y

ackee

Solana Handbook

An introduction to Solana

March 2024

by Ackee

Revision 1.4

Content

Document revisions

1.0	First version of the document published	Jun 15, 2022
1.1	Added Chapters 6 and 7	Aug 30, 2022
1.2	Renaming and change of content in Chapter 2	Sep 27, 2022
1.3	Formatting and updated chapters Ecosystem, Gaming, Non-Fungible-Tokens and SPL	Aug 29, 2023
1.4	Token 2022 and token extensions	March 22, 2024

About

Why Solana Handbook?

Solana was introduced by Anatoly Yakovenko in 2017 when the Solana Whitepaper [10] was published. However, the whitepaper is no longer up-to-date and we haven't found documentation that would meet our needs - to provide study material for the School of Solana.

That is why this handbook was created. Its purpose is to introduce the reader to the Solana blockchain. It should serve as an entry point for new Solana developers or blockchain enthusiasts wishing to learn about Solana.

This handbook is structured as follows: Beginning with Chapter 1, Blockchain Terminology, it clarifies fundamental concepts. Chapter 2, Solana Introduction, provides initial insights into Solana. Chapter 3, Solana Core Concepts, delves deeply into the platform's foundational ideas. Chapter 4, the Programming Model, talks about its operational framework. Continuing, Chapter 5, Solana Program Library, discusses available resources provided by the Program Library. Appendices cover various domains: Appendix A - Ecosystem, introduces you into the Solana Ecosystem, Appendix B - NFT, provides technical insights into NFTs, and Appendix C - Gaming, details about upcoming Web3 and Games. This comprehensive handbook functions as an ideal primer for those embarking on a journey to understand Solana. We wholeheartedly welcome any input, be it identifying errors or suggesting expansions, to enhance this resource.

Authors

This handbook was created with love by Ackee Blockchain, its authors are Andrej Lukačovič, Andrea Nováková, Tibor Tribus, Vladimír Marcin and Ondřej Řeháček. It is based

on a master thesis of Lukáš Kozák under the supervision of Josef Gattermayer, Ph.D., assistant professor at the Faculty of Information Technology, Czech Technical University in Prague and CEO of Ackee Blockchain.

Ackee Blockchain is a blockchain security company founded in 2021, specializing in audits and other security assessments. Its team of experts performs Ethereum and Solana audits and develops open-source security frameworks Wake and Trident.

By running free certification courses: the School of Solana and the School of Solidity, Ackee Blockchain contributes to a stronger blockchain ecosystem by sharing knowledge.

1. Blockchain terminology

This section introduces blockchain technology and explains key terminology.

Blockchain

A blockchain can be thought of as a series of blocks or an append-only data structure that resembles an ordered back-linked linked list, which uses hashes as pointers to previous blocks (Figure 1.1). This structure consists of blocks that form a chain, hence the term blockchain. It can be easily concluded that it is a very simple data structure.

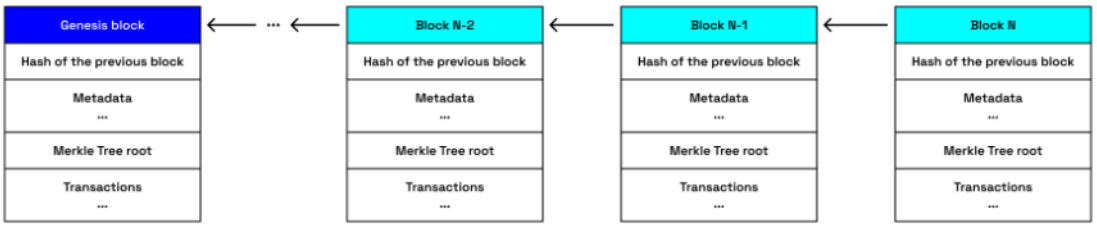

Figure 1.1: General Scheme of Blockchain [4]

Block

Block is a data structure that contains a header comprising three items – the hash of the previous block's header, metadata and a Merkle root [5]. Metadata depends on the protocol. The Merkle root is a root of the well-known Merkle tree, which can be used to verify later that transactions in a block have not been tampered with. After the header comes the core part of the block, the transaction.

Transaction

Transaction is a protocol-defined message that is stored as a part of a block, which is then stored as a part of a blockchain. The content usually consists of some kind of value transfer or on-chain program execution. Transactions are cryptographically signed by

their authors, proving their authenticity. In the case of a value transfer, ownership of the funds or tokens often represents some value in the real world.

Protocol

Protocol is a common set of rules network nodes must follow. It defines things like communication between P2P (Peer-to-Peer) nodes, transaction format for everyone intending to use the network, any special features, and everything else for the network to operate correctly and for the users to know how to transact over the network. An essential part of a good protocol for a decentralized blockchain network is the proper incentive setup; this creates the need for its native coin.

Coin

Coins motivate participation in the network. They are usually paid with every new block to miners or validators for their help in securing the network. Without the proper incentives, any decentralized blockchain network falls apart.

Nodes

Node is a term from graph theory or distributed systems; it is a single participant in a network. The nodes communicate with each other according to the protocol and in a P2P manner forming the whole blockchain network. There might be more types of nodes that are not equal, e.g., validator nodes securing the network or pure RPC (Remote Procedure Call) nodes used only to query the network and post new transactions. Their functions may overlap.

Consensus

To agree on a certain state of a blockchain, network nodes need to reach a consensus. We assume there are malicious nodes in the network. Therefore, the system must be able to withstand not only simple node failures but also attacks to a certain extent. BFT (Byzantine Fault Tolerant) is thus a desired property of such a distributed system.

Currently, only three viable consensus families can be used in practice. The first is the classic PBFT-like (Practical BFT) algorithm family [6]. The second is a so-called Nakamoto

consensus, which couples a Sybil protection mechanism of Proof-of-Work with the longest-chain rule, a novel consensus invented by Satoshi Nakamoto for Bitcoin in 2008 [2]. The third and newest family of consensus protocols known today is called Snow. Yet it is better known by its implementation name – Avalanche Consensus [7], introduced in 2018 and used for the Avalanche cryptocurrency.

Sybil Resistance

To prevent a single entity from taking over the network, there must be a mechanism put in place so that no one can just spawn more nodes that can mine or vote (depending on the network) and thus subvert the network reputation system. These dishonest nodes would be able to out-vote honest nodes and start censoring transactions, approving invalid transactions, or changing the entire protocol.

Currently, the two most common Sybil resistance mechanisms are PoW (Proof-of-Work) and PoS (Proof-of-Stake). The former employs a model where miners in the network are given a chance to mine a block that is proportional to their hashing power in the network and is used in Bitcoin [8]. The latter is a new type of model for voting-based networks, where a validator is given the power of their vote proportionally to staked coins.

Security

Consensus and Sybil resistance mechanism are often confused as the same thing, which is not true and is worth pointing out. One works in conjunction with the other. Let's look at how this works in both PoW-based and PoS-based networks.

Consider what makes Bitcoin, a PoW-based network, theoretically secure – it is the fact that only the longest chain is respected, also commonly known as the longest chain rule. This is why the consensus is called, as mentioned before, the Nakamoto consensus.

For a PoS-based network, the Sybil resistance mechanism is usually associated with a variant of a PBFT-like algorithm or the novel Avalanche consensus.

Smart contracts

A smart contract is a piece of code deployed on a blockchain with a cryptographically signed transaction. Users can then interact with it by sending transactions that invoke a specific function defined in the smart contract and the business logic is executed as stated in the deployed code [9].

Data relevant to the smart contract state are also stored on the blockchain. Hence we can look at smart contracts as programs on a decentralized computer that access files in its file system and modify them according to the predefined rules. If such a contract is made immutable, we can trust that the smart contract will not do anything else than what it is supposed to do.

It is worth noting that apart from storing the blockchain itself, each node creates a state as a result of transaction execution. The final state is the result of all processed transactions and can always be deterministically recreated from the blockchain history.

Code is compiled for a predefined ISA (Instruction Set Architecture) and executed in a VM (Virtual Machine) which understands it. The mentioned VM is a special runtime environment similar to well-known VMs such as JVM (Java Virtual Machine) or CLR (Common Language Runtime) from Microsoft's .NET ecosystem. The most commonly known VM for smart contracts, which is used by Ethereum [3], is EVM (Ethereum Virtual Machine); it includes its very own instruction set specialized for the needs of smart contracts.

Only transactions involving smart contract execution need to be processed by the VM. The standard execution path is to prepare the relevant smart contract data and smart contract byte code, launch the VM with said data and code, and observe possible failures. If the execution succeeds, the changes to the smart contract data made in the VM are taken, and the state outside the VM is changed; otherwise, the changes are discarded and the next transaction continues.

2. Solana introduction

Solana was founded by Anatoly Yakovenko in 2017 when the Solana Whitepaper [10] was published. It describes a novel clock mechanism for distributed systems called PoH (Proof-of-History) as a technique for keeping time between computers that do not trust each other. They were able to demonstrate this mechanism [11] on a testnet with a gigabit network and 150 nodes processing an *average* of 200 thousand TPS (transactions per second) with bursts over 500 thousand. Compared with Bitcoin's maximum throughput of 7 TPS and Ethereum's maximum capacity of 15 TPS [12].

Since this proof of concept, Solana has been developed into a fully functional blockchain smart contract platform and strives towards adoption.

Solana's main value proposition is solving the blockchain trilemma [1], i.e., delivering scalability, decentralization, and security without sacrificing any of the three mentioned features.

Solana is a single-chain blockchain using a slightly changed PBFT consensus called Tower BFT with Proof-of-Stake as a Sybil protection mechanism.

Leaders are known one full epoch in advance; their rotation is a function of the blockchain data. An epoch is a series of 432,000 slots, where the slot is a term for the time period the block is in the making by the leader.

The blocks are streamed as something called entries, so the creation of the block by the leader and the verification of the block by others can happen in parallel. Solana officially launched its mainnet, still labeled as beta, in March 2020. The native coin that incentivizes validator nodes and protects the network from spam by paying transaction fees with it is called SOL.

2.1. Comparison to Ethereum

Solana is a smart contract platform. Compared to Ethereum, smart contracts on Solana are called Programs. They can be executed in parallel. Parallelization is one of the key differences from other platforms. While Ethereum can be considered a single-threaded distributed computing platform, Solana can be viewed as a multi-threaded one.

Solana makes itself clear to focus on improving scalability from the engineering perspective. It is rethinking and reengineering core parts that were first seen in Ethereum and making them parallel and optimized, including the usage of Nvidia CUDA to speed up certain parts of the code and invent its own specialized horizontally scalable database system for state storage and other things that are supposed to make it possible to reach maximum TPS practically only bounded by the network throughput, memory throughput and the number of CUDA cores in modern Nvidia GPUs. Therefore over time, it should scale with better hardware available on the market and internet connectivity in the world.

2.2. Rust Development

Solana's ecosystem revolves around the Rust programming language and its ecosystem. The main and only implementation of the node software is written in it. Also, Solana programs are almost exclusively written in Rust. Although there is no technical barrier preventing from using C or C++, Rust is the most supported language for developing on Solana. All the libraries and supporting code that can be found are written in it, leaving practically no other option.

3. Core concepts

There are eight core concepts introduced in Solana that are supposed to make it as fast as developers claim. This section tries to cover them in as much detail as possible. Unfortunately, finding a proper explanation of some of the details is not always possible. Some of these are not yet or not fully implemented, so the source code does not answer the questions that arise while studying them.

3.1. Proof-of-History (PoH) – Virtual Clocks

Agreement on time in distributed systems has always been problematic. First, a high-level overview of this concept is described, followed by an in-depth description.

Solana leverages the so-called Proof-of-History (PoH) mechanism to synchronize local virtual clocks on all nodes [10]. PoH ensures that the timestamp in any message can be trusted and that any timeouts in the consensus protocol can be avoided because everyone knows the time and knows whether to start a new round of consensus or not. PoH allows minimizing the block time as there's no waiting overhead. In other words, thanks to synchronized clocks, communication can be replaced by local computation.

