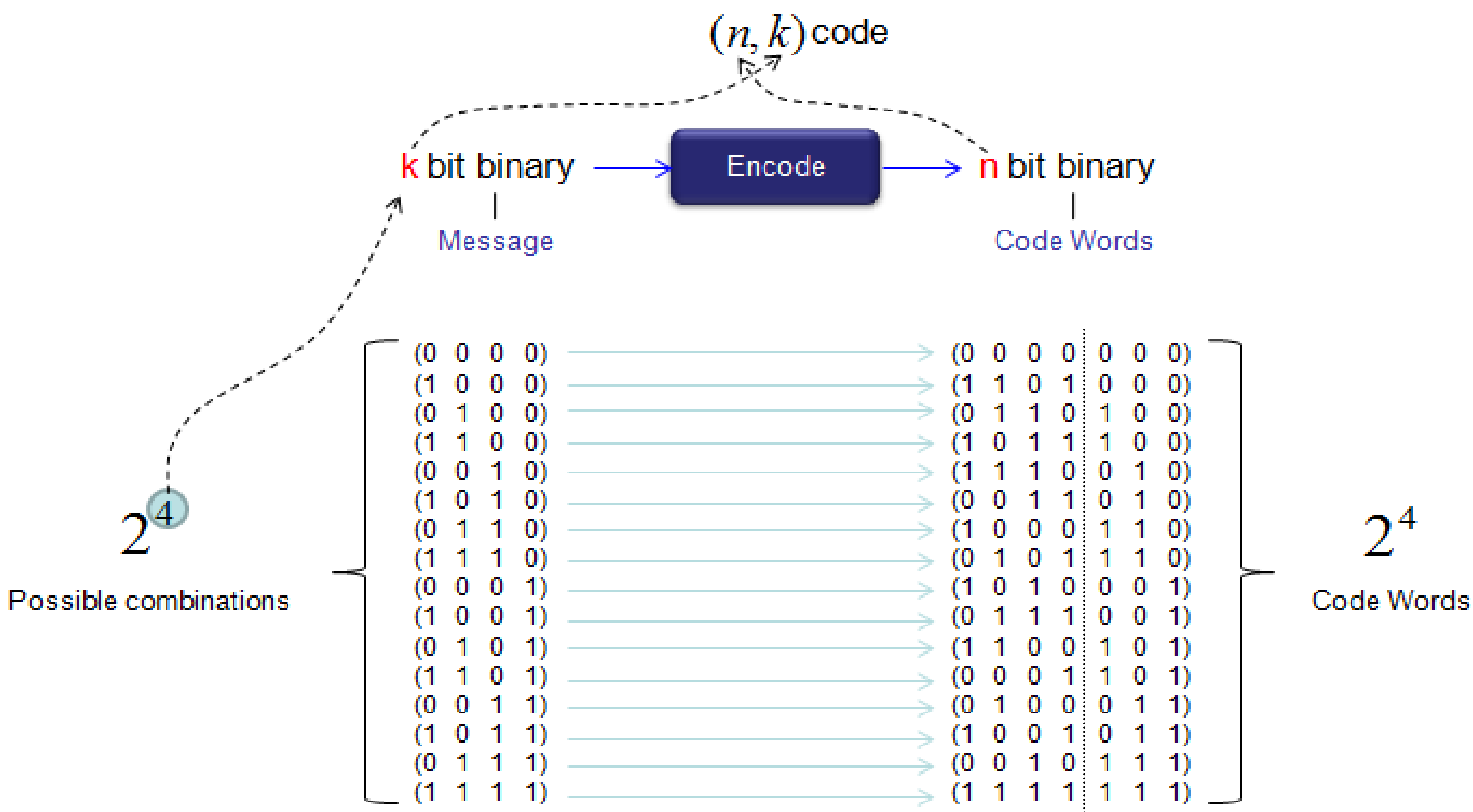
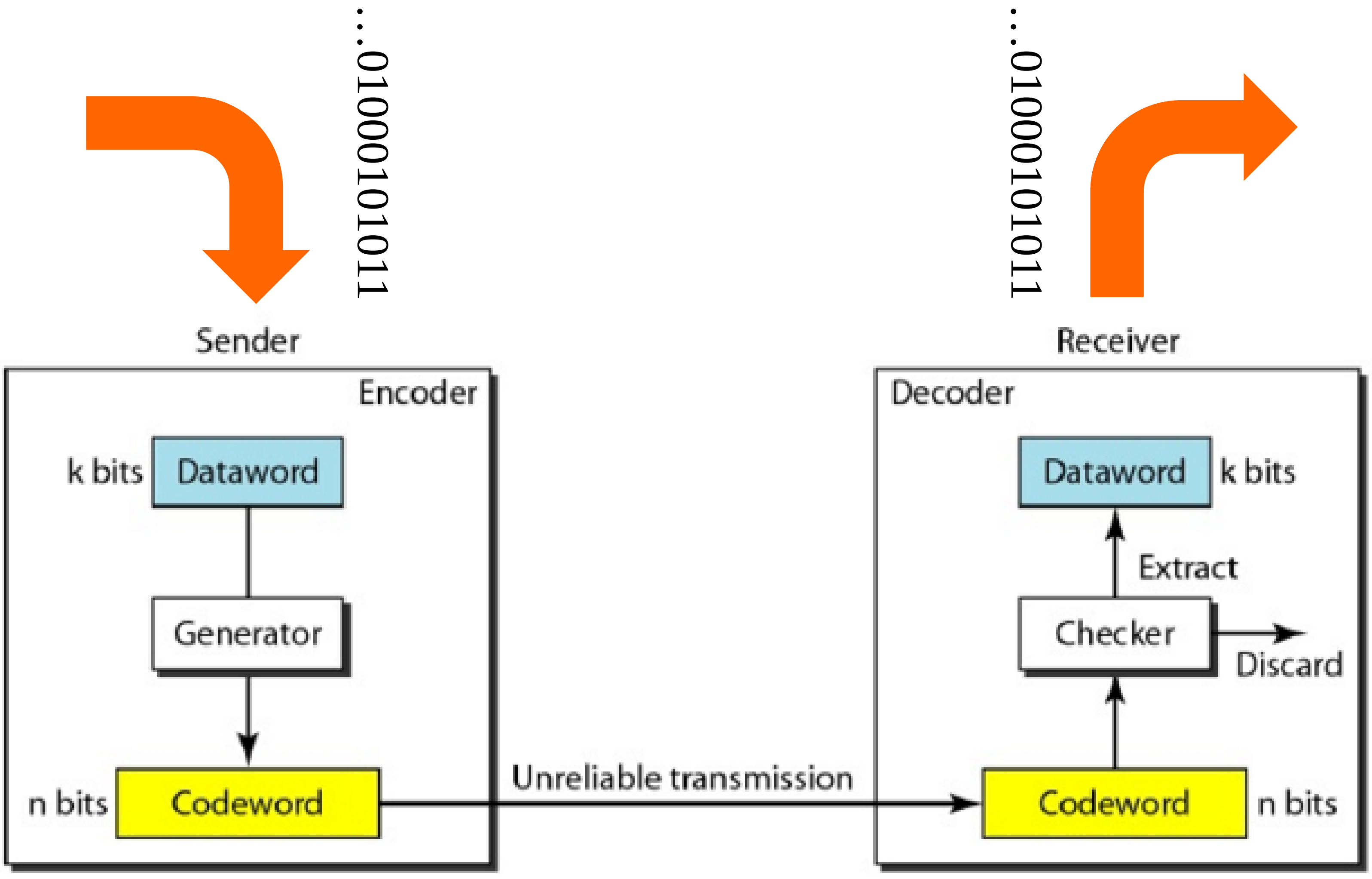
Block Coding

