# 并行 EVM ：深入理解 Web3 性能提升新叙事



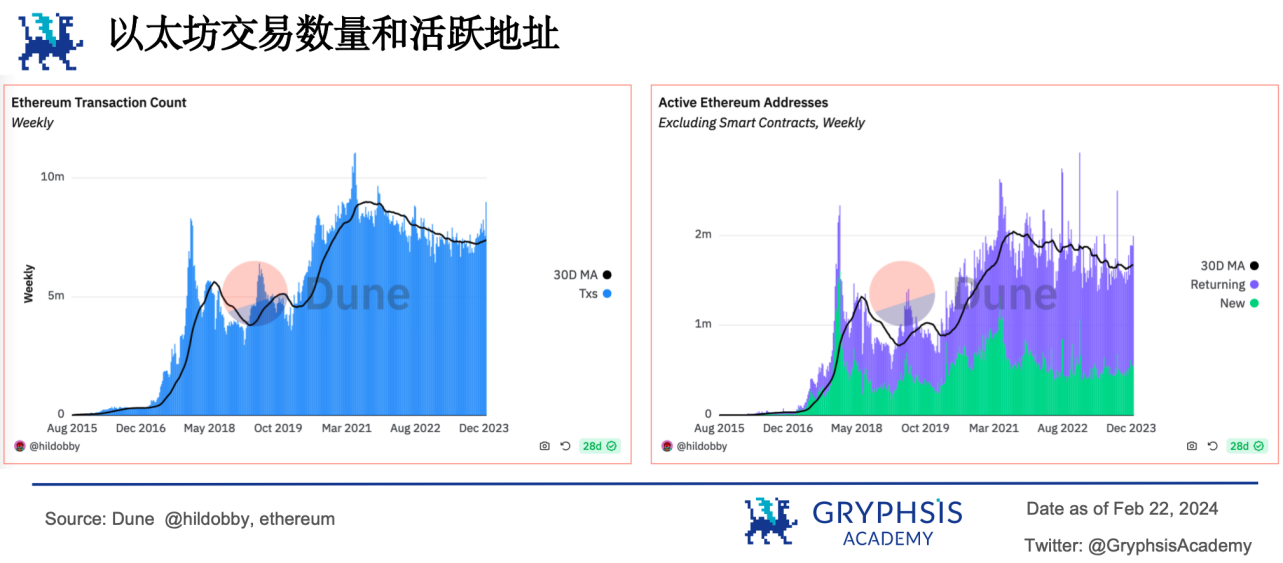
## TL;DR

1. 并行 EVM 是链上交易量发展到一定程度后出现的一种新叙事。并行 EVM 主要分为单体区块链和模块化区块链。单体区块链又分为 L1 和 L2 。并行 L1 公链分为两大阵营： EVM 和非 EVM 。并行 EVM 叙事处于发展的早期阶段，还有很大的成长空间；
2. 拆解并行 EVM 的技术实现路径，主要包含虚拟机和并行执行机制两大方面。在区块链的语境下，虚拟机是指对分布式状态机进行虚拟的进程虚拟机，用于执行合约；
3. 并行执行是指发挥多核处理器的优势，尽可能在同一时间同时执行多个交易，而保证最终状态与串行执行时结果一致；
4. 并行执行机制分为消息传递、共享内存、和严格状态访问列表三大类。共享内存又分为内存锁模型和乐观并行化。无论哪种机制均提高了技术的复杂性；
5. 并行 EVM 叙事既有行业增长的内在驱动因素，又需要从业者高度关注其可能存在的安全问题；
6. 并行 EVM 各标的项目均以不同的方式提供了并行执行思路，既有共性又有自己的独特性；
7. 并行 EVM 叙事一定会诞生好的项目和不错的投资机会，值得关注。

## 1. 行业综述

### 1.1 历史沿革

区块链网络为个人和企业进行交易创造了一种新的、去中心化的信任基础。以比特币为代表的第一代区块链网络以分布式记账的方式开创了去中心化的电子货币交易新模式，革命性地开创了一个新时代。随后，以以太坊为代表的第二代区块链网络则充分发挥想象力，提出以分布式状态机的方式实现去中心化应用（ dApp ）。从那时起，区块链网络开启了它自己十几年飞速发展的历史，从 Web3 基础设施到以 DeFi 、 NFT 、社交网络和 GameFi 等为代表的各种赛道，诞生了无数或技术或商业模式的创新。行业的蓬勃发展需要不断吸引新用户参与去中心化应用的生态建设，吸引新用户反过来又对产品体验提出了更高的要求。而 Web3 应用作为一种创新的产品形态，不但要在满足用户需要上有所创新（功能性需求），还要考虑怎样在安全性和性能之间取得平衡（非功能性需求）。自诞生以来，人们提出了各种各样的解决方案试图解决性能问题。这些解决方案大致可以分为两类：一类是链上扩容方案，如分片（ sharding ）和有向无环图（ DAG ）；一类是链下扩容方案，如 Plasma 、闪电网络、侧链和 Rollups 等。但这还远远跟不上链上交易快速增长的速度。

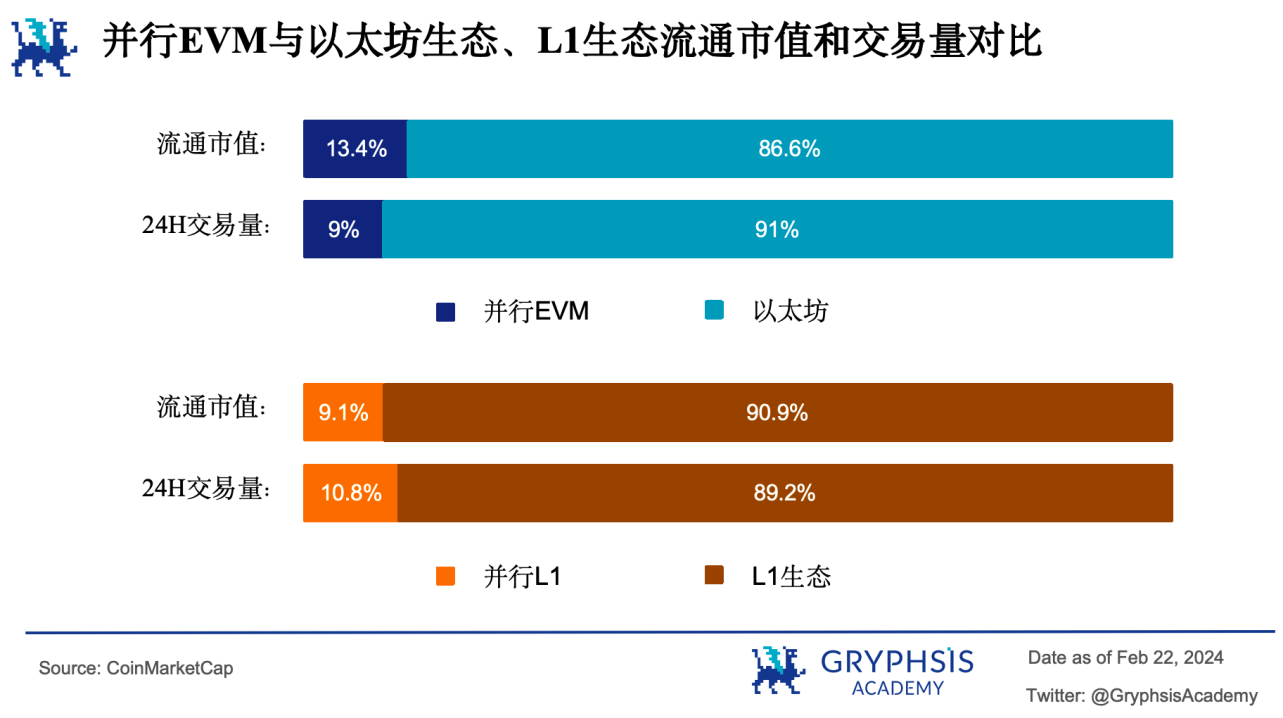


尤其是经历了2020年 DeFi Summer 以及2023年年末比特币生态中铭文的持续爆发，业界迫切需要新的性能提升方案来满足“高性能、低费率”的要求。并行 EVM 的新叙事就是在这样的背景下诞生的。

### 1.2 市场规模



以太坊对交易的处理是串行的，执行效率并不高，串行计算变并行计算将带来性能的巨大提升，这一点所有L1项目的技术团队都看到了，并分为两大阵营展开竞争。**非 EVM 阵营**的竞争者如 Solana 、 Aptos 和 Sui ，创造了自己的生态与以太坊竞争，代币流通市值分别达到450亿、33亿和19亿美元。**EVM 阵营**的项目则致力于赋能 EVM 的并行执行能力。 Sei 在其 v2 版本升级提案中高调宣称将成为“第一个并行 EVM 区块链”，预后还有更大的发展。与此同时当下营销热度第一的并行 EVM 新公链 Monad 潜力也不可小视。两大阵营的项目都进行了不同轮次的融资。资本的青睐一定程度上证明了赛道的价值。相比之下 L2 项目还在早期阶段，除 Neon 做到了6900万美元流通市值外，其他项目还缺少相关数据。

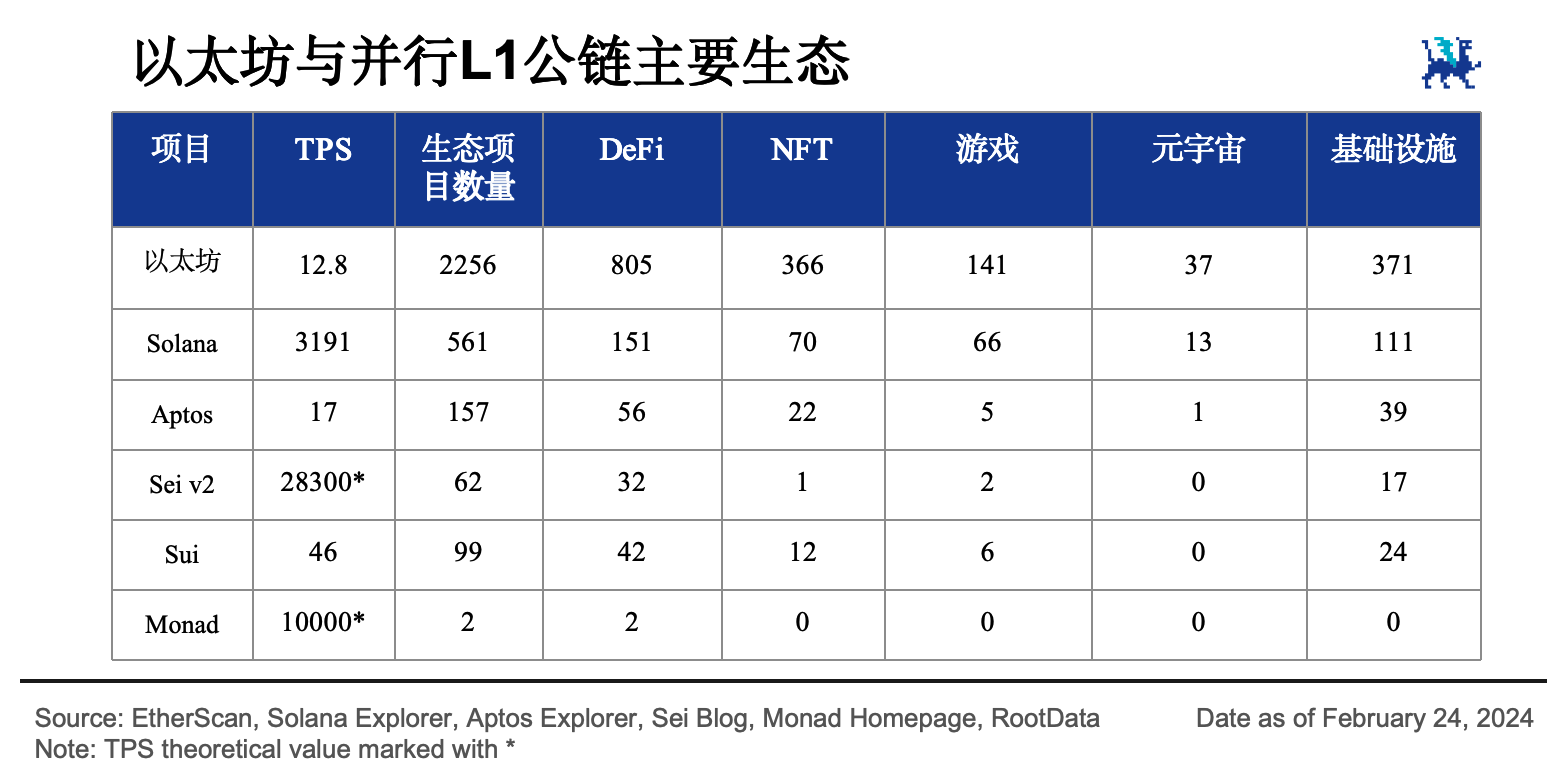


总的来看，并行 EVM 主要项目目前代币流通市值约为523.69亿美元，24小时交易量约为31.029亿美元。其中 L1 项目占比最大，其代币流通市值为523亿美元，24小时交易量为31亿美元。而当前 EVM 生态总市值约为3900亿美元，24小时交易量约为342亿美元。 L1 生态代币流通市值为5730亿美元，24小时交易量为288亿美元。无论是总体上与以太坊生态进行比较，还是单独观察 L1 项目与整个 L1 生态，并行 EVM 叙事都有很大的成长空间。

