

Fundamentos de tolerância a falhas

Códigos de detecção
e correção de erros

Taisy Silva Weber

Redundância de informação

- ✓ adição de informação redundante
 - ✓ para permitir mascaramento (compensação) ou detecção de erros
- ✓ códigos de **detecção** de erros

capacidade de indicar que a informação está incorreta
- ✓ códigos de **correção** de erros

possível recuperar a palavra de código correta a partir da palavra corrompida

Codificação

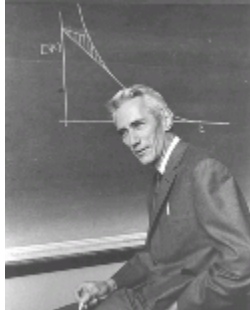
✓ código

- ✓ meio de representar dados usando um conjunto definido de regras
- ✓ codificação implica, geralmente, no aumento do número de bits
- ✓ e armazenamento e processamento extra



redundância

Um pouco de história



Shannon (1916 – 2001) "A Mathematical Theory of Communication", *Bell System Technical Journal* Vol. 27, pp 379-423 and pp 623-656, July, October 1948

Teoria da informação



Hamming (1915-1998) "Error Detecting and Error Correcting Codes", *Bell Systems Technical Journal*, Vol 29, pp 147-160, Apr. 1950

Teoria da codificação

Tipos comuns de erros

- ✓ grudado em
 - ✓ stuck-at-zero
 - ✓ stuck-at-one
- ✓ bit-flip
 - ✓ de 0 para 1
 - ✓ de 1 para 0
- ✓ unidirecional
 - ✓ todos os erros trocam
 - ✓ ou de 0 para 1
 - ✓ ou de 1 para 0
 - ✓ nunca os dois
- ✓ bits adjacentes
 - ✓ exemplos:
 - ✓ curto entre linhas
 - ✓ interferência cruzada
- ✓ rajadas
 - ✓ afeta grupos de bits adjacentes ou próximos

Johnson, Barry. An introduction to the design na analysis of the fault-tolerante systems, cap 1. **Fault-Tolerant System Design**. Prentice Hall, New Jersey, 1996

Unidirecionais versus randômicos

✓ erros randômicos

- ✓ igual probabilidade de trocas de 0s e 1s
- ✓ geralmente falhas transientes
- ✓ tb chamados erros simétricos

✓ erros unidirecionais

- ✓ maior probabilidade em uma única direção
- ✓ tanto falhas permanentes como transientes

✓ exemplos unidirecionais

- ✓ defeito de fonte
- ✓ detalhe de tecnologia
 - ✓ VLSI
 - ✓ flash
- ✓ curto em barramento

B. BOSE AND D. K. PRADHAN. Optimal Unidirectional Error Detecting/Correcting Codes. IEEE TRANS. ON COMPUTERS, VOL. C-31, NO. 6, JUNE 1982


Distância de Hamming

✓ conceito

- ✓ número de posições em que os valores dos bits diferem em duas palavras


✓ exemplos:

✓ 0000 e 0001



distância de 1

✓ 0000 e 0011



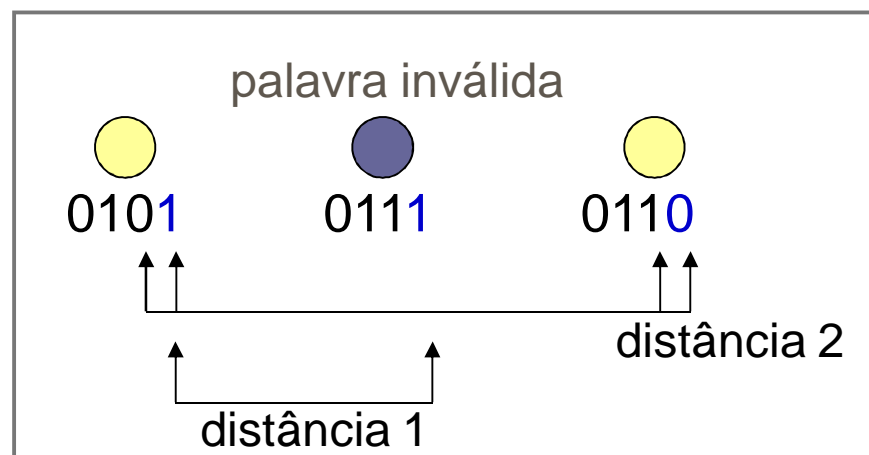
distância de ?

alterando apenas um bit → uma palavra se transforma na outra

Distância de Hamming do código

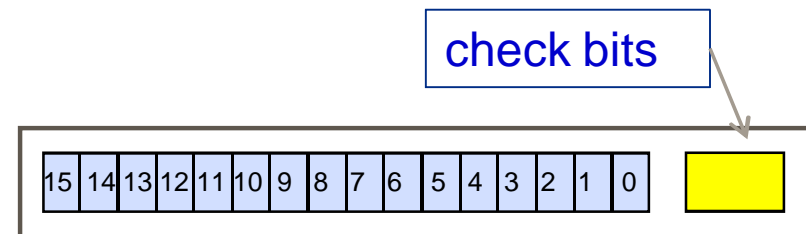
- ✓ distância mínima entre quaisquer duas palavras válidas do código
 - ✓ se um código tem distância 2, qualquer alteração em um bit gera uma palavra inválida neste código
- ✓ códigos com distância 2 permitem detecção de um bit
- ✓ códigos com distância n permitem detecção de $n-1$ bits ou correção de $n-2$ bits

paridade par para 3 bits;
palavra de código com 4 bits



Código separável

- ✓ código separável
 - ✓ informação original é anexada à nova informação para formar a palavra de código
 - ✓ informação nova: **code bits** ou **check bits**
 - ✓ decodificação = remoção de check bits



- ✓ código não separável
 - ✓ requer procedimentos de decodificação mais sofisticados

Exemplos de códigos

- ✓ Códigos de paridade

Single parity
Bit-per-word e bit-per-byte
Interlaced parity
Overlapping parity

- ✓ m-of-n

- ✓ Duplicação

- ✓ Checksums

- ✓ Códigos de Berger

- ✓ Códigos cíclicos

- ✓ Reed-Solomon

Códigos de paridade: uso

- ✓ memórias
- ✓ barramento
- ✓ unidades de E/S
- ✓ transmissão de dados

✓ paridade simples

- ✓ adição de **um bit** a uma palavra de dado
 - ✓ **ímpar**: bit adicionado resulta em um número ímpar de 1s
 - ✓ **par**: o bit adicionado resulta em número par de 1s

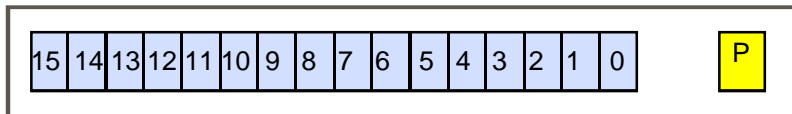
falhas simples (isoladas e únicas): ocorrem de forma independente

distância de Hamming = 2

- ✓ redundância pequena e simples determinação

Bit-per-word

- ✓ implementação de paridade simples
 - ✓ adiciona um bit de paridade por palavra
 - ✓ palavra pode qualquer número de bits



