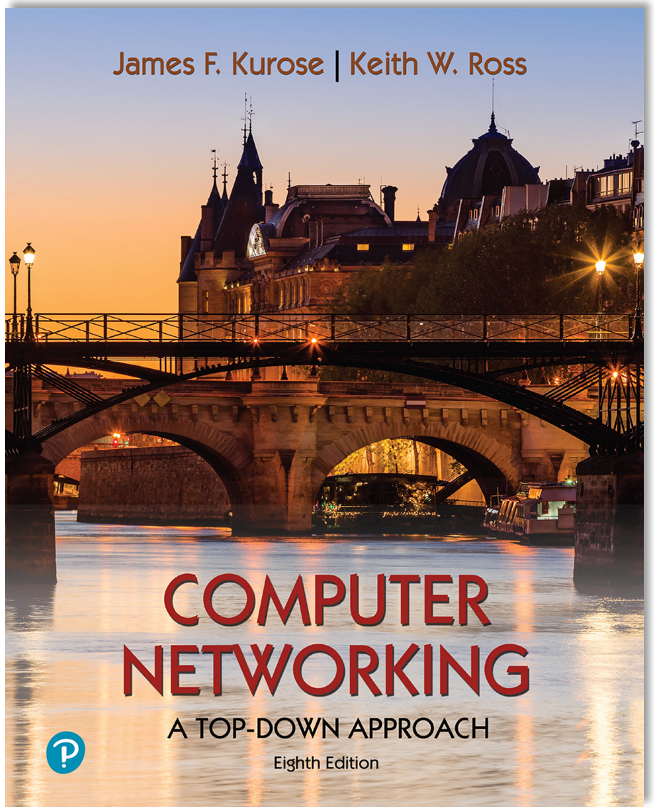
Chapter 6
The Link Layer and LANs



Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Acknowledgement: Based on the textbook's website:
https://gaia.cs.umass.edu/kurose_ross/index.php

Link layer and LANs: our goals

- understand principles behind link layer services:
  - error detection, correction
  - sharing a broadcast channel: multiple access
  - link layer addressing
  - local area networks: Ethernet, VLANs
- datacenter networks

- instantiation, implementation of various link layer technologies

Link layer, LANs: roadmap

- introduction
- error detection, correction
- multiple access protocols
- LANs
  - addressing, ARP
  - Ethernet
  - switches
  - VLANs
- link virtualization: MPLS
- data center networking

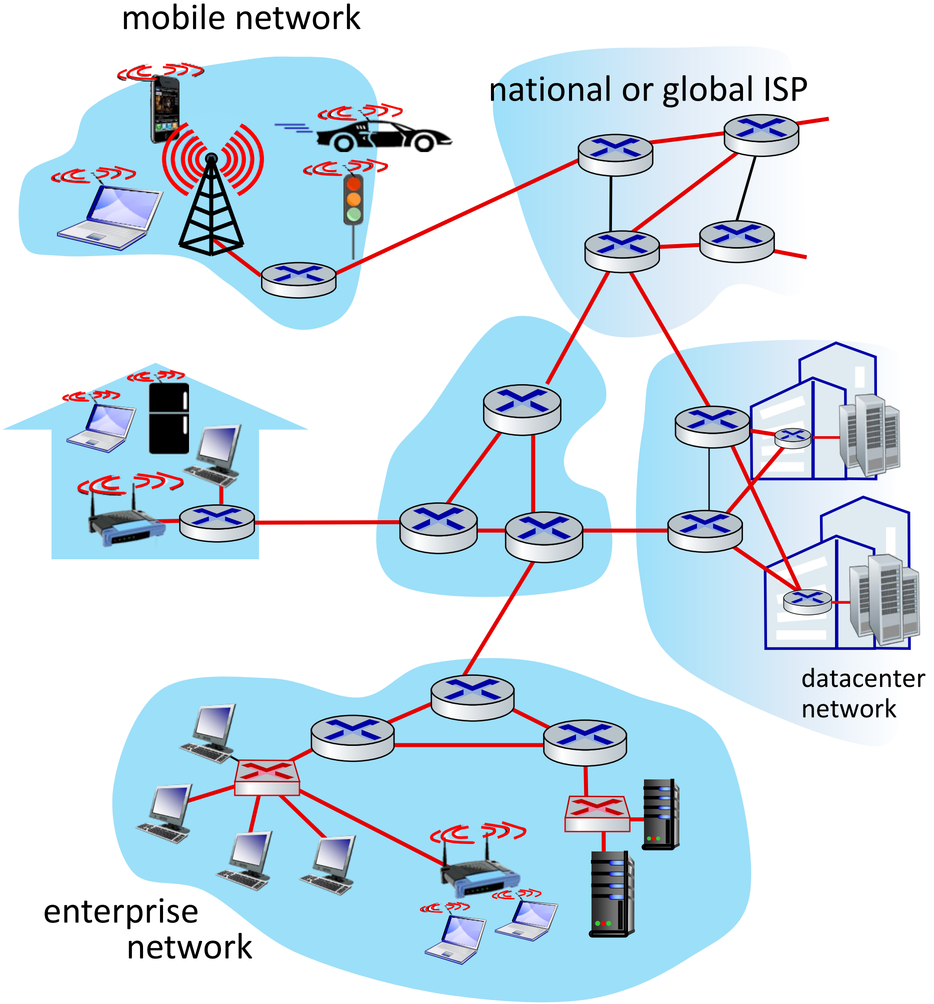
- a day in the life of a web request

Link layer: introduction

terminology:

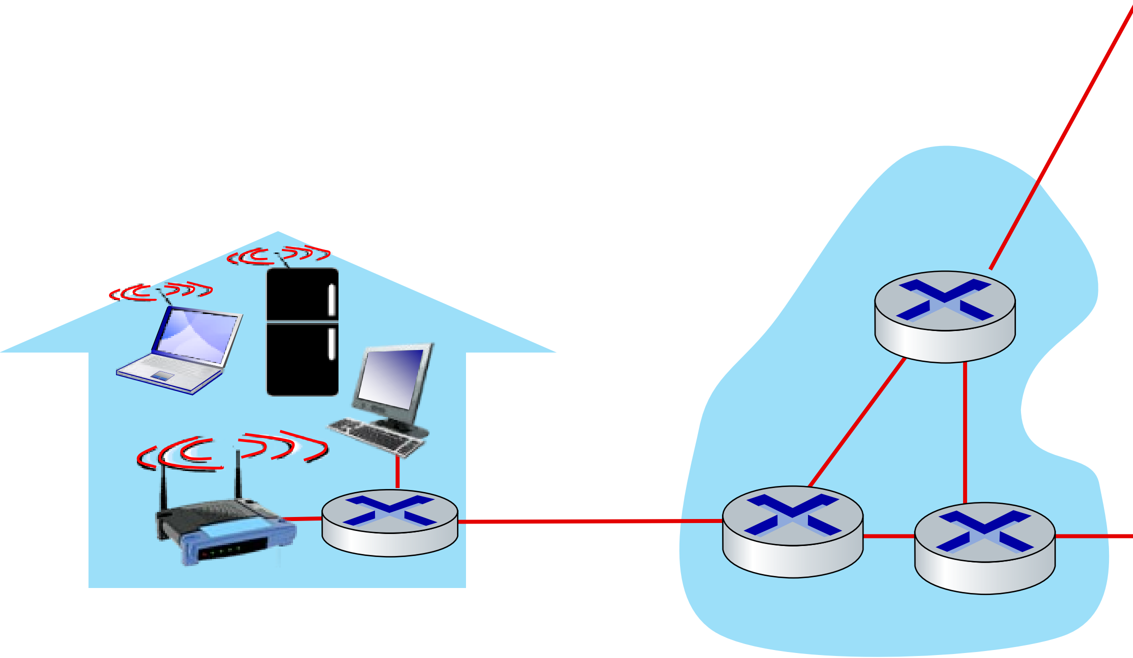
- hosts, routers: nodes
- communication channels that connect adjacent nodes along communication path: links
  - wired , wireless
  - LANs
- layer-2 packet: *frame*, encapsulates datagram

link layer has responsibility of transferring datagram from one node to physically adjacent node over a link

