BINANCE RESEARCH

Move: The Next Step in Smart Contract Programming?

Shivam Sharma

Table of Contents

Key Takeaways

- Move is a new smart contract programming language. Originally designed at Meta to power the Diem blockchain, the language is now being implemented by two of the newest projects in the L1 space: Aptos and Sui

- The smart contract language landscape is currently dominated by two main players: Solidity and Rust. To start this report we provide an overview of the current L1 market and their smart contract language choice, as well as, discuss the reasons as to why we need a new smart contract programming language in the first place

- We look more closely at Move in the second section of this report, starting with the genesis of the language. We then go through some of Move's key features, including the focus on user-defined assets, its various safety enhancements, as well as its provisions for developer flexibility

- The team behind Move use a two-pronged approach for their developer onboarding strategy, including targeting those developers who are yet to enter Web3. We further discuss the perceived network effects of Solidity and the EVM and whether they might be overstated

Introduction

Move is a new smart contract programming language that has been rapidly gaining attention in the past few months. Originally developed at Meta to power the Diem blockchain, Move was designed as a **platform-agnostic language** that could be implemented in blockchains of varying architecture. For example, Aptos, a recently launched Layer-1 ("L1") project utilizes a classic version of Move ("core Move"), while Sui, an upcoming L1, will look to implement their own version, dubbed "Sui Move".

Move is designed to **write smart contracts in a safe and flexible manner** and aims to provide a **developer-friendly alternative to the current options, primarily Solidity and Rust**. In this report, we describe the current state of the market, answer the question of why we even need another smart contract programming language, and provide a thorough overview of the key features of the Move language.

Current State of the Market

The smart contract language landscape is currently dominated by two main players: **Solidity and Rust**.

Solidity, developed by several Ethereum core contributors, is an **object-oriented, high-level programming language** that is used to write and implement smart contracts on **Ethereum** and other **Ethereum Virtual Machine ("EVM") compatible blockchains**.

The language was initially proposed in 2014 by Gavin Wood (ex-Ethereum CTO and founder of Polkadot and Kusama), launched in 2015, and is widely considered to be the first language developed for the sole purpose of deploying smart contracts. Solidity cites JavaScript, C++ and Python as primary influences.

Rust, originally developed within software company Mozilla, is a **multi-paradigm, general-purpose, low-level programming language** used to write smart contracts and build decentralized applications ("dApps") on **Solana**, and various other blockchains such as **NEAR Protocol and Polkadot**.

Unlike Solidity, Rust was not specifically created for smart contract development, and had been in existence for multiple years prior to the launch of Solana. Rust also cites C++ as a major influence. Rust itself is not an executable language, so technically speaking, developers do not write smart contracts in Rust, but rather via a software development kit ("SDK").

Solidity developers can write and deploy smart contracts on the two largest L1s in the space; Ethereum and BNB Chain. Together, these chains represent around a quarter of total crypto market cap (~US$200B out of ~US$800B at the time of writing). BNB Chain is the largest among the EVM-compatible chains, which also include Avalanche, Fantom and Polygon. Meanwhile, Rust devs can deploy smart contracts on the likes of Solana, Polkadot and Cosmos. **Weekly commits / active dev numbers for these Rust-based blockchains is notable** and could be seen as positive, given that dev statistics are generally a forward-looking metric. We also see **Haskell in the mix, which Cardano uses** (via its native Haskell-based smart contract language, Plutus). Lastly, we have **Move, with Aptos and Sui being the biggest names to be using this language** at present. Dev figures are decent, and not far from the likes of Avalanche.

