# THIS DOCUMENT HAS BEEN MOVED TO THE HIP REPO. GO HERE:

<https://github.com/kaleido-io/firefly-hyperledger-hip>

For further comments on this document, please create GitHub issues

# Hyperledger FireFly Proposal

<https://github.com/kaleido-io/hyperledger-hip/blob/gh-pages/HIPs/0000-template.md>

## HIP Identifier

Hyperledger FireFly

## Sponsors

Steve Cerveny ([steve.cerveny@kaleido.io](mailto:steve.cerveny@kaleido.io))

Peter Broadhurst ([peter.broadhurst@kaleido.io](mailto:peter.broadhurst@kaleido.io))

Gabriel Indik ([gabriel.indik@kaleido.io](mailto:gabriel.indik@kaleido.io))

Nicko Guyer ([nicko.guyer@kaleido.io](mailto:nicko.guyer@kaleido.io))

Jim Zhang ([jim.zhang@kaleido.io](mailto:jim.zhang@kaleido.io))

Guillaume Le Saint ([guillaume@atato.com](mailto:guillaume@atato.com))

Jon Labrie ([jlabrie@greenfence.com](mailto:jlabrie@greenfence.com))

Gari Singh ([gari.r.singh@gmail.com](mailto:gari.r.singh@gmail.com))

Patrick Schmid, PhD ([schmid@theinstitutes.org](mailto:schmid@theinstitutes.org))

David Metcalf, PhD ([dmetcalf@ist.ucf.edu](mailto:dmetcalf@ist.ucf.edu))

*(Note - more sponsors are being added over the next week)*

## Abstract

FireFly is a multiparty system for enterprise data flows, powered by blockchain. It solves all of the layers of complexity that sit between the low level blockchain and high level business processes and user interfaces. FireFly enables developers to build blockchain apps for enterprises up to 100x faster by allowing them to focus on business logic instead of infrastructure.

## Context

Enterprise blockchain emerged in 2015 as a new type of B2B collaboration software, helping companies to exchange data with common business logic in *smart contracts* while having a common record of transactions with a *shared ledger.* Enterprise blockchain use cases see companies organizing into business networks with each company deploying a private copy of the software stack including a blockchain node. The nature of a decentralized application with many independent parties means that sophisticated coordination across the parties is critical for end to end flows to execute smoothly. Multi-party flows necessarily achieve that coordination via event-driven designs including application level messaging and the triggering of actions from blockchain level event streams.

Further, most enterprise use cases require privacy within a given transaction - the exposure of data to a subset of members in the network on a “need to know” basis. This combined with the immutability of the shared ledger means that many use cases center their data flows in off chain interactions.

For these reasons, we have seen that enterprise blockchain use cases require a set of off chain technologies that serve as “plumbing” layers to reliably coordinate and facilitate the multi-party data flows. Rationalizing identity, connectivity and synchronization across these layers becomes a core challenge to building a blockchain project.

We’ve observed that the specific application and middleware components and how they are used have a remarkable number of similarities regardless of the industry, use case, or even blockchain protocol. Kaleido has six years of learnings on these use cases across Fabric, Ethereum, and Corda and has built these components out over a period of several years of active engineering. Hyperledger FireFly is born out of those learnings and engineering work. The code contribution is a direct refactoring of core Kaleido components which are run in production by large enterprises today.

## Dependent Projects

Hyperledger FireFly requires a blockchain protocol to run. Specifically, FireFly’s architecture addresses two critical aspects of decentralized multiparty systems, data immutability and global ordering, by building on top of an underlying blockchain protocol. Support is included for **Hyperledger Fabric**; Enterprise Ethereum variants including Quorum and **Hyperledger Besu**; and Corda.

In addition, Hyperledger FireFly is designed to be very pluggable, and current or future plug points for other technology include:

* IPFS for broadcast data
* NodeRed for low code/no code processing logic
* **Hyperledger Avalon** for verifiable off-chain processing logic
* **Hyperledger Cactus** for cross-chain asset transfers and DvP flows

# High Level Problem Statement

It’s incredibly hard to build a blockchain application and progress the network of participants to production. There are no shortage of examples of blockchain consortia that are years into their project without having the first use case successfully deployed. This is clearly not a good signal for the overall health and vibrancy of the technology sector.

While there are reasons for this slow time horizon that span technical, business, and governance, FireFly aims to eliminate one of the most significant - the “plumbing problem”. At a high level, Enterprise blockchain use cases attempt to coordinate data flows and transactions across disparate, separately owned systems (ERP, mainframe, commerce, etc.). To do this end-to-end, four things are required:

(1) the *single source of truth coordination system* across parties

(2) *per-member* *plumbing software* to enable reliable, private integration to and from the coordination system

(3) *cross-org plumbing software* that provides robust cross-connectivity among members alongside (1) for use when the coordination system is not suitable or not preferred as the data flow mechanism. Examples include sensitive data, low latency messaging, and large data flows

(4) *a control plane* to manage all of these runtime and configuration details

We define a **multi-party system** as the set of technology, infrastructure, and management capability to enable a consortium to build and deploy a use case which includes (1), (2), (3), and (4). We observe nearly all the real world consortia that we worked with over the last six years require (1), (2), (3), and (4) to build and deploy an enterprise blockchain use case. In our experience, consortia spend a huge amount of their resources building out the plumbing portion of the system. Furthermore, it is our observation that companies spend most of their budget on (2) and (3).

