ProximaX®

Sirius

Public Platform

Whitepaper

Version History

Version:	**Date:**
1.0	September 2019
2.0	November 2021

Contents Page

1. Sirius Platform

Blockchain development and infrastructure platforms are redrawing the structure and back-end of many industries as we know them today. However, these platforms usually consist of just a blockchain used principally for value transactions. ProximaX is changing this by significantly broadening the utility of blockchain to also cater to data transactions.

ProximaX Sirius is a platform that extends beyond traditional blockchain protocols by integrating blockchain with distributed and decentralized storage, streaming, contract, and database[1] service layers available in private, public, hybrid, and consortia network configurations.

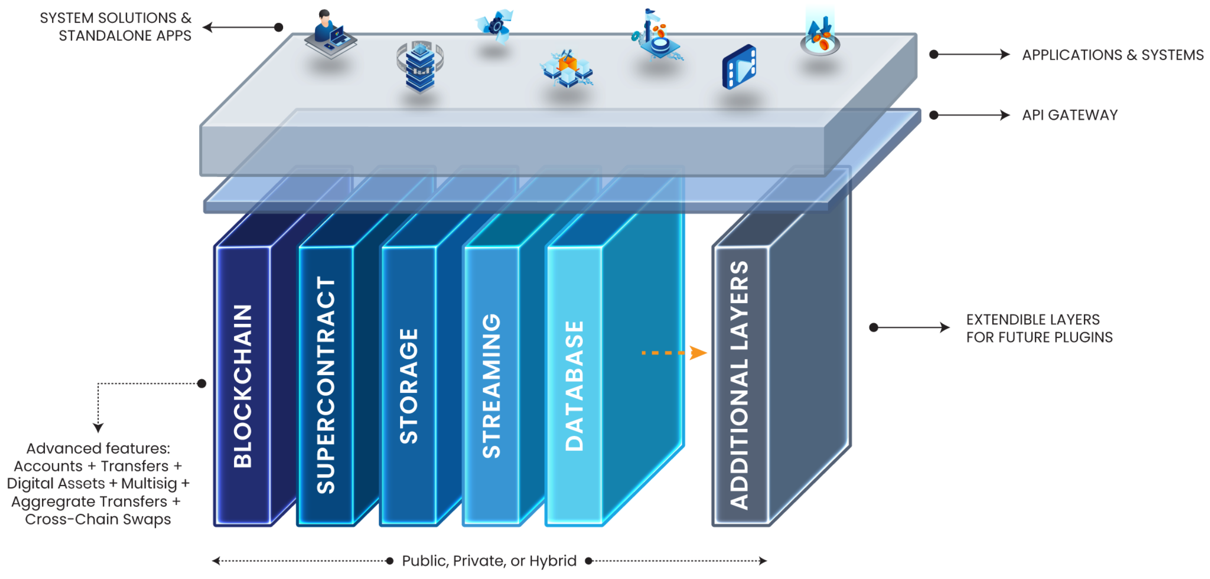

SYSTEM SOLUTIONS & STANDALONE APPS
APPLICATIONS & SYSTEMS
API GATEWAY
BLOCKCHAIN
SUPERCONTRACT
STORAGE
STREAMING
DATABASE
ADDITIONAL LAYERS
EXTENDIBLE LAYERS FOR FUTURE PLUGINS
Advanced features: Accounts + Transfers + Digital Assets + Multisig + Aggregrate Transfers + Cross-Chain Swaps
Public, Private, or Hybrid

Figure 1: ProximaX Sirius platform technology stack.

The platform has multiple servers distributed in a network that follows a "hub-and-spoke" design where the essential component is the blockchain, representing the "hub," and all other service layers represent the "spokes" held together by the blockchain. This design enables scalability as it facilitates the addition of more core services in the future without compromising the platform's performance.

Each layer has its ecosystem of node actors, providing unique services wrapped in an accessible application programming interface (API) and software development kits (SDKs) available in multiple common coding languages.[2]

[1] Available for private networks only.
[2] Available SDKs: Go; C++; Java; JavaScript; TypeScript; C#; Swift; Python; Dart; JavaScript CLI; PHP; Rust.

The superior utility and flexibility of ProximaX Sirius enable developers to build any solution vertically on top, including applications for banking and payments, digital identity and KYC, video streaming and chat, and gaming.

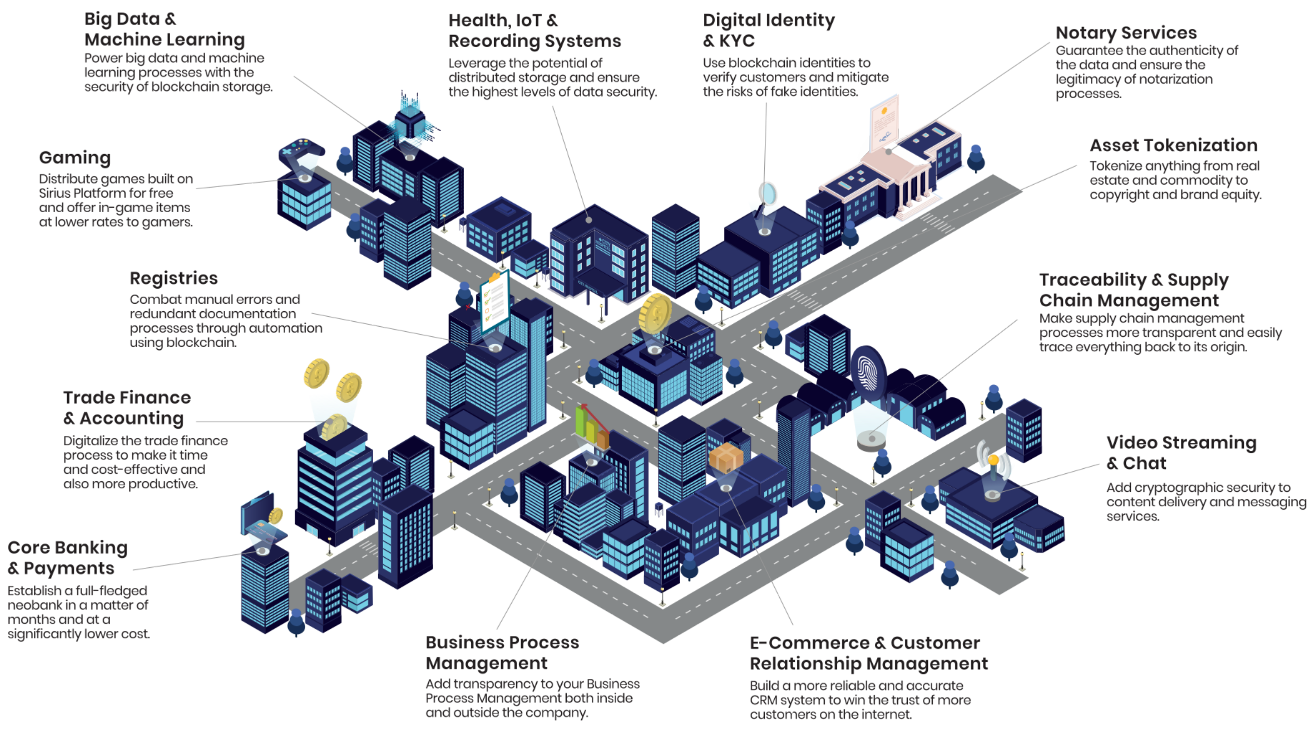

Big Data & Machine Learning
Power big data and machine learning processes with the security of blockchain storage.
Health, IoT & Recording Systems
Leverage the potential of distributed storage and ensure the highest levels of data security.
Digital Identity & KYC
Use blockchain identities to verify customers and mitigate the risks of fake identities.
Notary Services
Guarantee the authenticity of the data and ensure the legitimacy of notarization processes.
Gaming
Distribute games built on Sirius Platform for free and offer in-game items at lower rates to gamers.
Asset Tokenization
Tokenize anything from real estate and commodity to copyright and brand equity.
Registries
Combat manual errors and redundant documentation processes through automation using blockchain.
Traceability & Supply Chain Management
Make supply chain management processes more transparent and easily trace everything back to its origin.
Trade Finance & Accounting
Digitalize the trade finance process to make it time and cost-effective and also more productive.
Video Streaming & Chat
Add cryptographic security to content delivery and messaging services.
Core Banking & Payments
Establish a full-fledged neobank in a matter of months and at a significantly lower cost.
Business Process Management
Add transparency to your Business Process Management both inside and outside the company.
E-Commerce & Customer Relationship Management
Build a more reliable and accurate CRM system to win the trust of more customers on the internet.

Figure 2: Use-cases.

<u>Core service layers:</u>

i. Sirius Chain

The blockchain layer, the Sirius Chain, is a fork of a new open-source blockchain called Catapult. ProximaX selected Catapult because of its unique enterprise-grade features (see section 3.2.) not available in other blockchains. ProximaX augmented Catapult's capabilities by integrating new protocols so that Sirius Chain can perform the "heavy-lifting" required of a "hub."

ii. Sirius Storage

The storage layer, the Sirius Storage, enables public contributors to monetize their unused storage space. Features include decentralized Drive creation, data upload, data modification, and data download. Protocols enable collective decision-making for payments, verification, synchronization, and anonymous downloading.

iii. Sirius Stream

The streaming layer, the Sirius Stream, enables public contributors to monetize their unused internet bandwidth for live streaming. Features include decentralized live stream creation and transmission. Protocols enable sponsorship and mass content distribution.

iv. Supercontract

The contract layer, named Supercontract, is an improved version of traditional smart contracts, where self-executing logic is located off-chain in Sirius Storage to prevent the blockchain network from becoming "bloated" and affecting its performance. Dedicated nodes execute a Supercontract rather than all network nodes, reducing computational work. Unlike traditional smart contracts, authorized parties can stop, amend, and restart a Supercontract quickly and easily upon reaching a consensus.

v. Content Review

Content Review is an additional layer that enables network content to be categorized, searchable, censored, and banned.

2. Tokenomics

2.1. Tokens

As with all decentralized public networks, tokenomics design and implementation are crucial for their sustainability and governance. The purpose of tokenomics is to create common economic incentive frameworks for “work done” by network contributors, tied together with a reputation system.

The basic principles adopted in ProximaX’s tokenomics design include:

- Self-organization and synergy between multiple system elements.
- Adaptability to adverse tactics and internal attacks.
- The creation of Service Units used to quantify the provision of the platform’s core services.
- Extendible infrastructure to facilitate plug-ins for new core services and tokenomics.
- Creation of ecosystem solutions and applications that will leverage the blockchain to create more transactions, with XPX as the core economy driver.

ProximaX Sirius’s token ecosystem consists of the following:

Internal:

- **Native token (XPX):** Powers the blockchain layer and is used to pay for platform services.
- **Service Units:** Quantifiable units of measure for the provision of platform services.
- **Sirius Digital Assets[3] (SDAs):** Any user can create new types of SDAs, whether to power the internal economy of a decentralized application (DApp), represent digital security (security token), or an object which is fungible or non-fungible, such as a non-fungible token (NFT).

External:

- **Fiat and other cryptocurrencies:** Programmers can integrate any payment gateway, e.g., swapping XPX or SDAs into other networks and back.

[3] An SDA was previously known as a Mosaic.

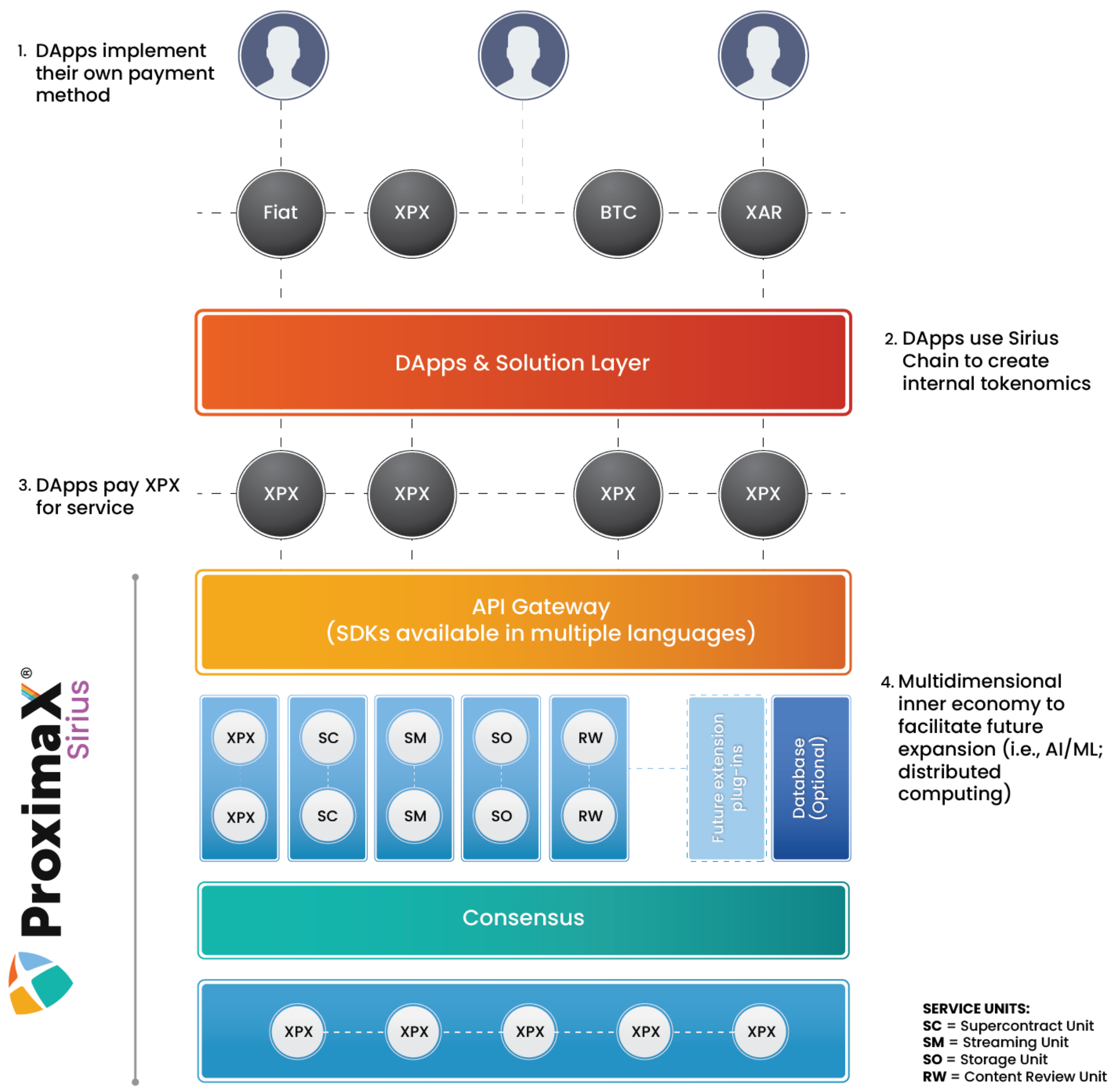

1. DApps implement their own payment method
Fiat
XPX
BTC
XAR
DApps & Solution Layer
2. DApps use Sirius Chain to create internal tokenomics
3. DApps pay XPX for service
XPX
API Gateway (SDKs available in multiple languages)
ProximaX Sirius
XPX
SC
SM
SO
RW
Future extension plug-ins
Database (Optional)
4. Multidimensional inner economy to facilitate future expansion (i.e., AI/ML; distributed computing)
Consensus
SERVICE UNITS:
SC = Supercontract Unit
SM = Streaming Unit
SO = Storage Unit
RW = Content Review Unit

Figure 3: Multidimensional inner economy.

Native Coin (XPX)

The blockchain protocol dictates the native currency that will be in circulation. For Sirius Chain, it is XPX. With the platform's multiple services tied to the blockchain protocol, users that want to use the platform's services must pay in XPX.

Service Units

XPX is used to subscribe to the ProximaX Sirius public network in exchange for Service Units through an automated inner exchange. Service Units are comparable to what some chains call a Gas token that is used to execute smart contracts. In ProximaX Sirius, several integrated services benefit from using several types of "Gas."

Types of Service Units:

- **Storage Unit (SO) for data storage:** A unit represents the ability to store data. 1 SO corresponds to 1 GB of space stored for a period of four weeks.

- **Streaming Unit (SM) for data streaming:** A unit represents the amount of data transferred between nodes or nodes and consumers. 1 SM corresponds to 1 streamed GB.

- **Supercontract Unit (SC) for executing Supercontracts:** 1 SC corresponds to 1 billion Supercontract operation codes (opcodes).

- **Review Unit (RW) for feedback and content review:** 1 RW corresponds to 1 review of content.

Service Units users:

1. **Consumers:** They consume services and make payments for such services.
2. **Network contributors:** They provide services to consumers in exchange for payments for providing these services.
3. **Internal consensus:** Used to represent the capacity and veracity of a node for providing a service.

This approach enables ProximaX Sirius to have a multidimensional inner economy where Service Units power the services rendered by different node actors. It also facilitates the future expansion of platform services.

2.2. Sirius DEX

Sirius DEX is a decentralized exchange (DEX) built to facilitate the automated exchange between the native token (XPX) and Service Units. Sirius DEX will be extended to include the exchange between SDAs at a later stage.

