第八章　容错性

- 容错性简介
- 进程恢复
- 可靠的C-S通信
- 可靠的组通信
- 分布式提交
- 恢复

容错性简介

- 容错意味着系统即使发生故障也能提供服务
- 容错与可靠性相联系，包含以下需求：
  - 可用性（**Availability**）：任何给定的时刻都能及时工作
  - 可靠性（**Reliability**）：系统可以无故障地持续运行
  - 安全性（**Safety**）：系统偶然出现故障能正常操作而不会造成任何灾难
  - 可维护性（**Maintainability**）：发生故障的系统被恢复的难易程度

故障模型

- 造成错误的原因称为故障
- 故障分为
  - 暂时故障：只发生一次
  - 间歇故障：反复间隔发生（接触不良）
  - 持久故障：持续存在的故障（软件错误、磁盘头损坏）

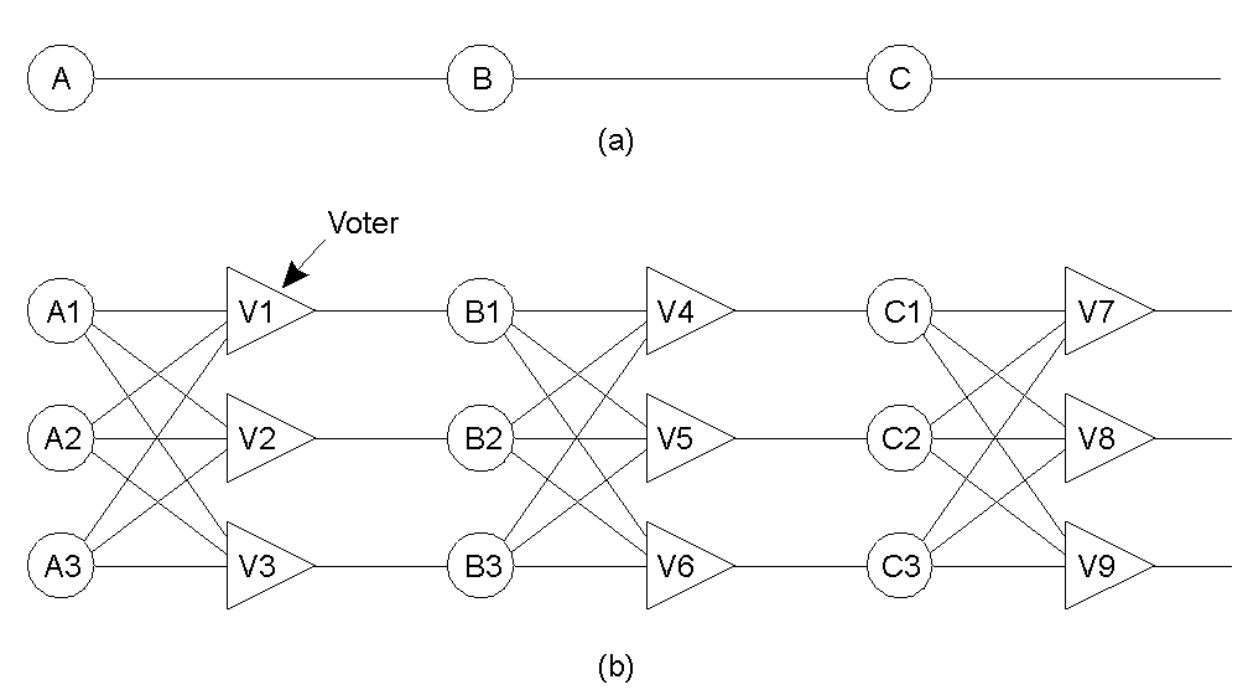
故障模型

不同类型的故障

故障类型	描述
崩溃性故障	服务器过早停机，但在之前工作正常
遗漏性故障 *接收故障* *发送故障*	服务器不能响应到来的请求 服务器不能接收到来的消息 服务器不能发送消息
定时故障	服务器的响应超出了指定的实时要求
响应故障 *值故障* *状态转换故障*	服务器的响应不正确 响应值错误 服务器偏离了正确的控制流
随意性故障	服务器可能在任意的时间产生任意的错误

