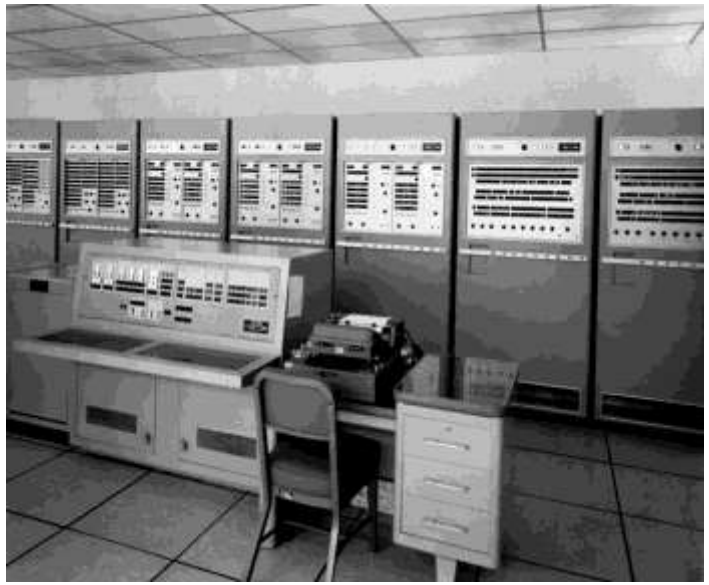
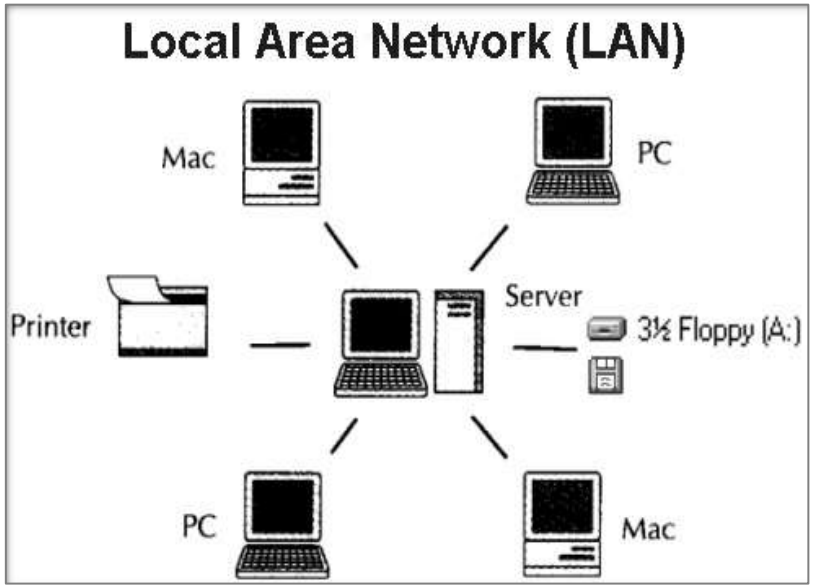
分布式系统安全

清华大学

分布式系统的起源

- **并行计算**：Burroughs Corporation在1962年推出了D825，这是一种四处理器计算机，可以访问多达16个共享的内存模块
- **ARPANET**：互联网的前身之一，于20世纪60年代末被推出，分布式系统开始拓展到主机这一粒度
- **LAN**：1970年，以太网等局域网技术的出现，标识着分布式系统开始广泛部署





分布式系统飞速发展

- 随着互联网技术的不断演进与用户数量的“爆炸式”增长，导致用户需求不断增加
- 基于单一节点的C/S架构所构建的服务无法承受如今的用户规模，分布式系统规模也在不断扩张

日点击量峰值超1500亿次 12306是如何应对下来的?

2018-02-08 22:51

一年一度的春运如期而至，中国又将迎来“人类最大规模的周期性迁徙”。每年的春运，对售票系统12306也是一场艰难的大考。

2018年在线流媒体的订阅用户人数达到6.13亿，全球媒体巨头纷纷入局视频流媒体[图]

2019年09月23日 14:19:25 字号: T | T

根据数据，在线流媒体2018年的订阅用户数达到6.13亿，同比增长27%（增量1.31亿），成功超越有线电视5.56亿订阅用户，用户数4年翻4倍。

分布式系统是什么?

如今所说的分布式系统通常是由分布在不同地理位置的系统组件所构成，这些系统组件通过网络传递消息完成动作协调，从而相互协作完成共同的任务

- 分布式系统的组成:
  - 交互网络 — 实现协作的前提，系统组件基于交互网络实现消息交互
  - 分布式算法 —协调分布式组建之间的交互，确保系统稳定运行
  - 信任机制 — 解决不同系统组件之间的信任问题，实现可信协作

分布式系统安全问题的根源

安全问题可能会造成系统分区，或是危害系统的一致性、有效性和隐私性

- 从分布式系统的组成来看，其安全问题的根源体现在交互网络、分布式算法和信任三个方面：
  - 攻击者可以攻击交互网络，监听交互过程中的隐私信息或造成网络分区
  - 协作过程中可能会出现冲突，故障等问题，从而影响一致性和可用性
  - 系统内部可能存在恶意组件，系统和客户端交互时也存在信任问题

本章的内容组织

第一节
建立安全、稳定的交互网络

- 交互网络的组成
- 交互信道安全
- 弹性覆盖网络
- 防御日蚀攻击

了解交互网络的组成以及如何构建安全的交互网络

第二节
分布式算法

- 协作过程中的安全隐患
- 时钟同步算法
- 并发问题
- 故障容错的一致性算法

挖掘分布式系统各组件之间如何进行协同

第三节
信任问题

- 访问控制
- 信任评价模型
- 拜占庭容错

了解分布式系统中的信任问题及解决方案

为分布式系统各组件之间的协同提供交互基础

为分布式系统添加信任面，实现可信协同

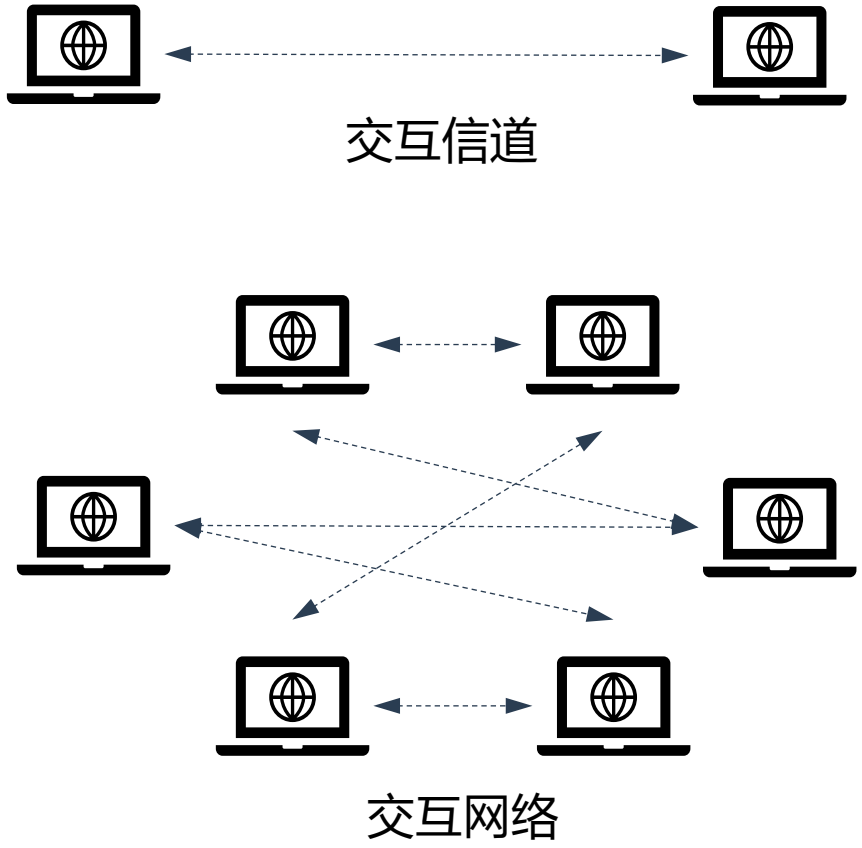
第1节 建立安全、稳定的交互网络

- 交互网络的组成
- 交互信道安全
- 弹性覆盖网络
- 防御日蚀攻击

交互网络的组成

在分布式系统中，节点之间可以通过交互信道实现消息传递；分布式节点选择邻居节点并建立交互信道，从而构成一个交互网络

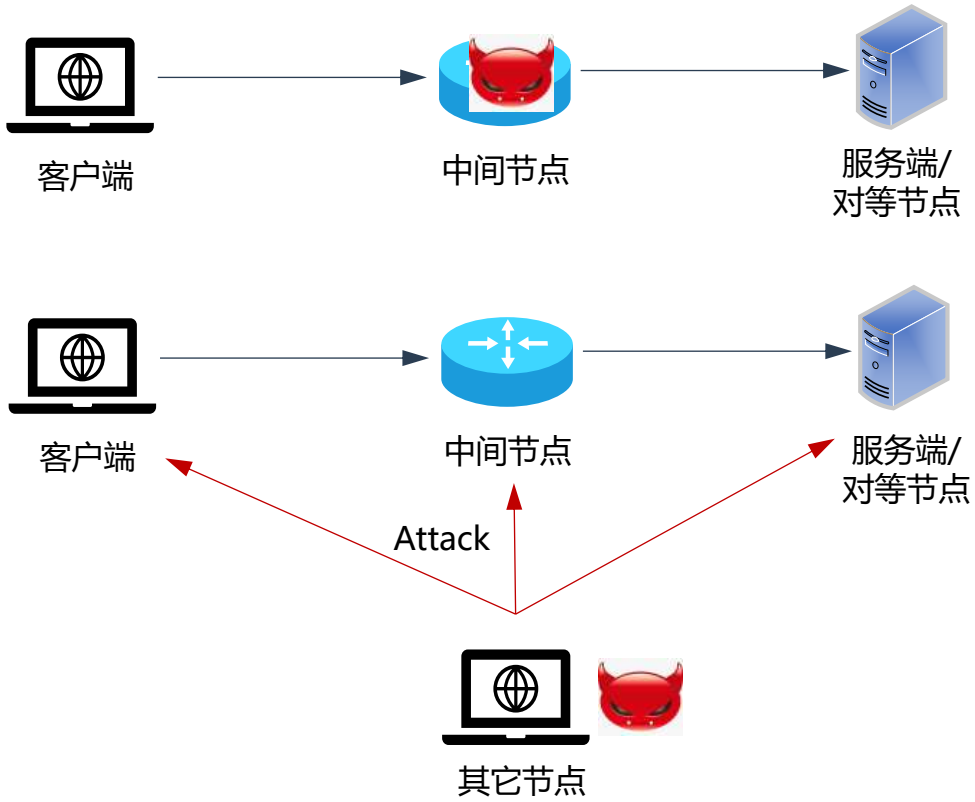
- **交互信道：**两个主机通过交互信道实现host-to-host的通信，常见的交互信道有UDP、TCP和TLS等
- **邻居选择机制：**每个节点依靠邻居选择机制选取系统中的部分或全部节点建立交互信道，从而建立邻居关系，节点可通过邻居节点与其它非邻居节点交互



交互信道中的安全隐患

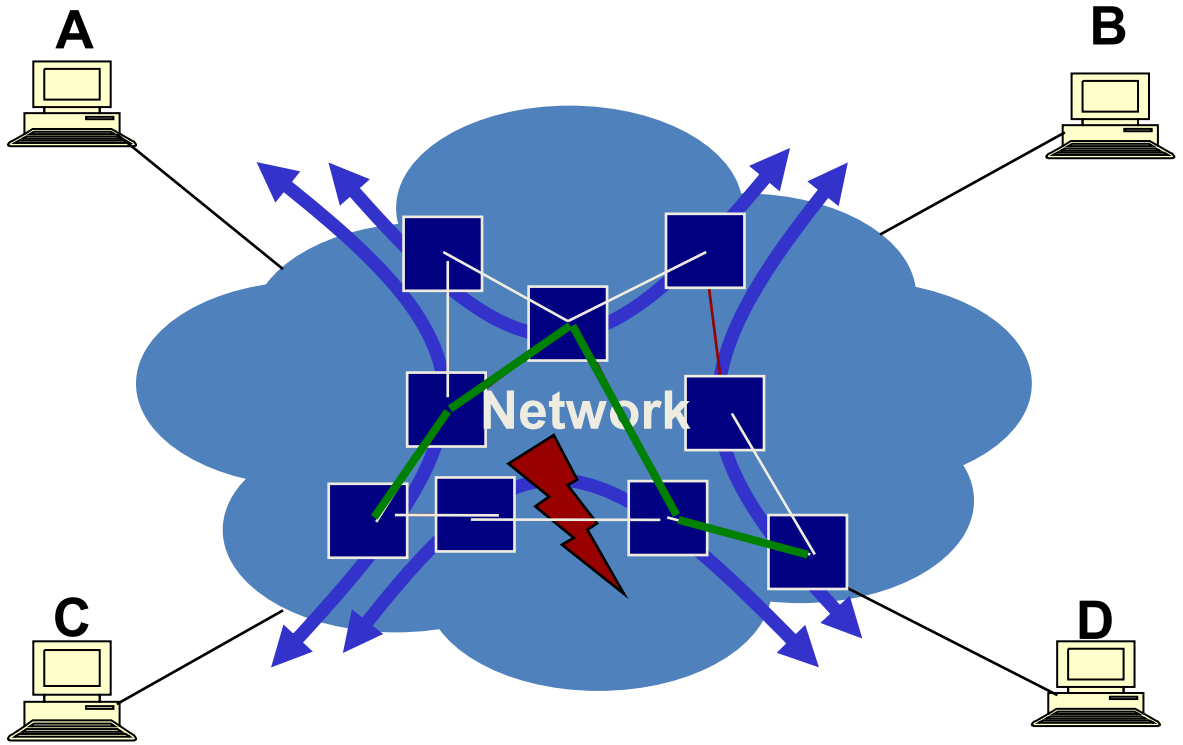
- 根据网络体系结构的不同层面进行分类
  - 网络层面：网络中传输的数据包可能出现**丢失**、**重复**和**泄漏**
  - 应用层面：节点间的交易请求可能出现**泄漏**、**重放**、**篡改和伪造**

- 根据攻击者的位置分类
  - **中间转发节点：**直接参与转发数据包，攻击能力强，可以监听、篡改、伪造、重放、延迟甚至丢弃转发信息
  - **其它节点：**攻击能力较弱，但能够通过DDoS、中间人等攻击影响交互信道



弹性覆盖网络RON

- 弹性覆盖网络是一种分布式覆盖网络体系结构，它可以使分布式应用检测到**路径的失效和周期性的性能降低现象**并能够迅速恢复
  - 恢复时间少于20秒
  - **对比：**在目前的Internet中，BGP恢复时间为几分钟

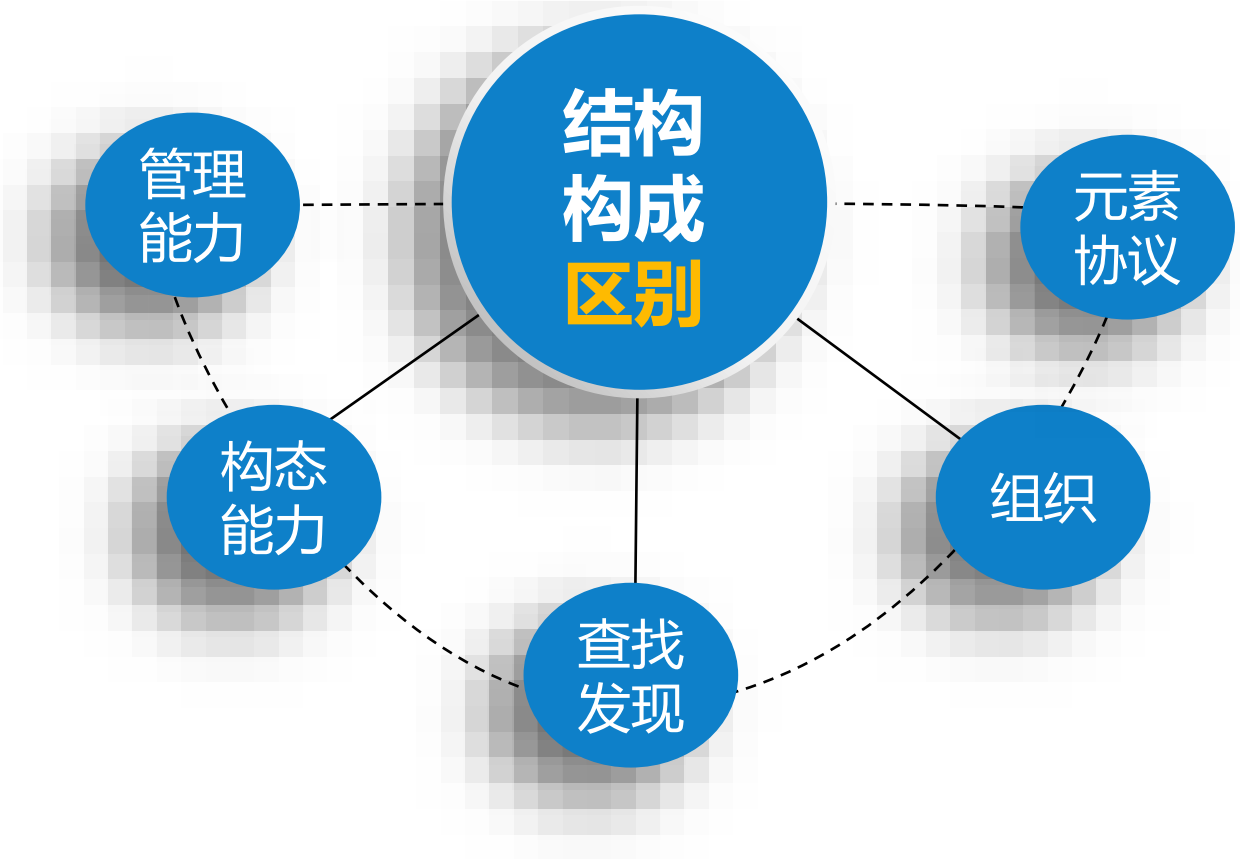


Packet switching and route around failures

- RON节点监控它们之间的路径的质量并根据这些信息决定
  - 直接利用Internet转发分组
  - 通过其他RON节点转发分组

P2P vs C/S

P2P (Peer-to-Peer)又称对等网络技术，也是建立在Internet之上的分布式Overlay网络，能够不依赖中心节点而只依靠对等协作实现资源发现与共享



相同

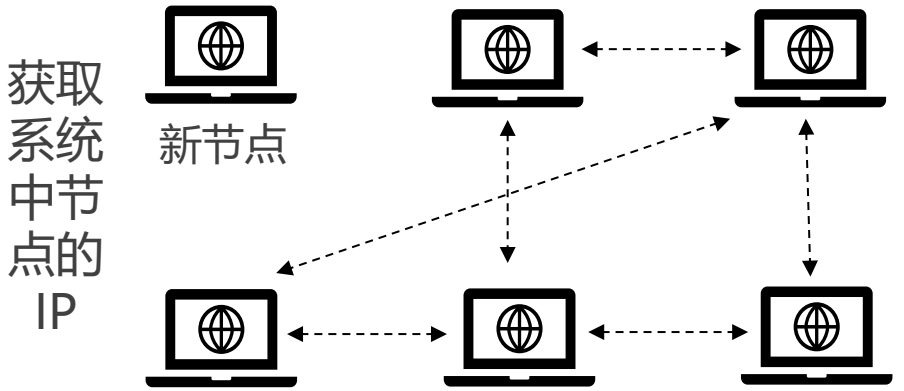
- 都能运行在不同（Internet / Intranet）平台
- 都能服务传统或新的应用：eBusiness eServices …