### 1.3 行业图谱

业界一般将区块链网络分为4层结构：

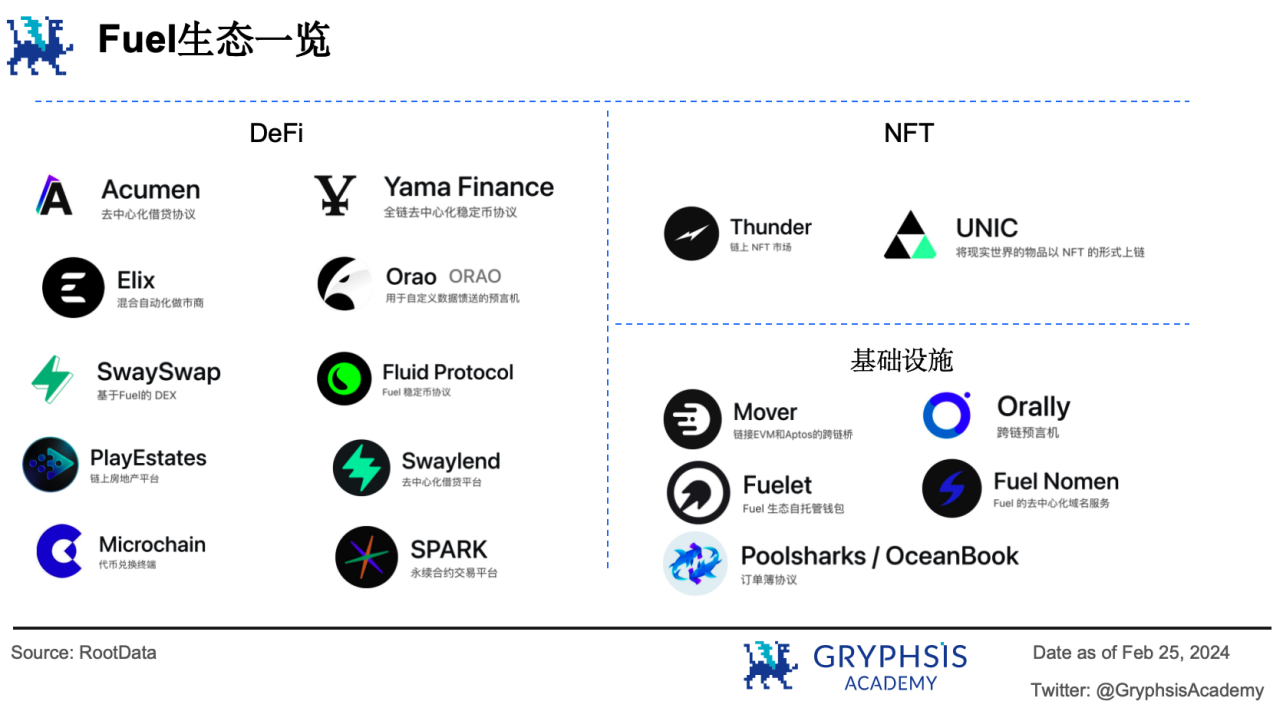
1. **Layer 0（网络）**：区块链底层网络，处理基础的网络通信协议
2. **Layer 1（基础设施）**：依赖各种共识机制对交易进行验证的去中心化网络
3. **Layer 2（扩展）**：依赖于 Layer 1 的各种二层协议，旨在解决 Layer 1 的各种局限性，尤其是可扩展性
4. **Layer 3（应用）**：依赖于 Layer 2 或 Layer 1 ，用于构建各种去中心化应用（ dApp ）



**并行 EVM 主要分为单体区块链和模块化区块链两大板块，单体区块链又分为 L1 和 L2 两类**。从项目总数和几个主要赛道的发展可以看出，各 L1 公链生态相比以太坊生态仍然存在很大的发展空间。对执行速度要求比较高的项目最容易出现在对“高速低费率”有要求的DeFi领域，以及对“强实时交互”有要求的游戏领域。而并行 EVM 叙事必然会给这些项目带来更好的用户体验，推动行业的发展进入到全新的阶段。

L1 是自带并行执行能力的新公链，是高性能基础设施。 L1 这一派中，以 Monad 和 Sei v2 为代表的项目自行设计并行 EVM ，兼容以太坊生态并提供更快速的交易处理能力。以 Solana 、 Aptos 和 Sui 为代表的项目则另起炉灶，成为以太坊强有力的竞争者。

L2 通过整合其他 L1 链的能力，提供跨生态合作的扩容能力，是 rollup 的显学。 L2 这一派中， Neon 是 Solana 网络上的 EVM 模拟器， Eclipse 利用 Solana 执行交易但在 EVM 上做结算。 Lumio 与 Eclipse 类似，只是把执行层换成了 Aptos 。



在上述单体区块链解决方案之外， Fuel 提出了自己的模块化区块链思路。它将在第二个版本中将自己定位成**rollup 操作系统**，提供更灵活、更彻底的模块化执行能力。 Fuel 专注于执行交易，而将其余部分外包给一个或多个独立层的区块链，从而实现更灵活的组合：既可以成为 L2 ，也可以成为 L1 ，甚至是侧链或状态通道。目前 Fuel 生态有17个项目，主要集中在 DeFi 、 NFT 和基础设施三个领域。不过只有 Orally 跨链预言机已投入实际应用。去中心化借贷平台 Swaylend 和永续合约交易平台 SPARK 上了测试网，其他项目还在开发中。

### 1.4 技术实现路径

要实现去中心化的交易执行，区块链网络必须履行4个职责：

* 执行：执行和验证交易
* 数据可用性：分发新区块到区块网络的所有节点
* 共识机制：验证区块，达成共识
* 结算：结算并记录交易的最终状态

**并行 EVM 叙事是对执行层的性能优化**。这又分为一层网络（ L1 ）解决方案和二层网络（ L2 ）解决方案两种。 L1 的解决方案引入交易并行执行机制，让交易在虚拟机中尽量并行执行。 L2 的解决方案本质上是利用已经并行化的 L1 虚拟机实现某种程度上的“链下执行+链上结算”。所以要理解并行 EVM 的技术原理，就要将其拆解开来：先理解什么是虚拟机（ virtual machine ）再理解什么是并行执行（ parallel execution ）。

#### 1.4.1 虚拟机

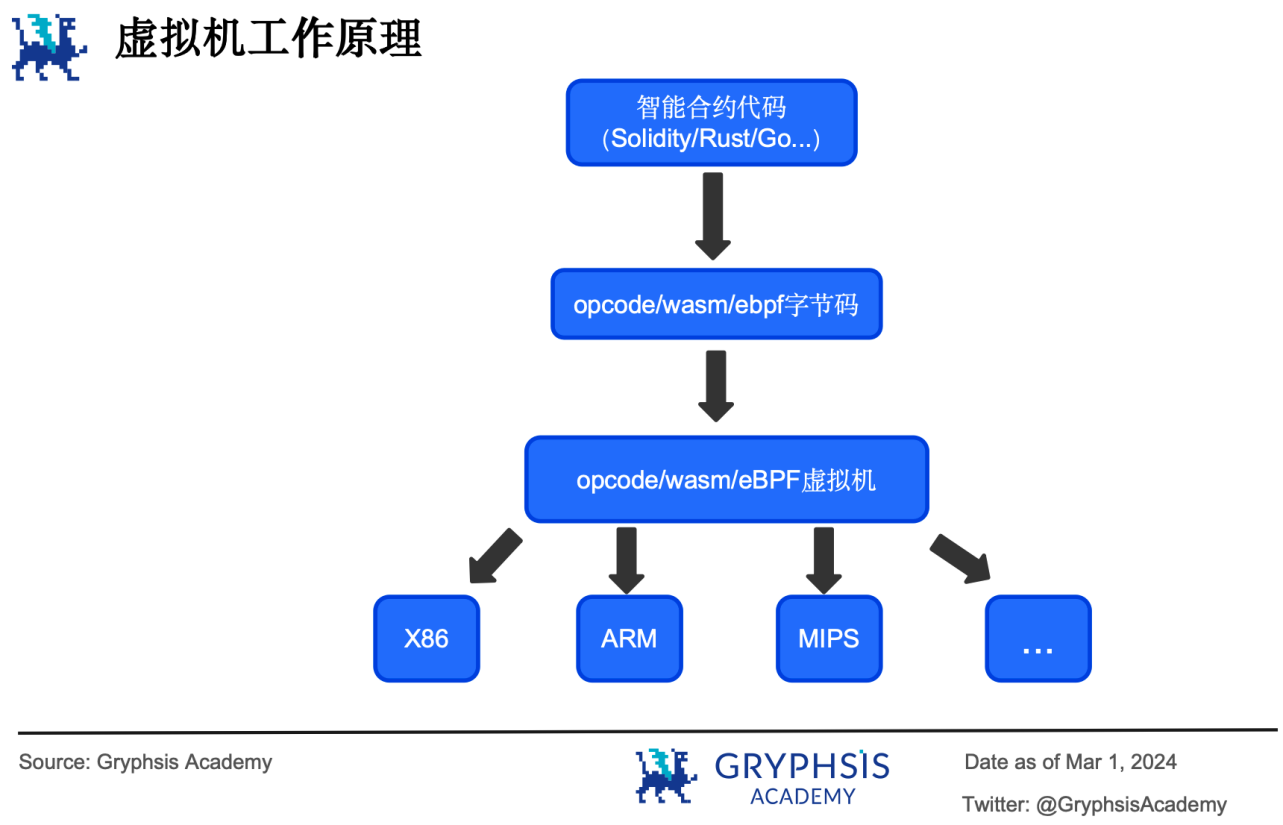
在计算机科学中，虚拟机是指对计算机系统进行的虚拟（ virtualization ）或者模拟（ emulation ）。虚拟机分为两种，一种叫系统虚拟机（ system virtual machine ），可以将一台物理机虚拟化为多台机器，运行多个操作系统，从而提高资源利用率。另一种叫进程虚拟机（ process virtual machine ），为某些高级编程语言提供抽象，让使用这种语言编写的计算机程序以一种平台无关的方式运行在不同平台上。 JVM 就是一种为 Java 编程语言设计的进程虚拟机。 Java 语言编写的程序首先被编译成 Java 字节码（一种中间状态的二进制代码）， Java 字节码由 JVM 解释执行： JVM 将字节码送给解释器，由解释器翻译成不同机器上的机器码，然后在机器上运行。

区块链虚拟机是进程虚拟机的一种。在区块链的语境下，虚拟机是指对分布式状态机进行的虚拟，用于分布式地执行合约，运行 dApp 。类比 JVM ， EVM 就是一种为 Solidity 语言设计的进程虚拟机，智能合约首先被编译成 opcode 字节码，然后由 EVM 解释执行。

以太坊之外的新兴公链在实现自己的虚拟机时，更多采用的是基于 WASM 或 eBPF 字节码的虚拟机。 WASM 是一种体积小、加载快、可移植且基于沙盒安全机制的字节码格式，开发人员可以使用多种编程语言（ C 、 C++ 、 Rust 、 Go 、 Python 、 Java 甚至 TypeScript 等）编写智能合约，然后编译成 WASM 字节码并执行。 Sei 公链上执行的智能合约正是采用了这一字节码格式。

eBPF 前身是 BPF （ Berkeley Packet Filter ，伯克利包过滤器），原本是用于网络数据包的高效过滤，后经过演化形成了 eBPF ，提供更丰富的指令集。它是一项允许在不改动源码的情况下对操作系统内核进行动态干预和修改其行为的革命性技术。后来这项技术从内核中走出来，发展出了用户态 eBPF 运行时，其具有高性能、安全和可移植性。在 Solana 上执行的智能合约都会编译成 eBPF 字节码并在其区块链网络上运行。

而其他的 L1 公链中， Aptos 和 Sui 使用 Move 智能合约编程语言，编译成特有的字节码在 Move 虚拟机上执行。 Monad 则自行设计了兼容 EVM opcode 字节码（ Shanghai fork ）的虚拟机。



#### 1.4.2 并行执行

并行执行是这样一种技术：

1. 能够发挥多核处理器的优势同时处理多个任务，增大系统吞吐量；
2. 确保得到的交易结果与按顺序串行执行交易时完全相同。

区块链网络常用 TPS （每秒处理的交易数量）作为衡量处理速度的技术指标。并行执行的机制比较复杂，也很考验开发人员的技术水平，要解释清楚并不容易。下面从一个“银行”的例子入手，解释什么是并行执行。

（1）首先，什么是串行执行？

情况1：如果我们把系统看成一家银行，把处理任务的 CPU 看成柜台，那么串行执行任务就好比这家银行只有一个柜台受理业务。此时来银行办业务的客户（任务）只能排成一条长龙，挨个办业务。对于每个客户，柜台工作人员都要重复同样的动作（执行指令）来为客户办理业务。没有轮到自己时客户只能等待，这就造成交易时间的延长。

