Sistemas Distribuídos

Aula 10 – Tolerância a Falha

DCC/IM/UFRRJ
Marcel William Rocha da Silva

Objetivos da aula

- **Aula anterior**
  - Modelos de falha
  - Resiliência de processo
  - Comunicação confiável cliente-servidor
  - Comunicação confiável em grupo
- **Aula de hoje**
  - Comunicação confiável em grupo (cont.)
    - Multicast virtualmente síncrono
  - Recuperação de falhas

Comunicação confiável de grupo

- **Cenário 2**
  - Processos podem falhar!
  - Deve-se chegar a um acordo sobre a real composição do grupo!
  - Objetivo:
    - Uma msg será entregue a todos os processos ou a nenhum deles
    - Mensagens são entregues na mesma ordem a todos os processos
    - Problema do **multicast atômico**!

Comunicação confiável de grupo

- **Cenário 2: Exemplo:** Banco de Dados replicado
  - Banco de dados é construído como um grupo de processos → um processo para cada réplica
  - Operações de atualização são enviadas em multicast a todas as réplicas (multicast confiável na entrega destas operações!)
  - Suponha que uma das réplicas caia durante a execução de uma das atualizações de uma sequência

Comunicação confiável de grupo

- **Cenário 2: Exemplo:** Banco de Dados replicado
  - **Se o sistema suporta multicast atômico**
  - A operação de atualização que foi enviada a todas as réplicas um pouco antes de uma delas cair ou é executada em todas as réplicas não faltosas ou em nenhuma
  - A atualização é realizada se as réplicas restantes concordarem que a réplica que caiu não pertence mais ao grupo
  - Após a recuperação, a réplica é validada como sendo do grupo e recebe as atualizações

Comunicação confiável de grupo

- **Cenário 2: Exemplo:** Banco de Dados replicado
  - **Multicast atômico** garante que processos não faltosos mantenham uma visão consistente do grupo de réplicas e força a reconciliação quando uma réplica se recupera e se junta ao grupo novamente

Multicast atômico: sincronismo virtual

- Cenário 2: Premissas
  - Camada de comunicação do SD (middleware) gerencia o recebimento das mensagens em um buffer local até que possa ser entregue à aplicação

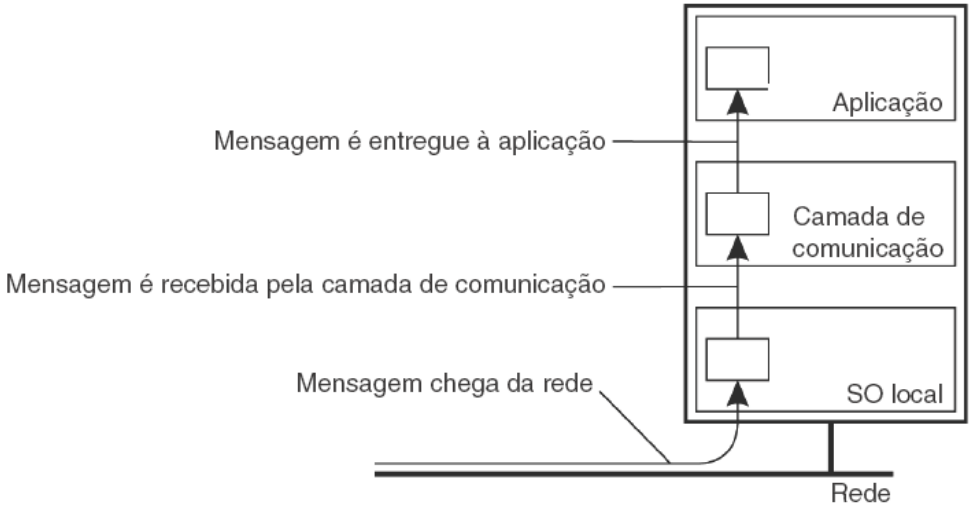


Figura 8.11 Organização lógica de um sistema distribuído para distinguir entre recebimento de mensagem e entrega de mensagem.