To prevent validators from skipping the validator that comes before them, PoH is used to force all validators to spend a minimum amount of time before they can even submit their block. Thus, if validator B follows validator A, B cannot attempt to skip A by chaining off its previous block because B has to run the Proof-of-History algorithm at least as long as A does, so A gets a fair chance to submit their block.

3.1.1. Verifiable Delay Function (VDF)

PoH is based on a Verifiable Delay Function (VDF). Specifically, Solana uses a recursive pre-image resistant SHA256 VDF, where the output of one SHA256 iteration is recursively used as the next iteration's input.

To create a block, the producer needs to compute the VDF with all new messages to be included in the block:

$$Message_1 \rightarrow Hash_1$$

$$Hash_1 + Message_2 \rightarrow Hash_2$$

$$Hash_2 + Message_3 \rightarrow Hash_3$$

...

$$Hash_{n-1} + Message_n \rightarrow Hash_n$$

Observations:

- From PoH, we have a proof of the Lower Bound on the time of $Message_i$ (i.e., $Message_i$ must have taken place after $Hash_{i-1}$).
- From PoH, we have a proof of the Upper Bound on the time of $Message_i$ (i.e., $Message_i$ must have taken place before $Hash_i$).
- Points 1 and 2 prove the exact order of the messages, which implies that VDF not only provides us virtual clocks, but everyone can trust the order of events.

Phases of PoH:

- Evaluation phase (leader): computation on only one CPU core as it is a strictly sequential computation by definition. This takes:

$$\frac{\text{Total number of hashes}}{\text{Hashes per second for 1 core}}$$

- Verification phase (voters): the block can be checked in parallel using GPU with thousands of cores as it can be easily sliced and the intermediate hashes are known; this takes:

$$\frac{\text{Total number of hashes}}{\text{Hashes per second for 1 core} * \text{Number of cores available}}$$

Thus, it can be concluded that PoH is difficult to produce but easy to verify. These are two important factors that are crucial for the use of PoH – it is not easy to falsify the PoH, but once it is finished, any validator can verify the results very quickly.

3.2. Tower BFT (TBFT) – PoH-based PBFT

As a consensus algorithm, Solana uses the Tower BFT (TBFT), which is a custom implementation of the well-known Practical Byzantine Fault Tolerance (PBFT) algorithm published in 1999 by Miguel Castro and Barbara Liskov [6].

PBFT consensus rounds are divided into three main phases (pre-prepare, prepare and commit); see Figure 1.2. A detailed description is beyond the scope of this handbook.

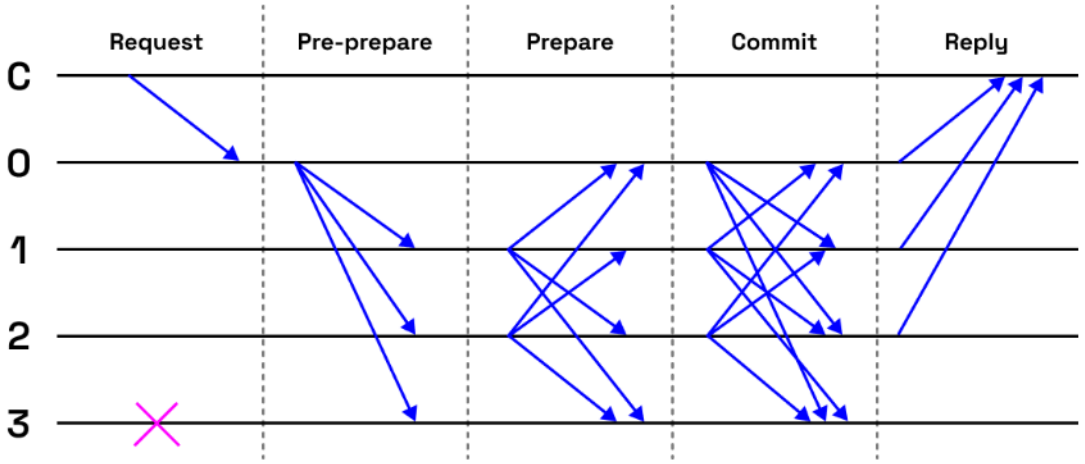

Figure 1.2: Normal operation of PBFT [6]

PBFT is focused on satisfying the properties of safety (results are valid and identical at all non-faulty nodes) and liveness (nodes that don't fail always produce a result). The safety guarantee is possible due to the deterministic nature of the process (executed on every node). The liveness guarantee is possible due to the View-change process. The network will not be stopped unless there are too many byzantine nodes. View-change allows nodes to switch leaders if they appear to be malicious or faulty.

3.2.1. View-change

View-changes are carried out when a leader appears to have failed, and so another node attempts to take his place by initiating an election process. It gets triggered by timeouts that prevent nodes from waiting indefinitely for requests to execute.

In addition, the timeout is postponed whenever the protocol detects that nodes are reaching an agreement on the current block.

3.2.2. TBFT vs. PBFT

TBFT is a derivation of PBFT, which differs in one fundamental thing. PoH provides a global source of time before consensus is reached and can therefore be used to enforce the exponentially-increasing timeouts introduced in the original PBFT algorithm. No messages are needed as the PoH itself enforces them.

The procedure is as follows. Voting on a new block is restricted to a fixed time period counted in hashes, this unit of time is called a slot. At the moment and with the current network settings, if we convert the number of PoH hashes to time, it is approximately 400ms for one slot. Thus every 400ms, a new potential rollback point occurs, but each new block that is voted on doubles the amount of time the network would have to stall before unrolling the original vote.

Consider that each validator has voted 32 times in the last few ~12 seconds ($32 \cdot 0.4$). The vote 12 seconds ago now has a timeout of 2^{32} slots, which converted to years with a constant time of a slot of 400ms, is roughly 54 years ($2^{32} \cdot 0.4/86400/365$). A transaction with 32 confirmations is also considered finalized.

3.3. Turbine – Block Propagation Protocol

Turbine is a name for a smart block propagation protocol that reduces the time needed for block propagation and the overall message complexity reducing the communication overhead of a node.

Turbine is a multi-layer propagation protocol. First, nodes in the network are divided into small partitions called neighborhoods. Nodes within a particular neighborhood are responsible for sharing received data with other nodes in the same neighborhood and

propagating the data to a small number of nodes in other neighborhoods (Figure 1.3 and 1.4). The data unit shared is called a shred, and one block is constituted of many shreds.

The partitioning of nodes into neighborhoods and how exactly are shreds shared within and out of their neighborhoods are implementation details.

Since we are in an adversarial environment, any node can decide not to rebroadcast the received shreds or broadcast incorrect data.

These two problems are solved with a series of countermeasures:

- Forward Error Code (FEC), specifically the Erasure Code, helps by broadcasting a block with more shreds than initially needed to reconstruct the entire block without errors, even if some shreds are lost along the way. With $N = 6$ data shreds and additional $K = 3$ shreds, we can lose up to 1/3 of the shreds and still be able to reconstruct the entire block fully.
- Propagation is prioritized according to their stake. Validators with the most stake are put closer to the current leader. A stake-weighted selection algorithm is used to create a tree where the risk of faulty or malicious nodes is minimized.

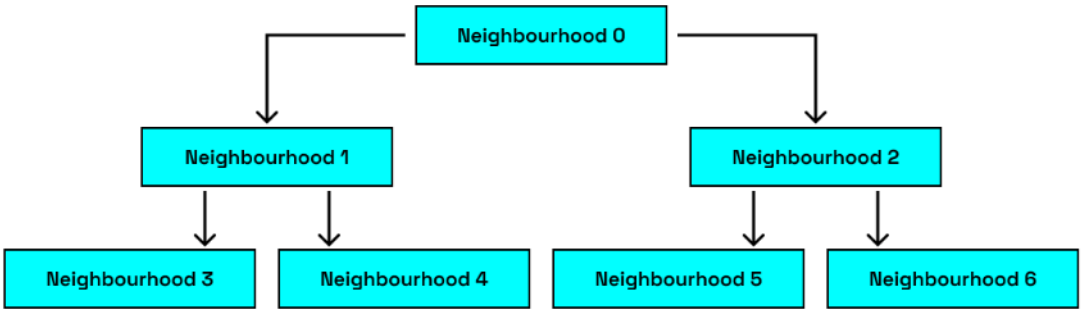

Figure 1.3: Shred propagation diagram [13]

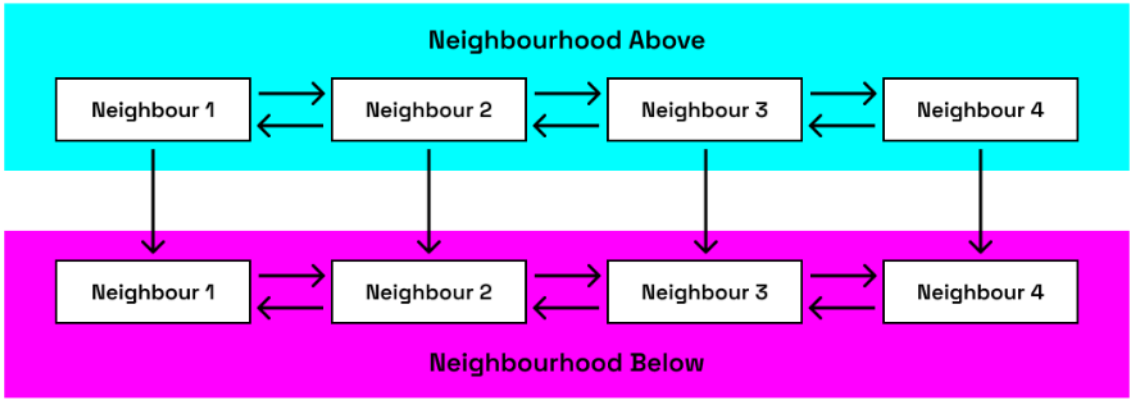

Figure 1.4: Shred propagation between two neighborhoods [13]

3.4. Gulf Stream – Transaction Forwarding Protocol

Gulf Stream is Solana's mempool-less solution for forwarding and storing transactions before processing them.

In traditional blockchains, each node reserves a part of its memory for the memory pool. This memory pool, more commonly referred to as mempool, is used to store transactions currently being broadcasted over the network but have not been processed and added to the blockchain as a part of a new block yet.

This implies a huge communication overhead where every transaction must reach every other node in the network. While not everyone necessarily needs to be aware of all transactions in the mempool, they are the most important for miner and validator nodes (depending on the type of a network), which must include them in new blocks.

If there are more transactions in the mempool than can fit in the block, the backlog of transactions is created. This can generally lead to increased transaction fees for users who need to push their transaction ahead of other transactions, as it is economically viable for the nodes securing the network to prefer transactions with higher fees. This is currently not possible on Solana, but on the other hand, the network is so fast with its ~400ms block rate that the aim is to process all remaining transactions almost instantaneously anyway.

Since Solana aims to process potentially hundreds of thousands of transactions, common gossip protocols used in other blockchains to propagate transactions to all nodes are infeasible.

3.4.1. The Solution

The solution that Solana devised is to avoid having a single shared mempool and instead push transactions to the edge of the network to the expected leader. The leader receives the transaction as quickly as possible and can process it immediately.

However, this solution has a catch. The expected leader must be known ahead. Leaders are known in advance; their rotation is a function of the blockchain data and is known one full epoch before. An epoch is the number of slots for which one leader's schedule is valid. It is set to 432,000 slots, and with a ~400ms block rate, it takes about two days.

3.5. Sealevel – Parallel Smart Contract Runtime

Other blockchains are single-threaded global state machines. The only thing they might do in parallel is signature verification. Solana introduced Sealevel, a parallelized transaction processing engine designed to scale horizontally across GPUs and SSDs.

Sealevel can theoretically process as many transactions as many cores are available to the system. According to the source code, Sealevel is not yet parallelized on the GPU level.

This is a major improvement that makes Solana a multi-threaded global state machine, a thing not seen until Solana. Other blockchains, including the leading Ethereum, can be considered single-threaded global state machines because only one smart contract invocation can be processed at a time.

The reason this is possible with Solana is that each and every Solana transaction describes all the states required to read and write to. Sealevel can then choose non-overlapping instructions to be executed in parallel and not only that. Transactions that only read certain states can be executed in parallel as well.