（2）那么什么是并行执行呢？

情况2：此时银行看到人满为患，就多开了几个柜台来处理业务，有4个柜员在柜台同时处理业务，速度就比原来快了约4倍，那么客户排队的时间大约也减少到了原来的1/4，银行办理业务的速度就提升。

（3）如果不做保护，两个人同时给另一个人转账会发生什么错误？

情况3： A 、 B 和 C 三个人，他们的账户上分别有2 ETH 、1 ETH 和0 ETH ，现在 A 和 B 分别要给 C 转账0.5 ETH 。在一个交易串行执行的系统中，不会出现任何问题（左箭头“<=”表示读取账本，右箭头“=>”表示写入账本，下同）：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 步骤 | A | B | C | 备注 |
| 1 | <= 2 | / | <= 0 | 任务1：从账本中读取 A 的账户余额和 C 的账户余额 |
| 2 | 2-0.5 | / | 0+0.5 | 任务1：计算收支平衡： A 账户余额减去0.5， C 账户余额增加0.5 |
| 3 | => 1.5 | / | => 0.5 | 任务1：将 A 和 C 的账户余额更新回账本 |
| 4 | / | <= 1 | <= 0.5 | 任务2：从账本中读取 B 的账户余额和 C 的账户余额 |
| 5 | / | 1-0.5 | 0.5+0.5 | 任务2：计算收支平衡：B账户余额减去0.5， C 账户余额增加 0.5 |
| 6 | / | => 0.5 | => 1 | 任务2：将 B 和 C 的账户余额更新回账本 |

但并行执行并没有它看上去的那么简单。有很多很微妙的细节，稍不注意就会导致**非常严重的错误**。如果 A 和 B 对 C 的转账交易是并行执行的，那么根据每个步骤执行先后顺序的不同，就有可能产生不一致的结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 步骤 | A | B | C | 备注 |
| 1 | <= 2 | / | <= 0 | 任务1：从账本中读取 A 和 C 的账户余额 |
| 2 | 2-0.5 | / | 0+0.5 | 任务1：计算收支平衡， A 账户余额减去0.5， C 账户余额增加0.5 |
| \*3 | / | <= 1 | <= 0 | 任务2：从账本中读取 B 和 C 的账户余额（注意，还是0！） |
| 4 | => 1.5 | / | => 0.5 | 任务1：将 A 和 C 的账户余额更新回账本 |
| \*5 | / | 1-0.5 | 0+0.5 | 任务2：计算收支平衡， B 账户余额减去0.5， C 账户余额增加0.5 |
| \*6 | / | => 0.5 | => 0.5 | 任务2：将 B 和 C 的账户余额更新回账本 |

并行任务1执行 A 到 C 的转账，并行任务2执行 B 到 C 的转账。表中带\*号的步骤都是有问题的：由于任务是并行执行的，在步骤2中并行任务1做的收支平衡计算还没来得及写入账本，在步骤3中任务2就把 C 的账户余额（此时仍然是0）读取出来了，并在步骤5中基于余额0做了错误的收支平衡计算，然后在步骤6的账本更新操作中把步骤4中已经更新的账户余额0.5错误地再次更新成了0.5。导致虽然 A 和 B 都同时转账了0.5 ETH 给 C ，而交易执行完毕后 C 的账户余额却只有0.5 ETH ，另外0.5 ETH 不翼而飞了。

（4）如果不做保护，没有依赖关系的两个任务并行执行不会出错

情况4：并行任务1执行 A （余额2 ETH ）转账0.5 ETH 给 C （余额0 ETH ），并行任务2执行 B （余额1 ETH ）转账0.5 ETH 到 D （余额0 ETH ）。可以看到两个转账任务之间**没有依赖关系**。那么不管两个任务的步骤如何交错执行，都不会有上面的问题：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 步骤 | A | B | C | D | 备注 |
| 1 | <= 2 | / | <= 0 | / | 任务1：从账本中读取 A 和 C 的账户余额 |
| 2 | / | <= 1 | / | <= 0 | 任务2：从账本中读取 B 和 D 的账户余额 |
| 3 | 2-0.5 | / | 0+0.5 | / | 任务1：计算收支平衡， A 账户余额减去0.5， C 账户余额增加0.5 |
| 4 | / | 1-0.5 | / | 0+0.5 | 任务2：计算收支平衡， B 账户余额减去0.5， D 账户余额增加0.5 |
| 5 | =>1.5 | / | =>0.5 | / | 任务1：将 A 和 C 的账户余额更新回账本 |
| 6 | / | =>0.5 | / | =>0.5 | 任务2：将 B 和 D 的账户余额更新回账本 |

从两种场景的对比可以分析得出，只要任务之间存在**依赖**，并行执行时就有可能发生状态更新错误，反之则不会发生错误。满足以下2个条件之一，就称任务（交易）之间是有依赖关系的：

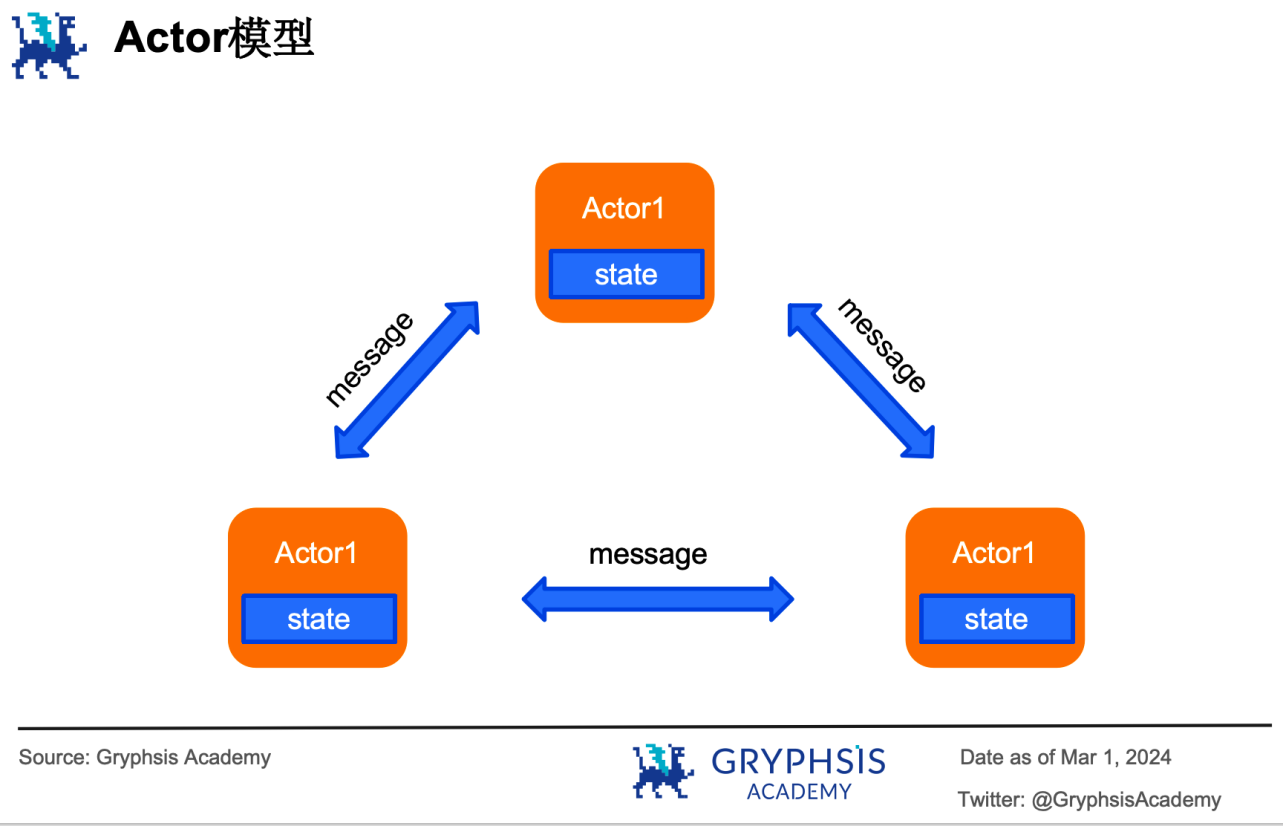
1. 一个任务写入的输出地址是另一个任务读取的输入地址；
2. 两个任务输出到同一个地址。

这并不是去中心化特有的。任何涉及到并行执行的场景都会因为多个有依赖的任务之间不受保护地访问共享资源（银行例子中的“账本”、计算机系统中的共享内存等）而出现数据不一致，称为**竞态条件问题（ data races ）**。业界在解决并行执行的竞态条件问题上提出了三种执行机制：**消息传递机制**、**共享内存机制**和**严格状态访问列表机制**。

#### 1.4.3 消息传递机制

情况5：假设银行有4个柜台同时为客户办理业务，现在给4个柜台的柜员每个人手里发一本专属账本，这个账本只有自己能修改。上面记录了自己所服务的客户的账户余额。每个柜员在为客户办理业务时，如果该客户的信息在自己的账本上能查到，那么就直接办理；否则就给别的柜员喊话告知客户要办理的业务，别的柜员听到后进行办理。

这就是消息传递模型的原理。**Actor 模型**就是消息传递模型的一种，每一个负责处理交易的执行者都是一个 actor （柜员），它们都有可以访问自己的私有数据（专属账本），如果要访问别人的私有数据，只能通过发消息来实现。

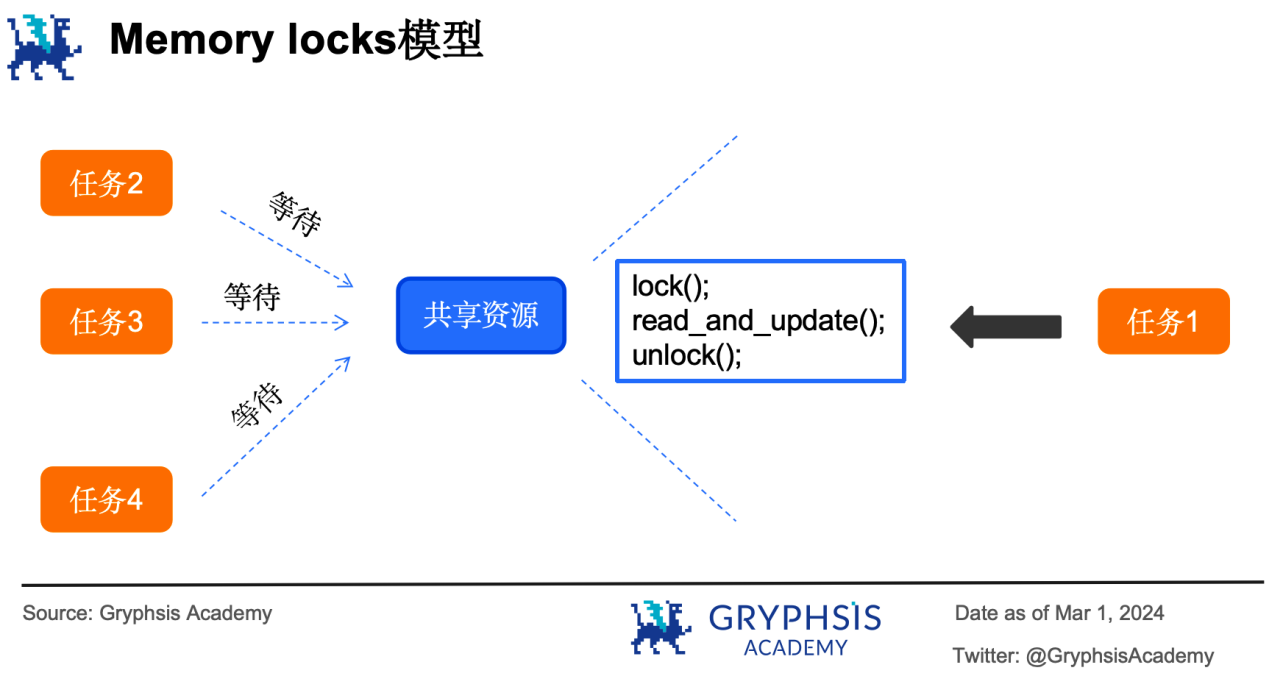


Actor 模型的优点在于每个 actor 都只能访问自己的私有数据，因此就不会出现竞态条件问题。它的缺点有2个，一是每个 actor 都只能串行地执行，在某些场景中并没有发挥并行优势（比如2号、3号和4号柜员同时发消息问1号柜员客户A的账户余额是多少，1号柜员在这样的模型中就只能一个一个的处理消息，而这本来是可以并行处理的）。二是没有一个全局的有关当前系统状态的信息，如果系统业务复杂，将很难了解全貌、定位和修复 bug 。