Figure 1: Solidity and Rust dominate the landscape

Project		Market Cap ($B)	Weekly Commits	Weekly Active Devs	Primary Smart Contract Language(s)
	Ethereum	148.8	7,437	1,496	Solidity
	BNB Chain	43.8	760	71	Solidity
	Cardano	11.1	510	104	Haskell
	Polkadot	6.5	9,716	707	Rust / ink!
	Solana	5.1	1,388	259	Rust
	Avalanche	3.9	242	35	Solidity
	Cosmos	2.9	2,182	446	Rust / Ethermint
	NEAR	1.6	1,407	110	Rust / JavaScript
	Aptos	0.5	56	23	Move
	Sui	n/a	59	24	Move

Source: CoinMarketCap / Gokustats

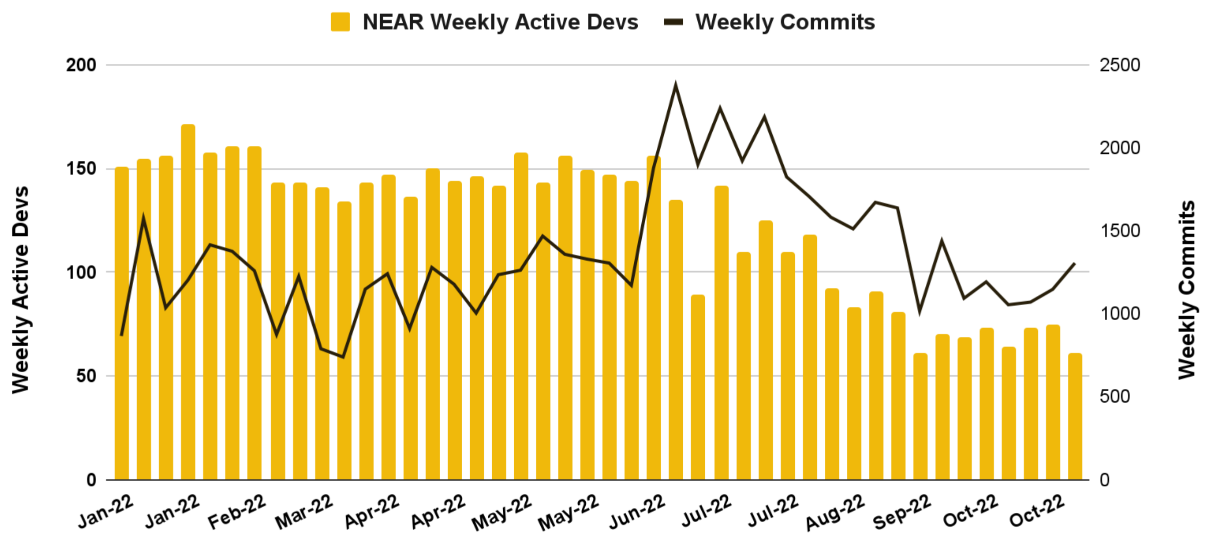
The Role of Software Development Kits

Software Development Kits are integrated kits of software development tools that help developers in building blockchain solutions on a given platform. SDKs usually include tools such as programming language compilers, code libraries, testing / analytical tools, database development environments, debuggers etc. One or many different Application Programming Interfaces ("APIs") will also generally feature in a SDK, as APIs help facilitate communication between the different tools mentioned previously.

Broadly speaking, we can think about **two groups** of SDKs. On one hand, we have **solutions like the Cosmos SDK, which is an open-source framework for building blockchains themselves**. These chains are generally referred to as application-specific blockchains and there are many notable examples of chains built in this manner, including the original Binance Chain in 2019.

On the other hand, we have **tools like the JavaScript SDK, released earlier this year by the NEAR Protocol**(1). This tool kit essentially allows JavaScript developers to build dApps on the NEAR Protocol, without learning Rust. While this sounds like it could be a significant step forward, particularly considering that there are approximately 16.5m active JavaScript developers relative to around 18K Web3 developers at this time, numbers for NEAR Protocol have remained relatively stable and in a slight downward trajectory since the SDKs' launch. While it is still early, and this is just one example, there is a thought in the community that to genuinely develop a high-quality and safe dApp, you should really be working directly in Rust, rather than via the SDK. Nonetheless, this is a developing area within the smart-contract language debate and one that should be considered when thinking about the topic.

