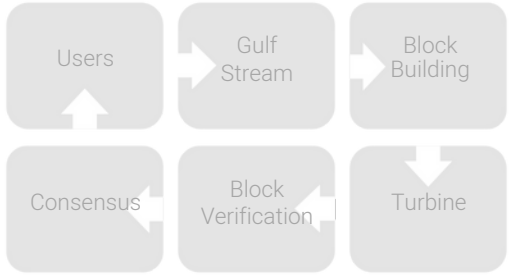
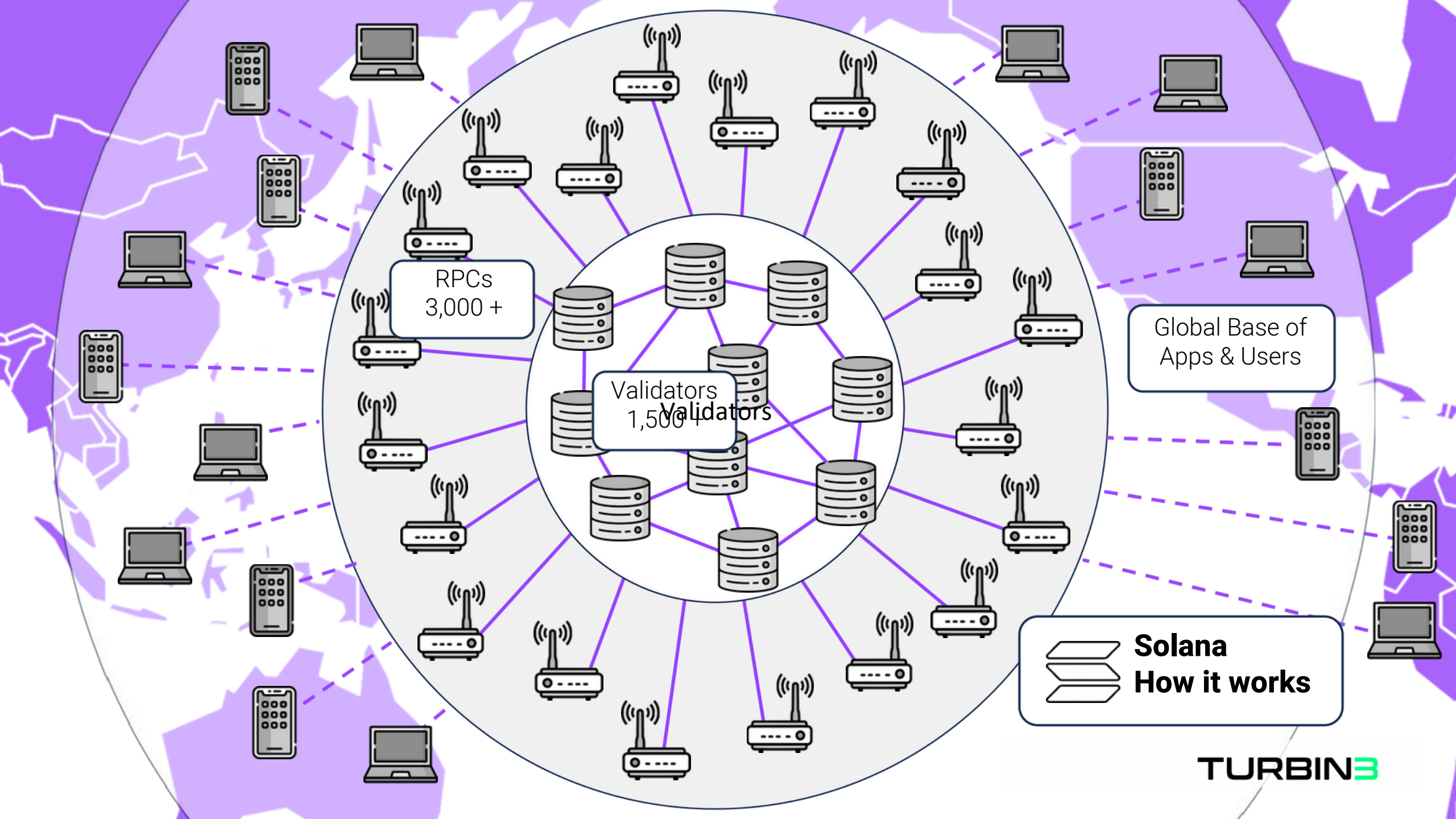
TURBIN3

Solana

How it works

An executive overview of the Solana protocol




RPCs
3,000 +
Validators
1,500 +
Validators
Global Base of
Apps & Users
Solana
How it works
TURBIN3

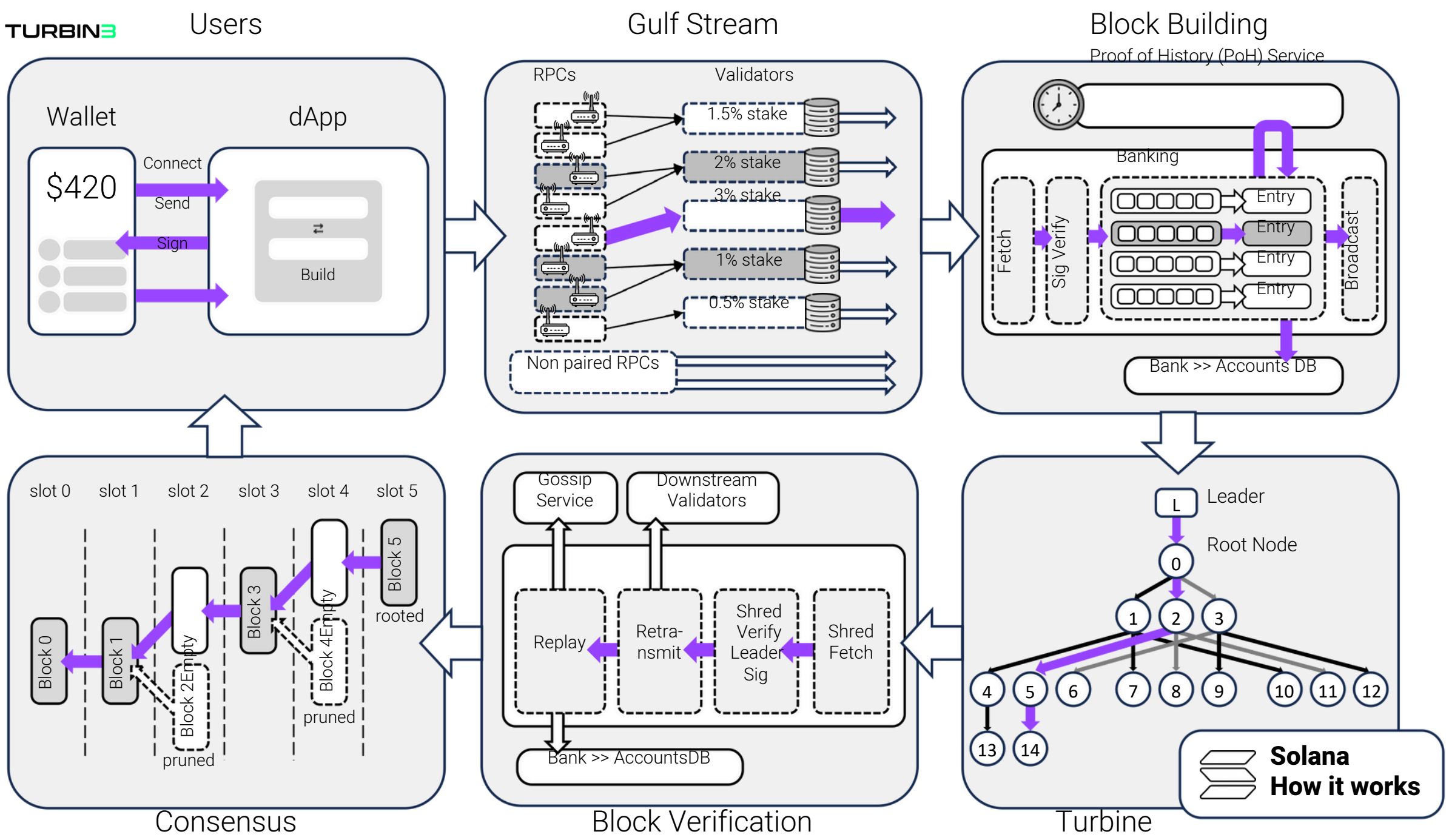

TURBINE
Users
Gulf Stream
Block Building
Proof of History (PoH) Service
Wallet
dApp
$420
Connect
Send
Sign
Build
RPCs
Validators
1.5% stake
2% stake
3% stake
1% stake
0.5% stake
Non paired RPCs
Banking
Fetch
Sig Verify
Entry
Entry
Entry
Entry
Broadcast
Bank >> Accounts DB
slot 0
slot 1
slot 2
slot 3
slot 4
slot 5
Block 0
Block 1
Block 2Empty
pruned
Block 3
Block 4Empty
pruned
Block 5
rooted
Gossip Service
Downstream Validators
Replay
Retra-nsmit
Shred Verify Leader Sig
Shred Fetch
Bank >> AccountsDB
L
Leader
Root Node
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Solana
How it works
Consensus
Block Verification
Turbine

Table of Contents

Disclaimers

The opinions and interpretations presented in this report are solely those of the author and do not represent the views of Turbin3, Helius, the Solana Foundation, Solana Labs, or any other entity. The author conducted this research through a review of documentation, literature analysis, code examination, interviews with ecosystem participants, and independent research. Nothing in this document should be misconstrued as an endorsement or recommendation to purchase the SOL token. This content is for informational purposes only and does not constitute investment advice.

Author's Note

This document aims to offer a holistic overview of the core Solana protocol that balances accessibility and depth. That is, this report aims to provide a resource that does not assume extensive industry knowledge while also not shying away from the complexities of the Solana protocol. Something easier said than done!

I hope this work can become a valuable reference to a wide range of professionals in industries adjacent to crypto, such as payments, finance, and e-commerce, for whom Solana is now appearing on their radar. While Solana is indeed more complex than traditional blockchains, it is by no means impenetrable, as this report aims to demonstrate.

My sincere thanks to 0xIchigo, dubbelosix, Jacob Creech, Maël Bomane, Nagaprasad Vr, and Rex St. John for reading earlier versions of this report and providing invaluable feedback. Lastly, I'd like to acknowledge the support and guidance of Jeff (@japarjam), Jack Sturt, and Mert in bringing this project together.

Lostin

Lostin

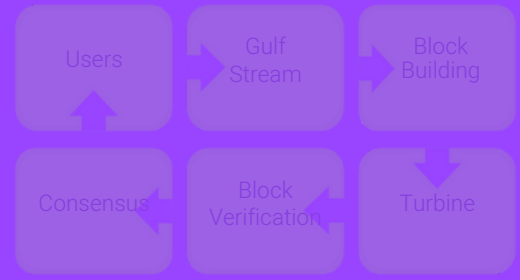
Solana enthusiast,
Technical writer, Helius

Twitter: @__lostin__

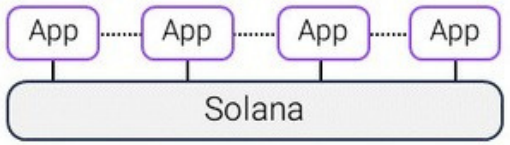
"We knew smaller, faster, cheaper better than anybody else in the world, and now we're applying those concepts to blockchain."
— Greg Fitzgerald, Solana co-founder

Global State Machine Synchronized at the Speed of Light

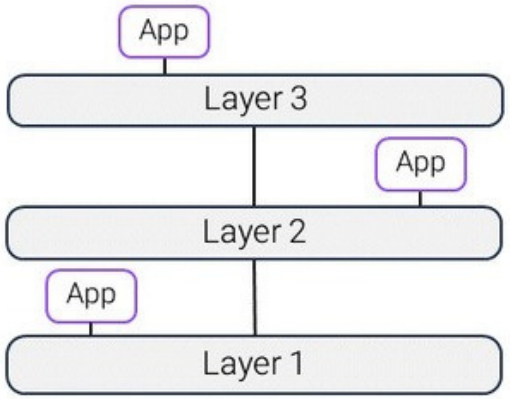
Introduction



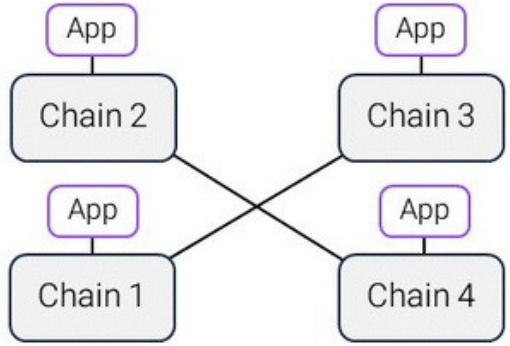
TURBIN3



Integrated



Layered



Modular

Solana is a high-performance, low-latency blockchain renowned for its speed, efficiency and focus on user experience. Its unique integrated architecture enables thousands of transactions per second across a globally decentralized network. With a block time of 400 milliseconds and transaction fees that are fractions of a cent, it delivers on both speed and cost-effectiveness.

This report delves into the intricacies of Solana's design and operation, exploring the key components and mechanisms that contribute to its capabilities. Solana takes an integrated approach to blockchain development, that leverages the founding team's decades of experience in building distributed systems. One of Solana's core principles is that software should never get in the way of hardware. This means that software exploits whatever hardware it runs on to the fullest and scales with it. As a unified ecosystem, all applications built on this single blockchain inherit composability, enabling them to interact and build upon each other.

This architecture also ensures a straightforward and intuitive user experience with no need for bridging, separate chain IDs, or fragmentation of liquidity. Solana is evolving rapidly, with recent developments such as SVM as well as important scaling solutions like ZK Compression. While these projects may one day come to shape our future perception of Solana, they are currently in the very early stages of development or adoption and will not be covered in this report.

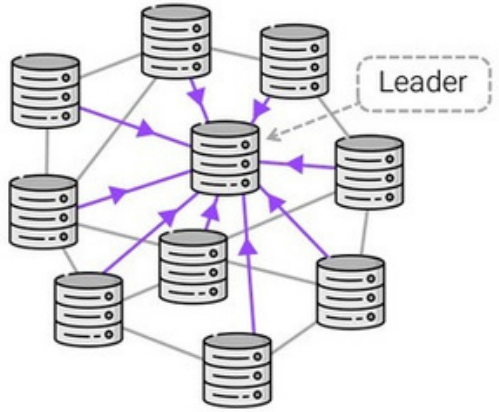
Transaction Lifecycle

Our primary lens for understanding Solana throughout this report will be the lifecycle of a typical transaction.

