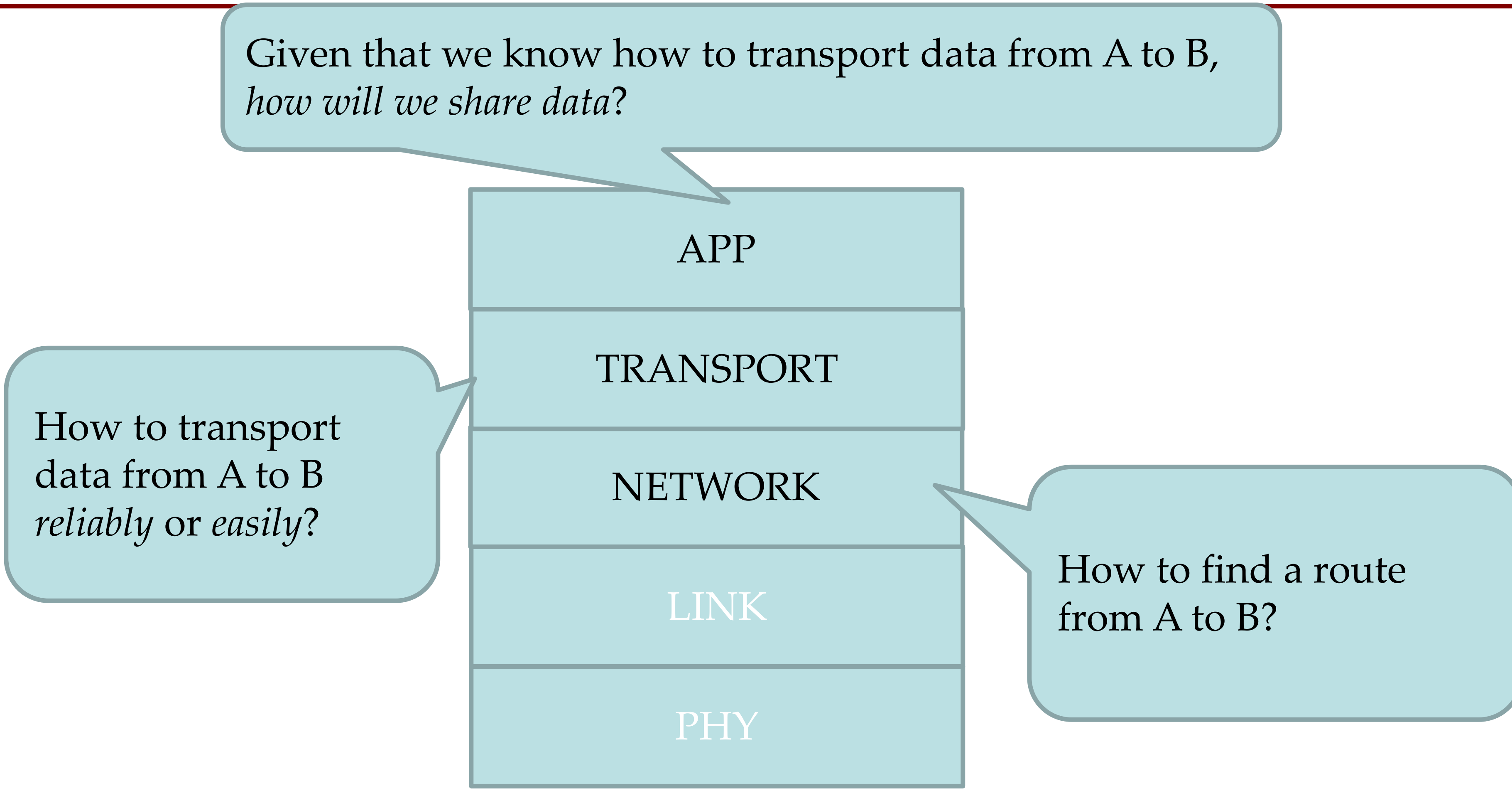
COSC264
Introduction to Computer Networks and the Internet

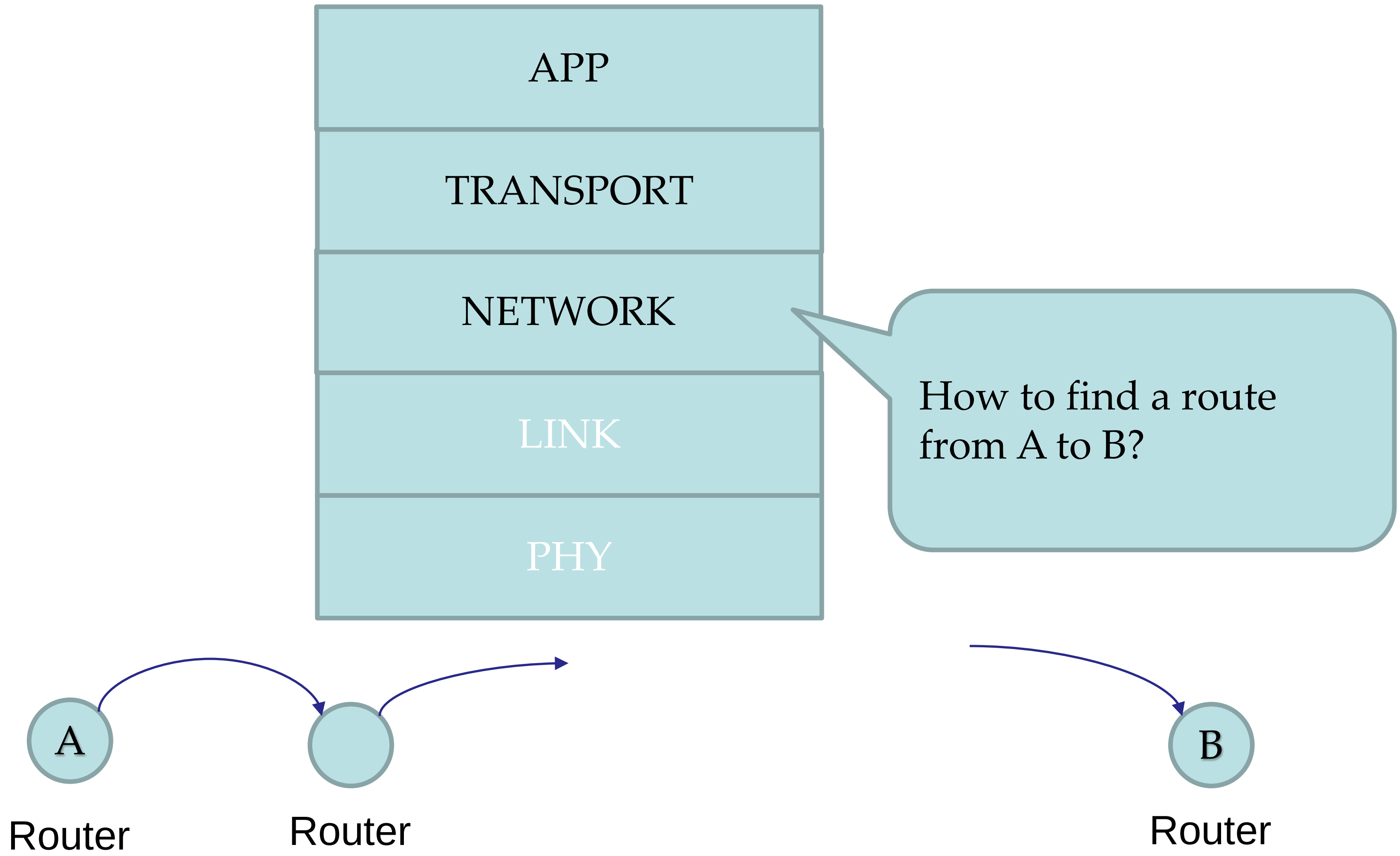
Reliable data transfer: Error Detection and Correction

Dr. Barry Wu
Wireless Research Centre
University of Canterbury
barry.wu@canterbury.ac.nz

