Arch Network

Arch.Network

Arch: Introducing Bridgeless Bitcoin Programmability

Arch Network Whitepaper

Whitepaper

August 2024

Abstract

Arch is a parallel execution layer that enhances Bitcoin's capabilities by enabling fast, secure, and fully-verifiable smart contracts. It creates a bridgeless platform for developers to build Turing-complete smart applications on the base layer. This enables role-dependent trust assumptions for Bitcoin users, including fully trustless use cases.

Unlike L2s and meta-protocols where users must bridge their assets first and then send the assets to a smart contract, Arch allows users to send their assets directly to the smart contract using a native UTXO-based Bitcoin transaction without bridging. This design offers more security and utility than L2s which fracture liquidity and add bridging risks in return for scalability benefits. It also allows for greater interoperability among emerging decentralized apps being built on Bitcoin, from DeFi and gaming to prediction markets and social networks.

It accomplishes this through three components. The specialized Arch VM handles state changes and complex off-chain computations, while the PoS Verifier Network decentralizes and incentivizes infrastructure and security. Lastly, a novel implementation of threshold signature schemes ensures that the network will only execute actions if a majority of decentralized verifiers have approved it. This architecture creates the type of conditional execution necessary for enabling programmability, without removing assets from the Bitcoin base layer.

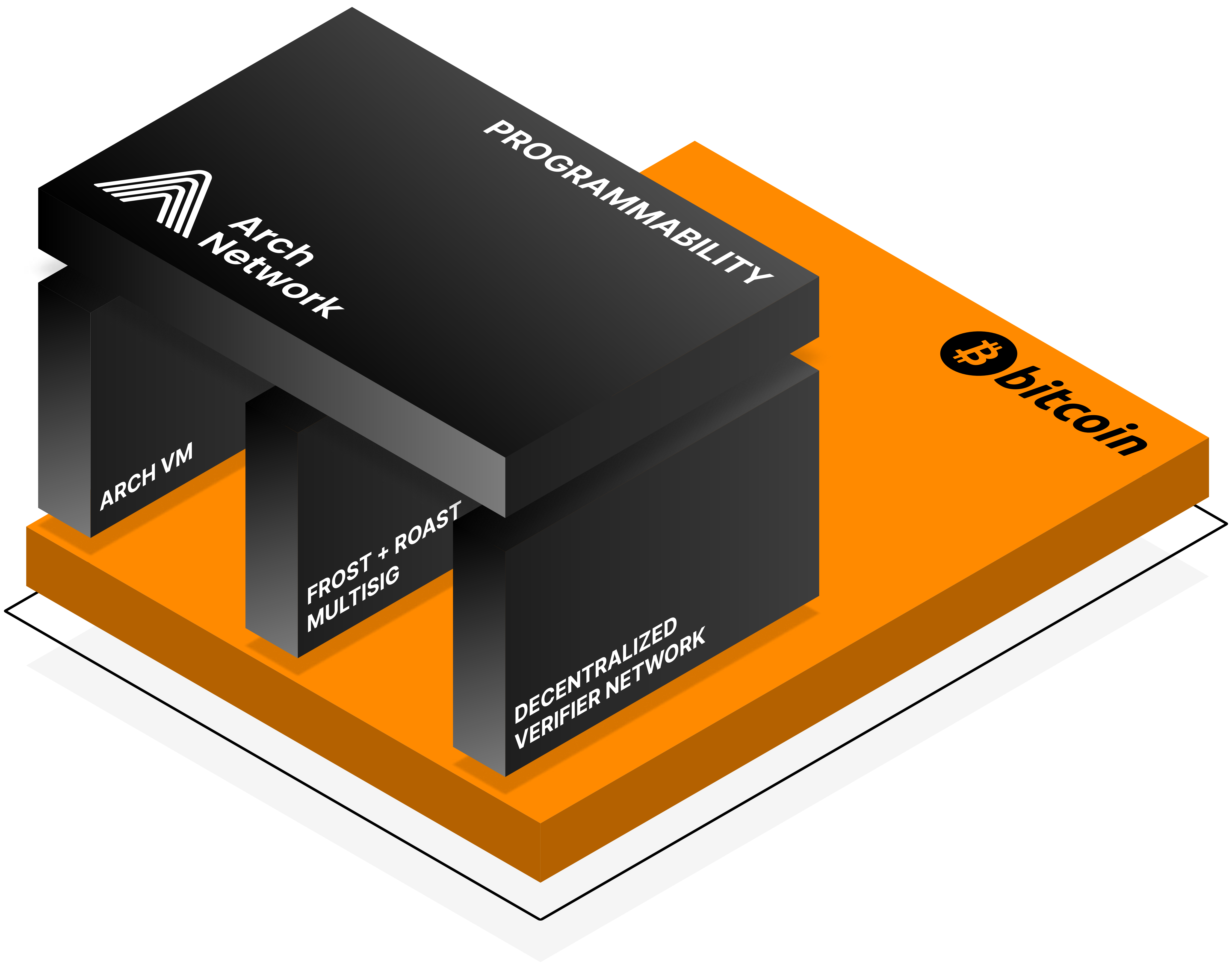


Table of Contents

Abstract 1

1.0 The Challenges Facing Bitcoin Programmability 3
1.1 Reliance on the UTXO model 3
1.2 Inability to create native fungible tokens 3
1.3 Slow block times and small block sizes 4

