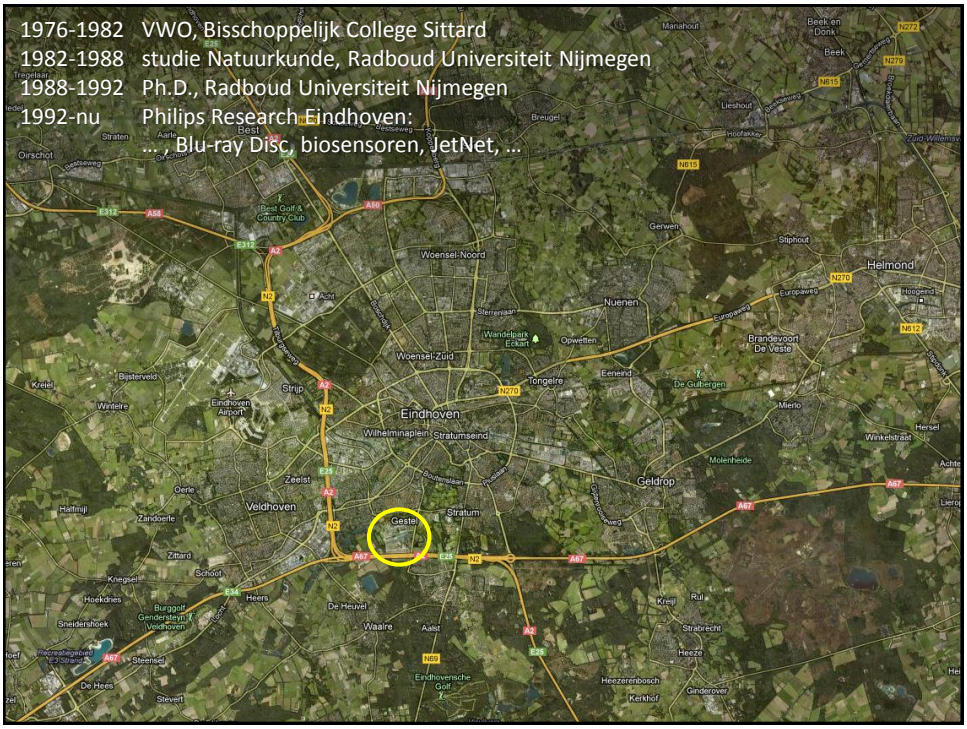
Blu-ray Disc en Wiskunde

De Magie van Fout Correctie

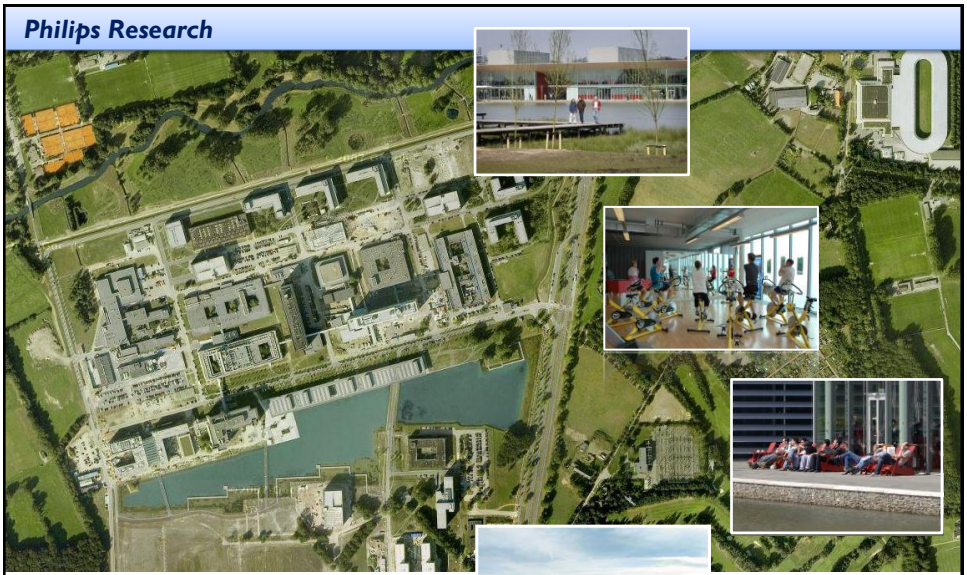
Dr. Jean Schleipen
Philips Research Eindhoven

CONTENT

- **Optical Data Storage – the Basic Principles**
  - From Audio to Bits
  - From CD to DVD to Blu-ray Disc
- **Information Theory and Coding Languages**
  - Sender, Communication Channel and Receiver
  - Letters, Codewords and Alphabets
- **Error Detection & Correction**
  - The Human Brain
  - Parity Check
  - Hamming Code
  - Reed Solomon Code
- **Philips JetNet - DVD Demonstrator model**



Philips Research



High Tech Campus Eindhoven
The Netherlands

Campus houses 25+ companies / knowledge institutes
More than 50 nationalities
8000 m² clean room facilities, 50.000 m² lab-space

Philips Research

Philips Research

Philips Research - Our People

- Physicists
- Chemists
- Biologists & Biochemists
- Medical doctors
- Electrotechnical eng.
- Mathematicians
- Construction eng.
- IT-specialists

Collaborations with

- External partners
- Universities
- Hospitals

- Gastlessen
- Profielwerkstukken
- Bedrijfsbezoeken

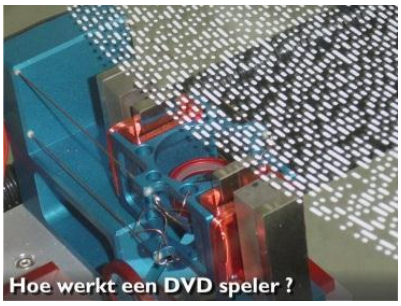
PHILIPS "Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

Philips Research

Philips JetNet - DVD Demonstration Model

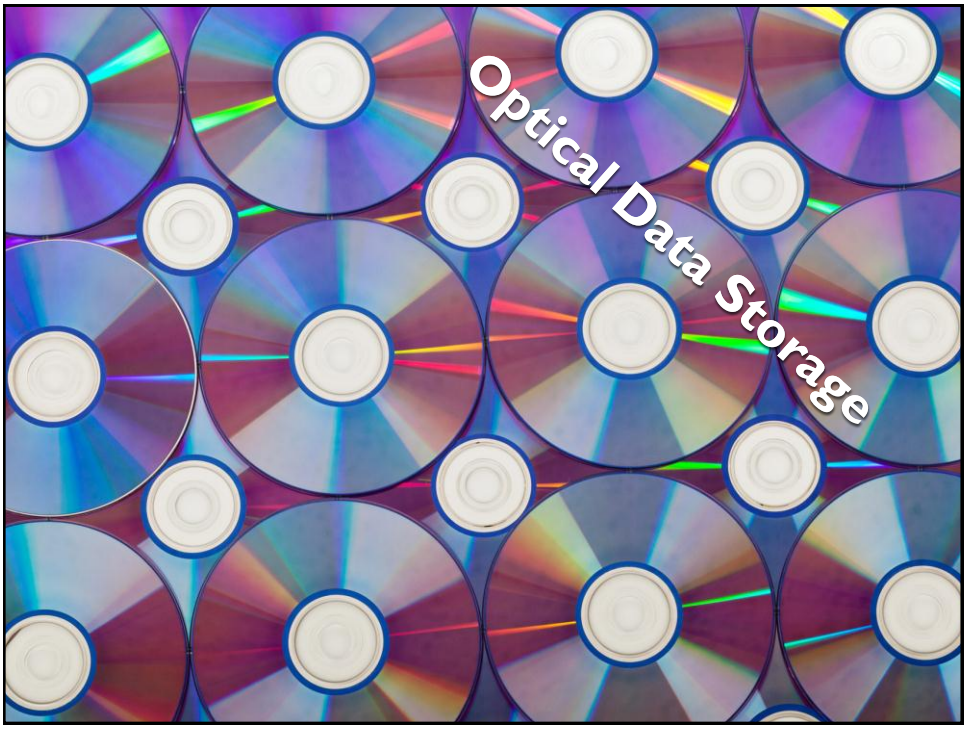
A 10:1 functional scale model of a DVD player, to illustrate the multi-disciplinary character of optical data storage.

- Physics: optics, diffraction, lasers and Lorentz force
- Chemistry: phase change materials, disc replication
- Mathematics: data coding, error correction
- …



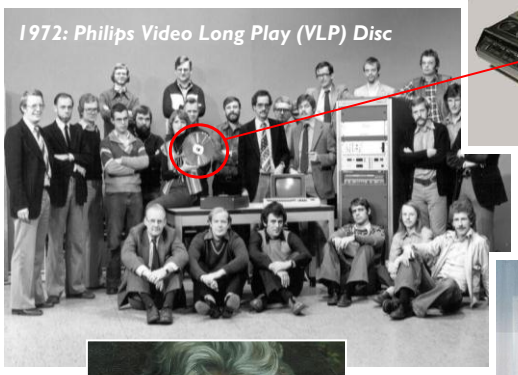
DVD Demonstrator setup
Booklet for students and teacher
Video instructions on DVD

PHILIPS "Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

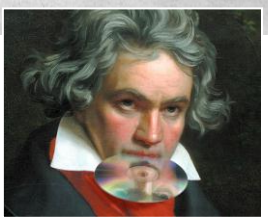


Introduction Optical Data Storage

History



1972: VLP
30 minutes video on a
30 cm diameter optical disc



COMPACT disc DIGITAL AUDIO

1982: Compact Disc

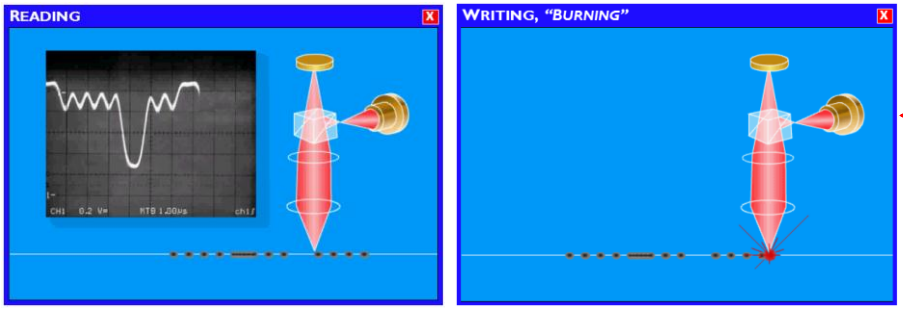
PHILIPS

"Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

Introduction Optical Data Storage

Scanning confocal microscope

An objective lens ***focuses*** a laser beam into a tiny spot on an optical disc, and ***collects*** the scattered light from the pits on the disc.

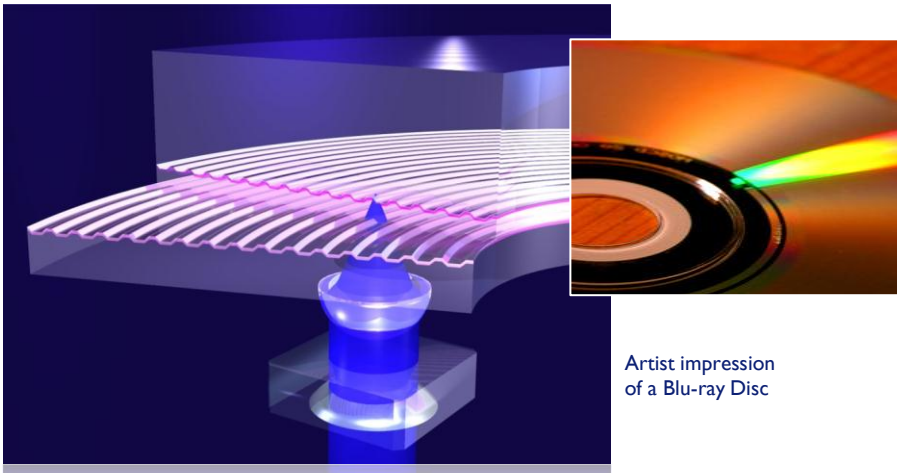


Reading: Reflected light from disc hits detector:
pit = high signal "1", no pit = low signal "0"

Writing: A binary data stream is send to laser:
"0" means low laser power; "1" means high laser power, burning data into disc

Introduction Optical Data Storage

The optical disc

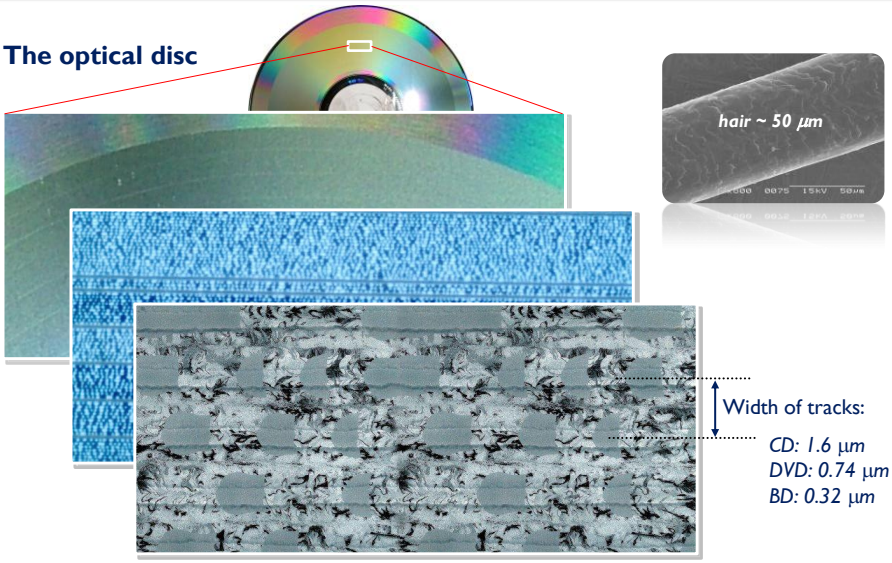


Artist impression
of a Blu-ray Disc

An optical disc contains a spiraling ***groove***, along which the binary data are recorded.
This groove is needed for ***tracking*** purposes.

