Introduction to Data Communications and Networking

Channel coding

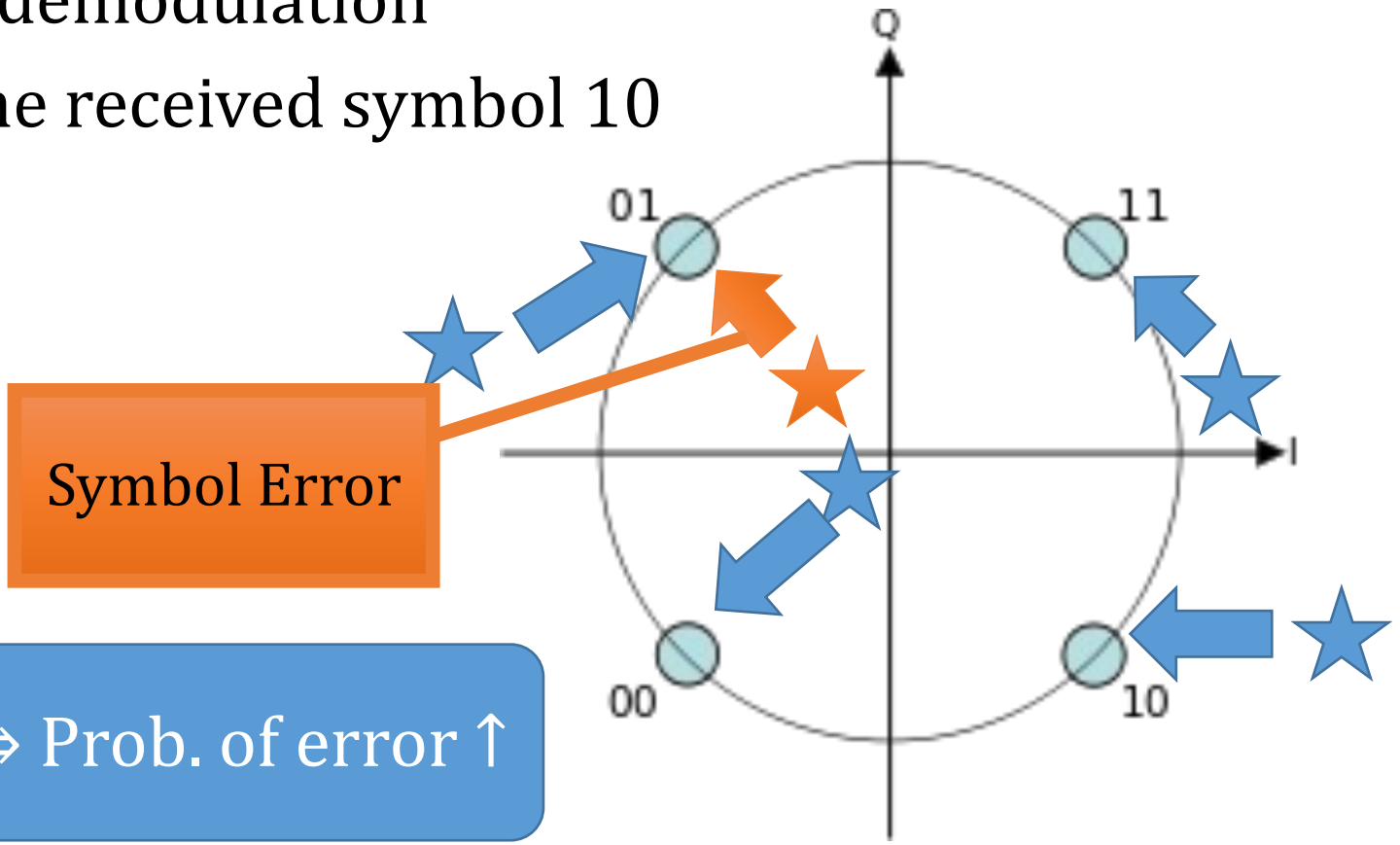
郭桐惟

Motivation

- Data can be corrupted due to traffic impairment
  1. Attenuation
  2. Atmospheric absorption
  3. Distortion
  4. Noise

Demodulation

- ★ : the received symbol
- ➡ : demodulation
- ★ : the received symbol 10



Noise ↑⇒ Prob. of error ↑

Motivation

- Many applications cannot tolerate error
- What should the receiver do to recover from error?

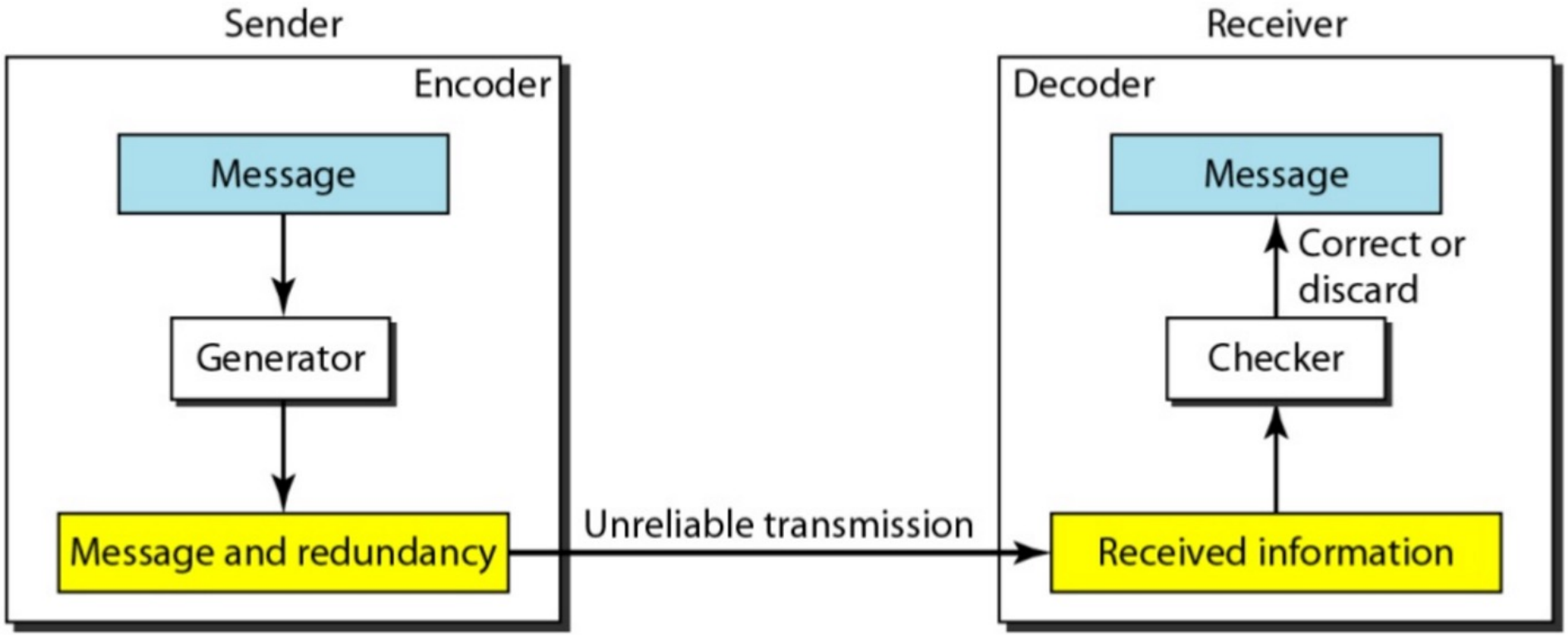
1. Detect error and ask the sender to retransmit
2. Correct error

- In this lecture, we will learn how to detect and correct error

The Basic Idea of Error Detection/Correction

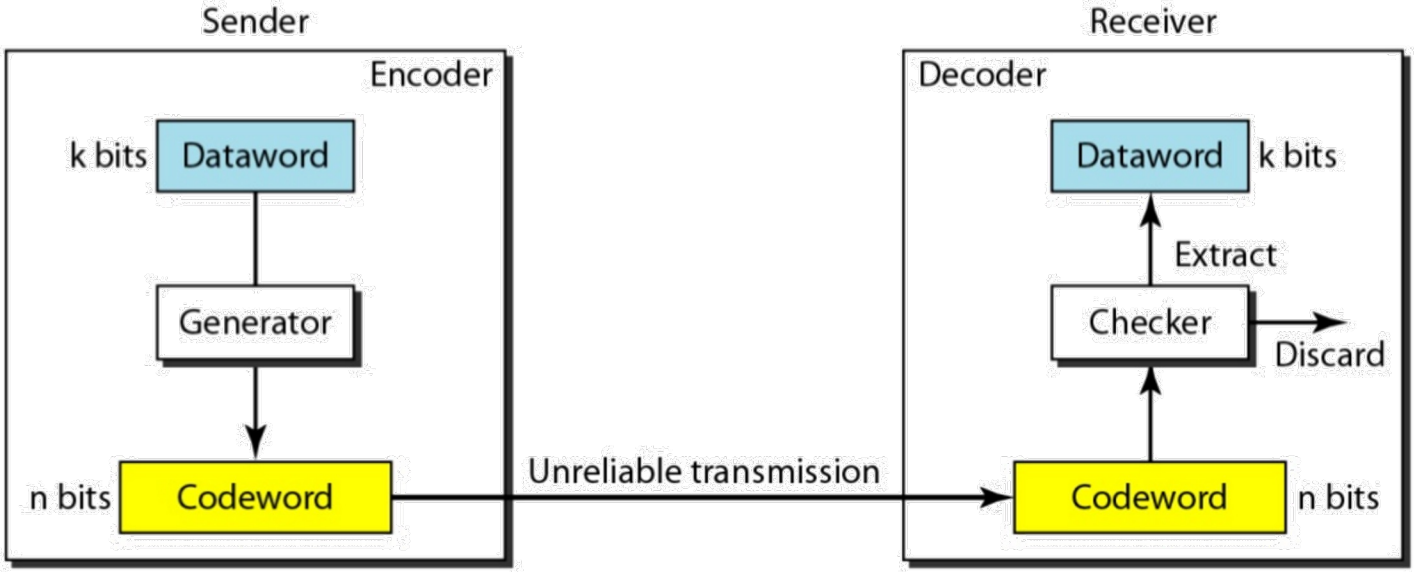
- The sender adds more bits to the data
- The receiver uses these additional bits to detect/correct error
- These additional bits are determined by **coding scheme**

The Basic Idea of Error Detection/Correction



Block Coding

- We divide our original message into blocks, each of k bits, called **datawords**
- We add r additional bits to each block
- The length of each block is $n = k + r$



Toy Examples of Error Detection & Correction

A Toy Example of Error Detection

- $k = 2, r = 1, n = 3$

Datawords	Additional Bits	Codewords
00	0	000
01	1	011
10	1	101
11	0	110

A Toy Example of Error Detection

- Assume the sender sends 01
- 01 is encoded and becomes 011
- If the receiver receives 011, the receiver extracts 01 from it

Datawords	Additional Bits	Codewords
00	0	000
01	1	011
10	1	101
11	0	110

A Toy Example of Error Detection

- Assume the sender sends 01
- 01 is encoded and becomes 011
- If the codeword is corrupted and the receiver receives 111, then the receiver knows that the codeword is corrupted. Why?

Datawords	Additional Bits	Codewords
00	0	000
01	1	011
10	1	101
11	0	110

A Toy Example of Error Detection

- Assume the sender sends 01
- 01 is encoded and becomes 011
- If the codeword is corrupted and the receiver receives 000, the receiver cannot detect the error. Why? cannot detect an error of 2 bits

Datawords	Additional Bits	Codewords
00	0	000
01	1	011
10	1	101
11	0	110

High-Level Idea of Error Detection

- If the received codeword is **invalid**, the receiver knows that the codeword is corrupted
- If the received codeword is **valid**, the receiver accepts the codeword and extract the dataword from it
  - Error might be undetected
  - That is, error makes a valid codeword become another valid codeword

The Performance of Error Detection

- If the length of datawords is k, then how many different datawords are there?
  - 2^k
  - Since one dataword is mapped to only one codeword, there are 2^k valid codewords
- If the length of codewords is $n = k + r$, then how many different codewords are there?
  - 2^{k+r}
- Hence, only $\frac{1}{2^r}$ of the codewords are valid

A Toy Example of Error Correction

- $k = 2, r = 3, n = 5$

Datawords	Additional Bits	Codewords
00	000	00000
01	011	01011
10	101	10101
11	110	11110

A Toy Example of Error Correction

- Assume the sender sends 01
- 01 is encoded and becomes 01011
- Assume that the codeword is corrupted and the receiver receives 01001. The receiver knows that the codeword is corrupted. Why?

Datawords	Additional Bits	Codewords
00	000	00000 2
01	011	01011 1
10	101	10101 3
11	110	11110 4

A Toy Example of Error Correction

- Assume the sender sends 01
- 01 is encoded and becomes 01011
- Assume that the codeword is corrupted and the receiver receives 01001. **If the receiver knows that only 1 bit is corrupted, the error can be corrected.** Why?

Datawords	Additional Bits	Codewords
00	000	00000
01	011	01011
10	101	10101
11	110	11110

A Toy Example of Error Correction

- Assume the sender sends 01
- 01 is encoded and becomes 011
- Assume that the codeword is corrupted and the receiver receives 111. **If the receiver knows that only 1 bit is corrupted, the error cannot be corrected.** Why?

