

Review

Lecture 1 - cryptocurrency

比特币作为加密货币代表迅速崛起，其使用use和交换exchange逐步广泛

但比特币同时有其黑暗面，例如洗钱（Money laundering），恶意挖矿（Rogue mining）等

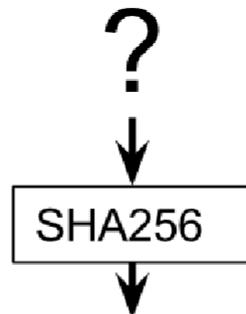
Hash Function

一种映射函数，将不同长度输入映射得到相同长度输出，通常使用256bits

即使只有1bit的输入差异，输出也会完全不同

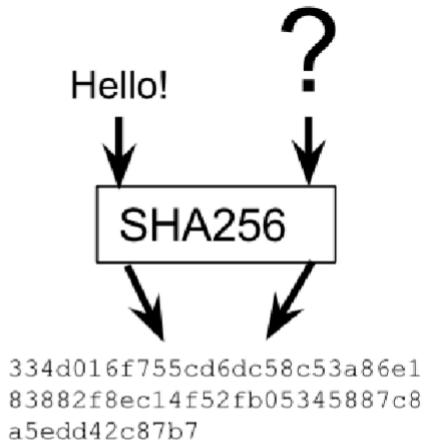
哈希函数属性：

- **抗前置像性 (Preimage Resistance)**：给定一个哈希值 h ，要找到任何能生成它的原始消息 m ，在计算上是极难的。只能够guess或暴力破解。单向性



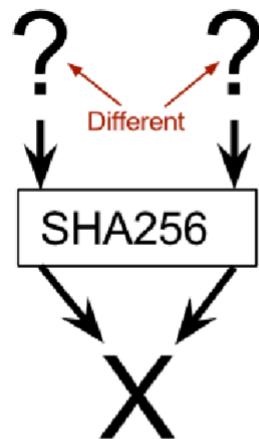
Preimage resistance

- **抗第二原像性 (Second Preimage Resistance)** 给定一个输入 m_1 ，很难找到另一个不同的输入 m_2 ，让 $\text{hash}(m_1) = \text{hash}(m_2)$ 。一旦公布哈希值，就难以反悔 regret /篡改 change 原始内容



Second-preimage resistance

- **抗碰撞性 (Collision-Resistance)**: 很难找到两个不同的消息 m_1 和 m_2 ，使得 $\text{hash}(m_1) = \text{hash}(m_2)$ 。保障区块链中的信任体系，防止构造两笔不同的拥有相同哈希值的交易，否则两则消息都是可信的



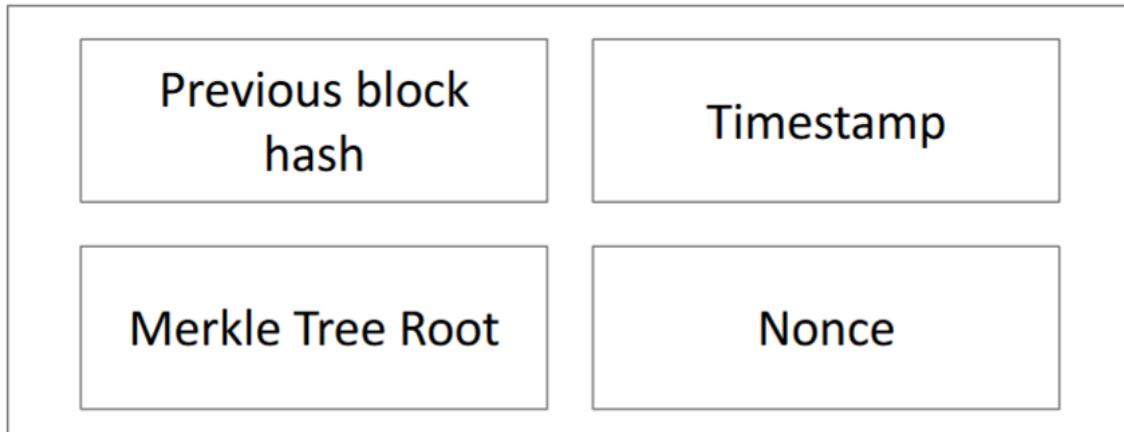
Collision resistance

BlockChain

如何验证交易数据？哈希链 + 黑克尔树

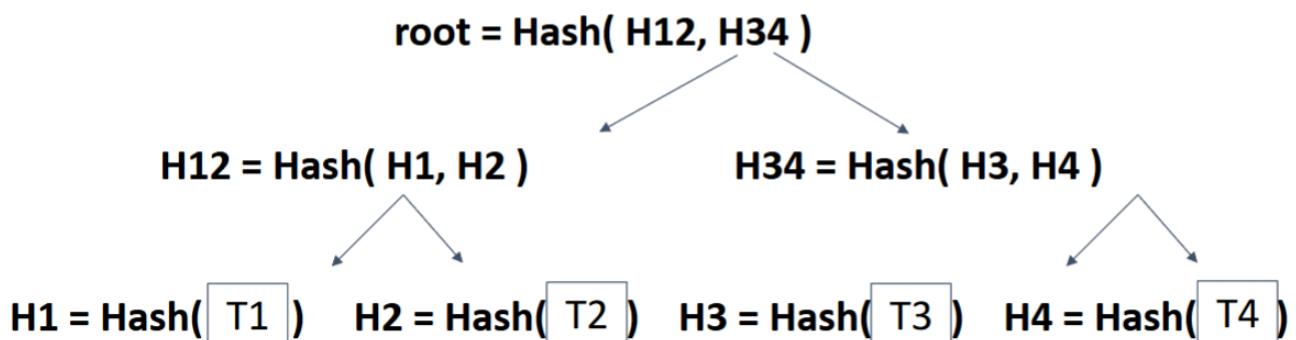
将每笔交易用结构化数据表示，如输入，输出，签名等。随后将它们计算得到一个 **交易哈希 (Tx Hash)** 用以标识一笔交易。

Bitcoin Block Header



一个区块包含多笔交易，这些Tx Hash将会被组织成为默克尔树（Merkle Tree），最终生成默克尔根（Merkle Root），作为“整个区块交易集合的哈希摘要”，存放在区块头（header）里

下图中， $\text{root} = \text{Hash}(H12, H34)$ 作为默克尔根包含了所有交易信息的摘要

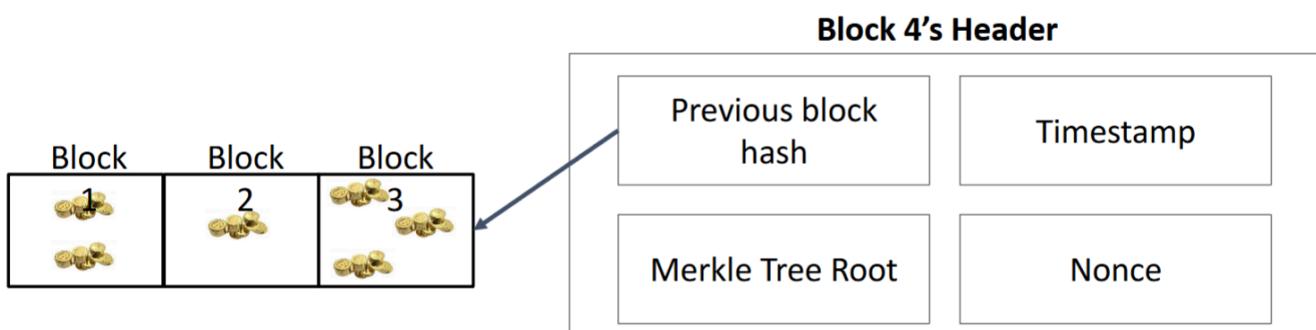


通过引入默克尔根，轻节点如手机就无需对整个区块进行验证，而是只需要获知哈希路径就可以验证

例如要验证 $T2$ 是否在块中，只需要请求全节点发送 $H1$ 和 $H34$ ，重建计算 $H2$ ，再重建计算 $H12$ ，随后使用 $H12$ 和 $H34$ 计算得到 root' ，与已有区块头中的 root 对比即可验证

如何串联？

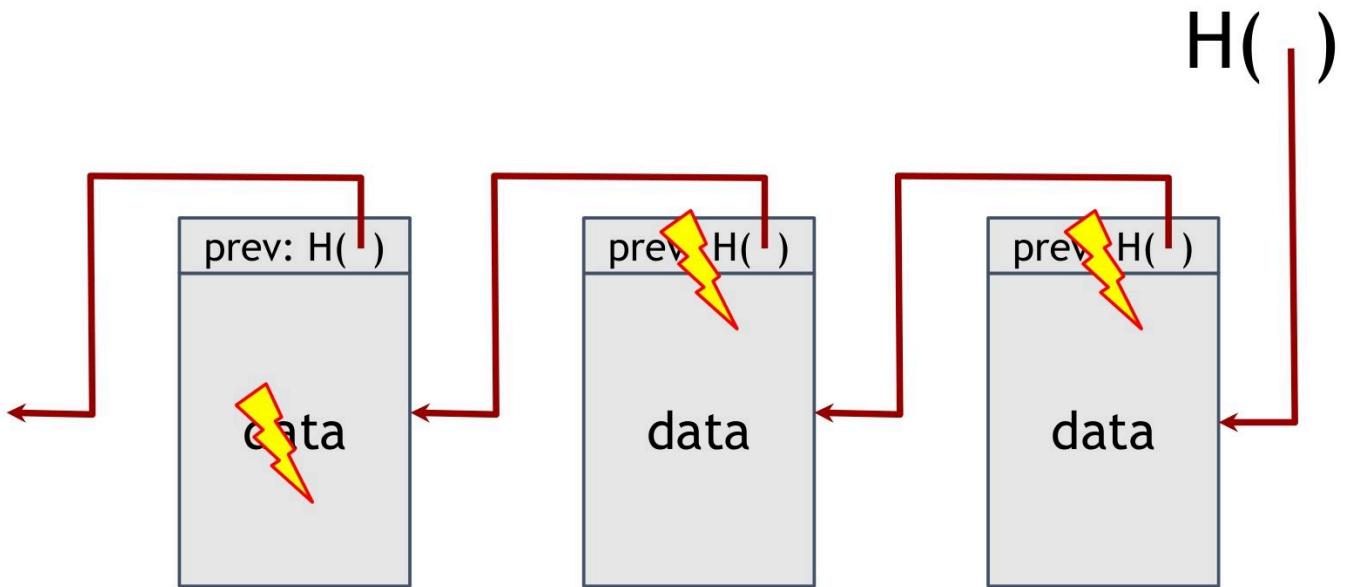
通过存储前一个区块头的哈希值串联



而这种串联结构可以检测篡改 tampering

如果篡改Block 2 的 “data” (交易数据或区块内容)，那么Block 2 的区块头哈希会巨变。而Block 3 的 `Previous block hash` 是 Block 2 的旧哈希，从而导致Block 3也失效.....

以此类推，后面所有区块的哈希链都会断裂，因此 篡改者必须重新计算“被篡改区块之后所有区块的哈希”，难度等同于 51% 算力攻击



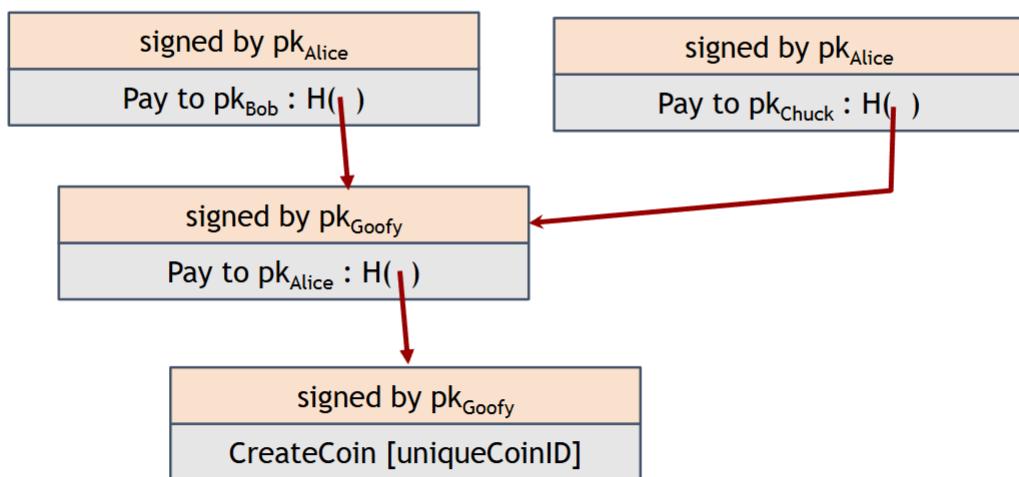
Digital Signature

在区块链中，用户用其私钥对消息进行签名，其他所有人都可以用用户的公钥对签名进行验证

用户的地址是Hash(PK)，用私钥可证明自己是地址的主人

Double-Spending Attack

攻击者试图让同一笔加密货币UTXO，同时支付给两个不同收款人，破坏“货币只能花一次”的规则



在中心化的银行系统里，通过账本ledger来记录交易，防止双花

而在非中心化的比特币中，通过以下方法防止双花：

- **多副本冗余 (Multiple complete copies)**：每个全节点（如矿工、大钱包服务商）都保存完整的区块链账本
- **公开可见性 (Public visibility)**：所有交易对全网公开
- **不可篡改性 (Immutable history)**：依赖 **默克尔根 (Merkle Root)** 和 **哈希指针 (Hash Pointer)** 实现
- **去中心化同步 (Quasi real-time synchronization)**：比特币用 **中本聪共识 (最长链规则)**，让分布式节点“最终达成一致”

中本聪共识 (Nakamoto Consensus)

全球分布的节点（矿工、全节点），如何对“区块链的状态（哪个链是最长链、哪些交易有效）”达成一致？

只认可最长链，而矿工的目标是“获得区块奖励”，所以会优先在 **最长链** 上挖矿（因为最长链最可能成为最终的权威链）

Identity Management

随机数d作为私钥

$P = dG$ (G 是椭圆曲线的生成元，公开参数) 得到对应公钥

账户地址 (Account): $A = \text{Hash}(P)$

每个用户可以有多个地址

拥有身份后，可以用私钥对交易签名 → 证明“这笔交易是你发起的”

此外，区块链交易包含 **发送者&签名、接收者、金额、手续费、附加数据**，签名作为交易的组成部分

Transaction Broadcast

1. 签名交易，随后广播到最近的区块链节点
2. 节点收到后使用对应公钥验证，有效则转发给其余节点直到全网扩散
3. 矿工收集到交易，打包成区块，随后全网同步更新账本。由此，交易得以被记录并永久保存

Mining

挖矿的过程就是将交易打包到区块链上的过程

1. 比特币矿工通过 **P2P 网络收集交易**，验证后暂存到自己的内存池 **mempool**，等待被打包进区块。未确认的交易全网可见
2. 每个矿工从自己的内存池 (mempool) 中选择未确认交易，组成一个“候选区块”，其中也包含了区块头（含前一区块哈希、默克尔根、时间戳、难度目标、随机数 **Nonce**）和区块体（交易列表）。

3. 矿工通过“算力竞争”争夺区块创建权——不断调整区块头中的Nonce，计算区块头哈希，直到哈希值满足“难度目标”(如哈希值前N位为0)。第一个找到符合条件Nonce的矿工获得区块创建权，随后广播给全网进行验证

激励 Incentive：当矿工成功打包区块（成为 leader），会获得一笔 **Coinbase 交易**（区块里的第一笔交易）。此外，还有交易发送者给的**手续费 transaction fee**

Coinbase 交易是比特币“发行新币”的唯一途径，只有挖矿能“创造新币”

矿工通常会根据手续费高低选择交易顺序order

比特币用 **工作量证明 (Proof of Work, PoW)** 来决定谁能创建区块，矿工不断调整nonce来满足**难度目标 target**，从而获得区块创建权。

比特币会动态调整**难度目标 Target**，使得出块速度稳定在10分钟左右，这个时间足够全网挖矿，验证和广播

在比特币系统中，一人一票（one vote per machine）不可行，因为单用户可以创建大量身份。因此使用PoW，根据算力体现诚意。攻击者必须掌握超过51%算力才能篡改账本，但成本过高甚至高于了老实挖矿的成本

两大核心安全属性：

- **安全性 Safety**（保证持久性）：要移除（篡改）一个已上链的区块，攻击者需要**控制超过 51% 的全网算力**
- **活性 Liveness**（保证可用性）：要阻止一笔交易被打包上链（审查交易），攻击者需要**控制超过 51% 的全网算力**

Nakamoto Consensus

中本聪共识的核心是“**最长链规则**”：诚实矿工永远在最长链上挖矿，让全网算力自动收敛到同一条链

既保障了区块链的“不可篡改性”(篡改需超过 51% 算力)，又让账本同步简单高效（无需中心化协调）

矿工永远选择**自己看到的最长有效链**，即使网络临时分叉，矿工们“各自选择最长链挖矿”的行为，会让全网**概率性地收敛到同一条链**

Node Types

Full Blockchain Node：“全账本节点”：存储并维护完整区块链账本，负责网络中交易和区块的转发、验证

Lightweight (SPV) wallet：“轻钱包节点”：通过“简单支付验证 (SPV)”验证交易，无需下载完整账本

比特币钱包的本质是**管理 ECDSA 密钥对**：私钥+公钥

UTXO

UTXO, Unspent Transaction Output 未花费交易输出

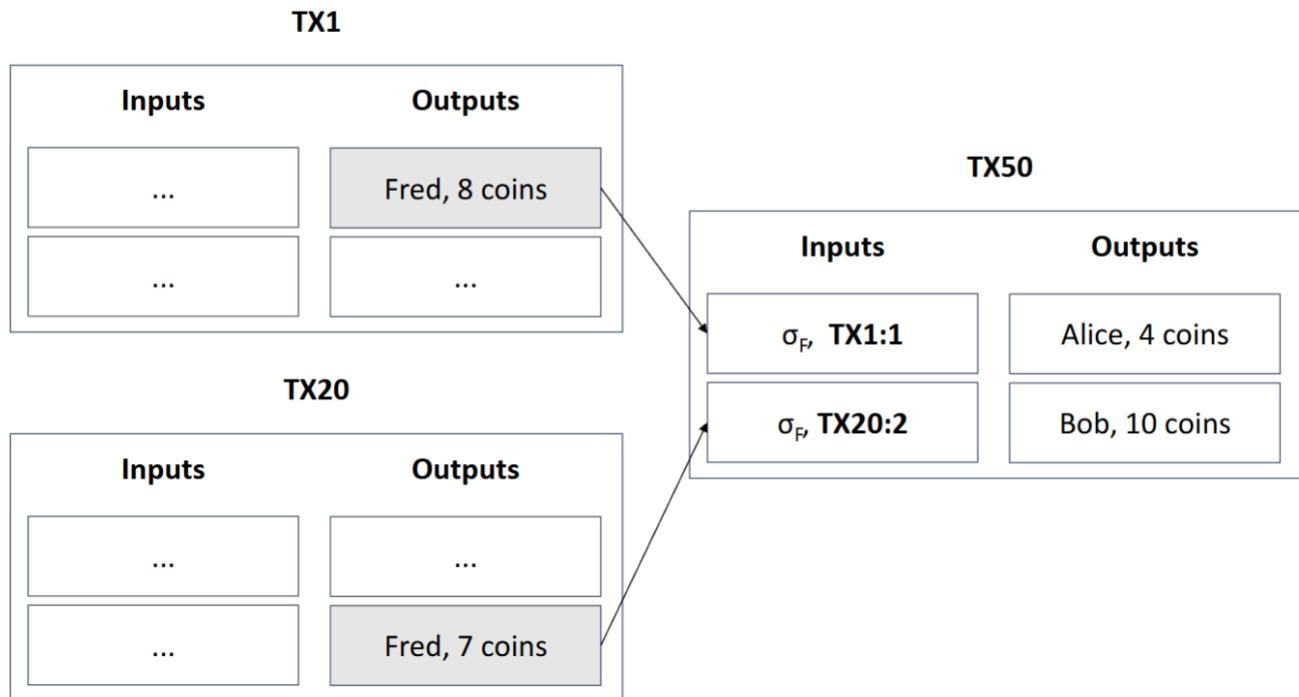
UTXO 是“未花费的交易输出”→ 本质是区块链账本里的一条数据库记录，记录“谁能花这笔钱，以及有多少钱”

ID	Script	Coins
hash(tx):2	Bob must sign the tx	8

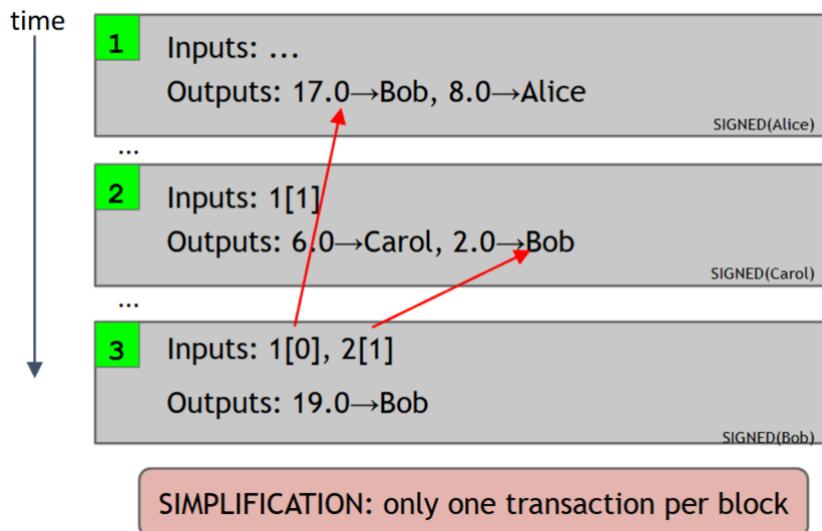
UTXO 模型的“交易 = 输入 + 输出”，销毁旧 UTXO（输入），创建新 UTXO（输出）