Multicast atômico: sincronismo virtual

- Cenário 2: Premissas
  - Uma mensagem **m** está associada com uma lista de processos aos quais deve ser entregue
  - Lista de entrega corresponde a uma **visão do grupo**
  - Todos os processos possuem a mesma visão, concordando que **m** deve ser entregue a cada um deles e a nenhum outro processo

Multicast atômico: sincronismo virtual

- Cenário 2:
  - Suponha que a mensagem **m** seja enviada em multicast no instante que seu remetente tem visão de grupo **G**
  - Se um processo entra ou sai do grupo, ocorre uma **mudança de visão**
  - Duas mensagens: **m** e a de entrada/saída do novo processo (**vc**)
  - Garantir que todos os processos em **G** recebam **m** antes de **vc**, evitando inconsistência

Multicast atômico: sincronismo virtual

- Cenário 2:
  - Multicast confiável garante que uma mensagem enviada em multicast para a **visão de grupo G** seja entregue a cada processo não faltoso em **G**
  - Se o remetente cair durante o multicast, a mensagem pode ser entregue a todos os processos restantes ou pode ser ignorada por cada um deles → **multicast virtualmente síncrono** (Birman e Joseph, 1987)
  - Mensagens multicast ocorrem entre mudanças de visão

Multicast atômico: sincronismo virtual

- Cenário 2:
  - Uma mudança de visão é considerada uma “barreira” pela qual nenhum multicast pode passar!

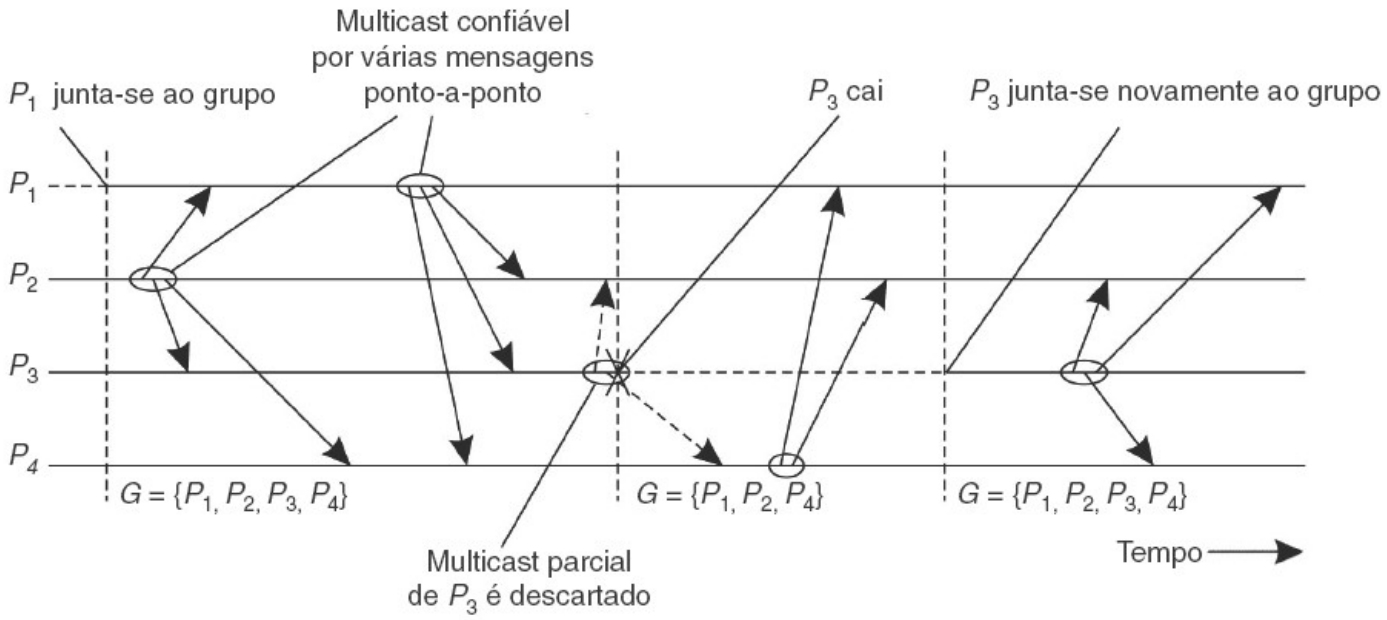


Figura 8.12 Princípio de multicast síncrono virtual.

