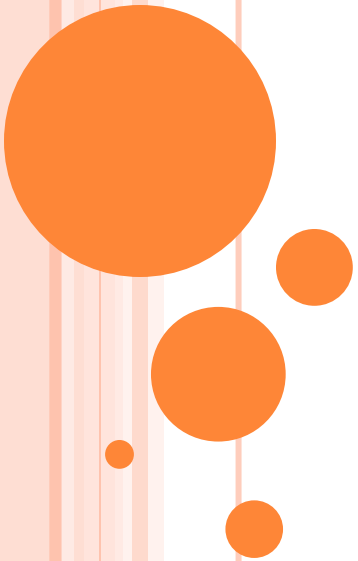
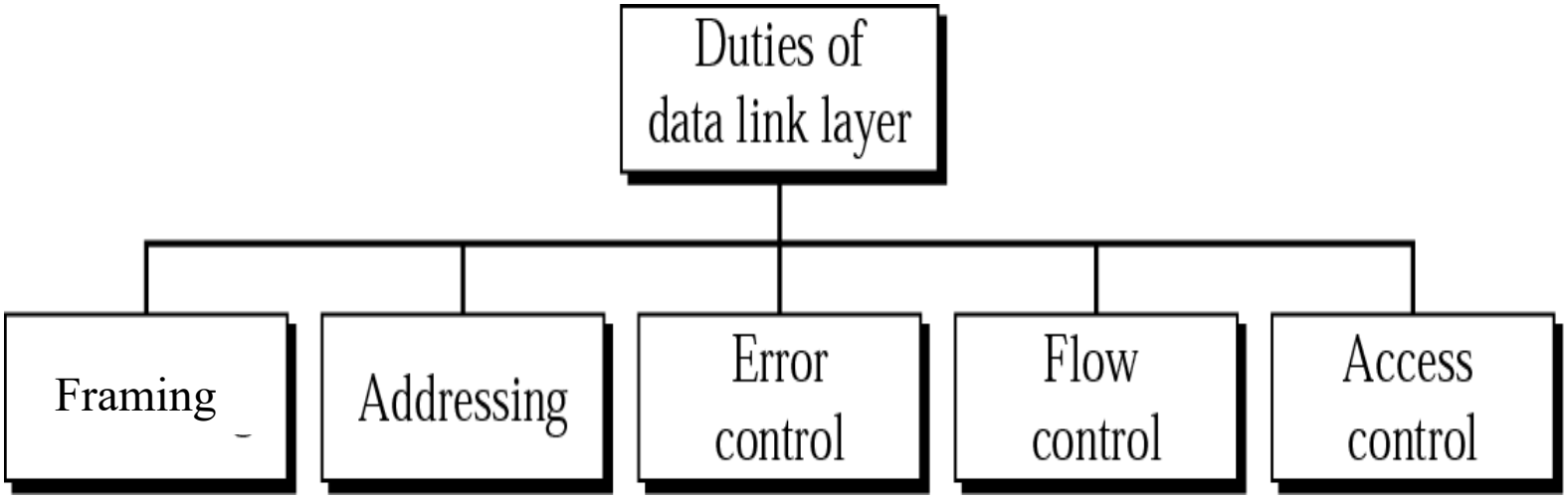
Data Link Layer



The Data Link Layer

- The **second layer** in our model
- Works with whole units of information called **frames** (rather than individual bits, as in the physical layer)
- This study deals with algorithms for achieving reliable, efficient communication


Duties of data link layer
Framing
Addressing
Error control
Flow control
Access control

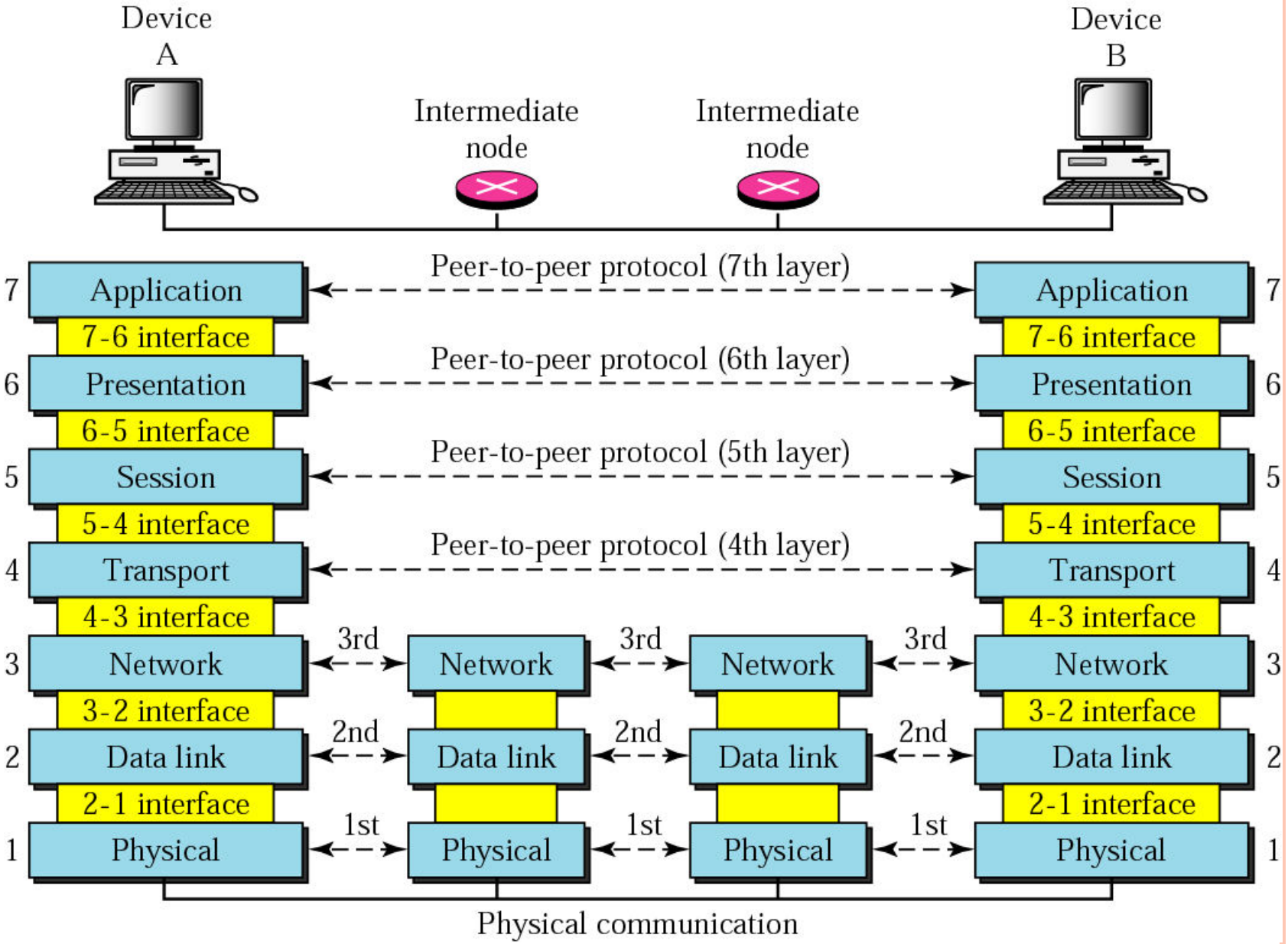
DATA LINK LAYER DESIGN ISSUES

The data link layer uses the services of the physical layer to send and receive bits over communication channels. It has a number of functions, including:

1. Providing a well-defined service interface to the network layer. (reference next slide)
2. Dealing with transmission errors.
3. Regulating the flow of data so that slow receivers are not swamped by fast senders.

Figure 2-2

Architecture of OSI Model



Services Provided to the Network Layer

The principal service is transferring data from the source machine's network layer to the destination machine's network layer.

The data link layer can be designed to offer various services. *Actual services that are offered vary from protocol to protocol*. Three reasonable possibilities that we will consider in turn are:

1. *Unacknowledged connectionless service.*
2. *Acknowledged connectionless service.*
3. *Acknowledged connection-oriented service.*

[no U/Ack + C-oriented service]

Unacknowledged connectionless service

- Unacknowledged connectionless service consists of having the source machine send independent frames to the destination machine without having the destination machine acknowledge them.

Acknowledged connectionless service

- The next step up in terms of reliability is acknowledged connectionless service.
- When this service is offered, there are still no logical connections used, but each frame sent is individually acknowledged.

Acknowledged Connection-Oriented Service

- The most sophisticated service the data link layer can provide to the network layer is connection-oriented service.
- With this service, the source and destination machines establish a connection before any data transfer happen.

FRAME

To accomplish it's goals, the data link layer takes the packets it gets from the network layer and encapsulates them into **frames** for transmission.

