Sistemas Tolerantes a Falhas

Introdução a Sistemas Computacionais Tolerantes a Falhas

Prof. Jó Ueyama

Aula de Hoje

- Tolerância a Falhas (TF) – Introdução ao assunto
  - Computação Ubíqua e Pervasiva
- Algumas Definições
  - Falhas, Erros, Fracasso (*failure*)
- Tipos de Falhas
- Redundância e seus tipos
- Software Confiável

Sistema Computacional

- Aumentou dramaticamente em escopo, complexidade e ambiente (*pervasive computing*).

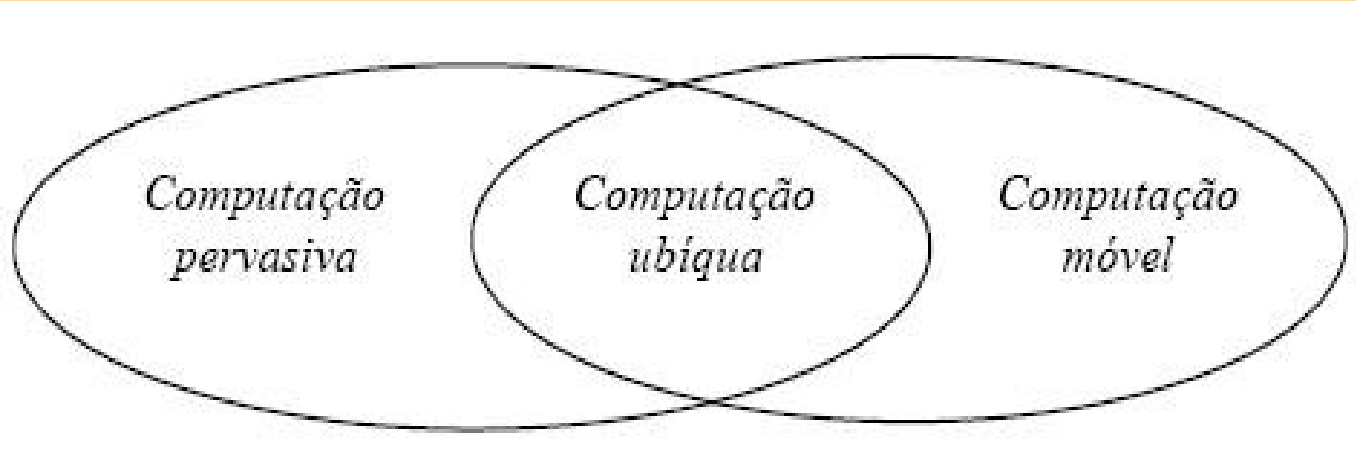
- Alguns sistemas necessitam um alto grau de complexidade
  - Em aviões, controle de tráfego aéreo, dispositivos médicos, reatores nucleares, trens de alta velocidade, serviços bancários, sistemas náuticos e militares, etc.
- As consequências das falhas destes sistemas pode ser
  - Médio ou catastrófico com até perdas de vida

Computação Pervasiva e Ubíqua

- Computação Móvel
  - O usuário pode levar o equipamento para qualquer lugar
  - Sem perder o poder computacional
- Computação pervasiva
  - computação invisível ao usuário
  - Obter informações do ambiente no qual está inserido
  - Ajusta a aplicação para melhor atender as necessidades do usuário

Computação Pervasiva e Ubíqua

- Computação Ubíqua
  - Une a mobilidade com a onipresença da pervasiva
  - O sistema pode funcionar ajustando dinamicamente de acordo com o contexto em que ele se encontra
  - e.g. receber notícias de uma cidade em que o usuário está visitando



Sistema Computacional

- Softwares normalmente assumem a maior parte da funcionalidade destes sistemas pervasivos e ubíquos críticos
  - Erros neles podem ser catastróficos
  - Apesar dos avanços, nem todos os erros são identificados
- As IDEs ajudam a minimizar estes erros
- Erros podem ser corrigidos utilizando artefatos (e.g. SOs conhecidos como o Windows usam os *patches*)

Exemplos de Incidentes

- Foguete Ariane V lançado pela Agência Espacial Europeia explodiu 40s após o lançamento
  - Prejuízo de meio bilhão de dólares
- Várias falhas tem sido encontradas em softwares dos aviões Air Bus A320
- Um problema de software causou um vazamento de radiação na unidade de Sellafield
- Uma falha de software no sistema do Patriot causou a perda de um míssil

Falhas em hardwares e softwares

- Existem desafios de falhas em softwares e nos hardwares
- Em hardwares
  - Normalmente, é mais previsível (durante o tempo de uso)
- Em softwares
  - É diferente, pois eles não 'deterioram' com o tempo
  - Os erros são mais de lógica, o que os tornam mais complexos
  - Não podemos simplesmente duplicar o sistema, pois isto duplicará os erros

Aula de Hoje

- Tolerância a Falhas (TF) – Introdução ao assunto
  - Computação Ubíqua e Pervasiva
- Algumas Definições
  - Falhas, Erros, Fracasso (*failure*)
- Tipos de Falhas
- Redundância e seus tipos
- Software Confiável

Algumas Definições

- Falha
  - Uma causa identificada ou hipotetizada que leva a um erro (e.g. calcular errado o DV do CPF)
- Erro
  - É um estado de um sistema que pode levar a um fracasso na execução do sistema (e.g. um sistema q aceita CPFs errados)
- Considera-se que há falhas quando os erros são detectados no sistema
- Diz-se que um fracasso (*failure*) ocorre quando o serviço desvia-se do especificado

Aula de Hoje

- Tolerância a Falhas (TF) – Introdução ao assunto
  - Computação Ubíqua e Pervasiva
- Algumas Definições
  - Falhas, Erros, Fracasso (*failure*)
- Tipos de Falhas
- Redundância e seus tipos
- Software Confiável

Tipos de Falhas de Hardware

- Permanente: reflete um elemento que está sem funcionamento de forma ”permanente”
  - E.g. Um led queimado, HD com *crash*
- Transiente: é causado por um malfuncionamento de um componente por um determinado tempo
  - Depois de um tempo o funcionamento volta a ser perfeito
  - E.g. Ruído temporário em uma ligação de telefone
- Intermitente: a falha nunca vai embora completamente; fica entre a quietude ou ativo
  - E.g. Uma conexão mal ligada

Tipos de Falhas de Hardware

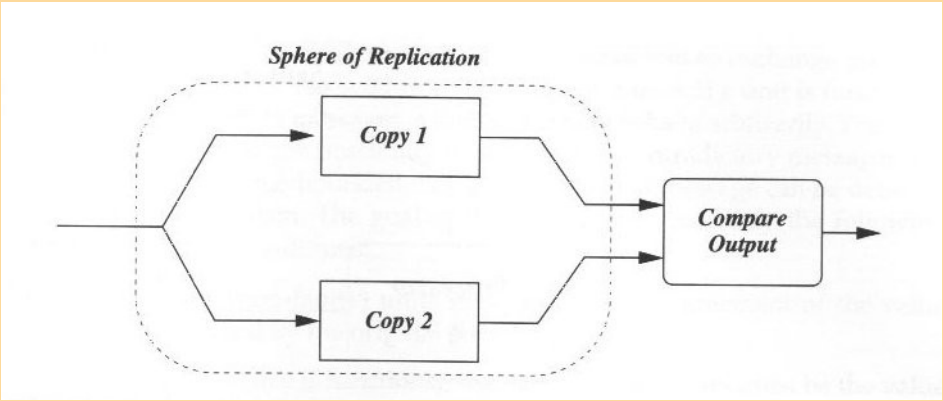
- Uma outra classificação: benigna ou maliciosa (*Byzantine*)
- Benigna: um elemento que deixa de funcionar completamente
  - Fácil de detectar e de retificar
- Maliciosa: falhas causadas quando um elemento funciona aparentemente de forma correta
  - E.g. Um sensor de altura de um avião que reporta 1000 pés no lugar de 2000 pés

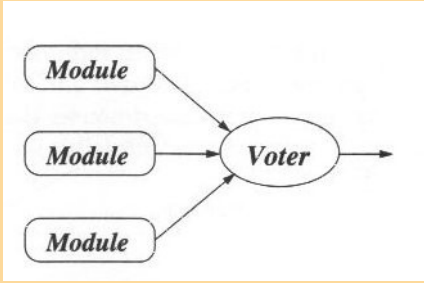
Aula de Hoje

- Tolerância a Falhas (TF) – Introdução ao assunto
  - Computação Ubíqua e Pervasiva
- Algumas Definições
  - Falhas, Erros, Fracasso (*failure*)
- Tipos de Falhas
- Redundância e seus tipos
- Software Confiável

Tipos de Redundância

- Toda TF é um exercício que explora e gerencia a redundância. Tipos de redundância:
- **Redundância de Hardware**: incluir um hardware extra para desempenhar uma função (e.g. CPU)
  - Dinâmicos ou estáticos se são ou não ativados quando os componentes redundantes estão disponíveis
  - Ativam a medida que os componentes falham





