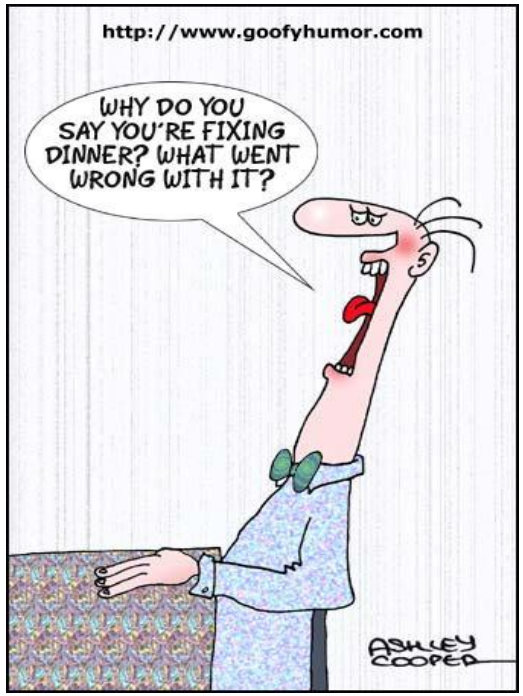
Fault-Tolerant Computing

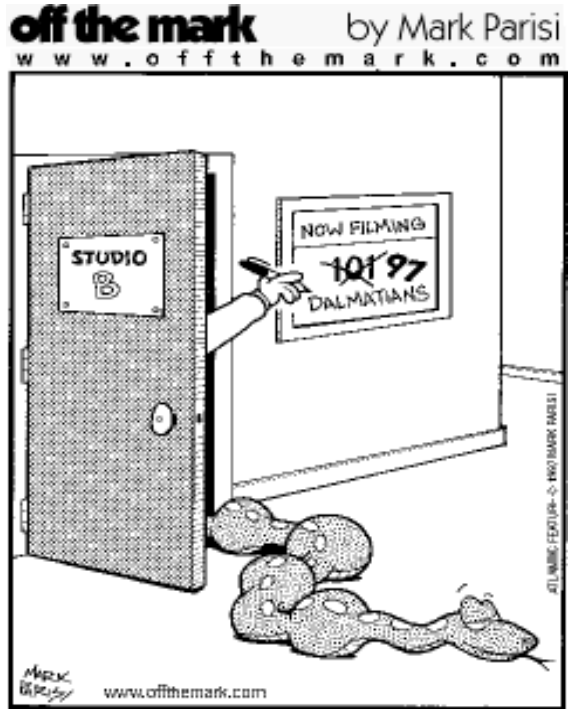
About This Presentation

This presentation has been prepared for the graduate course ECE 257A (Fault-Tolerant Computing) by Behrooz Parhami, Professor of Electrical and Computer Engineering at University of California, Santa Barbara. The material contained herein can be used freely in classroom teaching or any other educational setting. Unauthorized uses are prohibited. © Behrooz Parhami