Figure 2: NEAR Protocol developer numbers have been largely stable for a few months

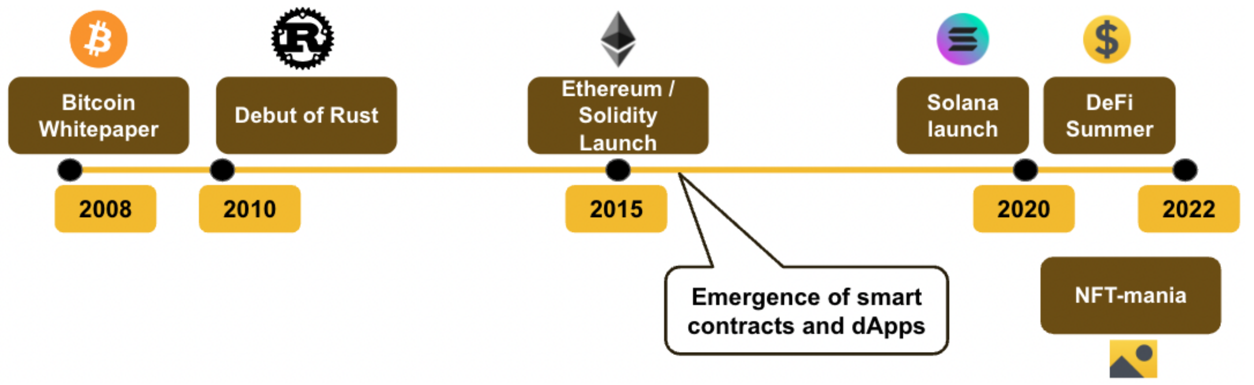


Source: Gokustats

Why do we even need another smart contract programming language?

Let's set the scene. As we can see in Figure 3, the origins of the story lie in the first major event in modern crypto space - the **release of the Bitcoin whitepaper on 31 October 2008**(2). In our extremely abbreviated timeline, the next major step change in the crypto world would be the advent of smart contracts through the **launch of Ethereum and Solidity in 2015**. Note that the **Rust language first debuted in 2010**, however, this was irrelevant to the nascent crypto industry at the time. Rust was launched as a general-purpose language and had been in development since 2006 - almost 10 years before the world knew anything about smart contracts.

Figure 3: A very abbreviated timeline of the crypto space



Source: Binance Research

The launch of Ethereum / Solidity is essentially the origin story of dApps in the crypto world, and in addition to opening up a whole new dimension for creators and builders, it also opened up a new avenue for hackers and exploiters. **Just over a year after launch, the young Ethereum community faced a tremendous test; "The DAO" hack.** The DAO was a decentralized autonomous organization ("DAO") - the first of its kind - and was launched in 2016 on Ethereum. After raising ~US$150m through a token sale, The DAO was hacked due to vulnerabilities in the code and ultimately, the Ethereum blockchain went through a contentious hard fork which resulted in the divergence of the network into two separate chains: Ethereum Classic and Ethereum.

Why is this relevant to us? Because the hack involved a **reentrancy exploit**, which is possible due to the way that code is structured in Solidity. This was not the only reentrancy exploit experienced on an EVM-chain, with numerous multi-million dollar examples of similar attacks across the years(3). **The Move language does not demonstrate the same vulnerabilities to reentrancy exploits (explained in more detail in the Move section), which should hopefully have a positive effect on the number of exploits experienced on chains that utilize the language.**

Fast-forward to 2020, and we see the relatively quiet launch of Solana, followed by the famous DeFi Summer, which was kickstarted by the liquidity mining program of Ethereum-based dApp - Compound Finance. After this period of DeFi-focused activity and increasing market euphoria through 2021, we saw a period of NFT-mania with some blue-chips selling for millions of dollars, all the while garnering significant media and Hollywood attention (Anyone remember Jimmy Fallon telling us about his Bored Ape earlier this year?).

