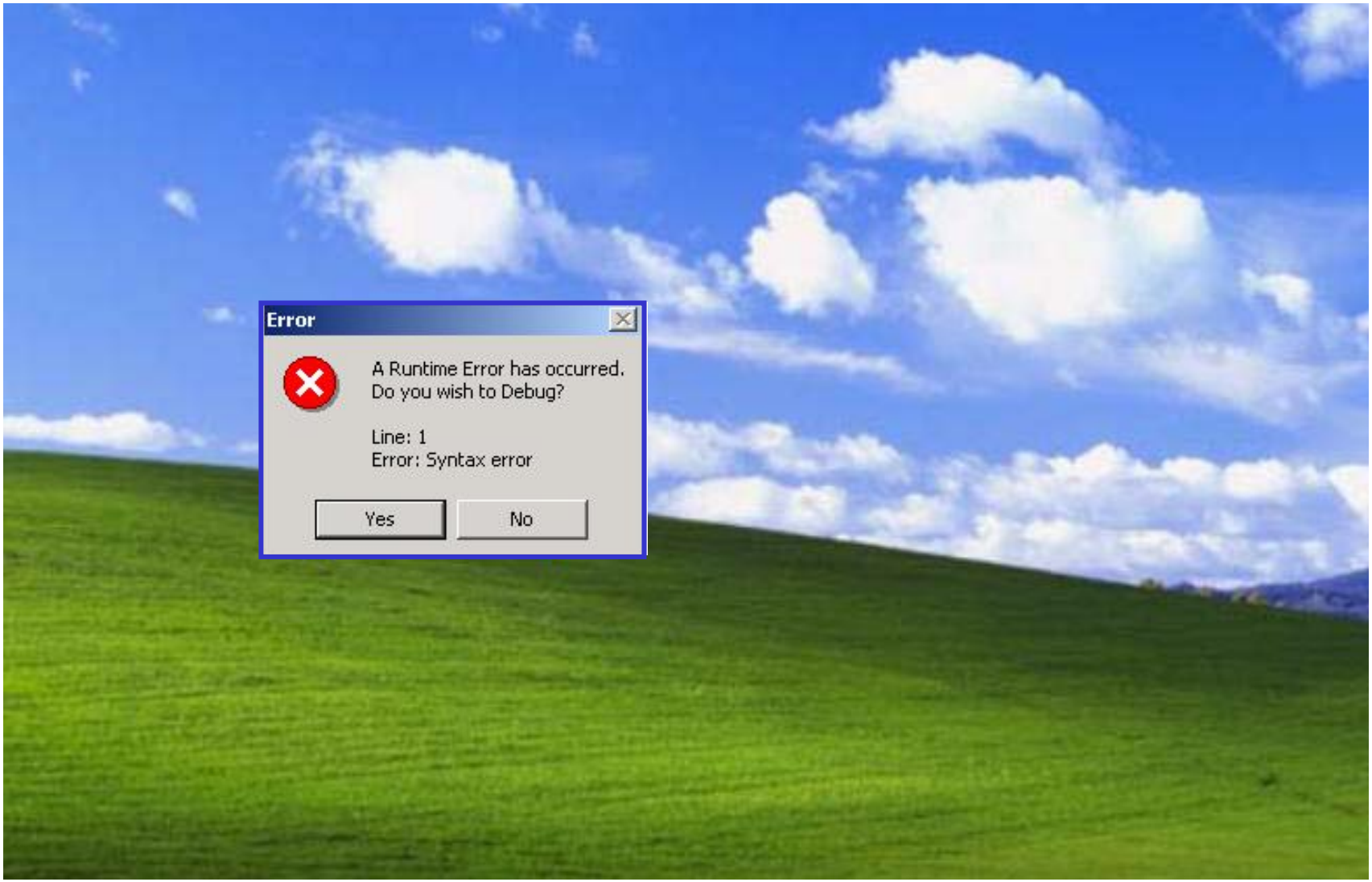
Fault-Tolerant Computing

About This Presentation

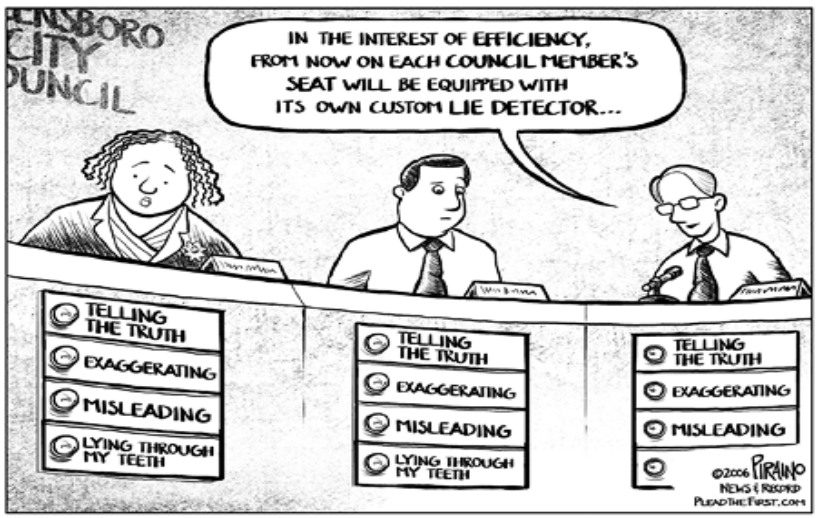
This presentation has been prepared for the graduate course ECE 257A (Fault-Tolerant Computing) by Behrooz Parhami, Professor of Electrical and Computer Engineering at University of California, Santa Barbara. The material contained herein can be used freely in classroom teaching or any other educational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

Edition	Released	Revised	Revised
First	Oct. 2006		

Error Detection



"Just among us we goofed. But officially it will go down as computer error."



"We rarely back up our data. We'd rather not keep a permanent record of everything that goes wrong around here!"