Block Coding – **encoding method in which the whole input data stream is split into small blocks (datawords) and replaced with another somewhat larger blocks (codewords)**

Example **[input stream broken into datawords of size=4 & replaced with codewords of size=7]**



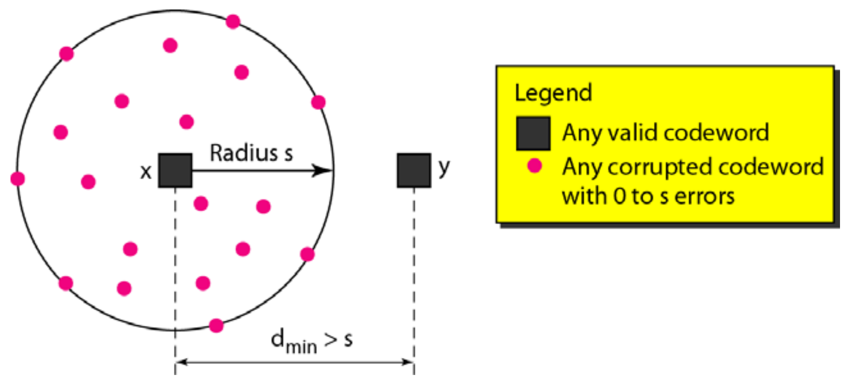
Example **[error detection using block coding]**



Hamming Distance (cont.)

