Tiamat: Transaction Chaining Layers for Cardano

Chris M. Hiatt
chris@metadex.fi

November 30, 2024

Abstract

We introduce Tiamat, a DApp development framework centered around the use of decentralized proof-of-stake systems to facilitate secure chain interactions, mempool consensus management, and, above all, the intense usage of transaction chaining. With Tiamat, DApps are enabled to deploy their very own Transaction Chaining Layer, and subsequently decrease reliance on centralized chain API services, and increase throughput of naive business logic implementations from around 20 seconds to sub-second timeframes per action. DApps based on Tiamat are instantly interoperable with other DApps on the Cardano Mainnet as well as DApps running on separate Tiamat instances. This is in stark contrast to more heavy-weight solutions based on isomorphic state channels or rollups, which aim to solve a different, and larger, set of challenges. Further, Tiamat includes convenient systems for creating state machines, complex data processing pipelines, data verification and serialization, as well as property testing. We hope the accompanying open source release of the full Tiamat codebase will inspire more developers to contribute to the project as well as the Cardano ecosystem as whole.

Contents

1 Introduction

One of the biggest strengths of Cardano - the extended UTxO model - at the same time constitutes one of its biggest weaknesses. While being elegant and incredibly useful from a technical standpoint, it buys this advantage by demanding more from individual DApp-developers aiming to build on it.

There is a very good reason Bitcoin was built on UTxO, and there is also a very good reason Ethereum opted for the account-based model instead - the latter is far closer to the way most programmers conceptualize the world, which is largely via objects, or, in other words, state machines. There is no need to elaborate on the drawbacks of that choice here, however.

Yet one of the consequences of the fact that the EUTxO model demands more from developers is that not only does it harm the influx of new developers into the ecosystem by severely punishing their naivete; it also means that the ones that do stay are not only slowed down by the requirement to develop their own scaffolding around that model before they even can start thinking about the actual implementation of their business logic; they also are at risk of developing subpar solutions.

This is rather tragic, as, as aforementioned, the EUTxO model is real fucking cool and useful in many ways. Again, this is not the place to elaborate on that; as a teaser, determinism and parallelizability ought to at least be mentioned though.

To add insult to injury, many of the solutions to that delta end up twisting the pristine back into the profane, because most are familiar only with the latter. This is still a large achievement, and not always avoidable.

However.

In this paper we describe a method that aims to unlock the native strengths of Cardano, allowing developers to go back to being naive, and implementing the business logic as they see fit, without pondering the boilerplate requirements for longer than they should have to. In particular - besides providing a framework for the implementation of EUTxO-based state machines, a very common design pattern that most Cardano DApp developers end up having to reinvent themselves - the core contribution of Tiamat is a framework that aims to unlock transaction chaining, a feature that has been around for some time, yet remains criminally underused for various reasons, which we aim to address.

As a side effect of our approach we provide a more decentralized chain API - a factor that is often overlooked by the community. But ask yourselves: Where does the chain data come from, that gets displayed in your frontend? Where are the signed transactions being sent? Where do whores go?[1]

The core of the solution revolves around the need to maintain a sort of consensus about the part of the mempool pertaining to the contracts in question, and prevent bad actors from "cutting in line" and therefore breaking the transaction chain. We tackle it by establishing a decentralized proof-of-stake system of nodes for that exact purpose.

Besides that, we also provide smaller subsystems to handle complex data processing, onchain data verification, data serialization, and property-testing.

We want to highlight again that the aim here is to provide code for a DApp-specific layer, not a layer. We aim to enable developers, not pull developers into our specific Layer 2 instance (yet). Each DApp has to provide their own token to back their own instance of said proof-of-stake system.

Finally, we outline a method to very easily augment the system to serve as simple Cardano sidechains, and our plans regarding the development of the ecosystem.

But we will start with providing historical context for the development of Tiamat, describing some technical preliminaries, and outline more or less related solutions we have seen thus far.

[1] You may ignore the last one. Don't overthink it neither.

2 Historical Background

When we first joined the Cardano community in summer of 2020 - right at the start of the Shelley era, when we deployed our own single stake pool, which has been in operation since then - things looked bright on the horizon. We were to get smart contracts in November 2020 at the latest, and the slow and steady approach was bound to finally pay off. As it turned out later, it was more on the slow than then steady side.

Nevertheless, we took the delay as a boon. Starting in January 2021, we set out to build a simple Uniswap clone on Cardano, a small two weeks project to get the blood flowing. That didn't quite work out that way, mainly due to our own unreasonable perfectionism. Because we proceeded to do math, and more math, and even more of the same, to try to unify ALL DEXes that currently existed on Ethereum, with one Swap to find them, and eternally bind them. We figured out how to do concentrated liquidity multi-asset pools with impermanent loss insurance and optimal trade split (which we almost managed to do in a closed form - this is where memory gets hazy however, so don't quote us on that). All in one.

One day after the Ides of March 2021 - the traditional festivities of Mercury, Roman God of Trade - we officially announced the project in the form of Mirqur.io - oriented on one latin spelling of His name, Mirqurios. Later, we rembranded to MetaDEX.fi, as, as we were informed, not everyone was able to utter the God's name properly, and, as was further pointed out, no one will understand if we don't have "Swap" in our name. As a compromise, we settled on the all-caps "DEX" in "MetaDEX"[2].

The project grew quickly, as did the team. But then, as some of you might remember, the Eclipse happened, and in Femto's Grand Birthing Ceremony, the Band of the Hawk perished. Or to put it in less dweeby terms: It turned out that simply understanding Haskell, Plutus, and the EUTxO model wasn't enough. One needed a way for the DApps to talk to the chain, to compose transactions. Back then, there was no way to do such a thing in the frontend; one needed to rely on the PAB[3], the Plutus Application Backend.

Our more sane competitors went with it, and said alright, let's run with it. Nay, said

[2]The "Meta" part came from the intention to develop into a DEX-aggregator ASAP. Haha. Ha.
[3]Trigger warning!

we[4], we are NOT running a centralized server like those fascists![5] We Will Figure Out How To Do Everything Decentralizedly All The Time.

As the reader might imagine - or remember - that didn't quite go so well. Half the team left, and the other half was let go. "We need to figure out the tech first, and until then, we don't need design, community management, or frontend development".

And thus, the Golden Age Arc matured into the Black Swordsman Arc.

That didn't go anywhere either, until Lucid came out - Lucid, for the uninitiated, was a library to solve that holy grail of a sticking point: It allowed us to conveniently compose, build, and generally handle transactions in the frontend only. We were so back, and started hacking, calculating, restarting, hacking, calculating, restarting, and finally, in the Summer of 2023, building the frontend for Euclid: The latest iteration of MetaDEX. Which we actually managed to release on the Cardano Testnet towards the very end of 2023.

As it turned out, the thing was rather buggy. And we didn't bother to initiate the setup of the company in time, which was a precursor requirement for a token sale, which in turn had to precede the audit, which now seemed unavoidable. We did actually take a quick detour into the often-advertised Formal Verification Of Smart Contracts On Cardano Which Is So Much Easier Because We Are Smart And Use A Purely Functional Language Onchain - which wasn't quite there yet neither. So we had to go back to plan A, which is short for Paying External Auditors Like Peasants. Oh well; Retraction of the Light and its consequences have been a desaster for the human race, and we should just get with the programme, we thought.

Jinx! Alas, we did not. Ok, we did a little bit, by accepting the profane demand for legal setups queueing into fundraising queueing into third party audits. But as a sort of payback, we had a thought: Since Euclid already makes some light use of transaction chaining, and needs some centralized servers here and there - why not just lean into it? Around the same time we read Zygomeb's rather cryptic paper on Leviathan, and it stuck. What if, we figured, we do something like that (or rather, what our brain guessed was meant in said scroll), but not as an entire big capital L capital 2 Layer 2, but just for the v2 of the DEX? That would allow us, we continued to think, go back to all the things we tinkered out in the very beginning of 2021, on paper, and just code it down naively. Hell yeah, we proclaimed, let's just do that

[4]Or, if we're being honest, "said the author against the begging and imploring of his team"...

[5]Yet again we ask the reader not to overthink it.

while the legal setup completes! It should not take more than a couple of weeks.

A couple of weeks[6] later, around late summer of 2024, we were in the final stages of testing Keter (the title of MetaDEX: v2). We had also hired some people, some new, some from before the Eclipse.

"Hey", it was suggested by one of them, "Why don't we apply for Catalyst with that Layer thing you're building?". Grand idea, we thought, let's just spend a week or so factoring out the DEX-stuff, then come back to it.

Yet, upon beholding the quickly written Keter part of the code we felt rather reminded of Italian cuisine than a Prussian army on the march. After quickly pondering the option of expecting the users of Tiamat (the DEX without the DEX) to write their own DApp-specific spaghetti-code analogously to our own, we said Nay! And wrote the Ganglion-framework too.

