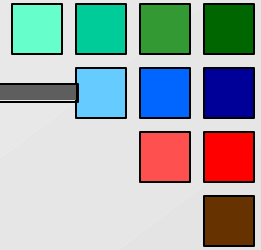
Sistemas Distribuídos

Ricardo Ribeiro dos Santos

ricrs@ec.ucdb.br

Curso de Engenharia de Computação – UCDB – Novembro/2003

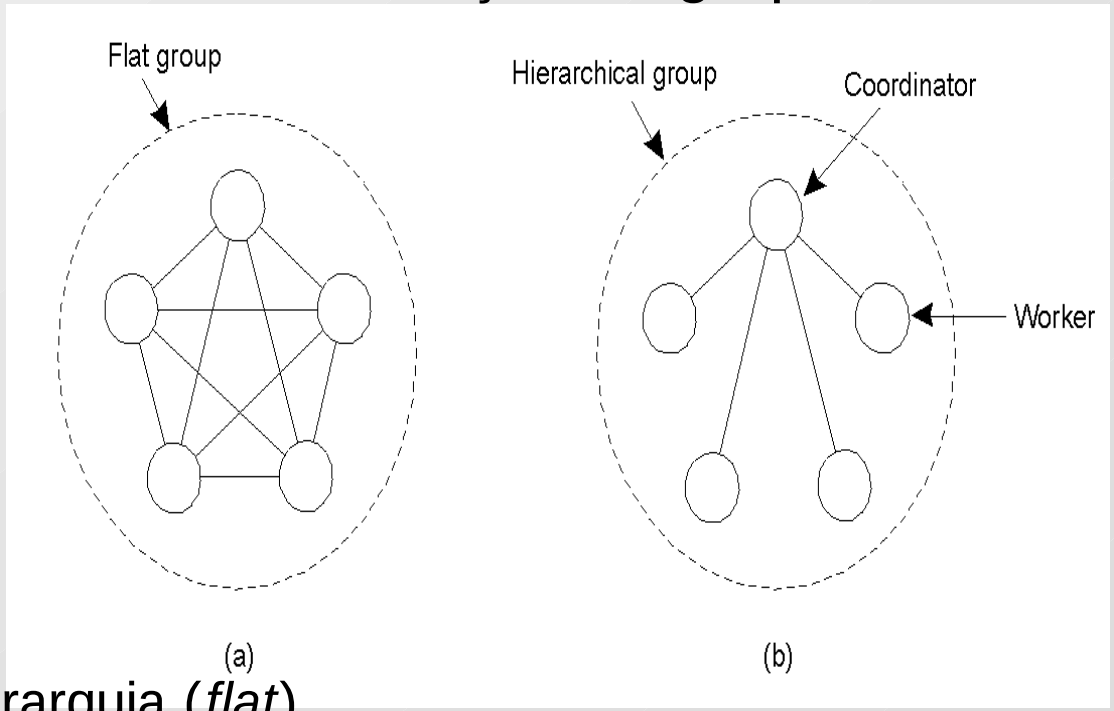
Tópicos



- Tolerância a falhas em comunicação em grupo
- Tolerância a falhas em comunicação ponto-a-ponto
  - Semânticas RPC na presença de falhas
- Recuperação de falhas