So what are we implicitly saying by continuing to build this new industry largely using Solidity and Rust? That the two best, most suitable and safest languages to take forward this US$1tn+ industry are Rust (a general-purpose language originally developed before the Bitcoin whitepaper) and Solidity (launched at a time before anyone understood the true possibilities of smart contracts and dApps)? Even if that is the case, we would argue that it is worth learning about the latest player to join the race and develop an understanding of some of the latest cutting-edge features they may bring to the table. Here comes Move...

"Are we really saying that the two best, most suitable, and safest languages to take forward this US$1tn+ industry are Rust (a general-purpose language developed before the Bitcoin whitepaper) and Solidity (launched before anyone understood the true possibility of smart contracts and dApps)?"

Move

"**Move is a language for programming with scarcity**" is how Sam Blackshear, the creator of Move, puts it. Originally created for Meta's Diem (formerly Libra) project, **Move is a platform-agnostic, Rust-based programming language for implementing safe and flexible smart contracts and custom transactions.**

Genesis

When at Meta, Sam and his team took a close look at the options available to them to build the Diem blockchain with, and they realized a couple of key things.

1. Looking at the type of programs that developers are trying to write, they are all to do with assets. **Yet, when we look at the conventional programming languages that developers are using - there is no concept or vocabulary to describe assets.** There is essentially no type nor value nor representation of an asset in these languages. Looking at **Solidity for example, the way you represent assets is through hash tables and bytes -** this makes it very hard to do the things you want to do with smart contracts and involves lots of tweaking and customisation, creating complexity and thus vulnerabilities that could be exploited

2. They realized that if you are going to build using Solidity / EVM for example, **because of the features that are natively baked into the languages e.g. transaction format or address format, you will inherit many inherent limitations and operate somewhat similarly to the EVM.** The team behind Move wanted their language and virtual machine ("VM") to be much more barebone, minimal and platform-agnostic. The team recognized that this is a rapidly growing industry and technology, and builders should have more flexibility and be able to experiment cross-platform

With this in mind, the team decided that building their own smart contract programming language was a better choice than to utilize existing options, especially for the long term. The purpose of Move was to build a language centered around the idea of programming with scarcity, while providing a structured representation of assets in a safe environment.

Key Features

First-Class Resources

As previously stated, **Move has been designed very specifically to deal with digital assets**. One of the **novel features of Move is the ability to define custom resource types**, taking inspiration from linear logic i.e. the idea that "a resource can never be copied or implicitly discarded, only moved between program storage locations"(4).

What does this mean in practice?

The concept of resources **allows developers to encode safe, yet customizable assets (i.e. coins, tokens, NFTs etc)**. Developers can define any variable through combinations of four distinct attributes: Copy, Key (index), Store and Drop (discard). After the variable is declared as a Resource, it is only editable by Key and Store, and not Copy nor Drop i.e. the Move language syntax itself ensures the scarcity of a resource.

Again, to reiterate, this means that Move lets developers work directly with well-defined assets which correspond to the scarcity characteristics of physical assets. This is in contrast to Solidity / EVM for example, whereby assets are represented as entries in hash maps, with asset updates working through updating entries in a map (instead of simply passing an asset over like in Move). Even to those not completely familiar with programming languages, it should hopefully be clear that the Solidity / EVM method seems to add a level of complexity and abstraction that does not feature in the Move / Move VM method of representing assets.