Each frame contains a frame **header**, a **payload field** for holding the packet, and a frame **trailer**.

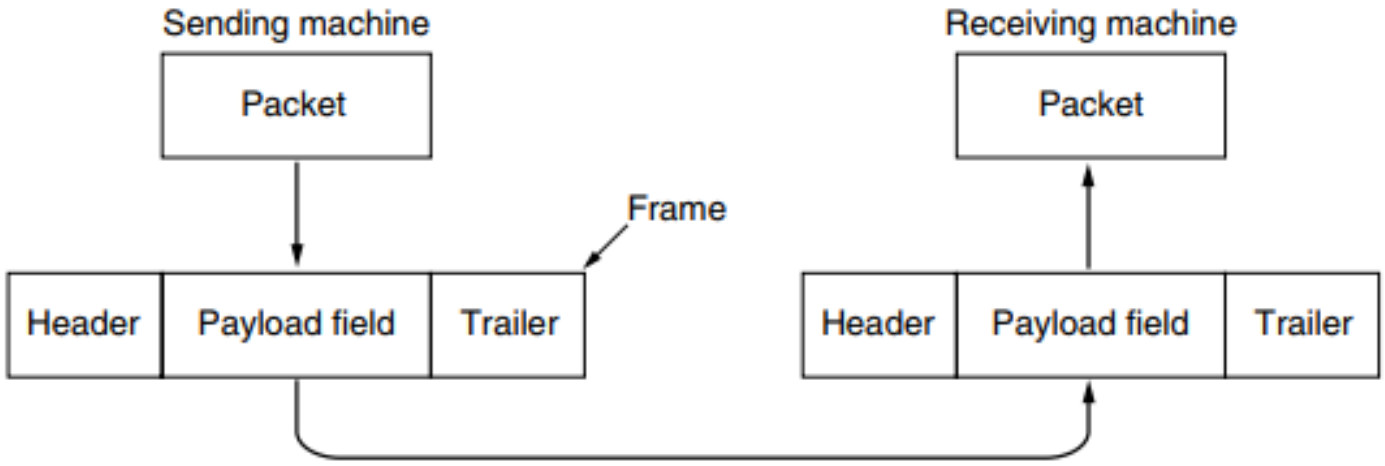


Figure 3-1. Relationship between packets and frames.

Framing

- Data link layer breaks up the bit stream into discrete frames.

Breaking up the bit stream into frames is more difficult than it at first appears:

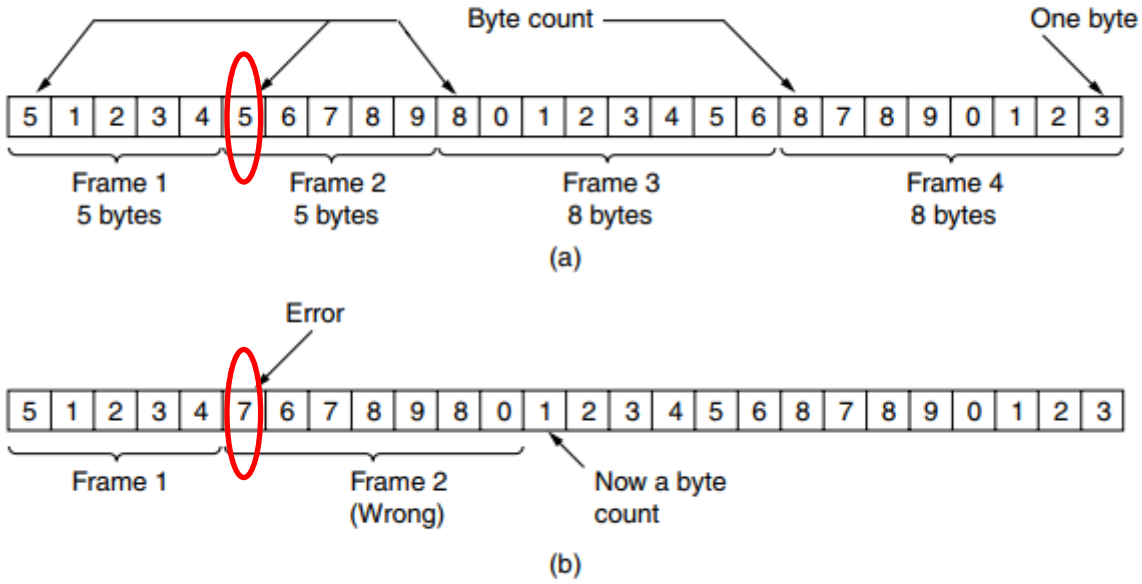
- Methods of framing:

1. ☑ Byte count.
2. ☑Flag bytes with byte stuffing.
3. ☑Flag bits with bit stuffing.
4. Physical layer coding violations

Byte Count

- The first framing method uses a field in the header to specify the number of bytes in the frame.
- When the data link layer at the destination sees the byte count, it knows how many bytes follow and hence where the end of the frame is.

Byte Count



Same frame wrong bit

Figure 3-3. A byte stream. (a) Without errors. (b) With one error.

- The trouble with this algorithm is that the count can be garbled by a transmission error.

Flag bytes with byte stuffing

- The second framing method gets around the problem of resynchronization after an error by having each frame start and end with special bytes.
- Often the same byte, called a **flag byte**.
- Two consecutive flag bytes indicate the end of one frame and the start of the next.
- However, there is a still a problem we have to solve. It may happen that the flag byte occurs in the data, especially when binary data such as photographs or songs are being transmitted. This situation would interfere with the framing.

FLAG BYTES WITH BYTE STUFFING

- One way to solve this problem is to have the sender's data link layer insert a special escape byte (ESC) just before each "accidental" flag byte in the data.

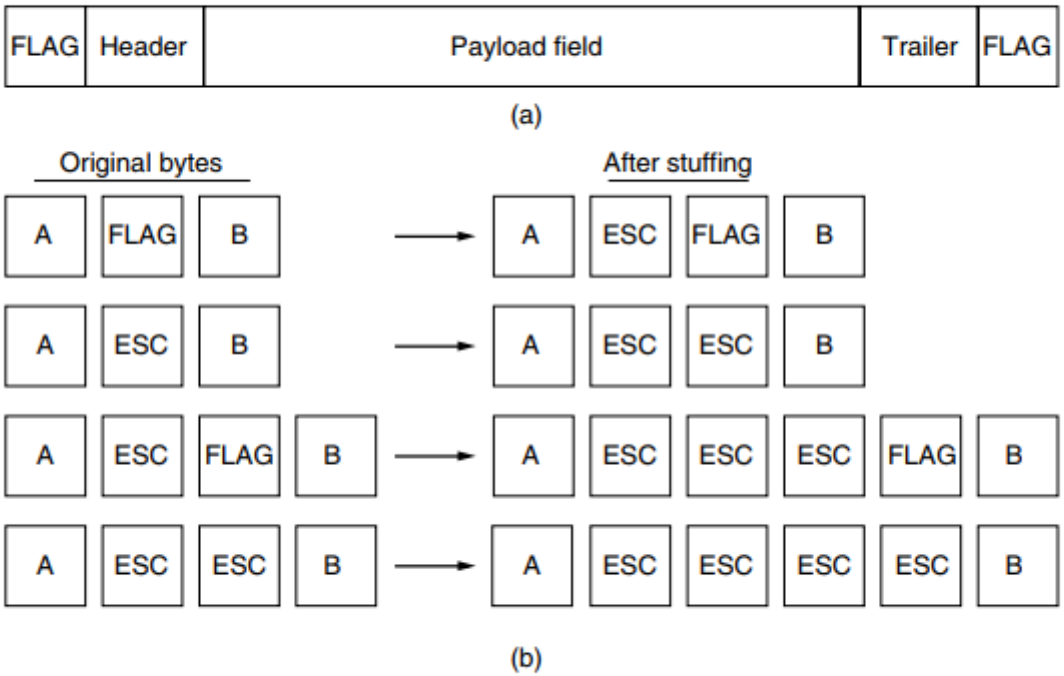


Figure 3-4. (a) A frame delimited by flag bytes. (b) Four examples of byte sequences before and after byte stuffing.

BIT STUFFING

- The third method of delimiting the bit stream gets around a disadvantage of byte stuffing, which is that it is tied to the use of 8-bit bytes. Framing can be also be done at the bit level
- so, frames can contain an arbitrary number of bits made up of units of any size.
- Each frame begins and ends with a special bit pattern, ***01111110*** or $(01111110)_2 = (126)_{10}$ or 0x7E in hexadecimal. This pattern is a flag byte.
- Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a 0 bit into the outgoing bit stream.