This is a high-level description of how it works:

- Sort millions of pending transactions.
- Schedule all the non-overlapping transactions in parallel.

3.5.1. SIMD approach with GPUs

There is a great potential for GPU parallelization and leveraging its SIMD capability. For example, in Nvidia CUDA, modern cards have thousands of CUDA cores and tens of Streaming Multiprocessors.

When a CPU invokes a kernel grid, the blocks of threads are distributed among streaming multiprocessors and executed using specific ALU execution units, usually called CUDA cores and other SFUs (special function units).

The executed code is the same for all cores. Imagine a situation where there is a single smart contract invocation but with numerous different inputs. This exact workload can be efficiently executed on GPU architectures, such as Nvidia CUDA.

Since Sealevel is not yet optimized for GPU offloading, GPUs today are only used to accelerate PoH and signature verification and only if it is available to the system and the algorithm decides it is worth the overhead of launching the kernel grid.

3.5.2. BPF – Berkeley Packet Filter

There is one important thing that has not been covered in Sealevel yet. What actually executes the code, and how it is done. The standard way is to use some sort of a Virtual Machine (VM) and compile the code for it from any supported language. This code gets deployed to the blockchain, and when the user sends a transaction invoking this contract, the code gets loaded into the VM and executed.

Ethereum does this using its own Ethereum Virtual Machine (EVM). Some other blockchains make use of Web Assembly (WASM). Solana iterated through all possible solutions and chose an unexpected VM called Berkeley Packet Filter (BPF).

Sealevel hands off transactions to be executed using an industry-proven bytecode called the Berkeley Packet Filter (BPF), designed for high-performance packet filters. It can also be used for non-networking purposes. BPF and the extended BPF (eBPF) are in-kernel VMs available in most UNIX-like operating systems. They are very performant because their primary use was for packet matching, which needs to be as fast as possible. It also has decades of development behind it.

The original version of BPF is now called classic BPF (cBPF), and this one could not be used for anything other than packet matching. The Linux kernel now includes only extended BPF (eBPF), a virtual machine with 64-bit registers. The eBPF is now called just BPF.

It is worth mentioning that new modern firewalls are being built on top of the extended BPF. BPF execution is currently parallelized only on the CPU level. What is used is a modified version of BPF called rBPF, which runs in the user space instead of the kernel. This was important as the kernel version of the BPF would not be able to facilitate certain operations.

3.6. Pipelining – Transaction Processing Optimizations

It is not enough to be able to form a consensus and share a block with the rest of the network quickly. A node must validate and execute all those transactions in received blocks before another block comes.

For this reason, the Solana team developed something called the Transaction Processing Unit (TPU) [14]. The TPU works as a processor and extensively uses pipelining, a common CPU optimization that helps keep the chip more utilized by splitting an instruction execution into stages. It is a general way to keep all the hardware parts busy instead of idle. This concept of pipelining was borrowed, and that is how the TPU was born.

The pipeline stages of TPU are following (Figure 1.6):

- Data fetch in kernel space via network card (I/O)
- Signature verification using GPU (very computation heavy if not offloaded)
- Change of the state using CPU (banking)
- Write to the disk in kernel space and send out via network card (I/O)

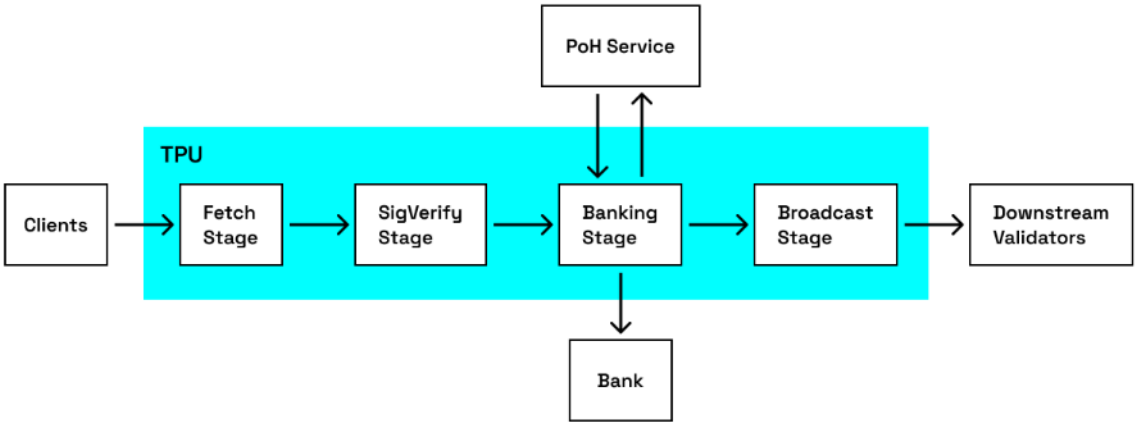

Figure 1.5: Transaction processing unit [14]

In fact, there are two TPUs in the Solana node software. The one called TPU is used for creating a new block, and the second one, TVU, where the V stands for validator or validation, is used for validating. They may slightly differ. However, the concept and functionality are very similar.

3.7. Cloudbreak – Horizontally-scalable Database

With fast computation, the obvious thing that becomes the new bottleneck is the memory. For example, the industry-standard local database for storing blockchain and state, LevelDB, does not support parallel reads and writes. That is fine for Bitcoin or Ethereum, but not for a massively parallel system like Solana.

We could ask the question, why not store everything in RAM? It is too big; even for enterprise machines and large servers, this becomes impossible over time. Therefore Solana had to invent its own database system that supports parallel reads and writes and scales easily with more disks.

This new database system is called Cloudbreak and makes use of memory-mapped files. The data is therefore stored in files that can be accessed independently. A memory-mapped file is a file that is mapped to the process' virtual memory address space and can be accessed directly without further system calls. The speed is still limited by the disk I/O, but we get less overhead, and the kernel can store a part of it in its page cache (also known as file cache).

Reads in Cloudbreak are randomly distributed among available disks, as the data is stored evenly. Writes in Cloudbreak use the Copy-on-Write semantics and are appended to a random disk. Hence we get the speed of sequential writing. This is all possible thanks to a clever system of bookkeeping. Old data entries are also garbage collected for future use.

The design of Cloudbreak makes it ideal for hardware setups, such as RAID 0 with fast NVMe SSDs. The Cloudbreak database was benchmarked by the Solana team. The results show that even with 10 million accounts (unit of data storage on Solana that will be described in the Programming model), a size that does not fit in the RAM (i.e., cannot be cached by page cache in the kernel), Cloudbreak still achieves reads and writes close to 1 million with a single SSD [15].

3.8. Archivers – Distributed Ledger Storage

Given that the Solana blockchain can grow at enormous speed, considering a full capacity of 1 Gbps (with no overhead) for 365 days, it is roughly 4 petabytes of data that each node would need to store to have a complete history. There is a concept of a distributed ledger storage that would store this data in a decentralized fashion for everyone else.

The idea is to offload the data from validators to these specialized network nodes. The data is split into many small pieces and replicated so that the full state can always be reconstructed. These specialized nodes are also contested on the protocol level to ensure they store the data they are supposed to store, and the data loss is prevented.

This concept is yet to be implemented. A potential implementation might be using a new decentralized protocol for permanent storage Arweave or Filecoin.

4. Programming Model

This section explains the programming model of Solana. There are a few fundamental topics that any programmer who wants to use Solana needs to know and study beforehand.

4.1. Communication with the network

Any user who chooses to interact with the network must interact with any of the network's nodes via a JSON-RPC or a WebSocket endpoint. The available methods are all listed publicly in the Solana documentation.

The methods vary from queries, such as specific account information, the network state (an example shown in Listings 1.1 and 1.2) to sending transactions.

```
curl http://localhost:8899 -X POST -H "Content-Type: application/json" -d ' {"jsonrpc":"2.0","id":1,
"method":"getBlockHeight"} '
```

Listing 1.1: Request of the getBlockHeight method

```
{ "jsonrpc": "2.0", "result": 0, "id": 1 }
```

Listing 1.2: Response of the getBlockHeight method

What is really important is the ability to send transactions. Sending a transaction is the only way we can change data on the Solana blockchain. Any write operation is done through the means of transactions.

Users are not required to use the RPCs directly. There are several libraries that provide convenient interfaces for languages such as Javascript, Rust and Python.

4.2. Overview

The following steps can be thought of as an overview of what happens when an app or any user interacts with the Solana network by sending a transaction. Terms, such as

instruction, account, or program, will be explained shortly, followed by a more in-depth explanation.

- An app or a user sends a transaction with one or more instructions to a Solana node that accepts RPC requests.
- The transaction gets validated and forwarded according to the deterministic leader schedule to the next leader.
- The transaction is validated and processed by the leader and included in a new block, which is then streamed to all other validators who also validate and process the transaction to reach the same final state.
- During processing, the instructions in the transactions are passed to programs that the developers have deployed in advance. This is the job of the Sealevel runtime. The relevant accounts are modified by code in those programs. Everything happens isolated in the VM. Instructions are executed sequentially and atomically, which means that either all instructions finish successfully or all changes made by any instruction within the transaction are discarded.

4.3. Transaction Key Elements

Some of the transaction key elements should be explained first:

Signature	Each digital signature is in the ed25519 binary format consuming 64 bytes.
Account	A record in the Solana ledger that either holds data or is an executable program.
Compact Array	An array-like data structure that begins with a specially encoded array length in the first 16 bits, followed by the array items.
Blockhash	A unique hash that identifies a block produced as a part of the Proof-of-History algorithm.
Program id	The address (public key) of an account containing a program.
Instruction	A structure specifying a program id for execution, relevant accounts, and opaque instruction data that the program can interpret.

4.4. Transaction Anatomy

A Solana transaction (Figure 1.6) consists of two major parts in the following order:

- A compact array of signatures.
- A message that contains a compact array of account addresses followed by a recent blockhash and ending with a compact array of instructions.

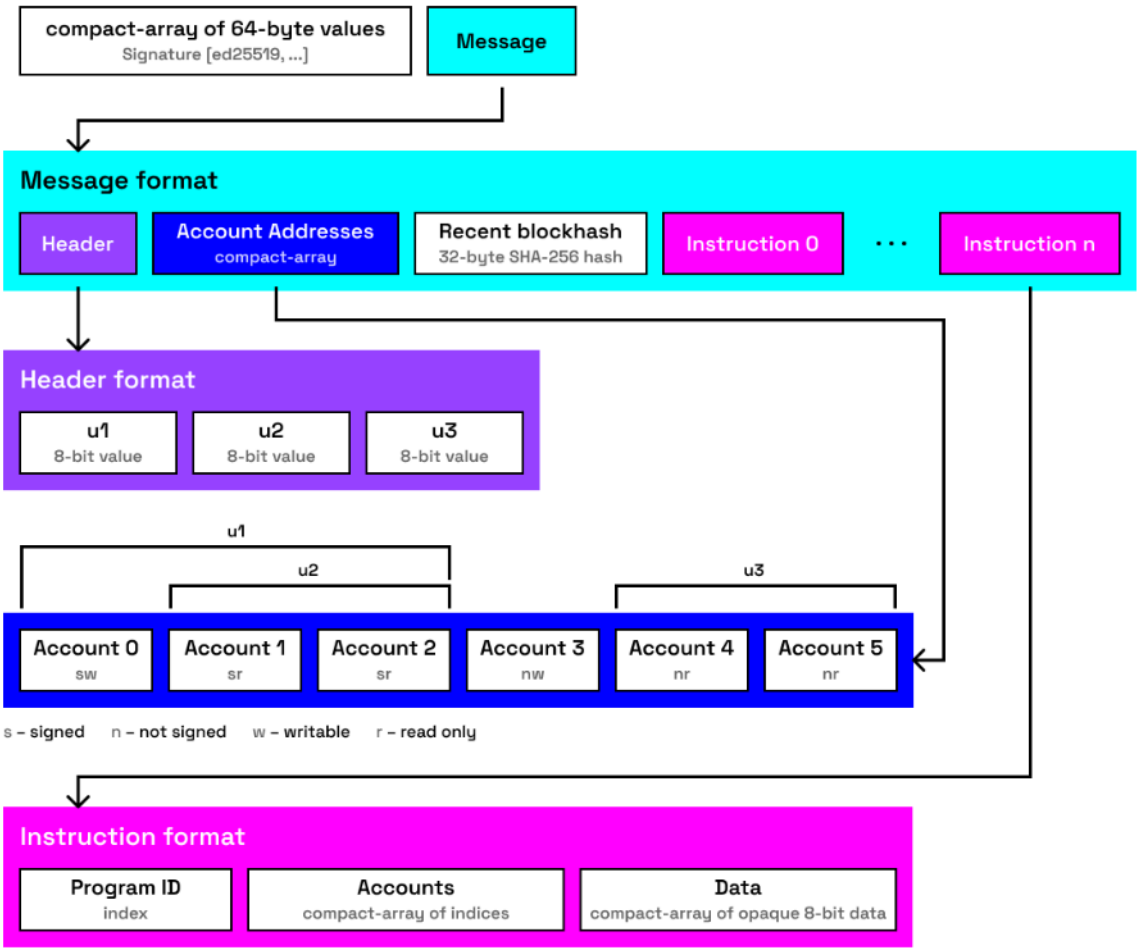

Figure 1.6: Transaction anatomy

4.4.1. Signatures

For signatures in the compact array of signatures, the Solana runtime verifies the following:

- The number of signatures must match the first 8 bits of the message header.

