Il data link layer (*layer 2*)
Error detection
Error correction

Liceo G.B. Brocchi – Bassano del Grappa (VI)
Liceo Scientifico – opzione scienze applicate
Giovanni Mazzocchin

Il data link layer

- Lo scopo del livello fisico è inviare singoli bit come segnali elettromagnetici
- Lo scopo del livello **data link** è trasmettere unità di informazione chiamate **frame** tra macchine adiacenti, ossia collegate tramite un mezzo trasmissivo (sia esso guidato o non guidato)
- I protocolli di questo livello sono implementati a livello di **NIC** (***N***_etwork_ ***I***_nterface_ ***C***_ard_ - hardware) o come driver del sistema operativo (software di sistema)
- I servizi principali offerti dal livello data link al livello superiore sono:
    1. framing
    2. controllo degli errori
    3. controllo di flusso

Controllo degli errori - esempio spaziale

- Da un po' di decenni ammiriamo fotografie scattate da sonde distanti milioni di chilometri
- NASA ed ESA ricevono segnali particolarmente potenti da Giove o da Saturno?
- Ci sono ripetitori installati su pianeti e asteroidi?
- È conveniente chiedere ad una sonda di ritrasmettere nel caso in cui vengano rilevati errori? Qual è la probabilità che la prossima volta i dati arrivino integri?
- Un discorso simile si può fare anche per le reti wireless (e.g. Wi-Fi) terrestri e la comunicazione satellitare (canali molto rumorosi)

Ridondanza

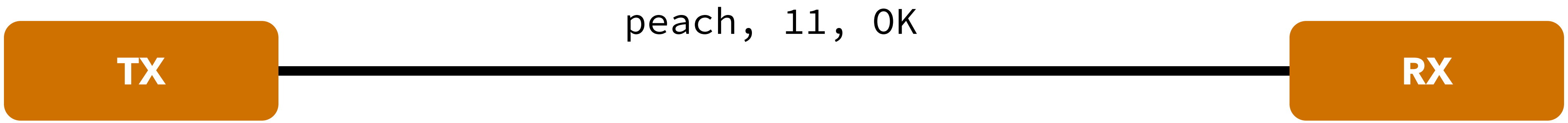
- **Ridondanza**: la presenza, nei messaggi emessi da una sorgente, di bit deducibili da altri bit del messaggio

- Vedremo che l'aggiunta di bit ridondanti ad un messaggio permette di rilevare (**detect**), e in alcuni casi correggere (**correct**), alcuni errori

Codice privo di ridondanza - problemi

fruit	code
banana	00
strawberry	01
orange	10
peach	11

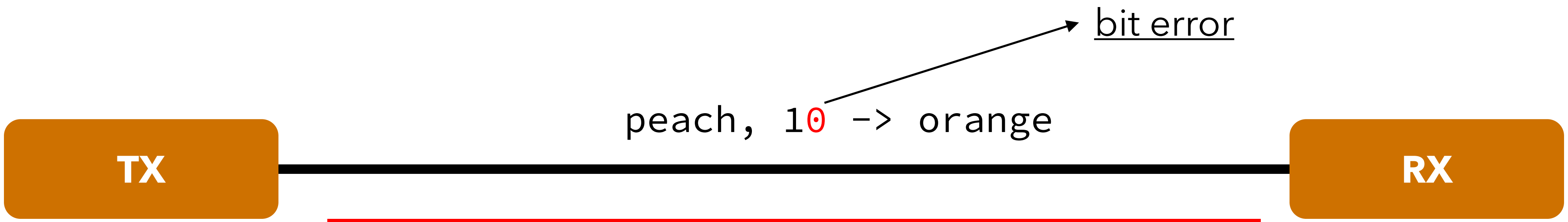
il codice non è ridondante:
2 bit per 4 frutti
(il numero minimo di bit per codificare 4 messaggi equiprobabili)



Codice privo di ridondanza - problemi

fruit	code
banana	00
strawberry	01
orange	10
peach	11

**il codice non è ridondante:
2 bit per 4 frutti
(il numero minimo di bit per codificare 4 messaggi equiprobabili)**



**TX ha inviato peach, ma RX riceve orange.
RX non ha modo di capire che c'è stato un errore**

Parity check

- **Parità**: *il fatto che un numero sia o pari o dispari*
- **Bit di parità pari**: bit ridondante aggiunto ai dati per fare in modo che il numero di 1 nella **codeword** (dati più bit di parità) sia pari
- **Bit di parità dispari**: bit ridondante aggiunto ai dati per fare in modo che il numero di 1 nella codeword sia dispari
- L'aggiunta di un *parity bit* permette di individuare un certo numero di bit error
- Vediamo degli esempi

Parity check

il trasmettitore vuole inviare questi 7 bit:

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

si aggiunge un bit di parità pari. Viene trasmessa la codeword seguente:

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

Parity check

se il ricevitore legge la codeword:

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

rileva un errore in quanto la parità è violata: 7 bit a `1`, ma la parità doveva essere pari (il bit di parità dovrebbe essere `0`)
Infatti, il terzo bit (evidenziato in rosso) è arrivato invertito

Parity check

se il ricevitore legge la codeword:

0	1	1	1	0	1	1	1

non rileva alcun errore in quanto la parità non è violata.
Ma i bit in rosso sono arrivati invertiti!

