

# Sistemas Distribuídos - ESP625

Prof<sup>a</sup> Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação  
Instituto Federal de Ciência e Tecnologia da Bahia  
Campus de Feira de Santana

*carolsoko@ifba.edu.br*

December 12, 2024

# DEPENDABILITY - Confiança no Funcionamento do Sistema

- 1 Tolerância a Falhas
- 2 DEPENDABILITY - Confiança no Funcionamento do Sistema
- 3 O que é Falha, Erro e Defeito?
  - FAULT (Falha)
  - ERROR (Erro)
  - FAILURE (Defeito)
- 4 Classificação das Falhas em Sistemas Distribuídos
  - Falhas Físicas X Falhas Humanas
  - Falhas de Outras Naturezas
  - Falhas Maliciosas
  - Falhas em Sistemas Críticos
- 5 Desafios da Tolerância a Falhas
  - Sistemas de Tempo Real Críticos e Não Críticos
  - Falhas são inevitáveis, mas o colapso do sistema não!
- 6 Técnicas para Alcançar DEPENDABILITY

# Tolerância a Falhas em Sistemas Distribuídos

# Tolerância a Falhas

## Definição:

A tolerância a falhas é a propriedade que garante a correta e eficiente operação de um sistema apesar da ocorrência de falhas em qualquer um dos seus componentes, ou unidades.

# Tolerância a Falhas

## Definição:

A tolerância a falhas é a propriedade que garante a correta e eficiente operação de um sistema apesar da ocorrência de falhas em qualquer um dos seus componentes, ou unidades.

Um SISTEMA DISTRIBUÍDO TOLERANTE A FALHAS é um sistema que garante a entrega das mensagens e o funcionamento de suas aplicações mesmo em presença de falhas, que são inevitáveis em redes heterogêneas como as redes inerentes aos Sistemas Distribuídos Modernos. [TANENBAUM e STEEN 2007]

# Tolerância a Falhas

## Definição:

As falhas e indisponibilidades são realmente intrínsecas aos sistemas computacionais.

# Tolerância a Falhas

## Definição:

As falhas e indisponibilidades são realmente intrínsecas aos sistemas computacionais.

É possível construir o sistema de forma que ele continue funcionando como um todo, mesmo que alguns de seus componentes falhem. Chamamos um sistema construído desta forma de tolerante a falhas.

# Tolerância a Falhas

## Definição:

As falhas e indisponibilidades são realmente intrínsecas aos sistemas computacionais.

É possível construir o sistema de forma que ele continue funcionando como um todo, mesmo que alguns de seus componentes falhem. Chamamos um sistema construído desta forma de tolerante a falhas.

A ideia é que o sistema continue produzindo as saídas para as quais foi construído, mesmo que alguns de seus componentes não estejam funcionando como deveriam.



# Tolerância a Falhas

## Definição:

Na verdade, a definição mais aceita de tolerância a falhas vai um pouco além, e diz ainda que o DESEMPENHO DO SISTEMA COM COMPONENTES FALHOS DEVE PERMANECER EM NÍVEIS ACEITÁVEIS. [Coulouris et al. 2013]

# Tolerância a Falhas

## Definição:

Na verdade, a definição mais aceita de tolerância a falhas vai um pouco além, e diz ainda que o DESEMPENHO DO SISTEMA COM COMPONENTES FALHOS DEVE PERMANECER EM NÍVEIS ACEITÁVEIS. [Coulouris et al. 2013]

É claro que para um sistema continuar funcionando, nem todos os seus componentes podem estar falhos, pelo menos alguns devem estar funcionando corretamente e devem, de alguma maneira, fazer a função dos componentes falhos.

# DEPENDABILITY

## Confiança no Funcionamento do Sistema

# DEPENDABILITY

## DEPENDABILITY:

A dependabilidade surge da necessidade de se poder depender de um sistema. Devido à evolução tecnológica e à crescente dependência humana da tecnologia, de um sistema poder ter a propriedade de se poder depender do mesmo.

# DEPENDABILITY

## DEPENDABILITY:

A dependabilidade surge da necessidade de se poder depender de um sistema. Devido à evolução tecnológica e à crescente dependência humana da tecnologia, de um sistema poder ter a propriedade de se poder depender do mesmo.

Um pouco mais formalmente, dizemos que um componente X depende de um componente Y se a corretude do comportamento de X depende da corretude do componente Y e dizemos também que um componente é “dependável” (DEPENDABLE) na medida em que outros podem depender dele.

# DEPENDABILITY

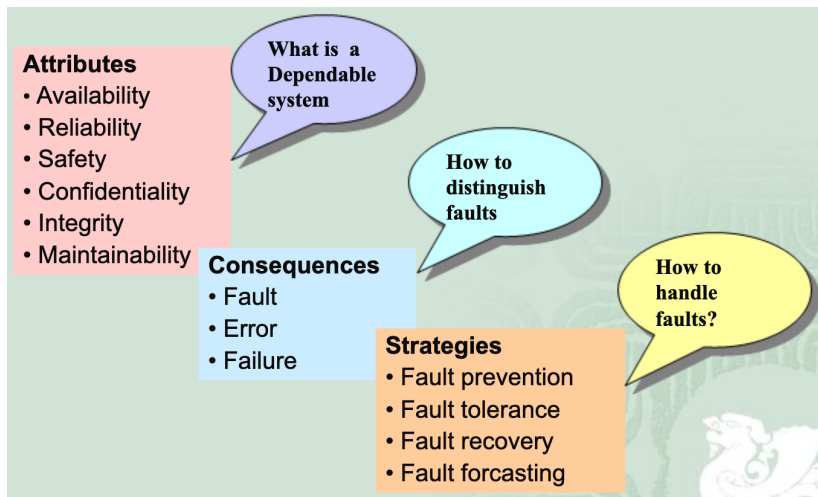
## DEPENDABILITY:

A dependabilidade surge da necessidade de se poder depender de um sistema. Devido à evolução tecnológica e à crescente dependência humana da tecnologia, de um sistema poder ter a propriedade de se poder depender do mesmo.

Um pouco mais formalmente, dizemos que um componente X depende de um componente Y se a corretude do comportamento de X depende da corretude do componente Y e dizemos também que um componente é “dependável” (DEPENDABLE) na medida em que outros podem depender dele.

A dependabilidade é essencial aos componentes de Sistemas Distribuídos.

# Resumindo... DEPENDABILITY



# ATRIBUTOS

## DEPENDABILITY:

A dependabilidade não é uma coisa só, ela engloba diferentes atributos. São eles:

- Disponibilidade (AVAILABILITY) - capacidade do sistema de estar disponível para uso, levando em conta inclusive a ocorrência de falhas que interrompam o serviço e a posterior recuperação com a retomada do serviço;
- Confiabilidade (RELIABILITY) - capacidade do sistema de oferecer o serviço correto continuamente, sem falhas e interrupções;



# ATRIBUTOS

## DEPENDABILITY:

A dependabilidade não é uma coisa só, ela engloba diferentes atributos. São eles:

- **Segurança (Safety)** - capacidade do sistema de evitar consequências catastróficas;
- **Segurança (Security)** - inclui os diversos atributos clássicos de segurança, como capacidade de oferecer confidencialidade e integridade;
- **Manutenibilidade (Maintainability)** - facilidade de manutenção e realizar alterações.

# ATRIBUTOS

Atributo	Significado
Dependabilidade ( <i>dependability</i> )	qualidade do serviço fornecido por um dado sistema
Confiabilidade ( <i>reliability</i> )	capacidade de atender a especificação, dentro de condições definidas, durante certo período de funcionamento e condicionado a estar operacional no início do período
Disponibilidade ( <i>availability</i> )	probabilidade do sistema estar operacional num instante de tempo determinado; alternância de períodos de funcionamento e reparo
Segurança ( <i>safety</i> )	probabilidade do sistema ou estar operacional e executar sua função corretamente ou descontinuar suas funções de forma a não provocar dano a outros sistema ou pessoas que dele dependam
Segurança ( <i>security</i> )	proteção contra falhas maliciosas, visando privacidade, autenticidade, integridade e irrepudiabilidade dos dados

## RELIABILITY:

A confiabilidade  $R(t)$  é a capacidade de atender a especificação, dentro de condições definidas, durante certo período de funcionamento e condicionado a estar operacional no início do período.

## RELIABILITY:

A confiabilidade  $R(t)$  é a capacidade de atender a especificação, dentro de condições definidas, durante certo período de funcionamento e condicionado a estar operacional no início do período.

Confiabilidade (Reliability) é a medida mais usada em sistemas críticos, ou seja nos seguintes tipos de sistemas: Sistemas em que mesmo curtos períodos de operação incorreta são inaceitáveis e Sistemas em que reparo é impossível.

## RELIABILITY:

A confiabilidade  $R(t)$  é a capacidade de atender a especificação, dentro de condições definidas, durante certo período de funcionamento e condicionado a estar operacional no início do período.

Confiabilidade (Reliability) é a medida mais usada em sistemas críticos, ou seja nos seguintes tipos de sistemas: Sistemas em que mesmo curtos períodos de operação incorreta são inaceitáveis e Sistemas em que reparo é impossível.

A definição acima implica algumas condições essenciais:

## RELIABILITY:

- **Especificação:** sem uma especificação do sistema, não é possível determinar se o sistema está operando conforme esperado ou não, quando mais formal e completa a especificação, mais fácil estabelecer essa condição. Não é possível estabelecer se um sistema sem especificação é confiável ou não.

## RELIABILITY:

- **Especificação:** sem uma especificação do sistema, não é possível determinar se o sistema está operando conforme esperado ou não, quando mais formal e completa a especificação, mais fácil estabelecer essa condição. Não é possível estabelecer se um sistema sem especificação é confiável ou não.
- **Condições Definidas:** as condições de funcionamento do sistema devem ser bem definidas. Um exemplo simples são as condições ambientais de temperatura e umidade. Outro exemplo são os dados ou estímulos de entrada que o sistema deve processar.

## RELIABILITY:

- **Período de Funcionamento:** o tempo de missão deve ser conhecido. O tempo de missão de uma viagem espacial é diferente do tempo de missão de um voo comercial doméstico. Um sistema pode ser altamente confiável para 12 horas de operação e depois necessitar de um longo período de repouso e reparo.