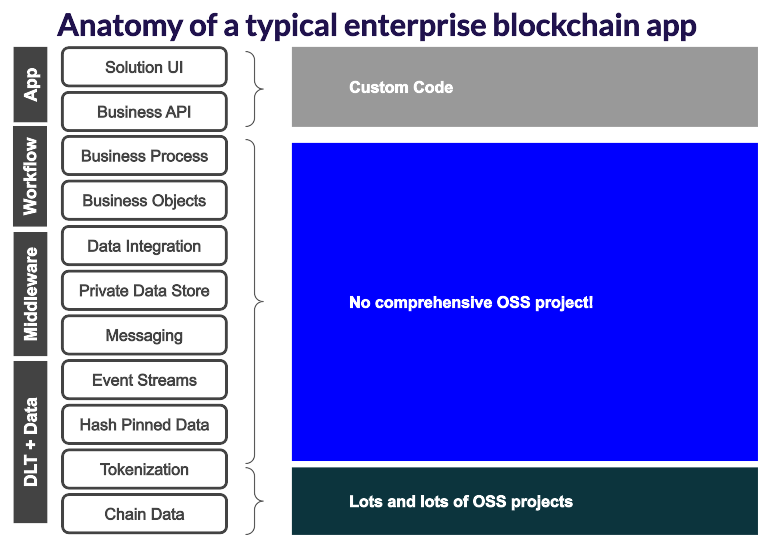
Hyperledger FireFly brings together (1), (2), (3), and (4) to offer a pre-integrated set of runtimes via a highly pluggable framework. Note that FireFly is NOT attempting to replace or invalidate any specific runtime technology, but rather pulls together multiple complementary technologies into a coherent, larger system. Indeed, many of the infrastructure runtimes FireFly relies on are separate open source projects.

Enterprise blockchains are purpose built to deliver (1) with invaluable attributes like *global ordering* and *finality*. FireFly will support Hyperledger Fabric, Enterprise Ethereum (such as Quorum and Besu), and will investigate Corda support. (1) also includes global storage providers such as IPFS. FireFly also provides a significant amount of technology for (2) and (3) including cross-party messaging, cross-party private document transfer, and event propagation. All of these runtimes and connectors into the FireFly node. For (4), Hyperledger FireFly also provides a network map and registry that unifies all of the identity and bindings information needed.

## Motivation

When building a system for multiparty business transactions based on blockchain technology, there are a number of problems that need to be solved. Problems such as:

* How do parties agree on standardized data types?
* How do I securely send private data between two parties in the system without other parties being able to view the data?
* How do I run business processes in response to blockchain transactions?
* How do I leverage blockchain technology, but not expose sensitive data stored on-chain?
* How do I process events originated from multiple parties in the correct order that is uniformly honored throughout the decentralized system?

Today, there is no comprehensive OSS project that solves all of these problems. FireFly fills this gap, and allows businesses to focus on solving business problems.

The gap includes functionality such as registering and verifying member identity within the network, defining custom data types, sharing private or public data with network members, hashing and pinning data to the blockchain, securely sending events to network members, and triggering business process flows in response to events.

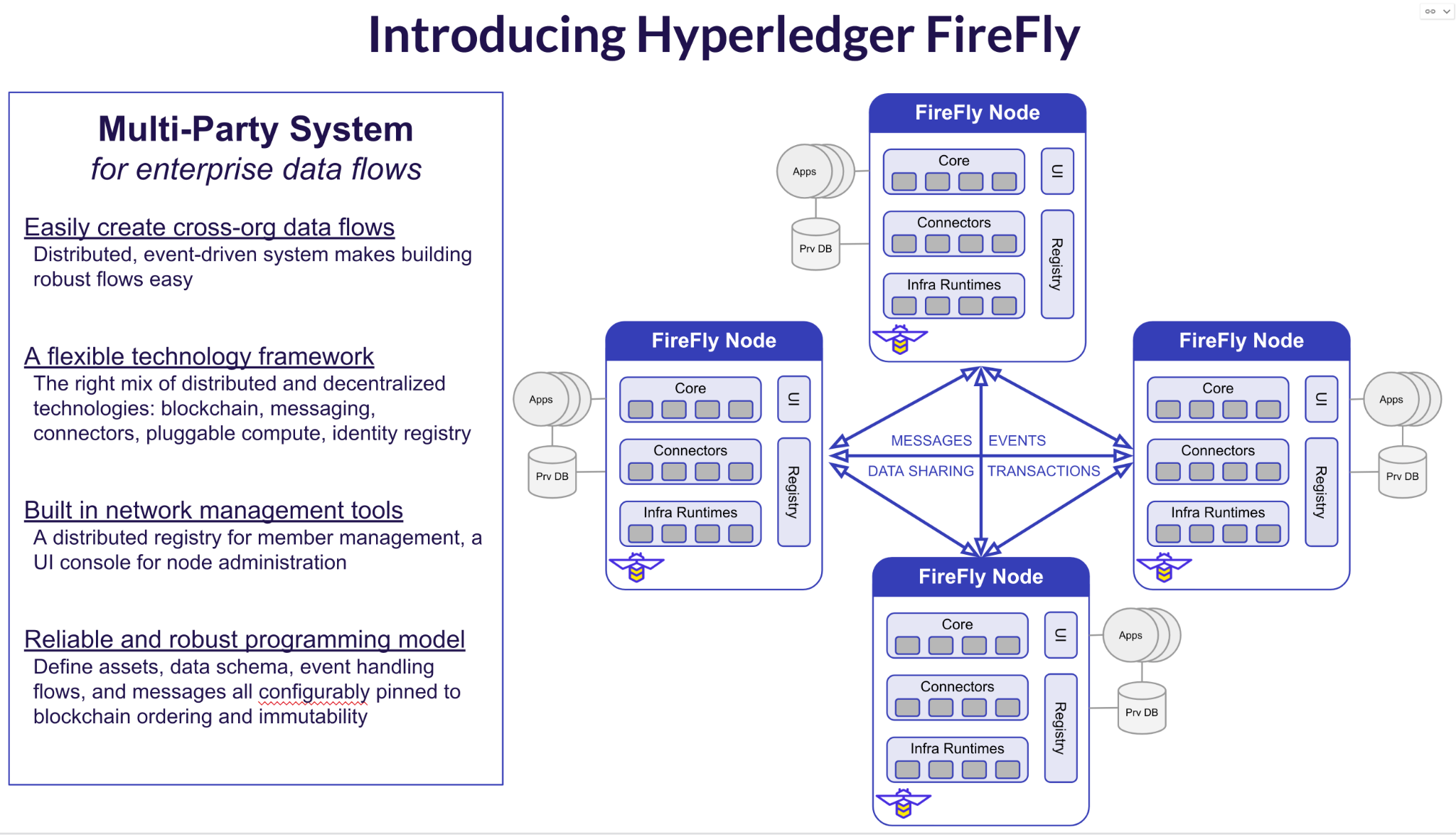
Because there is no existing OSS project that solves these problems, developers are left to themselves to create a solution. These solutions likely end up becoming specialized or tightly coupled to the project they were designed as a part of, and therefore are not reusable. Firefly seeks to fill this gap with a platform that is reusable for any multiparty business transaction backed by blockchain.