Edition	Released	Revised	Revised
First	Oct. 2006		

Error Correction



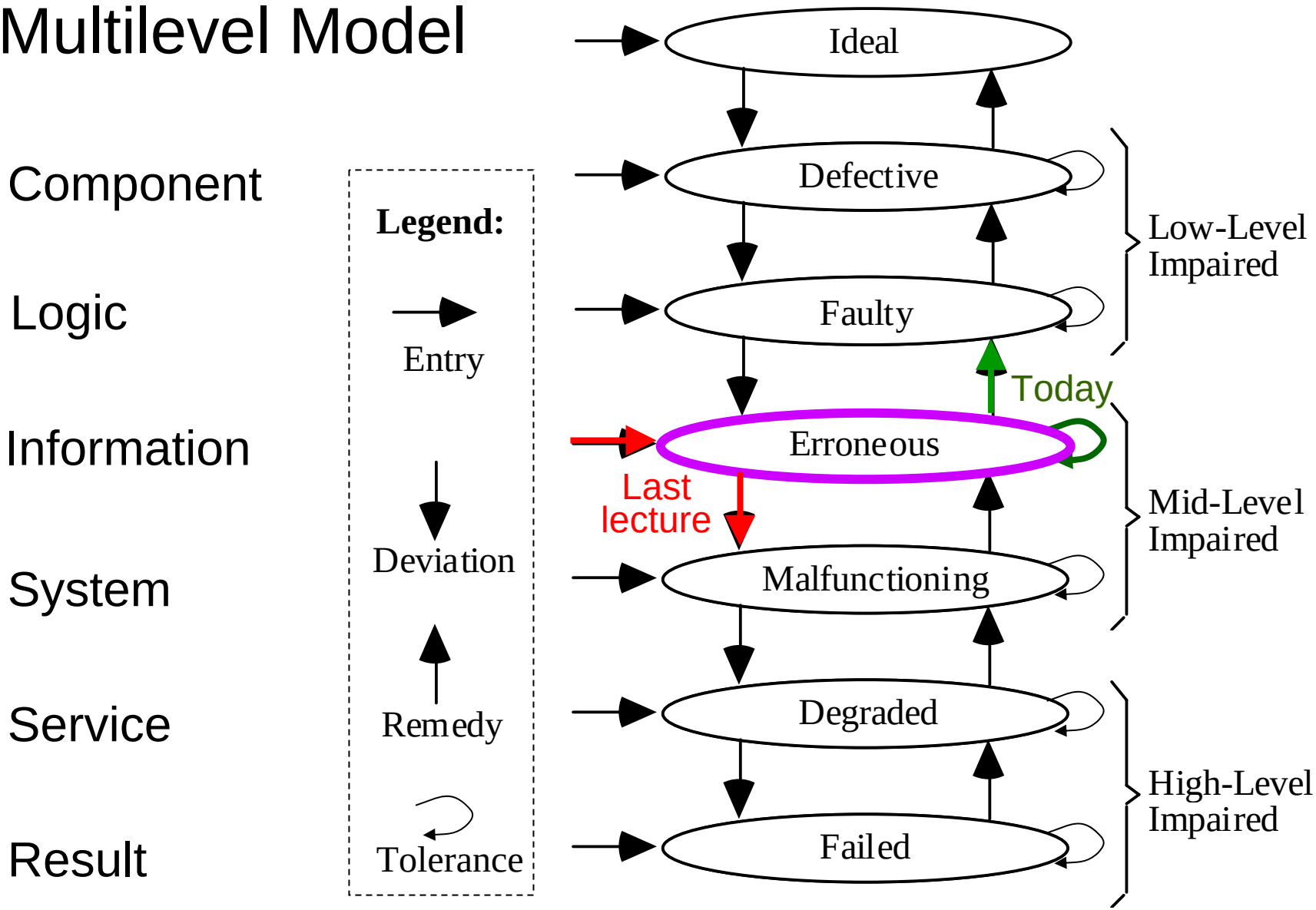


"Dear, your boss justcalled to tell you there was a slight mistake in your paycheck."

"We found the problem. You called your computer a moron and it's waiting for an apology."

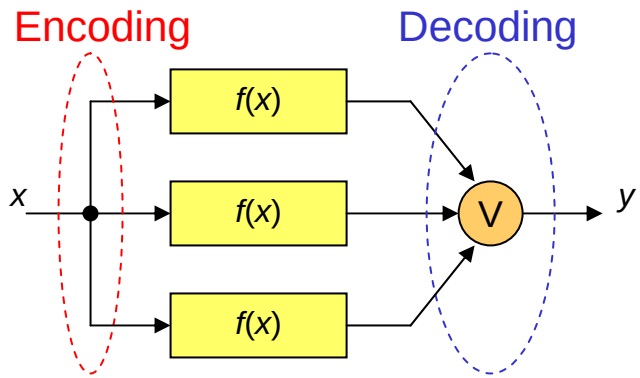
"THERE'S NOTHING WRONG WITH YOUR IPOD, DAD. IT'S JUST TOO EMBARRASSED TO PLAY THE KIND OF MUSIC YOU LIKE!"