Multicast atômico: implementação

- Cenário 2:
  - Assumindo apenas comunicação confiável ponto-a-ponto TCP
    - Multicast feito a partir do envio confiável para cada membro do grupo
  - Mensagens armazenadas por cada processo até saber que todos em **G** a receberam
  - Mensagem já entregue a todos é dita **estável**
  - Ao receber mensagem de mudança de visão, processo deve enviar em multicast todas as suas mensagens instáveis e, em seguida, enviar uma **mensagem de limpeza** também em multicast

Multicast atômico: implementação

- Cenário 2:
  - Exemplo

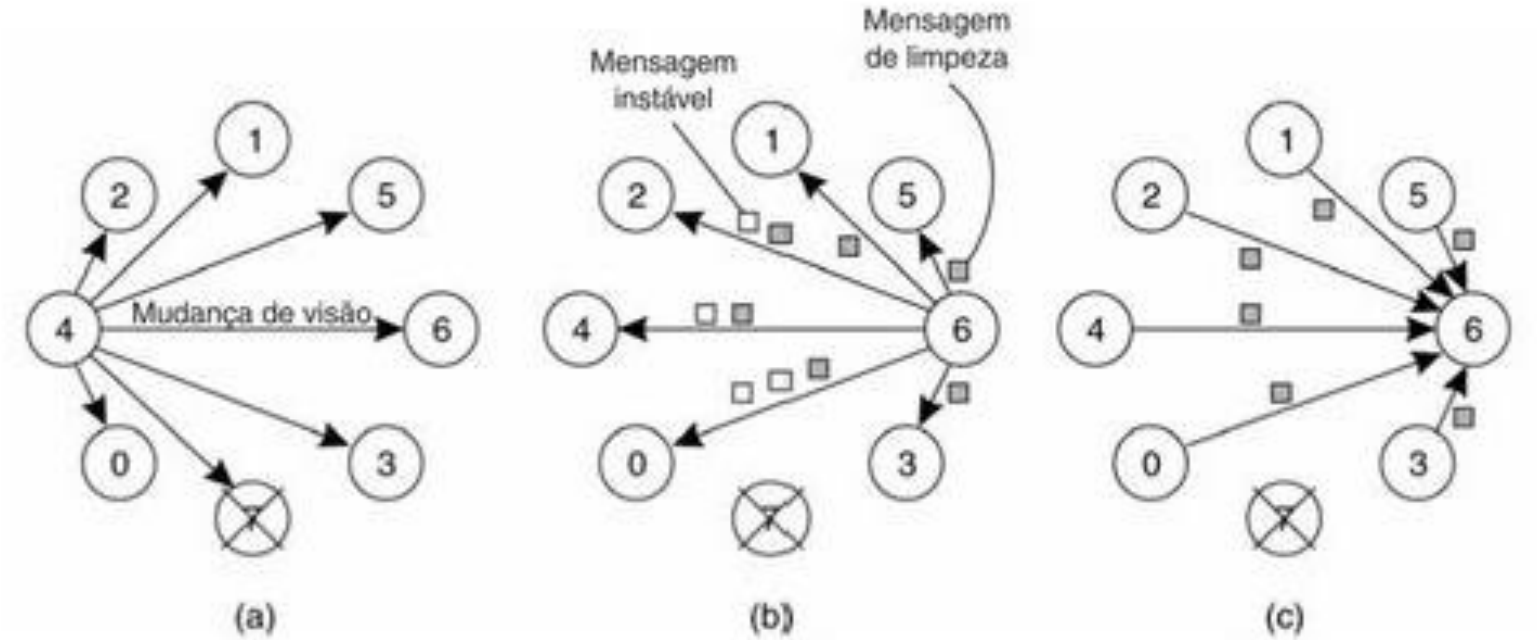


Figura 8.15 (a) O processo 4 percebe que o processo 7 caiu e envia uma mudança de visão. (b) O processo 6 envia todas as suas mensagens instáveis, seguidas por uma mensagem de limpeza. (c) O processo 6 instala a nova visão quando recebe uma mensagem de limpeza de todos os outros.

