SagaChain™ Technical Whitepaper

By: David Beberman, Michael Holdmann, Jay Moore

May, 2020

Table of Contents

Abstract *4*

SagaChain Sharded Consensus Main Loop *6*

Current State Blockchain Architectures *6*

SagaChain Sharded Consensus Main Loop *6*

Block creation by the Leader. **6**

Leader Operations **7**

Block creation Stage 1 7

Leader Gossip Stage 4 7

Validator Operations **7**

Byzantine Fault Tolerant Validation, stage 2 7

PoW Operations, stage 3 8

Validator Gossip, stage 4 ..... **8**

Distributed Proof of Work *9*

Extensible Blockchain Object Model (XBOM) for parallelization *10*

Individual Native Coin Transactions **11**

Smart Contracts for Sharding **11**

Smart Object Assets Help Enable Sharding **13**

Extensible Smart Object Assets *14*

Overview **14**

Use and access rights **14**

Examples of Assets 14

Asset Creation 14

Asset Object Reference 15

Fractionalization of Assets 15

Multitiered Fractionalization of Assets **17**

Creation of Baskets of Assets **17**

Example Use Case Business Shares As Assets **17**

Brief Comparison of Extensible Smart Object Assets to Smart Contracts 18

Summary **18**

SagaCoin ($PSC) Financial Model for Incentivizing *19*

Existing Cryptocurrency Supply Management **19**

Volume of Token Exchange **19**

Digital GDP **19**

Digital GDP and Token Value with Fixed Token Supply 20

Digital GDP and Token Supply Management 20
Managing the SagaCoin ($PSC) Supply for Value Accumulation and Currency Usability 21
External GDP 21
Digital GDP Measurement 22
Smart Objects Supporting Digital GDP Measurement 22
Some Assumptions for Digital GDP Enabled Smart Objects: 22
Decentralized $PSC Supply Management with Digital GDP Metrics 23
Creating Digital GDP Metrics for Non-Conforming Smart Object Classes 23
Digital GDP Metrics and Conforming Smart Object Classes 24
External Economies and "Oracles" 24
Governance XBOM Objects of Operational Smart Assets 24
Overview 24
Governance Capabilities in Abstract 24
Governed Operational Smart Asset Architecture 25
Governance Voting 25
Governance Voting Management 26
Governance Asset Management 27
Change Request Code 27
Governance & Financial Models 27
Governance 27
Technology Board: 28
Monetary Board: 28
Treasury Board: 28
Community Board: 28
Financial Models 29
Monetary Policy Model 29
Equation of Exchange 30
Treasury Model 31
Gas Price Model 31
Node Staking Model 31
Prasaga Foundation 31
Conclusion: 32

Abstract

Prasaga has created a new blockchain infrastructure, SagaChain that addresses three fundamental issues:

1. Scalability of the Consensus Algorithm
2. Parallelization (i.e. Sharding) of the Smart Contract Architecture
3. Cryptocurrency to Incentivize Node Validators (aka Miners) and that is used for transactions rather than just a store of value

SagaChain addresses these three aspects with the following:

- Distributed Proof-of-Work (D-PoW) for scalability
- Extensible Blockchain Object Model (XBOM) for parallelization
- The SagaCoin ($PSC) Financial Model for cryptocurrency

SagaChain addresses blockchain scalability with algorithms that increase transaction throughput directly proportional to increasing quantities of node validators (miners). That increase also increases the amount of blockchain cybersecurity directly proportional to increasing quantities of node validators (miners). This is accomplished through the use of parallelization of the blockchain, known as sharding, and through the use of the proof-of-work algorithm applied in an approach that distributes across the blockchain shards.

The XBOM uses a "first-class object model" concept coupled with an adaption of the "system object model" concept to create a blockchain object-oriented user account model. Each user account contains a directed graph of objects, with each object maintaining its own state. An account may have new objects added to its account, or may be transferred to other accounts, or deleted, depending on the transactions executed and the class methods implemented for each of the various objects. This is in direct contrast to the existing blockchain smart contract model where user accounts only contain a balance of the native token of the specific blockchain, and all state is stored in separate smart contracts, not in the user's account.

The XBOM users' state account model enables deterministic parallel transactions between any disjointed or unconnected set of user accounts.

As an example: Consider user accounts A, B, C, D. If accounts A and B have a transaction, and accounts C and D have a transaction, because all object state is local to the accounts, these two transactions can be executed in parallel. Therefore, these two transactions can be executed on separate blockchain shards. By contrast, the existing blockchain smart contract model depends on all transactions being serialized, i.e. one after the other, because the state of
the smart contract is stored in a single account. This means that both transactions must be executed one after the other, even though the transactions are completely unrelated to each other.

This fundamental change in state management permits scaling with a sharded blockchain, such as the D-PoW consensus algorithm of the SagaChain.

The $PSC financial model addresses several aspects of limitations of the current cryptocurrency models. The well-known Bitcoin model, and the Ethereum model, both can be characterized as deflationary currency models that are non-responsive to variations in the economic "climate".

This has resulted in widely varying, but generally increasing pricing against fiat currencies for these models. A deflationary currency model has the effect of encouraging owners to hold the currency instead of spending it. Any time an owner spends their Bitcoin,
they lose the future value of the Bitcoin. An extreme example of this is the $800 million dollar pizza. An early Bitcoin enthusiast paid 10,000 bitcoins for a pizza. Less than 10 years later, those coins were worth greater than $800,000,000.

In contrast to a deflationary currency model, an inflationary cryptocurrency model would be created simply by mining ever increasing larger amounts of tokens. An inflationary currency model would have the effect of encouraging the immediate expenditure of the token to some other value, such as fiat currency. As a result, such a token might be used as a very short term means of transfer of value between different currencies or as an immediate point-of-sale purchase medium but would not be suited to holding for any length of time in a user account on a blockchain. Although there may be high velocity of transactions, with this model, the cryptocurrency would have little to no appreciation in value. This has a negative effect in attracting miners to the blockchain. Without the miners, the security of the blockchain collapses, and the cryptocurrency completely fails.

The $PSC addresses both the deflationary and inflationary aspects through a model that supports managing the rate of new tokens being added in to circulation by the miners (called mining) and the rate of tokens being taken out of circulation, (called burning). To avoid centralized control of the financial model, and thus to provide credibility to the value of the $PSC as a viable cryptocurrency, the management of the financial model uses a decentralized democratic governance model (i.e. voting) to control the various parameters of the model. The voting is conducted on SagaChain itself, creating a self-managed, trustless financial model that is responsive to the SagaChain's blockchain economy. As a result, the $PSC is internally stable within SagaChain which makes it suitable to be used as a currency.

The following sections describe the major components of SagaChain technology: Main Loop, Individual Shard Consensus; D-PoW; XBOM; XSOA; and $PSC Financial Model. This is followed by a brief description of the Prasaga Foundation and concluding paragraphs.

SagaChain Sharded Consensus Main Loop

SagaChain is made up of multiple individual blockchains or "shards". SagaChain main loop executes on each individual parallel blockchain independently. SagaChain uses a pat. pend. RAFT style protocol extended with byzantine fault tolerance and additional features. The blockchain serves as the ledger for the RAFT style protocol.

Election of the Primary Leader, the Secondary Leader and future Leaders use the pat. pend. Verifiable Random Function (VRF) which in addition to providing a selection means for election, also minimizes opportunities for collusion attacks.

Current State Blockchain Architectures

Blockchain designs consisting of multiple individual blockchains, or "shards", enable scaling of aggregate throughput of transactions by increasing the number of shards. Each shard uses a blockchain protocol for adding new blocks containing transactions to itself. The state of the art blockchain protocols used for such shards make use of variations of byzantine fault tolerant (BFT) protocols commonly called proof-of- stake (PoS). Such protocols depend on asymmetric keys for signature authentication by validator nodes on each shard independently. Although it has been shown that such PoS protocols applied to blockchain shards offer minimal computational effort, and thus are an attractive to the much higher computational effort of existing proof-of-work (PoW) protocols, relying solely on PoS signature authentication has significant drawbacks:

If a supermajority of private keys of the validators on a shard are compromised, a fake alternative blockchain can be presented, indistinguishable from the real blockchain.

The rate of block production cannot be determined by inspection of the blockchain itself. Whereas PoW provides a frequency estimate with a given amount of hashpower.

As a result, the "longest chain" decision used by PoW blockchains (i.e. Bitcoin) is not applicable.

What is needed is a blockchain sharding design that incorporates both protocols, PoS and PoW to address both the reduction of computational effort inherent in PoS and the "longest chain" decision inherent in PoW.

Therefore, the following blockchain shard design called SagaChain Main Loop combines PoS and PoW to complement the weaknesses in each protocol with the strengths of the other. Additional features of the design: enable the Verifiable Random Function to be derived directly from its operation; and generate PoW solutions for SagaChain Distributed Proof-of-Work protocol, which creates the long-term immutability of SagaChain and the solution to forks caused by network partitioning.

SagaChain Sharded Consensus Main Loop

Each SagaChain chain has a current set of nodes assigned to it. Assignment of nodes to SagaChain chains is described in "Means for Node Registration and Random Selection using VRF[2].

Each SagaChain chain performs the same main loop if/until the chain is terminated. It consists of four main stages:

Block creation by the Leader.

- Byzantine Fault Tolerant validation by

the selected validator nodes for the current block. (Using PoS staking model).
- Proof-of-Work solution by all nodes in the current set of nodes assigned to SagaChain chain.
- Block gossip (Using Distributed PoW)

Details for each step are described in the following sections. The following is a brief overview.

Leader Operations

Block creation Stage 1

The Leader combines the following into each new block:

- New transactions from its transaction queue
- Authenticated validation results of previous block
- PoW Solution of the previous block
- VRF value from the PoW solution
- Leader management information

The Leader hashes and signs the block to authenticate it, and multicasts it to the validator nodes that are selected via the VRF. At this point, stage 1 is completed. The Leader uses a local timer to detect catastrophic failures.

