Распределенные системы

Отказоустойчивость

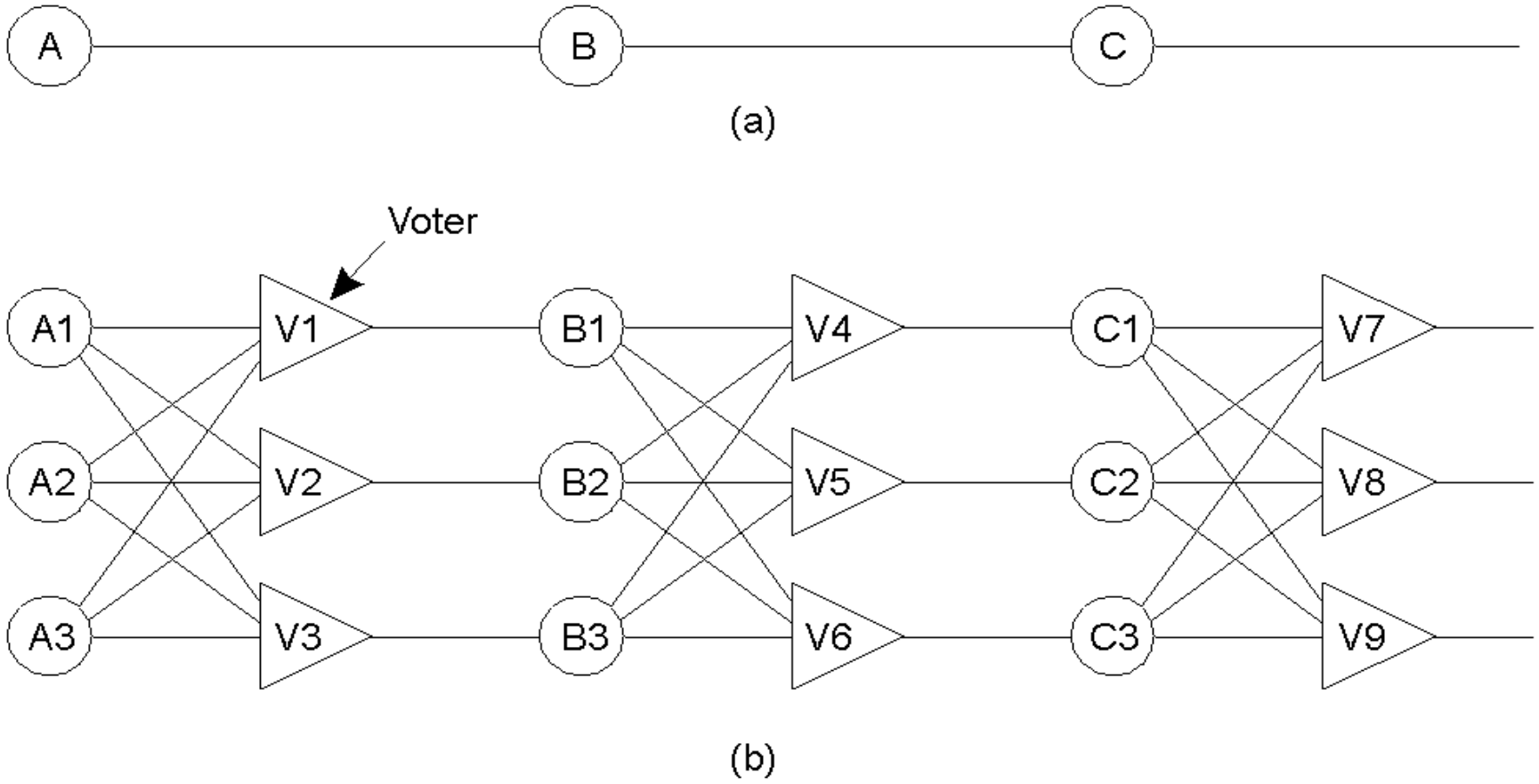
Основные концепции

- Надежность - это
  - Доступность
  - Безотказность
  - Безопасность
  - Ремонтопригодность

Модель отказов

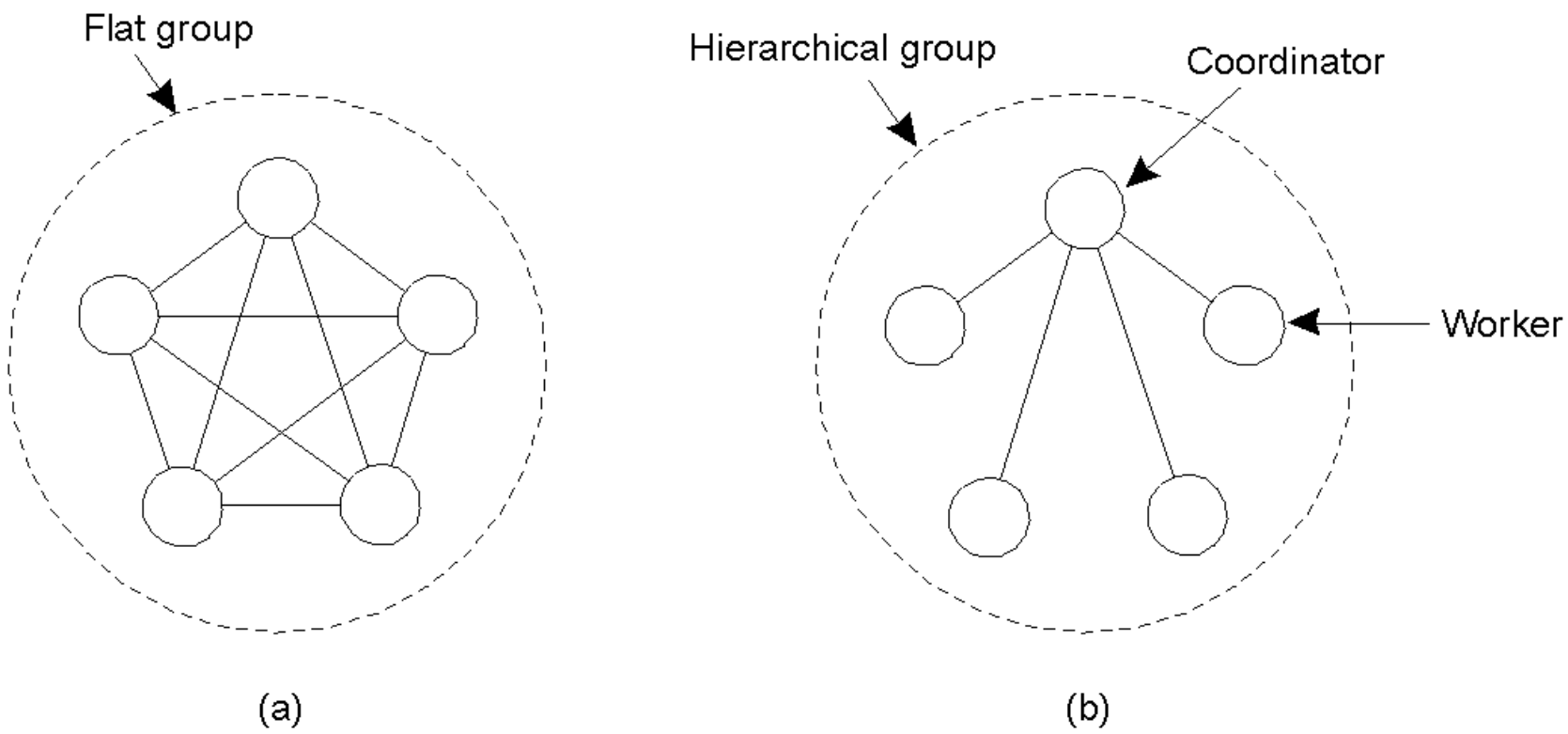
- Различные типы отказов
  - Поломка (crash)
  - Пропуск данных
    - Пропуск приема
    - Пропуск передачи
  - Ошибка синхронизации
  - Ошибка отклика
    - Ошибка значения
    - Ошибка передачи состояния
  - Произвольная ошибка

