# Blockchain Oracles

Concepts and Constructions

# Oracle Problem

The blockchain oracle problem outlines a fundamental limitation of smart contracts—they cannot inherently interact with data and systems existing outside their native blockchain environment.

Resources external to the blockchain are considered “off-chain,” while data already stored on the blockchain is considered on-chain.

By being purposely isolated from external systems, blockchains obtain their most valuable properties like strong consensus on the validity of user transactions, prevention of double-spending attacks, and mitigation of network downtime.

# Taxonomy of Oracles

Software v/s hardware v/s human oracles

Centralised v/s decentralised oracles

Inbound v/s outbound oracles

Compute enabled oracles

Cross chain oracles

# Software Oracles

Software oracles (also known as deterministic oracles) interact with online sources of information and transmit it to the blockchain. This information can come from online databases, servers, websites, essentially, any data source on the Web.

The fact that software oracles are connected to the Internet not only allows them to supply information to smart contracts but also to transmit that information in real-time.

This makes them one of the most common types of blockchain oracles. Information typically provided by software oracles can include exchange rates, digital asset prices, real-time flight information, or any other information we need it.

# Hardware Oracles

Some smart contracts need to interface with the real world. Hardware oracles are designed to get information from the physical world and make it available to smart contracts.

Such information could be relayed from electronic sensors, IoT, barcode/QR scanners, RFID tags, robot, and other information reading devices.

A hardware oracle essentially “translates” real-world events into digital values that can be understood by smart contracts.

# Human Oracles

Sometimes individuals with specialized knowledge/skills in a particular field can also serve as oracles.

They can research and verify the authenticity of information from various sources and translate that information to smart contracts.

Since human oracles can verify their identity using cryptography, the possibility of a fraudster faking their identity and providing corrupted data is relatively very low.

# Computation Oracles

Rather than just relaying the results of a query, computation oracles can be used to perform computation on a set of inputs and return a calculated result that may have been infeasible to calculate on-chain.

For example, one might use a computation oracle to perform a computationally intensive regression calculation in order to estimate the yield of a bond contract.

# Inbound and Outbound Oracle

Inbound oracles transmit information from external sources to smart contracts, while outbound oracles send information from smart contracts to the external world. An example of an inbound oracle is one that tells a smart contract what the temperature is measured by a sensor.

An example of an outbound oracle can be considered with a smart lock. If funds are deposited to an address, the smart contract sends this information through an outbound oracle to a mechanism that unlocks the smart lock.

# Categories of Inbound Oracles

Pull-based inbound Oracle — When a smart contract needs data, it requests the data from the oracle. After the oracle receives the request, it gathers the information from the off-chain data source and provides it back to the blockchain.

Push-based inbound Oracle — Oracle watches for any sort of changes in a particular off-chain data source. When there is a change in the data source, the changes will be transmitted to the blockchain.

# Categories of Outbound Oracles

Pull-based outbound oracle — When an off-chain resource needs to query data from the blockchain, it requests the data from the on-chain data source. After receiving the request, it provides the data.

Push-based outbound oracle — The smart contract watches for any changes in the blockchain data set. When there is a change, it will be communicated to the off-chain resource.

# Consensus based oracles

In contrast to software oracles, consensus-based oracles do not use a single source. There are several ways to create and use decentralized oracles. One would be a rating system inside a prediction market.

To reduce risk and provide more security, a combination of oracles might be used. For example you could take the average of 5 oracles. Or, 5 out of 7 oracles can determine the outcome of an event

# Centralised Oracles

Blockchain oracle mechanisms using a centralized entity to deliver data to a smart contract introduce a single point of failure, defeating the entire purpose of a decentralized blockchain application.

If the single oracle goes offline, then the smart contract will not have access to the data required for execution or will execute improperly based on stale data.

Even worse, if the single oracle is corrupted, then the data being delivered on-chain may be highly incorrect and lead to smart contracts executing very wrong outcomes

# Input Oracles

The most widely recognized type of oracle today is known as an “input oracle,” which fetches data from the real-world (off-chain) and delivers it onto a blockchain network for smart contract consumption.

These types of oracles are used to power Chainlink Price Feeds, providing DeFi smart contracts with on-chain access to financial market data.