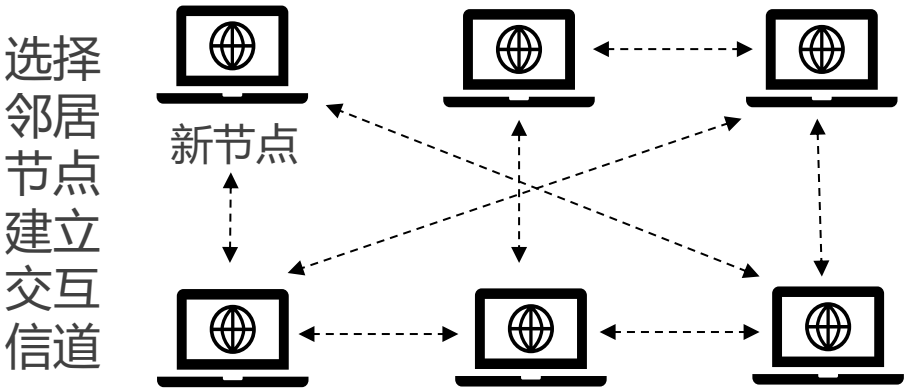
P2P网络的建立流程

P2P网络的建立主要包含网络节点发现和邻居节点维护两部分

- 网络节点发现
  - 新加入的节点获取系统中其它节点的IP地址，并存储在本地的网络信息表中

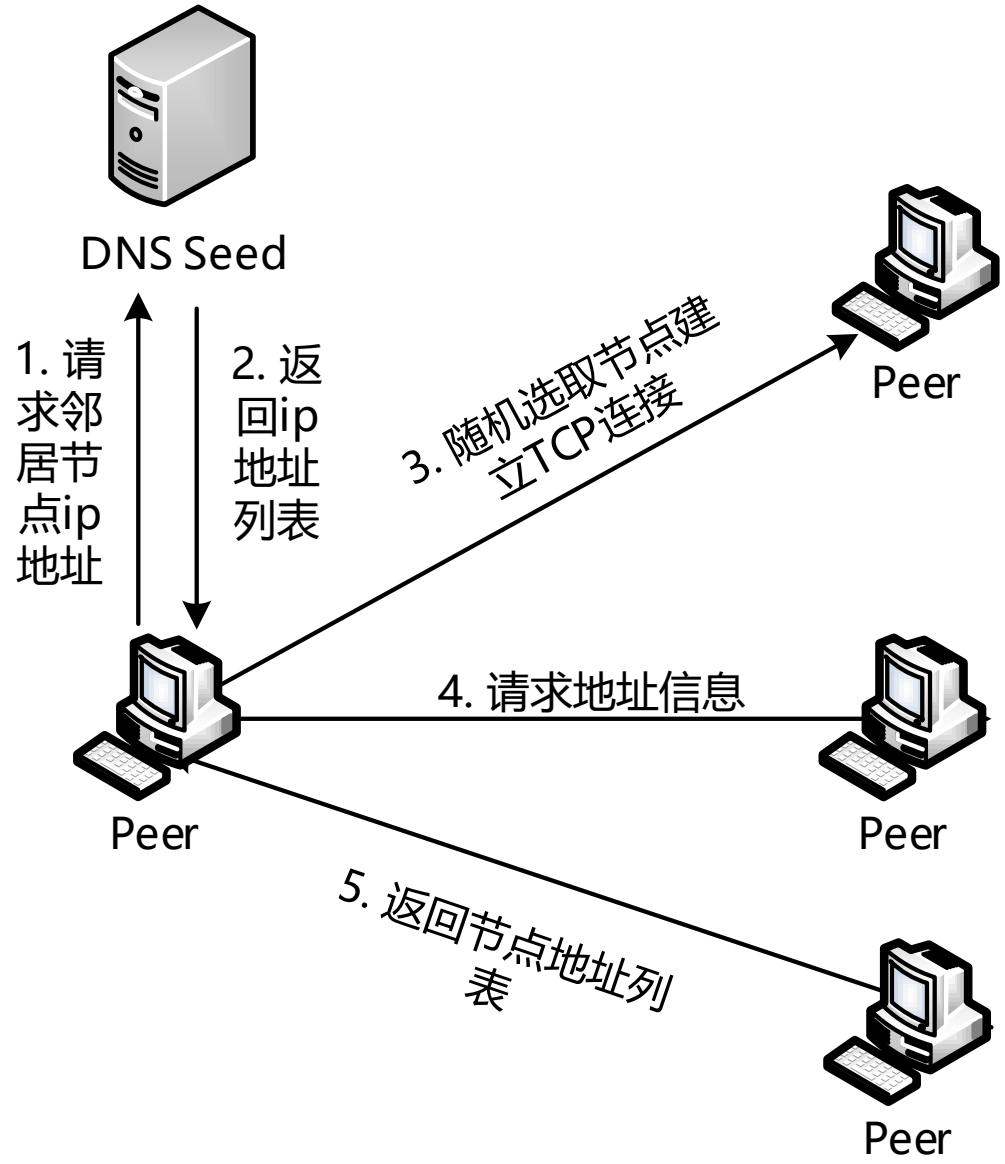
- 邻居节点维护
  - 节点从本地的网络信息表中选择IP地址，并请求建立邻居关系





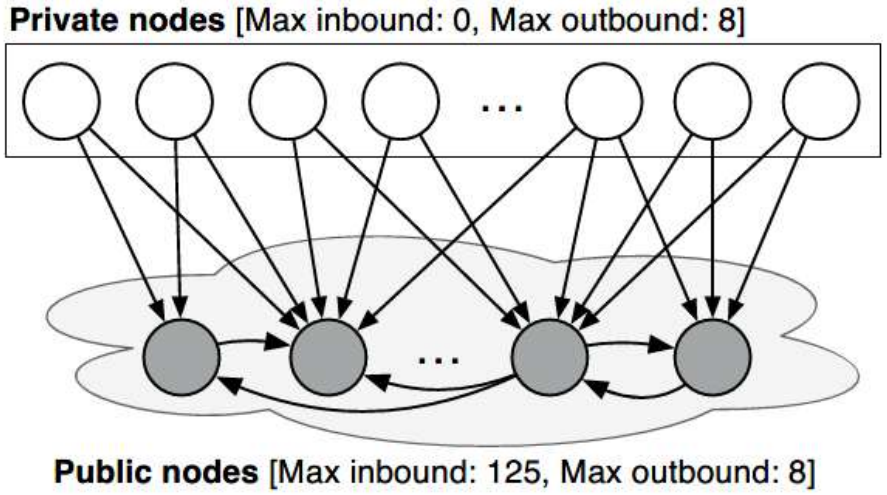
网络节点发现

- 程序硬编码
  - P2P客户端程序中写入权威节点IP，用于客户端建立初始连接
- DNS Seed
  - 向Seed服务器请求节点IP地址
- 向邻居请求
  - 已存在邻居后，和邻居节点定期交换所发现的新IP地址
- 节点查询
  - 针对特定的节点id，查询其IP地址，主要在有结构化的P2P网络中存在

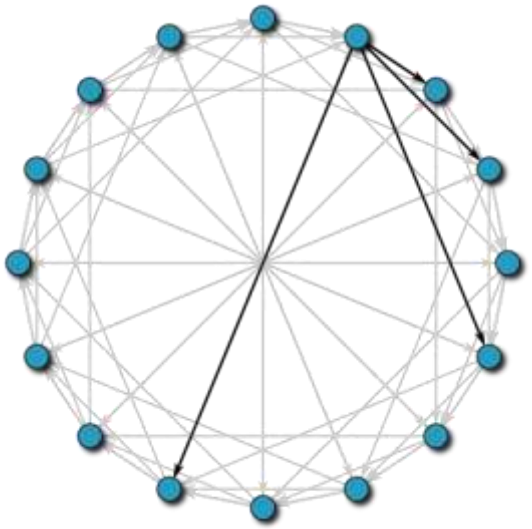


邻居节点维护

- 无结构化
  - 从发现的IP地址中，随机选择邻居
  - 通常限制节点的入向和出向连接数，入向指被请求建立的邻居关系，出向指主动请求建立的邻居关系



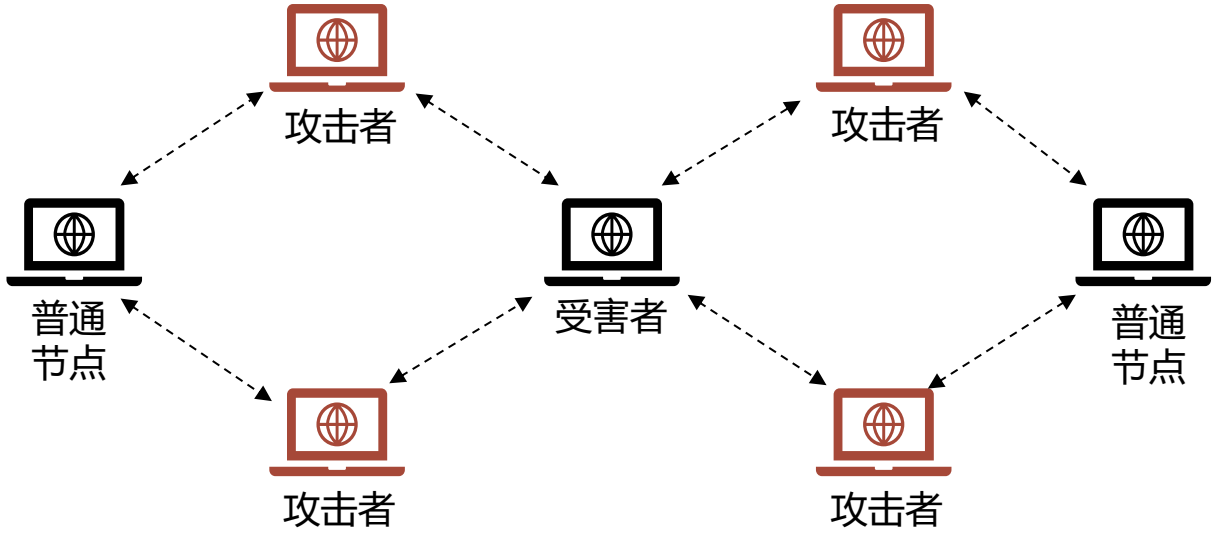
- 有结构化
  - 每个节点都有指定的id，并存在相应的公私钥用于id的认证
  - 节点基于id之间的距离按规定确定邻居节点



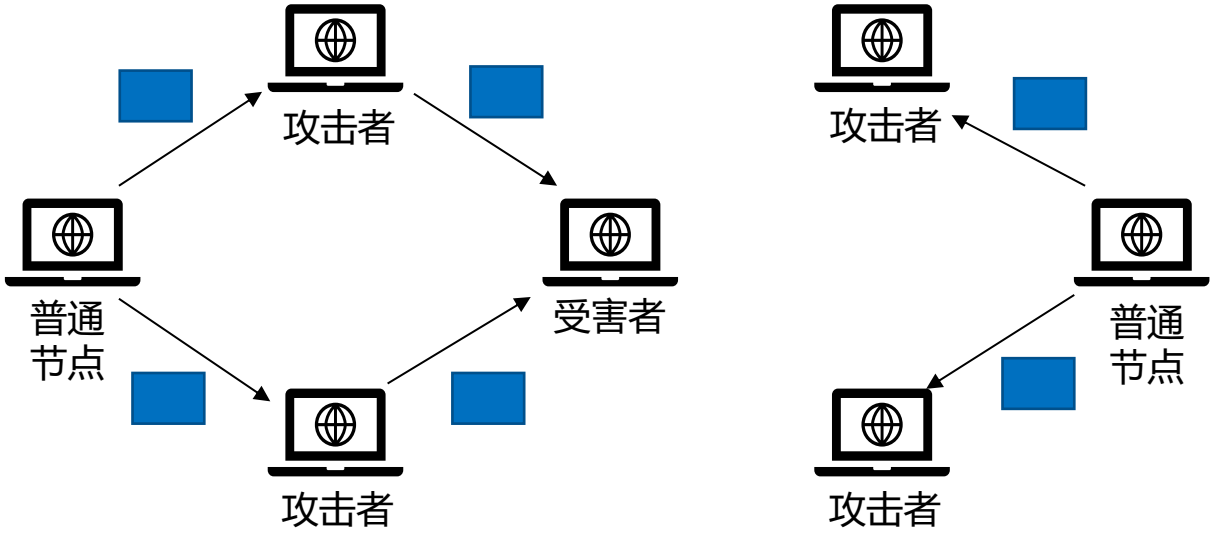
- 混合模式：有结构化P2P网络中，允许节点随机选择部分邻居节点

日蚀攻击

攻击目标：针对网络节点发现以及邻居节点维护两个过程进行攻击，使得受害节点的所有邻居节点都是由攻击者所控制的恶意节点



危害：攻击者可以选择性的转发对攻击者有利的消息给受害者，从而配合其它网络攻击；攻击者可以完全隔离受害者与网络中其它节点的信息交互



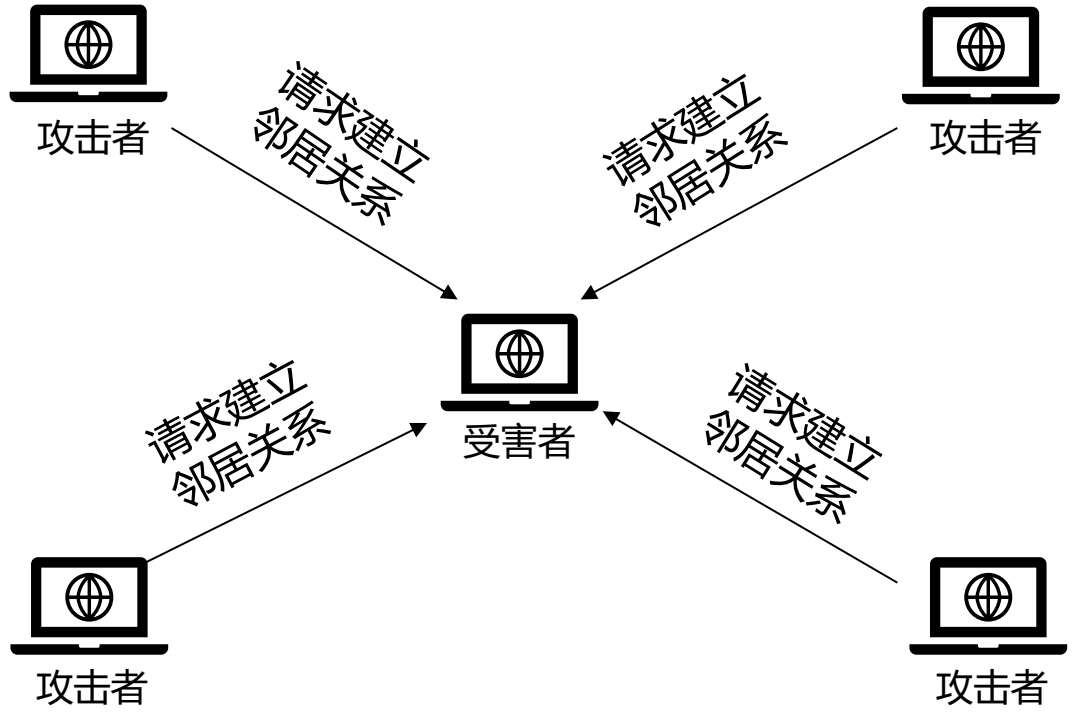
挤占入向连接

主要思路

- 攻击者控制大量节点主动向受害者请求建立邻居关系
- 若受害者已经和其它节点建立邻居关系，则可配合DDoS等其它网络攻击触发受害节点的重启，然后迅速挤占其入向连接