Маскирование ошибок за счет избыточности

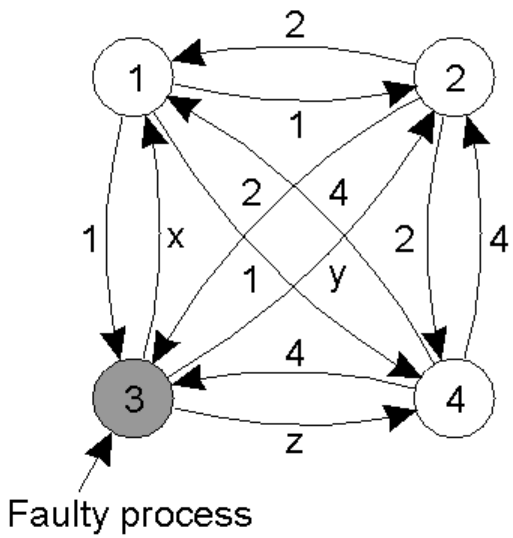


Отказоустойчивость процессов. Группы процессов

a) Одноранговые(плоские) группы
b) Иерархические группы



Соглашения в системах с ошибками



(a)

1 Got(1, 2, x, 4)
2 Got(1, 2, y, 4)
3 Got(1, 2, 3, 4)
4 Got(1, 2, z, 4)

(b)

1 Got	2 Got	4 Got
(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)

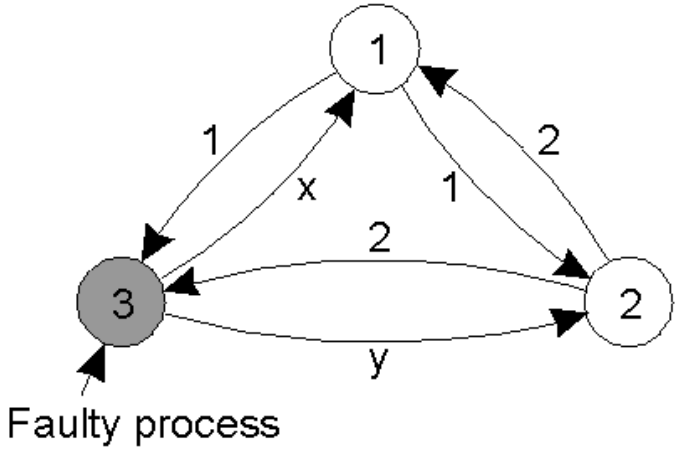
- Проблема Византийских генералов для 3-х лояльныз генералов и 1-го предателя

a) Генералы объявляют силу воиск

b) Векторы собранные каждым генералом

c) Обмен векторами

Соглашения в системах с ошибками



(a)

1 Got(1, 2, x)
2 Got(1, 2, y)
3 Got(1, 2, 3)

(b)

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

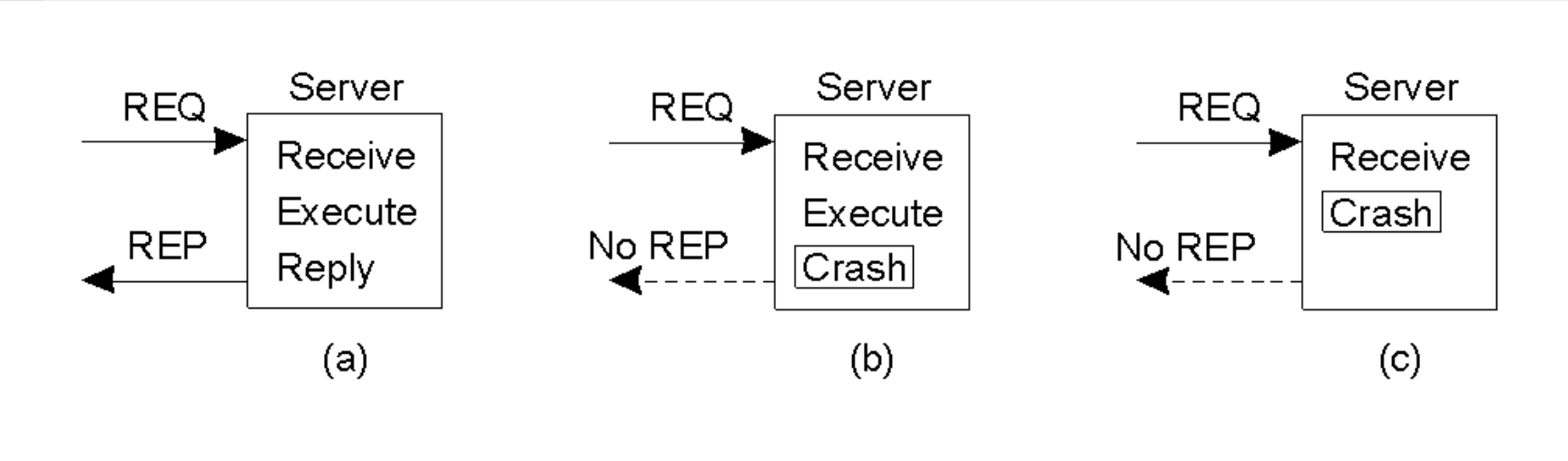
(c)