Safety

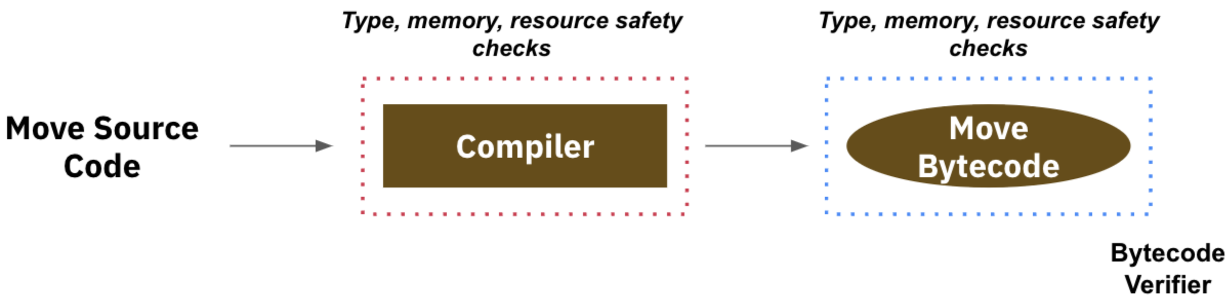
- **Bytecode Verifier -**
  - Move's bytecode verifier is where the bytecode gets analyzed before execution. This includes tests for universal safety properties like type safety, memory safety and resource safety. It is a **set of lightweight tests that are run on-chain that enforce general safety properties that must hold for any well-formed Move program.** No Move program / smart contract can be executed before passing through the bytecode verifier
  - Similar checks are also performed in the source code compiler. However, it is

beneficial to perform these checks on the bytecode itself too, because (1) the compiler is an expensive and large piece of software and better run off-chain, and (2) the bytecode verifier prevents someone from writing bytecode manually to get around the safety features in-built into Move

➢ This essentially **eliminates the compiler from the trusted computing base.** This means that the developers working in Move do not have to worry about possible failures or attacks in compilers. Given the compiler is much larger than the bytecode verifier, this could be a significant reduction in attack surface area

➢ Similar concepts exist in the Java VM and Common Language Runtime (the VM component of Microsoft's .net)

Figure 4: Move's Compilation includes checks on both the source code and the bytecode


Type, memory, resource safety checks
Type, memory, resource safety checks
Move Source Code
Compiler
Move Bytecode
Bytecode Verifier

Source: Binance Research

❖ **The Move Prover -**

➢ Move also introduces a **formal smart contract verification tool** - the Move Prover ("MVP"). Formal verification is the process of verifying the reliability and correctness of a program with respect to a formal specification

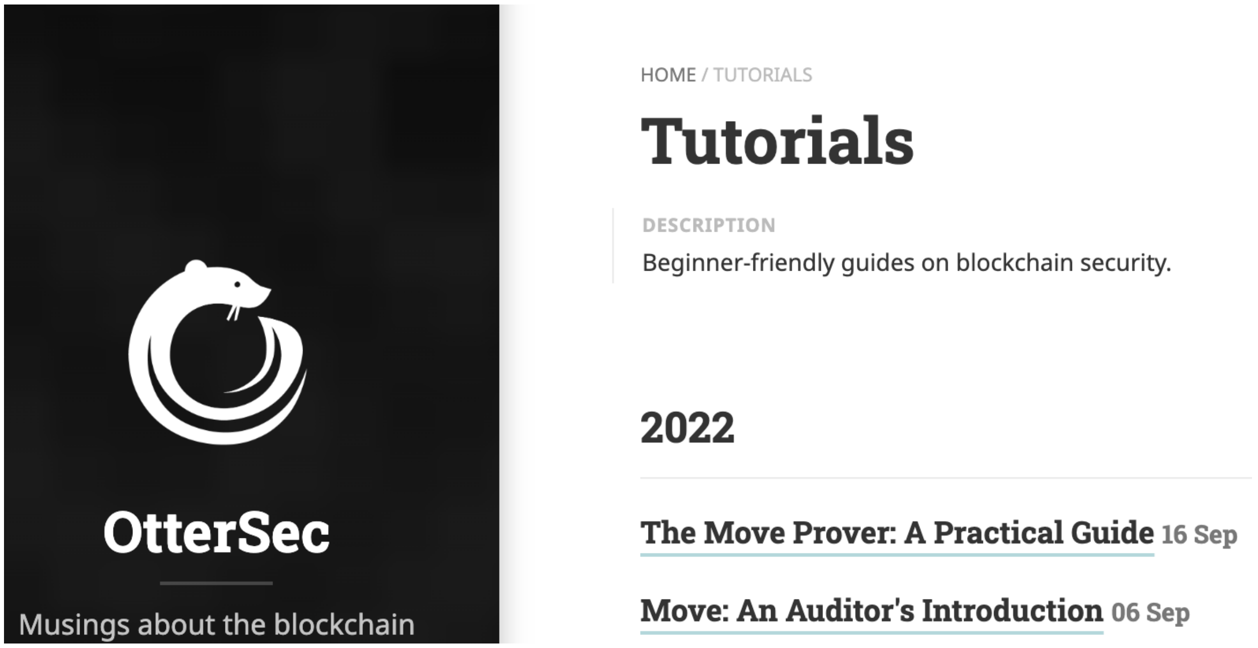
➢ The MVP was very closely developed alongside the Move language and allows developers to mathematically verify their code in an automatic and efficient fashion. The **MVP runs off-chain and checks user-defined and application-specific safety properties**

