

# Fundamentos de Tolerância a Falhas

---

Motivação

*Taisy Silva Weber*

# Motivação para tolerância a falhas

---

manter o serviço  
desejado mesmo na  
presença de falhas

evitar que o usuário do  
serviço seja o componente  
tolerante a falhas do sistema



- componentes de hardware mais confiáveis
- software e projeto cada vez menos confiáveis
- sistemas cada vez mais complexos

# Desafios atuais

---

- bugs no projeto de hardware e software
  - altíssima complexidade dos sistemas



# Desafios atuais

---

- paralelismo em alta escala

- sistemas distribuídos



- como aproveitar as novas plataformas?
  - para sistemas críticos e de missão crítica
  - para operação em tempo real

# Desafios atuais

---

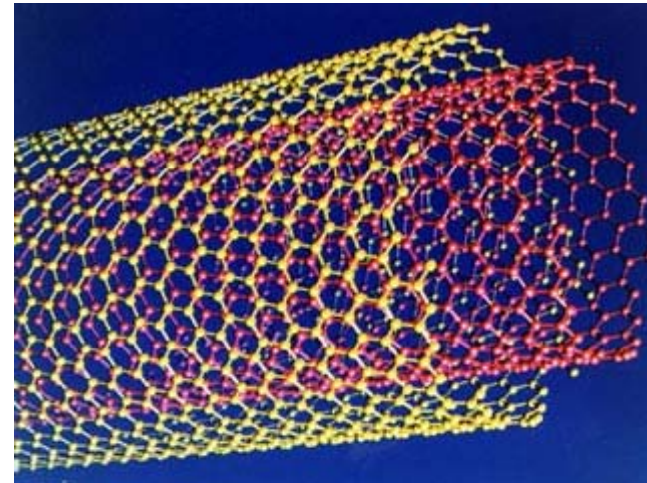
- computadores móveis
  - baixa potência, pequeno volume
  - difícil usar replicação de componentes



# Desafios atuais

---

- novas tecnologias
  - nanotecnologia,
  - biochips,
  - computação quântica



# Desafios atuais

- interação humano-computador
  - complexidade, interfaces amigáveis

The image shows a web form for the CALDO (Ciência sem Fronteiras) registration. The main form has sections for 'Área de Conhecimento' (Knowledge Area), 'Instituição de Graduação' (Graduation Institution), 'Proficiência' (Proficiency), and 'Área Prioritária do CsF' (Priority Area of CsF). A modal window titled 'Adicionar Instituição' (Add Institution) is open, showing a dropdown for 'País da instituição de destino' (Destination institution country) with 'Brasil' selected. A yellow callout box points to this dropdown with the text 'no campo país de destino só aparece Brasil' (only Brazil appears in the destination country field). The modal also includes a text input for 'Instituição de Graduação (por extenso)' (Graduation institution (full name)) and a dropdown for 'Função' (Function).

Área de Conhecimento

Área de Conhecimento: Sistemas de Computação

Buscar área:  Navegar

Instituição de Graduação

Instituição UF

Adicionar instituição

Proficiência

Tipo do teste realizado: TOEFL/IBT Internet Based T

Data da realização do teste: 25/05/2012 (dd/mm/aaaa)

Nota obtida: 90

Área Prioritária do CsF

Área Prioritária do CsF: Computação e Tecnologias da Informação Selecionar

Adicionar Instituição

País da instituição de destino: Brasil

Instituição de Graduação (por extenso):

Função: Instituição de Graduação

Adicionar instituição

no campo país de destino só aparece Brasil

# Desafios atuais

---

- unificação com security
  - *dark side of human nature*
  - unificação de conceitos, medidas, técnicas



figura: <http://www.techwind.com/>

Avizienis, Laprie, Randell, Landwehr. ***Basic Concepts and Taxonomy of Dependable and Secure Computing***. IEEE Trans. on dep. and secure comp. 2004



# Tolerância a falhas

---



- TF é um **meio** para alcançar dependabilidade

desenvolvedores tentam garantir dependabilidade de um sistema usando **técnicas** de Tolerância a Falhas

# Dependabilidade

---



- confiabilidade (*reliability*)
  - disponibilidade (*availability*)
  - segurança *funcional* (*safety*)
  - integridade
- 
- e vários outros atributos dependendo do autor

# Áreas de Aplicação de TF

---

- longa vida
- manutenção adiada
- aplicações críticas
- alta disponibilidade

Johnson, B.W. *Fault Tolerance*, The Electrical Engineering Handbook, Ed. Richard C. Dorf, Boca Raton: CRC Press LLC, 2000

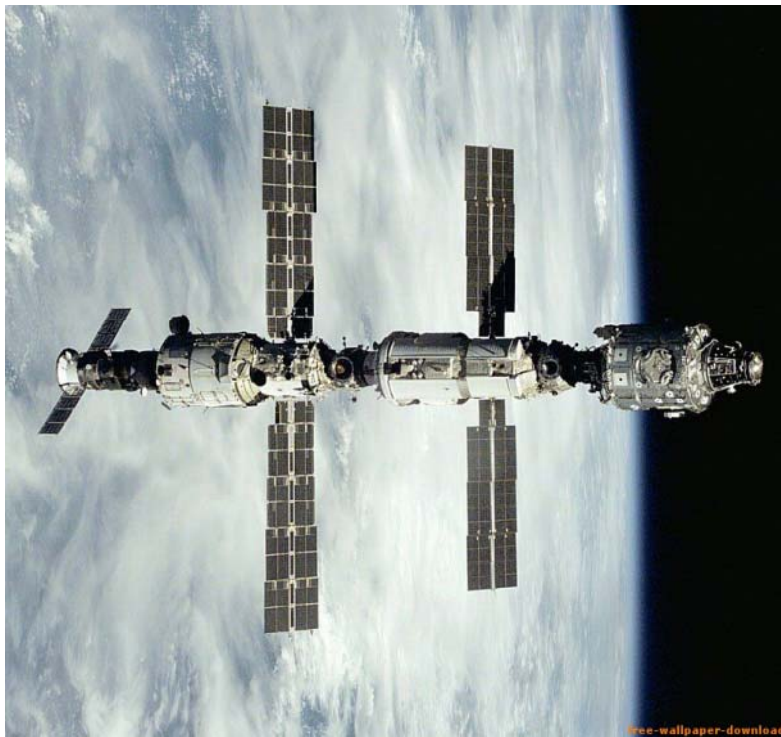
# Áreas de Aplicação

---

- longa vida

satélites e sondas  
espaciais:

probabilidade igual a  
0,95 de estar  
operacional após 10  
anos de missão



# Áreas de Aplicação

---

- manutenção adiada



manutenção é impossível ou extremamente cara:

lugares remotos ou só  
acessíveis  
periodicamente,  
aplicações espaciais

# Área de Aplicação: sistemas críticos

---

- aplicações críticas

segurança humana, proteção de equipamento ou segurança do meio ambiente

- sistemas militares
- controle de tráfego aéreo
- controle industrial
- instrumentação cirúrgica

aparecem também com o nome de **life-critical**



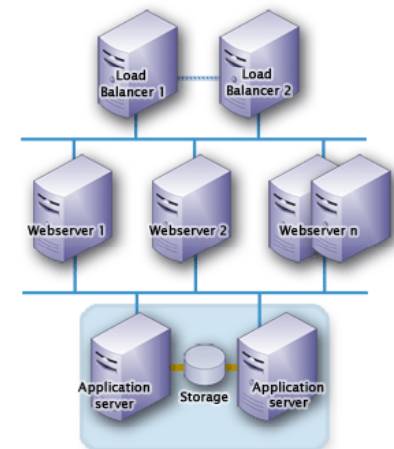
# Área de Aplicação: HA

---

- alta disponibilidade
  - transações financeiras e comerciais
  - sistemas de reservas internacionais
  - aplicações na Internet (e-commerce)
- exemplos:
  - Clássicos: Tandem Nonstop & Stratus
  - Atuais: servidores & HA-clusters

sistemas on-line,  
não confundir  
com tempo real

aparecem também com o  
nome de money-critical



# Bibliografia

---

Avizienis, Laprie, Randell, Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. on dep. and secure comp. 2004



Johnson, B.W. **Fault Tolerance**, The Electrical Engineering Handbook, Ed. Richard C. Dorf, Boca Raton: CRC Press LLC, 2000





# Fundamentos de Tolerância a Falhas

---

*Conceitos básicos*

*Taisy Silva Weber*

# Conceitos básicos

---

- falha, erro e defeito
- dependabilidade
  - confiabilidade, disponibilidade, segurança e outros
- taxonomia

Avizienis, Laprie, Randell, Landwehr. **Basic Concepts and Taxonomy of Dependable and Secure Computing.** IEEE Trans. on dep. and secure comp. 2004

**conceitos básicos** encontrados em livros de SO, redes, arquitetura, além de grande número de artigos (Laprie, Avizienis, Nelson, Cristian, Schneider, Siewiorek, Rennels...)

# Falha, erro ou defeito?

---

- estado errôneo (ou erro)
  - se processamento posterior pode levar a defeito
- falha
  - causa física ou algorítmica do erro

falhas podem ser toleradas, defeitos não

**falha → erro → defeito**

válido aqui

*fault → error → failure*

alguns grupos no Brasil usam:  
falta → erro → falha (tolerância a falta)

# Falha, erro e defeito

---

**falha** é a causa do erro



Zzzzz...

**falha**

um sistema está em estado errôneo (em **erro**) se processamento posterior pode levar a um defeito



!!!

**erro**

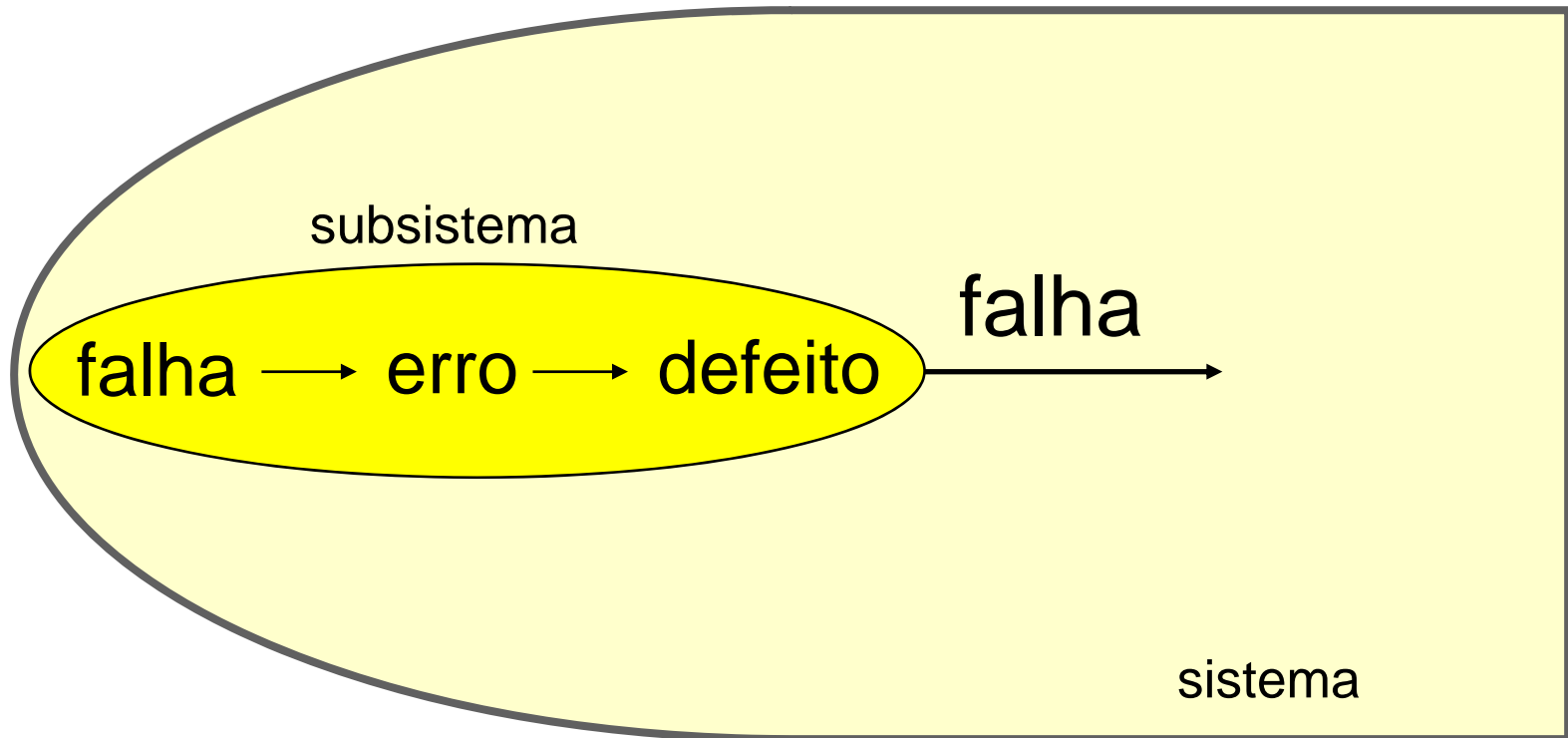
**defeito**



um **defeito** ocorre quando um sistema não fornece o serviço esperado

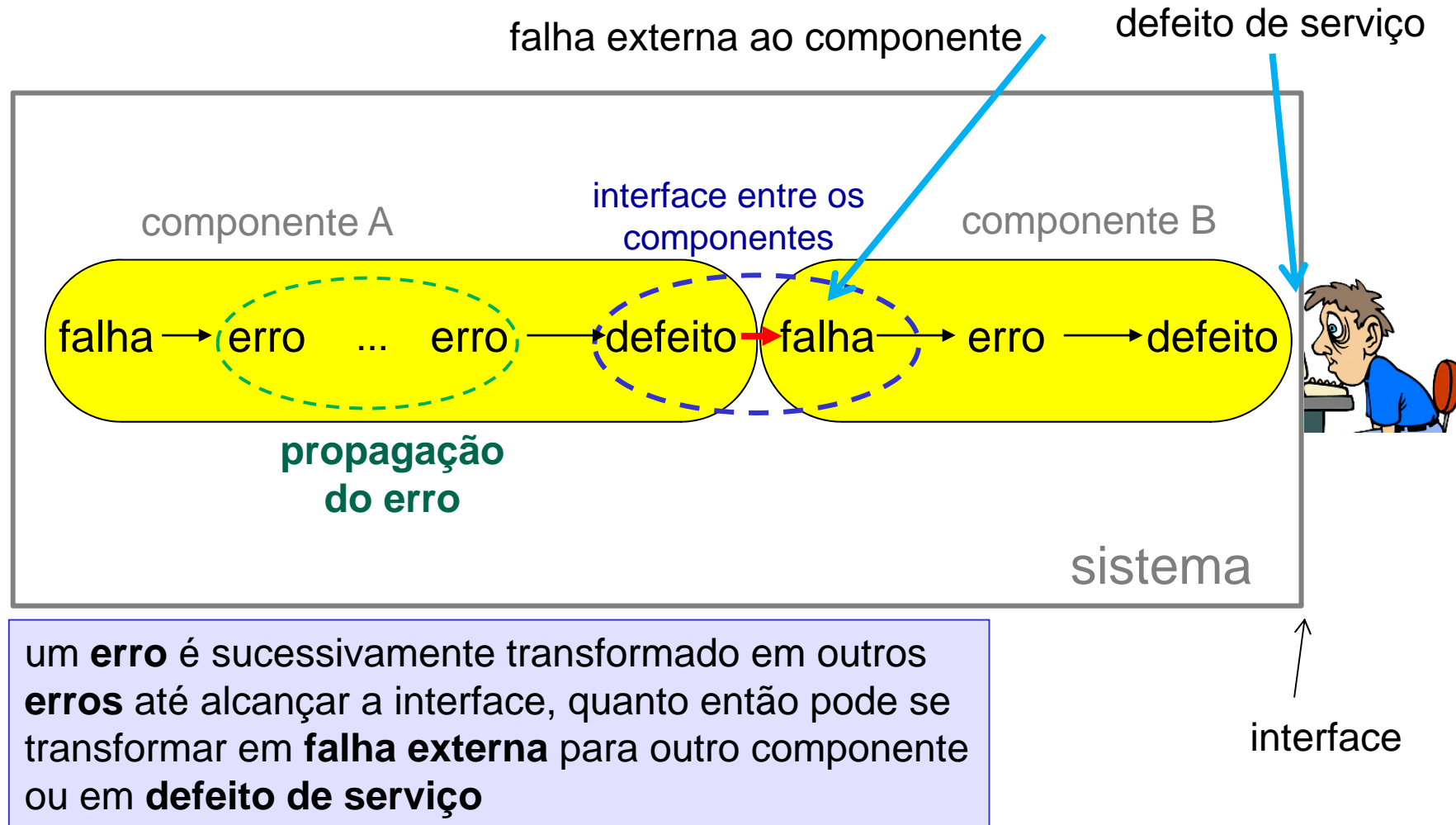
# Defeito de subsistema

---



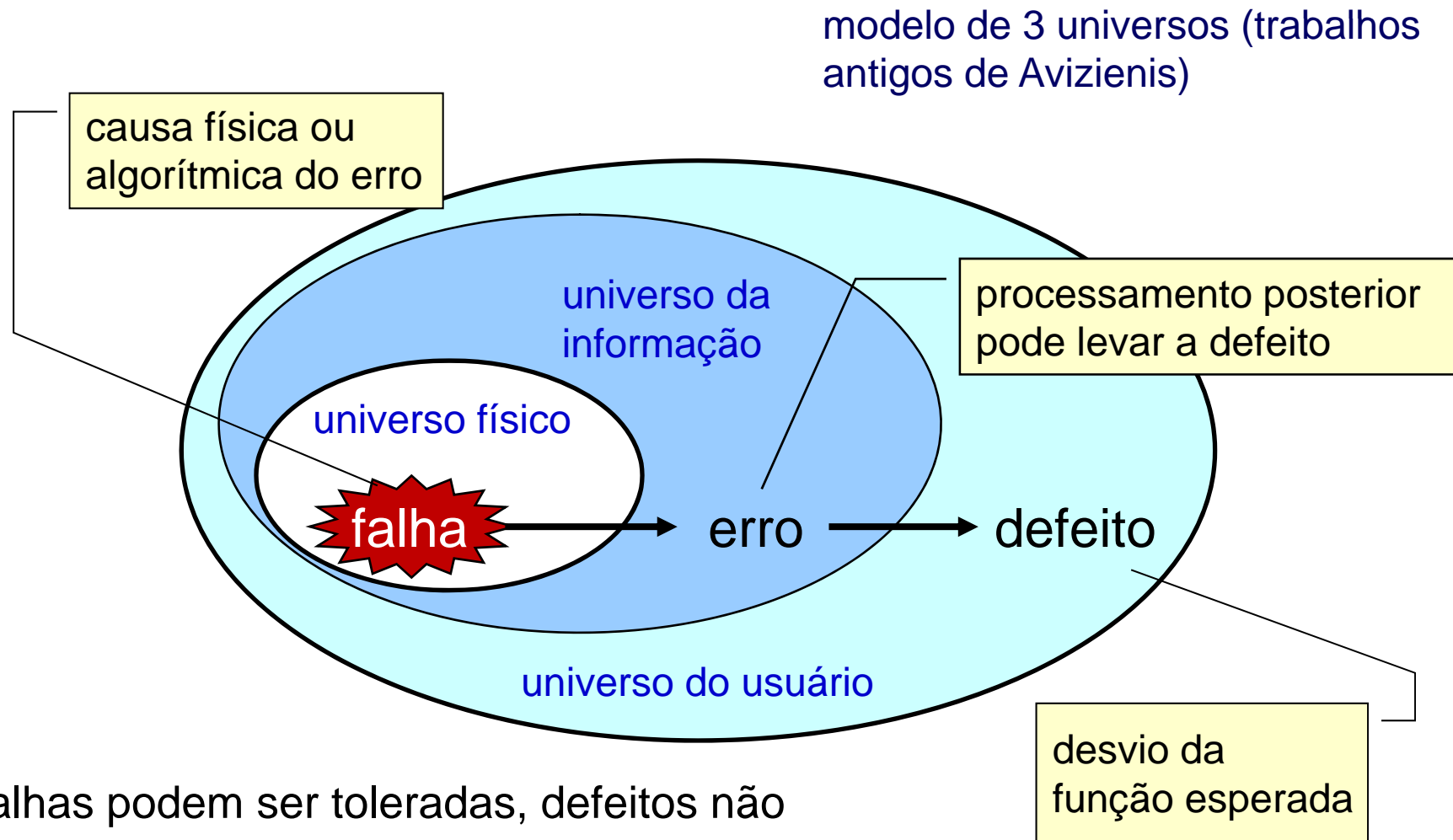
todo defeito é consequência de um erro, mas  
nem todo erro provoca um defeito

# Propagação de erro



# falha → erro → defeito

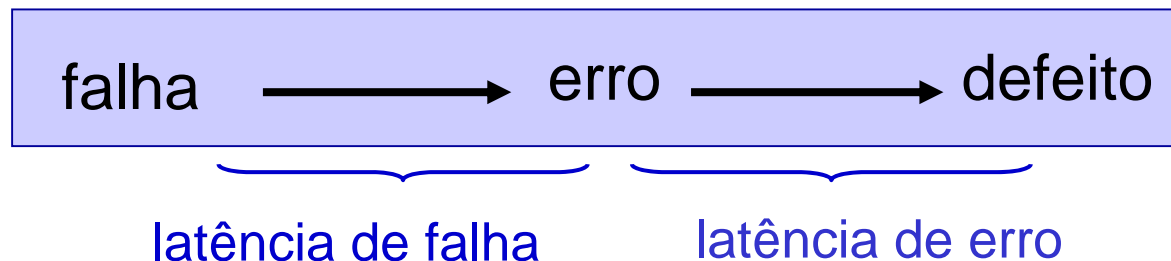
---



# Latência

---

- latência de falha
  - período de tempo desde a ocorrência da falha até a manifestação do erro devido aquela falha
- latência de erro
  - período de tempo desde a ocorrência do erro até a manifestação do defeito devido aquele erro

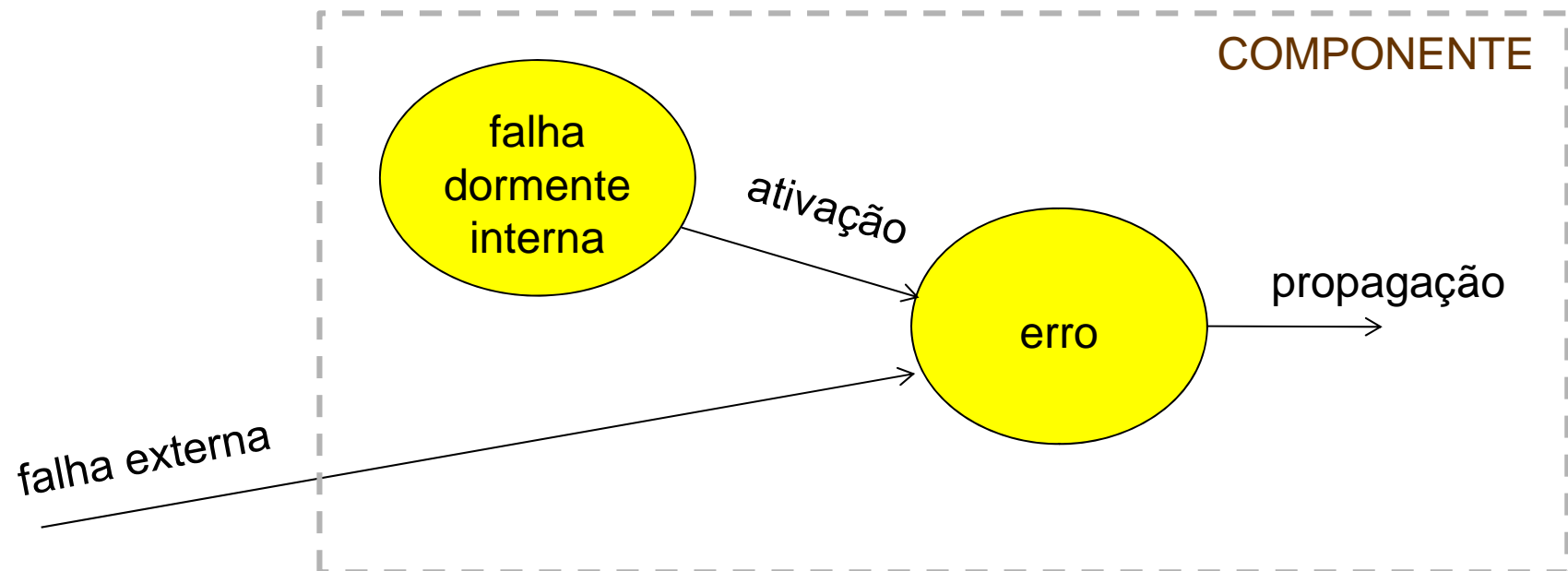




# Falha ativa

---

- falha
  - ativa quando produz um erro
  - inativa ou dormente



# Dormentes e latentes

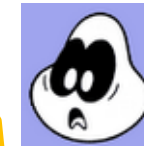
---

falhas presentes  
mas não ativadas  
são **dormentes**



Zzzzz...

**falha**



!!!

**erro**

erros presentes mas  
não detectados são  
**latentes**

**defeito**

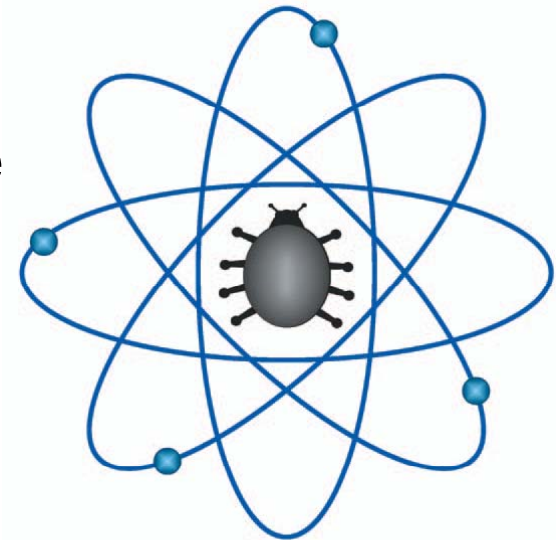


# Reprodutibilidade da falha

---

- falhas sólidas (*hard*)
  - ativação reproduzível
- falhas evasivas (*soft*)
  - ativação não reproduzível ou reproduzível muito dificilmente

falhas residuais  
geralmente soft



*Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate*, Grottko e Trivedi, IEEE Computer, fev. 2007.

# Dependabilidade

---

- *dependability*
  - habilidade de fornecer um serviço em que se pode depositar confiança
  - habilidade do sistema de evitar defeitos que sejam mais frequentes ou mais severos do que aceitável
- outras definições:
  - qualidade do serviço fornecido por um dado sistema
  - confiança **justificável** no serviço fornecido

# Atributos de dependabilidade

---

- confiabilidade (*reliability*):
  - continuidade do serviço correto
- disponibilidade (*availability*):
  - prontidão para serviço correto
- segurança (*safety*):
  - ausência de consequências catastróficas para o usuário ou ambiente
- integridade (*integrity*):
  - ausência de alterações impróprias no sistema
- *maintainability*:
  - facilidade de executar modificações e reparos



mais conhecidas

# Confiabilidade

---

*reliability*

- capacidade de **atender à especificação**
  - dentro de condições definidas
  - durante certo período de funcionamento
  - condicionado a estar operacional no início do período
- probabilidade que um sistema funcione **corretamente** durante um intervalo de tempo

# Confiabilidade: exemplo

---

- sistemas em que mesmo curtos períodos de operação incorreta são inaceitáveis
- aviação
  - tempo de missão: 10 a 12 horas



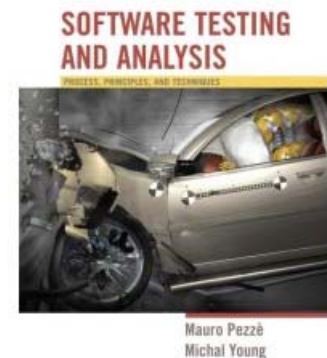
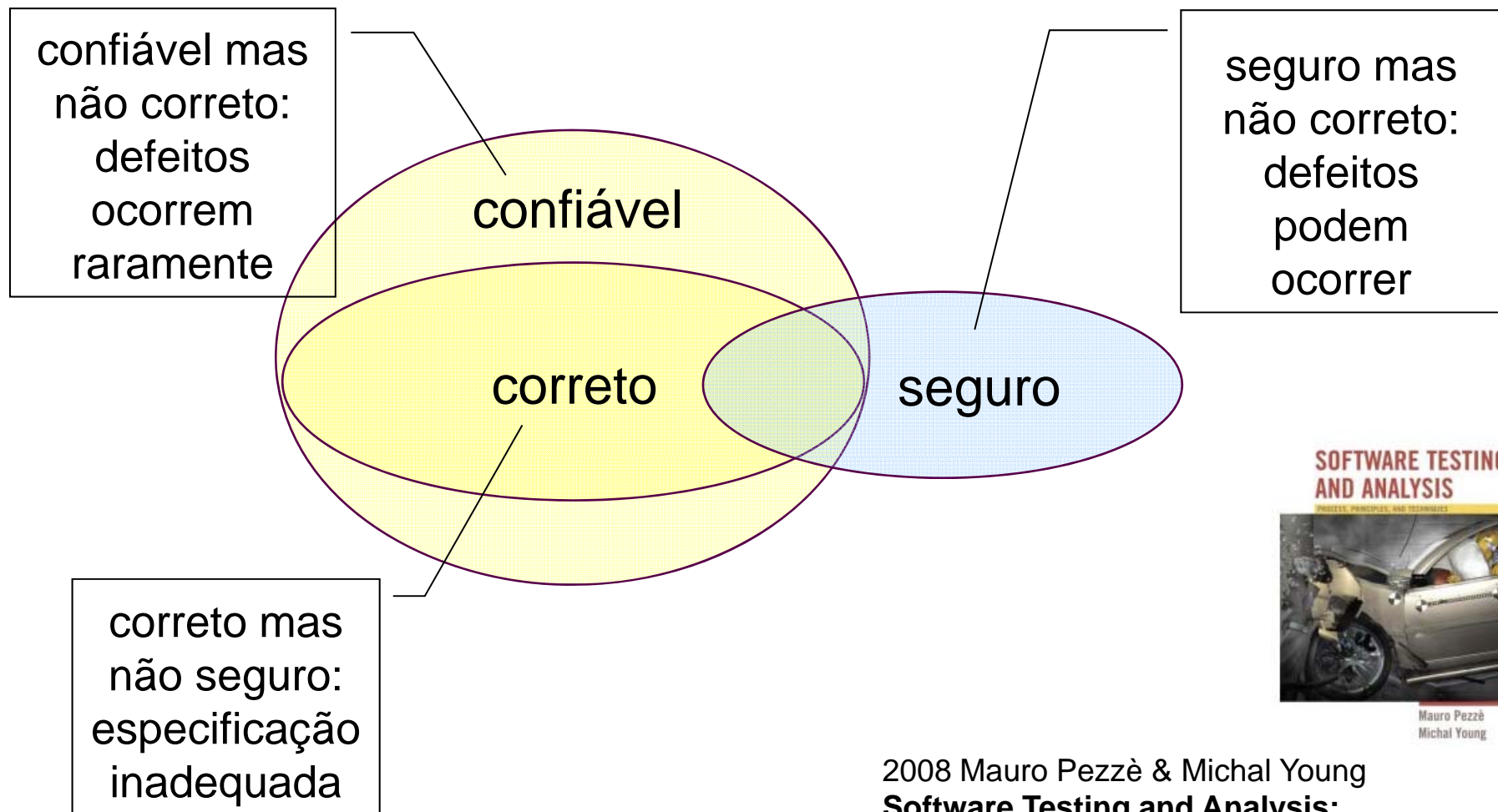
# Confiabilidade e correção

---

- confiabilidade é uma aproximação estatística da correção
  - **correção funcional** é sempre absoluta (um sistema é correto ou não)
  - confiabilidade de 100% = correção
- semelhança: ambas definidas em relação a uma **especificação**
- diferença:
  - confiabilidade é relacionada ao perfil de utilização
    - um sistema pode ser mais ou menos confiável dependendo de como é usado



# Confiabilidade, correção e safety



2008 Mauro Pezzè & Michal Young  
**Software Testing and Analysis:  
Process, Principles, and Techniques**

# Disponibilidade

*availability*

- alternância de períodos de funcionamento e reparo
  - um sistema pode ser altamente disponível mesmo apresentando períodos sem operabilidade

desde que esses  
períodos sejam  
curtos



# Disponibilidade vs confiabilidade

---



Taisy Weber

**disponibilidade** e **confiabilidade** são os atributos mais conhecidos e usados, muitas vezes aparecem como sinônimos de **dependabilidade**

não são sinônimos: um sistema pode ser altamente confiável e ter baixa disponibilidade

# Segurança funcional

---

*safety*

- ausência de consequências catastróficas

atributo usual na área de controle de processos industriais, transporte, aviação e instrumentação médica

necessita do conhecimento e especificação dos danos a serem evitados



# Outros atributos

---

- *maintainability*
  - facilidade de realizar a manutenção do sistema
- testabilidade
  - facilidade de realizar testes

probabilidade que um sistema com defeitos seja restaurado dentro de um dado período de tempo

relacionada a facilidade de manutenção

# Outros atributos

---

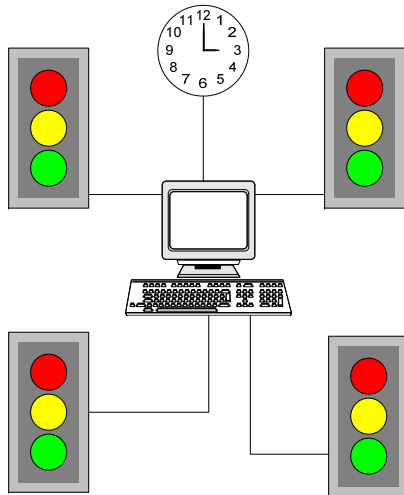
- *performability*
  - queda de desempenho provocada por falhas
- robustez
  - modo como o sistema apresenta defeito se usado fora das condições normais

sistema continua a operar, mas com queda de desempenho

um sistema é robusto quando em condições inesperadas apresenta defeito, mas sem provocar danos consideráveis

# Exemplo de atributos

---



- correção, confiabilidade
  - permitir tráfego de acordo com especificação e escalonamento
- robustez
  - degradação de função se possível:
    - amarelo piscando é melhor que todos desligados
    - desligados é melhor que todos verdes
- segurança
  - nunca todos verdes

# security & safety

---

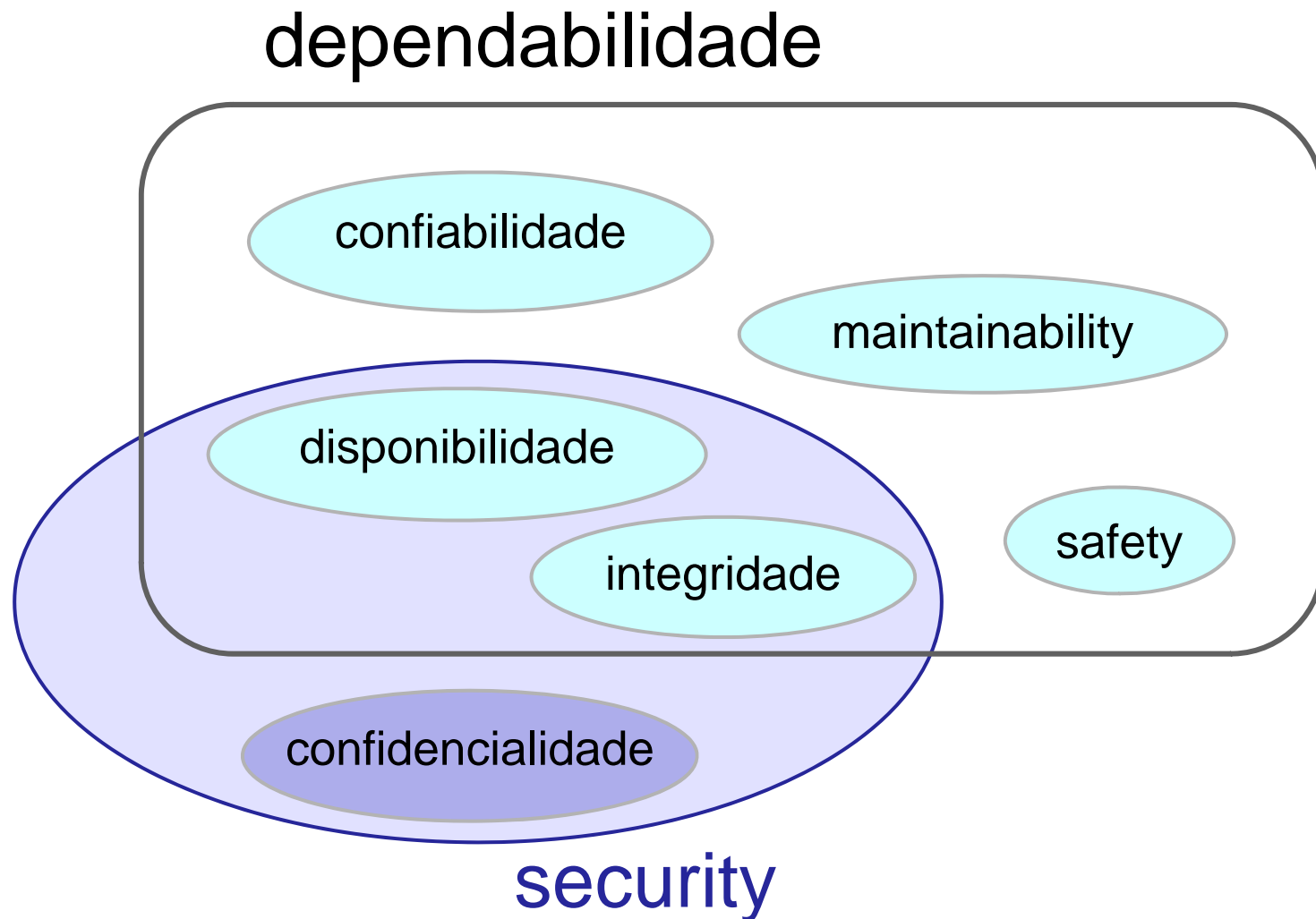
- safety: segurança funcional
  - ausência de consequências catastróficas para o usuário ou ambiente
- security: segurança computacional
  - **confidencialidade, integridade, autenticidade**
  - autenticidade
    - disponibilidade apenas para ações autorizadas
  - integridade é um atributo comum a dependabilidade e *security*

desejável que um sistema seja seguro nas duas acepções do termo



# security & safety

---



# Termos relacionados

---

- Dependability
  - habilidade do sistema de evitar defeitos que sejam mais frequentes ou mais severos do que aceitável
- High Confidence
  - consequências do comportamento do sistema são perfeitamente compreendidas e previsíveis
- Survivability
  - capacidade do sistema de cumprir sua missão no tempo adequado
- Trustworthiness
  - garantia que o sistema vai funcionar como esperado

# Trustworthiness

---

- fidedignidade (?)
- atributos de software
  - correção
  - segurança funcional (safety)
  - qualidade de serviço
    - disponibilidade
    - confiabilidade
    - desempenho
  - segurança computacional (security)
  - privacidade

no documento Grandes  
Desafios da Computação,  
SBC 2007, fidedignidade  
aparece como tradução de  
*dependability*

Wilhelm Hasselbring, Ralf Reussner  
**Toward Trustworthy Software Systems**  
Computer, april 2006

# Falhas são inevitáveis

---

- problemas de especificação
- problemas de projeto e implementação
- componentes defeituosos
  - imperfeições de manufatura
  - fadiga e/ou envelhecimento
- distúrbios externos
  - radiação,
  - interferência eletromagnética,
  - variações ambientais (temperatura, pressão, umidade),
  - problemas de operação

# Bibliografia

---

- artigos
  - Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, Carl Landwehr. ***Basic Concepts and Taxonomy of Dependable and Secure Computing***. IEEE trans. on dependable and secure computing, jan 2004, pp 11-33
  - Grottke e Trivedi, ***Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate***, IEEE Computer, fev. 2007.
  - Wilhelm Hasselbring, Ralf Reussner. ***Toward Trustworthy Software Systems***. Computer, april 2006

