区块链共识算法基础

51CTO学院微职位

课程介绍

课程模块	学习周期	主讲老师	作业	知识点
区块链基础知识	1周	邹均	1	区块链架构及组件，区块链信任机制，比特币基础，以太坊基础，其它区块链平台
区块链共识算法基础	1周	邹均	1	共识算法简史，拜占庭将军问题，FLP定理，CAP定理，数据库一致性算法，共识算法属性

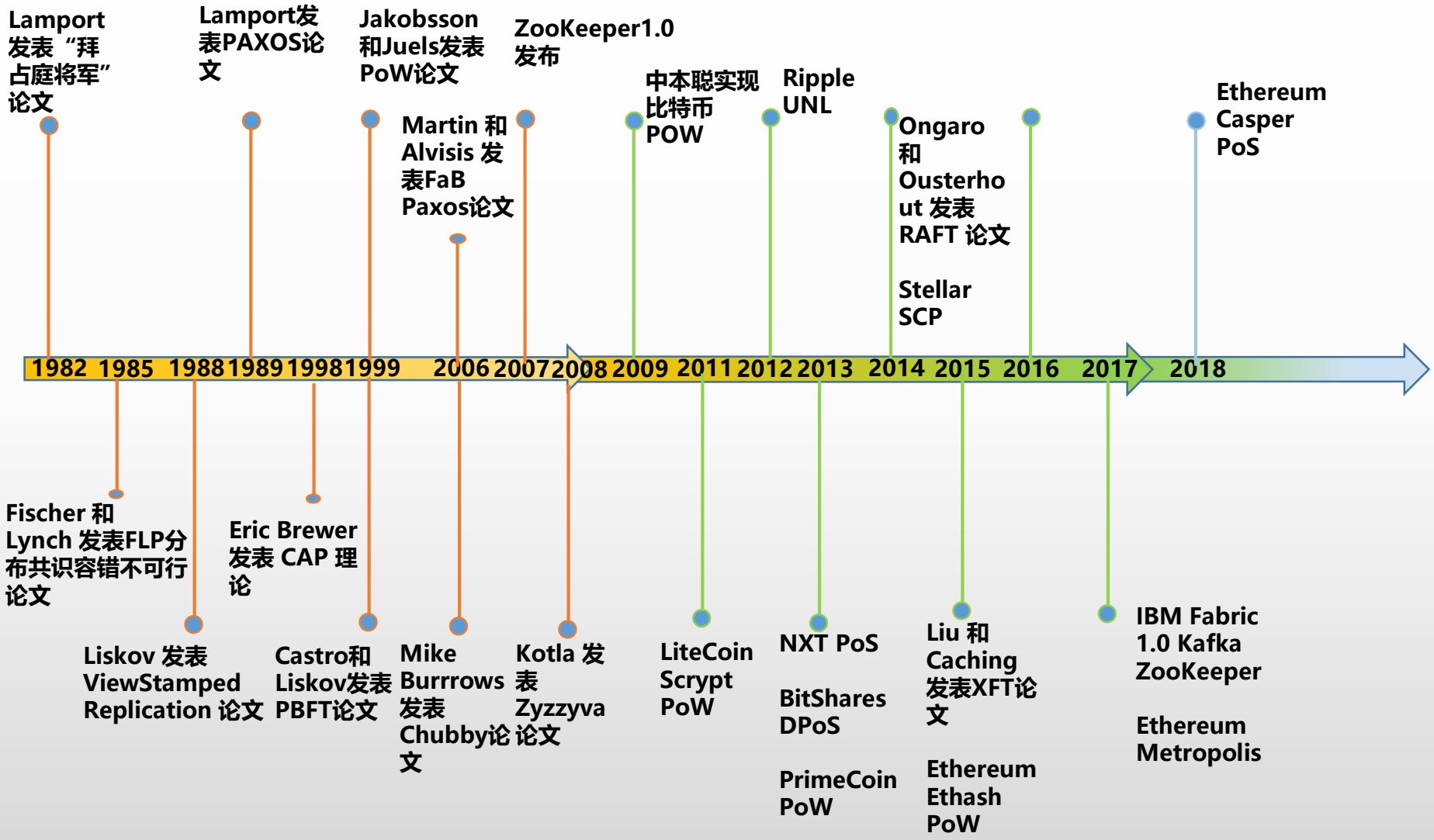
第二模块：区块链共识算法基础

课程介绍：学习区块链共识机制的相关理论和主流的共识算法

预期达到的目标：掌握区块链信任机制的基础 - 共识机制的基础理论，熟悉拜占庭将军问题，以及共识算法的一些理论基础，包括共识算法在一致性、正确性、活性方面的要求。另外也了解像安全性、扩展性方面方面的要求

总目录

区块链共识简史



共识算法大事记

共识算法大事记

1982年 Lamport发表“拜占庭将军”论文
1985年 Fischer 和 Lynch 发表FLP定理
1988年Liskov 发表ViewStamped Replication 论文
1989年Lamport发表PAXOS论文
1999年Castro和Liskov发表PBFT论文
1999年Jakobsson和Juels发表PoW论文
2000年Eric Brewer提出CAP定理
2006年Mike Burrrows 发表Chubby论文
2009年中本聪实现比特币PoW
2012年PeerCoin实现PoS
2012年Ripple 实现基于UNL共识算法
2013年比特股发表DPoS论文
2014年Tendermint发表Tendermint共识算法
2014年Stellar提出SCP共识
2016年Intel发布锯齿湖项目PoET共识
2016年IBM Fabric 0.6 PBFT
2017年Fabric 1.0采用 Kafka Zookeeper

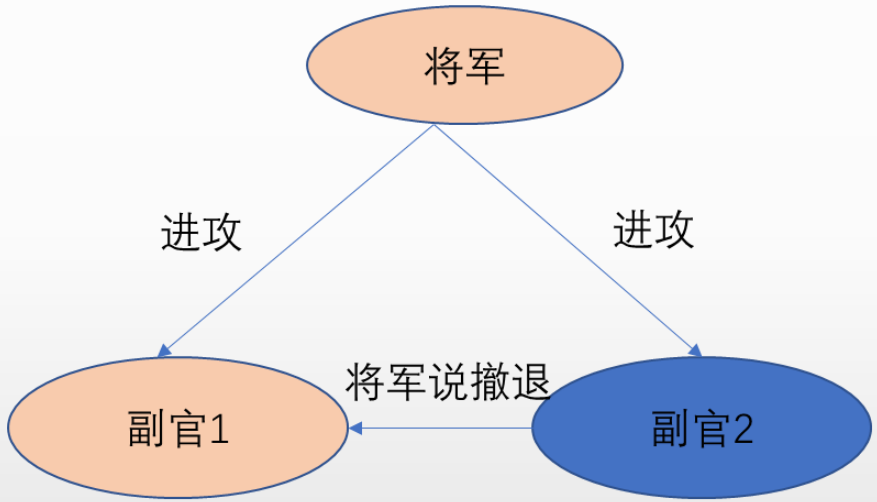
总目录

分布式系统中拜占庭将军问题（Lamport）

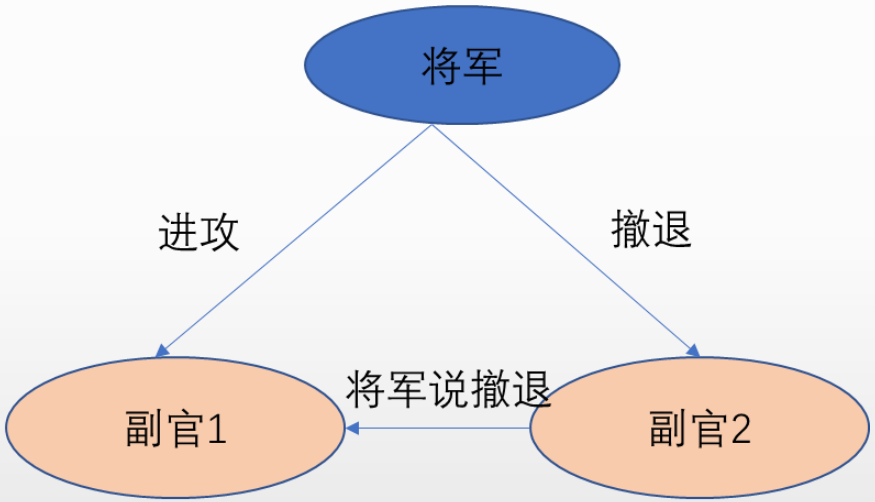
- **拜占庭将军问题**
  - 所有忠诚的将军决定一致的计划；
  - 少数叛徒不能使忠诚将军采用错误的计划。
- **简化版拜占庭问题**
  - 一个指挥官要发一个命令给n-1个副官，希望：
    - 所有忠诚的副官都执行同一命令；
    - 如果指挥官是忠诚的，每个忠诚的副官都执行该命令。