Multilevel Model



High-Redundancy Codes

Triplication is a form of error coding:
x represented as *xxx* (200% redundancy)
Corrects any error in one version
Detects two nonsimultaneous errors



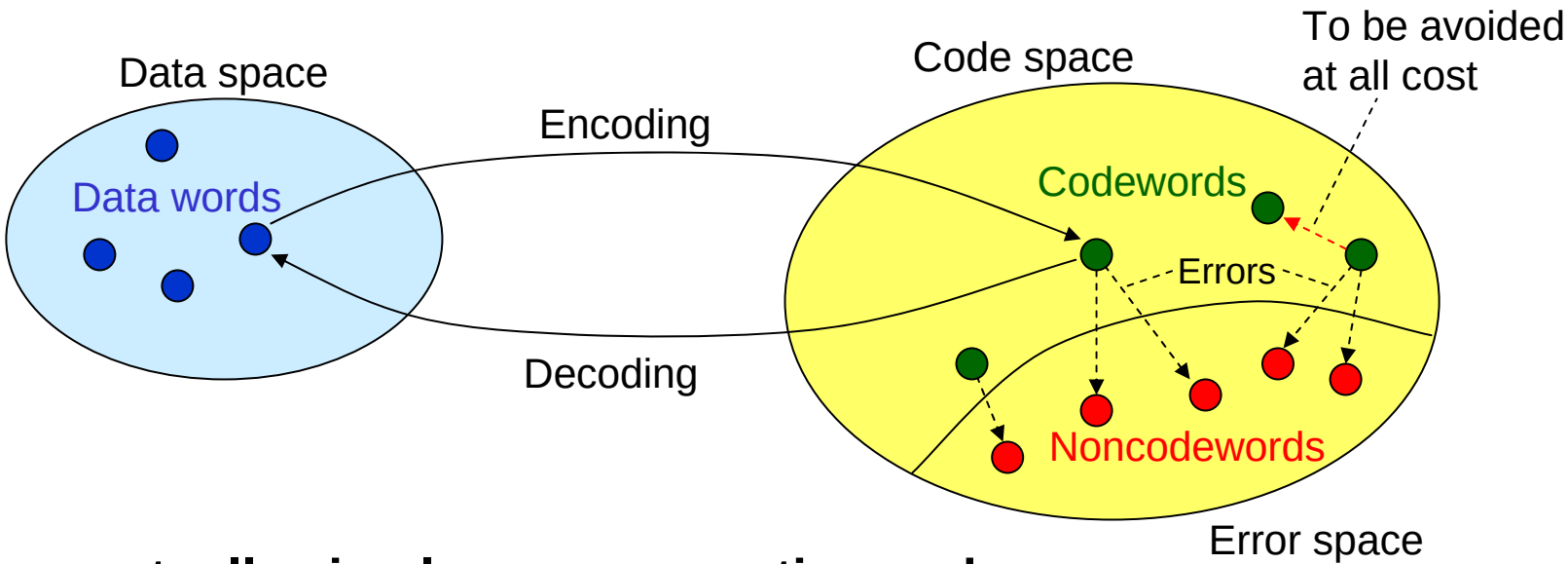
If we triplicate the voter to obtain 3 results, we are essentially performing the operation *f*(*x*) on coded inputs, getting coded outputs

With a larger replication factor, more errors can be tolerated

Our challenge today is to come up with strong correction capabilities, using much lower redundancy (perhaps an order of magnitude less)

To correct all single-bit errors in an *n*-bit code, we must have $2^r > n$, or $2^r > k + r$, which leads to about $\log_2 k$ check bits, at least

Error-Correcting Codes: Idea



A conceptually simple error-correcting code:

Arrange the k data bits into a $k^{1/2} \times k^{1/2}$ square array

Attach an even parity bit to each row and column of the array

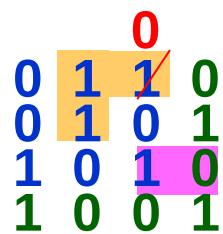
Row/Column check bit = XOR of all row/column data bits