2.0 Why L2s and Meta-protocols Aren't Enough 4

3.0 Introducing Arch: 5
3.1 Arch's Signature Scheme Model (FROST + ROAST) 6

4.0 How Arch Works 8
4.1 Bridgeless execution 9
4.2 Decentralized Verifier Network 9
4.3 State transitions anchored on Bitcoin 9
4.4 Minimized trust assumptions 10
4.5 The Role of the Arch Token + Proof of Stake 10

5.0 Step-by-Step Developer Journey on Arch 11

6.0 How Arch Unlocks Core Decentralized Applications 12
6.1 Using multi-party contracts to enable AMMs, LPs, and DEXs 9 12
6.2 Introducing deeper liquidity to Bitcoin DeFi 10 13
6.3 Enhancing interoperability between Bitcoin L2s and meta-protocols 11 13
6.4 Future accounts-based use cases, from gaming to prediction markets 14

7.0 Conclusion 14

1.0 The Challenges Facing Bitcoin Programmability

Bitcoin has unparalleled security and liquidity but historically-limited programmability that has restricted its utility. It has no native execution environment and a limited scripting language incapable of supporting state changes and asset transfers, the core components of Turing-complete smart contracts native to blockchains like Ethereum and Solana.

Recent solutions have attempted to add greater programmability to Bitcoin, including non-fungible and fungible token standards such as Ordinals, BRC-20, and Runes. However, those solutions have all had significant limitations due to the core infrastructure challenges of building on Bitcoin, which we list below.

1.1 Reliance on the UTXO model

The UTXO (Unspent Transaction Output) model on Bitcoin is fundamentally different from the account-based model used by other blockchains like Ethereum, which are more suited to smart contract functionality. In the UTXO model, each transaction output can only be spent once, and transactions must reference specific outputs, making it challenging to manage complex, stateful applications, such as those required for DeFi.

This model doesn't naturally support the execution of multi-step transactions, pooling of funds, or the creation of smart contracts that require continuous interaction with multiple parties. As a result, implementing features like decentralized exchanges, lending protocols, or automated market makers on Bitcoin require inadequate workarounds or centralized solutions.

1.2 Inability to create native fungible tokens

Bitcoin lacks the native ability to create, transfer, and interact with tokens that represent various assets, such as stablecoins, synthetic assets, or governance tokens, which are essential for most DeFi applications. In ecosystems like Ethereum, standards like ERC-20 provide a uniform set of rules for creating and managing fungible tokens, enabling seamless integration across decentralized exchanges, lending platforms, and yield farming protocols.

This absence of standardized tokens on Bitcoin means that developers have to rely on less secure and/or more centralized solutions, such as wrapped tokens on other blockchains. These alternatives move economic activity away from the Bitcoin base layer and undermine the decentralization and security principles that Bitcoin is known for.

1.3 Slow block times and small block sizes

Bitcoin's block time, set at approximately 10 minutes, was a deliberate choice by Satoshi Nakamoto to strike a balance between security and network efficiency. This interval provides sufficient time for the network to propagate blocks and for nodes to agree on a consistent state of the blockchain, minimizing the risk of chain splits or forks. However, that latency can lead to major delays for complex protocols that require multiple transactions or interactions with the blockchain.

Bitcoin's block size, initially set at 1 MB and later increased to 4 MB with SegWit (Segregated Witness), was designed to limit the amount of data that each block can store. This limitation was intended to preserve decentralization by ensuring even smaller nodes could participate in securing the network. However, it leads to data storage constraints that make it challenging to deploy more data-intensive applications on Bitcoin, as well as throughput limitations that can lead to network congestion and higher fees during periods of high demand.

2.0 Why L2s and Meta-protocols Aren’t Enough

Bitcoin Layer 2 solutions try to address the issues of Bitcoin by building their own execution layers, which users must bridge their assets onto. This process allows for scalable programmability, often showcasing much faster block times and larger block sizes. However, bridging their assets transfers trust away from the Bitcoin base layer — by far the most liquid, secure, and decentralized blockchain — to these L2s.

This isn’t appealing to many Bitcoiners, who have already had the ability to access programmability by bridging to Ethereum for years yet have largely chosen not to do so. The total value locked of “wrapped” BTC assets in DeFi — Bitcoin that is bridged onto Ethereum — is about $10 billion as of writing, less than 1% of the total Bitcoin market cap.

