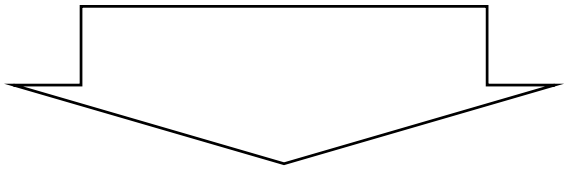
Data Link Layer

Slide Set were original prepared by Dr. Tatsuya Susa

1. Data Link Layer Functionality

- Recall:
  - Frame creation
  - Error detection and/or correction
  - Flow control



Creating the illusion of a reliable link

2. Error Control

- No physical link is perfect
- Bits will be corrupted
- We can either:
  - detect errors and request retransmission
  - or correct errors without retransmission

2.1 Error Detection

- Parity bits
- Polynomial codes or checksums

2.1.1 Polynomial Codes

- Can detect errors on large chunks of data
- Has low overhead
- More robust than parity bit
- Requires the use of a “code polynomial”
  - Example: $x^2 + 1$

Cyclic Redundancy Check

- CRC: Example of a polynomial code
- Procedure (at the sender):
  1. Let r be the degree of the code polynomial c(x). (Both sender and the receiver know the code polynomial.) Append r zero bits to the end of the message bit string. Call the entire bit string $S(x)$
  2. Divide $S(x)$ by the code polynomial c(x) using modulo 2 division.
  3. Subtract the remainder from $S(x)$ using modulo 2 subtraction. (call resulting polynomial t(x).)
  4. Transmit the checksummed message t(x).

Background

- s(x) = f(x)c(x) + remainder
- s(x) - remainder = f(x)c(x) = t(x)
  - sender transmits t(x)
  - note that t(x) is divisible by c(x)
  - if the received sequence at the receiver is not divisible by c(x), error has occurred

- Modulo 2 calculation

  Addition and subtraction are identical to EXCLUSIVE OR.

  No carries

- take modulo 2 addition of 1010 and 0110

$$t(x) = 1 \times x^3 + 0 \times x^2 + 1 \times x + 0 = x^3 + x$$

$$+ \; e(x) = 0 \times x^3 + 1 \times x^2 + 1 \times x + 0 = x^2 + x$$

$$1 \times x^3 + 1 \times x^2 + 0 \times x + 0 = x^3 + x^2$$

- Definition of Exclusive OR

	*E*xclusive OR
0 0	0
0 1	1
1 0	1
1 1	0

Generating a CRC
Example

Message: 1011 $\longrightarrow$ $1 \times x^3 + 0 \times x^2 + 1 \times x + 1$
$= x^3 + x + 1$

Code Polynomial c(x): $x^2 + 1$ (101)

Step 1: Compute $S(x)$

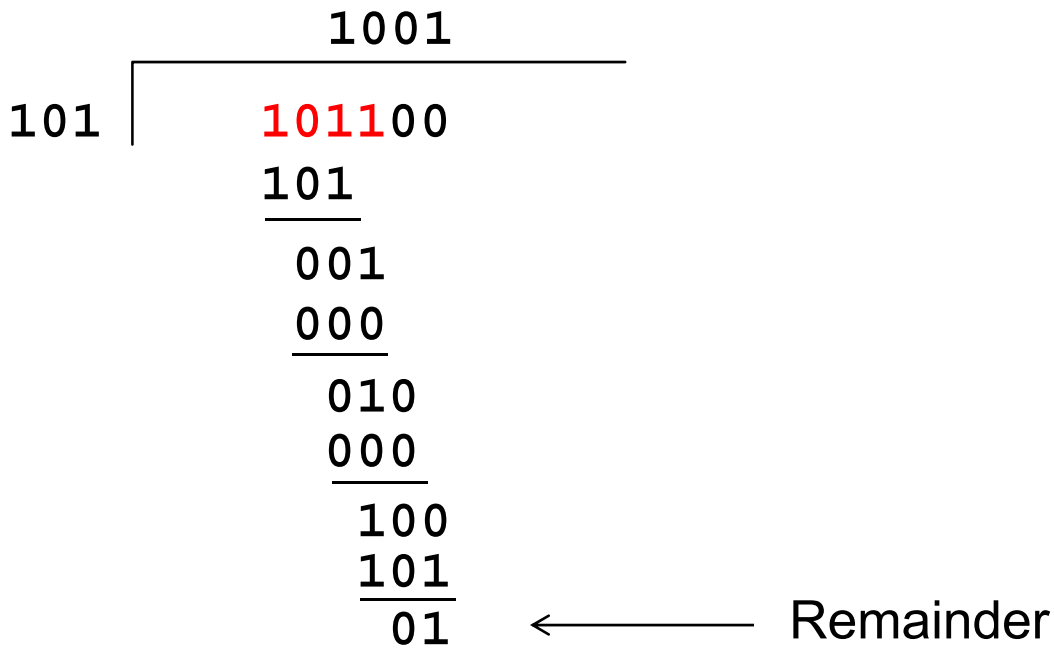
$r = 2$

$S(x) =$ 101100 $(x^5 + x^3 + x^2)$

Generating a CRC

Example (cont'd)

Step 2: Modulo 2 divide



Generating a CRC

Example (cont'd)

Step 3: Modulo 2 subtract the remainder from $S(x)$

```
101100
-   01
------
101101
```

Checksummed Message t(x)

$x^5 + x^3 + x^2 = (x^2 + 1)(x^3 + 1) - 1$

$x^5 + x^3 + x^2 + 1 = (x^2 + 1)(x^3 + 1)$

Transmit $x^5 + x^3 + x^2 + 1$ (101101)

Decoding a CRC

- Procedure (at the receiver)
  - Divide the received message by the code polynomial c(x) using modulo 2 division. If the remainder is zero, there is no error detected.

Choosing a CRC polynomial

- The longer the polynomial, the smaller the probability of undetected error
- Common standard polynomials:
  (1) CRC-12: $x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$
  (2) CRC-16: $x^{16} + x^{15} + x^2 + 1$
  (3) CRC-CCITT: $x^{16} + x^{12} + x^5 + 1$

2.2 Error Correction

- Parity bits and polynomial codes catch errors, but can we correct them without retransmitting information?
- Yes, using Hamming Codes

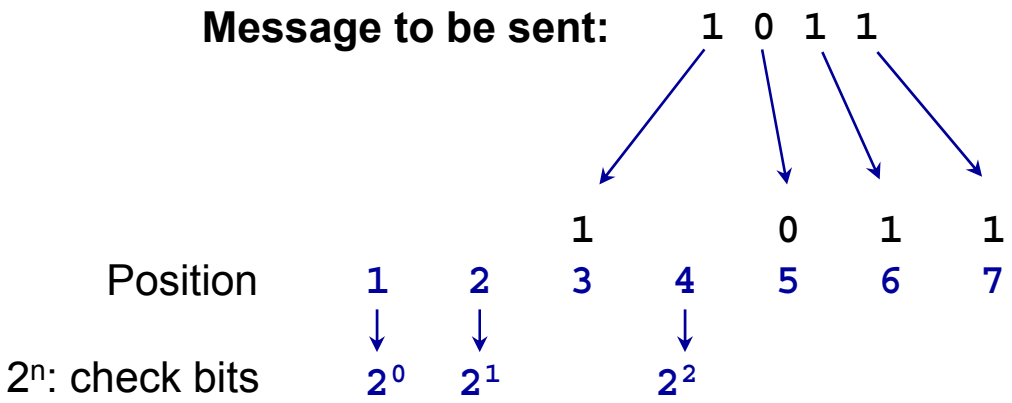
2.2.1 Hamming Codes

- Hamming codes, like polynomial codes, are appended to the transmitted message
- Hamming codes, unlike polynomial codes, contain the information necessary to locate a single bit error