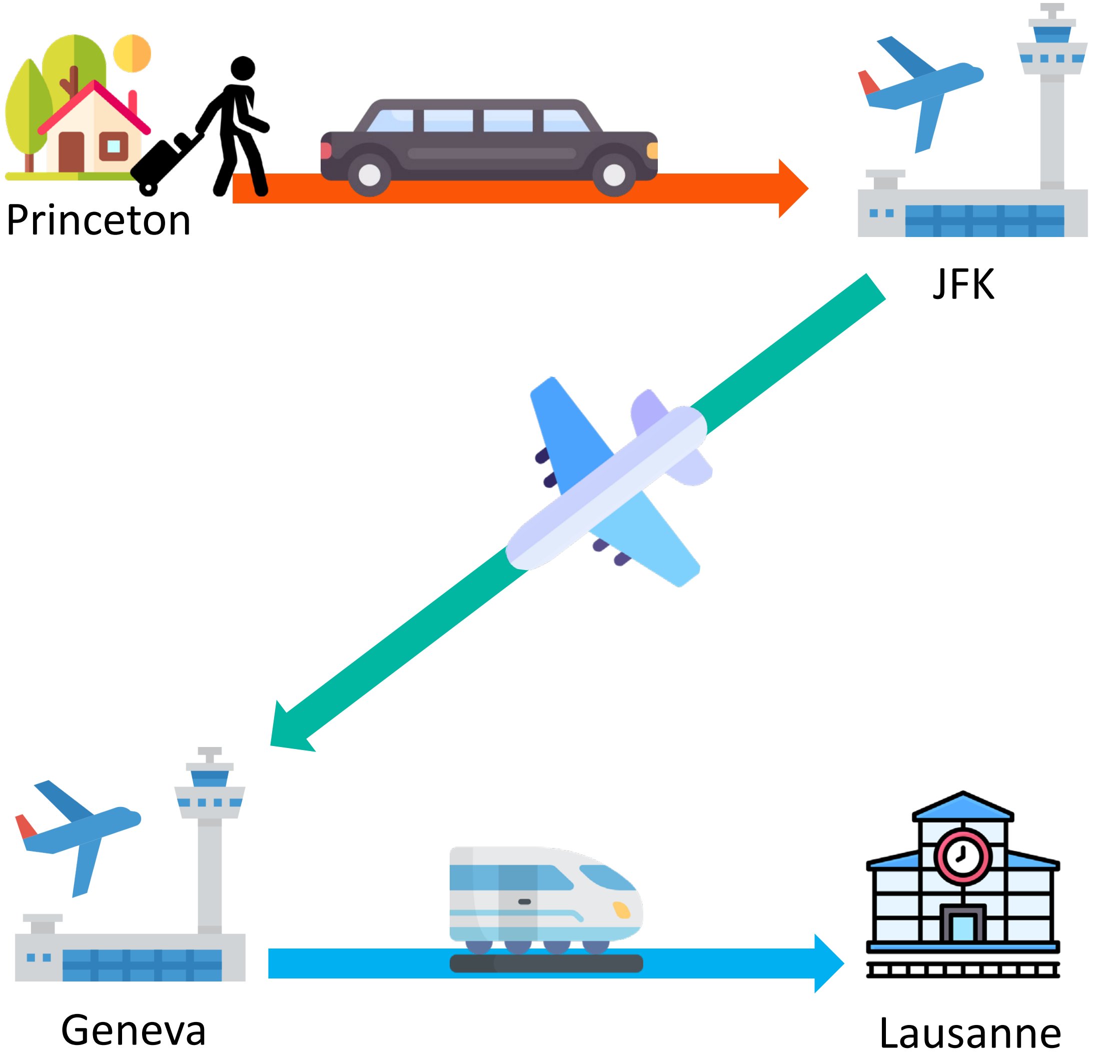


Link layer: context

- datagram transferred by different link protocols over different links:
  - e.g., WiFi on first link, Ethernet on next link
- each link protocol provides different services
  - e.g., may or may not provide reliable data transfer over link



Transportation analogy

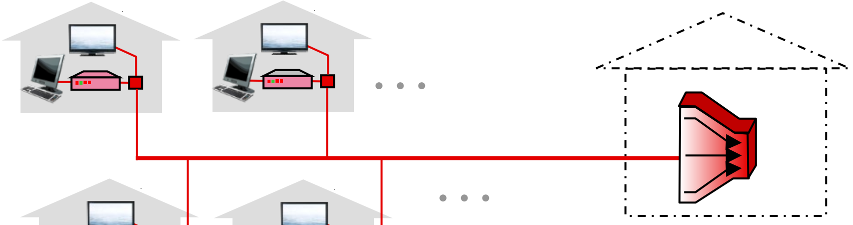


transportation analogy:

- trip from Princeton to Lausanne
  - limo: Princeton to JFK
  - plane: JFK to Geneva
  - train: Geneva to Lausanne
- tourist = datagram
- transport segment = communication link
- transportation mode = link-layer protocol
- travel agent = routing algorithm

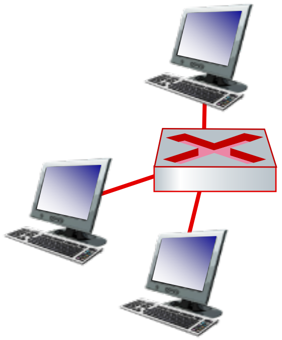
Link layer: services

- framing, link access:
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses in frame headers identify source, destination (different from IP address!)
- reliable delivery between adjacent nodes
  - we already know how to do this!
  - seldom used on low bit-error links
  - wireless links: high error rates
    - ***Q:*** **why both link-level and end-end reliability?**



Cable access

cellular

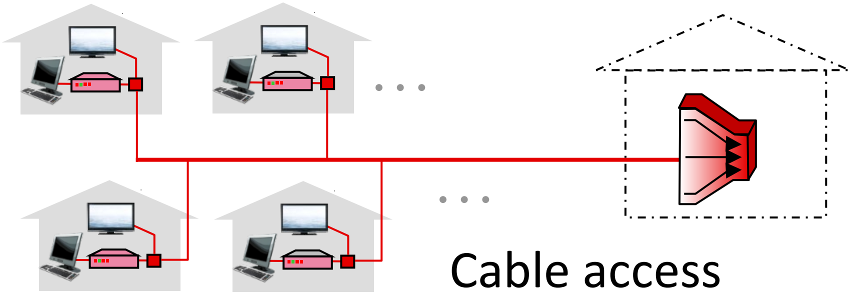


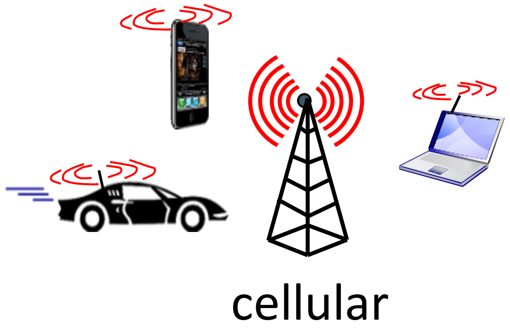
Ethernet LANs

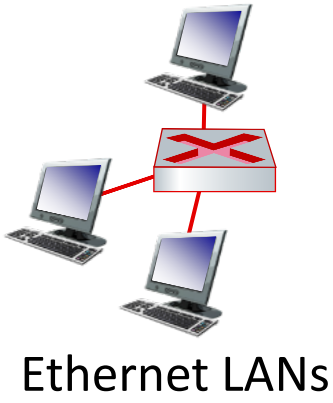
WiFi

Link layer: services (more)