Tipos de Redundância

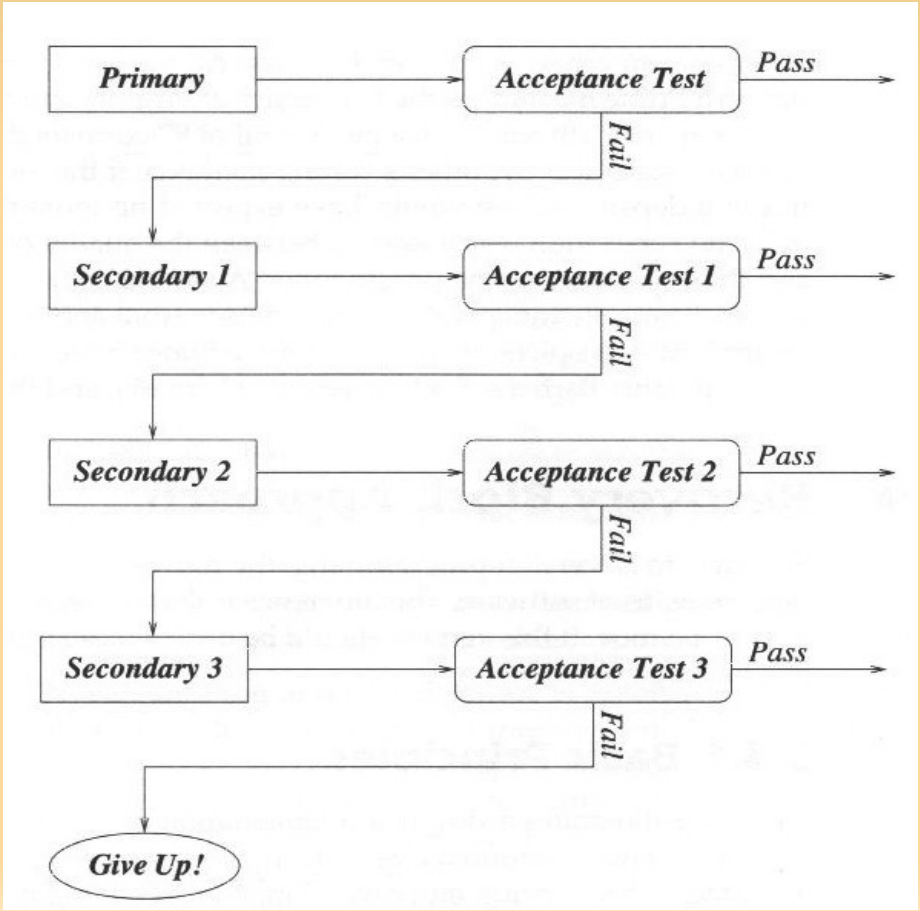
- **Redundância de informação**: adiciona bits extras para i) detectar erros e ii) corrigi-los
  - E.g. Usado em armazenamento de memória assim como em redes (e.g. Checksums)
- **Redundância de tempo**: re-execução de um mesmo software em um mesmo hardware
  - Pode detectar falhas de hardware transientes
  - Vantagem: não requer nem software extra nem um hardware extra
  - Desvantagem: *high performance penalty*

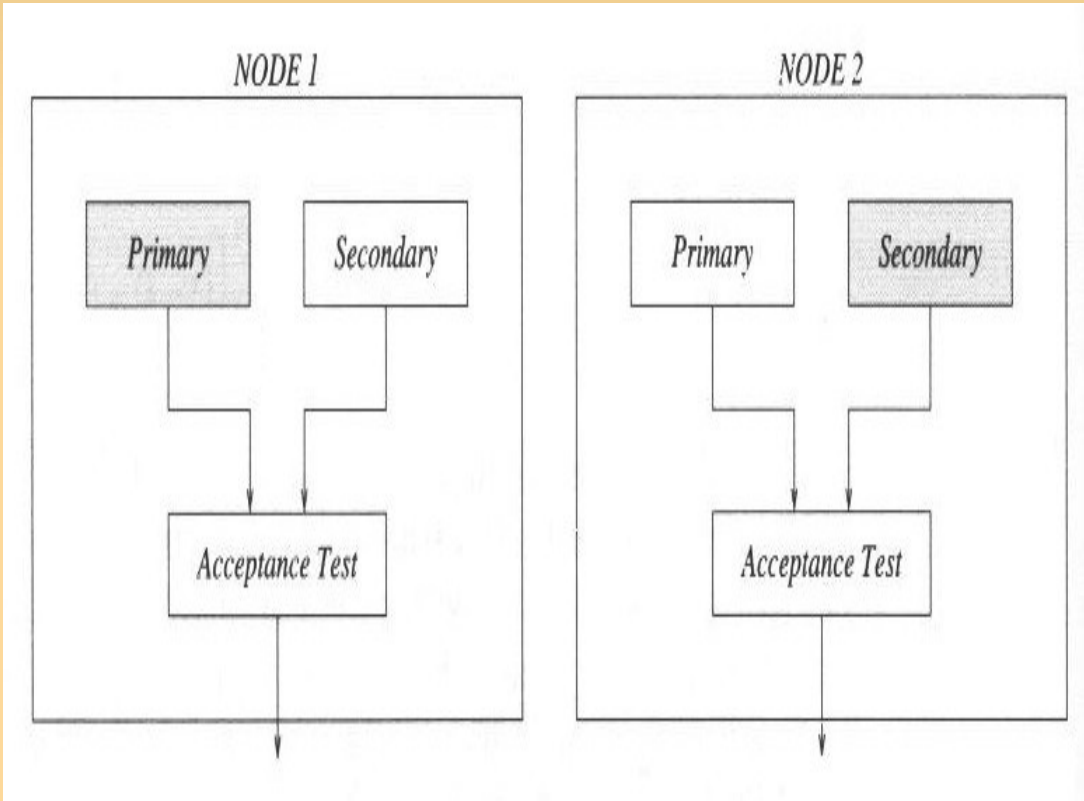
Tipos de Redundância

- **Redundância de software**: é utilizado principalmente para lidar com as falhas de software
- Todos os softwares construídos até hj tem possuído falhas. Solução?
- Construir componentes de software em locais distintos e por desenvolvedores ”desconectados”
- Utilizar algoritmos diferentes, por exemplo, utilizar heapsort e quicksort e comparar os resultados
  - Usar *voters* e *comparators*
- Os componentes redundantes podem ser executados em paralelo ou sequencialmente

Tipos de Redundância de Software

- Pode ser executados **sequencialmente** ou em **paralelo**
- Depende do que? Do hardware disponível





Aula de Hoje

- Tolerância a Falhas (TF) – Introdução ao assunto
  - Computação Ubíqua e Pervasiva
- Algumas Definições
  - Falhas, Erros, Fracasso (*failure*)
- Tipos de Falhas
- Redundância e seus tipos
- Software Confiável

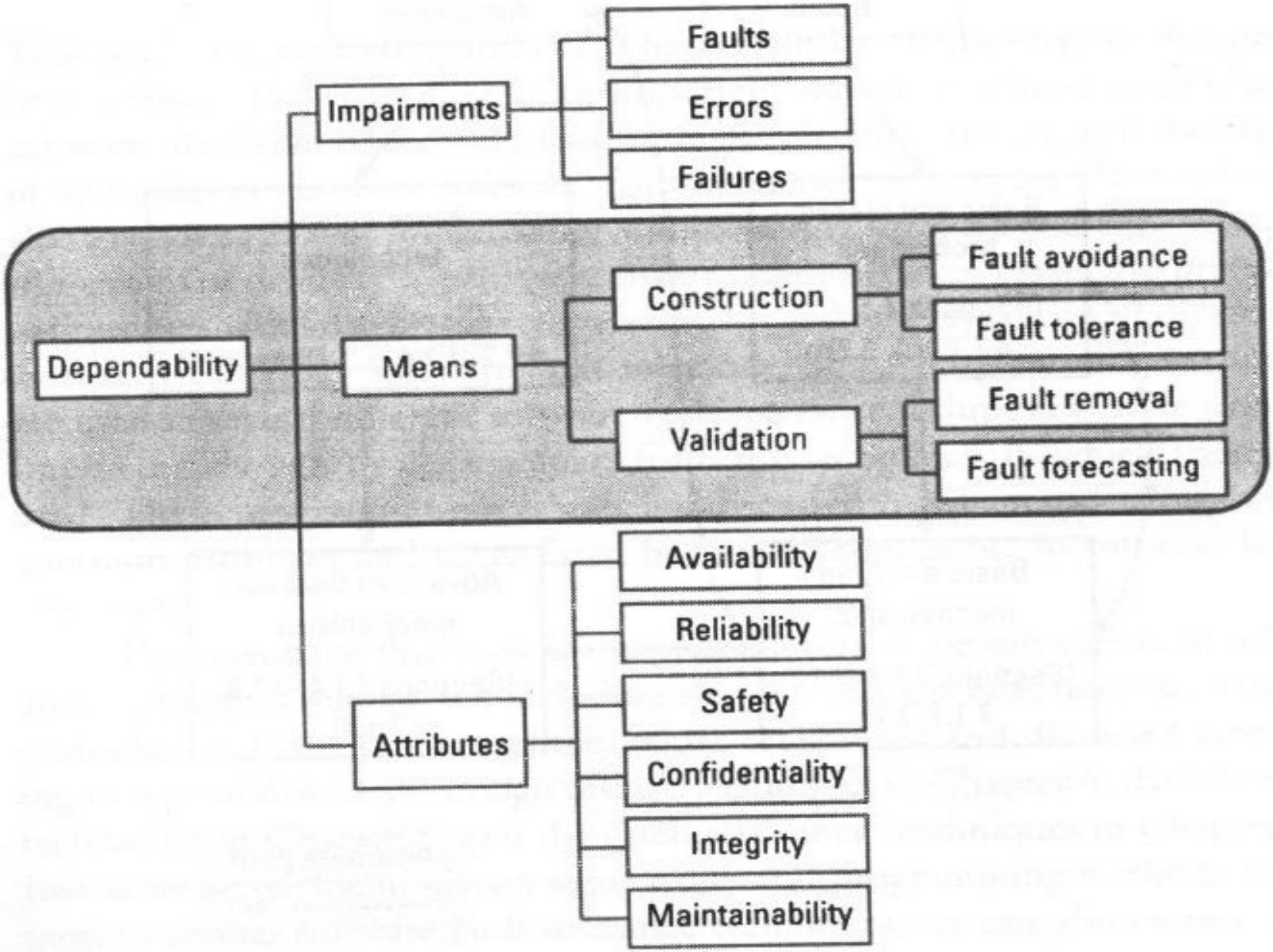
Software Confiável (*Dependable*)