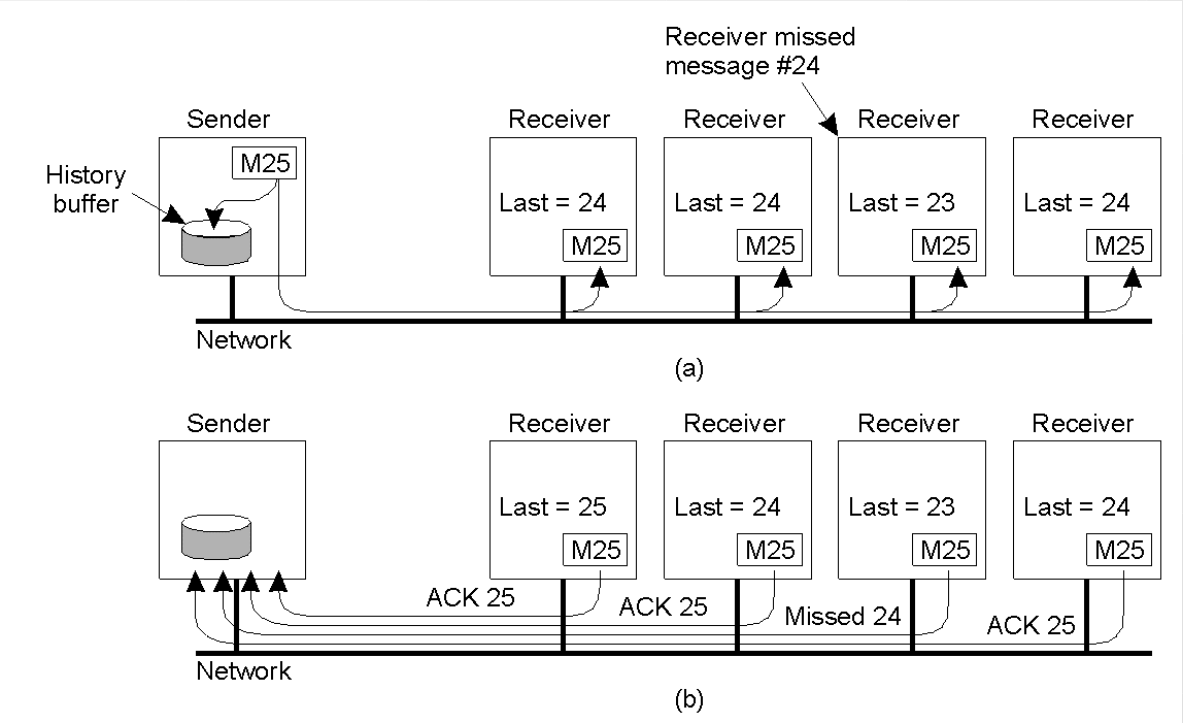
Tolerância a falhas

- O mecanismo básico para contemplar a tolerância a falhas é a replicação
- No contexto de comunicação, a tolerância a falhas é amplamente relacionada com a utilização de grupos de processos (*multicast*)



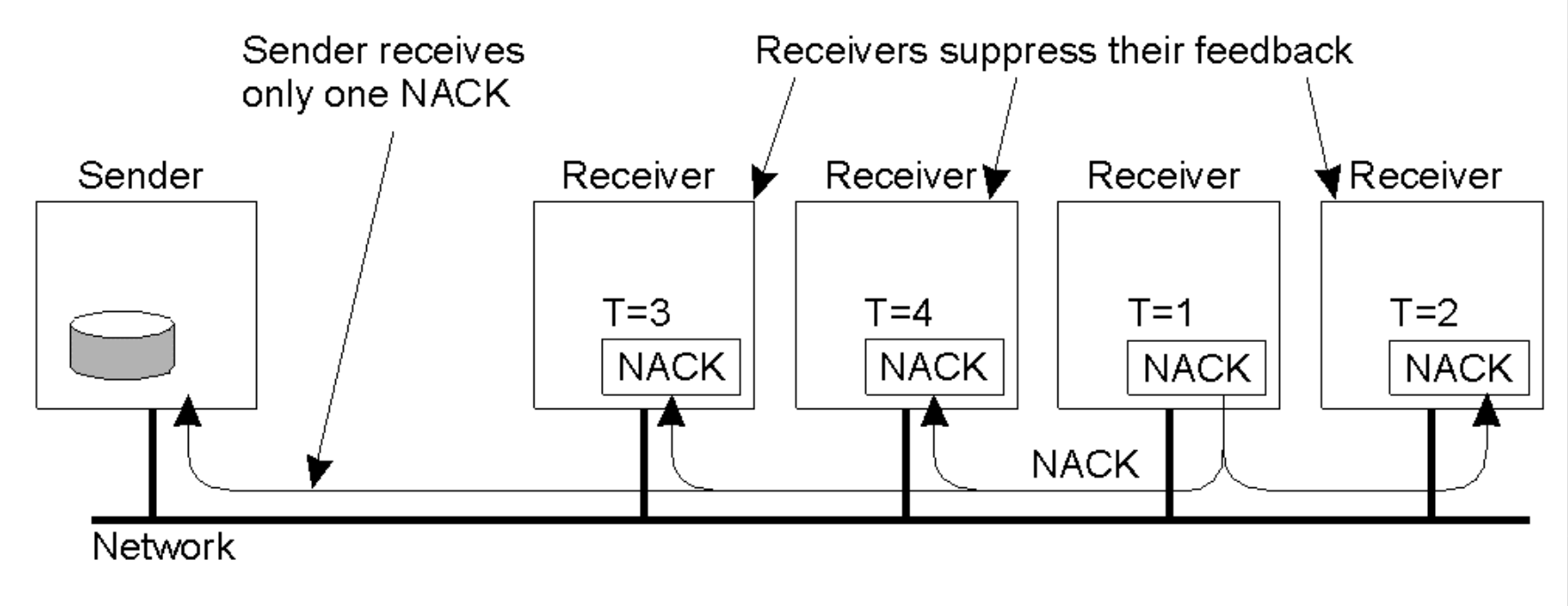
- (a) Um grupo sem hierarquia (*flat*)
- (b) Um grupo hierárquico