➢ Similar to the bytecode verifier, the MVP also analyzes the bytecode directly, and thus also benefits from the additional safety check this provides. The **MVP provides more sophisticated and inter-procedural, cross-module checks, when compared to the more isolated focus of the bytecode verifier**

➢ In terms of comparisons, there is some work around Solidity smart contract verification, but due to properties like dynamic dispatch (discussed more below), this is a much more difficult task in Solidity, when compared to Move

- ➢ One interesting point is that the **inclusion of the MVP might help to reduce the audit burden of smart contracts relative to other languages**, where formal verification might not be possible / harder e.g. Solidity. For example, developers can use the Move Prover to specify and verify the properties that are most important, and the auditor's job is reduced to making sure that the specifications are strong and make sense and the Prover will do the rest. While this certainly will not eliminate the need for auditing, it can be an interesting time saver, particularly given how lengthy development cycles traditionally are

Figure 5: Move's unique safety features might help reduce the scope of smart contract audits



HOME / TUTORIALS

Tutorials

DESCRIPTION

Beginner-friendly guides on blockchain security.

2022

The Move Prover: A Practical Guide 16 Sep

Move: An Auditor's Introduction 06 Sep

Source: OttserSec

❖ **No Dynamic Dispatch -**

- ➢ **Move as a language is designed to not support dynamic dispatch.** Technically speaking, this means that the code execution logic will be determined at compile time (statically) instead of at runtime (dynamically)
- ➢ For example, **in Solidity, when contract 1 calls contract 2's function, contract 2 can run code that was unanticipated by contract 1's developer, which can lead to re-entrancy vulnerabilities** (contract 1 accidentally executes contract 2's function to withdraw money before actually deducting balances from the account). Reentrancy exploits (as mentioned above) have been some of the

most notable category of hacks that have occurred in EVM chains over the last few years

➢ This type of **reentrancy exploit is not possible when using Move due to the static nature of the language**. Additionally, it **also makes it easy for verification tools to more precisely consider the effects of a procedure call without performing a complex call graph construction analysis**. This helps in the verification ability of Move code and could be another factor helping reduce the audit burden of Move smart contracts

Flexibility

One of the key features of the language that Sam and the team have stressed is **the relative minimalism of it**. For example, **Move does not have accounts or any sort of native tokens (or any token at all for that matter) or cryptography baked into the language**. The language is intentionally minimal so that it can be used in a cross-platform way. Developers should be able to use Move in whatever blockchain they want and make their own decisions on what their transactions will look like, what consensus mechanism they will use, what type of cryptography they will use etc.

The team behind Move recognize that crypto is an emerging space and want to **provide developers with the flexibility they need to experiment and find the best solutions for their vision**. They also recognize that this is a great way to develop a meaningful cross-platform community for the language, which will ultimately be the most important factor when thinking about the medium to long-term success of it.

One example we already have of this is the difference in implementation by the two biggest projects using the language so far. While **Aptos utilizes so-called "core Move", Sui uses "Sui Move"(6).** Sui Move has a few technical differences from some of the core Move design choices, which allow Sui to fully leverage the flexibility of the language and take advantage of the object-oriented architecture of the Sui L1. This further illustrates how the Move team has designed the language so that it can be tailored to the specific needs of different projects.