- O que leva a precisarmos de tolerância a falhas (TF)?
  - Onipresença do software (pervasivo) e o seu uso tanto em aplicações críticas e cotidianas
    - e.g. equipamentos domésticos
  - Aumento da complexidade
    - e.g. sistemas embarcados em aviões
  - Presença de software em aplicações críticas (e.g. em reatores nucleares)
  - Precisamos deles em ambientes críticos?

Software Confiável

- A árvore na próxima ilustração exibe os impedimentos, mecanismos e os atributos da confiabilidade
- Os impedimentos são as falhas, erros e o fracasso
- Os atributos são disponibilidade, confiabilidade, segurança, confidenciabilidade, integridade e facilidade de manutenção

Software Confiável



Macanismos para ``Dependability"

- São de dois tipos
  - Mecanismos empregados durante a construção do software, i.e., *construction*
    - *Fault avoidance* ou *prevention*: evitar que ocorram erros (e.g. verificar se o usuário digitou os dados corretamente)
  - Mecanismos que contribuem para validar o software após a sua construção, i.e., *validation*
    - *Fault removal* e *forecasting*: eliminar os erros e estimar a presença de erros e as suas consequências

Software Confiável

- Mecanismos na construção
  - *Fault avoidance*: evitar a introdução de erros
  - *Fault tolerance*: verificar o que o sistema pode garantir (dentro do especificado) apesar das falhas
- Mecanismos na validação
  - *Fault removal*: identificar a presença de falhas e eliminá-las
  - *Fault forecasting*: estimar a presença de falhas, assim como a ocorrência e as suas consequências
- A seguir, analisaremos cada mecanismo em detalhe

Fault Avoidance or Prevention

- Estes mecanismos são usados para aumentar a confiabilidade do sistema e reduzir falhas
  - utilizados durante a construção
- Faz o uso dos requisitos e das especificações do sistema
- Utiliza modelos matemáticos para prover uma maior confiabilidade do sistema
- Leva em consideração a reusabilidade de código
  - Aumenta a confiabilidade. Por que?

Fault Avoidance or Prevention

- 1) *System Requirements Specification*
  - Especificações de requisitos são muitas vezes consideradas como imperfeitas
  - Isto ocorre pela falta de comunicação entre os desenvolvedores e os que especificam o sistema
  - Muitas das ferramentas endereçam mais os erros de implementação do que os da engenharia de requisitos
  - Porém, existem ferramentas interativas que ajudam neste processo
  - Estudo de caso na construção de sistemas reais (e.g. ALEPA)

Fault Avoidance or Prevention

- 2) Projeto estruturado e técnicas de programação
  - Estas técnicas tem se mostrado eficiente
  - Reduzem a complexidade (e.g. evitar o *goto*)
  - *Decoupling* e modularização tanto na implementação e no projeto ajudam a reduzir falhas (i.e., menor interdepedência entre os 'módulos')
  - Sistemas coesos ajudam neste processo
  - Outra técnica é a componentização: reusabilidade em todos os níveis (multilinguagem)
  - Todas estas técnicas ajudam a reduzir a complexidade e a resolução dos problemas
    - Levam a sistemas com menos falhas

Fault Avoidance or Prevention

- 3) Formal methods
  - Os requisitos são analisados usando-se ferramentas que traduzem-os para modelos matemáticos
  - Mais usado em pequenos projetos. Por que?
  - As especificações matemáticas são praticamente do tamanho do software
  - Porém, partes do software poderão ter os seus riscos avaliados baseados em métodos formais
    - Avaliar apenas as partes críticas

Fault Avoidance or Prevention

- 4) *Software reuse*
  - O reuso de software permite uma maior confiabilidade. Por que?
  - Objetos e componentes testados e aprovados poderão ser reusados
  - Pode trazer outras vantagens além da confiabilidade: economia no processo de desenvolvimento
  - Promove a coesão, o que aumenta a confiabilidade
  - Reuso deve ser medido de caso a caso, pois há diferentes níveis de confiabilidade requisitados em cada sistema individual.

Fault Avoidance or Prevention

- Outros mecanismos são também perfeitamente aceitos
  - Refinamento de requisitos de forma interativa com o usuário final, por exemplo
  - Utilização de bons métodos de ES (e.g. XP)
  - Implementação de códigos legíveis (e.g. documentação de código)
- A técnica de métodos formais é baseada em mecanimos muito complexos
- O reuso de software deve ser avaliado em cada caso, pois os requisitos alteram de sistema para sistema

Falhas Vs. Erros

- Falha – pode ser um defeito de hardware ou um software com defeito de programação
- Error – uma manisfetação da falha
- Examplo: Um circuito adicionador com uma saída que só emite 1
- Isto é uma falha, mas (ainda) não é um erro
- Se torna um erro quando o adicionador é usado e o resultado deve ser 0

Fault Removal

- Apesar dos esforços para evitar falhas, elas vão existir
  - Por isso, a remoção delas é necessária
- É uma técnica utilizada durante a verificação e validação
- É baseado em três técnicas
  - *Software testing*
  - *Formal inspection*
  - *Formal design proofs*

Fault Removal

- 1) Software testing
  - A técnica mais comum para remover erros
  - Dificuldade chave: custo para realizar testes exaustivos
    - Testar o software sob todas as circunstância e com todos as possíveis entradas
  - A chave é ter uma cobertura adequada de testes e derivar uma boa qualidade nos testes (para cada sistema)
  - Testes multiníveis: componentes mais críticos devem receber uma maior gama de testes

Fault Removal

- 2) Formal Inspection
  - É uma técnica prática bastante utilizada pela comunidade da indústria
  - Possui um rigoroso processo acompanhado de uma documentação que foca em
    - Examinar o código fonte
    - Encontrar falhas
    - Corrigi-las
    - Certificar que elas foram retiradas
  - Esta técnica é realizada por um grupo de pessoas antes da fase de testes

Fault Removal

- 3) *Formal Design Proofs*
  - Esta técnica está relacionada com a dos métodos formais (*fault avoidance and prevention*)
  - Usa modelos matemáticos para assegurar a corretude do programa
    - e.g. fornecendo uma bateria de dados para testes
    - A ferramenta gera *test cases* automaticamente
  - Não encontra-se consolidado ainda
    - Alto custo para sistemas robustos
  - Apropriado para partes do software tidos como críticas, durante a avaliação de riscos

Fault Removal

- As técnicas de t*est* e *formal inspection* são largamente utilizadas
  - Podem ser mais enfatizadas em partes consideradas como críticas em termos de confiabilidade
  - Não é possível testar de forma exaustiva
  - Os testes podem acusar a presença de falhas, mas não a sua ausência
  - As falhas só serão identificados quando o sistema estiver em operação
- A técnica de *Formal Design Proof* é custosa e complexa

Fault Forecasting

- Remoção de falhas é imperfeita e por isso a previsão de falhas é necessária
- A previsão de falhas é mais uma técnica para aumentar a confiabilidade do sistema
- Ela estima a presença, a ocorrência e as consequências das falhas
- Estuda o ambiente operacional do sistema para a previsão
- Utiliza também um conjunto de dados para gerar falhas de forma 'proposital'

