Capítulo 2: Confiabilidade e Tolerância a Falhas

Objetivos

- **Entender os fatores que afetam a confiabilidade de um sistema e introduzir como falhas de projeto de software podem ser toleradas**
- **Introduzir conceitos sobre**
  - Segurança e Dependabilidade
  - Confiabilidade, falhas e Defeitos
  - Modos de Falhas
  - Prevenção de falhas e tolerância a falhas
  - Programação N-Version
  - Redundância Dinâmica

Escopo

Quatro fontes de Defeitos que podem resultar em falhas no sistema:

- **Especificação Inadequada — não será visto**
- **Erros de projeto de software — veremos agora**
- **Falhas no processador — não será visto**
- **Interferência no subsistema de comunicação — não será visto**

Segurança e Confiabilidade

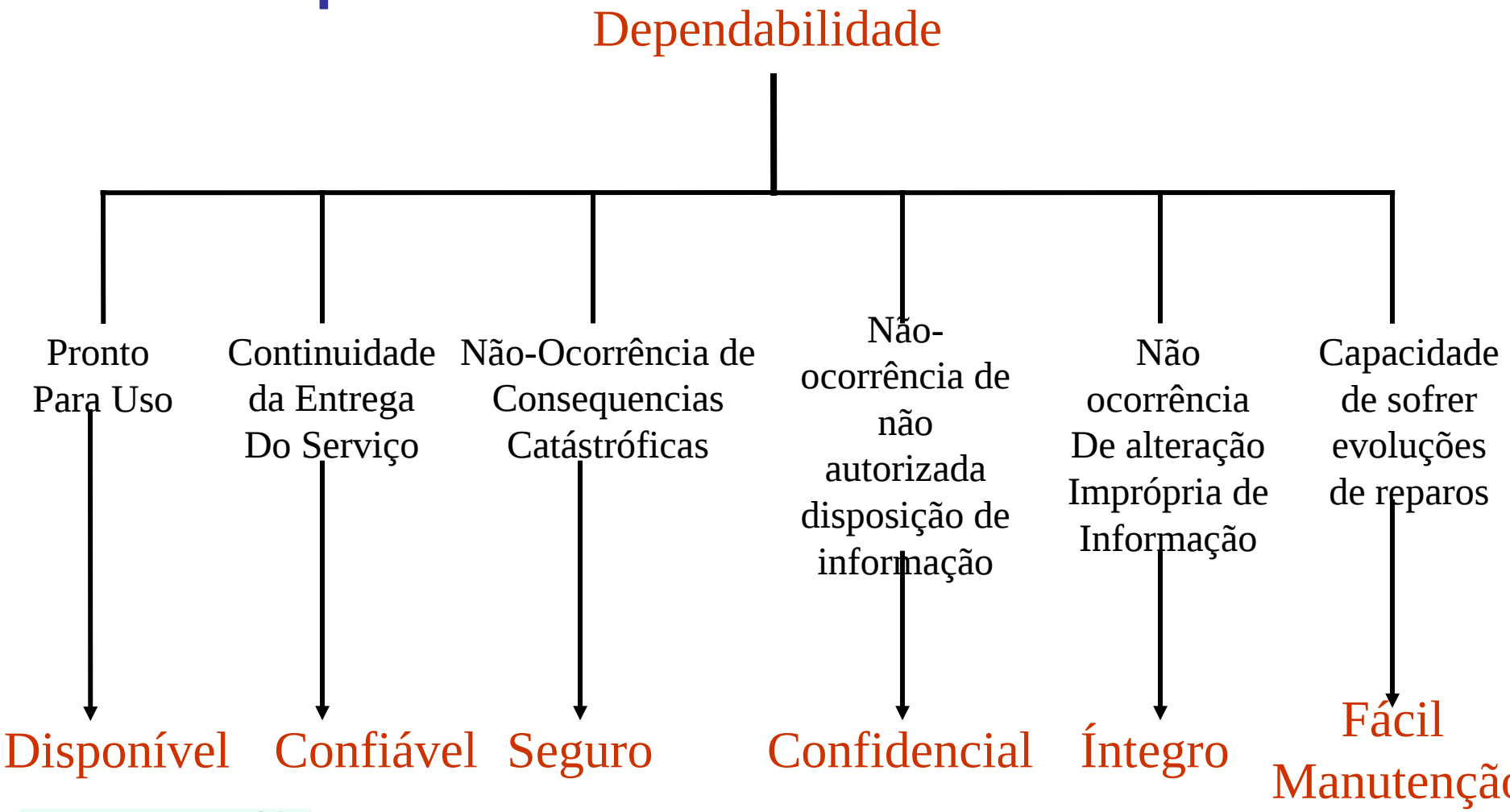
- **Segurança: não ter as condições que causam morte, ferimentos, debilitações ocupacionais, panes ou perdas de equipamentos, agressão ao meio ambiente**
  - Por esta definição, a maioria dos sistemas que tem um elemento de risco associado com seus usos são considerados inseguros
- **Confiabilidade: uma medida do sucesso com o qual um sistema condiz com sua especificação de base do seu comportamento**
- **Segurança é a probabilidade que as condições que levam a contratempos não ocorram se(ou não)a função desejada é executada.**

