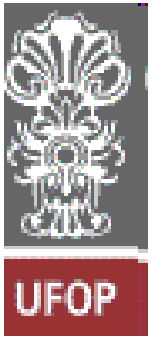


BCC 362 – Sistemas Distribuídos

Joubert de Castro Lima – joubertlima@gmail.com
Professor Adjunto – DECOM

UFOP

Figuras e textos retirados do livro:

Sistemas Distribuídos do Tanenbaum

** Fault Tolerance

An goal in distributed systems design is to construct the system in such a way that it can automatically recover from partial failures without affecting the overall performance.

Whenever a failure occurs, the distributed system should continue to operate in an acceptable way while repairs are being made.

** Basic Concepts

Fault tolerance is strongly related to dependable systems.
Dependability includes:

- *Availability*: Property that a system is ready to be used immediately.

- *Reliability*: Property that a system can run continuously without failure.

- *Safety*: Situation that when a system temporarily fails to operate correctly, nothing catastrophic happens.

- *Maintainability*: How easy a failed system can be repaired.

** Basic Concepts

Fail >> In Portuguese means *Faltar*, *falhar*…*(verbo)*

Fault >> In Portuguese means *Falta*, *defeito*… *(substantivo)*

A system is said to **fail** when it cannot meet its promises.

An **error** is a part of system's state that may conduct to a failure.

The cause of an error is called a **fault**.

** Basic Concepts

Fault classification

Transient faults occur once and then disappear.

Intermittent fault occurs, then vanishes of its own accord, then reappears, and so on.

Permanent fault is one that continues to exist until the faulty component is repaired.

Tipos de Falha

Tipo de falha	Descrição
Falha por queda	O servidor pára de funcionar, mas estava funcionando corretamente até parar.
Falha por omissão	O servidor não consegue responder a requisições que chegam
Omissão de recebimento	O servidor não consegue receber mensagens que chegam
Omissão de envio	O servidor não consegue enviar mensagens
Falha de temporização	A resposta do servidor se encontra fora do intervalo de tempo
Falha de resposta	A resposta do servidor está incorreta
Falha de valor	O valor da resposta está errado
Falha de transição de estado	O servidor se desvia do fluxo de controle correto
Falha arbitrária	Um servidor pode produzir respostas arbitrárias em momentos arbitrários

Tabela 8.1 *Diferentes tipos de falhas.*

Mascaramento de Falha por Redundância

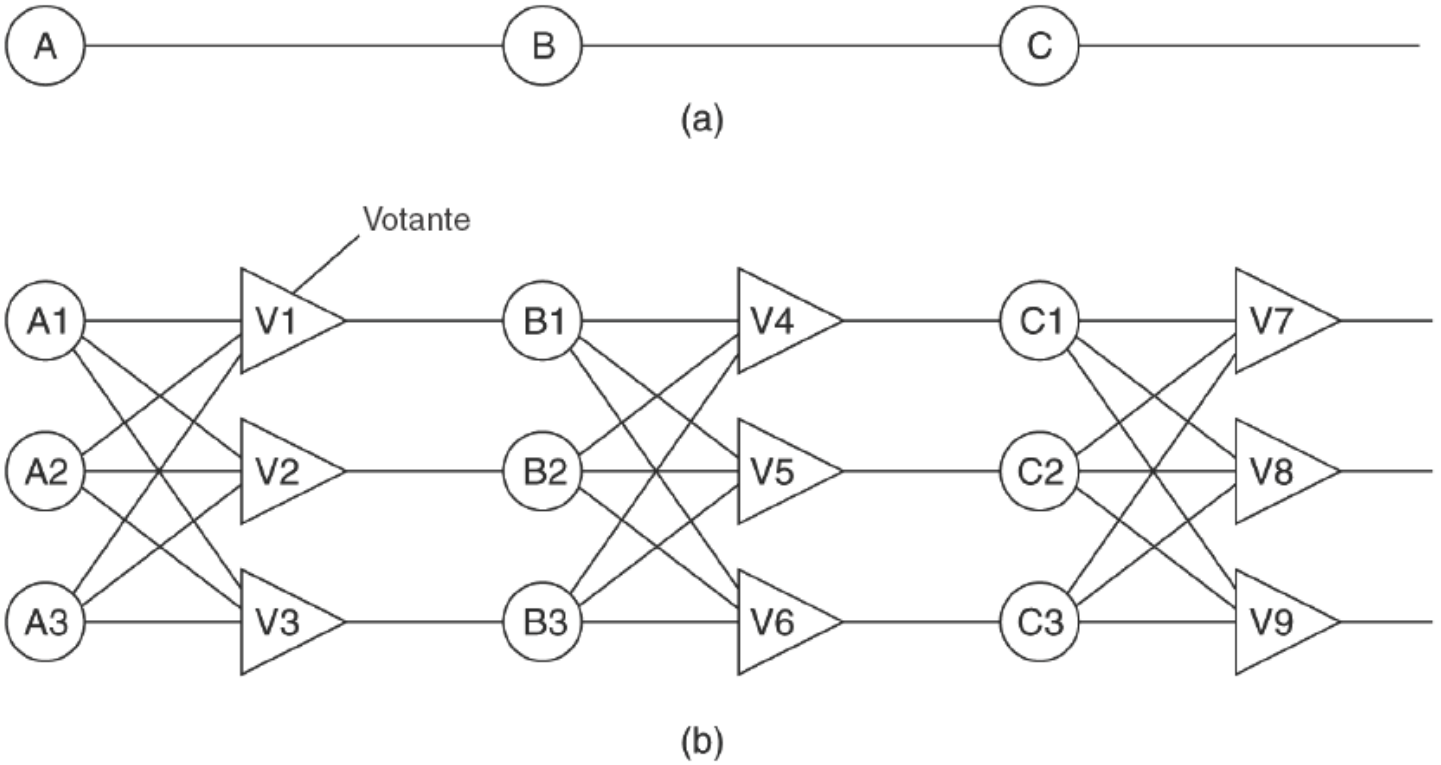


Figura 8.1 Redundância modular tripla.