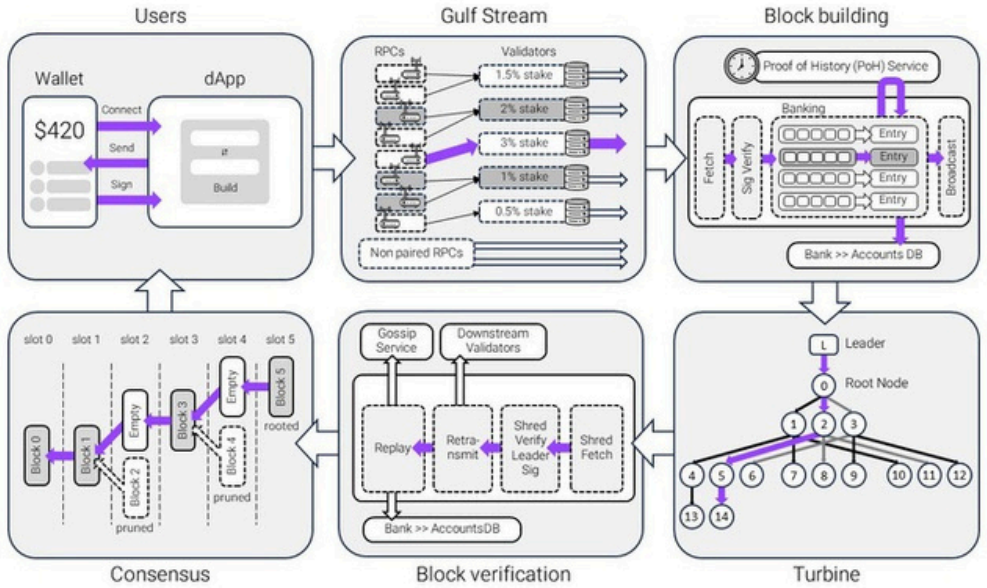
To construct a basic model for understanding Solana transactions, we can outline the process as follows:

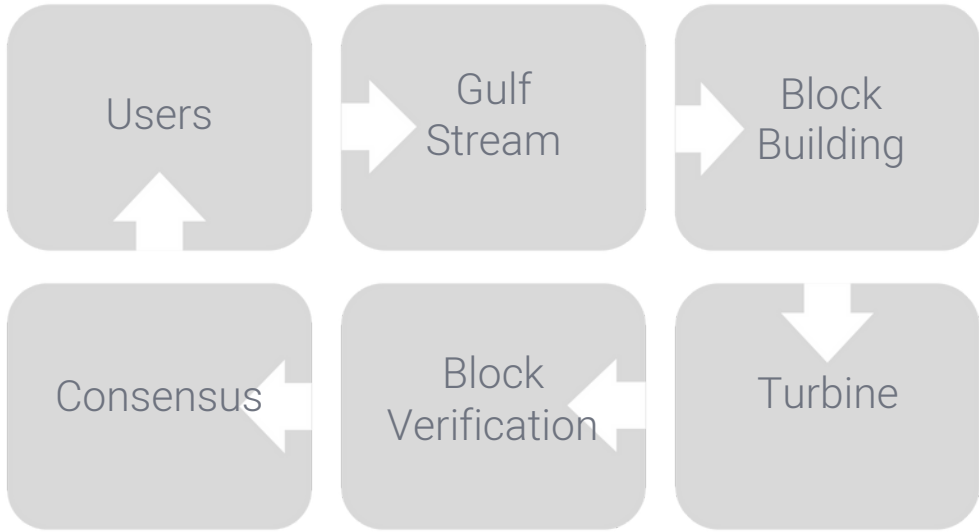
- Users initiate transactions, all of which are sent to the current lead block producer (known as the leader). The leader compiles these transactions into a block, executing them and thereby updating the blockchain's state.
- This block of transactions is then propagated throughout the network for other validators to execute and confirm.

The subsequent sections of this report will expand out this model and delve into this process in much greater detail, beginning with the key participants—the users.



Substantial changes to the core Solana protocol go through a formal, transparent process of submitting a Solana Improvement Document (SIMD) which community members and core engineering will publicly critique. SIMDs are then voted on by the network.





Six stages

We will reference the six-stage visual shown above throughout this report, as it offers us a consistent framework for understanding the relationships between Solana's core elements.

Earlier chapters are arranged according to these six stages. The final chapters—Gossip, Archive, Economics, and Jito—tie up any loose ends. However, it's important to note that some chapters will span multiple stages, and some stages will appear in multiple chapters.

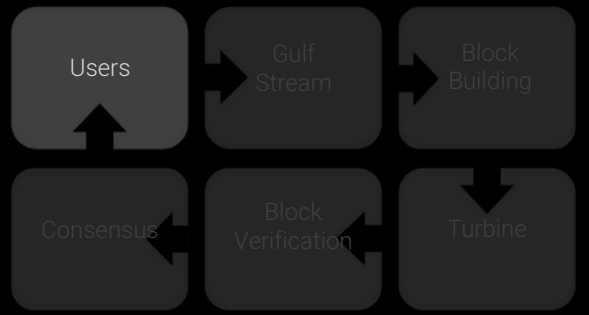
This overlap is unavoidable because the six-stage framework has its limitations. In reality, Solana is a complex distributed system with many interdependent elements.

II

"Solana has the potential to be the Apple of crypto,"
— Raj Gokal, Solana co-founder

Global State Machine Synchronized at the Speed of Light

Users



Key concepts: transactions, keypairs, Ed25519

TURBIN3

A user's journey typically begins by setting up and funding a wallet application. Multiple popular wallet applications are available for Solana, either as native mobile applications or browser extensions.
Wallets cryptographically generate user keypairs, consisting of public and private keys. The public key acts as the unique identifier for their account and is known by all participants in the network.

A user's account on Solana can be thought of as a data structure that holds information and state related to their interactions with the Solana blockchain. In this way, a public key is similar to a filename: just as a filename uniquely identifies a file within a file system, a Solana public key uniquely identifies an account on the Solana blockchain. Public keys on Solana are represented as 32-byte Base58-encoded strings.
A private key — also known as a secret key — can be considered as the password or access key that grants permission to access and modify the account. Signing with private keys is how blockchains handle authorization.
Knowledge of the private key gives absolute authority over the account. Solana private keys are also 32-bytes in length. Keypairs are the 64-byte combinations of public (first half) and private (second half) keys.

Public Key (Base58 string)

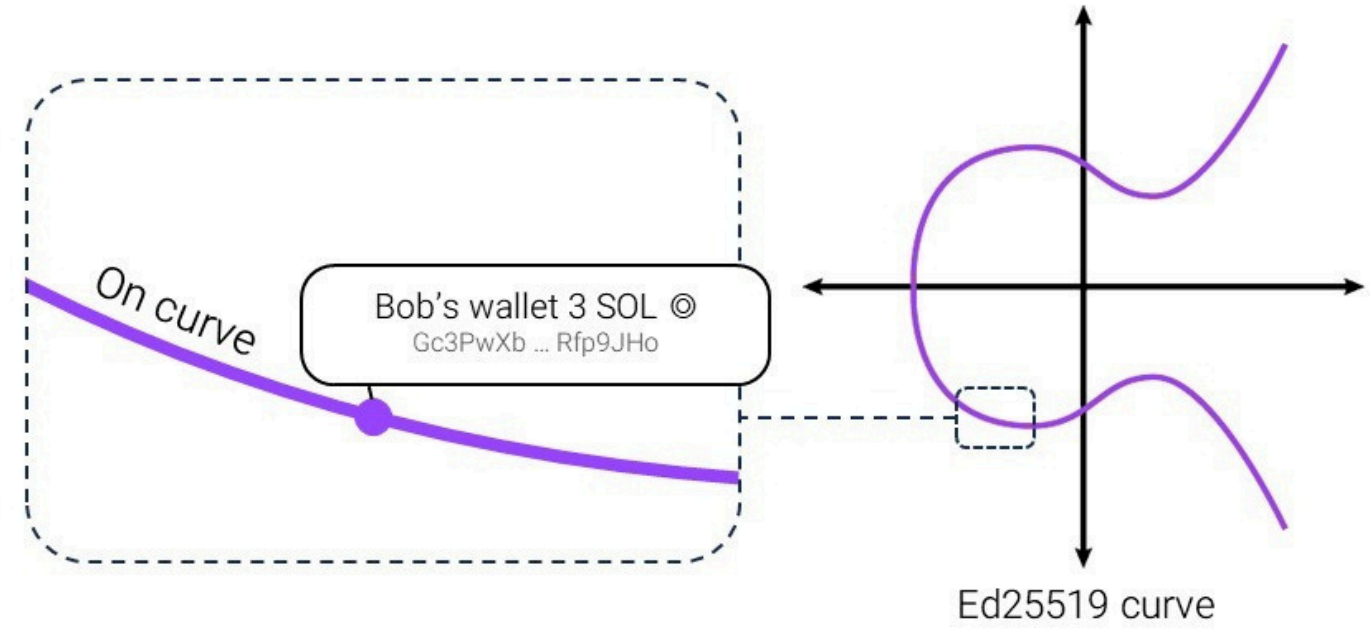
FDKJvWcJNae6wecbgDYDFPCfgs14aJnVsUfWQRYWLn4Tn

Keypair (Base58 string)

3j15jr41S9KmdfughusutvvqBjAeEDbU5sDQp8EbwQ3Hify2pfM1hiEsuFFAVq8bwGywnZpswrbDzPENbBZbd5nj

Keypair (integer array)

[63,107,47,255,141,135,58,142,191,245,78,18,90,162,107,197,8,33,211,15,228,235,250,30,185,122,105,23,147,115,115,86,8,155,67,155,110,51,117,0,19,150,143,217,132,205,122,91,167,61,6,246,107,39,51,110,185,81,13,81,16,182,30,71]

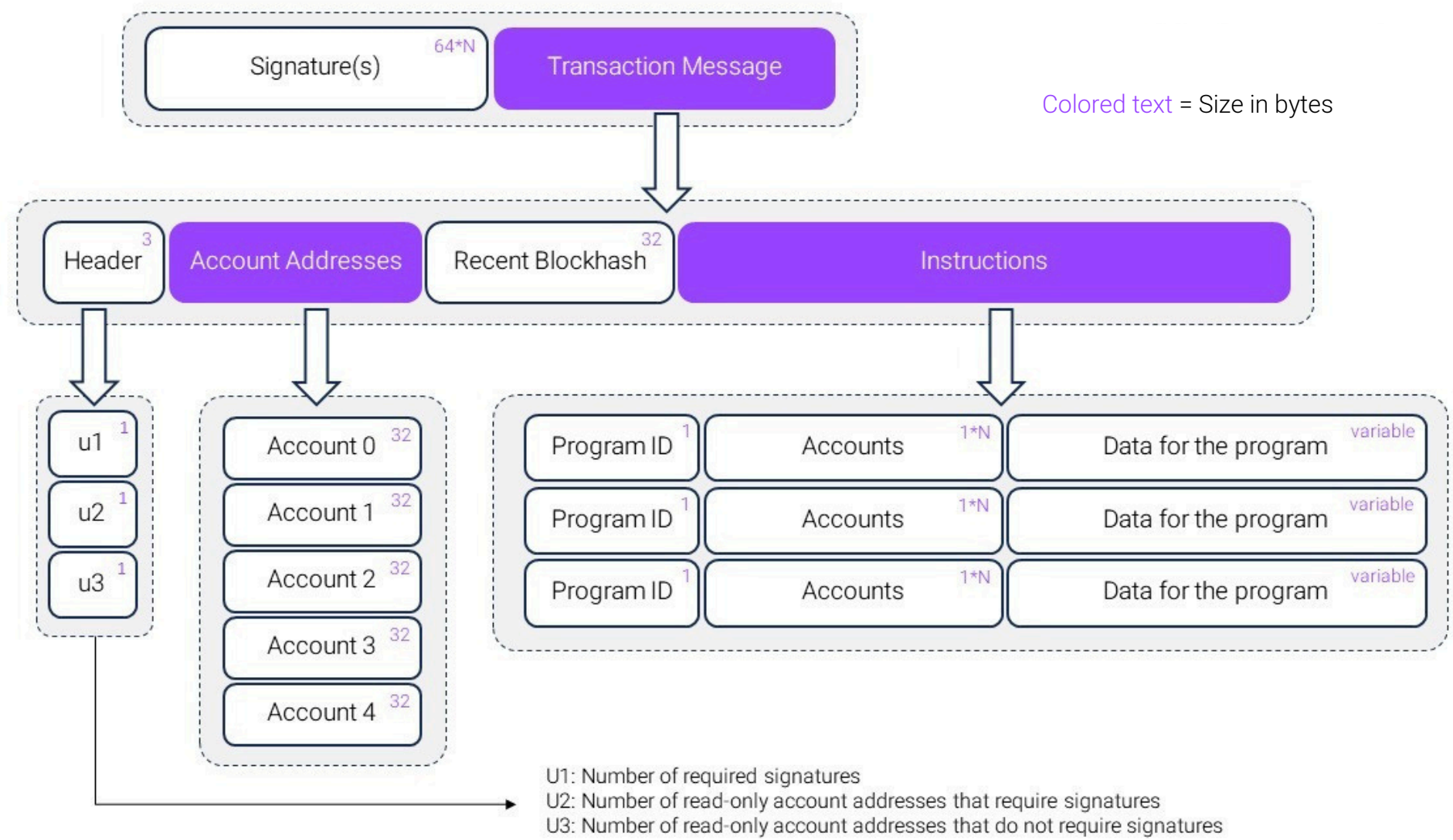


Private keys can also be derived from mnemonic seed phrases, usually consisting of 12 or 24 words. This format is often used by wallets for easier backup and recovery. Multiple keys can be derived deterministically from a single seed phrase.

Solana uses Ed25519, a widely-used elliptic curve, for its public-key cryptography needs. Ed25519 is favored for its small key size, small signature size, fast computation, and immunity to many common attacks. Each Solana wallet address represents a point on the Ed25519 elliptic curve.

The user signs transactions with their private key. This signature is included with the transaction data and can be verified by other participants using the sender's public key. This process ensures the transaction has not been tampered with and is authorized by the owner of the corresponding private key. The signature also acts as a unique identifier for the transaction. Sending a transaction is the only way to mutate state on Solana. Any write operation is performed through a transaction, and transactions are atomic—either everything the transaction attempts to do happens or the transaction fails.

A Solana Transaction



Colored text = Size in bytes

A transaction, more formally known as a "transaction message," comprises four sections: a header, a list of account addresses, a blockhash, and instructions. Account Addresses: This list includes all the accounts that will be read from or written to during the transaction. Building a transaction requires a unique list for every Solana and can be challenging for developers.

However, knowing in advance which pieces of state a transaction will interact with enables optimizations not possible on many other blockchains.
Header: The header contains references to the account address list, indicating which accounts must sign the transaction.
Recent Blockhash: This is used to prevent duplicate and stale transactions. A recent blockhash expires after 151 blocks (about 1 minute). By default, RPCs attempt to forward transactions every 2 seconds until the transaction is either finalized or the recent blockhash expires, at which point the transaction is dropped.
Instructions: These are the core of the transaction. Each instruction represents a specific operation (e.g., transfer, mint, burn, create account, close account).

