Nervos CKB

- Start Date: 2018-01-02
- version: 6eb7c7e85e1ff4c9cb63a3f1b4b6ed3d995fab84
- PR: #TBD

A general purpose common knowledge base.

Abstract

This document provides an overview to CKB, the core component of the Nervos project. Nervos is a decentralized application platform with a layered architecture. CKB is the foundational layer of Nervos, as a general purpose common knowledge base to provide data, asset and identity services.

Background

As more applications emerge, the current generation blockchain technology has shown its limitations in universality, scalability, incentive design and trust model, making it increasingly difficult to meet the demand of today's real world use cases.

Bitcoin is the first blockchain network in the world, designed as a peer to peer cash ledger. The Bitcoin ledger's application state is maintained by the Bitcoin network. UTXO (Unspent Transaction Output) is the basic storage unit of the ledger. Users can use wallets to spend current UTXOs, generate new UTXOs, and package them into transactions to send to the Bitcoin network for validation and consensus. UTXOs have both the cash amount and ownership information expressed with lock scripts. Users have to provide proper unlocking data to spend UTXOs. Due to limitation of the UTXO data structure and lock script, it's difficult to record other types of assets and data in the Bitcoin ledger. While solutions like Colored Coins, Meta Coins or hard forks are possible, they are unsafe, inflexibile, and expensive.

Ethereum brought us a general purpose distributed computation platform with the introduction of smart contracts. The ethereum network maintains a world state of accounts. Smart contracts are accounts with code stored inside, together with a 256 bits K/V store. Users can send two types of transactions on Ethereum: the first is to create a contract and deploy it on the blockchain; the second type is to send input data to a specific deployed contract. This executes the code stored in the contract and updates contract's internal state. Ethereum's smart contract design provides a more general computation model, allows more flexibility, and solves some of Bitcoin's problems, but it still has limitations:

- Scalability problems: Ethereum's design focuses on the state machine's events (Figure 1). With a Turing complete language and transactions containing state transition inputs (instead of new states themselves), it's difficult for full nodes to determine dependencies between transactions. This makes it difficult for nodes to process transactions in parallel. Because states are not stored on-chain, potential sharding solutions also need to put in extra effort on data availability problems.
- Undeterministic state transition: in Ethereum, the contract state is updated by the contract code, which depends on the execution context (such as the internal state of the callee contract). Users have no way to determine the exact execution result when they send transactions.
- Mono-Contract: Ethereum smart contracts tightly couple computation and storage. Users have to use the paradigm of accounts, EVM bytecode and the 256 bit K/V database to implement all business scenarios. This is not efficient nor flexible.

The economic models of current blockchains also face challenges. With more users and applications moving to blockchain platforms, the data stored on blockchains also increase. Current blockchain solutions care more about the cost of consensus and computation, making it possible for user to pay once, and have their data occupy full nodes' storage forever. Cryptocurrency prices also are highly volatile. Users may find it difficult to pay for high transaction fees when the price of the cryptocurrency increases.

Current blockchain technologies also pursue extreme decentralization, requiring full nodes in the network to be completely equal peers. This imposes constraints on the design of blockchain systems, making it harder for them to meet the demands of real world applications. The hardware cost of running full nodes becomes ever more expensive with the inflation of on-chain states. There are less and less users who're willing to run full nodes. At the same time, users are increasingly relying on mobile devices and mobile apps to access the Internet, instead of desktop based web apps. This exaberates the problems of completely equal peer design of full nodes. Having multiple types of blockchain nodes is going to be the norm of the future.

Based on all above, we conceived and designed Nervos CKB with a completely decoupled new paradigm for DApps. Nervos CKB supports more general-purpose computation and storage, comes with better scalability and more balanced economic incentives, and is more friendly to mobile devices. We hope Nervos CKB can become the common knowledge base of the 7.6 billion people, and the foundation of all kinds of decentralized applicaitons.

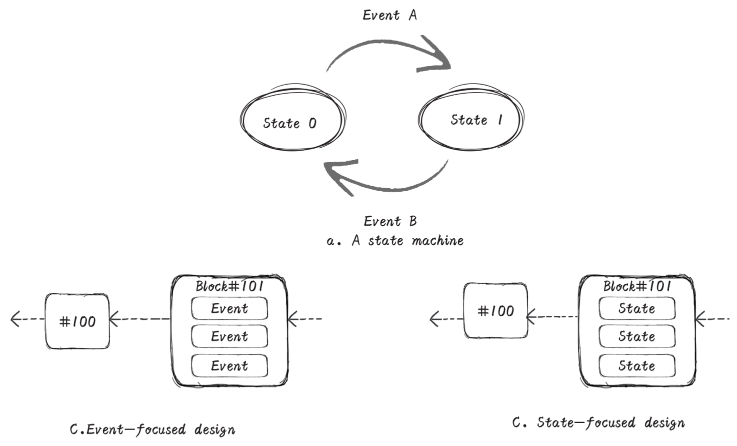


Figure 1. Event-focused vs. State-focused Design

Overview

Nervos CKB (CKB for short) is designed as a general purpose common knowledge base (Appendix: Common Knowledge Base). The CKB network consists of three types of nodes:

- Archive Nodes: Full nodes on the CKB network. They validate new blocks and transactions, relay blocks and transactions, and keep all historical transactions on disk. Archive Nodes can increase the overall robustness of the system, and provide query services for historical data.
- Consensus Nodes: Nodes that participate in the CKB consnesus protocol. Consensus Nodes listen to new transactions, package them to blocks, and achieve consensus on new blocks. Consensus Nodes don't have to store the entire transaction history.
- Light Clients: Users interact with the CKB network with Light Clients. They store only very limited data, and run on desktop computers or mobile devices.