# Output Oracles

The opposite of input oracles are “output oracles,” which allow smart contracts to send commands to off-chain systems that trigger them to execute certain actions.

This can include informing a banking network to make a payment, telling a storage provider to store the supplied data, or pinging an IoT system to unlock a car door once the on-chain rental payment is made.

# Cross Chain Oracles

Another type of oracle are cross-chain oracles that can read and write information between different blockchains.

Cross-chain oracles enable interoperability for moving both data and assets between blockchains, such as using data on one blockchain to trigger an action on another or bridging assets cross-chain so they can be used outside the native blockchain they were issued on.

# Compute Enabled Oracles

A new type of oracle becoming more widely used by smart contract applications are “compute-enabled oracles,” which use secure off-chain computation to provide decentralized services that are impractical to do on-chain due to technical, legal, or financial constraints.

This can include using Chainlink Automation to trigger the running of smart contracts when predefined events take place, computing zero-knowledge proofs to generate data privacy, or running a verifiable randomness function to provide a tamper-proof and provably fair source of randomness to smart contracts.

# Oracle types based on the frequency of extracting data

Immediate Read

Publish Subscribe

Request Response

# Immediate Read Oracles

Oracles that allow getting data which will be used to take immediate decisions. This type of oracle stores the data in the smart contract storage. The data will be updated from time to time. Since it uses contract storage, only the data that is highly relevant for the use case will be stored.

Example:

A smart contract which is used by a book store can use an oracle to provide the ISBN of a specific book. In this case, the oracle only stores the name and the ISBN of the book inside the contract storage since it is excessive to store attributes such as author name, year etc.

# Publish Subscribe Oracle

Oracles which provide data upon subscribing to the oracle belong to this pattern. It provides a broadcast service of the data to the smart contracts. The data will be provided to the smart contract via polling from the smart contract or running a worker off-chain and watching for updates in the oracle.

Example:

Let’s imagine that there is an exchange in the decentralized world, which will allow us to buy stocks. The smart contract of this exchange should know the latest price of a specific stock or all the stocks. The smart contract can then subscribe to an oracle which will provide the prices. From the side of the smart contract, it can either poll the oracle or the oracle can push the updates to all of its clients.

# Request Response Oracles

This pattern is similar to the client-server architecture where a request is sent from the client and is processed by the server.

The data relating to this oracle might be stored in an external infrastructure because it uses a larger dataset that cannot be stored in smart contract storage.

Because of this scenario and the necessity for extra performance, these types of oracles use off-chain infrastructure such as servers.

# Taxonomy of Oracles

By data sources, oracles are categorized mainly into software and hardware source ones in, as well as human source ones. By data transmission direction, oracles are categorized into inbound and outbound ones. Inbound oracles transmit data from off-chain side to on-chain side, and vice versa for outbound oracles.

Further, by data transmission approach, oracles can be categorized into pulling and pushing based ones. Pulling based oracles are executed at data requester side while pushing oracles are executed on data provider side.

Oracles can be categorised into centralized and consensus ones by their degree of centralization of data feeds.  Oracles are also categorised into voting-based and reputation based ones and their respective detailed divisions of data validation mechanisms.

Oracles are also categorized into prediction markets, centralized data feeds and oracle networks. Oracles are also divided data on-chaining into TLS-based, enclave-based and voting-based types.

# Decentralised Oracle Networks

A computer network made up of nodes. These nodes communicate and operate as peers in compliance with an agreed protocol to acquire knowledge that is external to the network.

These nodes verify the veracity of the acquired information and supply such verified information to other applications or networks that may need it

Thus a decentralised oracle network (DON) is capable of processing retrieve attest and deliver requests.

# Incentive model in a decentralised oracle network

In order to incentivize people to run nodes in a DON, the network can implement some form of native token that the participant sending a RAD request can use to reward the rest of participants for their work.

# Decentralised Oracle Networks - Possibilities

To reduce the computational, energetic and monetary cost of performing RAD tasks, a DON can include some kind of sharding feature that makes it possible to assign a RAD task only to a fraction of the participants.

A DON can implement a reputation scoring system that give participants different chances of being assigned tasks depending on their past performance in terms of honesty.

# Decentralised Oracle Networks

Decentralized oracle networks (DONs) enable the creation of hybrid smart contracts, where on-chain code and off-chain infrastructure are combined to support advanced decentralized applications (dApps) that react to real-world events and interoperate with traditional systems.

