



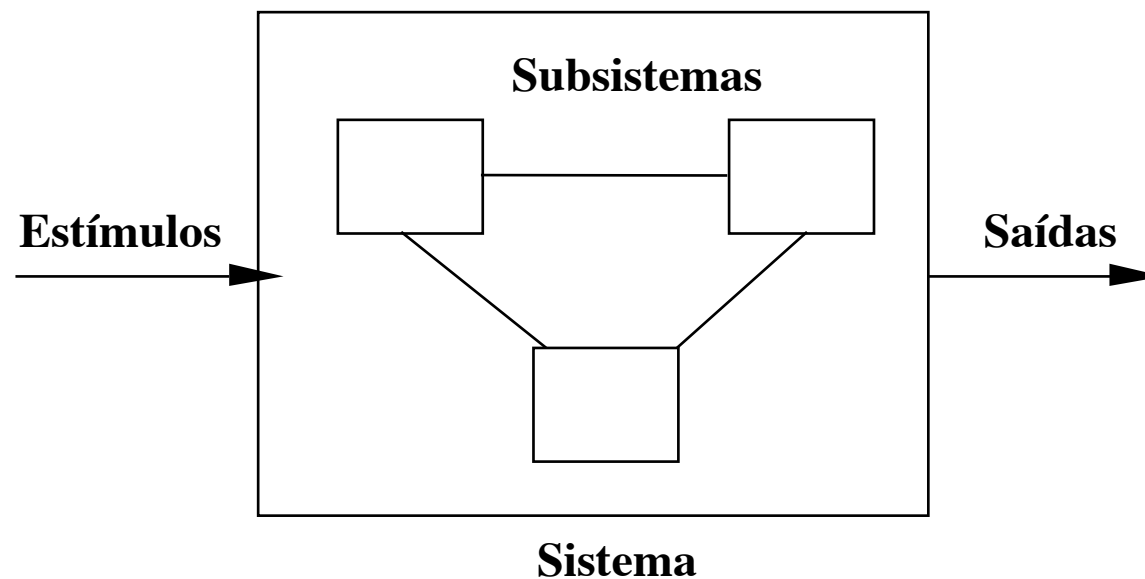
Tolerância a faltas



Tolerância a faltas

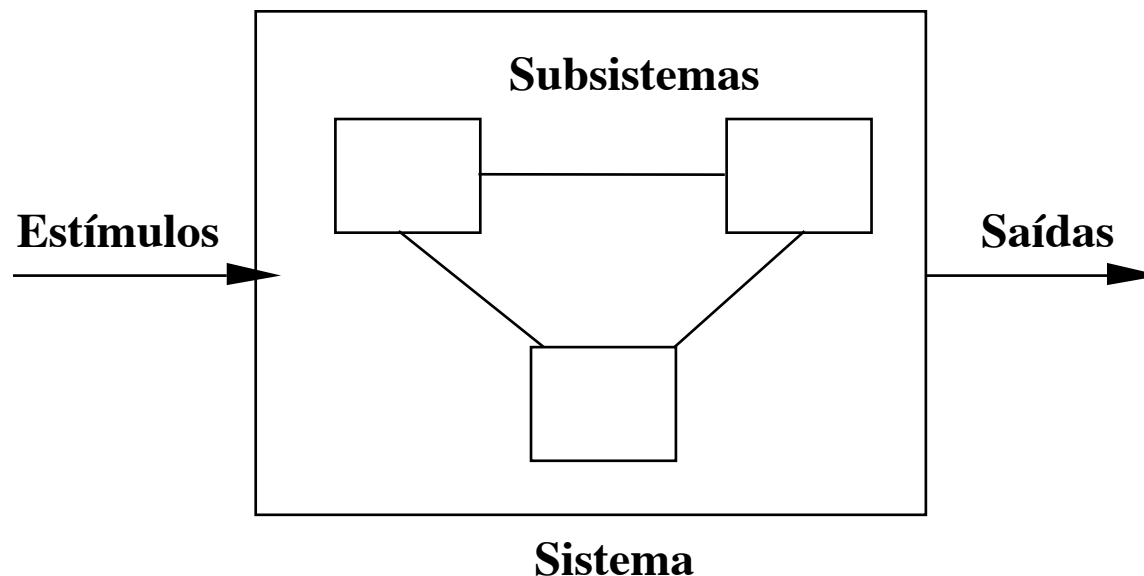
Terminologia

Sistema computacional



Um **sistema** tem uma especificação funcional do seu comportamento que define, em função de determinadas **entradas** e do seu **estado**, quais são as **saídas**.