Multilevel Model

Component

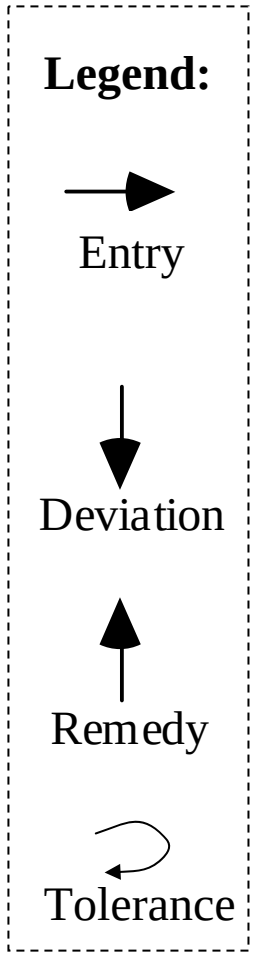
Logic

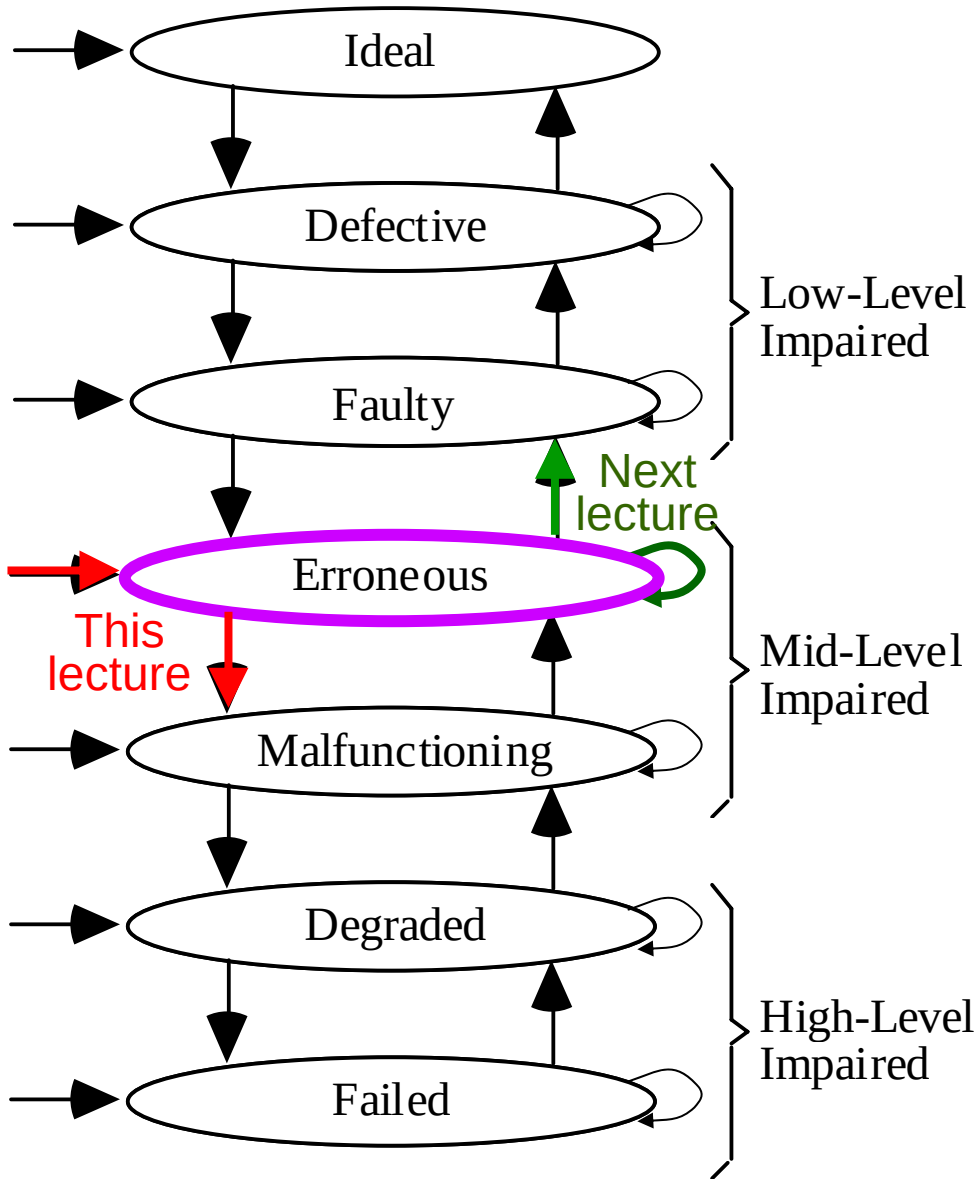
Information

System

Service

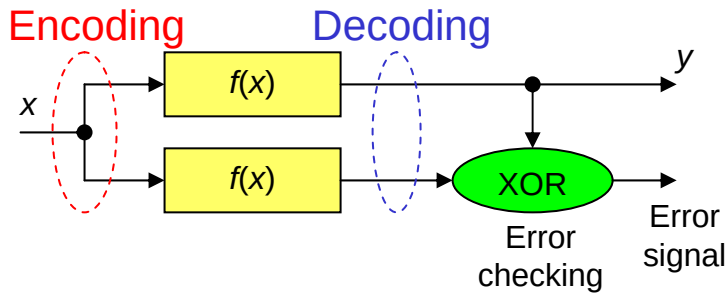
Result



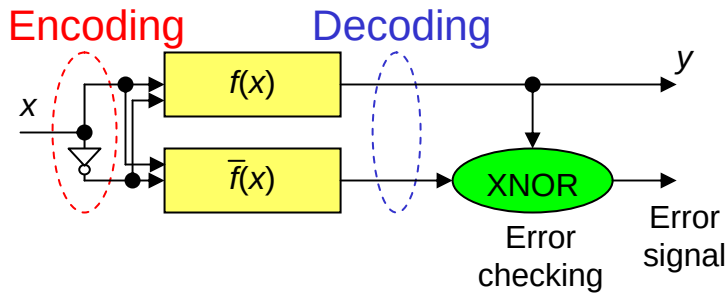


High-Redundancy Codes

Duplication is a form of error coding:
x represented as *xx* (100% redundancy)
Detects any error in one version

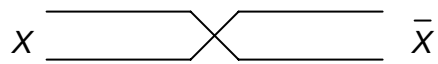


Two-rail encoding
x represented as $x\bar{x}$ (100% redundancy)
e.g., 0 represented as 01; 1 as 10
Detects any error in one version
Detects all unidirectional errors

