Tolerância a Falhas em Sistemas Distribuídos

Prof. Raul Ceretta Nunes
Universidade Federal de Santa Maria

Porque este curso?

- Popularização de serviços Web geram dependências no cotidiano
- Falhas são inevitáveis, mas suas conseqüências devem ser minimizadas
- O domínio da área de TF auxilia administradores e desenvolvedores de sistemas a avaliar a relação custo benefício para o seu caso específico e determinar a melhor técnica para seu orçamento
  - Ex: backup consome espaço e tempo enquanto redundância de equipamentos e espelhamento de discos exige investimentos sem aumento de desempenho

Sumário

1. Terminologia e Principais Conceitos
2. Técnicas de Redundância
3. Medidas de avaliação (Taisy Weber)
4. Tolerância a Falhas em Sistemas Distribuídos
5. Aspectos de Implementação de TF em SD

Terminologia
e
Principais Conceitos

Introdução

- O que é um Sistema Tolerante a Falhas?
  - É um sistema que continua provendo corretamente os seus serviços mesmo na presença de falhas de hardware ou de software.
  - Defeitos não são visíveis para o usuário, pois o sistema detecta e mascara (ou se recupera) defeitos antes que eles alcancem os limites do sistema (ponto de fuga da especificação).
- O que é Tolerância a Falhas?
  - É um atributo que habilita o sistema para ser tolerante a falhas. É o conjunto de técnicas utilizadas para detectar, mascarar e tolerar falhas no sistema.

Observação

- A indústria não aceita bem o termo TF, preferindo os termos:
  - Sistemas redundantes (visa confiabilidade)
  - Alta disponibilidade (visa disponibilidade)
- Tentativa de unificação em *segurança de funcionamento* confundiu com aspectos de *segurança*
- Atualmente um termo mas amplo, *dependabilidade,* está se tornando popular

Terminologia Fundamental

Uma Falha resulta num Defeito

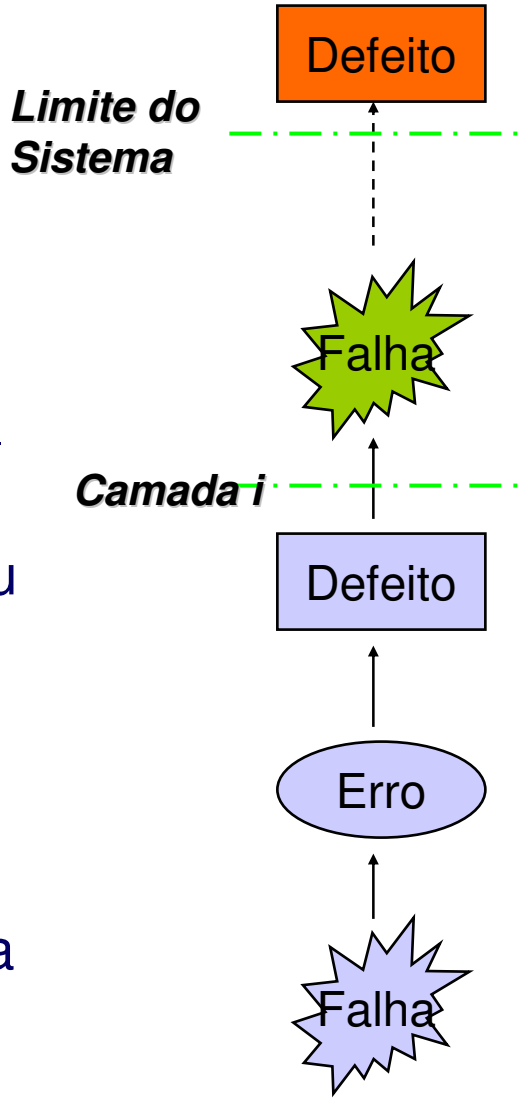
- Estado não especificado do HW ou SW

Um Erro é a manifestação de uma Falha no sistema

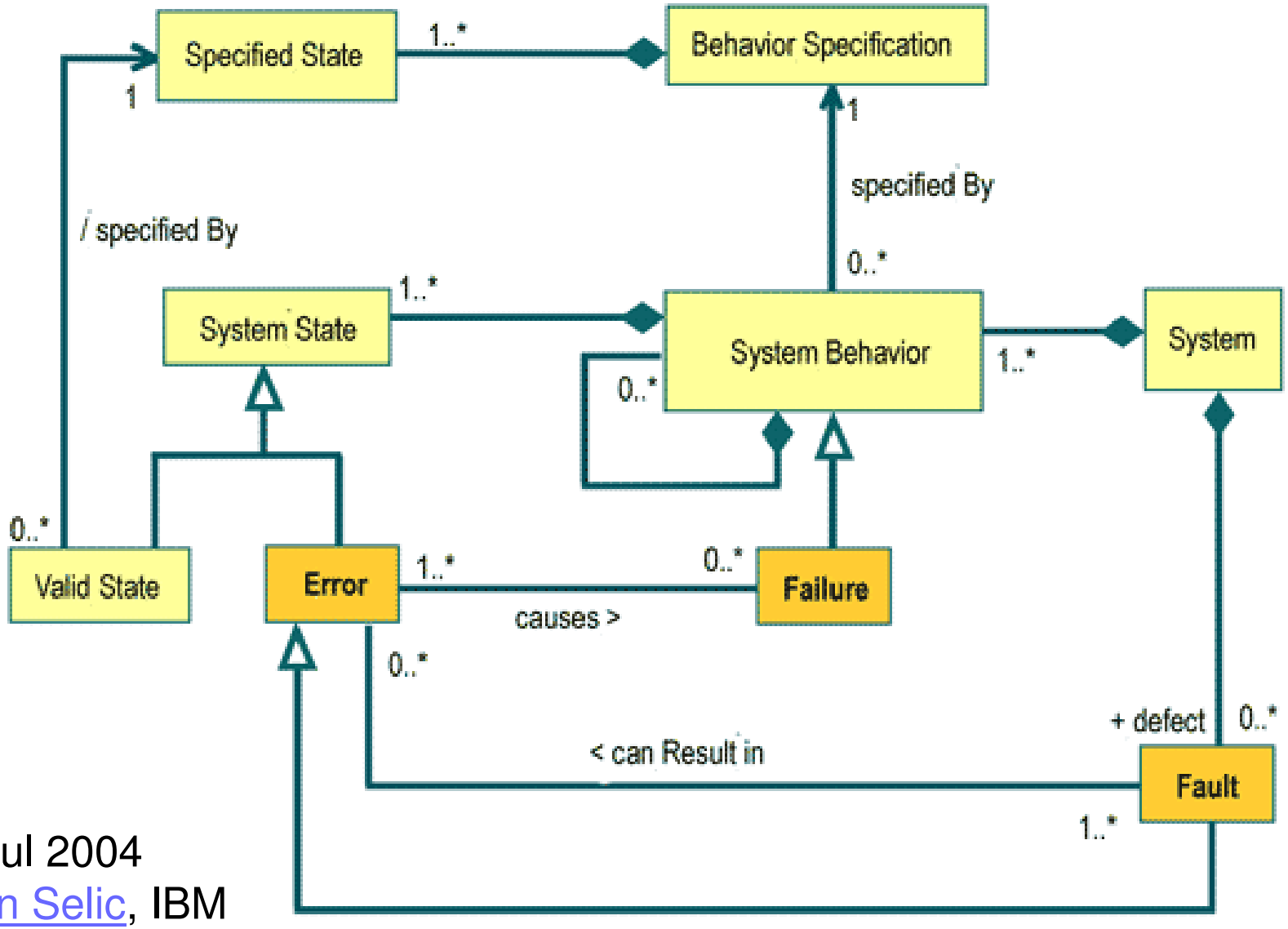
- O estado lógico do sistema difere do seu valor esperado

Um defeito é a manifestação do Erro no sistema

- O comportamento real do sistema deriva do seu comportamento esperado



Uma visão Orientada a Objetos



27 Jul 2004
Brian Selic, IBM

Terminologia Fundamental

Software TF	TF em software
software que mantém as suas especificações mesmo quando a plataforma computacional falha	técnicas usadas para mascarar falhas de software (bugs na lógica ou no projeto do SW)

Pode-se dizer o mesmo do Hardware TF em relação a TF em HW

Um breve histórico (1/4)

- **1834**: primeira afirmação sobre erros em cálculos computacionais (Dr. Lardner em Babbages's calculating engine)
- **Final 40 até meio 50**: técnicas baseadas em redundância para melhorar confiabilidade (códigos de controle de erros, duplicação com comparação, triplicação com votação, diagnóstico para localização de componentes, etc)

Um breve histórico (2/4)

- **1965**: teorias de mascaramento por redundância foram relacionadas ao conceito de *Failure Tolerance* (Pierce)
- **1967**: técnicas práticas de detecção de erros, diagnóstico de falhas e recuperação agrupadas no conceito de *sistemas tolerantes a falhas* (Avizienis)
- **1969**: conceito de cobertura de falhas, no campo da modelagem de confiabilidade (Bouricius, Carter e Schneider)

Um breve histórico (3/4)

- **1970**: criado o IEEE-CS TC on Fault-Tolerant Computing
- **1975**: conceitos sobre *software tolerante a falhas* (Randell)
- **1977**: Programação *n*-versão (Avizienis e Chen)
- **1980**: criado o IFIP WG 10.4 Dependable Computing and Fault Tolerance
- **1982**: sete *position papers* no FTCS-12 com conceitos e terminologias

Um breve histórico (4/4)

- **1985**: síntese dos conceitos e terminologias por Laprie
- **1990:** artigo sobre TFSD no FTCS
- **1992**: livro *Dependability: Basic Concepts and Terminology* (Laprie, Spring-Verlag)
- **1994**: livro Fault Tolerance in Distributed System (Jalote, Prentice Hall)
- **2000's:** integração de *security* no framework de *dependable computing*

Um breve histórico

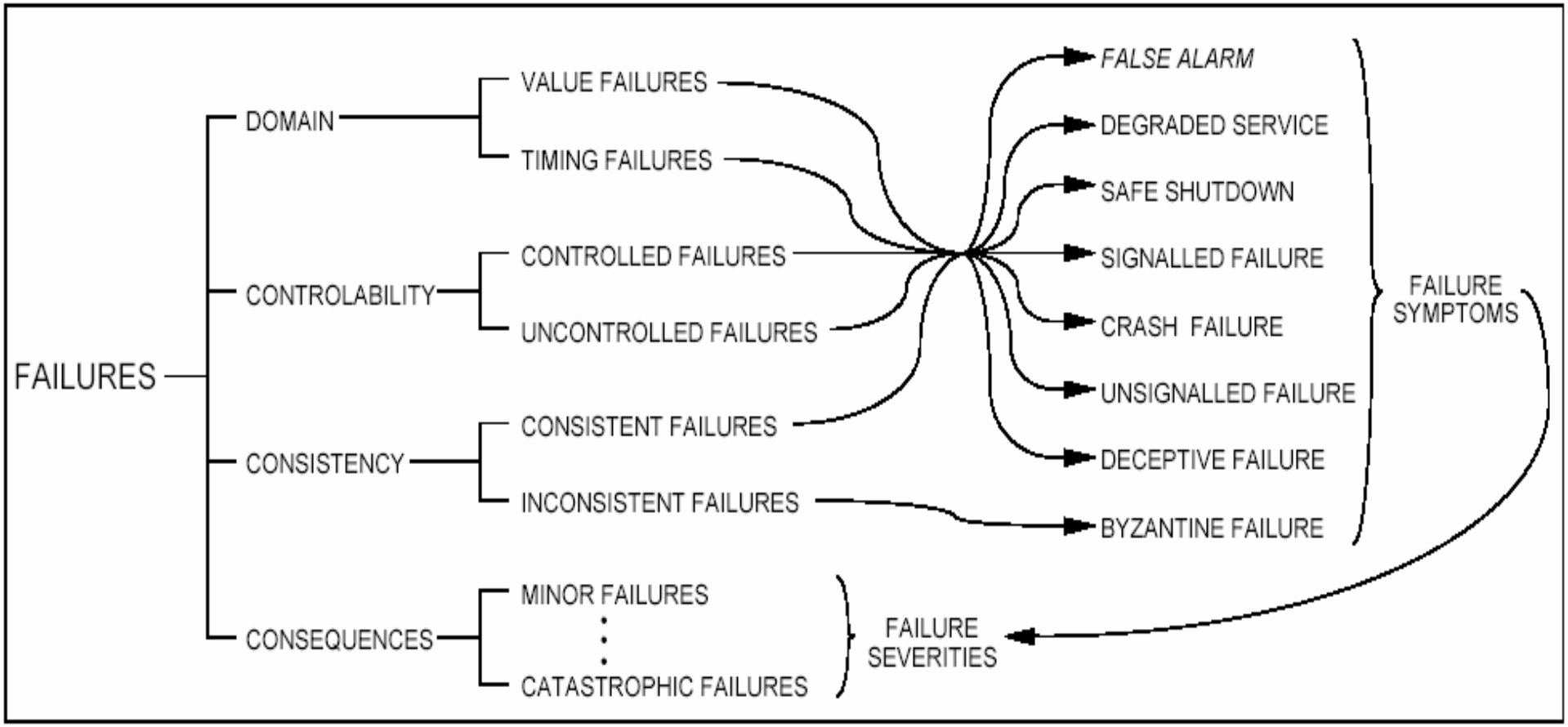
1834: primeira afirmação sobre erros em cálculos computacionais (Dr. Lardner em Babbages's calculating engine)
Final 40 até meio 50: técnicas baseadas em redundância para melhorar confiabilidade (códigos de controle de erros, duplicação com comparação, triplicação com votação, diagnóstico para localização de componentes, etc)
1965: teorias de mascaramento por redundância foram relacionadas ao conceito de *Failure Tolerance* (Pierce)
1967: técnicas práticas de detecção de erros, diagnóstico de falhas e recuperação agrupadas no conceito de *sistemas tolerantes a falhas* (Avizienis)
1969: conceito de cobertura de falhas, no campo da modelagem de confiabilidade (Bouricius, Carter e Schneider)
1970: criado o IEEE-CS TC on Fault-Tolerant Computing
1975: conceitos sobre *software tolerante a falhas* (Randell)
1977: Programação *n*-versão (Avizienis e Chen)
1980: criado o IFIP WG 10.4 Dependable Computing and Fault Tolerance
1982: sete *position papers* no FTCS-12 com conceitos e terminologias
1985: síntese dos conceitos e terminologias por Laprie
1990: artigo sobre TFSD no FTCS
1992: livro *Dependability: Basic Concepts and Terminology* (Laprie, Spring-Verlag)
1994: livro Fault Tolerance in Distributed System (Jalote, Prentice Hall)
2000's: integração de *security* no framework de *dependable computing*