如下图中：

1. Fred 用签名证明“我有权花 TX1:1 和 TX20:2 这两个 UTXO”→ 这两个 UTXO 被“销毁”(从账本中移除)
2. 交易生成新的 UTXO (Alice 的 4 币、Bob 的 10 币) → 这些新UTXO 进入账本，等待被后续交易消耗。



比特币没有“账户余额”的概念，所有价值转移都是“UTXO 的重组”



Lecture 2 - consensus

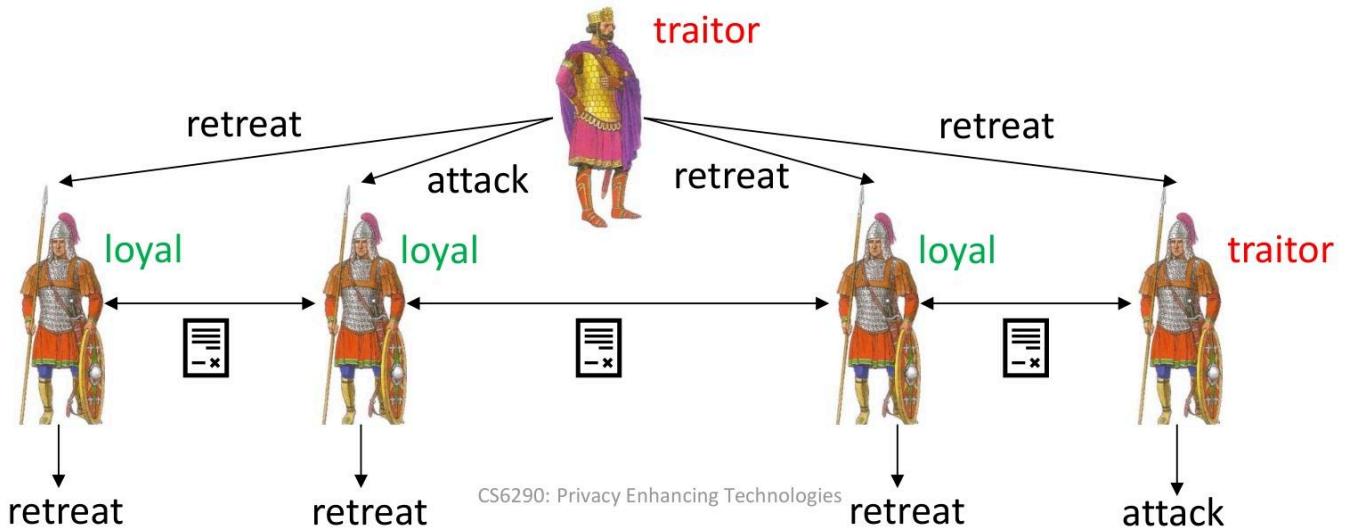
通过投票达成共识

但如果没有任何一个可信的第三方负责统票，如何让多个参与者通过自主交互达成一致？

分布式共识：让所有“正确节点（correct nodes）”对同一个值达成一致（decide on the same value）且最终共识的值，必须由某个**正确节点**提出（排除错误 / 恶意节点的无效提议）

Byzantine Generals Problem

系统中包含指挥官commander和将军general，其中可能有忠诚或者叛徒



最终目标是让所有忠诚将军都能在有叛徒干扰的情况下达成一致的行动

Adversary

将拜占庭将军问题扩展到分布式共识系统中

将军 General → 节点 Nodes

指挥官 Commander → 领导者 Leader

忠诚 Loyal → 诚实 Honest

叛徒 Traitor → 敌手 Adversary

敌手的能力：

Crash faults: 不发送任何消息

Omission faults: 选择性发送或丢弃消息

Byzantine faults: 撒谎，篡改消息

为了确保能达到共识，敌手数量必须有上限，例如 $f < n$, $f < n/3$

Communication

分布式系统中节点之间用**数字签名** + PKI 验明身份

此机制保证了敌手没法冒充诚实节点，这是共识协议能运行的前提

Asynchronous Model

分布式系统中网络环境分为三种：

- 同步网络 (**synchronous network**)：消息最多延迟 Δ 轮，超时算故障
- 异步网络 (**asynchronous network**)：消息可以拖很久，但不能永远不发
- 部分同步网络 (**partial synchronous**)：现实中最常见，比如网络有时快、有时慢，但不会一直卡

以上网络环境的区别在于消息多久到，但消息最终都一定会到

这样的网络环境为分布式共识的达成进一步加大了难度

Byzantine Broadcast (BB)

拜占庭广播：

- n 个固定节点，1 个是 **Leader** (领导者)
- 有 f 个恶意节点 (adversarial)，剩下的是 **honest** (诚实节点)
- 领导者的输入是 **0** 或 **1** (比如发“进攻”或“撤退”命令)

- 同步网络（消息延迟有限、可预测）

目标：

- **Agreement**: 所有诚实节点必须输出相同值（不能有的输出 0，有的输出 1）
- **Validity**: 领导者诚实时 → 所有诚实节点必须输出领导者的输入值；即使领导者是叛徒，诚实节点也得通过通信，达成 Agreement（行动一致）
- **Termination**: 所有诚实节点必须最终输出一个值（不能无限卡壳、不决策）

通过签名链 $v' : i, j, \dots, l, k$ 来验证发送人

Dolev-Strong

拜占庭广播的经典解决方案，用 $f+1$ 轮 消息对抗 f 个恶意节点

但 Dolev-Strong 协议有个强假设：消息必须在下一轮准时送达（messages get delivered by next round）而现实中往往不可能

$n > 3f$

假设有 n 个节点，其中有 f 个不诚实节点（可能是无意故障，故意撒谎或者故意不响应）

用户至少会收到 $n-f$ 个响应，并且不能继续等而是必须基于这些响应做判断。而这其中有可能包含了 f 个恶意节点的谎言

如果要让客户端能“过滤恶意响应”，必须满足 $n > 3f$ （比如 $n = 3f + 1$ ），这样的话诚实节点的数量 $n - f > 2f$ 。当收到 $n-f$ 个消息时，即使其中有 f 个谎言，诚实的响应数量仍大于谎言的数量

由此，要让客户端能“过滤恶意响应”，必须满足 $n > 3f$ （比如 $n = 3f + 1$ ）

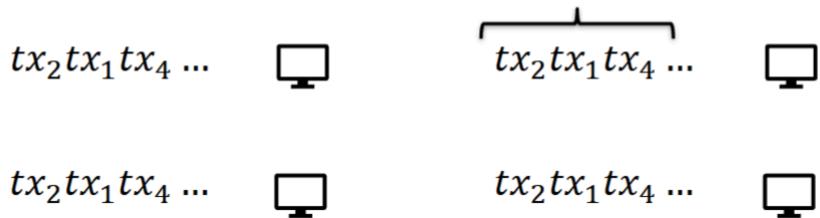
当 $n = 3f + 1$ ，客户端至少收到 $2f + 1$ 个响应，诚实响应数量一定大于其中谎言数量

State Machine Replication (SMR)

区块链本质是“分布式状态机”，靠共享账本让多节点同步执行交易、维护状态，替代传统中心化机构的“单点控制”，实现去中心化的可信协作

Blockchain (State Machine Replication)

Log (Ledger): an ever-growing, linearly-ordered sequence of transactions.



在SMR中有两大核心角色：

- **副本节点 (Replicas, 如矿工):** 接收客户端提交的交易并执行 SMR 协议，通过共识算法（如 PBFT、PoW）协商确定统一的交易日志 (Log) 顺序
- **客户端 (Clients):** 无需参与共识，只需从副本节点读取已达成一致的日志

最终目标是确保所有客户端学习到相同的交易日志 (Clients learn the same log)

在现实中，区块链钱包（如比特币、以太坊钱包）的实际工作模式就是类似SMR：

- 钱包向多个节点（如全节点、矿池节点）查询交易记录和账户余额
- 不直接参与区块共识，仅作为“数据消费者”读取已达成一致的区块链账本
- 确保用户无论连接哪个节点，看到的交易历史和余额都是一致的

SMR Security

在SMR中，日志必须按顺序拼接，且诚实节点的日志要满足严格的前缀关系

两个序列 **A** 和 **B** 是一致的 (consistent) ，当且仅当： **A** \leq **B** (**A** 是 **B** 的前缀) 或 **B** \leq **A** (**B** 是 **A** 的前缀)
或两者同时成立

例如： $LOG^{Alice} = tx_1tx_2tx_3tx_4$, $LOG^{Bob} = tx_1tx_2tx_3$, 则一致

状态机复制 (SMR) 安全协议的两大核心保障：

- **安全性 (Safety / Consistency):** 保证所有客户端的交易日志 (Log) 前缀一致，类似“拜占庭共识的 Agreement”
- **活性 (Liveness):** 保证“诚实节点提交的交易，最终会被所有客户端看到”，类似“拜占庭共识的 Validity 和 Termination”

安全性保证日志一致 (不能分叉从而防止双花)，活性保证交易最终确认 (不能卡壳)

SMR vs. Byzantine Broadcast

区别：

- 单次共识 vs. 多次共识 (Single shot vs. Multi-shot)：BB中每个节点只输出一个值，而SMR中每个客户端持续输出交易日志 (Log)
- 学习者 (Learner)：BB中每个节点都可以是学习者参与到共识过程中，而SMR中有固定的客户端作为学习者

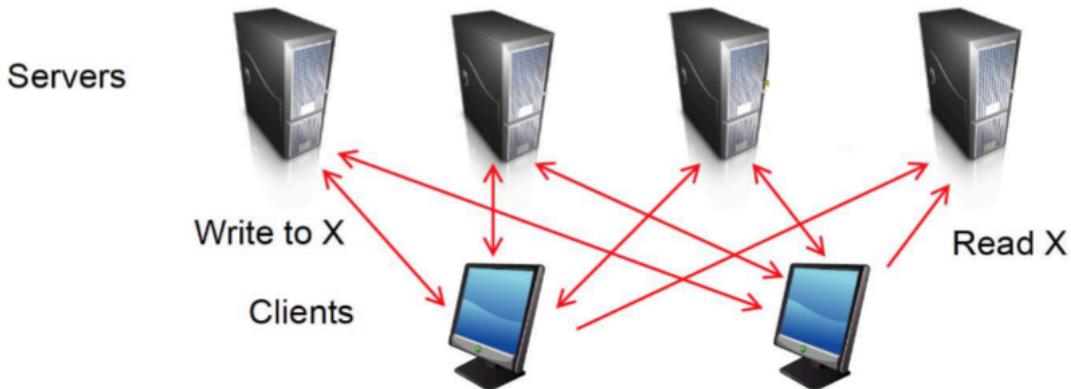
Practical Byzantine Fault Tolerance (PBFT)

PBFT 专门为“多副本状态机”设计，目标是让多个节点（副本）同步执行操作，保持状态一致。

其能容忍 f 个恶意节点（拜占庭故障），条件是：

- 总副本数 $n = 3f + 1$ (比如 $f=1$ 时，需要 4 个副本)；
- 网络模型是部分同步 (partial synchronous) —— 延迟存在但有限 (只是节点不知道具体延迟多大)。

如下图，客户端发起操作，随后服务器们通过PBFT协议同步操作，使得任意客户端都能访问到操作后的相同副本

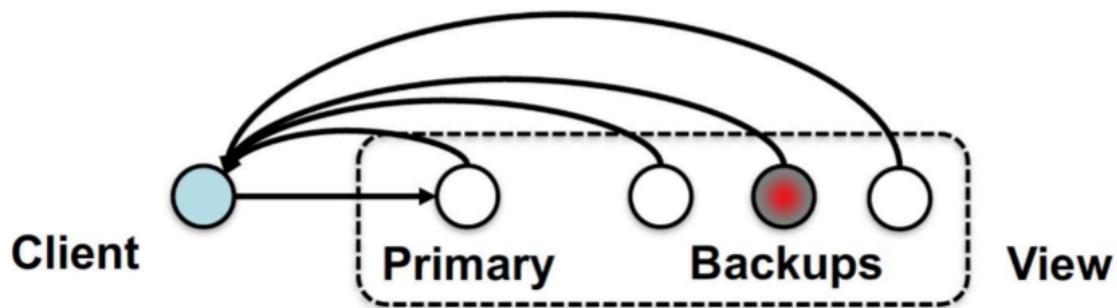


更具体地，PBFT中存在**主节点 (Primary)**，负责“执行操作”(performs operations)，比如接收客户端请求、组织交易排序、发起共识流程，是共识的“领导者”，但也可能作恶（所以需要备份节点监控）

另外还有**备份节点 (Backups)**，负责监控主节点的行为，如果发现主节点故障（比如不响应、发恶意消息、调换操作顺序），会触发**视图切换 (view change)** —— 换一个主节点继续共识

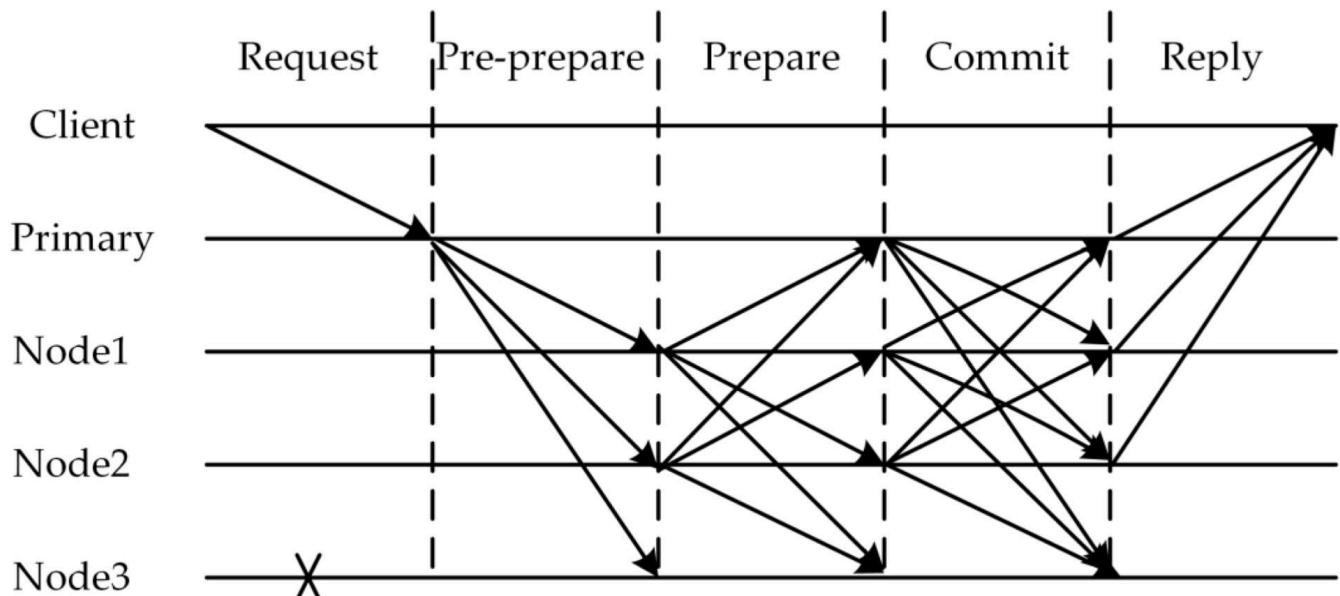
具体地：

1. 客户端发请求 (Client → Primary)
2. 主节点广播请求 (Primary → Backups)
3. 副本执行并响应 (Replicas → Client)
4. 客户端收集响应 (Client waits for $f+1$ 个一致结果)



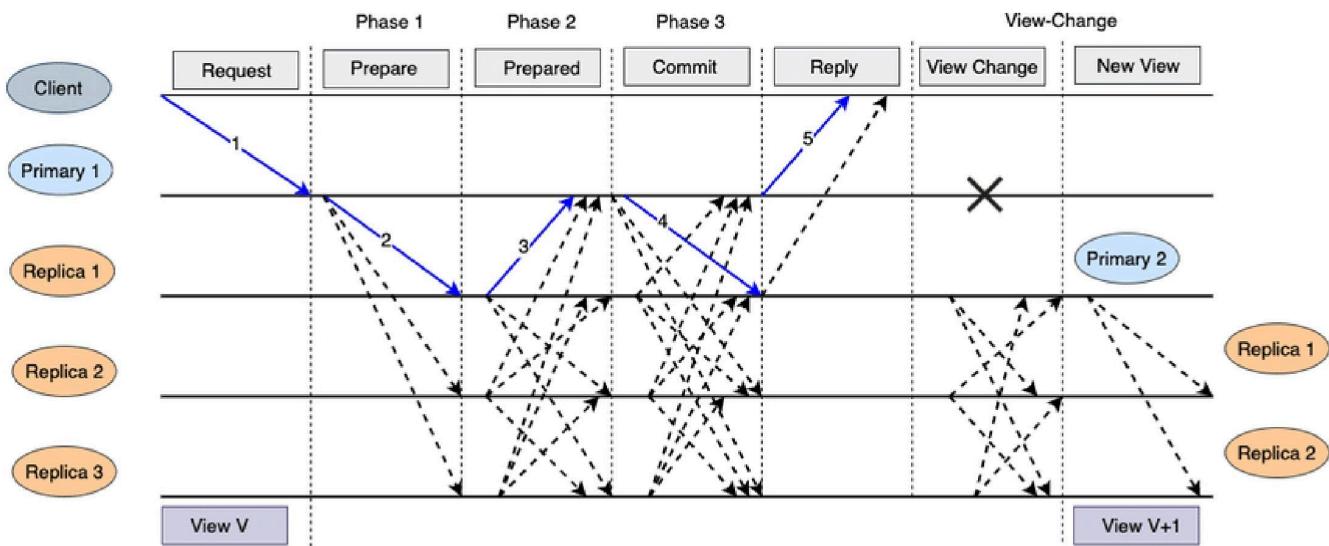
当主节点（Primary）诚实、不故障时，PBFT 用两阶段协议处理请求 r ，核心是通过多轮共识，让足够多的诚实节点 ($f+1$ 个) 对请求的“执行顺序”和“执行结果”达成一致：

- 阶段 1：Pre-prepare + Prepare（确定“执行顺序”），确保至少 $f+1$ 个诚实节点，同意“如果请求 r 在视图 v 中执行，它的序列号是 n ”
- 阶段 2：Prepare + Commit（确认“执行结果”）：确保至少 $f+1$ 个诚实节点，同意“请求 r 已经在视图 v 中，以序列号 n 执行完毕”



在视图切换（View Change）中，也包含两个阶段：

- 阶段1：发起视图切换请求（viewchange），怀疑主节点故障的备份节点，给所有副本节点（包括主节点和其他备份）发 viewchange 请求，其他节点收到后，需要“确认”(acks) 这个请求，形成共识
- 阶段2：选举新主节点，新主节点需要收集 $2f + 1$ 个响应，随后新主节点发 new-view 消息宣布进入新视图



Permissioned(less) SMR

许可制 permission 与否取决于节点是否可自由进入

对于许可制SMR：委员会已知易被攻击、节点多了通信爆炸、诚实多数难维持、领导者可预测易被贿赂，但不惧女巫攻击

而对于无许可（Permissionless）环境下，任何人都能自由加入 / 退出网络（open participation），典型代表是比特币、以太坊等公链

两大核心挑战（Challenges）

1. 女巫攻击（Sybil nodes）：

- 攻击者能创建大量虚假节点（Sybil nodes），试图控制协议（比如 51% 攻击）。

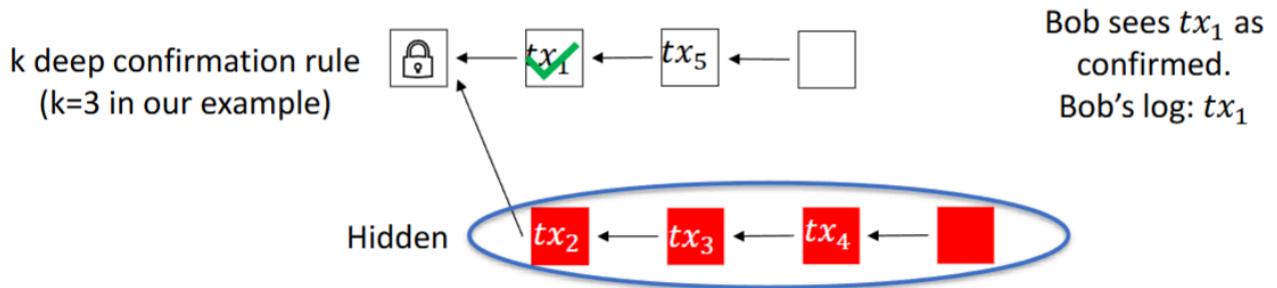
2. 节点动态性（Honest nodes come and go）：

- 诚实节点可随时加入 / 退出，网络节点数量动态变化，难维持共识。

对于比特币，其用 PoW（工作量证明）对抗女巫攻击，当攻击者算力 $<50\%$ 时能满足安全性和活性；通过“最长链共识”，让诚实节点即使随时进出，仍能维持区块共识