Il controllo di parità non permette l'error detection di un numero pari di bit invertiti (*flipped bits*)

Checksum

- **Cheksum**: <u>check + sum</u> (*controllo della somma*)
- Il messaggio viene visto come una sequenza di blocchi di n bit
- Esempio: questo messaggio di 12 bit:
  - **1 0 1 1 0 1 0 1 0 1 0 1**

  può essere suddiviso in blocchi di 3 bit, così:

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

- Si aggiunge al frame una sequenza di n bit ridondanti che rappresenta la somma dei blocchi di cui è composto il messaggio

Checksum

- <u>La somma può essere calcolata in vari modi, noi ne vediamo uno</u>
- Consideriamo questo messaggio:

  `10000100_00100100_11100010_10011001`

- Lo suddividiamo in blocchi di 8 bit e calcoliamo la somma binaria dei 4 blocchi:

1	0	0	0	0	1	0	0
0	0	1	0	0	1	0	0
1	1	1	0	0	0	1	0
1	0	0	1	1	0	0	1

Checksum

1	0	0	0	0	1	0	0
0	0	1	0	0	1	0	0
1	1	1	0	0	0	1	0
1	0	0	1	1	0	0	1
							1

Checksum

1	0	0	0	0	1	0	0
0	0	1	0	0	1	0	0
1	1	1	0	0	0	1	0
1	0	0	1	1	0	0	1
						1	1

Checksum

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

Checksum

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

Checksum

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

Checksum

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

Checksum

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

Checksum

		1							
		1	0	0	0	0	1	0	0
		0	0	1	0	0	1	0	0
		1	1	1	0	0	0	1	0
		1	0	0	1	1	0	0	1
1	0	0	0	1	0	0	0	1	1

Checksum

- Si aggiunge l'ultimo riporto alla somma limitata ad n bit:

0	0	1	0	0	0	1	1
						1	0
0	0	1	0	0	1	0	1

Checksum

- Si effettua il complemento a 1 della somma ottenuta sopra:

0	0	1	0	0	1	0	1
1	1	0	1	1	0	1	0

checksum

Checksum

- Il trasmettitore invia il messaggio seguito dal checksum calcolato sopra:

1	0	0	0	0	1	0	0
0	0	1	0	0	1	0	0
1	1	1	0	0	0	1	0
1	0	0	1	1	0	0	1
1	1	0	1	1	0	1	0

Checksum

- Il ricevitore esegue la somma di tutti i blocchi, compreso il checksum:
  - se il risultato è composto soltanto da `1`, `ACCEPT`
  - altrimenti, `REJECT`

- Vediamo cosa succede nei 2 casi

Checksum

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

caso 1: nessun errore

Checksum

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

caso 1: nessun errore

Checksum

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

caso 1: nessun errore

Checksum

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

caso 1: nessun errore

Checksum

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

caso 1: nessun errore

Checksum

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

caso 1: nessun errore

Checksum

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

caso 1: nessun errore

Checksum

		1							
		1	0	0	0	0	1	0	0
		0	0	1	0	0	1	0	0
		1	1	1	0	0	0	1	0
		1	0	0	1	1	0	0	1
		1	1	0	1	1	0	1	0
1	0	1	1	1	1	1	1	0	1

caso 1: nessun errore

Checksum

1	1	1	1	1	1	0	1
						1	0
1	1	1	1	1	1	1	1

all 1's -> ACCEPT

caso 1: nessun errore

Checksum

1	0	0	0	0	1	0	0
0	0	1	0	0	1	0	0
1	1	1	1	0	0	1	0
1	0	0	1	1	0	0	1
1	1	0	1	1	0	1	0
							1

caso 2: un bit invertito

Checksum

					1		
1	0	0	0	0	1	0	0
0	0	1	0	0	1	0	0
1	1	1	1	0	0	1	0
1	0	0	1	1	0	0	1
1	1	0	1	1	0	1	0
						0	1

caso 2: un bit invertito

Checksum

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

caso 2: un bit invertito

Checksum

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

caso 2: un bit invertito

Checksum

		10	*1*				
1	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0	**0**	**1**	**0**	**0**	**1**	**0**	**0**
1	**1**	**1**	**1**	**0**	**0**	**1**	**0**
1	**0**	**0**	**1**	**1**	**0**	**0**	**1**
1	**1**	**0**	**1**	**1**	**0**	**1**	**0**
			0	**1**	**1**	**0**	**1**

caso 2: un bit invertito

Checksum

	10	*10*					
1	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0	**0**	**1**	**0**	**0**	**1**	**0**	**0**
1	**1**	**1**	**1**	**0**	**0**	**1**	**0**
1	**0**	**0**	**1**	**1**	**0**	**0**	**1**
1	**1**	**0**	**1**	**1**	**0**	**1**	**0**
		0	**0**	**1**	**1**	**0**	**1**