- flow control:
  - pacing between adjacent sending and receiving nodes
- error detection:
  - errors caused by signal attenuation, noise.
  - receiver detects errors, signals retransmission, or drops frame
- error correction:
  - receiver identifies *and corrects* bit error(s) without retransmission
- half-duplex and full-duplex:
  - with half duplex, nodes at both ends of link can transmit, but not at same time

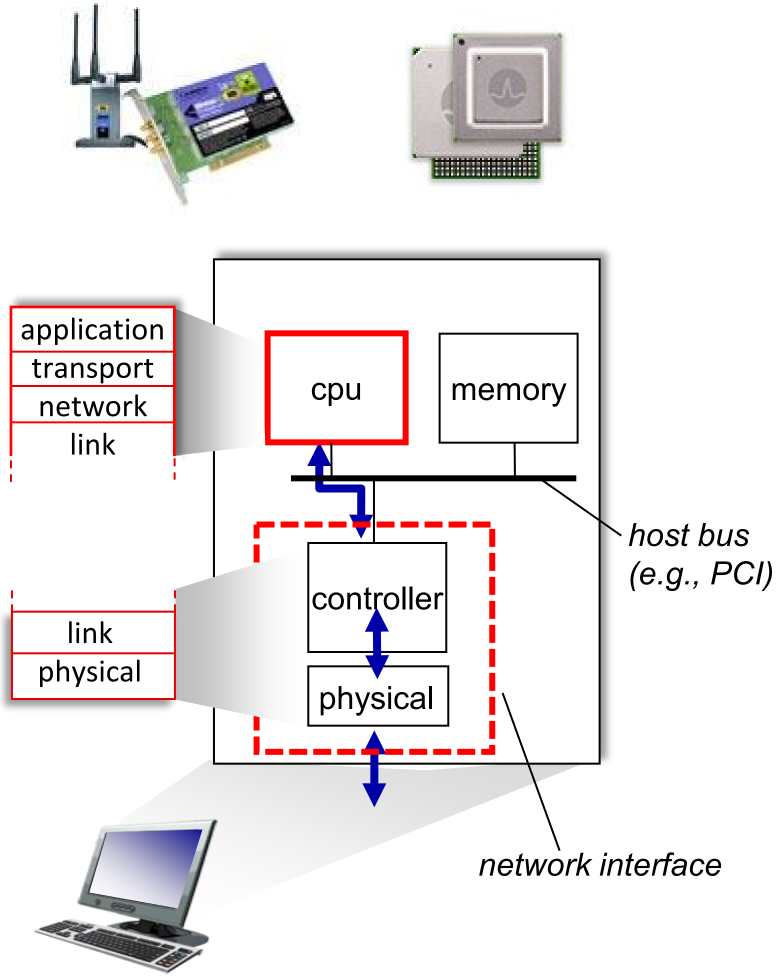




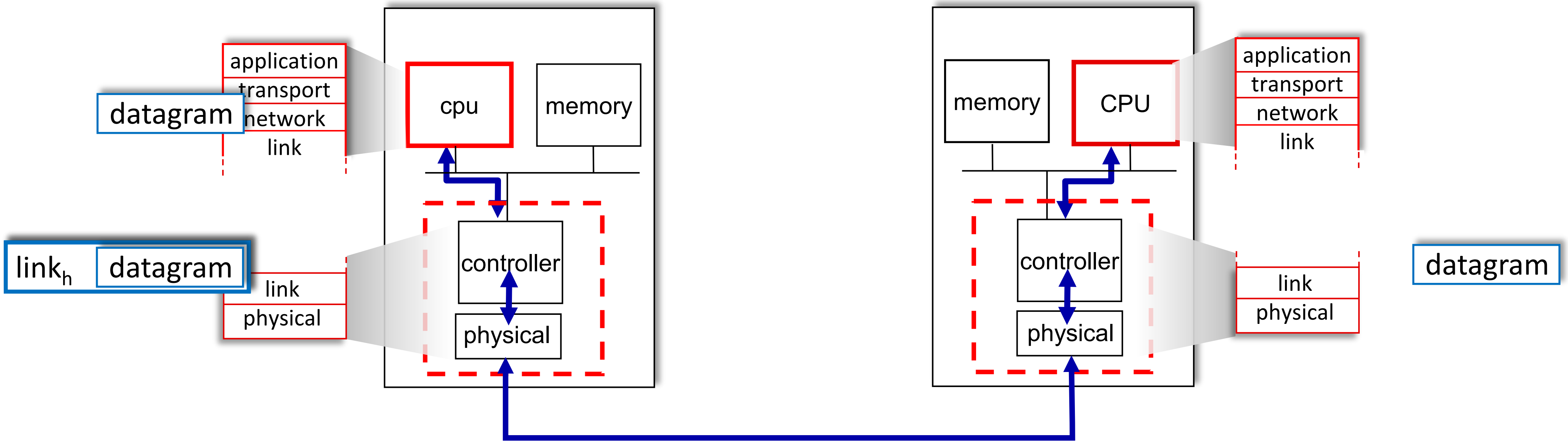


Host link-layer implementation

- in each-and-every host
- link layer implemented on-chip or in network interface card (NIC)
    - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware



Interfaces communicating



sending side:

- encapsulates datagram in frame
- adds error checking bits, reliable data transfer, flow control, etc.

receiving side:

- looks for errors, reliable data transfer, flow control, etc.
- extracts datagram, passes to upper layer at receiving side

Link layer, LANs: roadmap

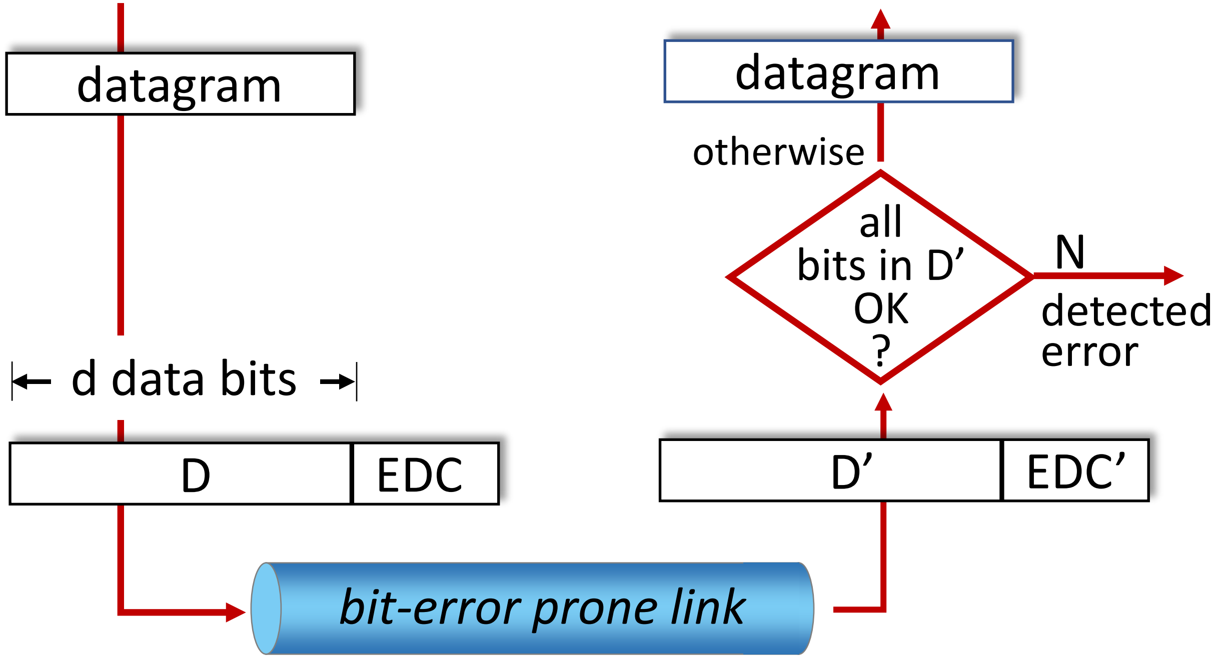
- introduction
- error detection, correction
- multiple access protocols
- LANs
  - addressing, ARP
  - Ethernet
  - switches
  - VLANs
- link virtualization: MPLS
- data center networking

- a day in the life of a web request

Error detection

EDC: error detection and correction bits (e.g., redundancy)

D: data protected by error checking, may include header fields



Error detection not 100% reliable!