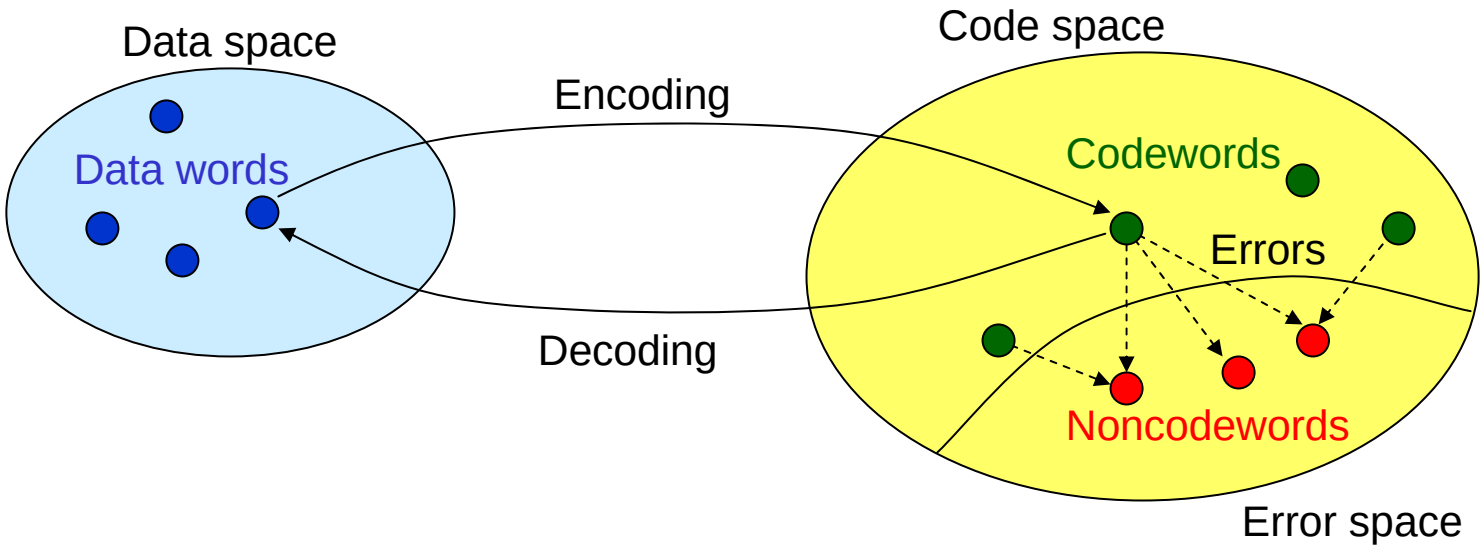


Two-rail logic elements

AND: $(a_0, a_1)\ (b_0, b_1) = (a_0 \vee b_0, a_1 b_1)$
OR: $(a_0, a_1) \vee (b_0, b_1) = (a_0 b_0, a_1 \vee b_1)$
NOT: $(a_0, a_1)' = (a_1, a_0)$
XOR: $(a_0, a_1) \oplus (b_0, b_1) = (a_0 b_1 \vee a_1 b_0, a_0 b_0 \vee a_1 b_1)$



The Concept of Error-Detecting Codes



The simplest possible error-detecting code:

Attach an even parity bit to each k-bit data word

Check bit = XOR of all data bits

0 0 1 0 1 0 ~~0~~ 1 0 1 1

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

Code space: All 2^k possible even-parity (k + 1)-bit codewords

Error space: All 2^k possible odd-parity (k + 1)-bit noncodewords

Detects all single-bit errors

Evaluation of Error-Detecting Codes

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
Separable codes have computation-free decoding

Capability: Classes of error that can be detected
Greater detection capability generally involves more redundancy
To detect d bit-errors, a minimum code distance of $d + 1$ is required

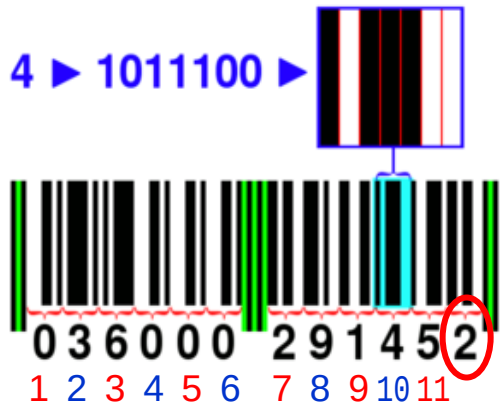
Examples of code detection capabilities:
Single, double, b-bit burst, byte, unidirectional, . . . errors

Closure: Arithmetic and other operations done directly on codewords
(rather than in 3 stages: decode, operate, and encode)

