better?

UNIVERSITY OF NOTRE DAME

- When symbols are perfectly known (BEC), all adjacent edges can be removed from the Tanner graph.



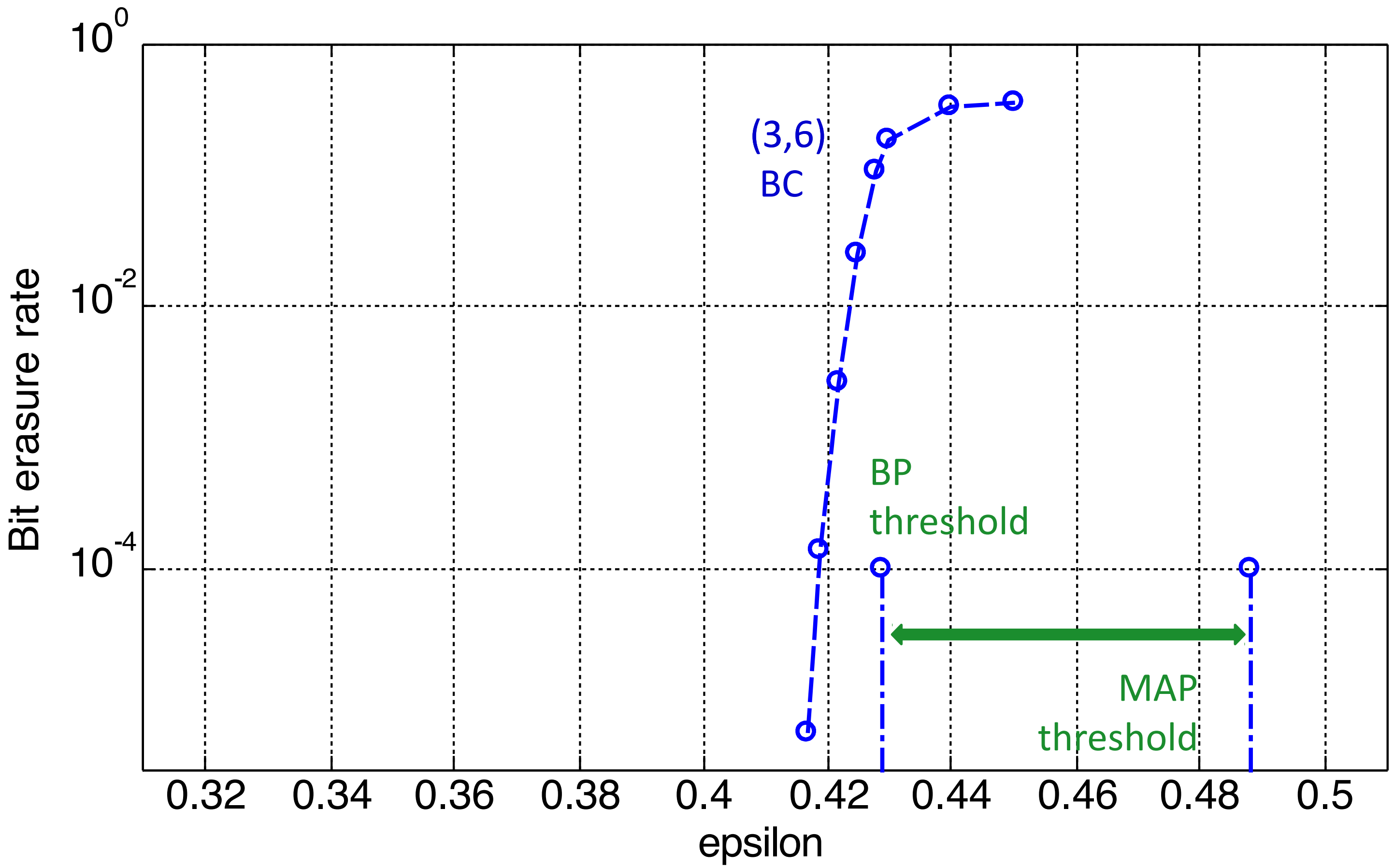
- The **threshold saturates** (converges) to a fixed value numerically indistinguishable from the **maximum a posteriori** (MAP) threshold of the (J, K)-regular LDPC-BC ensemble as $L \to \infty$ [LSCZ10].
- For a more random-like ensemble, this has been proven analytically, first for the BEC [KRU11], then for all BMS channels [KRU13].

[LSCZ10] M. Lentmaier, A. Sridharan, D. J. Costello, Jr., and K.Sh. Zigangirov, "Iterative decoding threshold analysis for LDPC convolutional codes," *IEEE Trans. Inf. Theory*, 56:10, Oct. 2010.
[KRU11] S. Kudekar, T. J. Richardson and R. Urbanke, "Threshold saturation via spatial coupling: why convolutional LDPC ensembles perform so well over the BEC", *IEEE Trans. on Inf. Theory*, 57:2, 2011
[KRU13] S. Kudekar, T. J. Richardson and R. Urbanke, "Spatially coupled ensembles universally achieve capacity under belief propagation", *IEEE Trans. on Inf. Theory*, 59:12, 2013.

Threshold Saturation (BEC)

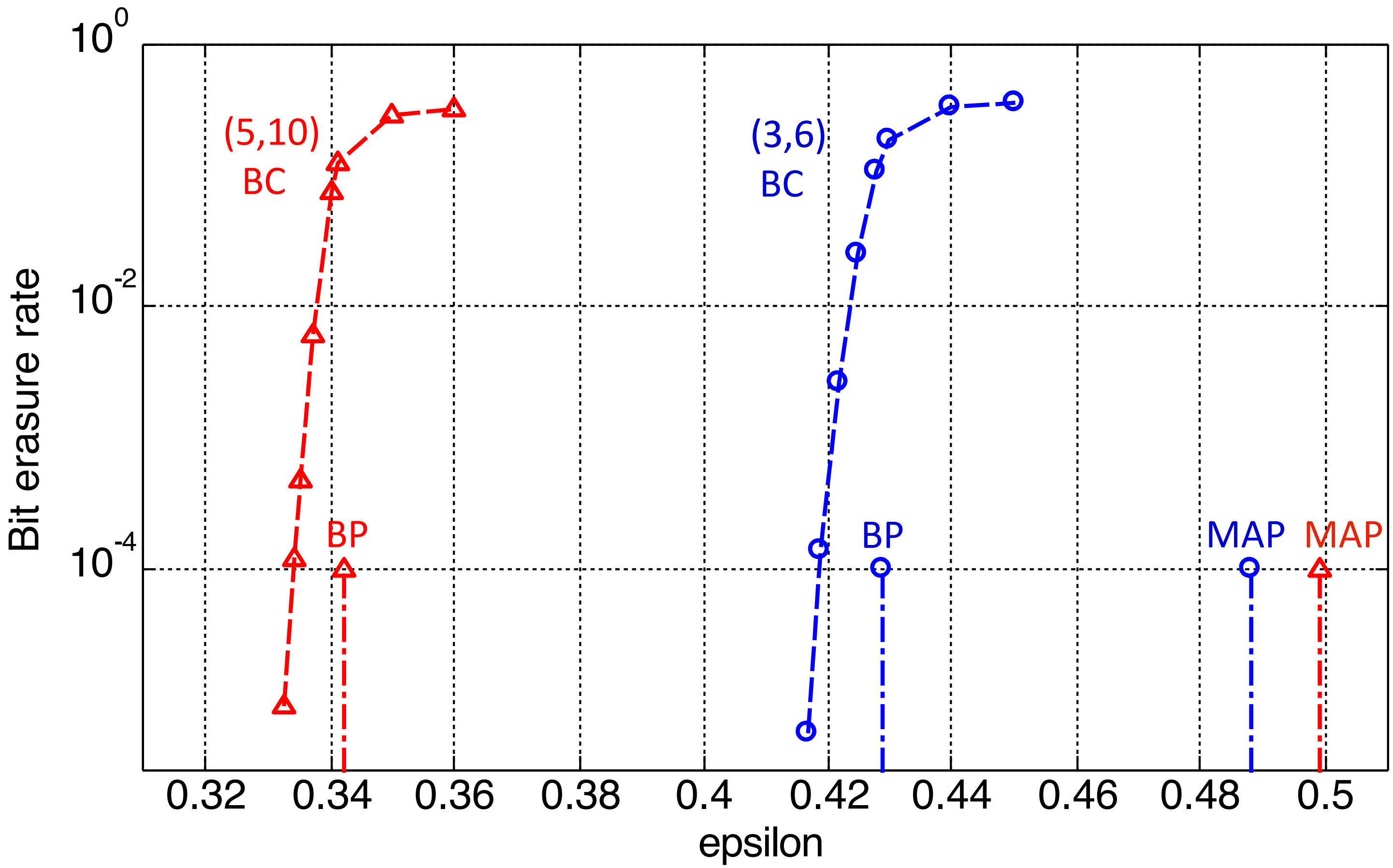
UNIVERSITY OF NOTRE DAME

BP = iterative (suboptimal) decoding threshold
MAP = (optimal) maximum a posteriori threshold



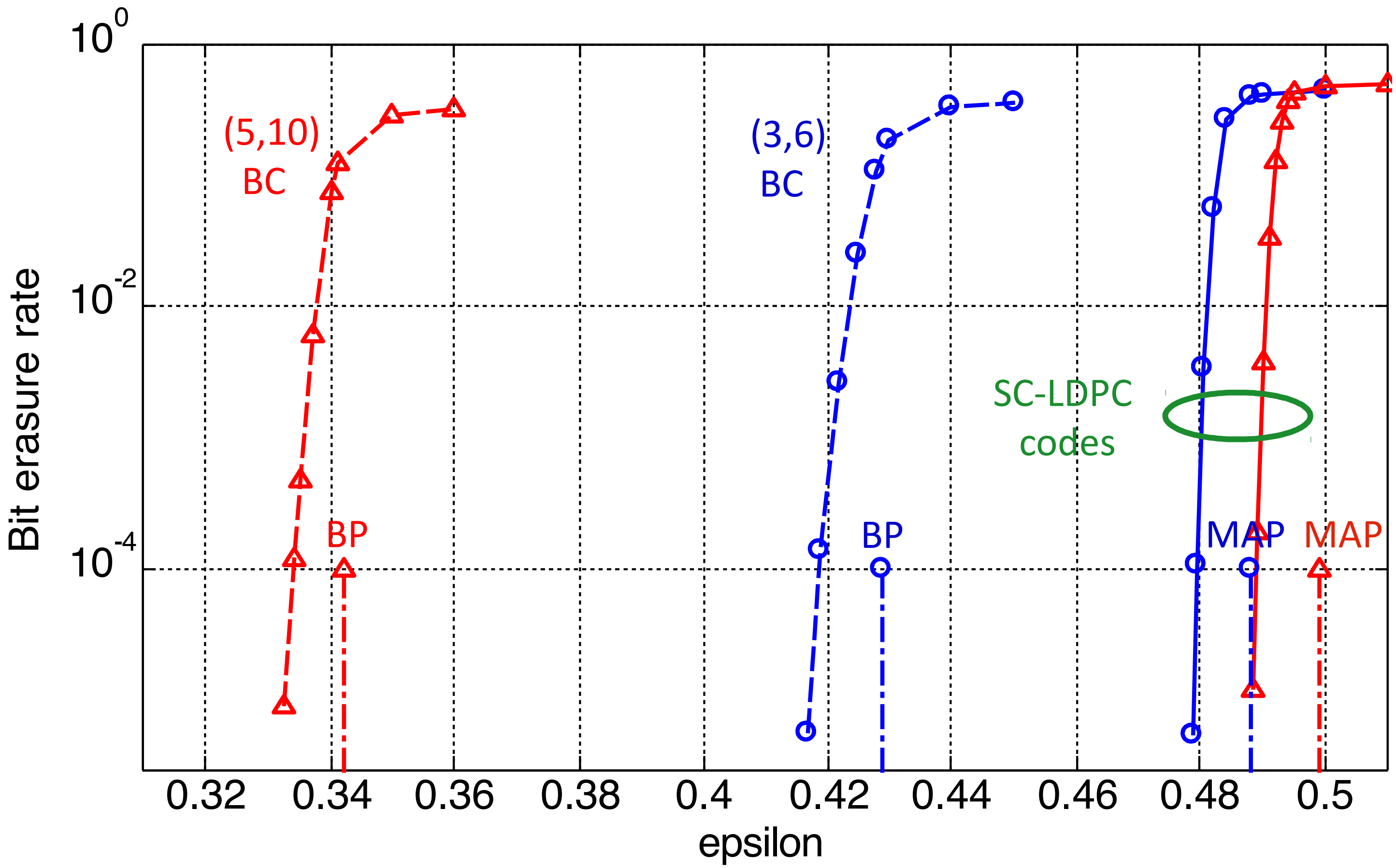
Threshold Saturation (BEC)

BP = iterative (suboptimal) decoding threshold
MAP = (optimal) maximum a posteriori threshold



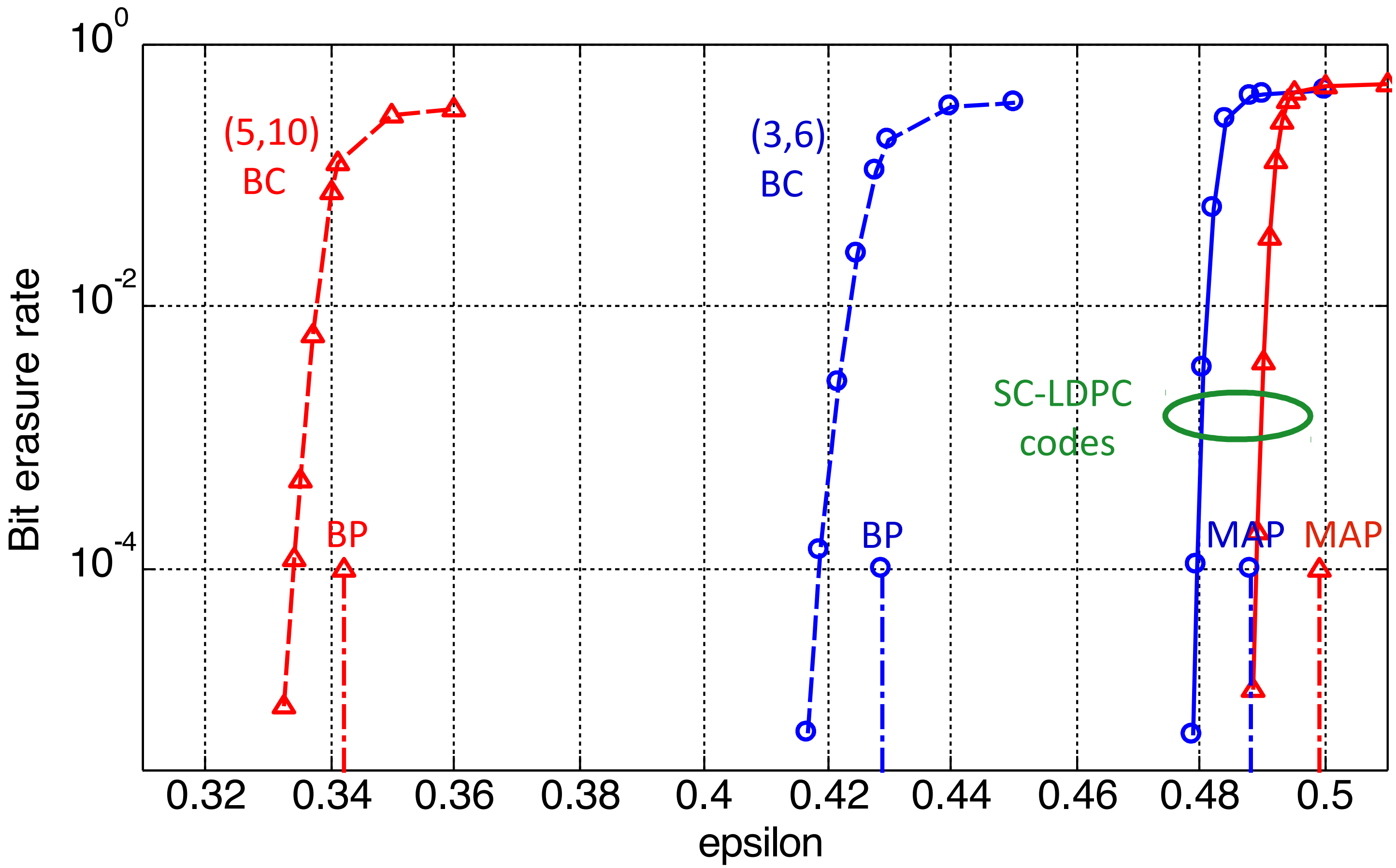
Threshold Saturation (BEC)

BP = iterative (suboptimal) decoding threshold
MAP = (optimal) maximum a posteriori threshold



Threshold Saturation (BEC)

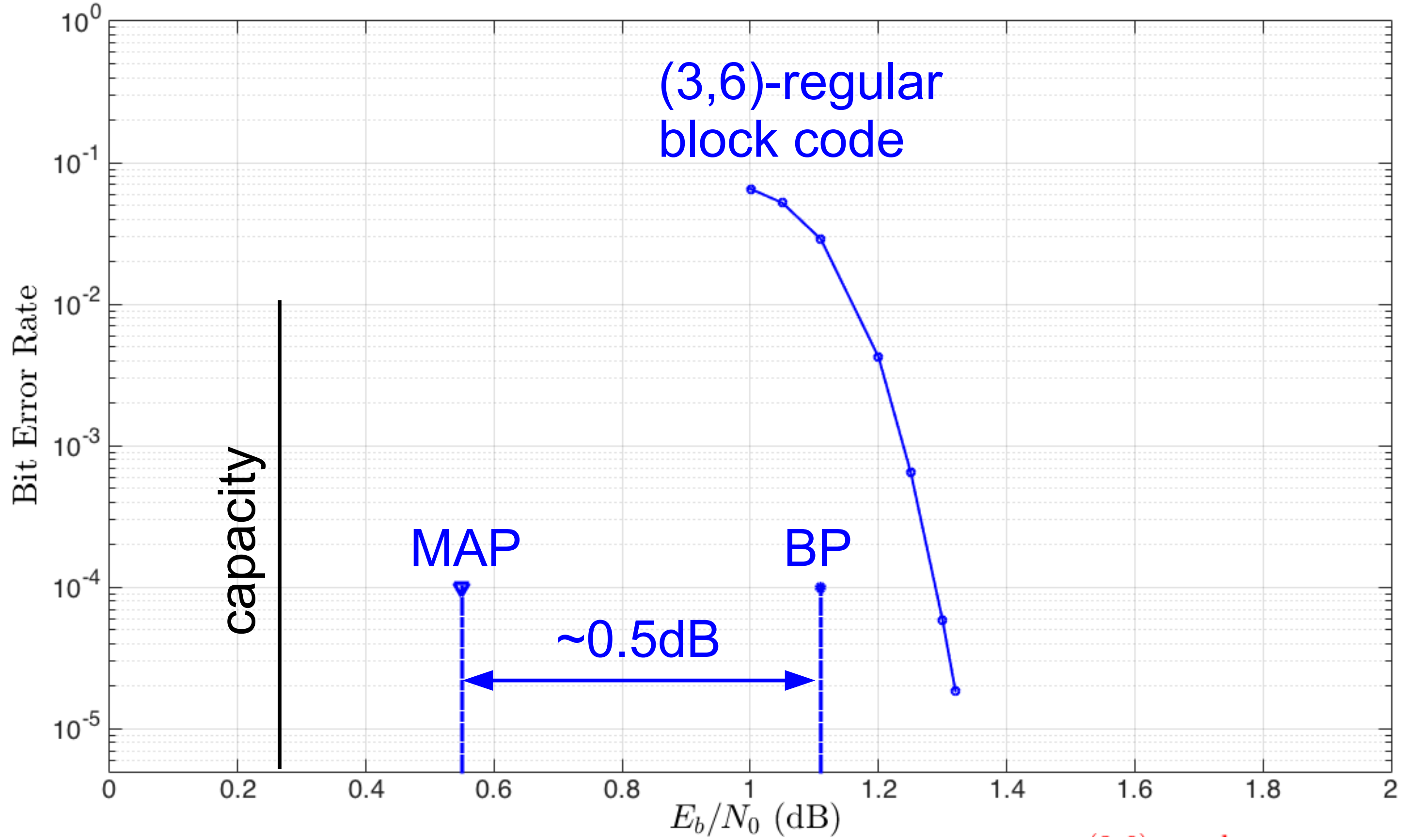
BP = iterative (suboptimal) decoding threshold
MAP = (optimal) maximum a posteriori threshold



➡ **optimal** decoding performance with a **suboptimal** iterative algorithm!

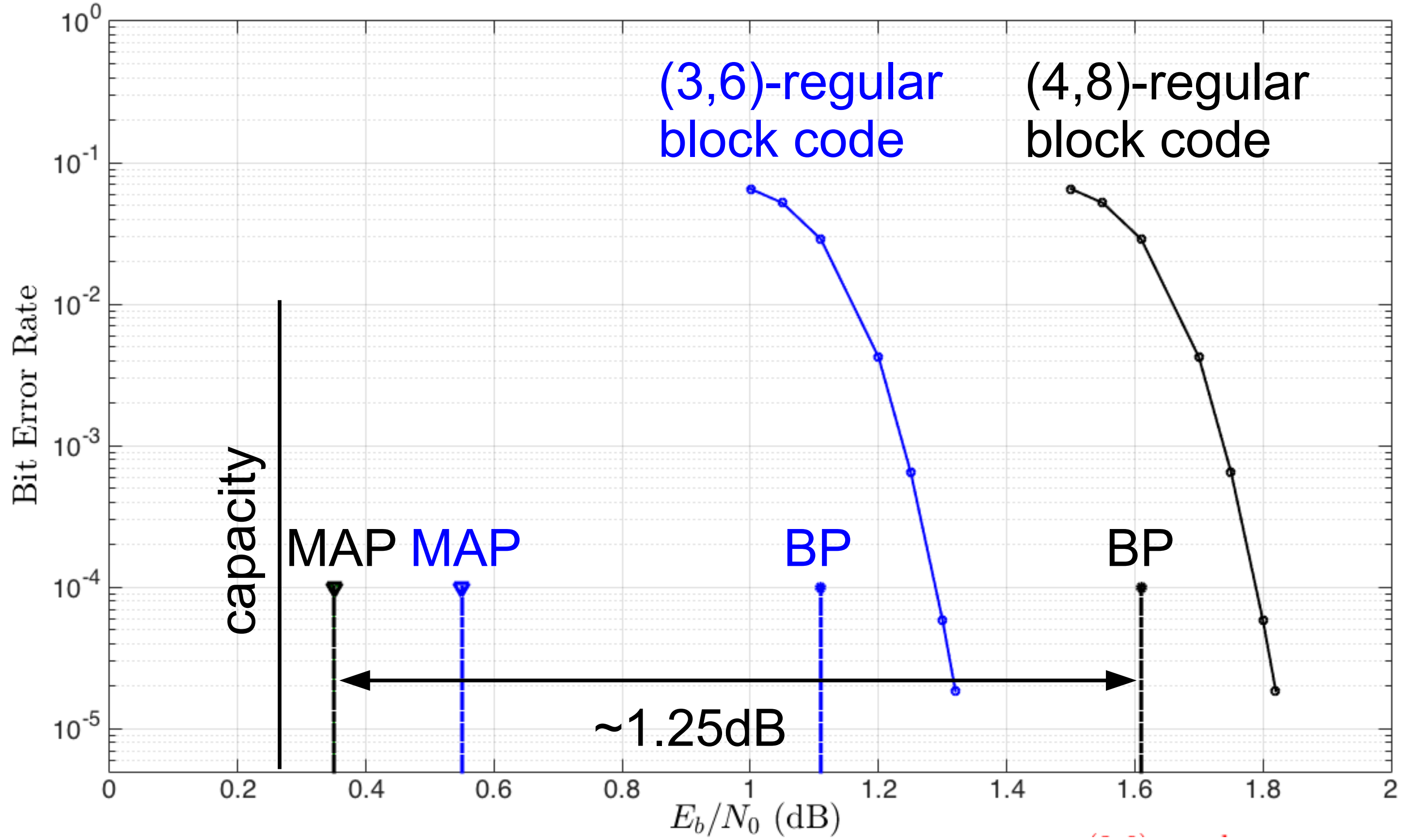
Threshold Saturation (AWGNC)

BP = iterative (suboptimal) decoding threshold
MAP = (optimal) maximum a posteriori threshold



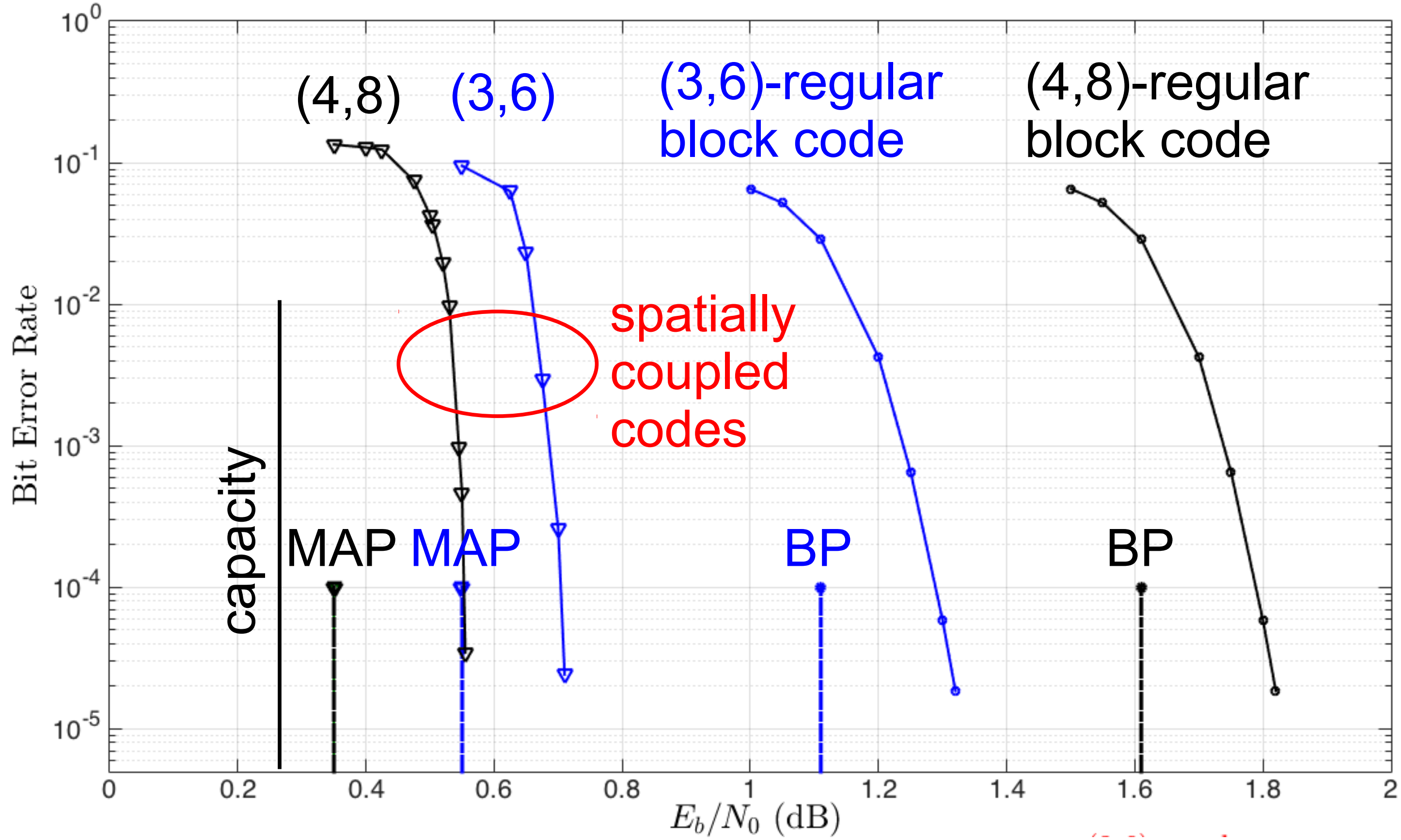
Threshold Saturation (AWGNC)

