Coasys Whitepaper

Collective Intelligence > Artificial Intelligence

Nicolas Luck, Tomis Paker, Joshua Parkin,
Leif Riksheim, Eric Yang
28th of July 2023

v1.4

Abstract

Coasys offers a contribution to the public's capacity to sense-make as a collective by providing the infrastructure for an open, interoperable web, a collaboratively curated knowledge graph, as well as the tools to improve our ability to find and share reliable information and coordinate effectively at any scale or complexity. By doing so, it tries to approach the meta-crisis with pragmatic technological upgrades to our society's digital communication infrastructure.

More and more people recognise the fact that our civilisation's ability to find true statements about complex issues is surprisingly low and potentially decreasing as systemic incentives promote the use of communication primarily to shape opinions of others in order to advance personal, financial, or political agendas. The advent of powerful AI through the recent breakthroughs of LLMs will likely exacerbate this problem, as realistic looking fake-news can be manufactured at low cost.

Decentralised cryptographic technologies give reason for hope since cryptographic signatures introduce a base-layer of data integrity and provenance. But for proper integrity management of complex semantic data, more than just basic cryptography or blockchains are needed. With **Holochain**—a technology that enables complex data-integrity mechanisms—distributed sense-making networks are now possible.

Over the last 3 years, the team behind Coasys has been building the **ADAM Layer**, which extends Holochain's agent-centric distributed software architecture into a novel internet layer. The ADAM layer enables a new interoperable web with social networking capacities where people have the ability to share and collaborate on semantic graphs (called Perspectives), independently of specific apps or base-layer technologies (like Holochain, blockchains, or central databases). Alongside and on top of the ADAM layer, the community platform **Flux** was built by the Coasys team which enabled us to test, prove, and iterate the groundbreaking ideas in ADAM.

Now, Coasys turns the ADAM network (and each app built on ADAM) into a sense-making tool by tapping into the network's collective intelligence. We introduce **Synergy Fuel**: *a Holochain-based mutual-credit currency backed by the network's capacity to provide trustworthy information based on real human connection and trust*. This is a perfect application of ADAM's technology-independent semantic "Perspectives" and its concept of "Social DNA", as well as Holochain's powerful, scalable and customisable validating DHTs. With Coasys, users will be able to define simple or complex data

queries by describing the properties of the desired results with ADAM's Social DNA. We have successfully prototyped the use of LLMs for the creation of Social DNA, which will provide users a simple way to create new queries with complex rules. SynergyFuel's main currency feature will be vaults where users lock-in currency units as reward for providing a result to their query. Using Holochain's distributed validation mechanism, only matching result-Perspectives will be able to unlock the provided funds in a decentralised fashion. Included in these results will be the "social stack", i.e. the path the query took through the network of agents. Queries can define what portions of the reward will be released for what role—that is, per hop through the social graph, as well as for the agent who ultimately delivers the result. Thus, SynergyFuel will be constructed as a currency coupled to, and incentivising both, the relaying of queries of others to the agent's trust-network, as well as responding with the sought after information.

The resulting system will be a fully decentralised search and synergy engine for semantic data. Being built on top of the ADAM layer, which allows existing Web 2.0 and Web3 platforms to be plugged in as data-storage layers (wrapped as ADAM "Languages"), the synergy engine works with data stored across all lower technology layers. ADAM Perspectives serve as semantic overlay graphs that associate data stored across the internet in various servers, blockchains, and Holochain DHTs. ADAM makes it possible to build distributed, agent-centric apps on Holochain and other network technologies which can have their unique value proposition completely independent from Coasys and this semantic search engine. Users of these apps, being in full control of their data, can choose to participate in these semantic searches by offering specific data (or all of it) and use their data to earn SynergyFuel.

In order to sustain the development of the ADAM layer, Flux and other ADAM apps, and to begin building Coasys and SynergyFuel within a non-profit, open source context, we are conducting a pre-sale of SynergyFuel units. This document describes the underlying technological innovations, mainly the ADAM layer, the first apps built on ADAM, the mechanics of the query engine, the SynergyFuel currency, and the structure of the pre-sale.

Addressing our broken information system

In today's interconnected world, the ability to make sense of complex issues is crucial for individuals, communities, and organisations to navigate an increasingly volatile and uncertain landscape. However, the current state of our information ecosystem presents significant challenges for effective sense-making, as it is inundated with misinformation, disinformation, and commercial and political interests. This compromised environment has far-reaching implications for the potential for constructive dialogue, meaningful collaboration, and our relationships with one another.

The complexity of modern challenges requires a collective approach to sense-making that can accommodate multiple perspectives and foster a shared understanding of the problems we face. As the global issues we face become increasingly urgent, the need for accurate information and effective sense-making has never been more critical. Unfortunately, the current information landscape is not conducive to these objectives, with numerous factors contributing to the breakdown of sense-making:

a) **Information Overload**: While the abundance of information has its benefits, it has also made it difficult for individuals to process and discern the credibility of the vast amount of content they encounter daily.
b) **Proliferation of Misinformation and Disinformation**: The internet has made it easier than ever for bad actors to intentionally spread misleading information, exploit cognitive biases and prey on emotions.
c) **Echo Chambers and Filter Bubbles**: Social media algorithms prioritise engagement, which can lead to the formation of echo chambers, filter bubbles, and the reinforcement of existing beliefs.
d) **Erosion of Trust in Traditional Information Sources**: The rise of the internet has revealed that traditional news and media outlets can be influenced by financial incentives, political pressures, personal biases, and even outright corruption as the Twitter files have demonstrated.

There have been attempts to address the breakdown of sense-making through centralized technology solutions. However, these systems not only perpetuate the issues related to trust and reliability but also create new vulnerabilities and limitations. Here's how:

a) **Centralisation of Power and Control**: Centralised platforms own and gate-keep our digital identities and data, making it easier to manipulate, censor, and control what is being disseminated.

b) **Censorship and Surveillance:** Since information is stored and distributed by a handful of centralised entities, it is much easier for governments and the platforms themselves to censor or track individuals.

c) **Susceptibility to Misinformation and Disinformation**: Centralised systems are inherently more vulnerable to the spread of false or misleading information, as they rely on a limited number of sources and gatekeepers.

d) **Monetization and Manipulation of User Data**: Many centralized platforms generate revenue through targeted advertising, which relies on the collection and analysis of vast amounts of user data. This monetization model not only compromises user privacy but also incentivizes the platforms to manipulate user behavior and information exposure to maximize profits, further eroding trust in the information ecosystem.

e) **Scalability and Security Concerns**: Centralized systems are more susceptible to single points of failure, both in terms of technical infrastructure and the potential for bad actors to compromise the system. As these platforms grow in size and complexity, they face increasing challenges in maintaining security, reliability, and performance.

In this context, addressing the broken sense-making ecosystem is of paramount importance. It requires the development of new mechanisms and platforms that can ensure the integrity of information and promote a more transparent and accountable information ecosystem. By empowering individuals to access, share, and verify information through decentralized and agent-centric networks, where cryptographic signatures and provenance of data are inherent, we can foster a renewed sense of trust in the information landscape and pave the way for more meaningful and effective collective discourse, sense-making, and action.

Why an agent-centric network can be a solution

Agent-centric networks offer a promising alternative to centralized technology solutions. By empowering individuals to take control of their data and interactions, these decentralized systems create a more transparent, reliable, and resilient foundation for collective sense-making. Some key reasons why agent-centric networks can serve as a solution include:

1. **Cryptographic Signatures and Data Provenance**: In an agent-centric system, all information is cryptographically signed, providing an intrinsic provenance of who expressed the data and when. This feature adds a layer of trust and accountability to the information ecosystem, as it allows users to verify the source of data and ensure its authenticity, reducing the likelihood of misinformation and manipulation.

2. **No Implied Objectivity**: Agent-centric networks acknowledge that there is no single "objective truth" in the information ecosystem. Instead, they recognize that there are fundamentally different perspectives held by different agents. By embracing this diversity of viewpoints, agent-centric systems foster a more nuanced understanding of complex issues and enable more effective collective sense-making.
3. **Decentralized Authority and Control**: Unlike centralized platforms, agent-centric networks distribute authority and control among all participants, reducing the potential for manipulation, censorship, and control by a few powerful entities. This decentralization allows for a more democratic and inclusive information ecosystem, in which diverse perspectives can be shared and considered equally.
4. **Enhanced Privacy and Security**: By giving users control over their data and interactions, agent-centric networks provide greater privacy and security. Users can choose what information they share and with whom, while cryptographic signatures help ensure the integrity and authenticity of the data. Additionally, decentralized systems are more resilient to single points of failure, resulting in a more robust and secure information infrastructure.
5. **Censorship and Surveillance Resistant:** Agent-centric networks are inherently resistant to censorship and surveillance as users retain full ownership over their data, identity, and interactions.
6. **Human-centered Design:** The emergence of new revenue models that are not incentivized by the need to maximize user attention and engagement could lead to healthier design patterns.
7. **Support for Diverse Perspectives and Synthesis**: Agent-centric networks facilitate the sharing and synthesis of various perspectives, enabling users to gain a more comprehensive understanding of complex issues. By incorporating diverse viewpoints, these systems encourage collective sense-making and foster a more informed and engaged user base.

By adopting agent-centric networks as the foundation for our information ecosystem, we can transform the way we access, share, and interpret information, promoting more effective collaboration and understanding across diverse perspectives and communities.

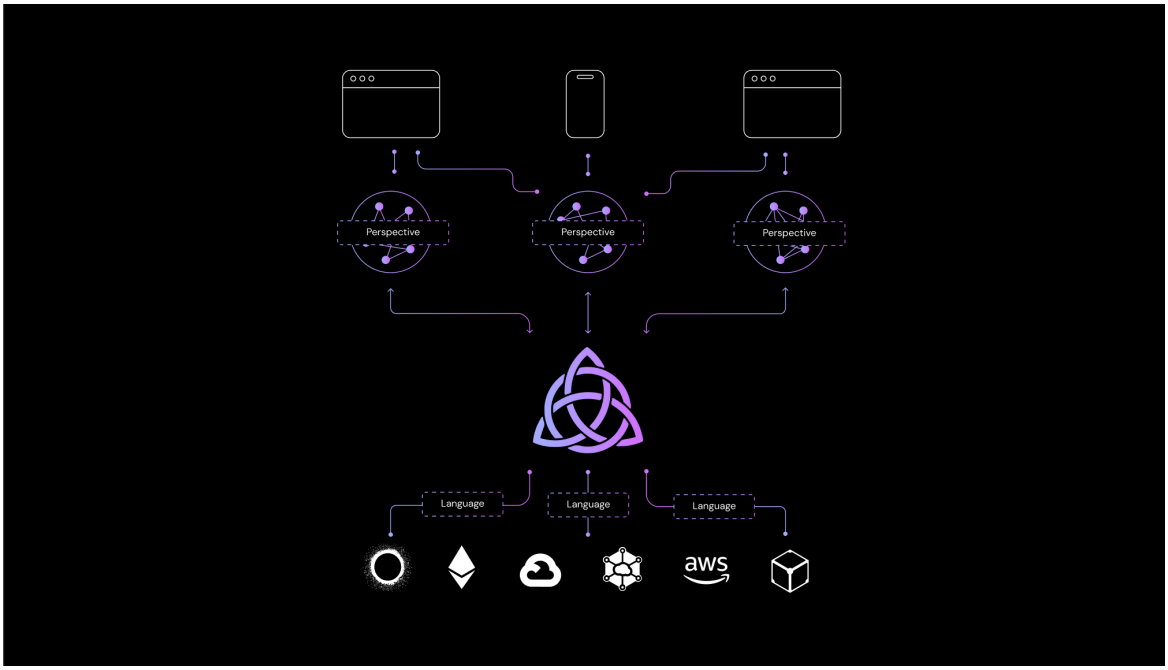
ADAM Layer

ADAM, an acronym for ***Agent-centric Distributed Application Meta-Ontology***, is a paradigm-shifting innovation designed to transform the way we interact with the digital world by bringing it closer to the natural dynamics of human communication. Regarded as a layer on top of the existing internet (TCP/IP) layers, ADAM aims to bridge the gap between various applications and technologies, fostering seamless collaboration, data sharing, and interoperability across diverse platforms and between agents (humans), directly.