Definição de Dependabilidade

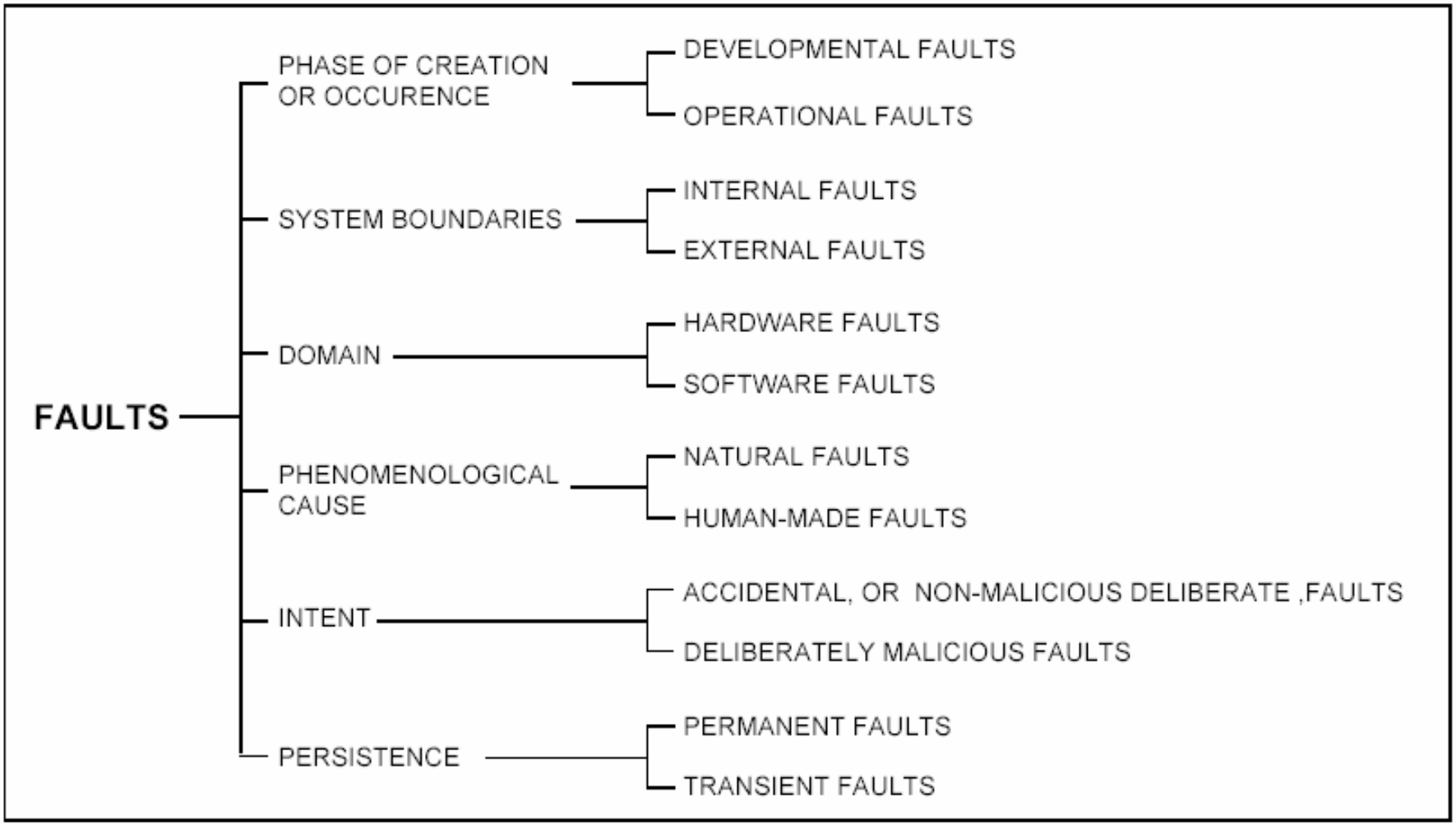
- Uma *propriedade* de um sistema computacional, tal como funcionalidade, usabilidade, desempenho e custo.

- Dependabilidade diz respeito a *habilidade* para entregar um serviço comprovadamente confiável (*trust*), ou seja, ***habilidade do sistema para evitar defeitos inaceitáveis para seus usuários***.

Modos de Defeitos (visão usuário)



Classes de Falhas Elementares



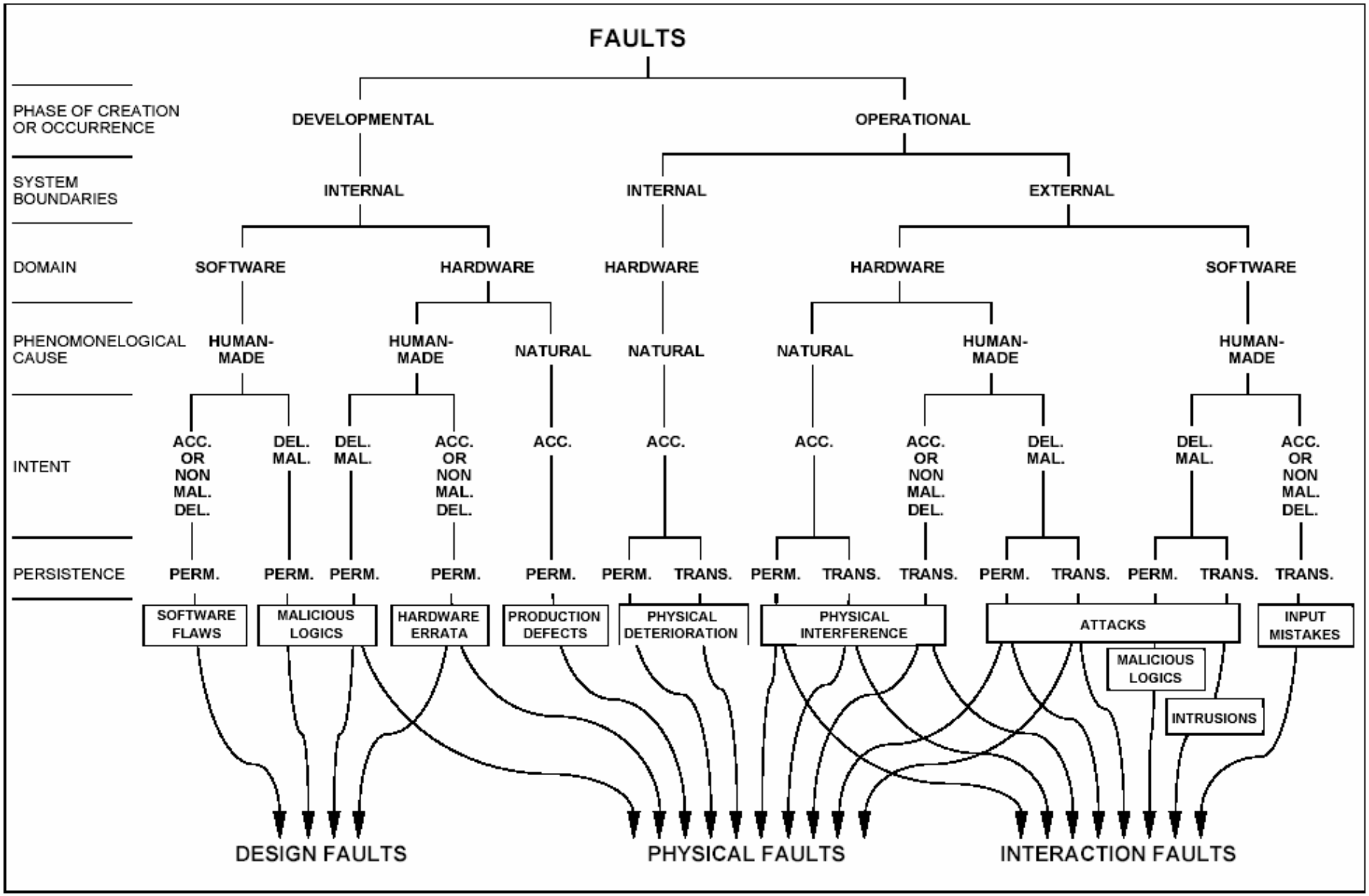
Ex: Possíveis Causas de Falhas

- Descuidos na especificação
  - especificação incorreta de algoritmos, arquiteturas ou projetos de HW e SW
- Descuidos na implementação
  - codificação equivocada ou utilização de componentes de baixa qualidade
- Defeitos de componentes
  - imperfeições na fabricação, ou defeitos randômicos
- Distúrbios externos
  - radiações, interferência eletromagnética

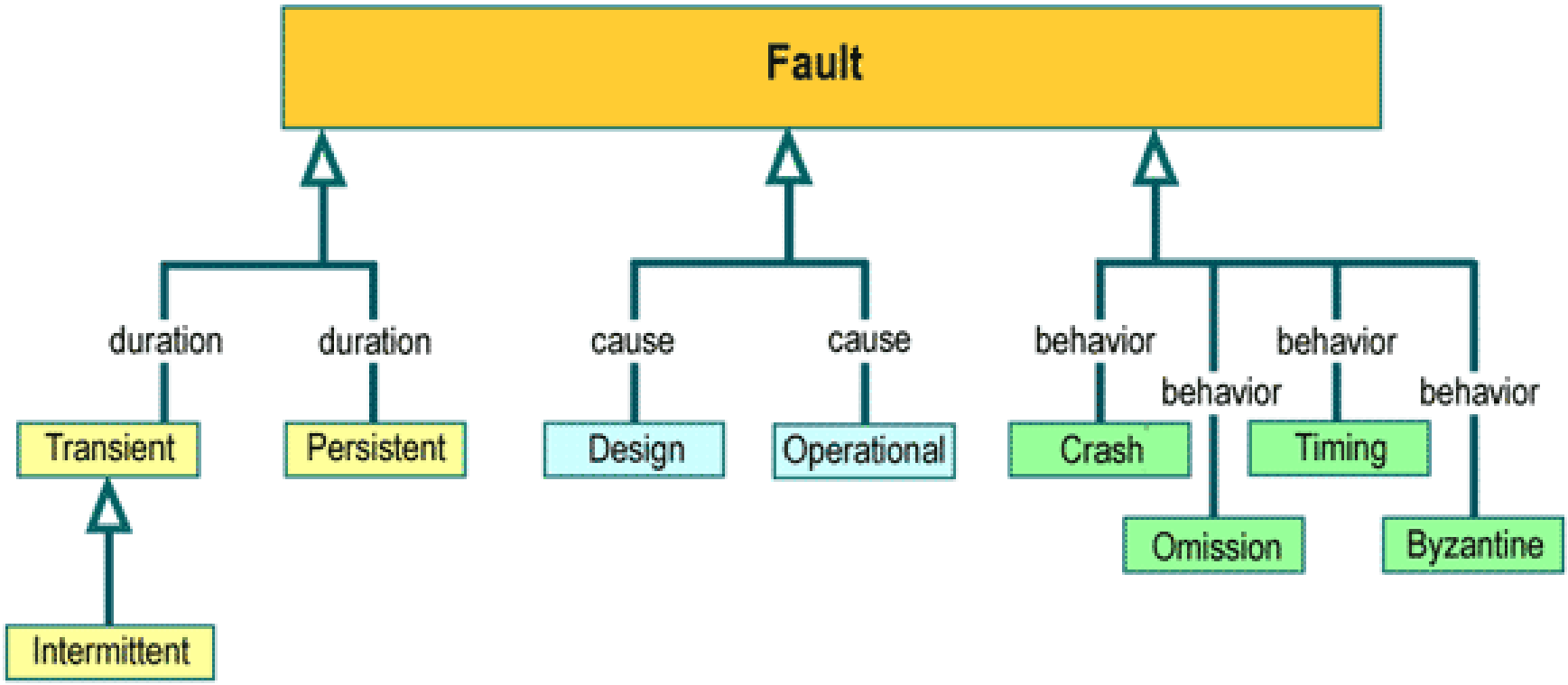
Ex: Possíveis Causas de Falhas

- Lógica maliciosa
  - falhas causadas por cavalos de tróia
  - programas ativados por tempo ou lógica (bombas lógicas)
  - falhas causadas por vírus ou worms
- Intrusão
  - exploração de falhas internas ou externas
- projeto de software mal estruturado
  - podem levar ao **envelhecimento do software** (inchaço ou esvaziamento da memória, bloqueio de arquivos, fragmentação, etc.)

Principais Classes de Falhas



Classificação simplificada em UML



27 Jul 2004
Brian Selic, IBM

Causas de Defeitos

Dependability of Computer Systems:
from Concepts to Limits
Jean-Claude Laprie - DCIA 98

Sistemas tradicionais				Redes cliente-servidor	
Não tolerante a falhas		Tolerante a falhas		(não tolerantes a falhas)	
MTBF: 6 a 12 semanas Indisponibilidade após defeito: 1 a 4 h		MTBF: 21 anos (Tandem)		Disponibilidade média: 98%	
Defeitos:		**Defeitos:**		**Defeitos:**	
hardware	50%	software	65%	projeto	60%
software	25%	operações	10%	operações	24%
operações	10%	hardware	8%	físicos	16%
comunicações / ambiente	15%	ambiente	7%		

Causas usuais de defeitos em sistemas de computação

http://www.cs.wits.ac.za/research/workshop/ifip98.html

Atributos da Dependabilidade

- **Disponibilidade** – diz respeito a média de tempo disponível para acesso
- **Confiabilidade** – diz respeito a continuidade da entrega de serviço correto
- **Integridade** – impedimento de alterações de estado impróprias
- **Segurança (*safety*)** – diz respeito a garantias de não haver defeitos catastróficos ao usuário ou ambiente
- **Confidencialidade** – impedimento de acesso indevido
- **Mantenabilidade** – habilidade para reparo e modificações eficientes
- **Segurança (*security*)** – proteção contra acessos, ou controle, não autorizados ao estado do sistema
- **Testabilidade** – facilidade para testar o sistema (ponto de teste, testes automatizados)

Meios Para obter Dependabilidade

- Prevenção de Falhas
  - Visa prevenir a ocorrência ou introdução de falhas
- Remoção de Falhas
  - Visa reduzir o número ou a severidade das falhas
- Previsão de Falhas
  - Visa estimar o número presente, a incidência futura e as conseqüências das falhas
- Tolerância a Falhas
  - visa entregar o serviço correto mesmo na presença de falhas

Prevenção de Falhas

- Aplicação de técnicas de controle de qualidade durante projeto (HW e SW)
  - Programação estruturada, OO ou OAspecto
  - Controle de informação, modularização
  - Regras de projeto rigorosas para prevenir falhas operacionais de HW
  - Definição de procedimentos para manutenção
  - Testes para prevenir falhas de interação
  - Firewalls ou similares para previnir falhas maliciosas

Remoção de Falhas

- Aplicado em 2 instantes
  - Fase de desenvolvimento
    - Verificação – estamos construindo certo o produto?
      - Estática
        - ✓ Verificação de modelos e prova de teoremas
      - Dinâmica
        - ✓ Injeção de falhas e execução simbólica (teste)
    - Diagnóstico – se não estiver certo, o que está errado?
    - Correção – corrige se os problemas
  - Fase operacional
    - Manutenção corretiva ou preventiva

Previsão de Falhas

- Para estimar o comportamento utiliza dois tipos de avaliações:
  - Qualitativa
    - identifica, classifica e elenca os modos de defeitos
  - Quantitativa
    - faz análise probabilística
- As estimativas servem para realizar ações que evitem falhas/defeitos

