Tolerância a falhas – como produzir o serviço correcto na presença de falhas

*Um sistema diz-se tolerante a falhas se a avaria de um componente, é mascarada, não se reflectindo no comportamento externo do sistema*

*"The most certain and effectual check upon errors which arise in the process of computation, is to cause the same computations to be made by separate and independent computers; and this check is rendered still more decisive if they make their computations by different methods."*

Dr. Dionysius Lardner in "Babbages's calculating engine",

Edinburgh Review, July 1834

1965 – Surge pela primeira vez o termo “Failure Tolerance”, W.H. Pierce, Academic Press

*O conceito pretendeu englobar diversas teorias que defendiam a construção de sistemas fiáveis a partir de componentes menos fiáveis, cujas falhas seriam mascaradas pela utilização de componentes redundantes*

1967 – Surge o conceito de “Fault-Tolerant Systems” A. Avizienis,

*Integração da teoria anterior com as técnicas existentes para detecção, localização e recuperação de erros*

Algumas Técnicas de Tolerância a Falhas:

1 – Redundância

"partes do sistema que não são necessárias para que este funcione"

*Todas as técnicas de tolerância a falhas envolvem alguma forma de redundância, com implicações no sistema em termos de tempo de execução e de recursos usados (memória, código, processador(es)).*

- Redundância de informação
- Redundância temporal
- Redundância de hardware
- Redundância de software

## Redundância de informação

Uso de técnicas de codificação para detecção de erros ou mascaramento de falhas

*Bits extra são armazenados ou transmitidos junto com os dados permitindo que erros nos bits de dados sejam detectados e eventualmente corrigidos.*

Ex.

Códigos de paridade – para cada n bits são armazenados n+1

O bit extra indica se o número de bits de dados com valor 1 é par (ou impar)

- permite detectar falhas simples de bits (i. é, que afectam apenas 1 bit de uma palavra)

Redundância de informação ...

Códigos de correcção de erros – Códigos de Hamming

múltiplos bits são adicionados aos dados, cada um associado com um subconjunto de bits

Códigos cíclicos – Cyclic Redundancy Codes (CRC) - código aplicado a blocos de dados e não a "words" independentes

Sequências de bits são associadas a polinómios. Usando um polinómio gerador de grau k, para calcular os bits extra, é possível detectar sequências de bits errados com comprimento menor que k.

## Redundância temporal

A computação é repetida no tempo. Evita o custo de hardware adicional

*Em sistemas onde o tempo não é critico, ou o processador tem tempos de ociosidade, pode ser usada para detectar falhas transitórias de hardware. Resultados diferentes indicam a provável ocorrência de uma falha de hardware.*

## Redundância de hardware

Replicação de componentes físicos.