- The signature is verified against the public key at the same index in the message's account addresses array.

4.4.2. Message

The message layout is shown in the following table:

Field	Description
Header	Message metadata
Accounts	Compact array of account addresses
Recent blockhash	Blockhash of recently produced block
Instructions	Compact array of instructions

Table 1.1: Message layout

- Header
  - # of required signatures in the transaction (8 bits).
  - # of read-only accounts requiring signatures (8 bits).
  - # of read-only accounts not-requiring signatures (8 bits).
- Accounts
  - Addresses that require signatures with read-write access.
  - Addresses that require signatures with read-only access.
  - Addresses that do not require signatures with read-write access.
  - Addresses that do not require signatures with read-only access.
- Recent blockhash
  - Transaction lifetime: transaction is deemed invalid if the blockhash is older than 32 blocks.
  - Transaction replay: identical txs get rejected, the blockhash can be changed and the exact same action repeated. It works in a similar way as nonce in Ethereum.
- Instructions with the following instruction anatomy (Figure 1.7):
  - Program id index.
  - Compact-array of account address indices (indices to Accounts).
  - Compact-array of opaque 8-bit data (what operations to perform and any additional data).

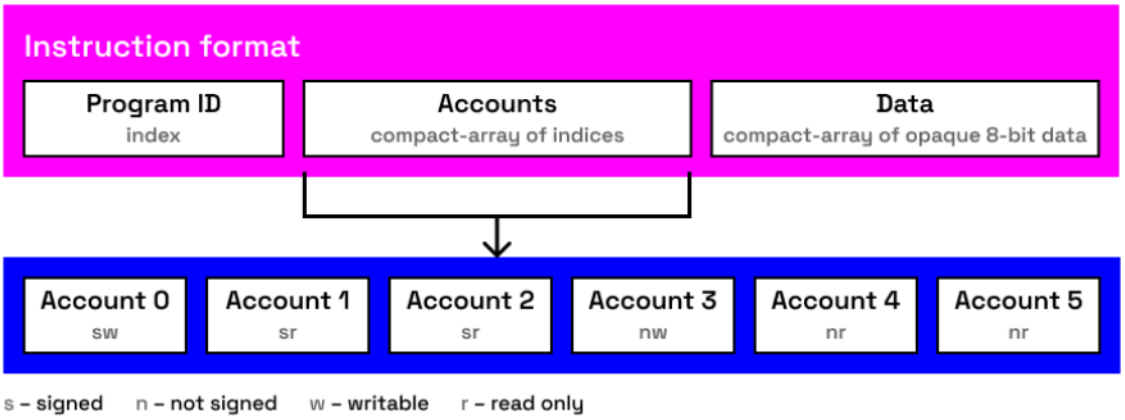

Figure 1.7: Instruction anatomy

4.5. Account Anatomy

Just as in UNIX "everything is a file", in Solana "everything is an account". In other words, an account is a memory buffer, an equivalent of a file in any file system. Its main purpose is to store states between instructions and transactions. An address often referred to as a public key or pubkey, is used to look up an account. Solana's account system can therefore be considered a key-value database system.

The key may be one of the following:

- An ed25519 public key.
- A program-derived account address or in short as PDA (32byte value forced off the ed25519 curve).
- A hash of an ed25519 public key with a 32-character string.

The structure of an account is shown in the following table:

Field	Description
Lamports	Lamports in the account.
Data	Data held in this account.
Owner	The program that owns this account; if executable, the program that loads this account.

Executable	This account's data contains a loaded program (and is now read-only).
Rent Epoch	The epoch at which this account will next owe rent.

Table 1.2: Account layout

- Lamports
  - Balance of the account in lamports.
  - 1 lamport = 10^{-9} SOL.
- Data
  - Vector of bytes.
  - Maximum size of 10 MB (10 KB for PDAs).
- Owner
  - The owner is a program id or a loader in case of an executable account.
  - If the owner matches the program id, the program is granted write access. Otherwise, it is only permitted to read its data and credit the account.
  - All new accounts are owned by the System program that allows transfers of lamports, allocating data, and assigning ownership to a different program id.
  - An account is always owned by a program or a loader.
- Executable
  - Turning a non-executable account into an executable one is a one-way only operation.
  - The account becomes read-only.
  - The owner of such an account is a loader that will load the code from the data field of the account and start executing it if invoked.
- Rent Epoch
  - For keeping accounts alive on Solana incurs a fee called rent.
  - An account is considered rent-exempt if it holds at least two years' worth of rent.
  - Rent Epoch is the epoch number when the runtime checks again whether the account should pay rent or is rent-exempt.

4.5.1. Account Types

There are three basic types of accounts on Solana. Note that this is not any sort of official classification.

- Data accounts storing data or user wallets with an empty data field
  - Program owned PDA (Program Derived Address) accounts
  - System owned accounts..
- Program accounts storing user-deployed executable bytecode.
- Native accounts indicating native programs or special runtime accounts.
  - System – lamports transfers, data allocation and ownership assignment.
  - BPF Loader – uploading and launching executable programs.
  - BPF Upgradeable Loader – uploading and launching of upgradeable executable programs.
  - Stake – program for staking SOL as a part of Proof-of-Stake mechanism.
  - Vote – program for voting as a part of the Tower BFT consensus.
  - Native Loader – the owner of native programs and their loader.
  - and more . . .

4.6. Runtime Policy

Runtime policy or Sealevel runtime account rules are a set of rules enforced by the Sealevel runtime to protect the security of the system and make Solana a safe and predictable environment for its users.

The following list of rules is taken from the official documentation [16]:

- Only the owner of the account may change owner.
  - And only if the account is writable.
  - And only if the account is not executable.
  - And only if the data is zero-initialized or empty.
- An account not assigned to the program cannot have its balance decrease.

- The balance of read-only and executable accounts may not change.
- Only the owner may change account size and data.
    - And if the account is writable.
    - And if the account is not executable.
- Executable switch is a one-way (false→true) operation, and only the account owner may set it.
- No one can make modifications to the rent_epoch associated with this account.

4.6.1. Compute Budget

Each transaction is given a compute budget to prevent abuse of the Solana nodes' resources that could potentially lead to network failures or denial of service. When the program consumes its entire compute budget or exceeds certain bounds, the runtime halts the currently running instructions and returns an error.

4.7. Calling between programs

Cross Program Invocation (CPI) is a facility that allows us to call other programs from within an instruction. The caller is halted until execution returns from the callee. An important term associated with CPIs is a Program Derived Address (PDA).

4.7.1. Program Derived Address (PDA)

Programs can issue instructions containing accounts that were not signed in the original transaction by using Program Derived Addresses (PDA). These accounts are called PDA accounts. Program-derived addresses allow programmatically generated signatures to be used when calling between programs.

PDA is an address deterministically derived from the program id and supplied keywords or more familiar seeds (Figure 1.8). If needed, the resulting address is checked against the Ed25519 curve and bumped off it with so-called bump seeds. Hence, there is no private key.

When a program tries to invoke a CPI with such an address, the runtime takes the supplied keywords and bump seeds, uses the caller's program id, and repeats the process. If the resulting PDA matches, the account is considered to be signed.

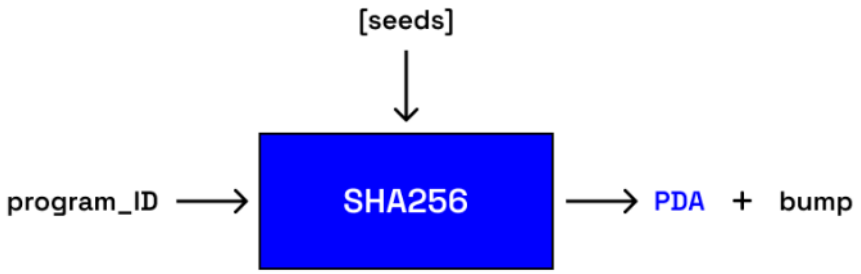

Figure 1.8: PDA generation

5. Solana Program Library

The Solana Program Library (SPL) [17] is a collection of on-chain programs such as SPL-Token that facilitates tasks such as creating and using tokens and a lot more.

5.1. Token Program

A Token program [18] on the Solana blockchain. This program defines a common implementation for Fungible and Non Fungible tokens.

All tokens on Solana, whether they are fungible tokens or NFTs (see Figure 1.9), are created using the SPL Token Program. If you're familiar with Ethereum, you can think of SPL tokens as a token standard such as ERC-20 or ERC-721. One key difference, however, is that Solana does not require you to deploy a new contract for each token you create. Instead, it simply requires you to send instructions to the Token Program and it will create and mint tokens on your behalf.

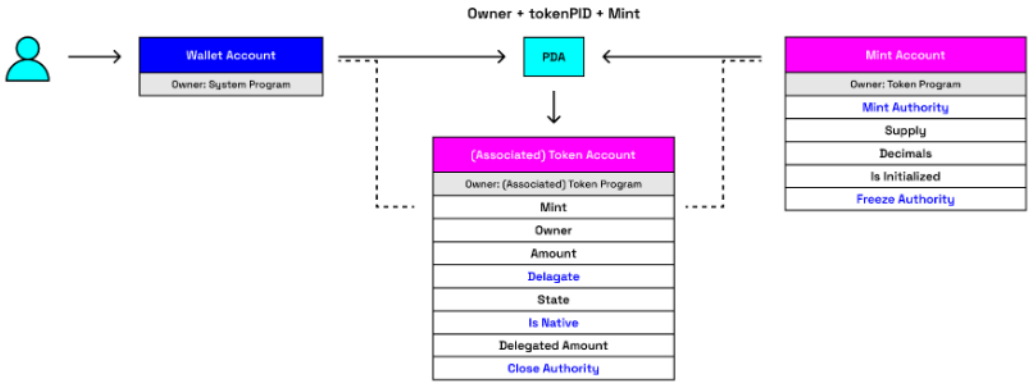

Figure 1.9: SPL Token

5.1.1. Creating a new Token

A new Token can be created [18] by initializing a new Mint with the InitializeMint instruction. The Mint is used to create or "mint" new tokens, and these tokens are stored in Accounts. A Mint is associated with each Account, which means that the total supply of

a particular token type is equal to the balances of all the associated Accounts (see Figure 1.9 for mentioned fields).

Once a Mint is initialized, the mint_authority can create new tokens using the MintTo instruction. As long as a Mint contains a valid mint_authority, the Mint is considered to have a non-fixed supply, and the mint_authority can create new tokens with the MintTo instruction at any time.

5.1.2. Transferring Tokens

Balances can be transferred [18] between Accounts using the Transfer instruction. The owner of the source Account must be present as a signer in the Transfer instruction when the source and destination accounts are different.

The image provided below depicts the process of token transfer. Further information about Associated Token Accounts (ATA) will be covered in subsequent sections.

5.1.3. Burning Tokens

The Burn instruction [18] decreases an Account's token balance without transferring to another Account, effectively removing the token from circulation permanently.