## RELIABILITY:

- **Período de Funcionamento:** o tempo de missão deve ser conhecido. O tempo de missão de uma viagem espacial é diferente do tempo de missão de um voo comercial doméstico. Um sistema pode ser altamente confiável para 12 horas de operação e depois necessitar de um longo período de repouso e reparo.
- **Estado Operacional no Início do Período:** não é possível falar em confiabilidade de sistemas que já partem operando com defeitos.

## RELIABILITY:

Confiabilidade é uma medida de probabilidade, pois a ocorrência de falhas é um fenômeno aleatório. Confiabilidade não pode ser confundida com disponibilidade.

## RELIABILITY:

Confiabilidade é uma medida de probabilidade, pois a ocorrência de falhas é um fenômeno aleatório. Confiabilidade não pode ser confundida com disponibilidade.

**UM SISTEMA PODE SER DE ALTA CONFIABILIDADE E DE BAIXA DISPONIBILIDADE.**

## RELIABILITY:

Confiabilidade é uma medida de probabilidade, pois a ocorrência de falhas é um fenômeno aleatório. Confiabilidade não pode ser confundida com disponibilidade.

**UM SISTEMA PODE SER DE ALTA CONFIABILIDADE E DE BAIXA DISPONIBILIDADE.**

Um exemplo seria um avião que precisa de reparos e manutenção nos intervalos de vôo.

## AVAILABILITY:

A Disponibilidade também é uma medida de probabilidade. Disponibilidade é a probabilidade do sistema estar operacional num instante de tempo determinado.

## AVAILABILITY:

A Disponibilidade também é uma medida de probabilidade. Disponibilidade é a probabilidade do sistema estar operacional num instante de tempo determinado.

É o atributo mais usado em sistemas de missão crítica. Sistemas de consulta de base de dados on-line, servidores de rede, servidores de páginas web, são alguns exemplos de sistemas onde alta disponibilidade é requerida. [TANENBAUM e STEEN 2007]

## AVAILABILITY:

DISPONIBILIDADE NÃO PODE SER CONFUNDIDA COM  
CONFIABILIDADE.

## AVAILABILITY:

**DISPONIBILIDADE NÃO PODE SER CONFUNDIDA COM CONFIABILIDADE.**

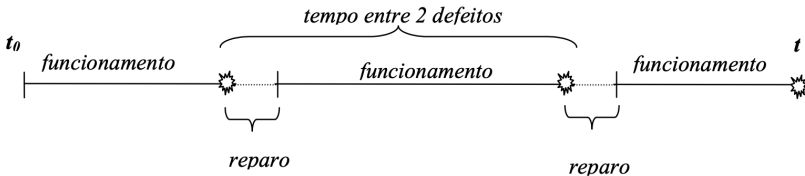
Um sistema pode ser altamente disponível mesmo apresentando períodos de inoperabilidade, quando está sendo reparado, desde que esses períodos sejam curtos e não comprometam a qualidade do serviço. Disponibilidade está muito relacionada com o tempo de reparo do sistema. Diminuir o tempo de reparo resulta em um aumento de disponibilidade.



## AVAILABILITY:

**DISPONIBILIDADE NÃO PODE SER CONFUNDIDA COM CONFIABILIDADE.**

Um sistema pode ser altamente disponível mesmo apresentando períodos de inoperabilidade, quando está sendo reparado, desde que esses períodos sejam curtos e não comprometam a qualidade do serviço. Disponibilidade está muito relacionada com o tempo de reparo do sistema. Diminuir o tempo de reparo resulta em um aumento de disponibilidade.



## SAFETY:

Segurança (SAFETY) é a probabilidade do sistema ou estar operacional, e executar sua função corretamente, ou descontinuar suas funções de forma a não provocar dano a outros sistemas ou pessoas que dele dependam.

## SAFETY:

Segurança (SAFETY) é a probabilidade do sistema ou estar operacional, e executar sua função corretamente, ou descontinuar suas funções de forma a não provocar dano a outros sistemas ou pessoas que dele dependam.

Segurança é a medida da capacidade do sistema de se comportar de forma livre de falhas (FAIL-SAFE).

## SAFETY:

Segurança (SAFETY) é a probabilidade do sistema ou estar operacional, e executar sua função corretamente, ou descontinuar suas funções de forma a não provocar dano a outros sistemas ou pessoas que dele dependam.

Segurança é a medida da capacidade do sistema de se comportar de forma livre de falhas (FAIL-SAFE).

Um exemplo seria um sistema de transporte ferroviário onde os controles de um trem providenciam sua desaceleração e parada automática quando não mais conseguirem garantir o seu funcionamento correto. Em um sistema FAIL-SAFE, ou a saída é correta ou o sistema é levado a um estado seguro. [Coulouris et al. 2013]

## Demais Atributos:

Outros atributos importantes de um sistema são: comprometimento do desempenho (PERFORMABILITY), manutenibilidade (MANTENABILITY) e testabilidade (TESTABILITY). Todas essas medidas são igualmente representadas por uma probabilidade.

## Demais Atributos:

Outros atributos importantes de um sistema são: comprometimento do desempenho (PERFORMABILITY), manutenibilidade (MANTENABILITY) e testabilidade (TESTABILITY). Todas essas medidas são igualmente representadas por uma probabilidade.

Comprometimento do desempenho (PERFORMABILITY) está relacionada à queda de desempenho provocado por falhas, onde o sistema continua a operar, mas degradado em desempenho.

## Demais Atributos:

Mantenabilidade (MANTENABILITY) significa a facilidade de realizar a manutenção do sistema, ou seja, a probabilidade que um sistema com defeitos seja restaurado a um estado operacional dentro de um período determinado. Restauração envolve a localização do problema, o reparo físico e a colocação em operação.

## Demais Atributos:

Mantenabilidade (MANTENABILITY) significa a facilidade de realizar a manutenção do sistema, ou seja, a probabilidade que um sistema com defeitos seja restaurado a um estado operacional dentro de um período determinado. Restauração envolve a localização do problema, o reparo físico e a colocação em operação.

Testabilidade (TESTABILITY) é a capacidade de testar certos atributos internos do sistema ou facilidade de realizar certos testes.



## Demais Atributos:

Mantenabilidade (MANTENABILITY) significa a facilidade de realizar a manutenção do sistema, ou seja, a probabilidade que um sistema com defeitos seja restaurado a um estado operacional dentro de um período determinado. Restauração envolve a localização do problema, o reparo físico e a colocação em operação.

Testabilidade (TESTABILITY) é a capacidade de testar certos atributos internos do sistema ou facilidade de realizar certos testes.

Quanto maior a testabilidade, melhor a manutenabilidade, e por consequência menor o tempo que o sistema não estará disponível devido a reparos.

# O que é Falha, Erro e Defeito?

Antes de mais nada, precisamos definir exatamente o que é uma “FALHA”. Poucos termos são usados na computação com tantos significados tão distintos.

# O que é Falha, Erro e Defeito?

Antes de mais nada, precisamos definir exatamente o que é uma “FALHA”. Poucos termos são usados na computação com tantos significados tão distintos.

É possível generalizar, dizendo que uma falha corresponde a um comportamento incorreto, fora da especificação. Desta forma, o sistema ou não produz resultado algum, ou produz resultado que não está entre os permitidos ou desejáveis.

# O que é Falha, Erro e Defeito?

Antes de mais nada, precisamos definir exatamente o que é uma “FALHA”. Poucos termos são usados na computação com tantos significados tão distintos.

É possível generalizar, dizendo que uma falha corresponde a um comportamento incorreto, fora da especificação. Desta forma, o sistema ou não produz resultado algum, ou produz resultado que não está entre os permitidos ou desejáveis.

No jargão da área de tolerância a falhas são identificados três “estágios” diferentes referentes à falha de um componente de um sistema, em inglês eles são: **Fault, Error e Failure**.

## FAULT (Falha):

Tudo começa com uma falha de um componente do sistema. Pode ser uma falha de *hardware* ou um *bug de software*. Um componente pode falhar, mas esta falha não ser “ativada”.

## FAULT (Falha):

Tudo começa com uma falha de um componente do sistema. Pode ser uma falha de *hardware* ou um *bug de software*. Um componente pode falhar, mas esta falha não ser “ativada”.

Por exemplo, pode ser que um procedimento específico de um programa tenha sido implementado incorretamente, mas se aquele procedimento não for executado, então a falha fica lá, dormente.

## FAULT (Falha):

O termo **Fault** se refere justamente a este tipo de incorreção. Em português, a comunidade de tolerância a falhas, adotam duas palavras distintas para FAULT em português: alguns traduzem como "falta", outros como "falha" mesmo.

## FAULT (Falha):

O termo **Fault** se refere justamente a este tipo de incorreção. Em português, a comunidade de tolerância a falhas, adotam duas palavras distintas para FAULT em português: alguns traduzem como "falta", outros como "falha" mesmo.

Veja que em inglês "TOLERÂNCIA A FALHAS" é "FAULT TOLERANCE", ou seja, o que se tolera é uma FAULT, que neste contexto podemos traduzir para "falha".



## FAULT (Falha):

O termo **Fault** se refere justamente a este tipo de incorreção. Em português, a comunidade de tolerância a falhas, adotam duas palavras distintas para FAULT em português: alguns traduzem como "falta", outros como "falha" mesmo.

Veja que em inglês "TOLERÂNCIA A FALHAS" é "FAULT TOLERANCE", ou seja, o que se tolera é uma FAULT, que neste contexto podemos traduzir para "falha".

No fim das contas, o que importa é compreender que FAULT corresponde a uma falha latente, intrínseca de um componente, que pode ou não se manifestar.

## ERROR (Erro):

Quando a falha (FAULT) se manifesta, ocorre o chamado erro (ERROR). Neste caso, em português todos concordam em traduzir como "erro".

## ERROR (Erro):

Quando a falha (FAULT) se manifesta, ocorre o chamado erro (ERROR). Neste caso, em português todos concordam em traduzir como "erro".

Retomando o exemplo anterior, se o procedimento implementado incorretamente é executado e produz uma saída fora de sua especificação, então acontece um erro, que é a manifestação da falha (FAULT).

## ERROR (Erro):

Veja que mesmo que o procedimento com falha seja executado, é possível que as saídas produzidas sejam (por alguma coincidência) corretas, neste caso não há erro.

## ERROR (Erro):

Veja que mesmo que o procedimento com falha seja executado, é possível que as saídas produzidas sejam (por alguma coincidência) corretas, neste caso não há erro.