- Только 2 лояльных генерала
- 2m+1 лояльных процессов, на m дефектных

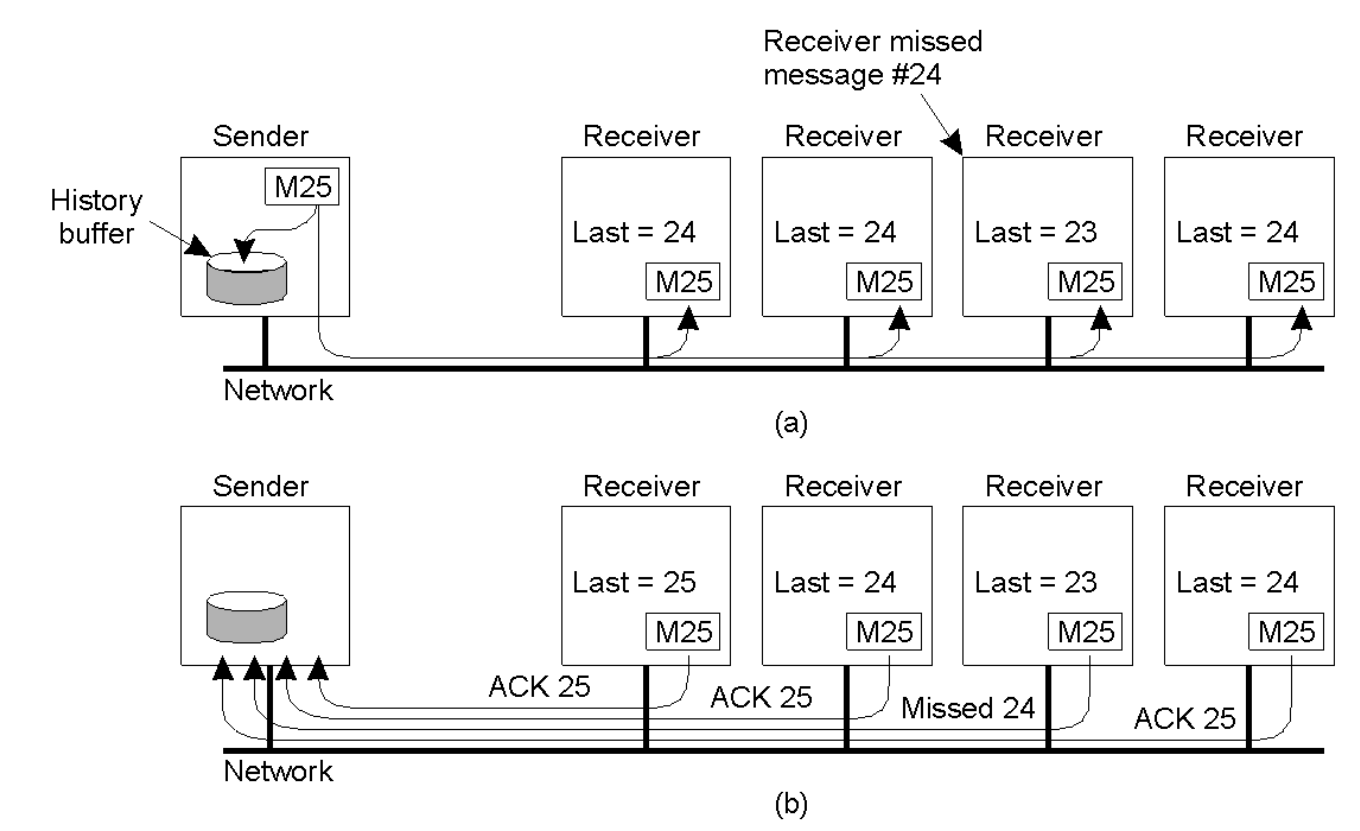
Надежная передача данных

- RPC взаимодействие, возможные ошибки
  - Сервер не обнаружен
  - Потеря сообщения с запросом от клиента к серверу
  - Сбой сервера после получения сообщения
  - Потеря ответного сообщения от сервера к клиенту
  - Поломка клиента после получения ответа

Потеря сообщения при сбое сервера



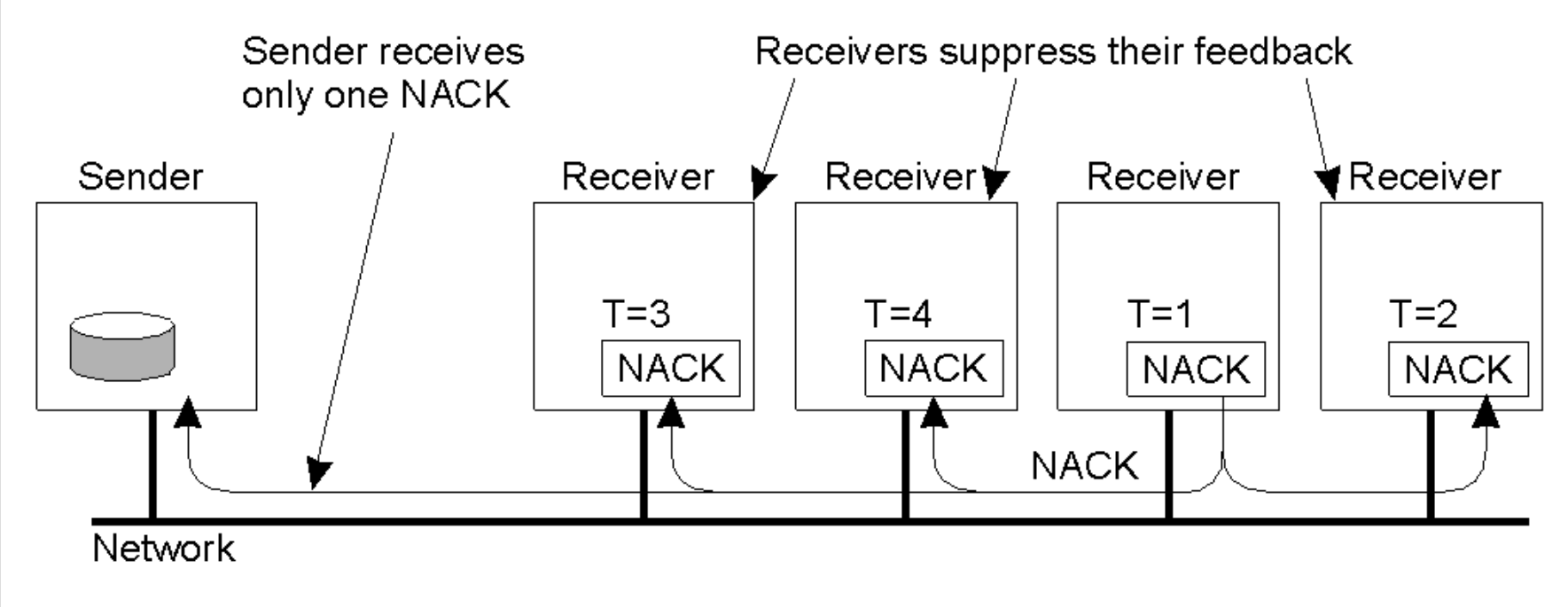
Базовая схема надежной групповой рассылки



