**CD**

**-Resumo Ch8-**

**Considerações Iniciais**

**Dependability:**

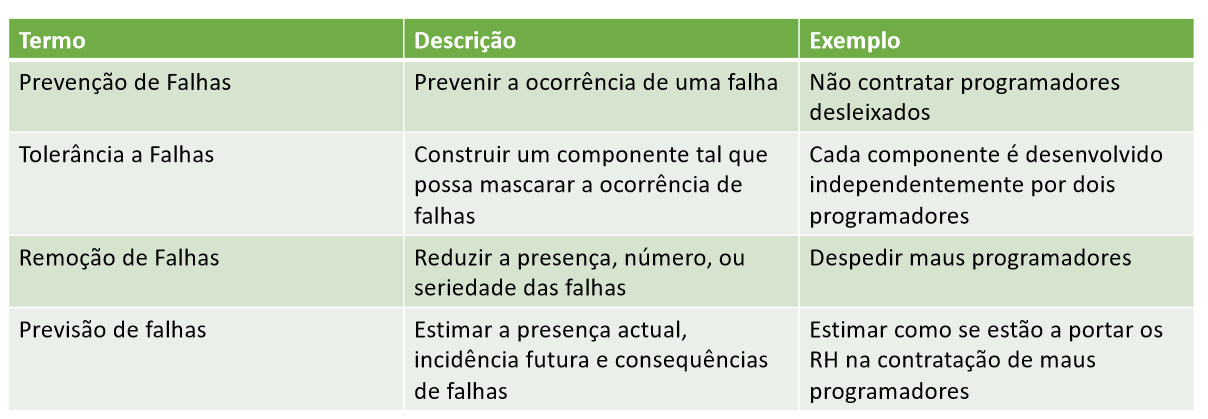
* Um sistema ser fault tolerant está relacionado com a chamada Dependability
* É um termo que cobre vários requesitos como:
  + **Availability** 
    - Sistema está pronto a ser utilizado a dado instante
    - Mede-se por instante de tempo
  + **Reliability**
    - Probabilidade de, dado um instante de tempo, o sistema estar operacional
    - Mede-se por intervalo de tempo
  + **Safety**
    - Situação na qual o sistema falha (não faz o pretendido), nada de catastrófico acontece
  + **Mantainability**
    - O quão fácil o sistema é de reparar
    - Um highly mantainable system por norma tambem tende a ser highly available, dado que os erros consigam ser encontrados rapidamente

**Reliability VS Availability & Metricas Tradicionais:**

* Para ser mais claros, vamos diferenciar reliability e availability
* **Availability** – A(t) – de um componente num time intervalo [0,t] é defenida como a fração média de tempo em que o componente tem estado a funcionar corretamente duranto o intervalo
* **Reliability** – R(t) – de um componente num time intervalo [0,t] é defenido como a probabilidade condicional de que o sistema tem estado a funcionar corretamente durante o intervalo de tempo dado que estava a funcionar corretamente no instante T=0
* Tradicionalmente, fault tolerance tem estado relacionada com 3 métricas:
  + **Mean Time To Failure** – Tempo medio ate que um componente falhe
  + **Mean Time To Repair** – Tempo medio necessario para reparar um componente
  + **Mean Time Between Failures** – MTTF + MTTR
* Note-se que:
  + **A = MTTF/MTBF = MTTF/(MTTF+MTTR)**

**Fault Tolerance:**

* Sistema consegue fornecer os seus serviços, mesmo na presença de eventuais Faults



**Failure VS Error Vs Fault:**

* Failure:
  + Um sistema diz ter falhado caso não consiga comprir o que prometeu fazer
* Error:
  + Parte do estado do sistema que possa levar (ou tenha levado) à ocorrência de um failure
* Fault:
  + Causa do erro
  + Varios tipos:
    - **Transient** – Ocorre uma vez e depois desaparece
    - **Intermitente** – Ocorre, depois não ocorre…depois ocorre…and soo n
    - **Permanent** – Ocorre e não desaparece até que seja tratada

**Failure Models**

Esquemas de categorização de failures dos quais um exemplo é a seguinte categorização:

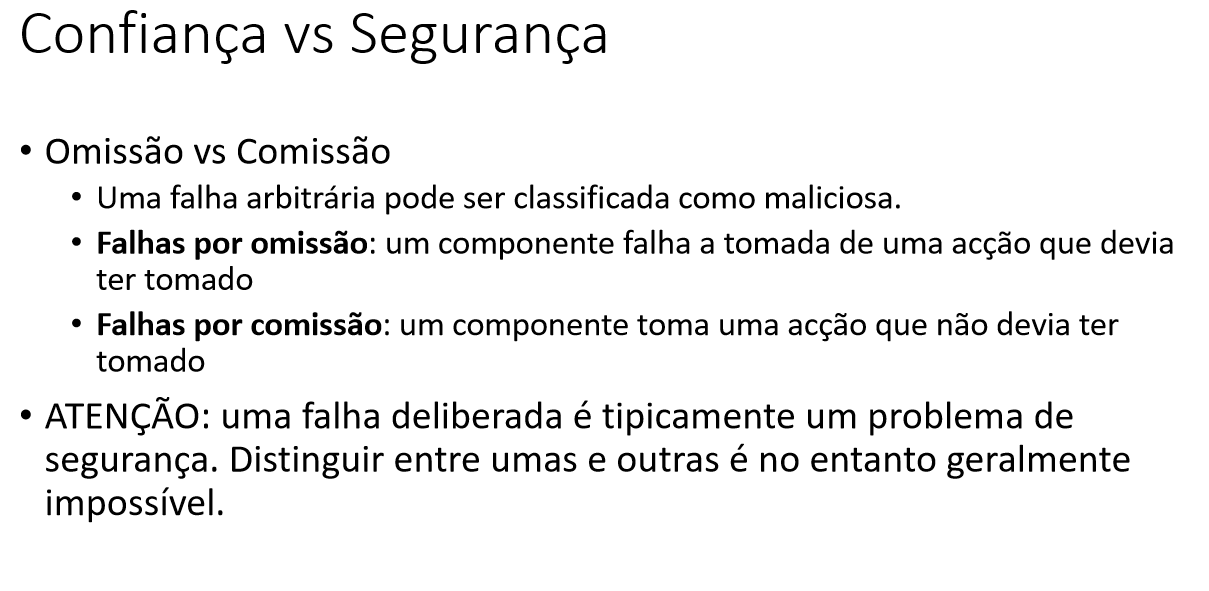
* **Crash Failure**
  + Ocorre quando um servidor permanentemente para, mas estava a funcionar corretamente ate ter parado
  + Nada mais é ouvido do servidor
* **Omission Failure:**
  + Servidor, por alguma razão, falha a responder a um pedido.
  + Pode ser dividido em:
    - **Receive Omission –** Servidor falhou a receber um pedido (servidor não vai ter mudado de estado)
    - **Send Omission** – Servidor falhou a enviar uma resposta (servidor fez trabalho e pode se encontrar num estado que reflita ter realizado trabalho…o que pode dar merda)
* **Timing Failure**
  + Resposta foi realizada fora de um specified time interval (pode ser causado por, p.ex, problemas de performance)
* **Response Failure**
  + Servidor enviou uma resposta incorreta.
  + Existem 2 sub-tipos:
    - **Value Failure –** Servidor simplesmente forneceu uma resposta incorreta ao request
    - **State-Transient Failure –** Servidor reagiu de forma inesperada a um pedido (p.ex, por não saber lidar com ele e executar um comportamento default)
* **Arbitrary Failure**
  + Muito graves
  + Tambem conhecidas por Byzantine Failures
  + P.ex, servidor produziu um output que nunca deveria, mas não consegue ser detetado como incorreto

**Tipos de DS:**

* Um sistema distribuído pode ser:
  + **Asincrono** – Não são feitas assumcoes sobre velocidades de execução de processos ou tempos de entrega de mensagens ; Não consegue perceber se, quando um processo deixa de responder, morreu ou esta a processar qualquer coisa demorada
  + **Sincrono** – Os temos de entrega de mensagens e velocidades de execução são bounded. Efeito contrario dos DS Asincronos
* Num caso real, porem, temos na maioria das vezes DS Partially Synchronous:
  + DS Age deforma síncrona na maioria das vezes, mas excecionalmente age de forma assíncrona, ou seja, normalmente conseguimos detetar halting failures

**Halt Failures Sub Categories:**

* Uma halting failure (como p.ex uma Crash failure) pode ser classificada como:
  + **Fail-Stop Failure**
    - Crash Failures que podem ser reliably detected
  + **Fail-Noisy Failure**
    - Crash failures que só EVENTUALMENTE são detetados
  + **Fail-Silent Failure**
    - Processo não consegue distinguir um crash failure de um omission failure (i.e falta de resposta)
  + **Fail-Safe Failure**
    - Arbitrary Failures benignas (i.e, nao prejudicam)
  + **Fail-Arbitrary Failure**
    - Arbitrary Failures que fodem esta merda toda
    - Unobservable e potentially harmful

**Nota do stor:**

**Failure Masking By Redundancy**

Se um sistema não conseguir ser fault tolerant, o melhor que pode fazer e tentar esconder a ocorrência de failures de outros processos através de redundancia

**Tipos de Redundancia:**

* **Information Redundancy**
  + Adicionar extra bits para permitir recovery de garbled bits
* **Time Redundancy**
  + Uma ação é realizada, e depois, se necessário, realizado outra vez
  + Implica que a ação seja idempotente, se não vai haver merda
* **Physical Redundancy**
  + Usar equipamento extra ou processos adicionais para que um sistema como um todo recupera da perda ou mau funcionamento de alguns dos seus componentes
  + Pode ser feito por soft ou hardware