Sistema computacional determinístico



Sistema determinístico: se as saídas e o estado seguinte forem uma **função (determinística)** dos estímulos e do estado atual

Falta, Erro, Falha

Falta (*fault*):

- Acontecimento que altera o padrão normal de funcionamento de uma dada componente do sistema

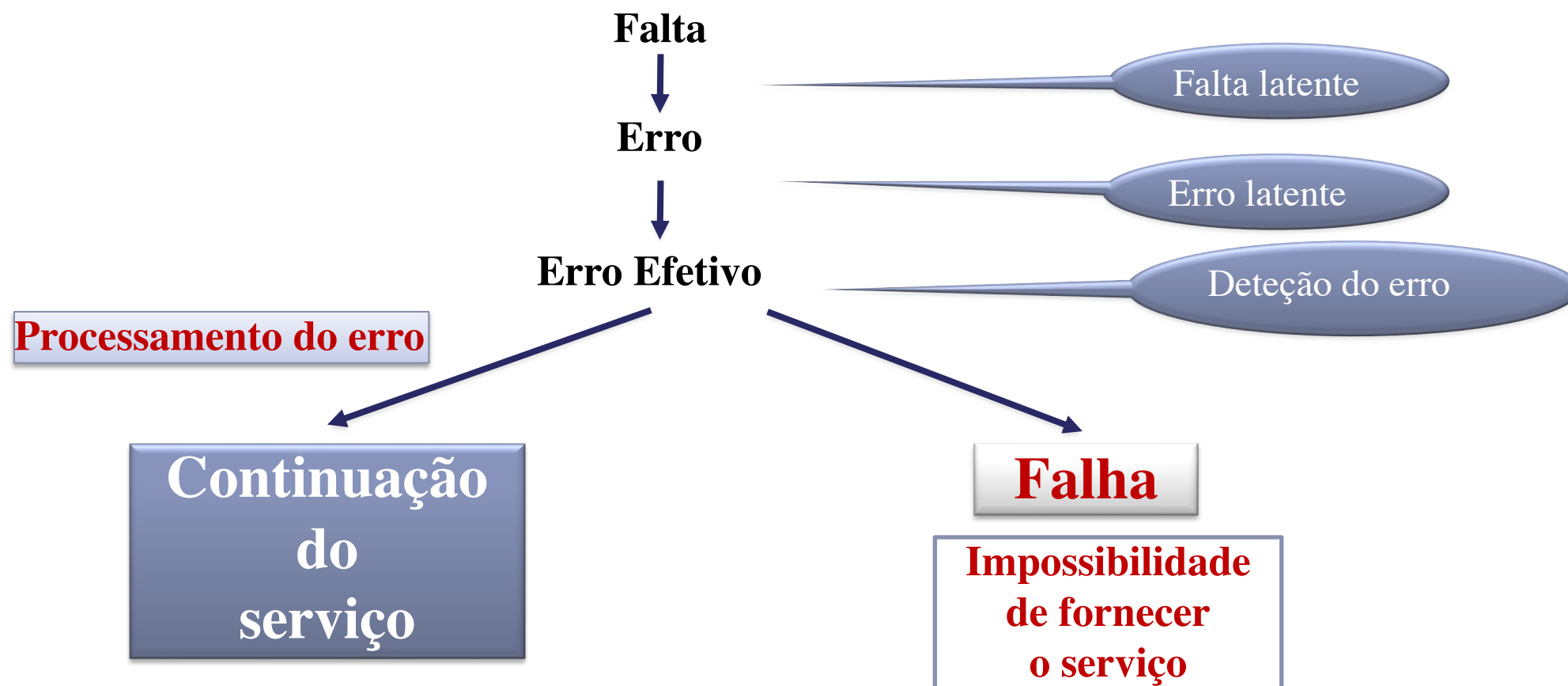
Erro (*error*):

- Transição do sistema, provocada por uma falta, para um estado interno incorreto
 - Estado interno inadmissível
 - Estado interno admissível, mas não o especificado para estas entradas

Falha (*failure*):

- Quando se desvia da sua especificação de funcionamento
- Num determinado estado, o resultado produzido por uma dada entrada não corresponde ao esperado