# Bibliografia

---

- livros
  - Koren, Israel, e C. Mani Krishna. 2007. **Fault-Tolerant Systems**. Morgan Kaufmann
  - Birman, K. **Reliable distributed systems**. Springer, New York, 2005.
  - Pezzè, Mauro, e Michal Young. 2008. **Software Testing and Analysis: Process, Principles, and Techniques**. Wiley.
- capítulo de livro
  - Johnson, Barry. An introduction to the design na analysis of the fault-tolerant systems, cap 1. **Fault-Tolerant System Design**. Prentice Hall, 1996

# Fundamentos de Tolerância a Falhas

---

*Taxonomia*  
*Taisy Silva Weber*

# Falhas: classificação

---

- inúmeras classificações diferentes
  - a classificação muda mesmo entre artigos do mesmo autor
    - exemplo: artigos do Avizienis
- classificação não é lei
  - apenas ajuda a entender os fenômenos e falar a mesma língua
- falhas maliciosas
  - mais importantes agora

Avizienis, Laprie, Randell, Landwehr. **Basic Concepts and Taxonomy of Dependable and Secure Computing.** IEEE Trans. on dep. and secure comp. 2004

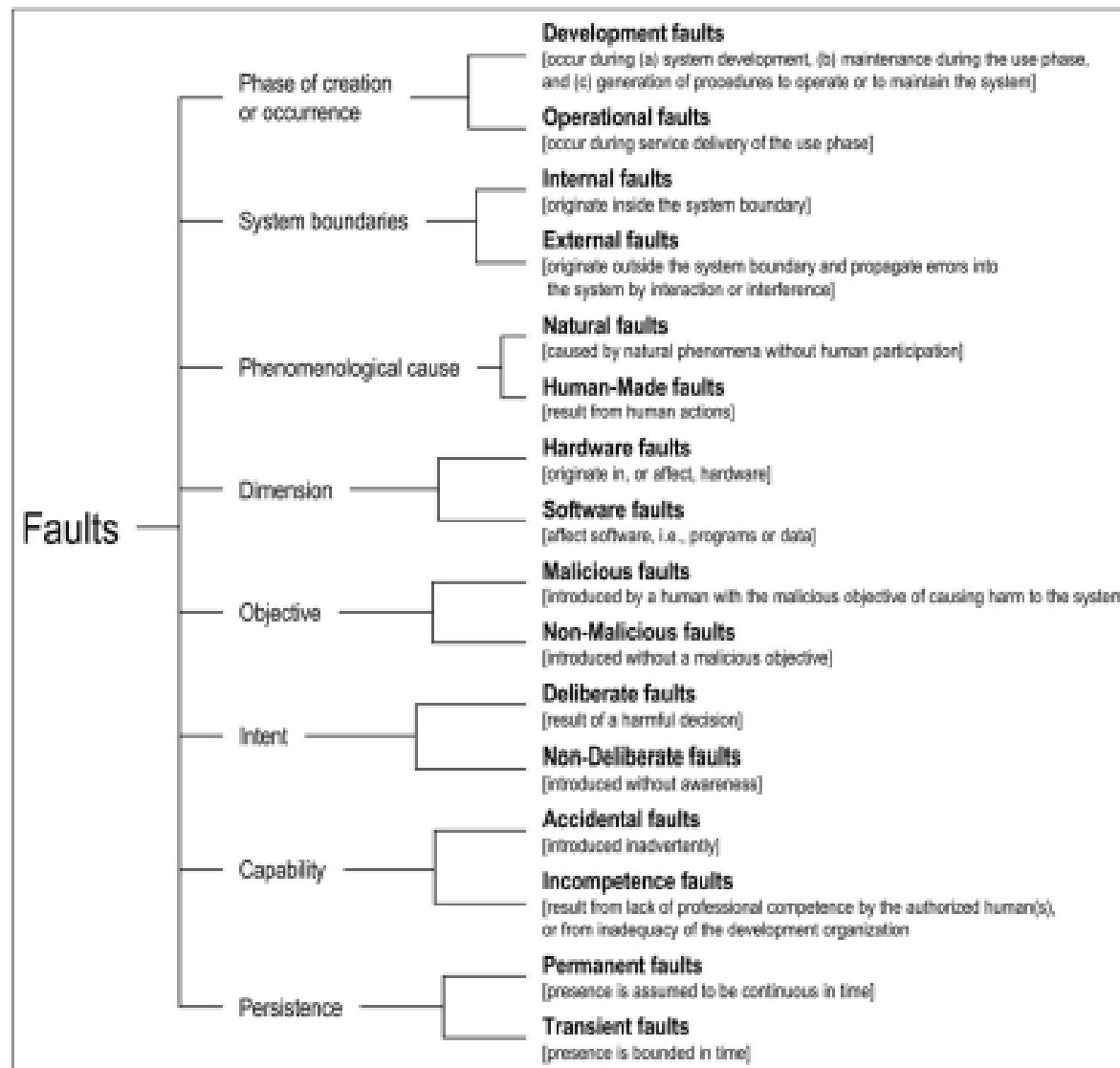


# Classes elementares de falhas

---

- fase:
  - desenvolvimento
  - ou operacional
- limites:
  - interna
  - externa
- causa:
  - natural
  - humana
- dimensão:
  - falha de hardware
  - falha de software
- objetivo:
  - maliciosa
  - não maliciosa
- intenção:
  - deliberada
  - não deliberada
- capacidade:
  - acidental
  - devida a incompetência
- persistência:
  - permanente
  - temporária

# Classes de falhas



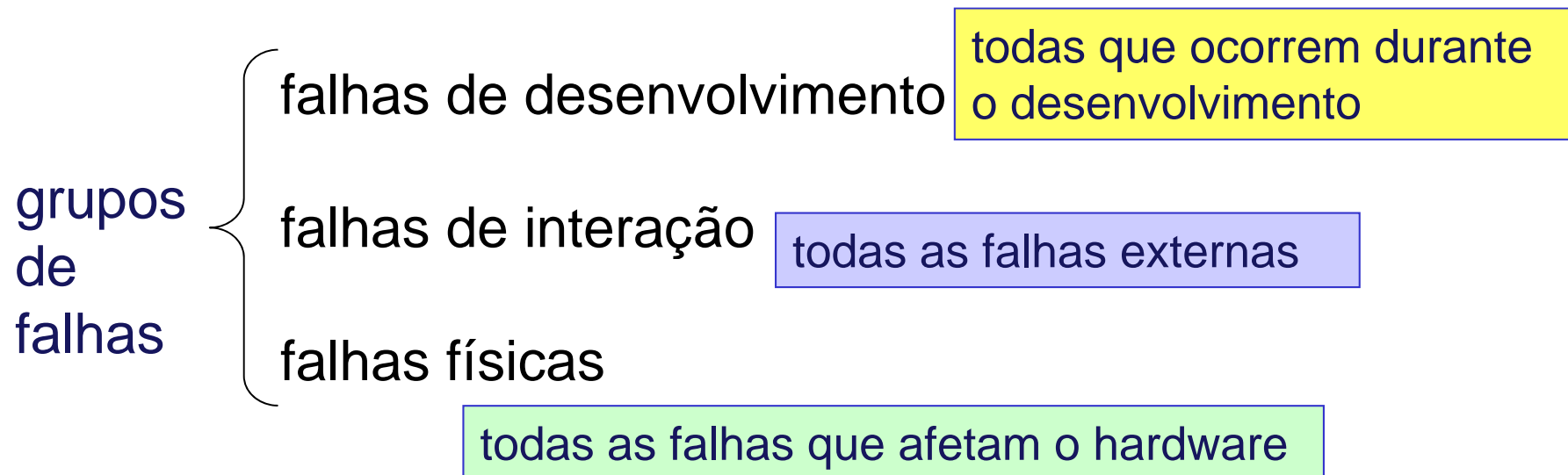
Avizienis, Laprie, Randell, Landwehr. **Basic Concepts and Taxonomy of Dependable and Secure Computing**. 2004

FIGURA 4

# Classes elementares de falhas

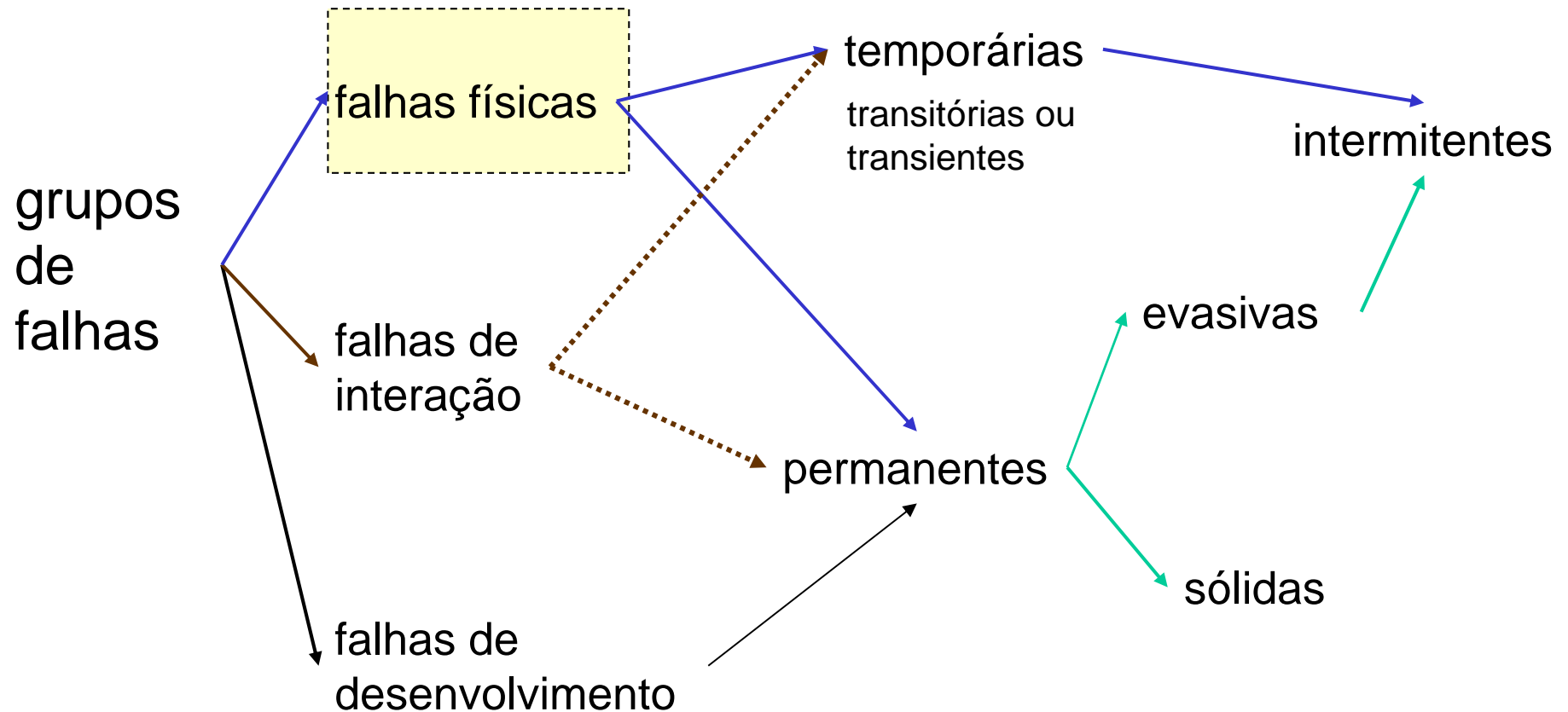
---

- 8 classes
  - podem ser combinadas entre si
  - nem todas as combinações fazem sentido
- as classes combinadas levam a 3 grupos parcialmente sobrepostos



# Falhas: grupos

---



falhas físicas, que afetam diretamente o hardware, foram mais estudadas

# Classes elementares de falhas

---

- fase:
  - desenvolvimento
  - ou operacional
- limites:
  - interna
  - externa
- causa:
  - natural
  - humana
- dimensão:
  - falha de hardware
  - falha de software
- objetivo:
  - maliciosa
  - não maliciosa
- intenção:
  - deliberada
  - não deliberada
- capacidade:
  - acidental
  - devida a incompetência
- persistência:
  - permanente
  - temporária

# Falhas naturais

classe: causa

- naturais falhas causadas por fenômenos naturais sem participação humana
  - durante desenvolvimento
    - problemas de produção (*production defects*)
  - durante operação
    - interna: processos naturais de envelhecimento e deterioração física
    - externa: processos naturais originados fora dos limites do sistema que causam interferência física
      - radiação
      - transientes de potência
      - ruídos nas linhas de sinais, ...

# Falhas humanas

classe: causa

- fase:
    - desenvolvimento
    - ou operacional
  - limites:
    - interna
    - externa
  - causa:
    - natural
    - **humana**
  - dimensão:
    - de hardware
    - de software
- objetivo:
    - maliciosa
    - não maliciosa
  - intenção:
    - deliberada
    - não deliberada
  - capacidade:
    - acidental
    - devida a incompetência
  - persistência:
    - permanente
    - temporária



# Falhas humanas

fenômenos naturais  
sem participação  
humana



área de  
security

causa

natural

humana

objetivo

não-maliciosa

maliciosa

intenção

não intencional

intencional

intencional

capacidade

acidental

incompetência

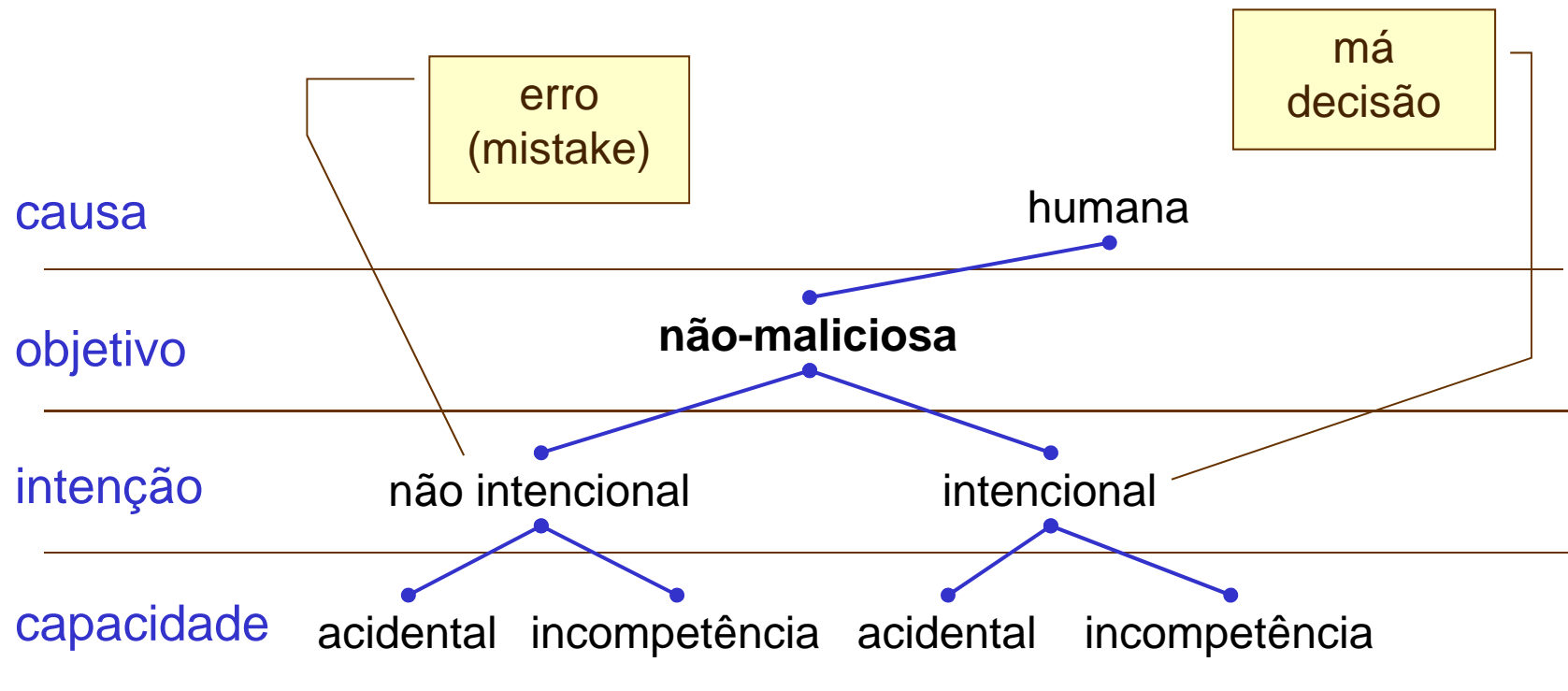
acidental

incompetência



# Falhas humanas não-maliciosas

podem ocorrer na **fase** de desenvolvimento ou na **fase** de operação do sistema



# Defeitos

---

- de serviço
- de desenvolvimento
- de dependabilidade e segurança



# Defeitos de serviço

---

de serviço  
de desenvolvimento  
de dependabilidade e segurança

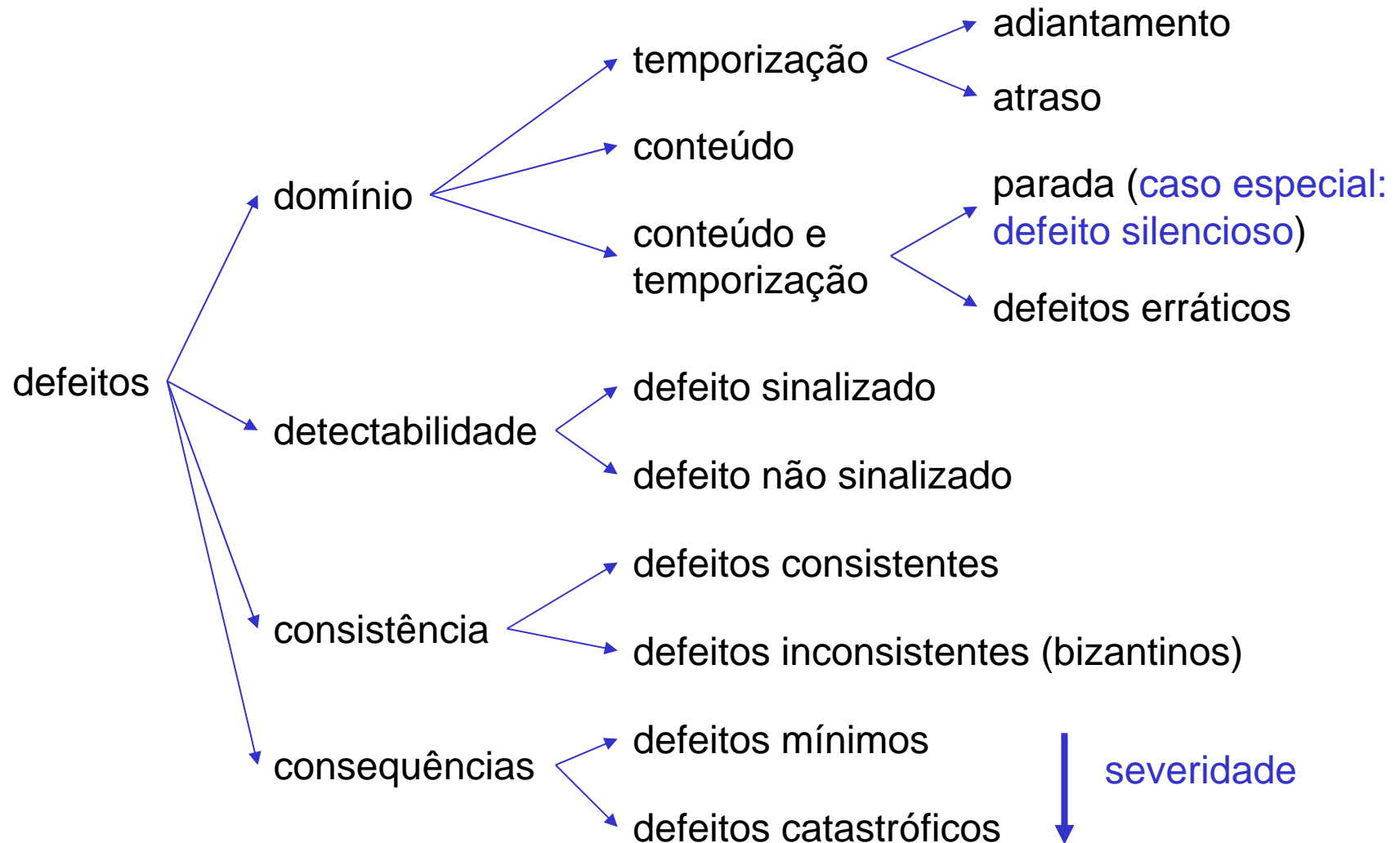
- quando o serviço oferecido se desvia de sua **função**

não quando se desvia da descrição da função, ou seja, da sua especificação

- **cuidado:**
  - a especificação do sistema identifica se um sistema é correto ou não
  - **mas** a especificação pode conter falhas

# Defeitos de serviço: pontos de vista

---



# Sistemas com controle de defeito

---

- sistemas projetados e controlados para apresentar defeito apenas nos modos de defeitos descritos na sua especificação de dependabilidade
- fail-halt ou fail-stop
  - defeitos de parada apenas
- fail-passive
  - serviço travado (congelado)
- fail-silent
  - defeito silencioso
- fail-safe
  - severidade mínima

# Defeitos de desenvolvimento

---

de serviço  
de desenvolvimento  
de dependabilidade e segurança

- tipos
  - defeitos completos de desenvolvimento
  - defeitos parciais
  - defeitos que só se manifestam na fase operacional
- origem
  - falhas de desenvolvimento
    - introduzidas por desenvolvedores ou
    - ferramentas de desenvolvimento ou
    - métodos de produção
- aspectos
  - orçamento
  - prazos

# Defeitos de desenvolvimento: causas

---

- **complexidade** do sistema subestimada
  - especificações incompletas ou com falhas
  - número excessivo de mudanças na especificação
  - projeto inadequado com respeito a funcionalidade ou desempenho
  - muitas falhas de desenvolvimento
  - capacidade inadequada de remoção de falhas
  - dependabilidade ou segurança computacional insuficiente
  - falha na estimativa dos custos de desenvolvimento

# Defeitos de dependabilidade

---

de serviço  
de desenvolvimento  
de dependabilidade e segurança

- a especificação de dependabilidade deve conter:
  - os objetivos de cada um dos atributos: disponibilidade, confiabilidade, segurança funcional, integridade, facilidade de manter, ...
  - identificação das classes de falhas
  - ambiente de uso (operação)
  - essa especificação também pode conter falhas
- defeito de dependabilidade
  - quando o sistema sofre defeitos de serviço mais frequentes ou severos do que o aceitável

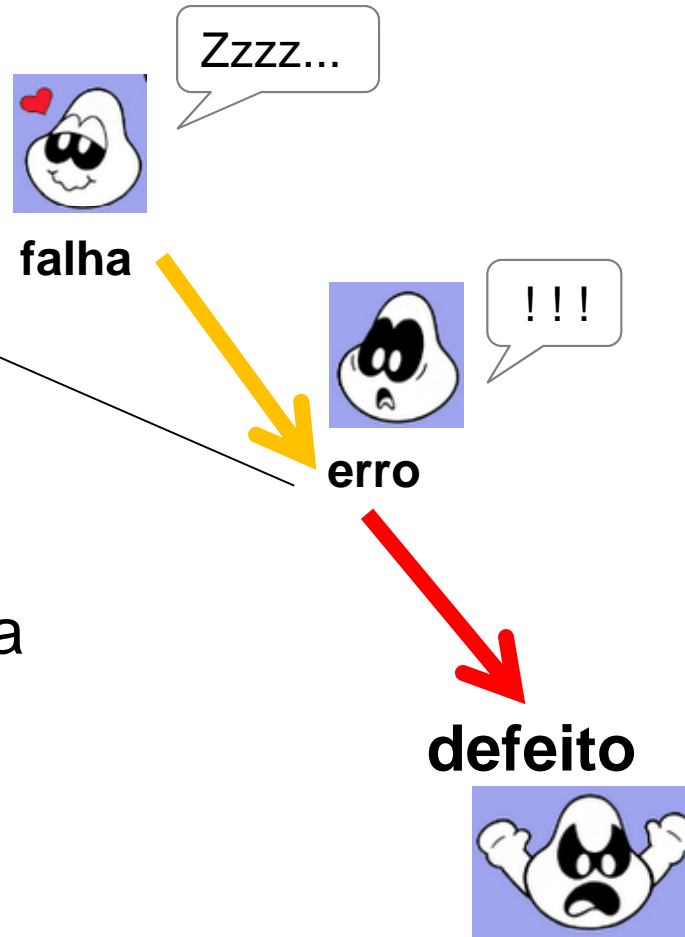


# Erros

---

erros presentes mas  
não detectados são  
**latentes**

um erro é detectado se  
sua presença é indicada  
por uma mensagem de  
erro ou sinal de erro



# Erros

---

- classificação considerando:

**defeitos** de serviço  
que originam

a **falha** que originou  
o erro e seu  
espalhamento

**número de bits**  
afetados (na área de  
códigos de detecção  
e correção de erros)

erro **simples**: falha afetou um  
único componente

erros **múltiplos** relacionados:  
falha afetou mais de um  
componente

erro simples, erro duplo,  
erro triplo, rajada, etc ..

# Bibliografia

---

- artigos
  - Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, Carl Landwehr. *Basic Concepts and Taxonomy of Dependable and Secure Computing*. IEEE trans. on dependable and secure computing, V. 1, n. 1, jan 2004, pp 11-33
  - VINCENZO DE FLORIO and CHRIS BLONDIA. *A Survey of Linguistic Structures for Application-Level Fault Tolerance*. ACM Computing Surveys, Vol. 40, No. 2, April 2008.
- capítulo de livro
  - Johnson, Barry. An introduction to the design na analysis of the fault-tolerante systems, cap 1. **Fault-Tolerant System Design**. Prentice Hall, New Jersey, 1996

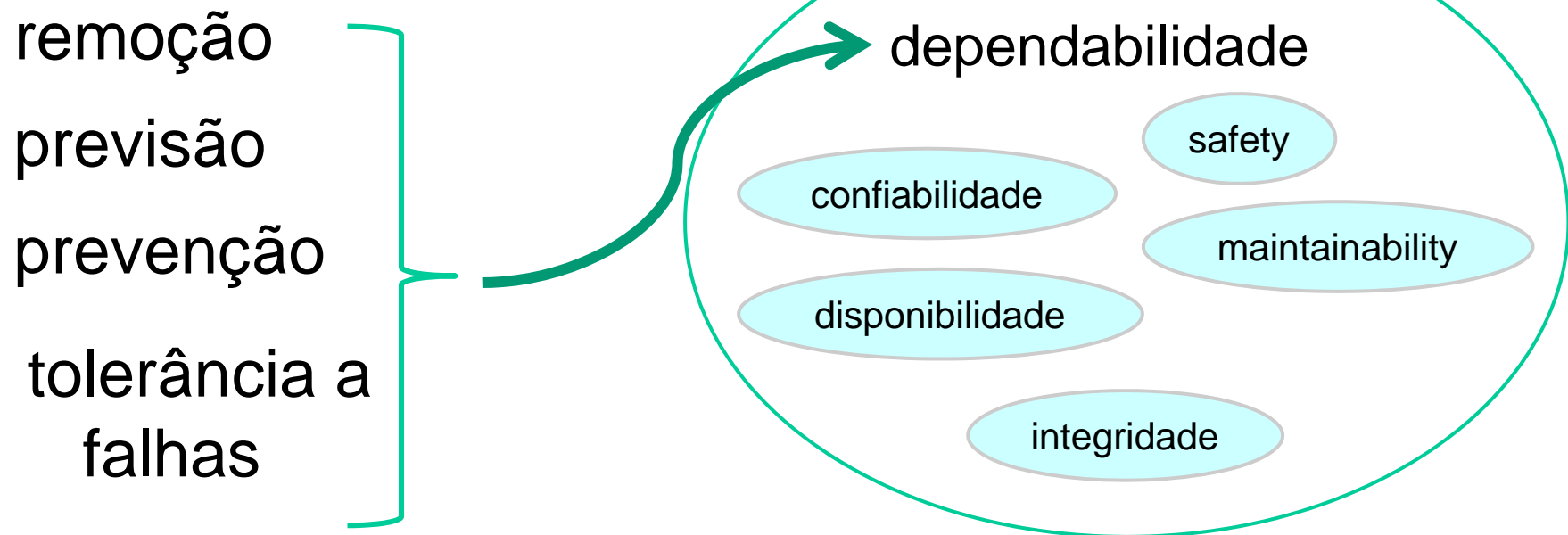
# Técnicas de Tolerância a Falhas

---

*Taisy Silva Weber*  
*UFRGS*

# Meios para alcançar dependabilidade

---

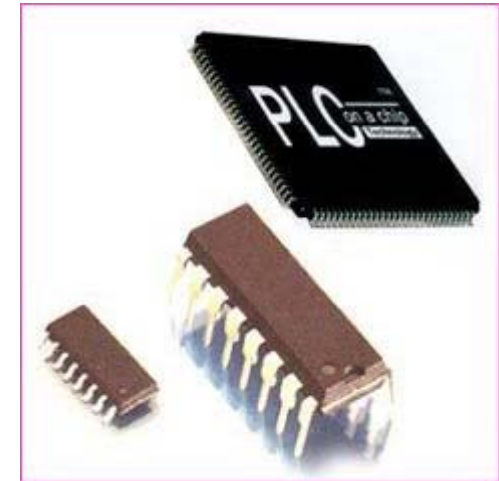


dependabilidade depende de **decisões** de projeto:  
para alcançar dependabilidade (e seus  
atributos) são necessária **técnicas de projeto**  
adequadas

# Dependabilidade sem TF?

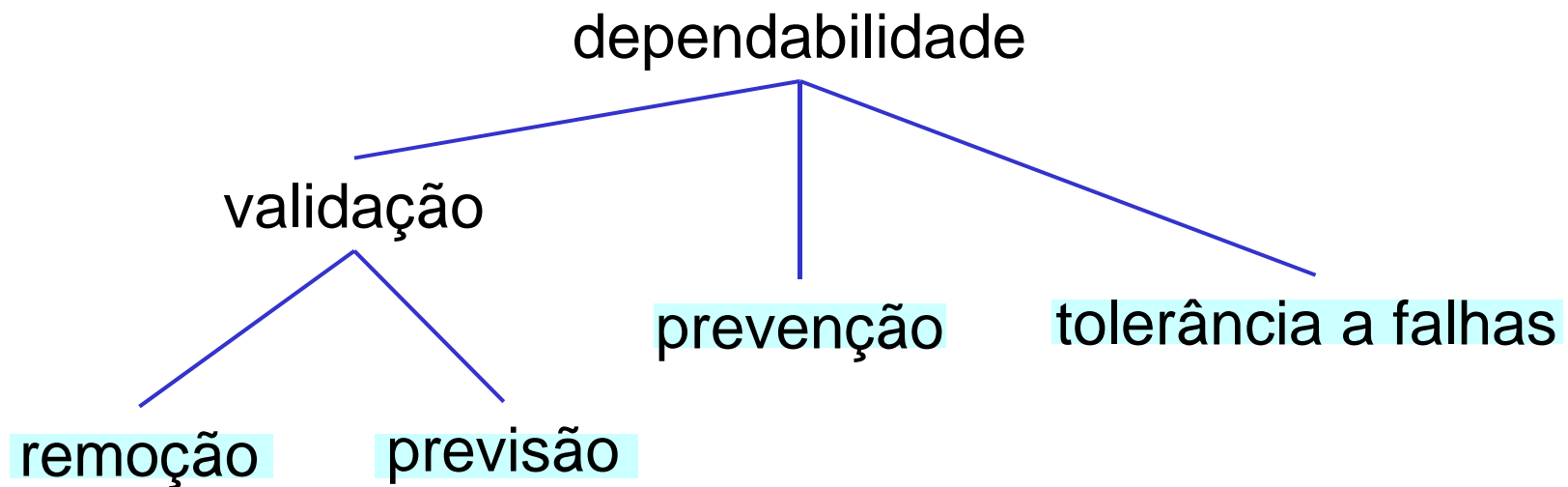
---

- dependabilidade pode ser alcançada sem TF
  - bons componentes podem levar a uma boa confiabilidade dos sistema
  - bons processos de produção e teste resultam em aumento de dependabilidade
  - manutenção frequente aumenta a qualidade



# Meios

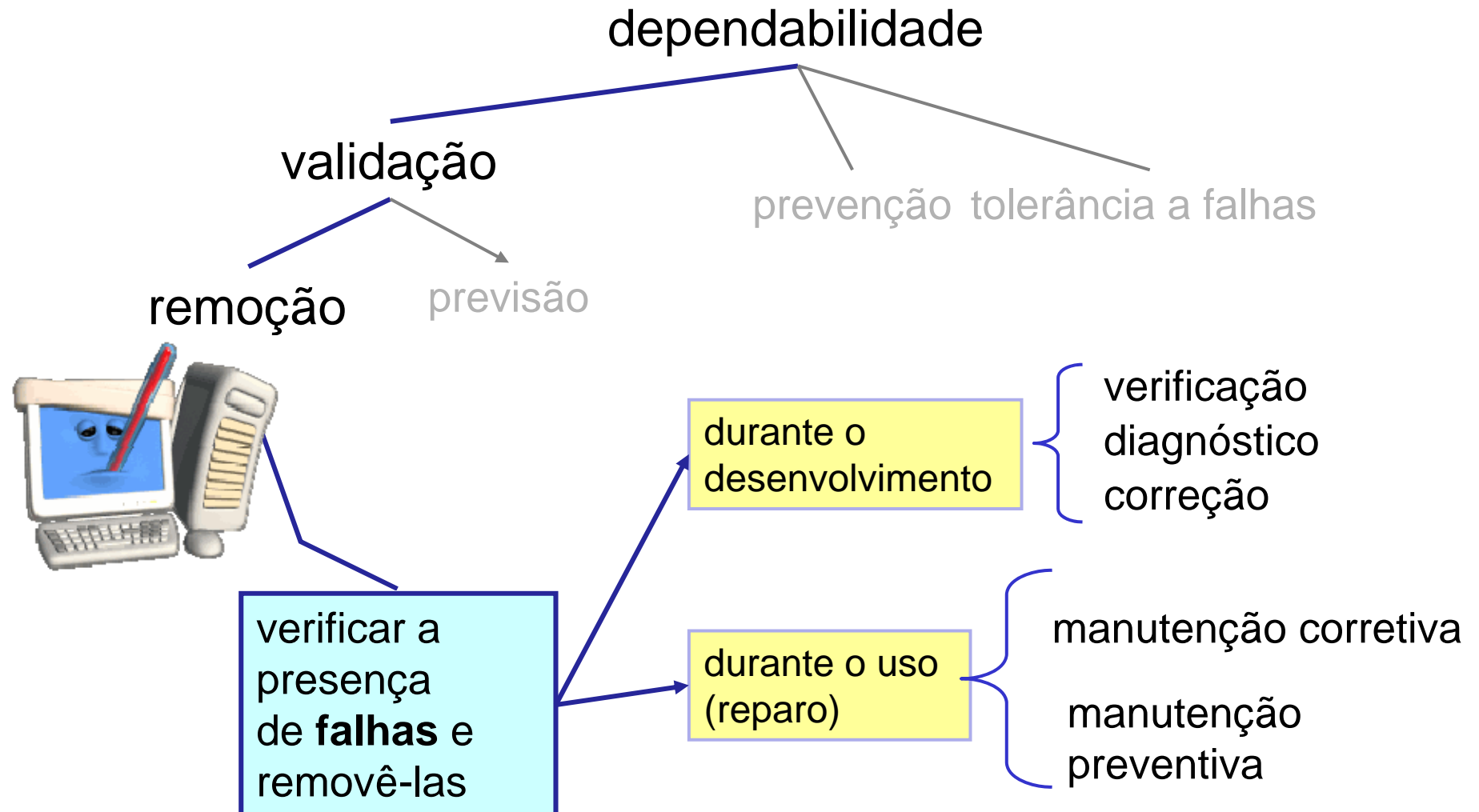
---



Avizienis - **meios** para alcançar dependabilidade: **remoção, previsão, prevenção e tolerância a falhas**