Introduction Optical Data Storage

The optical disc



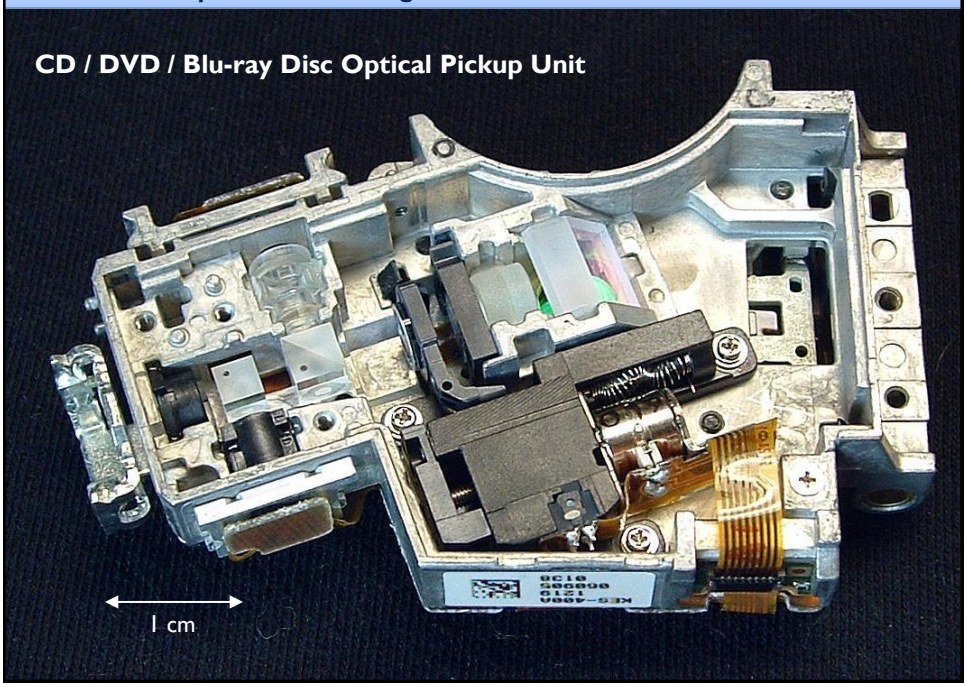
A dark area (amorphous phase) represents the "0's";
The lighter areas in between (crystalline phase) correspond to the "1's"
(re-writable disc)

PHILIPS

"Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

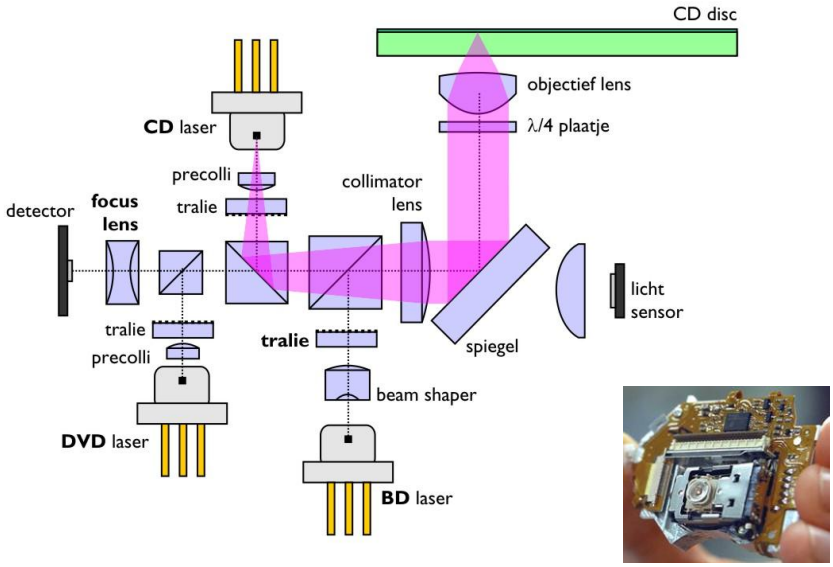
Introduction Optical Data Storage

CD / DVD / Blu-ray Disc Optical Pickup Unit



Introduction Optical Data Storage

Blu-ray Disc optical pickup unit

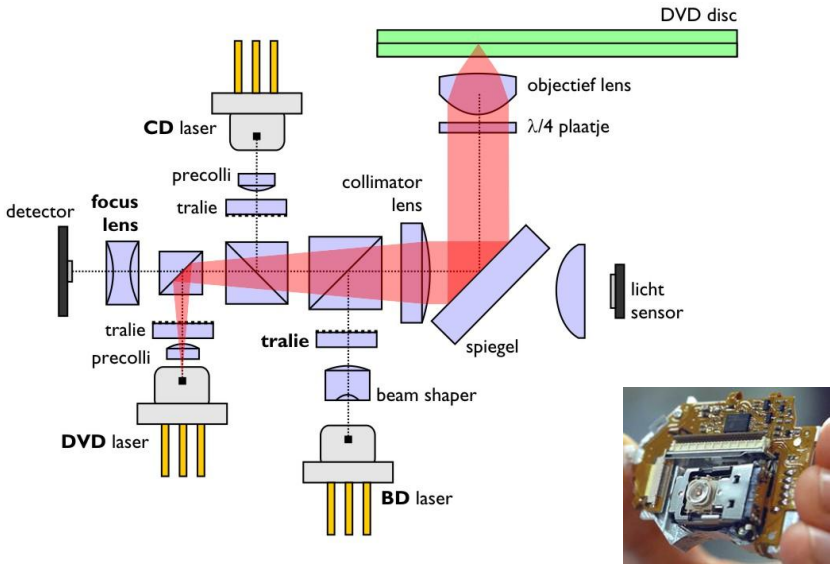


PHILIPS

"Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

Introduction Optical Data Storage

Blu-ray Disc optical pickup unit

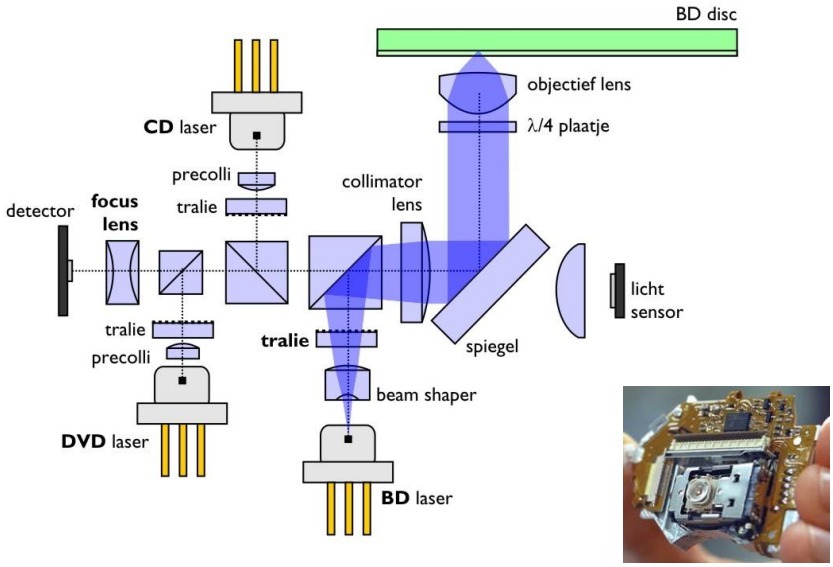


PHILIPS

"Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

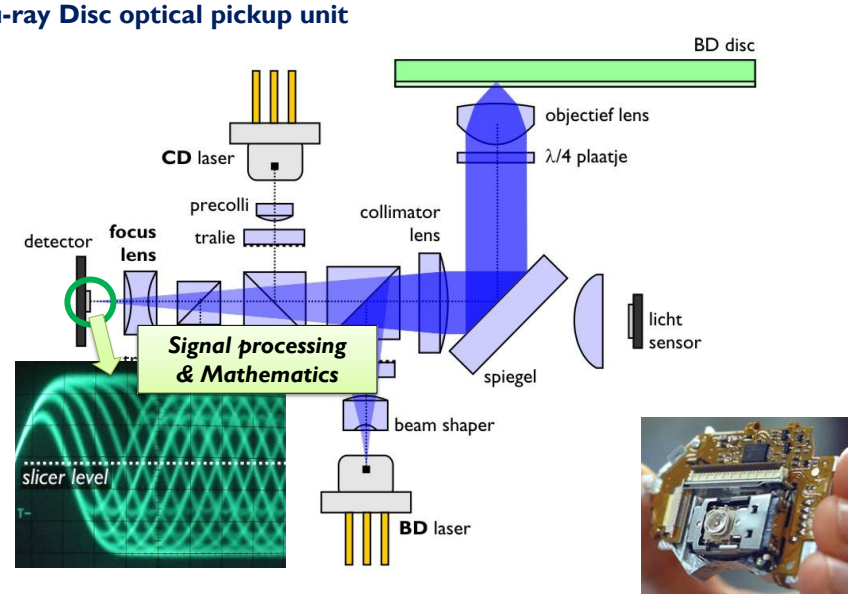
Introduction Optical Data Storage

Blu-ray Disc optical pickup unit



Introduction Optical Data Storage

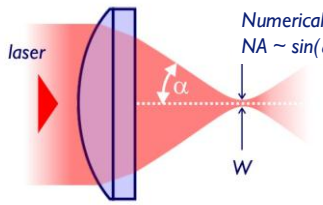
Blu-ray Disc optical pickup unit



Introduction Optical Data Storage

Diffraction and storage capacity

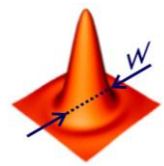
The storage capacity of an optical disc is mainly determined by diffraction of light.
The smallest effect that can be read is determined by the size of the laser spot.

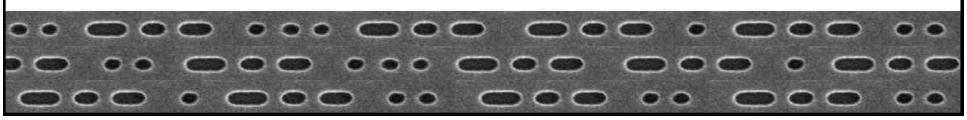


$$NA = \frac{1}{2 \cdot f\#}$$

…human vision
photography
microscopes
telescopes
lithography (IC's) …

Size of laser spot: $w = 1.22 \frac{\lambda}{NA}$

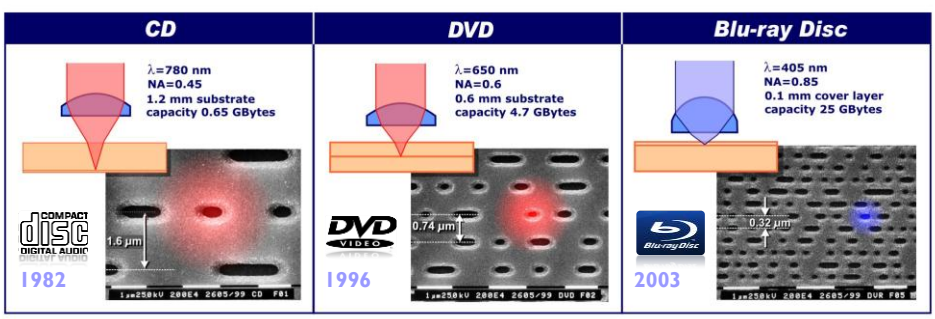




Introduction Optical Data Storage

From CD to Blu-ray Disc

The 3 generations:



650 MBytes | **4.7 GBytes** | **50 GBytes** 2-layer

PHILIPS

"Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

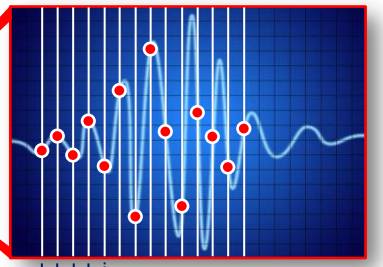
Introduction Optical Data Storage

From the Analog to the Digital Domain

Real world content = analog !
Audio (music, speech),
Photography & Video,
Written communication, ...

E.g. sound:

At each time interval T_S a new sample is taken



0.42 V = 00110110 01101011 (16 bits)
0.67 V = 01110110 11101001
0.38 V = 00110000 01001101
0.75 V = 01111101 10011100
...

Nyquist sampling theorem:

$f_S = 1/T_S = 2 \times f_{max}$
For humans: $f_{max} \sim 20$ kHz

For CD: $\mathbf{f_S = 44.1}$ **kHz**

Introduction Optical Data Storage

Q: How many bits needed for 1 hr of music ?

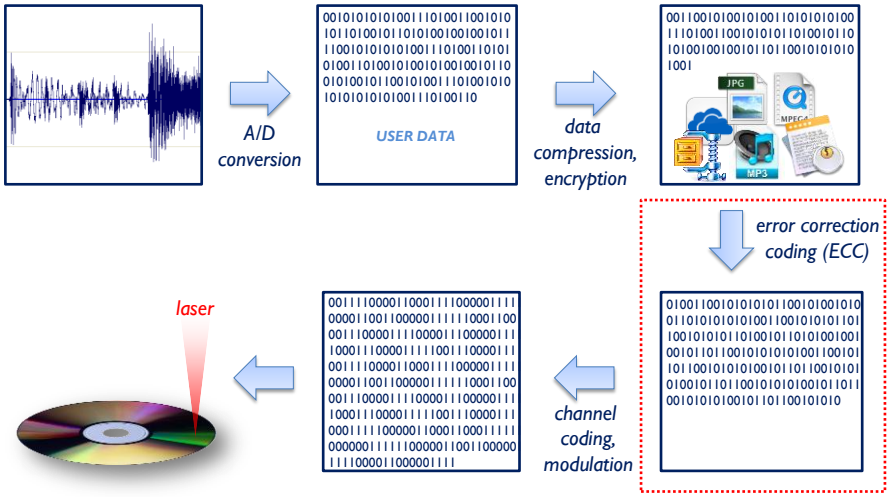
1 hr. Music	*= 60 min. = 3600 sec.*
Sample rate	*= 44100 samples / sec.*
Sampling depth	*= 16 bits / sample*
No. Channels (stereo)	*= 2*
	x
	= 5.08 x 10^9 bits
	= 635 MBytes

Q: How many bits needed for 1 hr of video ?