Fault Forecasting

- 1) Estimativa da Confiabilidade
  - Utiliza técnicas estatísticas para inferir o grau de confiabilidade do sistema atualmente
  - Em cada estágio temporal aplica-se tal técnica estatística
  - Verificado/estimado durante os testes e durante a operação do sistema
  - Ele provê uma visão do grau de confiabilidade que um sistema tem em um determinado estágio

Fault Forecasting

- 2) Previsão de confiabilidade
  - Esta técnica é aplicada em vários estágios de desenvolvimento de software
  - Prevê a confiabilidade futura do software usando métricas e medidas pertencentes a cada domínio
  - A medida que as falhas ocorrem, modelos de confiabilidade podem ser utilizados para analisá-las
  - Mais utilizado após o início dos primeiros testes realizados no sistema

Fault Forecasting

- Estas técnicas verificam através de previões futuras se eles possuem o comportamento desejado
  - Comparando-se com os requisitos definidos previamente
- A previsão revela se mais testes ou medidas/técnicas de aumento de confiabilidade são necessárias
- A previsão não é uma análise de requisitos e não aponta os requisitos que foram deixados de lado

Fault Tolerance (TF)

- TF é uma forma de aumentar a confiabilidade do software
- Técnicas que permitem que os softwares falhem após o seu desenvolvimento
- Quando tais falhas ocorrem, tais técnicas evitam que o 'fracasso' ocorra na execução
- Técnicas clássicas de TF já existem há mais de 20 anos
- Várias destas técnicas serão vistas a seguir

Fault Tolerance - Redefinindo

- Sistemas tolerantes a falhas – idealmente são sistemas capazes de executar suas tarefas corretamente, independentemente de falhas de hardware ou erros de software
- Na prática - nunca podemos garantir a perfeita execução de tarefas sob quaisquer circunstâncias
- Limita aos tipos de falhas e erros que são mais prováveis de ocorrer

Necessidades de um sistema TF

- 1.
- 2.
- 3.

Por que Tfs? Aplicações Críticas

Aeronaves, reatores nucleares, fábricas de produtos químicos, equipamentos médicos

- Um mal funcionamento de um computador em tais aplicações pode levar a uma catástrofe
- A probabilidade de falha deve ser extremamente baixa, possivelmente uma em cada bilhão de operação por hora
- Também inclui aplicações financeiras

Por que TFs? Ambientes Críticos

- Um sistema computacional operando em um ambiente hostil que é sujeito à
  - Distúrbios eletromagnéticos
  - Colisões com partículas
- Número muito grande de falhas significa: o sistema não irá produzir resultados úteis a menos que alguma tolerância a falhas é incorporada

Por que TFs? Sistemas Complexos

- Sistemas complexos consistem de milhões de equipamentos
- Cada dispositivo físico tem uma certa probabilidade de falha
- Um número muito grande de dispositivos implica que a probabilidade de falhas é elevada
- O sistema irá enfrentar falhas em uma freqüência que a torna inútil

- Então, vamos estudar técnicas para implementar a TF

Fault Tolerance (TF)

- 1) *Single version software environment*
  - Nesta técnica, a TF é limitada; ela monitora as falhas seguindo os seguintes mecanismos:
  - Atomicidade das ações – verifica se alguns procedimentos críticos implementam tais ações
  - Verificação das decisões – certifica de que as decisões estão dentro das normas de segurança delineadas previamente
  - Tratamento das exceções – analisa se algumas operações possuem tratamento de exceções, obrigando em alguns casos (e.g. já implementado em algumas linguagens com o IDE)

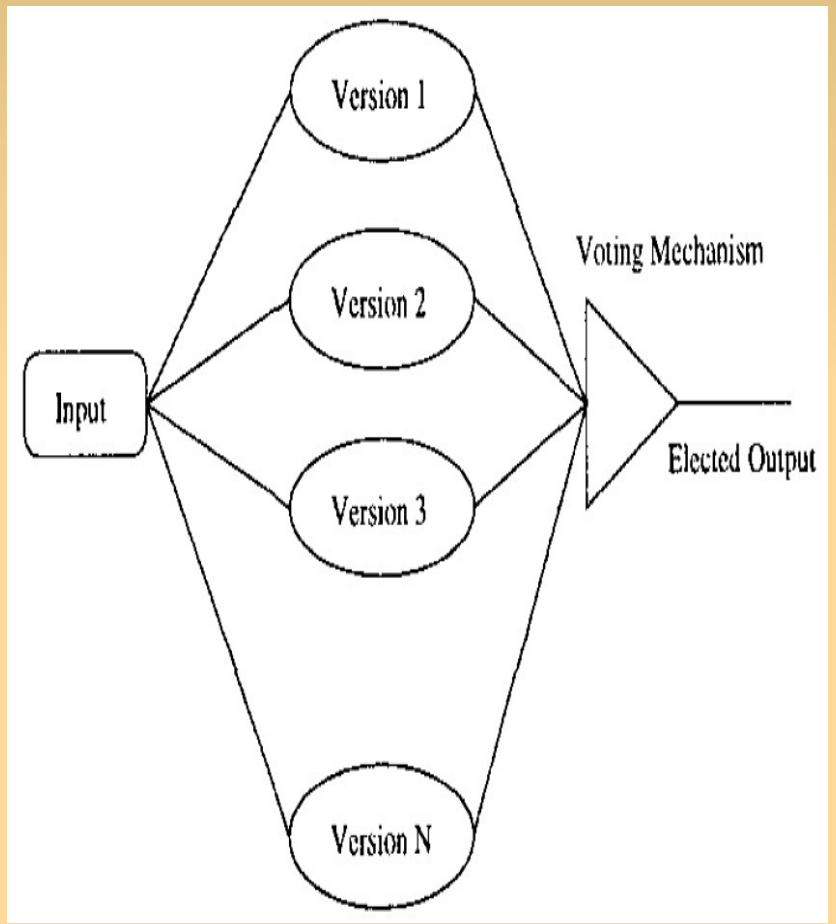
Fault Tolerance (TF)

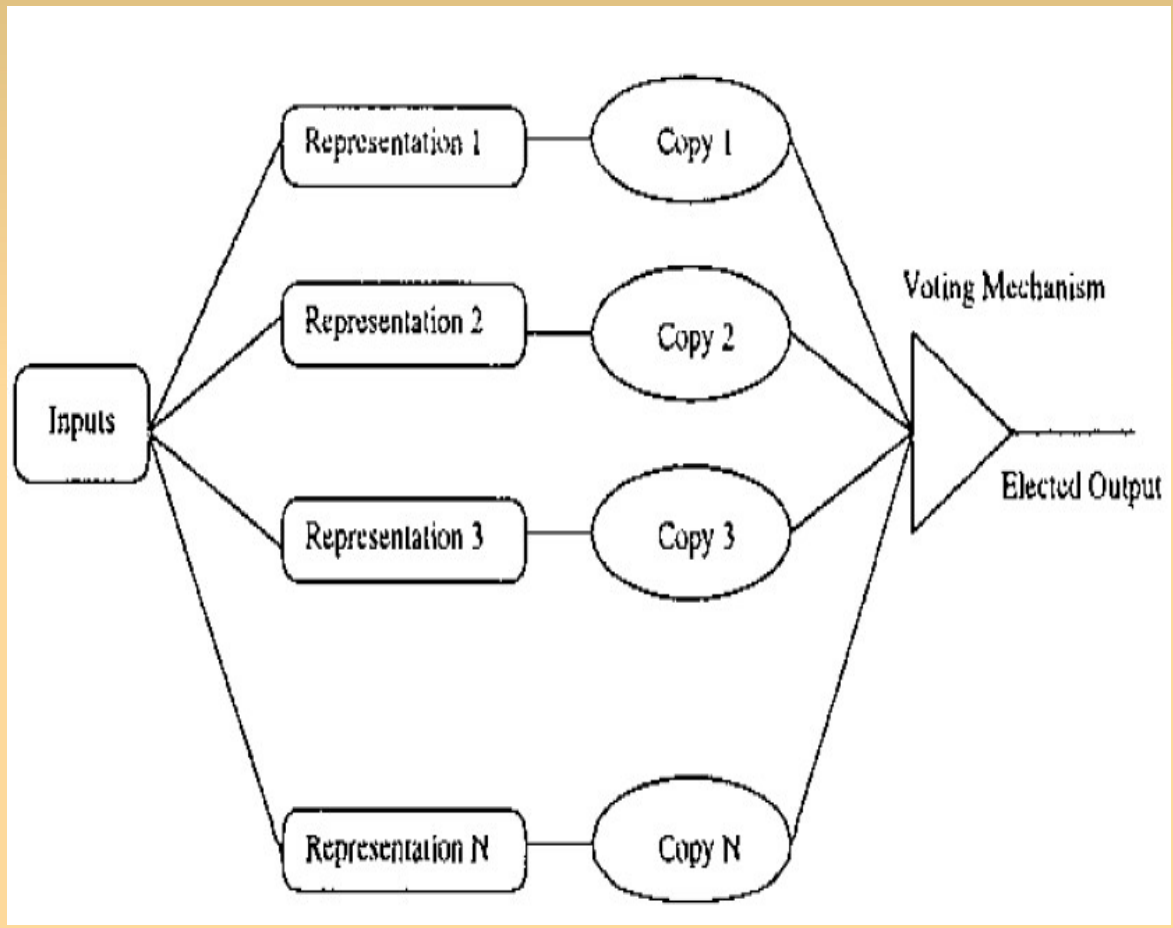
- 2) *Multiple version software environment* (MVSE)
  - Esta técnica utiliza múltiplas versões de um mesmo software desenvolvidos de forma independente
  - Existem várias técnicas que implementam o MVSE
    - Recovery blocks
    - N-version programming
    - N self-checking programming