Calculating a Hamming Code

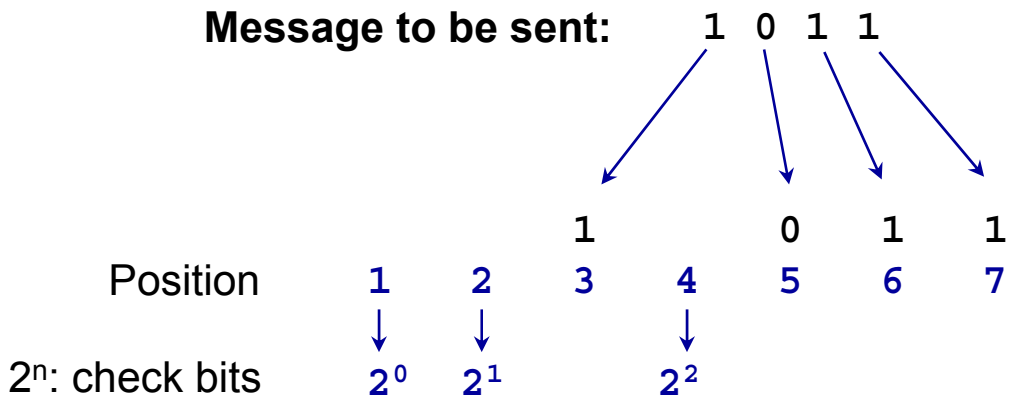
- Procedure:
  - Place message bits in their non-power-of-two Hamming positions
  - Build a table listing the binary representation each each of the message bit positions
  - Calculate the check bits

Hamming Code Example



Hamming Code Example

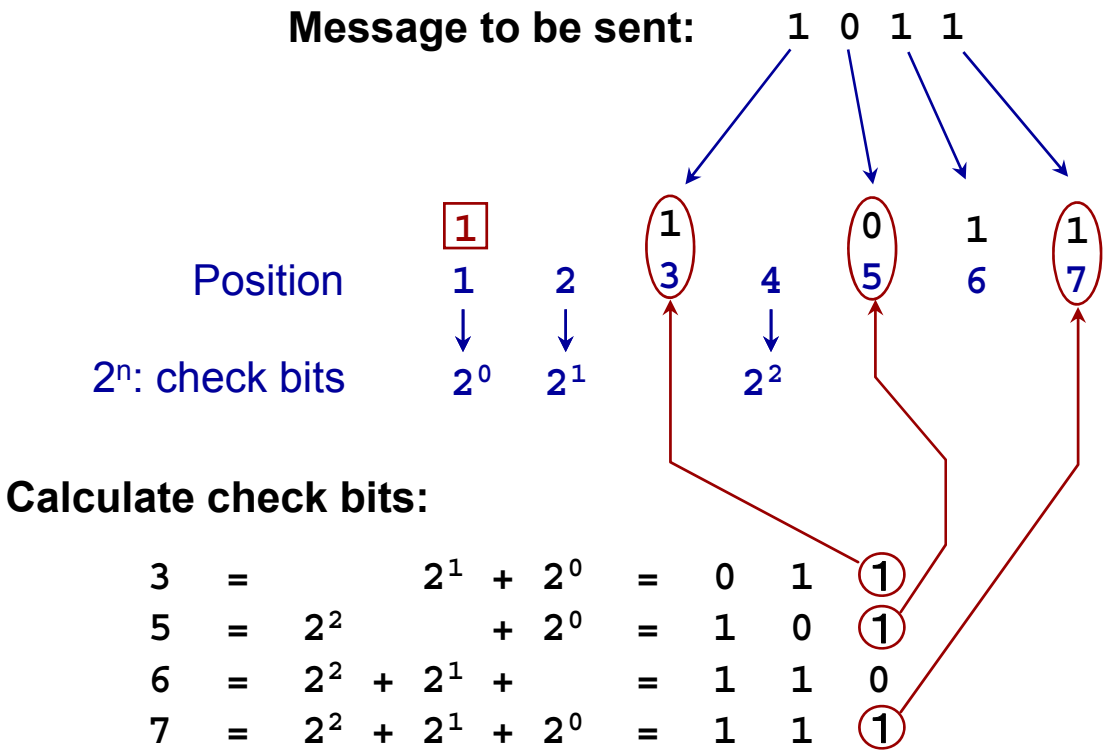
Message to be sent: 1 0 1 1



Calculate check bits:

3	=		2^1	+	2^0	=	0	1	1
5	=	2^2		+	2^0	=	1	0	1
6	=	2^2	+ 2^1	+		=	1	1	0
7	=	2^2	+ 2^1	+	2^0	=	1	1	1

Hamming Code Example



Calculate check bits:

3	=		2^1	+	2^0	=	0	1
5	=	2^2		+	2^0	=	1	0
6	=	2^2	+ 2^1	+		=	1	1
7	=	2^2	+ 2^1	+	2^0	=	1	1

Starting with the 2^0 position:

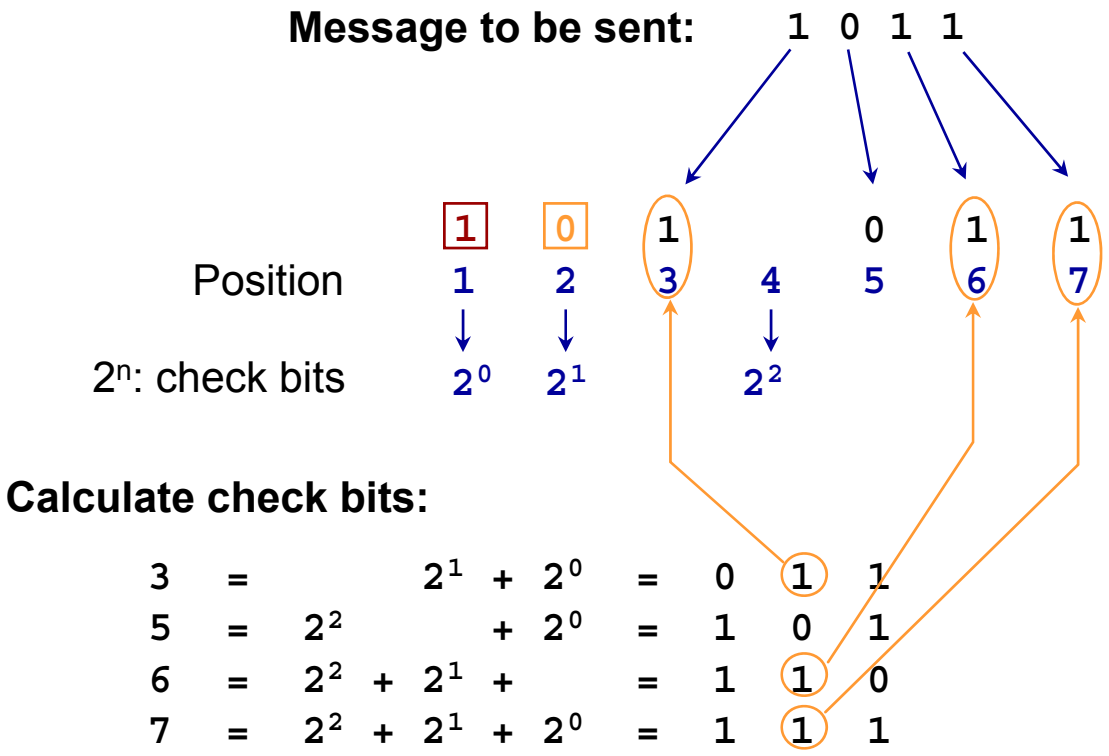
Look at positions with 1's in them

Count the number of 1's in the corresponding message bits

If even, place a 1 in the 2^0 check bit, i.e., use odd parity

Otherwise, place a 0

Hamming Code Example



Calculate check bits:

3	=		2^1	+	2^0	=	0	1	1
5	=	2^2		+	2^0	=	1	0	1
6	=	2^2	+ 2^1	+		=	1	1	0
7	=	2^2	+ 2^1	+	2^0	=	1	1	1

Repeat with the 2^1 position:

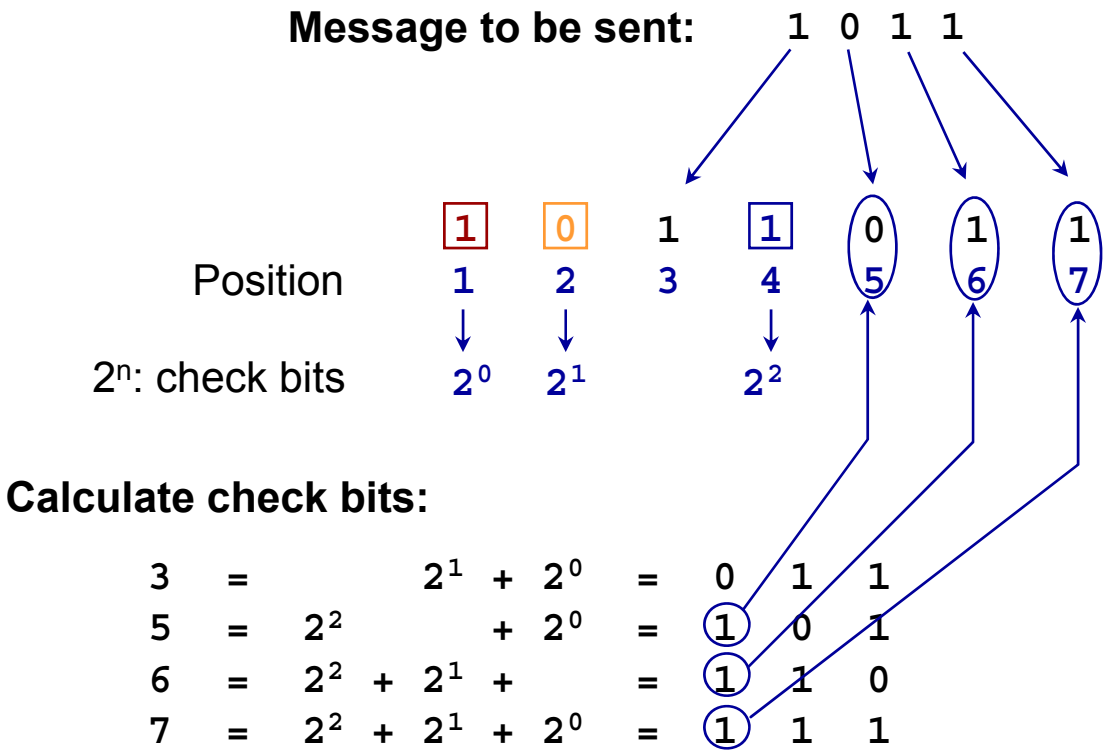
Look at positions those positions with 1's in them

Count the number of 1's in the corresponding message bits

If even, place a 1 in the 2^1 check bit

Otherwise, place a 0

Hamming Code Example



Repeat with the 2^2 position:

Look at positions those positions with 1's in them

Count the number of 1's in the corresponding message bits

If even, place a 1 in the 2^2 check bit

Otherwise, place a 0

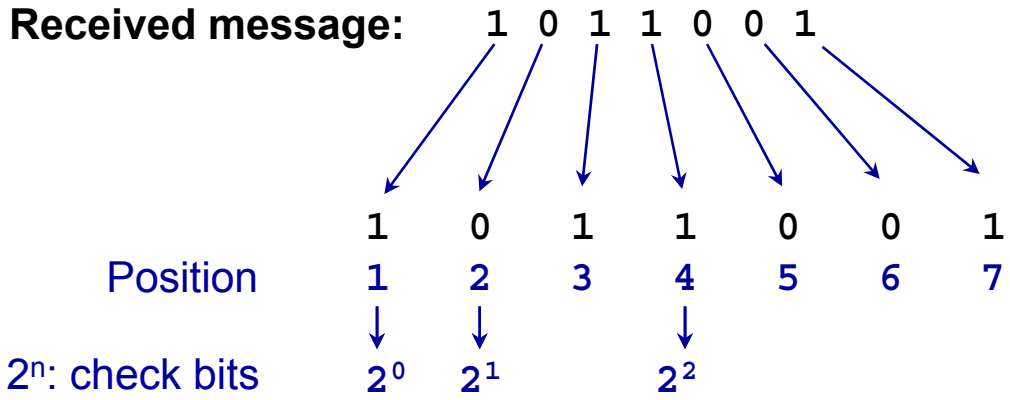
Hamming Code Example

Original message = 1011

Sent message = 1011011

Now, how do we check for a single-bit error in the sent message using the Hamming code?

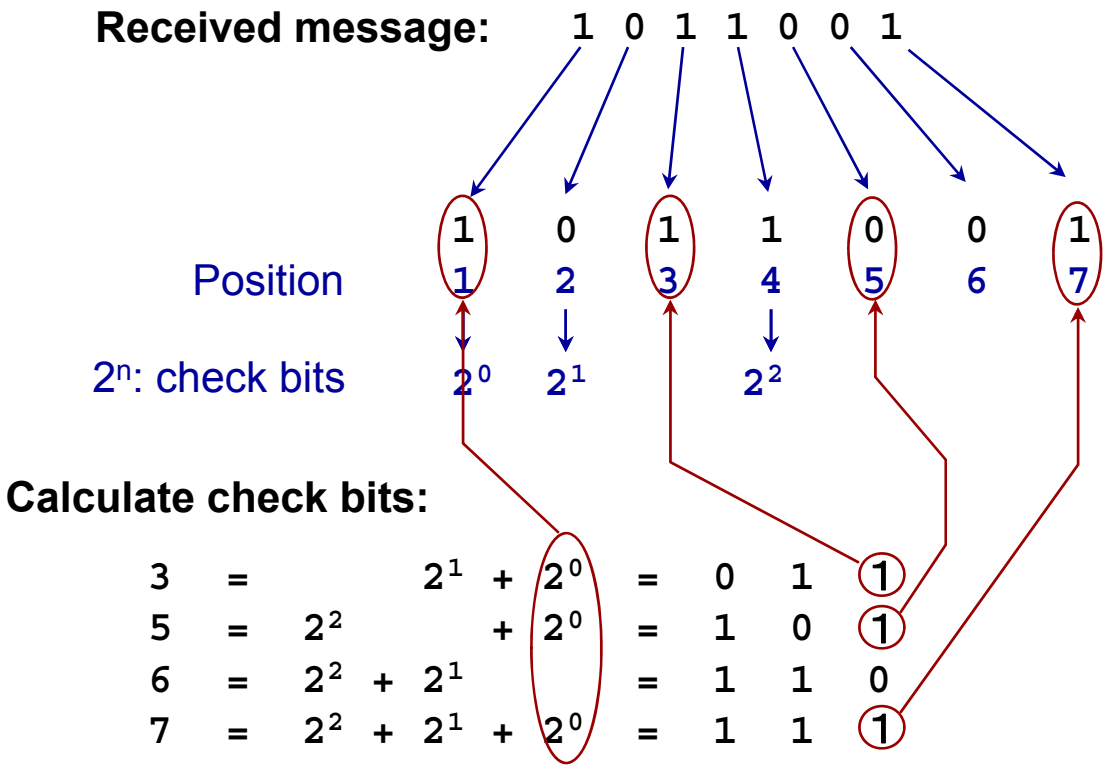
Using Hamming Codes to Correct Single-Bit Errors



Calculate check bits:

3	=		2^1	+	2^0	=	0	1	1
5	=	2^2		+	2^0	=	1	0	1
6	=	2^2	+ 2^1			=	1	1	0
7	=	2^2	+ 2^1	+	2^0	=	1	1	1

Using Hamming Codes to Correct Single-Bit Errors



Odd parity: No error in bits 1, 3, 5, 7

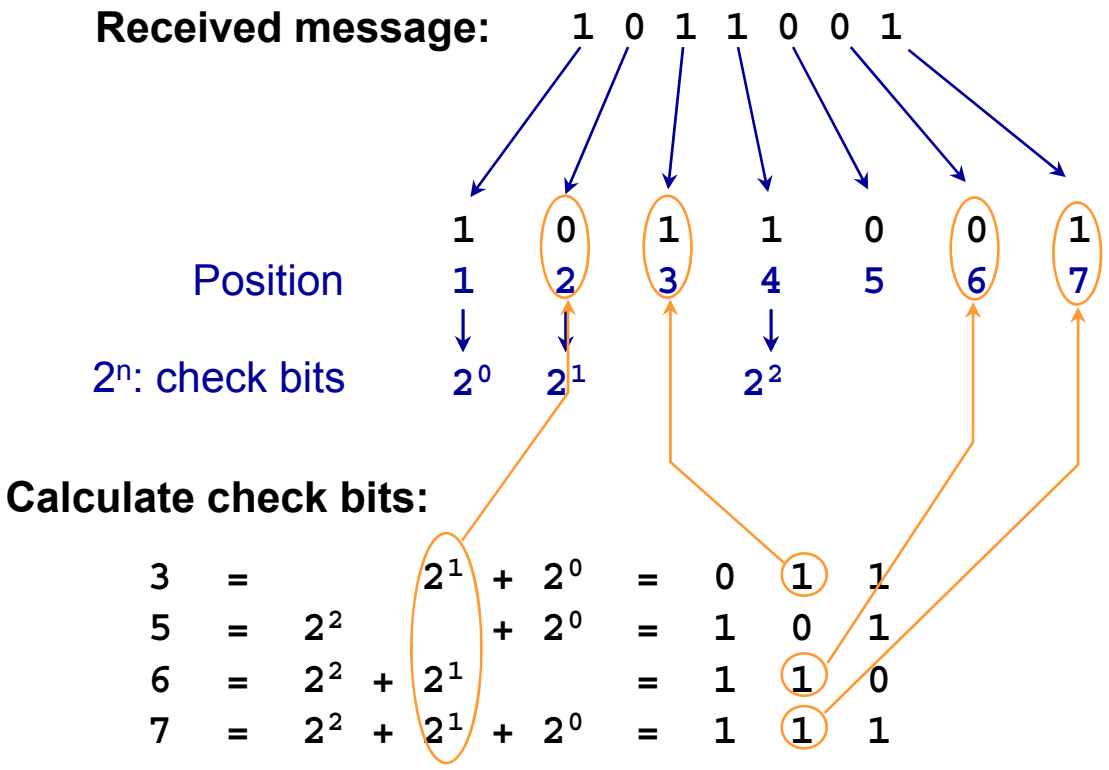
Starting with the 2^0 position:

Look at positions with 1's in them

Count the number of 1's in both the corresponding message bits and the 2^0 check bit and compute the parity.