BIT STUFFING

- When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically destuffs (i.e., deletes) the 0 bit. Just as byte stuffing is completely transparent to the network layer in both computers, so is bit stuffing. If the user data contain the flag pattern, 01111110, this flag is transmitted as 011111010 but stored in the receiver's memory as 01111110.

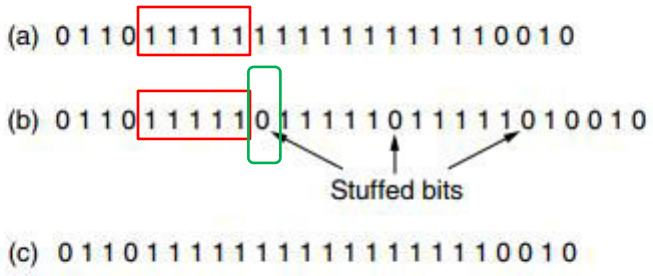


Figure 3-5. Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing.

Error Control

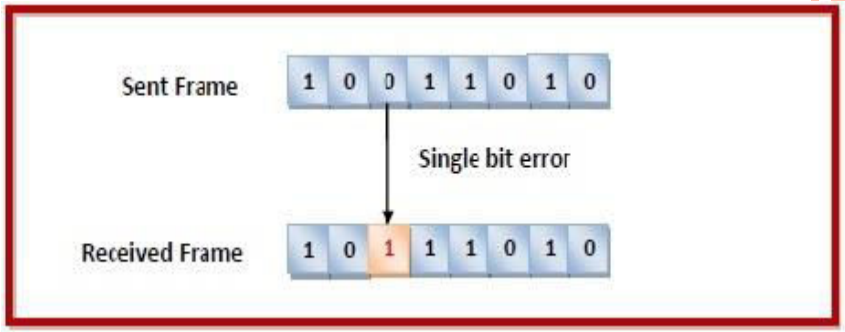
- Data-link layer uses error control techniques to ensure that frames, i.e. bit streams of data, are transmitted from the source to the destination with a certain extent of accuracy.

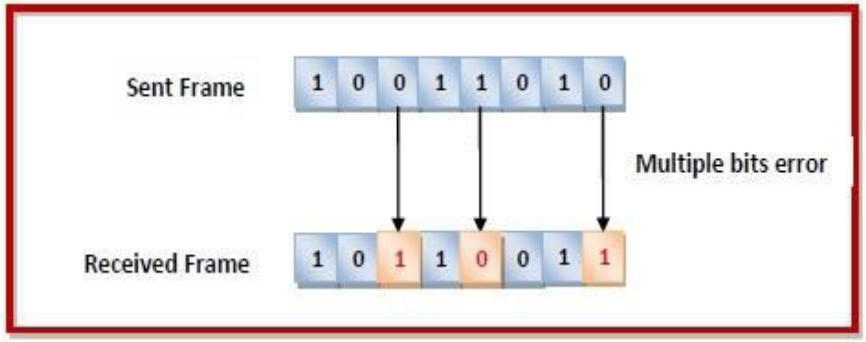
Error

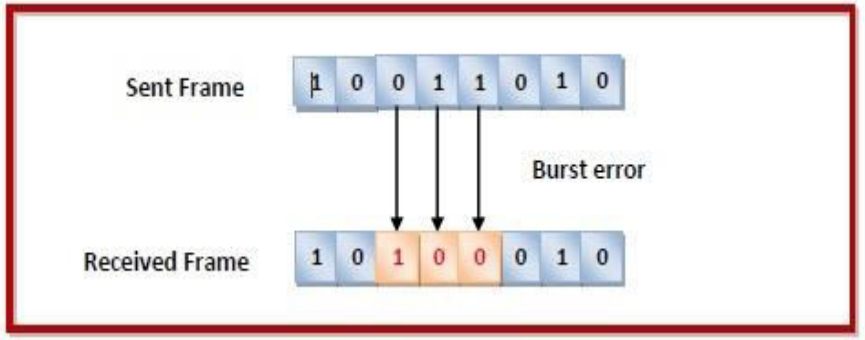
- Networks must be able to transfer data from one device to another with complete accuracy.
- The corrupted bits lead to false data being received by the destination and are called errors.
- Some application require that errors be detected and corrected.
- Errors can be of three types:
  1. single bit errors
  2. multiple bit errors
  3. burst errors.
- **Error detection and correction** are implemented either at the **data link layer** or the **transport layer** of the OSI model.

Error

- **Single bit error** – In the received frame, only one bit has been corrupted, i.e. either changed from 0 to 1 or from 1 to 0.
- **Multiple bits error** – In the received frame, more than one bits are corrupted.
- **Burst error** – In the received frame, more than one consecutive bits are corrupted.







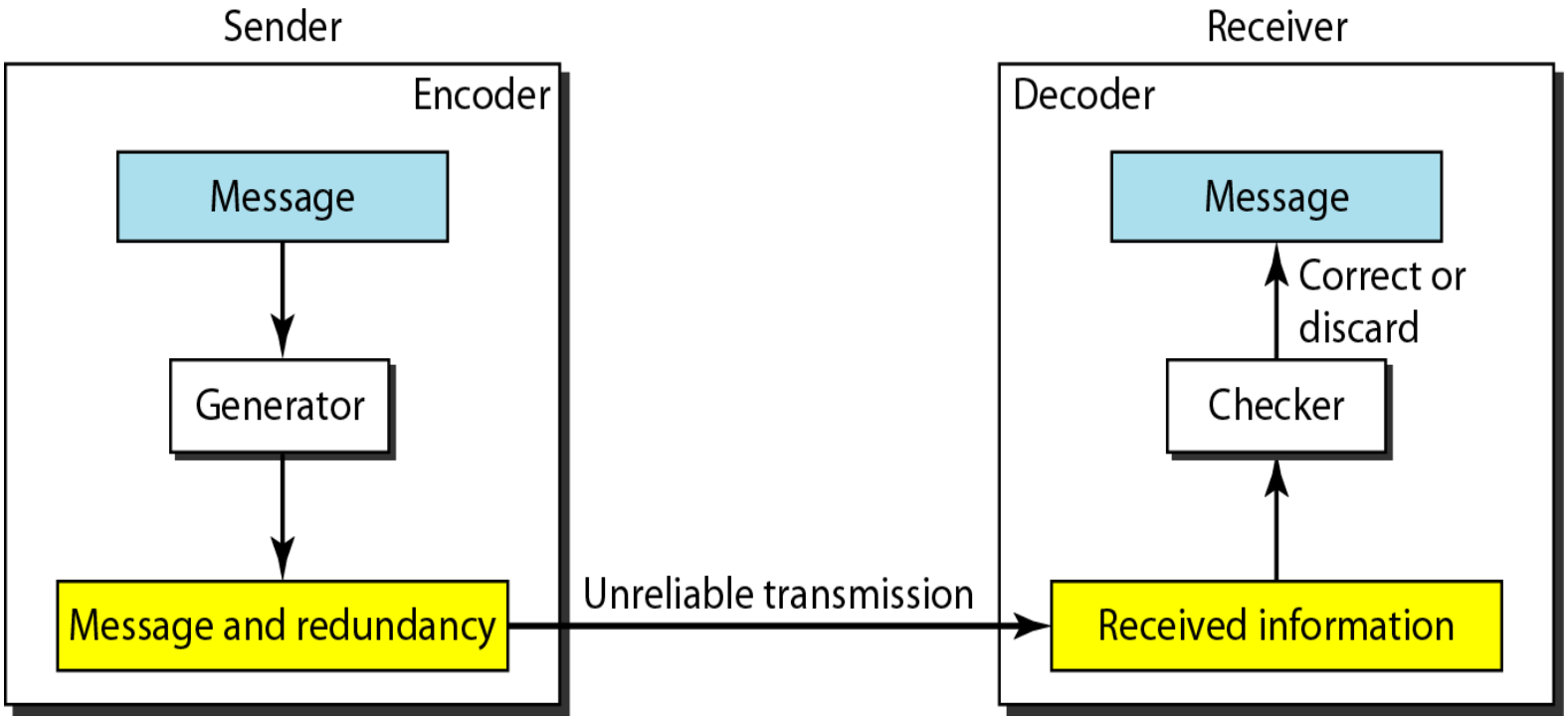
Error Detection & Correction

Error control can be done in two ways in DL Layer:

- **Error detection** – Error detection involves checking whether any error has occurred or not. The number of error bits and the type of error does not matter.
- **Error correction** – Error correction involves ascertaining the exact number of bits that has been corrupted and the location of the corrupted bits.