Modelo de base da tolerância a faltas



Exemplo: bit de memória preso (*stuck to one*)

Falta:

- Posição de memória em que um bit fica sempre com o valor 1
- **Falta latente** pois não dá origem a erro se:
 - Esta posição de memória não for utilizada
 - Se não for escrito um 0 naquele bit

Erro:

- Escrita de um octeto com o bit a 0
- Detectado pelo bit de paridade erro
- Possível evolução :
 - Erro processado (ex: código corrector de erros da memória) => Falta foi tolerada
 - Erro não processado => **Erro fica latente** até esta posição de memória ser lida

Falha:

- Leitura de um valor incorreto da posição de memória
- O erro torna-se efetivo e o sistema de memória falha, não funciona de acordo com o especificado

Exemplo: *bug software*

Falta:

- Engano de um programador ao definir a lógica do programa colocando uma instrução errada
- **Falta latente** enquanto instrução não é executada

Erro:

- Execução da instrução errada
- **Erro fica latente** até se manifestar uma falha do programa (ex.: dado incorreto na base de dados, variável com o valor errado, etc.)

Falha:

- O erro torna-se efetivo e o programa falha

Exemplo de falta: o primeiro bug



Mark II, general view of calculator frontpiece, 1948.


9/9

0800 Antan started
1000 " stopped - antan ✓

1300 (032) MP-MC ~~1.58264000~~ 1.58264000
(033) PRO 2 2.130476415
convd 2.130676415

Relays 6-2 in 033 failed special speed test
in relay 10,000 test.

1100 Relays changed
Started Cosine Tape (Sine check)
1525 Started Multi Adder Test.

1545  Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

1630 Antan started.
1700 closed down.

Relay 2145
Relay 3376



Tolerância a faltas

Classificação de faltas



Classificação de faltas

- A classificação de faltas permite **descrever as suas características**
- Faltas podem ser classificadas por:
 - Causa
 - Origem
 - Duração
 - Independência
 - Determinismo



Tipos de faltas

Causa

- Falta **física**
 - Fenómenos elétricos, mecânicos, ...
- Falta **humana**
 - Acidental: mau desenho, má operação, ...
 - Intencional: ataque premeditado (consideradas no capítulo de Segurança)



Tipos de faltas

Origem

- Falta **interna**
 - Componentes internos, programa, ...
- Falta **externa**
 - Temperatura elevada, falta de energia, ...



Tipos de faltas

Duração

- Faltas **permanentes**:
 - Mantêm-se enquanto não forem reparadas (ex.: cabo de alimentação desligado)
 - Fáceis de detetar
 - (Normalmente) difíceis de reparar
- Faltas **temporárias** ou **transientes**:
 - Ocorrem apenas durante um determinado período, geralmente por influência externa
 - Difíceis de reproduzir, detetar
 - Normalmente fáceis de reparar
 - As faltas transientes ficam reparadas imediatamente após terem ocorrido (ex.: perda de mensagem)



Tipos de faltas

Independência

- Faltas **independentes**:
 - Probabilidade de falta de uma componente é independente das outras componentes
 - Em geral, boa aproximação no *hardware*
- Faltas **dependentes**:
 - Probabilidades de falta correlacionadas
 - Exemplos:
 - Faltas no *software*, se for idêntico em várias máquinas
 - Múltiplos componentes *hardware* a correr no mesmo local, sujeitos à mesmas faltas externas
 - Incêndios, falhas de energia, roubos, etc.

Tipos de faltas

Determinismo

- Faltas **determinísticas**:
 - Dependem apenas da sequência de entradas (*inputs*) e do estado
 - Repetindo essa sequência, reproduzimos a falta
- Faltas **não-determinísticas** (“*Heisenbugs*”):
 - Dependem de outros fatores não determinísticos
 - E.g., escalonamento de *threads*, leituras do relógio, ordem de entrega de mensagens
 - Difíceis de reproduzir, depurar
 - Porque “desaparecem” quando se tenta inspecionar

- Faltas classificadas por:
 - Causa
 - Física, humana
 - Origem
 - Interna, externa
 - Duração
 - Permanente, temporária
 - Independência
 - Independente, dependente
 - Determinismo
 - Determinísticas, não determinísticas