redundância passiva – mascaramento de falhas

redundância activa – detecção, localização e recuperação

*substituição de módulos*

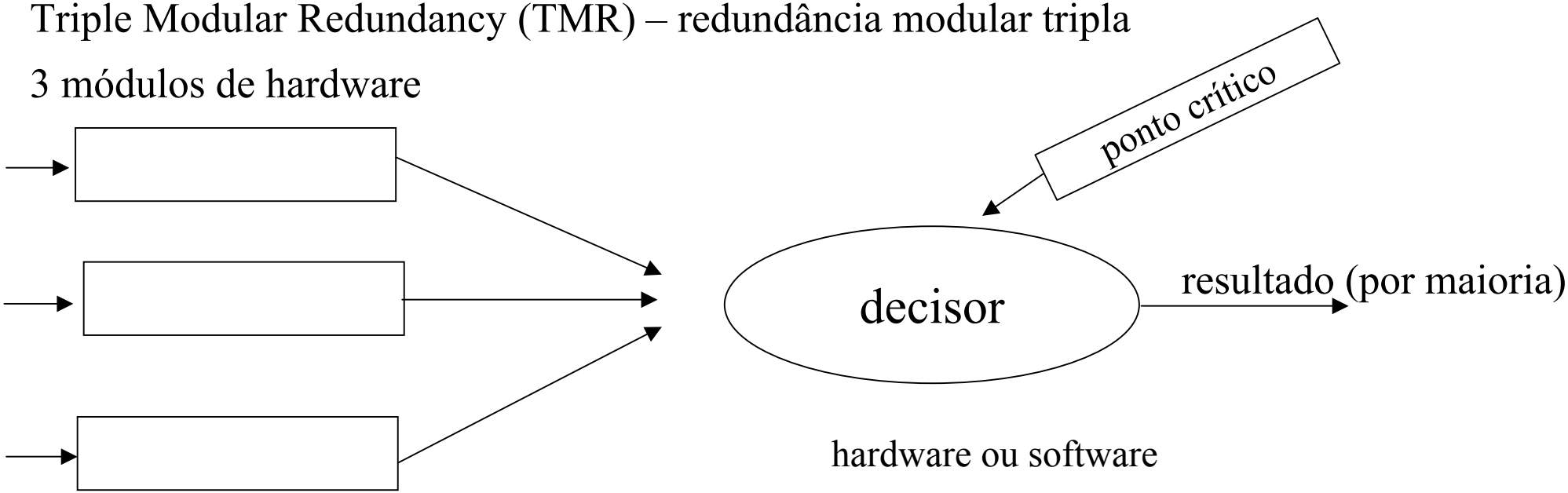
*usada em aplicações de vida longa*

## Redundância de hardware passiva

Elementos redundantes são usados para mascarar falhas. Todos os elementos executam a mesma tarefa e o resultado é determinado por votação

Triple Modular Redundancy (TMR) – redundância modular tripla

3 módulos de hardware




- Suporta apenas uma falha permanente. Usado para falhas transitórias supondo uma de cada vez.

- Computador de bordo do Space Shuttle usa redundância modular com 4 módulos e votação por software

## Redundância de hardware activa

Sistema com vários elementos de processamento, mas em que apenas um funciona de cada vez. Os outros são suplementes.

Se uma falha é detectada na unidade em operação, então esta é retirada e substituída por uma suplente através de um circuito de "switching"

O problema fundamental reside na detecção de falha.

Abordagens:

- testes periódicos
- watchdog timers
- circuitos auto-verificáveis

## Redundância de software

Detecção de falhas de Software

Diversidade (N-version programming)

São implementadas diversas soluções alternativas, idealmente por equipas diferentes, a solução é determinada por votação

- *Sistema implementado no Airbus A300 e A310*

- *No Space Shuttle, é usado um 5º computador, diferente em hardware e em software, que pode substituir os outros 4 em caso de colapso do sistema de redundância modular*

Blocos de recuperação (Recovery blocks)

Várias alternativas são também implementadas, mas a sua execução é sequencial. Após a execução da primeira alternativa, um teste de aceitação verifica os resultados. Se estes não verificam o teste, o segundo bloco é executado e assim por diante...

## Redundância de software

Alguns dos acidentes causados por falhas de software, *míssil Patriot, fogetão Ariane 5, plataforma Spleiner A, Therac-25, ...* poderiam ter sido evitados através de cálculos adicionais de verificação (asserções) relativamente simples

Asserção – condição lógica que permite verificar a correcção dos resultados

Dois tipos de Asserções, com elevada capacidade de detecção de erros, aplicáveis à maioria das operações de manipulação de matrizes:

- **A**lgorithm **B**ased **F**ault **T**olerance - ABFT
(Tolerância a Falhas Baseada nos Algoritmos)

- **R**esult-**C**hecking - RC
(Verificação do Resultado)

- Algorithm Based Fault Tolerance - ABFT
(Tolerância a Falhas Baseada nos Algoritmos)

*O ABFT foi inicialmente proposto para detectar falhas de hardware em arquitecturas do tipo "systolic arrays".*
*Pode detectar falhas de hardware em sistemas de utilização geral e também detectar falhas de software.*

Técnica em que as propriedades do algoritmo são exploradas de forma a introduzir cálculos adicionais mais simples que permitem testar a correcção da computação, quer em pontos intermédios quer no final verificando a validade do resultado.

- **R**esult-**C**hecking - RC
(Verificação do Resultado)

*O **RC** foi inicialmente proposto para detectar falhas de software. Pode também detectar falhas de hardware.*

Técnica que verifica o resultado de uma computação usando um cálculo simplificado.