BP = iterative (suboptimal) decoding threshold
MAP = (optimal) maximum a posteriori threshold



Threshold Saturation (AWGNC)

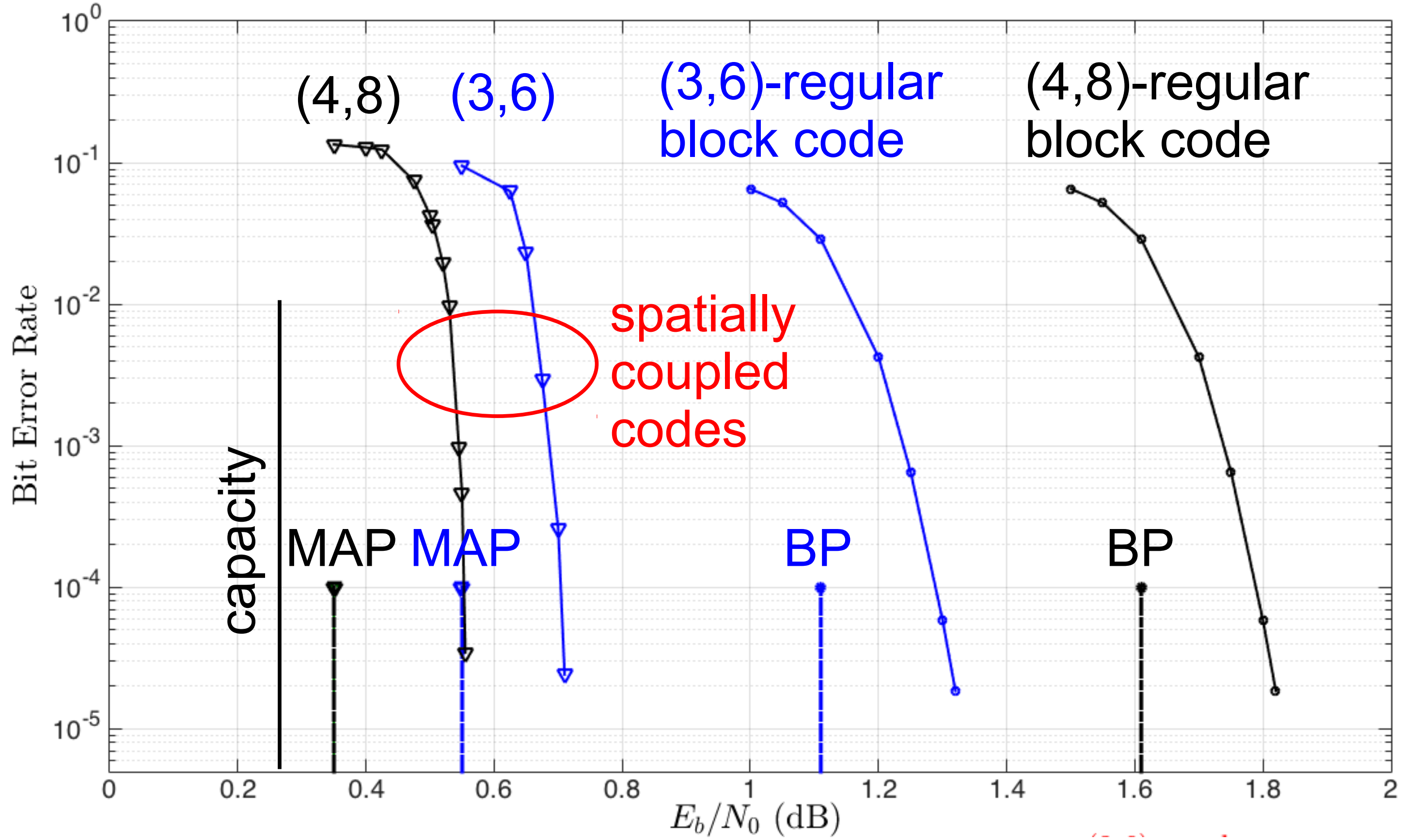
BP = iterative (suboptimal) decoding threshold
MAP = (optimal) maximum a posteriori threshold



Threshold Saturation (AWGNC)

UNIVERSITY OF NOTRE DAME

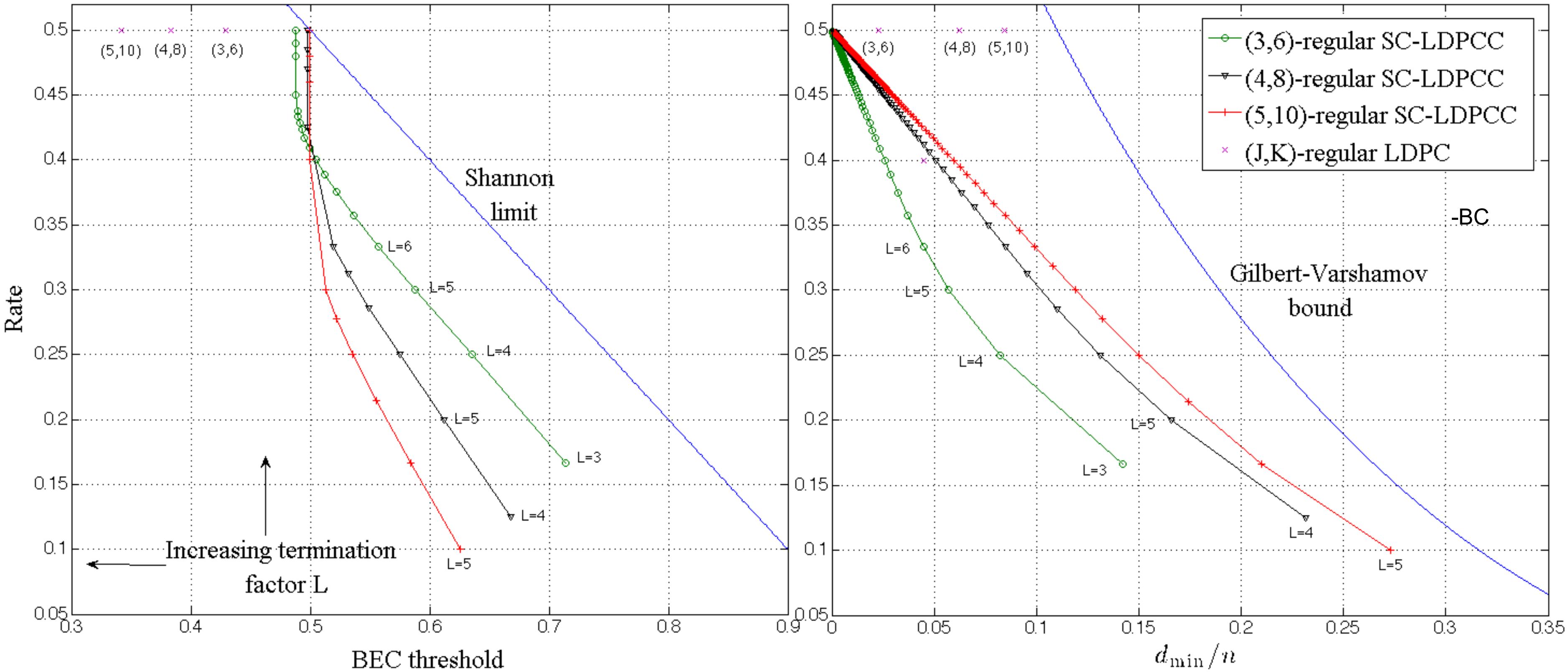
BP = iterative (suboptimal) decoding threshold
MAP = (optimal) maximum a posteriori threshold



➡ **optimal** decoding performance with a **suboptimal** iterative algorithm!

BEC Thresholds vs Distance Growth

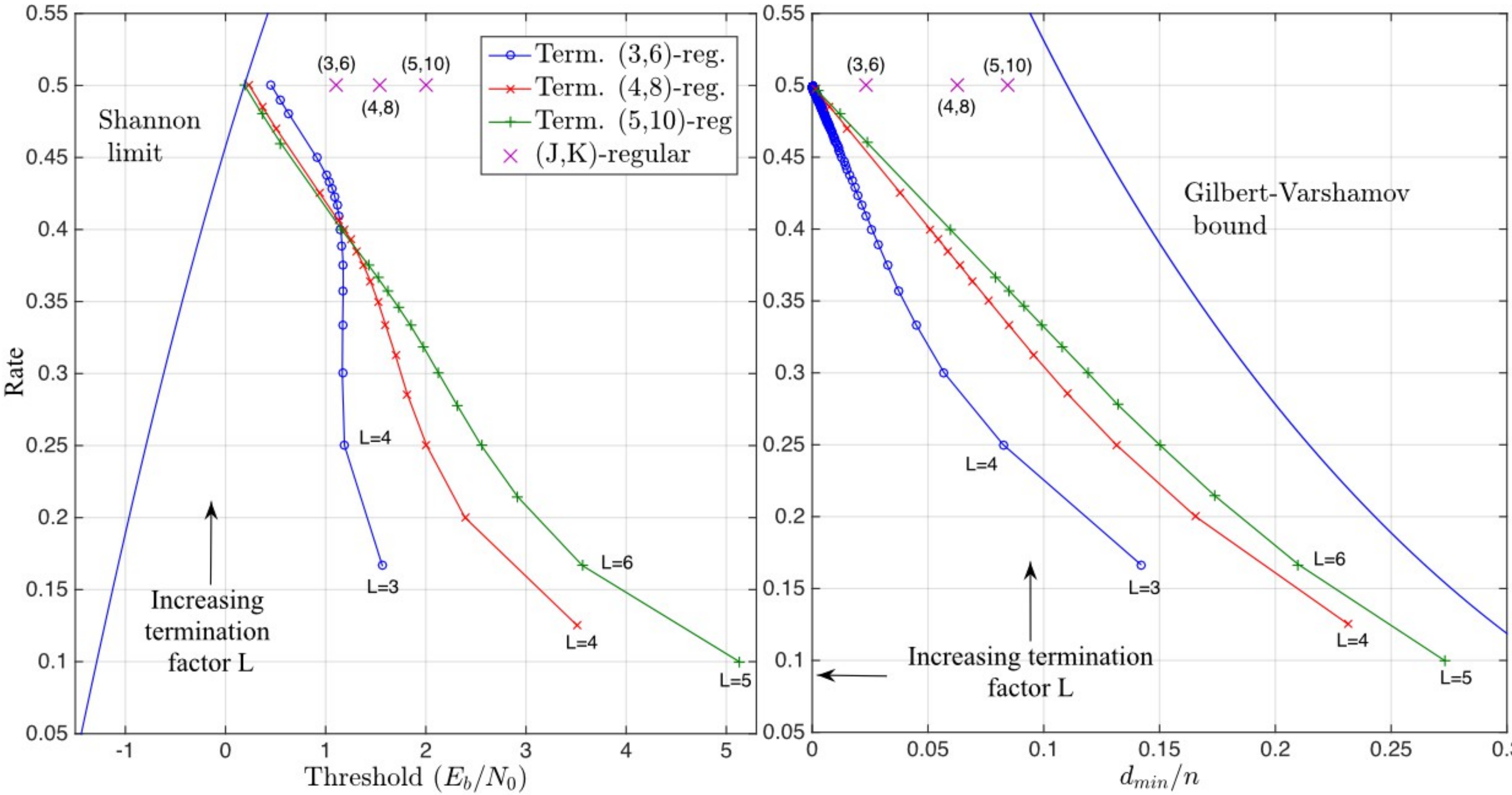
- By increasing J and K, we obtain **capacity achieving** (J,K)-regular SC-LDPC code ensembles with linear minimum distance growth.



- (J,K)-regular SC-LDPC codes combine the best features of irregular and regular LDPC-BCs, i.e., capacity approaching thresholds and linear distance growth.

AWGNC Thresholds vs. Distance Growth

- Similar results are obtained for the AWGNC



[MLC10] D. G. M. Mitchell, M. Lentmaier and D. J. Costello, Jr., "AWGN Channel Analysis of Terminated LDPC Convolutional Codes", *Proc. Information Theory and Applications Workshop*, San Diego, Feb. 2011.

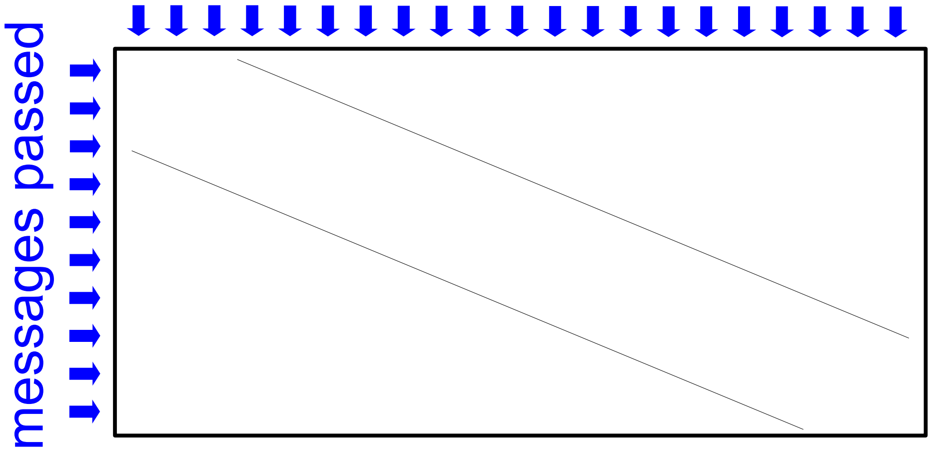
Outline

UNIVERSITY OF NOTRE DAME

- Introduction: From Shannon to Modern Coding Theory
  - ➡ Channel capacity, structured codes, random codes, LDPC codes
- LDPC Block Codes
  - ➡ Parity-check matrix and Tanner graph representations, minimum distance bounds, iterative decoding thresholds, protograph-based constructions
- Spatially Coupled LDPC Codes
  - ➡ Protograph representation, edge-spreading construction, termination
  - ➡ Iterative decoding thresholds, threshold saturation, minimum distance
- **Practical Considerations**
  - ➡ Window decoding, performance, latency, and complexity comparisons to LDPC block codes, rate-compatibility, implementation aspects

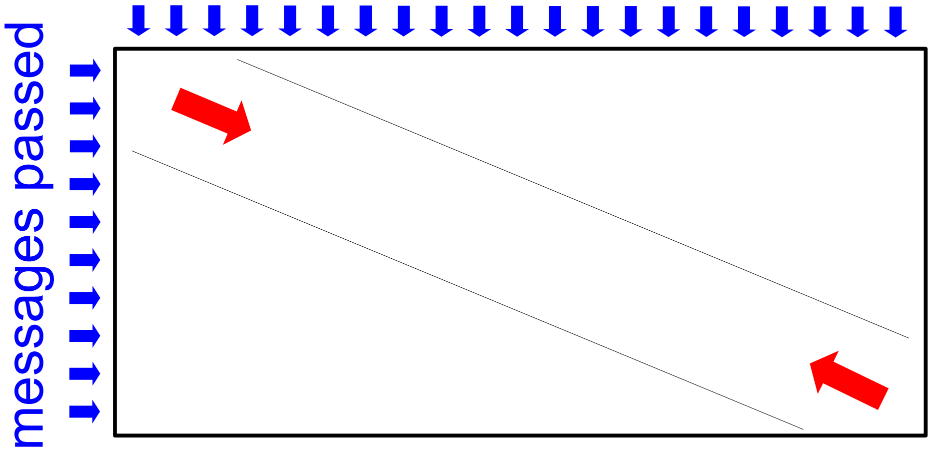
Block Decoding of SC-LDPC Codes

- SC-LDPC codes can be decoded with standard iterative decoding schedules.



Block Decoding of SC-LDPC Codes

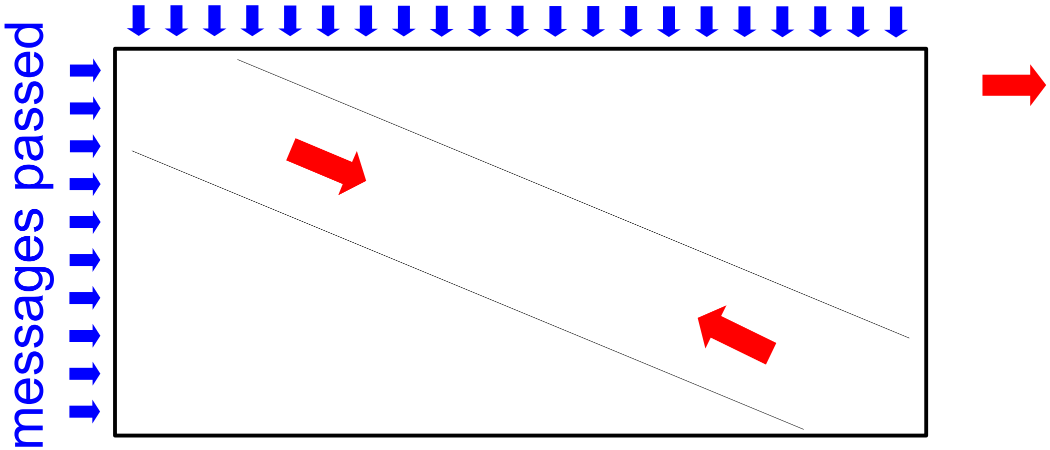
- SC-LDPC codes can be decoded with standard iterative decoding schedules.



Reliable messages from the ends propagate through the graph toward the center as iterations proceed.

Block Decoding of SC-LDPC Codes

- SC-LDPC codes can be decoded with standard iterative decoding schedules.

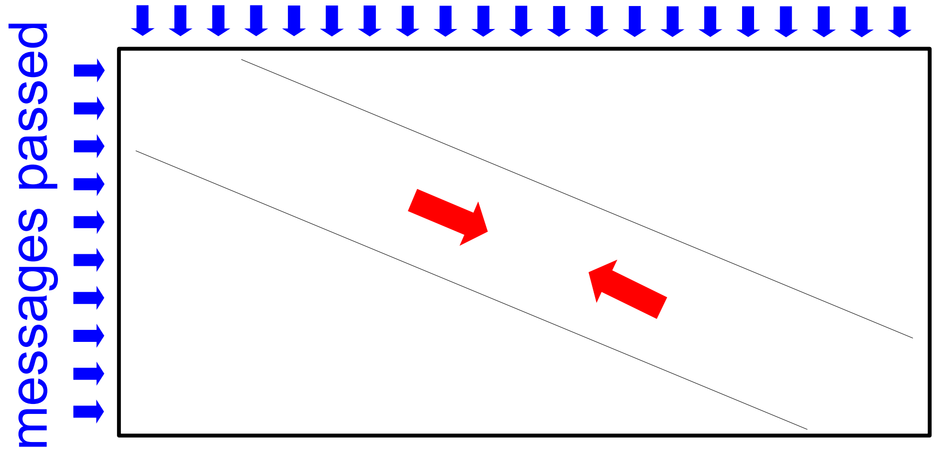


Reliable messages from the ends propagate through the graph toward the center as iterations proceed.

Block Decoding of SC-LDPC Codes

UNIVERSITY OF NOTRE DAME

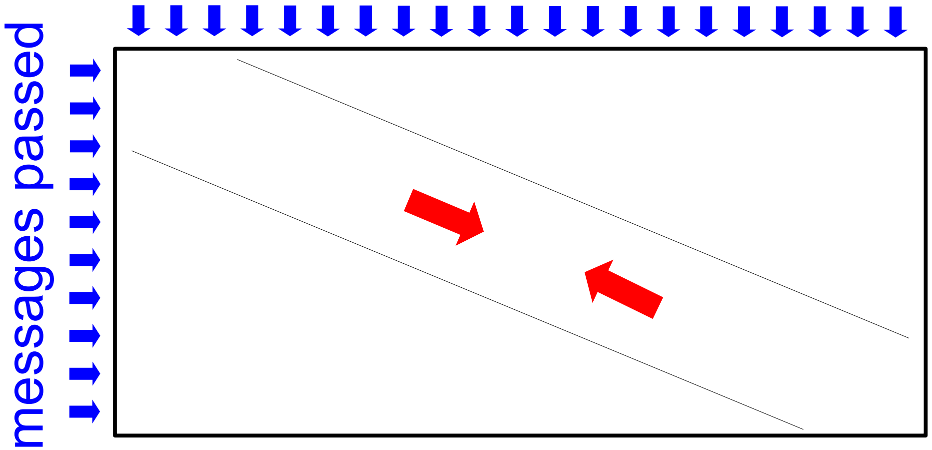
- SC-LDPC codes can be decoded with standard iterative decoding schedules.



Reliable messages from the ends propagate through the graph toward the center as iterations proceed.

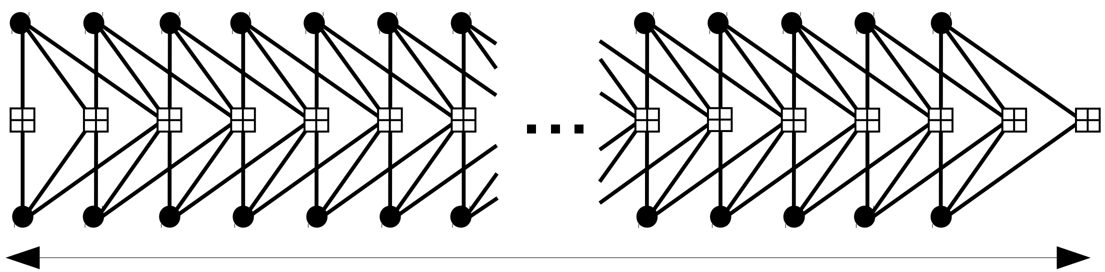
Block Decoding of SC-LDPC Codes

- SC-LDPC codes can be decoded with standard iterative decoding schedules.



➡ Reliable messages from the ends propagate through the graph toward the center as iterations proceed.

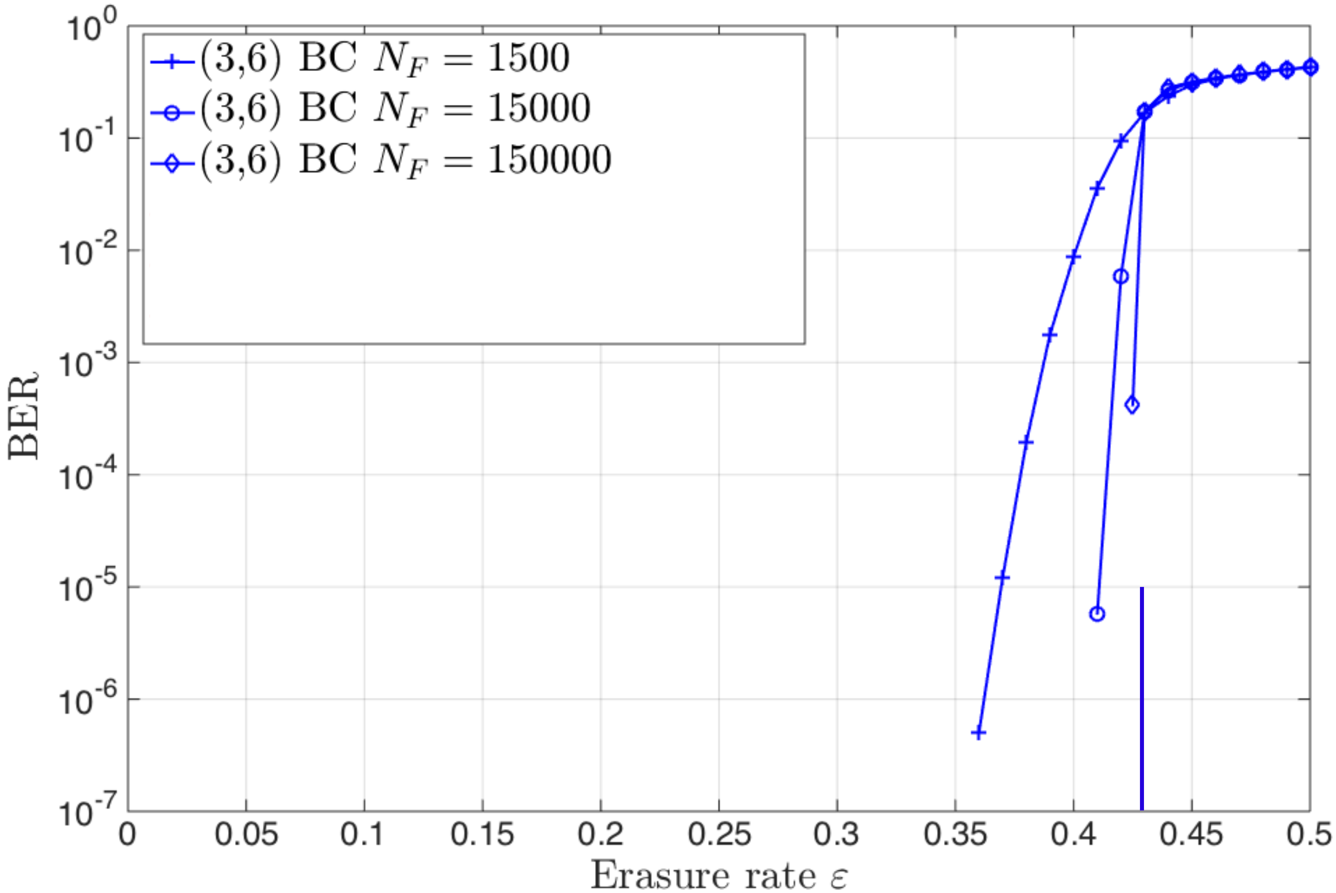
- The **frame error rate (FER)** of a terminated graph can be analyzed



➡ The **FER depends on L** (FER $\xrightarrow[L\to\infty]{} 1$)