A few emerging meta-protocols have also tried to add new capabilities to Bitcoin by increasing off-chain compute capacity. However, they can only handle state changes. They can’t do asset transfers, which require the ability to sign transactions programmatically and in a decentralized, secure fashion on the Bitcoin base layer.

In short, they have the same performance limitations of the Bitcoin L1 without the utility of Bitcoin L2s. Plus, most also require that users bridge their assets.

Assets that are bridged onto L2s and meta-protocols are siloed. They are unable to communicate with each other or the base layer. This means they lack the necessary interoperability components for multi-party contracts and cross-program invocation as seen on other decentralized networks.

This fragmentation lowers addressable liquidity, as different chains and meta-protocols cannot easily inter-operate on Bitcoin, making it harder to aggregate the types of deep liquidity sources needed to operate more mature DeFi applications.

To build decentralized utility and programmability, Bitcoin needs a truly bridge-free execution platform that is Turing-complete, meaning it is capable of executing both state changes and asset transfers directly on the base layer.

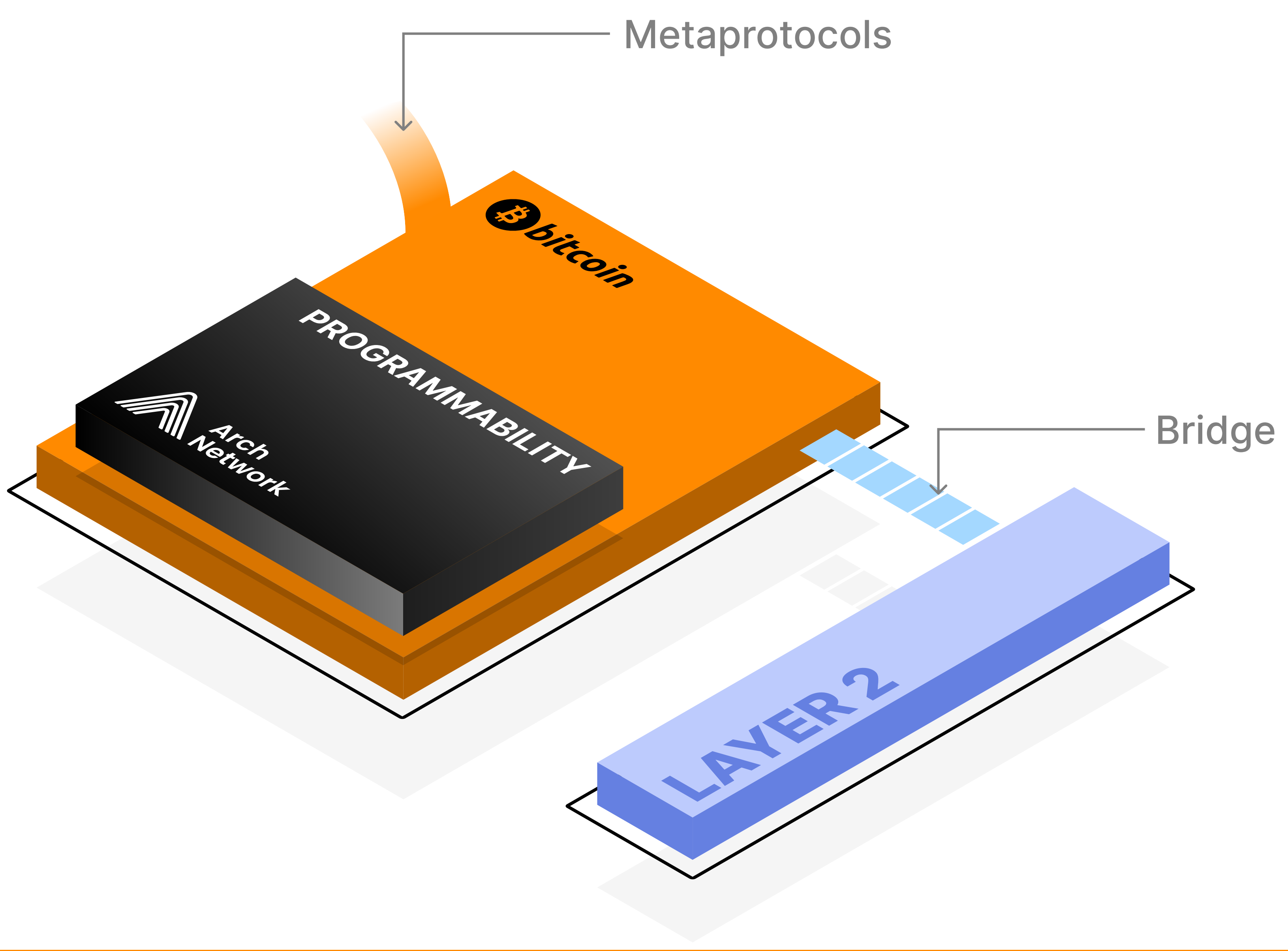
It must be able to power the complex, interoperable smart contract functionalities needed to support decentralized apps without fracturing liquidity or compromising security. And, critically, it must do so in a trust-minimized way that doesn't force Bitcoiners to trust their assets with bridges or insufficiently secure protocols.