1 hr. Video	*= 60 min. = 3600 sec.*
Frame rate	*= 25 frames / sec.*
No. Pixels per frame	*= 720 x 576 pixels / frame*
No. Colors / pixel	*= 3 (RGB)*
Color depth	*= 8 bits / pixel*
	x
	= 895.8 x 10^9 bits
	= 112 GBytes

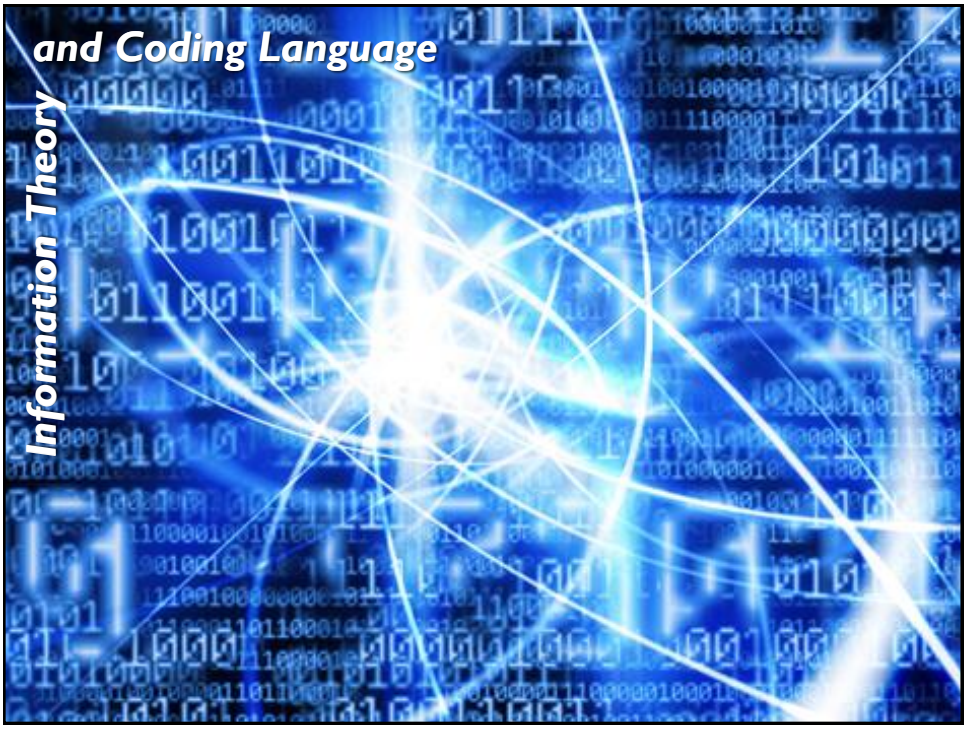
Introduction Optical Data Storage

From the Analog to the Digital Domain



PHILIPS

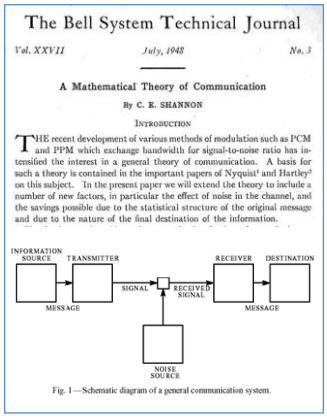
"Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research



Information Theory and Coding Language

Claude Elwood Shannon
1916-2001

American mathematician and electronic engineer
MIT, Bell Labs
"*The Father of Information Theory*"



The Bell System Technical Journal

Vol. XXVII *July, 1948* *No. 3*

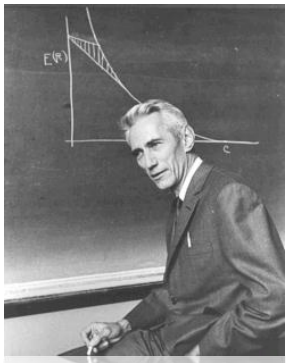
A Mathematical Theory of Communication

By C. E. SHANNON

INTRODUCTION

THE recent development of various methods of modulation such as PCM and PPM which exchange bandwidth for signal-to-noise ratio has intensified the interest in a general theory of communication. A basis for such a theory is contained in the important papers of Nyquist[1] and Hartley[2] on this subject. In the present paper we will extend the theory to include a number of new factors, in particular the effect of noise in the channel, and the savings possible due to the statistical structure of the original message and due to the nature of the final destination of the information.

Fig. 1 — Schematic diagram of a general communication system.

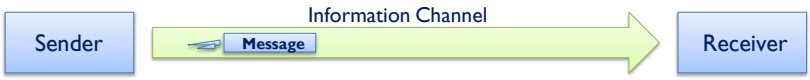


The Shannon Theorem:

"Given a noisy communication channel, we can compute a maximum channel ***capacity C*** (in bits/sec): then for any communication rate ***R*** < ***C***, ***coding schemes*** exist such that the error rate at the receiver becomes arbitrarily small."

Information Theory and Coding Language

Sender, Communication Channel and Receiver

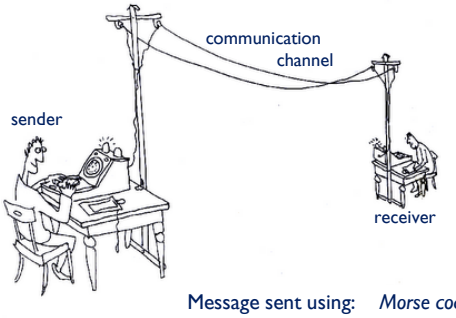


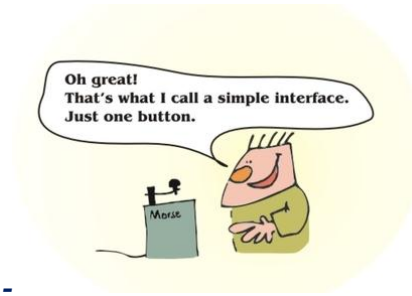
It's from your wife, she says 'Get take-away… diner is burnt…!'

Sender:	*wife*
Message:	*you'd better not come home …*
Receiver:	*husband*
Information channel:	*smoke signals*

Information Theory and Coding Language

Sender, Communication Channel and Receiver





Message sent using: *Morse code*
Alphabet: *dot • , dash –*
Information channel: *electric pulses over a copper wire*

PHILIPS "Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

Information Theory and Coding Language

Sender, Communication Channel and Receiver

Message:

- radio & TV
- telephony
- fax
- audio, speech
- video
- internet
- e-mail
- GPS
- telebanking
- cloud storage

…

Physical channel:

- electrical pulses over copper wire, computer interfaces
- optical signals over glass fiber
- electromagnetic waves through air (3G, 4G, satellite, …)
- optical disc readout (CD, DVD, Blu-ray)
- magnetic tape recording
- harddisc recording (magnetic)
- barcode reading

…

PHILIPS "Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

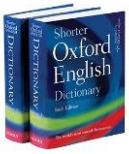
Information Theory and Coding Language

Letters, Codewords and Alphabets

In sending information across a channel, the receiver needs to know ***what signals*** to expect, and ***how to interpret*** these signals in order to extract the information.

For inter-human communication we agreed to use a standard language that we all understand:

Code = a certain ***language***
e.g. English, Dutch, French, Italian, …
Codewords = all the ***words*** in the vocabulary belonging to that language
Letters = all the ***characters*** making up the different words; the collection of all letters is called the ***alphabet***



In the digital domain, for transporting data from A to B:

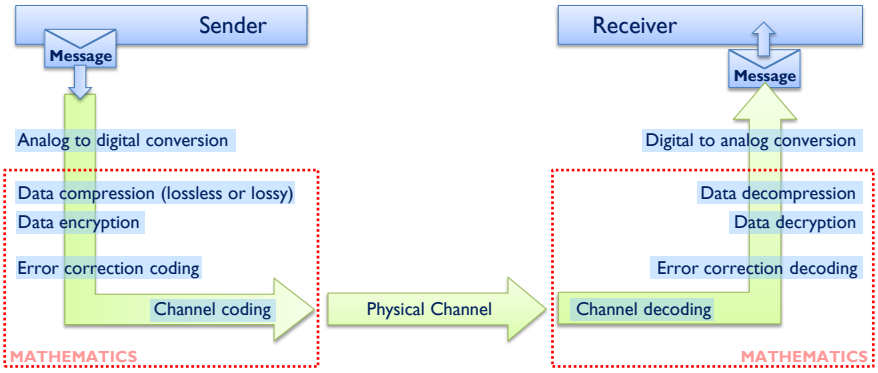
Code = different codes are being used,
e.g. 802.11 for WLAN, Reed-Solomon for error correction, …
Codewords = all the words, consisting of bits or symbols, being part of the code
Letters = ***bits*** or ***symbols;*** ***alphabet = {0,1}, or 'field'***

IEEE 802.11 standard

Information Theory and Coding Language

Different Types of Codes

The purpose of an information channel is to send information from sender to receiver
- as ***fast*** as possible (short waiting times),
- as ***accurate*** as possible (no errors and misunderstandings),
- as ***secure*** as possible (no cheating, no abuse)
- as ***condensed*** as possible (high storage capacity)



Information Theory and Coding Language

- **Error Detection**
- **Error Correction – Hamming codes**
- **Error Correction – Reed Solomon codes**

Error Detection

Noisy Information Channel

In real world all communication channels either contain some ***noise***, or have very ***low signal*** levels. As a result ***errors occur*** when transmitting messages from sender to receiver.

Q: - Can we detect a possible error ?
- And if so, can we correct it ?

Error Detection

The Human Brain

What about this mess … ?

Can you read this? Only 55 people out of 100 can:

I tdnuolc elveieb taht I dluoc yulaclta desdnatnru tahw I saw gdanier. Eht lhaonmneap rweop fo eht nuah dnim, goccdrnia ot a hcheearcr ta Emabrigdc Yinervtisu, ti tseno'd rtaetm ni tahw rerdo eht stterel ni a drow era, eht ylno tproamtni gihnt si taht eht trsif dna tsal rtteel eb ni eht tghir eclap. Eht tser nac eb a laott ssem dna uoy nac litls daer ti thotuiw a mboerlp. Siht si ecuseab eht nuamh dnim seod ton daer yrvee rtetel yb fstlei, tub eht drow sa a elohw. Gzanmia huh? Haey dna I swlyaa tghuhot glpelins saw tpmoranti!

Error Detection

The Human Brain

And now (first and last letter of each word at the right position) … ?

Can you read this? Only 55 people out of 100 can:

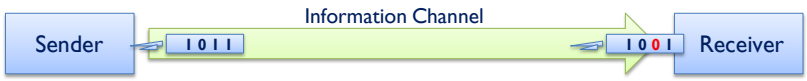
I cdnuolt blveiee taht I cluod aulaclty uesdnatnrd waht I was rdanieg. The phaonmneal pweor of the hmuan mnid, aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it dseno't mtaetr in waht oerdr the ltteres in a wrod are, the olny iproamtnt tihng is taht the frsit and lsat ltteer be in the rghit pclae. The rset can be a taotl mses and you can sitll raed it whotuit a pboerlm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe. Azanmig huh? yaeh and I awlyas tghuhot slpeling was ipmorantt!

Error Detection

Adding Redundancy – Parity or Control Bits

Suppose we want to transmit a message over a binary channel with alphabet {0,1}.
Let's take as message to send the numbers 0 … 15, represented by their binary equivalent;
a user data word (the user message) is then 4 bits long, e.g. (0100) or (1101) ...

Suppose we want to send over the user message (1011) and before the message reaches the receiver, an error occurs and a single bit flips: the receiver now reads (1001) in stead of (1011).



Q: Is there a way to predict whether an error has been made ?

A: No, since all received words, irrespective of any error that might have occured, are perfectly ***allowed codewords***.

In order to detect an error we are going to add ***redundancy*** to the user message. These additional ***parity*** bits allow the receiver to ***distinguish between valid and an invalid messages*** (codewords).

Single Parity Check

Adding Redundancy – Single Parity Check

In order to detect an error we are going to add ***redundancy*** to the user message. These additional bits, called ***parity bits***, have been chosen carefully such that after the (user message + parity bits) arrive at the detector, the receiver is able to judge whether an error has occured.

Now, before we send over the user message (1011), an additional parity bit **c** is added such that the total number of '1's in the message, i.e. the ***overall parity, is even***.
Or, in other words: the sum of all bits – modulo 2 (mod_2) is zero.

User message: (1011)
Code message: (1011**c**) and (1+0+1+1+**c**) $mod_2 = 0 \Rightarrow$ **c = 1**
hence, message = codeword = (10111)

Q: How to check whether an error has occured ?

A: Check the overall parity of the received message; if 1 then error, if 0 then no error

Q: Can we detect all errors ?

A: No, e.g. if two bits would have flipped, the overall parity would still be even;
with single parity check we can detect only 1 bit error.

Single Parity Check

Adding Redundancy – Single Parity Check

In general we can add a parity bit to a user data word (u_m, u_{m-1},…, u_2, u_1) with an arbitrary length, say m bits. The codeword that is sent to the receiver is then:

Codeword: (u_m, u_{m-1}, … , u_2, u_1, **c**) ; the codeword now has length $n = m + 1$

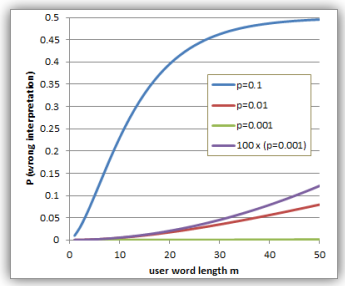
Let's assume that some white noise is added to the channel; as a result the overal probability of a single bit error becomes p. That is to say: on average one in every $1/p$ bits is detected wrong.

Q: What is the probability that a codeword is interpreted as being correct, but is actually wrong?

A:
- 0 errors: OK, no error detected
- 1 error: error correctly detected
- 2 errors: errors not detected
- 3 errors: errors correctly detected
- 4 errors: errors not detected
- …

$$P(n,p) = \binom{n}{2} \cdot p^2 \cdot (1-p)^{n-2} + \binom{n}{4} \cdot p^4 \cdot (1-p)^{n-4} + \ldots$$

$$= \sum_{i=1}^{n/2} \binom{n}{2i} \cdot p^{2i} \cdot (1-p)^{n-2i}$$



Single Parity Check

Adding Redundancy – Single Parity Check

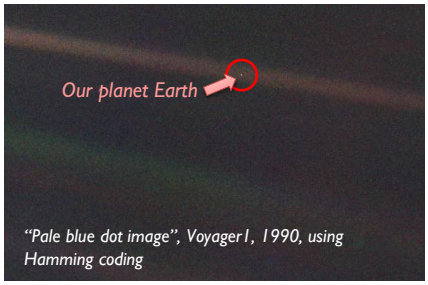
Once an error has been detected, the receiver might send back a signal to the sender with a request to resend the message. This is called ***Automatic Repeat Request*** (ARQ).

