区块链技术与应用

计算机科学与技术学院　李京

04章 区块链共识层

目 录

背景

电子货币要解决的问题

如何在一个去中心化的环境中，达成共识，进行安全又隐私的交易？

背景

在分布式网络中如何达成一致

共识

算法

在任何时候都能有相同的计算结果

4.1分布式系统模型与共识

- 分布式系统模型
- 拜占庭将军问题
- 共识算法的分类
- FLP定理和CAP定理
- 共识过程的主流模型
- 共识简史

1. 分布式系统模型-分布式系统及其特点

- 分布式系统是组件分布在网络计算机上且通过消息传递进行通信和行为协调的系统。对外呈现为一个完美的、可扩展的“虚拟节点”，相对单物理节点具备更优越的性能和稳定性。
- 分布式系统的特征：
  - 并发性
  - 缺乏全局时钟，没有一个全局正确的时间来协调各组件的行为
  - 组件故障的独立性

分布式系统的系统模型（System Models）

- 结构模型（Architectural model）
  - 构成系统各部分组件的位置、角色和它们之间的关系，定义了系统的各组件之间相互交互的方式以及它们映射到下面的计算机网络的方式。一般为：客户/服务器结构、对等结构（客户/服务器模型的变种）。
- 基础模型（Fundamental model）
  - 对体系结构模型中公共属性的一种更为形式化的描述，包括：交互模型、故障模型和安全模型。

分布式系统基础模型-交互模型（时序模型）

- 交互模型
  - 进程之间通过消息传递进行交互，实现系统的通信和协作功能。
    - 有较长时间的延迟。
    - 时间是进程间进行协调的基本的参照，在分布式系统中，很难有相同的时间概念。
  - 独立进程之间相互配合的准确性受限于上面两个因素。
- 同步分布式系统和异步分布式系统

同步系统

- 进程执行每一步的时间都有明确的上限和下限。
- 每一条消息会在已知的时间范围内确定被接收到。
- 本地时钟与实际时间的漂移率也在已知范围内。
- 可以根据超时(Timeout)来检测进程的故障。
- 实际系统很少有真正的同步系统，同步模型主要方便进行理论分析和测试。

异步系统

- 对进程执行速度、消息传递延迟和时钟漂移率都没有限制。
- 实际的分布式系统大多数是异步系统。

部分同步系统

- 对系统执行时间有一定信息，但不一定准确。
- 与完全同步和完全异步系统相比，部分同步系统理论还不完善。

分布式系统基础模型-故障模型

- 故障模型
  - 计算机或者网络发生故障，会影响服务的正确性。
  - 故障模型定义可能出现的故障形式，为分析故障带来的影响提供依据。
  - 设计系统时，知道应如何考虑容错的需求。

分布式系统基础模型-故障类型

崩溃故障

- 节点正确运行直至崩溃。
- 节点崩溃后不可恢复，如果其他节点可以检测到这种故障则称为“故障-停止”，否则称为“崩溃”。
- 节点崩溃后可以恢复运行，则称为“故障-恢复”。

遗漏故障

- 节点或信道未能执行本来的动作导致消息丢失。
- 根据消息丢失发生的所在步骤位置分为发送遗漏故障、信道遗漏故障和接收遗漏故障。

时序故障

- 节点过早或过迟的响应，仅适用于同步系统假设。
- 分为时钟故障、节点性能故障和信道性能故障。

拜占庭故障

- 亦称随机故障，指节点可能任意地、错误地，甚至恶意地执行某些未经许可的动作。

分布式系统基础模型-安全模型

- 分布式系统的模块特性以及开放性，使得它们暴露在内部和外部的攻击之下。
- 安全模型的目的是提供依据，以此分析系统可能收到的侵害，并在设计系统时防止这些侵害的发生。

分布式一致性（Consistency）

- 分布式一致性是分布式计算的核心问题。
  - 定义一：指多个节点对某一变量的取值达成一致，一旦达成一致，则变量的本次取值即被确定。
  - 定义二：指分布式系统中的多个服务节点，给定一系列的操作，在约定协议的保障下，使它们对外界呈现的状态是一致的。换句话说，也就是保证集群中所有服务节点中的数据完全相同并且能够对某个提案（Proposal）达成一致。
- 一致性分类：
  - 强一致性：当分布式系统中更新操作完成之后，任何多个进程或线程，访问系统都会获得最新的值。
  - 弱一致性：指系统并不保证后续进程或线程的访问都会返回最新的更新的值。
  - 最终一致性：最终一致性是弱一致性的特定形式。系统保证在没有后续更新的前提下，系统最终返回上一次更新操作的值。 也就是说，如果经过一段时间后要求能访问到更新后的数据，则是最终一致性。

共识（Consensus）

- 共识描述了分布式系统中多个节点之间，彼此对某个状态达成一致结果的过程。在实践中，要保障系统满足不同程度的一致性，核心过程往往需要通过共识算法来达成。。
- 共识和一致性常常被认为是等价的和可互换的。
- 共识侧重的是分布式节点达成一致性的过程和算法，是一种手段。而一致性侧重于节点共识过程最终达成的稳定状态，描述的是结果状态。达成某种共识并不意味着就保障了一致性（指强一致性）。只能说共识机制，能够实现某种程度上的一致性。

2. FLP定理和CAP定理

1985年Fischer、Lynch、Paterson发表FLP定理，后获得Dijkstra奖。

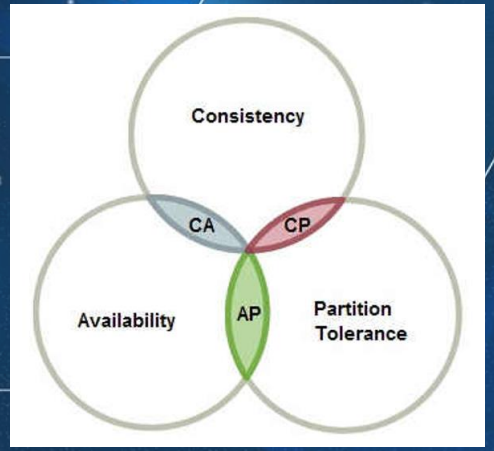
FLP定理

- 在含有多个确定性进程的异步系统中，只要有一个进程可能发生故障，那么就不存在协议能保证有限时间内使所有进程达成一致

FLP定理和CAP定理

CAP定理

- 网络服务不可能同时保证如下三个特点，最多只能保持两个：
- 一致性：（Consistency）
  - 指强一致性，分布式系统中的所有数据备份在同一时刻必须保持同样的值。
- 可用性：（Availability）
  - 集群的部分节点出现故障，系统仍可以处理用户请求，即所有读写请求可在一定时间内得到响应，不会一直等待。
- 分区容错性：（Partition-tolerance）
  - 出现网络分区、不同分区的节点间无法互相通信时被分隔的节点仍能正常对外服务。即允许丢失任意多的从一个节点发往另一个节点的消息。



在满足分区容错的前提下，不能同时满足一致性和可用性，只能兼顾。

FLP定理和CAP定理

CAP定理

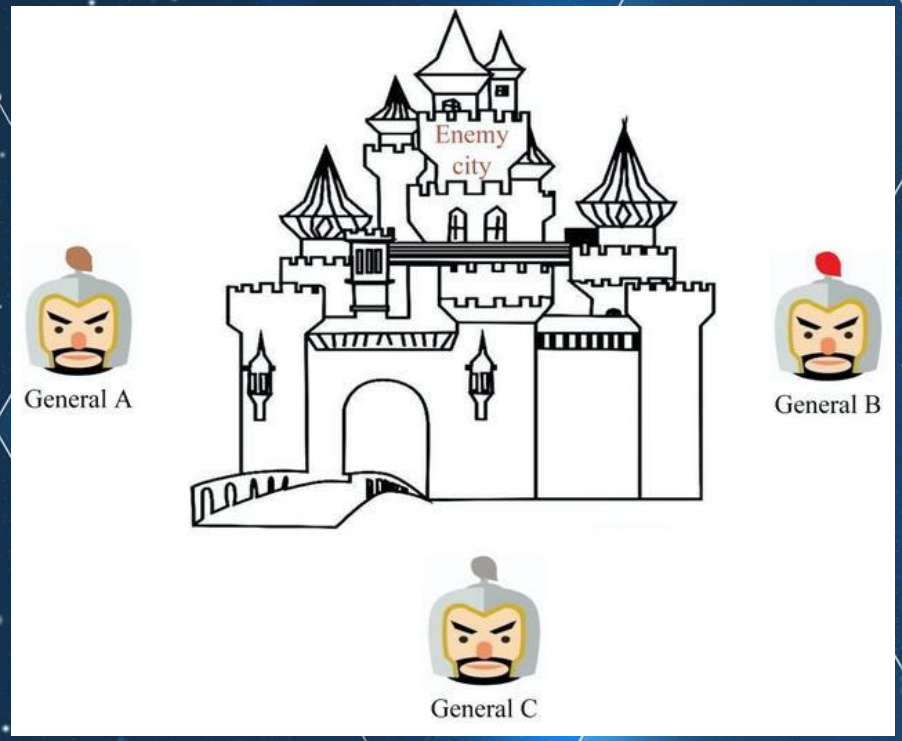
- 工程实践中，一般会适当放宽对特定性质的假设，例如放宽强一致性要求
  - 弱一致性：用户读到某一操作对系统特定数据的更新需要一段时间。
  - 最终一致性：在数据更新操作完成之后的某个时间点，分布式节点的数据最终达成一致。