Recuperação

- Em caso de falha, como levar o sistema para um estado livre de erros?
  - Questão essencial para a tolerância a falhas!

- Duas estratégias:
  - **Recuperação retroativa**: retorna o sistema a algum estado que antes estava correto (**pontos de verificação**), continuando a execução após a recuperação
  - **Recuperação para a frente**: tentativa de levar o sistema para um próximo estado correto

Estratégias de recuperação

- Exemplos:
  - Recuperação retroativa: Comunicação multicast confiável → retransmissão de pacote
  - Recuperação para frente: recuperação de pacotes a partir de outros pacotes (FEC ou *network coding*)

Estratégias de recuperação

- Desvantagens:
  - Recuperação retroativa:
    - Pontos de validação podem ser caros para serem implementados
    - Não existem garantias que o erro não acontecerá novamente
    - Em alguns casos não é possível retroagir a um estado sem erros (Ex: comando ‘rm -rf *’ !!)
  - Recuperação para frente:
    - É preciso saber quais erros podem ocorrer!

Recuperação retroativa + registro de mensagens

- Combinar pontos de verificação com registro da sequência de mensagens recebidas
- Funcionamento
  - Processos receptor registram (armazenam) mensagens antes de entregar para a aplicação (ou o emissor registra antes de enviar)
  - Quando processo cai → sistema é restaurado para o ponto de verificação mais recente e mensagens registradas são “reproduzidas”

Recuperação retroativa + registro de mensagens

- Vantagem:
  - No caso do uso isolado de pontos de verificação, os processos são restaurados para o ponto antes da falha e o comportamento pode ser diferente após a recuperação (ex. msgs podem ser entregues em ordenação diferente)
  - No caso do registro de mensagens, o comportamento é reproduzido do mesmo modo entre o ponto de recuperação e o ponto em que ocorreu a falha → recuperação já garante a ordenação das msgs

Pontos de verificação

- Recuperação retroativa de erros requer que o sistema salve periodicamente seu estado em armazenamento estável
  - **Fotografia distribuída** ou **corte consistente** → registro de um **estado global consistente**

- Em uma fotografia distribuída, se um processo **P** tiver registrado o recebimento de uma mensagem, então também deve existir um processo **Q** que registrou o envio dessa mensagem
  - Em outras palavras...

Cortes consistentes

- Um **corte** será **consistente** se:
  - Para cada evento **x** contido no corte (ou seja, anterior ao corte), também estiverem contidos no corte todos os outros eventos que **aconteceram antes** (*happened before*) de **x**

Cortes consistentes

- Exemplo 1: [livro Tanenbaum]