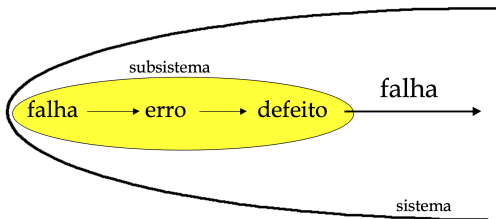
Um exemplo bastante impressionante de falha e erro é o que aconteceu com alguns modelos do processador Pentium da Intel, no começo dos anos 1990, que incrivelmente, computadores produzidos iguais davam ou não davam erro. Acontecia o erro em algumas máquinas e em outras não. Por algum motivo.

## FAILURE (Defeito):

Uma falha (FAULT) pode se manifestar como um erro (ERROR) e, por sua vez, o erro produzido pelo componente com defeito pode se propagar para a saída geral do sistema, causando então seu defeito (FAILURE).

## FAILURE (Defeito):

Uma falha (FAULT) pode se manifestar como um erro (ERROR) e, por sua vez, o erro produzido pelo componente com defeito pode se propagar para a saída geral do sistema, causando então seu defeito (FAILURE).



## FAILURE (Defeito):

O Defeito (FAILURE) do sistema corresponde ao seu colapso: o sistema como um todo produz resultado incorreto, fora da sua especificação.

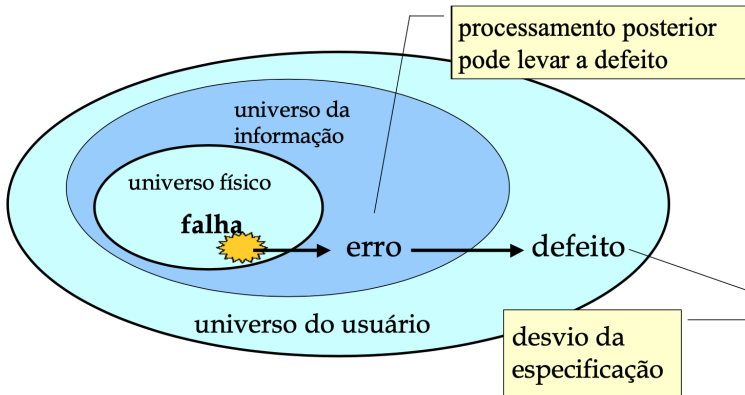


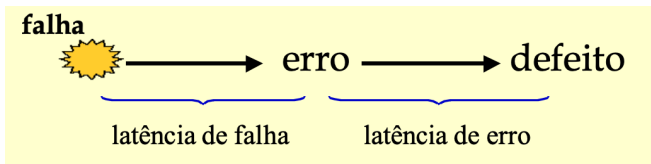
## FAILURE (Defeito):

O Defeito (FAILURE) do sistema corresponde ao seu colapso: o sistema como um todo produz resultado incorreto, fora da sua especificação.

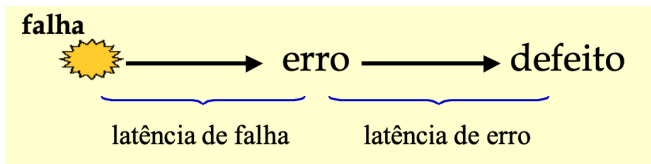
É possível que o erro de um componente não se propague até a saída do sistema. Por exemplo, se a saída de um módulo aritmético tem uma falha (FAULT) que se manifesta em erro (ERROR), mas este erro é então multiplicado por zero, o erro é neutralizado e não vai se propagar até a saída do sistema.

falha (falta) → erro → defeito



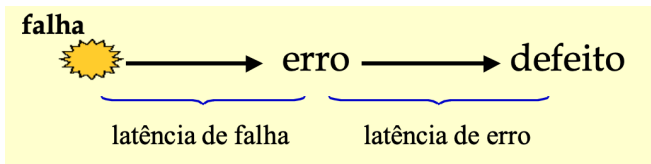


## Latência de Falha e de Erro:



### Latência de Falha e de Erro:

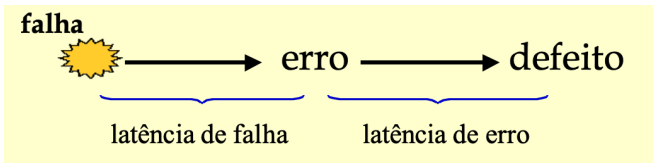
Define-se latência de falha como o período de tempo desde a ocorrência da falha até a manifestação do erro devido àquela falha.



### Latência de Falha e de Erro:

Define-se latência de falha como o período de tempo desde a ocorrência da falha até a manifestação do erro devido àquela falha.

Define-se latência de erro como o período desde a ocorrência do erro até a manifestação do defeito devido àquele erro.



### Latência de Falha e de Erro:

Define-se latência de falha como o período de tempo desde a ocorrência da falha até a manifestação do erro devido àquela falha.

Define-se latência de erro como o período desde a ocorrência do erro até a manifestação do defeito devido àquele erro.

Baseando-se no modelo de 3 universos, o tempo total desde a ocorrência da falha até o aparecimento do defeito é a soma da latência de falhas e da latência de erro.

## Classificação das Falhas:

As falhas aparecem geralmente classificadas em falhas físicas, aquelas afetam os componentes, e falhas humanas. Falhas humanas compreendem falhas de projeto e falhas de interação.

## Classificação das Falhas:

As falhas aparecem geralmente classificadas em falhas físicas, aquelas afetam os componentes, e falhas humanas. Falhas humanas compreendem falhas de projeto e falhas de interação.

As principais causas de falhas são problemas de especificação, problemas de implementação, componentes defeituosos, imperfeições de manufatura, fadiga dos componentes físicos, além de distúrbios externos como radiação, interferência eletromagnética, variações ambientais (temperatura, pressão, umidade) e também problemas de operação.



## Classificação das Falhas:

Além da causa, para definir uma falha considera-se também:

- Natureza: falha de hardware, falha de software, de projeto, de operação, etc.
- Duração ou persistência: permanente ou temporária (intermitente ou transitória)
- Extensão: falha restrita a um módulo, falha global
- Valor: determinado ou indeterminado no tempo

## Classificação das Falhas:

Vem crescendo a ocorrência daquelas falhas provocadas por interação humana maliciosa, ou seja, por aquelas ações que visam propositadamente provocar danos aos sistemas.

## Classificação das Falhas:

Vem crescendo a ocorrência daquelas falhas provocadas por interação humana maliciosa, ou seja, por aquelas ações que visam propositadamente provocar danos aos sistemas.

Essas falhas e suas consequências são tratados por técnicas de segurança computacional (security) e não por técnicas de tolerância a falhas.

## Classificação das Falhas:

Vem crescendo a ocorrência daquelas falhas provocadas por interação humana maliciosa, ou seja, por aquelas ações que visam propositalmente provocar danos aos sistemas.

Essas falhas e suas consequências são tratados por técnicas de segurança computacional (security) e não por técnicas de tolerância a falhas.

Deve ser considerado, entretanto, que um sistema tolerante a falhas deve ser também seguro a intrusões e ações maliciosas.

## Classificação das Falhas:

Falhas de software e também de projeto são consideradas atualmente o mais grave problema em computação crítica.

## Classificação das Falhas:

Falhas de software e também de projeto são consideradas atualmente o mais grave problema em computação crítica.

Sistemas críticos, tradicionalmente, são construídos de forma a suportar falhas físicas. Assim é compreensível que falhas não tratadas, e não previstas, sejam as que mais aborreçam, pois possuem uma grande potencial de comprometer a confiabilidade e disponibilidade do sistema.

# Desafios da Tolerância a Falhas

# Desafios da Tolerância a Falhas

## Desafios:

Para tornar populares soluções que nos garantam a confiança que depositamos em sistemas computacionais, vários desafios devem ser vencidos:

- Como evitar, detectar e contornar *bugs* no projeto de *hardware e software*?
- Como gerenciar a altíssima complexidade dos sistemas atuais de computação construídos com dezenas de chips de milhões de transistores e com *software* de centenas de milhares de linhas de código?
- Como explorar paralelismo para aumentar o desempenho sem comprometer a qualidade dos resultados, mesmo no caso de falha de um ou mais componentes do sistema?



# Desafios da Tolerância a Falhas

## Desafios:

- Como aproveitar novas tecnologias, mais rápidas, mais baratas e mais eficientes, mas ainda não totalmente provadas e testadas, sem saber qual é o seu comportamento em situações inesperadas sob falha ou sobrecarga?
- Como aproveitar, para aplicações críticas e para operação em tempo real, o modelo de sistemas distribuídos construídos sobre plataformas não confiáveis de redes, contornando os problemas de perdas de mensagens, particionamento de rede e intrusão de *hackers*?

# Desafios da Tolerância a Falhas

## Desafios:

- Como desenvolver computadores móveis e sistemas embarcados, garantindo confiabilidade e segurança nesses dispositivos, e assegurando simultaneamente baixo consumo de potência, sem recorrer a técnicas usuais de replicação de componentes que aumentam peso e volume?
- Finalmente, como conciliar alta confiabilidade e alta disponibilidade com as crescentes demandas por alto desempenho?

# Sistemas de Tempo Real Críticos e Não Críticos

## EXPLICANDO

Para alguns sistemas, a tolerância a falhas é imprescindível! Muitas vezes o sistema não pode sequer existir se não houver garantias matemáticas de sua capacidade de tolerar falhas.

# Sistemas de Tempo Real Críticos e Não Críticos

## EXPLICANDO

Para alguns sistemas, a tolerância a falhas é imprescindível! Muitas vezes o sistema não pode sequer existir se não houver garantias matemáticas de sua capacidade de tolerar falhas.

Ex: Sistemas que controlam aeronaves, trens, metrô. Sistemas bancários, sistema de compra/venda de ações da bolsa, etc

# Sistemas de Tempo Real Críticos e Não Críticos

## EXPLICANDO

Para alguns sistemas, a tolerância a falhas é imprescindível! Muitas vezes o sistema não pode sequer existir se não houver garantias matemáticas de sua capacidade de tolerar falhas.

Ex: Sistemas que controlam aeronaves, trens, metrô. Sistemas bancários, sistema de compra/venda de ações da bolsa, etc

Esses sistemas são Sistemas de Tempo Real Críticos e Não Críticos, e muitas vezes até sistemas pertencentes a ambas categorias, como os sistemas de aeronaves, sondas espaciais e controles industriais de tempo real, redes de sensores sem fio para monitoramento de florestas, campos de petróleo subaquático, entre outras aplicações de extrema necessidade de confiabilidade.

