# Sistemas Distribuídos

## Aula 3 – Tolerância a Falhas e Dependability

Prof. Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal da Bahia – Campus Feira de Santana

2026

# Agenda

# Tolerância a Falhas

Sistemas que sobrevivem às suas próprias falhas

# O que é Tolerância a Falhas?

## ▪ Definição

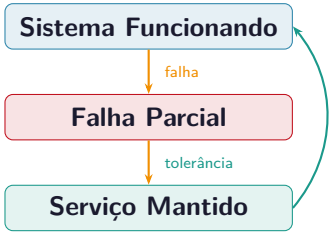
Propriedade que garante a **correta e eficiente operação** de um sistema apesar de falhas em qualquer de seus componentes. Um SD tolerante a falhas garante a **entrega de mensagens** e o funcionamento das aplicações mesmo em presença de falhas inevitáveis em redes heterogêneas. [1]

## ▪ Princípio fundamental

O sistema deve continuar produzindo as **saídas para as quais foi construído** mesmo que alguns componentes não estejam funcionando corretamente. E mais: o **desempenho com componentes falhos deve permanecer em níveis aceitáveis**. [2]

## ▪ Por que é essencial?

A vida *online* não admite mais que sistemas caiam. Perder dinheiro em bancos virtuais ou em arquivos na nuvem é inaceitável. Em sistemas críticos, a falha pode custar vidas.



# Dependability

Confiança no Funcionamento do Sistema

# O que é Dependability?

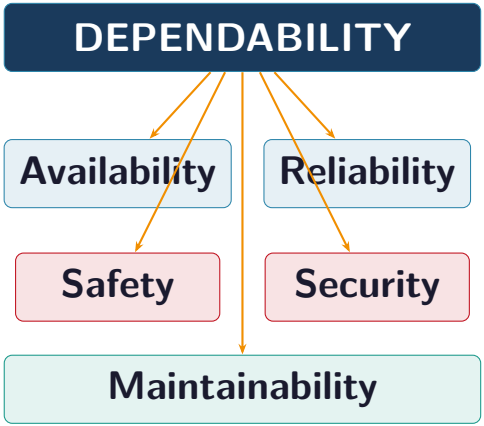
## Definição

Um componente X **depende** de Y se a corretude do comportamento de X depende da corretude de Y. Um componente é **dependente** na medida em que outros componentes podem depender dele.

A *Dependability* é a propriedade que engloba todos os atributos de confiança que um SD deve possuir.

## Os 5 atributos principais

| | |
|---|---|
| **Availability** | Disponibilidade de uso |
| **Reliability** | Serviço correto contínuo |
| **Safety** | Sem consequências catastróficas |
| **Security** | Confidencialidade e integridade |
| **Maintainability** | Facilidade de manutenção |




+ Performability, Testability

# Atributos de Dependability — Parte 1

## ▪ Reliability — Confiabilidade $R(t)$

Capacidade de atender à especificação, sob condições definidas, durante um **período de funcionamento**. É medida de **probabilidade** e não pode ser confundida com disponibilidade.

Mais usada em **sistemas críticos**, onde curtos períodos de operação incorreta são inaceitáveis ou onde o reparo é impossível (sondas espaciais, cirurgias robóticas).

## ▪ Availability — Disponibilidade

Probabilidade do sistema estar **operacional num instante de tempo determinado**. Atributo mais usado em sistemas de missão crítica on-line: bancos de dados, servidores web, servidores de rede. [1]

A disponibilidade está muito relacionada ao **tempo de reparo** diminuir esse tempo aumenta a disponibilidade.

| SLA | Downtime/ano |
|---|---|
| 99% | 87,6 h |
| 99,9% | 8,76 h |
| 99,99% | 52 min |
| 99,999% | 5,3 min |

# Alta Confiabilidade X Baixa Disponibilidade

**Atenção**

Um sistema pode ser de **alta confiabilidade** e de **baixa disponibilidade**. Ex: um avião que precisa de longos reparos entre voos — quando voa, nunca falha, mas fica muito tempo indisponível enquanto está consertando.

# Atributos de Dependability — Parte 2

## ▪ Safety — Segurança Física

Probabilidade do sistema estar operacional **ou** descontinuar suas funções **sem causar dano** a outros sistemas ou pessoas. O sistema deve ser *fail-safe*: ou a saída é correta, ou o sistema entra em um **estado seguro**.
Ex: trem que para automaticamente ao detectar falha.

## ▪ Security — Segurança Lógica

Confidencialidade, integridade e autenticidade das informações. Protege contra interceptação, acesso não autorizado e ataques.

## ▪ Maintainability

Probabilidade de restaurar um sistema com defeito a estado operacional em um período determinado. **Maior testabilidade** $\Rightarrow$ melhor mantenabilidade $\Rightarrow$ menor downtime.

## ▪ Performability

Capacidade do sistema de continuar operando (com desempenho degradado) mesmo com falhas em componentes.