## Status

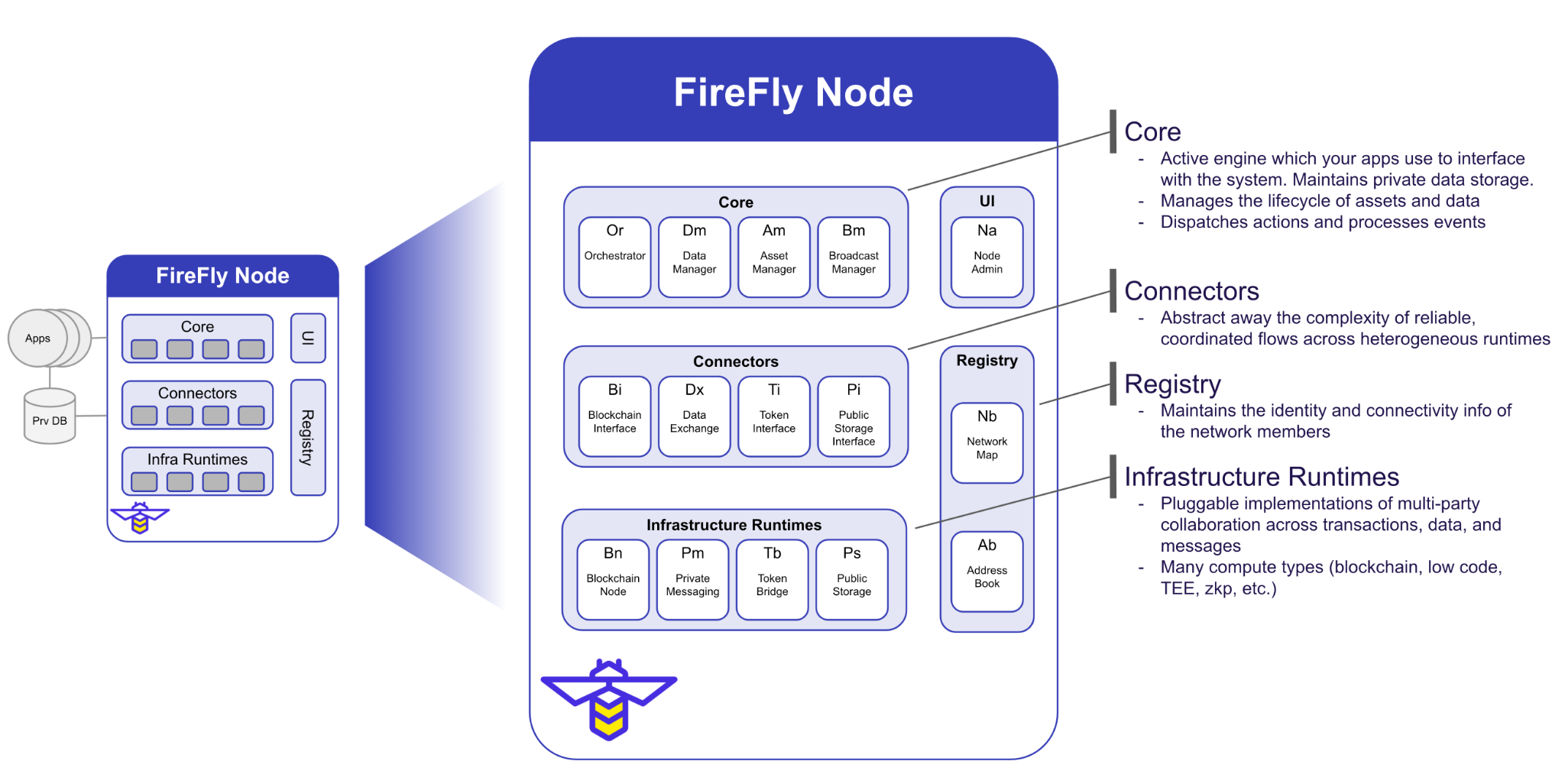
Proposal

## Solution

In a FireFly network, each member of the network deploys and runs one or more FireFly Nodes. Each Node consists of a FireFly Core application, as well as its dependencies. Each dependency is abstracted from the Core by a modular plugin system, so underlying technologies, such as the database for example, can be changed if necessary.



FireFly Core applications register themselves in a Registry, which allows other Nodes to discover them. Each Node has a series of Connectors and Common Utilities that all use plugins to connect them to the Core.



# Use Cases

There many hundreds of use cases which analysts have mapped out across all major industries, here are several illustrative examples:

* Insurance - first notice of loss, subrogation, re-insurance document flows
* Healthcare - coordination of care, provider data quality, clinical trial data management
* Financial Services - digital asset management, bill of lading, certificate of indemnity
* Supply Chain - track and trace, food safety, supplier payments

These are obviously a very small sample of the use case population. When we’ve worked across these industries and others, we’ve noticed that all of these use cases follow remarkably similar patterns:

* Multi-party business process
* Digital twin
* Digital Asset platform (STO lifecycle management)
* Document-based business transactions
* Data marketplace or sharing
* Chain of custody
* Delivery vs. payment

We can further decompose into a handful of common operations:

* **Blockchain-backed operations**
  + On-chain/off-chain event aggregation
  + Sequenced off-chain logic execution
  + Broadcast + multi-response
  + Broadcast + single-response
  + Private document transfer
  + Off-chain private or group exchange
* **Blockchain-based operations**
  + Orchestration of group logic execution
  + Digital asset lifecycle management
  + On-chain private or group exchange
* **Hybrid (both blockchain-backed and blockchain-based)**
  + Delivery vs. payment
  + On-chain authorization of off-chain coordinated actions

The FireFly core exposes a simple set of API’s to accomplish these common operations. All of the plumbing to reliably execute these operations is included in the FireFly node.