The agent-centric nature of ADAM allows humans to communicate with one another, independent of the app or technology they use. By wrapping these technologies as "*Languages*," ADAM enables users to exchange semantic statements as associations in semantic graphs, here called "*Perspectives*," effectively exchanging *contextual meaning* and facilitating richer interactions. This approach aligns more closely with the way humans naturally communicate, creating the basis for a more intuitive and meaningful digital experience, without implying or assuming a specific semantic, interaction pattern or underlying technology.

ADAM is *not only* a **spanning layer** that connects different technologies but also serves as an **application development framework**, empowering developers to build a wide range of applications with greater flexibility and interoperability. As a meta-ontology, ADAM provides the foundation for seamless integration between diverse apps and networks, paving the way for a more unified and coherent digital landscape.

By implementing the ADAM layer, we can move towards a more interconnected, human-centric digital ecosystem that supports meaningful communication and collaboration—ultimately unlocking the full potential of our digital experiences, **without locking users into one specific platform, network, technology, or app.**



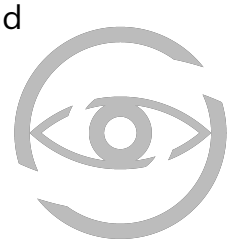
Agent-centric **Spanning Layer**

ADAM is (like the IP layer in TCP/IP) a spanning layer—but an agent-centric spanning layer. As such, it is a unique approach to connecting diverse applications, user interfaces (UIs), and networks through the central role of the agent, i.e. the human user. This means users can mix and re-mix Languages inside Perspectives and use these remixes within various ADAM apps (I.e. user interfaces built on top of ADAM and ADAM Perspectives).

The primary benefit of an agent-centric spanning layer is the ability to facilitate seamless communication and collaboration among users, *regardless of the specific applications, technologies or ways of expressing* they choose. By placing the user at the centre of the digital ecosystem, this spanning layer enables users to connect, interact with each other, and share information across various platforms, breaking down barriers that often exist between disparate technologies. It also allows new technologies to emerge without suffering the "cold start" problem where no users exist on that network, since it can inherently be used within an integrated data context, and leaned on more and more as users see its benefits without having to switch their entire operating context and data.

Another advantage of an agent-centric spanning layer is the increased flexibility it offers for application development. Developers can build applications that leverage the unique properties of different networks and data storage technologies, while still ensuring that their users can effectively communicate and collaborate with others across the network. This flexibility allows for the creation of more innovative and versatile applications that better serve the needs of their users.

In short: as a spanning layer, ADAM allows many-to-many mappings between apps/UIs on one side and Expression Languages with their choice of storage and distribution technologies on the other. It carves out the core aspects of social networks (users and their relationships) and provides that in a neutral, quintessential, and reusable form.



Base Ontology Classes: Agents, Languages, Perspectives

To achieve that level of interoperability, composability and evolvability, the ADAM layer introduces a base ontology composed of three primary classes: ***Agents, Languages, and Perspectives***. These classes serve as the foundation for building a *semantic, interoperable and evolvable digital ecosystem*. Let's dive into each of these base ontology classes in more detail.

1. **Agents**: In the context of the ADAM layer, an agent refers to the individual user that participates in the digital ecosystem. Agents are responsible for creating, sharing, and interacting with information across the network. ADAM uses the **W3C's DID standard,** providing agents with their own sovereign digital identity. ADAM then uses the public-private key pair of that DID to cryptographically sign and verify the data agents create, ensuring that the provenance of the information is intrinsically linked to its originator. Agents can resolve the DID URI of other agents and receive an agent expression that includes:
   a) the resolved agent's *Public Profile Perspective*—a built-in way for agents to provide public information about themselves in the form of a semantic website, and,
   b) The agent's *Direct Message Language*—an ADAM Language chosen by the agent to receive direct messages in. By having the agent provide this Language, they can choose the implementation of their inbox (Holochain DHT, email server, custom REST API, etc..)

$$Agent(DID) = \begin{cases} profile : Perspective \\ directMessage : Language \end{cases}$$

2. **Languages**: Languages in the ADAM layer represent ways for users to express themselves. A Language is both the class and the storage mechanism of its **Expressions**. Languages encapsulate the actual technology used to communicate, like Holochain, IPFS,

blockchains, centralized APIs, etc. Expressions are always created, and thus signed, by an agent. Languages map expression addresses to their actual data:

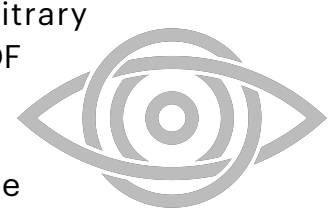
$$Language_i : Addr \rightarrow \{Expression_i\}$$

As such, Languages implement the *objective* part in the ADAM layer: every agent should get the same expression data for a given address. By abstracting these underlying technologies into Languages, the ADAM layer enables agents to seamlessly interact with information stored across various systems without being restricted by the specific implementations of those technologies. Encapsulation into Languages provides a means to bridge various data storage and integrity technologies, protocols, and applications within the agent-centric distributed application meta-ontology. To ensure compatibility and interoperability across the ADAM ecosystem, the implementation format of ADAM Languages was chosen to be both flexible, standardised, and simple: **EcmaScript Modules.**

The ADAM runtime includes a JavaScript/EcmaScript engine to execute sandboxed Language code. ADAM defines a few specialised Language interfaces for specific purposes (see Link Languages below and appendix or the ADAM documentation), but the main Language interface is as simple as the mapping from Expression address to Expression content.

Language sandboxes have network access so Languages can be used to wrap all kinds of networked services. The ADAM runtime also makes available its included Holochain conductor to the Language runtime so that Languages can easily include Holochain DNAs/hApps, register those with the ADAM runtime for installation, and then call zome functions easily. As ADAM evolves, we plan to include further decentralised nodes within the ADAM runtime through a plugin-system, to make the development of Languages for these networks easier. Using hosted nodes, like those provided by Infura for instance, already allows for the wrapping of Blockchain services within ADAM Languages.

3. **Perspectives**: Perspectives in the ADAM layer are semantic graphs, i.e. contexts that associate Expressions (of arbitrary Languages) with each other through **Links** (similar to RDF triplets). Perspectives always start as private and local to the agent holding it. Each Perspective is a collection of semantic statements that represent an agent's unique



view or understanding of a particular topic or domain—it can be regarded as a *subjective* graph over *objective* Expressions. Perspectives are the fundamental building-block of AD4M apps, to which they are like a local graph database to store the domain-logic state through meaningful links.
Technically, Perspectives are just graphs, represented as lists of link expressions (links with provenance):

$$Perspective = \{l \mid l \in LinkExpression\}$$

$$LinkExpression = \left\{ \begin{bmatrix} source \in URI \\ predicate \in URI \\ target \in URI \\ author \in DID \\ timestamp \in Date \\ proof \in Signature \end{bmatrix} \right\}$$

In conclusion, the base ontology classes of Agents, Languages, and Perspectives form the core building blocks of the ADAM layer. These classes enable (by recombination as we will see below) the creation of a semantic and interoperable digital ecosystem that promotes seamless communication, collaboration, and sense-making across different applications and technologies.

ADAM Expression **URIs**

ADAM defines a super-set of URIs by mapping Languages to URI schemas. The default case for URLs is treating every ADAM Language as a URL schema with the Language's address (=hash) as the schema-string and then leaving everything but the schema for the Language to resolve:

```
<language address>://<expression sub-address>
```

This creates a way of **globally addressing Expressions in different Languages** with *backwards-compatibility for existing http/https URLs*. Together with a built-in aliasing mechanic that allows for mounting of a Language under a given schema, especially https, Perspectives can integrate data from the already existing Web 1 and 2.0 next to distributed Web3 storages. Put differently: URLs like https://ad4m.dev are valid ADAM Expression URLs with the HTML document behind this link being the data of

that Expression. It is just a matter of registering 'https' as a special language in the ADAM runtime. The same holds for DID URIs, which get routed to the Agent bootstrap language.

Bootstrapping with core Languages

Next, we apply the concept of ADAM Languages to the concepts of the meta-ontology itself. This **self-recursive trick** will spawn a network that has a high degree of evolvability, as we will see in following sections.

The resulting ADAM Languages are called Bootstrap languages since their immediate function is to bootstrap an ADAM agent instance into the evolvable ecosystem that the ADAM layer spawns through its ontology. As long as an ADAM implementation comes with this set of bootstrapping Languages, it will be able to discover other agents, their public Perspectives and the Languages they are "speaking", and thus join the collective.

The minimal core of bootstrap Languages include one Language for each of the ontology's base entities:

1. **Language of Agents**: which *maps DID* names like did:key:zQ3shSKjGQKHSLhVLsTMwJDCrzcWoT6cyhMW4qSgnZ4WbEpLw *to Agent Expressions* as defined in the core ontology above. This is a public space for every agent to publish information about themselves which other agents can retrieve just by resolving a DID name.

2. **Language of Languages**: which *maps Language addresses/hashes to the Language source code and meta data*. New Languages developed by users can and should be published as Expressions in this Language such that other users can resolve and download Language hashes and install new Languages as they go. Using the concept of Languages and Expressions on itself adds provenance to shared Language source code. ADAM uses that when encountering and trying to resolve Expressions of new/unknown Languages (in Perspectives shared by others). **Resolving these expressions means downloading the source code of that new Language from this "Language of Languages" and installing it.** This will only happen automatically if trust in the provenance of that source code can be established—either because the author is a trusted agent (see below) or the Language is a parameterised clone of a trusted Language template (see appendix).

3. **Language of Perspectives**: which stores Perspectives as immutable snapshots to allow for Perspectives referencing Perspectives and build more complex semantic graphs with Perspectives as building-blocks.

ADAM runtime and **interface**

While the principles of ADAM as a meta-ontology can be applied across various implementations to foster interoperability, we have taken the initiative to create a complete runtime environment to actualise the potential of ADAM. This runtime environment is packaged as a launcher, which keeps the ADAM executor running in the background, ensuring that the agent-centric network is constantly active and available.