Leader Gossip Stage 4

Upon receiving a Proof-of-Work solution for the current block, with at least f+1 signatures:

$\sum$ node signatures >= f + 1 and f >= node count / 2

- The Leader gossips the block and the PoW solution to all nodes in the SagaChain.
- The Leader also unicasts to all other SagaChain Leaders as a distribution performance optimization.
- The Leader does not use SagaChain Modified BFT RAFT protocol for the unicast to the other SagaChain Leaders.
- No assumption of synchronization between SagaChain chains exists within SagaChain mainloop.
- A Leader receiving a validated block from the Leader of another chain, broadcasts the block to the nodes on its chain as a distribution optimization.

Validator Operations

Byzantine Fault Tolerant Validation, stage 2

A node on receiving a new block from the Leader, performs the following actions:

- Uses the VRF to verify it is a selected BFT validator for this block
  - if not, it forwards the block to the selected nodes given the VRF

- If the validator does not have the previous block or blocks requests the previous block or blocks
  - requests the previous block or blocks from the other selected BFT validators and the Leader
- if none of these nodes responds with the previous blocks needed, the validator requests from other nodes on SagaChain chain
- Validates the transactions in the block
- Creates a validation message with
  - the transaction validation results (i.e. votes)
  - the hash of the block
  - the validators signature of the hash
  - a hash of the message and signature of the hash
- The node multicasts its message to the other selected validators

- The node sends the message to the Leader

A node on receiving a validation message from another validator, performs the following actions:

- Authenticates the message
- Verifies the transaction validations agree with its own results
- Authenticates any signatures
- Adds its signature of the hash to themessage
- If the total signatures is less the 2f+1, and the validator has not heard from all other validators, broadcasts the message, to nodes it hasn't heard from, where 3f+1 = node count.
- If the total signatures is greater than 2f+1, and the validator has not heard from all other validators with 2f+1 signatures, sends the message to the Leader, and broadcasts to nodes it hasn't heard from.
- If the total signatures is greater than 2f+1, and the validator has heard from all other validators, does not broadcast the message.
- If the total signatures is greater than or equal to 2f+1, validator transitions to the PoW Solution stage, stage 3.

PoW Operations, stage 3

A node, on receiving a validation message with 2f+1 signatures, performs the following actions, in 2 rounds:

- Authenticates the message
- Authenticates the signatures
- Begins solving the PoW in round 1
- If solved the PoW and no other PoW solution was received in the interim:
  - do round signature authenticating PoW, send to signature and PoW to Leader, and broadcast.
- If node receives PoW solution from another node:
  - Verifies PoW solution and signatures, and stops any local PoW Solution work, otherwise ignores and continues PoW solution work
  - If signature count is less than 50%+1 of nodes, adds signature, sends to nodes that have not signed yet.
  - If signature count is greater than or equal to 50%+1 of nodes, begins round 2:
    - checks the second round of signature counts
    - If count is less than f, where 3f+1 = node count:
      - adds its signature to the second round, and sends a copy to the Leader
      - multicasts to nodes that have not signed the second round
    - If second stage count is >= f:
      - adds its signature to the second round
      - forwards copy to the Leader

Validator Gossip, stage 4

A node on completing stage 3, rounds 1 and 2 performs the following:

- Updates its state information using transactions in block B-1, that are verified in block B.
- Gossips block B and PoW solution referencing B

Leader does not have to participate as a validator.

Each new block contains the transaction validation results of the previous block, and new transactions to be validated. A transaction in block B is validated in block B + 1. This enables the validation stage to be

performed by the selected validator nodes, separate from the Leader. The Leader's responsibility is to create the blocks of transactions at the top of each cycle of the main loop, after the Leader has completed the gossip stage 4. A block is not considered complete until the transactions and the PoW solution have been gossiped.

If the Leader in stage 1 proposes a block B that references a PoW solution other than the that of block B–1 previously gossiped, any node receiving such a block requests a Leader change to a new Leader using the VRF from block B-2, and includes as proof of the Leader's error the PoW solution with the majority of signatures.

- Block numbering, and transactions at chain termination
- Blocks are numbered monotonically increasing. For each block number B, there is an associated transaction block and a PoW solution. On completion of the PoW solution, they are gossiped together.

A transaction block contains the verification of the transactions in block B-1 and new transactions. Therefore, a SagaChain blockchain that terminates normally shall have no new transactions in its terminal block Bt.

A SagaChain shard that terminates abnormally may have new transactions in its latest block, Bl, none of which shall be considered processed, and may be resubmitted.

Distributed Proof of Work
(D-PoW) for scalability

A computer-based method for combining individual hashpower of a plethora of shards that use a proof-of- work hash procedure such that each shard benefits from the hashpower from all other shards within the plethora of shards, whereby a chosen set of shards having a maximal combined individual hashpower, is referred to as a consensus.

PoW is only distributed in the sense that all nodes in Bitcoin or Ethereum style chains can independently mine and produce a block. Prasaga's SagaChain Distributed PoW (D-PoW) [1] is for distributing the PoW values among multiple independent chains. Thus, the PoW's for each chain are included in the other chains. This does not produce blocks or change the competitive nature of producing blocks.

The method to combine the hashpower of all the individual shards has the following prerequisites:

- The rate of block production for each shard is a known quantity, and may vary from shard to shard;
- PoW for each shard of a blockchain is produced periodically and independently;
- The value of the PoW difficulty for each shard is known and may vary from shard to shard.

The hashpower available on each shard or blockchain is derived by:

- hashpower = Block production rate / PoW difficulty (1)
- where a smaller PoW difficulty value implies a larger
- expenditure of computing resources to find a PoW solution, and vice-versa;
- and, where block production rate is defined in common units across all shards (e.g. seconds).

The combined hashpower of all the shards is the sum of the individual hashpowers:

combined hashpower = $\sum$ (Block production

rate / PoW difficulty)(2)

summed over the shards.

What D-PoW does, is to enable evaluating a group of chains over time for validity, where the determination uses the concept of largest hashpower for any competing groups of chains. Hashpower equates to expenditure of resources, which is identical to what both Bitcoin and Ethereum actually do.

Thus, the immediate consensus is a local matter for each blockchain, but the long-term probability of immutability takes all of the available hashpower into account. This protects directly against a long-term attack. The reason why this is particularly attractive is that it allows for signature-based consensus (PoS variants), but for the long term, eliminates the dependency on protection of private keys.

The evaluation stage which can take place at any time, and in particular helps with bringing up new nodes, can deal with adversary attacks that attempt to create multiple chains that appear to be valid forked off of the original chains. The grouping of connected, but disjoint sets of chains, enables the evaluation to determine which chains have maximum cumulative hashpower.

This approach also allows for dealing with short term network partitioning where one or more chains may be isolated, but rejoin their connection with other chains, as what will happen is that the maximum hashpower choice, will find those chains again.

When trying to support a sharded or parallel chain system, the issue of partitioning can either isolate one or more chains, or break the chains. In the former case, how to rejoin becomes an issue. One can either terminate the chains on rejoining the other chains or try to merge the state. Regardless, a solution of just assuming that such partitioning won't happen is not viable. Further, if a sharded system creates synchronous cross-chain dependencies, such that if one chain fails or is partitioned, the entire system stops.

Extensible Blockchain Object Model (XBOM) for parallelization

Most blockchains approach scaling by enabling parallel execution of multiple transactions on separate shards, without compromising the immutability and security of the blockchain, across all the shards. Prasaga looks at sharding from a different angle.
For the sake of argument, let's say that a consensus algorithm for sharding exists. Further, let's say that this algorithm runs on an open permission-less blockchain, and is available today. Even with this, do we get the scaling that is hoped for with sharding? To get a feel for this, we take a look at Amdahl's Law

$$S_{\text{latency}}(s) \leq \frac{1}{1-p}$$
$$\lim_{s\to\infty} S_{\text{latency}}(s) = \frac{1}{1-p}$$

This shows that the theoretical speedup of the execution of the whole task increases with the improvement of the resources of the system and that regardless of the magnitude of the improvement, the theoretical speedup is always limited by the part of the task that cannot benefit from the improvement. This essentially states that the throughput of a system, once all the parallelizable portions are maximized, is limited to the throughput of the serialized portions.

For blockchain sharding, Prasaga interprets this as meaning that the potential increase in throughput is currently limited to the quantity

of transactions that can be executed simultaneously on separate shards. That is, if a transaction on a shard needs data from another shard it has to synchronize the transfer of the data from the other shard. This is a point of serialization and given a large pool of shards, according to Amdahl's Law this dominates the throughput.

Individual Native Coin Transactions

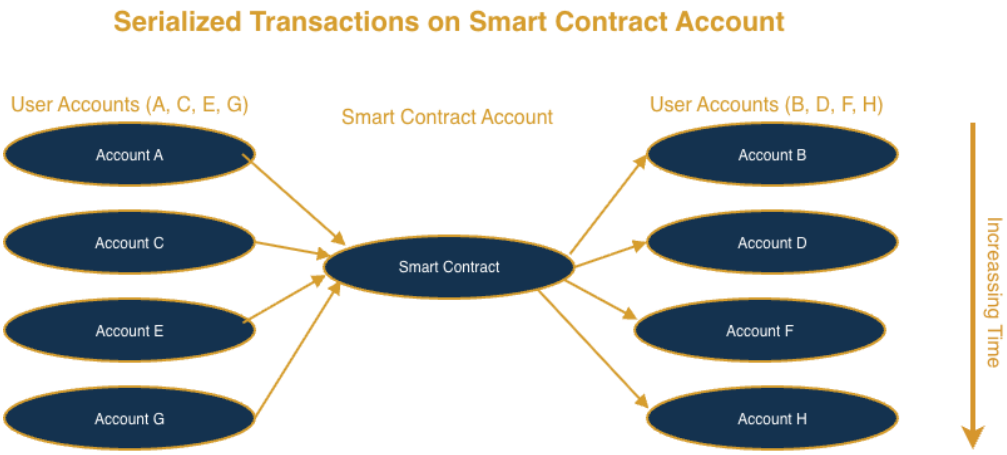
A transaction between two accounts limited explicitly to only transferring coin balances between the accounts, such as sending coin between two accounts, does not need data from any other account. Therefore, provided the data for both accounts is available on a particular shard, the transaction can be executed asynchronously with other transactions on other accounts. This scales with the number of shards and the number of disjoint transactions between pairs of accounts. As the number of accounts grows, one would expect that the opportunity for sharding such independent transactions grows as well. In the limit the throughput is dominated by the time it takes to execute a single transaction regardless of how many transactions are being executed at any given point in time. This is exactly the situation needed for improving blockchain throughput.

