[1. 实训目的 2](#_Toc186292214)

[2. 实训时间和地点 4](#_Toc186292215)

[3. 实训内容 5](#_Toc186292216)

[3.1. 实训期间完成的学习任务 5](#_Toc186292217)

[3.2. 实训期间完成的实践项目 5](#_Toc186292218)

[3.2.1. 需求分析 5](#_Toc186292219)

[3.2.2. 总体设计 7](#_Toc186292220)

[3.2.3. 详细设计 29](#_Toc186292221)

[3.2.4. 编码 43](#_Toc186292222)

[3.2.5. 测试 44](#_Toc186292223)

[4. 实践收获与体会 45](#_Toc186292224)

[5. 致谢 46](#_Toc186292225)

[参考文献 47](#_Toc186292226)

# 1. 实训目的

在当今信息化时代，大数据技术的发展迅猛，已成为各行各业不可或缺的一部分。作为软件学院的一名学生，我深刻认识到大数据领域的广阔前景和对专业人才的迫切需求。为了培养具备先进复合型、实用型和工程型特质的大数据人才，我校与多家IT企业紧密合作，推出了“大数据项目综合实训”的实施方案。这项实训不仅是我个人学习和成长的机会，更是我为未来职业生涯奠定基础的重要环节。

1. 提高项目实战能力

在大数据项目综合实训中，我将有机会参与大型项目的实战训练。通过实际操作，我的项目实战能力将得到显著提升。我将学习如何在真实的项目环境中应用所学的理论知识，掌握数据采集、存储、分析和处理的全过程。这不仅能够帮助我了解大数据项目的整体架构，还能让我熟悉项目实施中的各种挑战和解决方案。

2. 增强市场适应能力

随着市场对大数据人才需求的不断增加，具备良好的市场适应能力显得尤为重要。在实训中，我将与来自不同背景的团队成员合作，学习如何在团队中发挥自己的作用，处理复杂的项目需求。这种团队合作的经验将使我在未来的职场中更具竞争力，也能让我更好地适应快速变化的市场环境。

3. 提升就业率和就业质量

通过参与实际项目的开发和测试，我将积累宝贵的实践经验。这些经验不仅能够为我的简历增添亮点，还能帮助我在面试中更具优势。实训旨在实现高就业率和高就业质量的目标，确保我在毕业后能够顺利找到一份理想的工作。企业对实践经验的重视，使得我在求职时能够展示出与众不同的竞争力。

4. 掌握数据处理全流程

实训的一个重要目标是让我熟练掌握大数据项目中数据采集、存储、分析和处理的全流程。通过参与实际项目，我将深入理解每个环节的重要性和相互关系。掌握这些技能后，我将能够更加自信地面对未来的工作挑战，独立处理复杂的数据问题。

5. 积累实际项目经验

实训要求每位学生至少参与一个实际项目的开发和测试。我会主动争取在项目中担任重要角色，积极参与到项目的各个阶段。这种实践经验将帮助我建立起对项目生命周期的全面理解，让我在未来的工作中更加游刃有余。

6. 加强与IT企业的联系

实训过程中，我将有机会与IT企业的专业人士直接交流和互动。这不仅能让我获取行业前沿的信息，还能让我建立起与企业的联系，为未来的就业铺平道路。通过这一过程，我能够更好地理解企业对人才的期望，明确自己在职业发展中的方向。

这次实训不仅仅是一次学习的机会，更是我职业生涯的重要起点。希望通过这次实训，我能为自己的未来奠定坚实的基础，努力成为一名符合时代需求的优秀大数据人才，实现个人价值的最大化。

# 2. 实训时间和地点

1. 实训时间: 2024年10月25日 ~ 2025年1月12日

2. 实训地点: 辽宁工程技术大学(校内)

# 3. 实训内容

## 3.1. 实训期间完成的学习任务

## 3.2. 实训期间完成的实践项目

### 3.2.1. 需求分析

#### 3.2.1.1. 项目简介

区块链技术，是在一个由相互缺乏信任的节点组成的网络环境中，通过“竞争—验证—同步—竞争”的动态循环，解决各节点如何达到可信共识的问题，最终成为允许个体不经过第三方认证而开展有效可信合作的新型技术平台。

区块链去中心化的特性契合抗毁生存的军事需求。区块链采取分布式核算和存储，不依赖第三方管理机构，每一个节点都存储着完整的数据备份，一个节点出现问题，其他节点会继续数据的更新和存储，从理论上讲，只要有一个节点存在，就能保证全部信息不会丢失。

无人机集群由两架及以上数量的多功能无人航空飞行器共同组成，在人工智能的控制下，可以自主完成很多任务。多个无人机通过彼此的感知交互、信息传递来协同工作，在险恶的环境下可以低成本完成多样性的复杂任务。集群整体具有分布式自组织结构、简单自适应个体、灵活群体性模式、智能式紧密，可实现单个无人机行为自主决策、无人机间行为协同，最终产生能力涌现的自主式空中移动系统。

由此出发，本次实训，学生计划实现一个基于区块链原理的异构无人平台协同规划系统。

#### 3.2.1.2. 技术简介

#### 3.2.1.3. 功能需求

无人平台任务需求动态变化、平台性能异构以及通信条件受限等问题均制约了多无人平台的协作能力.同时无人平台节点可根据任务位置、执行时间对任务需求进行分析判断并自发移动，导致节点网络拓扑结构迅速发生改变，为多平台的自主任务规划带来极大挑战。为此需要建立无人平台的状态、使用约束的同一描述模型，提升动态环境下无人平台的自主规划能力。

突发任务通常具有重要度高和时效性快的要求，无人平台在受领这样的任务后，事先拟定的任务分配计划无法适用于变化后的任务需求，极易导致部分任务无法截止期限内完成。同时，无人平台存在能源耗尽以及损毁失效的问题，致使其正在执行的任务被迫中断，制约了系统对任务需求的响应能力。为此，需要构建基于快速响应机制的无人平台应急任务分配模型，为任务计划动态调整提供科学依据，以提升系统的敏捷性。

任务场景的日益大规模化和复杂化、无人平台资源的配置策略、协同调度之间的强耦合关系，使得平台配置成为一个难以直接求解的复杂联合优化问题。现有技术在求解此类复杂问题时，需要大量的运行时间和强大的硬件技术支撑。因此，需要在计算资源有限的情况下，为系统快速生成多无人平台配置策略，令多个平台合理分配任务节点。

由于现阶段无人平台的计算能力、存储能力以及通信能力等限制，无人系统在面临远距离跨域行动时存在较大问题，异构平台之间的信息交互链路无法保证实时通常，这也进一步降低了多平台协同调度的能力，极易导致任务执行失败。因此，依赖局部通信链路下的分布式预测模型，确保区域信息可达的邻居平台状态信息共享，提升无人系统稳定性和平台协同执行任务的能力。

围绕大规模的异构无人平台构建协同配置架构是保证无人系统平稳运转的首要问题。无人平台因其移动性强、灵活性高等特点，使得系统在执行任务期间资源的组织关系与拓扑结构高度动态。同时，由于地理位置分散、平台行动独立、资源载荷不同等特征，加大了无人平台协同完成任务的难度。因此需要设计适当的协同架构，实现对系统内无人资源的同一组织与规划，从而驱动系统更高效运转。

### 3.2.2. 总体设计

#### 3.2.2.1. 应用架构

区块链技术是分布式存储、点对点传输、共识算法、密码技术等计算机技术的融合创新应用。按照区块链通用分层架构，结合无人平台运行应用场景，规划本系统区块链技术应用架构。

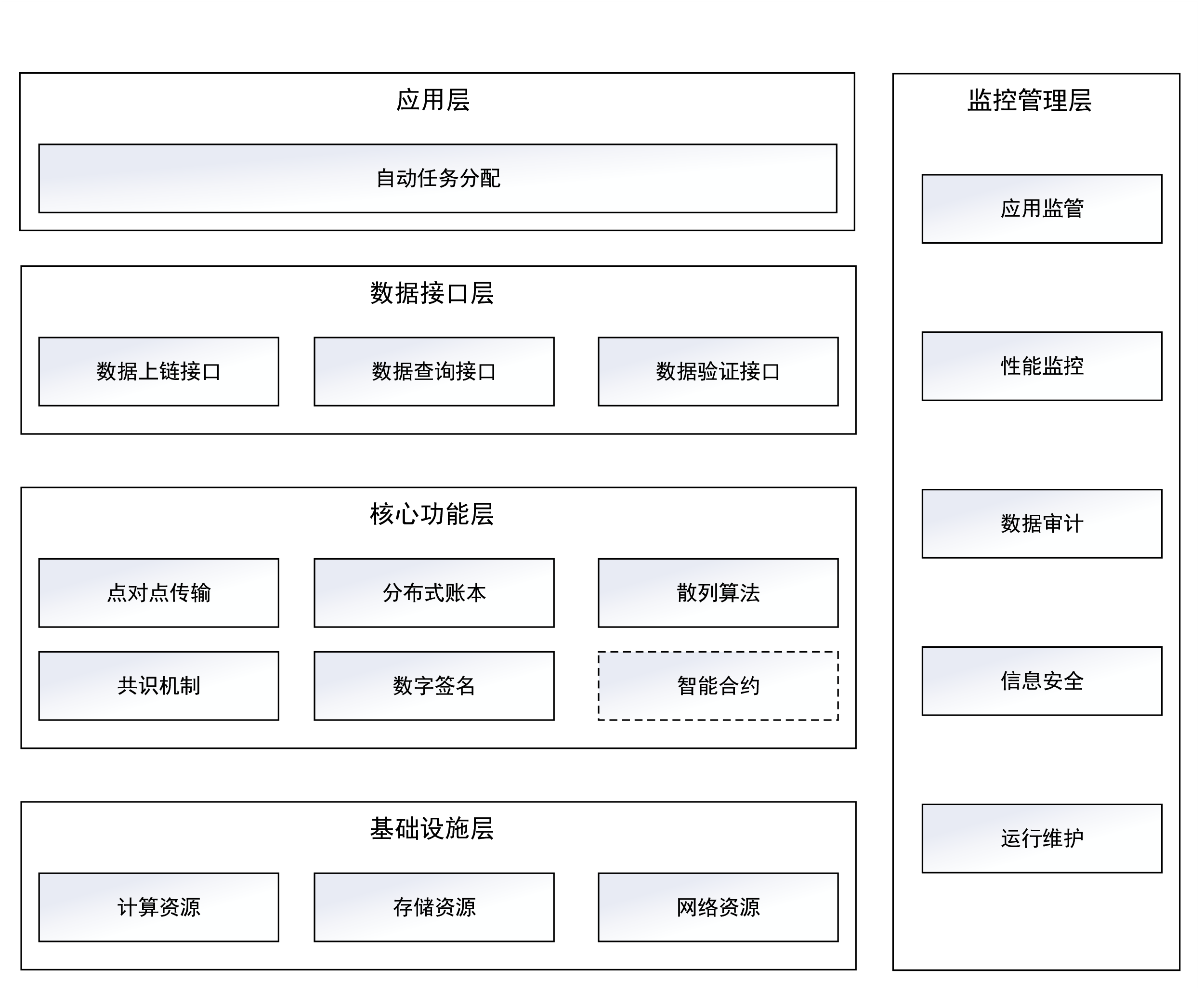


图1-1 区块链技术应用架构

(1) 基础设施层

基础设施层为区块链技术正常运行提供所需的物理承载及相关运行环境，包括以下要素：

a. 计算资源，为区块链系统的运行提供算力支持；

b. 存储资源，为区块链系统的运行提供数据写入、查询和存储支持；

c. 网络资源，为区块链系统的运行提供数据链接和数据路由支持。

(2) 核心功能层

核心功能层为区块链技术应用提供核心功能支持，包括以下要素：

a. 点对点传输，实现平台入网、任务发现以及上链数据的发布与扩散；

a. 分布式账本，实现区块链上数据的记录和维护；

c. 散列算法，实现数据散列值计算，降低区块链上数据的体量；

d. 共识机制，保证区块链上数据的一致性和可用性；

e. 数字签名，保障区块链参与者身份的真实性、重要数据的完整性、数据对应业务操作行为的不可否认行。

(3) 数据接口层

数据接口为使用区块链核心功能提供接口服务，包括以下要素：

a. 数据上链，响应系统数据上链请求，完成待认证数据的上链；

b. 数据查询，响应系统数据查询请求，完成待查询数据的上链状态查询；

c. 数据验证，响应系统数据验证请求，完成待验证数据的真实性和完整性校验。

(4) 应用服务层

应用服务层提供标准化的服务程序支撑，在本次实训中，包括以下要素：

a. 自动任务分配，根据平台入网情况、任务数据以及系统数据上链情况对待分配任务、突发任务进行自动分配。

(5) 监控管理层

监控管理层为基于区块链原理的异构无人平台协同规划模拟系统运行过程持续保持安全、合规和可靠提供支撑，包含以下要素：

a. 应用监管，提供数据溯源和数据检索服务；

b. 性能监控，实现对区块链系统运行性能的实时监控；

c. 数据审计，实现对区块链应用过程中的数据合规性的管理；

d. 信息安全，保障区块链运行过程中的信息安全；

e. 运行维护，为区块链系统运行异常恢复提供支持。

#### 3.2.2.2. 模块化设计

(1) 区块链

i. 介绍

网络（Network）中的每个完整节点（Peer）独立存储一个区块链，其中仅包含由该节点验证的区块（Block）。当多个节点的区块链中都有相同的区块时，它们被认为是共识的。这些节点为维持共识而遵循的验证规则称为共识规则（Consensus Rules）。