- protocol may miss some errors, but rarely
- larger EDC field yields better detection and correction

Parity checking

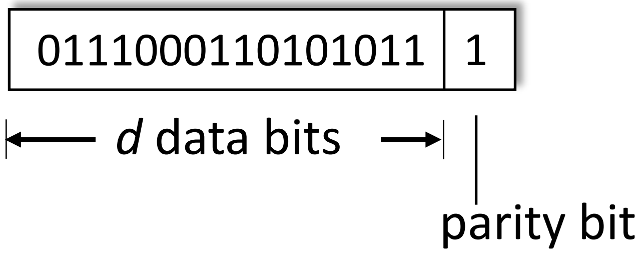
- Transmit data as a block of i rows of j bits per row and add parity bit to each row and each column.
- The bit in the lower-right corner is a parity bit that is the sum of row and column parities.
- If sum of the row parity errors and column parity errors is odd, then the parity bit in the lower-right corner also has error, so add 1 to the total number of parity bit errors.

IMPORTANT

Parity checking

single bit parity:

- detect single bit errors



Even/odd parity: set parity bit so there is an even/odd number of 1's

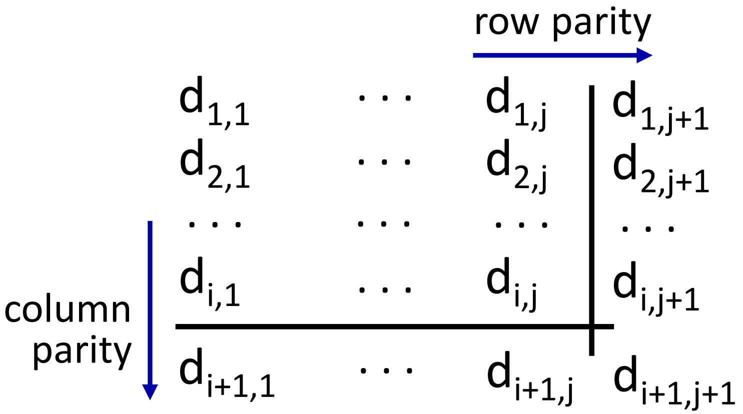
At receiver:

- compute parity of *d* received bits
  - e.g., even parity: 0 if even number of 1's; 1 otherwise
- compare with received parity bit – if different than error detected
- All rows are concatenated and sent out,

e.g., 1 0 1 0 1 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 1 0 1 0

Can detect *and* correct errors (without retransmission!)

- two-dimensional parity: detect *and correct* single bit errors

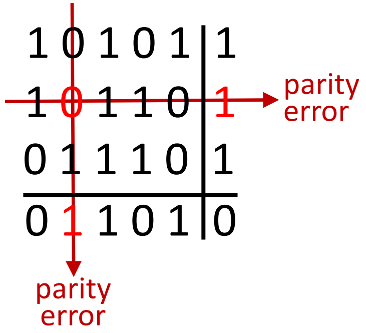


$$\begin{array}{cccc|c} d_{1,1} & \cdots & d_{1,j} & & d_{1,j+1} \\ d_{2,1} & \cdots & d_{2,j} & & d_{2,j+1} \\ \cdots & \cdots & \cdots & & \cdots \\ d_{i,1} & \cdots & d_{i,j} & & d_{i,j+1} \\ \hline d_{i+1,1} & \cdots & d_{i+1,j} & & d_{i+1,j+1} \end{array}$$

no errors:

$$\begin{array}{ccccc|c} 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 \end{array}$$

detected and correctable single-bit error:



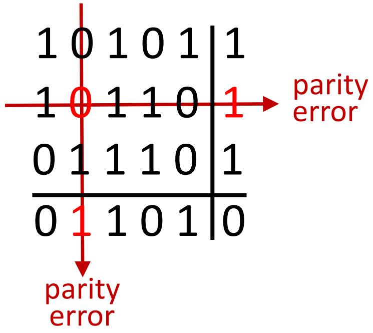
$$\begin{array}{ccccc|c} 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ \hline 0 & 1 & 1 & 0 & 1 & 0 \end{array}$$

IMPORTANT

Quiz: Number of Parity Errors

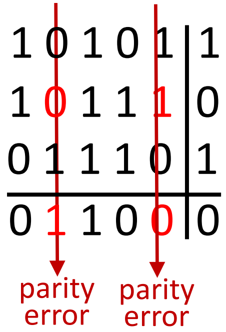
no errors:

```
1 0 1 0 1 | 1
1 1 1 1 0 | 0
0 1 1 1 0 | 1
-------------
0 0 1 0 1 | 0
```



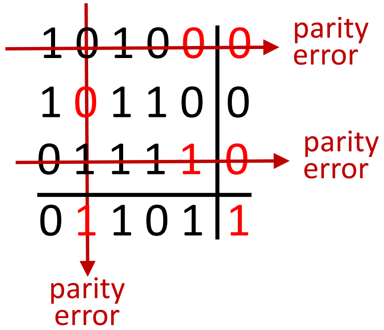
```
1 0 1 0 1 | 1
1 0 1 1 0 | 1
0 1 1 1 0 | 1
-------------
0 1 1 0 1 | 0
```

1-bit error causes 1 row parity error, 1 column parity error



```
1 0 1 0 1 | 1
1 0 1 1 1 | 0
0 1 1 1 0 | 1
-------------
0 1 1 0 0 | 0
```

2-bit error causes 2 row parity errors

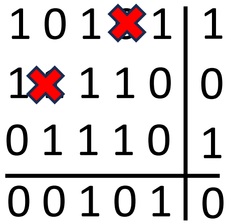


```
1 0 1 0 0 | 0
1 0 1 1 0 | 0
0 1 1 1 1 | 0
-------------
0 1 1 0 1 | 1
```

3-bit error causes 1 row parity error, 2 column parity error, and 1 corner parity error

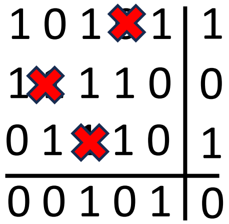
```
1 1 1 0 0 | 1
1 1 1 1 0 | 0
0 0 1 1 1 | 1
-------------
0 0 1 0 1 | 0
```

4-bit error causes NO parity error, hence cannot be detected



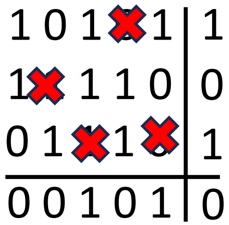
```
1 0 1 ✖ 1 | 1
1 ✖ 1 1 0 | 0
0 1 1 1 0 | 1
-------------
0 0 1 0 1 | 0
```

2-bit error causes
___row parity errors
___column parity errors
___row parity errors



```
1 0 1 ✖ 1 | 1
1 ✖ 1 1 0 | 0
0 1 ✖ 1 0 | 1
-------------
0 0 1 0 1 | 0
```

2-bit error causes
___row parity errors
___column parity errors
___row parity errors



```
1 0 1 ✖ 1 | 1
1 ✖ 1 1 0 | 0
0 1 ✖ 1 ✖ | 1
-------------
0 0 1 0 1 | 0
```