区块链系统的设计也必须遵从CAP定理

- 以比特币为代表的大多数公有链通常牺牲强一致性，同时满足最终一致性、可用性和分区容错性。
- 某些联盟链或私有链可能会牺牲可用性来满足强一致性和分区容错性。

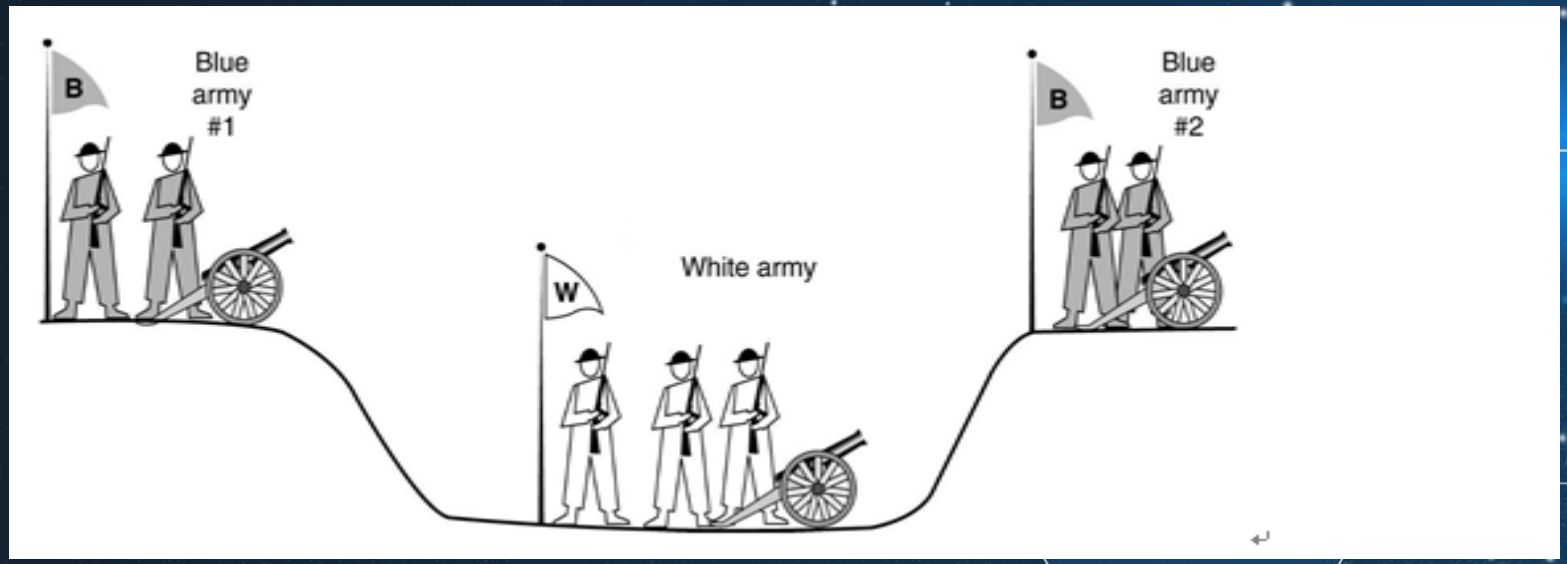
3. 拜占庭将军问题

拜占庭帝国想要进攻一个强大的敌人，为此派出了10支军队去包围这个敌人。这个敌人虽不比拜占庭帝国，但也足以抵御5支常规拜占庭军队的同时袭击。基于一些原因，这10支军队不能集合在一起单点突破，必须在分开的包围状态下同时攻击。他们任一支军队单独进攻都毫无胜算，除非有至少6支军队同时袭击才能攻下敌国。他们分散在敌国的四周，依靠通信兵相互通信来协商进攻意向及进攻时间。困扰这些将军的问题是，他们不确定他们中是否有内鬼，内鬼可能擅自变更进攻意向或者进攻时间。在这种状态下，拜占庭将军们能否找到一种分布式的协议来让他们能够远程协商，从而赢取战斗？这就是著名的拜占庭将军问题。



拜占庭将军问题不考虑通信兵是否会被截获或无法传达信息等问题，即节点不可靠，信道可靠。同步系统

两军问题



- 白军驻扎在沟渠里，蓝军则分散在沟渠两边。白军比任何一支蓝军都更为强大，但是蓝军若能同时合力进攻则能够打败白军。他们不能够远程的沟通，只能派遣通信兵穿过沟渠去通知对方蓝军协商进攻时间。是否存在一个能使蓝军必胜的通信协议，这就是两军问题。（节点可靠，信道不可靠）
- 经典情形下两军问题是不可解的，并不存在一个能使蓝军一定胜利的通信协议。
- TCP协议三次握手，是两军问题的工程解。

存在可能性

1. 在没有内鬼的情况下，如果 A 将军提议早晨 10 点进攻，并将这个消息依次传递给其他将军。再如果他幸运地收到了其他 5 位将军的同意，那这事儿就好办了。

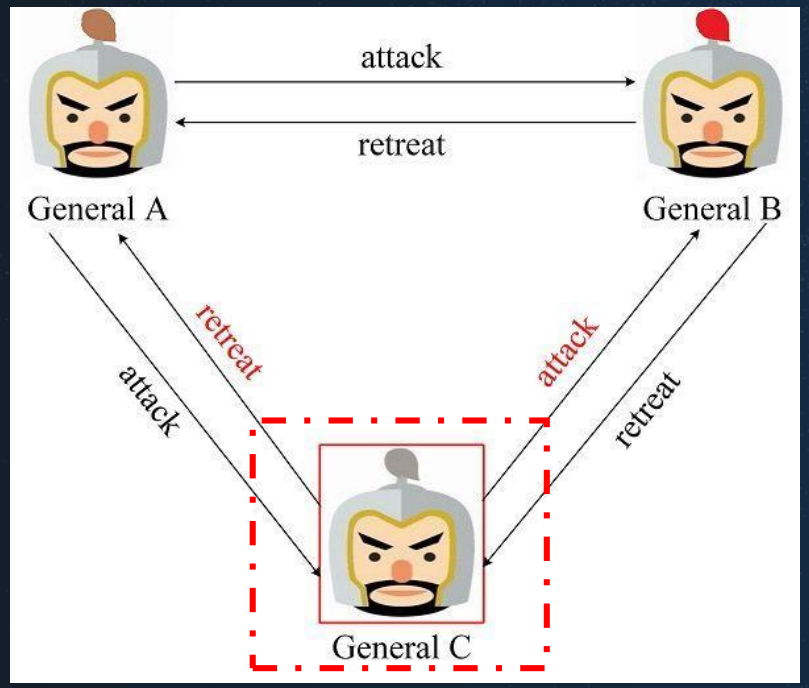
2. 在没有内鬼的情况下，如果 A 将军提议早晨 10 点进攻，并将这个消息传递给其他将军。但同时，B 将军也发出消息说上午 11 点进攻，C 将军又提议中午 12 点发起进攻。这样，将军们会收到不同的进攻消息。这时就可能 A 将军的提议得到了 3 个支持，B 将军的提议得到了 4 个支持，C 将军的提议也得到了 4 个支持，结果达不成 6 支或 6 支以上军队的一致，拜占庭肯定会失败。总之，发起提议的将军越多，造成的结果越乱。

3. 如果有内鬼，内鬼就可能向不同的将军发出不同的进攻提议，例如给 A 说 13 点，给 B 说 14 点，给 C 说 15 点，他也可能既同意 10 点的进攻提议，也同意 11点、12 点的提议，或者是他将别人的提议进行篡改后再往下传递，总之内鬼会想法促成一个不是 6 支以上军队都同意的决定，那么拜占庭的军队就只会失败。

进攻问题

一个将军发起具体的进攻命令

1. 将军本身就是叛徒，给不同的军队发出不同指令

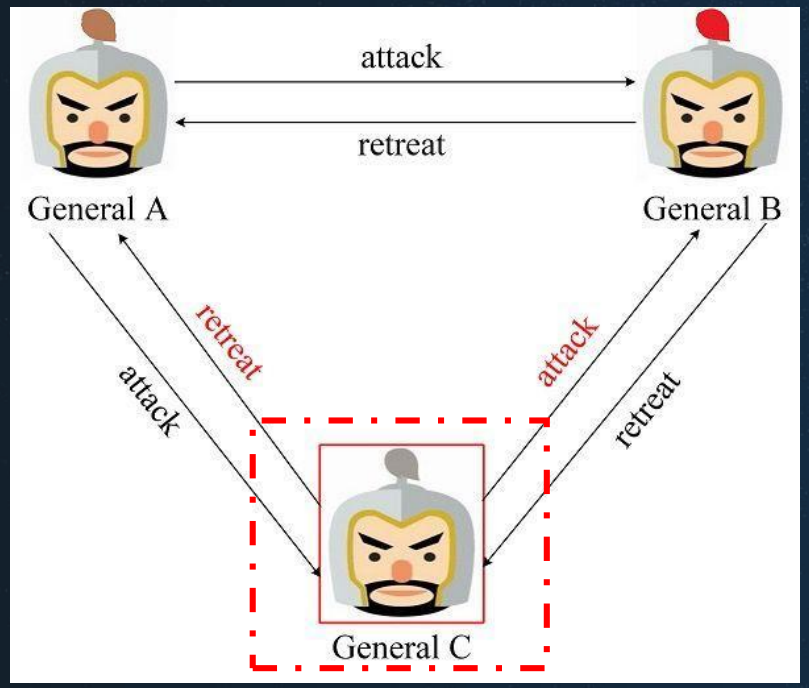


C将军是叛徒，将不同的指令发送给了AB军队

进攻问题

一个将军发起具体的进攻命令

2. 有的军队叛变了，将错误指令传递给别的军队



C军队叛变了是叛徒，将错误指令传递给了B军队

进攻问题

一个将军发起具体的进攻命令

1. 将军本身就是叛徒，给不同的军队发出不同指令
2. 有的军队叛变了，将错误指令传递给别的军队

如何在这种情况下，使得未叛变的军队保持一致的攻击指令?