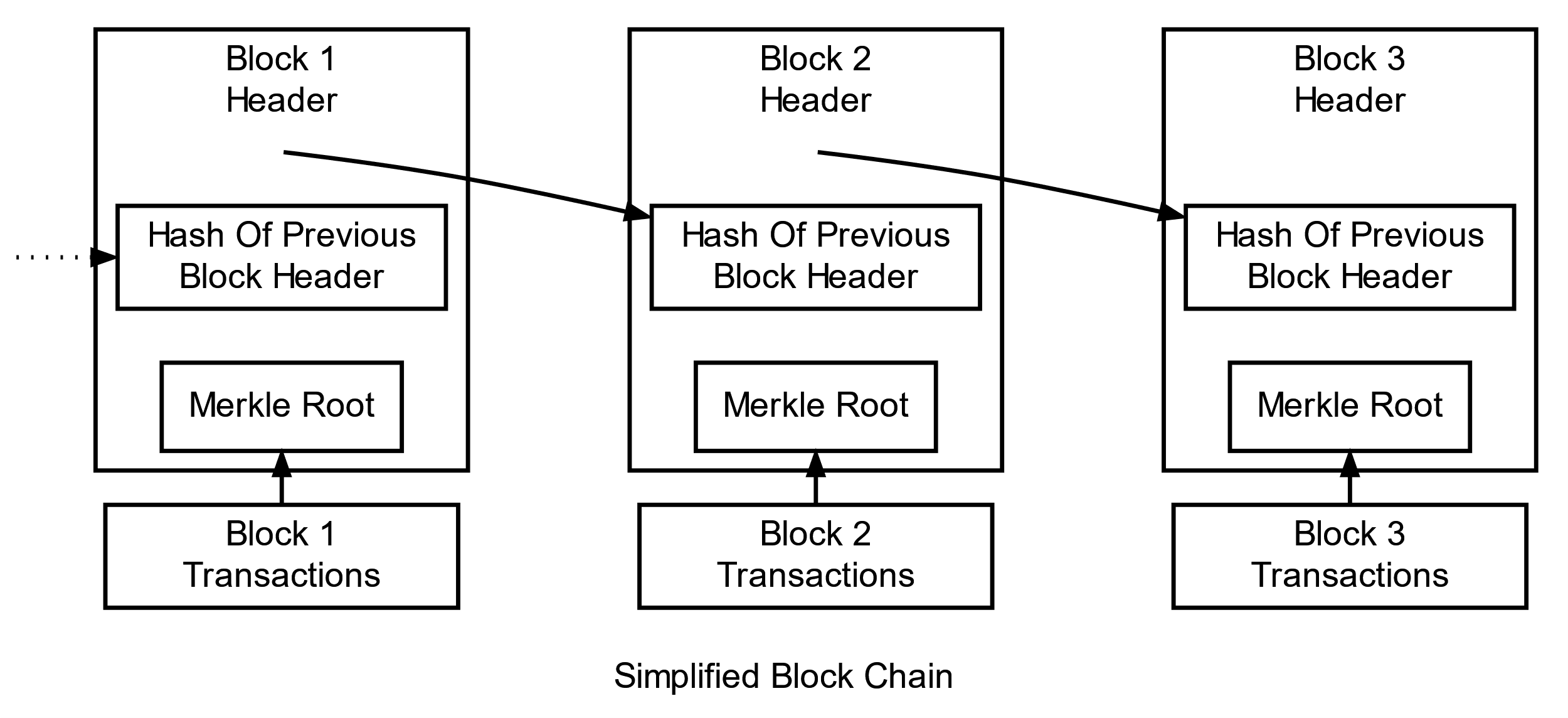


图 1-2 区块链概述

上图显示了区块链的简化版本。一个或多个新交易的区块被收集到区块的 Transactions 部分。每笔交易的副本都经过哈希处理（SHA256()），然后对哈希进行配对、哈希处理、再次配对和再次哈希处理，直到只剩下一个哈希值，即默克尔树（Merkle Tree）的默克尔根（Merkle Root）。

默克尔根存储在区块头（Header）中。每个区块还存储前一个区块头的哈希值（Previous Block Hash），并将区块链接在一起。这确保了如果不修改记录交易的区块和后续区块，就无法修改交易。

每笔交易都会花费一个或多个早期交易收到的概念币（Seon），即一笔交易的输入（Input）是前一笔交易的输出（Output），这样交易也被链接在了一起。

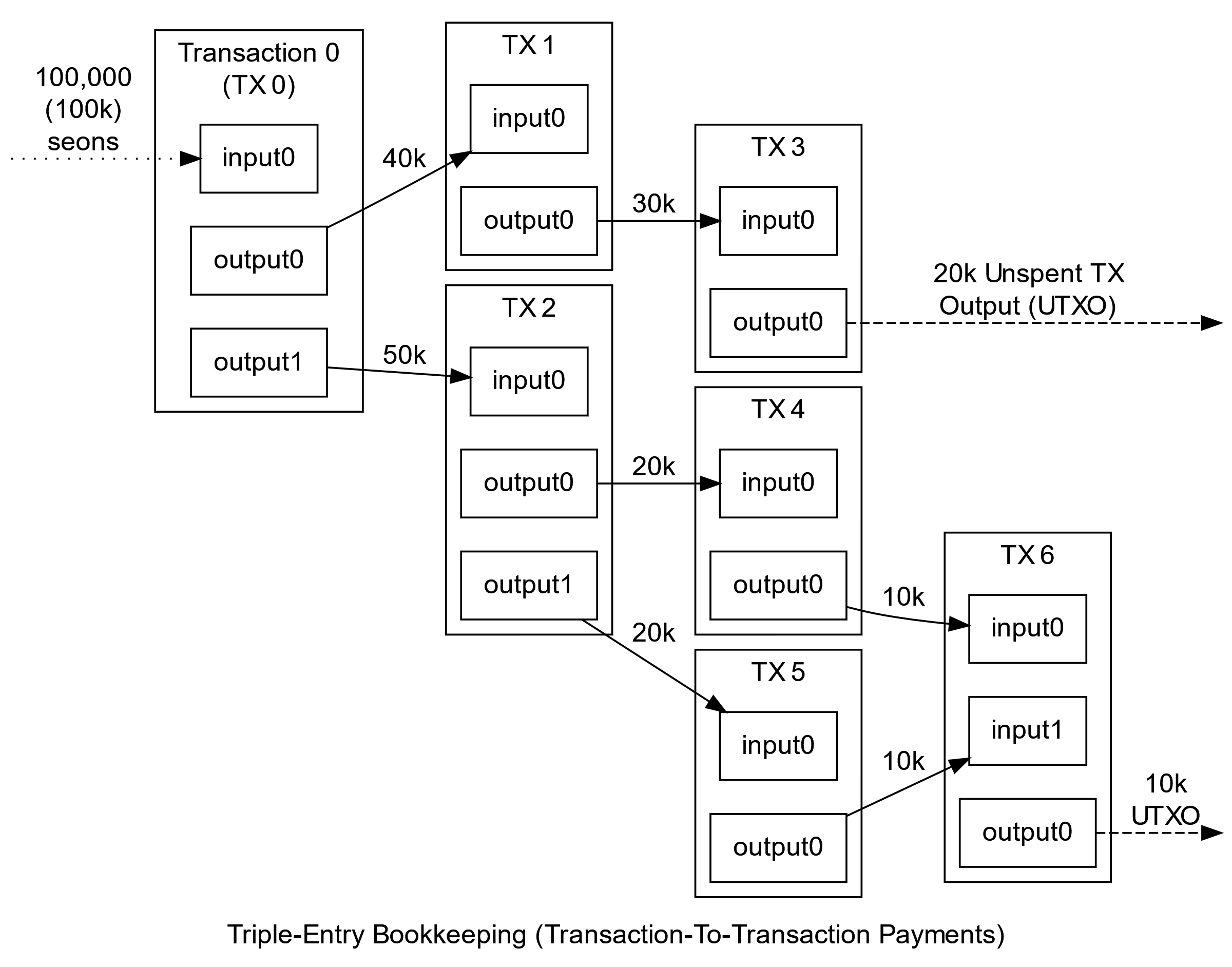


图 1-3 交易传播

当发送到多个地址时，单个交易可以创建多个输出。每个交易的输出只能作为输入被使用一次，任何后续的引用都是被禁止的。

每个输出都与特定的交易识别符（Transaction Identifier，TXID）相关联。TXID是签名交易的哈希值，用来唯一标识一笔交易。

由于每笔交易的输入都只能被使用一次，因此区块链上所有交易的输出可以分为两类，未花费交易输出（Unspent Transaction Outputs，UTXO）和已花费交易输出。尚未被用作其它交易的输入，任然可以被花费；已经被用作其它交易的输入，无法再次使用。为了保证交易的有效性，交易必须只使用 UTXO 作为输入。

ii. 共识规则

区块链是一个去中心化的数据库，由网络中的许多参与者共同维护，这些参与者是匿名的，彼此之间没有信任关系。为了确保区块链的安全性和完整性，要求每个新生的区块必须证明其创建过程中对任务做出了积极行动，这里将其称为“任务证明”或“行动证明”（Proof of Mission）。

如果某个节点（例如，不诚实的节点）试图修改区块链中已存在的块（例如，篡改交易记录），它必须对某任务重新进行积极行动。这意味着，该节点需要伪造一条从任务发现到完成或放弃的完备时间线并且至少具备对等体量的行动。而诚实的节点只需专注于添加新的区块，并积极响应当前任务，而不需要回溯修改旧块，这种设计确保了诚实参与者在网络中的活动是相对有效的。反之，试图进行欺诈行为的节点则必须付出更高的代价。

将区块链接在一起使得在不修改所有后续区块的情况下无法修改任何区块中包含的交易。因此，修改特定区块的成本随着添加到区块链中的每个新区块而增加，从而放大了任务证明的效果。

任务证明本质上是构建一条任务生命周期与节点行动记录的时间线。与一个好的加密哈希算法结合，任何试图修改链上数据以使哈希值变得可预测的尝试都是徒劳的。

iii. 区块高度和分叉

任何成功添加到区块链中的区块在区块链中被称为“区块高度”（Block Height），它指的是从创世区块（区块0，Genesis Block）到这个区块之间的区块数量。区块高度是用来标识不同区块的一个简单方法。

当两个或多个节点几乎同时添加新的区块时，这些区块会被赋予相同的区块高度。由于它们都是在相同的时间段内创建的，因此它们在区块链上都是有效的。由于多个节点同时添加新块，导致在同一高度上有多个区块，这种情况会在区块链中形成一个“分叉（Fork）”。

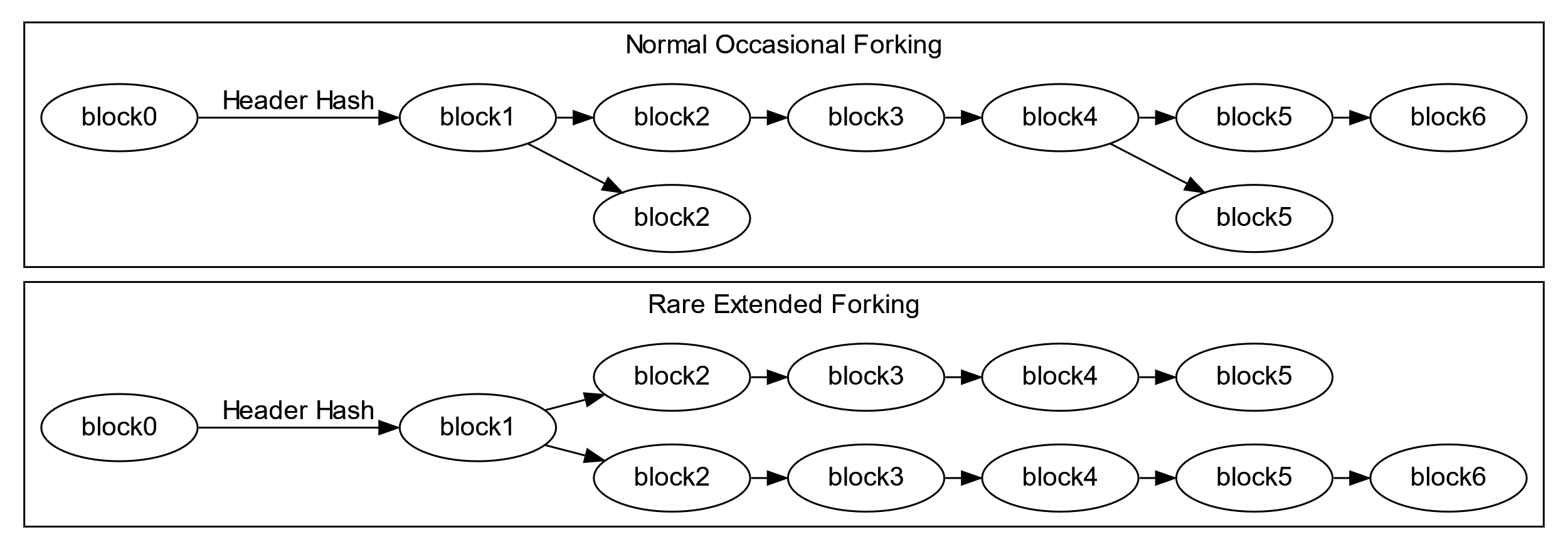


图 1-4 常见和不常见的区块链分叉

分叉通常是暂时的，随着时间的推移，网络中的节点总是遵循最难的链来继续添加新的区块。最终，较长的那条链会被视为有效链，丢弃属于较短分叉的陈旧区块（Stable Blocks）。这种机制保障了区块链的统一性和安全性。

由于在区块链分叉期间多个区块可以具有相同的高度，因此不将区块高度用作全局唯一标识符，通常通过其区块头的哈希值来引用。

iiii. 交易数据

每个区块必须包含一个或多个交易。这些交易中的第一笔被称为“清算交易（Audit-Mission，ATM）”。这个交易将对所有对该任务的交易进行验证并结算。

ATM 交易的UTXO具有特殊条件，在至少2个区块内，节点无法使用这个UTXO作为输入进行支出。这个限制的主要目的是在于防止节点在新区块确认之前就进行花费。在区块链分叉的情况下，某个区块可能被视为陈旧区块，即最终没有被纳入主链。此时，如果节点已经花费了ATM交易的UTXO，而这个交易实际上是无效的（因为它来自一个被丢弃的区块），那么节点就会面临损失。通过实施这个时间限制，网络能够确保节点在进行支出时更具谨慎性，从而减少因区块链分叉导致的潜在损失。

区块不需包含非ATM交易，但节点几乎总是包含额外的交易。所有交易，包括ATM交易，都以二进制原始交易格式编码到区块中。

对原始交易格式进行哈希处理以创建交易标识符（txid）。根据这些txid，通过将每个txid与另一个txid配对，然后将两个txid一起进行哈希运算，生成一个新的哈希值。如果txid的数量是奇数，则最后一个没有配对的txid会与它自己进行哈希运算。通过这种配对和哈希的方式，txid会逐层减少，直至最后形成一个根哈希（merkle root，默克尔根），它代表了整棵默克尔树的所有交易。

例如，如果交易只是联接（而不是哈希），则5笔交易的默克尔树将类似以下文本图：

      ABCDEEEE ........ Merkle root

      /       \

   ABCD        EEEE

  /    \      /

 AB    CD    EE ....... E is paired with itself

/  \  /  \  /

A  B  C  D  E ......... Transactions

默克尔树允许节点通过默克尔根与中间哈希列表来验证交易是否包含在区块中。中间哈希列表不需要被完全信任：伪造区块头的成本很高，而且中间哈希值不能伪造，否则验证将失败。