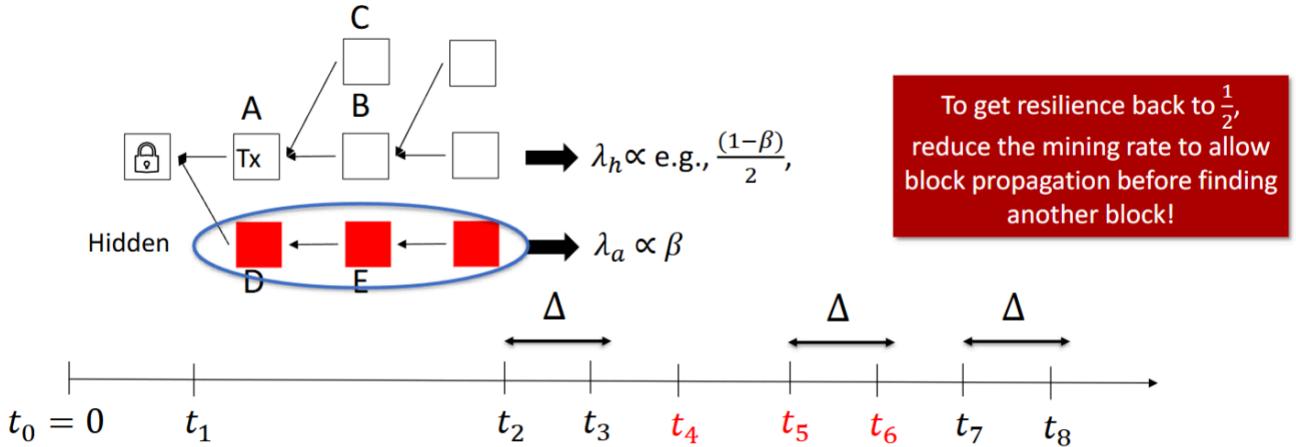
Nakamoto's Private Attack

如果攻击者控制的算力占比 $\geq 50\%$ ，此时攻击者的出块速率 \geq 诚实节点，那么攻击者具备“隐藏区块、制造分叉”的能力，从而创造了新的最长链，实现双花攻击



而当因为网络延迟 (network delay) 产生分叉 (Forking) 时，诚实节点因分叉，算力分散，实际出块速率低于 $(1 - \beta)$ ，而攻击者的隐藏区块不受影响，则有：

$$\beta \geq \frac{(1-\beta)}{2} \rightarrow \text{化简得 } \beta \geq 1/3, \text{ 即可让 } \lambda_a \geq \lambda_h$$



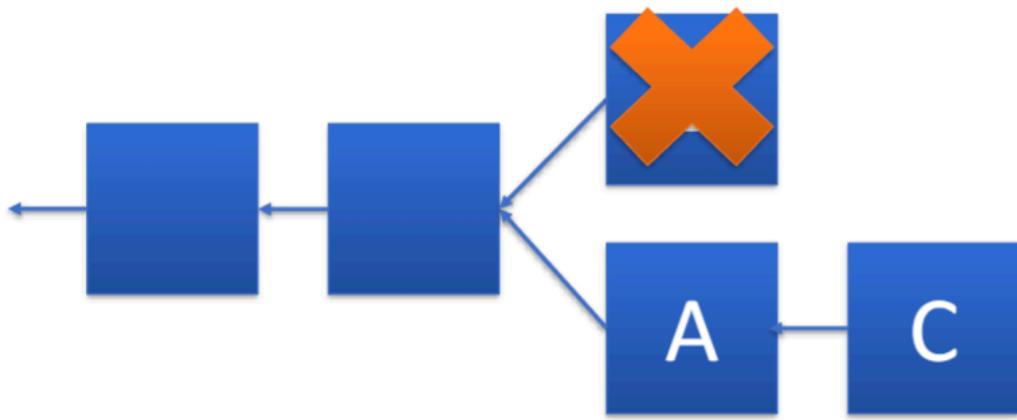
为防止分叉，应降低挖矿速率 (mining rate)，让区块在网络中充分传播后，再挖下一个块。正因如此，比特币通常每10分钟才产生一个块

中本聪共识的特性：匿名 Anonymous、动态、抗贿赂 Anti-bribing、容错 50% Up to ½ corruptions，但代价是慢、耗资源、无最终性

Forks and Orphans

合法挖出的区块因“没加入最长链”成为孤块，浪费算力和能源

Orphaned block



Software Fork

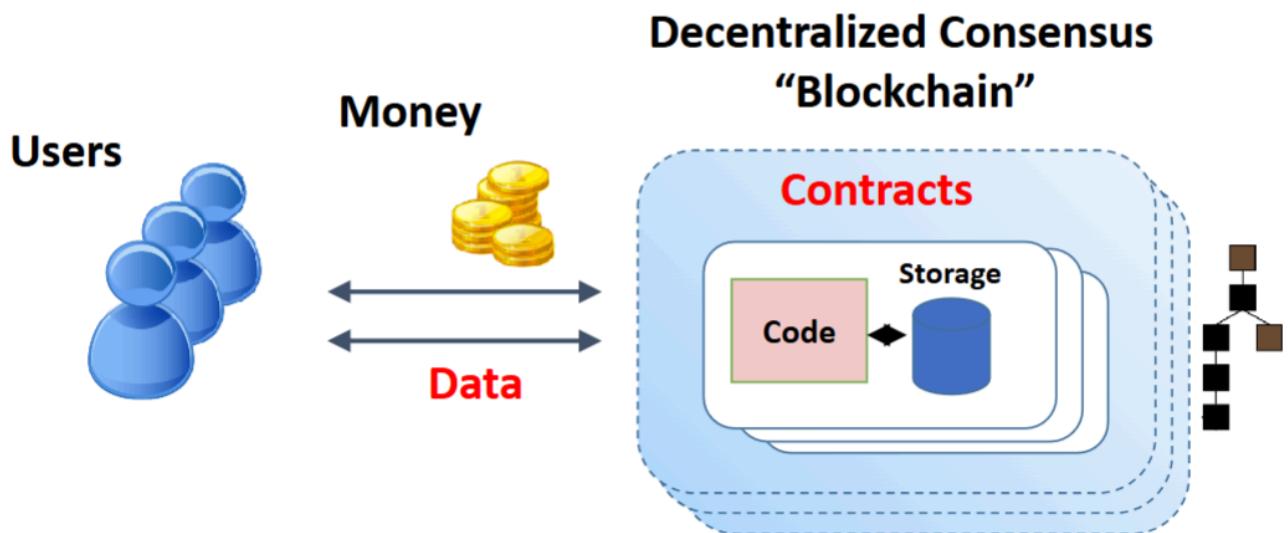
区块链根据如何达成“改规则”的共识，可能形成硬分叉和软分叉

软分叉 (Soft Fork) 的核心是：软分叉是“向后兼容”的协议升级，新规则更严格，但旧节点会认为新块合法，无需强制升级。例如区块大小缩小（1MB→500KB），未升级的比特币节点看到450KB块，会判断“小于1MB限制，合法”，接受该块

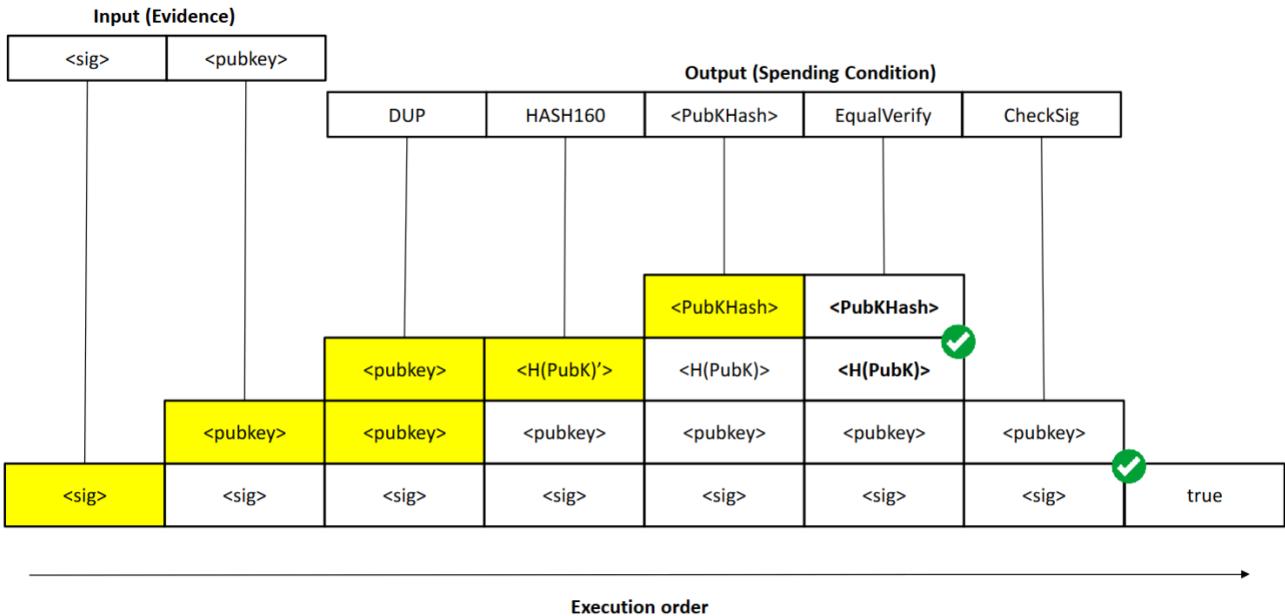
硬分叉 (Hard Fork) 核心是：硬分叉是“不向后兼容”的协议升级，新规则会让旧节点拒绝新块，导致链分裂，用户需主动选择是否升级。例如区块大小扩容（1MB→2MB），未升级的比特币节点看到2MB块，会判断“超过1MB限制，非法”，拒绝该块

Lecture 3 - ethereum

相比起比特币简单的代币交换应用 token exchange，ethereum具有智能合约（Contracts）来实现自定义功能
User-defined programs



比特币同样具有其脚本规则，是如图所示的栈式脚本验证。简洁、固定逻辑（围绕 UTXO 和脚本操作码）

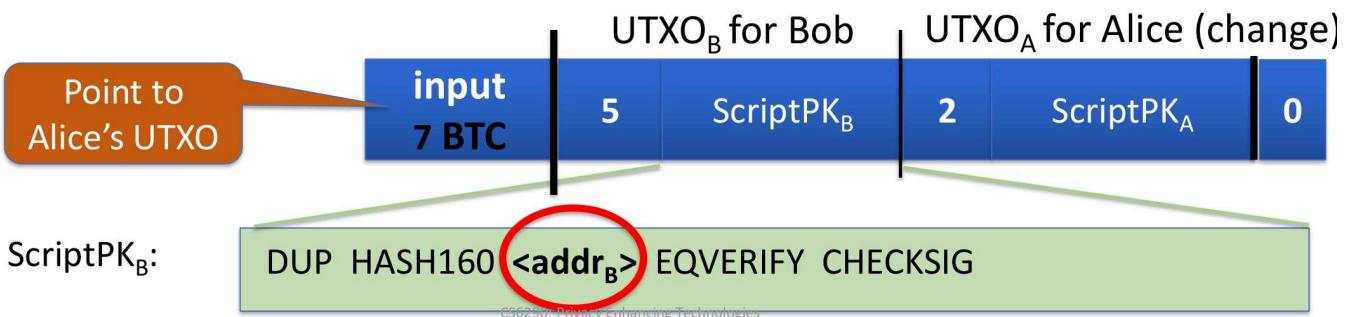


BitCoin——P2PKH

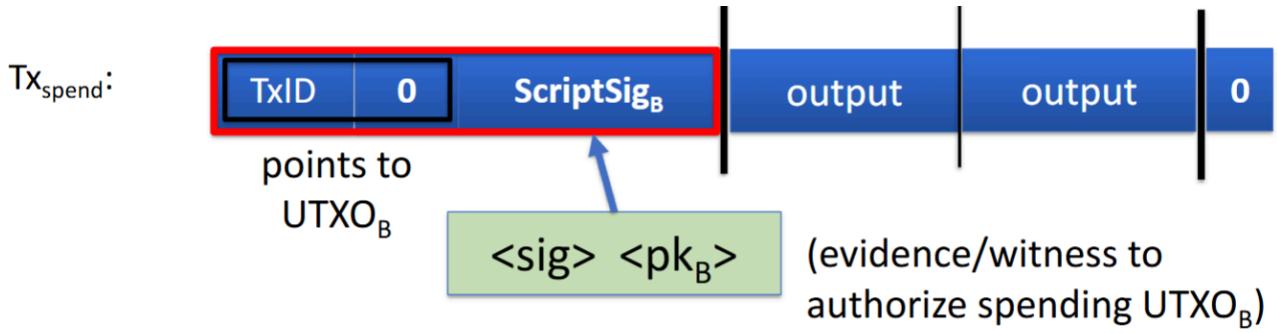
P2PKH (Pay-to-Public-Key-Hash, 支付给公钥哈希) 是比特币最基础的交易类型，通过“公钥哈希地址 - 栈式脚本验证 - UTXO 流转”三大机制，实现了无需中介的点对点价值转移。

如下图

1. Bob生成自己的公钥和私钥，通过公钥计算得到比特币地址
2. 输入是对Alice的UTXO的引用，其中包含了解锁脚本ScriptSig，Miner会运行脚本验证Alice的签名



之后，当Bob想要花费 $UTXO_B$ 时，矿工同样会进行Bob的签名验证工作



以上机制中有三个细节：

- **收款方公钥的延迟暴露机制：**Alice在创建 $UTXO_B$ 时不会放入Bob的公钥，只有由公钥产生的比特币地址。只有Bob需要花费时才会暴露其公钥
- **矿工不可篡改收款地址：**Alice签名时对整个 Tx 进行签名，其中就包含了收款地址

BitCoin——P2SH

P2SH (Pay-to-Script-Hash, 支付到脚本哈希)，是对P2PKH的扩展

付款方可以设定复杂赎回脚本（redeem script）作为比特币地址，而非仅简单的公钥哈希

这样可以支持复杂花费条件（如多签，时间锁等），在需要花费时，矿工验证脚本是否返回true，只有为true时才可花费

然而，比特币脚本 Bitcoin Scriptability 仍有局限性：

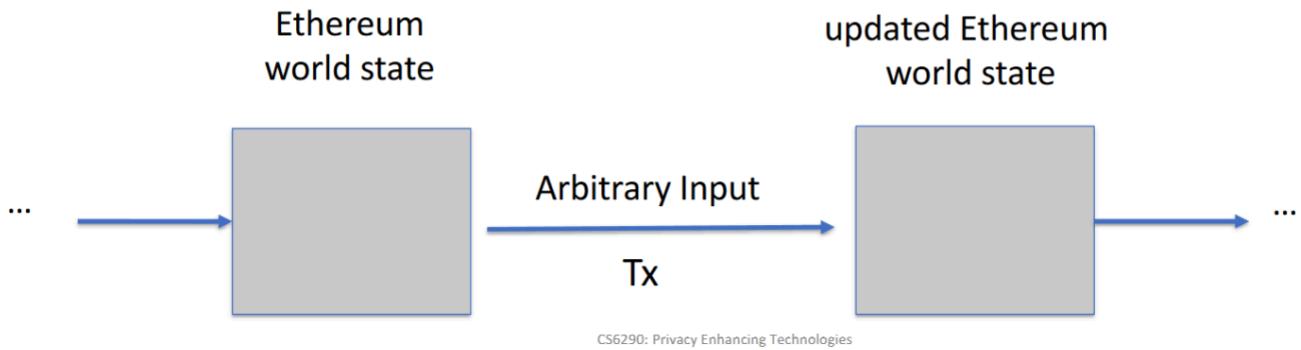
- 比特币脚本是“无状态 stateless”的，没有全局状态信息，导致脚本执行仍局限于单次交易，无法实现跨交易的状态连贯性（对比以太坊智能合约可通过 storage 存储状态变量）
- 比特币的 UTXO 是分散独立的，每个 UTXO 的花费条件仅作用于自身，无法对“同一用户的所有资产”或“某一类资产”施加统一规则

因此，比特币仍然只适用于简单价值转移

Ethereum——Transaction

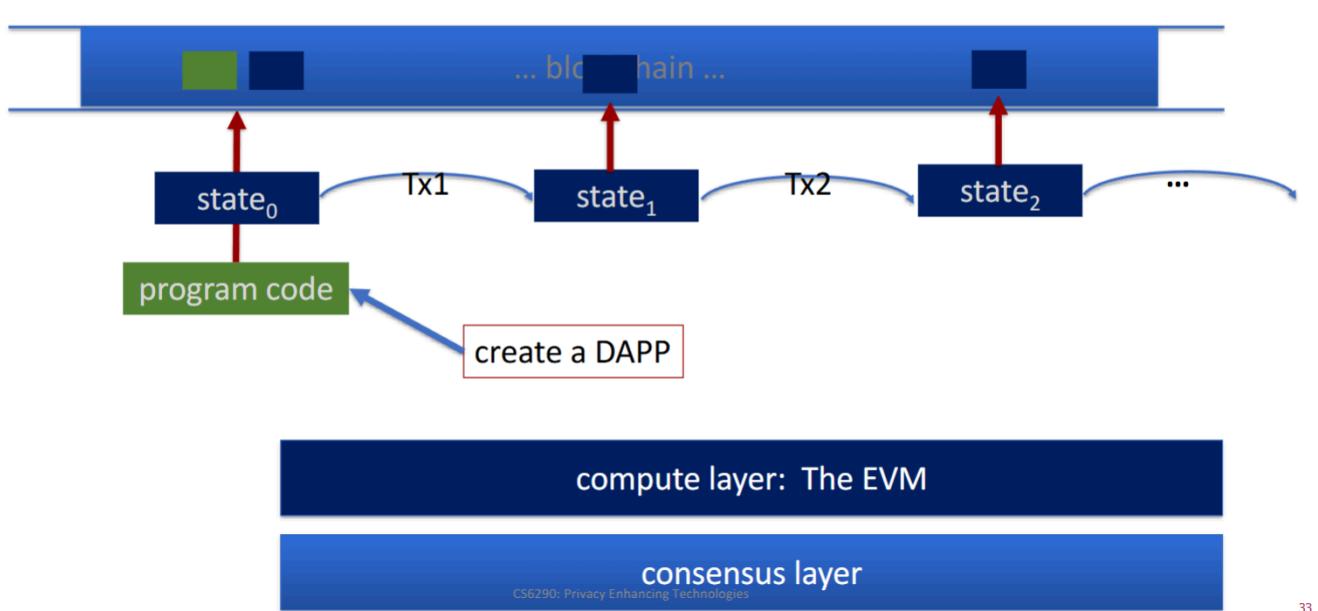
相比起比特币的简单世界状态（UTXO），固定输入和固定输出，Ethereum复杂灵活得多

Ethereum的“世界状态”是账户余额、智能合约存储（storage）等数据的集合，交易（Tx）可携带任意输入（Arbitrary Input），触发智能合约的完整逻辑（如转账、修改合约状态变量、调用复杂函数）。因此Ethereum可以执行智能合约的全部逻辑（如 DeFi 借贷、NFT 铸造），直接更新账户余额、合约存储等全局状态。



下图为区块链上运行程序（DAPP，去中心化应用）的架构示意图：

- 区块链中存储状态以及程序代码
- 用户运行程序时，在计算层中完成状态更新，随后通过共识层将新状态写入区块链



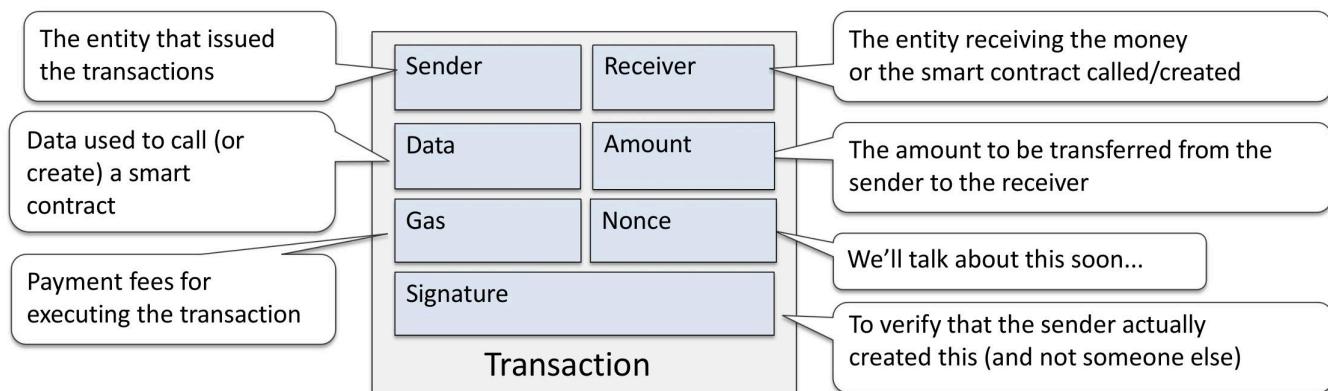
以太坊中数据库以账户的形式存在，所有节点维护同一套数据库，其中记录所有账户状态，包含：

- 智能合约账户（如`<hash>`）：由代码控制，部署后自动执行逻辑。其中包含了特殊的`code`（可执行逻辑）和`storage`（可存储状态变量）。其`Balance`存储ETH余额，`Nonce`存储创建子条约数量
- 普通账户（EOA，如 Alice）：由用户私钥控制，用于收发ETH、发起交易。无`storage`和`code`，`Balance`存储ETH余额，`Nonce`存储已发交易数量（防重放攻击）

Address	Storage	Code	Balance	Nonce
<code><hash></code>	<code><variables></code>	<code><code></code>	8	100
Alice	N/A	0	10	312

每笔交易包含如下核心字段：

- **Sender (发送者)**: 发起交易的实体（通常是 EOA 账户，由私钥控制）
- **Receiver (接收者)**: 接收资金的实体或被调用 / 创建的智能合约地址，若接收者是合约账户，交易将触发合约代码执行
- **Data (数据)**: 用于调用或创建智能合约的输入数据，例如调用合约函数时传递的参数（如 `transfer(to, amount)` 中的地址和金额），普通 ETH 转账时该字段为空
- **Amount (金额)**: 从发送者转移到接收者的资金量，单位为 Wei ($1 \text{ Wei} = 10^{-18} \text{ ETH}$)，直接对应账户模型中“修改账户 Balance 字段”的状态更新
- **Gas (gas 费)**: 执行交易的手续费，用于支付节点计算资源消耗，确保交易被矿工打包
- **Nonce (随机数)**: 发送者账户的交易计数器，用于防止重放攻击
- **Signature (签名)**: 发送者用私钥对交易的签名，用于验证交易确实由发送者创建