Now, finally, we can present that work. Well, that is a bit of a misrepresenation - we promised we would, in the catalyst proposal accompanying Tiamat, and as of the time of writing, we are already a couple hours late. The reader will be able to accept that, we are certain of it. Yet it is almost finished at this point, and in any case - whoever can say "this should be take a couple of weeks at the most" must also be able to utter the grim, so very grim phrase of "binding deadline".

In that spirit, we sincerely hope the reader will enjoy our work, ye mighty, and not despair, but become bvllish af from the implied implications for the entire Cardano ecosystem.

[6] about half a year is still a couple of weeks, depending on the definition of "a couple" and "weeks" and "half" and "year" and so on.

3 Technical Background

3.1 eUTxO Model

Coming soon!

3.2 Account Model

Coming soon!

3.3 State Machines

Coming soon!

3.4 Transaction Chaining

Coming soon!

3.5 Mempool

Coming soon!

3.6 Chain API

Coming soon!

4 Related Work

In this section we present related works we drew inspiration from, may constitute valid yet different alternatives to our approach, as well as those more old-fashioned approaches we seek to replace.

4.1 Transaction Chaining Layers

4.1.1 Leviathan

The main inspiration for this work lies in Leviathan, which was presented by Optim's Zygomeb in January 2021[13] - in particular, the idea to harness Cardano's TRANSACTION CHAINING capabilities by deploying an extra layer to facilitate that.

In addition, the paper mentions the need for some sort of "account abstraction". Our assumption was that the motivating force behind this was to prevent DOUBLE SPENDING of UTXOS involved in TRANSACTION CHAINS, thus standing a chance of interrupting them.

By the strong demand to make any TRANSACTION CHAIN-breaking absolutely impossible, the third concept was introduced, which was termed "time travel" and which we understand to be referring to asynchronicities between the Leviathan instance and the Cardano mainnet.

The biggest stopping point of such an undertaking appears to be Cardano's inability to draw collateral-UTXOS from smart contract addresses, thus making it impossible to prevent involved parties from interruping said chains, and potentially obliterating entire alternative timelines in the process.

4.2 Isomorphic State Channels

4.2.1 Hydra

Coming soon!

4.2.2 Hydrozoa

Coming soon!

4.2.3 Gummiworm

Coming soon!

4.3 Optimistic Rollups

4.3.1 Midgard

Coming soon!

4.4 ZK-Rollups

4.4.1 zkfold

Coming soon!

4.5 Batchers

Most dApps - in particular, decentralized exchanges (DEXes) - on Cardano solve the "one action per 20 seconds" issue with **batching**. This approach works as follows:

1. The user creates an order-UTxO at the smart contract address.
2. Some program running on a server takes note. This is called the "Batcher".
3. The Batcher creates another transaction, spending all the order-UTxOs as well as the pool-UTxO in one go, and returning the funds to the users.

Different projects have different approaches towards ensuring those Batchers do not abuse the powers vested in them. Herein lies the first downside of that approach: This is a problem to be addressed in the first place, and to be insisted upon by the community. Yet more often than not a mantra of "piecewise decentralization" is invoked to justify simply reconstructing the very same centralized power structures we aimed to escape in the first place.

The second drawback lies in the fact that the atomicity of the intended user-action is broken up, which can - and does - result in the need to cancel & resubmit in case of too steep a price change (in the given example), suddenly excluding the order from being considered valid and therefore executable.

The third issue lies in the breakage of **determinism**: While in the raw UTxO-model the user signs off the spending of a specific set of UTxOs, therefore ensuring there's no reordering or similar trickery possible (compare Section 5.13 on MEV), this elegant attribute gets lost here.

4.5.1 Frontend Batchers

We cannot not mention our own proposed solution to the matter from 2021, which we published on youtube[7].

The core idea was to implement the same functionality as the aforementioned Batchers, but in a way that does not require servers. The approach was to run a sort of "inclusion-auction" phase, in which the goal would be to record all relevant order-UTxOs, to prevent censorship (which, in the context of swaps, is heavily incentivised, as more competing buyers of the same asset will increase the slippage for everyone aiming to trade in that direction, at that point in time). In this phase, anyone could add any left-out orders submitted during the initial block, and the previous batch-composers would be punished accordingly. The auction then finishes after a certain "speak now or hold your peace forever" period. After that, any of the parties involved would be able to execute the swap, which would be computable deterministally and secured by the swap-contract.

While that sort of system would of course be running in a staggered, parallel, manner - we would not have to wait for the first cycle of order submission, batching auction, and swap execution to complete before starting the next one - it is rather impractical for a number of reasons:

- The users would need to sign off multiple transactions, which are only presented in 20 second intervals.
- The entire operation would take over a minute at the minimum.
- Some finer details regarding fair aggregation and edge case attacks (i.e. what we internally termed "probabilistic frontrunning") would need to be addressed, which would become even more complex with expansion of the feature-set of the DAPP.

In that spirit we consider it rather a pleasant thought that nevertheless may become useful again at a later date, in some evolved form.

5 Motivation

In this section we outline the different needs we perceive and attempt to address.

5.1 Enable new Developers to Build on Cardano as Intended, the Naive Way

Each UTxO can be spent only once. That means, without TRANSACTION CHAINING, one would have to wait for the BLOCK to be concluded and accepted by the Cardano Mainnet - which takes around 20 seconds. That means: In any naive implementation, the end user can only perform one subsequent action every 20 seconds[7].

Of course, the story didn't end there. Since Cardano people are smart, different workarounds were proposed, implemented, and have been running for years with great success. Yet they come with their own set of drawbacks:

- They have to be invented in the first place, which is not an easy thing to do, and excludes those who simply want to implement a very simple DAPP to wet their feet, but also have it working in any sort of reasonable manner.
- If they *do* manage to invent it, this will be time they could have spent implementing and improving their core business logic.
- Since due to financial incentives there is a strong hesitancy to fully open source, each team has to reinvent the wheel anew, even if the core principles are known.
- Reinventing the wheel not only wastes time, but has a higher risk of introducing bugs and security vulnerabilities. This gets compounded by the aforementioned hesistancy to open source - this bad NASH EQUILIBRIUM we as a community suffer from - and we at MetaDEX try to break, the Alexandrian way[8] - prevents third parties from looking for bugs - and worse, exploits - themselves.
- The end result often enough throws away many of the advantages we gained through the EUTXO model in the first place (compare 4.5).

[7] The reader may remember the "one swap per 20 seconds" FUD based on that. They *sort of* had a point...

[8] `https://en.wikipedia.org/wiki/Gordian_Knot`

This is quite the shame, because via TRANSACTION CHAINING, we theoretically already have a tool at our disposal to proceed as fast as the wire will bear[9] - however. In order to use that efficiently to full effect, and including multiple parties, a number of smaller challenges must be addressed. The simplest solution would be simply to use a centralized server to do so. In this paper, we outline our solution to do it the decentralized way.

The same applies, albeit to a lesser extent, to

- STATE MACHINES [3.3]
- potentially complex data processing [7]
- PROPERTY TESTING [5.10]
- onchain data serialization, deserialization, and verification [6.10]

We also provide solutions to the above.

5.2 Instantly Interoperable, Trustless Layer 2 Solutions

Layer 2 solutions, such as isomorphic state channels [4.2] and rollups [4.3, 4.4], have emerged as critical tools for scaling blockchain systems while retaining the underlying benefits of decentralization and security. By shifting computations and state transitions off-chain, these solutions offer several advantages:

- **Scalability:** Significantly increased throughput compared to Layer 1 blockchains by reducing on-chain computational and storage requirements.
- **Cost Efficiency:** Lower transaction fees by minimizing the amount of data that needs to be recorded on-chain.
- **Speed:** Faster transaction finality as most interactions occur off-chain.
- **Interoperability:** Enhanced ability to interact with other systems or applications through modular designs.

[9] trigger warning

Despite these benefits, Layer 2 solutions are not without their drawbacks. The primary challenges stem from the need to ensure trustless operations between participants and to reconcile off-chain state with on-chain security guarantees. Below, we detail these drawbacks:

5.2.1 Trust and Dispute Periods

One of the foundational principles of blockchain technology is trustlessness, where participants can operate without relying on centralized intermediaries or mutual trust. However, Layer 2 solutions often introduce temporary trust dependencies:

- **State Disputes:** In state channels and rollups, participants must trust that counterparties will honestly report off-chain interactions. If a dispute arises, it requires on-chain resolution mechanisms to verify and enforce correctness.
- **Dispute Periods:** To allow for dispute resolution, Layer 2 systems typically enforce a "challenge period" during which any participant can submit evidence of dishonest behavior. While necessary, these periods introduce delays in finalizing transactions and increase complexity.
- **Data Availability Risks:** Rollups depend on the availability of off-chain data for reconstructing the state. If this data becomes inaccessible or corrupted, it can lead to system failure or disputes that are difficult to resolve.