例如，要验证交易D是否已添加到区块，节点除了默克尔根之外，只需要C、AB和EEEE哈希的副本；节点不需要了解任何其它交易。

(2) 交易

交易使得节点花费概念币。每笔交易都由几个部分构成，这些部分既支持简单的直接支付，也支持复杂的交易。

i. 介绍

为简单起见，假装 ATM 交易不存在。

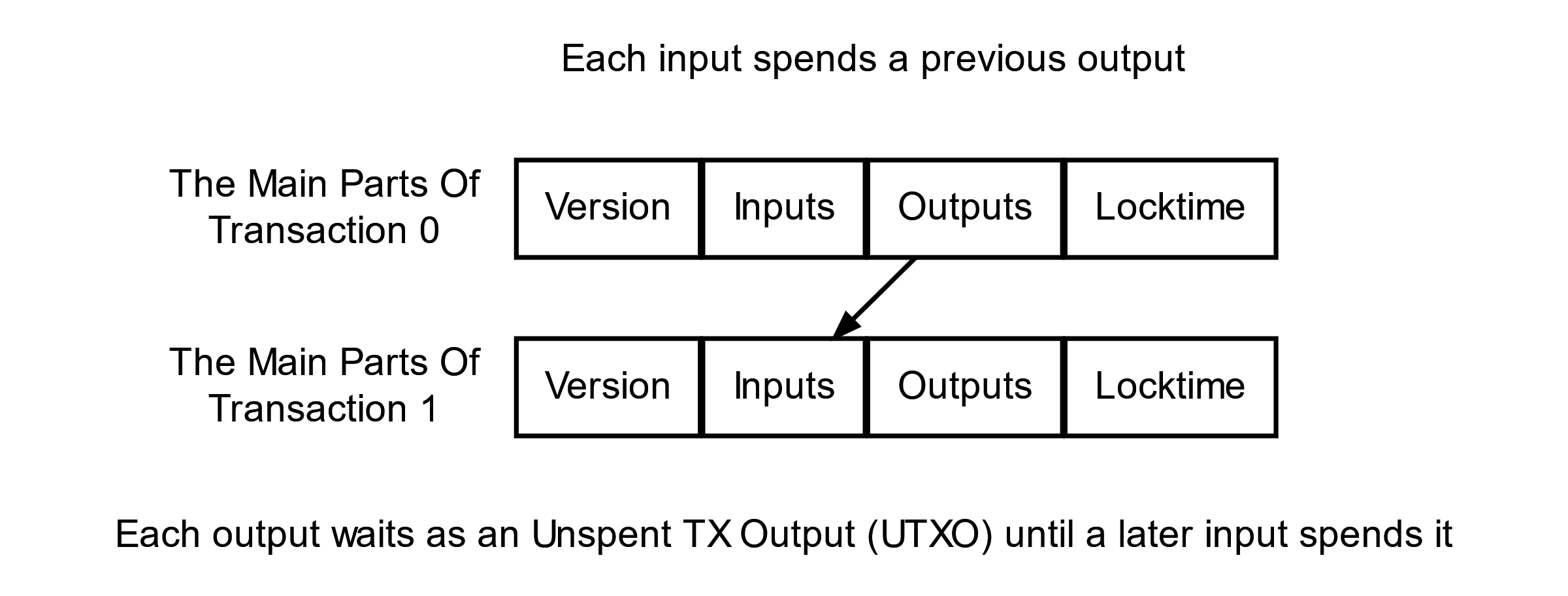


图 1-5 交易的组成部分

上图显示了交易的主要部分。每个交易至少有一个输入和一个输出。每个输入都会花费支付给前一个输出的概念币。然后每个输出都作为未花费的交易输出（UTXO）等待，直到后来的输入花费它。

每笔交易都以一个4字节的交易版本号为前缀，该版本号告诉节点使用哪组规则来验证。这允许开发人员为未来的交易创建新规则，而不会使以前的交易无效。

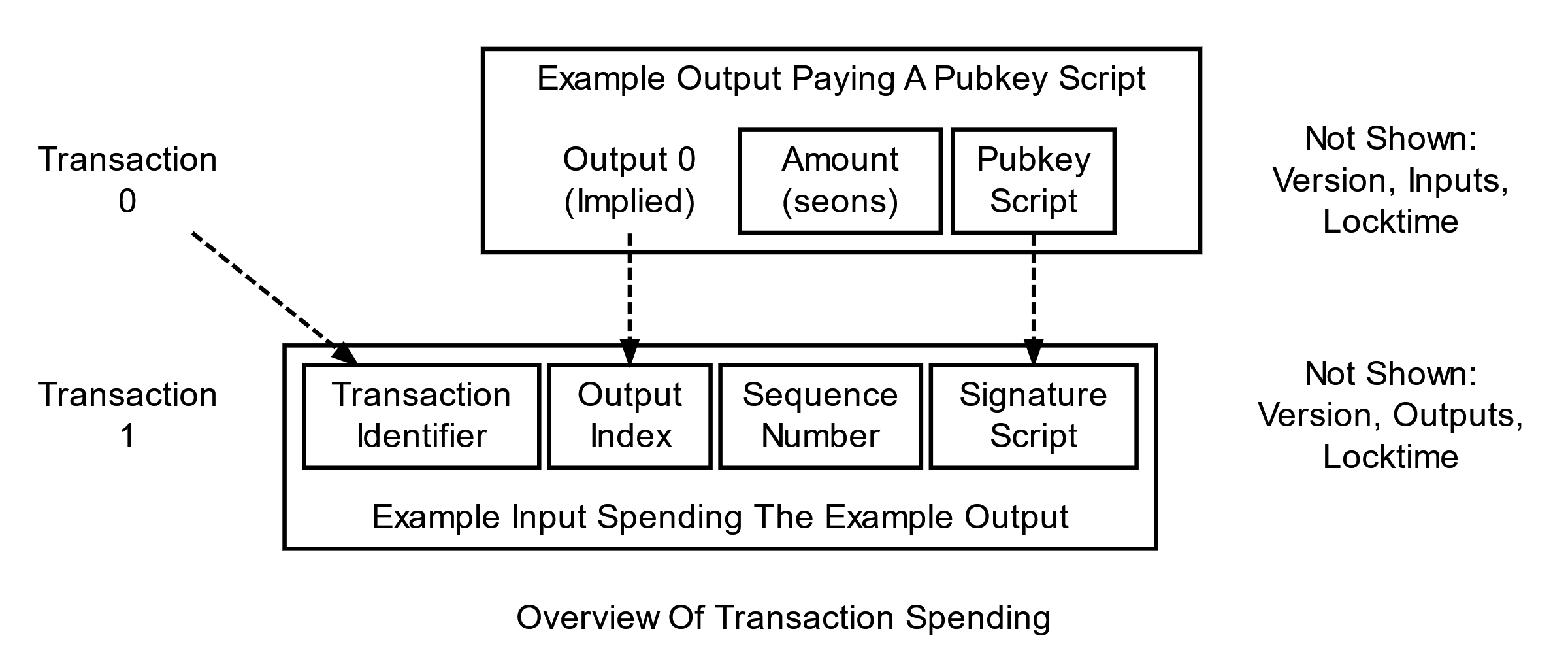


图 1-6 花费输出

每个交易输出都有一个隐含的索引号，这个索引号是基于其在交易中的位置来确定。输出中还有一个概念币，它将其支付给一个条件公钥脚本（scriptPubKey），任何能够满足该公钥脚本中条件的节点都可以花费这个输出，支取不能超过输出。

输出主要由交易标识符（txid）和输出索引号（output index numner，vout）来标识要花费的特定输出。它还有一个签名脚本，提供了满足条件公钥脚本中要求的参数。

下图通过显示节点A向B发送交易的工作流以及B稍后花费该交易的工作流，帮助说明如何使用这些功能。P2SH（Pay To Script Hash）让A创建一个包含的第二脚本（redeem script，赎回脚本）哈希的公钥脚本。

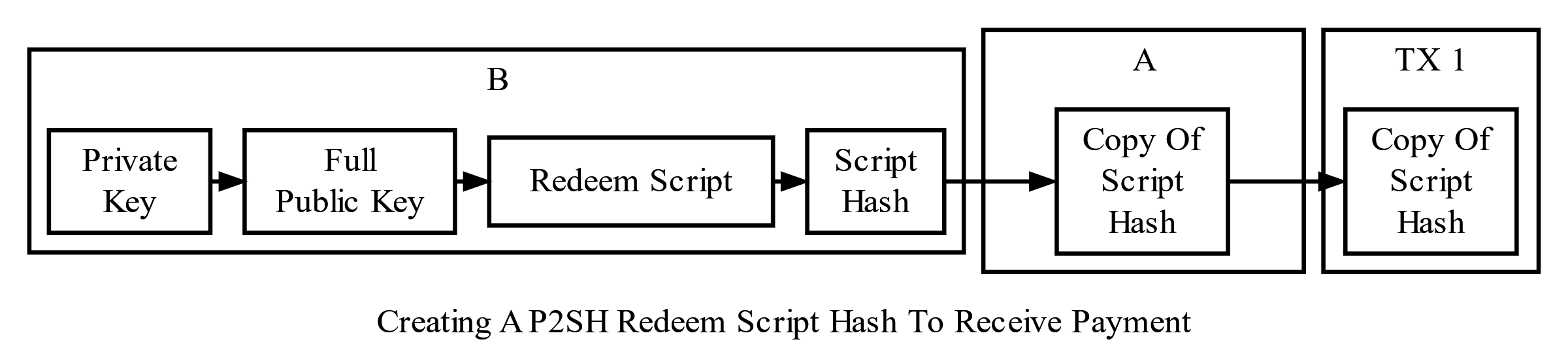


图 1-7 创建P2SH兑换脚本和哈希

B创建一个包含任意逻辑的赎回脚本，这个脚本定义了在什么条件下可以花费概念币；然后B对创建的赎回脚本进行哈希处理，产生一个唯一脚本哈希值；B将赎回脚本的哈希值提供给A，A使用B提供的赎回脚本哈希创建一个P2SH输出。

当B想要花费输出时，它的签名脚本中提供了它的签名以及完整的（序列化的）赎回脚本。点对点网络（Peer-To-Peer Network）确保完整的赎回脚本哈希值与A在它的输出中放入的脚本哈希值相同；然后，它完全按照主公钥脚本一样处理赎回脚本，如果赎回脚本不返回false，则让B花费输出。

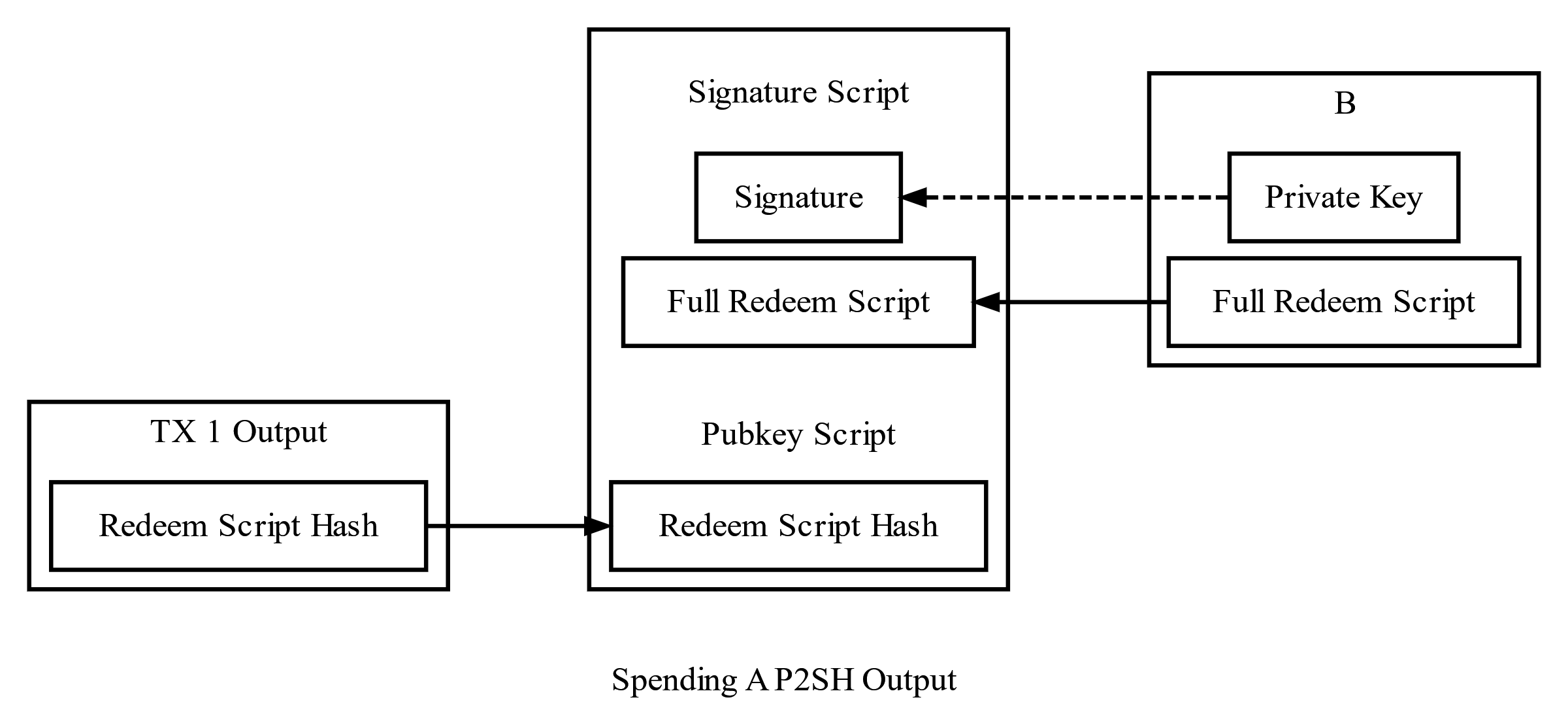


图 1-8 解锁P2SH输出已进行支出

ii. Pat To Script Hash

P2SH 用于将交易发送给脚本哈希。每个标准公钥脚本都可以用作P2SH兑换脚本，不包括 P2SH本身。P2SH最常见的用途是标准的多重签名公钥脚本。

另一个用于P2SH的常见赎回脚本是在区块链上存储文本数据。使用P2SH在区块链上存储文本的示例。

Pubkey script: OP\_HASH160 <Hash160(redeemScript)> OP\_EQUAL

Signature script: <sig> [sig] [sig...] <redeemScript>

(3) 账户

i. 介绍

账户创建公钥来接收概念币，并使用相应的私钥来花费这些概念币。

ii. 账户程序

允许接收和消费概念币是账户程序的唯一基本功能 — 但特定的账户程序不需要同时做这两件事。两个账户程序可以一起工作，一个程序分发公钥以接收概念币，另一个程序签署花费这些概念币的交易。