If even parity, there is an error in one of the four bits that were checked.

Using Hamming Codes to Correct Single-Bit Errors



Even parity: ERROR in bit 2, 3, 6 or 7!

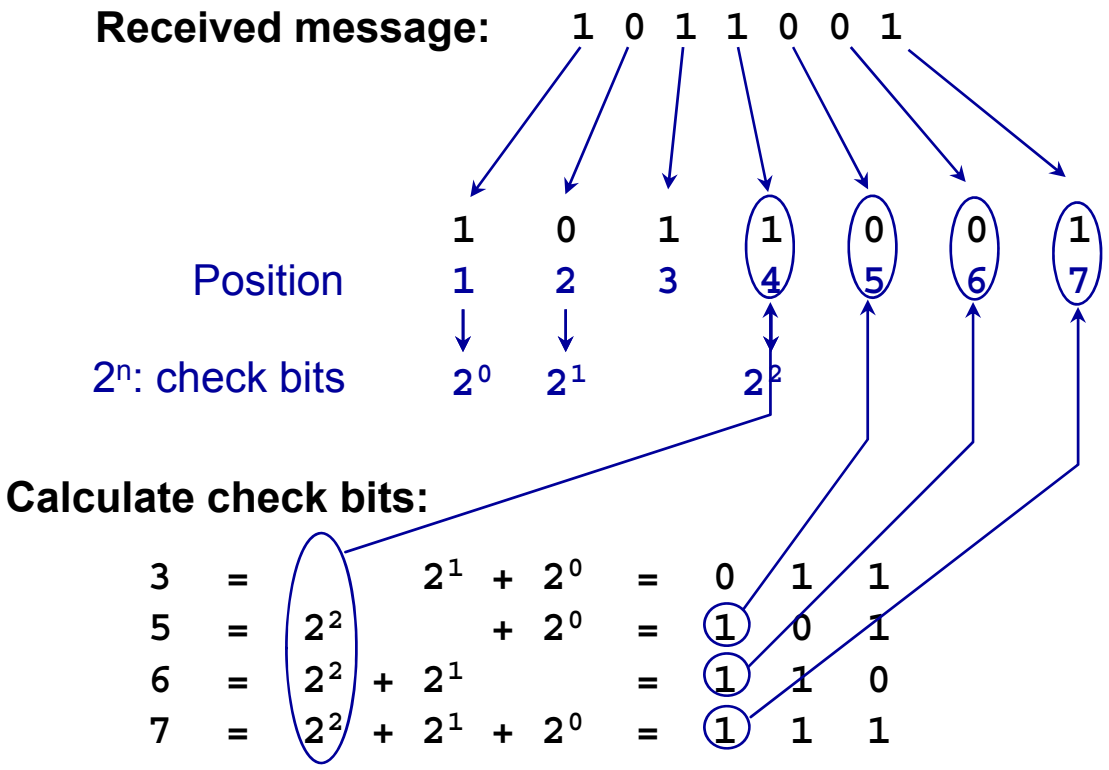
Repeat with the 2^1 position:

Look at positions with 1's in them

Count the number of 1's in both the corresponding message bits and the 2^1 check bit and compute the parity.

If even parity, there is an error in one of the four bits that were checked.

Using Hamming Codes to Correct Single-Bit Errors



Even parity: ERROR in bit 4, 5, 6 or 7!

Repeat with the 2^2 position:

Look at positions with 1's in them

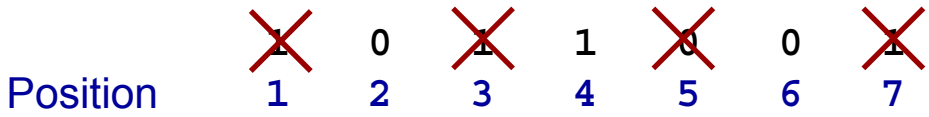
Count the number of 1's in both the corresponding message bits and the 2^2 check bit and compute the parity.

If even parity, there is an error in one of the four bits that were checked.

Finding the error's location

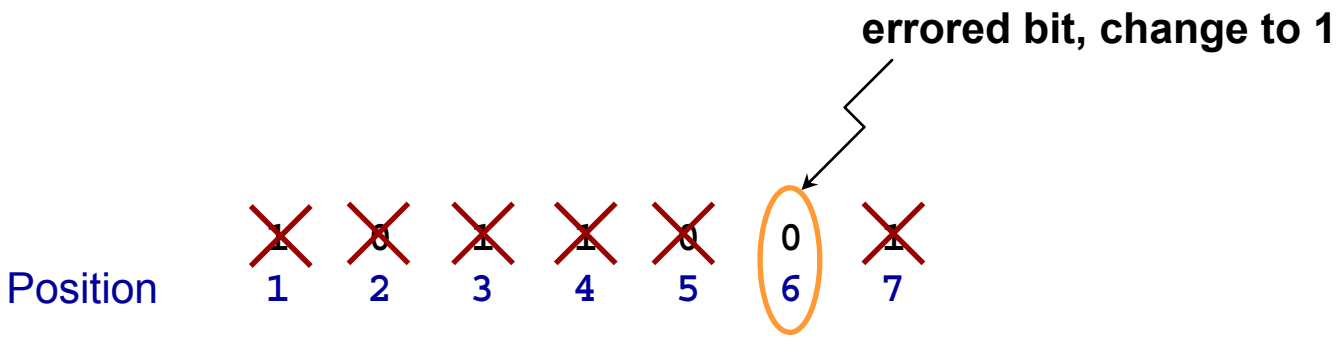
	1	0	1	1	0	0	1
Position	1	2	3	4	5	6	7

Finding the error's location



No error in bits 1, 3, 5, 7

Finding the error's location



No error in bits 1, 3, 5, 7

ERROR in bit 2, 3, 6 or 7

ERROR in bit 4, 5, 6 or 7

Error must be in bit 6 because bits 2, 5, 7 are correct, and all the remaining information agrees on bit 6

Finding the error's location

An Easier Alternative to the Last Slide

3	=		2^1	+	2^0	=	0	1	1
5	=	2^2		+	2^0	=	1	0	1
6	=	2^2	+ 2^1			=	1	1	0
7	=	2^2	+ 2^1	+	2^0	=	1	1	1

E E NE

↓ ↓ ↓

1 1 0 = 6

E = error in column

NE = no error in column

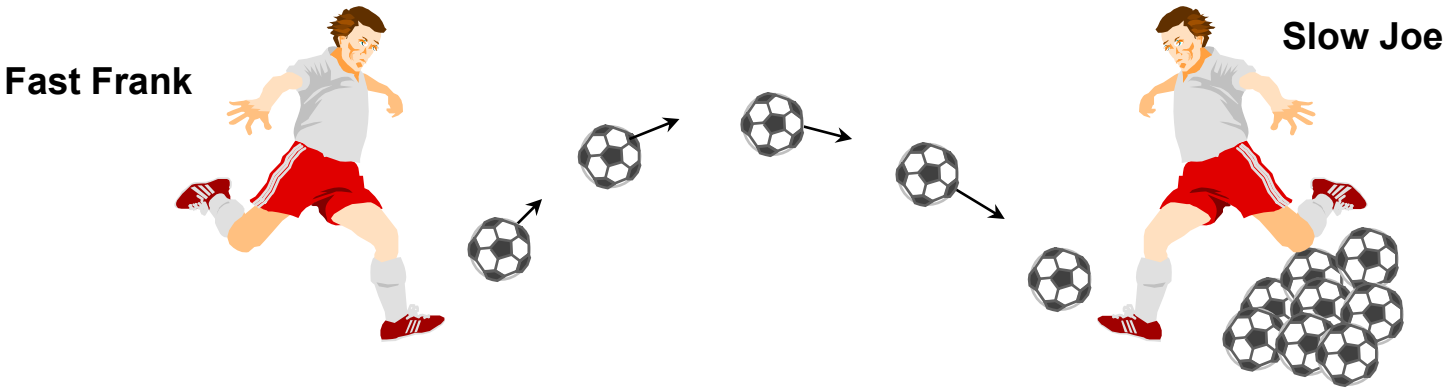
Hamming Codes

- Hamming codes can be used to locate and correct a single-bit error
- If more than one bit is in error, then a Hamming code cannot correct it
- Hamming codes, like parity bits, are only useful on short messages