Q: Try to think of a situation where ARQ is not feasible …

A: Life communication: Listening to a CD, watching a DVD, TV broadcasting, Skype, …

A: Satellite communication; actually one of the first real applications of error correction coding was in the Voyager program, sending a satellite to the outer rim of our solar system.

Voyager1, launched by NASA in 1977, sent an image back to earth when it was 6 billion km separated from the earth. It took the signals approx. 6 hrs. to travel to earth at the speed of light. Sending an ARQ would again take 6 hrs. to arrive at the spacecraft. In total the spacecraft would have travelled (at a speed of 64000 km/hr.) a distance of 770000 km in this time span of 12 hrs.



"Pale blue dot image", Voyager1, 1990, using Hamming coding

2-Dimensional Parity Check

Adding Redundancy – 2-Dimensional Parity Check

We can do a bit better than single parity check, by using ***2-dimensional*** code checking.

Suppose the user want to send the message …0110 0010 1011 0001 …
The message is split in blocks of e.g. 4 bits each, and e.g. 4 blocks of data are put in a matrix.
Now, for each row and each column a new parity bit is calculated.

User message: (0110) (0010) (1011) (0001)

2D-Parity matrix:

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

Codeword: (01100) (00101) (10111) (00011)(11101)

Now, suppose a ***burst error*** of 3 successive bits occurs, and the receiver reads:

Codeword: (01100) (11001) (10111) (00011)(11101)

2-Dimensional Parity Check

Adding Redundancy – 2-Dimensional Parity Check

So the receiver retrieves the following signal:

Codeword: (01100) (11001) (10111) (00011) (11101)

Next, we put these bits at the corresponsing position in the parity matrix and check the parities:

Parity matrix:

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

Now, after error detection we also know the position of the erroneous bits, meaning that we can also ***correct the detected error(s)*** ! The corrected codeword, originally sent by the sender must have been:

Codeword: (01100) (00101) (10111) (00011)(11101)

and the original user message: (0110) (0010) (1011) (0001)

2-Dimensional Parity Check

Adding Redundancy – 2-Dimensional Parity Check

The 2-dimensional parity check is a first step towards ***Forward Error Correction*** (FEC), but it is rather crappy … !

Q: Why is the 2-dimensional parity check not a very robust method ?

A: A single bit error can always be detected and corrected.

For 2 or more bit errors, depending on ***where*** the bit errors occurs, the 2D parity check ***may*** be able to detect multiple errors, but also can lead to erroneous decoding !

Suppose the following string of bits was read by the receiver (again 3 bit errors, different positions):

Codeword: (01101) (11101) (10111) (00011)(11101)

Parity matrix:

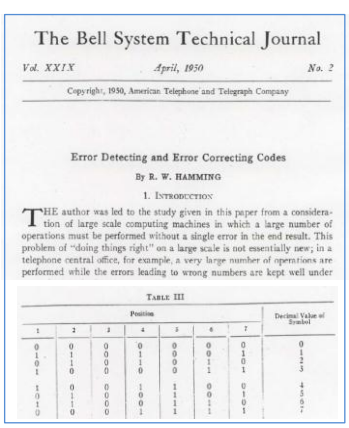
0	1	1	0	1
1	1	1	0	1
1	0	1	1	1
0	0	0	1	1
1	1	1	0	1

Error correction seams to be feasible by adding parity bits, but a more robust method is needed !

Error Correction

Richard Wesley Hamming
1915-1998

American mathematician
Manhattan project, Bell Labs, colleague of Shannon
"*The Father of ECC Theory*"



The Bell System Technical Journal

Vol. XXIX *April, 1950* *No. 2*

Copyright, 1950, American Telephone and Telegraph Company

Error Detecting and Error Correcting Codes

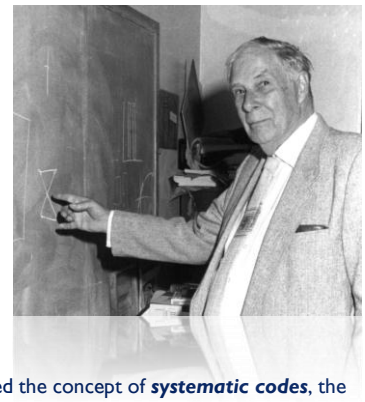
By R. W. HAMMING

1. INTRODUCTION

THE author was led to the study given in this paper from a consideration of large scale computing machines in which a large number of operations must be performed without a single error in the end result. This problem of "doing things right" on a large scale is not essentially new; in a telephone central office, for example, a very large number of operations are performed while the errors leading to wrong numbers are kept well under

TABLE III

Position							Decimal Value of Symbol
1	2	3	4	5	6	7	
0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	1
0	1	0	1	0	1	0	2
1	0	0	0	0	1	1	3
1	0	0	1	1	0	0	4
0	1	0	0	1	0	1	5
1	1	0	0	1	1	0	6
0	0	0	1	1	1	1	7



Introduced the concept of ***systematic codes***, the ***Hamming distance*** and Hamming bound.

Triggered a vast R&D effort on error correction coding leading to new coding theories and opening up new application spaces.

The Hamming Code

The Hamming Distance

Hamming was the first to describe the process of error detection and correction using a mathematical model. His theory is based on the concept of ***Hamming distance*** d_H: the number of bits one needs to flip in order to go from one to another codeword.

Example:

(10)	⟹	(11)	: Hamming distance $d_H = 1$
(11010)	⟹	(01101)	: Hamming distance $d_H = 4$
(11001100)	⟹	(11110000)	: Hamming distance $d_H = 4$

The Hamming distance d_H between two codewords w_1 and w_2 can be calculated by bit-wise mod$_2$-subtraction or XOR operation between the two codewords, and adding the individual bits.

$$d_{\text{Hamming}}(w_1, w_2) = d_H(w_1, w_2) \equiv \sum_{all\,bits\,i} XOR(w_1, w_2)_i$$

The ***Hamming distance*** d_H between a received word and the originally sent codeword is the ***number of bit errors*** that occured during transmission !

The Hamming Code

The Hamming Distance

Q: Calculate the Hamming distance between the original codeword w_1 and the received word w_2:

- code length n=4: w_1 = (1 0 0 1) and w_2 = (1 0 1 0)
- code length n=4: w_1 = (1 1 0 1 0 1 1 0 0 1 0 1 0 0 1 0 1) and
  w_2 = (1 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0)

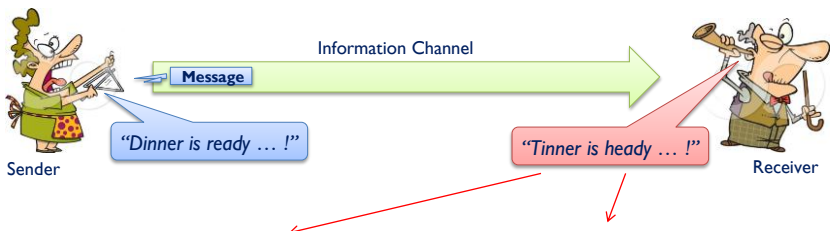
Example: w_1 = (1 0 0 1)
w_2 = (1 0 1 0)
XOR(w_1,w_2) = (0 0 1 1) ⟹ d_H = sum of bits = 2

Example: w_1 = (1 1 0 1 0 1 1 0 0 1 0 1 0 0 1 0 1)
w_2 = (1 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0)
XOR(w_1,w_2) = (0 1 0 1 1 1 0 0 1 0 0 1 0 1 1 0 1) ⟹ d_H = sum of bits = 9

The Hamming Code

The Hamming Distance

Actually, we human beings also use the Hamming distance between regular words "part of our code" in order to do error correction.



Valid words similar to "***Tinner***"

Similar words	Hamming distance
Tinker	1
Dinner	1
Tunnel	2
Banker	3

Valid words similar to "***heady***"

Similar words	Hamming distance
heavy	1
ready	1
handy	2
happy	3

The Hamming Code

The Hamming Distance

Now, back to the binary world. We are going to construct a code ***W***, using binary letters $b_i = 0$ and 1, and using codewords of length n. The codewords $\boldsymbol{w} \in \boldsymbol{W}$ consist of a user word $\boldsymbol{u}$ of length m, and k ***control*** or ***parity*** bits $\boldsymbol{c}$; consequently $n = m + k$. n is called the ***code length***.

$\boldsymbol{W} = \{ \boldsymbol{w} = (b_1, b_2, \ldots, b_n) \; ; \; b_i = (0,1), i = 1 \ldots n \}$; $\boldsymbol{w} = (b_1, b_2, \ldots, b_n) = (u_1, \ldots, u_m, c_1, \ldots, c_k)$

Example: the user words consist of 4 bits (m=4): $\boldsymbol{u} = (u_1, u_2, u_3, u_4) = (0110)$,
we add 3 parity bits (k=3): $\boldsymbol{c} = (c_1, c_2, c_3) = (110)$,
resulting in codewords of length $n = m + k = 7$: $\boldsymbol{w} = (0110\,110)$

Q: How many different codewords can we make for a code length of $n = 7$?

A: The total number of codewords of length n, using binary letters, is: 2^n , i.e. $2^7 = 128$

We define the ***code rate*** or the efficiency of the code as: $R = \frac{m}{n} = \frac{1}{1 + k/m}$

The higher the code rate, the more efficient the code is.

The Hamming Code

The Hamming Distance

Let us reconsider the ***single parity check*** case with user words of length 3: m=3, k=1; code length n=4.

In total we have 2^4=16 codewords available, out of which 8 have even parity and 8 have odd parity. Let's define the even codewords as $\boldsymbol{W_1}$ and the odd ones as $\boldsymbol{W_2}$, then $\boldsymbol{W_1}$ and $\boldsymbol{W_2}$ are subsets of $\boldsymbol{W} = \boldsymbol{W_1} + \boldsymbol{W_2}$. Codewords from $\boldsymbol{W_1}$ are ***valid*** codewords (even parity), codewords from $\boldsymbol{W_2}$ are ***invalid*** (odd parity).

W_1	W_2
0 0 0 0	0 0 0 1
0 0 1 1	0 0 1 0
0 1 0 1	0 1 0 0
0 1 1 0	0 1 1 1
1 0 0 1	1 0 0 0
1 0 1 0	1 0 1 1
1 1 0 0	1 1 0 1
1 1 1 1	1 1 1 0

W

The sender only transmits words $\boldsymbol{w} \in \boldsymbol{W_1}$.

Suppose a single bit error occurs: then the receiver ***detects*** a word $\boldsymbol{w} \in \boldsymbol{W_2}$ and knows that an error has occured.

The receiver is ***not capable of correcting*** the error, since it does not know which bit is likely being detected wrong !

Reason:
The Hamming distance between all codewords in $\boldsymbol{W_1}$ is 2 or larger !
Or: the ***minimum Hamming distance*** for the parity check code is 2.

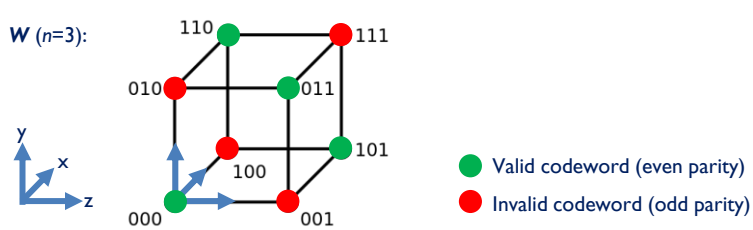
The Hamming Code

The Hamming Distance

Let us try to visualize all codewords of $\boldsymbol{W}=(b_1,b_2,b_3,b_4)$

In three dimensions, $\boldsymbol{W}=(b_1,b_2,b_3)$, i.e. n=3, this would be easy: the 8 points spanned by the **x**, **y**, and **z** unit vectors (x,y,z) in 3D Carthesian coordinates:

$\boldsymbol{W}$ (n=3):



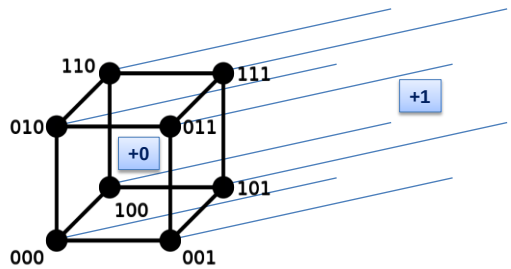
Q: Can we also visualize the ***fourth dimension*** ?

A: Yes, since we are only using binary numbers …

The Hamming Code

The Hamming Distance

Let us try to visualize all codewords of $\boldsymbol{W}=(b_1,b_2,b_3,b_4)$



Q: Can we also visualize the ***fourth dimension*** ?

A: Yes, since we are only using binary numbers …

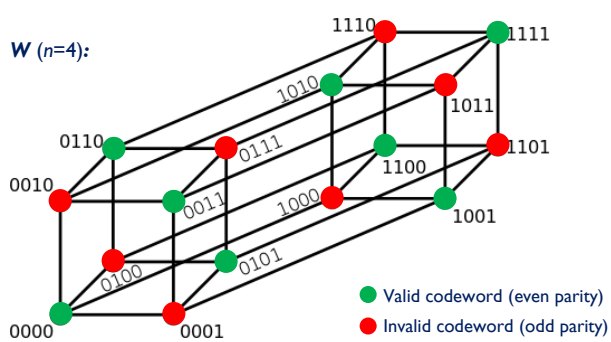
The Hamming Code

The Hamming Distance

Now back to code length n=4, let us try to visualize all codewords of $\boldsymbol{W}=(b_1,b_2,b_3,b_4)$

W_1	W_2
0 0 0 0	0 0 0 1
0 0 1 1	0 0 1 0
0 1 0 1	0 1 0 0
0 1 1 0	0 1 1 1
1 0 0 1	1 0 0 0
1 0 1 0	1 0 1 1
1 1 0 0	1 1 0 1
1 1 1 1	1 1 1 0

$\boldsymbol{W}$



Whenever a wrong codeword $\boldsymbol{w_2} \in \boldsymbol{W_2}$ is detected, there are always ***more than one*** valid codewords $\boldsymbol{w_1} \in \boldsymbol{W_1}$ (in this case 4) with Hamming distance d_H=1, closest to $\boldsymbol{w_2}$. Hence, the detector can by no means decide which codeword would have been the correct one.