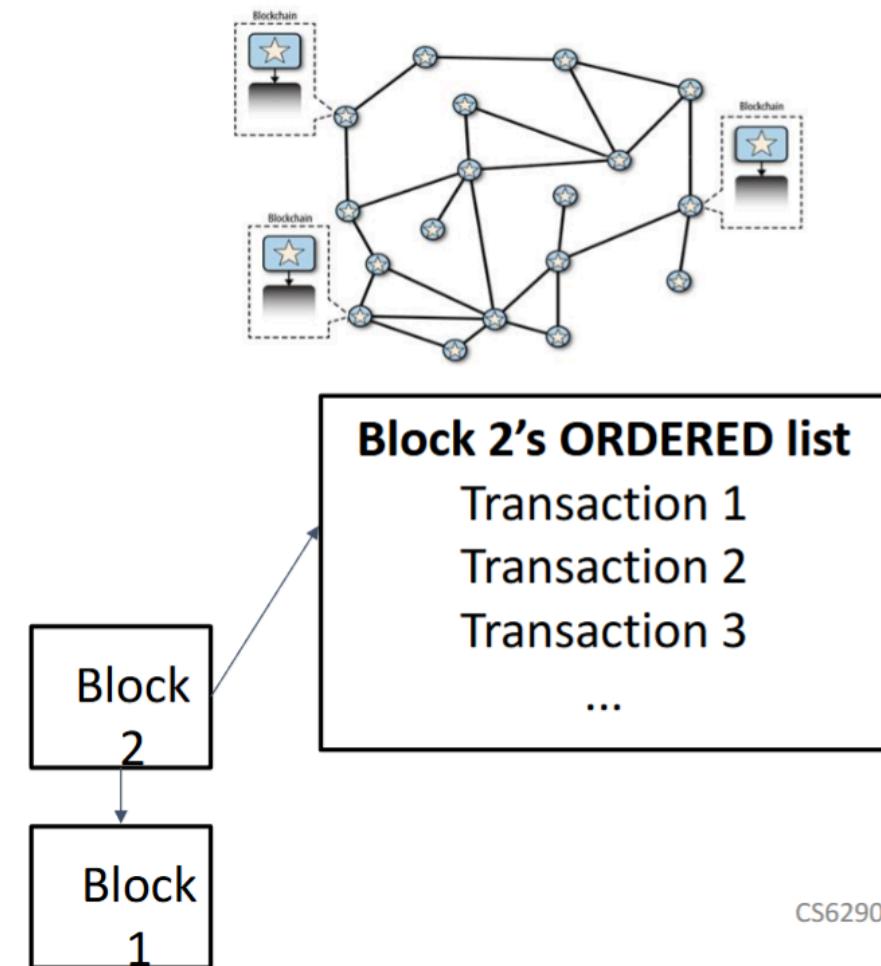


通过账户 Nonce 与交易 Nonce 的严格匹配，确保每笔交易仅能被执行一次，同时维护交易的顺序性

在以太坊中，为确保智能合约执行后所有节点达成一致状态，需要：

- **交易的有序性 agreed order of transactions**: 区块内交易按固定顺序执行
- **函数的确定性 deterministic functions**: 合约代码包含“输入相同则输出必然相同”的逻辑

通过以上设计，避免了因执行顺序不同而导致的世界状态不同

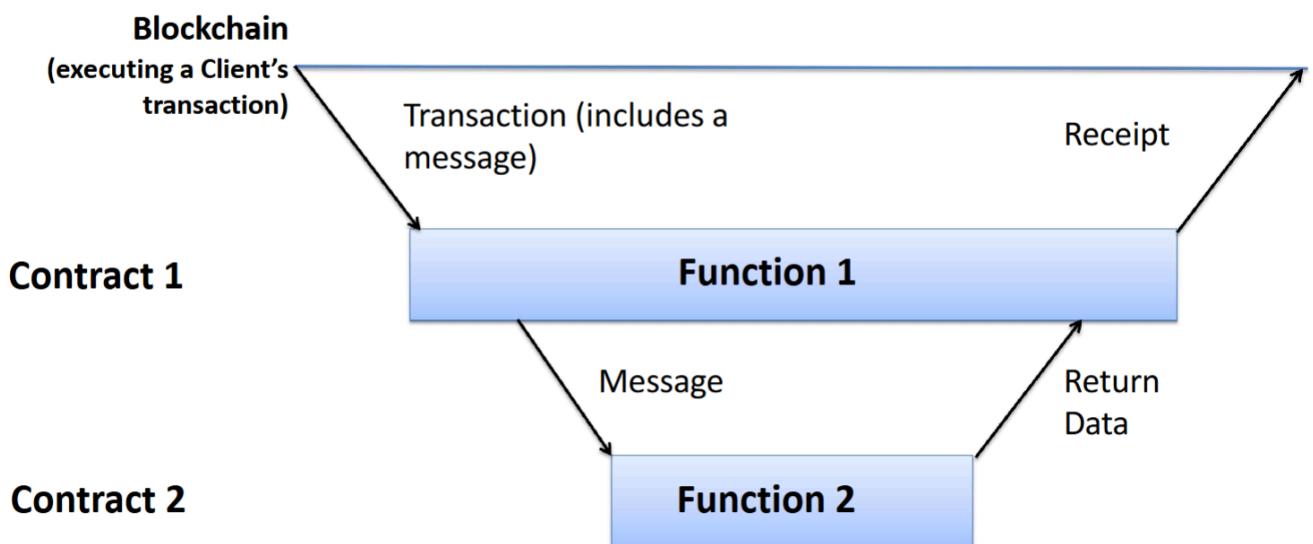


要验证交易是否有效，除验证Sender的余额Balance中是否能覆盖转账额和手续费，验证签名以外，还要检测Nonce是否正确匹配

Ethereum——Function Calling

当交易发起后，通过传递message，可以跨合约调用函数

但这个过程中每一步调用都消耗 Gas，并遵循 确定性执行 规则



合约可发起虚拟交易Virtual Tx，无需签名且可组合

因此用户发起的一笔交易（One Tx from user） 可触发多笔虚拟交易，形成“交易链”(如用户→合约 A→合约 B→合约 C→用户)

交易发生后，会生成对应的交易收据Transaction Receipts，包含资源消耗 GasUsed、状态 Status、日志 Log等

Ethereum——Gas

以太坊中通过Gas费用激励矿工

若发起者的余额足以支撑Gas费用，交易顺利进行，否则回滚交易，但仍然要支付部分Gas

以太坊中的交易执行根据Gas条件决定是否成功，这个过程中有以下特点：

- 顺序性 Sequential：避免冲突，统一执行顺序；
- 原子回滚 Rollback：容错，防止无效状态；
- 状态继承 State Inherit：支持复杂交互，让交易可以“依赖前序结果”。

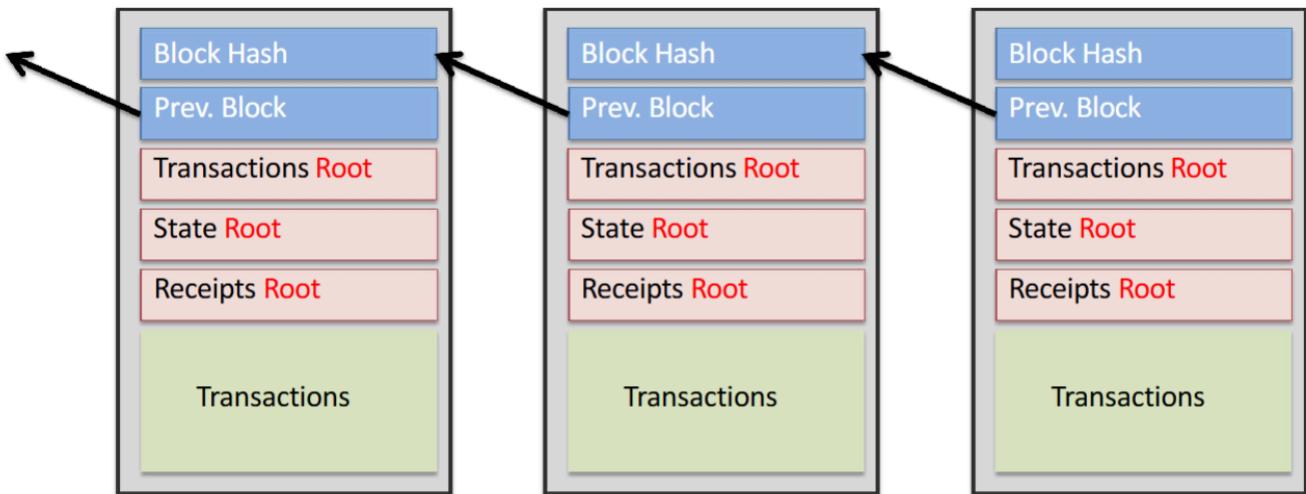
Ethereum——Block

以太坊区块的生成与验证，是“交易执行→状态更新→共识保障”的完整流程：

- 提议者串行执行交易，记录世界状态并打包成区块；
- 验证者重放新区块内的交易，验证状态真实性；
- 签名机制结合 PoS，得到足够签名后区块被最终确认，实现去中心化共识

以太坊的链中包含交易，世界状态，收据的Merkle根，支持快速验证

链式哈希结构避免了篡改



另外，以太坊的共识机制由PoW升级到PoS(Proof-of-Stake)，大大提升了区块生成速度

Ethereum——Script Language

开发者使用高级语言编写脚本，然后编译成字节码存储在链上，由EVM执行



其中Solidity是一门JavaScript-like的高级语言

```

pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public constant returns (uint) {
        return storedData;
    }
}

```

Ethereum——EVM

EVM是以太坊智能合约的“运行时环境”，开发者用 Solidity 等写代码 → 编译为 **EVM 字节码**；验证者（矿工 / PoS 节点）通过 EVM 执行字节码

EVM根据存储位置不同收取不同费用：

- Persistent storage (on blockchain): SLOAD, SSTORE (expensive)
- Volatile memory (for single Tx): MLOAD, MSTORE (cheap)

Ethereum——Functions

函数中常见数据类型：

- `uint256`：256 位无符号整数
- `address`：以太坊账户地址（外部账户 EOA 或合约账户 CA）
- `bytes32`：32 字节固定长度的字节数组
- `bool`：布尔类型

Require VS Assert：都用于触发交易回滚，但 `require` 处理用户错误，`assert` 调试内部漏洞

特性	<code>_address.transfer(value)</code>	<code>_address.send(value)</code>	<code>_address.call{value: ...}("")</code>
转发的 Gas	固定 2300 Gas	固定 2300 Gas	可自定义（默认转发所有剩余 Gas）
失败处理	回滚交易（revert）	返回 <code>false</code> ，不回滚	返回 <code>(false, ...)</code> ，不回滚
返回值	无（失败即回滚）	<code>bool</code> （成功 / 失败）	<code>(bool success, bytes data)</code> （含错误详情）
推荐度	✗ 不推荐	✗ 不推荐	✓ 推荐（需正确检查返回值）

函数的继承包含两种：

- **继承 Inheritance**：子合约继承父合约的状态随后复用父合约的逻辑
- **库 Libraries**：无状态，直接使用通用逻辑

通过使用 **ABI (Application Binary Interface)**，开发者能像调用本地函数一样交互其他合约。这一机制是 **DeFi 可组合性** 的技术基础

智能合约内有以下三大核心存储区域

维度	Calldata	Memory	Storage
生命周期	单次调用内有效（只读）	函数执行内有效（可写，销毁）	永久存在（跨交易）
Gas 成本	按字节（非零 16, 零 4）	二次方增长（临时便宜）	写入极贵，读取较贵
适用场景	传递函数参数（external 调用）	临时处理复杂数据	持久化状态（如余额、权限）
EVM 指令	CALLDATALOAD/CALLDATASIZE	MSTORE/MLOAD	SSTORE/SLOAD

此外，还有事件日志Event logs，用极低 Gas 记录关键事件，永久存于区块链

Bugs

常见安全考虑：

- Are we checking math calculations for overflows and underflows? 利用编译器特性（如 0.8+ 的溢出检查）减少人为错误
- What requirements and assertions should be made about function inputs, return values, and contract state? 通过 `require / assert` 构建防御边界；
- Who is allowed to call each function (access control)? 严格控制函数访问权限，避免“全民可调用”的关键操作
- Are we making any assumptions about the functionality of external contracts that are being called? 对外部合约保持“不信任”，通过校验、限流、锁机制降低风险

重入攻击：

以下代码的关键错误在于**交互在状态更新之前**

`call` 函数会调用外部函数，从而中断了函数执行流，恶意合约可能可以无限转账

```
contract Bank {
    mapping(address => uint) userBalances; // 用户余额映射

    function withdrawBalance() public {
        uint amountToWithdraw = userBalances[msg.sender]; // 1. 读取余额

        // 2. 交互：向调用者转账（外部调用，若调用者是合约，触发其fallback函数）
        if (!msg.sender.call{value: amountToWithdraw}("")) {
            revert("Transfer failed");
        }

        // 3. 状态更新：转账后，才将用户余额设为0
        userBalances[msg.sender] = 0;
    }
}
```

```

    }
}

```

针对以上代码，攻击者可构建如下代码：

每次 `fallback` 调用都会触发新的 `withdrawBalance`

```

contract Attacker {
    uint numIterations; // 重入次数限制（避免无限递归耗尽Gas）
    Bank bank; // 指向易受攻击的Bank合约实例

    constructor(address _bankAddress) {
        bank = Bank(_bankAddress); // 绑定目标Bank合约
        numIterations = 10; // 设定最大重入次数（如10次）

        // 1. 先存款（假设Bank有`addToBalance`函数，存入75 ETH）
        bank.addToBalance{value: 75}();
        // 2. 发起首次提款，触发攻击链
        bank.withdrawBalance();
    }

    // --- 重入机制：fallback函数 ---
    fallback() external payable {
        // 若还有剩余次数，继续重入
        if (numIterations > 0) {
            numIterations--;
            // 关键：在Bank的`withdrawBalance`未完成时，再次调用提款！
            bank.withdrawBalance();
        }
    }
}

```

防御思路是让状态更新放置在交互之前

ERC20

ERC20 是以太坊改进提案（EIP-20）定义的可替代代币接口规范 **Fungible Token Standard**，要求代币合约实现一组统一函数（如 `transfer`、`balanceOf`、`approve` 等），确保不同项目的代币能被钱包、交易所、DeFi 协议等通用兼容

- `totalSupply()`：总供应量查询。返回代币的全局总发行量（所有地址的代币余额之和）
- `balanceOf(address tokenOwner)`：地址余额查询。返回指定地址的代币余额，是最基础的“账户余额查询”
- `allowance(address tokenOwner, address spender)`：授权额度查询。返回 `tokenOwner` 授权给 `spender` 的剩余代币额度（即 `spender` 还能从 `tokenOwner` 提取多少代币）

这三个函数构成 ERC20 代币的“状态查询层”

`allowance` 的设计，让“用户资产控制权”与“第三方合约自动化”实现平衡

- `transfer`: 直接转账（点对点）。调用者 (`msg.sender`) 直接向目标地址转账指定数量的代币，是最基础的“一手交钱一手交货”
- `approve`: 授权额度（信任委托）。调用者 (`msg.sender`) 授权另一个地址 (`spender`)，允许其从自己账户提取最多 `value` 数量的代币
- `transferFrom`: 授权转账（第三方代操作）。被授权的地址 (`spender`) 从 所有者 (`from`) 账户向目标地址 (`to`) 转账，前提是 `from` 已通过 `approve` 授权足够额度

由此，ERC20实现了“流转双模式”：`transfer` 解决“直接交互”，`approve + transferFrom` 解决“授权交互”

每个 ERC20 合约都是 独立的“代币账本 token ledger”，例如USDC和WETH的数量是完全独立的，但通过ERC20又可以让它们被统一管理

Lecture 4 - pos

PoW和PoS都解决了：

- Sybil resistance (抗女巫攻击)
- Dynamic availability (动态可用性)

而PoS额外地提供了：

- Finality and accountable safety (最终性与可问责安全)：PoW 是“概率最终性”(交易需等多块确认，仍有回滚可能)，而PoS中交易一旦确认就不可逆转，且能追溯恶意验证者
- Slashing (惩罚机制)：PoS 中验证者质押了资产，通过Slashing可直接罚没资产，既有奖励又有惩罚

PoS——Accountably Safe

PoS实现了“质押 Stake→权益 Privilege→惩罚 Slashing→问责 accountable”的闭环逻辑

问责安全 accountable safe：

在Byzantine Fault Tolerant (BFT)中，需要设置做出最终决策所需的最小投票 / 签名数 Quorum

多数 BFT 系统中该阈值 $> 2/3$ 的总节点数 (如 30 个节点，Quorum 需 ≥ 21)，从而确保Safety (安全性) 和Liveness (活性)

当该阈值 $> 2/3$ 的总节点数则任意两个 Quorum 必须有重叠的成员。从而迫使决策组相互“知情”，阻止冲突决策同时通过。若 Quorum 阈值为 $> 2/3$ 节点，则 任意两个 Quorum 的交集必然 $> 1/3$ 节点，而恶意节点 f 必然 $< 1/3n$ ，因此一定有重叠的诚实节点发现“两个 Quorum 在确认冲突块”，从而拒绝认可冲突决策，确保 Safety

$2/3n$ 如何而来？

- Safe condition: $2q - n > f$ ，两个法定人数的重叠人数必须 超过恶意节点数，保证重叠中至少有 1 个诚实节点

- Liveness condition: $q < n-f$, 诚实节点 ($n-f$ 个) 必须能 **单独组成法定人数**, 保证系统在恶意节点离线时仍能推进

通过抓住没有诚实投票的 $1/3n$ 个恶意节点实现问责

相比起BFT委员会, PoS实现了**动态可用性**, 并通过问责制大大提高了作恶成本

而相比起PoW这样的普通 $n/3$ 容错, 即使作恶者超过了 $1/3$, 问责制也可以在活性被破坏的同时惩罚恶意节点

PoS——Finality

现实中大部分可问责安全协议 (如 PBFT、Tendermint), 都是基于“**部分同步**”模型设计的, 它包含两个阶段:

1. 异步期 (Asynchronous Period):

网络暂时混乱, 消息延迟 / 丢失不可预测 (比如某地区网络故障, 持续几小时)。

2. 稳定期 (After GST):

经过 **全局稳定时间 (Global Stabilization Time, GST)** 后, 网络恢复, 消息能**可靠、及时传递** (比如故障修复, 网络重新连通)

可问责安全协议通过“**允许暂时摆烂 (异步期保安全), 但最终必须支棱起来 (稳定期恢复活性)**”的部分同步模型, 扛住了混乱

相比起比特币的概率性篡改, PoS会直接宣布交易“**最终化 (Finalized)**”, 一旦最终化, 交易不可逆

在异步期PoS协议拒绝同时确认两个冲突的交易 / 区块

Blockchain CAP Theorem

在区块链中, 分布式系统在 **网络分区 (Partition Tolerance)** 下, 只能二选**一致性 (Consistency)** 或 **可用性 (Availability)**

当P (分区) 必然存在 (网络状况导致) 时:

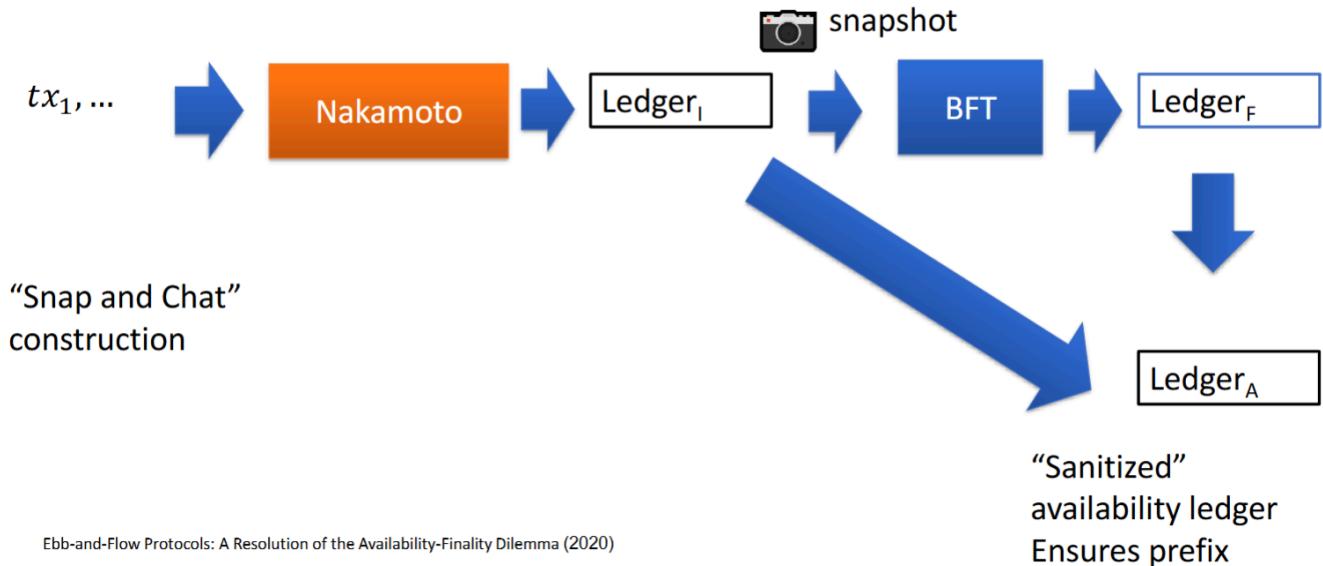
- 若保**动态可用性 (A)** → 分区节点各自决策, 出现冲突 (丢最终性 / C);
- 若保**一致性 (C)** → 分区时必须停摆 (不处理交易, 丢动态可用性 / A)。