# Hybrid Smart Contracts

Hybrid smart contracts combine code running on the blockchain (on-chain) with data and computation from outside the blockchain (off-chain) provided by Decentralized Oracle Networks.

Hybrid smart contracts enable advanced forms of economic and social coordination that have the tamper-proof and immutable properties of blockchains yet leverage secure off-chain oracle services to attain new capabilities, such as scalability, confidentiality, order fairness, and connectivity to any real-world data source or system.

# Oracle Use Cases

Major industries benefit from combining oracles and smart contracts are

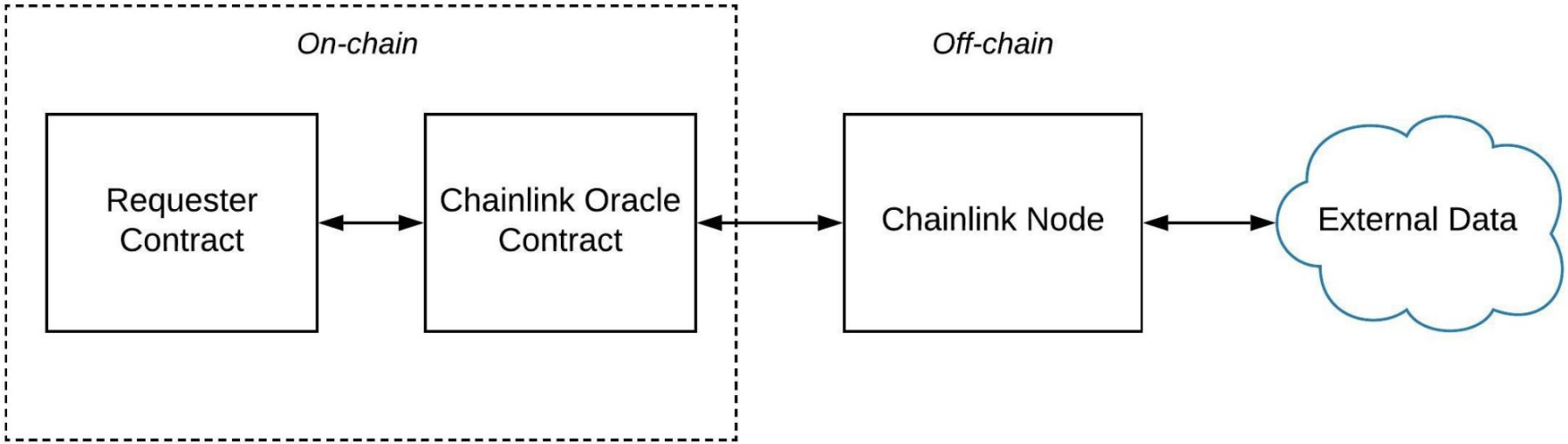
- Asset prices for finance,
- Weather information for insurance,
- Randomness for gaming,
- IoT sensors for supply chain,
- ID verification for government.

# ChainLink Onchain Architecture

As an oracle service, ChainLink nodes return replies to data requests or queries made by or on behalf of a user contract, which is denoted to as requesting contracts and denote by User Smart Contract.

ChainLink’s on-chain interface to requesting contracts is itself an on-chain contract that is denoted as ChainLink Smart Contract.