One of the essential aspects of the ADAM runtime is the **ADAM interface**, which has been built using *GraphQL*. This choice provides an interoperable, efficient, and flexible means of structuring and retrieving data and remote-control parts of the ADAM runtime (from UIs and apps), which is crucial for the kind of complex, semantically rich interactions that ADAM enables. Additionally, to ensure the security and integrity of data and interactions, *the ADAM interface is fortified with a capability-based security scheme*. This approach ensures that only authorised agents can perform specific actions, providing a robust and secure framework for agent-centric interactions.

To further facilitate the use of ADAM and its implementation, we have created **client libraries in JavaScript and Rust**. These libraries allow user interfaces to utilize ADAM as their persistence layer, streamlining the integration of ADAM's capabilities into various applications, or deployments. We envision in the future that ADAM could be deployed on distributed cloud hosting networks such as Holo, or run on your own private server. Since all your ADAM data is stored in a single directory, it is simple to move between different hosting mechanisms without losing data, or even use multiple at once with syncing setup between these deployments.

As will be shown in the coming sections, we have also developed tooling, especially related to Social DNA, offering an ergonomic way for app developers to define their app-specific ontology based on ADAM's meta-ontology. This allows developers to create apps that are inherently interoperable, further promoting the vision of an interconnected, agent-centric web. Through the ADAM runtime and interface, we are making it easier for developers and users alike to harness the power of ADAM, fostering a more robust, diverse, and user-centric internet ecosystem.

Neighbourhoods: Shared Perspectives and Group Collaboration

In the ADAM framework, Neighbourhoods are an essential component that serve as **shared Perspectives**. A Perspective, as we've discussed earlier, is the individual viewpoint of an agent—a subjective lens through which they perceive and associate objective Expressions to form and represent meaningful connections, represented as overlay graphs over the Expression of various ADAM Languages. *These Perspectives can get shared in a collaborative and dynamic way among a group of agents. We then refer to this collective space as a* ***Neighbourhood****.*

A Neighbourhood, therefore, is a dynamic, shared context where a group of agents interact and communicate subjective associations. Agents within a Neighbourhood maintain a local Perspective that is entangled with the Perspectives of the other members. This entanglement allows for the seamless synchronisation of changes within the group, thus fostering a coherent shared context. ADAM apps/UIs interface with Neighbourhoods like they do with local Perspectives—adding and retrieving triples/links.

This synchronisation is facilitated by a specific ADAM Language known as a **Link Language**. The contents of Perspectives, and by extension Neighbourhoods, are Links—semantic statements that form the fundamental building blocks of these shared contexts. These Link Languages exchange LinkExpressions (Expressions that represent Links) between agents. The usage of these LinkLanguages in the context of Neighbourhoods happens transparently to ADAM users/UIs, once setup.

Neighbourhoods can be created by any agent from any local Perspective. They are represented by Expressions in another bootstrap Language (i.e. **Neighbourhood Bootstrap Language**). This Language is aliased to the URL schema 'neighbourhood', which means that each Neighbourhood is addressable by a unique URI like “neighbourhood://Qm123abcd". Agent’s can join Neighbourhoods through such URLs. The Expression data behind that URL includes the address of the LinkLanguage that forms the backbone of the Neighbourhood, and a “meta” Perspective which can hold arbitrary meta information about this Neighbourhood:

$$NeighbourhoodExpression = \begin{bmatrix} linkLanguage \in LanguageAddress \\ meta \in PerspectiveSnapshot \\ author \in DID \\ timestamp \in Date \\ proof \in Signature \end{bmatrix}$$

With the Neighbourhood Language being public, this information about Neighbourhoods is publicly accessible for given Neighbourhood URIs. Wether an agent can join a Neighbourhood is determined by the chosen Link Language implementation. This can entail any kind of logic to reason about the agent to join. In the context of Holochain this would be the agent-validation, or membrane proof.

After resolving the Neighbourhood URI and receiving an Expression (see above format), the LinkLanguage can be downloaded from the Language of Languages and installed—granting the new agent access to the shared link data within the Neighbourhood, if they pass the Link Language's membrane validation.

Link Languages are Languages in which the Expressions are Link Expressions with additional interface functions that ADAM defines for Link Languages in order to enable proper CRDT-like synchronisation and push dynamics when other agents share changes to the shared Perspective:

```
/** Interface for "Link Languages" that facilitate the synchronization
 * between agents' local Perspectives inside a Neighbourhood.
 * The assumption is that every version of the shared Perspective
 * is labeled with a unique revision string.
 * Changes are committed and retrieved through diffs.
 * Think of a LinkSyncAdapter as a git branch to which agents commit
 * their changes to and pull diffs from their current revision
 * to the latest one.
 */
export interface LinkSyncAdapter {
    writable  : boolean
    public  : boolean
    others  : Promise DID

    /** What revision are we on now -> what changes are included in output of render() */
    currentRevision  : Promise string

    /**
     * Check for and get new changes,
     * notify others of local changes.
     * This function will be called every
     * few seconds by the ad4m-executor.
     *  */
    sync  : Promise PerspectiveDiff

    /** Returns the full, rendered Perspective at currentRevision */
    render  : Promise Perspective

    /** Publish changes */
    commit diff: PerspectiveDiff : Promise string

    /** Get push notification when a diff got published */
    addCallback callback: PerspectiveDiffObserver

    /** Add a sync state callback method */
    addSyncStateChangeCallback callback: SyncStateChangeObserver
}
```

When a community or group of agents create a new Neighbourhood, they choose the specific Link Language implementation they want to use. This choice determines the technological base-layer for their communication infrastructure, allowing communities to customise their Neighbourhood according to their needs. We have developed a working Link Language implementing a git-like graph of change-sets, built on Holochain—offering a robust, decentralised platform for the creation and maintenance of Neighbourhoods.

In summary, Neighbourhoods in ADAM provide a flexible, secure, and user-controlled framework for group collaboration. By facilitating shared Perspectives among agents, Neighbourhoods enable meaningful, interconnected communication within a diverse range of contexts. **ADAM Neighbourhoods are generic group collaboration contexts** *which allow their members to setup the group once, but compose and recompose application user interfaces and underlying Languages (networks and tech-stacks) within the life-time of that virtual group context.*

Social DNA: a language for semantic interaction patterns

In ADAM's approach towards interoperability, Social DNA plays a vital role in establishing common semantic patterns across different applications used within the same Neighbourhood. This **common semantic language**, expressed in the form of **Prolog predicates**, **describes graph patterns** that different applications are interested in.

There is a conceptual relationship between ADAM Social DNA and ADAM Languages: *objective and subjective aspects of data are distinctly handled through the use of ADAM Languages and Social DNA, respectively.* ADAM Languages are used to encapsulate objective facts, while Social DNA enables the coherence of subjective nuances within the shared semantic graph in order to establish inter-subjective patterns and data-structures.

Let's delve into this distinction through an example involving two applications: a social media application for creating posts with comments, and an event/calendar application:

Consider an event that's created using an ADAM Language specifically designed for events. An event, as an objective Expression, possesses certain

properties such as date, time, and place. These attributes are the same for all agents, regardless of their perspectives or the Neighbourhoods they belong to (the event Language will resolve a given URI to the same data, independent of the agent's context).

However, subjective aspects of this event, such as comments shared about the event within a particular Neighbourhood or an individual's intent to attend, would better be represented by links within a Perspective (or Neighbourhood). The Social DNA used in a Neighbourhood can describe these subjective graph patterns, enabling different applications to interact with the shared semantic graph in an interoperable way.

So, let's say that within our Neighbourhood, someone comments on the event, suggesting a casual meet-up. *Following the Social DNA for posts that was installed in our Neighbourhood prior, this comment would be captured as an association or a link from the event base, using the predicate "post://has_comment"*. Even though the base data is an event, it can now be interacted with and understood as a 'post' within the context of the social media app, thanks to the addition of these associations.

In this way, the most basic application of what ADAM defines as Social DNA is very **similar to the W3C's SHACL Shapes Constraint Language**: *defining constraints to match semantic graphs against, in order to find/detect meaningful structures*. There is more to ADAM's Social DNA as we will see in several sections below.

But first—some practical details to ground this discussion of SDNA in reality.

The ADAM runtime spawns a **Prolog engine for each Perspective** and provides access to apps/UIs via the client library's *infer* method:

```
let results = await perspective.infer(
    `triple("${base}", "post://has_comment", Comment)`.
)
```

This will run the given Prolog query

`triple(<base>, "post://has_comment", Comment)`

binding the free variable `Comment` to expression URIs that are linked from a given base with the predicate `post://has_comment`. When setting up the Prolog engine for a perspective, the ADAM runtime feeds all links of that

perspective into the Prolog engine as facts—both in a short form with the predicate `triple/3` which associates source, predicate and target of a link, as well as the full `link/5` predicate which also includes author and timestamp of a Link Expression.

This allows to use Prolog's backtracking-based query mechanism to detect arbitrarily complex graph patterns. *Also note that since all Link Expressions are signed by their respective authors (and discarded by the ADAM runtime if that signature is broken), we can use Prolog derivations to elegantly infer complex semantic properties based in cryptographically proven and thus tamper-resistant statements and actions of agents*. In other words, **ADAM's Social DNA is the basis for dynamically defining interaction patterns that can be checked and thus enforced between agents** without introducing physical or even logical centralisation.

Conceptually, ADAM's Social DNA shares the abstract core principles behind **Holochain's DNA validation rules**—but it's an application of these principles on a different and complementary level of abstraction. The main differences between ADAM Social DNA and Holochain DNA based validation rules are the following:

- ADAM SDNA is not static. A Neighbourhood can iterate quickly on different sets and compilations of various SDNA rules. This is by design: ADAM SDNA should enable users to define and share their own custom interaction patterns without having to require deep technological skills and without creating a new app. *Users should be able to use SDNA to adapt existing infrastructure, i.e. Languages, Neighbourhoods and apps/UIs, to their community's and culture's specific interaction patterns*.

- ADAM SDNA runs on Perspectives, i.e. base-layer-independent semantic graphs. As such, it marries the concepts of the semantic web with agent-centric principles and distributed validation as pioneered by Holochain. It defines interaction patterns independently of back-end and networking technologies used in Languages and thus allows for the inclusion of various technologies and network stacks in the definition and application of high-level semantic patterns.

Social DNA is like the transposition of Holochain's DNA to a higher octave. It enables an easier on-ramp into this novel agent-centric way of ensuring data integrity. But it comes with a cost: populating the Prolog engine with semantic link-facts assumes the provisioning of a full snapshot of the shared Perspective. ADAM Neighbourhoods are mechanisms for enabling full-synchronisation between agents. This makes sense for digital mappings of

most in-group social structures like teams, organisations, physical living communities, etc. And with the Social Organisms described below, ADAM offers a different approach to scaling by defining fractal holarchies of these structures. A major aspect that Social DNA is intended to facilitate is the frictionless introduction of new interaction patterns at, and from, the edge. This means enabling users to quickly and easily prototype new social dynamics and have their digital communication infrastructure develop with and in concordance with their (non-digital, real-world) social systems.

