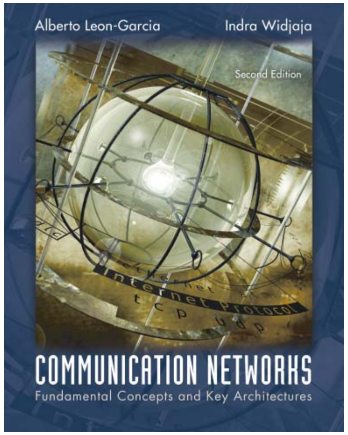
Chapter 3
Digital Transmission Fundamentals



Error Detection and Correction

CSE 3213, Winter 2010

Instructor: Foroohar Foroozan

Modulo-2 Arithmetic

Modulo 2 arithmetic is performed digit by digit on binary numbers.
Each digit is considered independently from its neighbours.
Numbers are not carried or borrowed.

$0 \oplus 0 = 0 \qquad 1 \oplus 1 = 0$

a. Two bits are the same, the result is 0.

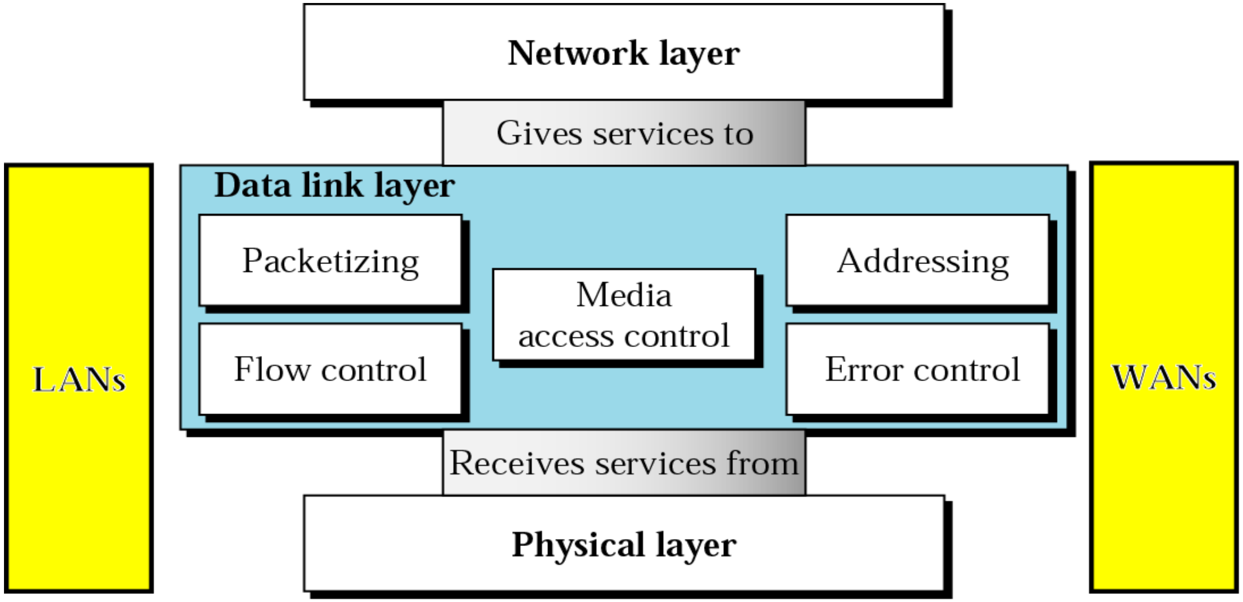
$0 \oplus 1 = 1 \qquad 1 \oplus 0 = 1$

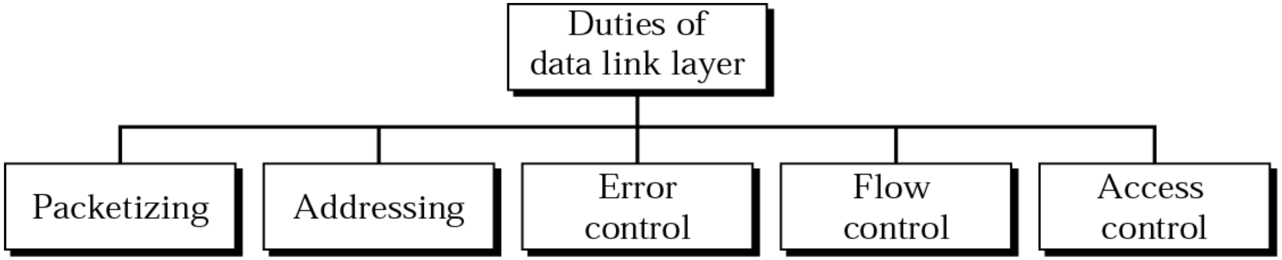
b. Two bits are different, the result is 1.