Segurança

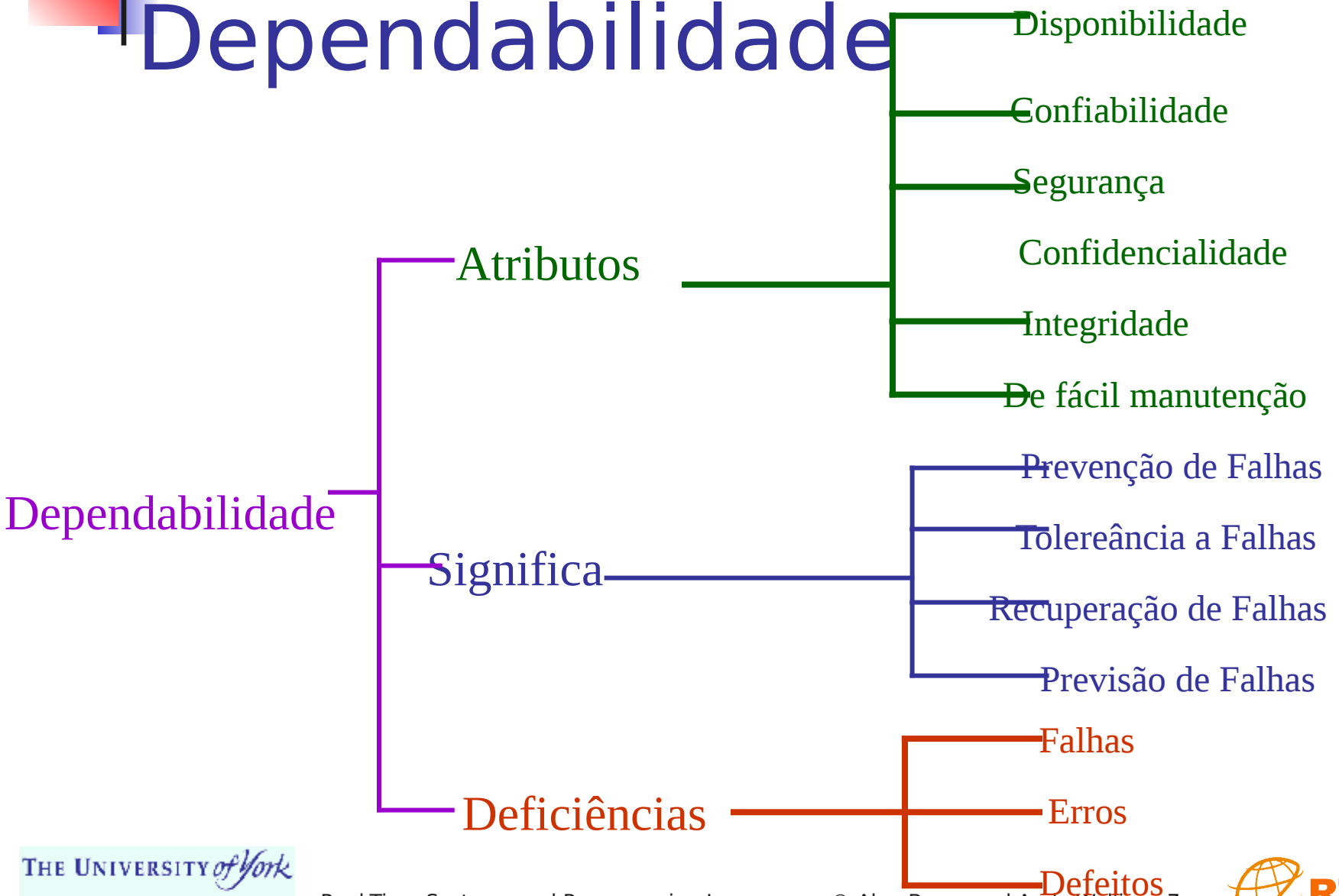
- **Ex.:, medidas que aumentam a garantia de uma arma disparar quando necessário pode também aumentar a possibilidade de sua detonação acidental**
- **De certa forma, o único avião seguro é aquele que nunca decola, entretanto, não é confiável.**

- **Assim como a confiabilidade, para garantir os requisitos de segurança de um sistema embarcado, analises de segurança do sistema devem ser feita em todos os estágios do desenvolvimento do seu ciclo de vida.**

Aspectos de Dependabilidade



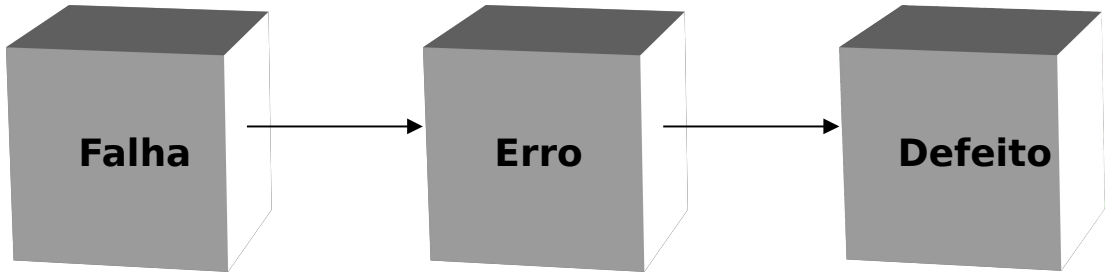
Terminologia da Dependabilidade



Confiabilidade, Defeitos e Falhas

- **A confiabilidade de um sistema é a medida do sucesso com o qual ele condiz com uma especificação autoritativa de seu comportamento**
- **Quando o comportamento de um sistema desvia daquele que foi especificado, isto é chamado um defeito**
- **Defeitos são resultantes de problemas internos ao sistemas inesperados que eventualmente se manifestam no comportamento externo do sistema**
- **Estes problemas são chamados de erros e suas causas mecânicas ou de algorítmo são conhecidas como falhas**
- **Sistemas são compostos de componentes que são em si sistemas, assim:**
  - > defeito -> falha -> erro -> defeito -> falha

Confiabilidade, Defeitos e Falhas



Relacionamento entre Falha, Erro, e Defeito.

Tipos de Falhas

- **Uma falha transiente incia em um tempo particular, se mantém no sistema por algum período e então desaparece**
- **Ex.: componentes de hardware que tem uma reação adversa a radiação**
- **Muias falhas em sistemas de comunicação são transientes**