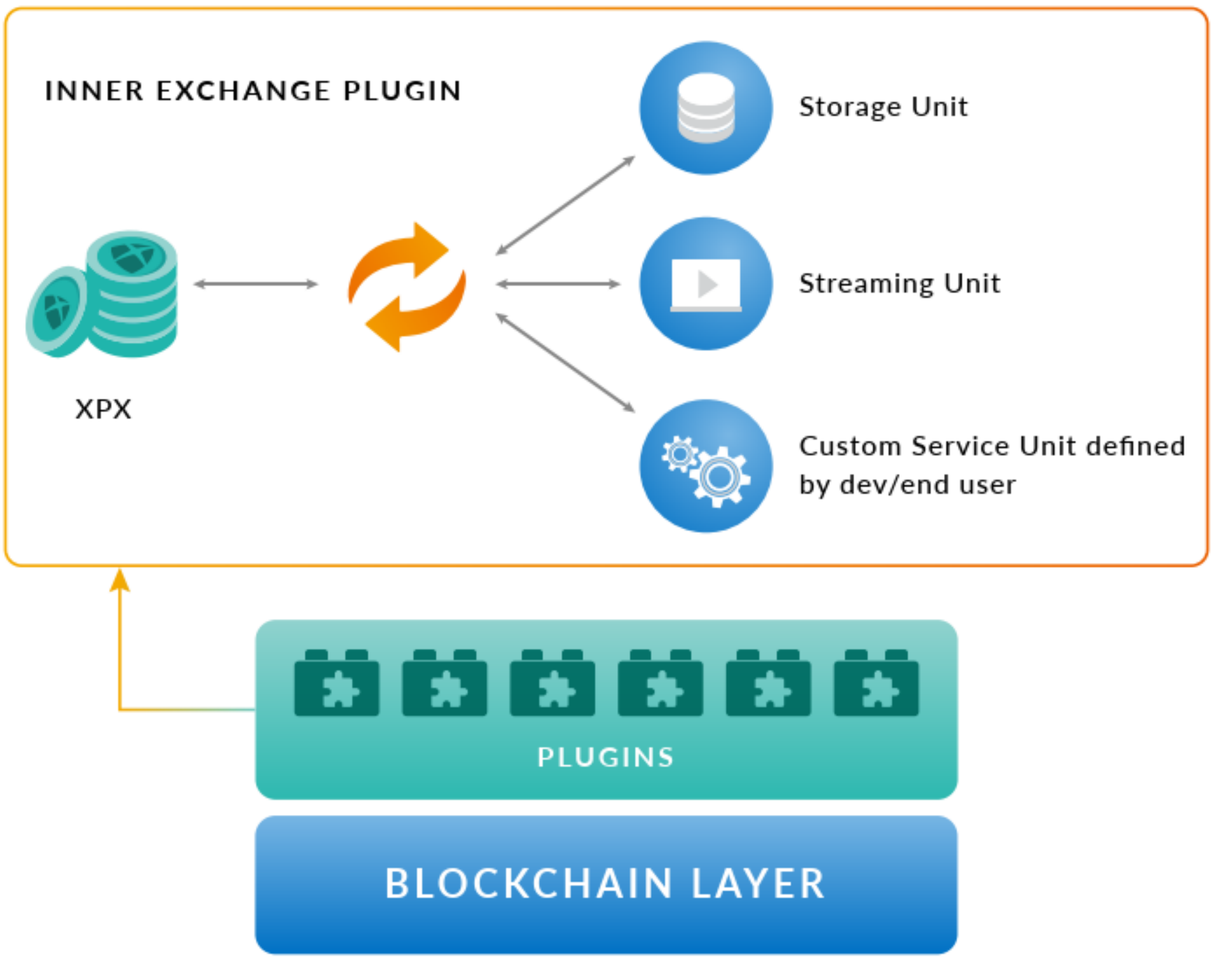


Figure 4: Sirius DEX.

As well as creating services via applications vertically on top of the platform, developers can also create new core services and corresponding token economies using the Sirius expandable plugin infrastructure. A new core service can use the Sirius DEX to exchange new types of Service Units. An example would be the creation of a new Service Unit for a search engine core service.

The Sirius DEX uses a tunable exchange rate algorithm to keep the price of platform services stable and competitive. For example, as data storage and internet bandwidth become universally cheaper over time, Service Unit prices should decrease.

3. Sirius Chain

3.1. Design

Actors

Node:	Role:
Validator	All platform nodes serve as a Validator. Validators validate and process all service transactions in a blockchain block (e.g., make payments; assign and remove nodes from services).
Harvester	A Harvester is a Validator that participates in block production thereby earning rewards.
Consumer	A platform user that initiates transactions.

Process Flow

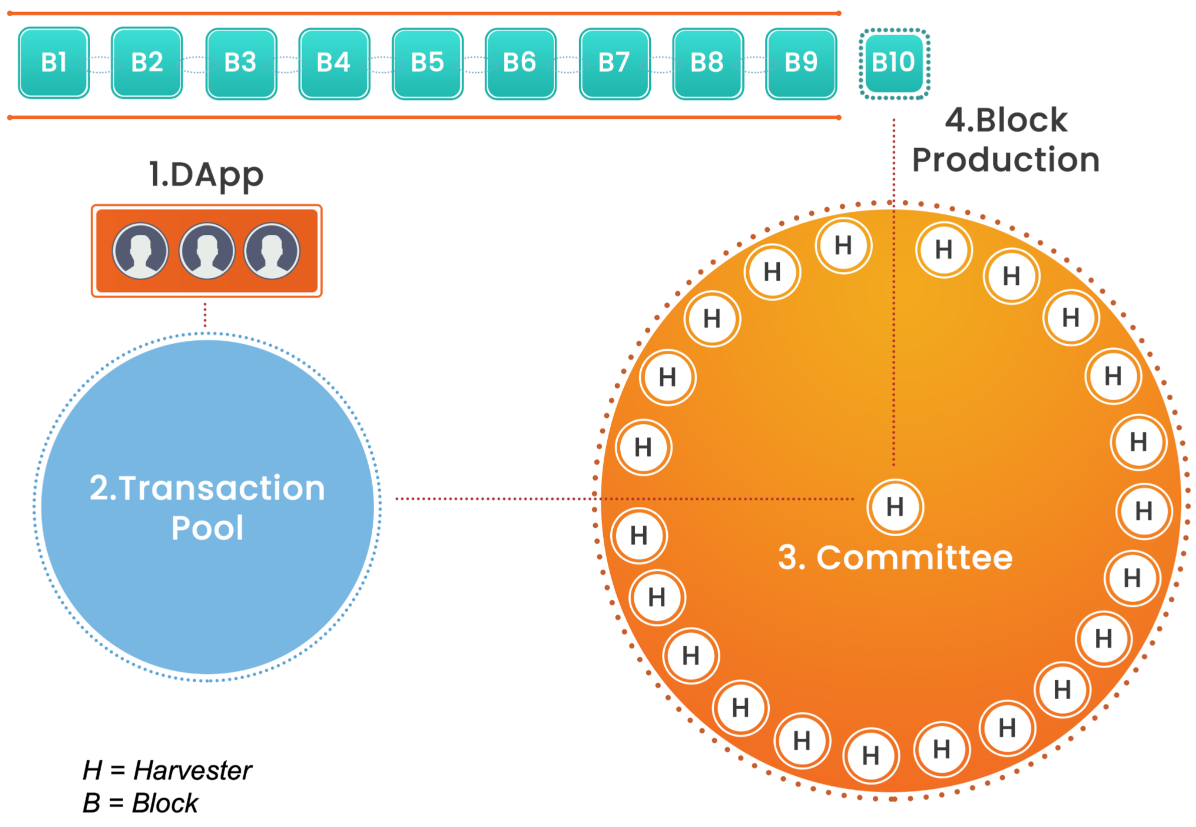


Figure 5: Sirius Chain transaction process flow.

3.2. Features

i. Account

An account is a mutable state stored on the Sirius Chain governed by a key pair: private and public keys. In other words, you have a “deposit box” in the blockchain, which only you can modify with your private key. As the name suggests, the account owner must keep the private key secret. Anyone with access to it will be able to control the account.

ii. Namespace

Namespace enables you to create on-chain unique and easy to remember names for your accounts and SDAs. A Namespace starts with choosing a name that is not already reserved, similar to an internet domain name. By announcing an alias transaction, you can associate a Namespace with an account or an SDA identifier (ID). Namespace works under a rental contract executed on-chain, paid in XPX, and with a duration period calculated according to block height.

iii. Sirius Digital Assets (SDAs)

You can create any digital asset on the platform. SDAs are built-in contracts defined on the blockchain protocol to enable consumers to make digital representations.

For example, an SDA can represent:

- a currency;
- a consumption measurement;
- a tangible asset;
- a non-fungible asset;
- a rewards point;
- a financial instrument, e.g. a derivative or a bond;
- a vote; or
- a status flag.

You can define SDAs by associating them with a Namespace that you own and customize their properties, which includes:

- divisibility - number of decimal places;
- duration - expiry time or never expiring;
- initial supply;
- supply mutability (i.e., changeable quantity); and
- transferability - freely transferable or transferable only between issuer and recipient.

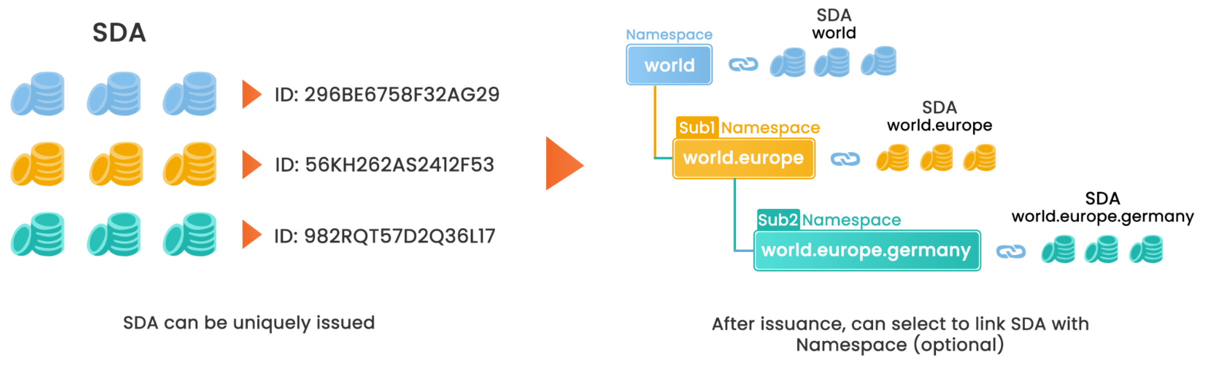


Figure 6: Token creation and namespace linking.

iv. Multi-level Multi-signature Transaction

Multi-signature is a cosignatory agreement (a one-time disposable contract) between account signatories (e.g., how many need to sign to execute a transaction or add/remove a signatory). Multi-level Multi-signature enables you to have multiple levels of agreements between other cosignatories, making it useful for comprehensive approval processes. Multi-signature agreements have an expiry time to execute, after which it will lapse.

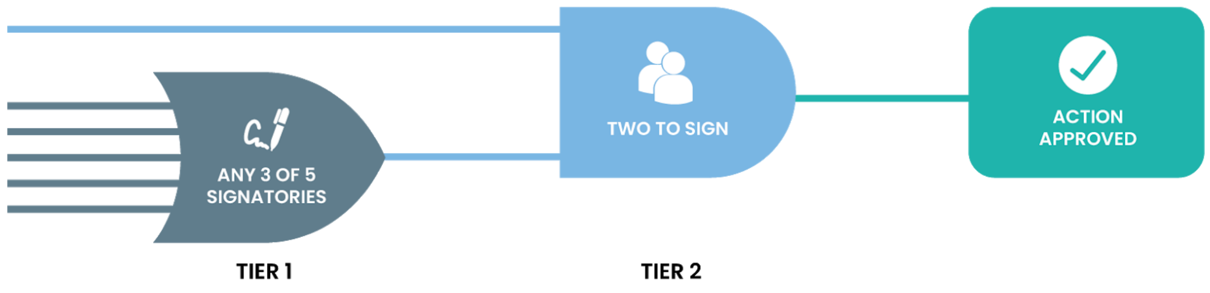


Figure 7: Multi-level multi-signing of a transaction.

v. Aggregate Transaction

An aggregate transaction merges multiple transactions into one, allowing trust-less swaps, escrows, and other advanced logic. Once all signatories sign the transaction, the exchange automatically and irrevocably executes. In our context, all these multiple transactions are called “inner transactions.” These inner transactions can be of any type of SDAs, i.e., it could involve XPX and a few other SDAs in one aggregate transaction. These aggregate transactions therefore, give us a very powerful exchange mechanism. One example is the exchange of NFT for XPX and commission fee for the introducer in a tri-party arrangement where the buyer pays XPX and the seller sells the NFT and the agent introducer gets the commission fee. When all parties sign, the three inner transactions get disbursed accordingly, with the NFT going to the buyer, some XPX going to the seller and the remaining XPX going to the agent introducer.

vi. Metadata

You can predefine objects on Sirius Chain by adding custom Metadata to transactions, accounts, Namespaces, and SDAs. Liken this as a note being tagged to the object of transaction.

vii. Cross-chain Transaction

A cross-chain transaction is the exchange of tokens between two blockchains. Also called "atomic swap," it involves locking up funds in the chain of the sending party and then issuing the token in the receiving chain.

For example, transactions can occur between:

- the public and a private chain; or
- two private chains; or
- two Catapult public chains, which opens ProximaX Sirius's services to external ecosystems such as the Symbol[4] blockchain.

3.3. Consensus

A consumer needs to pay XPX to use the public platform. A Harvester will receive XPX as a fee for forging or creating new blockchain blocks. Each block consists of transactions that have fees attached. The protocol adds the fees to create a block reward for Harvesters.

Like any public blockchain implementation, a fair and autonomous network ecosystem will need a consensus mechanism that determines the different rewards participating actors receive for providing a service. Sirius Chain uses Proof of Stake (PoS), Proof of Greed (PoG), and Fast Finality.

i. Proof of Stake

In contrast to Bitcoin's high-energy consuming Proof of Work (PoW) consensus, Sirius Chain uses the more efficient PoS to select and reward the next block producer. Sirius Chain's PoS is unique in that it gives block formation preference to Harvesters with a high XPX stake and considers node reliability and work activity to promote a healthy and performant network.

ii. Proof of Greed

Sirius Chain uses the Proof of Greed (PoG) consensus protocol, an extended reputation algorithm that keeps transaction fees closer to their actual cost and prevents Harvesters from becoming greedy.

How it works:

1. Consumers offer the maximum transaction fee they are willing to pay, which a wallet's software can auto-generate.

[4] Symbol public blockchain: www.symbolplatform.com

2. Consumers send the unconfirmed transactions to the network's Transaction Pool.
3. Harvesters take unconfirmed transactions from the pool, form blocks, and propose a fee amount that does not exceed the maximum specified by the Consumers.
4. The PoG algorithm assigns a Greed Value, the ratio between the fee proposed by the Harvester and the maximum amount offered by the Consumers.
5. The lower a Harvester's fee, the less "greedy" it is and the higher its chances of producing the next block and earning fees.

iii. Fast Finality

The Sirius Chain PoS protocol uses a Fast Finality weighted voting mechanism to quickly and efficiently reach a consensus before appending a new block onto the chain. This method ensures blocks are "final" and irrevocable once committed to the blockchain, making it unlikely that Validators will store different blockchain versions.

The protocol randomly elects Harvesters to form a Committee that performs the voting, giving preferences to Harvesters with higher stakes and reputation based on the number of blocks they have signed. Non-performing Harvesters that have not produced a block for more than a year lose their ability to become elected.

ProximaX formulated a mathematical framework for weighted voting to quantify Harvester voting profiles to ensure a performant Committee. The scheme, which applies once a Committee has formed, assigns to Harvesters scores based on the size of their stakes and contribution to protocol execution. The higher a Harvester’s score is, the more weight its vote carries. A Committee Member that fails to sign blocks, due to being faulty or offline, is assigned a lower reputation and is more likely to be removed from the Committee.

<u>Weighted voting - four stages:</u>

a. **Propose block**
The protocol elects a Committee Member (Harvester) with the lowest Greed Value as the next Block Proposer. The Greed Value derives from the accepted fees from the Harvester's last produced block. If the Harvester has not yet produced a block, a low Greed Value is assigned to facilitate participation by newcomers.

b. **Pre-voting**
The Block Proposer must then propose a block to the Committee for their preliminary acceptance, requiring at least ⅔ of the Committee by weight to cast a pre-vote. If the Block Proposer does not deliver the block or delivers an invalidly formed block, the Committee Members cast a nil vote and move on to the next elected Block Proposer.

c. **Pre-commit**
Each Committee Member checks that at least ⅔ by weight have favorably cast their pre-votes to pre-commit the block.

d. **Commit**
If at least ⅔ by weight have successfully pre-committed, the block is committed to the blockchain and a new block height is formed.