#### 1.4.4 共享内存机制

##### 1.4.4.1 内存锁模型

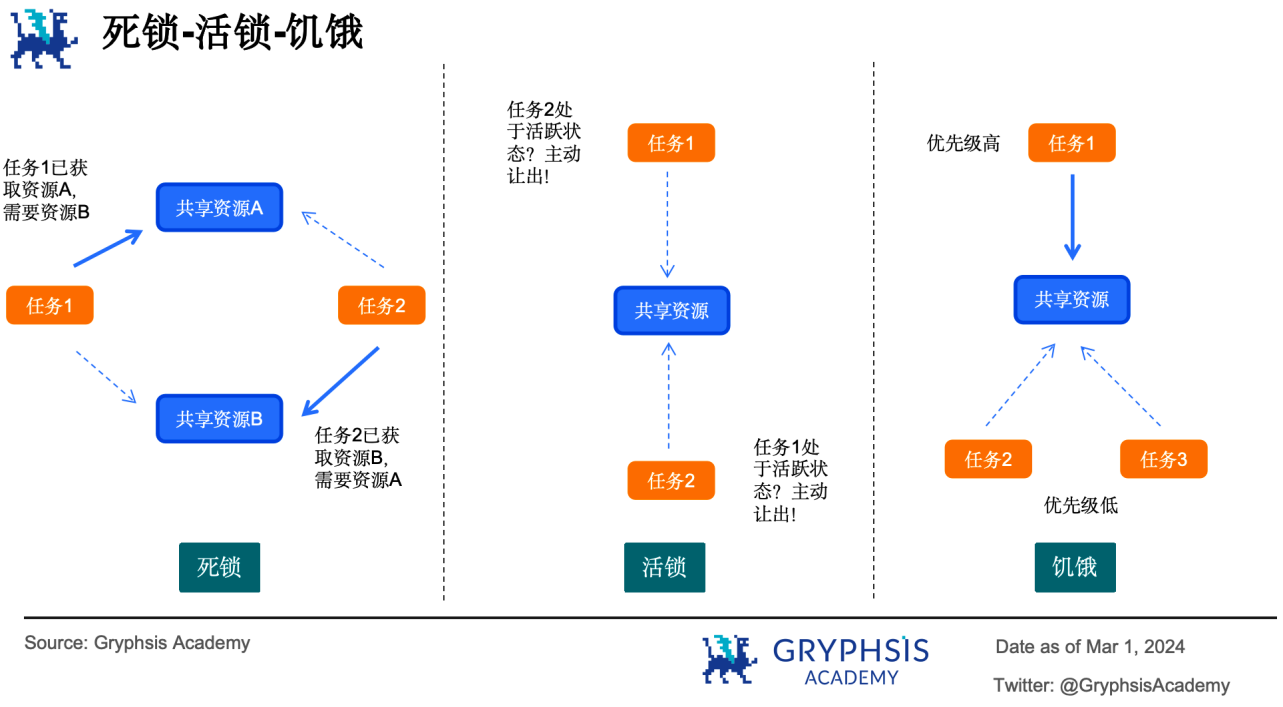
情况6：假设银行只有一个大账本，上面记录了所有客户的账户余额。账本旁边只有一支签字笔可以用来修改账本。在这种场景下，4个柜员在办理业务时就看谁跑得快了：一个柜员抢先拿到签字笔（加锁）开始办理业务修改账本，其他3个柜员就只能等着。直到柜员用完将笔放下（解锁），其他3个柜员再去争夺签字笔的使用权，如此循环下去，这就是**内存锁模型（ memory locks ）**。内存锁就是让并行执行的任务在访问共享资源的时候做一个锁（ lock ）的操作，锁住之后对共享资源进行访问，此时别的任务要等待它修改完之后解锁（ unlock ）才能再次锁住并访问。**读写锁（ read-write lock ）**的处理更精细化，可以对共享资源加**读锁（ read lock ）**或者**写锁（ write lock ）**。区别是多个并行任务可以加多次读锁并读取共享资源数据，此时不允许修改；而写锁只能加一把，且加了之后只能由加锁者独占访问。



Solana 、 Sui 和 Sei v1 采用的就是基于内存锁的共享内存模型。这种机制看上去简单，但实现起来很复杂，很考验开发人员对多线程编程的驾驭能力。一个不小心，就会留下各种各样的 bug ：

1. 情况1：任务锁住共享资源，但在执行的过程中因出错崩溃退出，共享资源被锁住无法访问；
2. 情况2：任务已经加锁，但是执行过程中因业务逻辑嵌套出现二次加锁，导致自己等待自己。

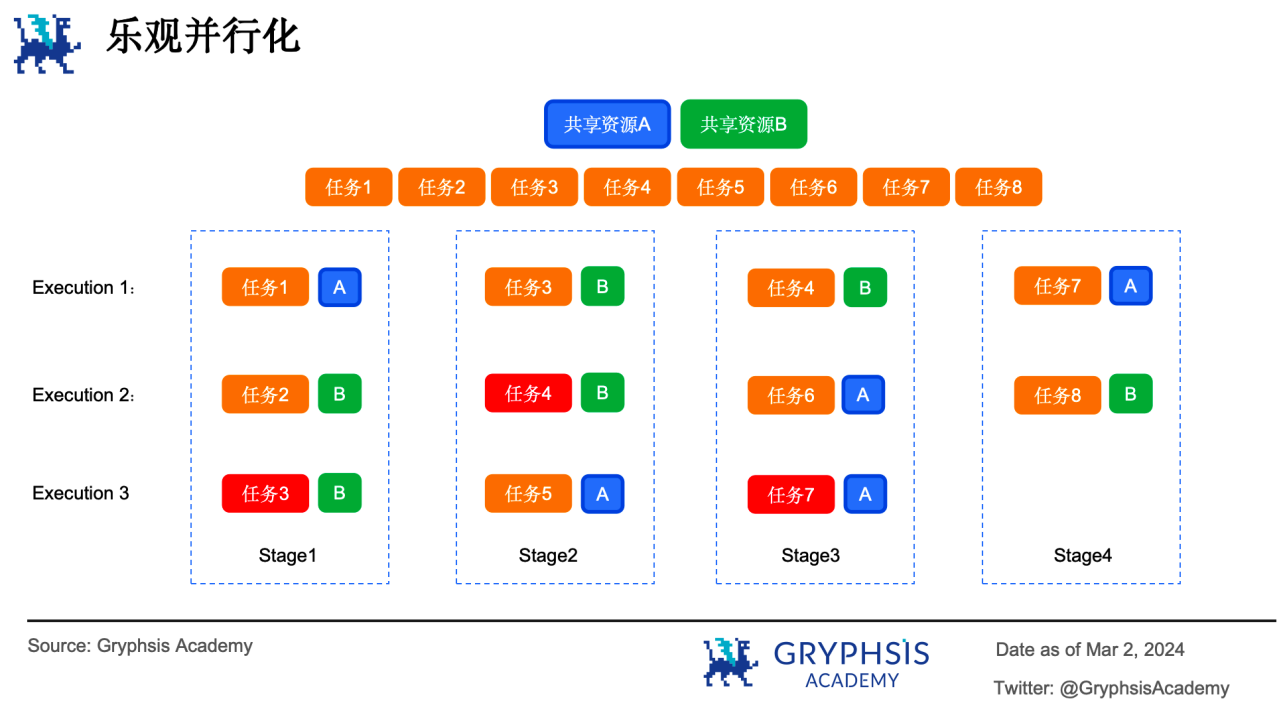
内存锁模型最容易出现**死锁（ dead lock ）**、**活锁（ live lock ）**和**饥饿（ starvation ）**等问题：（1）多个并行任务争夺多个共享资源，每个任务都占有了其中的一部分，都在等待对方释放资源，就会出现死锁问题；（2）并行任务检测发现还有其他并行任务处于活跃状态，于是主动让出自己占有的共享资源，导致你让过来我让过去，发生活锁；（3）优先级高的并行任务总是能获得共享资源访问权，其他低优先级任务长时间等待，发生“饥饿”。



##### 1.4.4.2 乐观并行化

情况7：银行4个柜员每个人在办理业务时都可以独立查阅和修改账本，而不管别的柜员在没在用账本。柜员使用账本时在自己查阅和修改了的内容上贴一个个人专属标签。每次办完业务后都会再浏览一遍，如果发现贴的不是自己的标签，说明记录被别的柜员修改过了，本次业务就要作废并重新办理。

这就是**乐观并行化**的基本原理。乐观并行化的核心思想是**先假设所有的任务都是相互独立的**。先并行执行任务，然后再验证每个任务，如果验证不通过，则把这个任务重新执行一遍，直到所有任务执行完毕。



假设有8个并行任务以乐观并行化的方式执行，期间需要访问2个共享资源 A 和 B 。阶段1执行时，任务1、任务2和任务3并行执行。但任务2和任务3同时访问共享资源 B 产生了冲突，因此任务3在下一阶段重新调度执行。阶段2执行时任务3和任务4同时访问了共享资源 B ，此时任务4重新调度执行，以此类推，直到所有任务执行完毕。可以看到整个过程中发生冲突的任务会不断地重复执行。

乐观并行化模型采用了一种**多版本内存数据结构（ multi-version in-memory data structure ）**用来记录每一个写入值及其版本信息（类似于银行柜员贴标签）。每个并行任务的运行分为两个阶段：执行（ execution ）和验证（ validation ）。执行阶段会把所有读数据和写数据的行为记录下来，形成读集（ read set ）和写集（ write set ）。验证阶段会用读集和写集与多版本数据结构进行对比，如果对比发现不是最新的，则验证不通过。

乐观并行化模型发端于**软件事务内存（ Software Transaction Memory ，简称 STM ）**，后者是数据库领域无锁编程的一种机制。由于区块链网络对交易的处理天然的具备一种确定的顺序，因此这一概念被引入并演化出了 Block-STM 机制。 Aptos 和 Monad 采用了 Block-STM 作为自己的并行执行机制。值得一提的是， Sei 公链在即将发布的 v2 版本中弃用了原来的内存锁模型，改为采用乐观并行化模型。 Block-STM 执行速度极快，实验环境中 Aptos 交易执行速度达到了惊人的160k tps ，比顺序执行交易快了18倍。 Block-STM 把复杂的交易执行和验证交给了实现底层机制的核心团队，开发者可以毫不费劲地像写顺序执行的程序一样编写智能合约。

#### 1.4.5 严格状态访问列表

消息传递和共享内存机制是基于**账户/余额**数据模型实现的，它记录了每个链上账户的余额信息。就好比银行的账本中记录的是客户 A 有余额1000元，客户 B 有余额600元，每次处理交易只需要修改一下账户余额状态。如果换一种思路，还可以在每次交易时只记录交易的具体内容，形成交易流水，也能通过交易流水来计算用户的账户余额，例如有如下的交易流水：

1. 客户 A 开户并存入1000元；
2. 客户 B 开户（0元）；
3. 客户 A 向客户 B 转账100元。

通过读取流水并进行计算，可以知道当前客户 A 账户余额为900元；客户 B 账户余额为100元。**UTXO （ unspent tx output ，未花费的交易输出）**就类似这样的交易流水数据模型，它是第一代区块链比特币用来表示数字货币的一种方式。每一笔交易都有输入（怎么获得的）和输出（怎么花掉的），而 UTXO 可以简单理解为还没有花掉的收款。比如客户A有6个 BTC ，他给客户B转账5.2个 BTC ，还剩0.8个 BTC ，从 UTXO 的角度就可以看成： A 的6个价值1 BTC 的 UTXO 被销毁， B 获得了1个价值为5.2 BTC 的新生成的 UTXO ，同时找零给 A 一个价值0.8 BTC 的新生成的 UTXO 。即6个 UTXO 被销毁，生成了2个新的 UTXO 。

交易的输入和输出串成链，并使用数字签名记录所有权信息，就形成了 UTXO 模型。采取这种数据模型的区块链需要对某个账户地址的所有 UTXO 求和，才能知道当前账户余额。**严格状态访问列表（ strict state access list ）**基于 UTXO 模型实现并行执行。它会提前计算每个交易要访问的账户地址，形成访问列表。访问列表有两个作用：（1）判断交易安全性：交易执行时如果访问了不在访问列表中的地址，则执行失败。（2）并行执行交易：根据访问列表形成交易的多个集合，每个交易集合之间在访问列表上没有交集（没有依赖），因此多个交易集合就可以并行执行了。

### 1.5 行业增长驱动力

从内在规律上看，任何事物的发展都会经历”从无到有“到”从有到优“的过程，人类对更快速度的追求是永恒的。为解决区块网络执行速度的问题诞生了各种各样或链上或链下的解决思路。以 rollup 为代表的链下解决方案已得到充分的价值发现，而并行 EVM 作为新叙事还有很大探索空间。

从历史背景上看，随着 SEC 批准现货比特币 ETF 和即将发生的比特币减半等事件，再叠加美联储可能的降息操作，加密货币预期将迎来一段大牛市，行业的蓬勃发展需要更大吞吐量的区块链网络基础设施作为坚实的基础。

从资源管理上看，传统的区块网络是串行地处理交易的，这种处理方法虽然简单，但也是对处理器资源的一种浪费。而并行 EVM 实现了计算资源的物尽其用，充分”榨干“了多核处理器的性能，提高了区块网络的整体效能。

从行业的发展上看，虽然各种技术和商业模式的创新层出不穷，但 Web3 的成长潜力仍然有待挖掘。中心化网络能做到每秒推送50000多条消息、发送340万封邮件、完成100000次谷歌搜索并让成千上万个玩家同时在线，而去中心化暂时还做不到。去中心化要与中心化分庭抗礼，拿下属于自己的半壁江山，不断优化并行执行机制，提高交易的吞吐量也是发展的必经之路。