拜占庭将军问题

- 取决于忠诚将军占将军的总数
  - 在少于或等于2/3的忠诚将军情况下无解



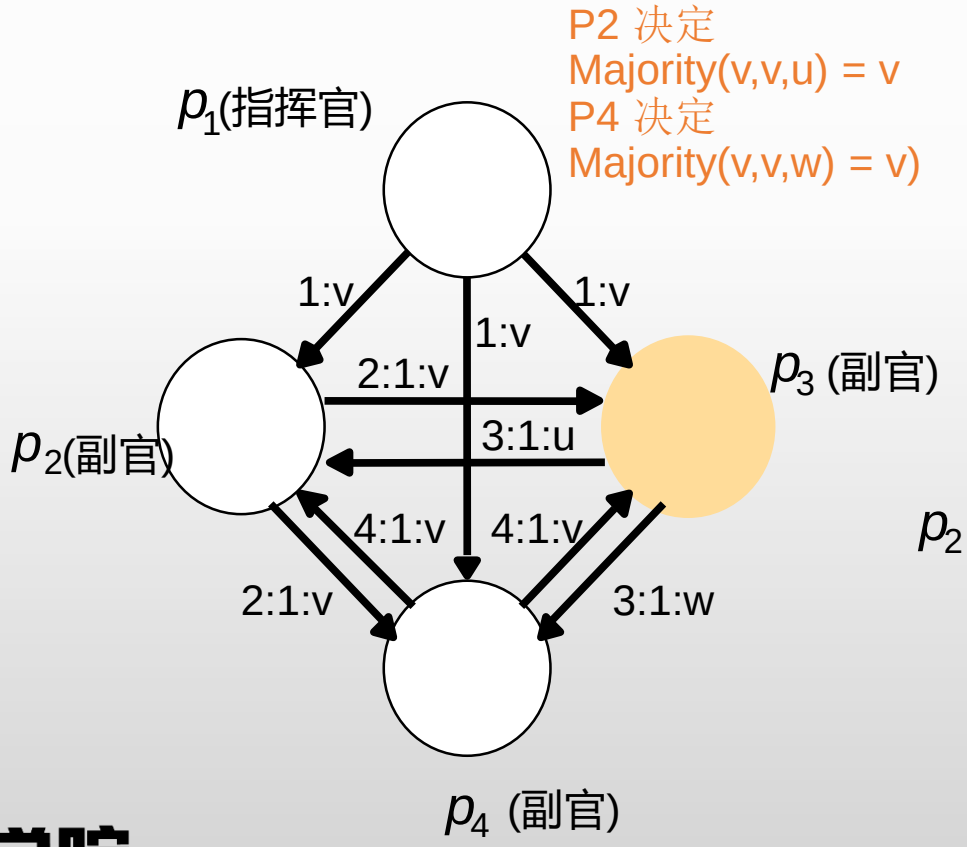
副官2是叛徒

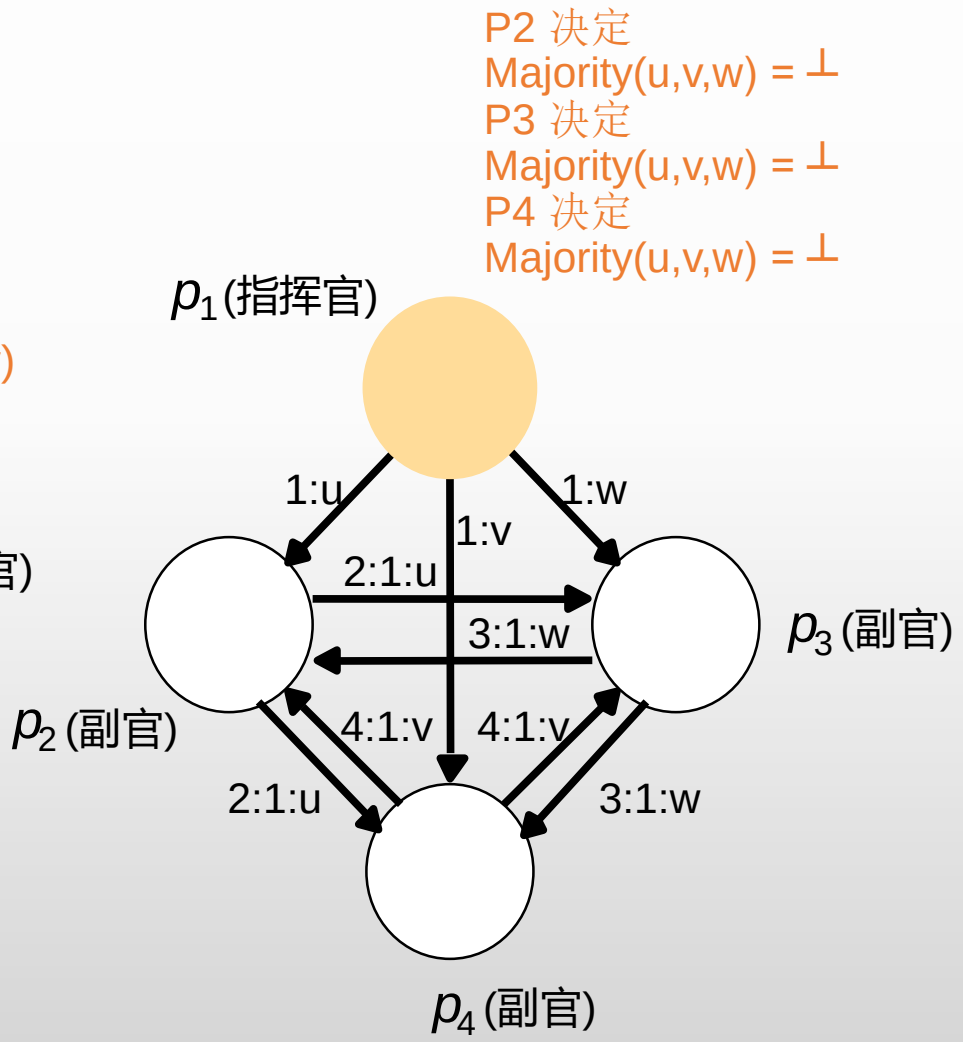


将军是叛徒

拜占庭将军算法（N=3f+1）

- 将军总数N，f是叛变将军数
- 两轮
  1. 指挥官发命令给副官
  2. 副官传递命令给别的副官





*黄色是故障节点

分布式系统与拜占庭将军问题

- **分布式系统节点类型**
  - 正常系统 – 忠诚的拜占庭将军
  - 任意故障系统 – 叛变的Byzantine将军
- **拜占庭缺陷（Byzantine Fault）**
  - 任何从不同观察者角度看表现出不同症状的缺陷
- **拜占庭故障（Byzantine Failure）**
  - 在需要共识的系统中由于拜占庭缺陷导致丧失系统服务
- **故障的分类**

极易混淆的与拜占庭将军相关的概念 - BA

- **拜占庭协议（Byzantine Agreement）**
  - **特征**
    - 单一来源，有一个初始值（Assume a system of n nodes, where a single source n_i has a private value v_i, ）
  - **属性**
    - 协议（Agreement）
      - 所有的非缺陷进程都必须同意同一个值。
    - 正确性（Validity）
      - 如果源进程是非缺陷的，那么所有非缺陷的进程所同意的值必须是源进程的初始值。(If n_i is non-faulty, then the agreed upon value by all non-faulty nodes is v_i.)
    - 可结束性（Termination）
      - 每个非缺陷的进程必须最终确定一个值。

极易混淆的与拜占庭将军相关的概念 - BC

- **拜占庭共识（Byzantine Consensus）**
  - **特征**
    - 所有进程都有一个初始值（Assume a system of n nodes, where each node n_i has a private value v_i, ）
  - **属性**
    - 协议（Agreement）
      - 所有的非缺陷进程都必须同意同一个值 V，$V \in (v_1; \ldots; v_n)$.。
    - 正确性（Validity）
      - 如果所有的非缺陷的进程有相同的初始值，那么所有非缺陷的进程所同意的值必须是同个初始值。(If all non-faulty nodes have the same initial value v, then the agreed upon value by all non-faulty nodes is v.)
    - 可结束性（Termination）
      - 每个非缺陷的进程必须最终确定一个值。

极易混淆的与拜占庭将军相关的概念 - IC

- **交互一致性（Interactive Consistency）**
  - **特征**
    - 所有进程都有一个初始值（Assume a system of n nodes, where each node n_i has a private value v_i）
  - **属性**
    - 协议（Agreement）
      - 所有的非缺陷进程都必须同意同一组值$A[v_1 \ldots v_n]$。
    - 正确性（Validity）
      - 如果进程 i 是非缺陷进程，那么它的初始值 v_i，必须被所有非缺陷进程同意为数组A中的第 i 个成员.
      - 如果进程 j 是缺陷进程，那么所有的非缺陷进程可以同意任意的$A[j]$值。
    - 可结束性（Termination）
      - 每个非缺陷的进程必须最终确定一个向量A的值。

三者关系（BA vs BC vs IC）

- 如何用解决BA问题的算法来解决IC的问题？
  - 运行BA N次，每次用其中一个进程做将军
- 如何用解决IC问题的算法来解决BC问题？
  - 如果大多数的进程是正常的流程，可以用解决IC的算法，然后再用多数函数来选取结果。
- 如何用BC的算法来解决BA的问题
  - 将军发提议值给他自己以及其他进程，然后每个运行BC

关于拜占庭将军问题的理论证明

- **Lamport**
  - 当叛徒多于将军总数的三分之一时，同时通信采用同步非防篡改方式，拜占庭将军问题无解。
  - 如果通信采用同步、认证和不可篡改方式，拜占庭将军问题可以在任意多叛徒情况下有解（如果将军总数少于叛徒数+2，问题就没有意义）
- **Fischer-Lynch-Paterson**
  - 在一个多进程异步系统中，只要有一个进程不可靠，那么就不存在一个deterministic（确定性）协议，此协议能保证有限时间内使所有进程达成一致。
  - 注意
    - 该异步系统指的是消息可以任意时间延迟，无从判断一个无响应进程是由于宕机还是消息延迟而造成

CAP定理 - Eric Brewer

- 用BASE 代替 ACID 来解决分布式系统的扩展性问题
  - Basic available, soft state, eventual consistency （基本可用，软状态，最终一致性）
- 一个分布式数据库系统有三个需要的属性
  - 对网络分隔的容忍（partition tolerance）
  - 一致性 (consistency)
  - 可用性 (availability)
- 在任意一个数据共享系统，最多只能在三个属性中选择满足两个属性。
- 证明
  - 首先假设所有的条件（consistency, availability, partition tolerance）能同时成立。
  - 因为任意网络至少两个节点可以分别处在两个非空、不重合的网络节点集合{G1，G2}。一个原子对象o有一个初始值 v0。我们定义α1是一个执行步骤，该步骤把在G1的o的值改成v1，v1 ≠v0。假设α1是唯一的一个用户请求。
  - 另外，我们假设G1和G2中的节点互相没有收到来自对方的消息。
  - 因为可用性的要求，我们知道α1是完成的，也就是说o现在在G1有v1的值。
  - 类似的，α2是一个在G2的一个单一的读o的执行步骤，在α2的执行过程中，G1和G2之间没有消息的传递。由于可用性的要求，我们知道α2也是完成的。
  - 如果我们执行α1和α2，G2只能看到α2（因为G2没有收到任何从G1来的消息，也看不到α1的请求。那么从α2读来的结果还是v0。
  - 但由于α2读的操作是在α1写的操作完成之后开始的，一致性的要求被违反。这就证明了我们不能同时保证三个属性都满足。

FLP与CAP定理关系

- 相同点
  - 都是在分布式系统中一些不能解决的问题
- 区别
  - FLP的假设是网络的异步通信是可靠的，消息虽然会有延迟，但不会丢失。CAP中的网络分区则会导致消息丢失。
  - FLP中的故障节点完全从网络中分隔，不会响应任何请求；而CAP中的可用性要求系统能响应请求。
  - FLP是关于分布式系统的共识问题，这个和CAP中的一致性有些相似，但CAP中主要是关于原子性数据存储的一致性，以及和可用性的关系。

分布式系统实际情况假设

- **系统假设**
  - 故障模型
    - 非拜占庭故障
    - 拜占庭故障
  - 通信类型
    - 同步
    - 半异步（延迟限制）
    - 异步（延迟无限制）
  - 通信网络连接
    - 节点间直连数
  - 信息发送者身份
    - 实名
    - 匿名
  - 通信通道稳定性
  - 消息认证性
    - 认证消息
    - 非认证消息

总目录

共识算法类型

- 工作量证明（POW）
- 权益证明（POS）
  - Peercoin，NXT，Waves
  - Tendermint， Casper
- 代议制权益证明（DPOS）
  - Bitshares， EOS
- 过去时间证明（POeT）
  - Intel Sawtooth Lake
- Paxos 类
  - Zookeeper, Chubby
  - RAFT
- BFT类
  - PBFT
  - ZAB
- UNL voting
  - Ripple
- 联邦式BFT
  - Stellar