Flooding Decoding Scaling (1) – BEC

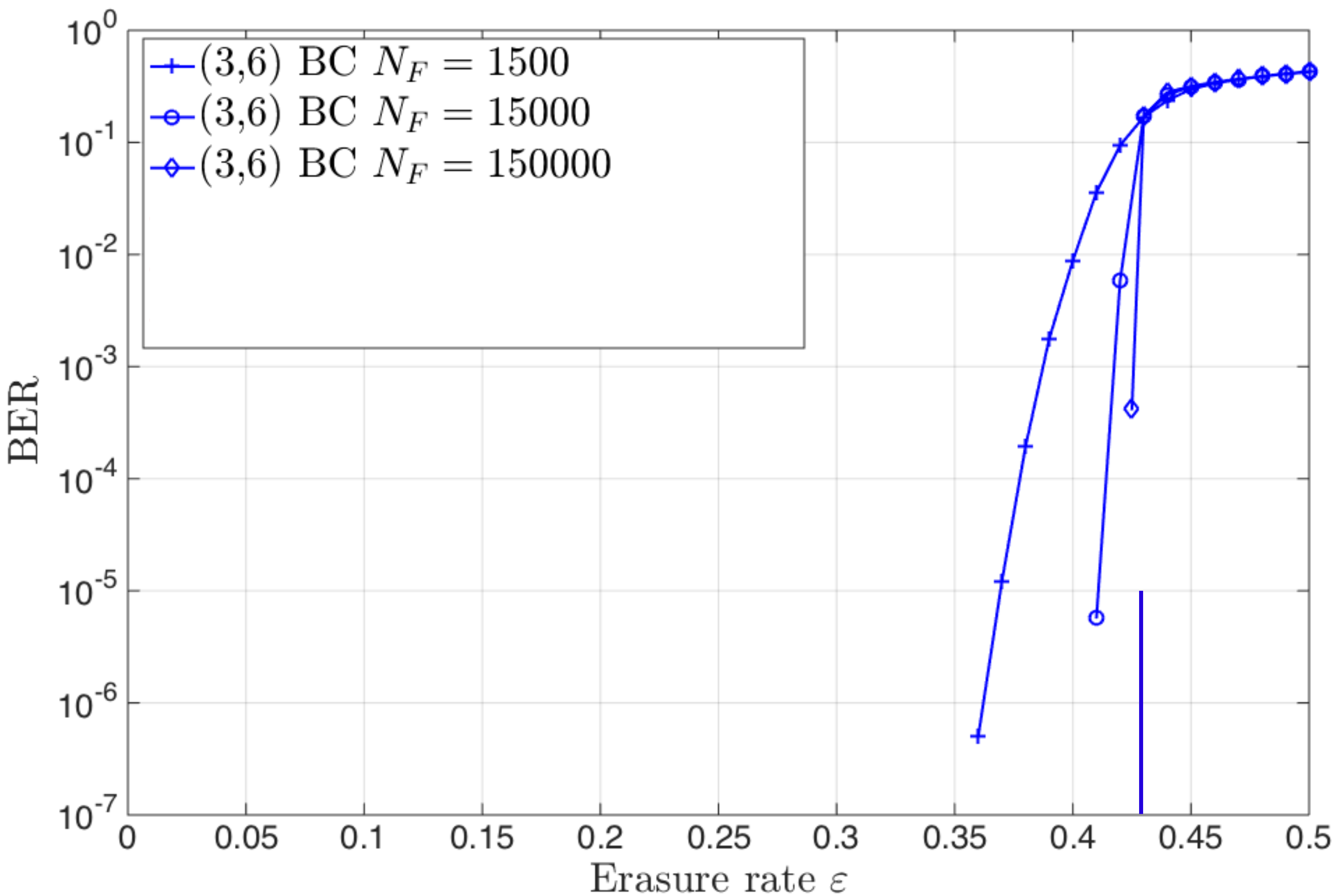
- Consider LDPC-BCs and SC-LDPC codes with increasing frame length N_F



9

Flooding Decoding Scaling (1) – BEC

- Consider LDPC-BCs and SC-LDPC codes with increasing frame length N_F

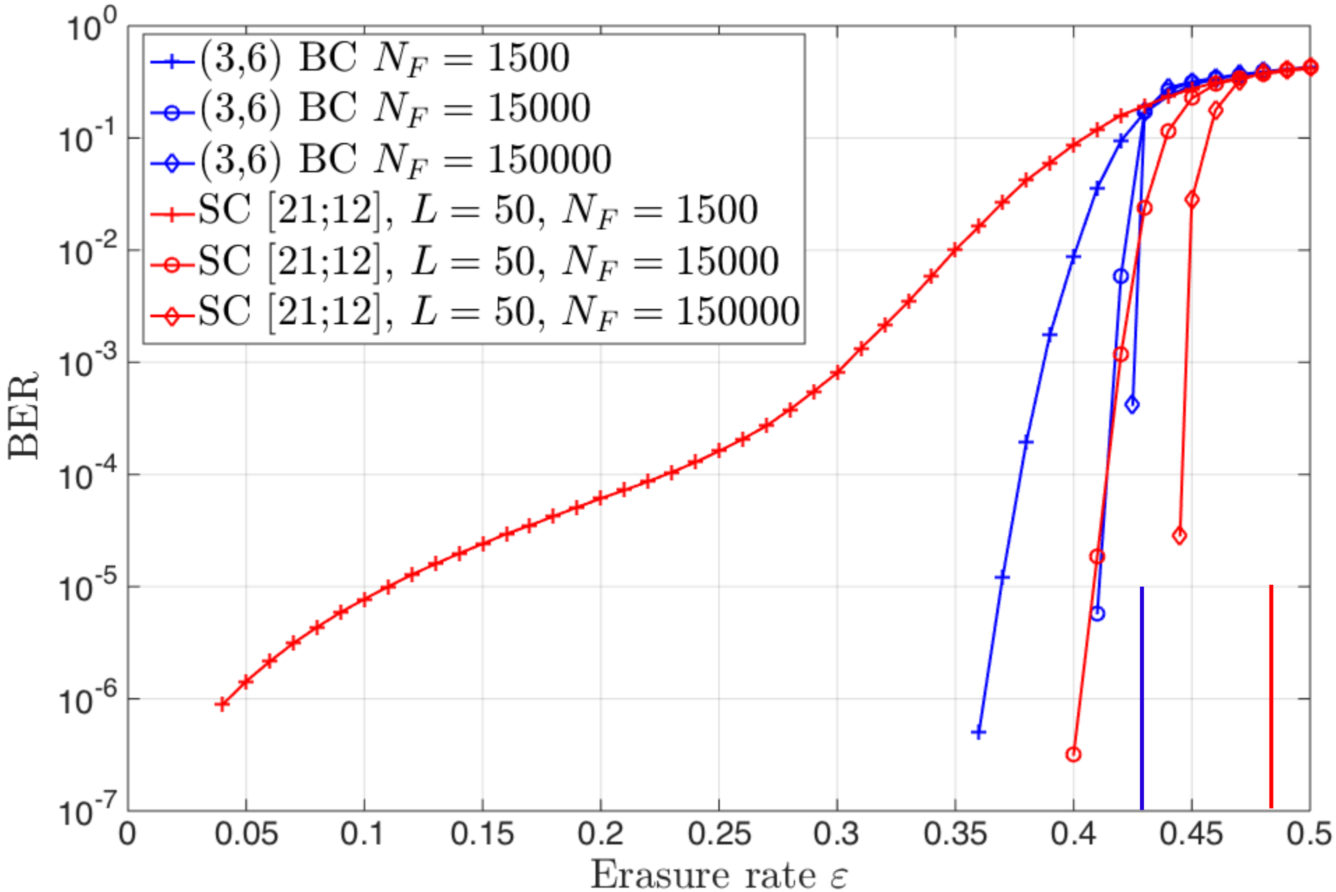


- As N_F increases the LDPC-BC performance approaches $\varepsilon^* = 0.429$

9

Flooding Decoding Scaling (1) – BEC

- Consider LDPC-BCs and SC-LDPC codes with increasing frame length N_F



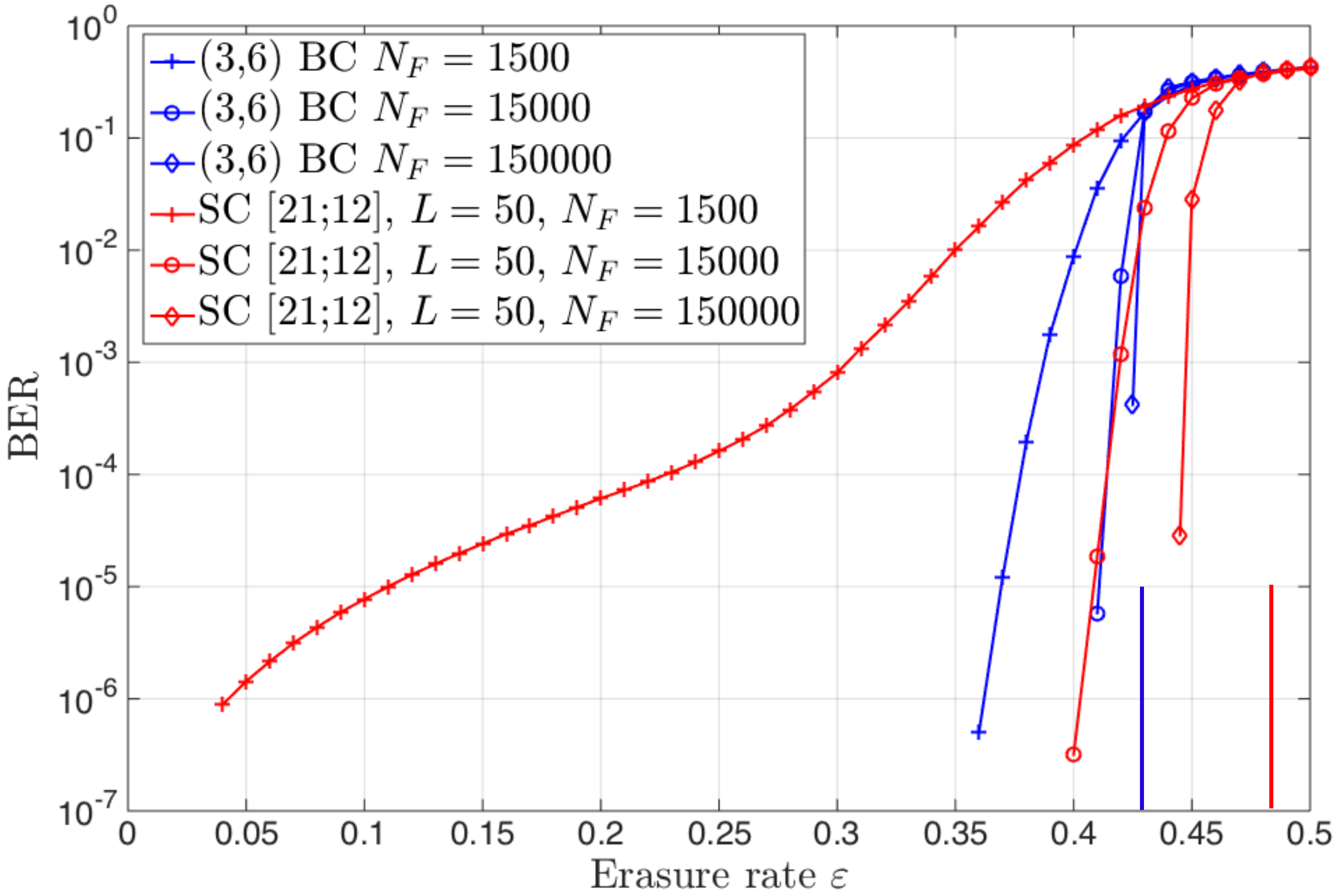
- As N_F increases the LDPC-BC performance approaches

$\varepsilon^* = 0.429$

9

Flooding Decoding Scaling (1) – BEC

- Consider LDPC-BCs and SC-LDPC codes with increasing frame length N_F



- As N_F increases the LDPC-BC performance approaches

  $\varepsilon^* = 0.429$

- As $N_F = 2LM$ increases (fixed L and increasing M = 15,150,1500), the SC-LDPC code performance approaches

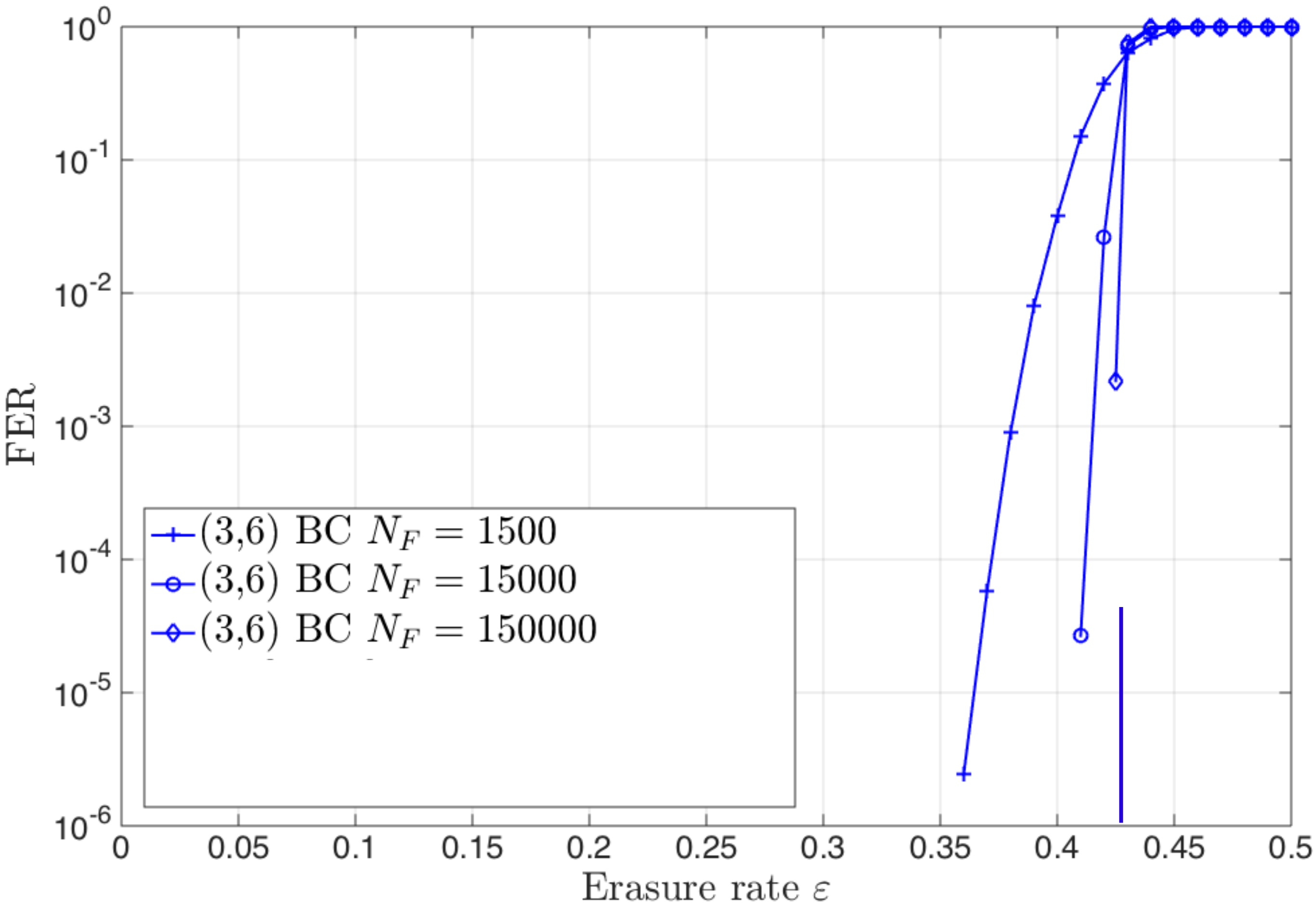
  $\varepsilon^* = 0.488$, outperforming the LDPC-BC

9

Flooding Decoding Scaling (2) – BEC

UNIVERSITY OF NOTRE DAME

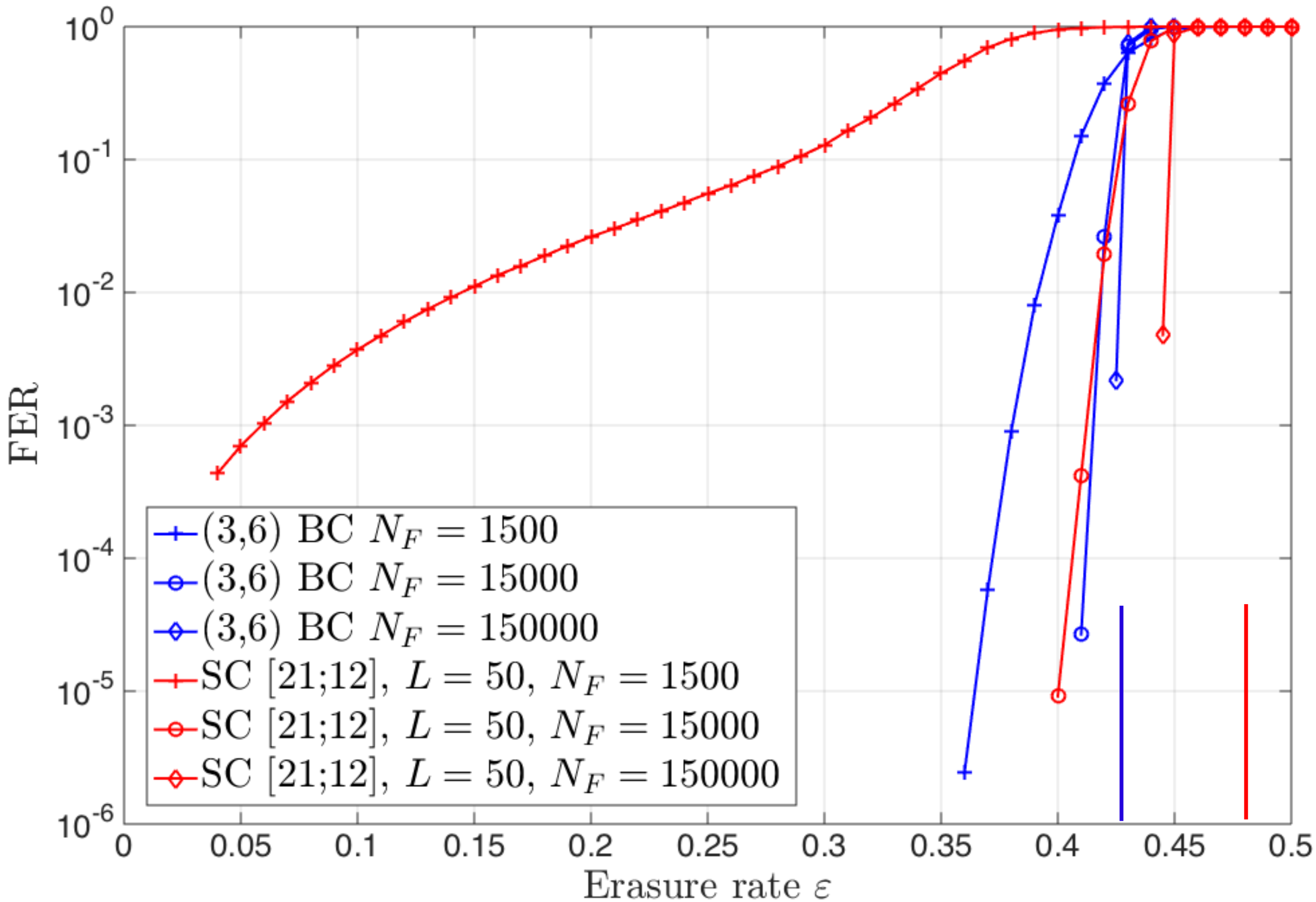
- Consider LDPC-BCs and SC-LDPC codes with increasing frame length N_F



- We now consider the FER performance

Flooding Decoding Scaling (2) – BEC

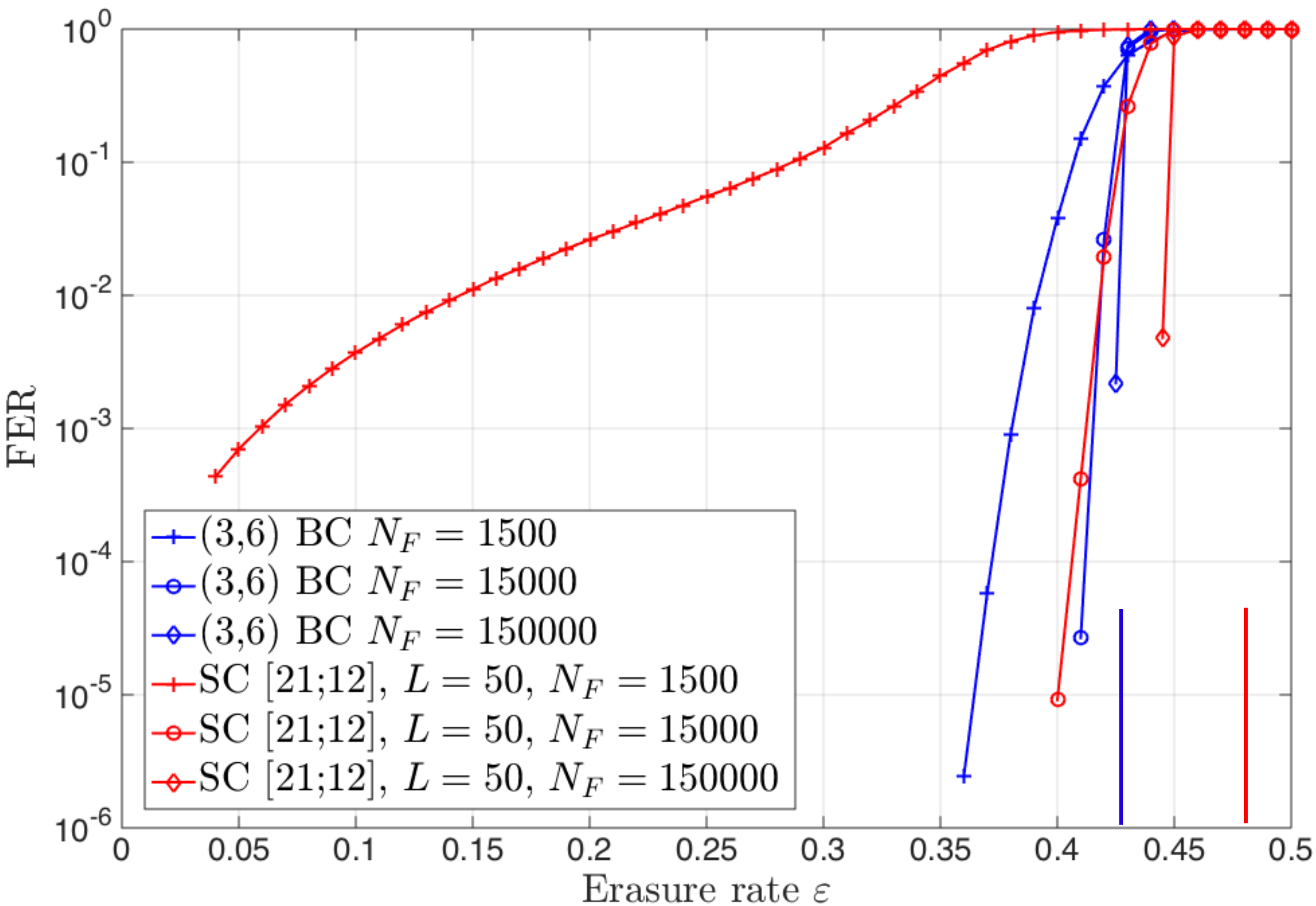
- Consider LDPC-BCs and SC-LDPC codes with increasing frame length N_F



- We now consider the FER performance

Flooding Decoding Scaling (2) – BEC

- Consider LDPC-BCs and SC-LDPC codes with increasing frame length N_F

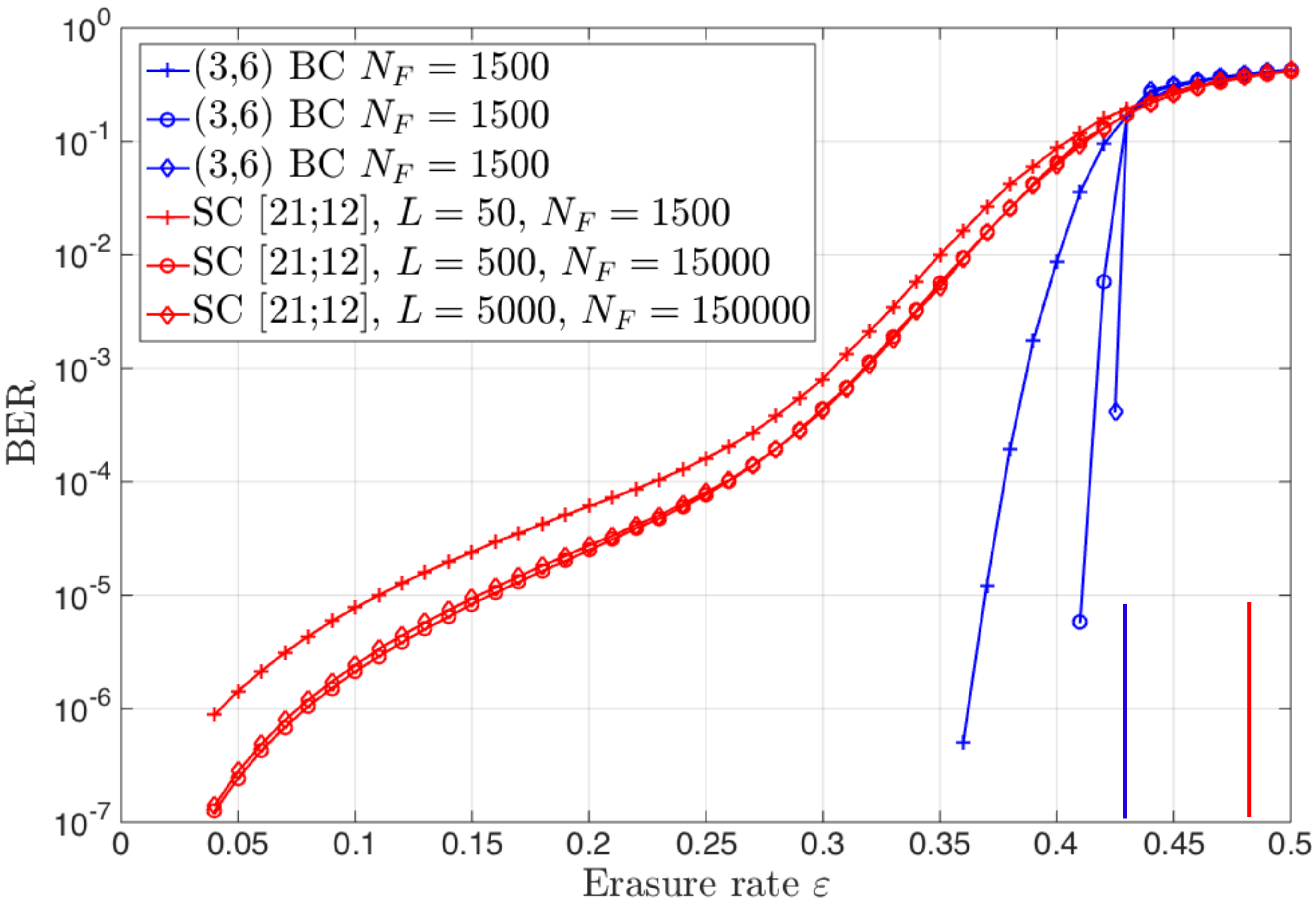


- We now consider the FER performance
- The FER curves display similar behavior for fixed L and increasing M