# Chainlink Integration Architecture



# Chainlink Integration Workflow

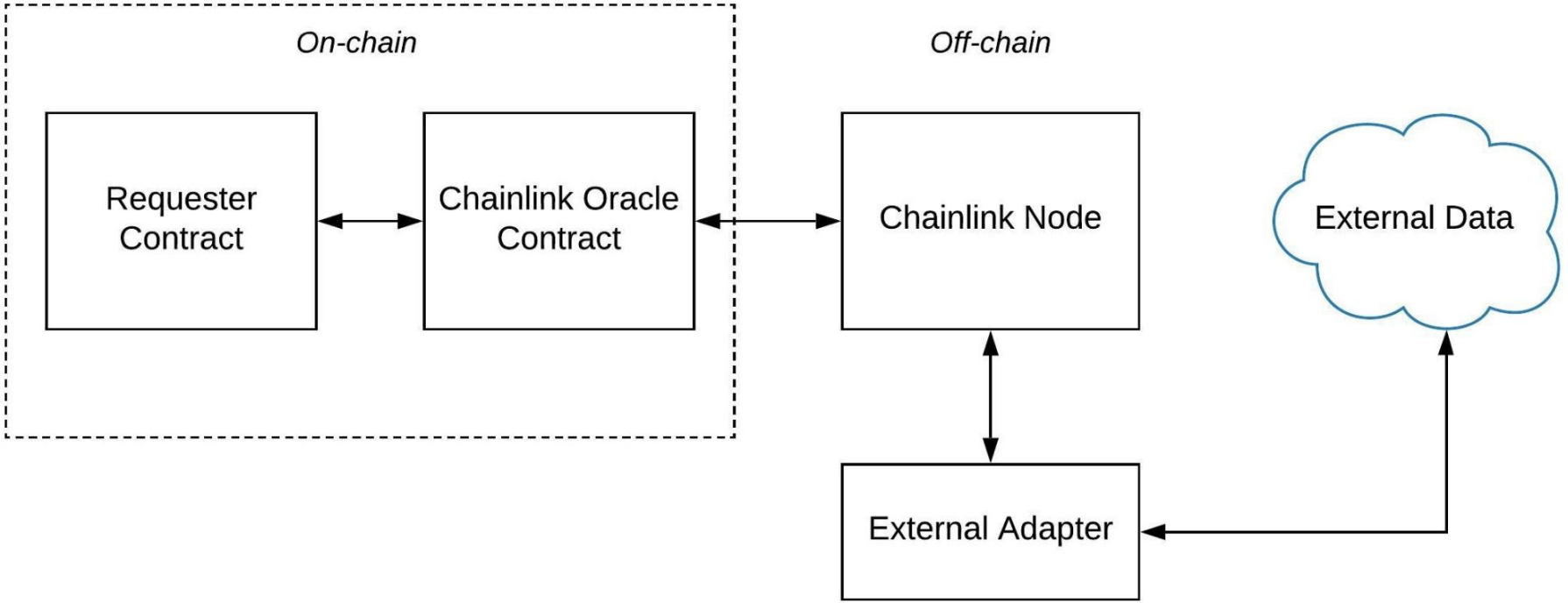
On-chain, a requesting smart contract (Requester) interfaces with a Chainlink oracle contract to make a request for data. The Requester Contract sends LINK tokens and data for the request.

The on-chain oracle contract receives the LINK tokens and request and emits an event to the off-chain Chainlink network.

Off-chain, Chainlink nodes are connected to the Ethereum blockchain and monitoring events. A Chainlink node will pick up an event and perform a request to an external API.

The API returns data, which the Chainlink node returns back to the on-chain oracle. The oracle aggregates the responses and passes them back as a single response to the requesting contract.

# Chainlink Integration Architecture



# Chainlink Integration Workflow

External adapters are not always necessary to use with the Chainlink network, but provide developers with the ability to customize an API response. They are also used with private APIs that require a secret key to be sent along with the request.

External adapters live between the Chainlink node and an external API and communicate with a simple JSON specification. They can be written in any language.

# ChainLink  Offchain Architecture

ChainLink initially consists of a network of oracle nodes connected to the Ethereum network. These nodes independently harvest responses to off-chain requests.

Their individual responses are aggregated via one of several possible consensus mechanisms into a global response that is returned to a requesting contract USER-SC.

The ChainLink nodes are powered by the standard open source core implementation which handles standard blockchain interactions, scheduling, and connecting with common external resources.

Node operators may choose to add software extensions, known as external adapters, that allow the operators to offer additional specialized off-chain services

# ChainLink Core

The core node software is responsible for interfacing with the blockchain, scheduling, and balancing work across its various external services. Work done by ChainLink nodes is formatted as assignments.

Each assignment is a set of smaller job specifications, known as subtasks, which are processed as a pipeline. Each subtask has a specific operation it performs, before passing its result onto the next subtask, and ultimately reaching a final result.

ChainLink's node software comes with a few subtasks built in, including HTTP requests, JSON parsing, and conversion to various blockchain formats.

# Jobs Pipeline

The core functionality of a single Chainlink node is to run a set of sequential processes, feeding the results of each process into the subsequent process, until a final result is reached. This is known as the "job pipeline."