Fault Tolerance (TF)

- 3) *Multiple data representation environment*
  - Esta técnica utiliza diferentes representações de dados de entrada para prover TF
  - Exemplos destas técnicas incluem
    - *Retry blocks* – utiliza um *acceptance test* para prosseguir com a execução dos *primary block* que processam dados com diferentes representações
    - *N-copy programming* – implementação *n-version* do data diversity
  - Estas técnicas serão revisitadas

Fault Tolerance (TF)





- Exemplos de *N-version e N-copy programmings*

Fault Tolerance (TF)

- As técnicas de TF 'toleram' falhas em sistemas após o seu desenvolvimento através de duas formas
  - *Single version*
  - *Multiple version*
    - *Design diversity*
    - *Data diversity*
- As técnicas de TF protegem a tradução de requisitos para implementação
- Mas, não 'protegem' a especificação de requisitos

Fault Tolerance (TF)

- As técnicas de TF tem sido utilizadas em diversas áreas que inclui
  - Aviação
  - Reatores nucleares
  - Saúde
  - Telecomunicação
  - Redes de sensores sem fio
  - Transporte

Tipos de Recuperação

- A recuperação é parte do processo de tolerância a falhas que envolve:
  - Detecção de erros
  - Diagnósticos de erros
  - Isolamento dos erros
  - Recuperação do erro

Tipos de Recuperação

- Detecção de erros
  - Identifica o estado de erro em que se encontra
- Diagnóstico de erros
  - A causa do erro é identificada
  - Avaliação dos prejuízos causados pelo erro
- Isolamento do erro
  - Prejuízos maiores são evitados
  - Evita a propagação de erros
- Recuperação do erro
  - Mudança de estado: 'erroneous' para 'error free'

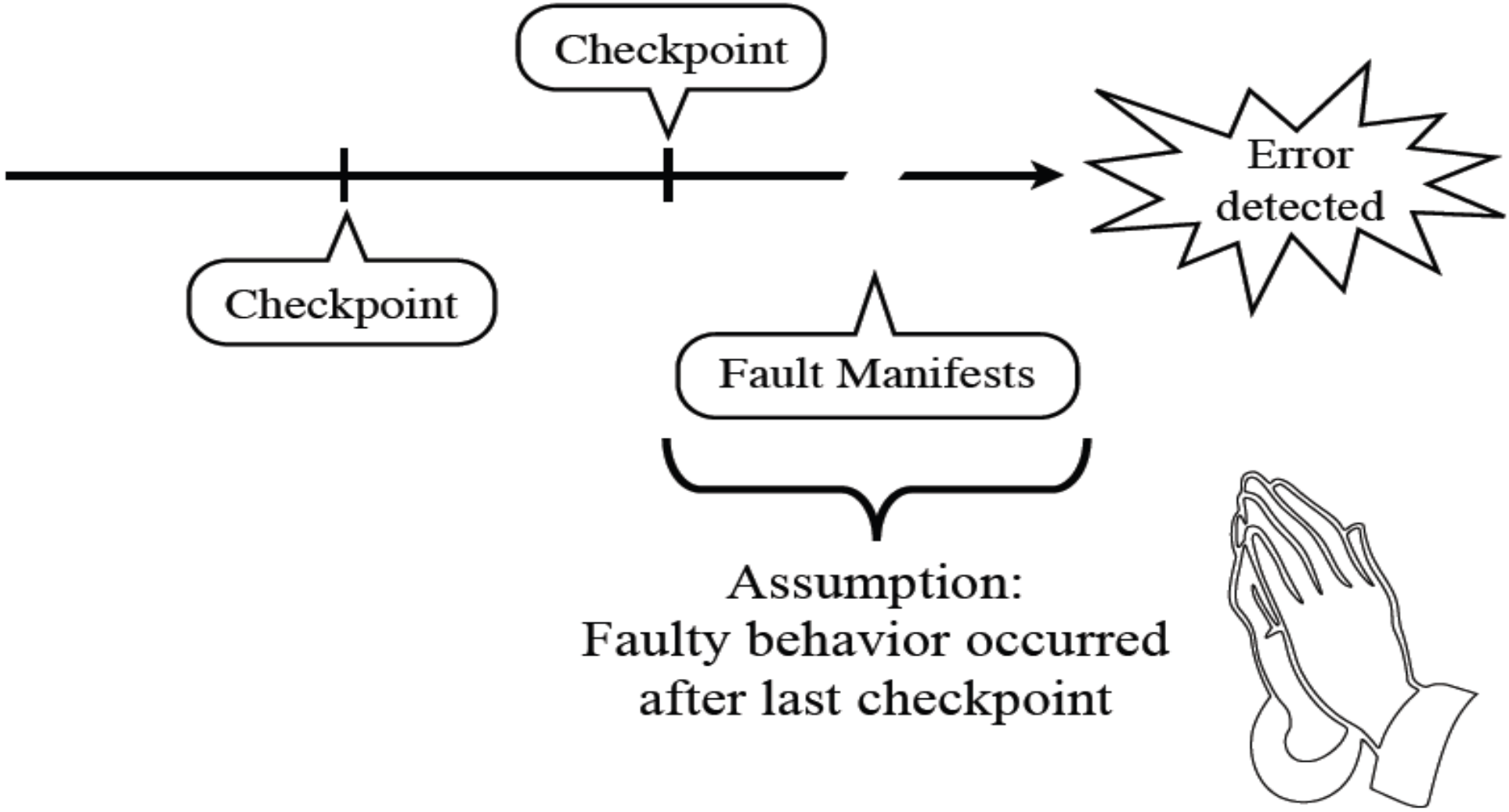
Tipos de Recuperação

- Existem dois tipos de recuperação de erros
  - *Backward Error Recovery*
    - Os estados do sistema são armazenados em um pontos pré-determinados chamados de checkpoint
    - Tais estados devem ser armazenados em repositórios persistentes
    - Recupera (volta) para o estado 'error-free' usando o checkpoint
  - *Forward Error Recovery*
    - Detecta o erro
    - Avaliação detalhada dos prejuízos causados
    - *State Reconstruction* (e.g. recupera do erro baseado no conhecimento prévio adquirido)

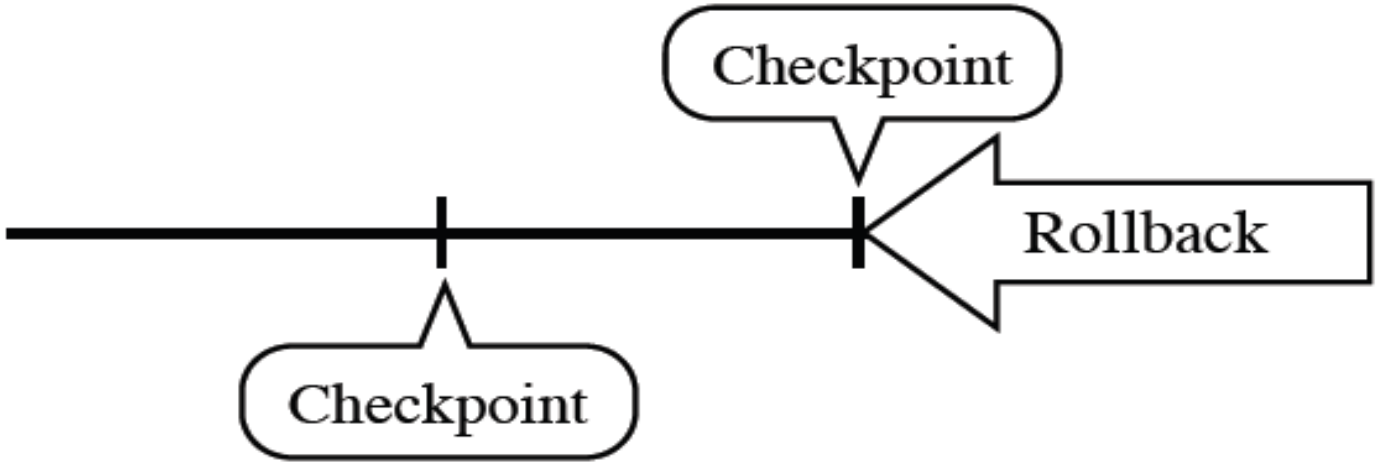
Tipos de Recuperação

- *Backward Error Recovery*
  - Mecanismo que tenta restaurar ou voltar (*rollback*) o sistema para um 'estágio' cujo estado esteja salvo
    - Assume-se que tal estágio esteja antes da falha ocorrer e seja *error-free*
  - O armazenamento dos estados em pontos prédeterminados é chamado de *checkpointing*
  - Deve ser armazenado em locais que não será afetado pelas falhas
  - Ocorreu falha?
    - O sistema volta e recomeça do ponto onde foi '*checkpointed*'