从去中心化应用的发展上看，要吸引更多用户，体验上一定要下功夫。性能优化是提升用户体验的方向之一。对于 DeFi 用户来说，要满足高交易速度，低费率的需求。对于 GameFi 用户来说，要满足实时交互的需求。这些都需要并行执行作为支撑。

### 1.6 存在的问题

去中心化、安全性和可扩展性三者只能满足其二，此为**区块链不可能三角**。既然“去中心化”是不可撼动的一极，那么“可扩展性”的提高就意味着“安全性”的降低。代码是人写的，是人写的就容易出现错误。并行计算所带来的技术复杂性为安全隐患的滋生提供了温床。多线程编程一是容易因为各种复杂的并发控制操作不当导致竞态条件问题；二是容易因为访问了无效的内存地址导致崩溃，甚至出现容易被攻击者利用的缓冲区溢出漏洞。

至少能从三个角度评估项目的安全性。一看**团队背景**。有系统编程经验的团队对多线程编程有丰富的经验，见过并能处理80%的疑难杂症。系统编程一般涉及如下领域：

* 操作系统
* 各类设备驱动
* 文件系统
* 数据库
* 嵌入式设备
* 密码学
* 多媒体编解码
* 内存管理
* 网络
* 虚拟化
* 游戏
* 高级编程语言

二看**代码可维护性**，编写可维护性强的代码是有章可循的，比如清晰的架构设计，合理运用设计模式实现代码的可复用性，运用了测试驱动开发技术编写了足够多的单元测试代码，通过合理的重构消除重复代码等等。三看采用的**编程语言**，有的新锐编程语言从设计上就强调内存安全的高并发性。编译器会检查代码，一旦发现代码有并发问题或者可能访问无效的内存地址，就会编译失败，从而强迫开发者写出足够健壮的代码。 Rust 语言就是出类拔萃者，这也就是为什么我们会看到绝大多数并行 EVM 项目都是用Rust语言开发的原因。有的项目甚至借鉴了 Rust 语言的设计实现了自己的智能合约语言，如 Aptos 和 Sui 的 Move 语言、 Fuel 的 Sway 语言等等。

## 2 标的梳理

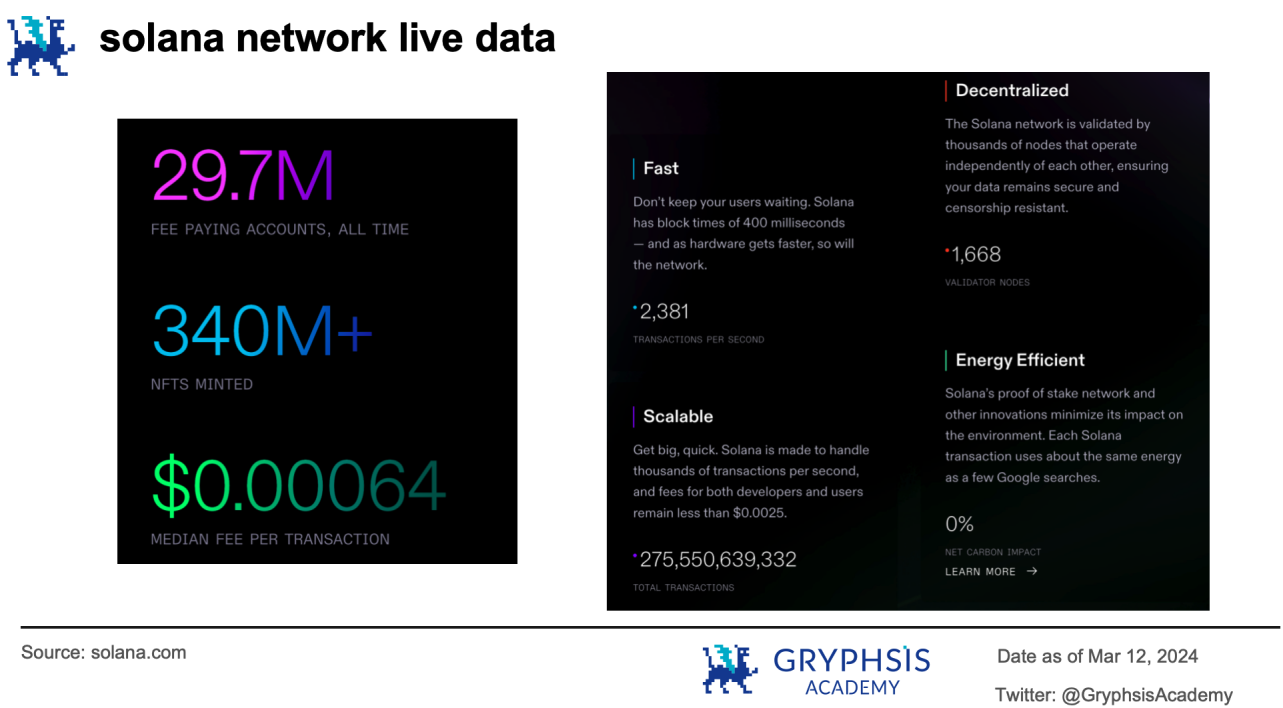
### 2.1 以太坊的强大竞争对手： Solana

成立于2017年的 Solana 是一种高性能的底层区块链协议，大规模应用于金融、 NFT 、支付和游戏等领域。具有开放、可互操作和去中心化的特点。项目目前由总部在瑞士日内瓦的 Solana 基金会负责维护。

作为 Layer1 基础设施， Solana 协议使用 Rust 语言进行开发，并实现了基于 eBPF 指令集的虚拟机 SVM 。在并行执行机制上， Solana 采用的是账户数据模型和内存锁并发多线程机制。开发人员使用 Rust 语言（也可以使用 C/C++/Python ）编写的智能合约（ Solana 称之为“链上程序“）由 LLVM 编译器编译成 BPF 字节码文件，并加载到 SVM 中执行。 Solana 首创了 Sealevel 并行智能合约运行时，具备同时并行处理成千上万智能合约的能力，并能最大化使用网络验证节点的多核 CPU 。每一笔 Solana 交易都包含一个指令向量，其中每条指令都包含该交易要读写的账户列表。 Sealevel 运行时会对待执行交易进行排序，并调度访问不相交账户列表的交易并行执行。 Sealevel 运行时还采用了 SIMD （ single instruction on multiple data ，单指令多数据）技术，允许相同的一段代码（主要是涉及密码学的数值运算）在多个数据流上执行，并在 CPU 和 GPU 之间水平扩展，进一步提高执行速度。

Solana 核心团队拥有高通、英特尔等知名科技公司背景。公开资料显示，有29个投资机构对其进行了4轮共计3.3576亿美元融资。 Solana 自己还对社交区块链游项目 Faraway Games 投资2100万美元。

经过多年的发展， Solana 形成了庞大的生态，横跨20种类别。整个 Solana 社区有2970万个活跃账户、3.4亿个 NFT 被铸造，每笔交易的平均成本低至$0.00064。目前 Solana 网络有1668个验证节点，每秒处理交易2381笔，交易总数已达到2700多亿。

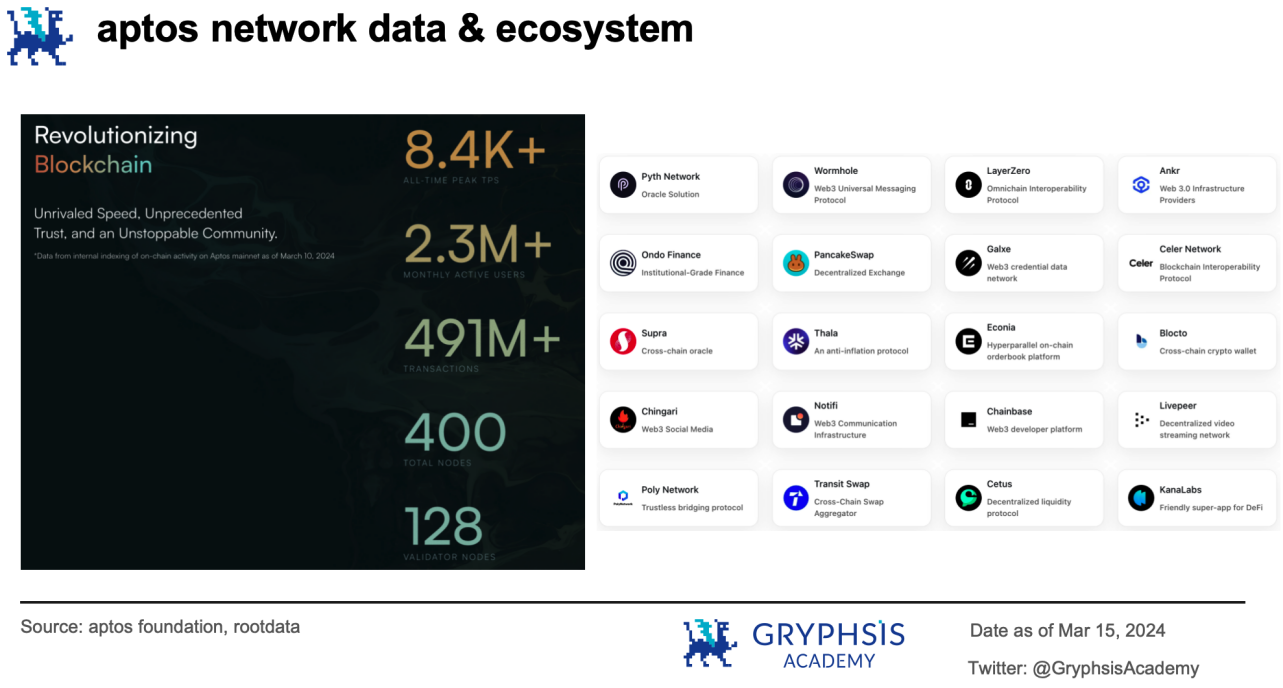


除了技术实力过硬， Solana 社区还十分注重营销，在各大主要社交平台运营了自己的账号，形成了社区矩阵。 Discord 社区目前拥有4万多名成员、 Twitter 有200多万关注者，官方经常举办各种黑客松大赛，社区热度很高。

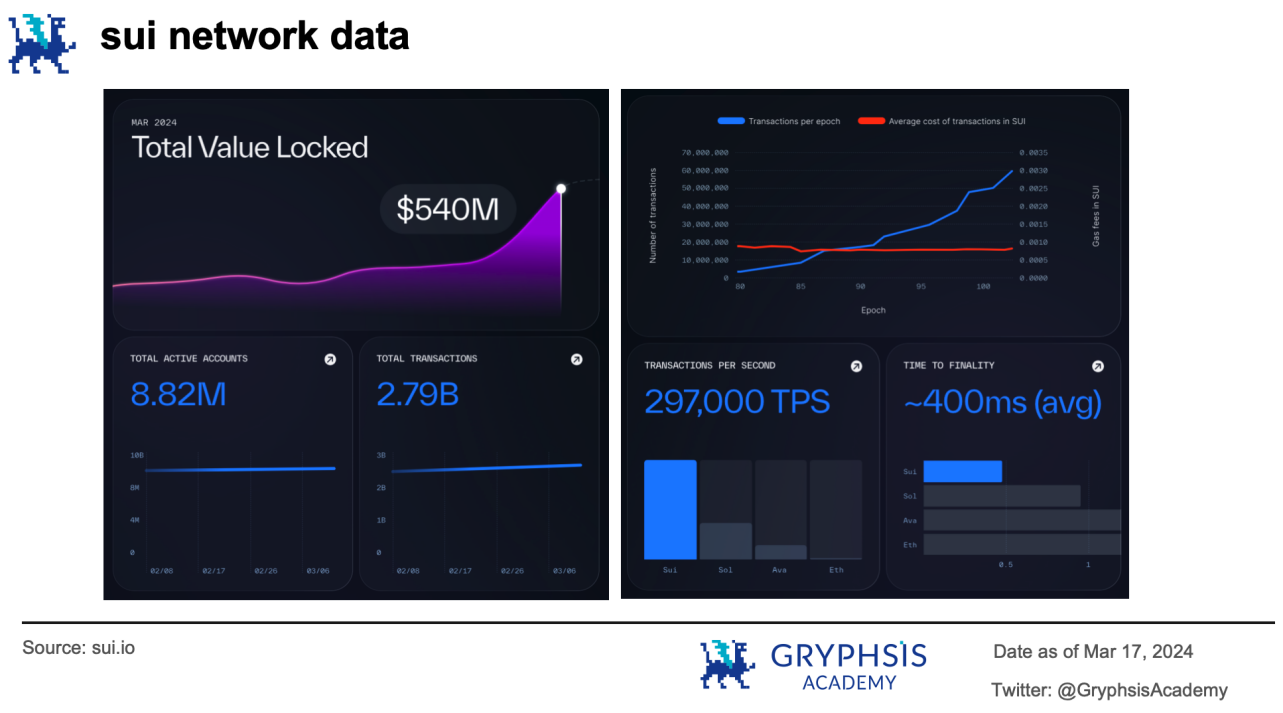


### 2.2 Move 系 L1 公链： Aptos & Sui

Move 系新公链发轫于 Meta（Facebook） 在2019年提出的 Libra/Diem 项目，是一种基于区块链的稳定币支付应用。虽然后来项目因为各种原因被叫停了，但相关团队成员没有放弃，纷纷出走自立门户，后来发展形成了两大公链： Aptos 和 Sui 。