3. Flow Control

- What happens if the sender tries to transmit faster than the receiver can accept?
- Data will be lost unless flow control is implemented

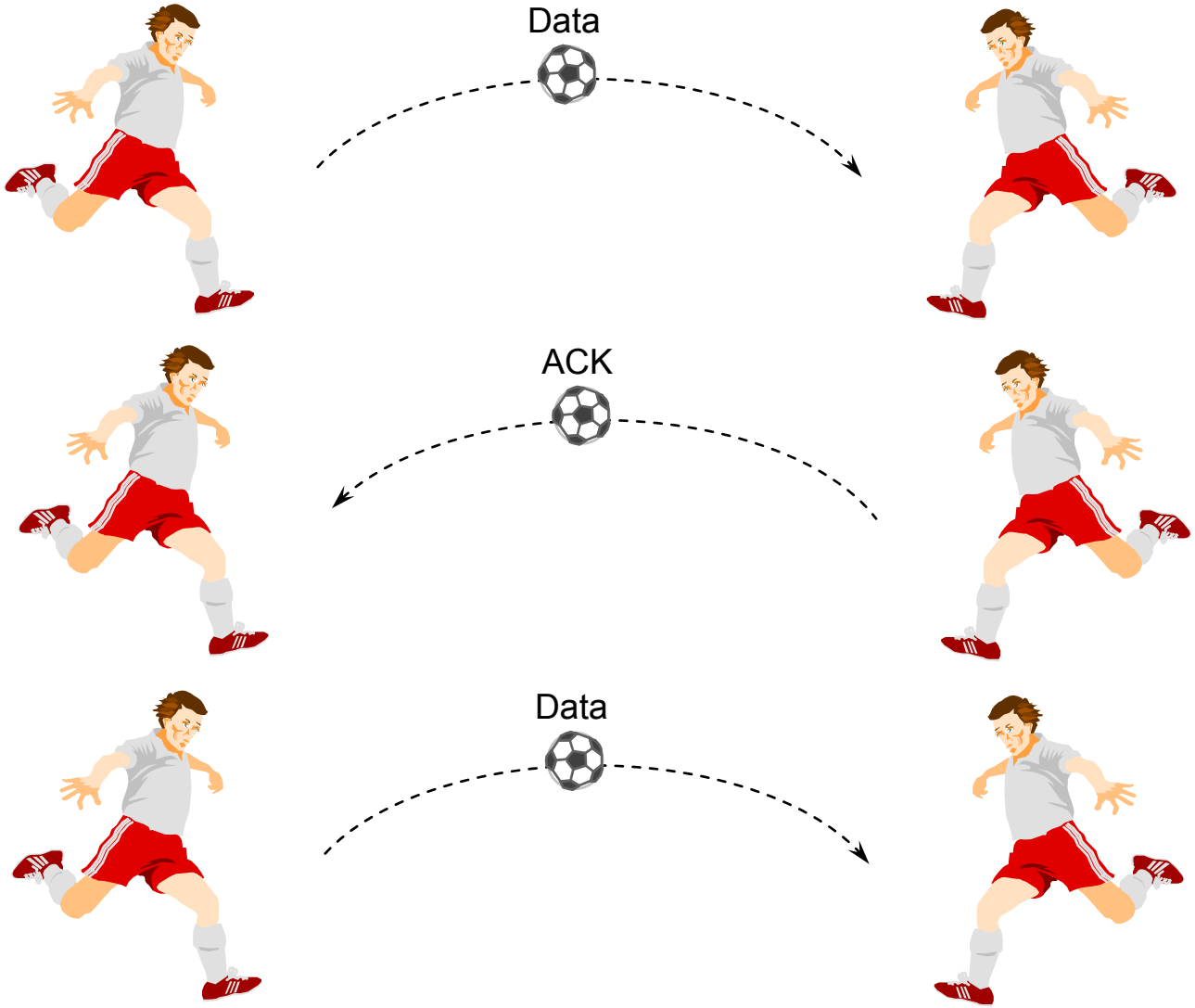
Controlling the Flow of Data



Stop and Wait with Noiseless Channels *(cont'd)*

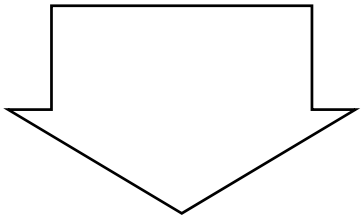
- Solution: Stop-and-Wait
  - The receiver sends an acknowledgement frame telling the sender to transmit the next data frame.
  - The sender waits for the ACK, and if the ACK comes, it transmits the next data frame.

Stop and Wait with Noiseless Channels *(cont'd)*



3.1 Full Duplex Flow Control Protocols

Data frames are transmitted in both directions



Sliding Window

Flow Control Protocols

Sliding Window Protocols
Definitions

Sequence Number: Each frame is assigned a sequence number that is incremented as each frame is transmitted

Sender's Window: Keeps sequence numbers of frames that have been sent but not yet acknowledged

Sender Window size: The number of frames the sender may transmit before receiving ACKs

Receiver's Window: Keeps sequence numbers of frames that the receiver is allowed to accept

Receiver Window size: The maximum number of frames the receiver may receive out of order

Sliding Window Protocols

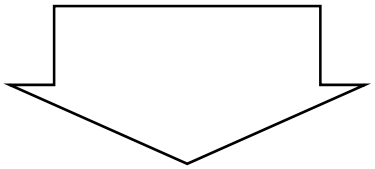
General Remarks

- The sending and receiving windows do not have to be the same size
- Any frame which falls outside the receiving window is discarded at the receiver
- Unlike the sender's window, the receiver's window always remains at its initial size

Sliding Window Protocols

Piggybacking Acknowledgements

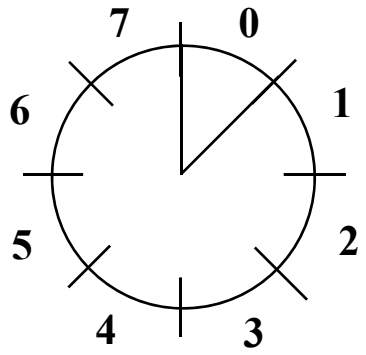
Since we have full duplex transmission, we can "piggyback" an ACK onto the header of an outgoing data frame to make better use of the channel



When a data frame arrives at an IMP, instead of immediately sending a separate ACK frame, the IMP waits until it is passed the next data frame to send. The acknowledgement is attached to the outgoing data frame.