$$\begin{array}{cccccc} & 1 & 0 & 1 & 1 & 0 \\ \oplus & 1 & 1 & 1 & 0 & 0 \\ \hline & 0 & 1 & 0 & 1 & 0 \end{array}$$

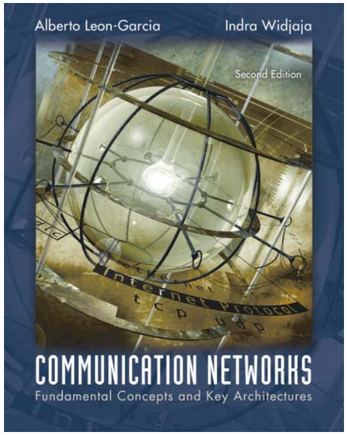
c. Result of XORing two patterns

Data Link layer





Chapter 3
Digital Transmission Fundamentals



Error Detection and Correction

Error Control

Why Error Control?

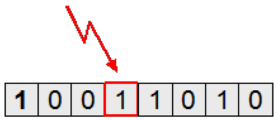
– data sent from one computer to another should be transferred reliably – unfortunately, the physical link cannot guarantee that all bits, in each frame, will be transferred without errors

- error control techniques are aimed at improving the error-rate performance offered to upper layer(s), i.e. end-application

Probability of Single-Bit Error

– aka bit error rate (BER) :

- wireless medium: $p_b = 10^{-3}$
- copper-wire: $p_b = 10^{-6}$
- fibre optics: $p_b = 10^{-9}$



Approaches to Error Control

(1) Error Detection + Automatic Retransmiss. Request (ARQ)

- fewer overhead bits ☺
- return channel required ☹
- longer error-correction process and waste of bandwidth when errors are detected ☹

(2) Forward Error Correction (FEC)