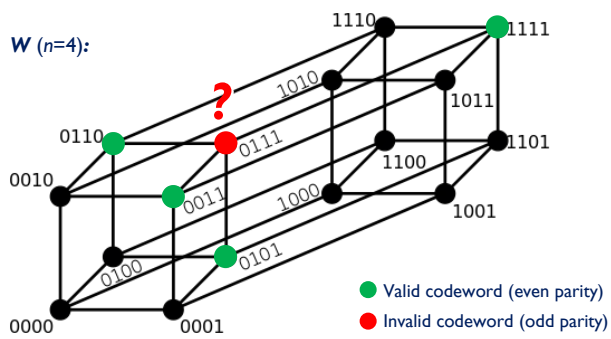
The Hamming Code

The Hamming Distance

All codewords of $W=(b_1,b_2,b_3,b_4)$:

W_1	W_2
0 0 0 0	0 0 0 1
0 0 1 1	0 0 1 0
0 1 0 1	0 1 0 0
0 1 1 0	0 1 1 1
1 0 0 1	1 0 0 0
1 0 1 0	1 0 1 1
1 1 0 0	1 1 0 1
1 1 1 1	1 1 1 0

W



Suppose we receive the codeword (0111). Since the parity is odd, we know an error occured. There exist 4 valid codewords with a minimum Hamming distance of 1, closest to (0111). It is by no means evident which of these is the correct one!

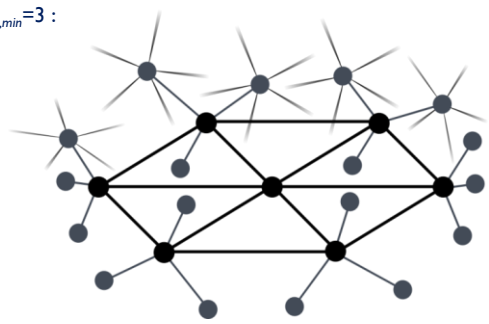
The Hamming Code

The Hamming Distance

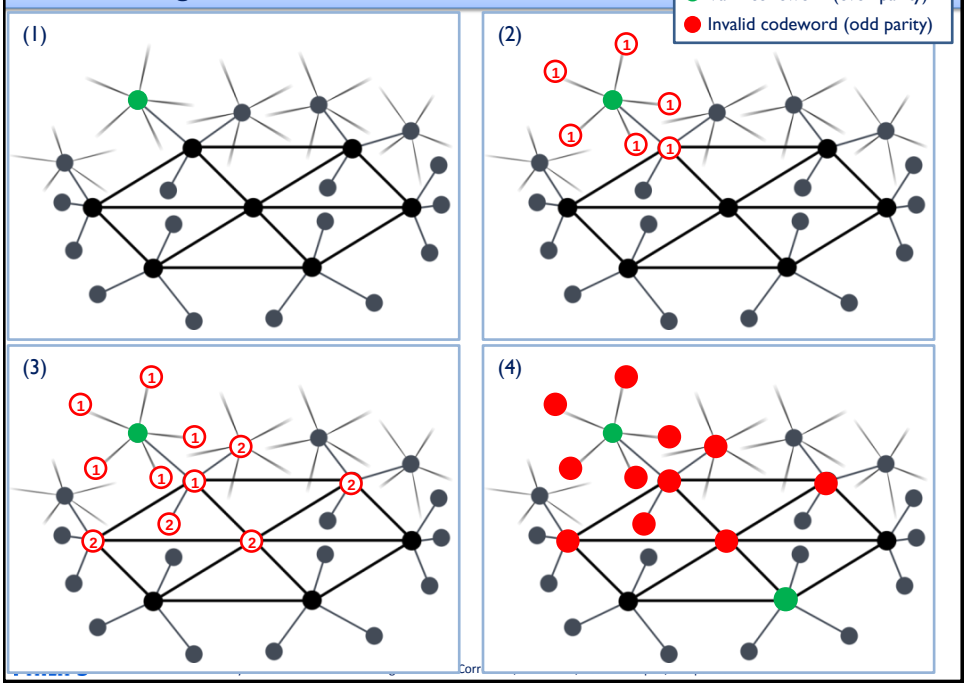
Q: Is there a way to determine which of the ***nearest*** valid codewords is the correct one ?

A: Yes: define a subset of valid codewords $W_1 \in W$ such that the ***minimum Hamming distance*** between all valid codewords is ***3*** (or more) ! The error checking method requires a more sophisticated parity checking algorithm (next slides).

W (n=6), $d_{H,min}$=3 :



The Hamming Code



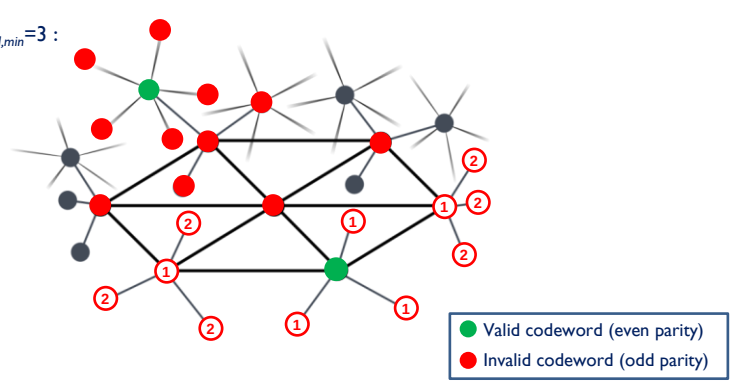
The Hamming Code

The Hamming Distance

Q: Is there a way to determine which of the ***nearest*** valid codewords is the correct one ?

A: Yes: define a subset of valid codewords $W_I \in W$ such that the ***minimum Hamming distance*** between all codewords is **3** (or more) ! The error checking method requires a more sophisticated parity checking algorithm (next slides).

W (n=6), $d_{H,min}$=3 :



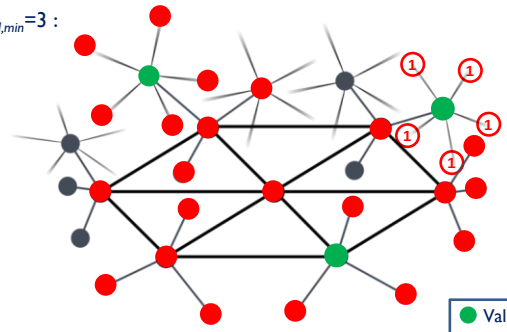
The Hamming Code

The Hamming Distance

Q: Is there a way to determine which of the ***nearest*** valid codewords is the correct one ?

A: Yes: define a subset of valid codewords $W_I \in W$ such that the ***minimum Hamming distance*** between all codewords is ***3*** (or more) ! The error checking method requires a more sophisticated parity checking algorithm (next slides).

W (n=6), $d_{H,min}$=3 :



Valid codeword (even parity)
Invalid codeword (odd parity)

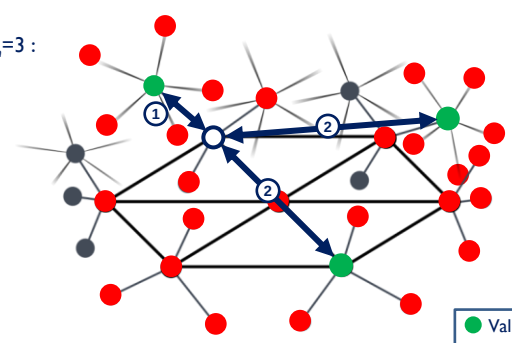
The Hamming Code

The Hamming Distance

Having defined the proper set $W_I \in W$ containing only codewords (●) with minimum Hamming distance 3 w.r.t. each other, we can now find a single valid codeword $w_I \in W$ closest, i.e. having Hamming distance 1, to a detected codeword w (○). Since this w_I only differs only 1 bitflip from the detected w, whereas the others differ in 2 positions, it is very likely that this w_I is the correct(ed) codeword.

W (n=6), $d_{H,min}$=3 :

○ Detected codeword w



Valid codeword (even parity)
Invalid codeword (odd parity)

The Hamming Code

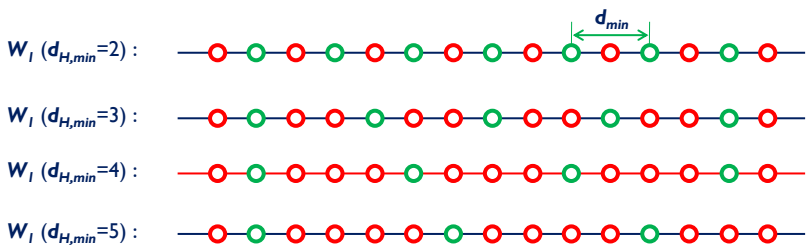
The Hamming Distance

The reasoning of Hamming was as follows: ***first***, we need to find a code W_l, where the Hamming distance between all allowed codewords $w \in W_l$ is at least a certain minimum distance $d_{H,min}$:

$$d_{H,\min} = Min\left(d_H(w_i, w_j);\ \forall i, j \in W; i \neq j\right)$$

Define a code $W_l = \{w\}$ with minimum Hamming distance $d_{H,min}$ and let's *visualize* all allowed codewords O and all not-allowed codewords O along a 1-dimensional line:

Then $W_l = \{$ O $\}$; all other codewords $\{$ O $\notin W_l$ $\}$



The Hamming Code

S : sent codeword

i : received codeword, *i* errors

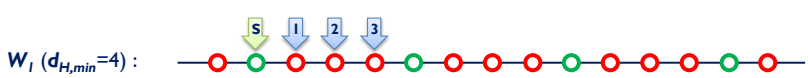
The Hamming Distance

W_l ($d_{H,min}$=2) :

No. erroneous bits that can be ***detected***: **1**
No. bits that can be ***corrected***: **0**

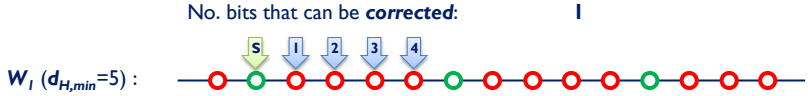
W_l ($d_{H,min}$=3) :

No. erroneous bits that can be ***detected***: **1**
No. bits that can be ***corrected***: **1**



W_l ($d_{H,min}$=4) :

No. erroneous bits that can be ***detected***: **2**
No. bits that can be ***corrected***: **1**



W_l ($d_{H,min}$=5) :

No. erroneous bits that can be ***detected***: **2**
No. bits that can be ***corrected***: **2**

The Hamming Code

The Hamming Distance

Summarizing:

- User data word length: m $(u_1, \ldots, u_m)$
- Number of parity bits: k $(c_1, \ldots, c_k)$
- Length of codeword: $n = m+p$ $(b_1, \ldots, b_n) = (u_1, \ldots, u_m, c_1, \ldots, c_k)$

If we are able to define a code $\boldsymbol{W_I} \in \boldsymbol{W} = \{ (b_1, .., b_n) ; b_i=(0,1), i=1 \ldots n \}$ with a certain minimum Hamming distance $\boldsymbol{d_{H,min}}$, then:

- The number of erroneous bits that can be *detected*: $\boldsymbol{N_{det.}}$= truncate[$\boldsymbol{d_{H,min}}/2$]
- The number of erroneous bits that can be *corrected*: $\boldsymbol{N_{corr.}}$= truncate[$(\boldsymbol{d_{H,min}}-1)/2$]

$$d_{H,\min} = 2 \cdot N_{corr.} + 1$$

Remaining questions to be answered:

- How many parity bits do we need ?
- How to define the parity bits ?
- How to generate the codewords ?

The Hamming Code

How Many Parity Bits ?

Suppose we have a code with length n. In order to be able to indicate not only whether an error occured, but also which bit was in error, we need to identify the erroneous bit unambiguously. Furthermore we assume we need to correct only one bit (only ***single bit errors***).

The number of the erroneous bit, e, in the detected codeword $\boldsymbol{w}$ ranges from 0 … n :
- e = 0 in case no error occured
- e = 1… n in case bit number e of the codeword $\boldsymbol{w}$ was erroneous.

This value of e is encoded in the numerical value of the parity word $\boldsymbol{c}$.

Example: $(u_1, u_2, \cancel{u_3}, \ldots, u_m)\ (c_1, \ldots, c_k)$: $m+k$ discrete positions

1 2 3 … m m+1 … m+k → e = 3

The numbers e = ***0*** (i.e. no error) … ***m+k*** need to be encoded in the k parity bits $\boldsymbol{c_k}$

With k parity bits we can binary encode 2^k distinct values ranging from ***0*** … $\boldsymbol{2^k-1}$.
In order to be able to uniquely identify the erroroneous bit, e, we have the condition:

$$2^k - 1 \geq m + k \quad \Rightarrow \quad 2^k - k \geq m + 1$$

The Hamming Code

How Many Parity Bits ?

Or, given the number of parity bits *k*, we can determine the maximum number of user bits *m* using the relation:

$$m \le 2^k - k - 1$$

Number of parity bits ***k***	Max. number of user bits ***m***	Code length ***n = m + k***	Code rate ***R = m / n***
1	0	1	0
2	1	3	0.33
3	4	7	0.57
4	11	15	0.73
5	26	31	0.84
6	57	63	0.90
…	…	…	…

… not very useful …

Q: How many parity bits are needed if we want to encode a user word of 6 bits length ?

Q: Why isn't this very efficient ?

The Hamming Code

How to Define the Parity Bits ?

We need to think of a ***systematic way*** to do a parity check on a certain number of *n* bits, and from this parity check obtain the value of e, i.e. the bit number that has been detected wrong.

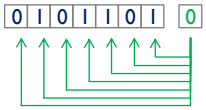
We know that with a single parity check this is not possible.