Esquemas de *multicasting* confiáveis



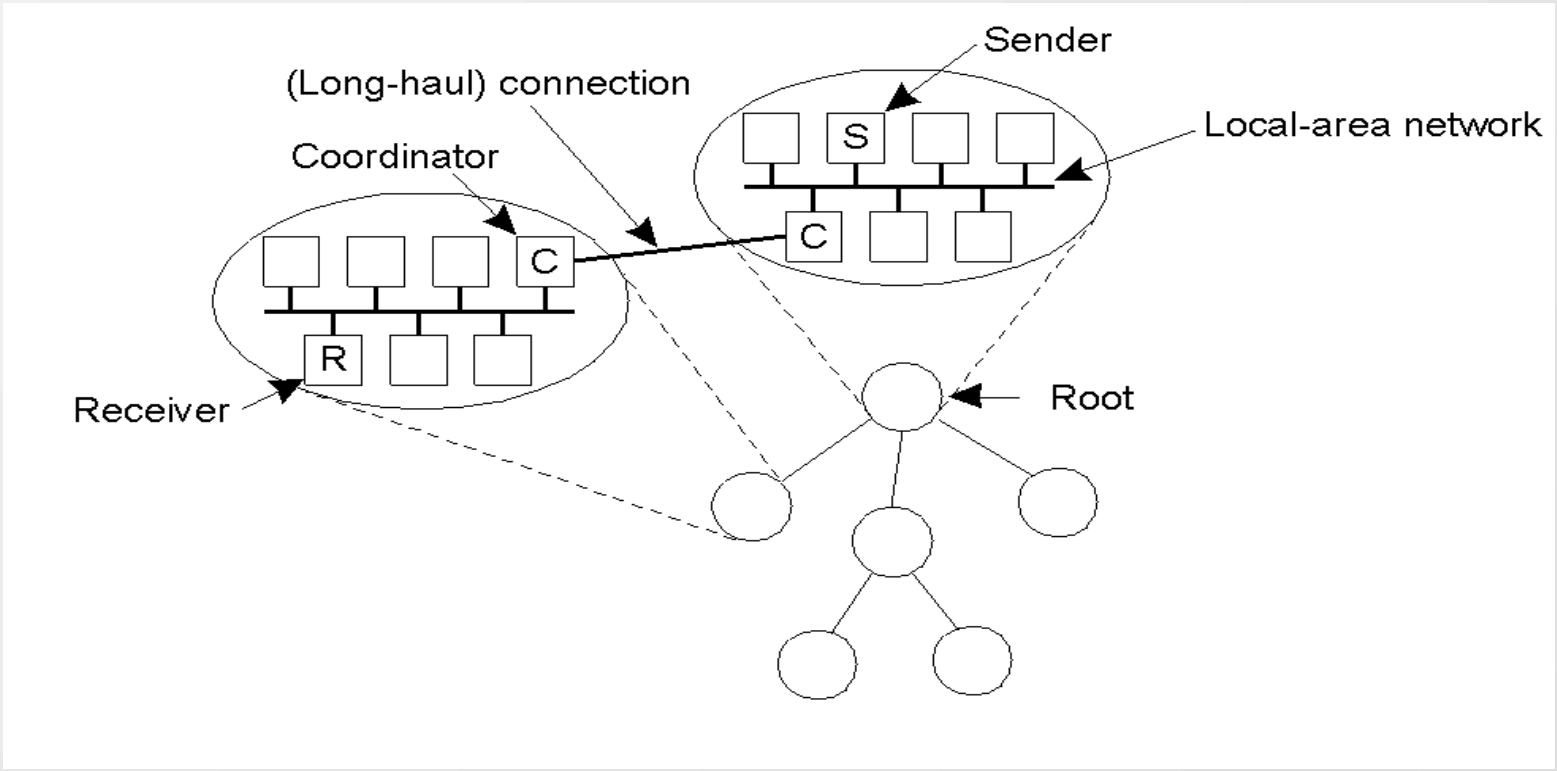
- Uma solução simples para multicasting confiável quando todos os receptores são conhecidos e assume-se que não falharão
  - (a) Transmissão de mensagens
  - (b) *feedback*