区块链共识算法分类

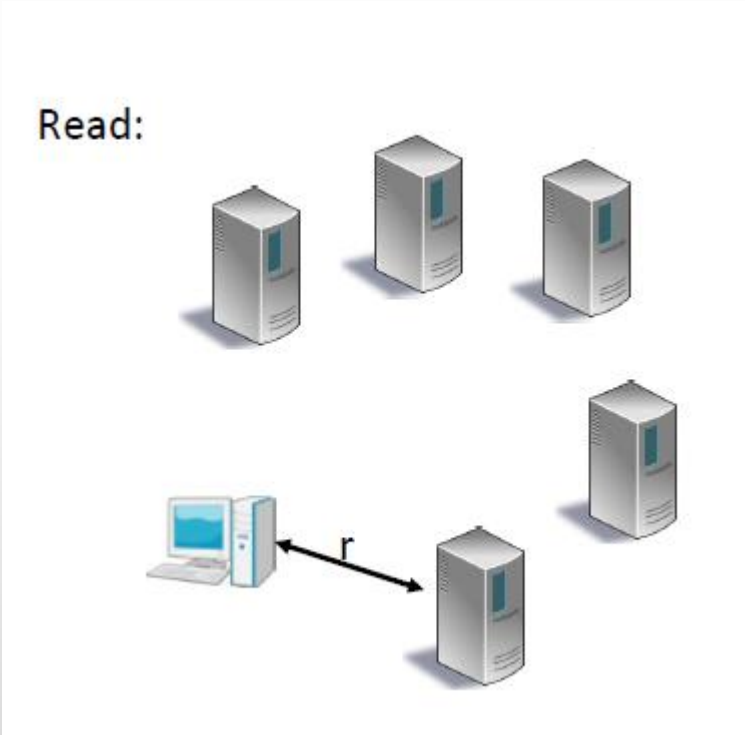
- 分类方式
  - 按共识机制分类
    - PoW, PoS, PoET, PoA
  - 按区块链部署模式
    - 公有链共识算法
    - 联盟链/私有链共识算法
  - 按容错类型分类
    - 宕机容错共识算法
    - 拜占庭容错共识算法
  - 按一致性类型分类
    - 强一致共识算法
    - 最终一致性共识算法
  - 按副本复制方式分类
    - Primary-backup 主从备份
    - State Machine Replication 状态机复制

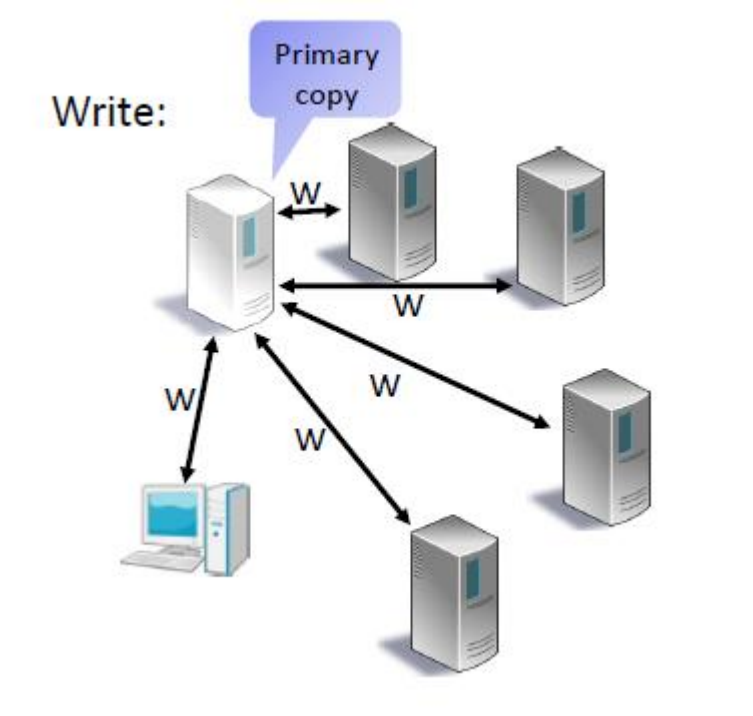
强一致性（Strong Consistency）共识算法

- **故障类型**
  - 宕机故障（Crash Failure）
    - 主副本, 2PC, 3PC提交
  - 宕机-恢复故障（Crash Recovery Failures）
    - Paxos, Chubby，ZooKeeper
    - RAFT
  - 拜占庭故障（Byzantine Failures）
    - PBFT（实用拜占庭容错），Zyzzyva