caso 2: un bit invertito

Checksum

10	*10*						
1	0	0	0	0	1	0	0
0	0	1	0	0	1	0	0
1	1	1	1	0	0	1	0
1	0	0	1	1	0	0	1
1	1	0	1	1	0	1	0
	0	0	0	1	1	0	1

caso 2: un bit invertito

Checksum

		10							
		1	0	0	0	0	1	0	0
		0	0	1	0	0	1	0	0
		1	1	1	1	0	0	1	0
		1	0	0	1	1	0	0	1
		1	1	0	1	1	0	1	0
1	1	0	0	0	0	1	1	0	1

caso 2: un bit invertito

Checksum

0	0	0	0	1	1	0	1
						1	1
0	0	0	1	0	0	0	0

not all 1's -> REJECT

caso 2: un bit invertito

Hamming codes (error correction)

message	code
rainy	0
sunny	1

il codice non è ridondante:
1 bit per 2 possibili messaggi

`rainy, 0, OK`

TX —— **RX**

Hamming codes

message	code
rainy	0
sunny	1

il codice non è ridondante:
1 bit per 2 possibili messaggi

bit error

`rainy, 1 -> sunny`

TX

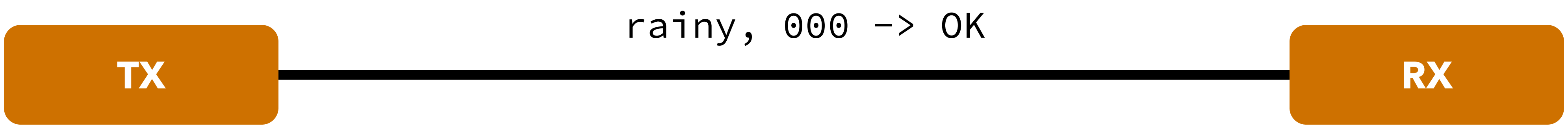
RX

TX ha inviato `rainy`, ma RX riceve sunny.
RX non ha modo di capire che c'è stato un errore,
né tanto meno di correggerlo

Hamming codes

message	code
rainy	000
sunny	111

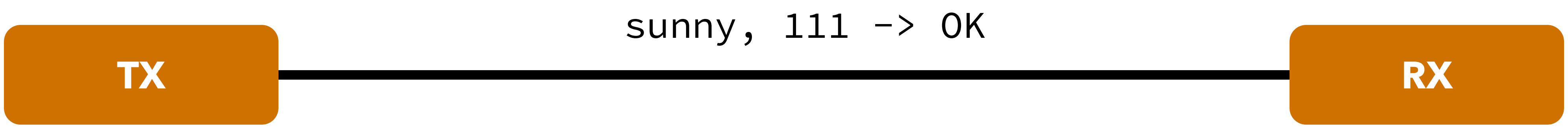
vengono inviati 2 bit in più del necessario... perché?



Hamming codes

message	code
rainy	000
sunny	111

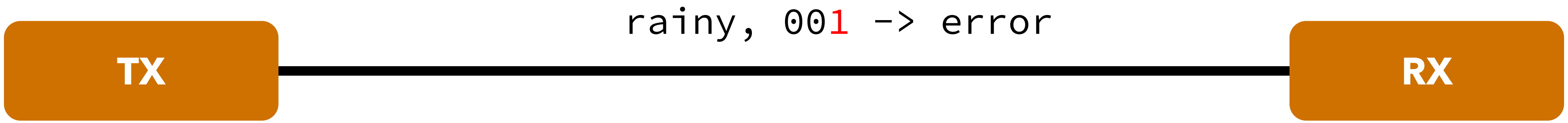
vengono inviati 2 bit in più del necessario… perché?



Hamming codes

message	code
rainy	000
sunny	111

vengono inviati 2 bit in più del necessario… perché?



1 bit error

Hamming codes

message	code
rainy	000
sunny	111

vengono inviati 2 bit in più del necessario... perché?

`rainy, 010 -> error`

TX

RX

1 bit error

Hamming codes

message	code
rainy	000
sunny	111

vengono inviati 2 bit in più del necessario… perché?

`rainy, 100 -> error`

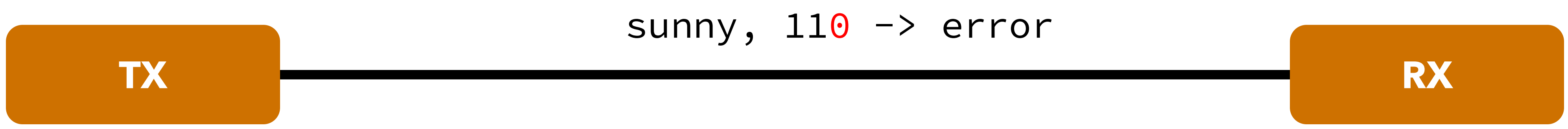
TX **RX**

1 bit error

Hamming codes

message	code
rainy	000
sunny	111

vengono inviati 2 bit in più del necessario… perché?