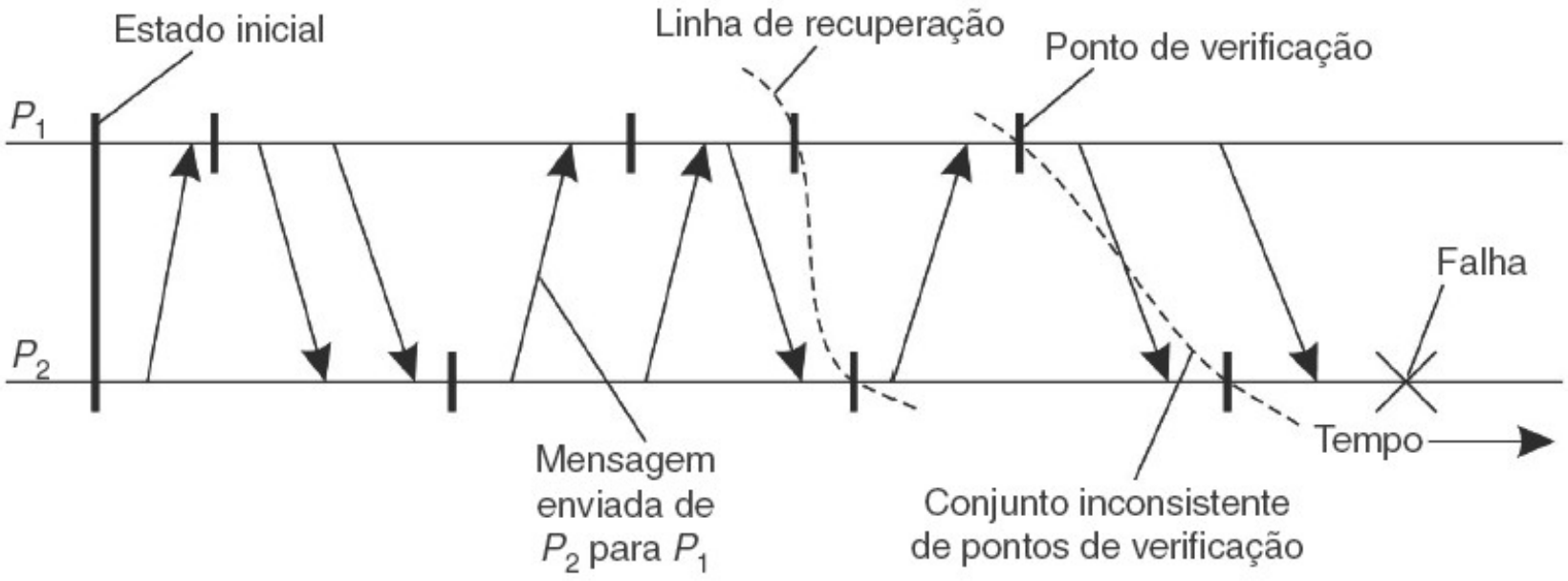
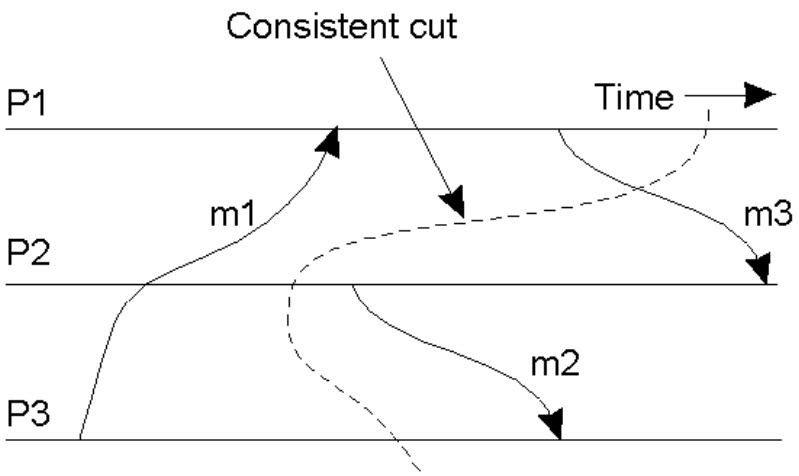


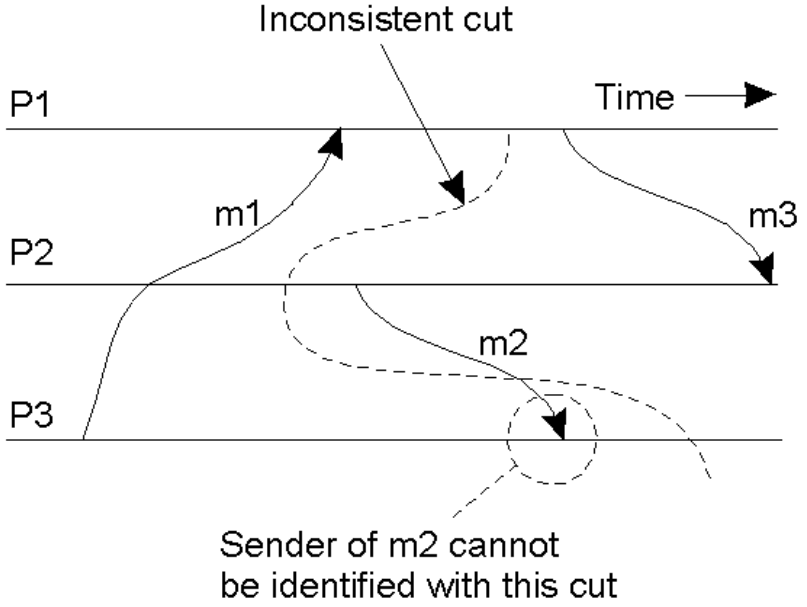
Figura 8.21 Linha de recuperação.