CKB nodes together form a peer to peer distributed network to relay and broadcast blocks and transactions. Consensus Nodes run a hybrid consensus protocol (hybrid consensus), to reach consensus on newly created blocks at a certain time interval. The new block will be acknowledged by all nodes and added to the end of CKB blockchain. The transactions in the new block will update CKB's states.

A New DApp Paradigm

CKB proposes a new decentralized application paradigm. The paradigm consists of the following five components:

- Cell
- Type
- Validator
- Generator
- Identity

With those five components, we completely decouple decentralized applications into 3 aspects: computation, storage and identity. On top of this, computation is further divided into state generation (Generator) and state validation (Validator); Storage (Cell) is made more generic to support any structured (Type) data. The DApps in the CKB use Types to define the appropriate data structure and store application data in Cells. The application logic is implemented with Generators, and the state validation logic is implemented with Validators. Generators run on the client side to generate new states, which get packaged into transactions and broadcast to the entire network. Consensus nodes in CKB network first authenticate the submitter of the transaction, then validate new states in the transaction with Validators and put valid transactions in transaction pool. Once a new block is generated/received, new states in the block are committed into the CKB.

The CKB's design of data flow and economic incentives is focused on states - transactions store new states, instead of events that trigger the state machine. Therefore, the CKB blockchain stores states directly, and states are synchronized together with blocks, with no need for extra state synchronization protocols. This reduces complexity of the system and increases system availability. A DApp's states are stored in Cells. Generators and Validators are pure functions without internal states, relying entirely on inputs to produce outputs, and can be easily combined to form more complex logics. CKB's computation paradigm is very close to the Lambda Calculus, which can achieve the same computation capability as the Turing machine.

Table 1 compares Bitcoin, Ethereum and Nervos CKB:

	Bitcoin	Ethereum	Nervos CKB
Knowledge Type	Ledger	Smart Contract	General
Storage	UTXO	Account K-V Store	Cell
Data Schema	N/A	N/A	Type
Validation Rule	Limited (Script)	Any (Contract)	Any (Validator)
State Write	Direct (User)	Indirect (EVM)	Direct (User)
State Read*	No	Yes	Yes

Table 1. Comparison of Bitcoin, Ethereum and Nervos CKB (* State Read refers to on chain readability only, which means if the state can be read during on chain validation. Chain state is transparent to off chain reader.)

State Generation and Validation

State generation and validation are separated in CKB. They can use different algorithms, or the same algorithm.

For general business scenarios, currently there are no generic ways to simplify a generation algorithm to get a more efficient validation algorithm. In this case we can use the same algorithm for state generation and validation: the client side uses the algorithm to generate new states, then full nodes run the same algorithm using the transaction's inputs and compare the output states with the new states in the transaction. If the states match, then the validation passes. When the same algorithm is used, state generation and validation have the same computational complexity, but running in different environments. The advantages of separating generation and validation are:

- Deterministic state transition: Certainty of transaction execution is one of the core pursuits of distributed applications. Certainty in transaction latency (see Hybrid Consensus) has seen a lot of attention, but certainty in transaction execution result hasn't seen much discussion. If new states are generated on full nodes, a transaction's creator can't be certain about its execution context, then it could generate unexpected outputs. In CKB, users generate new states on the client side. They can confirm the new states before propagating it to the network. The transaction outcome is certain: either the transaction passes validations and the new state confirmed by the user gets accepted, or the validation process fails without state changes (Figure 2).

- Parallelism: If new states are generated on full nodes, the nodes won't know the state read or written by the transaction beforehand, therefore nodes can't determine the dependencies between transactions. In CKB, because transactions include old states and new states explicitly, nodes can see dependency relationships between transactions (see Transaction). Independent transactions can be processed in parallel in many ways, such as on different CPU cores or sent to different shards. Parallelism is the key to blockchain scalability solutions.
- Distributed computation: the system's efficiency improves when we utilize computation resources on the clients and lower the computation load on nodes.
- More flexible client side implementation and integration: even when the algorithms are the same, generation and validation can be implemented differently. The client side has the flexibility to choose the programming language for better performance. It's also possible to integrate the generation logic into client sdie application runtimes to give the best user experience.

In many specific scenarios, we can find validation algorithms much more efficient than generation algorithms. The most typical examples are the UTXO transactions and asymmetric cryptographic signatures. Other interesting examples include sorting and searching algorithms: the computational complexity for quick sort, one of the best sorting algorithms for the average case, is `O(NlogN)`, but the algorithm to validate the result is just `O(N)`; Searching for the index of an element in a sorted array is `O(logN)` with binary search, but its validation only takes `O(1)`. The more complex business rules are, the higher probability that there can be asymmetric generation and validation algorithms with different computational complexity.

The throughput of nodes can greatly improve with asymmetric generation and validation algorithms. Moving more details of computation to the client side is also good for protection of the algorithms or privacy. With the advancement of cryptography, we may find methods to design generic asymmetric algorithms, such as general purpose non-interactive zero-knowledge proof technologis. CKB's architecture is designed to provide proper support when it happens.