# Falhas são inevitáveis, mas o colapso do sistema não!

## Voltando aos Desafios...

**Falhas são inevitáveis**, mas as consequências das falhas, ou seja O COLAPSO DO SISTEMA, A INTERRUPÇÃO NO FORNECIMENTO DO SERVIÇO E A PERDA DE DADOS, **podem ser evitados** pelo uso adequado de técnicas viáveis e de fácil compreensão.

# Falhas são inevitáveis, mas o colapso do sistema não!

## Voltando aos Desafios...

**Falhas são inevitáveis**, mas as consequências das falhas, ou seja O COLAPSO DO SISTEMA, A INTERRUPÇÃO NO FORNECIMENTO DO SERVIÇO E A PERDA DE DADOS, **podem ser evitados** pelo uso adequado de técnicas viáveis e de fácil compreensão.

O conhecimento dessas técnicas habilita o administrador de sistemas a implementar as soluções mais simples, ou exigir dos fornecedores e desenvolvedores de sistemas soluções que as incorporem. [TANENBAUM e STEEN 2007]

# Falhas são inevitáveis, mas o colapso do sistema não!

Com a expansão dos Sistemas Distribuídos e a espantosa popularização das redes de computadores, fornecendo os mais variados serviços, aumentou a dependência tecnológica de uma grande parcela da população na tecnologia.



# Falhas são inevitáveis, mas o colapso do sistema não!

Com a expansão dos Sistemas Distribuídos e a espantosa popularização das redes de computadores, fornecendo os mais variados serviços, aumentou a dependência tecnológica de uma grande parcela da população na tecnologia.

Falhas nesses serviços podem ser catastróficas para a segurança da população ou para a imagem e reputação das empresas. Para não ser o elo fraco de uma corrente, o mais simples dos computadores conectado a uma rede deve apresentar um mínimo de confiabilidade.

## Falhas são inevitáveis, mas o colapso do sistema não!

Com a expansão dos Sistemas Distribuídos e a espantosa popularização das redes de computadores, fornecendo os mais variados serviços, aumentou a dependência tecnológica de uma grande parcela da população na tecnologia.

Falhas nesses serviços podem ser catastróficas para a segurança da população ou para a imagem e reputação das empresas. Para não ser o elo fraco de uma corrente, o mais simples dos computadores conectado a uma rede deve apresentar um mínimo de confiabilidade.

**COMO DIZ O DITADO, UMA CORRENTE É TÃO FORTE QUANTO SEU ELO MAIS FRACO!**

Sistemas tradicionais		Redes cliente-servidor (não tolerantes a falhas)
Não tolerantes a falhas	Tolerantes a falhas	
MTTF: 6 a 12 semanas Indisponibilidade após defeito: 1 a 4 horas	MTTF: 21 anos (Tandem)	Disponibilidade média: 98%
<b>Defeitos:</b> hardware 50% software 25% comunicação/ambiente 15% operações 10%	<b>Defeitos:</b> software 65% operações 10% hardware 8% ambiente 7%	<b>Defeitos:</b> projeto 60% operações 24% físicos 16%

Causas usuais de defeitos em sistemas de computação  
(MTTF - *Mean time to failure*)

# Técnicas para Alcançar DEPENDABILITY:

## Prevenção de Falhas:

Impede a ocorrência ou introdução de falhas. Envolve a seleção de metodologias de projeto e de tecnologias adequadas para os seus componentes.

# Técnicas para Alcançar DEPENDABILITY:

## Prevenção de Falhas:

Impede a ocorrência ou introdução de falhas. Envolve a seleção de metodologias de projeto e de tecnologias adequadas para os seus componentes.

## Validação:

Remoção de falhas, verificação da presença de falhas.

# Técnicas para Alcançar DEPENDABILITY:

## Tolerância a Falhas:

Fornece o serviço esperado mesmo na presença de falhas. Técnicas comuns: mascaramento de falhas, detecção de falhas, localização, confinamento, recuperação, reconfiguração, tratamento.

# Técnicas para Alcançar DEPENDABILITY:

## Tolerância a Falhas:

Fornece o serviço esperado mesmo na presença de falhas. Técnicas comuns: mascaramento de falhas, detecção de falhas, localização, confinamento, recuperação, reconfiguração, tratamento.

## Previsão de Falhas:

Estimativas sobre presença de falhas e estimativas sobre consequências de falhas.

# As 4 fases da Tolerância a Falhas [Lee et al. 1990]

- 1 **Detecção de Erros:** Duplicação e comparação; testes de limites de tempo; testes reversos; codificação: paridade, códigos de detecção de erros, teste de razoabilidade, de limites e de compatibilidades; testes estruturais e de consistência; diagnóstico



# As 4 fases da Tolerância a Falhas [Lee et al. 1990]

- 1 Detecção de Erros:** Duplicação e comparação; testes de limites de tempo; testes reversos; codificação: paridade, códigos de detecção de erros, teste de razoabilidade, de limites e de compatibilidades; testes estruturais e de consistência; diagnóstico
- 2 Confinamento e Avaliação:** ações atômicas; operações primitivas; auto encapsuladas; isolamento de processos; regras do tipo tudo que não é permitido é proibido; hierarquia de processos; controle de recursos

# As 4 fases da Tolerância a Falhas [Lee et al. 1990]

- 1 Detecção de Erros:** Duplicação e comparação; testes de limites de tempo; testes reversos; codificação: paridade, códigos de detecção de erros, teste de razoabilidade, de limites e de compatibilidades; testes estruturais e de consistência; diagnóstico
- 2 Confinamento e Avaliação:** ações atômicas; operações primitivas; auto encapsuladas; isolamento de processos; regras do tipo tudo que não é permitido é proibido; hierarquia de processos; controle de recursos
- 3 Recuperação de Erros:** Técnicas de recuperação por retorno (Backward Error Recovery); técnicas de recuperação por avanço (Forward Error Recovery)

# As 4 fases da Tolerância a Falhas [Lee et al. 1990]

- 1 Detecção de Erros:** Duplicação e comparação; testes de limites de tempo; testes reversos; codificação: paridade, códigos de detecção de erros, teste de razoabilidade, de limites e de compatibilidades; testes estruturais e de consistência; diagnóstico
- 2 Confinamento e Avaliação:** ações atômicas; operações primitivas; auto encapsuladas; isolamento de processos; regras do tipo tudo que não é permitido é proibido; hierarquia de processos; controle de recursos
- 3 Recuperação de Erros:** Técnicas de recuperação por retorno (Backward Error Recovery); técnicas de recuperação por avanço (Forward Error Recovery)
- 4 Tratamento da Falha:** Diagnóstico e Reparo

# 1ª Fase → Detecção de Erros

Uma falha inicialmente se manifesta como um erro, para então ser detectada. Antes disso, ela está latente e não pode ser detectada. Eventualmente, a falha pode permanecer no sistema durante toda a sua vida útil, sem nunca se manifestar.

## 1ª Fase → Detecção de Erros

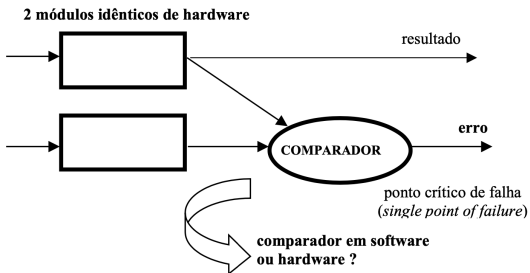
Uma falha inicialmente se manifesta como um erro, para então ser detectada. Antes disso, ela está latente e não pode ser detectada. Eventualmente, a falha pode permanecer no sistema durante toda a sua vida útil, sem nunca se manifestar.

Exemplo de detecção de erros: dois *hardware* idênticos executam a mesma computação e comparam suas saídas.

## 1ª Fase → Detecção de Erros

Uma falha inicialmente se manifesta como um erro, para então ser detectada. Antes disso, ela está latente e não pode ser detectada. Eventualmente, a falha pode permanecer no sistema durante toda a sua vida útil, sem nunca se manifestar.

Exemplo de detecção de erros: dois *hardware* idênticos executam a mesma computação e comparam suas saídas.



## 2ª Fase → Confinamento e Avaliação

Devido a latência da falha, após a ocorrência da falha até o erro ser detectado, pode ter ocorrido espalhamento de dados inválidos.

## 2ª Fase → Confinamento e Avaliação

Devido a latência da falha, após a ocorrência da falha até o erro ser detectado, pode ter ocorrido espalhamento de dados inválidos.

O confinamento estabelece limites para a propagação do dano, mas depende de decisões de projeto



## 2ª Fase → Confinamento e Avaliação

Devido a latência da falha, após a ocorrência da falha até o erro ser detectado, pode ter ocorrido espalhamento de dados inválidos.

O confinamento estabelece limites para a propagação do dano, mas depende de decisões de projeto

Os sistemas, por sua natureza, não provêm confinamento. Durante o projeto devem ser previstas e implementadas restrições ao fluxo de informações para evitar fluxos acidentais e estabelecer interfaces de verificação para detecção de erros.

## 3ª Fase → Recuperação de Erros

A recuperação de erros ocorre após a detecção e envolve a troca do estado atual incorreto para um estado livre de falhas.

### 3ª Fase → Recuperação de Erros

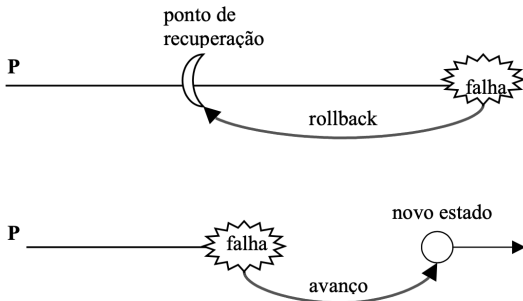
A recuperação de erros ocorre após a detecção e envolve a troca do estado atual incorreto para um estado livre de falhas.

A recuperação pode ser de duas formas: técnicas de recuperação por retorno (Backward Error Recovery) e técnicas de recuperação por avanço (Forward Error Recovery).

### 3ª Fase → Recuperação de Erros

A recuperação de erros ocorre após a detecção e envolve a troca do estado atual incorreto para um estado livre de falhas.