Smart Contracts for Sharding

Things do not work out so well dealing with smart contracts. A smart contract is implemented on the blockchain as a single ledger account with data state associated with the program code. Each transaction runs through the code changing the data state. Each change is recorded on the blockchain and verified with a hash representing the state.

To maintain a deterministic, consistent state of the blockchain, each smart contract can only be executed on one shard at a time, unless the state of the smart contract account itself can be sharded. In general, this makes the smart contract account execution the limiting factor for all of the accounts that are sending transactions to the smart contract.

As smart contracts are used for tokenization, it is highly likely that as a given token increases in circulation, its smart contract becomes a bottleneck for throughput, regardless of how many shards exists, as predicted by Amdahl’s Law. The following diagram depicts smart contract serialization:
With the current implementation model for smart contracts there appears to be only two possible ways for scaling with sharding:

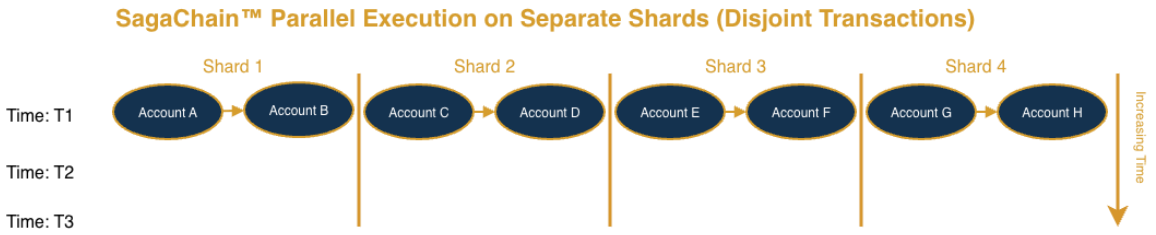


- Use multiple smart contracts segregated on shards
- Use deterministic multi-threaded smart contracts, (aka SIMD)

Multiple smart contracts can take advantage of sharding. For example, if each smart contract representing a token is assigned to a separate shard, then transactions for a given token do not affect transactions on other tokens. Although each individual token's transactions are limited to the throughput of the smart contract on its specific shard, with a large and growing number of tokens, the number of shards can grow linearly with them.

This doesn't solve the problem of the throughput of an individual smart contract, but it is an improvement over all the smart contracts on a single blockchain without sharding. Even with this approach, issues occur if any state is needed from a user account that is shared among the shards (e.g. native coin to pay for the transactions).

A technique from supercomputing, called vectorization, enables a program to execute sections of its code in parallel. This is known as single instruction multiple data ("SIMD"). Programs written for SIMD are written to be deterministic. In essence SIMD machines, which today are general purpose graphics processor units ("GPGPU"), shard their data across an array of processors. This works very well for certain classes of applications, such as matrix operations for graphics and similar.



Theoretically, this approach could be applied to blockchain smart contracts. That is, a smart contract could be written explicitly to support parallel execution of transactions in some manner. It is unclear exactly how this would be implemented. However, even with such a solution, writing a smart contract that is SIMD capable becomes significantly more complex, as one would expect.

Neither multiple smart contracts nor SIMD smart contracts are ideal solutions, although both may provide some opportunity for scalability.

Smart Object Assets Help Enable Sharding

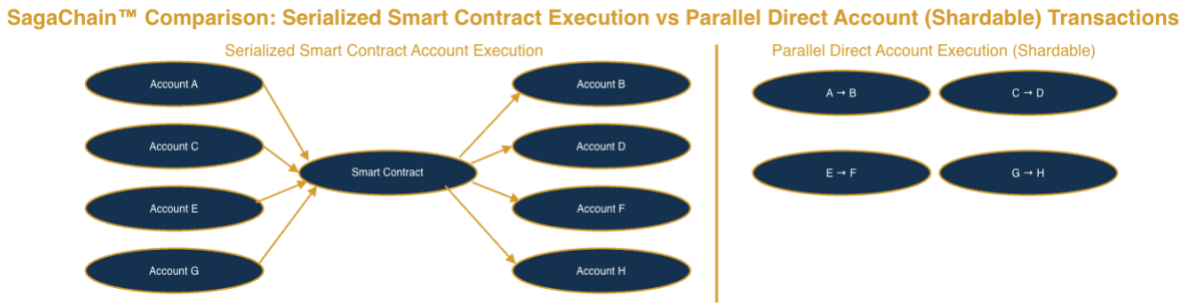
Limiting points for sharding with smart contracts is both sharding of state and the code execution. If there were a means to avoid transactions serializing on single smart contract state and code execution, sharding could increase throughput scaling. To put this another way, if there was a means for multiple instruction multiple data ("MIMD") execution, the opportunity for blockchain sharding would be significantly improved.

As was described in "Rethinking The Blockchain Account Concept", if each user account had its own state, instead of using separate smart contracts, then each user account could contain objects that represent assets, whether as tokens or other types of entities. As described in "extensible Smart Object Assets (XSOA)8, Smart Object Asset Ownership and Fractional Smart Object Asset Ownership With SagaChain Extensible Blockchain Object Model"9, XSOA's and references to XSOA's could be used to transfer ownership between accounts with transactions directly between the account states.

For example, given two sets of transactions, where each transaction is between different accounts, that is: one transaction is from account A to account B; and another transaction is between account C to account D, then the transactions can be executed on different shards simultaneously. Further, because the code for the XSOA's is independent of any of the accounts and may be different code for each of the transactions, sharding for a MIMD model can be accomplished, which means, different code on each shard and different data on each shard.

The limiting point for scale here is the number of transactions that can take place simultaneously between disjoint account sets. It would be expected that as the quantity of accounts grows, the opportunity for disjoint account sets within any group of transactions grows as well, which in turn would result in a growing opportunity for sharding.

Here is a side-by-side comparison of the serial versus parallel concept:



Using as a given the availability of a sharding consensus algorithm, an outstanding question is how to make use of such technology. Smart contracts inherently serialize transactions and other than a complex SIMD type solution, only offer scaling by using multiple separate isolated smart contracts. Even with that, each smart contract's throughput is limited to a single shard's throughput. By rethinking the user account to include state information, and using the XBOM model, SagaChain offers a solution to sharding scalability that scales with the number of accounts and disjoint transactions among the accounts. In addition to enabling inheritance and live code reuse, we believe that this is a significant solution to the blockchain scaling problem.

Extensible Smart Object Assets

Smart Object Asset Ownership and Fractional Smart Object Asset Ownership.

Overview

SagaChain introduces a new concept to blockchain solutions: The Extensible Smart Object Asset ("XSOA"). Using SagaChain Extensible Blockchain Object Model ("XBOM")10, virtually any reference to an asset whether a virtual or physical asset, may be owned by, and stored in a SagaChain account. Further it may be transferred (i.e. sold) to any other SagaChain account. As is described below this innovation enables a wide variety of asset ownership concepts including, but not limited to:

- Asset ownership and transfer
- Fractional asset ownership
- Asset ownership cashflows
- Voting rights

Use and access rights

Examples of Assets

An asset may be a single physical item such as a boat, car, house. It may also be an entity such as a business, or financial instrument.
An Extensible Smart Object Asset instance represents the asset on SagaChain in a one-to-one correspondence.

Asset Creation

An account holder creates a new asset by creating an object instance of ClassAsset or a subclass. The new smart object asset is created with the description information of the asset providing proof of ownership of the asset. The description information and proof of ownership of the asset that the asset object represents are specific to the asset, and as

such are determined by subclasses of the Class Asset. The newly created asset object instance is stored in the asset list in the creating account.