When creating a job specifcation, you must declare the job's Initiators and the job's Tasks. Initiators are the entry points for the job pipeline.

# Jobs Pipeline

Task specifications define the actual processes that will be run in order to produce a result. Every task spec has a type, and many take additional parameters to specify their behavior.

Each task type is handled by an Adapter which specializes in running the computations associated with each task.

The result of each Adapter's computation is passed on as the input to the next adapter, until the final adapter is reached and its result is the result of the entire Job Run.

# External Adapters

Beyond the built-in subtask types, custom subtasks can be defined by creating adapters. Adapters are external services with a minimal REST API.

By modeling adapters in a service-oriented manner, programs in any programming language can be easily implemented simply by adding a small intermediate API in front of the program.

Similarly, interacting with complicated multi-step APIs can be simplified to individual subtasks with parameters.

# Decentralised Data Model

Data Aggregation

Shared Data Source

Decentralised Oracle Network

- Consumer contract
- Proxy contract
- Aggregator contract

# Dynamics of Decentralised Oracle Network

Each data feed is updated by a decentralized oracle network. Each oracle operator is rewarded for publishing data. The number of oracles contributing to each feed varies.

In order for an update to take place, the data feed aggregator contract must receive responses from a minimum number of oracles or the latest answer will not be updated. You can see the minimum number of oracles for the corresponding feed at data.chain.link.

Each oracle in the set publishes data during an aggregation round. That data is validated and aggregated by a smart contract, which forms the feed's latest and trusted answer.

# Data Aggregation

Each data feed is updated by multiple Chainlink oracle operators.

AccessControlledOffchainAggregator aggregates the data on-chain.

Off-Chain Reporting (OCR) further enhances the aggregation process.

# Consumer Contract

A Consumer contract is any contract that uses Chainlink Data Feeds to consume aggregated data. Consumer contracts must reference the correct AggregatorV3Interface contract and call one of the exposed functions. Off-chain applications can also consume data feeds.

# Proxy Contract

Proxy contracts are on-chain proxies that point to the aggregator for a particular data feed. Using proxies enables the underlying aggregator to be upgraded without any service interruption to consuming contracts.

Proxy contracts can vary from one data feed to another, but the AggregatorProxy.sol contract on Github is a common example.

# Aggregator Contract

An aggregator is the contract that receives periodic data updates from the oracle network. Aggregators store aggregated data on-chain so that consumers can retrieve it and and act upon it within the same transaction. You can access this data using the Data Feed address and the AggregatorV3Interface contract.

Aggregators receive updates from the oracle network only when the Deviation Threshold or Heartbeat Threshold triggers an update during an aggregation round. The first condition that is met triggers an update to the data.

# Aggregator Contract Triggers

- Deviation Threshold: A new aggregation round starts when a node identifies that the off-chain values deviate by more than the defined deviation threshold from the on-chain value. Individual nodes monitor one or more data providers for each feed.
- Heartbeat Threshold: A new aggregation round starts after a specified amount of time from the last update.

# Off Chain Reporting

For Off-Chain Reporting aggregators, all nodes communicate using a peer to peer network. During the communication process, a lightweight consensus algorithm runs where each node reports its data observation and signs it.

A single aggregate transaction is then transmitted, which saves a significant amount of gas. The report contained in the aggregate transaction is signed by a quorum of oracles and contains all oracles' observations. This report is validated on-chain.