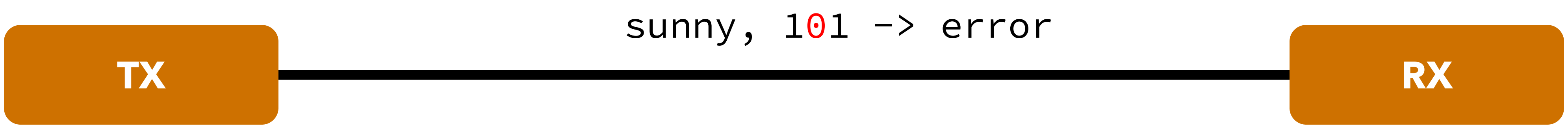


1 bit error

Hamming codes

message	code
rainy	000
sunny	111

vengono inviati 2 bit in più del necessario... perché?



1 bit error

Hamming codes

message	code
rainy	000
sunny	111

vengono inviati 2 bit in più del necessario… perché?

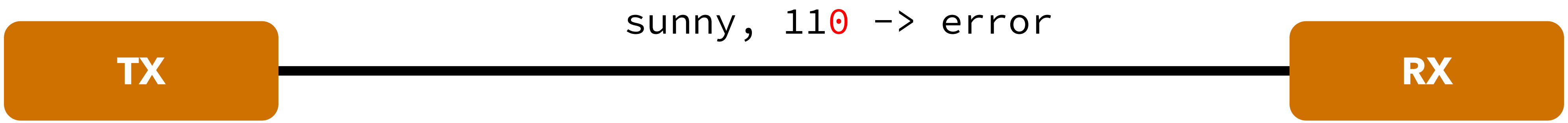
`sunny, 011 -> error`

TX

RX

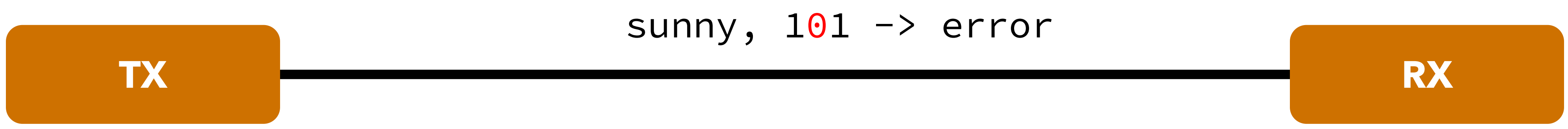
1 bit error

Hamming codes



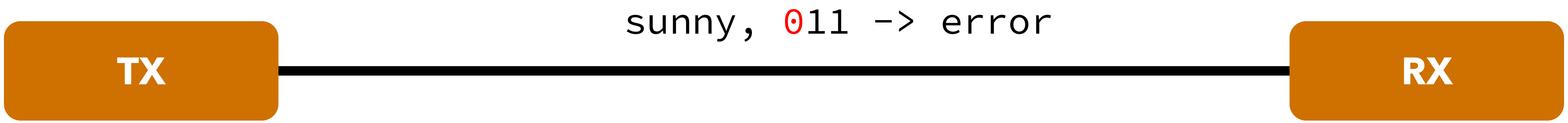
- RX riesce a correggere l'errore. Infatti, `110` è una codeword non valida. Quella valida più vicina è `111`. RX ha quindi *capito* che il bit invertito era il terzo
- RX inverte il terzo bit e decodifica `111`, ossia `sunny`

Hamming codes



- RX riesce a correggere l'errore. Infatti, `101` è una codeword non valida. Quella valida più vicina è `111`. RX ha quindi *capito* che il bit invertito era il secondo
- RX inverte il secondo bit e decodifica `111`, ossia `sunny`

Hamming codes



- RX riesce a correggere l’errore. Infatti, `011` è una codeword non valida. Quella valida più vicina è `111`. RX ha quindi *capito* che il bit invertito era il primo
- RX inverte il primo bit e decodifica `111`, ossia `sunny`

Hamming codes

valid codewords
000
111

invalid codewords
001
010
100
110
101
011

Hamming codes

$$m + r + 1 \leq 2^r$$

$\boldsymbol{m}$: numero di bit del messaggio
$\boldsymbol{r}$: numero di check bit

- Questa disequazione pone un *lower bound* (limite inferiore) al numero di check bit necessari per **correggere un errore su un singolo bit**
- Il metodo per raggiungere questo lower bound è stato elaborato da Richard Hamming nel 1950
- Studiamone il funzionamento

Hamming codes

Messaggio da trasmettere:

```
1 0 0 0 0 0 1
```

m: 7

Secondo la disequazione vista sopra, il valore minimo di r è 4.
Hamming(11, 7): una *codeword* sarà lunga 11 bit, di cui 7 di dati e 4 di controllo (*check bits*)

Quali valori assegneremo i 4 check bit? Per capirlo, lavoriamo su 11 scatolette indicizzate a partire da 1

Hamming codes

Inseriamo i bit del messaggio nelle scatolette il cui indice NON è una potenza di 2. I check bit andranno calcolati e inseriti nelle posizioni potenze di 2