Each instruction specifies the program to execute, the accounts required, and the data needed for the instruction's execution. The number of instructions in a transaction is limited first by its size, which can be up to 1,232 bytes. There are also limits around the number of accounts that can be referenced. Lastly, there are limits to a transaction's complexity measured in compute units (CUs). CUs quantify the computational resources expended in processing transactions.

total fee = prioritization fee + base fee

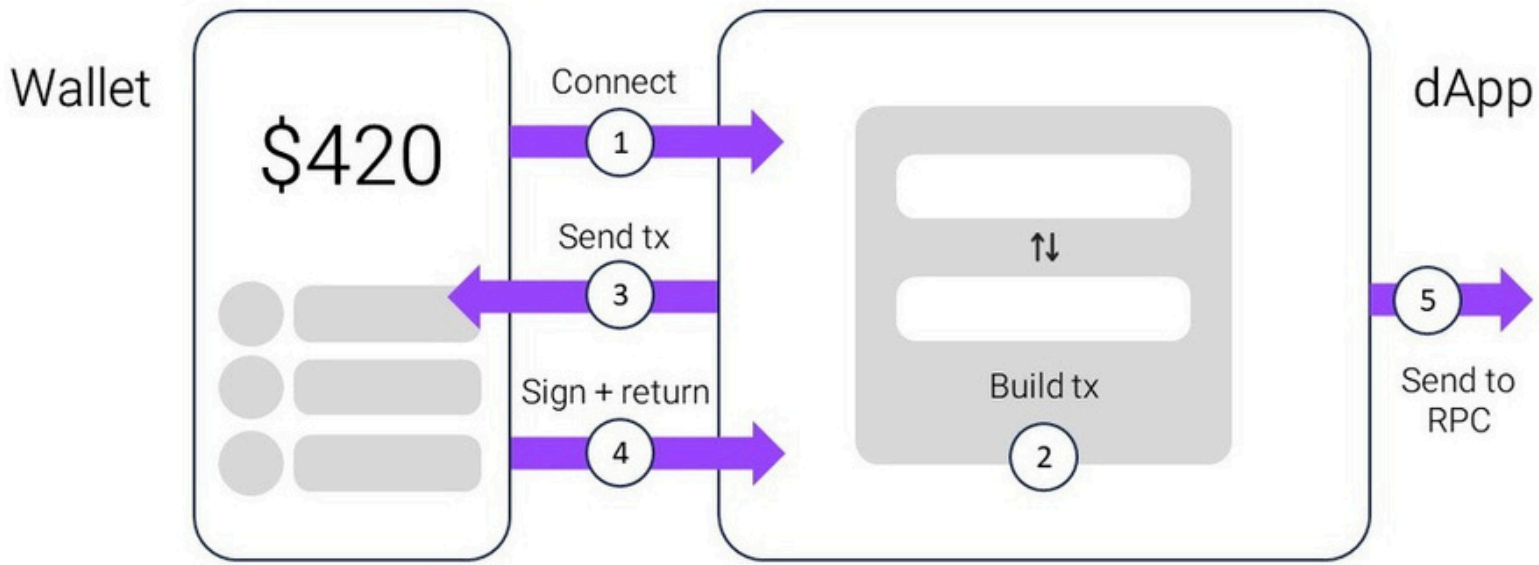
prioritization fee = compute unit price (micro-lamports) x compute unit limit

The cost in SOL to execute a transaction is separated into 2 parts - a base fee and a prioritization fee. The base fee is a fixed 5000 lamports per signature cost irrespective of the transaction's complexity, usually there is 1 signature per transaction. Prioritization fees are technically optional, but during periods of high demand for blockspace become necessary. These fees are priced in micro-lamports (millionth of a lamport) per compute unit.

Their purpose is to act as a price signal, making transactions more economically compelling for validator nodes to include in their blocks.

Currently, 50% of all transaction-related fees are burnt, permanently removing this SOL from circulation with the remaining 50% going to the block producer. A new change (SIMD 96) is soon to be introduced allowing for 100% of prioritization fees to go to the block producer. Base fees remain unchanged.

The smallest unit of SOL is known as a "lamport," equivalent to one billionth of a SOL, similar to a satoshi in Bitcoin. The lamport is named after Leslie Lamport, a computer scientist and mathematician whose research established many theoretical foundations of modern distributed systems.



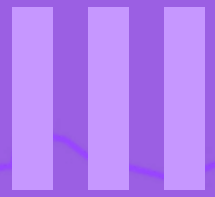
A user connects their wallet to the application, allowing the app to read the user's public key which remains encrypted and securely sandboxed in a separate environment.
The application builds the transaction message parameters based on the user's interactions. For example, if a user wanted to swap two tokens, they would specify the amount of tokens to buy, the corresponding tokens to sell, and an acceptable slippage rate.

Once the transaction message is ready, it is sent to the wallet to be signed with the user's private key. The user is prompted with a popup to confirm their willingness to transact. This popup may include a simulation of the transaction's results.

The term “failed transaction” on Solana is misleading and has caused considerable confusion. These transactions incur fees and are executed successfully by the runtime exactly as the signer intended. They “fail” due to the transaction’s own logic requiring them to do so. +80% of “failed” transactions come from error code 0x1771, the code for exceeding slippage amount. Notably, 95% of these transactions are submitted by only 0.1% of active Solana addresses, primarily automated bots attempting to take advantage of time-sensitive price arbitrage opportunities.

Once signed, the message and the signature are returned to the app, which forwards the transaction to an RPC provider. RPC (Remote Procedure Call) providers act as intermediaries between applications and the validators that build blocks.

They are an essential service that enables applications to submit or simulate signed transactions and efficiently retrieve on-chain data. Applications who wish to interact with the network do so via a JSON- RPC or a WebSocket endpoint.

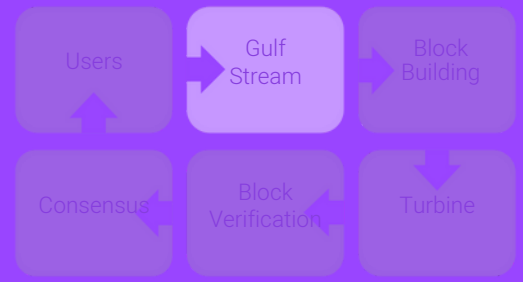


"Literally the goal of Solana is to carry transactions as fast as news travels around the world — so speed of light through fiber. Who we're competing with is NASDAQ and the New York Stock Exchange."

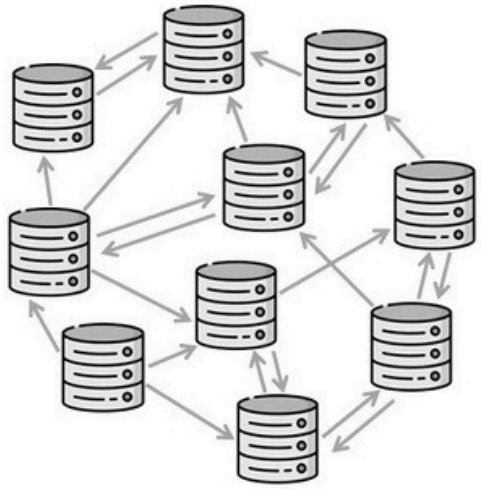
— Anatoly Yakovenko, Solana co-founder

Global State Machine Synchronized at the Speed of Light

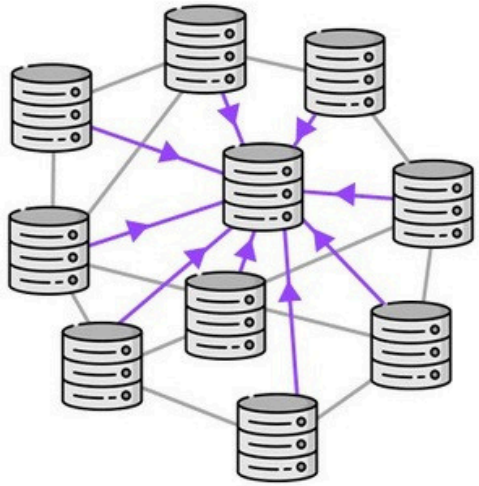
Gulf Stream



TURBIN3



Gossip network + Mempool



Gulf Stream + Leader schedule

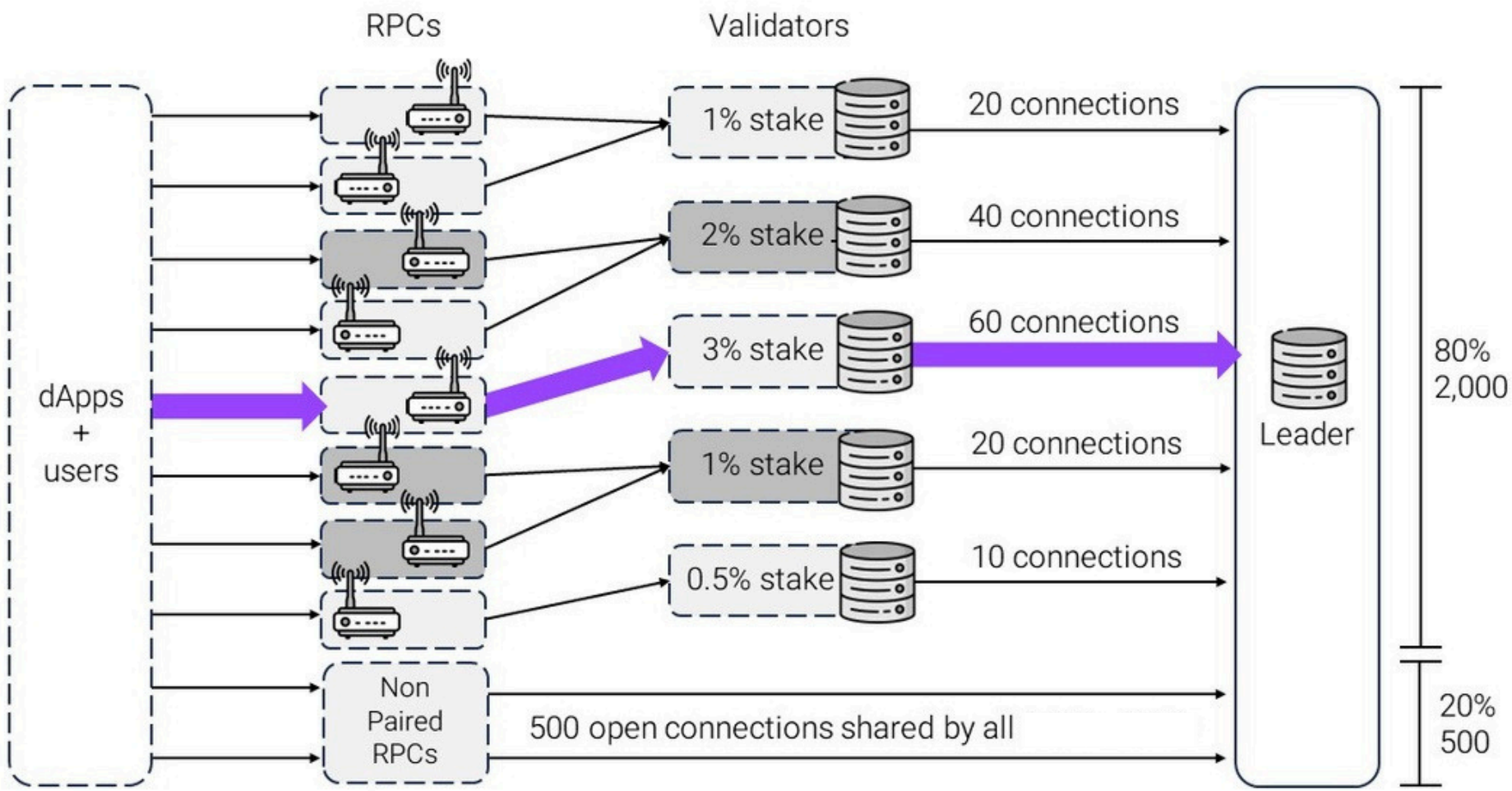
RPCs (Remote Procedure Calls) refer to RPC nodes. These nodes can be thought of as gateways to interact with and read data from the network. They run the same software as full validators but with different settings, allowing them to accurately simulate transactions and maintain an up-to-date view of the current state. As of this writing, there are over 4,000 RPC nodes on Solana.

Unlike full validator nodes, RPC nodes do not hold any stake in the network. Without stake, they cannot vote or build blocks. This setup differs from most other blockchains, where validator and RPC nodes are typically the same. Since RPC nodes do not receive staking rewards, the economics of running RPC nodes are distinct from those of validators with many operating as a paid service for developers. Solana stands out because it was designed from the outset to operate without a mempool. Unlike traditional blockchains that use gossip protocols to randomly and broadly propagate transactions across the network, Solana forwards all transactions to a predetermined lead validator, known as the leader, for each slot.

Solana operates four clusters: Localnet, Testnet, Devnet, and Mainnet-beta. When people refer to Solana or the Solana network, they almost always refer to Mainnet-Beta. Mainnet-Beta is the only cluster where tokens hold real value, while the other clusters are used solely for testing purposes.

Once an RPC receives a transaction message to be included in a block, it must be forwarded to the leader. A leader schedule is produced before every epoch (approximately every two days). The upcoming epoch is divided into slots, each fixed at 400 milliseconds, and a leader is chosen for each slot. Validators with more stake have a higher chance of being chosen as leaders. During each slot, transaction messages are forwarded to the leader, who has the opportunity to produce a block. When it is a validator's turn, they switch to "leader mode," begin actively processing transactions and broadcast them to the rest of the network. In early 2024, Solana introduced a new mechanism aimed at preventing spam and enhancing Sybil resistance, known as "Stake-Weighted Quality of Service" (SWQoS).

This system enables leaders to prioritize transaction messages that are proxied through other staked validators. Here, validators with a higher stake are granted a proportionally higher capacity to send packets to the leader. This approach effectively mitigates Sybil attacks from non- staked nodes across the network.


RPCs
Validators
dApps
+
users
1% stake
20 connections
2% stake
40 connections
3% stake
60 connections
1% stake
20 connections
0.5% stake
10 connections
Leader
80%
2,000
Non
Paired
RPCs
500 open connections shared by all
20%
500

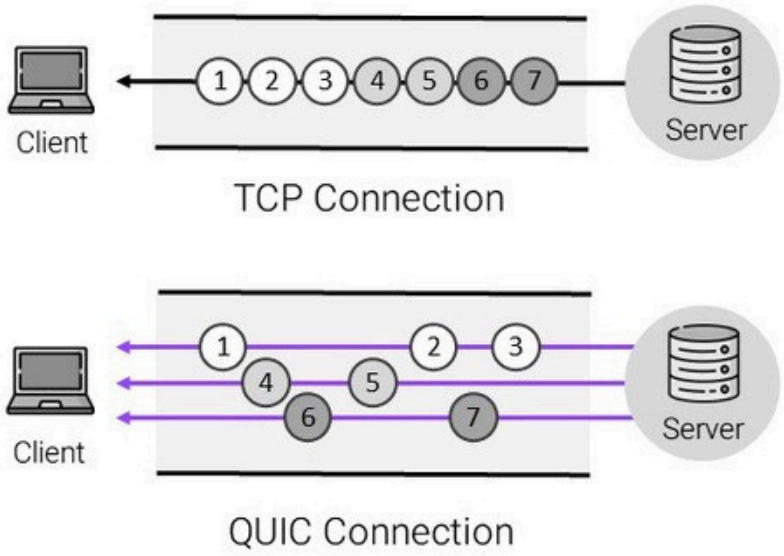
Under this model, validators can also engage in agreements to lease their stake-weighted capacity to RPC nodes. In return, RPC nodes gain increased bandwidth, allowing them to achieve greater transaction inclusion rates in blocks. Notably, 80% of a leader's capacity (2,000 connections) is reserved for SWQoS, while the remaining 20% (500 connections) is allocated for transaction messages from non-staked nodes. This allocation strategy mirrors priority lanes on highways, where drivers pay a toll to avoid traffic. SWQoS has impacted the Solana ecosystem by raising the requirements to forward transactions to the leader and reducing the effectiveness of spam attacks. The change has incentivized high-traffic applications to vertically integrate their operations. By running their own validator nodes, or by having access to staked connections, applications can ensure privileged access to the leader, enhancing their transaction processing capabilities.