Datawords	Additional Bits	Codewords	
00	0	000	3
01	1	011	1
10	1	101	1
11	0	110	1

High-Level Idea of One-Bit Error Correction

- To correct 1 bit error, we find the valid codeword that differs with the received codeword in 1 bit
- If there are multiple valid codewords that differ with the received codeword in 1 bit, we do not know how to correct error
- How to determine whether a code can correct one-bit error?
  - We do not want to test every possible corrupted codeword

Hamming Distance

Hamming Distance

- The Hamming distance between two words (of the same length) is the number of differences between the corresponding bits
- $d(x, y)$: the Hamming distance between x and y
- $d(000, 011) = 2$
- $d(10101, 11110) = 3$

Hamming Distance

- $d(01001,00000) = 2$
- $d(01001,01011) = 1$
- $d(01001,10101) = 3$
- $d(01001,11110) = 4$

Datawords	Additional Bits	Codewords
00	000	00000
01	011	01011
10	101	10101
11	110	11110

Hamming Distance

- $d(111,000) = 3$
- $d(111,011) = 1$
- $d(111,101) = 1$
- $d(111,110) = 1$

Datawords	Additional Bits	Codewords
00	0	000
01	1	011
10	1	101
11	0	110

Error Detection

How to Detect One-Bit Error

- Assume the sent codeword is c_s (must be valid)
- To detect one-bit error, any valid codeword c_v other than c_s must have $d(c_s, c_v) > 1$
- $d(c_s, c_v) \geq 2$, for any two different valid codewords c_s, c_v
- In other words, to detect one-bit error, the minimum Hamming distance between two valid codewords must be at least 2

s: sent
v: valid

Hamming Distance

- $d(000,011) = 2, d(000,101) = 2, d(000,110) = 2$
- $d(011,101) = 2, d(011,110) = 2, d(101,110) = 2$
- The minimum Hamming distance is 2

Datawords	Additional Bits	Codewords
00	0	000
01	1	011
10	1	101
11	0	110

Minimum Hamming Distance

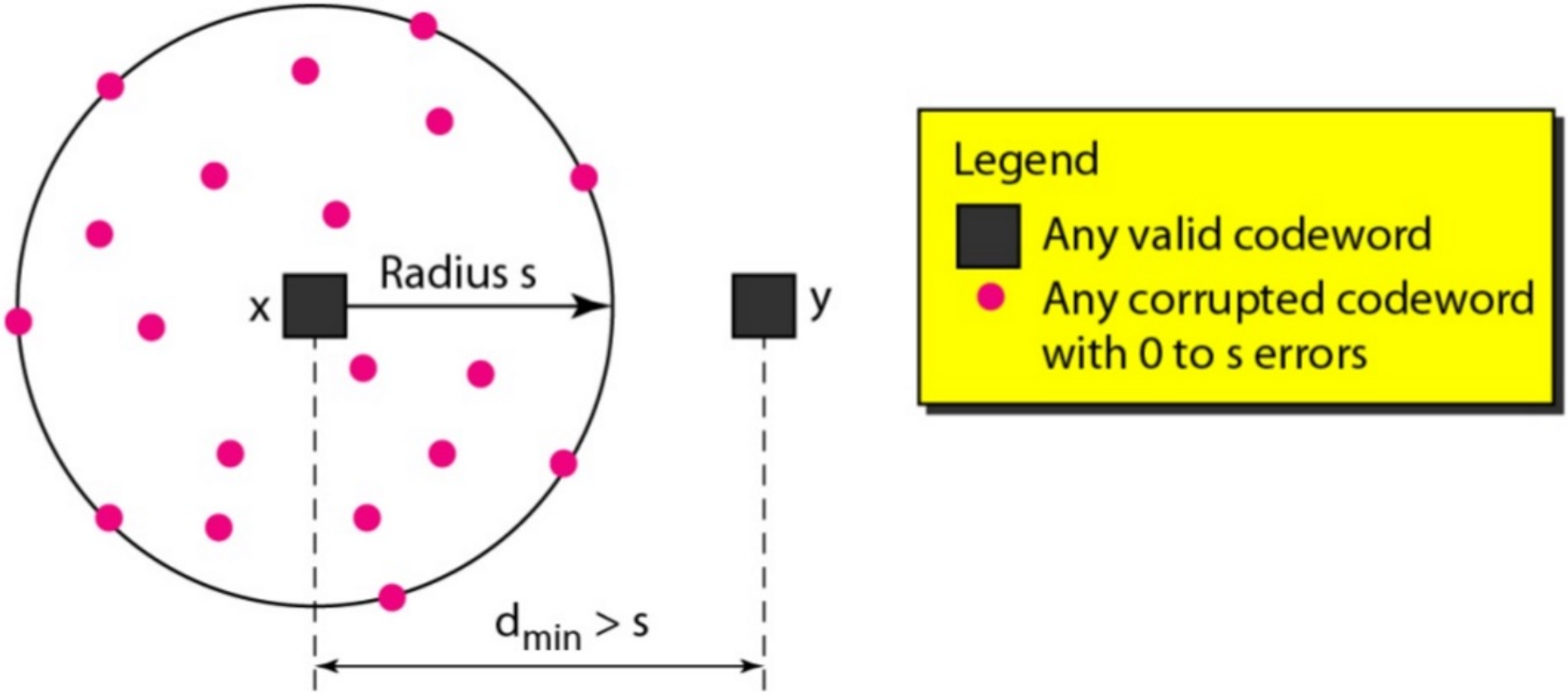
- $d(00000,01011) = 3, d(00000,10101) = 3$
- $d(00000,11110) = 4, d(01011,10101) = 4$
- $d(01011,11110) = 3, d(10101,11110) = 3$
- The minimum Hamming distance is 3
- $3 > 2 \Rightarrow$ This code is more powerful than the previous one?

Datawords	Additional Bits	Codewords
00	000	00000
01	011	01011
10	101	10101
11	110	11110

How to Detect Two-Bit Error

- Assume the sent codeword is c_s (must be valid)
- To detect two-bit error, any valid codeword c_v other than c_s must have $d(c_s, c_v) > 2$
- $d(c_s, c_v) \geq 3$, for any two different valid codewords c_s, c_v
- In other words, to detect two-bit error, the minimum Hamming distance between two valid codewords must be at least 3

Geometric Concept of Error Detection



Error Correction

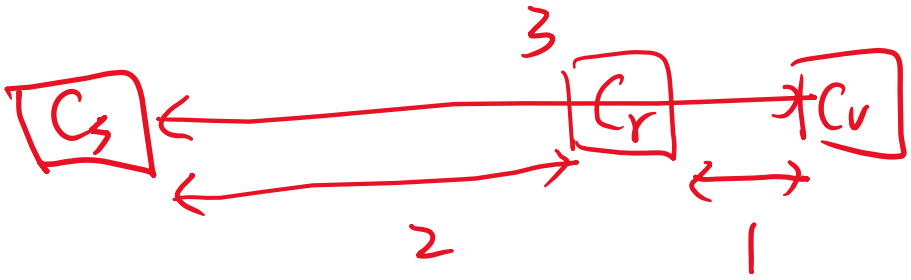
How to Correct One-Bit Error

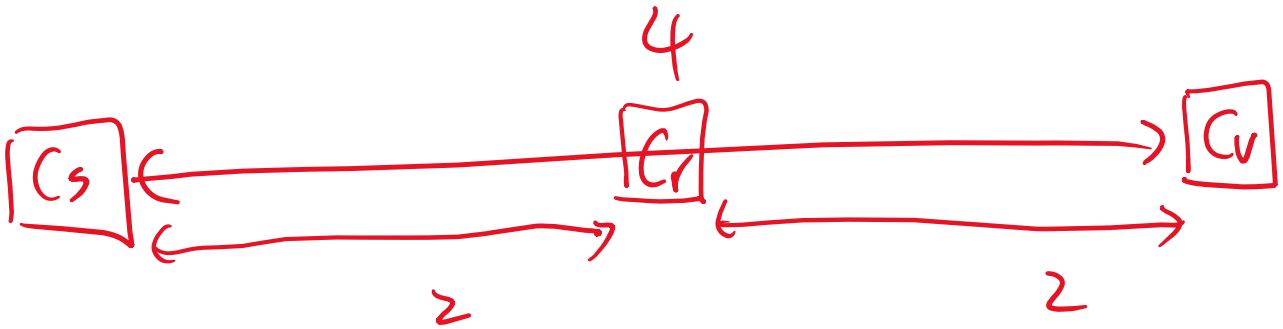
- Assume the sent codeword is c_s (must be valid)
- To correct one-bit error, any valid codeword c_v other than c_s cannot have $d(c_s, c_v) = 1$ or 2
- $d(c_s, c_v) \geq 3$, for any two different valid codewords c_s, c_v
- In other words, to correct one-bit error, the minimum Hamming distance between two valid codewords must be at least 3

Minimum Hamming Distance

- $d(00000,01011) = 3, d(00000,10101) = 3$
- $d(00000,11110) = 4, d(01011,10101) = 4$
- $d(01011,11110) = 3, d(10101,11110) = 3$
- The minimum Hamming distance is 3

Datawords	Additional Bits	Codewords
00	000	00000
01	011	01011
10	101	10101
11	110	11110

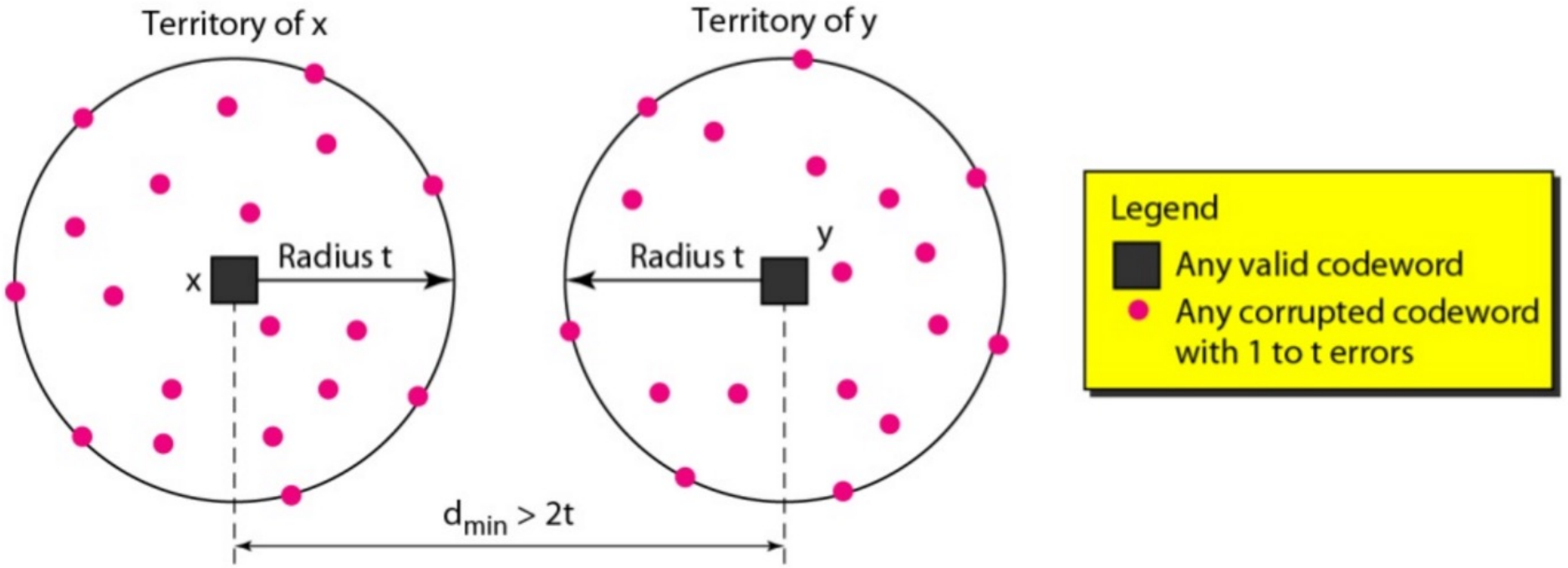

3
Cs
Cr
Cv
2
1


4
Cs
Cr
Cv
2
2

How to Correct Two-Bit Error

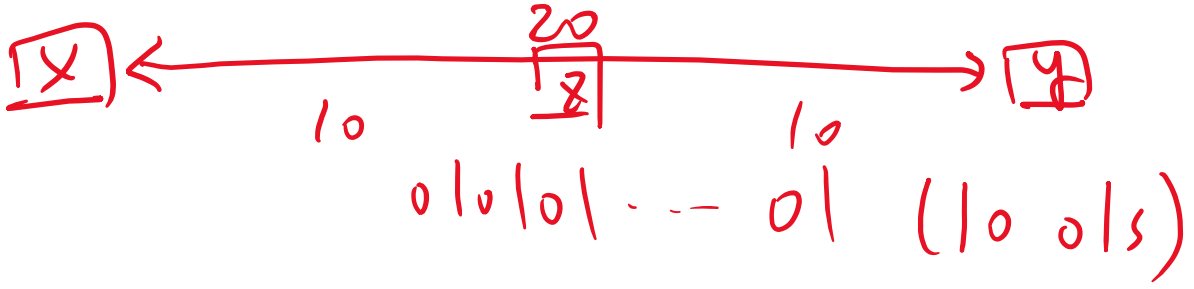
- Assume the sent codeword is c_s (must be valid)
- To correct two-bit error, any valid codeword c_v other than c_s cannot have $d(c_s, c_v) = 1,2,3 \text{ or } 4$
- Why not 3? Correction would be wrong
- Why not 4? There is a tie
- $d(c_s, c_v) \geq 5$, for any two different valid codewords c_s, c_v
- In other words, to correct one-bit error, the minimum Hamming distance between two valid codewords must be at least 5

Geometric Concept



Example

dataword		codeword
0	$\mapsto$	0000...0 (20 0s)
1	$\mapsto$	1111...1 (20 1s)



An Example

- A code scheme has a minimum Hamming distance 4. What is the error detection and correction capability of this scheme?
- It can detect up to 3 bit errors
- It can correct 1 bit error

To detect up to s bit errors,
the minimum Hamming distance
must be at least $s + 1$

To correct up to s bit errors,
the minimum Hamming distance
must be at least $2s + 1$

How to generate the code?

Modular-2 Arithmetic

- We can only use 1 or 0
- $0 + 0 = 0$ $\quad\quad$ $0 - 0 = 0$
- $0 + 1 = 1$ $\quad\quad$ $0 - 1 = 1$
- $1 + 0 = 1$ $\quad\quad$ $1 - 0 = 1$
- $1 + 1 = 0$ $\quad\quad$ $1 - 1 = 0$
- Addition is the same as subtraction
- The operation is identical to XOR

Simple Parity-Check Code

- One additional bit: parity bit
- Idea: make the number of 1s in the codeword even

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

Simple Parity-Check Code

- What is the minimum Hamming distance of it?
- If two datawords differ in 1 bit, then their parity bits must be different
- Hence, the minimum Hamming distance is 2

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

The Capability of Simple Parity-Check Code

- Since the minimum Hamming distance is 2, it can detect 1-bit error
- However, as you might already know, it can also detect odd number of bit errors
- How to detect errors?
- The receiver sums up (modulo-2) all the bits, the result is called the **syndrome**
- If the syndrome is 0, the receiver guesses that there is no error
- If the syndrome is 1, the receiver knows the codeword is corrupted

An Example

- Assume the sender sends dataword 1011
- The codeword is 10111
- Case 1: no error. The receiver receives 10111. The syndrome is 0. Dataword 1011 is created
- Case 2: 1 bit error. The receiver receives 10011. The syndrome is 1. No dataword is created
- Case 3: 1 bit error. The receiver receives 10110. The syndrome is 1. No dataword is created. However, none of the dataword bits is corrupted

An Example

- Assume the sender sends dataword 1011
- The codeword is 10111
- Case 4: Two bit errors. The receiver receives 00110. The syndrome is 0. Dataword 0011 is wrongly generated
- Case 5: Three bit errors: The receiver receives 01011. The syndrome is 1. No dataword is generated

Generalizations of Simple Parity-Check Code

- In simple parity-check code, we use only one parity bit to protect the dataword
- Can we use more parity bits?