## ABFT - como funciona?

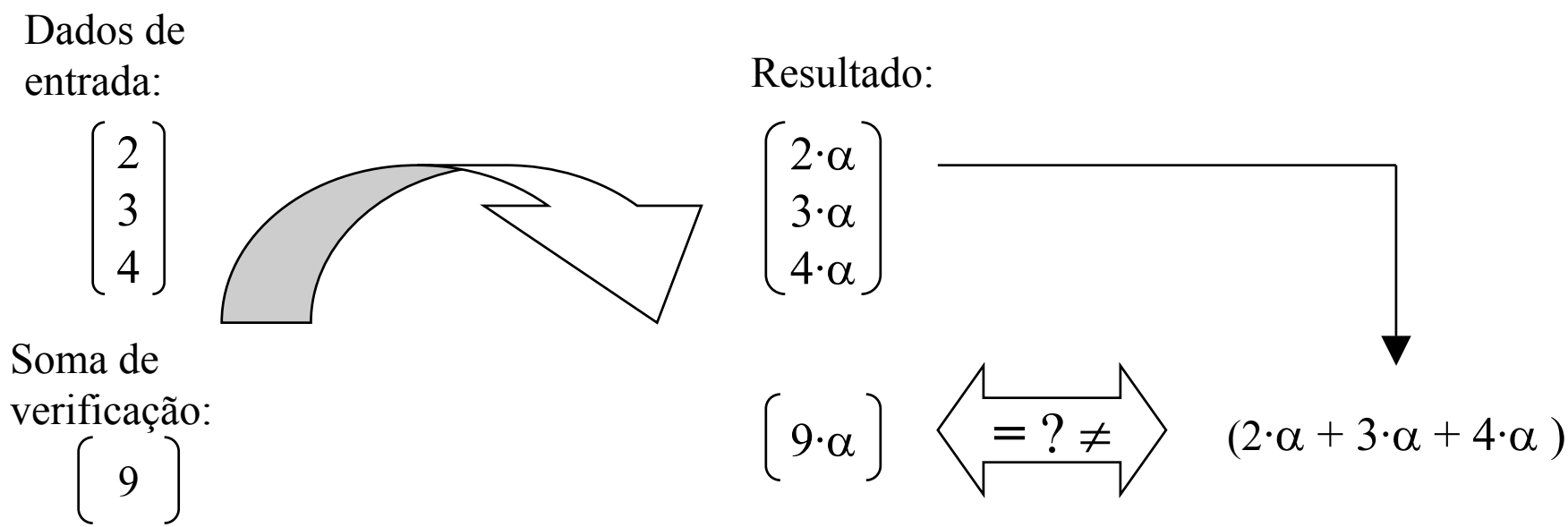
1 - Os dados de entrada são codificados através, por exemplo, da adição de somas de verificação.

2 - O algoritmo é redesenhado de forma a operar os dados codificados produzindo resultados codificados.

3 - O cálculo está correcto se as somas de verificação calculadas a partir dos resultados são iguais às produzidas pelo algoritmo.

*ABFT - um exemplo*

## Produto de um vector por uma constante, α , usando ABFT




*Produto de matrizes – cálculo: $O(n^3)$, ABFT: $O(n^2)$*

**Vantagens do ABFT:**

- Elevada cobertura de erros
- Baixo "overhead"
- Permite localização e recuperação de falhas

**Limitações do ABFT:**

- Operações que não preservam a codificação !
- Difícil de implementar em código já existente
- Localização só funciona se não houver propagação de falhas
- Disponível apenas para alguns algoritmos

(operações lineares sobre matrizes, transformadas de Fourier, ...

**R**esult-**C**hecking - RC (Verificação do Resultado)

*Verificar o resultado de um cálculo é muitas vezes mais fácil que efectuar o cálculo*

*Por exemplo, calculada a solução de um sistema de equações lineares, podemos verificar a solução, substituindo as variáveis pelos valores encontrados para a solução*

Dada uma função $f$: x $\longrightarrow$ y

Um **verificador para f** é um algoritmo que, dados **f**, **x** e **y**:

1 - Verifica se f(x) = y

2 - Tem um tempo de execução inferior a qualquer possível programa que calcule f(x)

3 - Devolve a resposta correcta (à questão 1) com elevada probabilidade.

