# 3.2 一组希望达成一致的智能体

# 01 首领与追随者

假设有一个数据库，可以用来存储一些非常重要的数据（比如银行需要存储的财务信息）。由于这些数据决不能丢失，因此你会尝试在很多其他的机器上复制存储备份这些数据，这些机器分别是$m_0$，$m_1$...$m_n$。

为了搭建这个系统，可以采用最直观的方式，即：**将其中一台机器指定为“首领（leader）”（假设由$m_0$来担任leader这个角色），其他的机器为“追随者（followers）”。**

如此一来，任何时候需要处理一笔新的客户交易时（如从Bob账户中扣除$10），leader机器将发出指令要求所有followers机器去执行这笔交易。首先，如果leader机器m0发出指令要求m3机器从Bob的账户中扣除$10，但这条消息发送失败了，并没有传递给m3。

**新的问题：不同机器上的数据可能存储不一致?**

# 02 信息的发出与确认

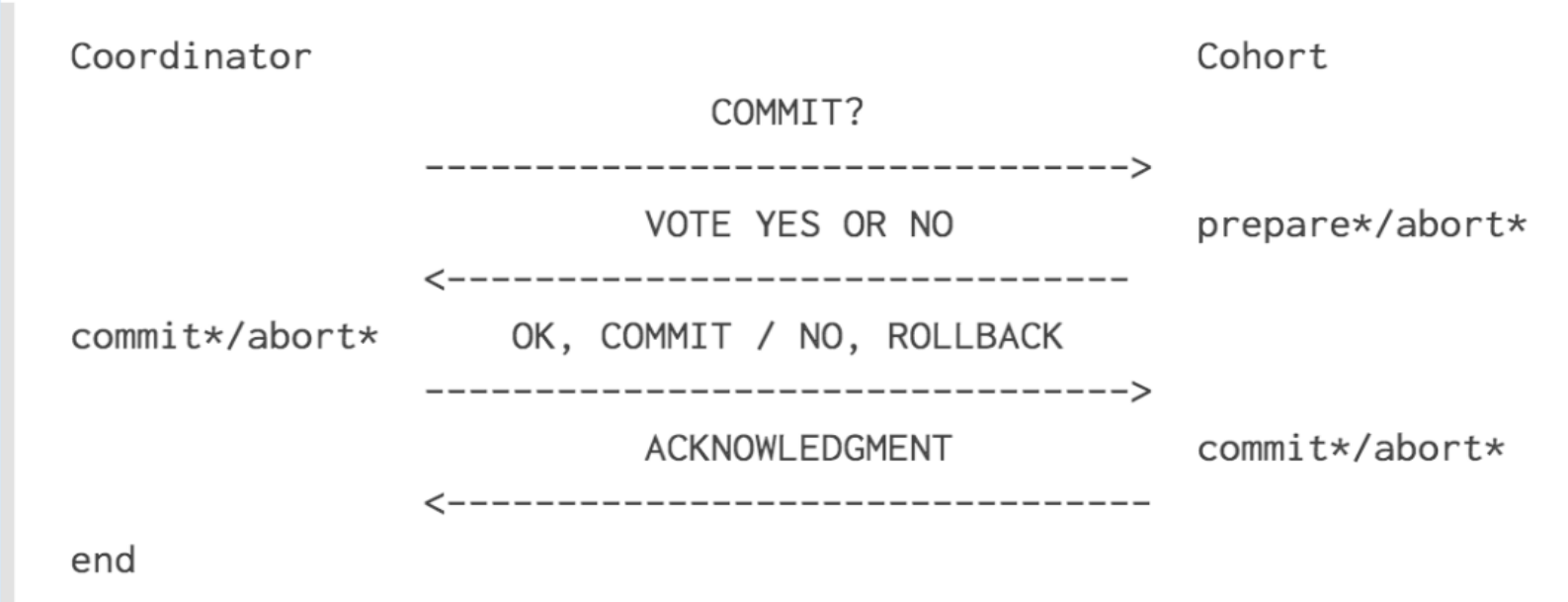
- 上述问题的一个基本解决方案是，确保m3机器已经收到消息。这就会涉及到了一个**原子提交协议，即二阶段提交（2PC，Two-Phase-Commit）。**

- 想象一下：机器m0并不会盲目地从客户端接收消息并将消息传递给其他机器，m0还会确保其他复制机在接收到消息之后会向m0确认自己已收到了该条消息。但是如果其中的某机器（假设是m6）出故障了，而其他的机器已经从Bob的账户中扣除了$10，这该怎么办呢？别担心，回滚（rollback）就可以了！

# 03 信息的发出与确认

在两阶段提交协议中，系统一般包含两类机器（或节点）：

- 一类为**协调者（coordinator）**，通常一个系统中只有一个；
- 另一类为**事务参与者（cohorts）**，一般包含多个，在数据存储系统中可以理解为多个数据副本。



# 04 信息的发出与确认

两阶段提交协议由两个阶段组成。在正常的执行下，这两个阶段的执行过程如下：

**阶段1：请求阶段（commit-request phase，或称表决阶段，voting phase）**

- 在请求阶段，协调者将通知事务参与者准备提交或取消事务，然后进入表决过程。在表决过程中，参与者将告知协调者自己的决策：同意（事务参与者本地作业执行成功）或取消（事务参与者本地作业执行故障）。

**阶段2：提交阶段（commit phase）**

- 在该阶段，协调者将基于第一个阶段的投票结果进行决策：提交或取消。当且仅当所有的参与者同意提交事务时，协调者才通知所有的参与者提交事务，否则协调者将通知所有的参与者取消事务。参与者在接收到协调者发来的消息后将执行响应的操作。

# 05 引入共识

**如果leader机器出现故障怎么办?**

首先，为了应对m0可能出现故障，在系统中引入了另一个leader机器（假设是m1）。但是，**存在两个leader机器会导致协调问题的出现**。比如说，假设Bob发送了两笔交易，其中一笔T1是“从Bob账户中扣除$10”，另一笔T2是“从Bob账户中扣除$20”，而Bob账户中恰好总共只有$20。

机器m0也许会将T1发布到网络中，而机器m1可能将T2发布出去。系统中的其他followers机器将根据这些操作的顺序来验证这两笔交易，并会出现**不一致的状态（inconsistent states）**。验证后，Bob账户里面还剩$10还是$0?

# 06 引入共识

**如果一些followers机器（节点）出现故障，那该怎么办呢?**

如果等待这些节点重新上线，尤其是需要通过人为操作才能恢复某些服务器的时候，可能会出现一些效率问题。而如果选择忽视一些出故障的机器（节点），那应该忽视几台机器呢?

更关键的是，假设某些机器恢复了，但它们的硬盘驱动器却因此而遭到了破坏（数据遭到损坏），这又该如何是好呢？虽然这些机器依旧会运行协议，但它们可能无法成功地将数据写入硬盘中。

# 07 引入共识

**最开始的那个问题：**

如何才能让一组机器就某条消息达成一致呢？

由Leslie Lamport发明的**Paxos是一种应对计算崩溃故障（crash fault）问题的共识协议。**

Paxos主要内容是：只要大多数的机器（节点）接受某个提议（proposal），这就可以保证系统中所有的机器都不会出现不一致的状态。