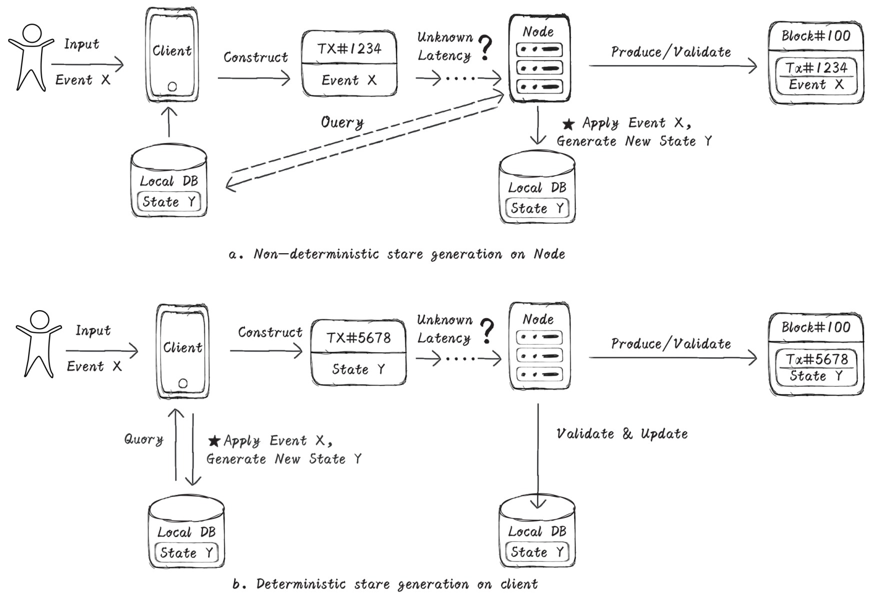


Figure 2. Non-deterministic vs. Deterministic State Generation

We're going to provide an overview of the Cell data model in CKB, with the goal to better explain the functionality of the CKB itself. In the actual implementation of the CKB, we need to consider other factors including incentives and execution efficiency, and the data structure will be more complicated. Technical details of CKB can be found in its technical documents.

Cell

There are two sources of trust for data in the CKB: the first type is when the data is independently verifiable, therefore trust is built-in, and the second type is when the data is endorsed by identities in the system. Therefore, the smallest storage unit in CKB must include the following elements:

- the data to be stored
- validation logic of the data
- data owner

Cells are the smallest storage units of CKB, users can put arbitrary data in it. A cell has the following fields:

- type: type of the cell (see Type)
- capacity: capacity of the cell. The byte limit of data that can be stored in the cell.

- data: the actual binary data stored in the cell. This could be empty. Total bytes used by cell, including data, should always be less than or equal to the cell's capacity.
- owner_lock: lock script to represent the ownership of the cell. Owners of cells can transfer cells to others.
- data_lock: lock script to represent the user with right to write the cell. Cell users can update the data in the cell.

Cell is an immutable data unit, and it can't be modified after creation. Cell 'updates' are essentially creating new cells with the same ownership. Users create cells with new data through transactions, and invalidate old cells at the same time (see Life Cycle. This means CKB is a versioned data store, with the set of new cells representing the current version, and invalidated cell set representing the history.

There're two rights with cells - ownership and usage right. Cell's `owner_lock` specifies its ownership, the right to transfer cell capacity; Cell's `data_lock` specifies its usage right, the right to update the cell's data. CKB allows users to transfer cell capacity all at once, or transfer only a capacity fraction, which will lead to cell creation (e.g. a cell with `capacity=10` becomes two cells with `capacity=5`). The rate of increase of CKB's total capacity is decided by Consensus Participating Rate and Liquid Voting.

Cell's lock scripts are executed in CKB's VM. When updating data or transferring ownerships, users need to provide the necessary proof as inputs of the lock scripts. If the execution of lock scripts with user provided proof returns true, the user is allowed to transfer the cell or update its data according to its validation rules.

The lock scripts are the authentication mechanism of the cells. The scripts can represent a single user, as well as a threshold signature or more complicated schemes. Cells come with better privacy. Users can use different pseudonyms (by using different lock scripts) to lock different cells. Cell's owners and rightful users can be the same or different users, which means users don't have to own cells to interact with the CKB. This lowers the barrier of entry of the system and encourages adoption.

Life Cycle

There are two phases in the life cycle of Cells. Newly created cells are in the first phase P1. Cells are immutable data objects. Updates to cells are done through transactions. Transactions take the P1 Cells to be updated as inputs, and output new P1 Cells with new states produced by the Generator.

Every P1 Cell can and only can be used once - they can't be used as inputs of two different transactions; after use, P1 cells enter the second phase P2. P2 Cells can't be used as transaction inputs. We call the set of all P1 Cells as P1CS (P1 Cell Set). The P1CS has all the current states of the CKB. We call the set of all P2 cells as P2CS (P2 Cell Set). The P2CS has all the historical states of the CKB.

Full nodes on the CKB only needs P1CS to validate transactions. They can deploy certain strategies to clear P2CS. P2CS can be archived on Archive Nodes or distributed storage. CKB light clients only need to store block headers and specific cells, and don't need to store the entire P1CS or P2CS.

Type

CKB provides a type system for cells and users can define their own types. With the type system, we can define different structures of common knowledge and their corresponding validation rules.

To define a new cell type we must specify two essential components:

- Data Schema: defines the data structure of cells
- Validator: defines the validating rules of cells

Data schema and Validator themselves are common knowledge, stored in cells. Each cell has one and only one type, while different cells could be of the same type or different types.

The data schema defines the data structure of cells in this type, so that the validator can interpret and interact with the data. Validators are verifying programs that run on nodes, in CKB's virtual machine. A validator uses the transaction's deps, inputs and outputs as program inputs (Transaction), and returns a boolean value on whether the validation is successful. The creation, update and destroy of cells can use different validation rules.