# Sample Scenario and how it works in FireFly

To further illustrate how Hyperledger FireFly acts as a MPS, let’s consider a global trade scenario as an example. We have 4 parties involved in a simplified end to end flow (the real-world use case has more parties, more documents and more subflows involved):

* Exporter
* Carrier
* Importer’s Bank
* Importer

Trade documents involved:

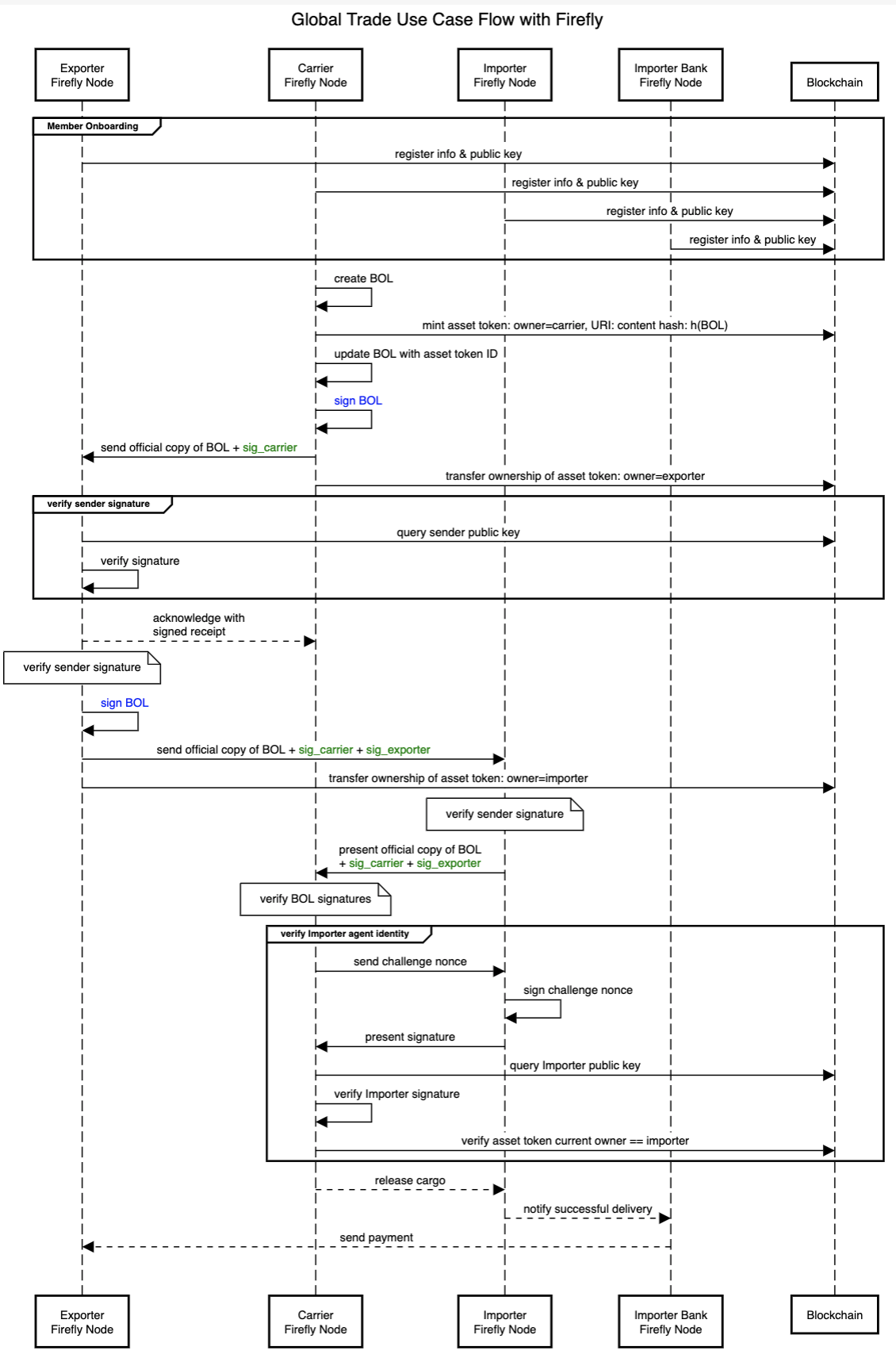
* Electronic Bill Of Lading
* Other shipping documents

## End-2-End Flow

The importer and the exporter have already concluded the necessary business negotiations and have agreed on the price and amount of the cargo. The financing side of the trade is omitted from the following flow for simplification.

1. The Exporter produces the cargo (say industrial equipments) and leases a few containers from the Carrier on a ship
2. The Exporter prepares the shipment and loads them onto the ship
3. The Carrier issues a Bill of Lading (BOL) document, signs it and gives it to the Exporter
4. The Exporter signs the BOL and forwards it to the Importer, who will present it to the ship captain in order to take delivery of the cargo
5. The ship’s captain verifies the authenticity of the BOL document, by verifying the signatures of the Carrier and the Exporter as well as the entitlement of the Importer to the cargo, and releases the cargo to the Importer
6. The Importer notifies the Importer’s Bank of successful delivery of the cargo
7. The Exporter gets paid by the Importer’s bank