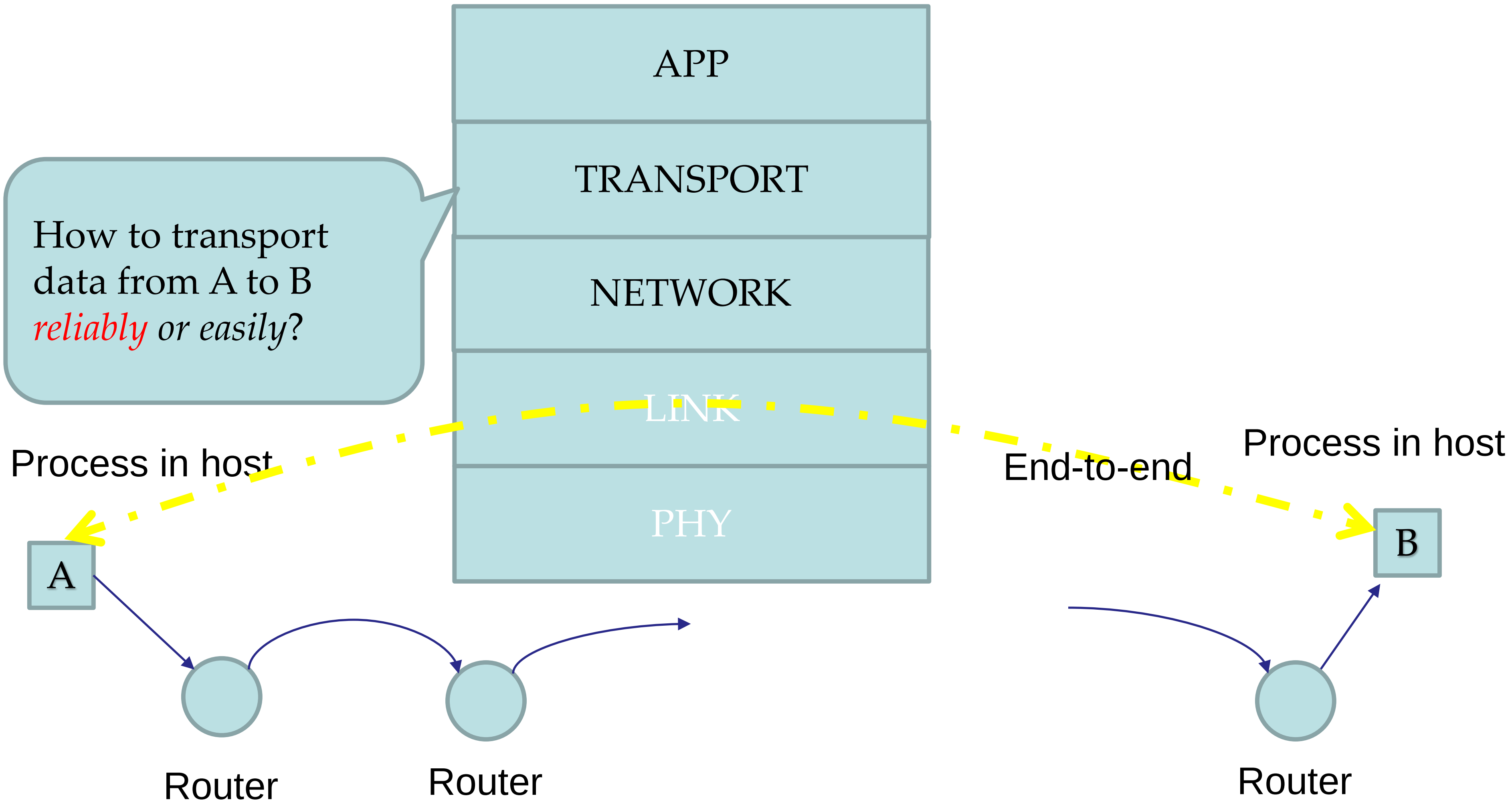
An overview for this term



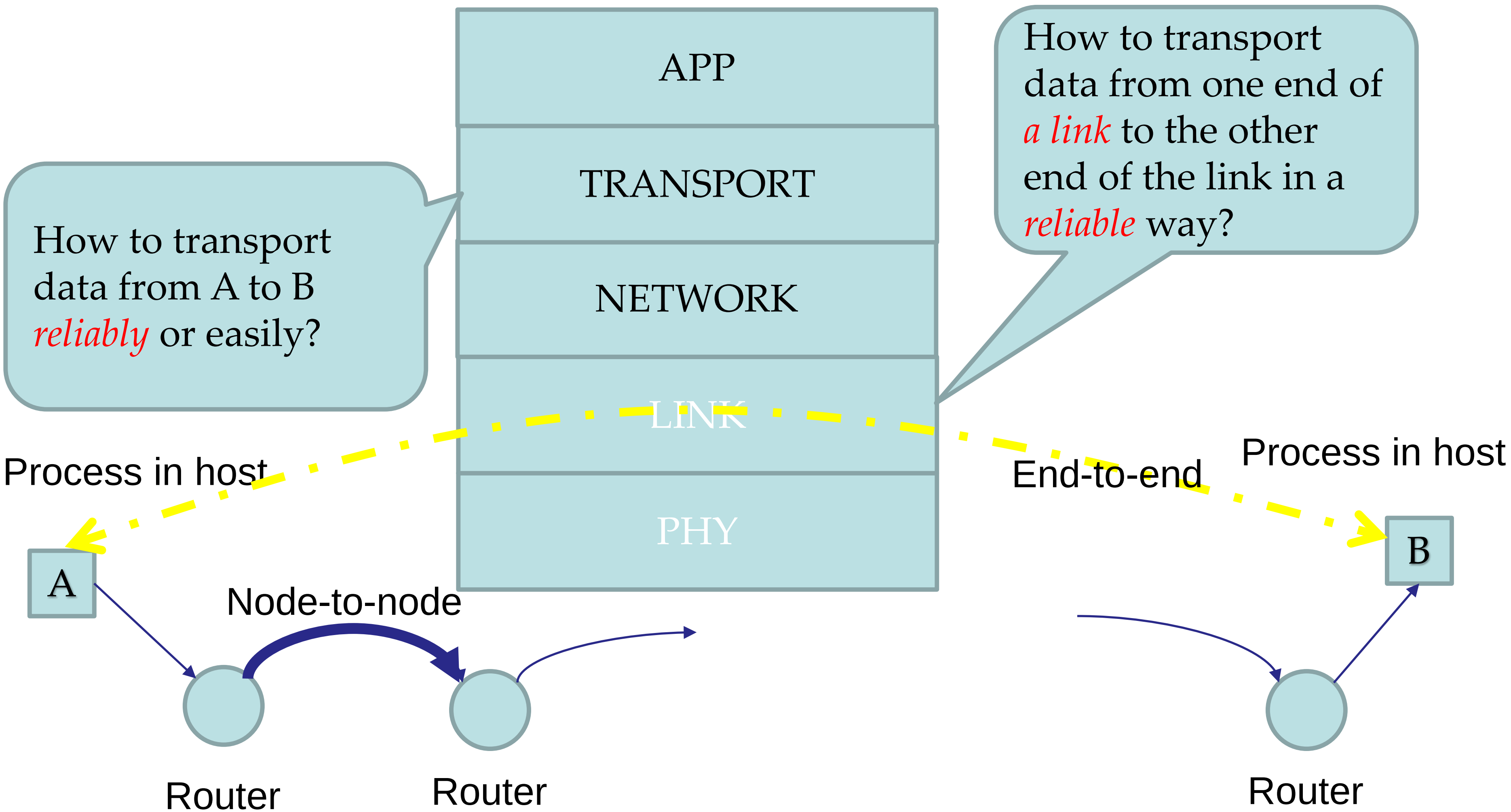
An overview for this term



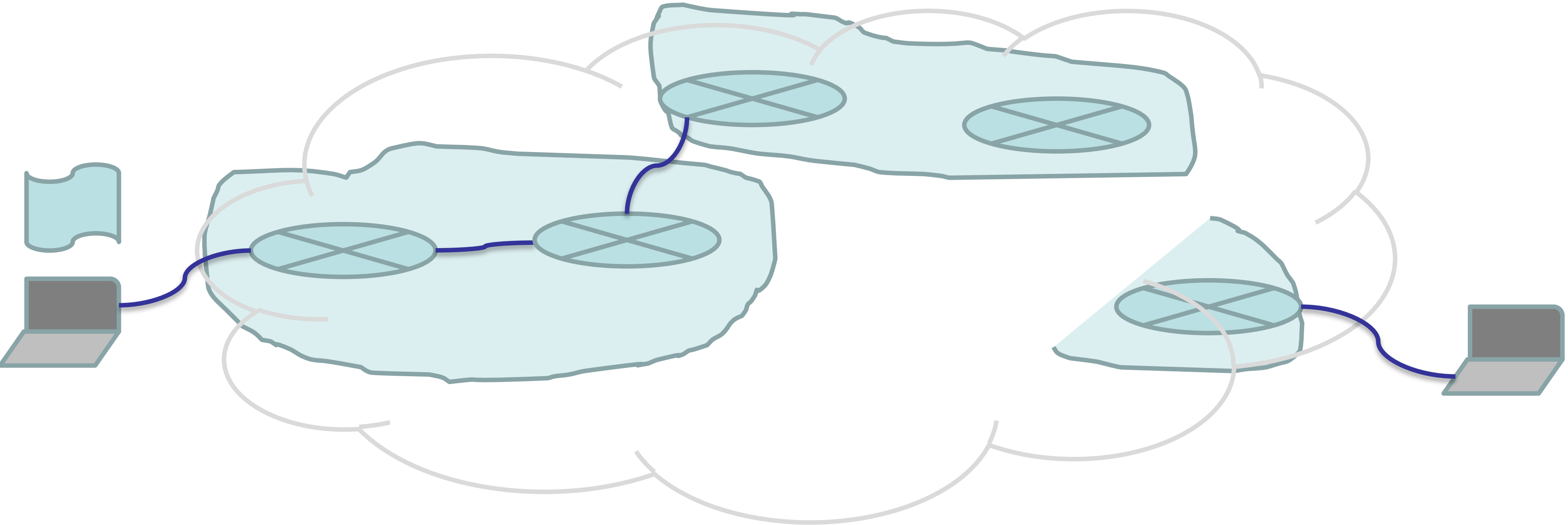
An overview for this term



An overview for this term



The journey of a packet

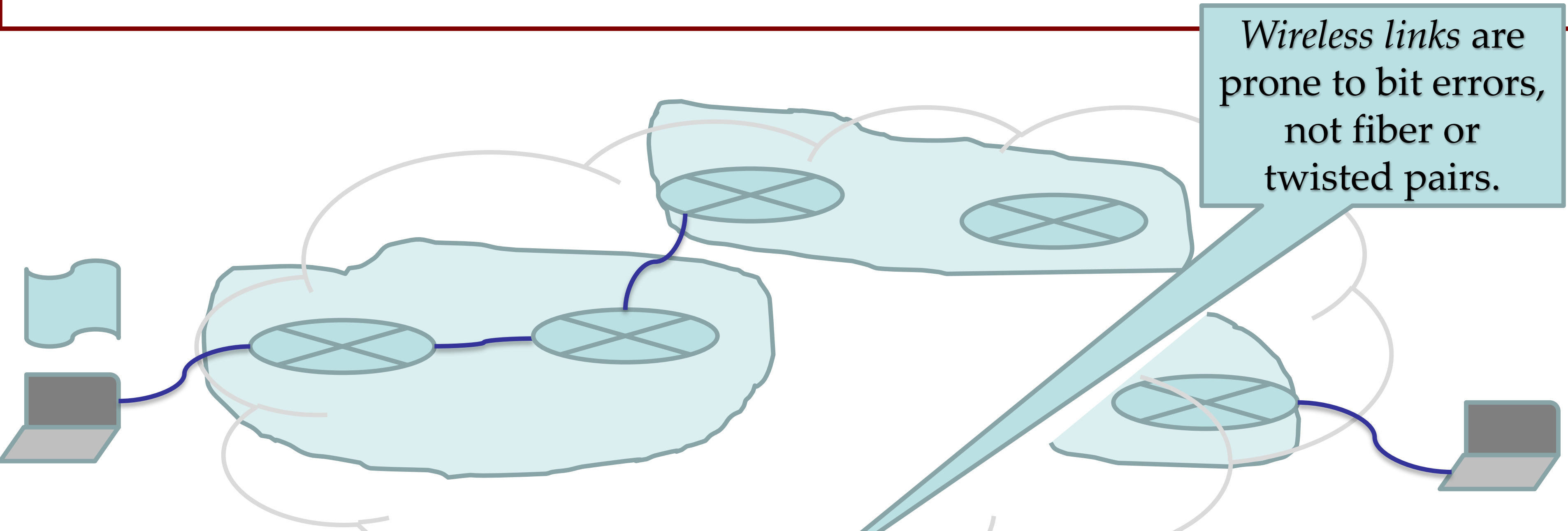


Causes for Transmission Errors

- Thermal noise
- Weak signal strength, interference, deliberate jamming, jitter. . .
- A crashing router / switch / station loses all packets currently in its memory
- Packets
  - are routed in the wrong direction
  - are dropped because of congestion
  - are dropped because of insufficient resources at receiver

Transmission errors present a design challenge for a network and a distributed application.

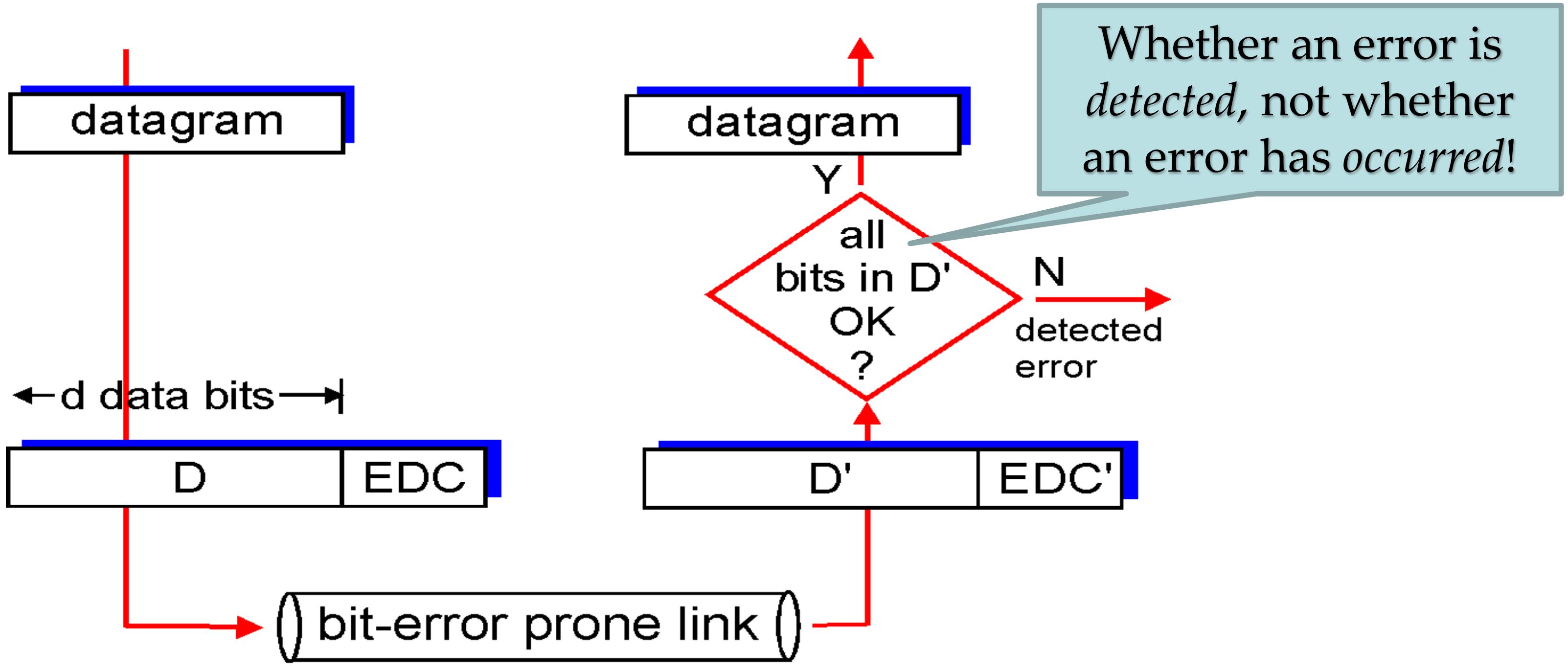
The journey of a packet



Problems	Causes	Solutions
Bit error	e.g., signal attenuation/noise	Error detection and correction
Buffer overflow	e.g., Speed-mismatch; Too much traffic;	Flow control and congestion control
Lost packet	e.g., buffer overflow at host/router	Acknowledgement and retransmission (ARQ)
Out of order	e.g. an early packet gets lost and retransmitted; a later one arrives first.	Acknowledgement and retransmission (ARQ)