问题模型

假设：

- 一个网络中存在n个节点，其中第i个节点发出的消息记为v_i
- 每个节点都会监听其他节点发送的消息，即$v_1, v_2, \ldots, v_n$
- 网络中存在m个恶意节点

求解问题：

- 在存在恶意节点的网络中，诚实节点能对决策问题达成一致

求解条件：

- 一致性>>每个诚实节点必须接收到相同的消息集合$v_1, v_2, \ldots, v_n$
- 正确性>>若节点i是诚实的，其他诚实节点必须以它发送的消息作为v_i

问题模型

求解条件：

- 一致性>>每个诚实节点必须接收到相同的$v_1, v_2, \ldots, v_n$

  ➡ 所有节点角度

- 正确性>>若节点i是诚实的，其他诚实节点必须以它发送的消息作为v_i

  ➡ 单个节点角度

- 一致性 ==>>转化为如下条件

  无论节点i是否诚实，任意两个诚实节点所保存的其消息均为v_i

  ➡ 单个节点角度

转化后的求解条件均从单个节点角度出发，因此可以设计针对单个节点的求解算法！

问题模型

转化问题：主从节点模式

转化条件：交互一致性条件

- IC1>>诚实的从节点会遵守相同的主节点消息
- IC2>>如果主节点诚实，每个诚实的从节点遵守主节点的消息

信道假设：

A1：每个消息都会被正确传递

A2：每个消息接受者都知道消息发送者

A3：消息的缺失可以被检测到

口头消息算法

口头消息（Oral Message, OM）算法（假设有m个非诚实节点）

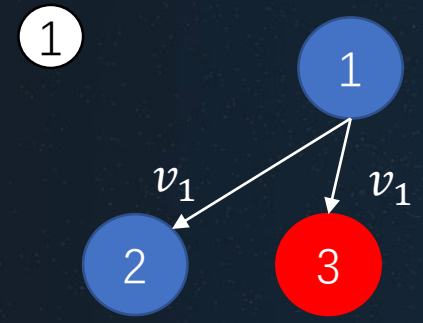
OM(0)
主节点向每个从节点发送消息
每个从节点接收消息，如果缺失则记为缺省值$\bar{v}$

OM(m)，m>0
(1)主节点向每个从节点发送消息v
(2)对任意从节点i，其接收的消息v记为v_i，其作为主节点运行OM(m-1)向剩余$n-2$个从节点发送v_i (3)对任意i和$j \neq i$，令v_j为OM（m-1)中从节点i从节点j接收的消息，从节点采用消息$majority$（v_1，v_2，……，v_{n-1}）。

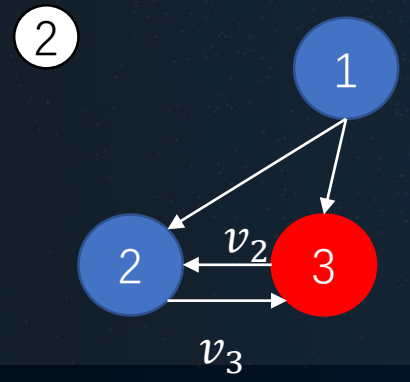
口头消息算法

口头消息（Oral Message, OM）算法

容错率小于1/3



$v_1 = 1$



$v_2 = 2$

$v_3 = 1$

node2 $maj(1, , 2) =?$

node3 $maj(1,1,)$

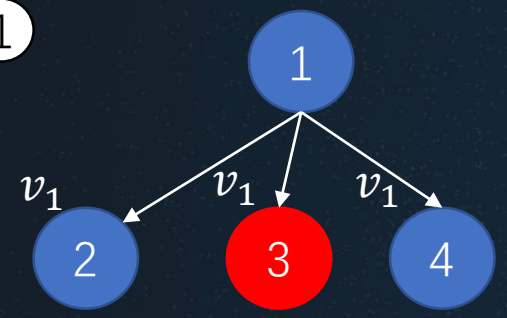
1. 当三个节点中存在一个非诚实节点时，诚实节点的行为会不确定
2. Lamport在文章中以此为基础证明容错率小于1/3

口头消息算法

OM(m)运行实例1

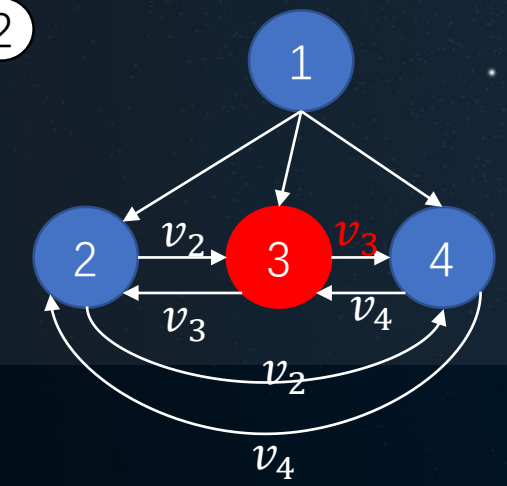
诚实节点对主节点的消息达成一致。

1



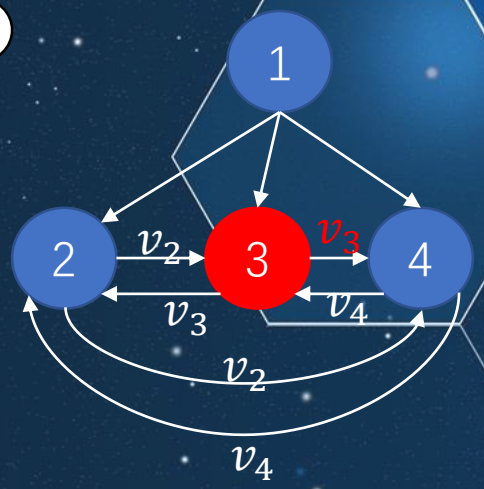
$v_1 = 1$

2



$v_2 = 1$
$v_2 = 1$
$v_3 = 1$
$v_3 = 2$
$v_4 = 1$
$v_4 = 1$

3

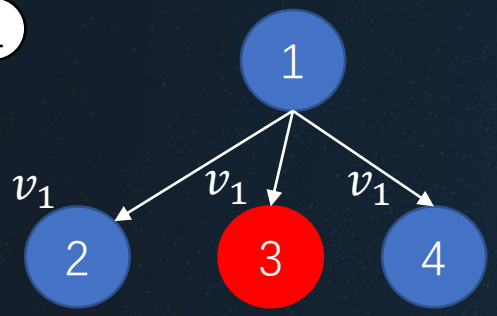


node2: $maj(1, ,2,1) = 1$
node3: $maj(1,1, ,1)$
node4: $maj(1,1,1,) = 1$

口头消息算法

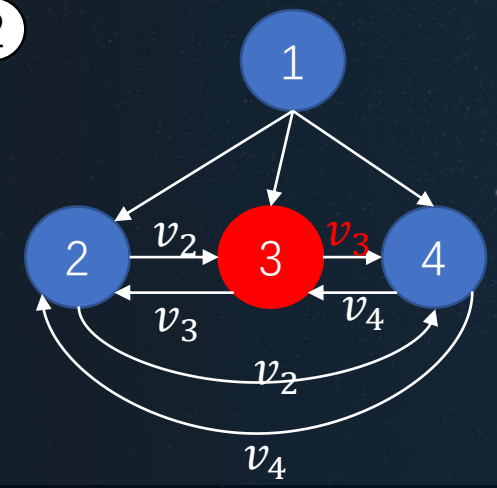
OM(m)运行实例1

1



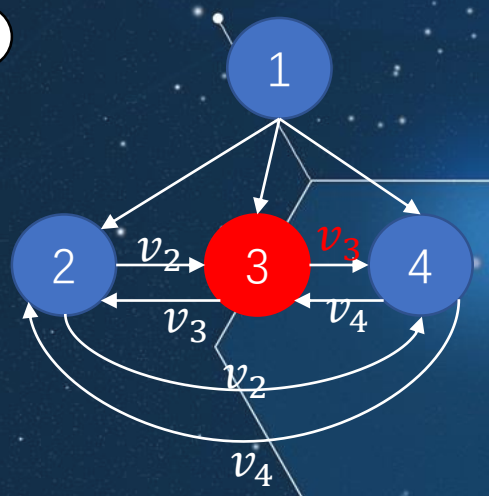
$v_1 = 1$

2



$v_2 = 1$
$v_2 = 1$
$v_3 = 1$
$v_3 = 2$
$v_4 = 1$
$v_4 = 1$

3



node2: $maj(1, , 2,1) = 1$
node3: $maj(1,1, , 1)$
node4: $maj(1,1,1,) = 1$

诚实节点对诚实主节点的消息达成一致。

当主节点为恶意节点时，诚实节点消息也能达成一致。（课下试试）

区块链解决拜占庭问题

我们给每个将军都配一台电脑。大家在电脑上发送信息，这样就不用派通信兵出去给其他将军传令了，先把时间节约下来。

然后我们再设定几个规则：

1.一个时间段内只能发起一个消息。比如 10 点到 10 点一刻，只能发出一个签名并盖上时间戳的进攻消息，别的将军想要发起别的进攻消息，那对不起，你得 10 点一刻以后才行。

2.消息传递出去以后，收到进攻消息的将军必须也要在消息上签名，确认各自的身份，并盖上时间戳，然后把这个信息拷贝下来传递给其他将军。

3.为防止有将军签假名，信息都加了密。系统中各个将军都有一个公用密码和私人密码，公用密码是公开的。A 将消息传递给 B 时，使用 B 的公用密码加密，而 B 则要用他的私人密码才能解密。B 签完名后，所有将军都可以通过他的公用密码来验证他签名的真实性。

