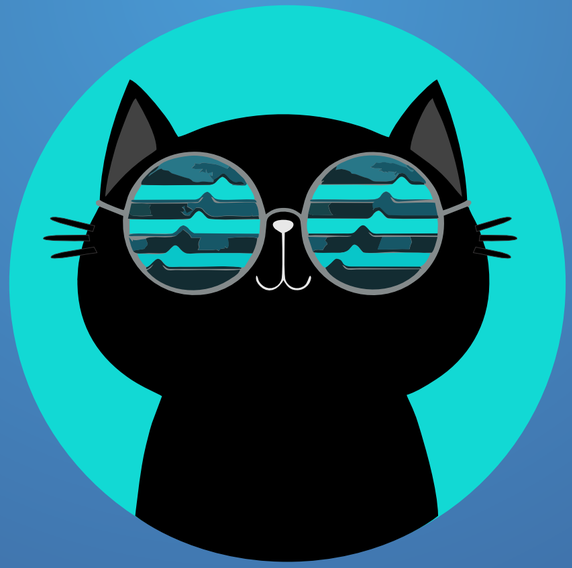
MAU

Understanding Aptos

By MAU protocol



@MAUtoken @Mau_Token

mauprotocol.com

Introduction

Aptos - Originating from Ohlone Native American word for the **“the people”**

Aptos is a new Layer-1 blockchain built to bring **Web3** to the masses. It is designed from the ground up to make developing on a blockchain easy and seamless.

The main issue **Aptos** addresses are blockchains suffer frequent outages, have high costs, low limits and many security concerns. Aptos solves these through a design which focuses on safety, scalability, upgradeability, and reliability.

Aptos is a layer-1 Proof of sake (PoS) Blockchain built upon **Move** programming language. Move was developed by Facebook's blockchain division from Diem. They pioneered social media and now they are pioneering Defi. Aptos eventually cut ties with Meta in March 2022 becoming an independent body and proceeded to raise to raise $200 Million in funding.

Aptos has been built for adoption! Theoretically it can reach 160,000 transactions per second whilst maintaining security and reliability. Aptos is perceived as more than reliable in comparison to it's rival Solana which has been prone to failures due to outages and downgrades.

	Aptos / Move	Solana / SeaLevel	EVM
Data storage	Stored within the owner's account	Stored within the owner's account associated with a program	Stored within the account associated with a smart contract
Parallelization	Capable of inferring parallelization at runtime within Aptos	Requires specifying within the transaction all accounts and programs accessed	Currently serial nothing in production
Transaction safety	Sequence number	Transaction uniqueness + remembering transactions	nonces, similar to sequence numbers
Type safety	Module structs and generics	Program structs	Contract types
Function calling	Static dispatch not on generics	Static dispatch	Dynamic dispatch

Understanding Parallel Transactions

Parallel computation can also be accomplished through parallel execution, which involves running independent transactions simultaneously. Transactions are considered independent if they do not read from or write to the same data. Regardless of the order in which the blockchain executes the transactions, the outcome will remain consistent. It is possible to conduct trades using multiple CPU cores or GPUs securely and concurrently.

The use of a resource model enhances the viability of parallel execution. Instead of storing data as variables within the contract account, Solana, Flow, Aptos, and Sui adopt a different approach and store their data in distinct storages under users' accounts, accounts, or objects. This reduces the likelihood of data access malfunctions and significantly improves the efficiency of parallel execution.

Despite sharing the resource model as the basis of their architecture, these blockchains exhibit some nuanced differences in their operations. Specifically, Solana and Sui demonstrate similarities in their transaction execution processes, albeit data is stored as accounts in Solana and as objects in Sui. To interact with the desired accounts or objects, dApps must identify them in their function calls, prompting the VM to conduct an analysis and identify the independent transactions.