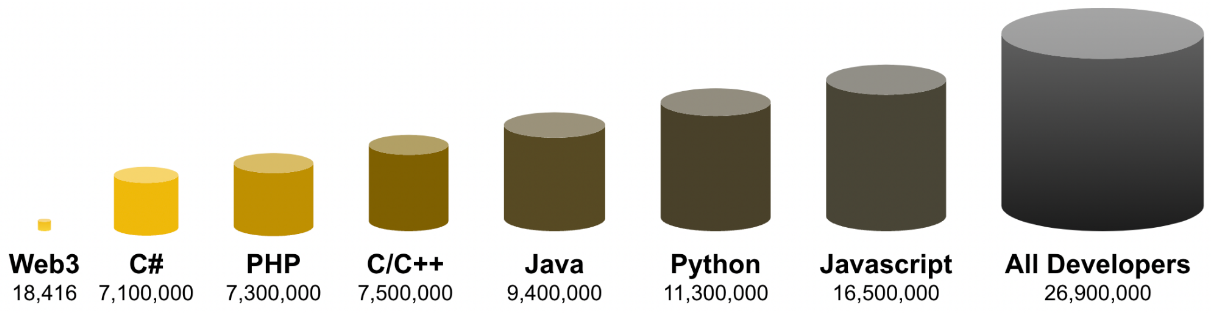
Move's developer strategy and the EVM's network effects

The team behind Move are targeting a **two-pronged approach** in terms of their developer strategy. **In the short-term, they aim to attract developers familiar with other smart contracting platforms.** Some of the more curious and well-informed of this group are already aware of Move and have perhaps already experimented with the language. Both Aptos and Sui have been active in organizing events, AMAs, hackathons, developer calls etc. to help increase the visibility of the language.

In the medium to long term, their primary goal is to grow the proverbial pie through attracting those that have not touched blockchain programming before. The idea behind Move is that if you are a product builder, you do not have to be an expert in smart contracts nor blockchain in order to pick up the language and write quality Web3 code. As a new developer, you do not have to worry about reentrancy nor dynamic behaviors and can try your hand at Web3 with fewer concerns. **Move wants to minimize the possible attack surface with its unique properties and make it friendly and accessible for those developers coming from Web2 programming to adjust to Web3.** If some things are fundamentally tricky about smart contracts, Move aims to take those away and help the new developer with the verifier, the MVP and the simple user-defined asset-focus of the language to guide them to the right path and onboard them into the crypto world.

One thing we should discuss here is the **commonly cited idea among many in Crypto Twitter; that the network effects of Solidity and the EVM are insurmountable due to how many of the top blockchains utilize it.** This is a frequent argument against the introduction and chances of success of newer languages like Move. However, looking slightly deeper into the facts and the picture becomes more muddy. As we can see in Figure 6, **Web3 developers make up a tiny fraction of the overall market, with some reputable estimates showing only < 0.1% of the overall developer market active in the space**. From this number, 4-6k are generally estimated to be active Solidity / EVM developers. When we look at this number and compare with the amount of developers there are overall, we can definitely see how a new smart contract language has the room to grow and expand as the industry itself grows. It further highlights how the network effects of the Solidity / EVM ecosystem might be somewhat overstated and how new languages like Move definitely have opportunity for market share.

Figure 6: Web3 Developers vs. other coding languages



Source: Electric Capital, Binance Research

The idea that smart contract programmers will get better on their own using the same languages as always makes little sense - when more developers enter the market, the average level of knowledge goes down, not up. **Given it is not reasonable to expect new developers to be more informed or technically proficient, one thing we can do from our side is to bring in safer languages, better tooling and bake in more safety checks at the base level.** This is what Move aims to do.

