[

Ethereum intent network

1024×1024 366 KB

](https://ethresear.ch/uploads/default/original/2X/2/269f3eb4c2a042598c3ffc03d97de6e966192830.jpeg)

Thanks to @0xapriori, @adrianbrink, @awasunyin, and @0xemperor for reviewing this post. This is an active draft and further feedback (both solicited and unsolicited) is very welcome. All opinions and errors within are my own.

One-line summary

Anoma brings a universal intent machine to Ethereum, allowing developers to write applications in terms of intents, which can be ordered, solved, and settled anywhere in the Ethereum ecosystem.

tl; dr summary

- An intent

is a commitment to user preferences and constraints over the space of possible state transitions.

- The Anoma protocol brings a universal intent machine to Ethereum, allowing developers to write applications in terms of intents instead of transactions.
- Anoma is an interface

, not an intermediary

- it's not another MEV redirection device.
- Anoma provides a universal intent standard which does not constrain what kinds of intents can be expressed, but allows for built-in state, network, and application interoperability.
- Intents and applications written with Anoma can be ordered, solved, and settled anywhere - on the Ethereum main chain, on EVM and non-EVM rollups, on Eigenlayer AVSs, on Cosmos chains, Solana, or any sufficiently programmable state machine.
- Anoma provides four key affordances: permissionless intent infrastructure

, intent-level composability

, information flow control

, and heterogeneous trust

- using three mechanisms: the resource machine

, the heterogeneous trust node architecture

, and languages for explicit service commitments

.

- Anoma is compatible with any topology the Ethereum network chooses.

Table of Contents

- Introduction
- Motivations
- Personal interlude
- Some definitions
- Interfaces not intermediaries
- Where are we at?
- What does a "universal intent machine" mean for Ethereum?
- Why use Anoma?

- Motivations
- Personal interlude
- Some definitions
- Interfaces not intermediaries
- Where are we at?
- What does a “universal intent machine” mean for Ethereum?
- Why use Anoma?
- Intent-centric applications with Anoma and Ethereum
- Architecture
- Affordances
- Permissionless intent infrastructure
- Intent-level composability
- Information flow control
- Heterogeneous trust
- Permissionless intent infrastructure
- Intent-level composability
- Information flow control
- Heterogeneous trust
- Mechanisms
- Resource machine
- Heterogeneous trust node architecture
- Service commitment languages
- Resource machine
- Heterogeneous trust node architecture
- Service commitment languages
- Example applications
- Multichat
- Public Signal
- Scale-free money
- Promise Graph
- Multichat
- Public Signal
- Scale-free money
- Promise Graph
- Affordances
- Permissionless intent infrastructure
- Intent-level composability
- Information flow control

- Heterogeneous trust
- Permissionless intent infrastructure
- Intent-level composability
- Information flow control
- Heterogeneous trust
- Mechanisms
- Resource machine
- Heterogeneous trust node architecture
- Service commitment languages
- Resource machine
- Heterogeneous trust node architecture
- Service commitment languages
- Example applications
- Multichat
- Public Signal
- Scale-free money
- Promise Graph
- Multichat
- Public Signal
- Scale-free money
- Promise Graph
- Topology
- Rollups and L2s
- Plasma
- EigenLayer and service providers
- Topology implementation requirements
- Rollups and L2s
- Plasma
- EigenLayer and service providers
- Topology implementation requirements
- Architecture
- Affordances
- Permissionless intent infrastructure
- Intent-level composability
- Information flow control
- Heterogeneous trust
- Permissionless intent infrastructure