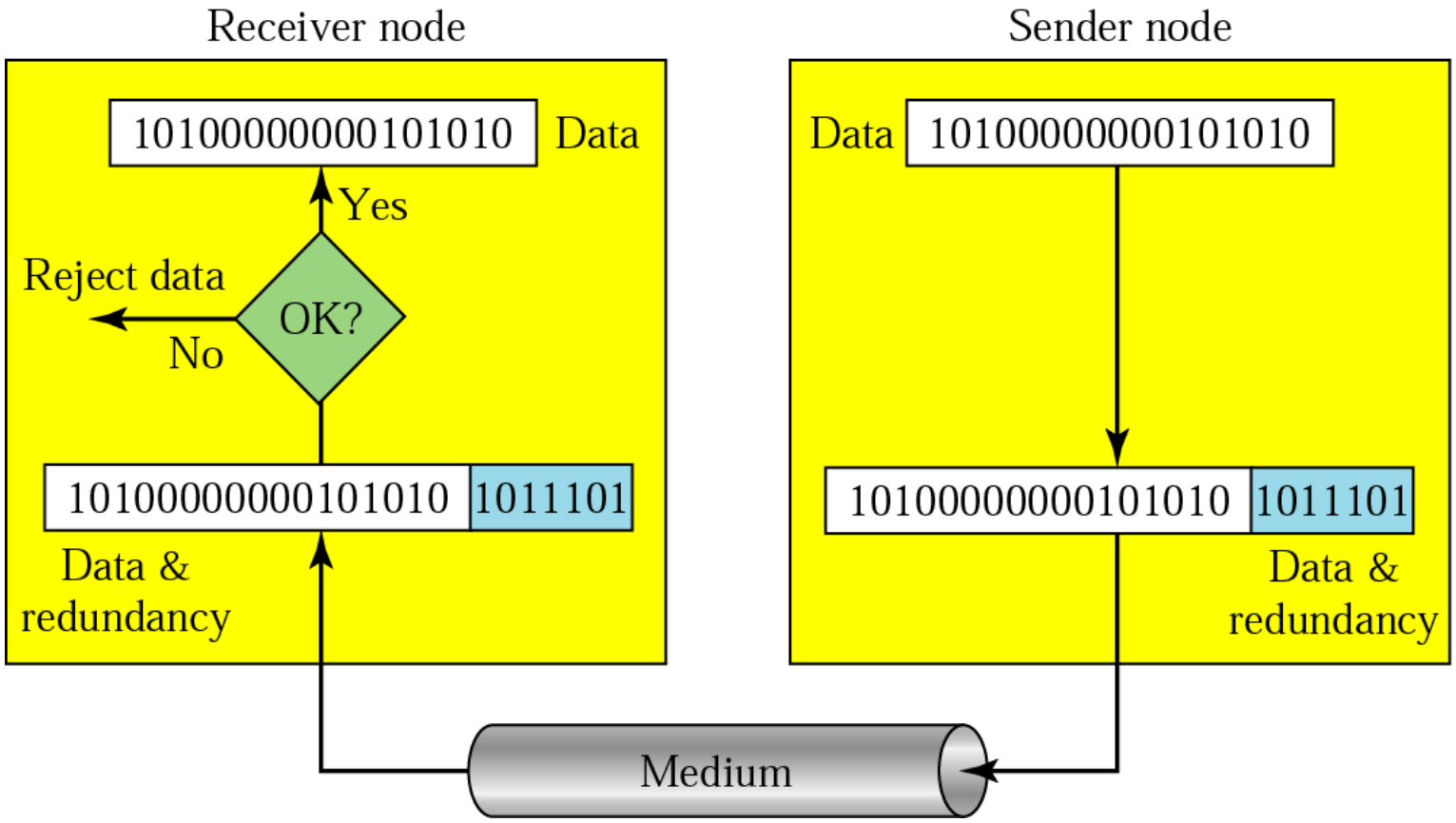
For both error detection and error correction, the **sender needs to send some additional bits** along with the data bits. The receiver performs necessary checks based upon the additional redundant bits. If it finds that the data is free from errors, it removes the redundant bits before passing the message to the upper layers.

The structure of encoder and decoder



DETECTION(CONT'D)

- Redundancy



Error Detection Techniques

Error Detection Techniques:

There are three main techniques for detecting errors in frames: *(used on fiber and high quality wire)*

1. Parity Check (1D[simple] and 2D)
2. Cyclic Redundancy Check (CRC).
3. Checksum

Simple Parity Checking (or One-dimension Parity Check)

- The most common and least expensive mechanism for error- detection is the simple parity check.
- The parity check is done by adding an **extra bit**, called **parity bit** to the data to make a number of **1s either even** (even parity) **or odd** (odd parity)
- While creating a frame, the sender counts the number of 1s in it and adds the parity bit in the following way:
  - If a number of 1s is even then parity bit value is 0
  - If the number of 1s is odd then parity bit value is 1.

Simple Parity Checking
(or One-dimension Parity Check)