*RC - um exemplo*

Multiplicação de Matrizes: **A, B** ⟶ **C**

$\cong O(n^3)$

**Verificador de Freivalds** para o produto de matrizes:

$$\left( C \right) * \left[ r \right] \stackrel{?}{=} \left( A \right) * \left[ \left( B \right) * \left[ r \right] \right]$$

$\cong O(n^2)$

( com **A**, **B**, **C** $\in R^{n x n}$ e **r** um vector de valores aleatórios $\in R^{n x 1}$ )

*Multiplicamos ambos os membros da equação por um vector de valores aleatórios, r, no final comparamos os dois vectores resultantes*

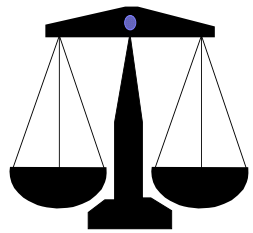
**Vantagens do RC:**

- Elevada cobertura de erros
- Baixo "overhead"
- Baseado no problema, sendo independente do algoritmo usado
- Verificador extremo a extremo (end-to-end)
- Fácil de implementar
(pode ser aplicado a bibliotecas já definidas)

**Limitações do RC:**

- Não permite localização e recuperação de falhas
- Disponível apenas para alguns problemas

Em resumo:



ABFT e RC são técnicas com:

. elevada cobertura de erros

. baixo overhead

Limitações:

. disponíveis apenas para alguns problemas

. ABFT permite localização e recuperação de erros

. RC é independente do algoritmo

.RC é mais fácil de implementar que o ABFT

Problemas associados ao ABFT e ao RC:

## 1 – Erros de arredondamento

Devido à representação finita de valores reais, dois cálculos apesar de matematicamente equivalentes podem produzir resultados diferentes.

Diferentes sequências de arredondamento vão dar origem a valores diferentes nos bits menos significativos do resultado.

Dois valores reais comparam-se usando um intervalo de tolerância (threshold):

. Esse intervalo estabelece o que consideramos um erro significativo

. Pode calcular-se previamente através de técnicas de análise numérica

Problemas associados ao ABFT e ao RC:

## 2 – Verificar o verificador

O verificador (RC, ABFT, ou outra qualquer asserção) é geralmente muito mais simples do que o cálculo a verificar e portanto a probabilidade da haver uma falha de software na verificação será muito pequena.

Para falhas de hardware:

Asserções robustas – técnica que permite detectar falhas de hardware na execução do teste de verificação assim como falhas que afectem os dados fora do âmbito de cobertura das asserções por exemplo antes da codificação dos dados ser efectuada, ou depois de o resultado ter sido calculado e verificado.

## Asserções robustas

Suponhamos um bloco de código que:

i) recebe dados que eventualmente poderão ser validades por asserções
ii) executa alguns cálculos sobre esses dados
iii) produz resultados cuja correcção poderá ser validada por asserções

Pretende-se:

- detectar falhas na execução das asserções
  *.dupla execução de cada asserção*
  *.uso de um número de verificação ("magic number") que funciona como um mecanismo de controlo de fluxo de execução*

- proteger os dados fora do âmbito de cobertura das asserções
  *. através de um código de detecção de erros, um CRC por exemplo*

# *Asserções Robustas*

. Dados de entrada
. Proteger / Codificar dados (que sejam usados em asserções)

. **Asserções de entrada**

. Cálculos
. Proteger resultados com CRC

. **Asserções de saída**

. SE ( todos os testes executados com sucesso )
enviar para o exterior
(resultados + CRC + número mágico)

. número mágico = 0

para cada asserção *i* *(com i=1,2,...)*

. Número mágico = 0
...

. **Asserção *i* (1)**
. SE ( sucesso1 ) ENTÃO
número mágico += número (2*i*-1)

. **Asserção *i* (2)**
. SE ( sucesso2 ) ENTÃO
número mágico += número (2*i*)

*Um número mágico correcto garante uma elevada probabilidade de que o resultado foi verificado com sucesso pelas várias asserções*

*Um CRC correcto garante que os dados não foram corrompidos depois de verificados*