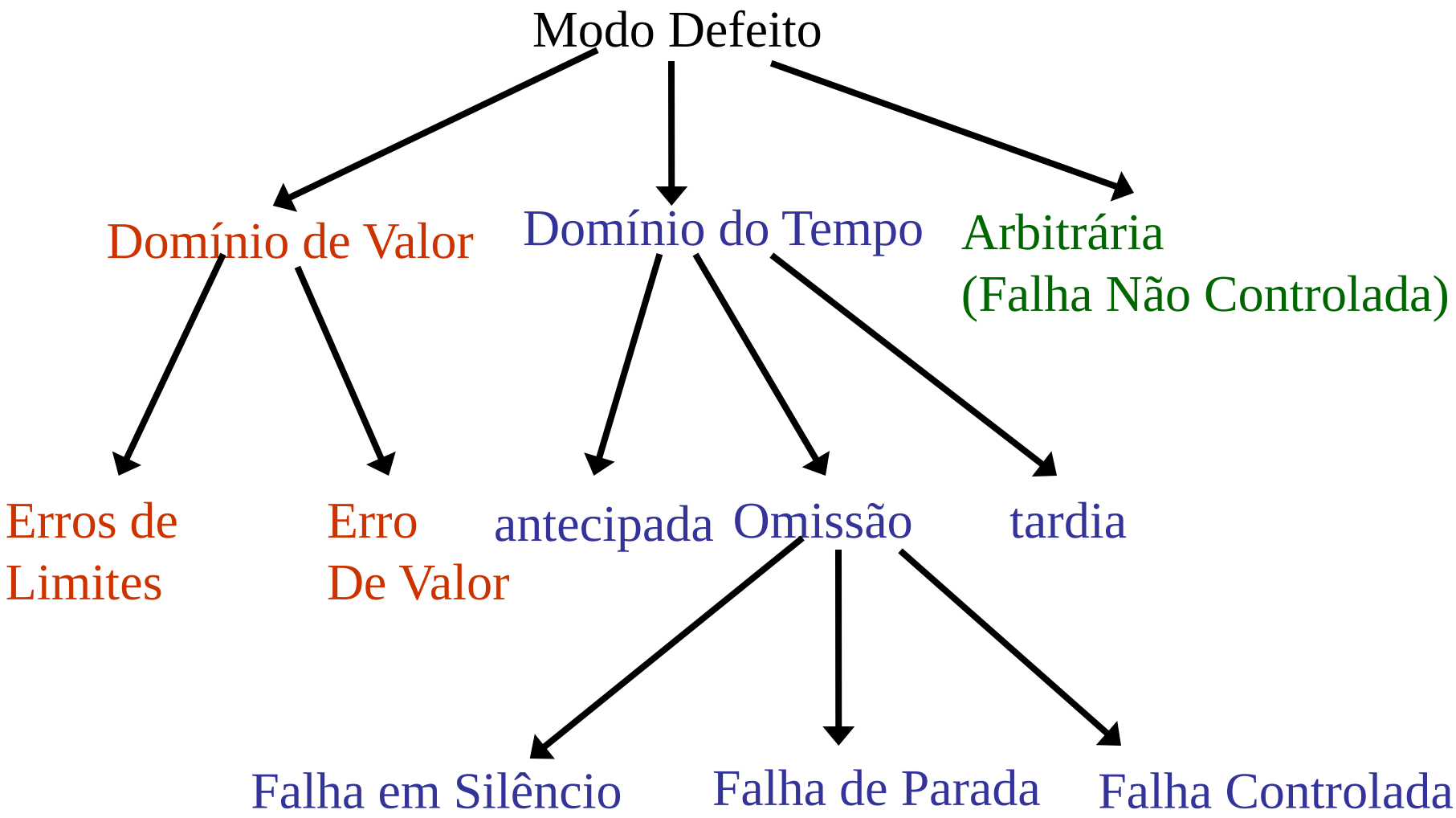
Tipos de Falhas

- **Falhas permanentes se mantem no sistema até que elas sejam reparadas; ex.: um fio quebrado ou um bug no software**
- **Falhas intermitentes são falhas transientes que ocorrem de tempos em tempos**
- **Ex.: componente de hardware que é sensível a calor, funciona por um tempo, para de funcionar, esfriam e voltam a funcionar**

Falhas de Software

- **Chamados de Bugs**
  - Bohrbugs: identificável e reproduzível
  - Heisenbugs: acontece somente em condições raras: ex.: race conditions
- **Software não se desgastam com o tempo: ele é ou correto ou incorreto, porém**
- **Falhas podem ficar despercebíveis por longos períodos de tempo**
  - Normalmente relacionado a uso de recursos ex.: estouros de buffers

Modos Defeito



Técnicas para se ter um Sistema Confiável

- **Prevenção de Falha tenta eliminar qualquer possibilidade de falhas rondando antes que ela aconteça**
- **Tolerância a Falhas permite um sistema continuar funcionando mesmo na presença de falhas**
- **Ambas técnicas tentam produzir sistemas que tem “modos de defeitos” bem definidos**

Prevenção de Falhas

- **Dois estágios: evitar falha and remoção de falhas**
- **Evitar falhas tenta limitar a introdução de falhas durante a construção do sistema através de:**
  - uso da maioria de componentes confiáveis dentro de um dado custo e limites de performance
  - Uso de técnicas rigorosamente definidas para a interconexão dos componentes e montagem dos subsistemas
  - Proteger o hardware para filtrar formas esperadas de interferência.
  - rigorosa, se não formais, especificação de requisitos
  - utilização de metodologias de projeto comprovadas
  - utilização das linguagens com recursos de abstração e modularidade
  - uso de ambientes de engenharia de software para ajudar a manipular os componentes de software e, assim, gerenciar a complexidade

Remoção de Falhas

- **Erros de projeto (hardware e software) vão existir**
- **Remoção de Falhas: procedimentos para encontrar e remover as causas de erros;**
  - ex.: revisões de projeto, verificação de programa, inspeções de código e testes de sistema
- **Teste de sistema podem nunca ser exaustivo e remover todas as falhas potenciais**
- **Um teste só pode ser utilizado para mostrar a presença de defeitos, não a sua ausência**
  - Às vezes, é impossível testar em condições reais
  - A maioria dos testes são feitos com o sistema no modo de simulação e é difícil garantir que a simulação é precisa
  - Erros analisados durante o desenvolvimento do sistema podem não manifestar-se até que torne-se operacional.

Técnicas de Prevenção de Falhas ou Defeitos

- **Apesar de todos os testes e técnicas de verificação, componentes de hardware irão falhar, a abordagem de prevenção de falha, portanto, é sem sucesso quando**
  - a freqüência ou duração dos tempos de reparo são inaceitáveis, ou
  - o sistema é inacessível para as atividades de manutenção e reparação