As soon as a new interaction pattern implemented in **Social DNA needs to scale up beyond the membrane of an ADAM Neighbourhood** there are three ways possible:

- Implement an ADAM Language which defines Expressions that resemble the semantic of the interaction. This is like traditional app development—the back-end part of it. While any technology may be used and chosen specifically for that new Language, the migration from Social DNA to Holochain DNA might be specifically adequate, given the similarity in its approach.

- Spawning ADAM Social Organisms that embody the given Social DNA and organising those in fractal holarchies. See below for more details on ADAM's Social Organisms

- Using the Social DNA in the Coasys Synergy Engine queries to find other agents, Neighbourhoods and Social Organisms that provide matching data and potentially embody the same Social DNA. Read below for an in-depth description of the Synergy Engine.

SDNA Subject Classes

As we explored earlier, ADAM and Social DNA provide a flexible and semantically rich environment for representing and reasoning about data. To make this environment **even more accessible and useful to app developers**, we have introduced the concept of "**Subject Classes**".

Subject Classes in Social DNA are akin to class definitions in Prolog. They represent ontology classes, complete with properties and collections. However, Subject Classes extend beyond traditional class definitions, incorporating the principles of subject-oriented programming, which focuses on representing objects from different subjective viewpoints.

In essence, a Subject Class provides a template for overlaying a graph structure over a base expression. This allows the same base expression to be interpreted and treated in various ways depending on the Subject Class applied. This flexibility makes ADAM and Social DNA a highly versatile persistence layer for apps, as it can cater to diverse requirements and use-cases with ease.

Consider the following example:

```
subject_class("Todo", c).
constructor(c, '[{action: "addLink", source: "this", predicate:
"todo://state", target: "todo://ready"}]').
instance(c, Base) :- triple(Base, "todo://state", _).

property(c, "state").
property_getter(c, Base, "state", Value) :- triple(Base, "todo://
state", Value).
property_setter(c, "state", '[{action: "setSingleTarget", source:
"this", predicate: "todo://state", target: "value"}]').

property(c, "title").
property_resolve(c, "title").
property_resolve_language(c, "title", "literal").
property_getter(c, Base, "title", Value) :- triple(Base, "todo://
has_title", Value).
property_setter(c, "title", '[{action: "setSingleTarget", source:
"this", predicate: "todo://has_title", target: "value"}]').
```

It defines a Subject Class called “Todo” with two properties: “state” and “title”. The used predicates “subject_class”, “property” and “property_getter” etc. enable a generic way for ADAM and UIs to interface with these classes and reflect on their properties. We’ve developed tooling around Subject Classes that is designed to make it seamless for app developers to leverage this power. With these tools, developers can define and interact with their app-specific ontology based on ADAM's meta-ontology easily, thus ensuring interoperability. The class definition above, for instance, got auto-generated from the following JavaScript class definition through the use of ADAM class decorators:

```
@SDNAClass({ name: "Todo" })
class Todo {
    @subjectProperty({
        through: "todo://state",
        initial:"todo://ready",
        writable: true,
        required: true
    })
    state: string = ""
```

```
            @subjectProperty({
                through: "todo://has_title",
                writable: true,
                resolveLanguage: "literal"
            })
            title: string = ""
        }
```

The concept of Subject Classes shares a **close relationship with SHACL** shapes, a standard for validating RDF graphs against a set of conditions. Just like SHACL shapes, Subject Classes allow for the definition of triple-based associations which get grouped to form higher-level virtual objects, or “subjects". Tooling for compatibility with SHACL is not implemented yet, but planned.

This approach provides an ergonomic way for developers to define complex and meaningful relationships within their data, thereby enhancing the app's ability to participate in the collective sense-making facilitated by ADAM and the Synergy Engine.

As of writing of this paper, all aspects of the ADAM Layer described above are implemented in the current version 0.3.4 (released March 2023) and usable in an alpha-version capacity, i.e. tested with regression and integration tests and field-tested with Flux in small networks of up to 70 users. Work on the last piece of the ADAM Layer ontology, Social Organisms, is in progress.

Social Organisms: The Emergence of **Collective Agents**

In a world that is entirely agent-centric and distributed, collective sense-making and decision-making become crucial yet challenging to achieve. This brings us to the concept of "**Social Organisms**," also referred to as Collective Agents.

Social Organisms are a specific type of ADAM Neighbourhoods. They embody a group of agents that, due to their fixed and deliberately chosen Social DNA, can be perceived as a single agent from an external viewpoint.

This facilitates group coherence, enabling the collective to act and respond as one.

A key requirement for a Social Organism is the **capacity to collectively author Expressions**, much like individual agents. However, the expressions — especially link expressions and thus graph patterns — must be in coherence with the internal state of the Social Organism. This internal state is reflected in the data shared within its Neighbourhood.

To ensure this coherence, Social DNA must define the types of expressions that a Social Organism can create. These definitions depend on the internal state of the Social Organism, making it an integral part of the collective's decision-making process.

However, one key challenge remains: how can we verify from an external perspective that an expression, shared by an individual agent on behalf of the Social Organism, is indeed coherent with the Organism? This is especially complex given that we want to avoid providing unrestricted access to the internals or the Social DNA of the collective.

Several additions are envisioned to facilitate this:

1. **SDNA Flows:** similar to the "Subject Classes" specification above, a Prolog predicate schema is defined that enables the dynamic definition of information flows, mapping inputs of the Social Organism to potential outputs. As Social DNA, these Flow definitions can generally be instantiated on any incoming Expressions (of any ADAM Language), and a given flow can also constrain to which Expression it applies. Flows define states (mapped to the interval [0, 1]) through which the processed Expressions transition. For a given state, the Flow defines what actions are possible on an Expression that gets processed. In other words: SDNA Flows are like finite state-machines with a starting- and an end-state. Mounted into a Social Organism, the end state means (per definition) that the processed Expression may be shared (by any of the member agents) on the outside as an expression of the whole. Consider the following example, implementing a simple "Todo" Flow with three states ("todo", "doing", "done"):

```
register_sdna_flow("TODO", t).
flowable(_, t).

flow_state(ExprAddr, 0, t) :-
     triple(ExprAddr, "todo://state", "todo://ready").
flow_state(ExprAddr, 0.5, t) :-
```

```
      triple(ExprAddr, "todo://state", "todo://doing").
flow_state(ExprAddr, 1, t) :-
      triple(ExprAddr, "todo://state", "todo://done").

start_action(‘[{action: "addLink", source: "this", predicate:
"todo://state", target: "todo://ready"}]', t).
action(0, "Start", 0.5,
      '[{action: "addLink", source: "this", predicate: "todo://
state", target: "todo://doing"}, {action: "removeLink", source:
"this", predicate: "todo://state", target: “todo://ready”}]’, t).
action(0.5, "Finish", 1,
'[{action: "addLink", source: "this", predicate: "todo://state",
target: "todo://done"}, {action: "removeLink", source: "this",
predicate: "todo://state", target: “todo://doing”}]', t).
```

This example Flow can apply to any expression (flowable(_, t)) and defines three states 0, 0.5 and 1 through the existence of a Link with predicate “todo://state" to the targets “todo://ready", “todo://doing”, “todo://done”, respectively. The initial start_action adds a link to the processed Expression that puts it into the “ready” state (0). The “Start” action is defined to transition from state 0 to 0.5 and replace the link accordingly. The “Finish” action works analogously.

2. **Coherent Social Organism Expressions:** There needs to be an automatic process within the ADAM implementation that flags an Expression as something being said on behalf of a Social Organism, and that shows (to any agent, especially those outside of the Social Organism) if that expression *is coherent*.

For that we add two fields to the generic Expression class:

a) socialOrganismSource?: string to ExpressionClass itself, which just signals that this Expression (though authored by an atomic agent) is spoken on behalf of the given Social Organism - if this field is present.

b) socialOrganismCoherence?: number which gets added to ExpressionProof and is, similar to valid/invalid, a virtual property that the AD4M executor calculates and adds to the response sent to clients/UIs. The number is understood as a percentage with 0 meaning not coherent (yet) at all and 1 meaning 100% coherent.

When an Expression gets created on behalf of a Social Organism, all other agents within will get notified by that event, executed by the agent creating the expression since they seem to have an incentive to get their expression to reach a high coherence number soon. When those other members get notified of a new SO Expression, they will retrieve that Expression and check if it is something the SO is allowed to say by running the Organisms Social DNA Flow on it. If that Social DNA action/

query returns true, they will add their signature to that Expression. If it doesn't their will sign and add a refutation-entry to the Expression. The Expression's coherence-number is a property calculated from the ratio of signatures per number of agents in the Social Organism. The presence of one refutation will set the coherence to 0.

3. **SO Bootstrap Language:** To make this work, we add another bootstrap Language for Social Organisms, which stores expressions that represent SOs, similar to the Neighbourhood language (with "so" alias):

so://<hash based unique address>

This language will also store the list of members per each SO and their external (public) expressions with their signature/refutation lists. Since the interaction with SO Expressions, (creating, concurring, rejecting) and the management of SO members go beyond the basic Expression Language interface, a specialised interface for this type of Language is introduced:

```
export interface SocialOrganismAdapter {
    create(organismAddress: string, messageURI: string)
    concur(organismAddress: string, messageURI: string)
    reject(organismAddress: string, messageURI: string)

    signatures(organismAddress: string, messageURI: string):
Promise<string[]>
    members(organismAddress: string): Promise<string[]>

    vouchForNewMember(organismAddress: string, newMemberDid:
string)
    rejectNewMember(organismAddress: string, newMemberDid:
string)

    newExpressionCallback()
    newMemberCallback()
}
```

4. **Social Organism SDNA specs.:** We require the definition of a Social DNA / Prolog interface (i.e. predicates) which is ADAM's expectation against any Neighbourhoods Social DNA that enables it to get upgraded to a Social Organism. There two main functions that need to be enabled by this interface:

a) Selection of Expressions, or graph patterns, that are valid Expressions of the whole. That is: an *Output Expression Predicate*. The previously defined SDNA Flows provide a reasonable interface for the interaction inside the Social DNA. The selection of output Expressions

in most cases will be as simple as selecting some or all of the defined Flows to render expressions in their final states (=1) to be shareable.

b) Membrane, i.e. management of the member list. This will be predicates taking in a DID (agent address) and return true if that agent shall be added to or removed from the member list.

These Social DNA predicates will be queried automatically by the executor to implement the described process.

Web-B: The Application of ADAM

Intended as the base-layer for interoperable collective sense-making and the digital infrastructure for new and evolvable patterns of collective action and interaction, the ecosystem spawned by ADAM is referred by us as Web-B, relating it to what has been described as Game-B by others. The result of applying ADAM on an ecosystem-scale will not just be an iteration or an enhancement of the existing web or blockchain-based Web3, but rather a new paradigm that enables our techno-sphere to shift from mainly competitive to mostly synergistic communication and interaction patterns.

At its core, ADAM is inherently a social network. However, it is a social network with a significant departure from the traditional sense. It does not rely on a specific User Interface (UI) or conform to the design restrictions of established social media platforms. Instead, it represents a shift from data ownership and centralization to a truly agent-centric, decentralized social and knowledge network.