Controle de *feedback* não-hierárquico



- Diversos receptores podem querer uma retransmissão, mas o primeiro pedido suprime as outras

Controle de *feedback* hierárquico



- Multicasting hierárquico confiável
- Cada coordenador local envia a mensagem para seus filhos
- Um coordenador local manipula requisições de retransmissão

Comunicação Ponto-a-Ponto

- Comunicação ponto-a-ponto pode acontecer através do uso de protocolos confiáveis como o TCP
- TCP mascara falhas de omissão através do uso de acks e retransmissões
- No entanto, falhas de *crash* não são mascaradas!

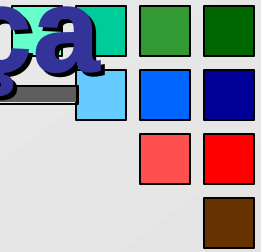
Semânticas RPC na presença de falhas

- Aplicam-se para qualquer sistema de comunicação baseado em RPC:
  - Java RMI
  - CORBA
- É difícil mascarar falhas em comunicação entre processos de maneira que tais falhas pareçam as mesmas falhas de sistemas centralizados

Semânticas RPC na presença de falhas

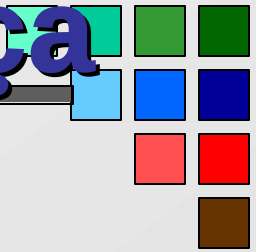
- Há basicamente 05 classes de falhas que podem ocorrer em um sistema RPC:
  - Cliente não consegue localizar o servidor
  - Mensagem de requisição do cliente é perdida
  - O servidor entra em crash depois de receber uma requisição
  - A mensagem de resposta do servidor para o cliente é perdida
  - O cliente entra em crash depois de enviar uma requisição

Semânticas RPC na presença de falhas



- Cliente não pode localizar o servidor
  - Servidor pode estar desligado
  - Versões diferentes de *stubs*
- Uma solução é fazer com que exceções sejam geradas!

Semânticas RPC na presença de falhas

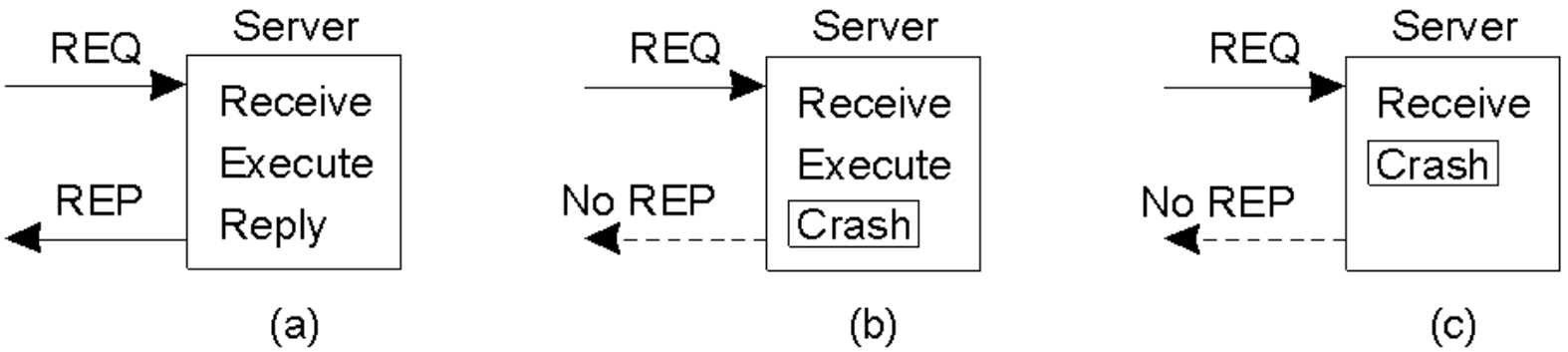


- Perda da mensagem de requisição
  - Cliente pode iniciar um timer
  - Se uma resposta ou ack não chega, a mensagem é enviada novamente
- Se as mensagens são sempre perdidas...
  - Quais as conclusões do cliente?
  - O servidor pode diferenciar uma requisição de uma retransmissão?