Error Detection in UPC-A

To obtain the check digit for 12-digit UPC-A universal product code:
Add the odd-indexed digits and multiply the sum by 3
Add the sum of even-indexed digits to previous result
Subtract the total from the next higher multiple of 10

Example:
Sum odd indexed digits: 0 + 6 + 0 + 2 + 1 + 5 = 14
Multiply by 3: 14 × 3 = 42
Add even-indexed digits: 42 + 3 + 0 + 0 + 9 + 4 = 58
Compute check digit: 60 – 58 = 2



Bar code uses 7 bits per digit, with different encodings on the right and left halves and different parities at various positions

Checking:
Verify that the modulo-10 sum of all 12 digits is 0

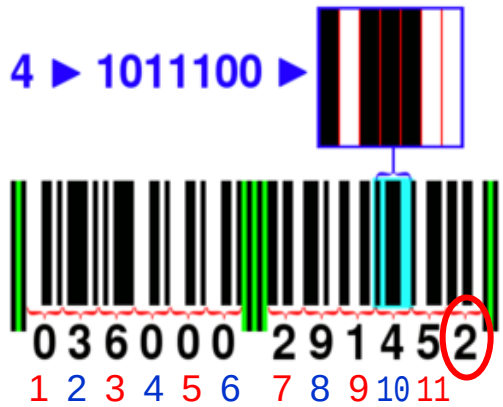
Capabilities:
Detects all single-digit errors
Detects most, but not all, transposition errors

Checksum Codes

Given a data vector $x_1, x_2, \ldots, x_n$, encode the data by attaching the checksum x_{n+1} to the end, such that $\sum_{j=1 \text{ to } n+1} w_j x_j = 0 \bmod A$

The elements w_j of the weight vector w are predetermined constants

Example:
For the UPC-A checksum scheme, we have
$w = 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1$
$A = 10$



Checking:
Verify that the modulo-A sum of all elements is 0

Capabilities:
Detects all errors adding an error magnitude that is not a multiple of A