Resiliência de Processo

How fault tolerance can actually be achieved in distributed systems

Resiliência de Processo

** Design issues

The key approach to tolerating faulty process is to organize several identical processes into a group.

The propose of introducing groups is to allow processes to deal with collections of processes as a single abstraction.

Resiliência de Processo

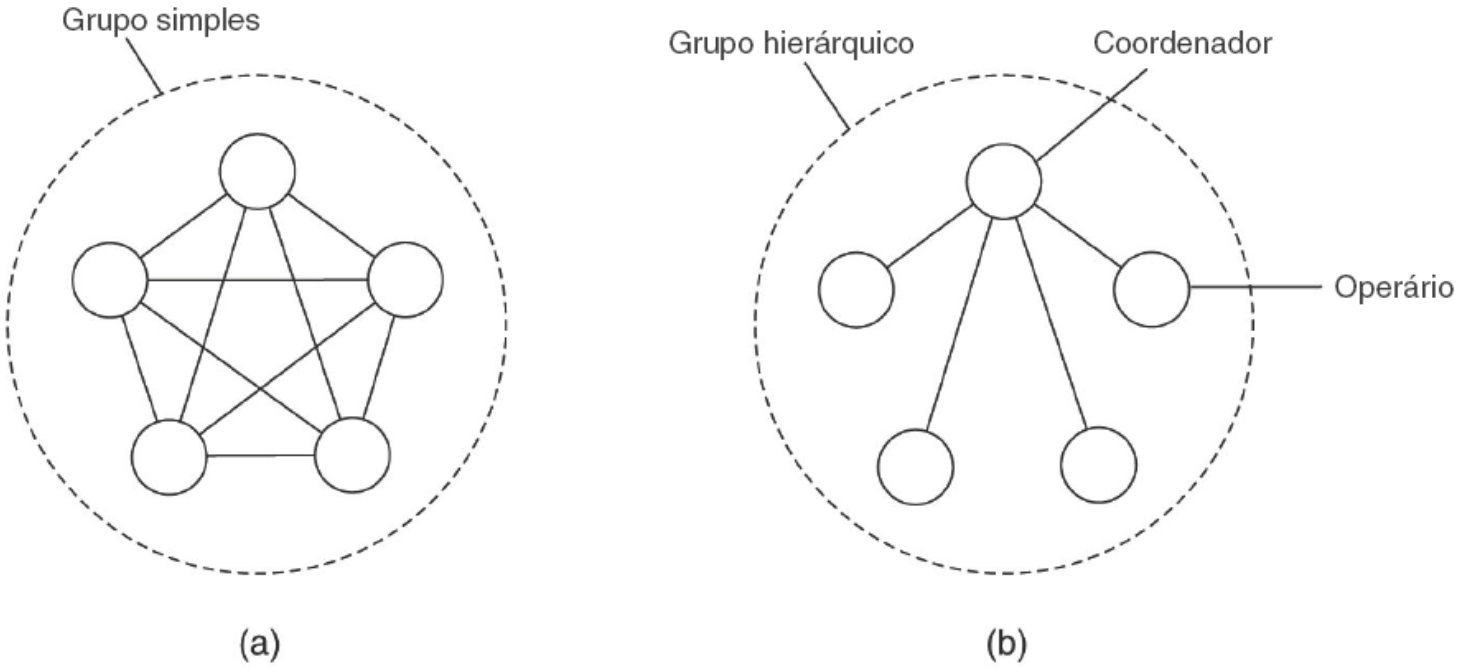
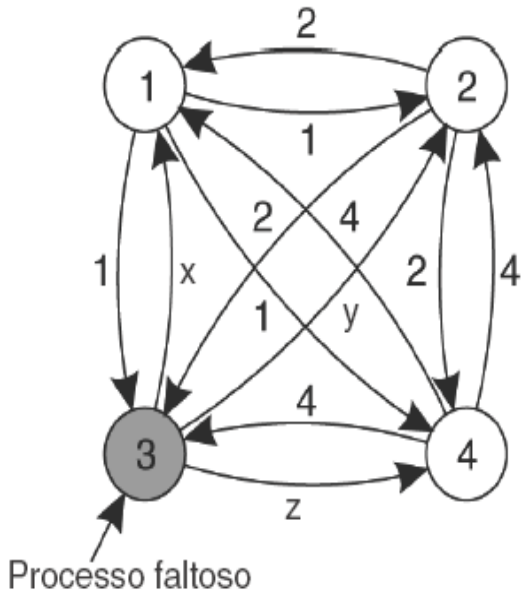


Figura 8.2 (a) Comunicação em um grupo simples. (b) Comunicação em um grupo hierárquico simples.