Cortes consistentes

- Exemplo 2: [slides Tanenbaum]



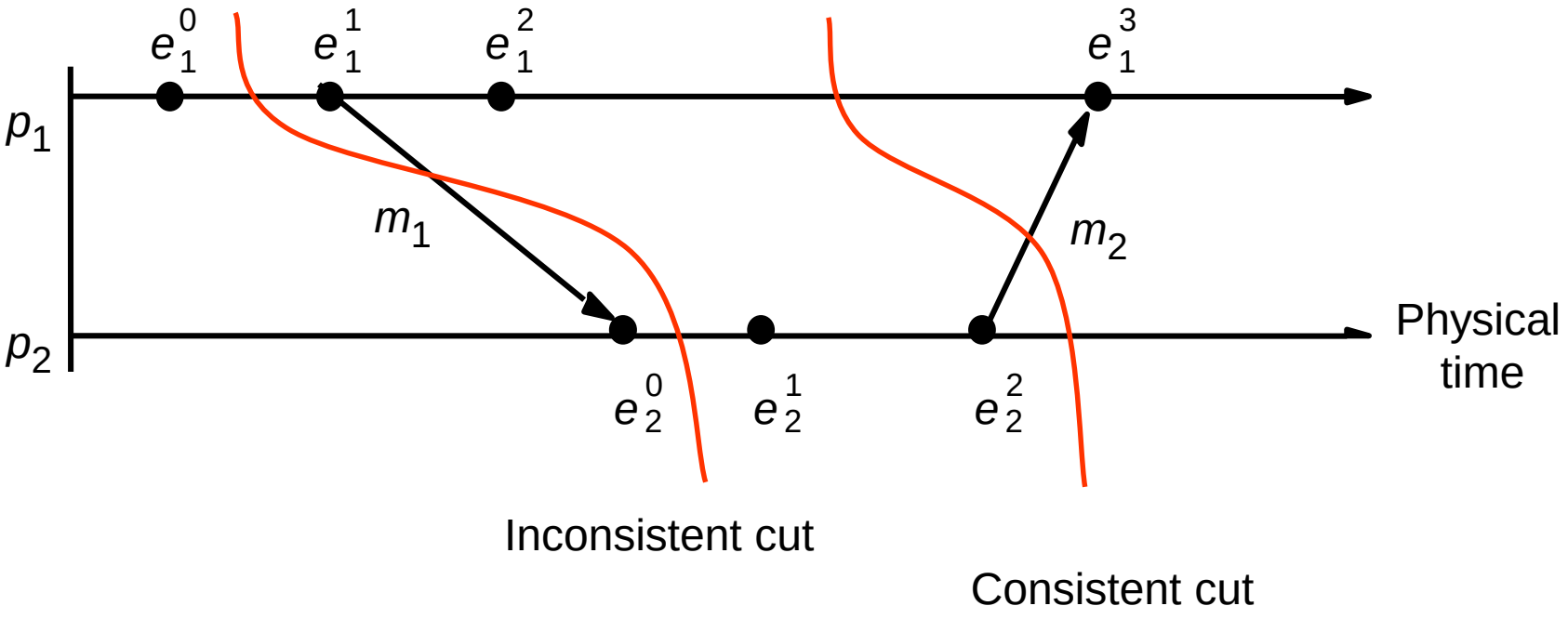
(a)



(b)

Cortes consistentes

- Exemplo 3: [livro Coulouris]