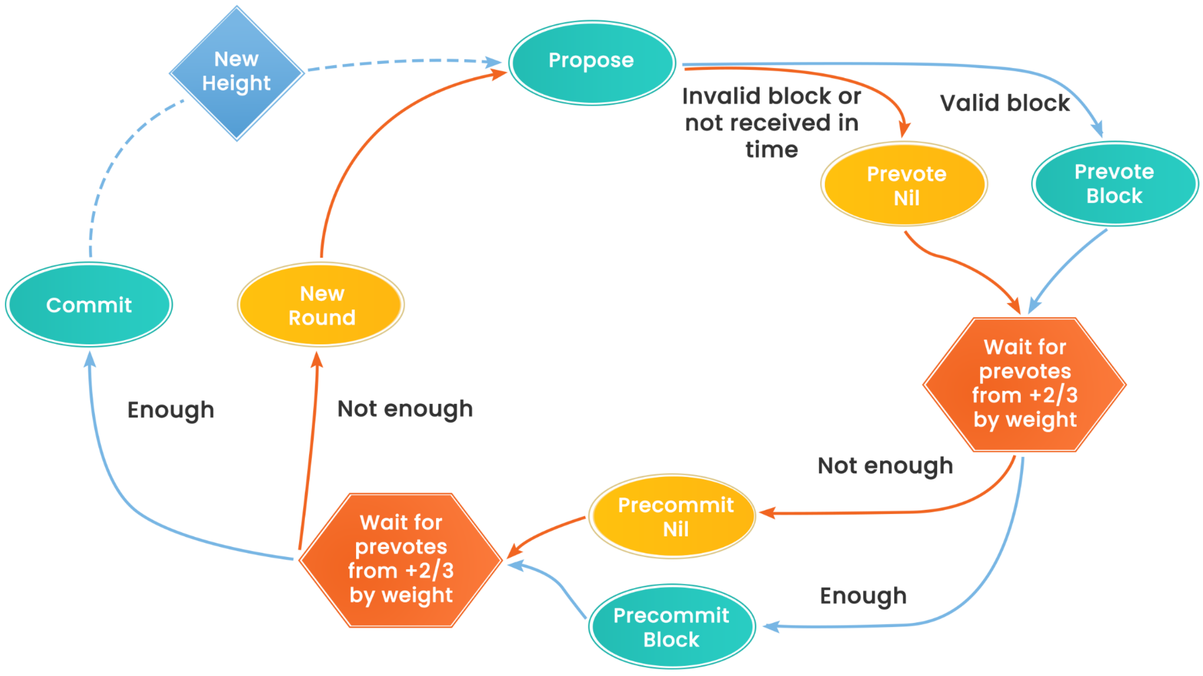
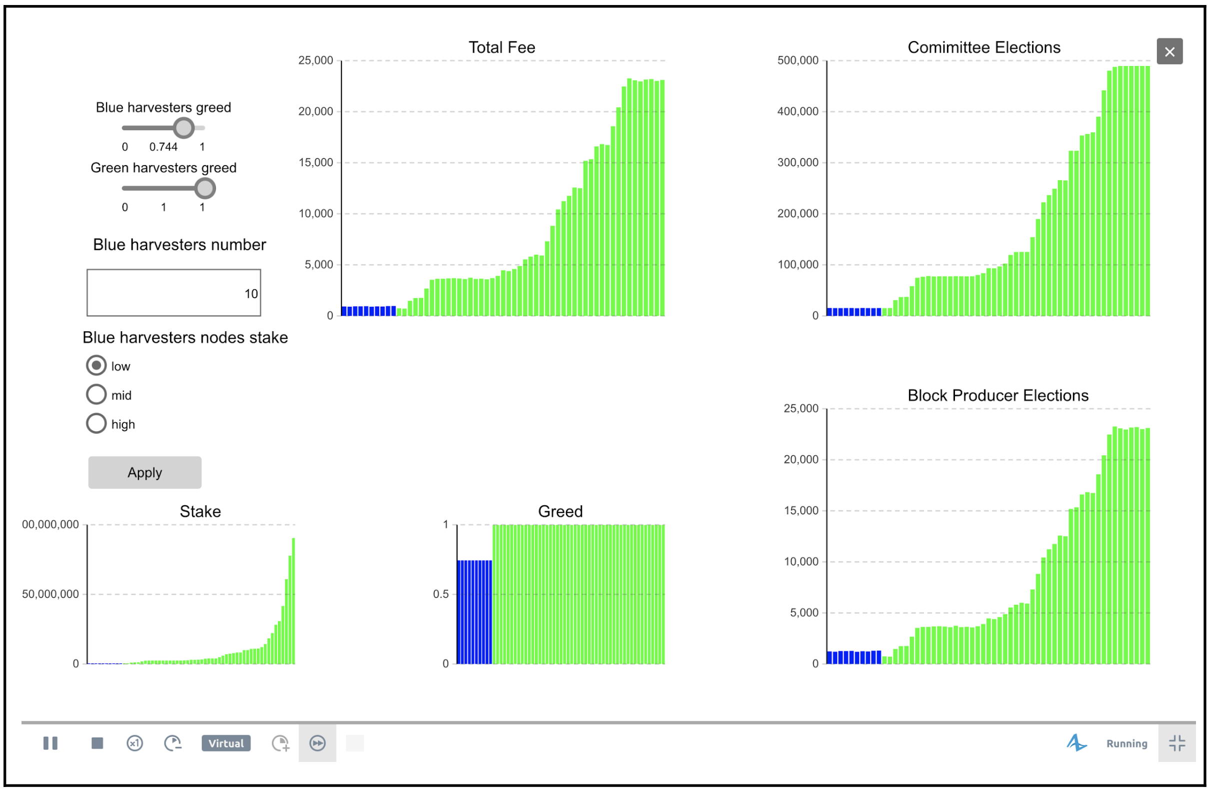


Figure 8: Weighted voting cycle.

3.4. Simulations

The below simulations show that Harvesters with low Greed Values have a better chance of being selected as a block producer and earn fees.



Blue harvesters greed: 0.744

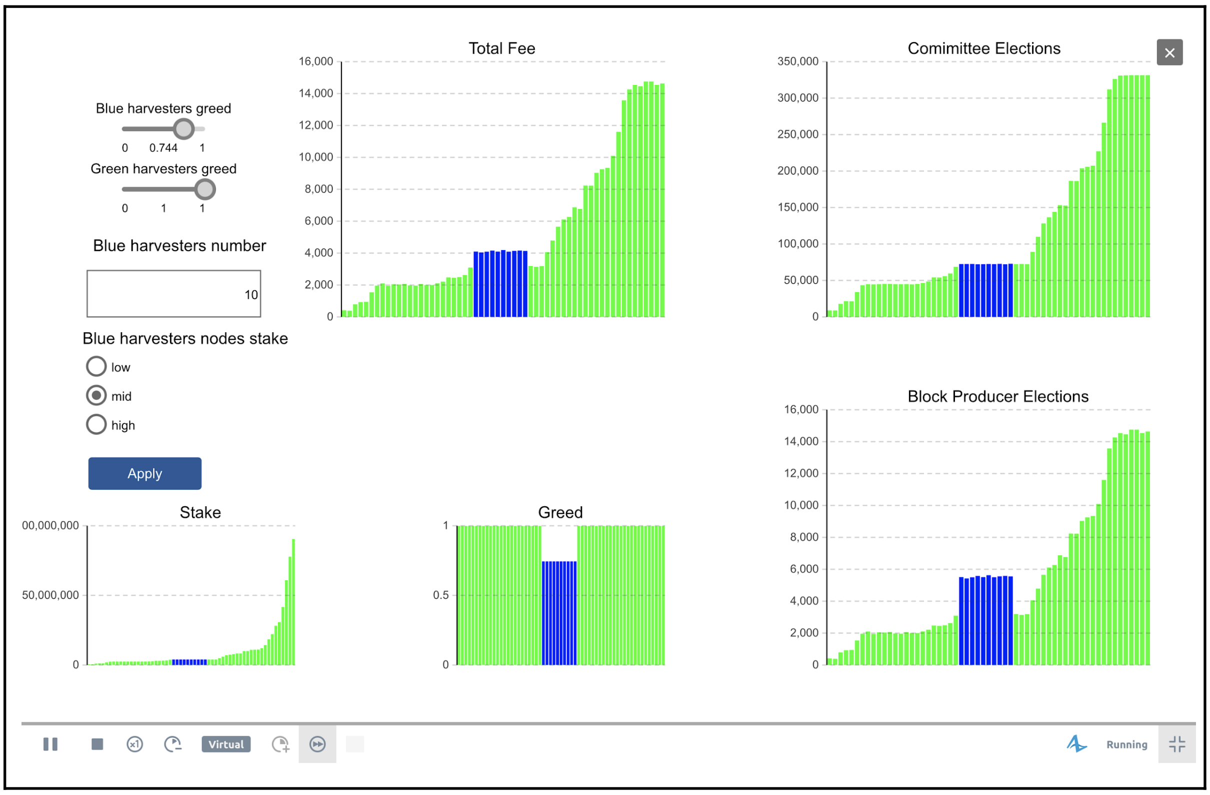
Green harvesters greed: 1

Blue harvesters number: 10

Blue harvesters nodes stake: [✓ low] mid high

Apply

Figure 9: Non-greedy Harvesters with a small stake.



Blue harvesters greed: 0.744

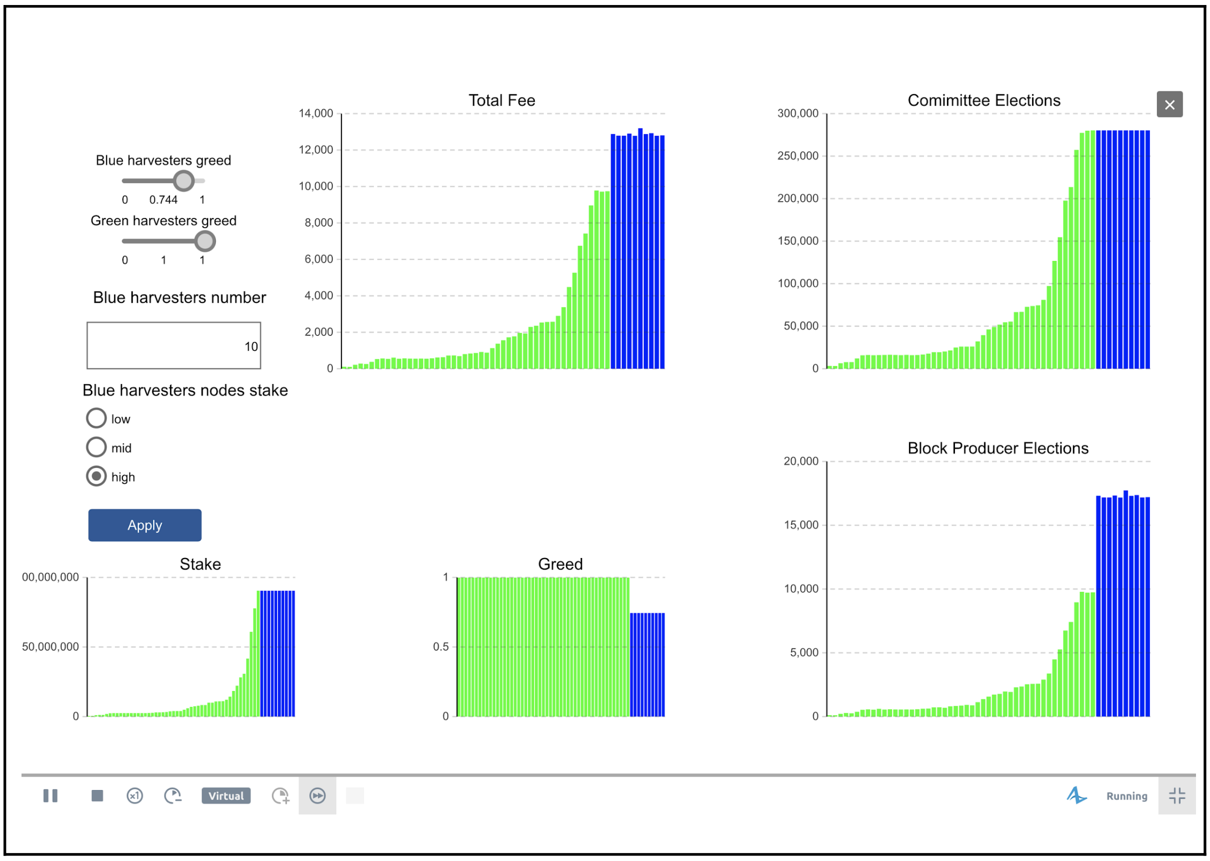
Green harvesters greed: 1

Blue harvesters number: 10

Blue harvesters nodes stake: low [✓ mid] high

Apply

Figure 10: Non-greedy Harvesters with a medium stake.



Blue harvesters greed

0 0.744 1

Green harvesters greed

0 1 1

Blue harvesters number

10

Blue harvesters nodes stake

low

mid

[✓ high]

Apply

Figure 11: Non-greedy Harvesters with a large stake.

3.5. Attacks

Sirius Chain's consensus protocols protect the network from potential attacks.

i. Large-stake Attack

As PoS gives preference to the wealthiest Harvesters, the potential vulnerability is that a malicious Harvester with a 51% stake can launch a Large-stake Attack, forge the most blocks, and take control of the network.

Sirius Chain's PoS algorithm prevents this by creating a fair spread when selecting and rewarding Harvesters, meaning that even a Harvester with a small stake has a chance of having its block recorded on Sirius Chain.

ii. Zero-fee Attack

As PoG penalizes greedy Harvesters, the potential vulnerability is that malicious Harvesters could cheat and manipulate the consensus through a Zero-fee Attack. Here, Harvesters charge zero fees, forge the most blocks, and take control of the network.

Sirius Chain's PoG algorithm eliminates the possibility of a Zero-fee Attack by giving preference to Harvesters that take an average fee rather than a minimum fee.

3.6. Priority Transactions

Sirius Chain's transaction pool protocol places unconfirmed transactions into a priority queue to better manage high transaction volumes.

High priority	Multi-signature approval transactions executed for prepaid Consumer services (e.g., **DataModificationApprovalTransaction** and **DownloadApprovalTransaction** introduced in this paper).
Medium priority	Single transactions for prepaid Consumer services (e.g., **PrepareDriveTransaction** and **DataModificationSingleApprovalTransaction** introduced in this paper).
Low priority	All other transactions.

Table 1: Priority queue for transaction pool management.

3.7. Temporary & Permanent Bans

Sirius Chain uses a reputation protocol that removes non-performant, faulty, and malicious Validators from the network.

A Validator will periodically drop existing node connections to make room for new interactions to avoid isolated groups from forming, thus promoting decentralization. At each selection round, a Validator inspects the age of all its connections and drops the oldest.

A communication between Validators is considered an interaction, and the protocol scores each interaction as either "successful," "neutral," or "failed." For example, when a Validator provides new valid data, the interaction is marked as "successful." If it gives no data, the interaction is "neutral." If it cannot interact due to low connectivity or has invalid or old data, the interaction is marked as "failed." A Validator can be banned temporarily or permanently from the network if the failures are regular and severe.

Failure	**Severity**	**Connection closed**	**Node can reconnect**	**Node can be selected**	**Node can send data**
Consecutive interaction failures	Light	No	Yes	Yes (after reconnecting)	Yes
Old data	Light	Yes	Yes	Yes (after reconnecting)	Yes (after reconnecting)
Invalid data	Severe	Yes	No	No	No

Table 2: Temporary and permanent bans for failed interactions.

4. Sirius Storage

4.1. Design

Actors

Node:	Role:
Drive Owner	A subscriber who rents disk drive space from the storage layer and therefore is the owner of the drive space. A Drive Owner instructs Replicators. This should not be confused with Harddisk Drive owners who participate to provide disk space in the storage layer.
Replicators	Store and modify the Drive and perform downloads and uploads based on incoming transactions.
Consumer	Downloads data from a Drive.

Transactions

Transaction:	Initiated by:	Purpose:
PrepareDriveTransaction	Drive Owner	Drive creation.
DataModificationTransaction	Drive Owner	Drive modification.
StoragePaymentTransaction	Drive Owner	Pay for storage.
DriveClosureTransaction	Drive Owner	Close Drive.
DataCancelModificationTransaction	Drive Owner	Cancel modification.
ReplicatorOffboardingByDriveOwnerTransaction	Drive Owner	Remove Replicator from Drive.
ReplicatorOnboardingTransaction	Replicator	Onboarding.
DataModificationApprovalTransaction	Replicators	Drive modification approval.
DownloadApprovalTransaction	Replicators	Download approval.
DataModificationSingleApprovalTransaction	Replicator	Single modification approval.
DriveVerificationTransaction	Replicators	Drive state verification.
DownloadTransaction	Consumer	Start downloading data from Drive.
FinishDownloadTransaction	Consumer	Stop downloading data from Drive.

Events

Events:	Initiated by:	Purpose:
JoinToDriveEvent	Validators	Join Replicators to a Drive.
LeaveDriveEvent	Replicator, or Validators, or Drive Owner.	Replicator off-boarding.
EndOfBillingPeriodEvent	It occurs automatically every four weeks for Drive Owners and every 24 hours for Consumers unless prematurely ended.	Payment for work done.