Error Detection

EDC= Error Detection and Correction bits (redundancy)
D = Data protected by error checking, may include header fields

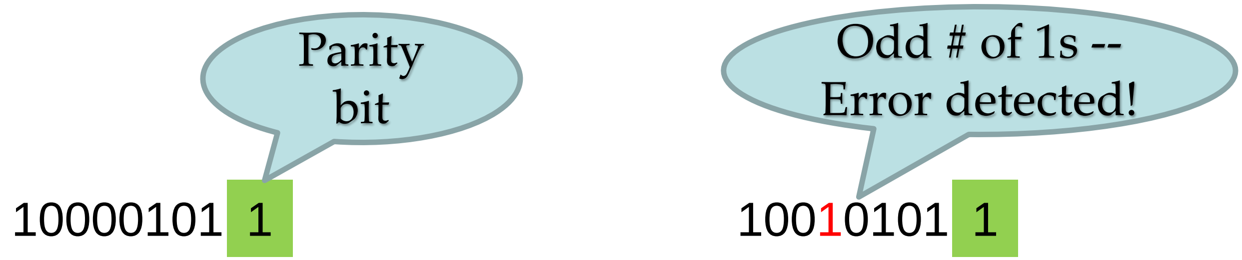


Outline

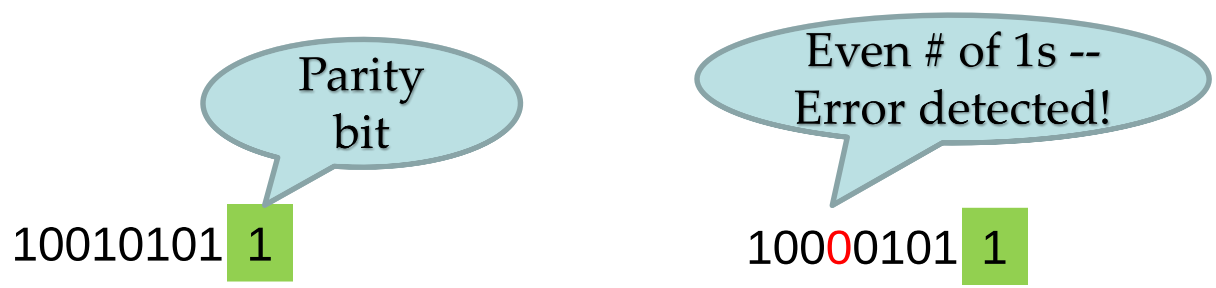
- Error Detection
  - Parity check
    - Parity bit / 2-D Parity check
  - Internet checksum
  - Cyclic Redundancy Check (CRC)
- Forward Error Correction
  - Block Code Principles
- Summary

Parity Check

- The simplest error **detecting** scheme is to append a parity bit to the end of a block of data
  - Even parity



  - Odd parity



Parity check

- If any even number of bits are flipped due to error, an undetected error occurs

Parity bit

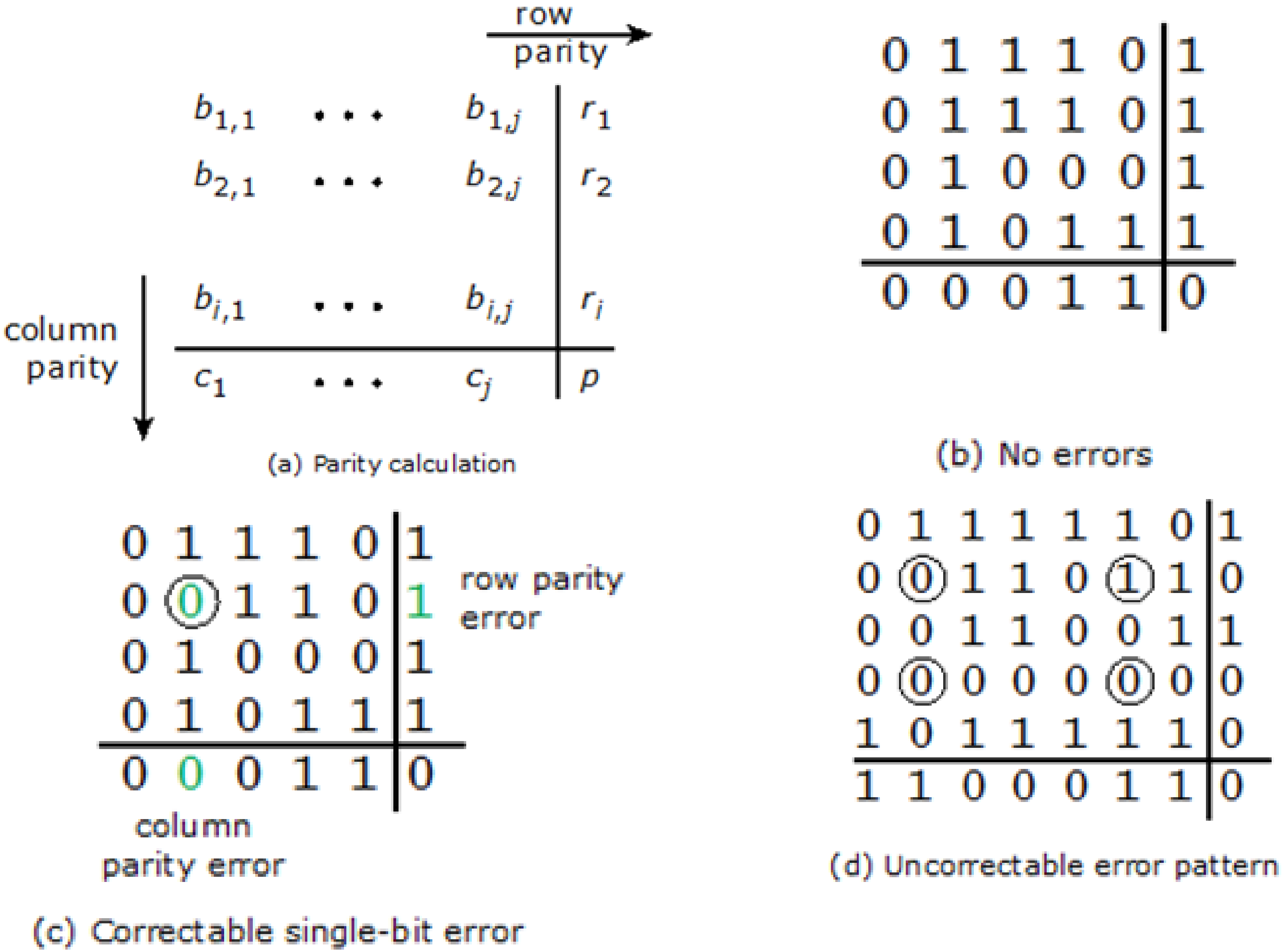
10000101 1

10010001 1

Even # of 1s – No error detected!

We need a more robust error-detection scheme!

2-D Parity Check



(a) Parity calculation

(b) No errors

(c) Correctable single-bit error

(d) Uncorrectable error pattern

The ability of the receiver to both detect and correct errors is known as forward error correction (FEC). (*No retransmission is needed!*)

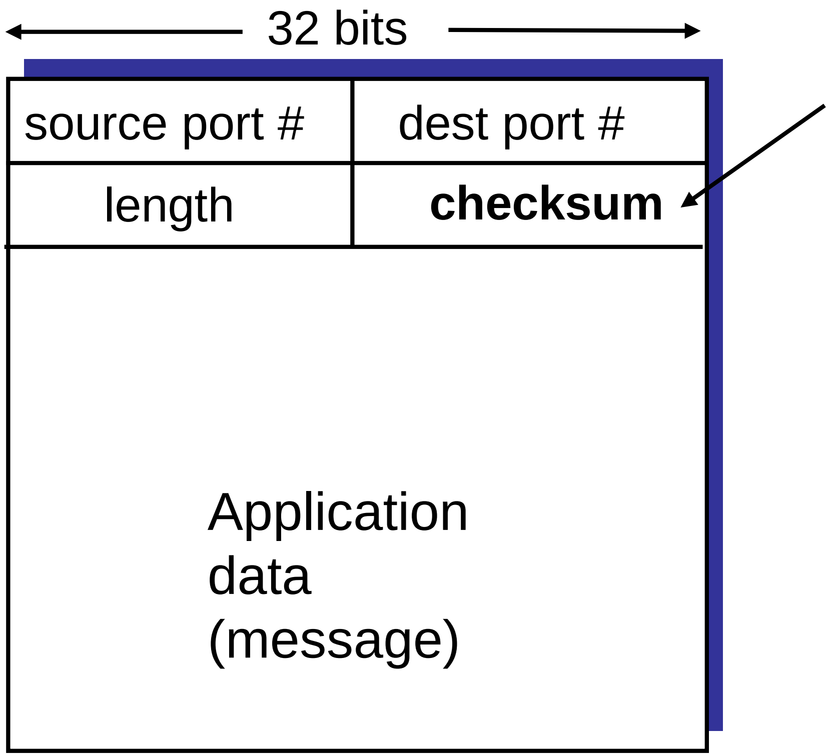
Outline

- Error Detection
  - Parity check
    - Parity bit / 2-D Parity check
  - **Internet checksum**
  - Cyclic Redundancy Check (CRC)
- Forward Error Correction
  - Block Code Principles
- Summary