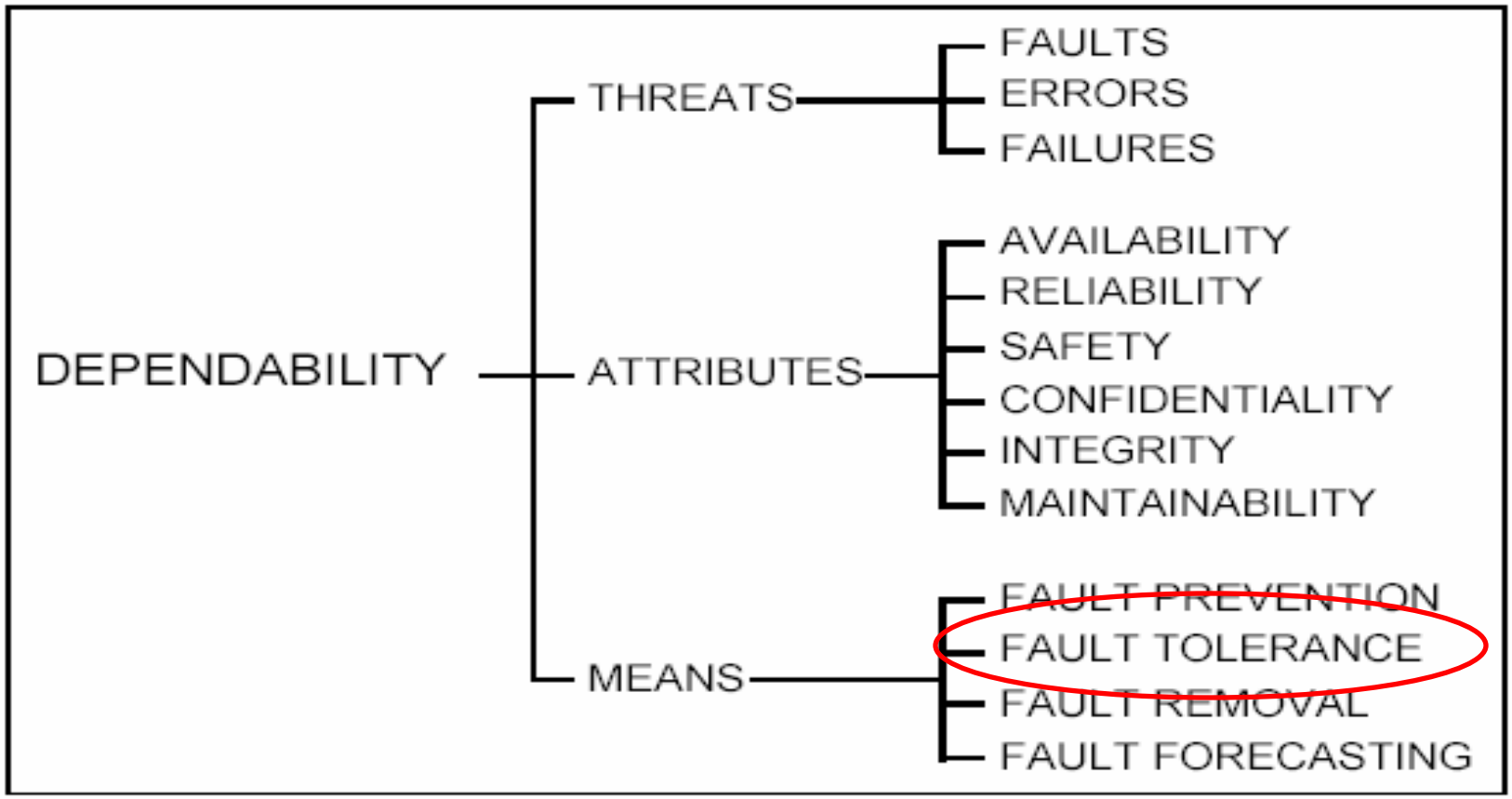
Tolerância a Falhas (1/2)

- Implementada por *mascaramento* ou *detecção de erros* seguida de *recuperação do sistema*
- Classes de detecção de erros:
  - Concorrente – execução concorrente ao serviço
  - Preemptiva – execução c/ suspensão do serviço
- Recuperação consiste reestabelecimento de estado correto
  - Implica em controle de erros e falhas

Tolerância a Falhas (2/2)

- Controle de erros
  - Rollback (checkpoint)
  - Compensação (redundância => mascaramento)
  - rollforward
- Controle de falhas
  - Diagnóstico de falhas
  - Isolamento de falhas
  - Reconfiguração do sistema
  - Reinicialização do sistema
  - Diversidade de projeto

Resumo das Definições



Tolerância a Falhas

- Requisito básico para construção:
  - REDUNDÂNCIA
- Estrutura básica composta de:
  - HARDWARE TOLERANTE A FALHAS
  - SOFTWARE TOLERANTE A FALHAS
- Nosso enfoque:
  - TÉCNICAS DE TF EM SISTEMAS DISTRIBUÍDOS

Técnicas de Redundância

Tipos de redundância

- Redundância de Hardware
  - Utiliza-se de hardware adicional
- Informação
  - Utiliza-se de bits adicionais
- Software
  - Replica componentes de software
- Temporal
  - Re-executa computações

Redundância de Hardware

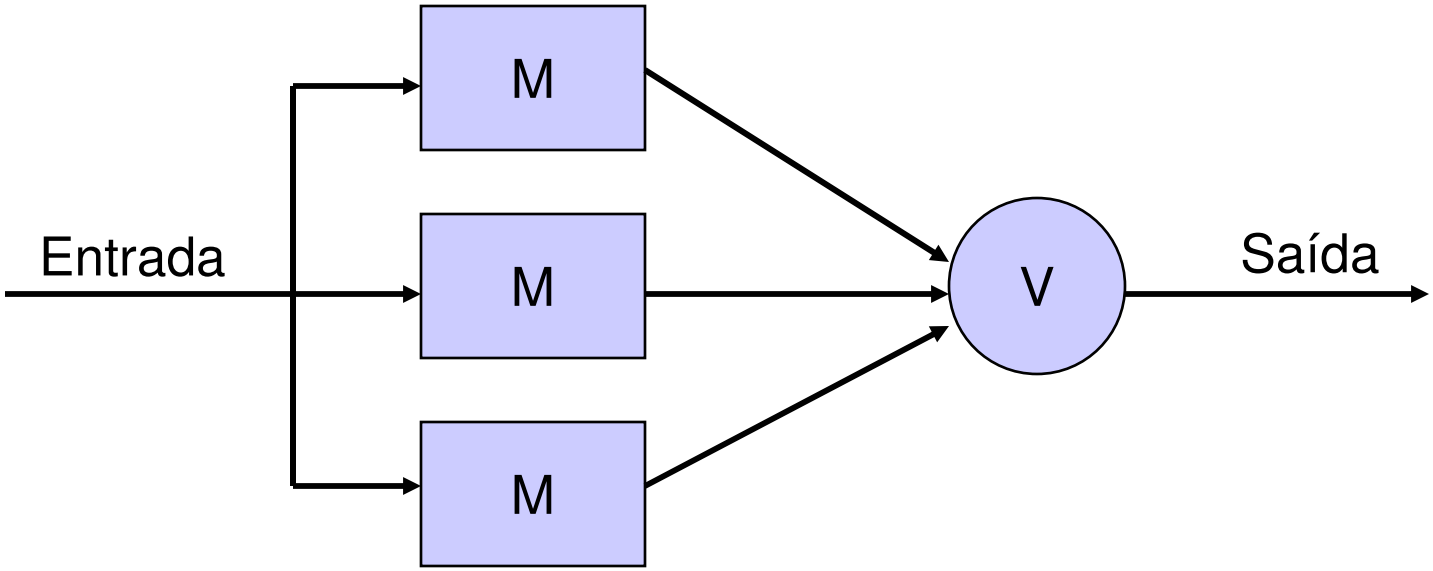
- Passiva ou estática
  - Visa mascaramento de falhas usando módulos adicionais
- Ativa ou dinâmica
  - Para obter redução de custo, baseia-se em detecção/localização, seguida de remoção e de reconfiguração/recuperação de falhas
- Híbrida
  - Visa o mascaramento de falhas, mas para obter melhor dependabilidade usa mecanismos de detecção, localização e recuperação de falhas. Removem falhas de hardware trocando componentes defeituosos por esperas
  - Mais eficiente mas cara
  - Técnica mais utilizada em aplicações críticas.

Redundância HW Passiva

- Objetivo: mascarar falhas
- Não requer operação do sistema ou operador
- Não detecta, simplesmente mascara falhas
- Costuma utilizar conceito de votação por maioria
- Técnica mais comum:
  - Redundância Modular Tripla (TMR)

Redundância Modular Tripla (TMR)

- Conceito básico: obtenção de mascaramento por triplicação do HW (processador, memória ou qualquer unidade de HW) e votação da saída.



Redundância Modular Tripla (TMR)

- Ponto crítico: votador
  - chamado *single-point-of-failure* - SPoF
- Confiabilidade (R(t)) nunca é melhor do que a confiabilidade do votador
- Generalização do TMR é o NMR (N-*modular redundance*)
  - N normalmente é um número ímpar para permitir votação por maioria.
- Limitadores: custo, potência consumida, espaço, etc

Exemplo:
Stratus ftServer W Series 6600 System

Processors

- Logical processors 2- or 4-way SMP
- Intel® XeonTM processor MP 2.0 GHz or 3.0 GHz
- Cache 1 MB iL3 or 4 MB iL3
- Front side bus 400 MHz

Memory

- Min/max memory 2 GB/24 GB DDR

Hardware Availability Options

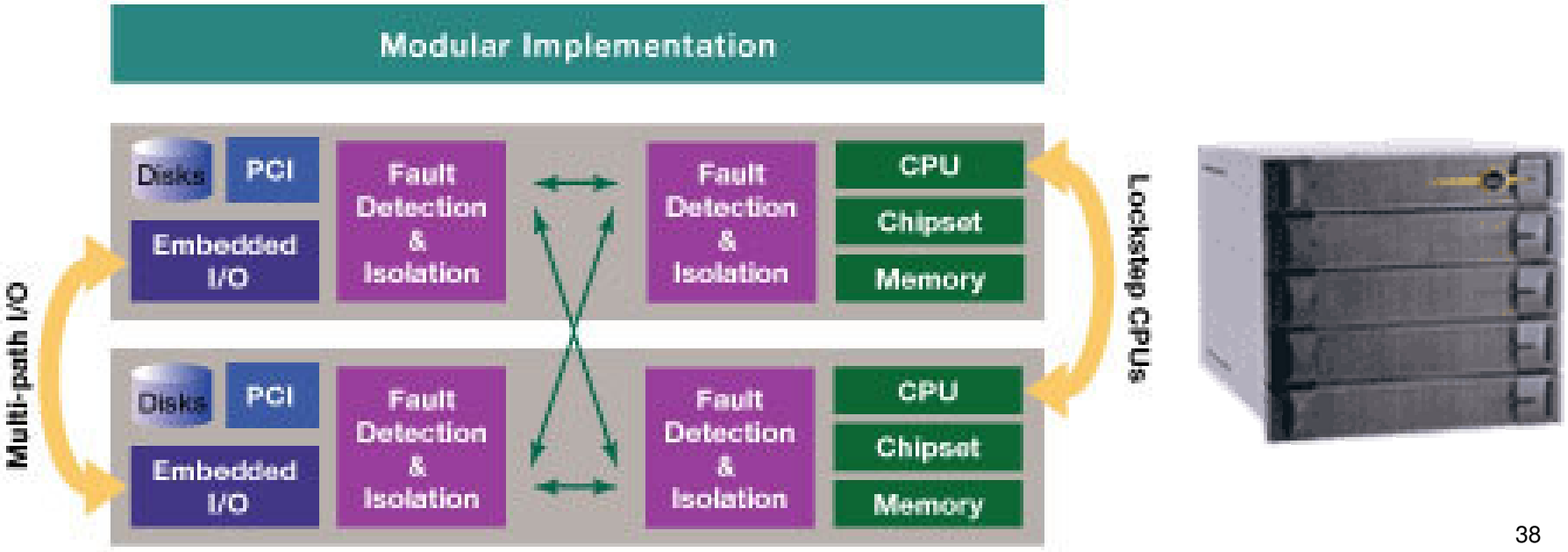
- Lockstep CPU enclosures DMR: 2 or TMR: 3

Serviceability

- Hot-Swappable Components CPU and I/O modules, disks

Operating System

- Microsoft Windows Windows Server 2003 Enterprise Edition



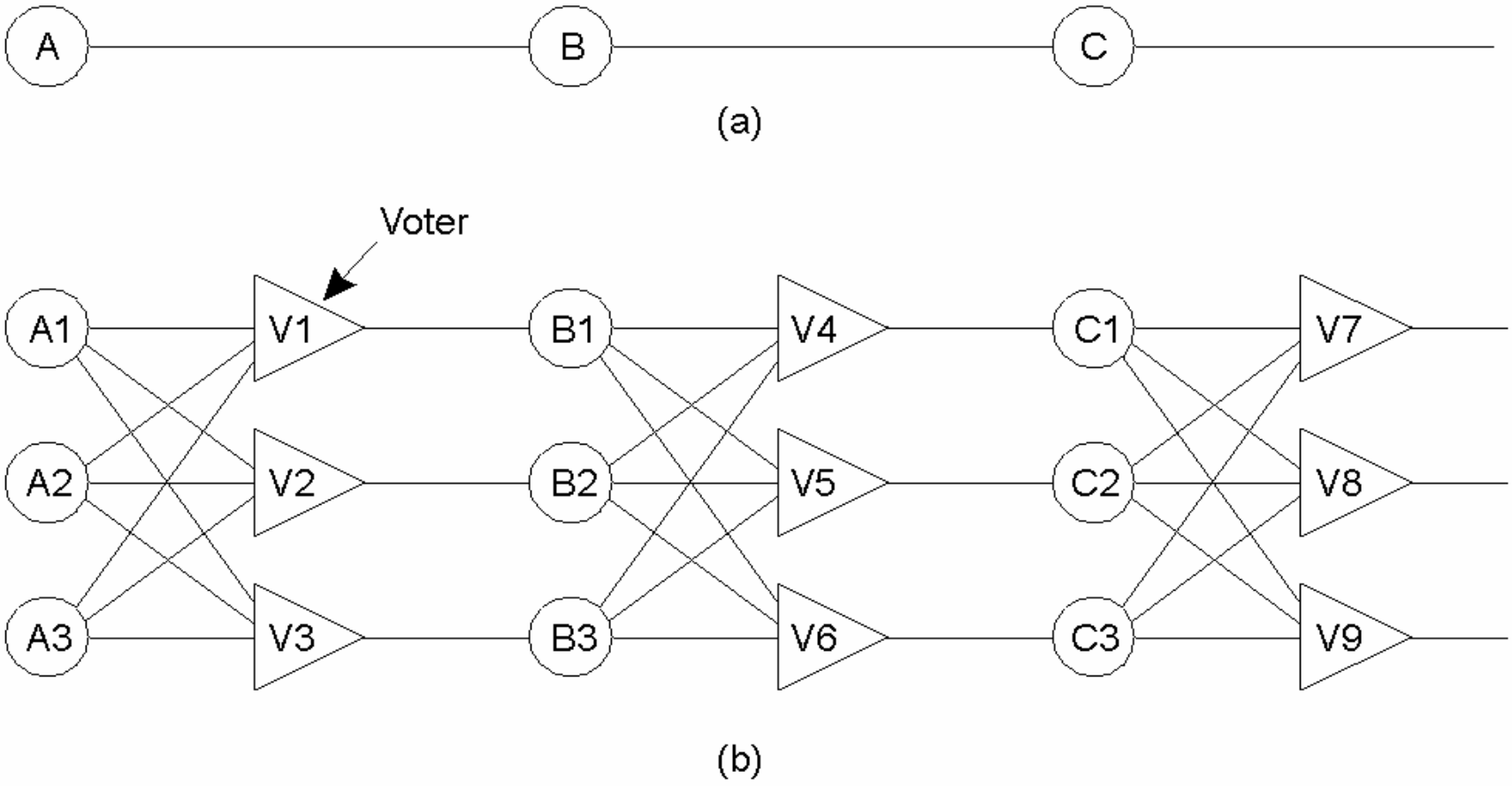
Otimizando a TMR

- Eliminação do SPoF: TMR com votador triplicado
  - Triplicação do votador permite recuperar erros, logo é chamado de *restoring organ*.
    - Ex: processadores do Tandem (pg 244 do Pradhan)
  - Pode se usar hierarquia de votadores para evitar a propagação de falhas.
    - Ex: em medições de sensores com cálculo de temperatura a votação antes do cálculo evita fazer o cálculo três vezes
- Aumento da confiabilidade do votador: em HW, circuito auto-testável