Qual o **pior
tipo de falta?**



Tolerância a faltas

Métricas



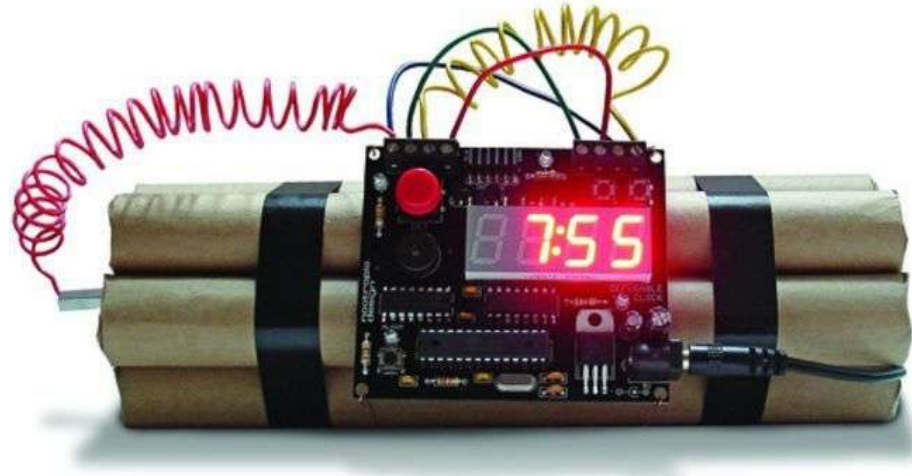
Métricas para a tolerância a faltas

- Permitem
 - **Quantificar** quão tolerante a faltas é um sistema
 - **Comparar** sistemas ou configurações diferentes



Métrica: fiabilidade (*reliability*)

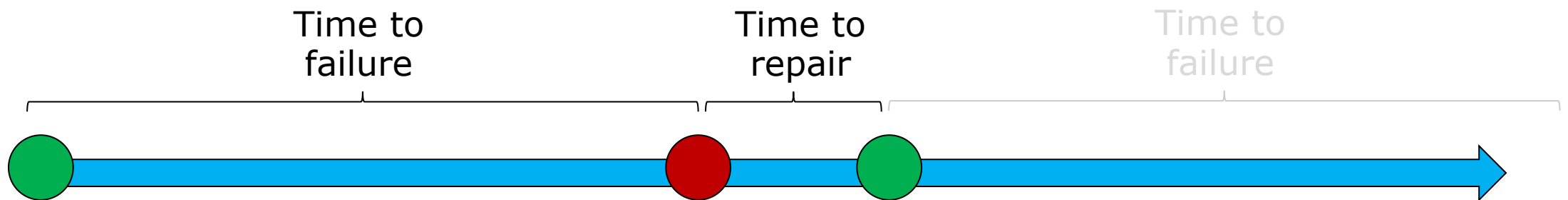
- Mede o **tempo médio desde o instante inicial** até à falha



- Para **sistemas não reparáveis**, é usado **Mean Time To Failure (MTTF)**



Métrica: fiabilidade em sistemas reparáveis



- Em sistemas **reparáveis**, a fiabilidade é normalmente dada pelo tempo médio desde reinício correto até à próxima falha
 - *Mean Time Between Failures (MTBF)*
- É possível também calcular o tempo médio da reparação
 - *Mean Time To Repair (MTTR)*

Exemplo: cálculo do MTBF

“Um serviço deveria operar corretamente durante 9 horas.

Durante este período, ocorreram 4 falhas.

O tempo de todas as falhas totalizou 1 hora.”

- Qual o valor de **MTBF** (tempo médio entre falhas)?

$$MTBF = (9-1) \text{ horas} / 4 \text{ falhas} = 2 \text{ horas}$$

Exemplo: cálculo do MTTR

“Um serviço deveria operar corretamente durante 9 horas.

Durante este período, ocorreram 4 falhas.