# Data Feeds

Price Feeds

Proof of Reserve Feeds

NFT Floor Pricing Feeds

L2 Sequencer uptime feeds

# Price Feeds

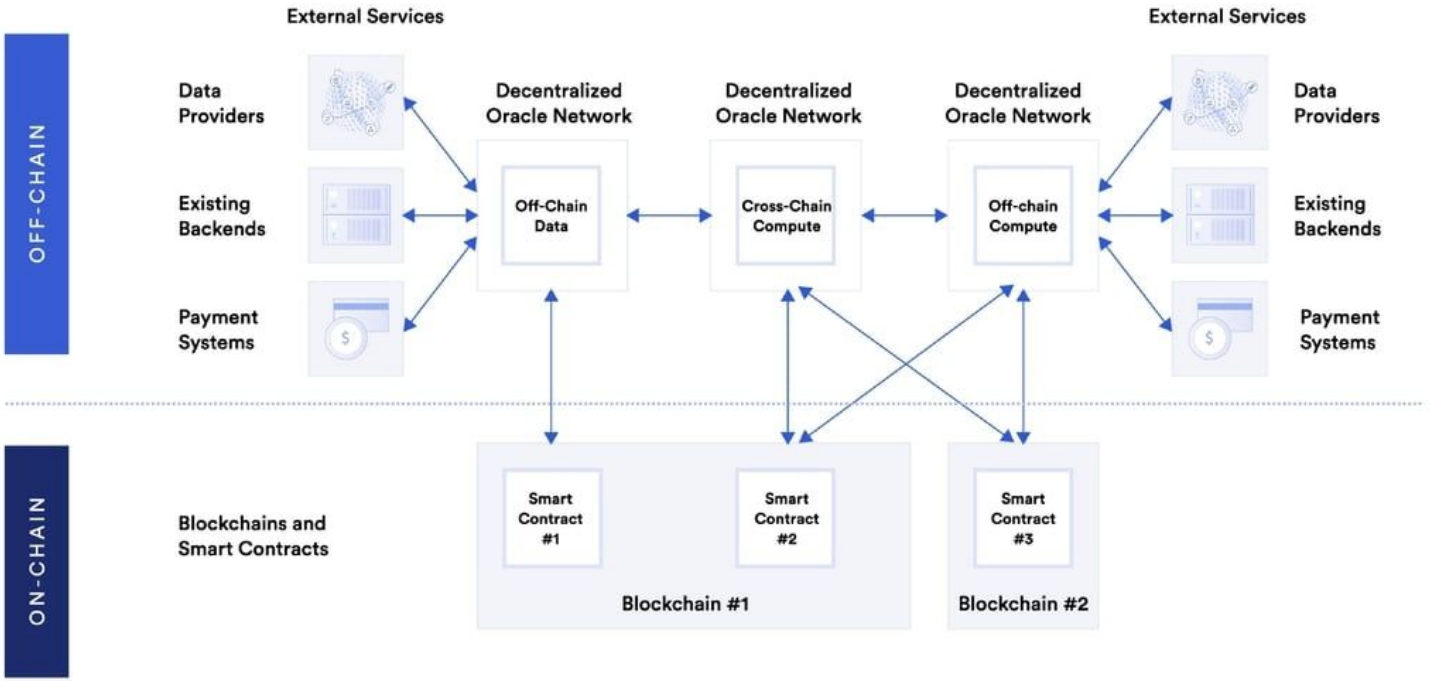
Chainlink Price Feeds, incorporate three layers of decentralization—at the data source, individual node operator, and oracle network levels—to eliminate any single point of failure.

For example, Synthetix uses Data Feeds to determine prices on their derivatives platform. Lending and borrowing platforms like AAVE use Data Feeds to ensure the total value of the collateral.

Data Feeds aggregate many data sources and publish them on-chain using a combination of the Decentralized Data Model and Off-Chain Reporting.

# Chainlink Price Feeds

# Summary of Oracle Models



# Proof of Reserve Feeds

Proof of Reserve Feeds provide the status of the reserves for several assets. We can consume these feeds the same way as Price Feeds.

Reserves are available for both cross-chain assets and off-chain assets. Token issuers prove the reserves for their assets through several different methods.