**Process Resilience**

**-Resiliencia por Process Groups-**

A chave para tolerar um faulty process e com a organização de vários processos similares num grupo

Todos os membros do grupo recebem as mesmas mensagens de pedido

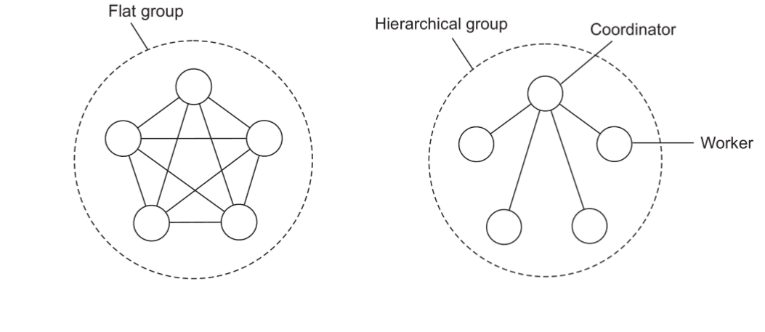
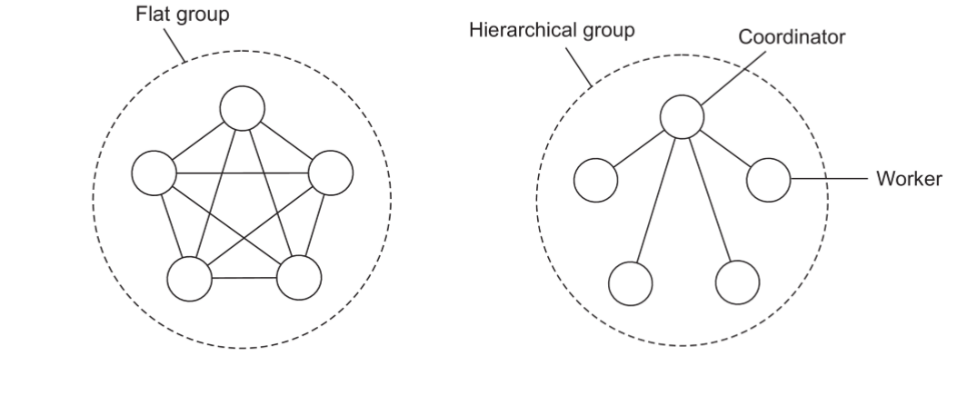
Se um processo falhar, esperemos que haja outro no grupo que tenha realizado o pedido corretamente

Pode ser dinâmicos (podemos criar e destruir novos grupos ; processos podem entrar e sair de grupos)

Processo pode estar em mais do que um grupo

O objetivo é portanto permitir que um processo lide com uma coleção de outros processos como uma única single abstraction (processo manda mensagem para um grupo Q, sem saber que processos pertencem a este grupo, pois para o processo que enviou o pedido, o grupo age como um único logical process)

**Organização dos grupos:**

* **Grupos hierarquicos**
  + Existe algum tipo de hierarquia (p.ex um coordenador que recebe e distribui trabalho)
  + +Coordenador pode fazer decisões de trabalho sem incomodar ninguém
  + -Se coordenador morre, precisamos de eleger um novo (usando algum Election Algorithm)
* **Grupos Flat**
  + Processos todos iguais e sem hierarquia
  + +Grupo simétrico e sem nenhum single point of failure
  + -Decisões são mais difíceis de realizar (precisamos de recorrer a votos)

**Process Resilience**

**-Failure Masking e Replication-**

Ter um grupo de processos idênticos permite-nos mascarar um ou mais faulty processes nesse grupo…thats the whole point

Para replicação normalmente usam-se Primary Based Replications para Hierarchical groups e Replicated Write protocols no caso dos flat groups

Temos que decidir O QUAO TOLERANTE QUEREMOS SER

Um sistema k-fault tolerant conseguem sobreviver a faults em k componentes e continuar fixe

Se os componentes falham silenciosamente, ter k+1 componentes no grupo é suficiente para fornecer k-fault tolerance no caso de k processos simplesmente pararem

Para processos que exibam arbitrary failures e continuem a mandar merda sem parar, precisamos de 2k+1 processes para atingir k-fault tolerance

**Process Resilience**

**-Consensus in faulty systems with crash failures-**

Para que um Process Group funcione temos que assumir que:

“*Num fault tolerant process group cada non faulty process executa os mesmos comandos e pela mesma ordem que todos os outros processos non faulty*”

Ou seja, o grupo tem de chegar a um consensos de que comando executar.

**Flooding Consensus:**

* Vamos assumir que os processos falham segundo fail-stop failure semantics (i.e, failures são reliably detected pelos membros do grupo)
* O Flooding Consensus algorithm opera em rondas