The Link Layer

COMP 30650: Networks and Internet Systems

Dr. Gavin McArdle
Email: gavin.mcardle@ucd.ie
Office: A1.09 Computer Science

RECAP

Modulation Schemes

- **Baseband Modulation**
  - Manchester Encoding, NRZ, NRZI, 4B/5B
- **Passband Modulation**
  - Carrier signal
  - Increasing bits through phase, frequency and amplitude key shifting.
  - Constellation Diagrams

TODAY'S PLAN

Link Layer

- Framing
  - Byte/Bit stuffing
- Errors
  - Detection
  - Correction

THE LINK LAYER

Moving <u>up</u> to the Link Layer

Application	- HTTP, DNS, CDNs
Transport	- TCP, UDP
Network	- IP, NAT, BGP
Link	- Ethernet, 802.11
Physical	- wires, fiber, wireless

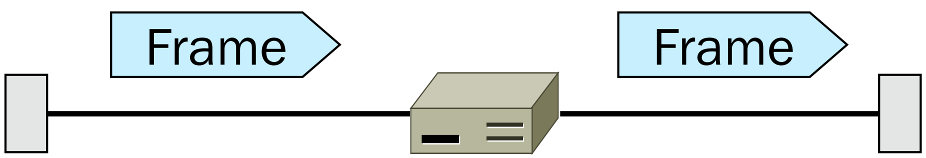
SCOPE OF THE LINK LAYER

- Responsible for delivering frames of information over a single link
- Establishes connections between neighbouring nodes to send data.
- Implements the actual topology of the local network that allows the internet layer to present an addressable interface.
- Achieves this by sending data to a physical address of another node in the network (MAC Address).
- Handles transmission errors and regulates the flow of data.

SCOPE OF THE LINK LAYER

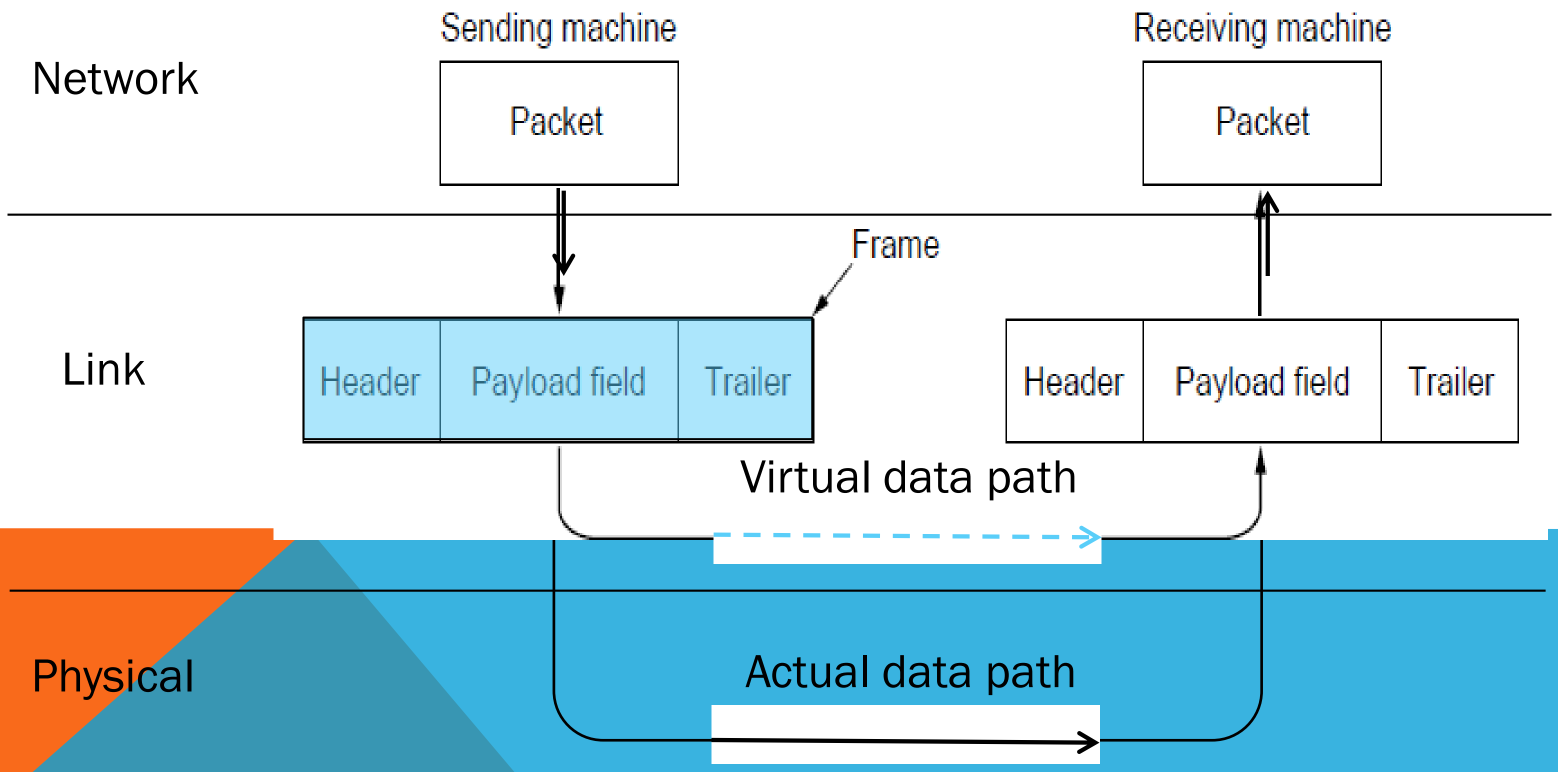
Concerns how to transfer messages between links