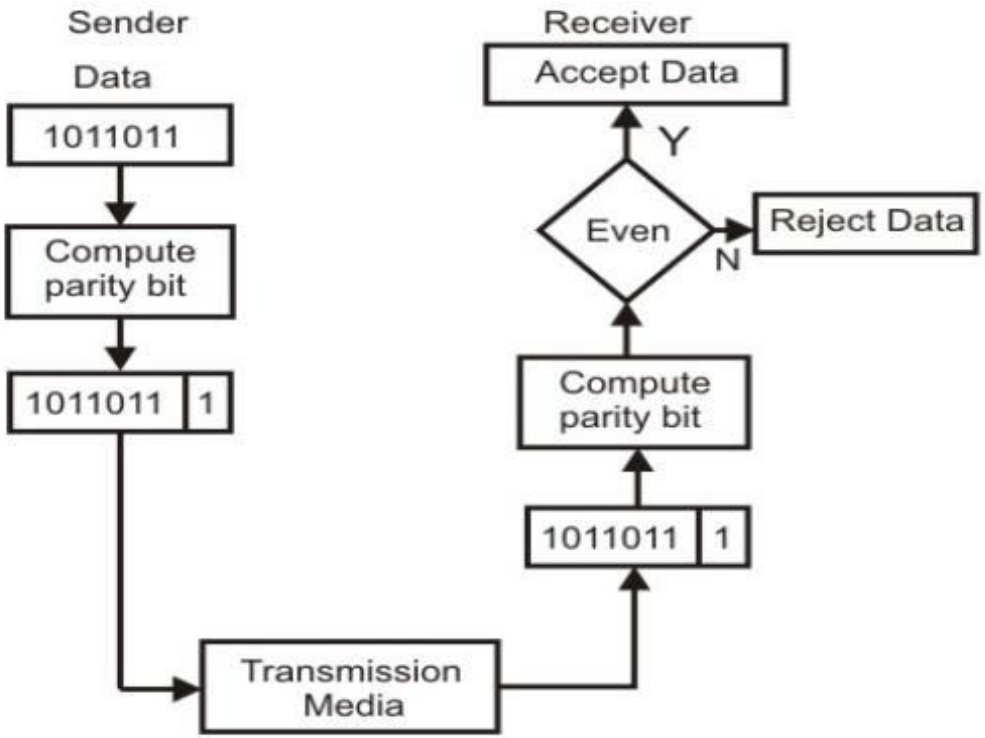


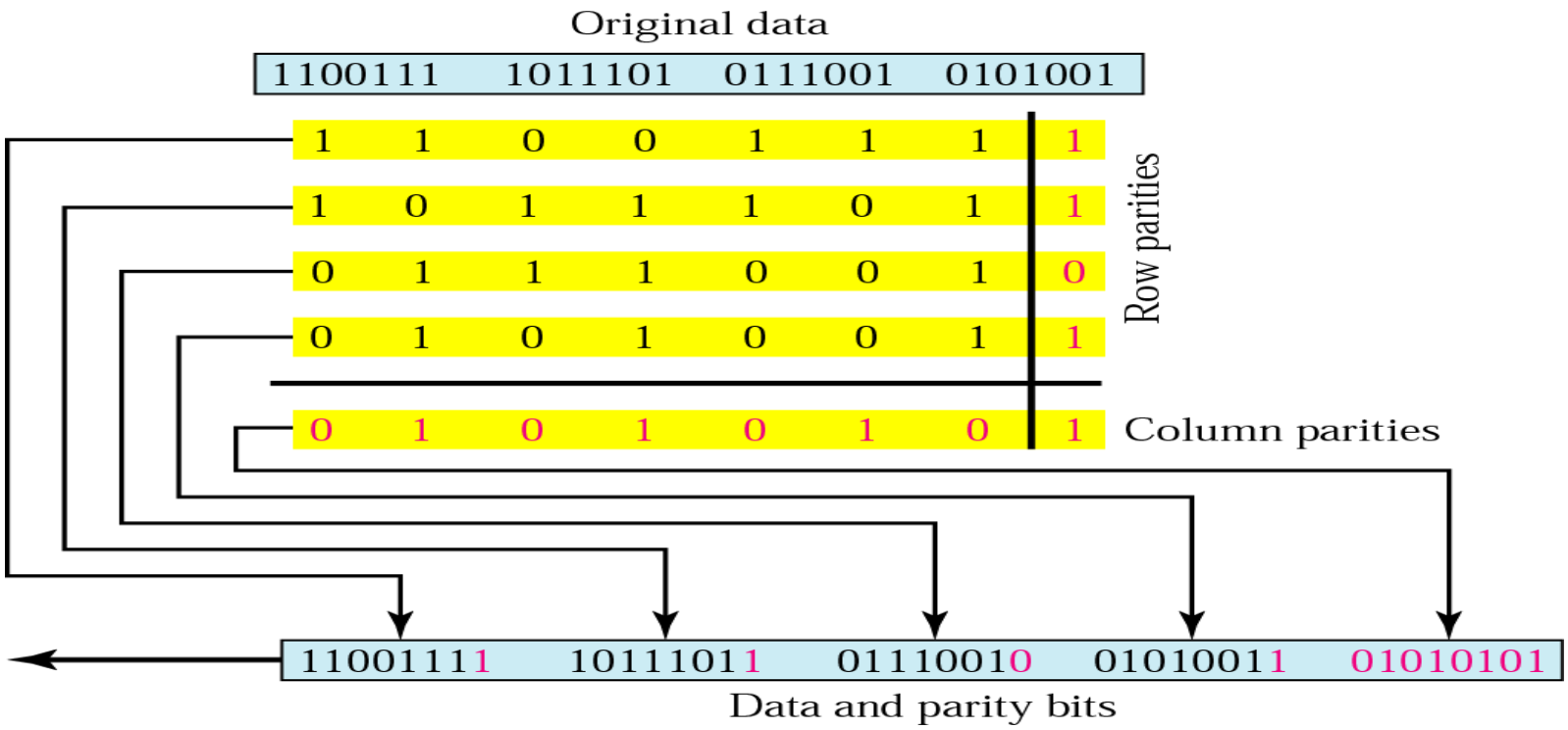
Figure 3.2.3 Even-parity checking scheme

- The parity check is suitable for single bit error detection only.
- A simple parity-cheek code can detect **odd** numbers of error.

2D Parity Checking

- Performance can be improved by using two dimensional parity check.
- Parity check bits are calculated for each row, which is equivalent to a simple parity check bit.
- Parity check bits are also calculated for all columns then both are sent along with the data.

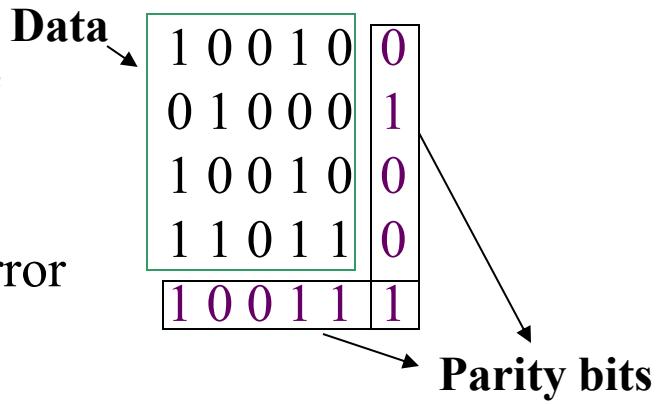
2D Parity Checking



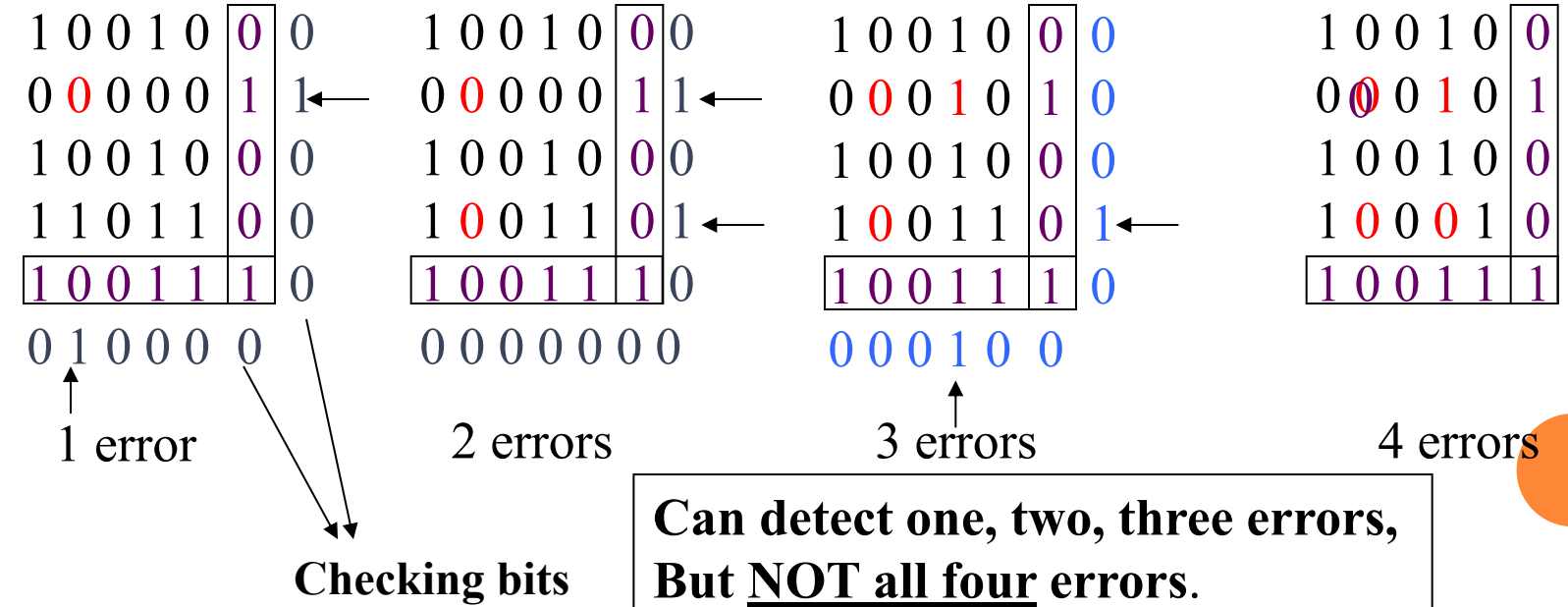
Two- Dimension Parity Checking increases the likelihood of detecting burst errors.

TWO-DIMENSION PARITY

1. Data blocks are organized into table
2. Last column: check bits for rows
3. Last row: check bits for columns
4. Can **detect** and **correct single** bit error