账户程序还需要与点对点网络交互，以从区块链获取信息并广播新交易。但是，分发公钥或签署交易的程序本身不需要与点对点网络交互。

账户系统的三个必要但可分离的部分：公钥分发程序、签名程序和网络程序。

iii. 全方位服务账户

最简单的账户是一个执行所有功能的程序：它生成私钥，派生相应的公钥，根据需要帮助分发这些公钥，监控这些公钥的输出，创建并签署使用这些输出的交易，并广播签名的交易。

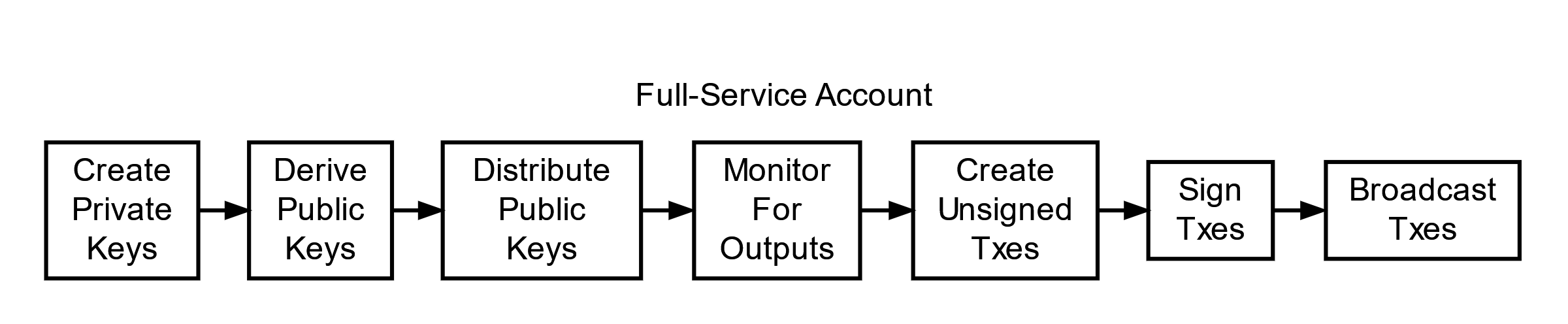


图 1-9 全方位服务账户

全方位服务账户的主要优点是易于使用。单个程序可以完成接收和消费概念币所需的一切。

全方位服务账户的主要缺点是它们将私钥存储在连接到Internet的设备上。此类设备遭到入侵是很常见的，互联网连接可以很容易地将私钥从被入侵的设备传输给攻击者。

iiii. 仅签名账户

为了提高安全性，私钥可以由在更安全的环境中运行的独立账户程序生成和存储。这些仅签名钱包与点对点网络交互的联网钱包一起工作。

仅签名账户程序通常使用确定性密钥创建来创建父私钥和公钥，这些私钥和公钥可以创建子私钥和公钥。

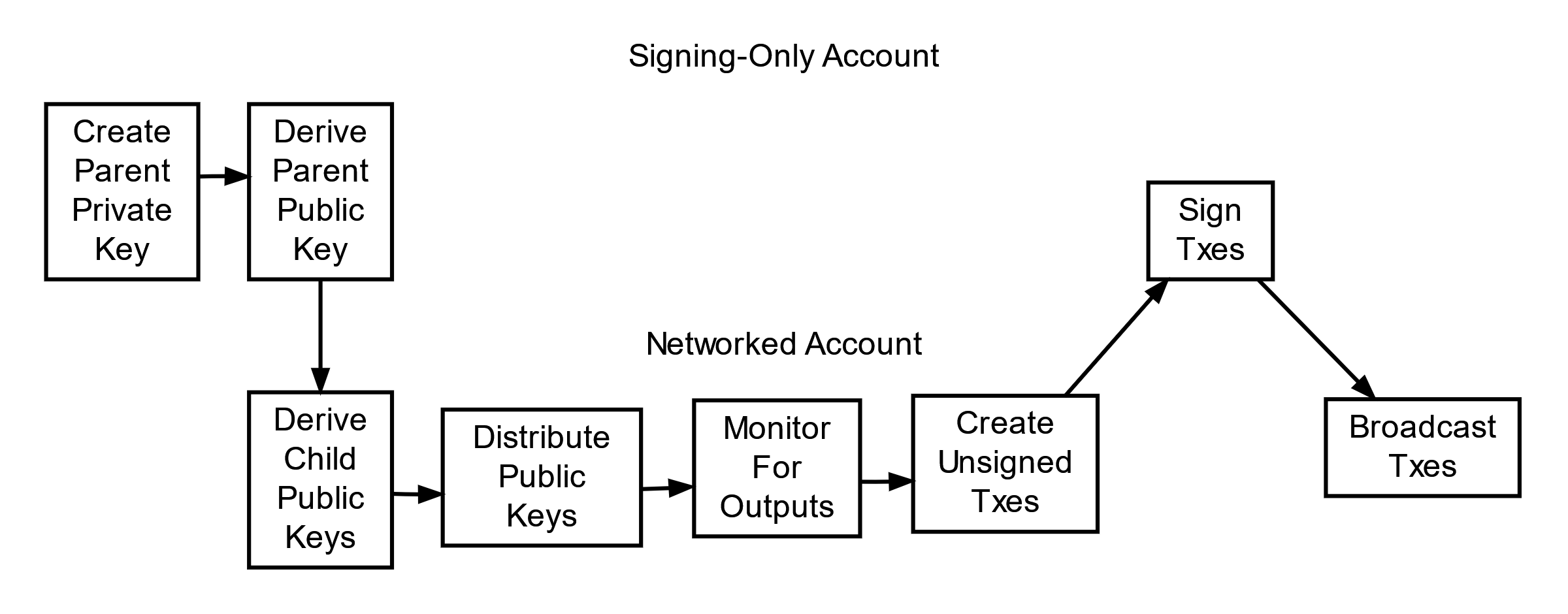


图 1-10 仅签名账户

首次运行时，仅签名账户会创建一个父私钥，并将相应的父公钥转移至联网账户；联网账户使用父公钥派生子公钥，可以选择帮助分发子公钥，监控花费这些公钥的输出，创建花费这些输出的未签名交易，并将未签名的交易转移至仅签名账户；在可选的审查步骤之后，仅签名账户使用父私钥来派生适当的子私钥并签署交易，将签名的交易交还给联网账户；然后，联网账户将签名的交易广播到点对点网络。

iiiii. 仅限分发的账户

在难以保证安全的环境（例如网络服务器）中运行的账户程序可以设计为分发公钥（包括P2SH地址）。设计这种极简账户有两种常见的方法：

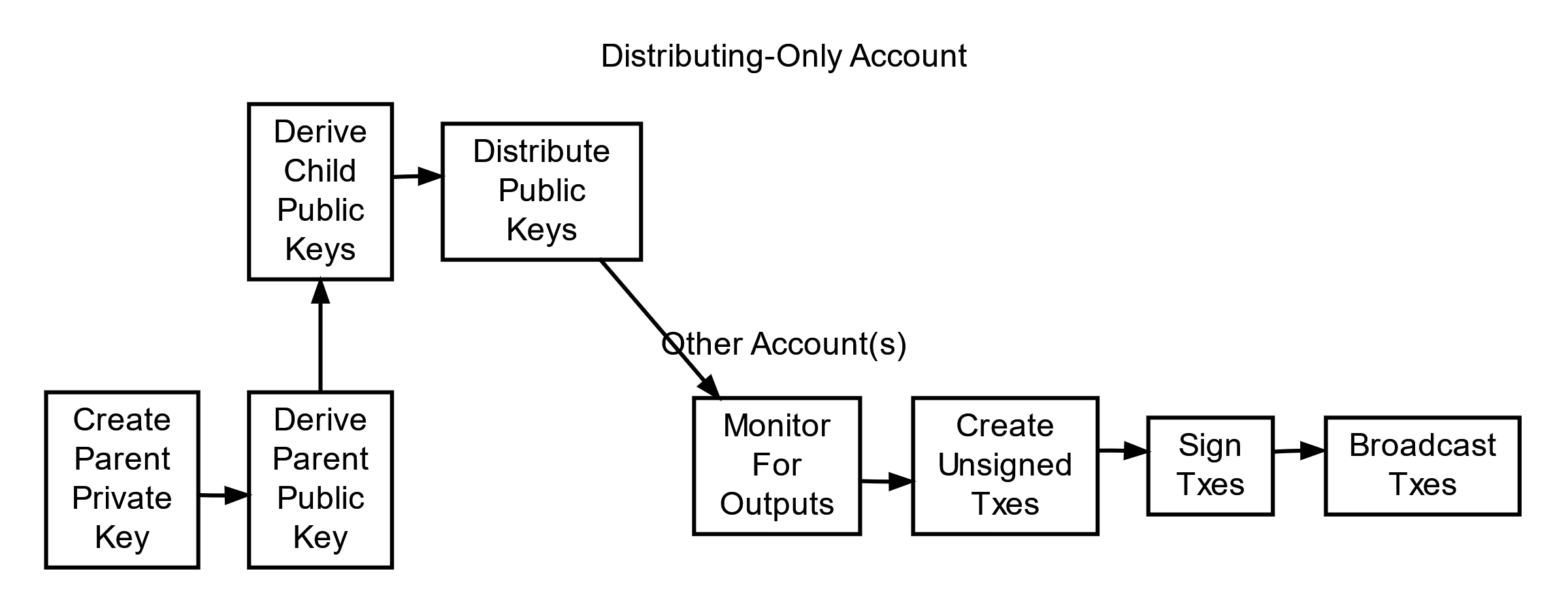


图 1-11 仅限分发的账户

a. 使用大量公钥或地址预填充数据库，然后根据请求使用其中一个数据库条目分发公钥脚本或地址。为避免密钥重用，Web服务器应跟踪已使用的密钥，并且永远不会用完公钥。

b. 使用父密钥创建子公钥。为避免密钥重用，必须使用一种方法来确保相同的公钥不会被分发两次。这可以是每个分发的建的数据库条目，也可以是指向键索引号的递增指针。

这两种方法都不会增加大量开销，尤其是在使用数据库将每笔传入的付款与单独的公钥相关联以进行付款跟踪时。

iiiiii. 账户文件

账户的核心是私钥的集合，可以以数字方式存储在文件中，也可以物理存储在纸上。

iiiiiii. 私钥格式

私钥用于从特定地址解锁概念币。这里定义，标准格式的私钥只是一个256位二进制数字，介于以下值之间：0x01 和 0xffff ffff ffff ffff ffff ffff ffff ffff ffff ffff ffff ffff ffff ffff ffff ffff。

iiiiiiii. 账户导入格式（Account Import Format，AIF）

为了使私钥的复制不易出错，定义账户导入格式。AIF在私钥上使用base58Check 编码，大大降低了复制错误的可能性。

a. 获取私钥

b. 未其前面添加一个0x80字节用于主网地址，或添加0xef用于测试地址。

c. 如果它应该与压缩的 public keys 一起使用，在其后附加一个 0x01 字节。如果与未压缩的公钥一起使用，则不附加任何内容。

d. 对扩展密钥执行 SHA256 哈希处理

e. 对 SHA256 哈希的结果执行 SHA256 哈希处理

f. 取第二个 SHA256 哈希的前四个字节，作为校验和

g. 将 e. 的四个校验和字节添加到 b. 的扩展密钥末尾。

h. 使用 Base58Check 编码将结果从字节字符串转化为 Base58 字符串。

该过程很容易逆转，只需使用 Base58 解码并去除填充。

iiiiiiiii. 迷你私钥格式

迷你私钥格式是一种将私钥编码为30个字符以内的方法，使私钥能够嵌入到较小的物理空间中，例如耐损失的二维码。

a. 迷你私钥的第一个字符是 'S'。

b. 为了确定迷你私钥的格式是否正确，将向私钥添加一个问好。

c. 计算 SHA256 哈希值。如果生成的第一个字节是 '00'，则它的格式很好。此密钥限制充当拼写错误检查机制。暴力使用随机数强制该过程，直到生成格式正确的迷你私钥。

d. 为了派生完整的私钥，只需获取原始迷你私钥的单个 SHA256 哈希值。这个过程是单向的，从派生密钥计算迷你私钥格式是很困难的。

iiiiiiiiii. 公钥格式

用 ECDSA公钥 表示 secp256k1 中定义的特定椭圆曲线（EC）上的一个点。在传统的未压缩行驶中，公钥包含一个标识字节、一个32字节的 X坐标 和一个32字节的 Y坐标。下面的极其简化的插图显示了使用椭圆曲线上的这样一个点 $y^2 = x^3 + 7$，位于连续数字字段上。

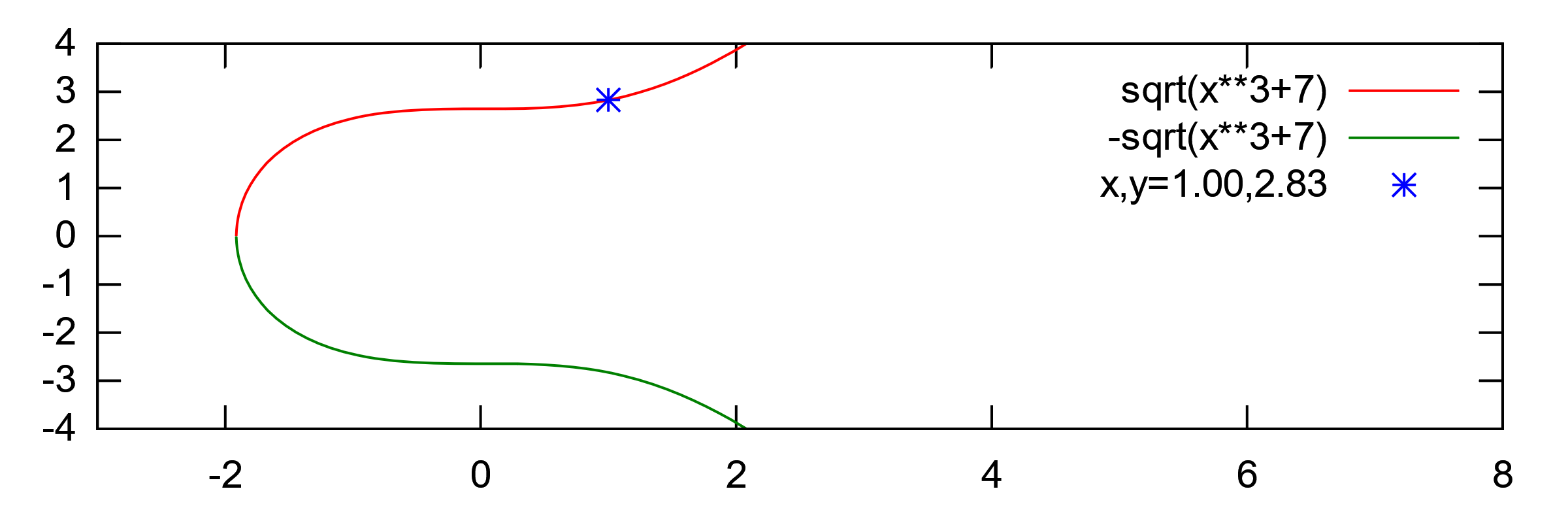


图 1-12 ECDSA 曲线上的点

Secp256k1 实际上通过一个大素数对坐标进行模运算，这会产生一个非连续整数数字字段和一个明显不太清晰的绘图。

通过删除 Y坐标，可以在不改变任何基本原理的情况下实现近 50% 的公钥大小减少。曲线上只有两个点共享任何特定的X坐标，因此可以将 32 字节的 Y坐标 替换为一个位，指示该点是位于图中显示位“顶部”还是“底部”的位置。