- Messages are <u>frames</u>, of limited size
- Builds on the physical layer

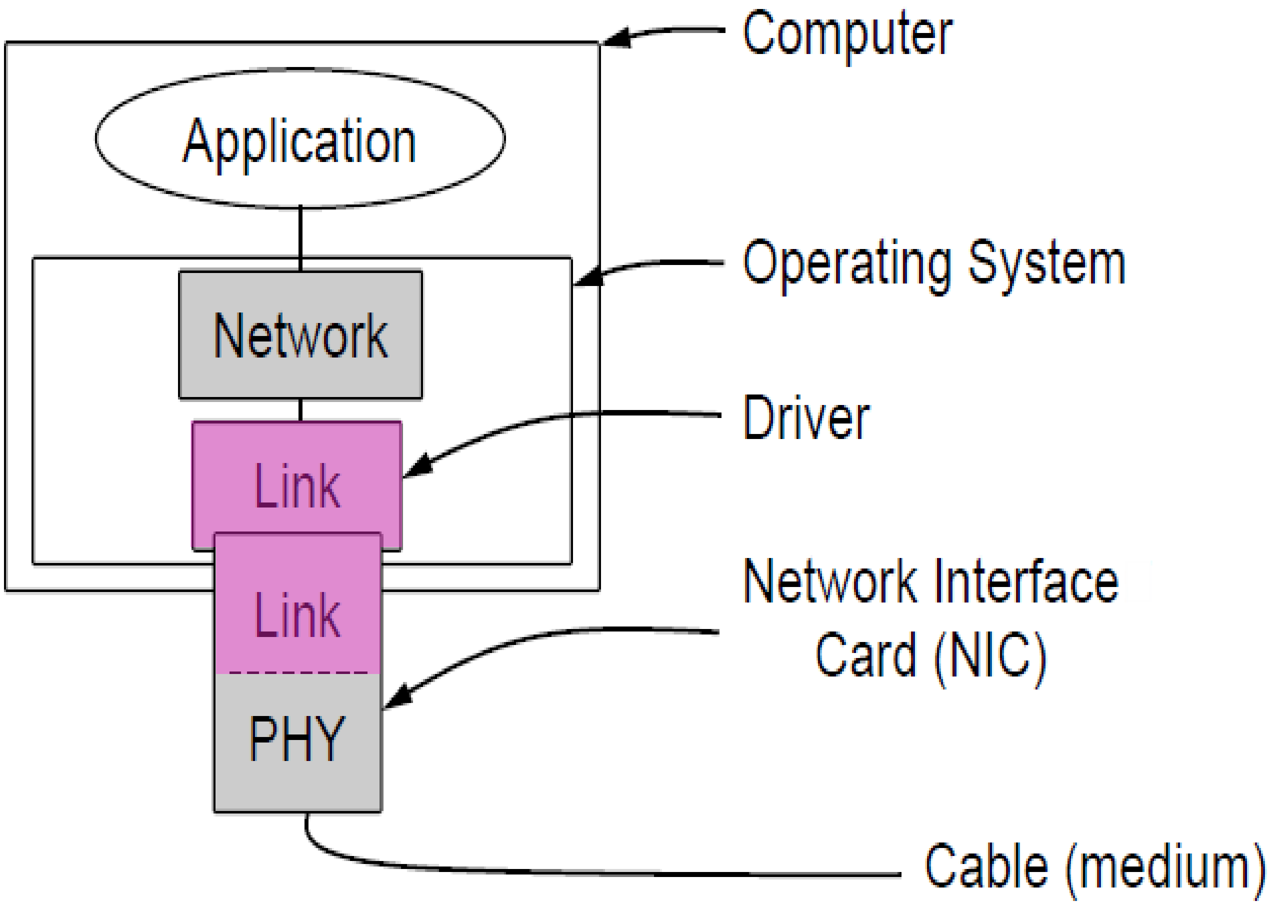


IN TERMS OF LAYERS …

Link layer accepts packets from the network layer, and encapsulates them into frames that it sends using the physical layer; reception is the opposite process



TYPICAL IMPLEMENTATION OF LAYERS

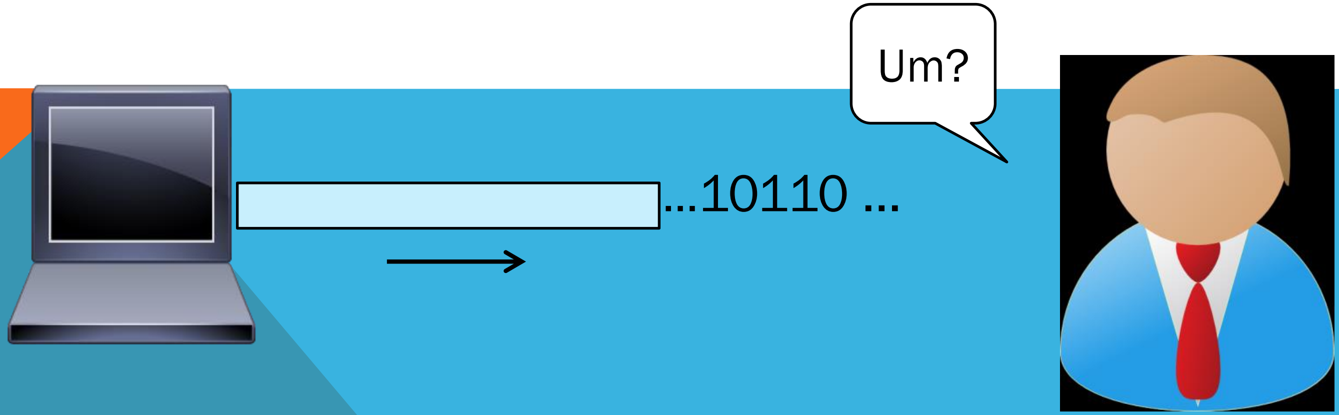


TOPICS OF THE LINK LAYER

1. **Framing**
   - Delimiting start/end of frames
2. **Error detection and correction**
   - Handling errors
3. **Retransmissions**
   - Handling loss
4. **Multiple Access**
   - 802.11, classic Ethernet
5. **Switching**
   - Modern Ethernet

FRAMING

- The Physical layer gives us a stream of bits. How do we interpret it as a sequence of frames?
- Framing provides a way for a sender to transmit a set of bits that are meaningful to the receiver
- The advantage of using frames is that data is broken up into recoverable chunks that can easily be checked for corruption.