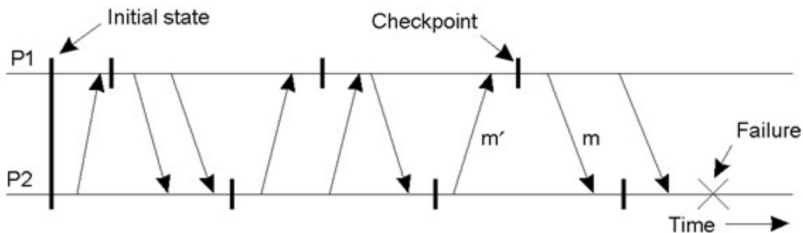
A recuperação pode ser de duas formas: técnicas de recuperação por retorno (Backward Error Recovery) e técnicas de recuperação por avanço (Forward Error Recovery).



### 3ª Fase → Recuperação de Erros

#### *Backward Error Recovery:*

Através da criação de pontos de *checkpoint*, pode-se retornar ao último ponto salvo. A cada *checkpoint* todo o estado do sistema deverá ser salvo para que ele possa ser totalmente recuperado.



## 3ª Fase → Recuperação de Erros

A recuperação é simples para um único processo, realizada através de *Rollback*, mas é complexa num Sistema Distribuído, com  $n$  processos executando, dependendo e cooperando entre si.

### 3ª Fase → Recuperação de Erros

A recuperação é simples para um único processo, realizada através de *Rollback*, mas é complexa num Sistema Distribuído, com  $n$  processos executando, dependendo e cooperando entre si.

Portanto, o *Backward Error Recovery*, em SDs, pode provocar EFEITO DOMINÓ. Desfazer a computação de um processo, muitas vezes, provoca a necessidade de desfazer a computação de diversos outros processos associados. Gerando um alto custo ao sistema.

### 3ª Fase → Recuperação de Erros

A recuperação é simples para um único processo, realizada através de *Rollback*, mas é complexa num Sistema Distribuído, com  $n$  processos executando, dependendo e cooperando entre si.

Portanto, o *Backward Error Recovery*, em SDs, pode provocar EFEITO DOMINÓ. Desfazer a computação de um processo, muitas vezes, provoca a necessidade de desfazer a computação de diversos outros processos associados. Gerando um alto custo ao sistema.



### 3ª Fase → Recuperação de Erros

A recuperação é simples para um único processo, realizada através de *Rollback*, mas é complexa num Sistema Distribuído, com  $n$  processos executando, dependendo e cooperando entre si.

Portanto, o *Backward Error Recovery*, em SDs, pode provocar EFEITO DOMINÓ. Desfazer a computação de um processo, muitas vezes, provoca a necessidade de desfazer a computação de diversos outros processos associados. Gerando um alto custo ao sistema.

Uma solução para este problema é impor restrições à comunicação entre os processos.

### 3ª Fase → Recuperação de Erros

A recuperação é simples para um único processo, realizada através de *Rollback*, mas é complexa num Sistema Distribuído, com  $n$  processos executando, dependendo e cooperando entre si.

Portanto, o *Backward Error Recovery*, em SDs, pode provocar EFEITO DOMINÓ. Desfazer a computação de um processo, muitas vezes, provoca a necessidade de desfazer a computação de diversos outros processos associados. Gerando um alto custo ao sistema.

Uma solução para este problema é impor restrições à comunicação entre os processos.

Técnicas de recuperação por retorno não são adequadas a Sistemas de Tempo Real, devido à necessidade de cumprir *Deadlines*. Nesses sistemas, recomenda-se utilizar *Forward Error Recovery*.

## 3ª Fase → Recuperação de Erros

<b>Técnica</b>	<b>Definição</b>	<b>Características</b>	<b>Implementação</b>
<i>forward error recovery</i>	condução a novo estado ainda não ocorrido desde a última manifestação de erro	eficiente, mas danos devem ser previstos acuradamente	específica a cada sistema
<i>backward error recovery</i>	condução a estado anterior	alto custo, mas de aplicação genérica	pontos de recuperação ( <i>checkpoints</i> ), pistas de auditoria, ...

## 4ª Fase → Tratamento

A última fase, tratamento de falhas, consiste em:

## 4ª Fase → Tratamento

A última fase, tratamento de falhas, consiste em:

Localizar a origem do erro (falha),

## 4ª Fase → Tratamento

A última fase, tratamento de falhas, consiste em:

Localizar a origem do erro (falha), localizar a falha de forma precisa,

## 4ª Fase → Tratamento

A última fase, tratamento de falhas, consiste em:

Localizar a origem do erro (falha), localizar a falha de forma precisa, reparar a falha

## 4ª Fase → Tratamento

A última fase, tratamento de falhas, consiste em:

Localizar a origem do erro (falha), localizar a falha de forma precisa, reparar a falha e recuperar o restante do sistema.



## 4ª Fase → Tratamento

A última fase, tratamento de falhas, consiste em:

Localizar a origem do erro (falha), localizar a falha de forma precisa, reparar a falha e recuperar o restante do sistema.

Nessa fase, geralmente é considerada a hipótese de uma única falha ocorrendo a cada vez.

## 4ª Fase → Tratamento

A última fase, tratamento de falhas, consiste em:

Localizar a origem do erro (falha), localizar a falha de forma precisa, reparar a falha e recuperar o restante do sistema.

Nessa fase, geralmente é considerada a hipótese de uma única falha ocorrendo a cada vez.

A localização da falha é realizada através do diagnóstico.

## 4ª Fase → Tratamento

A última fase, tratamento de falhas, consiste em:

Localizar a origem do erro (falha), localizar a falha de forma precisa, reparar a falha e recuperar o restante do sistema.

Nessa fase, geralmente é considerada a hipótese de uma única falha ocorrendo a cada vez.

A localização da falha é realizada através do diagnóstico.

O diagnóstico é um teste com comparação dos resultados gerados com os resultados previstos.

## 4ª Fase → Tratamento

Após a localização, a falha é reparada através da remoção do componente danificado.

## 4ª Fase → Tratamento

Após a localização, a falha é reparada através da remoção do componente danificado.

O reparo pode ser manual ou automático.

## 4ª Fase → Tratamento

Após a localização, a falha é reparada através da remoção do componente danificado.

O reparo pode ser manual ou automático. O reparo automático pode envolver:

## 4ª Fase → Tratamento

Após a localização, a falha é reparada através da remoção do componente danificado.

O reparo pode ser manual ou automático. O reparo automático pode envolver:

### Degradação Gradual:

Uma reconfiguração para operação com menor número de componentes.

## 4ª Fase → Tratamento

Após a localização, a falha é reparada através da remoção do componente danificado.

O reparo pode ser manual ou automático. O reparo automático pode envolver:

### Degradação Gradual:

Uma reconfiguração para operação com menor número de componentes.

### Substituição Automática:

Usada em sistemas com longo período de missão sem possibilidade de reparo manual, como por exemplo em sondas espaciais e satélites.



# Mascaramento de Falhas

O mascaramento de falhas garante resposta correta mesmo na presença de falhas.

# Mascaramento de Falhas

O mascaramento de falhas garante resposta correta mesmo na presença de falhas.

A falha não se manifesta como erro, o sistema não entra em estado errôneo e, portanto, erros não precisam ser detectados, confinados e recuperados.

# Mascaramento de Falhas

O mascaramento de falhas garante resposta correta mesmo na presença de falhas.

A falha não se manifesta como erro, o sistema não entra em estado errôneo e, portanto, erros não precisam ser detectados, confinados e recuperados.

Entretanto, em caso de falhas permanentes, a localização e o reparo da falha ainda são necessários.

# Mascaramento de Falhas

<b>Mecanismo</b>	<b>Aplicado a:</b>
replicação de componentes	qualquer componente de hardware
ECC (código de correção de erros)	informação transmitida ou armazenada
diversidade, programação n-versões	especificação, projetos, programas
blocos de recuperação	software

# Redundância

# Redundância

Redundância é a palavra mágica em tolerância a falhas.

# Redundância

Redundância é a palavra mágica em tolerância a falhas.

Redundância para aumento de confiabilidade é quase tão antiga como a história dos computadores.

# Redundância

Redundância é a palavra mágica em tolerância a falhas.

Redundância para aumento de confiabilidade é quase tão antiga como a história dos computadores.

Redundância está tão intimamente relacionada a tolerância a falhas que, na indústria nacional, o termo usado para designar um sistema tolerante a falhas é sistema redundante.



# Redundância

A aplicação de redundância para implementar técnicas de tolerância a falhas pode aparecer de várias formas:

# Redundância

A aplicação de redundância para implementar técnicas de tolerância a falhas pode aparecer de várias formas:

- Redundância de Informação

# Redundância

A aplicação de redundância para implementar técnicas de tolerância a falhas pode aparecer de várias formas:

- Redundância de Informação
- Redundância Temporal

# Redundância

A aplicação de redundância para implementar técnicas de tolerância a falhas pode aparecer de várias formas:

- Redundância de Informação
- Redundância Temporal
- Redundância de Hardware

# Redundância

A aplicação de redundância para implementar técnicas de tolerância a falhas pode aparecer de várias formas:

- Redundância de Informação
- Redundância Temporal
- Redundância de Hardware
- Redundância de Software

# Redundância

Apesar de ser a solução “mágica” para a TOLERÂNCIA A FALHAS os sistemas redundantes são muitas vezes custosos, lentos e grandes demais, por isso, nem sempre são soluções viáveis e devem ser usados com sabedoria.

# Redundância

Apesar de ser a solução “mágica” para a TOLERÂNCIA A FALHAS os sistemas redundantes são muitas vezes custosos, lentos e grandes demais, por isso, nem sempre são soluções viáveis e devem ser usados com sabedoria.

Redundância tanto pode servir para detecção de falhas como para mascaramento de falhas. O grau de redundância em cada caso é diferente.

# Redundância

Para mascarar falhas são necessários mais componentes do que para detectar falhas.