Otimizando a TMR

- Tipo de votador: implementado em HW ou SW?
  - SW- vantagem de eliminar HW adicional e usar poder de processamento disponível; facilidade de alteração da maneira de votação; desvantagem é o tempo de votação, pois HW dedicado é mais rápido
  - Decisão depende de:
    - disponibilidade para processador realizar a votação;
    - velocidade que votação necessita ser resolvida;
    - limitações de espaço, consumo e peso;
    - número de votadores diferentes que precisam ser implementados
    - flexibilidade requerida para projetos futuros.
- Tipo de votação: votação de valores não exatos: técnica de seleção de valor médio

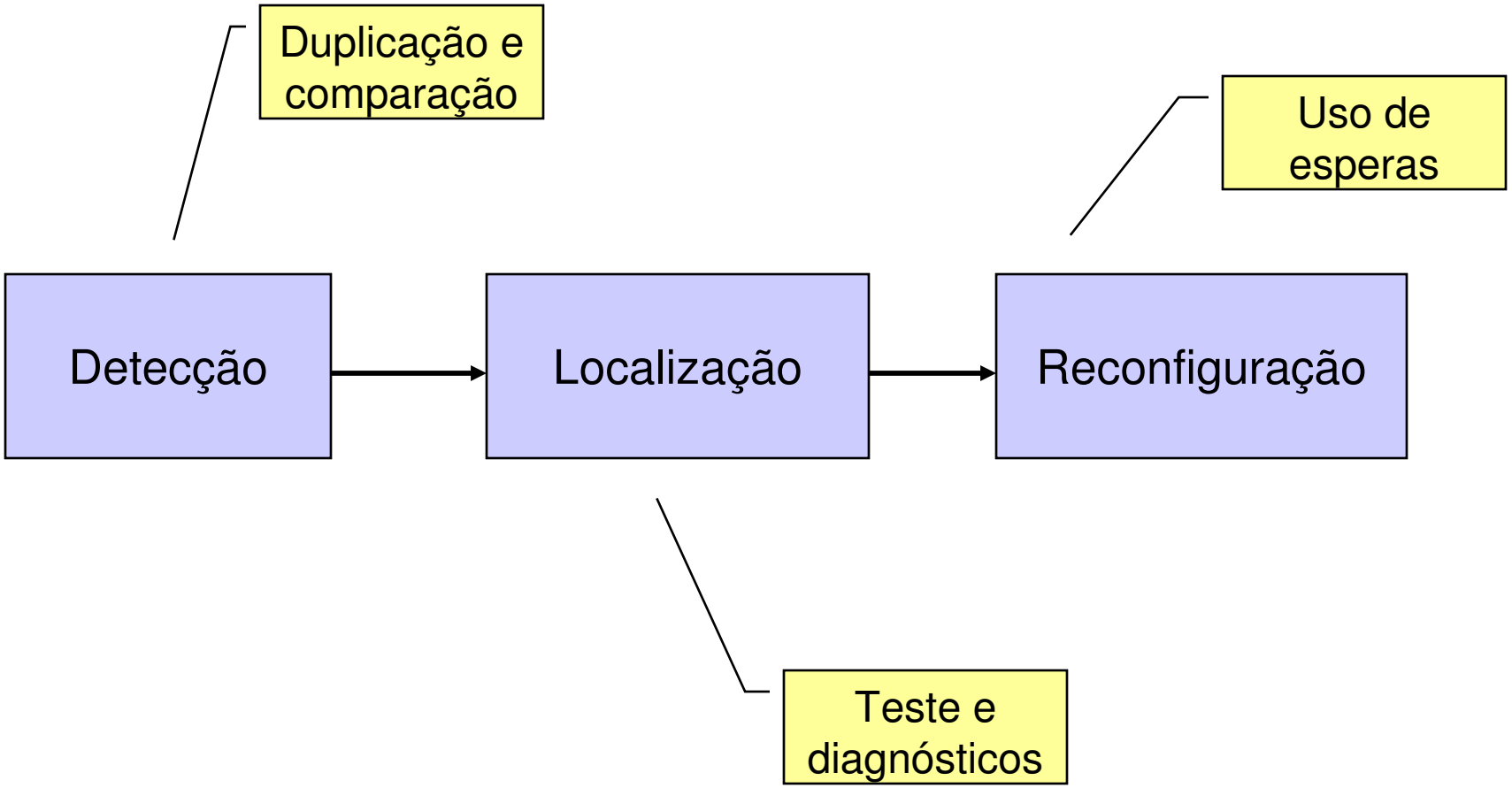
Exemplo



Redundância HW Ativa

- Conceito básico: replicação de módulos, com apenas um ativo (operando)
- Ponto crítico
  - Detectar que módulo ativo falhou
- Funcionamento
  - Detecta a presença de falhas e resolve alguma ação para eliminá-las
- Requer reconfiguração para tolerar falhas
- Requer componentes *standby*

Redundância HW Ativa



Tipos de espera (standby)

- Cold standby
  - Esperas desligados, ativação do zero
  - Não causa overhead durante operação
- Warm standby
  - Esperas desligados, ativação do último ponto de verificação (*checkpoint*)
- Hot standby
  - Esperas ligados, ativação do estado atual
  - Também chamado sistema *duplex*

Redundância HW Híbrida

- Realiza mascaramento de falhas
- Usa técnicas de detecção, localização e recuperação de falhas para melhorar a tolerância a falhas
- Remove falhas de hardware trocando componentes defeituosos por esperas
- É mais eficiente mas de alto custo;
- É a técnica mais utilizada em aplicações críticas.

Redundância de Informação

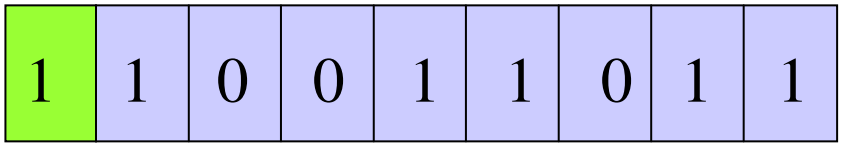
- Visa detectar erros ou mascarar falhas incluindo bits ou sinais extras à informação.
- Exemplos:
  - Paridade
  - Checksums
  - Duplicação de código
  - Códigos cíclicos
  - Códigos de correcção de erros

Código de Paridade

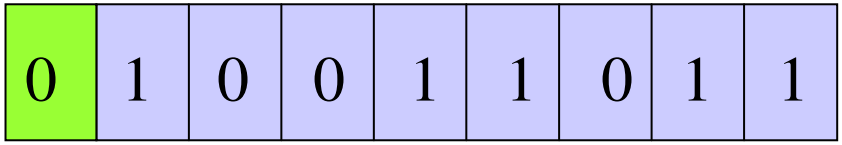
- Conceito básico: adicionar bit(s) para manter a palavra de código com um número par ou ímpar de 1s.
- Principal uso: detecção de erros no armazenamento de memória.
- 6 abordagens:
  - Paridade por palavra
  - Paridade por byte
  - Paridade por chip
  - Paridade por múltiplos chips
  - Paridade entrelaçada
  - Paridade transpassada (*overlapping*)

Paridade por palavra

- Paridade par

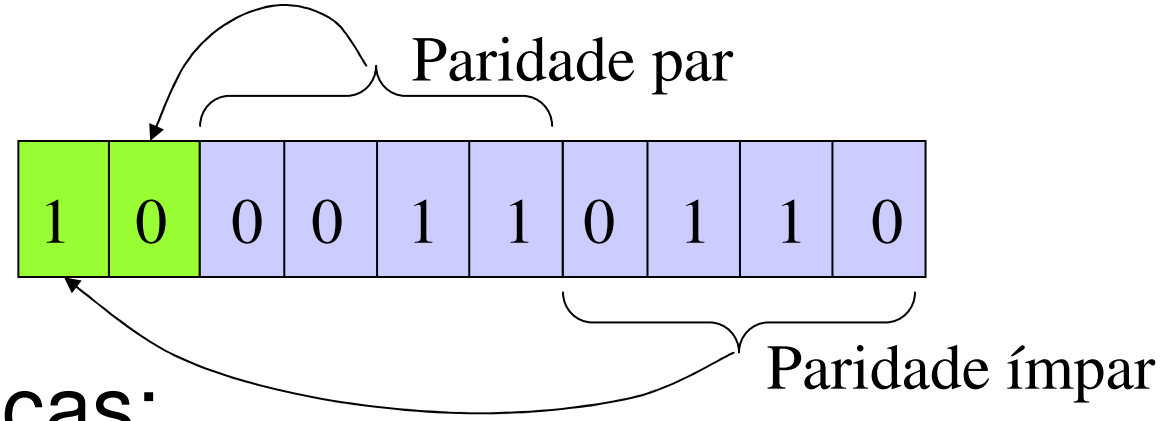


- Paridade ímpar



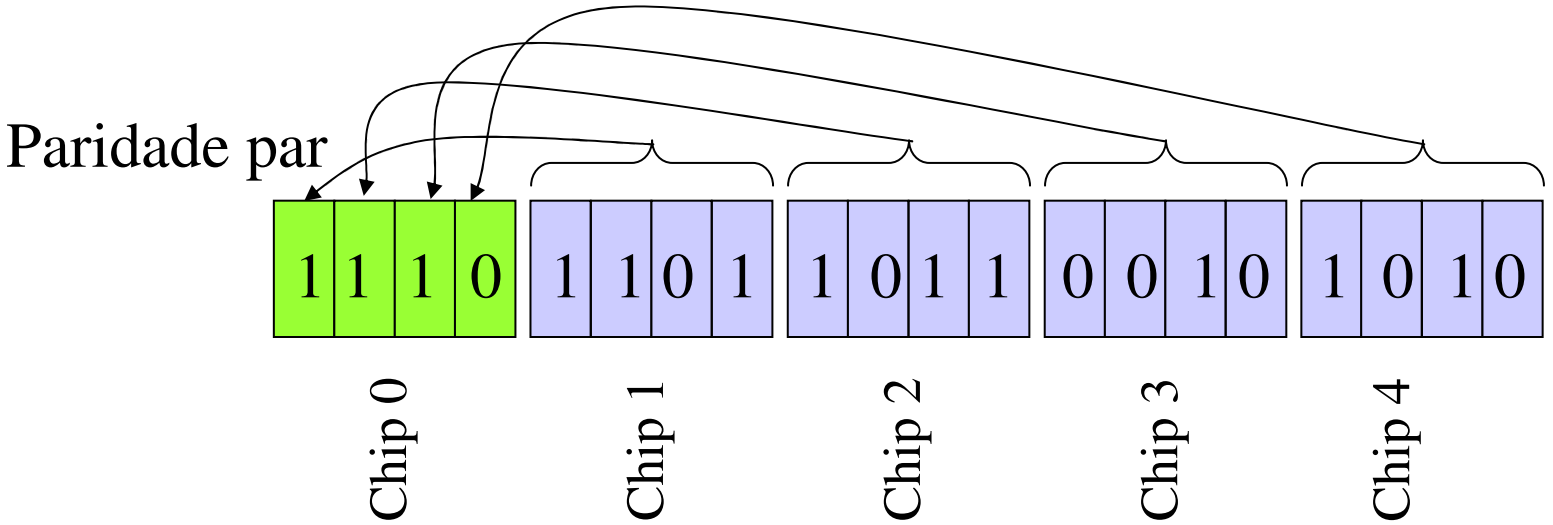
- Características:
  - Permite apenas detecção de erros simples
  - Não permite detecção de muitos erros múltiplos
  - Forma um código separável

Paridade por byte



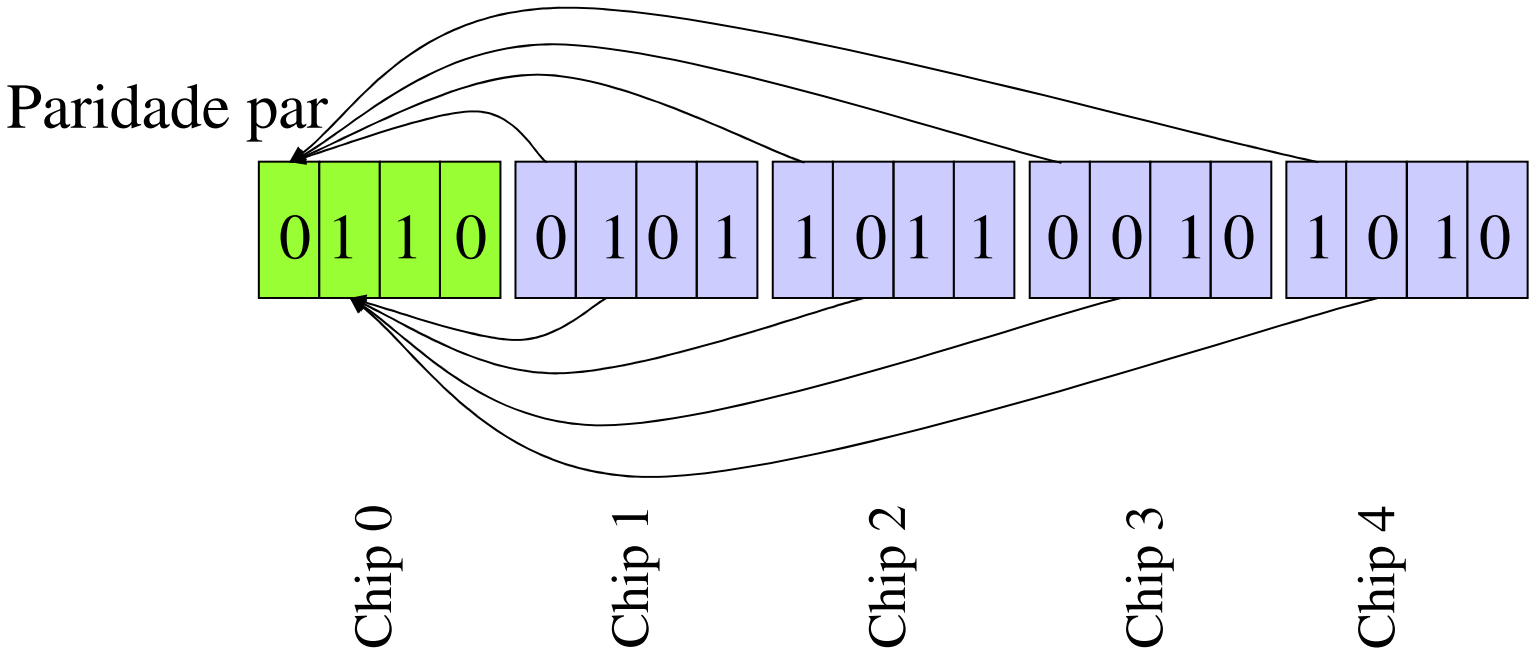
- Características:
  - Paridade por grupo de bits
  - Permite detecção de erros “tudo 1” ou “tudo 0”
  - Permite detecção de erros múltiplos, se erros forem em grupos distintos
  - Forma um código separável