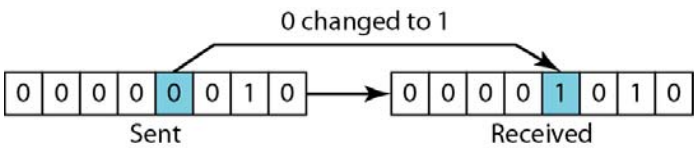
- error detection + error correction

Error Control

Types of Errors

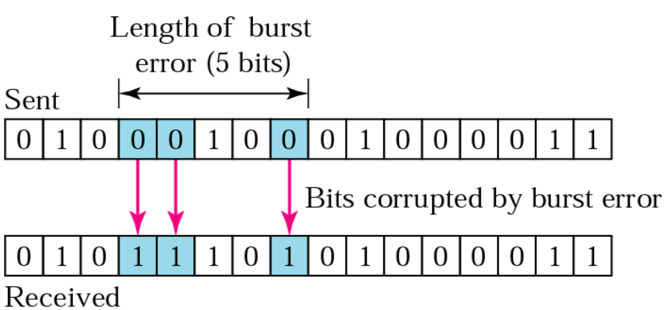
(1) Single Bit Errors

- only one bit in a given data unit (byte, packet, etc.) gets corrupted



(2) Burst Errors

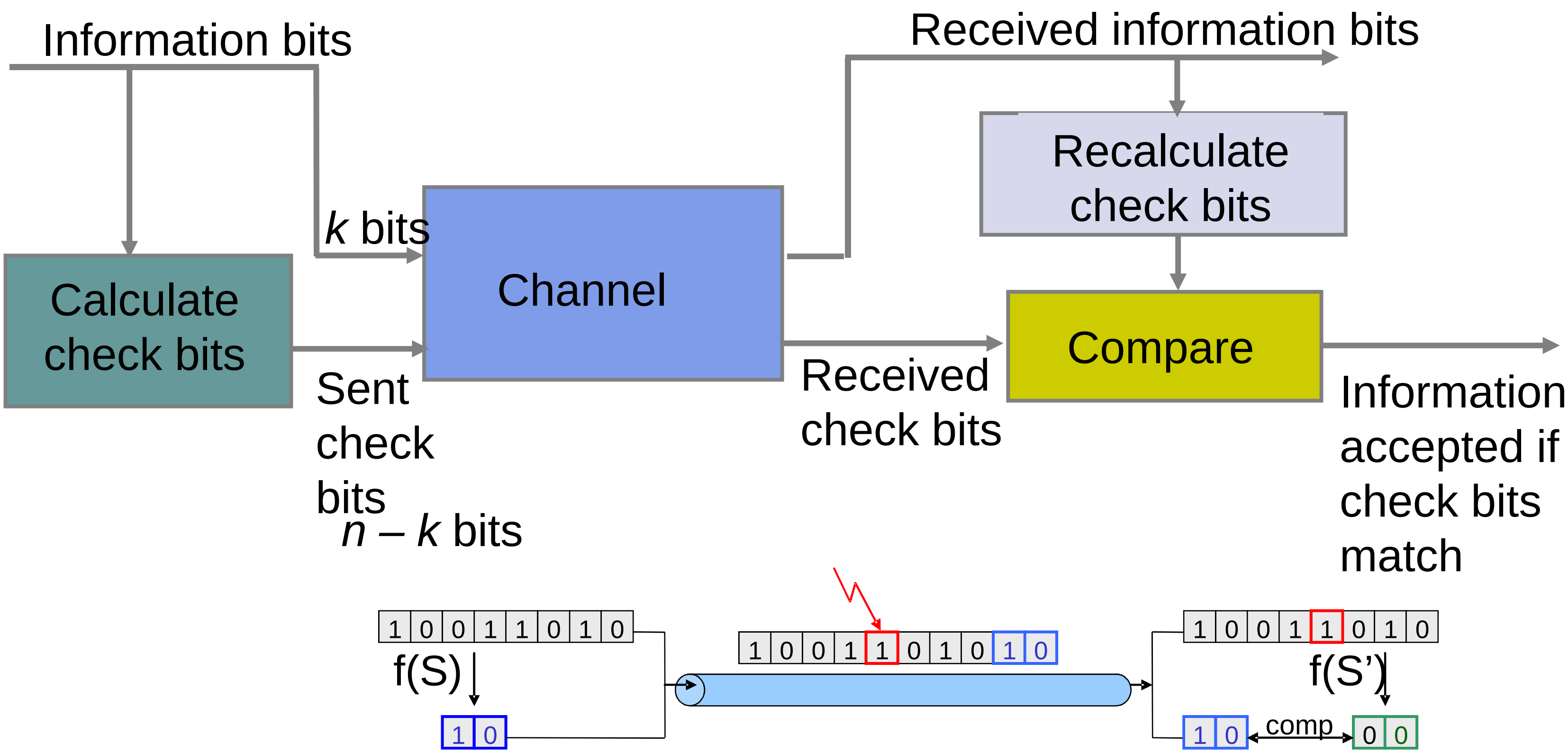
- two or more bits in the data unit have been corrupted
- errors do not have to occur in consecutive bits
- burst errors are typically caused by external noise (environmental noise)
- burst errors are more difficult to detect / correct



Key Idea

– redundancy!!! – add enough extra information (bits) for detection / correction of errors at the destination

- redundant bits = 'compressed' version of original data bits
- error correction requires more redundant bits than error detection
- more redundancy bits ⇒ better error control ☺ ⇒ more overhead ☹



Hamming Distance

Hamming Distance between 2 Codes

– number of differences between corresponding bits
  - can be found by applying XOR on two codewords and counting number of 1s in the result

Minimum Hamming Distance (d_{min}) in a Code

– minimum Hamming distance between all possible pairs in a set of codewords
  - d_{min} bit errors will make one codeword look like another
  - larger d_{min} – better robustness to errors

Example [k=2, n=5 code]