故障容错办法

- Replication – 主副本技术

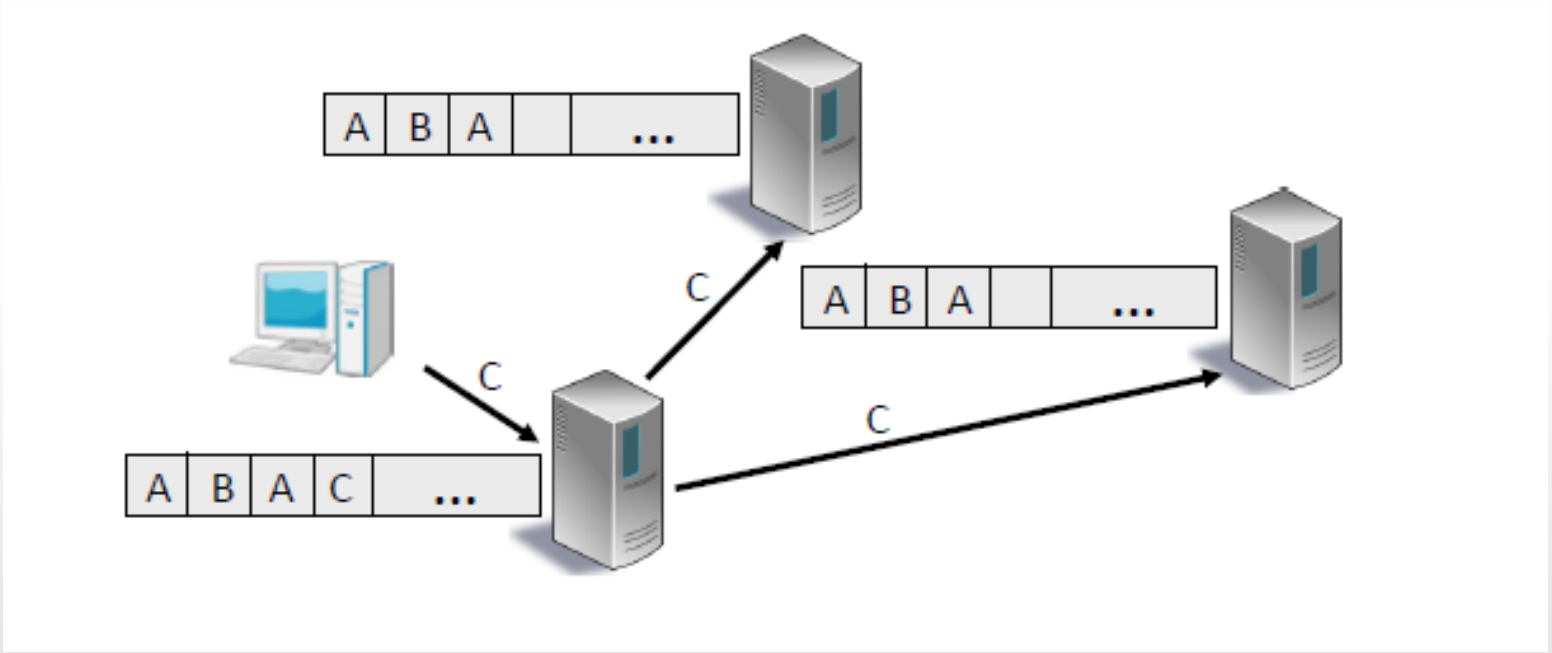




图片引自 Roger Wattenhofer ETH Zurich – Distributed Computing – www.disco.ethz.ch

如何保证一致性 – 共识机制

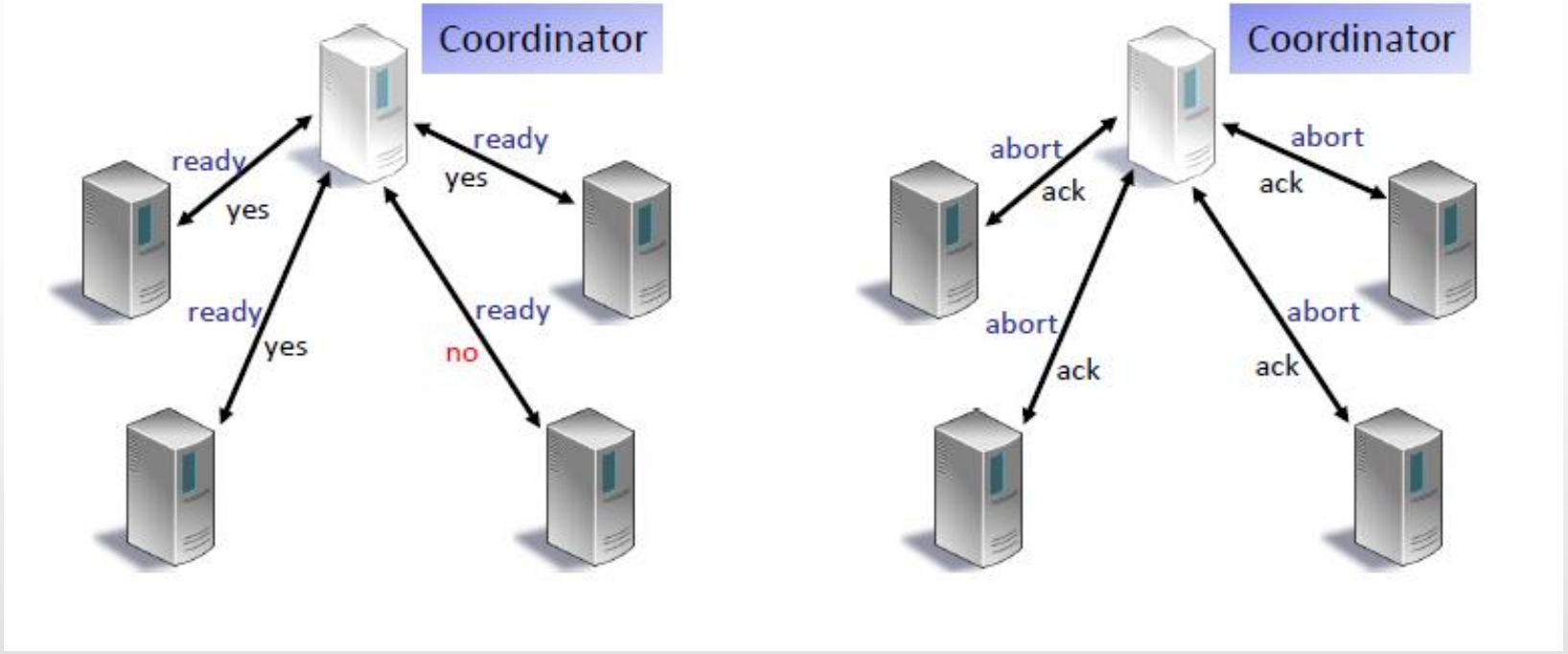
- 状态机复制



图片引自 Roger Wattenhofer ETH Zurich – Distributed Computing – www.disco.ethz.ch

如何保证在故障情况下的一致性？ - 交易的原子性

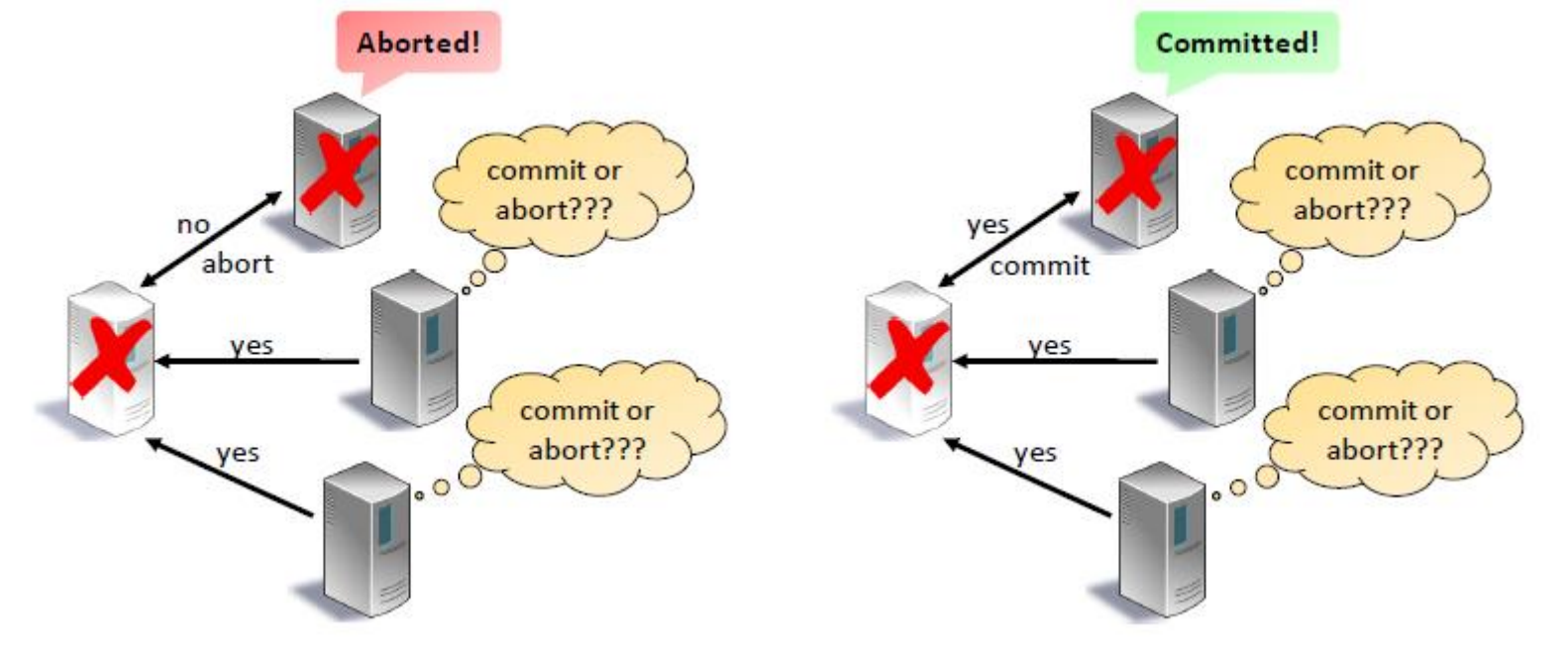
- 2PC 两阶段提交



图片引自 Roger Wattenhofer ETH Zurich – Distributed Computing – www.disco.ethz.ch

2PC两阶段提交不能解决多节点故障

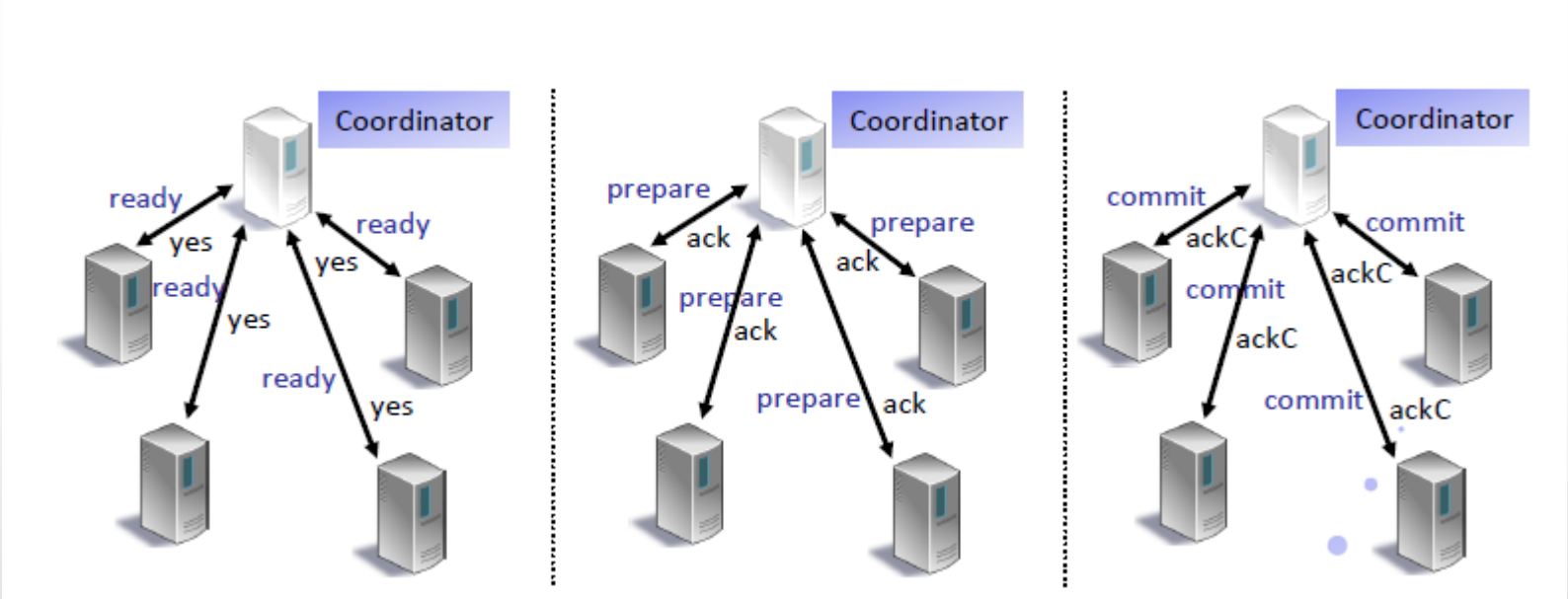
- 当Coordinator和其它一个节点发生故障，其它节点无法知道是应该提交还是回滚



图片引自 Roger Wattenhofer ETH Zurich – Distributed Computing – www.disco.ethz.ch

3PC提交可以避免多个节点故障情况

- 3PC提交
  - 把第二阶段再划分，从协议上使得第一阶段状态完全确定。
  - 其它节点第二阶段只能确认收到消息，不能提出回滚要求
  - 第三阶最后交易提交

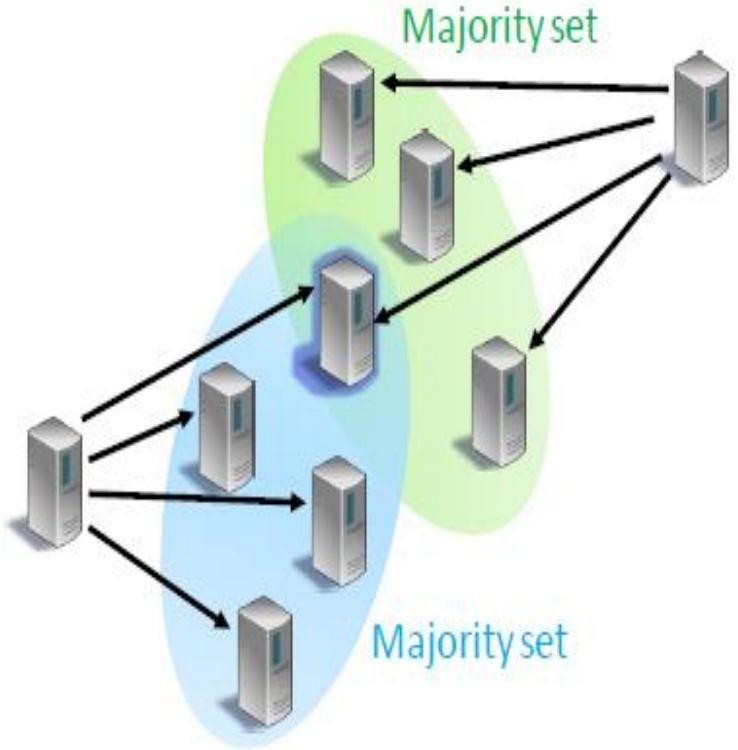


- 3PC的局限
  - 过于依赖协调节点，如果部分节点认为协调节点故障，将比较复杂
  - 如果一个Coordinator故障节点恢复上线（Crash Recovery），如何处理？

图片引自 Roger Wattenhofer ETH Zurich – Distributed Computing – www.disco.ethz.ch

Paxos – 避免对Coordinator的依赖，处理多节点Crash-Recovery 故障

- 没有协调者，只有多数集
- 三种角色
  - Proposer
    - 提议一个被希望接受的值
  - Accepter
    - 接受提议或拒绝提议
  - Learner
    - 不参与决策
    - 但必须知道最后结果



- Paxos 是一个两阶段协议，但比2PC有更强的故障恢复力
  - 没有Coordinator, 一个大多数的节点集被询问一个特定的值能否被接受。
- 询问大多数集就足够，因为两个大多数集之间的交集不是空集。如果一个大多数集选择一个值，没有其它多数集能选择另一值。

图片引自 Roger Wattenhofer ETH Zurich – Distributed Computing – www.disco.ethz.ch

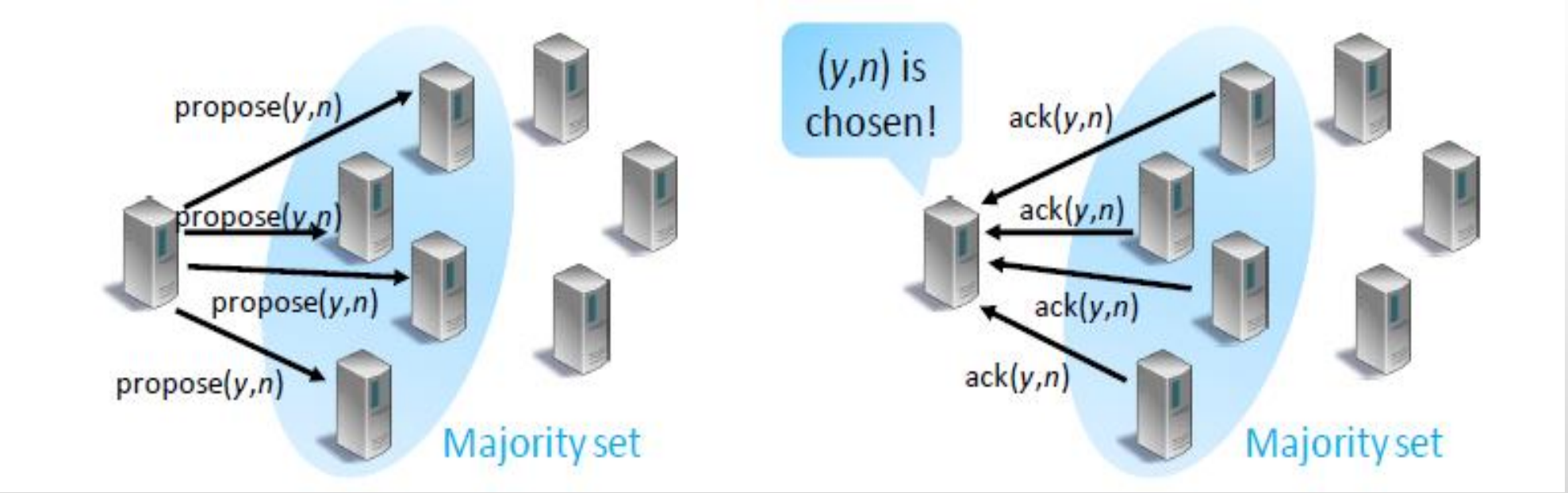
Paxos – Prepare 阶段

- Proposer 发 prepare（ x, n)给Accepters
  - 这个消息告诉对方它准备发提议 propose(x, n)
- Accepter 收到后，如果它接受过提议，而*n* 是它收到的请求中最大的数，它将承诺不接受比n小的提议。 Accepter将把过去收到的最大数提议（ y， m ）发回。
- 如果Accepter从没接受过提议，就发回(Ø,0)
- 如果Accepter不接受提议，可以不回答，或发回Nack
- Proposer知道被接受的提议。