Red bits are errors

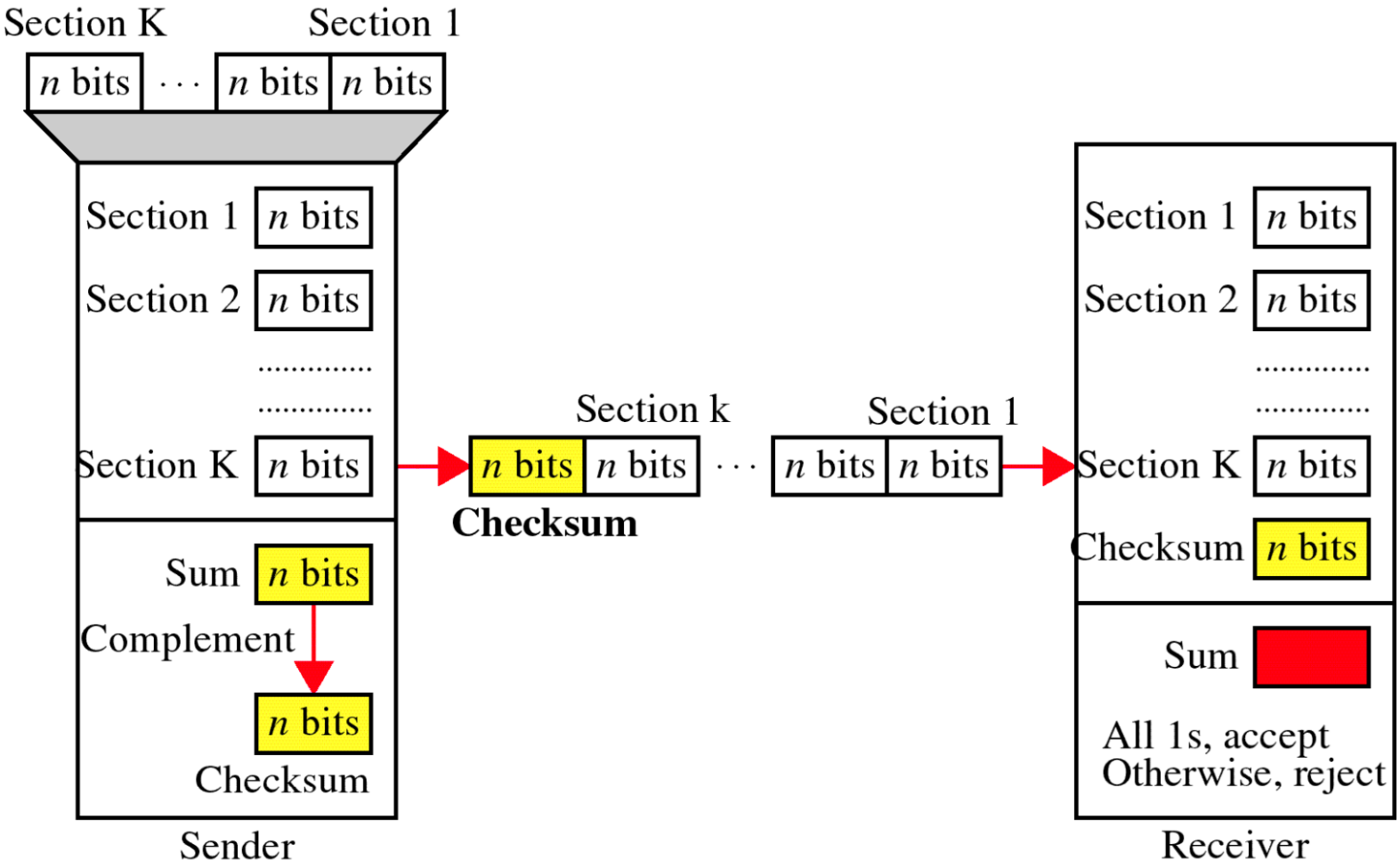


Can detect one, two, three errors, But <u>NOT all four</u> errors.

Checksum

- Sender site:
  - In checksum error detection scheme, the data is divided into k segments each of n bits.
  - In the sender's end the segments are added using 1's complement arithmetic to get the sum.
  - The sum is complemented to get the checksum.
  - The checksum segment is sent along with the data segments
- Receiver End:
  - At the receiver's end, all received segments are added using 1's complement arithmetic to get the sum
  - The sum is complemented.
  - If the result is zero, the received data is accepted; otherwise discarded

Checksum


Section K
Section 1
n bits
n bits
n bits
Section 1
n bits
Section 2
n bits
Section K
n bits
Sum
n bits
Complement
n bits
Checksum
Sender
Section k
Section 1
n bits
n bits
n bits
n bits
Checksum
Section 1
n bits
Section 2
n bits
Section K
n bits
Checksum
n bits
Sum
All 1s, accept
Otherwise, reject
Receiver

Original Data

10011001	11100010	00100100	10000100
1	2	3	4

k=4, m=8

Sender

```
1   10011001
2   11100010
    ---------
   (1)01111011
        1
    ---------
    01111100
3   00100100
    ---------
    10100000
4   10000100
    ---------
   (1)00100100
        1
    ---------
Sum: 00100101
CheckSum: 11011010
```

Reciever

```
1   10011001
2   11100010
    ---------
   (1)01111011
        1
    ---------
    01111100
3   00100100
    ---------
    10100000
4   10000100
    ---------
   (1)00100100
        1
    ---------
    00100101
    11011010
    ---------
Sum: 11111111
Complement: 00000000
```

Conclusion: Accept Data

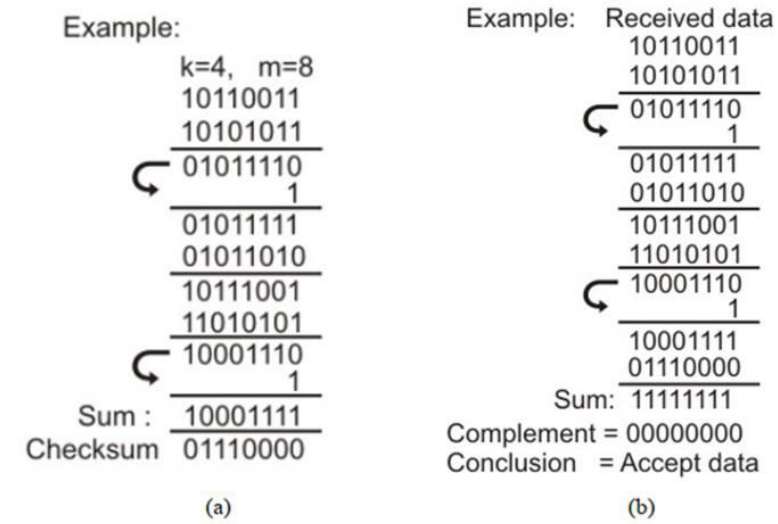


Figure 3.2.5 (a) Sender's end for the calculation of the checksum, (b) Receiving end for checking the checksum

Example

Suppose the following block of 16 bits is to be sent using a checksum of 8 bits.

10101001 00111001

What will be the data sent to the receiver?

Solution

The numbers are added using one's complement

```
                10101001
                00111001
                ----------
Sum             11100010
Checksum        00011101
```

The pattern sent is 10101001 00111001 00011101

Example

Now suppose the receiver receives given the pattern sent using 8-bit checksum:

10101001 00111001 00011101

Check if there is any error.

Solution

When the receiver adds the three sections, it will get all 1s, which, after complementing, is all 0s and shows that there is no error.

	10101001
	00111001
	00011101
Sum	11111111
Complement	00000000 means that the pattern is OK.

Error Correction Techniques

Error correction methods: (used on wireless links = high error environments)

- Hamming code
- Binary convolutional code
- Reed-Solomon and Low-Density Parity Check code (Mathematically complex, widely used in real systems)

Error Correction

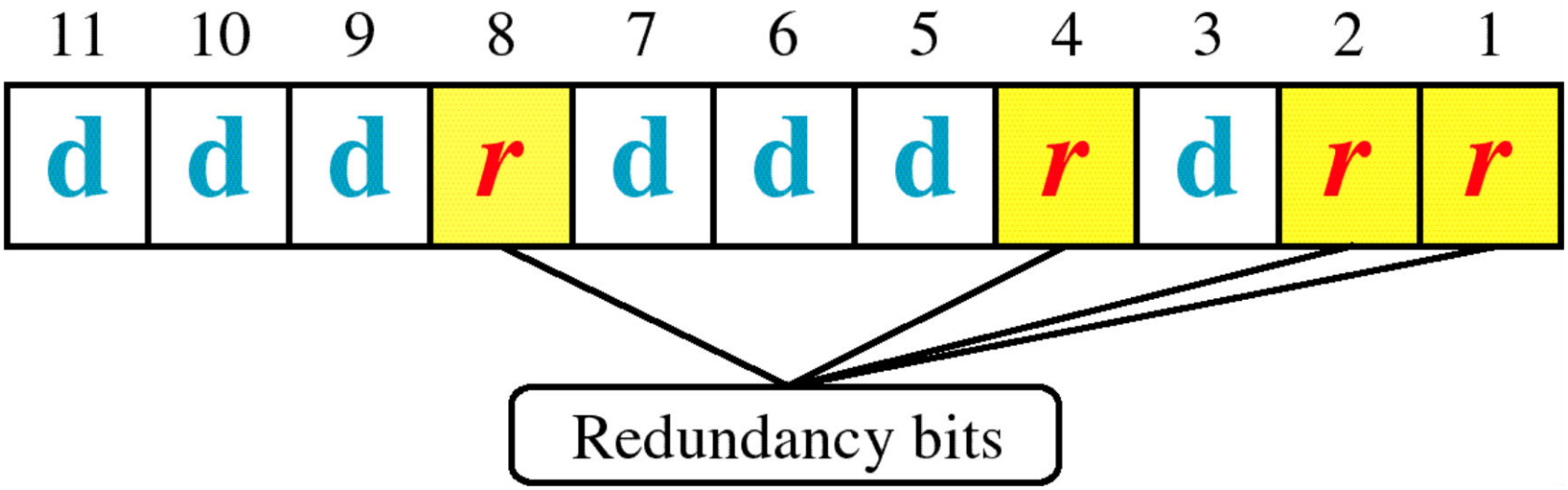
It can be handled in two ways:

1) receiver can have the sender retransmit the entire data unit.
2) The receiver can use an error-correcting code, which automatically corrects certain errors.