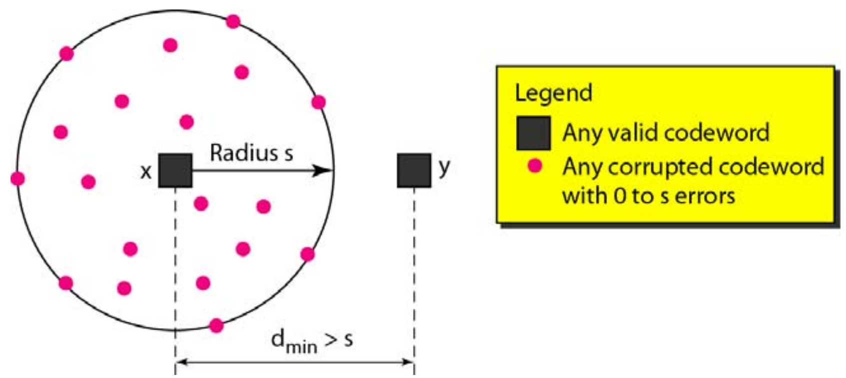
Code that adds 3 redundant bits to every 2 information bits, thus resulting in 5-bit long codewords.

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

Hamming Distance

Minimum Hamming Distance for Error Detection – to guarantee detection of up to s errors in all cases, the minimum Hamming distance must be

$$d_{min} = s + 1$$



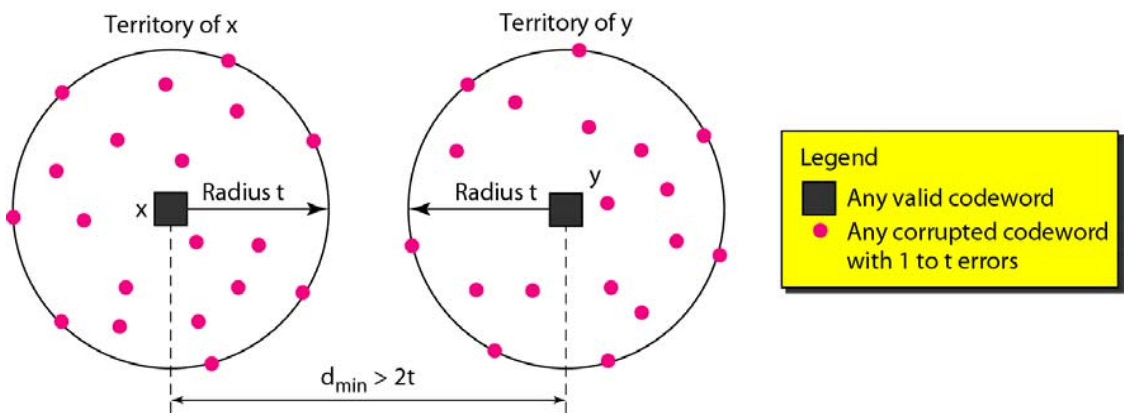
Example **[code with d_{min}=2 is able to detect s=1 bit-errors]**

Datawords	*Codewords*
00	000
01	011
10	101
11	110

Hamming Distance

Minimum Hamming Distance for Error Correction – to guarantee correction of up to t errors in all cases, the minimum Hamming distance must be

$$d_{min} = 2t + 1$$



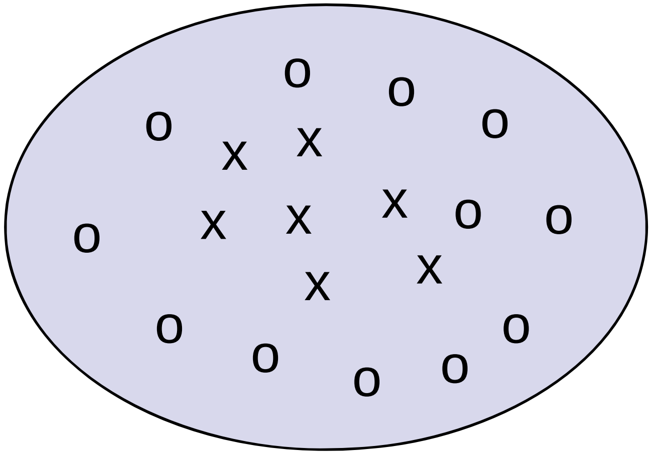
Example [Hamming distance]

A code scheme has a Hamming distance d_{min}=4. What is the error detection and error correction capability of this scheme?