- Intent-level composability
- Information flow control
- Heterogeneous trust
- Mechanisms
- Resource machine
- Heterogeneous trust node architecture
- Service commitment languages
- Resource machine
- Heterogeneous trust node architecture
- Service commitment languages
- Example applications
- Multichat
- Public Signal
- Scale-free money
- Promise Graph
- Multichat
- Public Signal
- Scale-free money
- Promise Graph
- Affordances
- Permissionless intent infrastructure
- Intent-level composability
- Information flow control
- Heterogeneous trust
- Permissionless intent infrastructure
- Intent-level composability
- Information flow control
- Heterogeneous trust
- Mechanisms
- Resource machine
- Heterogeneous trust node architecture
- Service commitment languages
- Resource machine
- Heterogeneous trust node architecture
- Service commitment languages
- Example applications
- Multichat

- Public Signal
- Scale-free money
- Promise Graph
- Multichat
- Public Signal
- Scale-free money
- Promise Graph
- Topology
- Rollups and L2s
- Plasma
- EigenLayer and service providers
- Topology implementation requirements
- Rollups and L2s
- Plasma
- EigenLayer and service providers
- Topology implementation requirements
- Challenges and tradeoffs
- Request for comment

Introduction

In this post, I will describe how Anoma can provide a universal intent machine for Ethereum - both the Ethereum base chain and the Ethereum ecosystem of rollups, searchers/solvers, and other network participants. What is

a universal intent machine, you might ask? In what follows I shall attempt to clearly define this concept and what it means for the Ethereum ecosystem - but first I want to explain my motivations, tell you a bit about my personal history with Ethereum, and clarify some essential background context.

Motivations

I have three key motivations in writing this piece. First, I want to establish a conceptual and discursive bridge to the Ethereum research community, which Anoma has historically operated somewhat separately from. I think that this separation was necessary in order for us to independently develop useful concepts, but the time has now come to reunite. Second, I want to clearly articulate to potential developers and users of applications built with Anoma what affordances Anoma can provide and why. I do this in order to set accurate expectations for what I think the technology - and often, any intent technology - can and cannot do, so that developers and users can make informed choices. Third, I want to reach out to Ethereum application developers in order to better understand their needs, see which of our ideas might be most helpful, and map out who specifically might be interested in collaborating on what.

Personal interlude

Forgive me a bit of self-indulgence here, but I think it will help contextualize the history.

My blockchain development journey actually started with Ethereum. In the summer of 2017, I started playing around with the system, and I was looking for a good starter Solidity development project to learn how smart contracts worked. These were the early days of decentralized exchanges (pre-Uniswap). At the time, I briefly used a since-defunct exchange called EtherDelta, and I thought it was a shame that someone had gone to all the trouble to build a complicated smart contract, orderbook server, and web frontend to support only ERC20 fungible tokens, when - to me - a key benefit of Ethereum's fully-featured VM was the ability to generalize and support any asset representation a user wanted. I thought the best way to test this hypothesis was to try to implement it myself, so I did - and I decided to call it "Wyvern".

Wyvern was really a proto-intent system - and the whitepaper I wrote at the time uses the word - but I didn't have a clear mathematical concept in mind then, nor an understanding of what a generalized intent system would really entail. Nevertheless, working on Wyvern taught me quite a bit about how Ethereum and the EVM worked, and also that they might

not be the best fit for intents - or, at least, didn't seem to have been designed for them. Around the time I published the first version of Wyvern, a startup company was founded by Devin Finzer and Alex Atallah called OpenSea. OpenSea - aiming to create a marketplace for non-fungible collectibles, and thus in need of a more generalized exchange protocol - became the first (and more-or-less only) user of Wyvern, which they used until 2022. I didn't do much work on Wyvern after 2018 - but luckily, since it was already generalized, OpenSea could add features such as batch sales and multi-asset payments without needing any changes to the core protocol.