# Meios: validação

---

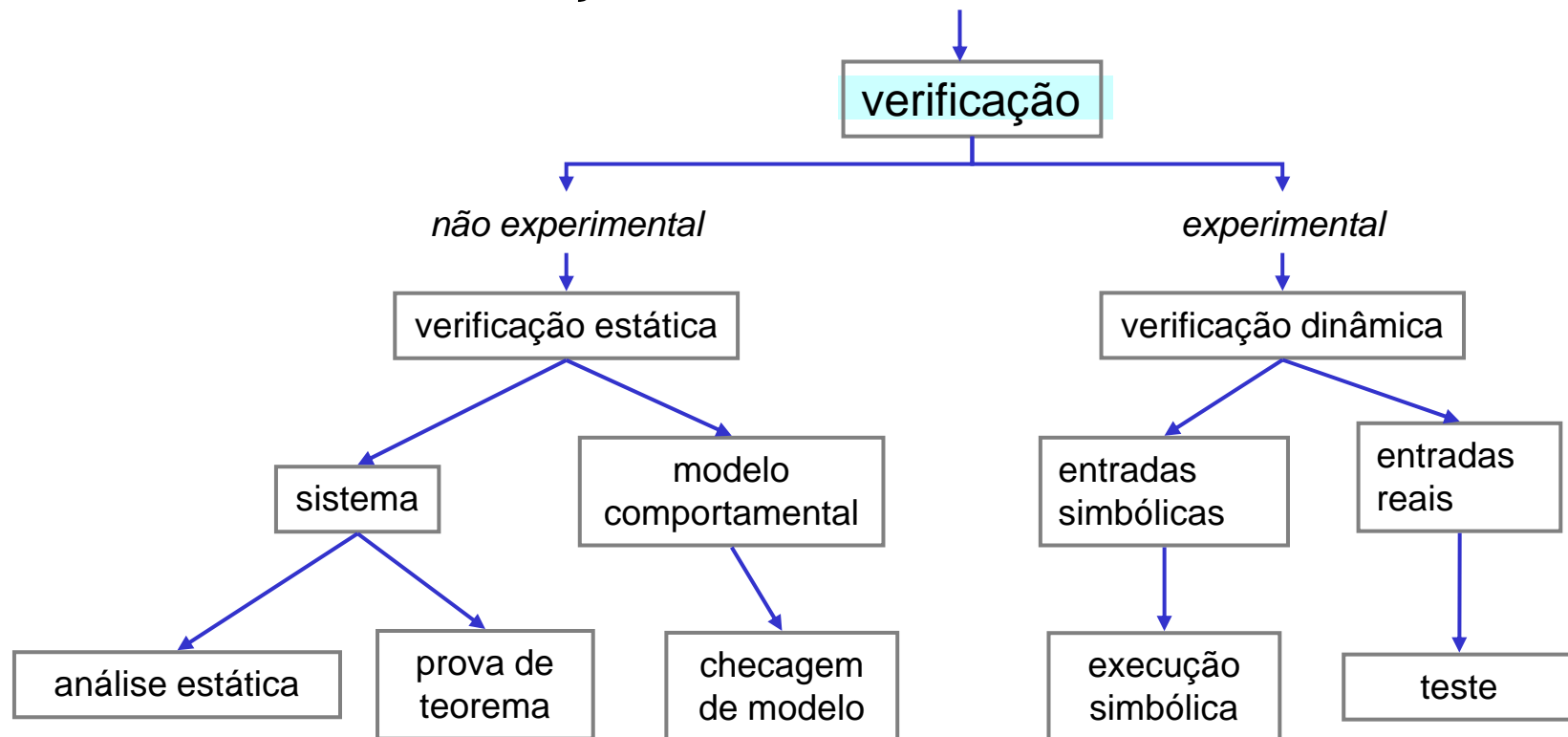




# Meios: remoção > verificação

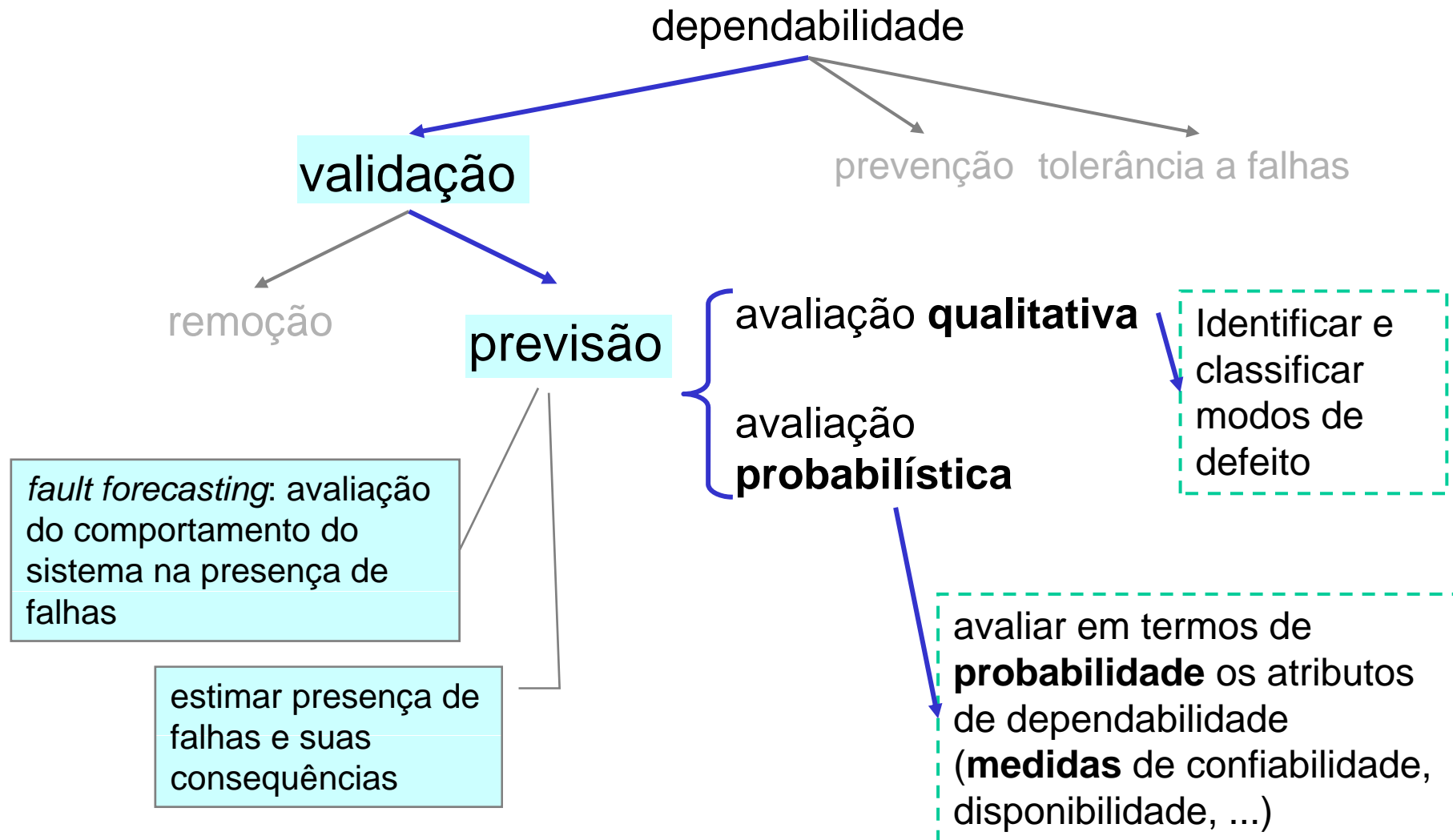
durante do desenvolvimento

dependabilidade → validação → remoção



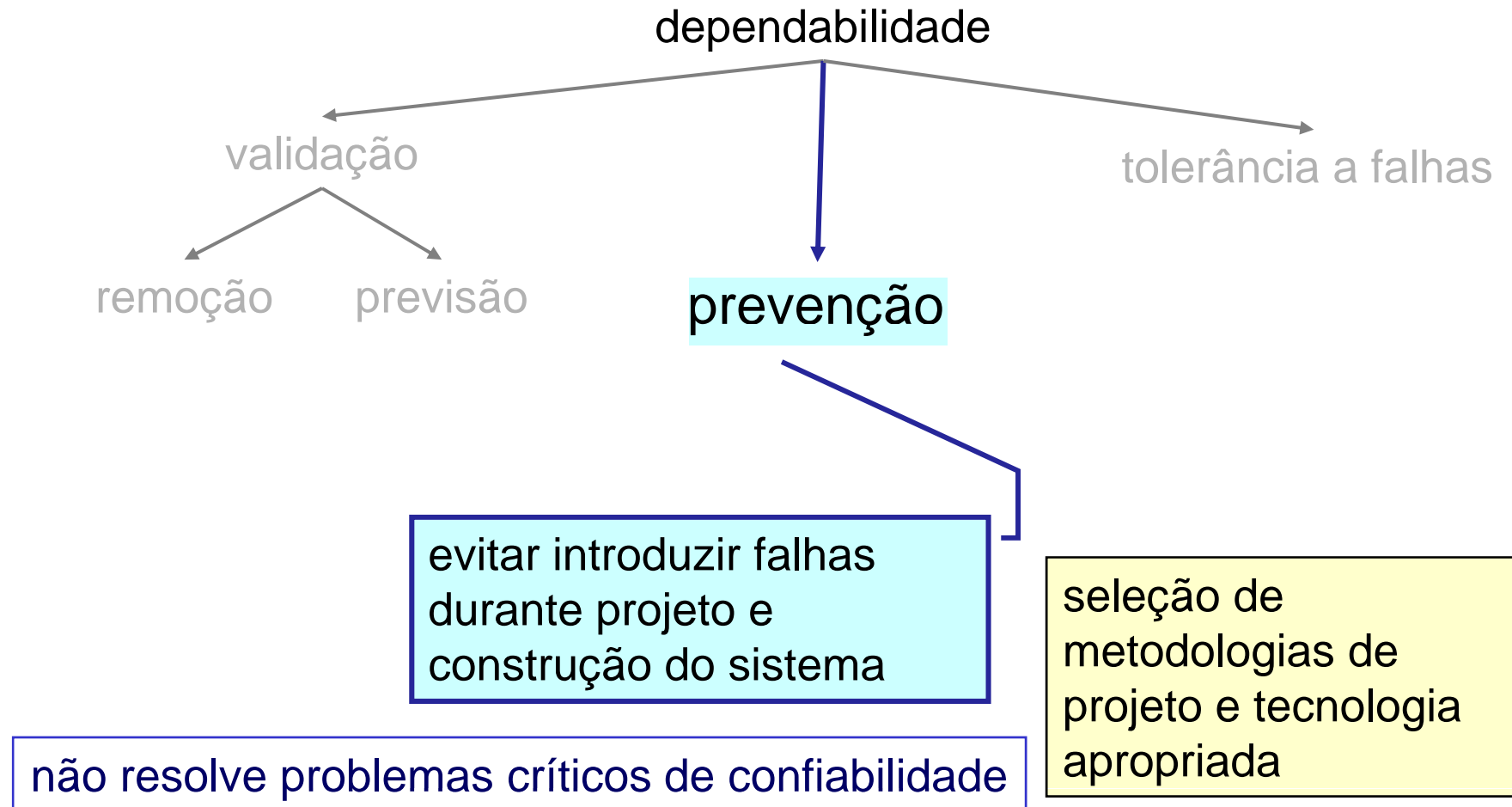
# Meios: previsão

---



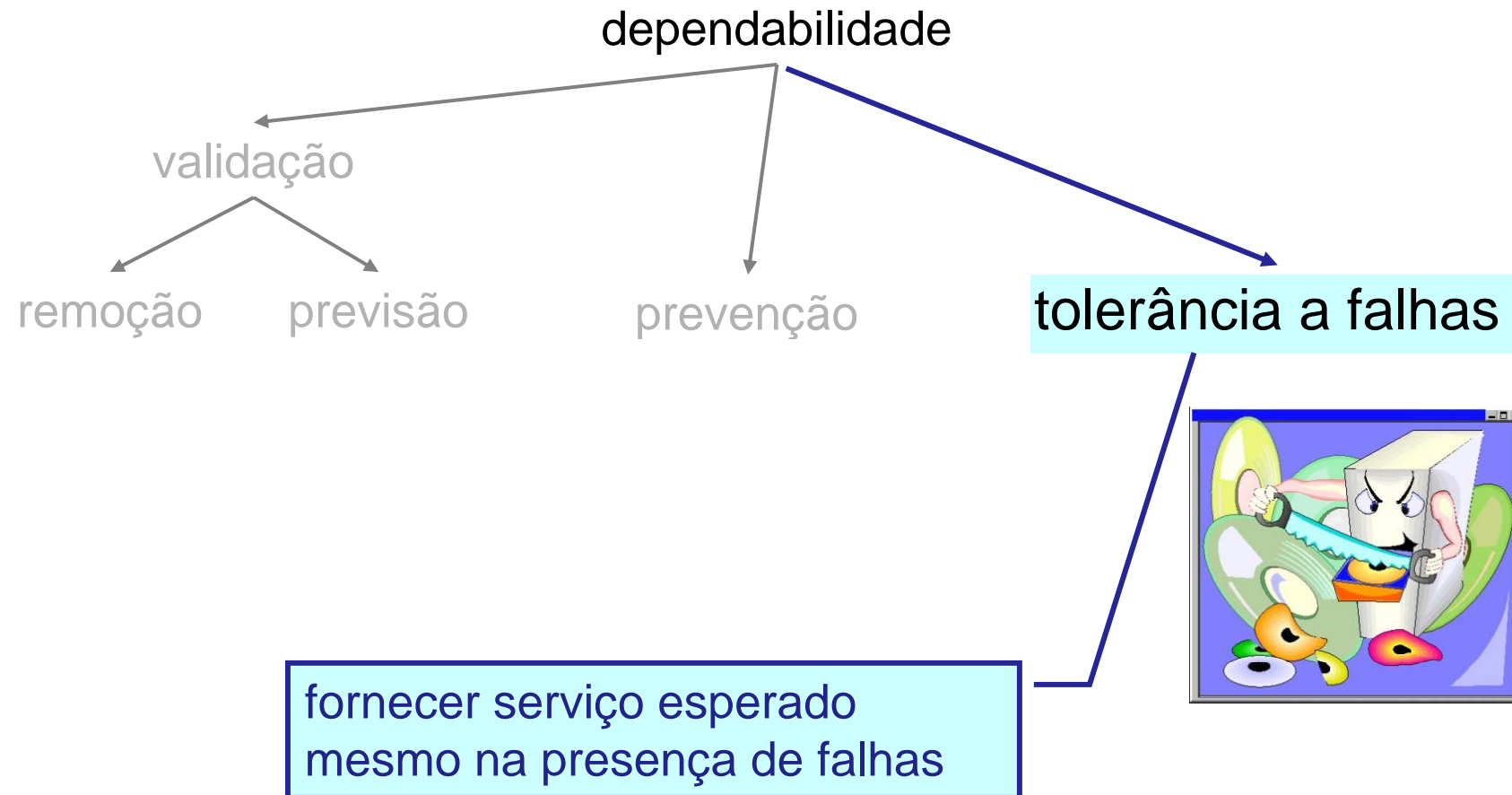
# Meios: prevenção

---



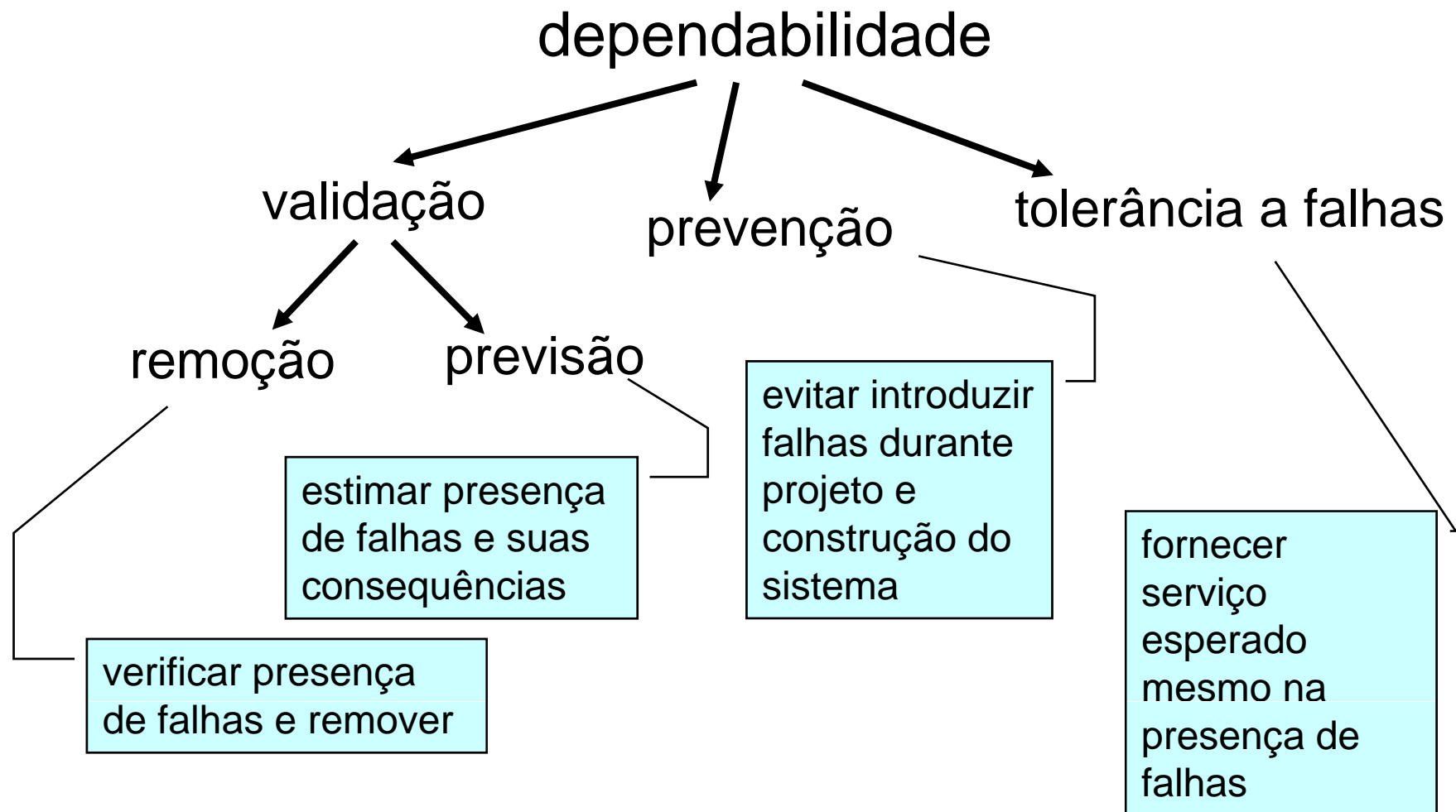
# Meios: tolerância a falhas

---



# Meios

---



# Técnicas de TF

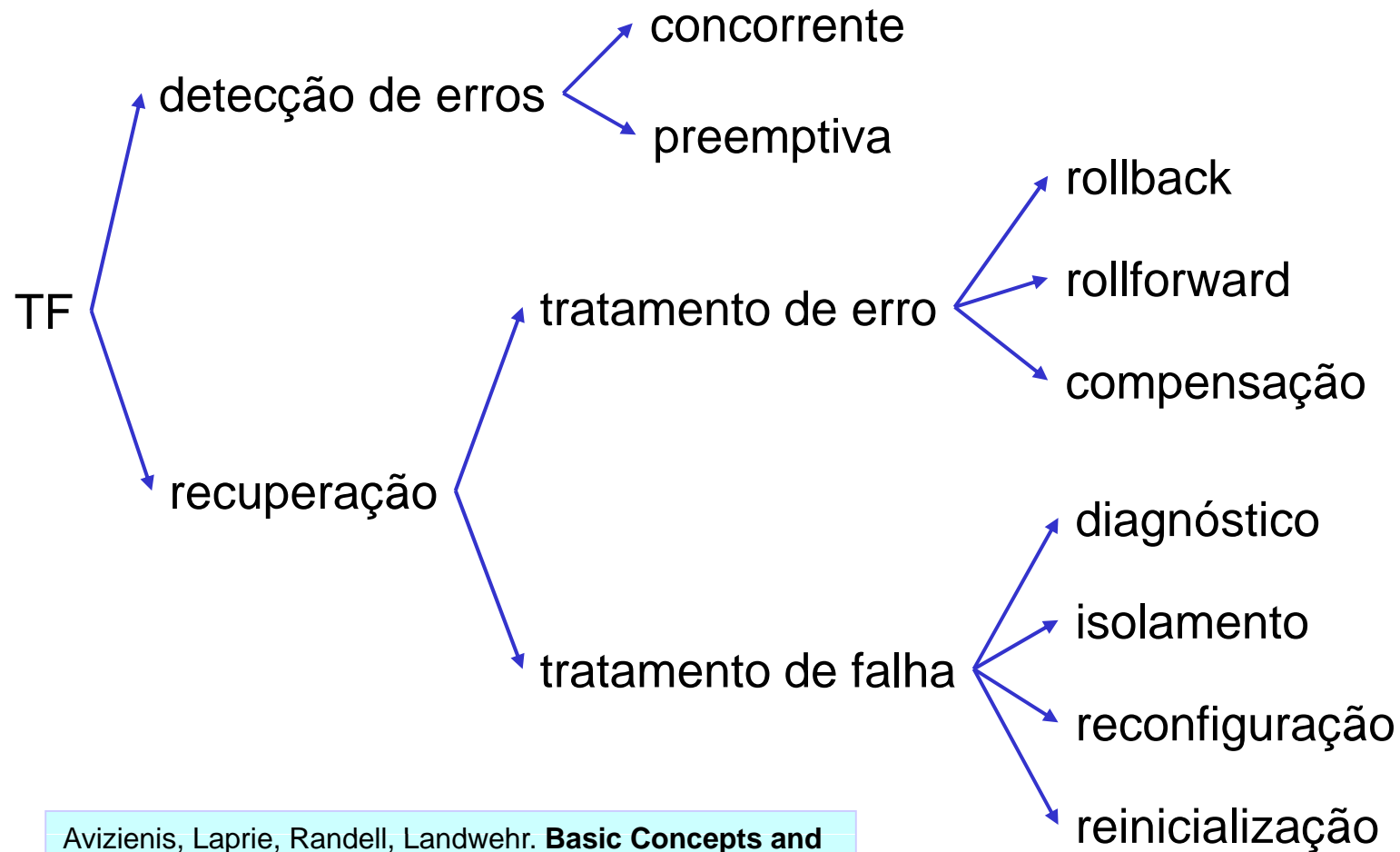
---

- prevenção e remoção de falhas não são suficientes:
  - quando o sistema exige alta confiabilidade,
  - ou alta disponibilidade
- técnicas de TF exigem
  - componentes adicionais
  - algoritmos especiais



# Técnicas de TF

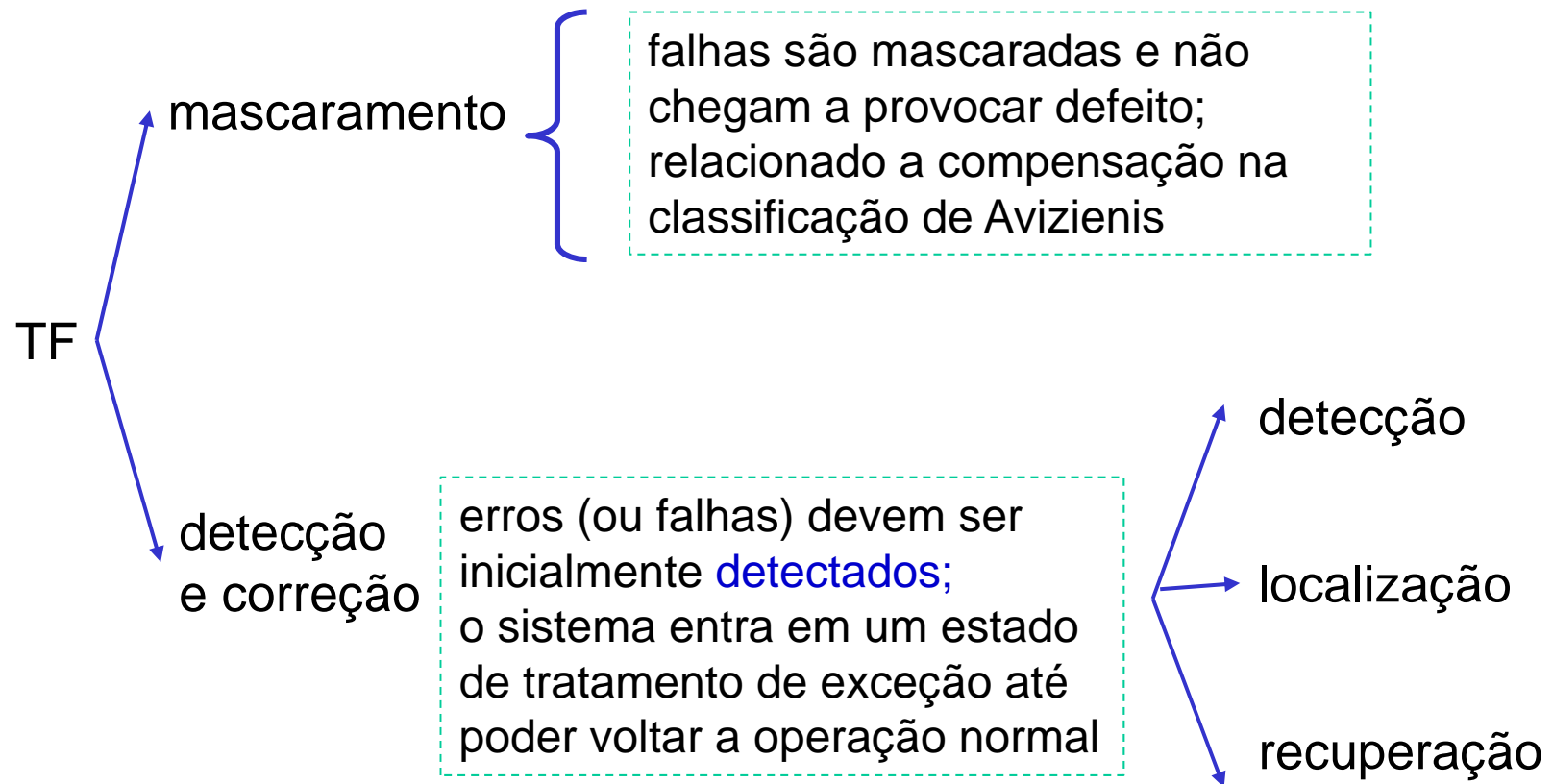
---



Avizienis, Laprie, Randell, Landwehr. **Basic Concepts and Taxonomy of Dependable and Secure Computing.** IEEE Trans. on dep. and secure comp. 2004

# Técnicas de TF : outra classificação

---



técnicas alternativas, mutuamente exclusivas



# mais classificações

---

- 4 fases (*Anderson & Lee*):

última fase

tratamento da falha

recuperação

confinamento e avaliação

primeira fase

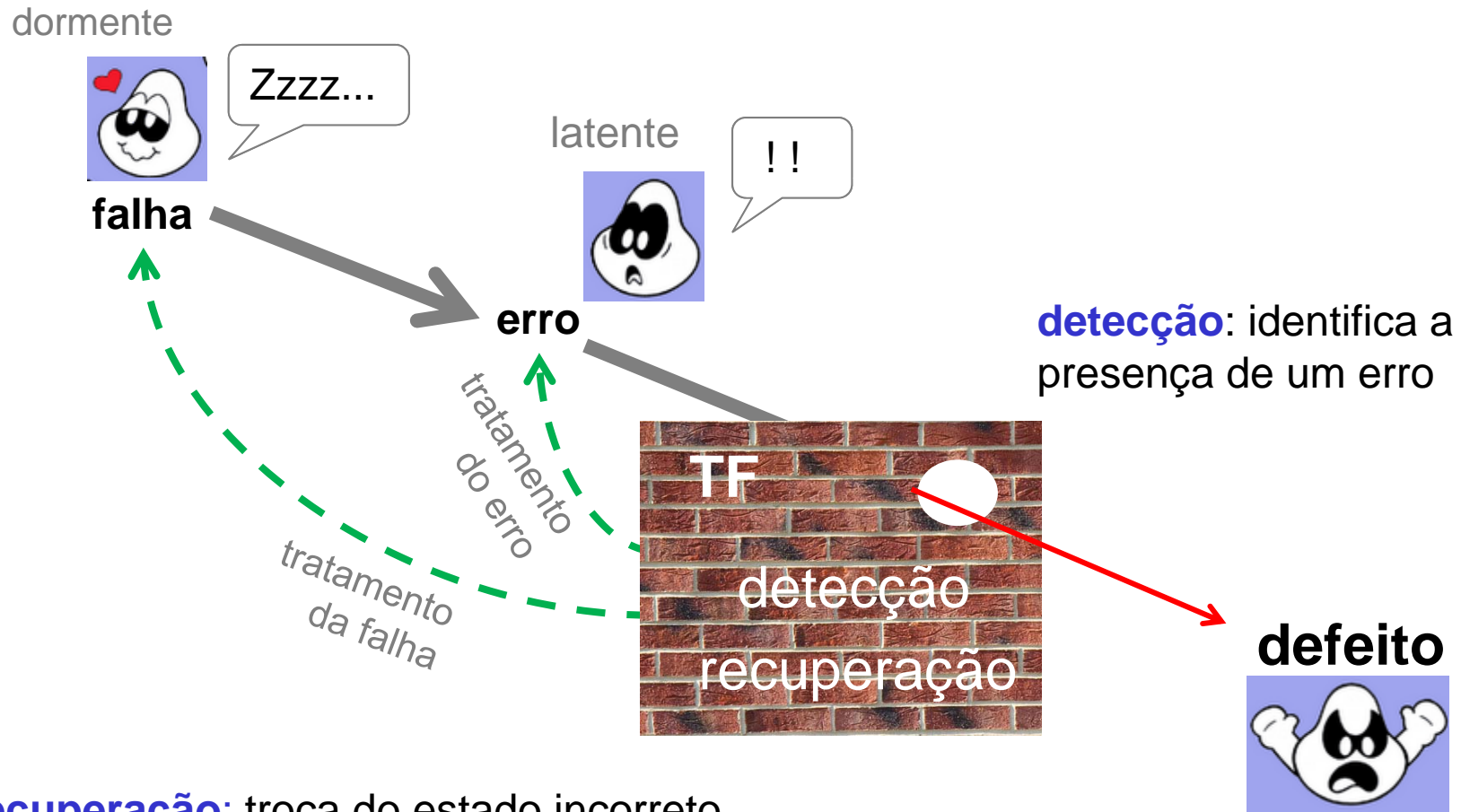
detecção

- múltiplas fases (*Nelson*)

- mascaramento, detecção, confinamento, diagnóstico, reparo, configuração, recuperação

Nelson, V. **Fault Tolerant Computing: Fundamental Concepts**. IEEE Computer, 1990

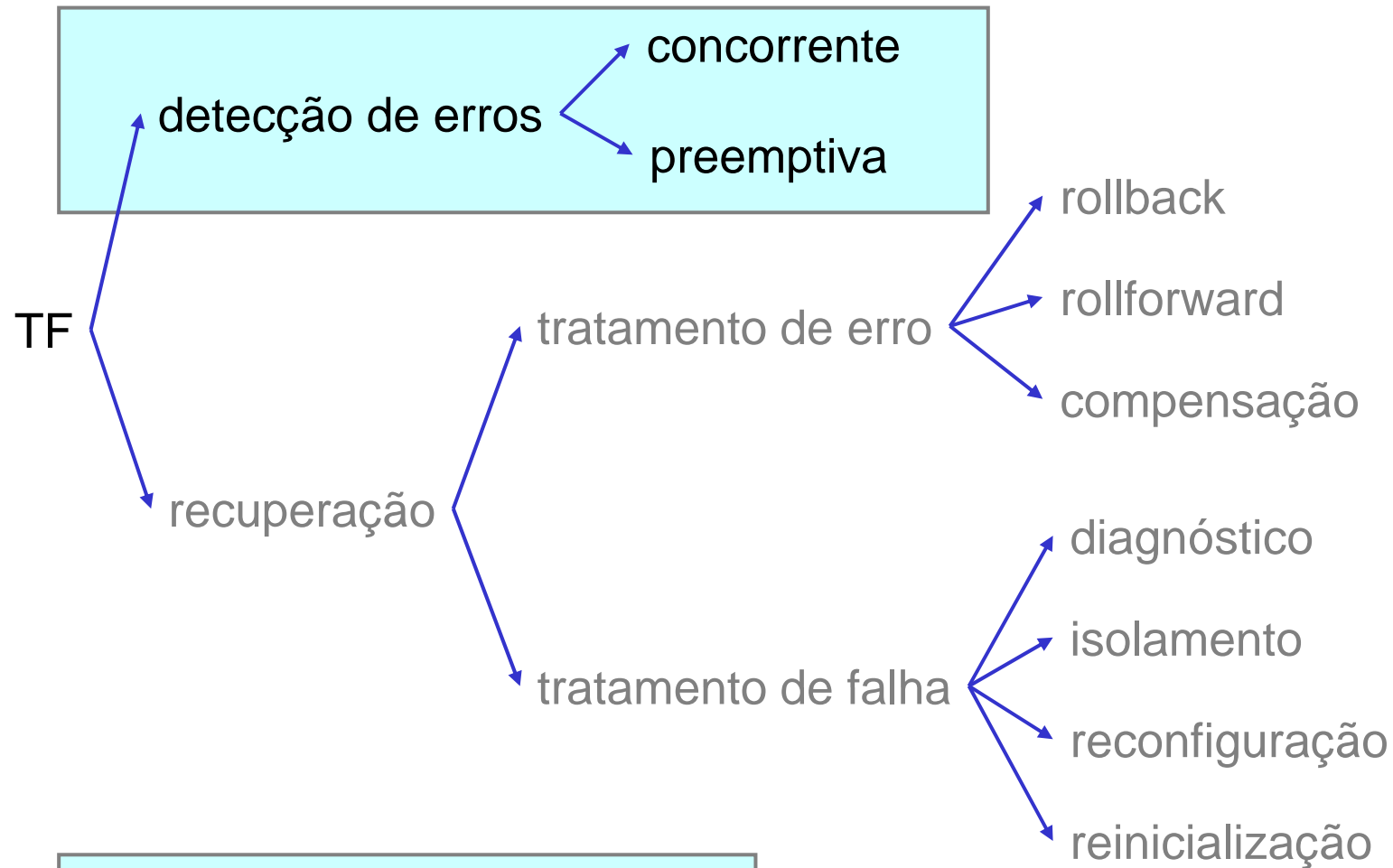
# Técnicas de TF



**recuperação:** troca do estado incorreto para um estado livre dos erros detectados e de falhas que possam ser ativadas

# Técnicas de TF: Avizienis

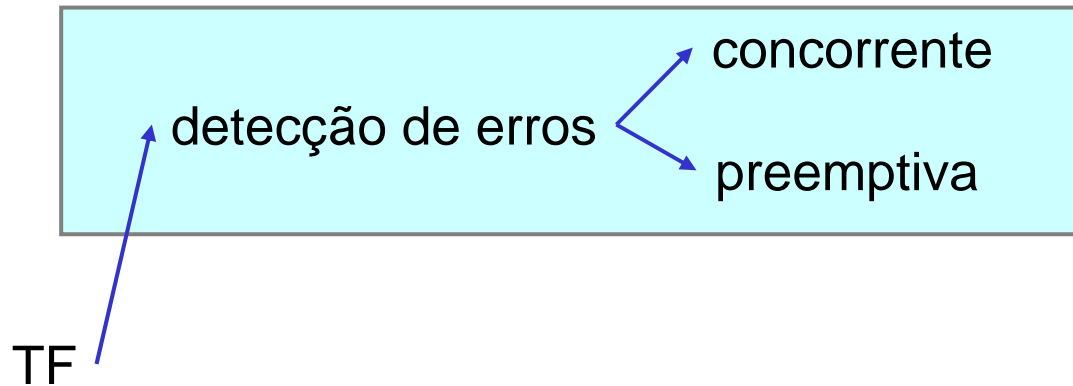
---



Avizienis, Laprie, Randell, Landwehr. **Basic Concepts and Taxonomy of Dependable and Secure Computing.** IEEE Trans. on dep. and secure comp. 2004

# Técnicas de TF: Avizienis

---



**concorrente**: durante período normal de operação

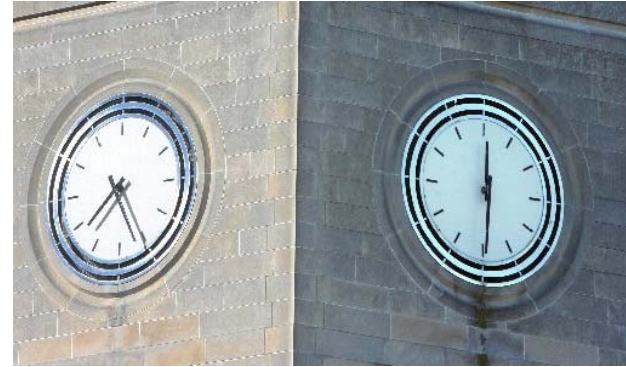
**preemptiva**: com serviço normal suspenso

Avizienis, Laprie, Randell, Landwehr. **Basic Concepts and Taxonomy of Dependable and Secure Computing**. IEEE Trans. on dep. and secure comp. 2004

# Técnicas de detecção de erros

---

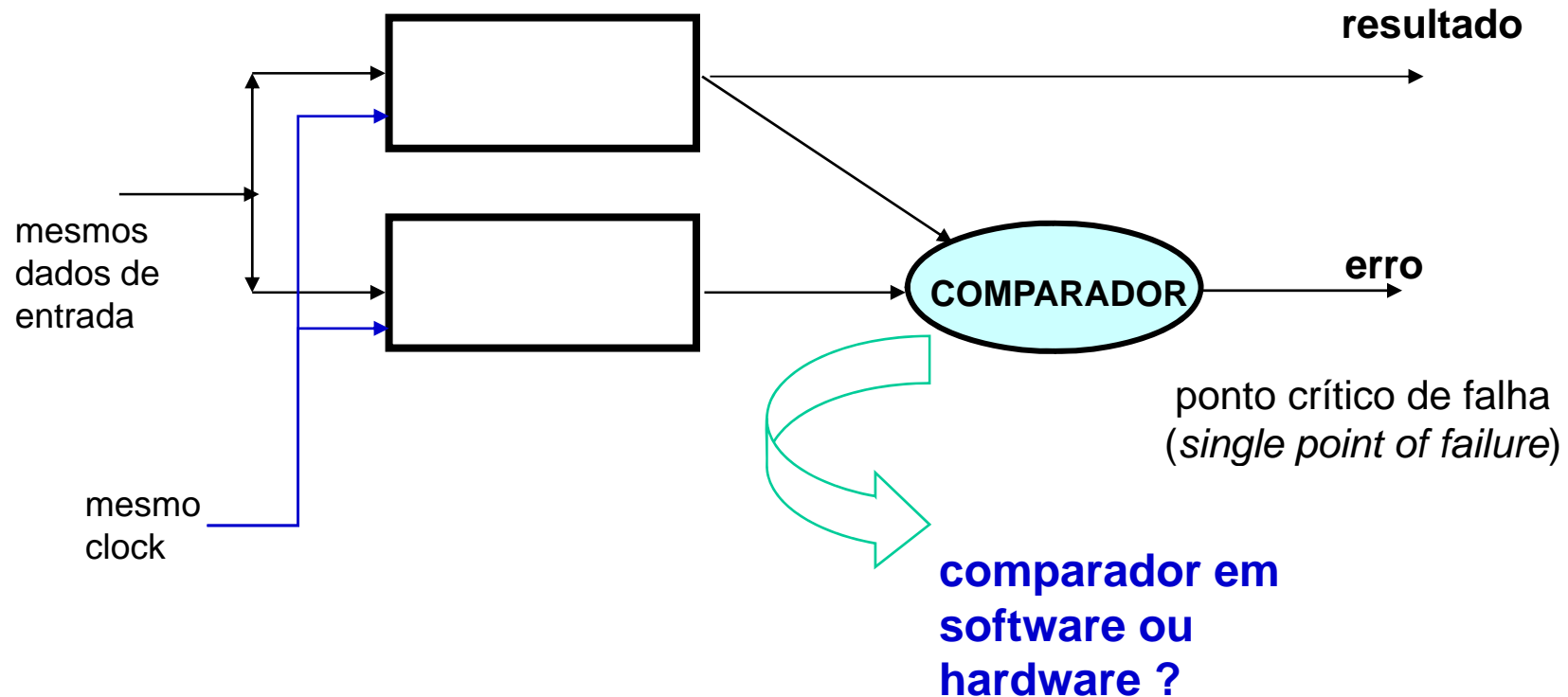
- duplicação e comparação
- codificação
- testes:
  - testes de limites de tempo
    - *time-out*, cão de guarda (*watchdog timers*)
  - testes reversos
  - teste de limites ou compatibilidade
  - testes de consistência
- diagnóstico
  - pode ser concorrente à operação normal ou preemptivo