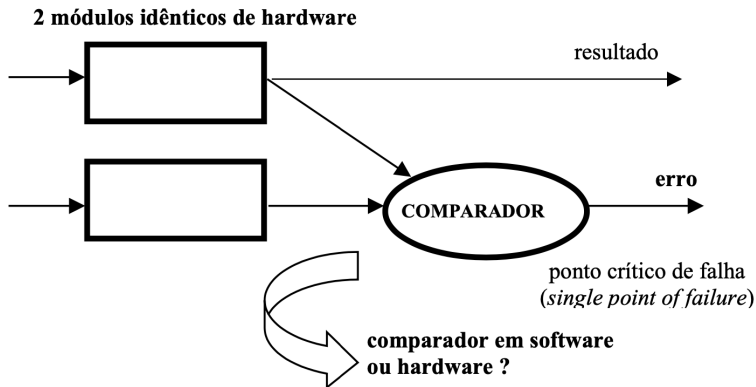


# Redundância

Para mascarar falhas são necessários mais componentes do que para detectar falhas.

Por exemplo, para detectar falhas em um microprocessador, muitas vezes é usado outro microprocessador idêntico, sincronizado ao primeiro, além de um comparador de sinais na saída de ambos (duplicação e comparação).

# Redundância



# Redundância

Qualquer diferença na comparação indica que o par de microprocessadores está em desacordo, e que portanto um dos dois está danificado (ou sofreu uma falha temporária).

# Redundância

Qualquer diferença na comparação indica que o par de microprocessadores está em desacordo, e que portanto um dos dois está danificado (ou sofreu uma falha temporária).

Entretanto, esta falha não pode ser mascarada. O resultado da comparação não indica quais as saídas são as corretas.

# Redundância

Qualquer diferença na comparação indica que o par de microprocessadores está em desacordo, e que portanto um dos dois está danificado (ou sofreu uma falha temporária).

Entretanto, esta falha não pode ser mascarada. O resultado da comparação não indica quais as saídas são as corretas.

Para mascarar, precisa de um terceiro microprocessador que indique, através de melhor de três, qual a saída correta.

# Redundância de Informação

# Redundância de Informação

Na redundância de informação, bits ou sinais extras são armazenados ou transmitidos junto ao dado, sem que contenham qualquer informação útil. Esses bits ou sinais extras servem apenas para detecção de erros ou mascaramento de falhas.

# Redundância de Informação

Na redundância de informação, bits ou sinais extras são armazenados ou transmitidos junto ao dado, sem que contenham qualquer informação útil. Esses bits ou sinais extras servem apenas para detecção de erros ou mascaramento de falhas.

Exemplos clássicos são códigos de paridade, onde para cada  $n$  bits são armazenados  $n + 1$  bits. O bit extra indica apenas se o número de bits com valor 1 nos restantes  $n$  bits é par (ou ímpar, dependendo do tipo de paridade usada).



# Redundância de Informação

Na redundância de informação, bits ou sinais extras são armazenados ou transmitidos junto ao dado, sem que contenham qualquer informação útil. Esses bits ou sinais extras servem apenas para detecção de erros ou mascaramento de falhas.

Exemplos clássicos são códigos de paridade, onde para cada  $n$  bits são armazenados  $n + 1$  bits. O bit extra indica apenas se o número de bits com valor 1 nos restantes  $n$  bits é par (ou ímpar, dependendo do tipo de paridade usada).

Bits de paridade servem para detecção de falhas simples, ou seja, aquelas falhas que afetam apenas 1 bit. Outros exemplos: *checksums*, códigos de duplicação, códigos cíclicos.

# Redundância de Informação

Mascaramento usando redundância de informação é provida por códigos de correção de erros, como ECC (error correction code).

# Redundância de Informação

Mascaramento usando redundância de informação é provida por códigos de correção de erros, como ECC (error correction code).

Códigos de correção de erros estão sendo exaustivamente usados em memórias e em transferência entre memórias e processadores.

# Redundância de Informação

Mascaramento usando redundância de informação é provida por códigos de correção de erros, como ECC (error correction code).

Códigos de correção de erros estão sendo exaustivamente usados em memórias e em transferência entre memórias e processadores.

Exemplos de ECC são os códigos de Hamming, uma combinação de bits de paridade que permite tanto detecção como correção.

## Redundância de Informação

Mascaramento usando redundância de informação é provida por códigos de correção de erros, como ECC (error correction code).

Códigos de correção de erros estão sendo exaustivamente usados em memórias e em transferência entre memórias e processadores.

Exemplos de ECC são os códigos de Hamming, uma combinação de bits de paridade que permite tanto detecção como correção.

Como a codificação da informação implica no aumento do número de bits, e os bits adicionais não aumentam a capacidade de representação de dados do código, é fácil perceber a razão da codificação também ser considerada uma forma de redundância.

# Redundância Temporal

# Redundância Temporal

A Redundância Temporal repete a computação no tempo.

# Redundância Temporal

A Redundância Temporal repete a computação no tempo.

Evita custo de hardware adicional, aumentando o tempo necessário para realizar uma computação.



# Redundância Temporal

É usada em sistemas onde o tempo não é crítico, ou onde o processador trabalha com ociosidade. Aplicações usuais da redundância temporal:

# Redundância Temporal

É usada em sistemas onde o tempo não é crítico, ou onde o processador trabalha com ociosidade. Aplicações usuais da redundância temporal:

**DETECÇÃO DE FALHAS TRANSITÓRIAS:**

# Redundância Temporal

É usada em sistemas onde o tempo não é crítico, ou onde o processador trabalha com ociosidade. Aplicações usuais da redundância temporal:

**DETECÇÃO DE FALHAS TRANSITÓRIAS:** Repetindo a computação. Resultados diferentes são uma forte indicação de uma falha transitória. Essa estratégia não é adequada para falhas permanentes, porque os resultados repetidos nesse caso serão sempre iguais.

# Redundância Temporal

É usada em sistemas onde o tempo não é crítico, ou onde o processador trabalha com ociosidade. Aplicações usuais da redundância temporal:

**DETECÇÃO DE FALHAS PERMANENTES:**

# Redundância Temporal

É usada em sistemas onde o tempo não é crítico, ou onde o processador trabalha com ociosidade. Aplicações usuais da redundância temporal:

**DETECÇÃO DE FALHAS PERMANENTES:** Repete-se a computação com dados codificados e decodifica-se o resultado antes da comparação com o resultado anterior. Essa estratégia provoca a manifestação de falhas permanentes. Por exemplo, considere um barramento com uma linha grudada em zero.

# Redundância Temporal

É usada em sistemas onde o tempo não é crítico, ou onde o processador trabalha com ociosidade. Aplicações usuais da redundância temporal:

**DETECÇÃO DE FALHAS PERMANENTES:** Repete-se a computação com dados codificados e decodifica-se o resultado antes da comparação com o resultado anterior. Essa estratégia provoca a manifestação de falhas permanentes. Por exemplo, considere um barramento com uma linha grudada em zero.

Realiza-se duas transmissões de dados, uma transferência com dado normal e a segunda com dado invertido. Se estiver com falha, a linha vai transferir sempre zero.

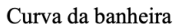
# Redundância de *Hardware*

## Redundância de *Hardware*

Os dispositivos de *Hardware* possuem uma vida útil determinada pela *curva da banheira*, que indica que depois de algum tempo, eles tendem a apresentar defeitos, causando falhas nos Sistemas que os utilizam. Precisando, muitas vezes, de substituição.



Os dispositivos de *Hardware* possuem uma vida útil determinada pela *curva da banheira*, que indica que depois de algum tempo, eles tendem a apresentar defeitos, causando falhas nos Sistemas que os utilizam. Precisando, muitas vezes, de substituição.



## Redundância de *Hardware*

Desta forma, prevendo a falha destes componentes, a Redundância de *Hardware* está baseada na replicação de componentes físicos:

## Redundância de *Hardware*

Desta forma, prevendo a falha destes componentes, a Redundância de *Hardware* está baseada na replicação de componentes físicos:

Redundância de hardware	Técnicas	Vantagens
redundância <b>passiva</b> ou <b>estática</b>	mascamamento de falhas	não requer ação do sistema, não indica falha
redundância <b>ativa</b> ou <b>dinâmica</b>	detecção, localização e recuperação	substituição de módulos, usada em aplicações de longa vida
redundância <b>híbrida</b>	combinação de ativa e passiva	usada para garantir mascaramamento e longa vida; geralmente de alto custo

## Redundância de *Hardware* Passiva ou Estática

Na Redundância de *Hardware* Passiva ou Estática, os elementos redundantes são usados para mascarar falhas.

# Redundância de *Hardware* Passiva ou Estática

Na Redundância de *Hardware* Passiva ou Estática, os elementos redundantes são usados para mascarar falhas.

Todos os elementos executam a mesma tarefa e o resultado é determinado por votação.

## Redundância de *Hardware* Passiva ou Estática

Na Redundância de *Hardware* Passiva ou Estática, os elementos redundantes são usados para mascarar falhas.

Todos os elementos executam a mesma tarefa e o resultado é determinado por votação.

Exemplos são: TMR (Triple Modular Redundancy) e NMR (*N*-Modular Redundancy).

# TMR - Redundância Modular Tripla

## TMR - Redundância Modular Tripla:

É uma das técnicas mais conhecidas de tolerância a falhas. Mascara falhas em um componente de *hardware* triplicando o componente e votando entre as saídas para determinação do resultado. A votação pode ser por maioria (melhor de 3) ou votação por seleção do valor médio.

# TMR - Redundância Modular Tripla

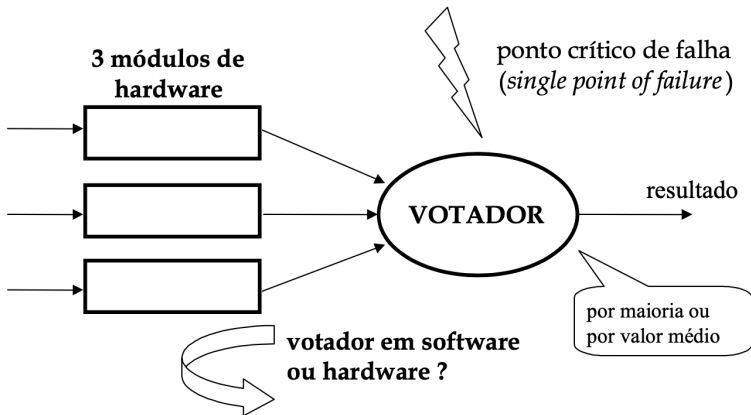
## TMR - Redundância Modular Tripla:

É uma das técnicas mais conhecidas de tolerância a falhas. Mascara falhas em um componente de *hardware* triplicando o componente e votando entre as saídas para determinação do resultado. A votação pode ser por maioria (melhor de 3) ou votação por seleção do valor médio.