The Internet Checksum

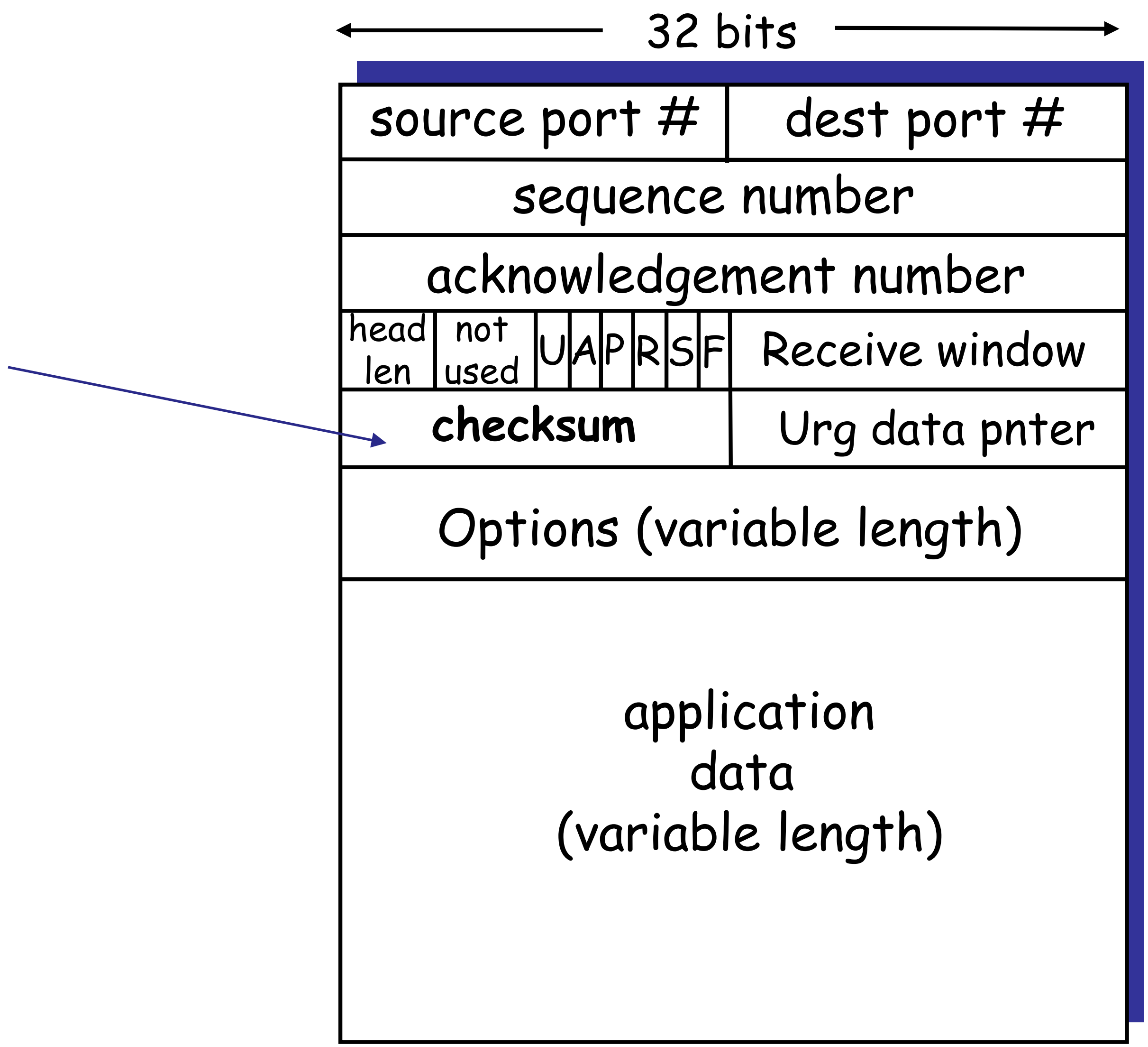
- The idea of the traditional checksum is simple.
- We show this using a simple example.
- Suppose the message is a list of five 4-bit numbers that we want to send to a destination.
  - In addition to sending these numbers, we send the sum of the numbers.
  - For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, 36), where 36 is the sum of the original numbers.
  - The receiver adds the five numbers and compares the result with the sum.
  - If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum.
  - Otherwise, there is an error somewhere and the message not accepted.

Internet Checksum

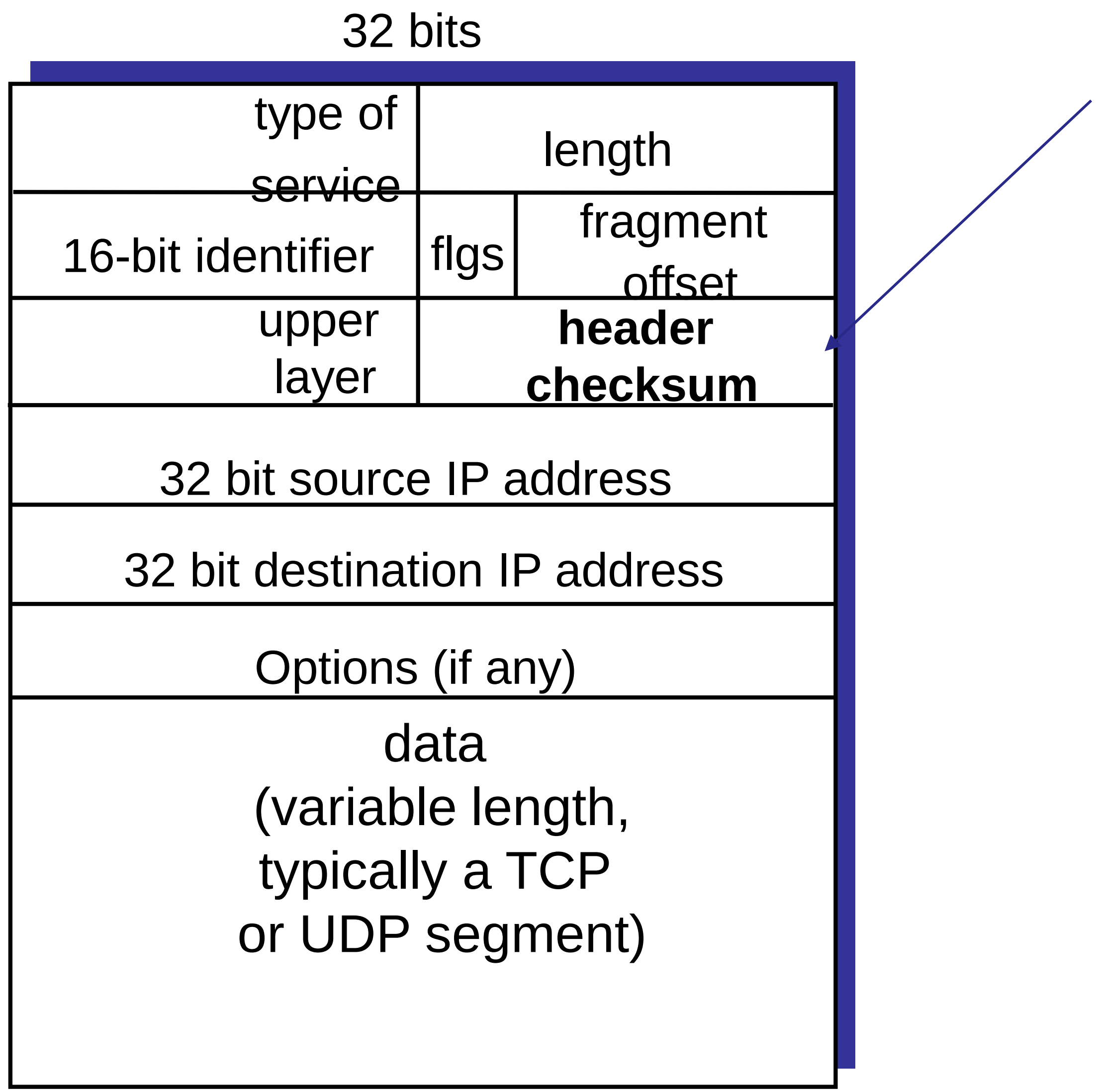


UDP segment format

TCP segment structure



IP datagram format



UDP checksum

Goal: detect “errors” (e.g., flipped bits) in transmitted segment

Sender:

- treat segment contents as sequence of 16-bit integers
- checksum: addition of segment contents (1’s complement of the sum)
- sender puts checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?* More later ….

Internet Checksum Example

- Note
  - When adding numbers, a carryout from the most significant bit needs to be added to the result
- Example: add two 16-bit integers

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
(partial) sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

At the receiver side, the receiver sums up all the segments *plus the checksum,* checking whether it is all 1 bits;

The Internet Checksum

https://en.wikipedia.org/wiki/Ones%27_complement

- The calculation makes use of:
  - Ones-complement **addition**
    1. The two numbers are treated as unsigned binary integers and added
    2. If there is a carry out of the leftmost bit, add 1 to the sum (*end-around carry*)

ex1.

```
  0011
 +1100
---------
  1111
```

ex2.

```
   1101
 + 1011
---------
  11000
 +    1
-------------
   1001
```

  - Ones-complement operation on a set of binary digits
    - Replace 0 digits with 1 digits and 1 digits with 0 digits

1010 1001 → 0101 0110

Internet Checksum Example

- Note
  - When adding numbers, a carryout from the most significant bit needs to be added to the result
- Example: add two 16-bit integers

```
                  1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
                  1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
              ------------------------------------
wraparound      1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
                                                 1
              ------------------------------------
(partial) sum     1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
     checksum     0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1
```

Using Hex

	1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
	1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
wraparound	1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
	1
(partial) sum	1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
checksum	0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1

hex:bin (dec)
A:1010 (10)
B:1011 (11)
C:1100 (12)
D:1101 (13)
E:1110 (14)
F:1111 (15)

```
                  E666
               +  D555
               -----------------
                 1BBBB
                     1
               -------------
(partial) sum     BBBC
     checksum     4 44 3
```

Ones complement in Hex:
F→0
E→1
D→2
C→3
B→4
A→5
9→6
8→7

The heading zero

- D341
- \+ 3396
- \-\-\-\-\-\-\-\-\-\-\-\-\-\-\-
- 106D7
- 1
- \-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-
- (partial) sum 06D8
- checksum F927
- The heading zero means 4 bits there and we need it for checksum calculation.

Ones complement in Hex:
F→0
E→1
D→2
C→3
B→4
A→5
9→6
8→7

hex:bin (dec)
A:1010 (10)
B:1011 (11)
C:1100 (12)
D:1101 (13)
E:1110 (14)
F:1111 (15)

The Internet Checksum (6)

- It is considerably *less* effective than the cyclic redundancy check (CRC), discussed next.
- The primary reason for its adoption in Internet protocols is ***efficiency***.
  - Most of these protocols are implemented in software and the Internet checksum, involving simple operations, causes very little overhead.
- It is assumed that
  - at the lower link level, a strong error-detection code such as CRC is used,
  - and so the Internet checksum is simply an additional end-to-end check for errors.

Updated by: 1141 INFORMATIONAL
Errata Exist

Network Working Group
Request for Comments: 1071

R. Braden
ISI
D. Borman
Cray Research
C. Partridge
BBN Laboratories
September 1988

Computing the Internet Checksum

Status of This Memo

This memo summarizes techniques and algorithms for efficiently computing the Internet checksum. It is not a standard, but a set of useful implementation techniques. Distribution of this memo is unlimited.

1. Introduction

This memo discusses methods for efficiently computing the Internet checksum that is used by the standard Internet protocols IP, UDP, and TCP.

An efficient checksum implementation is critical to good performance. As advances in implementation techniques streamline the rest of the protocol processing, the checksum computation becomes one of the limiting factors on TCP performance, for example. It is usually appropriate to carefully hand-craft the checksum routine, exploiting every machine-dependent trick possible; a fraction of a microsecond per TCP data byte can add up to a significant CPU time savings overall.

In outline, the Internet checksum algorithm is very simple:

(1) Adjacent octets to be checksummed are paired to form 16-bit integers, and the 1's complement sum of these 16-bit integers is formed.

Why do we need checksums at IP, UDP and TCP?