Paridade por chip



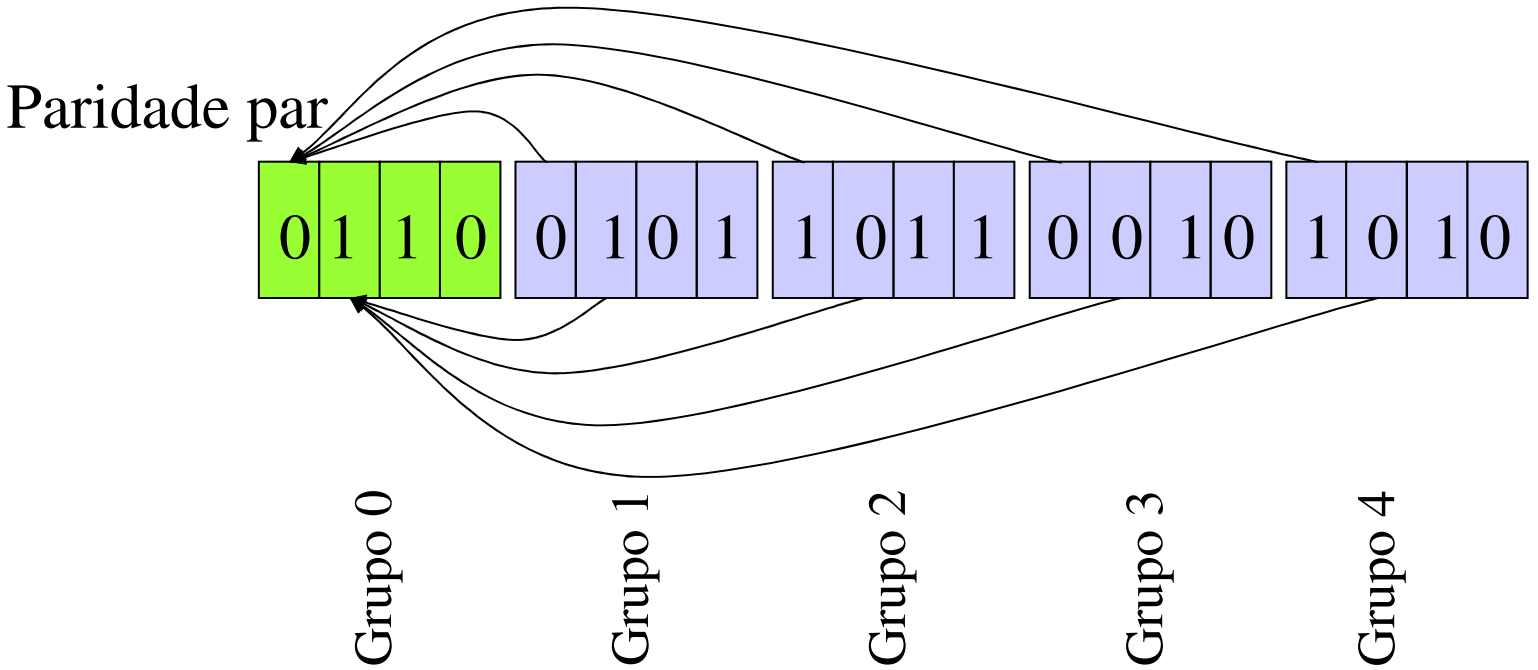
- Características:
  - Permite detecção e localização de erros
  - Permite detecção de erros múltiplos, se erros forem em chips diferentes

Paridade por múltiplos chips



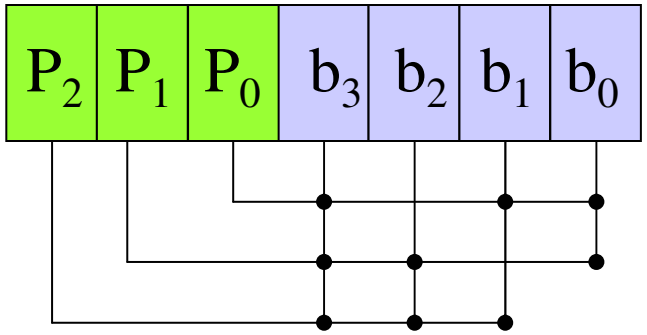
- Características:
  - Permite detecção de erros múltiplos (falhas de chips inteiros, erros consecutivos)

Paridade entrelaçada



- Características:
  - Permite detecção de erros múltiplos consecutivos

Paridade transpassada

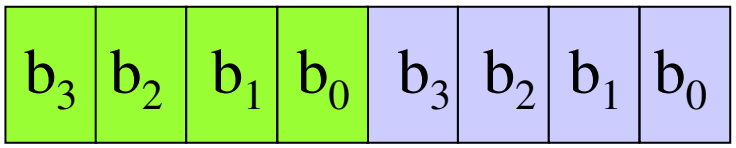


- Características:
  - Cada bit aparece em mais de um grupo de paridade
  - Erros podem ser detectados, localizados e corrigidos por complementação simples, se desejado
  - Conceito básico dos códigos de correção de erros de Hamming

Bit Err	Paridade Afetada		
b_3	P_2	P_1	P_0
b_2	P_2	P_1	
b_1	P_2		P_0
b_0		P_1	P_0
P_2	P_2		
P_1		P_1	
P_0			P_0

Duplicação de código

- Características:
  - 100% de redundância, logo caro
  - Exemplos:
    - retransmitir dado e comparar os dois recebidos
    - Escrever em duas posições da memória



Checksums

- Adiciona informação a um bloco de informação para possibilitar detecção de erros
- 4 tipos:
  - Precisão simples
  - Precisão dupla
  - Honeywell
  - Checksum residual

Checksums de precisão simples/dupla

- Precisão simples
  - realiza somatório do código e ignora bits que extrapolam tamanho da palavra (n).
  - Usa módulo n.
- Precisão dupla
  - realiza somatório e utiliza uma palavra de precisão dupla para armazenar resultado.
  - Ignora bits que extrapolam tamanho da palavra dupla.
  - Usa módulo 2^n.

Checksum Honeywell

- A idéia é agrupar os dados em palavras duplas (palavra formada por duas palavras de dados) e computar o checksum.
- Facilita a detecção de erros e “porções” dos dados, pois o erro tende a aparecer em duas posições.

Checksum residual

- Similar ao de precisão simples, mas soma o excedente junto.

Redundância de Software

- Formas em que aparece:
  - Linhas extras de código usadas para verificar a magnitude de sinais
  - Pequenas rotinas utilizadas para, periodicamente, testar a memória
  - Componentes de software
- Principais técnicas:
  - Verificação de consistência
  - Verificação de capacidade
  - Métodos de replicação de software (software TF)
    - Programação N-auto-verificável
    - Programação N-versão
    - Blocos de recuperação

Verificação de Consistência

- Usa conhecimento prévio sobre características da informação para verificar a corretude da informação
- Exemplos:
  - Verificação de excesso do limite (magnitude do valor ou intervalo da medida)
  - Verificação de endereço é válido
  - Verificação de códigos de instrução
  - Verificação de derivações excessivas em relação ao previsto (padrão de comportamento pré estabelecido)

Verificação de Capacidade

- Usa conhecimento prévio sobre características do sistema
- Exemplo:
  - Verificação se há memória suficiente
  - Verificação se todos processadores estão funcionando/acessíveis
  - Se unidades funcionais estão funcionando

Programação N-auto-verificável

- N versões do programa são escritas e cada uma realiza um teste de aceitação, permitindo suportar N 1falhas
- Testes de aceitação são testes realizados sobre os resultados produzidos por um programa e podem ser criados por verificação de consistência ou capacidade
- Falhas de software normalmente são geradas por erros de projeto ou de codificação
- Duplicação e comparação não detecta falhas de software, a menos que o software seja feito por equipes independentes
- Técnica análoga a técnica de hot standby sparing
- O processo de reconfiguração deve ser rápido

Programação N-versão

- Criada para tolerar falhas de projeto
- Conceito básico:
  - projetar e codificar o software N vezes e votar o resultado
- Cada módulo deve ser desenvolvido por equipe independente
- A técnica pode tolerar (N − 1)/2 falhas
- Dificuldades:
  - Desenvolvedores costumam ter as mesmas práticas de programação, não garantindo a independência das versões
  - Como especificação é a mesma, a técnica não tolera erros na especificação

Blocos de Recuperação

- Análoga a redundância de HW ativa (cold standby sparing)
- N versões, mas com um único teste de aceitação
- Uma versão é a primária e outras são secundárias
- Assumindo cobertura perfeita e falhas independentes, a técnica suporta N-1 falhas

Redundância Temporal

- Uma dada função é executada múltiplas vezes, com as mesmas entradas.
- Eventuais diferenças nas saídas indicam erros causados por defeitos físicos transientes (ou por ruído).
- **Garantindo** tempo para **as duas execuções** da tarefa em todas as respectivas ativações (inclusive no pior caso), pode-se conseguir uma **taxa de detecção de erros** acima de **99,9%.**

Número de Réplicas

- O **número de réplicas** por componente (incluíndo o próprio) depende do **número (k) de falhas** que se pretende tolerar e do respectivo **tipo**:
- • **k+1** componentes para falhas do tipo **falha-silêncio (fail-silent)**
- • **2*k+1** componentes para falhas do tipo **falha-consistente**
- • **3*k+1** componentes para falhas **maliciosas**.

Medidas de Avaliação

(slides da Prof. Taisy Weber – UFRGS)

Tolerância a Falhas em Sistemas Distribuídos

TF e Sistemas Distribuídos

Sistemas Distribuídos
deveriam ser Tolerantes a Falhas

E

Sistemas Tolerantes a Falhas
deveriam ser Distribuídos

TF e Sistemas Distribuídos

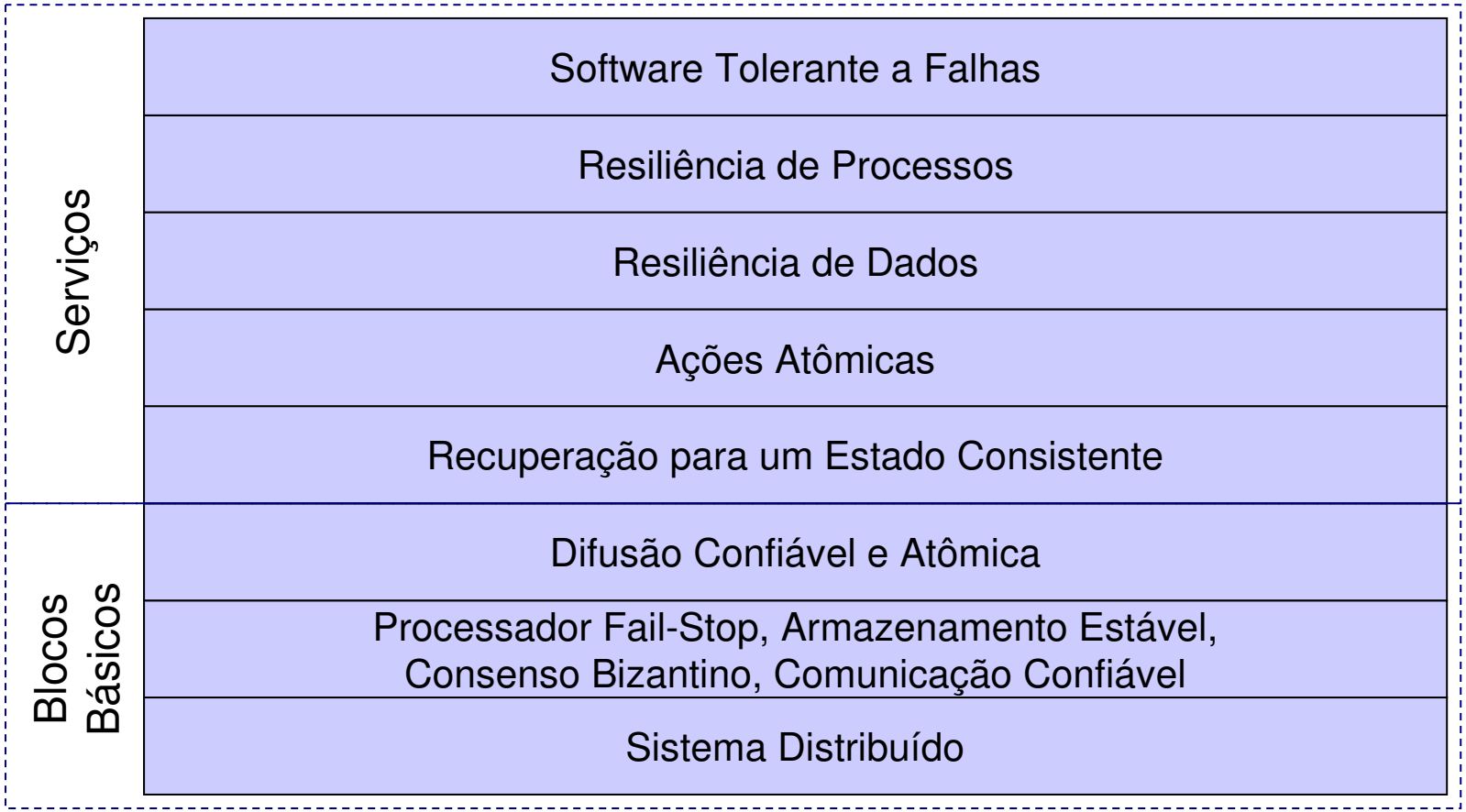
- Sistemas Distribuídos tem redundância intrínseca que pode ser utilizada para TF.
- Incluem múltiplos processadores independentes que podem incrementar o desempenho do sistema através de paralelismo, reduzindo custos da TF.
- Contém múltiplos componentes, o que incrementa o risco de defeitos, requerendo o uso de técnicas de TF.