使用冗余来掩盖故障

- 使用冗余来掩盖故障
  - 信息冗余：增加信息，如海明码校验
  - 时间冗余：多次执行一个动作，如事务
  - 物理冗余：增加额外的设备或进程



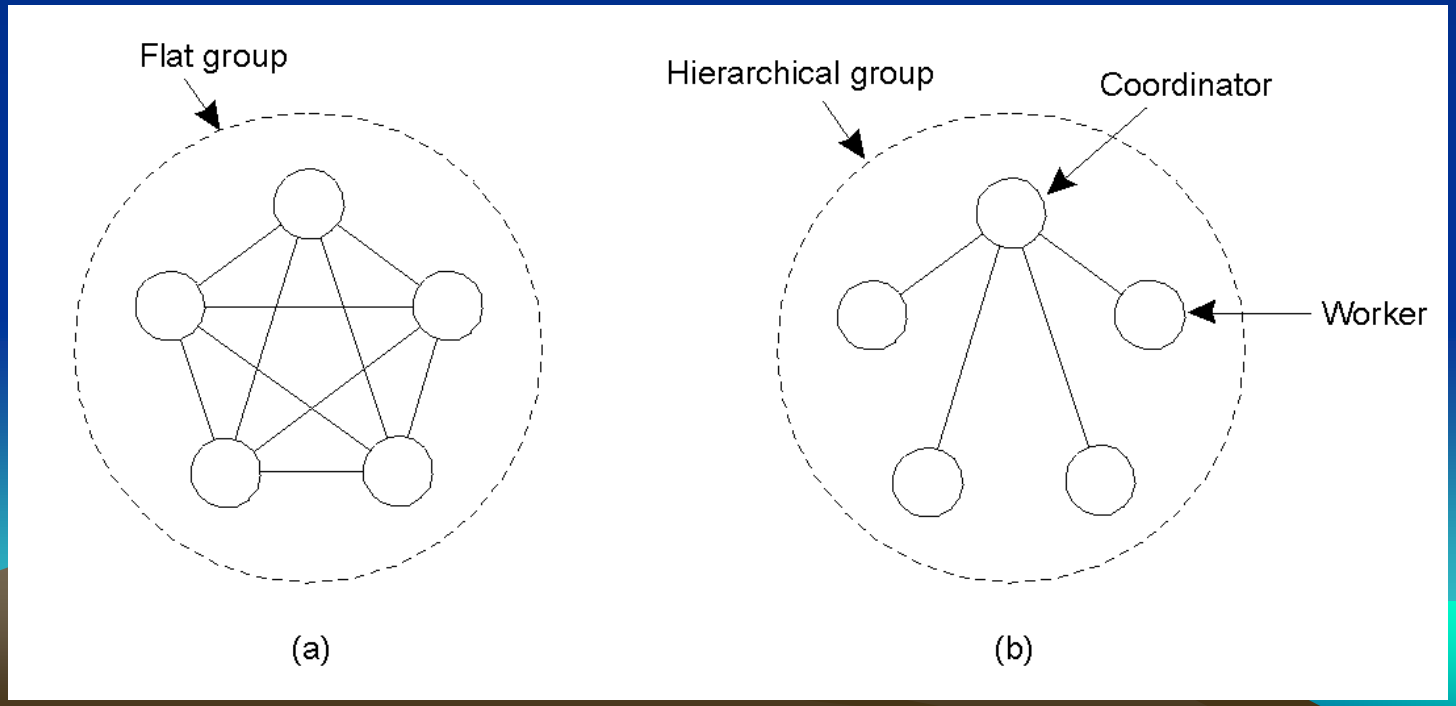
三倍的模块冗余

进程恢复
平等组与等级组

- 为防止进程失败，把进程复制到组
- 当消息发送到组时，组中所有成员都接收它，一个进程失败，其他进程可以接管它
- 进程组是动态的

a) 平等组通信：增加延迟和开销

b) 简单等级组通信：单点失效



组成员

- 组通信时，需要创建和删除组，以及允许进程加入和离开
  - 使用组管理器：存储相应数据库，直接、有效、容易实现；单点失败
  - 分布式的方法：
    - 加入组: 发消息给所有的组成员
    - 离开组: 发消息给所有的组成员，需考虑崩溃的情况
    - 进程加入和离开必须与数据消息的发送同步
    - 重建组

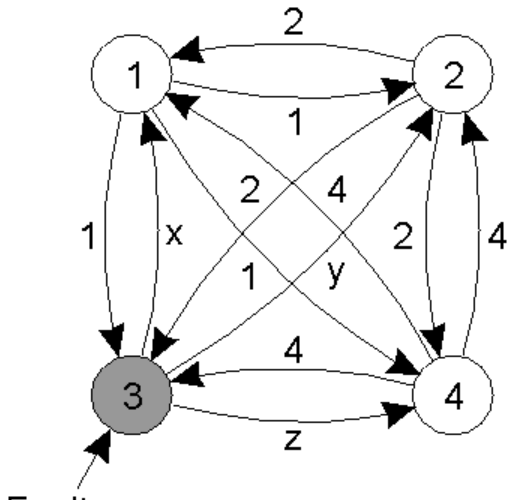
故障掩盖和复制

- 复制进程，用一个容错的进程组来代替一个脆弱的进程
- 需要多少复制?
  - 如果系统能经受K个组件的故障而且能满足规范的要求,被称为K容错的
  - 如果组件是失败沉默的，具有K+1个组件即可
  - 如果组件发生拜占庭错误（ Byzantine fault ），继续错误运行，则至少需要2K+1个组件才能获得K容错
  - 拜占庭错误：在非失败沉默模型下，一个有故障的进程可能会对其它进程发出干扰消息，从而影响这些进程的正常工作。
  - 拜占庭错误是所有故障类型中最严重的

故障系统的协议 (1)

- 分布式协议算法的目标是使所有的非故障进程就一些问题在有限步骤内达成一致
- 通信是否可靠：两军问题
- 进程故障：拜占庭将军问题
- **Lamport** 证明在具有**m**个故障进程的系统中，只有存在**2m+1**的正常工作的进程才能达成协议

故障系统的协议 (2)



(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)

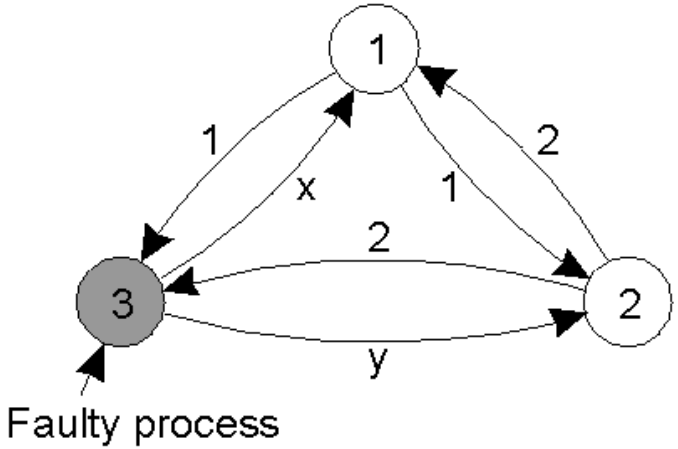
三个忠诚将军和一个叛徒的问题

a) 将军宣布他们的兵力

b) 在(a)基础上每个将军的向量

c) 每个将军收到的向量

故障系统的协议(3)



(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)

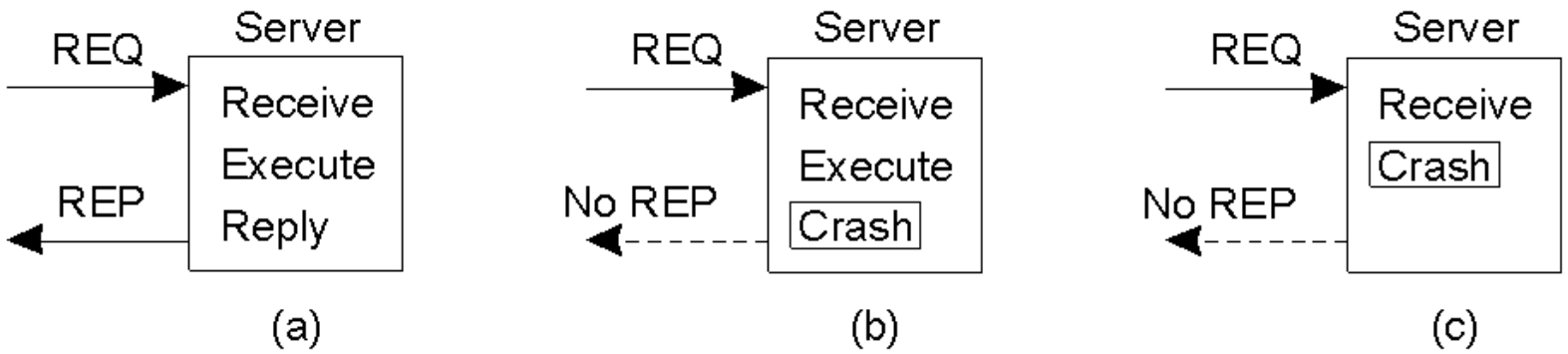
两个忠诚将军和一个叛徒的问题

可靠的C-S通信

RPC系统失败的五种情况：

- 客户不能定位到服务器
- 客户到服务器的请求消息丢失：使用定时器
- 服务器在收到请求后崩溃
  - 最少一次语义： 再次尝试操作，将应答传给用户，RPC最少执行一次
  - 最多一次语义： 放弃并报告失败，RPC最多执行一次
- 从服务器到客户的响应消息丢失：
  - 使用定时器;
  - 幂等操作;
  - 为每个请求分配一个序列号
- 客户在发送请求后崩溃：孤儿进程
  - 浪费资源
  - 处理孤儿进程的方法
    - 消灭：客户重启后根据客户端日志清除孤儿进程
    - 再生：将时间分为顺序编号的时期，客户重启后广播清除孤儿进程
    - 优雅再生：找不到拥有者，再清除孤儿进程
    - 到期：给每个RPC指定标准的执行时间