Backward Error Recovery



Backward Error Recovery



Assumption:
Faulty behavior occurred
after last checkpoint

Vantagens - *Backward Recovery*

- É um mecanismo genérico / *application independent*
  - o mecanismo não altera de aplicação para aplicação
  - Possui um padrão uniforme
- Não necessita nenhum conhecimento dos erros
- Pode lidar com erros arbitrários e falhas não previstas durante o processo de desenvolvimento
  - Desde que os estados gravados estejam intactos
- Particularmente voltado para erros transitórios
  - Bastando reicinilizar do último *checkpoint*

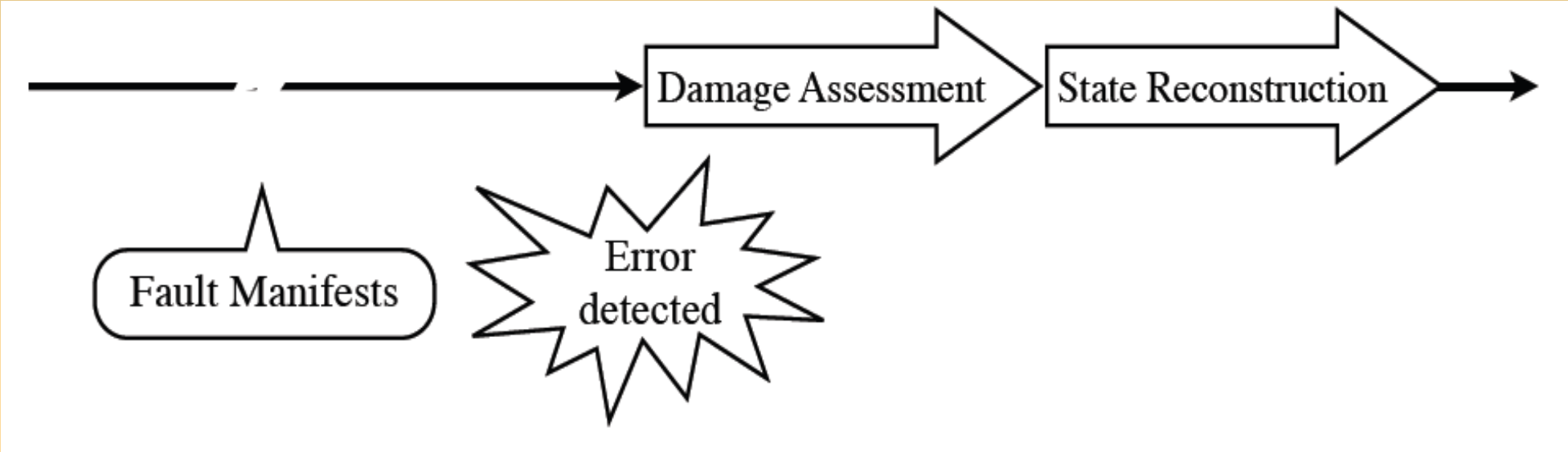
Desvantagens - *Backward Recovery*

- Necessita de recursos extras (e.g. tempo, computação, armazenamento confiável)
  - Para realizar o *checkpoint* e a recuperação
- *Checkpoint* requer:
  - Identificação estados consistentes
  - 'parada' do sistema; *slow down temporarily*
- Cuidados devem ser tomados em sistemas concorrentes
  - Para evitar o efeito 'dominó'
- Apesar dos problemas é o mais utilizado

Forward Error Recovery

- Após a identificação de erros, o sistema tenta transitar para um estado livre de erros
- Este mecanismo tenta atingir a este estado em que o sistema possa continuar operando
- Umas das formas de atingir isso é usar a redundância
  - Várias unidades redundantes são executadas
  - O resultado destas unidades são comparadas
    - Em uma eventual falha, a resposta julgada como correta é então utilizada
  - O módulo de *Error compensation* deriva e/ou julga a resposta 'correta'

Forward Error Recovery



Vantagens - *Forward Recovery*

- Eficiente (tempo e/ou memória)
  - Não precisa do *rollback*
- Se as características das falhas são bem conhecidas, este mecanismo é o mais eficiente
- Voltado particularmente para sistemas tolerantes a falhas
- *Missed deadline?*
  - Então é mais apropriado do que fazer o *rollback*
- Falhas antecipadas como a possível perda de dados na frente
  - Redudância e forward recovery são uma boa solução

Desvantagens - Forward Recovery

- *Application specific* (deve ser projetado para cada aplicação)
- Pode apenas recuperar-se eficientemente de erros 'previstos'
- O mecanismo requer um bom conhecimento dos erros
- Ele também não pode ajudar se o erro 'foge' do que foi especificado no *error compensation module*
- Em virtude disso, não é muito utilizado sobretudo em sistemas distribuídos, onde o *checkpoint* 'domina'

Métricas para TF (up e down)

- TF deve prover confiabilidade
- É importante possuir padrões de medida para estimar tal confiabilidade; duas medidas:
- Confiabilidade (*reliability*)
  - R(t) é a probabilidade de que um software esteja funcionando de [0,t]
- Disponibilidade (*availability*)
  - A(t) é a fração de tempo em que o sistema encontra-se funcionando
  - É importante em ambientes onde o funcionamento total não é vital, mas tem um custo elevado quando 'down'

Necessidade de mais Métricas

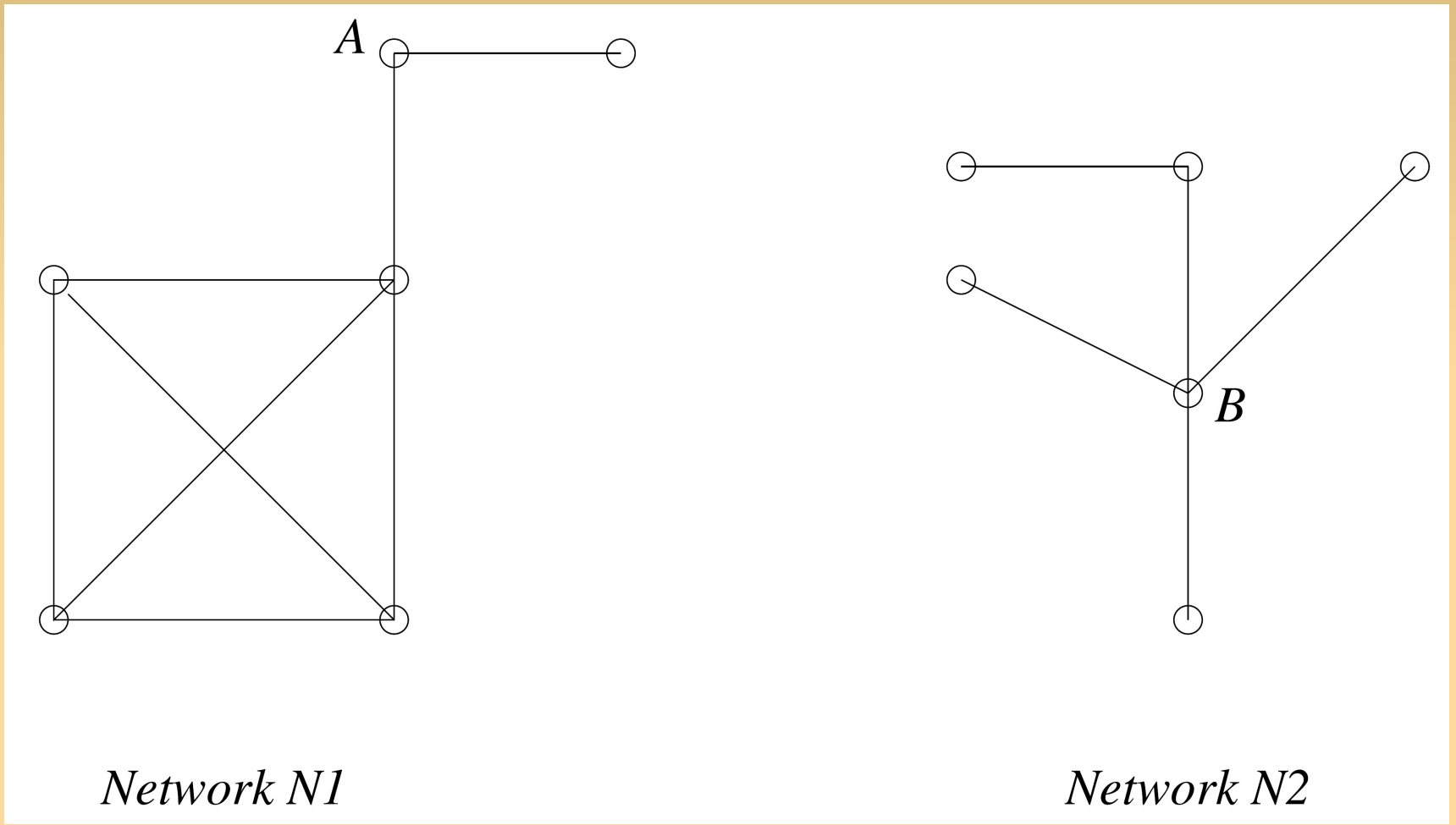
- A hipótese do sistema estar *up* ou *down* é um pouco limitante
- Examplo: Um processador com um de suas várias centenas de milhões de portas em 0 e o resto funcionando - pode afetar a saída do processador de uma vez em cada 25.000 horas de uso