1	2	3	4	5	6	7	8	9	10	11
		1		**0**	**0**	**0**		**0**	**0**	**1**

Scriviamo gli indici in binario:

0	0000	**6**	**0110**
1	**0001**	**7**	**0111**
2	**0010**	**8**	**1000**
3	**0011**	**9**	**1001**
4	**0100**	**10**	**1010**
5	**0101**	**11**	**1011**

Hamming codes

Il check bit `i`-esimo va calcolato come XOR dei bit della codeword il cui indice in binario ha 1 in posizione `i` (contando i bit degli indici da destra da 1 e tralasciando il check bit stesso; CW: *codeword*)

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
0		**1**		**0**	**0**	**0**		**0**	**0**	**1**
c1	*c2*		*c3*				*c4*			

```
c1 = CW[3] XOR CW[5] XOR CW[7] XOR CW[9] XOR CW[11] =
1 XOR 0 XOR 0 XOR 0 XOR 0 XOR 1 = 0
```

0	0000	**6**	**0110**
1	**0001**	**7**	**0111**
2	**0010**	**8**	**1000**
3	**0011**	**9**	**1001**
4	**0100**	**10**	**1010**
5	**0101**	**11**	**1011**

Hamming codes

1	2	3	4	5	6	7	8	9	10	11
0	**0**	**1**		**0**	**0**	**0**		**0**	**0**	**1**
c1	*c2*		*c3*				*c4*			

```
c2 = CW[3] XOR CW[6] XOR CW[7] XOR CW[10] XOR CW[11] =
1 XOR 0 XOR 0 XOR 0 XOR 1 = 0
```

0	0000	**6**	**0110**
1	**0001**	**7**	**0111**
2	**0010**	**8**	**1000**
3	**0011**	**9**	**1001**
4	**0100**	**10**	**1010**
5	**0101**	**11**	**1011**

Hamming codes

1	2	3	4	5	6	7	8	9	10	11
0	**0**	**1**	**0**	**0**	**0**	**0**		**0**	**0**	**1**
c1	*c2*		*c3*				*c4*			

```
c3 = CW[5] XOR CW[6] XOR CW[7] =
0 XOR 0 XOR 0 = 0
```

0	0000	**6**	**0110**
1	**0001**	**7**	**0111**
2	**0010**	**8**	**1000**
3	**0011**	**9**	**1001**
4	**0100**	**10**	**1010**
5	**0101**	**11**	**1011**

Hamming codes

1	2	3	4	5	6	7	8	9	10	11
0	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**
c1	*c2*		*c3*				*c4*			

```
c4 = CW[9] XOR CW[10] XOR CW[11] =
0 XOR 0 XOR 1 = 1
```

0	0000	**6**	**0110**
1	**0001**	**7**	**0111**
2	**0010**	**8**	**1000**
3	**0011**	**9**	**1001**
4	**0100**	**10**	**1010**
5	**0101**	**11**	**1011**

Hamming codes

Ipotizziamo che la codeword costruita sopra, ossia:

```
0 0 1 0 0 0 0 1 0 0 1
```

venga inviata su un canale rumoroso. Il ricevitore legge:

```
0 0 1 0 1 0 0 1 0 0 1
```

il ricevitore ricalcola i check bit, questa volta comprendendo i check bit stessi nel calcolo (`CW` è la codeword letta dal ricevitore):

```
c1 = CW[1] XOR CW[3] XOR CW[5] XOR CW[7] XOR CW[9] XOR CW[11]
= 0 XOR 1 XOR 1 XOR 0 XOR 0 XOR 1 = 1
```

Hamming codes

Ipotizziamo che la codeword costruita sopra, ossia:

```
0 0 1 0 0 0 0 1 0 0 1
```

venga inviata su un canale rumoroso. Il ricevitore legge:

```
0 0 1 0 1 0 0 1 0 0 1
```

il ricevitore ricalcola i check bit, questa volta comprendendo i check bit stessi nel calcolo (`CW` è la codeword letta dal ricevitore):

```
c2 = CW[2] XOR CW[3] XOR CW[6] XOR CW[7] XOR CW[10] XOR CW[11]
= 0 XOR 1 XOR 0 XOR 0 XOR 0 XOR 1 = 0
```

Hamming codes

Ipotizziamo che la codeword costruita sopra, ossia:

```
0 0 1 0 0 0 0 1 0 0 1
```

venga inviata su un canale rumoroso. Il ricevitore legge:

```
0 0 1 0 1 0 0 1 0 0 1
```

il ricevitore ricalcola i check bit, questa volta comprendendo i check bit stessi nel calcolo (`CW` è la codeword letta dal ricevitore):

```
c3 = CW[4] XOR CW[5] XOR CW[6] XOR CW[7]
= 0 XOR 1 XOR 0 XOR 0 = 1
```