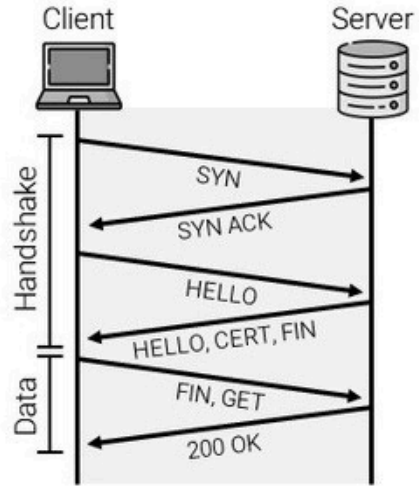
A QUIC Note

In late 2022, Solana adopted the QUIC networking protocol to manage transaction message transmission to the leader. This transition was prompted by network disruptions caused by bots spamming on-chain NFT mints. QUIC facilitates rapid, asynchronous communication. Initially developed by Google in 2012, QUIC attempts to offer the best of both worlds.

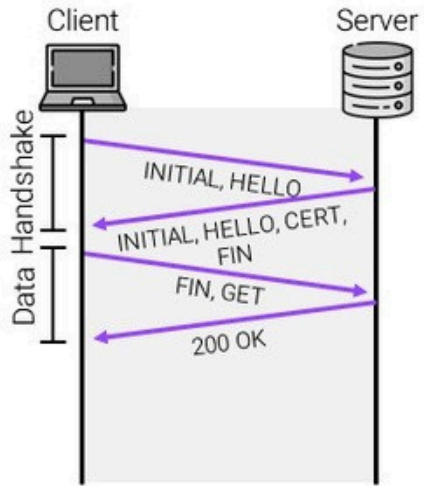
It facilitates rapid, asynchronous communication similar to UDP, but with the secure sessions and advanced flow control strategies of TCP. This allows for limits to be placed on individual sources of traffic so that the network can focus on processing genuine transactions. It also has a concept of separate streams; so if one transaction is dropped, it doesn't need to block the remaining ones. In short, QUIC can be thought of as attempting to combine the best characteristics of TCP and UDP.

Stake-weighting is a recurring principle found throughout Solana's core systems, encompassing voting rewards, turbine trees, leader schedules, gulf stream and the gossip network. Validators with greater stake are accorded higher trust and prioritized roles in the network.





TCP Handshake
+TLS/1.3



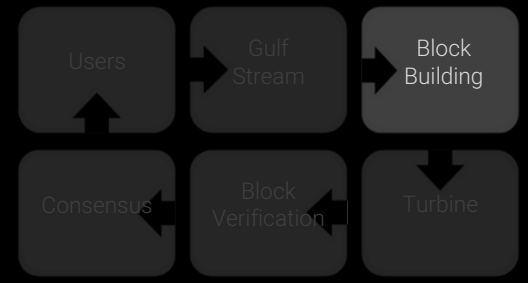
QUIC Handshake

IV

"We consider SVM (Solana Virtual Machine) the best in terms of virtual machine technology currently."
— Andre Cronje, Fantom Foundation

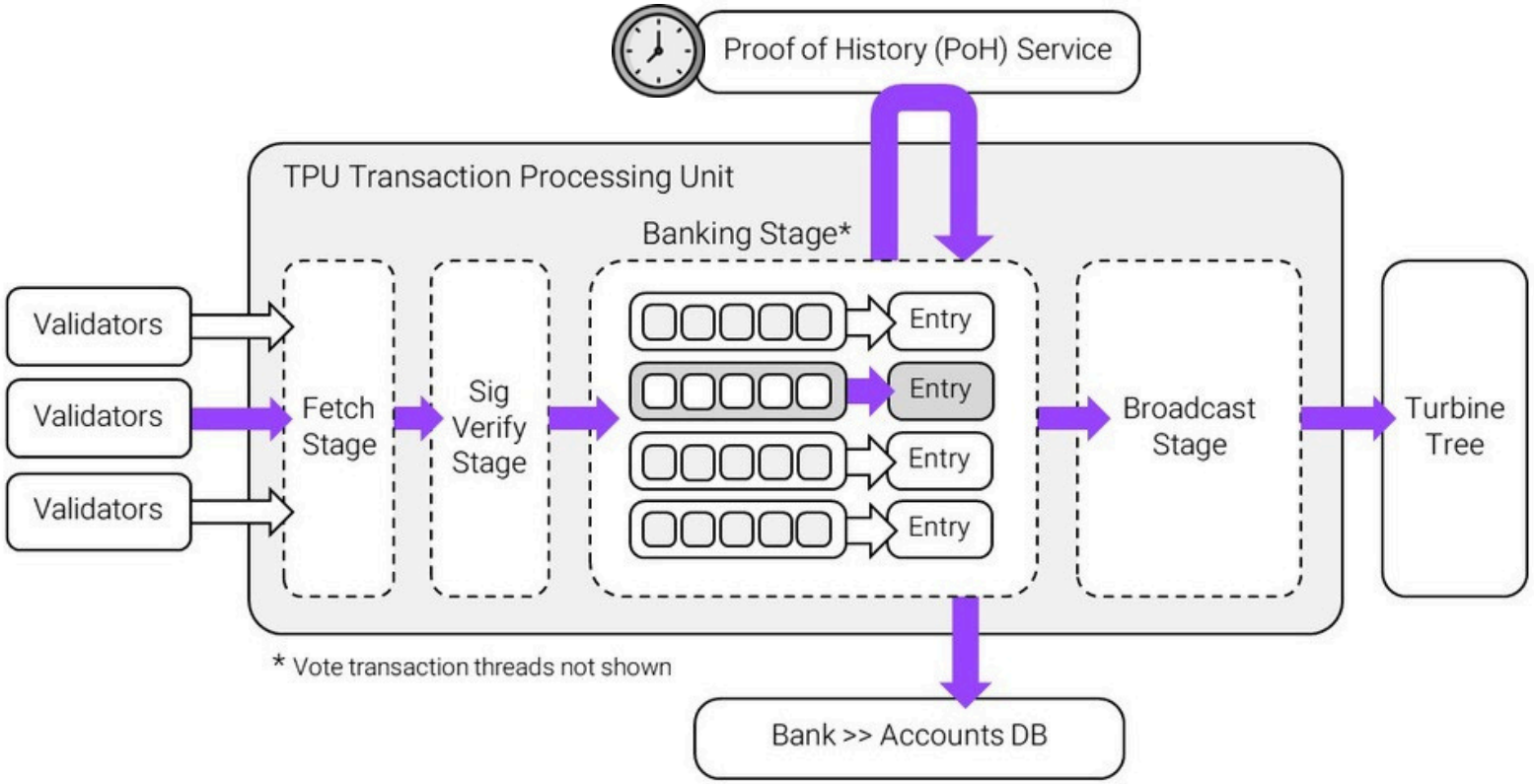
Global State Machine Synchronized at the Speed of Light

Block Building



TURBIN3

Many blockchain networks construct entire blocks before broadcasting them, known as discrete block building. Solana, in contrast, employs continuous block building which involves assembling and streaming blocks dynamically as they are created during an allocated time slot, significantly reducing latency. Each slot lasts 400 milliseconds, and each leader is assigned four consecutive slots (1.6 seconds) before rotation to the next leader. For a block to gain acceptance, all transactions within it must be valid and reproducible. Two slots prior to assuming leadership, a validator halts transaction forwarding to prepare for its upcoming workload. During this interval, over a gigabyte per second of traffic is buffered, as the entire network directs packets to the incoming leader. Upon receipt, messages enter the Transaction Processing Unit (TPU), the validator's core logic responsible for block production. Here, the transaction processing sequence begins with the Fetch Stage, where transactions are received via QUIC. Subsequently, transactions progress to the SigVerify Stage, undergoing rigorous validation checks. Here the validator verifies the validity of signatures, checks for the correct number of signatures, and eliminates duplicate transactions. The banking stage can be described as the block-building stage. It is the most important stage of the TPU, which gets its name from the "bank". A bank is just the state at a given block. For every block, Solana has a bank that is used to access state at that block. When a block becomes finalized after enough validators vote on it, they will flush account updates from the bank to disk, making them permanent. The final state of the chain is the result of all confirmed transactions. This state can always be recreated from the blockchain history deterministically.



Transactions are processed in parallel and packaged into ledger "entries," which are batches of 64 non-conflicting transactions. Parallel transaction processing on Solana is made easy because each transaction must include a complete list of all the accounts it will read and write to. This design choice places burden on developers but allows the validator to avoid race conditions by easily selecting only non-conflicting transactions for execution within each entry. Transactions conflict if they both attempt to write to the same account (two writes) or if one attempts to read from and the other writes to the same account (read + write). Thus conflicting transactions go into different entries and are executed sequentially, while non-conflicting transactions are executed in parallel.

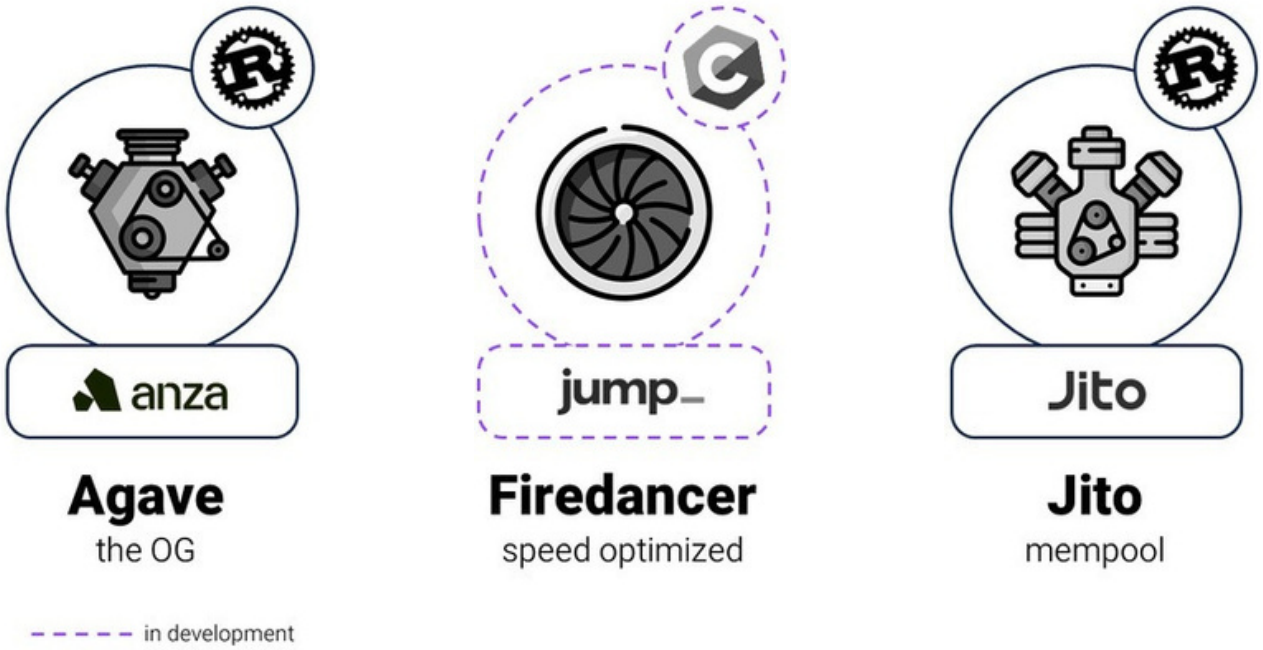
The term SVM can be ambiguous, as it may refer to either the "Solana Virtual Machine" or the "Sealevel Virtual Machine." Both terms describe the same concept, with Sealevel being the name of Solana's runtime environment. The term SVM continues to be loosely thrown around despite recent efforts to precisely define its boundaries.

There are six threads processing transactions in parallel, with four dedicated to normal transactions and two exclusively handling vote transactions which are integral to Solana's consensus mechanism. All parallelization of processing is achieved through multiple CPU cores; validators have no GPU requirements.

Once transactions have been grouped into entries they are ready to be executed by the Solana Virtual Machine (SVM).

The accounts necessary for the transaction are locked; checks are run to confirm the transaction is recent but hasn't already been processed. The accounts are loaded, and the transaction logic is executed, updating the account states. A hash of the entry will be sent to the Proof of History service to be recorded (more on this in the next section). If the recording process is successful, all changes will be committed to the bank, and the locks on each account placed in the first step are lifted. Execution is done by the SVM, a virtual machine built using the Solana fork of rBPF, a library for working with JIT compilation and virtual machines for eBPF programs.

Note Solana does not mandate how validators choose to order transactions within a block. This flexibility is a crucial point that we will return to later in the Economics + Jito section of this report.



Clients

Solana is a network comprising thousands of independently operated nodes collaborating to maintain a single unified ledger. Each node constitutes a high-performance machine running the same open source software known as a "client". Solana launched with a single validator client software — originally the Solana Labs client, now known as the Agave client — written in Rust. Expanding client diversity has been a priority ever since and one that will truly come to fruition with the launch of the Firedancer client.

Firedancer is a complete ground-up rewrite of the original client in the C programming language. Built by an experienced team from high-frequency trading firm Jump, it promises to be the most performant validator client on any blockchain.