- O processador não é livre de falhas, mas não pode ser definida como sendo 'down'
- Medidas mais detalhadas do que a confiabilidade geral e a disponibilidade são necessários

Métricas para Conectividade

- Foco: rede que conecta os processadores
- A clássica Linha de Conectividade e a Linha - o número mínimo de nós e linhas, respectivamente, que têm a falhar antes da rede ficar desconectado
- Métrica indica a vulnerabilidade da rede quanto à desconexão
- Uma rede desconectada pela falha de apenas um nodo (criticamente posicionado) é potencialmente mais vulnerável do que outra que exige vários nodos a falhar antes de se desconectar

Exemplos de Conectividade

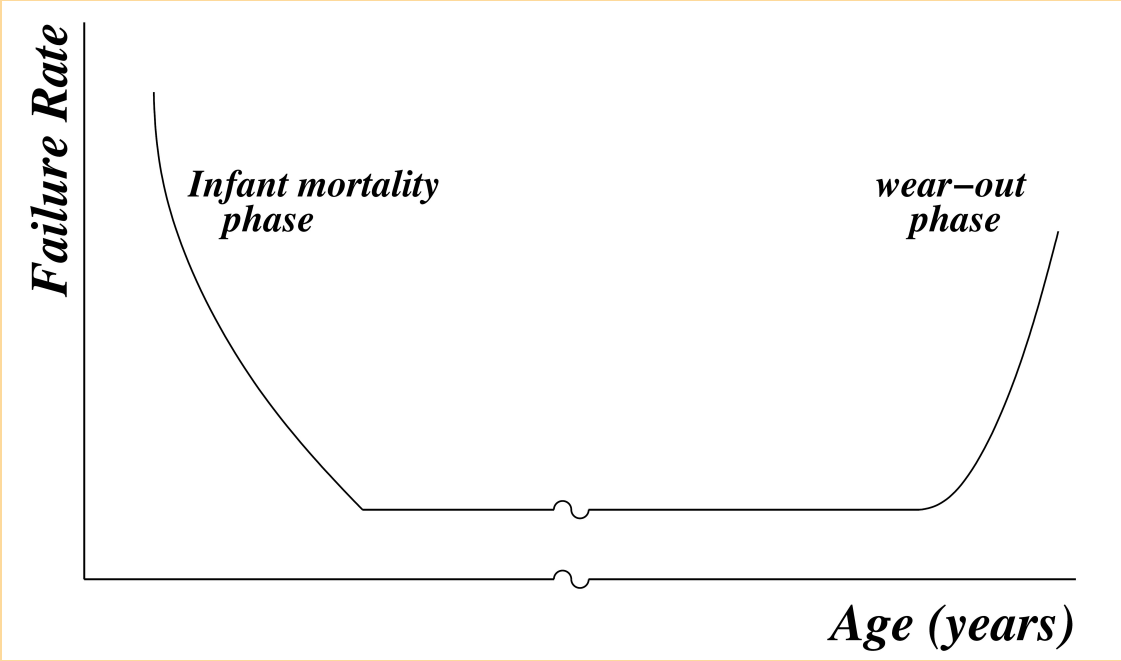


Métricas – Resiliência de Redes

- Conectividade clássica faz a distinção apenas entre os dois estados de rede: conectado e desconectado
- Não diz nada sobre o processo da rede degradar-se quando os nodos se tornam desconectados
- Duas métricas possíveis de serem utilizados
  - Distância média entre os pares de nodos
  - Diâmetro da rede – distância máxima entre os pares de nodos
- Ambos são calculados dado a probabilidade da falha do nodo e do link

Taxa de Falhas em Hardwares

- Rate at which a component suffers faults
- Depende da idade, temperatura ambiente, tensão em volts e/ou choques físicos
- O comportamento das falhas segue - bathtub curve



Bathtub Curve

- Componente novo – alta taxa de falhas
  - Alta chance de que algumas unidades defeituosas passou pelo controle de qualidade de produção e foram liberadas
- Componente 'maduro' – os componentes s/ defeitos tem uma taxa constante de falhas
- A medida que os componentes tornam-se muito 'velhos', os efeitos do envelhecimento podem aumentar a taxa de falhas

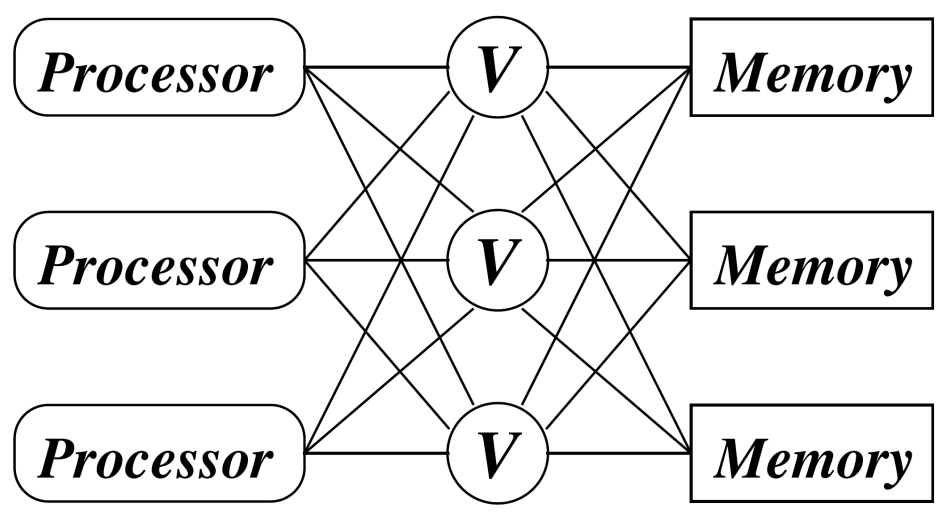
Voters

- Um votador recebe inputs de várias entradas e gera uma saída representativa
- Simplest voter – comparação bit-a-bit produzindo uma saída da maioria
  - Os mais críticos – só avança se todos os outputs forem iguais
- Só funciona se as saídas forem identicas e os processadores sincronizados

Plurality Voting

- Dizemos que duas saídas (X e Y) são idênticas quando se |x-y| < z παραυμ ζ εσπεχψχαδο
- Um k-plurality voter procura por um conjunto de pelo menos k saídas idênticas e pega uma delas (ou calcula a média delas) como representante
- Example – ζ = 0.1, cinco saídas
- 1.10, 1.11, 1.32, 1.49, 3.00
- O conjunto {1.10, 1.11} será selecionado por 2-plurality voter

Triplicated Processor/Memory System

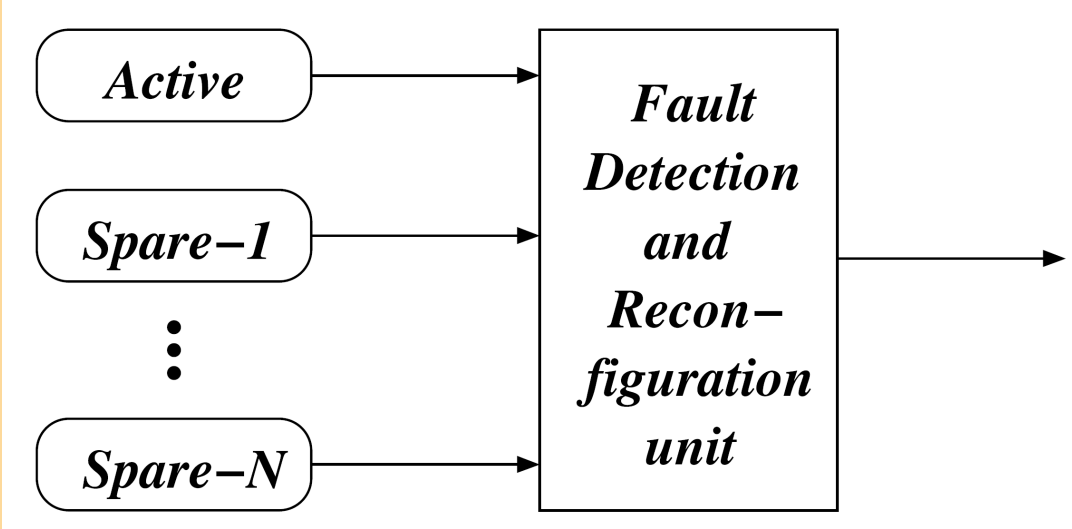


- Todas as conexões entre a CPU e RAM passam por um votador 'triplicado'
- Maior confiabilidade do que o mecanismo com um único votador com CPUs e RAM triplicados.
  - Por que?