2-bit error causes
___row parity errors
___column parity errors
___row parity errors

IMPORTANT

Internet checksum (review, see section 3.3)

Goal: detect errors (*i.e.,* flipped bits) in transmitted segment

sender:

- treat contents of UDP segment (including UDP header fields and IP addresses) as sequence of N-bit integers, where N may be 4, 8, 16…
- checksum: addition (one's complement sum) of the sequence of integers
  - One's complement sum is defined as sum modulo 2^N, and adding any overflow of high order bits back into low-order bits, then taking one's complement (invert all bits)
- checksum value put into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - not equal - error detected
  - equal - no error detected. (*But maybe errors nonetheless)*

IMPORTANT

Internet checksum: an example

One’s complement sum for 16-bit integers is defined as sum modulo 2^N, N=16, and adding any overflow of high order bits back into low-order bits, then taking one’s complement

		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
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

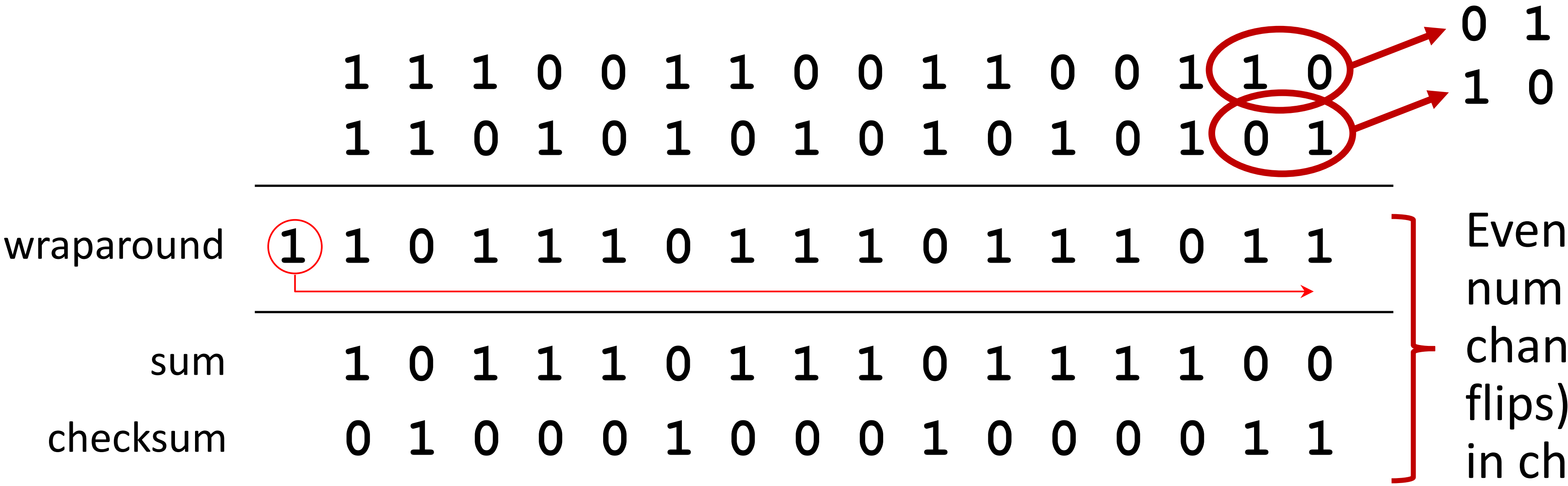
Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

IMPORTANT

Internet checksum: weak protection!

example: add two 16-bit integers



```
              1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0      0 1
              1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1      1 0
            ---------------------------------
wraparound  1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
            ---------------------------------
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
```

Even though numbers have changed (bit flips), *no* change in checksum!

Example in Decimal Number

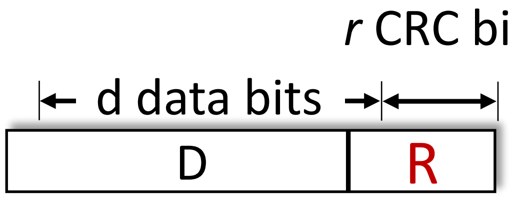
- Make a number divisible by 9
  - **Example**: 823 is to be sent

1. Left-shift: 8230
2. Divide by 9, find remainder: 4
3. Subtract remainder from 9: 9-4=5
4. Add the result of step 3 to step 1: 8235
5. Check that the received result is divisible by 9. If not, then has error

- Detects all single-digit errors: 7235, 8335, 8255, 8237
- Detects several multiple-digit errors: 8765, 7346, 7335, 8775, ...
- Does not detect transpositions: 2835

Cyclic Redundancy Check (CRC)

- more powerful error-detection coding
- D: data bits (given, think of these as a binary number)
- G: bit pattern (generator), of *r+1* bits (given, specified in CRC standard)



bits to send

$\langle D,R\rangle = D * 2^r \text{ XOR } R$

formula for these bits

sender: compute *r* CRC bits, R, such that <D,R> *exactly* divisible by G (mod 2)

- receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
- can detect all burst errors less than r+1 bits
- widely used in practice (Ethernet, 802.11 WiFi)

Exclusive OR (XOR) for Modulo 2 Arithmetic

- 0 XOR 0 = 0 (Same Bits)
- 1 XOR 1 = 0 (Same Bits)
- 1 XOR 0 = 1 (Different Bits)
- 0 XOR 1 = 1 (Different Bits)

- With XOR arithmetic, addition + and subtraction - operations are the same
  - 1111 + 1010
  - = 1111 - 1010
  - = 1111 XOR 1010
  - = 0101

Modulo 2 Addition, Multiplication

Addition:

1-bit

$$\begin{array}{r} 1 \\ +1 \\ \hline 0 \end{array} \quad \begin{array}{r} 0 \\ +0 \\ \hline 0 \end{array} \quad \begin{array}{r} 0 \\ +1 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ +0 \\ \hline 1 \end{array}$$

2-bit

$$\begin{array}{r} 00 \\ +11 \\ \hline 11 \end{array} \quad \begin{array}{r} 01 \\ +11 \\ \hline 10 \end{array} \quad \begin{array}{r} 10 \\ +11 \\ \hline 01 \end{array} \quad \begin{array}{r} 11 \\ +11 \\ \hline 00 \end{array}$$

3-bit

$$\begin{array}{r} 110 \\ +101 \\ \hline 011 \end{array}$$

Multiplication:

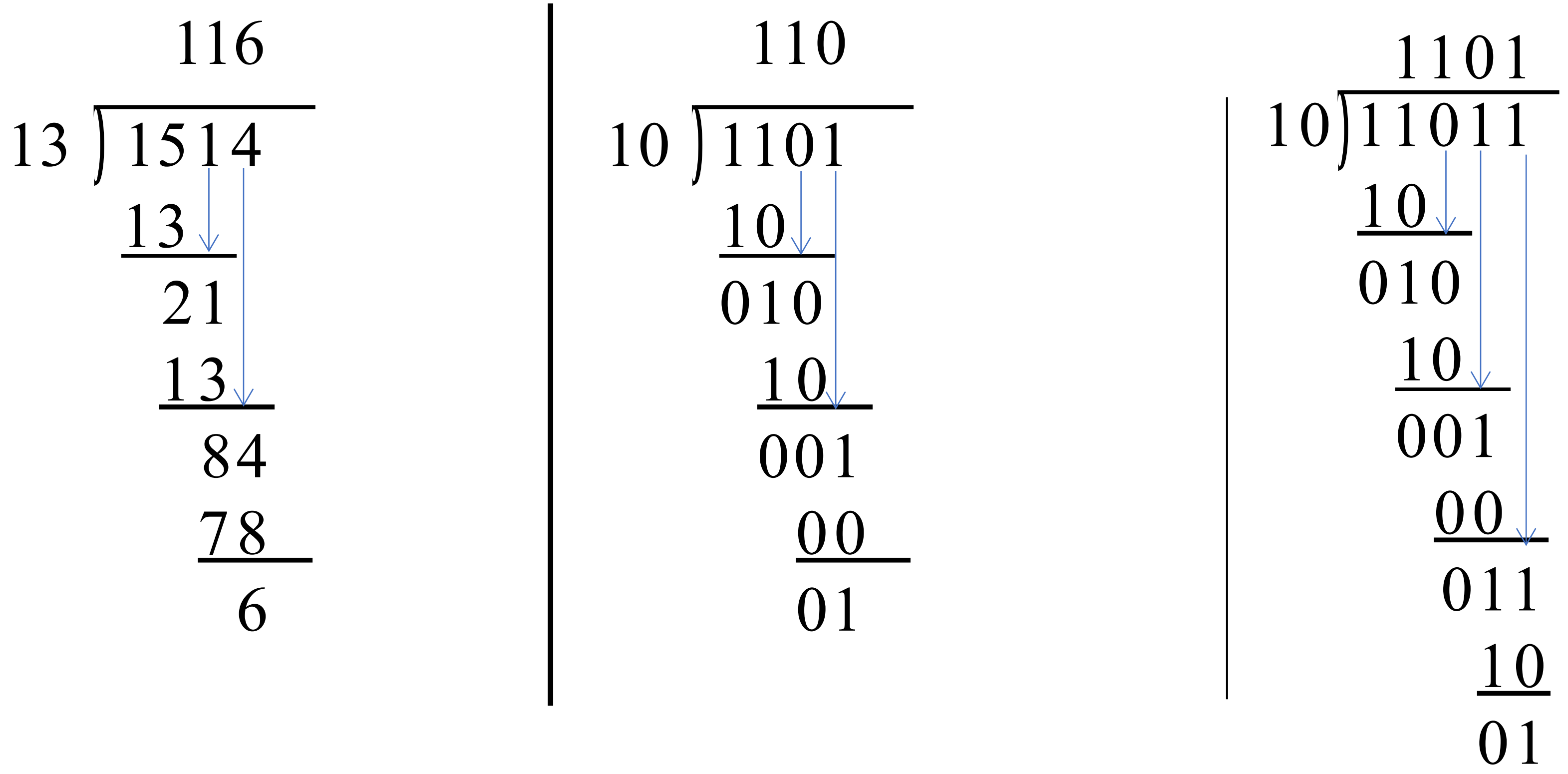
1-bit

$$\begin{array}{r} 1 \\ \times 1 \\ \hline 1 \end{array} \quad \begin{array}{r} 0 \\ \times 0 \\ \hline 0 \end{array} \quad \begin{array}{r} 0 \\ \times 1 \\ \hline 0 \end{array} \quad \begin{array}{r} 1 \\ \times 0 \\ \hline 0 \end{array}$$

2-bit

$$\begin{array}{r} 00 \\ \times 11 \\ \hline 00 \\ 00 \\ \hline 000 \end{array} \quad \begin{array}{r} 01 \\ \times 11 \\ \hline 01 \\ 01 \\ \hline 011 \end{array} \quad \begin{array}{r} 10 \\ \times 11 \\ \hline 10 \\ 10 \\ \hline 110 \end{array} \quad \begin{array}{r} 11 \\ \times 11 \\ \hline 11 \\ 11 \\ \hline 101 \end{array}$$

Modulo 2 Division



116
13) 1514
13
21
13
84
78
6

110
10) 1101
10
010
10
001
00
01

1101
10)11011
10
010
10
001
00
011
10
01

Decimal Arithmetic:
1514/13=116, w/ remainder 6

Mod-2 Arithmetic:
1101/10=110, w/ remainder 01
11011/10=1101, w/ remainder 01

Cyclic Redundancy Check (CRC): Example 1

Sender wants to compute R such that for some integer *n*:

$$D \cdot 2^r \;\; XOR \;\; R = nG$$

... or equivalently (XOR R both sides):

$$D \cdot 2^r = nG \;\; XOR \;\; R$$

... which says:

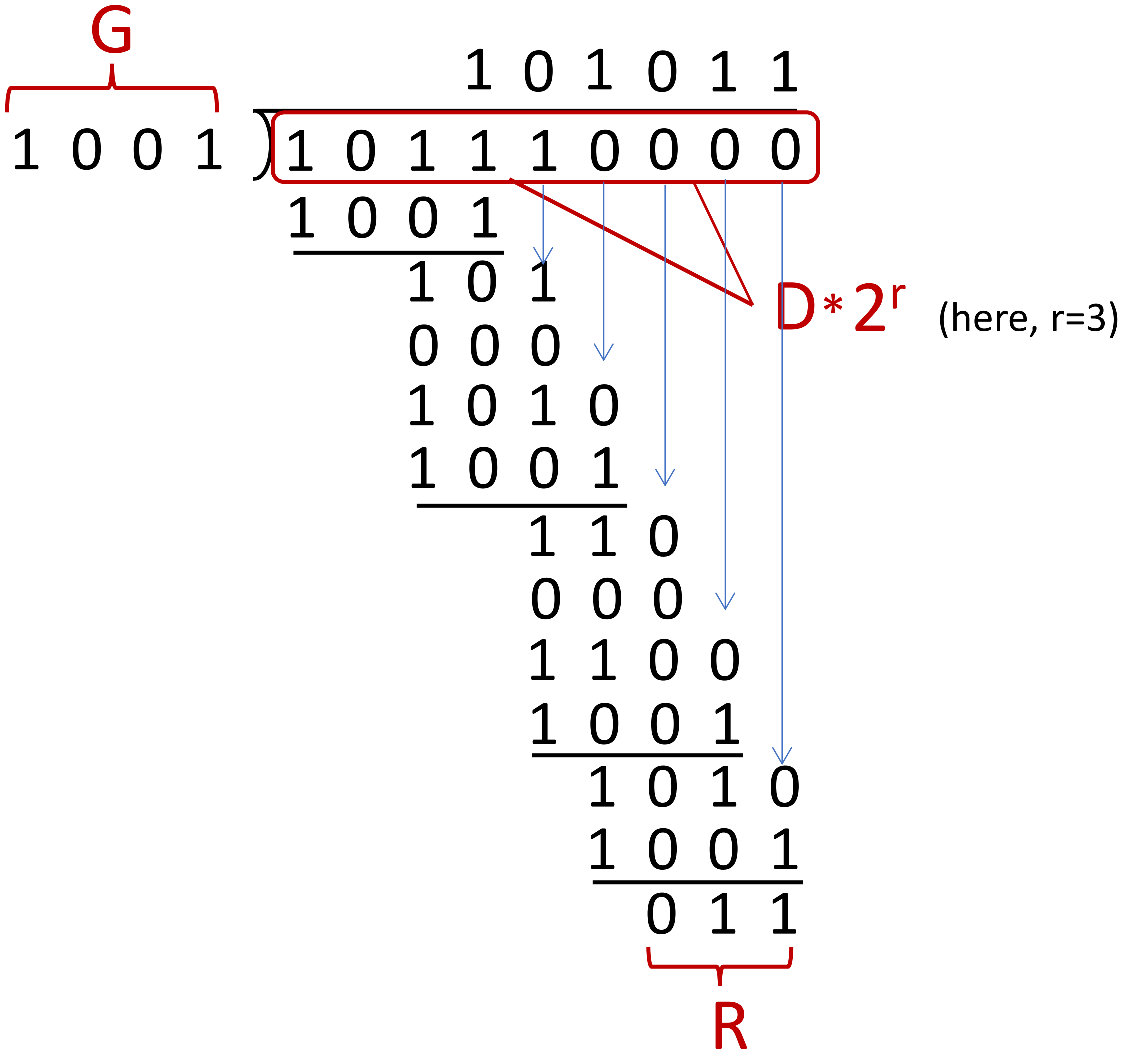
if we divide $D \cdot 2^r$ by G, we want remainder R to satisfy:

$$\boxed{R = remainder\left[\frac{D \cdot 2^r}{G}\right]}$$

algorithm for computing R

Example: D=101110, G=1001, r=3

$D \cdot 2^r$ = 101110000, R=011



```
                1 0 1 0 1 1
1 0 0 1 ) 1 0 1 1 1 0 0 0 0
          1 0 0 1
              1 0 1
              0 0 0
              1 0 1 0
              1 0 0 1
                  1 1 0
                  0 0 0
                  1 1 0 0
                  1 0 0 1
                    1 0 1 0
                    1 0 0 1
                      0 1 1
```