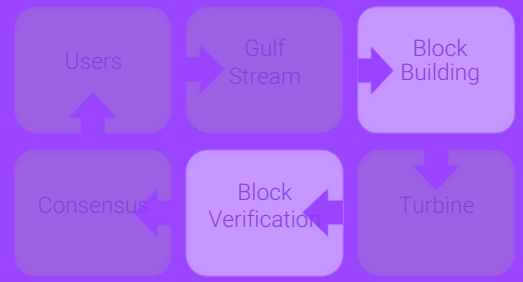
V

> “I hadtwo coffees and a beer, and I was up until 4:00 AM. I had thiseureka momentthat puzzle [sic] similar to proof of work using thesame SHA-256 preimage-resistant hash function… I knew that I had this arrow of time.”
> — Anatoly Yakovenko, Solana co-founder

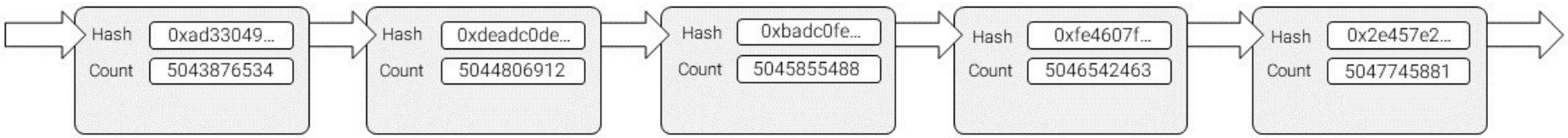
Global State Machine Synchronized at the Speed of Light

Proof of History

Key concepts: SHA256, PoH Service



TURBIN3



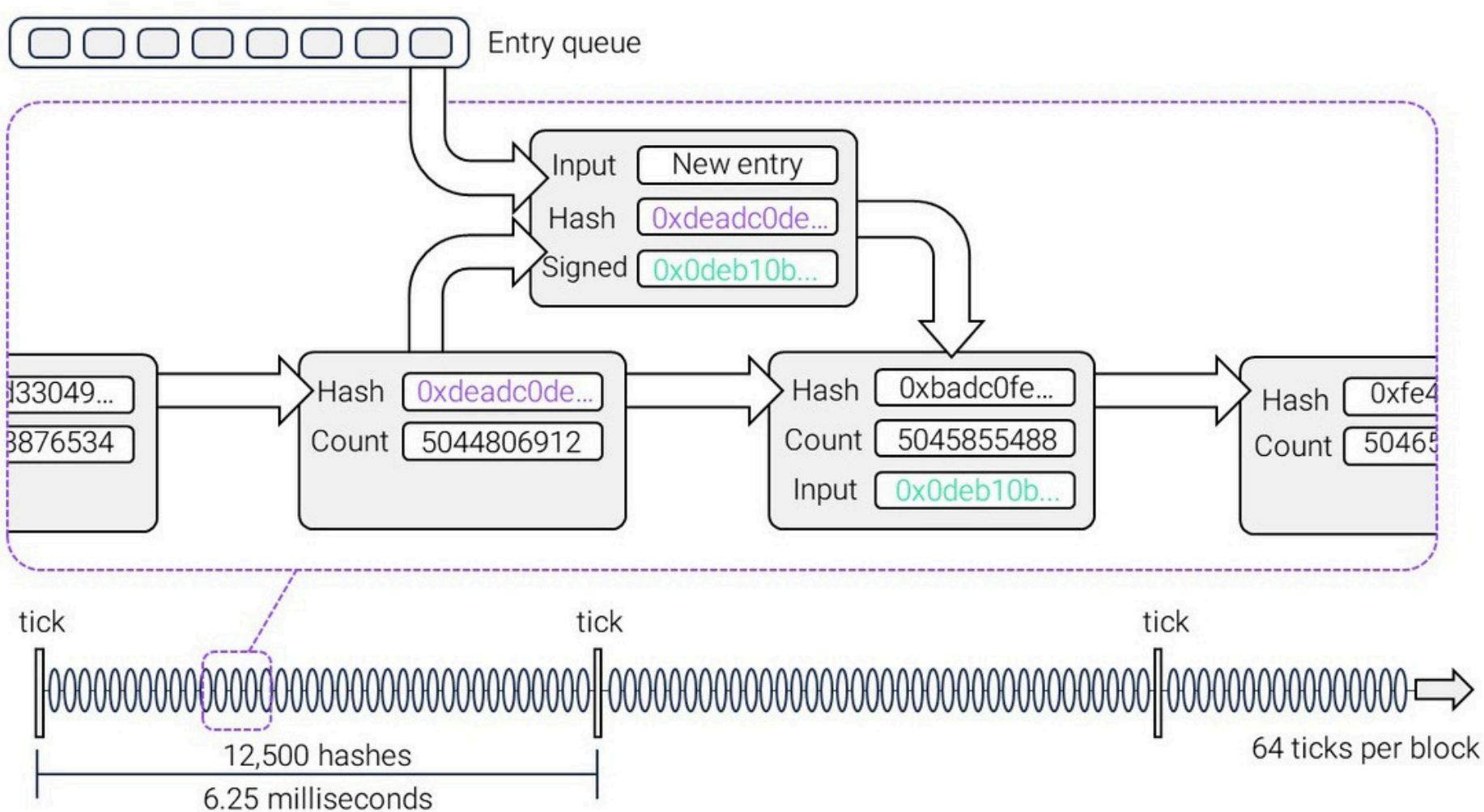
Proof of History (PoH) is Solana's secret sauce, functioning like a special clock in every validator that facilitates synchronization across the network. PoH establishes a reliable source of truth for the order of events and the passage of time. Most critically it ensures adherence to the leader schedule. Despite similar names, Proof of History is not a consensus algorithm such as Proof of Work. Communication overhead between nodes typically increases as networks expand, and coordination becomes increasingly complicated. Solana mitigates this by replacing node-to-node communication with a PoH local computation. This means validators can commit to a block with just a single round of voting. Trusted timestamps in messages ensure that validators cannot start their blocks prematurely and must wait a minimum amount of time before submitting their block.

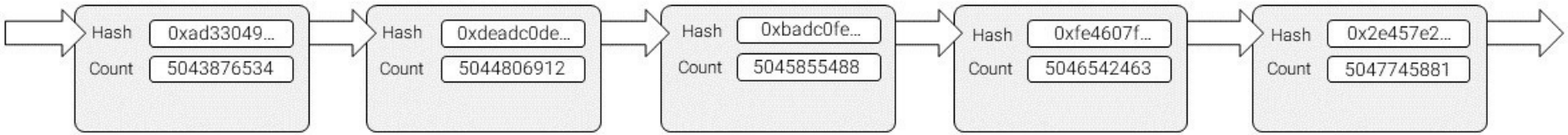
Underlying PoH are the unique properties of hashing algorithms, specifically SHA256:

- Deterministic: The same input will always produce the same hash.
- Fixed Size: Regardless of input size, the output hash is always 256 bits.
- Efficient: It is relatively fast to compute the hash for any given input.
- Preimage Resistance: Finding the original input from the hash output is computationally infeasible.
- Avalanche Effect: A small change in the input (even a single bit) results in a significantly different hash, a property known as the avalanche effect.
- Collision Resistance: It is infeasible to find two different inputs that produce the same hash output.

Within each validator client, a dedicated Proof of History service continually runs the SHA256 hash algorithm creating a chain of hashes. Each hash's input is the previous hash's output. This chain acts the same as a verifiable delay function, given that the hashing work must be done in sequence and the results of future hashes cannot be known ahead of time. If the PoH service creates a chain of a thousand hashes, we know time has passed for it to have calculated each hash sequentially— this can be thought of as a "micro proof of work." Yet, other validators can verify the correctness of the thousand hashes in parallel at a much faster rate than they were produced since the input and output of each hash have been broadcast to the network. Therefore, PoH is difficult to produce but easy to verify.

The range of performance in computing SHA-256 across different CPUs is surprisingly narrow, with only small differences among the fastest machines. A common upper limit has already been reached, despite significant time and effort being invested in optimizing this function, largely due to Bitcoin's reliance on it. During a leader's slot, the PoH service will receive newly processed entries from the banking stage. The current PoH hash plus a hash of all transactions in the entry are combined into the next PoH hash. This serves as a timestamp that inserts the entry into the chain of hashes, proving the sequence that transactions were processed. This process not only confirms the passage of time but also serves as a cryptographic record of the transactions. In a single block, there are 800,000 hashes. The PoH stream also includes "ticks," which are empty entries indicating the leader's liveness and the passage of time.


Entry queue
Input
New entry
Hash
0xdeadc0de...
Signed
0x0deb10b...
Hash
0xdeadc0de...
Count
5044806912
Hash
0xbadc0fe...
Count
5045855488
Input
0x0deb10b...
Hash
Count
tick
tick
tick
12,500 hashes
6.25 milliseconds
64 ticks per block



The key benefit of PoH is that it ensures the correct leader schedule must be adhered to, even if a block producer is offline — a state known as being "delinquent". PoH prevents a malicious validator from producing blocks before their turn.

A tick occurs every 6.25 milliseconds, resulting in 64 ticks per block and a total block time of 400 milliseconds.

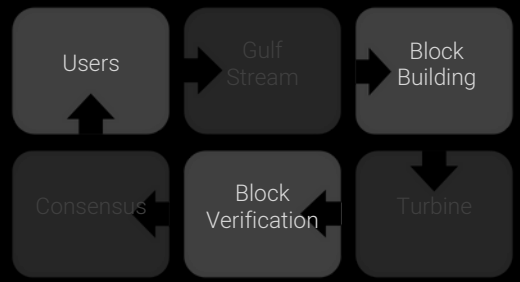
Validators continually run the PoH clock even when they are not the leader. It plays a pivotal role in the process of synchronization between nodes.

VI

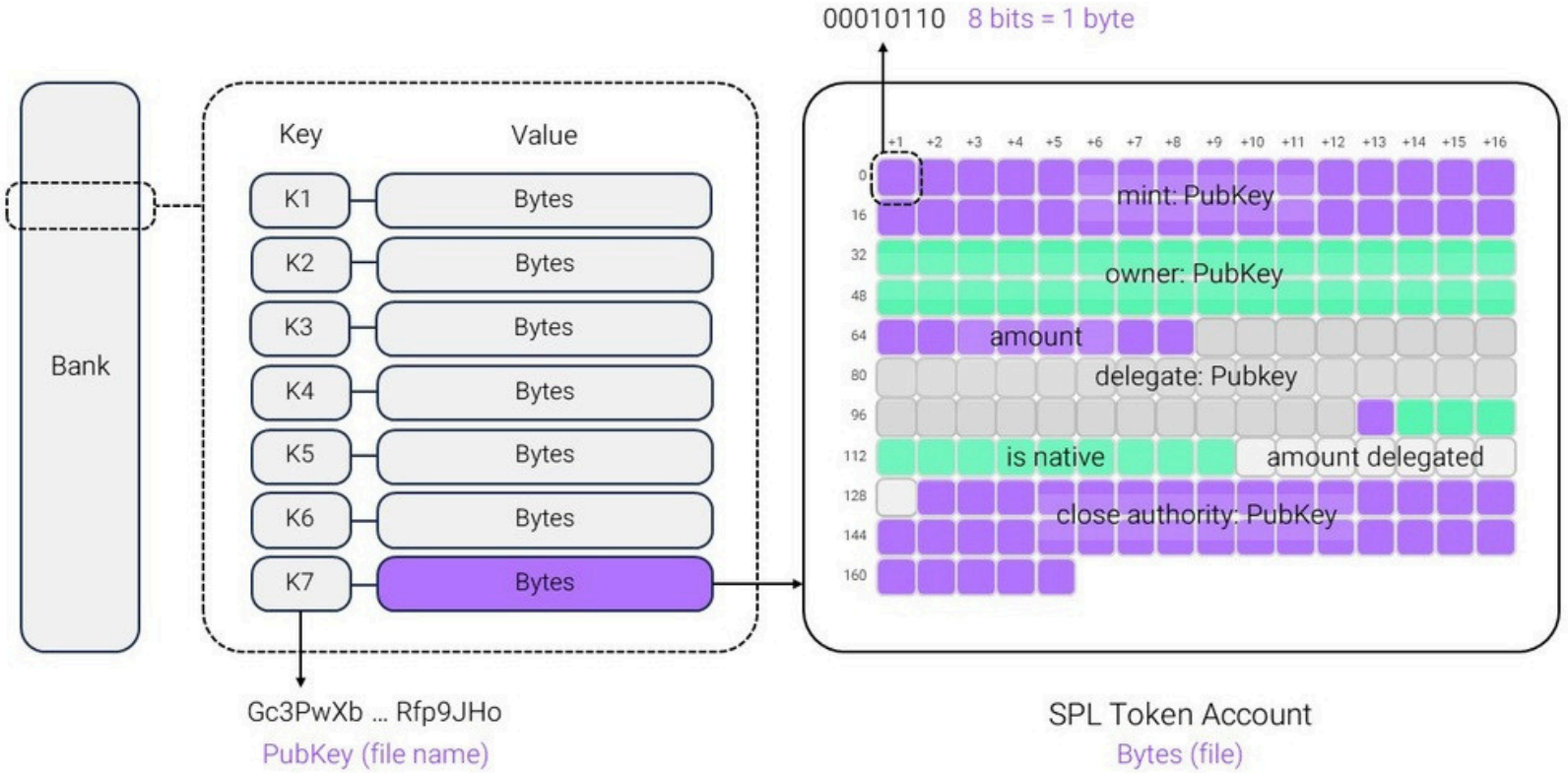
"Separating code and state in SVM was the best design decision. Blessed are the embedded system devs that religiously drilled this concept into my brain
— Anatoly Yakovenko, Solana co-founder

Global State Machine Synchronized at the Speed of Light

Accounts Model



TURBIN3



Within a Solana validator, the global state is maintained in the accounts database known as AccountsDB. This database is responsible for storing all accounts, both in memory and on disk.

The primary data structure in the account index is a hashmap, making AccountsDB essentially a vast key-value store. Here, the key is the account address, and the value is the account data.

As of writing, the number of Solana accounts is easily in the hundreds of millions. This large number is partly because, as Solana developers often say, "Everything on Solana is an account!"

Solana accounts

An account is a container that persistently holds data, similar to a file on a computer. They come in various forms:

- User accounts: These accounts have a private key and are typically generated by a wallet software for a user.
- Data accounts: These accounts store state information, such as the number of a specific token a user holds.
- Program accounts: These are larger accounts that contain executable bytecode, somewhat equivalent to a .exe file on Windows or a .app file on Mac.
- Native program accounts: These are special pre-deployed program accounts that perform various core functionalities of the network. Examples include the Vote Program and the BPF Loader.

All accounts have the following fields: (see table)

Field	Format	Description
Owner	Public key address	The owner of the account. By default this is the System Program.
Lamports	u64	The amount of SOL held by the account, measured in Lamports (1 billionth of a SOL)
Data	Bytes, variable length	This is the most important field, akin to the contents of a file.
Rent Epoch	u64	Legacy field for backward compatibility. More on rent later.
Executable	Boolean flag	Marked ‘true’ if the account holds an executable program.

Solana program accounts contain only executable logic. This means when a program is run it mutates the state of other accounts but remains unchanged itself. This separation of code and state differentiates Solana from other blockchains and supports many of its optimizations. Developers primarily write these programs in Rust, a general-purpose programming language known for its strong focus on safety and performance. Additionally, multiple SDKs in TypeScript and Python are available to facilitate the creation of application front-ends and enable programmatic interaction with the network.

Many common functionalities are provided out-of-the-box by native programs. For example, Solana does not require developers to deploy code to create a token. Instead instructions are sent to a pre-deployed native program that will set an account to store the token's metadata, effectively creating a new token.

