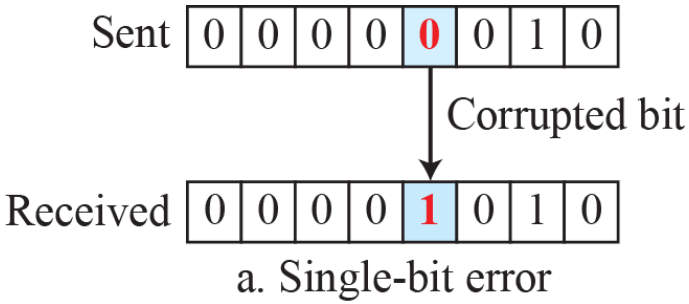
Chapter 10
오류 검출과 오류 정정

차례

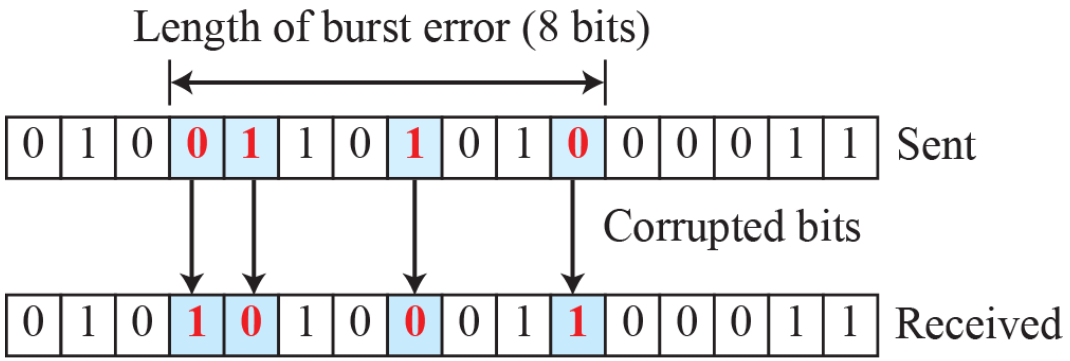
§1. 소개

- Single-Bit Error(단일-비트 오류) : 한 단위의 데이터 중 오직 한 비트만 변경되는 오류



a. Single-bit error

- Burst Error (폭주 오류) : 한 단위의 데이터 중 2개 이상의 연속적인 비트가 변경되는 오류



b. Burst error

- 중복(Redundancy)
  - ✓ 목적지에서 오류를 검출, 정정하기 위해서 데이터 워드에 여분의 비트(또다른 정보)를 추가하는 것
  - ✓ 오류검출은 약간의 중복 필요하고, 오류정정은 다수의 중복 필요함
- 오류 검출
  - ✓ 오류가 발생했는지 발생 유무만 확인
  - ✓ 재전송 등의 방법 필요
- 오류 정정
  - ✓ 몇 비트가 잘못 되었는지, 어디가 잘못 되었는지를 확인
    - 오류 검출을 포함하는 개념
  - ✓ 재전송 없이 잘못 전송된 비트 정정 가능

블록 코딩(blocking coding)

- **코딩**– 송신자는 중복 비트와 실제 데이터 비트들 사이에 어떤 관련을 짓게 하는 과정(즉,코딩)을 통해 중복비트들을 보냄
- **블록 코딩**– 에러를 검출하기위해 원본 데이터에 여분의 비트를 넣어서 전송하는 코딩 방법 중 하나
  - ✓ 메시지를 k비트의 블록으로 나눔 ➔ 데이터워드(dataword)라 부름
    - k비트를 사용하여 2^k개의 데이터워드를 만들수 있음
  - ✓ 각 블록에 r개의 중복 비트를 추가하여 길이 n(=k+r)개의 블록으로 만듬 ➔ 코드워드(codeword)라 부름
    - n비트를 사용하여 2^n개의 코드워드를 만들수 있음
    - n>k이므로 가능한 코드워드 개수가 더 많음
  - ✓ 즉, 2^n- 2^k개의 코드워드는 사용되지 않음
    → 이런 무효 코드를 수신하면 오류발생을 알 수 있음

■ 블록 코딩에서 오류 검출 방법

다음 두 조건이 맞으면 수신자는 원래 코드워드가 바뀐걸 확인

1. 유효 코드워드들을 찾아내거나 그 리스트를 가지고 있다
2. 원래의 코드워드가 **무효 코드워드로 바뀌었다**

■ 오류 검출 코드는 찾도록 설계된 오류 만을 찾아낸다.
다른 오류는 검출되지 못한다.

예제

- k=2이고 n=3일때, 데이터워드와 코드워드는 다음 표와 같다. (어떻게 코드워드를 만들어 내는지 여기서는 논의 안함)

- 오류 검색을 위한 코드

dataword	codeword
00	**000**
01	**011**
10	**101**
11	**110**

✓ 송신자가 데이터워드 01을 011로 코딩하여 수신자에게 보낼때,

- 수신자가 011을 수신 → 유효코드임. 수신자는 이로부터 데이터워드 01을 추출함
- 코드워드가 전송중 손상되어 111 수신(맨 왼쪽 비트가 깨졌음). → 유효 코드가 아님
- 코드워드가 전송 도중 손상되어 000 수신(오른쪽 두 비트가 깨졌음) → 유효 코드임
  → 수신자는 이로부터 부정확한 데이터워드인 00을 추출함

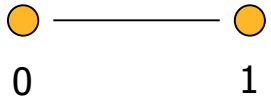
  → 두 개 비트가 손상되어 오류를 찾지 못함
(한 개의 비트 오류만 찾아낼 수 있다)

§2. n-cube의 기본 개념 및 활용

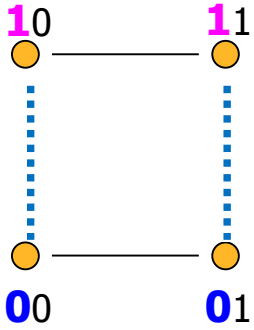
n-cube의 기본 개념

- ***n*** bit string을 ***n***−cube(***n***차원의 cube)의 꼭지점에 기하학적으로 가시화함
  - ✓ n-cube는 2^n개의 꼭지점 가지며, 각각 n bit string으로 표시
- 각 꼭지점에는 모서리로 연결된 ***n***개의 인접한 꼭지점이 존재함
  - ✓ 꼭지점과 그 꼭지점에 인접한 각 꼭지점들의 string들은 단지 한 bit만 다름
- ***n*** 차원의 ***n***−cube는 ***n***−1차원의 cube 2개로써 서로 대응되는 꼭지점끼리 연결하여 만듬

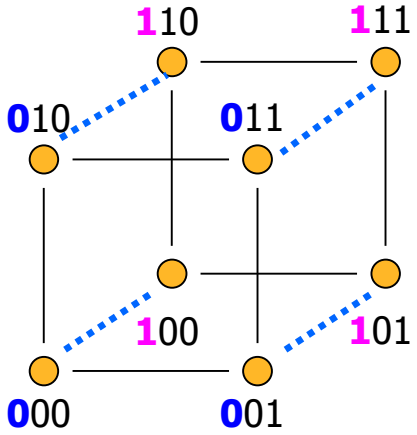
n=1, 2, 3, 4에 대한 n-cube



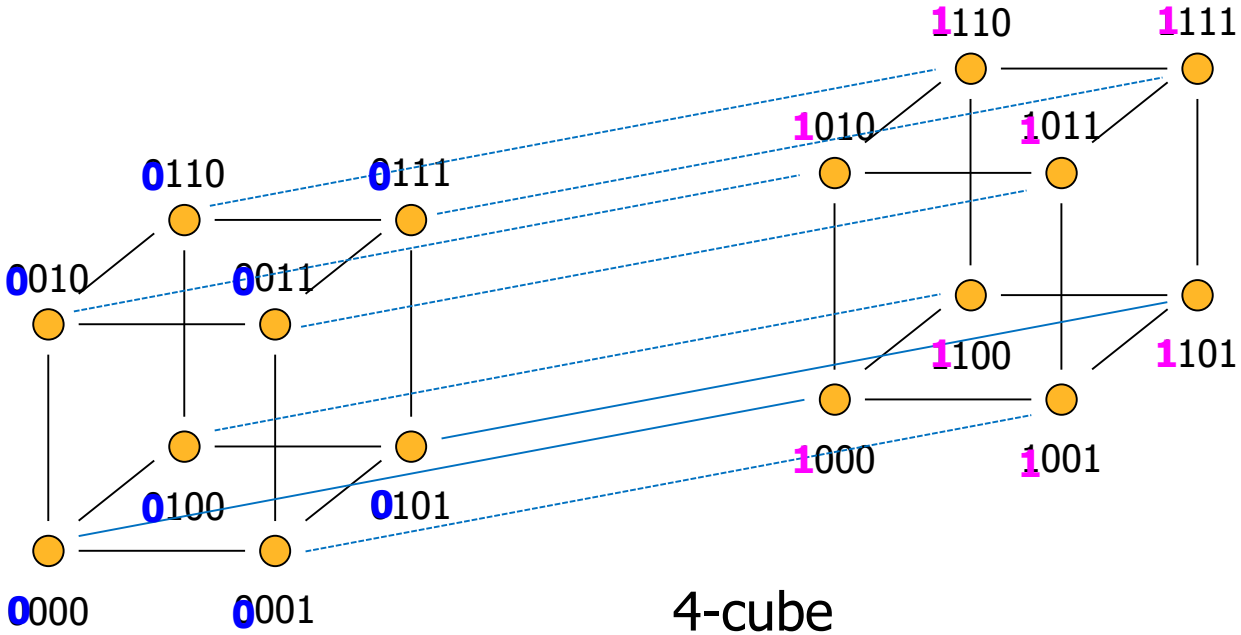
1-cube



2-cube



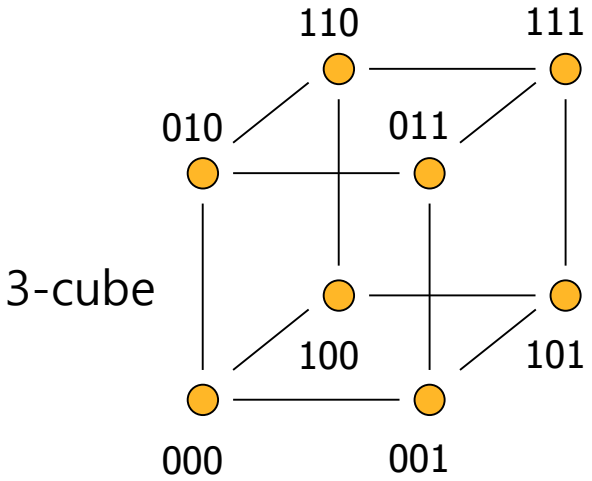
3-cube



4-cube

n-cube의 활용

- Hamming distance라고 불리는 거리의 개념에 대한 기하학적인 해석을 제공
  - ✓ 2개의 n bit string 사이의 Hamming distance(거리)는
    - 서로 다른 bit 자리의 개수
    - 두 string에 XOR 연산하여 얻은 결과에서 1의 개수
    - 2개의 대응하는 꼭지점들 사이에서의 최소 길이