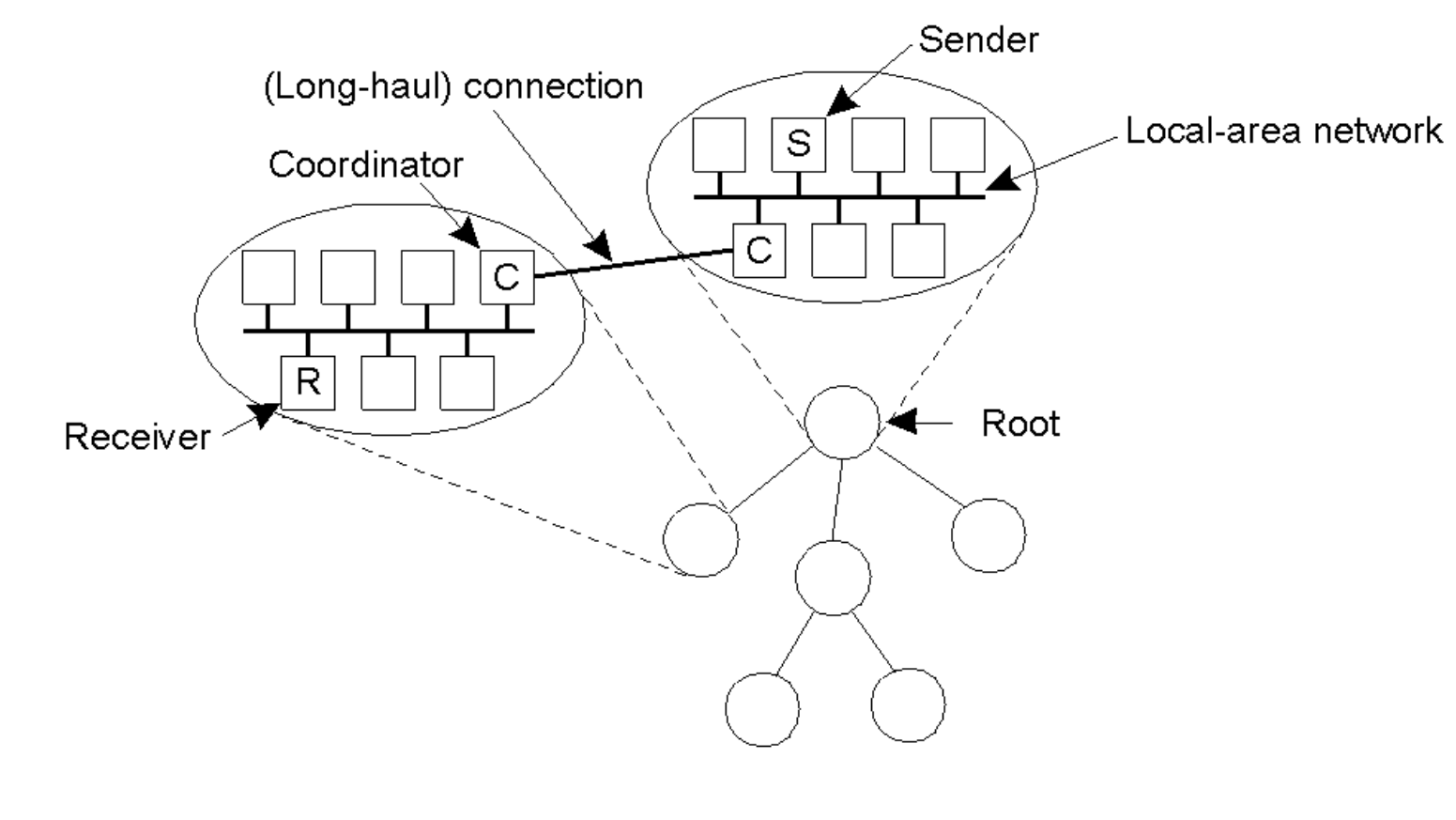
a) Отправка сообщения

b) Подтверждение

Неиерархическое управление обратной связью

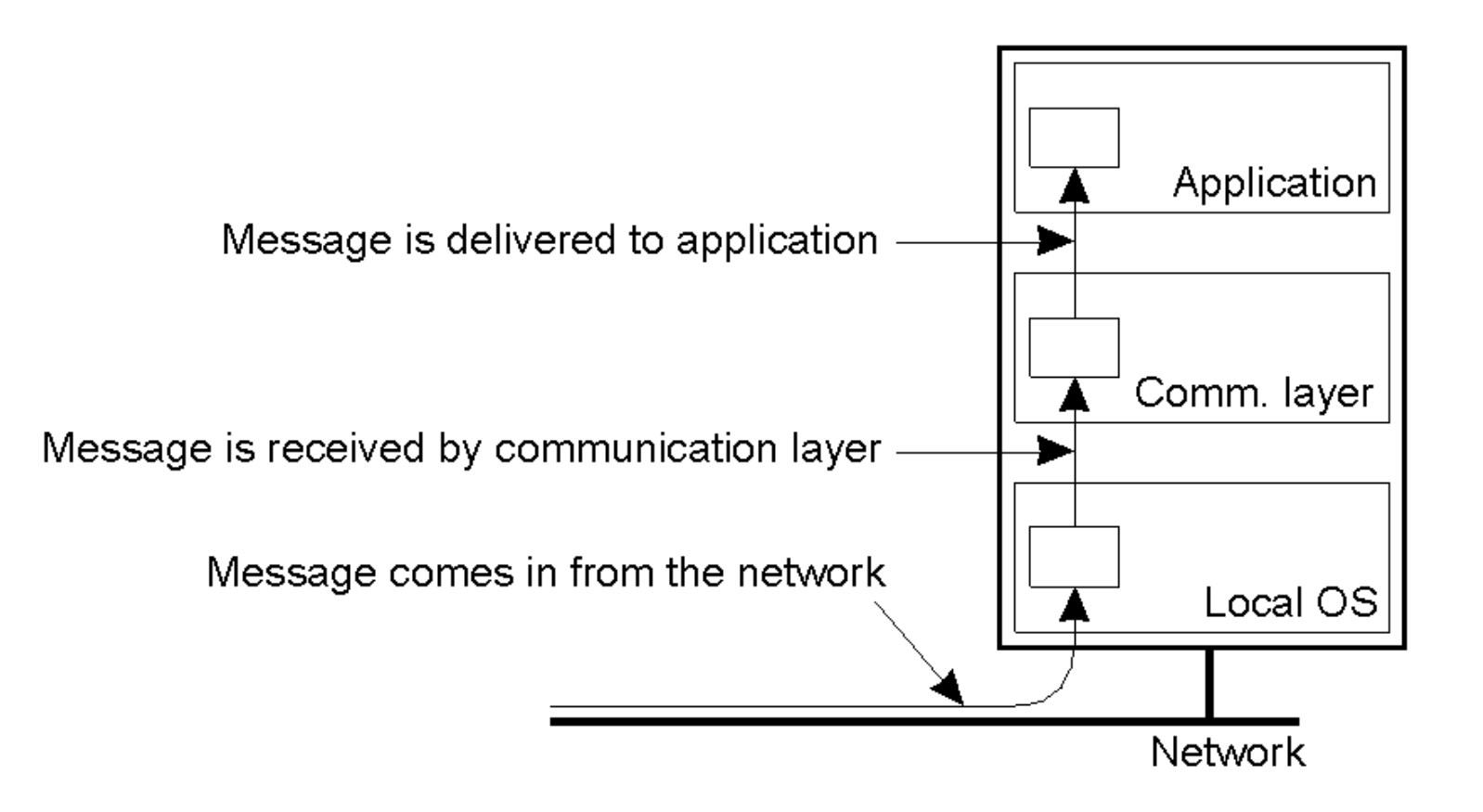


Иерархическое управление обратной связью

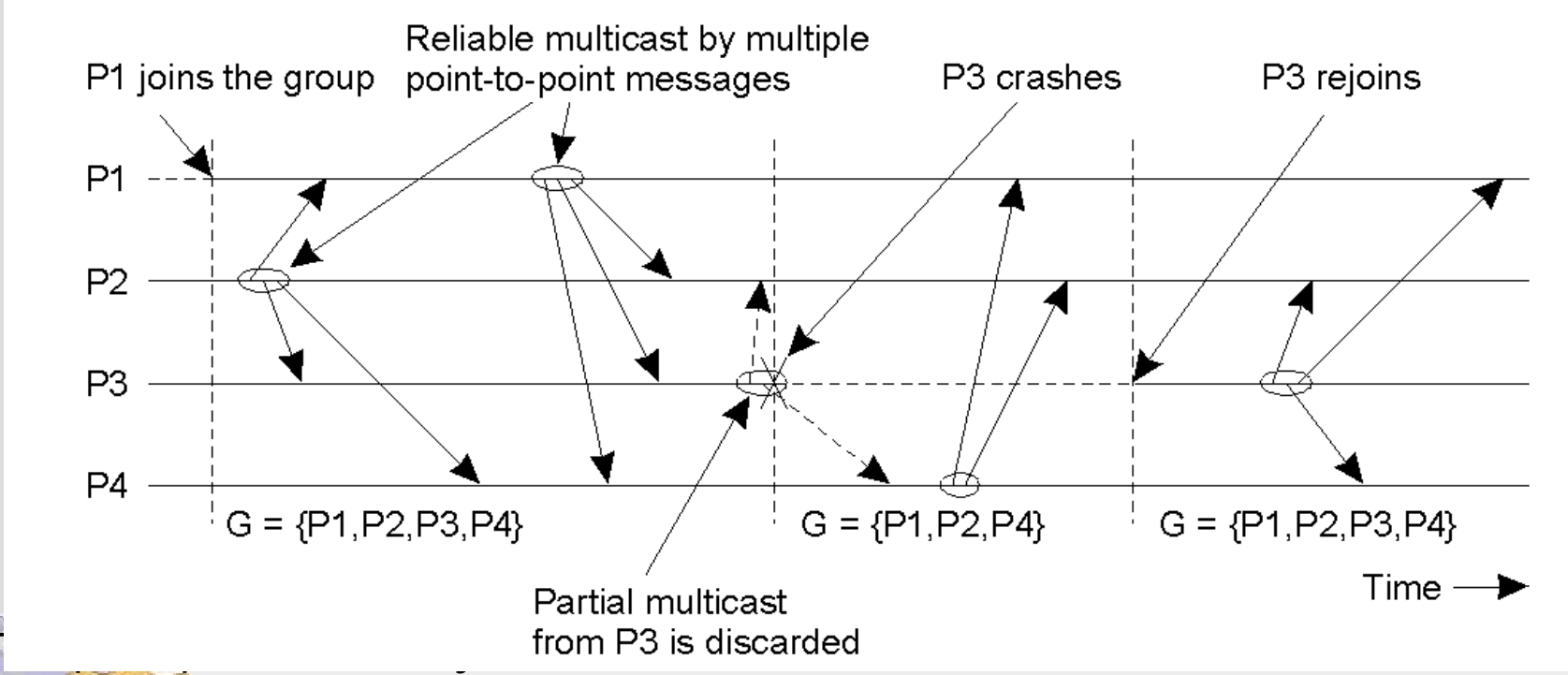


Виртуальная синхронность

- The logical organization of a distributed system to distinguish between message receipt and message delivery



Виртуальная синхронность



- T

Message Ordering (1)

Process P1	Process P2	Process P3
sends m1	receives m1	receives m2
sends m2	receives m2	receives m1