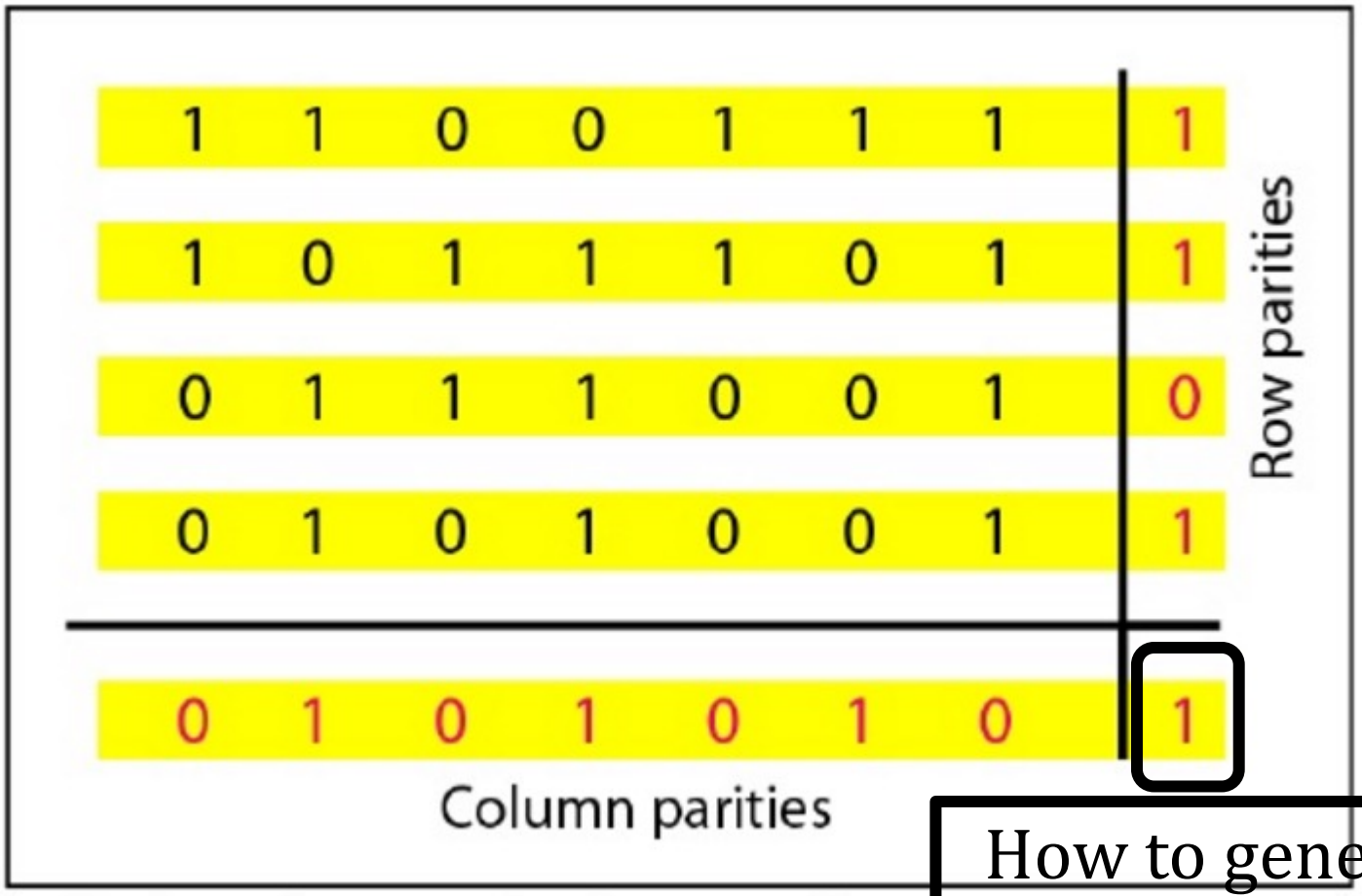
The First Generalization

- Idea: divide the dataword into small datawords
- For example:
- Dataword: 11001111011101011100101010 01
- Divide into:
- 1100111 1
- 1011101 1
- 0111001 0
- 0101001 1

0 1 0 1 0 1 0 1

- How many parity bits can you add?

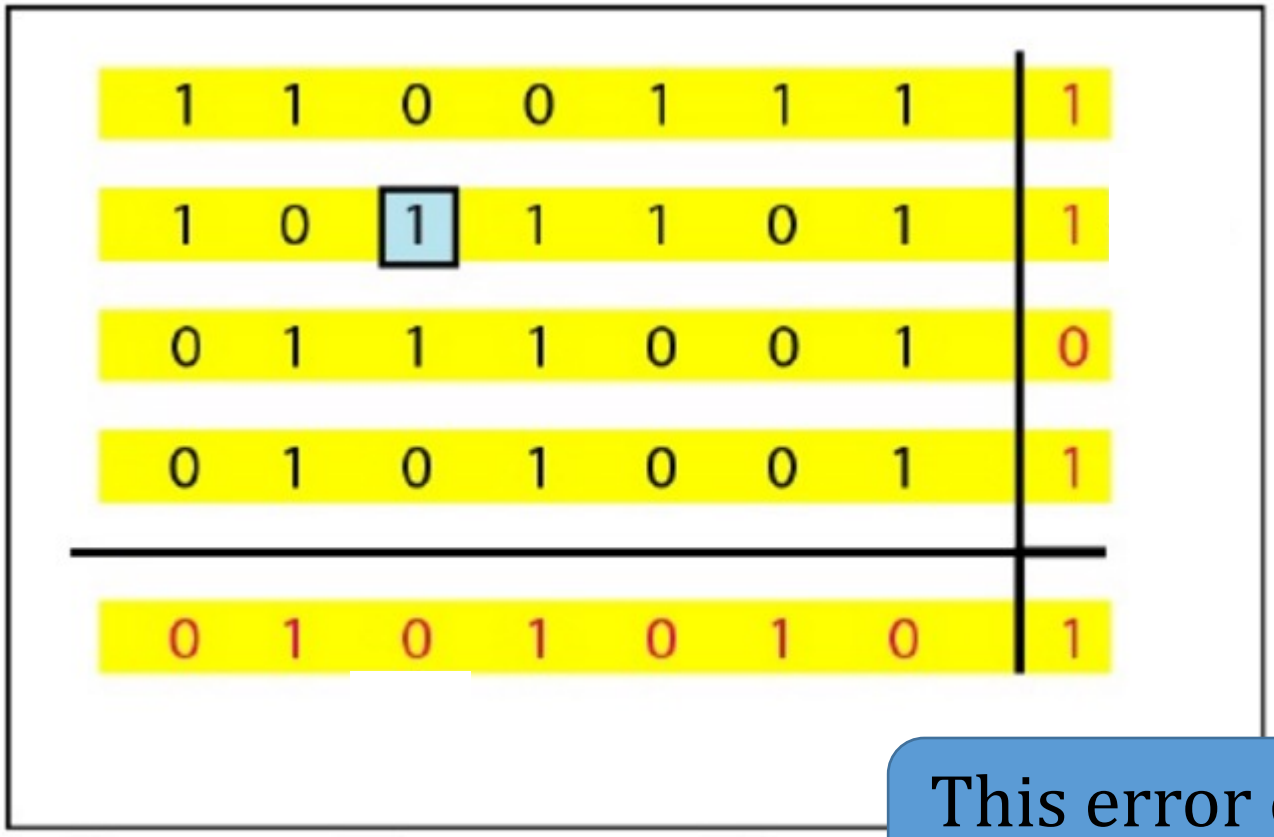
Two-Dimensional Parity Check



a. Design of row and column parities

How to generate this bit?

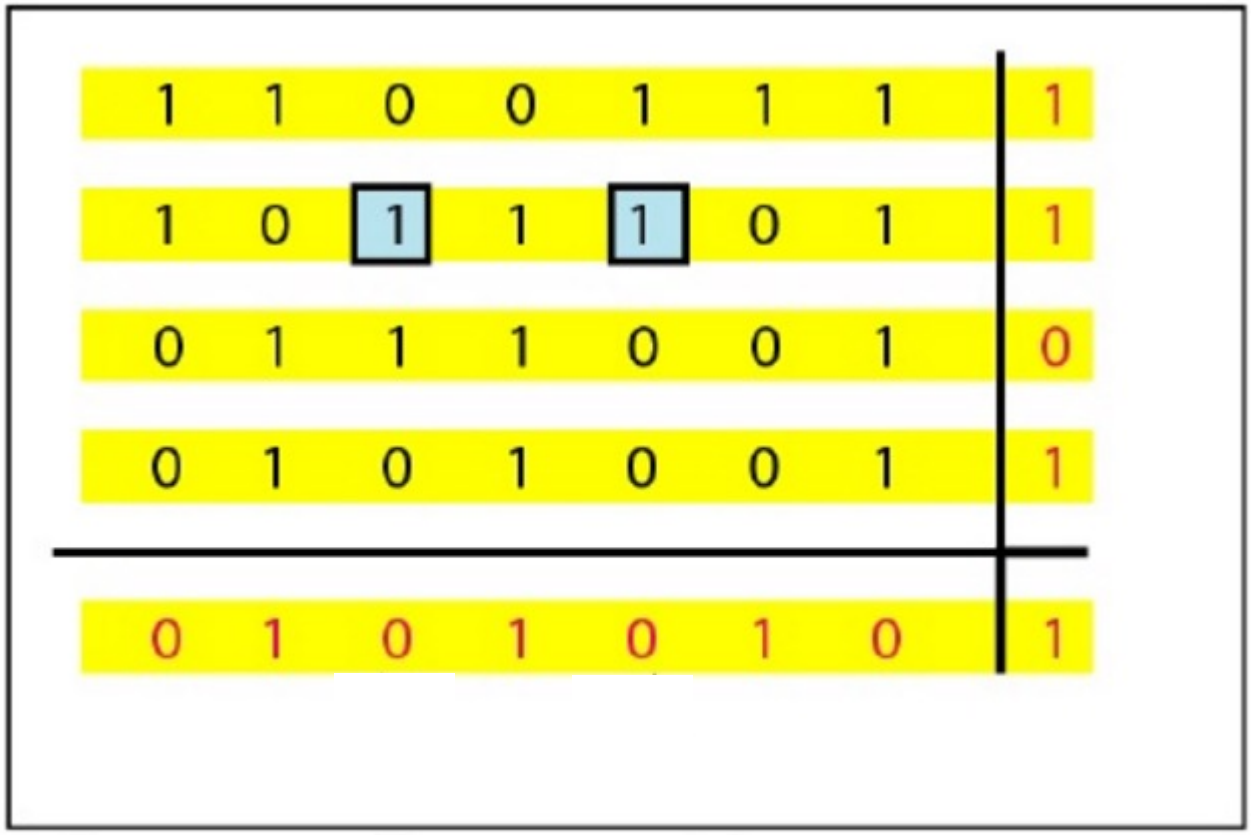
Two-Dimensional Parity Check



b. One error affects two parities

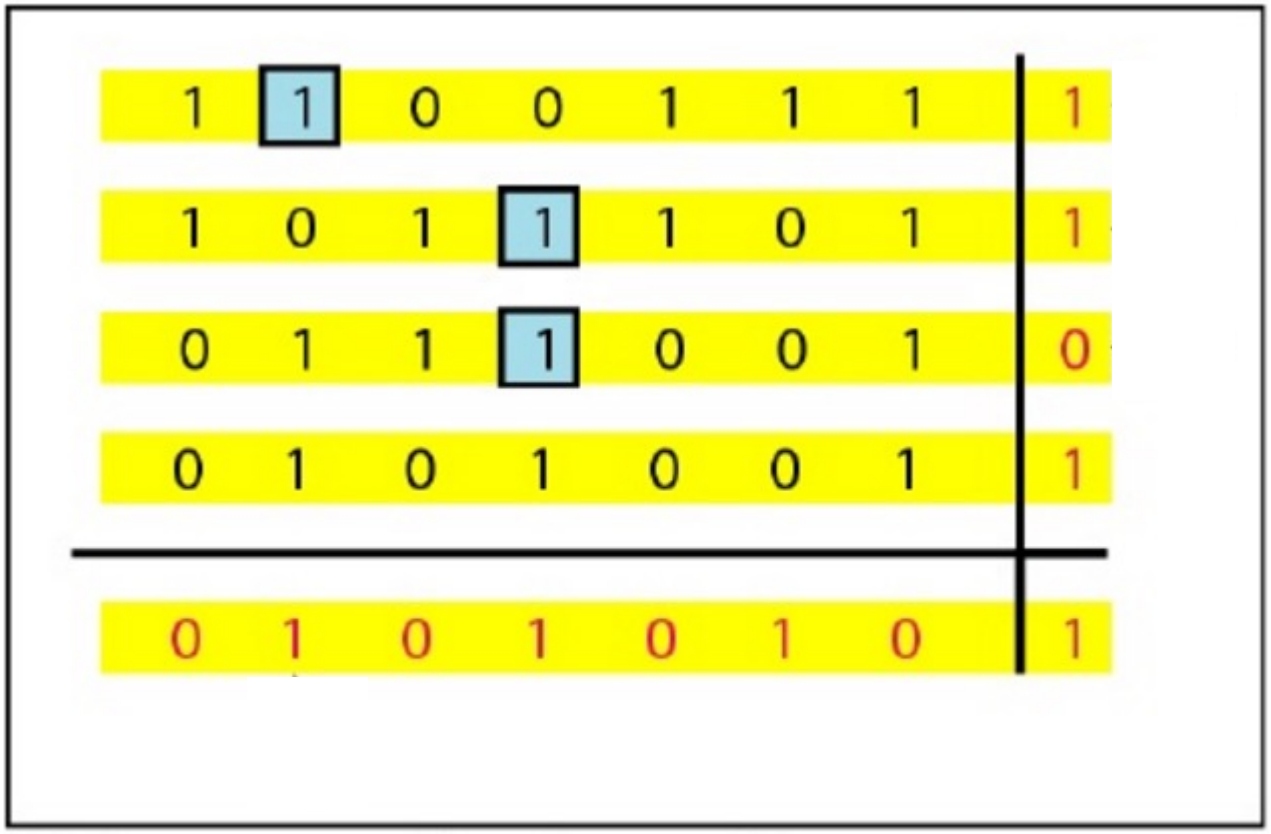
This error can be corrected. Why?