Active/Dynamic Redundancy

- Nos modelo anterior – hardware extra considerável para mascarar erros
- Em muitos casos, erros temporários são aceitáveis
  - O sistema pode detectar erros
  - Substituir o módulo com defeito por um outro s/ falhas
  - auto-reconfiguração
- Método chamado de Active/Dynamic Redundancy

Example:



Revisitando...

- O que é um
  - Voter?
  - Acceptance test?
  - Sequencial? Paralelo?
  - Mais complexo (normalmente)?
  - Variante?

Tipos de Redundância para TF

- Um conceito chave na TF
  - recursos adicionais que não serão necessários caso a TF não seja implementada
- Redundância provê funcionalidades adicionalidades e recursos necessários para:
  - Detectar e tolerar falhas
- Tipos de redundância
  - Hardware
  - Software
  - Informação
  - Tempo

Tipos de Redundância para TF

- A redudância de hardware inclui hardwares suplementares e replicados
- Software redundantes pode residir em hardwares replicados para tolerar falhas em hardware e em software.
- Informações redudantes são utilizados para auxiliar na TF em nível de software
- Redudância temporal permite a utilização de tempo adicional para os mecanismos de TF
- Vários tipos de redundância podem ser adotados em uma única técnica de TF (NVP)
  - Pode utilizar três softwares e hardwares redundantes

Redundância de Software

- É mais complexa do que a redundância de hardware porque esta efetivamente resolve as falhas de hardware (mais previsível - *aging*)
- As falhas de software são provindas do projeto e implementação do software (diferente do de hardware)
- Réplicas de software nem sempre resolvem o problema de TF (i.e. réplicas de erros)
- Solução: *diversity* chamados de variantes, versões ou alternativas
  - Introduzido através dos diferentes aspectos de desenvovimento de software

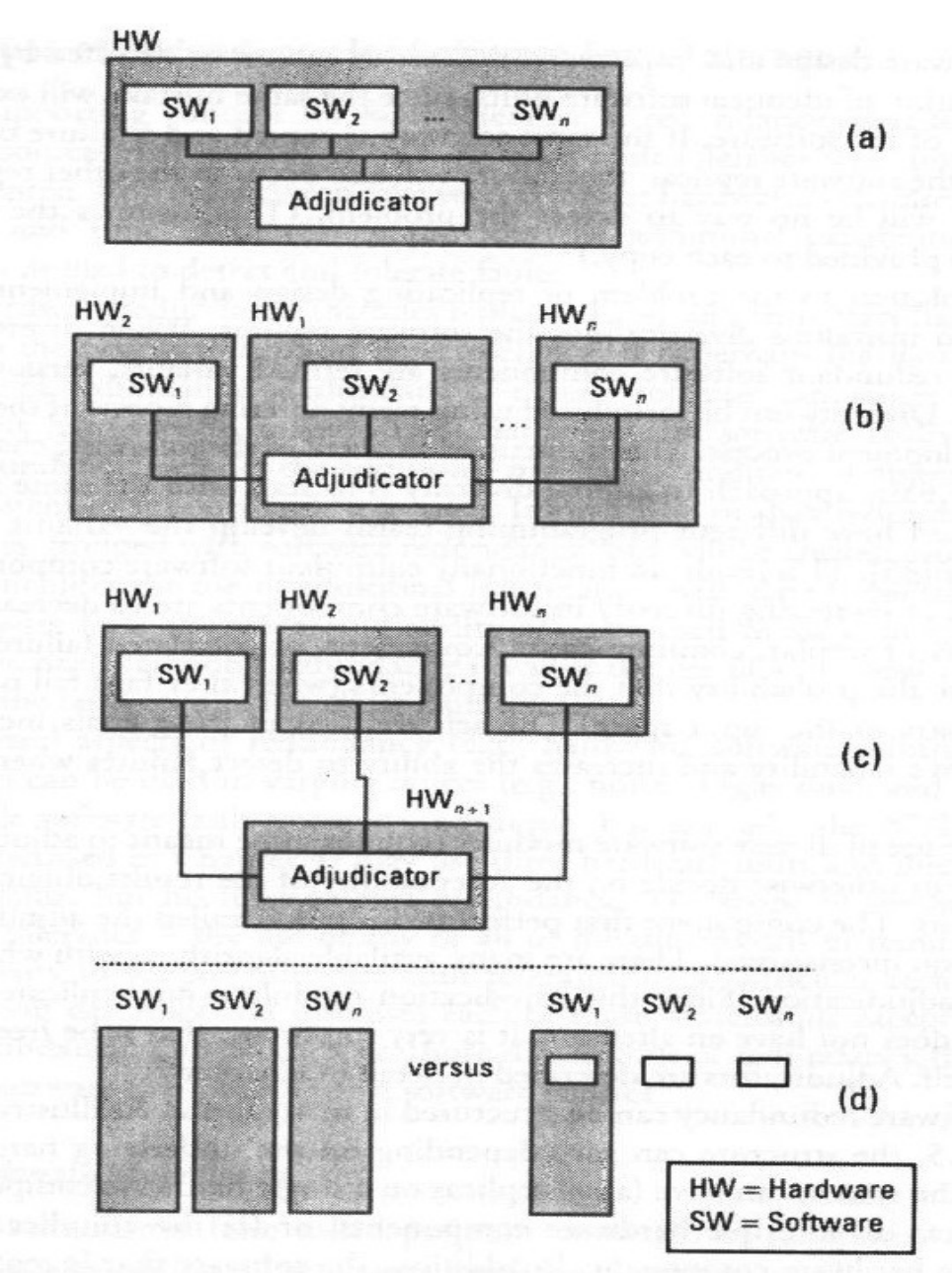
Redundância de Software

- As variantes devem ser desenvolvidas independentemente
  - Diminui a probabilidade de ter-se falhas 'coincidentemente' em pontos similares
  - Aumenta a confiabilidade
- O uso das variantes requer um mecanismo para 'julgar' (*adjudicator*) os resultados provindos de cada módulo replicado
  - Importante: o módulo que 'julga' deve ser *error-free*
  - O *adjudicator* não é replicado

Redundância de Software

- a) Réplicas em um único hardware
- b) Réplica em múltiplos hardwares
- c) O 'juiz' (*adjudicator*) é um hardware diferente
- O software replicado pode incorporar um programa inteiro (software monolítico)
- O software replicado pode incorporar apenas algumas linhas de código (até em Assembly)
- Diferença chave entre componentes de software e softwares orientado a objetos
  - Independe de linguagem (e.g. pode ser em Assembly)

Redundância de Software



Redundância de Dados/Informação

- Às vezes ligados a redudância de SW
- Inclui dados/informação para ajudar na TF
- CRC (*Cyclic Redundancy Check*) na transmissão de redes
  - Usados na transmissão de pacotes em redes TCP/IP
  - *Error detecting codes*
  - *Error correcting codes*
- Data re-expression algorithms
  - 5 div 0 = erro!, mas 5 div 0,00000000000001
  - Usado nas variantes

Redundância Temporal

- Técnica usada em TF de HW e SW
- Computação Redundante
- Essencialmente, usa tempo adicional para executar tarefas com TFs
- Repete a execução de partes do software onde as falhas ocorreram (e.g. atualizar um valor)
- Em sistemas multi-core, um núcleo executa a *primary copy* do código enqt que o outro core executa a mesma tarefa (dentro do deadline)
  - Hardware e software *fault tolerant systems*

Redundância Temporal

- Vantagem é que não requer *sempre* sw e hw adicionais
- Requer apenas tempo para reexecutar o processo que falhou
- É bastante utilizado em applicações onde o tempo encontra-se disponível
  - Sistemas interativos
- Não recomendado para sistemas de tempo-real
  - *miss deadlines?*
- E.g. forward recovery com redundância de sw

Conclusão

- Concluindo...
- Leiam o capítulo 1 do Software Fault Tolerance Techniques and Implementation
- Comecem logo a implementação do trabalho de cada um de vcs

Exercício Prático

- Professor
  - Roteador inteligente tolerante a falhas
  - Usa o algoritmo de *bully* para eleger o líder de um cluster de sensores Sun SPOT
  - O líder é o roteador de todo o cluster de sensores
  - Desenhar o diagrama
- Cada equipe de alunos
  - Derivar uma software básico que explora a tolerância a falhas de forma inteligente
  - Deve ser adaptativo para ressaltar a flexibilidade
  - Ambiente de computação ubíqua