- ✓ capacidade de detecção
 - ✓ single bit

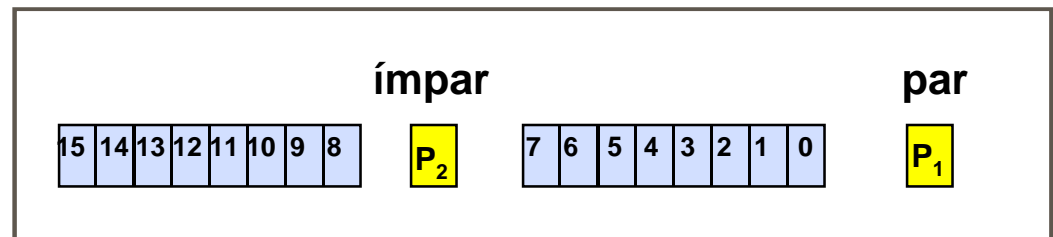
- ✓ erro múltiplo em toda a palavra

palavra transmitida:
qualquer uma

- ✓ stuck-at-one
 - ✓ palavra recebida: 1111 1111 1
 - ✓ se paridade par: erro detectado
 - ✓ se paridade ímpar: erro não detectado
- ✓ stuck-at-zero
 - ✓ palavra recebida: 0000 0000 0
 - ✓ se paridade par: erro não detectado
 - ✓ se paridade ímpar: erro detectado

Bit-per-byte

- ✓ cada **grupo de bits** do dado original é protegido por um bit de paridade
 - ✓ ideal que o grupo tenha um número par de bits
 - ✓ a paridade de um grupo deve ser ímpar e a do outro deve ser par



- ✓ vantagens:
 - ✓ detecta condições de *tudo-1* e de *tudo-0*
- ✓ desvantagens:
 - ✓ ineficiente para outros erros múltiplos

Bit-per-byte: exemplo

- ✓ erro múltiplo afetando toda a palavra
- ✓ stuck-at-one

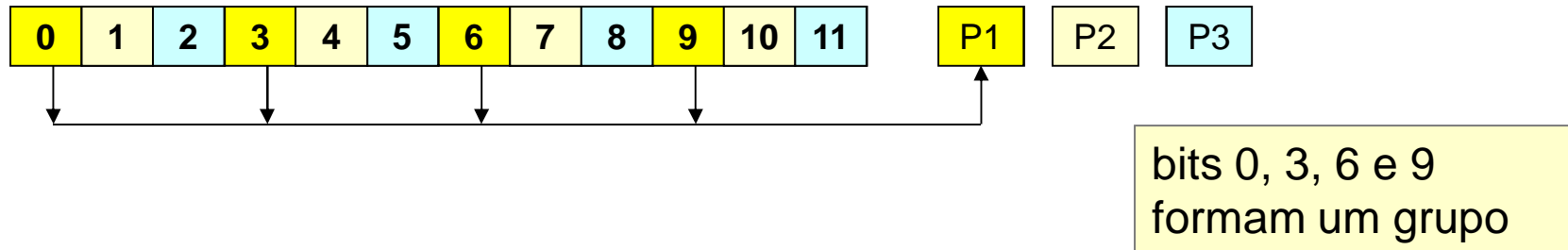
- ✓ palavra recebida: 1111 1111 1 1111 1111 1
 - ímpar
 - par
 - ✓ para o byte com paridade par: erro detectado
 - ✓ para o byte com paridade ímpar: erro não detectado

- ✓ stuck-at-zero

- ✓ palavra recebida: 0000 0000 0 0000 0000 0
 - ímpar
 - par
 - ✓ se paridade par: erro não detectado
 - ✓ se paridade ímpar: erro detectado

neste caso o erro é sempre detectado

Paridade entrelaçada



- ✓ separa uma palavra em grupos com igual tamanho
 - ✓ dois bits adjacentes não podem ficar no mesmo grupo
- ✓ insere um bit de paridade para cada grupo
- ✓ detecta erros em bits adjacentes
 - ✓ aplicação: erros em bits adjacentes com grande probabilidade de ocorrência
 - ✓ exemplo: em barramentos paralelos, bits adjacentes podem facilmente entrar em curto

Paridade sobreposta

- ✓ paridade sobreposta
 - ✓ detecção e localização de erro
 - ✓ possibilidade de correção do erro
 - ✓ se o bit corrompido foi localizado, basta invertê-lo

base para os códigos de correção de Hamming

Paridade sobreposta : exemplo

cada bit de paridade é composto a partir de um grupo de bits



um bit de informação aparece em mais de um grupo

determinação do número de bits de paridade :

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

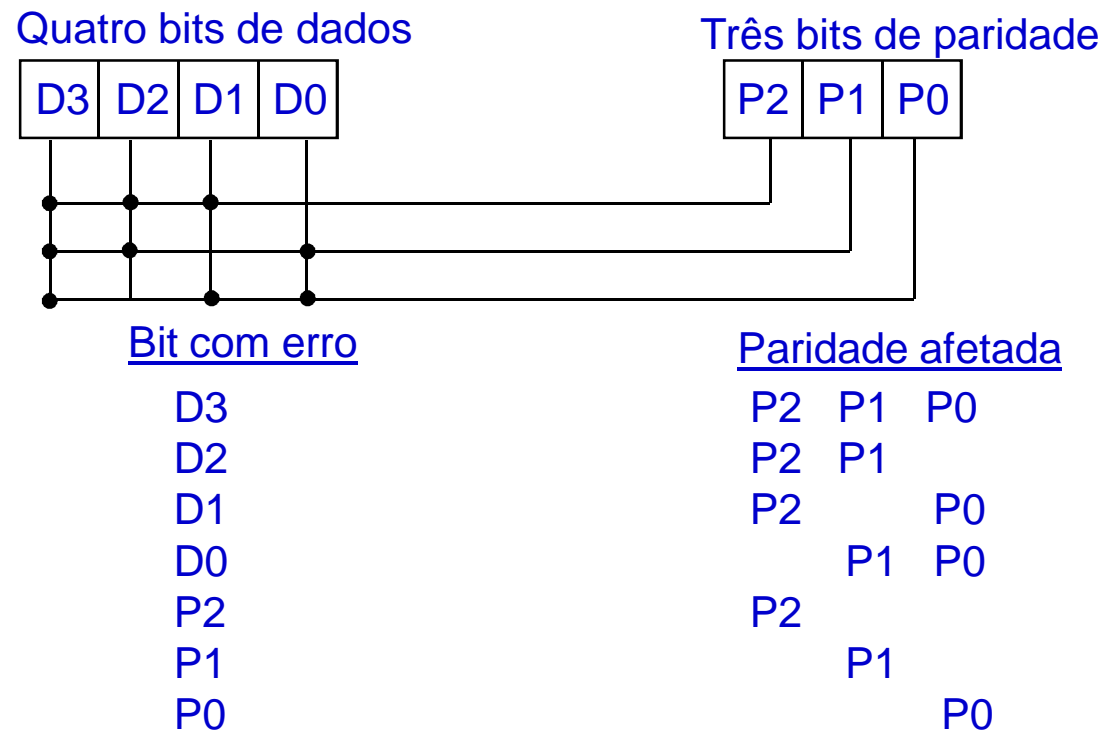
Paridade sobreposta

- ✓ cada bit de paridade é composto a partir de um grupo de bits
- ✓ cada bit do dado deve aparecer em mais de um grupo
- ✓ para cada m bits de dados são necessários k bits de paridade
- ✓ k deve satisfazer a relação $2^k \geq m + k + 1$