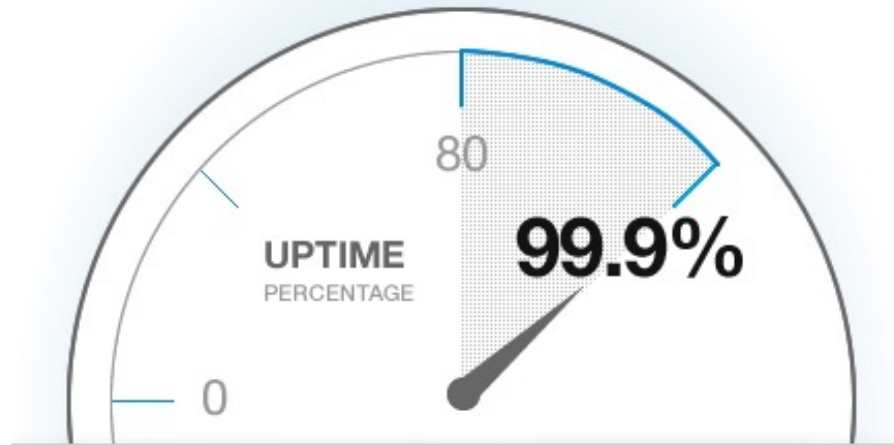
O tempo de todas as falhas totalizou 1 hora.”

- Qual o valor de **MTTR** (tempo médio para reparação)?

$$\begin{aligned} MTTR &= 1 \text{ hora} / 4 \text{ falhas} = \\ &0,25 \text{ hora/falha} = \\ &15 \text{ minutos para reparar falha (em média)} \end{aligned}$$

Métrica: **disponibilidade** (*availability*)

- Mede a **relação** entre o tempo em que um serviço é fornecido e o tempo decorrido



- **Disponibilidade** = $\text{MTBF} / (\text{MTBF} + \text{MTTR})$
 - MTTR = *Mean Time to Repair*
 - MTBF = *Mean Time Between Failures*

Exemplo: cálculo da disponibilidade

*“Um serviço deveria operar corretamente durante **9 horas**.*

*Durante este período, ocorreram **4 falhas**.*

*O tempo de todas as falhas totalizou **1 hora**.”*

- Disponibilidade = $MTBF / (MTBF + MTTR)$

$$Disponibilidade = 2 / (2 + 0,25) = 88\% \text{ uptime}$$



Melhoria da disponibilidade

- MTBF é uma medida básica da **fiabilidade** de um sistema
 - Queremos que o MTBF aumente
 - Por exemplo, através do aumento da qualidade do serviço
 - O sistema é mais fiável se o tempo entre falhas aumentar
- MTTR indica a **eficiência da reparação**
 - Queremos que o MTTR diminua
 - O sistema é mais fiável se o tempo de reparação diminuir
- Queremos disponibilidade a **convergir** para 100%

Classes de disponibilidade

Tipo	Indisponibilidade (min/ano)	Disponibilidade	Classe
Não gerido	52 560	90%	1
Gerido	5 256	99%	2
Bem gerido	526	99.9%	3
Tolerante a faltas	53	99.99%	4
Alta disponibilidade	5	99.999%	5
Muito alta disponibilidade	0.5	99.9999%	6
Ultra disponibilidade	0.05	99.99999%	7

D: Disponibilidade
(também chamado “número de **noves** de disponibilidade”)
Classe de Disponibilidade = $\log_{10} [1 / (1 - D)]$



Tolerância a faltas

Modelo de interação e modelo de faltas



Modelos fundamentais

- Antes de desenhar qualquer solução, é muito boa prática definir os modelos fundamentais
- Três modelos fundamentais:
 - **Modelo de Interação**
 - **Modelo de Faltas**
 - Modelo de Segurança



Modelo de interação

O que pressupomos sobre o **canal de comunicação**?

– **Latência**, que inclui:

- Tempo de espera até ter acesso à rede +
- Tempo de transmissão da mensagem pela rede +
- Tempo de espera em filas de espera na rede +
- Tempo de processamento gasto em processamento local para enviar e receber a mensagem

– **Largura de banda**

- Quantidade de informação que pode ser transmitida pela rede em cada intervalo de tempo





Modelo de interação (cont.)

O que pressupomos sobre o canal de comunicação?

- Canal assegura **ordem de mensagens**?
- Mensagem pode chegar **repetida**?
- *Jitter*
 - Que **variação no tempo** de entrega de uma mensagem é possível?

E sobre os relógios locais?