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

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



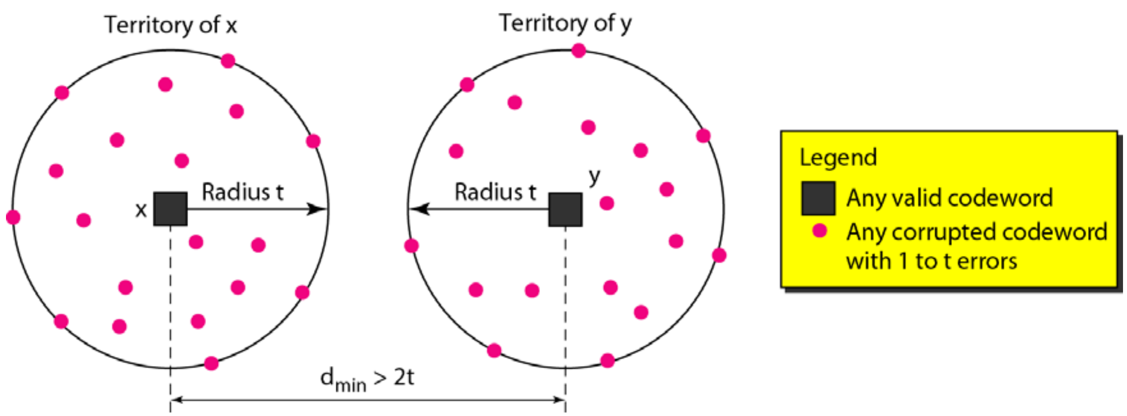
Example [code with d_{min}=2 is able to detect s=1 bit-errors]

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

Hamming Distance (cont.)

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

$$d_{min} = 2t + 1 \quad \Rightarrow \quad t = \left\lfloor \frac{d_{min-1}}{2} \right\rfloor$$



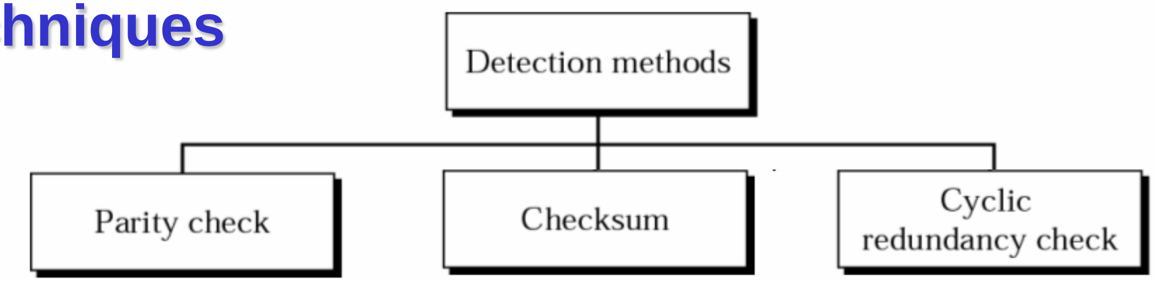
Example **[Hamming distance]**

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!

Error Detection: Single Parity Check

Error Detection Techniques



Single Parity Check (Even Parity)

– take k information bits and append a single check bit so that overall number of 1s is even !

Modulo 2 sum (i.e. XOR) of information bits!