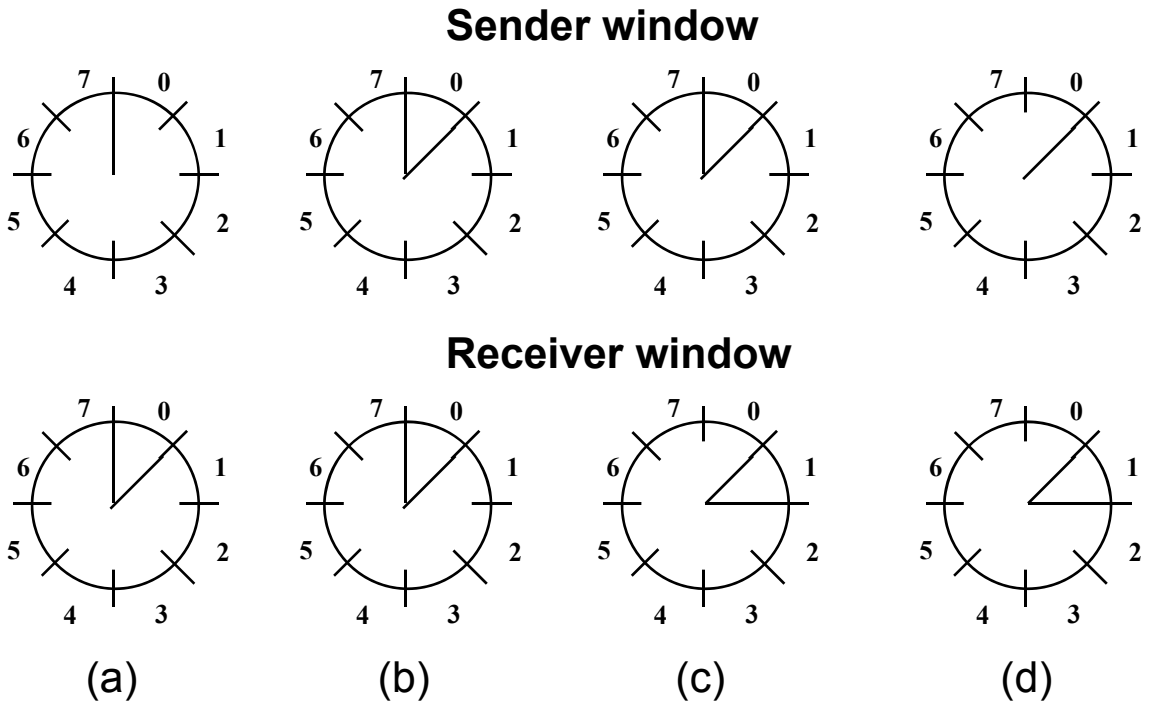
3.1.1 Simple Sliding Window with Window Size of 1

A sliding window with a maximum window size of 1 frame



Window for a 3-bit sequence number

Sliding Window example



(a) Initial state, no frames transmitted
(b) Sender transmits frame 0
(c) Receiver receives frame 0 and ACKs
(d) Sender receives ACK

Simple Sliding Window with Window size 1 *(cont'd)*

This protocol behaves identically to stop and wait for a noisy channel

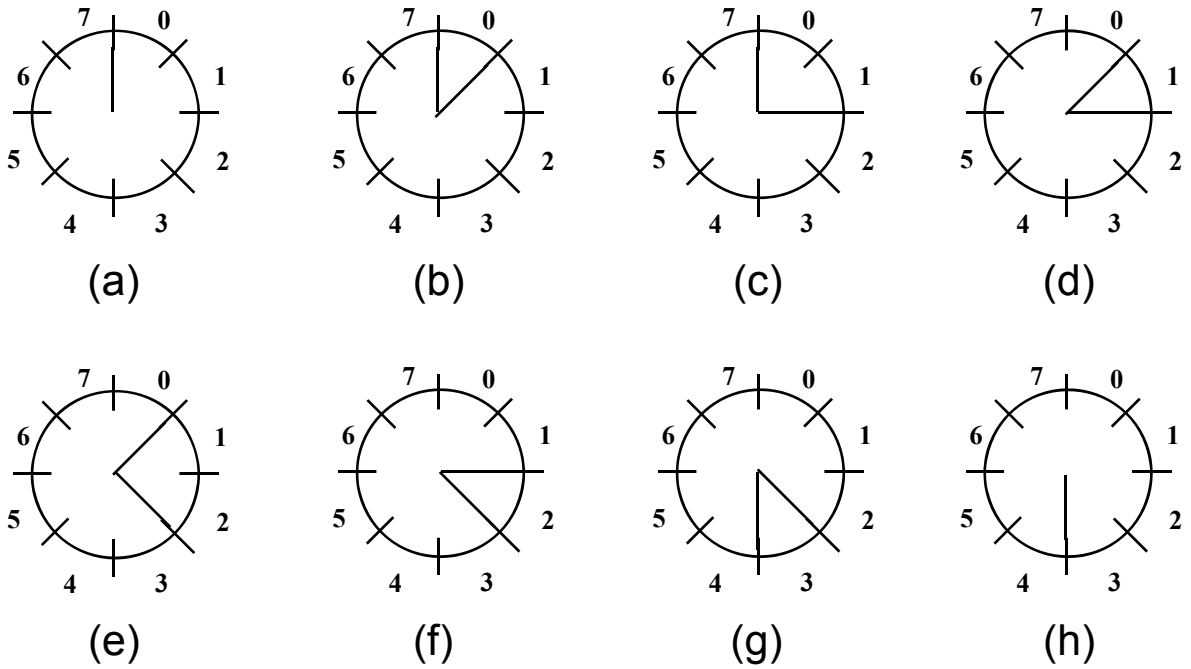
3.1.2 Sliding Window with Window Size *W*

With a window size of 1, the sender waits for an ACK before sending another frame

With a window size of W, the sender can transmit up to W frames before "being blocked"

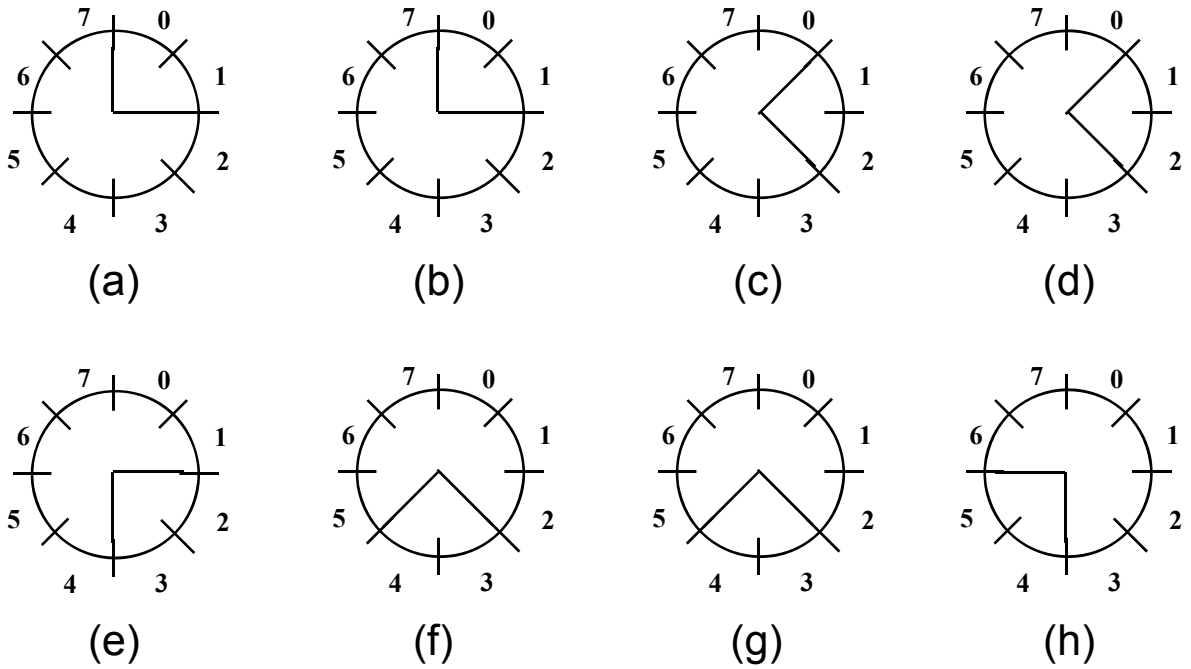
We call using larger window sizes **Pipelining**

Sender-Side Window with Window Size W=2



(a) Initial window state	(e) Send frame 2
(b) Send frame 0	(f) ACK for frame 1 arrives
(c) Send frame 1	(g) ACK for frame 2 arrives, send frame 3
(d) ACK for frame 0 arrives	(h) ACK for frame 3 arrives

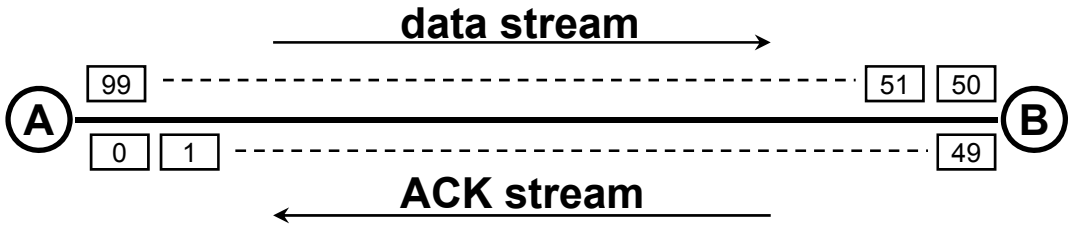
Receiver-Side Window with Window Size W=2



(a) Initial window state	(e) Frame 1 arrives, ACK frame 1
(b) Nothing happens	(f) Frame 2 arrives, ACK frame 2
(c) Frame 0 arrives, ACK frame 0	(g) Nothing happens
(d) Nothing happens	(h) Frame 3 arrives, ACK frame 3

Why do Pipelining?