- Taxa com que cada relógio local se **desvia do tempo absoluto**





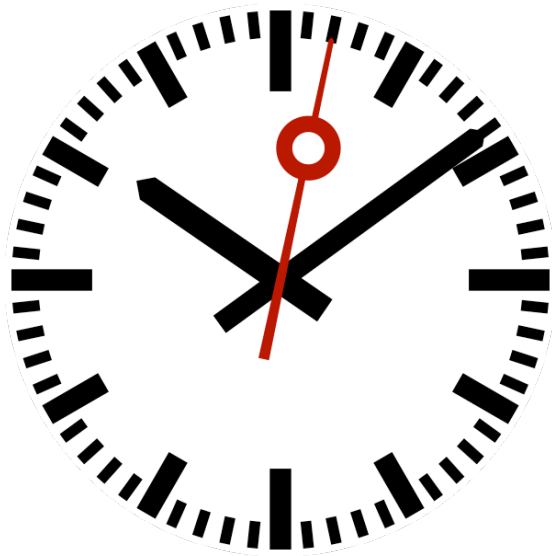
Modelo de interação: sistemas síncronos *versus* assíncronos

- Sistema **síncrono** é aquele em que são garantidos os seguintes **limites temporais**:
 - Cada mensagem enviada chega ao destino dentro de um tempo limite conhecido
 - O tempo para executar cada passo de um processo está entre limites mínimo e máximo conhecidos
 - A taxa com que cada relógio local se desvia do tempo absoluto tem um limite conhecido
- Caso algum destes limites não seja conhecido, o sistema é **assíncrono**



Modelo de interação

Síncrono



Mais fácil construir
algoritmos distribuídos

Assíncrono



Mais realista e genérico, mas
mais difícil a coordenação



Nota importante

- Não confundir estes conceitos com chamada remota síncrona
 - em que o cliente se bloqueia à espera da resposta
- e chamada remota assíncrona
 - em que o cliente não se bloqueia



Deteção de faltas num sistema distribuído

- Deteção num sistema **síncrono**
 - Assume-se a existência de uma **latência máxima** entre nós da rede e um **tempo máximo** de processamento de cada mensagem
 - Quando os limites de tempo são ultrapassados, deteta-se a falta
- Deteção num sistema **assíncrono**
 - Não é possível limitar a latência da rede ou o tempo de resposta do servidor
 - É **impossível a deteção remota** de falhas por paragem
 - Pode ser confundida com um aumento na latência



Modelo de faltas num sistema distribuído

- Num sistema distribuído o modelo de faltas é muito mais complexo que num sistema centralizado pois **vários componentes** do sistema podem falhar:
 - Falhas na **comunicação**
 - Falhas nos **nós**
 - Processadores/sistema
 - Processos servidores ou clientes
 - Meios de armazenamento persistente

Vamos considerar que todas estas faltas podem provocar a falha do nó



Tipos de faltas

- Faltas **silenciosas**
 - Quando o componente **pára e não responde** a nenhum estímulo externo
 - Falta pode ser detetável (**fail-stop**) ou não detetável (**crash**)
- Faltas **arbitrárias** / bizantinas
 - Quando **qualquer** comportamento do componente é possível (e.g., retornar um output *incorreto*)
 - É o **pior caso** possível
 - Útil para representar **erros de software** ou **ataques** no modelo



Faltas silenciosas dos processos

- Quando um processo em falha **deixa de responder** a estímulos do exterior

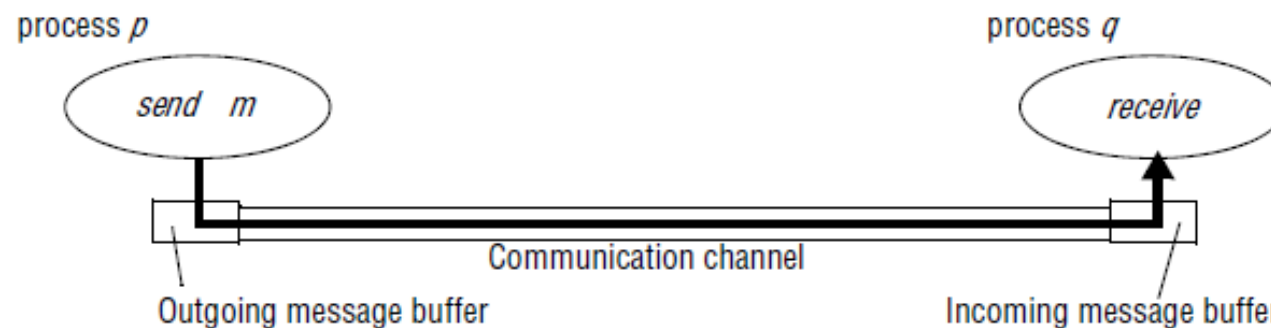


Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (0% complete)