Data space: All 2^k possible k-bit words

Redundancy: $2k^{1/2} + 1$ checkbits for k data bits

Corrects all single-bit errors (lead to distinct noncodewords)

Detects all double-bit errors (some triples go undetected)



Error-Correcting Codes: Evaluation

Redundancy: k data bits encoded in $n = k + r$ bits (r redundant bits)

Encoding: Complexity (cost / time) to form codeword from data word

Decoding: Complexity (cost / time) to obtain data word from codeword

Capability: Classes of error that can be corrected

Greater correction capability generally involves more redundancy
To correct c bit-errors, a minimum code distance of $2c + 1$ is required

Examples of code correction capabilities:
Single, double, byte, b-bit burst, unidirectional, . . . errors

Combined error correction/detection capability:
To correct c errors and additionally detect d errors ($d > c$),
a minimum code distance of $c + d + 1$ is required

Example: Hamming SEC/DED code has a code distance of 4

Hamming Distance for Error Correction

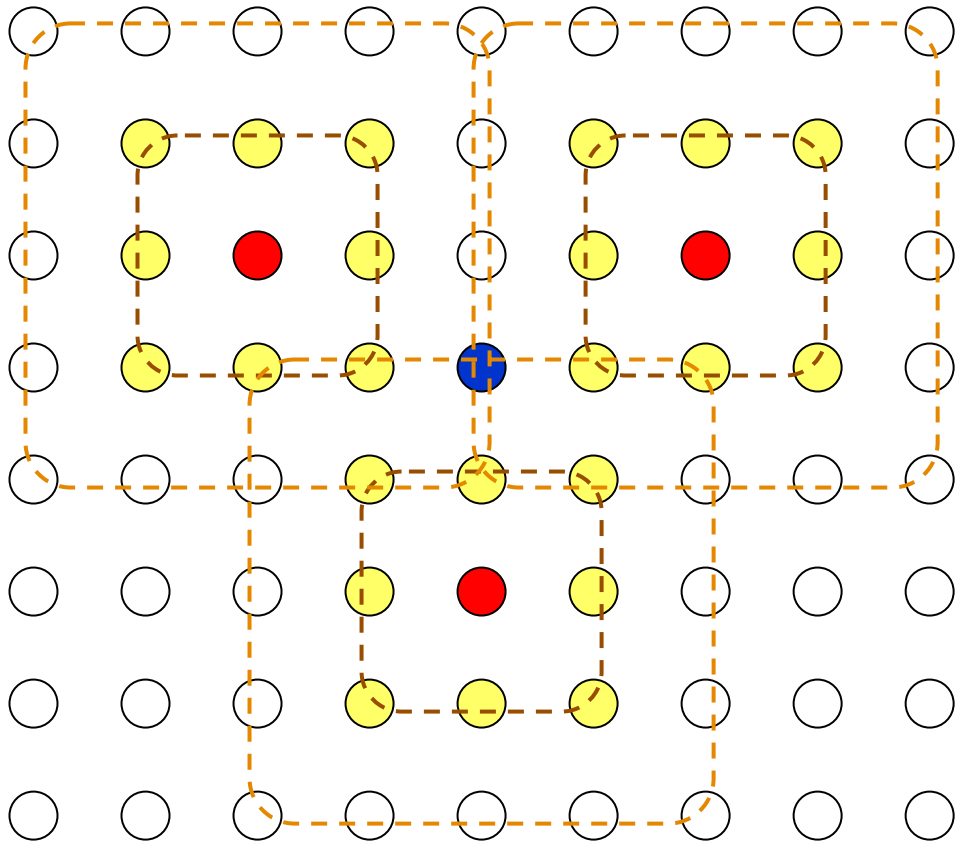
The following visualization, though not completely accurate, is still useful

Red dots represent codewords

Yellow dots, noncodewords within distance 1 of codewords, represent correctable errors

Blue dot, within distance 2 of three different codewords represents a detectable error