In other words, why have a window size greater than 1?



By allowing several frames into the network before receiving acknowledgement, pipelining keeps the transmission line from being idle

What about Errors?

What if a data or acknowledgement frame is lost when using a sliding window protocol?

Two Solutions:

Go Back N

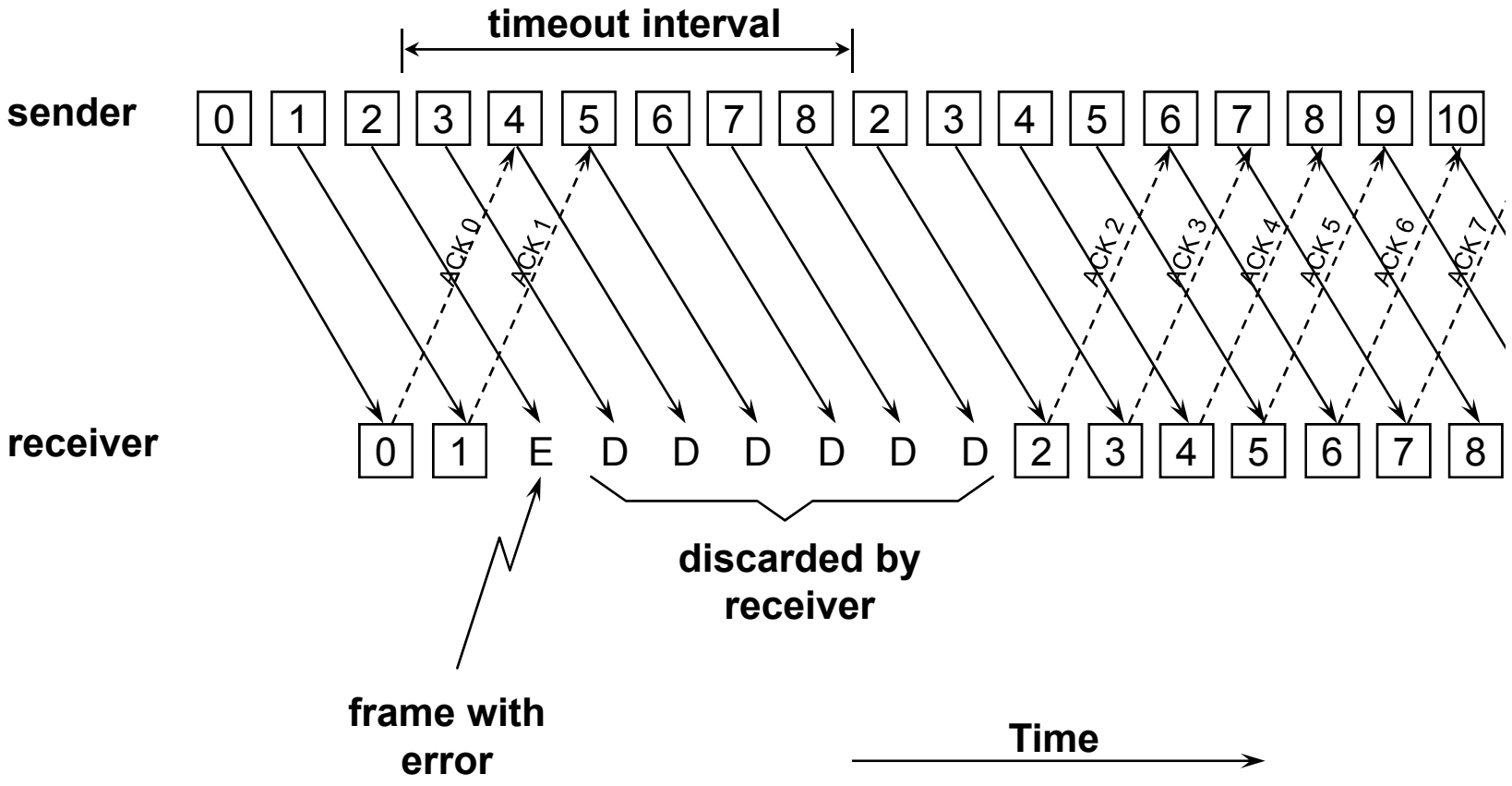
Selective Repeat

- One very important note about acknowledgement
  - Ack for frame n = I am expecting frame n+1 (not “I received fame n”)

3.1.3 Sliding Window with Go Back N

- When the receiver notices a missing or erroneous frame, it simply discards all frames with greater sequence numbers and sends no ACK
- The sender will eventually time out and retransmit all the frames in its sending window

Go Back N


timeout interval
sender
receiver
0 1 2 3 4 5 6 7 8 2 3 4 5 6 7 8 9 10
ACK 0
ACK 1
ACK 2
ACK 3
ACK 4
ACK 5
ACK 6
ACK 7
0 1 E D D D D D D 2 3 4 5 6 7 8
discarded by receiver
frame with error
Time

Go Back N *(cont'd)*

Go Back N can recover from erroneous or missing packets

But…

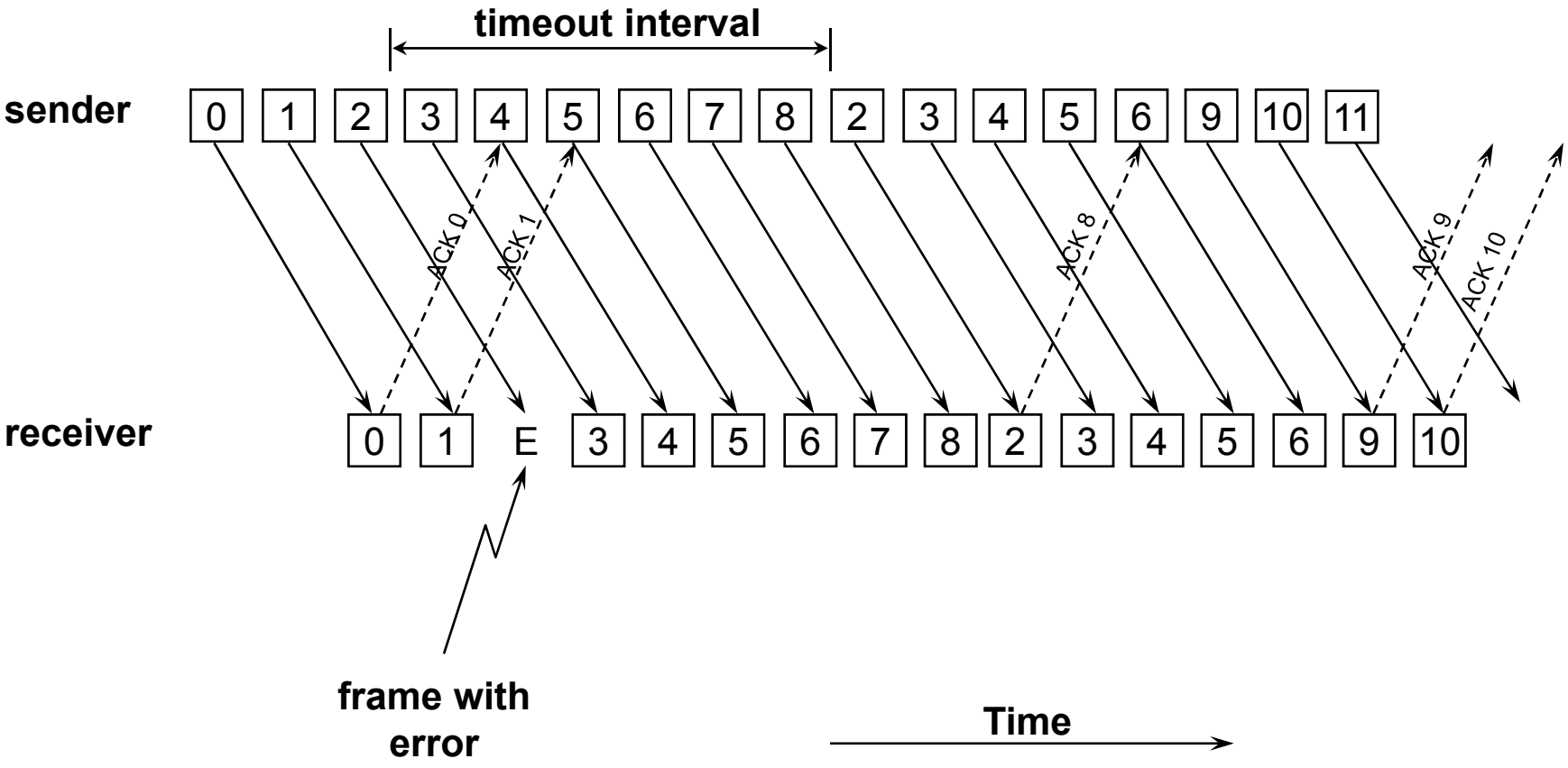
It is wasteful. If there are a lot of errors, the sender will spend most of its time retransmitting useless information