The code guarantees the detection of up to three errors (s=3), but it can correct only 1-bit errors!

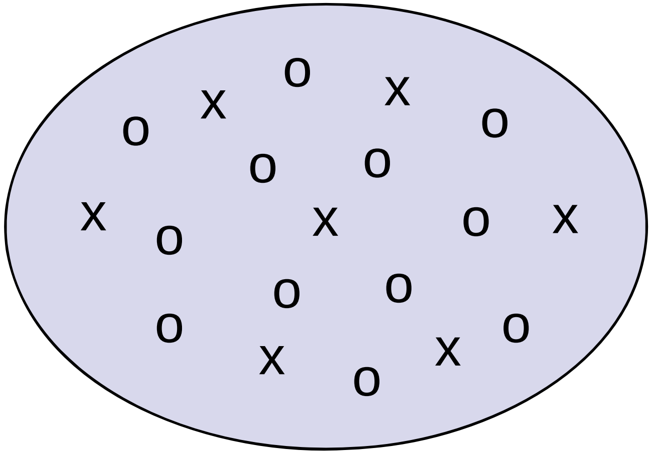
What is a good code?

- Many channels have preference for error patterns that have fewer # of errors
- These error patterns map transmitted codeword to nearby *n*-tuple
- If codewords close to each other then detection failures will occur
- Good codes should maximize separation between codewords



Poor distance properties

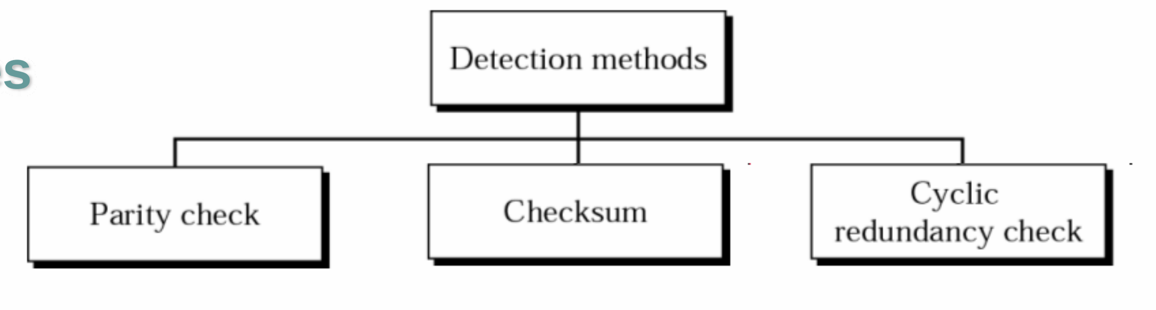
x = codewords
o = **noncodewords**



Good distance properties

Error Detection: Single Parity Check

Error Detection Techniques



Single Parity Check (Even Parity)

- Append an overall parity check to *k* information bits

  Info Bits: $b_1, b_2, b_3, \ldots, b_k$

  Check Bit: $b_{k+1} = b_1 + b_2 + b_3 + \ldots + b_k$ modulo 2

  Codeword: $(b_1, b_2, b_3, \ldots, b_{k,}, b_{k+!})$

- receiver checks if number of 1s is even
  - receiver <u>CAN DETECT</u> all <u>single-bit errors</u> and <u>burst errors with odd number of corrupted bits</u>
  - single-bit errors CANNOT be CORRECTED – position of corrupted bit remains unknown
  - all <u>even-number burst errors</u> are <u>undetectable</u> !!!