예) 3 bit string인 010과 111의 거리는 2

$$\begin{array}{r} 010 \\ \oplus\ 111 \\ \hline 101 \end{array}$$

예) 3 bit string인 010과 101의 거리는 3

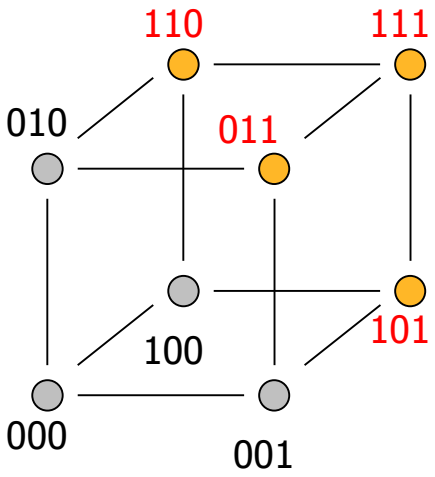
$$\begin{array}{r} 010 \\ \oplus\ 101 \\ \hline 111 \end{array}$$

  - ✓ 오류 검출 코드의 설계와 이해를 위해 중요

코드워드의 예

3비트 코드워드, 최소거리=1

데이터 워드	코드 워드
00	110
01	011
10	101
11	111



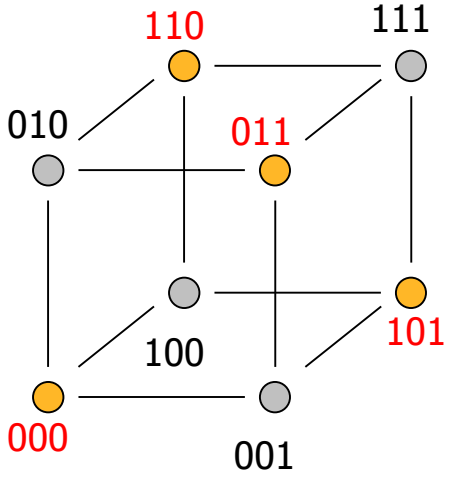
단일 1비트 오류일지라도 검출 못함

code word(유효코드)

non code word(무효코드)

3비트 코드워드, 최소거리=2

데이터 워드	코드 워드
00	000
01	011
10	101
11	110

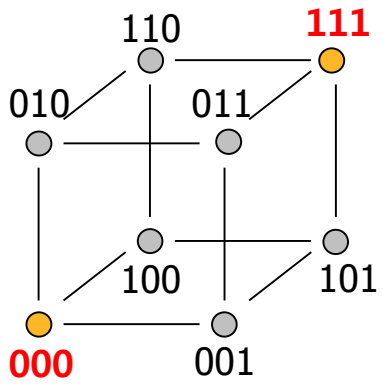


단일 1비트 오류의 경우 검출 가능

코드워드의 예

3비트 코드워드, 최소거리=3

데이터 워드	코드 워드
0	000
1	111



code word(유효코드)

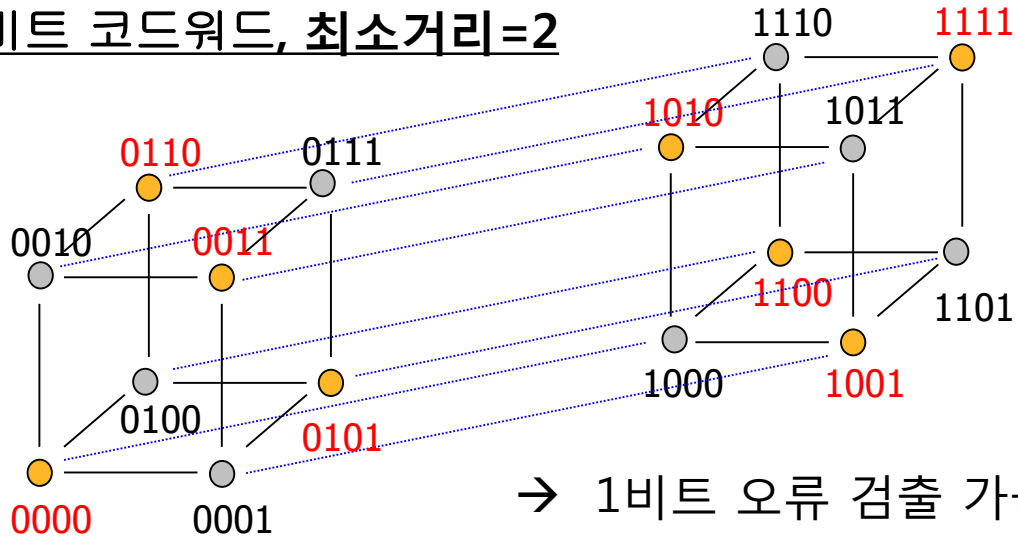
non code word(무효코드)

→ 2비트 오류까지 검출 가능

최소거리가 2인 코드워드

데이터워드	코드워드
000	0000
001	0011
010	0101
011	0110
100	1001
101	1010
110	1100
111	1111

4비트 코드워드, 최소거리=2

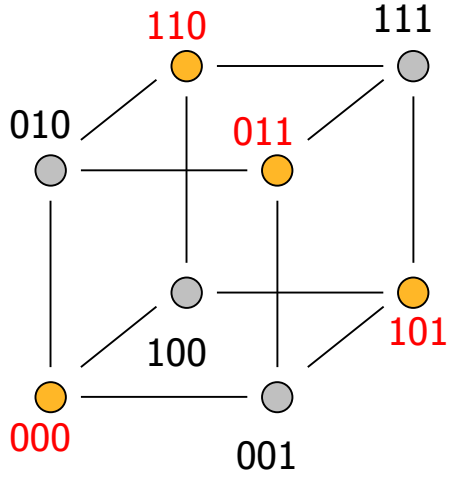


→ 1비트 오류 검출 가능

최소거리가 2인 코드워드

최소거리=2

데이터 워드	코드 워드
00	000
01	011
10	101
11	110



code word
non code word

단일(1비트) 오류 검출 가능!

- ✓ 모든 쌍의 code사이의 최소거리가 2라면, 그 코드에서 모든 **단일 오류를 검출할** 수 있다. 즉, 2^n개의 데이터워드를 단일오류 검출 코드로 만들기 위해서는 $2x2^n$개의 (즉, n+1 bits의) 코드 워드가 필요하다.
- ✓ 최소거리가 a이라면 a-1비트까지의 오류 검출 가능함

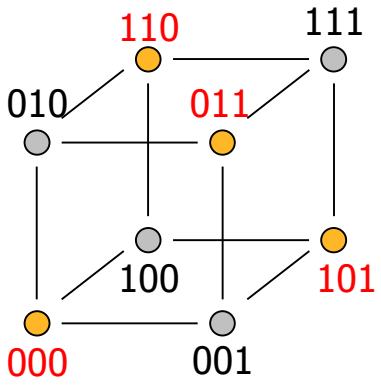
오류 검출/정정 코드

■ 코드워드의 최소거리가 a일때 a=2c+d+1 라고 두자.

- ✓ c+d bit 까지 오류 검출이 가능함
- ✓ c bit 이하의 오류 정정(검출&정정)이 가능함
  (d=0인 경우, c는 최대가 됨)

예1) 최소거리가 2인 코드워드: 2 =2c+d+1= 2×0+1+1
c=0, d=1이므로, c+d=1 bit 오류검출 가능
그러나, c=0 이므로, 오류 정정은 불가

3비트, 최소거리=2



➔ 1비트 오류 검출 가능
(오류 정정 불가)

오류 검출/정정 코드

예2) 최소거리가 3인 코드워드:

$2c+d+1$

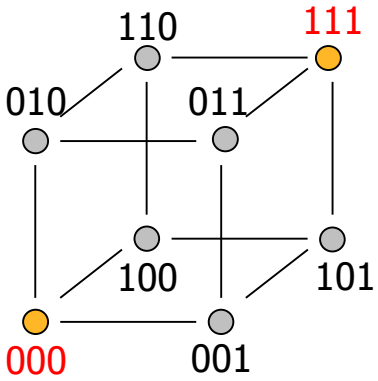
경우①: $3 = 2\times c + d + 1 = 2\times 0 + 2 + 1$

즉, c=0, d=2이므로 → c+d= **2 bit 까지 오류검출 가능** (정정불가)

경우②: $3 = 2\times c + d + 1 = 2\times 1 + 0 + 1$

즉, c=1, d=0이므로 → **1 bit 오류 검출&정정 가능**

3비트, 최소거리=3



오류 검출/정정 코드

예3) 최소거리가 4인 코드워드:

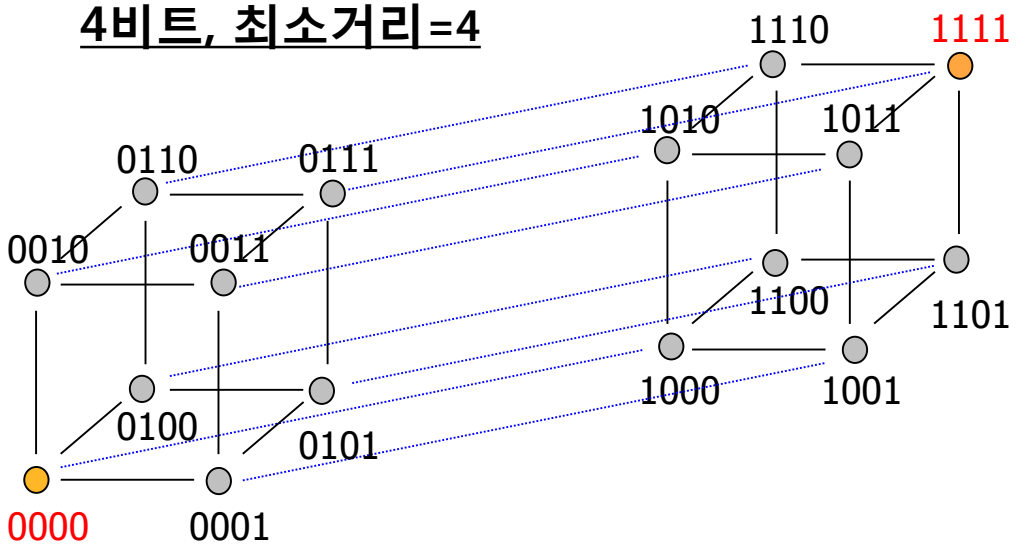
$2c+d+1$

경우①: $4 = 2\times c + d + 1 = 2\times 0 + 3 + 1$

즉, c=0, d=3이므로 → c+d= **3 bit 까지 오류검출 가능** (정정불가)

경우②: $4 = 2\times c + d + 1 = 2\times 1 + 1 + 1$

즉, c=1, d=1이므로 → **2 bit까지 오류검출, 1 bit 오류 정정 가능**



오류 검출/정정 코드

예4) 최소거리가 6인 코드워드:

$2c+d+1$

경우① : $6= 2\times c+ d+ 1 = 2\times 0+ 5 +1$

즉, c=0, d=5이므로 → **5 bit까지 오류검출** 가능, **오류정정 불가**

경우②: $6= 2\times c+ d+ 1 = 2\times 1+ 3 +1$

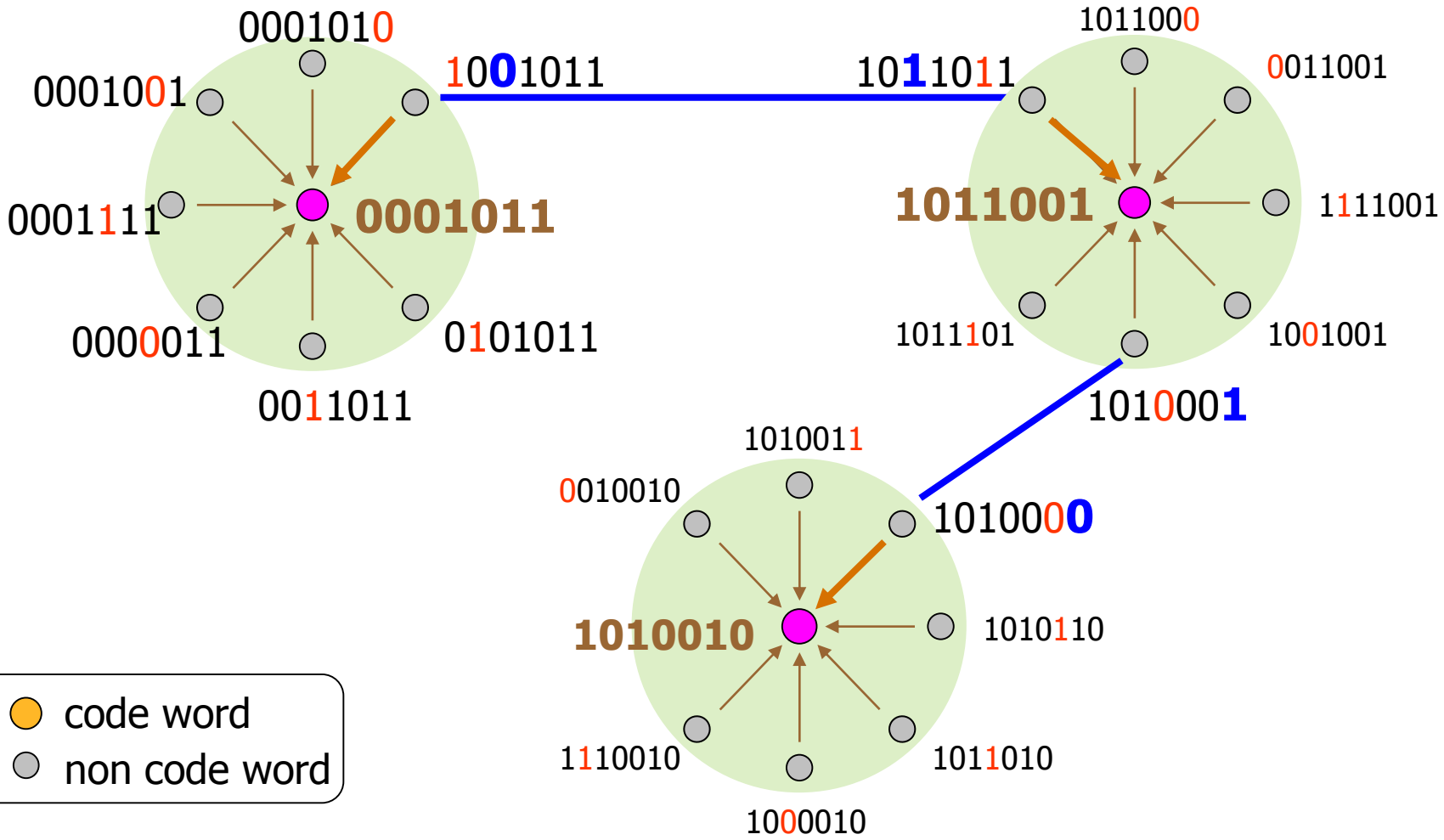
즉, c=1, d=3이므로 → **4 bit까지 오류검출, 1bit 이하 오류정정**

경우③: $6= 2\times c+ d+ 1= 2\times 2+ 1 +1$

즉, c=2, d=1이므로 → c+d=**3 bit까지 오류검출, 2bit 이하 오류정정**

7비트, 거리 3코드 예

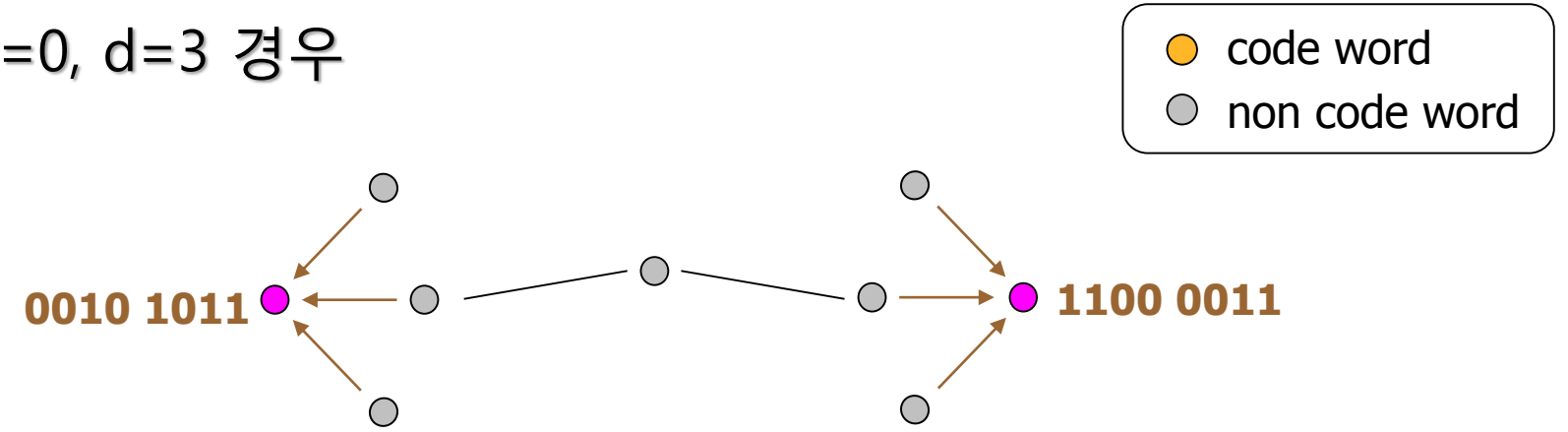
- **거리 3** : 3 = 2c+d+1
  1) c=0, d=2 → 2bit 오류 검출, 오류 정정 불가
  2) c=1, d=0 → 1bit 오류 검출, 1bit 오류 정정



8비트, 거리 4 코드 예

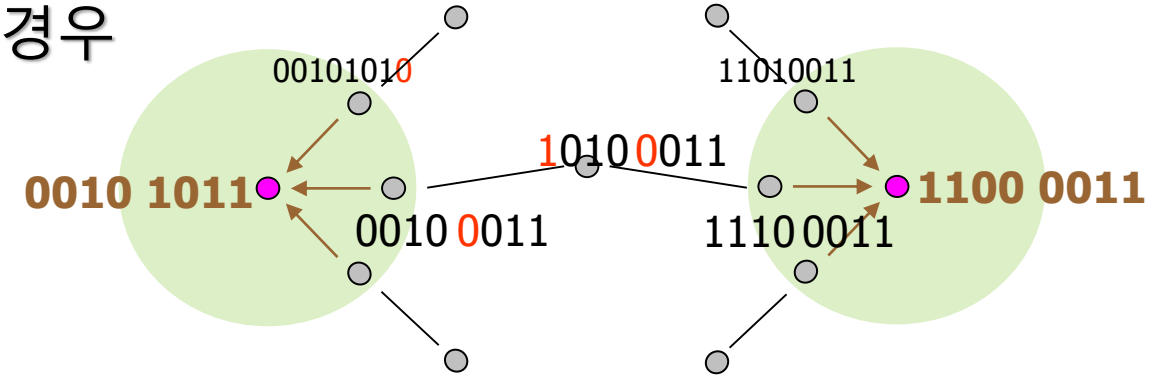
- **거리 4** : 4 = 2c+d+1
  1) c=0, d=3 → 3bit 오류 검출, 오류 정정 불가
  2) c=1, d=1 → 2bit 오류 검출, 1bit 오류 정정