Nested Chains

单链无法同时满足CAP, 那么可以拆分链, 将单链的功能**拆分为“最终化链 (Finalized Chain)”和“可用链 (Available Chain)”, 通过“嵌套”(最终化链是可用链的前缀)协调矛盾:**

- **最终化链 (Finalized Chain):** 可用链的**前缀 (Prefix)**, 即使网络混乱 (异步期) 也绝对不改变。只有当网络稳定时才会继续扩展
- **可用链 (Available Chain):** 处理实时交易的主链, 网络稳定时, 或节点动态变化时, 仍能**持续处理交易**。网络混乱时, 可能产生临时分叉, 但这些分叉会被最终化链“覆盖”

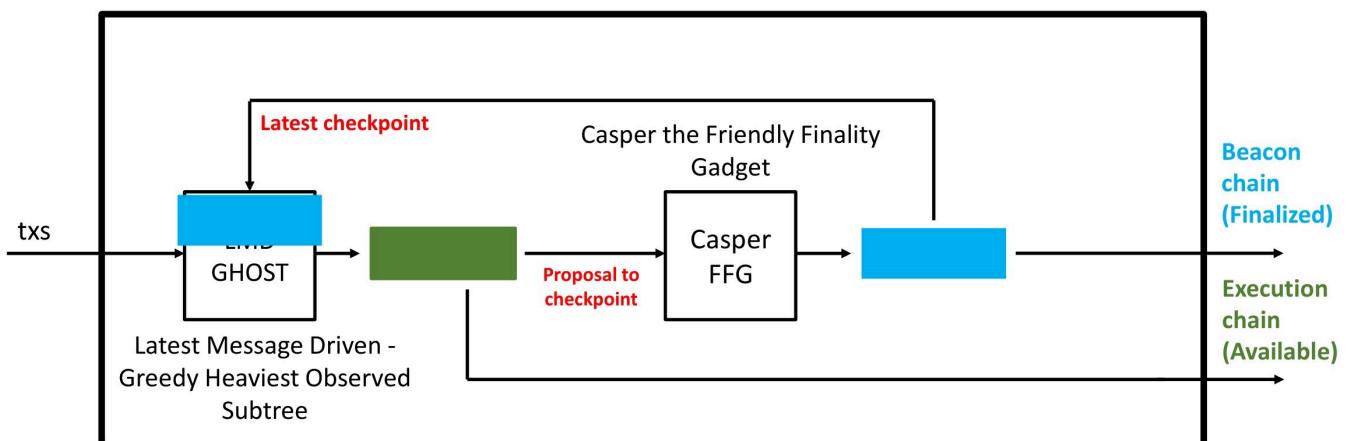
如下图 “Ebb and Flow” Cycle，Nakamoto 共识生成中间账本保障动态可用性，固定一个时间点使用快照 (Snapshot) + BFT 共识生成最终化账本保障最终性，之后新的账本会基于这个静态最终化账本继续扩展



Ethereum 2.0 Chains

PoS以太坊采用类似机制，通过“12秒 slot 保证动态可用（持续出块），2个 epoch 的 BFT 投票实现最终化（不可逆）”

以太坊 PoS 通过 **LMD GHOST** 维护“动态可用的实时链（Execution Chain）”，**Casper FFG** 打造“异步安全的最终化链（Beacon Chain）”，并通过质押 + 追责实现可问责安全



GHOST Protocol

以太坊具有超快的出块速度，因此有很多分叉

分叉中，最终未被选为主链的区块，成为 **孤块** (**Orphan Blocks**)，造成算力浪费

考虑使用**GHOST 协议** (**Greedy Heaviest Observed Sub-Tree**, 贪婪最重观察子树)，选择 **权重最大的子树 the heaviest Sub-Tree** 作为主链

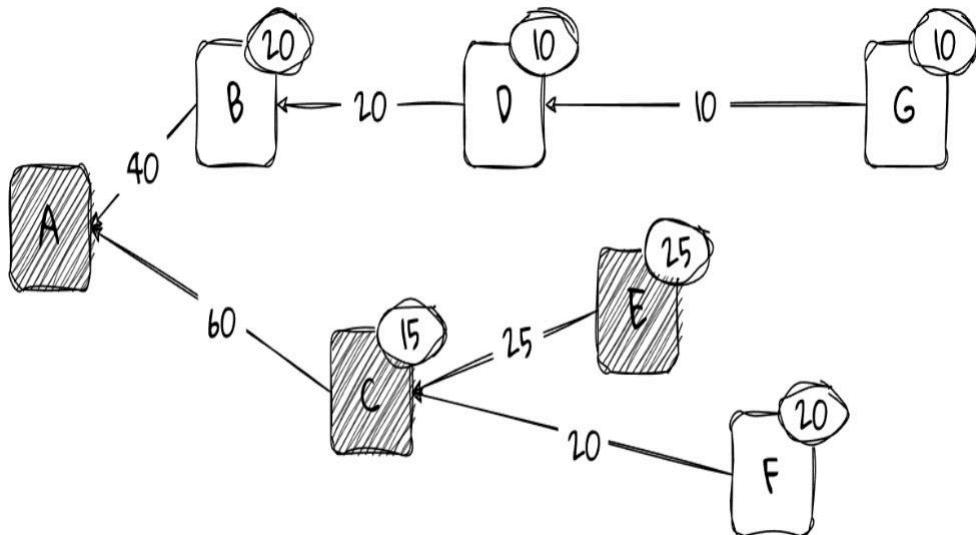
区块不再只引用 **直接父块**（上一个区块），还能引用 **叔块**，叔块的存在被纳入 **权重计算**，避免其“孤立无援”，让分叉的所有分支都能贡献权重

叔块能获得 **部分区块奖励**（**Block Reward**），但 **无法获得交易费**。因此激励了“诚实但运气差”的矿工，也避免了**自私挖矿 Selfish Mining**

PoS中使用LMD（Latest Message Driven） GHOST，基于 **活跃验证者的最新投票 / 证明 (attestation)** 给分支加权，确保权重反映验证者**实时态度**

对于没有被选为主链的分叉，区块会经历“**废弃 Discard**→**重组**→**交易回流**→**状态清零**”的过程

- **重组 (Re-organization)** —— 节点丢弃分叉链（B、D、G）带来的 **临时状态**（如账户余额、合约状态的临时变更）
- **交易回流 (Transactions Return)** —— 分叉链（B、D、G）中包含的 **有效交易**（未被规范链处理的交易），会被 **送回内存池 (Mempool)**，等待后续规范块收录
- **状态忽略 (State Unchanged by Them)** —— 分叉链（B、D、G）试图引发的 **状态变化**（如转账、合约调用）会被 **完全忽略**，最终区块链的状态 **仅反映规范链 (A→C→E)** 的结果



Gasper Protocol

Gasper = LMD-GHOST + Casper

Casper FFG：通过 **双纪元 (Epoch) 投票**，将 LMD-GHOST 选定的主链前缀 **永久锁定为“最终化链”**，确保异步网络（延迟、分区）中 **区块不可逆**

Gasper 通过“**LMD-GHOST 负责‘活’(动态出块、分叉收敛)，Casper 负责‘稳’(最终化、安全不可逆)**”的双核协作，让以太坊 PoS 既拥有高频交易的灵活性，又具备异步网络下的绝对安全性

Attestations

Attestations: 验证者 Staker 的“投票凭证”，对区块投票：

- 为 **LMD-GHOST (分叉选择规则)** 提供“最新投票”，决定 **当前最重主链**（支撑 12 秒高频出块的动态可用性）。有效质押（投票权重）会 **随时间变化**
- 为 **Casper FFG (最终化协议)** 提供“共识投票”，推动区块进入 **最终化状态**（异步下不可逆）。

以太坊中每 12 秒 (1 Slot) 创建一个区块，如果都需要所有人都投票就效率过低，因此每个 Slot (12 秒)，从 **当前活跃验证者池中 伪随机选出一个专属委员会**（如 ≥ 128 名验证者）再从中随机选出提议者 Proposer。

通过“**Slot 级随机选委员会 + 提议者，小范围高效投票**”的机制，在 12 秒内实现链的快速推进

每 32 Slot 作为 1 个 Epoch，聚合 Slot 的投票实现最终性（让链“不可逆”）

Randomness Beacon

如何不可预测地发布随机值从而随机选择委员会和提议者？

目前以太坊通过 **RANDAO 协议** 实现近似“随机信标”的功能

但最后参与者具有较强影响力 (**Last Actor Influence**)，可能通过查看之前提交者给出的结果从而间接操控随机结果

Verifiable Delay Function (VDF)

可验证延迟函数 (Verifiable Delay Function, VDF) 试图解决随机信标生成中“最后参与者操纵”的漏洞

其具有三个特性

- **函数特性 (Function)** “Soundness”: 对每个输入 x ，VDF 的输出 $y = F(x)$ **唯一确定**
- **延迟特性 (Delay)** “ σ -Sequentiality”: 计算 $F(x)$ 需要 **时间 T** ，且即使在并行计算机上，也无法在 $(1-\epsilon)T$ 时间内完成。因此最后参与者 **提交输入后，必须等待 T 时间才能得到结果**，无法提前预测
- **可验证性 (Verifiable)**: 无需重新执行耗时的 $F(x)$ 计算，只需通过 **短证明 π** ，就能 **高效验证 $y = F(x)$ 是否正确**（验证时间远小于 T ）

如何解决？

1. **提交输入时**: 最后参与者必须在 **不知道最终结果** 的情况下提交 x （因为计算 $F(x)$ 需要 T 时间，结果还没生成）。
2. **计算过程中**: 输入 x 固定后， $F(x)$ 开始计算，需耗时 T ，期间无法篡改输入或结果。
3. **结果验证时**: 其他节点通过短证明 π 快速验证结果，无需重复耗时计算，保证共识效率。

PoS Ethereum Incentive system

以太坊 PoS 的激励系统，通过：

- “奖励（交易费 + 质押收益）吸引诚实参与”
- “惩罚（Slashing）威慑恶意行为”

对于以太坊信标链（Beacon Chain）的验证者：

需直接质押或流动性质押（如 Lido，小用户将ETH存入该合约，间接参与验证并获得奖励）才能加入成为验证者

需要签名区块**Attestations**或随机成为提议者，同样具有奖励

对于交易的打包，为防止Gas费用被无限抬高从而导致效率低下，催生了EIP-1559

EIP-1559：设置了以下三大核心参数

- **gasLimit**：交易的 Gas 上限。一笔交易 **允许消耗的最大 Gas 量**，防止交易无限循环浪费资源
- **MaxFee**：用户承受的最高 Gas 价格（包括基础费Base Fee和小费 Priority Fee）。**控制交易成本风险**，防止基础费因拥堵飙升导致的超高费用
- **maxPriorityFee**：给提议者的“小费”。激励优先打包自己的交易

$$\text{gasPrice} \leftarrow \min(\text{maxFee}, \text{baseFee} + \text{maxPriorityFee})$$

通过以上参数，配合动态 Gas 价格计算，既让用户成本可控，又保障提议者激励，还能自动适配网络拥堵

此外，通过对基础费的销毁，抵消了ETH不断增发带来的通货膨胀 inflation。另外，销毁也避免了**链下协议退款**（Off-Chain Refund Deals）等作恶可能

Lecture5 - DeFi

DeFi vs. CeFi

DeFi 是 **开放 open**、**无需许可 permissionless**、**高度互操作 highly interoperable** 的金融基础设施，构建在 **公共智能合约平台 public smart contract platforms**（如以太坊）之上

- **开放**：协议对所有人可见、可访问；
- **无需许可**：无需机构审批，任何人可参与；
- **互操作**：不同 DeFi 协议可自由组合（如借贷协议与 DEX 结合）；
- **公共智能合约**：代码公开、运行在去中心化网络，而非中心化服务器。

DeFi vs. CeFi

CeFi	DeFi
需许可 Permissioned : 系统闭源，搭建在 中心化数据库 (如银行服务器) 上。第三方若要使用或开发功能，必须获得机构批准	无需许可 Permissionless : 系统开源，搭建在 无需许可的区块链 (如以太坊) 上。任何人可自由使用、互操作
托管型 Custodial : 资产由 持牌第三方 licensed third-parties 保管 (如银行存管资金、中心化交易所托管加密币)。用户依赖机构保障资产安全	非托管型 Non-custodial : 资产 不由单一第三方 托管，用户通过私钥自主控制 (如 Metamask 钱包)。资产安全由用户自己负责
中心化信任 + 治理 Centralized trust & governance : 单一实体 (如银行、公司) 掌握 升级、管理权限 (可冻结账户、修改规则)。用户需信任机构的决策和操守	去中心化信任 + 无信任 (Trustless): 无单一实体能决定升级，通常通过 社区投票 Community vote + 智能合约 Smart contract 治理
真实身份 : 用户需使用真实身份注册	伪匿名 Pseudonymous + 隐私 privacy : 用户无需提供实名信息

Market Maker

做市商是 **连接买方和卖方的中间枢纽**，主动提供 **双向报价** (同时报出“买价”和“卖价”)，让交易者无需等待对手方即可快速成交



做市商是市场的“润滑剂”：

- 对交易者：提供 **即时成交机会** (无需等待对手方)；
- 对市场：通过大量资金保障 **流动性 liquidity** 和 **活跃度**，降低交易摩擦；
- 对自身：通过 **价差和手续费** 盈利，但需承担价格波动风险。

DEX

交易所：主要负责货币之间的转换

相比中心化交易所CeX的安全性风险 (平台跑路)，**DEX (Decentralized Exchange)** 提供更高安全性

DEX 通过技术取代“中心化中介的信任依赖”，用区块链和智能合约构建信任，而非依赖 CeX 这类中心化机构
四个特点：

- **Programmable**
- **Transparent**
- **Permissionless**
- **Non-Custodial**

如何构建DEX? Automated Market Maker (AMM)

Automated Market Maker (AMM)

Automated Market Maker (AMM)的核心思想是用算法替代人工做市商，构建链上资金池 **on-chain pool**

1. 流动性提供 **Liquidity Provider, LP**: 用户向链上资金池存入两种资产（如 ETH 和 USDC），形成交易对；
2. 交易执行：用户直接与资金池交易，汇率由算法自动计算（如 Uniswap 的 $x \cdot y = k$ 常数乘积公式，交易后保持资产乘积不变）；
3. 历史与代表：2017 年 Vitalik 提出思路，代表项目有 Uniswap（双资产池）、Balancer（多资产池）、Bancor 等。

核心优势：

- **Gas 高效**：交易逻辑是简单的算法计算（而非复杂的订单匹配），大幅降低 Gas 成本；
- **支持合约访问**：资金池是链上智能合约，其他 dApp 可直接调用（如闪电贷协议可瞬时借用池内资产，实现“可组合性”）；
- **易启动**：无需大量做市商，只要有初始流动性注入，就能开始交易（bootstrap 成本极低）

LP存入资金后，每当有交易发生，他们都能赚取交易手续费 Transaction fee

$$x \cdot y = k$$

该公式要求资金池中两种资产（如 ETH 和 USDC）的数量 **乘积恒定** (k 为常数)

交易时，一种资产数量变化，另一种资产数量必须反向变化，以维持乘积不变

例如：

1. 资金池内有 $x = 10$ ETH, $y = 12$ USDC, 因此 $k = 10 \times 12 = 120$ 。
2. Alice 想从池中 **买入 4 ETH**（即池内 ETH 数量减少 4）
3. Alice 需存入 8USDC 来确保 k 不变， $(10 - 4) \times (12 + 8) = 12$

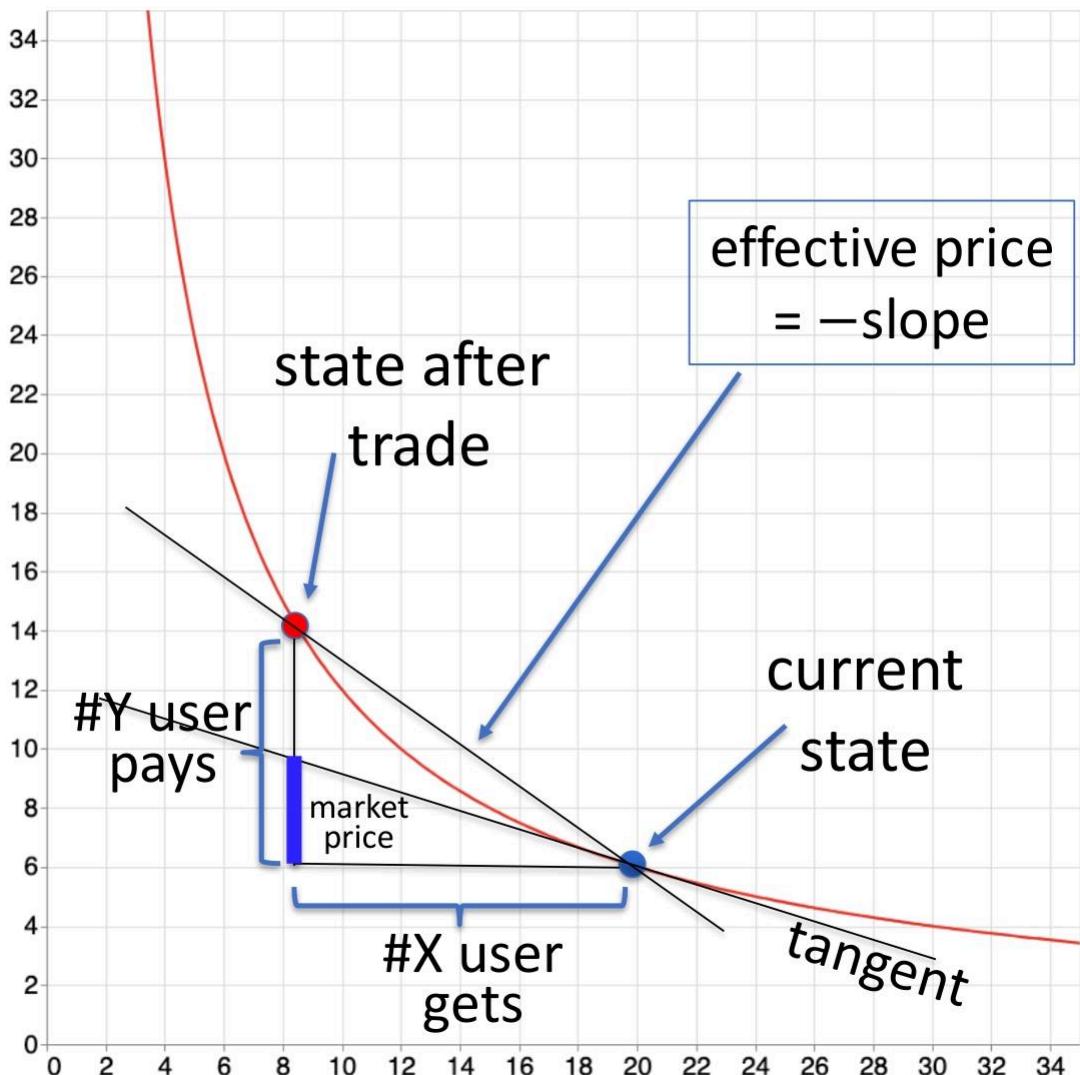
通过该公式自动维持了汇率

套利者 arbitrageurs 会根据外部市场价自发向外部市场**低买高卖，赚取差价**，从而自动维持市场定价

滑点 Slippage 问题

由于切线贴近双曲线的过程，交易规模越大，用户实际拿到的汇率越差（滑点越严重）

Uniswap 等 AMM 平台会 **设置滑点容忍度 Slippage Tolerance (如 0.5%)**：若交易导致的滑点超过阈值，交易将被拒绝，避免用户承受过大损失



三明治攻击 Sandwich Attack

通过抢先交易获取MEV的方法：

1. 攻击者打听到受害者即将购买货币x
2. 攻击者抢先进行交易，购买x
3. 受害者购买x，由于攻击者抢先进行了交易，汇率变差，造成亏损
4. 攻击者再将x卖出，由于受害人购买了x，这时x的汇率上升，从而导致攻击者获取差价利润

LP

LP通过向资金池注入资金获取手续费收益

LP会尽可能按比例 $y'/x' = y/x$ 将资金注入或取回，当LP的资金份额很小时，汇率相对保持不变

如果选择成为LP，那么就会获得收益，但相应也产生了价格改变 Price change 带来的无常损失 (Impermanent Loss, IL) 风险，套利者 Arbitrageurs 会通过“跨市场交易”收割价差 harvest spreads

而如果选择持有Hold，无收益但也无风险

DeFi Lending Protocols

银行在现实中发挥储蓄deposit和借贷lending的作用

而CeFi Lending也发挥类似银行的作用



此外，针对借贷人可能的违约行为，CeFi会要求 **超额抵押（Over-Collateralized）** —— 存入价值高于借款的资产，作为违约担保。如果借贷人违约就没收所有**抵押物 Collateral**