创建这些压缩的公有密钥不会丢失任何数据，只需少量的计算即可重建Y坐标并访问未压缩的公有密钥。

未压缩的公钥以 0x04 开头；压缩的公钥以 0x03 或 0x02 开头，具体取决于它们是大于还是小于曲线的中点。

(4) P2P网络

i. 介绍

全节点（Full Nodes）会下载区块链中的每一个区块和交易，并对其进行验证；在验证后，全节点会将这些有效的区块和交易转发给其它节点。档案节点（Archival Nodes）是全节点的一种，除了具备全节点的所有功能外，还存有整个区块链的历史记录；档案节点可以向其它节点提供历史区块的信息，允许其它节点查询过去的交易记录和区块信息。修剪节点（Pruned Nodes）也是全节点，但与档案节点不同的是，它们不存储整个区块链；修剪节点会定期删除旧的区块，以减少存储空间的需求，尽管如此，修剪节点仍能够验证交易和区块。

ii. 对等发现

首次启动，程序不知道任何活动全节点的IP地址。为了发现一些IP地址，它们查询一个或多个硬编码的DNS名称（称为DNS种子）。对查找的响应应该包括一个或多个DNS A记录，这些记录中包含了可以接受新连接的全节点的IP地址。

一旦程序连接到网络，其对等体就可以开始向它发送消息，其中包含网络上其它对等体的IP地址和端口号，从而提供一种完全去中心化的对等体发现方法。

但是，对等体可能离开网络或更改IP地址，因此程序可能需要在启动时进行几次不同的连接尝试，然后才能成功建立连接。

iii. 连接到对等体

通过向远程节点发送“version”消息来完成连接到对等点，该消息包含版本号、块和当前时间。远程节点同样使用“version”消息进行响应。然后两个节点都向另一个节点发送“verack”消息，以指示已建立连接。

连接后，向远程节点发送“addr”消息以收集其他对等节点。

为了保持与对等节点的连接，默认情况下，在与对等节点的连接有10分钟没有任何活动（即没有消息发送或接收），节点会向对方发送一条消息。如果20分钟内没有收到来自对等节点的任何消息，则假定对方不再活跃。

iiii. 初始区块下载（Initial Block Download，IBD）

在一个全节点能够有效验证未确认交易和最近新添的区块之前，它必须下载并验证从区块1（硬编码创世区块之后的区块）到区块链最佳状态（即链的最新区块）的所有区块。

事实上，也可以在需要下载大量区块时进行，例如当一个节点上时间离线后重新上线时。在这种情况下，这个节点会使用IBD方法下载自上次在线以来产生的所有区块。

默认情况下，节点会在本地最佳区块链的最后一个区块头时间超过30分钟时，启动IBD。

iiiii. 头部优先（Headers-First）

称之为Headers-first的初始区块下载方法。目标是下载最佳的区块头链。在下载区块头时，尽可能地进行部分验证；一旦确定了最佳区块头链，就可以并行地下载相应地区块。

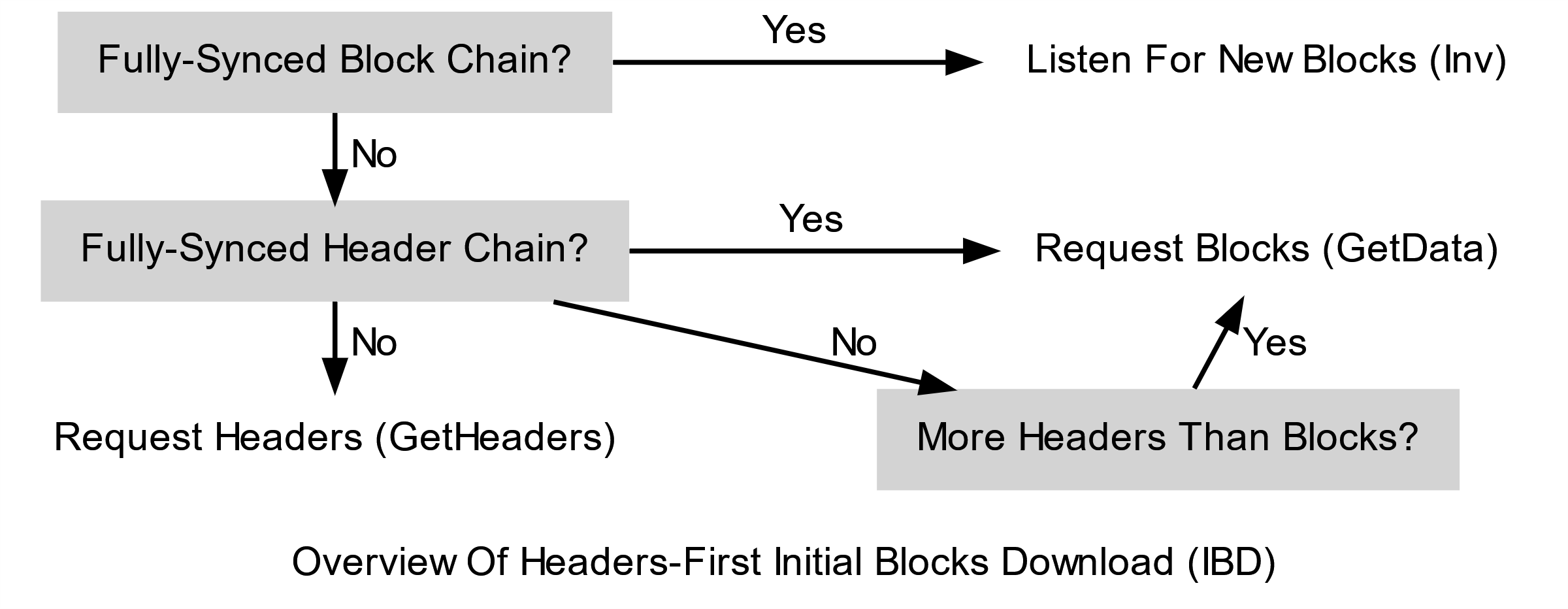


图 1-13 Headers-First 方法概述

节点第一次启动时，它的本地最佳区块链中只有一个区块 —— 硬编码的创世区块（区块0）.此节点选择一个远程对等节点，将其称为同步节点，并向其发送如下所示的“getheaders”消息。

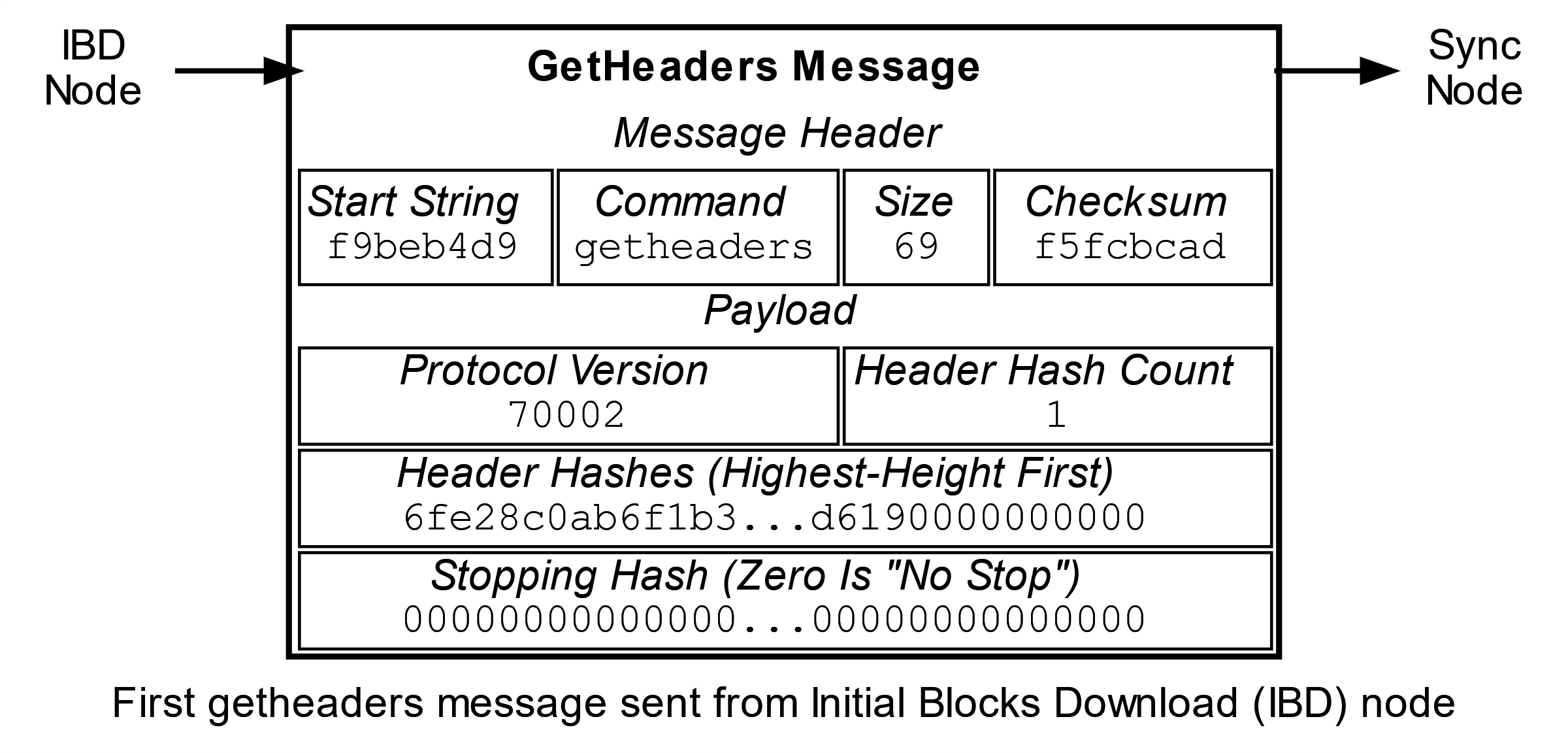


图 1-14 第一个GetHeaders消息

在“getheaders”消息的头哈希（header hashes）字段中，这个新节点发送它唯一拥有的区块的头哈希，即创世区块。它还将停止哈希（stopping hash）字段设置为全零，以请求最大大小的响应。

收到“getheaders”消息后，同步节点获取第一个（也是唯一一个）头哈希值，并在本地最佳区块链中搜索具有该标头哈希值的区块。它发现区块0匹配，因此它从区块1开始回复2,000个区块头（最大响应）。它在下图所示的“headers”消息中发送这些库存。