Not all “double errors” are correctable, however, because there are points within distance 2 of some codewords that are also within distance 1 of another



A Hamming SEC Code

Uses multiple parity bits, each applied to a different subset of data bits

Encoding: 3 XOR networks to form parity bits

Checking: 3 XOR networks to verify parities

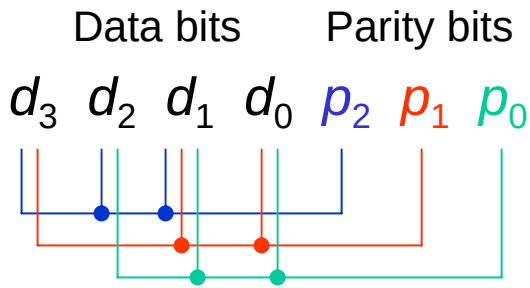
Decoding: Trivial (separable code)

Redundancy: 3 check bits for 4 data bits
Unimpressive, but gets better with more data bits
(7, 4); (15, 11); (31, 26); (63, 57); (127, 120)

Capability: Corrects any single-bit error

$s_2 = d_3 \oplus d_2 \oplus d_1 \oplus p_2$
$s_1 = d_3 \oplus d_1 \oplus d_0 \oplus p_1$
$s_0 = d_2 \oplus d_1 \oplus d_0 \oplus p_0$

$s_2\ s_1\ s_0$
Syndrome



$s_2\ s_1\ s_0$	Error
0 0 0	None
0 0 1	p_0
0 1 0	p_1
0 1 1	d_0
1 0 0	p_2
1 0 1	d_2
1 1 0	d_3
1 1 1	d_1

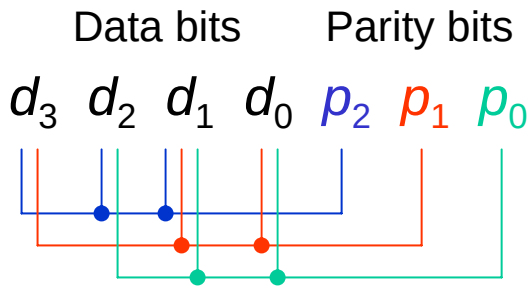
Matrix Formulation of Hamming SEC Code

Data bits Parity bits

$d_3\ d_2\ d_1\ d_0\ \ p_2\ p_1\ p_0$

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} d_3 \\ d_2 \\ d_1 \\ d_0 \\ p_2 \\ p_1 \\ p_0 \end{pmatrix} = \begin{pmatrix} s_2 \\ s_1 \\ s_0 \end{pmatrix}$$

Parity check matrix | Received word | Syndrome



$s_2\ s_1\ s_0$	Error
0 0 0	None
0 0 1	p_0
0 1 0	p_1
0 1 1	d_0
1 0 0	p_2
1 0 1	d_2
1 1 0	d_3
1 1 1	d_1

Syndrome matches the p_2 column in the parity check matrix

Matrix-vector multiplication is done with AND/XOR, instead of ×/+

Matrix Rearrangement for Simpler Correction

Data and parity bits

$p_0\ p_1\ d_0\ p_2\ d_2\ d_3\ d_1$

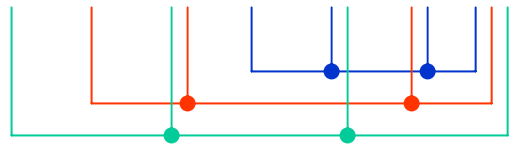
$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} p_0 \\ p_1 \\ d_0 \\ p_2 \\ d_2 \\ d_3 \\ d_1 \end{pmatrix} = \begin{pmatrix} s_2 \\ s_1 \\ s_0 \end{pmatrix}$$