3.0 Introducing Arch:

Arch is a parallel execution layer that creates a bridgeless platform for developers to build Turing-complete smart applications on the base layer. With no changes to Bitcoin itself, Arch can enable role-dependent trust assumptions for Bitcoin users, including fully trustless use cases — for instance, a taker of liquidity in DeFi markets can do so without taking on any additional trust assumptions beyond the Bitcoin L1.

Arch creates the conditional execution necessary for enabling programmability without removing assets from Bitcoin with the combination of its specialized Arch VM, PoS Verifier Network, and novel implementation of the FROST + ROAST signature schemes. It expands smart contract functionality for Bitcoin while maintaining block consensus and transaction data availability on the Bitcoin base layer.

This Architecture has the added benefit of offering greater interoperability between the L2s and metaprotocols that invoke smart contract-like programs through it, allowing them to communicate and transfer assets with each other in ways they previously could not on Bitcoin.



It may help to picture the Bitcoin ecosystem as a railway system.

- **Bitcoin** is the trusted main track, known for its sturdiness and popularity.
- **Layer 2s** create a separate track that allows some users to go faster or carry more cargo, but it adds some extra steps and risks because you have to shift your assets away from Bitcoin.
- **Metaprotocols** are like small sidetracks to handle specific tasks, such as loading and unloading cargo. Some may stay connected to Bitcoin, but are limited in scope, have some added risk, and can't do the heavy lifting that the main or new tracks can handle.
- **Arch** is different because it allows Bitcoin users to experience more novel applications without removing assets from the main track, fueling extra capacity and more complex operations — improvements that benefit L2s and Meta-protocols as well.

3.1 Arch’s Signature Scheme Model (FROST + ROAST)

Various signature schemes have been proposed to add to Bitcoin the same type of smart contract functionalities seen on blockchains like Ethereum and Solana, each with various pros and cons. Scripted multisigs — including P2MS (Pay-to-Multisig) and Tapscript multisigs — offer high security. However, they offer little privacy because all data is stored publicly on-chain, which also leads to them having high transaction fees and network bloat as a result. Scriptless multisigs, such as MuSig, are able to make transactions secure while putting less data on-chain. However, these protocols rely on all signers being honest and cooperative during the signing process. If even one participant behaves maliciously or provides incorrect data, the protocol fails to produce a valid signature, leading to potential delays or failures in completing execution of a transaction. In most cases, this makes them unsuitable for real-world application.

Some Bitcoin L2s, including Stacks and Botanix have tried to address this problem by using the FROST (Flexible Round-Optimized Schnorr Threshold) signature scheme. Unlike MuSig, this protocol allows transactions to go through so long as they can get a threshold of signers to participate and be responsive. This model of threshold-based consensus offers significant security while also reducing gas fees and higher execution rates.
However, its major weakness is that it can’t guarantee robustness. Networks must be able to guarantee that they’ll have a signature signed within a certain time range, but networks that rely on FROST alone can be prone to failure, leading to temporary disruptions or pauses in network activity. This failure is known as “liveness” failure and refers to a situation where the blockchain network fails to make progress by adding new blocks, even though there are no malicious activities such as double-spending or censorship.

Arch’s novel implementation is that it uses FROST, but adds on top of it a communication protocol called ROAST (Robust Asynchronous Schnorr Threshold Signatures). ROAST guarantees robustness against liveness failures by being adaptive, fault-tolerant, and responsive under varying network conditions. Its design ensures that block production continues even under less-than-ideal circumstances, thereby maintaining the liveness and reliability of the PoS network. It does so by coordinating with various nodes to make such failure significantly less likely.

Some key aspects of ROAST include...

- **Optimistic Responsiveness:** ROAST operates optimistically under normal conditions, meaning that it can make progress quickly when the network is healthy and most validators are behaving correctly. This allows the protocol to achieve low-latency block finality during times of network stability, contributing to consistent block production.

- **Asynchronous Safety:** ROAST leverages asynchronous communication, meaning that it can tolerate varying network conditions, such as delays or partial partitions, without sacrificing safety. Even if messages are delayed or lost, ROAST ensures that the consensus protocol remains safe and eventually makes progress once network conditions improve.

- **Adaptive Commit Rules:** The protocol uses adaptive commit rules that adjust based on the current network and validator conditions. If the network is experiencing delays or some validators are slow or unresponsive, ROAST can adapt by waiting longer for additional confirmations before finalizing blocks. This adaptability helps prevent stalling when some validators are offline or the network is temporarily unstable.