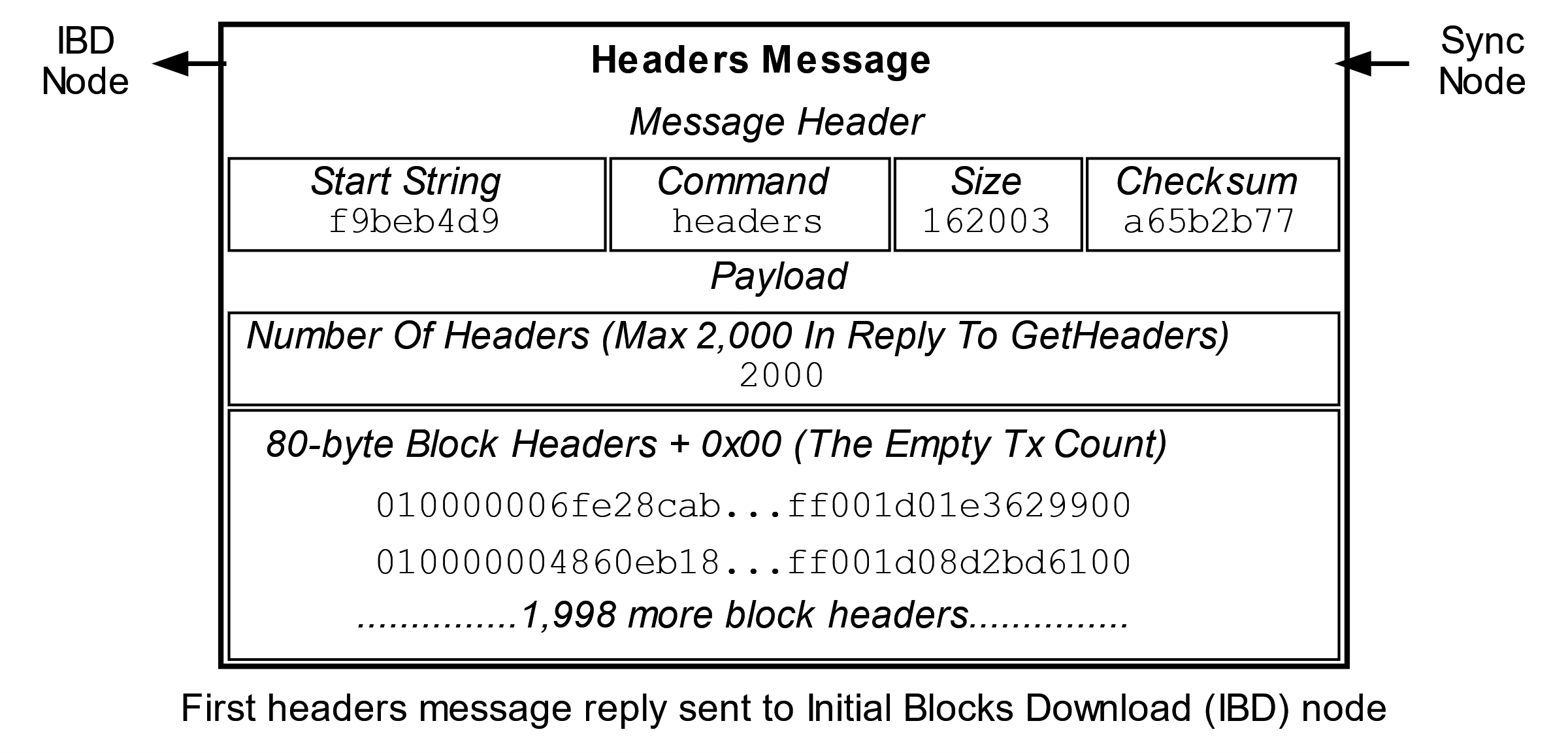


图 1-15 第一条Headers消息

IBD节点可以通过确保所有字段都遵循共识规则来验证这些区块头（完全验证需要获取相应区块地所有交易）。

在IBD节点部分验证了区块头后，它可以并行执行两件事：

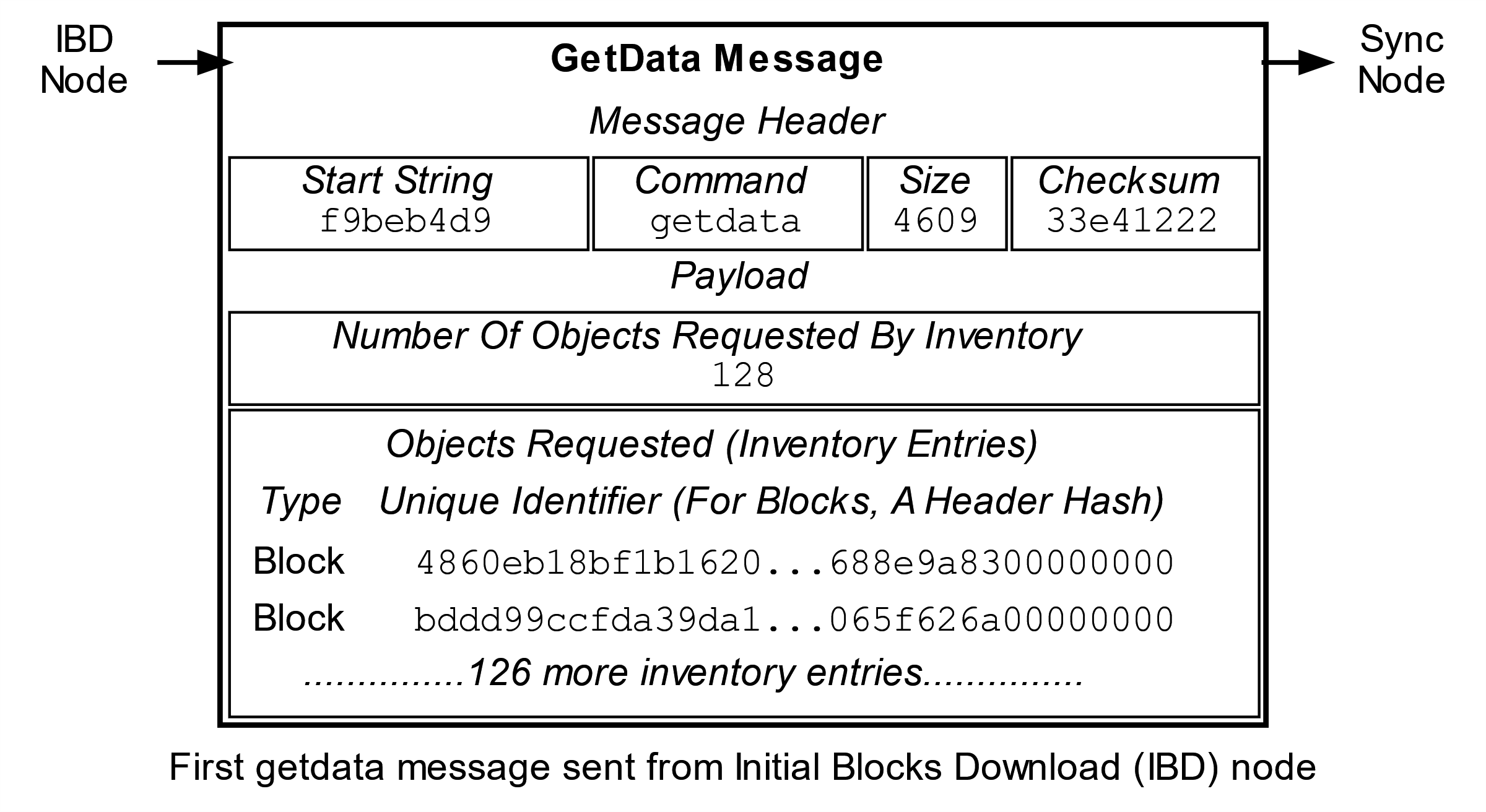


图 1-16 GetData消息

a. 下载更多标头： IBD节点可以向同步节点发送另一个“getheaders”消息，以请求最佳区块头链的下一批2,000个块头。可以立即验证这些块头，并重复请求，直到从同步节点收到块头少于2,000个的“headers”消息。

一旦IBD节点从同步节点收到少于2,000个区块头的“headers”消息，它就会向每个对等节点发送“getheaders”消息，以获取其最佳区块头链。通过比较响应，它可以轻松确定它下载的标头是否属于其对等体报告的最佳区块头链。这意味着即使不使用检查点，也会很快发现不诚实的同步节点（只要IBD节点连接到至少一个诚实的对等节点）。

b. 下载区块：IBD节点在下载区块头的同时，也会在下载完成后开始请求和下载每一个区块。IBD节点会利用从区块头链中计算出的区块头哈希，来创建“getdata”消息，从任何全节点请求其区块。

为了在多个对等节点之间分摊负载，一次最多从单个节点请求16个区块。结合其最多8个连接，这意味着头部优先将在IBD期间同时请求最多128个区块。

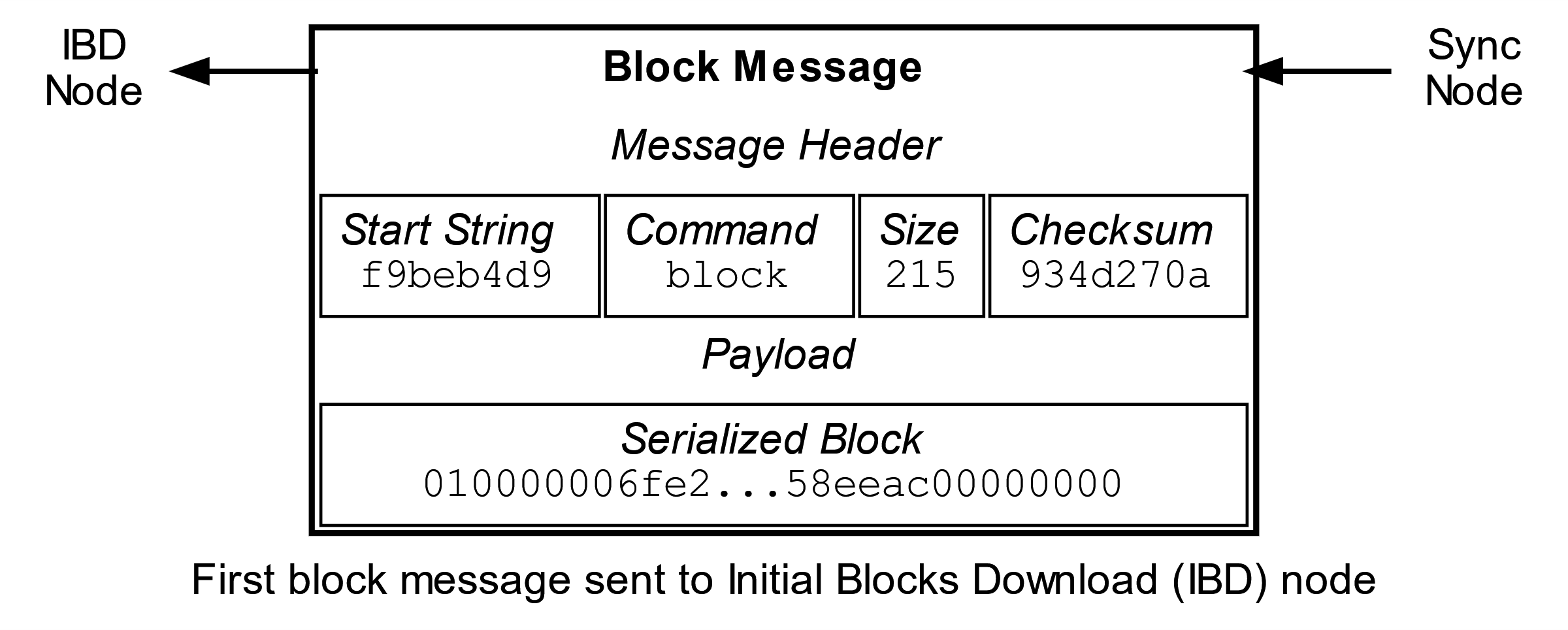


图 1-17 第一条Block消息

收到“getdata”消息后，节点会回复请求的每个数据块。每个数据块都采用序列化数据块格式，并以单独的“数据块”消息发送。发送的“block”消息（针对block 1）如上所示。

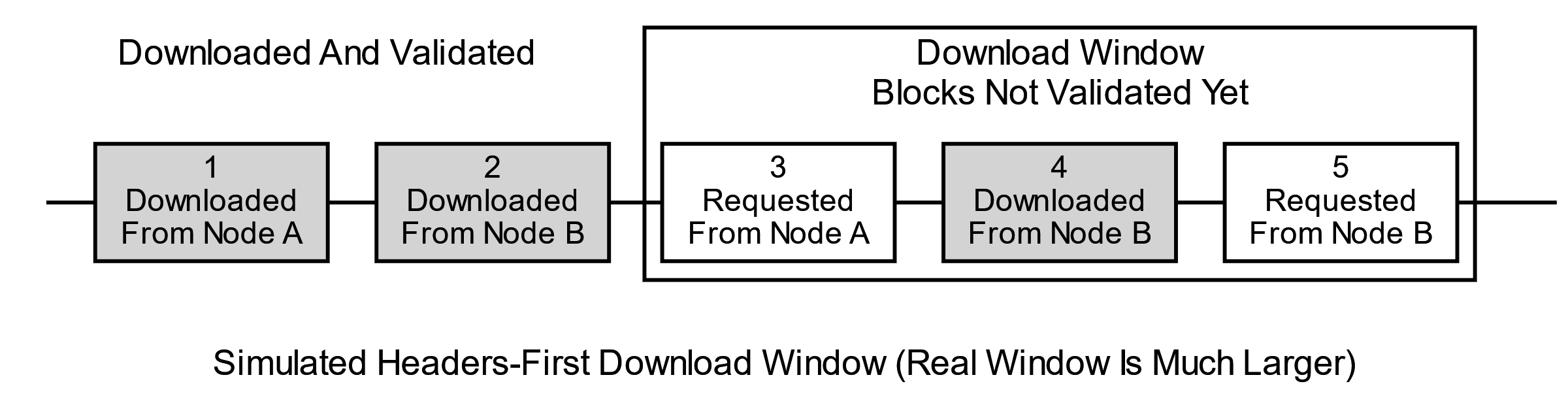


图 1-18 模拟区块头-第一个下载窗口

头部优先模式使用1,024个区块的移动下载窗口来最大限度地提高下载速度。窗口高度最低的块是下一个要验证的块；如果在准备号验证区块时，区块尚未到达，认为对方已停滞，则将至少再等待两秒，以便停滞节点发送区块。如果区块仍未到达，将断开与停滞节点的连接，并尝试连接到另一个节点。

一旦IBD节点同步到区块链的顶端，它将接收通过常规区块广播发送的区块。

iiiiii. 区块广播

当节点添加新区块时，它会使用以下方法之一将新区块广播到其他对等节点：

a. 未经请求的区块推送：节点向每个全节点对等体发送一条“block”消息。节点可以通过这种方式合理地绕过标准中继方法，因为它有理由相信其他对等体之前没有这个新发现地区块。

b. 标准区块中继：节点作为标准中继节点，向其每个对等节点发送一条“inv”消息，并带有针对新区块的库存列表，告知对等体有新块可用。对于希望获取新块的头部优先节点，它们会先回复一个“getheaders”消息，其中包括其最佳头链上最高高度的头哈希。这个消息通常还会包含一些更早的头部，以便进行分叉检测。在接收到这个请求后，头部优先节点会紧接着发送一个“getdata”消息，要求完整的新块。通过这种方式，头部优先节点可以拒绝孤儿区块。

默认情况下，节点使用直接头部公告向那些已经发送“sendheaders”信号的对等体广播新区块。对于那些没有发送“sendheaders”信号的对等体，则使用标准的区块中继方式。

iiiiiii. 孤儿区块

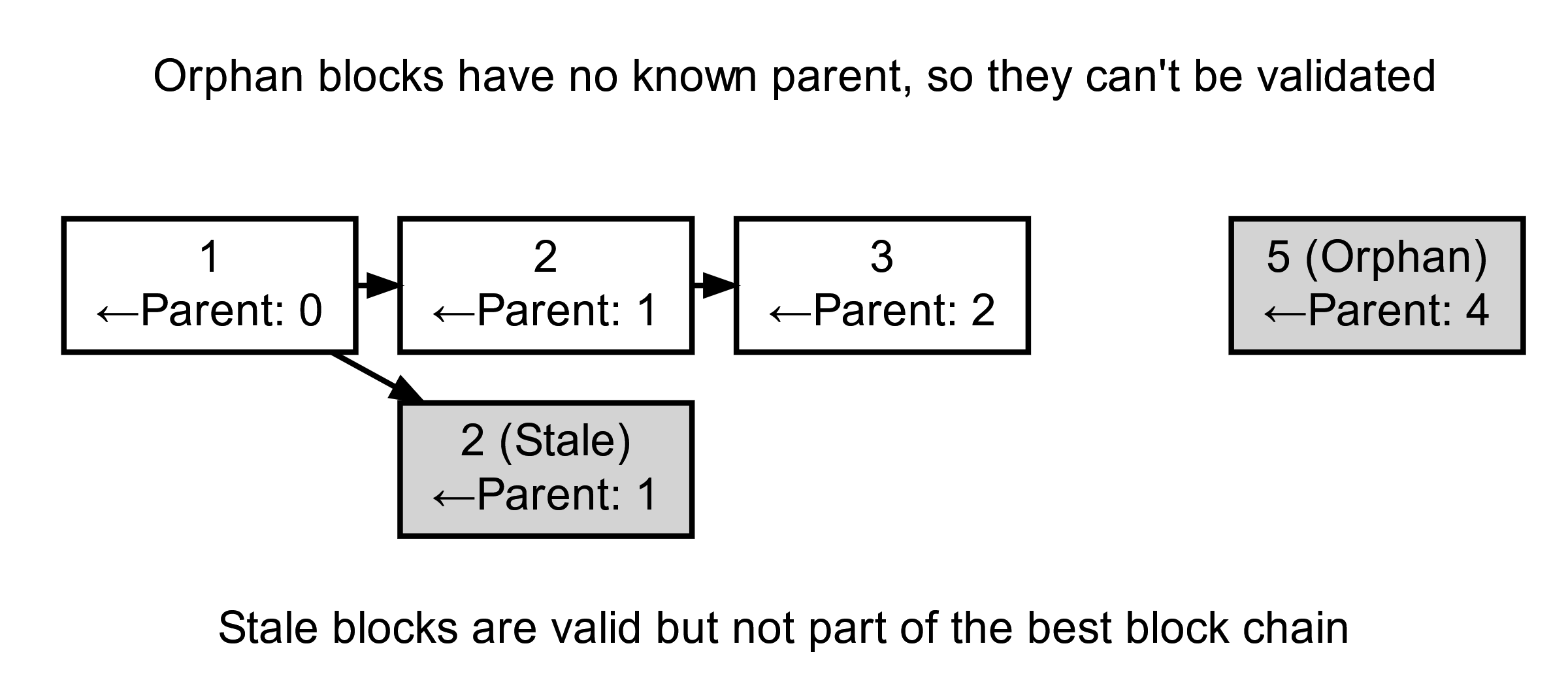


图 1-19 孤块与陈旧块之间的区别

头部优先节点在使用“getdata”消息请求区块之前，始终使用“getheaders”消息请求区块头。广播节点将发送一条“headers”消息，其中包含它认为下载节点需要到达最佳头链顶端的所有区块头（最多2,000个）；这些头部中的每一个都指向其父级，因此当下载节点收到“block”消息时，该区块不应该是孤儿区块—它的所有父级都是已知的（即使它们尚未被验证）。尽管如此，如果在“block”消息中收到的区块是孤儿区块，则头部优先节点将立即丢弃它。

iiiiiiii. 交易广播

为了将交易发送到对等节点，会发送“inv”消息。如果收到响应消息，则将具体的交易数据发送给接收节点。收到交易的节点在确认该交易是有效的之后，会以相同的方式向其他对等节点转发这个交易。