There is no other way to reduce supply on chain. This is similar to transferring to an account with an unknown private key or destroying a private key. But the act of burning by using Burn instructions is more explicit and can be confirmed on chain by any parties.

5.1.4. Freezing Accounts

The Mint may also contain a freeze_authority which can be used [18] to issue FreezeAccount instructions that will render an Account unusable. Token instructions that include a frozen account will fail until the Account is thawed using the ThawAccount instruction. The SetAuthority instruction can be used to change a Mint's freeze_authority.

5.1.5. Wrapping Sol

The Token Program can be used [18] to wrap native SOL. Doing so allows native SOL to be treated like any other Token program token type and can be useful when being called from other programs that interact with the Token Program's interface.

Accounts containing wrapped SOL are associated with a specific Mint called the "Native Mint"

These accounts have a few unique behaviors:

- InitializeAccount sets the balance of the initialized Account to the SOL balance of the Solana account being initialized, resulting in a token balance equal to the SOL balance.
- Transfers to and from not only modify the token balance but also transfer an equal amount of SOL from the source account to the destination account.
- Burning is not supported
- When closing an Account the balance may be non-zero.

5.1.6. Non-Fungible Tokens

An NFT is simply a token type where only a single token has been minted.

A more comprehensive discussion about NFTs is conducted in Appendix B.

5.2. Associated Token Account Program

This program defines [19] the convention and provides the mechanism for mapping the user's wallet address to the associated token accounts they hold.

5.2.1. Motivation

- A user may own arbitrarily many token accounts belonging to the same mint which makes it difficult for other users to know which account they should send tokens

to and introduces friction into many other aspects of token management. This program introduces a way to deterministically derive a token account key from a user's main System account address and a token mint address, allowing the user to create a main token account for each token they own. We call these accounts Associated Token Accounts.

- In addition, it allows a user to send tokens to another user even if the beneficiary does not yet have a token account for that mint. Unlike a system transfer, for a token transfer to succeed the recipient must have a token account with the compatible mint already, and somebody needs to fund that token account. If the recipient must fund it first, it makes things like airdrop campaigns difficult and just generally increases the friction of token transfers. The Associated Token Account program allows the sender to create the associated token account for the receiver, so the token transfer just works.

5.3. Token-2022

The Token-2022 Program [20] extends the functionality provided by the Token Program. This means that the Token-2022 program is backward compatible and includes all the functions of the original Token program, as well as additional functionality often referred to as token extensions. You can think of the extensions as a series of options, features, and capabilities built into the newest iteration of the Solana token program.

5.3.1. Benefits

- **Flexibility:** Token issuers can choose to enable any combination of token extensions.
- **Reduced risk:** Using audited and well-tested extensions reduces attack vectors and helps to protect protocols and funds.
- **Reduced testing costs:** Because the extensions are added by simply specifying the extensions in your code, the chances of defects and human error are greatly reduced, saving on testing time and costs.
- **Reduced development time:** Because the extensions are uniform and reusable, the time required to develop applications using the extensions is significantly reduced.

5.3.2. Extensions

Extensions can be of two types: mint and account extensions. All of these extensions can be used out-of-the-box.

Mint extensions are added on top of the original Solana Token Program and extend the abilities of tokens. Account extensions are added on top of Solana accounts and add account-related features.

Current mint extensions include 14 extensions. Some of the most important are:

- **Confidential transfers:** Allow confidential transfers between participating users without revealing the amount of the transfer.
- **Transfer fees:** Allow transfer fees to be charged on each transfer and sent to a defined account.
- **Mint close authority:** Allows owners to close mint accounts and reclaim the lamports on the mint account.
- **Transfer hook:** Allows calling specific programs with each token transfer.

Current account extensions include:

- **Memo required on transfer:** Requires an attached memo as a message during each token transfer. This could be used for regulatory compliance, reporting, and enhanced audit trails. Immutable owner: Makes it impossible to reassign ownership of an account.
- **Default account state:** Freezes all new token accounts so that users must interact with the project in some way to unfreeze the accounts/tokens.
- **CPI guard:** Restricts how other programs can interact with your token by prohibiting certain actions inside cross-program invocations.
- **Reallocate:** Some extensions can be enabled after account creation. Reallocate allows owners in this situation to reallocate their token account to create room for more extensions.

For a complete explanation and guide about the Token 2022 Program and its extensions see [20].

5.4. Account Compression Program

The Account Compression Program [21] is an innovative on-chain system designed to alleviate the rising concern of storage costs on the Solana blockchain. Its main application revolves around the utilization of SPL ConcurrentMerkleTrees, allowing for the on-chain verification of off-chain data edits. This innovative solution has been crafted in response to the challenges brought about by the increased creation of Non-Fungible Tokens (NFTs) on the Solana blockchain.

5.4.1. Motivation

- Solana's high throughput has fostered a significant increase in the creation of NFTs. The attractive features of NFTs, such as custodial ownership and censorship resistance, have contributed to their popularity. However, this widespread adoption has led to a critical concern: the network storage costs when creating NFTs at scale.
- While minting a single non-fungible token may be relatively inexpensive, the cost of storing the asset's data on-chain can quickly become uneconomical as the quantity increases. This issue presents a barrier to the practical and widespread use of NFTs, especially when they are produced en masse.
- The objective is to make the cost per token as close to zero as possible, ensuring affordability and scalability. The solution lies in storing a compressed hash of the asset data on-chain, while the actual data resides off-chain in a database.
- The Account Compression Program facilitates this by providing a means to verify the off-chain data on-chain and enabling concurrent writes to the data. A central component of this solution is the Concurrent Merkle Tree, a newly introduced data structure that prevents proof collision during concurrent writes.

5.4.2. Application

- The Account Compression Program is already in use in projects like the Metaplex Bubblegum Program. Its implementation has allowed for a reduction in on-chain storage costs, making it more economical to produce and manage NFTs at scale.

Appendix A - Ecosystem

A brief sample of Solana's ecosystem projects follows.

1. Wallets

The Solana ecosystem consists of various user-facing products like wallets and tools that allow anyone to easily create their token or a Non-Fungible Token (NFT) and use the network without much hassle.

There are various web wallets, Android and iOS app wallets for smartphones, browser extensions, and also official CLI tools. They differ in capabilities and out-of-the-box support for various Solana projects or being developer-oriented.

The major ecosystem wallet providers include:

- Phantom (iOS/Android apps and all major browser extensions)
- Solflare (Web wallet, iOS/Android apps, and chrome-only extension)
- Sollet (Developer-oriented web wallet and chrome-only extension)
- Backpack (Web wallet, iOS/Android coming soon)

(At the time of writing, summer 2023)The only supported hardware wallets to safely interact and store keys to access cryptocurrencies and other assets are now Ledger products:

- Ledger Nano S
- Ledger Nano X

2. Popular Projects

Solana's blockchain has become a fertile ground for innovation, attracting numerous high-profile projects due to its speedy, reliable, and scalable solutions for blockchain

applications and technologies. Below, we've curated a list of some of the best-known names in their respective domains, but don't be misled; this is just the tip of the iceberg. There are many more applications and projects thriving on the Solana platform, contributing to a vibrant and continually expanding ecosystem.

2.1. Derivatives

Derivatives are products including options, futures, collateralized loans, and prediction markets, where an arrangement or instrument has a value derived from the underlying asset.

- HXRO

2.2. Decentralized exchanges on Solana

Decentralized exchanges (DEXs) are a pivotal innovation in the world of cryptocurrency, allowing users to trade assets directly with one another without the need for an intermediary or centralized entity. Unlike traditional exchanges, where the exchange controls the user's funds, DEXs operate on smart contracts that handle the trading process. This design enhances security and privacy, as users retain control over their private keys. While offering benefits like reduced risk of hacking and censorship resistance, DEXs may face challenges such as lower liquidity and higher fees compared to centralized counterparts. Nevertheless, they represent a significant step towards a more decentralized and transparent financial ecosystem.

- Orca
- Raydium
- Saber
- Jupiter Aggregator
- Mango Markets
- Drift Protocol

2.3. Solana Liquid Staking

Liquid staking is a financial innovation within the crypto space that allows users to stake their assets while still having access to liquidity. Essentially, it lets participants earn staking rewards without locking up their tokens, by issuing synthetic or derivative tokens in return. This flexibility fosters greater participation and efficiency in proof-of-stake networks.

- Lido
- Marinade Finance

2.4. Decentralized Lending on Solana

Decentralized lending refers to peer-to-peer lending practices conducted on blockchain networks. By using smart contracts, it eliminates traditional financial intermediaries, allowing users to borrow or lend funds directly to one another. This offers more accessible and potentially lower-cost lending options, promoting financial inclusion and greater control over personal finance.

- Solend

2.5. Marketplaces

There are multiple NFT marketplaces to trade NFTs (see Chapter 7 for more about NFTs), which are theoretically supposed to represent a certificate of ownership of something.

The trend of NFTs is closely coupled with an old-new concept of Metaverse, an artificial world where people could meet as virtual avatars and show off their collectible NFTs.

NFTs may find useful applications in the future for representing real-world items, such as concert tickets, but it remains to be seen if they offer any real advantage in practice or if the technology continues to be used largely for speculation.

- MagicEden
- Tensor
- Solanart

2.6. Analytics

Analytics tools on blockchain provide insights into transactions, network health, and user behavior. They enable transparency, traceability, and data-driven decision-making, enhancing efficiency and trust within the decentralized ecosystem.

- DefiLlama
- Dune Analytics
- Nansen
- VybeNetwork
- Step Finance

2.7. Tooling

Tools that enhance the safety of smart contracts facilitate thorough testing and validation, detecting vulnerabilities and errors. They improve the overall workflow of developing decentralized applications, ensuring robustness and security. Additionally, they enhance the usability of existing smart contracts, making them more accessible and efficient for users.

- Solana Anchor Framework
- Trident Testing and Fuzzing Framework
- Solang
- Metaplex

2.8. Gaming

First games built on Solana are starting to appear, using the blockchain as a back end (see Appendix C for more about gaming) – this type of development paradigm is referred to as Web3. In general, any application using blockchain as a layer of data and business logic can be considered Web3.

When a user visits such a website, they can usually connect their wallet with the website and interact with it by sending transactions that could, e.g., post a message on a Web3 social media platform or place an order on a decentralized exchange.

- Star Atlas
- xNFTs with BackPack

2.9. Real World Projects

The remarkable benefits of the Solana Blockchain make it a magnet for projects affiliated with real-world applications.

- Helium
- Render Network
- HiveMapper

2.10. Bridges

Crosschain bridges enable the transfer of assets and information between different blockchain networks. They enhance interoperability, allowing seamless transactions across various platforms, thus expanding possibilities within the decentralized ecosystem.

- Wormhole
- DeBridge Finance

3. Solana Mobile Phone

As you may have observed, Solana has recently introduced its own mobile phone. In this section, We will outline essential details regarding the Solana Mobile Stack [22].

3.1. What's in the Solana Mobile Stack?

The Solana Mobile Stack (SMS) is a collection of key technologies for building mobile applications that can interact with the Solana blockchain.

3.2. Mobile Wallet Adapter

Mobile Wallet Adapter (MWA) is a protocol specification for connecting mobile dApps to mobile Wallet Apps, enabling communication for Solana transaction and message signing.

dApps that implement MWA are able to connect to any compatible MWA Wallet App and request authorization, signing, and sending for transactions/messages.

Why this is important: Developers no longer need to build in support for each individual wallet, and instead can just integrate once and use a single unified API to be compatible with every compliant Solana wallet!

3.3. Using the SDK

Solana Mobile maintains an official Mobile Wallet Adapter SDK that implements the protocol, originally written as a Android Kotlin/Java library.

The SDK is also ported other frameworks and is available for:

- React Native
- Flutter
- Unity
- Unreal Engine

3.4. Seed Vault

The Seed Vault is a system service providing secure key custody to Wallet apps. By integrating with secure execution environments available on mobile devices (such as secure operating modes of the processor and/or secure auxiliary coprocessors), Seed Vault helps to keep your secrets safe, by moving them to the highest privileged environment available on the device. Your keys, seeds, and secrets never leave the secure

execution environment, while UI components built into Android handle interaction with the user to provide a secure transaction signing experience to users.