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

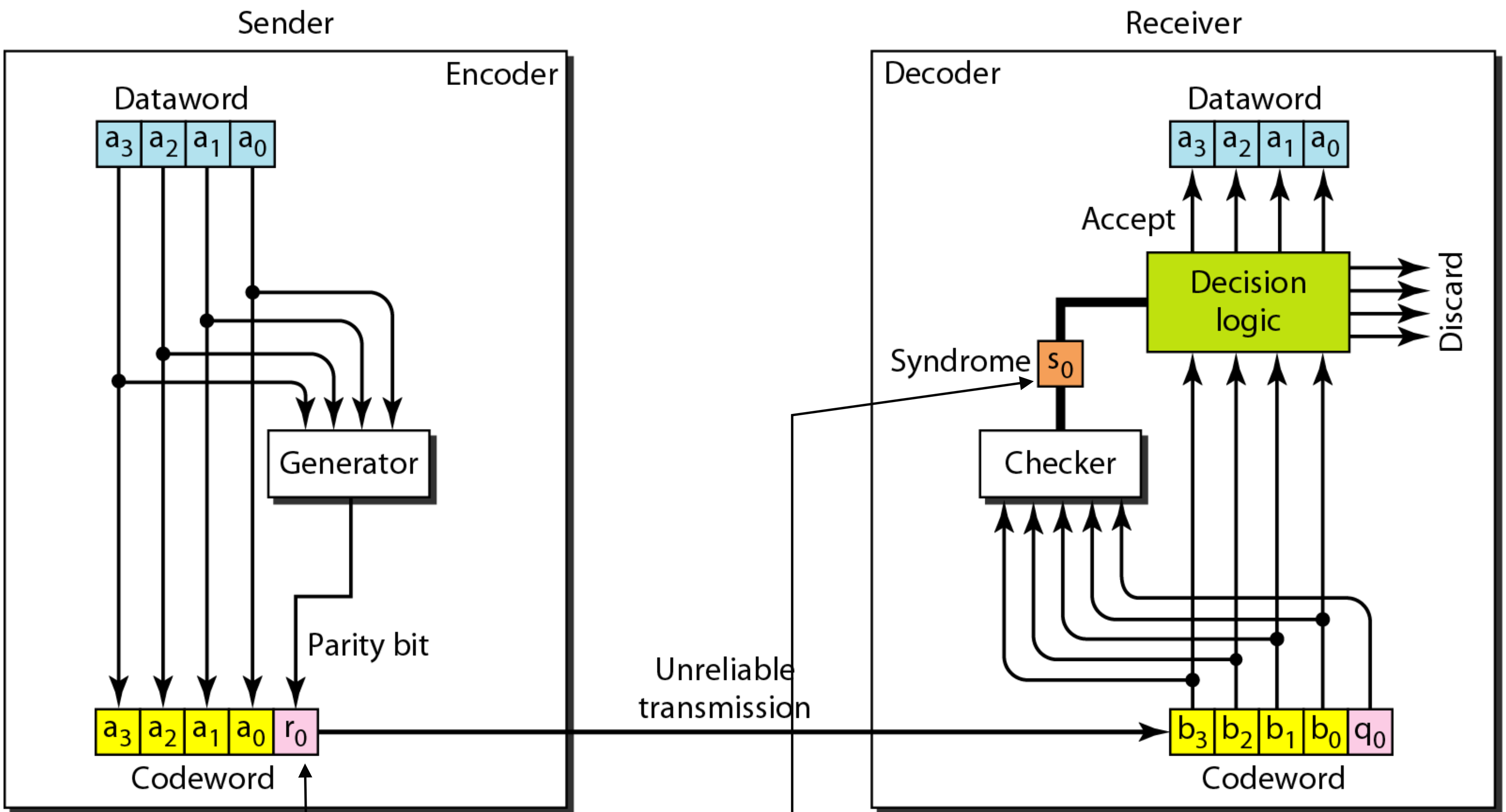
Check Bit: $b_{k+1} = b_1 \oplus b_2 \oplus b_3 \oplus \ldots \oplus b_k$

Codeword: $[b_1, b_2, b_3, \ldots, b_k, b_{k+1}]$

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

Error Detection: Single Parity Check (cont.)

Example **[encoder and decoder for single parity check code]**



ensures that the overall number of 1-s is even

s_0 = 0, if number of 1-s even, no error detected
s_0 = 1, if number of 1-s odd, error detected

Error Detection: Single Parity Check (cont.)

Example **[single parity check]**

- **Information (7 bits):** [0, 1, 0, 1, 1, 0, 0]
- **Parity Bit:** $b_8 = 0 + 1 + 0 + 1 + 1 + 0 \bmod 2 = 1$
- **Codeword (8 bits):** [0, 1, 0, 1, 1, 0, 0, 1]

- **If single error in bit 3 :** [0, 1, $\boxed{1}$, 1, 1, 0, 0, 1]
  - **# of 1's = 5, odd**
  - **Error detected ☺ !**

- **If errors in bits 3 and 5:** [0, 1, $\boxed{1}$, 1, $\boxed{0}$, 0, 0, 1]
  - **# of 1's = 4, even**
  - **Error not detected ☹ !!!**

- **If errors in bit 3, 5, 6 :** [0, 1, $\boxed{1}$, 1, $\boxed{0}$, $\boxed{1}$, 0, 1]
  - **# of 1's = 5, odd**
  - **Error detected ☺ !**

Error Detection: Single Parity Check (cont.)

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, $d_{min} = 2$**

Effectiveness of Single Parity Check

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

received codeword: $b' = [b'_1\ b'_2\ b'_3\ \ldots b'_n]$

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

1	0	0	1	1	0	0	1
1	1	0	1	1	1	0	1
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 (e) 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 (cont.)

(2) Random Bit Error Channel Model – **bit errors occur independently of each other** – **p_b = prob. 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 <u>particular</u>) position**

1	0	0	1	1	0	0	0

$$\xrightarrow{p_b}$$

1	0	**1**	1	1	0	0	0

$$P(w(e)=1)=\underbrace{\overset{e_1=0}{(1-p_b)}}_{\text{probability of correctly transmitted bit}}\cdot\overset{e_2=0}{(1-p_b)}\cdot\overset{e_3=1}{p_b}\cdot\ldots\cdot\overset{e_{n-2}=0}{(1-p_b)}\cdot\overset{e_{n-1}=0}{(1-p_b)}\cdot\overset{e_n=0}{(1-p_b)}$$

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

Error Detection: Single Parity Check (cont.)