Index

Users can create indices when they defin a data schema. The CKB provides extra support for indexed fields, such as conditional query and/or aggregate functions which can be used in validators or `owner_lock` / `data_lock` scripts. For example, to start a capped crowd sale, the starter of the sale can create an Identity Cell (Identity), and use conditional query and aggregate functions to tell whether the sum of tokens belong to this specific identity is larger than or equal to crowd sale's goal.

Identity

Identity is a System Type. Users can create any number of identity cells to represent themselves, which can be used as other cell's `data_lock/owner_lock` scripts. If a cell uses an identity cell as its `*_lock` script, its update or transfer requires unlock script of the identity cell's `data_lock` (Figure 3).

Identity in the CKB is generalized identity that could represent any sides of individuals or machines. Identity cell is the core component of the NIP (see Nervos Identity Protocol Paper for details). With the NIP, the Nervos network brings in the CA certificates system to be compatible with the current PKI system. Users can have identities in CKB, and decentralized applications can be built on top of those identities. Users can store their public profiles or digests of profiles in identity cells, and only provide details to decentralized applications when neccessary.

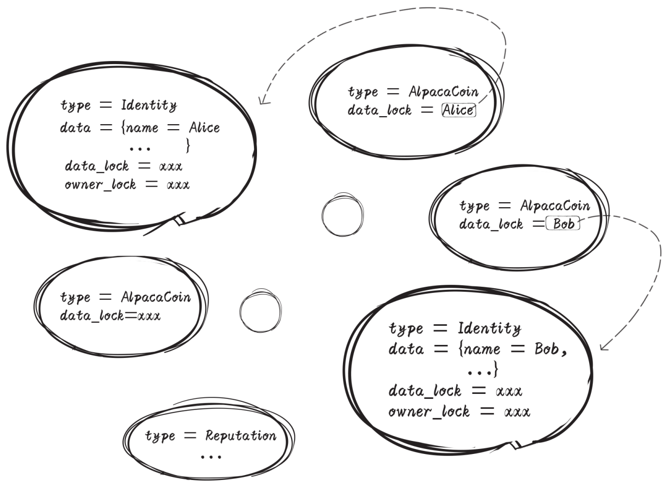


Figure 3. Identity Cell

Cell is a more generic storage model compared to the UTXO or the account model. Both the UTXO and the account models can express relationships between assets and their owners. The UTXO model starts with assets to define ownerships (with the lock script), while the account model starts with owners to define assets (with the balance). The UTXO model makes the ledger history more clear, but its lack of explicit accounts makes its already inexpressive scripts harder to use. There's also no way to store account metadata such as authorizations conveniently. The account model is easy to understand, supports authorizations and identities well, but is not easy to process in parallel. The Cell model, with types and identity takes the best of both worlds, to create a more generic data model.

Transaction

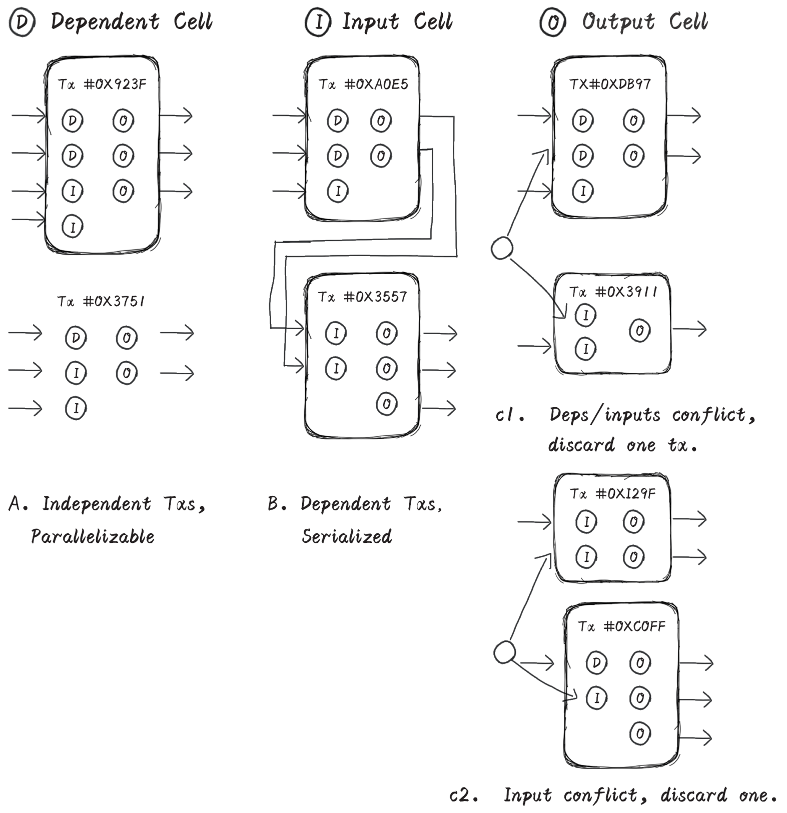
Transactions express the transfer and/or update of cells. In a single transaction user can transfer cells to others, or update one or more cells data. A transaction includes the following:

- deps: dependent cell set, provides read-only data needed by validation, must be references to P1 cells and/or user inputs.
- inputs: input cell set, includes cells to be transfered and/or updated, must be references to P1 cells with corresponding unlock scripts.
- outputs: output cell set, includes all newly created P1 cells.

Because cells are immutable, we can't modify cells directly, instead we create new versions of cells. All of cell versions can be linked together, forming a "cell version chain": a cell capacity transfer created the cell's first version, and following updates became its historical versions. The head of the cell version chain is its current version. The CKB is the set of all cell version chains. The set of all cell heads is the current version of the CKB.