Two-Dimensional Parity Check



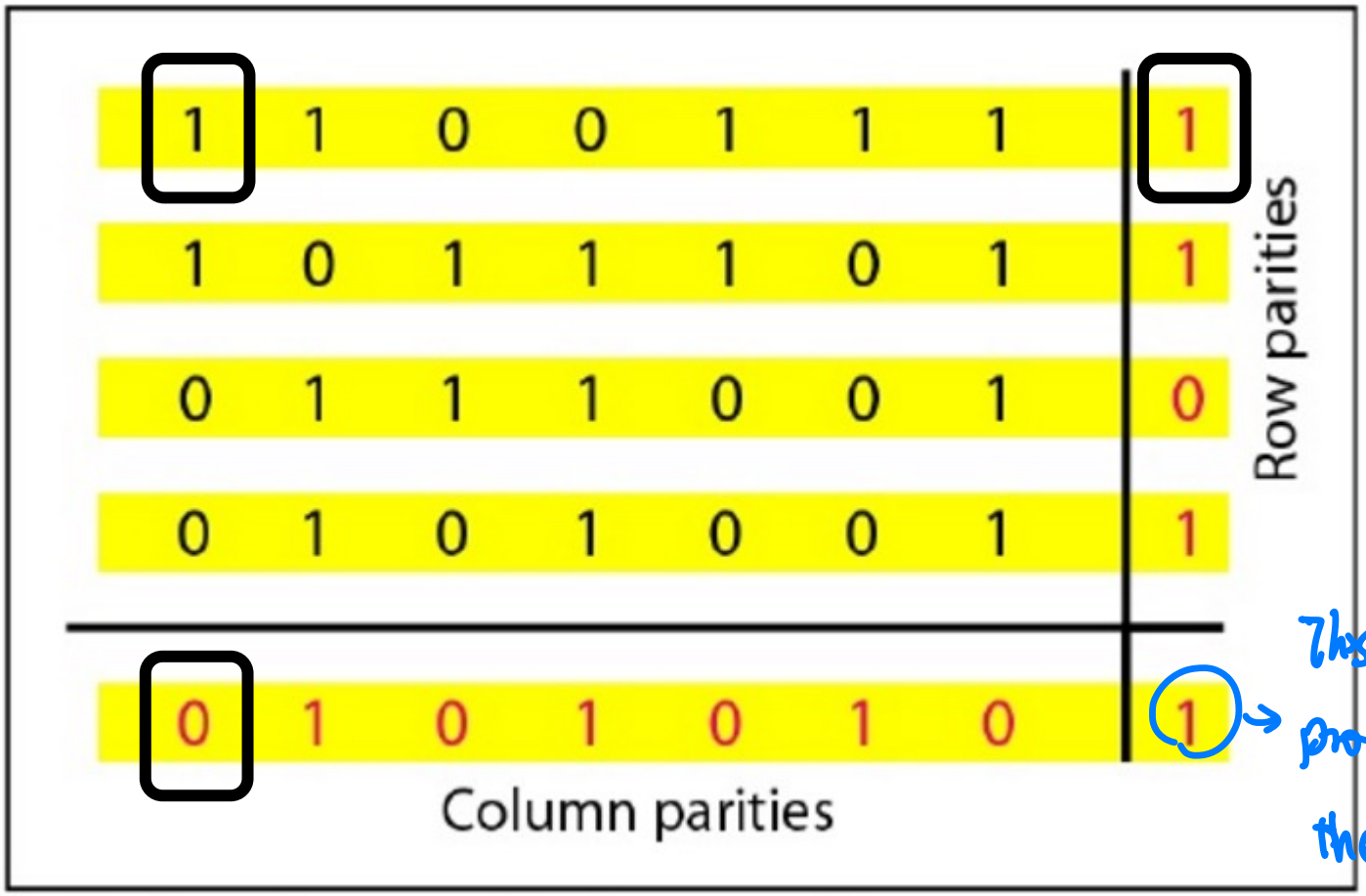
c. Two errors affect two parities

Two-Dimensional Parity Check



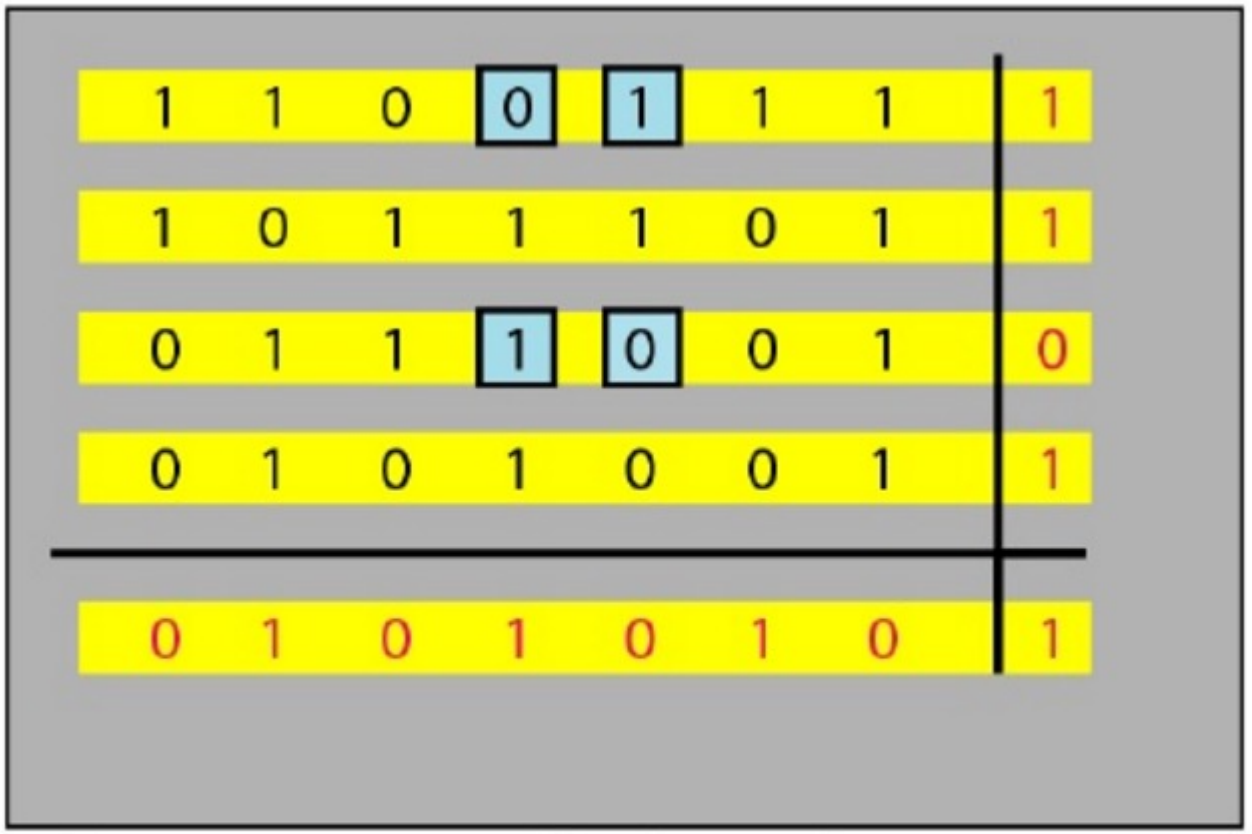
d. Three errors affect four parities

Two-Dimensional Parity Check



Can these three errors be detected? Yes

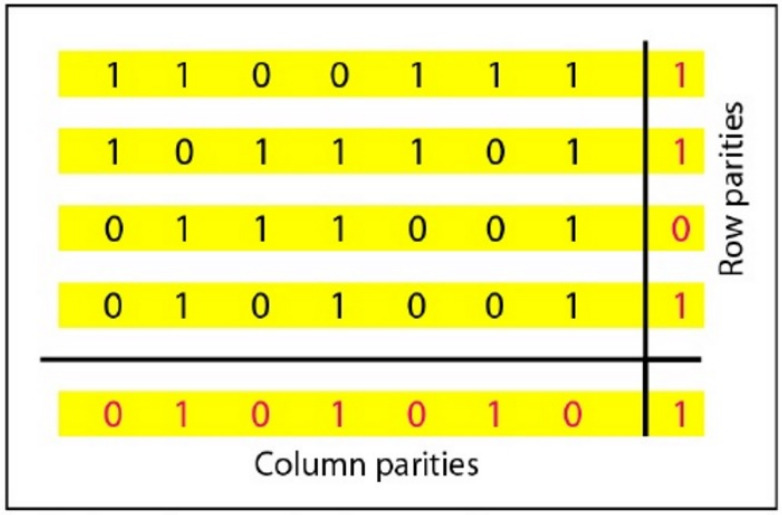
Two-Dimensional Parity Check



e. Four errors cannot be detected

Two-Dimensional Parity Check

- Each row is protected by a parity bit
- Each column is protected by a parity bit
- The parity bit row and parity bit column are protected by the corner parity bit



a. Design of row and column parities

Two-Dimensional Parity Check

4

- What is the minimum Hamming distance of the code?
- If two datawords differ in 1 bit, then the corresponding row parity and column parity must be different
- In addition, the corner parity bit is different
- 4 bits are different

Two-Dimensional Parity Check

- What is the minimum Hamming distance of the code?
- If two datawords differ in 2 bits, then at least 2 row parity bits or at least 2 column parity bits must be different
- 4 bits are different

Two-Dimensional Parity Check

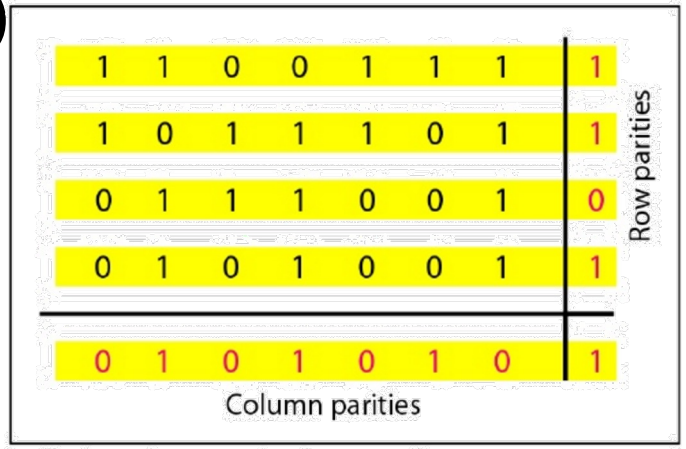
- What is the minimum Hamming distance of the code?
- If two datawords differ in 3 bits, then at least 3 parity bits are different
- Minimum Hamming distance of this code $= 4$

Two-Dimensional Parity Check

- Given 2 codewords c_1 and c_2, there are four cases
  1. c_1 and c_2 differ in 1 bits of the datawords. We just prove that this implies 3 parity bits are different
  2. c_1 and c_2 differ in 2 bits of the datawords. We just prove that this implies 2 parity bits are different
  3. c_1 and c_2 differ in 3 bits of the datawords. We just prove that this implies 3 parity bits are different
  4. c_1 and c_2 differ in at least 4 bits of the datawords.
- No matter which case happens, the Hamming distance between c_1 and c_2 is at least 4. Hence, 2D parity check code has Hamming distance 4

The Second Generalization

- In the previous code, we use one column parity bit and one row parity bit to locate the error
- We use 2 parity checks (bits) to locate the error
- If we have r parity checks (bits), we can locate $O(r^2)$ different 1-bit errors
- To locate k different 1-bit errors, we need $O(\sqrt{k})$ parity checks (bits)
- In the next code, we use less parity bits to locate the error



a. Design of row and column parities

Hamming Code

Hamming Code

- Assume the dataword has k bits
- Q: What is the minimum number of bits to represent k?
- A: $\log_2 k$ bits
- We add $O(\log_2 k)$ parity bits (checks)
- There are $2^{\log_2 k} = k$ different syndromes
- We may locate k different 1-bit errors by different syndromes
- We also need to consider errors on parity bits

Hamming Code

- An example of 4-bit dataword d_1, d_2, d_3, d_4
- Add 3 parity bits, p_1, p_2, p_3
- Hence, we have 7 bits in a codeword
- Since we have 3 parity checks, we have $2^3 = 8$ different syndromes
  - E.g., 1st & 3rd parity checks are correct, 2nd check is wrong
- Try to map syndromes to bit locations (1~7) or to indicate no error (0)
- Q: why 2 parity bits are insufficient?

Syndrome Mapping

- 3 parity bits, p_1, p_2, p_3
- How to generate the parity bits?
  - A parity bit is the sum of some bits in datawords
- Let c_1, c_2, c_3 be the parity check result (i.e., syndrome $=$ sum of the parity bit and the corresponding bits) with respect to p_1, p_2, p_3
- We want the following effect:
  - If $c_3 = 0, c_2 = 0, c_1 = 0$, then no error
  - If $c_3 = 0, c_2 = 0, c_1 = 1$, then the 1st bit corrupts
  - If $c_3 = 0, c_2 = 1, c_1 = 0$, then the 2nd bit corrupts
- $(c_3 c_2 c_1)_2$ indicates the location of 1-bit error

Generate Parity Bits

- $(c_3c_2c_1)_2$ indicates the location of the 1-bit error
- $c_1 = 1 \Leftrightarrow$ One of the $(001)_2$, $(011)_2$, $(101)_2$, or $(111)_2$ bits is wrong (1^{st}, 3^{rd}, 5^{th}, 7^{th} bits)

- $c_1 = 1^{st}$ bit in codeword + 3^{rd} bit in codeword + 5^{th} bit in codeword + 7^{th} bit in codeword
- One of the 1^{st}, 3^{rd}, 5^{th}, 7^{th} bits is p_1
- p_1 is the sum of the rest three bits
- The rest three bits are data bits

Generate Parity Bits

- $(c_3 c_2 c_1)_2$ indicates the location of the 1-bit error
- $c_2 = 1 \Leftrightarrow$One of the $(0\mathbf{1}0)_2$, $(0\mathbf{1}1)_2$, $(1\mathbf{1}0)_2$, or $(1\mathbf{1}1)_2$ bits is wrong (2^{nd}, 3^{rd}, 6^{th}, 7^{th} bits)
- $c_2 = 2^{\text{nd}}$ bit in codeword + 3^{rd} bit in codeword + 6^{th} bit in codeword + 7^{th} bit in codeword
- One of the 2^{nd}, 3^{rd}, 6^{th}, 7^{th} bits is p_2
- p_2 is the sum of the rest three bits
- The rest three bits are data bits

Generate Parity Bits

- $(c_3c_2c_1)_2$ indicates the location of the 1-bit error
- $c_3 = 1 \Leftrightarrow$One of the $(100)_2$, $(101)_2$, $(110)_2$, or $(111)_2$ bits is wrong (4^{th}, 5^{th}, 6^{th}, 7^{th} bits)
- $c_3 = 4^{\text{th}}$ bit in codeword + 5^{th} bit in codeword + 6^{th} bit in codeword + 7^{th} bit in codeword
- One of the 4^{th}, 5^{th}, 6^{th}, 7^{th} bits is p_3
- p_3 is the sum of the rest three bits
- The rest three bits are data bits

Generate Parity Bits

- One of the 1^{st}, 3^{rd}, 5^{th}, 7^{th} bits is p_1
- One of the 2^{nd}, 3^{rd}, 6^{th}, 7^{th} bits is p_2
- One of the 4^{th}, 5^{th}, 6^{th}, 7^{th} bits is p_3
- Parity bit = sum of some data bits
- Where to place the parity bits?
- 1^{st} bit in the codeword: p_1
- 2^{nd} bit in the codeword: p_2
- 4^{th} bit in the codeword: p_3
- The 3^{rd}, 5^{th}, 6^{th}, and 7^{th} bits are data bits