k bits de paridade e m bits de informação
 $2^k \geq m + k + 1$

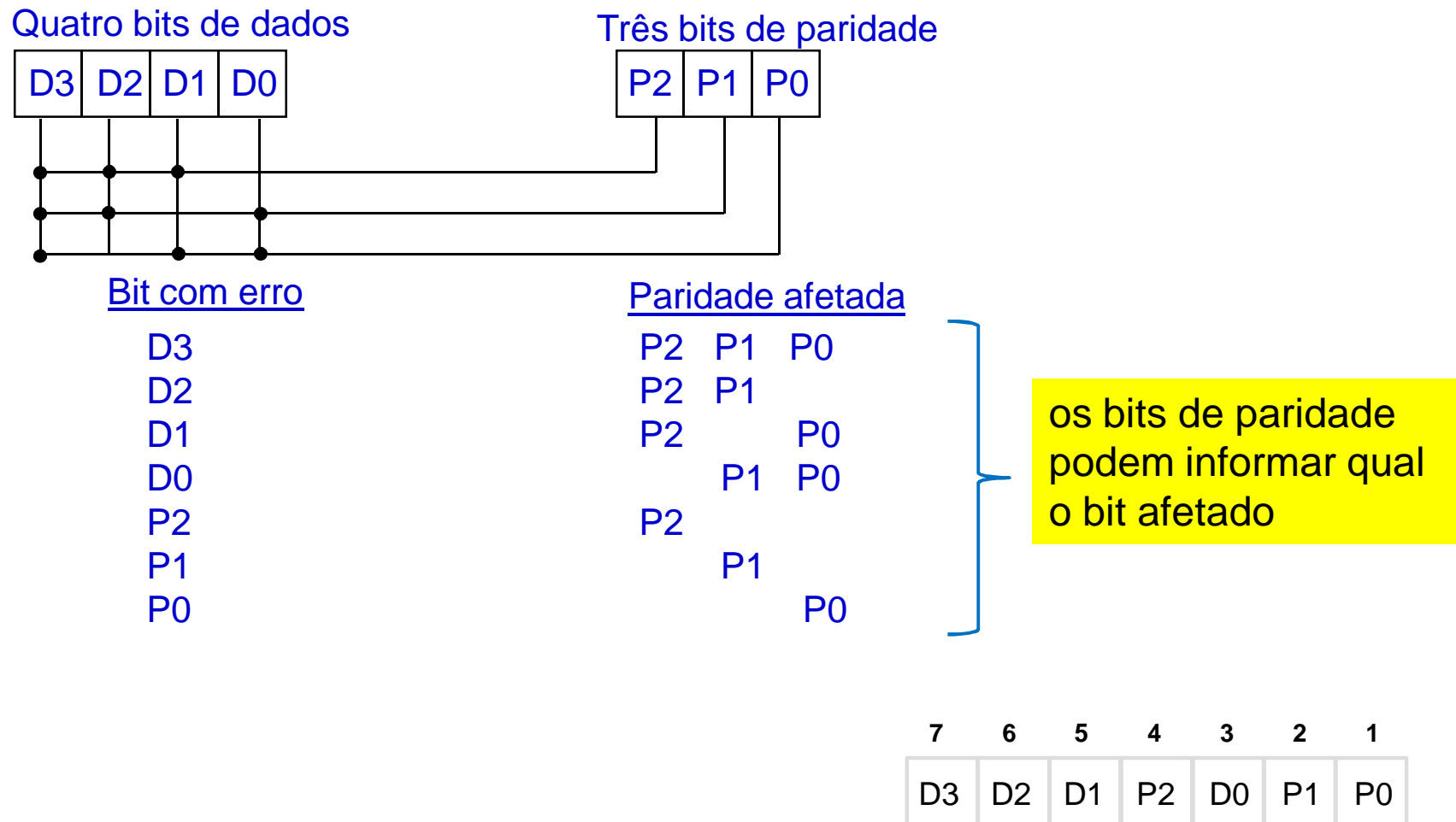
Paridade sobreposta : exemplo

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



neste exemplo redundância de 75%

Paridade sobreposta : exemplo



Código de Hamming

- ✓ código de correção de erros
 - ✓ muito usado em memórias
 - ✓ memória: 60 a 70% das falhas de um sistema
 - ✓ predominância de falhas transitórias
 - ✓ vantagens:
 - ✓ baixo custo (10% a 40% de redundância)
 - ✓ eficiência (rápido na correção)
 - ✓ circuito de correção simples
 - ✓ paridade sobreposta
 - ✓ básico para código de Hamming

m-of-n: definição

- ✓ n bits de comprimento, exatamente m bits em 1
- ✓ detecção de erros simples
 - ✓ palavra resultante com $m+1$ ou $m-1$ bits em 1 (distância 2)
- ✓ simplicidade do conceito
- ✓ alto custo na implementação da codificação, decodificação e detecção de erro
- ✓ i-de-2i
 - ✓ novos i bits anexados aos i bits originais da palavra
 - ✓ os bits anexados produzem palavras $2i$ -bits com exatamente i bits em 1

codificação m-de-n mais usada

codificação e decodificação fácil pois o código é **separável**
detecção de erros simples e **erros múltiplos unidirecionais**

desvantagem:
100% redundância

Decodificação:
remover os bits
anexados da palavra
de código e reter a
informação original

Informação original	Código 3-de-6
000	000 + 111
001	001 + 110
010	010 + 101
011	011 + 100
100	100 + 011
101	101 + 010
110	110 + 001
111	111 + 000

Duplicação

- ✓ código de duplicação de informação
 - ✓ palavra de i bits é duplicada, gerando $2i$ bits
 - ✓ detecção
 - ✓ quando as duas metades da palavra são diferentes
 - ✓ implementação
 - ✓ duplicação de hardware e/ou de tempo
- ✓ vantagens
 - ✓ fácil implementação
 - ✓ fácil obtenção da palavra original
- ✓ desvantagem: requer o dobro de bits

redundância de 100%

Checksums

- ✓ informação anexada a um bloco de dados para permitir detecção de erros
 - ✓ basicamente soma dos itens do bloco de dados
 - ✓ usada para grandes quantidades de dados em transferências ponto a ponto
 - ✓ código separável
- ✓ vários tipos
 - ✓ exemplos: precisão simples, precisão dupla, residual