图片引自 Roger Wattenhofer ETH Zurich – Distributed Computing – www.disco.ethz.ch

Paxos - Propose

- 如果Proposer收到所有答复，它将提出提议。
- 如果它只收到acc(Ø,0)，它将提出自己的提议值。
- 否则它将采用所收到的最大数的提议值 y。
- Proposer发出（y，n）
- 如果Proposer收到所有的确认ack(y,n)，该提议就被接受了。



图片引自 Roger Wattenhofer ETH Zurich – Distributed Computing – www.disco.ethz.ch

Paxos Pseudo-code

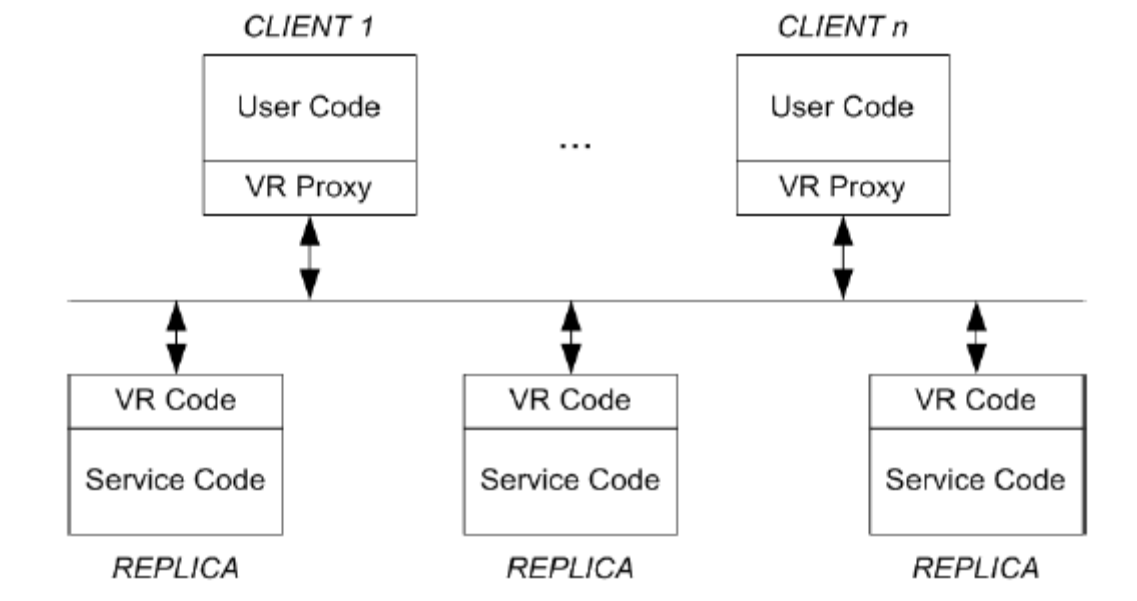
```
define runPaxos( allNodesThisPaxosInstance, myValue ) {
    allNodesThisPaxosInstance.sort()
    offset = allNodesThisPaxosInstance.indexOf( thisNode )
    for (i = 0; true; i++) {
        roundId = offset + i * allNodesThisPaxosInstance.size()
        prepareResult = doPreparePhase( roundId )

        if (!prepareResult.shouldContinue?)
            return

        if (prepareResult.hasAnyValue?)
           chosenValue = prepareResult.valueWithHighestRoundId
        else
            chosenValue = myValue

        acceptResult = doAcceptPhase( roundId, chosenValue )

        if (!acceptResult.shouldContinue?)
            return
    }
}
```

Paxos – 优点和局限

- 优点
  - Paxos是一个可以容忍 f<n/2 Crash-Recovery 故障的异步强一致性协议
- 局限
  - Paxos能保证协议性（Agreement）,正确性（Validity），但不能保证可终止性（Termination）
    - Paxos 只能保证如果一个值被选择，其它节点只能选择同意一个值。
    - 但它不能保证一个值能被选择到。

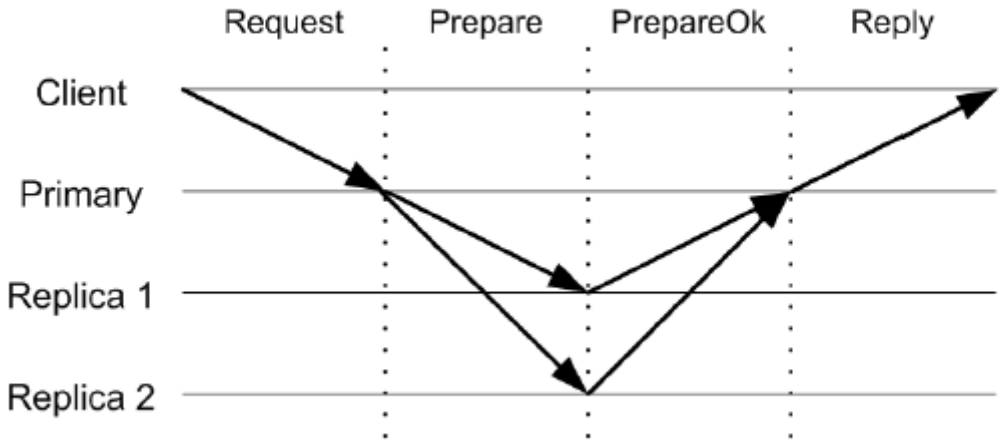
ViewStamped Replication 共识算法

- Viewstamped Replication（VR）
  - 1988年由Brian Oki在其导师Barbara Liskov指导下在其博士论文中发表的第一个分布式共识算法。
  - VR主要是一个复制协议，但它也能提供共识功能；
  - VR只处理宕机故障，VR的运行环境是一个异步网络环境，消息可能丢失，接收顺序可能不能保证，同一消息可能会重复接收。
  - VR的副本的总数设成2f+1个，VR保证在不超过f个副本出故障的情况下整个系统的可靠性和可用性
- VR架构



VR正常流程

- 客户端发一个请求（REQUEST）消息给主副本，让其执行某个操作，连同客户端id (*client_id*)和请求号 *request_number*.
- 主副本会检查*client_table*, 如果 *request_number* 小于在*client_ table* 表中的请求号，主副本会忽略该请求，并发送回复，如果该请求是最近被执行的一个请求。
- 主副本在*op-number*增加1，把请求加到日志*log* 并用新的请求号更新*client_table*。然后它发一个PREPARE消息给其它副本，连同当前的视图号 *view_number*, *op-number*, 客户的消息，和*commit_number* (最近提交的*op_number*)。
- 副本按顺序来接收消息，然后把消息加进日志，更新client_table并发一个PREPAREOK 的消息给主副本。这个消息表示该操作以及以前的操作都准备好了。
- 主副本等收到*f*个副本的回复之后才提交操作。它把*commit_number*加1. 当它确认所有在此之前的操作都执行后，它会调用服务代码执行当前的操作。一个REPLY的消息会发给客户端，包含有 *view_number, request_number* 以及服务代码返回的结果。
- 其它副本执行请求时，只要保证在此请求之前的请求都被执行，然后执行该请求，在*commit_number*上增加1，更新*client_table*, 但并不需要直接回复给客户端，只有主副本回复客户端。
- 如果一个客户端没有在一定时间内收到回复，它将重发请求给所有的副本。这样的话，如果VR已经转到一个新的视图，这个消息将最后到达新的主副本。副本会忽略客户请求，只有主副本处理客户请求。



VR异常处理和小结

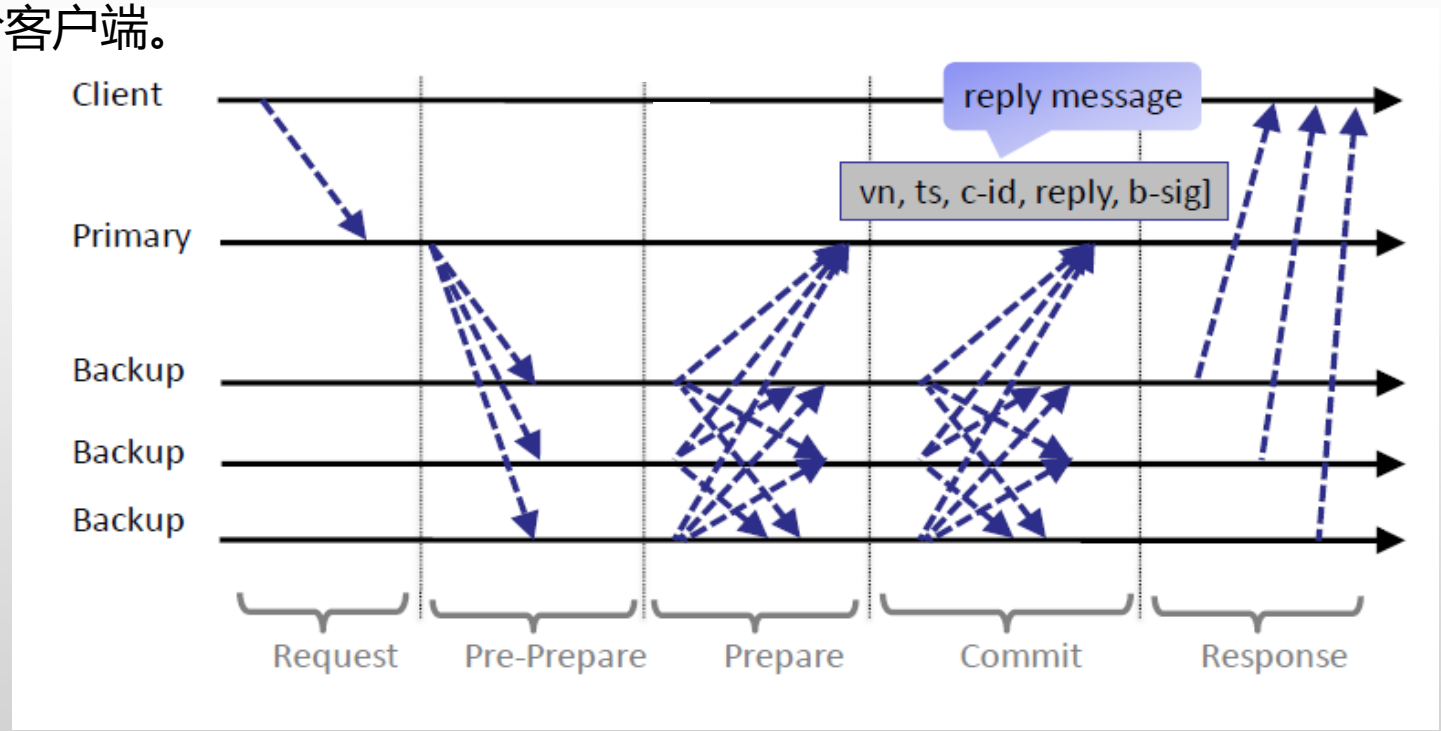
- 视图更换协议
  - 副本监控主副本，它们预期是间断性的能接到主副本消息。一般情况下主副本发“PREPARE”消息，但如果是空闲时段（没有客户请求），它发COMMIT消息给副本。如果在一个超时发生而没有收到主副本消息，副本会执行一个视图转换以选择新的主副本。
- VR副本节点故障恢复协议
  - 当一个副本从宕机恢复时，在它恢复到宕机之前的状态前，它不能参与处理请求和视图转换。否则系统就会出错。
  - VR副本需要记住它所执行的操作。VR副本节点的状态是保存在别的副本节点上，不需要通过磁盘存储来保持状态。VR副本的状态可以通过恢复协议来获得。
- VR小结
  - VR是一个异步通信环境的共识协议
  - 在FLP的异步通信假设中，没有一个同步的时钟可以访问，也就是说不能依靠超时来做判断。最后，也假设没法通过监控来判断进程是否出现故障。
  - VR则可以依靠超时来判断一个节点是否出故障。