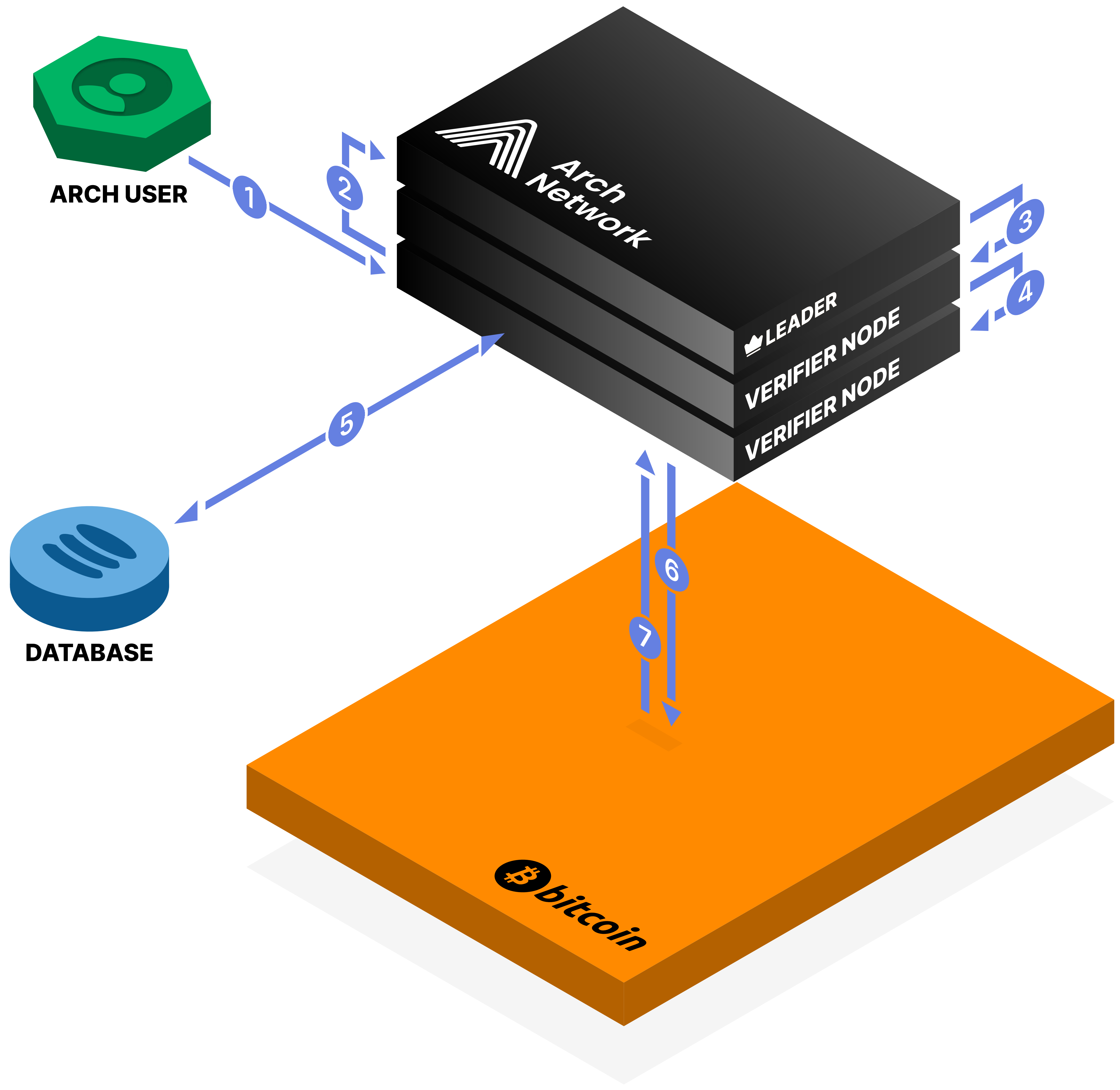
- **Fallback Mechanisms:** In the rare case where the network experiences severe issues, ROAST includes fallback mechanisms that allow the system to recover. These mechanisms might involve relaxing certain requirements temporarily to allow block production to continue or using alternate communication paths to bypass network issues.

- **Decoupling Liveness from Synchrony Assumptions:** One of the critical aspects of ROAST is that it decouples liveness from synchrony assumptions, meaning that the protocol does not rely on strict timing assumptions to make progress. This ensures that even under adverse conditions, where messages might be delayed or network partitions might occur, the protocol can still make progress as long as a sufficient number of validators are operational and honest.

This implementation of FROST + ROAST is a critical piece in enabling programmability while maintaining security of assets while they are in the Arch execution environment.