1 2 3 4 5 6 7

Position number

Data and parity bits

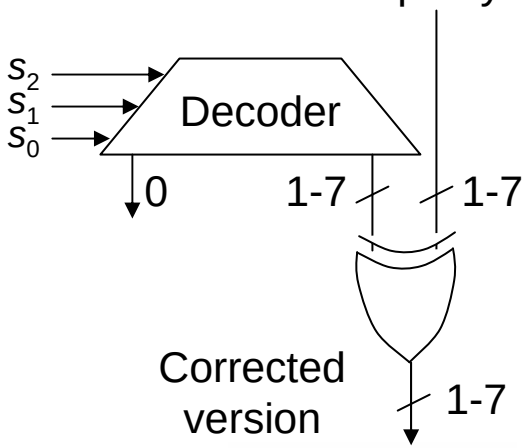
$p_0\ p_1\ d_0\ p_2\ d_2\ d_3\ d_1$



$s_2\ s_1\ s_0$	Error
0 0 0	None
0 0 1	p_0
0 1 0	p_1
0 1 1	d_0
1 0 0	p_2
1 0 1	d_2
1 1 0	d_3
1 1 1	d_1

Syndrome indicates error in position 4

Data and parity bits



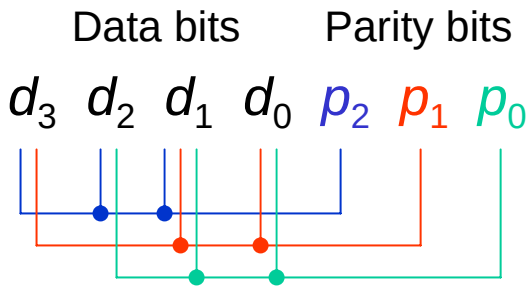
Hamming Generator Matrix

Data bits

$d_3\ d_2\ d_1\ d_0$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} d_3 \\ d_2 \\ d_1 \\ d_0 \end{pmatrix} = \begin{pmatrix} d_3 \\ d_2 \\ d_1 \\ d_0 \\ p_2 \\ p_1 \\ p_0 \end{pmatrix}$$

Generator matrix Data word Codeword



Recall that matrix-vector multiplication is done with AND/XOR, instead of ×/+

Generalization to Wider Hamming SEC Codes

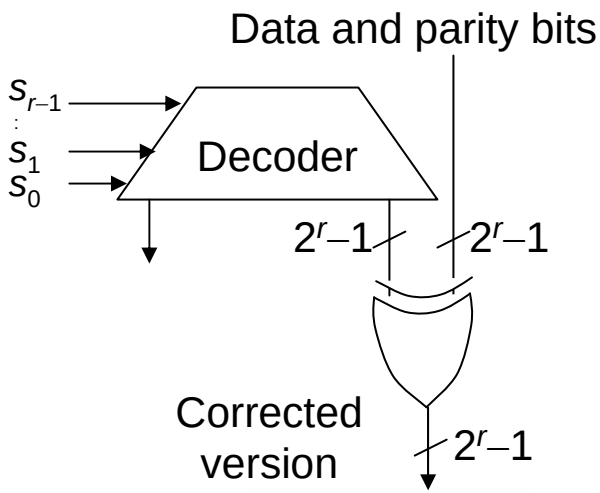
Data and parity bits

$p_0\ p_1\ d_0\ p_2\ .\ .\ .$

$$\begin{pmatrix} 0 & 0 & 0 & \ldots & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots & \vdots \\ 0 & 1 & 1 & \ldots & 0 & 1 & 1 \\ 1 & 0 & 1 & \ldots & 1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} p_0 \\ p_1 \\ d_0 \\ p_2 \\ . \\ . \\ . \end{pmatrix} = \begin{pmatrix} s_{r-1} \\ \vdots \\ s_1 \\ s_0 \end{pmatrix}$$

1 2 3 2^r-1

Position number



Condition for general Hamming SEC code:

$n = k + r = 2^r - 1$

n	$k = n - r$
7	4
15	11
31	26
63	57
127	120
255	247
511	502
1023	1013

A Hamming SEC/DED Code

Data and parity bits

$$\begin{pmatrix} p_0 & p_1 & d_0 & p_2 & . & . & . \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{\ldots} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 0 & \ldots & 1 & 1 & 1 & \mathbf{0} \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots & \vdots & \mathbf{\vdots} \\ 0 & 1 & 1 & \ldots & 0 & 1 & 1 & \mathbf{0} \\ 1 & 0 & 1 & \ldots & 1 & 0 & 1 & \mathbf{0} \\ \mathbf{1} & \mathbf{2} & \mathbf{3} & & & & \mathbf{2^r-1} & \end{pmatrix} \times \begin{pmatrix} p_0 \\ p_1 \\ d_0 \\ p_2 \\ . \\ . \\ . \\ \mathbf{p_r} \end{pmatrix} = \begin{pmatrix} \mathbf{s_r} \\ s_{r-1} \\ \vdots \\ s_1 \\ s_0 \end{pmatrix}$$

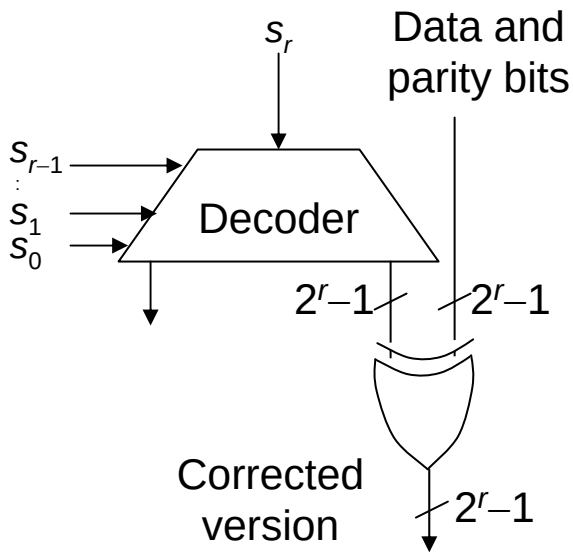
Position number

Parity check matrix

Received word

Syndrome

Add an extra row of all 1s and a column with only one 1 to the parity check matrix



Some Other Useful Codes

Hamming codes are examples of linear codes
Linear codes may be defined in many other ways

BCH codes: Named in honor of Bose, Chaudhuri, Hocquenghem

Reed-Solomon codes: Special case of BCH code

Example: A popular variant is RS(255, 223) with 8-bit symbols
223 bytes of data, 32 check bytes, redundancy ≈ 14%
Can correct errors in up to 16 bytes anywhere in the 255-byte codeword
Used in CD players, digital audio tape, digital television

Reed-Muller codes: Have a recursive construction, with smaller codes used to build larger ones

Arithmetic Error-Correcting Codes

Positive error	Syndrome mod 7	Syndrome mod 15	Negative error	Syndrome mod 7	Syndrome mod 15
1	1	1	–1	6	14
2	2	2	–2	5	13
4	4	4	–4	3	11
8	1	8	–8	6	7
16	2	1	–16	5	14
32	4	2	–32	3	13
64	1	4	–64	6	11
128	2	8	–128	5	7
256	4	1	–256	3	14
512	1	2	–512	6	13
1024	2	4	–1024	5	11
2048	4	8	–2048	3	7
4096	1	1	–4096	6	14
8192	2	2	–8192	5	13
16,384	4	4	–16,384	3	11
32,768	1	8	–32,768	6	7

Error syndromes for weight-1 arithmetic errors in the (7, 15) biresidue code

Because all the syndromes in this table are different, any weight-1 arithmetic error is correctable by the (mod 7, mod 15) biresidue code

Properties of Biresidue Codes

Biresidue code with relatively prime low-cost check moduli $A = 2^a - 1$ and $B = 2^b - 1$ supports $a \times b$ bits of data for weight-1 error correction