## Sequence Diagram

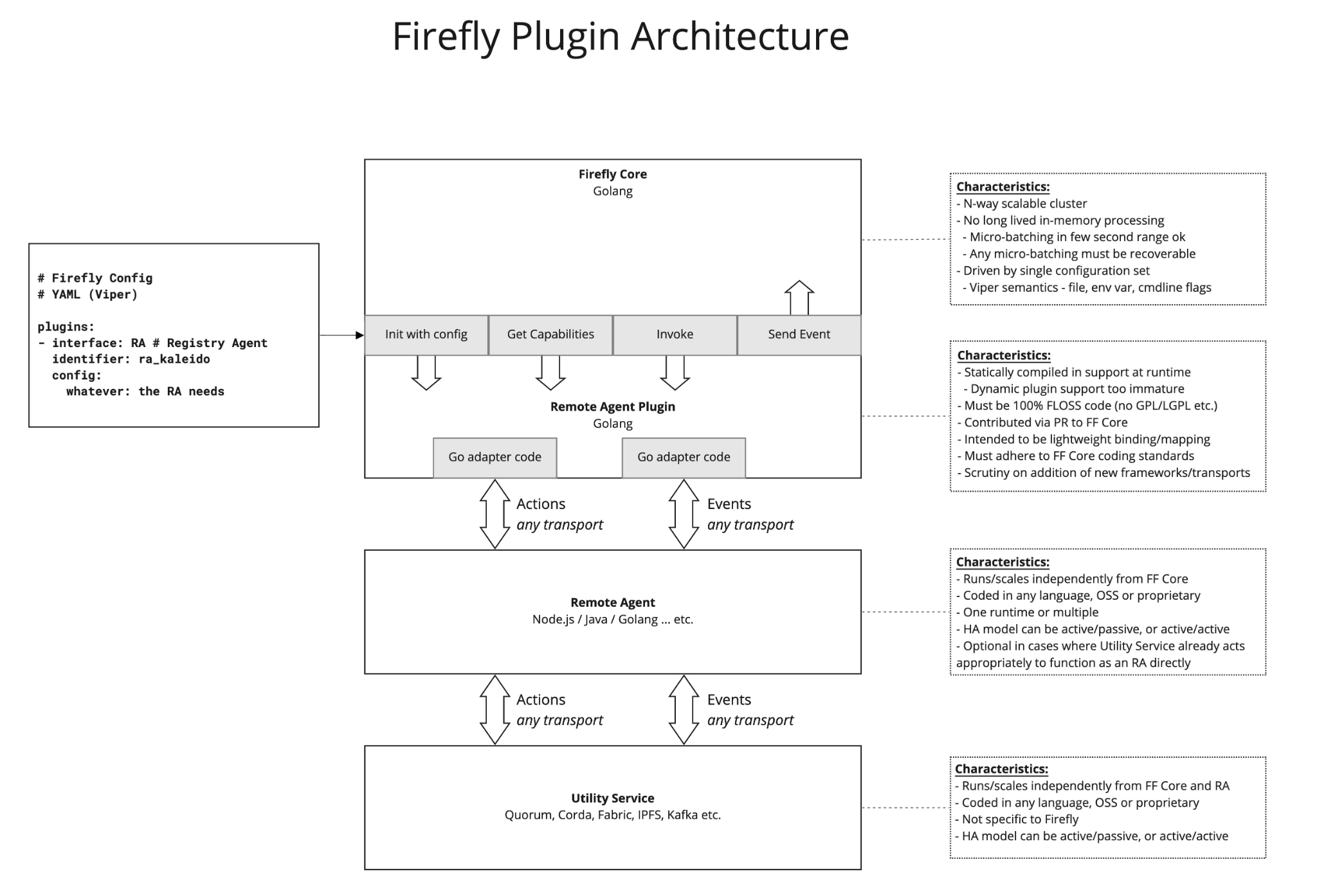


The following should be clear from the diagram:

* Blockchain is the trust foundation of the MPS
* Blockchain takes care of the critical aspects of the multi-party business transaction:
  + Identity of the document signers
  + Documents can’t be tempered with after sharing (commitment on chain)
  + The same BOL document can’t be sold more than once (double spend prevention with token representation)
  + Entitlement verification (current owner of the non-fungible token)
* Majority of the interactions in the system happen off-chain, with secure document transfers and encrypted private messages