Process Flow

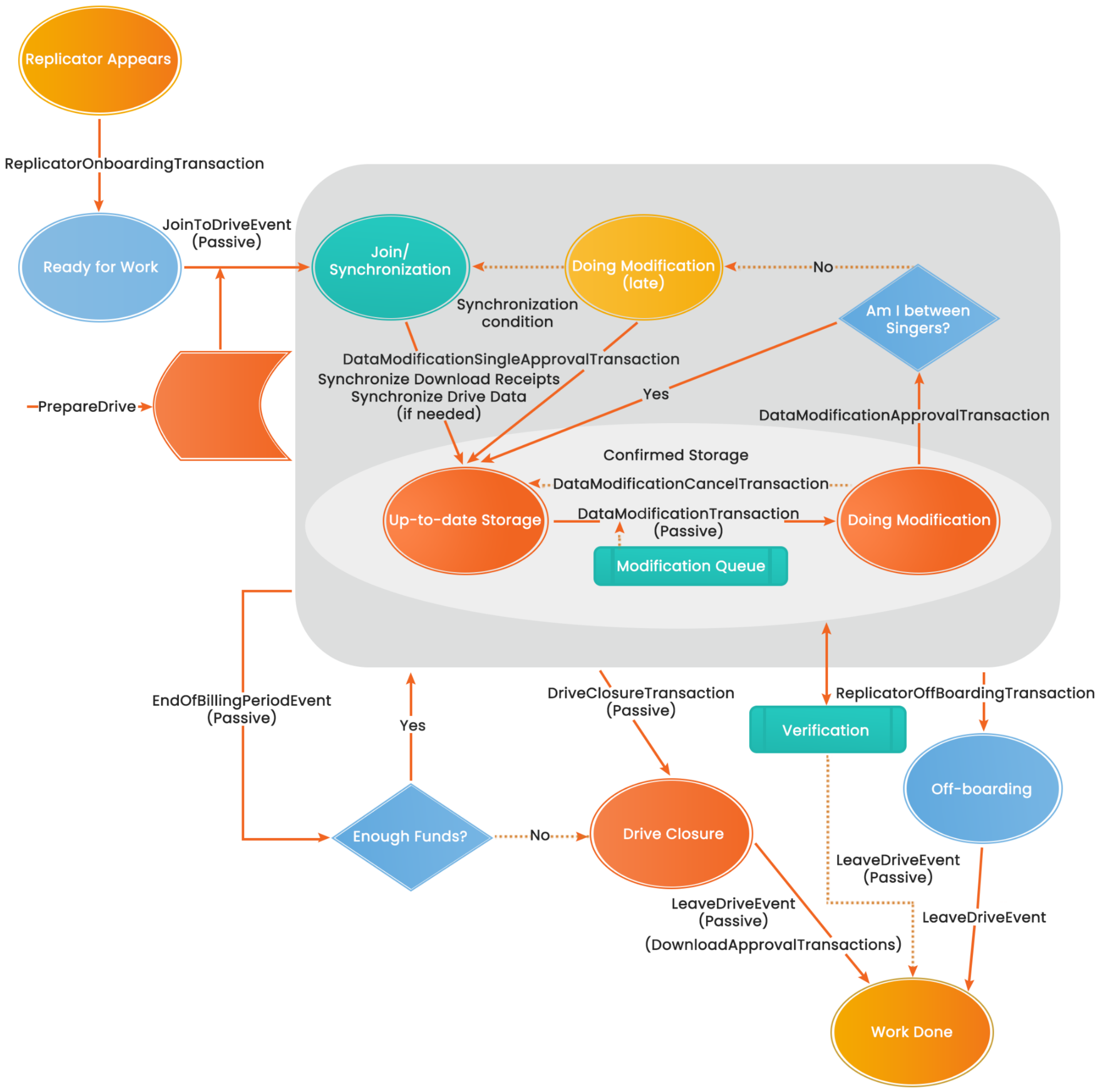

Replicator Appears
ReplicatorOnboardingTransaction
Ready for Work
JoinToDriveEvent (Passive)
PrepareDrive
Join/ Synchronization
Doing Modification (late)
No
Am I between Singers?
Synchronization condition
DataModificationSingleApprovalTransaction
Synchronize Download Receipts
Synchronize Drive Data (if needed)
Yes
DataModificationApprovalTransaction
Confirmed Storage
DataModificationCancelTransaction
Up-to-date Storage
DataModificationTransaction (Passive)
Doing Modification
Modification Queue
EndOfBillingPeriodEvent (Passive)
Yes
DriveClosureTransaction (Passive)
Verification
ReplicatorOffBoardingTransaction
Off-boarding
Enough Funds?
No
Drive Closure
LeaveDriveEvent (Passive)
LeaveDriveEvent (Passive)
(DownloadApprovalTransactions)
LeaveDriveEvent
Work Done

Figure 12: Sirius Storage process flow.

4.2. Drive Creation

To create a Drive, the Drive Owner uses software to broadcast a **PrepareDriveTransaction** and states within the transaction's Metadata the desired drive size and the number of replications. Each Replicator assigned to a Drive equates to one replication. Therefore, if the Drive Owner requires 100 replications for critical data or mass distribution, 100 Replicators are assigned to the Drive.

The Drive Owner needs to ensure sufficient Storage Units are available in the Drive account to execute the request. If there are insufficient Storage Units, the Drive Owner can post a **StoragePaymentTransaction** to top-up the account. The Drive size needs to be sufficiently large enough to enable data rollbacks. For example, if the Drive Owner wants to store a 1 GB file, he should allocate 2 GB storage space.

Validators, listening to the blockchain for trigger transactions, pick up the Drive creation request from the Drive Owner and assign Replicators with sufficient storage space to the Drive. After the **JoinToDriveEvent**, the Replicators start listening to all triggers for the corresponding Drive.

By default, all Drive data is publicly available to Sirius platform users. To prevent this, Drive Owners can elect to restrict the opening of Download Channels and encrypt all files at the client side.

4.3. Drive Modification

<u>Modification steps:</u>

i. Modification Initiation

To initiate a Drive modification, the Drive Owner must first choose the command options Replicators are required to perform from an Action List:

1. File modification.
2. Remove file.
3. Rename file.
4. Create a folder.
5. Remove folder.
6. Rename folder.

Once the Action List items are selected, the Drive Owner needs to post a **DataModificationTransaction** that contains the Content Download Information (CDI) on the Action List and files to be modified The Drive Owner pays for uploading the modified files to the Replicators. The payment amount equals the size of the uploaded data times the number of replications. This amount is locked by the Validators when the Drive Owner requests the modification.

At this point, the Drive Owner's software should prevent the Drive Owner from deleting the files to be downloaded by the Replicators until the blockchain confirms the subsequent **DataModificationApprovalTransaction** posted by the Replicators.

If the Drive Owner wishes to cancel the modification before the **DataModificationApprovalTransaction** is confirmed, the Drive Owner must post a **DataModificationCancelTransaction**. If canceled, the Replicators would then reject the **DataModificationTransaction** and remove it from the Drive's Modification Queue. The Drive Owner would still be required to pay any Replicators that already executed the modifications and needed to roll back the changes.

ii. Modification Queue

After the blockchain confirms the **DataModificationTransaction**, the Replicators add the transaction to the Drive's Modification Queue. Once the transaction reaches its turn in the queue, the Replicators download the necessary data modifications requested by the Drive Owner stated in the Action List and CDI and execute the modification request.

iii. Modification Sandbox

The Replicators first download all the necessary files and execute the requested actions in their respective local Sandboxes (a transitory or staging storage area that will be purposely configured) to test the modifications.

If the Sandbox modifications are successful, the Replicators post a **DataModificationApprovalTransaction** and apply all changes in a live state (non-Sandbox) as soon as the network confirms the transaction and records an imprint of the new Drive State on the blockchain.

If the Sandbox modification fails, the Replicators send the previous Drive State imprint to the blockchain. The Replicators still receive payment for work done.

Sandbox modification fails if:

- The downloaded data size exceeds the size declared in the **DataModificationTransaction**.
- The downloaded data does not contain the Action List.
- The Action List is corrupted or includes invalid actions.
- An error occurs upon interacting with the File System.
- The size of the Drive after applying the Action List exceeds the ordered Drive size.

The Replicators clear their Sandbox of any files if:

i. The Replicators apply the modifications to the Drive State.
ii. The Drive Owner cancels the data modification via a **DataModificationCancelTransaction**.
iii. The Replicators record a failed data modification on the blockchain.

iv. Modification Approval

If at least ⅔ plus 1 Replicators reach a consensus on the new file structure and its corresponding file hash, which acts as a unique fingerprint for identification, the Replictors sign a Multi-signature **DataModificationApprovalTransaction** and apply the modification to the new Drive State.

Replicators that were too late to sign the latest group Multi-signature **DataModificationApprovalTransaction** must catch up by posting a **DataModificationSingleApprovalTransaction** and synchronize their storage. For efficiency, Replicators can synchronize by downloading modified data from both other Replicators and the Drive Owner. Single approval Replicators are paid the same as other Replicators for work done, except they must cover the transaction cost.

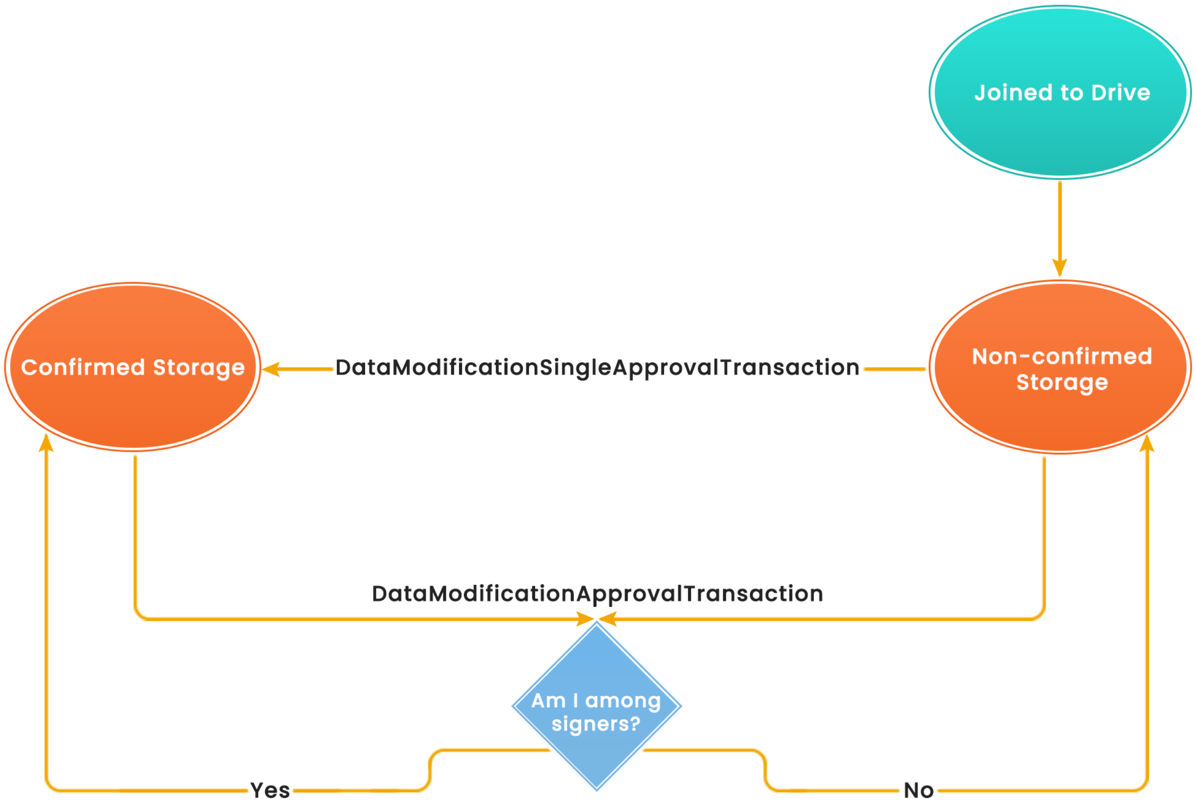


Figure 13: Data modification approval process.

4.4. Download Channel

Download steps:

i. Open Download Channel

A Consumer locates a file in the network using the file's CDI. The Consumer prepays Streaming Units via a **DownloadTransaction** to open a Download Channel with the file's Drive. The amount of Streaming Units deposited equates to the data download size made available to the Consumer. The Consumer also prepays XPX to pay for transactions made by the Replicators. The Consumer can top-up prepayment by posting further **DownloadTransactions**.

ii. Make payments

Validators lock the Consumer's prepaid Streaming Units in an escrow account. After each successful download of 1 MB data size, the Consumer signs a Receipt off-chain for work done and sends a copy to each of the Drive's Replicators. Each Replicator also sends all other Replicators the Receipts off-chain to ensure synchronization.

The Replicators verify that the Receipts:

- cumulatively reflect the amount downloaded, give or take a few MB (Consumer can download up to 1 MB for free);
- are not duplicates;
- are for consumed downloads and not future services; and
- do not exceed the Consumer's prepaid deposit.

Replicators ignore invalid Receipts. If a Receipt exceeds the prepaid deposit, the Replicators stop uploading data and only recommence once the Consumer tops up his account.

The Replicators then use the valid Receipts to form a collective opinion. If at least ⅔ plus 1 Replicators agree on work done, give or take a few MB, they post a Multi-signature **DownloadApprovalTransaction**. The Validators pay the Replicators the median value of the Opinions at the end of every 24 hours download billing period.

For each Download Channel, Replicators only store the last Receipt so that the storage space consumed by Receipts is kept minimal. The Replicators can remove all Receipts after the Download Channel is closed.

iii. Close Download Channel

A Consumer can close a Download Channel by posting a **FinishDownloadTransaction**. The Replicators reach a consensus on any payments owed and post the result via a Multi-signature **DownloadApprovalTransaction**. The Validators issue payment to the Replicators and return any unused prepaid funds to the Consumer.

4.5. Drive Closure

The Drive Owner posts a **DriveClosureTransaction** to close a Drive. Suppose the transaction brings a premature end to the current Drive billing period (less than four weeks). In this case, the Replicators post outstanding **DownloadApprovalTransactions** to prompt payment by the Validators for the services rendered until that point. The Validators return the Drive Owner any unused Storage Units from the Drive account and assign the Replicators to new Drives.

If the Storage Units in the Drive account run out, the Validators close the Drive. There is no grace period as Drive Owners cannot expect the decentralized network participants to work for free. The Drive Owner's software can provide adequate warnings when prepaid funds are running low and require a top-up.

4.6. Replicator Verifications

A Replicator needs to store the latest modifications to possess the latest Drive State by signing a Multi-signature **DataModificationApprovalTransaction**, or if it missed it, a **DataModificationSingleApprovalTransaction** followed by synchronization.

The storage verification protocol locates, penalizes, and removes out-of-sync Replicators. Verification occurs randomly, multiple times a day, where Replicators send each other proof of having the latest Drive State. The Replicators then post their opinions on the verification results via a multi-signature **DriveVerificationTransaction** that requires at least ⅔ plus 1 Replicators to sign.

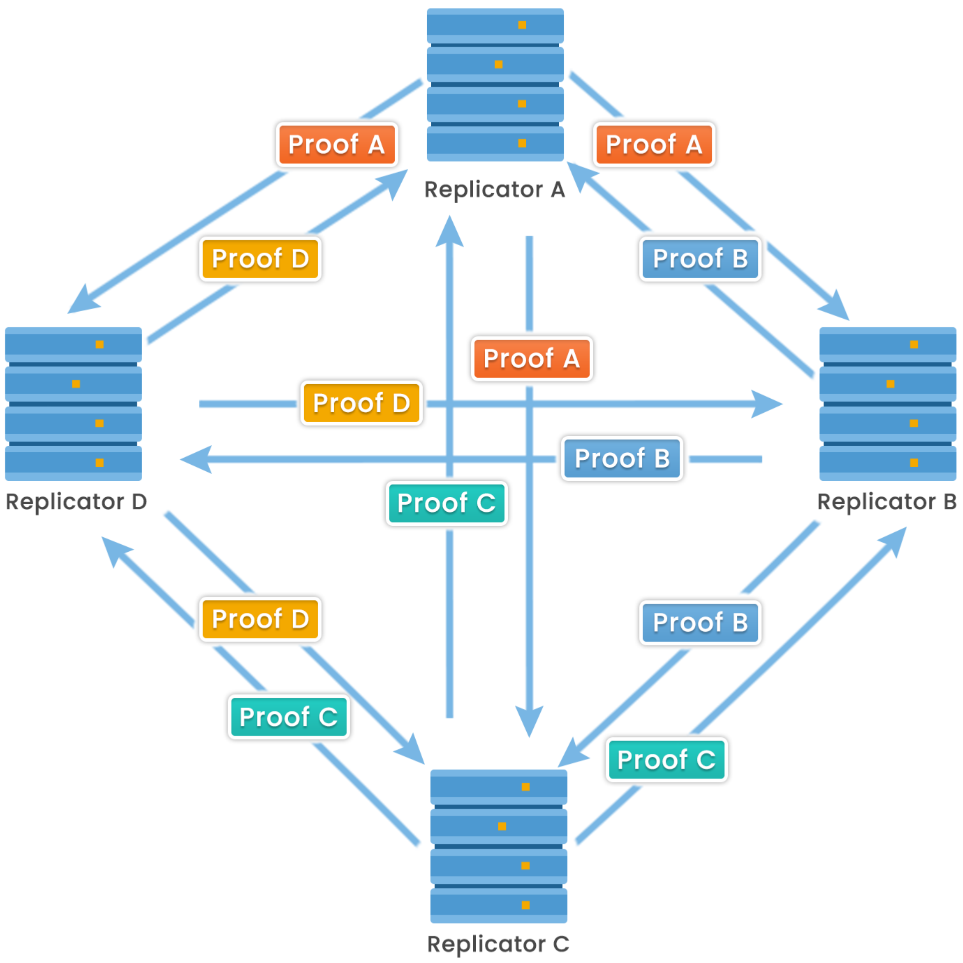


Figure 14: Replicator cross-verification.

Validators identify from the **DriveVerificationTransactions** any Replicators that have not participated in the verification process for more than two days, remove them from the list, and confiscate their deposit as punishment to deter non-performance.

4.7. Collective Decision Making

Drive Replicators give their collective opinions in the Metadata of Multi-signature transactions, where at least ⅔ plus 1 Replicators are required to reach a consensus and sign the transaction. Validators listening to the blockchain processes the transactions. It computes the median of all opinions to form a final decision (e.g., payments to be made or removing a Replicator from the Drive).

Multi-signature opinion transactions include:

- **DownloadApprovalTransaction.**
- **DataModificationApprovalTransaction.**
- **DriveVerificationTransaction.**

4.8. Replicator Priority Assignment

ProximaX developed an algorithm to prioritize the assignment of Replicators to Drives by Validators. As soon as a Harvester adds all transactions to a block and appends it to the blockchain, Validators form a list of the Drives with missing Replicators. As there can be periods where Validators cannot fill all Replicator vacancies on Drives due to low Replicator numbers in the network, the algorithm creates a priority queue that prioritizes Drives with less than four Replicators where data is most vulnerable. Validators will position Drives with the same level of priority in the queue in a deterministic way.