Example of Single Parity Code

- Information (7 bits): (0, 1, 0, 1, 1, 0, 0)
- Parity Bit: $b_8 = 0 + 1 + 0 + 1 + 1 + 0 = 1$
- Codeword (8 bits): (0, 1, 0, 1, 1, 0, 0, 1)
- If single error in bit 3 : (0, 1, 1, 1, 1, 0, 0, 1)
  - # of 1's =5, odd
  - Error detected
- If errors in bits 3 and 5: (0, 1, 1, 1, 0, 0, 0, 1)
  - # of 1's =4, even
  - Error not detected

Error Detection: Single Parity Check

Example **[single parity check code C(5,4)]**

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

Single Parity Check Codes and Minimum Hamming Distance (d_{min}) – for ALL parity check codes, d_{min} = 2

Error Detection: Single Parity Check

Effectiveness of Single Parity Check

original codeword: $b = [b_1\ b_2\ b_3\ \dots b_n]$

1	0	0	1	1	0	0	1

received codeword: $b' = [b_1'\ b_2'\ b_3'\ \dots b_n']$

1	$\boxed{1}$	0	1	1	$\boxed{1}$	0	1

error vector: $e = [e_1\ e_2\ e_3\ \dots e_n]$

0	1	0	0	0	1	0	0

$$e_k = \begin{cases} 1, & \text{if } b_k \neq b_k' \\ 0, & \text{if } b_k = b_k' \end{cases}$$

(1) Random Error Vector Channel Model

– there are 2^n possible error vectors – all error are equally likely

- e.g. e=[0 0 0 0 0 0 0 0] and e=[1 1 1 1 1 1 1 1] are equally likely
- 50% of error vectors have an even # of 1s,
  50% of error vectors have an odd # of 1s
- probability of error detection failure = 0.5
- not very realistic channel model !!!

Error Detection: Single Parity Check

(2) Random Bit Error Channel Model

- bit errors occur independently of each other –
  p_b = probability of error in a single-bit transmission

(2.1) probability of single bit error (*w(e)=1*)

- where *w(e)* represents the number of 1s in e
  - bit-error occurs at an arbitrary (but particular) position

1	0	0	1	1	0	0	0

1	0	1	1	1	0	0	0

$e_1=0$ $e_2=0$ $e_3=1$ $e_{n-2}=0$ $e_{n-1}=0$ $e_n=0$

$$P(w(e)=1) = \underbrace{(1-p_b)}_{\text{probability of correctly transmitted bit}} \cdot (1-p_b) \cdot p_b \cdot \ldots \cdot (1-p_b) \cdot (1-p_b) \cdot (1-p_b)$$

$$\boxed{P(w(e)=1) = (1-p_b)^{n-1} \cdot p_b}$$

Error Detection: Single Parity Check

(2.2) probability of two bit errors: *w(e)=2*

<1, since $p_b < 0.5$

$$P(w(e)=2) = (1-p_b)^{n-2} \cdot (p_b)^2 = \boxed{(1-p_b)^{n-1} \cdot p_b} \cdot \boxed{\left(\frac{p_b}{1-p_b}\right)}$$

$$\boxed{P(w(e)=2) \;=\; P(w(e)=1) \cdot \left(\frac{p_b}{1-p_b}\right) \;<\; P(w(e)=1)}$$

(2.3) probability of w(e)=k bit errors: *w(e)=k*

$$P(w(e)=k) = (1-p_b)^{n-k} \cdot (p_b)^k = \boxed{(1-p_b)^{n-1} \cdot p_b} \cdot \left(\frac{p_b}{1-p_b}\right)^{k-1} = P(w(e)=1) \cdot (a)^{k-1}$$

$$\boxed{P(w(e)=k) \;<\ldots<\; P(w(e)=2) \;<\; P(w(e)=1)}$$

1-bit errors are more likely 2-bit errors, and so forth!

Error Detection: Single Parity Check

(2.4) probability that single parity check fails?!

$$
\begin{aligned}
\mathrm{P}(\textit{error detection failure}) &= \mathrm{P}(\textit{error patterns with even number of 1s}) = \\
&= \mathrm{P}(\textit{any 2 bit error}) + \mathrm{P}(\textit{any 4 bit error}) + \mathrm{P}(\textit{any 6 bit error}) + \ldots = \\
&= \quad (\#\text{of } 2-\text{bit errors}) * \mathrm{P}(w(e)=2) + \\
&\quad + (\#\text{of } 4-\text{bit errors}) * \mathrm{P}(w(e)=4) + \\
&\quad + (\#\text{of } 6-\text{bit errors}) * \mathrm{P}(w(e)=6) + \ldots
\end{aligned}
$$