4.0 How Arch Works

Arch's FROST + ROAST signature scheme allows the Arch Verifier Network to collectively generate bitcoin addresses and sign bitcoin transactions, while retaining the ability to dynamically add and remove nodes from the signers/validators set (this allows the network to be both decentralized and robust/fault-tolerant). Each node runs an instance of the execution environment, maintains a database of the associated state, and provides an RPC interface for users to access the network.



1. Arch user sends Arch transaction
2. Gossip transaction occurs, and network waits for Leader to create a block
3. Leader generates and gossips unsigned Arch block
4. Verifier nodes execute transactions, sign UTXOs, gossip signed block
5. Verifier nodes save UTXOs to database
6. Verifier nodes send transactions to the Bitcoin Network
7. Verifier nodes retrieve latest Bitcoin Block

4.1 Bridgeless execution

Unlike L2s and meta-protocols where users must bridge their assets first and then send the assets to a smart contract, Arch allows users to send their assets directly through the smart contract invocation, as shown in Section 5 of this whitepaper. As such, all asset transfers can occur using native bitcoin transactions involving UTXOs.

4.2 Decentralized Verifier Network

Users submit transaction requests through the RPC interface, providing the associated bitcoin state anchors, input data, and bitcoin fees. Each node of the Decentralized Verifier Network distributes the request within the network, runs the program request, signs off on the result, and shares the signed result with the elected leader node. As soon as a threshold of signatures has been collected, the leader node submits the resulting bitcoin transaction.

The Decentralized Verifier Network is designed to be permissionless, maintaining the trustless environment characteristic of Bitcoin and allowing for anyone to participate in the ecosystem's security and verification processes. It uses a gossip protocol to efficiently propagate information across the network, ensuring that all verifiers receive updates in a decentralized and fault-tolerant manner, helping maintain the network's resilience and consistency.

4.3 State Transitions Anchored on Bitcoin

Although state data is only held within the Arch Network, each state transition is anchored with a bitcoin transaction. All transitions and their associated data can be requested from the Arch Network and cryptographically verified. While program execution is anchored on bitcoin transactions, throughput of the Arch Network is not bottlenecked by bitcoin block time, but only by validator hardware capability.

The Arch Network maintains real-time state, allowing for bitcoin transaction chains to be built up within the bitcoin mempool. Because these unconfirmed bitcoin transactions are controlled by the Arch Network distributed keys, the network can be confident that double spends of state transitions will not occur. All state transitions will eventually be incorporated into the bitcoin blockchain, ensuring finality.

4.4 Minimized Trust-Assumptions

Arch eliminates the need to trust a bridge with your Bitcoin to access programmability, simplifying the user experience and reducing trust assumptions by allowing users to access smart contract functionality using a native bitcoin transaction.

When invoking asset transfers, Arch users put trust in the Decentralized Verifier Network to securely manage the key generation scheme.

4.5 The Role of the Arch Token + Proof of Stake

The network will be secured by a delegated Proof-of-Stake consensus, with a fixed maximum number of validators still to-be-determined. It will include a rotating leader with a queue of validators that can be activated if one of the active nodes drops out.

The Arch token will be released to sufficiently incentivize the validators that verify the transactions and provide security on the network.

The Arch Token will be a gas token. As transactions are executed on Bitcoin and within the Arch Network, users will pay gas fees. Users can pay gas fees in either BTC or Arch Token. If BTC is used, it is automatically converted into Arch Token through a native swapping mechanism. All gas fees accumulated by the network are awarded to the network of validator nodes and stakers that make up the Decentralized Verifier Network.

Nodes can be booted out of the active set, or see their stake of Arch tokens slashed, for various behaviors detrimental to the network, such as spam or reyling bad messages. There will be various quality of service metrics, like uptime and signature activity, as is the case with other PoS networks, such as Ethereum or Solana.

The whitepaper will be updated with more information about the token and its role in securing the network after the pending publication of the Arch Tokenomics.

5.0 Step-by-Step User Journey on Arch

1. Users interacting with dApps built on Arch can invoke smart contracts by preparing, signing, and sending a transaction to Arch nodes without having to use anything but their bitcoin wallet (e.g. without bridging). When the transaction is signed by the user, they will in some cases pay a BTC fee UTXO, and, in all cases, pay Arch gas fees. Gas fees can be paid in either BTC or the Arch token.

Users send this message to an RPC through a dApp front-end, which includes essential information for execution, including State UTXOs, the related programs to execute, and custom program inputs (Bitcoin PSBTs).

Sample Arch transaction on Bitcoin:

```
pub struct RuntimeTransaction {
    pub version: u32,
    pub signatures: Vec<Signature>,
    pub message: Message,
}

pub struct Message {
    pub signers: Vec<Pubkey>,
    pub instructions: Vec<Instruction>,
}

pub struct Instruction {
    pub program_id: Pubkey,
    pub utxos: Vec<UtxoMeta>,
    pub data: Vec<u8>,
}
```

2. Transactions are sent through the RPC to the Arch VM, broadcast to the nodes using the gossip protocol, and then bundled into blocks before proceeding with execution.