### 3.2.3. 详细设计

#### 3.2.3.1. 区块链

(1) 区块头

区块头以下面描述的72字节格式进行序列化，然后作为任务分配算法的一部分进行哈希处理，使序列化头格式成为共识规则的一部分。

表 2-1 区块头格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| 4 | version | int32\_t | 区块链协议或客户端的版本号，方便版本管理和网络节点的兼容性。 |
| 32 | previous block header hash | char[32] | 一个SHA256哈希值，此前一个区块的头部哈希，确保在不更改此块头的情况下，无法更改以前的块。 |
| 32 | merkle root hash | char[32] | 区块体的SHA256哈希值。默克尔根源自此区块包含的所有交易的哈希值，确保在不修改此块头的情况下无法修改这些交易。 |
| 4 | time | uint32\_t | 区块时间，表示该区块被创建的确切时间，以 Unix 时间戳形式记录。 |

哈希按内部字节顺序排列；其他值都按little-endian顺序排列。

示例：

01000000 ........................... Block version: 2

b6ff0b1b1680a2862a30ca44d346d9e8

910d334beb48ca0c0000000000000000 ... Hash of previous block's header

9d10aa52ee949386ca9385695f04ede2

70dda20810decd12bc9b048aaab31471 ... Merkle root

00000000 ........................... [Unix time][unix epoch time]: 0

(2) 块版本

考虑到未来可能会对该系统作修改/升级，引入该字段。

(3) 默克尔树

默克尔树是使用该区块中交易的所有TXID构建的，但首先根据共识规则的要求，将TXID按顺序放置

a. ATM交易的TXID始终放在第一位。

b. 此数据块中的任何输入都可以花费此数据块中的输出。但是，与输出对应的TXID必须放在输入对应的TXID之前的某个位置。

如果一个区块只有ATM交易，则ATM TXID用作默克尔根哈希。

如果一个区块只有一个ATM交易和 一个其他交易，则这两个交易的TXID按顺序拼接，连接为64个原始字节，然后 SHA256 哈希计算得到默克尔根。

如果一个区块有3个或更多交易，则形成中间默克尔树行。从ATM交易的TXID开始，TXID按顺序放置并配对。每对进行连接为64个原始字节，并进行 SHA256 哈希计算得到第二行哈希。如果TXID的数量为奇数，则将最后一个TXID与自身副本进行连接并进行哈希处理。如果第二行中有两个以上的哈希值，则重复该过程以创建第三行（如有必要，进一步重复以创建其他行）。一旦获得仅包含两个哈希的行，这些哈希将被连接并进行哈希处理以得到默克尔根。

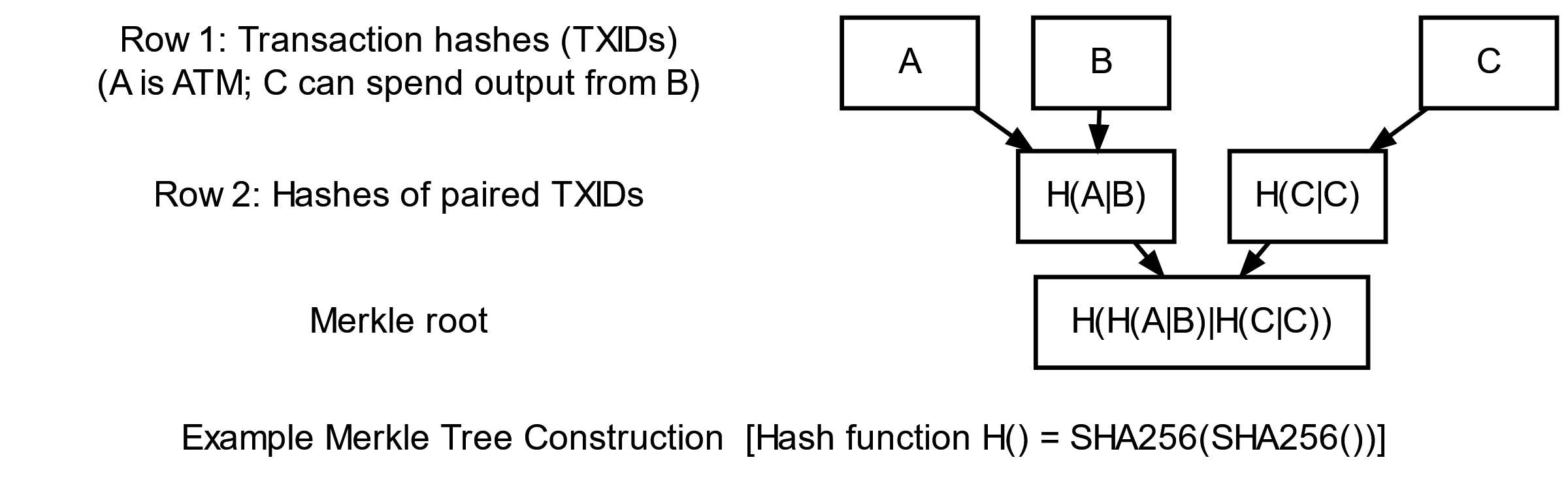


图 2-1 默克尔树结构示例

TXID和中间哈希在连接时始终按内部字节顺序排列，而生成的默克尔根在放入区块头时也按内部字节顺序排列。

(4) 序列化块

根据当前的共识规则，除非区块的序列化大小小于或等于1MB，否则该区块无效。下面描述的所有字段都计入序列化大小。

表 2-2 序列化块格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| 72 | block header | block\_header | 区块头部分中描述的格式。 |
| Varies | txn\_count | compactSize uint | 此区块中的交易总数，包括ATM交易。 |
| Varies | txns | raw transaction | 这个区块中的每笔交易，以原始交易格式进行依次排放。交易在数据流中的显示顺序必须与TXID在默克尔树的第一行中出现的顺序相同。 |

区块中的第一笔交易必须是ATM交易，该交易应对任务以及期间的活动记录进行结算。

#### 3.2.3.2. 交易

(1) Raw Transaction Format（原始交易格式）

交易以序列化字节格式（称为原始格式）在网络节点之间广播。这种形式的交易经过SHA256哈希处理以创建TXID，并最终创建包含该交易的区块的默克尔根，使交易格式成为共识规则的一部分。

表 2-3 原始交易格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| 4 | version | int32\_t | 交易版本号。 |
| Varies | tx\_in count | compactSize uint | 此交易中的输入数量。 |
| Varies | tx\_in | txIn | 交易输入。 |
| Varies | tx\_out count | compactSize uint | 此交易中的输出数。 |
| Varies | tx\_out | txOut | 交易输出。 |
| 4 | lock\_time | uint32\_t | Unix时间戳或区块号 |

一个交易可能有多个输入和输出，因此，txIn 和 txOut 结构可能会在交易中重复出现。CompactSize无符号整数是可变长度整数的一种形式。

(2) TxIn：交易输入（非ATM）

每个非ATM输入都会花费前一笔交易的outpoint。

表 2-4 TxIn格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| 36 | previous\_output | outpoint | 花费的前一个outpoint。 |
| Varies | script bytes | compactSize uint | 签名脚本中的字节数。最大为10,000字节。 |
| Varies | signature script | char[] | 满足 outpoint 的 公钥脚本中的条件的脚本语言脚本。 |
| 4 | sequence | uint32\_t | 序列号。默认都是0xffffffff。 |

(3) Outpoint：特定输出部分

由于单个交易可以包含多个输出，因此 outpoint 结构包括 TXID 和输出索引号，以引用特定输出。

表 2-5 Outpoint格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| 32 | hash | char[32] | 保存要花费的输出的交易的TXID。 |
| 4 | index | uint32\_t | 要从交易中花费的特定输出的输出索引号。第一个输出是0x00000000。 |

(4) TxOut：交易输出

每个输出花费一定数量的概念币，将它们置于任何能够满足提供的公钥脚本的节点的控制之下。

表 2-6 TxOut格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| 8 | value | int64\_t | 要花费的概念币的数量。可能为零。 |
| 1+ | pk\_script bytes | uint | 公钥脚本中的字节数。最大值为10,000字节。 |
| Varies | pk\_script | char[] | 定义花费此产出必须满足的条件 |

(5) Audit-Mission Input: 区块中第一笔交易的输入

区块中的第一笔交易必须只有一个输入，称为Audit-Mission（ATM）。ATM输入具有以下格式。

表 2-7 ATM格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| 32 | hash(null) | char[32] | 一个32字节的null，因为ATM没有前一个 outpoint。 |
| 4 | index(UINT32\_MAX) | uint32\_t | 0xffffffff，因为ATM没有前一个 outpoint。 |
| Varies | script bytes | compactSize uint | ATM脚本中的字节数。最大为100字节。 |
| Varies(4) | height | script | 区块高度。 |
| Varies | ATM script | None | ATM 字段：可以包含不超过100字节的数据，但要减去4个字节用于标识区块高度（height）。 |
| 4 | sequence | uint32\_t | 序列号。 |

#### 3.2.3.3. P2P网络

(1) 常量和默认值

表 2-8 网络配置

|  |  |  |
| --- | --- | --- |
| 网络 | 默认端口 | 起始字符串 |
| 主网 | 60000 | 0xf9beb4d9 |
| 测试网 | 69999 | 0x0b110907 |
| 本地网络 | 60001 ~ 69998 | 0xfabfb5da |

起始字符串是硬编码的常量，出现在网络上发送的所有消息的开头。

(2) 消息报头

网络协议中的所有消息都使用相同的格式，该格式提供所需的多字段消息标头和可选的有效负载。报头格式为：

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| 4 | start string | char[4] | 指示网络的特殊字节。 |
| 12 | command name | char[12] | ASCII字符串，用于标识有效负载中包含的消息类型。后跟nulls（0x00）以填充字节计数。 |
| 4 | payload size | uint32\_t | 负载（payload）中的字节数。当前允许的有效负载最大字节数（"MAX\_SIZE"）为32MB — 有效负载大小大于此大小的消息将被丢弃或拒绝。 |
| 4 | checksum | char[4] | SHA256哈希值的前4个字节（SHA256(payload)）按内部字节顺序排列。如果payload为空，则校验和始终为0x5df6e0e2（SHA256(<empty string>)）。 |

(3) 数据消息

以下网络消息都请求或提供与交易和区块相关的数据。

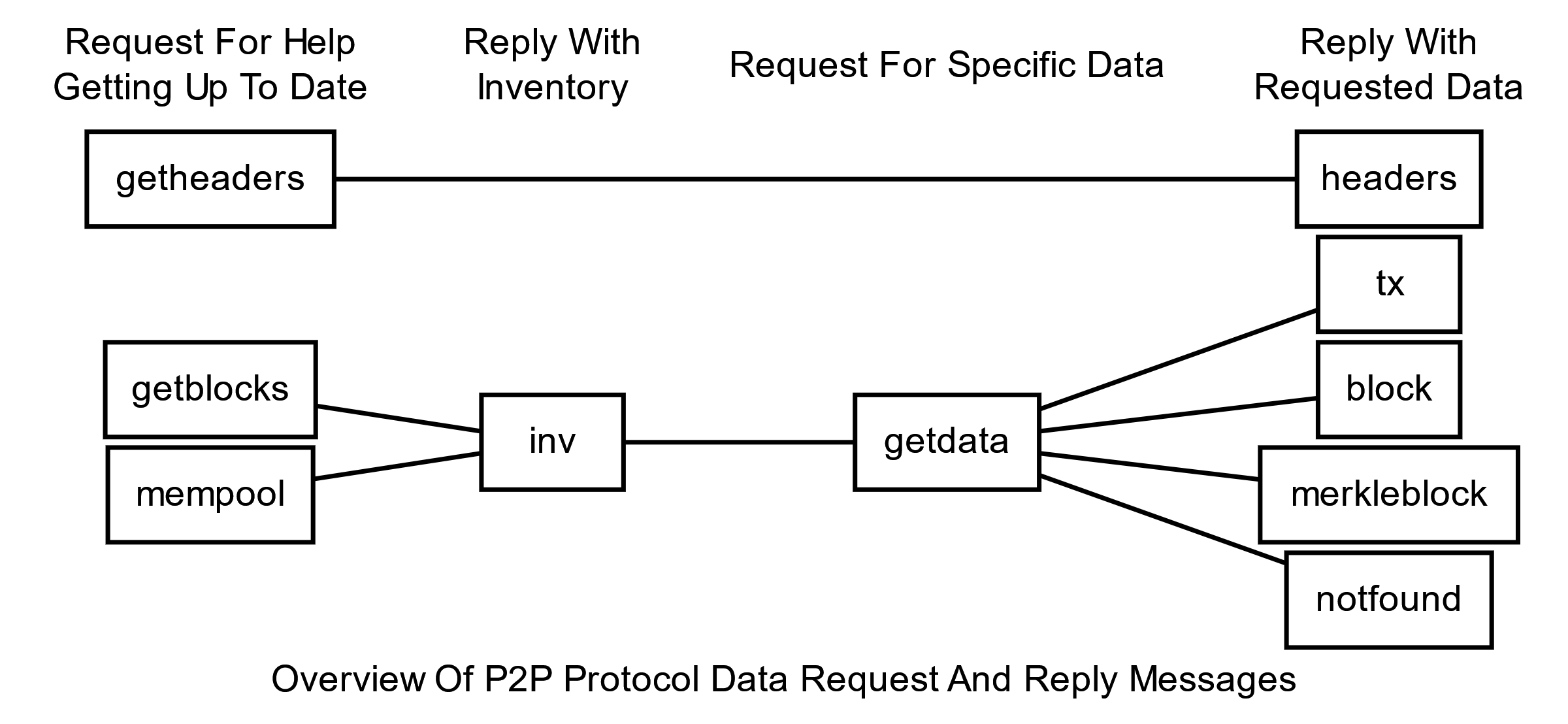


图 2-2 P2P协议数据请求和回复消息概述

许多数据消息使用库存作为交易和区块的唯一标识符。库存具有简单的36字节结构：

表 2-9 库存结构

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| 4 | type identifier | uint32\_t | 经过哈希处理的对象的类型。 |
| 32 | hash | char[32] | SHA256 对象的哈希值。 |