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

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

<1, since $p_b<0.5$

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

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

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

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

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

Error Detection: Single Parity Check (cont.)

(2.4) probability that single parity check fails?!

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

number of combinations ‘n choose k’:

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

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

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

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

progressively smaller components …

Error Detection: Single Parity Check (cont.)

Example [probability of single parity check 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.

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

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

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

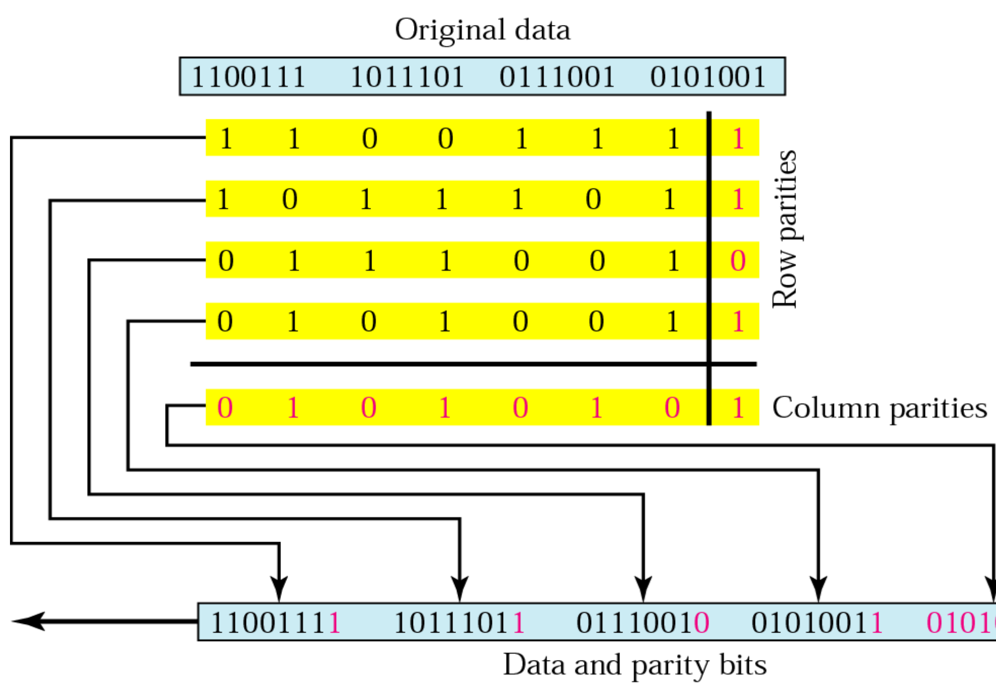
$$\boxed{P(\text{error detection failure}) = 496 * 10^{-6} = 4.96 * 10^{-4} \approx \frac{1}{2000}}$$