The `deps` and `inputs` in CKB transactions make it easier for transaction dependency determination and parallel transaction processing (Figure 4). Different types of cells can be mixed and included in a single Transaction to achieve atomic operation across types.

The design of CKB cell model and transactions is friendly to light clients. Since all the states are in blocks, block synchronization also accomplishes state synchronization. Light clients only need to synchronize blocks, and don't need to perform state transition computations. If we only stored events in blocks, we would've needed full nodes to also support state synchronization. This extra protocol can be difficult for large deployments, because the incentive to do so is not defined within blockchain protocol. We define state synchronization in the protocol itself, and this makes light nodes and full nodes more equal, leading to a more robust and distributed system.



D Dependent Cell
I Input Cell
O Output Cell
Tx #0X923F
Tx #0X3751
Tx #0XA0E5
Tx #0X3557
TX#0XDB97
Tx #0X3911
c1. Deps/inputs conflict, discard one tx.
Tx #0X129F
Tx #0XC0FF
A. Independent Txs, Parallelizable
B. Dependent Txs, Serialized
c2. Input conflict, discard one.

Figure 4. Transaction Parallelism and Conflict Detection

Generator

Generators are programs to create new cells for given types. Generators run locally on client side, use user inputs and existing P1 cells as program inputs, create new cells with new states as outputs. Generator used inputs and generated outputs together form a transaction (Figure 5).

Validator and Generator may use the same algorithms or different algorithms (Overview). Generator may take one or multiple references to cells with same or different types as inputs, and generate one or multiple new cells with same or different types as outputs.

By defining Data Schemas, Validators and Generators, we can implement any data type's validation and storage in the CKB. For example, we can define a new type for `AlpacaCoin`:

```
Data Schema = {amount: "uint"}
```

```
// pseudo code of checker check():
// 1. check all inputs have valid unlock scripts
// 2. Calculate the sum of all AlpacaCoin amounts in inputs as IN
// 3. Calculate the sum of all AlpacaCoin amounts in outputs as OUT
// 4. Return the equality of IN and OUT
Validator = validate(context ctx, inputs, outputs)

// pseudo code of generator gen():
// 1. Find all cells of the AlpacaCoin type that the user can spend
// 2. Based on the receiver address and amount given by user, generate
//    new AlpacaCoin type cells that belong to the receiver and the
//    change cells belong to the sender.
// 3. Return a list of all used cells and new created cells, which
//    would be used to create a transaction.
Generator = gen(context ctx, address to, uint amount, ...)
```

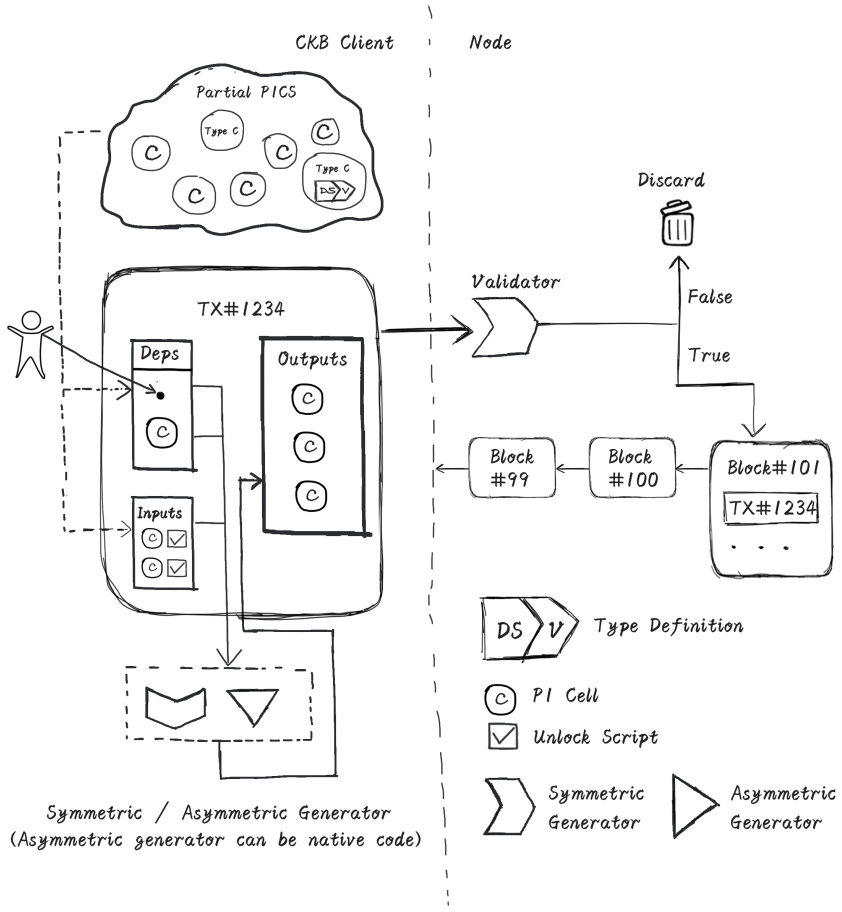


Figure 5. Transaction and Cell Generation/Validation

Layered Network

In Nervos network, CKB and the generators form a layered architecture. CKB is the common knowledge layer providing a data store for distributed applications, while generators is the data generation layer implementing application logic. CKB only cares about new states provided by generators, and doesn't care about how they are generated. Therefore generators can be implemented in many different ways (Figure 6).