Hamming Code

- Codeword = $p_1 p_2 d_1 p_3 d_2 d_3 d_4$

Bit #	1	2	3	4	5	6	7
Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

- For example, if c_1 and c_2 indicate errors, and we know it is an one-bit error, then d_1 is wrong

Hamming Code

- Dataword $= 0000$
- $p_1 = d_1 + d_2 + d_4 = 0$
- $p_2 = d_1 + d_3 + d_4 = 0$
- $p_3 = d_2 + d_3 + d_4 = 0$
- Codeword $= p_1 p_2 d_1 p_3 d_2 d_3 d_4 = 0000000$

Bit #	1	2	3	4	5	6	7
Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

Hamming Code

- Dataword = 0001
- $p_1 = d_1 + d_2 + d_4 = 1$
- $p_2 = d_1 + d_3 + d_4 = 1$
- $p_3 = d_2 + d_3 + d_4 = 1$
- Codeword = $p_1 p_2 d_1 p_3 d_2 d_3 d_4$=1101001

Bit #	1	2	3	4	5	6	7
Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

Hamming Code

- Dataword $= 0010$
- $p_1 = d_1 + d_2 + d_4 = 0$
- $p_2 = d_1 + d_3 + d_4 = 1$
- $p_3 = d_2 + d_3 + d_4 = 1$
- Codeword $= p_1 p_2 d_1 p_3 d_2 d_3 d_4 = 0101010$

Bit #	1	2	3	4	5	6	7
Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

Hamming Code

- Dataword = 0011
- $p_1 = d_1 + d_2 + d_4 = 1$
- $p_2 = d_1 + d_3 + d_4 = 0$
- $p_3 = d_2 + d_3 + d_4 = 0$
- Codeword = $p_1 p_2 d_1 p_3 d_2 d_3 d_4 = 1000011$

Bit #	1	2	3	4	5	6	7
Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

Hamming Code

Datawords	Codewords	Datawords	Codewords
0000	0000000	1000	1110000
0001	1101001	1001	0011001
0010	0101010	1010	1011010
0011	1000011	1011	0110011
0100	1001100	1100	0111100
0101	0100101	1101	1010101
0110	1100110	1110	0010110
0111	0001111	1111	1111111

Hamming Code

- If we send 1101001, and the receiver receives 1001001
- Parity check 1: 1+0+0+1=0 (OK)
- Parity check 2: 0+0+0+1=1 (Fail)
- Parity check 3: 1+0+0+1=0 (OK)
- The 010=2nd bit is wrong

Bit #	1	2	3	4	5	6	7
Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

Hamming Code

- If we send 1010101, and the receiver receives 1011101
- Parity check 1: 1+1+1+1=0 (OK)
- Parity check 2: 0+1+0+1=0 (OK)
- Parity check 3: 1+1+0+1=1 (Fail)
- The 100=4th bit is wrong

Bit #	1	2	3	4	5	6	7
Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

Hamming Code

- If we send 1010101, and the receiver receives 1000011
- Parity check 1: 1+0+0+1=0 (OK)
- Parity check 2: 0+0+1+1=0 (OK)
- Parity check 3: 0+0+1+1=0 (OK)
- No bit error is detected

Bit #	1	2	3	4	5	6	7
Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

Hamming Code

- If we send 1010101, and the receiver receives 1010100
- Parity check 1: 1+1+1+0=1 (Fail)
- Parity check 2: 0+1+0+0=1 (Fail)
- Parity check 3: 1+1+0+1=1 (Fail)
- The 111=7th bit is wrong

Bit #	1	2	3	4	5	6	7
Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

Hamming Code

- If the dataword has 1 bit, then how many parity bits do we need?
- 1 parity bit is not enough
  - There are 2 bits in the codeword
  - We need 2 bits to specify the location of error or indicate no error
- 2 parity bits are enough
  - There are 3 bits in the codeword
  - We need 2 bits to specify the location of error or indicate no error

Hamming Code

- Dataword: d_1
- Parity bits: p_1, p_2
- If $c_2 = 0, c_1 = 0$, then no error
- If $c_2 = 0, c_1 = 1$, then the 1^{st} bit corrupts
- If $c_2 = 1, c_1 = 0$, then the 2^{nd} bit corrupts
- If $c_2 = 1, c_1 = 1$, then the 3^{rd} bit corrupts
- $(c_2 c_1)_2$ indicates the location of the 1-bit error

Generate Parity Bits

- $(c_2 c_1)_2$ indicates the location of the 1-bit error
- $c_1 = 1 \Leftrightarrow$One of the $(01)_2$ or $(11)_2$ bits is wrong (1^{st}, 3^{rd} bits)
- $c_1 =$ 1^{st} bit in codeword + 3^{rd} bit in codeword
- One of the 1^{st}, 3^{rd} bits is p_1
- p_1 is the sum of the rest one bit
- The rest bit is a data bit

Generate Parity Bits

- $(c_2 c_1)_2$ indicates the location of the 1-bit error
- $c_2 = 1 \Leftrightarrow$One of the $(10)_2$ or $(11)_2$ bits is wrong (2^{nd}, 3^{rd} bits)
- $c_2 = 2^{\text{nd}}$ bit in codeword + 3^{rd} bit in codeword
- One of the 2^{nd}, 3^{rd} bits is p_2
- p_2 is the sum of the rest one bit
- The rest bit is a data bit

Generate Parity Bits

- One of the 1^{st}, 3^{rd} bits is p_1
- One of the 2^{nd}, 3^{rd} bits is p_2
- Parity bit = sum of some data bits
- 1^{st} bit in the codeword: p_1
- 2^{nd} bit in the codeword: p_2
- The 3^{rd} bit is a data bit

Hamming Code

- Dataword = 0
- $p_1 = d_1 = 0$
- $p_2 = d_1 = 0$
- Codeword = $p_1 p_2 d_1 = 000$

- Dataword = 1
- $p_1 = d_1 = 1$
- $p_2 = d_1 = 1$
- Codeword = $p_1 p_2 d_1 = 111$

Hamming Code

- If the dataword has 11 bits, how many parity bits do we need?
- 4 parity bits are enough
  - There are 15 bits in the codeword
  - We need 4 bits to specify 15 locations of error + (no error)
- If the dataword has 12 bits, then how many parity bits do we need?
- 5 parity bits are enough

1 0 1 1

	p_1	p_2		p_3				p_4							
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			d_1		d_2	d_3	d_4		d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}
p_1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
p_2	0										1				
p_3	0										0				
p_4	0							1	1	1	1	1	1	1	1

Hamming Distance

- Each data bit is protected by at least 2 parity bits
  - As long as the location of each data bit is not a power of 2, each data bit is protected by at least 2 parity bits
- Hence, 1-data-bit difference leads to 2-parity-bit difference

$1\,1\,0\,1 \Rightarrow p_1 = 1,\ p_3 = 0$

$1\,0\,0\,1 \Rightarrow p_1 = 0,\ p_3 = 1$

Bit #	1	2	3	4	5	6	7
Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

Minimum Hamming Distance

- Example: If 2 datawords differ in 1 bit, at least 2 parity bits are different
  - $d_1 d_2 d_3 d_4$
  - $d_1' d_2 d_3 d_4$
  - p_1 for $d_1 d_2 d_3 d_4 \neq p_1$ for $d_1' d_2 d_3 d_4$
  - p_2 for $d_1 d_2 d_3 d_4 \neq p_2$ for $d_1' d_2 d_3 d_4$

Bit #	1	2	3	4	5	6	7
Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

Hamming Distance

- Diff. data bits are protected by diff. sets of parity bits
- 2-data-bit difference leads to 1-parity-bit difference

Minimum Hamming Distance

- If 2 datawords differ in 2 bits, at least 1 parity bit is different
  - $d_1 d_2 d_3 d_4$
  - $d_1 d_2 d'_3 d'_4$
  - p_1 for $d_1 d_2 d_3 d_4$ $\neq$ p_1 for $d_1 d_2 d'_3 d'_4$

Bit #	1	2	3	4	5	6	7
Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

Minimum Hamming Distance

- If 2 datawords differ in 1 bit, at least 2 parity bits are different
- If 2 datawords differ in 2 bits, at least 1 parity bit is different
- The Hamming Distance of Hamming Code is 3
- What if 2-bit error occurs? E.g., 3rd and 5th bits are corrupted?

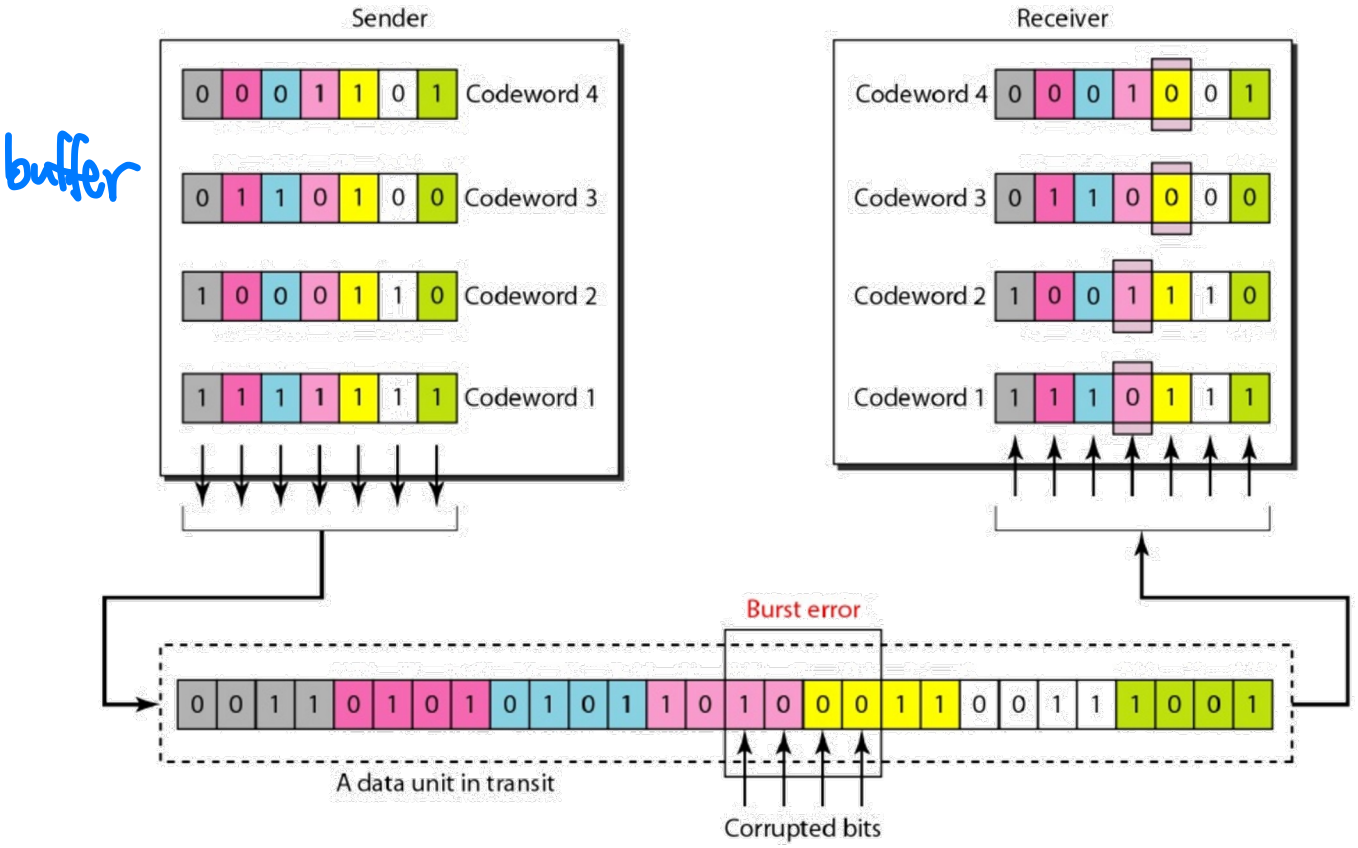
Bit #	1	2	3	4	5	6	7
Transmitted bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

How to Overcome Burst Error?

latency ↑

memory usage ↑

- Interleaving



Richard Hamming

- Born February 11, 1915
- Hamming code and Hamming distance first appear in 1950
  - Hamming, Richard W. (1950), "Error detecting and error correcting codes"
- Turing Award winner (1968)
- Died January 7, 1998 (aged 82)

Next, we will discuss a code that can detect burst error without interleaving

Intuition

- The sender and receiver agree upon a special large number g, e.g., 971
- Dataword: d, e.g., 109238
- Codeword $= (d, r = d\%g)$, e. g. , 109238, 486
- Receiver checks whether $(d - r)\%g = 0$
  - Yes ⇒ no error
  - No ⇒ error
- Intuitively, if the codeword is corrupted, so that the receiver receives (d', r'), it is very unlikely that $(d' - r')\%g = 0$ (given g is large enough)

Polynomial

- For ease of presentation we use polynomial to represent a binary code
- For example: $11000101 \Rightarrow x^7 + x^6 + x^2 + 1$
- The arithmetic on coefficients is modulo-2
- $x^3 + x^2 + x^2 + 1 = x^3 + 1$
- $(x^5 + x^3 + x^2 + x)(x^2 + x + 1)$

$= x^7 + x^6 + x^5 + x^5 + x^4 + x^3 + x^4 + x^3 + x^2 + x^3 + x^2 + x$

$= x^7 + x^6 + x^3 + x$

- $f(x) + f(x) =?$ 0

Cyclic Redundancy Check (CRC)

- When the receiver receives a codeword, the receiver calculates the remainder of $\frac{\text{codeword}}{g(x)}$
- If it is 0 ⇒ guess no error
- It it is not 0 ⇒ error

Cyclic Redundancy Check (CRC)

- Input:

1. Dataword (k bits): $d(x) = \sum_{i=0}^{k-1} d_i x^i$
2. Generator (s bits): $g(x) = \sum_{i=0}^{s-1} g_i x^i$

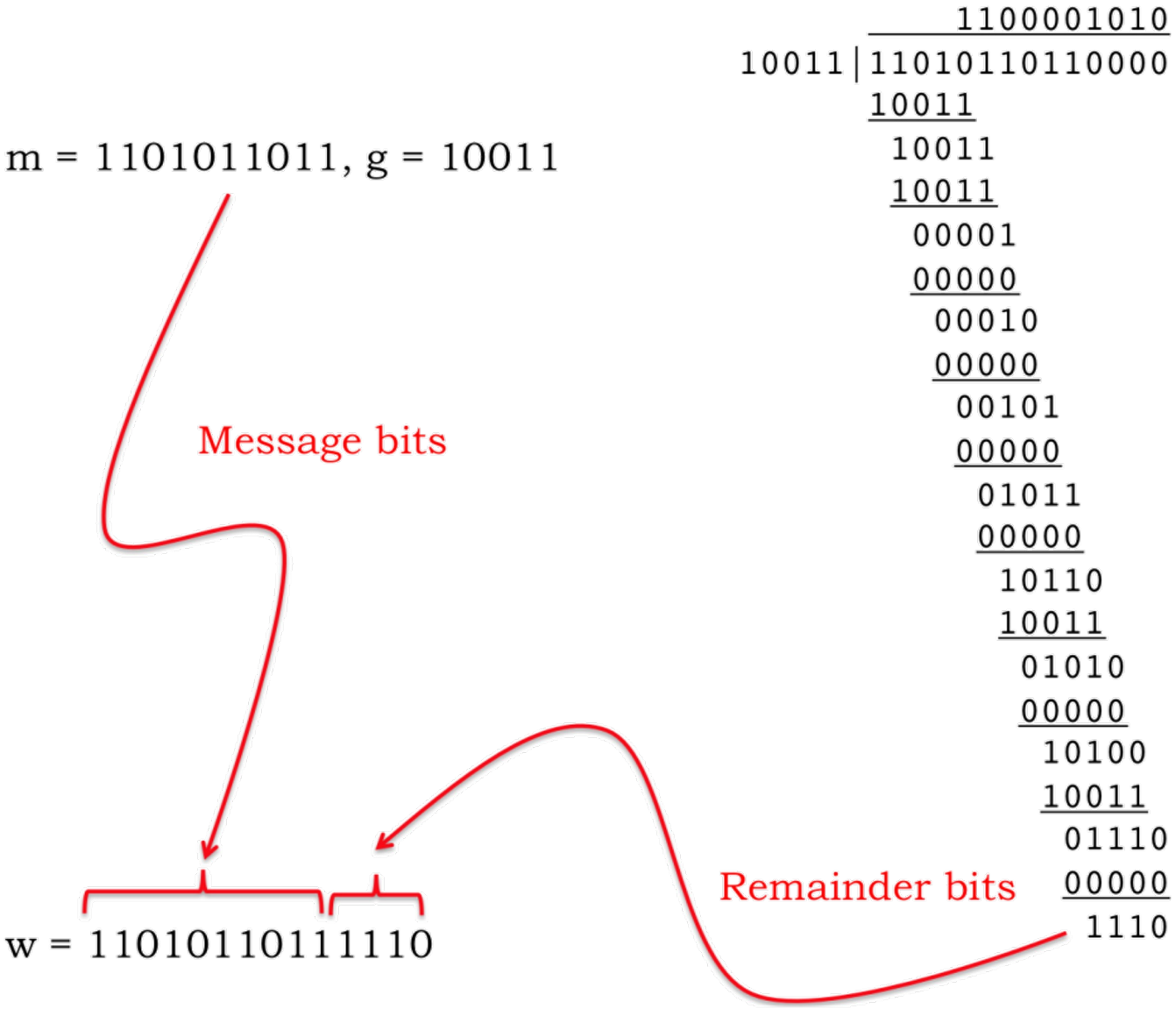
- Remainder ($s-1$ bits): $r(x) = \sum_{i=0}^{s-2} r_i x^i$
- Output:
- Codeword: $c(x) = d(x)x^{s-1} + r(x)$
  - Addition = subtraction in modulo-2 arithmetic

An Example

- $c(x) = d(x)x^{s-1} + \text{Remainder}\{d(x)x^{s-1}/g(x)\}$
- Example:
- $d = 11010111011$
- $g = 10011$
- $s = 5$
- We need to find the remainder of $11010111011\mathbf{0000}/10011$

An Example

m = 1101011011, g = 10011



Cyclic Redundancy Check (CRC)

- $c(x) = d(x)x^{s-1} + \text{Remainder}\{d(x)x^{s-1}/g(x)\}$
- $\frac{c(x)}{g(x)} = \frac{d(x)x^{s-1}}{g(x)} + \frac{\text{Remainder}\{d(x)x^{s-1}/g(x)\}}{g(x)}$

Remainder $= \text{Remainder}\left\{\frac{d(x)x^{s-1}}{g(x)}\right\}$

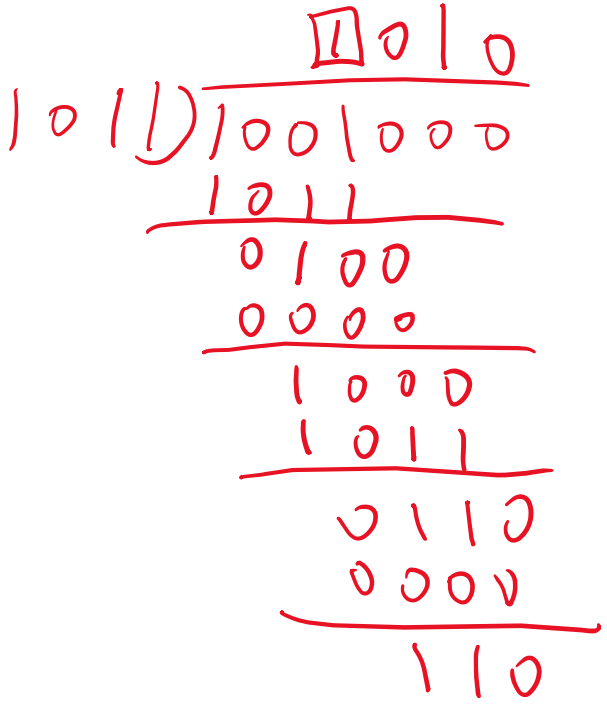
Remainder $= \text{Remainder}\left\{\frac{d(x)x^{s-1}}{g(x)}\right\}$

Remainder $= \text{Remainder}\left\{\frac{d(x)x^{s-1}}{g(x)}\right\} + \text{Remainder}\left\{\frac{d(x)x^{s-1}}{g(x)}\right\} = 0$

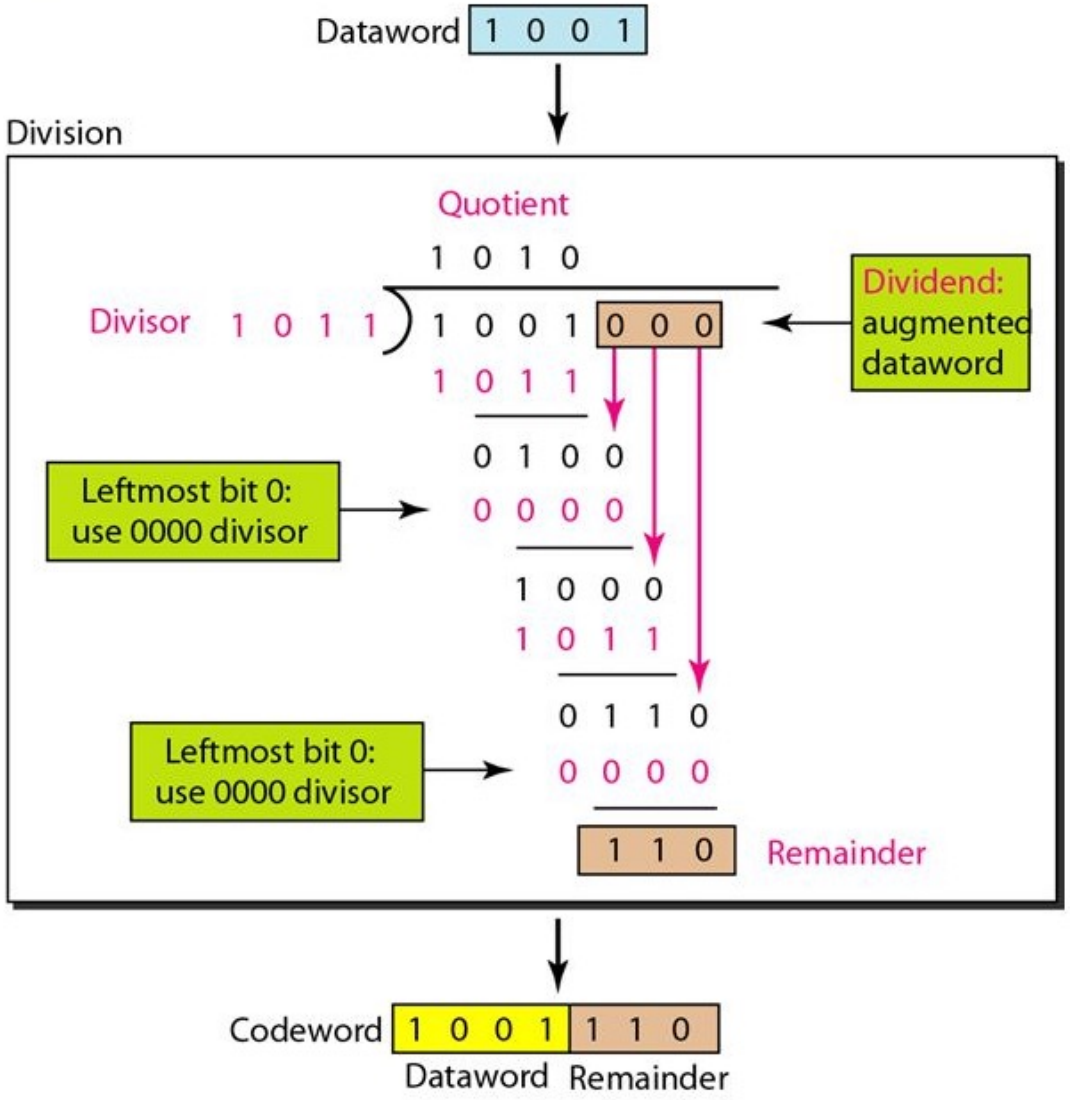
(modulo $-$ 2 arithmetic)

An Example

- $c(x) = d(x)x^{s-1} + \text{Remainder}\{d(x)x^{s-1}/g(x)\}$
- Example:
- $d = 1001 \Rightarrow d(x) = x^3 + 1$
- $g = 1011 \Rightarrow g(x) = x^3 + x + 1$
- $s = 4$
- We need to find the remainder of $(x^6 + x^3)/(x^3 + x + 1)$



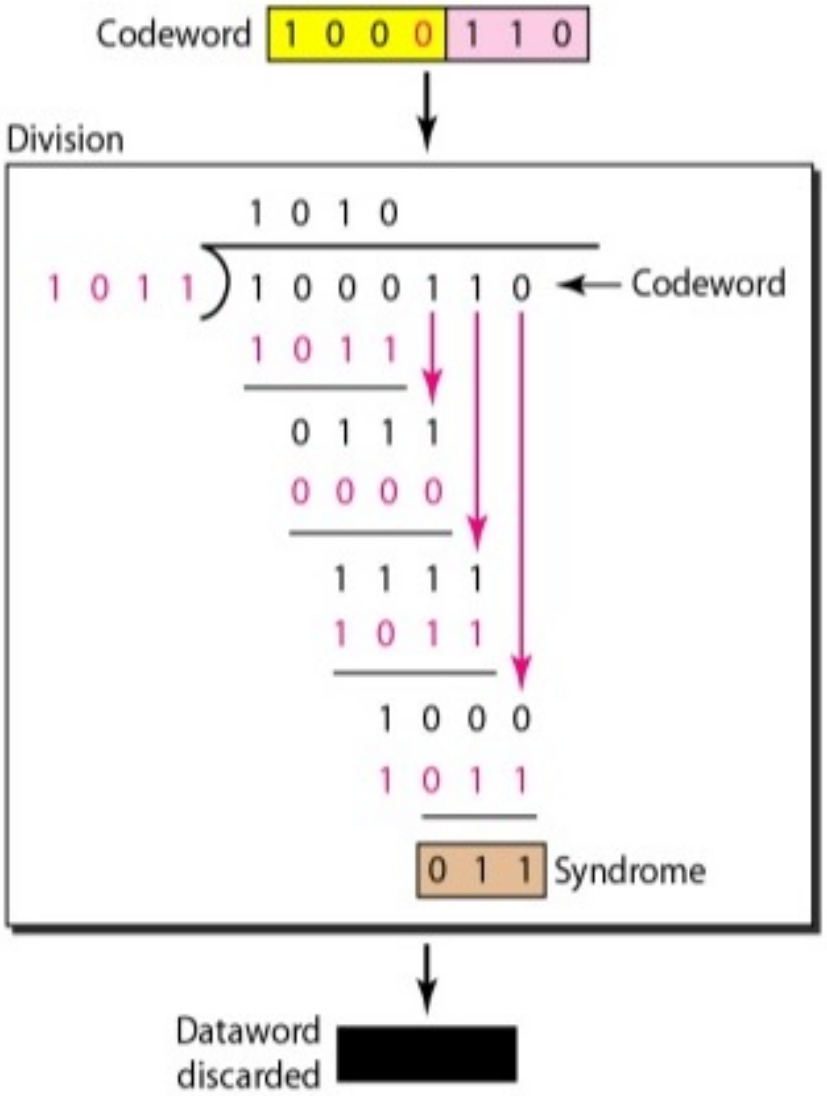
An Example


Dataword 1 0 0 1
Division
Quotient
1 0 1 0
Divisor 1 0 1 1
1 0 0 1 0 0 0
Dividend: augmented dataword
1 0 1 1
0 1 0 0
Leftmost bit 0: use 0000 divisor
0 0 0 0
1 0 0 0
1 0 1 1
0 1 1 0
Leftmost bit 0: use 0000 divisor
0 0 0 0
1 1 0
Remainder
Codeword 1 0 0 1 1 1 0
Dataword Remainder

An Example

- $d = 1001 \Rightarrow d(x) = x^3 + 1$
- $g = 1011 \Rightarrow g(x) = x^3 + x + 1$
- $s = 4$
- $c = 1001110 \Rightarrow c(x) = x^6 + x^3 + x^2 + x^1$
- Let's see what happens when the codeword is corrupted, so that the receiver receives 1000110

An Example


Codeword 1 0 0 0 1 1 0
Division
1 0 1 0
1 0 1 1 | 1 0 0 0 1 1 0 ← Codeword
1 0 1 1
0 1 1 1
0 0 0 0
1 1 1 1
1 0 1 1
1 0 0 0
1 0 1 1
0 1 1 Syndrome
Dataword discarded