Sistemas Distribuídos TF

- Implementar SDTF é difícil por muitas razões:
  - Sincronização
    - Deve evitar conflitos e deadlocks.
  - Detecção de Falhas/Defeitos
    - Deve ter uma visão consistente de quais componentes falham e em que ordem.
  - Recuperação
    - Deve prover mecanismos de recuperação para atuarem após defeitos e/ou recuperações.
  - Consistência
    - Deve manter uma visão consistente do sistema independente das falhas e recuperações.

Níveis de TF

Jalote, P. Fault Tolerance in Distributed Systems. Prentice Hall, Englewood Cliffs, New Jersey, 1994.



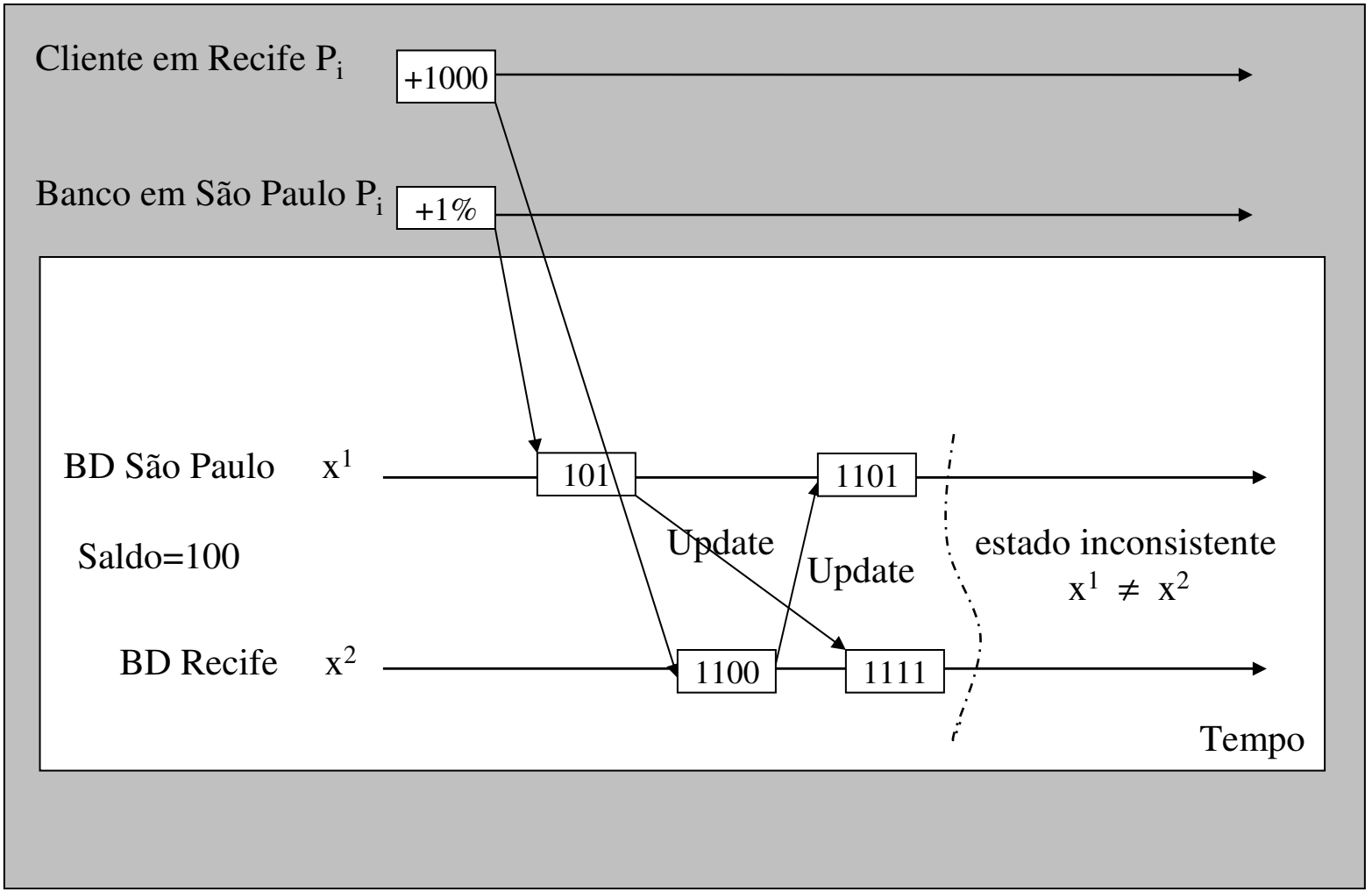
Tipos de Defeitos

- É importante saber como um componente ou um sistema se comporta no caso de defeitos
  - Pára e muda para um estado pré-definido que pode ser detectado (**Fail-stop**)
  - Defeito manifesta-se através da parada de um componente ou da perda do seu estado interno (**Colapso ou Crash**)
  - Simplesmente não responde a determinadas entradas, omitindo respostas (**Omissão**)

Tipos de Defeitos

- Responde ou muito cedo ou muito tarde, pois o atraso da resposta não é conhecido (**Temporização**)

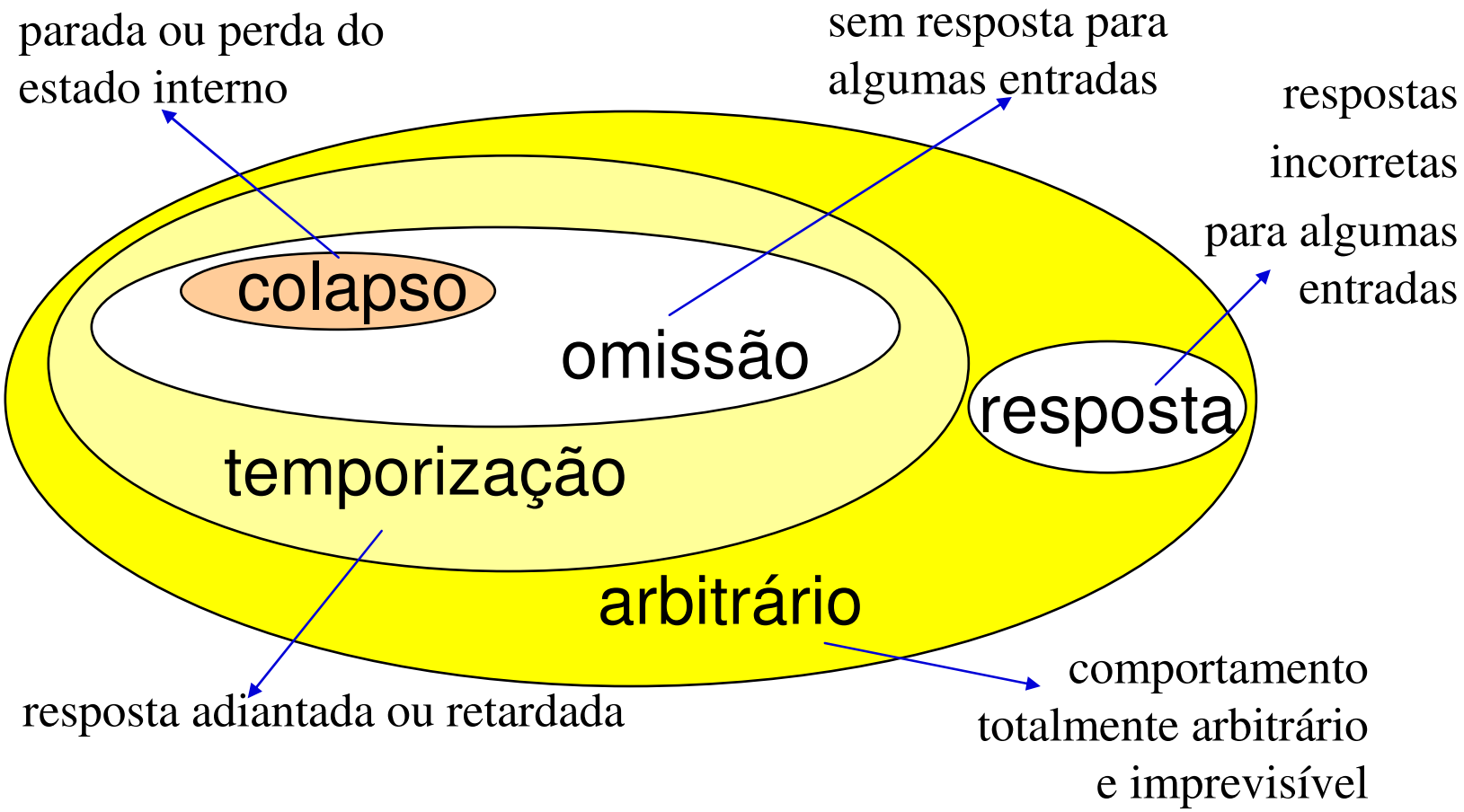
Conseqüência de um Defeito de Temporização



Tipos de Defeitos

- Responde ou muito cedo ou muito tarde, pois o atraso da resposta não é conhecido (**Temporização**)
- Os valores de saída são incorretos devido a uma computação incorreta para algumas entradas (**Valor ou Resposta**)
- Falha qualquer provoca comportamento totalmente arbitrário e imprevisível do durante o defeito (**Bizantina**)

Classificação de Defeitos



Objetivo da Tolerância a Falhas

Obter DEPENDABILIDADE

Disponibilidade: prontidão para ser utilizado

Confiabilidade: execução contínua sem defeitos

Segurança (Safety): recuperação de defeitos temporários sem qualquer acontecimento catastrófico

Mantenabilidade: versa sobre a facilidade com que um sistema é reparado

Desempenho: baixo custo computacional, degradação gradual

Testabilidade:facilidade para testar o sistema (pontos de teste, testes automatizados)

Paradigmas de Estruturação

Como estruturar um sistema para suportar TF?

Diferentes paradigmas de estruturação:

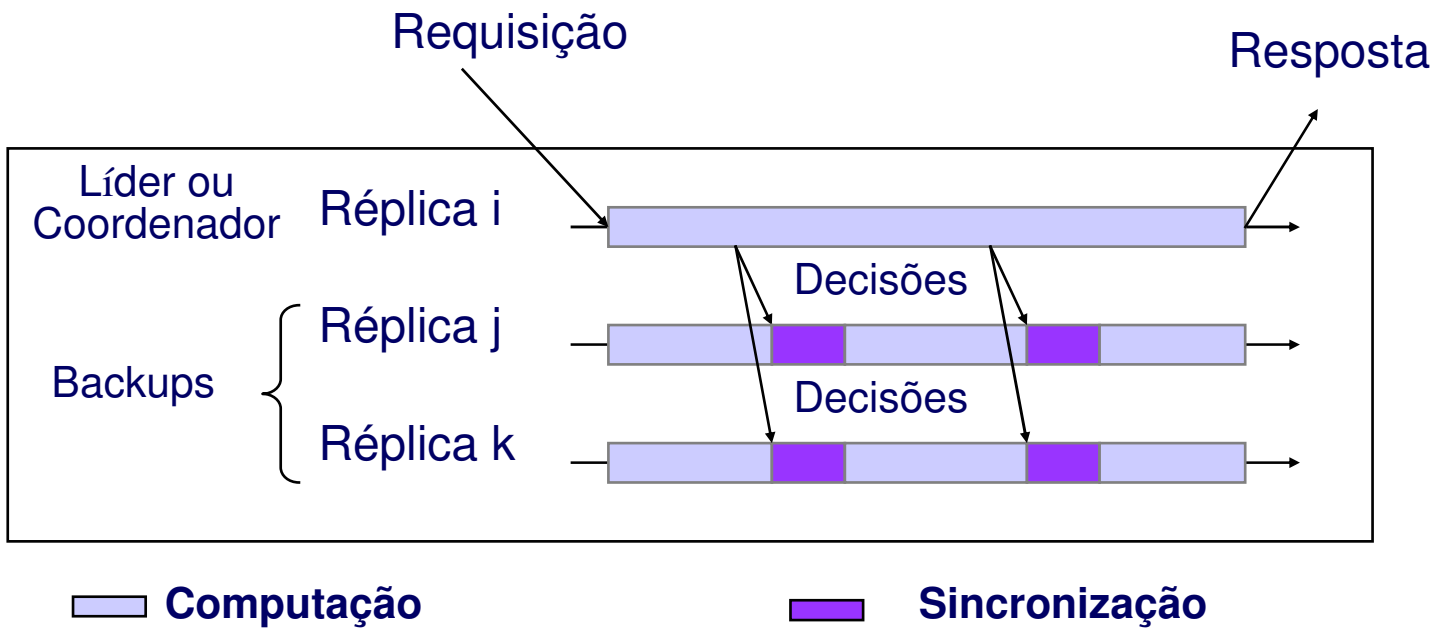
- Redundância de hardware (ativa, passiva e híbrida)
- Redundância de informação (códigos de detecção e/ou correção de erros)
- Redundância temporal (refazer computações)
- Redundância de software (programação *n* versão, recovery block)

Paradigmas de Estruturação

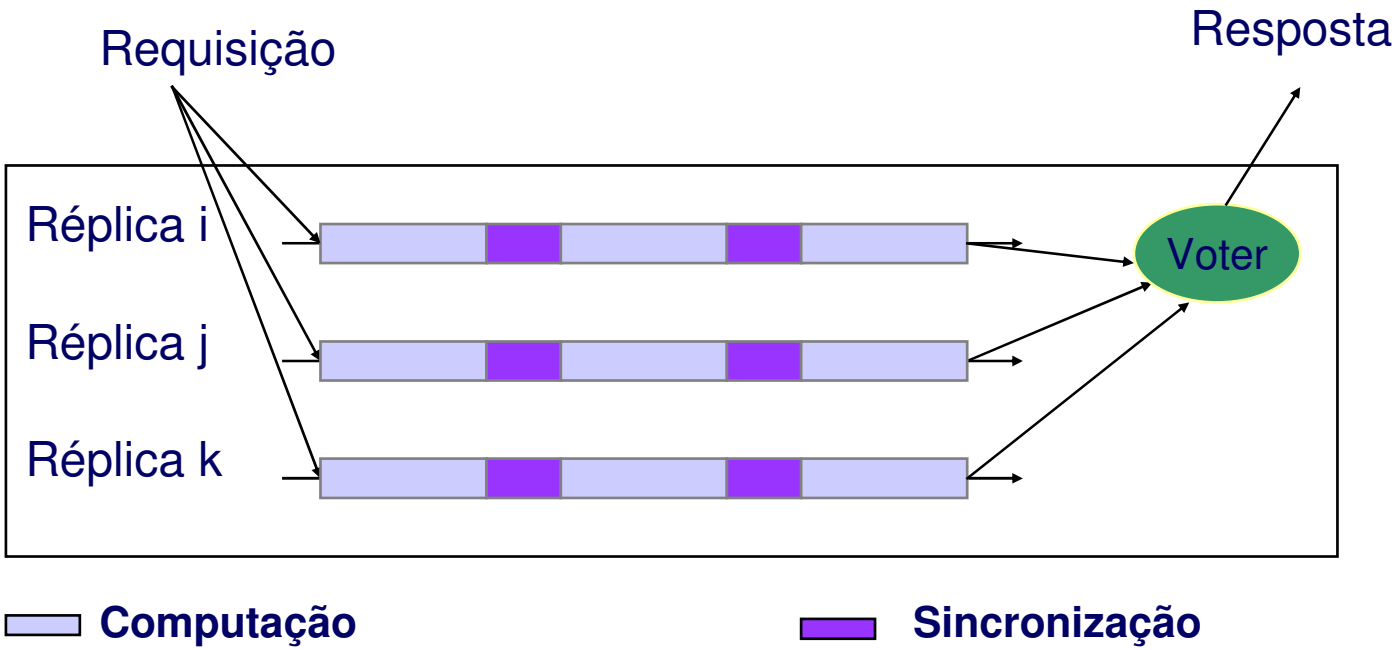
Principais estruturas ustilizadas em SD para alcançar TF:

- Replicação Passiva
- Replicação Ativa
- Replicação Semi-Ativa
- Replicação Paralela-Ativa

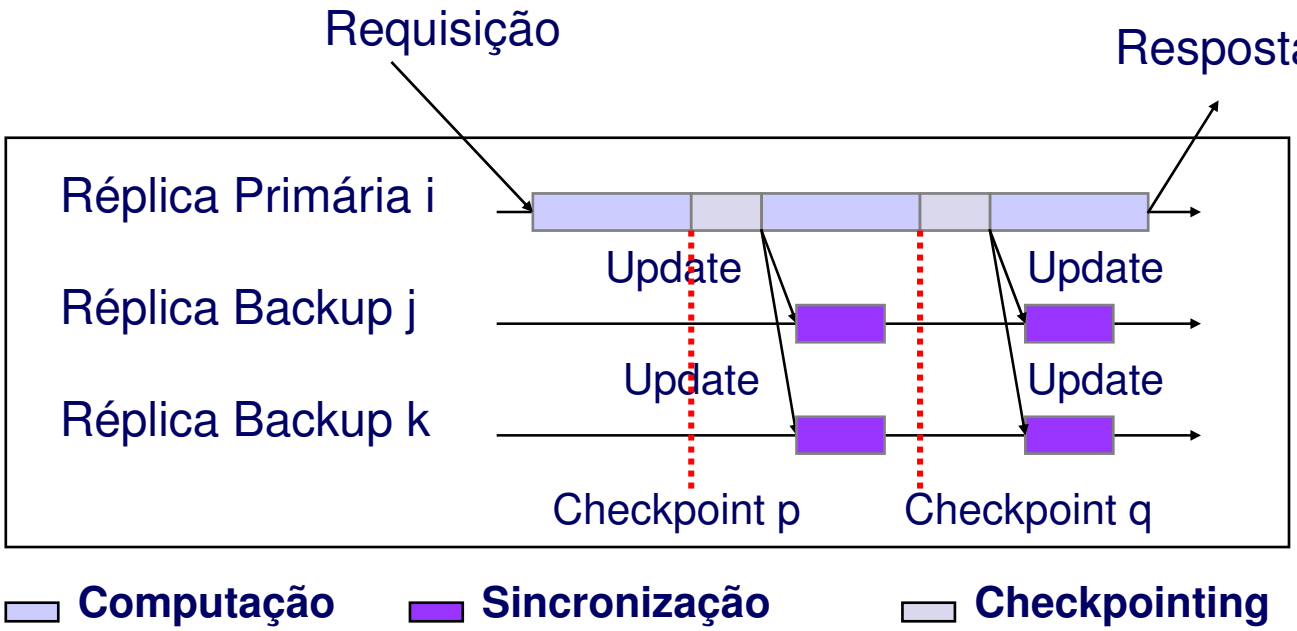
Replicação Passiva



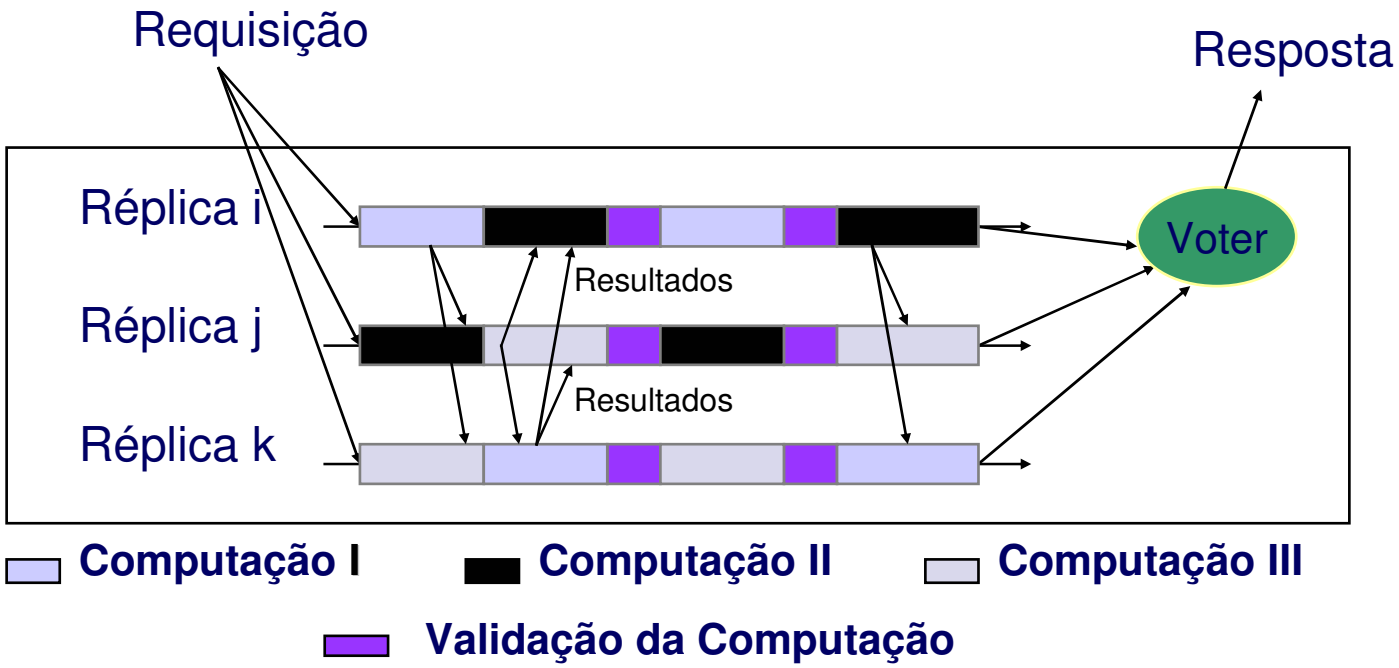
Replicação Ativa



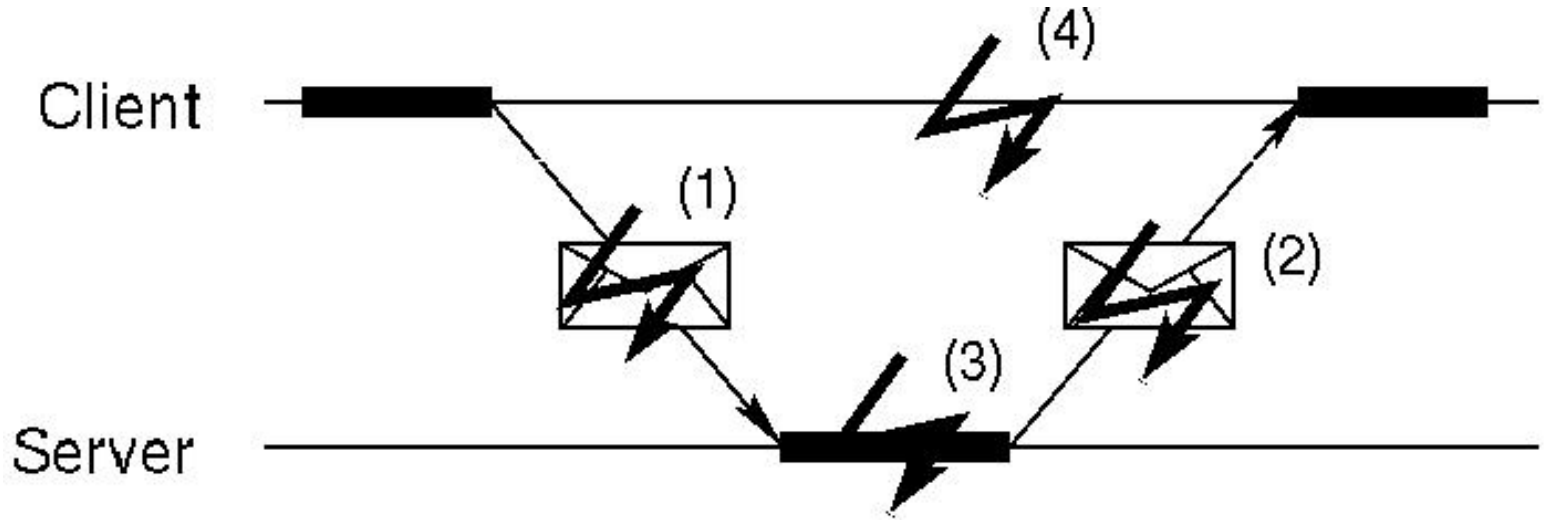
Replicação Semi-Ativa



Replicação Paralela-Ativa

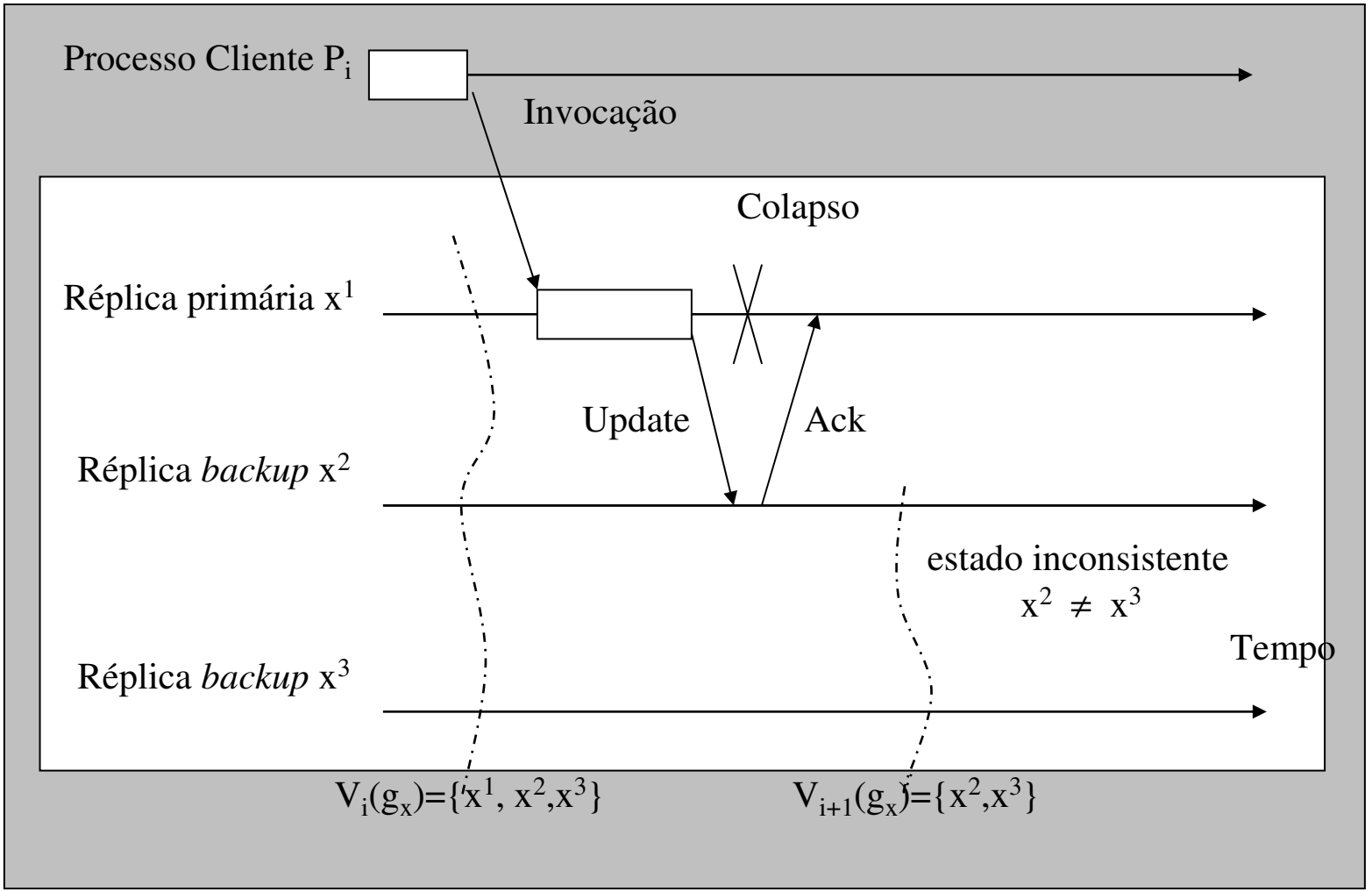


Falhas na Comunicação

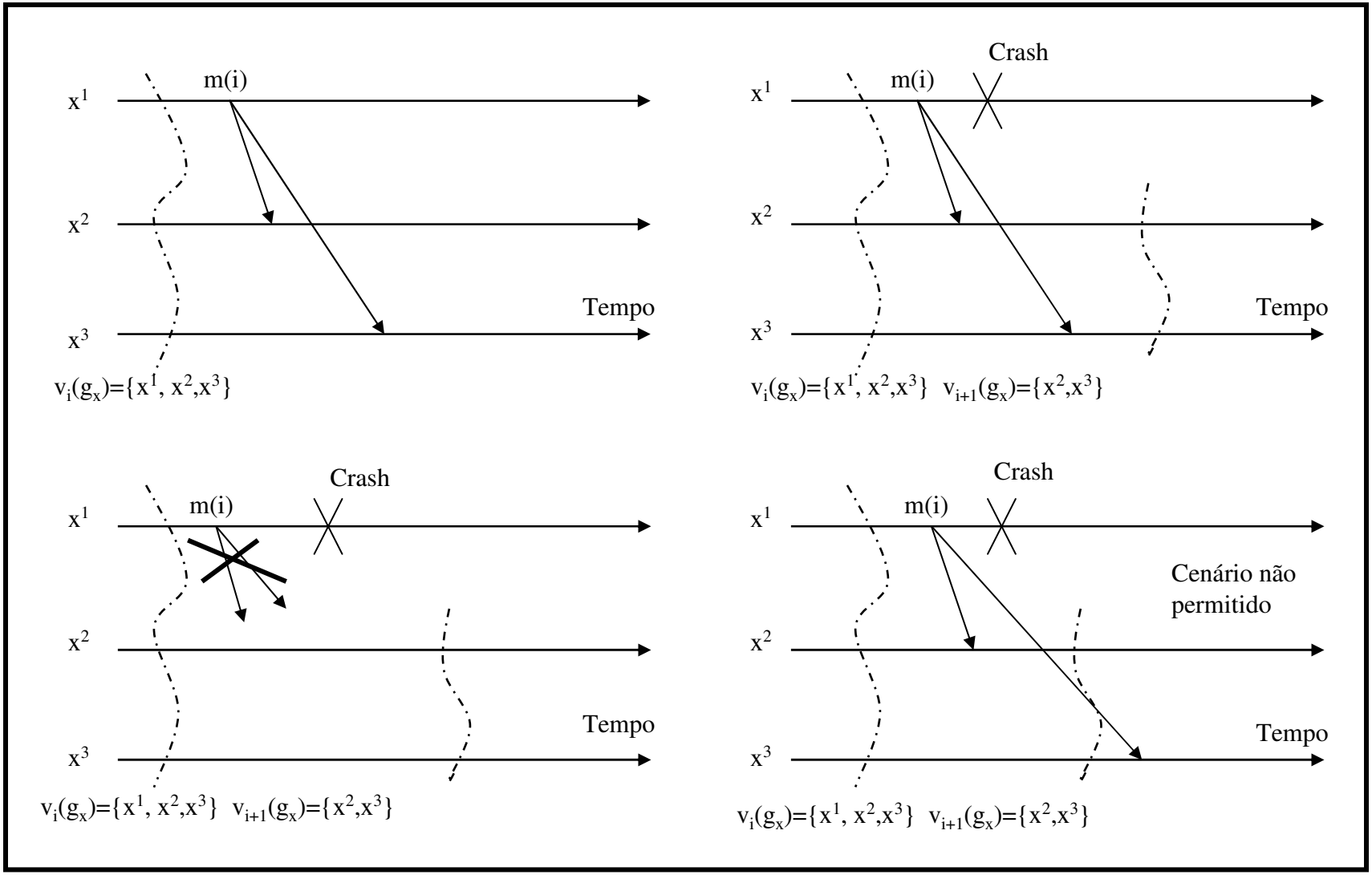


1. Requisição perdida
2. Resposta perdida
3. Colapso no servidor
4. Colapso no cliente

Possível Problema em SD



Solução: Ação Atômica



Síntese da TF em SD ou SDTF

- **Redundância** é o requisito chave para implementar qualquer Sistema Tolerante a Falhas.

- Nós necessitamos de técnicas de **gerenciamento de replicação** para facilitar a implementação de Sistemas Tolerantes a Falhas.

- As características de Tolerância a Falhas devem ser **transparentes aos programadores e usuários** do Sistema Tolerante a Falhas.

Síntese da TF em SD ou SDTF

- Tolerância a Falhas incrementa a **complexidade** do sistema.
- Implementar sistemas **paralelos e/ou distribuídos** como Tolerante a Falhas é uma tarefa difícil.
- Há necessidade por **ferramentas de suporte** à implementação de sistemas distribuídos TF.

Aspectos de implementação de TF em SD

Modelos e Hipóteses

- Algoritmos distribuídos TF
  - validação baseada em hipóteses: de falhas e de atrasos de comunicação
- Modelo de falhas (inclui processo e comunicação)
  - No processo: colapso, omissão, temporização (rápido ou lento) ou arbitrária (se contraditória, chamada bizantina)
  - Na comunicação: mensagem atrasada, perdida, duplicada ou fora de ordem
  - Alguns modelos incluem hipóteses sobre recuperação (*failure-stop* ou *failure-recovery*)

Modelos e Hipóteses

- Modelos de tempo
  - Síncrono
    - Tempo de transmissão e processamento conhecidos (*bounded time*)
    - Necessita escalonamento tempo real forte e técnicas de controle de fluxo
    - Apropriado para aplicações tempo real críticas
  - Assíncrono (*time free model*)
    - Tempo de transmissão e processamento desconhecidos (*unbounded time)*
    - Implementação simples
    - Acordo e broadcast confiável são impossíveis neste modelo

Modelos e Hipóteses

- Modelos de tempo
  - Prática = sincronismo parcial via timeouts
    - Um modelo síncrono (parcial) é assumido, mas sempre há uma probabilidade das hipóteses do modelo serem violadas
    - Indicado para aplicações não tão críticas
    - Timeouts podem ser grandes para minimizar a probabilidade de falhas suspeitas

Modelos e Hipóteses

- Modelos de tempo
  - Modelo alternativo: assíncrono temporizado.
    - Hipótese básica: drifts dos relógios locais conhecidos e conseqüente possibilidade de computação dos bounds no pior caso.
    - Interessante para implementar SD fail-safe

Consistência

- Preocupação em manter integridade dos dados frente a eventos concorrentes
- Técnicas utilizadas para minimizar esforço de programação:
  - Uso de tempo global
  - Uso de tempo lógico

Consistência

- Uso de tempo global
  - Motivação: difícil coordenação e ordenamento de eventos devido ausência de relógio global
  - Tempo físico global
    - Sincronização interna – relógio local lê valor de relógio externo; executa função de correção; ajusta relógio local.
    - Sincronização externa – relógio local periodicamente consulta servidor de tempo; ajusta relógio local.