4.大家都能从各自的电脑上看到信息的传递进度。

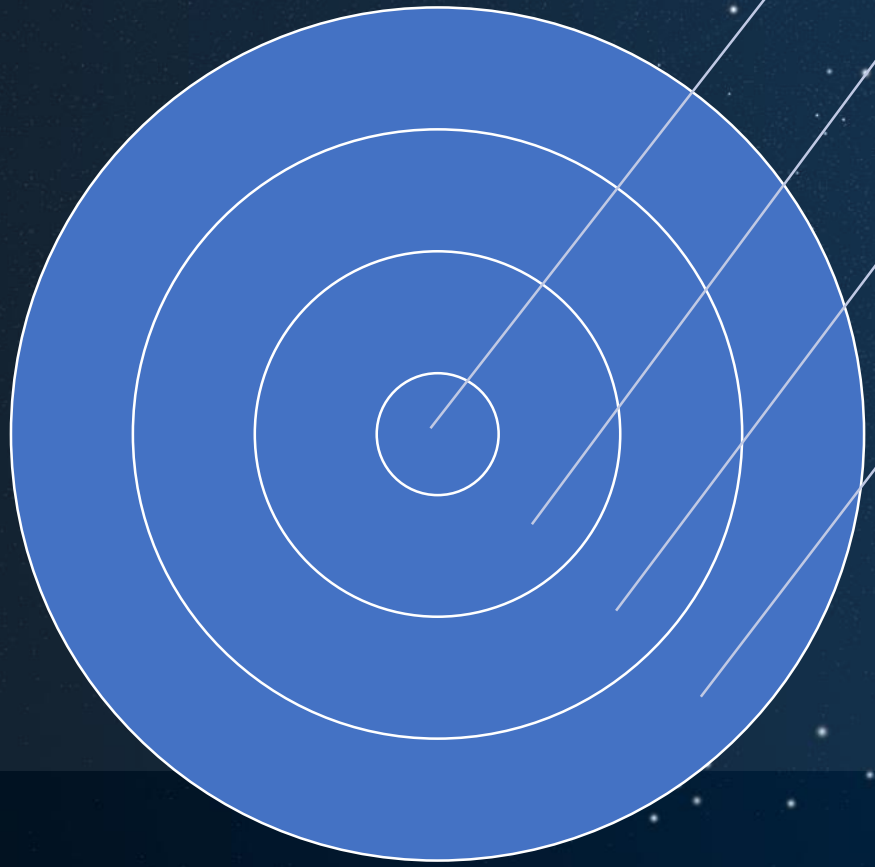
区块链解决方式优势

1.信息里每个将军都要签名验证身份，如果有将军篡改了消息，大家就能看到是哪些将军篡改了消息。

2.尽管有不一致的消息。但只要一个消息得到了6 名或 6 名以上将军的同意，那么大家就达成了共识。

这样每个将军都有一个与其他将军实时同步的消息记录，解决了内鬼问题，能让大家轻松地达成共识。

4. 共识过程的主流模型



记账节点

- 通过共识过程选定的记账节点

代表节点

- 特定算法选举出代表矿工节点参加共识过程

矿工节点

- 对数据或交易进行验证、打包、更新上链

数据节点

- 全体数据节点，生产数据或交易

共识过程的主流模型

选举共识

第一阶段：选主

选主是共识过程的核心，是通过选举、证明、联盟或混合等方式从全体矿工节点中选出记账节点的过程。

第二阶段：造块

第一阶段选出的记账节点根据特定的策略将当前时间段内全体节点生成的交易或数据打包到一个区块中，并将新生成的区块广播给全体矿工节点或代表节点。

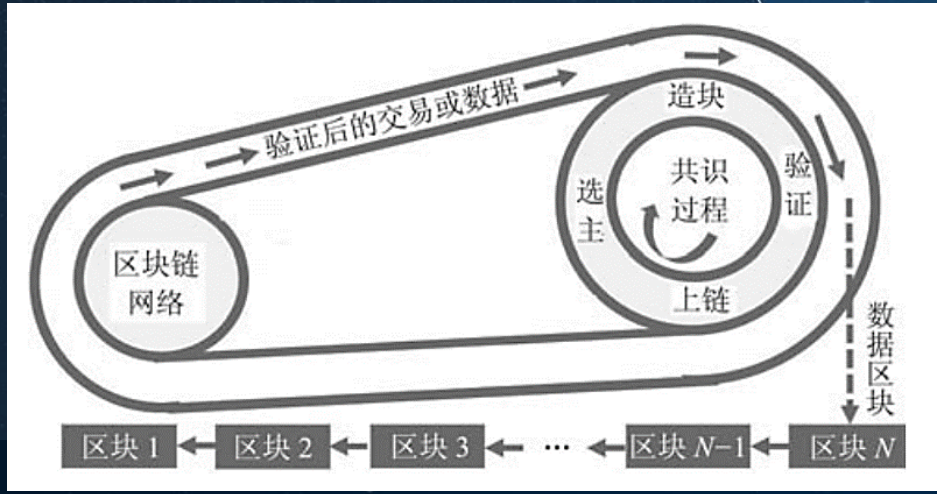
主链共识

第三阶段：验证

矿工节点或代表节点收到广播的新区块后，将各自验证区块内封装的交易或者数据的正确性和合理性。

第四阶段：上链

验证通过后记账节点将新区块添加到主链，形成一条从创世区块到最新区块的完整的、更长的链条。如果有分叉，需根据共识算法中的主链判别标准确定主链。



5. 共识算法的分类

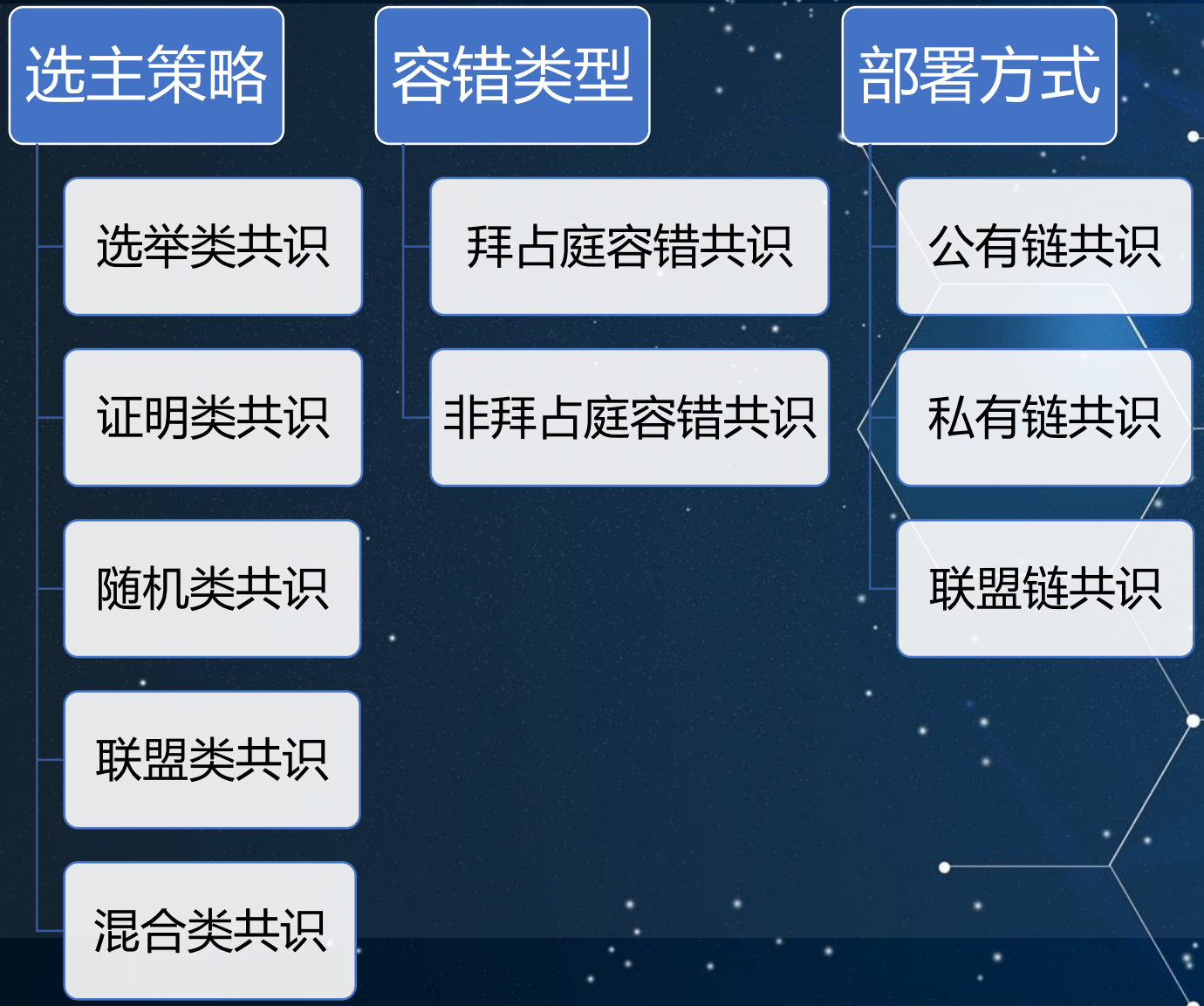
算法共识

- 研究在特定的网络模型和故障模型的前提下，如何在缺乏中央控制和协调的分布式网络中确保一致性，其实质是一种“机器共识”。
- 后续提到的区块链共识算法均指的是“算法共识”。