服务器崩溃
Server Crashes (1)



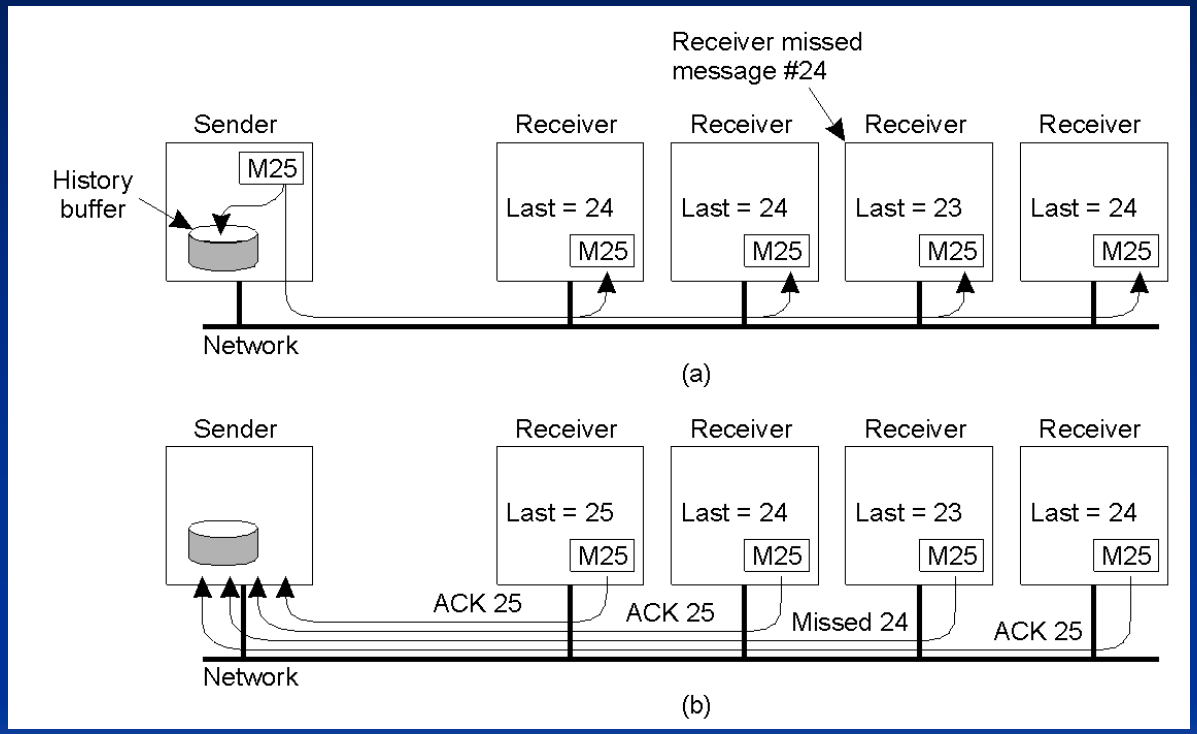
client-server 通信中的服务器

a) 通常情况
b) 执行后崩溃
c) 执行前崩溃

可靠的组通信

- 可靠多播:发送到一个进程组的消息被传递到该组的每个成员
- 问题:
  - 如果通信期间有进程加入
  - 如果通信期间一个（发送）进程崩溃
- 基本的可靠多播方法：
  - 假定所有的接收者已知而且假定不会失败的简单可靠多播方法
- 可靠多播的可扩展性
- 原子多播：实现存在进程失败的情况下的可靠多播

可靠的组通信--基本的可靠多播方法



当所有的接收者已知而且假定不会失败的简单的可靠多播方法

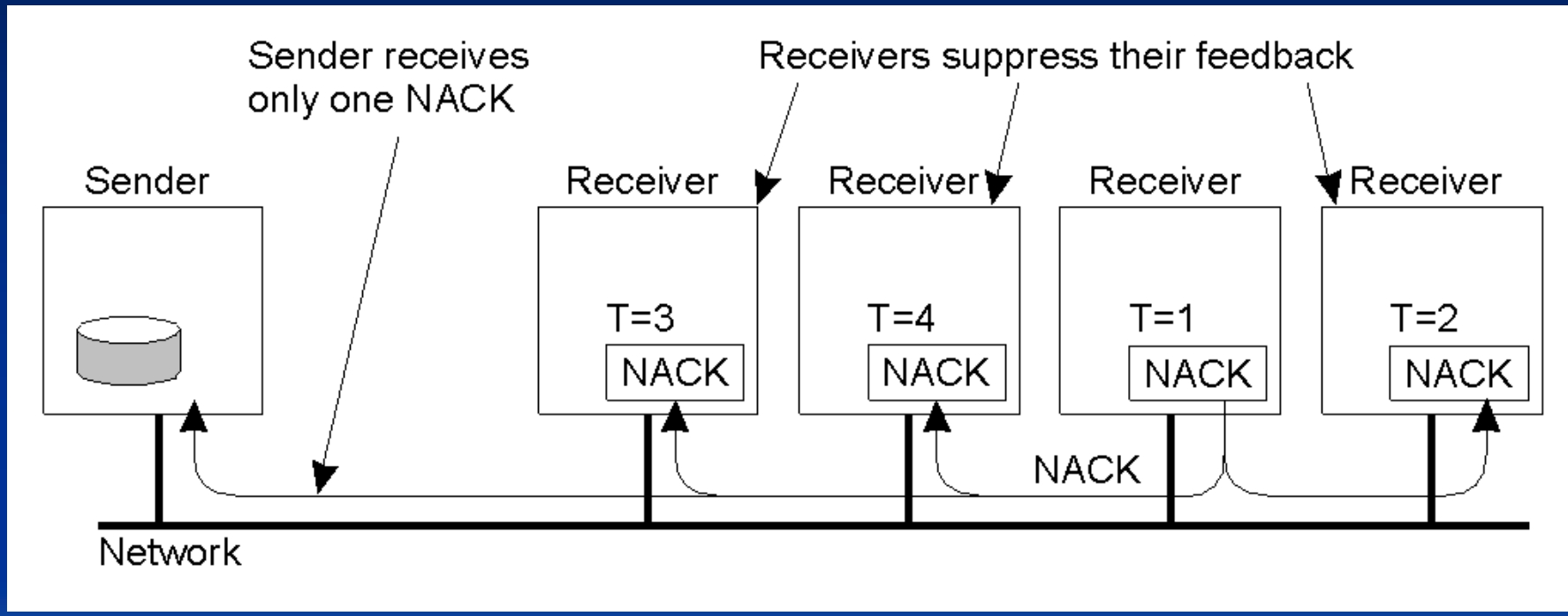
a)消息传递

b)反馈

可靠多播的可扩展性

- 上面介绍的可靠多播方法不能支持过多的接收者：反馈拥塞
- 解决办法：
  - 接收者不反馈，只有通知消息丢失时反馈一消息
    - 不能保证永远不发生反馈拥塞
    - 发送者需要一直在缓存器中保留消息
  - 无等级的反馈控制
  - 分等级的反馈控制