Aptos 起步较早，2021年在美国由 Libra/Diem 核心成员成立，2022年10月之前就完成了4轮融资，总额3.5亿美元（有两轮战略融资金额未披露）。2022年10月17日 Aptos 在主网上线，目前流通市值约53亿美元。官方数据显示， Aptos 网络有400个节点，其中有128个验证节点，月活用户230多万，迄今为止已处理近5亿笔交易，全时 TPS 峰值达到8400以上。 Aptos 生态有158个项目，主要集中在 DeFi 、 GameFi 、基础设施和 NFT 赛道。运营方面， Discord 社区有24万成员， Twitter 有55万关注者。



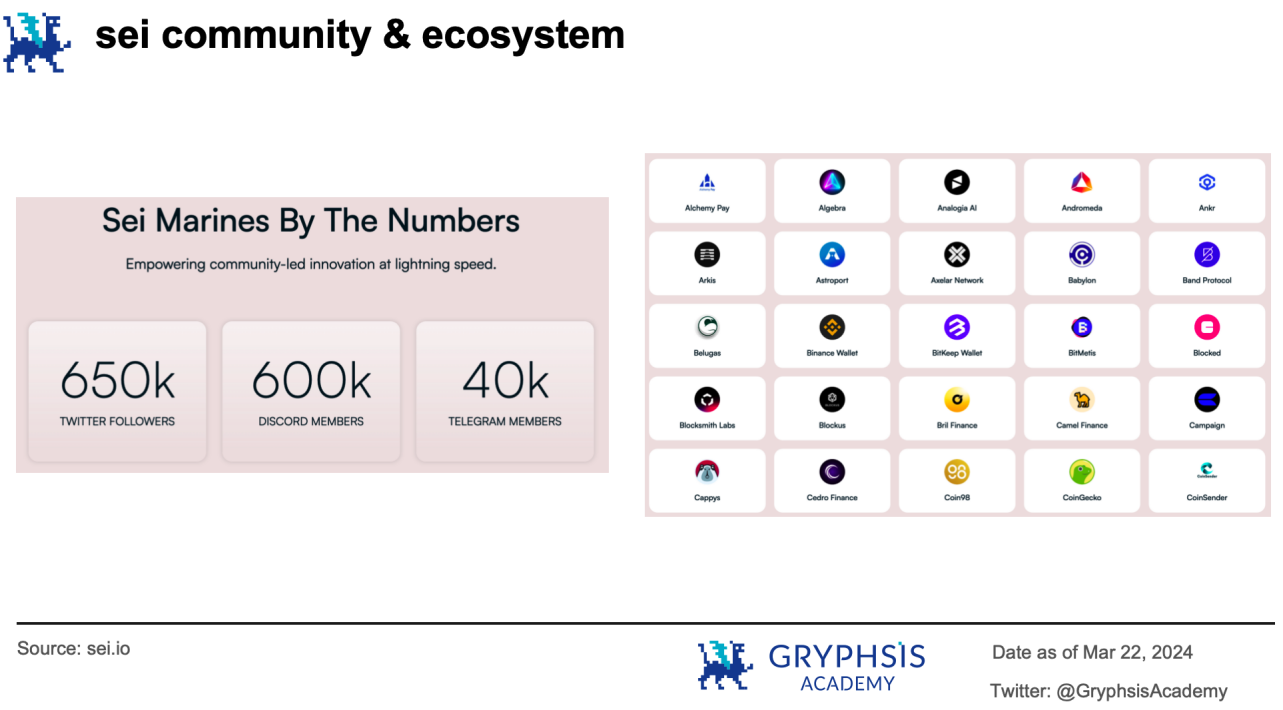
Sui 是后起之秀，于2022年在美国成立，但在2021年时就获得了一笔3600万美元的 A 轮融资，2022年9月又获得了一笔3亿美元 B 轮融资，融资总额3.36亿美元。2023年5月， Sui 在主网上线，目前流通市值19亿美元。官方数据显示， Sui 网络TVL达到5亿4千万美元，总活跃账户达8百万，迄今为止已处理20亿笔交易，峰值 TPS 达到惊人的297000，且交易费率很低，性能表现相当优秀。 Sui 生态目前有103个项目，主要集中在 DeFi 、 GameFi 、基础设施和 NFT 赛道。运营方面， Discord 社区有64万成员， Twitter 有66万关注者。

同为 Move 系公链，二者都是使用 Move 语言开发的。随 Libra/Diem 诞生的 Move 语言是专为区块链开发设计的，可以编译成字节码在原生的 Move 虚拟机上运行。 Move 语言受到 Rust 语言的启发，设计上强调资源稀缺性、保护和访问控制，提供了加密货币相关的语言级支持，使其成为安全、灵活，且适用于自定义交易和开发智能合约的语言。所不同的是二者在底层实现机制上有着细微的差别。

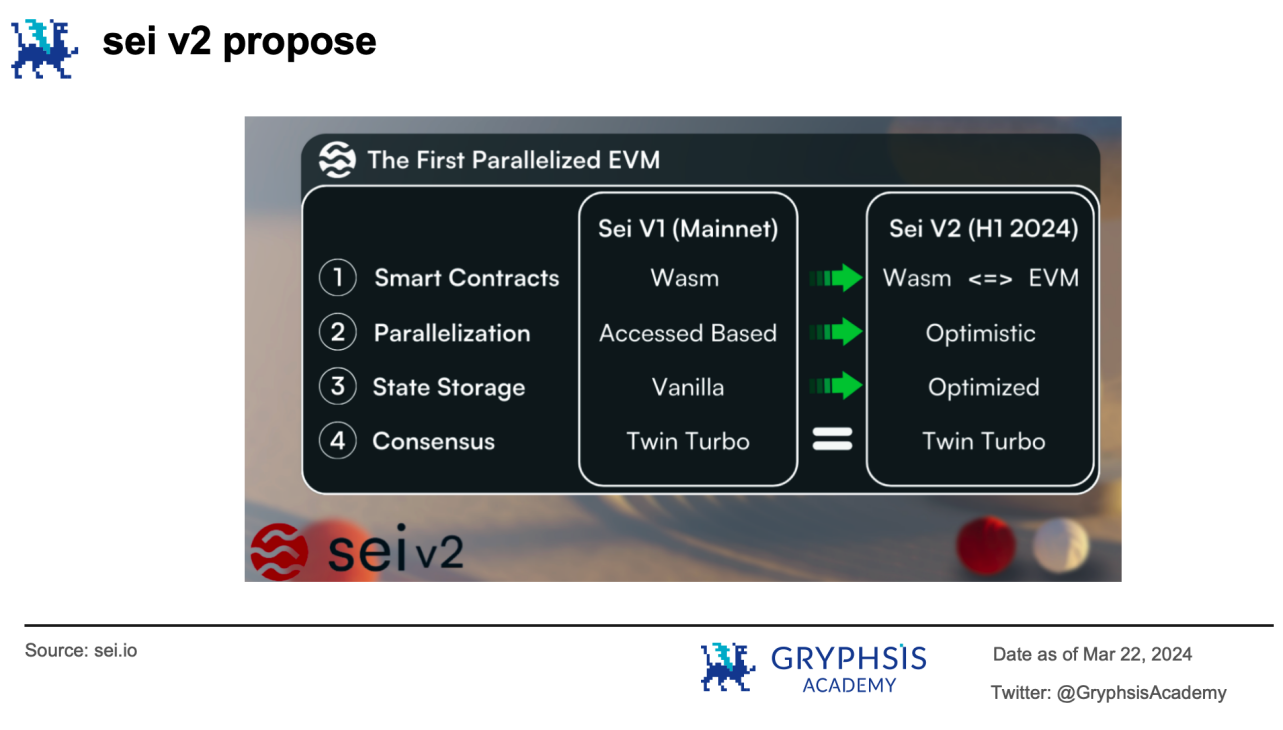
Aptos 参考 CPU 的超标量流水线技术，把区块的处理分为几个阶段，每个阶段完全独立且可并行运行。在交易的并行执行机制上使用了 Block-STM 乐观并行。对于访问数据重叠的交易， Aptos 自创了被称为“ delta writes ”的并行数据模型，用于记录对账户状态的修改集合，待所有的交易并行执行完毕后，再根据修改集合来确定账户最终状态。 Sui 采用的是内存锁的并行执行机制，但受益于其独特的数据模型设计，它做了更进一步的性能优化。

区别于大多数区块链以账户为中心构建的数据模型而言， Sui 最大的特点是构建了以对象为中心的数据模型，**它将合约管理的资产看作对象**。以账户为中心的设计需要在交易前后围绕账户数据做记账操作，也就需要一定的加锁/解锁进行并发控制。以对象为中心的设计则使得资产所有者信息只是对象模型（资产）中的一个元数据，它们的关系被“反转”，因此交易的并行处理变得更容易。交易以代表资产的对象为输入，对其进行读取、写入和修改，然后以修改过或新生成的对象为输出，每一个对象都知道自己是哪笔交易生成的，因此所有的交易和对象（资产）在结构上就构成了一个完整的有向无环图（ DAG ）。不但操作不同对象（资产）的交易可并行执行，而且对于转账交易（所有权变更），可以直接修改对象所有权，不需要共识层进行处理，这样就大大加快了交易的执行速度。

### 2.3 从内存锁走向乐观并行： Sei v2



Sei 是基于开源技术的通用公链，于2022年成立。两位创始人来自加州大学伯克利分校，团队其他成员也都具有海外名校背景。 Sei 共获得3轮融资，其中种子轮500万美元，第一轮战略融资3000万美元，第二轮战略融资未披露。 Sei Network 还募集了总共1亿美元基金，用于支持其生态发展。2023年8月， Sei 在主网上线，号称速度最快的 L1 公链，每秒可执行12500笔交易，最终确定性仅需380 ms，目前流通市值近22亿美元。目前 Sei 生态有118个项目，主要集中于 DeFi 、基础设施、 NFT 、游戏和钱包等赛道。社区目前在 Twitter 、 Discord 和 Telegram 分别有65万、60万和4万成员。



2023年11月末， Sei 在其官方博客宣布将于2024年上半年启动主网上线后最大的一次版本升级： Sei v2 。 Sei v2 号称**第一条并行 EVM 区块链**，本次版本升级将带来如下新功能：

* 对 EVM 智能合约的向后兼容：开发人员不用修改代码就可以迁移部署EVM智能合约
* 对诸如 Metamask 等常见工具/应用的重用
* 乐观并行化： Sei v2 将放弃内存锁的共享访问机制，转而采用乐观并行化
* SeiDB ：对存储层的优化
* 支持以太坊和其他链之间的无缝互操作性

Sei 网络原先的交易并行执行是基于内存锁模型实现的。在执行前所有待处理交易之间的依赖关系都会被解析并生成 DAG ，然后基于 DAG 对交易的执行顺序进行精确编排，这种方式增加了合约开发人员的心智负担，因为他们不得不在开发合约时将逻辑编写到代码中。正如上面技术原理部分所介绍的，新版本采用乐观并行化之后，开发人员就可以像写顺序执行的程序那样开发智能合约了。交易的调度、执行和验证等一系列复杂机制，都由底层模块负责处理。核心团队在优化的提案设计上还引入通过预填充依赖关系更进一步提升并行执行能力的设计。具体来说就是引入动态的依赖生成器，在执行前先分析交易的写操作并将其预填充到多版本内存数据结构中，优化潜在的数据争用。核心团队经过分析得出结论，这样的优化机制虽然不利于最好情况下的交易处理，但会显著提升最坏情况下的执行效率。