由此将风险从平台转移到借贷人身上

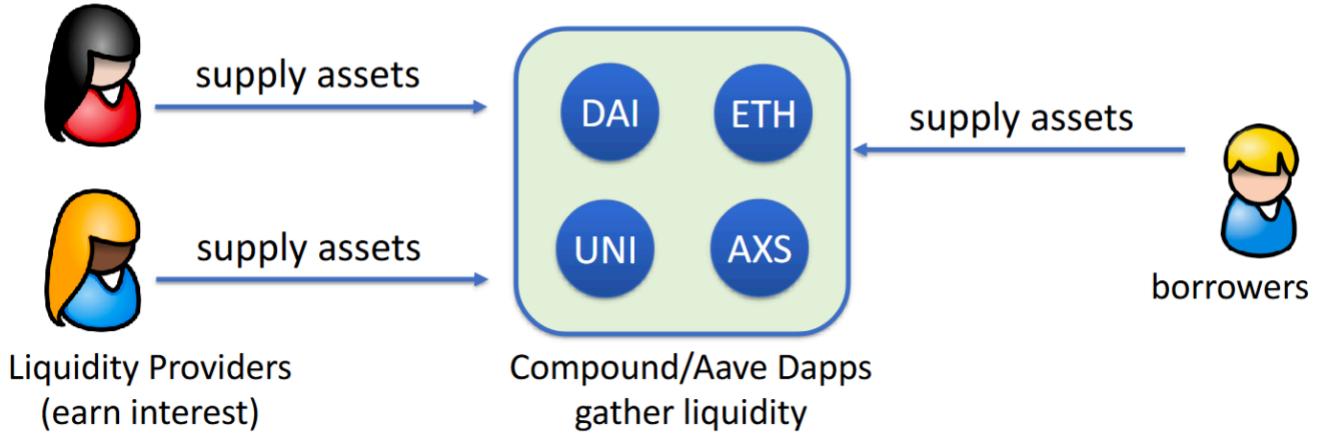
此外，如果货币价值上涨到即将超过抵押的程度，就可能触发清算（**Liquidation**），平台主动提前处理抵押品

CeFi的缺陷：

- 用户必须信任平台
- 借款人支付的利息会流入 CeFi 机构腰包，随后才分配给存款用户。这个过程中用户处于弱势方
- CeFi完全掌握利差

一个更好的DeFi借贷方式：Liquidity pools

- **流动性提供者（Liquidity Providers）** 向池子内存入资产
- **借款人（Borrowers）** 也存入资产作为抵押品，随后借款
- **池子 Pool**作为借贷的资金载体 carrier of capital



这种方式按照LP的存款比例直接提供收益，防止被CeFi机构抽成take a cut

Flash Loan

闪电贷是在同一笔区块链交易中完成“借款”和“还款”的借贷模式。

区块链交易的“原子性 Atomicity”(全成或全败)是核心支撑，回滚rollback机制

但闪电贷既可以用于合法套利 **legally arbitrage** (修复市场价差，提升效率)，也可能被用于恶意攻击 **malicious attack**

Cross-chain

DeFi具有以下特性：

- **互操作性 (Interoperability)** : 用户在一条区块链上持有的资金 / 资产 (如 ETH、NFT)，能被转移到另一条区块链
- **可组合性 (Composability)**: 一条链上的 DAPP，能调用另一条链上的 DAPP功能的能力

在现实中，通常通过跨链桥 (**Bridges**) 来实现跨链操作

一张跨链桥思路是包装币 (**Wrapped Coins**)

将链 A 的资产 X (如比特币链的 BTC)，在链 B (如以太坊) 上生成对应的“包装版资产 wrapped-X”(如 wBTC)

机制 (以 wBTC 为例)：

1. 用户将 BTC 存入托管方 (如 BitGo)；
2. 托管方在以太坊上发行等量 wBTC (ERC20 代币)；
3. 当用户赎回 BTC 时，销毁 wBTC，托管方返还 BTC。

Short Summary on DeFi

无需许可 (Permissionless) :任何人都能通过编写 Solidity 代码，开发并部署金融工具（如借贷、交易协议），**无需中心化机构审批**

透明 (Transparent): DApp 的 **代码 (智能合约) 和运行状态 (如资金池、交易记录) 完全公开**，任何人可通过区块链浏览器（如 Etherscan）查验

可组合 (Composable) :不同 DApp 可通过智能合约 **互相调用功能**，像积木一样组合出复杂金融产品

Maximal Extractable Value (MEV)

以太坊催生了**搜索者Searcher**

Searchers 通过实时监控区块链状态（如 DEX 价格、借贷协议抵押品健康度），发现**套利 Arbitrage、清算 Liquidation、MEV (最大可提取价值)** 等机会，再通过编写智能合约或交易脚本，以**最快速度提交交易**（甚至打包成交易束“bundle”）抢占利润。

当某位搜索者发现有利可图并提交交易tx到交易池中，其他验证者或搜索者可能提交同样的交易，并抢在那位搜索者之前被打包，那么他们就获取到了MEV

MEV 的本质是 “**交易透明性 + 打包权可操纵**” 导致的利益争夺 —— 公开的 mempool 让交易内容暴露，验证者和搜索者通过提高Gas费用 “插队交易” 提取价值

而这样提高Gas费用的情况可能伤害到普通用户的权益

缓解方法包括两种：

- **次级交易排序市场 Secondary Transaction Ordering Market**: 将“交易排序权”从单一验证者（Validator）手中，转移到**专业区块构建者（Builder）**，由其汇总所有交易束（加密非公开），优化排序 optimize order（平衡 MEV 收益和普通用户交易），生成“最优区块”
- **交易公平排序 Fair Ordering of Transactions**: 认为 **MEV 的根源是“区块提议者能自由选择交易顺序”**，因此试图通过技术手段**剥夺提议者的排序权力**，从源头减少 MEV。即让交易按**时间戳或密码学随机规则排序 (Time-Based Order-Fairness)** 或采用“**盲排序**” (**Blind Order-Fairness**)

Lecture 6 - zkp

Privacy

区块链中的**隐私 Privacy**

在区块链中隐私是有必要的：

- **供应链隐私 Supply chain privacy**保护商业机密
- **支付隐私 Payment privacy** 保护企业和个人隐私

- 业务逻辑隐私 Business logic privacy 防止智能合约代码被攻击者研究

隐私类型分两种：

1. **伪匿名性 (Pseudonymity)**：被定义为“弱隐私”(weak privacy)，其核心特征是每个用户拥有长期一致的化名（例如 Reddit 平台的用户名）。
 - 优点：用户可通过长期使用的化名积累声誉，便于在网络中建立信任关系。
 - 缺点：随着时间推移，这些化名可能通过交易模式、关联信息等被逐步关联到用户的真实身份，导致隐私泄露。
2. **完全匿名性 (Full anonymity)**：核心特征是用户的交易不可链接，即没有人能够判断两个交易是否来自同一个地址。这种隐私类型下，交易之间的关联性被彻底切断，极大降低了身份被追踪的风险。

完全隐私并不完美，因为会使得犯罪行为难以追踪 trace

如何在保护隐私的前提下，确保交易符合法律规范？

零知识证明 (zero knowledge proofs)

Ethereum本身完全公开不提供隐私

而比特币的“匿名”是**伪匿名**——地址看似随机，但交易图谱会暴露行为关联，再结合交易所 KYC、商家记录和货币输入等辅助信息，就能把地址和真实身份绑定

此外，**可互换性 (Fungibility)** 对货币很重要，其要求“同一面额的货币具有同等价值，与其历史无关”("a dollar is a dollar, regardless of its history")

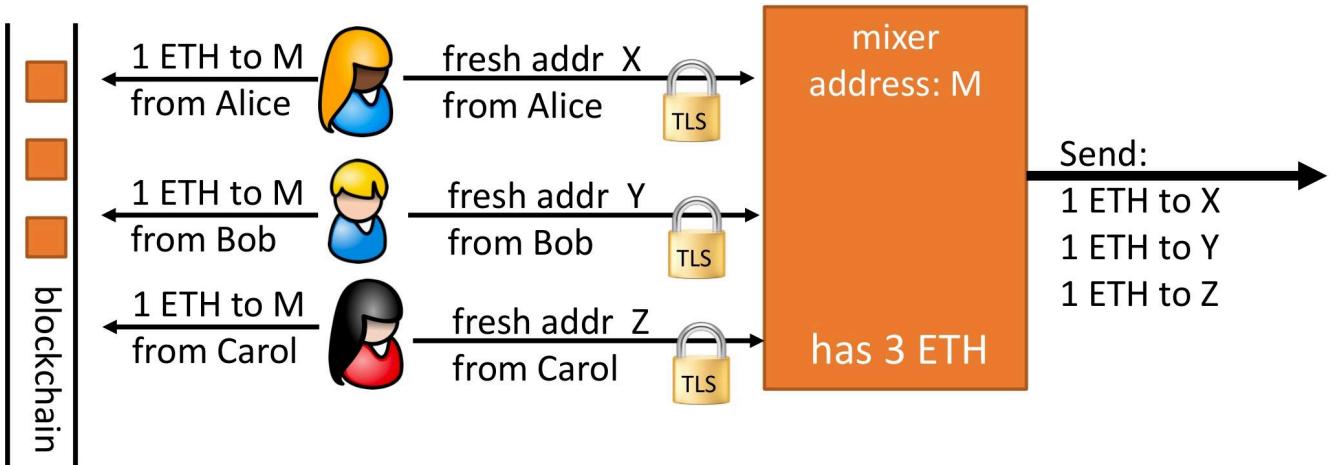
而比特币因交易历史公开，每枚硬币的“血统”可追溯，导致其缺乏严格的可互换性

Private coins on a Public Blockchain

一种思路：Simple mixing

通过混合器生成新地址，从而切断与原地址的联系

然而这样引入了新的第三方，其是否可信决定了方案的安全程度



是否可以考虑无混合器的安全混合？

BTC——CoinJoin

参与者有两类地址：

- 找零地址（Change Address）：回收用户 UTXO 中 未参与混合的剩余资金
- 混合地址（Mix Address）：接收 混合后资金，通过“等额分配 + 多用户协同”模糊资金归属

流程如下：

1. 用户准备：

- Alice 有 5 BTC UTXO (计划贡献 2 BTC 混合, 3 BTC 找零)；
- Bob 有 3 BTC UTXO (计划贡献 2 BTC 混合, 1 BTC 找零)；
- Carol 有 2 BTC UTXO (计划贡献 2 BTC 混合, 0 BTC 找零)。

2. 协同构建交易：

- 输入：Alice (5 BTC)、Bob (3 BTC)、Carol (2 BTC) 的 UTXO；
- 输出：
 - 混合地址：A_mix、B_mix、C_mix (各收 2 BTC, 共 6 BTC)；
 - 找零地址：Alice (3 BTC)、Bob (1 BTC)、Carol (0 BTC)。

3. 隐私效果：

- 混合地址的 2 BTC 无法关联到具体输入者 (匿名集大小为 3)；
- 找零地址的资金归各自用户，不参与混合，避免泄露关联。

inputs (not private):	A1: 5, B1: 3, C1: 2
outputs (private):	B2: 2, A2: 2, C2: 2
outputs (not private):	A3: 3, B3: 1

mix addresses

这个过程不需要中心化混合器，由用户 P2P 协同构建交易，资金直接在链上转移

通过 找零地址（处理个人剩余资金） 和 混合地址（等额分配模糊归属） 的配合，打破 UTXO 的输入 - 输出关联

但 交易规模和多方协作的设计 trade-off 使其在实际中面临 成本高（手续费由交易字节数决定）、效率低（必须等待多方签名）、抗干扰性差（若有人突然退出则无法进行） 的问题

Cryptographic Commitments

承诺 Commitments 是链上存储 “简短、隐藏且具有绑定性的承诺”（如哈希值）

- **隐藏性：**交易细节（金额、地址、合约逻辑）被加密，不公开；
- **绑定性：**承诺与真实交易数据绑定，无法篡改（若数据改变，承诺会变化，可被检测）。

承诺 ≈ 密封信：信的内容（交易细节）被隐藏，但封印（哈希）公开，保证内容存在且未篡改

承诺的工作流程：

- **密封阶段（Commit）：**

发送方（左侧人物）将数据（`data`）通过密码学算法生成 **承诺值**（如哈希值，对应“信封的封印”），并公开承诺值（但隐藏原始数据）。

- **揭露阶段（Open）：**

后续需要验证时，发送方公开原始数据，他人可通过算法验证“原始数据的承诺值是否与之前公开的一致”（对应“打开信封，检查内容是否被篡改”）。



Zero Knowledge Proof (ZKP)

零知识证明的本质——“证明某个陈述为真，却不泄露陈述背后的秘密”

核心特性：

- **完整性（Completeness）：**如果陈述为真，且证明者（Prover）和验证者（Verifier）都遵循协议，诚实的验证者将确信陈述的真实性
- **可靠性（Soundness）：**如果陈述为假，任何试图欺骗验证者的证明者（即使不诚实）成功欺骗的概率非常低（接近于零）
- **零知识性（Zero-Knowledge）：**如果陈述为真，验证者除了知道陈述的真实性外，不会获得任何关于秘密本身的额外信息

ZKP的存在使得在公开链上的隐私保护和公开验证成为现实

zk-SNARK：验证一个陈述为真的简短证明，而且保护陈述本身隐私

具备简洁性（证明短）、非交互性（无需多轮通信）、可公开验证的零知识证明

数独的零知识证明

是零知识证明（ZKP）核心思想的经典具象化案例，用于解释“如何证明‘知道数独的解’，却不泄露解的具体内容”：

存在：

- **证明者（P）**：声称自己知道某数独的解。
- **验证者（V）**：希望确认 P 的声明为真，但 **不想看到解的具体数字**（零知识性）

步骤如下：

步骤1：承诺（Commit）——隐藏解的细节

P 通过“物理遮盖”或“密码学承诺”隐藏解：

- **物理类比**：用卡片覆盖数独的每个单元格，仅露出网格的行、列、宫格边界（类似密封信封）。
- **密码学映射**：对每个单元格的数字 x ，生成 **哈希承诺** ($H(x, r)$) (r 是随机数，防止相同数字的承诺重复)，并公开所有承诺（相当于公开“密封后的信封哈希”）。

步骤2：挑战（Challenge）——随机选择验证维度

V 随机选择验证对象（行、列或宫格），例如：

- “我要检查第 5 行是否数字是否合法”；
- “我要检查第 3 列”；
- “我要检查左上角的 3×3 宫格”。
- **关键**：随机性让 P 无法提前作弊（若 V 固定选行，P 只需保证行合法，列和宫格可造假；但随机选择时，P 必须保证所有规则都正确）。

步骤3：回应（Response）——揭示局部，验证合法性

P 根据 V 的挑战，揭示对应行 / 列 / 宫格的数字：

- **物理场景**：揭开对应行 / 列 / 宫格的卡片，展示数字。
- **密码学场景**：公开对应单元格的 x 和 r ，V 验证 ($H(x, r)$) 是否与之前的承诺一致（确保数字未篡改），同时检查数字是否满足 1-9 不重复。

步骤4：重复验证——提高可信度

V 多次随机发起挑战（如 10 次，分别选行、列、宫格）。若 P 每次都能通过验证：

- **完备性**：若解正确，P 必然能通过所有挑战（诚实证明者一定成功）。
- **可靠性**：若解错误，P 几乎不可能通过随机挑战（作弊者被发现的概率随挑战次数指数级上升）

可通过 **密码学优化** 转为 **非交互式 ZKP**（如 zk-SNARK 的思路）：

1. 引入 **公共随机数** 模拟 V 的挑战（无需实时沟通）；
2. 用算术电路或多项式承诺压缩证明，让验证者只需一次验证即可。

Zcash

首个大规模应用零知识证明（zk - SNARK）技术的加密货币

核心目标是在公链上实现“交易可公开验证，但参与方、金额等细节完全保密”

Zcash创造了一种与BTC锚定的ZTC，能让交易的**发送方、接收方、金额完全加密**，仅向授权方（通过查看密钥）开放细节

隐私实现：

Zcash 依赖 **zk - SNARK**（零知识简洁非交互知识论证）实现“隐私 + 可验证”的平衡，其逻辑类似“数独的零知识证明”：

1. **证明交易合法**：当用户发起隐私交易时，zk - SNARK 生成一个**短证明**，证明“这笔交易的输入、输出符合规则（如未双花、余额足够）”。
2. **不泄露细节**：验证者（如矿工、节点）只需验证该证明，**无需知道交易的具体金额、地址**——就像验证数独解存在，但看不到解本身。

Zcash允许自由选择**透明交易（t-address）** 或**屏蔽交易（z-address）**

Tornado cash

ZK 混合器，通过“存款→混合→取款”三步，结合 zk-SNARK 实现隐私：

1. 存款（Deposit）

用户向 Tornado Cash 的智能合约**存入固定金额的代币**（如 1 ETH、100 DAI），同时生成一个**秘密票据（Note）**（包含随机密钥），并将票据的**哈希值**上链（隐藏真实票据内容）。

2. 混合（Mixing）

多用户的存款进入**资金池**，智能合约将资金混合。此时，zk-SNARK 发挥关键作用：

- 证明者（用户）需证明“我拥有一个合法的票据（即存过钱）”，但**不透露票据对应哪个存款地址**（零知识特性）。
- 验证者（合约或节点）只需验证证明的有效性，无需知道具体存款和取款的关联。

3. 取款（Withdrawal）

用户通过**新生成的匿名地址**发起取款，提交 zk-SNARK 证明“拥有合法票据”，合约验证后将资金转入新地址。

Lecture 7 - scaling - part I

BTC和Ethereum的交易速度相比起传统支付过慢，需要扩容以提升区块链交易吞吐量（Tx per second）

一个好方法：

Off-chain Protocols

既避免了主链的性能瓶颈，又规避了侧链 / 交易所的信任风险

设计目标：

- **Public transparency**（公开透明）数据库状态 对所有人公开可验证
- **Self-custody**（自我托管）：用户 通过私钥完全掌控自己的资产和状态
- **Censorship-resistance**（抗审查）：用户有权强制让自己的交易被执行，即使中心化的处理节点（如 Sequencer）拒绝处理。

State (payment) channels

通过链下通道大大降低交易次数：

1. **存款（链上交易 1 次）**：Alice 先在主链上向 Bob 存入 1 ETH（双方通过智能合约锁定资金）。
2. **链下交易（数百次咖啡）**：之后 Alice 买咖啡时，只需和 Bob 链下记录交易（比如 Bob 本地记“Alice 已消费 X ETH”），无需每次上链。
3. **结算（链上交易 1 次）**：月底 Bob 将剩余资金（1 ETH - 已消费金额）通过主链退还给 Alice。

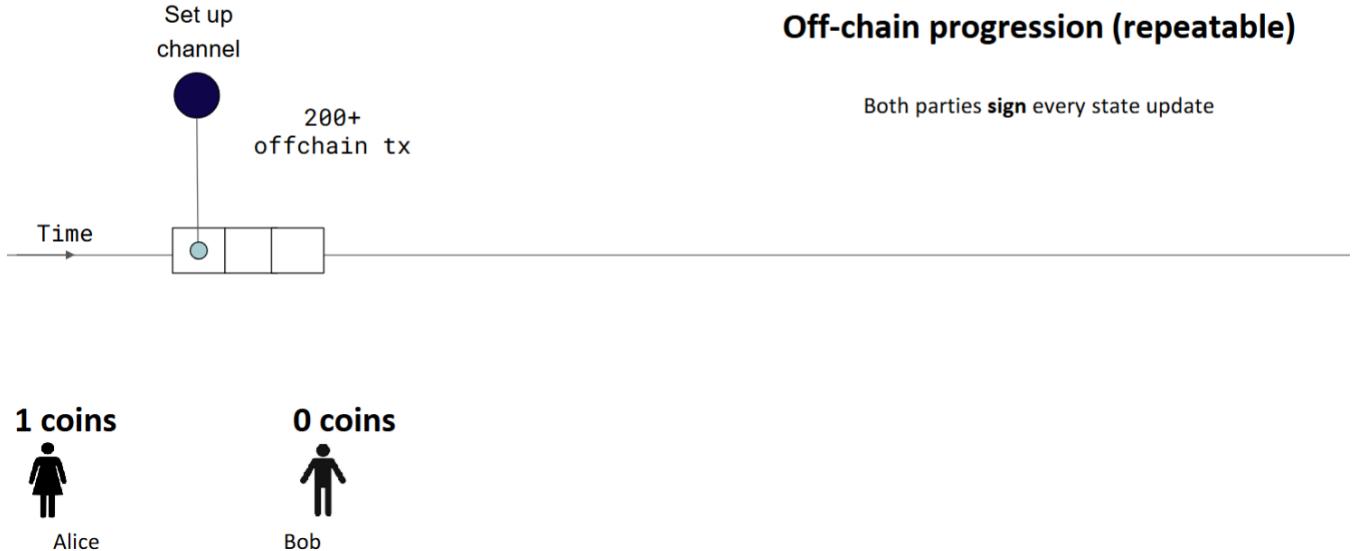
如下图，只需要进行一次链上交易例如存储deposit

随后双方在链下高频交易，进行链下状态更新（需要双方签名 both sign）

Off-chain Channels

Set up channel

One or both parties deposit coins



如何防止任意一方在链下离线off-line导致资金被困funds stuck?