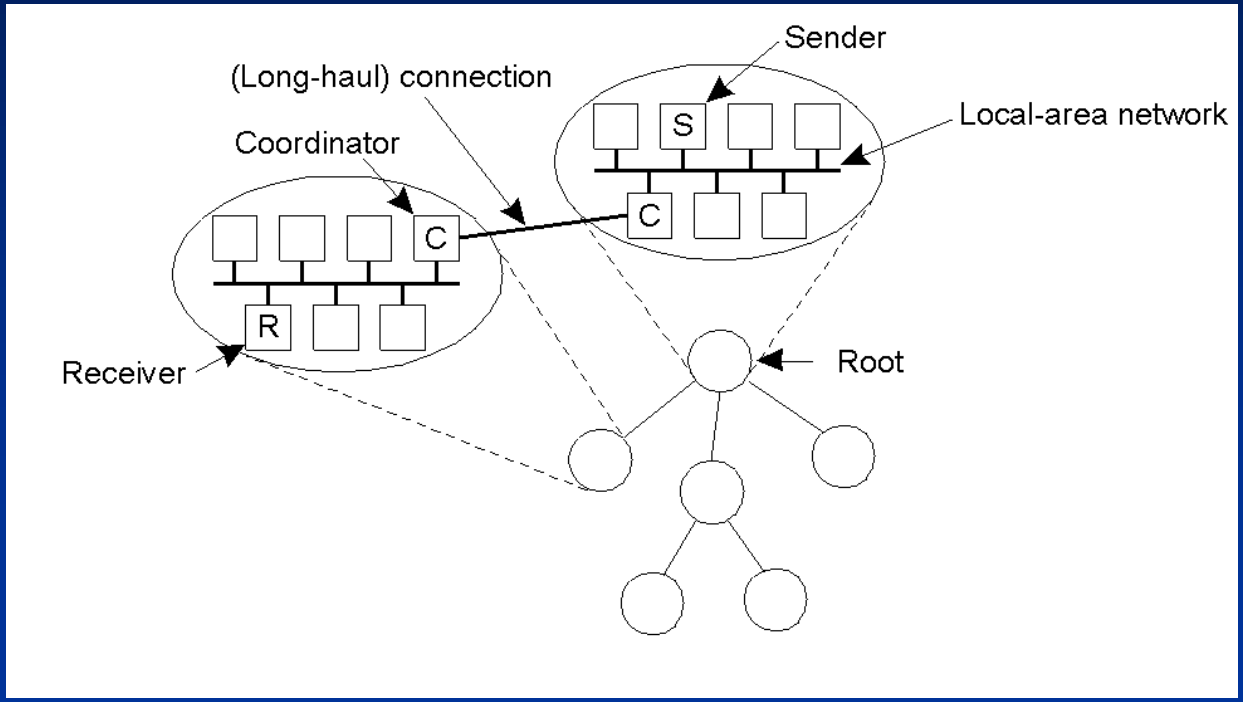
无等级的反馈控制
Nonhierarchical Feedback Control



反馈抑制：几个接收者要发送重发请求，但是第一个重发请求抑制了其他的请求。
具有很好的可扩展性
问题：

- 需要每个接收者对反馈消息进行准确的调度，否则还会有多个接收者同时反馈
- 中断其他成功接收消息的进程

分等级的反馈控制
Hierarchical Feedback Control



- 在非常大的接收组中获得扩展性
- 多等级的可靠多播：每个本地协调者都把消息转发给它的孩子然后再处理重发请求
- 每个子组内可使用适合小组的可靠多播方式
- 协调者有自己的缓存器，如果自身丢失消息，则请求父组的协调者重发消息
- 在基于确认的方法中，如果收到消息，协调者向父亲发送确认。如果协调者从子组的所有成员和它的孩子得到对消息m的确认，则删除消息m