FRAMING METHODS

We'll look at:

- Byte count
- Byte stuffing
- Bit stuffing

In practice, the physical layer often helps to identify frame boundaries

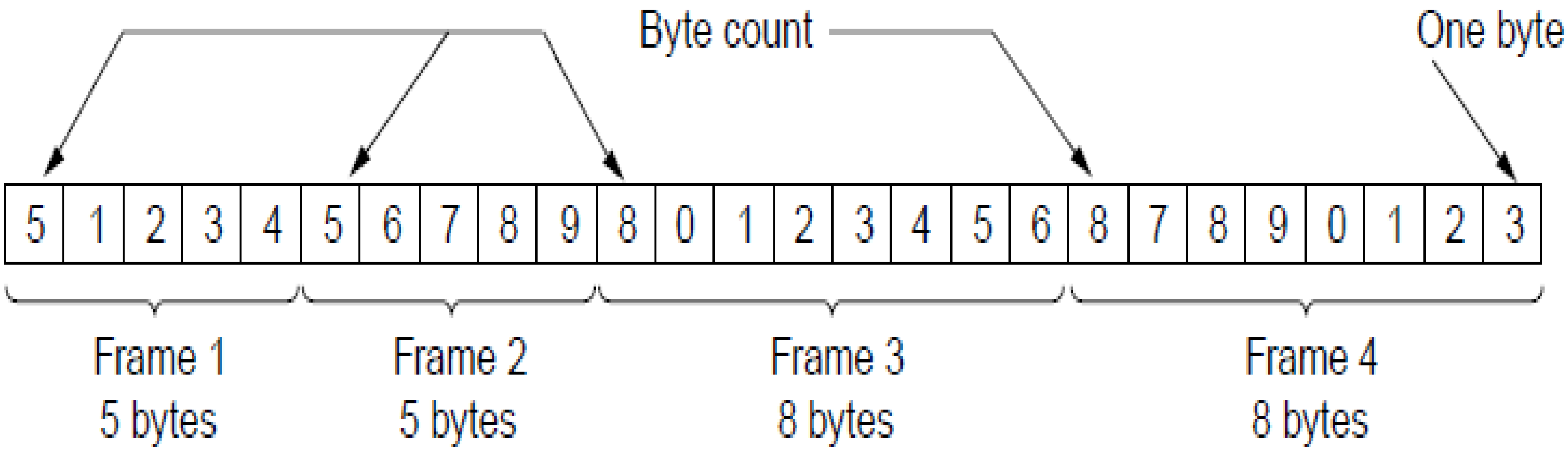
- Ethernet, 802.11
- Protocols

BYTE COUNT

First try:

- Let’s start each frame with a length field
- It’s simple, and hopefully good enough …

BYTE COUNT

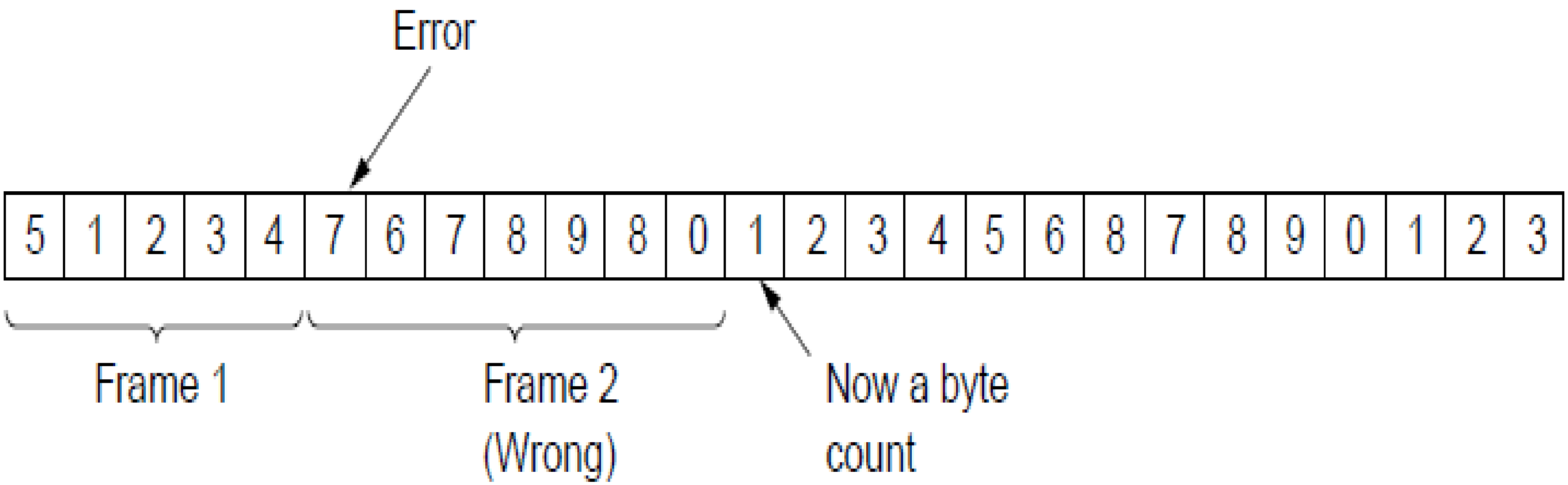


Byte count
One byte
5 1 2 3 4 5 6 7 8 9 8 0 1 2 3 4 5 6 8 7 8 9 0 1 2 3
Frame 1
5 bytes
Frame 2
5 bytes
Frame 3
8 bytes
Frame 4
8 bytes

BYTE COUNT

Difficult to re-synchronize after framing error

- Want a way to scan for a start of frame



BYTE STUFFING

Better idea:

- Have a special **flag** byte value that means start/end of frame

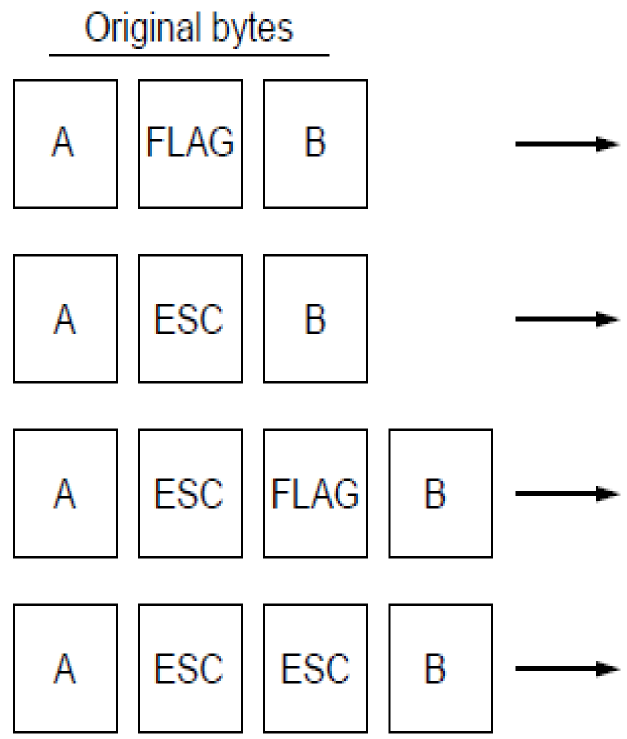
Complications

- Replace ("stuff") the flag inside the frame with an escape code
- Have to escape the escape code too!
- Longer, but easy to resynchronize after error