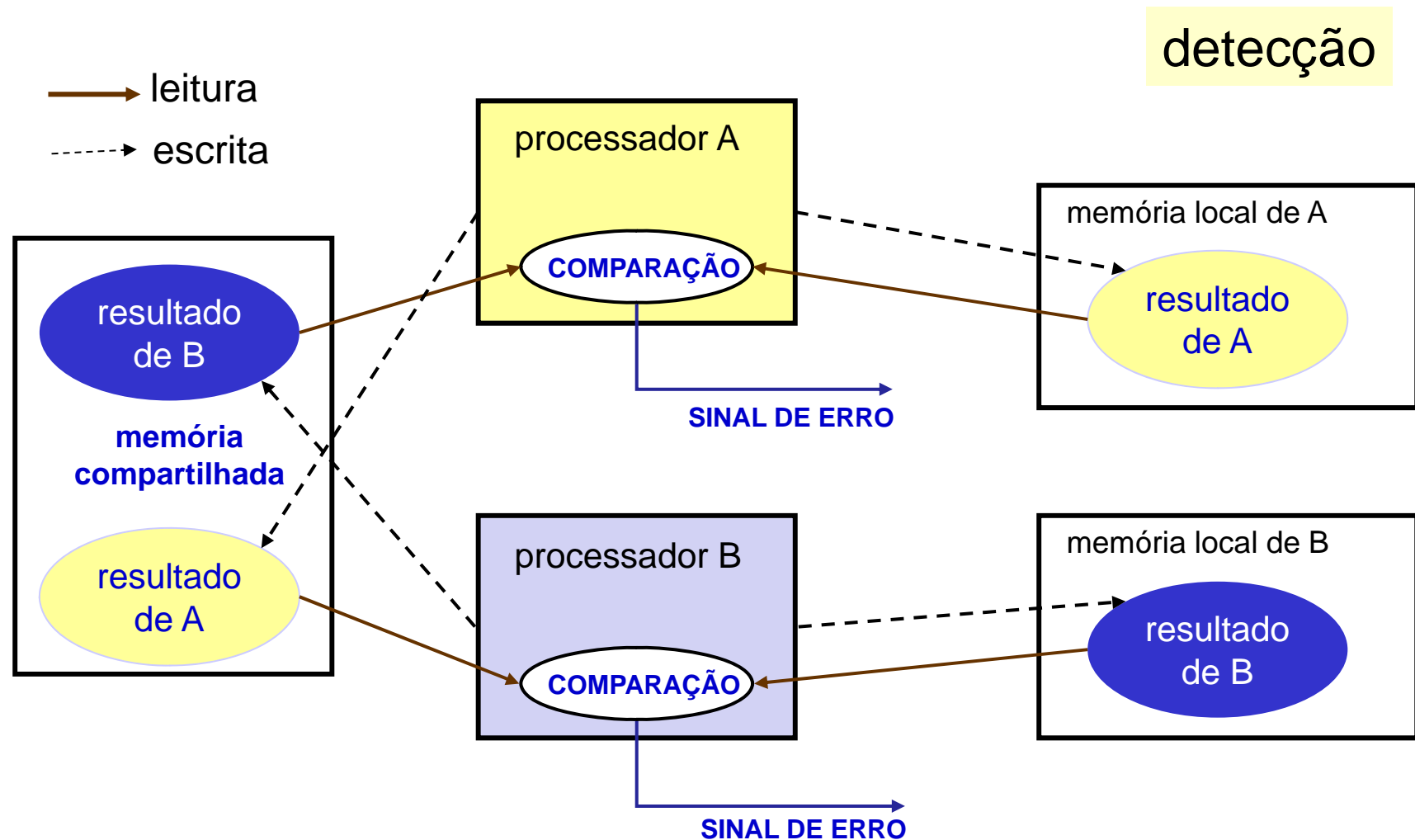
# Duplicação e comparação

detecção

2 módulos idênticos de hardware

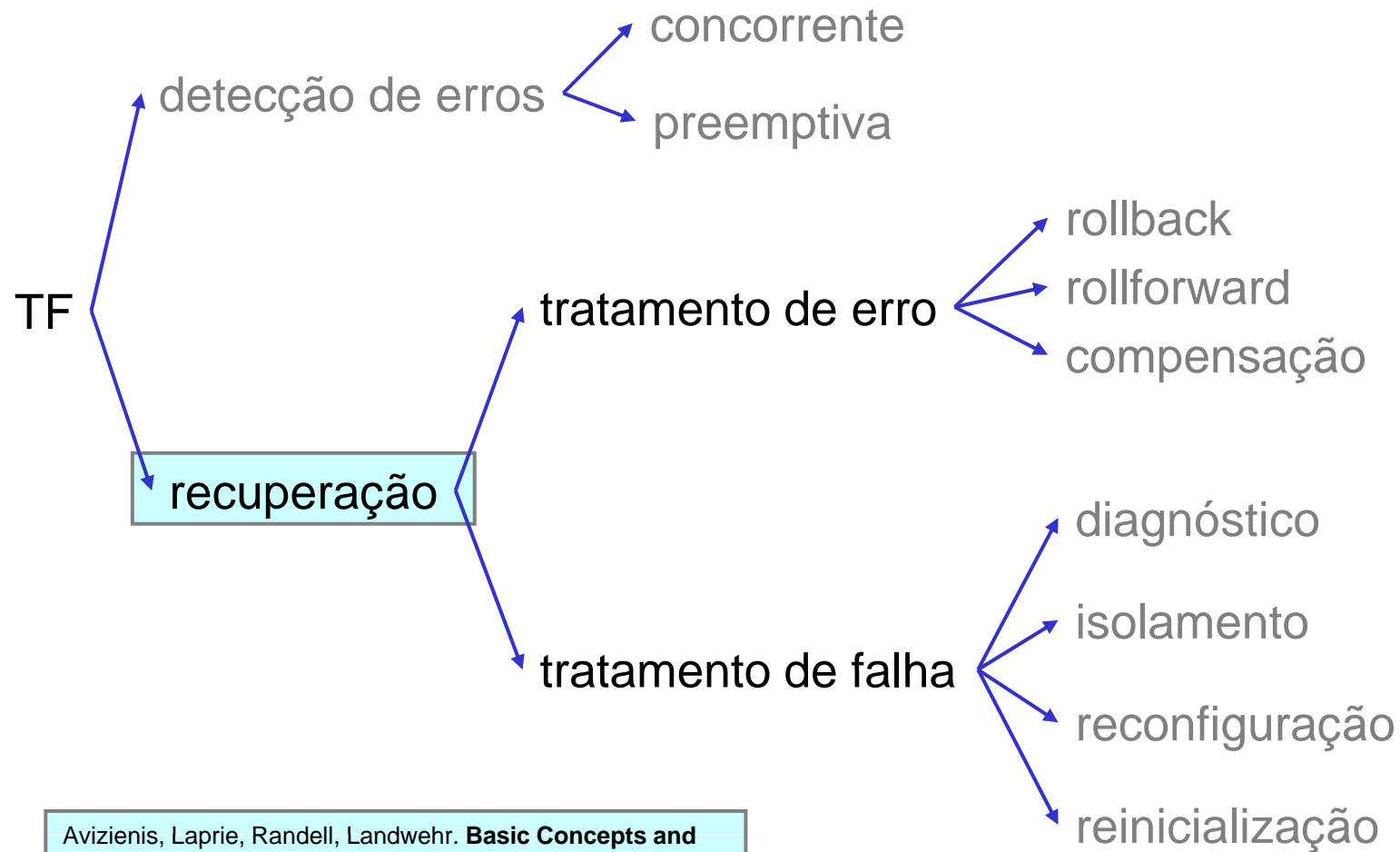


# Duplicação e comparação



# Técnicas de TF

---



Avizienis, Laprie, Randell, Landwehr. **Basic Concepts and Taxonomy of Dependable and Secure Computing.** IEEE Trans. on dep. and secure comp. 2004



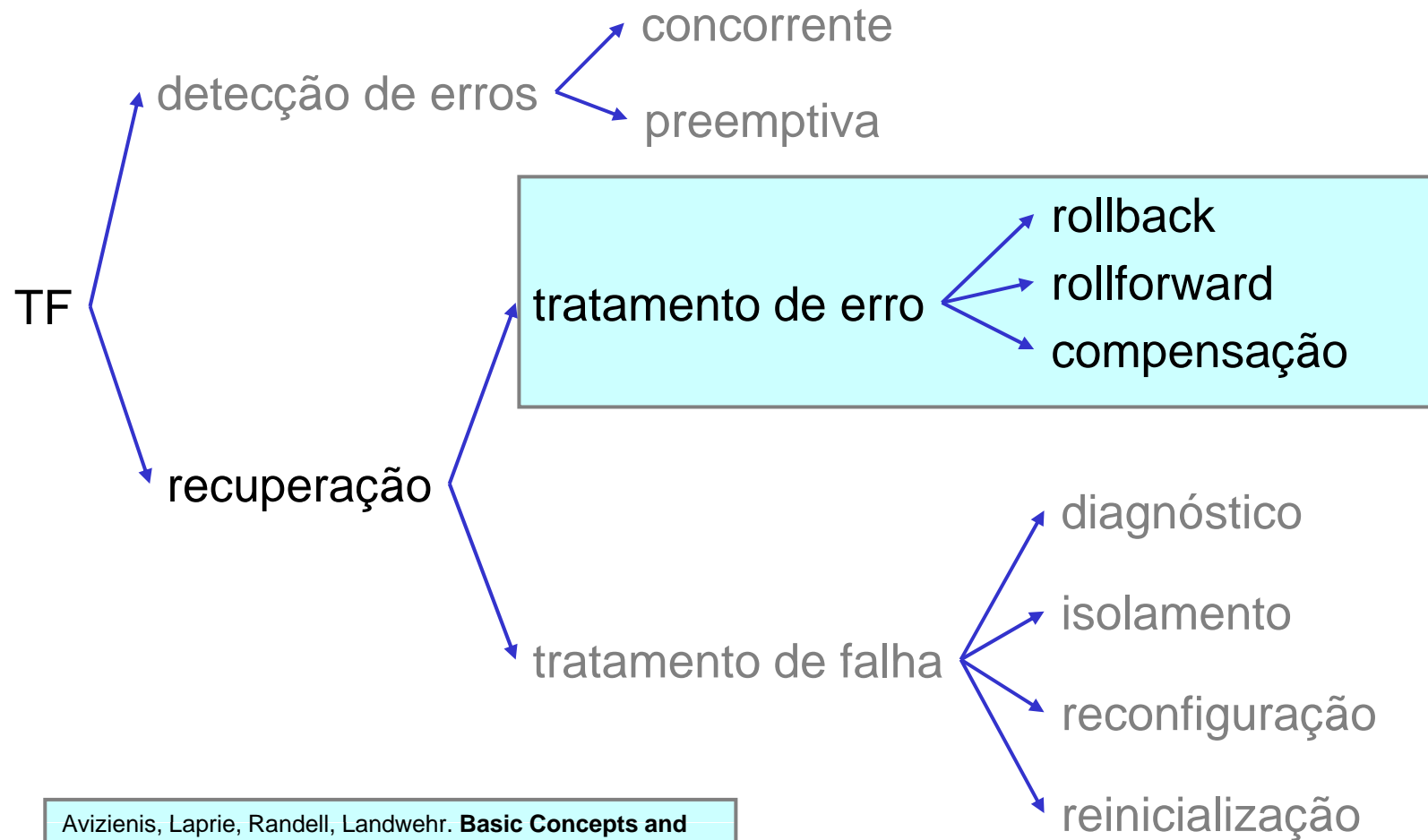
# Recuperação

---

- objetivo
  - troca do estado atual incorreto (que contém um ou mais erros e possivelmente falhas) para um estado livre dos erros detectados e de falhas que possam ser novamente ativadas
  - ocorre sempre após detecção
- tipos
  - tratamento de erros
    - elimina erros do estado do sistema
  - tratamento de falhas
    - previne reativação de falhas

# Técnicas de TF

---



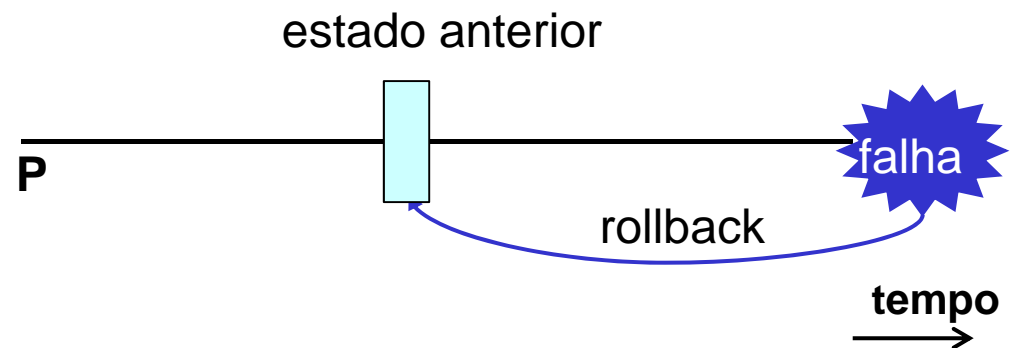
Avizienis, Laprie, Randell, Landwehr. **Basic Concepts and Taxonomy of Dependable and Secure Computing.** IEEE Trans. on dep. and secure comp. 2004

# Tratamento de erros

---

- recuperação por retorno

condução a **estado anterior**



- recuperação por avanço

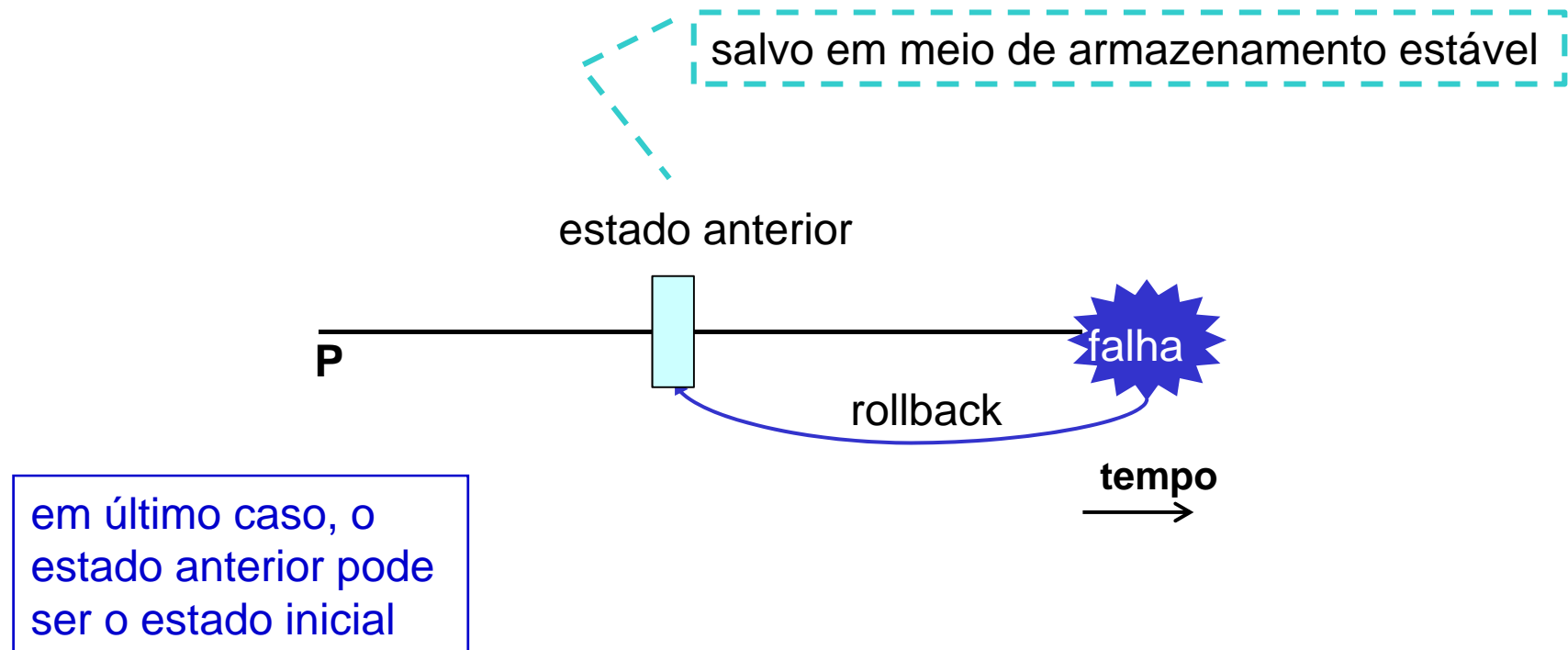
condução a **novo estado**



# Retorno

---

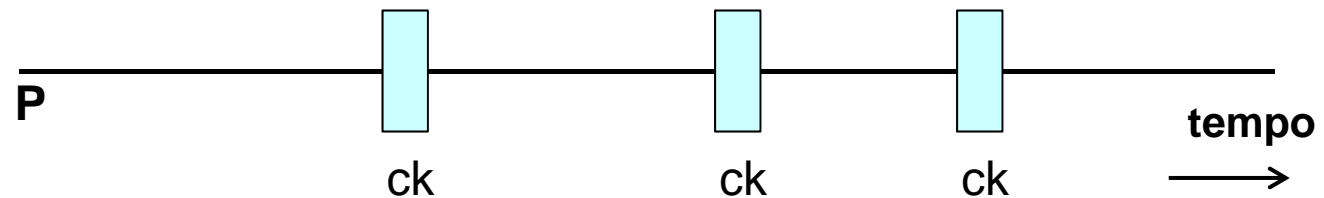
- condução a estado anterior consistente
  - implica no salvamento do estado anterior livre de erros
  - alto custo mas de aplicação genérica



# Retorno: exemplo

---

- salvamento de todo o estado do sistema periodicamente (**checkpoints**)



- simples em um único processo isolado
  - backup e log de operações
- complexa em processamento distribuído
  - sem restrições a comunicação pode provocar **efeito dominó**



# Avanço

---

- recuperação por avanço
  - condução a novo estado consistente ainda não ocorrido desde a última manifestação de erro
    - eficiente, mas específica a cada sistema
    - danos devem ser previstos acuradamente

- os dois tipos de recuperação (avanço e retorno) não são mutuamente excludentes



mais usada em sistemas de **tempo real**, onde o retorno para um estado anterior (no tempo) seja inviável

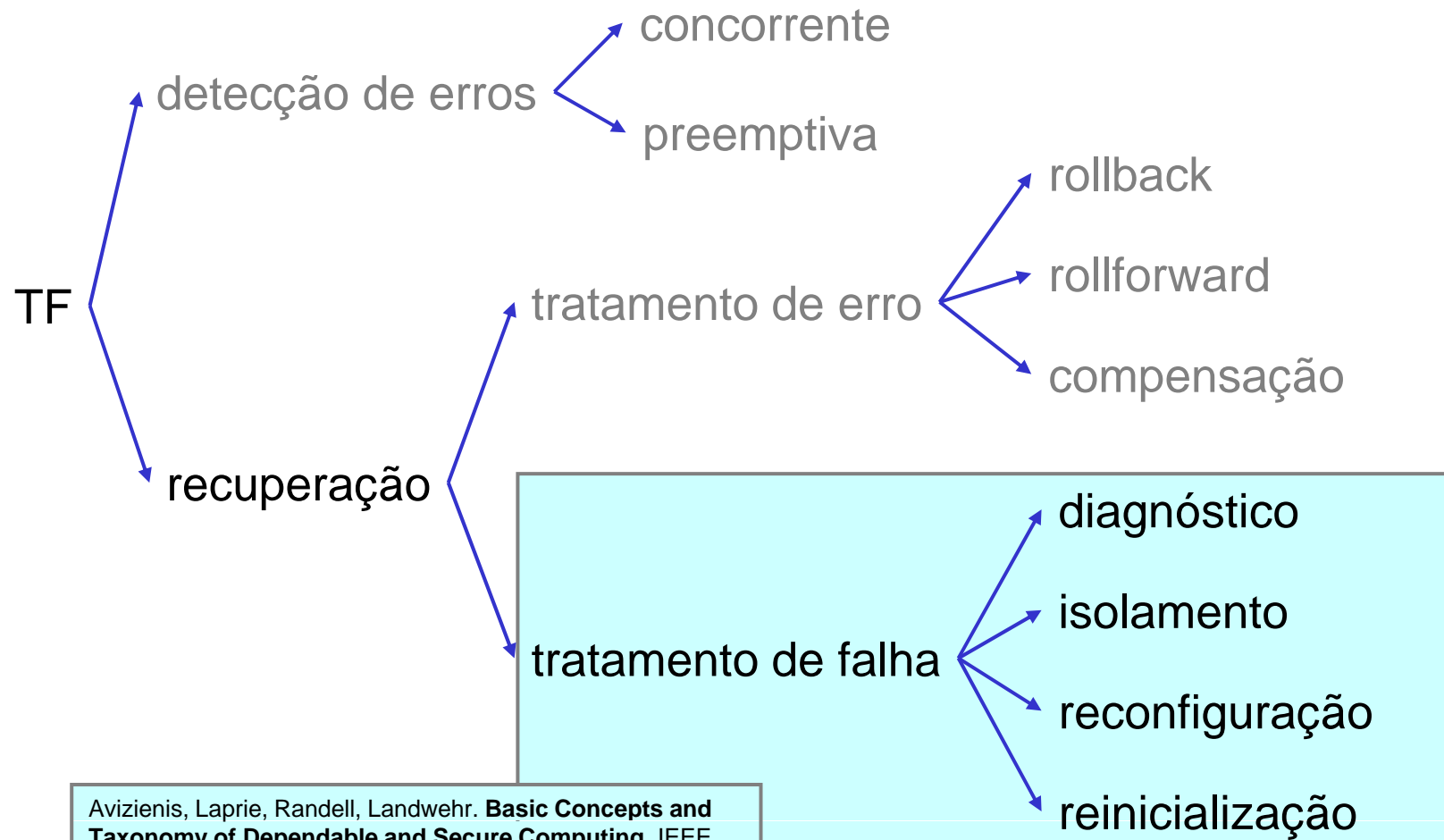
# Compensação

---

- o estado errôneo contém **redundância** suficiente para permitir que o erro seja mascarado
  - aplicação sistemática de compensação leva a mascaramento de falhas
    - mascaramento pode levar a perda progressiva da redundância de proteção (caso ocorram falhas permanentes e não envolva detecção de erros)
  - exemplos de mascaramento:
    - ECC
    - TMR (serão vistos posteriormente)

# Técnicas de TF

---



Avizienis, Laprie, Randell, Landwehr. **Basic Concepts and Taxonomy of Dependable and Secure Computing.** IEEE Trans. on dep. and secure comp. 2004

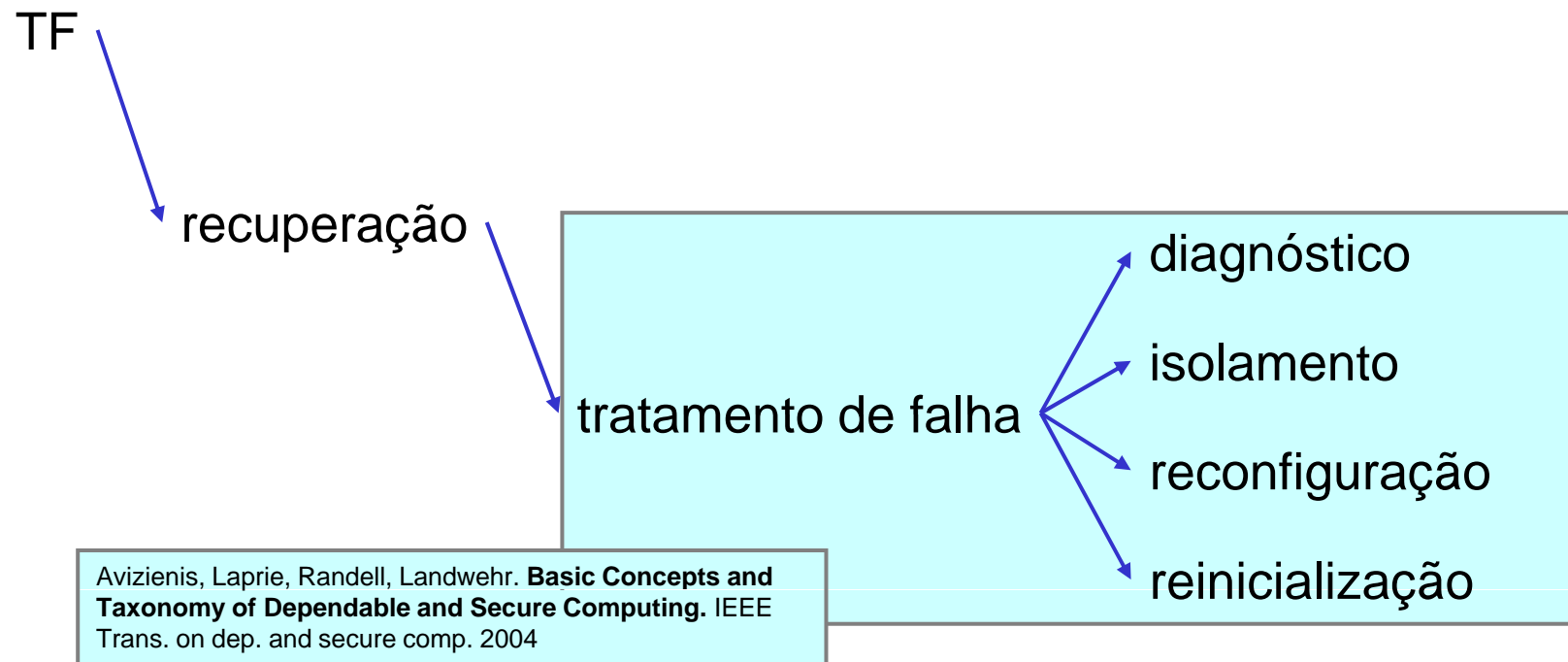


# Tratamento de falhas

---

tratamento da falha:

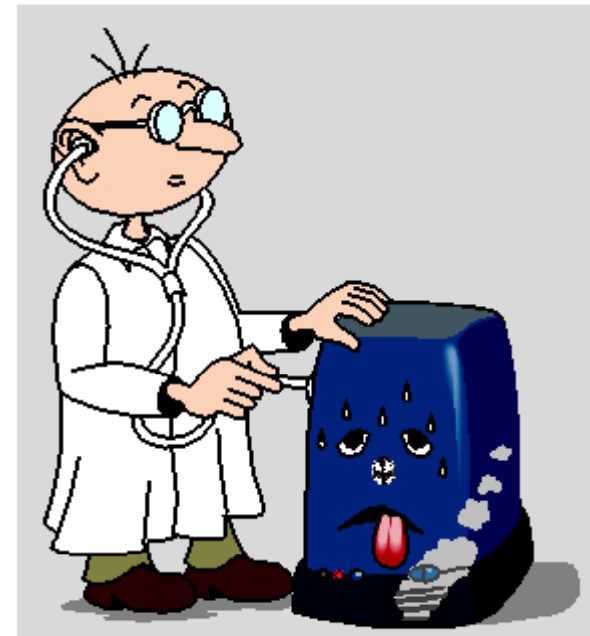
previne reativação de falhas;  
geralmente seguida de  
manutenção corretiva



# Tratamento de falhas

---

- diagnóstico
  - identifica a causa do erro e armazena informação de localização e tipo



# Isolamento

---

- confinamento

- latência de falha pode provocar espalhamento de dados inválidos
- o confinamento estabelece limites para a propagação do dano

depende de decisões de projeto do sistema

facilita isolamento

restringir fluxo de informações: evitar fluxos acidentais, estabelecer interfaces de verificação para detecção

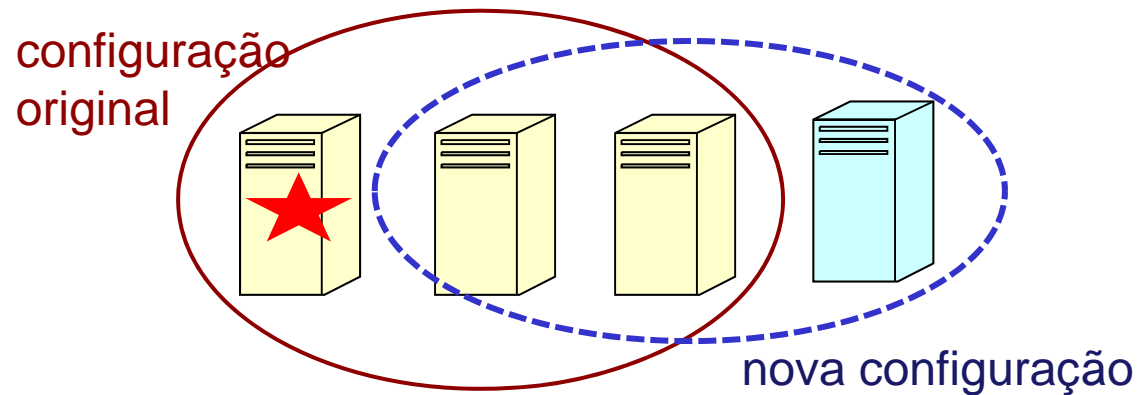
- isolamento

- exclui componente com falha do sistema
- o isolamento pode ser físico ou lógico
- falhas isoladas devem ser posteriormente removidas

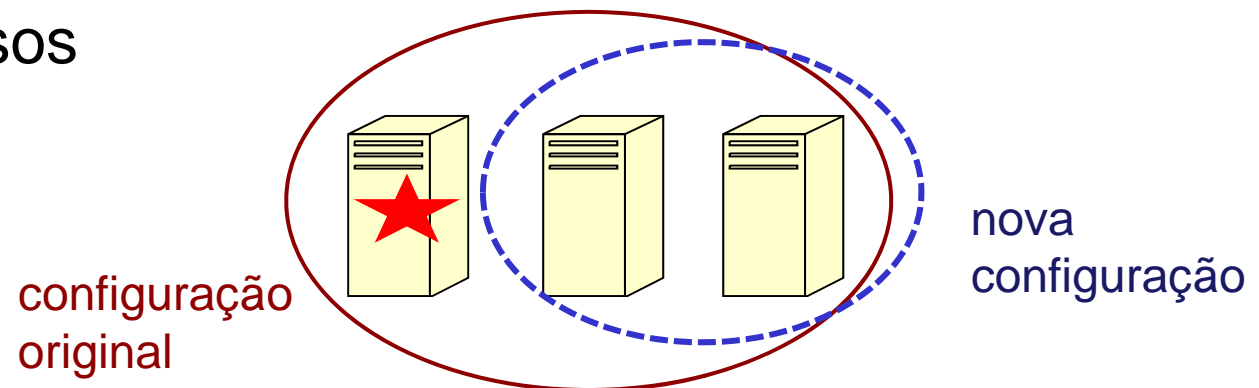
# Reconfiguração

---

- chaveia para componentes redundantes em espera



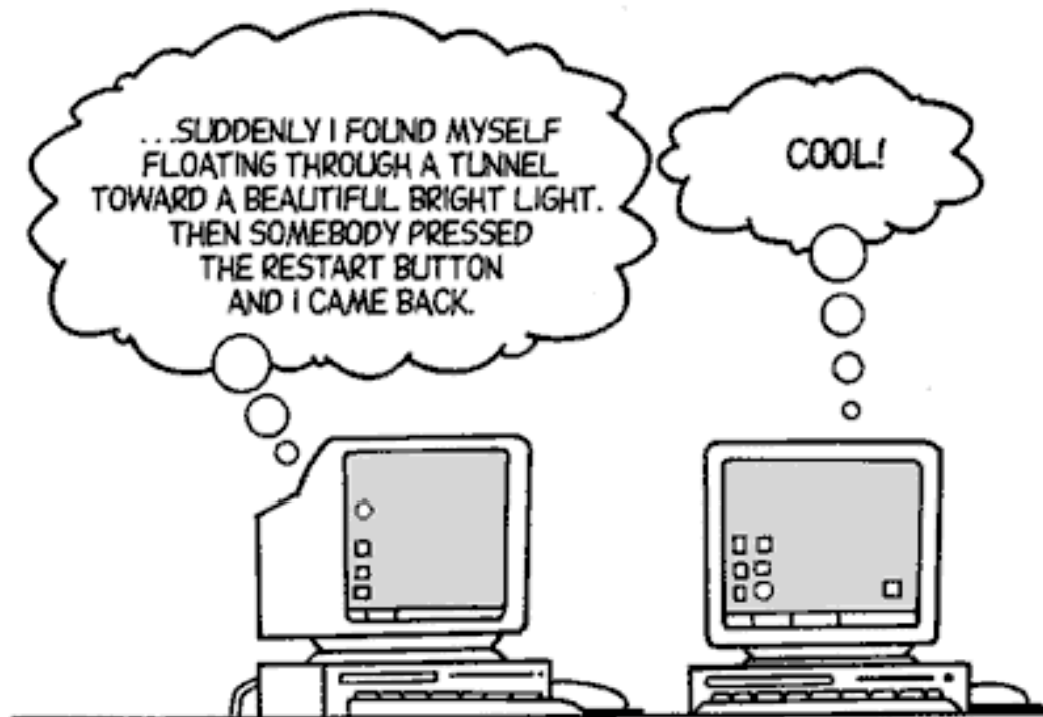
- redistribui tarefas entre componentes não defeituosos



# Reinicialização

---

- verifica, atualiza e guarda a nova configuração
- atualiza informações de configuração do sistema



© 1998 Randy Glasbergen. [www.glasbergen.com](http://www.glasbergen.com) E-mail: [randy@glasbergen.com](mailto:randy@glasbergen.com)

# TF vs manutenção

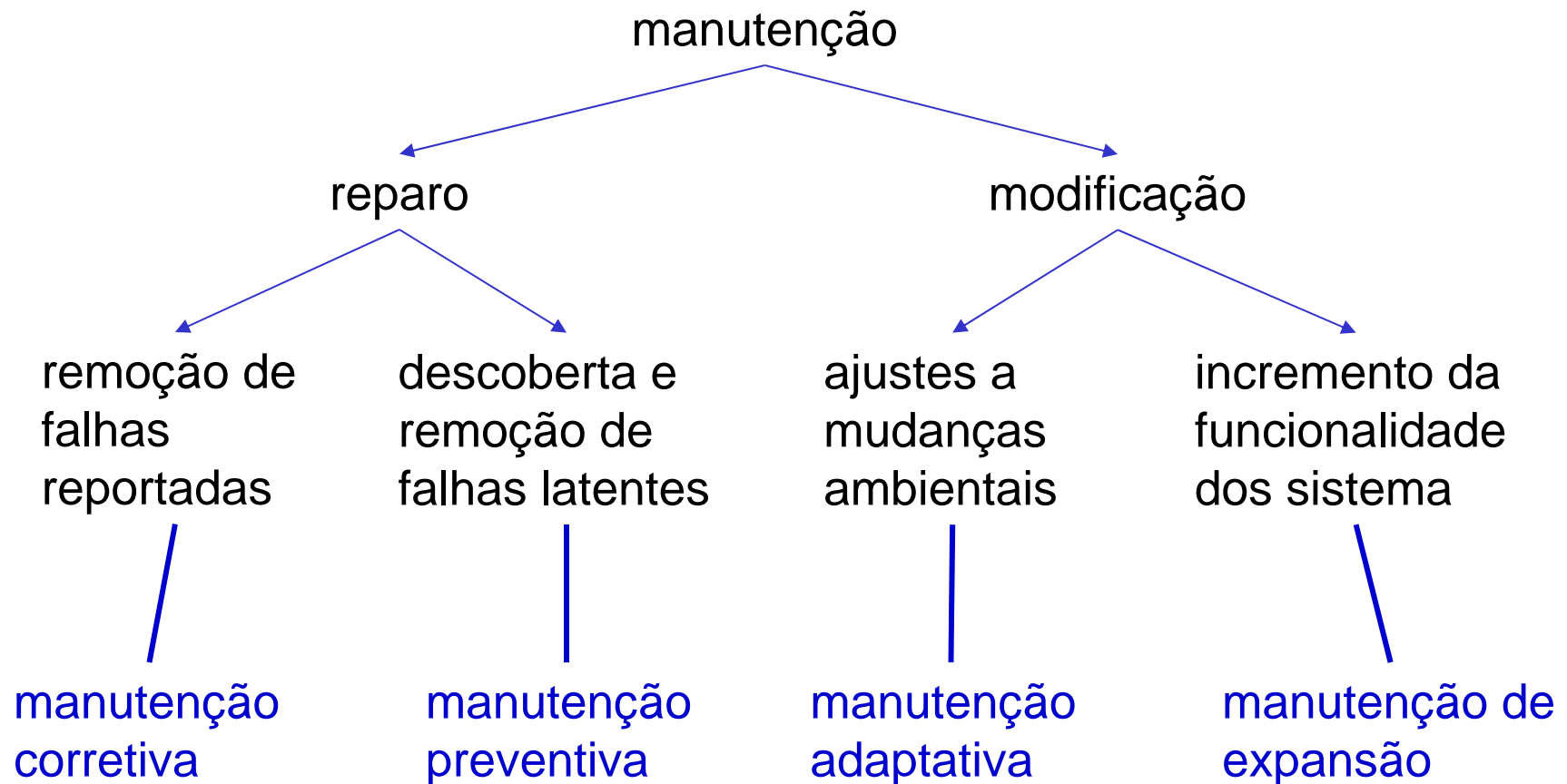
---

- reparo e TF são temas relacionados
  - manutenção envolve a ação de um agente externo
  - manutenção
    - reparo
    - modificação
  - reparo é uma atividade de remoção de falhas
    - remoção de falhas e tolerância a falhas são meios para alcançar dependabilidade

Avizienis, Laprie, Randell, Landwehr. **Basic Concepts and Taxonomy of Dependable and Secure Computing.** IEEE Trans. on dep. and secure comp. 2004

# Manutenção

---



# Bibliografia

---

- capítulo de livro
  - Johnson, Barry. An introduction to the design na analysis of the fault-tolerante systems, cap 1. **Fault-Tolerant System Design**. Prentice Hall, New Jersey, 1996
- artigos
  - Avizienis, Laprie, Randell, Landwehr. **Basic Concepts and Taxonomy of Dependable and Secure Computing**. IEEE Trans. on dep. and secure comp. 2004
  - Nelson, V. – Fault Tolerant Computing: Fundamental Concepts. IEEE Computer. 1990



# Fundamentos de TF

## Redundância

---

*Taisy Silva Weber*  
*UFRGS*

# Redundância

---

- redundância
  - de hardware
  - de software
  - de informação
  - no tempo
- impacto no sistema (custo):
  - desempenho, ou área (tamanho, peso), ou potência consumida, ...

toda redundância tem custo

# Redundância

---

- de hardware { passiva  
ativa  
híbrida
- de software
- de informação
- no tempo

Johnson, B.W. *Fault Tolerance*, The Electrical Engineering Handbook, Ed. Richard C. Dorf, Boca Raton: CRC Press LLC, 2000

# Redundância de hardware

---

- redundância **passiva** (ou **estática**)

- marcar falhas
- não requer ação do sistema
- não indica falha

- redundância **ativa** (ou **dinâmica**)

envolve detecção e recuperação

- redundância **híbrida**

- combinação das anteriores
  - mascaramento e longa vida
  - geralmente de alto custo

# TMR

---

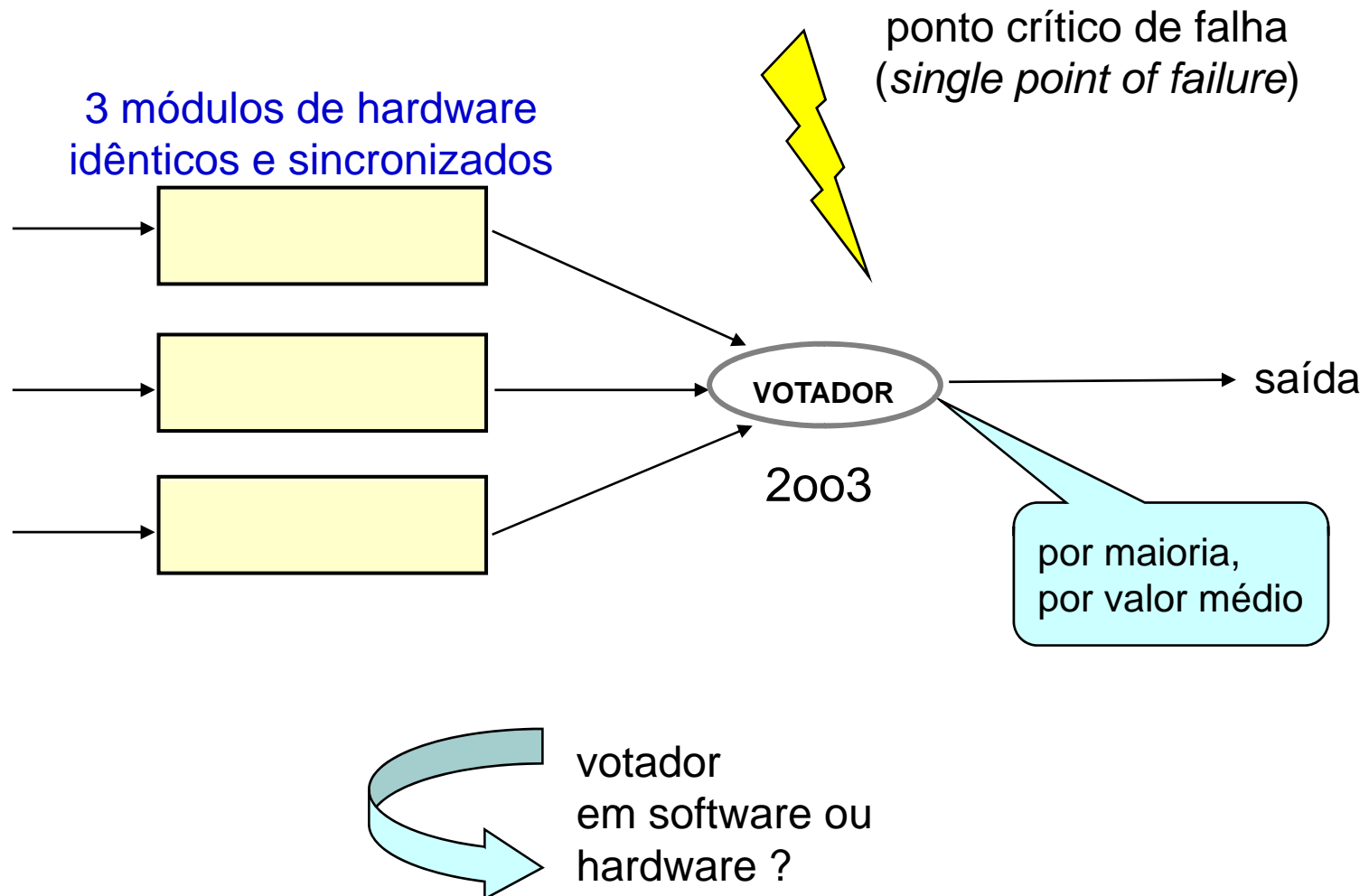
- redundância modular tripla
  - redundância de hardware **passiva**
    - mascara falha em **um** componente
  - votação
    - votação por maioria (2 em 1)
    - votação por seleção do valor médio
      - sinal selecionado reside no meio dos outros dois



mais usada em sistemas críticos,  
mas aparece também em sistema de alta  
disponibilidade como alguns servidores de rede