Problema do acordo Bizantino



(a)

1 Obteve (1, 2, *x*, 4)
2 Obteve (1, 2, *y*, 4)
3 Obteve (1, 2, 3, 4)
4 Obteve (1, 2, *z*, 4)

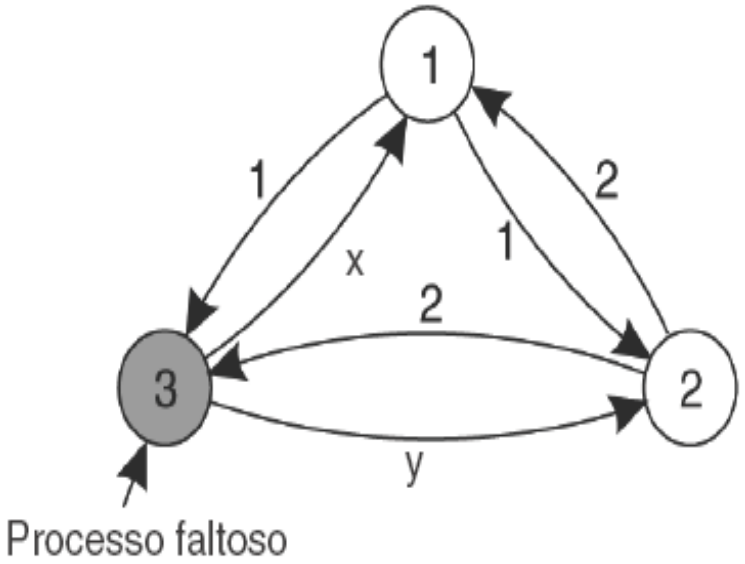
(b)

1 Obteve	2 Obteve	4 Obteve
(1, 2, *y*, 4)	(1, 2, *x*, 4)	(1, 2, *x*, 4)
(*a*, *b*, *c*, *d*)	(*e*, *f*, *g*, *h*)	(1, 2, *y*, 4)
(1, 2, *z*, 4)	(1, 2, *z*, 4)	(*i*, *j*, *k*, *l*)

(c)

Figura 8.4 Problema do acordo bizantino para três processos não faltosos e um faltoso. (a) Cada processo envia seu valor aos outros. (b) Vetores que cada processo monta com base em (a). (c) Vetores que cada processo recebe na etapa 3.

Problema do acordo Bizantino



(a)

1 Obteve (1, 2, *x*)
2 Obteve (1, 2, *y*)
3 Obteve (1, 2, 3)

(b)

1 Obteve	2 Obteve
(1, 2, *y*)	(1, 2, *x*)
(*a*, *b*, *c*)	(*d*, *e*, *f*)

(c)

Figura 8.5 Igual à Figura 8.4, exceto que, agora, há dois processos corretos e um processo faltoso.

Lamport et al.: com k processos faltosos, temos que ter no mínimo 2k+1 processos corretos.

Detecção de Falhas

Ou enviamos mensagens continuamente aos processos do grupo (PING, por exemplo)

Ou aguardamos as mensagens enviadas pelos processos em um tempo T. Se não houver atraso de comunicação limitado isto é possível. Caso contrário não podemos aguardar indefinidamente.

Reliable client-based communication

** RPC semantics in the presence of failures

To structure our discussion, let us distinguish between five different classes of failures in RPC systems.

1. The client is unable to locate the server
2. The request message from the client to server is lost
3. The server crashes after receiving a request
4. The reply message from the server to the client is lost
5. The client crashes after sending a request

** RPC semantics in the presence of failures

The client is unable to locate the server

One possible solution is to have the error raise an exception or signal handlers.

The drawback is that not every language has exception or signals support.

Another drawback is that having to write an exception or signal handler destroys the transparency.

** RPC semantics in the presence of failures

The request message from the client to server is lost

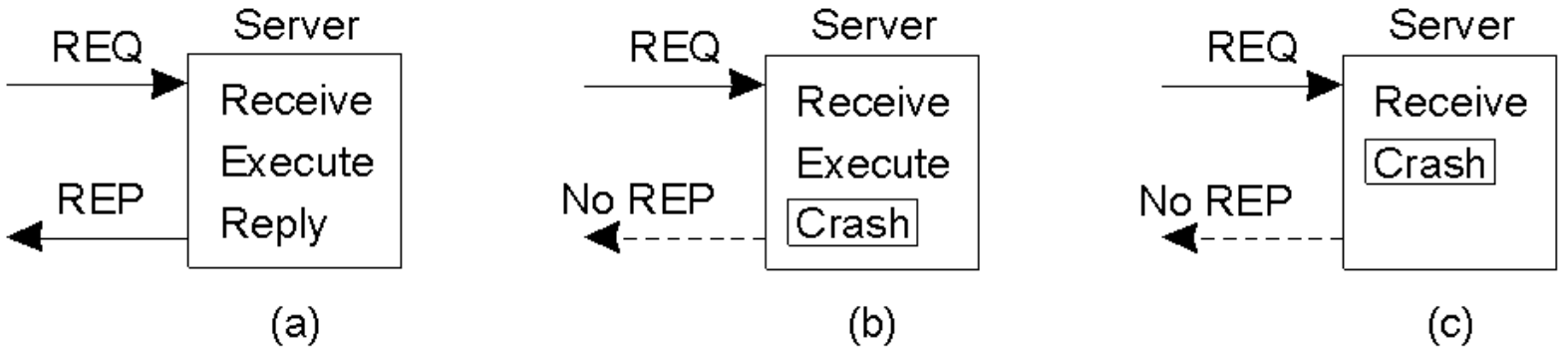
This is the easiest one to deal with: just have the operation system or client stub start a timer when sending the request.