1) c=0, d=3 경우

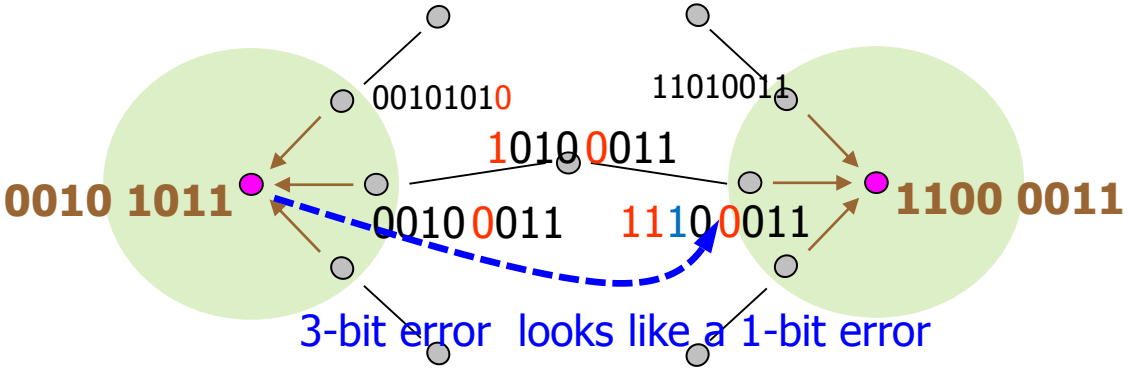


→ 3bit까지 오류 검출 가능, 오류 정정 불가

2) c=1, d=1 경우



→ 2bit까지 오류 검출, 1bit 오류 정정



→ 2 bit까지의 오류 검출하지만, 3bit 오류는 검출 불가

3 bit 오류인 경우는 1 bit 오류로 간주하여 "잘못된" 정정함

오류 검출 vs. 오류 정정

■ 오류 검출

- ✓ 오류가 발생했는지 발생 유무만 확인
- ✓ 오류가 몇 개인지 알 필요 없음
- ✓ 재전송 등의 방법 필요

■ 오류 정정

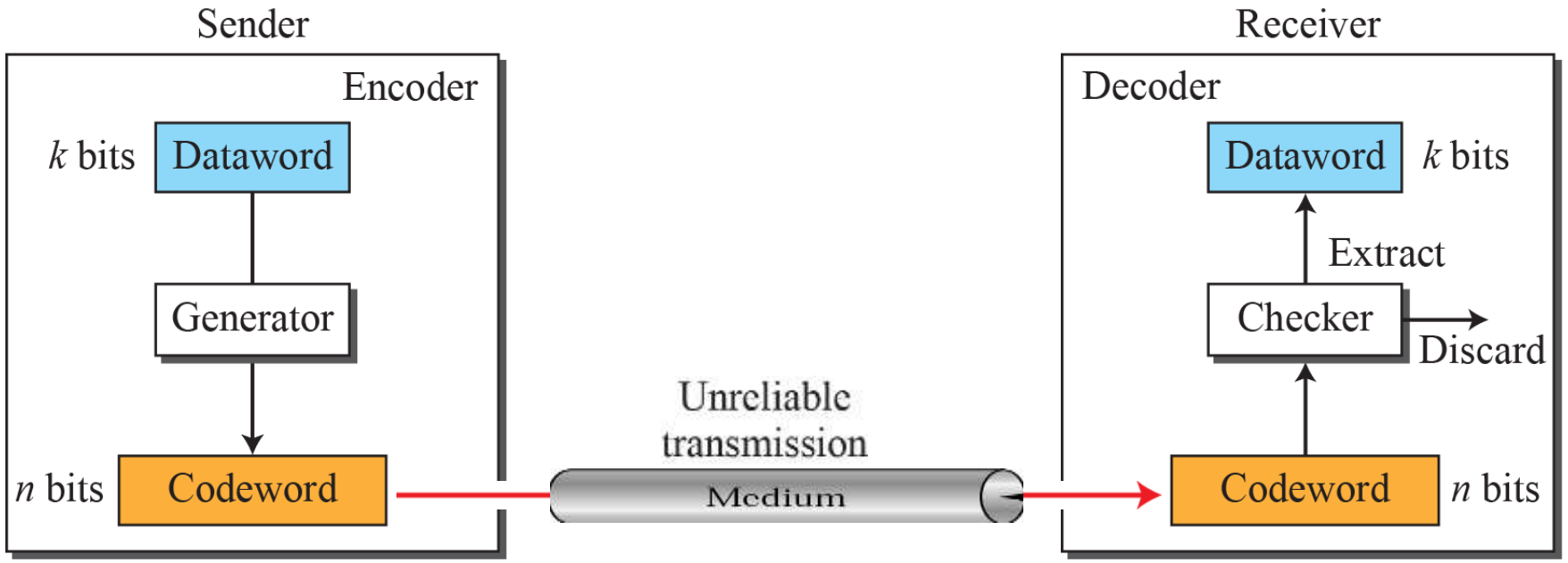
- ✓ 오류검출보다 정정이 더 어려움
  - 예) 8비트 데이터의 경우,
    - 단일 오류 정정시, 8개의 가능한 위치 고려해야 함
    - 2비트 오류 정정시, ${}_8C_2$=28개 가능한 위치 고려
- ✓ 정확히 몇 비트가, 어디에서, 잘못 되었는지를 확인
  - 오류검출을 포함하는 개념
- ✓ 재전송없이 잘못 전송된 비트를 정정 가능

§3. 오류 검출

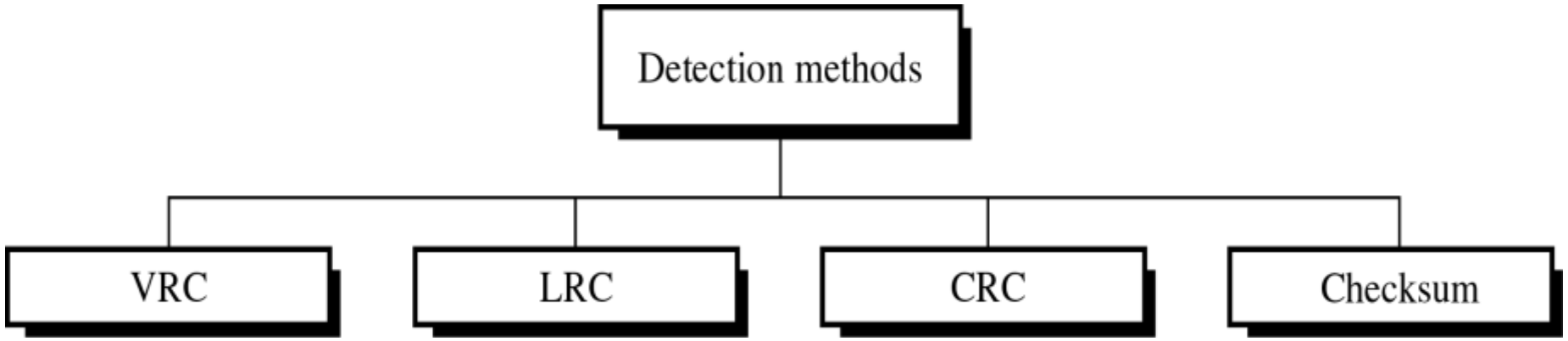
■ 오류 검출은
- ✓ 목적지에서 오류를 검출하기 위해서 데이터워드에 여분의 비트를 추가하는 중복 개념을 이용
- ✓ 송신자가 비트를 추가하여 보내고 수신자가 제거

■ 오류 검출 방법
- ✓ VRC(Vertical Redundancy Check) Parity Check
- ✓ LRC(Longitudinal Redundancy)
- ✓ CRC(Cyclical redundancy Check)
- ✓ Checksum

오류 검출을 위한 부호화기/복호화기



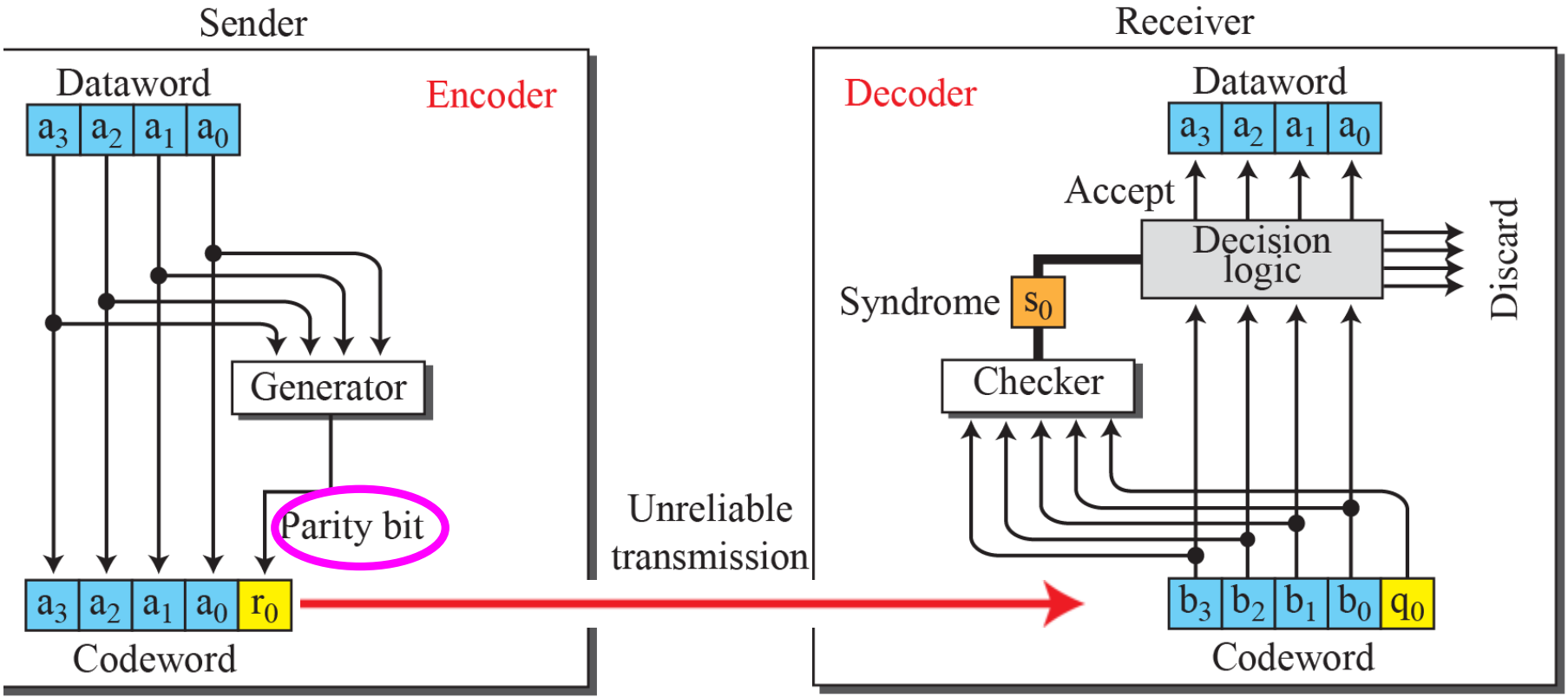
오류 검출 방법



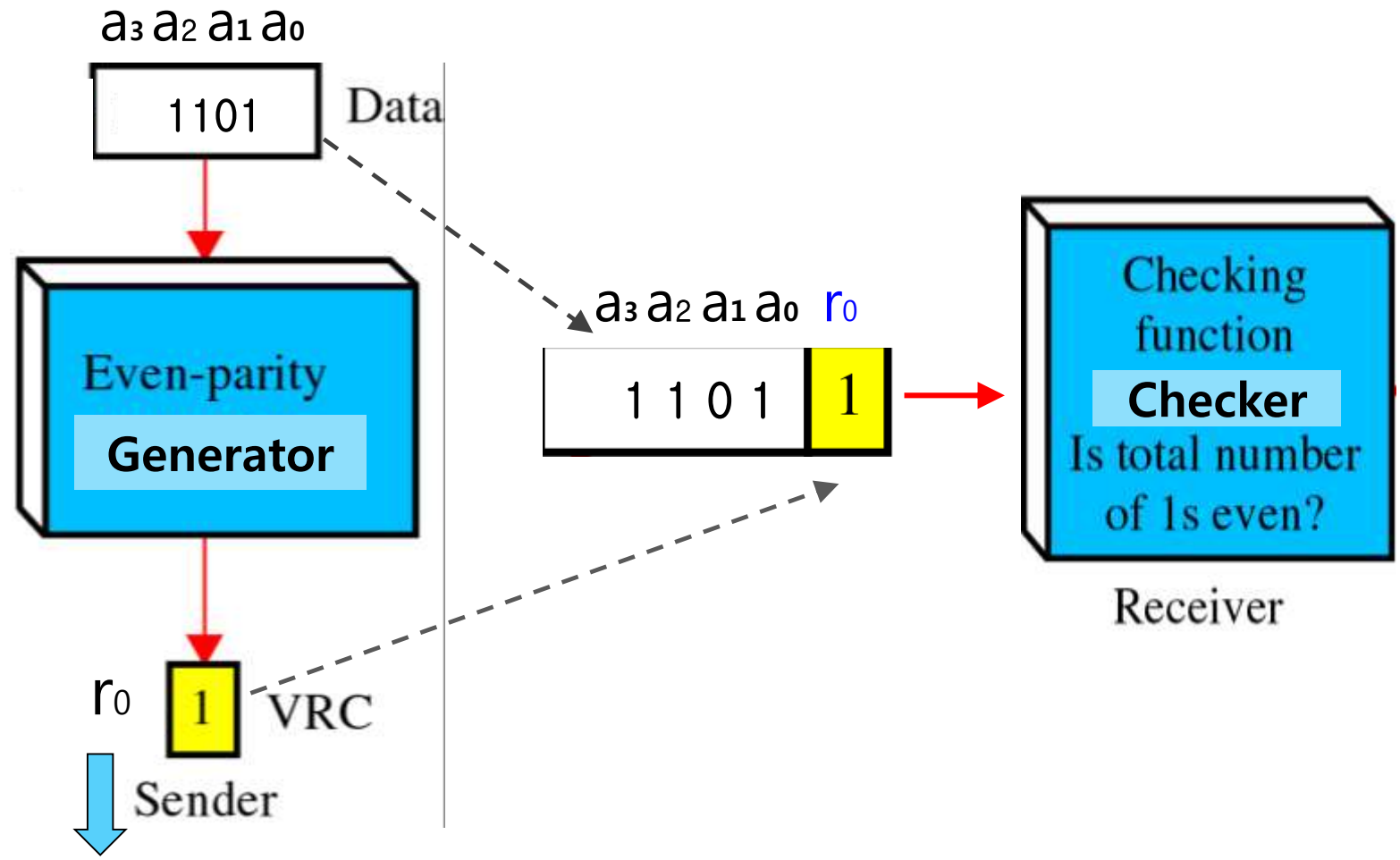
1) VRC(Vertical Redundancy Check)