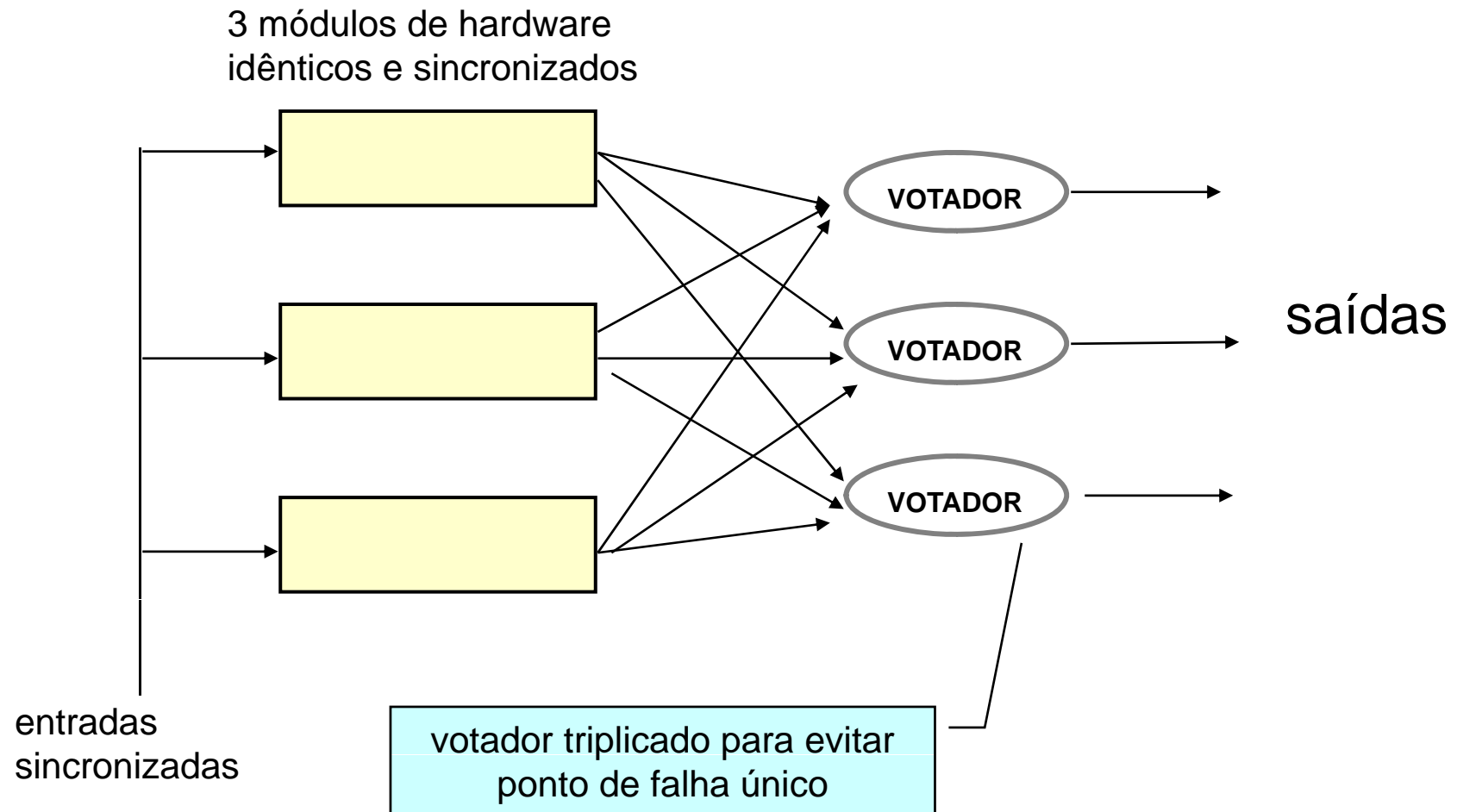
# TMR: redundância modular tripla

---



# TMR com 3 votadores

---



# Confiabilidade de TMR

---

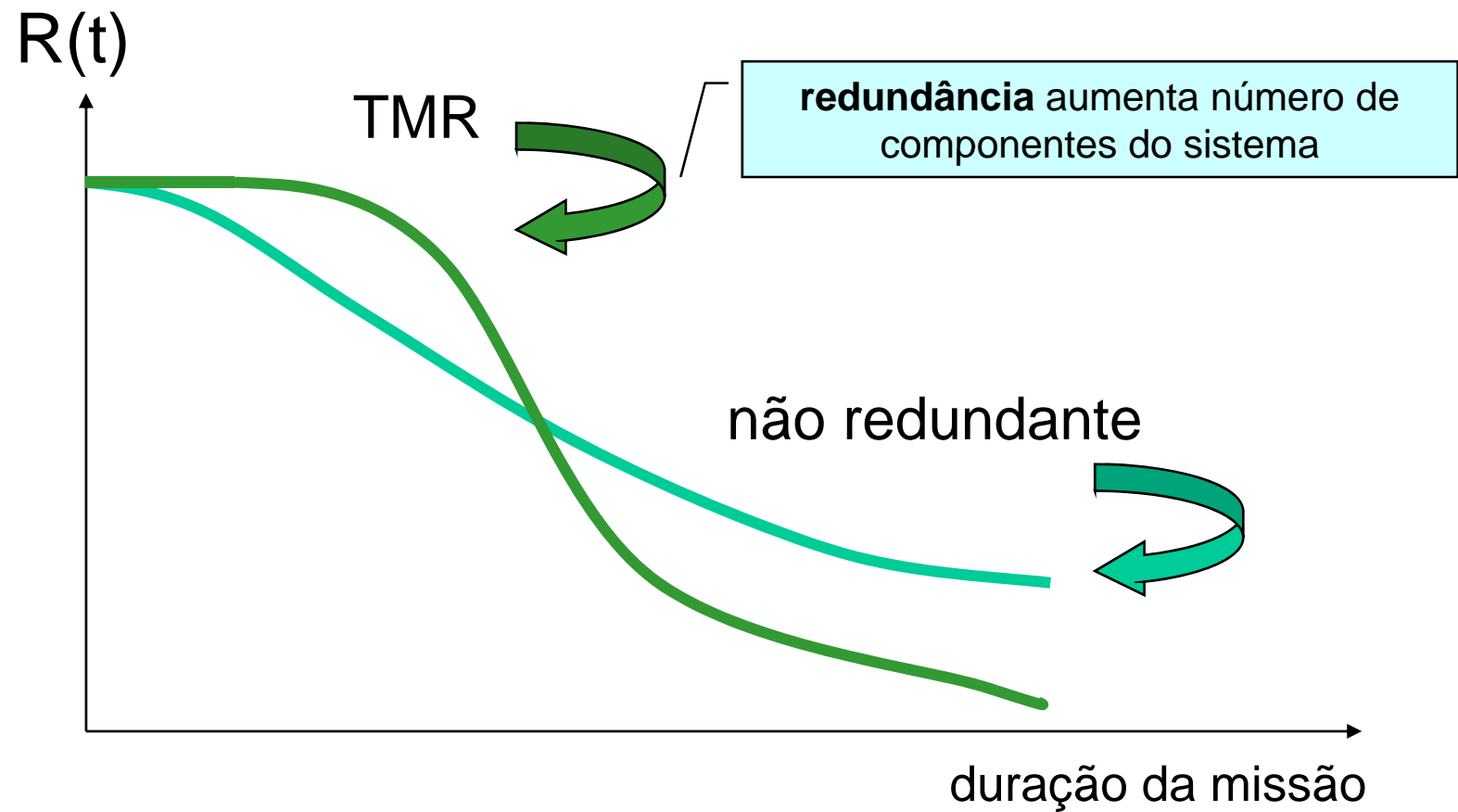
- redundância aumenta número de componentes do sistema
  - maior número de componentes, maior probabilidade de falha
- ideal para períodos não muito longos de missão
  - suporta falhas permanentes em um dos componentes (exceto votador) mas é ideal para falhas temporárias

pergunta: faz sentido usar TMR para detectar erros de software (triplicando um processo)?



# Confiabilidade de TMR

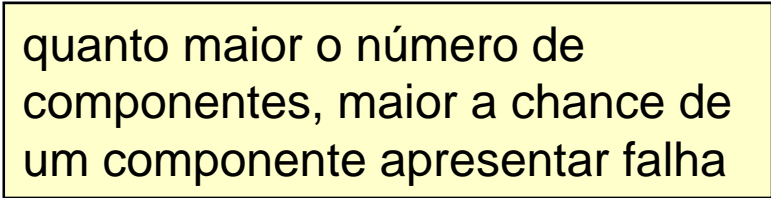
---



# NMR

---

- redundância modular múltipla - NMR
  - são usados N elementos
    - TMR = caso especial de NMR
  - idealmente número ímpar de componentes
    - N é ímpar para evitar empate
  - dificuldade:
    - determinação do N
    - custo, área, potência e até confiabilidade dos componentes



quanto maior o número de componentes, maior a chance de um componente apresentar falha

# Redundância dinâmica (ou ativa)

---

- detecção e recuperação

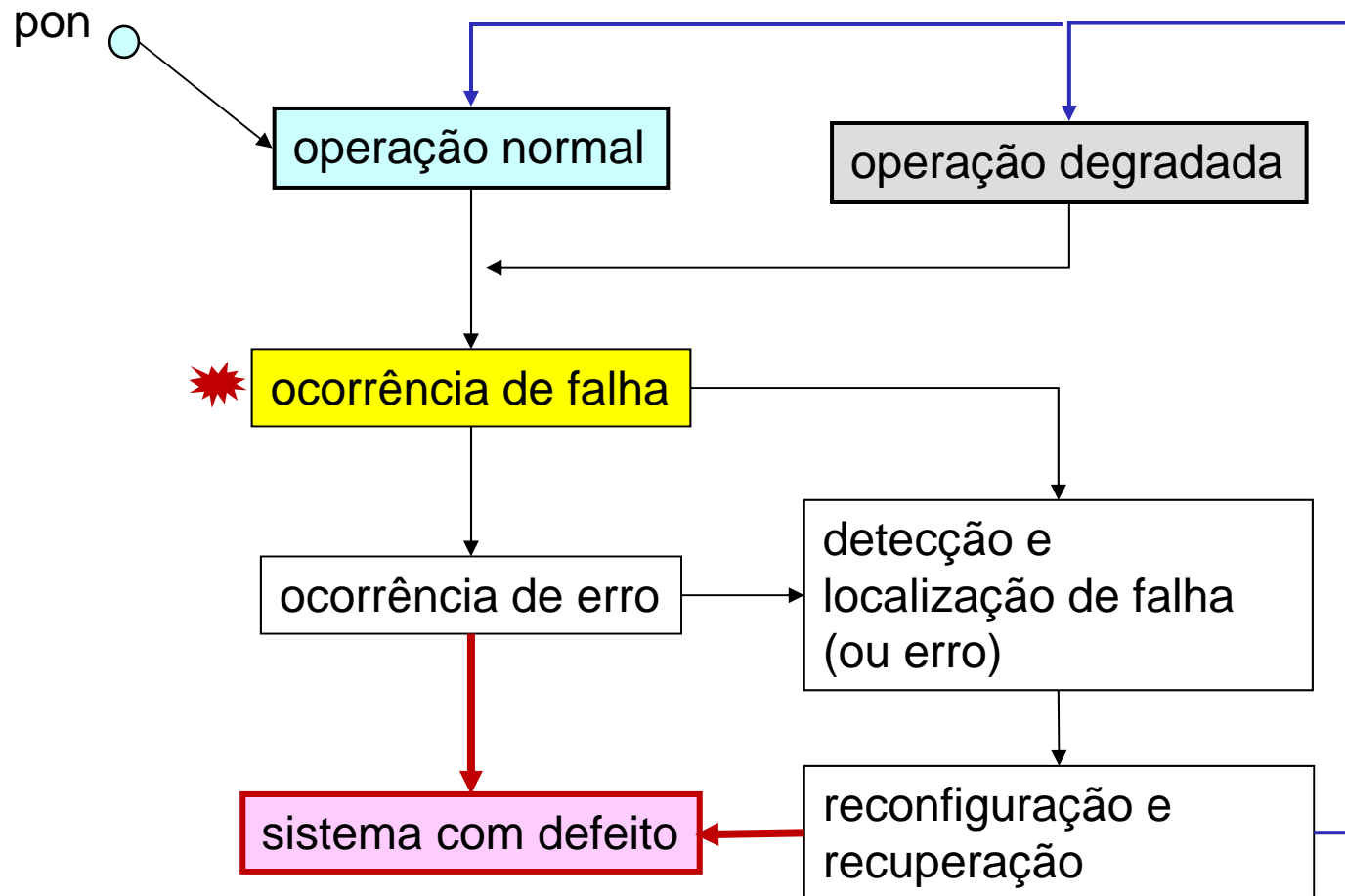
- sem mascaramento

detectar e recuperar leva um certo tempo

- aplicações

- sistemas que toleram resultados errados durante um curto período de tempo
  - exemplos:
    - manutenção adiada (satélites)
    - alta disponibilidade

# Estados na redundância ativa



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

# Exemplos de redundância dinâmica

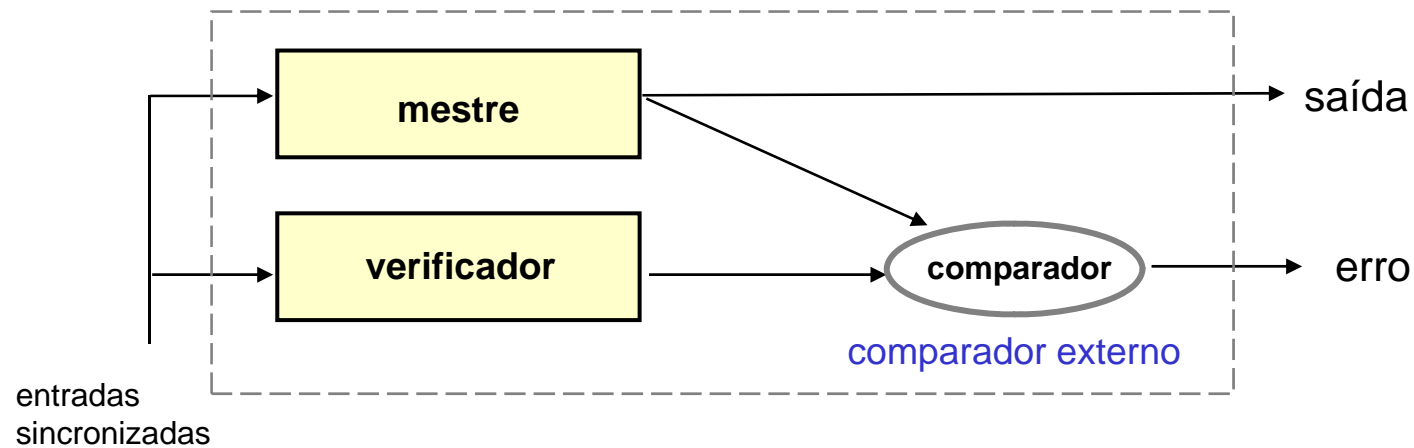
---

- *standby sparing*
  - módulo operacional vs. módulos estepe
- 2 tipos: 

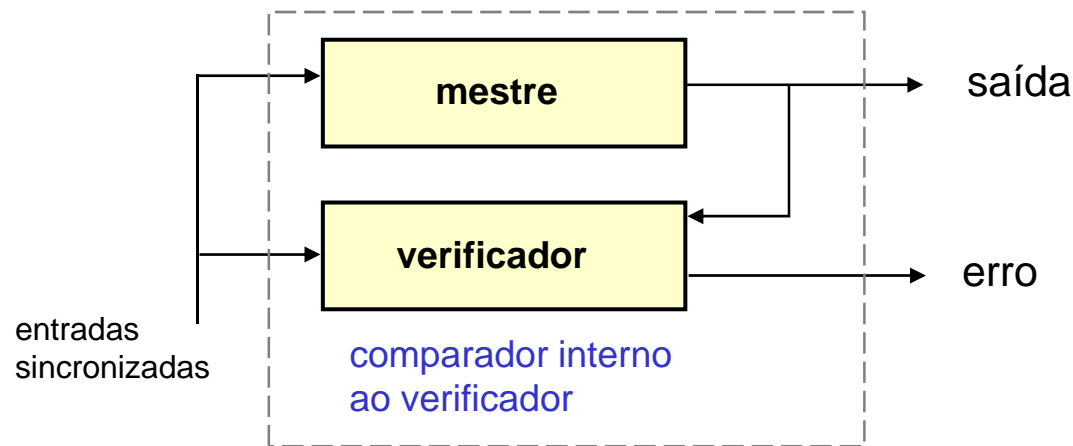
alimentados = ligados (com energia elétrica)

  - alimentados (*hot standby*)
    - minimiza a descontinuidade do processamento durante chaveamento entre os módulos
    - todos os componentes sofrem envelhecimento
  - não alimentados (*cold standby*)
    - aumenta a vida útil dos módulos
    - estepes (backups) só começam a operar após conectados e inicializados

# Componentes auto verificadores



Nelson, V. – Fault Tolerant Computing: Fundamental Concepts, IEEE Computer, 1990, pp 19-25

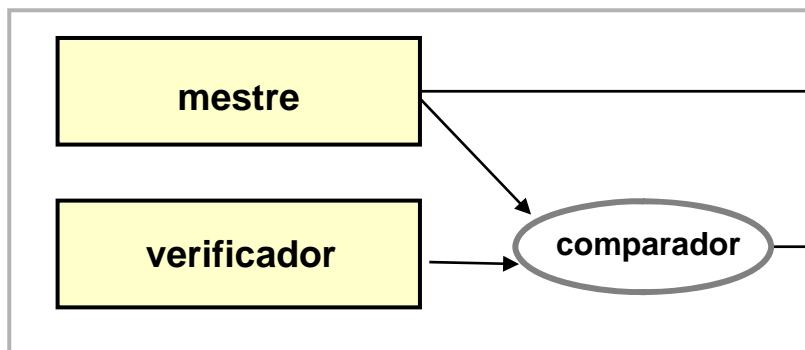


signal de erro  
permite  
implementar  
redundância ativa

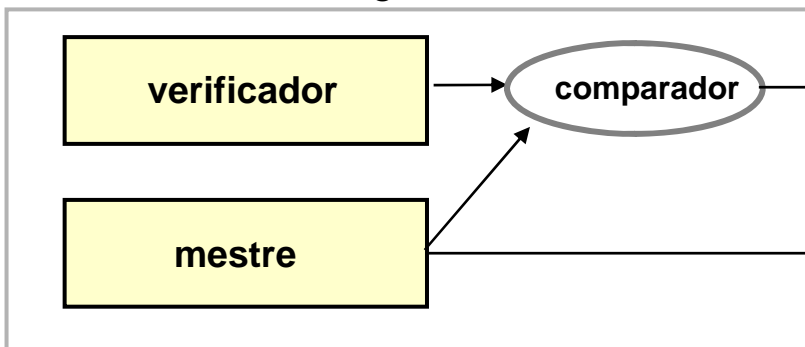
# Operação contínua

com componentes auto verificadores

módulo self-cheking A



módulo self-cheking B



saída

erro

chave

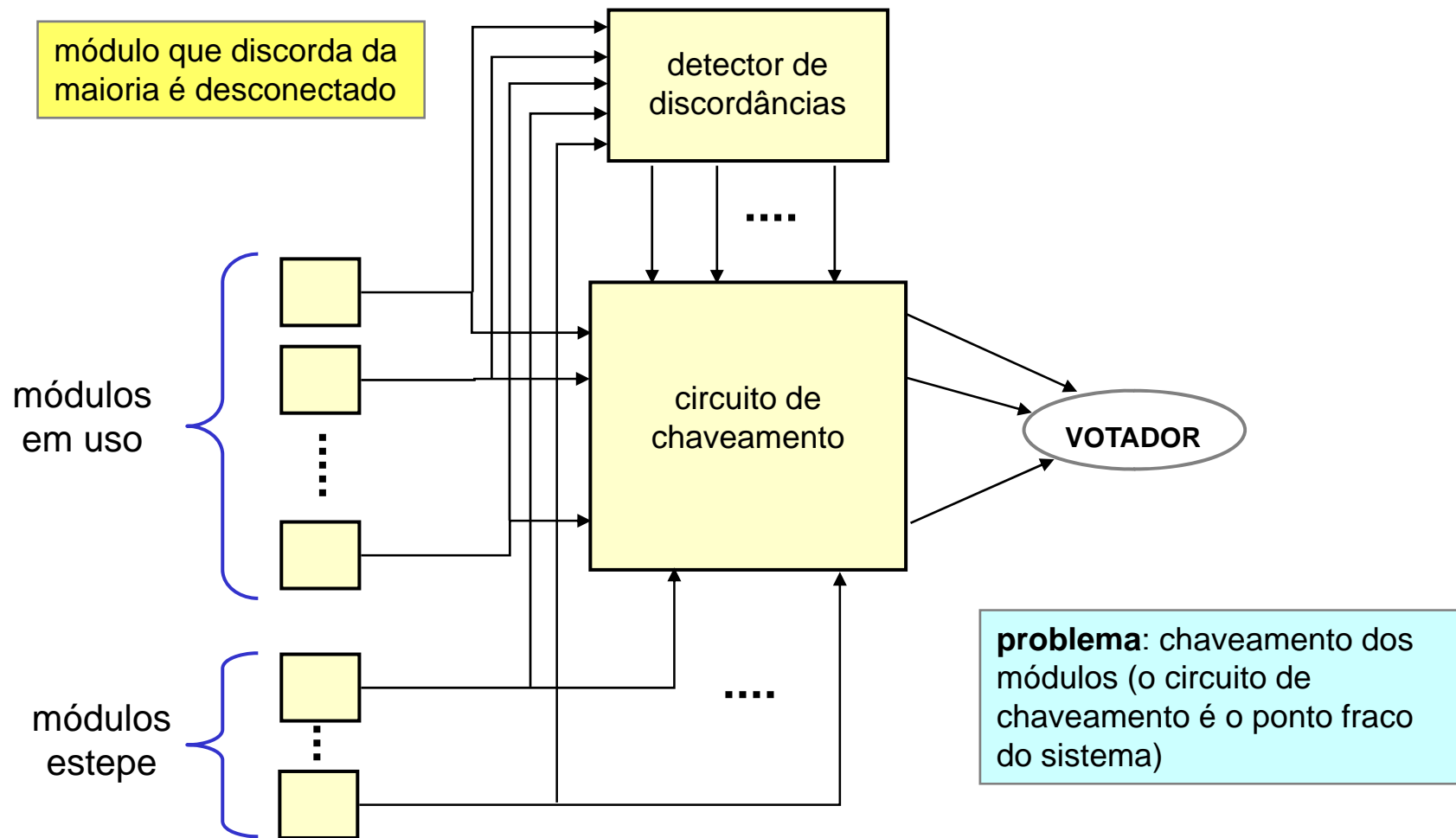
saída  
contínua

erro

saída

Nelson, V. – Fault Tolerant Computing:  
Fundamental Concepts, IEEE Computer,  
1990, pp 19-25

# Redundância híbrida



estepe substitui módulo com erro



# Redundância auto-eliminadora

---


- caso especial de redundância híbrida
  - também baseada em votação
  - mas módulo que discorda da maioria é isolado
  - não existe estepe
    - todos os módulos estão desde o início da operação conectados ao sistema

todos os módulos sofrem igualmente os efeitos do envelhecimento

# Redundância

---

- de hardware
- de software
- de informação
- no tempo



também são formas  
comuns de redundância

**sem** componentes físicos replicados  
(menor número de componentes, menor a  
probabilidade de defeito devido a falhas  
em um componente)

# Redundância de software

---

para **tolerar erros** de software

replicação de componentes de software idênticos  
é útil apenas para falhas evasivas

- exemplos:

- diversidade
  - de versões de programas
  - de projeto
  - de especificação
  - de implementação
- blocos de recuperação
- verificação de consistência


programação n-versões

programação  
diversitária

# Programação n-versões

---

- versões diferentes de um mesmo programa
  - votação na saída
  - erros devem se manifestar de forma diferente nas versões



correlação não permitida

- vantagens
  - facilidade de reconhecer erros na fase de teste
  - tolerância a falhas permanentes e intermitentes
  - tolerância a falhas externas
    - do compilador e do sistema operacional

# Desvantagens

---

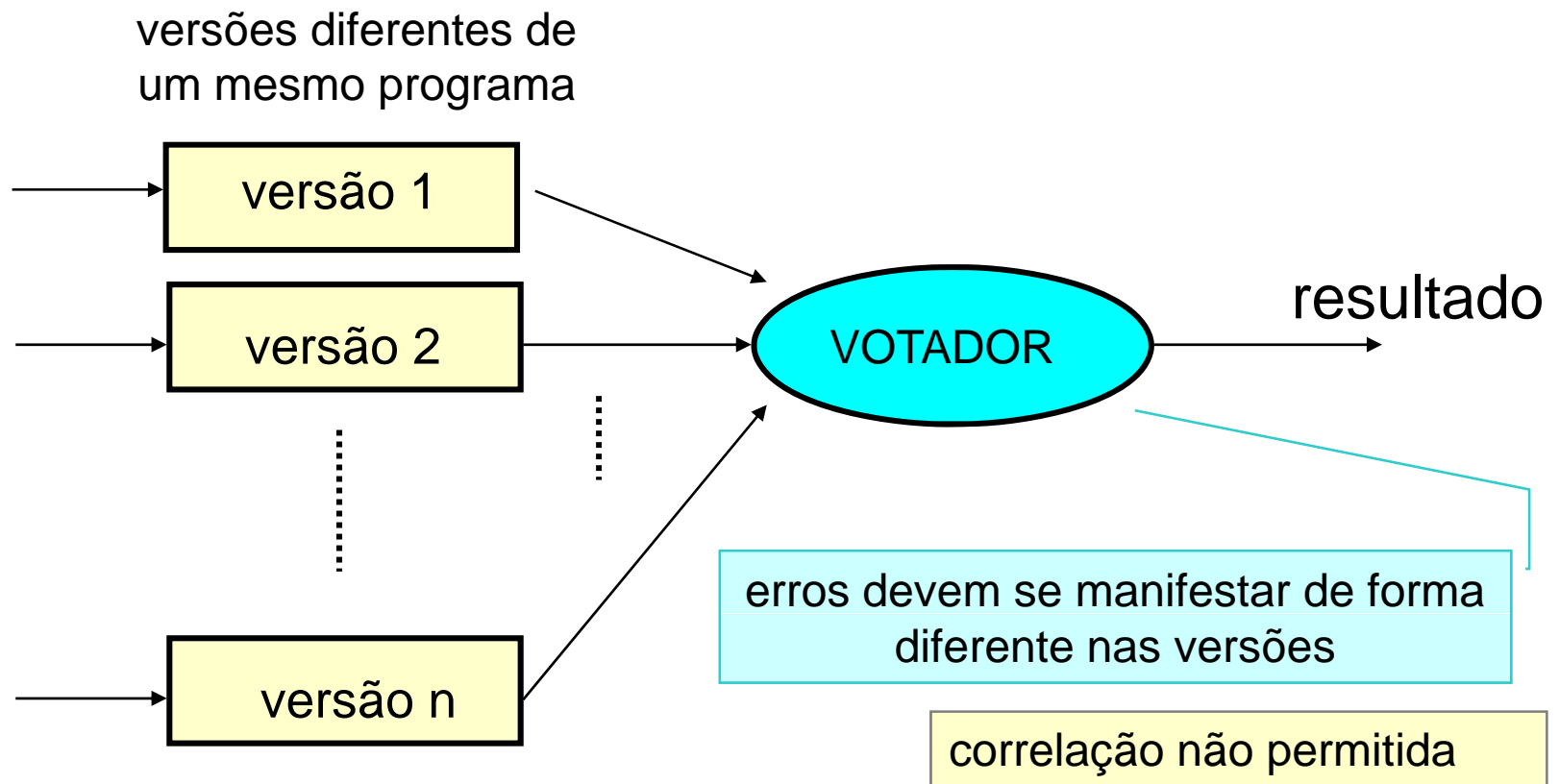
- aumento dos custos de desenvolvimento
- complexidade de sincronização
- dificuldade de evitar correlação
  - não há garantias que cópias diferentes não apresentem os mesmos erros
- falta prova formal de aumentar confiabilidade

experimentos comprovam o aumento de confiabilidade, por essa razão diversidade tem sido usada em projetos críticos

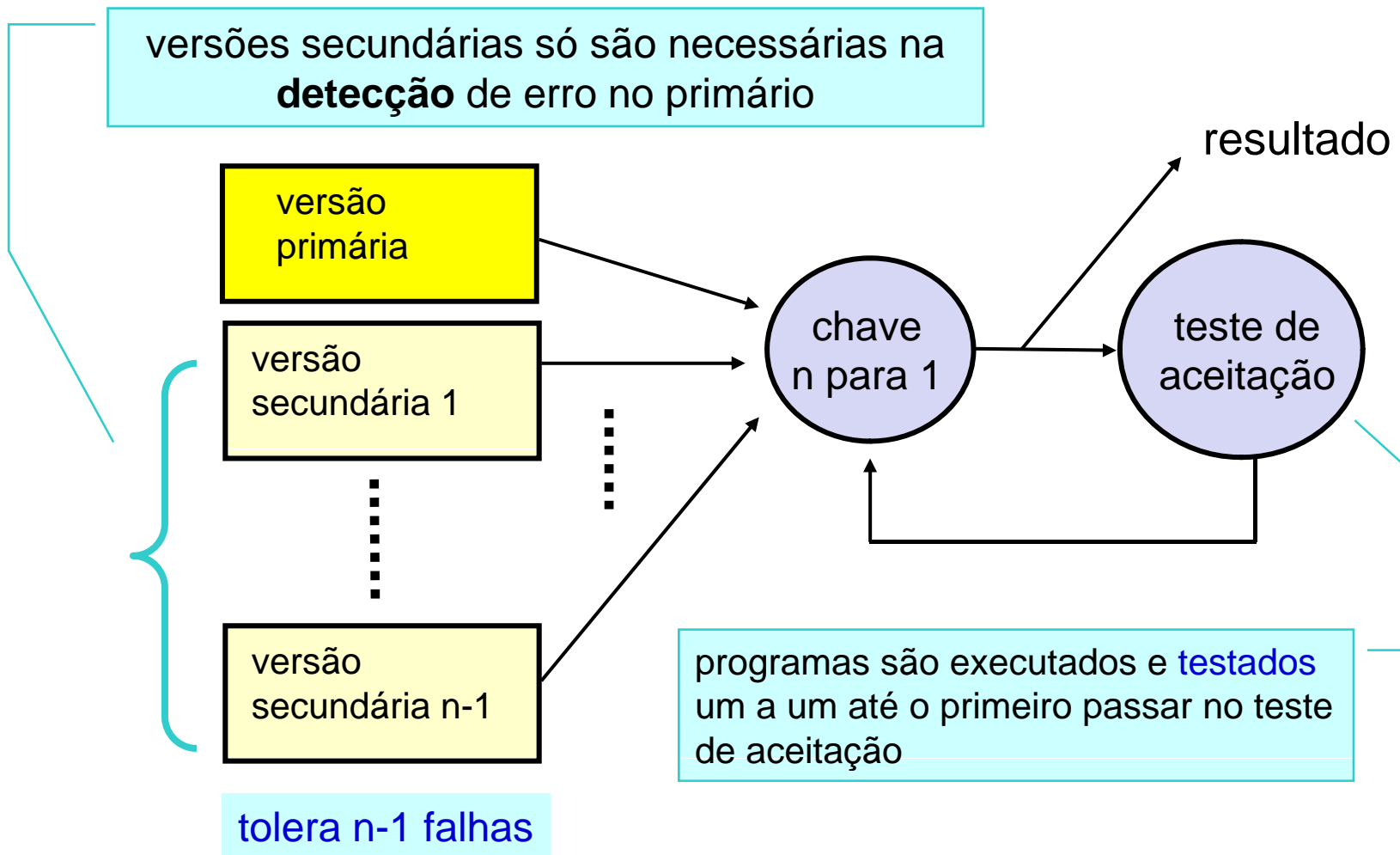
- dificuldade de determinar o número de réplicas

# Modelo

## Avizienis



# Blocos de recuperação



# Atenção

---

**se** o software fosse correto,  
**então** não seriam necessárias técnicas de tolerância a falhas para software

tolerância a falhas não substitui  
boas e tradicionais técnicas de  
programação e desenvolvimento de  
software



# Redundância temporal


---

- repete a computação no tempo
  - a computação é realizada duas ou mais vezes no mesmo equipamento
- evita custo de hardware adicional
  - mas há custo de desempenho
- aplicações usuais:
  - detecção de falhas transientes
    - falhas transientes provavelmente não afetam igualmente duas computações separadas no tempo
  - detecção de falhas permanentes

# Detecção de falhas permanentes

---

- tempo  $t_0$ 
  - computação normal
  - armazena resultado
- tempo  $t_1$ 
  - codifica dado
  - computação
  - decodifica dado
  - armazena resultado
  - compara resultados



codificação força uso  
diferente do hardware



indicação de erro

# Redundância de informação

---

- códigos de detecção de erros
    - capacidade de indicar que a informação está incorreta (ou inconsistente)
  - códigos de correção de erros
    - capacidade de correção e detecção
    - correção realizada por hardware
  - qualquer código implica no aumento do número de bits
- Exemplo: ECC
- redundância

# Bibliografia para redundância

---

- capítulo de livro
  - Johnson, Barry. An introduction to the design na analysis of the fault-tolerante systems, cap 1. **Fault-Tolerant System Design**. Prentice Hall, New Jersey, 1996
- artigos
  - Nelson, V. – **Fault Tolerant Computing: Fundamental Concepts**, IEEE Computer, 1990
    - biblioteca digital da IEEE
  - BEV LITTLEWOOD, PETER POPOV, LORENZO STRIGINI. **Modeling Software Design Diversity – A Review**. ACM Computing Surveys, Vol. 33, No. 2, June 2001, pp. 177–208.
    - biblioteca digital da ACM
  - VINCENZO DE FLORIO & CHRIS BLONDIA. **A Survey of Linguistic Structures for Application-Level Fault Tolerance**. ACM Computing Surveys, Vol. 40, No. 2, April 2008.
    - biblioteca digital da ACM

# Fundamentos de tolerância a falhas

---

Códigos de detecção  
e correção de erros

*Taisy Silva Weber*

# Redundância de informação

---

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

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

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

# Codificação

---

## ✓ código

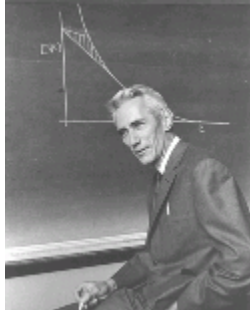
- ✓ meio de representar dados usando um conjunto definido de regras
- ✓ codificação implica, geralmente, no aumento do número de bits
- ✓ e armazenamento e processamento extra



redundância

# Um pouco de história

---



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

## Teoria da informação



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

## Teoria da codificação



# Tipos comuns de erros

---

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

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

# Unidirecionais versus randômicos

---

## ✓ erros randômicos

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

## ✓ erros unidirecionais

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

## ✓ exemplos unidirecionais

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

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

# Distância de Hamming


---

## ✓ conceito

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


## ✓ exemplos:

✓ 0000 e 0001



distância de 1

✓ 0000 e 0011



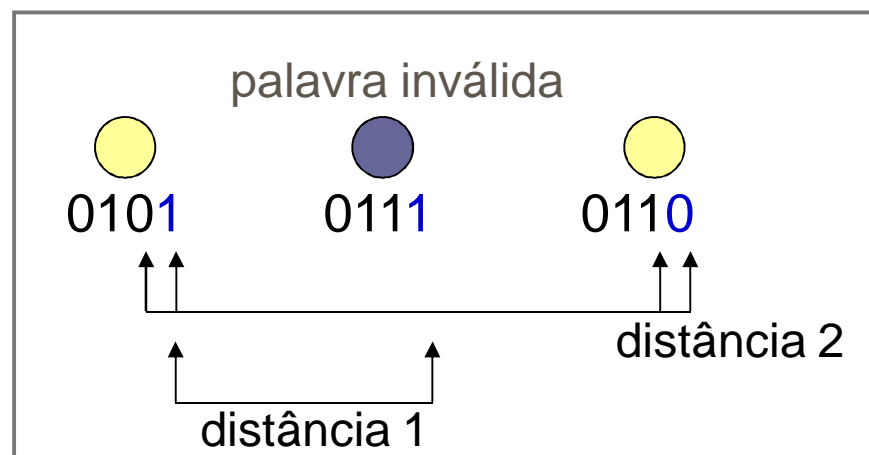
distância de ?

alterando apenas um bit → uma palavra se transforma na outra

# Distância de Hamming do código

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

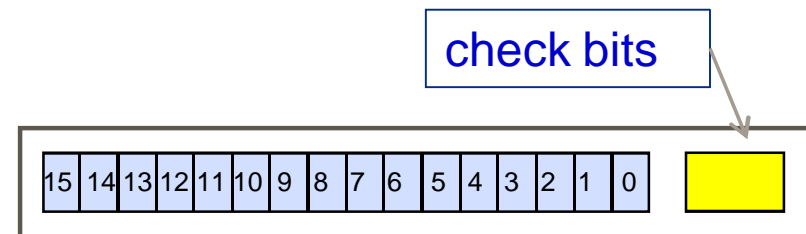
paridade par para 3 bits;  
palavra de código com 4 bits



# Código separável

---

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



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

# Exemplos de códigos

---

- ✓ Códigos de paridade

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

- ✓ m-of-n

- ✓ Duplicação

- ✓ Checksums

- ✓ Códigos de Berger

- ✓ Códigos cíclicos

- ✓ Reed-Solomon

# Códigos de paridade: uso

---

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

## ✓ paridade simples

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

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

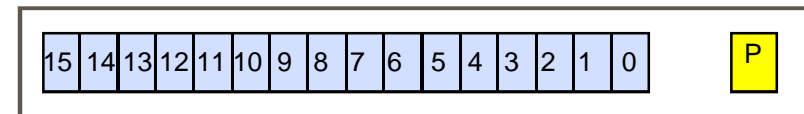
distância de Hamming = 2

- ✓ redundância pequena e simples determinação

## Bit-per-word

---

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



- ✓ capacidade de detecção
  - ✓ single bit



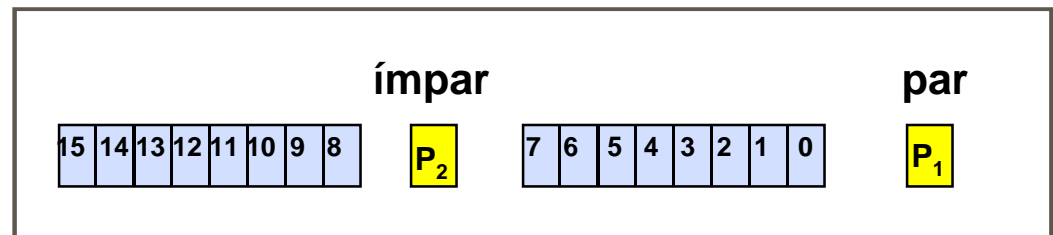
- ✓ erro múltiplo em toda a palavra

palavra transmitida:  
qualquer uma

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

# Bit-per-byte

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



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

## Bit-per-byte: exemplo

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

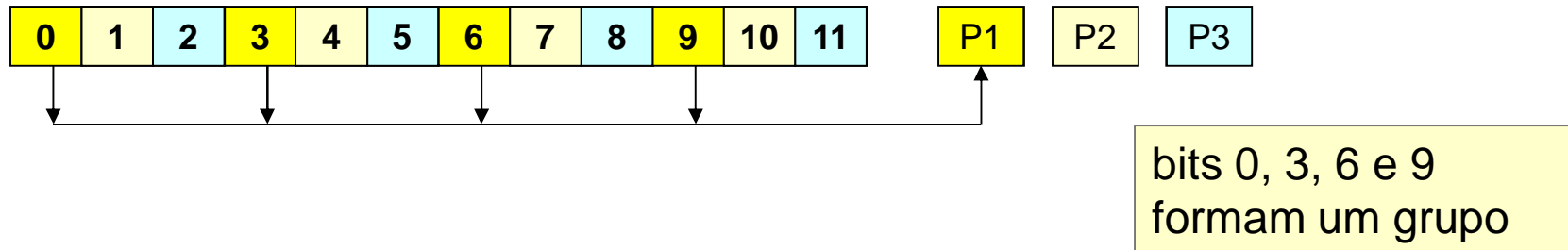
- ✓ palavra recebida: 1111 1111 1 1111 1111 1
  - ímpar
  - par
  - ✓ para o byte com paridade par: erro detectado
  - ✓ para o byte com paridade ímpar: erro não detectado