- Three communicating processes in the same group. The ordering of events per process is shown along the vertical axis.

Message Ordering (2)

Process P1	Process P2	Process P3	Process P4
sends m1	receives m1	receives m3	sends m3
sends m2	receives m3	receives m1	sends m4
	receives m2	receives m2	
	receives m4	receives m4	

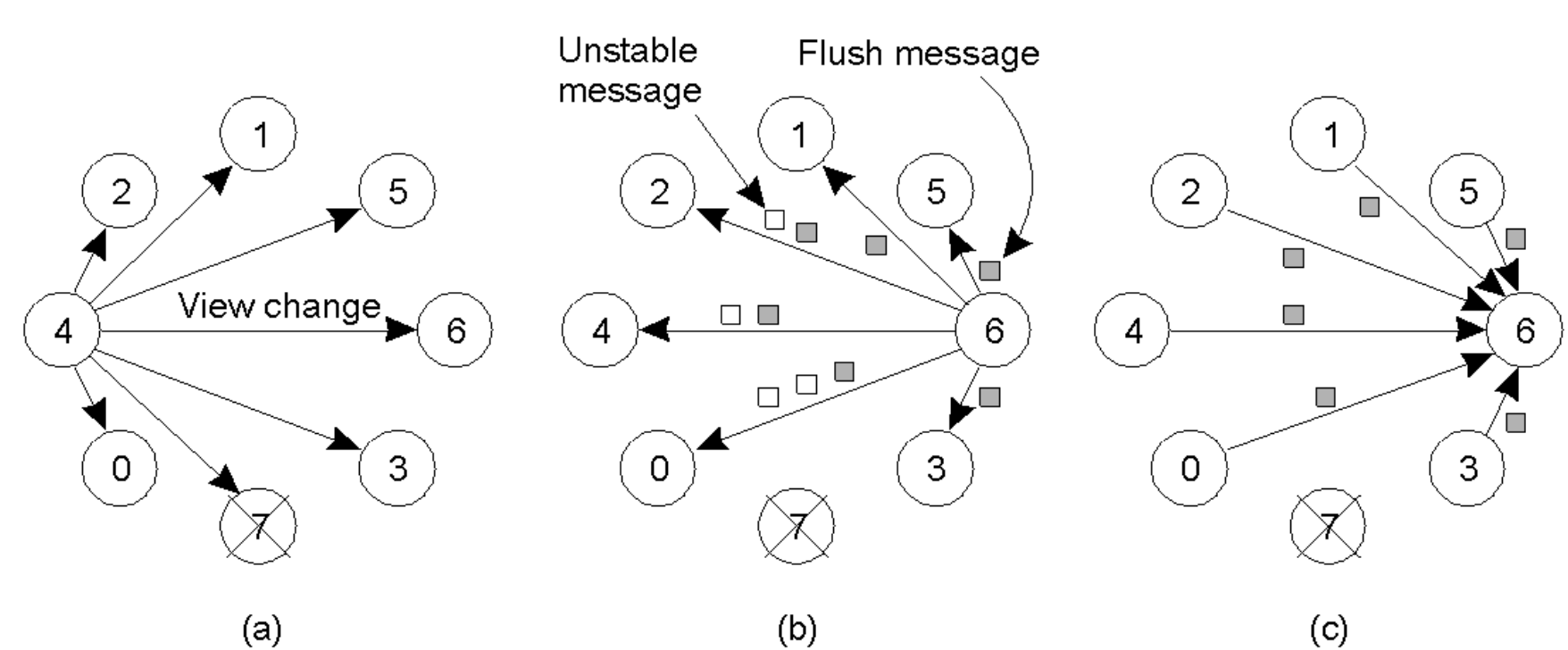
- Four processes in the same group with two different senders, and a possible delivery order of messages under FIFO-ordered multicasting

Implementing Virtual Synchrony (1)

- Six different versions of virtually synchronous reliable multicasting.