If you'd like to know more, you can search online later for this error: HAL_INITIALIZATION_FAILED

Faltas silenciosas do canal de comunicação

- Quando o canal **não entrega uma mensagem**
- Podemos distinguir entre:
 - **Send-omission**: mensagem perdeu-se entre o processo emissor e o *buffer* de saída para a rede
 - **Channel-omission**: mensagem perdeu-se no caminho entre os *buffers* cliente/emissor
 - **Receive-omission**: mensagem chegou ao *buffer* de entrada do recetor mas perdeu-se depois





Faltas arbitrárias dos processos

- Processo **responde incorretamente** a estímulos
- Processo **responde mesmo quando não há estímulos**
- Processo **não responde** a estímulos



Faltas arbitrárias do canal de comunicação

- Mensagem chega com **conteúdo corrompido**
- Entrega **mensagem inexistente** ou em **duplicado**
- **Não entrega** mensagem
- As faltas arbitrárias do canal de comunicação são **raras**, e por vezes os protocolos conseguem detetá-las
 - Como? O que fazem quando as detetam?



Porquê o nome faltas **bizantinas** para as faltas arbitrárias?



Falta arbitrária / bizantina

- Refere-se ao **Problema dos Generais Bizantinos**
 - Apresentado no artigo: Leslie Lamport, Robert Shostak, Marshall Pease. *The Byzantine Generals Problem*, 1982
 - Cada general comanda uma parte do exército
 - Os generais têm que **concordar** numa estratégia comum para evitar a derrota (falha), mas o **consenso** é difícil de alcançar...
 - Os generais apenas comunicam através de mensageiros, não diretamente entre si
 - Um ou mais generais **podem ser traidores e mentir**
 - Alguns dos mensageiros **podem ser traidores e trocar as mensagens**