The layered architecture separates data and computation, giving each layer more flexibility, scalability and the option to use different consensus methods. CKB is the bottom layer with the broadest consensus. It's the foundation of the Nervos network. Different applications prefer different consensus, forcing all applications to use CKB's consensus is inefficient. In Nervos network, application participants can choose appropriate generators and consensus based on their real needs, and the only moment they need to submit states to CKB, to get wider agreement, is when they need to interact with other services outside of their local consensus.

Possible generator forms include (but not limit to) following:

- Local generator on client

  Generators run directly on the client's devices. The generation algorithms can be implemented in any programming languages, using CKB light client API or CKB sdk.

- State services

  Users may use traditional web service to generate new states. All current web services may work with CKB in this way, which will gives more trust and liquity to the state of services. For example, game companies may define CKB types and rules for game props, implement its game as state services, which generate game props data and store it in CKB.

  Information sources may use identity and state services together to implement Oracles, to provide useful information for other distributed applications in Nervos network.

- State channels

  Two or more users may use peer to peer communication to generate new states. Participants of the state channels must register themselves on CKB, and obtain other participants' information. One participant may provide security deposits on CKB, to convince other participants on the security of the state channel. The participants may use consensus protocol or secure multi-party computation to generate new states.

- Generation chains

  A generation chain is a blockchain used to generate new states on CKB. Generation chain may be permissionless blockchains (such as an EVM compatible blockchain) or permissioned blockchains (such as CITA or Hyperledger Fabric). On permissioned blockchains consensus is reached in a smaller scope, users get better privacy and performance. On generation chains, participants perform computation together and validate the result with each other, and commit blockchain states to CKB to get wider agreement when neccessary.

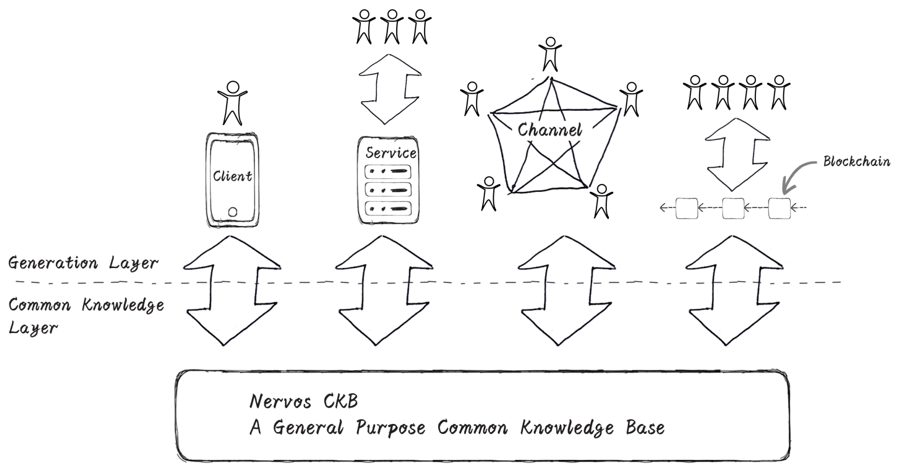


Figure 6. Layered Structure

Hybrid Consensus

In distributed environments with network delay and node faults, consensus algorithms aim to achieve two goals: correctness and performance. Correctness includes consistency (identical copies of data on each node) and availability (the system responds to user's requests within reasonable time). Performance includes transaction latency (the time between the submission of requests and the confirmation of execution results) and transaction throughput (number of transactions the system is capable of processing per second).

Permissionless blockchains run in open networks where nodes can join and exit freely, and there's no certainty when they're online. Those are difficult problems for traditional BFT consensus algorithms to solve. Satoshi Nakamoto introduced economic incentives and probalistic consensus to solve these problems, in turn so called Nakamoto consensus requires two more properties, openness and fairness, to guarantee its correctness. Openness allows nodes to join and exit the network freely, the blockchain must work properly no mattern there're 100,000 nodes or only 1 node in network; fairness ensures nodes get fair returns for their effort to keep the network functional and secure. Permissionless blockchain's consensus protocol also need to consider operational cost as part of their performance metrics.

The Nakamoto consenus, popularized by Bitcoin's Proof of Work, has excellent openness and availability. Nodes in the Bitcoin network may join and exit freely with barely cost, and the network performance remains constant with more participating nodes. However, its throughput is quite low - Bitcoin network's 7 transactions / second throughput can't meet the demand of real world business use cases. Even with 2nd layer scalability solutions (e.g. Lightening Network), where most of transactions can happen off chain, channel's opening and closing are still constrained by the performance of the root chain. The safety of 2nd layer solutions can be compromised when the root chain is too crowded. Nakamoto consensus use blocks as votes, which makes it take longer (up to 10 minutes to an hour) to confirm transactions, leading to inferior user experience. When there's a network partition, the Bitcoin network may continue to function, but it can't guarantee whether the transactions will be confirmed, therefore it's not suitable for business scenarios requiring high degree of certainty.

After 30 years of research, traditional Byzantine Fault Tolerance consensus algorithms can achieve throughput and transaction confirmation speed on par with centralized systems. However, it's difficult for nodes to join or exit freely, and the network's performance decreases rapidly with the increased number of consensus participating nodes. Traditional BFT consensus algorithms has lower tolerance to faults. When the network partitions, nodes can't achieve consensus and the network loses liveliness, making it difficult to meet the availability requirements of permissionless blockchains.

Through our own research and practice, we've come to realize that the traditional BFT algorithms function well in normal situations with simple logic, but need complicated logic to deal with fault situations; Nakamoto consensus functions with the same logic under either normal or fault situations, at the expense of happy path system performance. If we could combine Nakamoto Consensus and traditional BFT consensus algorithms in a secure way, the new hybrid consensus algorithm may give the best balance in consistency, availability, fairness and operational cost [3][4].