解决思路：

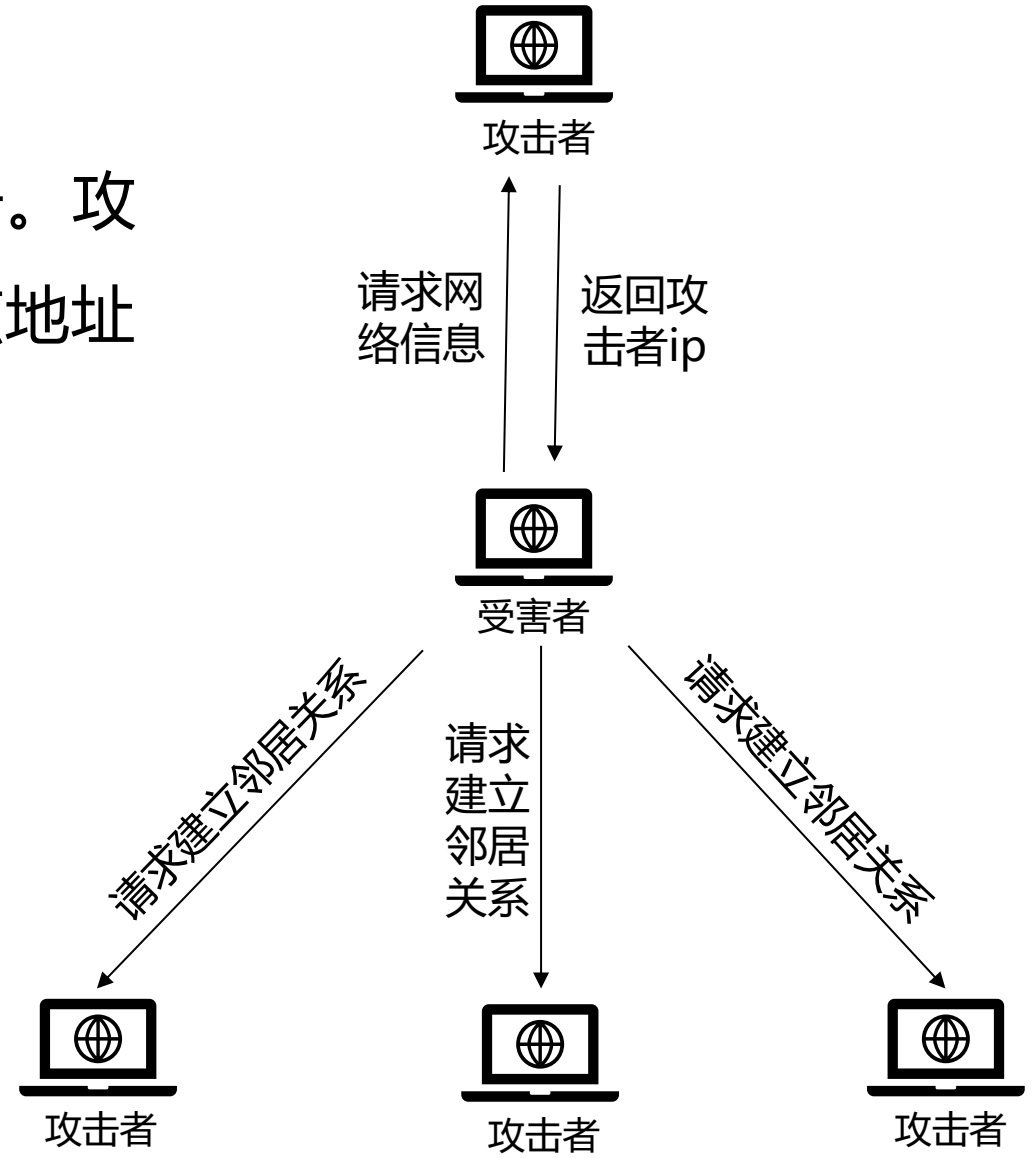
周期性更换邻居节点



挤占出向连接

主要思路：

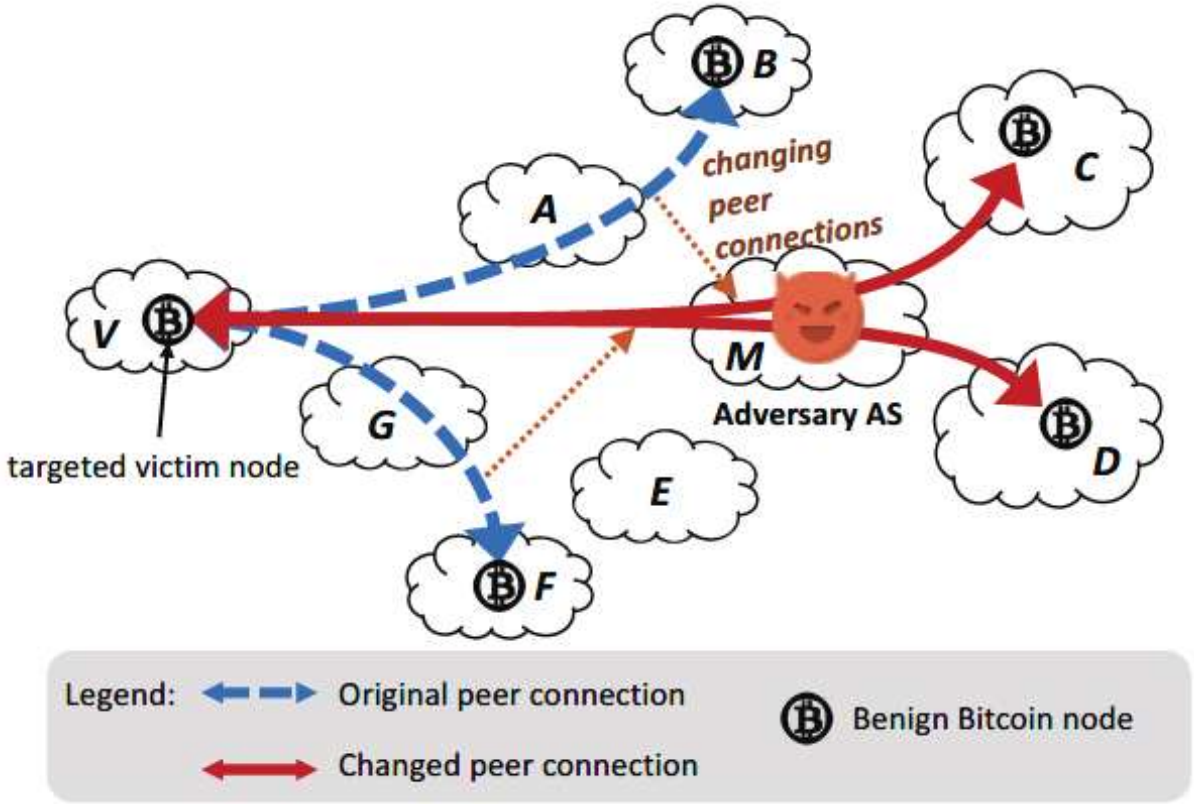
针对受害节点发现网络节点的过程进行攻击。攻击者向受害者发送大量由自己所控制的节点地址信息，等待受害者主动请求建立连接



解决思路：

- **增加网络信息表项的大小**
- **增加出向连接个数**
- **选择少量不易受攻击的可信权威节点作为邻居节点**

配合路由劫持攻击



主要思路：

- 向受害节点本地存储的节点地址列表注入大量影子IP（与受害节点间的通讯链路经过攻击者所控制的AS）
- 使受害节点与影子IP建立邻居关系，从而劫持受害者的所有邻居链路

解决思路：

邻居选择时将AS拓扑考虑进去

注. 图片来自来源：
Tran M, Choi I, Moon G J, et al. A stealthier partitioning attack against bitcoin peer-to-peer network[C]//2020 IEEE Symposium on Security and Privacy (S&P). IEEE, 2020: 894-909.

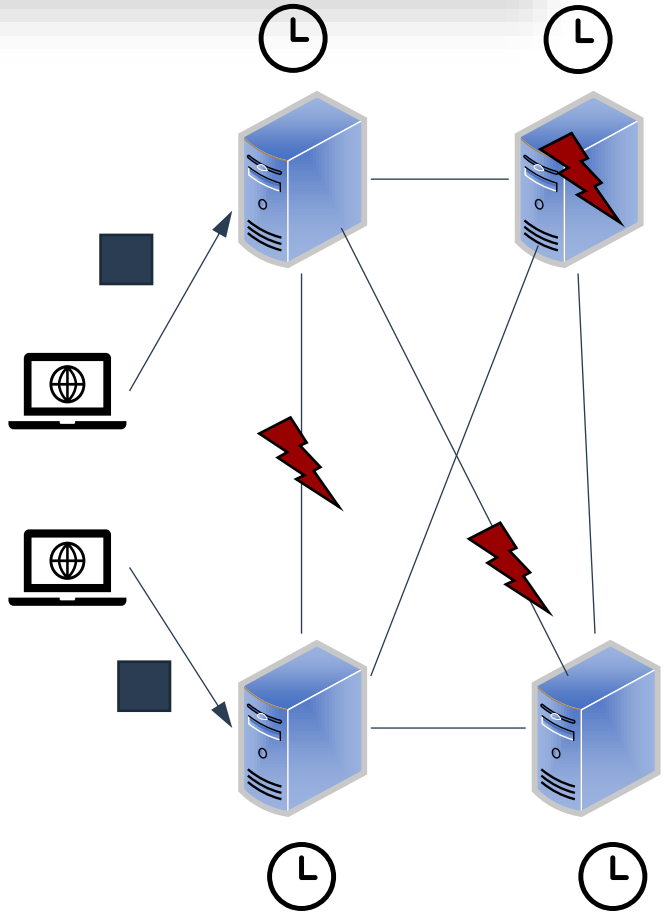
第2节 分布式算法

- 协作过程中的安全隐患
- 时钟同步算法
- 并发问题
- 故障容错的一致性算法

协作过程中的安全问题

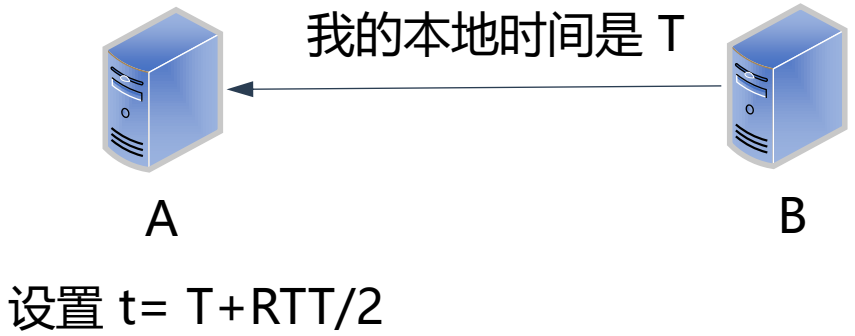
分布式算法的目的是协调分布式组件之间的交互和动作，从而确保它们能够稳定协作实现一个共同的任务

- 缺乏全局时钟：各节点本地时钟可能不同步
  - 物理时钟：基于时钟同步机制完成物理时钟同步
  - 逻辑时钟：基于共识对事件发生的顺序达成一致
- 并发：同一时刻，系统中的不同组件可能面临来自多个客户端的大量并发请求
- 故障：系统组件在协作过程中可能出现故障，从而导致信息的短暂丢失



时钟同步算法

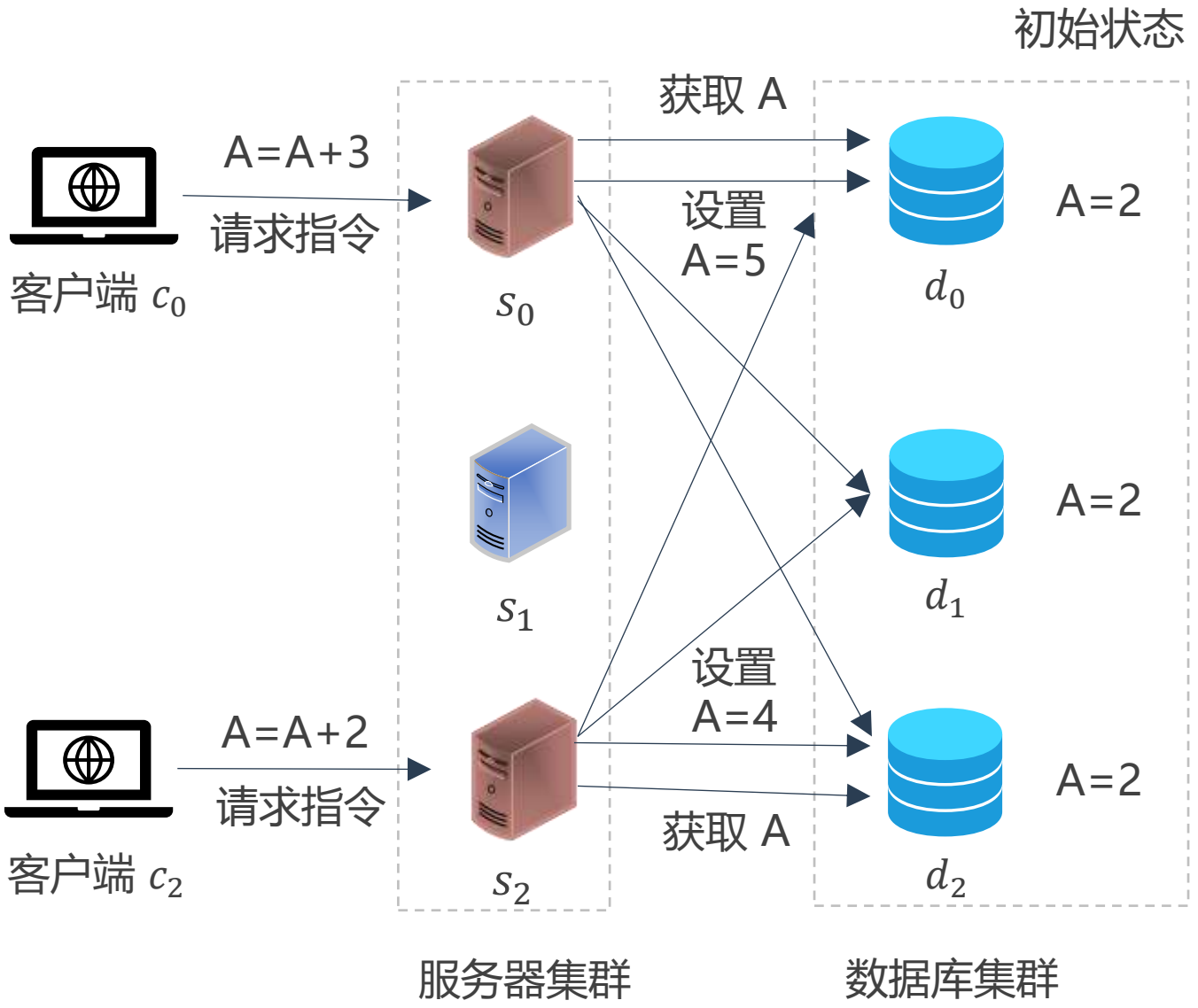
- **主要思想：**若节点A想与节点B完成时钟同步，则请求节点B发送本地时间T；然后预测A与B的网络时延RTT；之后将时间设置为T+RTT/2；RTT预测的越精确，时钟同步的误差越小



- **主要方式**
  - Cristian算法：使用一个时间服务器，其它节点基于该服务器完成时钟同步，主要用于企业网内部
  - Berkeley算法：主要用于分布式集群系统中各节点之间时钟同步，主要用于企业网内部
  - NTP协议：主要运用于互联网，确保网络节点之间能够时钟同步

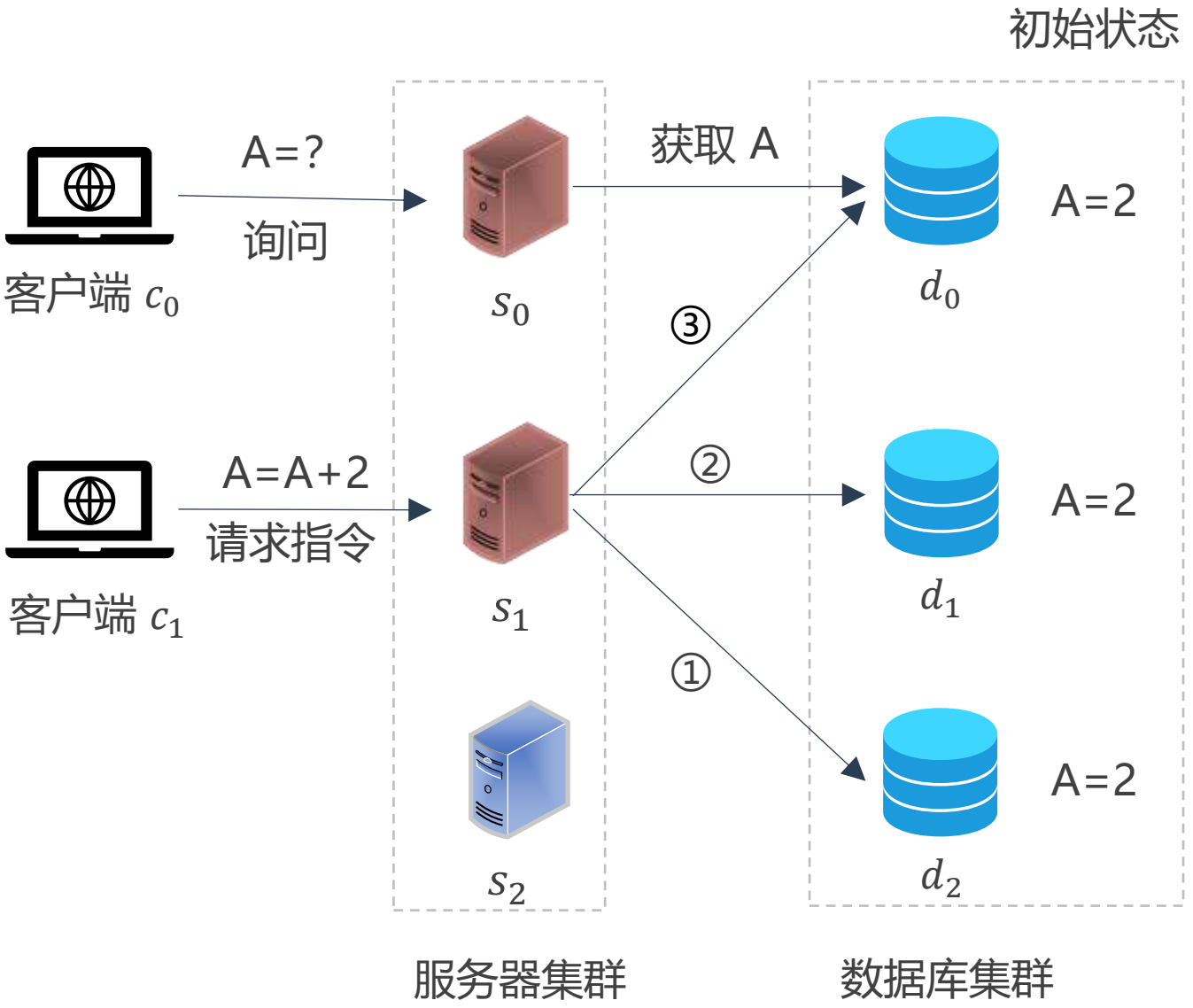
并发问题：破坏一致性

- 假设：状态A在三个数据库冗余存储，不存在并发控制措施
- 执行步骤
  - c_0向s_0请求将A的值加3
  - c_2向s_2请求将A的值加2
  - s_0和s_2分别向d_0和d_2读取A的值并完成计算
  - s_0和s_2依次向三个数据库更新A的值
- 由于指令的并发执行，可能导致三个数据库A的值不一致



并发问题：指令之间相互影响

- 假设：状态A在三个数据库冗余存储，不存在并发控制措施
- 执行步骤
  - c_0向s_0请求A的值
  - c_1向s_1请求将A的值加2
  - s_1依次将三个数据库中A的值修改为4
  - s_0向数据库d_0读取A的值
- 由于指令的并发执行，s_0读取的值可能为2，也可能为4



事务与分布式事务

事务是原子的一组请求，这组请求不受其它客户端干扰，并且要么全部完成，要么不对服务器造成任何影响

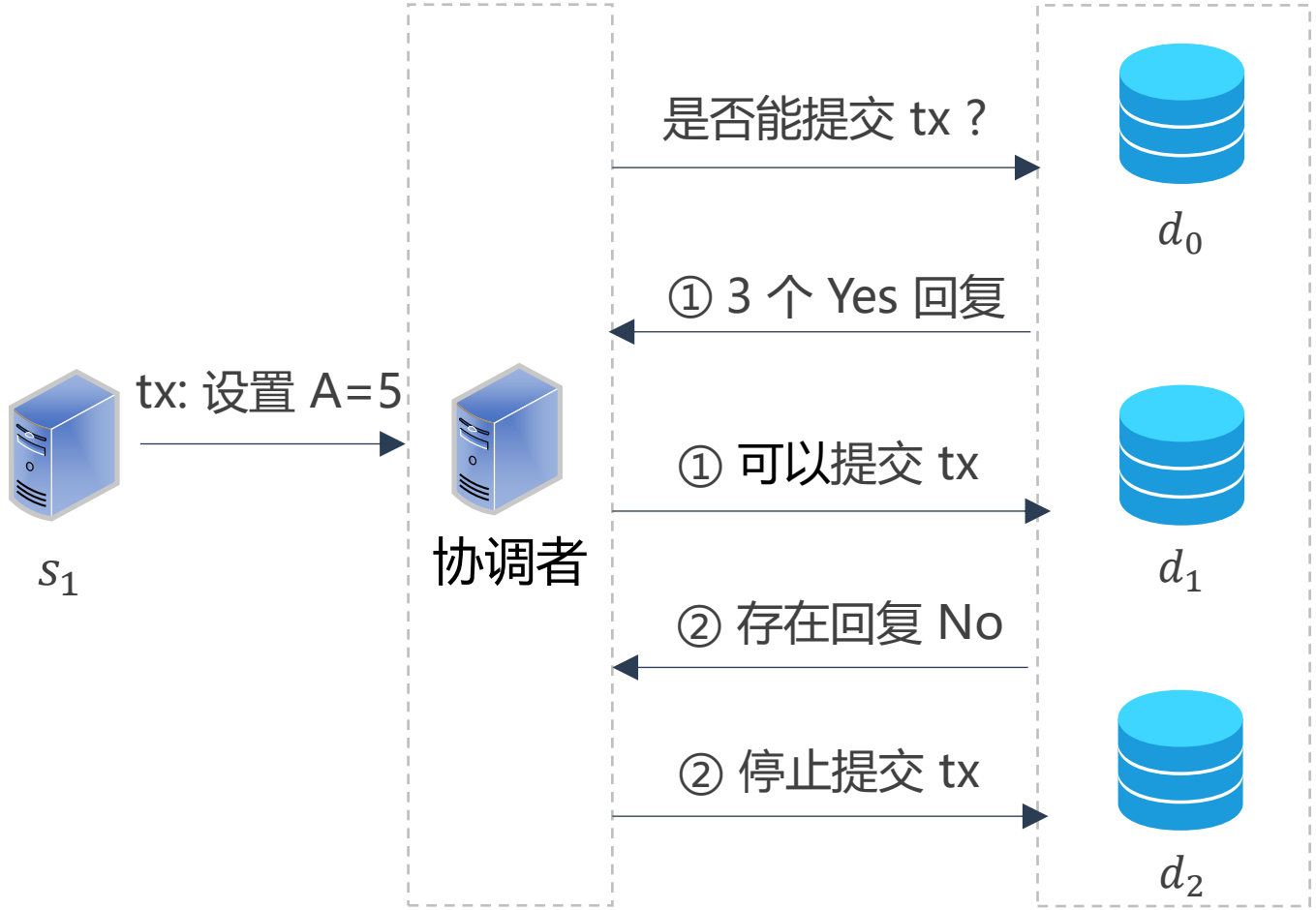
- 事务需要满足ACID特性
  - 原子性（Atomicity）：操作必须全部完成，或者不留下任何结果
  - 一致性（Consistency）：将系统从一个一致状态转换到另一个一致状态
  - 隔离型（Isolation）：每个事务的执行不受其它事务影响
  - 持久性（Durability）：一旦事务完成，其效果将被保存到持久存储中