BYTE STUFFING

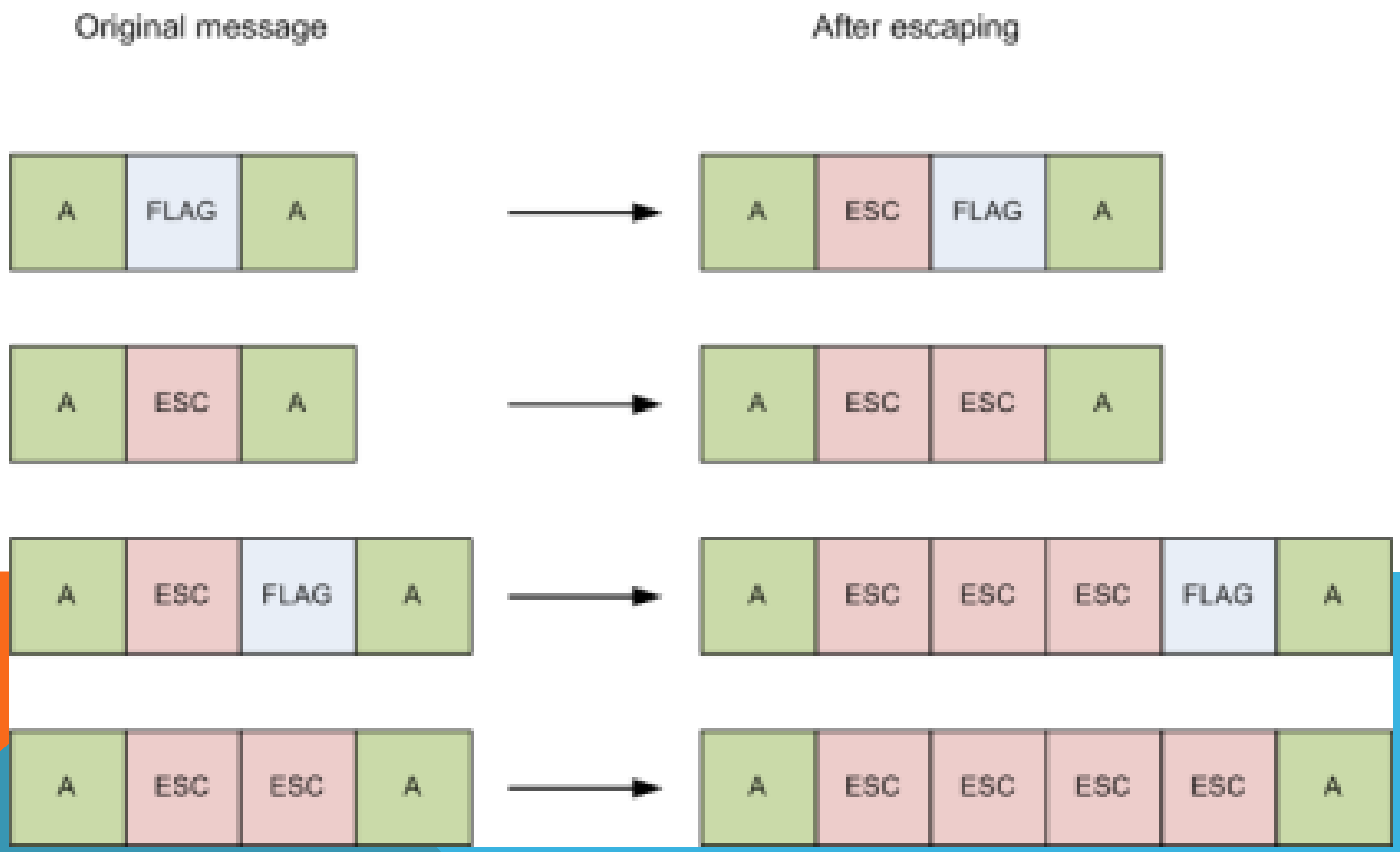
Rules:

- Replace each FLAG in data with ESC FLAG
- Replace each ESC in data with ESC ESC



BYTE STUFFING

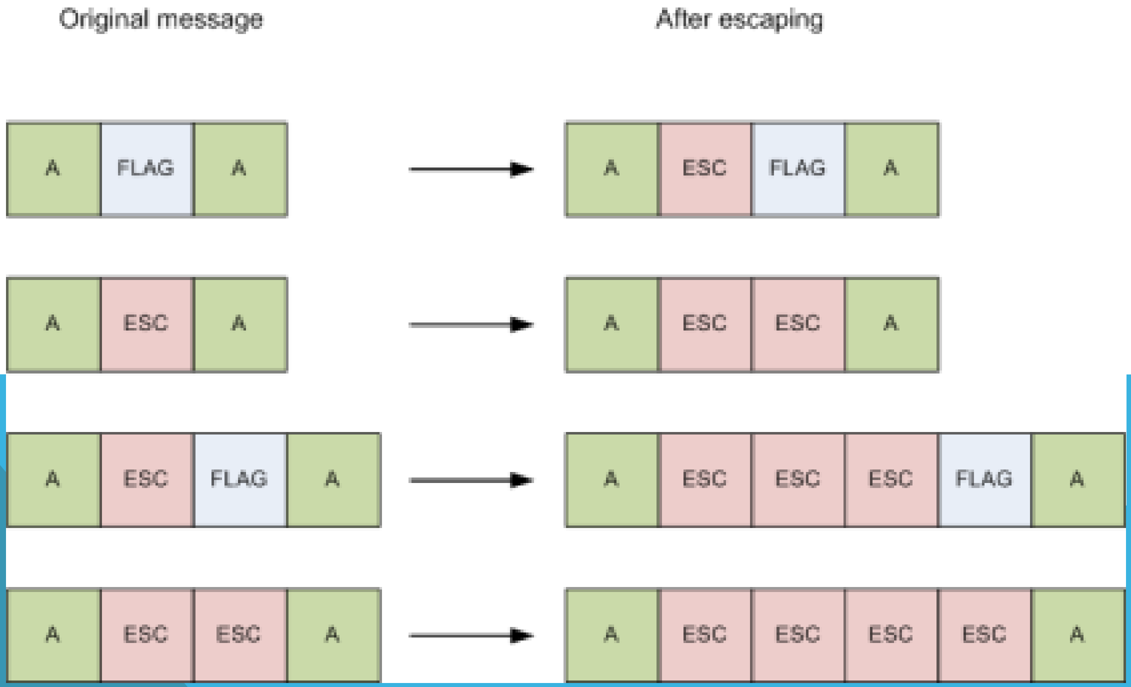
Now any unescaped FLAG is the start/end of a frame