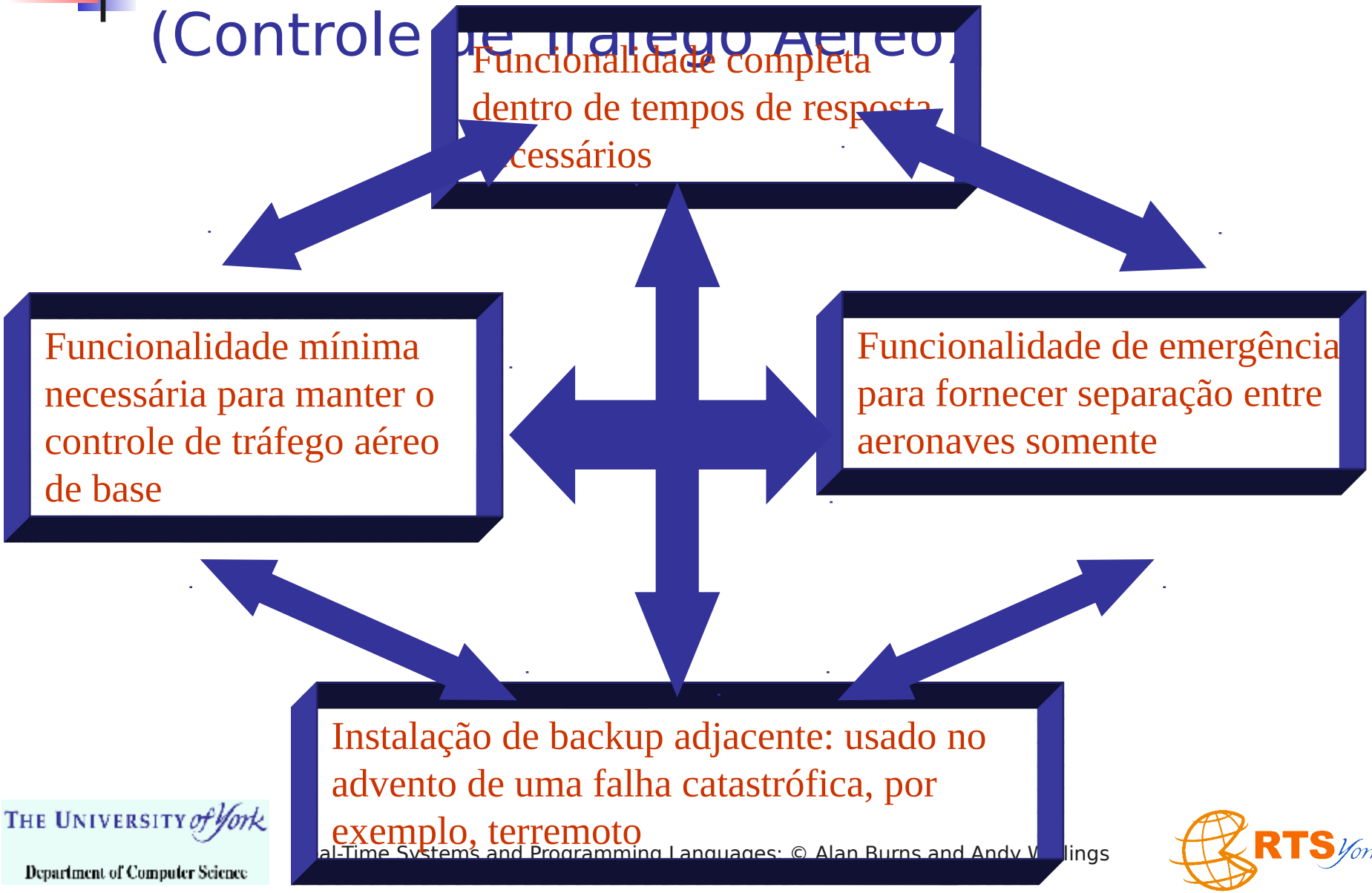
- **Um exemplo extremo do último é a sonda não tripulada Voyager (quando estava a 10 bilhões de milhas do sol!)**

- **Alternativa é Tolerância a Falhas**

Níveis de Tolerância a Falhas

- **Tolerância a Falhas Completa — o sistema continua a operar na presença de falhas, ainda que por um período limitado, sem perda significativa de funcionalidade ou desempenho**
- **Degradação graciosa (falha suave) — o sistema continua a operar, na presença de erros, aceitando uma degradação parcial da funcionalidade ou desempenho durante a recuperação ou reparação**
- **A prova de falhas — o sistema mantém a sua integridade e aceita uma interrupção temporária no seu funcionamento**
- **O nível requerido dependerá da aplicação**
- **A maioria dos sistemas críticos de segurança exigem tolerância a falhas completo, no entanto, na prática, muitos se contentam com a degradação graciosa**

Degradação Graciosa em um sistema ATC (Controle de Tráfego Aéreo)



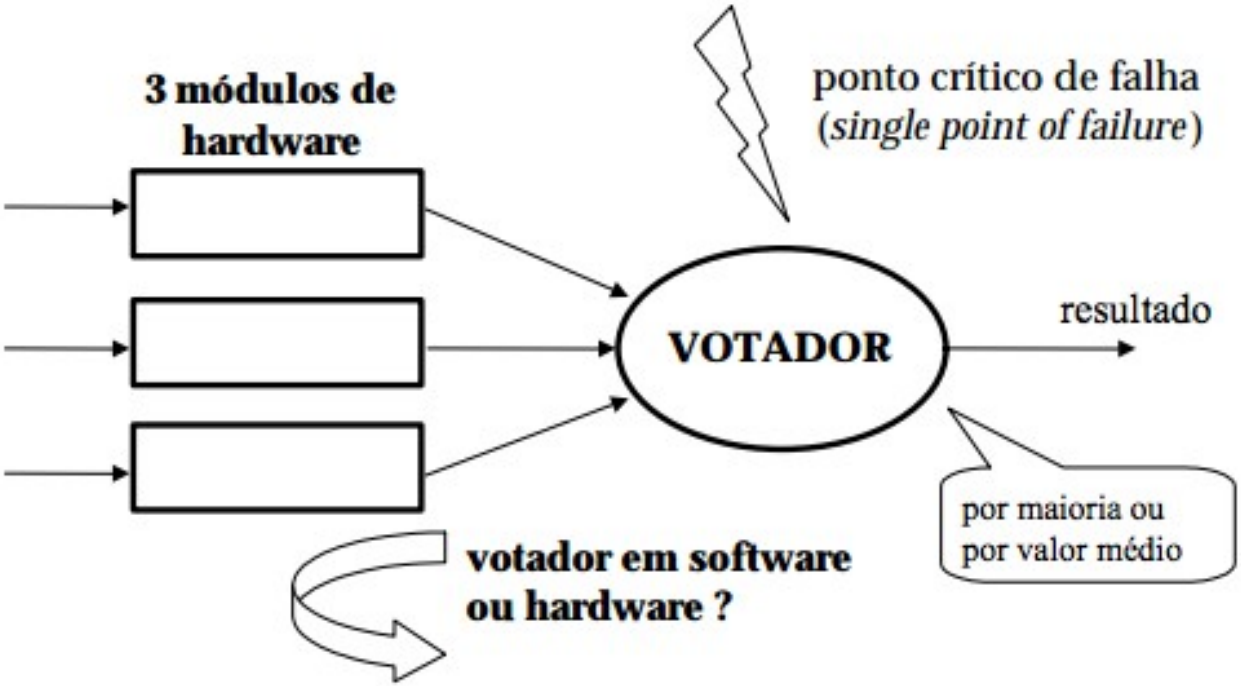
Redundância

- **Todas as técnicas de tolerância a falhas dependem de elementos extra introduzidas no sistema para detectar e recuperar de falhas**
  - Componentes são redundantes, pois não são necessários em um sistema perfeito
  - Frequentemente chamado de redundância de proteção
- **Objetivo: minimizar a redundância, maximizando a confiabilidade, sujeito às restrições de custo e tamanho do sistema**
  - Atenção: os componentes adicionados inevitavelmente aumentam a complexidade do sistema global
  - Isto em si pode levar a sistemas menos confiáveis
  - Ex.:, primeiro lançamento de um ônibus espacial
- **É aconselhável separar os componentes tolerantes a falhas do resto do sistema**