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Cyclic Redundancy Check (CRC): Example 1

- $D \cdot 2^r \ + \ R = D \cdot 2^r \ \ XOR \ \ R$

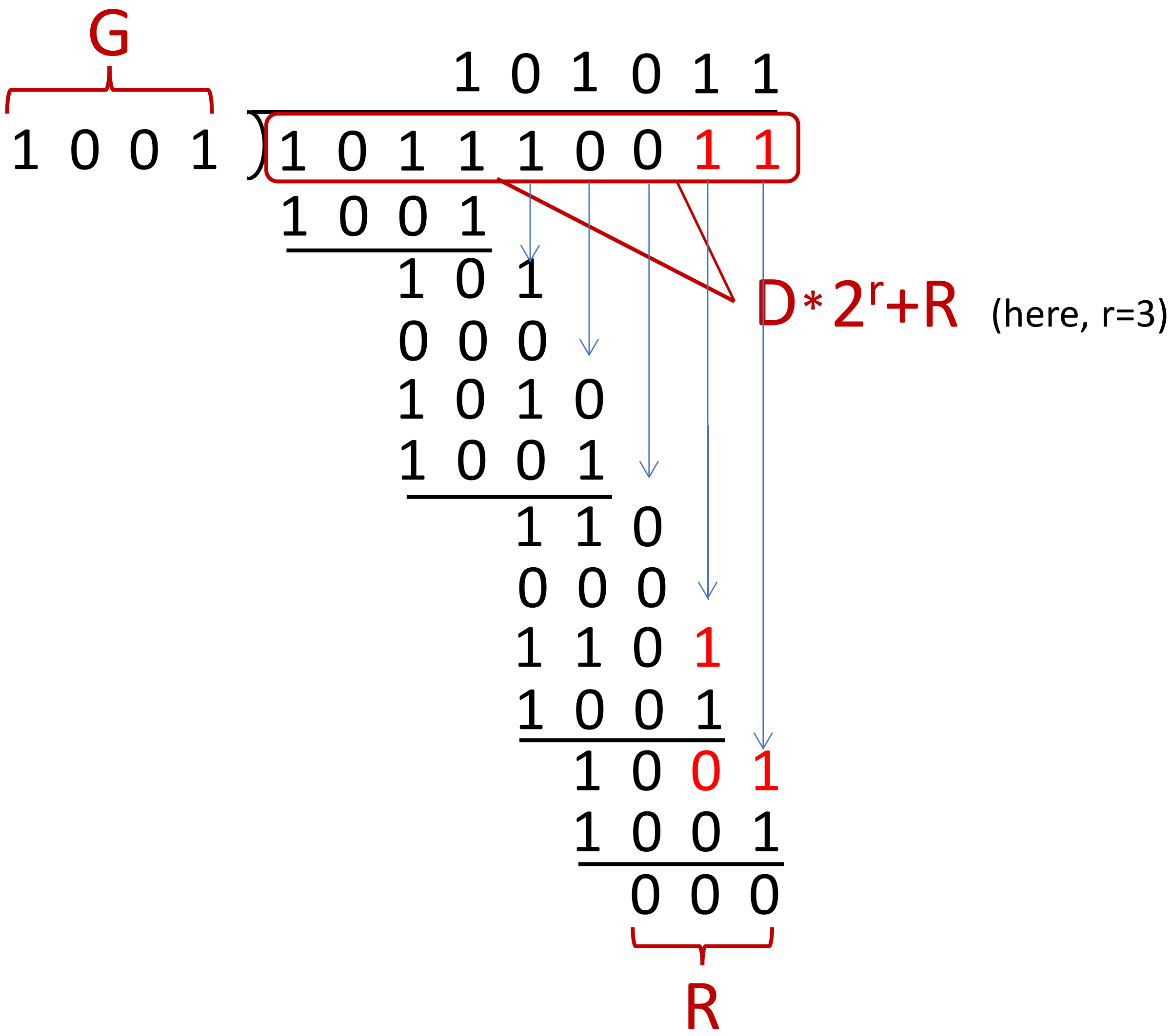
$= 101110000 \ XOR \ 011$

$= 101110011$

It must be divisible by G=1001,
i.e. remainder of the division is 0,

$D \cdot 2^r R$ is divisible by G, or

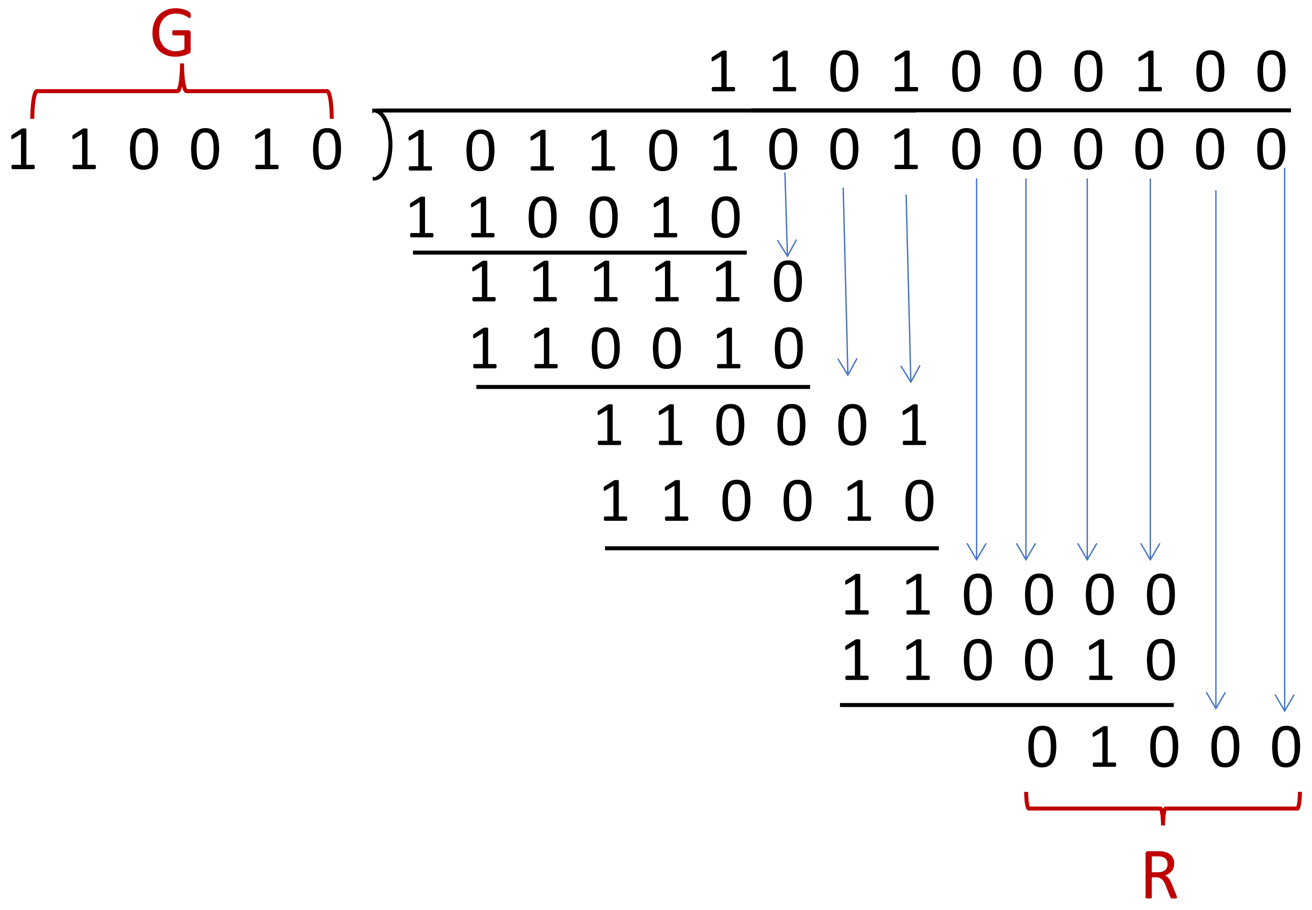
$D \cdot 2^r \ \ XOR \ \ R = nG$



Cyclic Redundancy Check (CRC): Example 2

Example 2: D=1011010010, G=110010, r=5

$D \cdot 2^r = 101101001000000$, R=01000



Error Detection and Correction 2: Cyclic Redundancy Check
https://www.youtube.com/watch?v=6gbkoFciryA

Cyclic Redundancy Check (CRC): Example 3

- $D \cdot 2^r + R = D \cdot 2^r \text{ XOR } R$

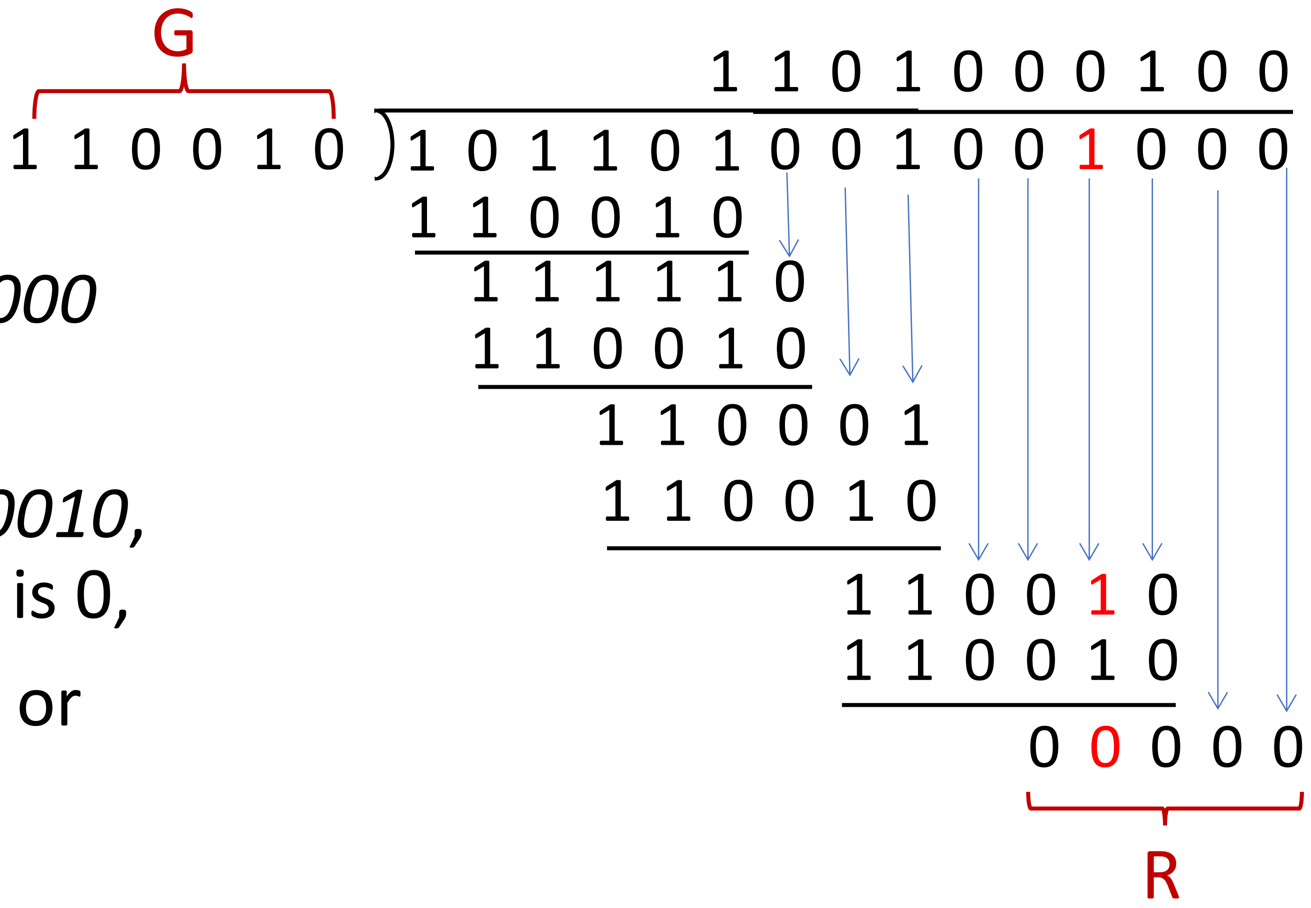
$= 101101001000000 \text{ XOR } 01000$

$= 101101001001000$

It must be divisible by G= *110010*, i.e. remainder of the division is 0,

$D \cdot 2^r \text{ XOR } R$ is divisible by G, or

$D \cdot 2^r \text{ XOR } R = nG$



Quiz 1 Parity Checking

- Q: Compute the parity bits for the following data matrix:
  - 1 0 1
  - 0 1 0
  - 1 1 1
- A:
  - 1 0 1 | 0
  - 0 1 0 | 1
  - 1 1 1 | 1
  - ------------
  - 0 0 0 | 0

- Q: Compute the parity bits for the following data matrix:
  - 1 0 1
  - 0 0 0
  - 1 0 1
- A:
  - 1 0 1 |
  - 0 0 0 |
  - 1 0 1 |
  - ------------
  - |

Quiz 2 Internet checksum