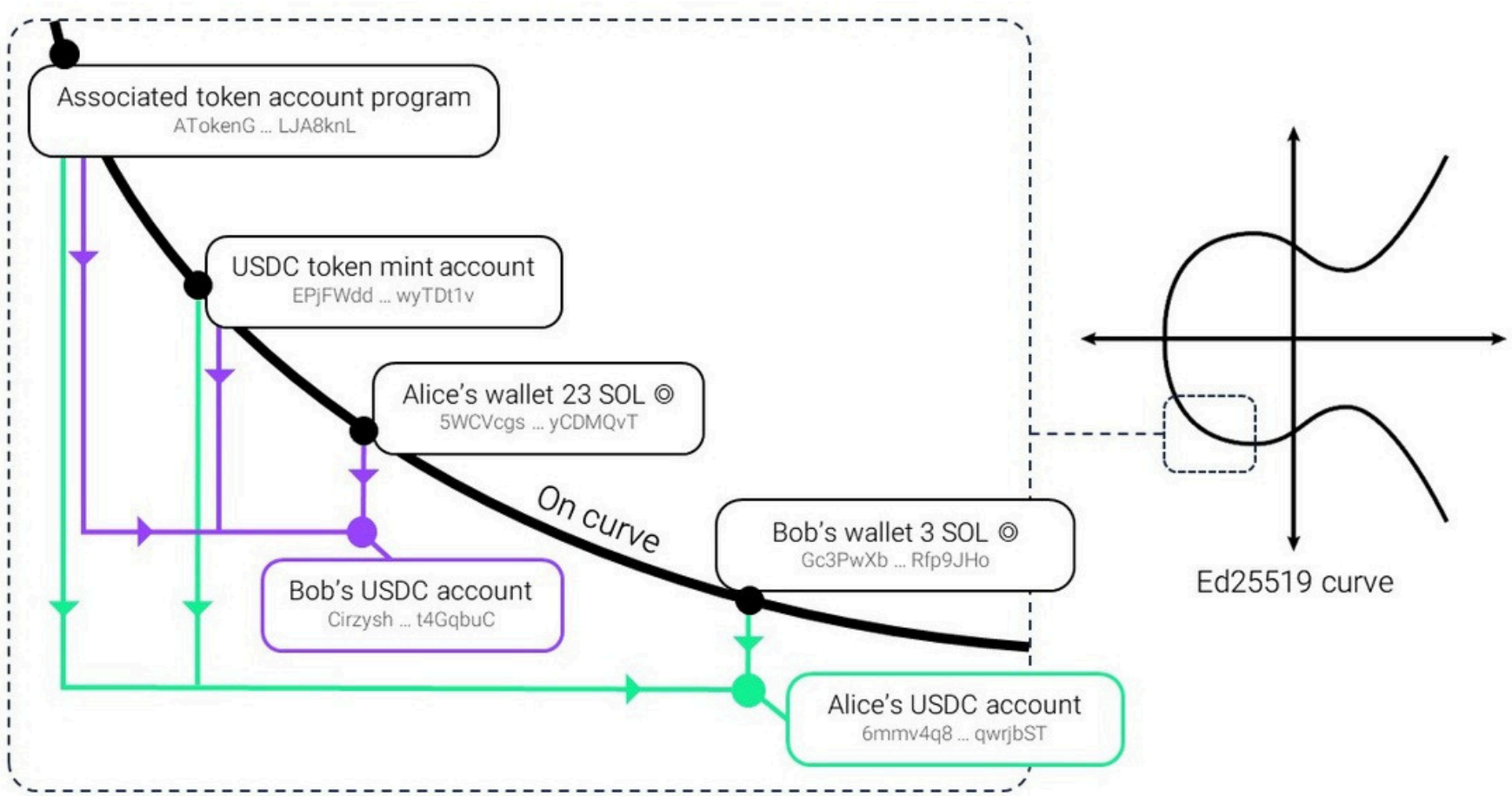
Rent is a mechanism designed to incentivize users to close accounts and reduce state bloat.

To create a new account, a minimum balance of SOL, known as the "rent-exempt" amount, must be held by the account. This can be considered a storage cost incurred to keep the account alive in a validator's memory. If the size of the account's data increases, the minimum balance rent requirement increases proportionally. When an account is no longer needed, it can be closed, and the rent is returned to the account owner. For example, if a user holds a dollar-denominated stablecoin, this state is stored in a token account. Currently, the rent-exempt amount for a token account is 0.002 SOL. If the user transfers their entire stablecoin balance to a friend, the token account can be closed, and the user will receive back their 0.002 SOL. Programs often handle account closures automatically for users. Several applications are available to help users clean up old, unused accounts and reclaim the small amounts of SOL stored in them.

Ownership Solana's ownership model enhances security by restricting who can modify an account's data. This concept is crucial for enforcing rules and permissions on the Solana blockchain. Every account has a program "owner". The owner of an account is responsible for governing it, ensuring that only authorized programs can make changes to the account's data. A notable exception to this rule is the transfer of lamports (the smallest unit of SOL), where increasing an account's balance is universally permitted, regardless of ownership.

Storage of State Solana programs, being read- only executable files, must store state using “Program Derived Addresses” (PDAs). PDAs are special types of accounts associated with and owned by a program rather than a specific user. While normal Solana user addresses are derived from the public key of an Ed25519 key pair, PDAs do not have a private key. Instead, their public key is derived from a combination of parameters—often keywords or other account addresses—along with the program ID (address) of the owning program.

PDA addresses exist "off-curve," meaning they are not on the Ed25519 curve like normal addresses. Only the program that owns the PDA can programmatically generate signatures for it, ensuring that it’s the only one that can modify the PDA's state.



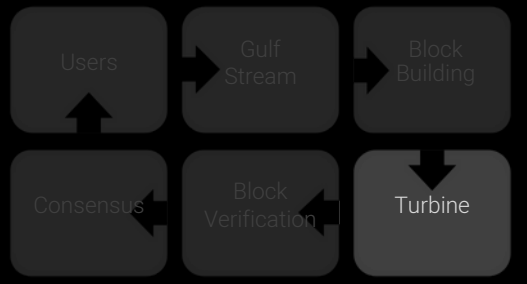
Above: Solana token accounts are specific examples of Program Derived Addresses (PDAs). They are used to hold tokens and live "off-curve." The Associated Token Account (ATA) program ensures that each wallet can only have one associated token account for each token type, providing a standardized way to manage token accounts.

VII

Turbin3 adopted our Solana name from the powerful shred propogation engine that is at the heart of the SVM.

Global State Machine Synchronized at the Speed of Light

Turbine



Key concepts: erasure coding, shreds, turbine tree

TURBIN3

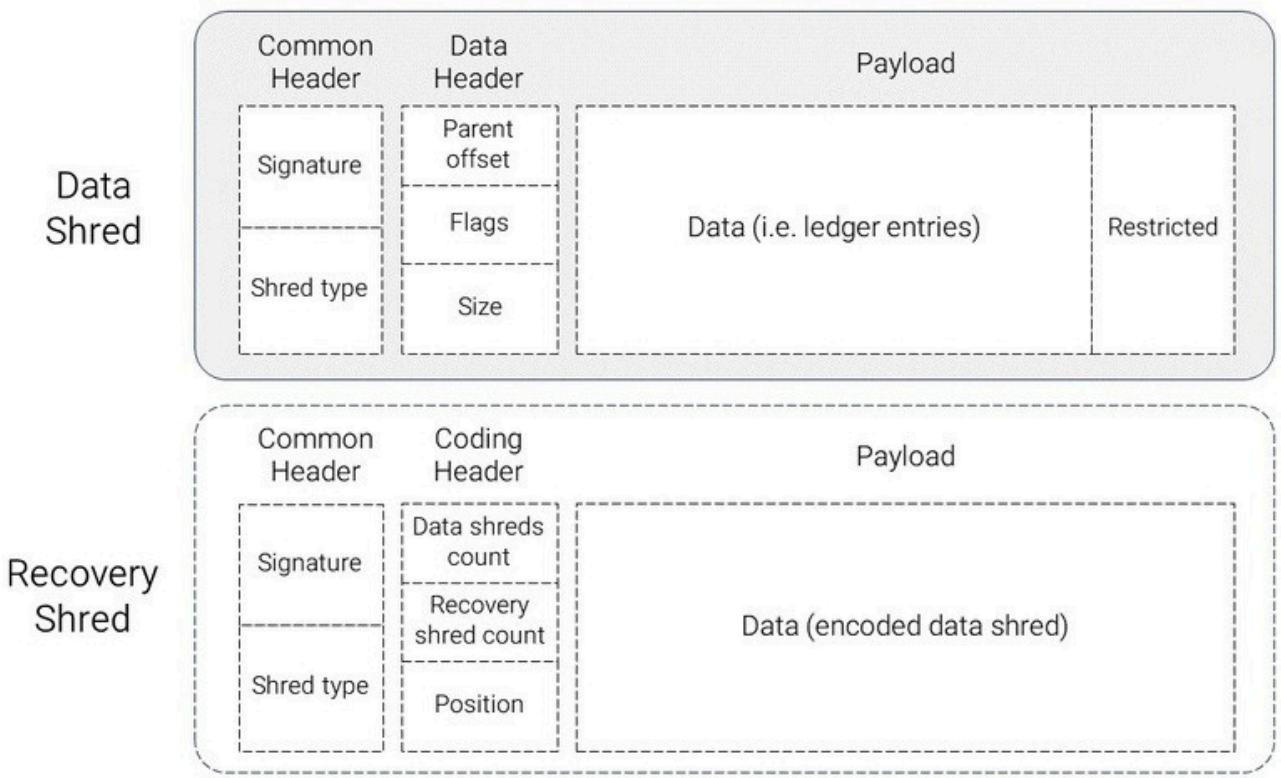
During the banking stage, transactions are organized into entries and sent to the Proof of History stream for timestamping. The block's bank is updated, and the entries are now ready for the next phase—Turbine. Turbine is the process through which the leader propagates their block to the rest of the network. Inspired by BitTorrent, it is designed to be fast and efficient, reducing

communication overhead and minimizing the amount of data a leader needs to send. Turbine achieves this by breaking down transaction data into "shreds" through a process called "shredding." Shreds are small packets of data, similar to individual frames in a video stream. When reassembled, these shreds allow validators to replay the entire block. The shreds are sent over the internet between validators using UDP and utilize erasure coding to handle packet loss or malicious dropping of packets.

Erasure coding, a polynomial-based error detection and correction scheme, ensures data integrity. Even if some shreds are lost, the block can still be reconstructed.

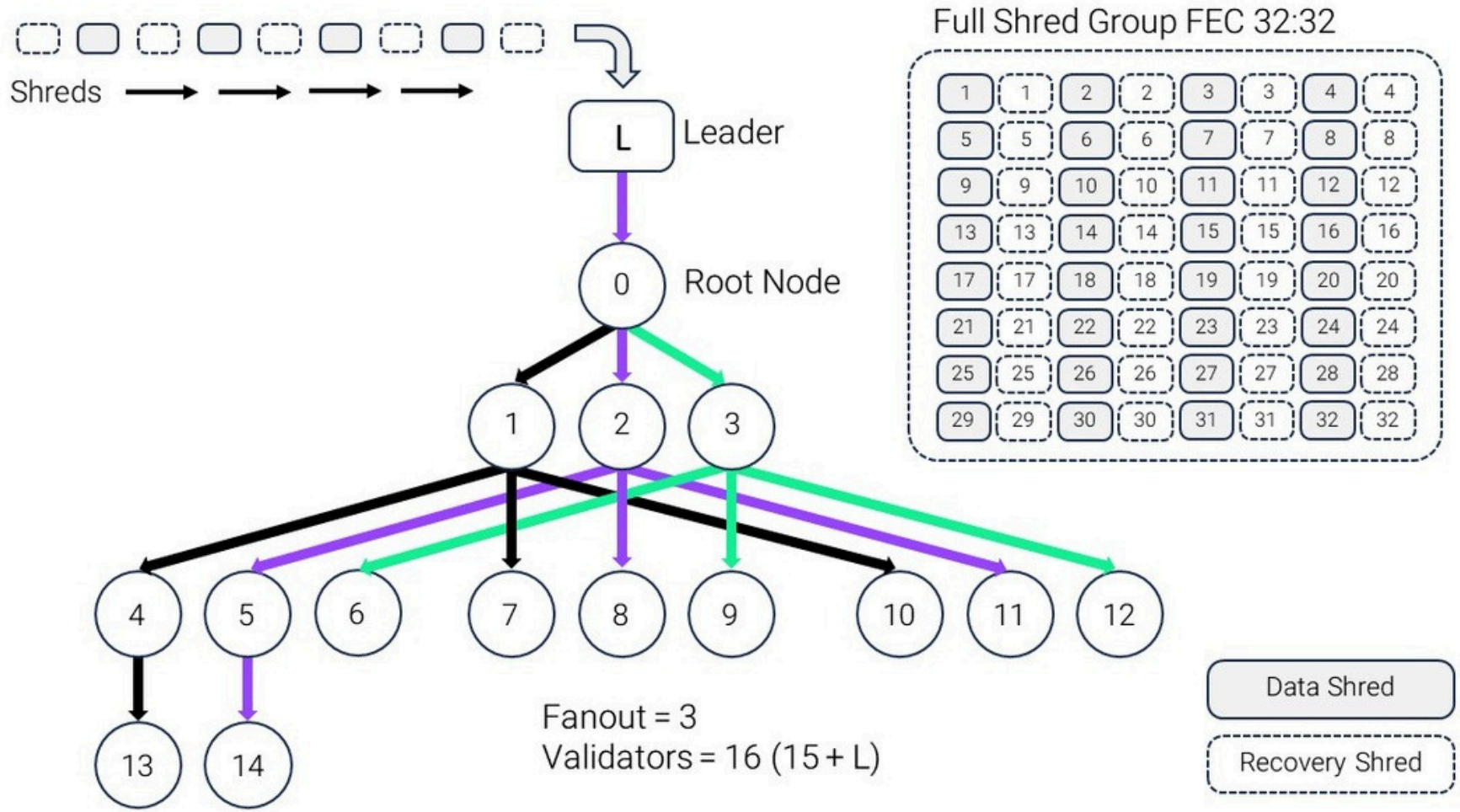
Shreds are grouped into batches known as forward error correction (FEC) batches.

By default, these batches consist of 64 shreds (32 data shreds + 32 recovery shreds). Data recovery occurs per FEC batch, meaning that up to half the packets in a batch can be lost or corrupted, and all the data can still be recovered. Each 64 shred batch is merkelized with the root being signed by the leader, and chained to the previous batch. This process ensures that shreds can be securely obtained from any node in the network that possesses them, as the chain of Merkle roots provides a verifiable path of authenticity and integrity.



The leader initially broadcasts to a single root node, which disseminates the shreds to all other validator nodes. This root node changes with each shred. Validators are organized into layers, forming the "Turbine Tree." Validators with a larger stake amount are typically positioned toward the top of the tree, while those with lower stakes are placed toward the bottom.
The tree (see next slide) usually spans two or three hops, depending on the number of active validators.

For security reasons, the order of the tree is rotated for each new batch of shreds. The primary goal of such a system is to alleviate the outbound data egress pressure on the leader and root node. By utilizing a system of transmit and retransmit, the load is distributed between the leader and the retransmitters, reducing the strain on any single node.



Above: For visual simplicity, a fanout of 3 is depicted, but Solana's actual fanout value is currently set to 200.