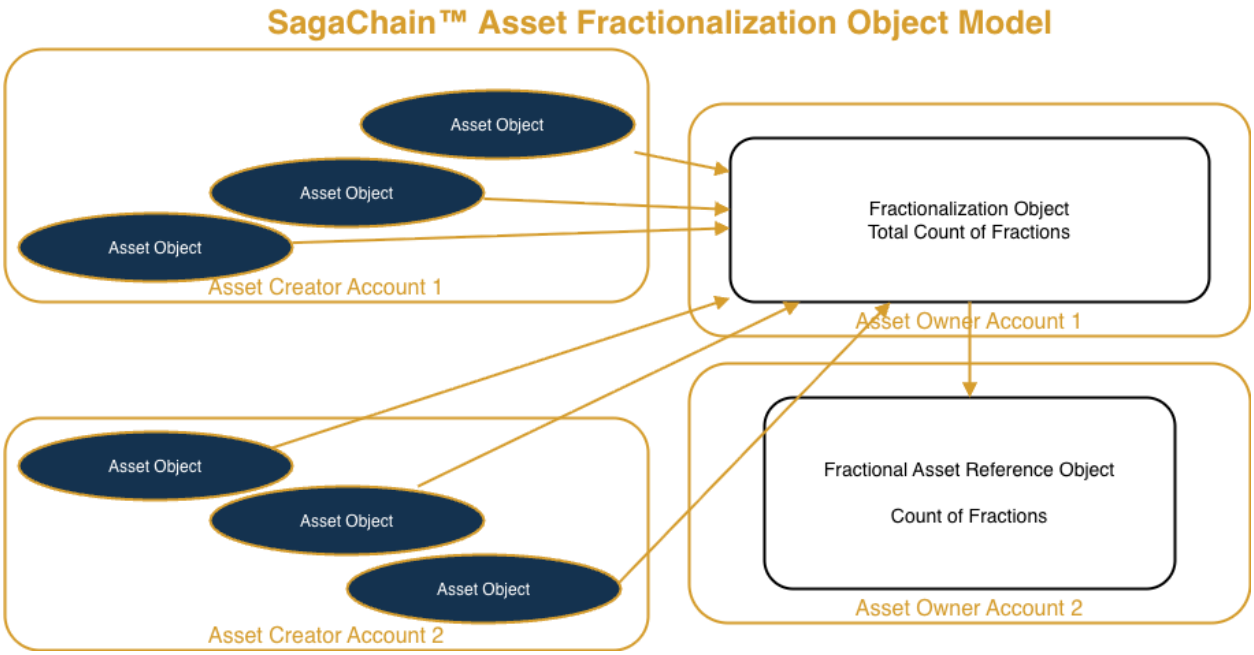
Asset Object Reference

Once an asset object has been created, if the creating account wants to enable transfer of ownership of the asset, the account holder creates a reference object to the asset object by creating an instance of ClassReferenceAsset. A ClassReferenceAsset object instance contains the object identifier of the asset object instance. These object instances act as proxies to the asset objects and are used to show ownership of the asset object.

Specifically, owning an asset object means by definition an account that contains an instance of a ClassReferenceAsset object that contains a reference to the asset object.

Implementation note: ClassReferenceAsset objects are passed between accounts for transfer of ownership. The underlying asset object is permanently stored in the creating account's state space, even if the creating account no longer owns the asset. As a result, the object identifier stored in the ClassReferenceAsset object is itself immutable.

The following depicts the fundamental asset ownership concepts:

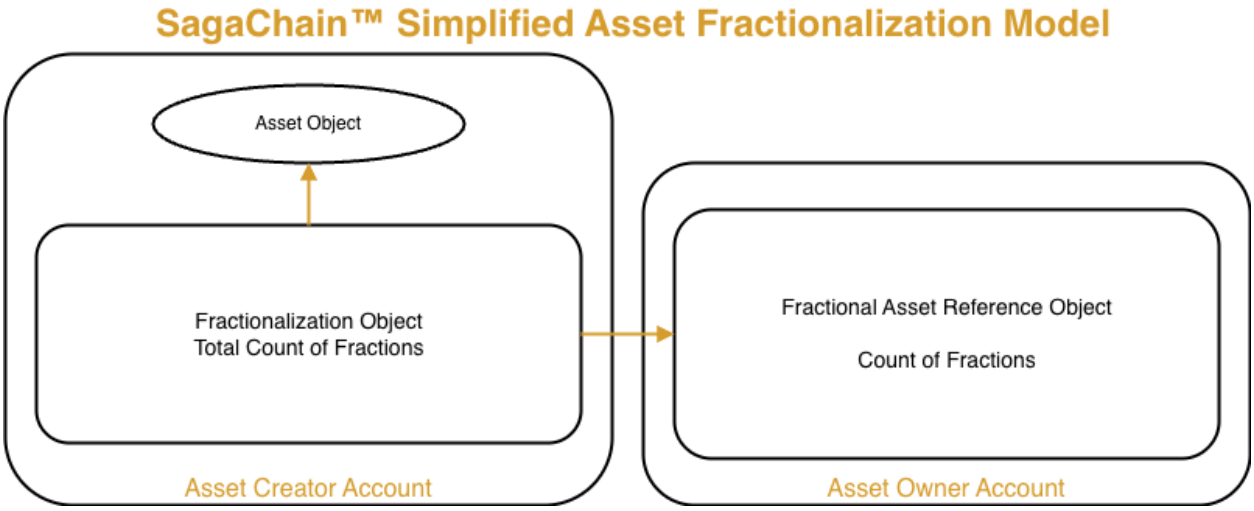


Each rounded rectangle represents an account on the SagaChain. The Asset Creator Account contains the asset objects that it instantiated. As is shown an account may create any number of asset objects. Two accounts are shown that own some of the assets created by the Asset Creator Account. Each owning account contains an Asset Reference Object instance with the object identifier of the owned asset.

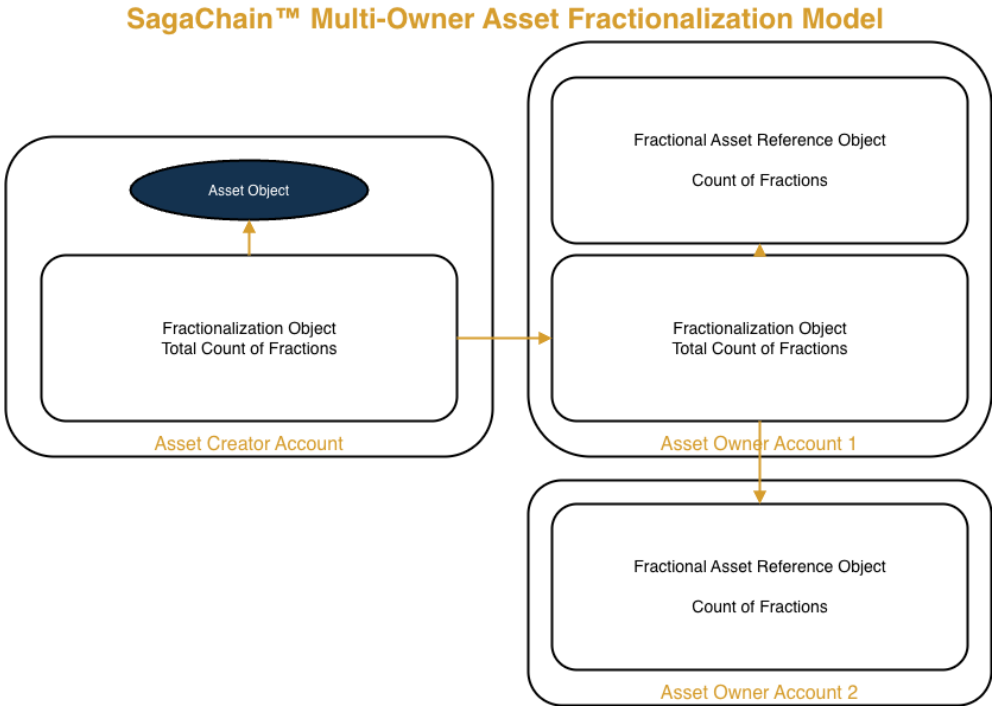
Fractionalization of Assets

The term fractional asset ownership refers to the general concept of owning a fraction of an asset. The most common use of fractional

asset ownership is shares in a corporation. Each share represents a fraction of ownership in the corporation. Other types of assets may be fractionally owned such as a yacht, corporate jet, vacation timeshare, etc. SagaChain classes support fractional asset ownership similarly to general asset ownership. The difference is that a fractionalization reference object is used to create the fractional ownership. This is depicted below:



A fractional asset reference object can be traded in a transaction just like an asset reference object and behaves functionally identically. For example, cashflows for a fractionalized asset would flow to the owners based on the fractions they own.

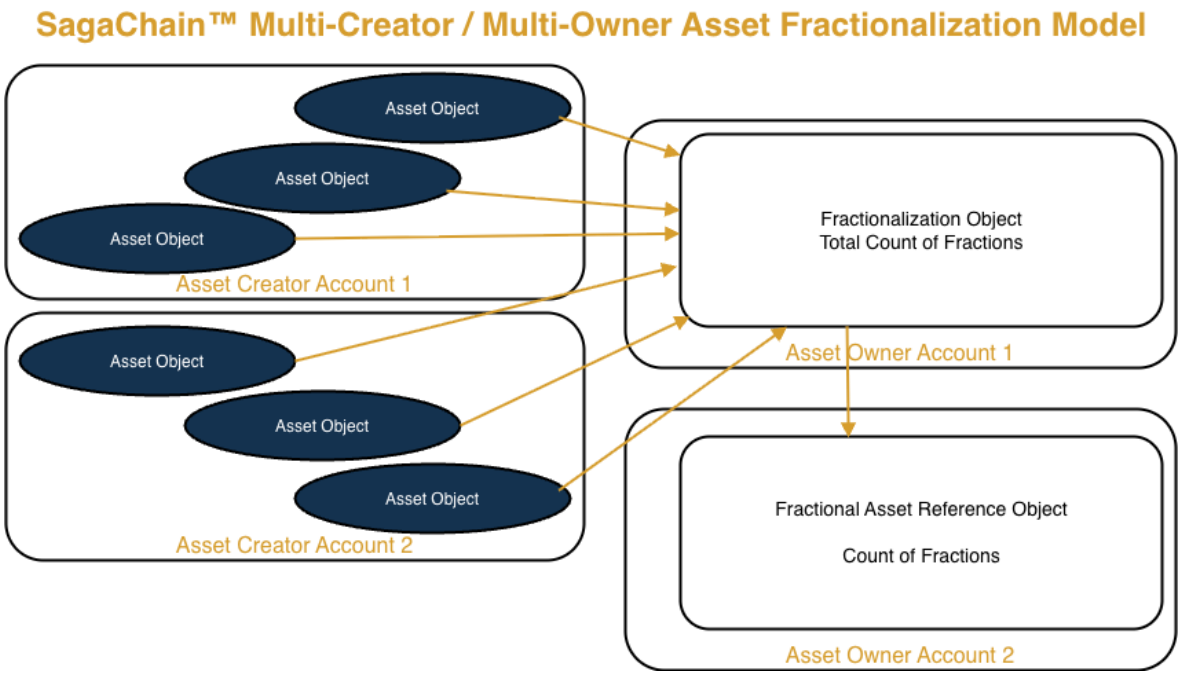


Multitiered Fractionalization of Assets

In this model, a fractional asset reference object is used as the asset object by a second fractionalization object. This new fractionalized asset can now be referenced by a new fractional asset reference object which can be transferred to new account. All ownership rights and cashflows would flow through the references.