Tolerância a Falhas em HW

- **Dois tipos: redundância estática (ou mascaramento) e dinâmica**
- **Estática: componentes redundantes são usados dentro de um sistema para esconder os efeitos de falhas; ex.: Triple Modular Redundancy**
  - TMR — 3 subcomponentes idênticos e circuitos votação por maioria; as saídas são comparadas e se difere dos outros dois, esta é mascarada
  - Assume que a falha não é comum (como um erro de projeto), mas ou é transitória ou devido à deterioração de componentes
  - Para mascarar falhas de mais de um componente é necessário NMR
- **Dinâmica: redundância fornecida dentro de um componente, que indica que a saída está em erro; proporciona uma facilidade de detecção de erro; recuperação deve ser fornecida por outro componente**
  - Ex.: checksum em comunicações e bits de paridade na memória

Tolerância a Falhas em HW



Sobre TMR

- **Avaliar o artigo Evaluating the Fault Tolerance of Stateful TMR**
  - Como funciona um TMR clássico?
  - O que o artigo propõe? E como funciona?
  - Quais as vantagens com relação ao modelo tradicional?

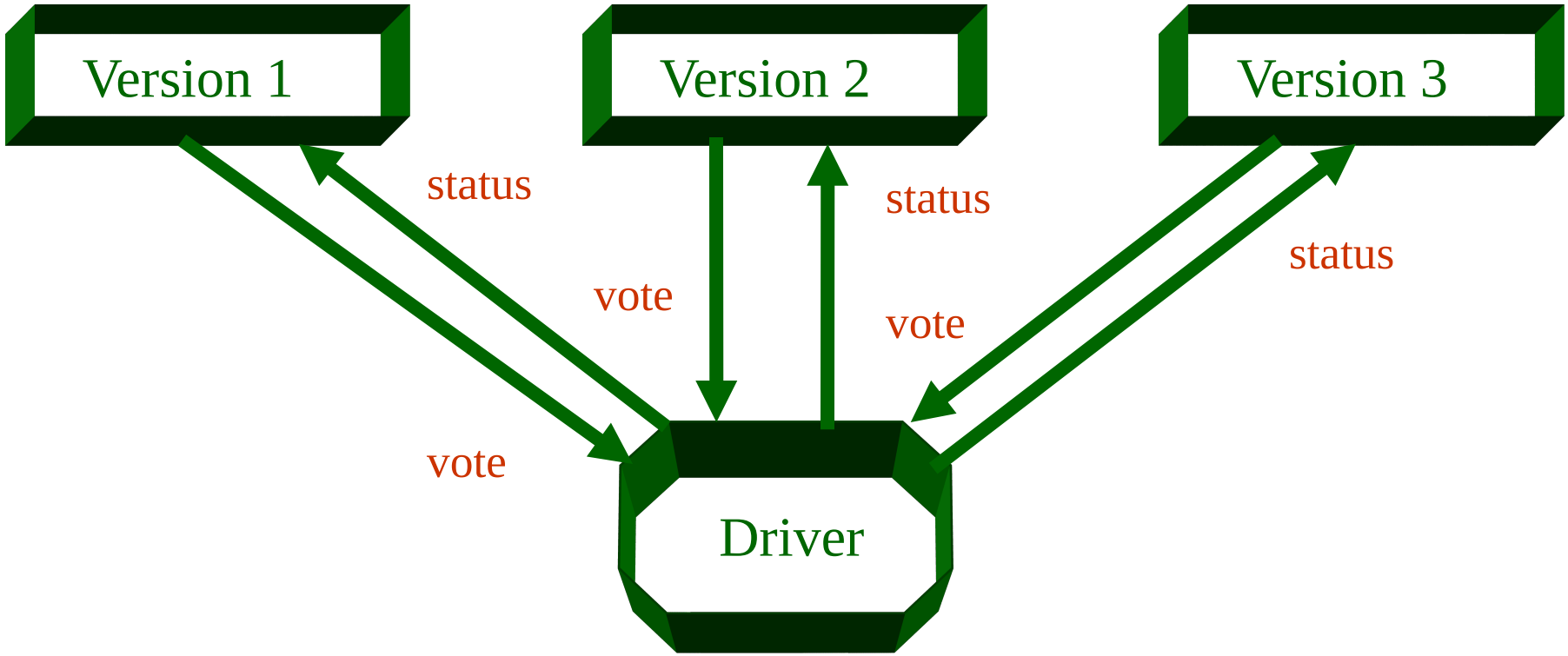
Tolerância a Falhas em SW

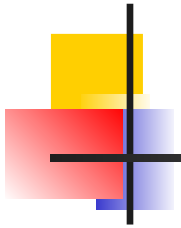
- **Usada para detecção de errros de projeto**
- **Estática — Programação N-Version**
- **Dinâmica**
  - Detecta e Recupera
  - Blocos de recuperação: recuperação de erros backward
  - Exceções: recuperação de erros forward

Programação N-Version

- **Diversidade de Projeto**
- **Geração independente de N (N > 2) funcionalmente equivalentes programas da mesma especificação inicial**
- **Sem interação entre os grupos**
- **Os programas executam concorrentemente com as mesmas entradas e seus resultados são comparados por um processo “driver”**
- **Os resultados (VOTOS) devem ser idênticos, e este é tomados como correto**