O votador realiza uma função simples, cuja correção é fácil de verificar. É interessante observar que o votador não tem a função de determinar qual o módulo de *hardware* discorda da maioria. Se essa função for necessária no sistema, ela deve ser realizada por um detector de falhas.



## TMR - Redundância Modular Tripla



# TMR - Redundância Modular Tripla

## TMR - Redundância Modular Tripla:

Apesar de simples, o votador, por estar em série com os módulos de *hardware* e ter a responsabilidade de fornecer o resultado, é o ponto crítico de falhas no esquema TMR. Se o votador apresentar baixa confiabilidade, todo o esquema vai ser frágil, tão frágil como o votador.

# TMR - Redundância Modular Tripla

## TMR - Redundância Modular Tripla:

Apesar de simples, o votador, por estar em série com os módulos de *hardware* e ter a responsabilidade de fornecer o resultado, é o ponto crítico de falhas no esquema TMR. Se o votador apresentar baixa confiabilidade, todo o esquema vai ser frágil, tão frágil como o votador.

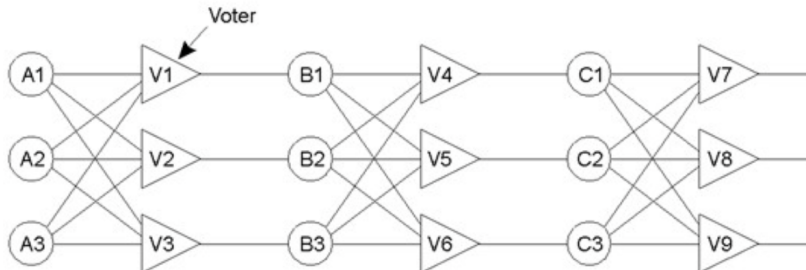
**A CORRENTE É TÃO FRACA QUANTO O SEU ELO MAIS FRACO!**

# TMR - Redundância Modular Tripla

## Soluções para contornar a fragilidade do votador:

- Construir o votador com componentes de alta confiabilidade.
- Triplicar o votador.
- Realizar a votação por software.

## TMR - Redundância Modular Tripla



# TMR - Redundância Modular Tripla

## Confiabilidade do TMR:

TMR apresenta uma confiabilidade maior que um sistema de um único componente até a ocorrência da primeira falha permanente.

# TMR - Redundância Modular Tripla

## Confiabilidade do TMR:

TMR apresenta uma confiabilidade maior que um sistema de um único componente até a ocorrência da primeira falha permanente.

Depois disso, perde a capacidade de mascarar falhas e vai apresentar uma confiabilidade menor que um sistema de um único componente.

# TMR - Redundância Modular Tripla

## Confiabilidade do TMR:

TMR apresenta uma confiabilidade maior que um sistema de um único componente até a ocorrência da primeira falha permanente.

Depois disso, perde a capacidade de mascarar falhas e vai apresentar uma confiabilidade menor que um sistema de um único componente.

Com o tempo, quando aumenta a probabilidade de componentes falharem, o TMR apresenta uma confiabilidade pior do que um sistema não redundante.



# TMR - Redundância Modular Tripla

## Confiabilidade do TMR:

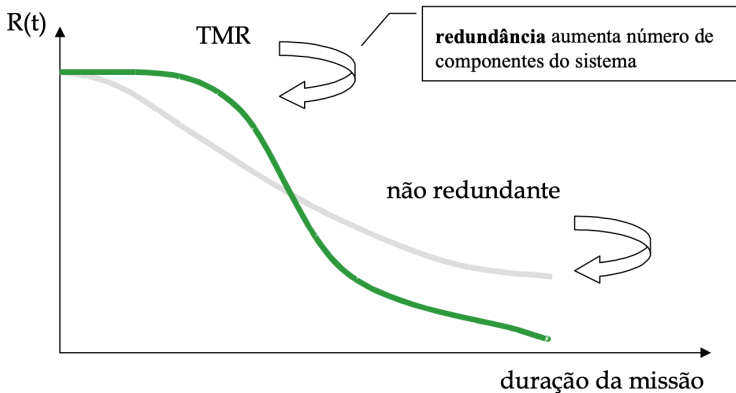
TMR apresenta uma confiabilidade maior que um sistema de um único componente até a ocorrência da primeira falha permanente.

Depois disso, perde a capacidade de mascarar falhas e vai apresentar uma confiabilidade menor que um sistema de um único componente.

Com o tempo, quando aumenta a probabilidade de componentes falharem, o TMR apresenta uma confiabilidade pior do que um sistema não redundante.

O TMR é ideal para períodos não muito longos. Pois suporta uma falha permanente apenas.

# Confiabilidade do TMR



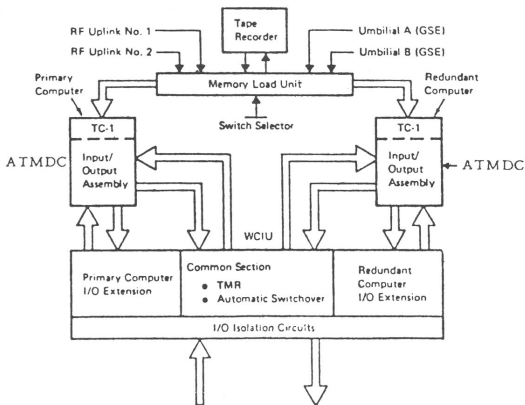
# NMR - Redundância Modular Múltipla

## NMR - Redundância Modular Múltipla:

É a generalização do conceito de TMR, ou seja, o TMR é um caso especial do NMR, onde  $n = 3$ .

O computador de bordo do SPACE SHUTTLE da NASA é um exemplo de NMR, com  $n = 4$  e votação por *software*.

# Exemplo de TMR e NMR - SPACE SHUTTLE

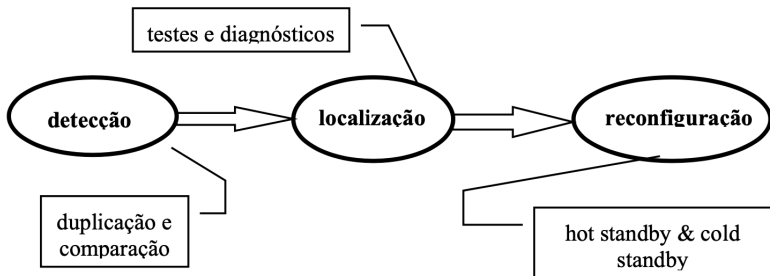


## Redundância de *Hardware* Dinâmica ou Ativa

Na Redundância de *Hardware* Dinâmica ou Ativa, a TOLERÂNCIA A FALHAS é provida através do emprego das técnicas de detecção, localização e recuperação. A redundância empregada, neste caso, não provê mascaramento.

## Redundância de *Hardware* Dinâmica ou Ativa

Na Redundância de *Hardware* Dinâmica ou Ativa, a TOLERÂNCIA A FALHAS é provida através do emprego das técnicas de detecção, localização e recuperação. A redundância empregada, neste caso, não provê mascaramento.



## Redundância de *Hardware* Dinâmica ou Ativa

Um exemplo de implementação de Redundância de *Hardware* Dinâmica são os módulos estepes (StandBy Sparing).

## Redundância de *Hardware* Dinâmica ou Ativa

Um exemplo de implementação de Redundância de *Hardware* Dinâmica são os módulos estepe (StandBy Sparing).

<b>Operação de módulos estepe</b>	<b>Comentários</b>
alimentados ( <i>hot standby</i> )	Módulo estepe conectado eletricamente ao sistema. Minimiza a descontinuidade do processamento durante o chaveamento entre os módulos.
não alimentados ( <i>cold standby</i> )	Módulo estepe só começa a operar quando conectado. Aumenta a vida útil dos módulos estepe.



## Redundância de *Hardware* Dinâmica ou Ativa

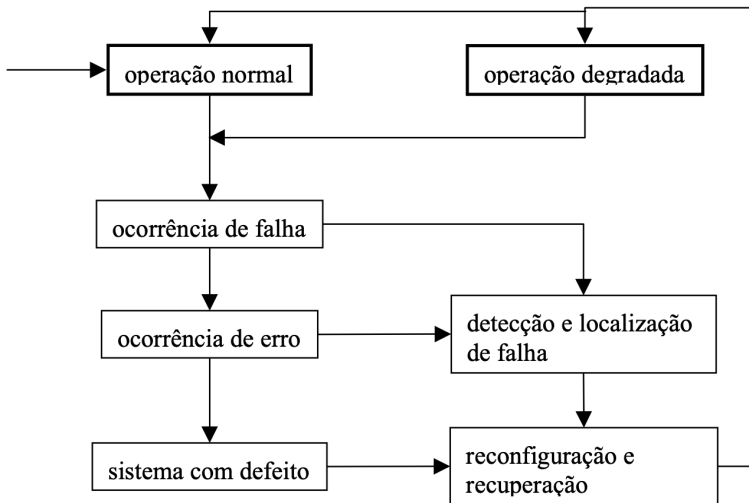
A Redundância de *Hardware* Dinâmica é usada em aplicações que suportam permanecer em um estado errôneo durante um curto período de tempo. Tempo esse necessário para a detecção do erro e recuperação para um estado livre de falhas.

## Redundância de *Hardware* Dinâmica ou Ativa

A Redundância de *Hardware* Dinâmica é usada em aplicações que suportam permanecer em um estado errôneo durante um curto período de tempo. Tempo esse necessário para a detecção do erro e recuperação para um estado livre de falhas.

Geralmente, é preferível defeitos temporários do que suportar o custo de uma grande quantidade de redundância que é necessária para o mascaramento de falhas.

## Redundância de *Hardware* Dinâmica ou Ativa



# Redundância de *Software*

# Redundância de *Software*

A simples replicação de componentes idênticos é uma estratégia de detecção e mascaramento de erros inútil em *software*.

Componentes idênticos de *software* vão apresentar erros idênticos.

## Redundância de *Software*

A simples replicação de componentes idênticos é uma estratégia de detecção e mascaramento de erros inútil em *software*.

Componentes idênticos de *software* vão apresentar erros idênticos.

Assim, não basta copiar um programa e executá-lo em paralelo ou executar o mesmo programa duas vezes em tempos diferentes, ou em máquinas diferentes.