Creation of Baskets of Assets

Assets may be grouped together into a "basket" and fractionalized. A fractionalization object may contain multiple asset object references from multiple accounts. This is depicted below with 2 create accounts.



As with the multitiered fractionalization of assets above, a basket of assets could consist of fractional assets as well as asset objects. All of the flow-through would behave identically.

Example Use Case Business Shares As Assets

Given the above fractional asset model, a business might do the following:

- create an account on the SagaChain
- create an instance of asset object to represent the business
- create an instance of a fractionalization object, initialized with N fractions

The business could then sell the fractions of ownership to any other account, with the new account containing a fraction asset reference object. The new account owner may sell the reference object to other accounts including back to the original business account.
If/when the business declares a cashflow (i.e. dividend), the account holder of the fraction

asset reference object sends a transaction to the fraction asset reference object to collect its cashflow from the fractionalization object, which in turn collects the cashflow from the underlying business asset object.

Brief Comparison of Extensible Smart Object Assets to Smart Contracts

A smart object asset, and the associated asset object references as described above differ from the smart contract model significantly and enable new ways of working with assets on blockchains.

Smart contracts are implemented as a single account, with associated code (e.g. Solidity code), and a single data space. If a Smart contract implements a token, such as an "altcoin" or a "security token", any account that owns a token is recorded as an account address in an array list in the smart contract account. Each new token is a new smart contract, with separate code and a separate single account data space.

The smart object asset model stores the asset reference objects in each individual owner accounts data space. This in sharp contrast to the smart contract approach where the account address (which functions as a reference) is stored in the single smart contract. As described above, the smart object asset concept implicitly supports multiple relationships without adding or changing any code. All that is required is creation of the relevant objects as standard transactions. This flexibility enables near limitless relationship structures between asset owning accounts without introducing the opportunity for programming mistakes.

Brief Comment on Sharding and Scalability with Extensible Smart Object Assets

The account ownership of the extensible smart object assets model has a fortuitous side effect with respect to scalability. The transfer of ownership of a smart object reference asset between two or more accounts is a local matter between the accounts. That is, if multiple smart object reference assets are being transferred between unrelated disjoint accounts, all of the transfers may happen simultaneously on separate blockchain shards. For example, if account A is transferring to account B, and account C is transferring to account D, both transfers may happen simultaneously on separate shards.

In comparison, a smart contract account must process transactions serially, which implies the state transition must take place on a main blockchain or a single shard. Therefore, in the above example only one of the two transactions can take place at a given time. Sharding does not provide any scalability in the smart contract situation.

Summary

The Extensible Smart Object Asset model introduces the concept of ownership of arbitrary objects to the blockchain and cryptocurrency model. Since an XSOA can represent virtually anything and be traded with any account, many blockchain applications that are currently implemented as separate smart contracts and individual tokens may be far more naturally implemented as XSOAs. Using classes defined with the XBOM, transactions for all XSOA's may be handled uniformly. This extends naturally to wallets containing smart object assets as well as buying and selling of assets. The above diagrams and descriptions are not meant to be exhaustive. The XSOA model can be readily extended to support arbitrarily complex relationships between accounts and types of assets.

SagaCoin ($PSC) Financial Model for Incentivizing

There are two opposing goals for the value of the SagaCoin ($PSC):

The first is the desire to create a usable currency. For this, it needs to be stable and related to externalities, such as national economies reflected in currency exchange rates. If the $PSC is stable, it can see adoption for use as a means of commerce for a decentralized global economy.

The second is the desire for the $PSC to gain in value against other currencies so we can exchange some of it (i.e. USD or EUR) during an initial phase. That is, the $PSC should have a lower inflation rate than other currencies during an initial phase, and a negative internal price inflation rate.

To accomplish either of these goals a means of currency supply management of the $PSC is needed.

Existing Cryptocurrency Supply Management

Existing cryptocurrency supply management approaches consist of three solutions:

Incentive rewards for performing the block mining algorithm (i.e. proof-of-work). The amount of reward per block received diminishes over time, eventually reducing to zero. This is the Bitcoin model. Eventually, all of the Bitcoin tokens[1] that will ever exist will be mined. The token supply is fixed in the long term.

Similar to the Bitcoin model, incentive rewards for performing the block mining algorithm, except the amount of reward per block is fixed and never diminishes. In this case, the token supply increases forever. However, the percentage of increase per incentive reward as related to the total token continuously decays.

A token generation event (TGE), which creates a fixed supply of tokens. These tokens are then distributed to accounts via some mechanism (i.e. “airdrops). Although theoretically there can be multiple TGE’s for a given token, in general such designs, envision a single TGE, or at most predetermined periodic TGE’s. In summary 2 out of the 3 solutions are essentially a fixed supply of tokens, and the 3rd ‘s rate of change of the token supply drops to negligible amounts over time, thus effectively making it also a fixed supply. In short, currency supply management does not exist in the blockchain and cryptocurrency implementations.

Volume of Token Exchange

The measurement of volume of exchange of a specific token can be found on crypto exchange listing such as CoinMarketCap. This measurement shows the relationship between the token and other currencies, usually the USD. A higher volume indicates a higher demand for the token. Since the supply of any specific token is essentially fixed, as described above, the price of the token marked to other currencies increases. That is, the token itself becomes a rare commodity, resulting in large price fluctuations. Although this satisfies the second goal above, that of the increasing value of the token during an initial phase, it doesn’t satisfy the first and primary goal, a usable currency.

Digital GDP

The measurement of exchange pricing of a token with other currencies, as described above does not measure any pricing of that token used directly as a currency. If we

consider a token on a blockchain used directly for instances of commerce either B2B, B2C or any other form, the token and the associated blockchain can be thought of as forming their own economy. The measurement of this economy can be thought of in similar terms to any economy as having a gross domestic product (GDP). To distinguish this from a national, regional or other geographically defined economies, we are introducing the term "Digital GDP". Thus, the more commerce, the more things traded directly on the token's blockchain, the larger the digital GDP of that blockchain.

Digital GDP and Token Value with Fixed Token Supply

The demand for a token as determined by the volume of currency exchange on an exchange listing does not take into account demand for that token within the economy of its blockchain. As the digital GDP grows within that token's economy, the demand for the token will increase. This has the effect of reducing the amount of token available for currency exchange since the token supply is fixed as described above, and in increasing use within the token's blockchain economy. This simultaneously has the effect of internal price deflation within the token's blockchain economy, while increasing the exchange rate pricing. The most notorious example of this is the "$800 million dollar pizza".[2]

Thus, with a fixed token supply, the internal price deflates continuously and becomes more valuable externally making it unattractive to use as a currency of exchange. Instead, it becomes more of a store of value. Mining dominates over use as a currency, which in the short term is very attractive and lucrative for the miners. But in the longer term, and in the extreme with dropping usage, a collapse can occur. When a large majority of the tokens are being held and not used for commerce on the blockchain, and the volume of exchange starts to drop off, hyperinflation sets in, making the coin worthless to exchange.

Digital GDP and Token Supply Management

To create a stable token with respect both to internal pricing within its digital GDP and to external exchange pricing, the supply of the token must be managed. Thus, using any of the three fixed token supply models described above cannot result in a stable usable token for general commerce. Although this may be good for speculators and the initial blockchain developers, it doesn't satisfy the first primary goal for the $PSC.

In general, a well-managed economy[3] needs a money supply management approach that controls the money supply in a manner that reflects the change in the size of the economy. As the economy is measured in GDP (nominal and real), the money supply, as measured in supply and volume, must reflect such changes. If this is accomplished then pricing, such as measured by the CPI, remains stable, and the primary goal of a stable currency is achieved. Therefore, a decision to inflate the money supply would reflect an increasing GDP, and conversely, a decision to deflate the money supply would reflect a decreasing GDP. Since it is hard to track true GDP growth exactly in national economies, the usual objective is to have moderate price inflation instead. The theory is that this essentially keeps the money supply growing in line with a generally growing GDP. This makes the value of the money stable in the long run.

Relating this to the $PSC in SagaChain economy, to realize a stable token usable for general commerce, the supply management

of the $PSC must reflect the digital GDP on SagaChain blockchain. If this is accomplished, the phenomenon of the "$800 million pizza" is eliminated creating both internal stable pricing, and external exchange rates. This accomplishes the primary goal above.

As described above, the two goals are in opposition to each other. Since it is desirable that there is some continuous growth in store of value for the $PSC, $PSC supply management, tracking the internal economy to eliminate price inflation and deflation means there would be some minimal increase in the $PSC value with respect to currency exchange rates. This would eliminate the accumulation of value in the $PSC as an investment opportunity, while supporting a growing economy

What is required is a high initial growth of the $PSC value, that slows down eventually to match digital GDP growth of the internal economy, and eventually targets zero internal price inflation. The initial growth satisfies the secondary goal making the $PSC an investable entity, while the target of zero internal price inflation satisfies the primary goal of a cryptocurrency usable for general commerce.

Given that certain national economies, such as the US, target continuous price inflation as a policy, an eventual zero internal price inflation for the $PSC would enable an exchange rate that increases slowly in value against other economies, (USD and EUR in particular). Stated another way, the $PSC supply growth needs to reflect the growth of the internal economy as time goes on, instead of a fixed supply (either as a TGE or overtime). Most people in the crypto world will not accept this concept currently, given the influence of the current success of Bitcoin as valued on crypto exchanges, focusing exclusively on deflation and store-of-value.