"The idea that smart contract programmers will get better on their own using the same languages as always makes little sense - when more developers enter the market, the average level of knowledge goes down, not up. "

Current Usage of Move

The current usage of Move is naturally relatively limited, as would be expected from a new programming language. At present, Move is utilized by:

- **Starcoin -** a proof-of-work blockchain which was the first public blockchain to support Move
- **Aptos -** one of the major new players in the L1 space. Aptos' mainnet launched in October 2022
- **Sui -** a notable upcoming L1 project with mainnet expected late 2022 / early-2023
  - You can check out **our latest report on Aptos and Sui** here to learn more about the L1 projects and what they aim to bring to the table
- **0L -** a fork of the original Diem blockchain
- **Celo -** a partnership between Celo and Mysten Labs was announced in September 2021(5) which would add support for Move to the Celo L1. The support is likely to be deployed after Sui's mainnet launch

Outlook

When thinking about introducing a new smart contract programming language, there are essentially two high level areas to think about. The **first part is the design - how does it work? What is it designed to do? Is this a major improvement from the other options in the market?** This first part can be tackled without too much battle-testing and is something that the team behind Move is confident in. The development of the language has been worked on by some of the brightest engineers and developers in the space, owing to Move's roots in Meta. The **next major area to be concerned with is the actual implementation and battle testing - that can only be done with time.** Move's founding team talks about their 'second mover' advantage over earlier smart contract languages and believe that despite the huge inertia there is around existing ecosystems, given some time and an opportunity for Move-based chains to showcase themselves, they can win some market share. In particular, **their focus on targeting those developers that may not know much about crypto or blockchain and trying to simplify their onboarding could be an important catalyst for them.**

Another point worth noting is that the team at **Move stresses that mainstream smart contract languages do not currently satisfy all of the requirements that are needed from a smart contract language in today's market**. As we discussed previously, Solidity was founded and launched before anyone understood the real possibilities of smart contracts and dApps, while Rust is a general-purpose language developed before the Bitcoin whitepaper. Yes, there are methods and ways to tweak things in these languages to address their shortcomings, but ultimately every time you do this you take something very complicated and add further restrictions to it. Often these tweaks mean that a developer will not be able to leverage existing tooling, further restricting experimentation and flexibility in this emerging space. **In this setting, where we care so much about correctness and safety, simplicity is crucial**. Thus, the Move team believes that it makes more sense to build and use something that has the desired properties you want for the future, rather than repeatedly modifying existing tools which might be fundamentally unsuitable for the job.

It will be very interesting to see if Move can continue to generate buzz within the community and how much developer onboarding the language can achieve in the medium term. We will keep a close eye on how things progress and continue to report on the latest developments.

References

1) https://near.org/blog/near-releases-javascript-sdk-bringing-web3-to-20-million-developers/
2) https://bitcoin.org/bitcoin.pdf
3) https://github.com/pcaversaccio/reentrancy-attacks
4) https://diem-developers-components.netlify.app/papers/diem-move-a-language-with-programmable-resources/2020-05-26.pdf
5) https://docs.sui.io/learn/sui-move-diffs
6) https://www.businesswire.com/news/home/20210921006104/en/Celo-Sets-Sights-On-Becoming-Fastest-EVM-Chain-Through-Collaboration-With-Mysten-Labs

About Binance Research

Binance Research is the research arm of Binance, the world's leading cryptocurrency exchange. The team is committed to delivering objective, independent, and comprehensive analysis and aims to be the thought leader in the crypto space. Our analysts publish insightful thought pieces regularly on topics related but not limited to, the crypto ecosystem, blockchain technologies, and the latest market themes.

Shivam Sharma, Macro Researcher

Shivam is currently working for Binance as Macro Researcher. Prior to joining Binance, he worked as an Investment Banking Associate / Analyst at Bank of America on the Debt Capital Markets desk, specializing in European Financial Institutions. Shivam holds a BSc Economics degree from the London School of Economics & Political Science (“LSE”) and has been involved in the cryptocurrency space since 2017.

Read more
https://research.binance.com/en/analysis

Share your feedback
https://tinyurl.com/bnresearchfeedback