Aside: Believe it or not, the Wyvern contracts areapparently still #1 in Ethereum DEX state consumption. In retrospect, I really wish I had optimized my Solidity code. (I also looked for historical gas consumption data, but didn't easily find it)

After building Wyvern, I wanted to explore other aspects of blockchain systems design - particularly interoperability - and I was lucky enough to land a position at the (now defunct) Tendermint company, where I worked on the Cosmos project, specifically IBC, until the launch of IBC in 2021. After Wyvern and IBC, I felt like I had a few key pieces of the protocol puzzle, but not the whole picture - so I decided to co-found Anoma with @awasunyin and @adrianbrink to figure out if I could piece together the rest. Anoma started with a vision, but little idea of how exactly to implement it. This is the primary reason why we didn't engage with the Ethereum ecosystem earlier on - we didn't have a clear idea of what we wanted to do, and we didn't know what Anoma could offer. Through years of research and design - most of it not by myself but rather by the brilliant and compassionate folks in the Anoma research ecosystem - we've come to a much better understanding of what Anoma is, and we now have a much better idea of what Anoma can bring to the Ethereum ecosystem. I'm happy to be coming back.

Some definitions

Before discussing Anoma and Ethereum, I want to clarify what I mean by those words. In common language, words for blockchain networks, such as "Ethereum", "Cosmos", and "Anoma", are commonly used to bundle together six distinct components:

- A protocol

, defining what transactions do - in Ethereum's case, the EVM.

- A specific security model

, defining how blocks are produced - in Ethereum's case, Gasper.

- A network

or ecosystem

of connected machines (physical and virtual) - in Ethereum's case, the Ethereum ecosystem, including the base chain validators, rollups/L2s, bridges, off-chain services, etc.

- A history

(all blocks since genesis)

- A nominal asset
- in Ethereum's case, ETH.
- A community

of people who self-identify as members - in Ethereum's case, the Ethereum community.

In this post - which is addressed to the Ethereum community

- I shall be talking only

about possible relationships between the Ethereum and Anoma protocols

and networks

- so when I use the word "Ethereum" or "Anoma", I mean only these components. Possible relationships between histories

, assets

, and security models

are a fascinating topic, but one which I think had better be covered in a separate post.

Interfaces not intermediaries

I also want to clarify that Anoma aims to provide a universal intent machine interface

for applications - not

an intent intermediary

. What's the difference? An interface

, as I use the word here, is simply a protocol which translates, or represents, one semantics - in this case, declarative intent semantics - in terms of another - in this case, imperative ordering, compute, and storage semantics of underlying machines. TCP/IP, for example, is an interface - TCP translates the semantics of declarative ordered packet delivery into imperative send, retry, and window management semantics of underlying network hardware. An intermediary

, on the other hand, is a particular network participant (possibly a chain or network) through which data (such as intents) flow. Many bridges in the Ethereum ecosystem - multisignature and chain alike - are intermediaries, as are, for example, banks in the US banking system which simply reissue dollars. Interfaces

are simply code - freely copied, implemented anywhere, and usable by everybody. Intermediaries

, however, are actors in a network - sometimes valuable ones - but their privileged positions often allow to extract rents, introduce additional security assumptions, or add unnecessary latency and complexity. The Anoma protocol is an interface - not an intermediary. There is no "Anoma chain" to which you must send intents.

Where are we at?

The final (I promise!) clarifying note: Anoma is in active development and not yet finished. Ongoing research will continue to evolve the live system. We've finished enough research (indexed here) that the overall design is pretty clear, but some details and priorities in implementation and deployment are not yet fixed. We think the design is at a stage which is possible to articulate and communicate, and we want to get feedback on it from Ethereum's perspective (hence this post). The first full implementation of Anoma (in Elixir!) is also in the works, and will be open source soon.

What does a "universal intent machine" mean for Ethereum?

At a high level, three things:

1. Using Anoma, developers can write applications in terms of intents and distributed intent machines, instead of transactions and specific state machines. These intents can request particular network actors to perform roles such as computational search, data storage, intent matching, and transaction ordering, and they can disclose specific information to selected parties chosen either initially by the user authoring the intent, or programmatically during the intent matching and settlement process.

2. Anoma provides a universal standard for intent and application formats which does not constrain what kinds of intents can be expressed (beyond the fundamental constraints - e.g. intents must be computable functions), but allows for state, network, and application interoperability. Broadly, Anoma standardizes what it takes to verify that an intent has been satisfied - but not how the solution is computed.