Hamming codes

Ipotizziamo che la codeword costruita sopra, ossia:

```
0 0 1 0 0 0 0 1 0 0 1
```

venga inviata su un canale rumoroso. Il ricevitore legge:

```
0 0 1 0 1 0 0 1 0 0 1
```

il ricevitore ricalcola i check bit, questa volta comprendendo i check bit stessi nel calcolo (`CW` è la codeword letta dal ricevitore):

```
c4 = CW[8] XOR CW[9] XOR CW[10] XOR CW[11]
= 1 XOR 0 XOR 0 XOR 1 = 0
```

Hamming codes

Riscriviamo i bit calcolati in ordine dall'ultimo al primo:

`0 1 0 1`

Interpretiamo la sequenza di bit come numero binario e convertiamolo in decimale. Questo numero è detto **error syndrome**:

$0101_{bin} = 5_{dec}$

Questo significa che il bit in posizione 5 è sbagliato. Il ricevitore lo inverte, correggendo così l'errore e recuperando il messaggio originale

Hamming codes

Messaggio da trasmettere:

`1111000010101110`

m: 16

Il valore minimo di r è 5.

Hamming(21, 16): una *codeword* sarà lunga 21 bit, di cui 16 di dati e 5 di controllo (*check bits*)

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
		1		**1**	**1**	**1**		**0**	**0**	**0**	**0**	**1**	**0**	**1**		**0**	**1**	**1**	**1**	**0**

Scriviamo gli indici in binario:

0	0000	**6**	**0110**	**12**	**1100**	**18**	**10010**
1	**0001**	**7**	**0111**	**13**	**1101**	**19**	**10011**
2	**0010**	**8**	**1000**	**14**	**1110**	**20**	**10100**
3	**0011**	**9**	**1001**	**15**	**1111**	**21**	**10101**
4	**0100**	**10**	**1010**	**16**	**10000**		
5	**0101**	**11**	**1011**	**17**	**10001**		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0		**1**		**1**	**1**	**1**		**0**	**0**	**0**	**0**	**1**	**0**	**1**		**0**	**1**	**1**	**1**	**0**

```
P1 = CW[1] =
CW[3] ^ CW[5] ^ CW[7] ^ CW[9] ^ CW[11] ^ CW[13] ^ CW[15] ^ CW[17] ^ CW[19] ^ CW[21] = 0
```

0	0000	**6**	**0110**	**12**	**1100**	**18**	**10010**
1	**0001**	**7**	**0111**	**13**	**1101**	**19**	**10011**
2	**0010**	**8**	**1000**	**14**	**1110**	**20**	**10100**
3	**0011**	**9**	**1001**	**15**	**1111**	**21**	**10101**
4	**0100**	**10**	**1010**	**16**	**10000**		
5	**0101**	**11**	**1011**	**17**	**10001**		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	1		1	1	1		0	0	0	0	1	0	1		0	1	1	1	0

```
P2 = CW[2] =
CW[3] ^ CW[6] ^ CW[7] ^ CW[10] ^ CW[11] ^ CW[14] ^ CW[15] ^ CW[18] ^ CW[19] = 0
```

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*
0	**0**	**1**	**0**	**1**	**1**	**1**		**0**	**0**	**0**	**0**	**1**	**0**	**1**		**0**	**1**	**1**	**1**	**0**

P_3 = CW[4] =
CW[5] ^ CW[6] ^ CW[7] ^ CW[12] ^ CW[13] ^ CW[14] ^ CW[15] ^ CW[20] ^ CW[21] = 0

0	0000	**6**	**0110**	**12**	**1100**	**18**	**10010**
1	**0001**	**7**	**0111**	**13**	**1101**	**19**	**10011**
2	**0010**	**8**	**1000**	**14**	**1110**	**20**	**10100**
3	**0011**	**9**	**1001**	**15**	**1111**	**21**	**10101**
4	**0100**	**10**	**1010**	**16**	**10000**		
5	**0101**	**11**	**1011**	**17**	**10001**		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	1	0	1	1	1	0	0	0	0	0	1	0	1		0	1	1	1	0

P_4 = CW[8] =
CW[9] ^ CW[10] ^ CW[11] ^ CW[12] ^ CW[13] ^ CW[14] ^ CW[15] = 0

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**

```
P5 = CW[16] =
CW[17] ^ CW[18] ^ CW[19] ^ CW[20] ^ CW[21] = 1
```

0	0000	**6**	**0110**	**12**	**1100**	**18**	**10010**
1	**0001**	**7**	**0111**	**13**	**1101**	**19**	**10011**
2	**0010**	**8**	**1000**	**14**	**1110**	**20**	**10100**
3	**0011**	**9**	**1001**	**15**	**1111**	**21**	**10101**
4	**0100**	**10**	**1010**	**16**	**10000**		
5	**0101**	**11**	**1011**	**17**	**10001**		

Hamming codes

Ipotizziamo che la codeword costruita sopra, ossia:

`0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 1 0 1 1 1 0`

venga inviata su un canale rumoroso. Il ricevitore legge:

`0 0 1 0 1 1 1 0 0 0 1 0 1 0 1 1 0 1 1 1 0`

il ricevitore ricalcola i check bit, questa volta comprendendo i check bit stessi nel calcolo (`CW` è la codeword letta dal ricevitore)

ricalcoliamo i check bit passo passo nelle prossime slide

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**

P_1 =

`CW[1] ^ CW[3] ^ CW[5] ^ CW[7] ^ CW[9] ^ CW[11] ^ CW[13] ^ CW[15] ^ CW[17] ^ CW[19] ^ CW[21] = 1`

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	1	0	1	1	1	0	0	0	1	0	1	0	1	1	0	1	1	1	0

P_2 =
CW[2] ^ CW[3] ^ CW[6] ^ CW[7] ^ CW[10] ^ CW[11] ^ CW[14] ^ CW[15] ^ CW[18] ^ CW[19] = 1

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	1	0	1	1	1	0	0	0	1	0	1	0	1	1	0	1	1	1	0

P_3 =
CW[4] ^ CW[5] ^ CW[6] ^ CW[7] ^ CW[12] ^ CW[13] ^ CW[14] ^ CW[15] ^ CW[20] ^ CW[21] = 0

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	1	0	1	1	1	0	0	0	1	0	1	0	1	1	0	1	1	1	0

P_4 =
CW[8] ^ CW[9] ^ CW[10] ^ CW[11] ^ CW[12] ^ CW[13] ^ CW[14] ^ CW[15] = 1

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	1	0	1	1	1	0	0	0	1	0	1	0	1	1	0	1	1	1	0

```
P5 =
CW[16] ^ CW[17] ^ CW[18] ^ CW[19] ^ CW[20] ^ CW[21] = 0
```

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

Riscriviamo i bit calcolati in ordine dall'ultimo al primo:

`0 1 0 1 1`

Interpretiamo la sequenza di bit come numero binario e convertiamolo in decimale. Questo numero è detto **error syndrome**:

$01011_{bin} = 11_{dec}$

Questo significa che il bit in posizione `11` è sbagliato. Il ricevitore lo inverte, correggendo così l'errore e recuperando il messaggio originale

Hamming codes

Ipotizziamo che la codeword costruita sopra, ossia:

```
0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 1 0 1 1 1 0
```

venga inviata su un canale rumoroso. Il ricevitore legge:

```
0 0 1 0 1 1 1 1 0 0 0 0 1 0 1 1 0 1 1 1 0
```

il ricevitore ricalcola i check bit, questa volta comprendendo i check bit stessi nel calcolo (`CW` è la codeword letta dal ricevitore)

ricalcoliamo i check bit passo passo nelle prossime slide

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	1	0	1	1	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0

```
P1 =
CW[1] ^ CW[3] ^ CW[5] ^ CW[7] ^ CW[9] ^ CW[11] ^ CW[13] ^ CW[15] ^ CW[17] ^ CW[19] ^ CW[21] = 0
```

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**

P_2 =
CW[2] ^ CW[3] ^ CW[6] ^ CW[7] ^ CW[10] ^ CW[11] ^ CW[14] ^ CW[15] ^ CW[18] ^ CW[19] = 0

0	0000	**6**	**0110**	**12**	**1100**	**18**	**10010**
1	**0001**	**7**	**0111**	**13**	**1101**	**19**	**10011**
2	**0010**	**8**	**1000**	**14**	**1110**	**20**	**10100**
3	**0011**	**9**	**1001**	**15**	**1111**	**21**	**10101**
4	**0100**	**10**	**1010**	**16**	**10000**		
5	**0101**	**11**	**1011**	**17**	**10001**		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	1	0	1	1	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0

P_3 =
```
CW[4] ^ CW[5] ^ CW[6] ^ CW[7] ^ CW[12] ^ CW[13] ^ CW[14] ^ CW[15] ^ CW[20] ^ CW[21] = 0
```

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**

```
P4 =
CW[8] ^ CW[9] ^ CW[10] ^ CW[11] ^ CW[12] ^ CW[13] ^ CW[14] ^ CW[15] = 1
```

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	1	0	1	1	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0

```
P5 =
CW[16] ^ CW[17] ^ CW[18] ^ CW[19] ^ CW[20] ^ CW[21] = 0
```

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

Riscriviamo i bit calcolati in ordine dall'ultimo al primo:

`0 1 0 0 0`

Interpretiamo la sequenza di bit come numero binario e convertiamolo in decimale. Questo numero è detto **error syndrome**:

$01000_{bin} = 8_{dec}$

Questo significa che il bit in posizione 8 è sbagliato. Il ricevitore lo inverte, correggendo così l'errore e recuperando il messaggio originale

Hamming codes

Ipotizziamo che la codeword costruita sopra, ossia:

`0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 1 0 1 1 1 0`

venga inviata su un canale rumoroso. Il ricevitore legge:

`0 0 1 0 1 1 1 1 0 0 0 0 1 0 1 1 0 1 0 1 0`

il ricevitore ricalcola i check bit, questa volta comprendendo i check bit stessi nel calcolo (`CW` è la codeword letta dal ricevitore)

ricalcoliamo i check bit passo passo nelle prossime slide

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	1	0	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0	1	0

```
P1 =
CW[1] ^ CW[3] ^ CW[5] ^ CW[7] ^ CW[9] ^ CW[11] ^ CW[13] ^ CW[15] ^ CW[17] ^ CW[19] ^ CW[21] = 1
```