Semânticas RPC na presença de falhas

- *Crash* do Servidor
  - O servidor pode falhar em diferentes momentos:
    - Após executar a requisição
    - Antes de executar a requisição
  - Dependendo do “momento” da falha, diferentes tratamentos devem ser dados

Mensagens de requisições perdidas e *crashs* do Servidor



- Um servidor em comunicação cliente-servidor
  - (a) Caso Normal
  - (b) Execução depois de um *crash*
  - (c) Execução antes de um *crash*

Semânticas RPC na presença de falhas

- *Crash* do Servidor
  - Existência de semânticas para tratamento desse tipo de falha:
    - Pelo menos uma vez: Tentar até que, pelo menos uma execução completa seja realizada
      - Nesse modelo, a execução deve ocorrer pelo menos uma vez, possivelmente até mais que uma.
    - No máximo uma vez: A execução deve ocorrer no máximo uma vez mas, pode não ocorrer
    - O ideal seria uma semântica: **exatamente uma vez!**
    - Veja os exemplos a seguir...

Estratégias de invocação do cliente mediante *crashes* do servidor

CLIENTE **SERVER**

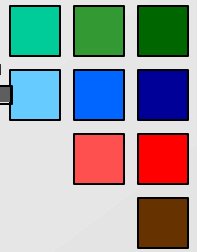
Estratégia de Re-invocação da chamada	Estratégia M -> P			Estratégia P -> M		
	MPC	**MC(P)**	**C(MP)**	**PMC**	**PC(M)**	**C(PM)**
Sempre	DUP	OK	OK	DUP	DUP	OK
Nunca	OK	ZERO	ZERO	OK	OK	ZERO
Apenas quando recebe Mensagem	DUP	OK	ZERO	DUP	OK	ZERO
Apenas quando não recebe Mensagem	OK	ZERO	OK	OK	DUP	OK

- Diferentes combinações de estratégias cliente e servidor na presença de *crashes* do servidor
  - M=Mensagem; P=*Print;* C=*Crash;*
  - (X) = Ação X não será executada pois o servidor entrou em *crash*

Semânticas RPC na presença de falhas

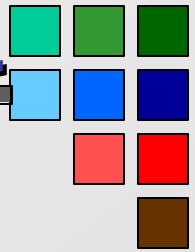
- Perda de mensagens de resposta
  - Qual seria uma solução (simples?) para esse problema?

Semânticas RPC na presença de falhas



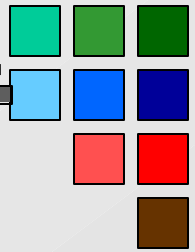
- Perda de mensagens de resposta
  - Uma possível solução seria basear-se em um temporizador
  - Questões que podem surgir:
    - A requisição foi perdida?
    - A resposta foi perdida?
    - O servidor está lento?
- Deve-se atentar para o caso de operações que não são idempotentes!
  - Ex: operações bancárias
- Operações idempotentes são aquelas que podem ser repetidas sem causar inconsistências no sistema

Semânticas RPC na presença de falhas



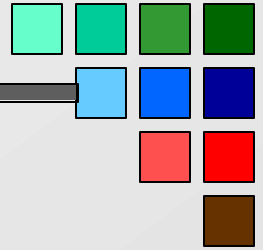
- Cliente entra em *crash*
  - O que acontece se um cliente envia uma requisição e entra em crash?
  - Quais seriam possíveis tratamentos para essa falha?

Semânticas RPC na presença de falhas



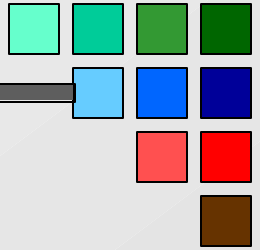
- Cliente entra em crash
  - As respostas do servidor ficam órfãs!
  - Soluções:
    - Antes de enviar a requisição, fazer um log em disco
      - Órfãos são eliminados após um reboot do cliente
    - Dividir o tempo em épocas que devem ser iniciadas após um reboot do cliente
      - Órfãos são eliminados após um reboot do cliente
      - Existem variantes dessa solução!
  - Quais são os problemas de eliminar uma requisição órfã?