PBFT – 实用拜占庭容错

- PBFT 需要5轮通信步骤
  - 第一轮
    - 客户端发命令 op 给主节点
  - 第二轮
    - Pre-prepare
  - 第三轮
    - Prepare
  - 第四轮
    - Propose
  - 第五轮
    - 客户端接收服务端的回复
      - 如果 *f*+1 (authenticated) 回复一样，这个结果被接受。
      - 因为只有 f 个拜占庭服务器，至少有一个正确的服务器支持这个结果。

PBFT – 协议流程

- 客户端发op给主节点，包括时间戳 ts，客户端id c-id和客户签名 c-sig
- 在第二轮，主节点把消息发给所有备份m = [op, ts, c-id, c-sig] ，包括视图号码 vn，分配的顺序号码 sn，消息m的digest D(m), 及其签名p-sig
- 第三轮，如果一个备份 b接受了 pre-prepare消息，它将发[P, vn ,sn, D(m), b-id, b-sig] 给其它备份，并将Prepare 消息存在log中
- 如果一个备份 b 有消息 m, 一个接受的 pre-prepare 消息，和2f接受的准备消息，它将[C, vn, sn, D(m), b-id, b-sig] 群发到其它备份服务器，并存储commit 消息
- 备份返回回复消息给客户端。



图片引自 Roger Wattenhofer ETH Zurich – Distributed Computing – www.disco.ethz.ch

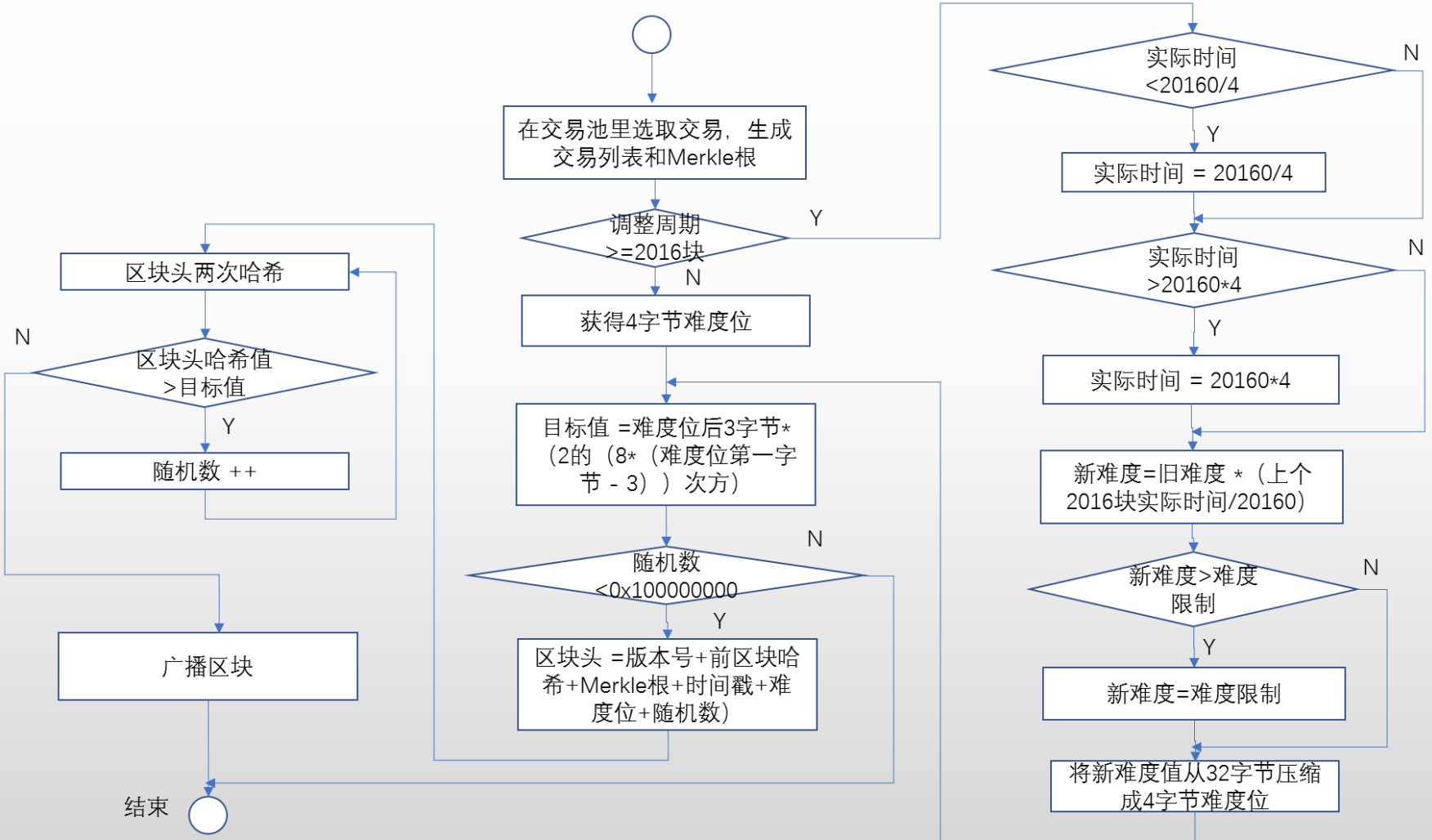
PBFT – 强一致性共识算法

- PBFT在一个异步系统中能达成共识，容忍f<n/3拜占庭故障
- PBFT的异步通信是一个有固定延迟限制的异步通信系统（接近同步通信）
- PBFT采用认证的消息
  - 可以验证服务器是否发过一个消息
- 在最坏情况下，PBFT需要用f轮才能达到共识。

首个区块链共识—公有链比特币PoW机制

- 比特币PoW – 最终一致性共识
  - **挖矿过程**
    - 生成铸币交易，并与其他所有准备打包进区块的交易组成交易列表，通过Merkle树算法生成Merkle 根哈希；
    - 把Merkle根哈希及其他相关字段组装成区块头，将区块头的80字节数据作为工作量证明的输入；
    - 不停的变更区块头中的随机数即nonce的数值，并对每次变更后的区块头做双重SHA256运算（即SHA256(SHA256(Block_Header))）
    - 将结果值与当前网络的难度目标值做对比，如果小于目标值，工作量证明完成
  - **难度调整**
    - 由于矿工数量动态变化，比特币系统动态调整挖矿难度，使得大约每10分钟产生一个区块
      - 新难度值=旧难度值*(过去2016个区块花费时长/20160 分钟)
      - 目标值 = 难度位后三字节 * （2^(8*(难度位第一字节 - 3））)
  - **共识区块链**
    - 最长区块链为共识区块链，代表具有最大工作量的区块链
  - **概率性拜占庭 （Probabilistic BA）**
    - **协议（Agreement）**
      - 在不诚实节点总算力小于50%的情况下，同时每轮同步区块生成的几率很少的情况下，诚实的节点具有相同的区块的概率很高。
    - **正确性（Validity）**
      - 大多数的区块必须由诚实节点提供
      - （严格的说，当不诚实算力非常小的时候，才能使大多数区块由诚实节点提供）
    - **可终止性（Termination）**
      - 约每10分钟完成一次共识

比特币POW过程



比特币POW共识算法的正式分析 – Garay （2015）

- **概率性拜占庭协议（Probabilistic BA）**
  - **协议（Agreement）**
    - 在不诚实节点总算力小于50%的情况下，同时每轮同步区块生成的几率很少的情况下，诚实的节点具有相同的区块的概率很高。
    - （严格说应该是：当任意两个诚实节点的本地链条截取K个节点，两条剩下的链条的头区块不相同的概率随着K的增加呈指数型递减）
  - **正确性（Validity）**
    - 大多数的区块必须由诚实节点提供
    - （严格的说，当不诚实算力非常小的时候，才能使大多数区块由诚实节点提供）

以太坊1.0 PoW共识算法

- **Ethereum Ethash PoW**
  - Ethash （ Dagger-Hashimoto 修改版本）
    - 挖矿不但需要计算能力，还需要内存，对ASIC挖矿和算力中心化有一定抵抗力
    - 流程
      - 扫描所有区块的区块头，可以为每一个区块生成一个种子；
      - 从种子可以生成一个16M的伪随机缓存，轻节点存储该缓存；
      - 从缓存可以生成一个1GB的数据集（DAG），该数据集中每条记录只依赖于在缓存中的小量记录。全节点和挖矿节点保存该数据集，数据集随着时间线性增长。
      - 挖矿工作是随机的获取数据集中的少量记录并做哈希运算。验证只需要从缓存中生成该少量记录，不需要生成DAG，因此验证只需要存储缓存。
      - 该数据集每30000个块（100小时窗口，叫epoch）更新一次，由于和高度有关，因此可以预生成
      - 采用SHA3_256(Keccak-256)和SHA3_512(Keccak-512)哈希算法
  - 难度动态调整，平均每12秒出块。
  - 比特币PoW基础上引入GHOST（Greedy Heavest-Observed Sub-Tree), 在共识区块链计入叔区块

公有链其它PoW共识算法

- **PrimeCoin PoW**
  - 工作量证明不像比特币那样做无用功，而是寻找质数，具有科学价值，可称为“有用工作量证明”（Proof of Useful Work”）
    - 寻找*Cunningham chain* 质数
      - 一个质数的2倍+1仍然是质数：2, 5, 11, 23, 47
  - 一些最大的质数是通过PrimeCoin的工作量证明找到的
- **工作量证明机制**
  - 对一个前区块的哈希值x，取前m位，寻找一个对任何k 长度的Cunningham chain， 第一个质数是一个n位的质数，同时和 x有相同的前m位。
  - 可以调整n和k来使得难度加大
    - 增加k的长度，使得难度指数型增加，
    - 增加n的长度，难度线性增长
    - M只要足够大，可以在没有得到前区块之前做一些预运算来判别可行性