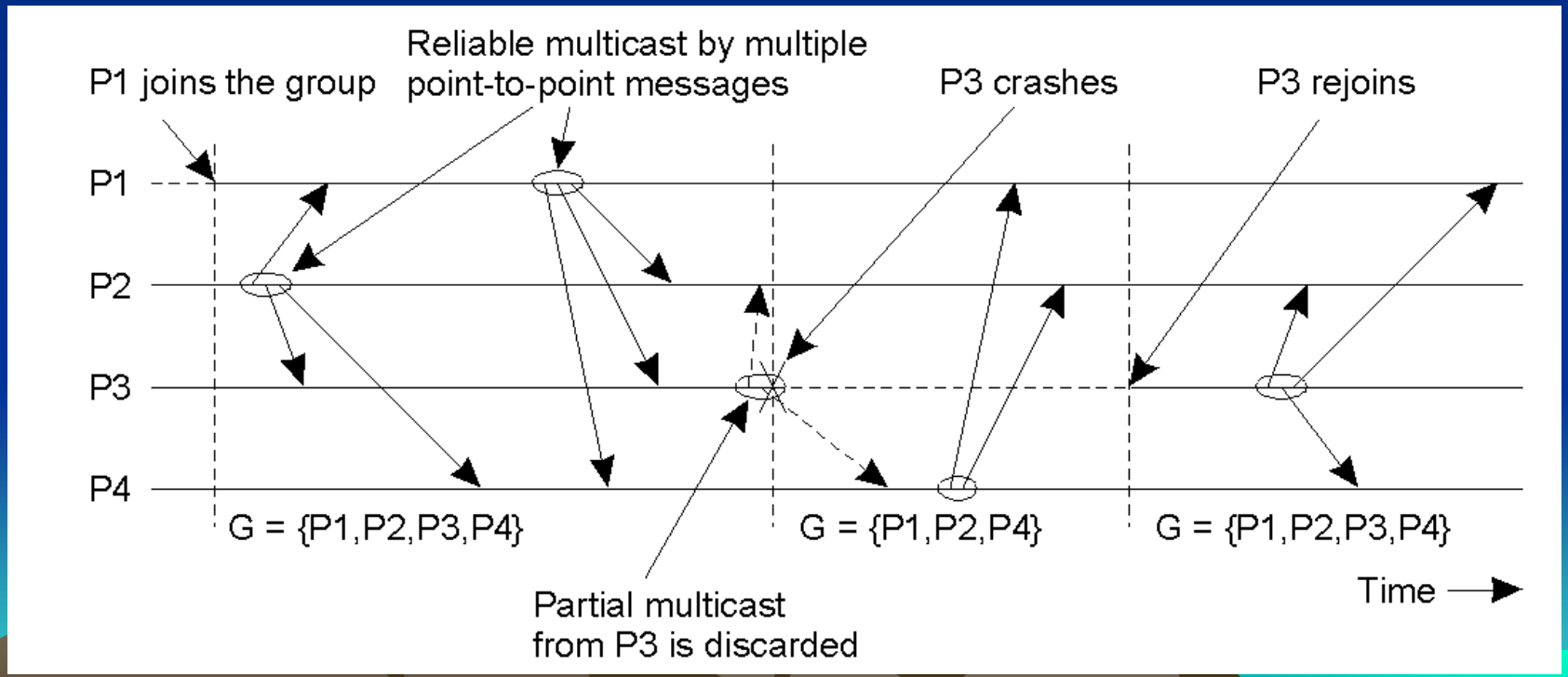
原子多播

- 需要在存在进程失败的情况下获得可靠多播的情况
- 原子多播:
  - 消息要么发送给所有进程，要么一个也不发送
  - 通常需要所有的消息都按相同的顺序发送给所有的进程
  - 分布式系统中的复制数据库
  - 原子多播确保没有故障的进程对数据库保持一致；当一个副本从故障中恢复并重新加入组时，原子多播强制它与其他组成员按保持一致
- 虚拟同步
- 消息排序

虚拟同步
Virtual Synchrony (2)

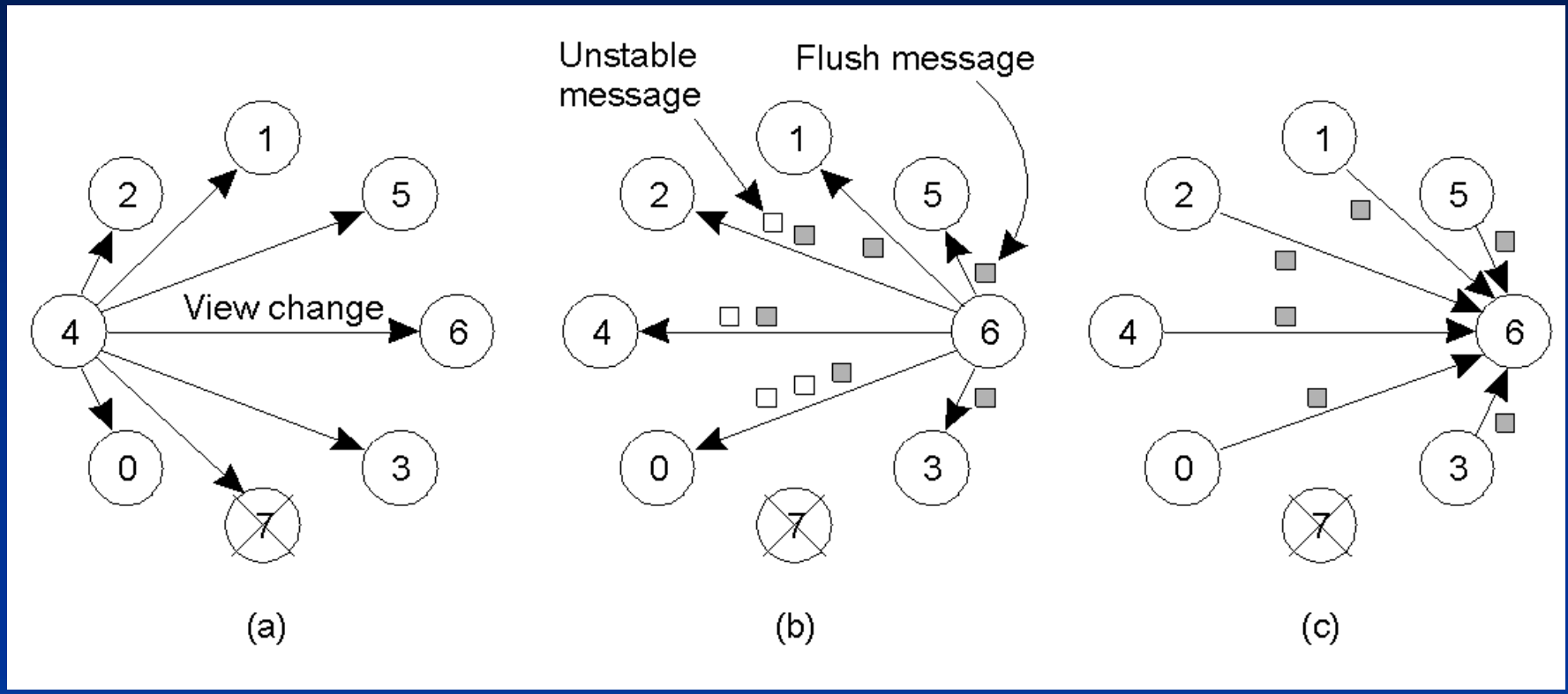
虚拟同步：

- 保证多播到组视图的消息被传送给组中的每个正常进程
- 如果发送消息的进程在多播期间失败，则消息或者传递给剩余的所有进程，或者被每个进程忽略
- 所有多播都在视图改变之间进行



虚拟同步多播的原理

实现虚拟同步



保证发送到组视图G的所有消息在组成员关系改变之前发送到G中的所有正常进程

- 进程 4 注意到进程 7 已经崩溃, 发送一个视图改变
- 进程 6 发送所有的不稳定消息（所有进程都收到的消息称为稳定消息），然后发送一个flush 消息
- 当进程 6 从其他每个进程收到 flush 消息后，建立一个新的视图

消息排序

- 可靠不排序的多播：对接收不同进程发送的消息的次序不做任何保证
- FIFO顺序的多播：按照消息发送的顺序传送同一进程的消息，对不同进程发送的消息的传送顺序没有约束
- 按因果关系排序的多播：按因果关系排序多播来保留消息间的因果关系
- 全序多播：无论消息传送是无序、 FIFO顺序还是按因果关系排序，对所有的组成员按相同的次序传送