5.2.2 Operational Complexity and User Experience

The mechanisms required to ensure trustlessness in Layer 2 solutions often result in additional operational overhead:

- **Verification Overhead:** On-chain verification of off-chain state transitions can be computationally expensive, negating some scalability benefits.
- **User Burden:** Participants must monitor the blockchain during dispute periods to protect their interests, which can be impractical for everyday users.

- **Integration Challenges:** Achieving seamless interoperability between Layer 2 solutions and the base Layer 1 blockchain can require significant development effort, particularly when ensuring consistent serialization and deserialization of state data.

These drawbacks highlight the need for innovation in Layer 2 systems to achieve greater trustlessness and instant interoperability without sacrificing the benefits of scalability, speed, and cost efficiency. Any comprehensive solution must address these pain points while maintaining robust security and usability for participants across the ecosystem.

5.3 Unlocking the Full Potential of Multi-Party Transaction Chaining

As mentioned in 5.1, TRANSACTION CHAINING is a powerful tool that is already part of the Cardano protocol. However, in order to use it to its full potential, one would need to ensure that

- the TRANSACTION CHAINS are not interrupted by successful DOUBLE SPENDING of any of the UTXOS involved, which would cancel all subsequent transactions
- the intermediary transactions are valid in the first place
- other users are made aware of the pending UTXOS such that they may chain their own transactions after them
- there is consensus about the former (compare 5.5)
- all the above are served in a strictly decentralized manner. Not "decentralized", but decentralized.

5.4 Prevent Breaking of Transaction Chains

TRANSACTION CHAINS can be broken if a single transaction in the chain fails. This can be due to

- the transaction being malformed
- the block producer censoring it maliciously
- the block producer not including it because they didn't receive it in time
- the block producer not including it because the block is already full (congestion)
- Double spending of any input-UTxO of the transaction

Within our framework, we aim to address as many of those as well as we can.

5.5 Mempool Availablity and Consensus

Using transaction chaining to its full effect - meaning: involving multiple users - requires not only awareness of the relevant section of the mempool at all times, but a consensus over it. If the users don't know - or disagree about - the actual pending UTxOs available in the mempool which are to be consumed by their subsequent transactions, they will be evidently be unable to do so (in the former case) or will have to expect a certain rate of transaction failure (in the latter case).

We are especially highlighting here the need for the **consensus** over the mempool state, as the resulting drawback of a lack thereof compounds upon itself with growing length of average transaction chains. Not only will each failure cancel a higher number of subsequent transactions, but each disagreement will introduce a "fork" in those transaction chains; the resulting tree-structure will result in failure of all transactions that are not part of the singular winning path.

5.6 Avoid Excessive Password Prompts

By design, any wallet browser extension worth its salt will prevent dApps from spending the user's funds without their expressed permission. This in practice takes the shape of password-prompts.

However, this intrudes a few difficulties:

- The transaction will be delayed, at which point the underlying UTxOs may have already been spent otherwise, thus invalidating it.
- Some dApp architectures may require multiple transactions, which in turn would result in the subpar user experience of having to process multiple password-prompts for a single action. This of course compounds with the previous point.
- We cannot have the offchain application automatically retry, nor automate it in any other manner. This means we are reliant on parties beyond our local environment to execute more complex strategies for us, which in turn implies one of the following:
  - delegation to some centralized server, which, besides the obvious and known drawbacks, can use that information imbalance to great detriment to the community[10]
  - making our intention publicly known, and requiring smart contract fees to be paid to process them[11].
  - requirement of strange and otherwordly "moon math" based around zk-snarks and the like, which at the minimum are not yet fully available.

  So, in summary, the simplest way for users to automate their strategies may simply require them to keep a browser window open.

5.7 Avoid Ongoing Operational Costs during Low Activity

Not every dApp can expect a high load, all the time. This means there is a certain risk involved in desiging architectures in a way that has ongoing operational costs.

[10]No names will be named here. But the informed reader may already know who the parties discussed may be.

[11]Which is in our opinion the biggest criticism that can be levvied against AXO [1] - if a user is sophisticated enough to compose complex trading strategies, why would they prefer to not only put their strategy on a public blockchain for everyone to see, and counter, but also pay higher smart contract execution fees - instead of simply setting up their own Hummingbot [8] instance on a cheap cloud service?

5.8 State Machines

State Machines are an incredibly useful primitive in computer science. We have already seen many implementations of them on Cardano, based on the eUTxO model. However, in order to prevent developers from having to reinvent the wheel and to avoid the accompanying negative consequences (compare 5.1), we benefit from a framework for this.

5.9 Fast complex data processing

While building MetaDEX v2: Keter, we found ourselves designing rather complex data processing pipelines. That is a need other may share as well. This endeavor will be greatly enhanced by providing a framework for that very purpose, both in accuracy, speed, and developer experience.

5.10 Property Based Testing and Data Serialization/Deserialization/Verification

Property-based testing is a software testing methodology that focuses on verifying that a program's behavior satisfies a set of high-level properties or invariants, rather than relying on individual examples or specific test cases. Instead of hardcoding particular input-output pairs, the developer specifies general properties that must hold true for a wide range of inputs. A testing framework then generates a variety of input data, often randomly or systematically, to validate these properties across a broad spectrum of scenarios.

For example, in testing a sorting algorithm, a property might state that for any input list, the output should:

- Be sorted in ascending order.
- Contain the same elements as the input.

The testing framework would then automatically generate numerous input lists (e.g.,

empty lists, lists with duplicates, reverse-sorted lists, etc.) to ensure that the sorting function satisfies these properties.

5.10.1 Supporting End-to-End Property Testing with Custom Types

Current property-based testing frameworks excel at generating standard types (e.g., integers, strings, lists) but often lack robust support for complex, user-defined types, particularly in systems that span multiple execution environments such as on-chain and off-chain code. This limitation becomes critical when testing dApps, where data structures and logic must remain consistent across both domains.

An ideal framework must address these needs by:

- Allowing developers to define custom types and their associated constraints.
- Generating valid subtypes and associated data automatically based on these definitions.
- Ensuring seamless testing of both on-chain and off-chain components, providing end-to-end coverage.
- Unifying the serialization, deserialization, and verification processes for on-chain and off-chain data.

5.10.2 Serialization, Deserialization, and Verification

Serialization and deserialization are critical processes in decentralized systems, as they bridge the communication gap between on-chain smart contracts and off-chain applications. Ensuring that these processes are consistent and error-free is essential to maintaining system integrity. A unified framework can leverage property-based testing to:

- Verify that serialized data adheres to the expected format and can be correctly deserialized back into the original custom types.
- Ensure that user-defined types maintain their invariants across serialization and deserialization boundaries.

- Test the entire end-to-end workflow, from generating valid data for custom types to verifying its correctness after on-chain and off-chain processing.

By combining property-based testing with serialization verification, the framework ensures not only that the application's logic is robust but also that its data encoding and decoding mechanisms are reliable. This comprehensive approach significantly reduces the risk of subtle bugs and inconsistencies, enabling developers to build more dependable decentralized systems.

5.11 Decentralized Chain API

While decentralized systems promise resilience and trustlessness, it is often overlooked that most interactions with blockchain networks rely heavily on centralized servers. In the Cardano ecosystem, Blockfrost—a service similar to Infura for Ethereum—serves as a crucial intermediary for accessing the blockchain. Blockfrost enables developers to query the blockchain and submit transactions without the need to run a full Cardano node locally. This convenience is indispensable in many scenarios, particularly for lightweight applications such as those running in a web browser or mobile environments, where the computational and storage requirements of running a full node are prohibitive.

A chain API serves two primary purposes:

- **Querying Data:** Retrieving information from the blockchain, such as transaction details, account balances, or smart contract state.
- **Submitting Transactions:** Broadcasting signed transactions to the blockchain network for validation and inclusion in a block.

These functionalities are essential for dApps to operate effectively, enabling interactions between users and the blockchain.

However, reliance on centralized services like Blockfrost introduces several significant concerns:

- **Centralization Risk:** By depending on a single entity to provide access to the blockchain, developers and users are exposed to potential downtime, censorship, or service interruptions.
- **Trust Dependency:** Centralized APIs act as gatekeepers, undermining the trustless nature of blockchain systems. Users must trust that these services accurately represent the blockchain state and reliably broadcast transactions.
- **Potential Bottlenecks:** Centralized services can become performance bottlenecks under high network demand, limiting scalability.