任意一方都可以强制将有双方已签名的余额提交到链上进入纷争状态 dispute，随后双方提交证据evidence。纷争结束后会强制链上更新，之后既可以关闭通道也可以保持通道开启双方继续交易

那么假如有一方提交旧状态，如何确保旧状态失效，确保最新状态被主链认可？

1. 激励型 (Incentive-Based, 如 Spilman Channels): 通过 经济奖惩，让提交旧状态的行为“无利可图甚至亏损”，从而抑制欺诈
2. 撤销型 (Revocation, 如 Lightning Network): 通过 密码学机制，让 新状态的生成直接使旧状态失效，无需 依赖经济奖惩

Spilman One-way Channel

单向通道中，只有其中一方Alice可以付款，另一方Bob只能收款

此外，有一个退款时间refund time t，如果时间到了后Bob没有广播，那么钱将会退回给Alice

Bob 收到交易后，可以暂不签名或广播，而是选择“囤积”这些交易（因为每笔交易代表自己能获得的资金，越多越好）

但在时间t 截止前，Bob 必须签名并广播“对自己最有利的交易”，否则将会自动退款

这种设计使得Bob会主动选择 最新、金额最大的交易（因为越晚的交易，累计金额越高），天然实现“最新状态胜出”

无需复杂密码学机制

这一种通道速度快但仅支持单向支付，不够灵活

Lightning Channel

闪电通道是 **两个节点**（如 Alice 和 Bob）之间的双向连接

双方在链下高频交换资金，仅在通道建立和关闭时与比特币主链交互

如果有一方离线：

- 若 Bob 离线或拒绝合作，Alice 可提交最新签名的状态到主链，触发强制关闭。
- 主链会设置**挑战期**（如 1 周），若 Bob 未反驳，主链按 Alice 提交的状态结算；若 Bob 提交更新的状态（如更晚的交易），则以最新状态为准。

如果作弊会怎样？

- **撤销密钥交换**：每次生成新状态时，双方会交换“**撤销密钥**”(Revocation Key)。
- **旧状态失效**：若一方（如 Bob）试图广播**旧状态**（如 earlier 的 Alice: 0.5 BTC, Bob: 1.5 BTC），Alice 可使用对应的撤销密钥，提交正义交易 Justice Transaction，向主链证明该状态已被**更晚的状态取代**，主链将拒绝旧状态，并对作弊方施加惩罚（如没收其通道内的资金）。

在闪电通道中存在 **瞭望塔Watchtowers**这样的专业的在线服务节点，其随时监控状态的新旧程度，但要求节点**定期更新**“最新的撤销密钥（Revocation Key）”

State Channels

像以太坊这样的更智能平台可以支持支付通道之外的通用状态通道

支持任意 2 方智能合约（如拍卖、游戏），仅合约创建和终止上链 on-chain，中间步骤链下 off-chain 完成。

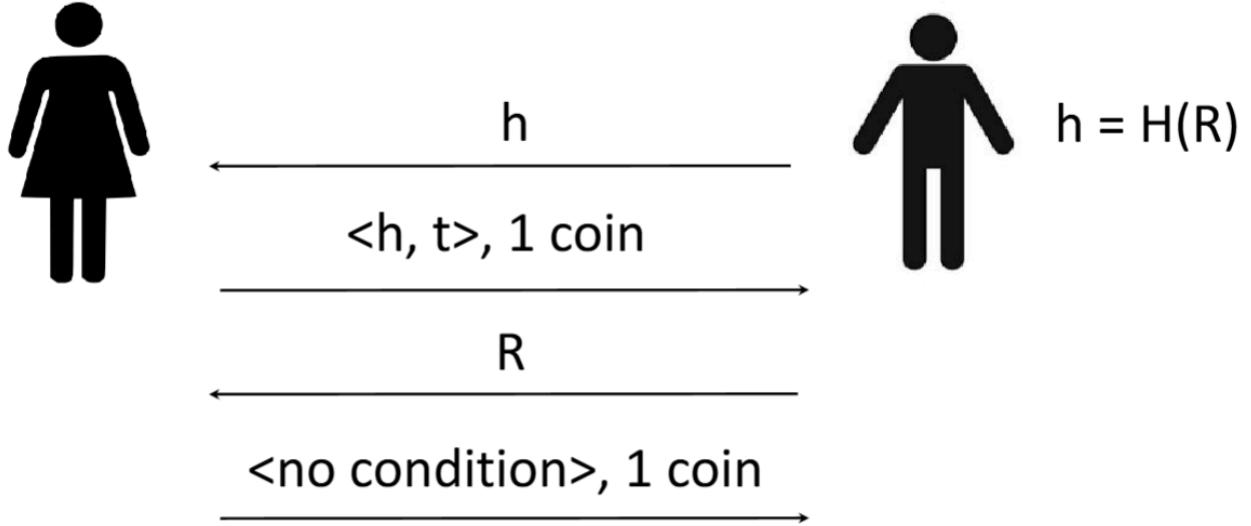
机制：通过“**版本递增 Increment version**”确认状态（每次更新版本号，双方签名），主链纠纷时以最高版本状态为准。

纠纷流程：

1. 一方触发纠纷，设置响应时间。
2. 各方提交最新状态（含版本号）。
3. 到期后，主链以最高版本状态结算。

The conditional transfer

条件转账：资金的转移 依赖特定条件的满足，条件不满足则转账不执行（或反向执行）



Hashed Time-locked Contract

条件转账也有类似时间锁机制

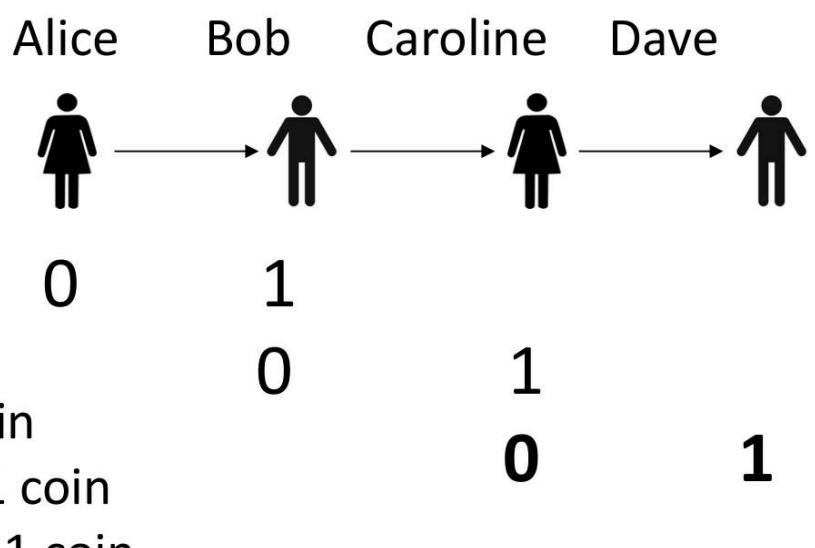
如果Alice离线等被动不配合，Bob可以提交状态和证据，主链强制按条件执行

如果Alice主动不配合，那么会失去所有资金

Multi-hop payments

Alice 和 Dave 之间 没有直接的链下通道，但通过 中间节点（Bob、Caroline）的通道接力，实现间接支付

每一步转账都是 链下操作（仅更新通道状态，不上主链），只有当通道关闭时才会触发主链交易



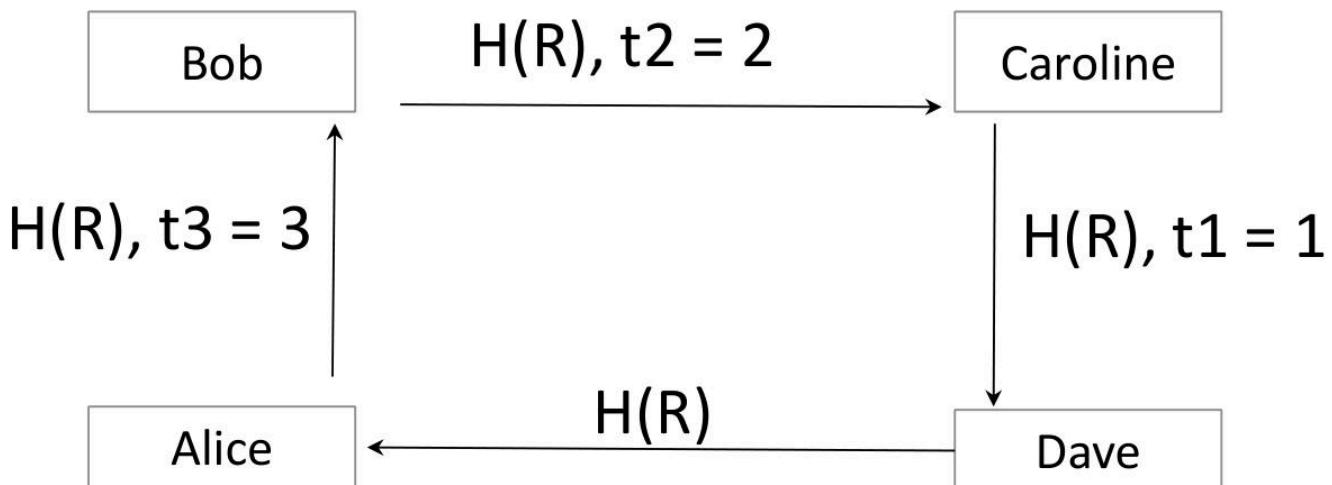
多跳网络同样支持哈希时间锁合约 (HTLC, Hash Time-Locked Contract)

收款人提供哈希值 ($H(R)$)：Dave 生成随机数 R ，计算其哈希值 $H(R)$ ，并将 $H(R)$ 提供给 Alice（付款人）

每个中间节点 (Bob、Caroline) 锁定硬币创建 HTLC，承诺：

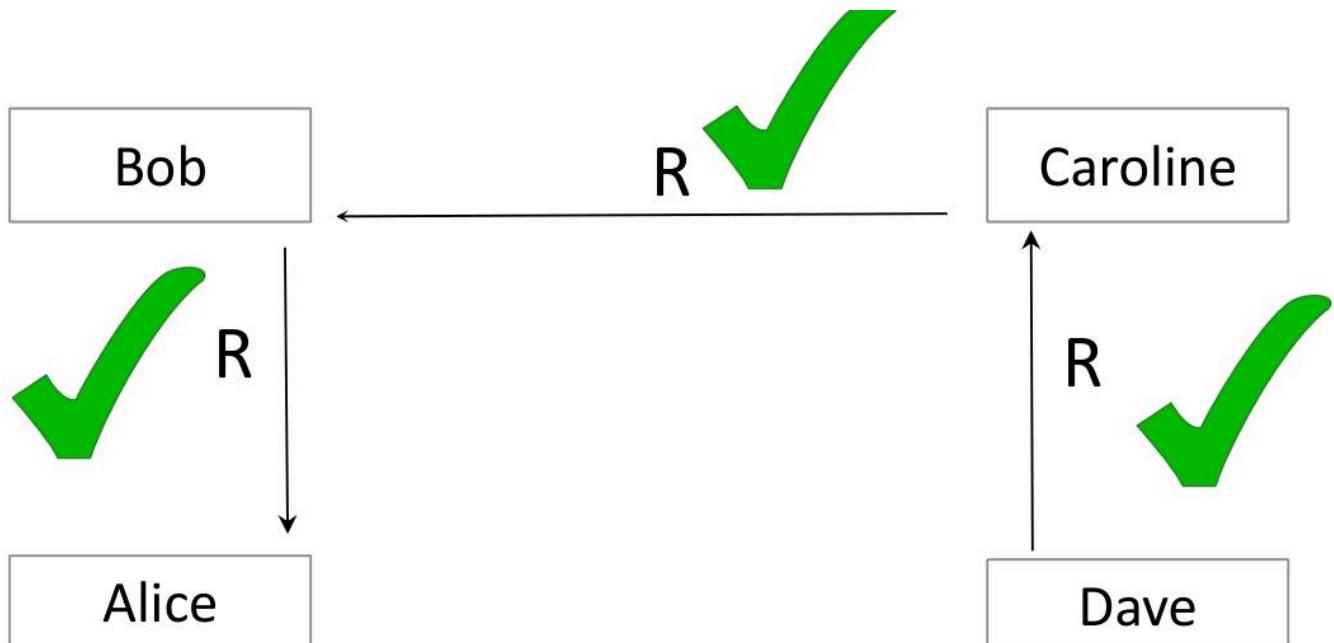
- 若收到预像 R (证明是收款人触发)，则将资金转给下一跳；
- 若超时 (时间锁到期)，则资金退回上一跳。

时间锁设计：每一跳的时间锁 **严格递减** (如 $t_3=3 > t_2=2 > t_1=1$)，确保资金只能 **从上游→下游** 流动 (下游超时更早，迫使收款人尽快揭示 R)。



最终，Dave 反向一跳跳地发送 R ，直到给到 Alice

Alice 收到 R 后更新通道状态



由于中间节点都锁定了硬币，中间通道如果作弊或等到时间锁到期，其资金都无法收回

如果最终交易正确进行，中间节点会获得手续费作为奖励

通过哈希时间锁定合约（HTLC）实现原子性

Lecture 8 - scaling - partII

Bridge contract

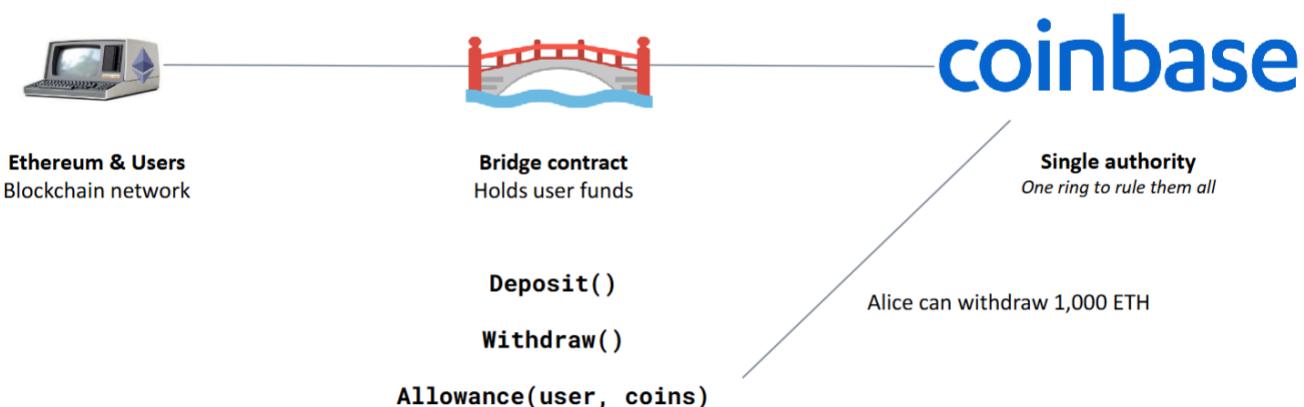
另一种扩容方法——桥接

如下图存在部署在以太坊上的桥接合约

用户将钱存储在桥接合约内容

中心化机构如coinbase作为唯一权威 **Single authority** 内部记录用户的可提取额度

用户能否取款取决于桥接合约是否收到唯一权威的授权



但链下数据库由第三方控制（可能是中心化机构、联盟或链下共识系统），如何让去中心化的桥合约“信任”中心化 / 半中心化的链下状态？这是桥接技术的核心挑战

除单权威桥 Single authority，还有多权威桥 Multi authority，加密经济桥 Crypto-economic，“共识桥”(Consensus Bridge) 等，但都依赖链下判断，不是真正的去信任 Trustless 方案

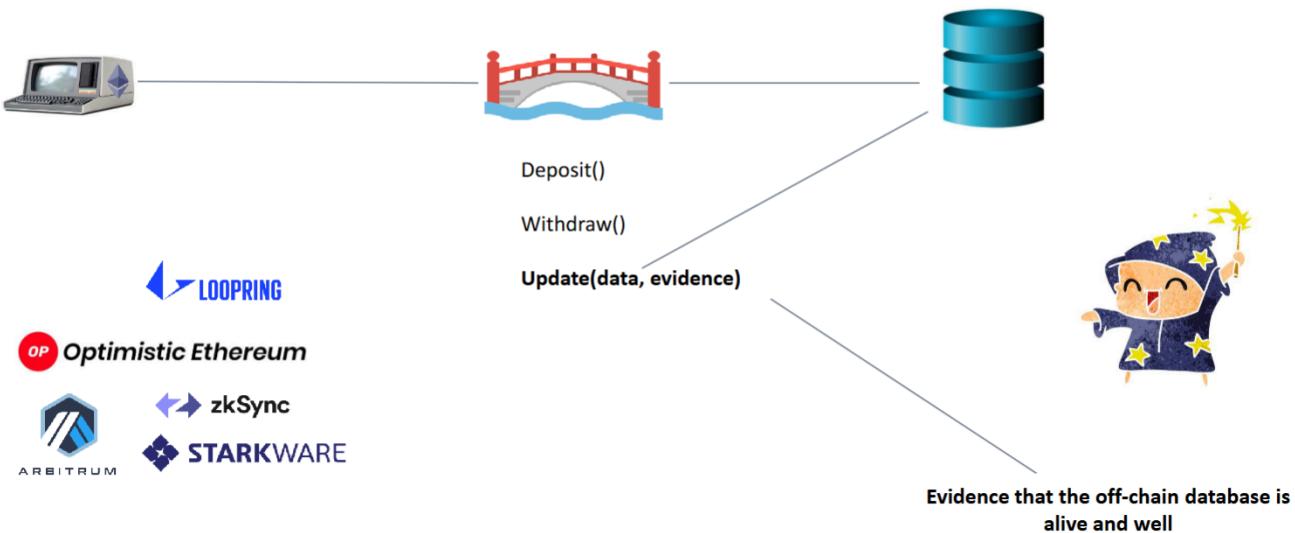
Validating Bridge

Plasma是验证桥的开端

如下图是验证桥的基本模式

链下数据库出示证据evidence

桥接合约验证证据并更新数据状态 `update(data, evidence)`



验证桥的所有验证逻辑都部署在以太坊合约上，利用以太坊的 **去中心化共识、不可篡改特性** 保障：若证据无效（如 ZK 证明错误、欺诈证明被成功挑战），桥合约会拒绝状态更新，保护用户资产

角色：

- **Honest user (诚实用户)**: L2 的普通参与者，无恶意行为，正常使用链下服务
- **Sequencer (排序者)**: 链下收集并排序用户交易，将零散交易整合成“交易批次”(决定交易执行顺序)
- **Executor (执行者)**: 执行 Sequencer 排好序的交易，更新 L2 状态。随后与 L1 桥合约交互如提交 ZK 证明

典型的流程如下：

1. 链下交易收集 (Sequencer 负责)

- 动作：用户发起 L2 交易（如转账）→ **Sequencer 收集交易**，暂存为“Pending 交易池”(可等待更多交易优化打包)。

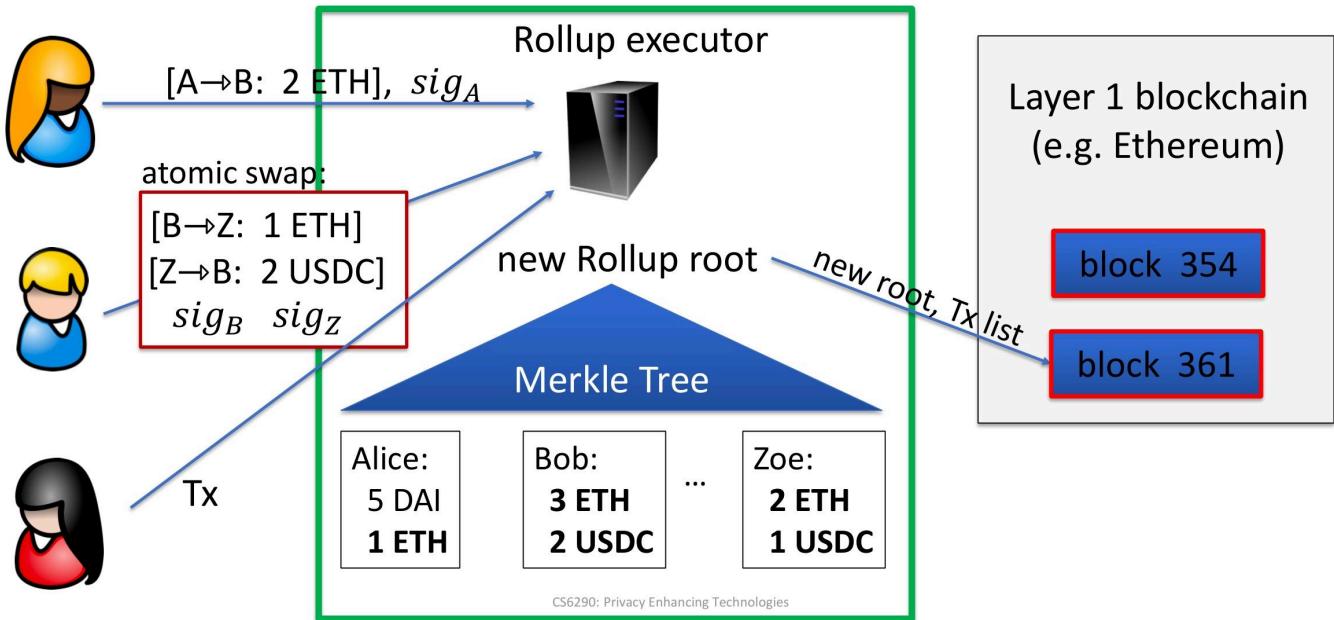
2. 链上排序锚定 (Bridge 合约负责)

- 动作：Bridge 合约读取 Pending 交易→ **按规则（时间、手续费）链上排序** → 标记为“待执行”(仍 Pending，需验证合法性)。

3. 执行 + 验证上链 (Executor 负责)

- 动作：Executor 执行排序后的交易→ **生成验证证据**（如 ZK 的有效性证明、Optimistic 的交易数据）→ 提交 Bridge 合约验证→ 验证通过后，**链上确认交易，更新 L2 状态**（资产转移、合约状态变更生效）。

L2 将 **数百笔 L2 交易打包成 1 笔 L1 交易**，随后编码得到默克尔树并提交根哈希给L1，L1只需要验证这 1 笔批量交易的合法性即可更新 L1 上记录的 Rollup 状态（从“当前世界状态”到“更新后世界状态”）



L2之内的交易都很迅速，而一旦涉及到存 deposit 取款 withdraw之类的与L1交互的交易，不但速度慢而且很昂贵

传统 L1 上的智能合约（如 Uniswap）每次交互都需上链，gas 高昂。Rollup 通过“**链下复制合约逻辑 + 状态批量上链**”，让用户能**低成本交互 DeFi 协议**（如 Uniswap），同时保留 L1 的安全兜底