- ✓ stuck-at-zero

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

neste caso o erro é sempre detectado

# Paridade entrelaçada



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

# Paridade sobreposta

---

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

base para os códigos de correção de Hamming

# Paridade sobreposta : exemplo

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



um bit de informação aparece em mais de um grupo

determinação do número de bits de paridade :

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

# Paridade sobreposta

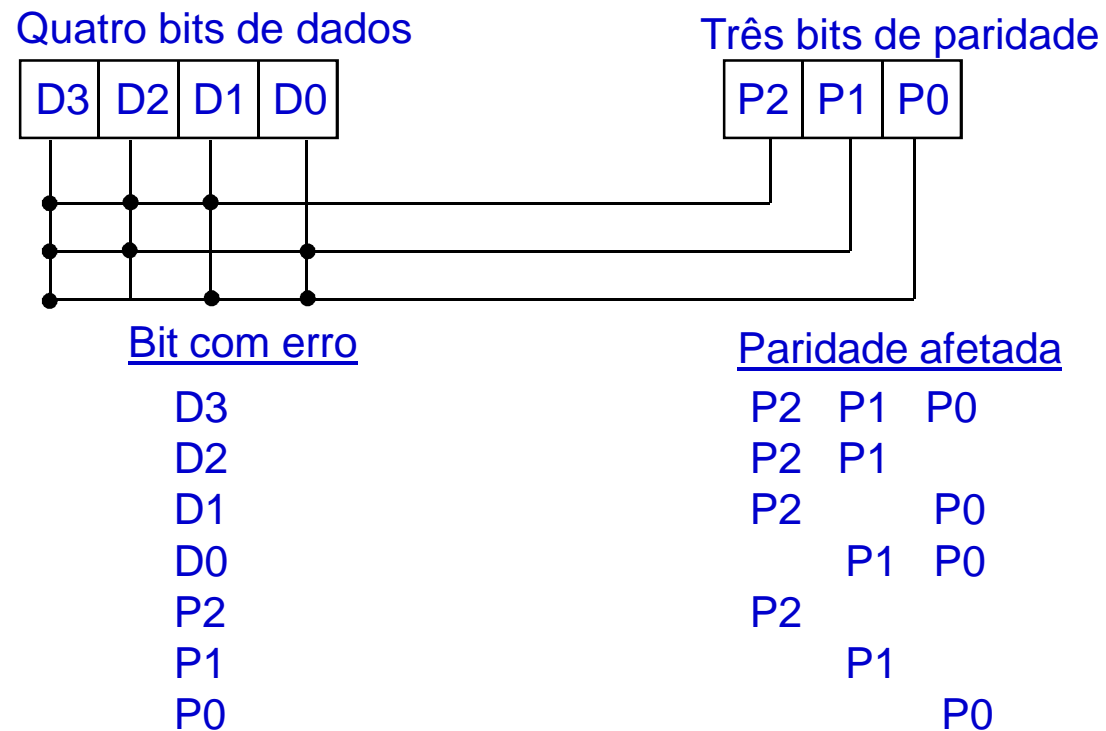
---

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

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

# Paridade sobreposta : exemplo

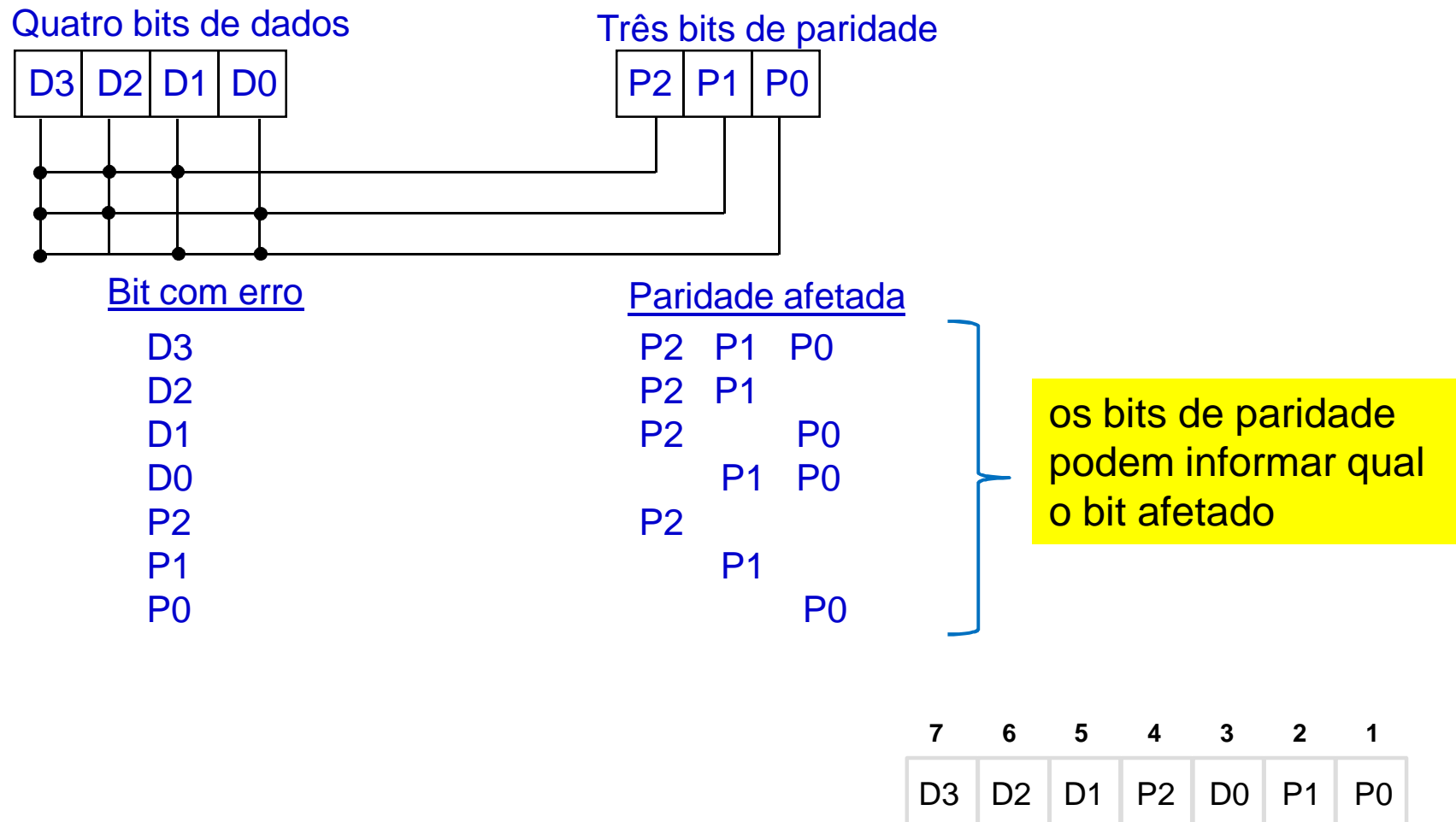
$$2^k \geq m + k + 1$$



neste exemplo redundância de 75%



# Paridade sobreposta : exemplo



# Código de Hamming

---

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

## m-of-n: definição

---

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

codificação m-de-n mais usada

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

desvantagem:  
100% redundância

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

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

# Duplicação

---

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

redundância de 100%

# Checksums

---

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

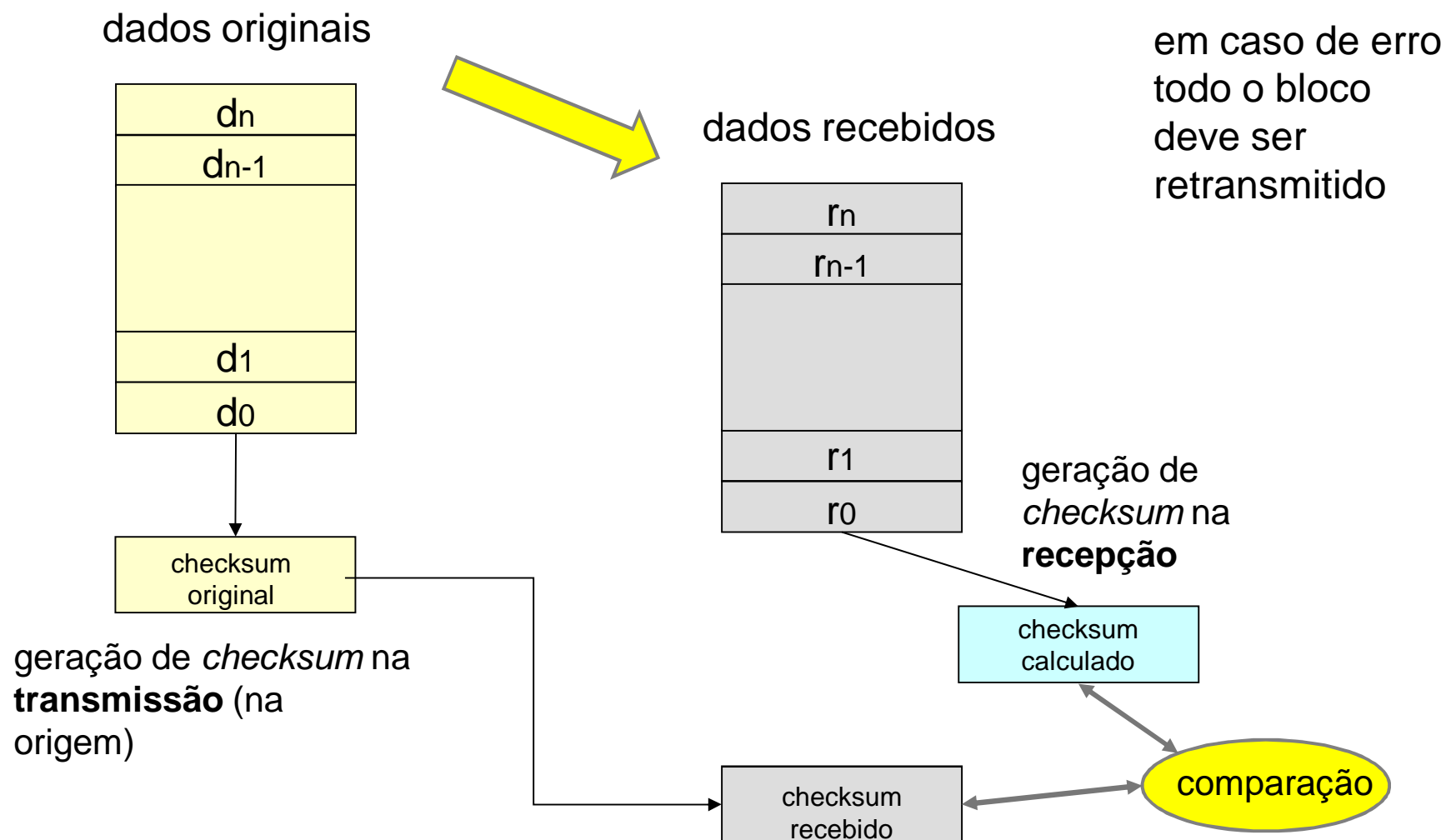
diferenças na forma como a soma é realizada

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

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

**residual**: carry-  
out da soma é  
somado ao  
checksum

# Checksum



# Códigos de Berger

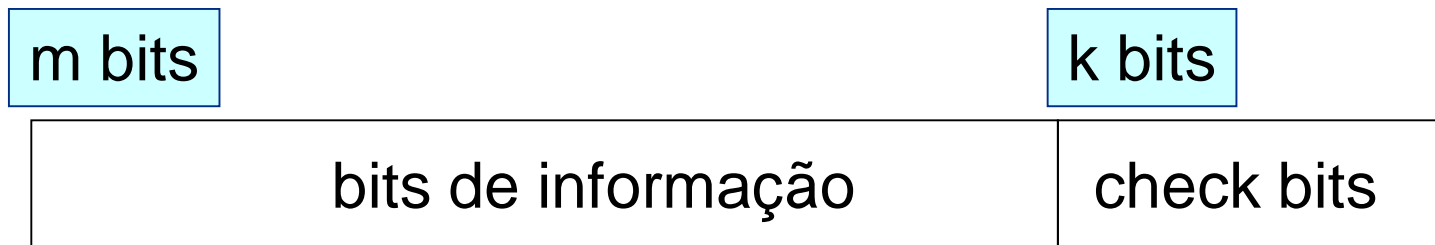
---

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



# Códigos de Berger: estrutura

---



número total de bits

$$n = m + k$$

relação entre  $K$  e  $m$

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

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

Por exemplo:

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

# Códigos de Berger: exemplo

- ✓ código de Berger para a palavra 0111010

- ✓  $m = 7$  bits de informação
- ✓  $k = \log(7+1) = 3$  check bits
- ✓ 4 bits em um
  - ✓ 4 em binário  $\rightarrow 100$
- ✓ complemento de 100 é 011
- ✓ check bits obtidos = 011

- ✓ código de Berger para 0111010 é

0111010011

check bits

# Códigos cíclicos

---

## ✓ CRC

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

## ✓ aplicações

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

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

# CRC: codificação

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



(código não-separável)

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

Grau dos polinômios:

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

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

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

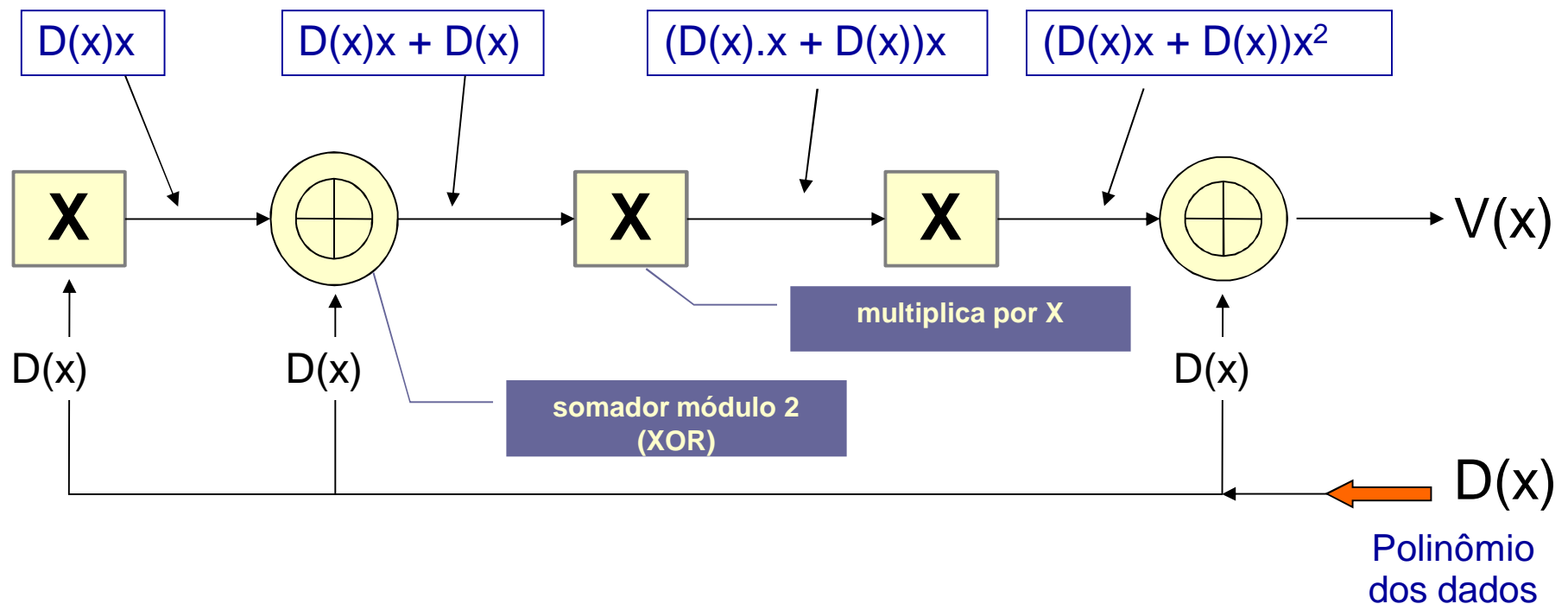
Onde,

$n$  = tamanho do código gerado

$k$  =  $n^0$  de bits da mensagem não codificada

# CRC: exemplo

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



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

## CRC: decodificação

---

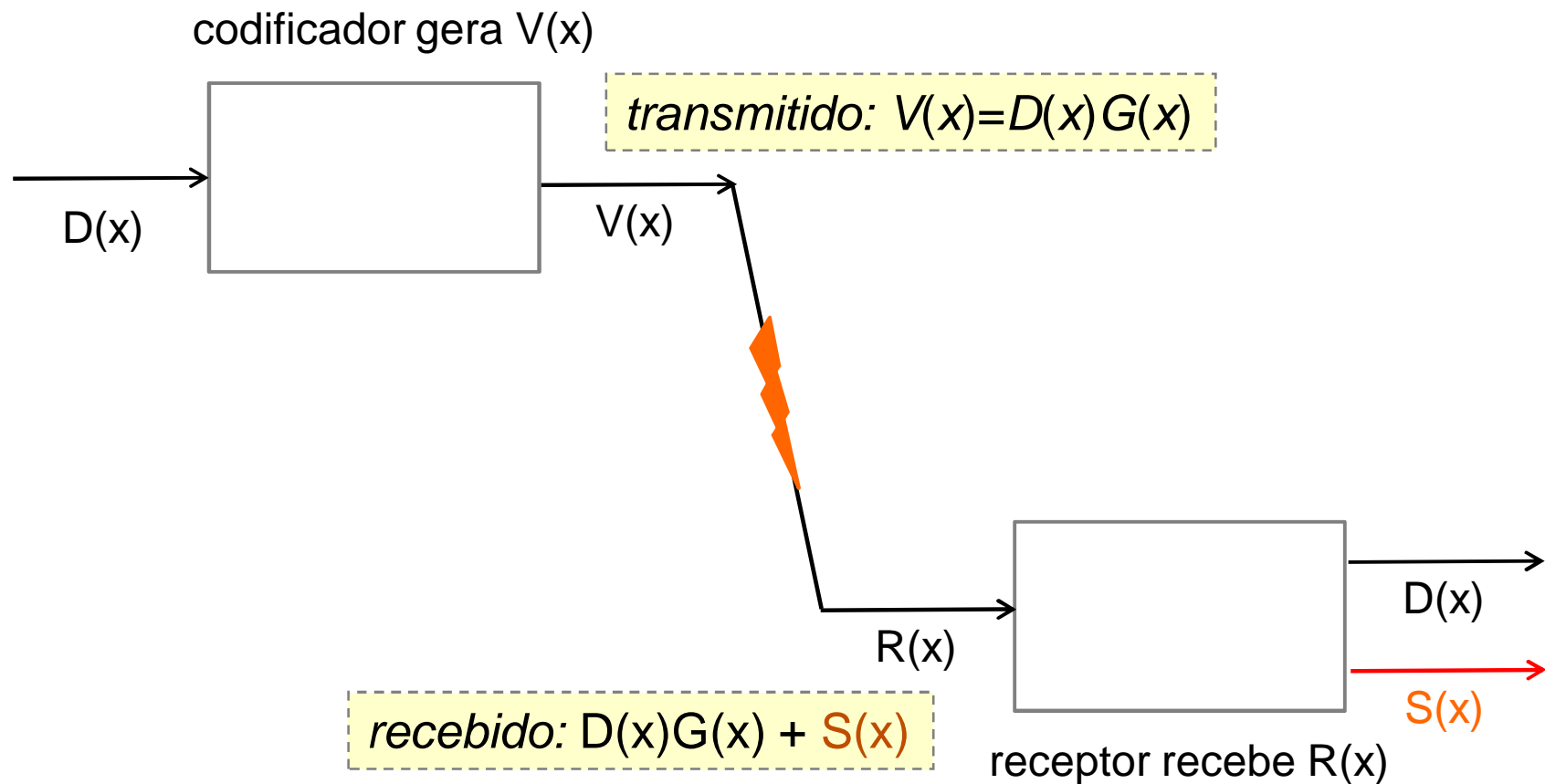
$$R(x) = D(x)G(x) + S(x)$$

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

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

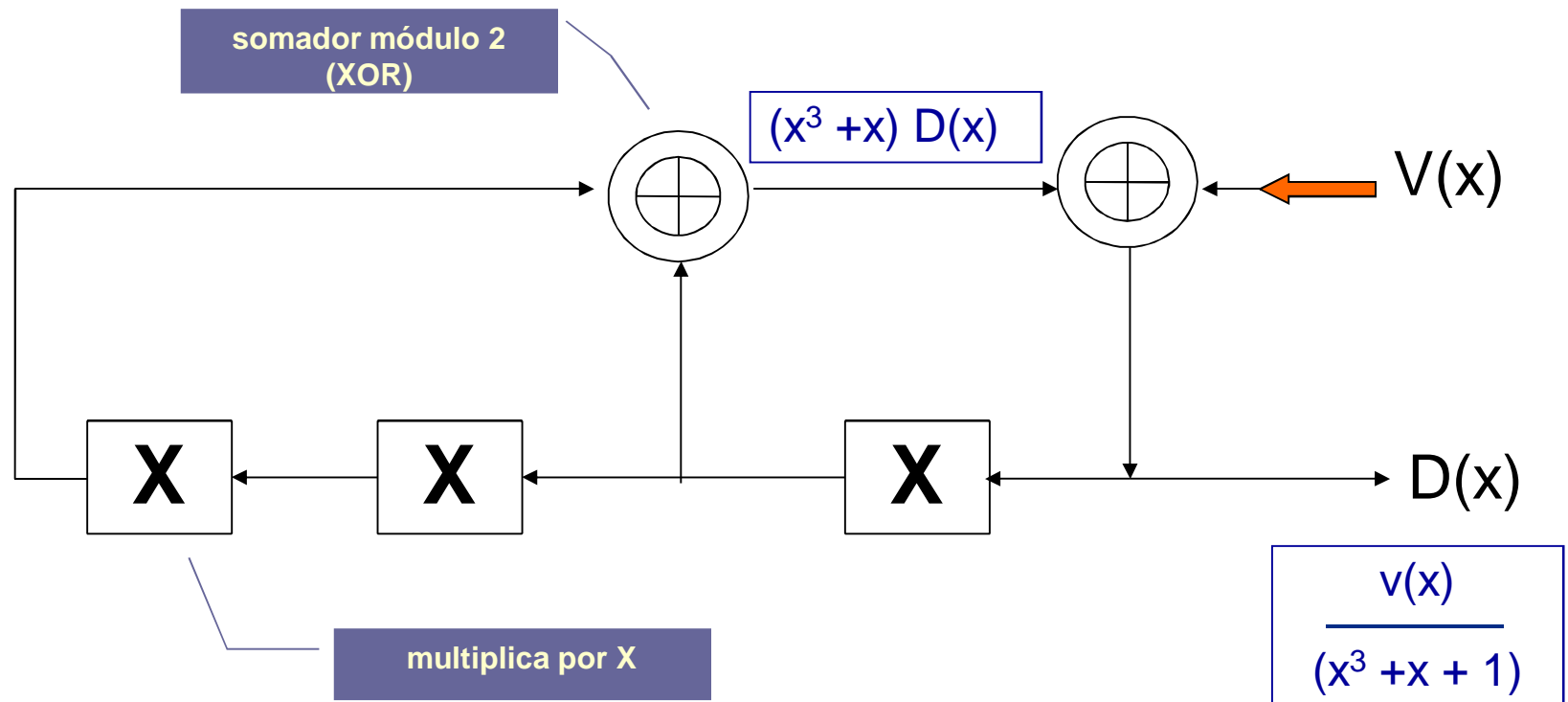
$S(x)$  é zero se não há erros

# CRC: operação



# CRC: exemplo de decodificação

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





# CRC: vantagens e desvantagens

---

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

grau do polinômio gerador

# CRC: cobertura de erros

---

## ✓ CRC-16:

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

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

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

## ✓ CRC-32:

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

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

# Reed-Solomon

---

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



[reedsolomon.tripod.com](http://reedsolomon.tripod.com)

**Irving Stoy Reed &  
Gustave Solomon**

**Irving Stoy Reed**



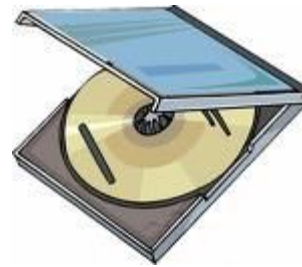
[http://ee.usc.edu/faculty\\_staff/faculty\\_directory/reed.htm](http://ee.usc.edu/faculty_staff/faculty_directory/reed.htm)

# Reed-Solomon

---

- ✓ implementação

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



- ✓ inúmeras variações e melhoramentos

- ✓ complementado com vários outros algoritmos para funções específicas

# Reed-Solomon: aplicações

---

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



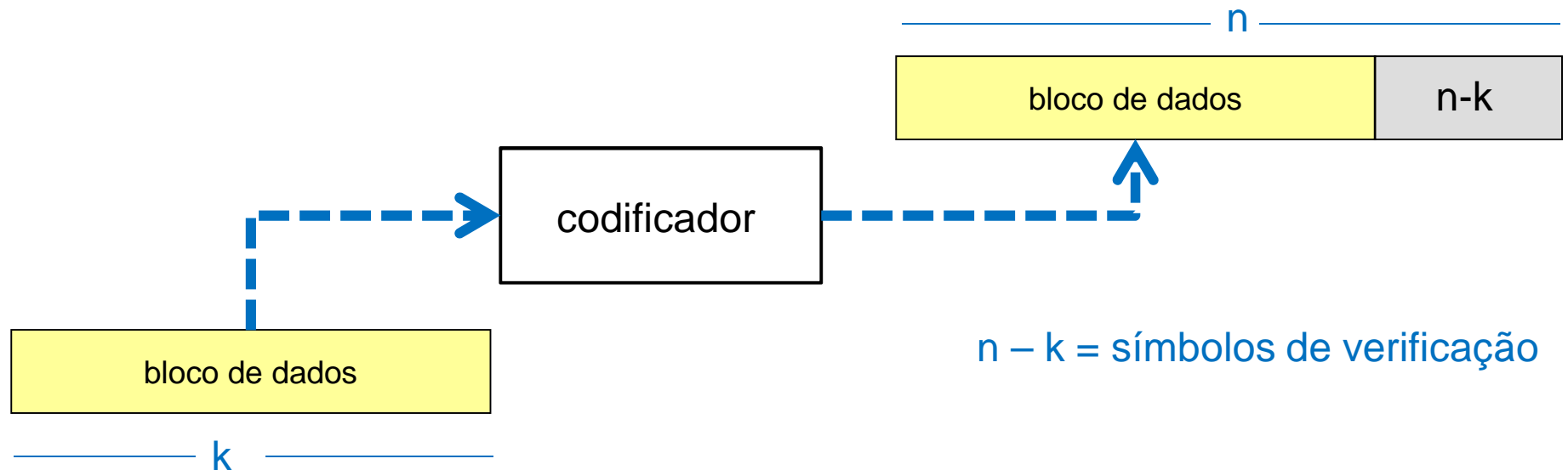
# Características

- ✓ generalização

- ✓ não é um algoritmo de codificação específico

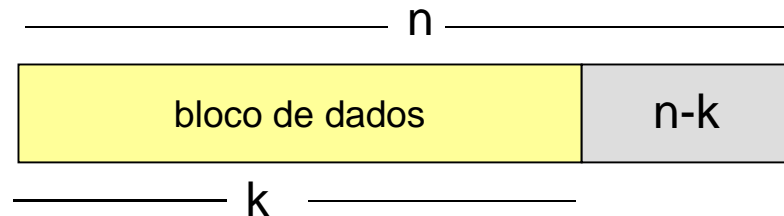
- ✓ código linear de bloco

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



# Características

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



$n - k =$  símbolos de verificação

$RS(n, k)$



# Reed-Solomon: exemplo

---

- ✓ RS (255, 223)

RS( $n, k$ )

- ✓ padrão popular

- ✓  $n = 255, k = 223$

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

- ✓  $n - k = 32$

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

- ✓ tamanho da palavra de código =  $s$

- ✓  $n = 2^s - 1$

- ✓  $s = 8 \rightarrow n = 255$

# Capacidade de correção

---

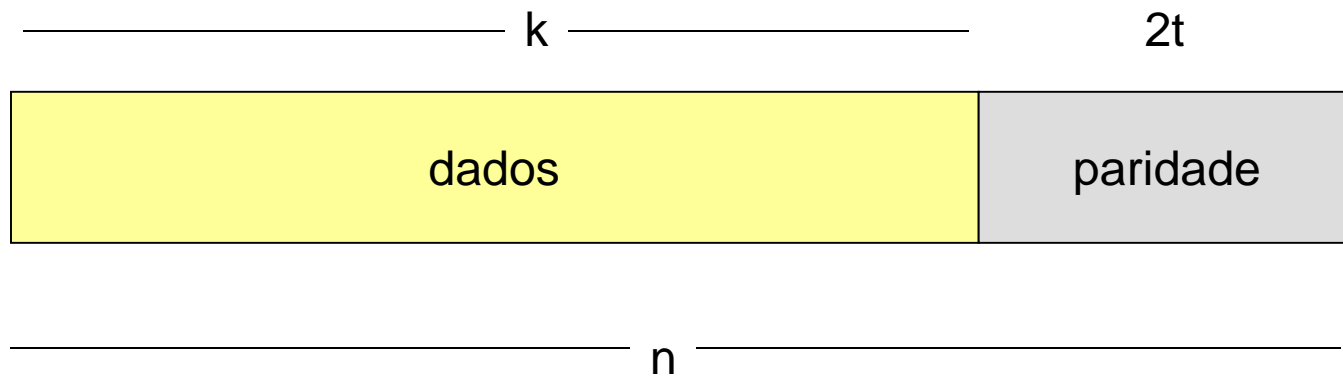
RS( $n, k$ )

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

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

# Esquema de bloco

---



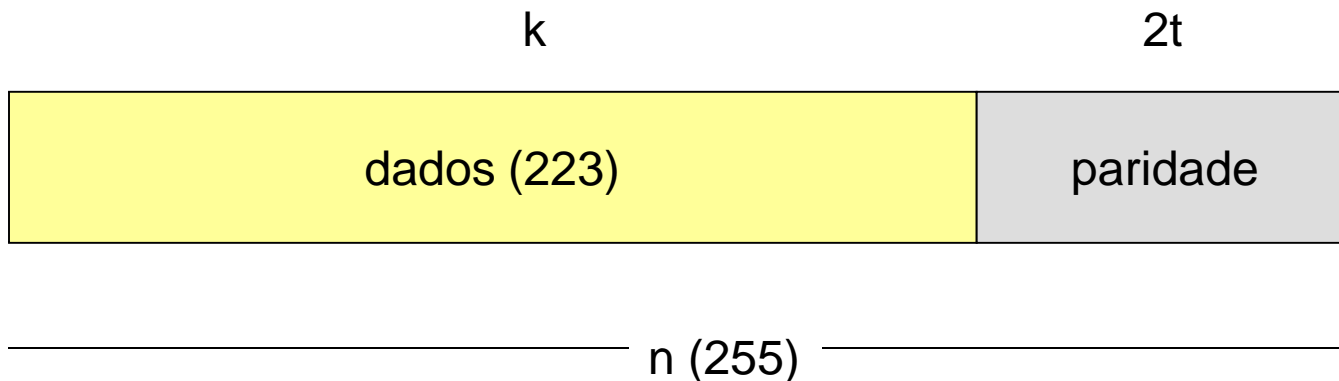
pode corrigir até  $t$  erros ou até  $2t$  erasures  
(erro com localização conhecida - omissões)

# Esquema de bloco

---

exemplo: RS(255,223)

quantas palavras erradas consegue corrigir?



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

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

mais eficiente para rajadas do que para bits isolados

# Princípio de funcionamento

---

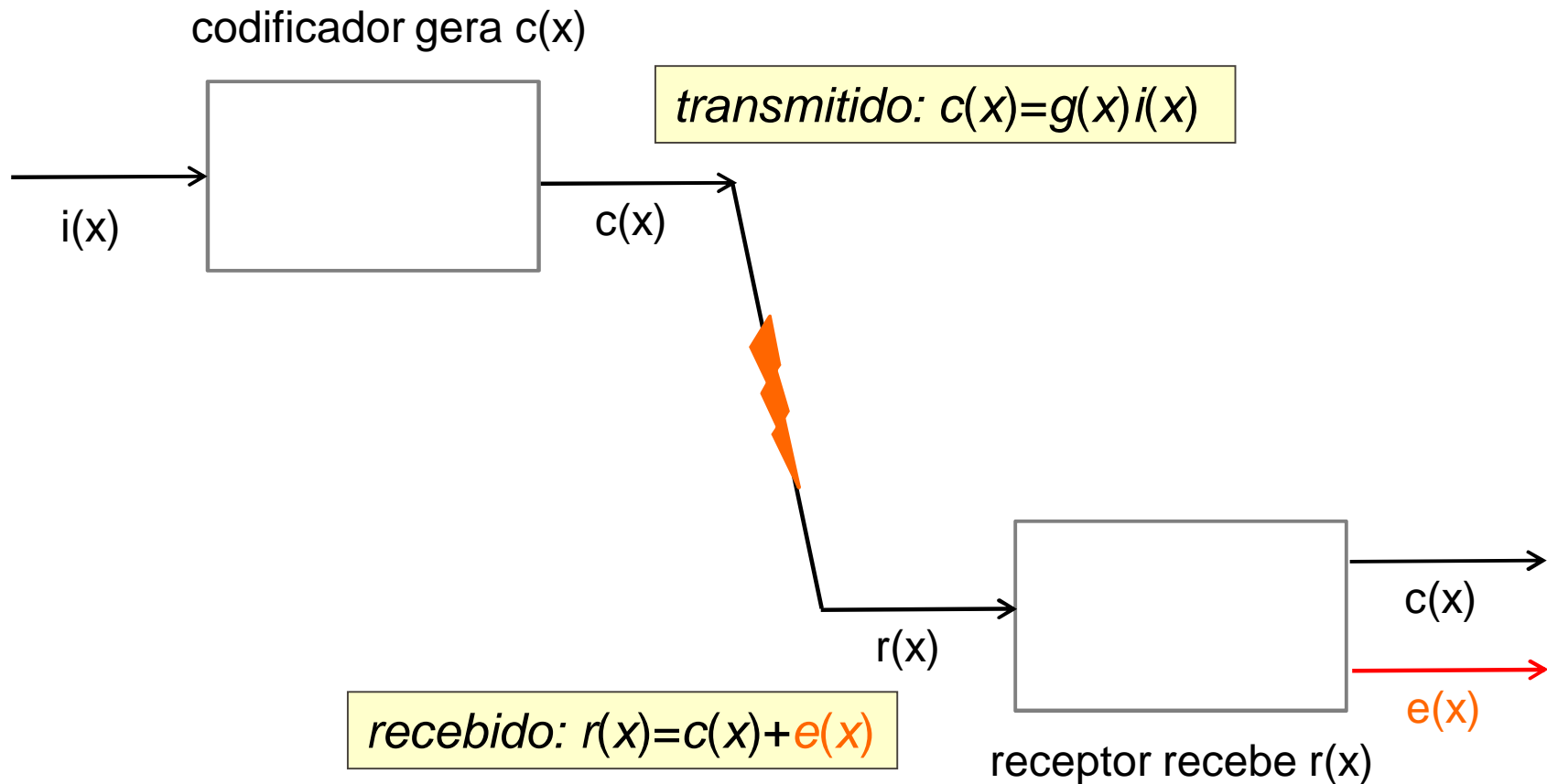
## ✓ Reed-Solomon

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

✓  $c(x) = g(x)i(x)$

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

# RS funcionamento



# Decodificação

---

- ✓ no receptor

- ✓ recebido  $r(x)$

- ✓ pode conter erros

- ✓  $r(x) = c(x) + e(x)$

- ✓ erros:  $e(x)$

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

- ✓ como determinar posição e magnitude (valor)?