Suppose we want to send the user word (*m*=7): u = (0 1 0 1 1 0 1)

Single parity check: c = (0)

Codeword to be sent: w = (0 1 0 1 1 0 1 0)

The single parity bit here checks ALL user bits:

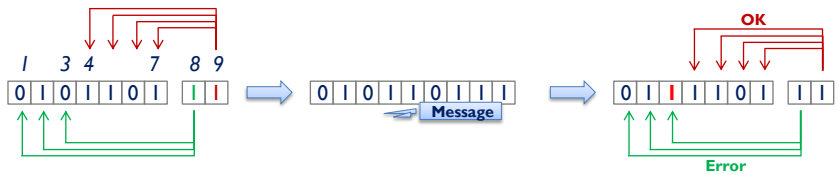


The Hamming Code

How to Define the Parity Bits ?

What if we were to use 2 parity bits, with the following parity check rule ?

"The first parity bit (bit 8) checks bits **1 to 3**, and the second parity bit (bit 9) checks bits **4 to 7**":



Now we can conclude that there must have been an error somewhere in bits 1…3 or 8 !
The parity check at bits 4…7 and 9 seems to be OK and these bits should be fine.

The Hamming Code

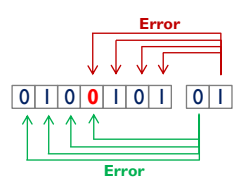
How to Define the Parity Bits ?

And what about the following parity check rule ?

"The first parity bit (bit 8) checks bits **1 to 4**, and the second parity bit (bit 9) checks bits **4 to 7**":

Again we can conclude that there must have been an error somewhere in bits 1…3 or 8 !

But what if the fourth bit (b_4) would have been erroneous ?
Both parity checks indicate an error! Since we assume that there has been only 1 bit error, the erroneous bit must have been at position 4 !!



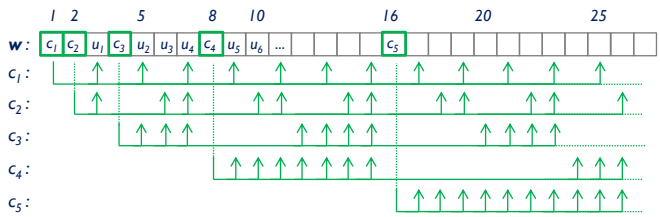
Error correction can be achieved by ***collaborating parity bits*** !

The Hamming Code

How to Define the Parity Bits ?

And now the way Hamming did it in a more ***systematic way***: he realized that each integer number can be represented by its ***unique binary equivalence***.

	c_5	c_4	c_3	c_2	c_1
1		0	0	0	1
2		0	0	1	0
3		0	0	1	1
4		0	1	0	0
5		0	1	0	1
6		0	1	1	0
7		0	1	1	1
8		1	0	0	0
9		1	0	0	1
10		1	0	1	0
11		1	0	1	1
12		1	1	0	0
13		1	1	0	1
14		1	1	1	0
15		1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1



The first parity bit checks the parity of all bits at odd positions

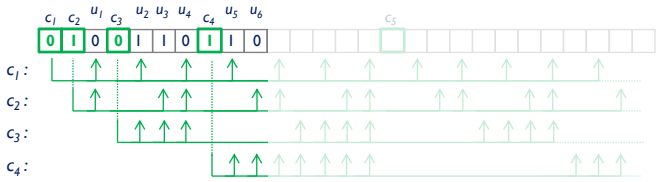
The 2nd parity bit checks the bits at position 2, 3, 6, 7, 10, 11, 14, 15, …

The 3rd parity bit checks the bits at position 4, 5, 6, 7, 12, 13, 14, 15, …
etc…

The Hamming Code

How to Define the Parity Bits ?

Let's take an example: user data word ***u*** = (0 1 1 0 1 0):
m = 6, requires 4 parity bits, *k* = 4 ⟹ codelength *n* = 10



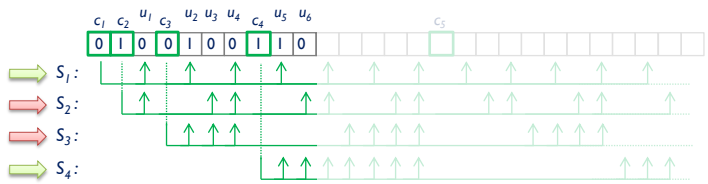
Hence, the codeword sent over to the receiver is: (0 1 0 0 1 1 0 1 1 0)

The Hamming Code

How to Define the Parity Bits ?

An error occurs in the communication channel and a single bit flips. The receiver now reads the following codeword:

(0 1 0 0 1 0 0 1 1 0)



The same parity check rules used for determining the parity bits ***c***, are now being used for calculating the ***syndrome*** vector ***S*** = (S_1, S_2, S_3, S_4): S_i=0 if parity is even, S_i=1 if parity is odd.

Apparently, the syndrome S_2 and S_3 indicate a read error. Supprisingly, the position e of the erroneous bit can now be easily calculated: the error has been detected and corrected !

$$e = \sum_{i=1}^{k} S_i \cdot 2^{i-1} \quad = 0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 = 6$$

The Hamming Code

How to Generate the Codewords ?

Hamming derived a thorough description of the underlying mathematics and came to a more systematic approach of generating the codewords and detecting and correcting errors.

(i) Given the number of user bits *m*, determine the number of parity bits *k* ; resulting in codewords of length $n = m + k = 2^k - 1$

(ii) The codeword ***w*** belonging to the user data word ***u*** is given by simple matrix algebra, using the ***Hamming parity check matrix H*** and the ***Generator matrix G***:

$$H = \begin{array}{c} \\ k \end{array} \overbrace{\left(\begin{array}{c|c} H' & I_k \end{array} \right)}^{m \quad k} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \quad (k=3,\ m=4)$$

Containing all k-dimensional binary vectors, except the 0-vector

Note: I_x = *x-dim identity matrix*

$$\bar{w} = \bar{u} \cdot G$$

$$G = \begin{array}{c} \\ m \end{array} \underbrace{\left(\begin{array}{c|c} I_m & H'^T \end{array} \right)}_{m \quad k} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \quad (k=3,\ m=4)$$

The Hamming Code

How to Generate the Codewords ?

Let us take a closer look at the (n=7, m=4) Hamming code.

The user is sending the user data word ***u*** = (1 0 1 1).

Q: Calculate the codeword ***w*** that is to be transmitted to the receiver.

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\vec{w} = \vec{u} \cdot G = \begin{pmatrix} 1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

The Hamming Code

How to Generate the Codewords ?

Let us take a closer look at the (n=7, m=4) Hamming code.

The codeword ***r*** is received at the detector: $\vec{r} = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & \boxed{0} \end{pmatrix}$
The syndrome ***S*** can now be calculated as follows:

$$\boxed{\vec{S} = H \cdot \vec{r}}$$

$$\vec{S} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

⇒ The syndrome vector corresponds with the 7th column in the parity check matrix, i.e. the 7th bit must have been wrong !!

The Hamming Code

The Hamming Code & Hamming Distance ?

We did see earlier that for a code with minimum Hamming distance $d_{H,min}$, the number of errors that can be detected is:

$$N_{corr.} = truncate\left[(d_{H,\min} - 1)/2\right] \quad \text{or} \quad d_{H,\min} = 2 \cdot N_{corr.} + 1$$

For the single error correcting Hamming code the minimum Hamming distance should be 3.

Let's check this for the (n=7, m=4) Hamming code.
The total number of valid codewords is 2^m = 16: $\mathbf{w}_i$, i=1..16

$$\bar{w} = \bar{u} \cdot G = \begin{pmatrix} u_1 & u_2 & u_3 & u_4 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} = \ldots \qquad \text{with} \quad u_i = (0,1); \quad i = 1 \ldots 4$$

The Hamming Code

The Hamming Code & Hamming Distance ?

(n=7, m=4) Hamming code:

$$[D]_{i,j} \equiv \left[d_{\text{Hamming}}(w_i, w_j)\right] = \left[\sum_{all\,bits}\left(XOR(w_i, w_j)\right)\right]$$

W = { (0000000)
(0001011)
(0010101)
(0011110)
(0100110)
(0101101)
(0110011)
(0111000)
(1000111)
(1001100)
(1010010)
(1011001)
(1100001)
(1101010)
(1110100)
(1111111) }

$$[D] = \begin{bmatrix} 0 & 3 & 3 & 4 & 3 & 4 & 4 & 3 & 4 & 3 & 3 & 4 & 3 & 4 & 4 & 7 \\ 3 & 0 & 4 & 3 & 4 & 3 & 3 & 4 & 3 & 4 & 4 & 3 & 4 & 3 & 7 & 4 \\ 3 & 4 & 0 & 3 & 4 & 3 & 3 & 4 & 3 & 4 & 4 & 3 & 4 & 7 & 3 & 4 \\ 4 & 3 & 3 & 0 & 3 & 4 & 4 & 3 & 4 & 3 & 3 & 4 & 7 & 4 & 4 & 3 \\ 3 & 4 & 4 & 3 & 0 & 3 & 3 & 4 & 3 & 4 & 4 & 7 & 4 & 3 & 3 & 4 \\ 4 & 3 & 3 & 4 & 3 & 0 & 4 & 3 & 4 & 3 & 7 & 4 & 3 & 4 & 4 & 3 \\ 4 & 3 & 3 & 4 & 3 & 4 & 0 & 3 & 4 & 7 & 3 & 4 & 3 & 4 & 4 & 3 \\ 3 & 4 & 4 & 3 & 4 & 3 & 3 & 0 & 7 & 4 & 4 & 3 & 4 & 3 & 3 & 4 \\ 4 & 3 & 3 & 4 & 3 & 4 & 4 & 7 & 0 & 3 & 3 & 4 & 3 & 4 & 4 & 3 \\ 3 & 4 & 4 & 3 & 4 & 3 & 7 & 4 & 3 & 0 & 4 & 3 & 4 & 3 & 3 & 4 \\ 3 & 4 & 4 & 3 & 4 & 7 & 3 & 4 & 3 & 4 & 0 & 3 & 4 & 3 & 3 & 4 \\ 4 & 3 & 3 & 4 & 7 & 4 & 4 & 3 & 4 & 3 & 3 & 0 & 3 & 4 & 4 & 3 \\ 3 & 4 & 4 & 7 & 4 & 3 & 3 & 4 & 3 & 4 & 4 & 3 & 0 & 3 & 3 & 4 \\ 4 & 3 & 7 & 4 & 3 & 4 & 4 & 3 & 4 & 3 & 3 & 4 & 3 & 0 & 4 & 3 \\ 4 & 7 & 3 & 4 & 3 & 4 & 4 & 3 & 4 & 3 & 3 & 4 & 3 & 4 & 0 & 3 \\ 7 & 4 & 4 & 3 & 4 & 3 & 3 & 4 & 3 & 4 & 4 & 3 & 4 & 3 & 3 & 0 \end{bmatrix}$$

The Hamming Code

The Hamming Code & Efficiency

The Hamming code is one of the very first error correcting codes, capable of correcting only 1 error in a sequence of bits. How realistic is this? Or, what is the probability of a double or higher bit error occuring, compared to a single bit error?

Define the probability of a single bit error as *p*.

For a *(n,m)* Hamming code with length *n* we have the following probabilities:

- Probability of no error occuring = $P_0 = (1-p)^n$

- Probability of single error occuring = $P_1 = n \cdot p \cdot (1-p)^{n-1}$

- Probability of >1 error occuring = $P_{2+} = \sum_{i=2}^{n} \binom{n}{i} \cdot p^i \cdot (1-p)^{n-i}$

With error correction: $P_{error} = \sum_{i=2}^{n} \binom{n}{i} \cdot p^i \cdot (1-p)^{n-i}$ $\quad P_{no\ error} = P_0 + P_1 = (1 + p \cdot (n-1)) \cdot (1-p)^{n-1}$

No error correction: $P_{error} = \sum_{i=1}^{n} \binom{n}{i} \cdot p^i \cdot (1-p)^{n-i}$ $\quad P_{no\ error} = P_0 = (1-p)^n$

$$\binom{n}{i} \equiv \frac{n!}{i! \cdot (n-i)!}$$

PHILIPS "Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

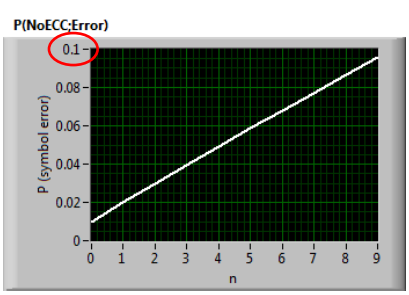
The Hamming Code

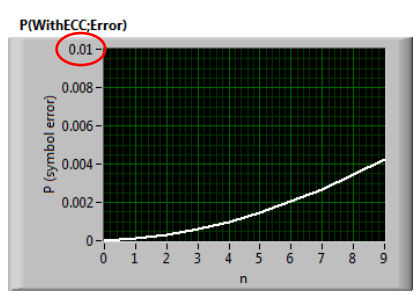
The Hamming Code & Efficiency

With error correction: $P_{error} = \sum_{i=2}^{n} \binom{n}{i} \cdot p^i \cdot (1-p)^{n-i}$ $\quad P_{no\ error} = P_0 + P_1 = (1 + p \cdot (n-1)) \cdot (1-p)^{n-1}$

No error correction: $P_{error} = \sum_{i=1}^{n} \binom{n}{i} \cdot p^i \cdot (1-p)^{n-i}$ $\quad P_{no\ error} = P_0 = (1-p)^n$

Let's look at the performance of the Hamming code, for a single bit error probability of e.g. ***p*=0.01**
The probability of a symbol error is now:

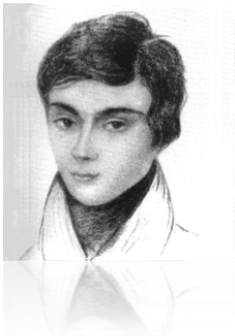




PHILIPS "Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

Finite Field Coding

Galois Fields and Reed-Solomon Coding



Evariste Galois
1811-1832

French mathematician, the founder of modern group theory and finite field algebra: Galois Field theory

Irving Reed 1923-2012 (left)
Gustave Solomon 1930-1996

Both american mathematicians and electrical engineers, major contributions in the field of error detecting and correcting coding theory

Finite Field Coding

Finite Field Algebra

Finite field, or Galois field, algebra is based on a field containing a ***finite number of elements***.