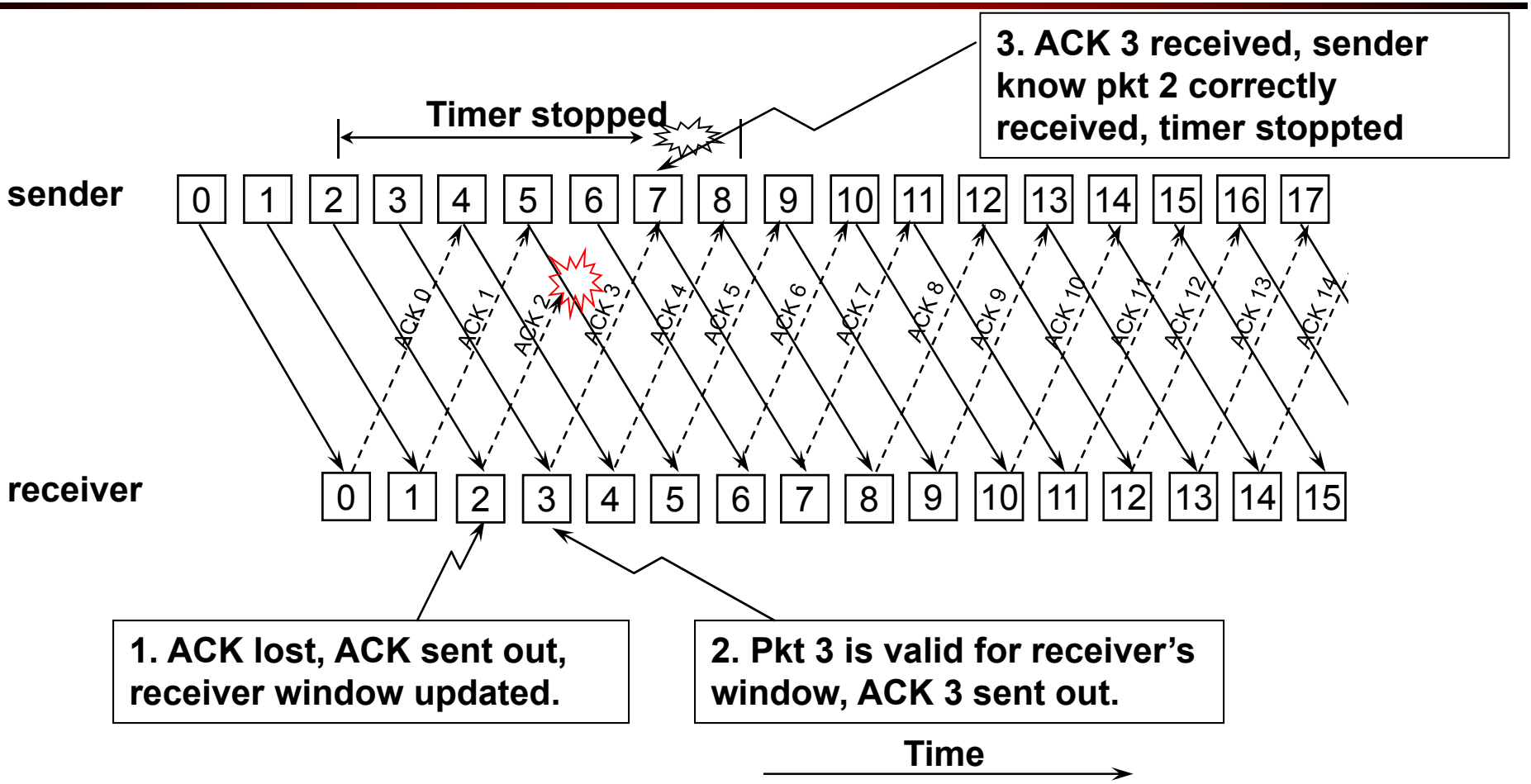
3.1.4 Sliding Window with Selective Repeat

The sender retransmits only the frame with errors

- The receiver stores all the correct frames that arrive following the bad one. (Note that this requires a significant amount of buffer space at the receiver.)
- When the sender notices that something is wrong, it just retransmits the one bad frame, not all its successors.

Selective Repeat




3. ACK 3 received, sender know pkt 2 correctly received, timer stoppted
Timer stopped
sender
receiver
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
ACK 0
ACK 1
ACK 2
ACK 3
ACK 4
ACK 5
ACK 6
ACK 7
ACK 8
ACK 9
ACK 10
ACK 11
ACK 12
ACK 13
ACK 14
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1. ACK lost, ACK sent out, receiver window updated.
2. Pkt 3 is valid for receiver's window, ACK 3 sent out.
Time

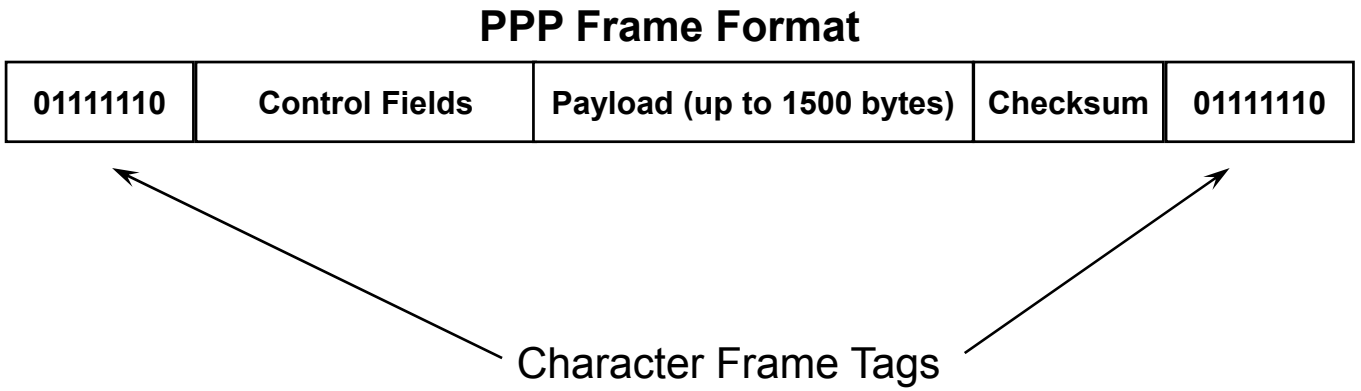
- In this scheme, every time a receiver receives a frame, it sends an acknowledgement which contains the sequence number of the next frame expected

4. Bit-Oriented and Character-Oriented Protocols

1. Character Oriented Protocol
   - Basic unit: character
   - Frame length is a multiple of character size
   - Example: Internet
2. Bit Oriented Protocol
   - Basic unit: Bit
   - Frame length is not a multiple of character size
   - Example: HDLC (High-level Data Link Control)

4.1 Example Character-Oriented Protocol

- PPP = Point-to-Point Protocol
- Used widely in the Internet
- PPP uses frame tags with character stuffing



Character Oriented Protocols
(cont'd)

With character oriented protocols, a frame is composed of characters in some character code (e.g., ASCII, EBCDIC, UNICODE)

A computer with 9-bit characters cannot send arbitrary messages in ASCII code. They must be chopped and repacked into units of 8 bits.

Bit oriented protocols do not require such chopping up and repackaging of characters.

4.2 Example Bit Oriented Protocol

- HDLC = High-level Data Link Control
- Used in X.25 networks
- HDLC uses frame tags with bit stuffing