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

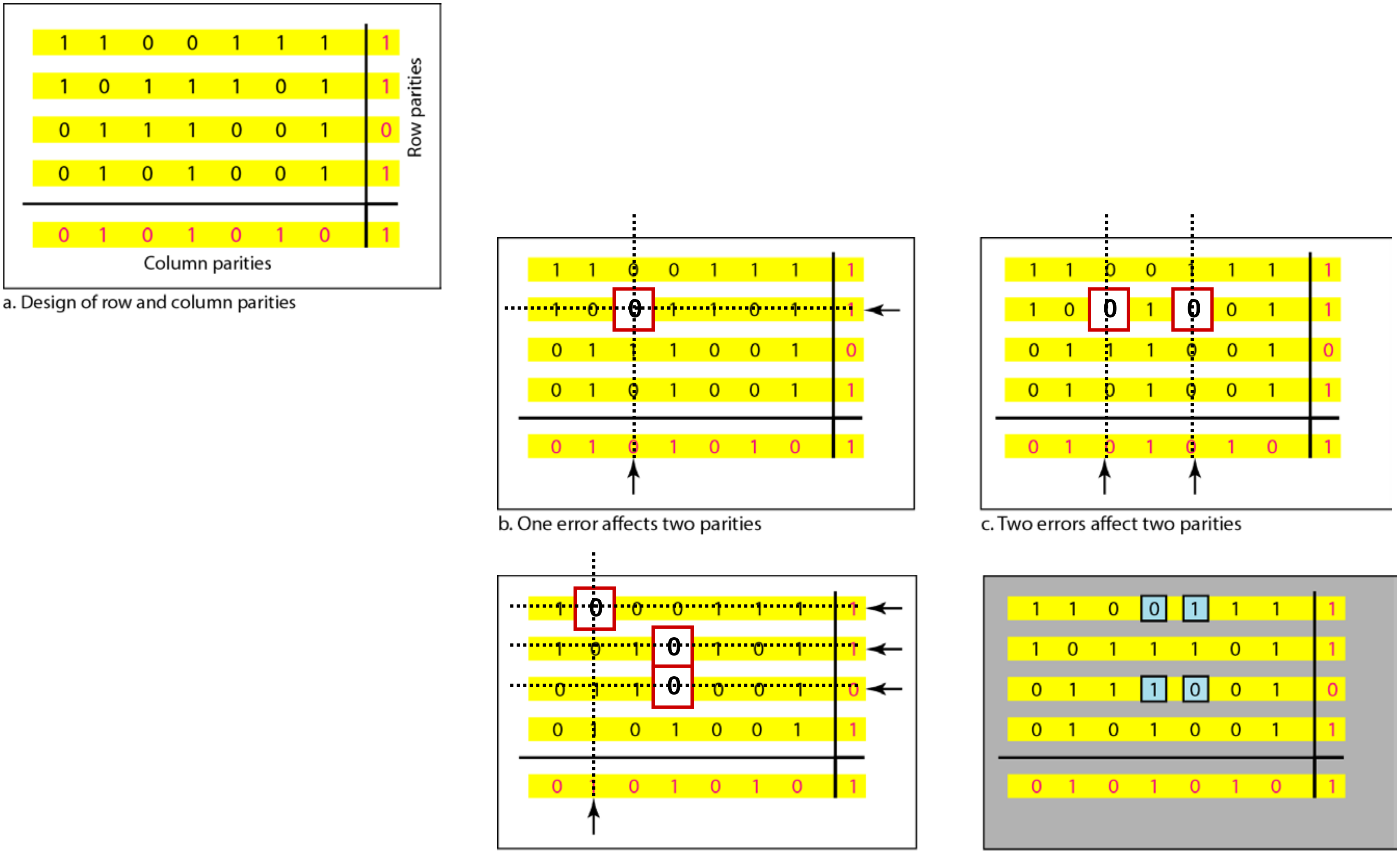
Two Dimensional Parity Check – a block of bits is organized in a table (rows & columns) a parity bit is calculated for each row and column

- 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 !!!



Error Detection: 2-D Parity Check (cont.)

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

Error Detection: 2-D Parity Check (cont.)

Example **[2-D parity check]**

Suppose the following block of data, error-protected with 2-D parity check, is sent:
10101001 00111001 11011101 11100111 10101010.

However, the block is hit by a burst noise of length 8, and some bits are corrupted.
10100011 10001001 11011101 11100111 10101010.

Will the receiver be able to detect the burst error in the sent data?

1010100 1	**1010001 1**
0011100 1	**1000100 1**
1101110 1	**1101110 1**
1110011 1	**1110011 1**
1010101 0	**1010101 0**

Signed Number Representation

http://en.wikipedia.org/wiki/Signed_number_representations

8 bit signed magnitude

Binary	Signed	Unsigned
00000000	+0	0
00000001	1	1
...	...	...
01111111	127	127
10000000	−0	128
10000001	−1	129
...	...	...
11111111	−127	255