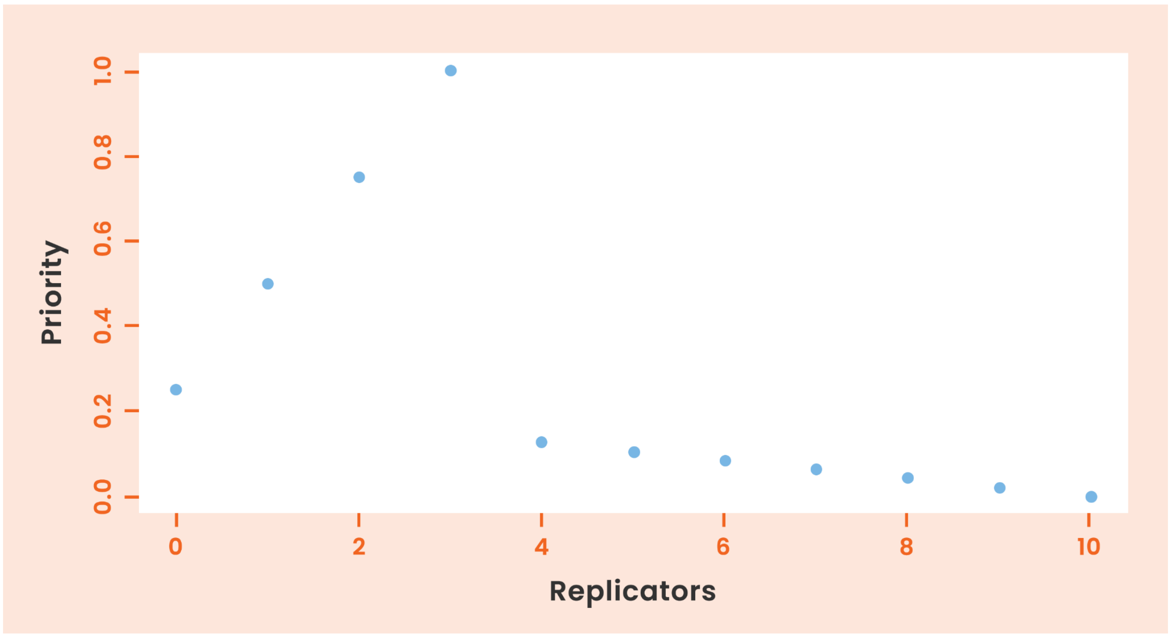


Figure 15: Replicator assignment prioritization.

4.9. Mass Content Distribution

A Drive with many Replicators (e.g., 100) for critical data or mass content distribution would function inefficiently, as too many Replicators would need to communicate with one another and sign Multi-signature transactions.

The storage protocol resolves this inefficiency by splitting Replicators into sub-groups of no more than 20 Replicators each. Here, Replicators download data, perform verifications, and express opinions within their sub-group.

Data upload to Consumers becomes more efficient as only one randomly selected sub-group Download Channel is created.

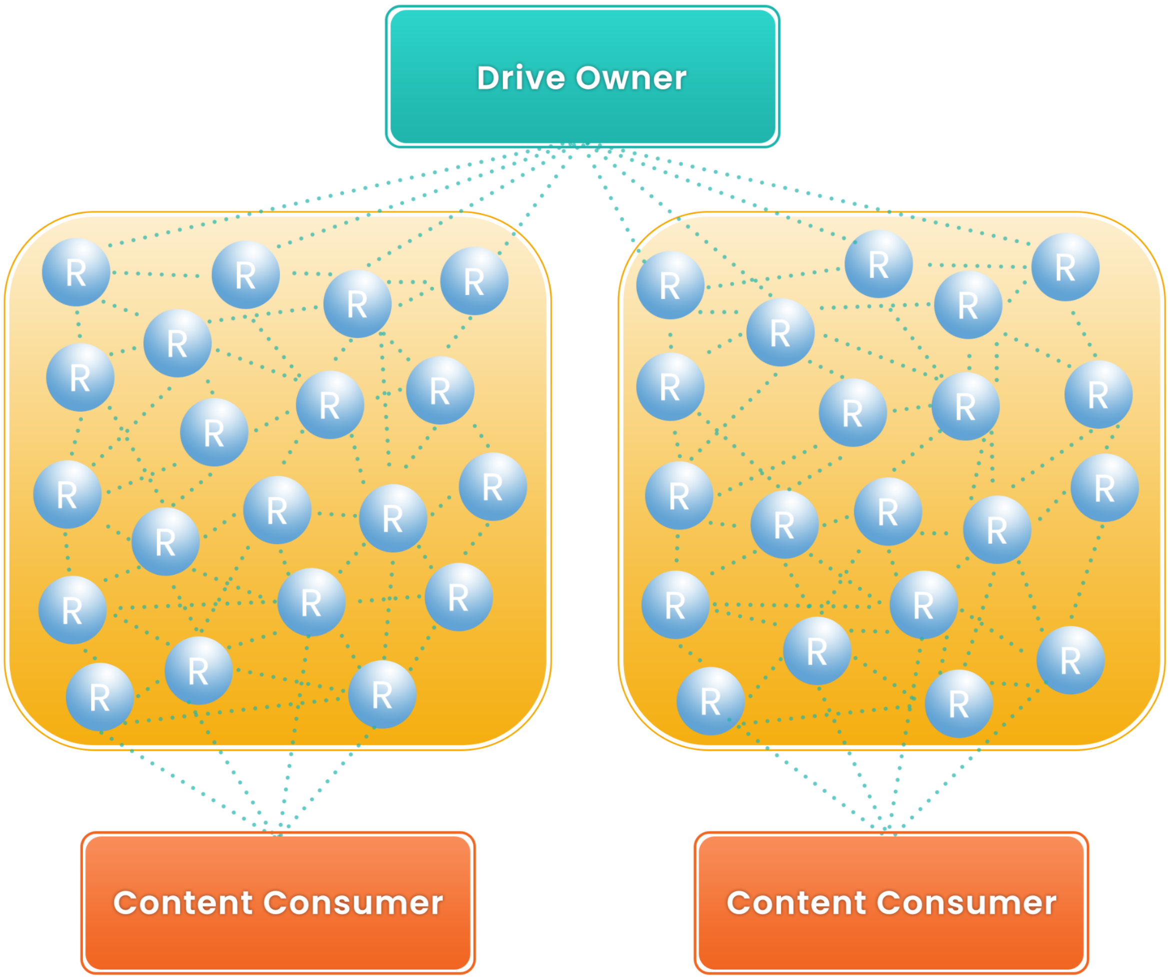


Figure 16: Replicator sub-groups with not more than 20 per sub-group.

4.10. Privacy

Public keys are publicly visible, which means anyone can track network activity. It is possible to take a VPN-like approach to provide the option of anonymous access to Sirius Storage and Sirius Stream services.

The below conceptual design illustration shows how Replicators can form an off-chain Virtual Drive with zero allocated memory and connect to Download Channels on behalf of Consumers, removing Consumers' direct interaction with Drives that causes the exposure of their public keys. As a Virtual Drive acts only as a conduit for data download, no storage space is required. Virtual Drives are off-chain so that there is no blockchain transaction record between Consumers and the Virtual Drive Replicators.

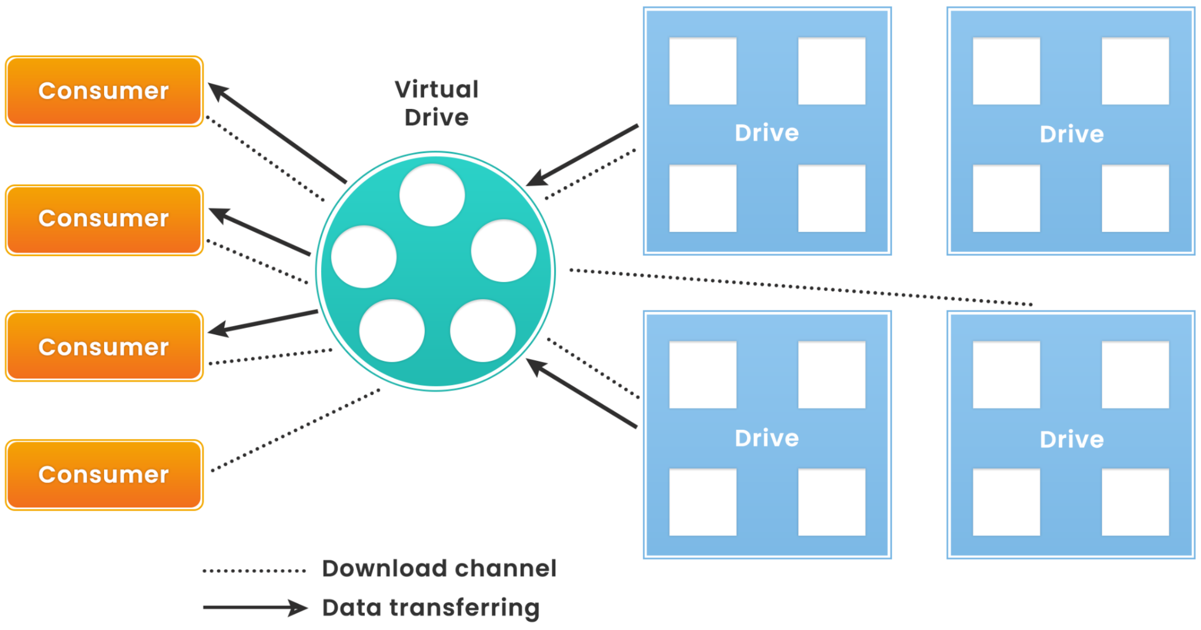


Figure 17: Virtual Drive for anonymous downloading.

Virtual Drives interact with other Drives on behalf of Consumers as a Consumer would by opening a Download Channel, making a deposit for the consumption of services, and sending Receipts to the Drive's Replicators or Distributors (see next chapter on Sirius Stream).

Consumers must pay extra for the service, as there is the need to pay the Virtual Drive Replicators for work done and cover the payments they make to obtain services.

4.11. Sponsorship

A Sponsor, such as an advertiser, can sponsor storage and streaming services to make them free of charge for Consumers and Receivers (see next chapter on Sirius Stream).

The Sponsor creates a Download Channel by posting a **DownloadTransaction** and can make it available to selected Consumers or Receivers by including their public keys in the transaction or set the value to zero (0) to allow anyone to view the stream.

The Sponsor can close the channel by posting a **FinishDownloadTransaction**, after which Validators return any unused prepaid funds.

4.12. Replicator Onboarding & Offboarding

i. Onboarding

To contribute storage space to the network, a Replicator needs to post a **ReplicatorOnboardingTransaction** to trigger a **JoinToDriveEvent** and prove its available storage space by depositing collateral Service Units converted from XPX by Validators using the Sirius DEX.

The Validators assign the Replicator to a Drive(s). After that, the Replicator listens to all incoming transactions (triggers) belonging to its respective Drive. The Validators then register the Replicator's assigned storage space for the Drive as "in use."

ii. Offboarding

To offboard from a Drive, a Replicator needs to post a **ReplicatorOffboardingTransaction** and state the Drive's Public Key, triggering a **LeaveDriveEvent**. When offboarding, the Validators return to the Replicator all of its deposited Storage Units. However, the Replicator will forfeit Streaming Units to pay for the data upload work of a new Replicator that joins the Drive.

A Harvester will not add a **ReplicatorOffboardingTransaction** to a block and allow a Replicator to leave if it means the Drive will end up with less than four Replicators. Suppose a Replicator nonetheless decides to stop performing its role as a Replicator. In that case, the Replicator verification process will identify this, and the Validators will offboard the Replicator and confiscate its deposit (see 4.6. above).

The Drive Owner can post a **ReplicatorOffboardingByDriveOwnerTransaction** to remove a low-performing Replicator (e.g., uploads data slowly) if at least four Replicators remain assigned to the Drive to ensure sufficient data availability. The Replicator is paid a prorated amount for the billing period and has its deposit returned. The Drive Owner must compensate the network for work done to replace the Replicator.

4.13. Replicator Locations Allowlist & Blocklists

During Drive creation, a Drive Owner can include in the **PrepareDriveTransaction**:

1. **Replicator Allowlist:** Countries where its assigned Replicators can be located.
2. **Replicator Blocklist:** Countries where its assigned Replicators cannot be located.

When Validators assign Replicators to a Drive, they assign Replicators according to the Drive Owner location requests.

Replicators state their location during the **ReplicatorOnboardingTransaction**.

4.14. Storage Tokenomics

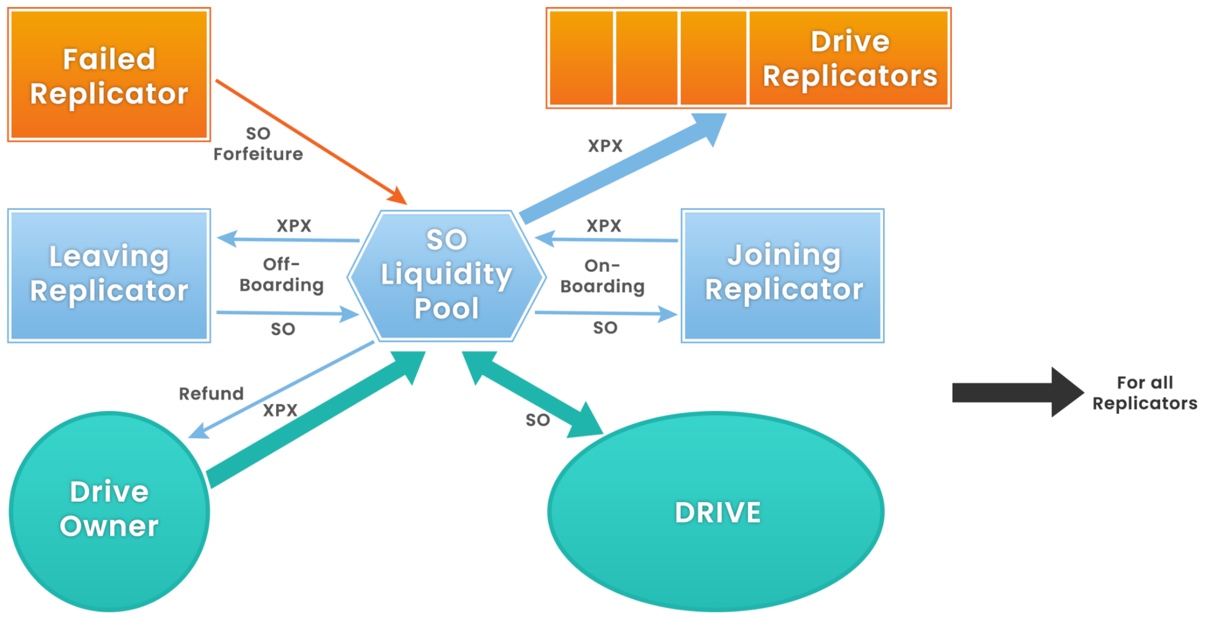


Figure 18: Storage Units flow.

The Validators manage token flows based on incoming transactions and make payments. The Drive Owner pays only in XPX, and the Replicators only get paid in XPX. According to Sirius DEX's latest exchange rates, the Validators make all necessary Service Unit conversions on behalf of the Drive Owners, Replicators, and Consumers.

Once per billing period (every four weeks for Drive Owners and every 24 hours for Consumers), the Validators make XPX payments to the Replicators for work done by converting Storage Units and Streaming Units locked in the Drive account and the Consumers' escrow accounts.

A Replicator only receives payment for the periods it has stored the latest Drive State. Validators will pay a Replicator a prorated amount if there are periods where the protocol confirmed that a Replicator has been non-performing (not storing the latest Drive State). Validators identify non-performing Replicators through the **DriveVerificationTransactions** and excessive **DataModificationSingleApprovalTransactions** posted on the blockchain. If a Replicator does not prove that it possesses the latest Drive State for more than two days, it forfeits its deposit (see section 4.6.).

5. Sirius Stream

5.1. Design

Actors

Node:	**Role:**
Stream Sender	Creator of the data stream and instructs Distributors.
Stream Distributors	Replicator that distributes a data stream based on incoming transactions.
Redistributors	Stream Receivers that participate in downstream redistribution.
Stream Receiver	Viewer of the stream.

Transactions

Transaction:	**Initiated by:**	**Purpose**
StreamStartTransaction	Sender	Start stream.
StreamFinishTransaction	Sender	End stream.
DataModificationCancelTransaction	Sender	Cancel stream.
StreamPaymentTransaction	Sender	Make prepayment.
DownloadTransaction	Receiver	Open Download Channel and make prepayment.
FinishDownloadTransaction	Receiver	Close Download Channel.
DataModificationApprovalTransaction	Distributors	Modify Drive and make payments.
DataModificationSingleApprovalTransaction	Distributor	Single modification approval.
IncludeToPaymentTransaction	Redistributor	Receive reward for redistribution.

Process Flow

Sirius Storage doubles up as Sirius Stream for live streaming. Much of the processes and features described in the previous chapter apply to Sirius Stream.

A Stream Sender (Sender) is a Drive Owner, and the Drive's Replicators perform the role of Stream Distributors (Distributors). The network treats streaming as a form of Drive data modification.