To address these issues, there is a pressing need for a decentralized alternative to chain APIs. Such a solution would:

- Distribute the responsibilities of querying and submitting data across multiple nodes, ensuring redundancy and resilience.
- Remove single points of failure, making the network more robust against censorship and outages.
- Preserve the trustless principles of blockchain systems by allowing users to interact with the chain directly, without intermediaries.

By decentralizing this critical layer, we can achieve the full promise of blockchain technology: a system where users and developers can interact with the chain securely and reliably, without depending on centralized intermediaries.

5.12 Ensuring High Availability

High availability is a critical requirement in decentralized systems, where functionality depends on a distributed network of independently operated nodes. Unlike centralized systems, where a single entity ensures uptime, decentralized systems rely on the collective efforts of node operators. This creates inherent challenges, as individual nodes may become unavailable due to hardware failures, connectivity issues, or operators simply leaving the network.

When a node becomes unavailable, its absence can disrupt essential services. For users and applications that rely on constant access, these disruptions can lead to delays, reduced reliability, and even loss of trust in the system. The decentralized nature of blockchain ecosystems makes it difficult to predict or prevent node downtime, further complicating the problem.

In high-load environments, where dApps and off-chain components interact heavily with the blockchain, high availability is essential to ensure a seamless user experience. Without mechanisms to address node availability, decentralized systems risk losing one of their core promises: resilience through decentralization. Ensuring that services remain operational despite individual node failures is therefore fundamental to maintaining the integrity and usability of the network. This need becomes even more pronounced as adoption grows and applications demand higher reliability to support critical use cases.

5.13 Mitigate MEV

> Maximal extractable value (MEV) refers to the maximum value that can be extracted from block production in excess of the standard block reward and gas fees by including, excluding, and changing the order of transactions in a block. [...] Maximal extractable value was first applied in the context of proof-of-work, and initially referred to as "miner extractable value". This is because in proof-of-work, miners control transaction inclusion, exclusion, and ordering. ([4])

Ignoring the semantics regarding the naming of the block producers, one of the big advantages of the UTxO model is that it inherently prevents that. As each user signs a transaction that exactly specifies which UTxOs it spends, reordering is not possible. While transaction censorship is still possible, that is not nearly as impactful. Yet, transaction reordering would enable the dreaded sandwich attack:

> Also known as front-running, a sandwich attack is a type of maximal extractable value (MEV) tactic, where the attacker spots a pending transaction in the network and deliberately 'sandwiches' the transaction by placing orders right before and after the targeted transaction. It is also

> good to note that while front-running is one of the more prominent forms of MEV, there are also other forms of MEV such as back-running, transaction ordering manipulation, and more. ([12])

Now, with the various workarounds the community had to come up with for the "one action every 20 seconds" problem, this possibility gets reintroduced into our ecosystem. In turn, that necessitated solutions - which in turn suffers from the same issues outlined in 5.1.

This implies: For our own solution, we must take great care to avoid this.

5.14 Value Capture does not Incentivise Contribution

The spirit of FOSS - free, open-source software - is that it belongs to everyone, so everyone is invited to participate, and motivated to contribute. We speak of the "open source community" not without reason; yet: We in the "Cardano community" fail to translate said spirit of belonging to the heart, to where it matters, to the technology.

No one wants to waste their preciously limited time building other people's bags. Each and every developer has their own passion projects they painfully ignore for their day jobs. The goal, then, must be to make our project their passion project, by yielding control, truly yielding it, and giving it to them, so thay may make it their own.

We have been advised an uncountable number of times to not open source too early, because there are predators around who will simply fork and rebrand us. To those I pose the question: How did that go?[12]

We place our trust in the Cardano community to continue sanctioning such vile behaviour. We further believe that the inherent effect of the best of people coordinating instintively, super-rationally, will tremendously outshine any attempt at value capture, in the long run.

Finally, we believe in the principle of giving people enough rope to hang themselves with. In that sense we consider our approach a litmus test for the Cardano commu-

[12]Or, in other words - Why is project not doing so well?

nity - will they hold true to their values, or be yet another dead ship at sea, overrun, gutted, and slain by the hordes of mindless moonfolk?

5.15 Summary

As we have listed a plethora of demands, we want to highlight the core needs that Tiamat aims to address - the ones related to the high-load, multi-party TRANSACTION CHAINING core protocol:

- **5.1 Enable new Developers to Build on Cardano as Intended, the Naive Way**: The framework remains easy to use and understand, especially for new developers.
- **5.4 Prevent Breaking of Transaction Chains**: We need to ensure transactions are well-formed and can not cause harm by attempted DOUBLE SPENDING.
- **5.5 Mempool Availablity and Consensus**: We require availability of and consensus over the relevant part of the mempool at all times.
- **5.7 Avoid Ongoing Operational Costs during Low Activity**: As the particular DAPP instances building on Tiamat may not have an ongoing demand, and therefore income, we cannot design the framework in a way that incurs ongoing operational costs irrespective of usage.
- **5.12 Ensuring High Availability**: The system is available at all times, even if individual nodes are not as reliable.
- **5.13 Mitigate MEV**: There will be absolutely no funny business by the node operators of any sort. This becomes doubly important when considering the fact that each DAPP building on Tiamat cannot be expected to provide sufficient trust guarantees, as history has shown that users simply will not care, and use those products anyways.

6 Approach

In this section we describe our solutions to the aforementioned.

6.1 Architecture

The essence of Tiamat is a collection of nodes that run alongside Cardano nodes which are respected as data sources by the Cardano network - in practice, that means: Stake pools. Those nodes are called Vectors, and the set of them then naturally is termed the Matrix.

Those Vectors run a proof-of-stake-system not unlike Cardano itself - however, there are some differences:

- To avoid confusion, we name our "slot leaders" Eigenvectors, the "stake" of each Vector its Eigenvalue and our "blocks" Cycles.
- Stake can only be withdrawn into a vesting contract to better align incentives (compare 11.2)
- The rewards are paid in ADA, yet the staked token is defined by the dApp in question and strongly recommended to be a dedicated one, for the same reasons as outlined in 11.2. This dedicated staking-token we term Eigenwert. Note that this is a variable name, not a string - each Tiamat instance will have to fill in its own asset here.
- There are more than one Eigenvectors per Cycle to increase availablility and disincentivise misbehaviour of Eigenvectors.
- Instead of being a rather rare occurence "slot battles" are the norm - in particular, they are expected to happen for each and every transaction submitted via Tiamat. This is part of the scheme to make collusion between dishonest Eigenvectors very hard - and as a nice side effect, it incentivises aiming for the best of speeds that can be achieved.

The duties of those Eigenvectors are as follows:

- Provide decentralized chain API services, meaning: state queries and transaction submission.
- Maintain a consensus over the current state of the MEMPOOL as far as it pertains to the contract-addresses in question, and push updates thereof instantly to any subscribed USERS via websocket.
- Co-sign smart contract interactions that are meant to leverage Tiamat's TRANSACTION CHAINING capabilities to the fullest.

That last point requires some elaboration: Since TRANSACTION CHAINS can be interrupted by DOUBLE SPENDING of input-UTXOs, we need to prevent that from happening. We address the handling of non-contract-input-UTXOs in 6.3; yet for contract-UTXOs (at least those that are subject to the Tiamat-instance) we ensure this by requiring enough co-signatures of any such a transaction buy a sufficient number of EIGENVECTORS. Those we term the SUPPORT-VECTORS of said transaction. In that way we leverage the onchain smart contract code to enforce the very same entities tasked with maintaining the MEMPOOL consensus are also required to greenlight any changes to the UTXO-sets in question.

6.2 Redundant and Competing Slot Leaders

By having a redundant set of EIGENVECTORS each CYCLE, we increase the expected availability of the network in relation to the expected availability of a single VECTOR.

6.2.1 Probability Calculation

The probability of achieving QUORUM - the minimum number of required SUPPORT-VECTORS - depends on:

- **Total Number of Eigenvectors** (n)
- **Quorum Size** (k): Minimum number of eigenvectors required.
- **Availability Probability** (p): Probability that an eigenvector is online and responsive.

The probability of reaching QUORUM is given by:

$$P(Q) = \sum_{i=k}^{n} \binom{n}{i} p^i (1-p)^{n-i}$$

6.2.2 Sample Data

Table 1: Quorum Availability $P(Q)$ with Varying Parameters (rounded)

# **Eigenvectors** (n)	**Quorum** (k)	$p =$ **85%**	$p =$ **90%**	$p =$ **95%**
1	1	85.0 %	90.0 %	95.0 %
3	2	93.9 %	97.2 %	99.3 %
5	3	97.3 %	99.1 %	99.9 %
7	4	98.8 %	99.7 %	100 %
11	6	99.7 %	100 %	100 %

6.2.3 No Time For Collusion

Further, and perhaps more importantly, by having each CYCLE's SUPPORT-VECTOR-SETS compete for each transaction, we summon time pressure - that in turn not only serves to speed things up, but more importantly makes collusion very tricky. The reader is referred to Section 8.1 for a deeper elaboration of the process.