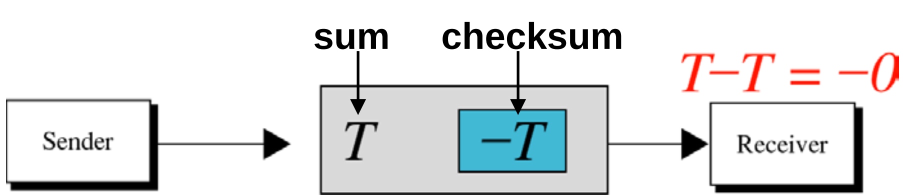
8 bit ones' complement

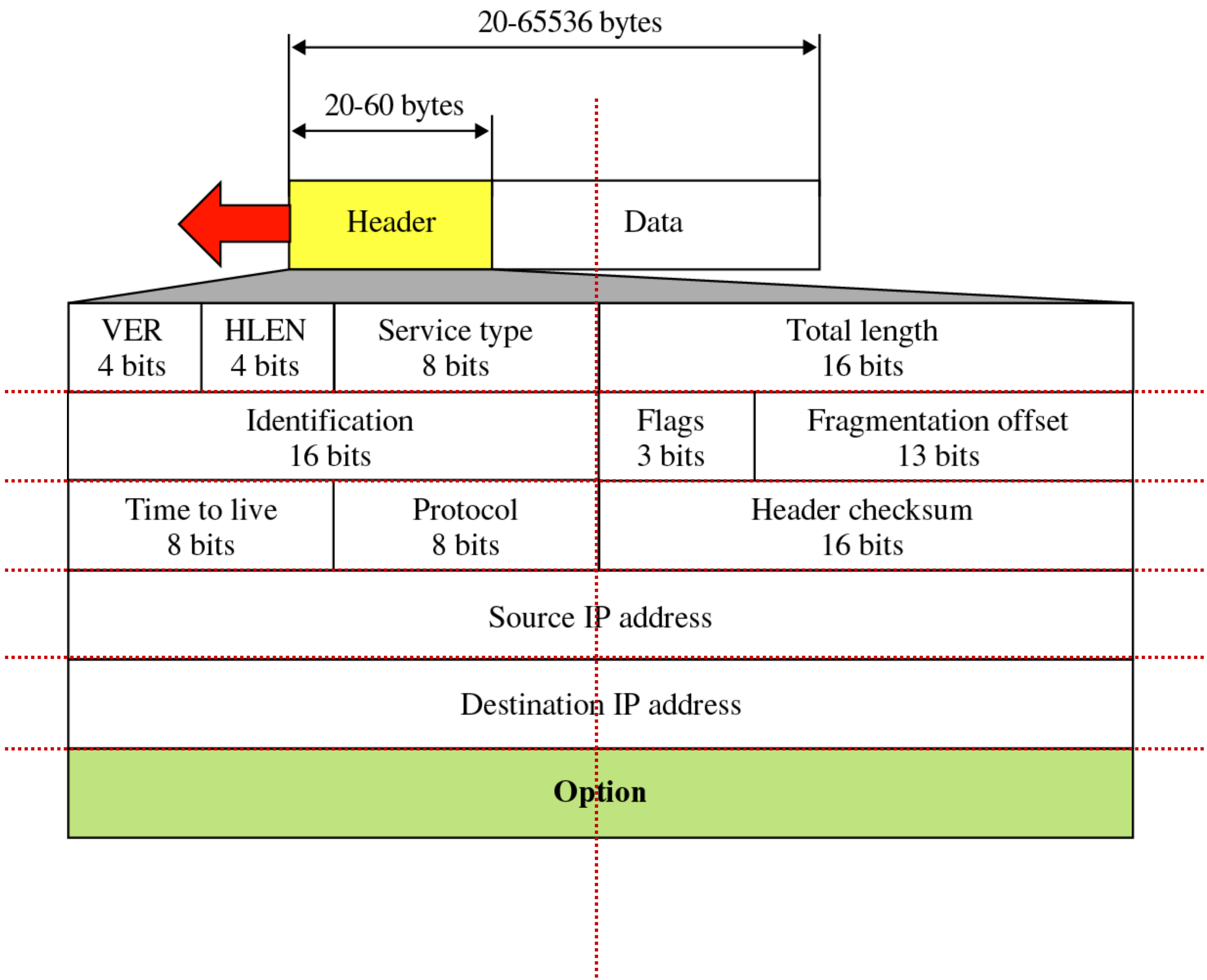
Binary value	Ones' complement interpretation	Unsigned interpretation
00000000	+0	0
00000001	1	1
...	...	...
01111101	125	125
01111110	126	126
01111111	127	127
10000000	−127	128
10000001	−126	129
10000010	−125	130
...	...	...
11111110	−1	254
11111111	−0	255

Error Detection: Internet Checksum

(Internet) Checksum – error detection method used by IP, TCP, UDP !!!

- checksum calculation:
  - IP/TCP/UDP packet is divided into n-bit sections
  - n-bit sections are added using "1-s complement arithmetic" – the sum is also n-bits long!
  - the sum is complemented to produce checksum (complement of a number in 1-s arithmetic is the negative of the number)
- advantages:
  - relatively small packet overhead is required – n bits added regardless of packet size
  - easy / fast to implement in software
- disadvantages:
  - weak protection compared to CRC – e.g. will NOT detect misordered bytes/words !!!
  - detects all errors involving an odd number of bits and most errors involving an even number of bits




20-65536 bytes
20-60 bytes
Header
Data
VER
4 bits
HLEN
4 bits
Service type
8 bits
Total length
16 bits
Identification
16 bits
Flags
3 bits
Fragmentation offset
13 bits
Time to live
8 bits
Protocol
8 bits
Header checksum
16 bits
Source IP address
Destination IP address
Option