- ✓ usando algoritmos específicos

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

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

# Localização do erro

---

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



## Exemplo: CD

---

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



# Implementação

---

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

## ✓ geral

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

## ✓ referências para CRC

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

# Referências para Reed-Solomon

---

- ✓ artigo

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

- ✓ alguns links

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

# Tolerância a Falhas: medidas

---

*Taisy Silva Weber*

# Medidas

---

- taxa de defeitos
- curva da banheira
- tempos médios (*mean times*)
  - MTTF, MTBF, MTTR
  - exemplos de cálculo de tempos médios
- confiabilidade
- disponibilidade
- cobertura

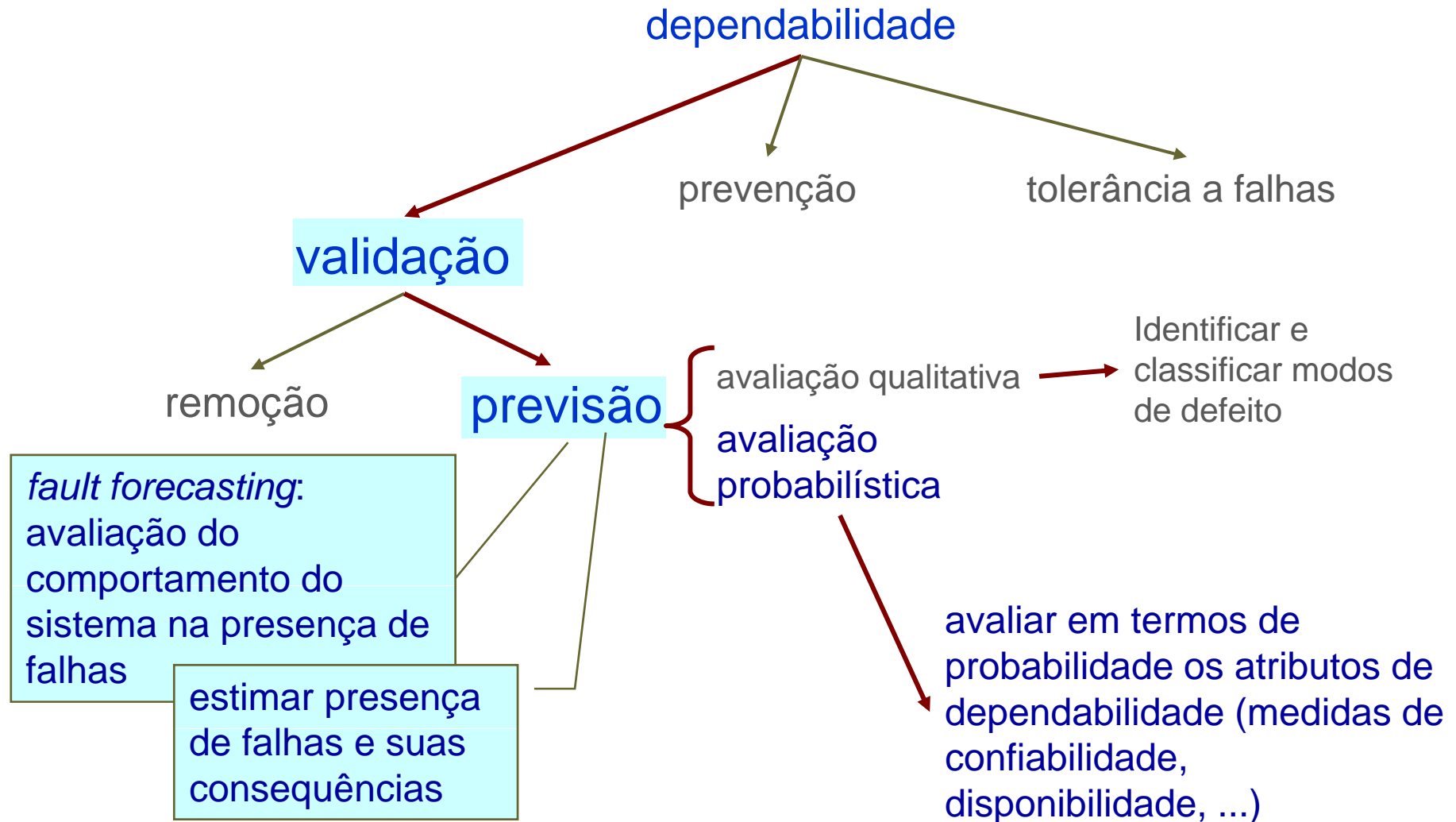
Barry W. Johnson. **Design and Analysis of Fault-Tolerant Digital Systems.**  
Addison-Wesley, 1989 (cap 4)

Johnson, B.W. **“Fault Tolerance”**  
*The Electrical Engineering Handbook*  
Ed. Richard C. Dorf  
Boca Raton: CRC Press LLC, 2000

Barry Johnson,  
**cap. 1**, livro-texto Pradhan96

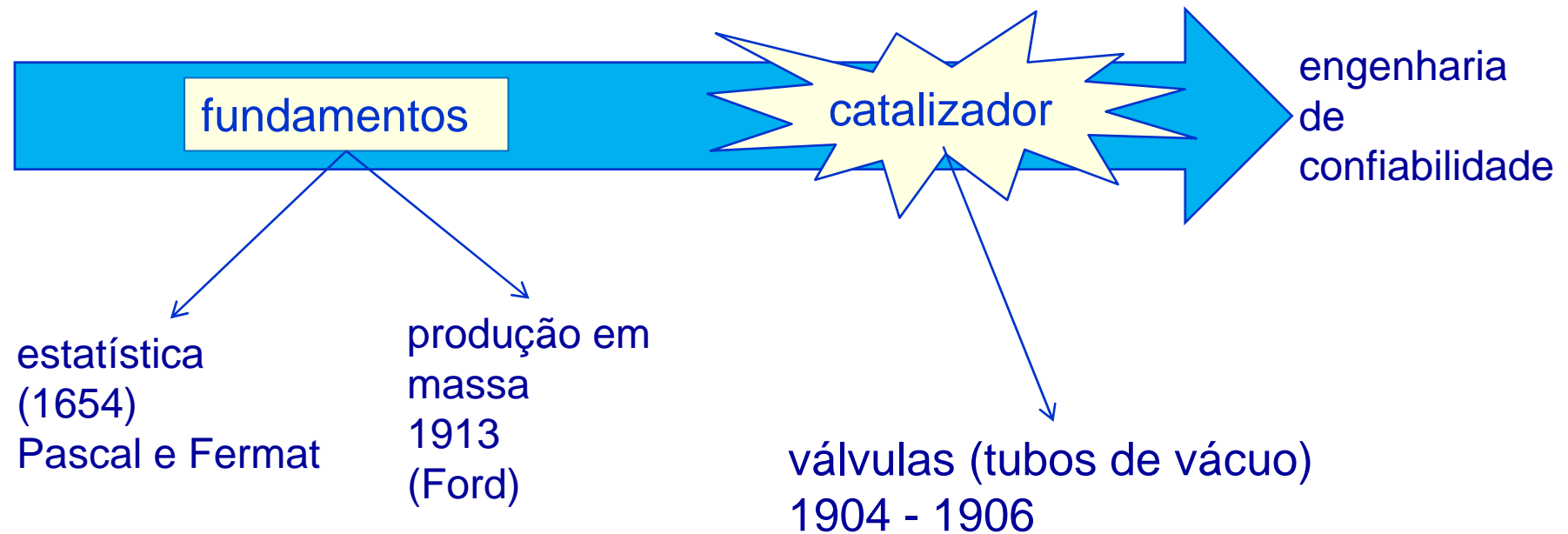
# Meios: previsão

---



# Reliability engineering

---



J.H. Saleh, K. Marais. Highlights from the early (and pre-) history of reliability engineering. Reliability Engineering and System Safety 91 (2006) 249–256



# Reliability engineering

---

- Walter Shewhart
  - década de 20 – métodos estatísticos para o controle de qualidade na produção
    - métodos não foram bem recebidos
- segunda guerra
  - acelerou a adoção de técnicas estatísticas de controle de qualidade
- Edward Deming
  - década de 50
    - não teve muito sucesso nos USA
    - se mudou para o Japão e teve enorme sucesso

# Reliability engineering

---

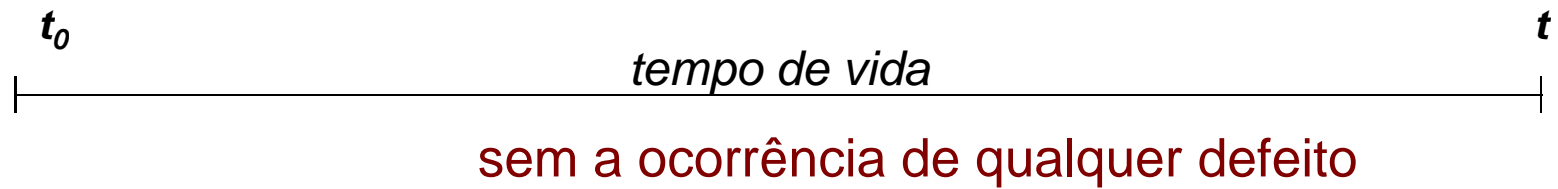
- data oficial de nascimento: 4 junho de 1954
  - relatório AGREE
    - Advisory Group on Reliability of Electronic Equipment (AGREE),
  - 1952
    - department of defense
    - industria eletrônica americana
- previsão de falhas (ou de defeitos de componentes)
  - falhas permanentes (randômicas)
  - medidas para software - problema ainda em aberto

problemas com válvulas  
(em uso até o final dos 60s)

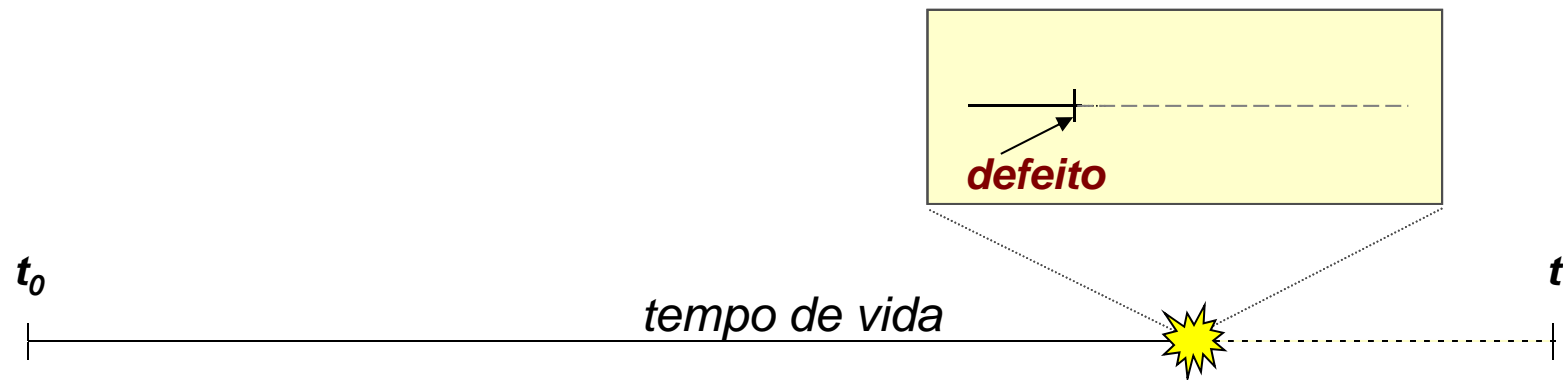
# Comportamento ideal x real

---

- ideal

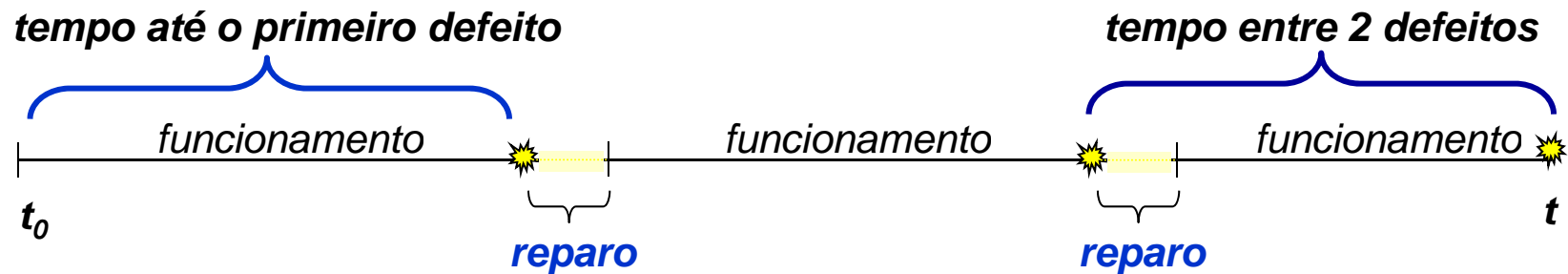


- real



# O que medir?

---



- com que frequência ocorrem defeitos?
- qual o tempo entre um defeito e outro?
- qual o tempo até o primeiro defeito?
- qual o tempo gasto para reparar cada defeito?
- quais as chances do sistema funcionar sem defeitos durante um determinado período de tempo?
- quais as chances do sistema estar funcionando em um determinado instante?

# Taxa de defeitos

---

- com que frequência ocorrem defeitos?

- taxa de defeitos

número esperado de defeitos em um dado período de tempo (*failure rate*)

- geralmente assumido valor constante
    - na verdade não é constante
    - boa aproximação: curva da banheira
  - unidade: defeitos por unidade de tempo

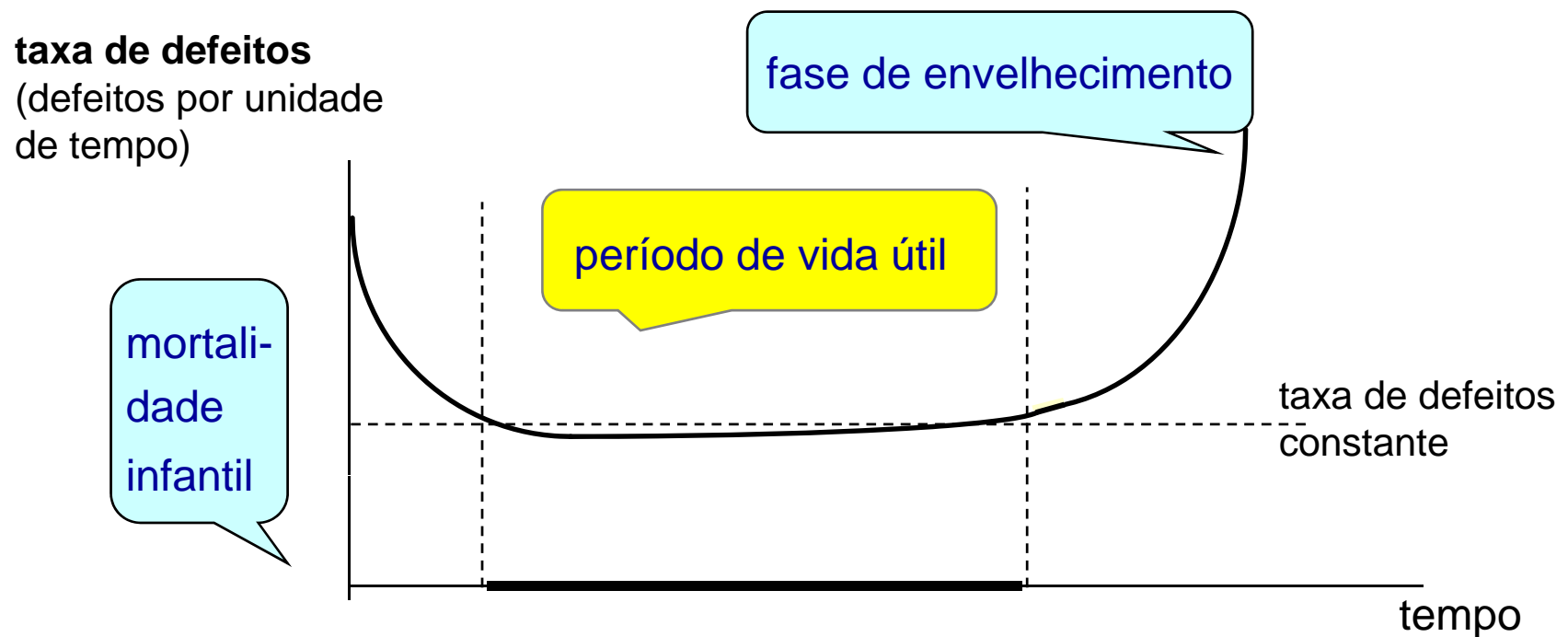
Exemplo: um computador apresenta defeito a cada 2000 horas, qual a taxa de defeito?

- função:

- $z(t)$  - hazard function, hazard rate ou taxa de defeitos (*failure rate*)

# Curva da banheira

fases de mortalidade infantil e envelhecimento muito pequenas comparadas ao período de vida útil



válido para hardware (componentes eletrônicos)

# Mortalidade infantil

---

- alta taxa de defeitos que diminui rapidamente no tempo
  - componentes fracos e mal fabricados

mortalidade infantil é uma fase de curto período de duração

- burn-in: remoção de componentes fracos
  - operação acelerada de componentes antes de colocá-los no produto final
  - só entram em operação componentes que sobreviveram à mortalidade infantil

# Envelhecimento

---

- taxa de defeitos aumenta rapidamente com o tempo
  - devido ao desgaste físico do componente
  - conhecendo o início da fase de envelhecimento é possível substituir o componente
    - sistema volta a operar na fase de vida útil

envelhecimento é também uma fase de curto período de duração

ideal é evitá-la



# Tempo de vida útil

---

$\lambda$  – taxa de defeitos **constante**

- unidade: defeitos por hora

$\lambda$  corresponde a taxa de defeitos no tempo de vida útil

- essa fase apresenta um serviço mais previsível em relação a falhas
- relação exponencial entre confiabilidade e tempo
  - usa  $\lambda$  – taxa de defeitos constante
  - válido para hardware
    - será visto mais adiante

$$R(t) = e^{-\lambda t}$$

# Curva da banheira em software

---

- software comporta-se diferente do hardware
  - erros (bugs) são constantemente removidos
    - taxa de defeitos continua caindo com o tempo
    - confiabilidade aumenta com o tempo ?

considerar alterações, adaptações ou mudança de plataforma (sisop e hardware)

- fase de envelhecimento de software ?
  - obsolescência dos programas
  - alterações nas plataformas
  - *aging*

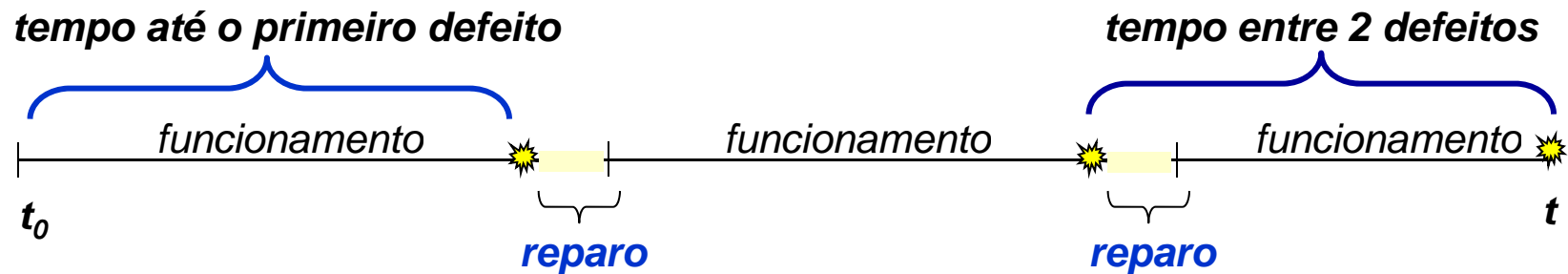
# Exemplos de taxa de defeitos

---

<b>categoria</b>	<b>módulo</b>	<b>taxa de defeitos</b>
CPU	processador	$19,0 \times 10^{-6}$ /hrs
	memória	$13,0 \times 10^{-6}$ /hrs
saída	relé	$9,2 \times 10^{-6}$ /hrs
	triac	$33,8 \times 10^{-6}$ /hrs
entrada	conversor A/D	$10,4 \times 10^{-6}$ /hrs
comunicações	controlador de barramento	$19,8 \times 10^{-6}$ /hrs
outros	fonte	$33,0 \times 10^{-6}$ /hrs
	rack	$2,6 \times 10^{-6}$ /hrs

fator de incerteza = 10

# Tempos médios



- com que frequência ocorrem defeitos?
- qual o tempo entre um defeito e outro?
- qual o tempo até o primeiro defeito?
- qual o tempo gasto para reparar cada defeito?
- quais as chances do sistema funcionar sem defeitos durante um determinado período de tempo?
- quais as chances do sistema estar funcionando em um determinado instante?

taxa de defeitos

# Medidas

---

- MTTF

- tempo esperado até a primeira ocorrência de defeito

mean time to failure

- MTTR

- tempo médio para reparo do sistema

mean time to repair

- MTBF

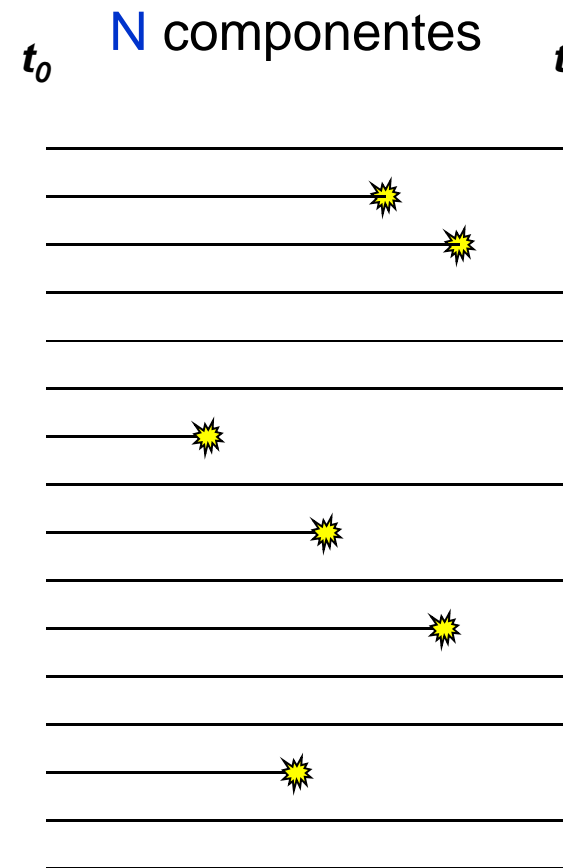
- tempo médio entre defeitos do sistema

mean time between failures

# MTTF - mean time to failure

---

- tempo esperado de operação do sistema antes da ocorrência do primeiro defeito
  - considera-se  $N$  sistemas idênticos colocados em operação a partir do tempo  $t=0$
  - mede-se o tempo de operação  $t_i$  de cada um até apresentar defeito
  - **MTTF** é o tempo médio de operação



# MTTF - mean time to failure

---

- tempo esperado de operação do sistema antes da ocorrência do primeiro defeito

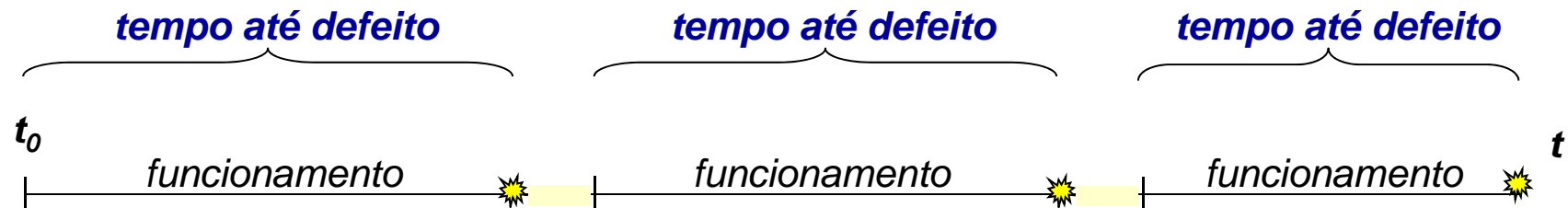
$$MTTF = \sum_{i=1}^N \frac{t_i}{N}$$

quanto maior a quantidade de amostras **N**, mais próximo do valor real será o **MTTF** estimado

considerando  $R(t) = e^{-\lambda t}$

$$MTTF = 1/\lambda$$

# MTTF



$$MTTF = \sum_{i=1}^N \frac{t_i}{N}$$

para **um** único sistema o procedimento é semelhante:  $t_i$  passa a ser  $\Delta t_i$ , o intervalo de tempo em operação entre os defeitos, e  $N$  o número de defeitos



# MTTR - mean time to repair

---

- tempo médio de reparo do sistema
    - difícil de estimar
      - geralmente usa-se **injeção de falhas**
      - injeta-se uma falha de cada vez e mede-se o tempo
    - nova constante  $\mu$ 
      - taxa de reparos
- $\mu$  = número de reparos por hora

$$MTTR = \frac{1}{\mu}$$

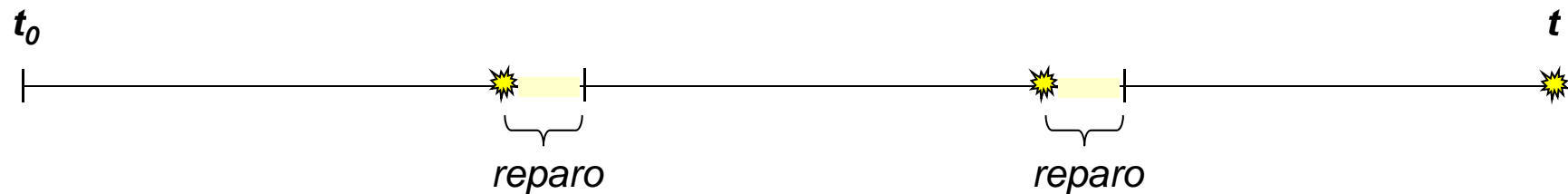
em sistemas de **alta disponibilidade**, é importante diminuir o **tempo de reparo** para aumentar a disponibilidade do sistema

# MTTR

---

$R_i$  tempo de reparo da falha  $i$

$n$  número de falhas



$$MTTR = \sum_{i=1}^n R_i / n \quad \text{ou} \quad MTTR = 1/\mu \quad \text{sendo } \mu = \text{taxa de reparo}$$

quanto maior o número de amostras, melhor

# MTTR: Exemplo

grandemente simplificado



tempo de reparo do 1º defeito ( $R_1$ ) =  $0,5 h$

tempo de reparo do 2º defeito ( $R_2$ ) =  $1 h$

$$MTTR = (R_1 + R_2) / n^{\circ} \text{ reparos}$$

$$MTTR = 1,5 / 2$$

$$MTTR = 0,75 h$$

# Mean Time Between Failure

---

- $MTBF = MTTF + MTTR$ 
  - diferença numérica pequena em relação a MTTF
    - os tempos de operação são geralmente muito maiores que os tempos de reparo
    - na prática valores numéricos muito aproximados (não faz diferença usar um ou outro)
  - considera-se:
    - reparo coloca sistema em condições ideais de operação

e se o MTBF for maior que o tempo até obsolescência?

# MTBF

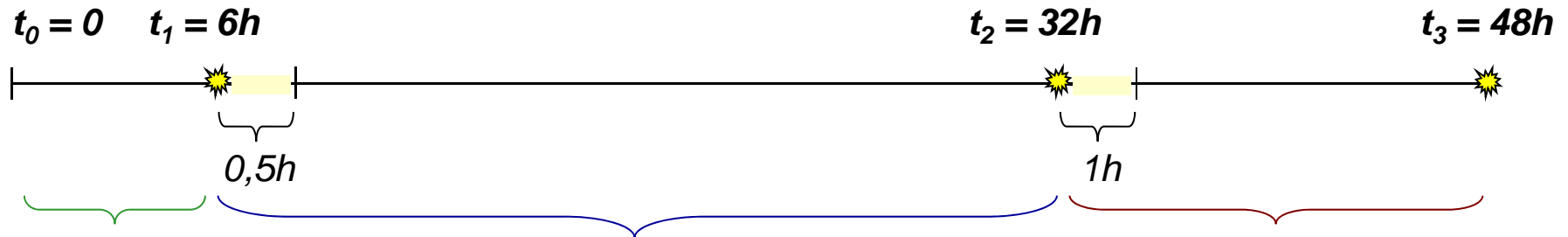
---



$$\text{MTBF} = \sum_{i=1}^n \Delta d_i / n \quad \text{ou} \quad \text{MTBF} = \text{MTTF} + \text{MTTR}$$

# MTBF: Exemplo

grandemente simplificado



tempo entre o início e o 1º defeito ( $\Delta d_1$ ) = 6 h

tempo entre 1º e 2º defeitos ( $\Delta d_2$ ) = 26 h

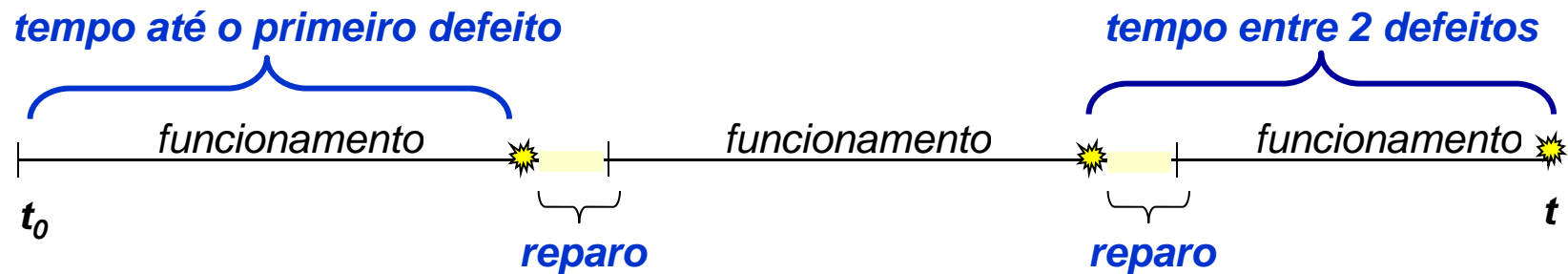
tempo entre 2º e 3º defeitos ( $\Delta d_3$ ) = 16h

$MTBF = (\Delta d_1 + \Delta d_2 + \Delta d_3) / n^\circ \text{ defeitos}$

$MTBF = 48 / 3$

**$MTBF = 16 \text{ h}$**

# Demais medidas



- com que frequência ocorrem defeitos?
- qual o tempo entre um defeito e outro?
- qual o tempo até o primeiro defeito?
- qual o tempo gasto para reparar cada defeito?
- quais as chances do sistema funcionar sem defeitos durante um determinado período de tempo?
- quais as chances do sistema estar funcionando em um determinado instante?

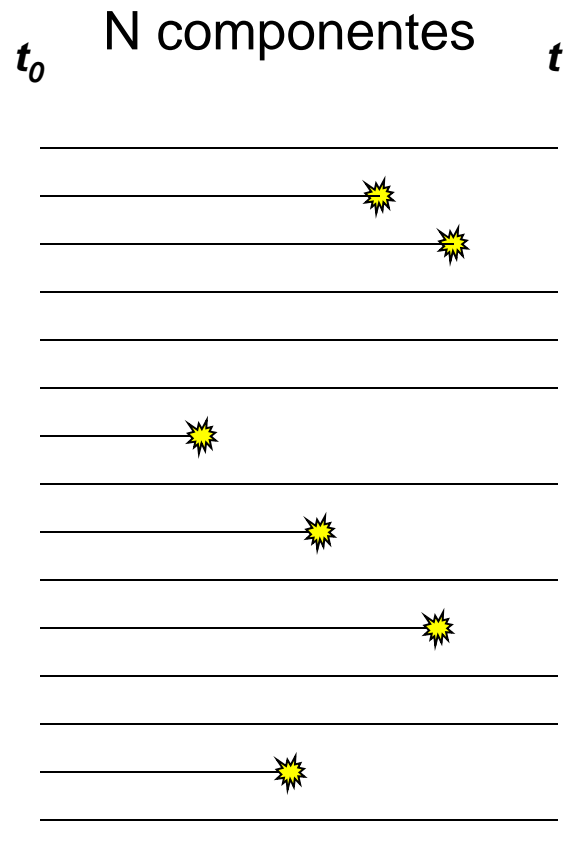
taxa de defeitos

MTBF

MTTF

MTTR

# Confiabilidade e taxa de defeitos



$N$  componentes idênticos, operacionais em  $t_0$

$N_f(t)$  número de componentes com defeito em  $t$

$N_o(t)$  núm. de componentes operacionais em  $t$

$$R(t) = N_o(t) / N = N_o(t) / (N_o(t) + N_f(t))$$

**confiabilidade:** a probabilidade que um componente tenha sobrevivido no intervalo

$Q(t)$  é a não confiabilidade

$$Q(t) = N_f(t) / N = N_f(t) / (N_o(t) + N_f(t))$$

$$R(t) = 1,0 - Q(t) = 1 - N_f(t) / N$$



# Confiabilidade e taxa de defeitos

---

$$R(t) = 1,0 - Q(t) = 1 - N_f(t) / N$$

fazendo a **diferencial** da confiabilidade em relação ao tempo

$$dR(t)/dt = (- 1/N) dN_f(t) / dt$$

$dN_f(t) / dt$  é a **taxa instantânea** em que componentes estão falhando.

$$dN_f(t) / dt = (- N) dR(t) / dt$$

Dividindo esta **taxa** por **No(t)**

# Confiabilidade e taxa de defeitos

---

$$dN_f(t) / dt = (-N) dR(t) / dt$$

dividindo por  $N_o(t)$

$$z(t) = dN_f(t) / dt \cdot 1/N_o(t) = (-N / N_o(t)) \cdot dR(t) / dt$$

$$R(t) = N_o(t) / N$$

$$z(t) = -1/R(t) \cdot dR(t) / dt$$

$z(t)$  - hazard function ou taxa de defeitos

$$dR(t) / dt = -R(t) \cdot z(t)$$

solução geral dessa equação é

$$R(t) = e^{-\int z(t) dt}$$

considerando  $z(t)$  constante então:

$$R(t) = e^{-\lambda t}$$

# Confiabilidade

---

probabilidade de que um sistema funcione corretamente durante um intervalo de tempo  $[t_0, t]$

- para um taxa de defeitos constante  $\lambda$  a confiabilidade  $R(t)$  varia exponencialmente em função do tempo
  - sistema na fase de *vida útil*: taxa de defeitos constante  $\lambda$
- $R(t) = e^{-\lambda t}$  *exponential failure law*
  - é a mais usada relação entre confiabilidade e tempo
  - válida principalmente para componentes eletrônicos

# Distribuição de Weibull

Weibull W. A statistical distribution function of wide applicability. J Appl Mech 1951;18:293–7.

- se taxa de defeitos varia com o tempo
  - $z(t)$  distribuição de Weibull
    - importante para modelagem de software onde a confiabilidade pode inclusive aumentar com o tempo
  - $z(t) = \alpha\lambda(\lambda t)^{\alpha-1}$  para  $\alpha > 0$  e  $\lambda > 0$

- $R(t) = e^{-(\lambda t)^\alpha}$

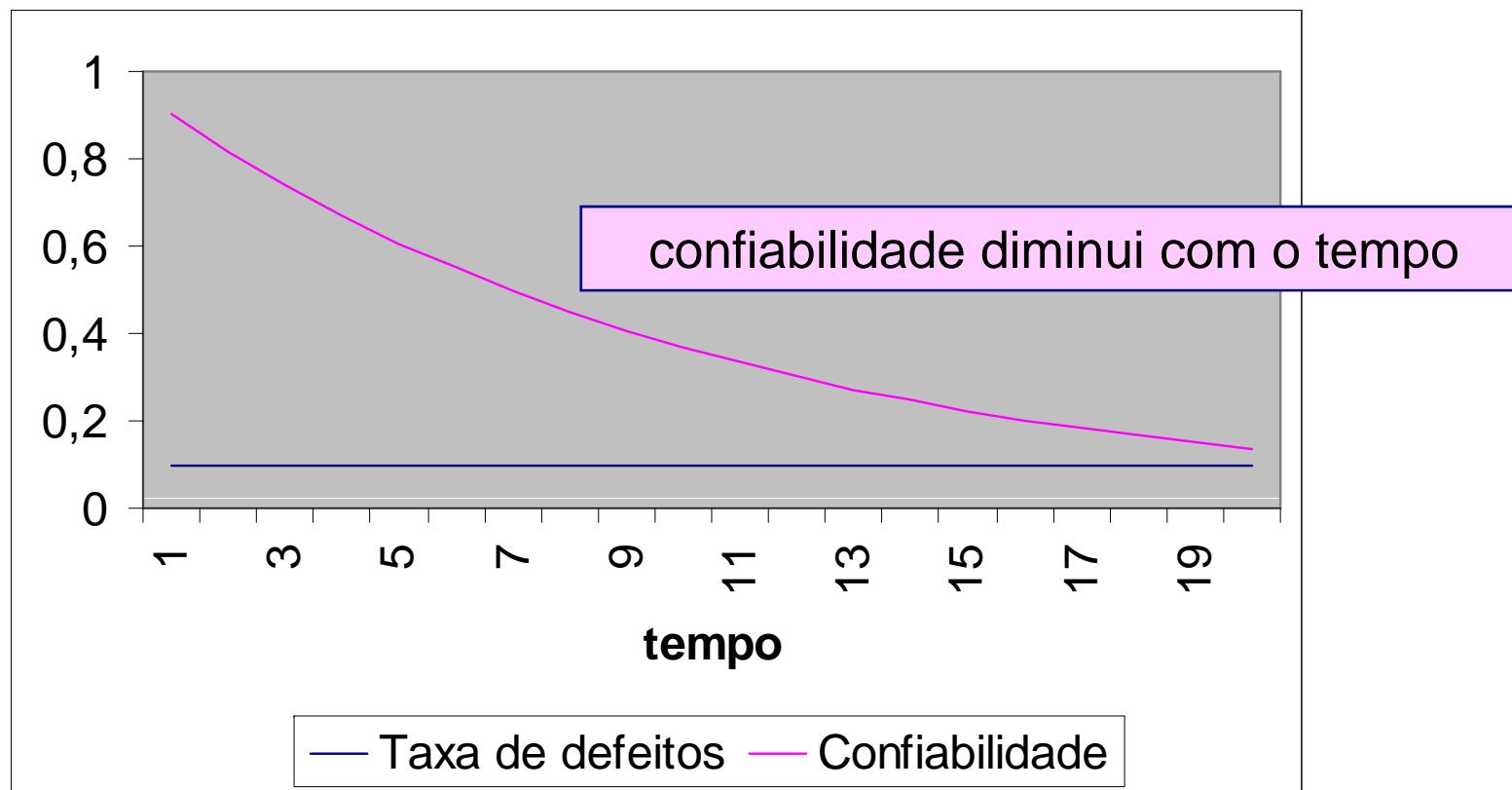
$\alpha$  e  $\lambda$  são constantes que controlam a variação de  $z(t)$  no tempo

- para  $\alpha=1$   $z(t) = \text{constante} = \lambda$
- para  $\alpha>1$   $z(t) = \text{aumenta com o tempo}$
- para  $\alpha<1$   $z(t) = \text{diminui com o tempo}$

# Confiabilidade

- para:  $\alpha=1$   $\lambda=0,1$

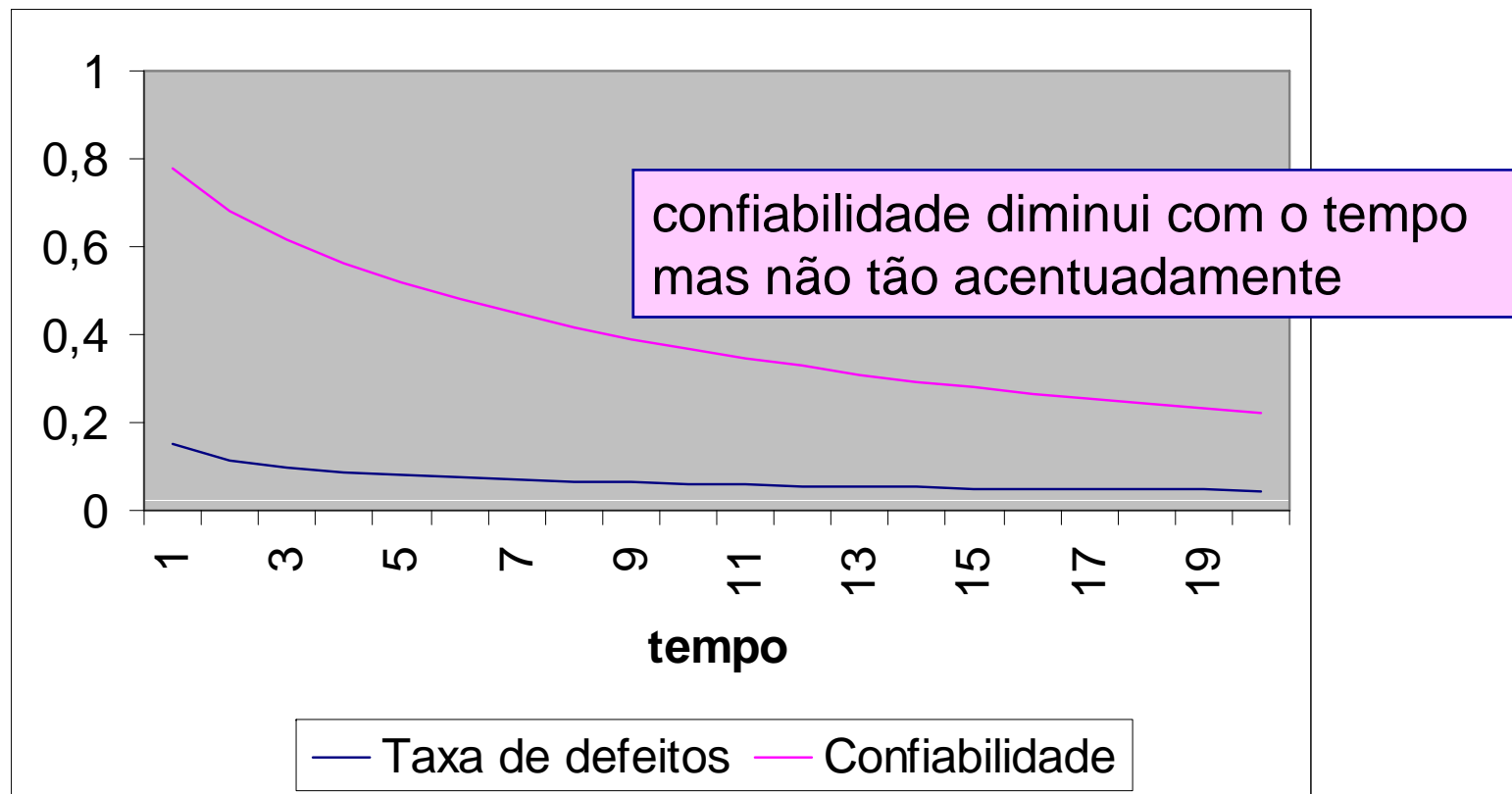
taxa de defeitos constante



# Confiabilidade

- para:  $\alpha=0,6$   $\lambda=0,1$

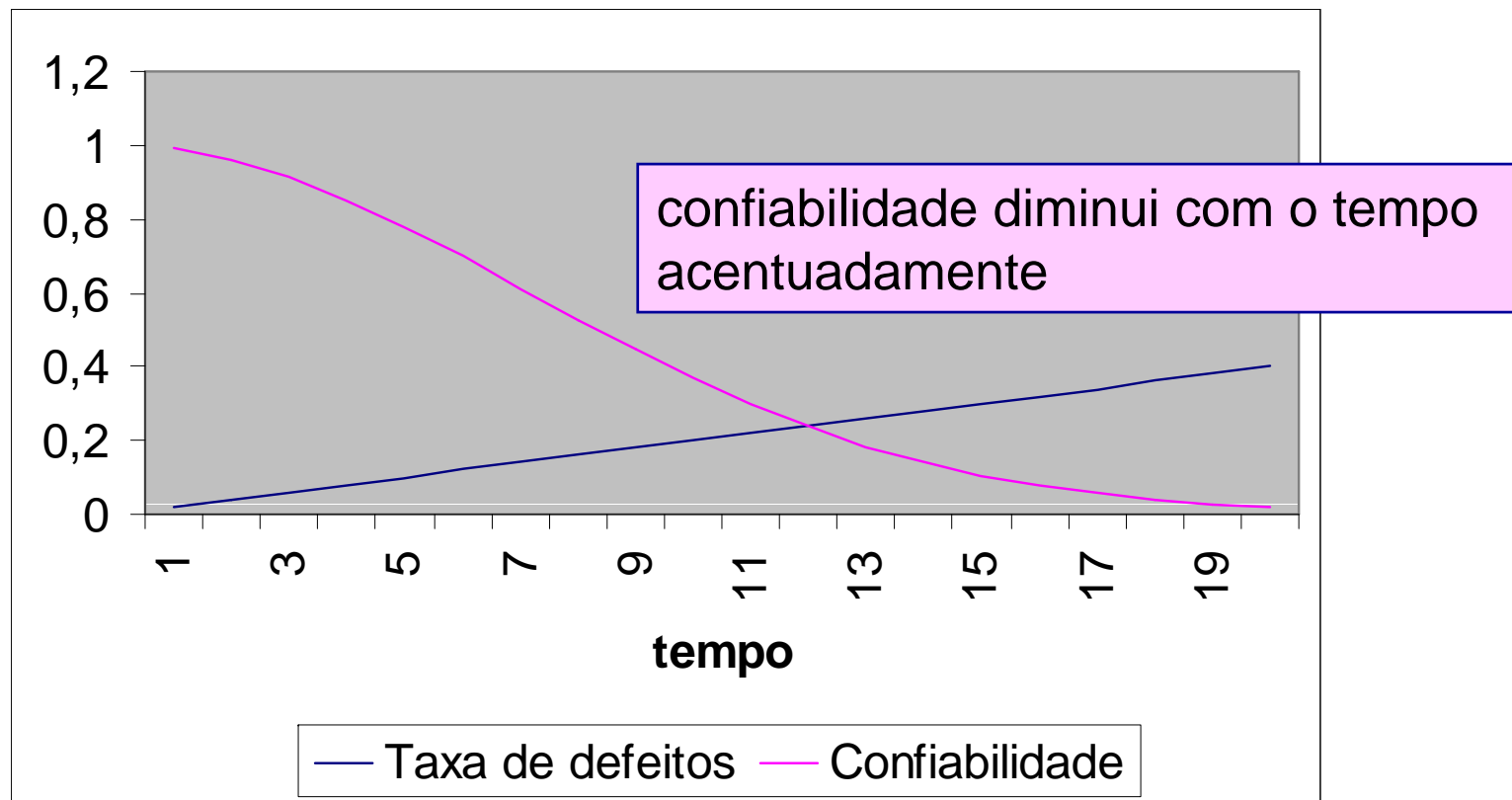
taxa de defeitos  
diminui com o tempo



# Confiabilidade

- para:  $\alpha=2$   $\lambda=0,1$

taxa de defeitos aumenta linearmente com o tempo



# Disponibilidade

---

- probabilidade do sistema estar operacional no instante  $t$  (disponível para o trabalho útil)
  - alternância entre funcionamento e reparo  
 $A(t) = R(t)$  quando reparo tende a zero