- Q: Suppose that a packet 1001 1100 1010 0011 is transmitted using Internet checksum (N=4-bit integer). What is the value of the checksum?
- A: One's complement sum for 4-bit integers is defined as sum modulo 2^N, N=4, and adding any overflow of high order bits back into low-order bits, then taking one's complement.
- 0011+1010 = 1101
- 1101+1100 = 1001+1 = 1010
- 1010+1001 = 0011+1 = 0100.
- So, the Internet checksum is 1011, the one's complement of 0100.

Quiz 3 CRC

- Q: A bit stream 1001 is transmitted using the standard CRC method. The generator is 1011. Show the actual bit string transmitted. Suppose that the first bit sent is inverted during transmission error. Show that this error is detected at the receiver's end. Give an example of bit errors in the bit string transmitted that will not be detected by the receiver.
- A: *D=1001, G=1011, r=3*. Compute *R*
- The message after appending three zeros is 1001000. The remainder on dividing 1001000 by 1011 is 110. So, the actual bits transmitted are 1001110.
- The received bit stream with an error in the first bit is 0001110. Dividing this by 1011 produces a remainder of 101, which is different from 0. Thus, the receiver detects an error.
- If the transmitted bit stream is converted to any multiple of 1001, the error will not be detected. A trivial example is if all ones in the bit stream are inverted to 0.

Quiz 4 CRC

- *D=110011, G=1001, r=3. Compute R*

- *D=1001100, G=1011, r=3.* Compute *R*