分布式事务是访问由多个服务器管理的对象的事务

两阶段提交确保一致性

假设：状态A在三个数据库冗余存储，存在一个协调者引导二阶段提交的执行

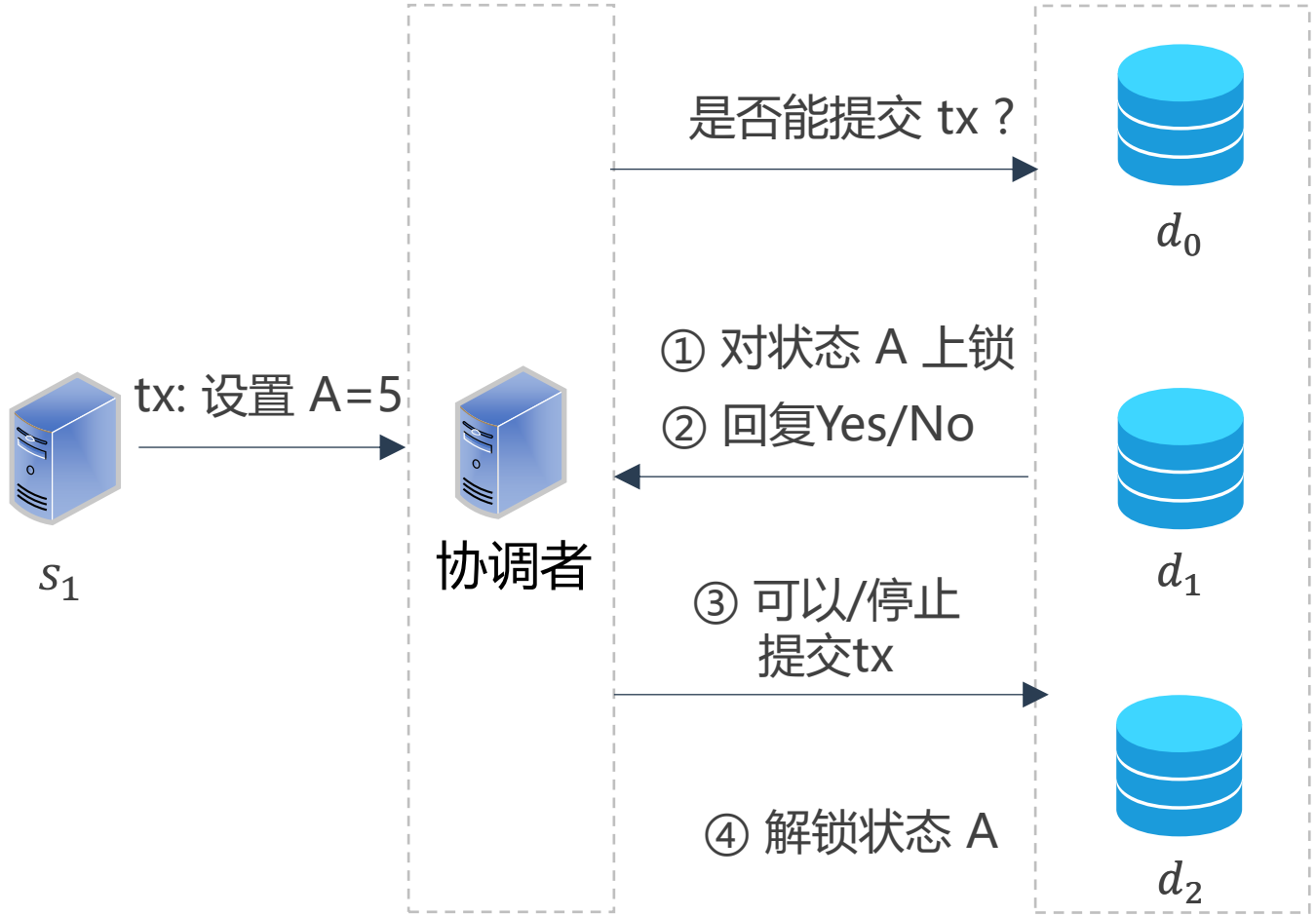
- 执行步骤
  - s_1向协调者请求将A设为5
  - 协调者向所有存储状态A的数据库询问是否可提交事务
  - 数据库若判断不冲突，则返回Yes；否则返回No
  - 当协调者收到所有数据库的Yes，通知它们提交该事务；如果收到No，则通知停止



基于锁机制确保隔离性

假设：状态A在三个数据库冗余存储，存在一个协调者引导二阶段提交的执行

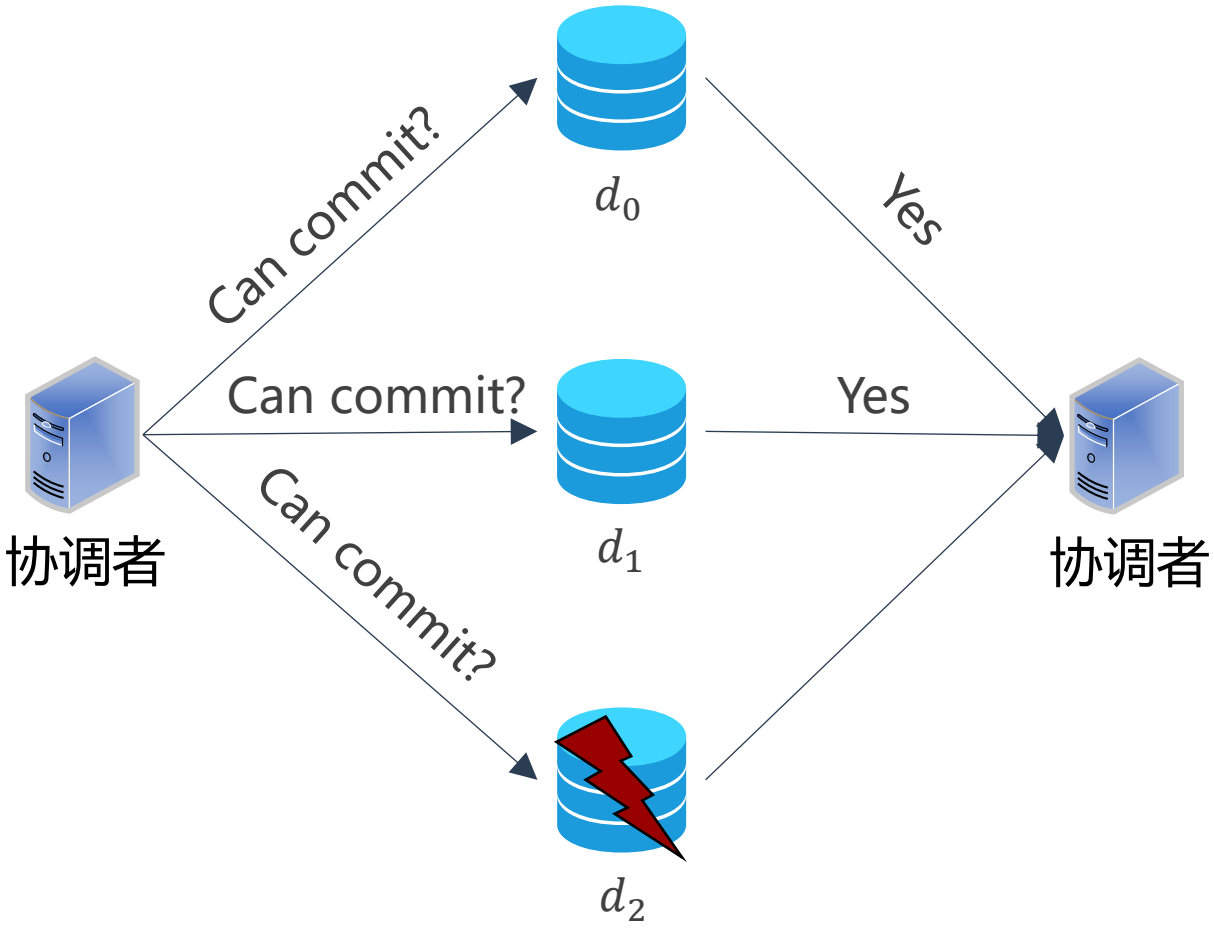
- 执行步骤
  - s_1向协调者请求将A设为5
  - 协调者向存储状态A的数据库询问是否可提交事务
  - 数据库若判断不冲突，则返回Yes，并对状态A上锁
  - 数据库等待协调者的通知，若收到Do commit，完成对状态A的设置并解锁状态A；若收到Abort，则直接解锁状态A



数据库节点故障

存在一个协调者和三个存储相同状态变量A的数据库，其中一个数据库出现故障

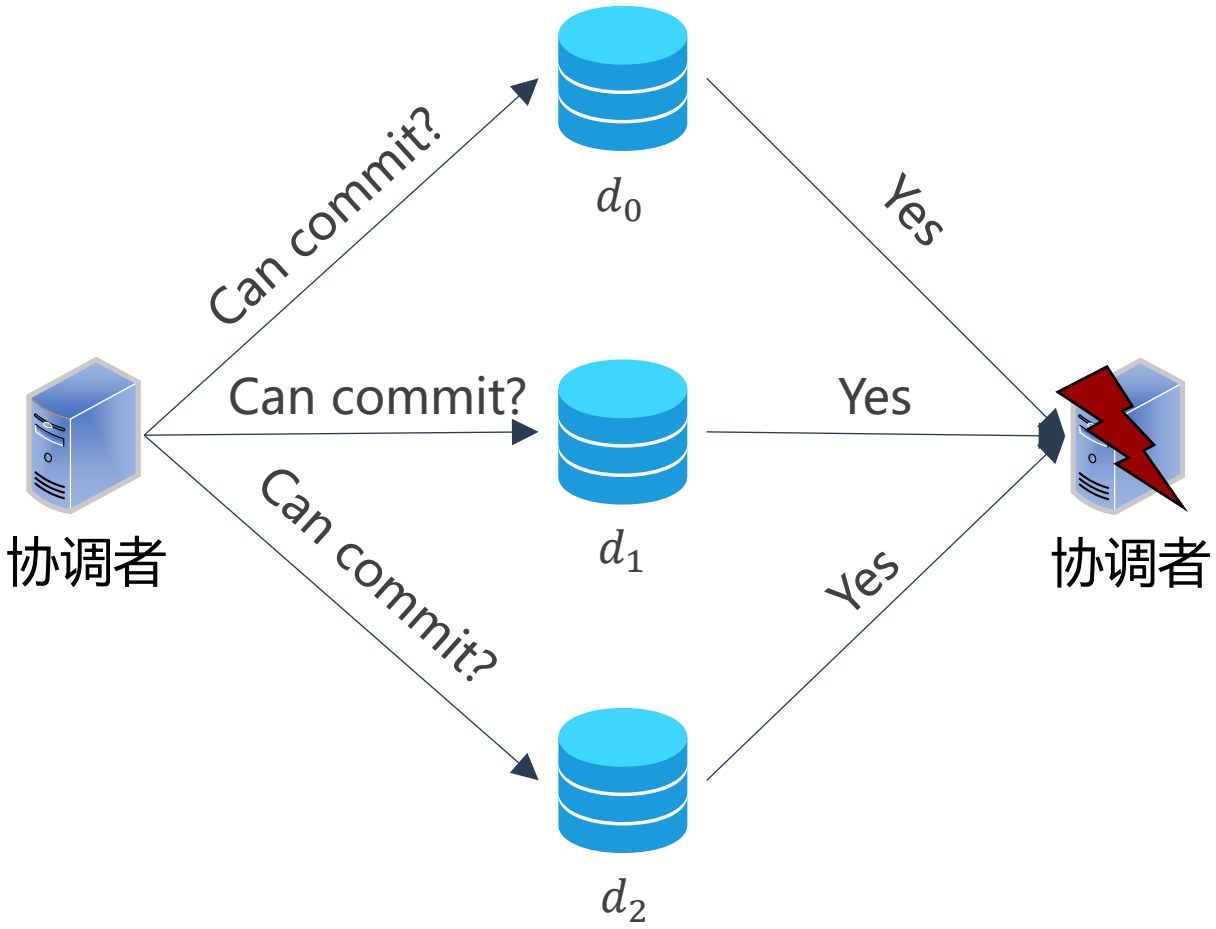
- 执行步骤
  - 协调者向三个数据库询问是否可以提交一个更改A值的事务
  - 数据库d_0和d_1向协调者返回Yes
  - 数据库d_2出现故障，没有反应
  - 协调者无法收到足够的Yes，陷入无限等待，或者采用计时机制，超时后通知所有节点停止提交



协调者故障

存在一个协调者和三个存储相同状态变量A的数据库

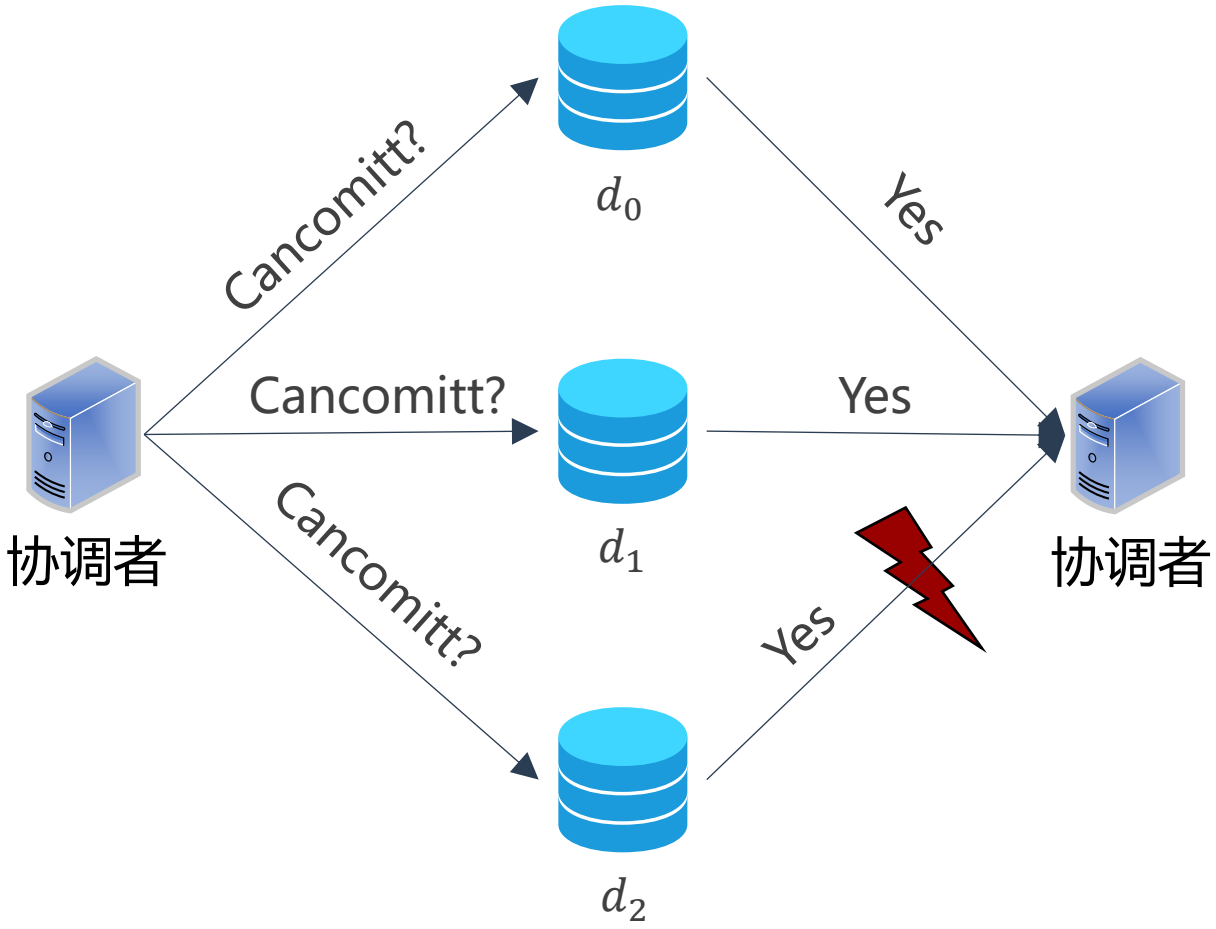
- 执行步骤
  - 协调者向三个数据库询问是否可以提交一个更改A值的事务
  - 三个数据库都向协调者返回Yes
  - 此时协调者出现故障
  - 数据库一直无法等到协调者的确认提交或停止提交消息，因此一直处于等待状态，且相关资源一直处于上锁状态



网络拥塞/链路故障

存在一个协调者和三个存储相同状态变量A的数据库

- 执行步骤
  - 协调者向三个数据库询问是否可以提交一个更改A值的事务
  - 三个数据库都向协调者返回Yes
  - 数据库d_2与协调者之间的通信链路故障，导致Yes消息没被收到
  - 协调者无法收到足够的Yes，陷入无限等待，或者采用计时机制，超时后通知所有节点停止提交