Programação N-Version

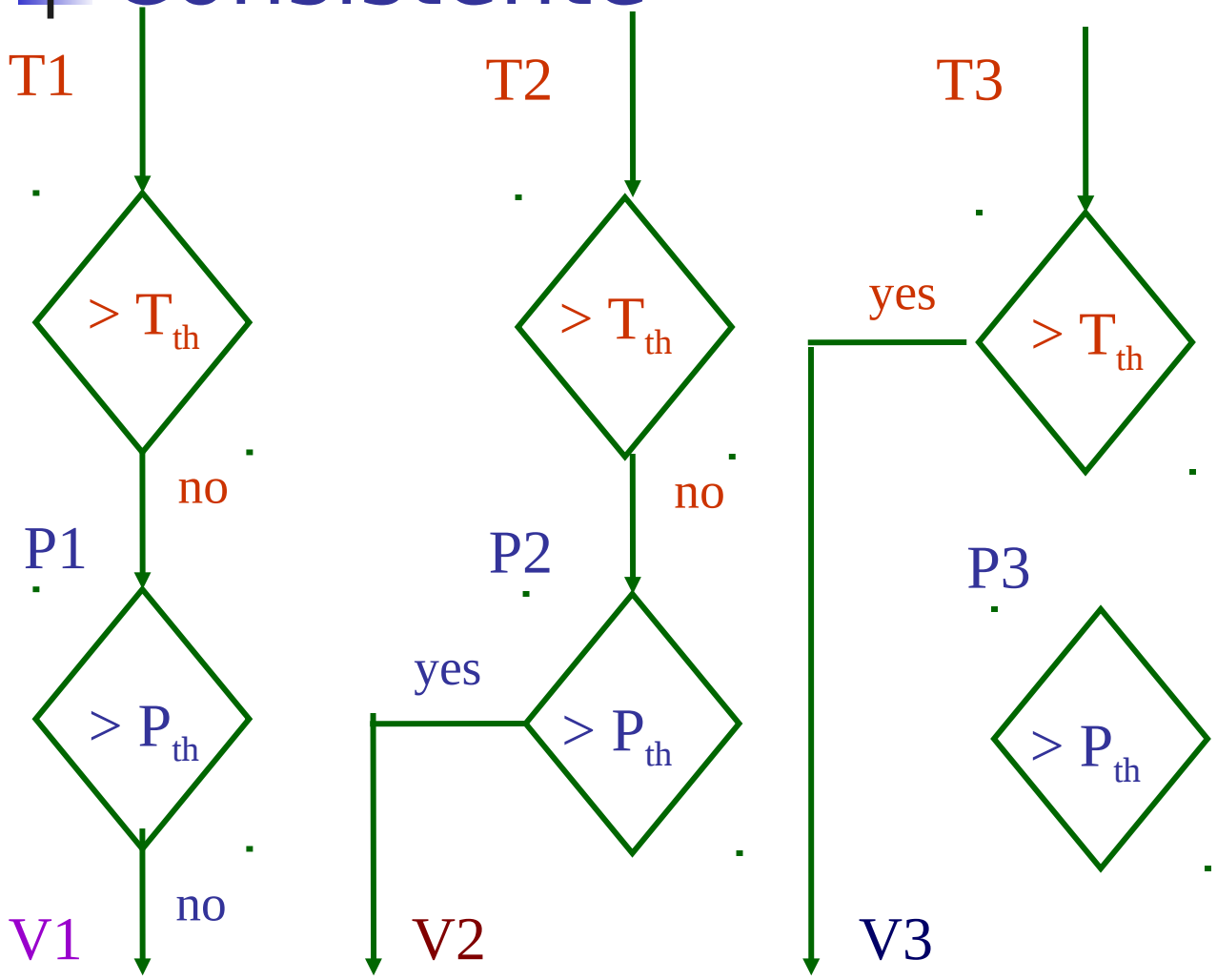




Comparação de Votos

- **Com o que pode ser comparado votos?**
- **Texto ou aritmética inteira irá produzir resultados idênticos**
- **Números Reais => valores diferentes**
- **Necessário técnicas de comparação não necessariamente exatas**

Problema da Comparação Consistente



Cada versão vai produzir um resultado diferente, mas correta

Mesmo se as técnicas de comparação inexatos são utilizados, o problema ocorre

Programação N-version depende de

- **Especificação Inicial — A maioria das falhas de software derivam especificação inadequada**
  - Isto irá se manifestar em todas as N versões de implementação
- **Independência de esforço — Experiências produzem resultados conflitantes**
  - Partes complexas de uma especificação levam a uma falta de compreensão dos requisitos
  - Se há referências a entradas que raramente ocorrem, erros comuns de projeto podem não ser detectados na etapa de testes
- **Orçamento Adequado — O custo maior é o software.**
  - Um sistema 3-versão vai triplicar a exigência de orçamento e causar problemas de manutenção
  - Um sistema mais confiável seria produzido se os recursos potencialmente disponíveis para a construção de um N versões fossem usados para produzir uma única versão?

Uso Militar versus indústria de Aviação Civil

Redundância Dinâmica de SW

Alternativa para redundância estática: quatro fases

- **Detecção de erros — nenhum esquema de tolerância a falhas pode ser utilizado até que o erro associado é detectado**
- **confinamento de danos e avaliação — em que medida o sistema foi corrompido?**
  - O atraso entre a ocorrência de falhas e a detecção do erro, informação com erro poderia espalhar por todo o sistema
  - recuperação de erro — técnicas devem procurar transformar o sistema corrompido em um estado do qual ele pode continuar a sua operação normal (talvez com perda de funcionalidade)
- **tratamento de falhas e serviço continuado — um erro é um sintoma de uma falha, embora o dano seja reparado, a falha ainda pode existir**

Detecção de Erros

- **Detecção no Ambiente**
  - hardware — ex.: instrução ilegal
  - O.S/RTS — ponteiro nulo
- **Detecção na Aplicação**
  - Verificadores de Replicação
  - Verificadores por Tempo (ex.: watch dog)
  - Verificadores Reversos
  - Verificadores de Código (dados redundantes (extras), ex.: checksums)
  - Verificações de razoabilidade (afirmação, por exemplo)
  - Verificadores Estruturais (por exemplo, os ponteiros redundantes numa lista encadeada)
  - Verificação da razoabilidade dinâmica

Confinamento de danos e Avaliação

- **Avaliação dos danos está intimamente relacionado com as técnicas usadas de confinamento de danos**
- **Confinamento de dano está relacionado com a estruturação do sistema de forma a minimizar o dano causado por um componente defeituoso (também conhecido como firewall)**
- **Decomposição modular oferece confinamento de danos estático; permite que os dados fluam através de caminhos bem definidos (assumindo linguagem fortemente tipada)**
- **Ações atômicas fornecem confinamento de danos dinâmica, eles são usados para mover o sistema de um estado consistente para outro**

Recuperação de Erro

- **Provavelmente a fase mais importante de qualquer técnica de tolerância a falhas**
- **Duas abordagens: forward e backward**
- **Forward error recovery continua de um estado errôneo, fazendo correções seletivas para o estado de sistema operacional**
  - Isto inclui fazer o ambiente seguro e controlado que pode ser perigoso ou danificado por causa do defeito
  - É um sistema específico e depende de previsões precisas do local e causa de erros (ou seja, avaliação de danos)
  - Exemplos: ponteiros redundantes em estruturas de dados e o uso de auto-correção de códigos, tais como códigos de Hamming

Backward Error Recovery (BER)