<u>number of combinations 'n choose k':</u>

$$(\#\text{of k}-\text{bit errors}) = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

1	0	0	$\boxed{1}$	1	0	$\boxed{1}$	0

1	$\boxed{1}$	0	0	1	$\boxed{1}$	0	0

$$\boxed{\mathrm{P}(\textit{error detection failure}) = \binom{n}{2} p_b^2 (1-p_b)^{n-2} + \binom{n}{4} p_b^4 (1-p_b)^{n-4} + \binom{n}{6} p_b^6 (1-p_b)^{n-6} + \ldots}$$

progressively smaller components …

Error Detection: Single Parity Check

Example [probability of error detection failure]

Assume there are n=32 bits in a codeword (packet). Probability of error in a single bit transmission $p_b = 10^{-3}$. Find the probability of error-detection failure.

$$P(\textit{error detection failure}) = \binom{32}{2} p_b^2 (1-p_b)^{30} + \binom{32}{4} p_b^4 (1-p_b)^{28} + \binom{32}{6} p_b^6 (1-p_b)^{26} + \ldots$$

$$\binom{32}{2} p_b^2 (1-p_b)^{30} \approx \frac{32*31}{2}(10^{-3})^2 = 496*10^{-6}$$

$$\binom{32}{4} p_b^4 (1-p_b)^{28} \approx \frac{32*31*30*29}{2*3*4}(10^{-3})^4 = 35960*10^{-12}$$

$$\boxed{P(\text{error detection failure}) = 496*10^{-6} = 4.96*10^{-4} \approx \frac{1}{2000}}$$

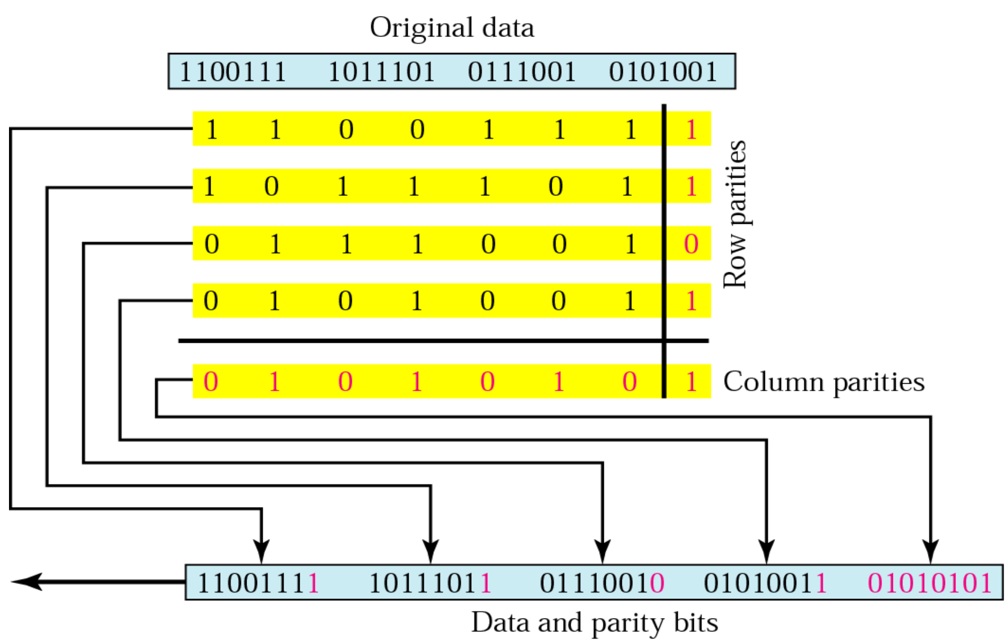
Approximately, 1 in every 2000 transmitted 32-bit long codewords is corrupted with an error pattern that cannot be detected with single-bit parity check.