- lembrar que  $MTBF = MTTF + MTTR$ 
  - intuitivamente

$$A(t) = t_{op} / (t_{op} + t_{reparo})$$

*A(t):availability*

$t_{op}$  tempo de operação normal

$t_{reparo}$

tempo de reparo



# Disponibilidade

---

- $MTBF = MTTF + MTTR$
- $A(t) = t_{op} / (t_{op} + t_{reparo})$
- genericamente

$$A(t) = MTTF / (MTTF + MTTR)$$

nessa relação, o significado de **alta disponibilidade** fica mais claro

diminuindo o tempo médio de reparo, aumenta a disponibilidade

# Cobertura

fault coverage

- cobertura de falhas

significado intuitivo

- habilidade do sistema de realizar detecção, confinamento, localização, **recuperação** ...
- habilidade do sistema de tolerar falhas
  - geralmente se refere a habilidade de realizar **recuperação** de falhas

- significado matemático:

- probabilidade condicional que dada uma falha o sistema se recupere

extremamente difícil de calcular

# Cobertura

---

- geralmente assumido valor constante
- determinação:
  - listar falhas possíveis e falhas que o sistema pode tolerar e calcular o percentual
- usada no modelo de Markov
- muito usada também em experimentos de injeção de falhas

falhas simuladas são injetadas no sistema e se observa a reação do mecanismo de TF

relação entre falhas injetadas e falhas percebidas pelo mecanismo de TF

# Problemas com medidas

---

- defeitos são eventos aleatórios
  - podem demorar muito para ocorrer, não ocorrer ou ocorrer em um momento não apropriado
- custo de avaliação experimental é alto
  - necessária uma grande quantidade de amostras
  - necessário tempo grande de avaliação
- é importante avaliar durante o projeto do sistema
- injeção de falhas

# Bibliografia para medidas

---

- capítulo de livro
  - Johnson, Barry. An introduction to the design na analysis of the fault-tolerante systems, cap 1. Fault-Tolerant System Design. Prentice Hall, New Jersey, 1996
  - Johnson, B.W. “Fault Tolerance” *The Electrical Engineering Handbook*. Ed. Richard C. Dorf. Boca Raton: CRC Press LLC, 2000
- livro
  - Dunn, Willian R. Practical design of safety critical computer systems. Reliability Press. 2002. (cap 4 e 5).
  - Barry W. Johnson. Design and Analysis of Fault-Tolerant Digital Systems. Addison-Wesley, 1989

[http://dream.eng.uci.edu/eecs224/johnson\\_pt1.pdf](http://dream.eng.uci.edu/eecs224/johnson_pt1.pdf)  
[http://dream.eng.uci.edu/eecs224/johnson\\_pt2.pdf](http://dream.eng.uci.edu/eecs224/johnson_pt2.pdf)  
[http://dream.eng.uci.edu/eecs224/johnson\\_pt3.pdf](http://dream.eng.uci.edu/eecs224/johnson_pt3.pdf)

# Bibliografia para medidas

---

- normas
  - MIL-HDBK-217F-2, *Reliability Prediction of Electronic Equipment*, Department of Defense.
- artigos
  - J.H. Saleh, K. Marais. Highlights from the early (and pre-) history of reliability engineering. *Reliability Engineering and System Safety* 91 (2006) 249–256

# Modelagem para cálculo de confiabilidade

---

*Taisy Silva Weber*

# Modelagem

---

- Modelos de confiabilidade
- Modelos combinatórios série e paralelo
  - Aplicação a TMR
- Modelos de Markov
  - Aplicação a TMR
- Modelos para confiabilidade, safety e disponibilidade

Barry Johnson,  
cap. 1, livro-texto Pradhan96

Barry W. Johnson. **Design and Analysis of Fault-Tolerant Digital Systems**. Addison-Wesley, 1989 (cap 4)



# Cálculo de confiabilidade

---

- $R(t)$  = um dos principais atributos de dependabilidade
  - quase a totalidade das especificações determina que certos valores para  $R(t)$  devem ser alcançados e demonstrados
- estimar  $R(t)$  para sistema com mais de um componente
- usual: **métodos analíticos**
  - métodos analíticos mais comuns:
    - modelos combinatórios
    - modelos de Markov

para aplicar o método, um **modelo** do sistema deve ser criado

# Modelos combinatórios

---

- baseado em técnicas probabilísticas
  - enumerar as diferentes maneiras de um sistema permanecer operacional
  - calcular a probabilidade dos eventos que levam o sistema a permanecer operacional
  - usar a confiabilidade dos componentes individuais
- dois modelos mais comuns na prática:
  - sistema em série
  - sistema em paralelo

não confundir com **circuitos** em série e paralelo

# Modelos combinatórios: série

---

- **cada** elemento no sistema deve operar corretamente para o sistema operar corretamente
  - usado em sistemas que **não** apresentam **redundância**
- confiabilidade:
  - probabilidade que nenhum elemento apresente falha ou
  - probabilidade que todos os componentes operem corretamente
- sistema em série:

$$R_{\text{série}}(t) = \prod R_i(t)$$

# Modelos combinatórios: paralelo

---

- apenas **um** elemento no sistema precisa operar corretamente para o sistema operar corretamente
  - sistema **com redundância**
- não confiabilidade  $Q(t)$ :
  - probabilidade que todos os componentes falhem
- sistema em paralelo

$$Q_{\text{paralelo}}(t) = \prod Q_i(t)$$

$$R_{\text{paralelo}}(t) = 1 - \prod (1 - R_i(t))$$

assume-se que as falhas que afetam os componentes paralelos são randômicas e independentes

# Modelos combinatórios

---

- sistema em série

$$R_{\text{série}}(t) = \prod_{i=1}^n R_i(t)$$

todos devem operar corretamente

- sistema em paralelo

$$R_{\text{paralelo}}(t) = 1 - \prod_{i=1}^n (1 - R_i(t))$$

ao menos um deve operar corretamente

sistemas mistos são tratados com uma combinação destas fórmulas

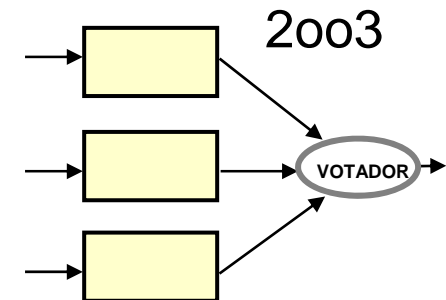
# Sistemas M-of-N

- generalização de um sistema paralelo ideal
  - $M$  de um total de  $N$  módulos idênticos devem operar corretamente
    - aparece também como MooN ( $M$ -out-of- $N$ )
  - exemplo: TMR: 2-of-3

$$R_{M\text{-of-}N}(t) = \sum_{i=0}^{N-M} \binom{N}{i} R^{N-i}(t) (1.0 - R(t))^i$$

combinação de  $N$   
elementos  $i$  a  $i$

$$\binom{N}{i} = \frac{N!}{(N-i)! i!}$$



# Desvantagens

---

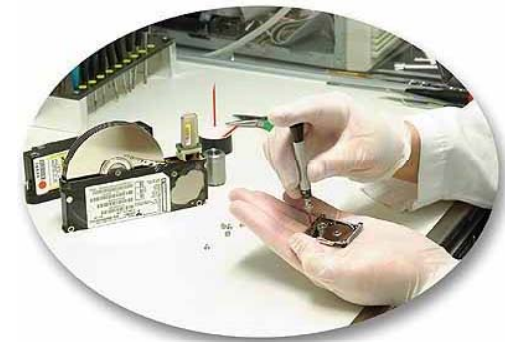
- modelos combinatórios:
  - geram equações grandes e complexas
- outras dificuldades:
  - incorporar **reparos** no modelo
  - incorporar fator de **cobertura**
    - **cobertura da detecção ou diagnóstico de falhas**
      - falhas detectadas podem ser tratadas
      - se a falha for tratada (ou o erro corrigido) o sistema pode ser recuperado ou ir para um estado seguro



# Modelos de Markov

---

- modelam sistemas complexos
  - incorporam a **cobertura** imperfeita de falhas
  - incorporam probabilidade de estados **seguros**
  - consideram **reparo**
    - podem ser colocados links entre os estados conhecendo a taxa de reparo dos componentes
- elementos básicos do modelo
  - diagramas de transição de estados
  - **estados** e **transições**



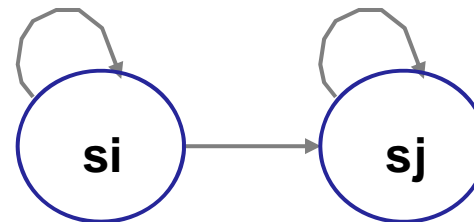
From Computer Desktop Encyclopedia  
© 2005 ActionFront Data Recovery Labs



# Markov: transição de estados

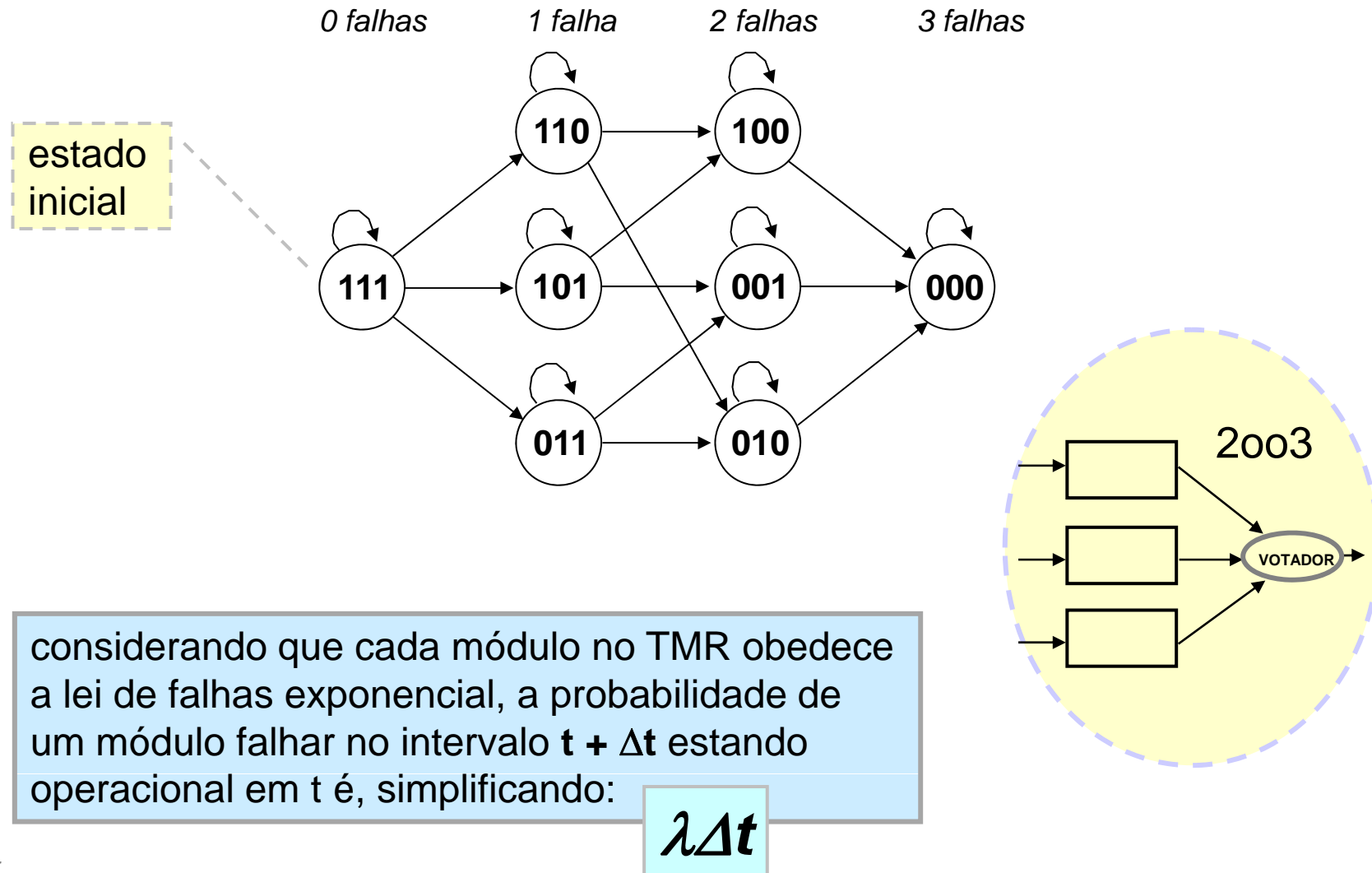
---

- **estados** = combinações de componentes operacionais e falhos

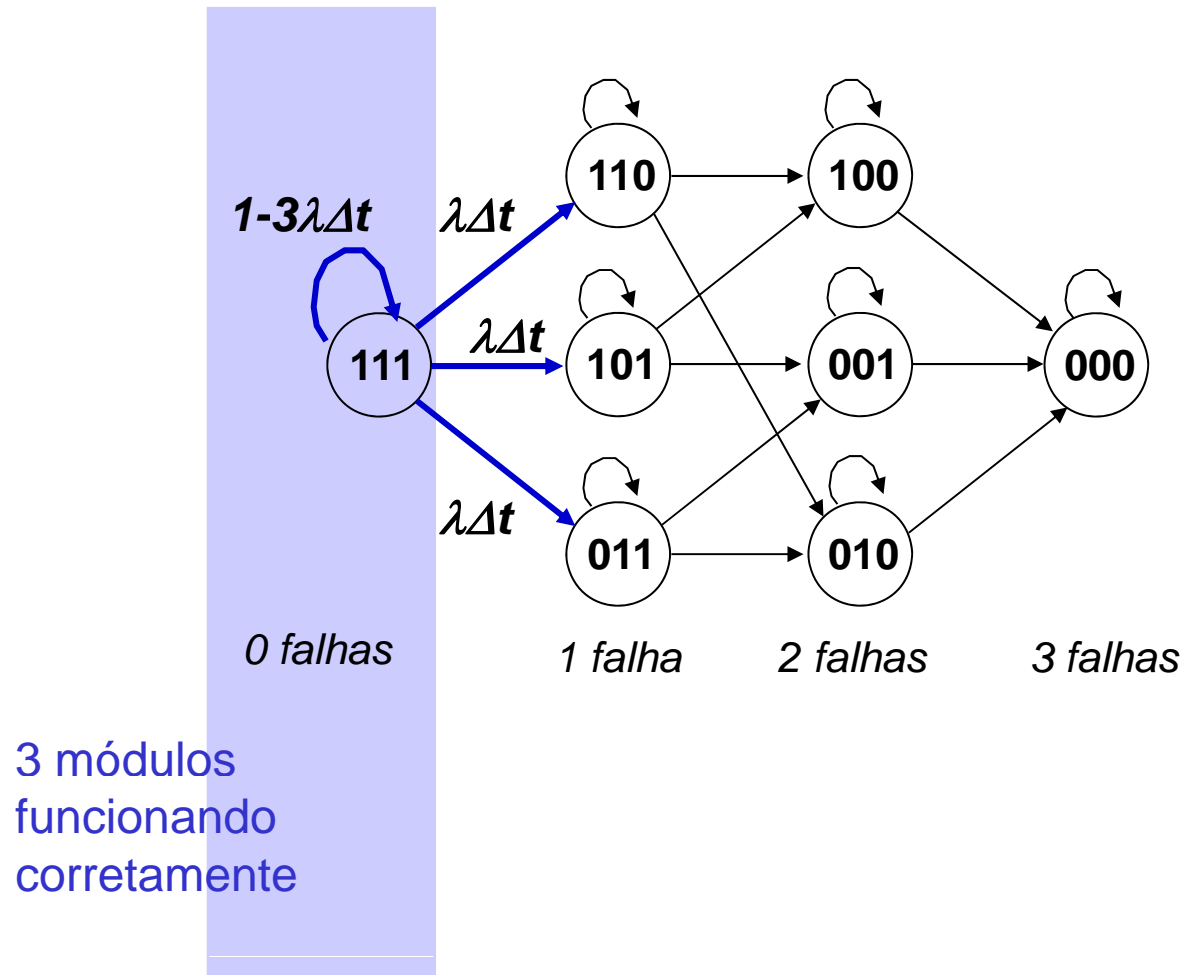


- **transições** = probabilidades para cada mudança de estado
  - probabilidade de **ocorrência de uma falha**
  - probabilidade de **cobrir uma falha**
  - probabilidade de **reparo**
- probabilidade de uma transição de **si** a **sj** é independente do método de chegada em **si**

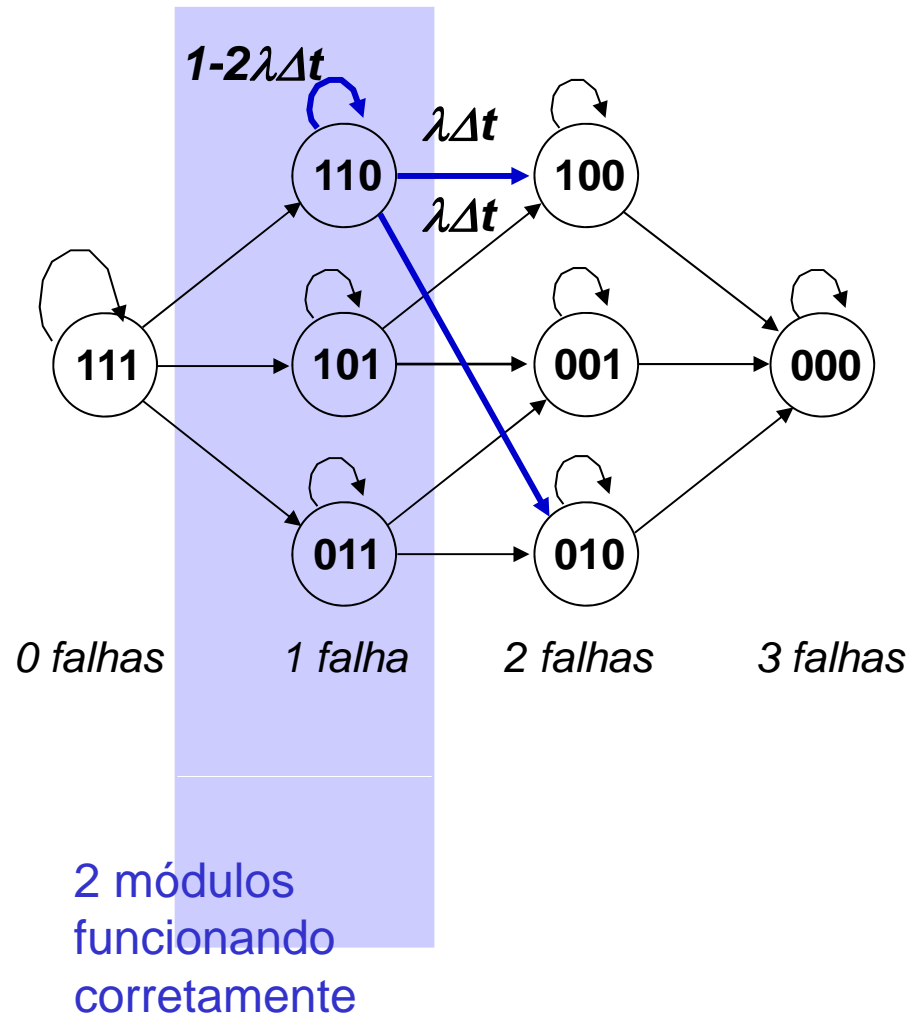
# exemplo: TMR



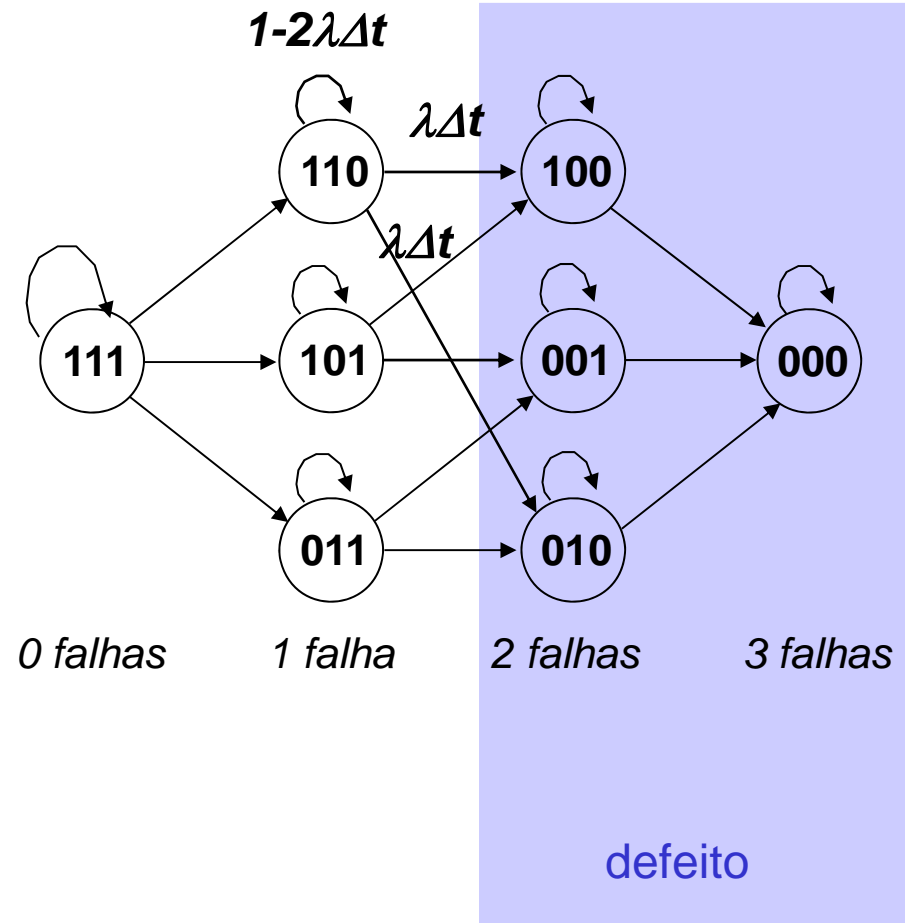
# exemplo: TMR



# exemplo: TMR

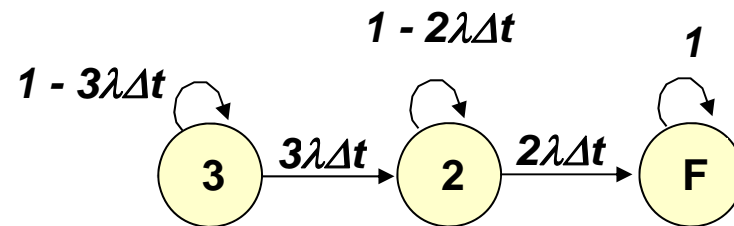
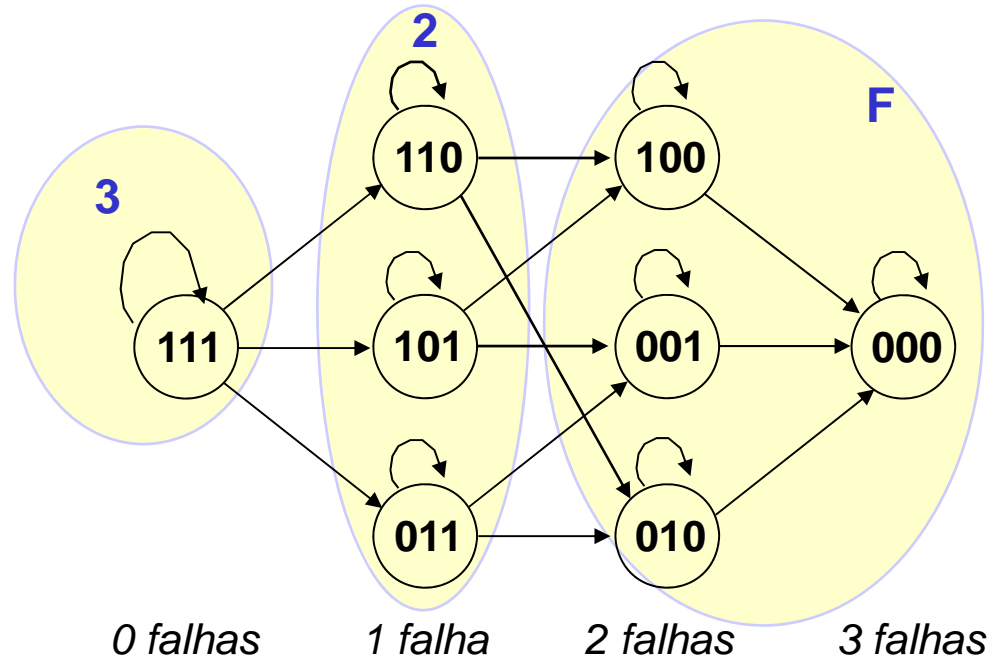


# exemplo: TMR



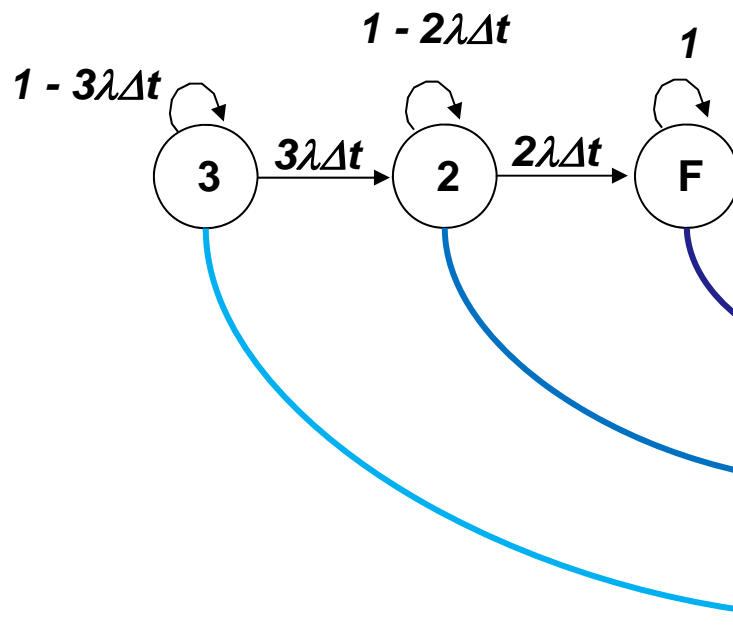
quando em  
defeito,  
qualquer  
transição  
permanece  
em defeito

# Markov reduzido



modelo de Markov reduzido  
para TMR

# Confiabilidade a partir de Markov



seja  $p_3(t)$ ,  $p_2(t)$  e  $p_F(t)$  a probabilidade de estar no estado 3, 2, F no tempo  $t$

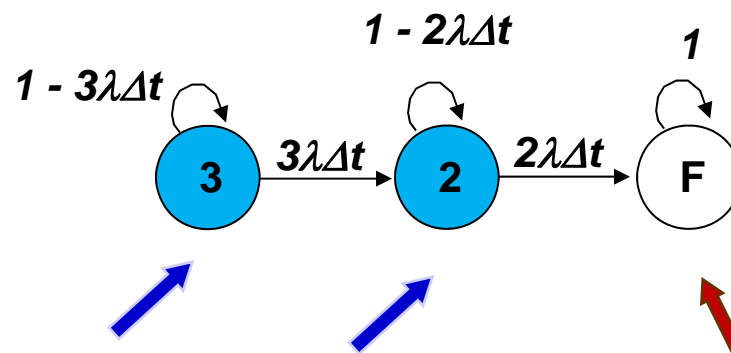
$$p_F(t + \Delta t) = 2\lambda\Delta t p_2(t) + p_F(t)$$

$$p_2(t + \Delta t) = 3\lambda\Delta t p_3(t) + (1 - 2\lambda\Delta t) p_2(t)$$

$$p_3(t + \Delta t) = (1 - 3\lambda\Delta t) p_3(t)$$

probabilidade de permanecer no estado somada a probabilidade de chegar ao estado

# Confiabilidade a partir de Markov

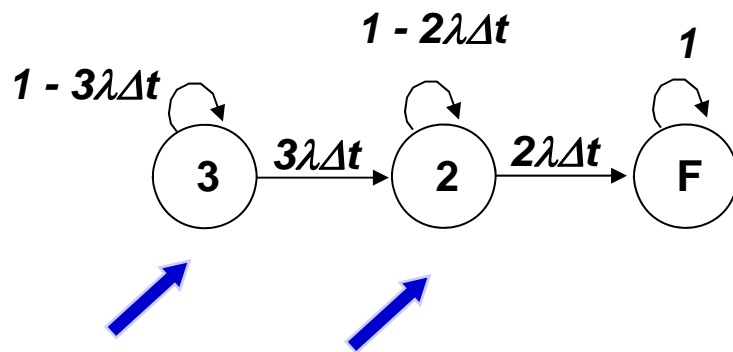


confiabilidade é a  
probabilidade de **estar no**  
estado **3** ou **2**

ou a probabilidade  
de **não** estar no  
estado **F**



# Confiabilidade do TMR



confiabilidade é a probabilidade de estar no estado 3 ou 2 ou não estar no estado F

Considerando que no limite **delta t** se aproxima de zero, podemos construir uma série de equações diferenciais.

Derivando as equações, a solução é:

$$\rightarrow p_3(t) = e^{-3\lambda t}$$

$$\rightarrow p_2(t) = 3e^{-2\lambda t} - 3e^{-3\lambda t}$$

$$p_F(t) = 1 - 3e^{-2\lambda t} + 2e^{-3\lambda t}$$

# Modelagem de safety

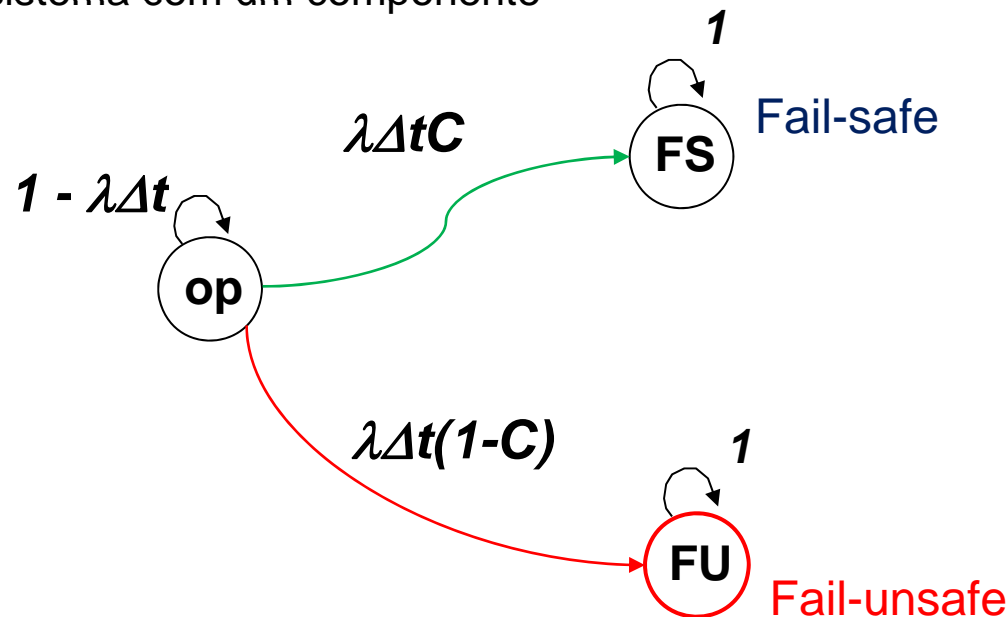
---

- importante a **cobertura de falhas** da estratégia de safety
  - Cobertura perfeita:  $C = 1.0$
  - Cobertura imperfeita:  $0 \leq C \leq 1.0$
- cobertura de falhas
  - probabilidade condicional que dada uma falha o sistema se recupere
  - probabilidade que dada uma falha o sistema **a trate corretamente**

por exemplo, indo  
para o estado seguro

## 2 transições

sistema com um componente



Cada estado operacional apresenta duas transições: uma para um estado seguro e outra para um estado inseguro.

A segurança do sistema é a probabilidade de estar em op ou FS.

→  $p_{op}(t) = e^{-\lambda t}$

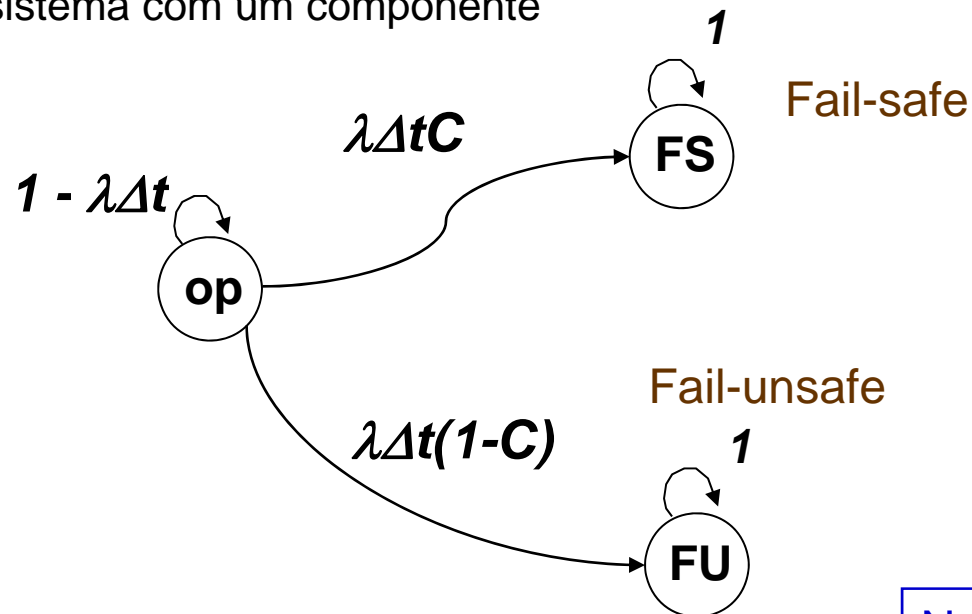
$$p_{FU}(t) = (1-C) - (1-C)e^{-\lambda t}$$

→  $p_{FS}(t) = C - Ce^{-\lambda t}$

$$S(t) = C + (1-C)e^{-\lambda t}$$

# Safety e cobertura

sistema com um componente



$$S(t) = C + (1 - C)e^{-\lambda t}$$

No tempo  $t = 0$ , safety é 1.  
No tempo infinito, safety igual a  $C$ .

$$S(\infty) = C$$

A situação estável de um sistema seguro (*safety*) depende diretamente de sua cobertura de falhas

# Modelagem de disponibilidade

---

- noção de reparo
  - a capacidade de retornar de um estado menos operacional ou com defeito para um estado mais operacional ou totalmente operacional
  - TAXA DE REPARO: número de reparos que se espera ocorrer num dado período de tempo

$$MTTF = 1/\lambda$$

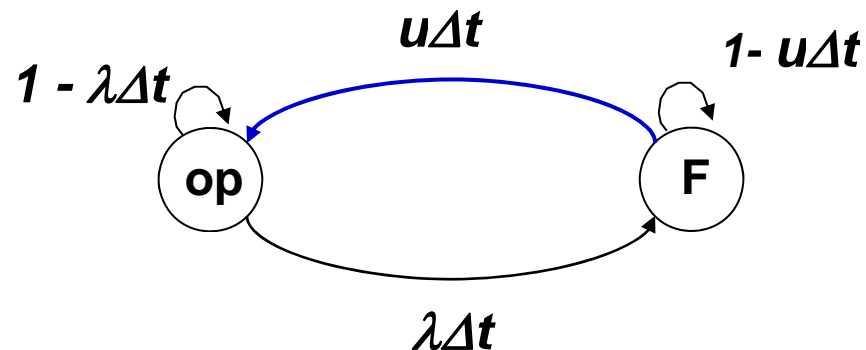
$$MTTR = \frac{1}{\mu}$$

para sistemas simplex

# Markov com reparo



o reparo leva o sistema de F (com defeito) para op (operacional)



o reparo permite uma cadeia de Markov cíclica

# Conclusão

---

- métodos analíticos
  - permitem obter medidas sobre um sistema antes de sua implementação
  - não dispensam medidas experimentais posteriores (sobre um protótipo)
  - exigem
    - domínio de probabilidade e estatística
    - ou o uso de programas apropriados que permitam modelar e estimar atributos de dependabilidade de um sistema

# Bibliografia para modelagem

---

- capítulo de livro
  - Johnson, Barry. An introduction to the design na analysis of the fault-tolerante systems, cap 1. *Fault-Tolerant System Design*. Prentice Hall, New Jersey, 1996
  - Stiffler, J.J. Reliability Estimation, cap 6. *Fault-Tolerant System Design*. Prentice Hall, New Jersey, 1996
- Barry W. Johnson. **Design and Analysis of Fault-Tolerant Digital Systems**. Addison-Wesley, 1989

[http://dream.eng.uci.edu/eecs224/johnson\\_pt1.pdf](http://dream.eng.uci.edu/eecs224/johnson_pt1.pdf)  
[http://dream.eng.uci.edu/eecs224/johnson\\_pt2.pdf](http://dream.eng.uci.edu/eecs224/johnson_pt2.pdf)  
[http://dream.eng.uci.edu/eecs224/johnson\\_pt3.pdf](http://dream.eng.uci.edu/eecs224/johnson_pt3.pdf)