And of course - "Collusion" also encapsulates the trivial case of a singular slot leader "colluding" with itself.

6.3 Jailing Chain-breakers

Besides ensuring the contract-UTXOS cannot suffer from DOUBLE SPENDING as outlined in [6.1], we need to ensure the users cannot do so with their own UTXOS.

Our solution is very simple: We identify such behaviour, and ban transgressors from using the system, commonly known as "jailing". This can be extended over various timeframes. Further, in scenarios of heavy interference, freshly created wallets can be prohibited from participation alltogether. Yet it must also be noted that such chain-breaking transactions still incur fees, so an extended attack should be made infeasible by the exact same mechanism the Cardano Blockchain defends itself against any other attack of this sort.

We further note that our system is designed for a normal mode of operation of the Cardano Main Chain - in scenarios of congestion, we don't expect Tiamat-based dApps to run seamlessly neither. That means we simply exclude the edge case of "what if, due to mainnet congestion, the Tiamat-transaction chains become extremely long, and therefore would take not only really long to settle on the base ledger, but also become increasingly vulnerable to said interruptions?" per definition. The intended use will simply cut off any transaction chains that are not included in the last block of each Cycle, as the next set of Eigenvectors take over, and pick off only from what they can find on the main chain.

We note that this is the biggest drawback of systems like Tiamat and Leviathan when compared to more "heavy" Layer 2 solutions based on isomorphic state channels and rollups, and the reason we cannot stop stressing how the approaches are meant to complement each other instead of competing [12].

6.4 Users Provide Collateral-UTxOs

In Cardano, each transaction that spends UTxOs from smart contract addresses need to be accompanied by Collateral-UTxOs. The purpose of those is to reimburse the system for effort wasted with attempting to execute potentially very costly smart contract code, only for it to fail - in which case the fees would not be collected.

Those Collateral-UTxOs must come from a simple user address. Now, this opens the issue of double spending breaking the transaction chain - even if we managed to secure all other input-UTxOs via smart contract first, this aspect would still allow for said interruptions.

Our solution is simple: We have the user provide those as well, and defer to the solution for double spending of any other user-input-UTxOs - Jailing [6.3]. This

also carries the advantage of simplifying the cost model - we do not have to devise complex schemes where the EIGENVECTORS have to front the costs, and various edge cases need to be identified and addressed. If some malicious user wants to play games, they bear the cost themselves in a very straightforward manner.

6.5 Atomic Actions

A fundamental design principle in UTXO-based systems is ensuring that users only sign transactions that fully execute their intended actions in a single, atomic step. This approach minimizes risks and preserves user control by avoiding scenarios where funds or assets are sent to intermediaries or third parties for processing unless absolutely necessary.

Atomic actions ensure that the entirety of a user's intent is executed within the same transaction. When interacting with a DAPP, a transaction should either complete the desired action (e.g., a token swap or other form of contract interaction) in full or fail entirely, without any partial execution. This principle eliminates ambiguity, reduces the attack surface for malicious actors, and provides a clear and predictable user experience.

This design principle is not always adhered to various solutions to the "one interaction per 20 seconds" challenge. For example, batcher-based systems [4.5] like all current AMM-DEXes require the user to send their funds to a smart contract address, to form an order-UTXO, which then is processed by said batcher via another, subsequent transaction. While misbehaviour of those entities can be mitigated via smart contract and/or other - more lax, a.k.a. game-theory-based - schemes, it still is not atomic. In the case of swaps, for example, it opens the possibility for sandwich-attacks [5.13] or simply bad luck - when the price changes for whatever reason past the user-defined slippage tolerance recorded in the order-UTXO, they have to make the uncomfortable choice whether to wait - and hope - for the price to return, to to pay for two more transactions; one to cancel, and one to re-submit - at an worse price point, of course.

By allowing DAPP developers to implement their business logic naively, the Cardano/UTXO-native way, the atomic way, we exclude those inconveniences as well as further needs for reliance of the sound implementation of those systems.

6.6 Servitor-Wallets and Account-SVMs

In addition to the regular Owner-Wallet, which is administrated by the browser-extension of choice, we create another wallet that lives in the browser, with the key stored in its local storage.

This is of course less than secure. Were the user to wipe their local storage, visit malicious websites without taking the appropriate precautions, or even install malicious browser extensions, the funds could be lost.

On the flipside, it allows us to go real fast wooosh - not having for the slow Human to input their password each time, possibly multipe times, allows us to greatly enhance the user experience.

We solve this conundrum by only ever sending a very small amount of ADA to said Servitor-Wallet - just enough to cover the fees for a couple of transactions.

It must also be noted that for other use cases, for example token swaps, we may want to have that instant-ness combined with assets of non-trivial value, after all. For this purpose we use Account-SVMs - special state machines that can only be spent either back into the Owner-Wallet, or by executing the order defined therein.[13]

6.7 Lazy Elections

Our Elections are **lazy**, which means: We do not require the current electorate to be recorded onchain until an action is performed. This allows us to save ongoing tx fees. Further: The first user of each Cycle pays the transaction fees for that Cycle's Election.

The Eligible Eigenvectors of each Cycle depend on the previous Election's hash and the time interval of the Cycle, so it still changes with each new Cycle. This means their set is predictable; but this predictability gets broken as soon as the onchain Election gets updated, which changes the hash input.

[13]No, this is not the same as the "order-UTxOs we talked smack about just earlier - but the reader will have to wait patiently for the release of MetaDEX v2: Keter to learn more about why that is so.

As the next Cycle's Eligible Eigenvectors are deterministic, we can connect in advance to ensure smooth operation.

6.8 Support Vector Machines

We implemented a generic framework for State Machines based on the eUTxO-model that also ties directly into the transaction chaining-layer- and Euclidean property-testing/serialization/deserialization/verification frameworks [6.1, 6.10, 10].

By adding the requirement for co-signatures from a sufficient number of Support-Vectors to state transitions we arrive at the logical term "Support Vector Machines"[14] - or short: SVMs.

6.9 Ganglion Data Framework

We implement the Ganglion Data Framework for complex, push-based, and efficient data pipelines meant to run in the frontend. Refer to section 7 for further elaboration.

6.10 Euclidean Type System and Proptesting Framework

We are reusing the extensive framework for integrated property-based testing, chain data serialization, deserialization, and verification that was developed for MetaDEX v1: Euclid[6]. Section 10 goes into more detail.

6.11 Open Source DApp Development Framework

While we have plans for creating our own Tiamat-based full "Layer 2" [13.4], that is further on the horizon. The core intent for Tiamat is to be a dApp-specific transaction chaining-framework. It is meant to be fully open sourced, and usable by anyone.

[14] Any resemblance of the machine learning algorithm of the same name is purely coincidental.

This principle needs to be highlighted, as the concept often seems somewhat counter-intuitive to more business-minded people. "How in the hells", they ask, "did you computer science freaks ever arrive at the thought that it may be an even remotely good idea to give away anything for free? Aren't we here to make money by selling things? You aren't a communist, are you?[15]"

Well.

Free and Open Source Software (FOSS) has become a cornerstone of modern software development due to its unique advantages in fostering innovation, collaboration, and transparency. At its core, FOSS refers to software that grants users the freedom to view, modify, and distribute its source code. When coupled with copyleft licenses, which require derivative works to remain free and open, these principles create a robust ecosystem for sustainable software development.

One of the primary advantages of FOSS is its **transparency**. With publicly available source code, developers and users can verify the functionality and security of the software, ensuring there are no hidden vulnerabilities or backdoors. This transparency is particularly important in decentralized systems, where trustlessness is a foundational principle.

FOSS also encourages **collaboration** by enabling a global community of developers to contribute improvements, fix bugs, and extend functionality. This collective effort often results in software that is more reliable and feature-rich than proprietary alternatives, as it benefits from the diverse expertise and perspectives of its contributors.

Another key benefit is **adaptability**. Users are not constrained by vendor lock-in or proprietary restrictions; they can modify FOSS to meet specific needs, making it highly versatile for a wide range of use cases. This is especially valuable in fast-evolving fields like blockchain and decentralized applications, where innovation often outpaces the capabilities of closed-source solutions.

When distributed under copyleft licenses, FOSS ensures that these freedoms are preserved. Copyleft guarantees that any modifications or derivative works are also released under the same open terms, preventing companies or individuals from privatizing the software's benefits. This creates a virtuous cycle, where improvements

[15] `https://www.reddit.com/r/DebateaCommunist/comments/tz34s/why_do_people_here_keep_saying_that_foss_software/`

made by one party can benefit the entire community.