Pontos de verificação independentes

- Cada processo salva pontos de verificação periodicamente em um armazenamento local permanente (Ex: disco)
  - **Estado local**
- A recuperação após uma falha requer a construção de um **estado global consistente** com base nesses **estados locais**
- Melhor alternativa é recuperar o corte consistente mais recente → mais recente conjunto consistente de estados locais

Pontos de verificação independentes

- Descobrir uma linha de recuperação requer que cada processo seja revertido a seu estado salvo mais recente
- Se, em conjunto, os estados locais não formam uma fotografia distribuída, é preciso reverter ainda mais para trás
- O processo de reversão em cascata pode resultar no **efeito dominó**

Pontos de verificação independentes

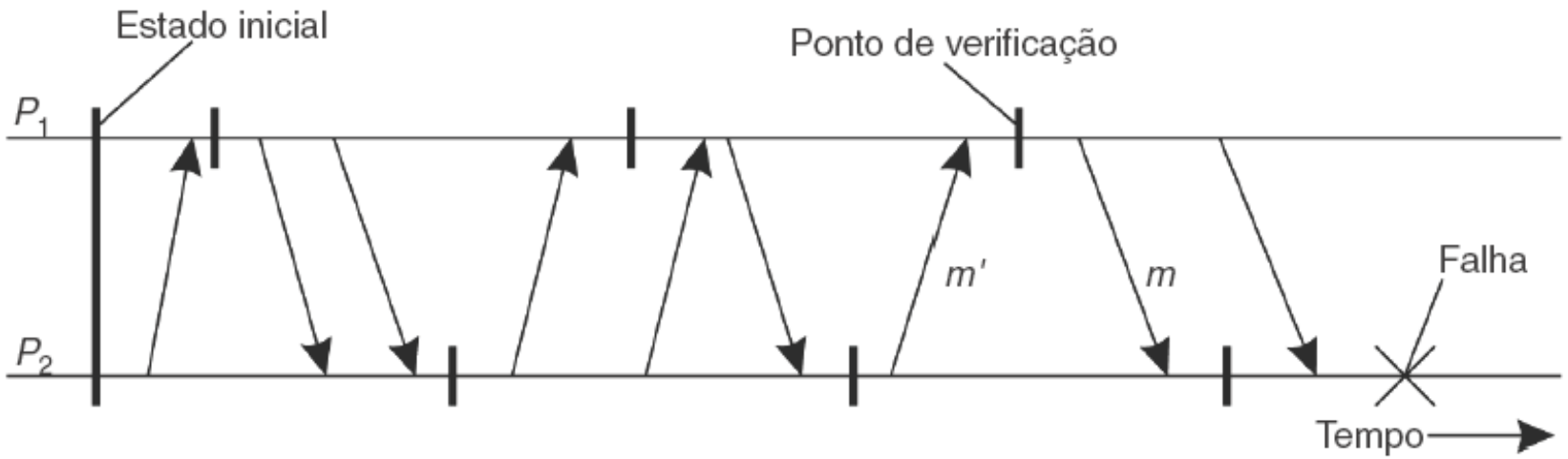


Figura 8.22 Efeito dominó.

Como gerar um estado global consistente?