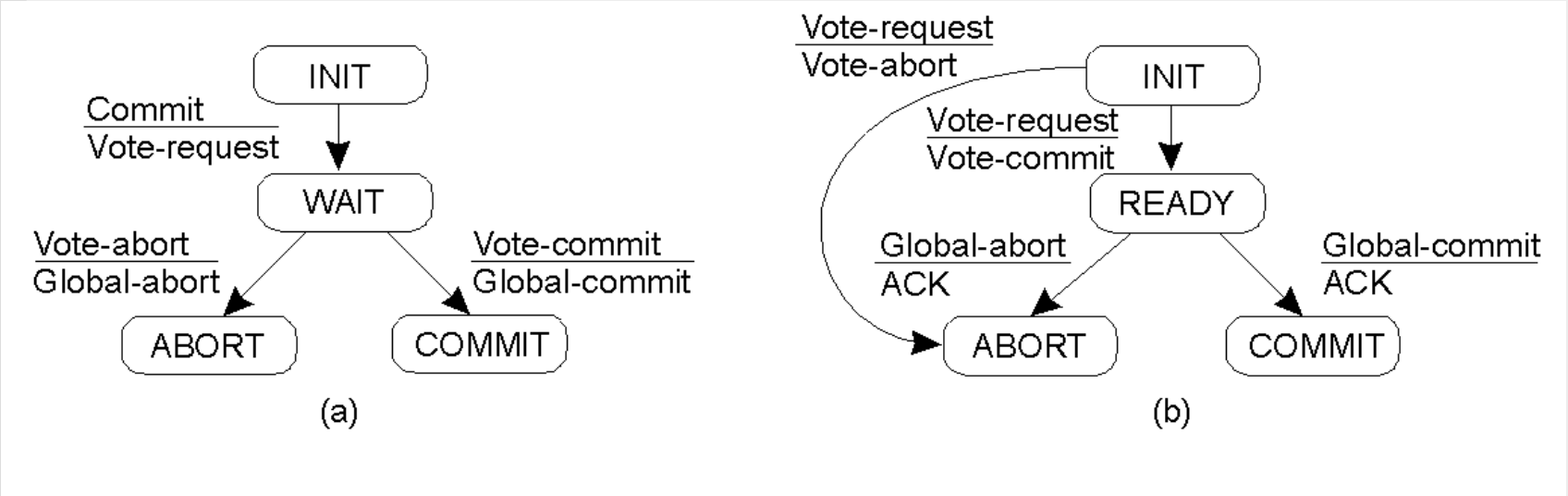
Multicast	Basic Message Ordering	Total-ordered Delivery?
Reliable multicast	None	No
FIFO multicast	FIFO-ordered delivery	No
Causal multicast	Causal-ordered delivery	No
Atomic multicast	None	Yes
FIFO atomic multicast	FIFO-ordered delivery	Yes
Causal atomic multicast	Causal-ordered delivery	Yes

Implementing Virtual Synchrony (2)



a) Process 4 notices that process 7 has crashed, sends a view change
b) Process 6 sends out all its unstable messages, followed by a flush message
c) Process 6 installs the new view when it has received a flush message from everyone else

Two-Phase Commit (1)



a) The finite state machine for the coordinator in 2PC.

b) The finite state machine for a participant.

Two-Phase Commit (2)

State of Q	Action by P
COMMIT	Make transition to COMMIT
ABORT	Make transition to ABORT
INIT	Make transition to ABORT
READY	Contact another participant

- Actions taken by a participant *P* when residing in state *READY* and having contacted another participant *Q*.

Two-Phase Commit (3)

actions by coordinator:

```
while START _2PC to local log;
multicast VOTE_REQUEST to all participants;
while not all votes have been collected {
    wait for any incoming vote;
    if timeout {
        while GLOBAL_ABORT to local log;
        multicast  GLOBAL_ABORT to all participants;
        exit;
    }
    record vote;
}
if all participants sent VOTE_COMMIT and coordinator votes COMMIT{
    write GLOBAL_COMMIT to local log;
    multicast GLOBAL_COMMIT to all participants;
} else {
    write GLOBAL_ABORT  to local log;
    multicast GLOBAL_ABORT to all participants;
}
```

- Outline of the steps taken by the coordinator in a two phase commit protocol

Two-Phase Commit (4)

- Steps taken by participant process in 2PC.

actions by participant:

```
write INIT to local log;
wait for VOTE_REQUEST from coordinator;
if timeout {
    write VOTE_ABORT to local log;
    exit;
}
if participant votes COMMIT {
    write VOTE_COMMIT to local log;
    send VOTE_COMMIT to coordinator;
    wait for DECISION from coordinator;
    if timeout {
        multicast DECISION_REQUEST to other participants;
        wait until DECISION is received; /* remain blocked */
        write DECISION to local log;
    }
    if DECISION == GLOBAL_COMMIT
        write GLOBAL_COMMIT to local log;
    else if DECISION == GLOBAL_ABORT
        write GLOBAL_ABORT to local log;
} else {
    write VOTE_ABORT to local log;
    send  VOTE ABORT to coordinator;
}
```

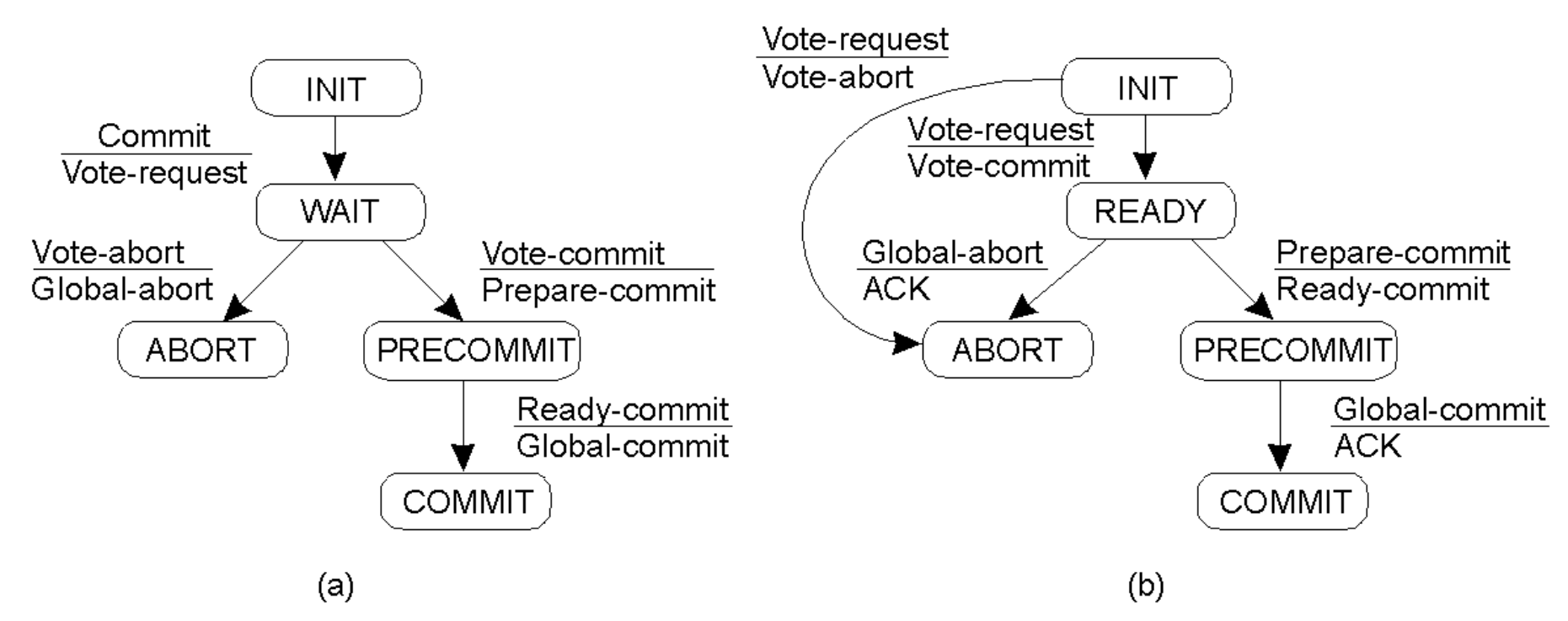
Two-Phase Commit (5)