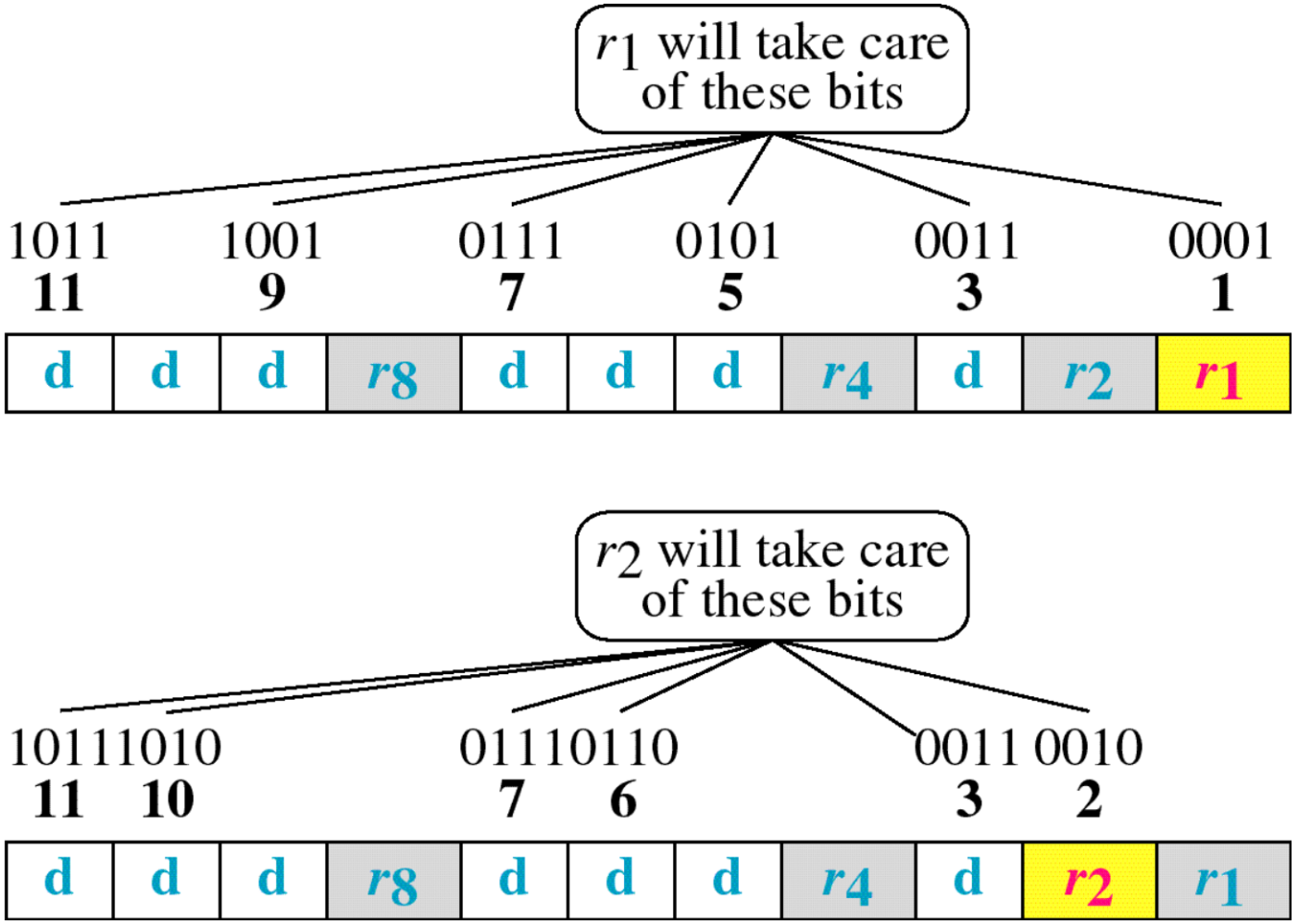
Error Correction

- A technique developed by R.W.Hamming provides a practical solution
- The solution or coding scheme he developed is commonly known as Hamming Code.
- Basic approach for error detection by using Hamming code is as follows:
  - To each group of m information bits k parity bits are added to form (m+k) bit code
  - The k parity bits are placed in positions 1, 2, …, 2k-1 positions.–K parity checks are performed on selected digits of each codeword
  - At the receiving end the parity bits are recalculated. The decimal value of the k parity bits provides the bit-position in error, if any.