With ADAM, the social network is formed by the interconnections between agents and their perspectives, their information, as well as the relationships they form with others, the common Languages they speak, and the Social DNA they share. All these connections are built and stored on an open, shared foundation that goes beyond the boundaries of traditional UI, centralised databases and even application boundaries.

This approach represents a new kind of web - Web-B. It is a web where meaningful relationships, connections, and collaborative knowledge generation are the driving forces. In Web-B, every application becomes an interface into this shared pool of knowledge, tapping into the vast interconnected network that ADAM facilitates.

In this chapter, we will delve into the possibilities that this new paradigm opens, the profound changes it can bring about, and the immense potential it holds for shaping our digital future.

Flux: The First App of the New Web

As the inaugural application built on the ADAM infrastructure, Flux is a testament to the unique advantages of this new paradigm. You can try Flux (and thus ADAM) for free at https://fluxsocial.io.

Flux is a fully decentralized social toolkit that enables communities to coordinate with greater privacy, agency, and collaboration.

Here are some of the **key features** that make **Flux** an embodiment of the Web-B vision:

- **Highly Customizable and Composable:** Flux allows communities to compose dynamic experiences by building their own features, customizing their UI, and integrating their favorite third-party tooling.
- **Private, P2P Networks:** Flux allows for the creation of private, p2p networks that exist only between its members and remain independent from centralized servers or third-party entities.
- **Sovereign Data and Identity:** The information and identity you create is portable across all applications in the ADAM ecosystem.
- **Dynamic Communication:** Users can can create various channels including p2p messaging, forums, audio & video, and project management. This can evolve into any form a community desires.
- **Community Channel Types (app store):** This will allow users to install games, tools, and other apps made by the community.

Future implementations in Flux promise to enrich its functionality even further. Some of the exciting features on the horizon include:

- **Distributed Governance:** Flux plans to integrate ADAM's built-in Social Organism capabilities to implement DAO equivalents, facilitating the design of value flows within your social space.
- **Native Web3 Functionality**: Flux has plans to build native crypto features such as p2p crypto payments, wallet integrations, token gating, and more.
- **Integrations**: Flux is primed for integrations with any popular Web2 or Web3 services. Some examples include Discord or Telegram channel plug-ins, decentralized asset management, decentralized proposals and voting (Snapshot), viewing a contact list for crypto payments, and more.

Coasys app: The General Purpose Browser for ADAM

As we push the boundaries of the ADAM ecosystem, we're preparing to launch the **Coasys app**, a general-purpose browser specifically designed for ADAM data. Coasys builds on the experimental work undertaken with **Perspect3ve**, advancing the ideas and features into a more refined and user-friendly product.

Coasys aims to bring a holistic experience for interacting with Perspectives, Neighbourhoods, and Social Organisms within the ADAM network with the Coasys app. It will also serve as a host for the Synergy Engine queries, a key component of the ADAM infrastructure.

Coasys inherits core principles and ambitions from Perspect3ve, the experimental agent-centric browser and social network developed in line with ADAM concepts. Perspect3ve was intended to provide a visual and generic means for users to create and manage Perspectives, as well as to establish and modify expressions within these Perspectives. Coasys will continue this pursuit, providing an intuitive and comprehensive interface to the ADAM universe.

Much like its predecessor, Coasys will not only serve as a browser but also as a generic social app and collaboration tool. ADAM's inherent group semantics and spaces (Neighbourhoods) enable Coasys to facilitate group collaboration in a versatile manner. The capacity to interpret a Neighbourhood's Social DNA and adapt its UI accordingly allows Coasys to be tailor-made for any group's needs.

Building on the feature set developed for Perspect3ve, Coasys is designed to offer a wide array of functionalities:

- **Perspective Management:** Create, read, update, and delete Perspectives with ease.
- **Graph-Based Perspective View:** Visualize Perspectives in a graph-based view, making it easier to understand and navigate.
- **Expression Handling:** Facilitate the creation and linking of expressions.
- **Neighbourhoods:** Enable the publishing of Perspectives as Neighbourhoods and joining of Neighbourhoods.
- **Social DNA Prolog Rules:** Provide functionalities for creating, reading, updating, and deleting Social DNA Prolog rules.
- **Custom Expression Actions:** Customize expression actions to suit your needs.

- **AI Chat**: Leverage AI to streamline the creation of Social DNA and to interact with local data through a local run LLM.
- **Expression Filtering:** Filter expressions through the predicate `hiddenExpression(X)`.
- **Custom Icons:** Personalize expression widgets with custom icons.
- **Virtual Icons**: Utilise widgets representing complex graph patterns.
- **Peer/Friend View and Messaging:** Facilitate peer/friend view and direct messaging.
- **'Canonical' Neighbourhood View:** Display a 'canonical' view of a Neighbourhood.
- **Social Organisms Management:** Facilitate the creation of and interaction in Social Organisms.

In sum, Coasys aims to be a flexible and comprehensive tool, bringing ADAM's agent-centric vision to life and offering users a versatile platform to engage with the ADAM ecosystem.

Reusable Components

The nature of ADAM as a framework encourages and fosters an ecosystem where components are designed to be shared and reused across applications and user Perspectives.

Languages in ADAM are essentially tools for describing and interpreting data. Their flexibility and universality mean they can be reused in various applications, providing a consistent way to interpret and describe data patterns. This reduces redundancy and encourages interoperability between apps. Users can remix these Languages within their Perspectives, enabling them to create unique ways of viewing and interacting with data. The same set of data can yield different insights and functionalities based on how users choose to apply various ADAM Languages.

Social DNA serves as the blueprint for how data is organized and interacted within a community. Like Languages, Social DNA can be reused and remixed within different Neighbourhoods, allowing for consistent patterns of interaction to be applied across different groups. This encourages consistency while allowing for unique permutations based on the needs of each community.

The user interfaces (UIs) in ADAM is another reusable component. Users can choose with which UI they want to access their Perspectives,

Neighbourhoods and Social Organisms. Developers can create UIs that can be used across different combinations of Languages and Social DNA and even remixed by users within their own Perspectives. This flexibility allows for a high level of customization and personalization, promoting user engagement and ownership over their digital experiences.

This emphasis on reusability and remixability of components significantly lowers the barrier for co-creation. As components are reused and remixed, new applications, functionalities, and user experiences can be created with minimal effort and resources. New applications that provide new ADAM Languages, Social DNA and UI add components that can be remixed and used in the context of existing applications. This fosters an ecosystem where synergy and cooperation might outperform competition.

By leveraging the reusability of ADAM components, we enable a profound shift from a competitive model of software development to a cooperative one. *Simply by being apart of the ADAM ecosystem, everyone works synergistically, enhancing creativity, reducing redundancy, and creating a collaborative environment that benefits all users*. This model has the potential to drive a significant evolution in how software is developed, used, and shared.

Synergy Engine

So far, we have introduced the concept of the ADAM layer and how it acts as a fundamental shift in the way we interact with the web, transitioning from a traditional, centralised model to an agent-centric, distributed one. This paradigm shift spawns a new kind of distributed, semantic and interoperable web. It leverages our natural ways of human communication to form a vast, interoperable social network that is not bound by the constraints of specific apps or technological platforms.

However, this distributed nature of the ADAM ecosystem brings forth a **crucial question**: *how do we find data in such a network?* Unlike traditional, centralised systems where data is stored in a few controlled locations, the ADAM network—with its agent-centric design—decentralises data storage. Each agent, or user, retains ownership and control of their data. This unique arrangement ensures privacy and sovereignty over personal data but also presents a challenge when it comes to locating information.

This challenge, though seemingly technical, is intrinsically tied to the broader subject of sense-making. In a world where information is abundant and often conflicting, finding true, reliable information is not simply about accessing data; it's about understanding the sources of information and evaluating their trustworthiness. It is also about non-linear and semantic connections between data. Following Barbara Marx Hubbard's visionary descriptions of what is possible, we want to emphasise the important aspect of enabling more synergistic connections between existing social functions and organs and associate the introduced query mechanism to the term "Synergy Engine" introduced by her.

The **Synergy Engine** serves as the foundational tool within the ADAM ecosystem that facilitates the process of finding information while addressing the challenge of sense-making. It does so by creating a system that not only locates data, but does so by checking intrinsic properties of the information requested and also the trustworthiness (or other properties as defined in the query) of its source—thereby providing a more holistic and reliable approach to information retrieval.

In the following sections, we will delve deeper into the workings of the Synergy Engine, its query mechanism, and how it leverages the unique properties of the ADAM layer to promote efficient sense-making in the agent-centric distributed web.

Social DNA based queries

In the ADAM layer, a key component of its operational framework is the concept of Social DNA. Social DNA, essentially, is a collection of Prolog programs (facts and rules) that are used to define graph patterns in ADAM Perspectives. These rules specify the properties and interrelationships of the information within a perspective. In the context of queries within the ADAM network, Social DNA plays an integral role.

A query in Coasys' Synergy Engine is defined by a specific Social DNA. This implies that for a given (potential) result, which is essentially a snapshot of a Perspective, any ADAM node can validate its relevance by running the query's Social DNA against it in the ADAM Prolog engine. This provides an efficient and decentralised means of verifying the accuracy of a query result.

However, the utility of Social DNA extends beyond simple verification. Since it provides for a flexible and powerful way to objectively check the relevance of a given potential result, it plays a pivotal role in propagating the query through the ADAM network. Each agent in the network can receive a query and check against their own (private and shared) perspectives if they hold the requested data, i.e. match the pattern implied by the Prolog program. If a match is found, the data can be sent back as a result. If not, they can further propagate the query to their connections in the network. **This forms a distributed, peer-to-peer search mechanism that efficiently traverses the ADAM network, harnessing the power of collective intelligence to find and validate information.**

This process inherently decentralises trust, allowing each node in the network to verify results independently, based on the defined Social DNA of the query. By doing so, it reduces the reliance on a centralised authority or algorithm for verification and instead leverages the power of the network and the collaborative effort of the agents within it. This not only fosters a greater sense of collective intelligence but also promotes a more transparent, reliable, and efficient means of sense-making within the ADAM network.

Let's consider a simple example of a Social DNA rule in Prolog that could be used for a query in the ADAM network. Let's say we want to find an Expression of a given “Post” Language that has at least 5 "likes" by different agents, whereas a "like" is expressed as a reaction from the Flux’ ontology Language with “thumbs up”. The Social DNA could be written as follows:

```
popular_post(Post) :-
    languageAddress(Post, 'Qm123abcd'),
    setof(
      Author,
      link(Post, “flux://reaction", “flux://thumbs_up", Author, _),
      Authors
    ),
    length(Authors, NumLikes),
    NumLikes >= 5.
```

This rule can be read as “a post is popular if its an Expression of Language Qm123abcd, there is a set of authors who each have authored a link with that post as source, “flux://reaction" as predicate and “flux://thumbs_up" as target, and that set’s size is 5 or higher.

When this query is propagated through the ADAM network, each agent would verify the rule against each of their perspectives, or unions of perspectives, and if for any of these instances the Prolog engine can find a match for popular_post, they would return a snapshot of that perspective.