# Redundância de *Software*

A simples replicação de componentes idênticos é uma estratégia de detecção e mascaramento de erros inútil em *software*.

Componentes idênticos de *software* vão apresentar erros idênticos.

Assim, não basta copiar um programa e executá-lo em paralelo ou executar o mesmo programa duas vezes em tempos diferentes, ou em máquinas diferentes.

Erros de programas idênticos vão se apresentar, com grande probabilidade, de forma idêntica para os mesmos dados de entrada.

# Redundância de *Software*

Existem outras formas de aplicar redundância em *software* para detecção e mascaramento, que não envolve cópias idênticas de *software*.



## Redundância de *Software*

Existem outras formas de aplicar redundância em *software* para detecção e mascaramento, que não envolve cópias idênticas de *software*. Exemplos dessas outras formas de redundância são:

# Redundância de *Software*

Existem outras formas de aplicar redundância em *software* para detecção e mascaramento, que não envolve cópias idênticas de *software*. Exemplos dessas outras formas de redundância são:

- diversidade (ou programação n-versões)

# Redundância de *Software*

Existem outras formas de aplicar redundância em *software* para detecção e mascaramento, que não envolve cópias idênticas de *software*. Exemplos dessas outras formas de redundância são:

- diversidade (ou programação n-versões)
- blocos de recuperação

# Redundância de *Software*

Existem outras formas de aplicar redundância em *software* para detecção e mascaramento, que não envolve cópias idênticas de *software*. Exemplos dessas outras formas de redundância são:

- diversidade (ou programação n-versões)
- blocos de recuperação
- verificação de consistência

# Redundância de *Software*

Existem outras formas de aplicar redundância em *software* para detecção e mascaramento, que não envolve cópias idênticas de *software*. Exemplos dessas outras formas de redundância são:

- diversidade (ou programação n-versões)
- blocos de recuperação
- verificação de consistência

É interessante lembrar que se o *software* foi projetado corretamente desde o início, então não são necessárias técnicas de tolerância a falhas para *software*.

# Redundância de *Software*

Existem outras formas de aplicar redundância em *software* para detecção e mascaramento, que não envolve cópias idênticas de *software*. Exemplos dessas outras formas de redundância são:

- diversidade (ou programação n-versões)
- blocos de recuperação
- verificação de consistência

É interessante lembrar que se o *software* foi projetado corretamente desde o início, então não são necessárias técnicas de tolerância a falhas para *software*.

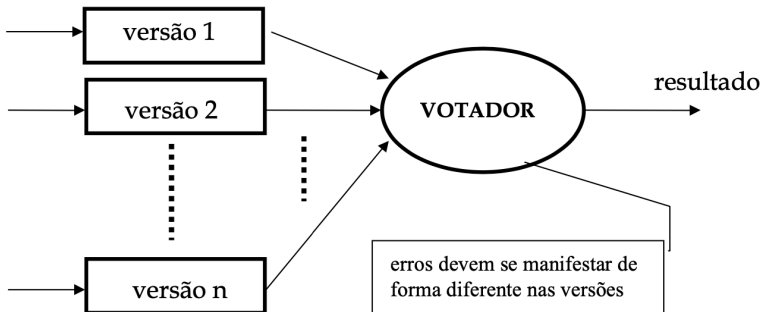
**TEORIA, NA PRÁTICA NÃO CONSEGUIMOS GARANTIR  
PROGRAMAS CORRETOS!**

## DIVERSIDADE - PROGRAMAÇÃO N-VERSÕES:

Diversidade, também chamada programação diversitária, é uma técnica de redundância usada para obter tolerância a falhas em *software*. A partir de um problema a ser solucionado, são implementadas diversas soluções alternativas, sendo a resposta do sistema determinada por votação.

## DIVERSIDADE - PROGRAMAÇÃO N-VERSÕES:

Diversidade, também chamada programação diversitária, é uma técnica de redundância usada para obter tolerância a falhas em *software*. A partir de um problema a ser solucionado, são implementadas diversas soluções alternativas, sendo a resposta do sistema determinada por votação.





## DIVERSIDADE - PROGRAMAÇÃO N-VERSÕES:

Diversidade pode ser utilizada em todas as fases do desenvolvimento do *software*, desde sua especificação até o teste, dependendo do tipo de erro que se deseja detectar.

## DIVERSIDADE - PROGRAMAÇÃO N-VERSÕES:

Diversidade pode ser utilizada em todas as fases do desenvolvimento do *software*, desde sua especificação até o teste, dependendo do tipo de erro que se deseja detectar.

Essa técnica é chamada de projeto DIVERSITÁRIO quando o desenvolvimento do sistema é realizado usando diversidade de metodologias e equipes, e de programação em n-versões, quando se restringe à implementação.

## DIVERSIDADE - PROGRAMAÇÃO N-VERSÕES:

Diversidade pode ser utilizada em todas as fases do desenvolvimento do *software*, desde sua especificação até o teste, dependendo do tipo de erro que se deseja detectar.

Essa técnica é chamada de projeto DIVERSITÁRIO quando o desenvolvimento do sistema é realizado usando diversidade de metodologias e equipes, e de programação em n-versões, quando se restringe à implementação.

Diversidade pode ser também usada como técnica de prevenção de falhas. Nesse último caso, várias alternativas de projeto ou de implementação são desenvolvidas para que, na fase de teste, erros eventuais possam ser localizados e corrigidos de uma forma mais simples e efetiva.

## DIVERSIDADE - PROGRAMAÇÃO N-VERSÕES:

No final da fase de teste, é então escolhida a alternativa em que se detectou a menor ocorrência de erros, e apenas esta alternativa é integrada ao sistema.

## DIVERSIDADE - PROGRAMAÇÃO N-VERSÕES:

No final da fase de teste, é então escolhida a alternativa em que se detectou a menor ocorrência de erros, e apenas esta alternativa é integrada ao sistema.

A facilidade no reconhecimento de erros na fase de teste do sistema, a tolerância tanto de falhas intermitentes quanto de permanentes e a atuação potencial contra erros externos ao sistema como por exemplo: erros do compilador, do sistema operacional e até mesmo falhas de hardware), são vantagens atribuídas a programação DIVERSITÁRIA.

## DIVERSIDADE - PROGRAMAÇÃO N-VERSÕES:

Entretanto, desvantagens da técnica também devem ser citadas, como o aumento dos custos de desenvolvimento e manutenção, a complexidade de sincronização das versões e o problema de determinar a correlação das fontes de erro.

## DIVERSIDADE - PROGRAMAÇÃO N-VERSÕES:

Entretanto, desvantagens da técnica também devem ser citadas, como o aumento dos custos de desenvolvimento e manutenção, a complexidade de sincronização das versões e o problema de determinar a correlação das fontes de erro.

Enquanto pode ser provado formalmente que a redundância de elementos de hardware aumenta a confiabilidade do sistema, tal prova não existe para a diversidade em software.

## DIVERSIDADE - PROGRAMAÇÃO N-VERSÕES:

Vários fatores influenciam a eficácia da programação

DIVERSITÁRIA: as equipes podem trocar algoritmos entre si, os membros das equipes podem, por formação, tender a adotar os mesmos métodos de desenvolvimento, ou as equipes podem buscar suporte nas mesmas fontes. Qualquer uma dessas correlações imprevisíveis se constitui uma fonte potencial de erros.



## DIVERSIDADE - PROGRAMAÇÃO N-VERSÕES:

Vários fatores influenciam a eficácia da programação

DIVERSITÁRIA: as equipes podem trocar algoritmos entre si, os membros das equipes podem, por formação, tender a adotar os mesmos métodos de desenvolvimento, ou as equipes podem buscar suporte nas mesmas fontes. Qualquer uma dessas correlações imprevisíveis se constitui uma fonte potencial de erros.

Um exemplo de diversidade é o sistema de computadores de bordo do SPACE SHUTTLE. Quatro computadores idênticos são usados em NMR. Um quinto computador, diverso em hardware e em software dos outros quatro, pode substituir os demais em caso de colapso do esquema NMR.

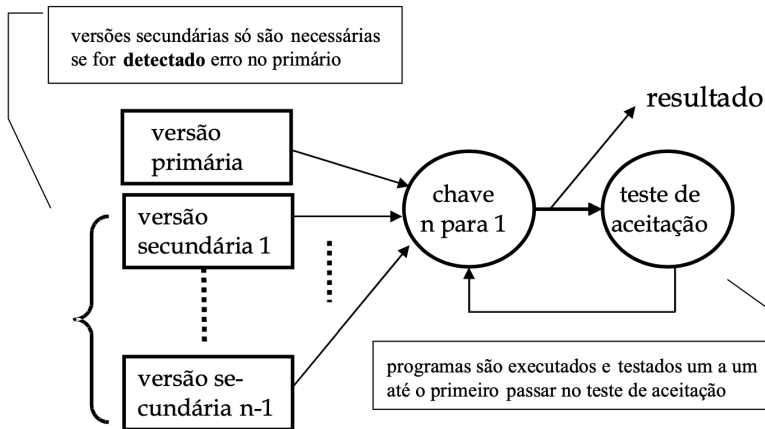
## BLOCOS DE RECUPERAÇÃO:

Semelhante a PROGRAMAÇÃO N-VERSÕES, mas nessa técnica programas secundários só serão necessários na detecção de um erro no programa primário. Essa estratégia envolve um teste de aceitação.

## BLOCOS DE RECUPERAÇÃO:




Semelhante a PROGRAMAÇÃO N-VERSÕES, mas nessa técnica programas secundários só serão necessários na detecção de um erro no programa primário. Essa estratégia envolve um teste de aceitação.

Programas são executados e testados um a um até que o primeiro passe no teste de aceitação. A estratégia de blocos de recuperação tolera  $(n - 1)$  falhas, no caso de falhas independentes nas  $n$ -versões.



# Referências

# Referências

-  COULOURIS, G. et al. *Sistemas Distribuídos: Conceitos e Projetos*. 5. ed. [S.I.]: Bookman, 2013. v. 1.
-  LEE, P. A. et al. *Fault Tolerance: Principles and Practice*. 2nd. ed. Berlin, Heidelberg: Springer-Verlag, 1990. ISBN 0387820779.
-  TANENBAUM, A.; STEEN, M. V. *Sistemas Distribuídos - Princípios e Paradigmas*. 2. ed. [S.I.]: Prentice Hall, 2007. v. 1.