actions for handling decision requests: /* executed by separate thread */

```
while true {
    wait until any incoming DECISION_REQUEST is received; /* remain blocked */
    read most recently recorded STATE from the local log;
    if STATE == GLOBAL_COMMIT
        send GLOBAL_COMMIT to requesting participant;
    else if STATE == INIT or STATE == GLOBAL_ABORT
        send GLOBAL_ABORT to requesting participant;
    else
        skip;  /* participant remains blocked */
```

- Steps taken for handling incoming decision requests.

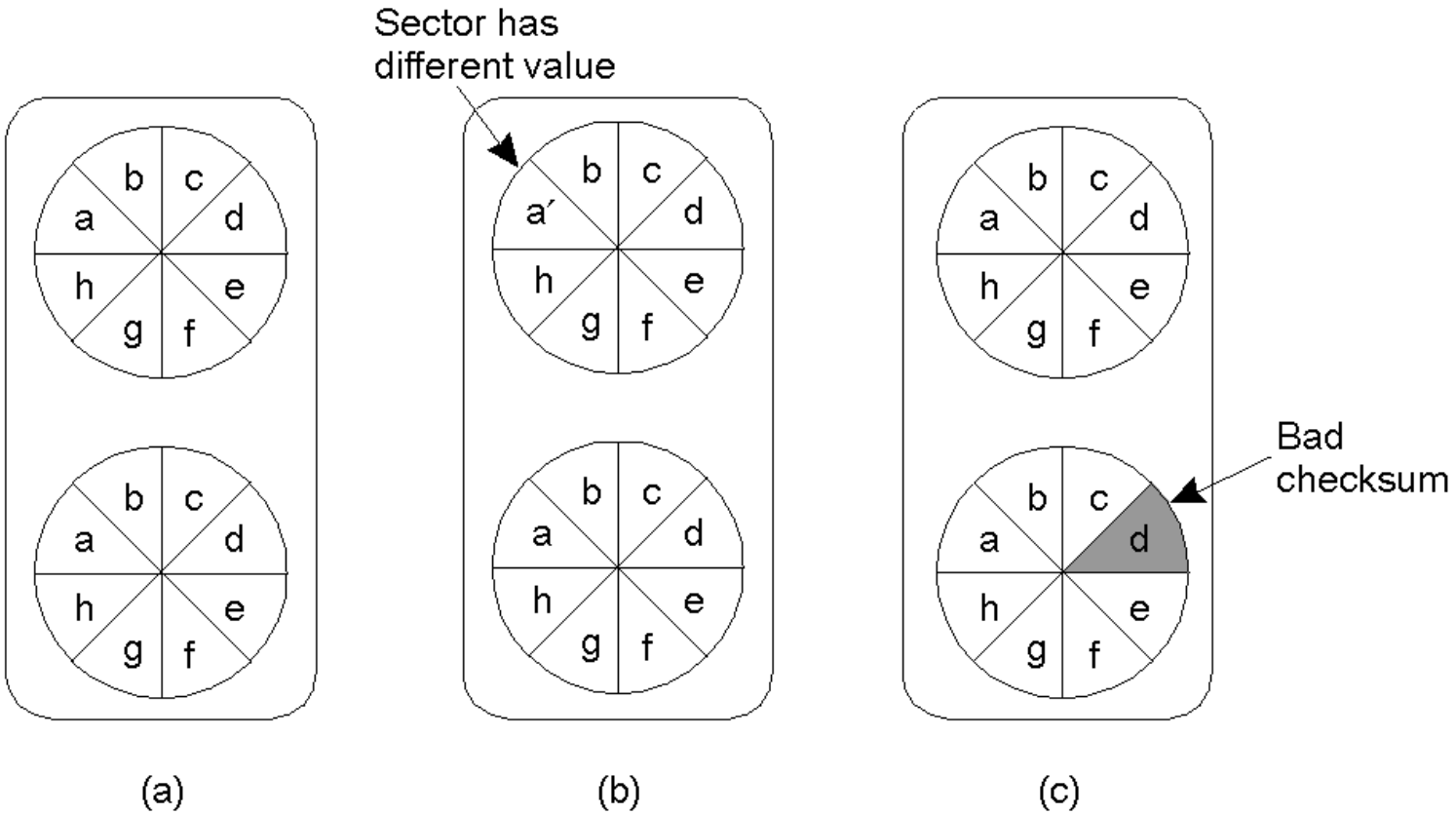
Three-Phase Commit



a) Finite state machine for the coordinator in 3PC

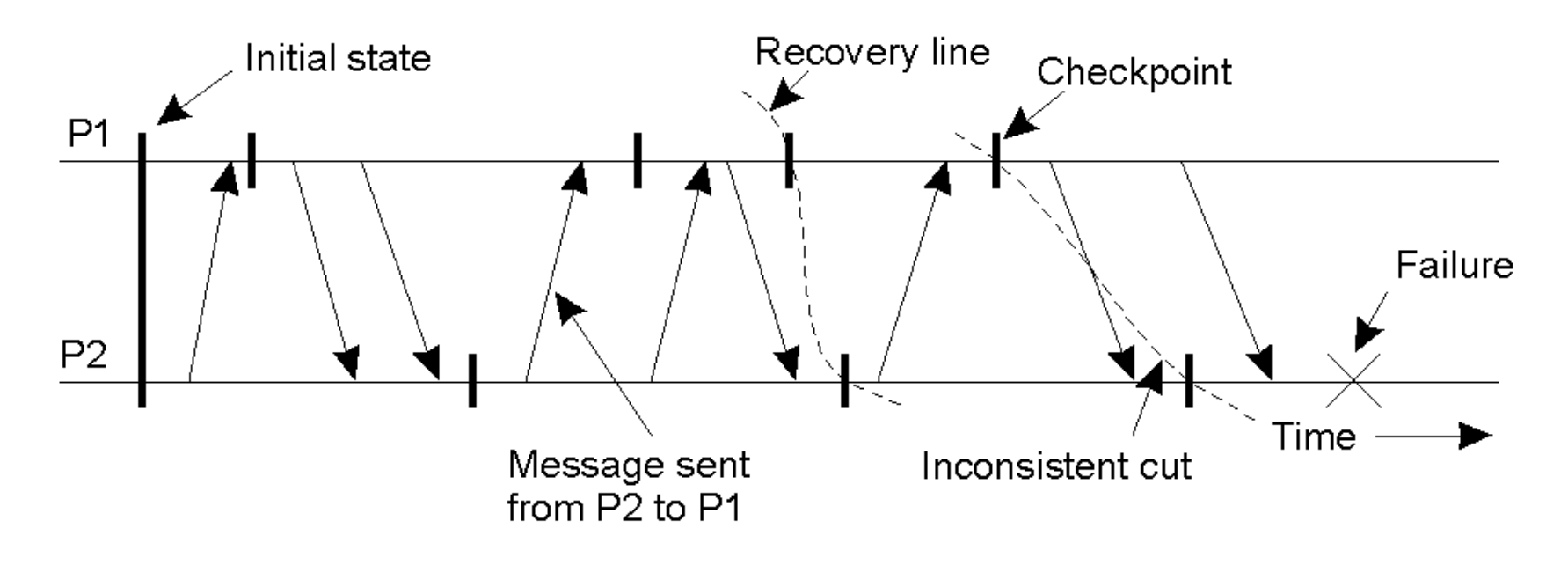
b) Finite state machine for a participant

Recovery Stable Storage

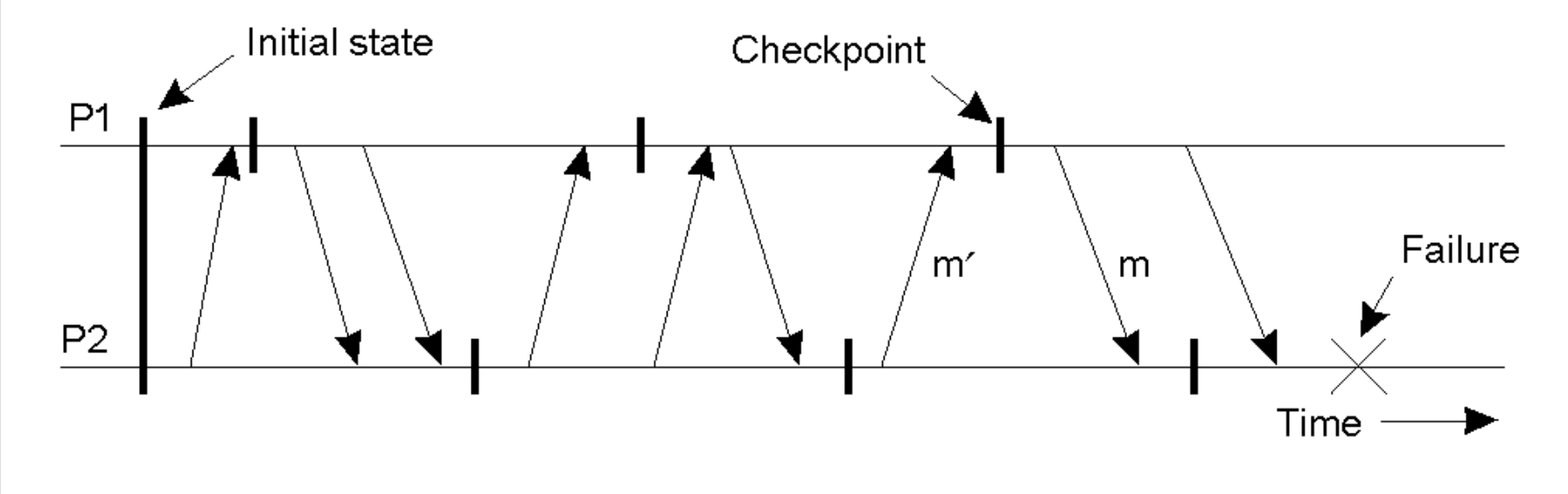


a) Stable Storage
b) Crash after drive 1 is updated
c) Bad spot

Checkpointing

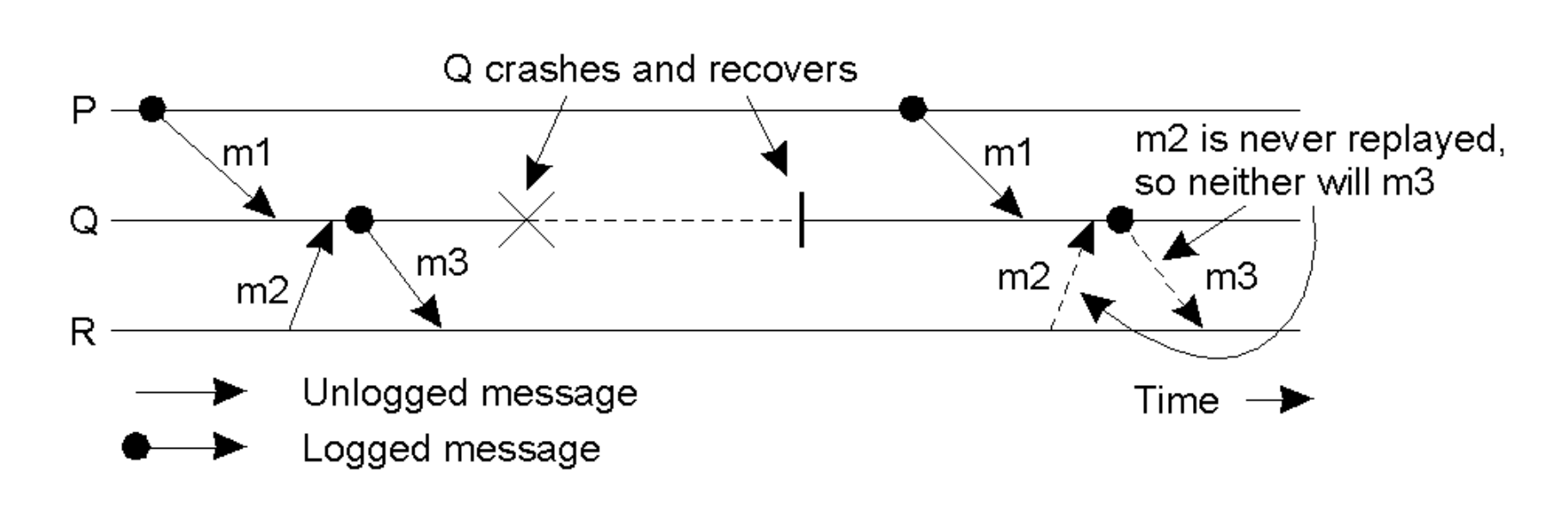


Independent Checkpointing



- The domino effect.

Message Logging



- Incorrect replay of messages after recovery, leading to an orphan process.