决策共识

- 研究无中心的群体决策中，如何就最优的决策达成一致的问题，例如关于比特币系统扩容问题和分叉问题的社区讨论与路线选择，其实质是“人的共识”。

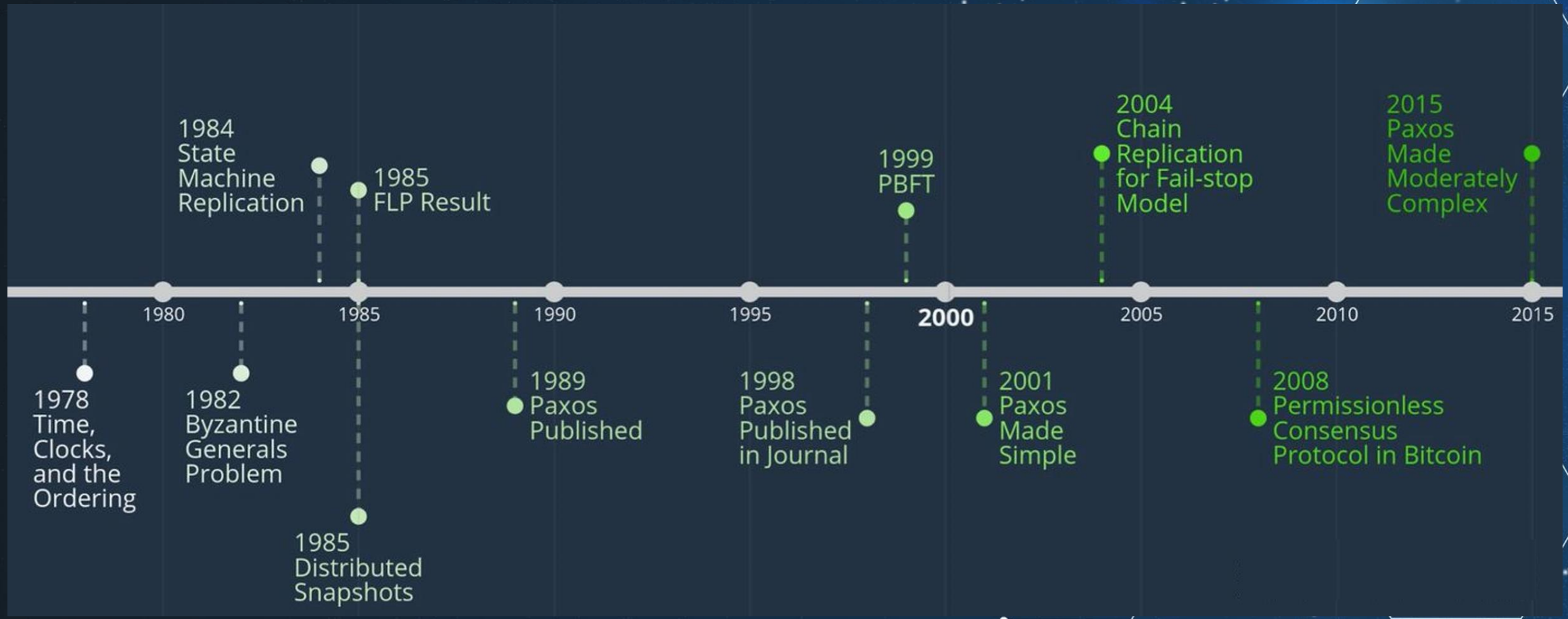
共识算法的分类


选主策略
选举类共识
证明类共识
随机类共识
联盟类共识
混合类共识
容错类型
拜占庭容错共识
非拜占庭容错共识
部署方式
公有链共识
私有链共识
联盟链共识

区块链共识算法分类

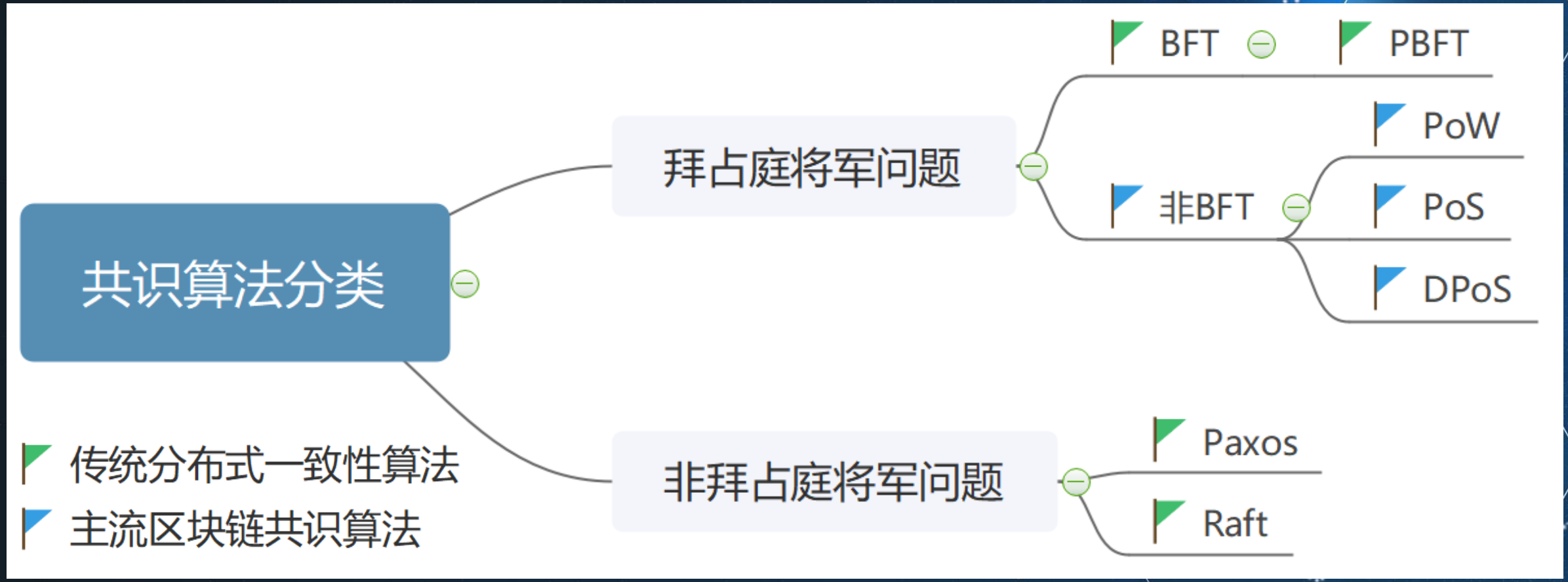
- 选举类共识：即矿工节点在每一轮共识过程中通过投票选举的方式选出当前轮次的记账节点,首先获得半数以上选票的矿工节点将会获得记账权 。例如 Paxos 和 Raft 等。
- 证明类共识：也可称为 Proof of X 类共识,即矿工节点在每一轮共识过程中必须证明自己具有某种特定的能力，证明方式通常是竞争性地完成某项难以解决但易于验证的任务, 在竞争中胜出的矿工节点将获得记账权。例如 PoW 和 PoS 。
- 随机类共识：即矿工节点根据某种随机方式直接确定每一轮的记账节点。例如 Algorand 和 PoET 。
- 联盟类共识：即矿工节点基于某种特定方式首先选举出一组代表节点，而后由代表节点以轮流或者选举的方式依次取得记账权。这是一种以代议制为特点的共识算法，例如 DPoS 等 。

6. 共识简史


1978
Time, Clocks, and the Ordering
1980
1982
Byzantine Generals Problem
1984
State Machine Replication
1985
Distributed Snapshots
1985
FLP Result
1989
Paxos Published
1990
1995
1998
Paxos Published in Journal
1999
PBFT
2000
2001
Paxos Made Simple
2004
Chain Replication for Fail-stop Model
2005
2008
Permissionless Consensus Protocol in Bitcoin
2010
2015
Paxos Made Moderately Complex
2015

一致性共识

一致性共识分类示意图



通常情况下，公有链基于拜占庭共识实现一致性，联盟链基于非拜占庭共识实现一致性。

传统分布式一致性算法

- 1975年，Akkoyunlu等人提出了计算机领域的两军问题及其无解性证明，对于共识机制的研究从此开始。

  两军问题表明，在不可靠的通信链路上试图通过通信达成一致共识是不可能的。

- 1982年，Lamport等人正式提出拜占庭将军问题，研究在可能存在故障节点或恶意攻击的情况下，非故障节点如何对特定数据达成一致，成为分布式共识的基础。

  拜占庭假设是对现实世界的模型化，强调的是由于硬件错误、网络拥塞或断开以及遭到恶意攻击，计算机和网络可能出现的不可预料的行为。

- 此后，分布式共识算法可以分为两类，即拜占庭容错（Byzantine Fault Tolerance, BFT）和非拜占庭容错（Crash Fault Tolerance, CFT）类共识。

传统分布式一致性算法

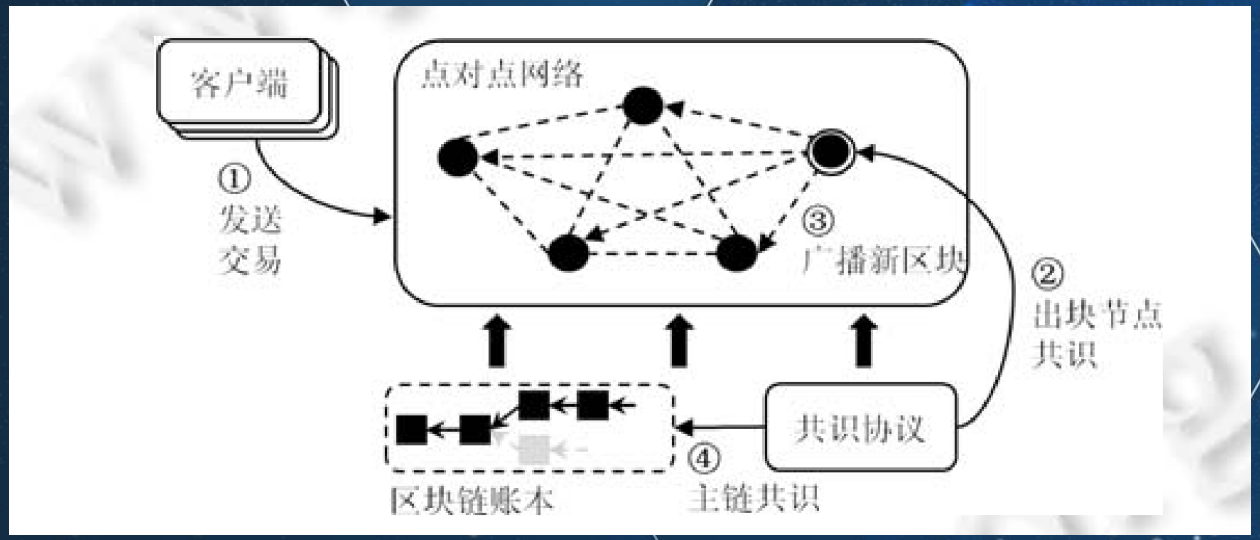
- 1988年, Oki和Liskov提出了VR一致性算法，采用主机-备份 (Primary-backup) 模式，规定所有数据操作都必须通过主机进行，然后复制到各备份机器以保证一致性。
- 1989年，Leslie Lamport 提出Paxos算法，是基于消息传递的一致性算法，主要解决分布式系统如何就某个特定值达成一致的问题。
- 2013年，斯坦福大学的DiegoOngaro和John Ousterhout提出了Raft共识算法。Raft的初衷是为设计一种比Paxos更易于理解和实现的共识算法。
- 传统分布式一致性算法一般均假设系统中不存在拜占庭故障节点，因而是非拜占庭容错的共识算法。除此之外，还有两阶段提交(Two-phase commit) 算法、三阶段提交 (Three-phase commit)算法、Zab、Zyzzyva、Kafka等早期共识算法。这些算法广泛应用于分布式数据库、联盟链和私有链。

区块链共识协议

1、出块节点选举：在出块节点选举阶段,某个节点(或多个节点)成为出块节点，提出新区块。由于分布式网络中可能存在的恶意节点及分叉块的影响，其他节点在收到新区块以后不能直接将其加入自己的本地区块链中。

2、所有节点需要利用主链共识对新区块及其构成的主链达成一致。

- 出块节点选举机制和主链共识共同保证了区块链数据的正确性和一致性，从而为分布式环境中的不可信主体间建立信任关系提供技术支撑 。



主流区块链共识算法

- 1993年，哈佛大学教授 Cynthia Dwork 首次提出了工作量证明思想，用来解决垃圾邮件问题。

  该机制要求邮件发送者必须算出某个数学难题的答案来证明其确实执行了一定程度的计算工作，从而提高垃圾邮件发送者的成本.