In conclusion: If it true what they say about Cardano, and what we like to say about ourselves - that we are the Linux of Blockchain, the good ones, and the nerds - this is the way to go, this is the way to attract more developers to contribute to a shared interest.

And if it is not, it would be preferable to learn that fact rather sooner than later.

7 Ganglion Data Framework

In this section we describe our Ganglion data processing framework.

7.1 Frontend data pipeline DAG

The Ganglion framework aims to address the need for

- complex
- efficient
- fast
- data processing
- in the frontend.

That means we need to:

- be able to express potentially complex pipelines in a clean manner.
- only execute the computations that are affected by upstream data updates.
- deploy a pub-sub/push-based architecture.

The data pipeline represents a DAG: a directed acyclic graph. Each node - called a Ganglion - is potentially connected to multiple upstream and downstream Ganglia. When it receives a data update,

1. re-runs it's own associated computations with the latest data from upstream,
2. compares the result with its previous state, and
3. if there was a change, pushes said update downstream.

7.2 WebSocket Data Feeds

The root of those data updates are constituted by the current Cycle's Eigenvectors, to which the User automatically maintains resp. updates websocket connections, and forwards any new contract addresses they are subscribing to.

7.3 Intents and Precons

Each possible Action a User can take when interacting with a dApp is wrapped in an Intent - the duty of which is to

- watch the end result of the relevant, processed, onchain data updates,
- ingest the Human inputs,
- process said Human inputs together with the aforementioned latest data updates, update the display, and give feedback regarding the validity of the choices and currently available options,
- allow the Human to execute the associated action by
  - checking the Precons of the action. Those are in turn bundles of individual preconditions that need to be met for the actual payload Action-Tx to succeed, as well as instructions on how to fix those that are not. Those fixes are then being applied in the form of a singular Fixing-Tx to precede the Action-Tx.
  - potentially composing the Fixing-Tx for all unmet Precons
  - composing the Action-Tx
  - composing the Tipping-Tx
  - signing, sending, and submitting all the above
- monitor the chain for any relevant updates regarding those transactions.
- automatically re-try in case of failure of any of the transactions above
- allow the user to cancel said re-trying attempt

The process is also outlined in Section 8.1 with more detail.

7.4 Tiamat Core Precons

Tiamat Core comes with two PRECONS:

- **The Servitor-Precon** ensures that the SERVITOR-WALLET is funded sufficiently, as we do not want to involve the OWNER-WALLET in more transactions than necessary.
- **The Election-Precon** ensures the currently ELIGIBLE EIGENVECTORS are reflected onchain in the NEXUS-UTXO.

8 Protocol Flow

In this section we outline the process of the protocol and the interactions with it.

8.1 Revolving Actions

1. Each Cycle a number of Eigenvectors become Eligible. This can be anticipated by Users and Vectors, as discussed in "lazy elections"
2. Users and Eligible Eigenvectors connect to each other via websockets. Note that in order to get the initial set of Eigenvectors, the User has to rely on blockfrost or some other data source. But from then on, the Eigenvector system can provide the information about updates to the onchain election, which, recall, would break predictability of future elections.
3. The User determines which addresses to subscribe to and notifies the current Eigenvectors via websockets. This also gets repeated when connecting to new Eigenvectors.
4. The Intent/Precon-system informs the Human ASAP about the processed chain updates, as far as it pertains to the action in question, handles Human input, gives feedback, and, if valid, allows them to instruct the execution of the action.
5. The Precons of the Intent in question get checked, and if they are not filled, compose a Fixing-Tx that addresses all of them in a single transaction. This is not submitted yet. Recall that in Tiamat Core, those Precons will be the Servitor-Precon (which ensures servitor is funded) and the Election-Precon (which ensures the Eligible Eigenvectors are recorded onchain).
6. All minimal subsets of the current Eigenvectors (Eligible or Elected, depending) that meet the minimum Quorum are computed by the User. Those are the possible Support-Vector-Sets that could co-sign the smart contract transaction.
7. The action-payload-transaction (Action-Tx) is created and chained after the still unsubmitted Fixing-Tx, or created anew in case the former is not required. This Action-Tx also creates a new utxo at the servitor-address that

constitutes the Tiamat protocol fees for the VECTORS, called the "TIPS". This is also not submitted yet. Note that this ACTION-TX is not supposed to spend owner-funds, only servitor-funds, to avoid the need to wait for the HUMAN to enter their password into the browser wallet extension.

8. For each SUPPORT-VECTOR-SET, a TIPPING-TX is constructed, which is chained after the ACTION-TX and spends the TIPPING-UTXO to the VECTORS of that SUPPORT-VECTOR-SET. Note that those conflict with each other, as they attempt to double-spend the TIPPING-UTXO. This is intended.

9. The INTENT signs and submits the FIXING-TX to the current EIGENVECTORS, if applicable. This may require entering a password if it involves the OWNER-WALLET, which might create a delay long enough to invalidate the subsequent ACTION-TX. In which case we can recompute a new one without requiring user input, as, recall, the ACTION-TX does not involve the OWNER-WALLET. "Submitting" means: sending it to each EIGENVECTOR via websocket.

10. The EIGENVECTORS receive the FIXING-TX and forward it to their local nodes. They only need to execute the co-signing process (eleaborated upon below) it if it includes the fix of the ELECTION-PRECON, which will update the onchain record to list them explicitly. Otherwise, it will not have to involve any smart contract that is bound to the Tiamat-consensus, so a simple forwarding suffices.

11. The INTENT signs and submits the bundle of ACTION-TX and TIPPING-TXS (in a single websocket message) to each of the EIGENVECTORS. Here, each EIGENVECTOR receives the TIPPING-TXS for the SUPPORT-VECTOR-SETS of which they are part of.

12. The EIGENVECTORS receive the bundle of ACTION-TX and TIPPING-TXS pertaining to them.

13. The EIGENVECTORS check the validity of those transactions, and whether the tipping is in order. This is a shelling point determined by a Tiamat protocol parameter. Due to the fact that the offchain code automatically adheres to that, any divergent behaviour would go against that consensus, and be expected to fail. I.e. requiring higher TIPS from the EIGENVECTOR's side will simply give the "business" to the SUPPORT-VECTOR-SETS that don't include it, and trying to pay a lower TIP on the USER side will simply have their transaction rejected by the majority of EIGENVECTORS.

14. If the Eigenvectors are satisfied with the Action-Tx and Tipping-Txs, they co-sign them, and forward the updated bundles to the other members of the relevant support vector sets whose signatures are still missing.

15. When an Eigenvector ends up adding the final signatures to one of those bundles, they submit both the Action-Tx and Tipping-Tx to chain. This has the following implications:

    - The Eigenvectors can only pay themselves if the Action-Tx passes, as that is the one that creates the Tipping-UTxO, which is then chained into the Tipping-Tx, which pays them.
    - Since the Tipping-Txs attempt to double-spend the Tipping-UTxO, only one Tipping-Tx can pass. This means that there is a race between all Support-Vector-Sets, which makes collusions of minorities very hard. Collusions of majorities are addressed via the stake vesting and dedicated token, see (section).
    - Since the Tipping-Tx is different from the Action-Tx, which is the same for all Support-Vector-Sets, we don't break the consensus regarding the contract-utxos which we require for transaction chaining, as only the Action-Tx will affect those.

16. The Intent waits for any update from the chain.

17. If the Action-Tx fails, the Intent automatically computes a new one, as well as the new corresponding Tipping-Txs, and tries to submit those again. Note that this "failure" doesn't incur any cost to the User, as no Action-Tx was actually processed, and said "retrying" doesn't require their input, as we only use the Servitor-Wallet in Action-Txs and Tipping-Txs. Of course the Human has the option to interrupt this process at any time. The only exceptions to this are

    - If the Action-Tx fails because the Fixing-Tx fails (which is chained in front of it) - in which case the Intent also will automatically retry the entire process, and it still comes at zero cost to the User, but it might require another password-prompt.
    - If the Fixing-Tx succeeds, the Action-Tx fails, and the Human cancels the retry-attempt, they will still have paid the transaction-fees for the action transaction (and might have a few ADA hanging around in their servitor wallet which they may want to send back to their Owner-Wallet, as well as any dapp-specific consequences of the Fixing-Tx).

18. When the Action-Tx finally succeeds, the Intent updates the Human display accordingly.

8.2 Starting, Halting, and Unhinged Actions

Those are very straightforward and function like any other transaction - although they may include the Precon-system, they do not require co-signatures from Eigenvectors, and therefore no Tipping-UTxOs nor Tipping-Txs. They also lack the Election-Precon for the same reason.

Tracking of and connecting to Eligible Eigenvectors is still part of the User framework, but this is not enforced onchain.