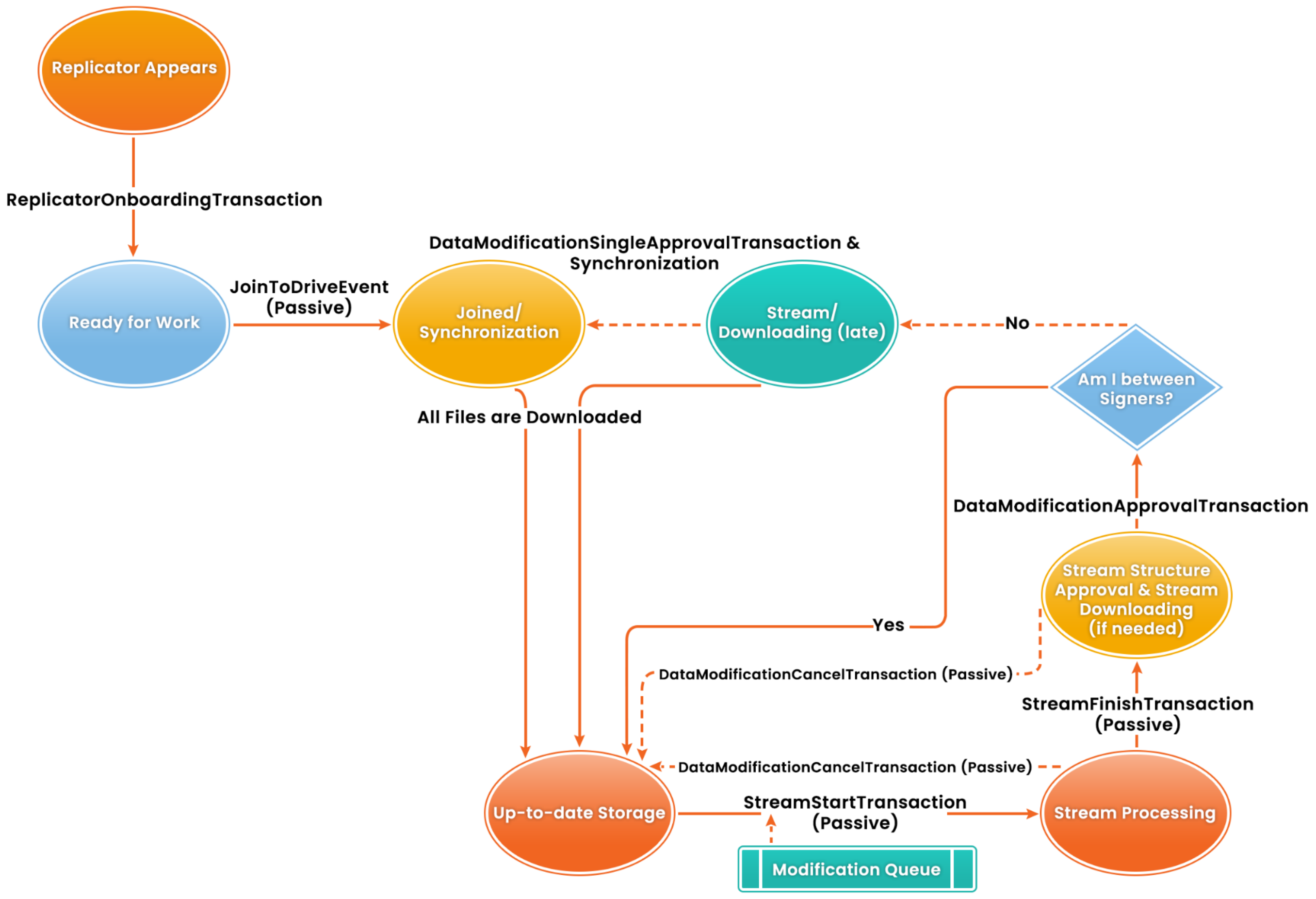

Replicator Appears
ReplicatorOnboardingTransaction
Ready for Work
JoinToDriveEvent (Passive)
Joined/ Synchronization
DataModificationSingleApprovalTransaction & Synchronization
Stream/ Downloading (late)
No
Am I between Signers?
All Files are Downloaded
DataModificationApprovalTransaction
Stream Structure Approval & Stream Downloading (if needed)
Yes
DataModificationCancelTransaction (Passive)
StreamFinishTransaction (Passive)
DataModificationCancelTransaction (Passive)
Up-to-date Storage
StreamStartTransaction (Passive)
Stream Processing
Modification Queue

Figure 19: Sirius Stream process flow.

5.2. Live Stream Creation

The Sender must perform the following actions to create a stream:

1. If not yet done, order a Storage Drive of the required size.
2. Make a prepayment for the stream (Streaming Units) by posting a **StreamPaymentTransaction**. The Sender can post the same transaction during a live stream to increase the prepaid amount.
3. Post a **StreamStartTransaction** to commence the stream. The start transaction states the expected stream size, which needs to match the prepaid amount.

The **StreamStartTransaction** triggers the following:

a. The Validators lock up the prepayment in the Drive's account to pay for the Distributors' work once the stream has ended.
b. The Replicators add the request to the Drive's Modification Queue, as the Replicators process the stream as a type of data modification.
c. The Replicators commence the role of Distributors and transmit the stream when the data modification reaches its turn in the queue.

By default, all live streams are publicly available to Sirius platform users. To prevent this, Drive Owners can elect to restrict the opening of Download Channels and encrypt streams so that only permitted users can consume these streams.

5.3. Live Stream Transmission

When the **StreamStartTransaction** is next in line in the Modification Queue, the following occurs:

i. Playlist Creation by Sender

The stream segments are put chronologically into playlists by the Sender's software so that the Stream Receivers (Receivers) can view the segments as a complete stream. The playlists contain information on each stream segment's identification number, sequential order, and data size. The Sender signs each playlist with his private key so that the Distributors and Receivers can verify their origin.

ii. Download by Distributors

The Distributors begin downloading media segments once they have verified that (a) the Sender has signed each playlist and (b) the expected download size mentioned in the **StreamStartTransaction** does not exceed the Sender's prepaid amount. The Distributors download streaming segments in the correct chronological arrangement, matching each segment's sequential number with the corresponding playlist.

iii. Loop Recording for Lag-free Streaming

The Distributors store each segment as streaming loops in their respective sandbox to enable continuous streaming. If the incoming segments exceed the available sandbox space that equals the total Drive size, the Distributors delete the oldest segments and save the most recent one to the streaming loop.

iv. Download by Receivers

Receivers locate the stream on the network using its Content Distribution Information (CDI) and post a **DownloadTransaction** to make a prepayment deposit to open a Download Channel to view the stream. Receivers can top-up the deposit by posting another **DownloadTransaction** at any point during the stream if prepaid funds are running low.

The Receivers start downloading the stream segments in the correct chronological arrangement, including recorded segments, to create a buffer zone for lag-free live streaming. The network's buffer zone default is 60 segments, which amounts to approximately 60 seconds. The Receivers' software can adjust the buffer zone default.

v. Communication between Nodes

The Distributors notify Receivers about new segments to be uploaded. Distributors notify the Sender of each successfully downloaded segment.

The Receivers send Receipts to Distributors for every 1 MB streamed. Using the same Sirius Storage Receipt communication protocol, the Distributors verify that the Receipts:

a. cumulatively reflect the amount downloaded, give or take a few MB (Receiver can download up to 1 MB for free);
b. are not duplicates;
c. are for consumed downloads and not future services; and
d. do not exceed the Receiver's prepaid deposit.

If the Receipts do not pass the verification process, the Download Channel is closed. If Receipts pass verification, at the end of the 24 hour billing period, the Distributors form a collective opinion (at least ⅔ plus 1 Distributors) on work done using the Receipts and include the result in a **DownloadApprovalTransaction**. Validators then pay the Distributors the median value of opinions using the Receiver's deposit and return any unused funds to the Receiver.

As usual, Distributors that have not signed the Multi-signature approval catch up by posting a **DataModificationSingleApprovalTransaction** for the latest modification, just like in Sirius Storage. They then synchronize Drive data with the other Distributors or the Drive Owner.

vi. Close Download Channel

A Receiver can prematurely close a Download Channel before a stream ends by posting a **FinishDownloadTransaction**. The Distributors reach a consensus on any payments owed (at least ⅔ plus 1 Distributors) and post the result via a Multi-signature **DownloadApprovalTransaction**.The Validators issue payment to the Distributors and return any unused prepaid funds to the Receiver.

5.4. End Live Stream

The Sender posts a **StreamFinishTransaction** to end a live stream. The transaction's Metadata contains information on the playlists and segments streamed and instructions on whether the Replicators are required to save them to the Drive for storage streaming (streaming on-demand).

The transaction triggers the following:

i. Verification

Distributors verify whether the stream structure (playlists and media segments) declared by the Sender is consistent with local copies stored by each Distributor during the live stream. The verification ensures that the Sender cannot cheat the system and pay less by declaring a smaller streamed size.

ii. Approval

Distributors collectively verify and arrive at a consensus (at least ⅔ plus 1 Distributors) that:

a. the Sender has correctly declared the total stream size (give or take a default parameter); and
b. on the payment to be received for work done.

The Distributors sign a Multi-signature **DataModificationApprovalTransaction** and store to the Drive any requested data for streaming on-demand. The Validators then pays the Distributors for work done and returns any locked unused streaming funds to the Sender.

iii. Rejection

If the Distributors identify inconsistencies with the Sender's declaration, they do not sign the modification approval transaction, and the Drive is made unavailable to the Sender. The Drive can become available again after the Sender's software posts a **DataModificationCancelTransaction**.

5.6. Mass Live Streaming

As described in section 4.9., the formation of sub-groups is crucial to avoid bottlenecks in service caused by a large group of Distributors having to communicate amongst each other.

With Distributor sub-groups, decentralized live streaming becomes more efficient. Receivers only download a stream from one randomly selected Distributor sub-group. Distributors only exchange playlists, streaming segments, and Receipts within their sub-group.

However, even the formation of sub-groups may not cope with the significant bandwidth demands of thousands of Receivers. The network solves this by allowing Receivers to contribute their bandwidth and become downstream Redistributors, where Receivers collect and submit Receipts for work done and gain payment from Validators.

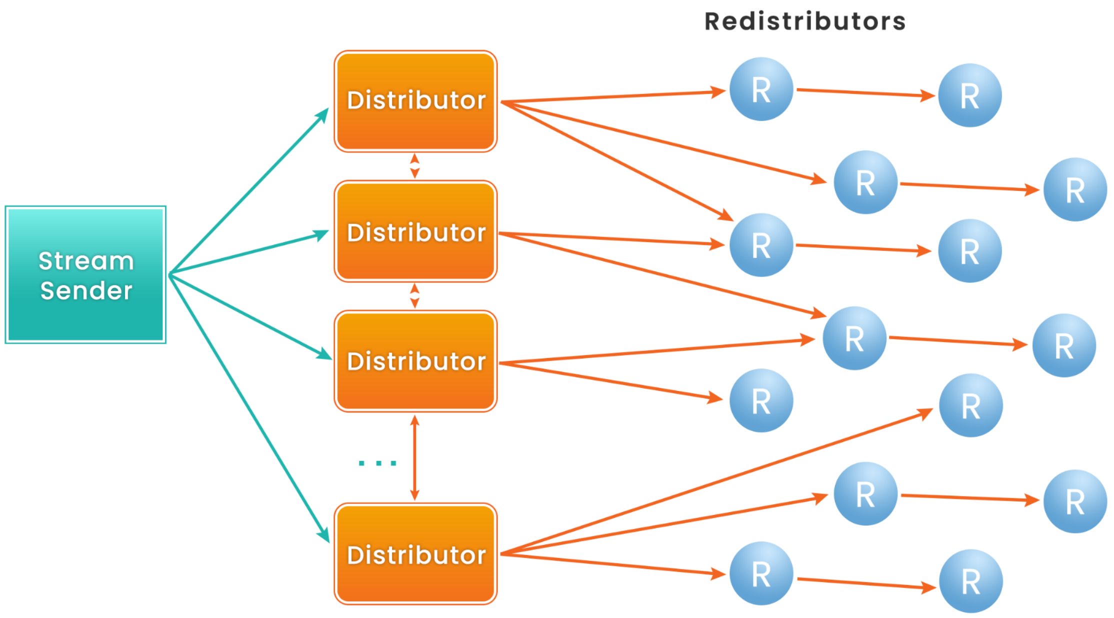


Figure 20: Redistributors.

<u>How it works:</u>

i. Redistributor Collects Receipts & Pays Fee

A Receiver sends Receipts to a Redistributor for every 1 MB downloaded from the Redistributors, just like how it would do with a Distributor. Likewise, the Redistributor checks that the Receipts:

- cumulatively reflect the amount downloaded, give or take a few MB (Receiver can download up to 1 MB for free);
- are not duplicates; and
- are for consumed downloads and not future services.

Before the end of the 24 hour billing period, the Redistributor posts an **IncludeToPaymentTransaction** to append its reward claim to a **DownloadApprovalTransaction** posted by Distributors. The **IncludeToPaymentTransaction** requires the addition of a small fee paid by the Redistributor to make network spamming or claiming small rewards (less than the cost of service) by the Redistributor unprofitable.

ii. Distributors Process Receipts & Form Consensus

If the Receipts pass the Redistributor's verifications, the Redistributor sends the Receipts to the Distributors. The Distributors also perform the same verifications as the Redistributor and ensure the Receipts do not exceed the Receiver’s prepaid deposit.

The **IncludeToPaymentTransaction** triggers the Distributors to form an opinion on the Redistributor's reward and include it in the **DownloadApprovalTransaction**.

iii. Validators Verify Fee & Issue Reward

The rewards issued by the Validators are the median of all Distributors' Opinions. Before issuing payment to a Redistributor, The Validators checks that the Redistributor posted an **IncludeToPaymentTransaction** and attached the required fee.

5.7. Streaming Tokenomics

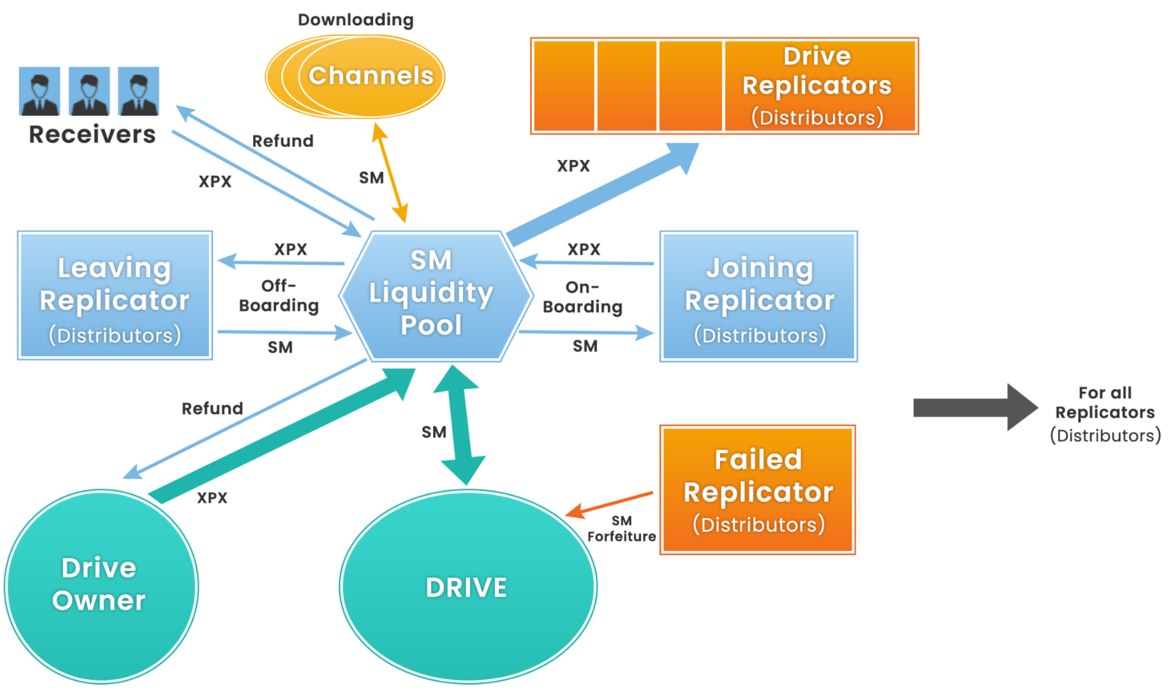

Downloading
Channels
Drive Replicators (Distributors)
Receivers
Refund
XPX
SM
XPX
Leaving Replicator (Distributors)
XPX
Off-Boarding
SM
SM Liquidity Pool
XPX
On-Boarding
SM
Joining Replicator (Distributors)
Refund
SM
XPX
Drive Owner
DRIVE
SM Forfeiture
Failed Replicator (Distributors)
For all Replicators (Distributors)

Figure 21: SM Units flow.

Similar to Sirius Storage tokenomics process flow (see section 4.13.), the Validators manage the token flows based on incoming transactions and payments. Streaming Units (SM Units) is the primary Service Unit used during live streaming.

6. Supercontract

6.1. Design

Actors

Node:	Description:
Creator	The creator of a Supercontract.
Caller	A user that calls a Supercontract.
Executors	Replicators that execute Supercontracts based on incoming transactions.

Transactions

Transaction:	Initiate by:	Purpose:
DeployTransaction	Creator	Deploy Supercontract to Drive.
EndDeployTransaction	Replicators	Multisig deployment verification results.
EndBatchExecuteTransaction	Replicators	Multisig batch execution result.
EndBatchExecuteSingleTransaction	Replicator	Late batch run drive synchronization.
ReleaseTransactionsTransaction	Replicators	Multsig triggered by Supercontract #1.
StartExecuteTransaction	Caller	Run a specific Supercontract function.

Process Flow

Color key:
Green - before executing a Supercontract.
Yellow - preparation for Supercontract work.
Blue - confirmed storage.
Orange - late/out of synchronization.