Recuperação de falhas



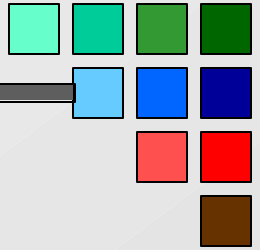
- A idéia geral é, passar de um estado de erro para um estado livre de erros
- Duas técnicas básicas:
  - Recuperação para trás
  - Recuperação para frente

Recuperação de falhas



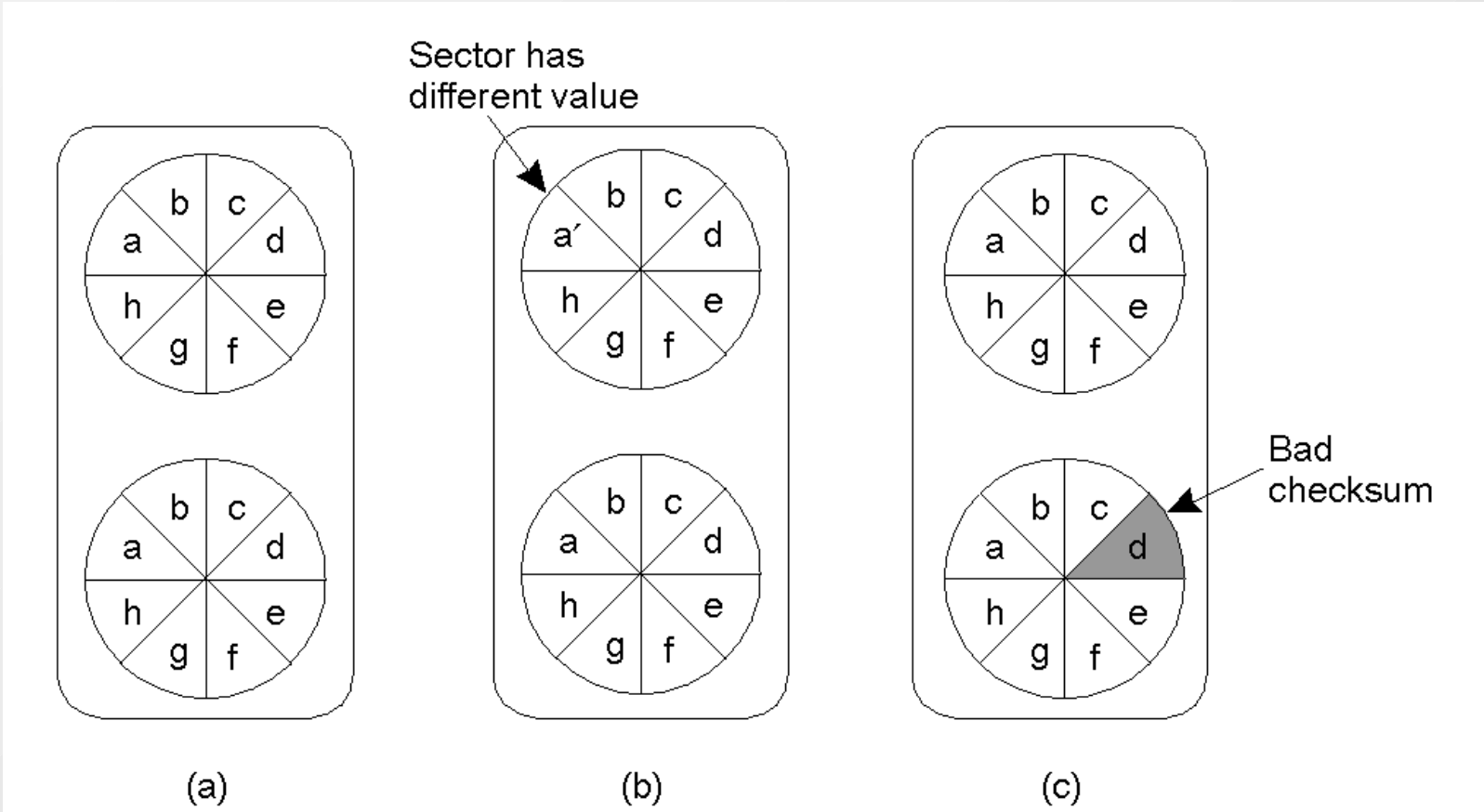
- Exemplo: Recuperação de pacotes perdidos em uma rede
  - Retransmissão de pacotes
    - Recuperação para trás
  - Reconstruir o pacote perdido a partir de outro
    - Recuperação para frente