无法容错的根源

在简单的二阶段提交协议中，必须所有相关节点都进行回应后才成功提交，因此协议无法容忍任何故障

印度

我要当常任！

提案

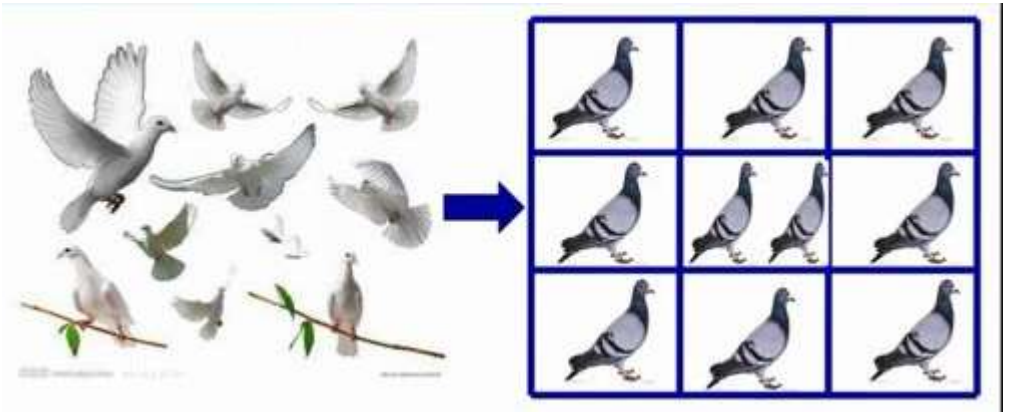
（同意，同意，同意，同意，反对）

五常国

必须五常国全部同意才能通过，只要保持每次投票的同意票数为4，提案永远无法通过

Quorum设计准则

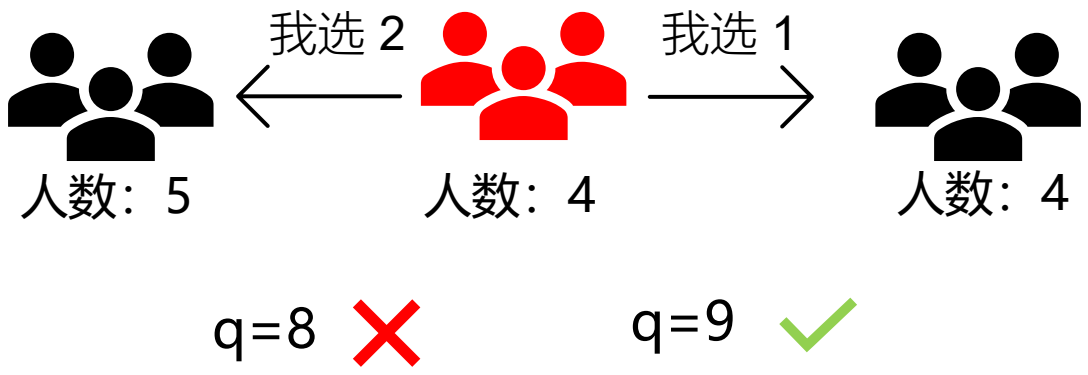
- **鸽巢原理：如果k+1个鸽子要放在k个鸽巢中，那么至少有一个鸽巢存在两个鸽子**

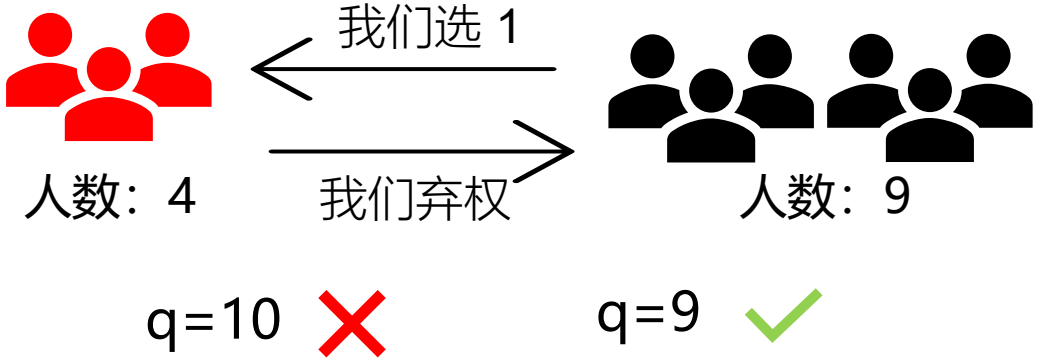


- 假设N个副本，更新操作在W个副本中更新成功之后，才认为此次更新操作成功
- 对于读操作而言，至少需要读R个副本才能读到此次更新的数据。其中，W+R>N，即W和R有重叠，一般，**W+R=N+1**
- 假设系统中有5个副本，W=3，R=3。初始时数据为(V1，V1，V1，V1，V1）当某次更新操作在3个副本上成功后，认为此次更新操作成功，数据变成：(V2，V2，V2，V1，V1）最多只需要读3个副本，一定能够读到V2(此次更新成功的数据)

Quorum设计准则

基本规则：分布式系统每阶段提案中，诚实节点最多只对一个提案投一次票





- 目的1-安全性：Quorum值q应该足够大，避免通过两个**矛盾提案**

  解决方案： 针对两个提案可能产生的最高总投票数 ≤ 2*q – 1

- 目的2-活性：Quorum值q应该足够小，避免错误节点不投票使提案**无法通过**

  解决方案： q ≤ 最低可能产生的投票数；q > 拜占庭节点个数

故障容错算法-Paxos

This submission was recently discovered behind a filing cabinet in the *TOCS* editorial office. Despite its age, the editor-in-chief felt that it was worth publishing. Because the author is currently doing field work in the Greek isles and cannot be reached, I was asked to prepare it for publication.

The author appears to be an archeologist with only a passing interest in computer science. This is unfortunate; even though the obscure ancient Paxon civilization he describes is of little interest to most computer scientists, its legislative system is an excellent model for how to implement a distributed computer system in an asynchronous environment. Indeed, some of the refinements the Paxons made to their protocol appear to be unknown in the systems literature.

The author does give a brief discussion of the Paxon Parliament's relevance to distributed computing in Section 4. Computer scientists will probably want to read that section first. Even before that, they might want to read the explanation of the algorithm for computer scientists by Lampson [1996]. The algorithm is also described more formally by De Prisco et al. [1997]. I have added further comments on the relation between the ancient protocols and more recent work at the end of Section 4.

Keith Marzullo
University of California, San Diego

Leslie B. Lamport

美国著名的计算机科学家，微软研究院首席研究员，
著名的排版系统LaTeX的作者，
拜占庭将军问题和Paxos算法的提出者，2000年获得了Dijkstra奖，2012年获得冯·诺依曼奖，**2013年获得图灵奖**

Paxos的主要思想

存在有Leader和无Leader两种模式，基于Quorum机制确保一致性和容错性

Paxos算法中的3种角色，一个节点可同时担任多个角色，通常情况下由客户端担任提议者，服务端担任接受者和学习者

- Proposer(提议者)，Acceptor(接受者)，Learner(学习者)

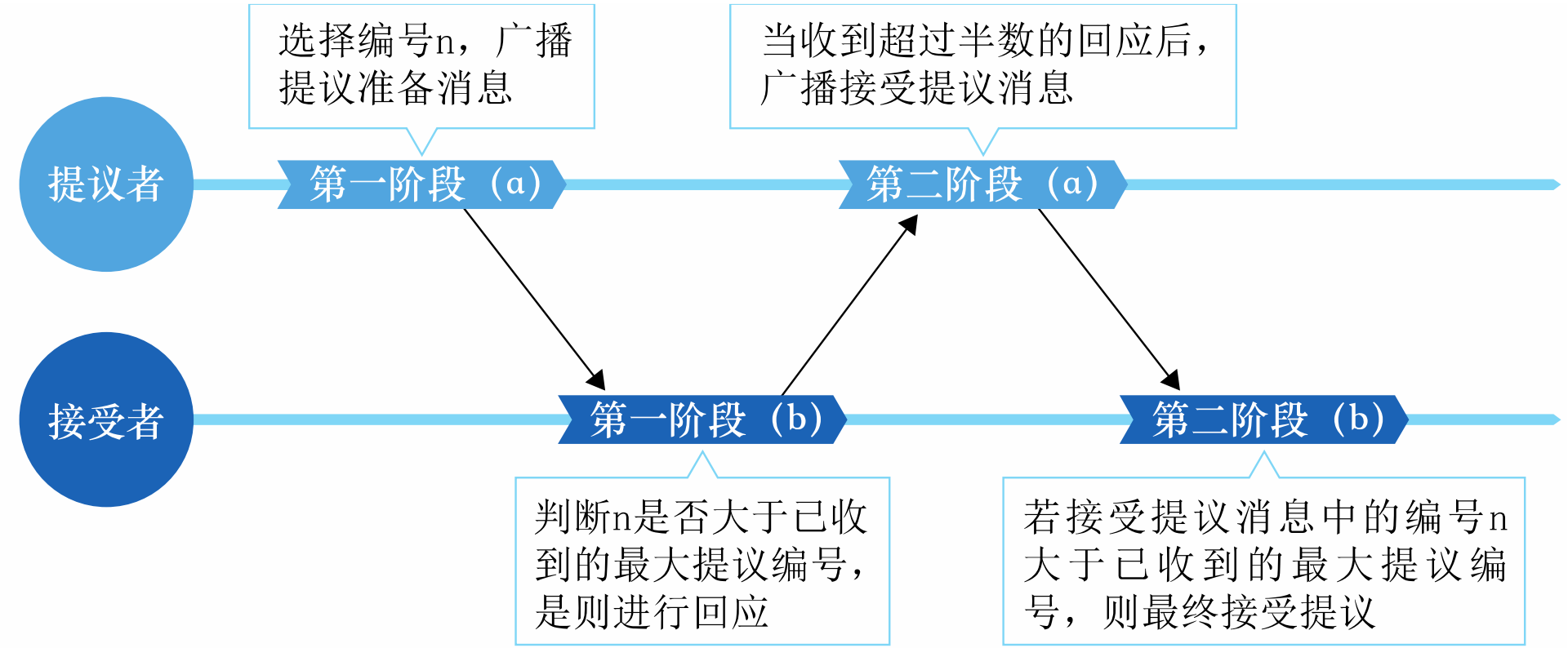
Paxos的基本流程

- 提议者为交易分配一个序号，从而形成提案，然后将其发送给所有的接受者
- 接受者选择需要接受的提案，然后对提议者进行回应
- 当提议者收到超过半数接受者的回应后，则通知接受者最终接受该提案
- 接受者接受提案后告知学习者，学习者收到多数接受者信息后学习该提案

注. 详情可参考：
Lamport L. Generalized consensus and Paxos[J]. 2005.

无Leader模式下的运行流程

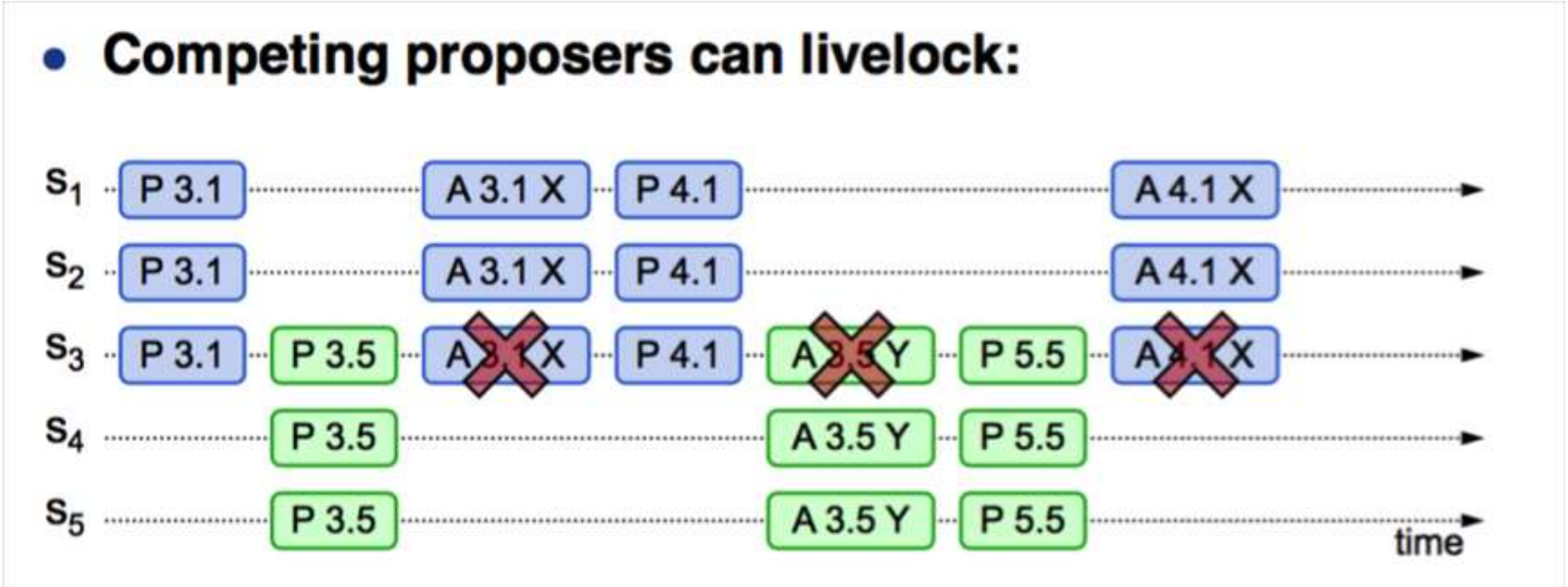
一个理想的共识流程



- 实际情况：多个提议者存在，每个提议者单独提议各自的提案；接受者面临不同提议者的提案，可能需要不断接受提议编号更大的提案
- 共识达成条件：提案完成前三次交互之前无更大提议号提案出现

一种永远无法达成共识的极端场景

每个提议者独立提议时，不仅效率低下，还可能产生活锁，即不断产生冲突



解决思路：在提议者中选取一个领导者进行提议，其它提议者不参与提议

注. Paxos及更多改进的详细介绍可参考：
Lamport L. Fast paxos[J]. Distributed Computing, 2006, 19(2): 79-103.

Raft的主要思想

Raft基于Leader完成共识过程，并基于Quorum机制确保一致性和容错性

- **在Raft中，服务器可以扮演下面角色之一：**
  - Leader: 处理所有客户端交互，日志复制等，一般一次只有一个Leader
  - Follower: 类似选民，完全被动
  - Candidate: 候选人，可能被被选为新的Leader

- **一般状态下Raft运行的基本流程：**
  - 所有交易被发往Leader，由Leader为其分配序号发给所有Follower
  - 当收到超过半数Follower的回应时，提交并执行交易，然后回复客户端
  - Follower提交交易后，在后续的交互中通知Follower提交相同的交易

第3节 信任问题

- 访问控制
- 信任评价模型
- 拜占庭容错

零信任网络

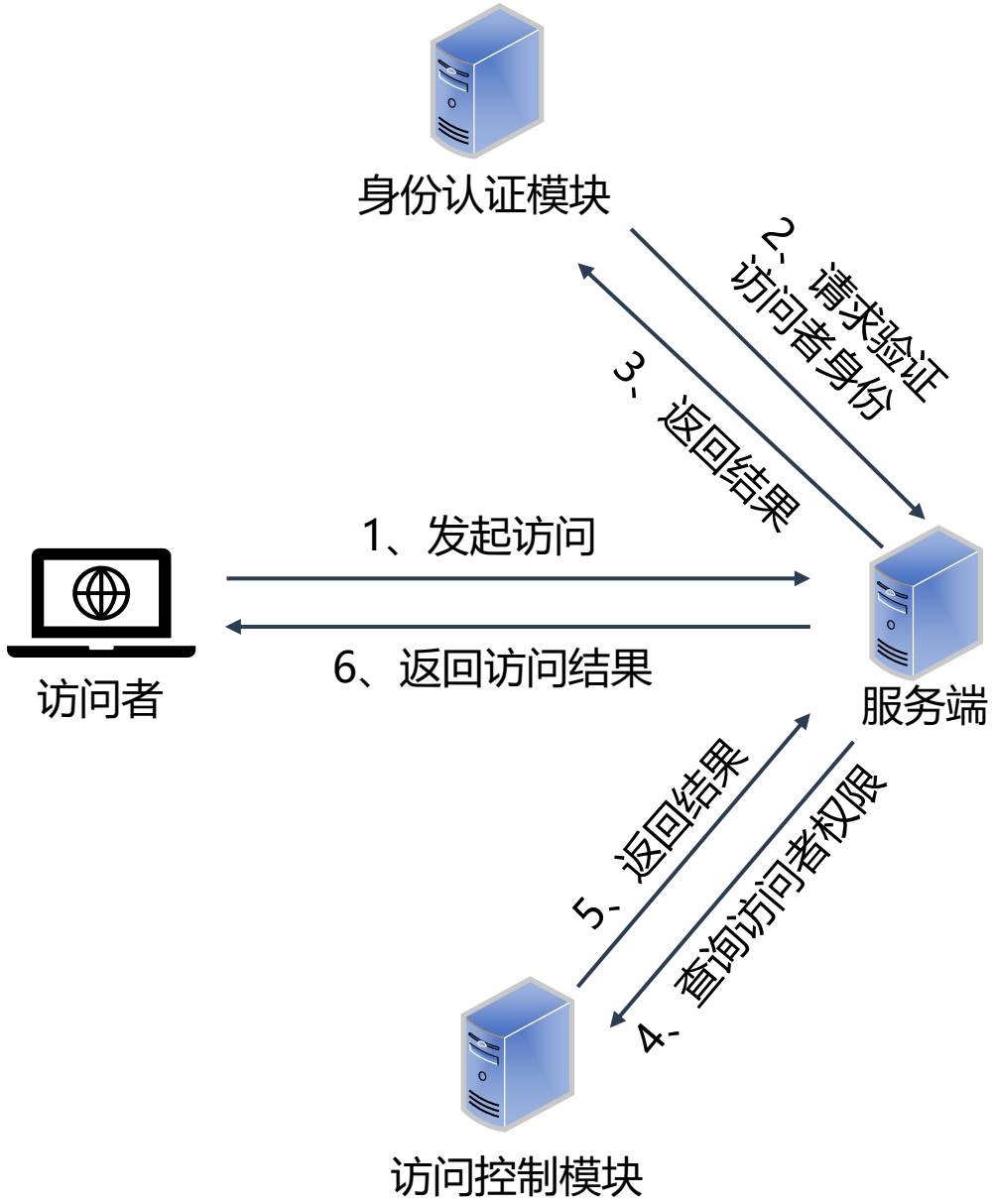
“从来不信任，始终在校验” （Never Trust, Always Verify）

- 零信任模型不依靠建立隔离墙来保护可信的资源，而是接受“不可信”或“坏人”无处不在的现实，试图让全体资源都拥有自保的能力
- 零信任默认不应该信任企业网络内部和外部的任何人/设备/应用，需要基于认证和授权重构访问控制的信任基础

零信任对传统访问控制机制进行了范式上的颠覆
其本质是以身份为基石的动态可信访问控制

零信任下访问的基本流程

- 访问者发起对服务端的访问
- 服务端收到访问指令后对其进行验证，主要包含以下两个步骤
  - 请求身份认证模块验证访问者身份
  - 根据身份以及被访问资源请求访问控制模块判断访问者是否存在权限
- 若验证都通过后，服务端允许此次访问，并返回访问结果