Variant: Vector elements may be XORed rather than added together

Hamming Distance

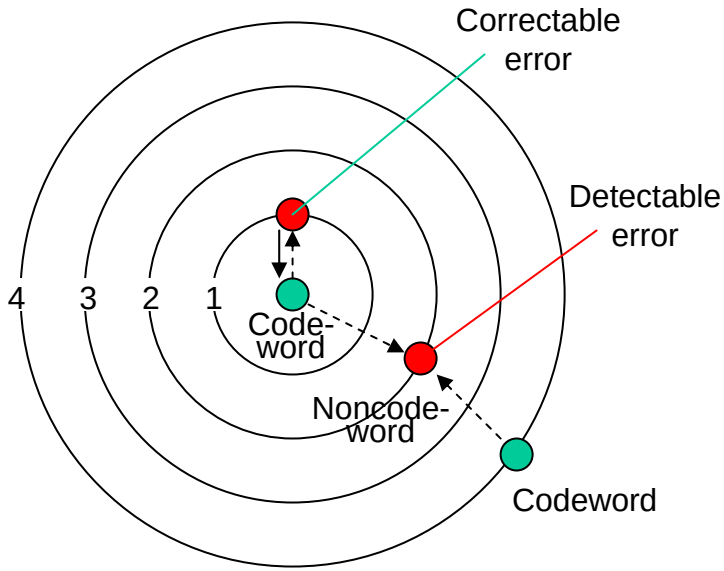
Definition: Hamming distance between two bit-vectors is the number of positions in which they differ

A distance-2 code:

00011
00101
00110
01001
01010
01100
10001
10010
10100
11000

00110 → 00111 (0→1 error)

00110 → 00100 (1→0 error)



Min H-dist	Code capability
2	$d = 1$; SED
3	$c = 1$; SEC or ($d = 2$; DED)
4	$c = 1$ and $d = 2$; SEC/DED
5	$c = 2$ or ($c = 1$ and $d = 3$; SEC/3ED)
h	cEC/dED such that $h = c + d + 1$

Error Classification and Models

Goal of error tolerance methods:

Allow uninterrupted operation despite presence of certain errors
Error model – Relationship between errors and faults (or other causes)

Errors are detected/corrected through:

Encoded (redundant) data, plus code checkers
Reasonableness checks, activity monitoring, retry

Errors are classified as:

Single or Multiple (according to the number of bits affected)
Inversion or Erasure (symbol or bit changed or lost)*
Random or Correlated (correlation in the form of byte or burst error)
Symmetric or Asymmetric (regarding $0 \rightarrow 1$ and $1 \rightarrow 0$ inversions)

* Nonbinary codes have substitution rather than inversion errors
Also of interest for nonelectronic systems are transposition errors

Errors are permanent by nature; transient faults, not transient errors

Application of Coding to Error Control

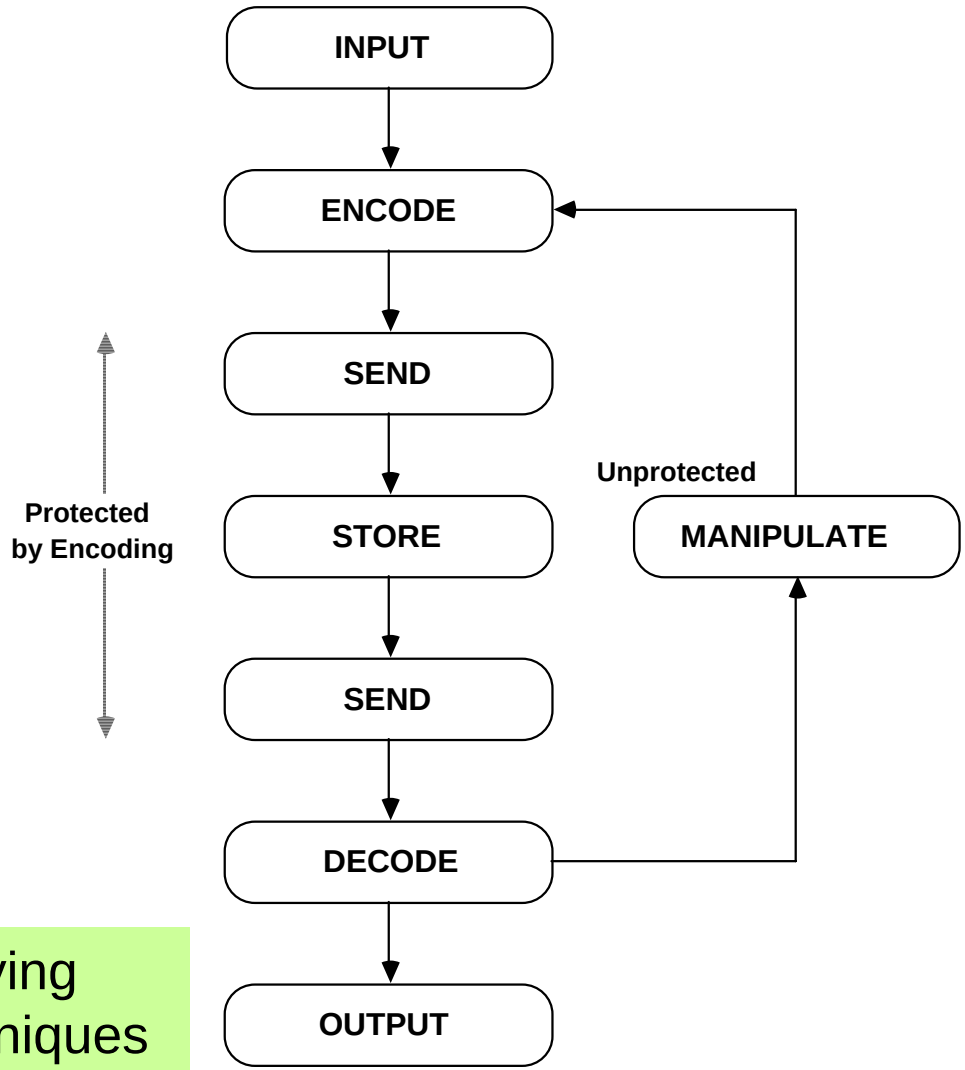
Ordinary codes can be used for storage and transmission errors; they are not closed under arithmetic / logic operations