3. These intents and applications written with Anoma can be ordered, solved, and settled anywhere - on the Ethereum main chain, on EVM and non-EVM rollups, on Eigenlayer AVSs, on Cosmos chains, Solana, etc. - anywhere an Anoma protocol adapter (which the post will cover in more detail later) is deployed.

The rest of this post will expand on the details here - but first, I want to motivate why we think this kind of universal intent machine standard may be compelling.

Why use Anoma?

In my view, a universal intent machine standard has three key benefits: intent-level and intent machine composability

, application portability

, and the availability of permissionless intent infrastructure

. I'll expect upon these aspects further in what follows, but in brief:

- Intent-level composability allows applications to compose interactions at the intent level, not just the transaction level. This unifies liquidity (as much as possible given heterogeneous preferences) and allows users to precisely select which decisions they would like to make themselves, which decisions they would like to delegate to specific network operators, and what constraints they would like to be enforced on those delegated decisions.

- Intent machine composability allows users to treat a distributed system of many computers, chains, and networks, as if it were a single intent machine, while that intent machine is internally composed of many smaller or special-purpose intent machines networked together.

- Application portability allows applications to move freely across concurrency and security domains without additional development work or protocol incompatibility barriers. Applications written for Anoma can treat the entire Anoma network as a virtualized state space - they do not need to be deployed separately to different chains, integrate different bridges, or pick any specific security model.
- Permissionless intent infrastructure allows applications to make use of existing nodes and chains running the Anoma protocol. Of course, what nodes choose to do is up to them - but a standardized protocol allows application developers to think only about their application, make use of services provided by existing node operators, and not spend any time building complicated, custom off-chain infrastructure - and it allows node operators to focus purely on the services they want to provide and the conditions under which they want to provide them, without needing to care (or even know) which applications are using those services.

Now, I'd like to hope that maybe you're interested (or, if not, you probably wouldn't be reading this line of text anyways). What do intent-centric applications actually look like?

Intent-centric applications with Anoma and Ethereum

For more information on what Anoma means by the word "intent", seethis blog post.

I'm going to split this section into two parts: architecture

and topology

. Let me first define what I mean by those concepts. The architecture

of a protocol is a mathematical specification of what the protocol is and does - typically, what (complex) pattern of messages will be sent in response to (complex) patterns of messages received. The topology

of a network is the specific structure of connections - in the case of intent-based systems, connections induced by user choices of what intents to craft and where to send them. I find drawing a clear line between architecture

and topology

very helpful in designing and analyzing distributed, networked systems, for three reasons:

- The architecture

and topology

are always cleanly separable. Particular identities or connections can only be configuration parameters of a particular protocol - one could always copy the protocol (in design or in code), and change the particular parameters of who to connect to - thus keeping the same architecture

but picking an arbitrarily different topology

. More limited architectures may support only a certain subset of topologies, or may provide certain guarantees only for a certain subset of topologies - but the particular topology is always a matter of runtime configuration.

- The architecture

and topology

are chosen by different parties - the architecture

of a particular protocol is chosen by whoever designed the protocol, while the topology

of a network using that protocol is chosen (in a distributed fashion) by the users of that network making choices. Wearing my protocol designer hat, I consider it my responsibility to clearly define the architecture

- but I have neither desire nor ability to influence the topology

, which is a function of decisions made by network participants, themselves often influenced by cryptoeconomic parameters.

- Although the protocol designers may be subject to incentives of their own, the architecture

of a particular protocol is fixed - in a sense, it is what defines the protocol - so one can easily analyze a particular architecture as a discrete, static mathematical object. The topology

of a live network, however, changes all the time, and is subject to incentives both inside and outside the domain of what is legible to the protocol. In other words, architectural definition

precedes topological analysis

(using game theory, mechanism design, or similar). In order to understand how participants might use a language, one must first define the semantics which that language can express.