消息排序 (1)--不排序的多播

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

同组中的三个通信进程，每个进程中时间的顺序按垂直顺序排列

消息排序 (2) -- FIFO顺序的多播

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	

同组中的四个通信进程，具有两个不同的发送者，在FIFO多播下可能的消息传送顺序

消息排序 (3) --全序多播

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

对所有的组成员按相同的次序传送

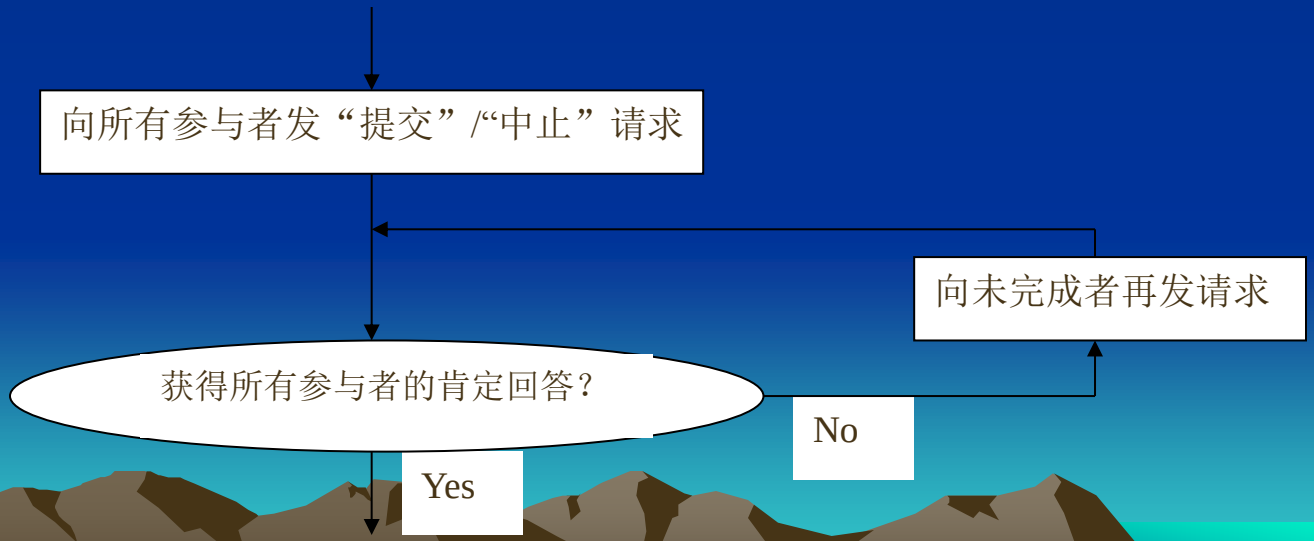
消息排序 (4)

- 六种不同的虚拟同步可靠传播
- 提供了全序的消息传送的虚拟同步可靠多播称为原子多播

多播	基本的消息排序	完全排序传送
可靠多播	无	不
FIFO 多播	FIFO排序传送	不
按因果关系多播	按因果关系传送	不
原子多播	无	是
FIFO 原子多播	FIFO排序传送	是
按因果关系的原子多播	按因果关系传送	是

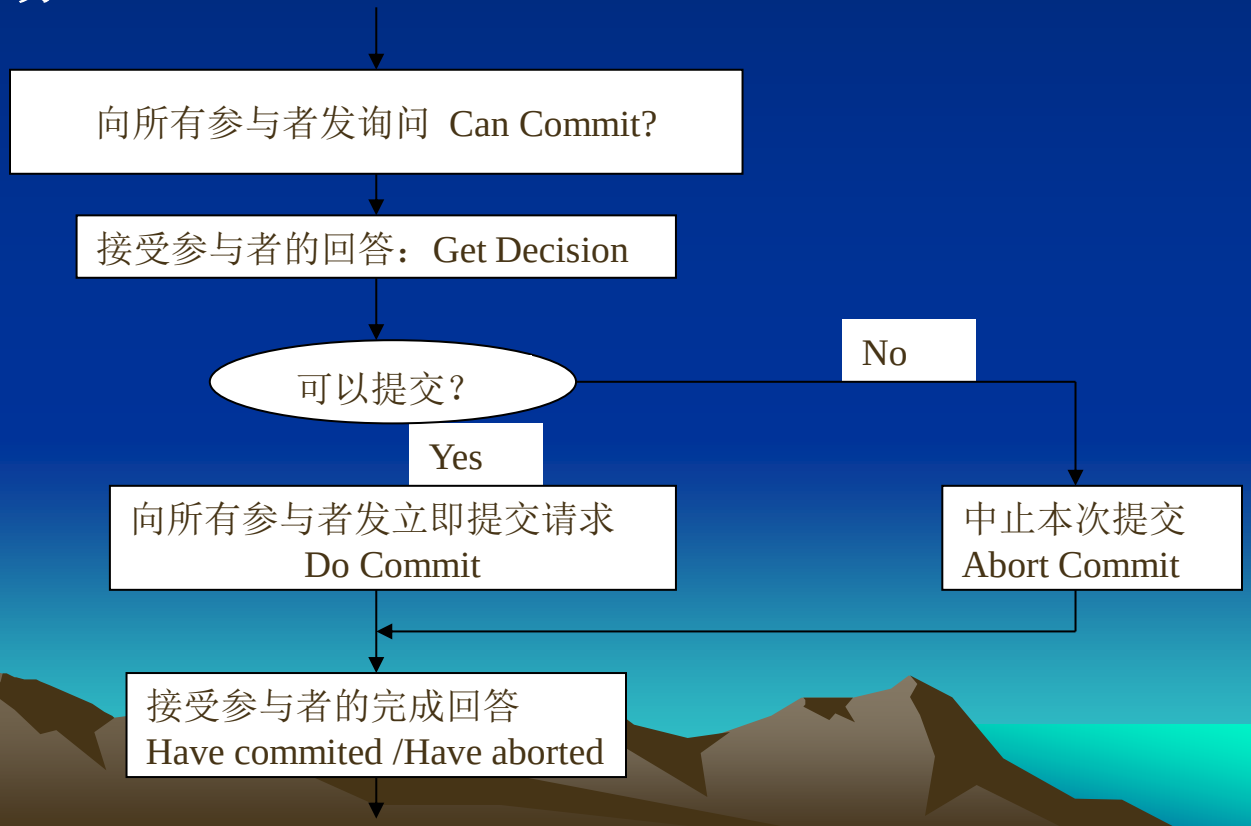
分布式提交—单阶段提交

- 分布式提交：具有原子性：要使一个操作被进程组中每一个进程都执行或都不执行。通常使用协调者。
- 协调者向事务所有的参与者发出提交或者中止请求，并不断重复该请求，直到所有参与者报告它们执行了该请求。
- 问题：如果遇到某一问题，导致某一参与者无法完成提交，本协议就无法实现。