3.5. Solana dApp Store

The Solana dApp Store is an alternate app distribution system, well suited to distributing apps developed by the Solana ecosystem.

It will provide a distribution channel for apps that want to establish direct relationships with their customers, without other app stores' rules restricting the relationship or seeking a large revenue share. The goal of the Solana dApp Store is to empower the Solana community to eventually play a key role in managing the contents of this app store.

3.6. Solana Pay for Android

The Solana Pay protocol was developed independently of the Solana Mobile Stack, but combining payments with a mobile device is a natural fit for Solana Pay.

The Solana blockchain confirms transactions in less than a second and costs on average $0.0005, providing users a seamless experience with no intermediaries.

Businesses and developers can use Solana Pay to accept payments in SOL or any SPL token without intermediaries. It offers frictionless and portable integration options like payment links, pay now buttons or QR codes on your app, dApp, website, blog, and so much more.

3.7. eCommerce Platform Integrations

Solana Labs has started a reference implementation for Shopify which you can see here to get a sense of how this might work.

Here are some of the top eCommerce platforms [23] that they're looking to integrate to:

- WooCommerce
- Magento
- BigCommerce
- Wix
- Squarespace

Appendix B - NFTs

The phenomenal rise of non-fungible tokens and innovative developments in the NFT landscape have encouraged many people to mint their own NFTs. Solana blockchain has emerged as a popular solution for minting NFTs with its Metaplex platform. The Metaplex is a Solana-based protocol that helps in creating NFTs and auctions. The Candy Machine Metaplex protocol serves as the primary foundation for minting and distribution of NFT collections.

1. What is Metaplex?

Metaplex [24] is a decentralized protocol built on the Solana blockchain designed to simplify the creation, sale, and use of NFTs. By offering a suite of development tools, smart contracts, and open standards, Metaplex enables NFT communities to issue, manage, and own digital assets on-chain.

Metaplex continuously develops and maintains several programs and standards to enhance user experience and meet the evolving needs of its community.

Currently, major projects are

- Token Metadata - the NFT standard for Solana
- Candy Machine v3™ - a Profile Picture (PFP) focused tool that works like the gumball-style candy machines of old. Candy Machine V3 supports the minting of programmable NFTs.
- Auction House - a decentralized sales protocol for NFT marketplaces
- Creator Studio - No-code tools to create, sell, and manage NFTs on Solana.
- Fixed-Price Sale - A program that enables creators to build/distribute membership NFTs at a fixed-price that grants holders access to exclusive content and events. It supports the restriction of sales to specific collection holders.

- Compression - A program for creating and interacting with compressed Metaplex NFTs. Compressed NFTs are secured on-chain using Merkle trees.

2. Token Metadata Program

The Token Metadata program [25] is one of the most important programs when dealing with NFTs on the Solana blockchain. Its main goal is to attach (Figure 1.11) additional data to Fungible or Non-Fungible Tokens on Solana.

It achieves this using Program Derived Addresses (PDAs) that are derived from the address of Mint Accounts. You should be familiar with Solana's Token program from SPL (for reminder, Figure 1.10), Mint Accounts are responsible for storing the global information of a Token and Token Accounts store the relationship between a wallet and a Mint Account.

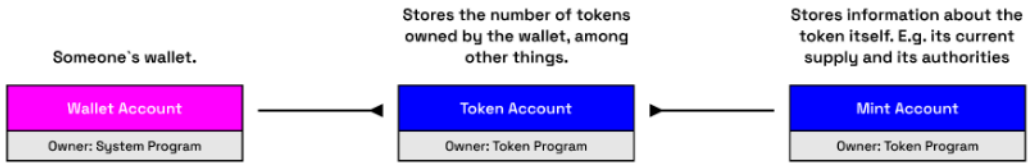

Figure 1.10: Simplified SPL Token

Whilst Mint Accounts contain a few data attributes such as its current supply, it doesn't offer the ability to inject standardized data that can be understood by apps and marketplaces. This is why the Token Metadata program offers a Metadata Account that attaches itself to a Mint Account via a PDA.

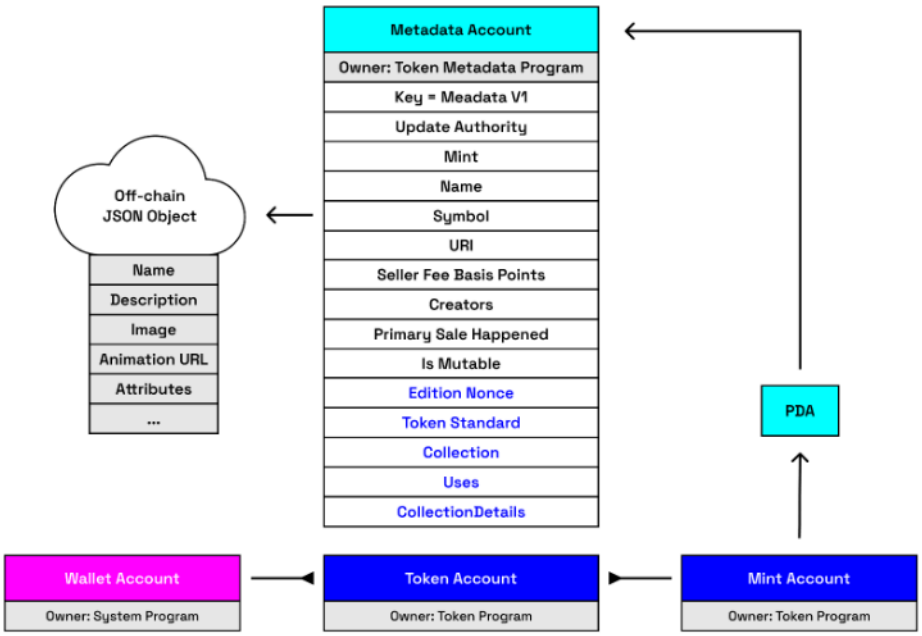

Figure 1.11: Metaplex Metadata Account

3. What has this got to do with NFTs?

Well, NFTs are special tokens that are Non-Fungible. More precisely,

NFTs on Solana are Mint Accounts with the following characteristics (Figure 1.12):

- It has a supply of 1, meaning only one token is in circulation.
- It has zero decimals, meaning there cannot be such a thing as 0.5 tokens.
- It has no mint authority, meaning no one can ever mint additional tokens.

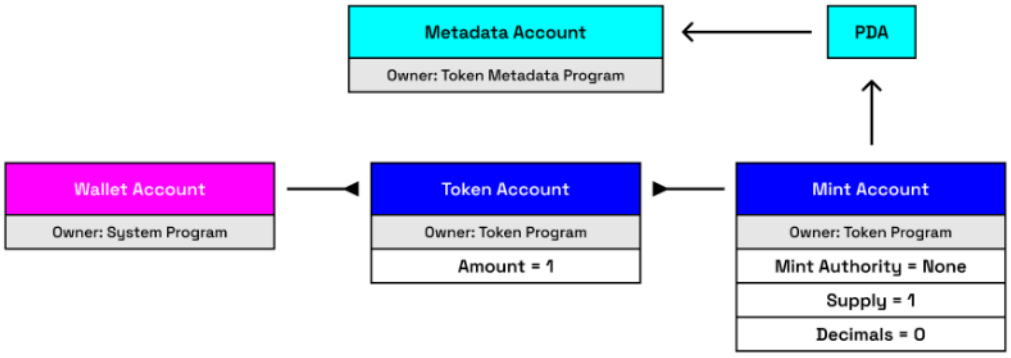

Figure 1.12: Mint Account with NFT-like fields

Metaplex suite offers essential tools for NFT creation on Solana, including the NFT Metadata program itself, minting tools and marketplex, and auction toolkits.

Outside of the essential tools, the toolkit also contains many other rather experimental tools like Fireball, Fuse, and Gumball that enable NFT creators to do many different things with their NFTs on Solana.

3.1. Master Edition

Additionally, the Token Metadata program offers another account specifically for NFTs called the Master Edition Account (Figure 1.13). This account is also a PDA derived from the Mint Account.

Before creating this account, the Token Metadata program will ensure the special characteristics of Non-Fungible Tokens listed above are met. Thus, the existence of the Master Edition account acts as proof of Non-Fungibility for that Mint Account. As shown in Figure 1.13, the Master Edition Account incorporates several fields that haven't been addressed yet. While these fields hold significance, they aren't pivotal for understanding the core principle. In essence, Metaplex offers the capability to create copy of NFTs, wherein the Master Edition, and likewise the Edition Account, functions as evidence of either replication or originality.

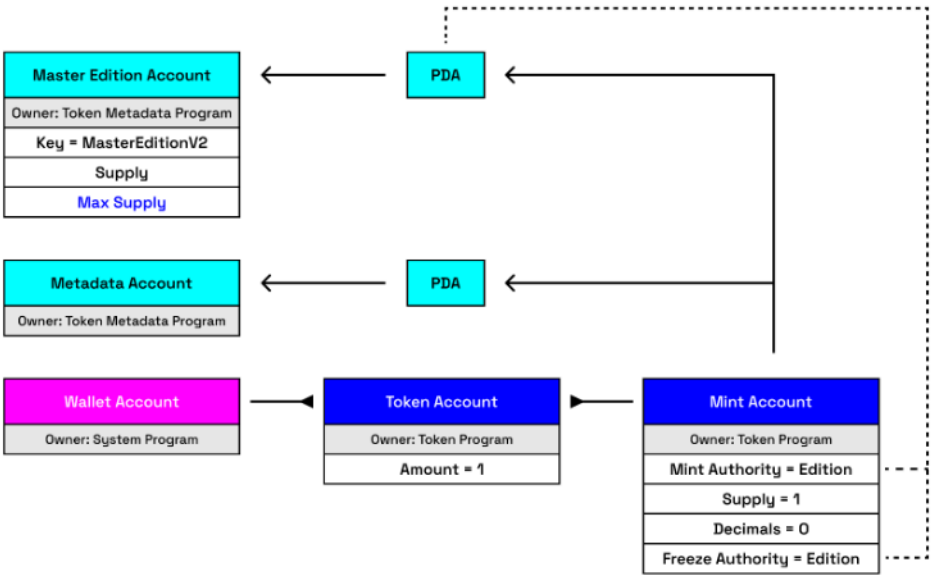

Figure 1.13: Metaplex Master Edition

4. Is this all ? “No, Token Standard”

As token usage has evolved on Solana, it has become clear that there are more types of tokens than simply ”fungible” and ”non-fungible” tokens [26].

An example is something the community is calling a ”semi-fungible token”, an SPL token with a supply greater than 1 but which has typical NFT attributes such as an image and an attributes array in the JSON metadata.

The consensus seems to be that these should be stored in wallets in the same view as standard NFTs, or in their own view but separate from ”standard” fungible SPL tokens. These tokens are becoming popular in gaming contexts to support fungible items such as a kind of sword or a piece of wood, etc. but which are in a different league from typical fungible SPL tokens such as USDC.

The Token Standard field (in Metadata Account) can have the following values:

- NonFungible: A non-fungible token with a Master Edition.
    - Examples of these are Solana Monkey Business, Stylish Studs and Thugbirdz.

- FungibleAsset: A token with metadata that can also have attributes, sometimes called Semi-Fungible.
    - An example of this kind of token is something the community has been calling "semi-fungible tokens" often used to represent a fungible but attribute-heavy in-game item such as a sword or a piece of wood.
- Fungible: A token with simple metadata.
    - USDC, GBTC and RAY.
- NonFungibleEdition: A non-fungible token with an Edition account (printed from a Master edition).
    - very similar to NonFungible (more here).
- ProgrammableNonFungible: A special NonFungible token that is frozen at all times to enforce custom authorization rules.
    - This standard is similar to the Non-Fungible standard above, except that the underlying token account is kept frozen at all times to ensure nobody can transfer, lock or burn Programmable NFTs without going through the Token Metadata program. This enables creators to define custom authorization rules for their NFTs such as enforcing secondary sales royalties.