FRAMING – BYTE STUFFING

Frame format

Stuffing examples



BIT STUFFING

Can stuff at the bit level too

- Call a flag six consecutive 1s
- On transmit, after five 1s in the data, insert a 0
- On receive, a 0 after five 1s is deleted

BIT STUFFING

Example:

Data bits 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Transmitted bits with stuffing

BIT STUFFING

So how does it compare with byte stuffing?

Data bits	0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0
Transmitted bits with stuffing	0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0

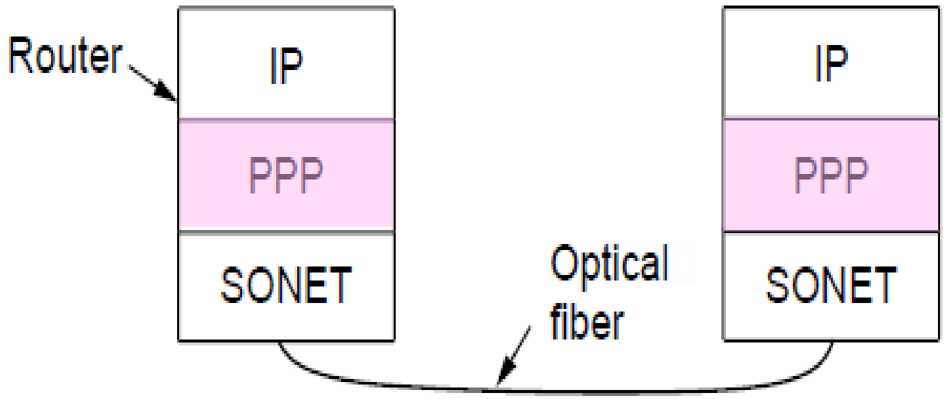
Stuffed bits

EXAMPLE PROTOCOLS

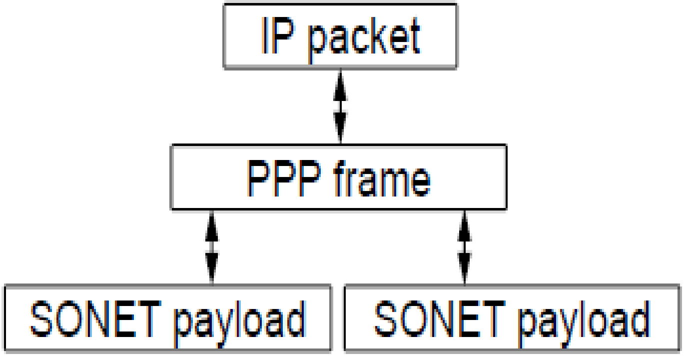
- **PPP is a Point-to-Point Protocol**
  - uses Byte Stuffing
  - used to frame IP packets that are sent in **Synchronous Optical Networking** (SONET) links
  - Flag: FLAG is **0x7E**
- **HDLC (High-level Data Link Control)**
  - uses Bit Stuffing
  - used in serial connections e.g. USB
  - some Point-to-Point networks
  - Flag: 0111 1110
- **Wifi**
  - Preamble and Start Frame Delimiter
  - 80 bits alternation 1s and 0s followed by 1111 0011 1010 0000 (F3A0)
- **Ethernet**
  - Preamble and Start Frame Delimiter
  - 56 bits 80 bits alternation 1s and 0s followed by 10101011 (0xD5)

LINK EXAMPLE: PPP OVER SONET

Think of SONET as a bit stream, and PPP as the framing that carries an IP packet over the link



Protocol stacks

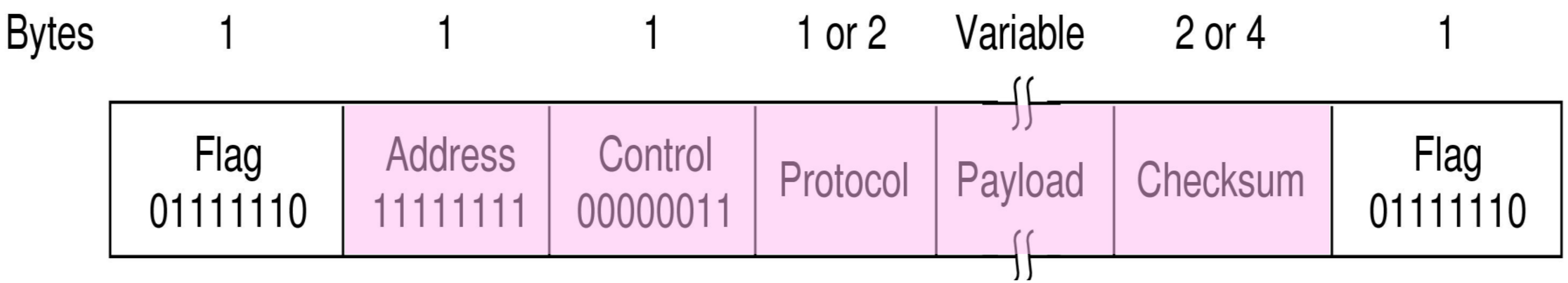


PPP frames may be split over SONET payloads

LINK EXAMPLE: PPP OVER SONET

Framing uses byte stuffing

- FLAG is **0x7E** and ESC is 0x7D



LINK EXAMPLE: PPP OVER SONET

Byte stuffing method:

- To stuff (unstuff) a byte, add (remove) ESC (0x7D), and XOR next byte with 0x20
- Removes FLAG from the contents of the frame completely

Esc: 0x7D : 1111101
Flag: 0x7E : 1111110 → 1111110
0x20 : 0100000 → 0100000 XOR
1011110 → 5E

XOR Rules

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

LINK EXAMPLE: PPP OVER SONET

Byte stuffing method:

- To stuff (unstuff) a byte, add (remove) ESC (0x7D), and XOR byte with 0x20
- Removes FLAG from the contents of the frame completely

  0x7E (flag) appears in the data so we escape it using 0x7D and *XORing* 0x7E with 0x20 – results in 5E

LINK EXAMPLE: PPP OVER SONET

Byte stuffing method:

- To stuff (unstuff) a byte, add (remove) ESC (0x7D), and XOR byte with 0x20
- Removes FLAG from the contents of the frame completely

  0x7E (flag) appears in the data so we escape it using 0x7D and *XORing* 0x7E with 0x20 – results in **5E**

  0x7D(Esc character) appears in the data we escape it using 0x7D and the *XORing* 0x7D (next byte) with 0x20 – results in **5D**

LINK EXAMPLE: PPP OVER SONET

Before Framing:

41	7D	42	7E	50	70	46

After Byte Studding and Framing:

7E	41	7D	5D	42	7D	5E	50	70	46	7E

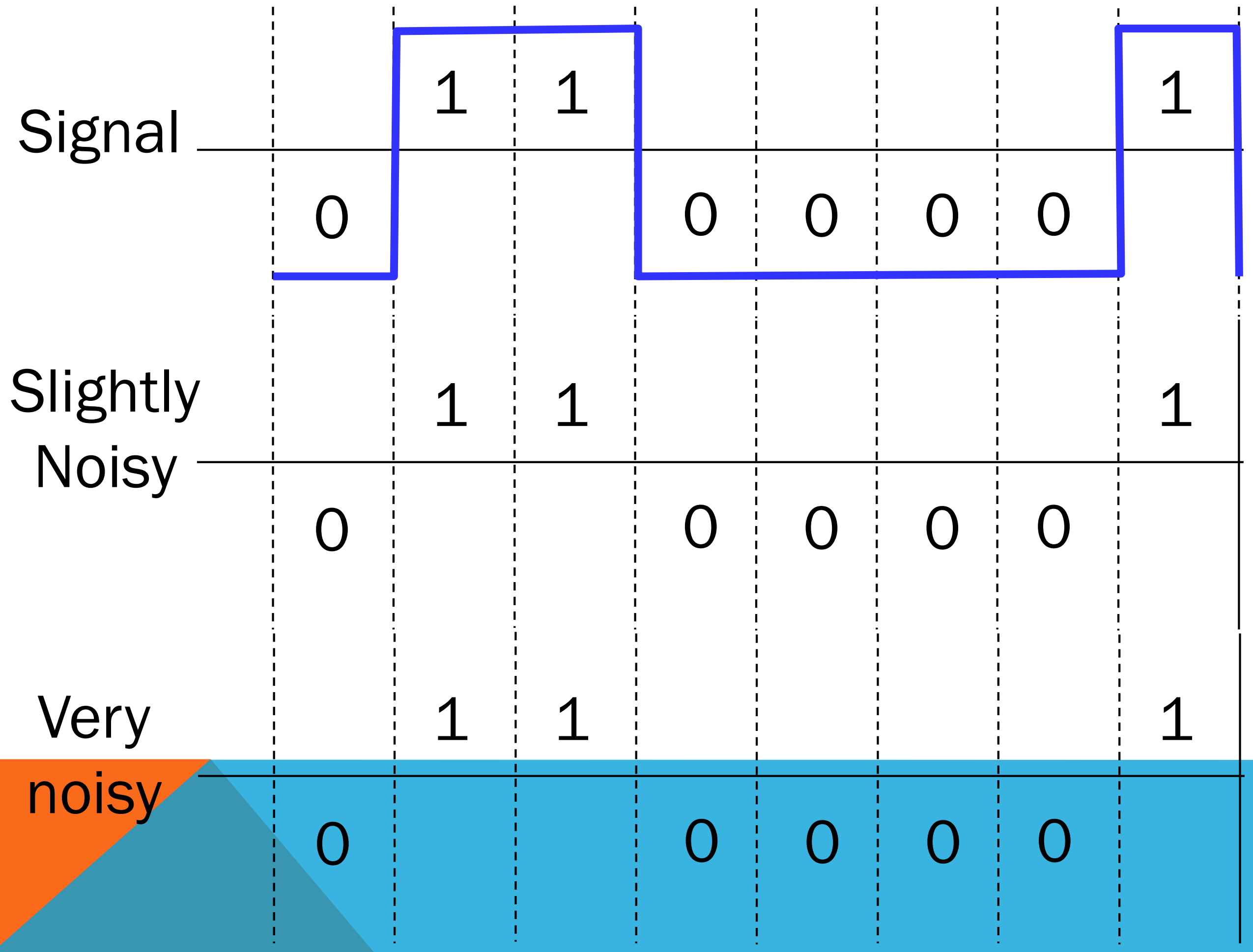
ERROR HANDLING

Reliability is a concern that cuts across the layers

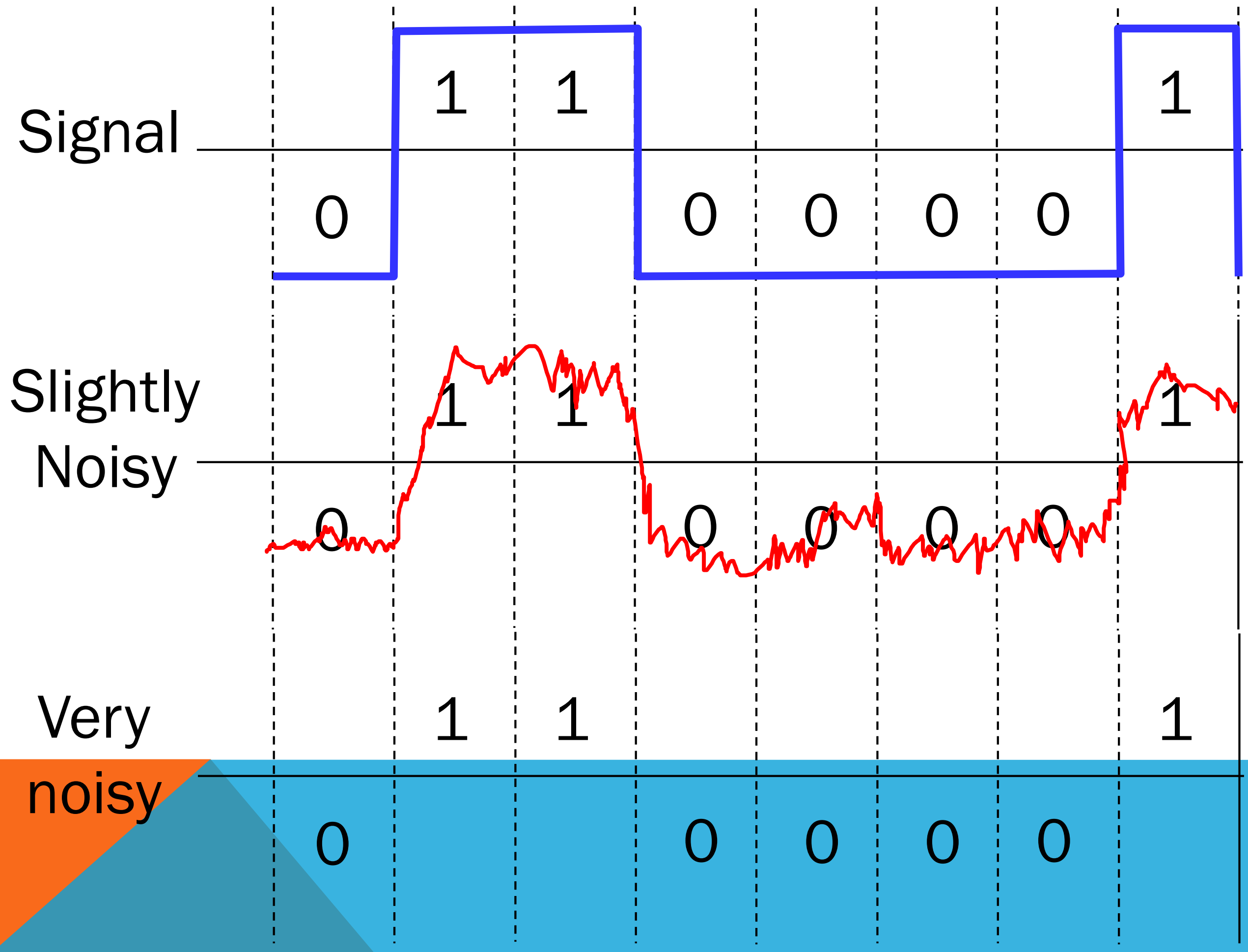
Some bits will be received in error due to noise. What can we do?

- **Detect** errors with codes »
- **Correct** errors with codes »
- **Retransmit** lost frames