- 1999年，Barbara Liskov 等提出了实用拜占庭容错算法（Practical Byzantine fault tolerance，PBFT），解决了原始拜占庭容错算法效率不高的问题，将算法复杂度由指数级降低到多项式级，使得拜占庭容错算法在实际系统应用中变得可行。
- 2008年10月，中本聪发表的比特币创世论文催生了基于区块链的共识算法研究。

主流区块链共识算法

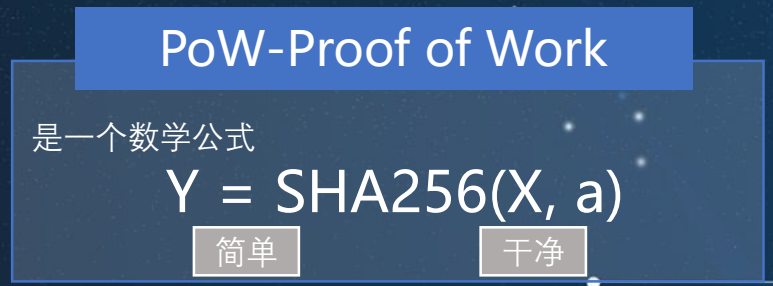
- 2011年7月，一位名为Quantum Mechanic的数字货币爱好者在比特币论坛(www.bitcointalk.org)首次提出了权益证明PoS共识算法。Sunny King在2012年8月发布的点点币 (Peercoin, PPC) 中首次实现。
- 2013年8月，比特股(Bitshares)项目提出了一种新的共识算法，即授权股份证明算法 (Delegated proof-of-stake, DPoS)。

PoW共识

POW（Proof of Work），工作量证明，引入了对一个特定值的计算工作。

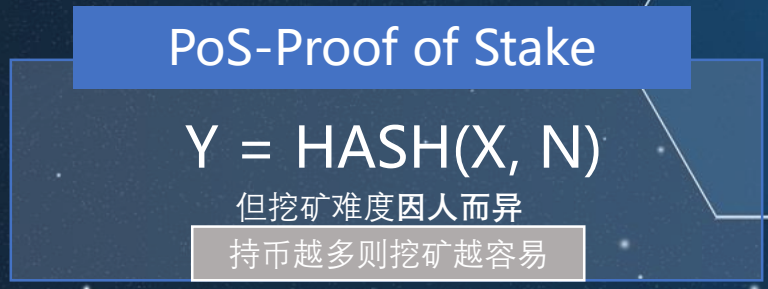
比特币采用的共识算法就是POW，矿工们在挖一个新的区块时，必须对SHA-256密码散列函数进行运算，区块中的随机散列值以一个或多个0开始。随着0数目的上升，找到这个解所需要的工作量将呈指数增长，矿工通过反复尝试找到这个解。

在这其中，如果想要对业已出现的区块信息进行修改，攻击者必须完成该区块外加之后所有区块的工作量，并最终赶上和超越诚实节点的工作量。



PoS共识

POS（Proof of Stake），权益证明，试图解决POW机制中大量资源被浪费的情况。这种机制通过计算你持有占总币数的百分比，包括你占有币数的时间来决定记账权。



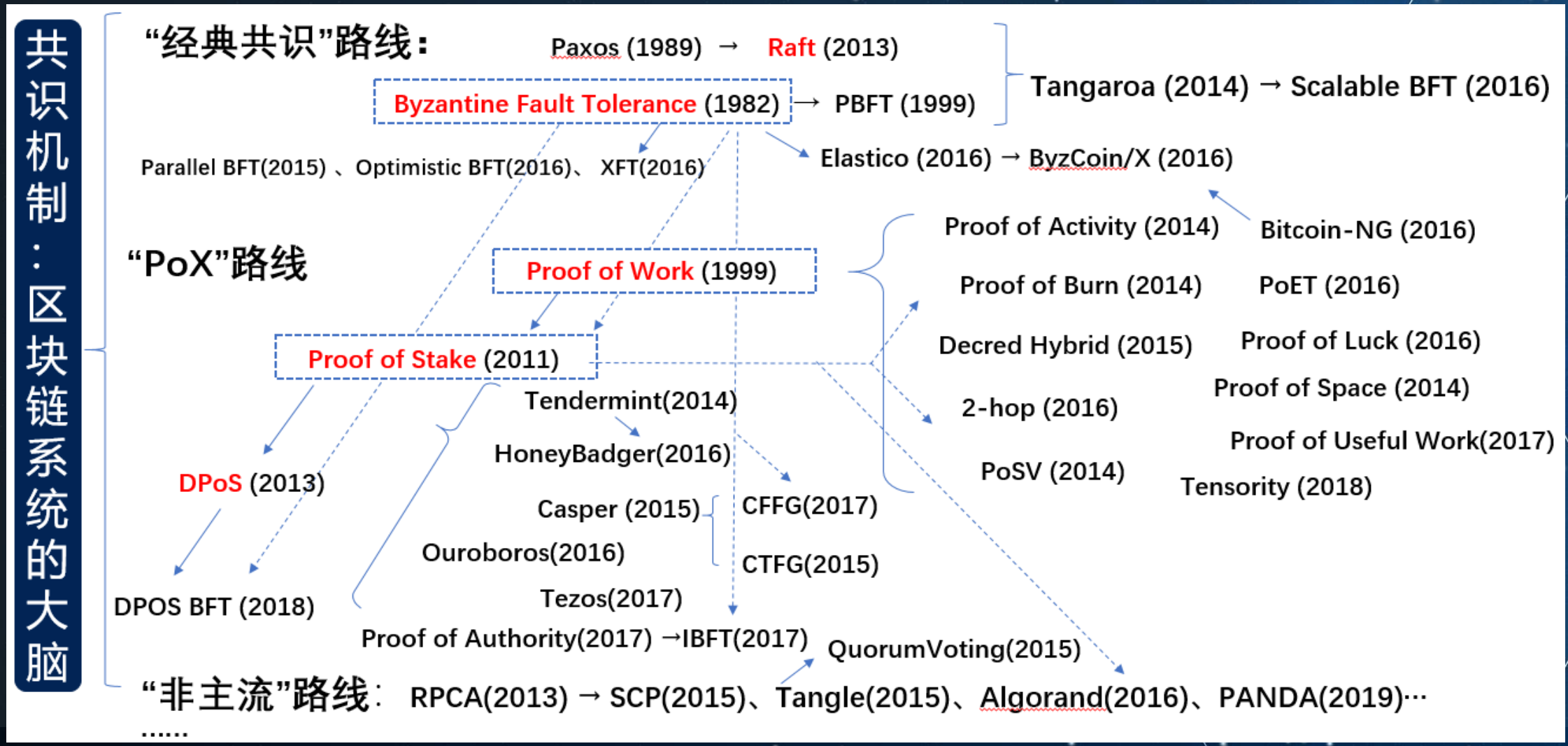
区块链共识算法一览表

名称	提出年份	拜占庭容错	基础算法	代表性应用
Viewstamped replication	1988	否	无	BDB-HA
Paxos(族)	1989	否	无	Chubby
PBFT	1999	是(<1/3)	BFT	Hyperledger v0.6.0
PoW	1999	是(<1/2)	无	Bitcoin
PoS	2011	是(<1/2)	无	Peercoin, Nxt
DPoS	2013	是(<1/2)	PoS	EOS, Bitshares
Raft	2013	否	无	etcd, braft
Ripple	2013	是(<1/5)	无	Ripple
Tendermint	2014	是(<1/3)	PoS+PBFT	Monax
Tangaroa(BFTRaft)	2014	是(<1/3)	Raft+PBFT	--
Proof of acitvity	2014	是(<1/2)	PoW+PoS	Decred
Proof of burn	2014	是(<1/2)	PoW+PoS	Slimcoin
Proof of space	2014	是(<1/2)	PoW	Burstcoin
Proof of stake velocity(PoSV)	2014	是(<1/2)	PoW+PoS	ReddCoin
Casper	2015	是(<1/2)	PoW+PoS	Ethereum
Quorum voting	2015	是(<1/3)	Pipple+Stellar	Sawtooth Lake

区块链共识算法一览表

名称	提出年份	拜占庭容错	基础算法	代表性应用
Stellar(SCP)	2015	是(<1/3)	Pripple+BFT	Stellar
Algorand	2016	是(<1/3)	PoS+BFT	ArcBlock
Bitcoin-NG	2016	是(<1/2)	PoW	--
Byzcoin	2016	是(<1/3)	BTC-NG	--
dBFT	2016	是(<1/3)	PoS+pBFT	NEO
Elastico	2016	是(<1/3)	PBFT+PoW	--
HoneyBadger	2016	是(<1/3)	Tendermint	--
PoET	2016	是(<1/2)	PoW	Sawtooth Lake
Proof of luck	2016	是(<1/2)	PoW	Luckychain
Scalable BFT	2016	是(<1/3)	Tangaroa	Kadena
2-hop	2017	是(<1/2)	PoW+PoS	--
ByzCoinX	2017	是(<1/3)	ByzCoin+Elastico	OmniLedger
Proof of authority	2017	是(<1/2)	PoS	Parity
Proof of useful work	2017	是(<1/2)	PoW	--
Ouroboros	2017	是(<1/2)	PoS	Cardano
Sleepy consensus	2017	是(<1/2)	PoS	--

区块链共识算法发展脉络



4.2 分布式一致性算法

- Paxos算法
- Raft算法

1. Paxos算法

- Paxos算法由Lamport在1998年发表的《The Part-Time Parliament》中提出。2001年，Lamport又发表了《Paxos Made Simple》，对这个算法进行了更加易于理解的描述

问题描述
希腊岛屿Paxos上的执法者（legislators，后面称为牧师priest）在议会大厅（chamber）中表决通过法律，并通过服务员传递纸条的方式交流信息，每个执法者会将通过的法律记录在自己的账目（ledger）上。问题在于执法者和服务员都不可靠，他们随时会因为各种事情离开议会大厅，并随时可能有新的执法者进入议会大厅进行法律表决，使用何种方式能够使得这个表决过程正常进行，且通过的法律不发生矛盾。
说明：不难看出故事中的议会大厅就是我们的分布式系统，牧师对应节点或进程，服务员传递纸条的过程就是消息传递的过程，法律即是我们需要保证一致性的值（value）。牧师和服务员的进出对应着节点/网络的失效和加入，牧师的账目对应节点中的持久化存储设备。上面表决过程的正常进行可以表述为进展需求（progress requirements）：当大部分牧师在议会大厅呆了足够长时间，且期间没有牧师进入或者退出，那么提出的法案应该被通过并被记录在每个牧师的账目上。

Paxos算法解决的问题是在一个可能发生消息延迟、丢失、重复的分布式系统中如何就某个值达成一致，保证不论发生以上任何异常，都不会破坏决议的一致性。（非拜占庭故障）