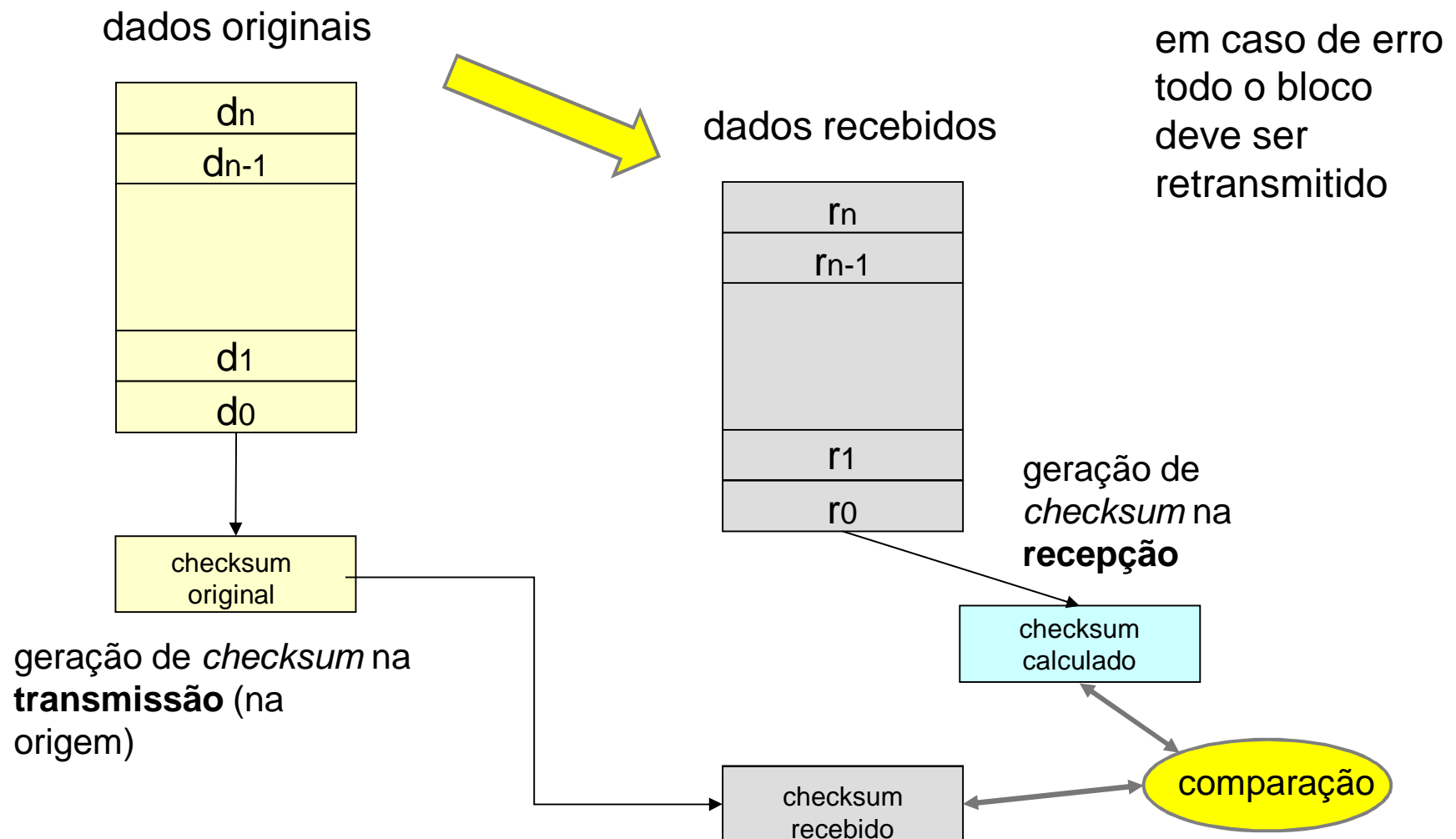
diferenças na forma como a soma é realizada

precisão **simples**:
dados de n bits;
resultado da soma com
 n bits; overflow é
ignorado

precisão **dupla**:
dados de n bits;
checksum com $2n$
bits; overflow é
ignorado

residual: carry-
out da soma é
somado ao
checksum

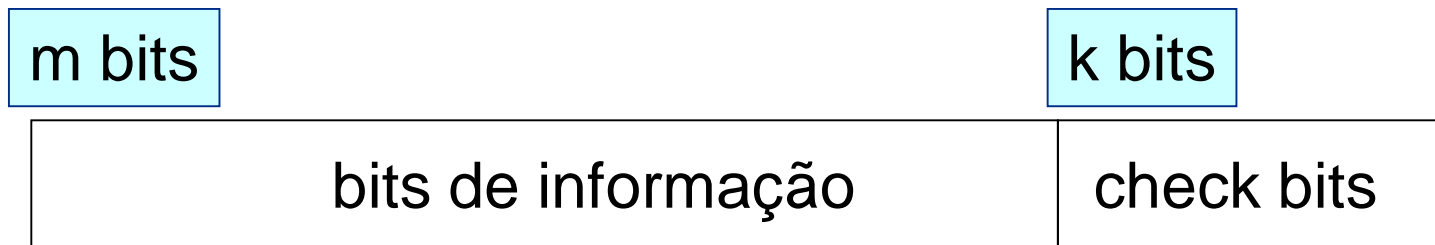
Checksum



Códigos de Berger

- ✓ adição de um conjunto de bits extras
 - ✓ *check bits*
 - ✓ complemento em binário do número de bits em um (1) na área de dados
 - ✓ bits de informação: não mudam
- ✓ vantagens
 - ✓ código é *separável*
 - ✓ detecta todos os erros *unidirecionais*
 - ✓ para a capacidade de detecção permitida, o código de Berger apresenta o menor número de *check bits* comparado com outros códigos separáveis

Códigos de Berger: estrutura



número total de bits

$$n = m + k$$

relação entre K e m

$$k = \lceil \log (m+1) \rceil$$

Se é válida a relação $m = 2^k - 1$
o código é de **tamanho máximo**.

Por exemplo:

$m = 7$ e $k = 3$; $m = 15$ e $k = 4$

Códigos de Berger: exemplo

✓ código de Berger para a palavra 0111010

✓ $m = 7$ bits de informação

✓ $k = \log(7+1) = 3$ check bits

✓ 4 bits em um

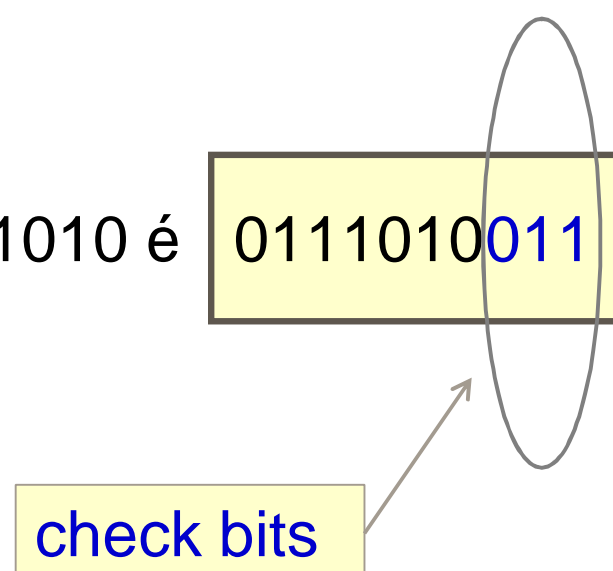
✓ 4 em binário $\rightarrow 100$

✓ complemento de 100 é 011

✓ check bits obtidos = 011

✓ código de Berger para 0111010 é

0111010011



check bits

Códigos cíclicos

✓ CRC

- ✓ uma rotação realizada na palavra de código gera uma nova palavra de código
- ✓ codificação implementada com registradores de deslocamento e realimentação de bits

✓ aplicações

- ✓ comunicação ou transferência sujeita a erros de rajada (*burst errors*)
 - ✓ palavra transmitida serialmente pode ter diversos bits adjacentes corrompidos por uma única perturbação

CRC pode ser usado como uma forma de checksum, sendo transmitido no final de um bloco (o bloco é enviado sem codificação)