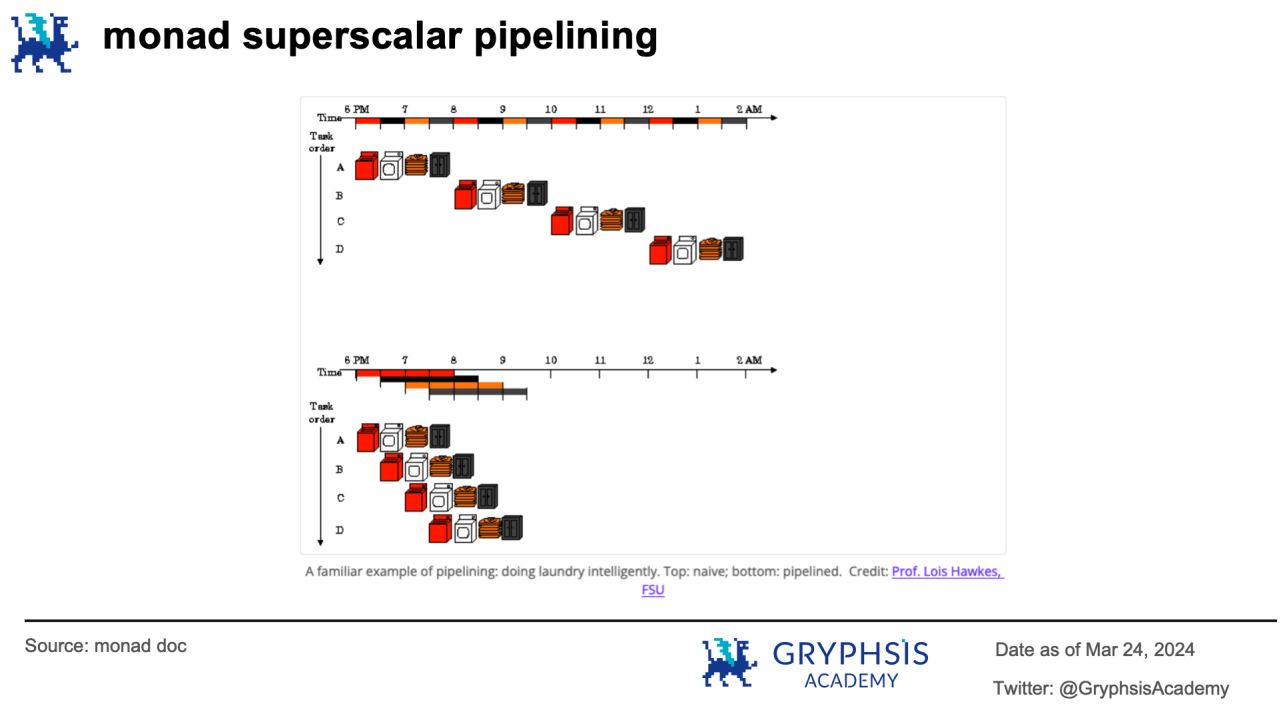
### 2.4 L1赛道潜在颠覆者： Monad

如果你错过了上面一众公链的发展，那么 Monad 你一定不能错过。因为它被誉为 L1 赛道的潜在颠覆者。 Monad 由 Jump Crypto 的两位高级工程师于2022年创立，项目于2023年2月完成1900万美元种子轮投资，2024年3月， Paradigm 就领投 Monad 的一轮超2亿美元的融资进行谈判，如果成功，这将是自开年以来最大的一笔加密货币融资。目前项目已经成功实现了上线内部测试网的里程碑任务，正朝着下一步开放公共测试网而努力。



Monad 深受资本青睐有两个突出的原因：一是技术背景过硬，二是善于营销炒作。 Monad Labs 团队核心成员有30人，均在高频交易、内核驱动和金融科技领域深耕数十年，在分布式系统领域有丰富的开发经验。项目的日常运营也十分的“接地气”：持续地对其 Twitter 上20万关注者和 Discord 上15万成员进行“魔性营销”。比如每周举办 meme 大赛，向社区征集各种奇奇怪怪紫色动物的表情包或者视频，在社区进行“精神传播”。

Monad 的愿景是成为面向开发者的智能合约平台，为以太坊生态带来极致的性能提升。 Monad 为以太坊虚拟机引入了两项机制：一是**超标量流水线技术**，二是**改进的乐观并行机制**。



**超标量流水线技术将交易的执行阶段并行化**。官方文档举了一个很形象的例子。洗衣服就像区块链对交易的处理，也要分多个阶段完成。传统的处理方式是对每堆脏衣服进行清洗、烘干、折叠和储存，然后再处理下一堆。超标量流水线则是在第一堆衣服烘干的时候就开始第二堆衣服的清洗，第一堆衣服在折叠的时候第二堆、第三堆衣服已经在分别烘干和清洗了，因此每一个阶段的处理都不闲着。

**乐观并行机制将交易的执行并行化**。 Monad 采用了乐观并行化实现并行执行。它还实现了自己的静态代码分析器，用于预测交易之间的依赖关系，仅在前置的依赖交易执行完毕后再调度后续交易执行，这样就大大减少了因验证失败导致的交易重复执行。目前性能达到10000 TPS 并能在1秒的时间出块。随着项目的进展，核心团队还会继续探索更多的优化机制。

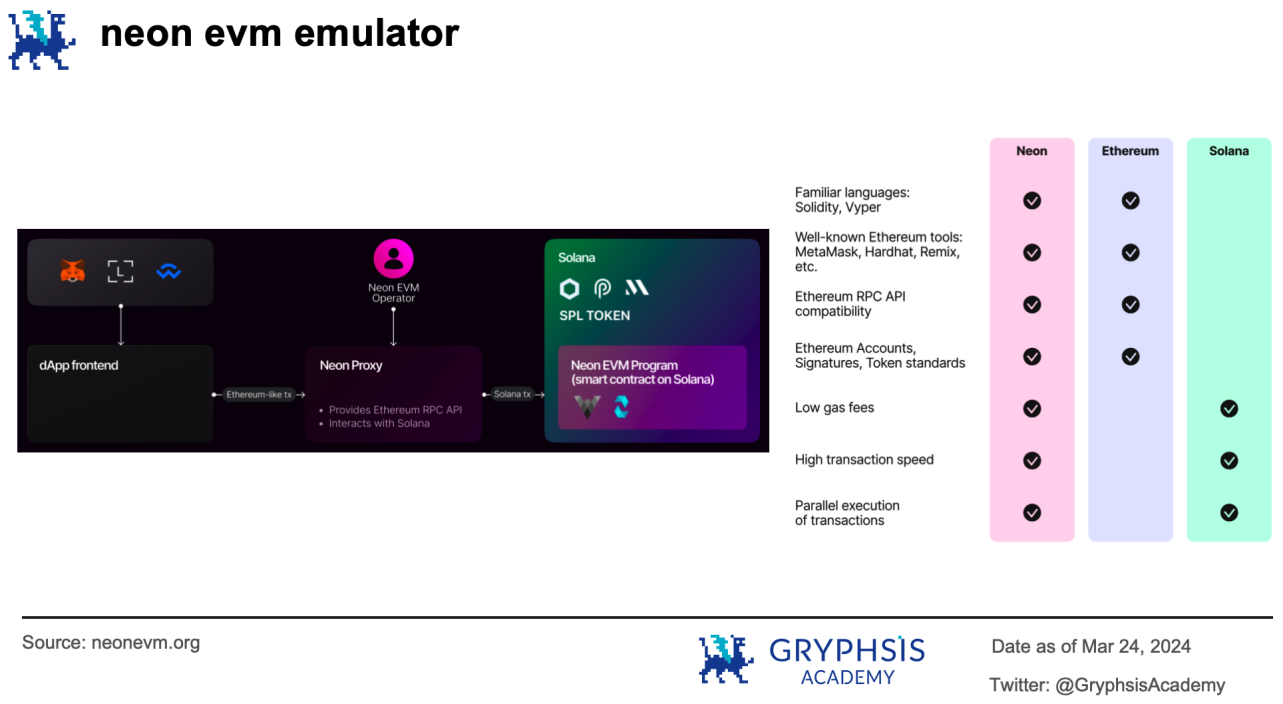
### 2.5 三种L2解决方案对比： Neon 、 Eclipse 和 Lumio

L2 解决方案有个共性：它们将两种虚拟机的能力结合起来，提升交易的执行速度。具体地说就是利用并行 L1 来执行交易，但是与其他链进行兼容（双虚拟机支持）。所不同的是不同的项目采取的兼容机制不一样。这方面 Neon 、 Eclipse 和 Lumio 颇具代表性。

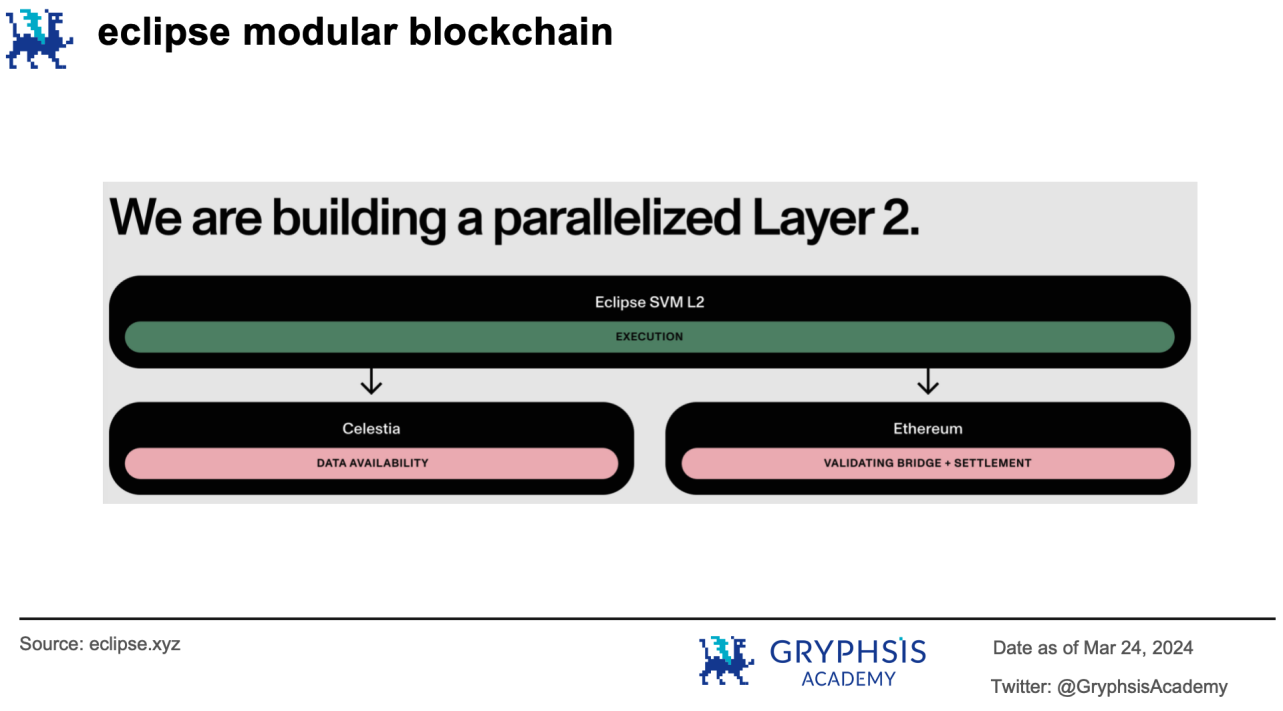
Neon 号称 Solana 生态第一个并行 EVM 项目，开发者可以利用它无缝地将以太坊生态项目迁移到 Solana 生态中来。 Eclipse 是另一个 Solana 生态中兼容 EVM 的最快二层协议，采用模块化架构构建。这三个项目中只有 Neon 发行了自己的代币，并做到了7800多万的流通市值。其他两个项目还处于比较早期的阶段。 Lumio 则结合了 Aptos 和以太坊构建了 optimistic rollup 二层协议，以 Move VM 的速度高效执行以太坊应用。

从融资情况上看， Neon 分别在2021年11月和2023年6月完成4000万美元和500万美元融资，共计4500万美元。 Eclipse 分别在2022年8月、2022年9月和2024年3月完成600万美元、900万美元和5000万美元融资，共计6500万美元。 Lumio 暂未融资。三个项目均未形成规模性的应用生态，但在各主要社交媒体平台都有数万到数十万不等的关注者或成员，有不小的社区活跃度，如下表所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 项目/社媒 | Twitter | Discord | Telegram |
| Neon | 32.7K | 45.5K | 2K |
| Eclipse | 94K | 60K | 6K |
| Lumio | 125.1K | 62.6K | 16.6K |



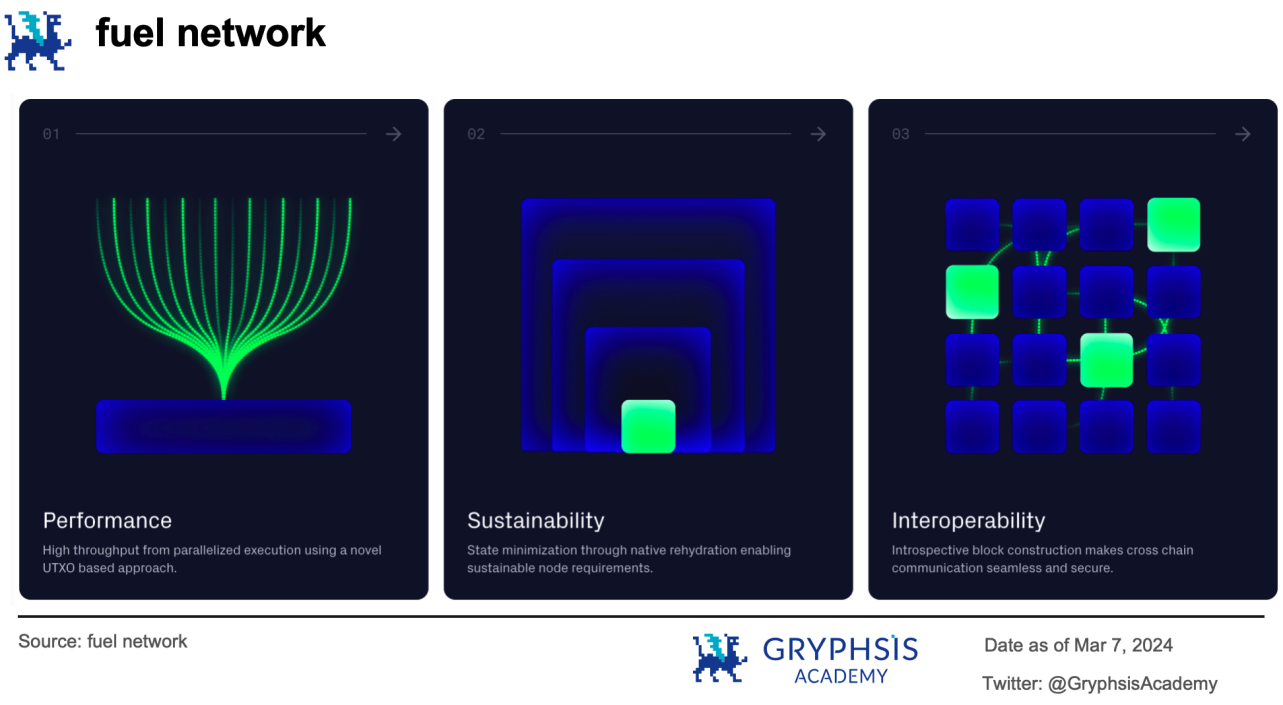
从实现机制上看， Neon 是 Solana 网络上的**EVM 模拟器**，以智能合约的形式运行。开发人员可以使用诸如 Solidity 、 Vyper 这样的语言编写 dApp 应用，并可以使用 MetaMask 、 Hardhat 、 Remix 等以太坊工具链和兼容的以太坊 RPC API 、账户、签名和代币标准等。与此同时享受 Solana 带来的低费率、高交易执行速度以及并行执行的能力。以太坊 dApp 前端发来的以太坊交易经过代理转换生成 Solana 交易，然后在模拟器中执行，修改链上状态。好比我们常在 PC 上使用的游戏模拟器，能让我们在台式机上玩 Switch、PS 等游戏平台上的独占游戏， Neon 能让以太坊开发人员在 Solana 网络上运行以太坊应用。