Some general characteristics of finite fields, denoted as ***GF*** or ***F*** :

- Within this algebra, ***addition*** and ***multiplication*** is defined amongst its field members; the outcome of an addition or multiplication is again an element of the field.

- The number of elements of the field, *q*, the ***order***, must be equal to an integer power *n* of a prime number *p*: $\boldsymbol{q = p^n\ (n>0)}$. *p* is also called the ***characteristic*** of the field.

- Field algebra is done using a polynomial representation of the field elements; in short this comes down to some kind of ***cyclic arithmetic***, like Mod_2 for binary numbers.

- Ex.: $F_{2^8} = F_{256} = \{(b_1,\ldots,b_8);\ \ b_i = (0,1),\ i = 1..8\}$ with Mod_2 arithmetic

$F_{3^2} = F_9 = \{(b_1,b_2);\ \ b_i = (0,1,2),\ i = 1..2\}$ with Mod_3 arithmetic

$\Rightarrow$ e.g.: $(0,2,1) + (1,2,2) = (1,1,0)$

The Reed Solomon Code

Reed Solomon Coding

Reed-Solomon codes are ***cyclic***, ***non-binary*** error detection and correction codes.

On CD the Reed-Solomon codewords are part of a finite field $F_{2^8} = F_{256}$ using binary arithmetic.
Here we demonstrate the principle of RS-coding using a somewhat smaller finite field: F_{31}.

Making the example a bit more practical: let's assume we are going to encode the letters of our alphabet including some special reading characters, as being part of this finite field F_{31}.
The field F_{31} then contains the following symbols: { A, B, .., Y, Z, +, - , ! , ? , . }.

Let's consider a book having 400 pages, 4000 characters each page.
And assume that the book has been printed on a crappy printer: the printer makes a printing mistake every 1 character on 1000. Consequently, the probability of a symbol error is 0.001.

In order to cope with this we are going to add additional ***parity symbols***, to be used for error correction.

Rule 1:
For every 4 user characters (m=4) we are adding 2 additional parity symbols (k=2).

The Reed Solomon Code

Reed Solomon Coding

Let us consider the user word ***u*** = "CODE": u_1="C"=3, u_2="O"=15, u_3="D"=4 and u_4="E"=5.
Next, we add 2 parity symbols ***c*** = (c_1,c_2), such that:

$$\boldsymbol{w} = (w_0, w_1, w_2, w_3, w_4, w_5) = (c_1, c_2, u_1, u_2, u_3, u_4)$$

Rule 2: Encoding scheme

$$(1): \sum_{i=0}^{n-1} w_i = 0 \quad \Leftrightarrow \quad w_0 + w_1 + w_2 + w_3 + w_4 + w_5 = 0 \quad Mod_{31}$$

$$(2): \sum_{i=0}^{n-1} i \cdot w_i = 0 \quad \Leftrightarrow \quad w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 = 0 \quad Mod_{31}$$

Now we fill in the numbers for w_i for the symbol "CODE" and we obtain:
from (2):

$$1 \cdot c_2 + 2 \cdot 3 + 3 \cdot 15 + 4 \cdot 4 + 5 \cdot 5 = c_2 + 92 = c_2 + 30 = 0 \;\Rightarrow\; c_2 = 1 = "A"$$

and from (1):

$$c_1 + 1 + 3 + 15 + 4 + 5 = c_1 + 28 = 0 \;\Rightarrow\; c_1 = 3 = "C"$$

The actual code word *that will be printed* is ***w*** = "CACODE".

The Reed Solomon Code

Reed Solomon Coding

Now suppose we read the following word from the printed book: "XGFOPT"
Leaving out the control symbols we would actually read "FOPT", which is an existing word !
However, is it correct? The word read by the detector is denoted as $\boldsymbol{w^d}$.

$$\boldsymbol{w^d} = (w^d_0, w^d_1, w^d_2, w^d_3, w^d_4, w^d_5) = (24, 7, 6, 15, 16, 20)$$

By putting the w_i's in (1) we get: $\sum_{i=0}^{5} w^d{}_i = 24+7+6+15+16+20 = 88 = 26 \quad Mod_{31} \neq 0$

This means that an error must have been made somewhere. Lets assume the value of the error was **e** and occured at symbol position *j*. The detected word can then be written as:

$$\vec{w}^d \equiv \left(w^d{}_i,\ i = 0 \ldots n-1\right) = \left(w_i,\ i \neq j;\ w_j + e,\ i = j\right)$$

Equations (1) and (2) can now be rewritten and result in the decoding scheme for RS-codes:

Rule 3: Decoding scheme

$$(1):\ \sum_{i=0}^{n-1} w^d{}_i = e + \sum_{i=0}^{n-1} w_i \quad \Rightarrow \quad e = \sum_{i=0}^{n-1} w^d{}_i$$

$$(2):\ \sum_{i=0}^{n-1} i \cdot w^d{}_i = j \cdot e + \sum_{i=0}^{n-1} i \cdot w_i \quad \Rightarrow \quad j \cdot e = \sum_{i=0}^{n-1} i \cdot w^d{}_i$$

The Reed Solomon Code

Reed Solomon Coding

Let us fill in the numbers for $\boldsymbol{w^d}$: "XGFOPT" = (w_0, w_1, w_2, w_3, w_4, w_5) = (24, 7, 6, 15, 16, 20) .
From (1):

$$e = \sum_{i=0}^{n-1} w^d{}_i = (24+7+6+15+\underset{j=4}{16}+20) = 88 = 26 = -5$$

By putting the w^d_i's in (2) we get:

$$j \cdot e = \sum_{i=0}^{n-1} i \cdot w^d{}_i = (0 \cdot 24 + 1 \cdot 7 + 2 \cdot 6 + 3 \cdot 15 + 4 \cdot 16 + 5 \cdot 20) = 228 = 11 = -20$$

Concluding: the value of the error was **e**=-5, and the symbol position must have been *j*=4.

$$w^d{}_j = w_j + e \quad \Rightarrow \quad w_j = w^d{}_j - e = 16 - (-5) = 21 = "U"$$

So the incorrectly read word "XGFOPT" is being corrected into "XGFOUT" and the original user word was "FOUT" !

Q: How many ***symbol*** errors can we correct in a single word with this method?

Q: How many ***bit*** errors can we correct in a single word with this method?

The Reed Solomon Code

Reed Solomon Coding - Efficiency

400 Pages, each page contains 4000 characters or symbols with a probability of a single symbol error of p = 0.001. Without error correction we have on average ***4 symbol errors per page*** !

Now how well does the RS-code perform?

First we have to mention two additional complicating factors:
(i) Since the rate of the proposed code R = 4/6, and we want to print the same number of user characters on a single page, we need to print in total 1.5 x 4000 = ***6000*** symbols = 1000 words.
(ii) If we want to print more characters on the same page, the characters (*the font*) become smaller, and consequently the probability of a printing error goes up, let's say a factor of 2: ***p = 0.002*** .

A word of 6 symbols is interpreted correctly if it contains zero or only one symbol error.
The probability of an uncorrectable word error is then given by:

$$P_{>1\ symbol\ error} = \sum_{i=2}^{6} \binom{6}{i} \cdot p^i \cdot (1-p)^{6-i} = \sum_{i=2}^{6} \binom{6}{i} \cdot 0.002^i \cdot (0.998)^{6-i} \approx 6 \cdot 10^{-5}$$

With 1000 words per page this means we have on average ***0.06 erroneous words per page***.

An improvement of a factor of ~ 30 !! (assuming approx. 2 symbol errors per word error).

… but still not good enough …

The Reed Solomon Code

Reed Solomon Coding - Efficiency

We can do even better than this: RS-codes can also correct multiple errors !

In this example we again use a code over $\boldsymbol{F_{31}}$, but now a single code word consists of 8 user symbols $\boldsymbol{u}$ (m=8) and 4 parity symbols $\boldsymbol{c}$ (k=4). The rate of this code again is 8/12 = 2/3.
The codewords are given by $\boldsymbol{w} = (w_0, \ldots, w_{11}) = (c_1, c_2, c_3, c_4, u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8)$

The encoding rules now become:

$$\sum_{i=0}^{11} w_i = 0 \quad Mod_{31}$$

$$\sum_{i=0}^{11} i^k \cdot w_i = 0 \quad Mod_{31} \quad for\ k = 1,2,3$$

When decoding we get 4 linear equations with 4 unknowns, meaning, we can detect and correct for 2 symbol errors $\mathbf{e}_{j1}$, $\mathbf{e}_{j2}$ at symbol positions $\boldsymbol{j_1}$ and $\boldsymbol{j_2}$:

$$e_{j_1} + e_{j_2} = \sum_{i=0}^{11} w^d{}_i$$

$$j_1{}^k \cdot e_{j_1} + j_2{}^k \cdot e_{j_2} = \sum_{i=0}^{11} i^k \cdot w^d{}_i \qquad for\ k = 1,2,3$$

Q: At what expense … ? Computational complexity !

The Reed Solomon Code

Reed Solomon Coding - Efficiency

Since the rate of this code is also 2/3, the probability of a single symbol error remains p=0.002 . However: now we can detect 2 symbol errors in a single word!

Q: What is the difference between the first RS-code (1 error correcting out of 4 user symbols) and the second RS-code (2 errors correcting out of 8 user symbols) ?

The probability of an uncorrectable readout now becomes:

$$P_{>2\ symbol\ error} = \sum_{i=3}^{12} \binom{12}{i} \cdot p^i \cdot (1-p)^{12-i} = \sum_{i=3}^{12} \binom{12}{i} \cdot 0.002^i \cdot (0.998)^{12-i} \approx 1.8 \cdot 10^{-6}$$

On a total of 6000 symbols per page, we now have 6000/12 = 500 words per page. The average probability of making a readout error now becomes 0.0009 word errors on single page, which comes down to approx. 2 characters in a 400 pages book (~5 symbol errors per word error)

	#Symbol errors per page	#Characters wrong in whole book
No coding	4	1600
RS(6,4)	0.12	48
RS(12,8)	0.006	2

The Reed Solomon Code

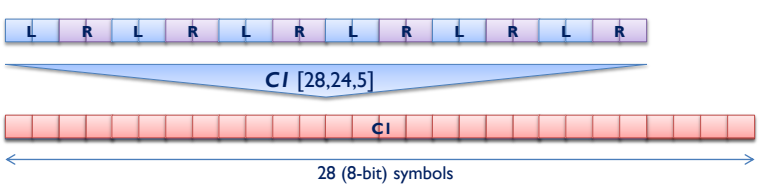
CIRC - Cross Interleaved Reed Solomon Code

In the Compact Disc system two Reed Solomon codes ***C1*** and ***C2*** work together in an interleaved fashion. As a result the system becomes very robust to ***burst errors***.

C1 is a [28,24,5] RS code on F_{256} : code length 28, 24 data / 4 parity symbols, Hamming distance 5
C2 is a [32,28,5] RS code on F_{256} : code length 32, 28 data / 4 parity symbols, Hamming distance 5

The CD records sound at a sampling rate of 44100 Hz (i.e. sample clock is T_s = 22.68 μsec) with 16 bits accuracy. The number of bits acquired during 6 sample moments ($6T_s$=0.136 msec) is then

N_{6T} = 6 (samples) x 2 (stereo) x 16 bits = 192 bits = 24 x 8 bits = 24 symbols in F_{256} .
These symbols are then encoded using the ***C1*** RS-code yielding 28 symbols.



The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

During sampling the ***C1*** encoder feeds its symbols to a memory buffer:

C1:

A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

time

PHILIPS

The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

During sampling the ***C1*** encoder feeds its symbols to a memory buffer:

C1:

B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

time

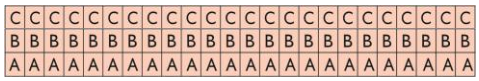
PHILIPS

The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

During sampling the ***C1*** encoder feeds its symbols to a memory buffer:

C1:



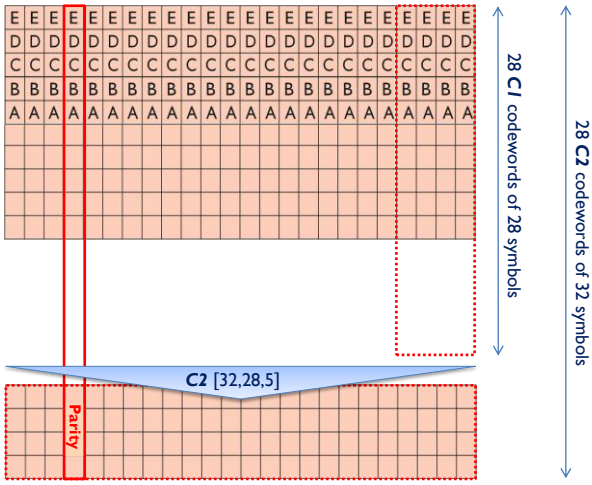
time

The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

After 28 x 6 sample clock intervals (28 x 0.136 msec = 3.81 msec) 28 ***C1*** codewords are available:

C1:

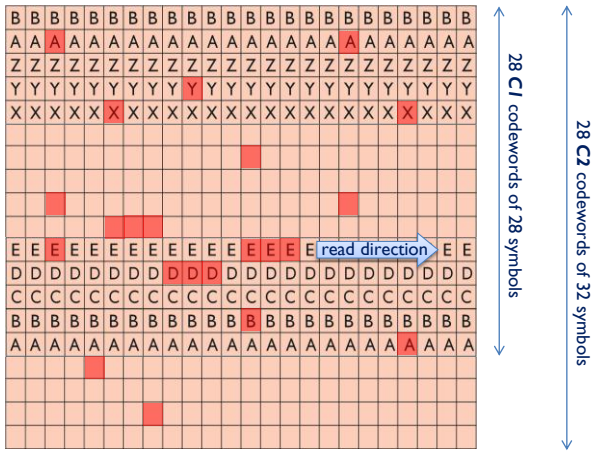


time

The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

Both ***C1*** and ***C2*** RS-codes can detect 3 and correct 2 symbol errors



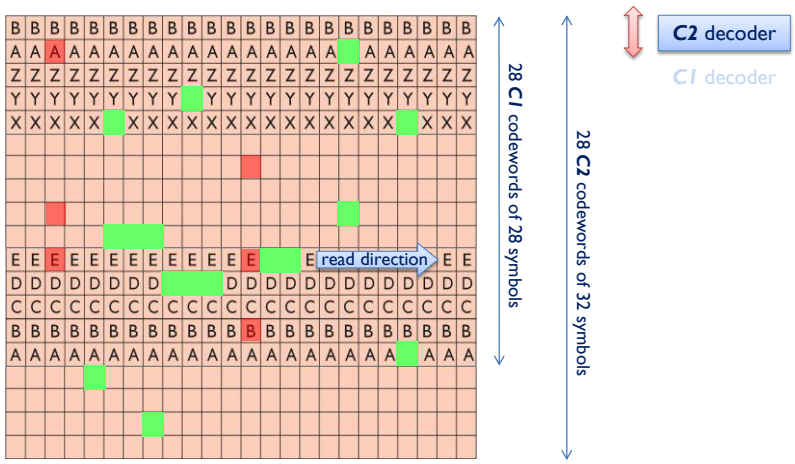
Random errors (noise) Burst errors (dust, scratches)