CRC: codificação

$$V(x) = D(x)G(x)$$



(código não-separável)

detecta todos erros simples
e erros afetando menos
que $n - k$ bits adjacentes

Grau dos polinômios:

grau de $G(x) \geq (n - k)$

grau de $V(x) = (n - 1)$

grau de $D(x) = (k - 1)$

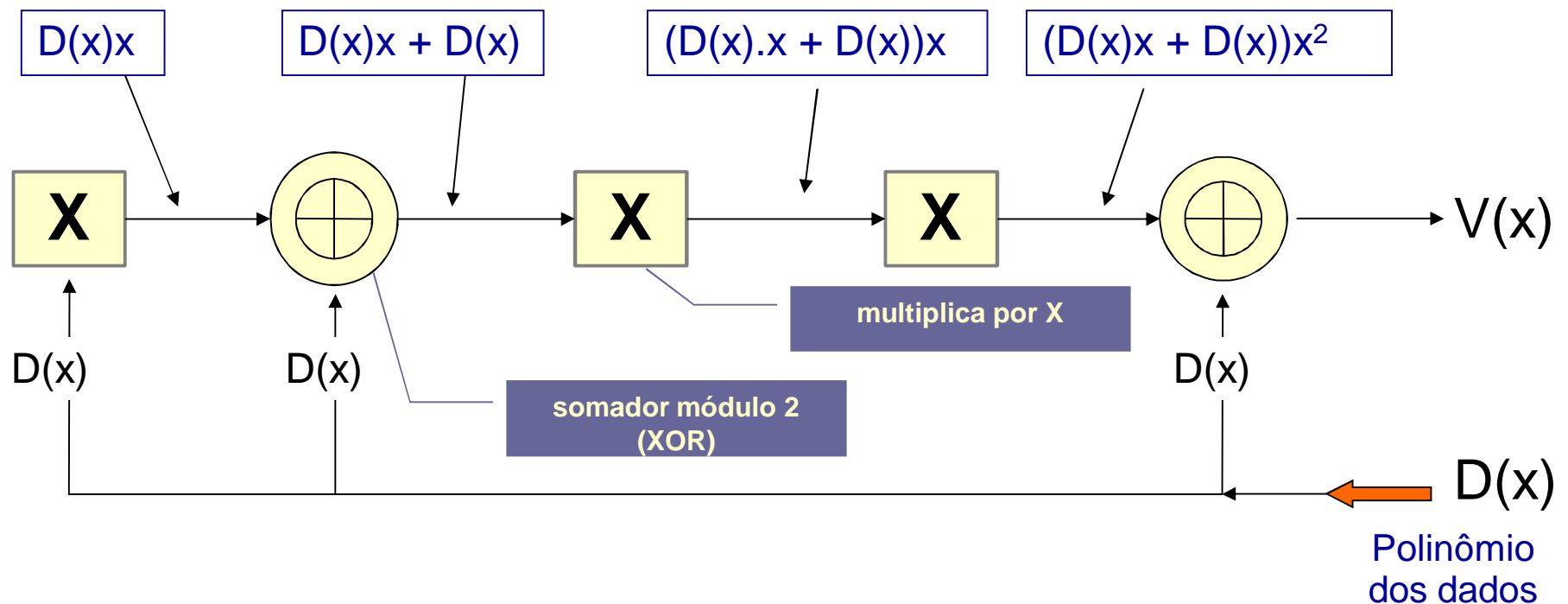
Onde,

n = tamanho do código gerado

k = n^0 de bits da mensagem não codificada

CRC: exemplo

Polinômio Gerador $G(x) = 1 + X + X^3$



$$V(x) = (D(x).x + D(x))x^2 + D(x) = D(x) x^3 + D(x) x^2 + D(x)$$

CRC: decodificação

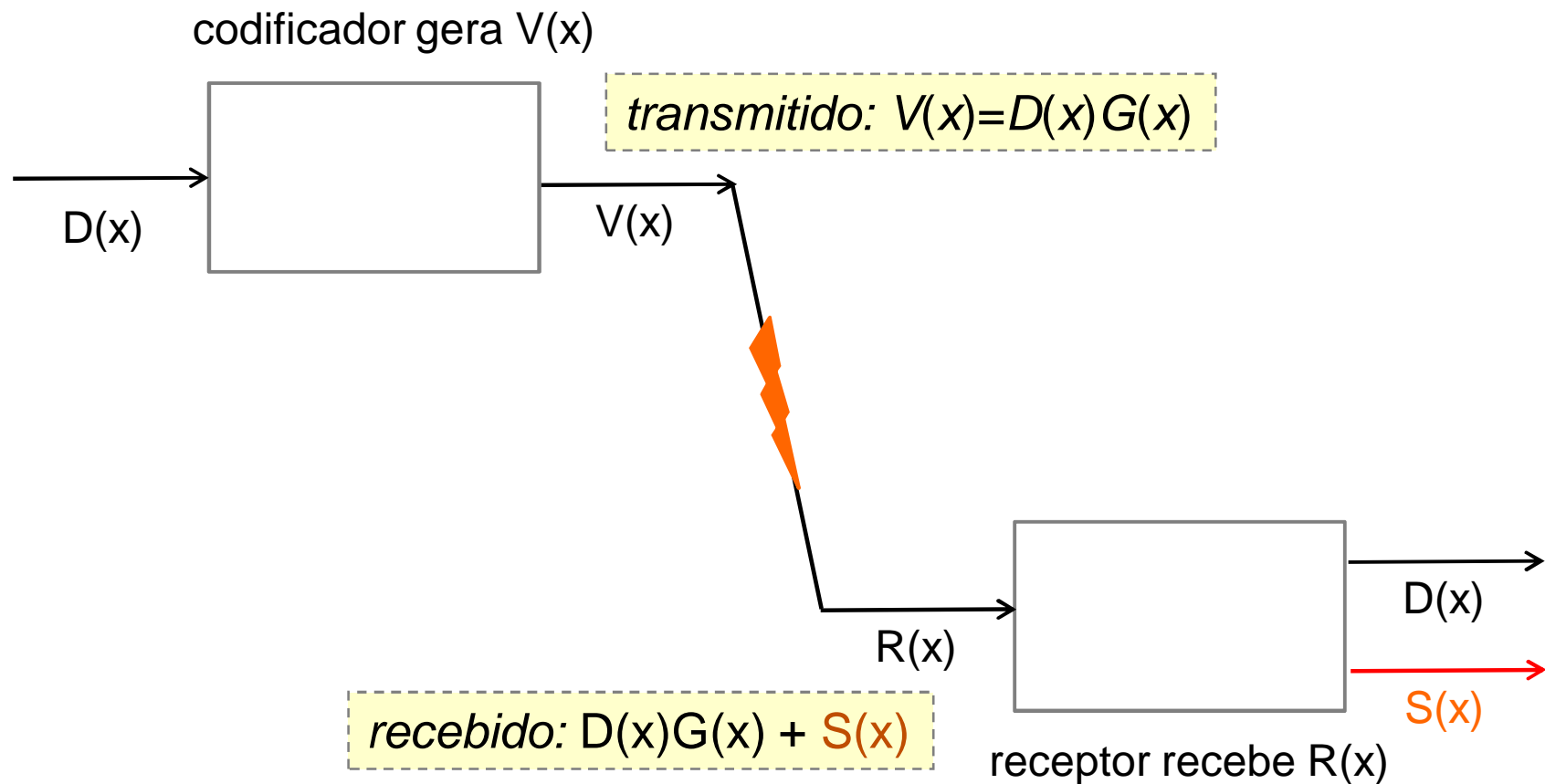
$$R(x) = D(x)G(x) + S(x)$$

The diagram illustrates the components of the CRC equation $R(x) = D(x)G(x) + S(x)$. Four blue arrows point from the terms in the equation to their corresponding labels below:

- $R(x)$ points to "Polinômio codificado"
- $D(x)$ points to "Polinômio dos Dados"
- $G(x)$ points to "Polinômio Gerador"
- $S(x)$ points to "Resto da divisão"

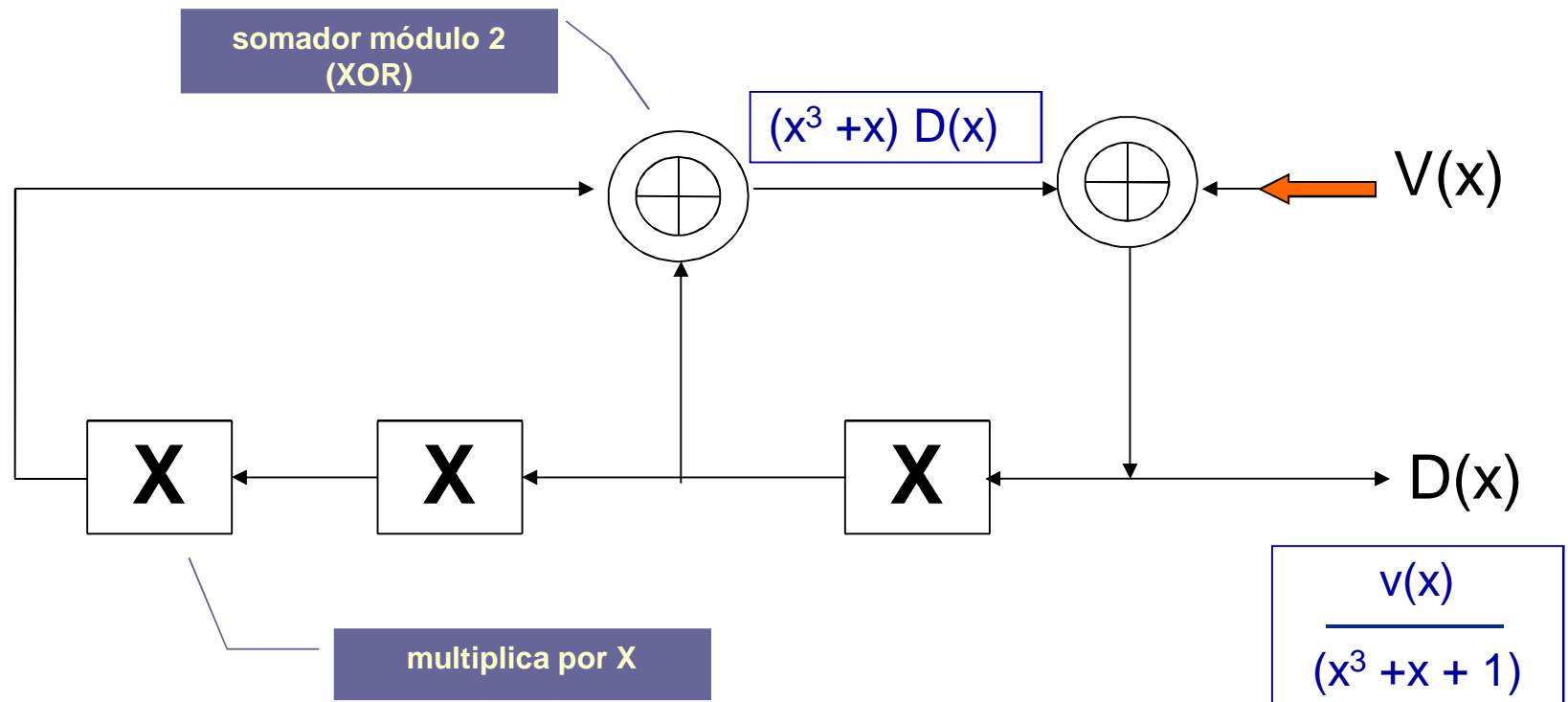
$S(x)$ é zero se não há erros

CRC: operação



CRC: exemplo de decodificação

$$G(x) = X^3 + X + 1$$



CRC: vantagens e desvantagens

- ✓ facilidade de implementação
 - ✓ registradores de deslocamento com realimentação
 - ✓ operações de soma na codificação e na decodificação
- ✓ erros detectados
 - ✓ todos erros simples
 - ✓ erros adjacentes de até $n-k$ bits
- ✓ restrições
 - ✓ código não separável: decodificação não é apenas separar os bits de dados dos de redundância

grau do polinômio gerador

CRC: cobertura de erros

✓ CRC-16:

- ✓ todas falhas simples, duplas, triplas e com núm. ímpar de bits
- ✓ todas as falhas em sequências (burst) de até 16 bits

detecta todos erros simples e erros afetando menos que $n - k$ (ou seja 16) bits adjacentes

- ✓ 99.997% das falhas em sequências de 17 bits
- ✓ 99.998% em sequências de 18 bits ou mais

✓ CRC-32:

- ✓ chance de receber dados corrompidos é de 1 em 4.3 bilhões (0,99999999976744186046511627906977);

W. W. Peterson and D. T. Brown, **Cyclic Codes for Error Detection**. Proceedings of the IEEE, volume 49, pp 228-235, January 1961

Reed-Solomon

- ✓ correção de erros múltiplos
 - ✓ forward error correction (FEC)
- ✓ breve histórico
 - ✓ algoritmo desenvolvido em 1960 por Irving S. Reed e Gustave Solomon, no MIT
 - ✓ artigo:
 - ✓ Polynomial Codes over Certain Finite Fields



reedsolomon.tripod.com

**Irving Stoy Reed &
Gustave Solomon**

Irving Stoy Reed



http://ee.usc.edu/faculty_staff/faculty_directory/reed.htm

Reed-Solomon

- ✓ implementação

- ✓ exigia muitos recursos computacionais, não disponíveis na época da publicação do artigo
- ✓ primeira implementação:
 - ✓ 1982 para CDs



- ✓ inúmeras variações e melhoramentos

- ✓ complementado com vários outros algoritmos para funções específicas

Reed-Solomon: aplicações

- ✓ dispositivos de armazenamento
 - ✓ Compact Disk, DVD, barcodes, etc
- ✓ RAID
- ✓ comunicação Wireless ou móvel
 - ✓ telefones celulares, microwave links, etc ...
- ✓ comunicação com satélites ou naves espaciais
- ✓ televisão digital
- ✓ modems de alta velocidade
 - ✓ ADSL, xDSL, etc ...