Paxos-问题描述

- Paxos算法将分布式系统的节点分为三种角色：
  - 提议者（Proposer）负责向acceptor发起提案
  - 接受者（Acceptor）负责响应提案，对提案进行回应以表示自己接受提案
  - 学习者（Learner）不参与前面的决策过程，只从别人那里学习已经确定的、达成一致的提案结果
  - 一个节点可以同时拥有这三种身份，也可以只有部分身份。
- 如果一个提案被半数以上Acceptor接受，它就被选定了（Chosen），并由Learner负责执行选定的提案。

Paxos-问题描述-提案

- 提案Proposal由两部分组成：提案编号+提案值
- 提案编号id由Proposer自行选择决定，一般是相互独立、不可重复的递增序列。一种可选的方案是M个Proposer均被分配一个1到M之间的唯一数字，对于 i Proposer，其第n次提案的编号可以为M(n-1)+i。
- 提案值（Value）是要等待达成共识的数据值。Paxos要保证在众多被提出来的Value中，只有一个会被最终选定。

Paxos-问题描述-约束条件

- 对Value：如果没有Value被提出，就不应该有Value被选定。
  - A：只有被提出的值Value才可以被选定
  - B：只有一个值Value可以被选定
  - C：除非一个值Value被选定，否则它不会被执行

Paxos-问题描述-约束条件

- 对于提案：如果只有一个提案被提出的话,那么这个提案应该被最终选定，Acceptor必须能够接受多个不同的Value。
  - P1：每个 Acceptor必须接受它收到的第一个提案
  - P2：如果一个提案（其值为v）已经被选定，那么对于所有编号更大的被选定的提案，它们所提议的值也必须是v。
    - P2a：如果一个提案（其值为v）已经被选定，那么对于任何Acceptor接受的编号更大的提案，它们的值也是v。
    - P2b：如果一个提案（其值为v）已经被选定，那么对于任何Proposer提出的编号更大的提案，它们的值也是v。
    - P2c：对于任意的v和n，如果提案（编号为n，值为v）被提出，那么存在一个由大多数Acceptors构成的集合S，满足下述两个条件之一：
      - ① 没有成员接受过小于n的提案;
      - ② 成员接受过的提案中，编号最大者的值为v。
  - 上述约束条件的关系是P2<P2a<P2b<P2c，即如果满足P2c就可以确保满足P2，Paxos算法就是建立在P2c上。

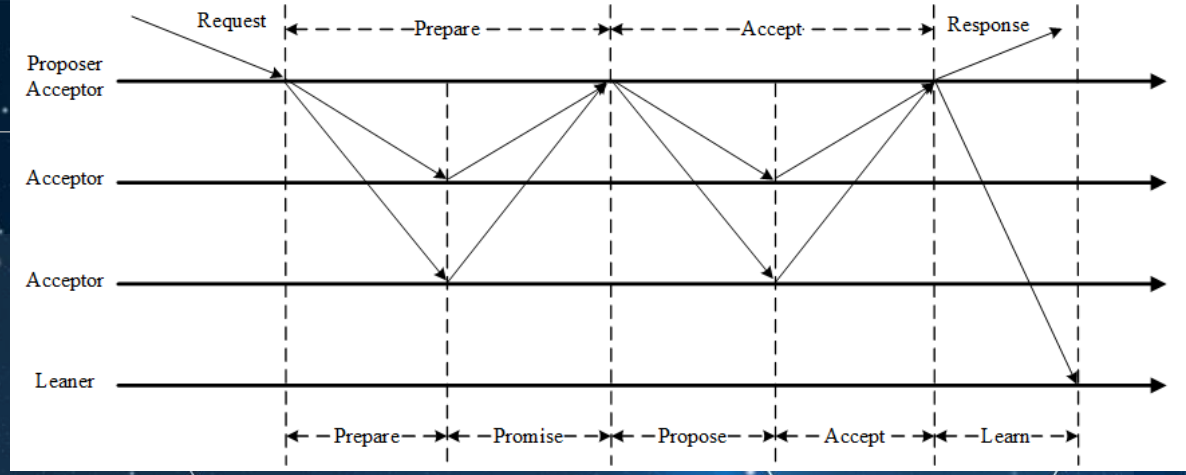
Paxos-算法流程

- Paxos算法分为两个阶段：
- Phase 1
  - (1)准备(Prepare)：一个Proposer创建一个提案(编号N)，并向超过半数的Acceptors发送包含提案编号的Prepare(N)消息
  - (2)承诺(Promise)：每个Acceptor收到消息后，检查提案的编号N是否大于它曾接受过的所有提案的编号。如果是，它会回应以Promise(Nx,Vx)消息，承诺不会接受任何编号小于N的提案；否则它将不予回应。其中Nx和Vx是它曾接受过的提案中编号最大的提案的编号与值，如果没有接受过提案，Nx和Vx为NULL。
- Phase 2
  - (1)请求接受(Accept Request)：如果Proposer收到了超过半数Acceptors的Promise消息，它需要先找到这些消息中编号最大的提案的值Vn，然后向这些Acceptors发送Accept(N,Vn)消息；如果所有Promise消息中Nx和Vx都为NULL，则Proposer可以选择任意的值作为V。
  - (2)接受(Accepted)：当Acceptor收到Accept(N,V)消息，它首先检查是否已承诺过编号大于N的提案，如果答案是否定的，它就接受该提案N，并发送Accepted（N,Vn)；否则就拒绝。

Paxos-算法流程

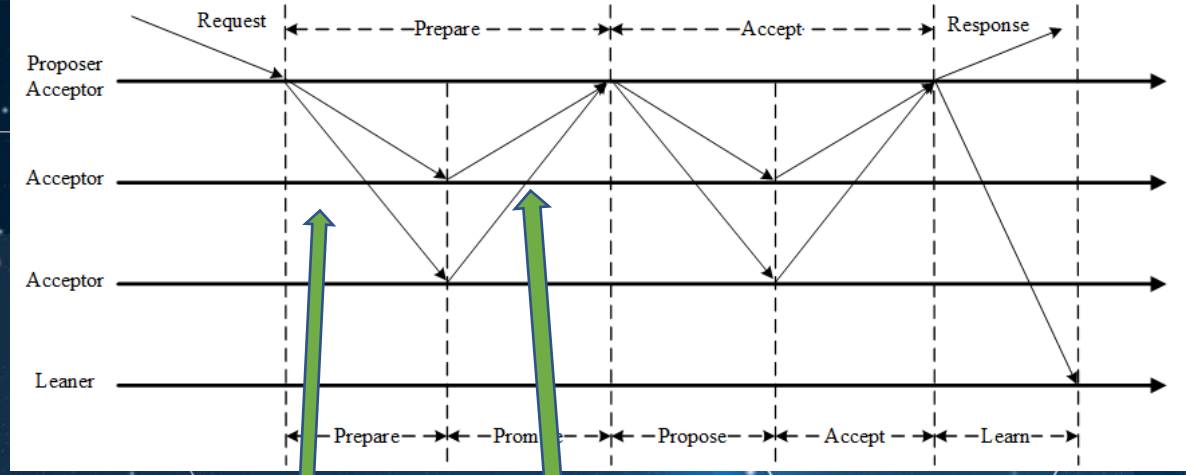
- 当获得半数以上Accepted返回后，该提案被选定，并提交Learner执行。Learner可以通过三种方式获取被选定的值value：
  - 方式一：Acceptor每接受一个提案，就将该提案发送给所有Learner。这种方式可以使得Learner快速获取被选定的value，但是由于每个Acceptor都要与每个 Learner通信，所需的通信次数等于二者数量的乘积。
  - 方式二: Acceptor每接受一个提案，就将该提案发送给主Learner；当提案值被最终选定后，再由主Learner发送给其他Learner。这种方式将通信次数降低为 Acceptor和Learner的数量之和，但是主Learner可能会发生单点故障问题，降低了系统可靠性。
  - 方式三: Acceptor每接受一个提案，就将该提案发送给一个Learner集合，该集合中的每个Learner都可以将选定的提案值发送给所有的Learner。显然,这是方式一和方式二的折中方案，Learner集合的数量越多，系统可靠性就越好，但通信复杂度也相应地越高。

Paxos



- 下面介绍一轮Paxos算法的过程
- 一轮Paxos只对一个值达成共识
- Acceptor本地记录以下几个值
  - minProposal 自身响应的提案id最大prepare请求的提案id
  - acceptedProposal 自身响应的accept请求中提案编号最大的提案id
  - acceptValue 自身响应的accept请求中提案编号最大的提案值

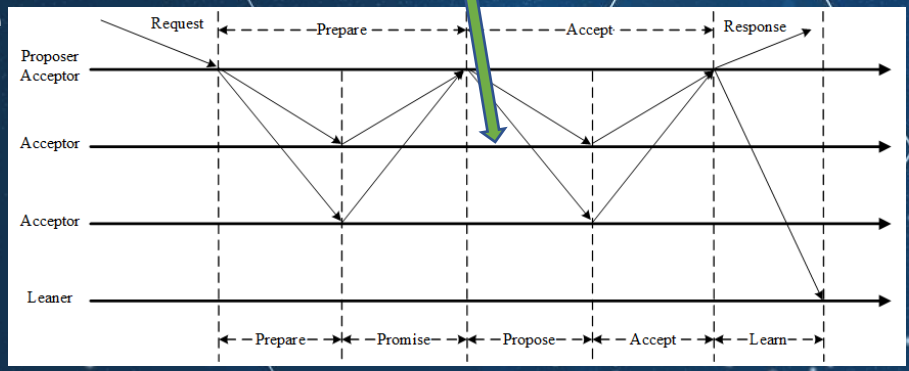
Paxos



- 下面介绍一轮Paxos算法的过程
- 一轮Paxos只对一个值达成共识
- 只考虑单个proposer，Paxos算法的流程如图（图中proposer自己也是一个acceptor）：
  - Prepare阶段：
    - Proposer向Acceptor发送prepare请求。Prepare消息包含这次提案的id。
    - Acceptor收到后向proposer返回Promise消息。Promise消息包含该Acceptor accepted的拥有最大编号提案的值以及该提案的id（即acceptedProposal和acceptedValue），如果没有则留空（accept的具体含义之后说明）。然后Acceptor将该prepare请求的提案id记为自己的minProposal
      - Acceptor在对一个编号为n的提案做出响应后，不会再对编号小于n的prepare请求做出响应，也不会对编号小于n的accept请求作出响应（accept请求定义之后说明）。因此如果一个Acceptor在这之前已经对一个编号大于n的提案做出响应，则该Acceptor在这里不会返回promise消息。