- IP just does header checksum, a double check; (no header checksum in IPv6).
- The *celebrated end-to-end principle* in system design:
  - Some link-layer does not do error-detection;
  - Bit errors could happen anywhere (router’s memory);
  - *Certain functionality must be implemented on an end-to-end basis*;
  - A final end-to-end check is always needed!

END-TO-END ARGUMENTS IN SYSTEM DESIGN

J.H. Saltzer, D.P. Reed and D.D. Clark*

M.I.T. Laboratory for Computer Science

This paper presents a design principle that helps guide placement of functions among the modules of a distributed computer system. The principle, called the end-to-end argument, suggests that functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level. Examples discussed in the paper include bit error recovery, security using encryption, duplicate message suppression, recovery from system crashes, and delivery acknowledgement. Low level mechanisms to support these functions are justified only as performance enhancements.

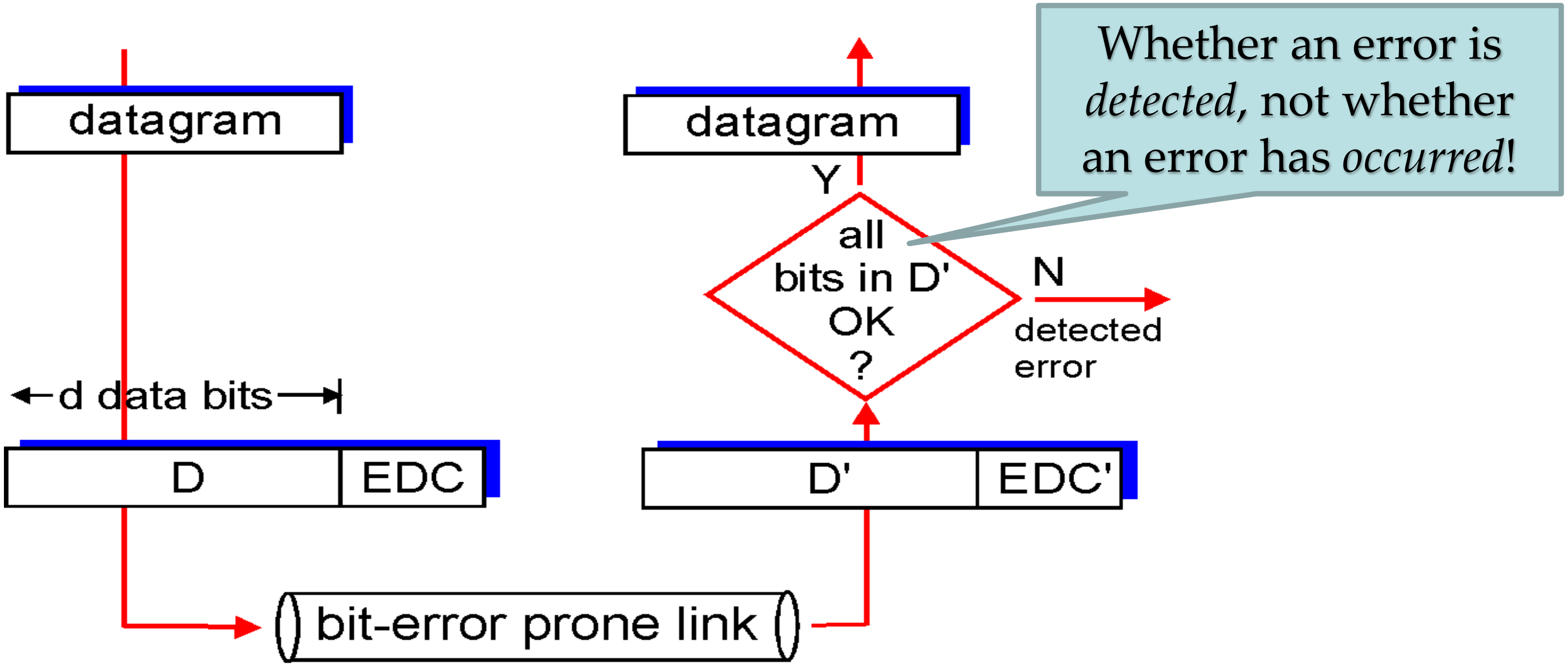
J. Saltzer, D. Reed, D. Clark, "End-to-end argument in system design," *TOCS*, 2(4), Nov. 1984.

Outline

- Error Detection
  - Parity check
    - Parity bit / 2-D Parity check
  - Internet checksum
  - **Cyclic Redundancy Check (CRC)**
- Forward Error Correction
  - Block Code Principles
- Summary

Error Detection

EDC= Error Detection and Correction bits (redundancy)
D = Data protected by error checking, may include header fields

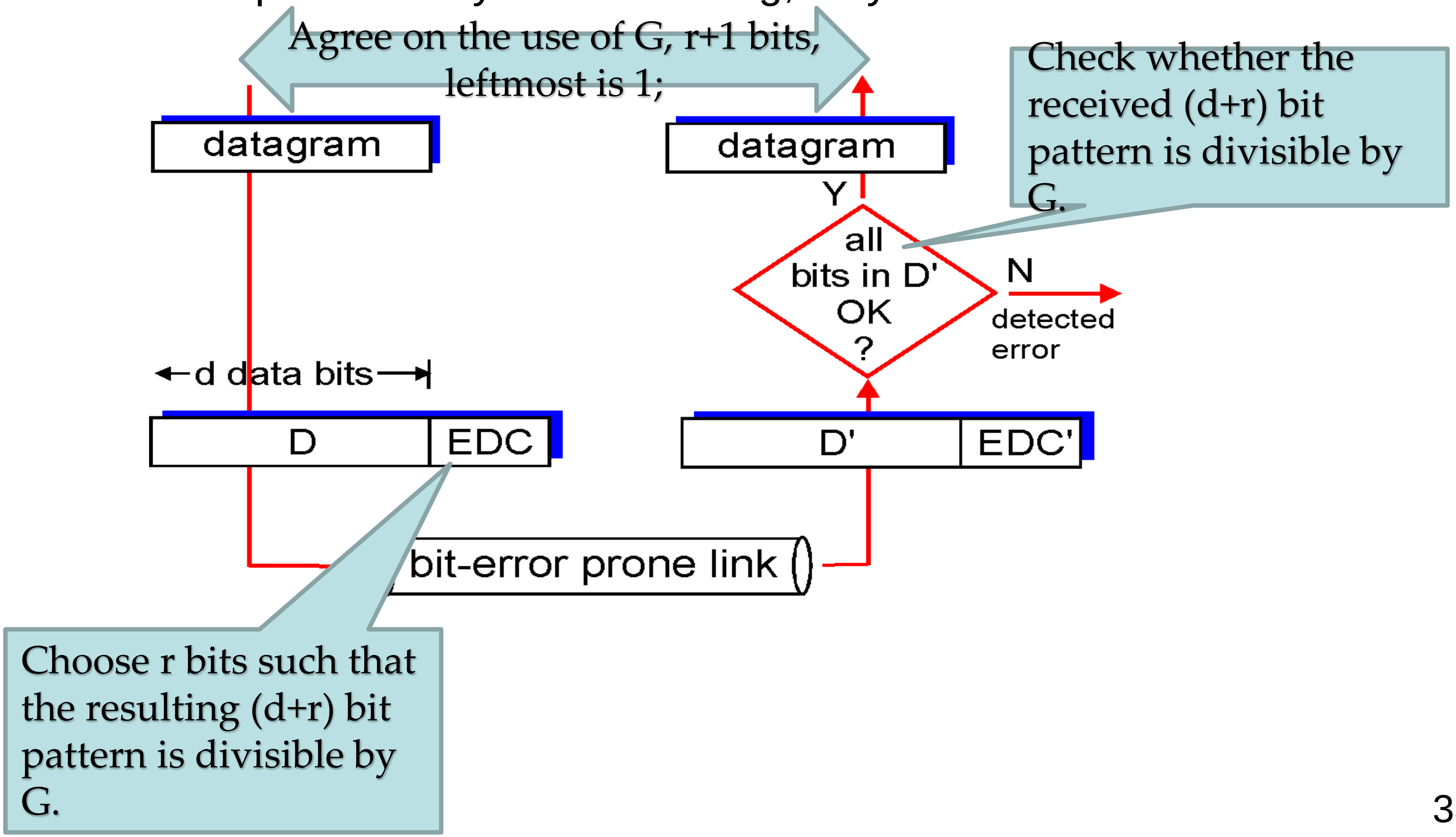


Cyclic Redundancy Check (CRC)

- *d*-bit data to be sent;
- *Sender and receiver first agree on an r+1 pattern (generator); leftmost bit is 1;*
- Sender chooses *r* additional bits appending the *d*-bit data *such that the d+r bits data is divisible by the generator using modulo-2 arithmetic;*
- Receiver divides the incoming *d*+*r* bits data by the generator
  - If there is no remainder, assume there is no error.

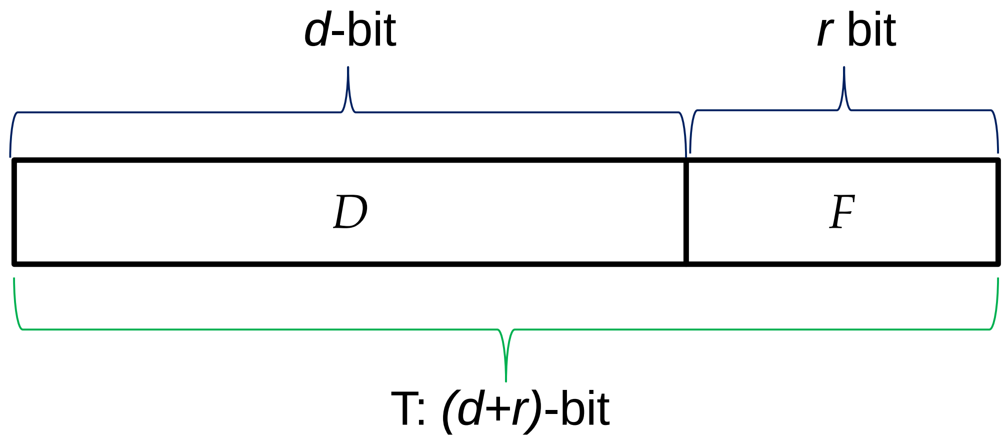
Error Detection

EDC= Error Detection and Correction bits (redundancy)
D = Data protected by error checking, may include header fields



CRC - notations

- Now define
  - *D* = *d*-bit block of data
  - *F* = *r* additional bits (Frame Check Sequence)
  - *T* = *(d+r)*-bit frame to be transmitted
  - *G* = pattern of *r+1* bits; this is the predetermined divisor



- We would like T is divisible by G (using *modulo-2 arithmetic*).

CRC – modulo-2 arithmetic

- Modulo-2 arithmetic
  - Addition and subtraction: Uses binary addition with no *carries, just XOR*

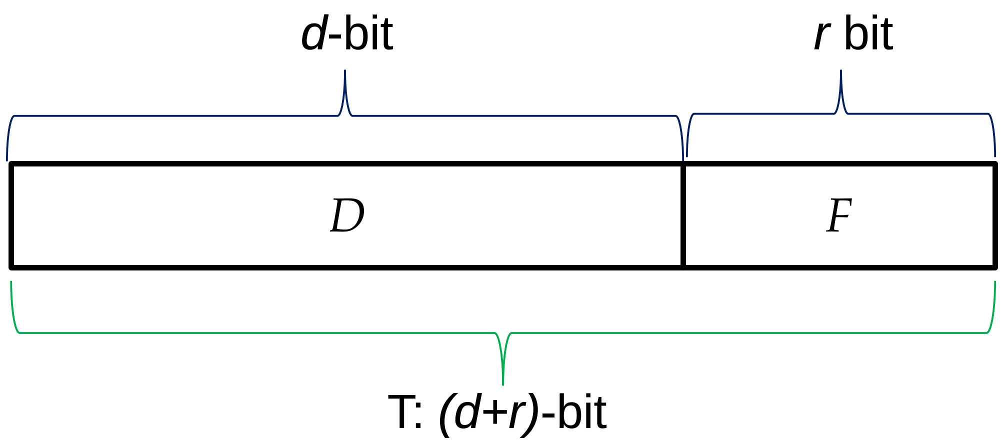
```
 1111
+1010
--------
 0101
```

```
 1111
-0101
--------
 1010
```

  - Multiplication and division: the same as in base-2 arithmetic, but add or sub is done without carries or borrows (XOR, exclusive-or).