Eclipse 采取了另一种实现思路：**通过 SVM 执行交易，通过 EVM 结算交易**。 Eclipse 采取模块化区块链的架构，即它只负责交易的执行，而把其他的职责“外包”出去，通过模块化组合形成统一解决方案。比如利用 Celestia 管理数据可用性，利用以太坊执行交易的结算。 Eclipse 利用 SVM 保证了执行速度，通过以太坊的验证和结算保证了安全性。

Lumio 采用的是一种与执行层和结算层无关的设计思路，可支持多种虚拟机，兼容各种 L1/L2 网络： Ethereum、Aptos、Optimism、Avalanche、zkSync ，诸如此类。它**通过 Move VM 执行交易，通过 EVM 结算交易**，这样一来就把以太坊生态和 Aptos 生态连接起来了。然而 Lumio 的雄心并不止步于此，它的愿景是提供跨虚拟机调用，**以最快的速度和最低的费率实现多种区块链流动性的互联**。

### 2.6 模块化 rollup 操作系统： Fuel



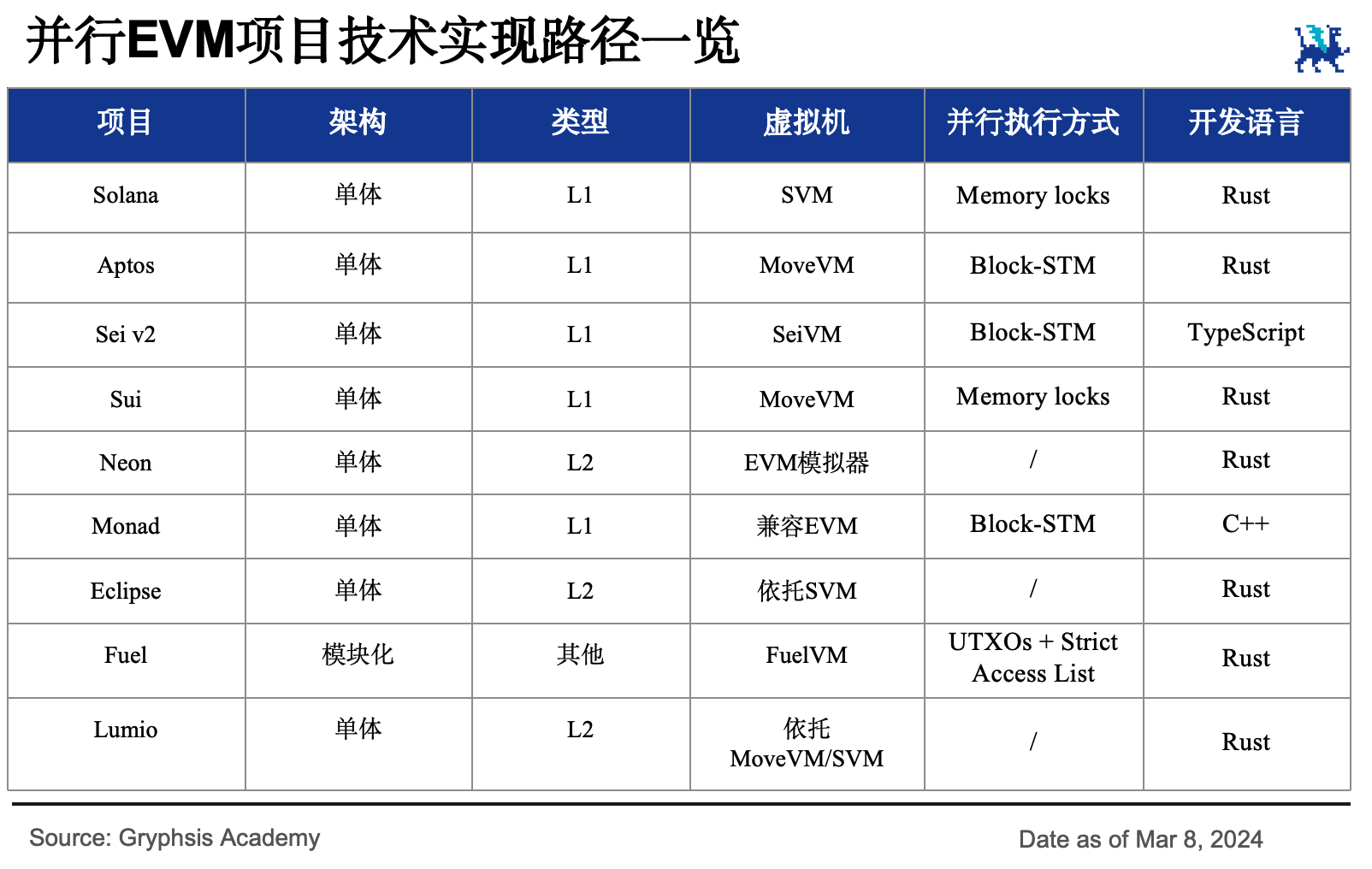
Fuel 由虚拟机 FuelVM 、受 Rust 启发的合约开发语言 Sway 和相关工具链组成，是为以太坊生态量身定制的模块化“ rollup 操作系统”。 Fuel 项目成立于2019年，2020年12月， Fuel Labs 启动了以太坊上第一个 optimistic rollup 执行层 Fuel v1 ，经过3年多的发展，项目终于要在2024年第三季度上线主网。 Fuel 分别在2021年和2022年完成了150万美元和8000万美元融资。核心团队拥有60多名工程师，创始人 John Adler 是数据可用性解决方案 Celestia Labs 的联合创始人，还是 optimistic rollup 方案最早的提出者之一。运营方面，项目在 Twitter 和 Discord 分别有27万和39万成员。

在链上一笔一笔执行交易要支付 gas 费、要争夺宝贵的区块空间，速度还慢。因此自然而然的就会想到各种扩容方案，比如批处理交易的执行，然后再一起打包到链上去结算，加快执行速度。 rollup 就是一种在 L1 之外运行的扩容解决方案，它在链下批量执行交易，然后向 L1 发送交易数据或执行证明，通过 DA 层保证安全性并对交易进行结算。 rollup 有两种主要类型：基于乐观机制（ optimistic ）的和基于零知识证明（ ZK ）的。 optimistic rollup 假设交易是有效的，一旦发现恶意或错误交易就生成欺诈证明交给 L1 回滚处理。 ZK rollup 在不暴露交易细节的前提下通过复杂运算生成交易有效性证明，发布到 L1 ，以证明 rollup 正确执行了交易。所以 rollup 是一种区块链执行层技术。

尽管 rollup 加快了交易的执行速度，但现有的实现大多是针对单体区块链设计的。开发人员不得不在技术上做出各种各样的妥协，这就限制了 rollup 充分发挥其性能。而针对模块化区块链的新趋势，业界又没有适配的 rollup 方案。 Fuel 就是为填补这一空白而诞生的。

Fuel 使用 UTXO 作为数据模型，采用这种数据模型有一个优点：其交易输出只有两种状态，要么已花费，永久记录于区块的交易历史中；要么未花费，可用于未来的交易中。进而做到链上每个节点存储**状态数据最小化**。在此基础上， Fuel 检查每一笔交易访问的账户信息，在执行交易之前找出依赖关系，调度无依赖关系的交易并行执行，提高交易处理的吞吐量。

以上就是目前与并行 EVM 叙事有关的主要项目，如下图所示。



## 3 结论与展望

人们爱把比特币比作“分布式账本”，把以太坊比作“分布式状态机”。如果把运行区块链网络的所有节点看作是一台计算机的话，那么并行 EVM 叙事本质上就是研究如何榨干这台“计算机”的处理器资源，以实现执行速度的最大化。这是计算技术不断发展的历史必然性，就像处理器从单核发展到多核，操作系统从单用户单线程发展到多用户多线程一样。这对于推动行业的不断发展有着非凡的意义。

基础设施的优化会带来更快的速度、更低的费用以及更高的效率。 Web3 创业者可以大胆的进行商业模式创新，为全世界创造更好的去中心化产品用户体验，进一步繁荣行业生态。

而对技术创新的追求也是无止境的。并行 EVM 叙事还只是对处理器资源展开了研究。随着未来对存储资源、网络资源、文件系统资源、设备资源乃至于对“资产”的研究，还会有一波又一波新叙事的不断涌现。

最后，对于 Web3 价值投资而言，仅仅关注技术那是远远不够的。在选择投资标的的时候，投资者不但要看叙事，还要看市值和流动性，要选择“好叙事”、“低市值”和“高流动性”的项目，然后研究其业务（解决什么问题？）、团队背景、经济模型、市场营销、生态项目等方方面面，从而发现潜力项目，找到适合的投资途径。

并行 EVM 是“好叙事”，而且还处在发展的早期阶段。 Solana 、 Aptos 、 Sei 和 Sui 市值已经比较高了。但 Neon、Monad、Eclipse、Fuel 和 Lumio 还处于价值未被充分发现的阶段，值得关注，尤其是 Monad 和 Fuel 。 Fuel 代表着模块化区块链的热门发展方向，而从 Monad 的营销风格来看，不但其本身值得关注，而且未来其生态中的 meme 项目也很值得留意，可能会有因炒作热度产生的暴富神话。

## 4. 参考文献

1. [「并行 EVM 」叙事抬头，哪些项目值得重点关注？](https://www.theblockbeats.info/news/49248)
2. [Technical Deep Dive: Parallel Execution](https://www.binance.com/en/research/analysis/technical-deep-dive-parallel-execution)
3. [virtual machine](https://en.wikipedia.org/wiki/Virtual_machine)
4. [Userspace eBPF Runtimes: Overview and Applications](https://medium.com/@yunwei356/userspace-ebpf-runtimes-overview-and-applications-d19e3c84c7a7)
5. [MT Capital 研报：全面解读并行 EVM，项目一览与未来展望](https://news.marsbit.co/20240119153041358158.html)
6. [区块链不可能三角：终极指南](https://medium.com/@NervosCN/%E5%8C%BA%E5%9D%97%E9%93%BE%E4%B8%8D%E5%8F%AF%E8%83%BD%E4%B8%89%E8%A7%92-%E7%BB%88%E6%9E%81%E6%8C%87%E5%8D%97-85c069f21adc)
7. [2024 年将会是并行EVM之年？并行EVM叙事分析与展望](https://news.marsbit.co/20240112161732277166.html)
8. [Layer 1 vs. Layer 2 vs. Layer 3 Blockchains Explained](https://blog.ambire.com/blockchain-layers-differences-explained/)
9. [Sealevel — Parallel Processing Thousands of Smart Contracts](https://medium.com/solana-labs/sealevel-parallel-processing-thousands-of-smart-contracts-d814b378192)
10. [The Aptos Blockchain: Safe, Scalable, and Upgradeable Web3 Infrastructure](https://aptosfoundation.org/whitepaper/aptos-whitepaper_en.pdf)
11. [All About Parallelization](https://blog.sui.io/parallelization-explained/)
12. [Sei v2 - The First Parallelized EVM Blockchain](https://blog.sei.io/sei-v2-the-first-parallelized-evm/)
13. [Parallel Execution](https://docs.monad.xyz/technical-discussion/execution/parallel-execution)
14. [Neon EVM doc](https://docs.neonevm.org/docs/about/why_neon)
15. [Eclipse Offical Site](https://www.eclipse.xyz/)
16. [Lumio doc](https://docs.lumio.io/xlumio-virtual-machines#execution)
17. [The Rollup OS for Ethereum](https://fuel.mirror.xyz/uYMT39LiB538Q61Mgf4bRrsAwApJSeNiG_afDHhOhGs)
18. [Fuel Official Site](https://fuel.network/)