Error Detection: 2-D Parity Check

Two Dimensional Parity Check

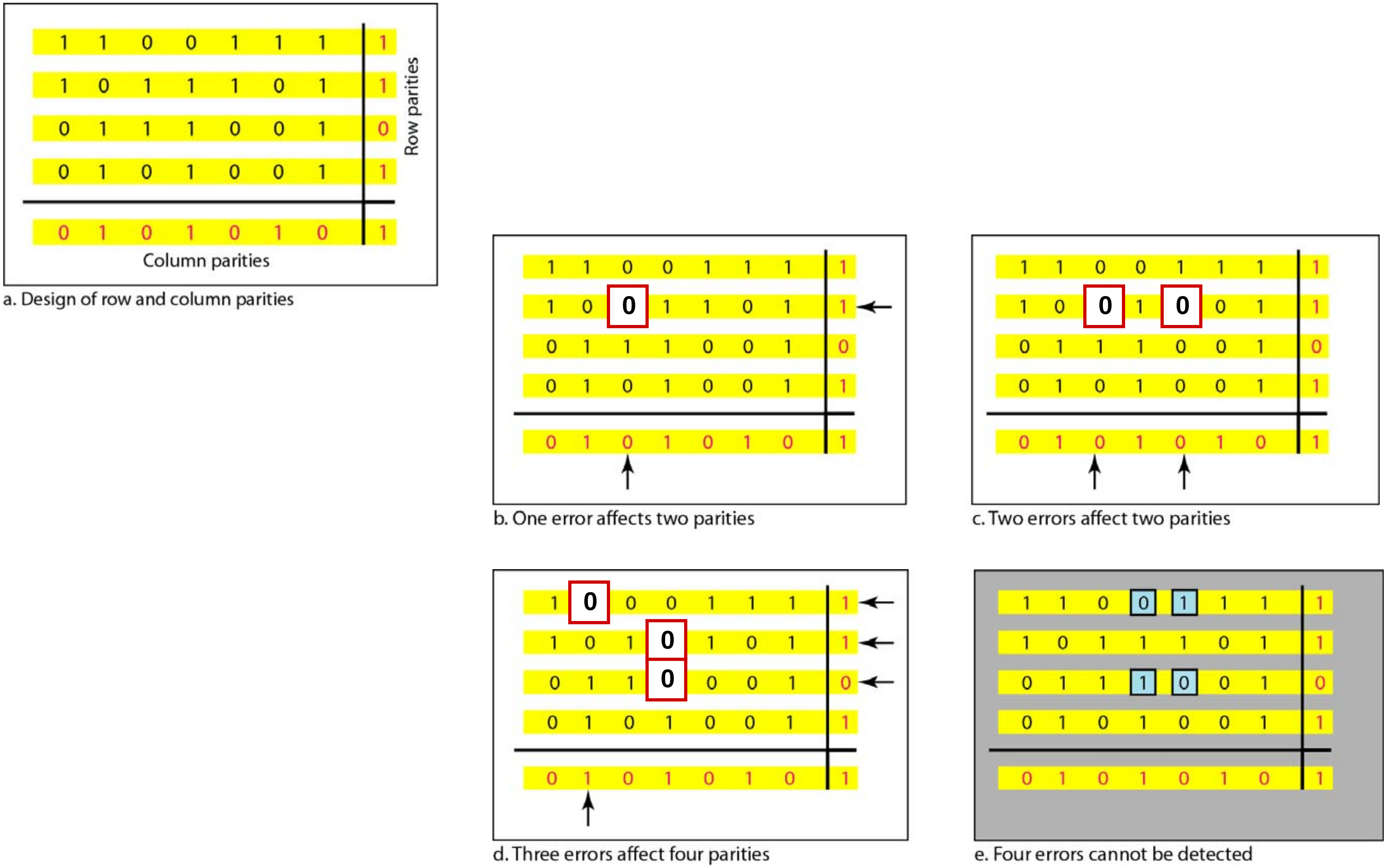
– a block of bits is organized in a table (rows + columns)
a parity bit is calculated for each row and column

- 2-D parity check increases the likelihood of detecting burst errors
  - all 1-bit errors CAN BE DETECTED and CORRECTED
  - all 2-, 3- bit errors can be DETECTED
  - 4- and more bit errors can be detected in some cases
- drawback: too many check bits !!!



Two-Dimensional Parity Check

Example [effectiveness of 2-D parity check]



a. Design of row and column parities

b. One error affects two parities

c. Two errors affect two parities

d. Three errors affect four parities

e. Four errors cannot be detected