Even Parity Bit

- 선형 블록 코드(Linear Block Code) 중 하나
- 오류 검출에 가장 널리 사용



Even Parity VRC (짝수 패리티 VRC)



Even-parity generator :

$r_0 = (a_3+a_2+a_1+a_0)\%2$

VRC 예

■ 예

"world"라는 단어 송신

- 1110111 1101111 1110010 1101100 1100100

parity bit 적용(even)후 전송

- 11101110 11011110 11100100 11011000 11001001

i) 수신된 정보(accept)

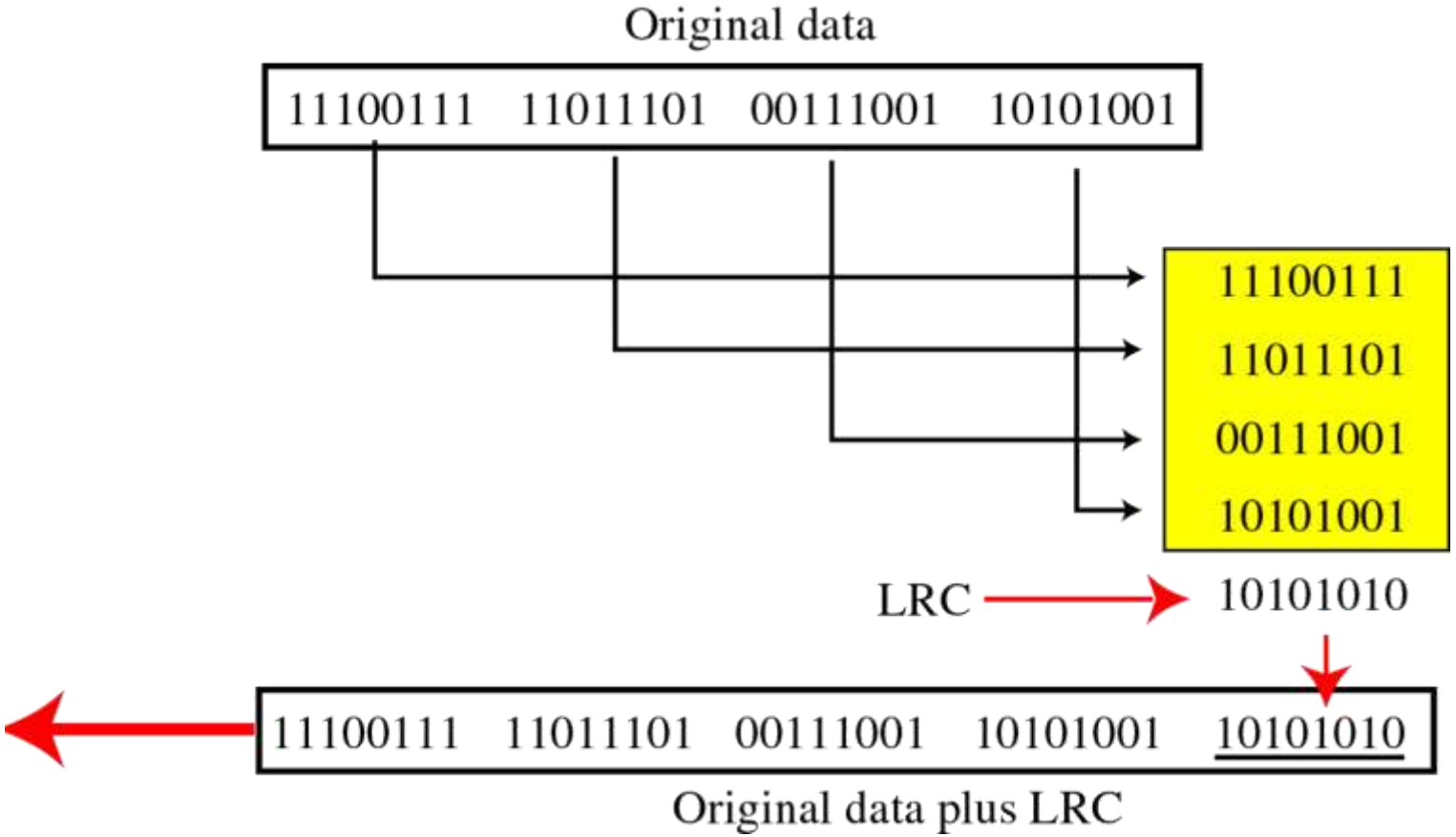
- 11101110 11011110 11100100 11011000 11001001

ii) 수신된 정보(reject, 재전송 요구)

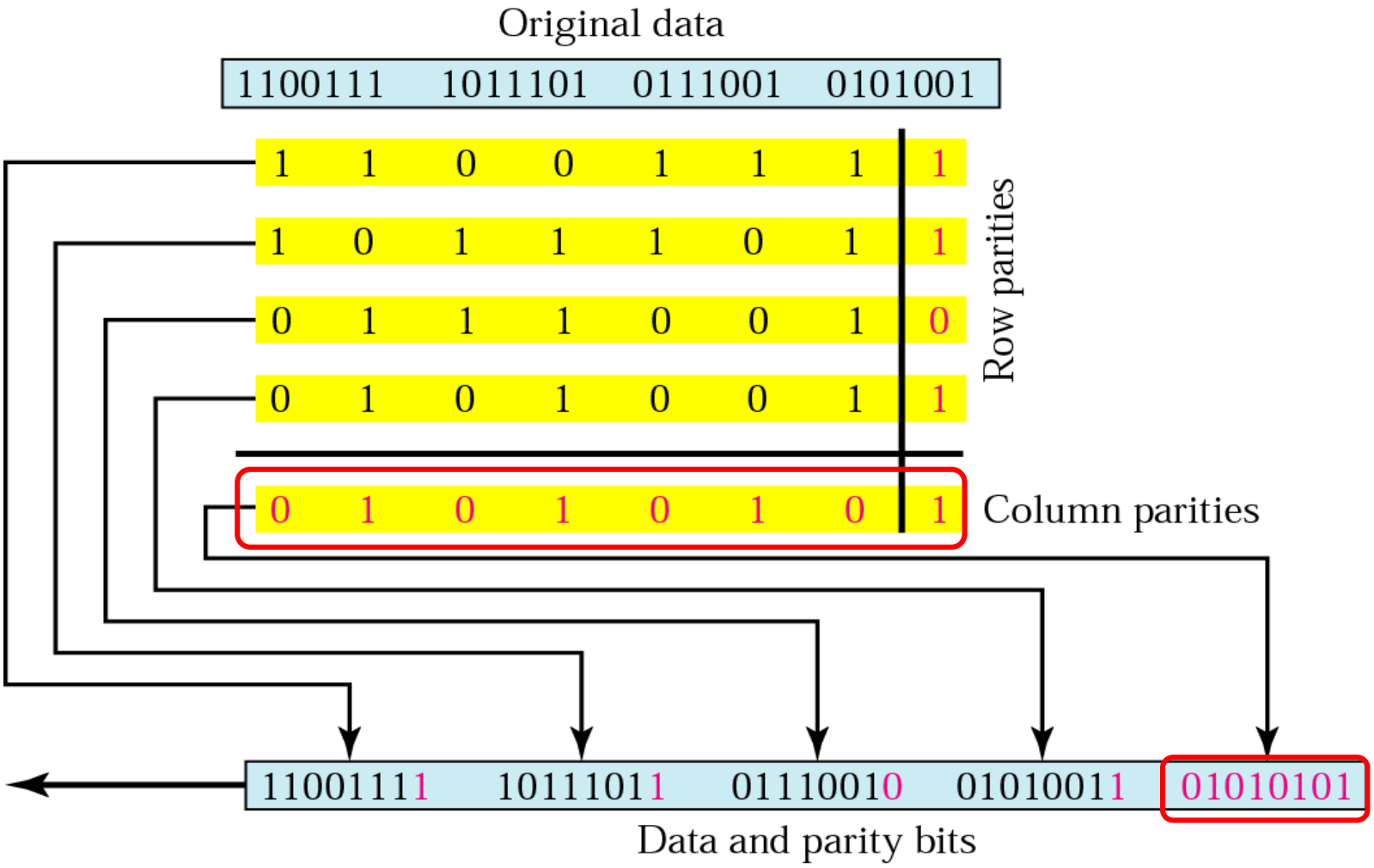
- 11111110 11011110 11101100 11011000 11001001

2) LRC(Longitudinal Redundancy Check)

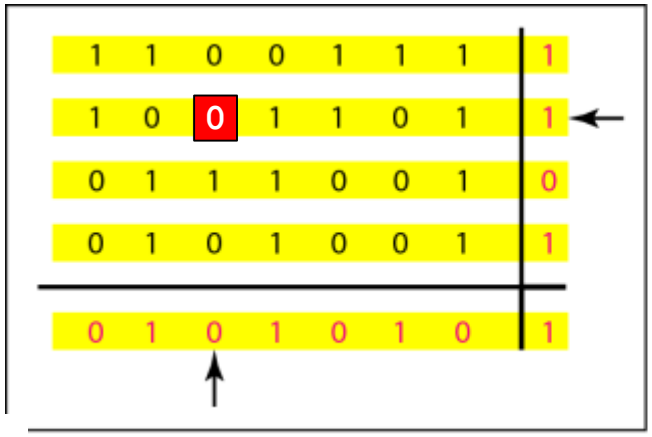
- 모든 바이트의 even parity를 모아 데이터 단위로 만들어서 데이터 블록의 맨 뒤에 추가