Error-detecting, error-correcting, or combination codes (e.g., Hamming SEC/DED)

Arithmetic codes can help detect (or correct) errors during data manipulations:

1. Product codes (e.g., 15*x*)
2. Residue codes (*x* mod 15)

A common way of applying information coding techniques



Constant-Weight Codes

Definition: All codewords have the same number of 1s

A weight-2 code:

00011
00101
00110
01001
01010
01100
10001
10010
10100
11000

Can detect all unidirectional errors

Maximum number of codewords obtained when weight of n-bit codewords is $n/2$

Berger Codes

Definition: Separable code that has the count of 0s within the data part attached as a binary number that forms the check part

Alternative – attach the 1's-complement of the number of 1s

A Berger code:

000000	110
000001	101
000010	101
000011	100
. . .	
100111	010
101000	100
. . .	
111110	001
111111	000

Check part

Can detect all unidirectional errors

$\lceil \log_2(k + 1) \rceil$ check bits for k data bits

Cyclic Codes

Definition: Any cyclic shift of a codeword produces another codeword

A k-bit data word corresponds to a polynomial of degree $k - 1$
Data = 1101001: $D(x) = 1 + x + x^3 + x^6$ (addition is mod 2)

The code has a generator polynomial of degree $r = n - k$
$G(x) = 1 + x + x^3$

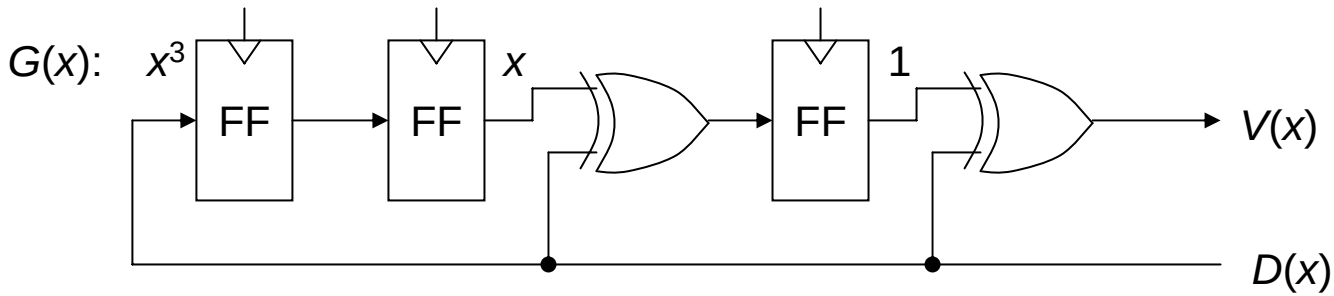
To encode data (1101001), multiply its associated polynomial by $G(x)$

$$1 + x + x^3 + x^6$$
$$\times \quad \underline{1 + x + x^3}$$
$$1 + x + x^3 + x^6 + x + x^2 + x^4 + x^7 + x^3 + x^4 + x^6 + x^9$$
$$1 + x^2 + x^7 + x^9$$
$$1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1$$