We will design and implement a new hybrid consensus algorithm for CKB. By combining Nakamoto consensus and the traditional BFT consensus, we retain the system's openness and availability, and take advantage of the excellent performance of the traditional BFT consensus under normal conditions. This minimizes transaction latency and greatly improves system throughput.

CKB uses Cell Capacity as consensus participation token. Nodes that wish to participate in the consensus process need to bond Cell Capacity as security deposit. The system uses the bond Cell Capacity for vote weight calculation and block reward distribution. If byzantine behaviors are observed on consensus nodes, other nodes can submit proof to have their deposits confiscated. The deposit mechanism increases the cost of byzantine behaviors and the security of the consensus.

Please check CKB Consensus Paper for more details.

Economics

The economic model is at the core of permissionless blockchains. Bitcoin solved the open network's consensus challenge for the first time with the introduction of economic incentives. Every blockchain network is an autonomous community bound together with economic incentives. A well designed economic model should incentivize all participants to contribute to the success of the community and maximize the utility of the blockchain.

CKB's economic design aims to motivate users, developers and node operators to work towards the common goal of forming and storing common knowledge. CKB's economic model is also designed to focus on states, using increased Cell capacity and transaction fee as rewards to provide incentives.

The creation and storage of states on the CKB have incurred cost. The creation of states needs to be validated by full nodes, incurring computational cost; the storage of states needs full nodes to provide storage space on an ongoing basis. Currently permissionless blockchains only charge one time transaction fees, and allow data from the transactions to be stored on all full nodes, occupying storage space forever.

In CKB, cells are basic storage unit of state. Unoccupied cell capacity is transferable, which brings cell liquidity. Occupied cell capacity can not be transfered thus they lose liquidity. Therefore, cell owners pay with liquidity of cell capacity for the storage cost. The larger capacity and longer time they occupy, the higher liquidity cost they pay. The advantage of liquidity payment, compared to upfront payment, is that it avoids the problem that the upfront payment could be used up, and the system would have to recycle the cells.

The price of cell capacity is direct measurement on the value of the common knowledge stored in CKB. Note that cells could have different owners and users, and owners can pay the liquidity cost on behalf of their users. Updating cell's data or transferring their ownerships incur transaction fees. Nodes can set the transaction fee levels that they're willing to accept. Transaction fees are determined by the market. Owners can also pay transition fees on behalf of their users.

Another obstacle preventing blockchains mass adoption is that transaction fees have to paid with native tokens of the blockchain. This requires users to acquire the native tokens before using the services, raising the barrier of use. On the other hand, users are used to the freemium business model and mandatory transaction fees doesn't meet the habits of mainstream users. By allowing cell owners to pay on behalf of users, CKB solves both of the problems, and provides to the developers more business model choices.

The majority of system transaction fees go to the block generation nodes. The rest of the fees will be used to support research, development and operations of the Nervos project. The percentage of this transaction fee distribution is determined with the liquid voting (Liquid Voting) process.

Other than paying for the cost of common knowledge's creation and storage, cell capacity can also be used for other purposes such as consensus security deposit and liquid voting. The security of CKB is directly related to the security deposits of the full nodes. The higher the total consensus deposits, the higher cost malicious behaviors will incur, and the more secure the system as a whole will be. Ensuring adequate consensus deposit, therefore system security, is one of the goals of Nervos' monetary policy. Adjusting inflation rate changes the risk free rate of return for consensus participants, and it can be used to adjust consensus nodes participation.

Please check Nervos CKB Economic Paper for details of the economic model.

Governance

As the core infrastructure of Nervos network, CKB has to evolve with the ecosystem on top of it. It has to keep functioning while adjusting runtime parameters or performing deeper system upgrades. We can see from the history that community consensus cost will stifle innovation and stagnate the ecosystem.

Therefore, CKB has built-in mechanisms for liquid voting and hot deployment, making the system a self evolving network.

Liquid Voting

The voting mechanism is important for the long term stability of the Nervos system. For example, CKB relies on a set of system parameters to function. While some of the parameters can be adjusted automatically, others need opinions from community and voting. Fixing bugs or the proposal/deploy of new feature also need support from community.

CKB supports liquid voting [5] (See figure 7). Every cell owner, weighted by their cell capacity, can participate in the decision making process of CKB's development. In liquid voting, users can set their own delegates, and delegates can also set their own delegates. Taking into account a proposal's technicality and incentives, different proposals can have different acceptance criteria, such as participation rate and support ratio.

Note that CKB's liquid voting is a tool to express community consensus, not one to form consensus. Before the votes, the community should use various communication channels to study the proposals in detail and form rough consensus.

Please check Nervos Governance Paper for details of the the liquid voting mechanism.

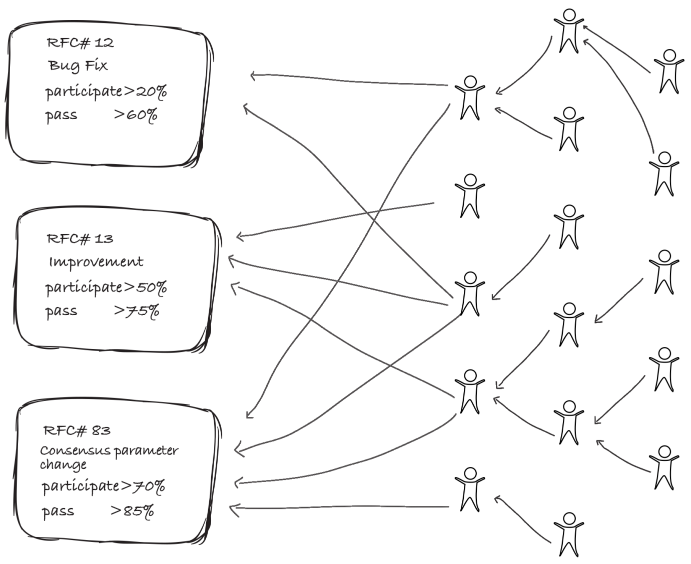


Figure 7. Liquid Voting

Neuron

Benefiting from the generality of cell model, we may implement and store some CKB's function modules in cells. We call this type of cells Neurons. Neurons are of a special cell type, and the user of the neurons are the CKB itself.