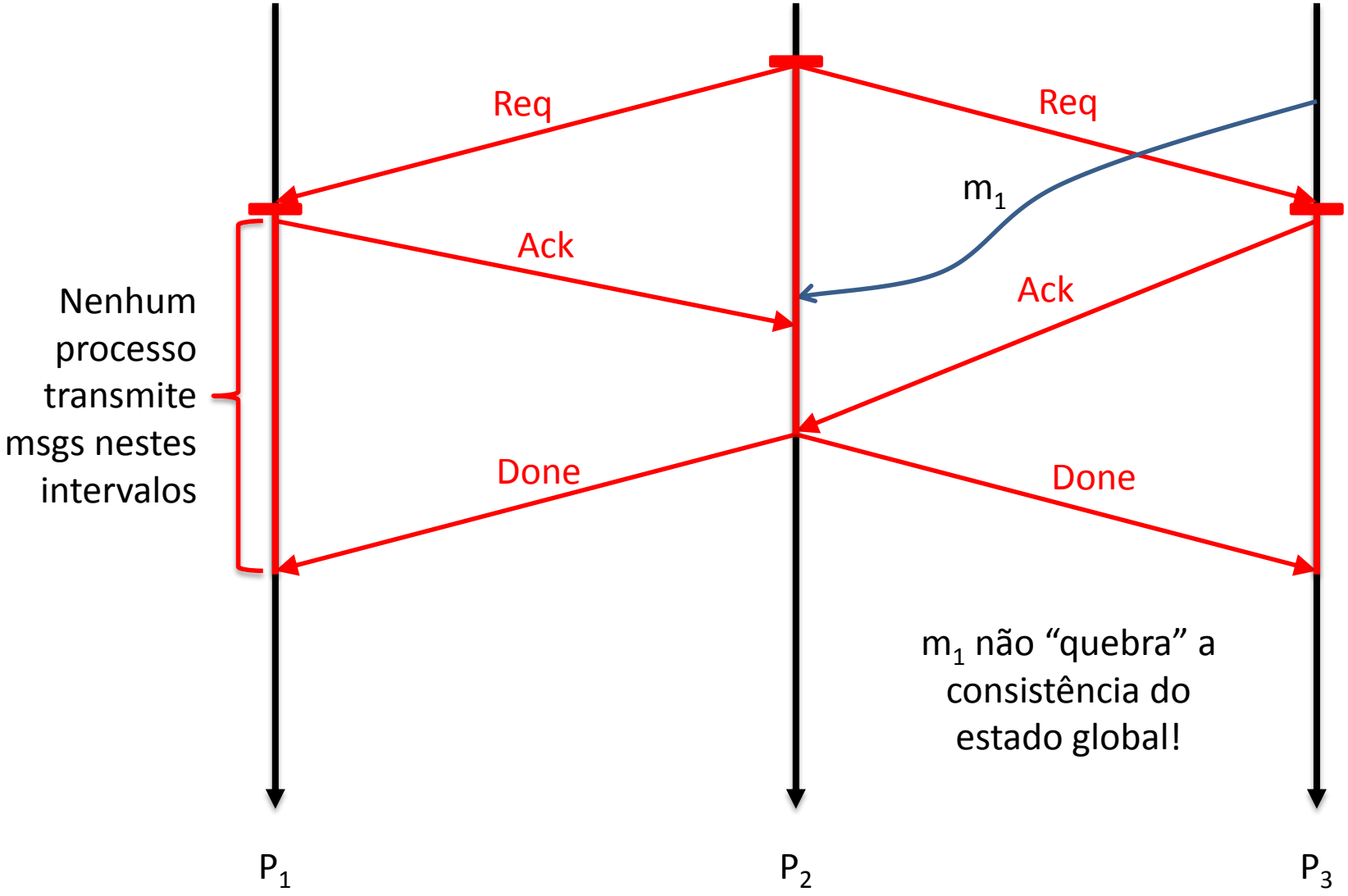
Pontos de verificação coordenados

- Todos os processos sincronizam para escrever, em conjunto, seu estado para armazenamento local
- O estado salvo é globalmente consistente, evitando as reversões em cascata que levam ao efeito dominó

Pontos de verificação coordenados

- Solução: Protocolo de bloqueio de duas fases
  1. Coordenador salva seu **estado local** e envia uma msg CHECKPOINT_REQ em multicast a todos os processos
  2. Quando um processo recebe a msg anterior, salva seu **estado local** e enfileira qualquer nova msg a ser enviada e envia msg de reconhecimento (ACK) ao coordenador indicando que já salvou seu estado local
  3. Após receber todos os “ACKs”, o coordenador envia uma msg CHECKPOINT_DONE em multicast para desbloquear os outros processos
  4. Ao receber o CHECKPOINT_DONE os outros processos estão livres para enviar as msgs enfileiradas

Pontos de verificação coordenados



Pontos de verificação coordenados

- Qualquer mensagem que vier após uma requisição para estabelecer um estado local não é considerada como parte do estado local salvo

- Mensagens que estão saindo são enfileiradas no local até a mensagem CHECKPOINT_DONE ser recebida