LiteCoin PoW

- LiteCoin – Scrypt
  - 采用scrypt算法，不但需要计算能力，还需要内存，可以防止暴力破解，对ASIC挖矿也有一定的抵抗力
  - 伪代码

```
1   def scrypt(N, seed):
2       V = [0] * N // initialize memory buffer of length N
        // Fill up memory buffer with pseudorandom data
3       V[0] = seed
4       for i = 1 to N:
5       V[i] = SHA-256(V[i-1])
        // Access memory buffer in a pseudorandom order
6       X = SHA-256(V[N-1])
7       for i = 1 to N:
8           j = X % N // Choose a random index based on X
9           X = SHA-256(X ^ V[j]) // Update X based on this index
10      return X
```

Proof-of-Elapsed Time – Intel 锯齿湖项目

- **Intel SawtoothLake PoET**
  - 共识与交易隔离
  - 采用新的CPU安全指令，通过可信任运行环境（TEE）如 Intel® Software Guard Extensions (SGX) ，根据验证者等待时间来确定Leader，实现公平、随机选取共识Leader
    - Enclave
      - 一个应用内存地址隔离的代码和数据区域
      - Enclave里的数据只能被同一个Enclave的代码访问
    - 每个验证者向一个enclave 请求一个等待时间，具有对一个区块最短等待时间的验证者会被选成Leader.
    - 函数CreateTimer
      - 建立一个出块定时器
    - 函数CheckTimer
      - 验证定时器由一个enclave建立，同时验证共识参与者等待了一定时间才获得出块权
  - 实现“一个CPU一票”

工作量证明机制 - 工作量证明要求的属性

- 难以计算
  - 没有快捷的方法去寻找答案，只能遍历所有的可能性
- 容易验证
  - 验证答案比较简单、成本低
- 难度可调
  - 当新矿工加入或离开，算力增减的时候可以动态调节，防止攻击又能保证出块的连续性和节奏
- Progress-Freeness（非按部就班）
  - 按算力比例的机会出块，算力小的也有机会出块，而非完全没机会
- 抗ASIC
  - 避免挖矿由专门设备垄断而出现中心化趋势
- 投资收益
  - 工作量的投资收益大于成本

权益证明共识算法 – POS

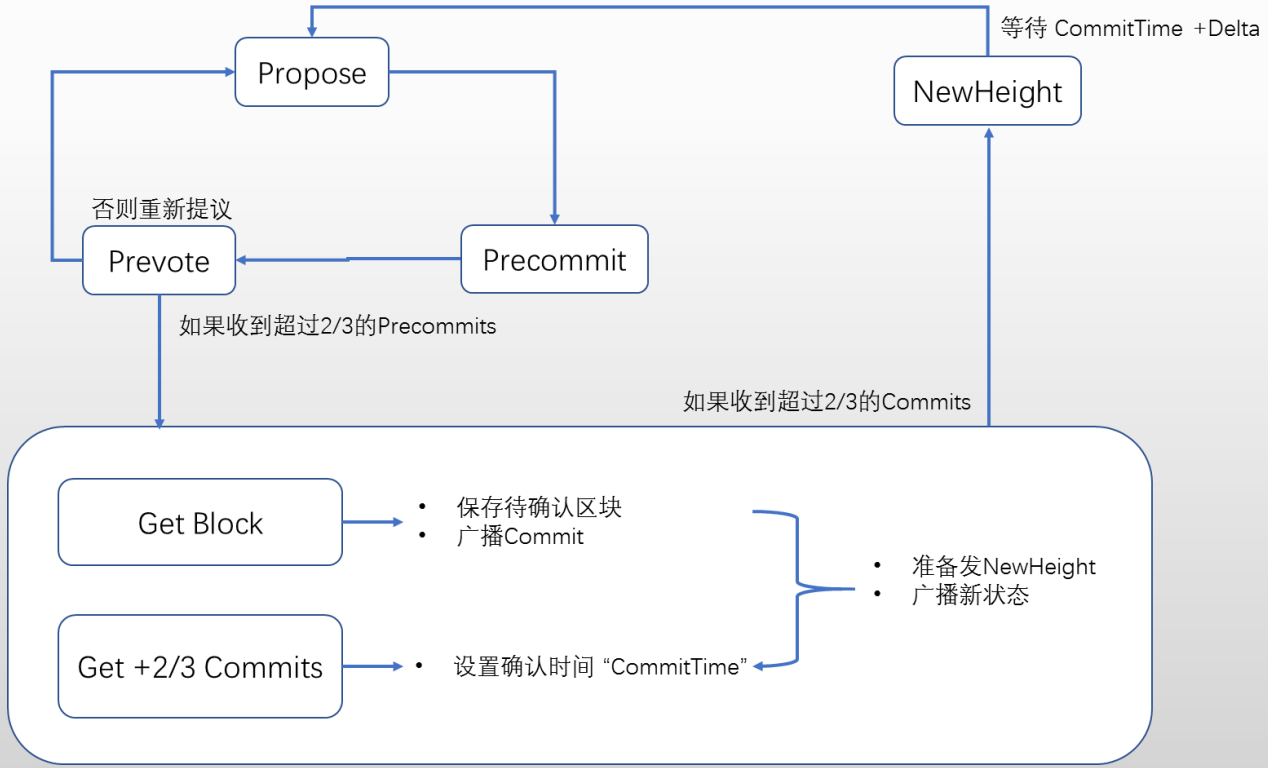
- **权益证明Proof-of-Stake (PoS)共识算法**
  - **PeerCoin POS**
    - 结合POW和POS
      - CoinBase 和 CoinStake区块
    - 权益证明机制结合了随机化与币龄的概念，未使用至少三十天的币可以参与竞争下一区块，越久和越大的币集有更大的可能去签名下一区块。
    - 寻找下一区块的最大概率在90天后达到最大值
    - 工作量证明机制的目标值和权益证明机制的目标值是不停的调整的
    - 最大的消耗币龄的链条被选为共识区块链
  - **NXT、Blackcoin POS**
    - 采用随机方法预测下一合法区块，使用公式查找与权益大小结合的最小哈希值， 由于权益公开，每个节点都可以以合理地准确度预计哪个帐户有权建立区块。

权益证明共识算法 – LPOS， DPOS

- **Waves**
  - 传统的PoS机制，只有小份额的权益人只有非常小的机会获得出块记账机会
  - Waves引入Lease Proof-pf-Stake (LPOS)
    - 可以将币租给其它节点来参与POS共识
    - 租借的资金还是由权益人控制，同时在租借到期后可以花掉或者转走。出块后的收益由租借人和出块人共同分享
- **BitShares DPOS**
  - DPoS利用权益人投票的权力来公平民主的解决共识问题。所有的网络参数，从交易费用，生成块的时间以及交易大小，都可以通过选出来的代理人来调整。
  - 引入了见证人这个概念，见证人可以生成区块，每一个持有比特股的人都可以投票选举见证人。
  - 确定性的选择出块人可以平均一秒的速率确认交易
  - 见证人的候选名单每个维护周期（1天）更新一次。见证人然后随机排列，每个见证人按序有两秒的权限时间生成区块，若见证人在给定的时间片不能生成区块，区块生成权限交给下一个时间片对应的见证人。

BFT POS 共识算法 - Tendermint

- **Tendermint**
  - 验证者将押金锁定，投票权力和押金相等。验证者如果作弊，押金会被销毁。
  - 投票步骤：Propose, Prevote, Precommit， Commit, NewHeight
  - Precommit和Commit需超过2/3投票

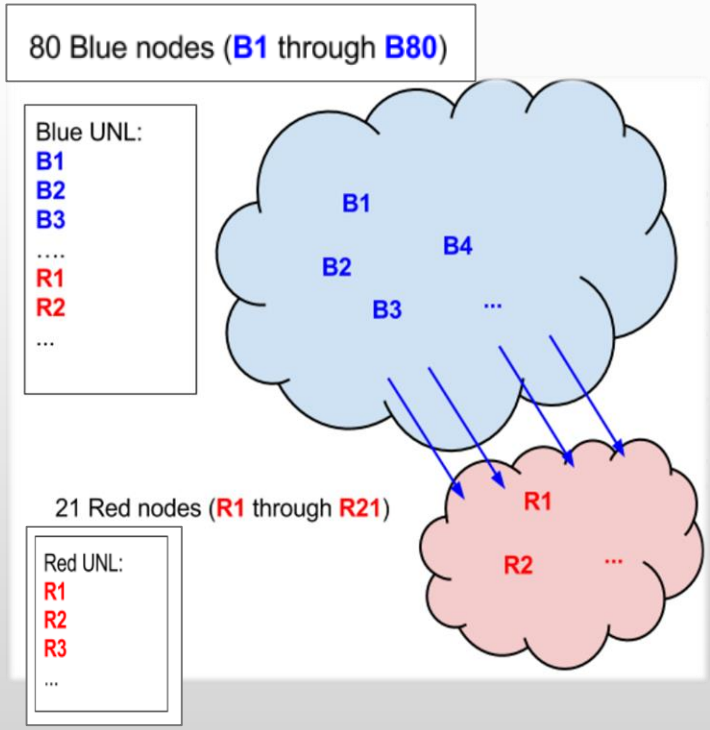


Ethereum PoS（Casper）

- **Ethereum PoS (Casper)**
  - 第一版本
    - PoW和PoS混合
      - 继续使用PoW来验证绝大多数区块，POS将用来在每100个区块做“Checkpoint”也就是永久确认。这个给以太坊区块链带来交易确认的最终性。
    - 权益持有者通过用押金方式来赌下一个区块，赌中有奖，不中会扣掉一部分押金
  - 过程
    - 验证者广播用自己私钥签名投票消息，投票消息中包含4个信息，其中两个是checkpoint s 和 t, s是源checkpoint, 指的是被确认过的checkpoint，t是目标确认checkpoint区块。另外两个是s和t所对应的高度 h(s) 和 h(t)。这里的高度指的是从创世纪区块开始顺着链下来100个区块的倍数。因为Casper只是用来验证并最终确认每100个区块（checkpoint），因此这里的高度是指距离创世区块100的倍数。
    - “苛刻条件”（Slashing Conditions），
      - 防止不兼容的两个区块同时被确认，同时防止验证者同时投票确认两个区块。
      - 假设验证者广播两个投票消息： s1; t1; h(s1); h(t1) 和 s2; t2; h(s2); h(t2)。苛刻条件主要有两个条件：
      - 第一个是 $h(t1) \neq h(t2)$，也就是说，一个验证者不能发布两个不同的投票信息，其中的两个checkpoint的高度一样。
      - 第二个是验证者不能发布一个嵌套于另一个checkpoint的checkpoint, 也就是说不能发布两个满足下面条件的验证消息，使得：$h(s1) < h(s2) < h(t2) < h(t1)$。违反苛刻条件的验证者会被没收押金。