Aptos, on the other hand, adopts an innovative strategy by utilizing a variant of Software Transactional Memory (STM) called Block-STM. In this approach, the transactions within each block are arranged in a predetermined sequence and allocated to processor threads for execution. To optimize performance, the blockchain operates under the assumption that there are no dependencies between the nodes and executes all transactions, while keeping track of the memory locations altered by each transaction. Following the validation of all transaction results, any transactions discovered to access memory locations modified by prior transactions are invalidated. The failed transaction is then re-executed, and the cycle is repeated until all transactions are executed.

In contrast to Solana and Sui, **Aptos** offers a more seamless developer experience as there is no requirement to indicate the data to be accessed in a transaction. **Aptos** reports suggest that Block-STM and other mechanisms can significantly enhance cross-node communications, allowing the network to achieve an impressive 160K TPS.

Blockchain	Ticker	Time to finality	Max TPS
Solana	SOL	2.34s - 46s (different tests)	120.000 (710.000 on a 1 GB network)
Aptos	N/A	less than 1s	160.000
Avalanche	AVAX	0.15 (record) 1.3-3.4s	4.500 per subnet
Internet Computer	ICP	1-2s	11.500
Fantom	FTM	1s	4.500
Ethereum	ETH	78s (6 confirmations)	45
Terra	LUNA	6s	10.000
Bitcoin	BTC	60m (6 confirmations)	7
BSC	BNB	75s	160

Move basic's

Move has been specifically designed for blockchain applications and addresses multiple unique challenges faced by programmers building with alternate languages.

Move has been designed with safety and security and that front of mind. The language includes several built-in features that help prevent common errors which can lead to security vulnerabilities, such as. Integer overflow and divide-by-zero errors.

Move also has a formal verification system that allows developers to mathematically prove the correctness of their code.

Move has also introduced “resource types”. Resource types represent unique assets on the blockchain which allow developers to define the properties and behaviours of these assets in a way that is secure and consistent across the blockchain.

Move is also designed to be a flexible and extensible language! It includes a modular architecture that allows developers to build and reuse libraries of code. Alongside this there is a bult in package manager for controlling dependencies.

Overall, **Move** has been designed to deliver a safe, scalable, and secure language to build applications on the blockchain whilst reducing vulnerabilities.

ChainCastle ← Back ENGLISH ▼

Chapter 3 Variable binding and data type

Variable binding

Variable binding refers to binding some values to a variable so that they can be used later.

Similar to Rust, Move uses `let` to bind variables, which follows the format of `let variable: variable type;` . For example,

```
let x: u64;
```

After variable binding, we can use `=` to initialize the variable. For example,

```
x = 5;
```

It is worth noting that variables should be bound at first in any process. In other words, all `let` lines should be put at the very beginning; otherwise, inserting any `let` line within the code block will lead to the compilation error.

Data Types

At present, Move supports the following data types:

- boolean
- uint64
- address
- bytes
- struct
- resource (a unique data type in Move)

CONTRACT.MVIR

```
modules:
module BuildCastle {

}

script:
import Transaction.BuildCastle;
main() {
    // Enter the code here

    return;
}
```

HINT

```
modules:
module BuildCastle {

}

script:
import Transaction.BuildCastle;
main() {

    let SerialDigits: u64;
    SerialDigits = 8;
    return;
}
```

HIDE ANSWER CHECK ANSWER PREVIOUS 3 / 10 NEXT

@MAUtoken @Mau_Token mauprotocol.com

The Role of MAU

Historically, Layer-1 chains have greatly benefited from a flagship meme. The role of this is to encourage adoption, spread awareness of the chain through a light-hearted, fun, and accessible medium to communities. Binance smart chain enjoyed Doge, whilst Ethereum had Shiba. Considering Aptos's great technological advancements, we saw the perfect opportunity to challenge status quo and stand out! That's why **MAU** is here. **MAU** is the friendly feline built to fuel **Aptos** adoption and foster a culture of creativity.

Come and join the **MAU** community and enjoy collaborating with other members passionate about **Aptos** and **Web3**.

https://t.me/MAUtoken

Remember, Aptos is “The People”

@MAUtoken @Mau_Token mauprotocol.com