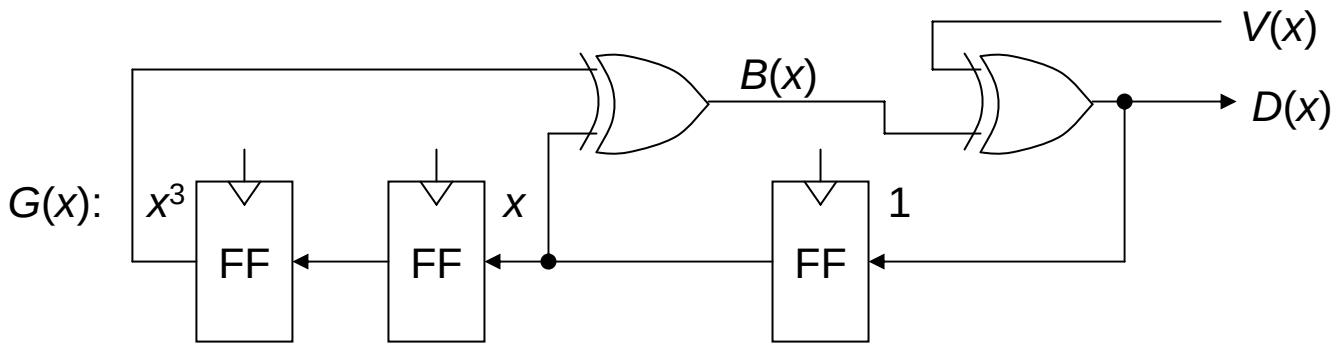
Detects all burst errors of width less than $n - k$

Cyclic Codes: Encoding and Decoding

Encoding: Multiplication by the generator polynomial $G(x)$



Decoding: Division by the generator polynomial $G(x)$



$B(x) = (x + x^3)\, D(x)$ $\qquad$ $V(x) = D(x) + B(x) = (1 + x + x^3)\, D(x)$

Separable Cyclic Codes

Let $D(x)$ and $G(x)$ be the data and generator polynomials

Encoding:

Multiply $D(x)$ by x^{n-k} and divide the result by $G(x)$ to get the remainder polynomial $R(x)$ of degree less than $n - k$

Form the codeword $V(x) = R(x) + x^{n-k}D(x)$, which is divisible by $G(x)$

Example: 7-bit code with 4 data bits and 3 check bits, $G(x) = 1 + x + x^3$

Data = 1 0 0 1, $D(x) = 1 + x^3$

$x^3D(x) = x^3 + x^6 = (x + x^2) \bmod (1 + x + x^3)$

$V(x) = x + x^2 + x^3 + x^6$

Codeword = 0 1 1 | 1 0 0 1

Check part | Data part

The Arithmetic Weight of an Error

Unsigned addition		0010 0111 0010 0001
	+	0101 1000 1101 0011
Correct sum		0111 1111 1111 0100
Erroneous sum		1000 0000 0000 0100

↑ Stage generating an erroneous carry of 1

How a single carry error can lead to an arbitrary number of bit-errors (inversions)

The *arithmetic weight* of an error: Min number of signed powers of 2 that must be added to the correct value to turn it into the erroneous result (contrast with Hamming weight of an error)

	Example 1	Example 2
Correct value	0111 1111 1111 0100	1101 1111 1111 0100
Erroneous value	1000 0000 0000 0100	0110 0000 0000 0100
Difference (error)	$16 = 2^4$	$-32752 = -2^{15} + 2^4$
Min-weight BSD	0000 0000 0001 0000	$^-$1000 0000 0001 0000
Arithmetic weight	1	2
Error type	Single, positive	Double, negative

Codes for Arithmetic Operations

Arithmetic error-detecting codes:

- Are characterized by arithmetic weights of detectable errors
- Allow direct arithmetic on coded operands

We will discuss two classes of arithmetic error-detecting codes, both of which are based on a check modulus *A* (usually a small odd number)

- Product or *AN* codes
  - Represent the value *N* by the number *AN*
- Residue (or inverse residue) codes
  - Represent the value *N* by the pair (*N*, *C*), where *C* is *N* mod *A* or (*N* – *N* mod *A*) mod *A*

Product or *AN* Codes

For odd *A*, all weight-1 arithmetic errors are detected

Arithmetic errors of weight ≥ 2 may go undetected

e.g., the error 32 736 = $2^{15} - 2^5$ undetectable with *A* = 3, 11, or 31

Error detection: check divisibility by *A*

Encoding/decoding: multiply/divide by *A*

Arithmetic also requires multiplication and division by *A*

Product codes are *nonseparate* (*nonseparable*) codes
Data and redundant check info are intermixed