The time expires, the request is resent.

Many message is lost, so the client conclude that the server is down.

RPC semantics in the presence of failures

The server crashes after receiving a request



A server in client-server communication: (i) Normal case; (ii) Crash after execution ; and (iii) Crash before execution.

Option to recovery the crash: (i) To keep trying until a reply has been received; (ii) To give up immediately and reports back failure; (iii) To guarantee nothing.

** RPC semantics in the presence of failures

The reply message from the server to the client is lost

The obviously solution is just to rely on a timer again that has been set by the client's operating system.

If no reply is forthcoming within a reasonable period, just send the request once more.

** RPC semantics in the presence of failures

The client crashes after sending a request: This causes a unwanted computation, called orphan. For example, the client reboots and does the RPC again, but the reply from the orphan comes back immediately. What can be done about orphans?

(i) Before the client stub sends a RPC message, it makes a log entry telling what it is about to do.

(ii) When a client reboots, it broadcast a message to all machine declaring the start a new epoch. So, old computations of that client are killed.

(iii) When an epoch broadcast comes in, each machine checks to see if it has any remote computations, and if so, tries to locate their owner. Only if the owner cannot be found is the computation killed.

(iv) The RPC receives a standard amount of time to do the job. When the client reboot, all orphans are sure to be gone.

Reliable group communication

Um esquema básico

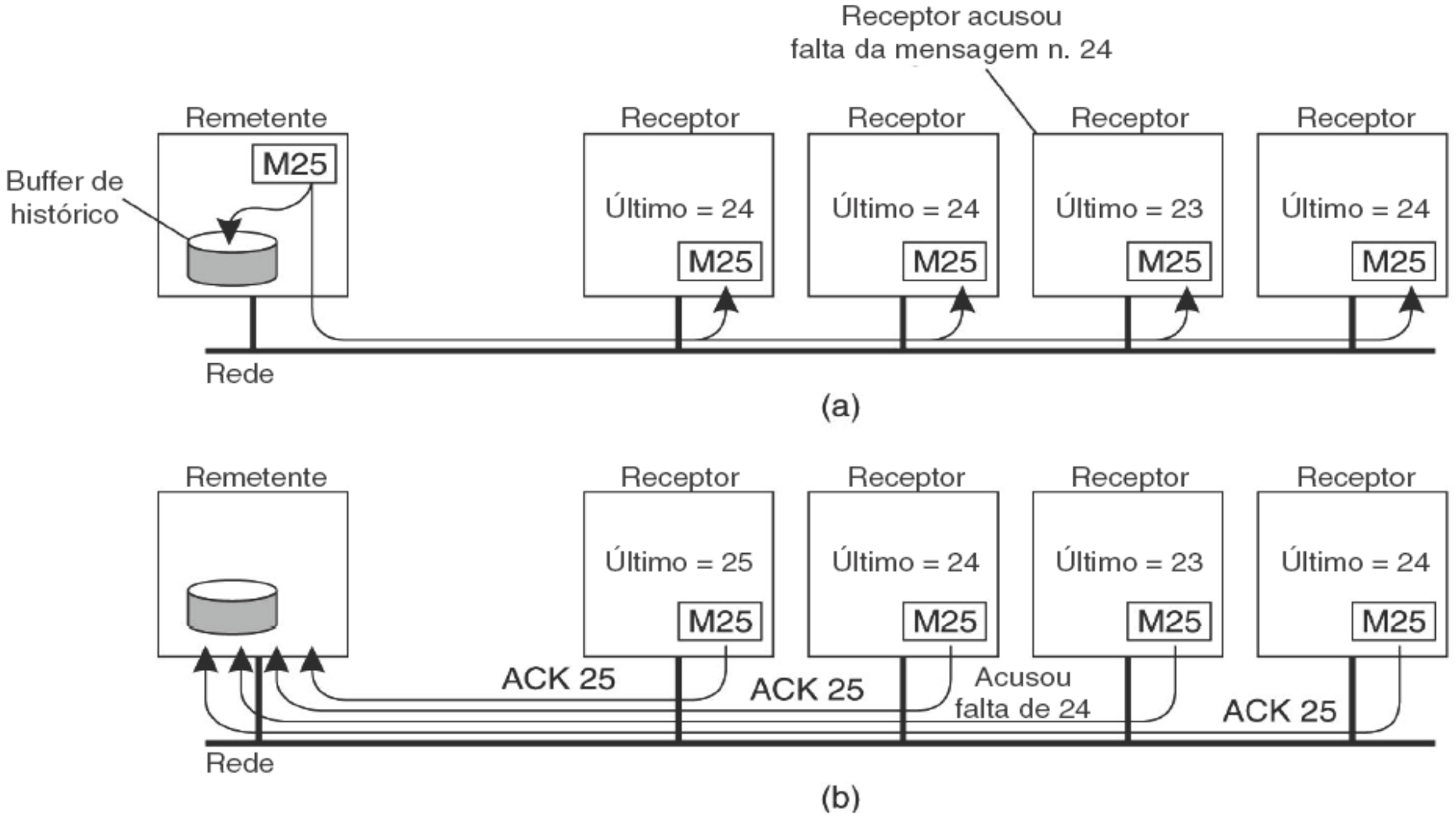


Figura 8.8 *Uma solução simples para multicast confiável quando todos os receptores são conhecidos; a premissa é que nenhum falhe. (a) Transmissão de mensagem. (b) Realimentação de relatório.*