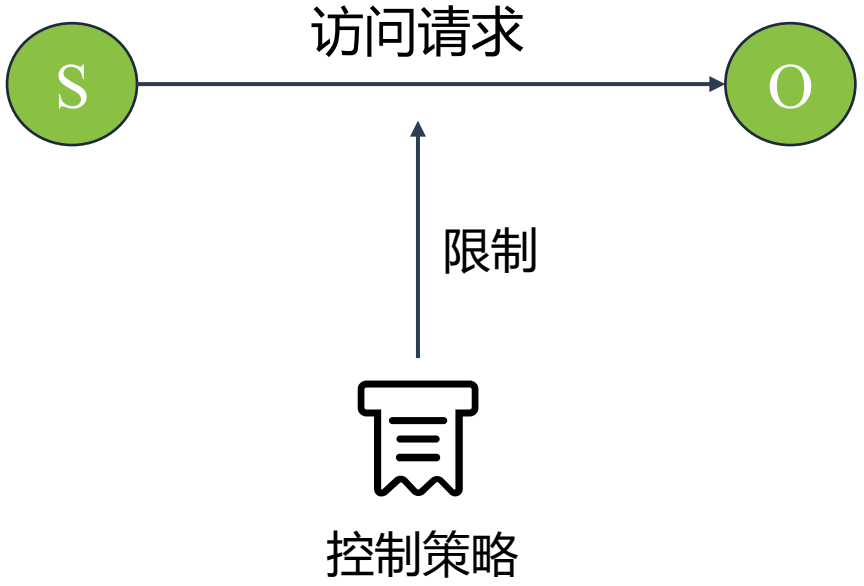


访问控制

访问控制（Access Control）指系统按用户身份，以及该身份所在的策略组来限制用户对某些信息项的访问，或限制其对某些控制功能的使用

- 主体S (Subject)
  - 提出对资源访问的具体请求
- 客体O (Object)
  - 提出对资源访问的具体请求
- 控制策略A (Attribution)
  - 是主体对客体的相关访问规则集合，即属性集合

- 访问控制规定了主体（用户或者进程）对客体（数据资源）的访问权限



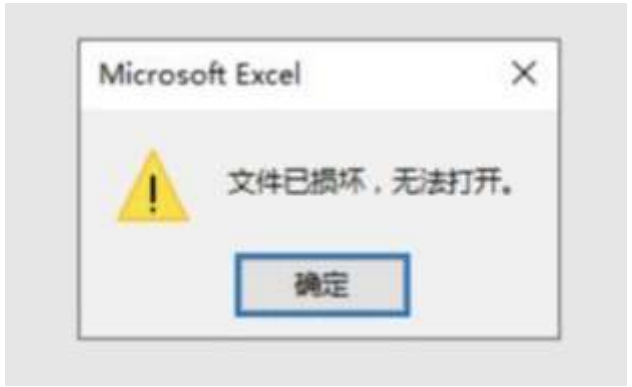
访问控制的局限性

访问控制只能解决被访问者对访问者的信任问题，
但无法解决访问者对被访问者的信任问题，访问者可能访问到恶意、无效资源

- 访问者对被访问者的信任问题主要出现在：
  - 文件下载
  - 资源共享
  - 访问网站链接

- 典型场景：P2P文件共享解决方法：信任评价模型

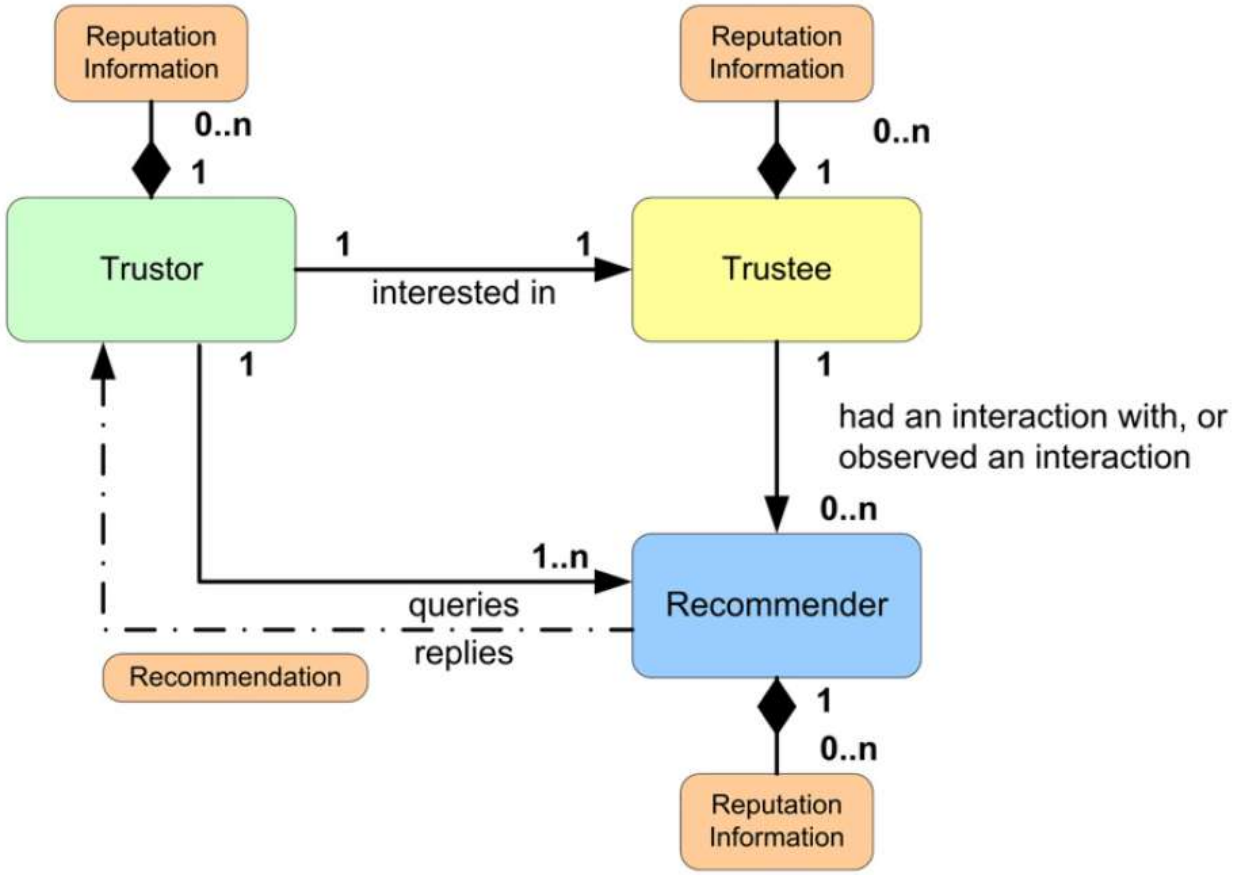
gnutella



信任评价模型

- 问题：如何描述？如何计算？如何传递与更新
- 信任模型细化：行为模型；计算模型；传播模型

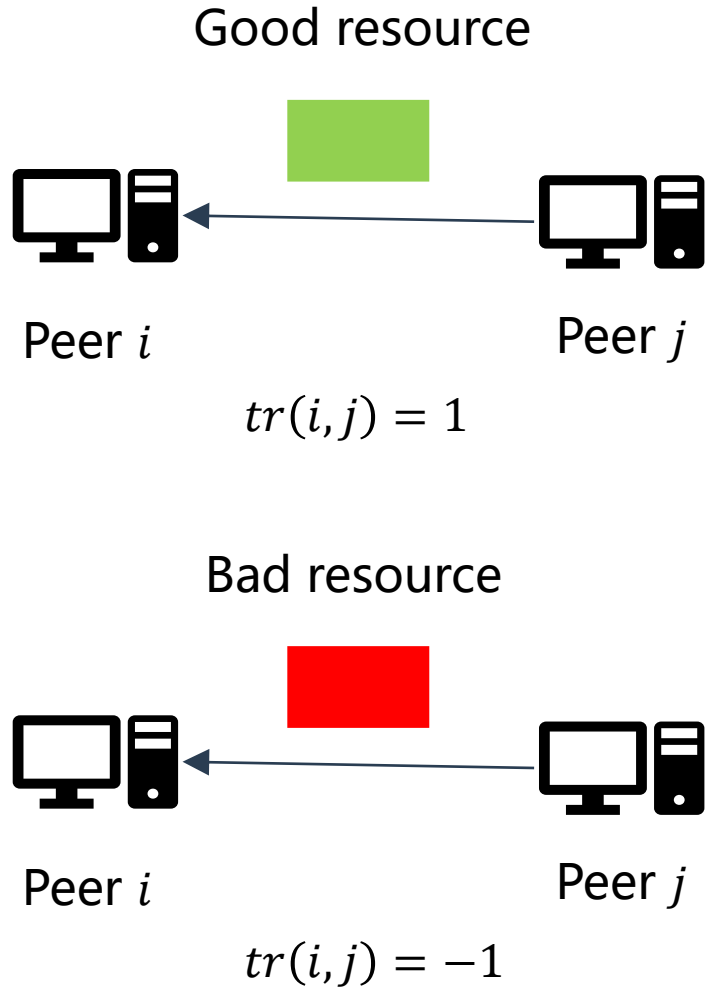
- 包含角色
  - Trustee：待评价节点
  - Trustor：期待与Trustee交互节点
  - Recommender：与Trustee交互过



基本概念

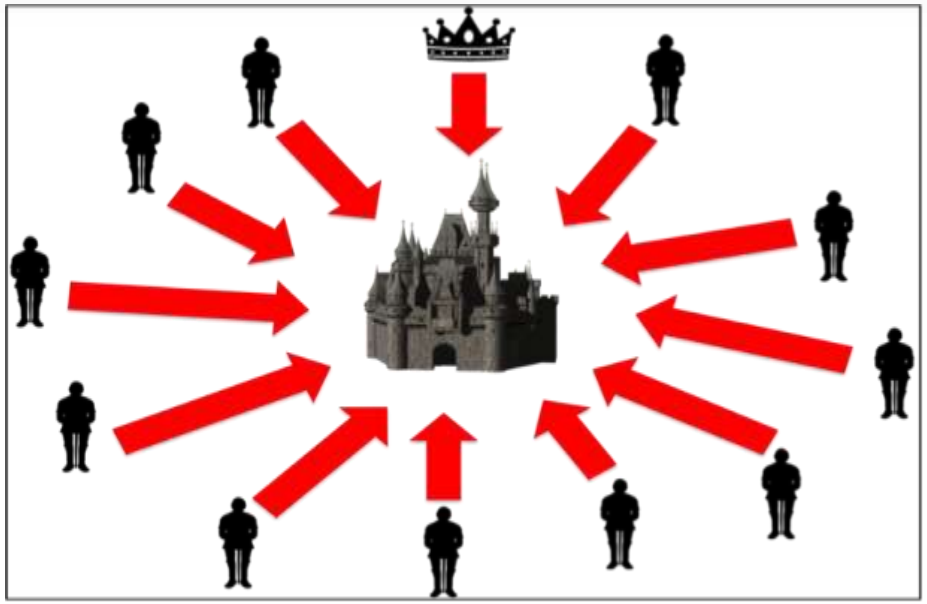
对节点的本地信誉值为与其交互的历史好评数减去历史差评数

- 对于节点 i 和节点 j 之间的每次交互，两者之间存在相互评价
  - $tr(i,j)=1$ 表示节点 i 认为节点 j 表现正常
  - $tr(i,j)=-1$ 表示节点 i 认为节点 j 表现不正常
- 节点 i 对节点 j 评价正常的总次数为 $sat(i,j)$
- 节点 i 对节点 j 评价不正常的总次数为 $unsat(i,j)$
- 节点 i 对节点 j 本地信任值为 s_{ij}
  - $s_{ij}=sat(i,j)-unsat(i,j)=\sum tr(i,j)$

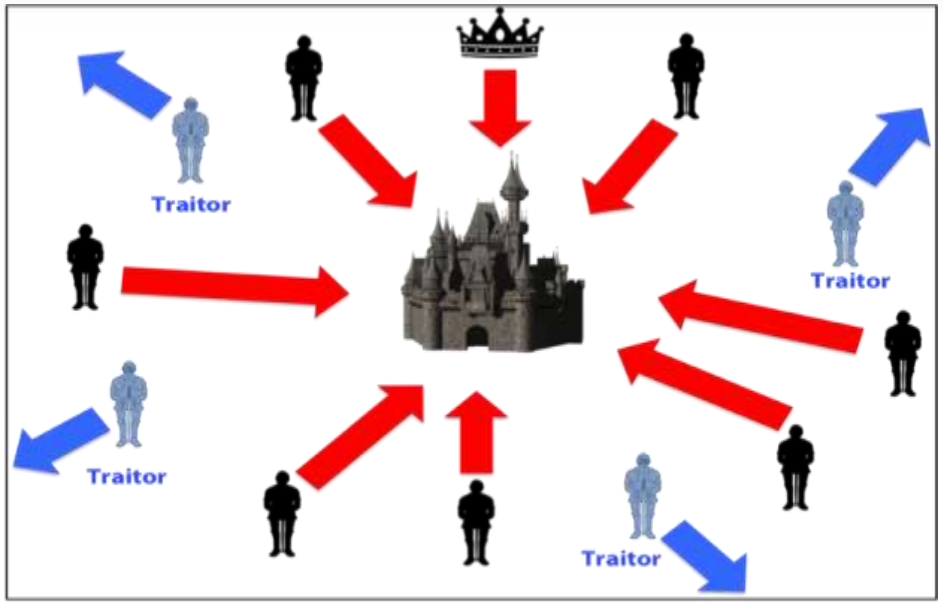


拜占庭将军问题

- **背景**：拜占庭罗马帝国疆域辽阔，许多将军守卫疆土；战时，部队相隔远，只能靠信差传递消息；至少超过半数的将军对是否共打敌人达成一致，否则失败；内部可能存在叛徒
- **问题**：存在叛徒时，忠诚的将军们如何达成共识



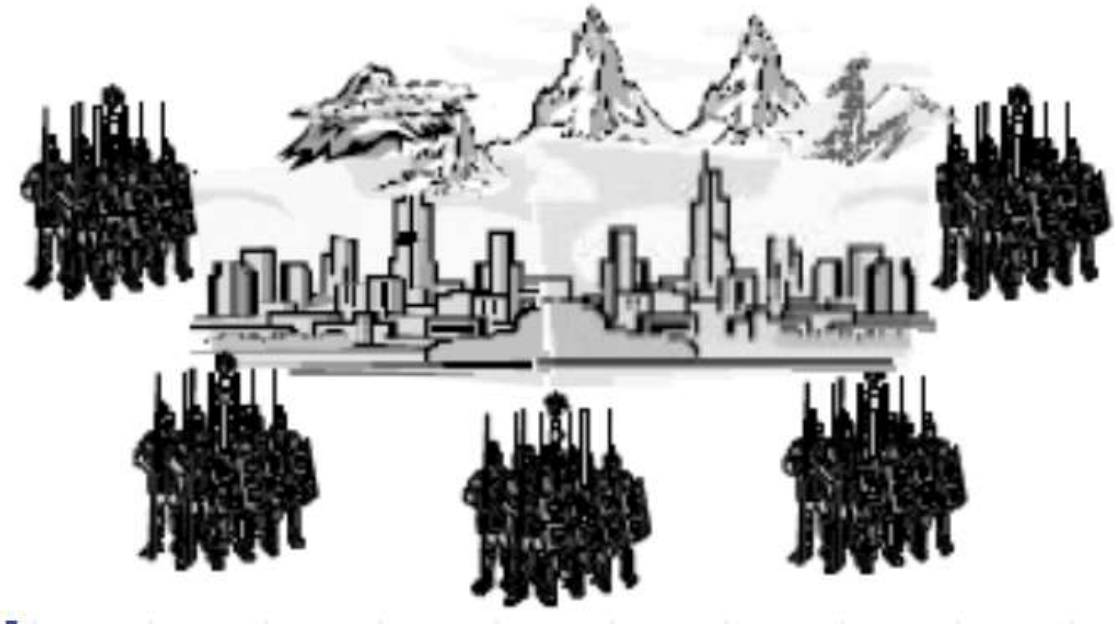
Coordinated Attack Leading to Victory



Uncoordinated Attack Leading to Defeat

拜占庭将军问题

- **概念：**将军 等价于 计算机系统的部件
- **问题抽象：**
  - 每个部队由一个将军领导
  - 一共有n个将军，其中一些是叛徒
  - 所有军队将敌方包围，观察敌军
  - 将军间靠通信兵交流



- **目标：**存在所有忠诚的将军采取**同样**的行动；**少量**叛徒无法使诚实的将军们接受一个**坏**的计划
- **注意：**不必识别出叛徒

拜占庭将军问题

The Byzantine Generals Problem

Leslie Lamport (SRI International),
Robert Shostak (SRI International),
Marshall Pease (SRI International)

Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement. It is shown that, using only oral messages, this problem is solvable if and only if more than two-thirds of the generals are loyal; so a single traitor can confound two loyal generals. With unforgeable written messages, the problem is solvable for any number of

This research was supported in part by the National Aeronautics and Space Administration under contract NAS1-15428 Mod. 3, the Ballistic Missile Defense Systems Command under contract DASG60-78-C-0046, and the Army Research Office under contract DAAG29-79-C-0102.
Authors' address: Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025.

Paper originally published in *Transactions on Programming Languages and Systems*, 4(3), July 1982, pp. 382–401.

Leslie B. Lamport

[PDF] The Byzantine generals problem

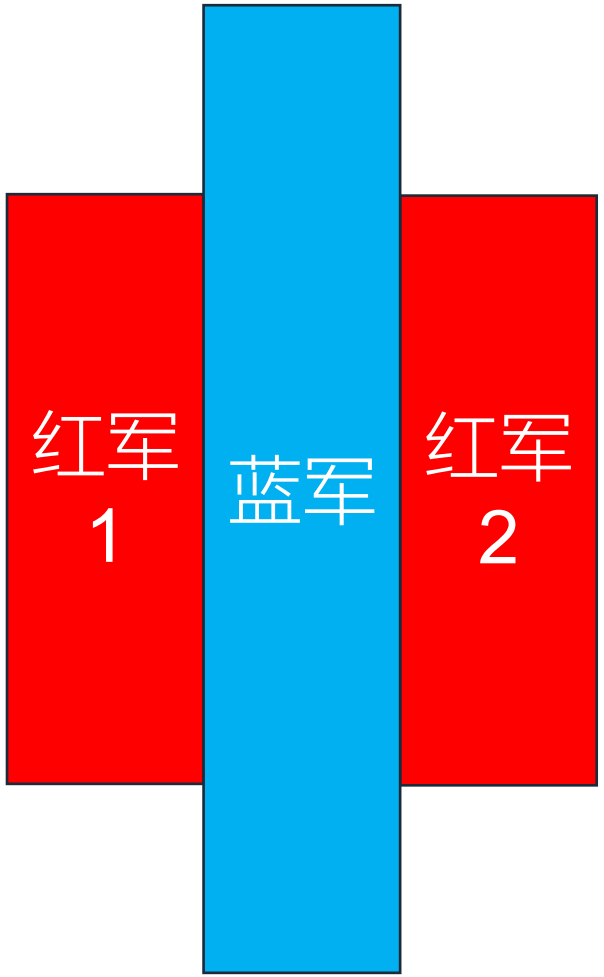
L Lamport, R Shostak, M Pease - Concurrency: the Works of Leslie …, 2019 - dl.acm.org

Page 1. The Byzantine Generals Problem Leslie Lamport (SRI International), Robert Shostak (SRI International), Marshall Pease (SRI International) … 382–401. Page 2. 204 The Byzantine Generals Problem generals and possible traitors …

☆ 99 被引用次数: 7814 相关文章 所有 199 个版本

问题定义

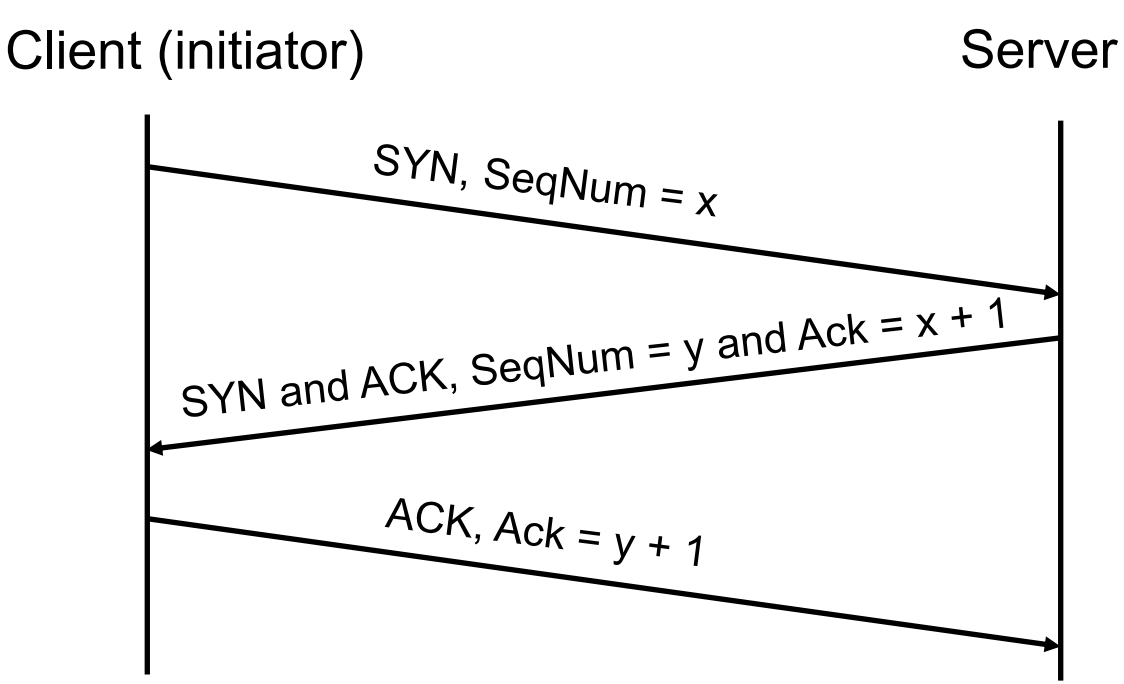
- 拜占庭将军问题的前提
  - 可靠信道：信息可以准确无误的传达给接收方
- 不可靠信道：两军问题
  - 红军必须在同一时间攻打蓝军才能胜利
  - 红军1、2通信必须经过蓝军区域，消息可能被截获、篡改
  - 如何确定攻打时间?