9 Implementation

9.1 Tech Stack

Tiamat uses

- **Aiken** for the onchain code.[10]
- **Typescript and Butane's Blaze** for the offchain code.[3]

The offchain module is in principle the same both for the servers running the VECTORS and the frontend webapps. Certainly we can compress it further by deriving separate compiled packages for each of the two.

9.2 Communicating with the Node

VECTORS-operators run the Tiamat offchain code within a docker alongside their Cardano Node of choice. Since the only requirement towards the latter is that the rest of the network heeds it as a valid source of transactions, those will in practice be SPOs (Stake Pool Operators) too. Note that it suffices to run the VECTOR-code alongside the Relay-Nodes of the State Pool - further enhancing security, as Tiamat is not required to come anywhere near the Block Producing Nodes.

The VECTOR code communicates with the Cardano Node via Ogmios[5] and Kupo[9], as implemented by Butane's Blaze[3].

This has the advantage of keeping the VECTOR code separated from the actual Cardano Node by solutions that are yet maintained by other third parties, which are also open source - and well used by many, who we can hope will find any bugs or potential for exploits even without having to inspect the Tiamat-code itself.

9.3 SVMs from UTxOs

One of the biggest differences between the account-based model of Ethereum and the like, and the UTxO-based models of Bitcoin and Cardano is that the former has persistent objects with variable state, and the latter have transient objects with immutable state. In other words: The former has State Machines, the latter do not.

Now, State Machines are an ubiquitous primitive in computer science - after all, they describe many real-world entities quite well too. On Cardano, there have already been many dApps making use of them. The way they are constructed from raw UTxOs is as follows:

- A single state-UTxO exists at each point during the State Machine's lifetime.
- This carries the state, which consists of the UTxO's Datum and Assets.
- Further, it is marked as such by a special Thread-NFT. This is required, because there are no constraints on the creation of UTxOs - anyone can send any glsplasset they own to any address, with any Datum they want attached. But by adding in - and requiring - such a Thread-NFT, the contract code has a way to discern the actual state-UTxO from any potential impostors.
- Upon any attempt to spend this state-UTxO with the aim to enact a state transition, the contract tries to find an UTxO within that transaction's outputs that is created at the same address, and includes the correct Thread-NFT. If such a one is not found, the transaction fails.
- Now, the contract compares the Datums of the old and new state-UTxOs with the optional Redeemer attached by the sender, and checks that the state transition rules are met. If that is the case, the transaction passes; otherwise, it fails.
- Halting a State Machine works somewhat similarly, but instead of trying to acquire a new state-UTxO, the contract instead ensures the Thread-NFT is burned.
- Finally, starting a State Machine needs to check that the Thread-NFT is minted in the correct quantity and sent to the correct address, with the correct

initial state. The astute reader may notice that this check must now part of the Thread-NFT minting policy instead of the State Machine spending validator.

We augment those State Machines further by adding the option to force them to adhere to Tiamat's mempool consensus, by binding certain state transitions to the greenlighting of a Support-Vector-Set meeting Quorum. We thus rename the resulting two types of state transitions as

- **Revolving** State Transitions
- **Unhinged** State Transitions

As can be inferred, the former are the ones subjected to Support-Vectors' approval, and the latter are those that are not.

Finally, we add a sort of "escape hatch" to the former - if the Vectors get very lazy for too long a time, anyone can initiate a state transition, as if the action were Unhinged. This is to ensure user funds are never trapped in a scenario of total system death.

9.4 Tiamat Core SVMs

Tiamat Core comes with the following SVMs:

- **Matrix-SVM** carries the "ground truth" of the Tiamat-instance. In particular, the dApp's protocol parameters, the registrations of the Vectors as well as each's Eigenvalue, and the total balance of staked Eigenwert.
- **Nexus-SVM** handles the Elections and keeps a copy of the latest dApp's protocol parameters. It is included as read-input in revolving state transitions.
- **Vesting-SVMs** implement a linear vesting schedule. Their use case in Tiamat Core is to consitute the only way to unstake Eigenwert from the Matrix - it has to go into Vesting first. This is meant to implement a sort of smooth

stake lock (compare 11.2 to learn why that seems beneficial). Of course the vested balance may also be withdrawn immediately if it sent back into the MATRIX for re-registration of the VECTOR.

9.5 Payment in ADA

While the staked ASSET in any Tiamat-based DAPP is said instance's EIGENWERT, the payment from users towards VECTORS happens in ADA. This has the following advantages:

- No negative price impact on the EIGENWERT token as VECTOR-operators sell their rewards for fiat in order to fund their operations.
- No need to inflate the EIGENWERT token to pay the VECTOR-operators.
- No need for the users to acquire the EIGENWERT token before being able to use the DAPP.

Generally speaking: There is no need to use any other ASSET than ADA unless there is a very particular justification to do so, as for example in the case of protecting proof-of-stake-system from a dishonest majority [11.2].

10 Euclidean Type System and Proptesting Framework

In this section we present our framework for property-based testing and data serialization/deserialization/verification.

It was developed in the context of MetaDEX v1: Euclid[6], from which we derive the name.

10.1 Plutus Core Data Model

The (only) data types available on-chain in Cardano are:

- An **Integer** in Plutus Core is an arbitrary-precision signed integer. It can represent any whole number without size limitations, constrained only by the protocol parameters for practical purposes.
- A **ByteString** is a sequence of bytes, essentially an array of 8-bit unsigned integers. It is used to represent binary data and can store arbitrary sequences of bytes.
- A **List** is an ordered collection of data items of the same type. Lists can be of varying size.
- A **Tuple** is a fixed-size collection of two or more items, potentially of different types.
- A **Map** is a collection of key-value pairs, where each key is unique within the map. Keys and values can be of any data type, but all keys must be of the same type, and all values must be of the same type.
- The **Constr** type allows for the construction of user-defined data types by associating an integer *constructor index* with a list of arguments. This enables the representation of complex, algebraic data types.

10.2 Motivation

In contrast to those rather limited and abstract types, our offchain code written in typescript may want to use of more powerful object-oriented classes. While the low-level serialization and deserialization of the raw CBOR data into the schema of 10.1 is already well-managed by other modules, we not only still may want to verify their correct shape; but especially we may want to translate to and from more complex typescript objects, with potential additonal constraints of various forms.

Also compare section 5.10 for an introduction to property-based testing.

10.3 Implementation

10.3.1 Overview

The core of our Euclidean type system is the PType[16]. This typescript interface defines the following functions:

- `plift`: Parsing Plutus Core data into typescript objects.
- `pconstant`: Serializing typescript objects into Plutus Core data.
- `pblueprint`: Similar to `pconstant`, but instead of translating into the Plutus Core format, we target the format of Blaze's "Blueprint"[2]
- `genData`: generating random data of that type.
- `genPType`: generating random subtypes. This is technically not part of said interface, as those methods are static. The purpose here is to fill in generic types - for example, if we want to test a "list", we need to be able to generate the types of the entries in that list before we can generate the actual entries.
- `showData`: printing data of that type.
- `showPType`: printing the type.

[16]Which originally was a nod towards Plutarch, but can be also interpreted as shorthand for "Parsed Type".

10.3.2 Utility PTypes

Coming soon!

11 Security Analysis

In this section we discuss our various considerations regarding the security aspects.

11.1 Atomic Actions

11.1.1 Control of Funds

As Tiamat allows to to go back to naive implementations of most business logics, we do not have to deploy schemes where the user yields control of their funds as much. We rather can either implement our actions atomically straight away, or at a minimum require co-signatures from the user in order to do anything affecting their funds. We can even do so without impacting user experience by requiring only the Servitor-Wallet to sign off certain actions - it is increasingly unlikely that an attacker would manage to compromise the user's browser *and* a suffient share of the Vectors *at the same time*. And even then, as mentioned, the Owner-Wallet could still be required for the more impactful decisions. In concert with the servitor-system we do not need to worry too much about the displeasure of having to enter our password into our browser-wallet-extension once, then letting the servitor handle the rest.

11.1.2 MEV Mitigation

Since all action-effects are encapsulated by a single transaction, no frontrunning or other sorts of MEV is possible. At worst, the Eigenvectors can censor the transaction, which would require collusion of the majority of that Cycle's Eigenvectors, which in turn would not be maintainable over multiple Cycles, unless the malicious parties control a majority of the stake.

Which leads us to

11.2 Dishonest Majority

The scenario of a dishonest minority in any sort of proof-of-stake system is straightforwardly dealt with - they incur one sort of punishment or the other, be it via direct slashing or missing out on rewards.

The more interesting challenge however is posed by a dishonest majority - how are they to be sanctioned? After all, the contracts do not know right from wrong; the only way blockchain systems can reason about that is by relying on some sort of HUMAN ground truth data input - and the shape this takes is, generally speaking, that of consensus. Or, in other words, the majority determines who is in the right, and who is to be punished.