The Design of Generators

goal: noise is not a multiple of generator

$1\,0\,0\,1\,1\,1\,0$ → codeword

$1\,0\,1\,1$ → generator

noise $= x^4 + x^3 + x + 1$

bit errors: $1\,0\,1\,1\,1\,0 \Rightarrow x^5 + x^3 + x^2 + x$

$\Rightarrow 1\,1\,0\,1\,0\,1 \Rightarrow x^5 + x^4 + x^2 + 1$ ↙ $+ x^4 + x^3 + x + 1$

noise is also a polynomial

rec. codeword

= sent codeword + noise

Odd number of bit errors

✓ ✓ ✓

1 0 1 1 1 0 ⇒ sent codeword

⇒ 1 1 0 1 0 0 ⇒ rec. codeword

3 bit errors ⇒ 3 terms in the noise's polynomial

noise = $x^4 + x^3 + x$

The Design of Generators

- Can we set $g(x) = 1$?
- Let $e(x)$ be the noise so that the receiver receives $c(x) + e(x)$
- The Key: $g(x)$ cannot be a factor of $e(x)$
- Singe bit error: $e(x) = x^i$ for some $0 \leq i \leq n - 1$
- Q1: What should $g(x)$ be to detect single bit error?
- A1: $g(x)$ should have at least 2 terms, e.g., $x^2 + 1$

The Design of Generators

- Odd number of errors, e.g., $e(x) = x^3 + x^2 + x$
- Q2: What should $g(x)$ be to detect odd number of errors?
- A2: $g(x)$ should have the factor $x + 1$. Why?
- We will show that any multiple of $x + 1$ has even number of terms
- Hence, $e(x)$, which has an odd number of terms, cannot be a multiple of $x + 1$
- Thus, $x + 1$ is not a factor of $e(x)$, and we can detect $e(x)$

Detect odd number of bit errors

$g(x)$ should have an (even) number of terms.

proof: We will show that $e(x)$ cannot be a multiple of $g(x)$

$e(x)$

↓

odd number of terms

when $x=1$

$g(x)$

↓

even # of terms

$g(x) = 0$

$e(x) = 1$

$\Rightarrow$ $e(x)$ cannot be a multiple of $g(x)$

The Design of Generators

- Any multiple of $x + 1$ has even number of terms
- Proof:
- $f(x)$ has even number of terms $\Leftrightarrow f(1) = 0$

- Any multiple of $x + 1$ can be represented as $(x + 1)h(x)$ for some $h(x)$
- $(1 + 1)h(1) = 0h(1) = 0$
- $(x + 1)h(x)$ has even number of terms

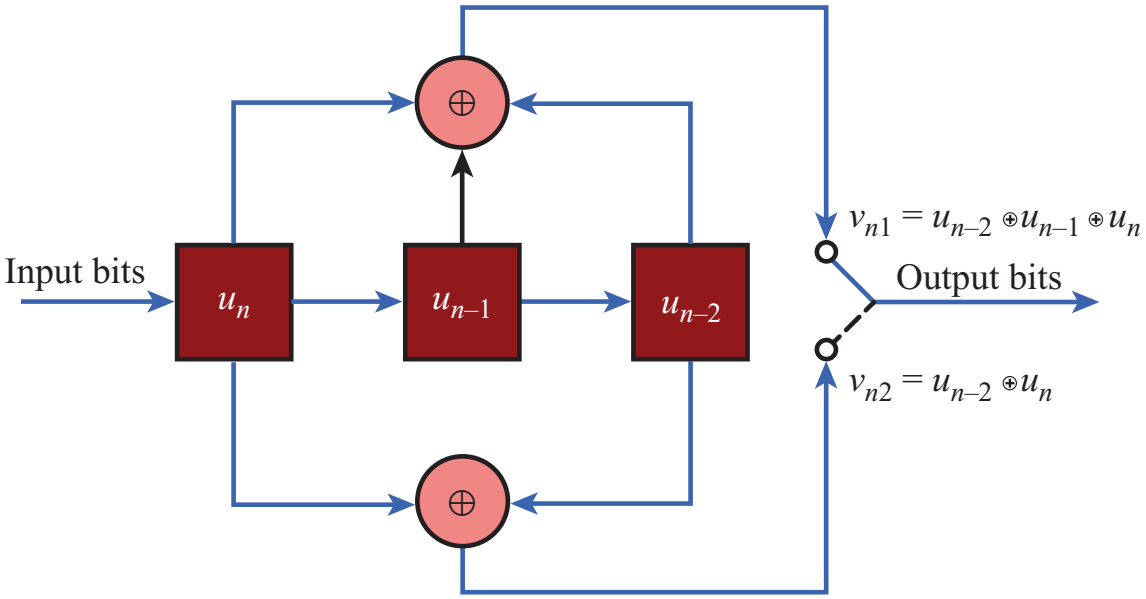
The Design of Generators

length of burst error

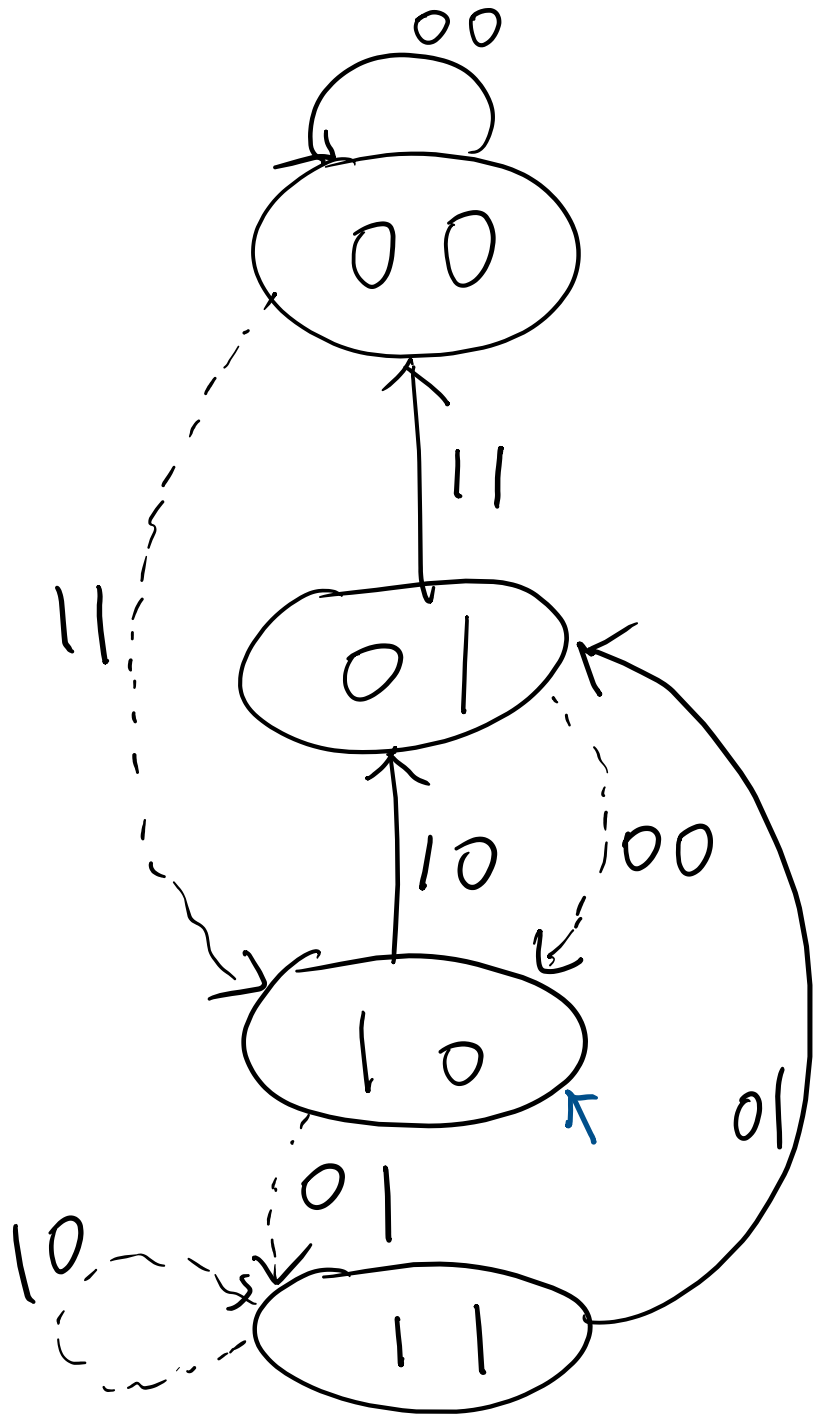
- Burst error, $e(x) = x^j + \cdots + x^i$,
- For example, $e(x) = x^{10} + x^8 + x^4 + x^2$
- Q3: What should $g(x)$ be to detect burst errors?
- We can write $e(x)$ as $x^i(x^{j-i} + \cdots + 1)$
- We do not need to worry x^i, because $g(x)$ can detect single bit error
- To detect $x^{j-i} + \cdots + 1$, we need to make sure that the remainder of $\frac{x^{j-i}+\cdots+1}{g(x)}$ is not 0
- Set $g(x)$ to be a polynomial of degree $j - i + 1$. That is, the length of g is the length of the burst error

Convolutional Codes

- An example:
- Input: a bit stream
- Output: for every bit u_i in the input stream, generate 2 bits
- Generate the 2 bits based on the current bit u_i, and the previous two bits u_{i-1} and u_{i-2}
  - Initially, u_{-1} and u_{-2} are 0



(a) Encoder shift register



$\longrightarrow$ $u_n = 0$

$\cdots\cdots\rightarrow$ $u_n = 1$

u_{n-1} u_{n-2}

encode

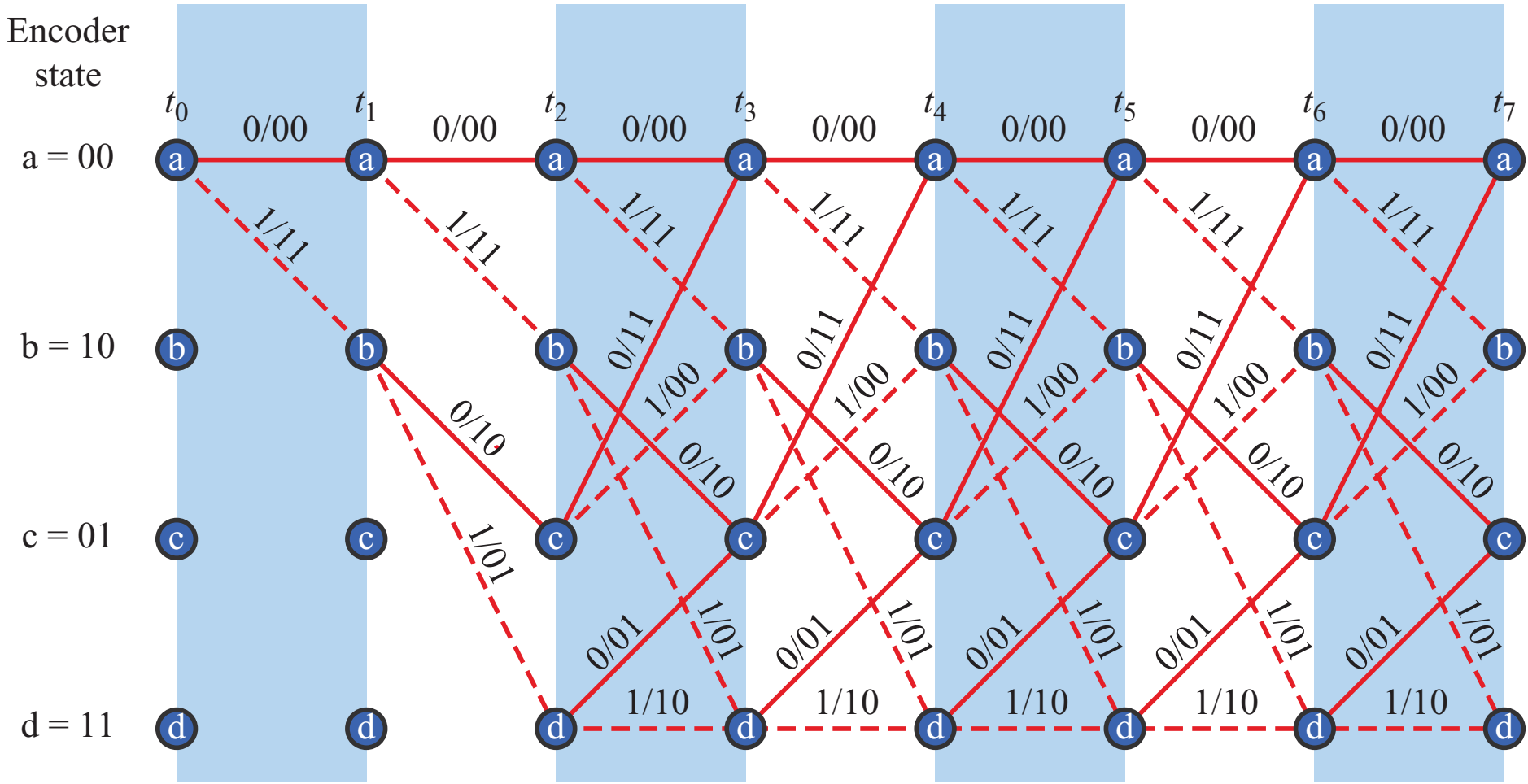
1 0 1 1 0 0 0

1 1 0 0 0 0 0 0 1 1 0 0

decode

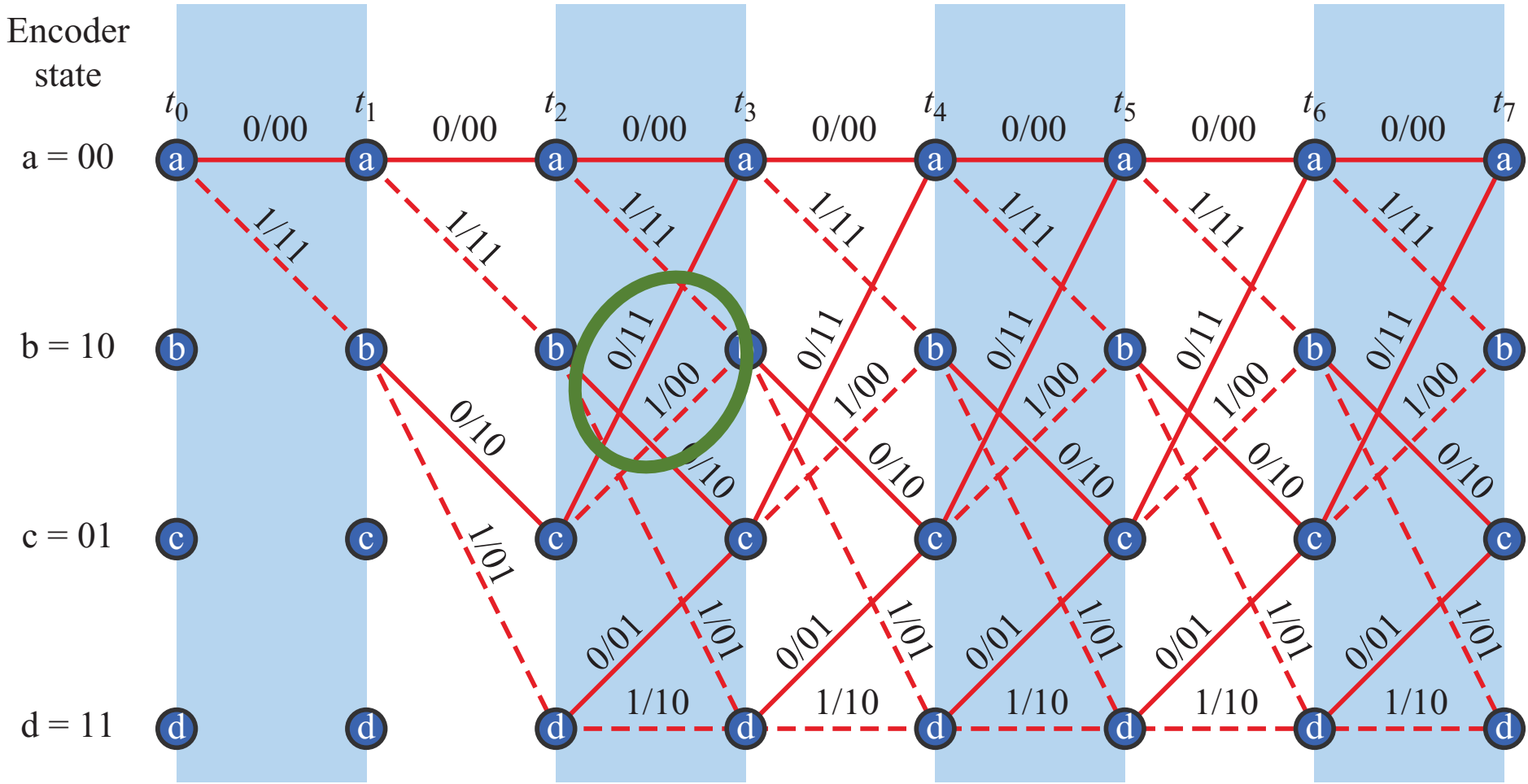
1

How to decode?
For example,
111000010111100



How to decode 11 10 00 01 01 11 00?