How to generate the *r* bits

- The (*d*+*r*) bits data is $D * 2^r$ XOR F;
- Then
  - $D * 2^r$ XOR F = nG;
  - $D * 2^r$ = nG XOR F;
  - F = remainder of $(D * 2^r)/G$ *(modulo-2 arithmetic)*;



CRC – an example $r = 5$

```
                          1 1 0 1 0 1 0 1 1 0
                 ________________________________
G ──> 1 1 0 1 0 1 / 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0  <── D*2^r
                    1 1 0 1 0 1
                    ------------------------------
                      1 1 1 0 1 1
                      1 1 0 1 0 1
                      ------------------------------
                          1 1 1 0 1 0
                          1 1 0 1 0 1
                          ------------------------------
                              1 1 1 1 1 0
                              1 1 0 1 0 1
                              ------------------------------
                                  1 0 1 1 0 0
                                  1 1 0 1 0 1
                                  ------------------------------
                                    1 1 0 0 1 0
                                    1 1 0 1 0 1
                                    ------------------------------
                                        1 1 1 0  <── R
```

A divisor is said "to go into" a dividend if the dividend has equal number of bits as the divisor! (heading zeros not counted)

- The remainder is added to 2^5D to give T= 1010001101<u>**01110**</u> ;
- If there are no errors, the receiver receives *T* intact (i.e., no damage). The received frame is divided by *G*;

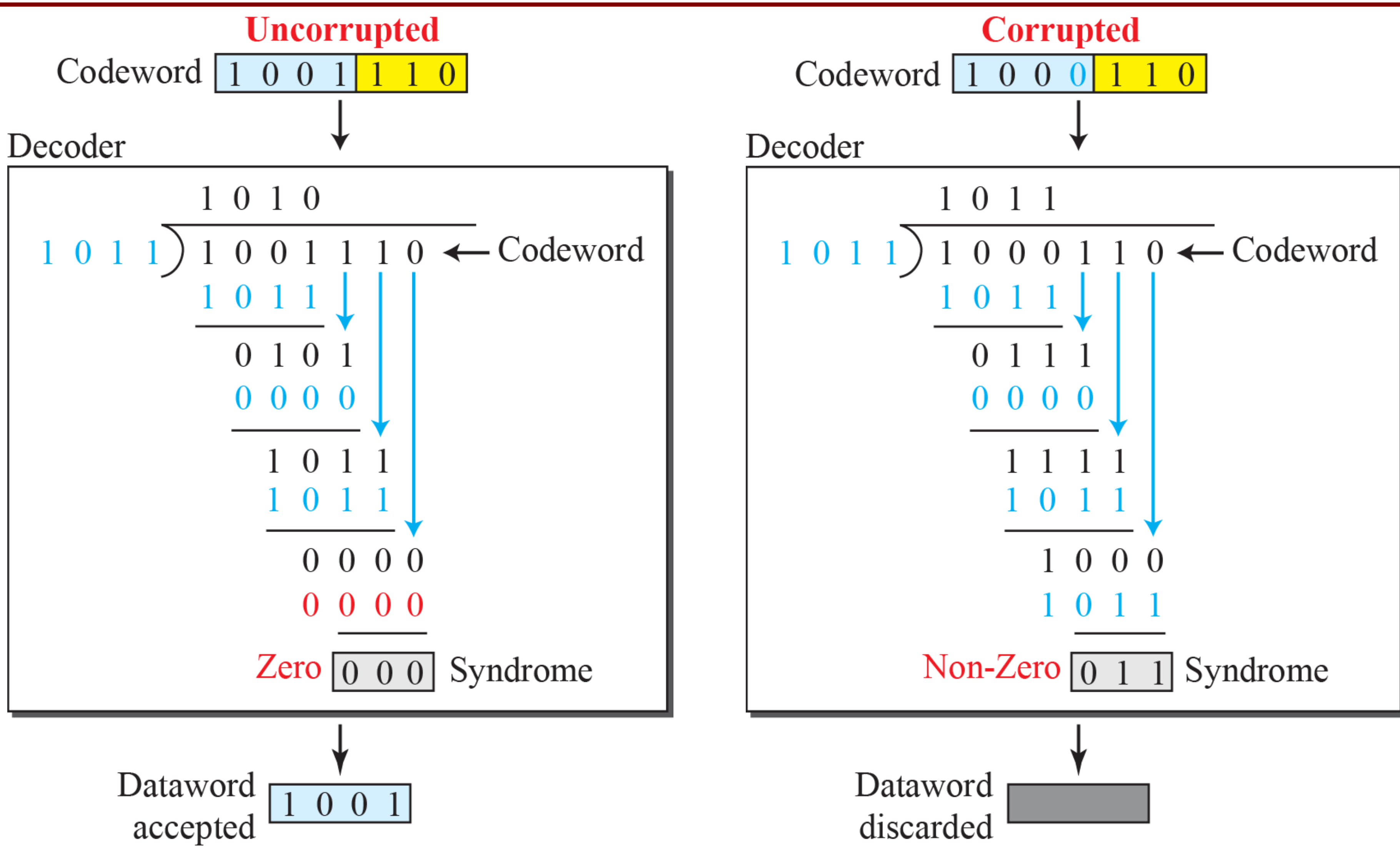
CRC (6)

```
                          1 1 0 1 0 1 0 1 1 0
                   _______________________________
G → 1 1 0 1 0 1  /  1 0 1 0 0 0 1 1 0 1 0 1 1 1 0  ← T
                    1 1 0 1 0 1
                    -----------
                      1 1 1 0 1 1
                      1 1 0 1 0 1
                      -----------
                          1 1 1 0 1 0
                          1 1 0 1 0 1
                          -----------
                              1 1 1 1 1 0
                              1 1 0 1 0 1
                              -----------
                                  1 0 1 1 1 1
                                  1 1 0 1 0 1
                                  -----------
                                    1 1 0 1 0 1
                                    1 1 0 1 0 1
                                    -----------
                                              0  ← R
```

If we do a circular shift, e.g., left shift 1 place, 0100011010111011; we find that it is divisible by G (110101)!

- Because there is no remainder, it is assumed that there have been no errors

Corrupted bits

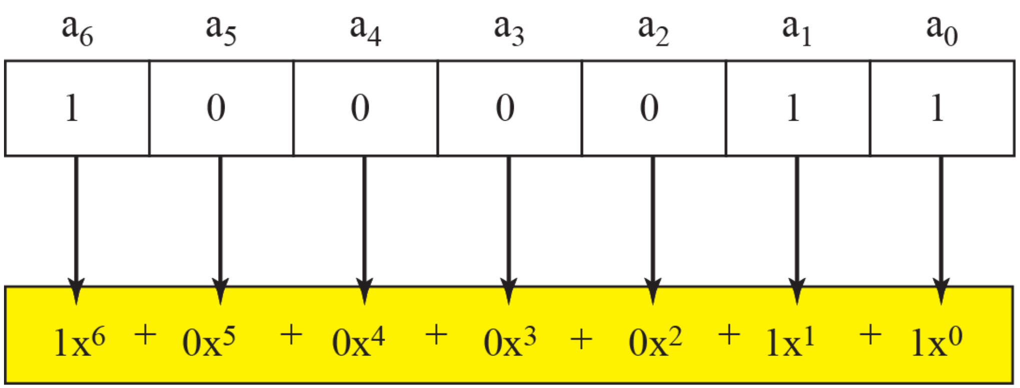


For CRC codes with an *r+1* bit pattern G, they can detect consecutive bit errors of *r* bits or fewer!

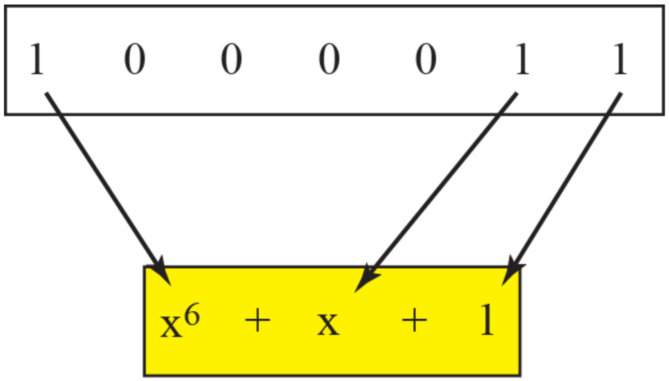
[BAF5]

CRC, also known as polynomial codes

- The power of each term shows the position of the bit;
  - the coefficient shows the value of the bit.