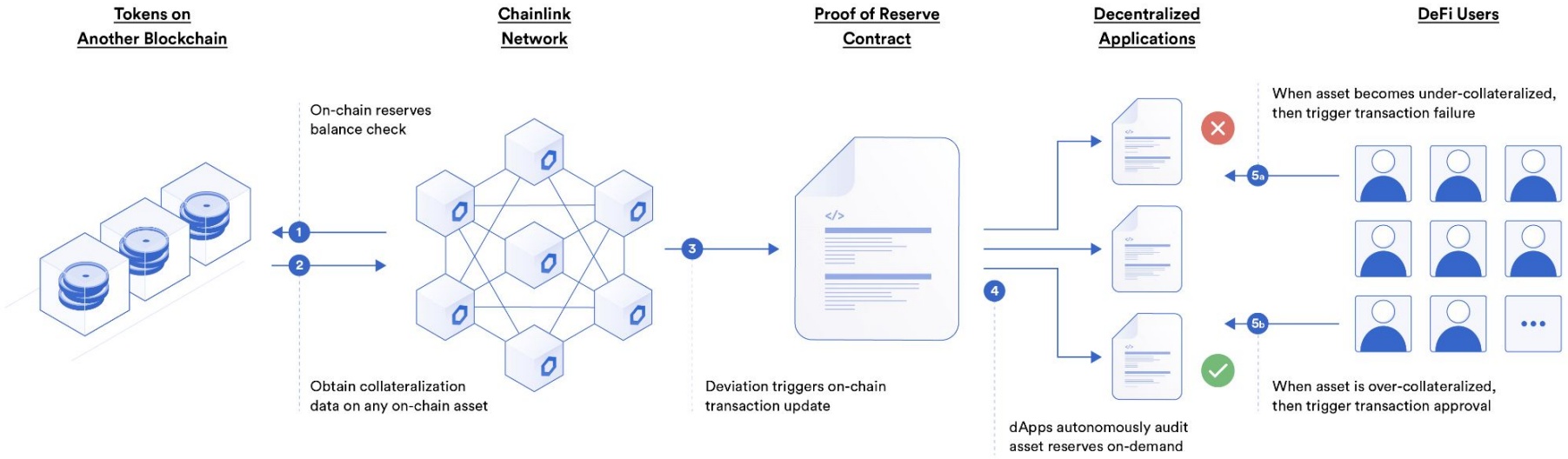
- Cross-chain reserves:
  - Wallet address manager
  - Self-attested wallet API
- Off-chain reserves:
  - Third-party API
  - Custodian API
  - Self-attested API

# Cross Chain Reserves

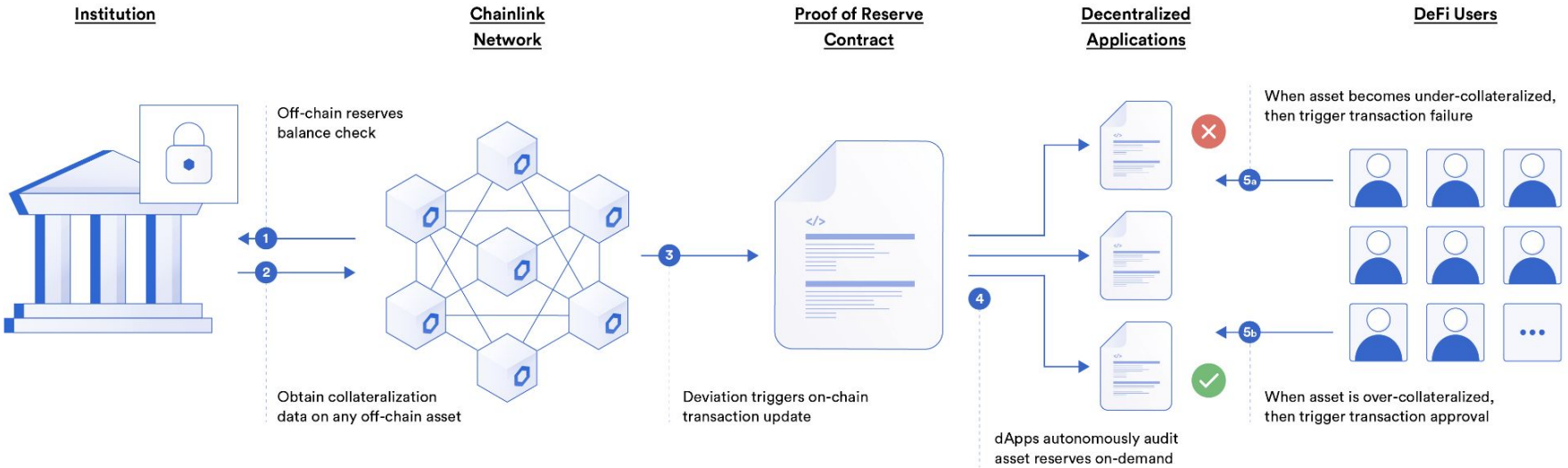
Cross-Chain reserves are sourced from the network where the reserves are held. This includes but is not limited to networks including Bitcoin, Filecoin, Cardano, and chains where Chainlink has a native integration.