# Fault, Error e Failure

Os 3 estágios de uma falha

# Os 3 Estágios — Fault, Error e Failure

## ▪ Fault (Falha latente)

Incorreção intrínseca de um componente (bug de SW, defeito de HW) que pode ou não se manifestar. Uma *fault* pode ficar **dormente** se o código defeituoso nunca for executado.
*"Fault Tolerance"* = tolerância a falhas latentes.

## ▪ Error (Erro)

Quando a *fault* se manifesta e produz saída fora da especificação. **Mesmo com a fault ativada**, se a saída coincide com o esperado (por acaso), não há error.
Exemplo clássico: algumas máquinas apresentam a mesma falha, enquanto outras não.

## ▪ Failure (Defeito)

O error se propaga até a saída do sistema, causando **colapso observável**. Um erro multiplicado por zero não vira failure é neutralizado.

# Classificação das Falhas

Tipos, causas e características

# Classificação das Falhas

## Quanto à causa

- **Físicas** — componentes HW defeituosos, fadiga
- **De projeto** — erros de especificação ou implementação
- **De operação** — erros humanos na operação do sistema
- **Maliciosas** — ações propositais (tratadas por *security*)

## Quanto à persistência

- **Permanente** — componente definitivamente defeituoso
- **Transitória** — manifesta-se esporadicamente e desaparece

## Tipos de falha em SDs

- **Omissão** — processo omite tarefa/mensagem que deveria realizar
- **Byzantina** — comportamento arbitrário e imprevisível; o componente pode enviar mensagens incorretas
- **Sincronização** — processos excedem limites de tempo
- **Latência de comunicação** — atrasos na rede afetam SLA
- **Consistência** — cópias de dados divergem entre nós
- **Segurança** — interceptação, acesso não autorizado, DDoS

# Desafios da Tolerância a Falhas

## Desafios técnicos

- Evitar, detectar e contornar *bugs* de HW/SW
- Gerenciar a **altíssima complexidade** dos sistemas modernos (chips com bilhões de transistores, SW com milhões de linhas)
- Explorar **paralelismo** sem comprometer resultados mesmo com falhas
- Usar tecnologias novas e baratas sem saber seu comportamento sob falhas

## Desafios de sistemas distribuídos

- Usar SD sobre plataformas **não confiáveis** (perda de mensagens, particionamento, hackers)
- Garantir confiabilidade em **dispositivos móveis e embarcados** com restrições de peso, volume e energia
- Conciliar **alta disponibilidade** com **alto desempenho**

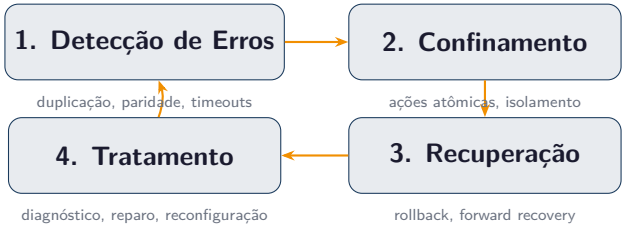
## Lição fundamental

**Falhas são inevitáveis**, mas o colapso do sistema, a interrupção do serviço e a perda de dados **podem ser evitados** com técnicas adequadas.

# As 4 Fases da Tolerância a Falhas

Detecção, Confinamento, Recuperação e Tratamento

# Visão Geral das 4 Fases




## ▪ 1. Detecção de Erros

Duplicação e comparação, testes de limites de tempo, códigos de paridade/ECC, diagnóstico, testes de consistência.
*Ex.: dois HWs idênticos executam a mesma computação e comparam as saídas.*

## ▪ 2. Confinamento e Avaliação

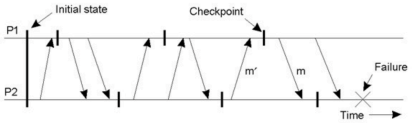
Limita a propagação do dano após detecção. Usa ações atômicas, isolamento de processos, hierarquia de processos e controle de recursos.

## Backward Error Recovery

Retorna ao **último checkpoint** salvo. Todo o estado do sistema é salvo periodicamente; ao detectar um erro, faz *rollback*.

**Problema em SDs:** pode provocar o **Efeito Dominó** — desfazer a computação de um processo força desfazer a de outros processos associados.

**Não adequado para Sistemas de Tempo Real** (deadlines).




## Forward Error Recovery

Em vez de retroceder, **avança para um novo estado correto** a partir do estado errôneo, descartando os dados corrompidos e continuando.

Preferível em **Sistemas de Tempo Real**. Exige que o sistema saiba como reconstruir um estado consistente a partir do estado corrompido.

## 4. Tratamento da Falha

1. Localizar a origem do erro (fault)
2. Reparar a fault (manual ou automático)
3. Recuperar o restante do sistema

Reparo automático: **degradação gradual** ou **substituição automática** (satélites, sondas).