➡ The SC-LDPC codes outperform LDPC-BCs for large N_F

Flooding Decoding Scaling (3) – BEC

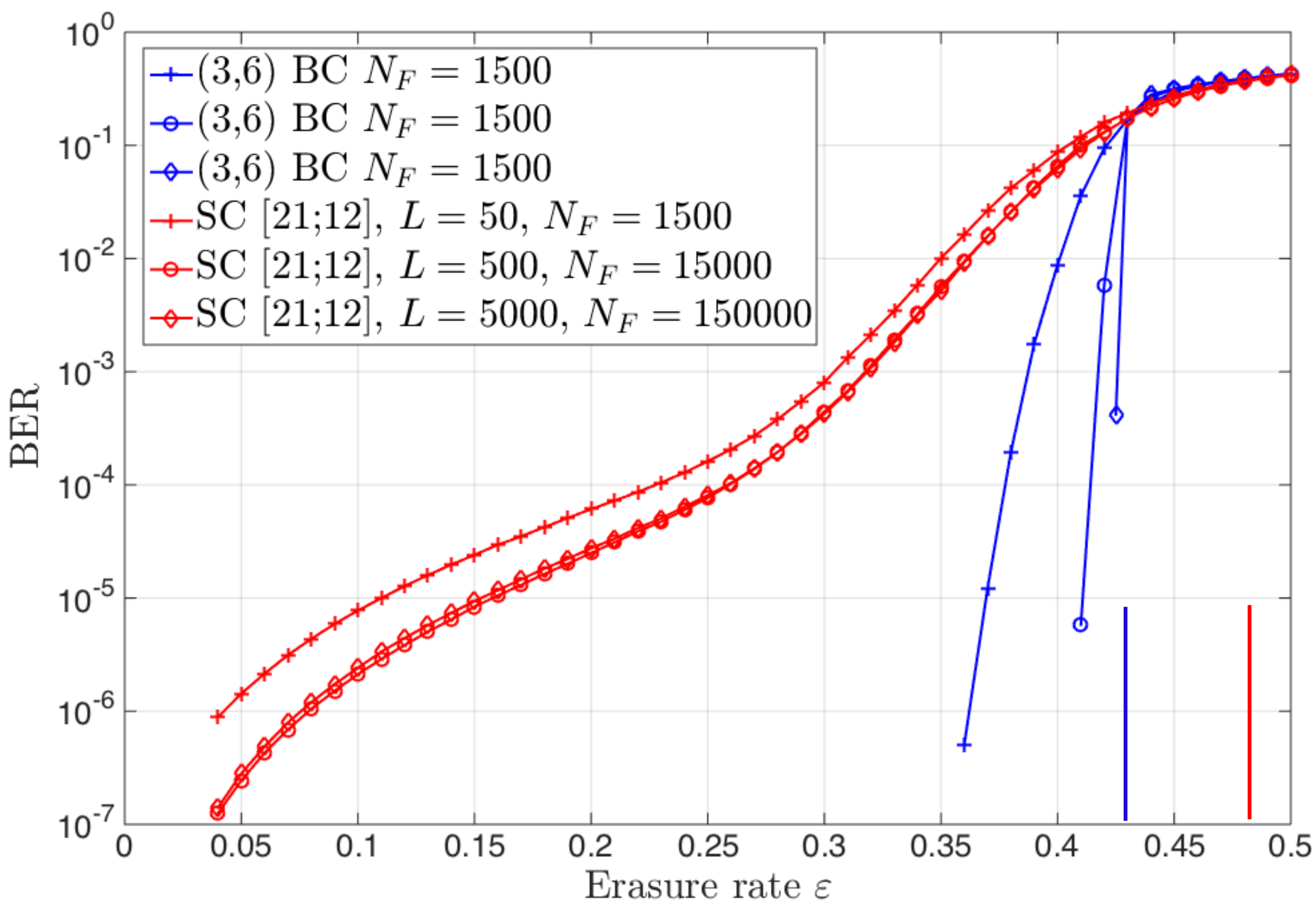
- Consider LDPC-BCs and SC-LDPC codes with increasing frame length N_F



- We now consider increasing the frame length $N_F = 2LM$ with fixed M = 15 and increasing L

Flooding Decoding Scaling (3) – BEC

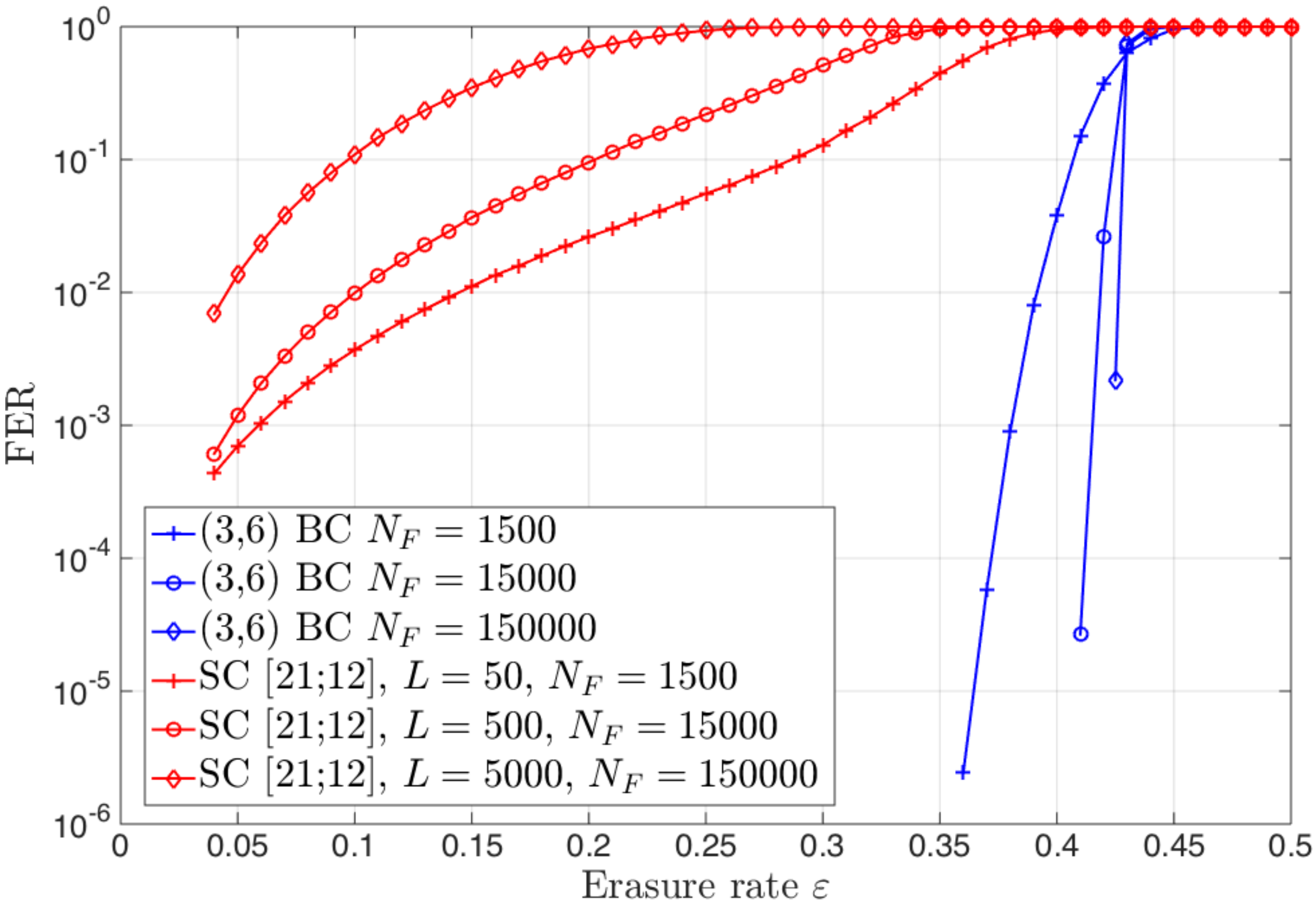
- Consider LDPC-BCs and SC-LDPC codes with increasing frame length N_F



- We now consider increasing the frame length $N_F = 2LM$ with fixed M = 15 and increasing L
- Note that the BER performance is poor for the SC-LDPC codes (small M) and does not improve substantially with increasing L

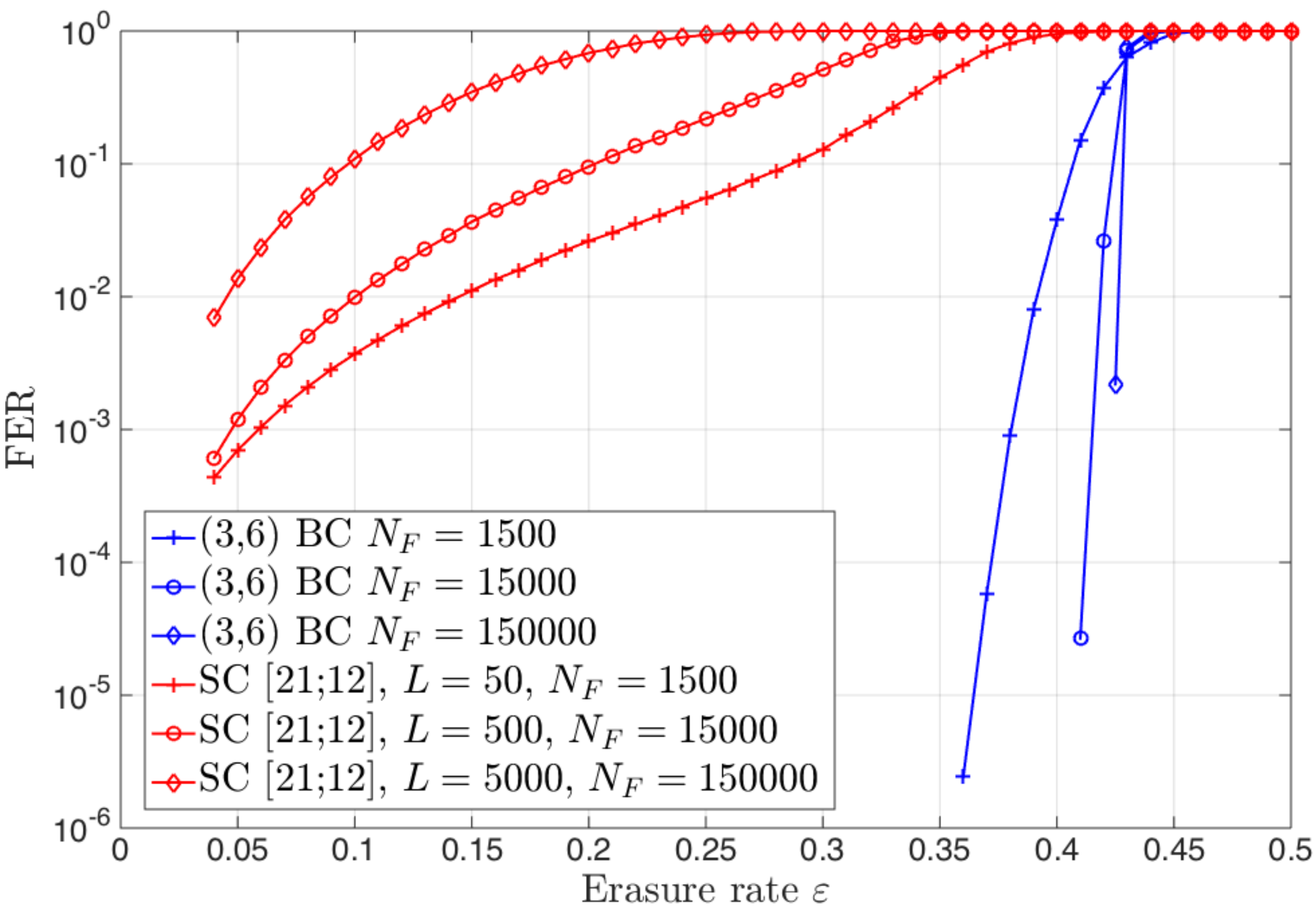
Flooding Decoding Scaling (4) – BEC

- Consider LDPC-BCs and SC-LDPC codes with increasing frame length N_F



Flooding Decoding Scaling (4) – BEC

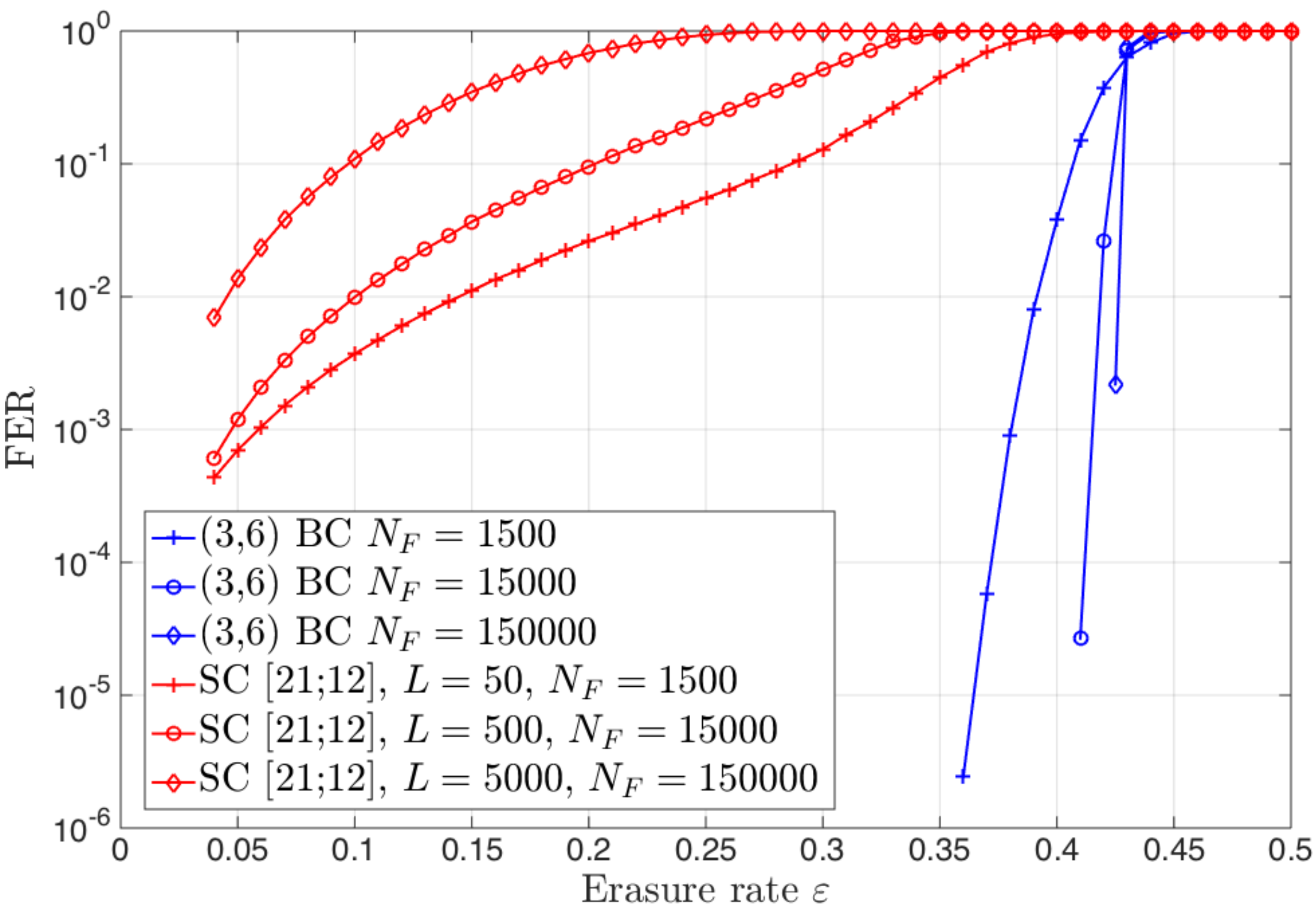
- Consider LDPC-BCs and SC-LDPC codes with increasing frame length N_F



- The FER performance is even more extreme since the **FER depends on L** (FER $\xrightarrow[L\to\infty]{} 1$)

Flooding Decoding Scaling (4) – BEC

- Consider LDPC-BCs and SC-LDPC codes with increasing frame length N_F



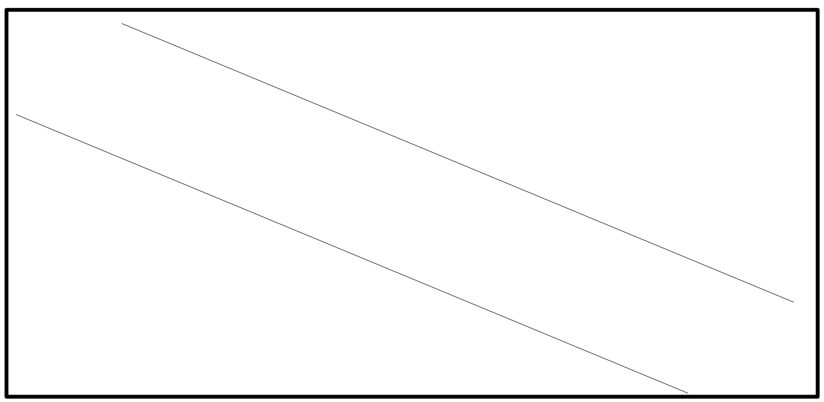
- The FER performance is even more extreme since the **FER depends on L**

  (FER $\xrightarrow[L\to\infty]{} 1$)

- Correspondingly, we note that the larger L codes perform worse (the order is reversed)

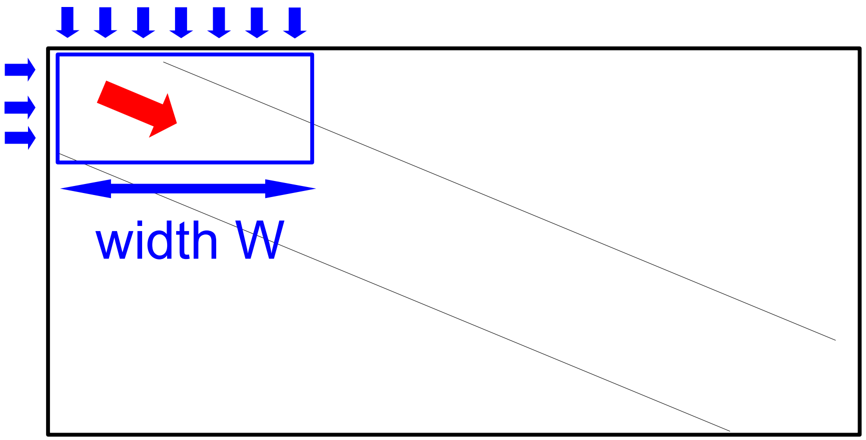
Window Decoding of SC-LDPC Codes

- The **highly localized (convolutional) structure** is well-suited for efficient decoding schedules that reduce memory and latency requirements.



Window Decoding of SC-LDPC Codes

- The **highly localized (convolutional) structure** is well-suited for efficient decoding schedules that reduce memory and latency requirements.

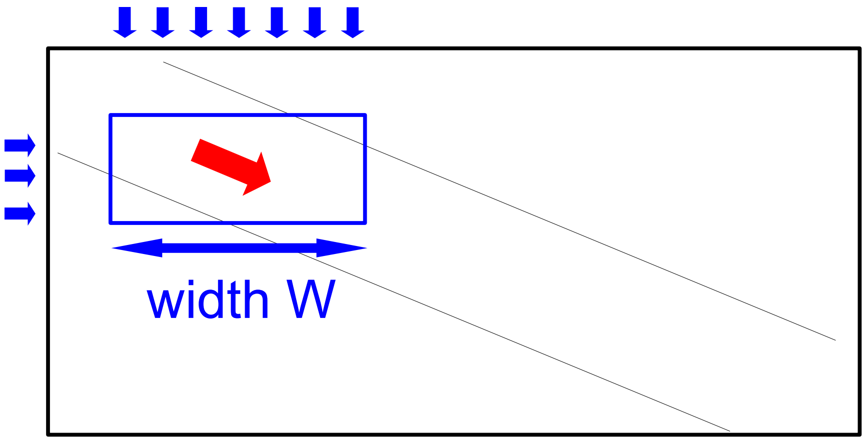


➡ **Sliding window decoding** (WD) updates nodes only within a localized window and then the window shifts across the graph [Lentmaier et al '10, Iyengar et al '12].

Window Decoding of SC-LDPC Codes

UNIVERSITY OF NOTRE DAME

- The **highly localized (convolutional) structure** is well-suited for efficient decoding schedules that reduce memory and latency requirements.

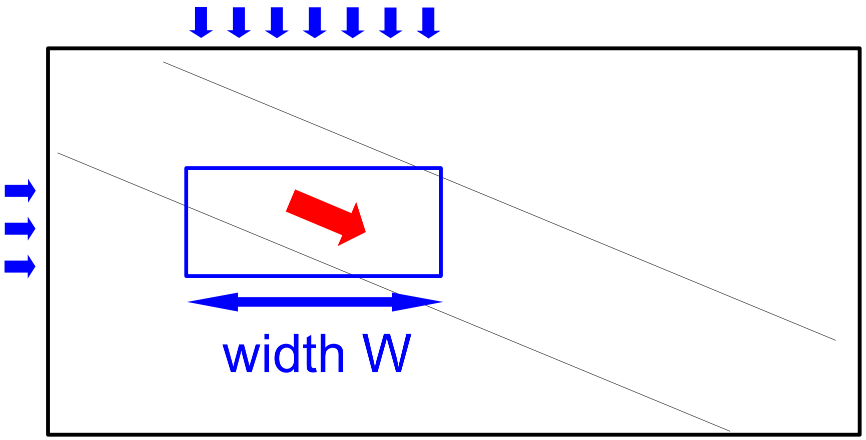


➡ **Sliding window decoding** (WD) updates nodes only within a localized window and then the window shifts across the graph [Lentmaier et al '10, Iyengar et al '12].

Window Decoding of SC-LDPC Codes

UNIVERSITY OF NOTRE DAME

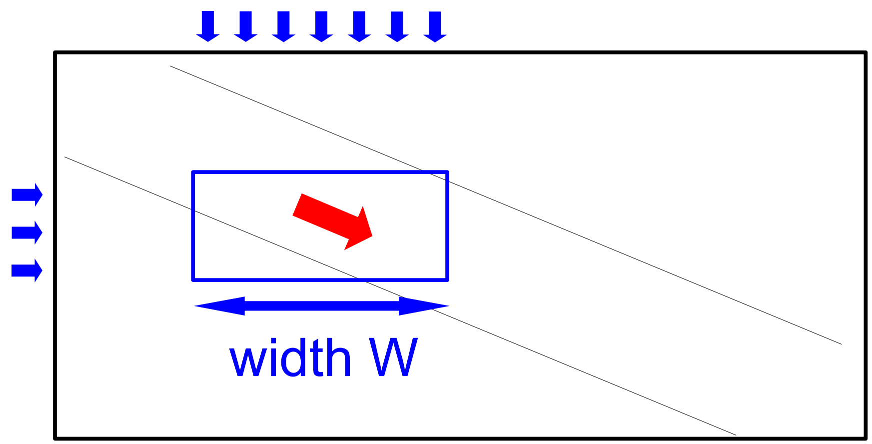
- The **highly localized (convolutional) structure** is well-suited for efficient decoding schedules that reduce memory and latency requirements.



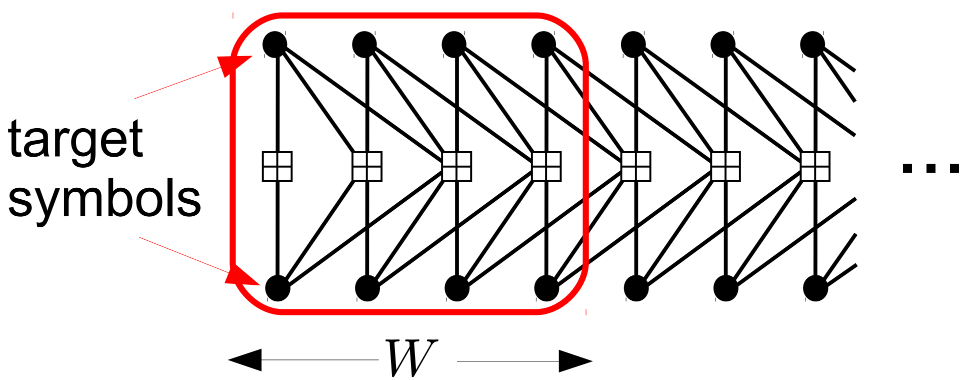
➡ **Sliding window decoding** (WD) updates nodes only within a localized window and then the window shifts across the graph [Lentmaier et al '10, Iyengar et al '12].

Window Decoding of SC-LDPC Codes

- The **highly localized (convolutional) structure** is well-suited for efficient decoding schedules that reduce memory and latency requirements.



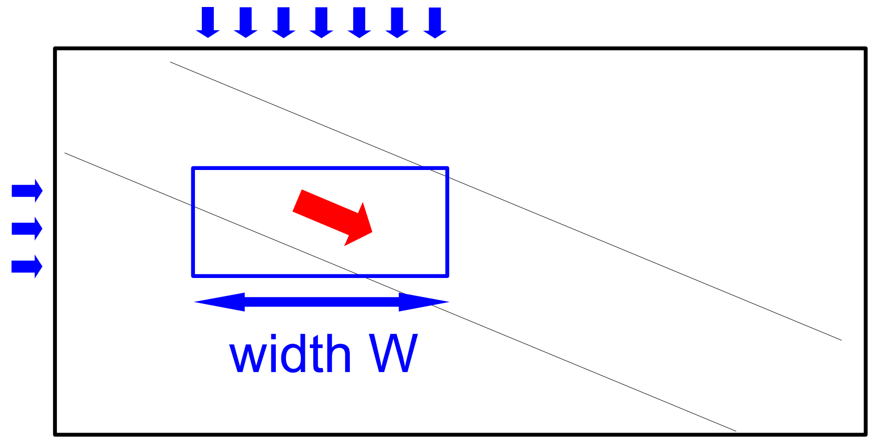
➡ **Sliding window decoding** (WD) updates nodes only within a localized window and then the window shifts across the graph [Lentmaier et al '10, Iyengar et al '12].



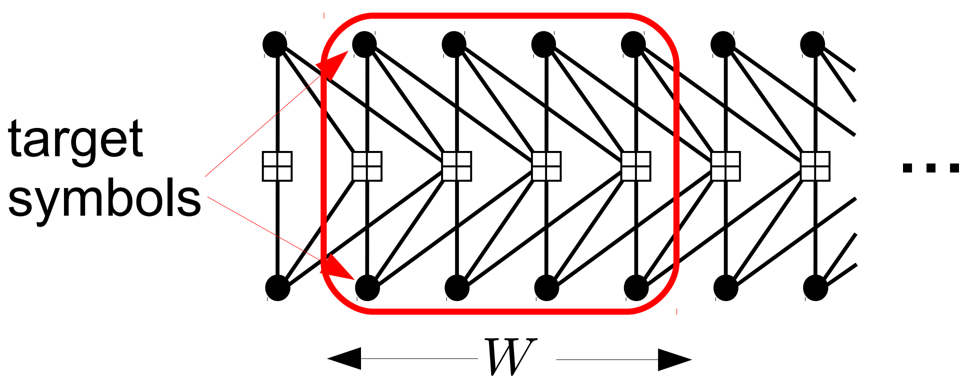
➡ One block of cM **target symbols** is decoded in each window position

Window Decoding of SC-LDPC Codes

- The **highly localized (convolutional) structure** is well-suited for efficient decoding schedules that reduce memory and latency requirements.



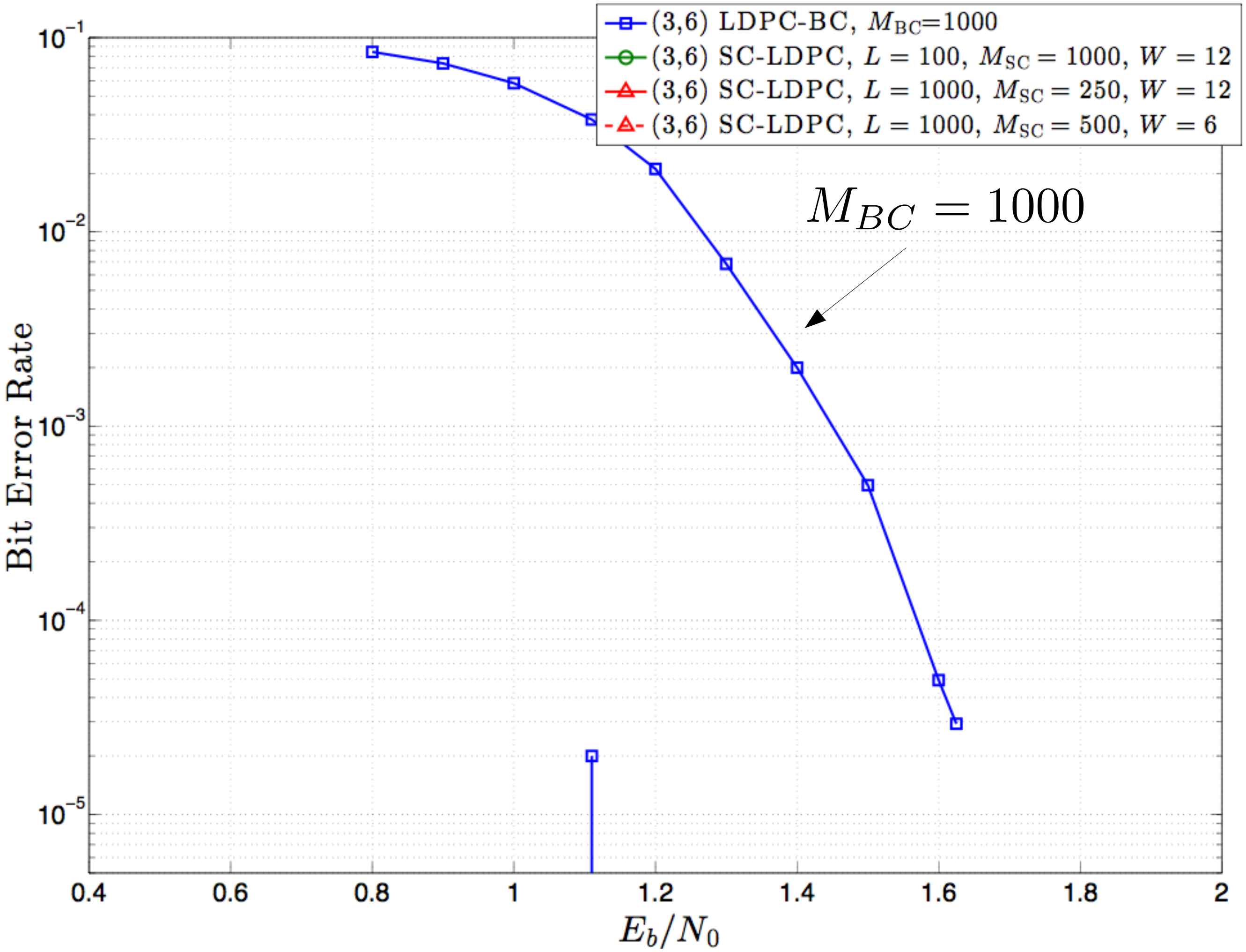
➡ **Sliding window decoding** (WD) updates nodes only within a localized window and then the window shifts across the graph [Lentmaier et al '10, Iyengar et al '12].