## FireFly Details and Internals

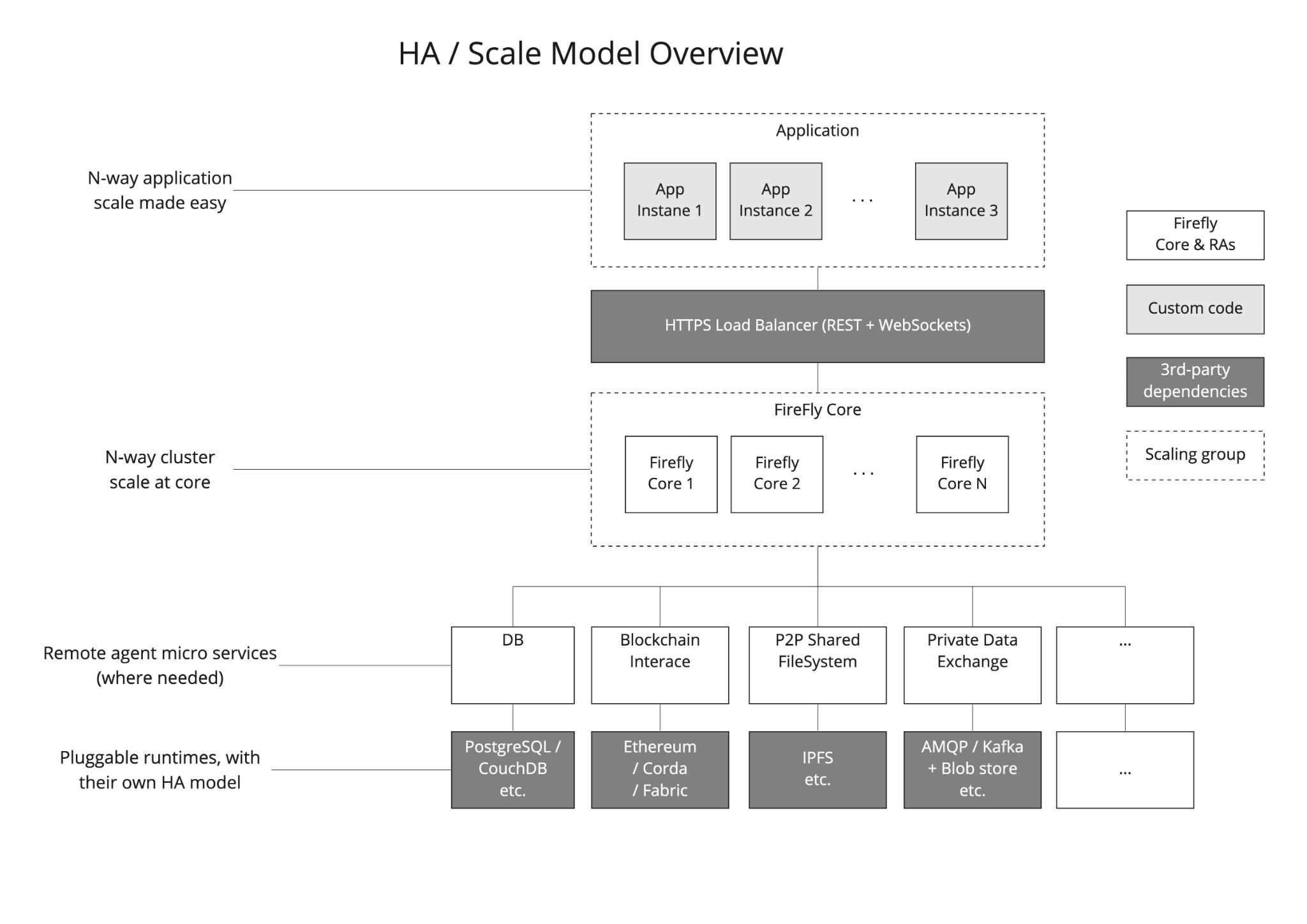
### Pluggable Architecture



The plugins comprise of a thin layer of Go code, that is embedded into the FireFly Core runtime (built statically today, but extensible to dynamic plugins as the Go plugin framework matures). Then a remote agent can be written in any language, for heavy lifting required in-between the FireFly core runtime and the underlying component. This remote agent layer is optional, but encouraged for any non-trivial REST/WebSocket interactions and scrutiny is placed on the embedded Go code to ensure the layer does not compromise the HA model of the core runtime.

### Scalability / High Availability

#### Active/Active HA Model

FireFly Core is designed to be scaled horizontally, in an active/active HA design.

The pluggability of the architecture allows other layers in the stack to have different scalability goals or constraints. For instance, if PostgreSQL is chosen as the database for a deployment of FireFly Core, PostgreSQL might be configured for replicated HA (via Patroni etc.). Then the blockchain event streaming service for your chosen blockchain technology might have a different HA strategy, requiring active/passive failover for example. This inherent pluggability is core to the architecture of FireFly.

#### Leader Election

One of the critical tasks of FireFly is to sequence events. To order events that are sent by applications through FireFly into the network, and most importantly to deliver those events back to applications in the exact order that has been agreed via pinning to a blockchain. These sequencing requirements mean there are some processing jobs that must run as a singleton instance within the FireFly core cluster. Examples include batch assembly to pack 100s of messages being sent off-chain into a single pinning transaction on the blockchain, and delivery of events to an individual application instance when the application itself is horizontally scaled. For these tasks, FireFly requires leader election when running in n-way cluster mode. Leader election is itself a pluggable interface.

### End User Features

#### Developer friendly REST+WebSocket API

The FireFly API provides building blocks designed to solve 90% of the needs of a multi-party system, without expensive and error prone custom development of on-chain smart contracts. When custom on-chain smart contract logic is required, the API allows it to be invoked via the same event-driven programming paradigm at the application level.

Developers build to this convenient REST API using their existing API/web/mobile development skill-set. They scale their application using normal n-way cluster scale approaches, and can think about the multi-party system tier in the same way they think about their other infrastructure components, like databases and messaging systems.

The model is event-driven, encouraging application design that allows robust business transactions to execute within a multi-party system. By embracing this event-driven model through simple HTTP and WebSockets in the application tier, responsive user experiences come naturally.

#### Building blocks for enterprise multi-party systems

Enterprise multi-party systems have a number of common attributes, which are distinct from the patterns prevalent in consumer blockchain applications.

Each of the following is encompassed in the design of FireFly:

* Data privacy by default
  + Regulatory requirements on privacy of data
  + Regulatory requirements over data retention and deletion
  + Competitive value of business data
  + Competitive value of transaction pattern analysis, and other metadata leakage
  + Limits the use of complex on-chain business logic
* Organizational identity
  + Employees act on behalf of the business, within a role
  + Employees identify themselves through internal single sign-on (SSO)
  + Organizations identify themselves through organizational signing keys
  + High value organizational signing keys require advanced key management
* Data integration
  + A core task of many business networks is agreeing on a data interchange format
  + Each member must integrate their core systems, with this agreed format
  + Batch, real-time and on-demand models for data interchange must coexist
  + A per-member data cache is critical for transaction history and rich query
  + A solution database needs to be continually updated as transactions occur
* Multi-party process orchestration / automation
  + A core value of these systems is to digitize inefficient existing processes
  + Members have unique, and proprietary, business decision rules for their steps
  + More non-deterministic logic than deterministic logic, is the prevalent pattern
  + A mix of automated and manual processing exists
  + A mix of network agreed, and member customizable, processing must coexist
  + A single outcome must be agreed by all parties - so sequence must be agreed
  + Incentives to follow network rules can be a mix of contractual and automated
* Broadcast and private data exchange
  + Off-chain transfer of data one-to-one, one-to-many, and one-to-all
  + Pin evidence of that transfer to a shared ledger, with identity + locked sequence
  + Perform off-chain only transfers for latency intensive operations (state channels)
  + Record off-chain delivery receipts
* Document handling
  + Business processes involve the exchange and attestation of many documents
  + These need to flow between parties, and often must remain private
  + Parties have access to different pieces of data, within a single transaction

#### Network registry

Permissioning is a core tenant of enterprise multi-party systems, as is the ability to scale and grow the network dynamically over time.

The FireFly network registry provides a pluggable governance framework for onboarding of new organizations into the network. Then a distribution framework to publish and distribute the node/endpoint connectivity information (on-chain and off-chain) for that organization, as well to allow that organization to publish identities to the network that are used for transactions.