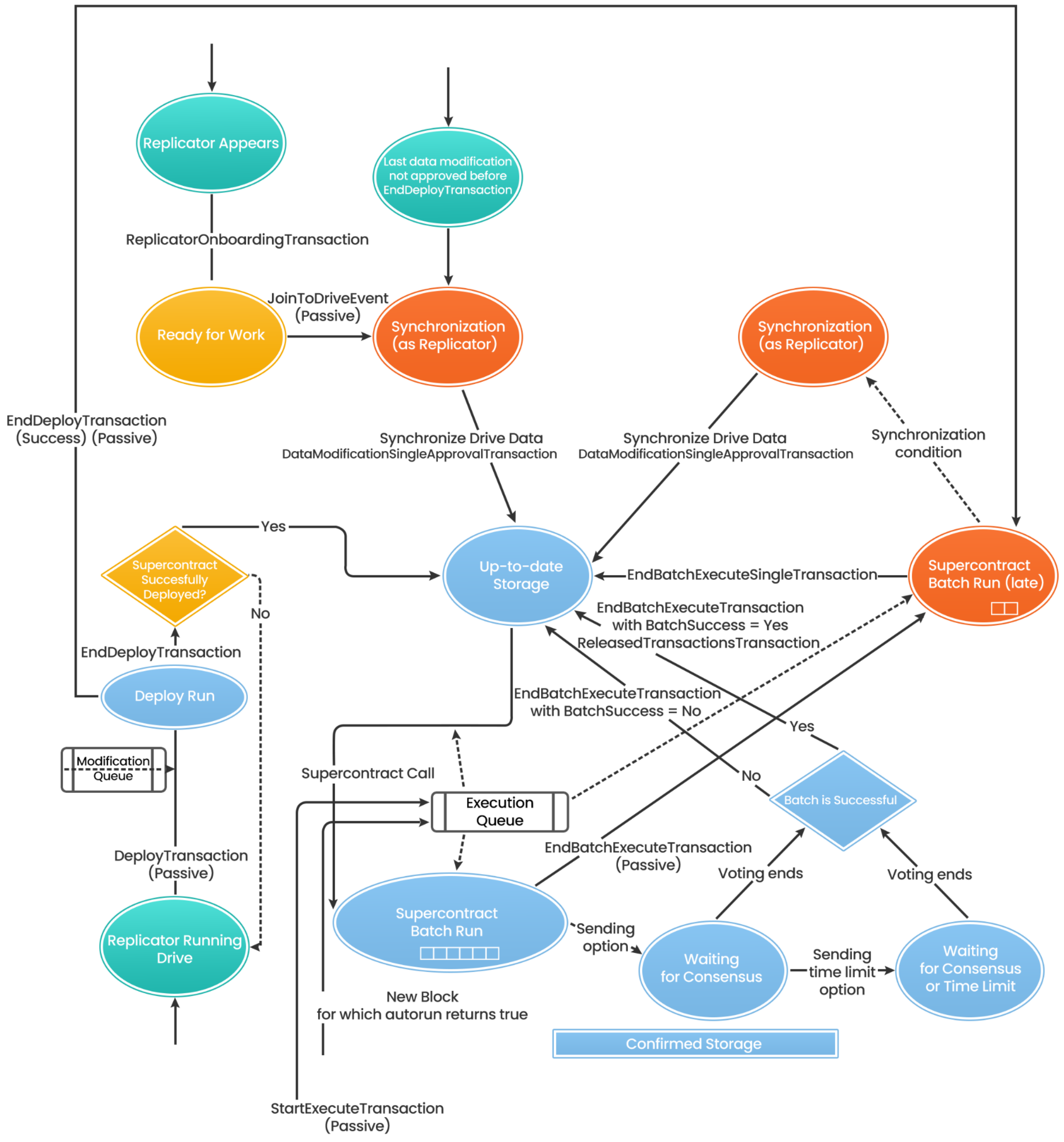

Replicator Appears
Last data modification not approved before EndDeployTransaction
ReplicatorOnboardingTransaction
Ready for Work
JoinToDriveEvent (Passive)
Synchronization (as Replicator)
Synchronization (as Replicator)
EndDeployTransaction (Success) (Passive)
Synchronize Drive Data
DataModificationSingleApprovalTransaction
Synchronize Drive Data
DataModificationSingleApprovalTransaction
Synchronization condition
Yes
Supercontract Succesfully Deployed?
No
Up-to-date Storage
EndBatchExecuteSingleTransaction
Supercontract Batch Run (late)
EndBatchExecuteTransaction with BatchSuccess = Yes
ReleasedTransactionsTransaction
EndDeployTransaction
Deploy Run
EndBatchExecuteTransaction with BatchSuccess = No
Yes
Modification Queue
Supercontract Call
No
Batch is Successful
Execution Queue
EndBatchExecuteTransaction (Passive)
DeployTransaction (Passive)
Voting ends
Voting ends
Supercontract Batch Run
Sending option
Waiting for Consensus
Sending time limit option
Waiting for Consensus or Time Limit
Replicator Running Drive
New Block for which autorun returns true
Confirmed Storage
StartExecuteTransaction (Passive)

Figure 22: Supercontract process flow.

6.2. Contract Creation

A Supercontract is essentially a "function box" that you can create and program to trigger the execution of specific actions upon the occurrence of some agreed event(s).

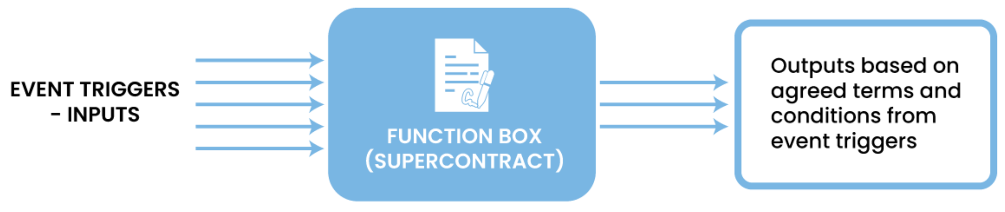

EVENT TRIGGERS - INPUTS
FUNCTION BOX (SUPERCONTRACT)
Outputs based on agreed terms and conditions from event triggers

Figure 23: A Supercontract.

A Creator can use SDKs available in multiple common coding languages (e.g., Javascript, Golang, C++, Rust) to create a Supercontract.

Depending on their design, Supercontracts run when called by a Caller or automatically upon being triggered by the occurrence of an event. These events could be a date (e.g., when interest is due) or the occurrence of an agreed action (e.g., upon delivery or a result), or an oracle feed signaling a trigger action (e.g., price value), or simply upon a transaction being sent into a specific account.

6.3. Contract Deployment

Deployment steps:

i. Deployment Request

To deploy a Supercontract, the Creator creates a storage Drive of an adequate size to host its code. At this point in the process, the Creator is a Drive Owner.

The Creator then posts a **DeployTransaction**, which triggers the Drive's Replicators to add the deployment to the storage Modification Queue. When the transaction is next in line, the following occurs:

- Generation of an identifier for the Supercontract named a Supercontract Public Key (equals the hash of the Drive’s Public Key) so that the Validators can associate the Drive with the Supercontract and temporarily ignore any Drive modification and closure transactions.
- Replicators run the Supercontract Installer, a WebAssembly (Wasm) function, to verify that the Supercontract code can run.
- If present, the Replicators check the validity of any auto-run file that automatically runs a Supercontract upon the occurrence of an event.

ii. Successful Deployment

If the Replicators find no issues with the Supercontract, the Replicators notify the network of the result via an **EndDeployTransaction** that triggers the following:

- Replicators upload the Supercontract code to the Drive.
- Replicators become the Supercontract's Executors and start listening to all corresponding transaction triggers.
- The Replicators store any auto-launch file in their memory and listen for the auto-launch transaction triggers.
- Validators permanently associate transactions for the Drive with the Supercontract and ignore any **DataModificationTransactions** and **DriveClosureTransactions**.
- The Supercontract will only stop working if it runs out of Storage Units, which any network user can top up. This feature allows for the Supercontract to run perpetually if the deploying party is no more in existence or no more interested in maintaining the Supercontract.

iii. Unsuccessful Deployment

If the **EndDeployTransaction** notifies the network of any deployment failures, then the following occurs:

- The Replicators stop the deployment and run the Drive in a standard mode for storage only.
- The Validators no longer associate the Supercontract with the Drive.
- The Validators return any unused funds to the Drive Owner.
- The Drive Owner can fix any failures or close the Drive.

6.4. Contract Execution

<u>Execution steps:</u>

i. Initiate Execution

Any user can become a Supercontract Caller. A Caller can:

1. Execute Supercontract functions.
2. Explore execution results from the Drive via a Download Channel.
3. Explore what calls the Executors have run and their status (success or fail).
4. Explore how many Supercontract Service Units (SC Units) Callers have spent.
5. Support the Supercontract Drive as a sponsor by topping up its account with Storage Units (SO units).

The Caller pays SC Units to execute a Supercontract and Streaming Units (SM Units) if the execution requires Executors to download data to the virtual machine from the Internet.

The Caller runs a specific Supercontract function by posting a **StartExecuteTransaction** containing the following information:

- Supercontract identifier.
- The name of the called function.

- Input parameters for this function.
- The type and number of units provided to make the call.

ii. Execution Queue

Executors create an Execution Queue for each Supercontract, where they place all received calls, whether manual or automated, according to the order they appear on the blockchain.

To ensure the Execution Queue does not slow down execution, especially for popular Supercontracts, execution occurs in sequential batches. Executors add all **StartExecuteTransactions** in a new blockchain block as a batch in the queue.

If an **EndExecuteTransaction** is received, the Executors remove the corresponding call request from its batch in the queue.

iii. Running the Code

When a batch is next in line in the Execution Queue, the Executors run each call within the batch in their respective sandboxes to establish a batch execution result.

Each Executor runs the Supercontract's code dynamically using a virtual machine capable of Wasm. A Supercontract call execution is processing a specific function of the Wasm code in a virtual machine. The Wasm code cannot directly interact with an external environment, such as the Internet, without using the virtual machine that enables contract call functions.

A Supercontract code execution should always follow a set sequence, meaning each Executor should deliver the same result. If there has been an incorrect execution, a rollback occurs using stored sandbox data to correct the error.

There are several reasons why a Supercontract execution may fail:

1. The Wasm file is corrupted or absent.
2. There are not enough SC Units for contract execution.
3. There is not enough space in the Supercontract Drive to store files due to insufficient SO Units in the Drive account.
4. There are not enough SM Units for downloading data from the Internet if required.

iv. Collective Decision-making

Once at least ⅔ plus 1 Executors arrive at a consensus via a group decision-making protocol on the batch execution result, they sign a Multi-signature **EndBatchExecuteTransaction** consisting of the corresponding collective opinions on whether the batch execution was successful or not.

Executors that miss out on signing the Multi-signature transaction need to catch up by signing an **EndBatchExecuteSingleTransaction** or synchronize with the other Executors (see section 6.6.).

If the protocol does not achieve at least a ⅔ plus 1 consensus for reasons such as Executors being offline, the batch times out, and the Executors sign an **EndBatchExecuteTransaction** to record the failure. The Supercontract Owner sets the decision-making time limit during the creation of the Supercontract.

v. Execution Results

The **EndBatchExecuteTransaction** contains:

- A reference to the transactions that initiated the calls within the batch.
- Executors' opinions on whether the batch execution was a success or failure.
- The Drive's new state hash.
- Data needed for network rewards for work done.

Supercontract executions create changes to the Supercontract's Drive State (additions, deletions, modifications).

6.5. Configurable Batch Executions

For network efficiency, Supercontract Executors post an **EndBatchExecuteTransaction** to record the execution results of a batch of calls rather than that of each individual call. A batch consists of all calls contained in a blockchain block. The drawback is that if one call in the batch fails, the Executors mark the whole batch as failed - "one bad apple can spoil the bunch."

A Supercontract Owner can solve this potential issue by predefining upon Supercontract creation how many execution calls in a blockchain block form a batch:

1. <u>All calls in a block:</u>
   This is the network default, suitable for stable and basic Supercontracts in regular use that rarely produce execution failures.

2. <u>Some calls in a block:</u>
   A specified maximum number of calls from a batch to reduce the likelihood of a batch execution failure.

3. <u>One call at a time:</u>
   Here, Executors will need to post an **EndBatchExecuteTransaction** for each call within a blockchain block, useful for Supercontracts prone to execution failures such as those dependent on Internet download data or Supercontracts that Callers rarely use.

A Caller can pay more to elect for a call result to be recorded on the blockchain individually and not as a batch, reducing the chances of an execution failure result.

Executors can also change the batch size dynamically, reducing the number of calls if the previous batch was marked as failed.

6.6. Late Execution

An Executor that has not signed an **EndBatchExecuteTransaction(s)** on time (e.g., due to going offline) has two options to catch up:

1. If the Executor obtains the same execution results:
   After executing each missed batch one by one off-chain and achieving the same results shown in the **EndBatchExecuteTransactions**, the Executor can post an **EndBatchExecutationSingleTransaction**. The Executor will receive rewards for all the missed batches except those that are more than an hour old as the protocol will consider the Executor too far behind. The Executor will need to cover the transaction fee for the single transaction.

2. If the Executor does not obtain the same execution results:
   Suppose the Executor does not receive the same results shown in an **EndBatchExecuteTransactions** after executing each missed batch (e.g., due to different conditions at the time of execution). In that case, the Executor will need to synchronize its Drive State with that of other Executors. The Executor will receive no rewards.

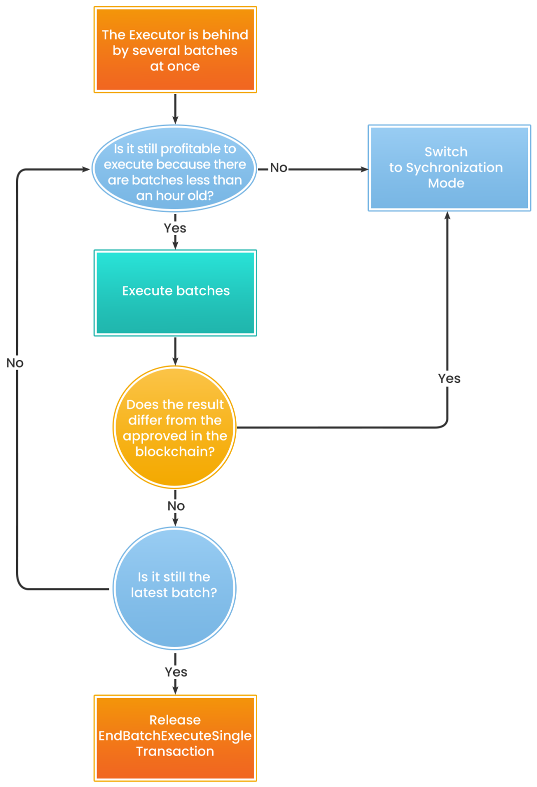

The Executor is behind by several batches at once
Is it still profitable to execute because there are batches less than an hour old?
No
Switch to Sychronization Mode
Yes
Execute batches
No
Yes
Does the result differ from the approved in the blockchain?
No
Is it still the latest batch?
Yes
Release EndBatchExecuteSingle Transaction

Figure 24: Late execution and synchronization.

If an Executor consecutively misses the collective group decision-making and signing of the Multi-signature transaction, Validators remove the non-performing Executor from the Supercontract Drive and confiscate its deposit.

6.7. Proof of Execution

Executors need to prove to Validators that they have executed a Supercontract Batch through ProximaX's Proof of Execution (PoEx) protocol that protects the network from non-performing Executors attempting to claim rewards for work not done.

The protocol makes an Executor prove Supercontract execution through key-pair cryptography using an elliptic curve created using a Supercontract's execution data.

Executors must prove execution of the latest batch to Validators via the protocol within an hour of an **EndBatchExecuteTransactiontion** to receive rewards, encouraging Executors to execute on time and remain up-to-date.

6.8. Aggregate Contract Executions

Supercontracts can run nested or cascading Supercontracts if they contain the necessary call parameters and funds to execute, creating Aggregate Executions.

For example, a Caller runs Supercontract #1 with a **StartExecuteTransaction**. The Executors process this call, releasing a **ReleasedTransactionsTransaction** that contains a new **StartExecuteTransaction** to run Supercontract #2.

The execution of a Supercontract can also release inbuilt automated transactions, more specifically, aggregate **ReleasedTransactionsTransaction**.

Supercontract #1 will make payment for running Supercontract #2 using additional funds from the initial Caller.

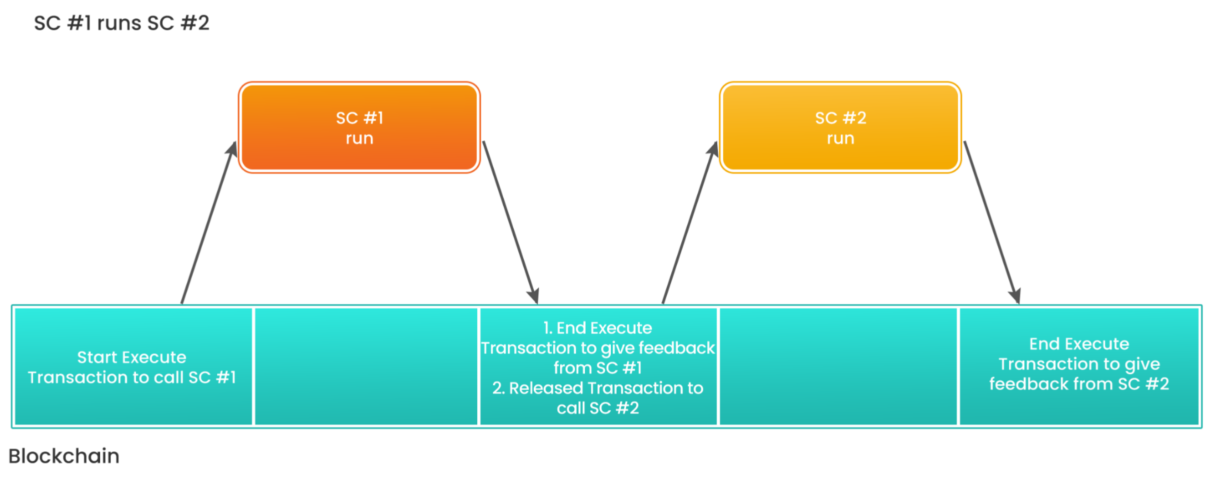

SC #1 runs SC #2
SC #1 run
SC #2 run
Start Execute Transaction to call SC #1
1. End Execute Transaction to give feedback from SC #1
2. Released Transaction to call SC #2
End Execute Transaction to give feedback from SC #2
Blockchain

Figure 25: Aggregate Supercontract executions.