➡ One block of cM **target symbols** is decoded in each window position

➡ The window then shifts to the right

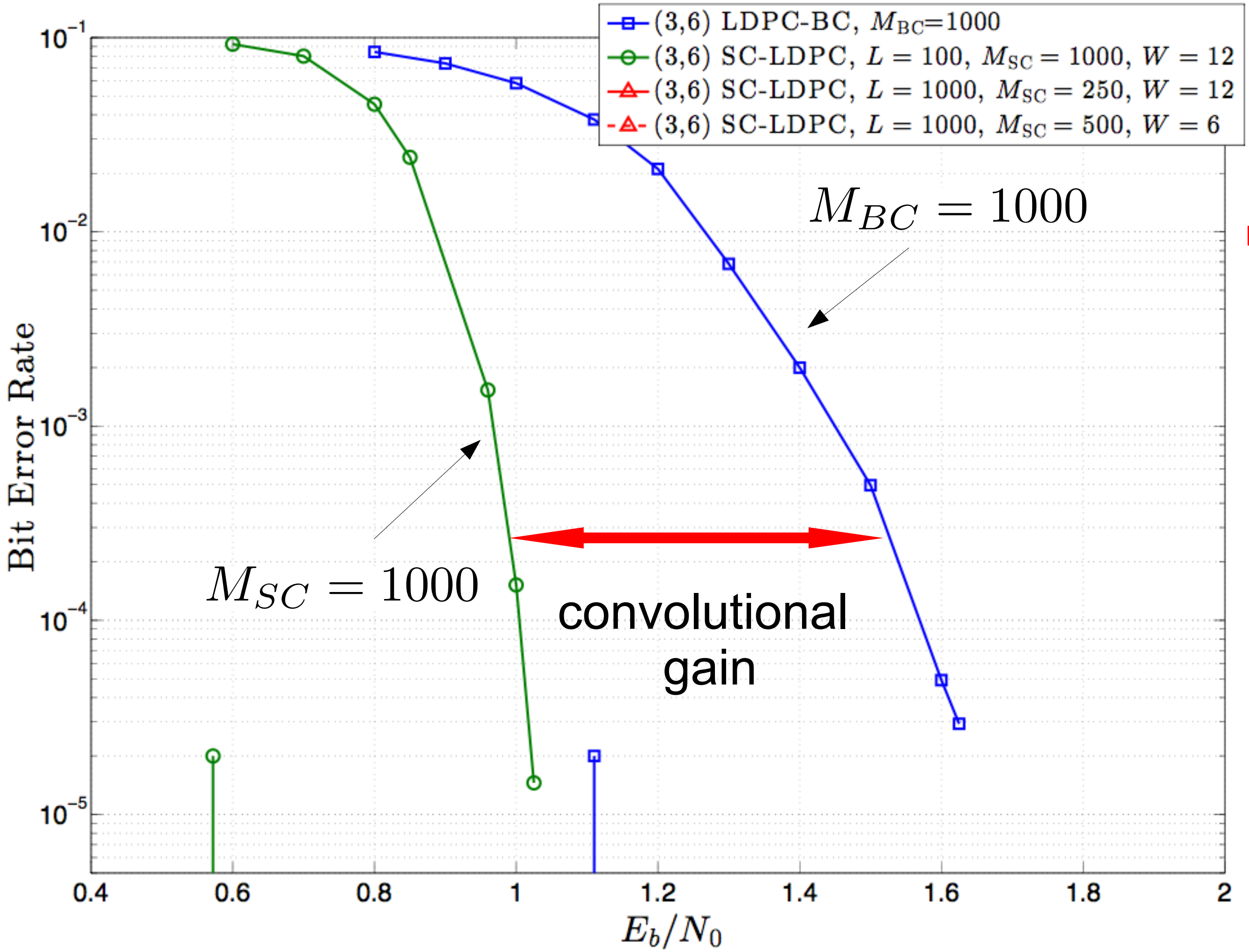
Window Decoding Performance



Latencies:
LDPC: $6M_{BC}$
SC-LDPC: $2M_{SC}W$

[LPF11] M. Lentmaier, M. M. Prenda, and G. Fettweis, "Efficient Message Passing Scheduling for Terminated LDPC Convolutional Codes", *Proc. IEEE ISIT*, St. Petersburg, Russia, July 2011.

Window Decoding Performance

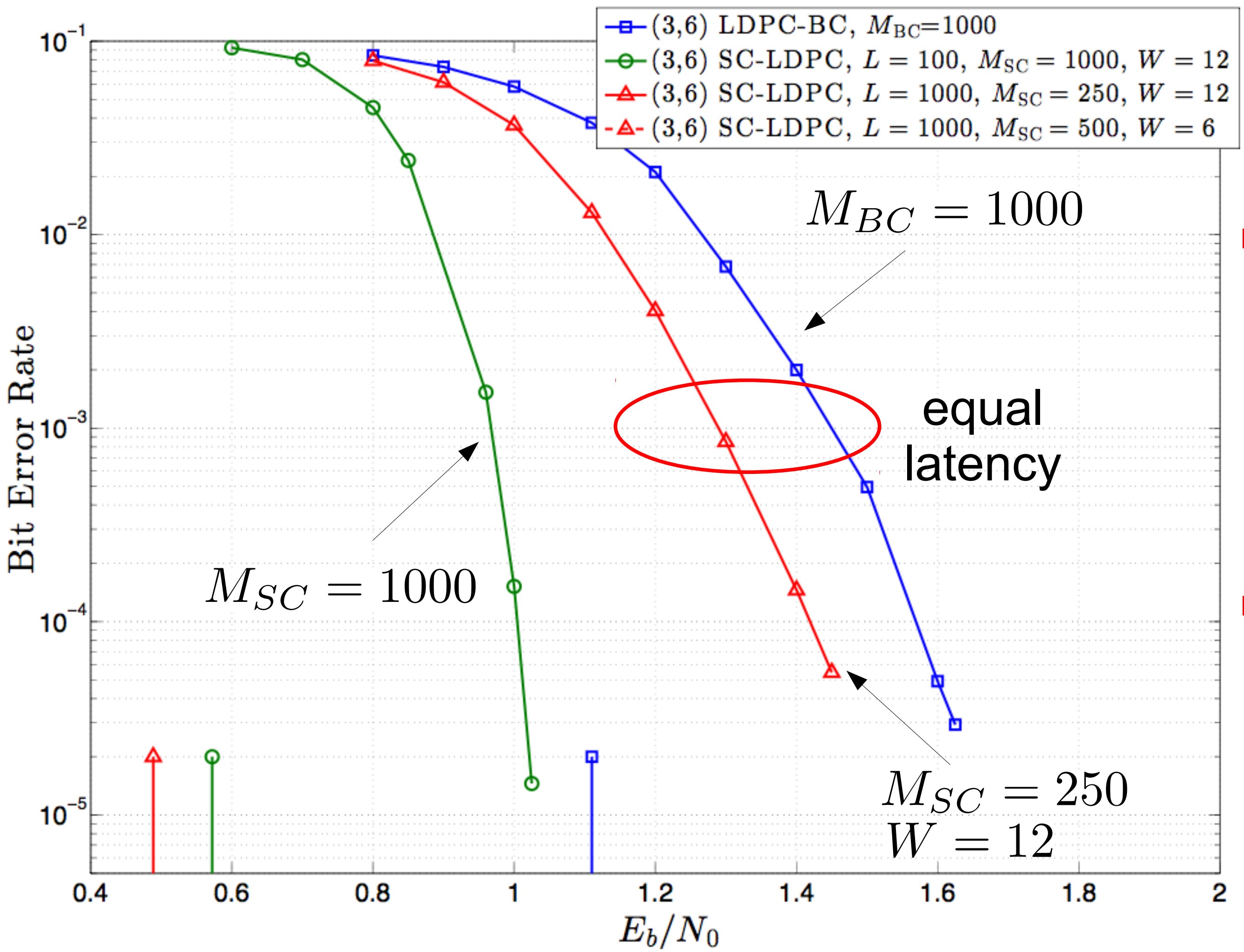


Latencies:
LDPC: $6M_{BC}$
SC-LDPC: $2M_{SC}W$

- For **equal lifting factors**, SC-LDPC codes display a large **convolutional gain** at the cost of increased latency.

[LPF11] M. Lentmaier, M. M. Prenda, and G. Fettweis, "Efficient Message Passing Scheduling for Terminated LDPC Convolutional Codes", *Proc. IEEE ISIT*, St. Petersburg, Russia, July 2011.

Window Decoding Performance

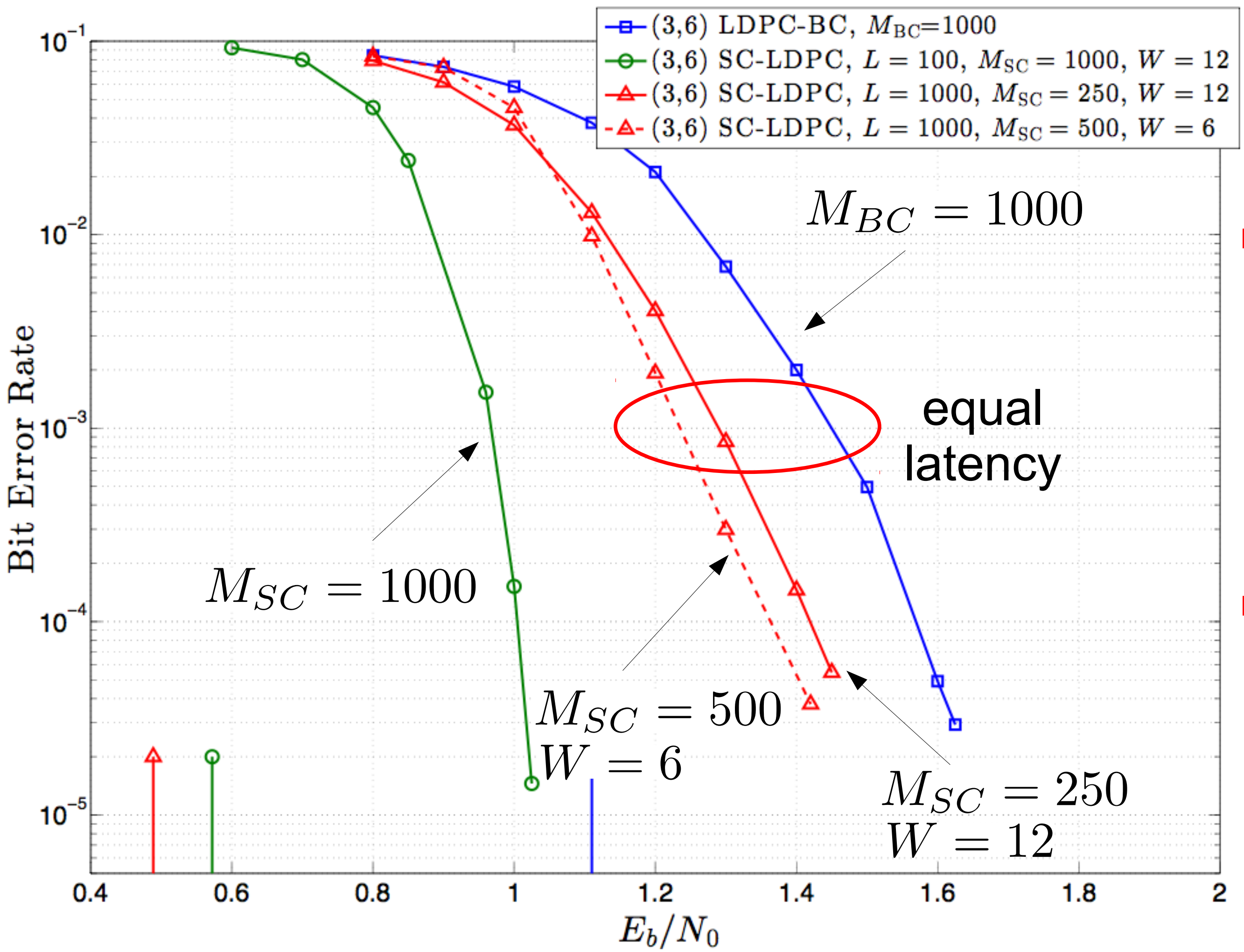


Latencies:
LDPC: $6M_{BC}$
SC-LDPC: $2M_{SC}W$

- For **equal lifting factors**, SC-LDPC codes display a large **convolutional gain** at the cost of increased latency.
- For **equal latency**, SC-LDPC codes still display a significant **performance gain**.

[LPF11] M. Lentmaier, M. M. Prenda, and G. Fettweis, "Efficient Message Passing Scheduling for Terminated LDPC Convolutional Codes", *Proc. IEEE ISIT*, St. Petersburg, Russia, July 2011.

Window Decoding Performance

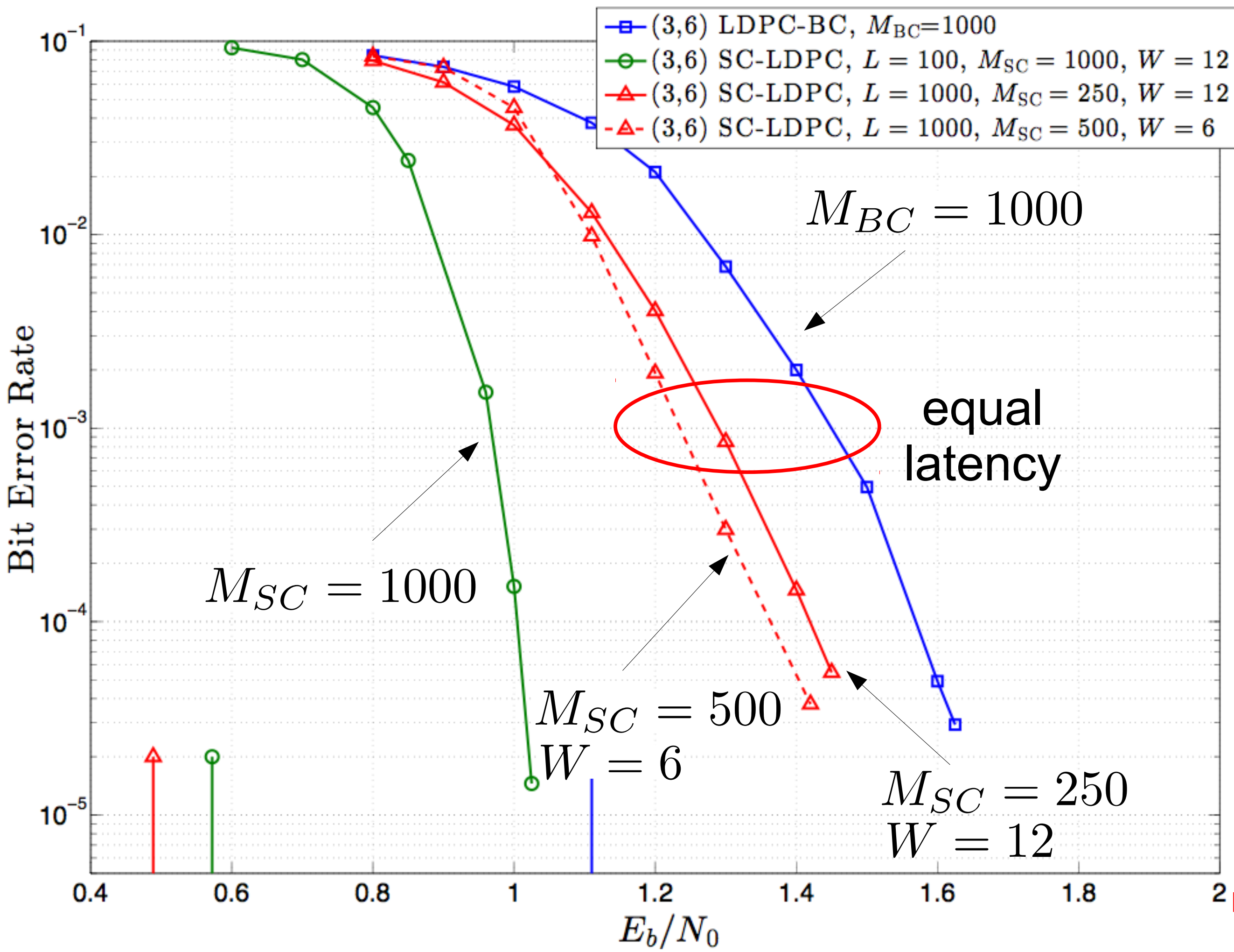


Latencies:
LDPC: $6M_{BC}$
SC-LDPC: $2M_{SC}W$

- For **equal lifting factors**, SC-LDPC codes display a large **convolutional gain** at the cost of increased latency.
- For **equal latency**, SC-LDPC codes still display a significant **performance gain**.

[LPF11] M. Lentmaier, M. M. Prenda, and G. Fettweis, "Efficient Message Passing Scheduling for Terminated LDPC Convolutional Codes", *Proc. IEEE ISIT*, St. Petersburg, Russia, July 2011.

Window Decoding Performance



Latencies:
LDPC: $6M_{BC}$
SC-LDPC: $2M_{SC}W$

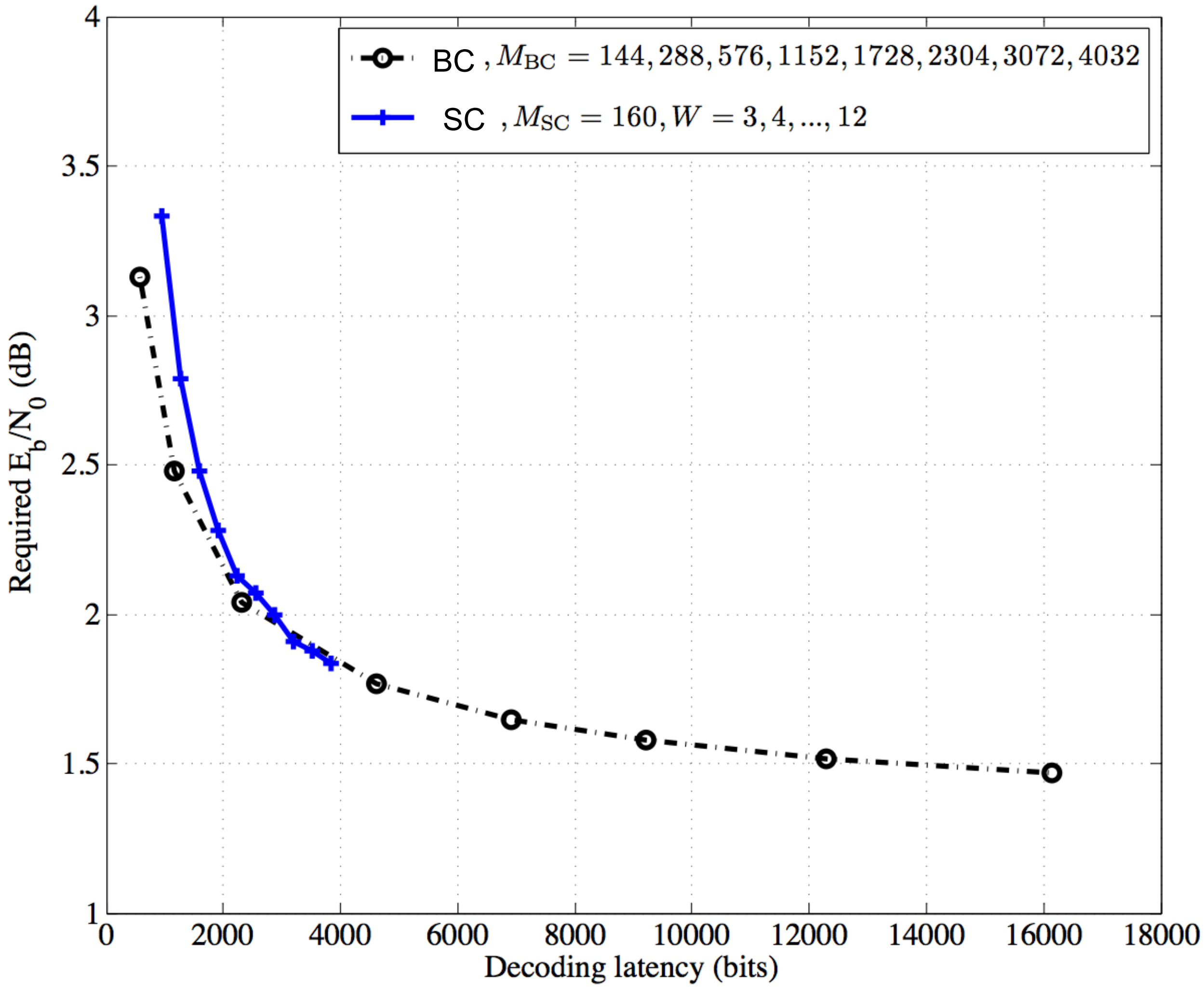
- For **equal lifting factors**, SC-LDPC codes display a large **convolutional gain** at the cost of increased latency.
- For **equal latency**, SC-LDPC codes still display a significant **performance gain**.
- Trade-off in M vs W

[LPF11] M. Lentmaier, M. M. Prenda, and G. Fettweis, "Efficient Message Passing Scheduling for Terminated LDPC Convolutional Codes", *Proc. IEEE ISIT*, St. Petersburg, Russia, July 2011.

Equal Latency Comparison for (3,6)-Regular LDPC Codes

UNIVERSITY OF NOTRE DAME

- Required E_b/N_0 to achieve a BER of 10^{-5} as a function of latency:

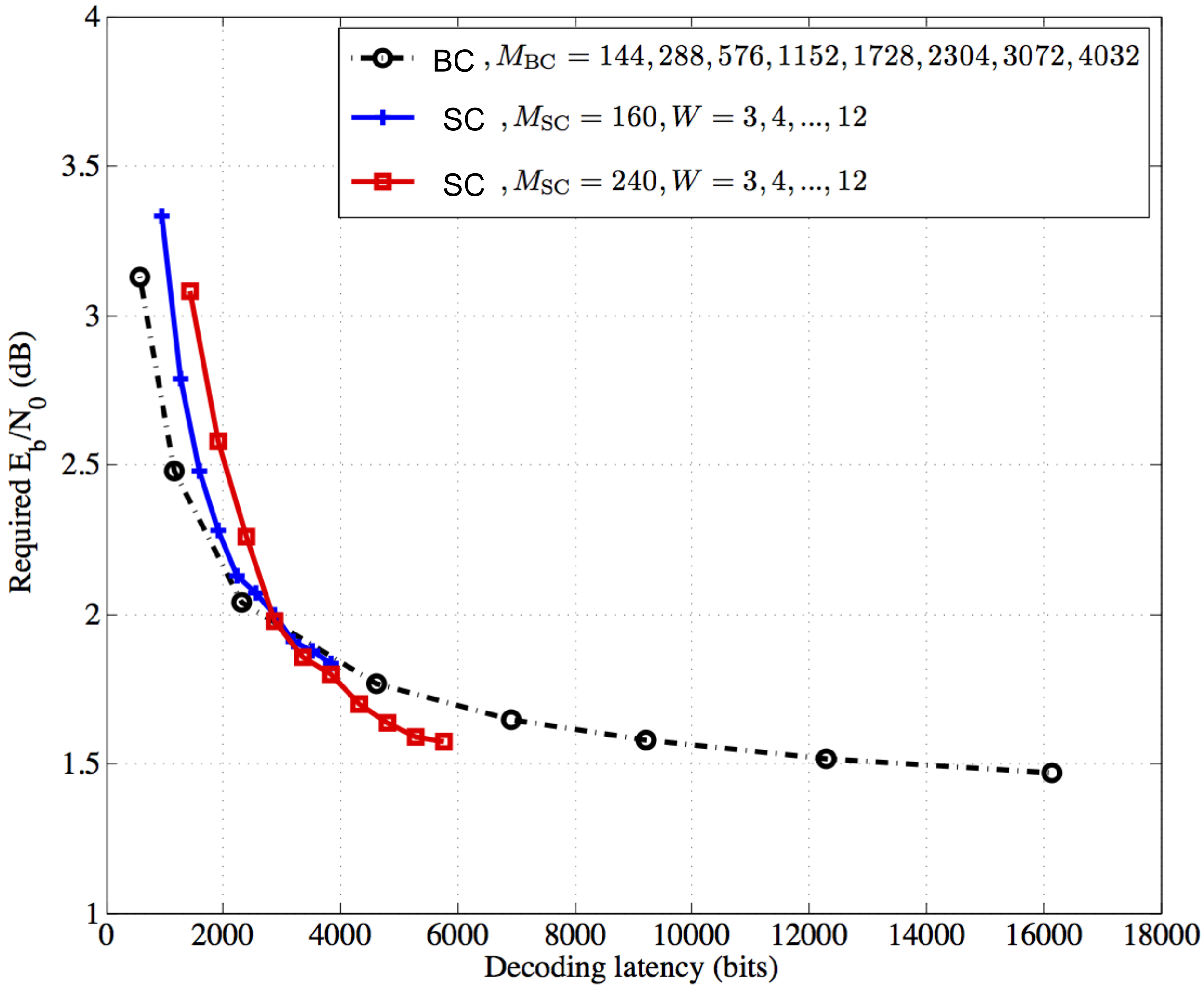


Latencies:
LDPC: $4M_{BC}$
SC-LDPC: $2M_{SC}W$

- ➡ decreases as W (and thus the latency) increases.
- ➡ does not decrease significantly beyond a certain W $(W \approx 10)$.