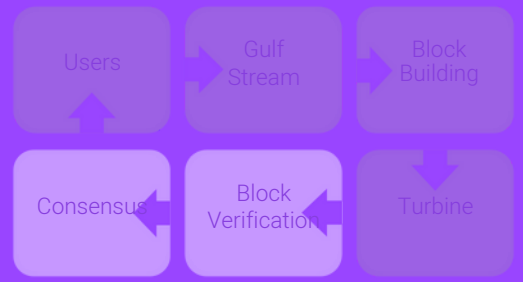
VIII

"Some smart people tell me there is an earnest smart developer community in Solana… I hope the community gets its fair chance to thrive"
— Vitalik Buterin, Ethereum co-founder

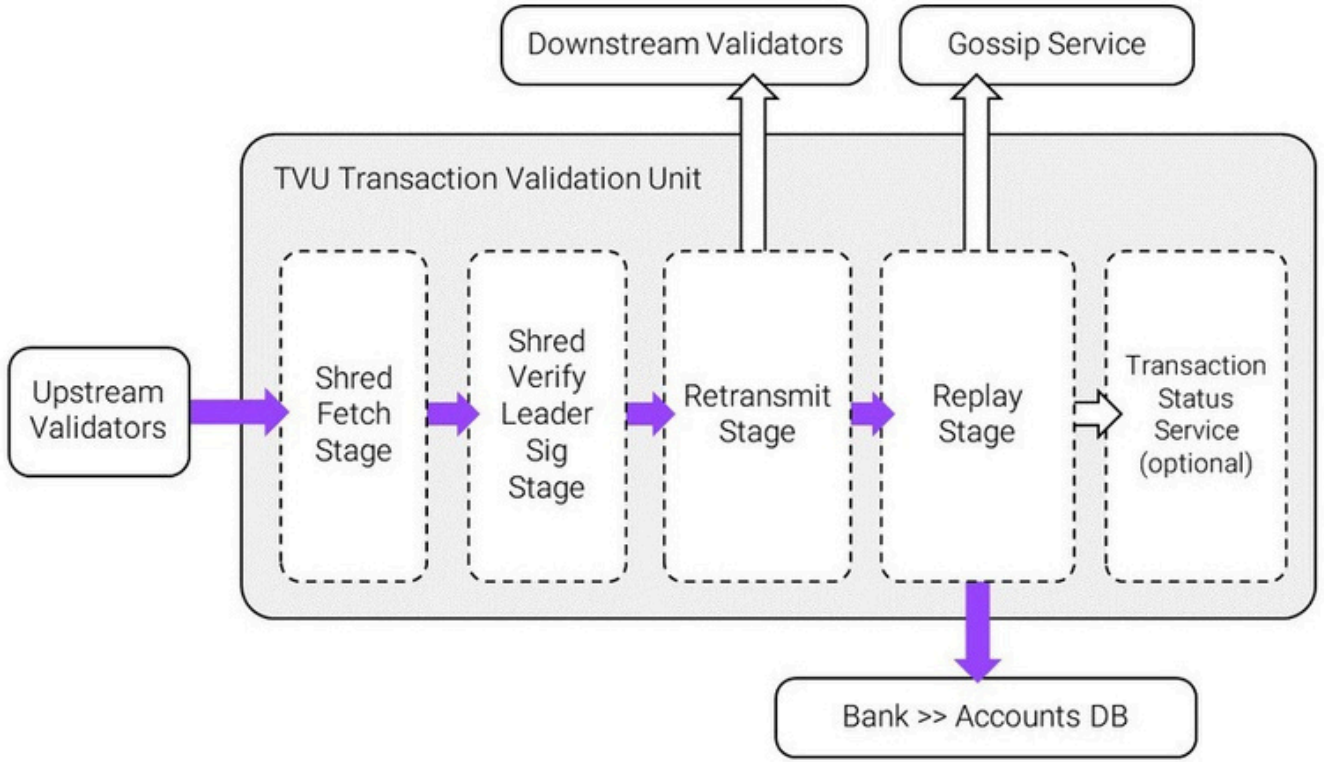
Global State Machine Synchronized at the Speed of Light

Consensus

Key concepts: Tower BFT, forks, TVU

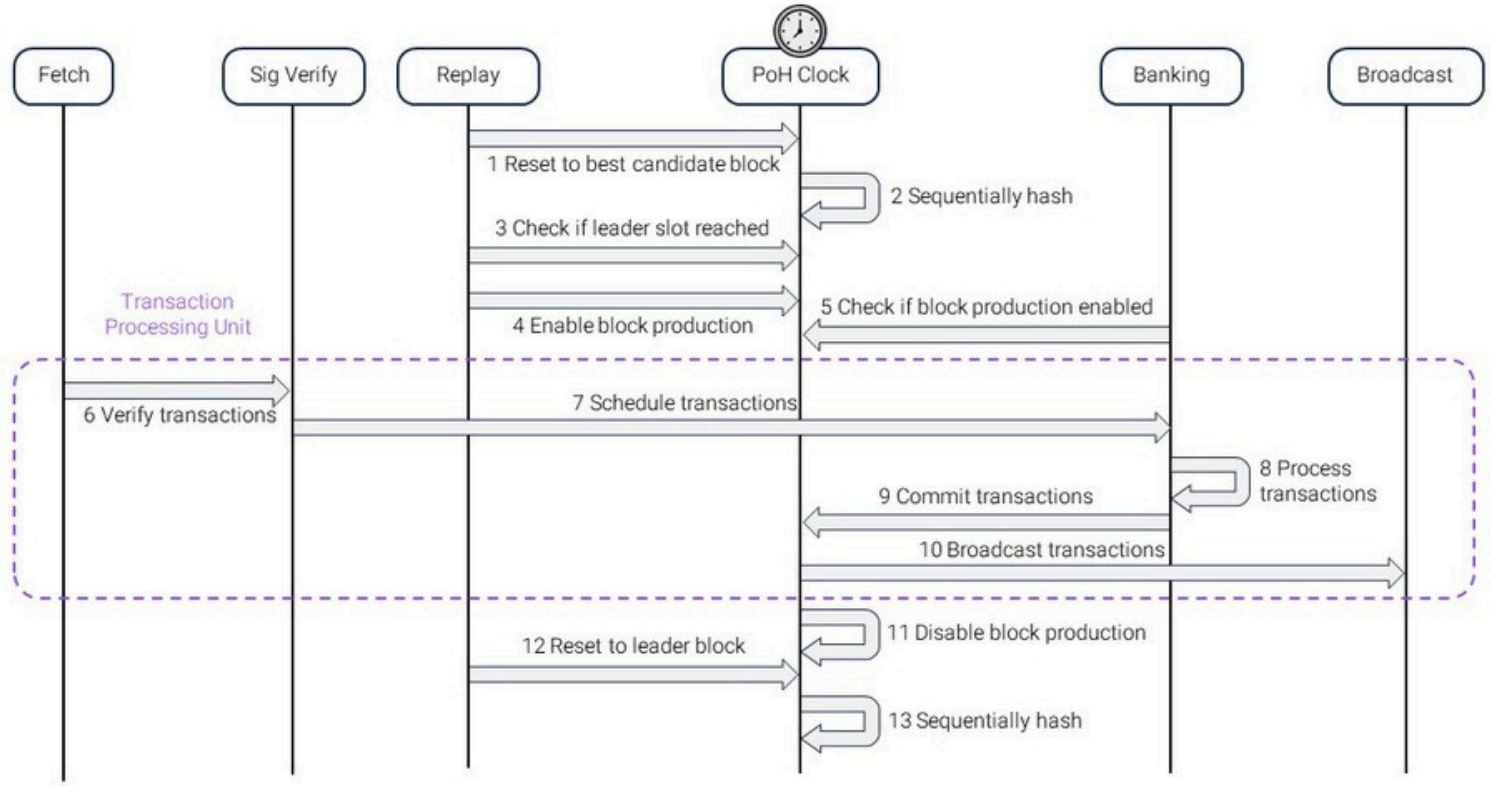


TURBIN3



Once a validator receives a new block from the leader via Turbine, it must validate all transactions within each entry. This involves replaying the entire block, validating the PoH hashes in parallel, recreating the transactions in the sequence dictated by PoH, and updating its local bank. This process is handled by the Transaction Validation Unit (TVU), which is analogous to the leader's Transaction Processing Unit (TPU), serving as the core logic responsible for processing shreds and block validation. Like the TPU, the TVU flow is broken down into several stages, starting with the Shred Fetch Stage where shreds are received over Turbine.

Original visual: Justin Starry, Anza



Above: the replay stage is responsible for switching the validator into leader mode and beginning block production.

In the subsequent Shred Verify Leader Signature Stage, the shreds undergo multiple sanity checks, most notably the verification of the leader's signature, which ensures that the received shreds originated from the leader.
The Retransmit stage, is where the validator forwards the shreds to the appropriate downstream validators.

In the Replay Stage, the validator recreates each transaction exactly and in the correct order while updating its local version of the bank. The Replay Stage is analogous to the banking stage in the TPU; it is the most important stage and can be more directly described as the block validation stage. Replay is a single threaded process loop that orchestrates many key operations, including voting, resetting the PoH clock, and switching banks.

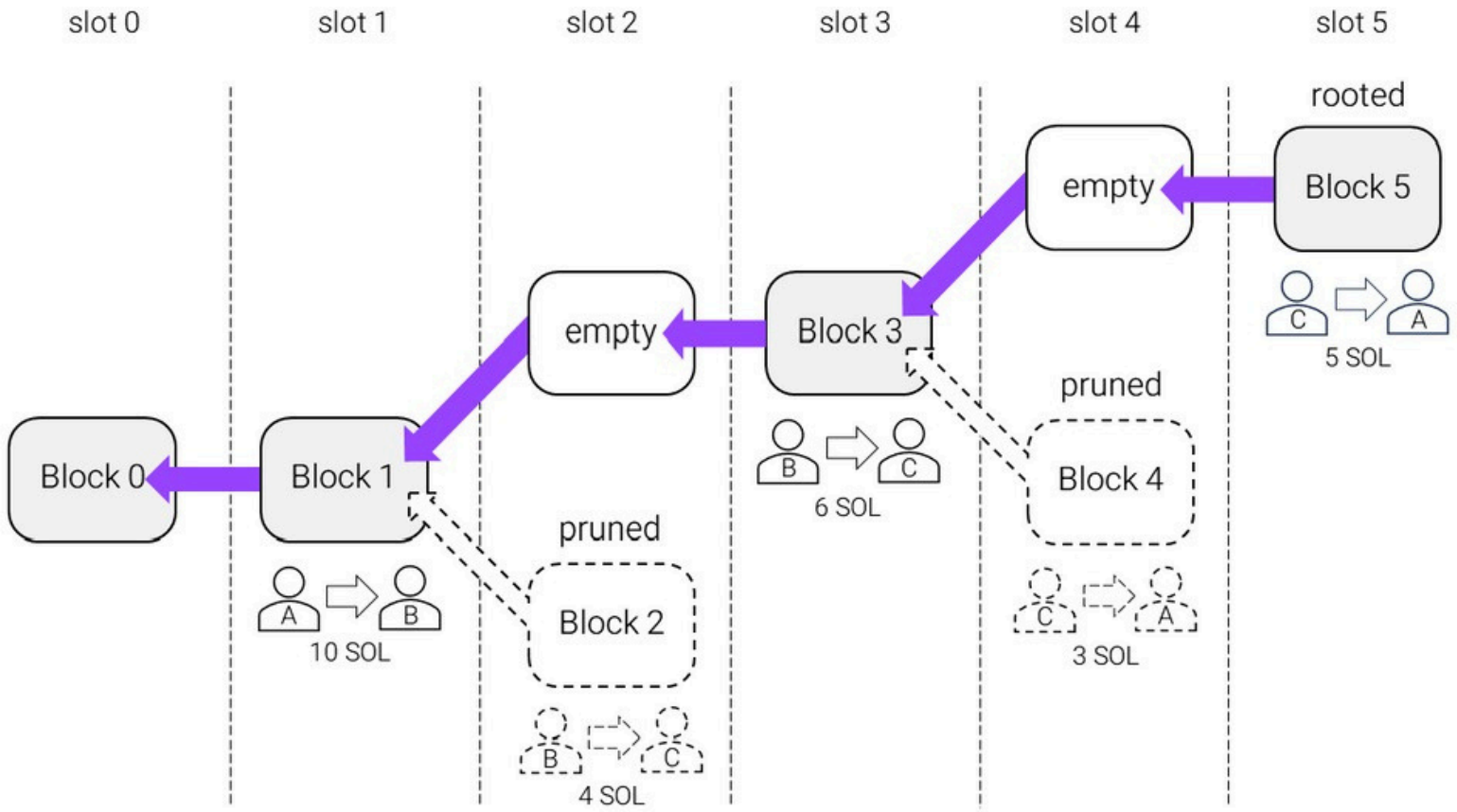
To achieve consensus, Solana uses Tower BFT (TBFT), a custom implementation of the well-known Practical Byzantine Fault Tolerance (PBFT) algorithm, commonly used by most blockchains to agree on the state of the chain. Like all blockchains, Solana assumes the presence of malicious nodes in the network, so the system must withstand not only node failures but also certain levels of attacks.

Tower BFT differentiates itself from other chains by leveraging the synchronized clock provided by Proof of History. While traditional PBFT requires multiple rounds of communication to agree on the transaction order, Solana nodes utilize the pre-established order of events, significantly reducing the messaging overhead.
To participate in consensus and earn rewards, validators submit votes for blocks they believe are valid (i.e., free of issues like double spends or incorrect signatures) and should be considered canonical. Validators pay a transaction fee for these votes, which are processed by the leader and included in a block along with regular user transactions. This is why Solana transactions are often categorized into vote and non-vote transactions. When validators submit a correct and successful vote, they earn a credit.

This mechanism incentivizes validators to vote on the fork they believe has the best chance of being included, i.e., the “heaviest” fork.

Forks

Part of Solana’s design, which makes it so fast, is that the network doesn’t wait for all validators to agree on a newly produced block before producing the next one. As a result, it's not uncommon for two different blocks to be linked to the same parent block, creating forks.



Above: once a block becomes rooted, account updates from earlier banks that are not ancestors of the finalized bank are pruned

Solana validators must vote on these forks and use a consensus algorithm to determine which one to adopt. When competing forks exist, only one fork will ultimately be finalized by the network, while blocks in discarded forks are abandoned.

Each slot has a predetermined leader, and only that leader's block will be accepted for the slot; there cannot be two proposed blocks for a single slot. Therefore the number of potential forks is limited to a "there/not-there" skip list of forks that can emerge at the boundaries of leader rotation slots.

Once a validator chooses a fork, it is committed to that fork until a lockout time expires, meaning they must stick with their choice for a minimum period. The Solana "skip rate"—the percentage of slots in which a block was not produced—varies typically from 2% to 10%, with forks being the primary reason for these skipped slots.

Other possible reasons for skipped slots include the start of a new epoch, a leader being offline or the production of an invalid block. For every block, Solana uses a bank to access the state at that block. When a bank is finalized, the account updates from that bank and its ancestors are flushed to disk. Additionally, any account updates from banks that are not ancestors of the finalized bank are pruned. This process allows Solana to maintain multiple potential states efficiently.

The status of a transaction on Solana varies depending on its current stage in the consensus process:

- Processed: The transaction has been included in a block.
- Confirmed: The transaction's block has been voted on by a two-thirds supermajority.
- Finalized: More than 31 blocks have been built on top of the transaction's block.

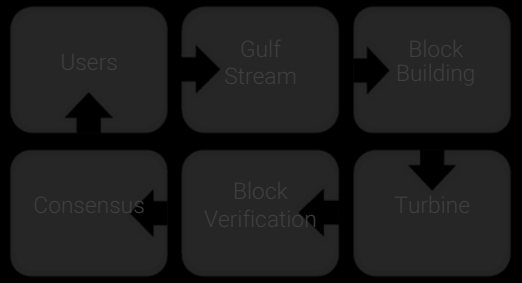
To date, there has never been an instance in Solana's history where an (optimistically) confirmed block did not become finalized.

IX

"A blockchain requires a clever combination of cryptography, distributed systems, operating systems and programming languages. Solana's superpower was the willingness to run away screaming from the most interesting problems in each discipline." — Greg Fitzgerald, Solana co-founder

Global State Machine Synchronized at the Speed of Light

Gossip & Archive

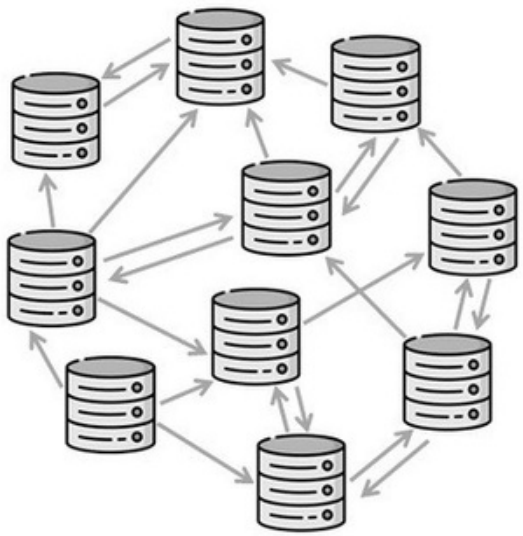


TURBIN3

The gossip network can be thought of as the control plane for the Solana network. Unlike the data plane, which handles transaction flows, the control plane disseminates crucial metadata about the blockchain's state, such as contact information, ledger height, and vote information.