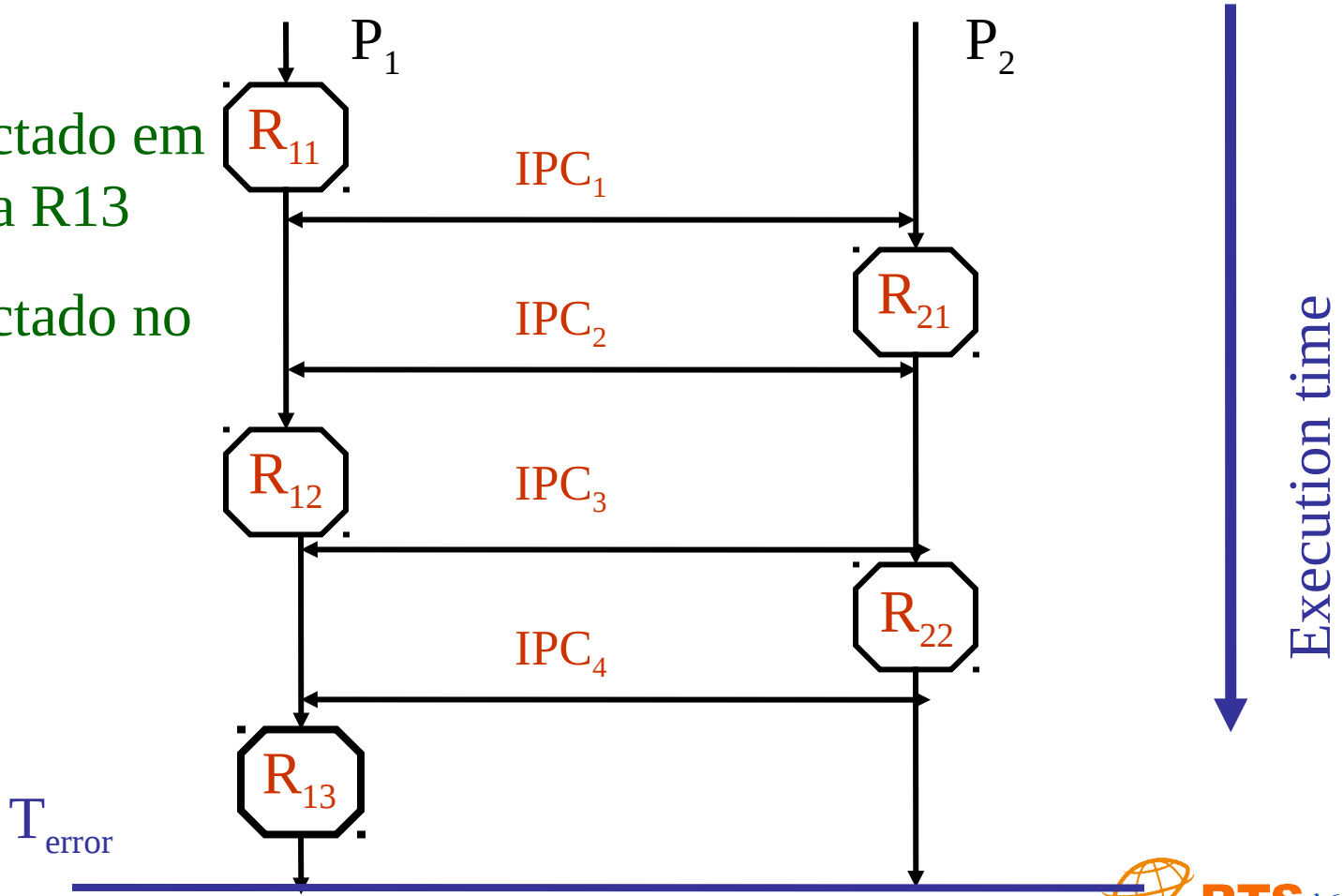
- **BER depende de restaurar o sistema para um estado anterior seguro e execução de uma seção alternativa do programa**
- **Tem a mesma funcionalidade, mas usa um algoritmo diferente (ver Programação N-Version) e, portanto, sem falha**
- **O ponto no qual um processo é restaurado é chamado de ponto de recuperação e o ato de estabelecer é denominado de checkpointing (salvando o estado do sistema)**
- **Vantagem: o estado errôneo desaparece e não depende de encontrar o local ou a causa da falha**
- **BER pode, portanto, ser utilizado para recuperar falhas imprevistas incluindo erros de concepção**
- **Desvantagem: não pode desfazer erros de ambiente de operação!**

O Efeito Dominó

- **Com processos concorrentes que interagem uns com os outros, BER é mais complexa. considere:**

Se o erro é detectado em P1 ele volta para R13

Se o erro é detectado no P2?



Tratamento de falhas e Continuidade do serviço

- **Recuperação de erro retornando o sistema para um estado livre de erros, no entanto, o erro pode ocorrer novamente, a fase final do T.F. é erradicar a falha do sistema**
  - O tratamento automático de falhas é difícil e sistema específico
  - Alguns sistemas assumem que todas as falhas são transitórias, outras que as técnicas de recuperação de erros pode lidar com falhas recorrentes
- **Tratamento de falhas pode ser dividido em duas fases: localização de falhas e reparação do sistema**
  - As técnicas de detecção de erro podem ajudar a identificar a falha de um componente. Para, o hardware do componente pode ser substituído
  - Uma falha de software pode ser removida em uma nova versão do código
  - Em non-stop aplicatições, será necessário modificar o programa enquanto ele está executando!

A abordagem de recuperação de blocos em T.F.

- **Suporte da linguagem para BER**
- **A entrada de um bloco é um ponto de recuperação automática e a saída um teste de aceitação**
- **O teste de aceitação é usado para testar se o sistema está num estado aceitável após a execução do bloco (módulo principal)**
- **Se o teste de aceitação falhar, o programa é restaurado para o ponto de recuperação, no início do bloco e um módulo alternativo é executado**
- **Se o módulo alternativa também falhar o teste de aceitação, o programa é restaurado para o ponto de recuperação e ainda um outro módulo é executado, e assim por diante**
- **Se todos os módulos falharem, então o bloco de falha e recuperação deve ter lugar a um nível mais elevado**

Recovery Block Syntax

```
ensure <acceptance test>
by
        <primary module>
else by
        <alternative module>
else by
        <alternative module>
        ...
else by
        <alternative module>
else error
```

- **Blocos de recuperação podem ser aninhados**
- **Se todas as alternativas em um bloco de recuperação aninhado falharem no teste de aceitação, o ponto de recuperação de nível externo será restaurada e um módulo alternativo ao bloco executado**

Example: Solution to Differential Equation

```
ensure Rounding_err_has_acceptable_tolerance
by
   Explicit Kutta Method
else by
   Implicit Kutta Method
else error
```

- **Método Explícito de Kutta é rápido mas é impreciso quando as equações são rígidas**
- **O Método Implícito de Kutta mais despendioso mas pode lidar com equações mais rígidas**
- **O código acima aceita os dois métodos**
- **que também irá potencialmente tolerar erros de projeto no Kutta explícito se o teste de aceitação é flexível o suficiente**