Recuperação de falhas



- Para conseguir recuperar os dados para um estado seguro é necessário possuir um armazenamento estável
  - Em termos práticos, consiste na adoção de uma matriz de discos que faz a redundância/distribuição dos dados

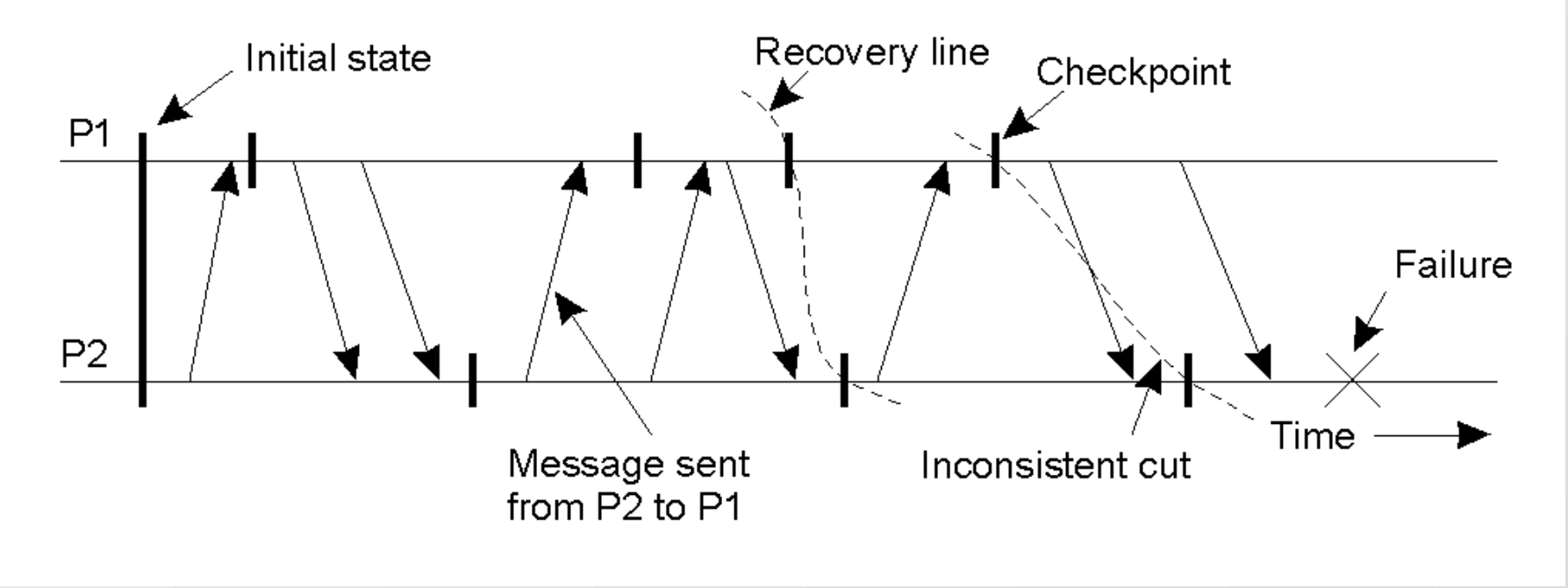
Recuperação de armazenamento estável



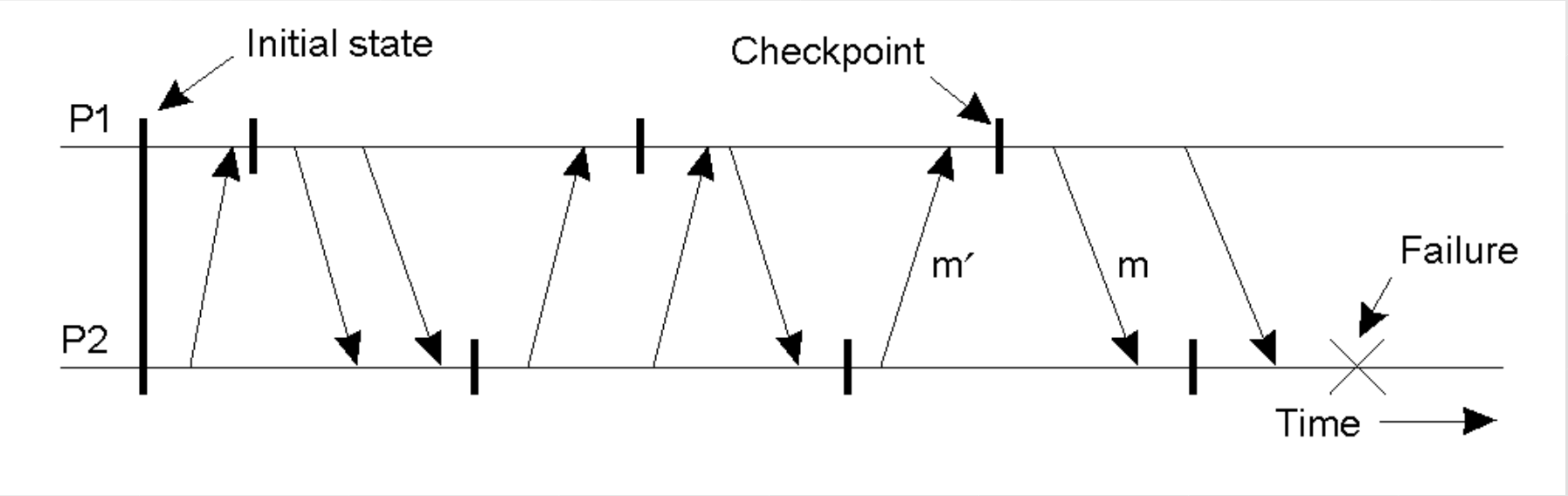
- (a) Armazenamento estável
- (b) *Crash* de atualização no disco 2