      - Exemplo: Internet Network Time Protocol (NTP) oferece offset de algumas dezenas de milisegundos

Consistência

- Uso de tempo lógico – ordena eventos de acordo com a relação aconteceu-antes
  - Implementados por contadores em cada processo
  - Carregados por piggy-back nas mensagens
  - Implementa relação de ordem parcial
  - Vetor de clock – versão mais elaborada
  - Implementa relação de ordem causal

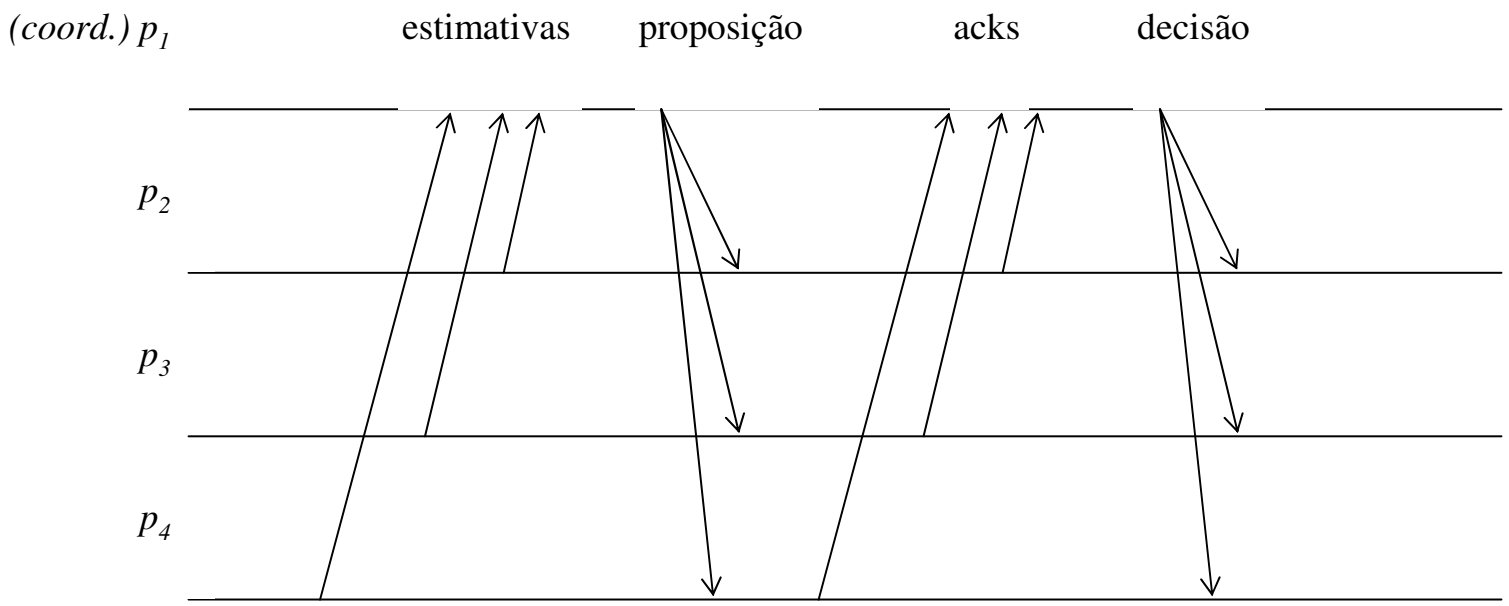
Consenso

- **Problema fundamental em TF em SD**
- **Definição tradicional**
  - considerando um conjunto de processos, onde cada um tem um valor inicial, eles devem decidir sobre um dos valores iniciais proposto por um deles.
- Deve satisfazer as seguintes propriedades:
  - TERMINAÇÃO: todo processo correto toma uma decisão num tempo finito;
  - ACORDO: se dois processos corretos tomam uma decisão, eles terão a mesma decisão;
  - VALIDADE: se um processo correto decide sobre um valor *d*, então *d* foi proposto por algum processo.

Consenso

- consenso equivale a um acordo
- Acordo na presença de falhas arbitrárias em processos é um acordo bizantino
- Acordo sobre um vetor de valores é chamado consistência interativa
- Consenso é impossível se:
  - Comunicação for não confiável
  - Tempo de comunicação for indeterminado

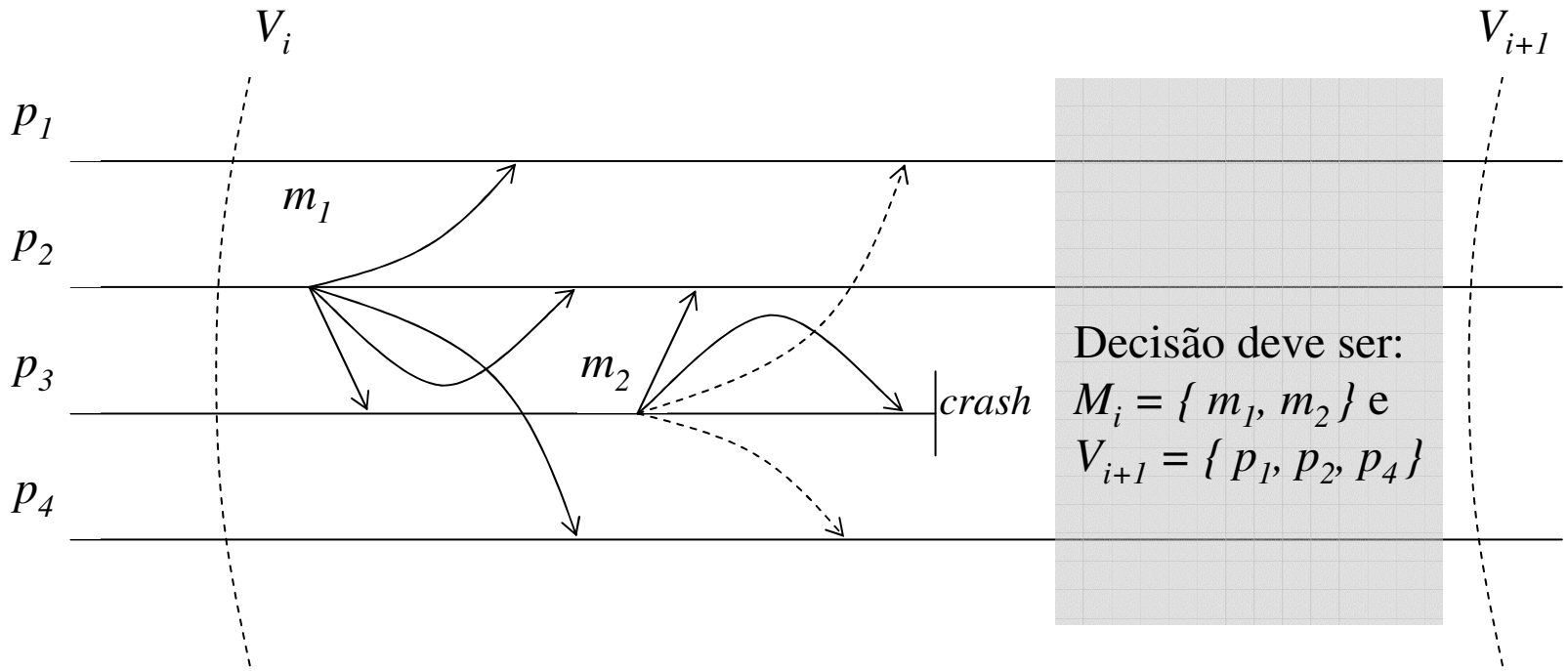
Algoritmo de Chandra e Toueg para resolver o Consenso



Algoritmo de Chandra e Toueg para resolver o Consenso

- todos participantes enviam suas estimativas para o coordenador do grupo *px*;
- ao receber todas as estimativas *px* envia para o grupo a sua proposta de nova visão;
- um processo *py* que recebe a proposta de *px* ou aceita a proposta e retorna um *ack*, ou não aceita a proposta e retorna um *nack*. Em ambos os casos *py* vai para a próxima rodada (*round*);
- quando *px* recebe a maioria das mensagens de *ack*, ele envia para o grupo uma mensagem de decisão. Ao receber a mensagem de decisão, todos os membros corretos devem assumir aquela visão.

Comunicação em Grupo



- Vide também outra seqüência de slides

Comunicação em Grupo

- Reconsiderando as três propriedades do consenso para o caso de comunicação em grupo:
  - TERMINAÇÃO: todo processo correto entrega um conjunto de mensagens *Mi* na visão *Vi* e instala uma nova visão *Vi+1*, num tempo finito;
  - ACORDO: se dois processos *px* e *py* entregam um conjunto de mensagens *Mi,x*, *Mi,y* na visão *Vi*, e instalam novas visões *Vi+1,x* e *Vi+1,y*, então *Mi,x* = *Mi,y* e *Vi+1,x* = *Vi+1,y*;
  - VALIDADE: um processo *px* na visão *Vi*, que não foi indicado como suspeito por nenhum outro processo da visão *Vi*, é um membro da visão *Vi+1*.

Comunicação em Grupo

- Especificamente em relação ao *membership* do grupo, considerando uma dada visão *V* pode-se dizer que:
  - se um processo *px* alcança um valor de decisão *Vi+1* na visão *Vi*, então todo processo em *Vi* alcança o valor de decisão *Vi+1* ou falha;
  - se o processo *px* alcança o valor de decisão *Vi+1* na visão *Vi*, então *px* não alcança um valor de decisão diferente *Vj*;
  - valor de decisão alcançado por *px* não é pré-determinado.

Técnicas de TF em SD

- Dois propósitos:
  - Para minimizar os problemas oriundos da distribuição
    - exemplo: impacto negativo de um processo ou nó falho na disponibilidade de um serviço distribuído
  - Para tornar um serviço distribuído tolerante a falhas
    - requer replicação de processos e/ou dados sobre múltiplos nós