This simple example illustrates the utility of Social DNA based queries. They allow us to create arbitrary, semantic and specific queries that can be independently verified by each agent in the network. The use of Prolog for defining the Social DNA allows for the creation of highly flexible and expressive queries, leveraging its strength as a declarative logic programming language.

Social Stack

As queries propagate through the network, they trace a path through multiple agents. Each agent in this path represents a "hop" in the query's journey. Like a call-stack tracking the function calls of an execution thread in programming, **the Social Stack is essentially a record of this path, listing all the agents that the query has passed through.** However, the Social Stack goes beyond being a mere list of agents. Each entry in the stack is an ADAM link that describes the relationship between two agents: the agent who forwarded the query (the previous agent) and the agent who received it (the next agent). Importantly, each ADAM link is signed by the previous agent. **This provides a cryptographic proof of the interaction between these two agents and of their relationship.**

If an agent receives a query from one of their friends (together with the Social Stack collected up until that point) and is not able to resolve it with the data available to them, they can choose to pass it on to all or some of their friends. But before passing on, they have to create a statement in the form of an ADAM Link with them (their DID) as the source and the receiving agent as the target. The predicate can point to an Expression carrying information about the kind of relationships. That link has to be signed as an Expression itself and thus represents cryptographic proof that the source agent declares a certain relationship towards the receiving agent.

One critical feature of the Social Stack is its inclusion in query results. When an agent responds to a query with a result, they have to include the Social Stack as part of their response. An empty Social Stack, or one that can't show a connection between query source and the result providing agent will fail the built-in checks of query results independently of the content of the result—unless explicitly allowed in the query. Moreover, with the Social Stack a query can specify certain conditions relating to the stack that must be met. For instance, a query might require that all agents in the Social Stack receive a certain level of trust from their predecessor. This level of control over the Social Stack enables users to define more complex requirements with regards to the agents delivering valid results. For example, stating that each agent along the path of the query needs to have at least on "popular post" as defined above:

```
social_stack_hop(_,_,Agent), popular_post(Post),
expressionAuthor(Post, Agent).
```

Searching via other people's social graphs

This distributed search and query process is intrinsically tied to social connections, enabling a unique form of networked exploration. Instead of relying solely on algorithmic computation, ADAM leverages the social graphs of agents to facilitate information discovery. This approach of navigating through shared, interconnected knowledge landscapes enables the discovery of what could be called "social cliques".

Especially when running queries with complex and high-level semantic requirements, different results can be possible. The result delivered obviously depends on the data available to the agents asked. A social clique is a group of agents who, due to their interconnected relationships and mutual trust, tend to either have similar perspectives, or trust relationships amongst a

similar portion of the network—and thus yield the same results when a query is propagated amongst them.

This fact can be regarded as problem or as an advantage, depending on perspective. Excluding certain agents and results from the set of possible results might be what the querying agent wants (to exclude bots from providing manipulated results, for instance). It could also be a consequence of an agent being in a local (filter-) bubble.

From a systemic, holistic view-point, these cliques might even represent the harmonisation of shared perspectives within a specific community or network. However, as we move between different social cliques, we may encounter different results to the same query. This variability is a reflection of the diversity of perspectives and experiences across a distributed, agent-centric network. It acknowledges the fact that knowledge and understanding are not monolithic or objective but shaped by our individual and collective contexts.

The implication of these distinct cliques is profound. By observing the responses of different cliques to the same query, we can gain an overview of the social landscape of "memetic tribes"—groups formed around shared beliefs, perspectives, and interests. This approach not only fosters the discovery of knowledge and perspectives but also reveals the contours of the social terrain itself.

In both cases,

1. The querying agent wanting to include data outside their filter bubble, and
2. Running a “controversial” query multiple times from different source points in the social network to map the landscape of metic-tribes and filter-bubbles

it would be beneficial to have the ability to run a query starting from a foreign agent. Therefore Coasys intends to build infrastructure to find and connect with agents outside of the users social milieu, applying and supporting the formation of global public indexes for instance.

To cater for the holoptic approach of running the same query from multiple globally sampled starting agents in order to render a map of the memetic boundaries and partial shared perspectives within the collective network, a specific application is envisioned. In a public forum style app, Synergy Engine queries could be grouped by query-DNA and thus listing all different starting points and results. Query topics would become their own discussion threads, logging results and showing an overview of the collectively held data.

With well crafted, sophisticated queries, scientific research programs can be spawned and maintained on this infrastructure in a low-cost, distributed, and inclusive fashion.

In essence, ADAM's approach to searching via other people's social graphs is a mechanism for understanding collective sense-making. It moves beyond the individualistic approach to knowledge acquisition and embraces the complex, networked, and socially-informed nature of our understanding. By doing so, it offers a dynamic, comprehensive, and nuanced way to navigate the information ecosystem.

Privacy concern and solution

The nature of ADAM's query mechanism, which relies on agents sharing information through their interconnected networks, raises a privacy concern. Especially in the context of monetarily incentivised queries as discussed in the follow section. *The core of the concern lies in the potential misalignment of incentives: agents could be rewarded when sharing confidential data they have access to, but whose source wouldn't want that data to be shared in public query results*. In essence, this could inadvertently turn friends into spies, as they might share data that was initially communicated in a private context, such as a direct message, private Neighbourhood, or within a Social Organism.

To address this, ADAM will be extended with a systemic solution: **the inclusion of a watermark in every ADAM Expression**. This watermark, included in the signed bytes of every expression, serves as a privacy membrane, preserving the integrity of private information by representing the sharing context and intended visibility rights. If an agent attempts to share data marked with a watermark in a query result, they would reveal their break of trust publicly and the agents whose privacy was compromised and trust broken will likely downgrade their trust towards the misbehaving agent and exclude them from Neighbourhoods and Social Organisms. Removing the watermark prior to sharing private information is not possible without breaking the signature which renders the data unusable in query results.

Moreover, any data shared within a particular perspective that is to be used to fulfill a query must be collectively submitted by the Social Organism that holds that data. The watermark in the link expressions indicates the ownership of that data, and only the Social Organism with the correct watermark can validate and submit the query result. Individual agents who

attempt to provide the data will fail the query result check, thus maintaining the privacy membrane.

This mechanism ensures that data privacy is respected within the ADAM network. It provides a safeguard against potential misuse of confidential information, thereby reinforcing trust among agents and ensuring that privacy concerns do not undermine the network's functionality or credibility.

Relationship to AI and LLMs

In this context of the Synergy Engine and distributed, semantic queries on ADAM, we witness a thoughtful blend of artificial and collective intelligence. The vision for ADAM involves a thriving ecosystem of applications that agents use to fuel their privately held perspectives which the owner can then choose to include in the creation of query results. When these myriad data points are accessed through a query, **the quality of responses can, at some level of scale, challenge those produced by centralised large language models (LLMs).**

The key principle here is that collective intelligence, the cumulative and coordinated wisdom of a group, has the potential to offer more authentic and trustworthy results than a centralised AI model. This perspective marks a significant departure from current models where large, centralised entities control most of the information.

However, this doesn't suggest that AI or LLMs are rendered irrelevant within the ADAM ecosystem. In fact, LLMs have a crucial role, specifically in translating natural language queries into the Prolog-based Social DNA queries utilised by ADAM. *Coasys plans to train small enough models to run locally within the ADAM runtime to interface with local Perspectives and to—especially— create Social DNA from natural language.*

This is where the Synergy Engine truly shines: it enables users to interact with the ADAM network as if they were querying a centralised AI model. The core difference lies in the source and reliability of the information. Here, query results are derived based on human trust relationships and up-to-date, distributed data sources, offering not only data but a sense of confidence in the integrity and authenticity of that data.

In essence, ADAM integrates AI, not as the primary source of information but as a powerful tool to enhance the usability and efficiency of the user experience. It combines the convenience of AI interfaces with the

trustworthiness of collective intelligence. Thus, while users can interface with the system as they would with a large AI model, the responses they receive are more reliable and grounded in the reality of human trust relationships.

SynergyFuel

In our exploration of the ADAM ecosystem and the Coasys Synergy Engine so far, we have considered how agents can interact, share, and access information across the network. But an important question remains: *why would agents actively participate in sharing or relaying queries from others at all?* What is the incentive for users to fuel the network with their data and efforts?

The answer lies in SynergyFuel, a unique form of cryptocurrency specifically designed for the ADAM ecosystem and for the distributed validation of query results. **Agents will be able to use SynergyFuel to pay other agents for the relaying and resolving of their queries.** The implementation of SynergyFuel will be intrinsically linked to the interaction pattern of posing and resolving queries. Similar to the concept of smart contracts in data-centric blockchains, SynergyFuel will enable locking-in of the query reward with the query-DNA code. Only a matching query result will be able to unlock those funds and earn the reward.

SynergyFuel will be implemented as a mutual-credit, asset-backed currency on Holochain. This will enable the integration of a Prolog engine into the currency code for result verification and the construction of an intrinsic value of this currency: the network's capacity to meaningfully respond to semantic queries based on real human trust relationships.

In this manner, SynergyFuel creates a symbiotic relationship between individual user engagement and the overall health and effectiveness of the network. The incentive to earn, hold, and utilize SynergyFuel drives users to contribute to the network, thereby enhancing the network's collective intelligence and capacity for sense-making. The intended outcome is a vibrant, self-sustaining, and evolvable ecosystem that enriches all participants while continually growing in value and capacity.

Holochain based mutual-credit currency

SynergyFuel, as an agent-centric mutual-credit, asset-backed currency, operates on *principles quite similar to those of HoloFuel*, another cryptocurrency built on the Holochain framework.

HoloFuel is a mutual-credit currency, which means that every unit of currency in existence is accounted for as a credit to one account and a debit

to another. There is no central authority issuing new currency; rather, currency is created and destroyed through transactions between users, in a validated way. This arrangement ensures that the total amount of HoloFuel in existence always remains at zero, but the amount of units in circulation can adjust in a kind of homeostasis, only based on the distributed validation rules of the network. In the case of HoloFuel, it is based on the computing capacity provided by users: if an agent can (cryptographically) prove that they offer computational resources to the network and have created an income in HoloFuel with it, the distributed validation rules allow other agents to grant them a (higher) credit line, thus adding currency units to the effective supply which adequately represent the newly added computational capacity of the network.

Similar to HoloFuel, SynergyFuel operates on a mutual-credit system. However, SynergyFuel's value is not tied to general computational power, but to the network's capacity to resolve semantic queries effectively and with the ability to reason about the confidence put into the result and its author. It's a system that incentivises users to contribute to the network's collective intelligence, by 1) responding to queries, but also 2) by relaying queries to people they trust. In order to achieve a similar homeostasis between the amount of currency units effectively in circulation and the network's capacity to resolve queries, agents' credit limits will depend on their participation in the query process and thus to their average income through query relaying and resolving.

Note that this kind of agent-centric mutual-credit currency is very different to data-centric blockchain tokens. Without Holochain and its validating DHT, it's hard to imagine such a currency design to be viable to begin with. The reason for choosing this currency design is simple: *it is the one that best matches the need of a real-world, non-corruptible, distributed and growing ecosystem*.