Um esquema escalável

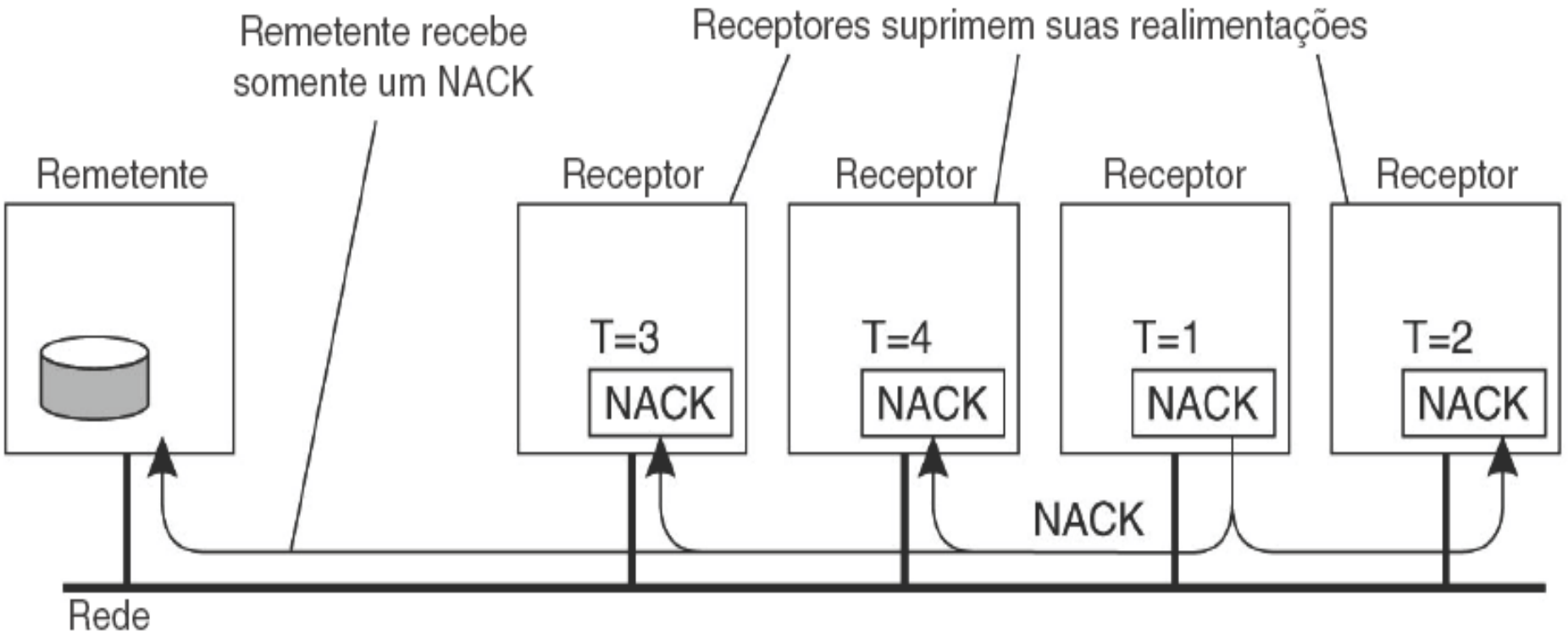


Figura 8.9 *Vários receptores escalonaram uma requisição para retransmissão, mas a primeira requisição de retransmissão resulta na supressão de outras.*