6.9. Data Download from other Drives

Executors can download the data needed for running a Supercontract from the Internet and other Drives on the network.

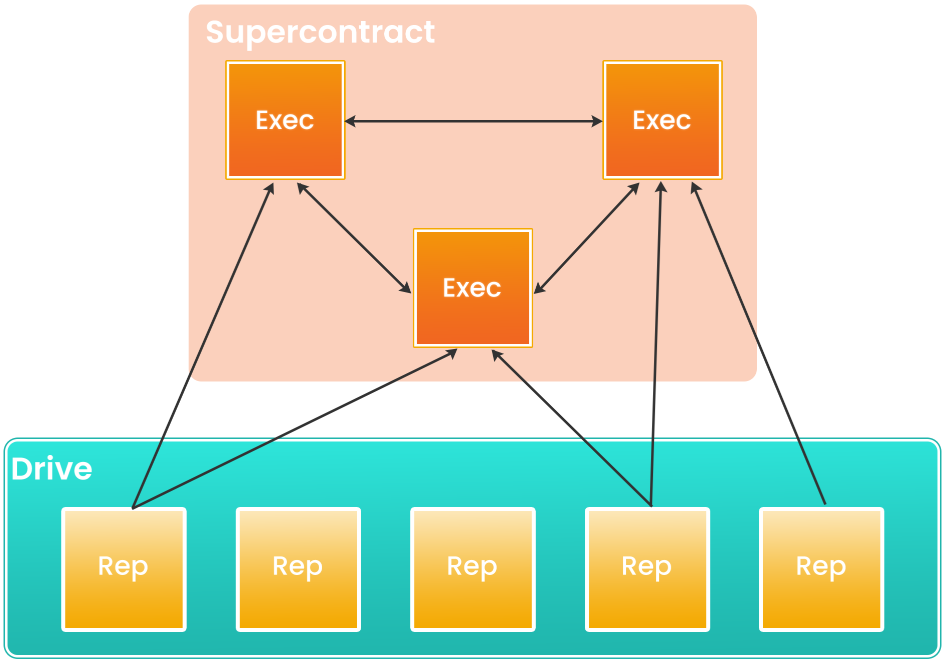


Figure 26: Supercontract data download from another Drive.

Executors can synchronize with Replicators and Executors that have already downloaded the data. Executors receive extra payments to cover the cost of downloading data from other Drives. Executors share opinions to arrive at a consensus on expenses, work done, and compensation to be received.

6.10. Contract Closure

The Supercontract stops functioning once the Supercontract Drive account runs out of Storage Units (SO Units). Anyone can top up the Supercontract Drive account with SO Units to keep the Supercontract running.

A Creator cannot close a Supercontract Drive with a **DriveClosureTransaction** as the Validators will ignore the transaction. Neither can a Creator close the Supercontract by transferring SO Units out from the Supercontract account because the Creator does not hold the private keys to the Supercontract Drive account, therefore, cannot transfer funds out.

But the Creator can include during Supercontract creation advanced logic in the Supercontract code to block the execution of certain or all functions. The design can include executing a Multi-signature transaction by approved parties or a public vote to stop the functions.

When a Supercontract is closed, Executors cancel all pending executions in the Execution Queue except the first one, as Executors would have already started work on it.

ProximaX has designed Supercontract in this way so that Callers can rely on Supercontracts without worrying that they will be arbitrarily closed by a Creator. For that reason, the Creator is a "Creator" and not the "Owner" of a Supercontract.

6.11. Executor Onboarding & Offboarding

When a new Replicator is assigned by a Validator to a Supercontract Drive to become an Executor, the Replicator must follow the same Sirius Storage Replicator onboarding steps outlined above. The new Replicator first signs a **DataModficationSingleApprovalTransaction** to perform data synchronization. If the Replicator misses an **EndBatchExecuteTransaction**, it must sign an **EndBatchExecuteSingleTransaction** to catch up with the other Executors. The new Replicator then becomes an Executor for the Supercontract Drive. Off-boarding follows the same process as Sirius Storage.

6.12. Supercontract Tokenomics

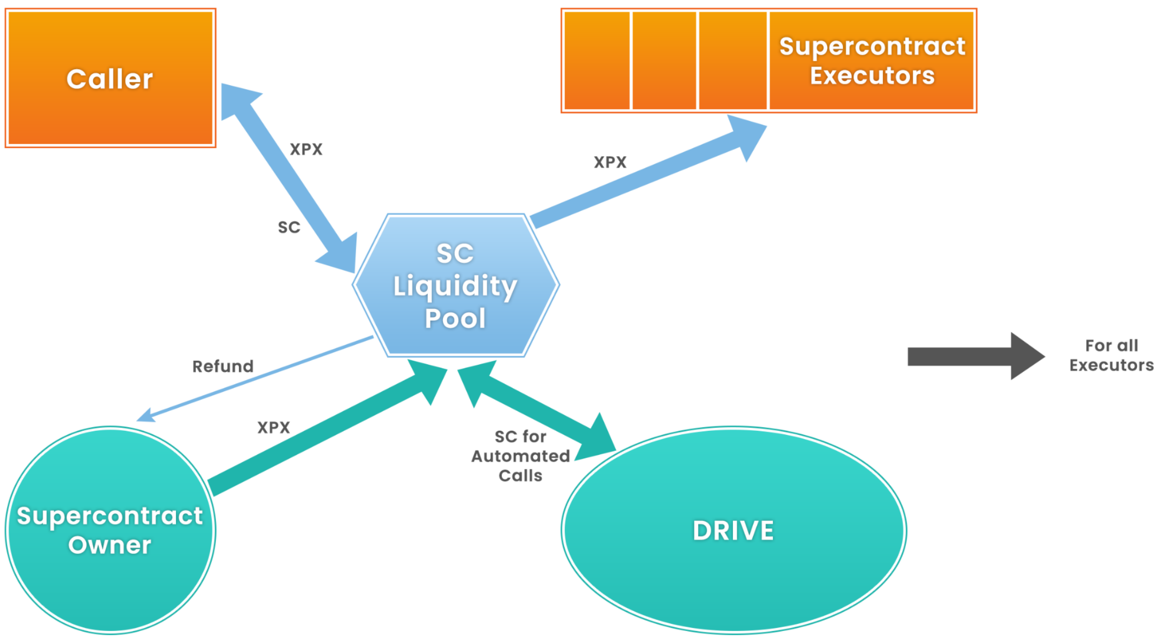


Figure 27: SC Units flow.

A Supercontract Drive requires Storage Units (SO Units) to function.

A Caller pays Supercontract Units (SC Units) to execute a Supercontract and Streaming Units (SM Units) if the execution requires data downloading to the virtual machine via the Internet or another Drive on the network.

1 SC Service Unit corresponds to 1 billion Supercontract operation codes (opcodes), corresponding to one tick in the virtual machine WebAssembly.

According to the latest exchange rate, Validators swap SC and SM Units on behalf of Callers using the Sirius DEX.

When a Caller posts a **StartExecuteTransaction**, the Validators lock the required amounts of SC and SM Units in the Caller's deposit account. After every **EndExecutionTransaction**, Validators issue payments for work done and return unused funds to the Caller.

Validators make payments to Executors regardless of whether their execution results are successful or not. The rewards issued are the median of all Executors' Opinions.

7. Content Review

7.1. Design

Actors

Node:	Description:
Drive Owners	Owns a Drive and instructs Replicators.
Consumers	Downloads data from a Drive.
Global Moderator	Bans content across the network and initiates content removal from all Drives.
Local Moderator	Creates a network warning for content that should be locally banned.

Transactions

Transaction:	Initiate by:	Purpose:
ReviewTransaction	Drive Owners & Content Consumers	Assign Description Tags.
GlobalBanTransaction	Global Moderator	Ban and remove content.
LocalBanTransaction	Local Moderator	Issue warning for content.

7.2. Content Classification

Drive Owners and Content Consumers can select Description Tags from a fixed list to classify Sirius Storage content (e.g., age restriction, genre) and make content searchable.

Application software (e.g., a Google search-type app or a YouTube-type app) can then use the Description Tags to organize and display content accordingly.

7.3. Moderators & Network Bans

Sirius Storage content can also be banned and removed from the network to prevent the distribution of illegal content.

A third party named a Moderator (e.g., a private or government organization) can moderate stored content. There are two types of Moderators:

1. **Global Moderator:** Censors content across the entire network.
2. **Local Moderator:** Censors content for a local jurisdiction.

i. Global Ban

A Global Moderator can identify content based on Content Download Information (CDI) that contains a file's unique hash (like a fingerprint) and Description Tags. Once a Global Moderator bans a file by posting a **GlobalBanTransaction**, the network removes the file from all Drives, and Content Consumers can no longer download it.

The **GlobalBanTransaction** serves as a special modification for all the Drives containing the file as it triggers a free of charge **DataModificationApprovalTransactions** by Replicators of a Drive that stores the banned file.

If a Drive Owner attempts to upload a globally banned file, the Replicators ignore the request and do not save it to the Drive.

ii. Local Ban

A local ban can be executed via a **LocalBanTransaction** by a Global Moderator or a Local Moderator. A local ban creates a warning for the content users and does not trigger the automatic removal of the content from Drives.

Upon being notified of the local ban:

1. **Drive Owner:** Can remove the file.
2. **Replicator:** Can offboard from the Drive.
3. **Content Consumer:** Can cancel downland.

Developers can pre-programmed all these actions in application software.

iii. Supercontract Moderator

A Supercontract can be used to automate the role of a Moderator.

How it can work:

1. The Supercontract detects Description Tags or complaints submitted via the Internet of banned content.
2. The Supercontract checks whether the hash of the file matches banned content.
3. If the content is banned, the Supercontract releases the corresponding ban transaction.

7.4. Content Review Tokenomics

Review Unit (RW Unit) is the Service Unit used for the Content Review layer. Validators send to Drive Owners and Content Consumers 1 RW Unit for every SM Unit spent. Platform users can only use RW Units to classify content.

Process flow:

1. A Drive Owner or Content Consumer posts a **ReviewTransaction**.
2. The transaction has attached single or multiple Description Tags that the network can apply to one or multiple files.
3. The transaction also specifies how many RW Units the network is to assign for each Description Tag. The more RW Units assigned to a Description Tag, the more weight the tag carries for classification.

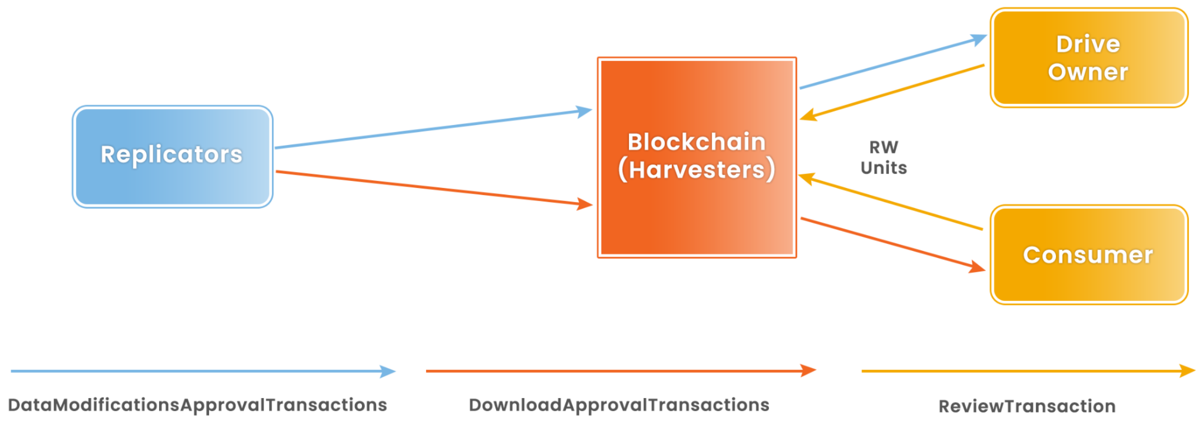


Figure 28: RW Units flow.

Glossary

Aggregate Bonded Transaction	An aggregate transaction merges multiple transactions into one, allowing trust-less swaps, escrows, and other advanced logic.
Aggregate Executions	An aggregate transaction that merges multiple Supercontract execution transactions into one.
Automated Market Making	The automated trading of XPX with Service Units using Liquidity Pools.
Batch Execution	Execution of all execution calls contained in a blockchain block.
Billing Period	Every 24 hours for content Consumers and Receivers, and every four weeks for Drive Owners.
Block Proposer	A Harvester that proposes a new block for the Committee’s vote.
Caller	A node that calls a Supercontract function.
Committee	Harvesters that form a group and cast votes on whether a Block Proposer's block is appended to the blockchain.
Committee Member	A Harvester that is a member of the Committee.
Content Download Information or CDI	Information that is tagged to a file, that includes its hash for file discovery and download.
Content Review	The Sirius platform’s decentralized service layer for content review.
Consumer	A node that downloads data from a storage Drive.
Creator	A node that creates a Supercontract.
DApp or Decentralized Application	An application (App) that runs on a decentralized network.
Description Tag	A label used to classify content in the Content Review layer.
DEX or Decentralized Exchange	A peer-to-peer exchange that does not require an operator or intermediaries.
Distributor	A Replicator that distributes a data stream.
Download Channel	A channel that a Consumer or Receiver opens with a Drive to download data.
Drive	A decentralized data store.
Drive Account	The account assigned to a Drive where the Drive Owner deposits funds used to pay for core services.

Drive Owner	Creator and owner of a Drive.
Drive Public Key	A Drive’s public identifier created using asymmetric cryptography.
Drive State	A hash (i.e., fingerprint) of all the Drive’s data.
Executor	A node that executes a Supercontract’s functions.
Fast Finality	A weighted voting mechanism to reach a consensus on appending new blocks to the chain.
File System	Controls how data is stored and retrieved from a Drive.
Global Moderator	Bans content across the network and initiates content removal from all Drives.
Greed Value	The value assigned to a Harvester by the Proof of Greed algorithm based on the accepted fees from the Harvester's last produced block.
Harvester	A Harvester is a Validator that participates in block production.
Liquidity Pool	A pool of assets in the Sirius DEX used to create a market for the exchange of XPX for Service Units.
Liquidity Provider	A platform user that deposits XPX into the Sirius DEX Liquidity Pool to earn exchange fees.
Local Moderator	Creates a network warning for content that should be locally banned.
Metadata	On-chain data attached to transactions, Accounts, Namespaces, and tokens.
Modification Queue	A queue where data modification requests are added for processing.
Multi-level Multi-signature	Multiple levels of agreements between cosignatories, making it useful for comprehensive approval processes.
Multi-signature or Multisig	A cosignatory agreement between account signatories (e.g., how many need to sign to execute a transaction or remove a signatory).
Moderator	A third party (e.g., a private or government organization) that moderates content in the network.
Namespace	A unique on-chain name, like an internet domain name, that you can link to an Account or a digital asset.
Opinions	Opinions by Replicators, Distributors, Redistributors, and Executors on Receipts collected, work done, verification results, and rewards to be received.
Proof of Execution or PoEx	A Supercontract verification protocol that protects the network from non-performing Executors attempting to claim rewards for work not done.

Proof of Greed or PoG	A Sirius Chain reputation consensus algorithm that keeps transaction fees closer to their actual cost and prevents Harvesters from becoming greedy.
Proof of Stake or PoS	A Sirius Chain consensus algorithm used to select the next block producer, giving preference to Harvesters with a high stake as well as considering reliability and work activity.
Receipts	Off-chain pieces of data produced by storage Consumers and stream Receivers stating what services were consumed.
Receiver	A node that receives a live stream (viewer).
Redistributors	Stream Receivers that contribute their bandwidth and facilitate downstream redistribution to earn rewards.
Replicator	A node that stores and modifies the Drive and performs downloads and uploads based on incoming transactions.
RW Unit or Content Review Unit	A Service Unit where 1 RW corresponds to 1 XPX worth of paid content.
Sandbox	Additional Drive space used by Replicators, Distributors, and Executors to test the provision of services in an isolated environment and for loop recording during live streaming.
SC Unit or Supercontract Unit	A Service Unit where 1 SC token corresponds to 1 billion Supercontract operation codes (opcodes).
SDA or Sirius Digital Asset	Digital assets that any user can create on the platform.
Service Unit	Internal tokens used as units of measure for the provision of services.
Sirius Chain	The ProximaX Sirius platform’s blockchain.
Sirius DEX	The Sirius platform’s decentralized exchange used for the automated exchange of XPX for Service Units.
Sirius Storage	The Sirius platform’s decentralized storage service layer.
Sirius Stream	The Sirius platform’s decentralized streaming service layer.
SM Unit or Streaming Unit	A Service Unit where 1 SM token corresponds to 1 streamed GB.
SO Unit or Storage Unit	A Service Unit where 1 SO token corresponds to 1 GB of space stored for a period of four weeks.
Sponsor	A platform user that Sponsors the provision of storage and streaming services.
Stake	An amount of XPX owned by a Harvester to increase its chances of being selected for block production by the Proof of Stake algorithm.
Stream Distributor	Replicator that distributes a data stream based on incoming transactions.

Stream Sender	Creator of the data stream that instructs Distributors.
Supercontract	The Sirius platform’s decentralized contract service layer.
Validator	All platform nodes serve as a Validator. Validators validate and process all service transactions in a blockchain block (e.g., make payments; assign and remove nodes from services).
Virtual Drive	Off-chain Drive used for anonymous access to services.
XPX	The Sirius platform’s native coin used to pay for platform services.