- (c) Soma errada no disco 2

Checkpoint

- Salvar o estado dos processos de tempos em tempos

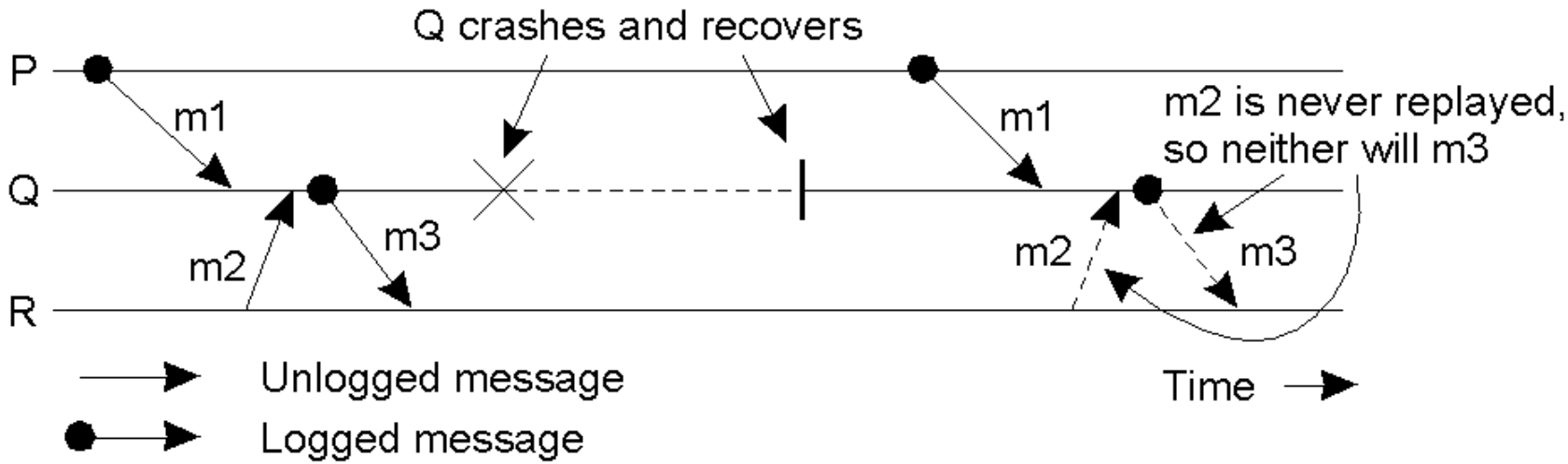


Checkpoint independente



- Pode causar um efeito dominó na recuperação para um estado livre de erros
- Qual a solução?
- Como seria um *checkpoint* coordenado?

Logging de mensagens



- Repetição incorreta de mensagens depois da recuperação, gerando um processo órfão