# Redundância

A palavra mágica da Tolerância a Falhas

# Tipos de Redundância

## ▪ Redundância de Informação

Bits/sinais extras **sem informação útil** para detectar ou corrigir erros. Ex: paridade, checksums, CRC, códigos de Hamming (ECC).

**Paridade**: detecta falha simples em 1 bit.
**ECC**: detecta e corrige erros  usado amplamente em memórias de servidores.

## ▪ Redundância Temporal

Repete a computação no tempo, sem hardware adicional.

**Falhas transitórias**: resultados diferentes indicam falha intermitente.
**Falhas permanentes**: transmite o dado normal e, depois, o invertido; linha presa em 0 sempre entrega 0 — detectável.
Não adequada a Sistemas de Tempo Real.

# Tipos de Redundância

## ▪ Redundância de Hardware

Replicação de componentes físicos.

- **Passiva/Estática** (TMR, NMR): todos executam e um votador determina o resultado — **mascaramento**
- **Dinâmica/Ativa** (Hot/Cold Standby): componentes reserva assumem ao detectar falha — **recuperação**

## ▪ Redundância de Software

Cópias idênticas não funcionam — erros iguais para as mesmas entradas. Soluções:

- **N-versões**: múltiplas implementações independentes + votação
- **Blocos de recuperação**: versões alternativas testadas sequencialmente
- **Verificação de consistência**: checar invariantes do sistema

# TMR — Triple Modular Redundancy

## Como funciona

Três módulos idênticos executam a mesma operação. Um **votador** determina o resultado pela maioria. Mascara falhas em **1 componente**.

**Ponto fraco**: o votador em si é crítico — se falhar, todo o esquema colapsa.

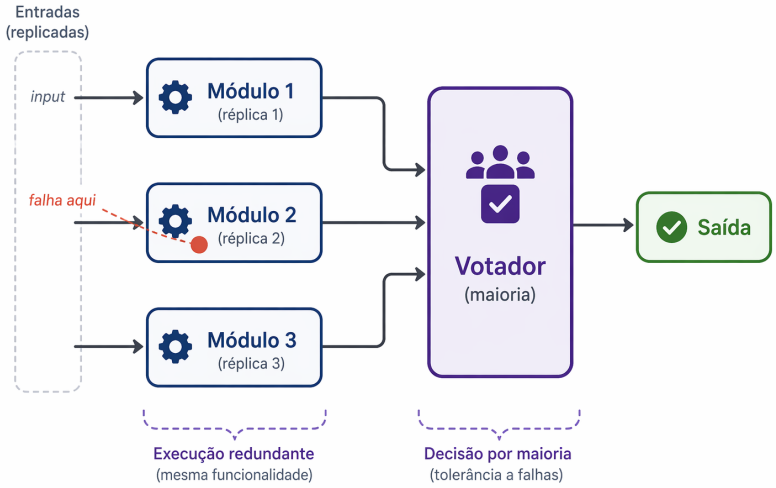
Soluções: componentes de alta confiabilidade, triplicação do votador ou votação por meio de software.

**NMR**: generalização do TMR ($n$ módulos). Ex: Space Shuttle da NASA usou NMR com $n = 4$ + um 5ž computador diverso em SW e HW.

## Limite do TMR

TMR é ideal para **períodos curtos** de missão. Com o tempo, aumenta a probabilidade de 2+ falhas e o TMR passa a ser pior do que o de um sistema simples.

# TMR — Triple Modular Redundancy



# Programação N-Versões e Blocos de Recuperação

## ▪ N-Versões (Diversidade)

A partir da mesma especificação, **equipes independentes** implementam $n$ soluções distintas. O resultado é determinado por **votação**.

**Vantagens**: tolerância a falhas de compilador, SO e HW; funciona contra falhas intermitentes e permanentes.
**Desvantagens**: custo elevado; equipes tendem a adotar métodos similares (correlação de erros); não há prova formal de que aumenta a confiabilidade, como ocorre com HW redundante.

## ▪ Blocos de Recuperação

Programas alternativos são executados **um a um** até que um passe em um **teste de aceitação**.
Programas secundários só são acionados **se o primário falhar**. Tolera $n-1$ falhas em $n$ versões. Mais econômico que N-versões na maioria das execuções (só usa o primário na maior parte do tempo).

## ▪ Realidade

Se o SW fosse projetado corretamente desde o início, técnicas de redundância de SW seriam desnecessárias. Na prática, **não conseguimos garantir programas corretos**.

# Referências I


- TANENBAUM, A. S.; VAN STEEN, M. *Distributed Systems: Principles and Paradigms*. 2. ed. Pearson, 2007.
- COULOURIS, G. et al. *Distributed Systems: Concepts and Design*. 5. ed. Addison-Wesley, 2013.
- LEE, P. A.; ANDERSON, T. *Fault Tolerance: Principles and Practice*. 2. ed. Springer, 1990.