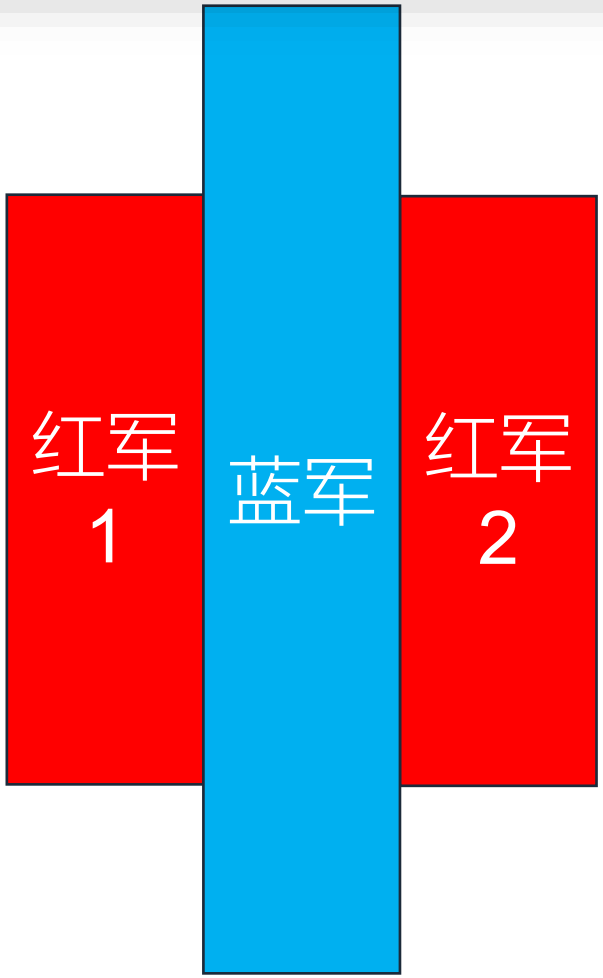


问题定义

两军问题实例:TCP通信

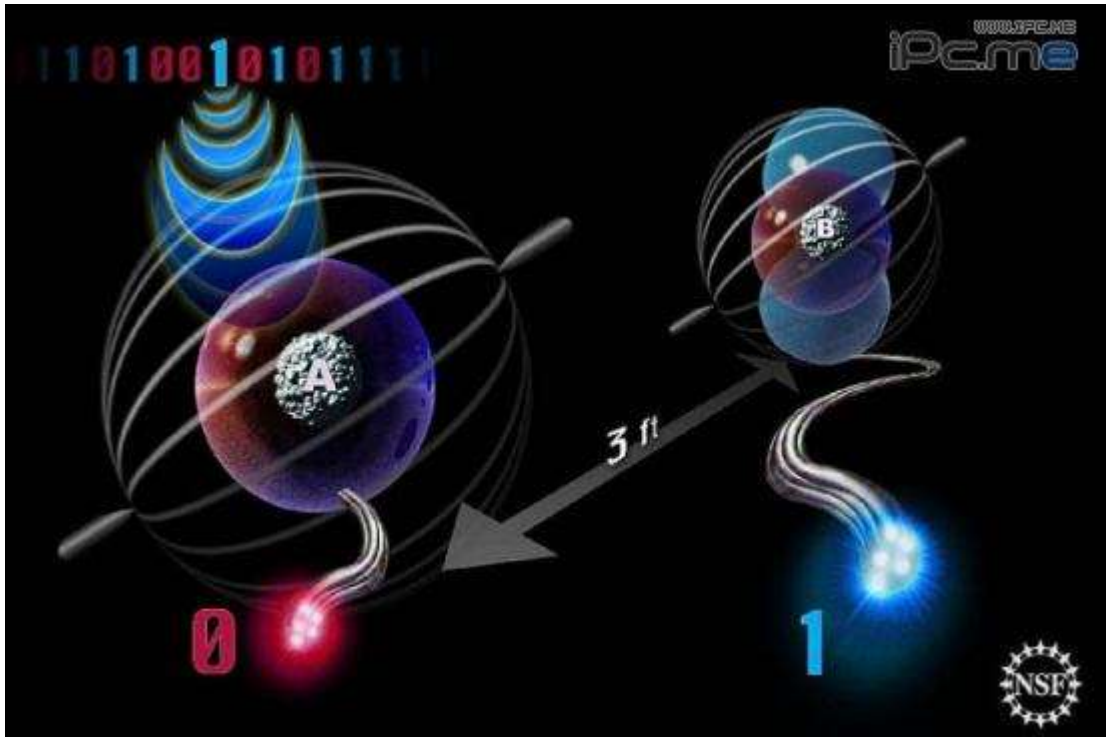
TCP三次握手

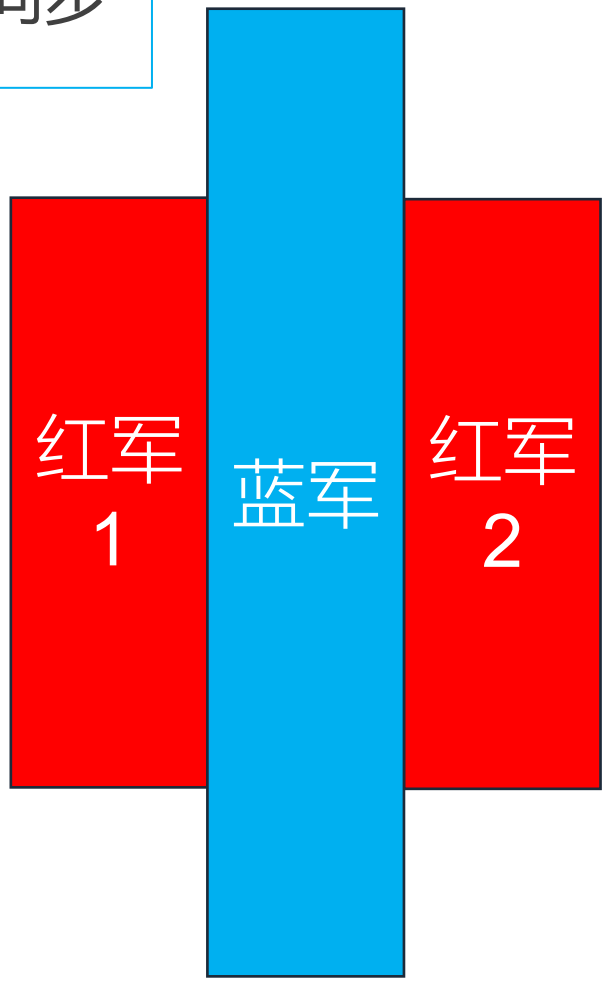




问题定义

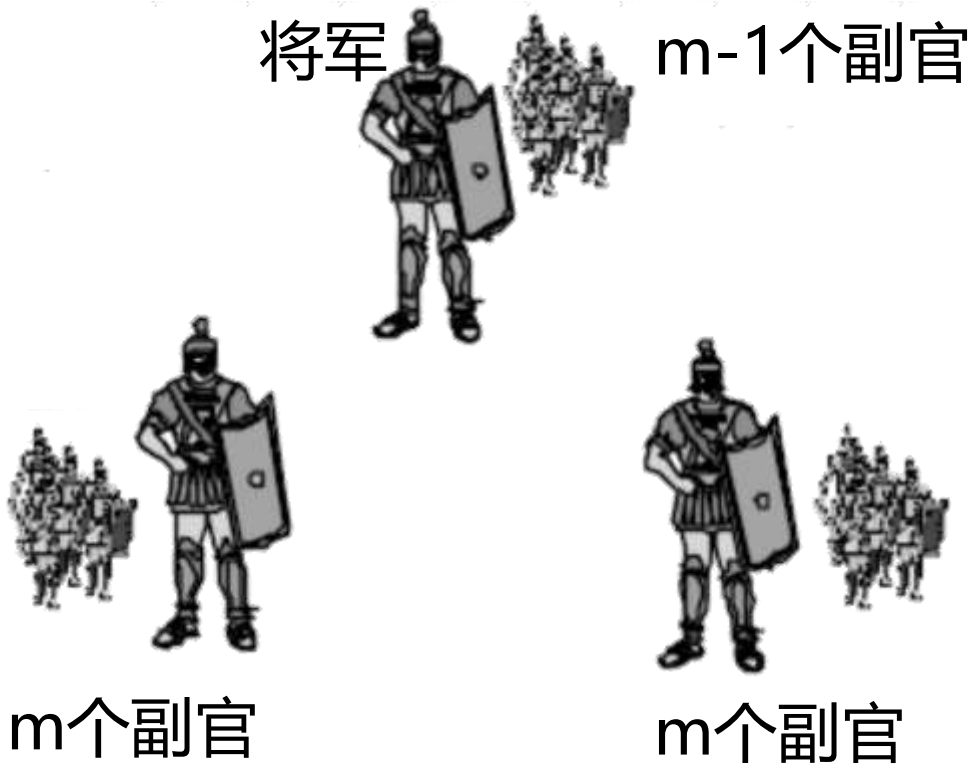
- 两军问题终极解决方案：量子通讯
- 处于量子纠缠的两个粒子，无论相隔多远都可以同步





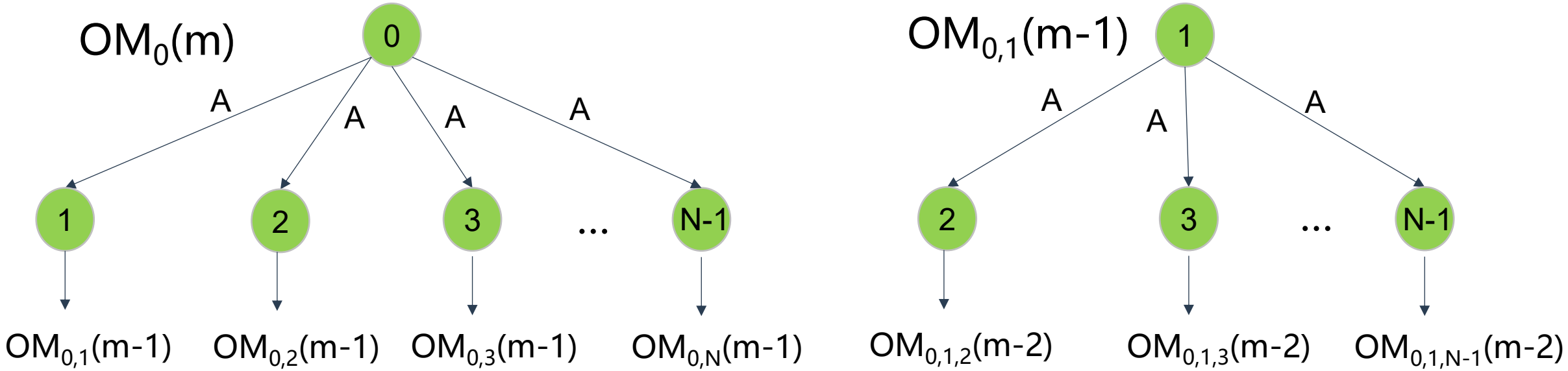
口头消息下的不可能性

- 一般情况下，将军与将军之间基于口头消息传播信息时
  - 在有m个叛徒时，少于3m+1个参与者（包含叛徒），问题无解
- 通过反证法证明：
  - 假设共有3m个将军，其中m个是叛徒
  - 将问题简化为3将军问题
- 3m个将军有解 等价于 3将军问题有解

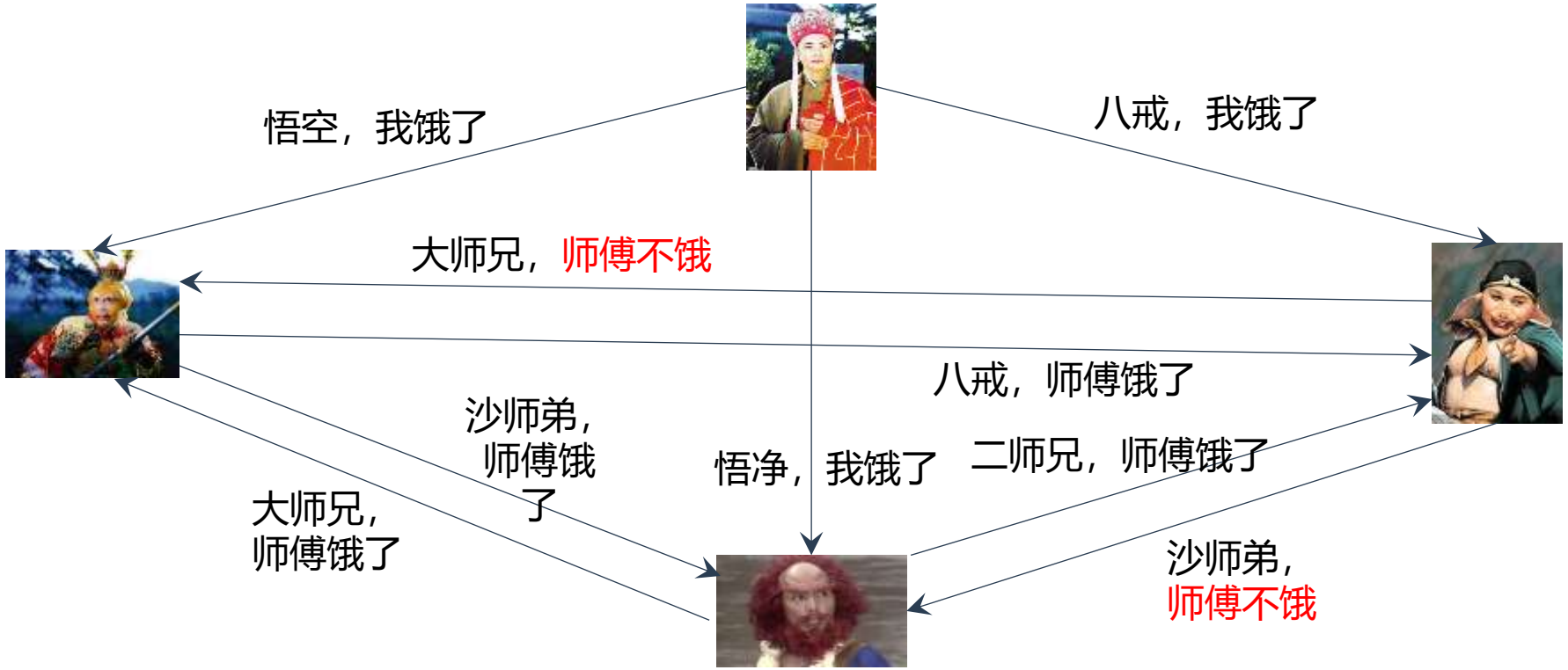


口头消息协议

- 假设N个副官编号分别为0,1,...,N-1，初始状态下0为将军，并执行$OM_0(m)$
- $OM_0(m)$的运行流程为：将军0给其它副官发送指令，然后其它副官分别执行$OM_{0,i}(m-1)$，将从将军处收到的指令发给剩余副官
- $OM_{k1,k2,...,kv,i}(m-v)$表示的OM算法是在除去副官k1,k2,...,kv后剩余的副官中执行，且副官i将从副官kv执行$OM_{k1,k2,...,kv}(m-v+1)$时发送的指令发给剩余副官



四将军问题实例：懒惰的八戒



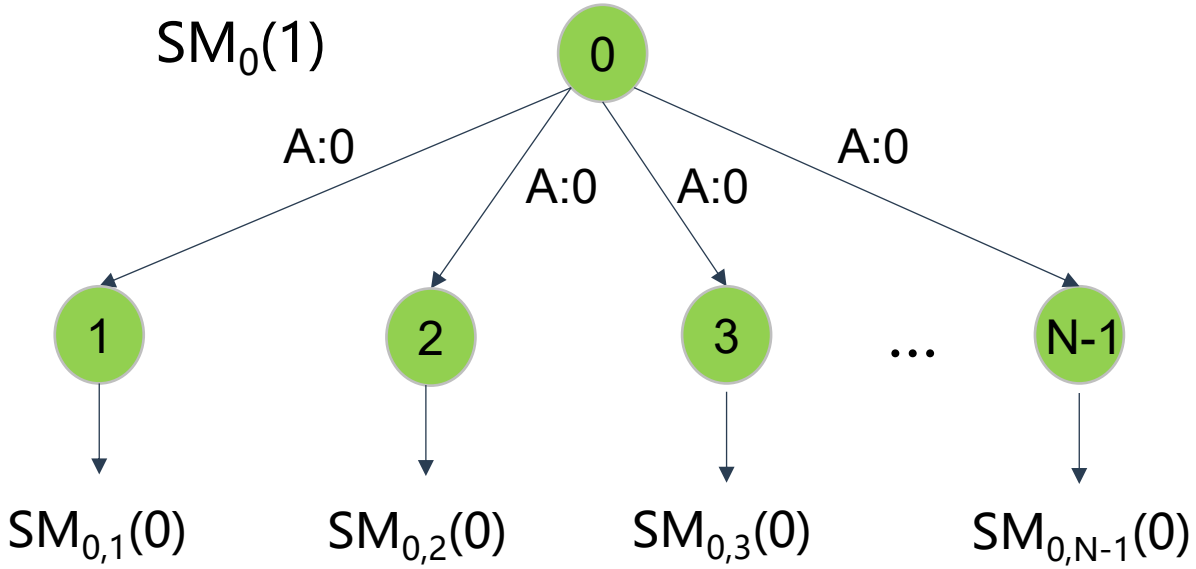
- 悟空和沙僧都收到了（饿了，饿了，不饿），最后都认定师傅饿了
- 满足了IC1，师傅、悟空、沙僧的意见是一致的
- 满足了IC2，悟空和沙僧都听师傅的