Managing the SagaCoin ($PSC) Supply for Value Accumulation and Currency Usability

So, the question and challenge are how to come up with a mix that satisfies short term ROI for the early investors and early adopters, while making sure in the long run that the $PSC stabilizes and becomes a general, usable, currency.

The answer to this, is that the money supply inflation initially must be less than the digital GDP growth, which will cause $PSC deflation and thus accumulation in value, then allow for price stabilization, and adjust money supply against digital GDP resulting in longer term $PSC price stability and usability as a general currency.

This poses the question of, how to come up with a mix that satisfies short term value accumulation and thus an ROI for the early investors and early adopters, while making sure in the long run that the $PSC stabilizes.

The way to satisfy this dilemma, is to combine the experience of both the cryptocurrency models and the national economy models. The cryptocurrency models, typified by the Bitcoin model for initial coin incentive rewards give us the deflationary model and the value accumulation and ROI desired, but add a second term in $PSC supply management equation that inflates and/or deflates for digital GDP which initially has no weighting in the supply management decision, but becomes heavier weighted over time, eventually completely dominating the supply management decision.

External GDP

There are two fundamental GDP questions: what is digital GDP; and how do we measure the GDP of the rest of the world economies. Note: The assumption is that fiat currency economies remain dominant and do not go away (e.g. USD and EUR).

For external GDP, several indices are used, meaning GDP measurements, excluding the likes of the DOW. Both a real GDP and a nominal GDP measurement of world economies needs to be taken as these types of statistics are readily available.

Note that since the accuracy and availability of such measurements may change over time, the $PSC supply management algorithms support changing and replacing such sources of statistics, under governance.

Digital GDP Measurement

Measuring GDP in a sector can be reduced to taking total sales, dividing by average unit price, and adjusting for inflation against a baseline. However, how does one measure total sales and units on the blockchain?

A blockchain such as Bitcoin does not directly have any concept of sale of units of anything, which makes the concept of internal digital GDP unmeasurable. Bitcoin is sort of like having a bank account ledger where you can see the debits and credits but you can't see what any of them were for.

Smart contracts as implemented on blockchains such as Ethereum, create a separate token for each smart contract. Although it may be possible to categorize each smart contract into a market sector and determine the number of units of an entity that each token represents, then use exchange listings to come up with a measurement of internal GDP, this does not lend itself to a general token supply management solution. Further, such smart contracts do not have any common structure between them which makes any such evaluation extremely difficult.

If there were a means for internal digital GDP measurement, then there would be a means to look at the P*Y side of the monetary equation for $PSC supply management. If not, one can only go with an arbitrary money supply inflation model, just like any other cryptocurrency. That would mean the primary goal above is likely never to be fulfilled.

Prasaga proposes a solution that the initial incentive model is a decaying model, similar to Bitcoin, but coupled with a digital GDP inflation/deflation tracking model that dominates in the long run. This function is written into SagaChain using SagaChain smart object model. The target objective is that the deflationary incentive model dies off after a period of 10 years instead of the longer Bitcoin model. It is believed that the digital GDP concept is a critical missing component of crypto currency in general.

Smart Objects Supporting Digital GDP Measurement

Smart object classes that enable measuring digital GDP can easily be designed. They would include fields that specify units, price, and market segment at a minimum. As all transactions are recorded on SagaChain and are priced in $PSC, a digital GDP measurement may be derived directly.

Some Assumptions for Digital GDP Enabled Smart Objects:

Allow for a mix of smart objects classes that provide measurements and those that are arbitrary transfers of $PSC.

Define “one-shot” exchanges which are often wallet- to-wallet, versus multi-payment which are usually instances of commerce of some kind.

Use the smart object class concept to create “buckets” of types of commerce similar to sectors in economies.

Smart objects that do not follow any models are considered in their own sector type. These can be distinguished as one shot and multipayment only. They are prorated for the digital GDP based on their volume in units where a unit equals the one-shot, or a transaction for the multipayment time's average price, against total SagaChain volume. Their contribution to digital GDP is then weighted by this. These are essentially then “virtual units” and are figured in to the total digital GDP.

By coupling these approaches with the first type of smart object classes, specifically for the IoT data markets, as an example, and by using the XBOM smart object model, any Dapp designer can “inherit” the means for recording the metrics for digital GDP by using the smart object classes in the foundation classes of SagaChain .

Given the significant improvement in ease of use with the XBOM smart objects, and the expected Dapps that depend on them, it is expected that they will become part of standard usage and thus digital GDP measurement will become increasingly more accurate as of the usage of SagaChain increases.

Decentralized $PSC Supply Management with Digital GDP Metrics

The Digital GDP metrics derived from SagaChain are calculated locally by each node given the current consensus state of the blockchain. The metrics are input to the $PSC supply management function and result in changes in the rate of increase or decrease of the $PSC using the incentive reward and token burning mechanisms implemented in SagaChain monetary policy.

Creating Digital GDP Metrics for Non-Conforming Smart Object Classes

As described above, there are three sectors: one-shot; multi-payment; and wallet-to-wallet. To determine their digital GDP, the number of transactions of each type are summed and divided by the total transactions, to obtain their weighting. This then gives the average unit price as a total for each, divided by the transaction count. Then the comparison of the change in transaction count against a baseline developed historically is applied. A positive change times average unit price as a weighted is considered an increase in GDP, produces a usage metric.

Wallet-to-wallet transfers imply token utility, but the utility is unknown. If all basic ledger transfers of $PSC are considered a form of general commerce, then wallet-to-wallet falls under general commerce. If the global average of “unit price of goods”, is taken as a measurement for general commerce measurements, then the unit price of goods would be taken across all sectors defined by all smart object classes, which gives another metric.

The implication is that something of value must have changed hands for the instances of general commerce as well. This works reasonably well if the conforming smart object classes dominate the transactions overall. Or, more specifically if, the smart object classes track the average unit price of goods reasonably well. The assumption is made that this will be the case based both on

ease-of-use of the smart object classes, and a Schelling point hypothesis.

Digital GDP Metrics and Conforming Smart Object Classes

The XBOM foundation classes implemented on SagaChain shall include the categories of commerce as currently defined by the US, Europe or by other governments globally. Using these as a basis creates a means to establish a common set of metrics that can be understood both with the internal SagaChain economy and external economies. As the $PSC token supply management initially is using a deflationary decaying incentives reward model, that essentially ignores all GDP metrics during the initial phase, the initial phase will be used to establish baselines for tracking.

The foundation will create classes and objects that include all current categories of commerce defined by the US, Europe (e.g G20) with amending capabilities for expansion. That way there is some means to try to establish a common set of metrics. It is essential that initially, a decaying incentivesmodel is used. This will allow tracking to commence before it is actually used in controlling the economy. This allows for the establishment of long-term money supply goals used as the token management supply transitions from $PSC value accumulation to price stability.

External Economies and "Oracles"

Although the Digital GDP metrics can be determined locally and independently by all SagaChain nodes, and thus are decentralized by design, the metrics of national economies are centralized and reported by often a single source. By definition, such sources are reported as oracles with respect to the SagaChain. The smart objects that implement the token supply management on SagaChain shall initially define the oracle sources for all such external reporting. These smart objects include methods to replace the oracle sources if and when such a need arises. A voting body shall be established consisting of stakeholders in the SagaChain, defined as owning sufficient $PSC, and/or certification of authentic identification for the purposes of controlling changes to all parameters to the $PSC supply management implementation, as the form of governance. The XBOM Smart Objects can enable complete security against any unauthorized changes through the use of threshold multiple signature designs.

Governance XBOM Objects of Operational Smart Assets

Overview

Smart contracts, as implemented currently in the blockchain community follow the model that "code is law". Although this sounds attractive, in practice, any single mistake in a smart contract can be catastrophic. Further, SagaChain uses smart assets internally to manage critical aspects of its operation. Therefore, SagaChain has added a new concept Smart Asset, governance. A governed smart asset includes various abilities defined below. The governance aspect is implemented as a committee membership of voters. Governance enables control and modification of smart assets. Governed smart assets are an optional capability that complement the "code-is-law" model.

Governance Capabilities in Abstract

The following abstract capabilities needed for governance:

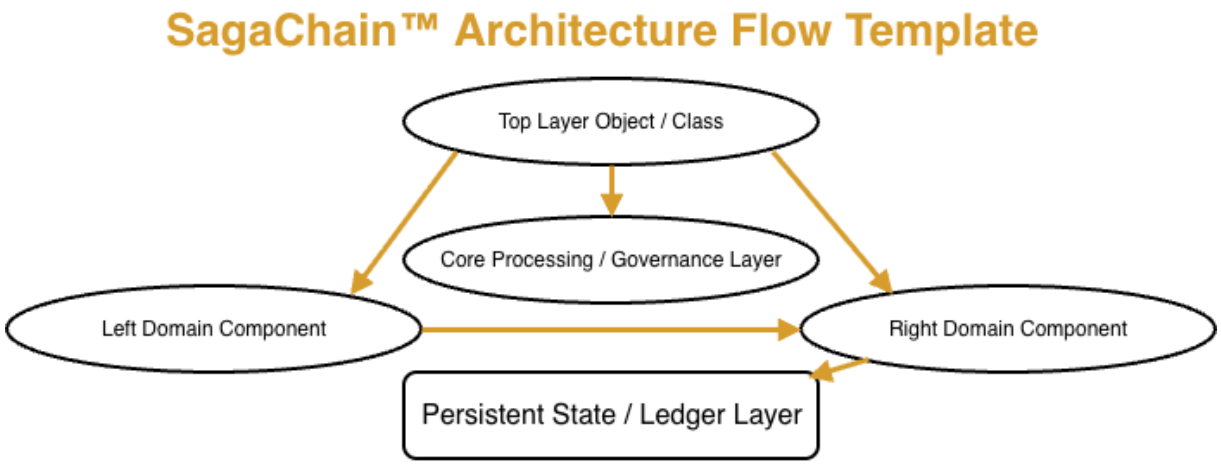
- Means for describing who can vote, adding/deleting