When system upgrade proposals are implemented as neurons, the community votes on its deployment with liquid voting. After community consensus is achieved, new neurons will be deployed to provide new features or fix bugs. Fine grained neuron upgrades significantly lower CKB's evolution friction.

Light Client

Strictly uniform architecture (every node has the same role, do the same thing) is currently facing serious challenges. On permissionless blockchains, the hardware capability of nodes vary a lot. Strictly uniform architecture not only puts high demand on user's hardware, but also fails to utilize the potential of high performant nodes. We see more and more users give up running full nodes and choose to run light clients or even clients rely on centralized service. Full nodes validate all blocks and transaction data, requiring minimum external trust. But they are costly and inconvenient to run. Clients rely on centralized service give up validations and trust the central service completely to provide data. Light clients trade a minimum trust for great cost reduction of validation, lead to much better user experience.

At the same time, mobile devices are becoming the main way people access the Internet. Native applications are also becoming more popular. Mobile friendliness is one of the design principles of CKB. Nervos dapps should be able to run smoothly on mobile devices and integrate with mobile platforms seamlessly.

CKB supports light clients. CKB aims to use authenticatable data structure to organize block headers, in order to substantially accelerates light clients synchronization. Benefiting from CKB's state focused design, light clients may obtain the latest states (P1CS) without having to repeat the computation. Light clients may also only subscribe to a small subset of P1 cells that they care about. With minimized local storage and bandwidth requirements, Nervos' light clients could provide better DApp experiences.

Summary

Nervos CKB provides a common knowledge layer for an exciting distributed application network. The design of Nervos CKB focuses on state, create a more general storage model, more balanced incentives, and a scalable paradigm for distributed applications.

References

1. Alonzo Church, Lambda calculus, 1930s
2. Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System", 2008
3. Vitalik Buterin, Virgil Griffith, "Casper the Friendly Finality Gadget", 2017
4. Rafael Pass, Elaine Shi, "Thunderella: Blockchains with Optimistic Instant Confirmation", 2017
5. Bryan Ford, "Delegative Democracy", 2002

Appendix. Common Knowledge Base

Common Knowledge

Common Knowledge is knowledge that's accepted by everyone in a community. Participants in the community not only accept the knowledge themselves, but know that others in the community also accept the knowledge. Generally, by the way they are formed, there can be three types of common knowledge:

The first type of common knowledge can be independently verified with abstract algorithms. For example, the assertion that "11897 is a prime number" can be independently verified with a primality test algorithm. In this context, the statement can become a piece of common knowledge, regardless whether the person who makes the assertion can be trusted.

The second type of common knowledge relies on a delegated verification process, typically to a trusted authority. For example, science discoveries require empirical evidence, in the forms of peer review and result reproducibility, to be accepted as common knowledge of the science community.The general public doesn't have the ability on their own to verify the evidence, but delegates their trust to the science community.

The third type of common knowledge requires a trusted party, and it's pervasive in business transactions. For a piece of data to become common knowledge to facilitate transactions, the participants of transactions have to all trust the party that backs the data. For example, in a centralized exchange, transactions imply trust on the exchange, thereby the accuracy of its data feed and the fairness of its match making algorithm. In the credit card point of sale context, the consumer and the business can complete a transaction based on their mutual trust on the financial intermediaries such as banks and credit card companies.

Blockchains are Common Knowledge Bases

In the past, the common knowledge is scattered in people's heads, and its formation requires repeated communications and confirmations. Today with the advancement of cryptography and distributed ledger technologies, algorithms and machine are replacing humans as the medium for the formation and storage of common knowledge. Every piece of data in the blockchain, including digital assets and smart contracts, is a piece of common knowledge.

Blockchain systems are common knowledge bases. Participating in a blockchain network implies accepting and helping validate the common knowledge in the network. Transactions are stored in the blockchain, together with their proofs. Users of the blockchain can trust the validity of the transactions, and know other users trust it too.

General Purpose Common Knowledge Base

A general purpose common knowledge base that's suitable for generation and storage of all types of common knowledge should have the following features:

- State focused, not event focused (Figure 1)
- Data model that's generic enough, with enough abstraction power that users can use to express the business domain
- A validation engine of common knowledge that's generic enough with sufficient abstraction power that users can use to express data validation rules.

If distributed ledgers are the "settlement layer" of digital assets, general purpose common knowledge bases are the "settlement layer" of all types of common knowledge. The goal of Nervos CKB is to become the state layer of the Nervos network as a general purpose common knowledge base. It provides the state and trust foundation for distributed applications.

"The various ways in which the knowledge on which people base their plan is communicated to them is the crucial problem for any theory explaining the economic process, and the problem of what is the best way to utilizing knowledge initially dispersed among all the people is at least one of the main problems of economic policy - or of designing an efficient economic system." - "The Use of Knowledge in Society", Friedrich A. Hayek, 1945