O Teste de Aceitação

- **O teste de aceitação fornece o mecanismo de detecção de erro, que é crucial para a eficácia do regime de B.R.**
- **Há um trade-off entre oferecer testes de aceitação e manter sobrecarga ao mínimo, para que a parte sem-falhas de execução não seja afetada**
- **Note-se que o termo usado é aceitação de não exatidão, o que permite um componente fornecer um serviço degradado**
- **Todas as técnicas de detecção de erro discutidos anteriormente podem ser utilizadas para formar os testes de aceitação**
- **Deve ser tomado cuidado em como um teste de aceitação defeituoso pode conduzir a erros**

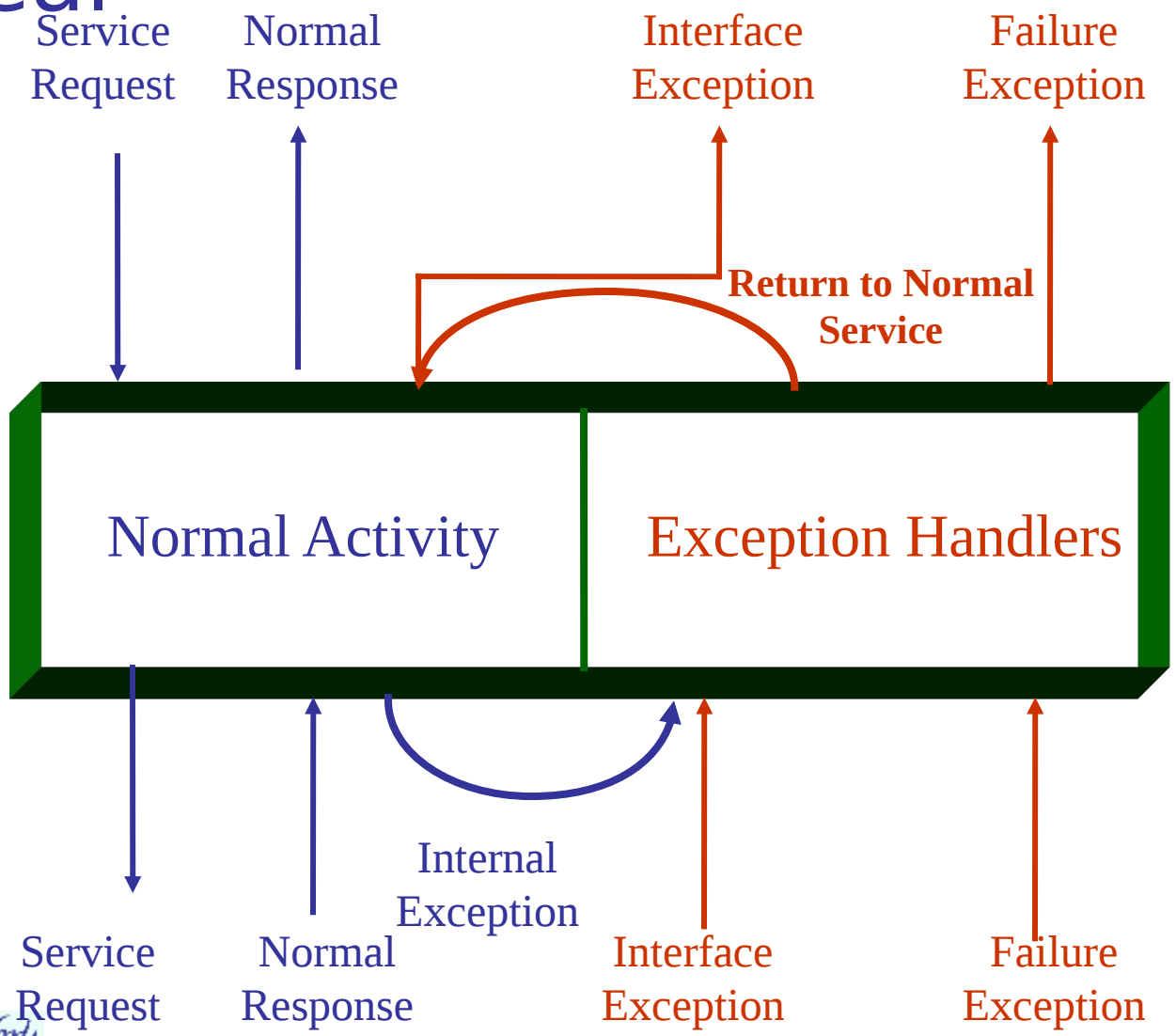
Programação N-Version vs Blocks de Recuperação

- **Redundância Estática (NV) versus Dinâmica (B.R.)**
- **Sobrecarga de Projeto — ambos requerem algorítmos alternativos, NV requer driver, BR requer teste de aceitação**
- **Sobrecarga de Execução — NV requer N * recursos, BR requer pontos de estabelecimento de recuperação**
- **Diversidade do Projeto — ambos suceptíveis a erros nos requisitos**
- **Detecção de Erro — comparação de voto (NV) vs teste de aceitação(BR)**
- **Atomicidade — NV vota antes de gerar a saída para o ambiente, BR deve ser configurado para somente liberar a saída após a passagem de um teste de aceitação**

Redundância Dinâmica e Exceções

- **Manipulação de exceção é um mecanismo de forward error recovery, como não há nenhum rollback para um estado anterior, em vez disso o controle é passado para o manipulador de modo que os procedimentos de recuperação possa ser iniciado**
- **Tratamento de exceção pode ser usado para:**
  - perante condições anormais que surgem no ambiente
  - permitir falhas de projeto do programa serem toleradas
  - fornecer detecção de erros de propósito geral e facilidades de recuperação

Componente Tolerante a Falha Ideal



Sumário I

- **Definidos dependability, safety, reliability, failure, faults**
- **Falhas:**
  - acidentais ou intencionalmente
  - transiente, permanente or intermitente
- **Prevenção de Falha consiste de evitar falha e remoção de falha**
- **Tolerância a Falha: estática e dinâmica**
- **Programação N-version: geração independente de N programas funcionalmente equivalentes a partir da mesma especificação**
- **Baseado na permissa que um programa pode ser completamente, consistentemente e não ambiguamente especificado, e que programas que são desenvolvidos independemente falharão independentemente**

Sumário II

- **Redundância Dinâmica: detecção de erro, confinamento de danos e avaliação, recuperação de erros, e tratamento de falhas e serviço continuado**
- **Com backward error recovery, é necessário para os processos de comunicação para alcançar os pontos de recuperação consistentes para evitar o efeito de dominó**
- **Para sistemas seqüenciais, o recovery block é um conceito de linguagem apropriada para BER**
- **Embora forward error recovery é específico do sistema, exception handling tem sido identificada como uma estrutura apropriada para a sua aplicação**
- **O conceito de um ideal fault tolerant component é introduzido com o uso de exceções.**