Um esquema escalável e hierárquico

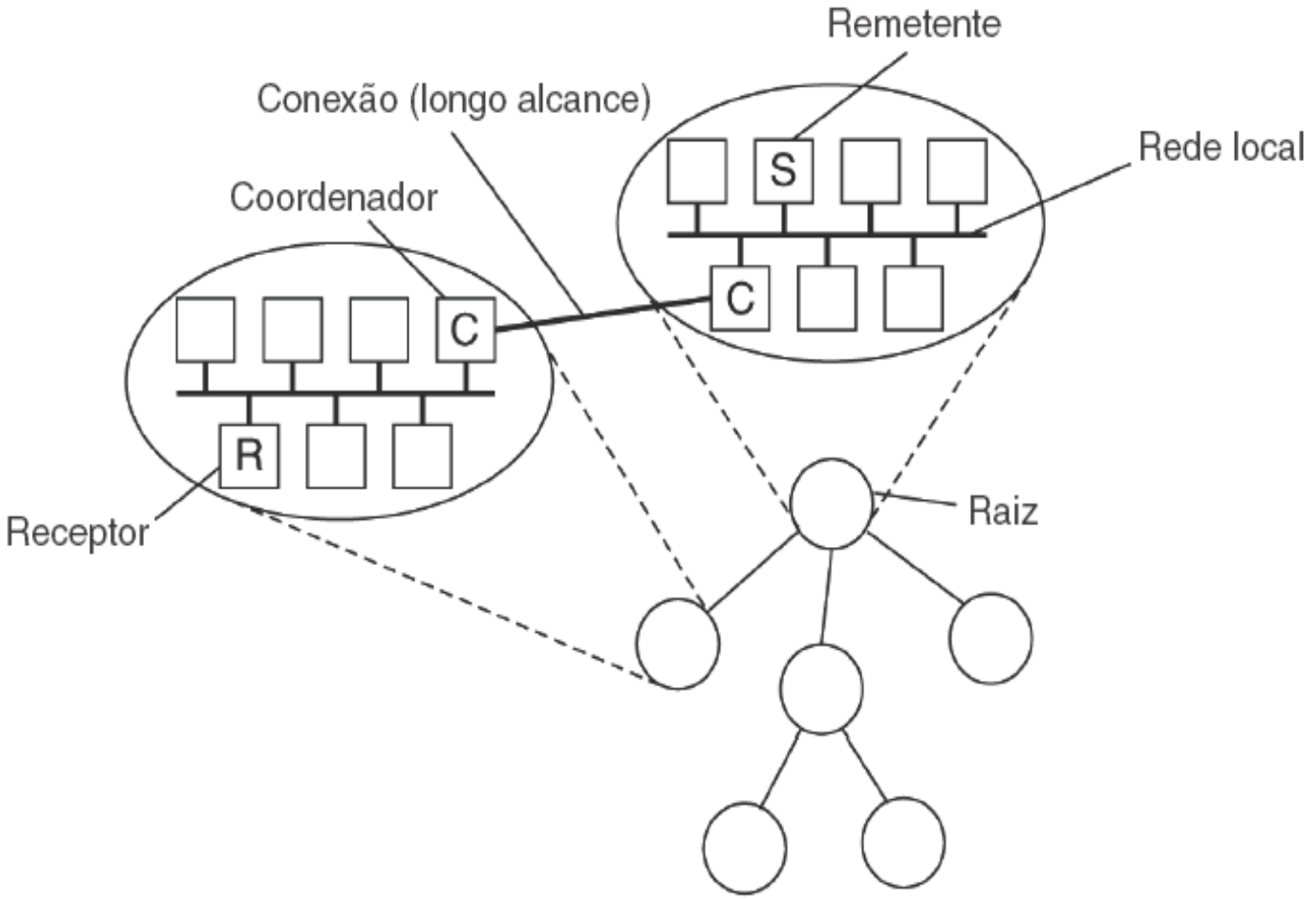


Figura 8.10 Essência do multicast confiável hierárquico. Cada coordenador local repassa a mensagem a seus filhos e mais tarde manipula requisições de retransmissão.

** Atomic multicast

The atomic multicast problem occurs when a message for a group is lost because a process crash.

Example: Consider a replicated database. Update operations are always multicast to all replicas. During the update a replica crashes. That update is lost for that replica but it is performed at the other replicas.

The atomic multicast is performed only if the group have agreed that the crashed replica no back to the group. When the replica recovers, a new contract must be made.

This is performed by virtual synchrony.

** Atomic multicast

Message ordering

Unordered multicast is a virtual synchronous multicast in which no guarantees are given concerning the order in which received messages are delivered by different processes. More simple way.

FIFO-ordered multicast, the communication layer is forced to deliver incoming messages from the same process in the same order as they have been sent.

Totally-ordered multicast is required that when messages are delivered, they are delivered in the same order to all group members.

Processo P_1	Processo P_2	Processo P_3
envia m_1	recebe m_1	recebe m_2
envia m_2	recebe m_2	recebe m_1

Figura 8.13 *Três processos que se comunicam no mesmo grupo. A ordenação de eventos por processo é mostrada ao longo do eixo vertical.*

Processo P_1	Processo P_2	Processo P_3	Processo P_4
envia m_1	recebe m_1	recebe m_3	envia m_3
envia m_2	recebe m_3	recebe m_1	envia m_4
	recebe m_2	recebe m_2	
	recebe m_4	recebe m_4	

Figura 8.14 *Quatro processos no mesmo grupo com dois remetentes diferentes e uma possível ordem de entrega de mensagens em multicast ordenado em Fifo.*

Multicast Confiável

Multicast	Ordenação básica de mensagens	Entrega totalmente ordenada?
Multicast confiável	Nenhuma	Não
Multicast Fifo	Entrega ordenada em Fifo	Não
Multicast por causalidade	Entrega ordenada por causalidade	Não
Multicast atômico	Nenhuma	Sim
Multicast atômico em Fifo	Entrega ordenada em Fifo	Sim
Multicast atômico por causalidade	Entrega ordenada por causalidade	Sim

Tabela 8.2 Seis versões diferentes de multicast confiável virtualmente síncrono.

Distributed commit

** On-phase commit

In *on-phase commit* the coordinator tells all other processes that are also involved, if or not to perform the operation in question.

The drawback is that if one of the participants cannot actually perform the operation, there is no way to tell the coordinator.

** Two-phase commit

Two-phase commit consist of voting and decision phases, described in the following steps:

1. The coordinator sends a VOTE_REQ message to all participants.

2. When a participant receives a VOTE_REQ message, it returns either a VOTE_COMMIT or a VOTE_ABORT.

3. The coordinator collect all votes from the participants. If all participants votes to commit the transaction the coordinator sends a GLOBAL_COMMIT, otherwise it sends a GLOBAL_ABORT message.