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	1	0	1	1	1	0	0	0	0	0	1	0	1	1	0	1	0	1	0

P_2 =
CW[2] ^ CW[3] ^ CW[6] ^ CW[7] ^ CW[10] ^ CW[11] ^ CW[14] ^ CW[15] ^ CW[18] ^ CW[19] = 1

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**

P_3 =
`CW[4] ^ CW[5] ^ CW[6] ^ CW[7] ^ CW[12] ^ CW[13] ^ CW[14] ^ CW[15] ^ CW[20] ^ CW[21] = 0`

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**

```
P4 =
CW[8] ^ CW[9] ^ CW[10] ^ CW[11] ^ CW[12] ^ CW[13] ^ CW[14] ^ CW[15] = 0
```

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**

```
P5 =
CW[16] ^ CW[17] ^ CW[18] ^ CW[19] ^ CW[20] ^ CW[21] = 1
```

0	0000	6	0110	12	1100	18	10010
1	0001	7	0111	13	1101	19	10011
2	0010	8	1000	14	1110	20	10100
3	0011	9	1001	15	1111	21	10101
4	0100	10	1010	16	10000		
5	0101	11	1011	17	10001		

Hamming codes

Riscriviamo i bit calcolati in ordine dall'ultimo al primo:

`1 0 0 1 1`

Interpretiamo la sequenza di bit come numero binario e convertiamolo in decimale. Questo numero è detto **error syndrome**:

$$10011_{bin} = 19_{dec}$$

Questo significa che il bit in posizione `19` è sbagliato. Il ricevitore lo inverte, correggendo così l'errore e recuperando il messaggio originale

Hamming codes

Messaggio da trasmettere (rivedere l'esempio *meteorologico*):

`1`

m: 1

Il valore minimo di r è 2 ($m + r + 1 \ \leq 2^r$)

<u>**Hamming(3, 1)**: una *codeword* sarà lunga 3 bit, di cui 1 di dati e 2 di controllo (*check bits*)</u>

Assegniamo i valori corretti ai check bit utilizzando l'algoritmo di Hamming

Hamming codes

Inseriamo i bit del messaggio nelle scatolette il cui indice NON è una potenza di 2. I check bit andranno calcolati e inseriti nelle posizioni potenze di 2:

1	*2*	*3*
		1

Scriviamo gli indici in binario:

0	0000
1	**0001**
2	**0010**
3	**0011**
4	**0100**
5	**0101**

Hamming codes

1	2	3
1		**1**

```
c1 = CW[3] = 1
```

0	0000
1	0001
2	0010
3	0011

Hamming codes

1	*2*	*3*
1	**1**	**1**

```
c2 = CW[3] = 1
```

0	0000
1	**0001**
2	**0010**
3	**0011**

Hamming codes

Messaggio da trasmettere (rivedere l'esempio *meteorologico*):

`0`

m: 1

Assegniamo i valori corretti ai check bit utilizzando l'algoritmo di Hamming:

1	*2*	*3*
0		**0**

0	0000
1	**0001**
2	**0010**
3	**0011**

```
c1 = CW[3] = 0
```

Hamming codes

Messaggio da trasmettere (rivedere l'esempio *meteorologico*):

`0`

m: 1

Assegniamo i valori corretti ai check bit utilizzando l'algoritmo di Hamming:

1	*2*	*3*
0	**0**	**0**

0	0000
1	**0001**
2	**0010**
3	**0011**

`c2 = CW[3] = 0`

`111` e `000` sono proprio le codeword che avevamo trovato intuitivamente all'inizio della lezione

Hamming codes

Ipotizziamo che la codeword per il messaggio `1`, ossia:

```
1 1 1
```

venga inviata su un canale rumoroso. Il ricevitore legge:

```
1 0 1
```

il ricevitore ricalcola i check bit, questa volta comprendendo i check bit stessi nel calcolo (`CW` è la codeword letta dal ricevitore)

ricalcoliamo i check bit passo passo

Hamming codes

1	*2*	*3*
1	**0**	**1**

```
P₁ =
CW[1] ^ CW[3] = 0
```

0	0000
1	**0001**
2	**0010**
3	**0011**

Hamming codes

1	*2*	*3*
1	**0**	**1**

P_2 =
CW[2] ^ CW[3] = 1

0	0000
1	**0001**
2	**0010**
3	**0011**

Hamming codes

Riscriviamo i bit calcolati in ordine dall'ultimo al primo:

1 0

Interpretiamo la sequenza di bit come numero binario e convertiamolo in decimale. Questo numero è detto **error syndrome**:

$$1\,0_{\text{bin}} = 2_{\text{dec}}$$

Questo significa che il bit in posizione 2 è sbagliato. Il ricevitore lo inverte, correggendo così l'errore e recuperando il messaggio originale

Da vedere/leggere/visitare a casa

- Error Correction – Computerphile
- Error Detection and Flipping the Bits - Computerphile