PHILIPS "Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

Both ***C1*** and ***C2*** RS-codes can detect 3 and correct 2 symbol errors



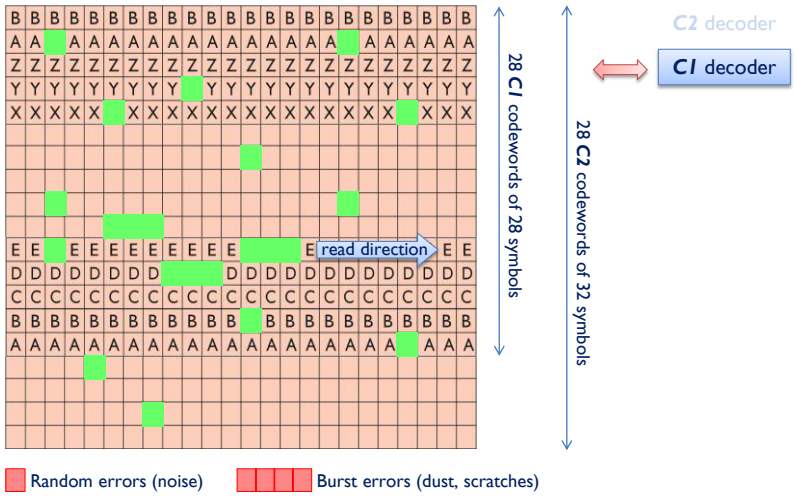
Random errors (noise) Burst errors (dust, scratches)

PHILIPS "Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

Both ***C1*** and ***C2*** RS-codes can detect 3 and correct 2 symbol errors



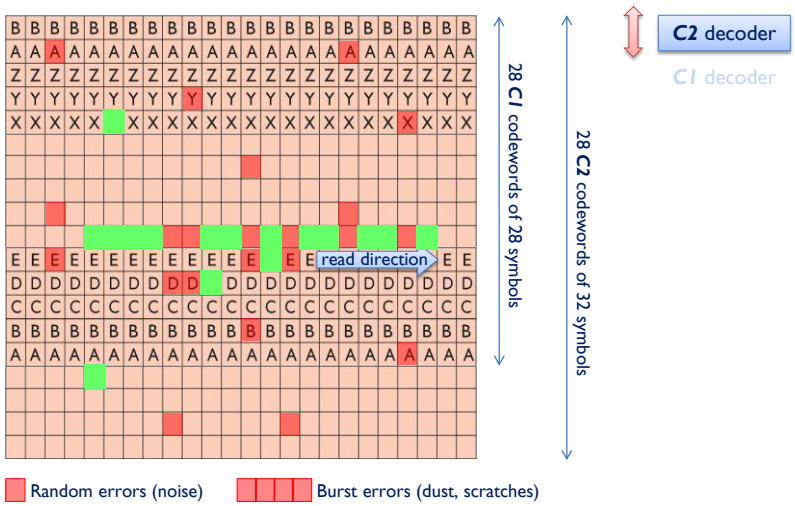
PHILIPS

"Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

What happens if longer burst errors occur ?



PHILIPS

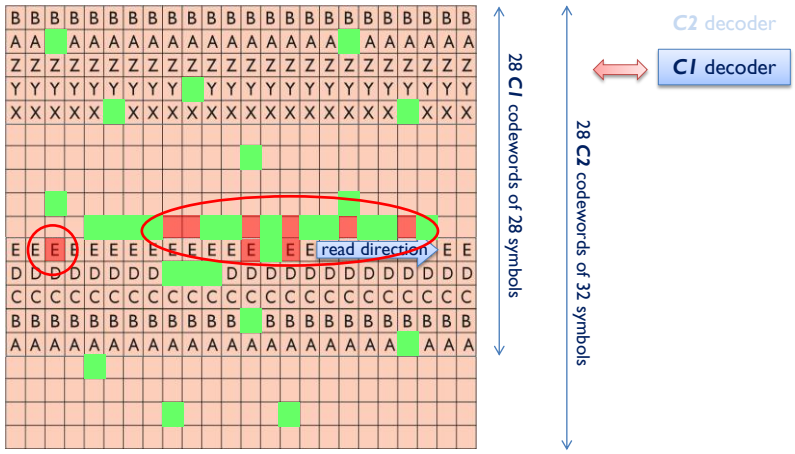
"Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

What happens if longer burst errors occur ?

Burst errors are difficult to correct !!



Random errors (noise) Burst errors (dust, scratches)

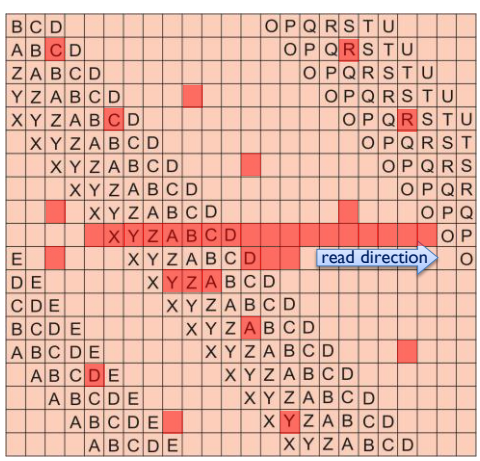
PHILIPS

"Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

By ***interleaving*** the data on the disc, the burst errors occur in different codewords !



Random errors (noise) Burst errors (dust, scratches)

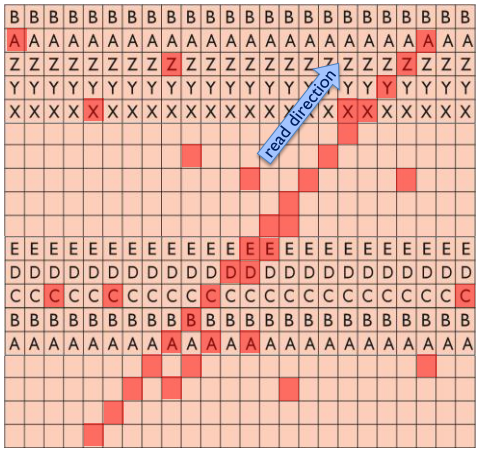
PHILIPS

"Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

By ***interleaving*** the data on the disc, the burst errors occur in different codewords !



In stead of having many (>2) symbol errors in one codeword we get many codewords having only 1 or 2 symbol errors !

And codewords with less than 3 symbol errors are correctable …

Random errors (noise) Burst errors (dust, scratches)

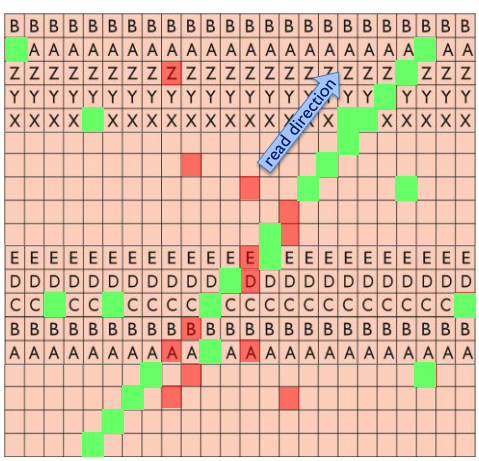
PHILIPS

"Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

Both ***C1*** and ***C2*** RS-codes can detect 3 and correct 2 symbol errors



C2 decoder

C1 decoder

Random errors (noise) Burst errors (dust, scratches)

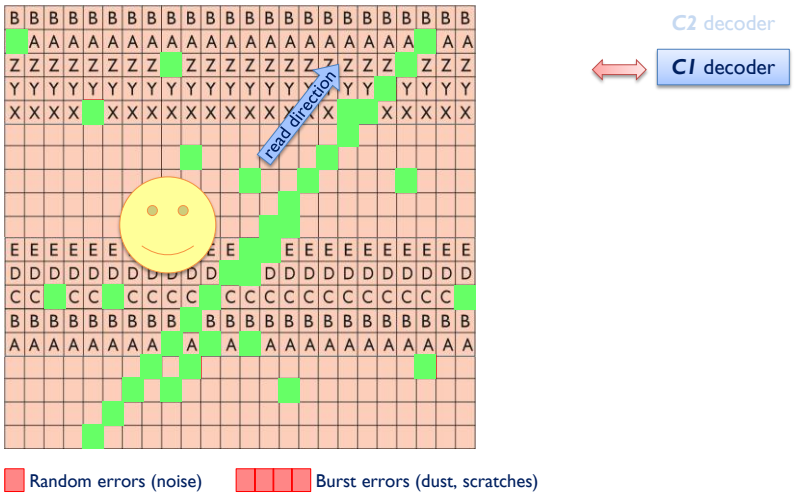
PHILIPS

"Blu-ray Disc en Wiskunde – De Magie van Fout Correctie", NWD2013, Jean Schleipen, Philips Research

The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

Both ***C1*** and ***C2*** RS-codes can detect 3 and correct 2 symbol errors



The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

Using these two RS-codes in succession, what is the probabilty of an uncorrectable error ?

Lets assume that the probability of making a single symbol error when reading the Compact Disc is slightly less than 1%: $p = 0.008$.

Since both codes can correct 2 errors, the probability that ***C2*** cannot correct a codeword is:

$$P_{C2,\,codeword\ error} = \sum_{i=3}^{32} \binom{32}{i} \cdot p^{i} \cdot (1-p)^{32-i}$$

Now the probability of a certain symbol of this codeword of ***C2*** being wrong, is given by:

$$P_{C2,\,Symbol} = \sum_{i=3}^{32} \binom{32}{i} \cdot \frac{i}{32} \cdot p^{i} \cdot (1-p)^{32-i}$$

Next the ***C1*** code is going to correct the symbols obtained by the ***C2*** decoding, and the error of having an uncorrectable codeword now is:

$$P_{C1,\,codeword\ error} = \sum_{i=3}^{28} \binom{28}{i} \cdot P_{C2,Symbol}{}^{i} \cdot (1-P_{C2,Symbol})^{32-i}$$

The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

Q: On a Compact Disc we have 700 Mbytes available, the equivalent of 1 hr music. Suppose the CD player has a probability of making a symbol read error of 0.008. How many "hickups" do we have during 1 minute, **without error correction** ?

A: 0.008 x 1 hr = 29 seconds; 2 hickups each minute !

The Reed Solomon Code

CIRC - Cross Interleaved Reed Solomon Code

Filling in the numbers:

$$P_{C2,\,Symbol} = \sum_{i=3}^{32} \binom{32}{i} \cdot \frac{i}{32} \cdot p^{i} \cdot (1-p)^{32-i} = 2.0 \times 10^{-4}$$

$$P_{C1,\,codeword\ error} = \sum_{i=3}^{28} \binom{28}{i} \cdot P_{C2,Symbol}{}^{i} \cdot (1-P_{C2,Symbol})^{32-i} = 2.8 \times 10^{-8}$$

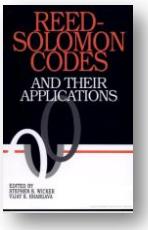
We had 28 codewords available in 3.81 msec.

- Consequently in 1 second we need to process 7350 codewords;
- With a probability of making a read error of 2.8×10^{-8}, in 1 sec we have 2.8×10^{-8} x 7350 = 0.00021 uncorrectable codewords;
- This means that on average only **one read error** occurs once every 4911 sec = **82 minutes** !!

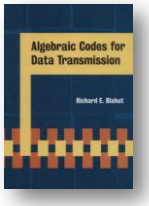
IMPRESSIVE … !!

Note:
On a Compact Disc we have 700 (user) Mbytes available = 29.2×10^{6} symbols ;
With a symbol error rate of ***p***=0.008, ***without error correction*** we would have ~ 2.3×10^{5} errors

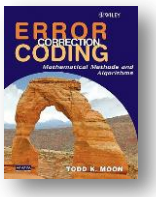
Some further reading … :



"Reed Solomon codes and their applications",
Stephen B. Wicker, Vijay K. Bhargava
ISBN 0-7803-5391-0 (1994)



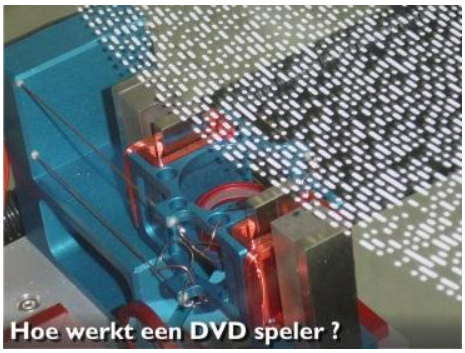
"Algebraic codes for data transmission",
Richard E. Blahut
ISBN 0-521-55374-1 (2003)



"Error correction coding:
mathematical methods and algorithms",
Todd K. Moon
ISBN 0-471-64800-0 (2005)

… and many freely accessible **pdf's** and **Youtube** movies on the internet …

Philips JetNet - DVD Demonstration Model



Lesmateriaal + ondersteunende video (DVD)

Contact:

Inhoud lesmateriaal & demonstrator: *jean.schleipen@philips.com*
Reserveren van demonstrator: *anja.welvaarts@philips.com*

Philips JetNet: *http://www.philips.nl/research/jet-net*
JetNet algemeen: *http://www.jet-net.nl/*