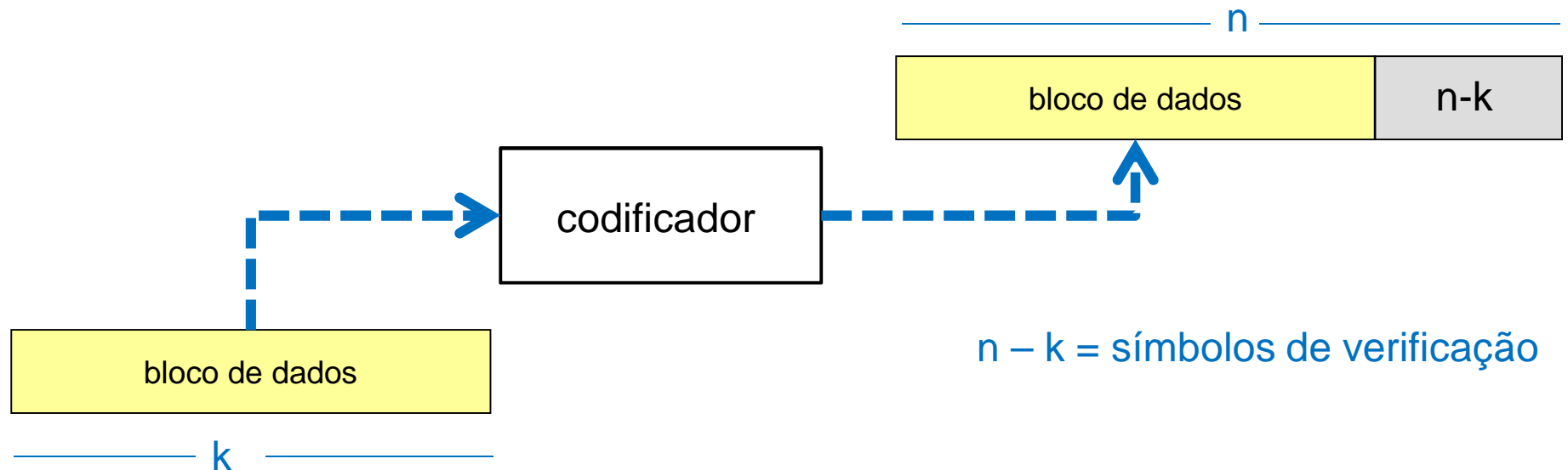
Características

- ✓ generalização

- ✓ não é um algoritmo de codificação específico

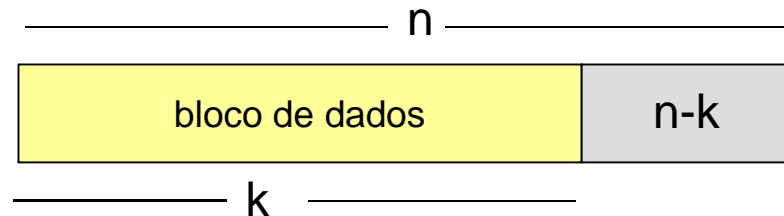
- ✓ código linear de bloco

- ✓ um bloco de entrada de tamanho fixo (k) gera um bloco de saída de tamanho fixo (n)



Características

- ✓ $RS(n, k)$
 - ✓ símbolos (palavras) de s bits
 - ✓ a k símbolos são adicionados alguns símbolos de verificação (ou símbolos de paridade) para fazer um código de n símbolos



$n - k =$ símbolos de verificação

$RS(n, k)$

Reed-Solomon: exemplo

- ✓ RS (255, 223)

RS(n, k)

- ✓ padrão popular

- ✓ $n = 255, k = 223$

- ✓ 223 símbolos de entrada (cada um com 8 bits) são codificados em 255 símbolos de saída

- ✓ $n - k = 32$

- ✓ adicionados 32 bytes de paridade (símbolos de verificação)

- ✓ tamanho da palavra de código = s

- ✓ $n = 2^s - 1$

- ✓ $s = 8 \rightarrow n = 255$

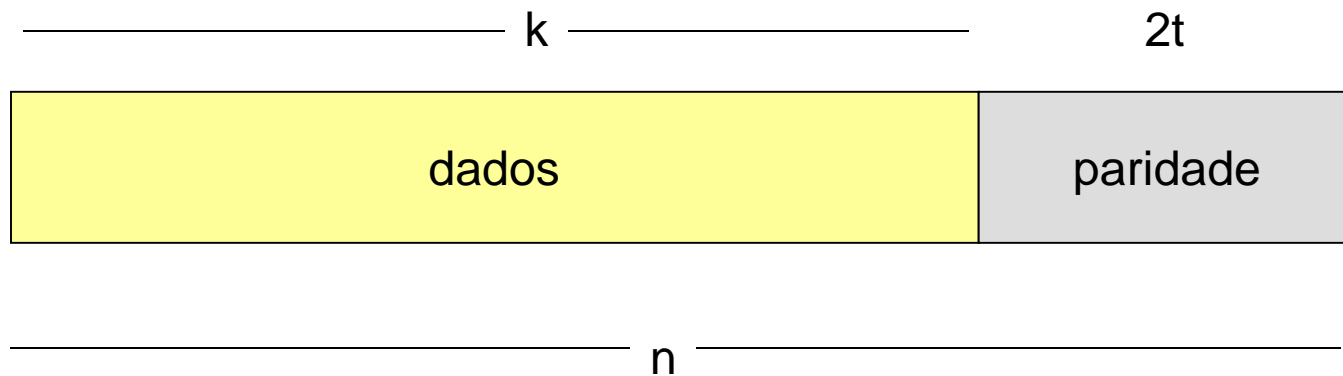
Capacidade de correção

RS(n, k)

RS (255, 223): 223 símbolos de entrada (com 8 bits) são codificados em 255 símbolos de saída

- ✓ $2t = n - k$
- ✓ códigos Reed-Solomon corrigem:
 - ✓ até t erros (símbolos com erros)
 - ✓ ou $2t$ erasures
 - ✓ *erasure* = erro com localização conhecida = omissão

Esquema de bloco

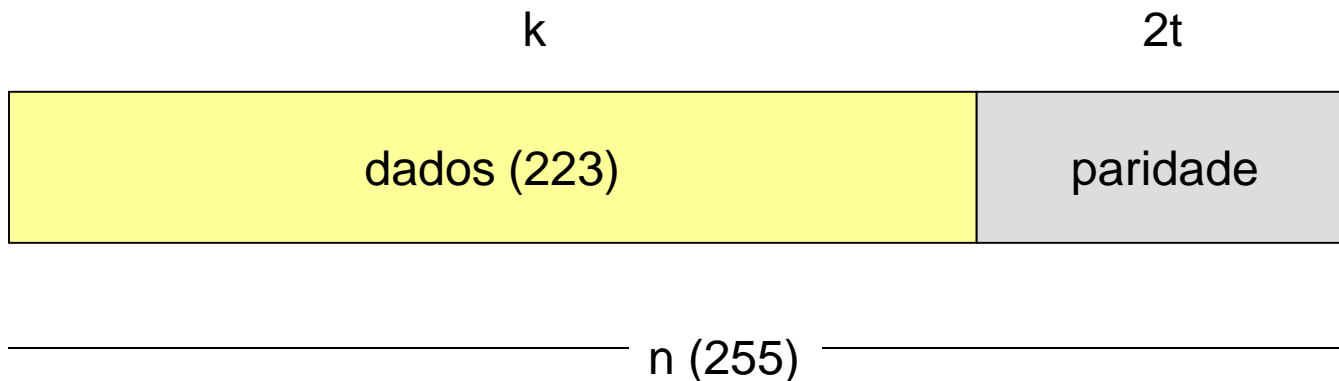


pode corrigir até t erros ou até $2t$ erasures
(erro com localização conhecida - omissões)

Esquema de bloco

exemplo: RS(255,223)

quantas palavras erradas consegue corrigir?