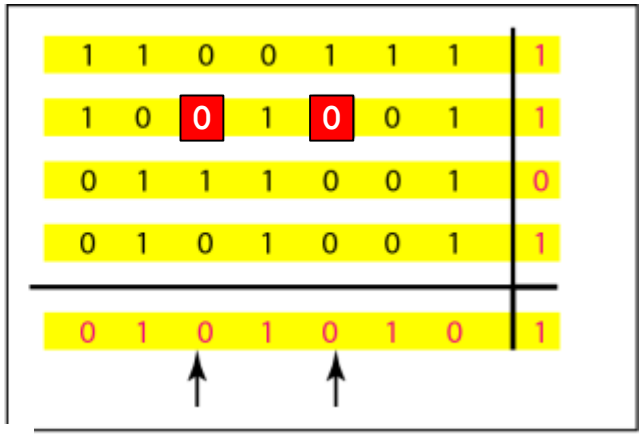
2차원 패리티


Original data
1100111 1011101 0111001 0101001
1 1 0 0 1 1 1 1
1 0 1 1 1 0 1 1
0 1 1 1 0 0 1 0
0 1 0 1 0 0 1 1
Row parities
0 1 0 1 0 1 0 1
Column parities
11001111 10111011 01110010 01010011 01010101
Data and parity bits

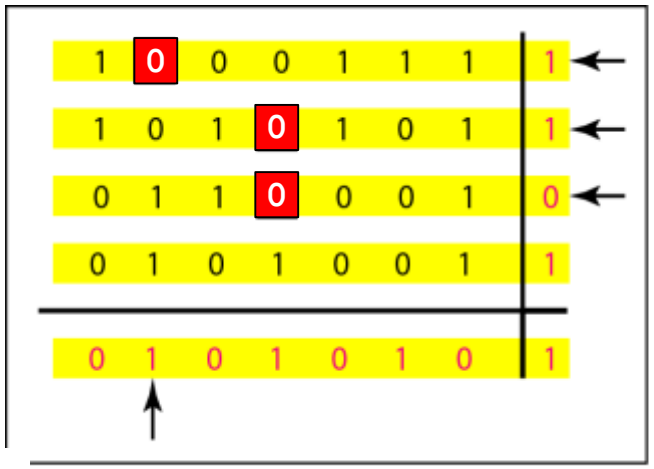
2차원 패리티의 오류 검출 예



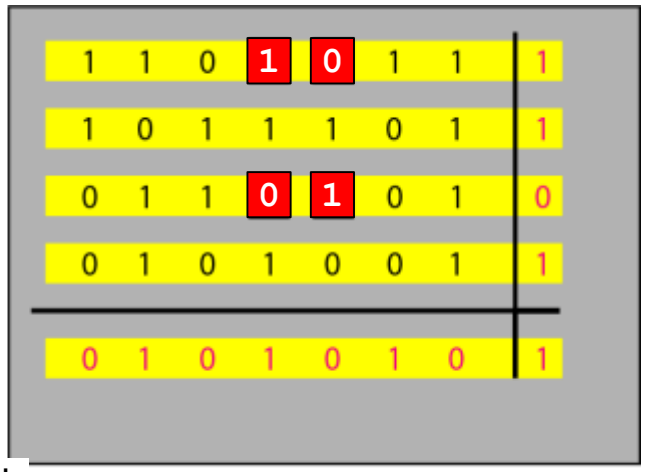
a. One error affects two parities



b. Two errors affect two parities



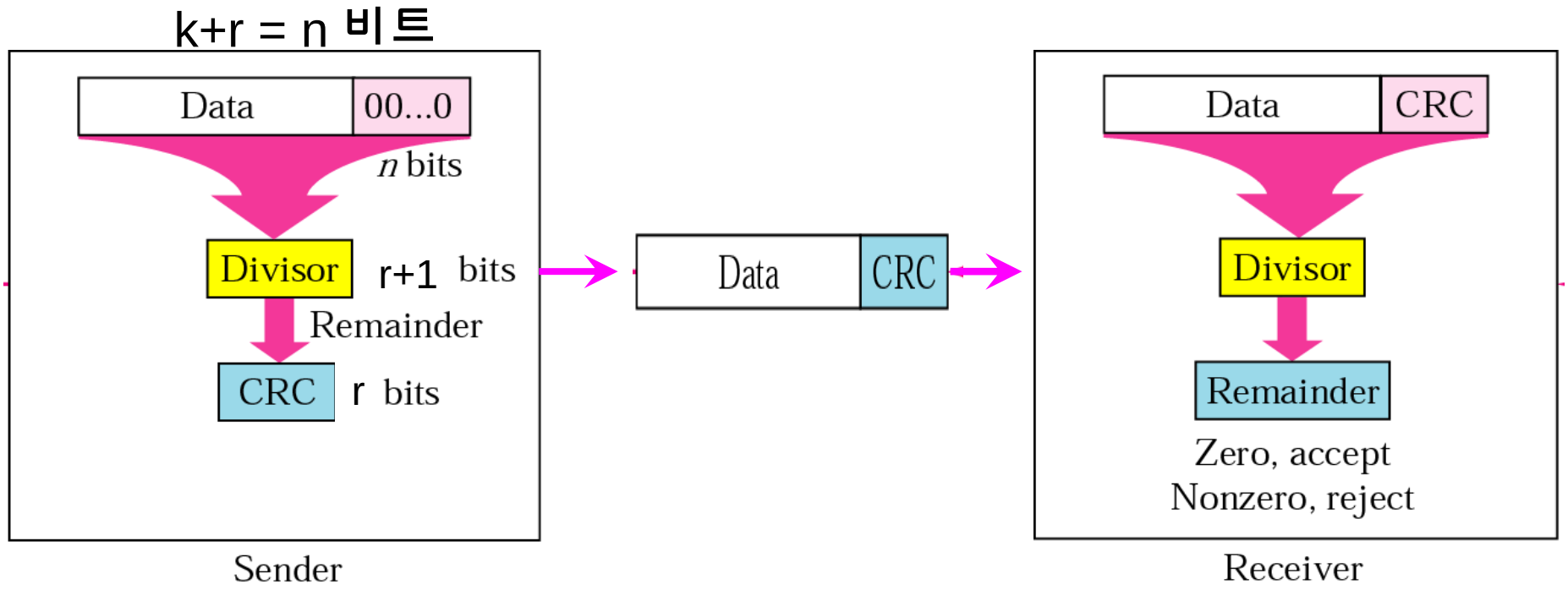
c. Three errors affect four parities



d. Four errors cannot be detected

3) CRC(Cyclic Redundancy Check) 순환중복검사

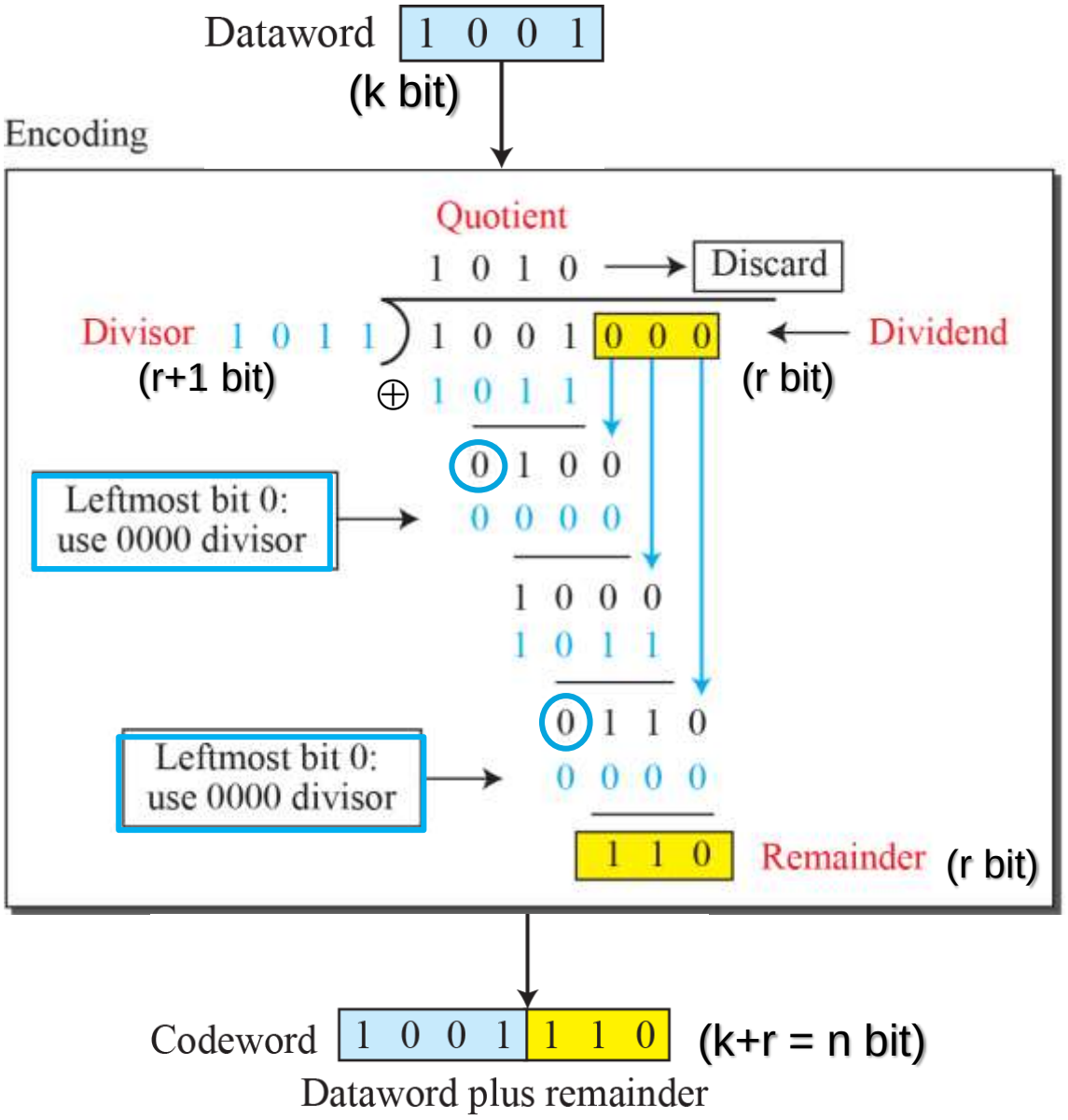
- 선형(linear)블록코드의 일종
- 순환코드를 사용하여 modulo-2 나눗셈 연산을 이용
  - ✓ **XOR**ing of two single bits
- LAN 등에서 널리 사용