Without gossip, validators and RPCs wouldn't know which addresses and ports are open for communication across various services. New nodes also rely on gossip to join the network. Solana's gossip protocol uses informal, peer-to-peer communication with a tree broadcast approach inspired by a modified PlumTree algorithm. This method efficiently propagates information without relying on any central source. Gossip operates somewhat as an isolated system, independent from most other validator components. Validators and RPCs share signed data objects every 0.1 seconds over UDP via gossip, ensuring information availability across the network. All gossip messages must be less than or equal to the maximum transmission unit (MTU) of 1280 bytes, referred to as the "packet struct" in the codebase.

Gossip records are the actual data objects shared between nodes. There are approximately 10 different types of records, each serving different purposes. Gossip records are signed, versioned, and timestamped to ensure integrity and currency.



Gossip data is stored in a Cluster Replicated Data Store (CrdsTable). This data structure can grow very large and needs to be periodically pruned.

There are four types of gossip messages: •Push: The most common messages, sharing information with a subset of "push peers." •Pull & Pull Response: Periodically checks for missed messages, with pull responses sending back the information that nodes don't have. •Prune: Allows nodes to selectively reduce the number of connections they maintain. •Ping & Pong: Health checks for nodes —if a ping is sent, a pong is expected back, indicating the peer node is still active.

Archive

Solana stands out from other blockchains by not requiring the entire history to determine the current state of an account. Solana's account model ensures that the state at any given slot is known, allowing validators to store the current state of each account without processing all historical blocks. RPCs and validators, by design, do not retain the entire historical ledger. Instead, they typically store only 1 or 2 epochs' (2-4 days) worth of transaction data, which is sufficient to validate the tip of the chain.

Archives are currently managed by "warehouse nodes," operated by professional RPC service providers, the Solana Foundation, and other ecosystem participants interested in ensuring transaction history is available.

Warehouse nodes usually maintain one or both of the following:

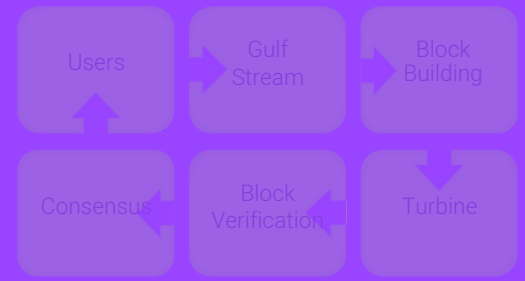
- Ledger Archive: Uploads raw ledger and AccountsDB snapshots suitable for replaying from scratch.
- Google Bigtable Instance: Stores block data from the genesis block onward, formatted to serve RPC requests.

X

"People are realizing that Solana is the only chain available today that will support mainstream consumer apps."

— Ted Livingston, founder Code

Global State Machine Synchronized at the Speed of Light

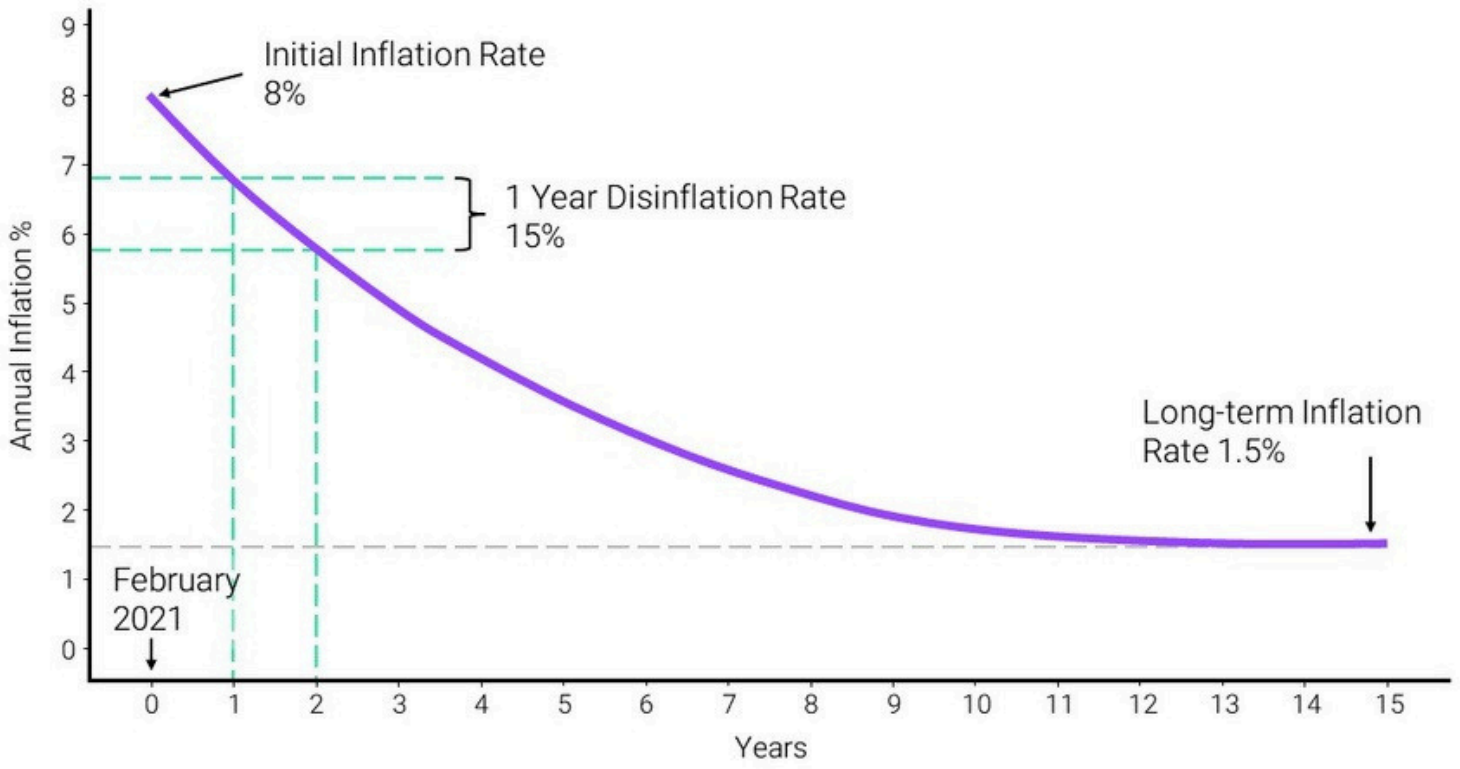


Economics & Jito

Key concepts: staking, mem-pool, bundles, tips

TURBIN3

Solana employs inflation to distribute staking rewards by generating new SOL tokens each epoch. This process causes non-stakers' network share to decrease relative to stakers, leading to a wealth transfer from non-stakers to stakers. Inflation started in early 2021 at an initial rate of 8% which decreases by 15% annually until it stabilizes at a long-term rate of 1.5%. Any SOL token holder can earn rewards and help secure the network by staking tokens to one or more validators.



Assigning tokens to a validator is known as delegating. Delegating tokens to a validator indicates trust in the validator. However, it does not give the validator ownership or control over the tokens. All staking, unstaking, and delegation actions are executed at the beginning of the next new epoch.

When a validator submits a vote they earn a credit if the vote is accurate and successful. Voting transactions cost 0.000005 SOL and are exempt from priority fees. Voting expenses amount to roughly 1 SOL per day per validator, making it the main operational cost of running a validator. Throughout an epoch, validators accumulate credits from voting, which they can exchange for a share of the inflation at the end of the epoch.

The top-performing validators successfully vote on approximately 90% of slots. Note that the percentage of slots without blocks (skipped slot rate) ranges from 2% to over 10%, and these slots cannot be voted on. The average validator votes successfully on about 80% of slots, earning 345,600 credits in an epoch of 432,000 slots. The total inflation pot is first divided based on the credits earned for the epoch. A validator's share of the total credits (their credits divided by the sum of all validators' credits) determines their proportionate reward. This is further weighted by stake.

Therefore, a validator with 1% of the total stake should earn roughly 1% of the total inflation if they have an average number of credits. If they have above or below the average number of credits, their rewards will fluctuate accordingly.

Differences in voting performance is one reason why the returns (measured in APY) that validators offer to stakers vary.

Another factor is the commission rate charged by validators, which is a percentage of the total inflation rewards directed to validator their Additionally, a validator being offline or out of sync with the blockchain (known as delinquency) significantly impacts returns. Validators designated as the leader for block a receive specific additional block rewards.

These rewards comprise 50% of the base fees and 50% of the priority fees of all transactions within the block, with the remaining fees being burned.
Only the validator who produced the block receives these rewards. Unlike staking rewards, which are distributed per epoch, block rewards are instantly credited to the validator's identity account when the block is produced.

Liquid staking

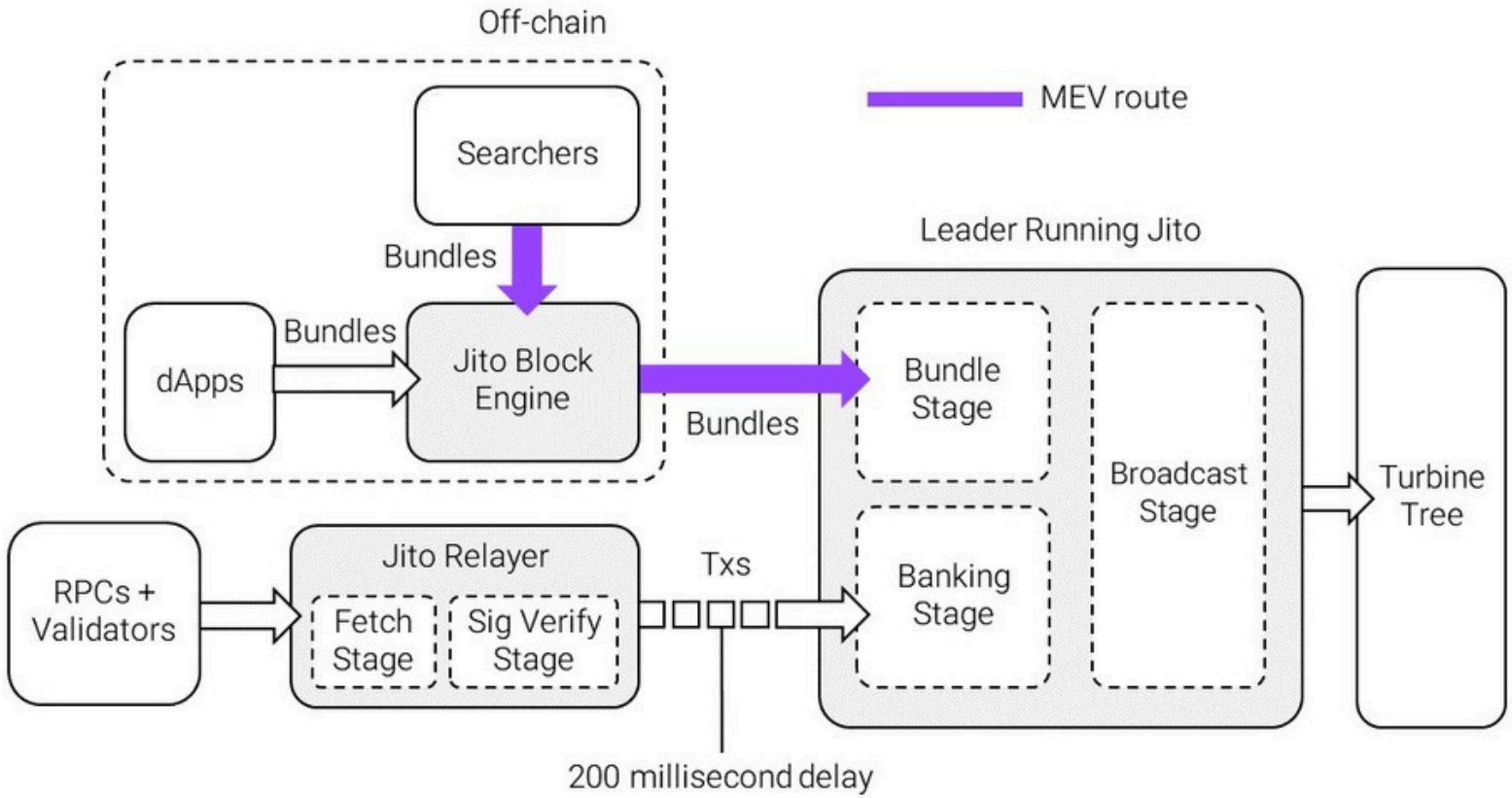
Liquid staking has become a popular alternative to native staking. Participants receive a token, known as a Liquid Staking Token (LST) or Liquid Staking Derivative (LSD), in return for staking their SOL, usually in a stake pool that delegates their tokens across multiple validators.
The newly received LST tokens represent the user's share of the staked SOL.

These tokens can be traded back and forth, or used across Solana's defi ecosystem, while still earning staking rewards, significantly enhancing capital efficiency.
With traditional native staking, over time, the staker will directly accrue more SOL. Whereas with liquid staking, rewards are reinvested back into the pool increasing the fair value of the LST. As long as there is a mechanism to redeem LSTs for the underlying staked SOL, arbitrage traders will ensure that the token price remains rational.

Price of LST = (total staked SOL in pool * price of SOL) / total LST minted

Jito

As of writing, over 80% of the stake on Solana utilizes the Jito client validator software. This client, a fork of the original Agave client, introduces an out-of-protocol blockspace auction that provides validators with additional economic incentives through tips. This extra incentive is a major factor in the widespread adoption of the Jito client among validators. When leaders use the Jito client validator, their transactions are initially directed to the Jito-Relayer.



This open-source software functions as a transaction proxy router. Other network nodes remain unaware of the Jito-Relayer's existence, as they simply send transactions to the address and port configuration the leader has advertised over the gossip network as their ingress_socket, assuming it to be the leader's.
The relayer holds all transactions for 200 milliseconds before forwarding them to the leader. This "speed bump" mechanism delays incoming transaction messages, providing a brief window for conducting auctions.

After 200 milliseconds, the relayer optimistically releases the transactions regardless of the auction results.
Blockspace auctions occur off-chain via the Jito Block Engine, allowing searchers and applications to submit groups of atomically executed transactions known as bundles. These bundles typically contain time-sensitive transactions such as arbitrages or liquidations. Jito charges a 5% fee on all tips, with a minimum tip of 10,000 lamports.

Tips operate entirely out-of-protocol, separate from in-protocol priority and base fees. Previously, Jito operated a canonical out-of-protocol mem-pool service, which has now been deprecated.

TURBIN3

Turbin3 is the Solana arm of the Web3 Builders Alliance, an education, research, and development organization serving the Solana Ecosystem. The program is designed to enhance developer onboarding and retention within the Solana ecosystem by providing comprehensive training, resources, and hands-on project experience. Turbin3 offers structured courses, mentorship, and community support to help developers navigate the complexities of Solana development.
https://turbin3.com/

Solana

How it works

An executive overview of the Solana protocol