Representational redundancy = $(a + b)/(ab) = 1/a + 1/b$

a	*b*	*n=k+a+b*	*k=ab*
3	4	19	12
5	6	41	30
7	8	71	56
11	12	143	120
15	16	271	240

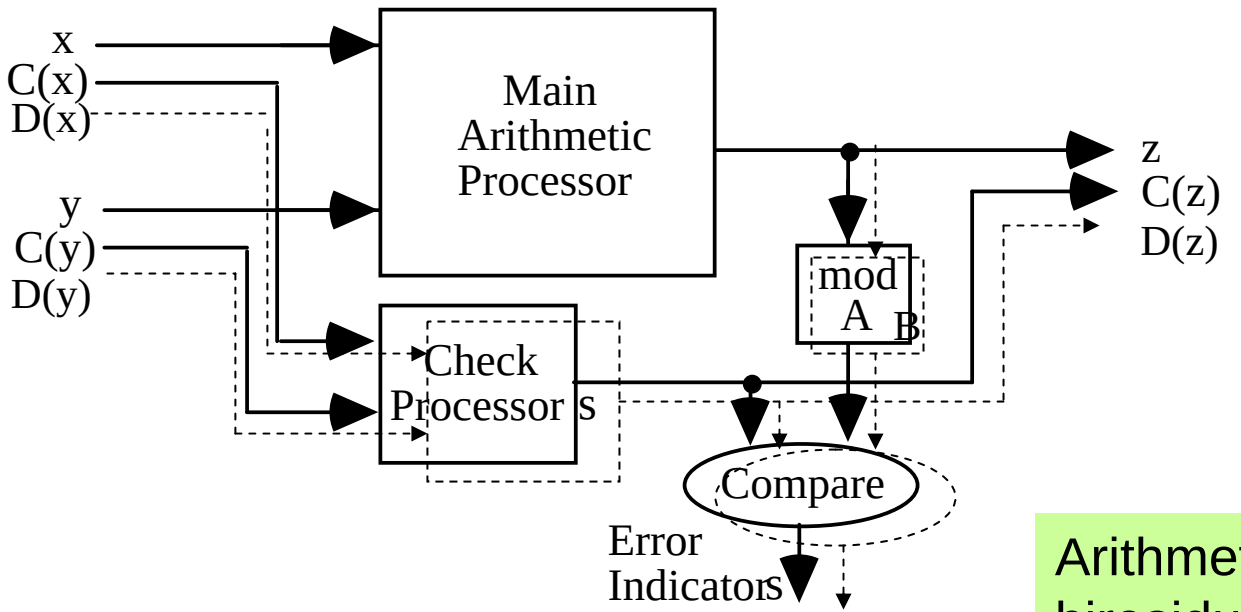
Compare with Hamming SEC code →

n	*k*
7	4
15	11
31	26
63	57
127	120
255	247
511	502
1023	1013

Arithmetic on Biresidue-Coded Operands

Similar to residue-checked arithmetic for addition and multiplication, except that two residues are involved

Divide/square-root: remains difficult



Arithmetic processor with biresidue checking.

Higher-Level Error Coding Methods

We have applied coding to data at the bit-string or word level

It is also possible to apply coding at higher levels

Data structure level – Robust data structures

Application level – Algorithm-based error tolerance

Preview of Algorithm-Based Error Tolerance

Error coding applied to data structures, rather than at the level of atomic data elements

Example: mod-8 checksums used for matrices

If $Z = X \times Y$ then $Z_f = X_c \times Y_r$

In M_f, any single error is correctable and any 3 errors are detectable

Four errors may go undetected

Matrix M

$$M = \begin{pmatrix} 2 & 1 & 6 \\ 5 & 3 & 4 \\ 3 & 2 & 7 \end{pmatrix}$$

Row checksum matrix

$$M_r = \begin{pmatrix} 2 & 1 & 6 & 1 \\ 5 & 3 & 4 & 4 \\ 3 & 2 & 7 & 4 \end{pmatrix}$$

Column checksum matrix

$$M_c = \begin{pmatrix} 2 & 1 & 6 \\ 5 & 3 & 4 \\ 3 & 2 & 7 \\ 2 & 6 & 1 \end{pmatrix}$$

Full checksum matrix

$$M_f = \begin{pmatrix} 2 & 1 & 6 & 1 \\ 5 & 3 & 4 & 4 \\ 3 & 2 & 7 & 4 \\ 2 & 6 & 1 & 1 \end{pmatrix}$$