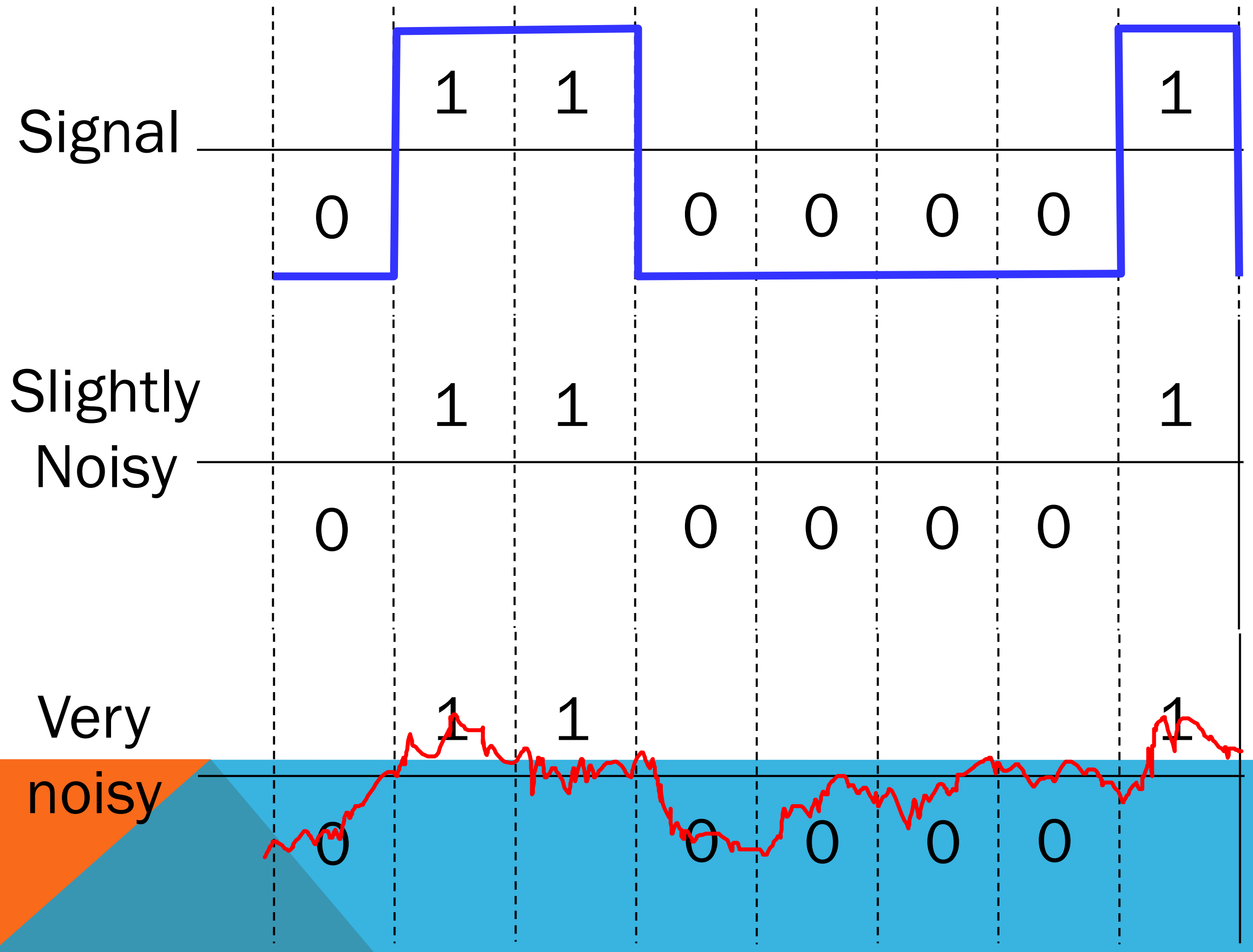
PROBLEM – NOISE MAY FLIP RECEIVED BITS



PROBLEM – NOISE MAY FLIP RECEIVED BITS



PROBLEM – NOISE MAY FLIP RECEIVED BITS



APPROACH – ADD REDUNDANCY

Error <u>detection</u> codes

- Add <u>check bits</u> to the message bits to let some errors be detected

Error <u>correction</u> codes

- Add more <u>check bits</u> to let some errors be corrected

Key issue is how to structure the code to detect many errors with few check bits and modest computation

EXAMPLE

A simple code to handle errors:

- Send two copies. There is an error if different.
  - Message: 010010

How good is this code?

- How many errors can it detect/correct?
- Correct: None
  - 010:011
- Detect: Maybe three
  - If there are 3 differences between the parts
- How many errors will make it fail?

EXAMPLE

A simple code to handle errors:

- Send two copies. There is an error if different.
  - Message: 010010

How good is this code?

- How many errors can it detect/correct?
- Correct: None
  - 010:011
- Detect: Maybe up to three
  - If there are 3 differences between the parts
- How many errors will make it fail? 2 errors will make it fail
  - 011:011

50% of overhead on error detection!

EXAMPLE

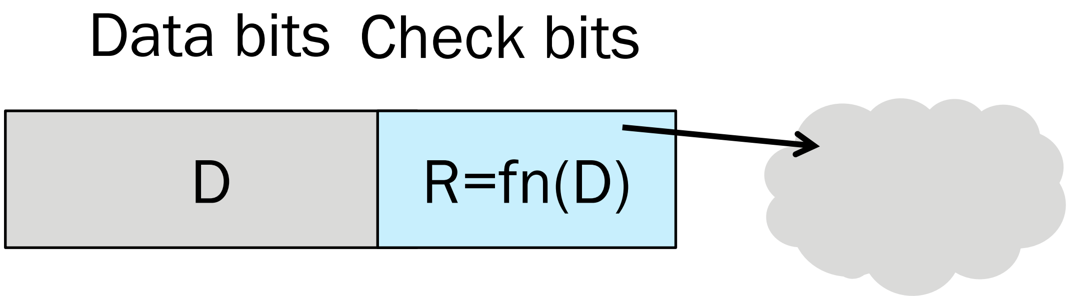
We want to handle more errors with less overhead

- Will look at better codes; they are applied mathematics
- But, they can't handle all errors
- And they focus on accidental errors

USING ERROR CODES

Codeword consists of D data plus R check bits (=systematic block code)

- operate on a **block** of bits of a time – e.g. A frame,
- append check bits



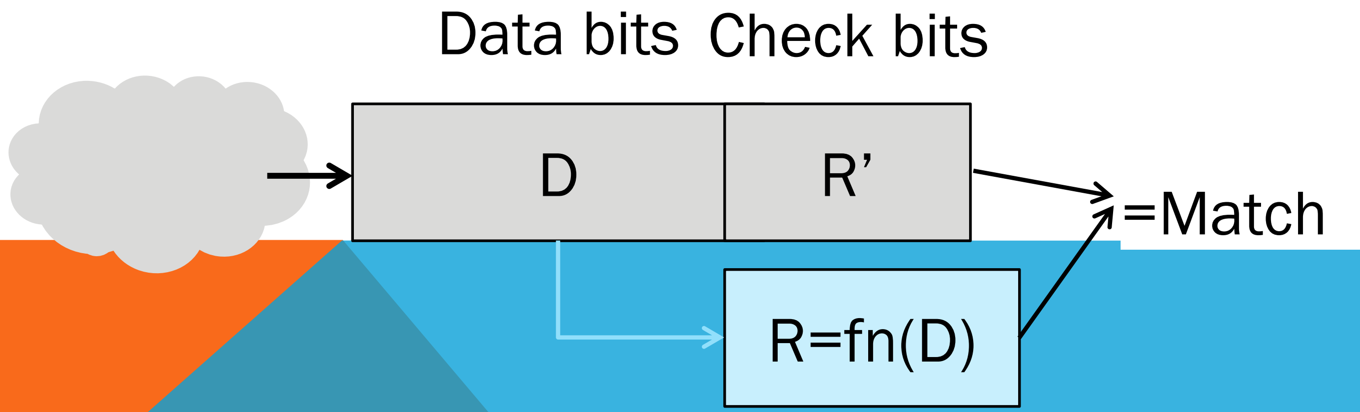
Sender:

- Compute R check bits based on the D data bits; send the codeword of D+R bits

USING ERROR CODES

Receiver:

- Receive D+R bits with unknown **errors**
- Recompute R check bits based on the D data bits; error if R doesn't match R'



HAMMING DISTANCE (HD)

Distance is the number of bit flips needed to change $D+R_1$ to $D+R_2$

- 000 -> 111 = number of bit flips needed is 3 = distance between codewords

<u>Hamming distance</u> of a code is the minimum distance between any pair of codewords (from all codewords).

Above we have just 2 code words and the HD is 3

HAMMING DISTANCE

Error <u>detection</u>:

- For a code of HD d+1, up to d errors will always be detected
- HD= d+1 = 3 -> d = 2
- Can detect up to 2 bit errors

000	111
011	001
110	010
101	100

None are valid code words so will be detected

HAMMING DISTANCE

Error correction:

- For a code of HD 2d+1, up to d errors can always be corrected by mapping to the **closest codeword**
  - HD = 2d+ 1 = 3
  - d = 1

Valid codewords: 000 111

Received Code words: 010

Received Code words: 110

Received Code words: 010