可用的类型标识符（type identifier）包括：

|  |  |  |
| --- | --- | --- |
| 类型标识符 | 名称 | 描述 |
| 1 | “MSG\_TX” | 哈希值是一个TXID。 |
| 2 | “MSG\_BLOCK” | 是区块头的哈希值。 |
| 3 | “MSG\_FILTERED\_BLOCK” | 区块头的哈希；与 "MSG\_BLOCK" 相同。用于 “getdata”消息，表示响应应该是“merkleblock”消息。仅用于“getdata”。 |
| 4 | “MSG\_CMPCT\_BLOCK” | 区块头的哈希值；与 MSG\_BLOCK 相同。用于“getdata”消息，表示响应应该是“cmpctblock”消息。仅用于“getdata”。 |
| 1+ | “MSG\_WITNESS\_TX” | 哈希值是一个TXID。用于“getdata”消息，表示响应应该是交易消息，如果 witness 结构非空，则将使用序列化。仅用于“getdata”。 |
| 2+ | “MSG\_WITNESS\_BLOCK” | 是区块头的哈希值；与 MSG\_BLOCK 相同。用于“getdata”消息，表示响应应该是一个区块消息，其中包含具有使用序列化witness的交易。 |

(4) Block消息

“block”消息以序列化块部分中描述的格式传输单个序列化块。

a. GetData响应：节点将始终发送该数据以响应请求库存类型为"MSG\_BLOCK"的区块的“getdata”消息（前提是节点具有可用于中继的区块）。

b. 不请自来：一些节点会发送未经请求的“block”消息，向所有对等节点广播新上链的区块。

(5) GetBlocks消息

“getblocks”消息请求一条“inv”消息，该消息提供从区块链中的特定点开始的区块头哈希值。它允许已断开连接或首次启动的对等节点获取请求它未见过的块所需的数据。

已断开连接的对等节点在其本地存储的区块链中可能有过时的区块，因此“getblocks”消息允许请求的节点向接收的节点提供多个区块头的哈希值，这些哈希值是在请求方本地链上不同高度的区块头。接收方会检查接收到的区块哈希列表，寻找它们之间最后一个共同的区块头哈希。一旦找到了这个共同的区块头哈希，接收方就会回复请求方，提供所有在此之后的区块头哈希。

如果接收方没有找到共同头哈希，它将假设最后一个共同区块是创世区块，接收方会发送一个“inv”消息，这个消息中包括从block 1开始的区块头哈希。

表 2-10 GetBlocks消息格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| 4 | version | uint32\_t | 协议版本号；与“version”消息中发送的相同。 |
| Varies | hash count | compactSize uint | 表示提供的区块头哈希的数量，不包括停止哈希。 |
| Varies | block header | char[32] | 一个或多个区块头哈希值（每个32字节），哈希值按区块高度的反顺序排列，因此最高高度的哈希值在前，最低高度的哈希值在最后。 |
| 32 | stop hash | char[32] | 最后一个请求的区块头哈希。如果设置为全零，则表示请求一个“inv”消息，其中包含所有后续的区块头哈希。作为回应，最多将发送500个哈希；如果需要超过500个哈希，必须用一个新的“getblocks”消息发送更高高度的头哈希作为区块头哈希字段的第一个条目。 |

以下显示一个“getblocks”消息。

71110100 ........................... Protocol version: 70001

02 ................................. Hash count: 2

d39f608a7775b537729884d4e6633bb2

105e55a16a14d31b0000000000000000 ... Hash #1

5c3e6403d40837110a2e8afb602b1c01

714bda7ce23bea0a0000000000000000 ... Hash #2

00000000000000000000000000000000

00000000000000000000000000000000 ... Stop hash

(6) GetData消息

“getdata”消息从另一个节点请求一个或多个数据对象。请求的数据对象通过库存来指定，请求节点通常是在之前通过“inv”消息接收这些库存。

对“getdata”消息的响应可以是“tx”、“block”、“merkleblock”、“cmpctblock”或“notfound”。

此消息不能用于请求任意数据。例如无法请求那些已经不在内存池或中继集合中的历史数据。完整节点可能会因为已经从其区块数据库中修建掉老旧的交易而无法提供更早的区块。因此，“getdata”消息通常只向那些通过“inv”消息表明自己拥有所请求数据的节点发起请求。

“getdata”消息的格式与“inv”消息相同；只有消息报头不同。

(7) GetHeaders消息

“getheaders”消息请求一条“headers”消息，该消息提供从区块链中的特定点开始的区块头。它允许已断开连接或首次启动的对等节点获取它尚未看到的区块头。

“getheaders”消息与“getblocks”消息几乎相同，但有一个细微的区别：“getblocks”消息的回复包含不超过500个区块头哈希值；“getheaders”消息的回复包含多达2,000个区块头。

(8) Headers消息

“headers”消息将区块头发送到一个节点，该节点之前使用“getheaders”消息请求某些区块头。headers消息可以为空。

表 2-11 Headers消息格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| Varies | count | compactSize uint | 区块头数量，最大为2,000。 |
| Varies | headers | block\_header | 区块头：每个72字节的区块头都采用区块头部分中描述的格式，并带有额外的0x00后缀。此0x00称为交易计数，但由于“headers”消息不包含任何交易，因此交易计数始终为零。 |

以下显示一个“headers”消息。

01 ................................. Header count: 1

02000000 ........................... Block version: 2

b6ff0b1b1680a2862a30ca44d346d9e8

910d334beb48ca0c0000000000000000 ... Hash of previous block's header

9d10aa52ee949386ca9385695f04ede2

70dda20810decd12bc9b048aaab31471 ... Merkle root

00000000 ........................... [Unix time][unix epoch time]: 0

00 ................................. Transaction count (0x00)

(9) Inv消息

“inv”消息用于传输一个或多个对象的库存。发送方可以主动发送“inv”消息，而不需要接收方的请求。这种方式通常用于广播新的交易或区块。此外，“inv”消息也可以作为其他消息的回复发送，例如：“getblocks”消息或“mempool”消息。

接收方可以将“inv”消息中的库存与他已知的库存进行比较，然后使用后续消息请求尚未见过的对象。

表 2-12 Inv消息格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| Varies | count | compactSize uint | 库存条目的数量。 |
| Varies | inventory | inventory | 一个或多个库存条目。 |

以下显示带有两个库存条目的“inv”消息。

02 ................................. Count: 2

01000000 ........................... Type: MSG\_TX

de55ffd709ac1f5dc509a0925d0b1fc4

42ca034f224732e429081da1b621f55a ... Hash (TXID)

01000000 ........................... Type: MSG\_TX

91d36d997037e08018262978766f24b8

a055aaf1d872e94ae85e9817b2c68dc7 ... Hash (TXID)

(10) Mempool消息

“mempool”消息请求接收节点已验证为有效但尚未出现在区块中的交易的TXID。对“mempool”消息的响应是一个或多个“inv”消息，其中包含所请求的TXID，并按照常规的库存格式进行发送。

当程序（例如一个全节点）首次连接到区块链网络时非常有用。全节点发送“mempool”消息可以迅速获取网络上大部分或全部未确认的交易。

(11) Notfound消息

“notfound”消息是对“getdata”消息的回复。当请求特定对象时，如果接收节点没有这些对象可供转发，它会发送“notfound”作为回复。

“notfound”消息的格式与“inv”消息相同；只是消息报头不同。

(12) Addr消息

（IP地址）消息中继网络上对等体的连接消息。每个想要接收传入连接的对等节点都会创建一个“addr”或“addr2”消息，以提供其连接信息。然后主动地将消息发送给其它对等节点。一些节点将该消息发送给它们已连接的对等节点，这些节点进一步分发，从而允许去中心化的网络节点发现。

也可以发送“addr”消息以响应“getaddr”消息。

表 2-13 Addr消息格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| Varies | IP address count | compactSize uint | IP地址条目数，最大为1,000。 |
| Varies | IP addresses | network IP address | IP地址条目。 |

每个封装的network IP address使用以下结构：

表 2-14 IP地址结构

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| 4 | time | uint32\_t | Unix时间戳。广播自己IP地址的节点会将其设置为当前时间；已连接IP地址的节点会将其设置为最后一次连接的时间；仅转发IP地址的节点不进行修改。 |
| 8 | services | uint64\_t | 节点在其“version”消息中公布的服务。 |
| 16 | IP address | char[16] | big-endian排序的IPv6地址。IPv4地址作为Ipv4映射的IPv6地址（::ffff:<IPv4地址>）。 |
| 2 | port | uint16\_t | big-endian排序的端口号。 |

以下显示“addr”消息的一部分。

fde803 ............................. Address count: 1000

d91f4854 ........................... [Epoch time][unix epoch time]: 1414012889

0100000000000000 ................... Service bits: 01 ([network][network] node)

00000000000000000000ffffac000001 ... IP Address: ::ffff:172.0.0.1

ea60 ............................... Port: 60000

[...] .............................. (999 more addresses omitted)

(13) GetAddr消息

“getaddr”消息从接收节点请求“addr”消息。传输节点可以使用这些地址快速更新其可用节点数据库。

(14) Ping消息

“ping”消息有助于确定接收对等节点仍处于连接状态。

表 2-15 Ping消息格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| 8 | nonce | uint64\_t | 分配给此“ping”消息的随机数。响应的“pong”消息将包含此随机数，以识别它正在回复。 |

(15) Pong消息

“pong”消息回复“ping”消息，向ping节点证明该节点任然处于活动状态。

为了允许节点跟踪延迟，“pong”消息会发回“ping”消息中的随机数。

“pong”消息的格式与“ping”消息相同。只有消息报头不同。

(16) SendHeaders消息

“sendheaders”消息告诉接收节点使用“headers”消息而不是“inv”消息发送新的区块广播。

(17) VerAck消息

“verack”消息确认先前收到的“version”消息，通知连接节点可以开始发送其他消息。

(18) Verson消息

“version”消息在连接开始时向接收节点提供有关传输节点的消息。在两个对等节点交换“version”消息之前，不会接收其他消息。

如果接收了“version”消息，则接收节点应发送“verack”消息。

表 2-16 Version消息格式

|  |  |  |  |
| --- | --- | --- | --- |
| 字节 | 字段 | 数据类型 | 描述 |
| 4 | version | int32\_t | 传输节点理解的最高协议版本。 |
| 8 | services | uint64\_t | 位域（bit-fields）编码的传输节点支持的服务。 |
| 8 | timestamp | uint64\_t | 传输节点当前的时间戳。节点会拒绝时间戳超过两小时的区块；可以帮助其他节点确认时钟是否正确。 |
| 8 | addr\_recv services | uint64\_t | 位域（bit-fields）编码的接收节点支持的服务。 |
| 16 | addr\_recv IP address | char[16] | big-endian排序的接收节点的IPv6地址。IPv4地址作为Ipv4映射的IPv6地址（::ffff:<IPv4地址>）。 |
| 2 | addr\_recv port | uint16\_t | big-endian排序的接收节点的端口号。 |
| 8 | addr\_trans services | uint64\_t | 位域（bit-fields）编码的发送节点支持的服务。 |
| 16 | addr\_trans IP address | char[16] | big-endian排序的发送节点的IPv6地址。IPv4地址作为Ipv4映射的IPv6地址（::ffff:<IPv4地址>）。  2 | addr\_trans port | uint16\_t | 必需 | big-endian排序的发送节点的端口号。 |
| 2 | addr\_trans port | uint16\_t | big-endian排序的发送节点的端口号。 |
| 8 | nonce | uint64\_t | 检测自身连接。如果为0，则忽略；如果是其他任何内容，则节点应在收到“version”消息时发现nonce与发送的nonce相同，应该终止连接。 |
| 4 | start\_height | int32\_t | 发送节点的最佳区块链的高度。 |

已分配以下服务标识符。

表 2-17 服务标识符

|  |  |  |
| --- | --- | --- |
| 值 | 标识符 | 描述 |
| 0x00 | Unnamed | 此节点不是完整节点。除了发起交易之外，无法提供任何数据。 |
| 0x01 | NODE\_NETWORK | 完整节点，可以请求完整的区块。 |
| 0x02 | NODE\_GETUTXO | 完整节点，可以响应“getutxo”请求。 |
| 0x04 | NODE\_BLOOM | 完整节点，能够并愿意并行处理布隆过滤请求的。 |
| 0x08 | NODE\_WITNESS | 完整节点，可以请求区块和交易。 |
| 0x10 | NODE\_XTHIN | 完整节点，允许在不下载完整区块的情况下，快速接收和验证区块数据。 |
| 0x20 | NODE\_NETWORK\_LIMITED | 与NODE\_NETWORK相同，但至少保存最近288个区块。 |

以下显示一个“version”消息。

72110100 ........................... Protocol version: 70002

0100000000000000 ................... Services: NODE\_NETWORK

bc8f5e5400000000 ................... [Epoch time][unix epoch time]: 1415483324

0100000000000000 ................... Receiving node's services

00000000000000000000ffffc61b6409 ... Receiving node's IPv6 address

208d ............................... Receiving node's port number

0100000000000000 ................... Transmitting node's services

00000000000000000000ffffcb0071c0 ... Transmitting node's IPv6 address

208d ............................... Transmitting node's port number

128035cbc97953f8 ................... Nonce

cf050500 ........................... Start height: 329167

### 3.2.4. 编码

### 3.2.5. 测试

# 4. 实践收获与体会

# 5. 致谢

参考文献

“大数据项目综合实训”成绩评定表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 班级 |  | 姓名 |  | 学号 |  |
| 实训单位 | |  | | | |
| 成 绩 评 定 | | | | | |
| 实  训  评  语 | （从实训态度、实训期间的表现、出勤、实训中完成的学习任务、完成的实践项目质量、实训报告质量、实训答辩质量等方面进行评价）  指导教师签字：    年 月 日 | | | | |
| 成  绩 |  | | | | |
| 备  注 |  | | | | |