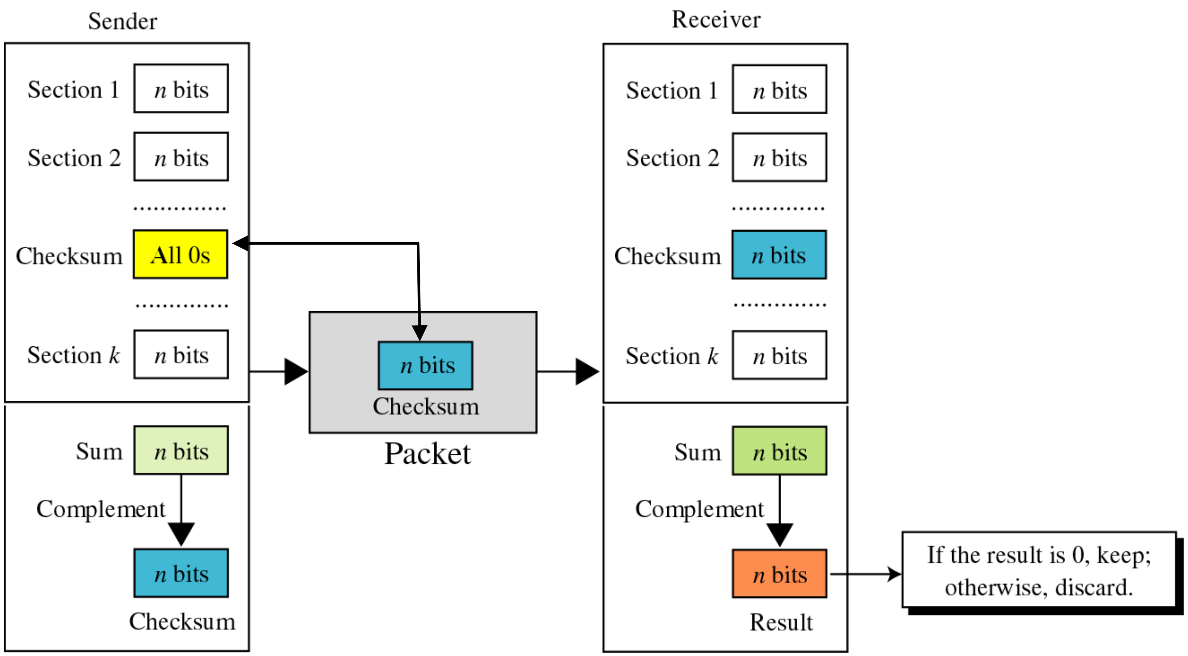
Error Detection: Internet Checksum (cont.)

Sender:

- **data is divided into k sections each n bits long**
- **all sections are added using 1-s complement to get the sum**
- **the sum is bit-wise complemented and becomes the checksum**
- **the checksum is sent with the data**

Receiver:

- **data is divided into k sections each n bits long**
- **all sections are added using 1-s complement to get the sum**
- **the sum is bit-wise complemented**
- **if the result is zero, the data is accepted, otherwise it is rejected**



Error Detection: Internet Checksum (cont.)

Example [Internet Checksum]

Suppose the following block of 8 bits is to be sent using a checksum of 4 bits: 1100 1010. Find the checksum of the given bit sequence.

```
                              1100
                              1010
                              0000
                         ----------
                 sum:        10110

                              0110
                                 1
                         ----------
1-s complement addition:      0111   (7)

            checksum:         1000   (-7)
```

1-s complement addition: Perform standard binary addition. If a carry-out (>nth) bit it produced, swing that bits around and add it back into the summation.

Negative binary numbers: Negative binary numbers are bit-wise complement of corresponding positive numbers.

Error Detection: Internet Checksum (cont.)

Suppose the receiver receives the bit sequence and the checksum with no error.

	1100
	1010
	1000
sum:	**$\boxed{1}1110$**
1-s complement addition:	**1111**
bit-wise complement:	**0000**

When the receiver adds the three blocks, it will get all 1s, which, after complementing, is all 0s and shows that there is no error.

> **If one or more bits of a segment are damaged, and the corresponding bit of opposite value in a second segment is also damaged, the sums of those columns will not change and the receiver will not detect the problem. ☹**

Error Detection: Internet Checksum (cont.)

Example [Internet Checksum]

Suppose the following block of 16 bits is to be sent using a checksum of 8 bits. 10101001 00111001. The numbers are added using one's complement:

	10101001
	00111001
	00000000

Sum	11100010
Checksum	00011101

The pattern sent is 10101001 00111001 00011101.

Now suppose the receiver receives the pattern with no error.
10101001 00111001 00011101
When the receiver adds the three blocks, it will get all 1s, which, after complementing, is all 0s and shows that there is no error.

	10101001
	00111001
	00011101
Sum	11111111
Complement	00000000 means that the pattern is OK.

Error Detection: Internet Checksum (cont.)

Example [Internet Checksum]

Now suppose that in the previous example, there was a burst error of length 5 that affected 4 bits.

10101111 11111001 00011101

When the receiver added the three sections, it got

	10101111	
	11111001	
	00011101	
Partial Sum	**1 11000101**	
Checksum	**11000110**	
Complement	**00111001**	**the pattern is corrupted.**