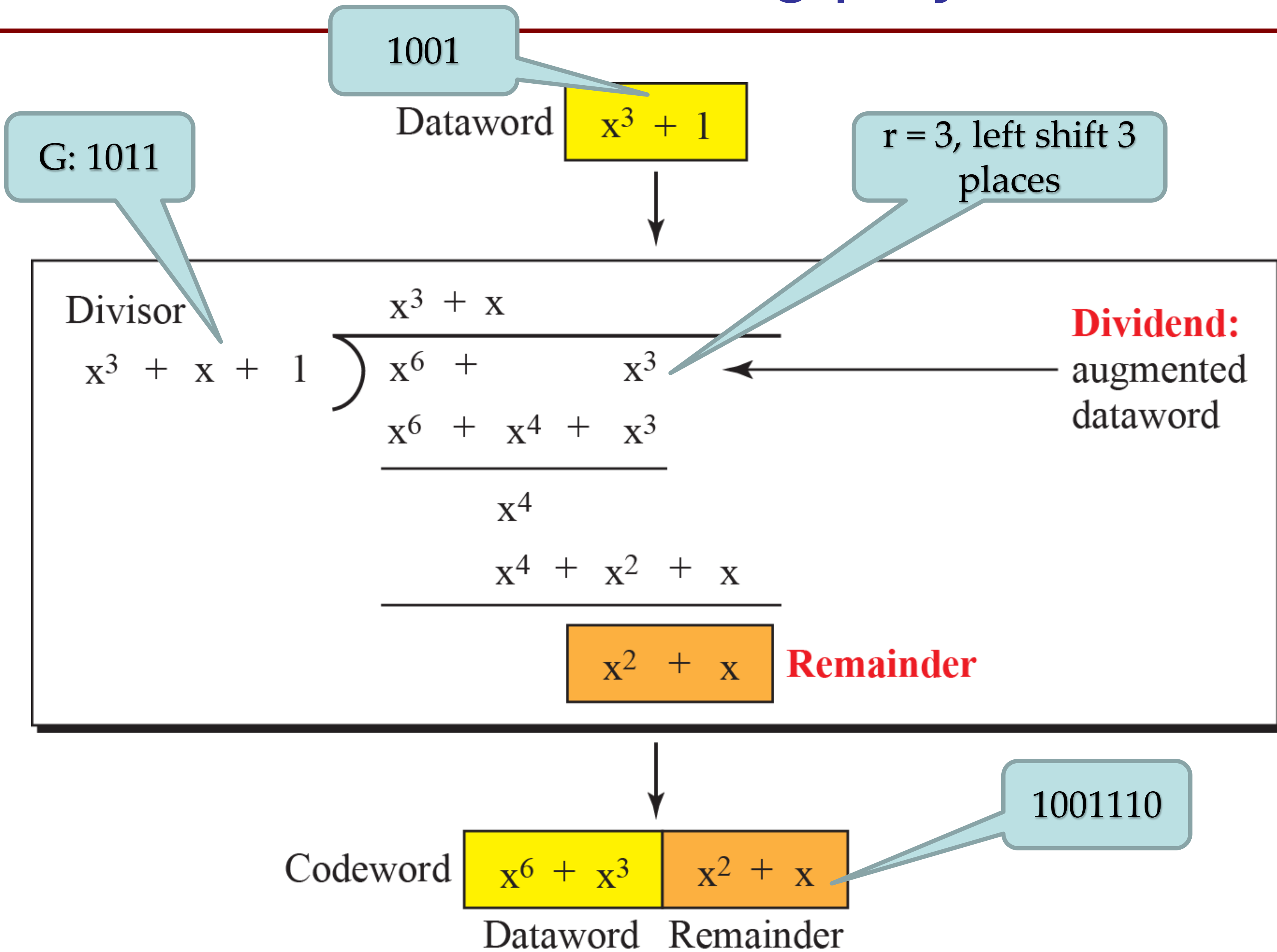


a. Binary pattern and polynomial



b. Short form

CRC division using polynomials

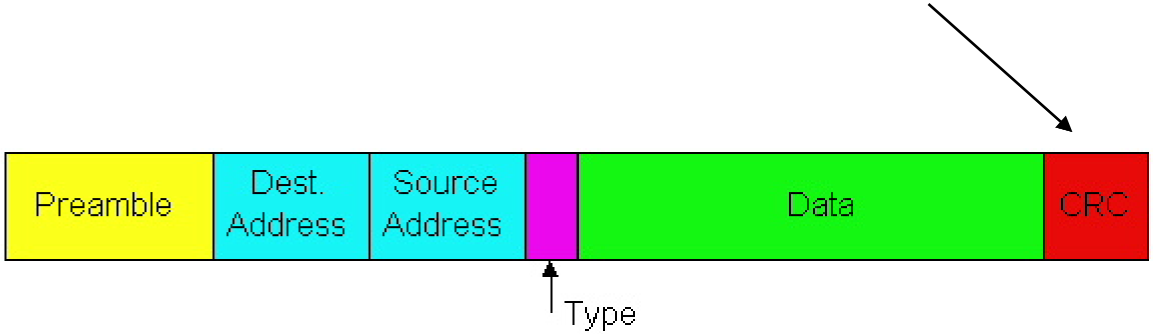


CRC – Some Standard Polynomials

- Four versions of *G* are widely used.

CRC-12	$X^{12} + X^{11} + X^3 + X^2 + X + 1$
CRC-16	$X^{16} + X^{15} + X^2 + 1$
CRC-CCITT	$X^{16} + X^{12} + X^5 + 1$
CRC-32	$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$

  - CRC-32: is specified as an option in some point-to-point synchronous transmission standards and is used in **IEEE 802 LAN** standards



Outline

- Introduction
  - Learning objectives
  - Background
  - Types of errors
- Error Detection
  - Parity check
    - Parity bit / 2-D Parity check
  - Internet checksum
  - Cyclic Redundancy Check (CRC)
- **Forward Error Correction**
  - **Block Code Principles**
- Summary

Forward Error Correction (FEC)

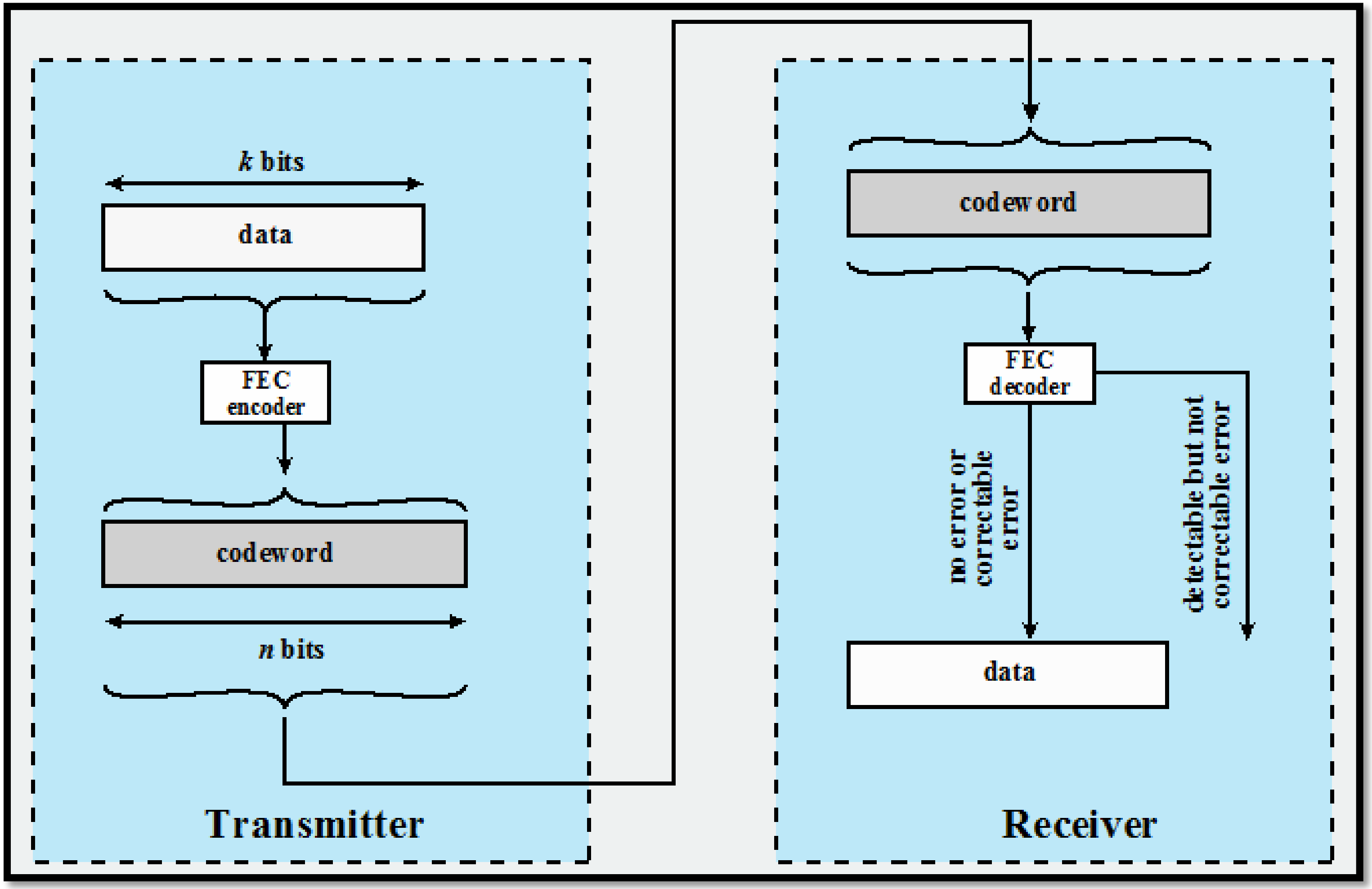
- Typically, retransmission is needed when there are bit errors;
- Not good enough for wireless applications:
  - High bit error rate (BER) on a wireless link (retransmissions)
  - Propagation delay
- **Need to correct errors on basis of bits received;**

Univursity of Canterbery

Christchurdh

FEC Process

- We divide our message into blocks (*block coding*), each of k bits, called *datawords*. FEC encoder encodes datawords; the resulting *n*-bit blocks are called *codewords*.



An example

- Let us assume that $k = 2$ and $n = 3$. The following table shows the list of datawords and codewords.

datawords	codewords
00	000
01	011
10	101
11	110

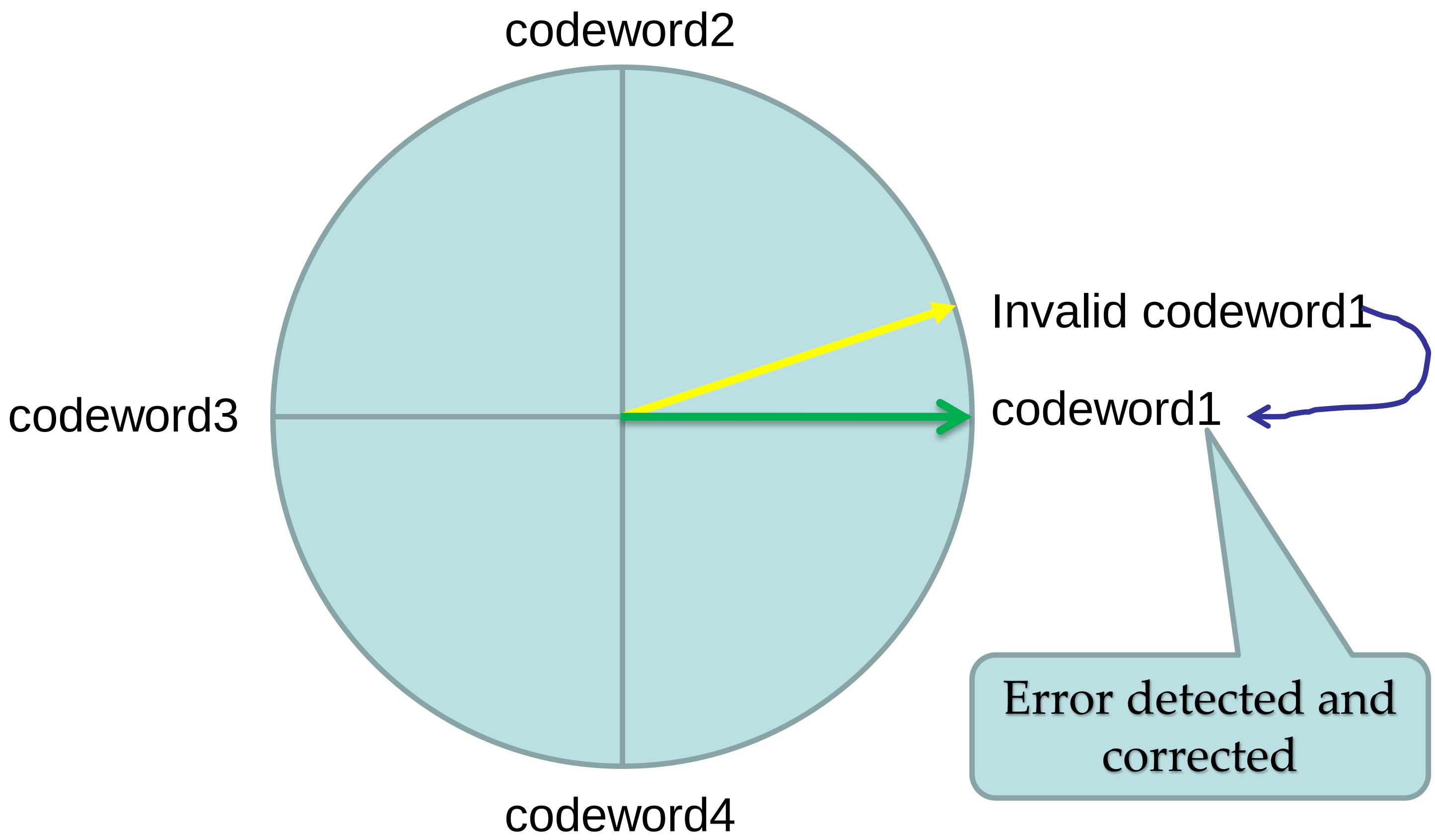
Case 1: dataword = 00; 000 is received; → 00
Case 2: dataword = 11; 111 is received; → bit error detected; discard;
Case 3: dataword = 10; 110 is received; → 11, no bit error detected;

Where is the ability of error correction?