Chainlink Node operators can report cross-chain reserves by running an external adapter and querying the source-chain client directly. In some instances, the reserves are composed of a dynamic list of IDs or addresses using a composite adapter.

# Cross Chain Reserve



# Off-chain Reserve



# NFT Floor Pricing Feeds

Chainlink NFT Floor Pricing Feeds provide the lowest price of an NFT in a collection. These feeds operate the same way as other Chainlink Data Feeds.

NFT Floor Pricing Feeds are supported by Coinbase Cloud's aggregation algorithm and Chainlink's oracle infrastructure to help eliminate extreme price outliers and make these feeds resistant to market manipulation.

You can use NFT Floor Pricing Feeds for use cases that rely on high-quality NFT data, including lending and borrowing, on-chain derivatives, dynamic NFTs, gaming guilds, CeFi products, prediction markets, and more.

# L2 Sequencer Uptime Feeds

Optimistic rollup protocols move all execution off the layer 1 (L1) Ethereum chain, complete execution on a layer 2 (L2) chain, and return the results of the L2 execution back to the L1. These protocols have a sequencer that executes and rolls up the L2 transactions by batching multiple transactions into a single transaction.

If a sequencer becomes unavailable, it is impossible to access read/write APIs that consumers are using and applications on the L2 network will be down for most users without interacting directly through the L1 optimistic rollup contracts. The L2 has not stopped, but it would be unfair to continue providing service on your applications when only a few users can use them.

To help your applications identify when the sequencer is unavailable, you can use a data feed that tracks the last known status of the sequencer at a given point in time. This helps you prevent mass liquidations by providing a grace period to allow customers to react to such an event

# Chainlink Smart Contracts - Components

There are three main smart contracts

A Reputation contract

An Order matching contract

Aggregating contract

# ChainLink Smart Contracts

The reputation contract keeps track of oracle-service provider performance.

The order-matching smart contract takes a proposed service level agreement (SLA), logs the SLA parameters, and collects bids from oracle providers. It then selects bids using the reputation contract and finalizes the oracle SLA.

The aggregating contract collects the oracle providers' responses and calculates the final collective result of the ChainLink query. It also feeds oracle provider metrics back into the reputation contract.

# ChainLink Workflow

Oracle selection

Data reporting

Result aggregation

# Oracle selection

An oracle services purchaser specifies requirements that make up a SLA proposal.

The SLA proposal includes details such as query parameters and the number of oracles needed by the purchaser.

Additionally, the purchaser specifies the reputation and aggregating contracts to be used for the rest of the agreement.

# Matching Contract

Manual matching is not possible for all situations. For example, a contract may need to request oracle services dynamically in response to its load.

Automated solutions solve this problem and enhance usability. For these reasons, automated oracle matching is also being proposed by ChainLink through the use of order-matching contracts.

# Matching Contract

Once the purchaser has specified their SLA proposal, instead of contacting the oracles directly, they will submit the SLA to an order-matching contract.

The submission of the proposal to the order-matching contract triggers a log that oracle providers can monitor and filter based on their capabilities and service objectives.

# Result Aggregation

Once the oracles have revealed their results to the oracle contract, their results will be fed to the aggregating contract.

The aggregating contract tallies the collective results and calculates a weighted answer. The validity of each oracle response is then reported to the reputation contract.

# Oracle Reputation and On-chain performance

Reputation in blockchain oracle systems gives users and developers the ability to monitor and filter between oracles based on parameters they deem important.

Oracle reputation is aided by the fact that oracles sign and deliver their data onto an immutable public blockchain ledger, and so their historical performance history can be analyzed and presented to users through interactive dashboards

Reputation frameworks provide transparency into the accuracy and reliability of each oracle network and individual oracle node operator. Users can then make informed decisions about which oracles they want to service their smart contracts.

# Appendix

# Basic Request Model

The client contract that initiates this cycle must create a request with the following items:

- The oracle address.
- The job ID, so the oracle knows which tasks to perform.
- The callback function, which the oracle sends the response to.

# Chainlink Client

ChainlinkClient is a parent contract that enables smart contracts to consume data from oracles.

The client constructs and makes a request to a known Chainlink oracle through the transferAndCall function, implemented by the LINK token.

This request contains encoded information that is required for the cycle to succeed. In the ChainlinkClient contract, this call is initiated with a call to sendChainlinkRequestTo.