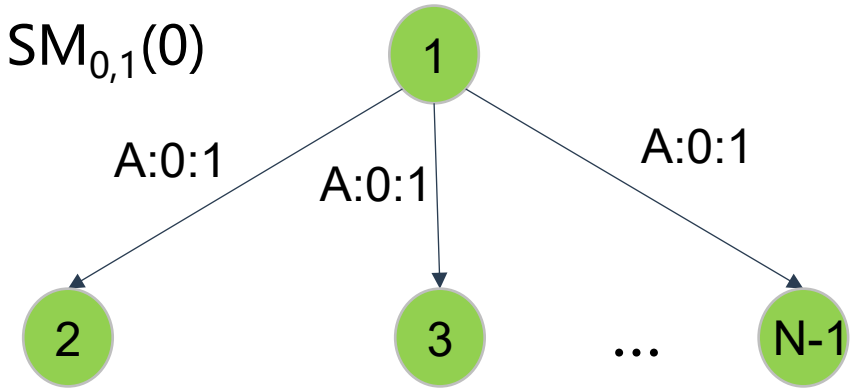
口头消息的缺点

- 开销巨大
  - OM(m) 调用 n – 1 次 OM(m-1)
  - OM(m-1) 调用 n – 2 次OM(m-2)
  - OM(m-2) 调用 n – 3 次OM(m-3)
  - ……
  - OM(m-k) 被调用 n-1, n-2, …, n-k次
  - $O(n^m)$
- 口头消息不能溯源（无法确定消息是否被篡改），导致需要传输大量的信息

签名消息协议

- 假设N个副官编号分别为0,1,…,N-1，初始状态下0为将军，并执行$SM_0(1)$，从而给其它副官发送附带签名的指令x:0
- 收到x:0后，副官i执行$SM_{0,i}(0)$，即在签名指令x:0上附带自己签名得到x:0:i，然后将其发送给其它副官
- 每个副官收集来自所有副官的指令（包括自己和将军在内），当出现不同指令时则撤退（说明将军是叛徒）；否则当超过半数为相同指令时选择该指令



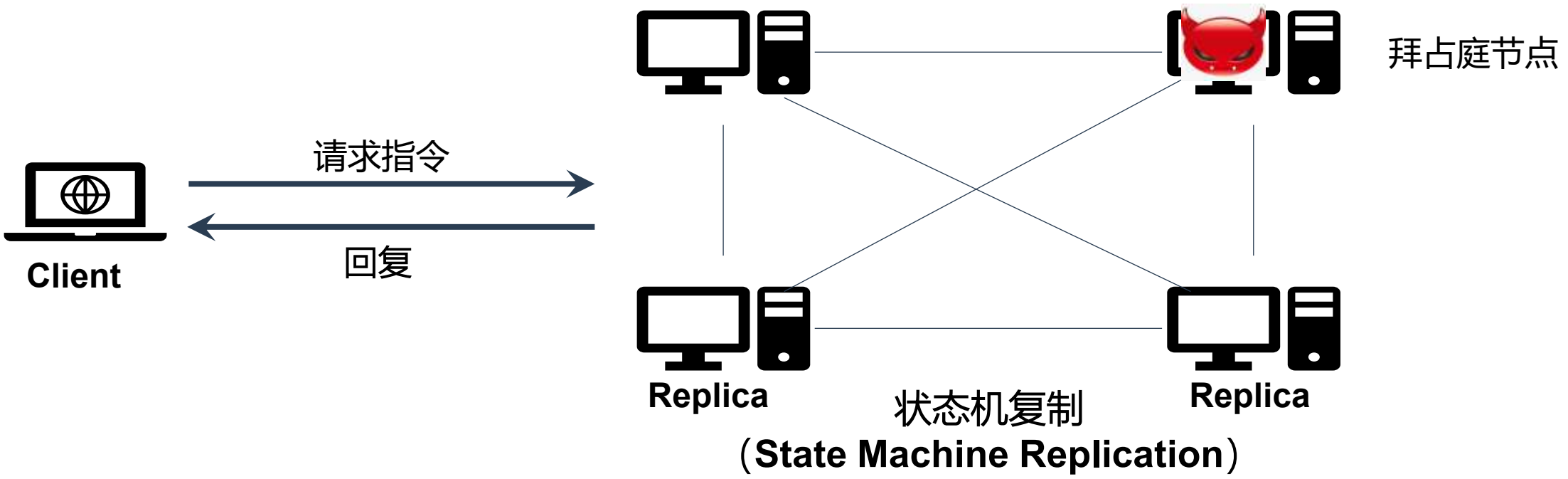


拜占庭容错共识的应用场景：区块链

区块链：构造一个对客户端来说可信的分布式账本；容忍少数拜占庭节点作恶

解决分布式系统不同节点间协作时的信任问题

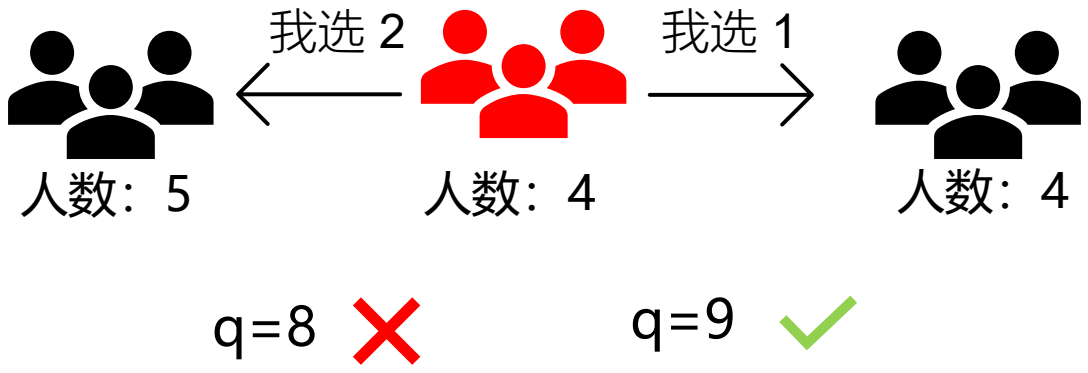
实际模型—状态机复制

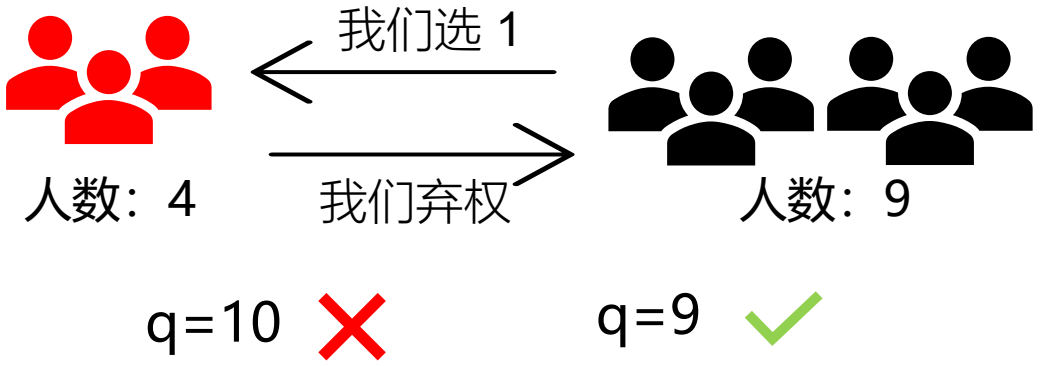


- **安全性（Safety）**：对于客户端的所有请求，服务端任意两个正确节点都能获得一致的排序结果并执行（something “bad” will never happen）
- **活性（liveness）**：服务端最终会就结果达成一致，因此对于任意客户端的请求，服务端最终总能产生正确的答复（something “good” will must happen (but we don’t know when)）

拜占庭共识：Quorum设计准则

基本规则：分布式系统每阶段提案中，诚实节点最多只对一个提案投一次票





- 目的1-安全性：Quorum值q应该足够大，避免通过两个**矛盾提案**

  解决方案： 针对两个提案可能产生的最高总投票数 ≤ 2*q – 1

- 目的2-活性：Quorum值q应该足够小，避免错误节点不投票使提案**无法通过**

  解决方案： q ≤ 最低可能产生的投票数；q > 拜占庭节点个数

异步BFT共识中的Quorum设计

异步网络模型：实际网络延迟不存在任何延迟上限，但节点间经过无限重传能确保消息最终被正确接收

为了容忍f个拜占庭节点，Quorum的值q以及节点总数N的设计:

具体设计： 针对两个提案可能产生的最高票数为 N+f;

最低票数为 N-f;

边界条件： $f+1 \le q \le N-f$, $N+f \le 2*q-1 \Rightarrow N \ge 3f+1, 2f+1 \le q \le N-f$

最优解： $q = 2f+1$， $N = 3f+1$。

实用拜占庭容错-PBFT

Barbara Liskov

美国杰出计算机科学家，麻省理工学院电子电气与计算机科学系教授

2004年约翰·冯诺依曼奖得主，2008年图灵奖得主

美国第一位计算机科学女博士，历史上第二位获得图灵奖的女性

- Practical Byzantine Fault Tolerance，OSDI 1999
- 状态机复制系统
- 目标：即使在少部分服务器为拜占庭节点的情况下，使来自客户端的请求在每个服务器上都按照一个确定的顺序执行
- 适用于有强一致性要求的私有链和联盟链场景，IBM超级账本可选共识协议

PBFT的主要思想

PBFT基于Leader完成共识过程，并基于Quorum机制确保安全性和活性；
PBFT采用部分同步假设，假设故障节点个数不超过m个，总节点3m+1个，
Quorum设计q=2m+1

- PBFT在Paxos、Raft等故障容错算法的二阶段共识的基础上增加了一个阶段
- 二阶段共识后的交易处于准备完成状态，此时，节点广播信息告诉其它所有节点自己已完成准备
- 当节点收到q个准备完成的投票时提交并执行该交易
- 所有节点都对Leader节点进行监测，若长时间未收到Leader发送的消息，则触发视图变更协议，更换Leader节点

Hotstuff

HotStuff: BFT Consensus with Linearity and Responsiveness

——PODC 2019

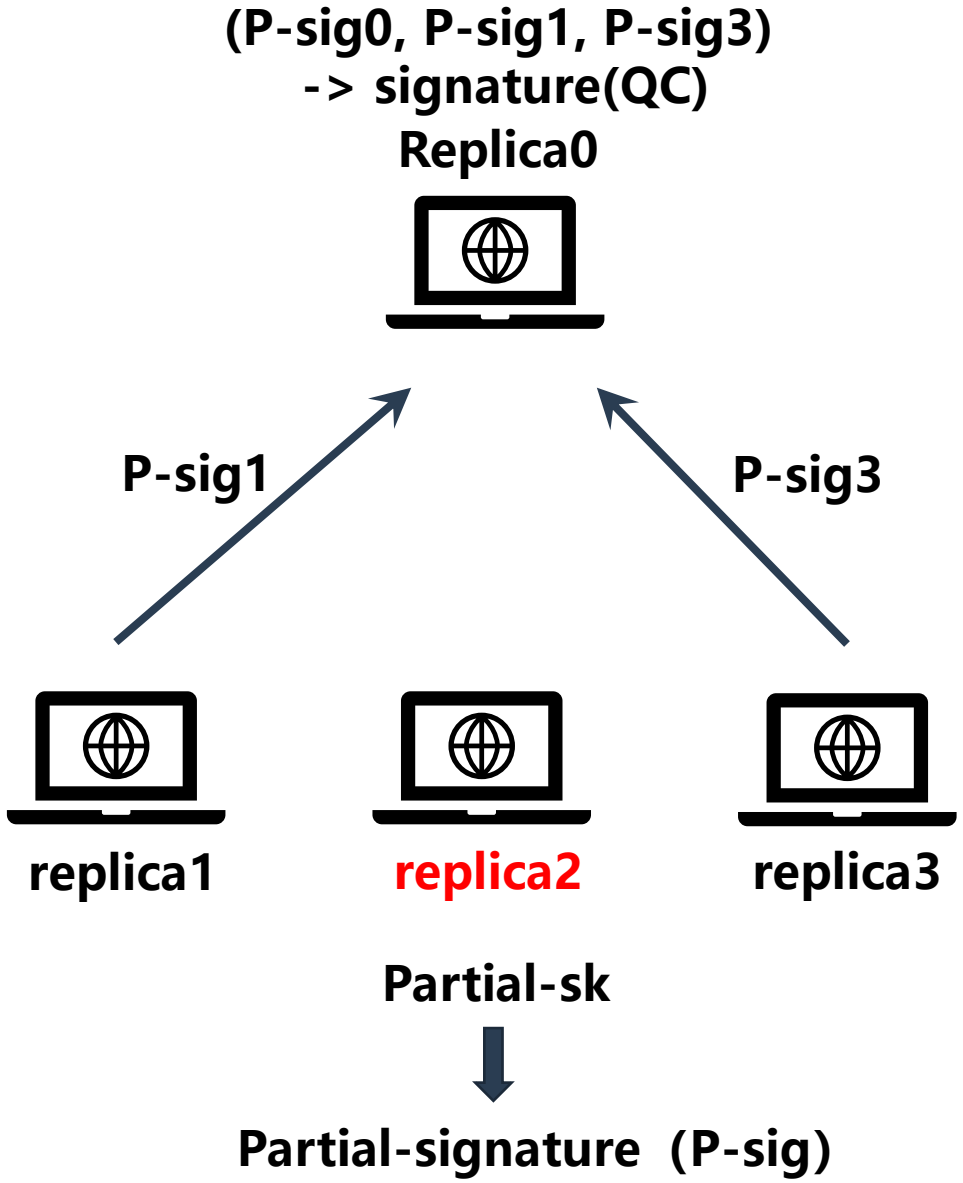
- **目标：实现具备线性复杂度的拜占庭共识**

HotStuff与libra的关系：

Facebook在2019年发布了libra白皮书，里面提到的LibraBFT共识是基于HotStuff演化而来

协议基础

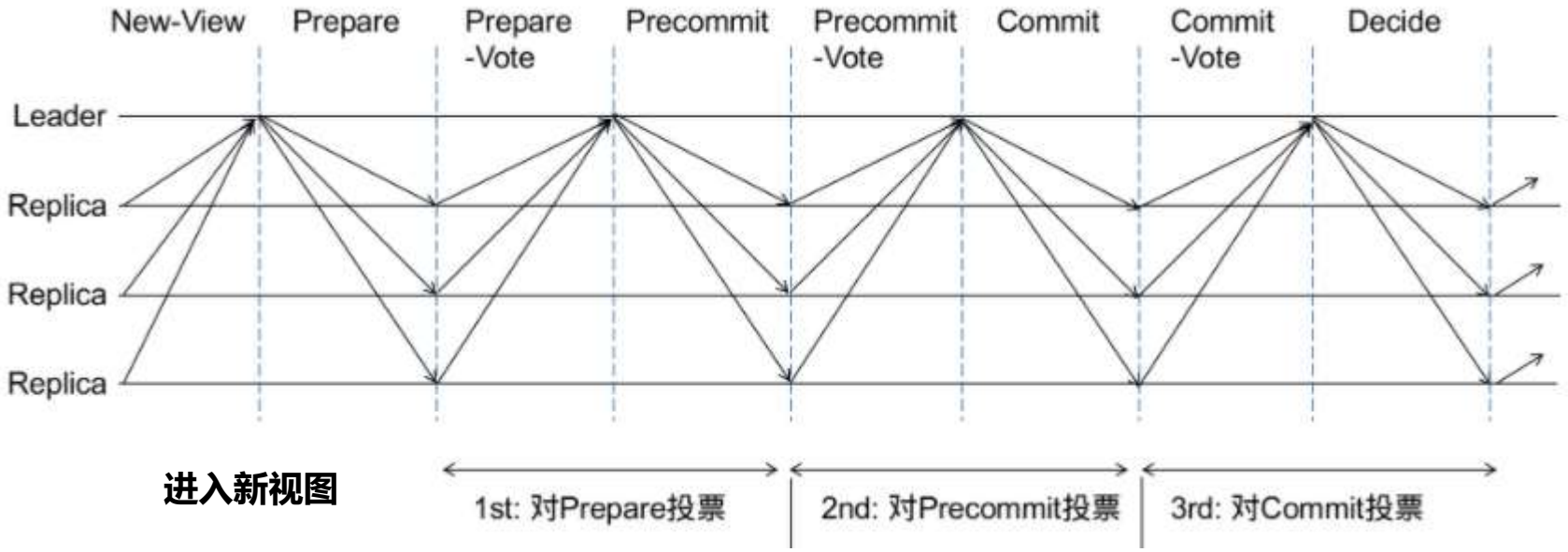
- **门限签名：**在（k，n）门限签名中，每个节点拥有部分私钥Partial-sk，可进行部分签名，任意k个节点的部分签名可以合成一个完整签名，并能被全局公钥验证
- **View**：共识视图，每个视图由不同leader节点引导完成一轮共识
- **投票**: 节点对提案进行部分签名
- **QC(Quorum Certificate)**：由2m+1个不同节点的部分签名合成的一个全局签名，可通过全局公钥验证并证明2m+1个节点进行了投票



Hotstuff（基本方案）

通过门限签名把PBFT的流程改成了四阶段算法（去除了客户请求和回复客户两个阶段），存在以下特点：

- 每次交互只在Leader和其他节点间进行，无需所有节点都进行广播，减少通信复杂度（$O(n^2)$ -> $O(n)$）
- 每个阶段都是**同构**的



第4节 总结

总结

剖析分布式系统安全问题的根源，探讨如何从交互网络、分布式算法和信任三方面构建安全、稳定、可信的分布式系统

第一节
建立安全、稳定的交互网络

- 交互网络的组成
- 交互信道安全
- 弹性覆盖网络
- 防御日蚀攻击

了解交互网络的组成以及如何构建安全的交互网络

第二节
分布式算法

- 协作过程中的安全隐患
- 时钟同步算法
- 并发问题
- 故障容错的一致性算法

挖掘分布式系统各组件之间如何进行协同

第三节
信任问题

- 访问控制
- 信任评价模型
- 拜占庭容错

了解分布式系统中的信任问题及解决方案

展望

新的应用场景对分布式系统的研究提出了新的挑战



- 随着比特币的出现，**区块链**技术得到飞速发展，但是如何进一步提升区块链性能并拓展其应用范围对分布式系统研究提供了新的方向
- 近些年来，为了打破数据孤岛，**联邦学习**被提出并得到发展，如何协调互不信任的组织之间实现可信数据协同成为新的挑战
- 互联网的发展**暴露了传统设计的弊端**，如何构建可信互联网基础设施至关重要