Equal Latency Comparison for (3,6)-Regular LDPC Codes

- Required E_b/N_0 to achieve a BER of 10^{-5} as a function of latency:



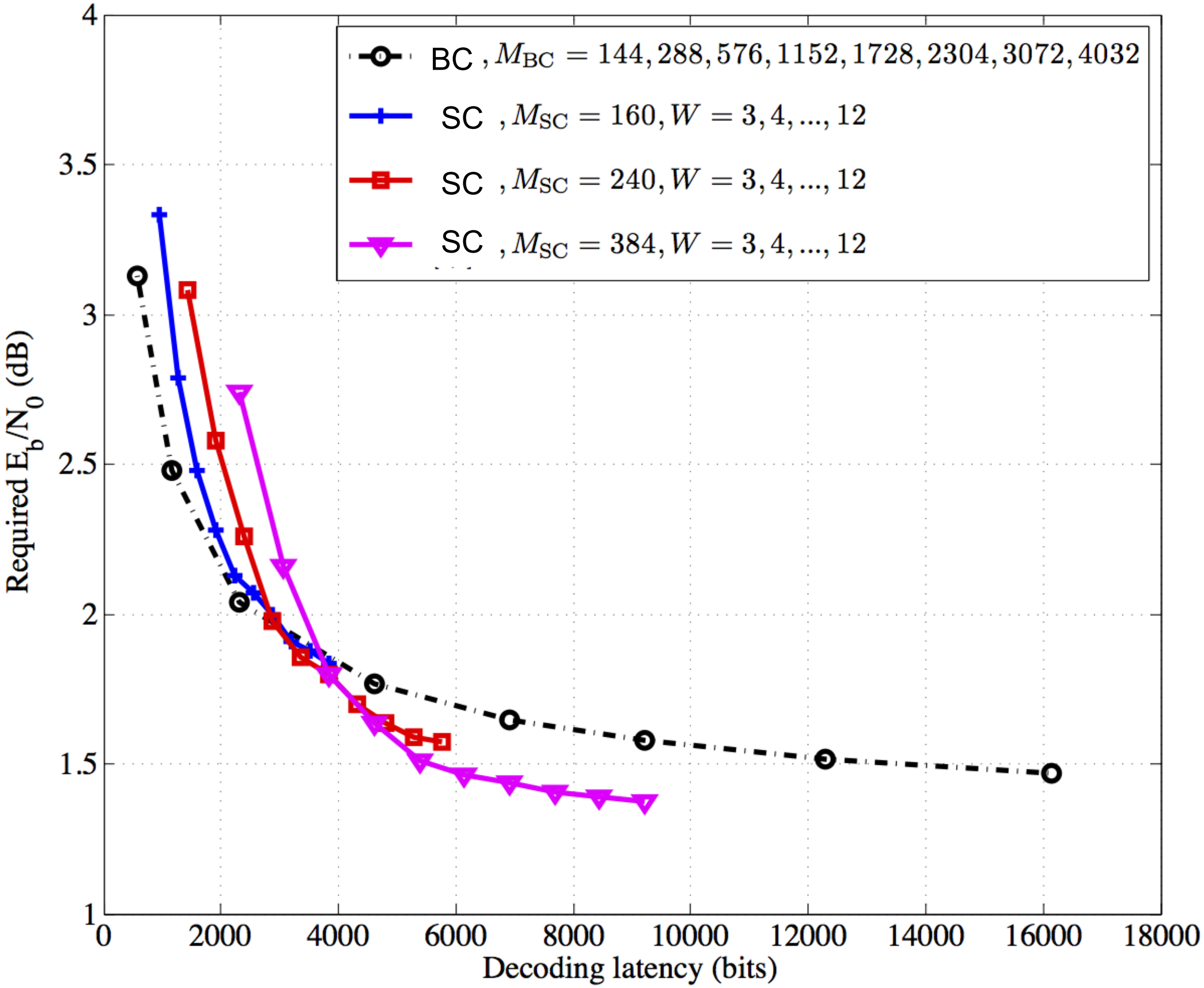
Latencies:
LDPC: $4M_{BC}$
SC-LDPC: $2M_{SC}W$

- ➡ decreases as W (and thus the latency) increases.
- ➡ does not decrease significantly beyond a certain W $(W \approx 10)$.

Equal Latency Comparison for (3,6)-Regular LDPC Codes

UNIVERSITY OF NOTRE DAME

- Required E_b/N_0 to achieve a BER of 10^{-5} as a function of latency:



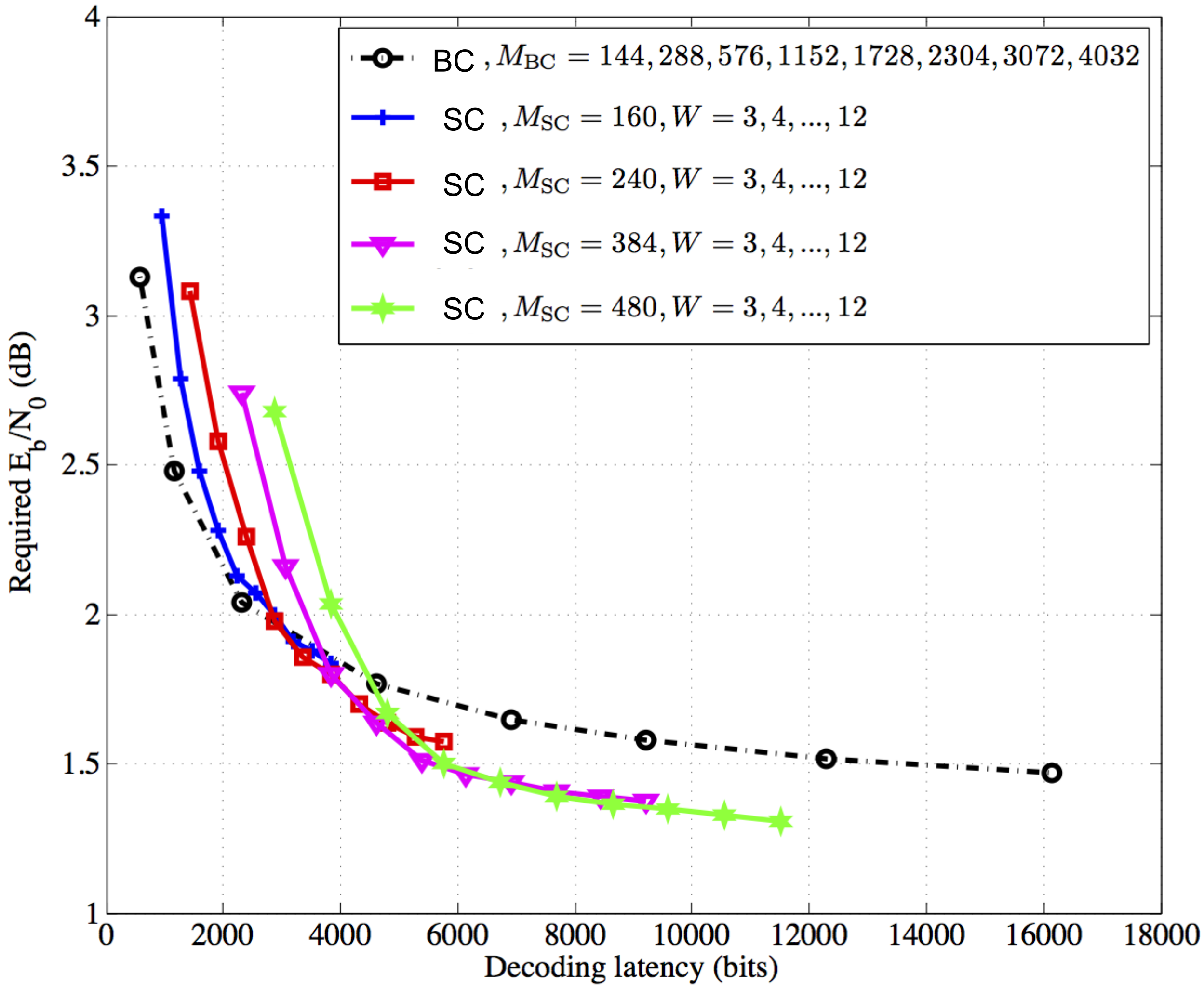
Latencies:
LDPC: $4M_{BC}$
SC-LDPC: $2M_{SC}W$

- ➡ decreases as W (and thus the latency) increases.
- ➡ does not decrease significantly beyond a certain W $(W \approx 10)$.

Equal Latency Comparison for (3,6)-Regular LDPC Codes

UNIVERSITY OF NOTRE DAME

- Required E_b/N_0 to achieve a BER of 10^{-5} as a function of latency:



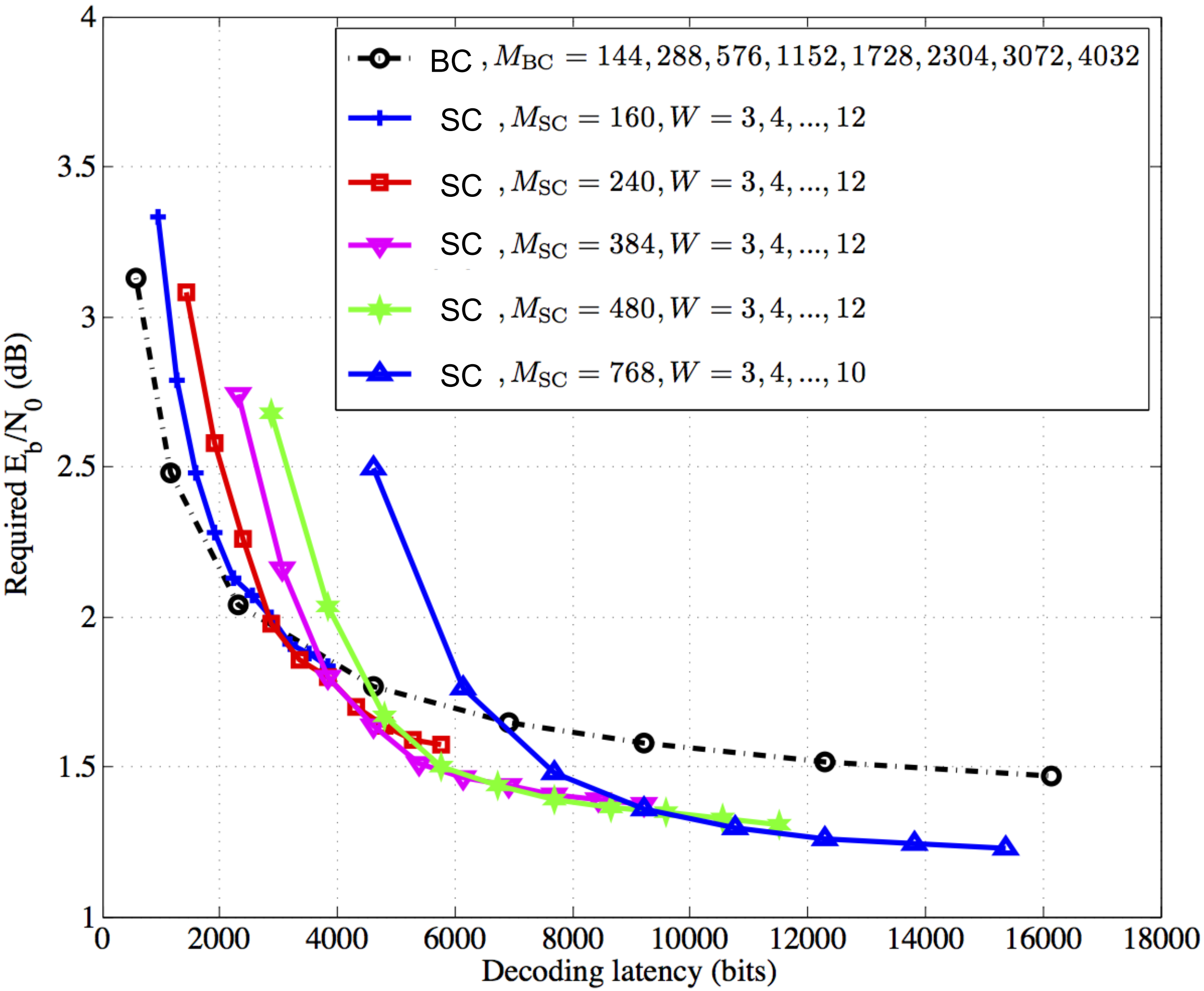
Latencies:
LDPC: $4M_{BC}$
SC-LDPC: $2M_{SC}W$

- ➡ decreases as W (and thus the latency) increases.
- ➡ does not decrease significantly beyond a certain W $(W \approx 10)$.

Equal Latency Comparison for (3,6)-Regular LDPC Codes

UNIVERSITY OF NOTRE DAME

- Required E_b/N_0 to achieve a BER of 10^{-5} as a function of latency:



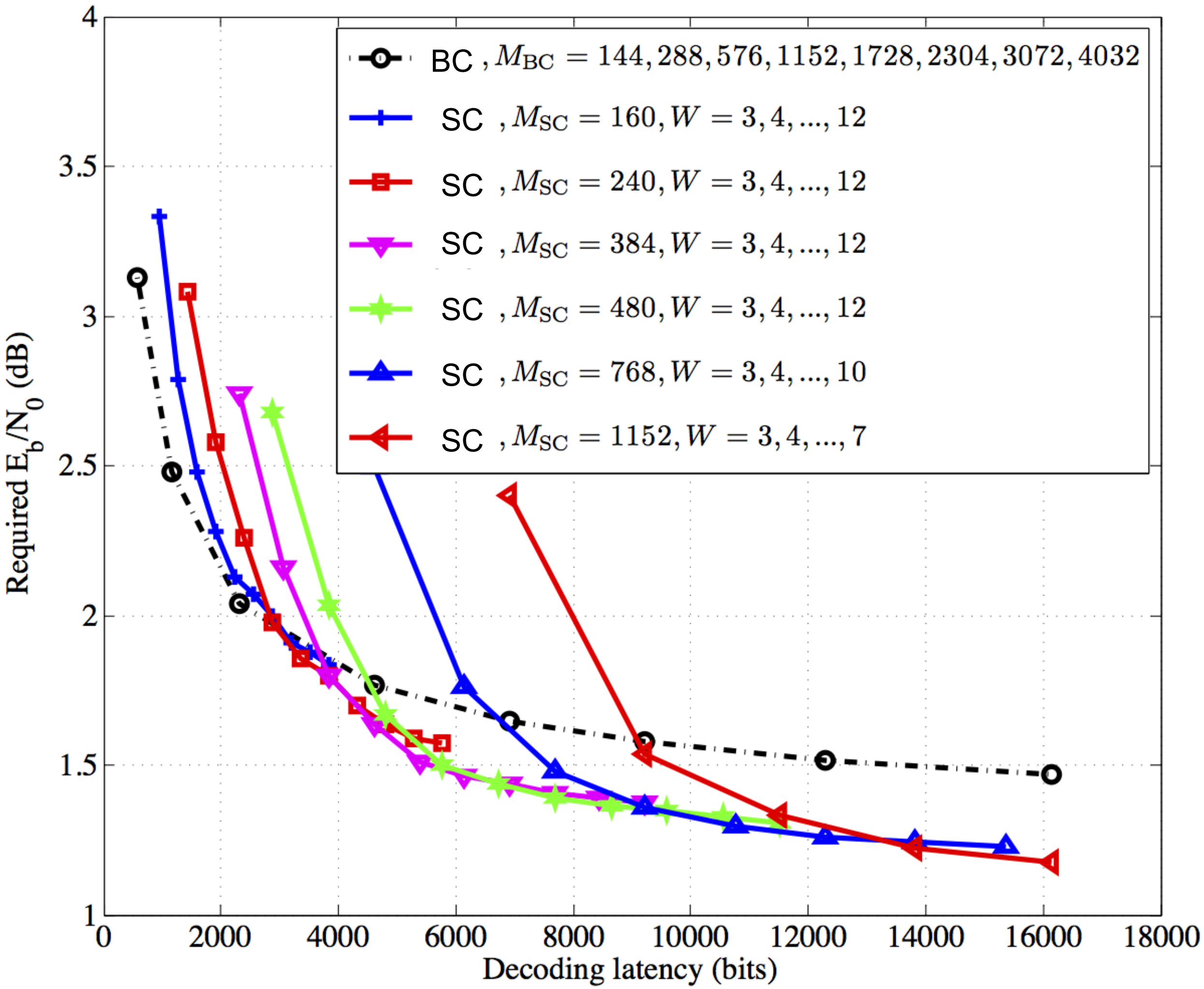
Latencies:
LDPC: $4M_{BC}$
SC-LDPC: $2M_{SC}W$

➡ decreases as W (and thus the latency) increases.

➡ does not decrease significantly beyond a certain W $(W \approx 10)$.

Equal Latency Comparison for (3,6)-Regular LDPC Codes

- Required E_b/N_0 to achieve a BER of 10^{-5} as a function of latency:

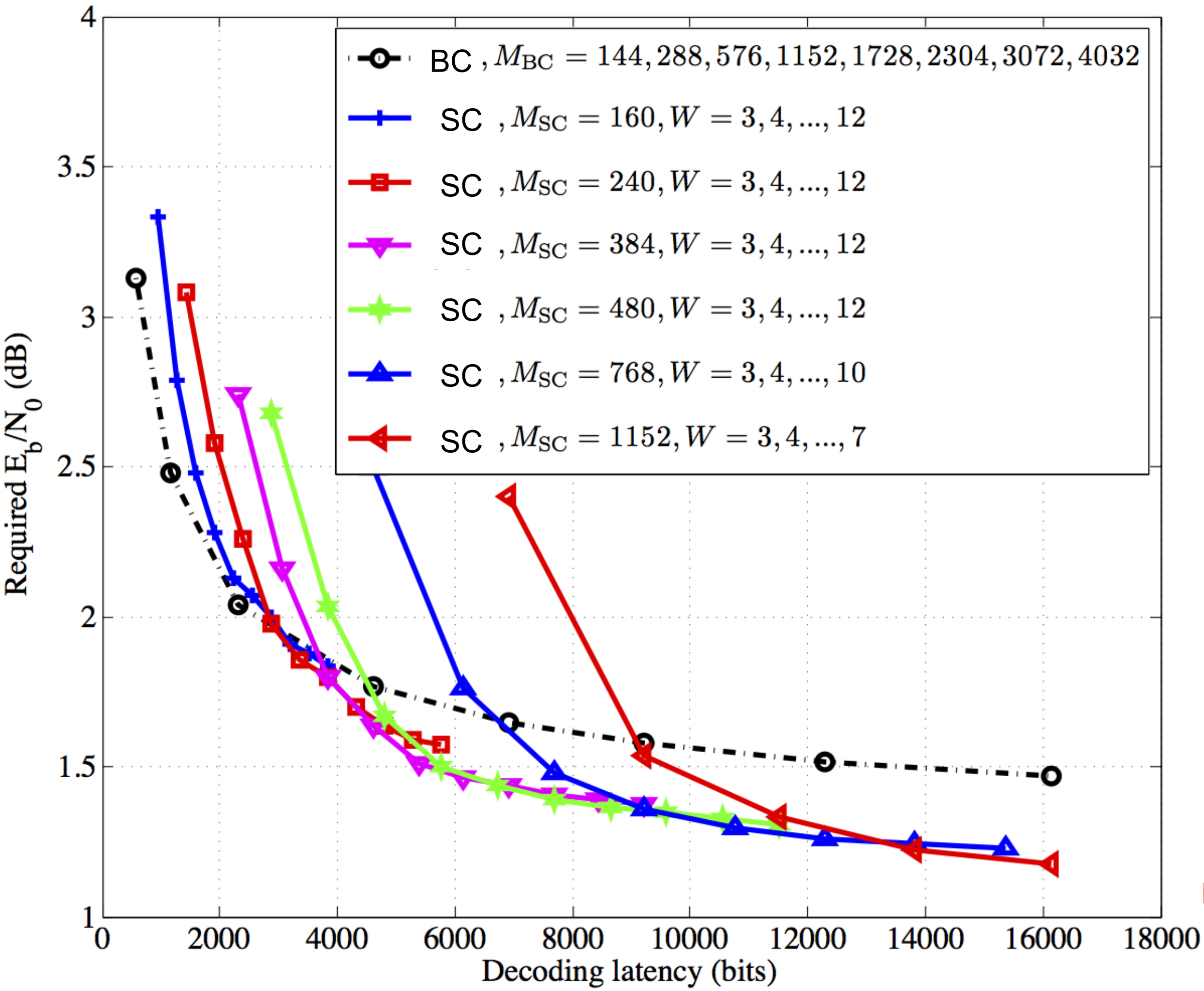


Latencies:
LDPC: $4M_{BC}$
SC-LDPC: $2M_{SC}W$

- ➡ decreases as W (and thus the latency) increases.
- ➡ does not decrease significantly beyond a certain W $(W \approx 10)$.

Equal Latency Comparison for (3,6)-Regular LDPC Codes

- Required E_b/N_0 to achieve a BER of 10^{-5} as a function of latency:



Latencies:
LDPC: $4M_{BC}$
SC-LDPC: $2M_{SC}W$

➡ decreases as W (and thus the latency) increases.

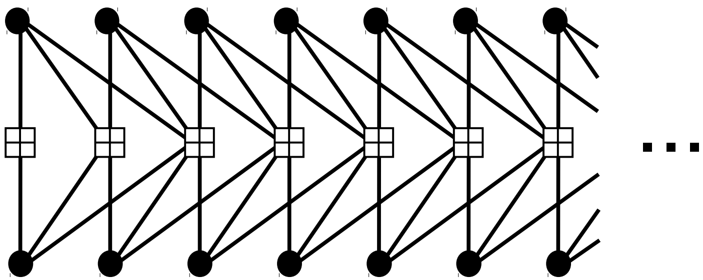
➡ does not decrease significantly beyond a certain W $(W \approx 10)$.

- When choosing parameters:

➡ large M_{SC} improves code performance.

➡ large W improves decoder performance.

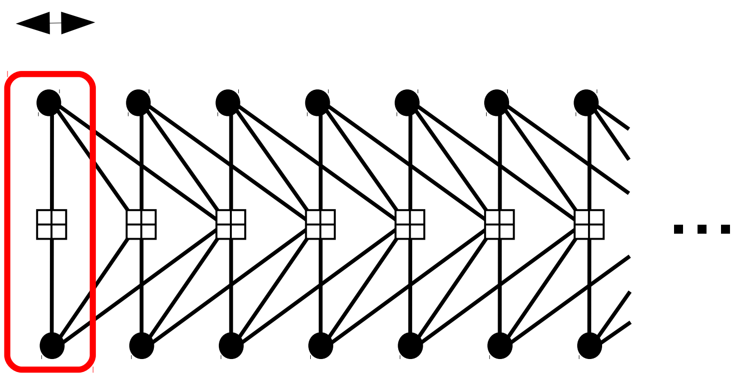
Protograph design



Protograph design

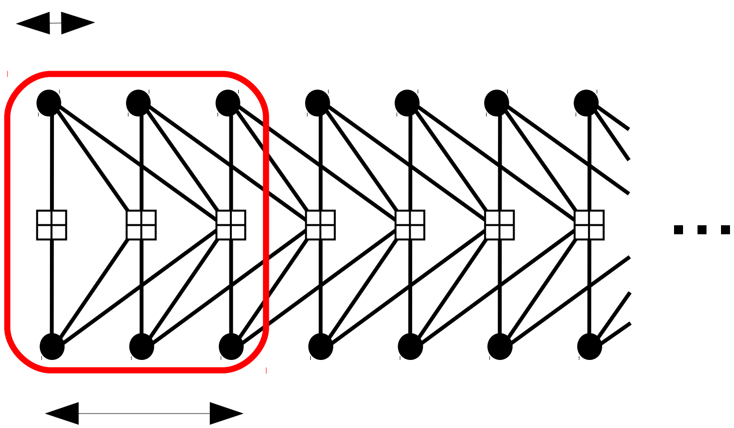
block size

$N_B = cM = 2M$



Protograph design

block size
$N_B = cM = 2M$

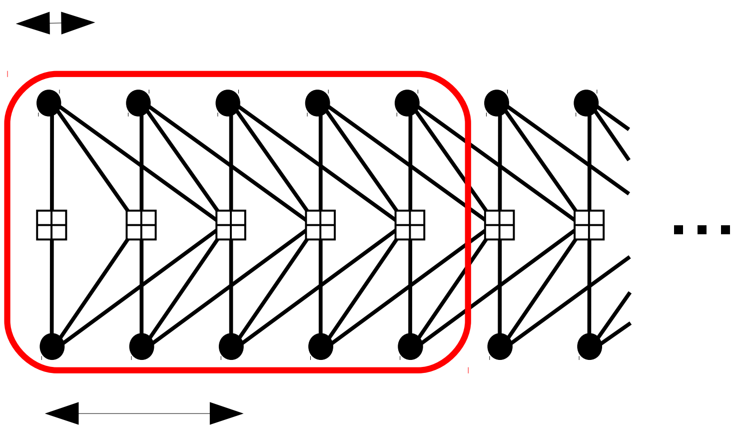


coupling width w=2 ⟷ constraint length $\nu = cM(w+1) = 6M$

Protograph design

block size

$N_B = cM = 2M$



coupling width w=2 ⟷ constraint length $\nu = cM(w+1) = 6M$

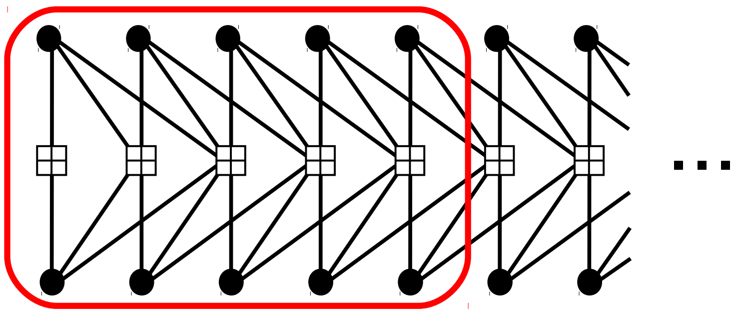
window span $W = 5 \geq w+1$ ⟷ window size $S = cMW = 10M$

Protograph design

UNIVERSITY OF NOTRE DAME

block size

$N_B = cM = 2M$

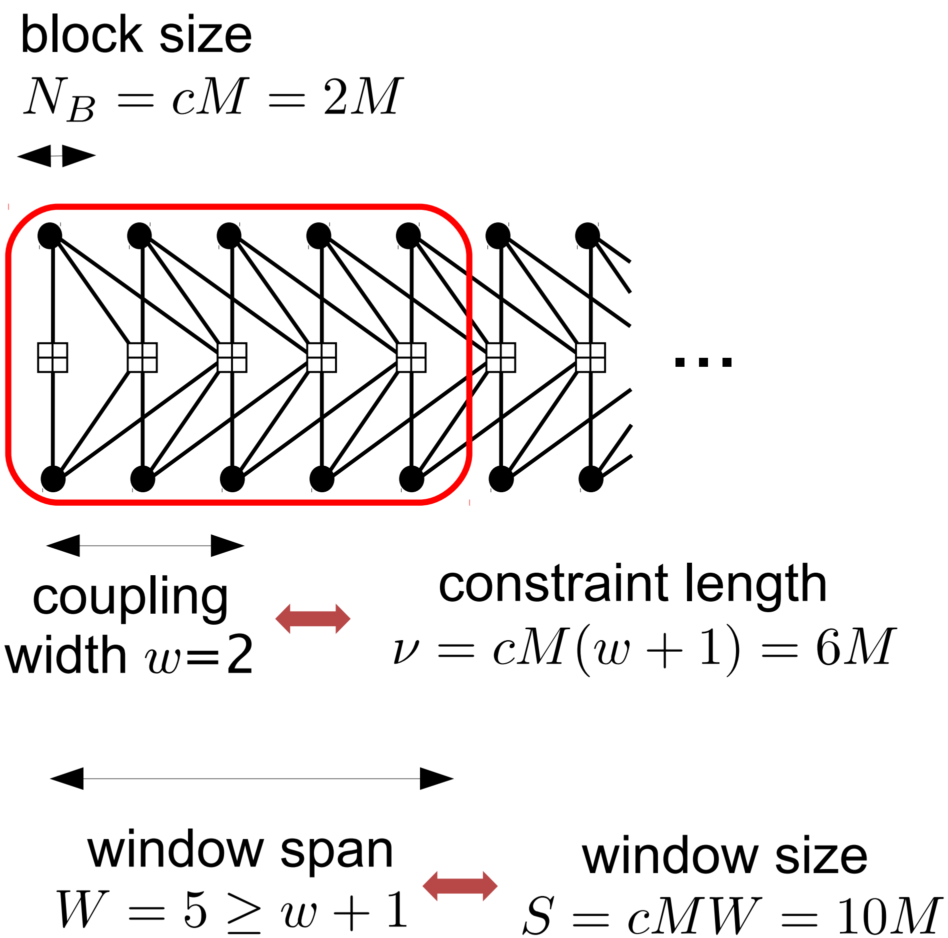


coupling width w=2 ⟷ constraint length $\nu = cM(w+1) = 6M$

window span $W = 5 \geq w + 1$ ⟷ window size $S = cMW = 10M$

- The **strength** of a convolutional code grows with **constraint length** $\nu = cM(w+1)$

Protograph design

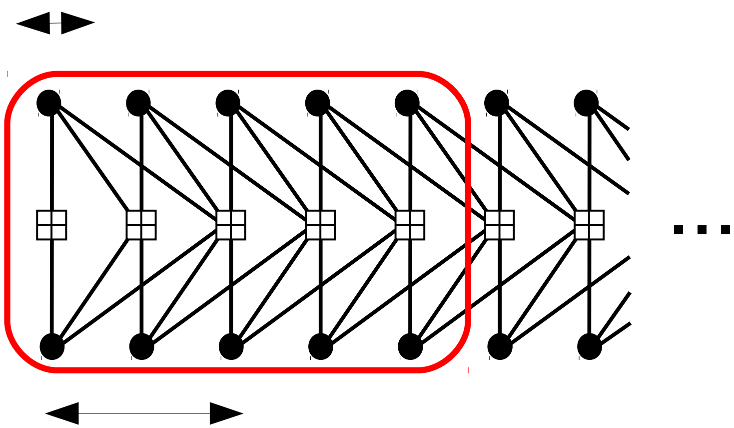


- The **strength** of a convolutional code grows with **constraint length** $\nu = cM(w+1)$
- The **latency** of the decoder is given by the **window size**

  $S = cMW \geq cM(w+1)$

Protograph design

block size
$N_B = cM = 2M$



coupling width w=2 ⟺ constraint length $\nu = cM(w+1) = 6M$

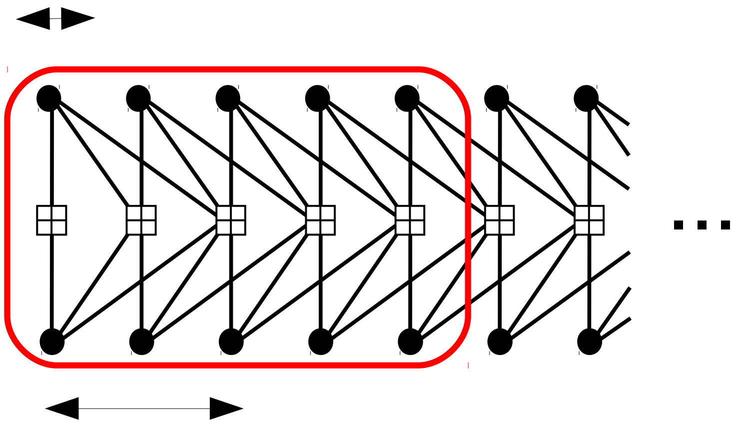
window span $W = 5 \geq w+1$ ⟺ window size $S = cMW = 10M$

- The **strength** of a convolutional code grows with **constraint length** $\nu = cM(w+1)$
- The **latency** of the decoder is given by the **window size**
  $$S = cMW \geq cM(w+1)$$

➡ For fixed ν or S, we obtain many small blocks or few large blocks by varying w and M.