Once execution is complete, the smart contract produces a bitcoin transaction with a resulting state and an unsigned transaction reflecting state transitions and asset transfers from the execution (this guarantees that you can’t have the asset transfer happen without the state transition also occurring, or vice versa).

The VM also produces a receipt proving the execution, which is then broadcast over a gossip protocol to the Arch Decentralized Verifier Network.

3. A leader node, selected from the pool of validators, receives the receipt and initiates the verifying process. Validators provide signatures that are aggregated using the FROST + ROAST signature schemes.

4. Once the transaction is signed by the requisite number of validators, the network achieves consensus, and broadcasts the new bitcoin transactions across the Bitcoin network.

5. Execution of the program is complete.

6.0 How Arch Unlocks Core Decentralized Applications

Despite significant innovations in recent years, programmability on Bitcoin has had significant limitations. In order to buy, sell, and exchange assets without giving up custody, Bitcoin-native marketplaces have previously been limited to basic trading of non-fungible assets like Ordinals. Indexers have few command options, which has made it impossible to form non-custodial AMM-style DEXs for fungible tokens, such as BRC-20 and Runes.

By enabling Turing-complete smart contract functionality on Bitcoin, Arch lays the foundation for a robust DeFi system and other significant decentralized use cases on the world's most valuable blockchain.

6.1 Using multi-party contracts to enable AMMs, LPs, and DEXs

Developers and dApp users have limited access to liquidity in the current Bitcoin DeFi system. They can do peer-to-peer transactions on Discreet-Log-Contracts (DLCs), which limits them to drawing assets from a single account. Or they can use an orderbook, which has similar liquidity challenges and only executes when buy/sell prices are matched perfectly. They can also try to use one of the few DEXs that currently exist on the Bitcoin base layer, but these still require holders to give up custody through essentially centralized multisigs to use them.

Arch's structure makes it possible to execute the multi-party contracts necessary for Liquidity Pools (LPs), which are foundational to building the Automated Market Makers (AMMs) that drive trust-minimized DEXs. These liquidity pools serve as an instant, permissionless source of liquidity from multiple sources, which borrowers can tap into without relying on DLCs or order books managed off-chain.

Example: A Bitcoin DEX

Multi-party contracts on Bitcoin enable the creation of an AMM that powers a Uniswap-style DEX like Saturn on Arch. Bitcoin holders can contribute their BTC to a liquidity pool governed by a smart contract. Traders can swap Bitcoin with other assets in the pool, with the contract determining prices based on the ratio of assets in the pool. This setup provides continuous liquidity for Bitcoin trading pairs on the DEX, allowing for seamless swaps without requiring individual trade counterparts. Liquidity providers earn a portion of the trading fees, incentivizing them to contribute to the pool.

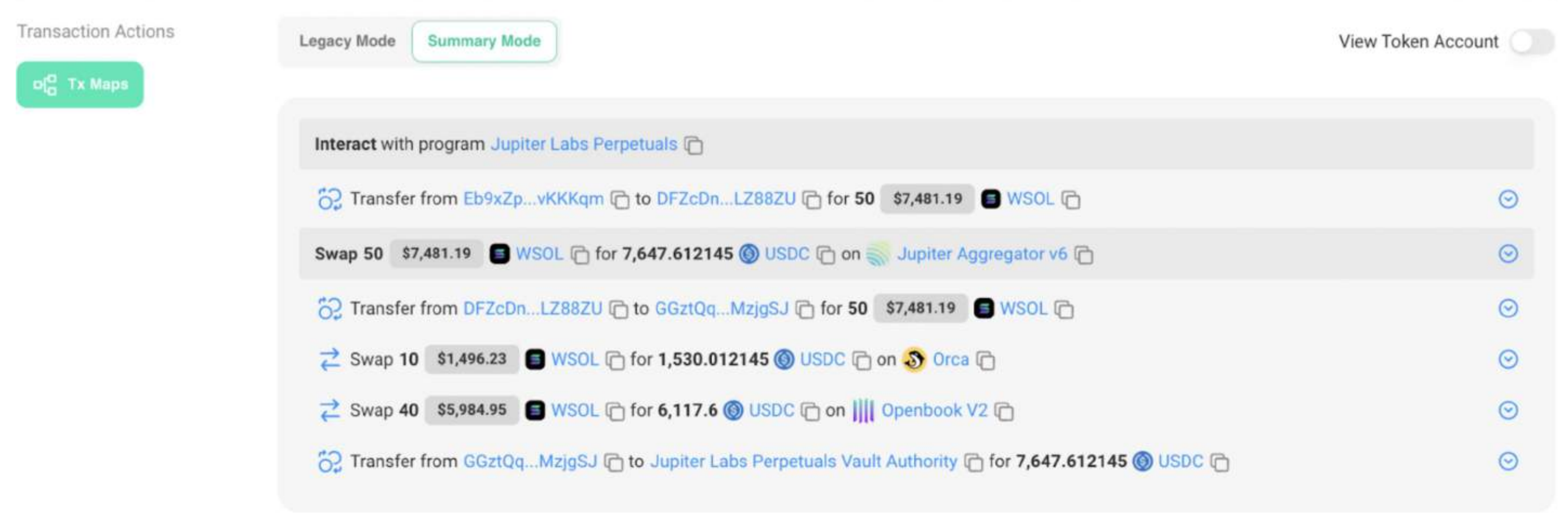
6.2 Introducing deeper liquidity to Bitcoin DeFi