Low-Cost Product Codes

Use low-cost check moduli of the form $A = 2^a - 1$

Multiplication by $A = 2^a - 1$: done by shift-subtract

$(2^a - 1)N = 2^aN - N$

Division by $A = 2^a - 1$: done a bits at a time as follows

Given $y = (2^a - 1)x$, find x by computing $2^a x - y$

. . . xxxx 0000 – . . . xxxx xxxx = . . . xxxx xxxx

Unknown $2^a x$ Known $(2^a - 1)x$ Unknown x

Theorem: Any unidirectional error with arithmetic weight of at most $a - 1$ is detectable by a low-cost product code based on $A = 2^a - 1$

Arithmetic on *AN*-Coded Operands

Add/subtract is done directly: $Ax \pm Ay = A(x \pm y)$

Direct multiplication results in: $Aa \times Ax = A^2ax$

The result must be corrected through division by A

For division, if $z = qd + s$, we have: $Az = q(Ad) + As$

Thus, q is unprotected
Possible cure: premultiply the dividend Az by A
The result will need correction

Square rooting leads to a problem similar to division

$\lfloor \sqrt{A^2x} \rfloor = \lfloor A\sqrt{x} \rfloor$ which is not the same as $A\lfloor \sqrt{x} \rfloor$

Residue and Inverse Residue Codes

Represent N by the pair (N, $C(N)$), where $C(N) = N \bmod A$

Residue codes are *separate* (*separable*) codes

Separate data and check parts make decoding trivial

Encoding: Given N, compute $C(N) = N \bmod A$

Low-cost residue codes use $A = 2^a - 1$

To compute $N \bmod (2^a - 1)$, add a-bit segments of N, modulo $2^a - 1$ (no division is required)

Example: Compute 0101 1101 1010 1110 mod 15
0101 + 1101 = 0011 (addition with end-around carry)
0011 + 1010 = 1101
1101 + 1110 = 1100 The final residue mod 15

Arithmetic on Residue-Coded Operands

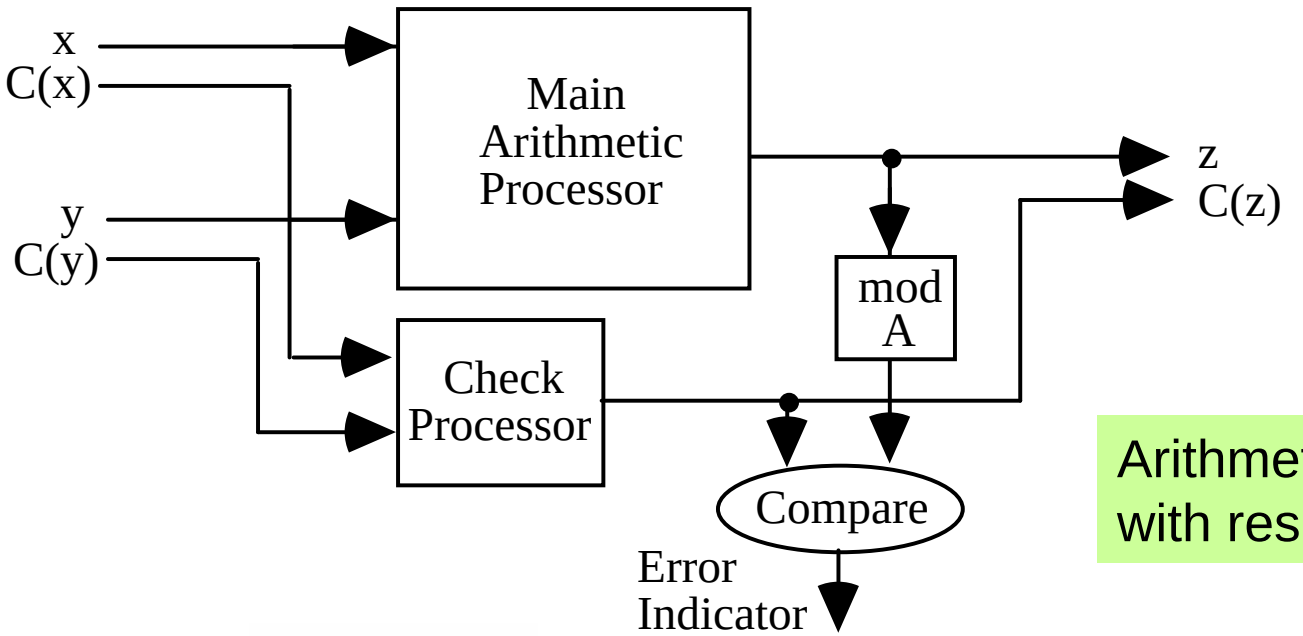
Add/subtract: Data and check parts are handled separately

$$(x, C(x)) \pm (y, C(y)) = (x \pm y, (C(x) \pm C(y)) \bmod A)$$

Multiply

$$(a, C(a)) \times (x, C(x)) = (a \times x, (C(a) \times C(x)) \bmod A)$$

Divide/square-root: difficult



Arithmetic processor with residue 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