- Means for bringing something up for vote
- Means for voting
- Means for implementing the result of the vote
- Means for verifying the implementation

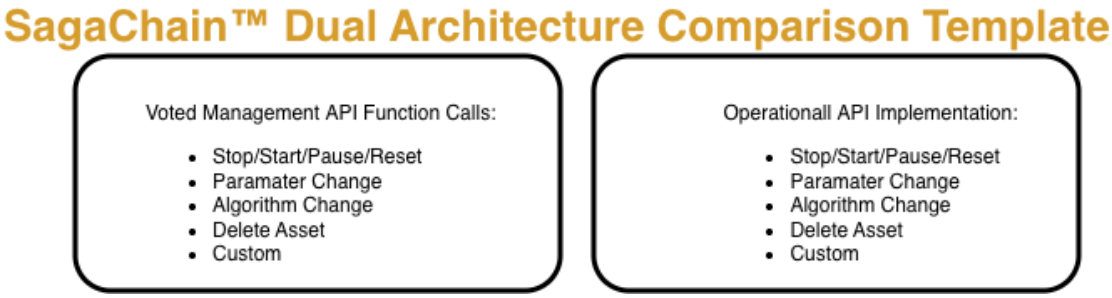
Abstract operational capabilities of voting governance:

- Control a smart asset (start/stop/pause/delete)
- Modify parameters to a smart asset
- Smart asset algorithmic sections replacement

The following sections describe how governance of governed smart assets is implemented abstractly. The following diagrams depict the relationships between the abstract components of the governance architecture:



Governed Operational Smart Asset Architecture



Governance Voting

Any voter that is registered for the given governance committee and is authorized based on any staking or other requirements can submit a change request to the governed

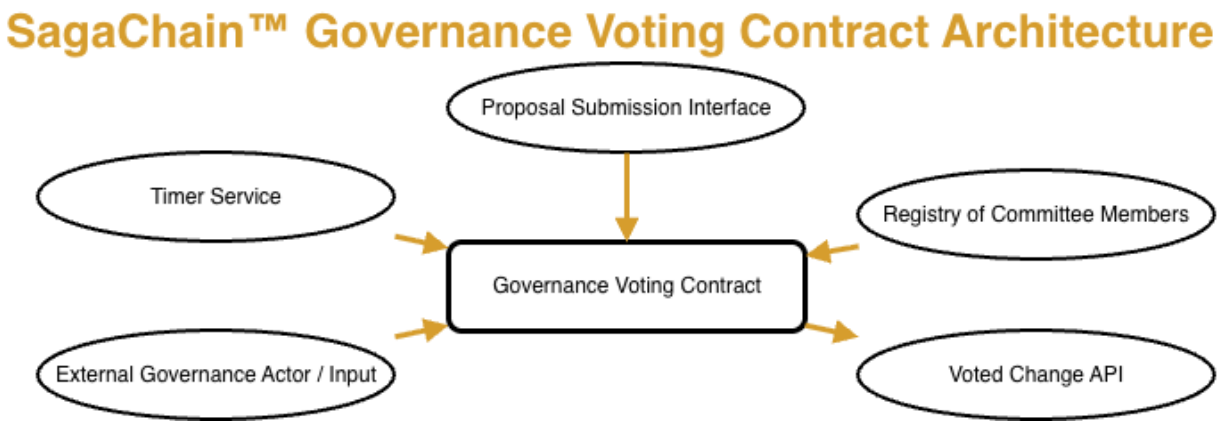
operational smart asset. A change request is a piece of executable code. The code, with a text is submitted to the governance committee. Upon a successful vote of committee AND majority and/or super majority community vote as required by governance mandate, the code is executed against the operational smart asset. If the vote is unsuccessful the change request code is abandoned. The conceptual steps for a change request are the following:

- A committee member proposes a change request
  - consisting of executable code
  - and text description
  - sent as a transaction to the governance asset
- Governance asset verifies the committee members
  - committee member must be registered
  - a committee member may only have one outstanding change request
- If there is an ongoing vote, the change request is queued in a FIFO

The next change request is sent to all committee members, using the means designated by each member's registration:

  - email
  - text
  - etc.
  - via outbound smart asset calls
    - Or via a blockchain scanning model

- The voting duration timer is sent to the timer service
- If the vote is successful (majority, supermajority or unanimous) then
  - The change request is sent to community vote
  - If plenary vote is successful (majority, supermajority or unanimous) then
  - the change request code is executed using the APIs on the operational smart asset
- else if the vote is unsuccessful then
  - the change request is abandoned
- else if the time expires before sufficient votes are collected then
  - the change request is abandoned, and late votes are ignored
- The FIFO queue is checked for the next vote. If present, then start the next vote.
- Otherwise exit.



Governance Voting Management

There is no central control or leader of the

governance asset. Any committee member may propose a vote at any time. Instances of the governance asset may impose various limits such as the number of votes a member may propose per unit time, how long a vote may last, and similar.

Governance Asset Management

A governance asset is considered immutable once instantiated. No modification to the governance asset code is allowed. A governance asset may optionally allow itself to be voted out of existence by its governance committee and community. In such cases, the operational smart asset associated with the governance asset shall be terminated as well.

Note: Built-in assets to SagaChain are immutable and non-cancellable.

Change Request Code

Change request code consists of two main categories: changes to parameters; and changes to algorithms.

Parameter changes are enabled through simple parameter setter functions. Change request code to change a parameter is of the form (pseudo-code):

parameter = <value> |
<value> (operation) <old parameter value>
Parameter changes may be grouped together, voted on, and executed as a single transaction.

Algorithmic changes enable changing functional models of an operational smart asset. An asset that supports such modifications must have one or more functions that are called using indirection. Change request code to change a function is of the form (pseudo-code):

- <Function> = <New Function (parameters)>
- <New Function> {implementation}

The new function parameter signature must match the old function signature exactly.
Multiple function changes may be grouped together, voted on, and executed as a single transaction.

Parameter changes and Function changes may be grouped together, voted on, and executed as a single transaction.

The governance asset verifies that each change request is syntactically and request is syntactically and semantically correct with respect to parameter names, defined value ranges, and function signatures, prior to proposing a vote on each such change request.

Governance & Financial Models
Governance

The governance refers to any actions carried out on the network that change the rules of the decentralized system. The governance model from Prasaga takes inspiration from modern democratic and monetary policy systems. Recommendation Boards will be comprised of Subject Matter Experts from a range of communities including developers, cryptographers, economists, legal experts and other domain experts who specialize and contribute to the network development.

To democratize the process, key policy and ultimately implementation decisions will be based on a majority or super majority plenary vote from SagaChain community. To balance the voting, the weight of each individual will not be determined by how many validator nodes or tokens held, to lessen the self-interested impact on the governance decision- making process, we envision a one

person one vote system which is further enabled by the XBOM account and smart capabilities.

There are four proposed vertical areas of governance:

- Technology
- Monetary
- Treasury
- Community

Each Recommendation Board will comprise of a panel of experts, at first appointed by the foundation and tasked with day-to-day responsibilities to engage with Prasaga ecosystem and outside experts. The boards will propose recommendations to SagaChain Community for final vote before changes or new features for the ecosystem are implemented.

An outline of responsibilities for each Board is noted below, we fully expect there will be sub-committees that report up to the main boards:

Technology Board: work related to building and releasing updates of SagaChain , and other technical issues;

Monetary Board: work related to monetary policy around token supply and treasury, and other economic and commercial governance issues;

Treasury Board: work related to managing the budgets for the Prasaga Foundation, the Prasaga Philanthropic spending

Community Board: work relating to marketing and promoting SagaChain globally, and the appointment and management of ambassadors for outreach and education.

Prasaga Foundation, as the originator of the network, will help initiate and coordinate the Boards and sub-committees in order to get them established and self-sufficient. It is intended that the Boards will as soon as practically possible be independent from Prasaga and comprised of a majority of non-Prasaga members. Any member of the community can apply to join an expert Board and/or Sub-Committee, there will need to be a vetting process to ensure best representation for the community in their ecosystem of experts and final plenary vote. Each board shall include seven members with a term of 5-7 yrs.

There are four core financial models in the SagaChain. These are the following:

- Monetary Policy Model
- Treasury Model
- Gas Price Model
- Node Staking Model

Financial Models

The Monetary Policy Model manages the supply of the SagaCoin ($PSC), changing the rate of increase or decrease based on economic information.

The Treasury Model manages the budgets for the Prasaga Foundation, the Prasaga Philanthropic spending, the wealth redistribution function, and a short-term surplus.

The Gas Price Model manages the average gas price with respect to the total SagaChain economy, to maintain profitability for node operators and enable competitive pricing for differentiated service.