Right now, Bitcoin liquidity is fractured. There are more than 80 Bitcoin L2s, metaprotocols, and other projects in development, and that number is growing every day. They need to be able to communicate with each other and transfer assets between them, but currently aren't able to programmatically do so when bridging is required to interact with their individual protocols.

To do this, Bitcoin needs a mechanism for cross-program invocation — the ability other chains have to make one smart contract contingent on the fulfillment of another smart contract within a shared network. Various Bitcoin L2s and metaprotocols can access these functions by interacting with Arch in a way that is similar to DeFi applications that work together to access liquidity on programmability-focused chains like Ethereum and Solana.

Example: Bitcoin Exchange Aggregators

The Jupiter Swap is a decentralized exchange aggregator on the Solana blockchain. It uses cross-program invocations to do so, routing trades through multiple liquidity sources to ensure optimal prices, low slippage, and efficient transaction execution. As a result, users benefit from accessing deep liquidity and the ability to perform complex token swaps in a single transaction — and Bitcoin developers can build aggregators of their own with the added cross-program invocation functionality that Arch brings to Bitcoin.



Interact with program Jupiter Labs Perpetuals
Transfer from Eb9xZp...vKKKqm to DFZcDn...LZ88ZU for **50** $7,481.19 WSOL
Swap 50 $7,481.19 WSOL for **7,647.612145** USDC on Jupiter Aggregator v6
Transfer from DFZcDn...LZ88ZU to GGztQq...MzjgSJ for **50** $7,481.19 WSOL
Swap **10** $1,496.23 WSOL for **1,530.012145** USDC on Orca
Swap **40** $5,984.95 WSOL for **6,117.6** USDC on Openbook V2
Transfer from GGztQq...MzjgSJ to Jupiter Labs Perpetuals Vault Authority for **7,647.612145** USDC

6.3 Enhancing interoperability between Bitcoin L2s and meta-protocols

Today, assets moved onto bridge-based options like L2s and meta-protocols are unable to communicate with the Bitcoin base layer until they are bridged back onto it. This makes it impossible for those protocols to transfer assets or run smart contracts between each other on Bitcoin.

By running bridge-free execution with UTXOs through Arch, a wide array of Bitcoin-based protocols suddenly become interoperable with each other — capable of running complex transactions between them, with state transitions and asset transfers all anchored and deployed on the Bitcoin base layer, using the common "language" between them that is Arch.

This process can facilitate the unification of various standards like Runes, BRC-20 tokens or other metaprotocols and L2s as they emerge, allowing those assets to be traded between one another in ways they can't currently.

6.4 Future Accounts-based Model and State-Only Execution

Arch seeks to leverage the core strength of Bitcoin, its liquidity, rather than optimizing for scalability initially. As such, Arch will immediately be conducive to decentralized projects where the paramount concern is interoperability with the base layer, rather than the immediacy of execution.

The base layer will eventually be bottlenecked by its lower speeds and storage capacity.
However, Arch will also remain valuable for Bitcoin projects that require more performance and lower latency.

Naturally, accounts-based execution models can allow for more scalability and performance. As such, Arch will introduce an additional type of execution — state-only execution — where developers remove the anchor to Bitcoin and purely rely on the Decentralized Verifier Network to manage state transitions. This transaction model will have trust assumptions and performance more similar to L2s.
With this flexibility, Arch is able to support a Bitcoin version of essentially any type of decentralized application that is already being built through the accounts-based execution models found on Ethereum, Solana, and other blockchains. Eventually this will lead to emerging Bitcoin use cases in gaming, prediction markets, oracles, socialFi, DAOs, and other decentralized apps, which will be added to this whitepaper as they emerge.

7.0 Conclusion

Arch enables DeFi and other significant use cases on Bitcoin by unlocking bridgeless, trust-minimized programmability through its novel VM and Decentralized Verifier Network architecture. Its parallel execution layer creates breakthrough opportunities to build the next generation of dApps on the world's most valuable blockchain without sacrificing the decentralization principles that are critical to the Bitcoin community.