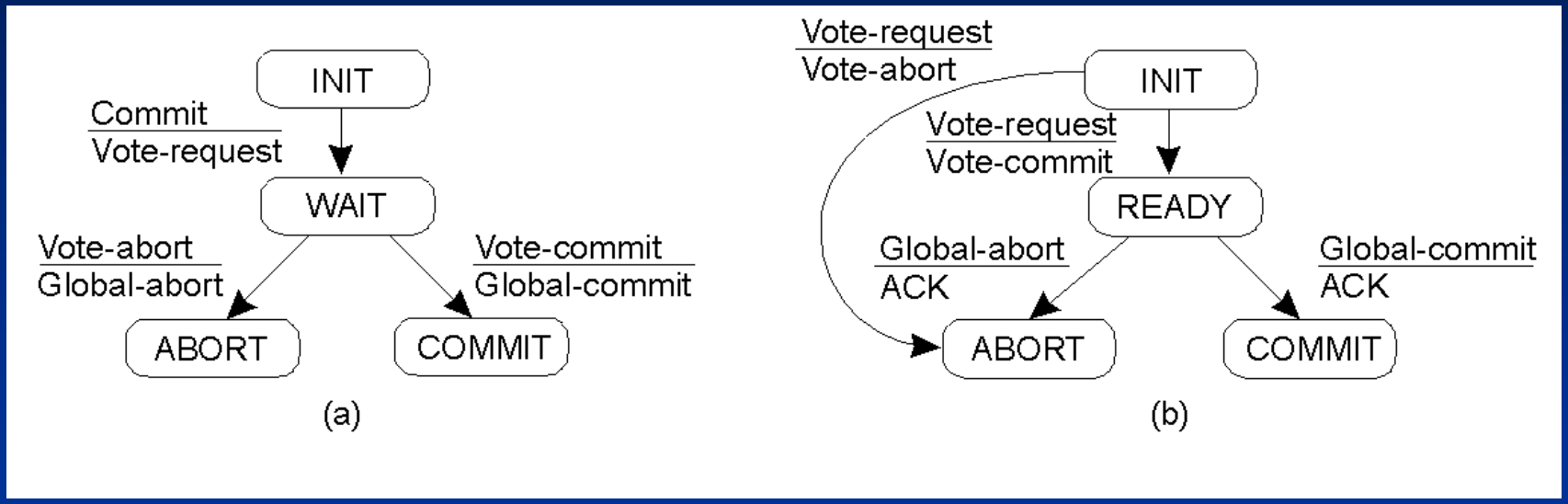


分布式提交—两阶段提交 (1)

简单、实用、可靠，成为事实上的工业标准。
在两段提交协议中，将提交分成两个阶段，
•第一阶段（表决阶段），事务的协调者询问各个参与者是否可以提交，此时，各个参与者将回答消息发给协调者。协调者根据收到的消息，看是否可以真正提交。
•第二阶段（完成阶段），如果可以提交，则通知各参与者立即执行提交，否则，通知它们中止此事务。



两阶段提交 (2)



a) 2PC中的协调者的有限状态机
b) 2PC中的参与者的有限状态机

参与者一旦投票，则失去自主能力，必须等待协调者的最终决定，可能造成阻塞

可能的阻塞状态：

- 参与者在INIT状态等待协调者的VOTE_REQUEST消息
- 协调者在WAIT状态等待来自每个参与者的表决
- 参与者在READY状态等待协调者发送的全局表决消息

两阶段提交 (3)

State of Q	Action by P
COMMIT	转换到 COMMIT
ABORT	转换到 ABORT
INIT	转换到 ABORT
READY	与其他参与者联系

参与者 **P** 在 **READY** 状态下与另一个参与者 ***Q*** 联系时采取的行动

当所有运行的参与者都处于 **READY** 状态时，尽管都已同意提交，但可能有崩溃的参与者（不一定同意提交），因而无法做出决定（即使选举出新的协调者），只能等待原协调者恢复

两阶段提交 (4)

```
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;
}
```

2PC中协调者采取的操作

两阶段提交 (5)

```
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;
}
```

2PC中参与者采取的操作

两阶段提交 (6)

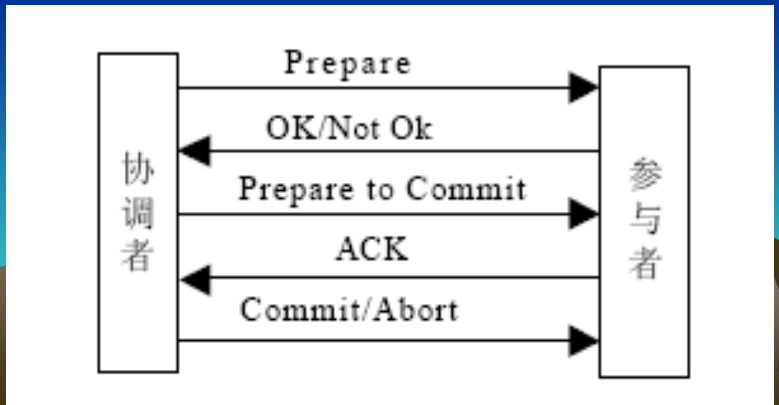
```
actions for handling decision requests: /* executed by separate
thread */

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 */
```