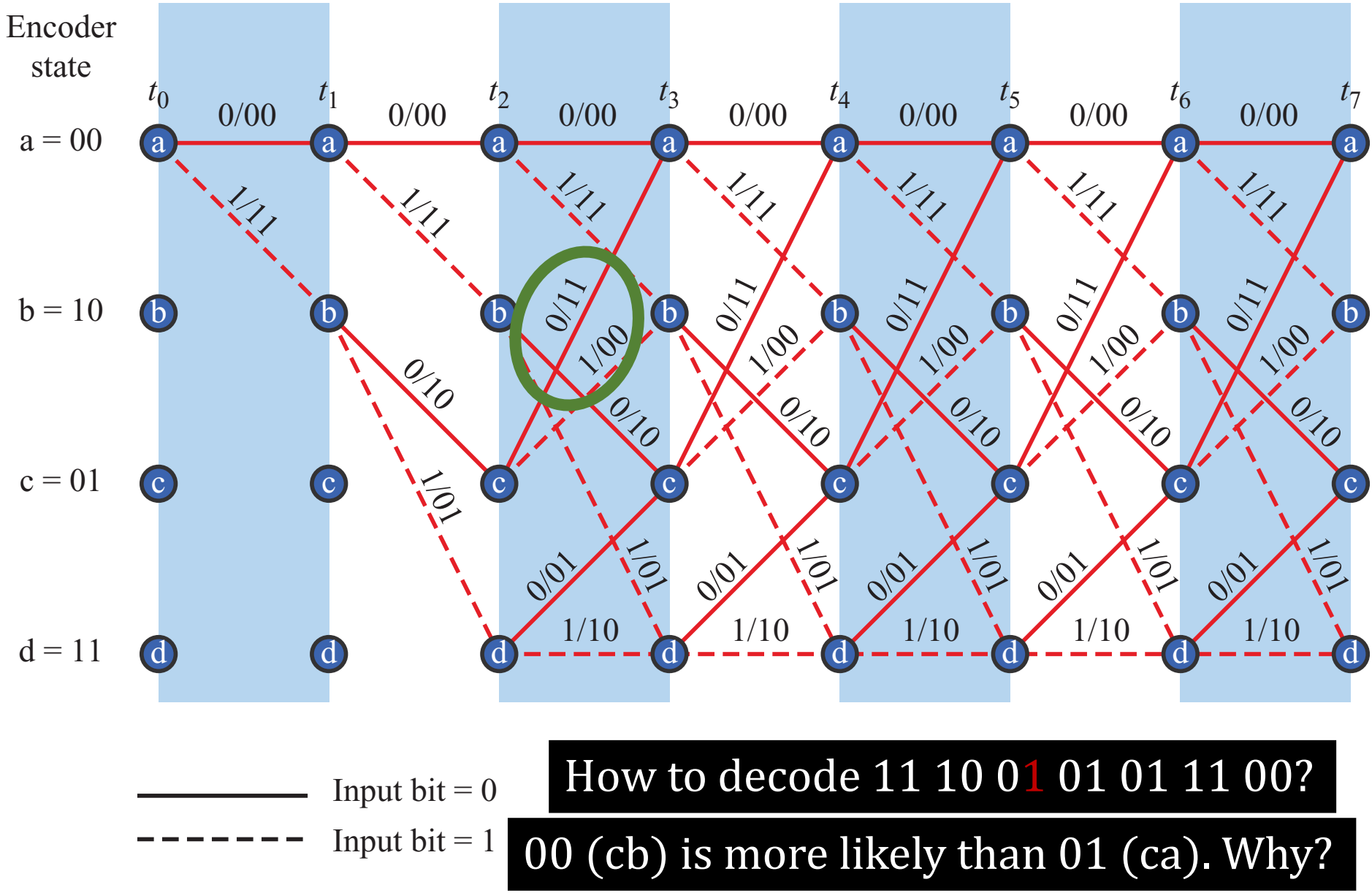
11 10 00 01 01 11 00 → 1011000



How to decode 11 10 01 01 01 11 00?

We know there is an error

Is it possible to **correct** the error?


Encoder state
a = 00
b = 10
c = 01
d = 11
t_0
t_1
t_2
t_3
t_4
t_5
t_6
t_7
0/00
1/11
0/11
1/00
0/10
1/01
0/01
1/10
Input bit = 0
Input bit = 1
How to decode 11 10 01 01 01 11 00?
00 (cb) is more likely than 01 (ca). Why?

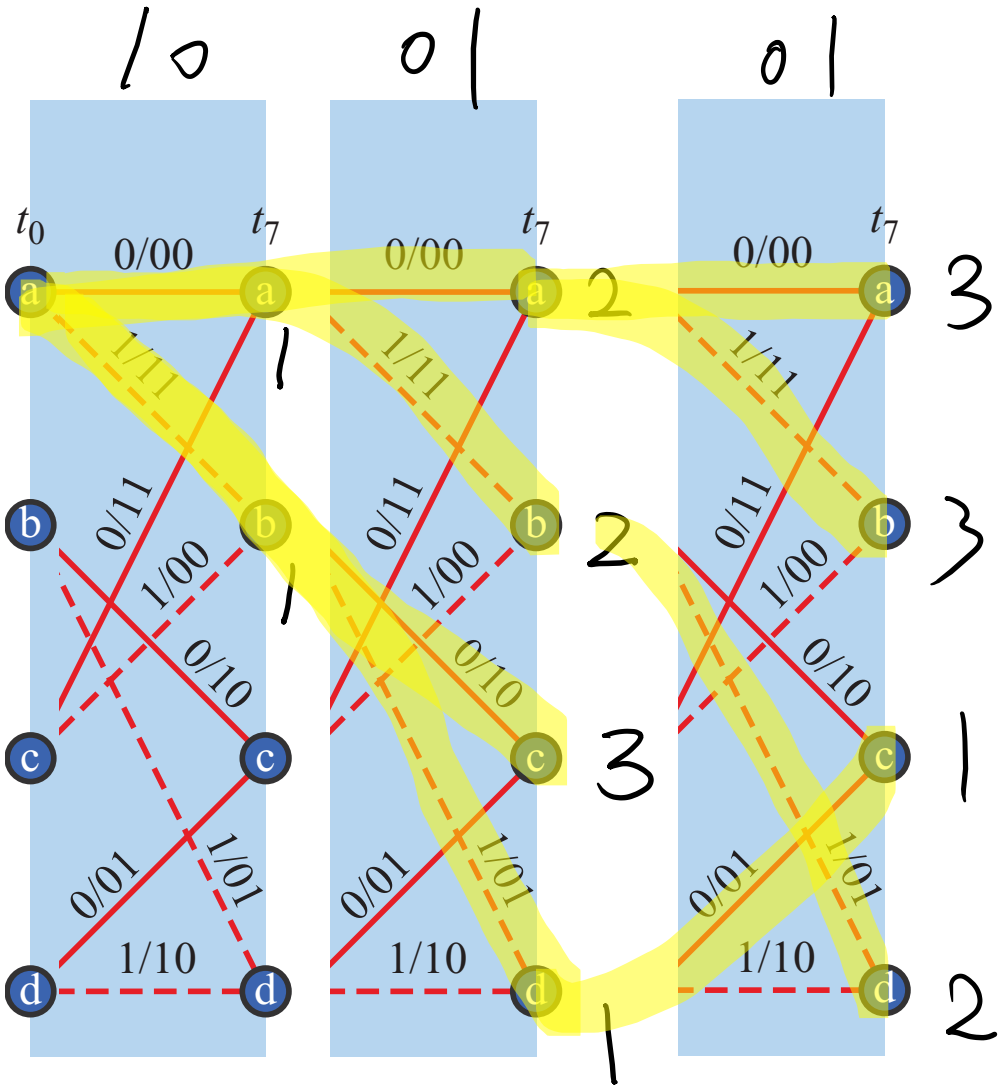
Error Correction

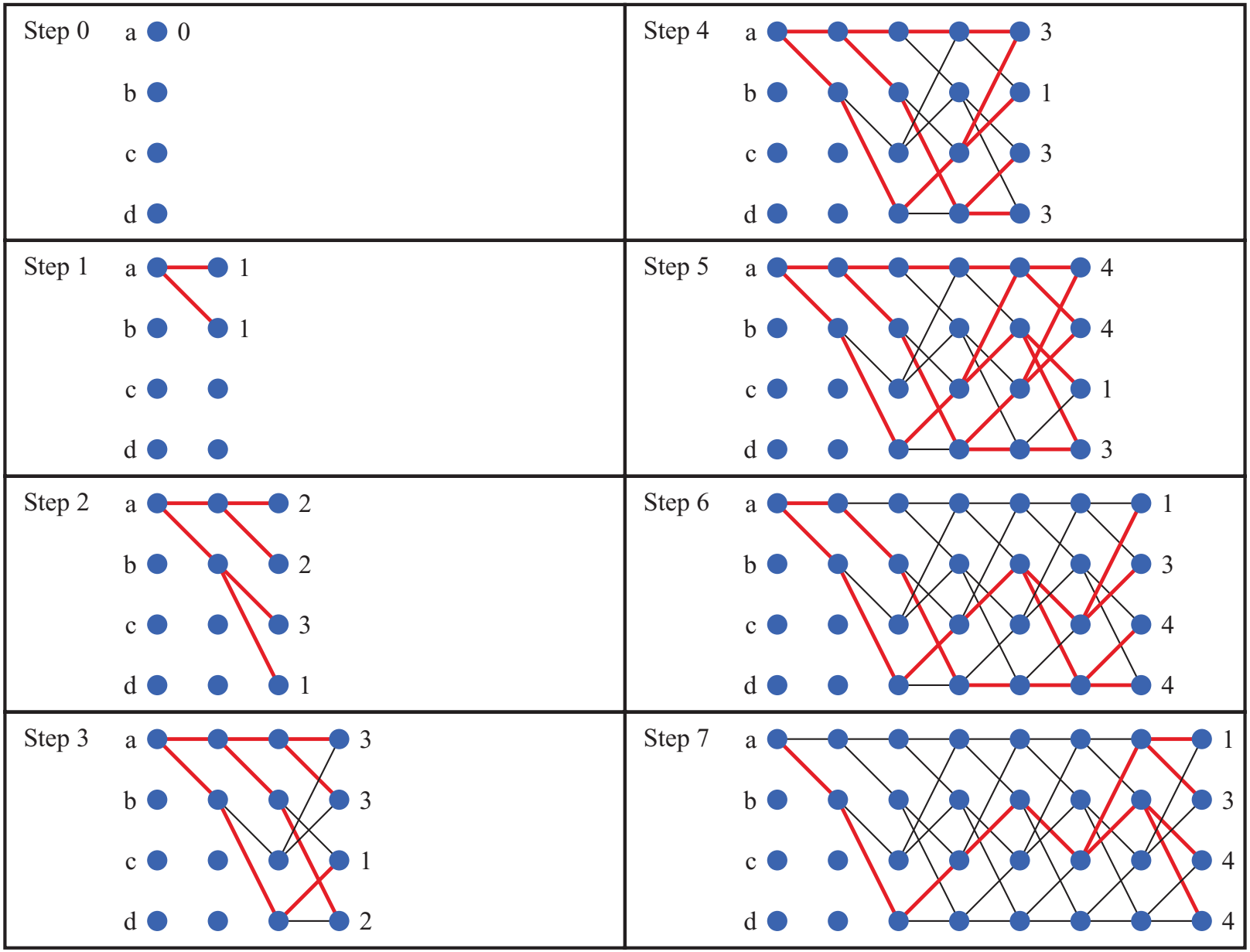
- Idea: find a valid codeword (path) that is closest to the received codeword
- What does “closest” mean?
- Minimum Hamming distance
- To decode (correct) n bits, we consider nb bits at a time (b is the decoding window length)
- Example: Decode 10, with $b = 7$
- Consider: 10 01 01 00 10 11 00

10 01 01 00 10 11 00

↓

11




Step 0
a
b
c
d
0
Step 1
a
b
c
d
1
1
Step 2
a
b
c
d
2
2
3
1
Step 3
a
b
c
d
3
3
1
2
Step 4
a
b
c
d
3
1
3
3
Step 5
a
b
c
d
4
4
1
3
Step 6
a
b
c
d
1
3
4
4
Step 7
a
b
c
d
1
3
4
4

Reference

- Wireless Communication Networks and Systems 1st edition, Global edition Cory Beard, William Stallings © 2016 Pearson Education, Ltd.
- B.A. Forouzan, Data Communications and Networking, McGraw Hill, 2007.
- https://en.wikipedia.org/wiki/Hamming(7,4)
- https://en.wikipedia.org/wiki/Hamming_code
- http://web.mit.edu/6.02/www/f2010/handouts/lectures/L7.pdf
- https://en.wikipedia.org/wiki/Richard_Hamming
- http://web.mit.edu/6.02/www/f2011/handouts/8.pdf
- http://www.cs.huji.ac.il/~nati/PAPERS/expander_survey.pdf