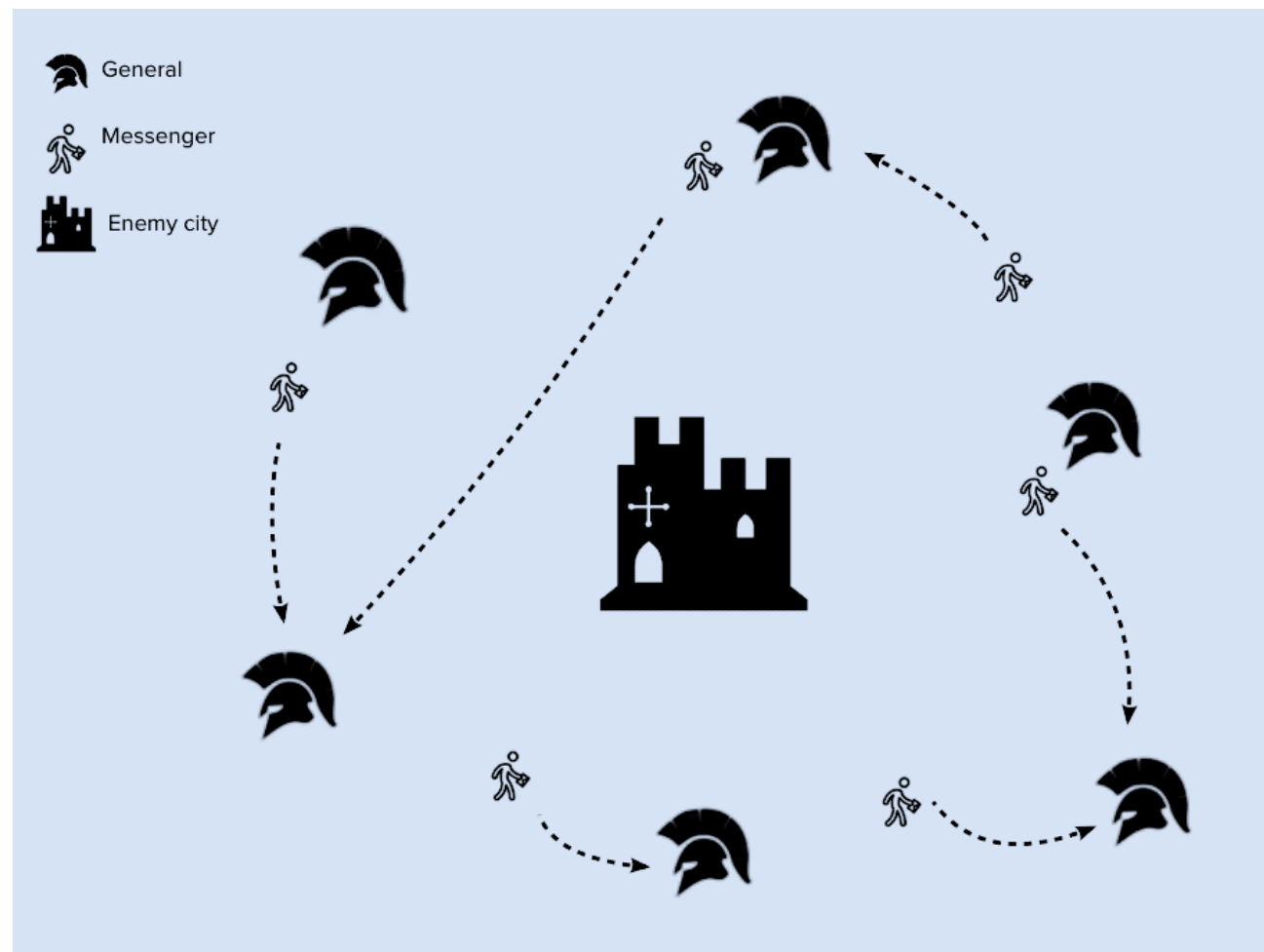


Império Bizantino, Império Romano do Oriente [395-1453]

Atacar ou retirar?

Desafio

Manter **integridade** da
decisão apesar de
existirem traidores





Tolerância a faltas

Políticas de tolerância a faltas



Políticas de tolerância a faltas

- Qualquer política de tolerância a faltas baseia-se na existência de um **mecanismo redundante** que possibilite que a função da componente comprometida seja obtida de outra forma
- A **redundância** pode assumir diversas formas:
 - **Física** ou espacial, com duplicação de componentes
 - Exemplo: cópia de salvaguarda de ficheiro
 - **Temporal**, com repetição da mesma ação
 - Exemplo: retransmissão de mensagem
 - De **informação**, com algoritmos que calculam um estado correto com base no estado atual
 - Exemplo: bit de paridade para recuperar erro num bit



Políticas de tolerância a faltas: recuperação “para trás”

- **Substitui** um estado errado por um estado correto, podendo tornar sem efeito algumas etapas do processamento já efetuado.
- Esta política implica:
 - **Detetar** o erro
 - **Calcular** um estado anterior **correto** (chamado *checkpoint*)
- Durante o tempo de recuperação o sistema fica **indisponível**, afetando a disponibilidade



Políticas de tolerância a faltas: compensação

- Recuperação “para a frente”
- Computa um estado correto a partir de **componentes redundantes**
 - Apesar de um estado interno possivelmente errado!
 - Não é necessário detetar o estado errado
- Desde que haja redundância suficiente, **não há tempo gasto na recuperação** do sistema
 - **Maximiza a disponibilidade** do sistema

Replicação passiva vs ativa

Replicação passiva (*primary-backup*)

Os clientes interatuam com um servidor principal.

Servidor principal altera estado e **envia novo estado** aos outros servidores

Os restantes servidores estão de reserva (*backups*), quando detetam que o servidor primário falhou, um deles torna-se o primário;

Replicação ativa

Todos os servidores recebem, pela mesma ordem, os pedidos dos clientes, **efetuem a operação**, determinam qual o resultado correto por votação, e respondem ao cliente.



Replicação passiva vs ativa

- Replicação passiva (primário-secundário)
 - Suporta operações não determinísticas (o líder decide o resultado)
 - Se o líder produzir um valor errado, este valor é propagado para as réplicas
- Replicação ativa (de máquina de estados)
 - Se uma réplica produzir um valor errado não afecta as outras replicas
 - As operações necessitam de ser determinísticas



Políticas de tolerância a faltas: híbridas

Compensação + recuperação

- Combinam ambas as abordagens

Nas próximas aulas estudaremos soluções concretas em ambas as abordagens...



Bibliografia recomendada

- [Coulouris et al]
 - Secção 18.1 e 18.3
- [van Steen and Tanenbaum]
 - Secções 8.1 e 8.6