The registry publication system uses the same model for distribution and sequencing of new data, as applications that build on top of FireFly. This means the same state of the registry can be deterministically reached by all members, while each member has their own fast local cache of data.

#### Integration of core blockchain constructs

Core blockchain constructs are composable building blocks of multi-party solutions, and can coexist seamlessly with traditional private messaging and back-office system integration.

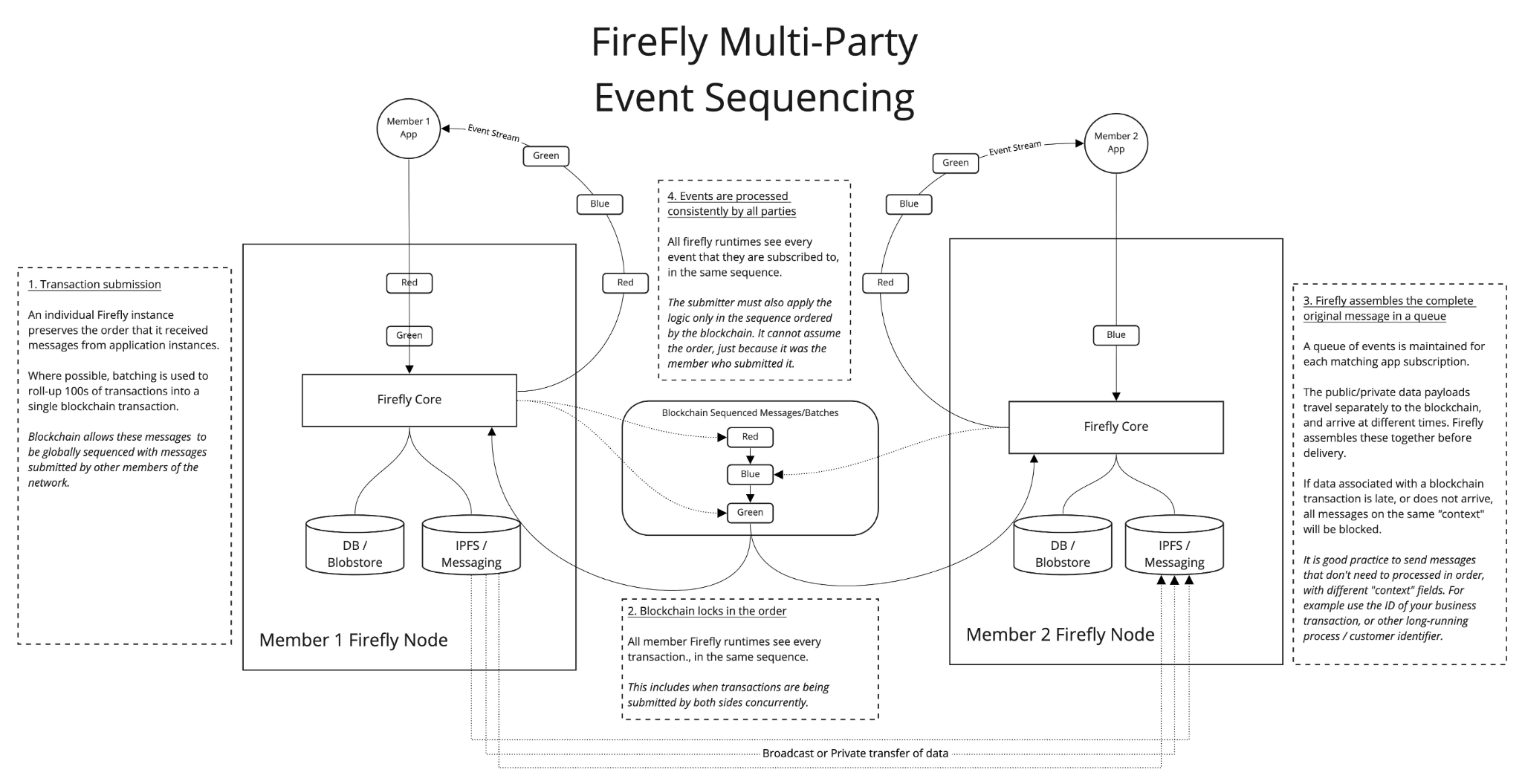
* Tokens
  + Non-fungible tokens (NFTs) - universally unique
  + Fungible tokenized value exchange - total conversation of value
* Signatures
  + Attestation to data, at a particular point in time
* Multi-party event sequencing
  + The game-changing feature of blockchain for enterprise multi-party systems
* Deterministic logic
  + Custom on-chain smart contracts
  + Trusted execution environments (TEEs) *- plug point*
  + Zero-knowledge Proofs (ZKPs) - *plug point*
  + Multi-party Compute (MPC) - *plug point*