4.1. Programmable Non-Fungible-Tokens

Because the Token Metadata program builds on top of the Solana Token program, anyone can transfer tokens (fungible or not) without going through the Token Metadata program. In other words, Metadata Account contains field Seller Fee Basis Points, which holds information about secondary sales royalties. The problem is, if we want to transfer tokens (fungible or not), we don't use Metaplex (as its only purpose is to store additional data), however we use SPL for Token Transfer. Whilst there is Seller Fee Basis Points attribute on the Metadata account, it is purely indicative and anyone could create a marketplace that does not honor royalties — which is exactly what happened [25].

Programmable NFTs were introduced to solve this problem. They are a new opt-in Token Standard that keeps the underlying token accounts frozen at all times. That way, nobody

can transfer, lock or burn Programmable NFTs without going through the Token Metadata program.

It is then up to the creators to define custom operation-specific authorization rules (Figure 1.14) that will be enforced by the Token Metadata program. These are defined in a special RuleSet account which is attached to the Metadata account. An example of such a RuleSet could be an allowlist of program addresses that honor royalties. RuleSets are part of a new Metaplex program called Token Auth Rules.

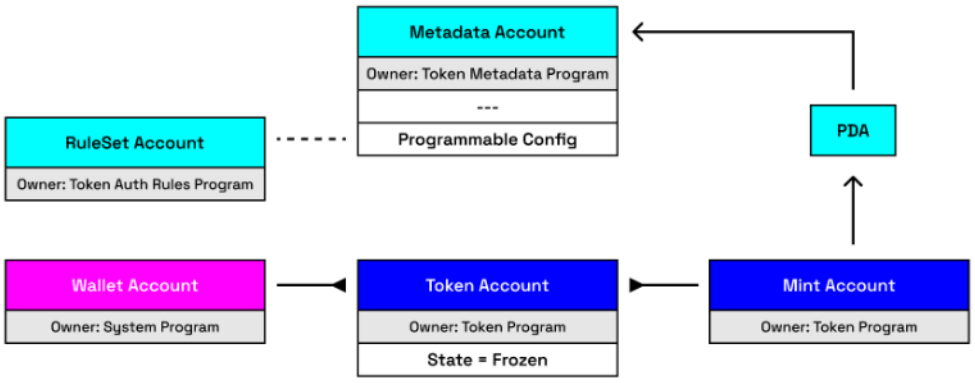

Figure 1.14: Simplified Metadata Account wih RuleSet Account

5. Candy Machine

The Metaplex Protocol Candy Machine [27] is the leading minting and distribution program for fair NFT collection launches on Solana. Much like its name suggests, you can think of a Candy Machine as a temporary structure which is first loaded by creators and then unloaded by buyers. It allows creators to bring their digital assets on-chain in a secure and customisable way.

The name refers to the vending machines that dispense candy for coins via a mechanical crank. In this case the candy are NFTs and the payment is SOL or a SPL token.

By September 2022, 78% of all NFTs in Solana were minted through Metaplex's Candy Machine. This includes most of the well known NFT projects in the Solana ecosystem.

Here are some of the features it offers.

- Accept payments in SOL, NFTs or any Solana token.
- Restrict your launch via start/end dates, mint limits, third party signers, etc.
- Protect your launch against bots via configurable bot taxes and gatekeepers like Captchas.
- Restrict minting to specific NFT/Token holders or to a curated list of wallets.
- Create multiple minting groups with different sets of rules.
- Reveal your NFTs after the launch whilst allowing your users to verify that information.
- And so much more!

Figure 1.15: Illustrative example of Candy Machine

For illustrative purposes, we've included the Figure above. However, we won't delve deep into the intricate logic behind the Candy Machine Program, as such a detailed exploration would be too exhaustive for this guide. Our primary aim is to familiarize you with Solana and its burgeoning ecosystem. Those seeking a deeper understanding can delve further into this documentation.

6. Compression

Compressed NFTs use State Compression and merkle trees to drastically reduce the storage cost for NFTs (Figure 1.16). Instead of storing an NFT's metadata in a typical Solana account, compressed NFTs store the metadata within the ledger. This allows compressed NFTs to still inherit the security and speed of the Solana blockchain, while at the same time reducing the overall storage costs.

Even though the on-chain data storage mechanism is different than their uncompressed counterparts, compressed NFTs still follow the exact same Metadata schema/structure. Allowing you to define your Collection and NFT in an identical way.

Number of NFTs	Solana Today	Solana with Compression	Ethereum	Polygon
1,000	12 ◎ $253.68	2.57 ◎ $54.33	17.69 Ξ $33,774.35	28.94 $33
10,000	120 ◎ $2,536.80	3.49 ◎ $73.78	176.09 Ξ $336,231.23	288.14 $328.48
100,000	1200 ◎ $25,368	4.22 ◎ $89.21	1760.09 Ξ $3,360,800.03	2880.14 $3,283.36
1,000,000	12000 ◎ $253,680	5.35 ◎ $113.10	17600.09 Ξ $33,606,488.03	28800.14 $32,832.16
10,000,000	120000 ◎ $2,536,800	10.76 ◎ $227.47	176000.09 Ξ $336,063,368.03	288000.14 $328,320.16
100,000,000	1200000 ◎ $25,368,000	56.45 ◎ $1,193.35	1760000.09 Ξ $3,360,632,168.03	2880000.14 $3,283,200.16
1,000,000,000	12000000 ◎ $253,680,000	507.13 ◎ $10,720.73	17600000.09 Ξ $33,606,320,168.03	28800000.14 $32,832,000.16

Figure 1.16: Mint prices across blockchains

This chart is based on a snapshot taken on April 5, 2023 and based on a price of SOL at $21.14, MATIC at $1.14, and ETH at $1,909.45 [29].

State compression is already being used by teams across the Solana ecosystem to power large, user-friendly experiences

- Dialect

    - a blockchain-based messaging service, uses state compression for compressed NFTs to cover the minting cost of NFT stickers to thousands of users.

- Crossmint
    - an NFT and API tooling company, is using state compression to create integrations that power deeper customer loyalty for companies around the world.

And that's just the beginning — projects like user-owned wireless network Helium, NFT distributor DRiP, and on-chain publisher Wordcel, are using state compression to bring their scalable, user-first experiences to Solana.

Although state compression can be used to store any sort of data on-chain, the first use of this innovative technology is compressed NFTs. Compressed NFTs are just like regular NFTs, only drastically cheaper — minting 100 million compressed NFTs costs about ◎50 to store on-chain, compared to ◎1.2mm for their uncompressed counterpart. In fact, because every incremental compressed NFT is solely a modification of an existing tree, the cost of an NFT on Solana is now as little as the cost of a single transaction (◎0.000005)!

7. Storing Metadata Off-chain

One important attribute of the Metadata Account is the URI attribute that points to a JSON file off-chain. This is used to safely provide additional data whilst not being constrained by the fees involved in storing on-chain data. That JSON file follows a certain standard that anyone can use to find useful information on tokens.

Note that, this JSON file can be stored using a permanent storage solution such as Arweave to ensure it cannot be updated. Additionally, one can use the Is Mutable attribute of the Metadata Account to make it immutable and, therefore, forbid the URI attribute — and other attributes such as Name and Creators — to ever be changed. Using this combination, we can guarantee the immutability of the off-chain JSON file.

7.1. Arweave

Arweave [28] is a decentralized, trust-minimized, censorship-resistant data storage network designed to retain data permanently, making it a great fit for NFTs. To cover the cost of storing your media forever, storage and mining fees are paid at the time of upload and distributed to storage providers participating in the network.

Arweave storage fees:

- Storage fees are based on the total size of the files you upload to the network during NFT creation. Each NFT consists of three files
    - The asset itself (image, video, audio, etc)
    - The accompanying metadata file (attributes etc.)
    - A generated manifest which creates a logical grouping or relationship between your files
- The cumulative size of these files (in bytes) is submitted to the Arweave storage cost estimation service which returns the real time estimated fee for storage, priced in winstons. We then convert the winstons to SOL for payment.

7.2. Other possibilities

- AWS S3
- IPFS
- NFT.Storage
- Shadow Drive

For further details, refer to the documentation.

8. Executable NFT

Executable NFTs [30], available in the open beta of new open source wallet Backpack, represent an entirely new way to build applications that are safer, better to use, and in line with the decentralized values blockchain is built on. By combining executable websites and applications that can run locally on a user's computer with a crypto wallet, xNFTs in Backpack show a promising new type of hybrid web3 application – not quite an application, not quite a website.

xNFTs take a radically practical approach to solving two of web3's main problems today, decentralization and distribution, with profound implications. While smart contracts run on globally distributed and decentralized networks like Solana, nearly everyone who interacts with smart contracts does so through a website. These centralized, web2 interfaces, and the services that run them, often require the project to register a company, open a bank account, obtain a business debit or credit card, register a domain, and a whole host of other actions just to provide users with a graphical user interface. Each step in that process takes time, money, and adds a potential source of risk for the project's creators.

xNFT Collections unlock whole new avenues for program distribution. A game developer can mint their entire game as limited-edition xNFTs. A DeFi protocol can distribute early access to a new front-end based on wallet addresses. A two-factor code can be generated as an NFT, and automatically cycle through. An artist can create beautiful immersive 3d experiences that can't be copied, and run directly in the wallet. The possibilities are almost limitless.

Because they are, at their core, NFTs, xNFTs can bring a new level of interactivity to traditional collections. The first xNFT Collection, Mad Lads, showcases the power of an executable profile picture collection — the pictures themselves act as a chatroom for users which is rendered entirely within the NFT.

xNFTs are not some new smart contracting language, a scaling solution, or a core technology that will take years to deploy. They exist today. This is just one of the many ecosystem-wide innovations that have supercharged the Solana protocol and greater community.

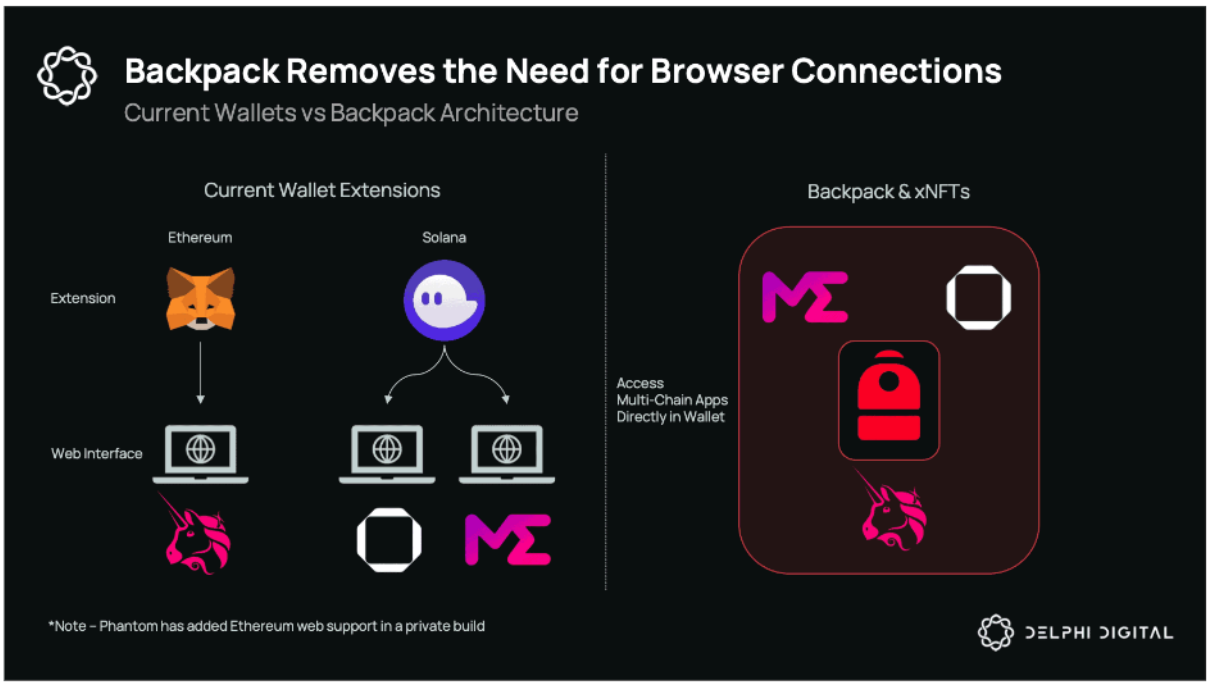

Figure 1.17: Backpack vs. current wallet solutions

Appendix C - Gaming

The gaming space in the Solana ecosystem is expanding rapidly [31]. Integrating with Solana can provide numerous benefits for games, such as enabling players to own and trade their assets via NFTs in games, building a real in-game economy, creating composable game programs, and allowing players to compete for valuable assets.