Let's briefly contrast the alternatives: Bitcoin, and most other crypto-currencies, take the approach of having a static or at least predefined supply —they are artificially made scarce to have value. While this rewards people who buy-in early, it can become a problem to the growth of the ecosystem later-on when there is more real-world value than there is liquid currency. This deflationary situation limits growth of the ecosystem as new users will have to buy-in from earlier buyers. While this rewards early movers for just holding the currency, it slows down adoption. This problem could be solved if we would detect such situation and then mint new currency units— basically following the schema that fiat currencies and central-banks are implementing. The problem with that is: who should have those systemic

powers? If we allow such a role it introduces central control, corruption and the problem of overcorrection since this central bank would need to have omnipotent analytical powers to adequately adjust the supply (a problem we arguably see perpetuated in the end-phases of economic systems).

This new model of Holochain based mutual-credit, asset-backed currencies solves all these problems since it does *not only* distribute the power over existing currency units to the users, but it also allows for a fully *distributed adjustment of the total supply*, based on the state of the network. With Holochain's high degree of freedom with regards to specific validation rules, this "state of the network" can be a complex, high-level characteristic—like in our case: the networks ability to respond to semantic queries.

Query Vaults

Holochain's novel, agent-centric architecture is instrumental in implementing not only the basic currency model applied with SynergyFuel, but also enabling vaults, i.e. programmable, agent-centric smart contracts, that are validated by complex Prolog queries, i.e. ADAM's Social DNA (without requiring external oracles). By integrating a Prolog engine into SynergyFuel's validation rules, it's possible to implement 'vaults' that unlock SynergyFuel units based on the validity of a query result.

Agents initiating a query will create a vault (inside the SynergyFuel hApp) that holds the query-DNA and the reward they have selected to spent on that query. Since we are talking about an agent-centric Holochain app, we will have to explain what we mean by "the vault holds SynergyFuel". An agent selecting an amount of SynergyFuel as reward will earmark that amount on their source chain, making it impossible to spent it otherwise. That mark is an entry on the agent's source chain as well, which will reference the vault entry in the DHT.

Vaults will have validated links to potential query results. When an agent has found/constructed a matching result, they can post it to the vault's results-list. That will trigger the validation of that result entry which will run the given Prolog program (the query DNA) against the result within Holochain's DHT validation. If the result is valid, it will change the vaults status to "resolved". This will have a consequence on the querying agent's ability to spend further SynergyFuel. If an agent has an amount earmarked for spending through a vault (which already can't be moved) and that vault is resolved but the earmarked amount is not yet sent to the agent who provided the result, that agent's source chain is frozen. That means, SynergyFuel

validation rules will make it invalid for other agents to accept money from them, until they pay the reward. In the Coasys app and the SynergyFuel wallet app, this will happen automatically anyways, but this validation scheme makes it impossible to cheat even if these apps or even ADAM or Holochain nodes get hacked.

Only these payouts of vaults will count as income for the agent receiving the reward in the context of calculating their credit limit.

Intrinsic Value of a Currency Unit

One of the defining aspects of SynergyFuel is the intrinsic value of each unit of currency. This value is intrinsically tied to the ADAM network's ability to facilitate collective sense-making, **and it is measured in terms of "hops."**

A "hop" is a step along the path that a query takes through the social network. As a query travels through the network, it moves from one agent to another, or "hops" from node to node. Each of these hops represents a discrete unit of effort—the presence of utilisable trust relationships and is this effort and trust that SynergyFuel is designed to incentivise, reward, and represent.

The cost of these hops is paid out to the agents along the path of the query. The agent providing the final result only receives the remainder of the reward, after the cost of the hops has been accounted for. This system ensures that every agent who contributes to the propagation and resolution of a query is appropriately rewarded for their contribution.

Each hop in the query's path includes an addition to the social stack, and the agent responsible for that addition is free to set a cost for their hop—tracked in that entry in the social stack. **By default, the network will set that cost to 1 unit of SynergyFuel per hop**, but agents can adjust this cost as they see fit. Also queries can set bounds on hop prices or grant higher prices only to certain hops, relationships, or relaying agents. This provides a degree of flexibility and autonomy to individual agents, enabling them to negotiate their value within the network.

However, this also introduces a dynamic element to the system. As the social stack grows and more SynergyFuel is "virtually spent," the potential reward for later agents in the chain decreases. If the remaining reward becomes too small, agents may choose to stop propagating the query or choose to (be configured to) not respond with data that would resolve the

query. This mechanism helps maintain balance within the network, ensuring that the effort required to resolve a query is always appropriately rewarded.

Payouts to hops only occur when that path leads to a valid result. Or put differently: only valid end-results can trigger payouts—but if that happens, all agents in that results social stack will benefit. This means that hops are only of value if they lead to the requested result. Basing the value of SynergyFuel hops, thus, results in SynergyFuel be intrinsically bound to the networks capacity to resolve semantic queries based on trust relationships between agents.

Earning SynergyFuel

The SynergyFuel ecosystem is designed in such a way that every interaction within the network becomes an opportunity to earn. Fueling data into private ADAM perspectives increases the probability of earning rewards. This is essentially akin to mining with your private app data; **every ADAM app becomes a source of potentially monetisable data.**

By engaging with the network and connecting with other agents, users can further increase their chances of earning SynergyFuel. Every interaction, every connection, and every contribution to the network could potentially be rewarded. This provides a strong incentive for participation and encourages users to actively engage with the ADAM network.

Moreover, this system incentivises the use of ADAM apps. By simply using these apps and contributing to the network, users can earn SynergyFuel. This not only encourages the adoption of ADAM apps but also promotes the development of new ones. Developers are incentivised to build ADAM apps that focus on storing meaningful data and storing it in a semantically interconnected way.

By ensuring that data is stored in a manner that enables Social DNA to detect and reason about it, developers can create apps that contribute significantly to the network's collective sense-making capabilities. *The more valuable and interconnected the data, the more opportunities there are for earning SynergyFuel.* **This creates a positive feedback loop that benefits everyone involved—from individual users and app developers to the ADAM network as a whole.**

The ADAM layer's Synergy Engine and its query mechanism address the challenge of finding data in a decentralized network by leveraging Social DNA and Prolog-based queries. However, this raises the question of information fabrication. To counter this, two strategies are being used.

Firstly, Social DNA empowers agents to define complex queries using Prolog logic. For instance, an agent looking for events in Bristol could specify that each event must be authored by an agent with at least two endorsements from their friends. This not only promotes authenticity but also helps ensure that the information retrieved is trustworthy.

Secondly, agents can employ double signing as a strategy for ensuring legitimate results. For example, if an agent is searching for a taxi service, they could specify that the query reward will only be given to results that they sign as legitimate. With this approach, the release of funds can be held in custody until the query maker enters the taxi and verifies the result, thus lowering the incentive for fabricating false taxi offers.

These mechanisms, in tandem, provide a reliable and efficient approach to information retrieval that fosters transparency, authenticity, and trust within the agent-centric ADAM network.

Funding Campaign

In order to support the development and growth of the Synergy Engine and ADAM, we are launching a funding campaign. This campaign will involve a pre-sale of an **ERC-20 Synergy Token**, which will be swapped on a 1:1 basis with **SynergyFuel** once the Synergy Engine v1.0 goes live (test and beta networks might be test-run with the ERC-20 Synergy Token before the swap).

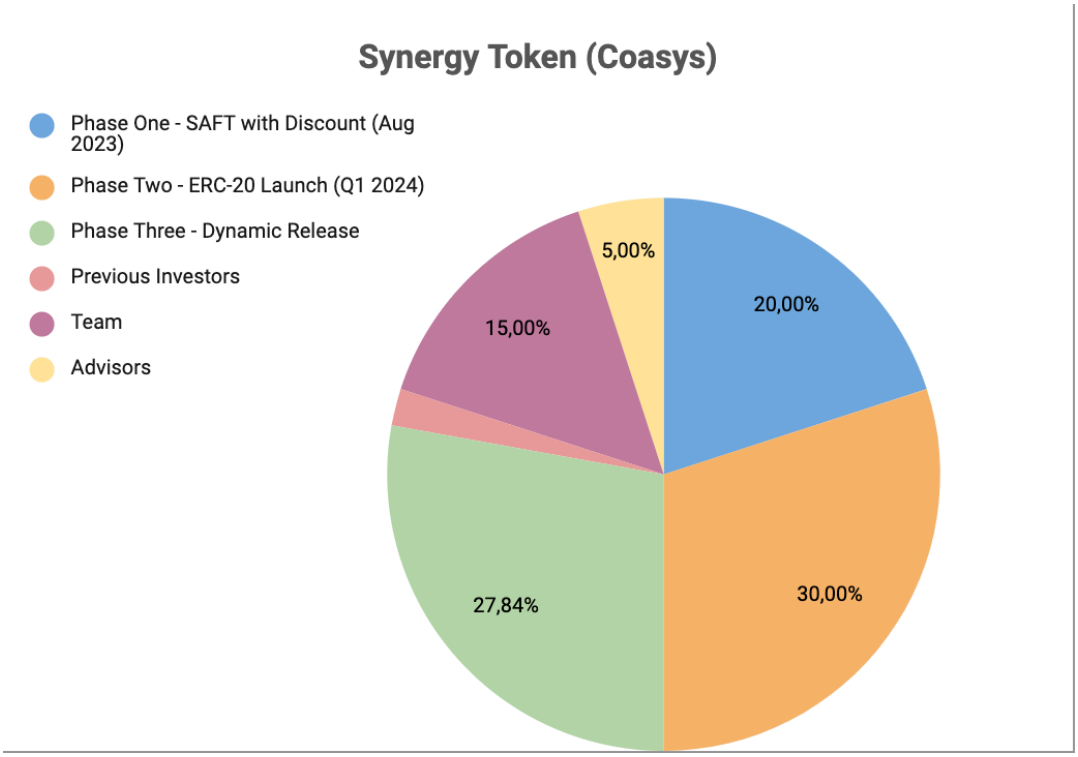
Phases of the Pre-sale

The pre-sale of SynergyFuel will occur in several phases:

1. **SAFT with Discount**: In the first phase, we will conduct a pre-sale of Simple Agreement for Future Tokens (SAFT) with a discount, for up to €3 million. It starts on September 12th and ends December 15th 2023 or when the goal of €3 million is reached.
2. **ERC-20 Launch**: In the second phase, alongside the launch of the ERC-20 Synergy Token and the release of an early test-net, an additional €6 million worth of tokens will be made available without discount.
3. **Dynamic Release**: The third phase will involve a dynamic release of tokens based on certain factors, which will be disclosed closer to the time of the phase. We aim at timing the start of phase 2 concurrently with the release of an early test-net, in which the ERC-20 token can already be used to gain access to test tokens. Actual usage data of that test-net might indicate that the release of more tokens might be justified in order to approach the supply dynamics of the full mutual-credit implementation of SynergyFuel.

Supply and Distribution

The Synergy Token will be issued with an absolute maximum supply of 2 billion tokens. This figure has been chosen to ensure ample liquidity in the system and to allow for micro-transactions, such as hops, which we aim to be valued around €0.01 each. Those 2 billion include the upper cap for the dynamic and potential supply of phase 3—it might be less.