其它共识算法 – Ripple 高扩展性BFT算法

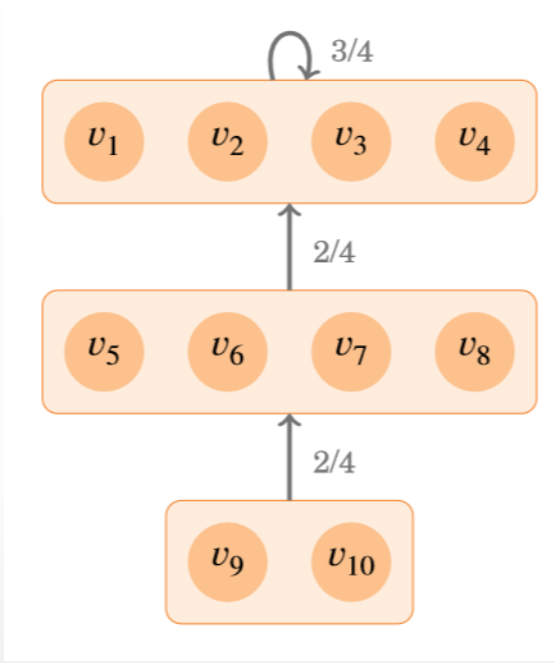
- **Ripple用子网共识提升共识效率，但能保证一致性吗？**
  - Ripple共识
    - 假设拜占庭节点数少于所有节点数的20%（f <= (n-1)/5）
    - 共识基于有信任关系的独一节点群UNL（Unique Node List)
    - UNL 的任意一个节点串通作恶概率小于20%
    - 任意两个UNL间重合的节点数至少占最大UNL节点数20%
    - 验证的交易需获得80%以上UNL成员投票
  - 极端情况下Ripple能保证不分叉吗？
    - UNL1: 80 蓝 + 21红
      - 80个节点，都投Yes票
      - 3个红节点投Yes票
      - 共识结果是Yes > 80
    - UNL2：21红
      - 21个节点，其中18个投No
      - 共识结果是No >80%
    - UNL 1和 UNL2 共识结果分叉

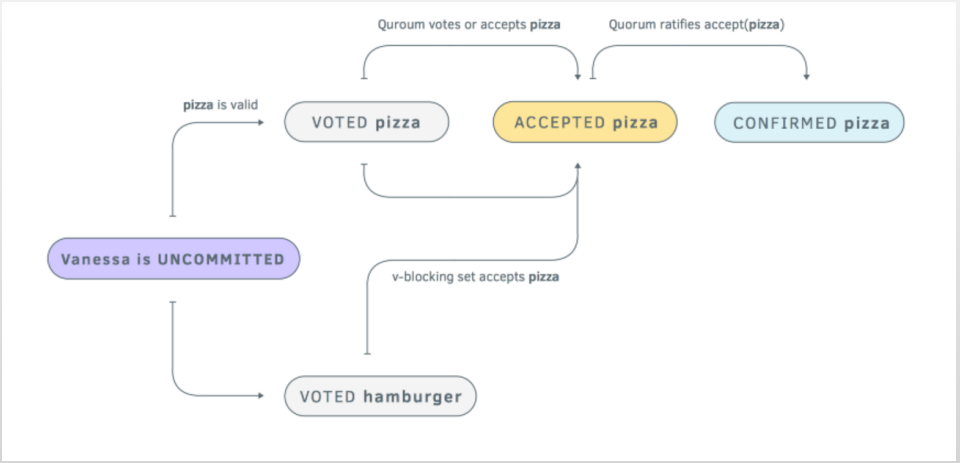


*source: Ripple Forum:
https://forum.ripple.com/viewtopic.php?f=2&t=7801

其它共识算法 – Stellar联邦式BFT

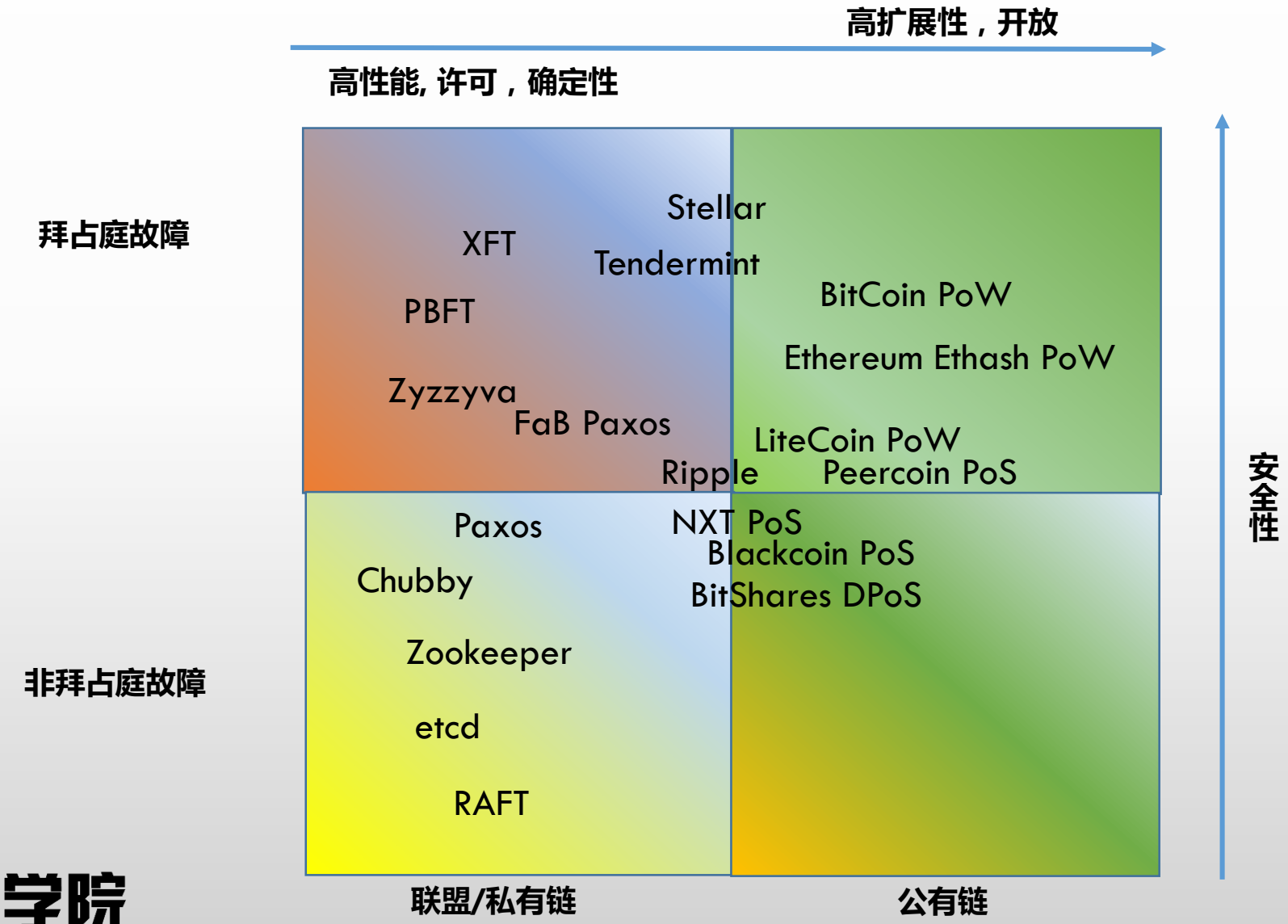
- **Stellar联邦式BFT共识算法**
  - SCP共识
    - 特点 – 分散控制、低延迟、灵活信任、渐近安全
      - 开放成员，每个节点可以决定信任对象 （ quorum slices)
    - 共识条件
      - Quorum
        - 达成一致的多数集
      - Quorum Slice
        - Quorum中的子集，能说服一个节点达成一致
      - Quorum Intersection 条件
        - 任意一个节点的Quorum Slices函数必须满足任意两个Quorum之间必须有一个共同的节点的条件
      - 删除破坏节点后还能达到Quorum Intersection
    - SCP协议 – 提名协议（ Nomination Protocol ）
      - 生成候选值
    - SCP协议 – 投票协议 （ Ballot Protocol)
      - Prepare （ 准备 ）
      - Confirm （ 确认 ）
      - Externalize （ 外部化 ）





*source：Stellar White Paper and Technical Summary

区块链共识现状 – 多数单一场景，受限于FLP，CAP



总目录

区块链共识算法的选择考虑

- 公有链
  - 最终一致性（Eventually Consistency）
  - 故障类型: 拜占庭故障
  - 共识成本
    - POW vs POS
  - 共识效率
  - 安全性
- 联盟链 / 私有链
  - 强一致性（Strong Consistency）
  - 故障类型
    - 非拜占庭故障
    - 拜占庭故障
  - 共识效率

私有链上共识算法存在问题

- **私有链多数采用状态机复制技术（SMR）**
  - Paxos-based
    - Chubby, Zookeeper, etcd, RAFT
  - BFT-based
    - PBFT
    - Zyzzyva
- **SMR技术扩展性不好**
  - 成员固定
  - 一般记账节点10-20个
  - 多轮消息往返
    - PBFT：五阶段， 节点平方级的消息量
  - 节点增加，性能降低

公有链上共识算法存在问题

- **比特币– POW机制**
  - 能源消耗大，工作量没有实际价值
  - 交易速度慢
    - 1秒钟最多7笔交易
  - 算力集中
    - 矿池
    - ASIC矿机
- **POS 机制**
  - 破坏者攻击成本小，安全性成疑
  - 公平性有问题
- **不能保证交易时序性**
  - 不能保证区块链中的交易顺序
- **有用工作量机制 – Proof-of-useful Work**
  - Primecoin
  - 如何满足容易验证，难度可调，几率和贡献工作量成正比（Progress-free）

区块链主要技术核心 – 共识机制

- **共识算法属性**
  - 一致性（Agreement）
  - 正确性（Validity）
  - 活性（Liveness）
  - 性能（Performance）
  - 扩展性（Scalability）
  - 公平性（Fairness）
  - 安全性（Security）
- **共识算法的理论限制**
  - Fischer-Lynch-Paterson定律
    - 在一个多进程异步系统中，只要有一个进程不可靠，那么就不存在一个协议，此协议能保证有限时间内使所有进程达成一致。
- **实用共识算法**
    - 实际情况下假设不同的条件
      - 同步/异步
      - 身份认证/匿名
      - 宕机-恢复故障 / 拜占庭故障
      - 故障节点占比

小结

- 区块链是信任机器
  - 信任由共识产生
- 共识算法有强一致性和最终一致性共识算法
  - 强一致性
    - Paxos，PBFT
  - 最终一致性
    - PoW， PoS，DPoS
- 区块链共识算法因公有链、联盟链、私有链不同而有所不同
  - 公有链：最终一致性共识算法
  - 联盟链，私有链
    - 强一致性共识算法

区块链共识算法发展趋势

- **发展趋势 – 融合**
  - 未来区块链平台支持可插拔共识模块
    - Hyperledger Fabric, Ethereum， Hyperledger Sawtooth Lake
    - 根据不同场景，选择合适的共识算法
      - Fabric1.0， 提升交易吞吐率，采用Kafka
  - 未来更多的融合，取长补短
    - 多链，侧链，闪电网络
    - 公有链/联盟链融合
    - 去信任 + 信任
      - Ripple, Stellar
  - 行业区块链
    - R3CEV Corda -专注于金融行业，（不是区块链的区块链DLT）
      - 数据只在相关方共享，不交给无关第三方
      - 交易正确性共识
        - 通过合约运行和签名校验来验证
      - 交易唯一性共识
        - 需要事先设置的独立公证

技术成就梦想