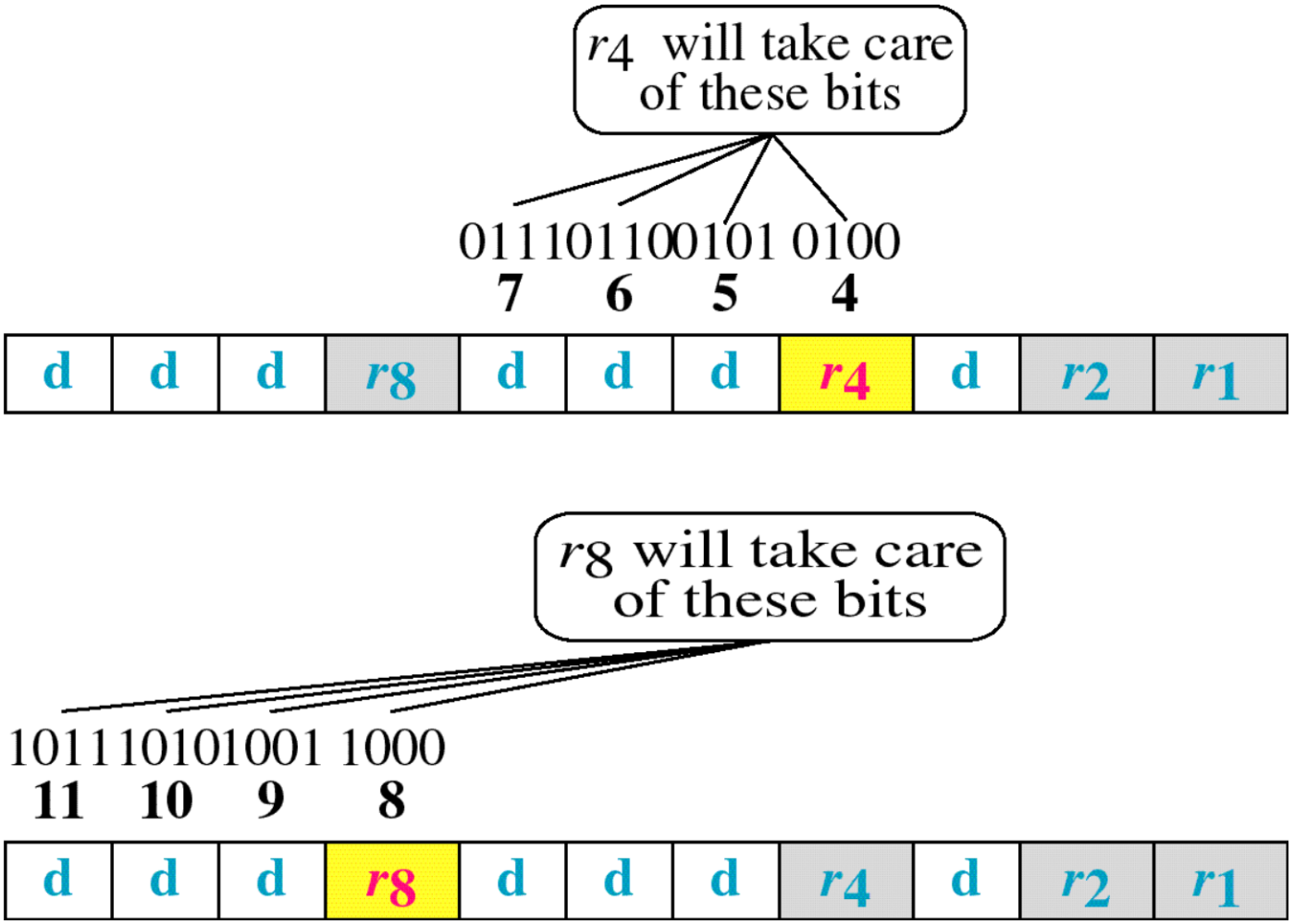
Hamming Code



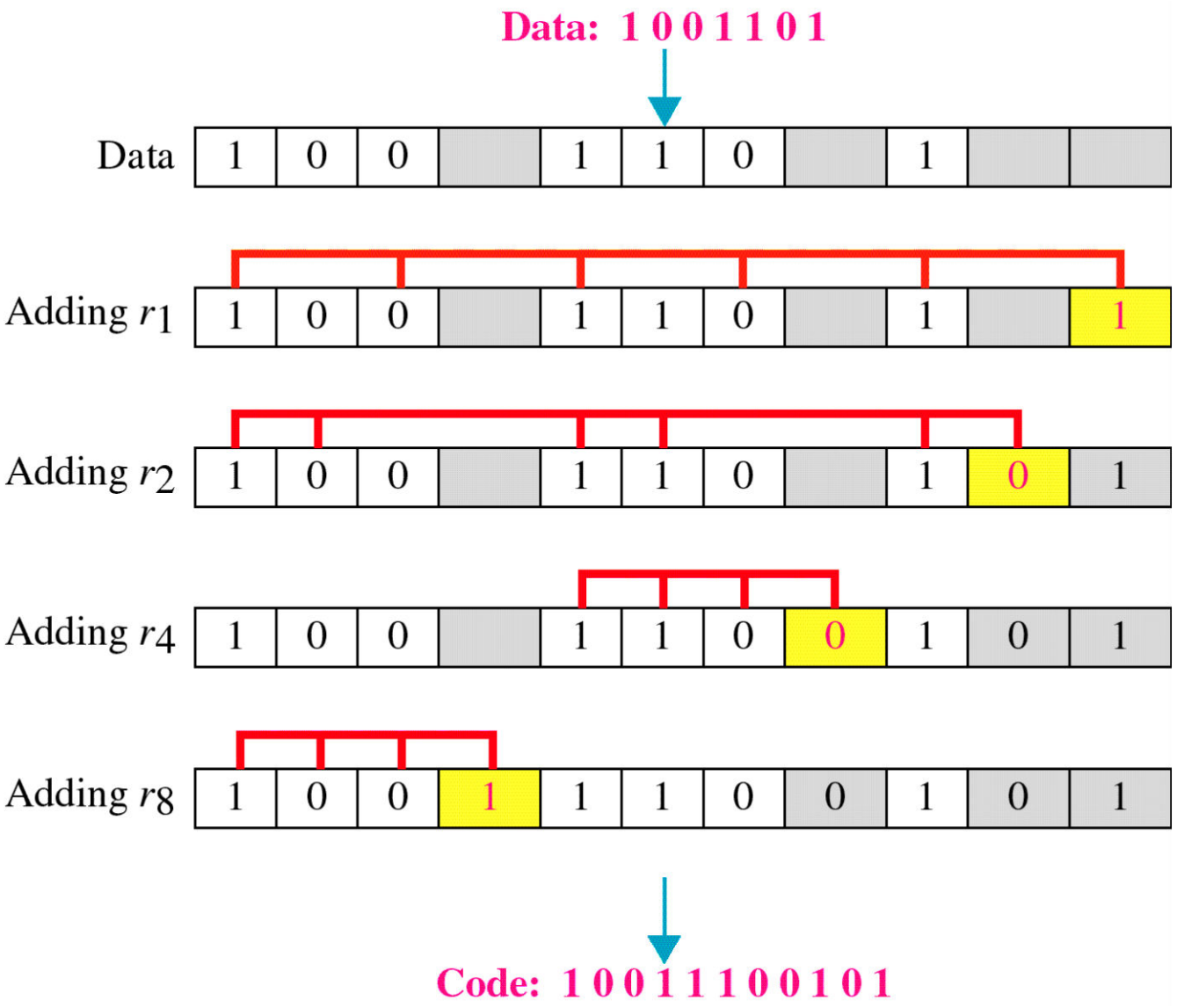
Hamming Code



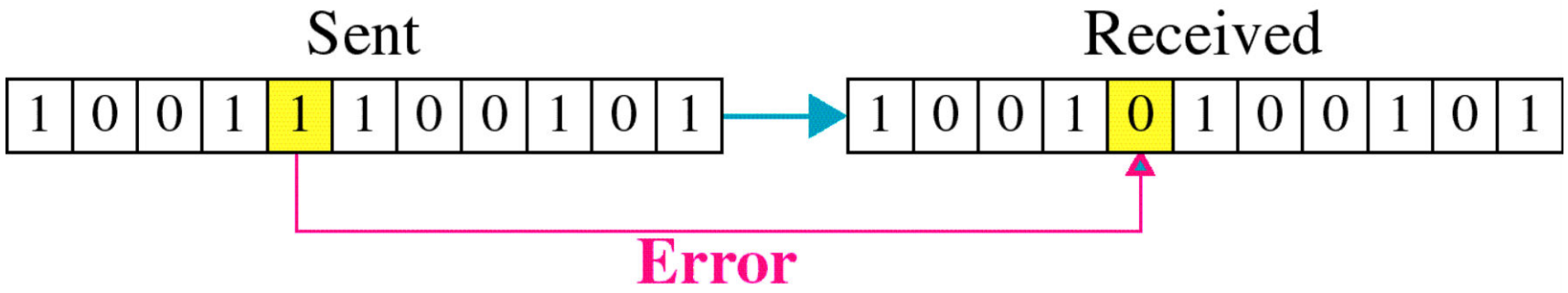
Hamming Code



Example of Hamming Code

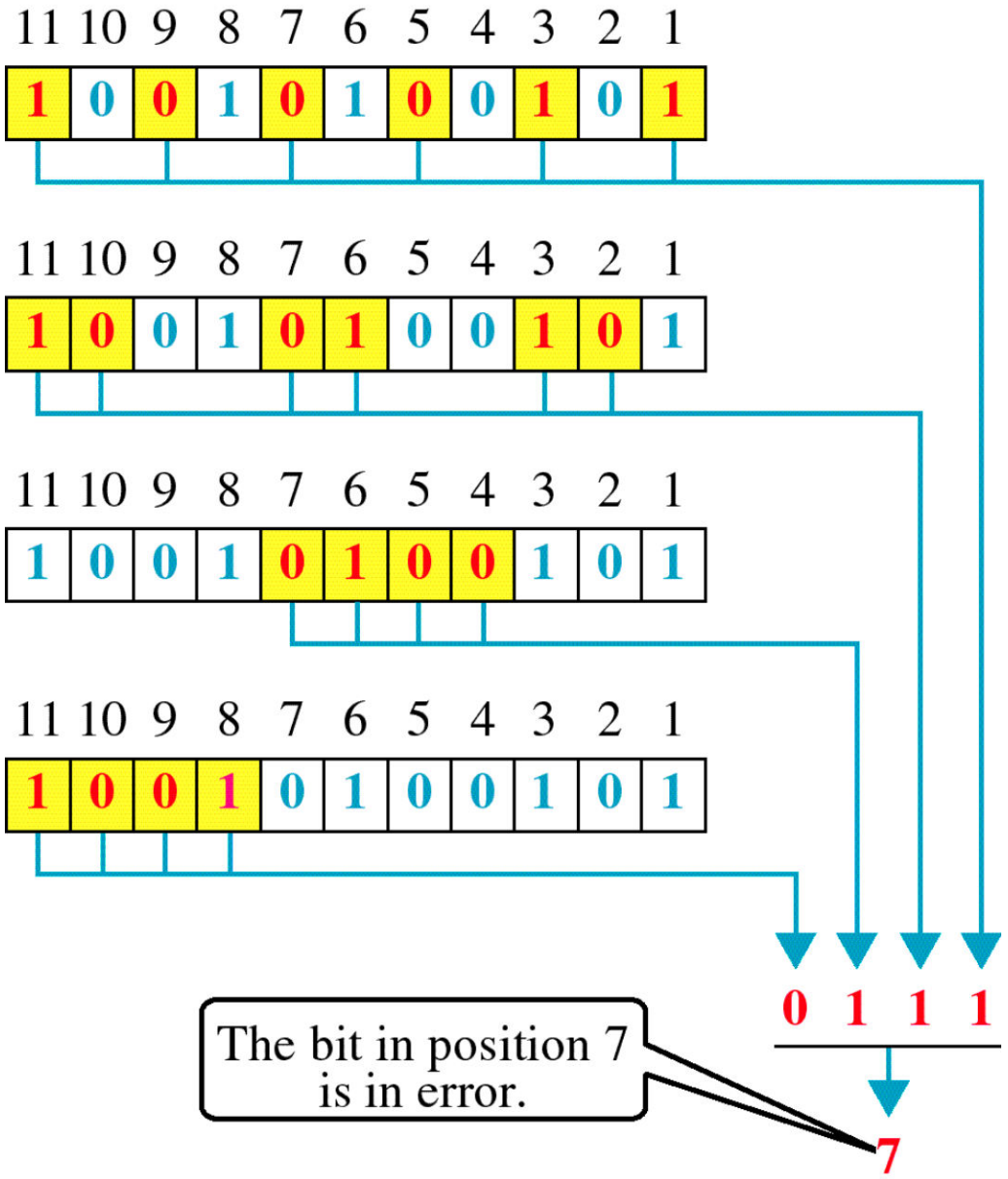


Single-bit error



$2^r >= m + r + 1$

Error Detection



Hamming code (Watch this vid.)

The End