The initial two phases of the pre-sale will aim to raise a total of €9 million:

1. **Phase One - SAFT with Discount**: The first phase aims to raise €3 million. Tokens will be sold at a discount, starting with 25%, effectively pricing each token at €0.0075. Thus, we will distribute a maximum of **400 million tokens** during this phase, to be released upon TGE with no vesting requirements. The discount will decrease over time, dropping to 20% starting November 1st 2023, and dropping a second time to 15% on December 1st 2023.
2. **Phase Two - ERC-20 Launch**: In the second phase, we aim to raise an additional €6 million. The token price will be set at €0.01, the nominal price, leading to the distribution of another **600 million tokens**.

In total, **these two phases will distribute 1 billion tokens**, which is **50%** of the maximum total supply.

3. **Dynamic Release**: Depending on usage statistics and other performance indicators yet to be defined, a variable amount of tokens, but **no more than 556.8 million additional units**, might be released for sale during the third phase.

Together, we get the following overview:

Bucket	Ratio*	Synergy Tokens
Phase One - SAFT with Discount	20.00%	400M
Phase Two - ERC-20 Launch	30.00%	600M
Dynamic Release	27.84%	556.8M
Team	15.00%	300M
Advisors	5.00%	100M
Previous Investors	2.16%	43.125M

*given maxed-out Dynamic Release

Earning Back Negative Balance

When transitioning from ERC-20 token to Holochain-based SynergyFuel, Coasys will accrue a large negative balance in this mutual-credit currency as it has to go negative in order to sent out currency units during the swap. Coasys intends to earn back this debt by taking part in the ecosystem as a data provider. Since the credit-limit granted to Coasys (within the mutual-credit currency implementation) during swap is a needed exception, currency units earned back will effectively be taken out of supply, i.e. this will be equivalent to the burning of tokens. This will be a strategic tool to ease the transition during a phase in which the ecosystem and other data providing agents and apps are still being built-out.

Allocation of Funds

The funds raised from the token sale will be used for the following purposes:

- 30% Building the SynergyFuel currency and the Synergy Engine.
- 25% Further development of the ADAM layer.
- 25% Development of ADAM apps (Flux and more).
- 20% Growth and expansion of the ADAM ecosystem (documentation, education, events).

Through this funding campaign, we aim to accelerate the development of the Synergy Engine and ADAM, bringing about a new era of collective sense-making on the internet.

Milestones

Our road map is carefully designed to ensure that each step forward leads to a meaningful development in the project and adds value to our community. Here's an outline of the major milestones we plan to achieve:

1. **Completing ADAM Documentation:** The first step is to complete and polish the ADAM documentation. This will pave the way for more ADAM applications to be built by giving developers the knowledge they need to create innovative and useful apps.

2. **ADAM Hosting and Remote Access:** To close a significant UX gap, we will build scripts and UI flows to host an ADAM agent on any machine, access your ADAM agent remotely from another machine, as well as remove/revoke access. This allows a user to access ADAM applications at all times, via proxy from any device.

3. **Making Flux Production Ready:** We intend to make Flux production-ready and transition our team's operations from Discord to Flux. This will enable us to constantly identify and resolve any bugs or issues in Flux and/or ADAM, ensuring the platform's stability and reliability.

4. **Social Organism Implementation in ADAM:** The next significant task is to implement Social Organisms in ADAM. This represents the final piece of the ADAM ontology puzzle, marking a significant development milestone, allowing Flux and other apps to map organisational hierarchies within the ADAM layer.

5. **ADAM AI:** We plan to train small-scale Large Language Models (LLMs) on local ADAM perspective data and Social DNA. This step will enable natural language interfaces for all local ADAM data, significantly improving usability and accessibility.

6. **Coasys App:** We will develop the Coasys app, a general-purpose ADAM app and browser for Neighbourhoods and Social Organism. This will serve as the central hub for users to interact with the broader ecosystem.

7. **Synergy Engine MVP:** The next step involves integrating query distribution, relaying, and automatic answering into the ADAM runtime. Concurrently, we plan to build a centralized mock version of SynergyFuel, enabling us to conduct early full integration tests.

8. **Introduce ERC-20 SynergyToken:** We plan to introduce the ERC-20 SynergyToken and commence the second phase of the presale. The Synergy Engine MVP will provide utility for the ERC-20 token from day one by enabling staking of the token to retrieve test tokens from the centralized mock version of SynergyFuel.

9. **Build Partnerships:** As the ecosystem grows, we aim to build partnerships with projects that are building ADAM apps and specialized Synergy Engine query DNA for specific use-cases. These partnerships will help diversify the ecosystem and introduce innovative solutions to our users.

10. **Implement full Holochain based SynergyFuel:** The final milestone is the full implementation of the Holochain-based mutual-credit currency SynergyFuel with Social DNA based vaults. We plan to conduct rigorous testing and code auditing before transferring the ERC-20 token to SynergyFuel, effectively completing the scope of this campaign.

Each of these milestones is a stepping stone towards our ultimate vision of a self-sustaining, agent-centric, and decentralised knowledge ecosystem. We look forward to your support as we embark on this exciting journey.

Financial Sustainability

Coasys is aware of and interested in exploring various income streams as the ADAM network develops in order to sustain the development of the ADAM layer, security components, and the Coasys application. We intend to develop a revenue model based on one or more of the following potential income streams:

1. **Fees on Synergy Engine Queries:** When a user writes a query for the Synergy Engine, they would be making a micro-transaction that compensates the other agents that pass along their query and to any agent that resolves it. Coasys could simply take a small percentage of SynergyFuel as a query fee for all queries that are resolved.

2. **Centralized ADAM Agent Hosting Services:** One of the major UX challenges in the distributed, p2p internet space is the hosting of applications and data in such a way that allows easy and always-available access to the user, while retaining full sovereignty and ownership of data. ADAM's agent-centric architecture allows a user to host their ADAM agent on any device. Coasys will build tools to support remote access to ADAM and the possibility of providing a centralized hosting service, allowing the convenience of centralized hosting while retaining the autonomy over one's data.

3. **Application Marketplace:** We anticipate a fast-growing interest from developers in creating new ADAM applications, as the process to do so is extremely efficient with reusable components and a decoupled front/back-end. We hope to facilitate an open database of applications that anyone can use; however, doing so may create a challenge for users in knowing which applications are safe, secure, and effective. Coasys will likely provide support in identifying ADAM applications that meet a certain criteria for being considered safe and secure, as well as UI for a users to browse, sort, and find new applications. We see various income opportunities for Coasys in facilitating this process.

Company / Team

The Coasys project is stewarded by a **non-profit DAO LLC**, incorporated in the Republic of Marshall Islands. This structure provides us with the necessary flexibility to work in a rapidly evolving technological and regulatory environment, allows opportunities of co-governance, and ensures our activities are fully aligned with our mission.

We believe that the Synergy Engine and the ADAM network are valuable public goods that can transform the information ecosystem, and they should be maintained as such. A non-profit would ensure that these resources are managed for the benefit of all users, rather than for private gain. Our structure as a DAO also offers a clear path to decentralizing the governance of these assets more and more over time. By putting the maintenance and development of the ADAM layer and the Synergy Engine in the hands of a non-profit DAO, we can ensure that our technology stays true to its mission of fostering a healthier and more trustworthy information ecosystem.

The DAO smart contract is deployed on the Ethereum blockchain at address: 0x7dF4fD70a2aD7c520f1d3c056D07d83c3442cF96 (ENS: coasys.dao.eth). Ultimately, the goal is to transition our non-profit DAO LLC's infrastructure from an Ethereum based DAO over to an ADAM Social Organism as soon as the implementation is finished and tested.

Team members

(Alphabetically)

Moritz Bierling

Universal amateur, network operator, and technician of the soul. 9+ years living and working between domains, industries, and organizations. Writes, speaks, programs, organizes, facilitates, bootstraps, supports, and so much more.

Dora Czovek

Administrative operator with 15+ years of experience.

Bård Hovde

Senior frontend engineer with 13+ years of experience. Previously worked for some of Norway's largest companies.

Nicolas Luck

Co-founder and Chief Architect of Coasys and ADAM. Senior developer with 20+ years of experience. Former lead developer of Holochain core. Developer at Holo Ltd., leading the implementation of their €20m ICO.

Tomis Parker

Co-founded the Agile Learning Centers, a self-directed education framework used by 70+ communities worldwide. Partner and Facilitator with Grow Dialogue, an organizational culture consulting firm serving Fortune 100 companies.

Joshua Parkin

Co-founder of Coasys, ADAM, and Flux. Senior developer with 9+ years of experience. Previously CTO at an AI company.

Fayeed Pawaskar

Senior level frontend engineer with 5+ years of experience in web and mobile development.

Leif Riksheim

Leads Coasys frontend development. Senior engineer with 9+ years of experience. Previously worked for some of Norway's largest companies.

Eric Yang

Co-founder of Flux and leads Coasys strategic partnerships and fundraising efforts. 7+ years of experience at the intersection of product design, finance, and engineering.

Advisors

(Alphabetically)

David Atkinson

Arthur Brock

Andrea Harding

Bret Warshawsky

Harlan Wood

Partners

Cacoon (https://www.thecacoon.com/)

Corenexus (https://corenexus.is/)

Holo Ltd. (https://holo.host/)

Symphonics (https://www.symphonics.life/)

TrustGraph (https://trustgraph.net/)

we { collective } (https://weco.io/)

Acknowledgements

The work presented in this paper draws from previous and related work and conversations with other visionary creators in the field. These key influences include, but are not limited to, knowledge sharing between the respective teams or team members listed below.

Arthur Brock's and **Eric Harris-Braun'**s work on Holochain forms a basis in ontological approach but also as a pragmatic enabler of the concepts and solutions that are in many cases built/designed on-top and around of Holochain and its paradigm-shifting approach. Also their work on CEPTR and a protocol-for-protocols is somewhat related to ADAM's positioning as a meta-ontology.

Symphonic's, and esp. **Andrea Harding'**s, conceptualisations about Social Organisms and "Organomics" have been a guiding framework and targeted application for the technological infrastructure described by ADAM and the Synergy Engine. **Bret Warshawsky** has played a vital role in connecting this work with other related projects and providing crucial inspiration by pointing out synergistic associations.

Many conversations and collaborations between **Chris Larcombe** and Nicolas Luck in the years between 2016 and 2019, in the context of the projects Noomap and h4ome have laid important groundwork and framing for an approach that goes beyond apps and allows for a different kind of data and app architecture which doesn't foster or create data silos. In several ways, ADAM is a rethinking of those shared ideas from a purely agent-centric, Holochain based approach. Chris is continuing to work on related solutions in the context of his project called **HoloWeb**.

The **Neighbourhoods** project is approaching a similar shift of replacing apps with social spaces, i.e. Neighbourhoods. The usage of the term "Neighbourhoods" in ADAM (for mutably shared perspectives) goes back to **Siddharth Sthalekar's** generic definition of this term as "generic tech & specific culture", as an antipode to the common Web 2.0 pattern of using the same app for many different cultures "specific tech & generic culture" - of which ADAM's shared perspectives is a specific implementation of.