Protograph design

block size
$N_B = cM = 2M$



coupling width w=2 ⟷ constraint length $\nu = cM(w+1) = 6M$

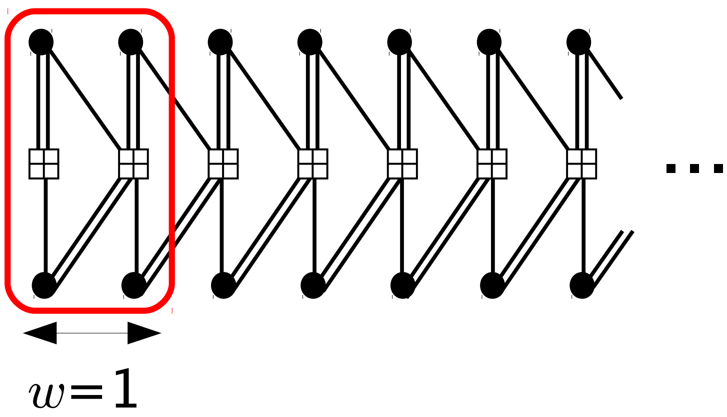
window span $W = 5 \geq w+1$ ⟷ window size $S = cMW = 10M$

- The **strength** of a convolutional code grows with **constraint length** $\nu = cM(w+1)$
- The **latency** of the decoder is given by the **window size**
  $S = cMW \geq cM(w+1)$
  - ➡ For fixed ν or S, we obtain many small blocks or few large blocks by varying w and M.
- Density evolution does not tell us how to choose these parameters to **optimize finite-length performance**

Examples: Protograph design

- We start from $\mathbf{B} = [3\ 3]$ and spread the edges such that $\sum_{i=0}^{w} \mathbf{B}_i = \mathbf{B}$

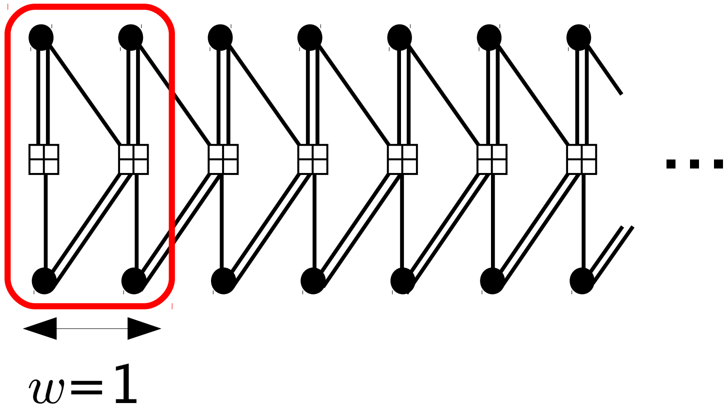
$$[\mathbf{B}_0;\ \mathbf{B}_1] = [2\,1;1\,2]$$



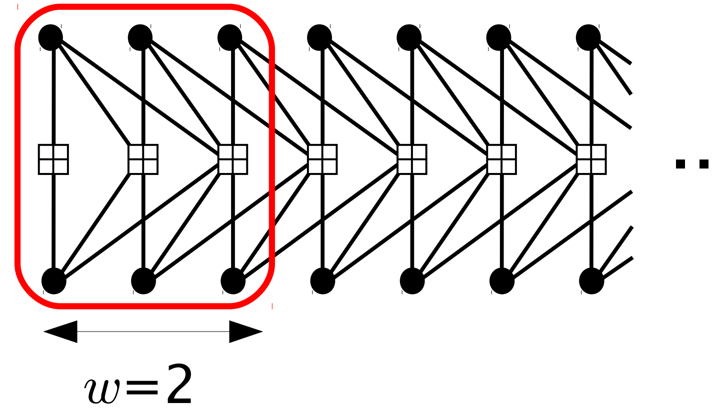
Examples: Protograph design

- We start from $\mathbf{B} = [3\ 3]$ and spread the edges such that $\sum_{i=0}^{w} \mathbf{B}_i = \mathbf{B}$

$$[\mathbf{B}_0;\ \mathbf{B}_1] = [2\,1;1\,2]$$



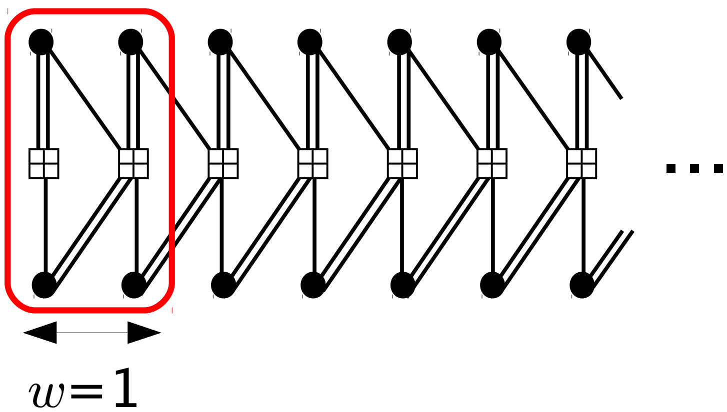
$$[\mathbf{B}_0;\ \mathbf{B}_1;\ \mathbf{B}_2] = [1\,1;1\,1;1\,1]$$



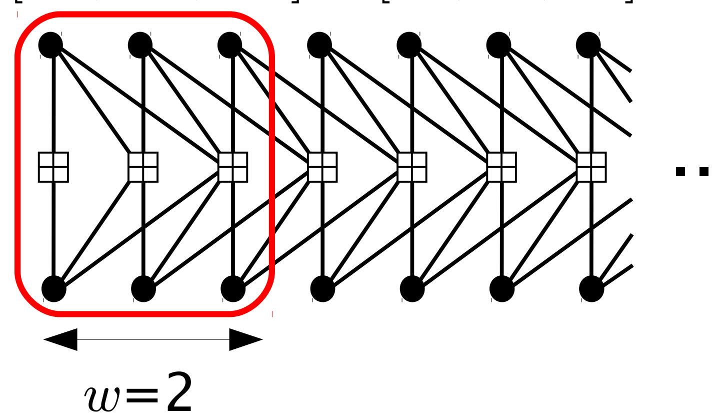
Examples: Protograph design

- We start from $\mathbf{B} = [3\ 3]$ and spread the edges such that $\sum_{i=0}^{w} \mathbf{B}_i = \mathbf{B}$

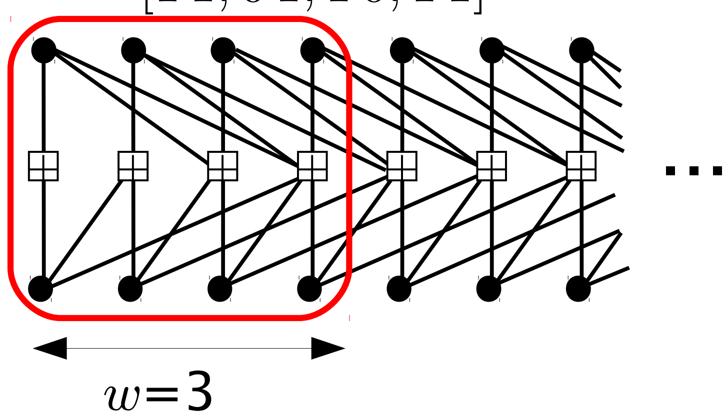
$[\mathbf{B}_0;\ \mathbf{B}_1] = [2\,1; 1\,2]$



$[\mathbf{B}_0;\ \mathbf{B}_1;\ \mathbf{B}_2] = [1\,1; 1\,1; 1\,1]$



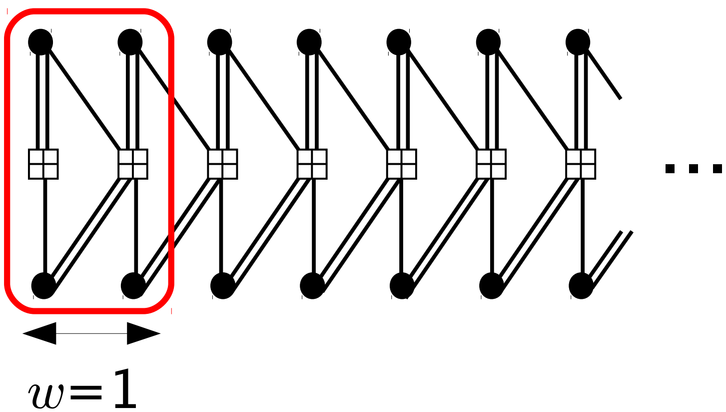
$[1\,1; 0\,1; 1\,0; 1\,1]$



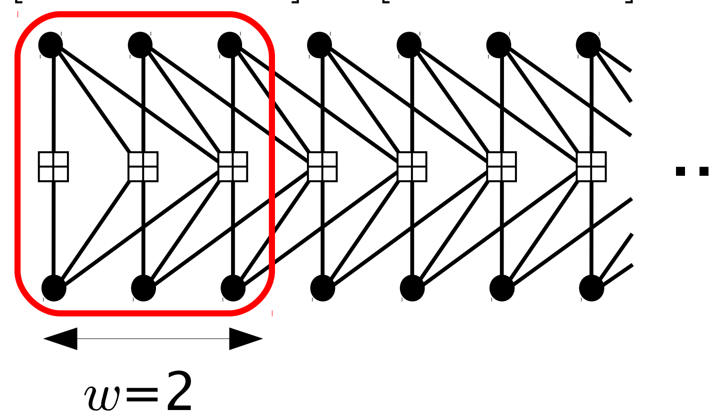
Examples: Protograph design

- We start from $\mathbf{B} = [3\ 3]$ and spread the edges such that $\sum_{i=0}^{w} \mathbf{B}_i = \mathbf{B}$

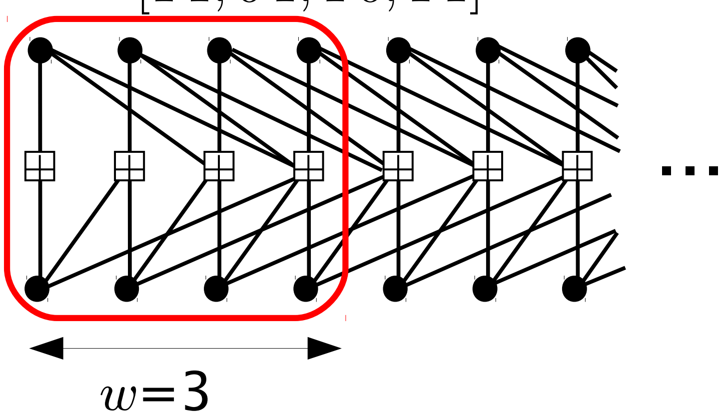
$[\mathbf{B}_0;\ \mathbf{B}_1] = [2\,1;\,1\,2]$



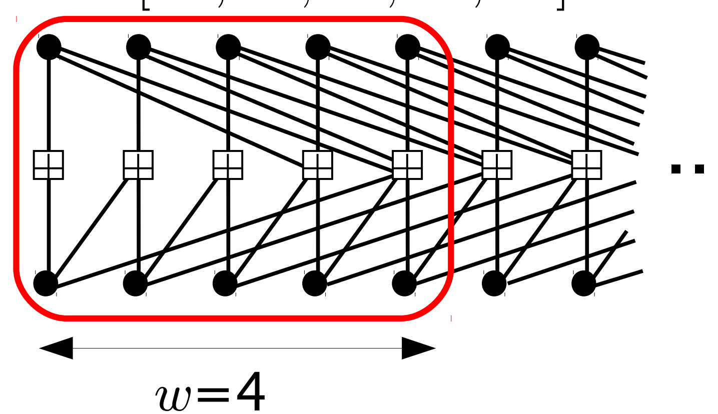
$[\mathbf{B}_0;\ \mathbf{B}_1;\ \mathbf{B}_2] = [1\,1;\,1\,1;\,1\,1]$



$[1\,1;\,0\,1;\,1\,0;\,1\,1]$



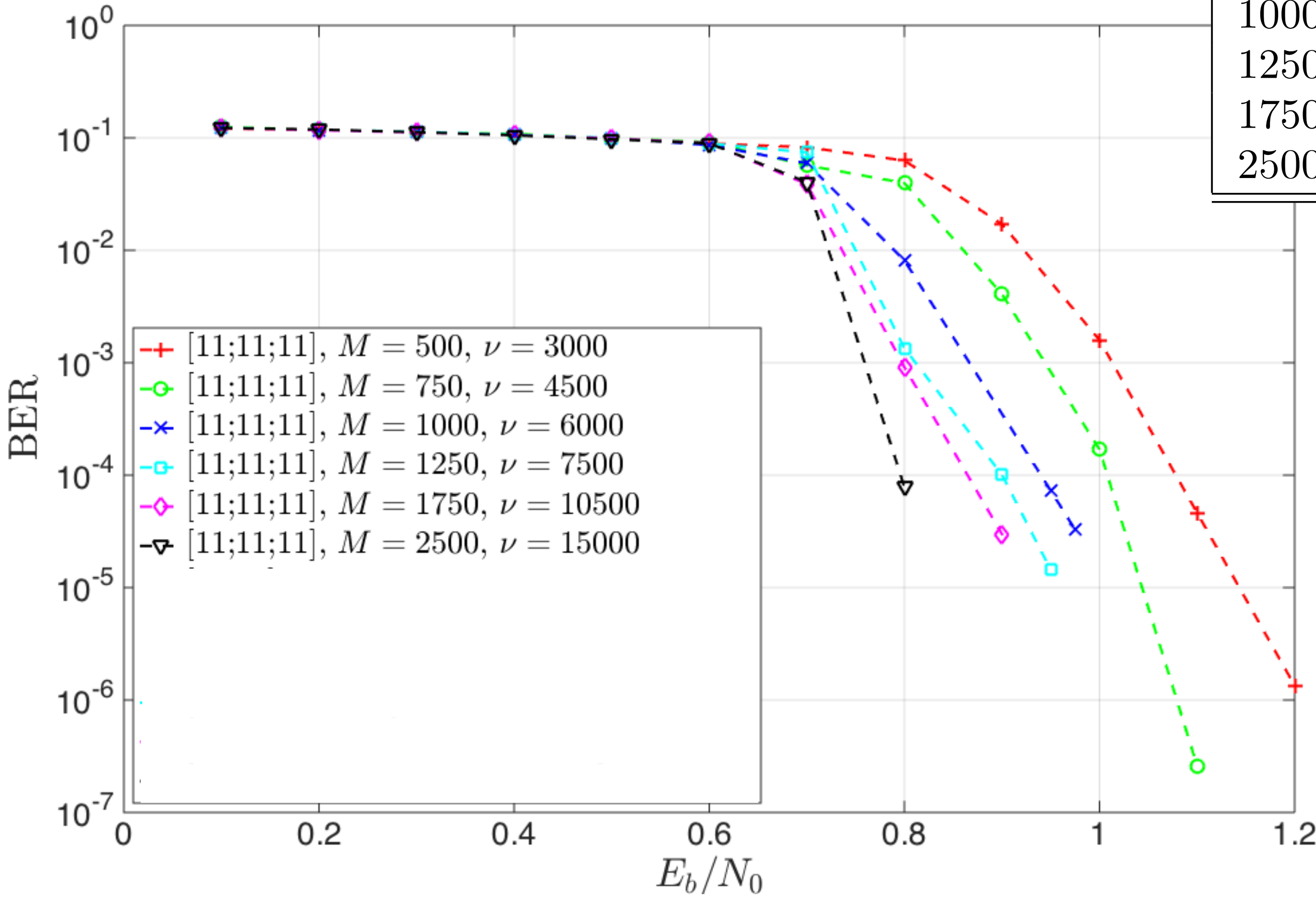
$[1\,1;\,0\,1;\,0\,0;\,1\,0;\,1\,1]$



WD Scaling - AWGNC

- BER of the w = 2, [11;11;11] codes demonstrating the scaling with $\nu = 2M(w+1)$ for increasing M and fixed w

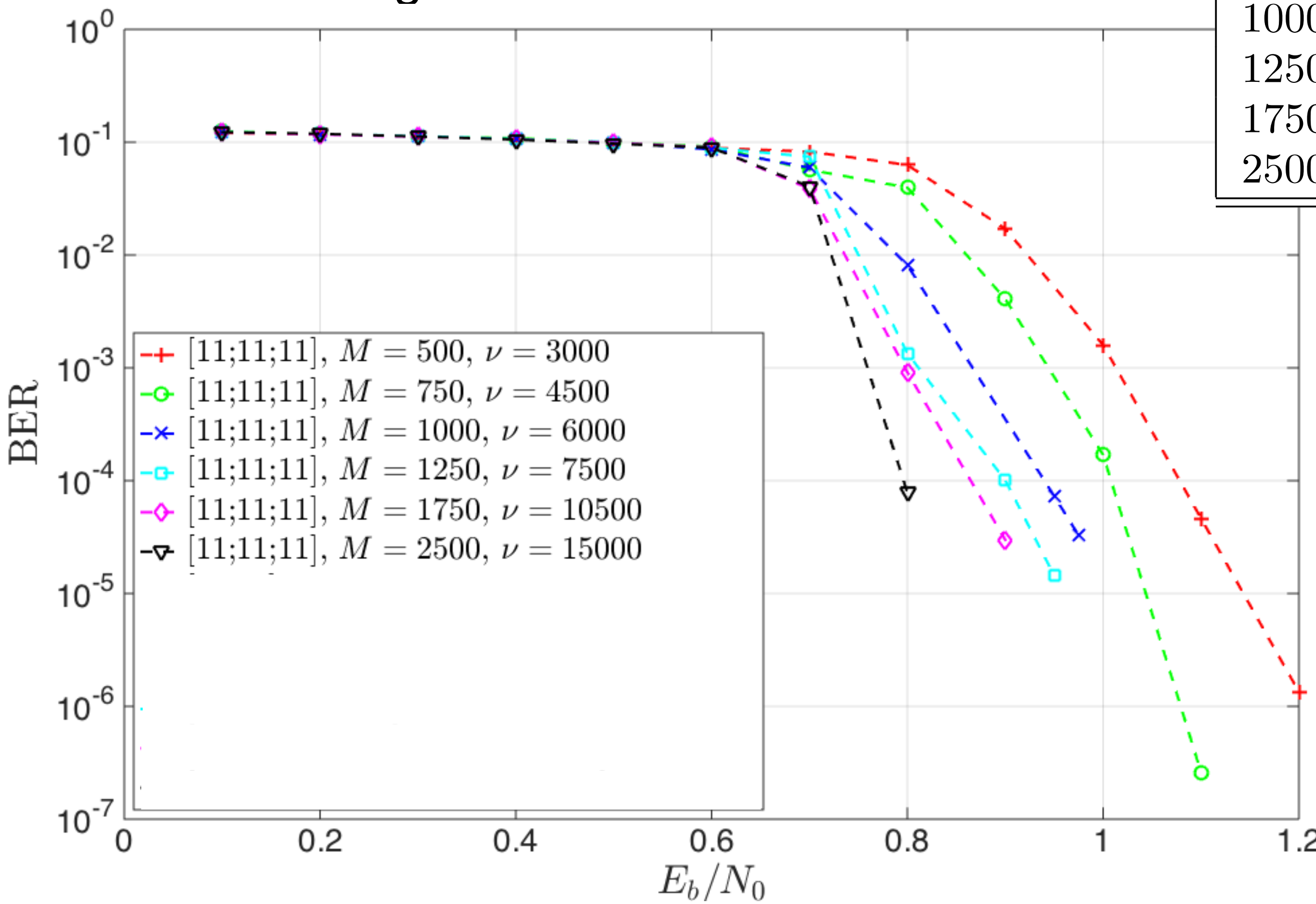
M	N_B	ν	S
500	1000	3000	12,000
750	1500	4500	18,000
1000	2000	6000	24,000
1250	2500	7500	30,000
1750	3500	10500	42,000
2500	5000	15000	60,000



WD Scaling - AWGNC

- BER of the w = 2, [11;11;11] codes demonstrating the scaling with $\nu = 2M(w+1)$ for increasing M and fixed w

M	N_B	ν	S
500	1000	3000	12,000
750	1500	4500	18,000
1000	2000	6000	24,000
1250	2500	7500	30,000
1750	3500	10500	42,000
2500	5000	15000	60,000

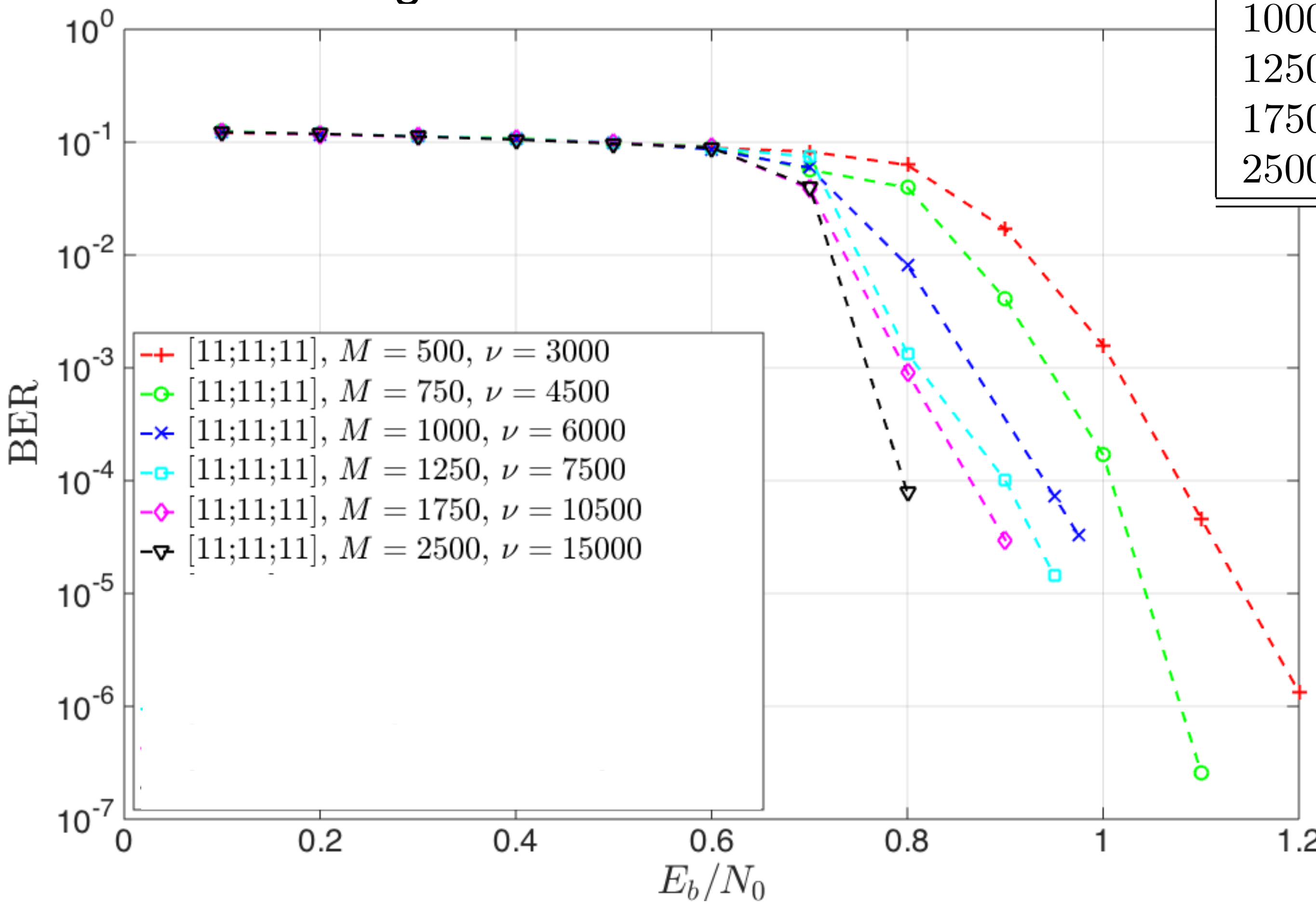


- $\nu = 2M(w+1)$ can also be increased with w for fixed M

WD Scaling - AWGNC

- BER of the w = 2, [11;11;11] codes demonstrating the scaling with $\nu = 2M(w+1)$ for increasing M and fixed w

M	N_B	ν	S
500	1000	3000	12,000
750	1500	4500	18,000
1000	2000	6000	24,000
1250	2500	7500	30,000
1750	3500	10500	42,000
2500	5000	15000	60,000

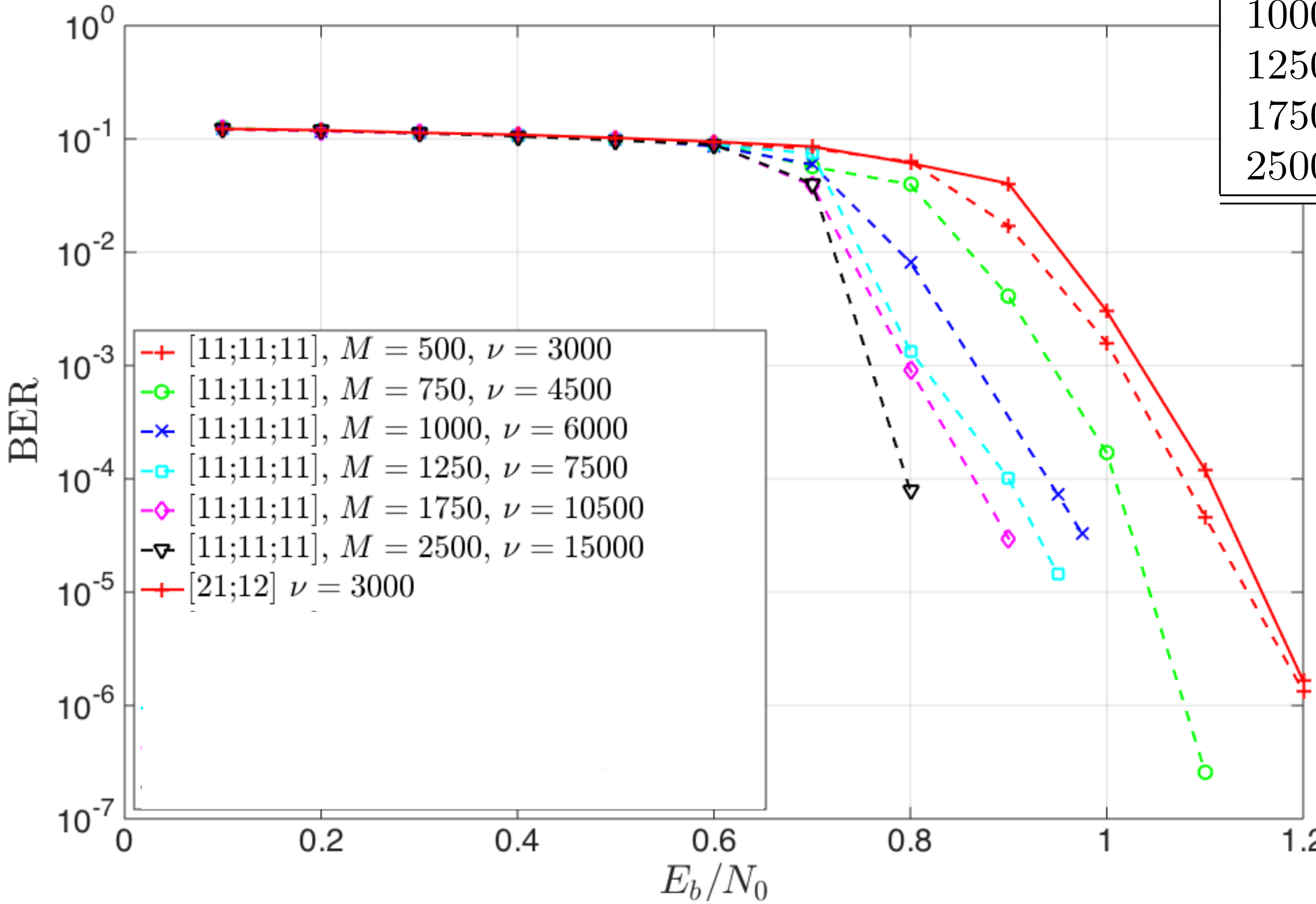


- $\nu = 2M(w+1)$ can also be increased with w for fixed M
- For fixed M = 750 (N_B = 1500) we observe improved scaling with w.