Now, there exists an escape hatch - market forces.

Consider that a dishonest majority would require a significant investment in order to proceed. If that investment were to be made in some sort of currency with other use cases - say, USD as backing for the Cardano blockchain instead of the native ADA, or ADA for any DAPP running on Cardano instead of a dedicated asset - any actor with enough capital at their disposal could simply out-stake the honest minority, completely dominate the ground truth of the system in question, do what they want, and leave again unharmed (and likely with their pockets lined fraudulently).

Yet if one considers the variant where there's a dedicated staking-token for the project in question, the "leaving unharmed" is not as easy. In fact - and especially when using stake locking mechanisms - the market would soon be notified about such an abuse by the honest minority, which would send the token price to zero. The honest minority could then proceed to fork everything, and re-issue a new copy of the token in the same proportions of the previous one at the last snapshot before the fork, with the only exception that the dishonest majority would not receive any of it. The market then would be able to reallocate their capital to the fork (which, being purged of the rapscallions, is presented in an even better light than before, depending on interpretation of the entire happening, of course).

Therefore, in Tiamat, we use a dedicated token for staking, and a soft stake-lock in the form of linear vesting of unstaked EIGENWERTE.

As a final note: One needs to take care to ask about the value-at-risk-to-stake-ratio as well. Because if the total value poised to be stolen exceeds the cost of the attack

outlined, the latter becomes feasible yet again. In Tiamat Core, however, there is no value at risk from that route, as the user needs to sign off any movements of their funds.

11.3 SVM escape hatches

In case the system goes offline, SVMs have a built-in "escape hatch" - meaning, if they haven't been revolved in some time, anyone can do so. This means that users will be able to access any funds left in the system in such a scenario. Compare Section 9.3.

This also implies that in the case of a dishonest majority - or rather, supermajority, as they would be able to dominate *every single* CYCLE for the duration of the attack - aiming to "stagger"-lock users' funds by censoring all their actions, ongoing payments would be needed to keep this up, in order to prevent said escape hatch from activating.

11.4 Malicious Data Misrepresentation

Another potential avenue for malicious VECTORS is **data misrepresentation**. This, however, is easily dealt with in a number of ways:

- Adding blockfrost as a secondary data source to double-check settled UTXOs. This does not work on the current MEMPOOL, however, as Blockfrost can only provide MEMPOOL data submitted to their endpoints. Which would be somewhat tricky to process to begin with.
- Noting that malicious data misrepresentation does not affect the transactions users sign - after all, those include exact output-UTXOs. In other words: The user signs off what they are paying, and what they are receiving. If any of that is wrong, the transaction will fail. It is however the duty of the browser-wallet-extension to display correctly the funds involved in the transaction the user is prompted to sign.
- Noting that malicious data misrepresentation does require collusion of **all** current EIGENVECTORS, not just a majority.

12 Comparison

Coming soon!

13 Future Work

In this section we outline the future direction the project may take in the near term.

13.1 Delegated Proof-of-Stake

With extremely minor changes we can allow anyone to delegate their EIGENWERT to any VECTOR without having to run the software themselves. However, in order to to cement the appropriate profit sharing in the protocol, some additional considerations will be required.

13.2 Side Protocols

Likewise, with some minor adjustments Tiamat can easily be developed into a sort of lightweight "sidechain" framework, or a suite thereof. Those would of course be rather rudimentary compared to more sophisticated solutions; the purpose would not be to manage financial assets, but rather serve as a multi-purpose device for any sort of inconsequential protocols, like for example **gaming**.

13.2.1 Gaming

By being interlinked with the Cardano Mainnet, we can still manage the relevant assets on the security-focused chain, while only referencing them on the free side protocol. This is inspired by the approach of Pavia[11].

That would further allow us to re-use in-game items in multiple games, which includes forks and mods of the same game. One may imagine for example a "sword" item, ownership of which would be represented securely by an NFT on the Cardano Mainnet, which would carry various meanings in multiple games, similar or different; i.e.

- A "sword" card in a card game

- A "sword" card in an otherwise identical fork of said card game
- A "sword" card in a fork of said card game that implements a mod
- A "sword" item in some sort of real time action rpg
- A "goose" item, hypothetically speaking
- etc.

In this manner we will be able to combine the security of Cardano with the low cost required for most internet applications.

13.2.2 AI

Of course, no whitepaper written in current year is complete without the mention of artificial intelligence. Yet the author has been thinking about this for over six years now; in fact, the original motivation to enter the blockchain space in the first place was the line of thought "Would it not be preferable to allow anyone to contribute to this exciting development, and take a fair share of the fruit of their labor, in proportion to their contribution, instead of having to go through the bottleneck of limited and comparatively uninspired machine learning positions?"[17]

Some of the drafts we developed already require some low-cost side-protocols. Ideally we would leverage Tiamat here as well, to bring yet unseen innovation in the area to the Cardano ecosystem.

The reader may please apologize our occasional lapse in constraint and the resulting, in our view, somewhat uncouth salesmanship. But rest assure, we do believe it, and we hope that the community may too, when more information will finally be ready to be made available to the public.

[17]This was before we averaged about twenty AI thinkpieces a day, and the whole matter was still looked down upon by anyone but a few freaks, one of which was - and is - yours truly

13.3 Ecosystem Incentives

As already disussed in Sections 5.14 and 6.11, the idea is to spark a vibrant developer community around this project. Besides releasing the code, we intend to

- Host regular hackathons,
- Provide ample documentation and in-person developer support as well as courseware, and
- Fund projects building on Tiamat.

By combination of those four we hope to catapult Cardano's developer scene to the next level.

13.4 MetaDApp

While Tiamat is intended first and foremost to host a single dApp, nothing prevents us from developing their capabilities far beyond the original, concentrated use case.

The logical next step would be a sort of "Meta-DApp", not unlike WeChat or the App Store, where individual developers can contribute their particular smaller applications. The main advantage of such a thing would be that all those dApps would be secured by the same underlying Eigenwert token; however, a counterpoint may be that Tiamat Core already is designed with the idea of trustlessness in mind.

We also want to say there may be synergistic effects, but again, the core Tiamat architecture is already designed in a manner to allow smooth and instant - within the same transaction, even - interoperability between different Tiamat-instances.

Nevertheless, the very thought is too exciting to us to not share it here.

14 Conclusion

We present Tiamat, a novel framework for developing high speed dApps on the Cardano blockchain. Besides a number of utility features, the main contribution we make is the unleashing of the full potential of multi-party transaction chaining.

We achieve this by creating a secure proof-of-stake system with redundant nodes, called Vectors, which maintain a mempool consensus, and a corresponding on-chain contract suite which enforces adherence thereto.

We proclaim our intent to spark an independent and vibrant developer community by releasing this project freely and open source, both around Tiamat, MetaDEX, Cardano, and the blockchain ecosystem as whole.

Blessed be the Spirit of this Device.

References

[1] Axo Finance. *Axo*. URL: https://www.axo.trade/.

[2] Butane Team. *Blaze Blueprint*. URL: https://github.com/butaneprotocol/blaze-cardano/tree/main/packages/blaze-blueprint.

[3] Butane Team. *Blaze: The hottest transaction building library for Cardano*. URL: https://github.com/butaneprotocol/blaze-cardano.

[4] *Ethereum MEV Documentation*. Ethereum Foundation, 2024. URL: https://ethereum.org/en/developers/docs/mev/.

[5] Grav and Hugo. *Ogmios*. URL: https://ogmios.dev/.

[6] Chris M. Hiatt. *Euclid Offchain Type System and Property Testing Framework*. 2023. URL: https://github.com/metadex-fi/euclid-offchain/tree/main/src/types.

[7] Chris M. Hiatt. *The UTXO-DEX-issue (and my idea of a solution)*. 2021. URL: https://www.youtube.com/watch?v=_wVpC7XWN1M.

[8] Hummingbot Foundation. *Hummingbot*. URL: https://hummingbot.org/.

[9] KtorZ. *Kupo*. URL: https://github.com/CardanoSolutions/kupo.

[10] Lucas Rosa, Kasey White, Matthias Benkort. *The Aiken Programming Language*. URL: https://aiken-lang.org/.

[11] *Pavia Metaverse*. Pavia.io. URL: https://www.pavia.io/.

[12] *Sandwich Attacks in Crypto and How to Prevent Them*. CoinGecko, 2024. URL: https://www.coingecko.com/learn/sandwich-attacks-prevention-crypto.

[13] Zygomeb. *Leviathan: Turning Liveness Into Finality*. Optim Finance, 2024. URL: https://www.optim.finance/docs/Leviathan_Whitepaper_2.pdf.