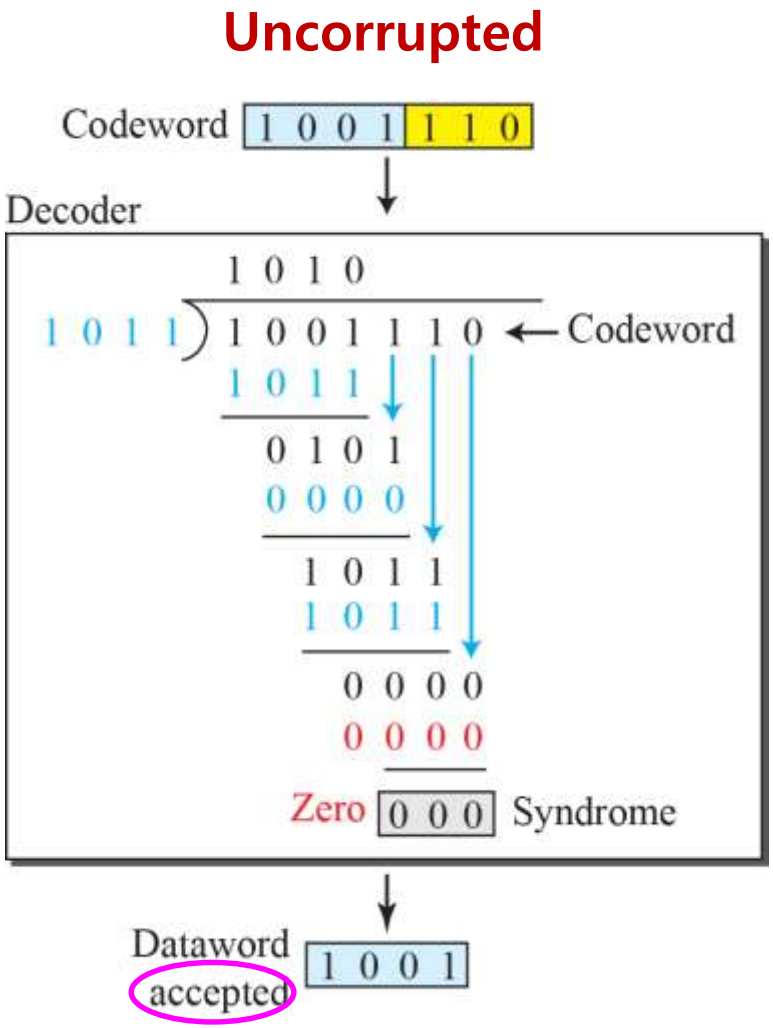
CRC generator(부호기)에서의 binary division

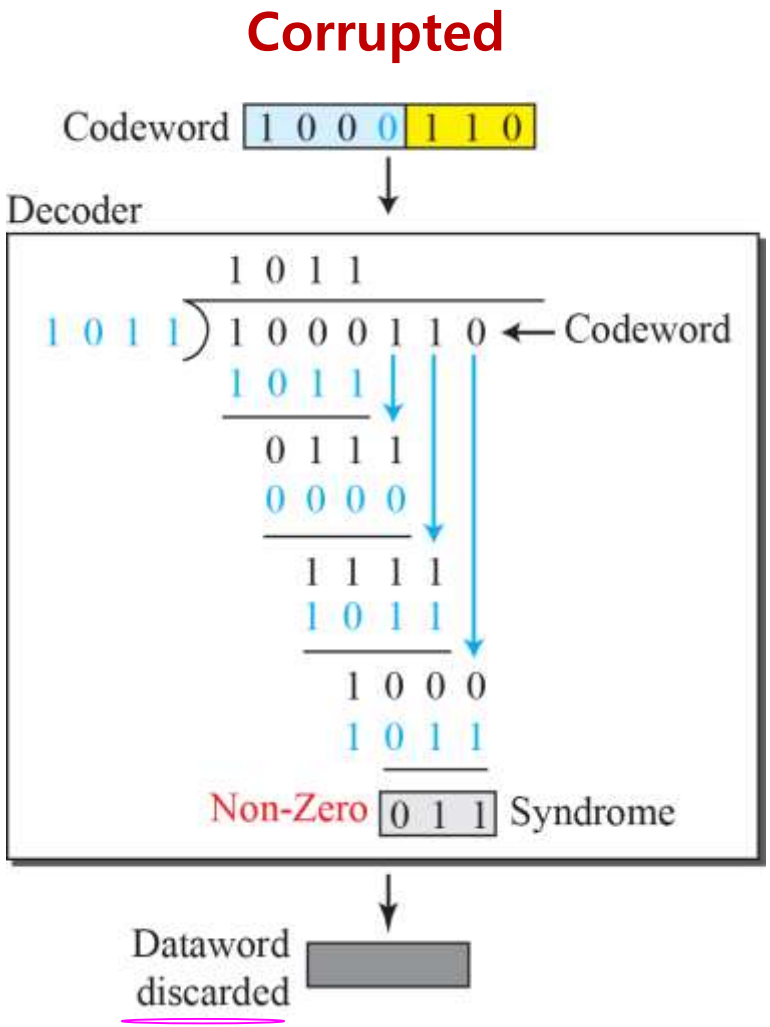
Note:

Multiply: AND
Subtract: XOR



CRC checker(복호기)에서의 binary division





CRC의 이점

- 단일 비트, 두 비트, 홀수개 비트 및 폭주 오류를 검출하는데 우수

- H/W나 S/W로 쉽게 구현 가능
  - ✓ H/W 구현시 특히 빠른 처리 가능

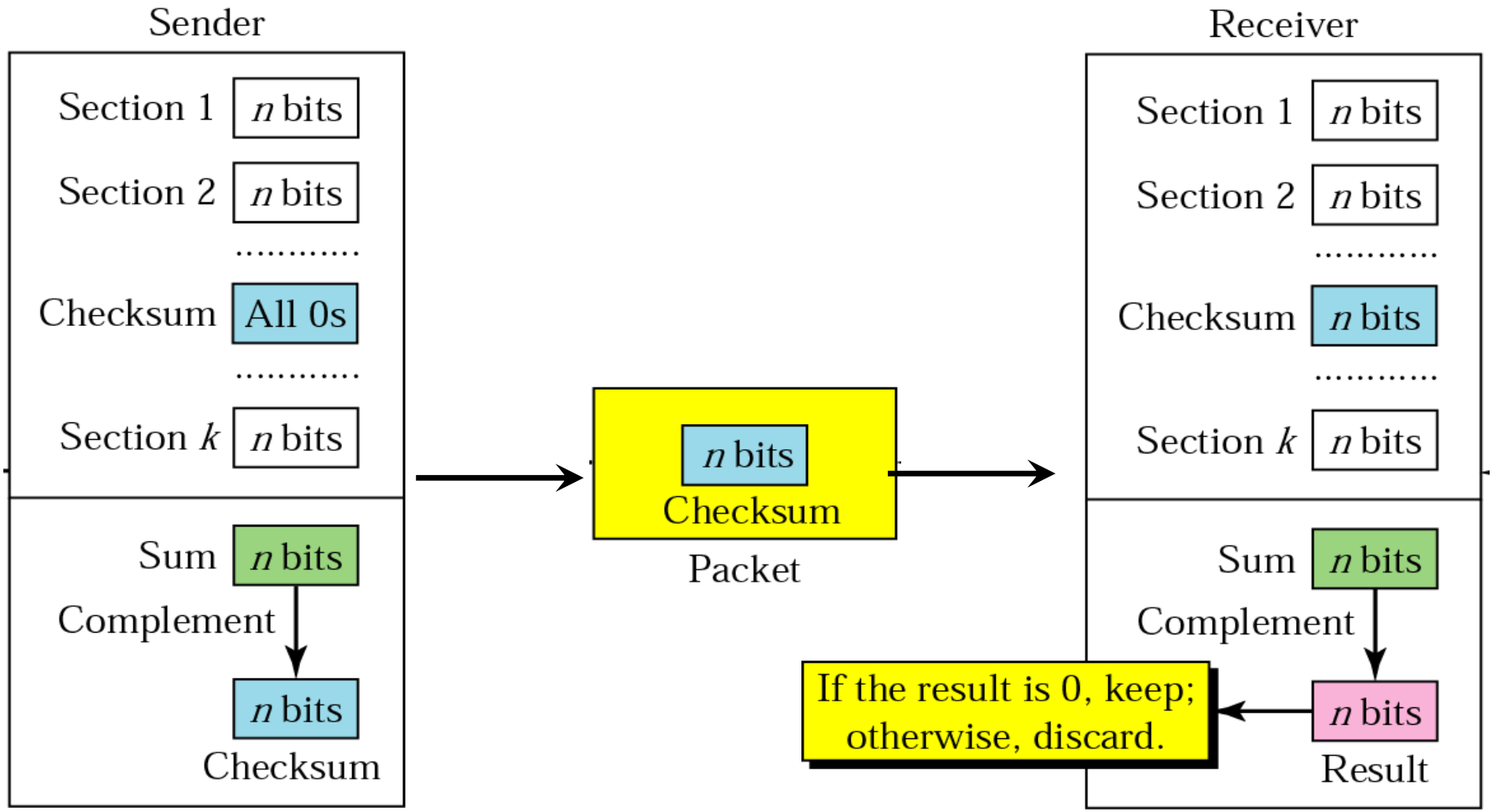
- 많은 네트워크에서 CRC 사용

4) Checksum (검사합)