4. Each participant wait for the coordinator decision. If it receives a GLOBAL_COMMIT message, it commits the transaction. Otherwise, if it receives a GLOBAL_ABORT message, the transaction is aborted.

Three-Phase Commit

A problem with a two-phase commit is that when the coordinator has crashed. Three-phase commit avoids blocking processes in the presence of fail-stop crashes.

Recovery

**Introduction

In *backward* recovery, the main issue is to allow a system in an erroneous state back into a previously correct state. To do this some snapshots of the system must be performed.

In *forward* recovery, when the system has just entered an erroneous state, the system passes for a correct new state from which it can continue to execute.

Armazenamento Estável

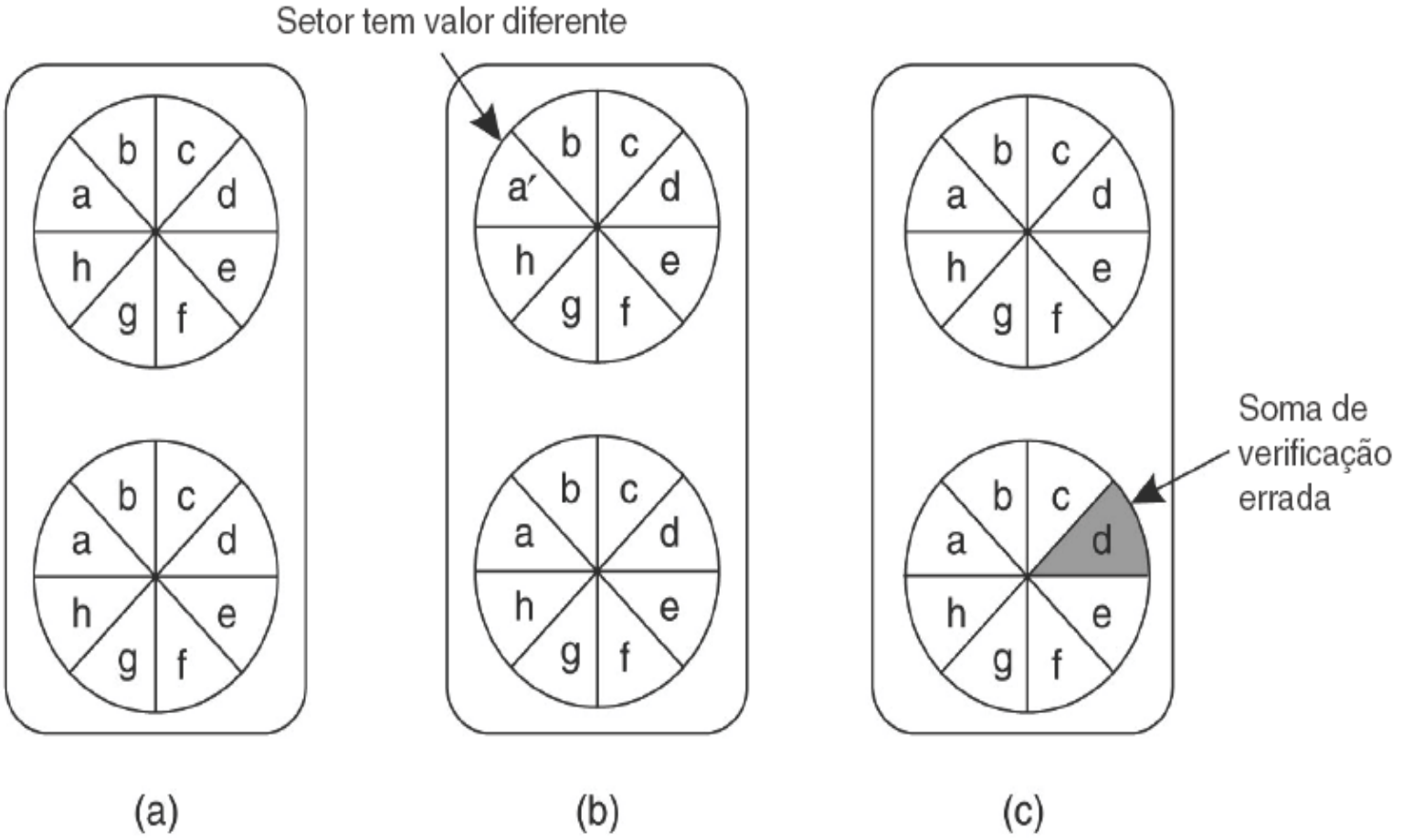
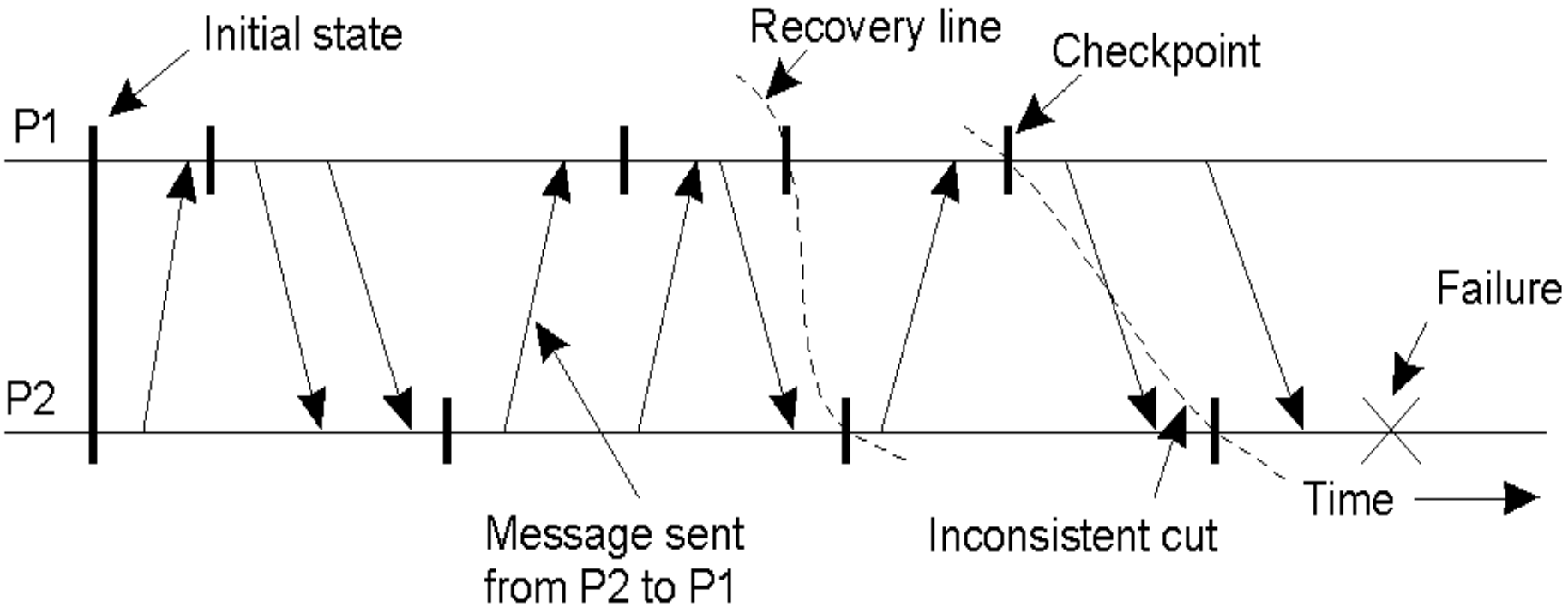