capaz de corrigir até 16 erros (16 bytes) no bloco

se for considerado um erro de **rajada curta de até 8 bits**,
então tem capacidade de corrigir até 16 pequenos erros de
rajada

mais eficiente para rajadas do que para bits isolados

Princípio de funcionamento

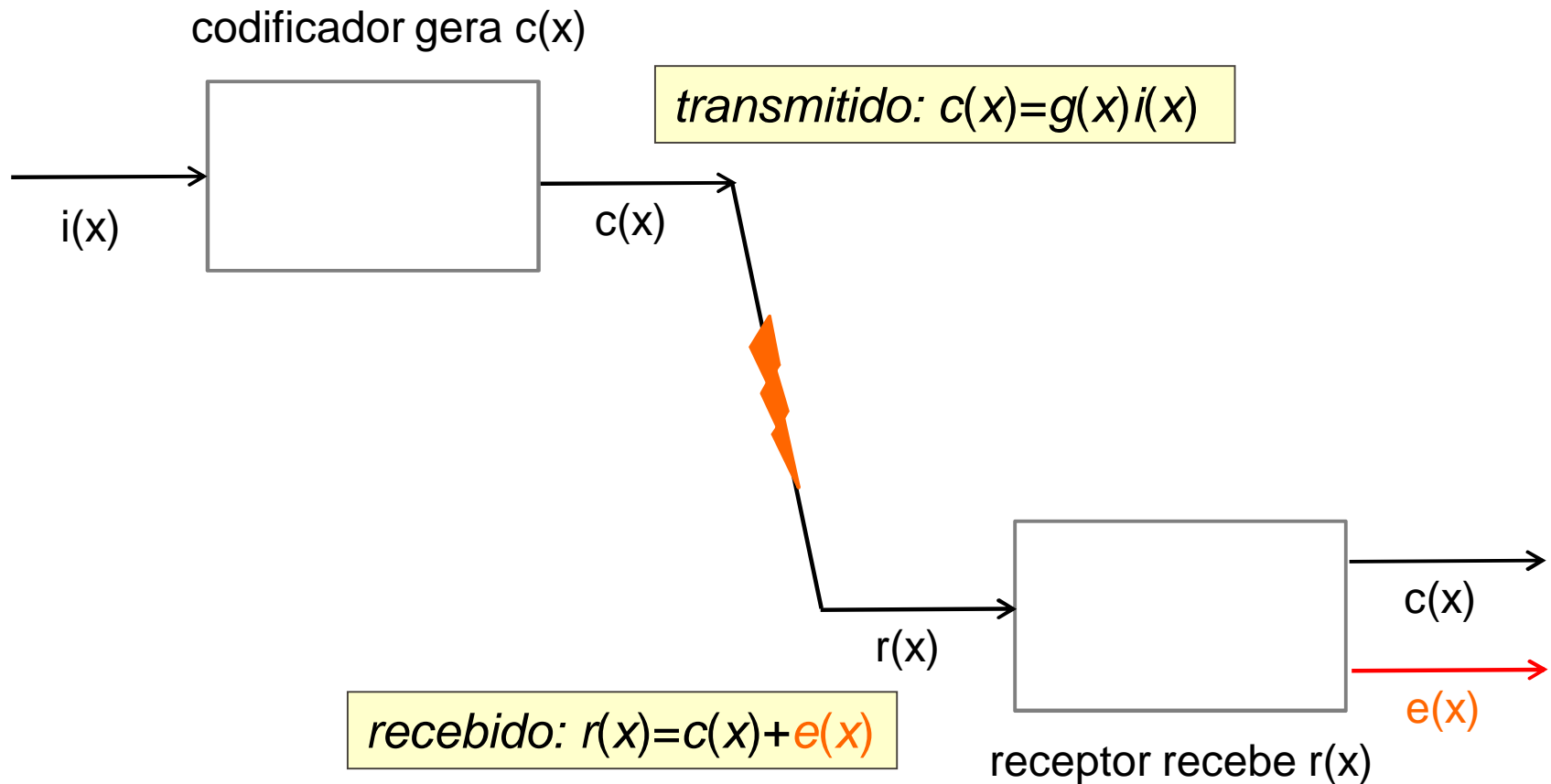
✓ Reed-Solomon

- ✓ baseado em aritmética de campo finito
- ✓ cada palavra codificada é gerada usando um polinômio gerador
- ✓ todas as palavras do código válidas são exatamente divisíveis pelo polinômio gerador

✓ $c(x) = g(x)i(x)$

- ✓ $g(x)$ é o polinômio gerador
- ✓ $i(x)$ é a informação
- ✓ $c(x)$ um código válido

RS funcionamento



Decodificação

- ✓ no receptor

- ✓ recebido $r(x)$

- ✓ pode conter erros

- ✓ $r(x) = c(x) + e(x)$

- ✓ erros: $e(x)$

- ✓ o decodificador identifica a **posição** e a **magnitude** de até **t erros** e os corrige

- ✓ como determinar posição e magnitude (valor)?

- ✓ usando algoritmos específicos

transmitido: $c(x) = g(x)i(x)$

recebido: $r(x) = c(x) + e(x)$

Localização do erro

- ✓ localização do símbolo com erro
 - ✓ resolvendo uma equação com t incógnitas
 - ✓ dois passos:
 - ✓ determinar o polinômio de localização do erro
 - ✓ 2 algoritmos: Berlekamp-Massey e Euclids
 - ✓ Berlekamp-Massey é mais eficiente em software e hardware
 - ✓ Euclids é o mais usado por ser mais fácil de implementar
 - ✓ algoritmo Chien: usado para achar as raízes do polinômio de localização de erro
 - ✓ uma vez localizados os erros, o símbolo correto é determinado resolvendo uma equação com t incógnitas
 - ✓ usando o algoritmo de Forney

Exemplo: CD

- ✓ CD (áudio)
 - ✓ Cross-Interleaved Reed-Solomon Coding (CIRC)
 - ✓ nível C2 de codificação (erros de produção)
 - ✓ 24 palavras de 8 bits, RS(28,24)
 - ✓ 4 check símbolos, corrige 2 bytes
 - ✓ dados são entrelaçados em 109 quadros (frames)
 - ✓ nível C1 de codificação (marca de dedão e arranhões)
 - ✓ depois do entrelaçamento RS(32,28) e novamente entrelaçamento



Implementação

- ✓ geralmente em hardware
 - ✓ devido a capacidade computacional exigida e o tipo de aritmética
- ✓ tipo de aritmética:
 - ✓ aritmética de campo finito ou aritmética de Galois
 - ✓ todos os resultados devem estar no campo

✓ geral

- ✓ Johnson, Barry. An introduction to the design na analysis of the fault-tolerante systems, cap 1. **Fault-Tolerant System Design**. Prentice Hall, New Jersey, 1996

✓ referências para CRC

- ✓ W. W. Peterson and D. T. Brown, **Cyclic Codes for Error Detection**. Proceedings of the IEEE, volume 49, pp 228-235, January 1961.
 - ✓ mostra como calcular as percentagens
- ✓ Andrew S. Tanenbaum. **Computer Networks**. Prentice Hall, Inc. Englewood Cliffs, New Jersey, 1981.

Referências para Reed-Solomon

- ✓ artigo

- ✓ James S. Plank. **A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems.** *Software Practice & Experience*, 27(9), September, 1997, pp. 995-1012

- ✓ alguns links

- ✓ http://www.cs.cmu.edu/afs/cs/project/pscico-guyb/realworld/www/reedsolomon/reed_solomon_codes.html
 - ✓ Martyn Riley and Iain Richardson. An introduction to Reed-Solomon codes: principles, architecture and implementation. 1998
- ✓ <http://www.aero.org/publications/crosslink/winter2002/04.html>
 - ✓ Charles Wang, Dean Sklar, and Diana Johnson. Forward Error-Correction Coding, 2002