Solana is purpose-built for games, with its 400ms block time and lightning-fast confirmations making it a real-time database that's free for all. It's perfect for genres like strategy games, city builders, turn-based games, and more.

However, not everything needs to be put on the blockchain. Smaller integrations using NFTs that represent game items, for example, can be easily done. Transaction fees are extremely cheap, and there are many tools and SDKs available to start building today. You can build your game in Javascript and Canvas, Flutter, or use one of the Solana Game SDKs for the two biggest game engines - UnitySDK, UnrealSDK.

There are several ways to integrate Solana into your game:

- Give players digital collectibles for in-game items or use them as characters.
- Use tokens for in-app purchases or micro-payments in the game.
- Use the player's wallet to authenticate them in the game.
- Run tournaments and pay out crypto rewards to your players
- Develop the game entirely on-chain to reward your players in every step they take.

1. Solana Gaming SDK-s

1.1. Unity SDK

The Unity game engine is known for its beginner friendly approach and cross platform support including WebGL, ios and android. Build once export everywhere. The Solana

Unity SDK comes with NFT support, transactions, RPC functions, Phantom Deep lLinks, WebGL connector, WebSocket connection support, mobile wallet-adapter and anchor client code generation.

1.2. Unreal SDK

Unreal engine is known for its great visuals and node based scripting framework. The Solana sdk was originally built by the team of Star Atlas.

1.3. Flutter

Flutter is an open source framework by Google for building beautiful, natively compiled, multi-platform applications from a single codebase.

1.4. Next.js/React + Anchor

One of the easiest ways to build on Solana is using the Web3js Javascript framework in combination with the Solana Anchor frameworks. For more complex games We would recommend using a GameEngine like Unity or Unreal though.

1.5. Python

Python is an easy to learn programming language which is often used in AI programming. There is a framework called Seahorse which lets you build smart contracts in Python.

1.6. Native C#

The original port of Web3js to C#. It comes with a bunch of functionality like transactions, RPC functions and anchor client code generation.

2. NFTs in Games

Non-fungible tokens (NFTs) are rapidly gaining popularity as a means of integrating Solana into games. These unique digital assets are stored on the Solana blockchain and come with a JSON metadata attached to them. The metadata allows developers to store important attributes and information about the NFT, such as its rarity or specific in-game capabilities. NFTs can be used in games to represent anything from weapons and armor to

digital real estate and collectibles, providing a new level of ownership and scarcity for players.

2.1. Token gating with NFTs

Using NFTs, you can gate access to a particular part of a game based on owning the NFT. This can form a more tight-knit community within your game.

2.2. Bonus Effects with NFTs

In addition to providing new revenue streams, NFTs can also be used to provide in-game benefits and bonuses to players. For instance, a player who owns a "coin doubler" NFT may receive double the amount of coins for as long as they hold the NFT in their wallet. Additionally, NFTs can be used as consumables, allowing players to use them to gain temporary effects such as potions or spells. Once consumed, the NFT is burned, and the effect is applied to the player's character.

2.3. Using NFT Metadata for Player Stats

NFTs also have Metadata, which can be used for all kinds of traits for game objects. For example an NFT could represent a game character and his traits Strength/Intelligence/Agility could directly influence how strong the character is in the game.

2.4. Fusing NFTs Together

The Metaplex Fusion Trifle program allows you to have NFTs own other NFTs. For example you could create a plant plot NFT and then combine it with a water NFT and a seed NFT to create a tomato NFT.

2.5. Use 3D Nfts in a game

Every NFT metadata can also have an animation url. This url can contain a video, gif or a 3d file. These 3d files usually use the format .glb or .gltf and can dynamically be loaded into a game.

2.6. Customize NFTs with items and traits

With the Raindrops Boots program you can have an adventure character which owns a sword and a helmet. When the Character NFT would be sold on a market place the other NFTs it owns would be sold as well.

3. Porting a program to Unity

When using Anchor you will be able to generate an IDL file which is a JSON representation of your program. With this IDL you can then generate different clients. For example JS or C# to Unity.

4. Distribution

Distribution of your game depends highly on the platform you are using. With Solana, there are game SDKs you can build for IOS, Android, Web and Native Windows or Mac. Using the Unity SDK you could even connect Nintendo Switch or XBox to Solana theoretically. Many game companies are pivoting to a mobile first approach because there are so many people with mobile phones in the world. Mobile comes with its own complications though, so you should pick what fits best to your game.

Solana has a distinct edge over other blockchain platforms due to its offering of a crypto-native mobile phone, named Saga, that comes equipped with an innovative dApps store. This store enables the distribution of crypto games without the limitations imposed by conventional app stores such as Google or Apple.

Publishing Platforms:

- Fractal - A game publishing platform that supports Solana and Ethereum. They also have their own wallet and account handling and there is an SDK for high scores and tournaments.
- Elixir - Platform for web3 games that also offers a PC launcher
- Self-hosting - Just host your game yourself.

- Solana Mobile dApp Store - The Solana alternative to Google Play and the Apple App Store. A crypto first variant of a dApp store, which is open source free for everyone to use.
- Apple App Store - The Apple app store has a high reach and is trusted by its customers. The entrance barrier for crypto games is high though. The rules are very strict for everything that tries to circumvent the fees that Apple takes for in app purchases. As soon as an NFT provides benefits for the player for example Apple requires you for example to have them purchased via their in app purchase system.
- Google Play Store - Google is much more crypto friendly and games.
- xNFT Backpack - Backpack is a Solana wallet which allows you to release apps as xNFTs. They appear in the users wallet as soon as they purchase them as applications. The Unity SDK has a xNFT export and any other web app can be published as xNFT as well.

References

Kozák, Lukáš. *Developer tooling for Solana* [online]. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022. Available from: https://dspace.cvut.cz/bitstream/handle/10467/101064/F8-DP-2022-Kozak-Lukas-thesis.pdf.

[1] BUTERIN, Vitalik. *Sharding FAQ* [online]. 2017 [visited on 2022-04-07]. Available from: https://vitalik.ca/general/2017/12/31/sharding_faq.html.

[2] NAKAMOTO, Satoshi. *Bitcoin: A Peer-to-Peer Electronic Cash System* [online]. 2008 [visited on 2022-04-07]. Available from: https://bitcoin.org/bitcoin.pdf.

[3] BUTERIN, Vitalik. *Ethereum Whitepaper* [online]. 2014 [visited on 2022- 04-12]. Available from: https://ethereum.org/en/whitepaper/.

[4] KOZÁK, Lukáš. *Security Analysis of Hardware Crypto Wallets*. 2020. Bachelor's Thesis. Czech Technical University in Prague, Faculty of Information Technology.

[5] BECKER, Georg. *Merkle signature schemes, merkle trees and their cryptanalysis*. Ruhr-University Bochum, Tech. Rep. 2008, vol. 12, p. 19.

[6] CASTRO, Miguel; LISKOV, Barbara, et al. *Practical byzantine fault tolerance*. In: OsDI. 1999, vol. 99, pp. 173–186. No. 1999.

[7] ROCKET, Team; YIN, Maofan; SEKNIQI, Kevin; RENESSE, Robbert van; SIRER, Emin Gün. *Scalable and Probabilistic Leaderless BFT Consensus through Metastability*. arXiv, 2019. Available from doi: 10.48550/ ARXIV.1906.08936.

[8] ANTONOPOULOS, Andreas M. *Mastering Bitcoin: Programming the Open Blockchain*. Sebastopol, CA: O'Reilly Media, 2014. isbn 978-1491954386.

[9] ANTONOPOULOS, Andreas M.; WOOD, Gavin. *Mastering Ethereum: Building Smart Contracts and DApps*. Sebastopol, CA: O'Reilly Media, 2018. isbn 978-1491971949.

[10] YAKOVENKO, Anatoly. *Solana: A new architecture for a high performance blockchain* [online]. 2017 [visited on 2022-04-07]. Available from: https://solana.com/solana-whitepaper.pdf.

[11] SOLANA. *Documentation: History* [online]. 2020 [visited on 2022-04-08]. Available from: https://docs.solana.com/history.

[12] BACH, Leo Maxim; MIHALJEVIC, Branko; ZAGAR, Mario. *Comparative analysis of blockchain consensus algorithms*. In: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). 2018, pp. 1545–1550.

[13] SOLANA. *Documentation: Turbine Block Propagation* [online]. 2022 [visited on 2022-04-09]. Available from: https://docs.solana.com/cluster/turbine-block-propagation.

[14] SOLANA. *Documentation: Validator – TPU* [online]. 2022 [visited on 2022-04-09]. Available from: https://docs.solana.com/validator/tpu.

[15] YAKOVENKO, Anatoly. *Cloudbreak: Solana's Horizontally Scaled State Architecture* [online]. 2019 [visited on 2022-04-09]. Available from: https://solana.com/news/cloudbreak---solana-s-horizontally-scaledstate-architecture.

[16] SOLANA. *Documentation: Runtime* [online]. 2022 [visited on 2023-20-08]. Available from: https://docs.solana.com/developing/programming-model/runtime.

[17] SOLANA. *Program Library: Introduction* [online]. 2023 [visited on 2023-20-08]. Available from: https://spl.solana.com/.

[18] SOLANA. *Program Library: Token program* [online]. 2023 [visited on 2023-20-08]. Available from: https://spl.solana.com/token.

[19] SOLANA. *Program Library: Associated Token Account program* [online]. 2023 [visited on 2023-20-08]. Available from: https://spl.solana.com/associated-token-account.

[20] SOLANA. *Program Library: Token-2022 program* [online]. 2023 [visited on 2023-20-08]. Available from: https://spl.solana.com/token-2022.

[21] SOLANA. *Program Library: Account Compression program* [online]. 2023 [visited on 2023-20-08]. Available from: https://spl.solana.com/account-compression.

[22] SOLANA Mobile. *Solana Mobile Stack Overview* [online]. 2023 [visited on 2023-20-08]. Available from: https://docs.solanamobile.com/getting-started/overview.

[23] SOLANA. Documentation: Solana Pay [online]. 2023 [visited on 2023-20-08]. Available from: https://docs.solanapay.com/.

[24] Metaplex Documentation. Introduction [online]. 2023 [visited on 2023-20-08]. Available from: https://docs.metaplex.com/.

[25] Metaplex Documentation. Token Metadata Overview [online]. 2023 [visited on 2023-20-08]. Available from: https://docs.metaplex.com/programs/token-metadata/overview.

[26] Metaplex Documentation. Token Metadata Token Standard [online]. 2023 [visited on 2023-20-08]. Available from: https://docs.metaplex.com/programs/token-metadata/token-standard.

[27] Metaplex Documentation. Candy Machine Overview [online]. 2023 [visited on 2023-20-08]. Available from: https://docs.metaplex.com/programs/candy-machine/overview.

[28] Metaplex Documentation. Resources Storage Providers [online]. 2023 [visited on 2023-20-08]. Available from: https://docs.metaplex.com/resources/storage-providers.

[29] Solana Foundation. *State compression brings down cost of minting 1 million NFTs on Solana to ~ $110* [online]. 2023 [visited on 2023-21-08]. Available from: https://solana.com/news/state-compression-compressed-nfts-solana.

[30] Solana Foundation. Solana keeps accelerating with ecosystem-born innovations like xNFTs [online]. 2023 [visited on 2023-21-08]. Available from: https://solana.com/news/solana-ecosystem-innovation-xnft.

[31] Solana Cookbook. Gaming [online]. 2023 [visited on 2023-20-08]. Available from: https://solanacookbook.com/gaming/intro.html.

ackee | blockchain security

Thank you!

Ackee Blockchain

Prague, Czech Republic

@AckeeBlockchain

https://discord.gg/wpM77gR7en