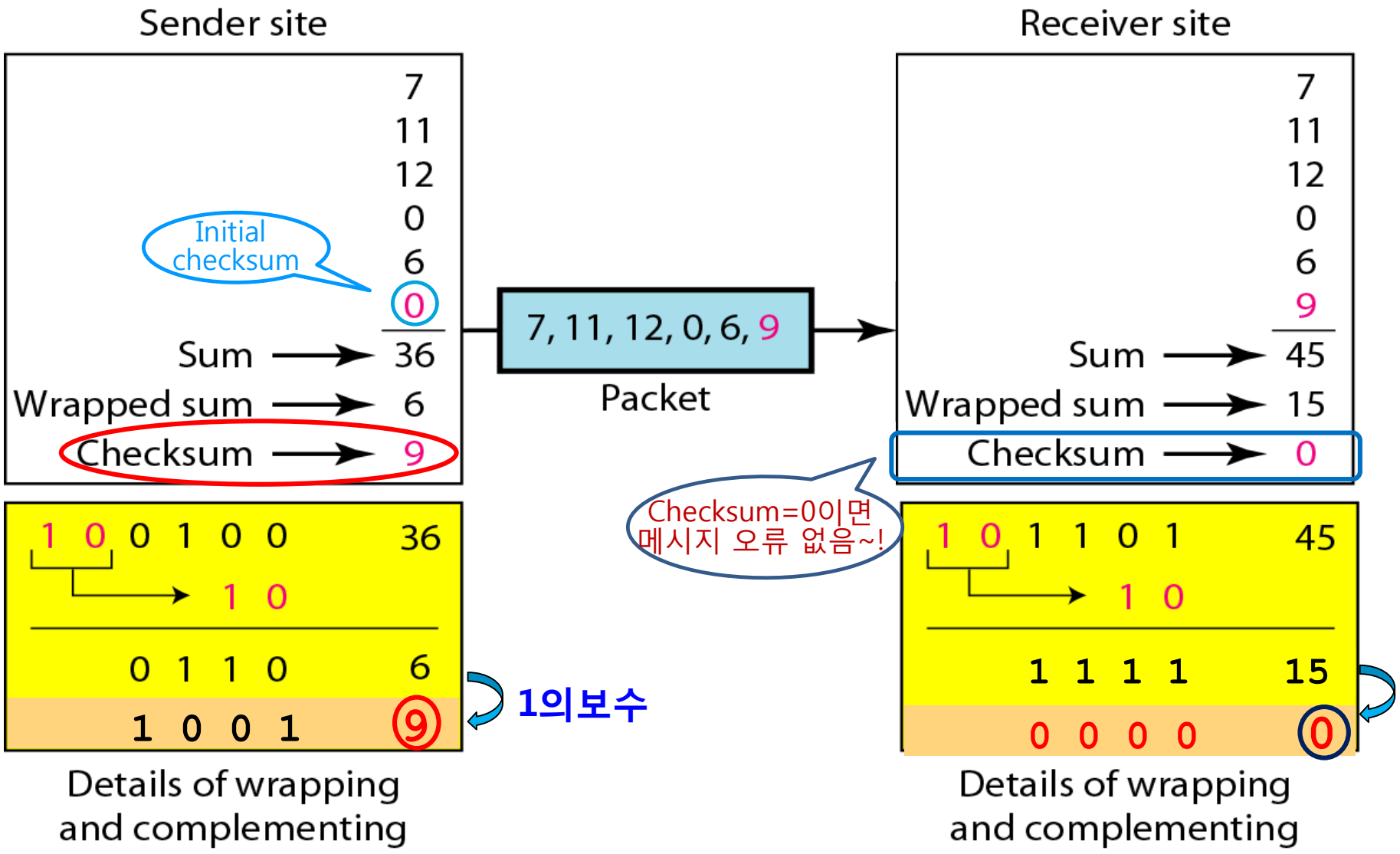
- 어떤 길이의 메시지에도 적용시킬 수 있는 오류 검출 기법

- 인터넷에서는 데이터링크 층보다 네트워크층, 전송층에서 검사합 기법이 주로 사용됨

Checksum 생성기



Checksum의 예


Sender site
7
11
12
0
6
0
Initial checksum
Sum → 36
Wrapped sum → 6
Checksum → 9
7, 11, 12, 0, 6, 9
Packet
Receiver site
7
11
12
0
6
9
Sum → 45
Wrapped sum → 15
Checksum → 0
Checksum=0이면 메시지 오류 없음~!
1 0 0 1 0 0 36
1 0
0 1 1 0 6
1 0 0 1 9
1의보수
Details of wrapping and complementing
1 0 1 1 0 1 45
1 0
1 1 1 1 15
0 0 0 0 0
Details of wrapping and complementing

§4. 오류 정정

1) 재전송에 의한 오류 정정

- ✓ 수신자는 오류가 발견되면 송신자에게 재전송 요청
  (11장 흐름제어와 오류 제어 프로토콜에서 다룸)

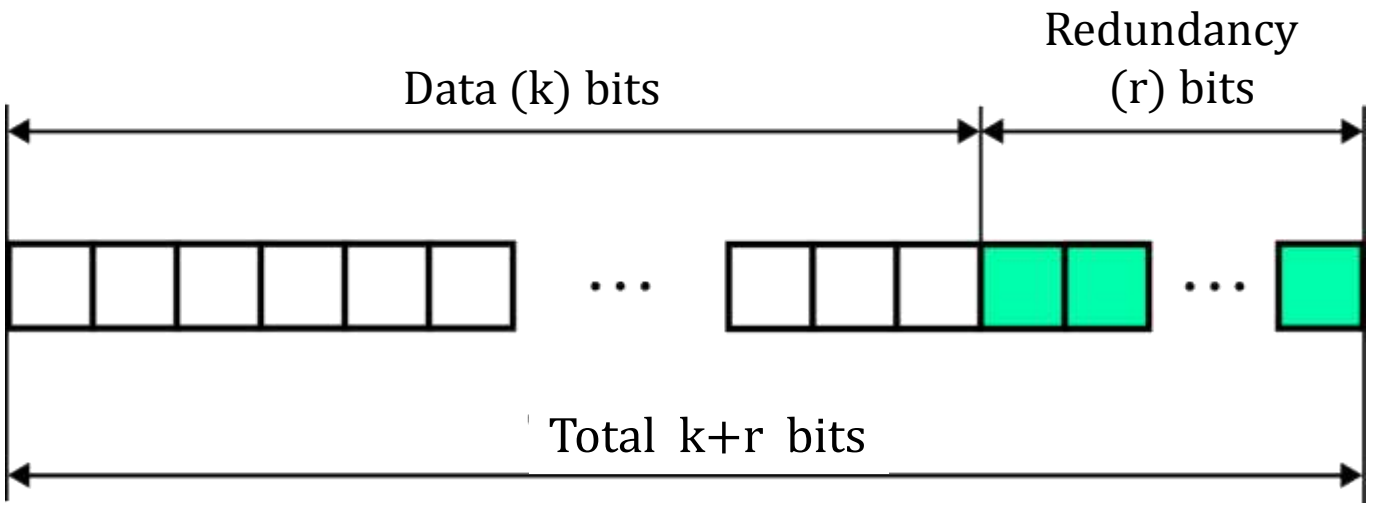
2) 오류 정정 코드의 사용

- ✓ 수신자가 오류 교정 코드를 이용하여 자동으로 수행
- ✓ 오류 검출 코드보다 훨씬 복잡하고, 더 많은 중복 비트를 요구 함

단일 비트 오류 정정

- 오류 발생 여부 뿐만 아니라
  어느 bit에서 발생했는지도 알 수 있어야 함

- Redundancy bit
  - ✓ 주어진 데이터 비트 수(k)를 정정하기 위해 요구되는 중복 비트 수(r)를 계산하기 위해 k와 r의 관계를 알아야 함



[참고] 단일비트 오류 정정을 위해 필요한 비트 수

■ 비트의 전체 수가 k+r이면 r은 적어도 다음 식을 만족해야 함

$$2^r \geq k + r + 1$$

예) 7비트 k에 대해 가장 적은 r 값은 4이다

$$2^4 \geq 7 + 4 + 1$$

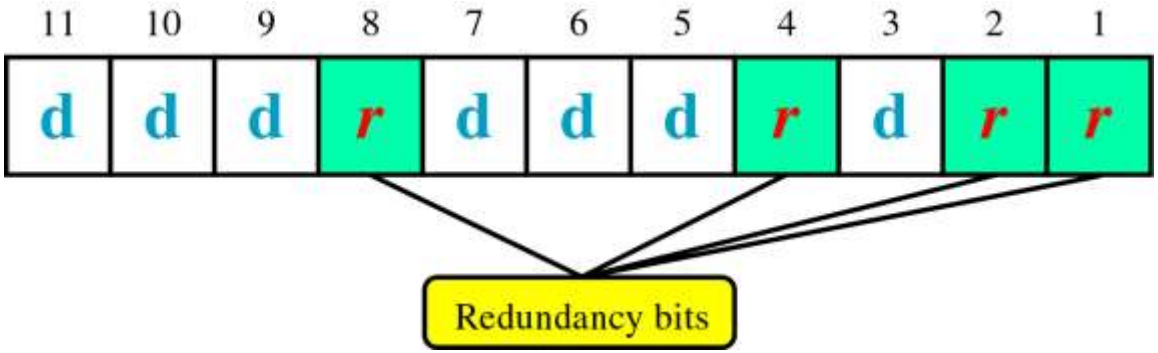
< 데이터와 중복 비트간의 관계 (단일비트 오류 정정) >

Number of data bits (k)	Number of redundancy bits (r)	Total bits (k + r)
1	2	3
2	3	5
3	3	6
4	3	7
5	4	9
6	4	10
7	4	11
8	4	12

해밍 코드 방식

- R.W. Hamming에 의해 개발
- 수신측에서 오류 발생 검출하여 직접 수정하는 방식
  - ✓ 1bit 오류만 정정가능
- 어떤 길이의 데이터 단위에도 적용 가능
  - ✓ 정보비트 이외 많은 잉여비트 필요
  - ✓ 전송비트 중 1,2,4,8,16,32,..., 2^n번째를 패러티 비트로 사용함

- **예:** 7bit 데이터 코드에는 4개의 중복비트 요구



Single error correction (Data:00111001)

■ 데이터(8비트) + 중복비트(4비트) = 12비트

bit position	position number	check bit	data bit	실제 data
12	1 1 0 0		D12	0
11	1 0 1 1		D11	0
10	1 0 1 0		D10	1
9	1 0 0 1		D9	1
8	1 0 0 0	P8		**0**
7	0 1 1 1		D7	1
6	0 1 1 0		D6	0
5	0 1 0 1		D5	0
4	0 1 0 0	P4		**1**
3	0 0 1 1		D3	1
2	0 0 1 0	P2		**1**
1	0 0 0 1	P1		**1**

P1 = D3 XOR D5 XOR D7 XOR D9 XOR D11
= 1 XOR 0 XOR 1 XOR 1 XOR 0 = **1**

P2 = D3 XOR D6 XOR D7 XOR D10 XOR D11
= 1 XOR 0 XOR 1 XOR 1 XOR 0 = **1**

P4 = D5 XOR D6 XOR D7 XOR D12
= 0 XOR 0 XOR 1 XOR 0 = **1**

P8 = D9 XOR D10 XOR D11 XOR D12
= 1 XOR 1 XOR 0 XOR 0 = **0**

➔ 중복비트 **0111**

12비트 해밍코드 생성

0011 0100 1111

해밍 코드에서의 오류 검출/정정(1)

sent → received

0011 0100 1111 ➔ 0011 0110 1111

bit position	position number	check bit	data bit	실제 data
12	1 1 0 0		D12	0
11	1 0 1 1		D11	0
10	1 0 1 0		D10	1
9	1 0 0 1		D9	1
8	1 0 0 0	P8		0
7	0 1 1 1		D7	1
6	0 1 1 0		D6	1
5	0 1 0 1		D5	0
4	0 1 0 0	P4		1
3	0 0 1 1		D3	1
2	0 0 1 0	P2		1
1	0 0 0 1	P1		1

P1 = D3 ⊕ D5 ⊕ D7 ⊕ D9 ⊕ D11

= 1 ⊕ 0 ⊕ 1 ⊕ 1 ⊕ 0 = **1**

P2 = D3 ⊕ D6 ⊕ D7 ⊕ D10 ⊕ D11

= 1 ⊕ 1 ⊕ 1 ⊕ 1 ⊕ 0 = **0**

P4 = D5 ⊕ D6 ⊕ D7 ⊕ D12

= 0 ⊕ 1 ⊕ 1 ⊕ 0 = **0**

P8 = D9 ⊕ D10 ⊕ D11 ⊕ D12

= 1 ⊕ 1 ⊕ 0 ⊕ 0 = **0**

➔ 중복비트 **0001**

받은 패킷의 중복비트(즉, check bit)는 0111인데,
수신자가 계산한 값은 0001 → 불일치!

해밍 코드에서의 오류 검출/정정(2)

■ 전송 후 check bit를 다시 계산 & 비교

0 1 1 1 (전송받은 패킷의 check bit)
⊕ 0 0 0 1 (새로 계산된 check bit)
---
0 1 1 0 ⇒ **bit position 6에서 error 발생했음!**

10장 - 끝 -