WD Scaling - AWGNC

- BER of the w = 2, [11;11;11] codes demonstrating the scaling with $\nu = 2M(w+1)$ for increasing M and fixed w

M	N_B	ν	S
500	1000	3000	12,000
750	1500	4500	18,000
1000	2000	6000	24,000
1250	2500	7500	30,000
1750	3500	10500	42,000
2500	5000	15000	60,000



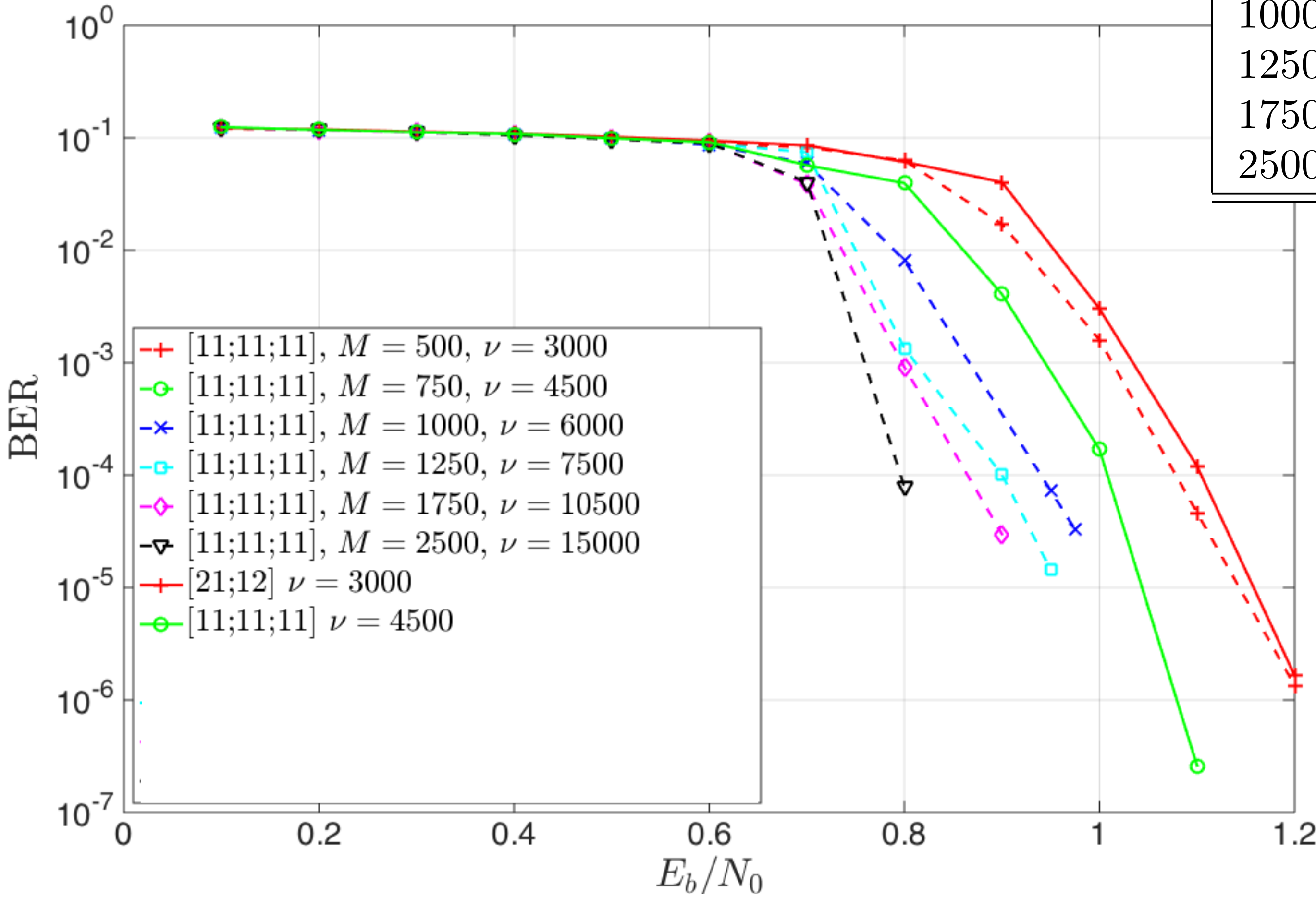
- $\nu = 2M(w+1)$ can also be increased with w for fixed M

- For fixed M = 750 (N_B = 1500) we observe improved scaling with w.

WD Scaling - AWGNC

- BER of the w = 2, [11;11;11] codes demonstrating the scaling with $\nu = 2M(w+1)$ for increasing M and fixed w

M	N_B	ν	S
500	1000	3000	12, 000
750	1500	4500	18, 000
1000	2000	6000	24, 000
1250	2500	7500	30, 000
1750	3500	10500	42, 000
2500	5000	15000	60, 000



- $\nu = 2M(w+1)$ can also be increased with w for fixed M

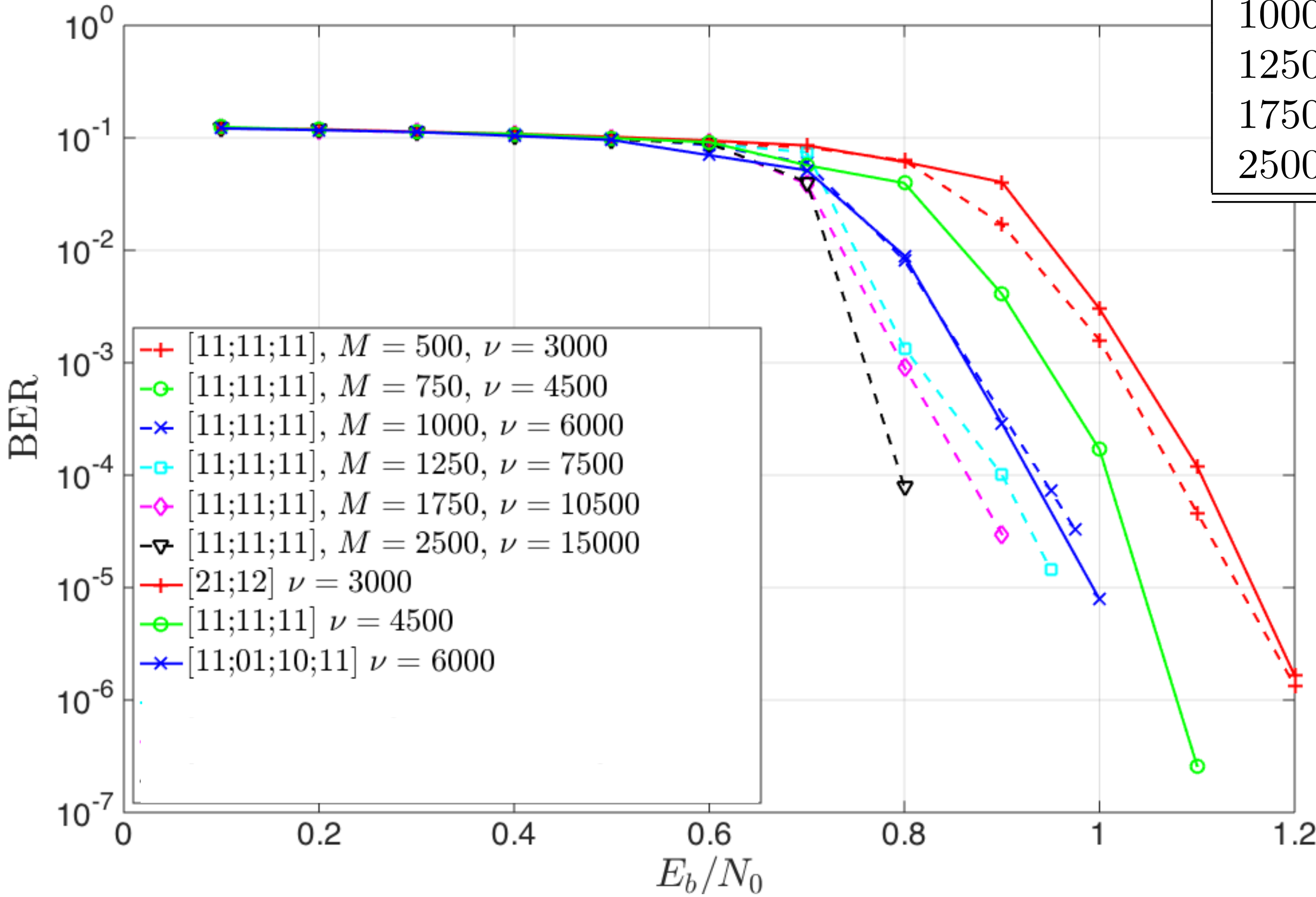
- For fixed M = 750 (N_B = 1500) we observe improved scaling with w.

WD Scaling - AWGNC

UNIVERSITY OF NOTRE DAME

- BER of the w = 2, [11;11;11] codes demonstrating the scaling with $\nu = 2M(w+1)$ for increasing M and fixed w

M	N_B	ν	S
500	1000	3000	12,000
750	1500	4500	18,000
1000	2000	6000	24,000
1250	2500	7500	30,000
1750	3500	10500	42,000
2500	5000	15000	60,000

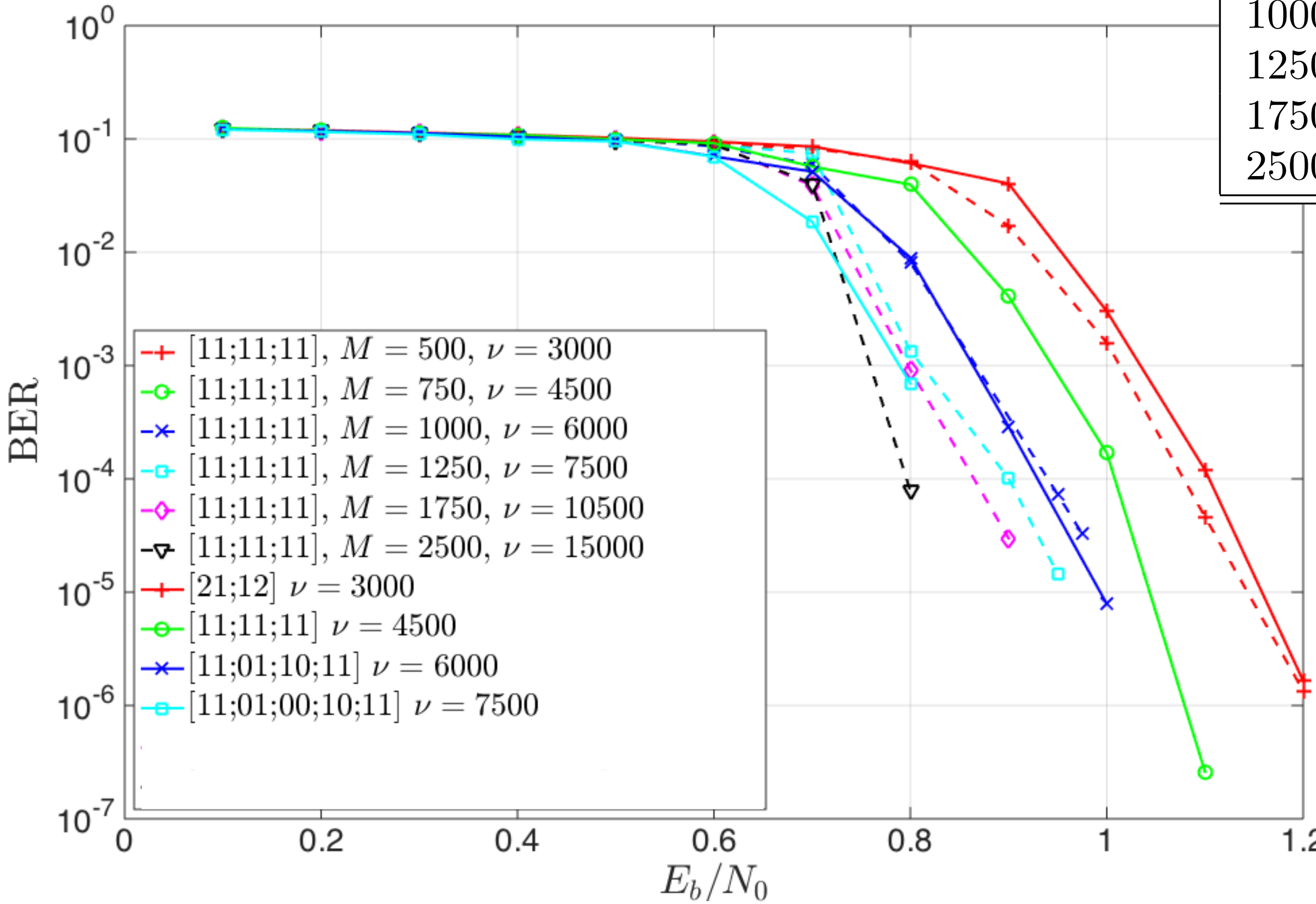


- $\nu = 2M(w+1)$ can also be increased with w for fixed M
- For fixed M = 750 (N_B = 1500) we observe improved scaling with w.

WD Scaling - AWGNC

- BER of the w = 2, [11;11;11] codes demonstrating the scaling with $\nu = 2M(w+1)$ for increasing M and fixed w

M	N_B	ν	S
500	1000	3000	12,000
750	1500	4500	18,000
1000	2000	6000	24,000
1250	2500	7500	30,000
1750	3500	10500	42,000
2500	5000	15000	60,000



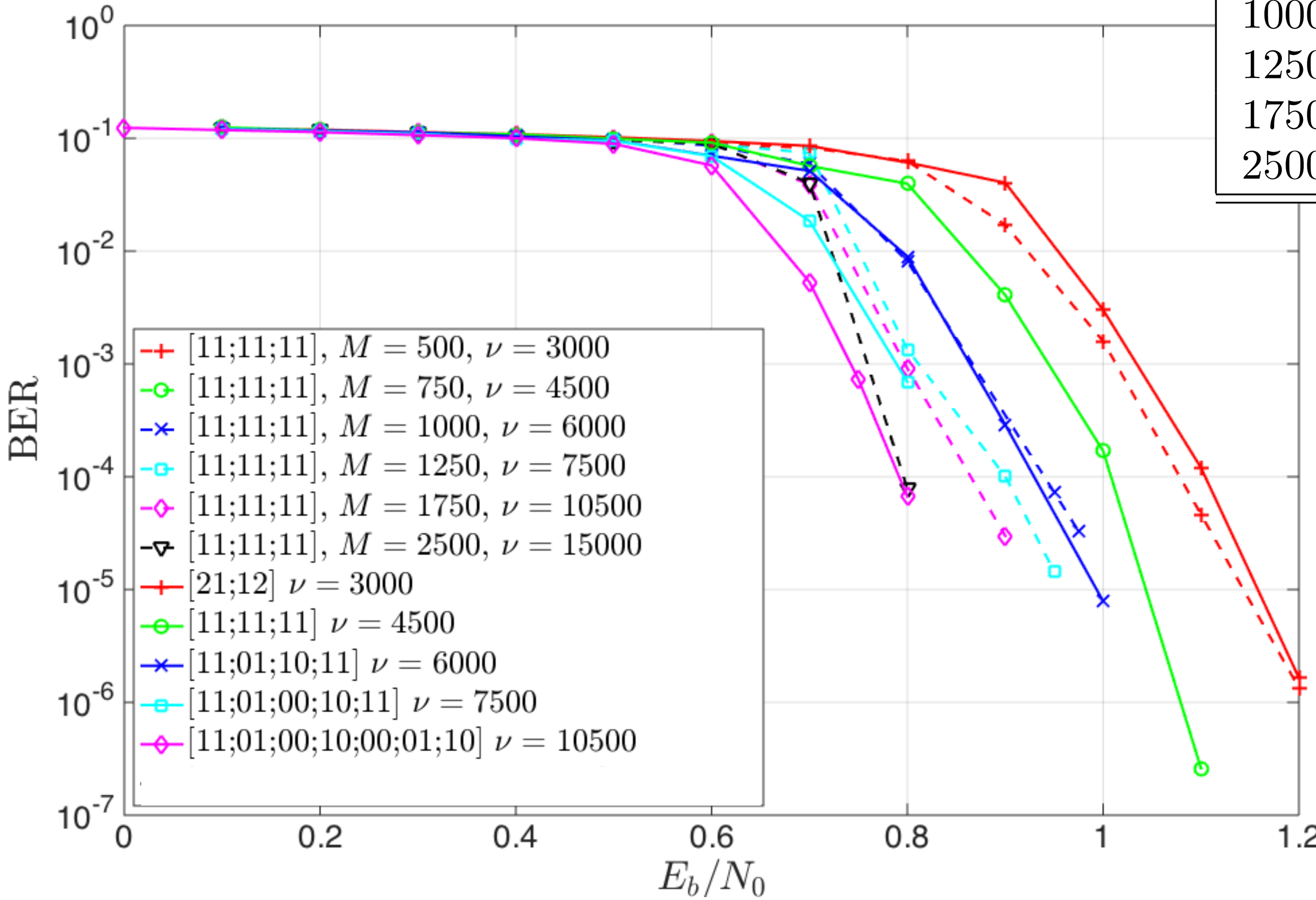
- $\nu = 2M(w+1)$ can also be increased with w for fixed M

- For fixed M = 750 (N_B = 1500) we observe improved scaling with w.

WD Scaling - AWGNC

- BER of the w = 2, [11;11;11] codes demonstrating the scaling with $\nu = 2M(w+1)$ for increasing M and fixed w

M	N_B	ν	S
500	1000	3000	12,000
750	1500	4500	18,000
1000	2000	6000	24,000
1250	2500	7500	30,000
1750	3500	10500	42,000
2500	5000	15000	60,000

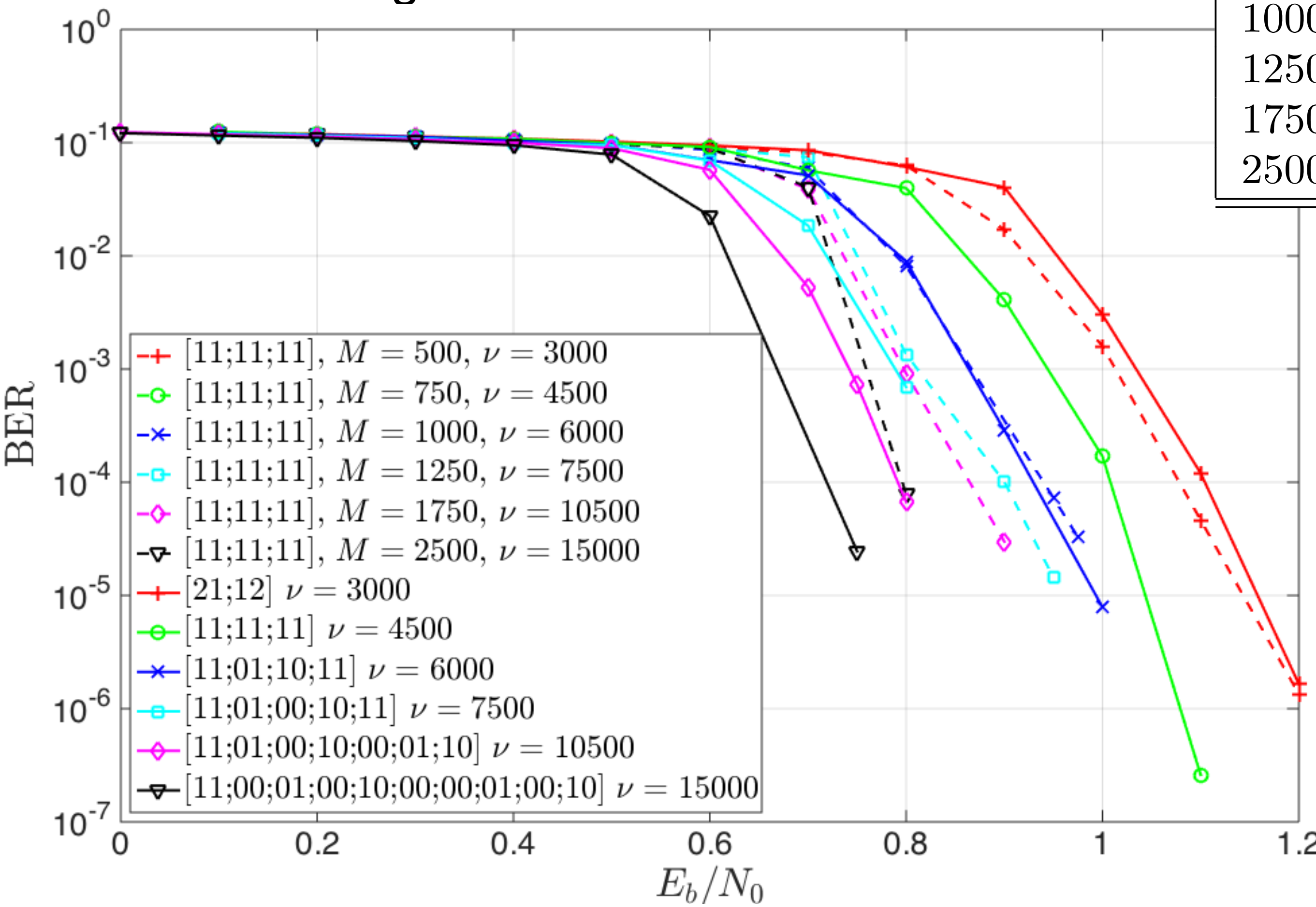


- $\nu = 2M(w+1)$ can also be increased with w for fixed M
- For fixed M = 750 (N_B = 1500) we observe improved scaling with w.

WD Scaling - AWGNC

- BER of the w = 2, [11;11;11] codes demonstrating the scaling with $\nu = 2M(w+1)$ for increasing M and fixed w

M	N_B	ν	S
500	1000	3000	12,000
750	1500	4500	18,000
1000	2000	6000	24,000
1250	2500	7500	30,000
1750	3500	10500	42,000
2500	5000	15000	60,000



- $\nu = 2M(w+1)$ can also be increased with w for fixed M
- For fixed M = 750 (N_B = 1500) we observe improved scaling with w.

Implementation Aspects

- As a result of their capacity approaching performance and simple structure, regular SC-LDPC codes may be attractive for future coding standards. Several key features will require further investigation:
  - ➡ Hardware advantages of QC designs obtained by circulant liftings
  - ➡ Hardware advantages of the 'asymptotically-regular' structure
  - ➡ Design advantages of flexible frame length and flexible rate obtained by varying M, L, and/or puncturing

Implementation Aspects

UNIVERSITY OF NOTRE DAME

- As a result of their capacity approaching performance and simple structure, regular SC-LDPC codes may be attractive for future coding standards. Several key features will require further investigation:

  - ➡ Hardware advantages of QC designs obtained by circulant liftings
  - ➡ Hardware advantages of the 'asymptotically-regular' structure
  - ➡ Design advantages of flexible frame length and flexible rate obtained by varying M, L, and/or puncturing

- Of particular importance for applications requiring extremely low decoded bit error rates (*e.g.*, optical communication, data storage) is an investigation of error floor issues related to **stopping sets**, **trapping sets**, and **absorbing sets**.

Conclusions

UNIVERSITY OF NOTRE DAME

- Spatially coupled LDPC code ensembles achieve **threshold saturation**, i.e., their iterative decoding thresholds (for large L and M) approach the MAP decoding thresholds of the underlying LDPC block code ensembles.
- The threshold saturation and linear minimum distance growth properties of (J,K)-regular SC-LDPC codes combine the best asymptotic features of both regular and irregular LDPC-BCs.
- With window decoding, SC-LDPC codes also compare favorably to LDPC-BCs in the finite-length regime, providing flexible tradeoffs between BER performance, decoding latency, and decoding complexity.
- Flexible frame lengths and rates can be obtained by varying M, L, and/or puncturing.