Figura 8.20 (a) Armazenamento estável. (b) Queda após a unidade 1 ter sido atualizada. (c) Local com erro.

Linha de Recuperação

Checkpointing



Registro de Mensagens

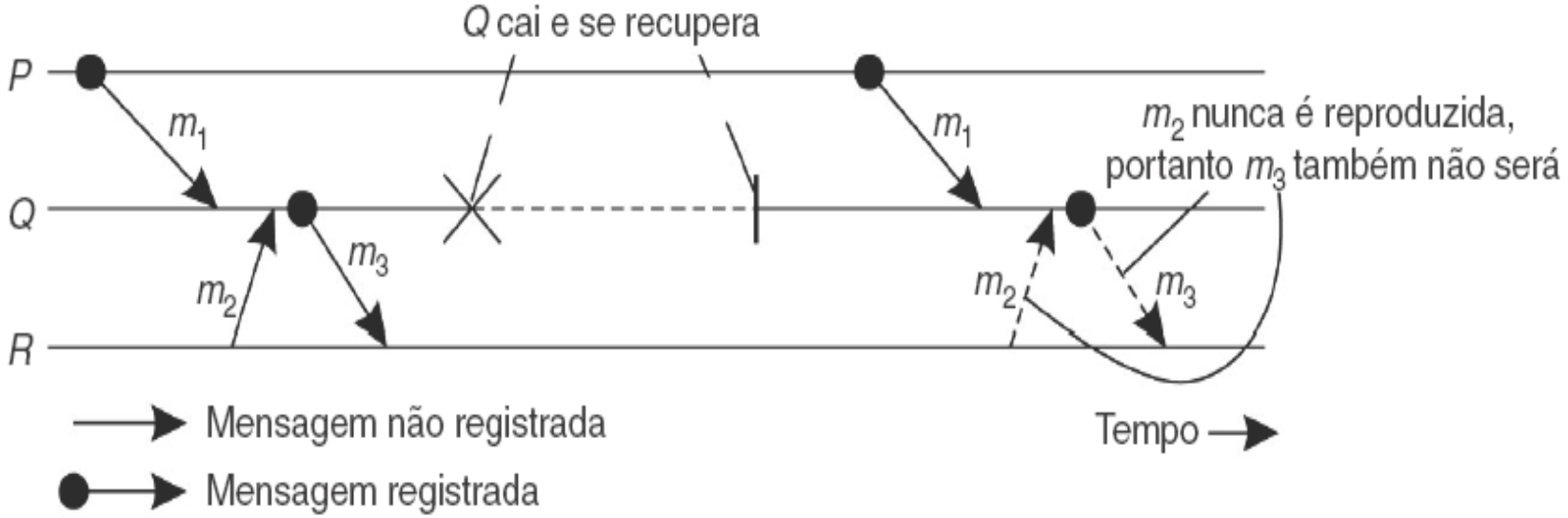


Figura 8.23 Reprodução incorreta de mensagens após recuperação, resultando em um processo órfão.