- 初始化时，将L1的链上状态复制到L2
- 随后在L2上执行智能合约逻辑
- 最后将L2的合约执行后状态生成默克尔树提交给L1并更新

Adversarial threat model

然而，在L2中由于Sequencer享有对交易排序的能力，L2中的交易仍有被作恶提交恶意状态的可能性

但只要任何一个诚实用户提交欺诈证明（Optimistic Rollup）或验证有效性证明（ZK Rollup），L1就可以根据证明回滚恶意状态

对抗者 Adversarial 的四大能力（威胁维度）

1. 消息流控制（Message flow control）

- 对抗者可**任意控制 L2 消息的顺序、丢弃交易，但无法阻止发往 L1 的消息**

2. 腐蚀多数参与者（Corrupt nearly all parties）

- 对抗者可**收买所有 Sequencer + N-1 个用户，但无法腐蚀“至少 1 个诚实用户 + L1 区块链（智能合约）”**

3. 经济动机（可选，Financially motivated）

- **约束：**对抗者需在 L1 质押**安全保证金**，若被发现欺诈（如提交虚假状态根），保证金会被没收。

4. 无法破解密码学 (Cannot break cryptography)

- 假设：对抗者无法攻破 哈希、数字签名、SNARK 等密码学工具。

Security properties

核心目标是保障链下数据库的安全性safety和活性liveness

因此验证桥检查：

- 数据可用性 (Data availability)：**链下数据库的所有状态更新（如交易、账户变化）是否公开可获取？
- 状态转换完整性 (State transition integrity)：**链下数据库的所有状态更新是否合法、格式正确？
- 抗审查性 (Censorship-resistance)：**用户能否自主确保交易最终被执行（即使遭遇审查）？

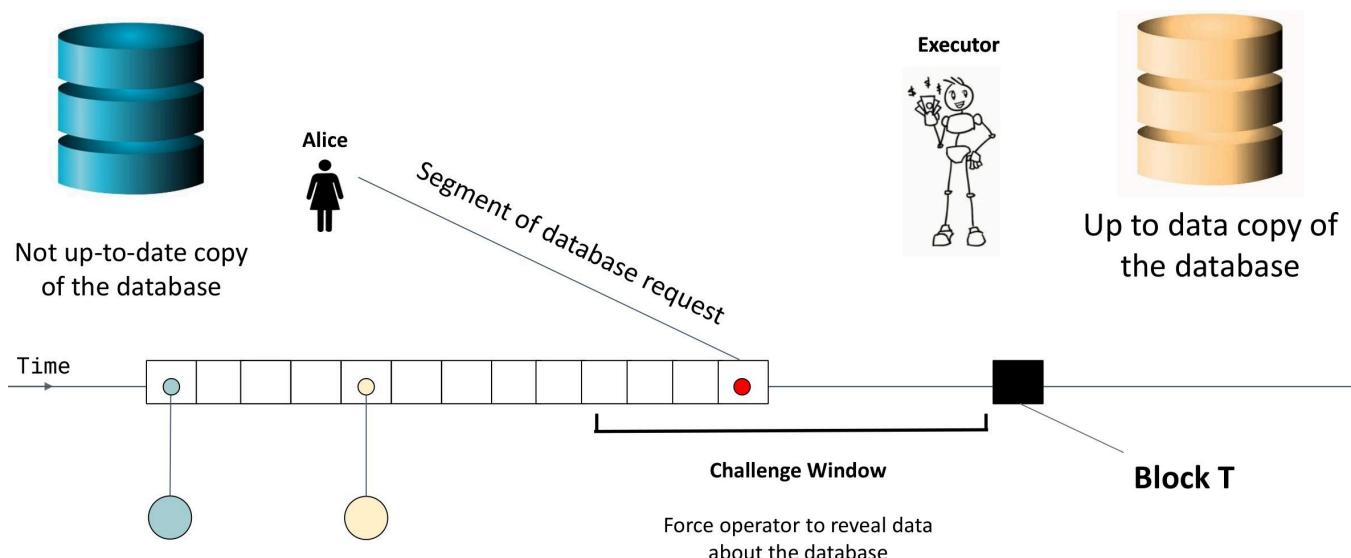
Data availability

解决方案众多，本质都是“强制让交易数据对公众可见或可验证”，一种典型方式是：

挑战流程 (Challenge Process)：任何人可挑战“交易数据不可用”，若成功，攻击者受罚（如 Optimistic Rollup 的 7 天挑战期）

Plasma 的数据可用性挑战流程：

- 执行者暗改链下数据，只上链哈希（检查点）；
- 用户发现本地数据没同步，要求执行者公开交易记录；
- 若执行者耍赖（扣留数据），用户在 L1 触发挑战，强制执行者限时暴露数据片段，否则罚没质押金。



然而Plasma的数据可用性挑战有以下局限性：

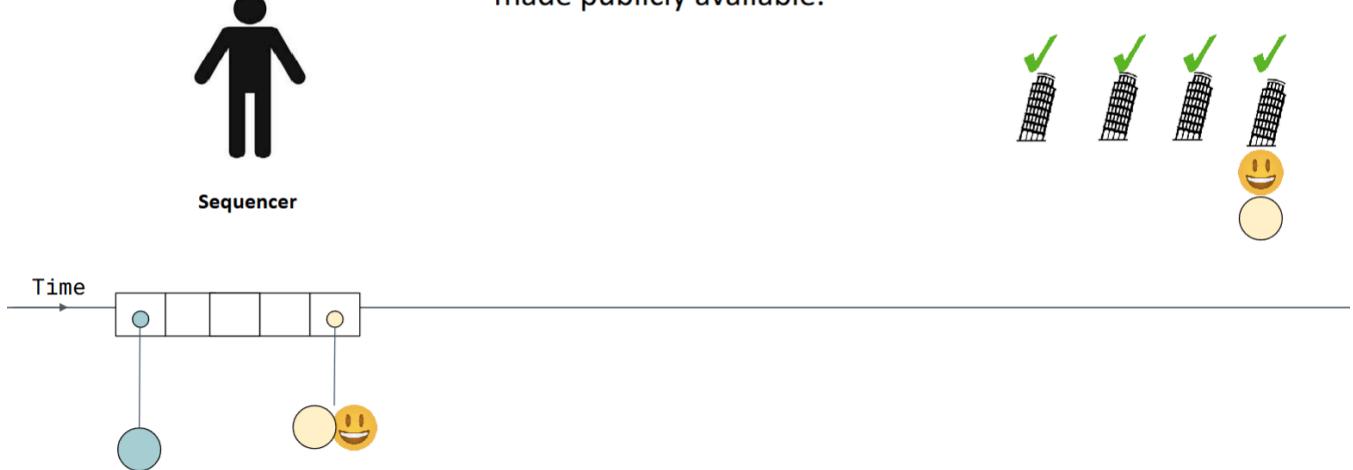
- 请求盲盒：**该查余额、存储还是全量更新？没标准；
- 验证超载：**既要证数据存在，又要证状态合法，UTXO 硬凑仍复杂；

3. 功能瘸腿：仅支持转账，撑跑 DeFi 等智能合约；
4. 容灾兜底：链下数据太大，L1 存不下，用户离线就被卡资产。

另一种解决方案是数据可用性委员会（Data Availability Committee）

Sequencer（排序者）处理链下交易后，数据可用性委员会必须签字，证明“链下数据已公开可查”

Data availability committee must sign-off and vouch the off-chain data was indeed made publicly available.



但委员会存在叛变作恶的可能

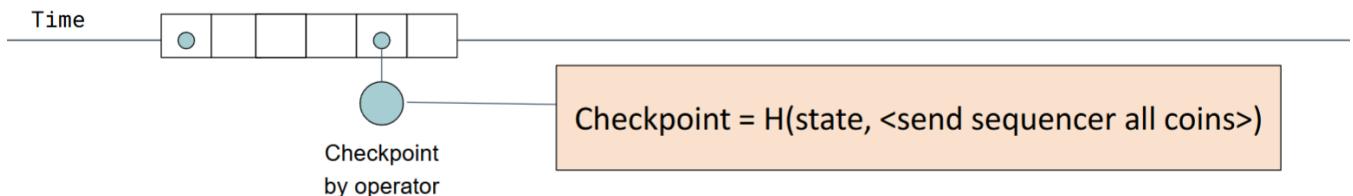
对于**Optimistic Rollup**（乐观 Rollup），委员会影响 safety。若委员会和 Sequencer 合谋 扣留数据 → 没人能提交欺诈证明（因为拿不到交易数据）→ 恶意状态会被默认接受，攻击者可偷资产（如篡改余额）→ 委员会的失败，直接摧毁 Optimistic 的安全根基

而对于**ZK Rollup**（有效性 Rollup），委员会只影响 liveness。ZK 证明本身已 数学验证状态合法性（即使数据被扣留，之前的证明仍有效），所以 资产安全不受影响（没人能篡改状态）。但用户可能 无法执行 L1 退出机制（如提款）→ 因为生成提款证明需要链下数据，委员会扣留则流程卡壳

State transition integrity

这一安全属性的目标是确保链下数据库的更新正确

L2 的操作者（如 Sequencer）可能会提交 **检查点（Checkpoint）**，但其中的交易是恶意的（比如图中“把所有币转给自己”），试图篡改 L2 数据库的状态



有两种思路：

1. Optimistic Rollup (乐观派)：

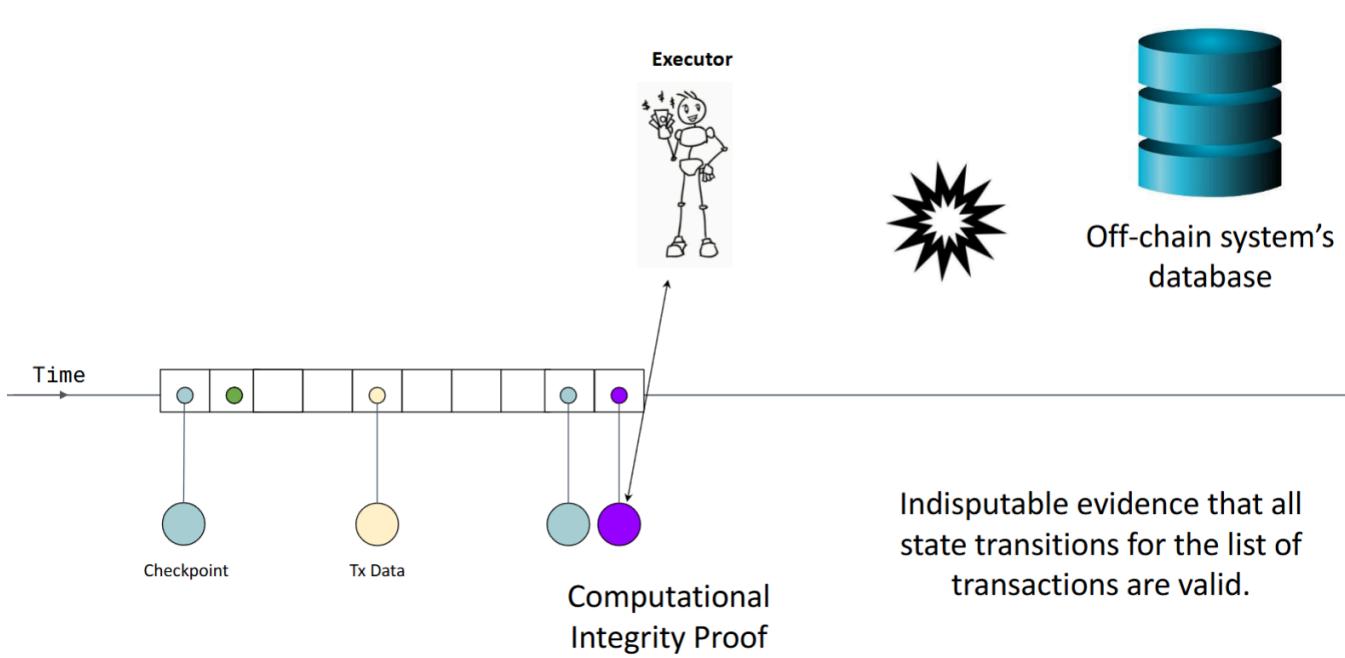
- 先默认交易有效，**开放挑战期**（如 7 天）。
- 若有人发现恶意交易，可提交**欺诈证明**（比如证明签名伪造、余额不足），推翻恶意状态。

2. ZK Rollup (密码学派)：

- 操作者提交状态时，必须附带**有效性证明 (ZK 证明)**，数学上证明“所有交易都合法”。
- L1 直接验证证明，无需等待挑战，从根源杜绝恶意交易上链。

ZK Rollup:

1. 执行者 (Executor) 收集 L2 交易，执行后生成**交易数据 (Tx Data)**
2. 执行者计算**计算完整性证明 (Computational Integrity Proof)** 如SNARK（证明签名有效，账户真实，状态转移State Transition正确）
3. 把证明提交 L1，L1 直接验证，**一旦通过，状态更新就不可争议**



ZK Rollup将信任从实体身上转移到了对密码学证明身上

Optimistic Rollup:

- 1.**第一步：无证明提交：**协调者直接把链下状态更新提交到 L1（默认交易合法，不上传证明，省成本）。
- 2.**第二步：7 天挑战期：**若状态无效（如恶意转账、合约逻辑错误），任何人可提交欺诈证明：

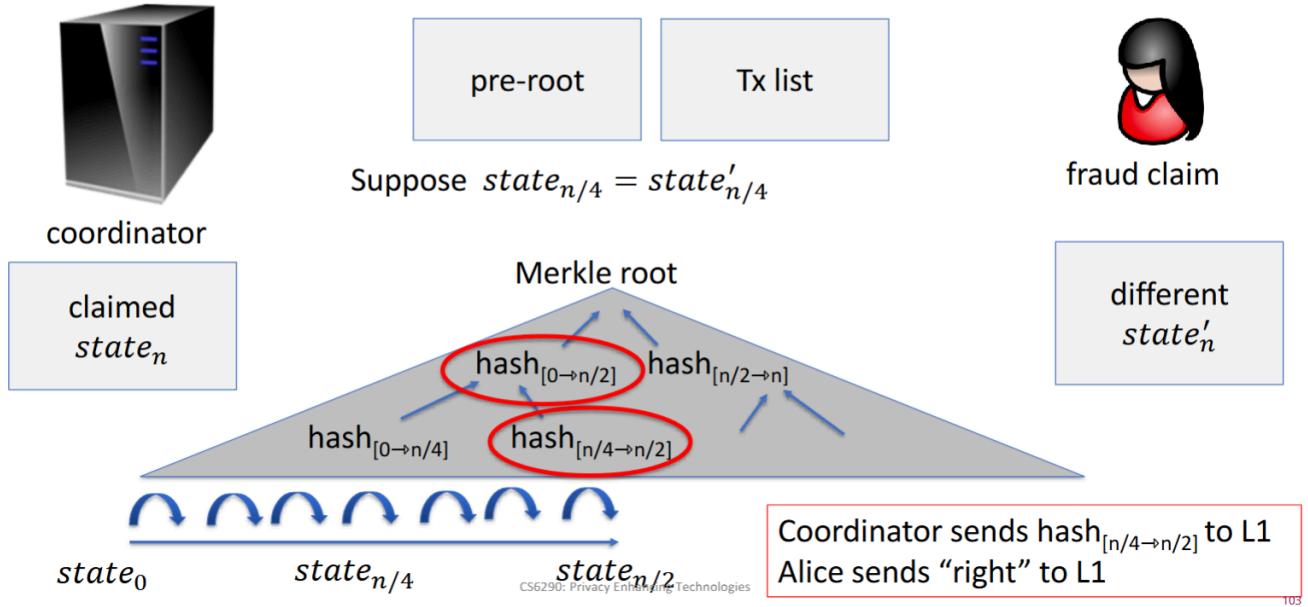
- 成功：协调者质押金被罚没 (slash)；
- 失败：挑战者付手续费（防止乱挑战）

然而，如何低成本证明欺诈 Prove Fraud?

如果重新计算所有疑似作恶的交易来证明，L1上的计算成本太过高昂

一种方法是进行**二分搜索 Binary Search**

通过拆分验证的方式减少验证计算量



Optimistic Rollup本身具有一个巨大缺点：7天挑战期的存在严重影响用户体验

Summary on L2

security	SNARK validity proofs	Fraud proofs
data availability	Tx data on L1 chain	zkRollup
	Tx data in a DAC	Validium (reduced fees, but higher risk)

	SNARK 有效性证明 (ZK)	欺诈证明 (Optimistic)
Tx 数据上 L1 链	zkRollup (如 zkSync、Polygon zkEVM) <input checked="" type="checkbox"/> L1 级安全 (证明无法伪造) <input checked="" type="checkbox"/> 交易数据公开, 无信任假设 <input checked="" type="checkbox"/> 即时最终性 (无挑战期)	Optimistic Rollup (如 Arbitrum、Optimism) <input type="checkbox" warning=""/> 7 天挑战期 (最终性延迟) <input type="checkbox" warning=""/> 依赖博弈, 需承担“没人挑战”的风险 <input checked="" type="checkbox"/> 交易数据公开, 安全锚定 L1

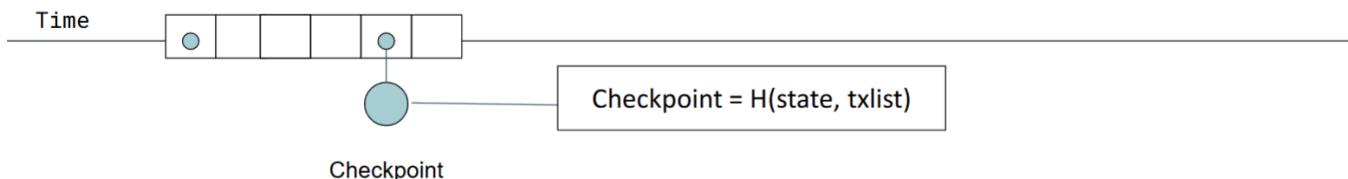
	SNARK 有效性证明 (ZK)	欺诈证明 (Optimistic)
Tx 数据在 DAC	Validium (如 StarkEx 的可选模式) <input checked="" type="checkbox"/> ZK 证明保障状态合法 ⚠ 数据依赖 DAC, 若委员会藏数据，用户无法验证 <input checked="" type="checkbox"/> 成本极低 (省 L1 存储费)	Plasma 系 (如早期 Plasma MVP) ⚠ 数据依赖 DAC + 博弈挑战，风险叠加 ⚠ 功能局限 (仅支持支付，难跑智能合约) ✖ 已被 Rollup 取代，仅存理论意义

Censorship resistance

当 Rollup 的协调者 (Coordinator)、排序者 (Sequencer) 或执行者 (Executor) 拒绝处理某笔交易 (即 “审查 Censorship”) 时，用户应怎么做？

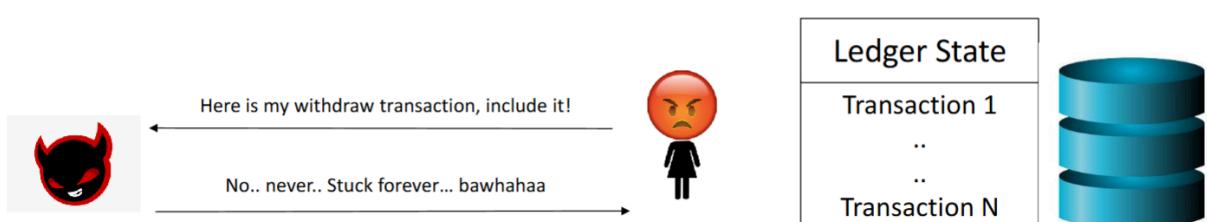
一种方法：

1. 用户直接将交易提交到 L1 的 Rollup 合约
2. L1 合约会记录这笔 “被审查的交易”，并触发规则：此后所有提交到合约的状态更新（如批次交易、状态根），必须包含这笔交易，否则合约直接拒绝接受更新
3. 若排序者在窗口时间内仍坚持不处理这笔交易，Rollup 的状态更新会被 L1 合约永久拒绝 → 整个 Rollup 系统冻结



How can I withdraw my funds if the sequencer does not cooperate?

Layer-2 database



用户永远可以向 L1 求助

一些常见攻击方式和解决：

作恶场景	核心解法
摆烂不处理交易	L1 强制检查点最低交易量，执行者质押金违约罚没。
卡时间 / 低费挤占交易	交易按 Gas 费排序，开放无许可执行者准入（诚实用户可竞争接管）。

作恶场景	核心解法
Optimistic 下恶意挑战拖延	挑战者需质押，无效挑战扣押金（抬高作恶成本）。
通用逻辑	用经济博弈（质押）和去中心化竞争（无许可准入）替代信任，平衡安全、效率、成本。

Security properties (summarised)

1. 数据可用性 (Data Availability)

核心问题：诚实用户如何获取 L2 交易历史，复现与全网一致的账本？

解法：

- **ZK Rollup/Optimistic Rollup**: 交易数据上 L1 链（如 Calldata），依赖 L1 的去中心化存储，任何人可查；
- **Validium/Plasma**: 数据存在 DAC（委员会），成本低但依赖其诚实（风险更高）。

意义：若数据不可见，用户无法验证自己的资产状态，L2 会沦为“黑箱”。

2. 状态转换完整性 (State Transition Integrity)

核心问题：如何让 L1 相信 L2 的所有交易都合法（签名对、余额够、合约逻辑正确）？

解法：

- **ZK 系 (如 zkRollup)**: 用 SNARK 证明“批量交易的执行完全合法”，L1 只需验证证明（数学上的铁证）；
- **Optimistic 系 (如 Arbitrum)**: 先“乐观”提交状态，开放挑战期，若欺诈则通过多轮博弈（如二分法）在 L1 验证争议步骤。

意义：杜绝 L2 随意篡改状态（如凭空造币），确保资产变化符合规则。

3. 抗审查性 (Censorship Resistance)

核心问题：诚实用户如何绕开 Sequencer 等中间层，强制提取 L2 资产？

解法：

- **链上撤离机制**: 用户直接向 L1 提交默克尔证明（证明资产归属），触发强制提款；
- **强制交易包含规则**: L1 合约规定“新状态更新必须包含被审查交易”，否则冻结 L2 系统，倒逼中间层配合。

意义：防止中间层垄断交易（如拒绝处理提款），保障用户对资产的最终控制权。