Block Code Principles

- **Hamming distance**
  - $d(v_1, v_2)$ between two n –bit binary sequences v_1 and v_2 is *the number of bits in which v_1 and v_2 disagree*.
  - E.g. v_1=011011, v_2=110001, $d(v_1,v_2)$ = 3

Error correction by checking min-distance



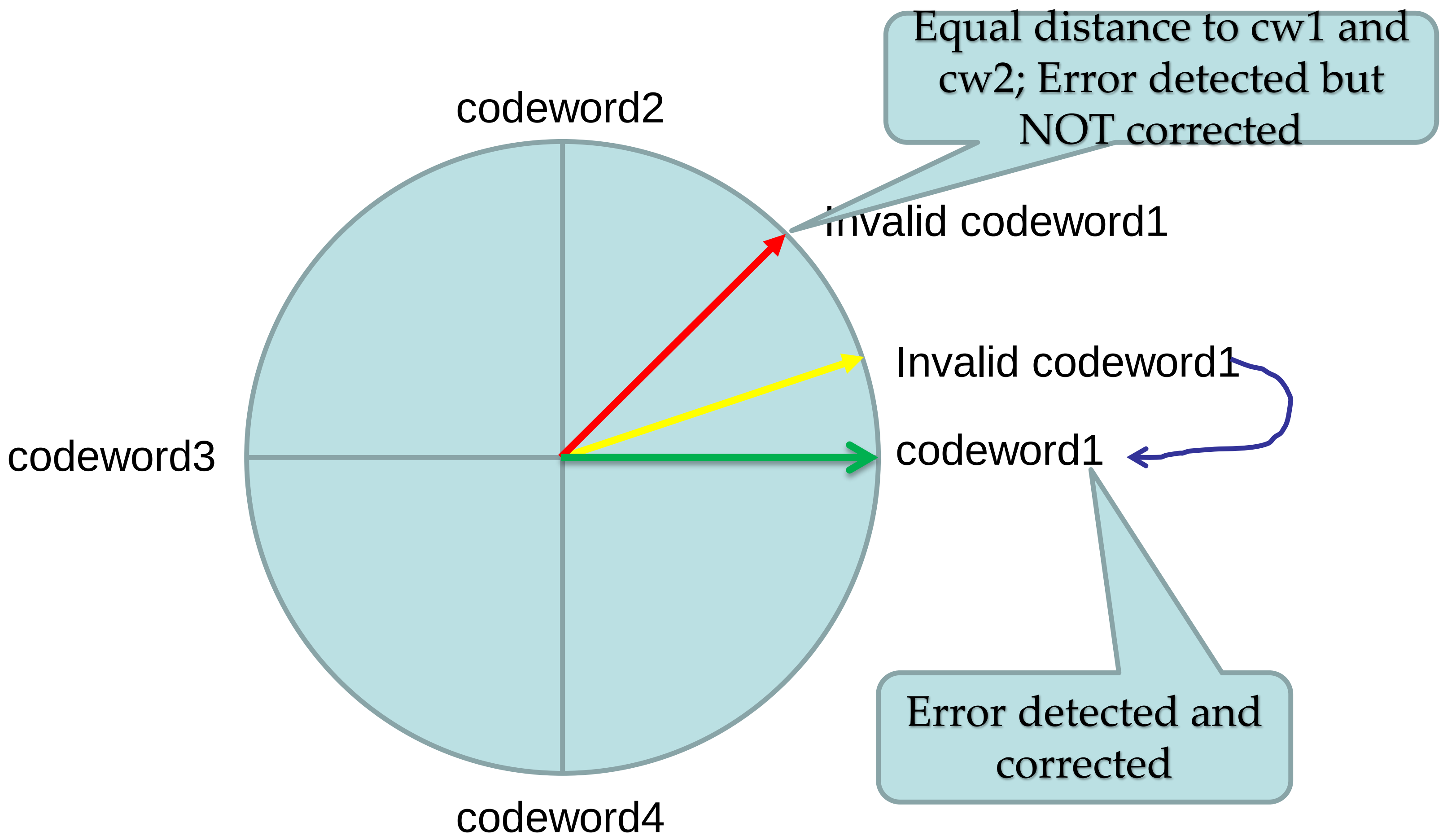
Block Code Principles

- The sender sends “00”; a codeword block is received with the bit pattern 00100;

		Codeword	Dataword
00100	Single bit change →	00000	00
00100	Two-bit change →	00111	01
00100	Four-bit change →	11001	10
00100	Three-bit change →	11110	11

This is an error correction!

Limitation



Equal distance to cw1 and cw2; Error detected but NOT corrected
codeword2
Invalid codeword1
Invalid codeword1
codeword3
codeword1
Error detected and corrected
codeword4

An example

- There are $2^5 = 32$ possible codwords of which 4 are valid, leaving 28 invalid codewords.

Data Block (dataword)	**Codeword**
00	00000
01	00111
10	11001
11	11110

An example

Invalid codeword	Minimum Distance	Valid Codeword	Invalid Codeword	Minimum Distance	Valid Codeword
00001	1	00000	10000	1	00000
00010	1	00000	10001	1	11001
00011	1	00111	10010	2	**00000 or 11110**
00100	1	00000	10011	2	**00111 or 11001**
00101	1	00111	10100	2	**00111 or 11001**
00110	1	00111	10101	2	**00111 or 11001**
01000	1	00000	10110	1	11110
01001	1	11001	10111	1	00111
01010	2	**00000 or 11110**	11000	1	11001
01011	2	**00111 or 11001**	11010	1	11110
01100	2	**00000 or 11110**	11011	1	11001
01101	2	**00111 or 11001**	11100	1	11110
01110	1	11110	11101	1	11001
01111	1	00111	11111	1	11110

An example

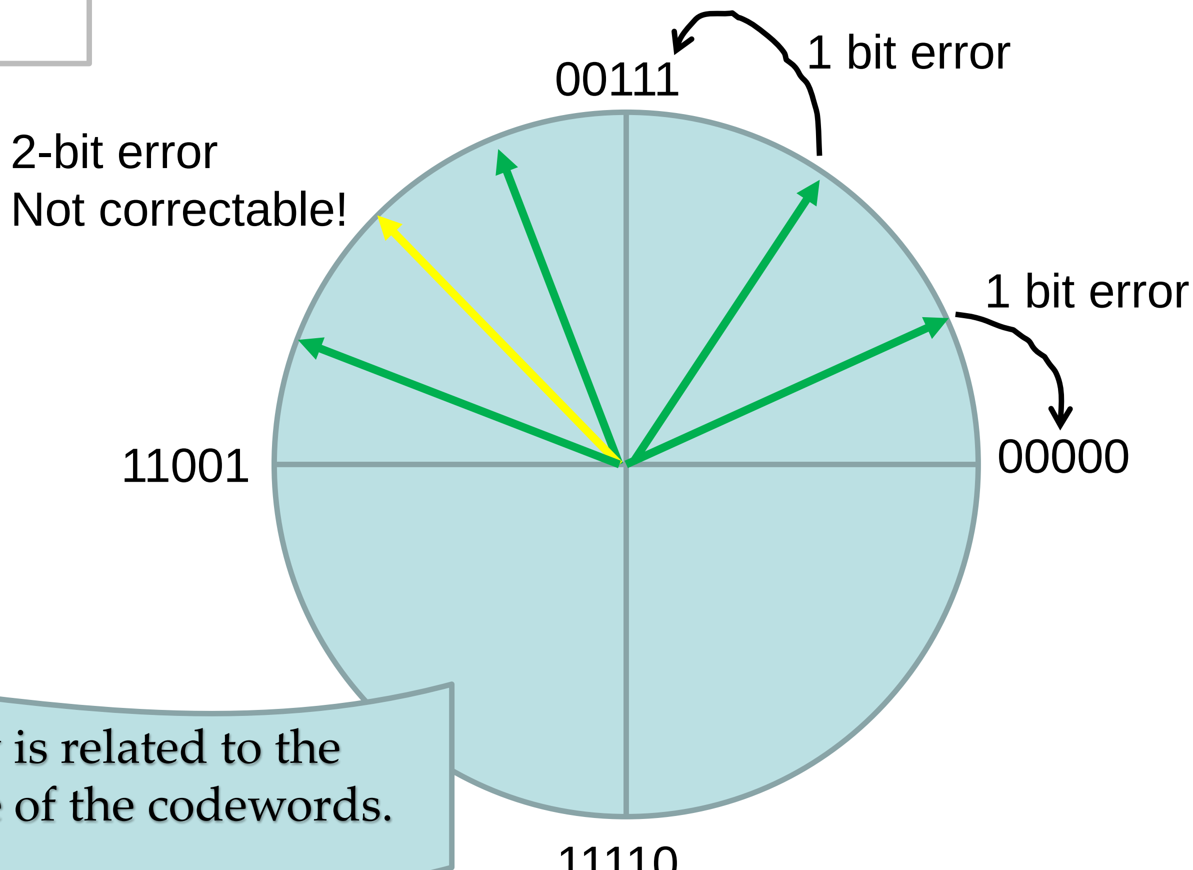
Invalid codeword	Minimum Distance	Valid Codeword	Invalid Codeword	Minimum Distance	Valid Codeword
00001	1	00000	10000	1	00000
00010	1	00000	10001	1	11001
00011	1	00111	10010	2	**00000 or 11110**
00100	1	00000	10011	2	**00111 or 11001**
00101	1	00111	10100	2	**00111 or 11001**
00110	1	00111	10101	2	**00111 or 11001**
01000	1	00000	10110	1	11110
01001	1	11001	10111	1	00111
01010	2	**00000 or 11110**	1		
01011	2	**00111 or 11001**			
01100	2	**00000 or 11110**	1		
01101	2	**00111 or 11001**	1		
01110	1	11110	1		
01111	1	00111	11111	1	11110

Whenever the receiver receives such codewords, it knows 2 bit errors could have happened but cannot correct them!

Error correction ability

Data Block (dataword)	Codeword
00	00000
01	00111
10	11001
11	11110

d(00000,00111) = 3;
d(00000,11001) = 3;
d(00000,11110) = 4;
d(00111, 11001) = 4;
d(00111, 11110) = 3;
d(11001,11110) = 3;



The error-correction ability is related to the minimum Hamming distance of the codewords.

Summary

- Error Detection
  - Parity check
  - Internet checksum
  - Cyclic Redundancy Check (CRC)
- Forward Error Correction
  - Block Code Principles

References

- [KR3] James F. Kurose, Keith W. Ross, *Computer networking: a top-down approach featuring the Internet*, 3rd edition.
- [PD5] Larry L. Peterson, Bruce S. Davie, *Computer networks: a systems approach*, 5th edition
- [TW5] Andrew S. Tanenbaum, David J. Wetherall, *Computer network*, 5th edition
- [LHBi]Y-D. Lin, R-H. Hwang, F. Baker, *Computer network: an open source approach*, International edition
- [BAF5] B. A. Forouzan, *Data communications and networking*, 5th edition.

Acknowledgements

- All slides are developed based on slides from the following two sources:
  - Dr DongSeong Kim’s slides for COSC264, University of Canterbury;
  - Prof Aleksandar Kuzmanovic’s lecture notes for CS340,Northwestern University, https://users.cs.northwestern.edu/~akuzma/classes/CS340-w05/lecture_notes.htm