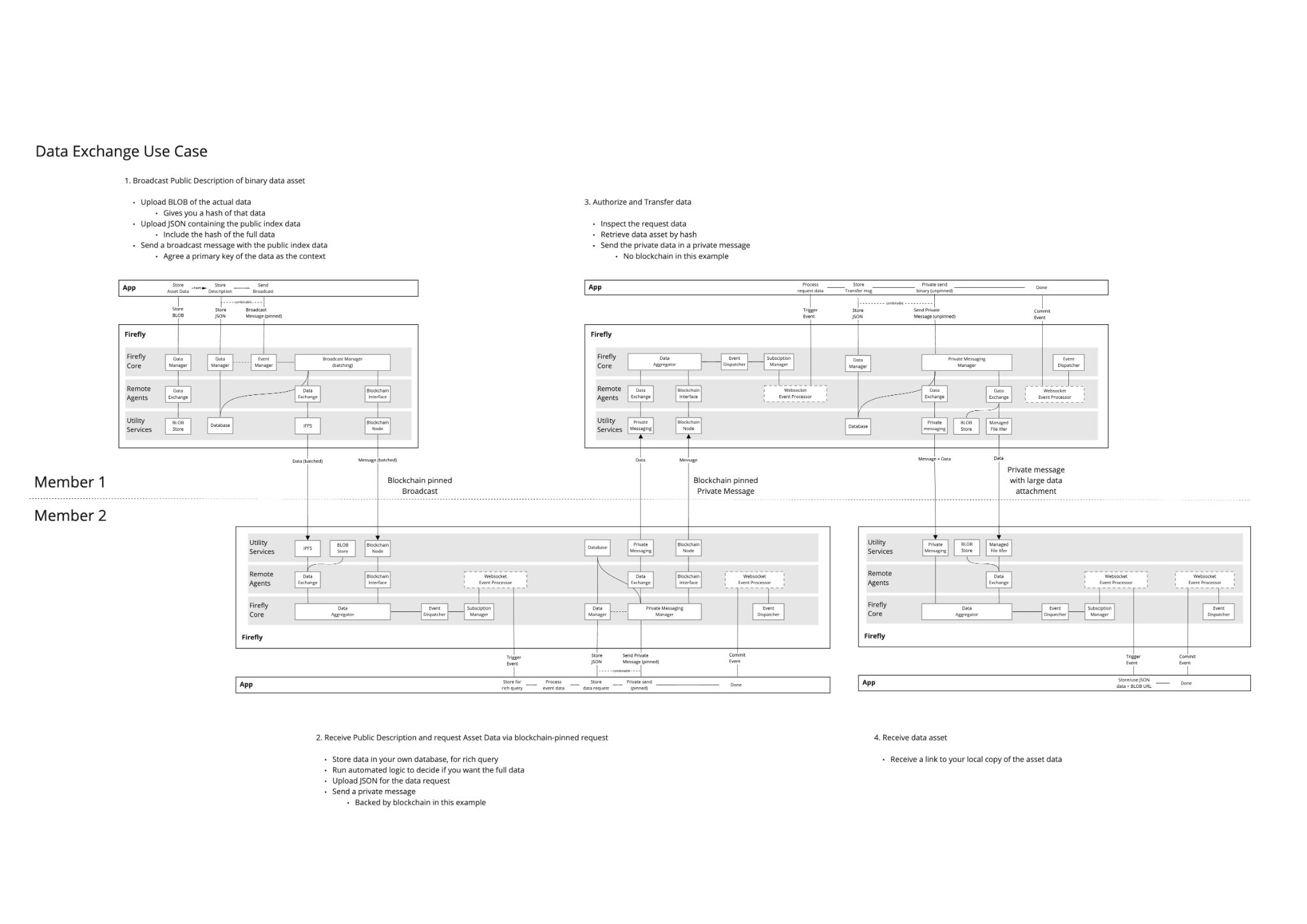
#### Coordination of on-chain and off-chain interactions

Sometimes underestimated is the importance, and complexity, of synchronizing the exchange of on-chain and off-chain data. Yet tackling this problem is almost universal in enterprise multi-party system development.

FireFly provides a rugged, enterprise-grade runtime and simple event-driven REST API model, so that developers can focus on solving interesting business problems - rather than re-building plumbing. Second order problems are also addressed, like high throughput through batching of on-chain transactions, and providing n-way application scale while retaining deterministic ordering.

The pluggable architecture allows the underlying infrastructure components to be swapped out, without changing the application. That includes the database, blob store, private messaging technology, and even the blockchain itself.



[](https://drive.google.com/file/d/1n6xX6hf0dQK7JwiAguSkB8NI0YVNIux7/view?usp=sharing)

### Pre-built implementations for the big three enterprise blockchain platforms

Remote agents for the blockchain interface are provided as follows:

* Enterprise ethereum (Hyperledger Besu / Quorum)
  + REST Gateway and Event Streams via sister project: [*ethconnect*](https://github.com/kaleido-io/ethconnect)
  + On-chain smart contracts supplied with FireFly Core
  + Full sequencing, and token support (ERC-20 / ERC-721)
* Hyperledger Fabric
  + *Under development (lead by Jim Zhang)*
* Corda
  + REST Gateway and Event Streams via sister project: *source opening soon*
  + On-chain smart contracts supplied with FireFly Core
  + Implements block based sequencing of events, within a privacy group

## Effort and resources

The FireFly code base’s current version is in production with multiple large consortiums, so Kaleido is fully committed to continued support for the open source project with currently 6 developers working full time on the code base.

We have the following companies committed to contributing:

* Kaleido: 6 developers to deliver FireFly core, plugins for Enterprise Ethereum, Fabric and Corda, and administrative console UI
* Atato: 1 developer to deliver enhancement to ethconnect and token related features

*(more committed resources being confirmed)*

## How to

*Please note, the tools in this section are still a work in progress at the time of writing.*

It is very quick and easy for a developer to get started building an app on top of FireFly using a local development environment. The FireFly CLI will walk a developer through the process. FireFly CLI uses Docker to run many parts of the system, so Docker is a prerequisite for running a local instance of FireFly for development purposes.

<https://github.com/kaleido-io/firefly-cli>

### Install FireFly CLI

$ go get github.com/kaleido-io/firefly-cli

After the CLI is installed you can create a new dev stack by running the following command:

$ firefly-cli init my-new-stack

This command will ask several questions about which specific plugins you would like to use. It will then create a new stack called “my-new-stack”

To launch the stack, run:

$ firefly-cli start my-new-stack

Once all the components in the system have started up, information such as endpoints and port numbers will be printed in the terminal. A link will also be provided to the FireFly Console web interface which can be used to view the environment.

To stop the stack run:

$ firefly-cli stop my-new-stack

To completely remove the stack, including all of its data, run:

$ firefly-cli remove my-new-stack

For developers looking to make contributions to the FireFly project, please see the GitHub repositories

<https://github.com/kaleido-io/firefly>

<https://github.com/kaleido-io/firefly-cli>

<https://github.com/kaleido-io/firefly-ui>

<https://github.com/kaleido-io/ethconnect>

<https://github.com/kaleido-io/firefly-dataexchange-https>

## References

See [citation guide](http://www.chicagomanualofstyle.org/tools_citationguide.html).

## Closure

The goal of FireFly is to speed up adoption of enterprise blockchain. The success of the project can be measured by the number of projects that are built on top of it.