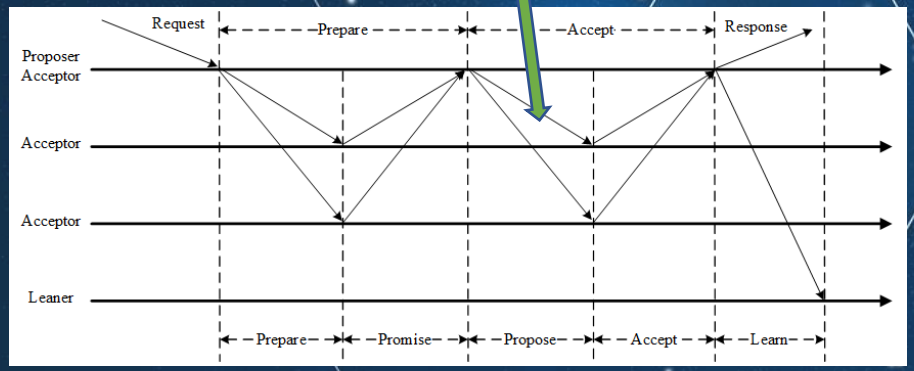
Paxos

- 只考虑单个proposer，Paxos算法的流程如图（这里proposer自己也是一个acceptor）：
  - 在Proposer收到过半的Acceptor的promise消息后，进入propose阶段，向Acceptor发送Accept请求。
    - 请求同样要带有之前提到的提案id n
    - 请求还要带有一个值，这个值为对要达成共识的值的一个提案
      - 如果之前收到的所有promise消息中都没有附带acceptedValue，则这个值为Proposer自己选中的值
      - 如果至少有一个收到的promise消息中带有acceptedValue，则从这些消息中找出acceptedProposal最大的消息对应的acceptedValue



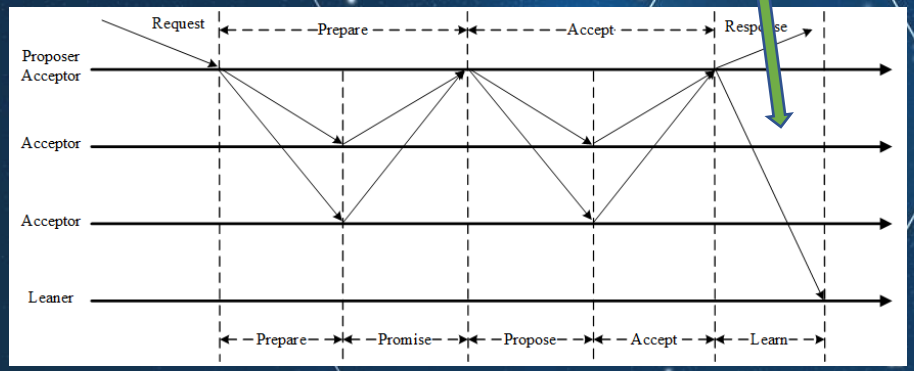
Paxos

- 只考虑单个proposer，Paxos算法的流程如图（这里proposer自己也是一个acceptor）：
  - Acceptor在收到accept请求之后：
    - 首先检查这个accept请求的提案id n，如果n小于自己最后响应的其他prepare请求的id即minProposal则向Proposer返回自己的minProposal值
    - 否则，该Acceptor会接受这个accept请求。Acceptor会记下这个请求的id以及其包含的值作为自己的acceptedProposal（同时也作为minProposal）和acceptedValue，然后同样也会向Propose返回自己的minProposal
  - Proposer等待收到过半Acceptor的响应
    - 如果响应中包含的minProposal值大于自己当前的提案id，则放弃本轮
    - 否则说明已经得到了过半Acceptor的支持，即认为已经达成了一致



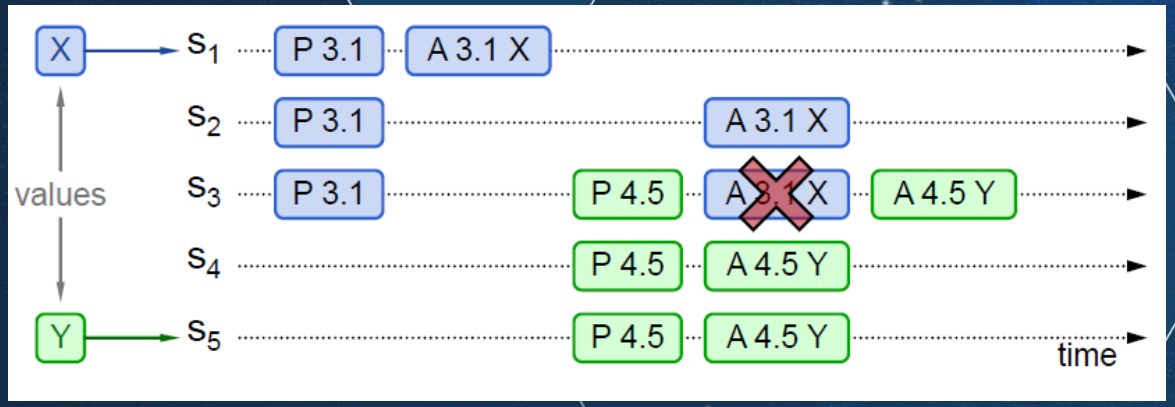
Paxos

- 只考虑单个proposer，Paxos算法的流程如图（这里proposer自己也是一个acceptor）：
  - 在确认已经达成了一致之后，Proposer就可以将结果发送给learner令其记下结果
    - 除了这种方式，也可以让Acceptor在每次接受一个值后就告诉Learner，Learner在收集到过半的相同id的值后就可以确定结果
    - 还有其他方法，具体使用哪种方法取决于对通信代价和速度的取舍



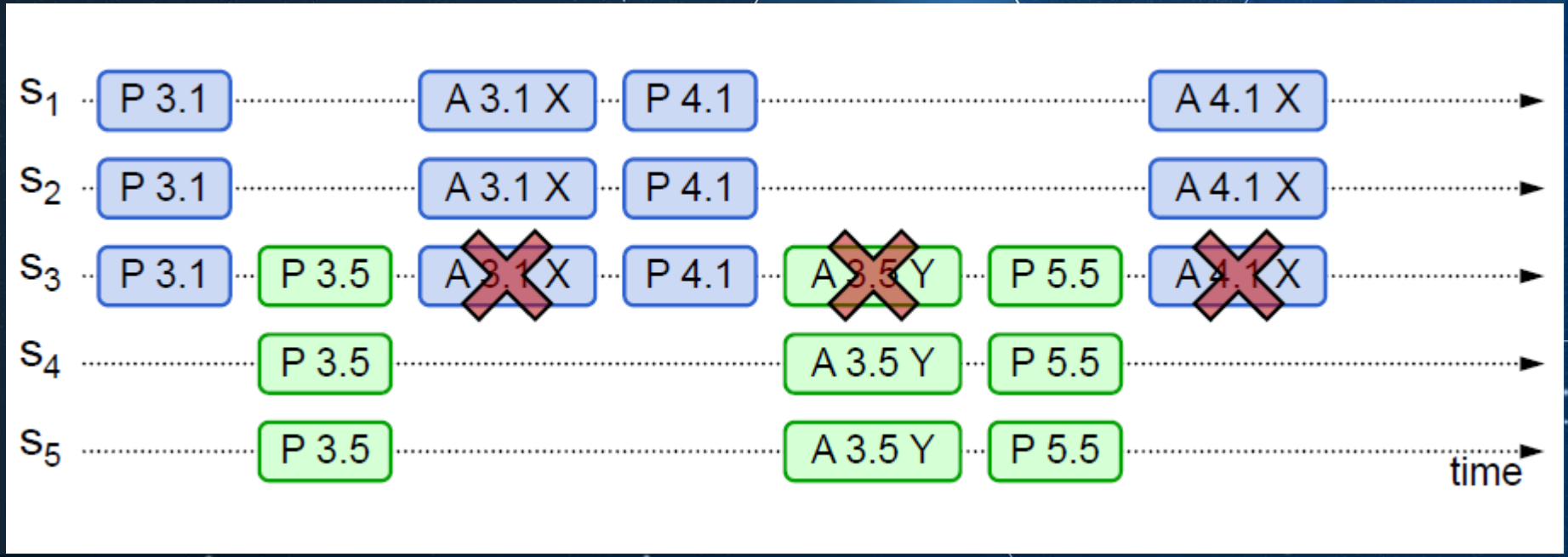
Paxos-活锁

- 上述过程只考虑了一个Proposer的情况，所以一般两轮广播+收集就能直接完成。如果有多个Proposer先后发起请求，后一个prepare请求就会使前一个prepare请求后续的accept请求失效，除非这个accept请求在后一个prepare发出之前就收到了足够的响应达成共识。
  - 图中S1的prepare（id 3.1）完成之后就发起accept请求，然而在S3收到请求并做出回应之前S5发来了有更大id（4.5）的prepare请求，这就使得S1对S3发出的这个accept请求失效。



Paxos-活锁

- 这种情况可能连续、交替发生
  - 图中S1 S5每当自己的请求确认失败后马上用新的请求id发起新一轮prepare，所以二者的prepare不断使对方的请求失效，这个过程有可能一直持续从而形成活锁。这种情况下系统无法继续运行下去



Paxos-活锁

- 为了解决这样的问题，在实际使用时可以进行这样的修改：
  - 在集群中选出一个节点作为leader，在对每条数据/每条事务达成一致性的时候，只有leader会作为Proposer而其他节点不会发起提案。对每条事务都运行一次上述的一轮Paxos，由于此时只有一个Proposer，所以就不会出现活锁的情况。
  - Leader可以向其他节点定时发送一个心跳包来声明自己存活。一旦其他节点认为leader消失了，则可以发起竞选来尝试成为leader，选出leader的过程就也能直接使用一轮Paxos算法来选出。

Raft

- Raft算法在《In Search of an Understandable Consensus Algorithm》中被提出
- 文章中多次提到Paxos算法不易于理解，并强调Raft算法比起Paxos算法更加易于理解。文章甚至在最后进行性能评价的时候加入了Understandability作为一个评价的指标

9.1 Understandability

To measure Raft's understandability relative to Paxos, we conducted an experimental study using upper-level undergraduate and graduate students in an Advanced Operating Systems course at Stanford University and a Distributed Computing course at U.C. Berkeley. We recorded a video lecture of Raft and another of Paxos, and created corresponding quizzes. The Raft lecture covered the content of this paper except for log compaction; the Paxos

谢谢!