处理（来自其他 **READY** 进程）决定请求的步骤

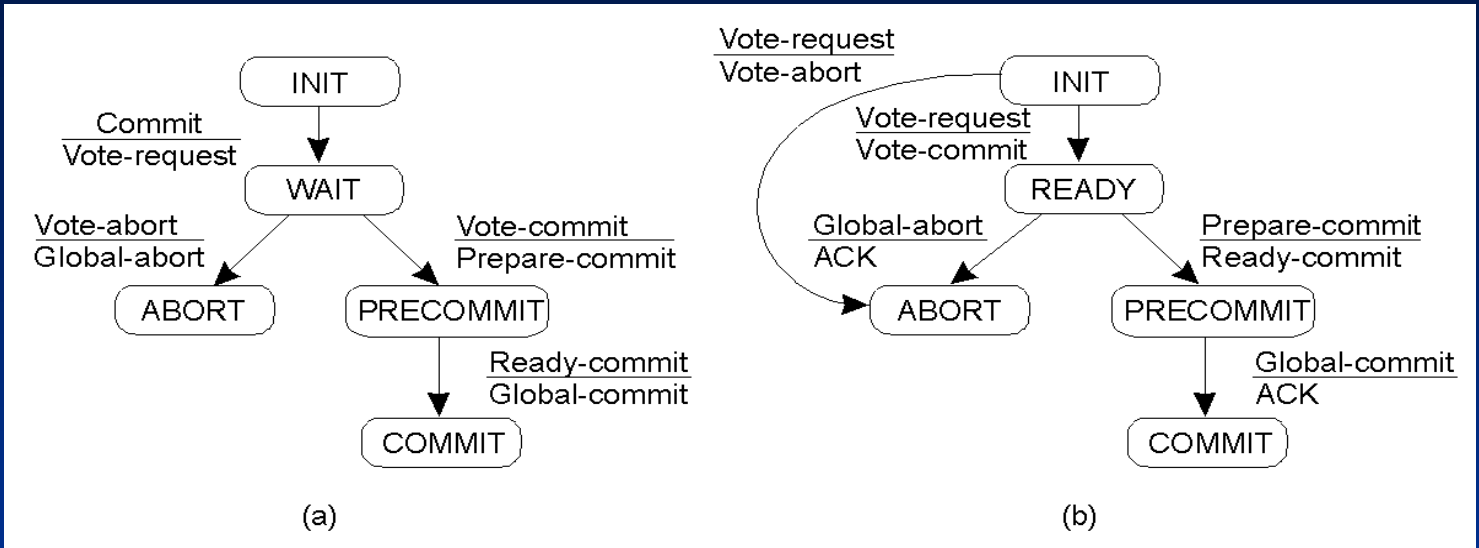
三阶段提交

三阶段提交

- 阶段1
  - 同两阶段方式
- 阶段2
  - 收到有一个 abort T ,则 abort T
  - 收到所有 ready T ,则 precommit T
  - 节点 precommit T之后，写 Log，发出 acknowledge T
- 阶段3
  - 收到所有ack，则 commit T
  - 节点 commit 后，发出 ack T
  - 收到所有 ack T后，complete T



三阶段提交



a)3PC中的协调者的有限状态机
b)3PC中的参与者的有限状态机

- 没有一个直接转换到COMMIT和ABORT的状态
- 不存在这样的状态：它不能做出最后决定，而且可以直接转换到COMMIT状态

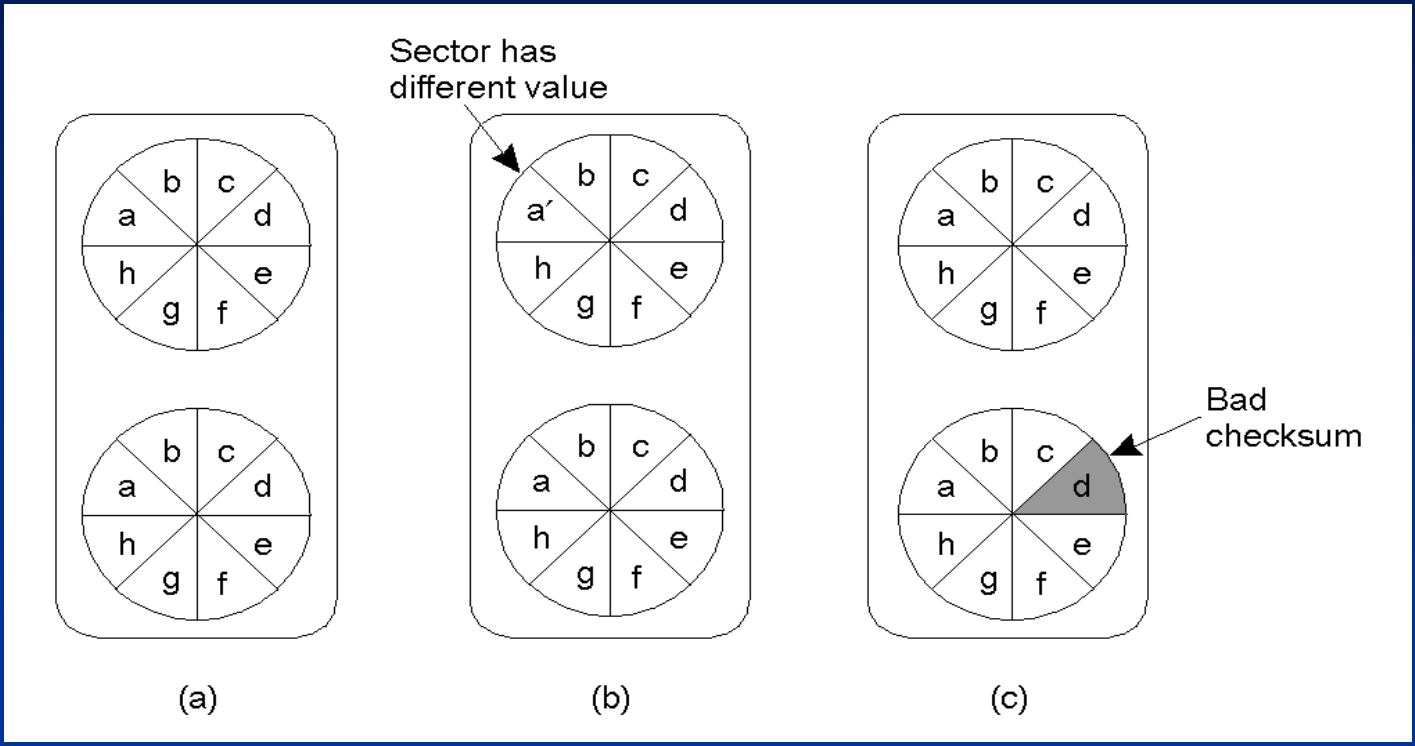
可能的阻塞状态：

- 参与者在INIT状态等待协调者的VOTE_REQUEST消息
- 协调者在WAIT状态等待来自每个参与者的表决
- 协调者在PRECOMMIT状态等待来自每个参与者的表决: 超时时，认为参与者已崩溃，但已投票参与事务，所以可以多播提交
- 参与者在READY状态等待协调者发送的全局消息：超时时，认为协调者已崩溃，联系其他参与者的状态 ABORT-〉 ABORT； PRECOMMIT-〉PRECOMMIT；都处于READY-〉事务可以被安全中止（因为出故障的参与者还未作出提交的决定），选举出新的协调者继续执行
- 参与者在PRECOMMIT状态等待协调者发送的全局消息：超时时，认为协调者已崩溃，联系其他参与者的状态 COMMIT-〉COMMIT； 都处于PRECOMMIT-〉事务安全提交

恢复

- 恢复：用正确的状态代替错误的状态
- 回退恢复（backward recovery）：
  - 从当前错误的状态回到先前正确的状态
  - 需要记录系统的状态（检查点）
  - 通用的技术，不依赖特定的系统或进程
- 向前恢复（forward recovery）：
  - 尝试从可以继续进行的某点开始把系统带入一个正确的状态
  - 需要预先知道会发生什么错误
- 回退恢复技术存在的问题
  - 开销较大
  - 还可能发生相同的错误
  - 有些状态无法回退

恢 复--稳定存储



要恢复到先前的状态，需要存储所需的信息

存储分类：RAM、磁盘和稳定存储

a. 稳定存储（一对磁盘实现）

b. 在驱动器1更新后崩溃

c. 坏点

小 结

- 容错性简介
- 进程恢复
- 可靠的C-S通信
- 可靠的组通信
- 分布式提交
- 恢复

习 题

- 下面情况下，最少一次语义合适还是最多一次语义合适？
  - 从文件服务器读写文件
  - 编译一个程序
  - 远程银行
- 下图中，**FIFO**与全序结合情况下。可能的消息发送顺序?

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	