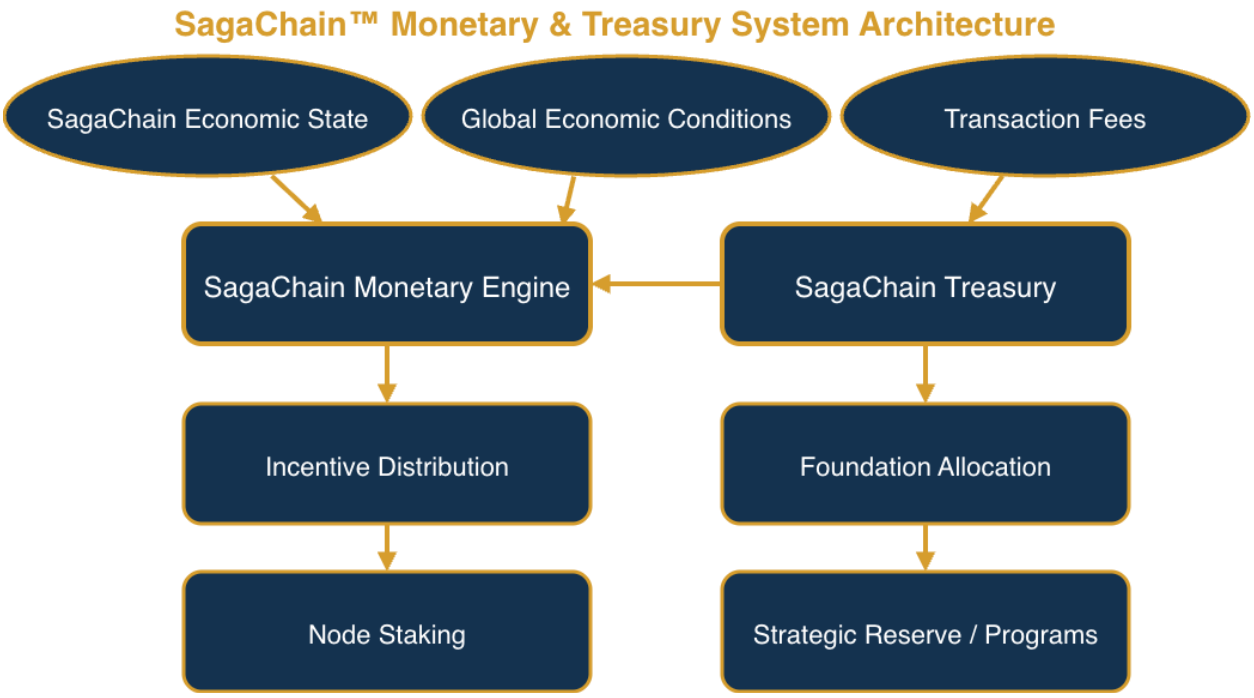
The Node Staking Model manages the node $PSC staking requirement for (re)registering a node for the SagaChain. The objective for

node staking is to discourage various forms of attacks by attaching monetary costs to such attacks, while balancing the staking amount with encouraging a large diverse pool of node operators.

Combined, these four models manage the main aspects of the internal financial aspects of the SagaChain.
Monetary Policy and Treasury Models

The Monetary Policy Model and Treasury Models are independent of each other and are controlled by independent governance bodies. However, they have both direct and indirect relationships. Therefore, these are discussed together. The diagram below shows the direct conceptual relationship between these two models.
Both models are discussed below:

Monetary Policy Management and Treasury Management Relationship



Monetary Policy Model

The responsibility of the Monetary Policy Model is to manage the supply of $PSC coins in the total SagaChain economy. This is equivalent in concept to decentralized money supply management. To accomplish this there must be a means to increase the supply of $PSC as well as decrease the supply of $PSC.

Increasing the $PSC supply is accomplished through block incentives. As such, the Monetary Policy Model affects the increase of the $PSC supply by managing the amount of $PSC per-block incentive.

Decreasing the $PSC supply by definition means burning $PSC. Psychologically burning coin is difficult, even though theoretically it increases the value of the remaining coins in circulation. That is, a node operator on validating a block expects to earn transaction fees and possibly an incentive

reward. Burning some of the transaction fee and eliminating the incentive reward, if needed to manage the $PSC supply is unlikely to be a welcome state.
Therefore, the Monetary Policy Model uses a novel approach. An optional, variable $PSC bonus is given to each node as it (re)registers on the SagaChain. This bonus, just like the stake, earnings and incentive rewards are not received until the node completes a lifecycle. The amount of the bonus may vary and may be zero, based on the Monetary Policy Model's determination of the target $PSC supply. Any bonus that is not delivered to the node is burned.

Burning the $PSC instead of delivering it as a bonus is not sufficient on its own. The critical missing piece is the source of the $PSC for the bonus. The bonus $PSC is supplied from the Treasury account, which in turn is collected from the transaction fees taxes. As a result, burning some or all of the bonus directly reduces the $PSC supply. Because the bonus only impacts the node registration stage, when $PSC supply reduction, or reduction in the rate of increase is needed, the Monetary Policy Model reduces bonus returned to the node through the Monetary Policy smart contract, the nodes already registered on SagaChain do not feel the impact directly.

Equation of Exchange

The fundamental equation is M*V = P * Y, where

M = money supply
V = velocity of money P = price of goods
Y = real GDP

The money supply, M, for SagaChain is the $PSC coin supply. Unlike a fiat currency, all of the $PSC in all the accounts can be accounted for. However, the use of the account cannot be determined (e.g. savings versus checking in a traditional bank account). Thus, determining the effective money supply is difficult.

GDP, Y, for SagaChain is termed “digital GDP”. There are several means to measure the digital GDP. One of the best means is the Commerce classes. It shall be defined to include a unit volume per transaction and a price per unit. Subclasses of the Commerce class shall include market segment information. For non-commerce class ledger transactions, an average transaction size for a period can be measured and the rate of change of size of the transactions and change of volume across periods can be used to estimate change in digital GDP.

Price, P, is defined as the average price of the average unit. For the SagaChain, P*Y can be calculated directly.

Velocity, V, measures the velocity of money, $PSC. Velocity is measured simply as the number of $PSC that were spent per period.

The objective of the Monetary Policy Model is to manage the $PSC supply in the short term with respect to the velocity, and in the long term with respect to nominal digital GDP.

For short term management, the nominal digital GDP is considered constant. Thus, an increase in velocity can result in an increase in demand for $PSC. Therefore, the $PSC supply is incrementally increased by increasing the rate of incentives or depending on the current state a decreasing in the burning rate.

For long term management, the velocity is considered constant, and the nominal digital GDP growth is considered. The $PSC supply is increased (decreased) based on the

projected digital GDP growth. To manage SagaChain digital GDP versus external economy's GDP, the ratio of rate of projected growth is used to target a price growth rate moderately below the external economy.

The short-term adjustment period occurs every NN days defined in blocks.

The long-term adjustment period occurs quarterly lagging the external economy published values.

Treasury Model

The Treasury Model uses the following equation:

XX% * Transaction fees = Foundation Budget + Wealth Redistribution Budget + Philanthropic Budget + Surplus Budget.
XX% is the percent of the transaction fee for each block that is contributed to the Treasury (i.e. tax).

Transaction fees are the sum of all transaction fees for a given period.

Foundation Budget is the contribution to maintaining the Foundation operations.

Wealth Redistribution Budget is the $PSC available for node (re)registration bonus or burning.

Philanthropic Budget is available for projects determined by the Philanthropic Governance.

Surplus Budget is intended to provide a buffer for short term fluctuations to avoid shortfalls in the other budgets.

The Treasury Model including the budgets and the XX% are determined by Treasury Governance.

Gas Price Model

The Gas Price Model uses the following equation:

Average Transaction Gas Price = (P*Y * GG%) / transaction count.

SagaChain average transaction throughput gas price is set periodically using the above formula. Essentially the intent is that the $PSC spent on gas is on average GG% of the economy.
GG% is managed by Gas Price Governance.

Node Staking Model

The Node Staking Model uses the following equation:

Node Registration Stake = (P*Y * SS%)/node count

The node registration stake price is set periodically using the above formula.
The intent of the node registration stake is to discourage a majority attack by setting the per-node stake such that a 50+% attack (P*Y * SS%/2) is a large enough financial commitment to deter such an attack.

SS% is managed by Node Staking Governance.

Prasaga Foundation

Note that changes to the parameters of the $PSC supply management functions do not impact the functionality, but it does allow the governance body to perhaps manipulate the money supply by manipulating the source oracles for external economy metrics.

Thus, the value of the $PSC could become hostage to such a governance body. If confidence is lost in the governance body,

then a malicious governance body might change the GDP measurements, causing either hyperinflation or stagnation. It is not viable to mandate that the Prasaga Foundation always controls this. That would substitute centralization in the Foundation creating the concern that the Foundation would engage in similar manipulations.

Therefore, the Prasaga Foundation shall perform at least the following functions:

1: Running a root chain permanently. This is a continuity requirement to protect against catastrophic global network failures.
2: Running full backup nodes — these may be used for downloading by new joining nodes optionally
3: supporting SagaChain and XBOM source code and the Foundation Institute
For consideration, the Foundation could include a charter to also provide structure for a $PSC supply management governance body.

Conclusion

Prasaga has designed the next evolution of blockchain architecture. A blockchain that actually achieves the promise that all blockchains have aspired to. One that provides the highest level of resistance against attacks. One that rewards its ecosystem of contributors. And, one that, perhaps most importantly, scales in throughput as more resources are added to the network, providing the maximum possible increase in speed.

SagaChain catalyzed by our eXtensible Blockchain Object Model (XBOM) builds a ledger that puts the coding, execution and settlement of all asset transactions directly onto the blockchain — delivering an evolved approach to Smart Contracts that provides increased speed of development, higher quality, and a future proofed development infrastructure.

The D-PoW, XBOM and $PSC combined, enable a global, scalable, currency stabilized, independent, decentralized blockchain.
Current status:

The D-PoW and related algorithms, the XBOM, and the $PSC financial are the subject of a series of pending patents.
A proof-of-concept implementation of the XBOM realized on top of the Hyperledger Fabric blockchain platform for non-cryptocurrency enterprise applications is available in an MVP form. A set of foundational classes are provided in source code format.
The Product suite of Prasaga creates an underlying global foundation for all transactions financial and other while protecting individual sovereignty and ownership/access of all assets.
The source code and development environment is available at xbom.io for experimentation and testing.