Architecture

Let's start with Anoma's architecture. What kind of architecture does Anoma have, and what can this architecture provide? In this section, I will define the architecture in two levels: the affordances

which Anoma's architecture offers to applications, and the mechanisms

with which Anoma provides these affordances. I will then detail a few example applications

which I'm particularly excited about.

Note: I have tried to select a level of abstraction which will provide a solid intuition for how Anoma works and why without getting lost in extraneous implementation details - but I certainly will not have selected optimally for all audiences (or even any), so please let me know where you would like more detail and which parts don't make sense.

Affordances

I've borrowed the word "affordance" from cognitive psychology - in this context it means, simply, what application developers can do

with Anoma - what capabilities of application design and instantiation Anoma offers that they didn't have before. In contrast to, for example, the perspective of the protocol, affordances describe the perspective of the application developer. To applications, Anoma provides four key affordances: permissionless intent infrastructure

, intent-level composability

, information flow control

, and heterogeneous trust

. I will detail each of these in turn.

Permissionless intent infrastructure

Permissionless intent infrastructure

means that - once Anoma is live - developers will be able to write complex intent-centric applications, which might need solvers, various intent pools, and multiple consensi, and deploy them to the Anoma network directly without needing to develop or operate any bespoke off-chain infrastructure of their own. The word "permissionless" can be a bit misleading - in any multi-user interaction, you're interacting with someone

, and they could always choose not to interact with you - but here I mean simply that Anoma provides intent infrastructure with no specific

party (or parties) whose permission you must seek in order to use it.

This is possible because Anoma nodes are topology-agnostic. Depending on the operator's configuration and the intents received, they can act as consensus nodes, storage providers, solvers, gossip forwarders, or any other role - not just for one chain but for any number. For Anoma, topology is a matter of runtime configuration - often even negotiated over the network. What consensi users want shifts as demand shifts for particular applications and atomicity between particular partitions of state, and consensus providers must themselves shift to meet this demand.

One concern with intent systems raised by, among others, Paradigm and Flashbots, is that intent systems could lead to centralization if specific parts of the intent lifecycle are performed by designated trusted parties (who would then have an undue influence). On the other hand, Chitra, Kulkarni, Pai, and Diamandis recently showed that competitive solver markets could drive most solvers out and lead to oligopoly. Permissionless intent infrastructure does not change the mechanism design landscape, but it does remove all protocol barriers to decentralization and mechanism experimentation. The likelihood of designated trusted infrastructure or oligopoly is much lower when intents are built into the core protocol and nodes can be spun up on demand, and tracking intent satisfaction in the protocol allows for users to much more easily automatically switch between providers (or credibly threaten to) when their interests aren't being sufficiently optimized for.

Intent-level composability

Intent-level composability

means that application interactions can be composed at the intent level instead of at the transaction level. If applications use intents, but can be composed only with transactions, liquidity is fragmented between applications, and users cannot access

the entire intent pool without writing an application intent aggregation interface on top of all the applications which provide liquidity of a relevant nature. Intent-level composability unifies this intent liquidity pool, such that users' intents can be composed even with intents from another application the user has never heard of and never needs to think about - arbitrary composition is possible as long as the criteria in the user's intent are satisfied.

Intent-level composability also opens up efficiency improvements possible only when users can articulate precisely the nature and granularity of their preferences, such that their intents can be more flexibly composed - and more surplus returned back - than if the user had articulated more specific preferences than they actually have. Suppose that I want to swap USDC for ETH, and I'm happy to receive ETH on either the main chain, Optimism, or zkSync. With intent-level composability, my intent can be matched with the best offer available to settle on any of those chains - or perhaps even partially settled in multiple places, if I'm willing to accept the promise of a staked solver with liquidity in multiple places.

Information flow control

Information flow control

means that developers of Anoma applications - and users of these applications - can reason precisely about what information actions taken in their application disclose to whom. Anoma provides information flow control at three distinct levels of the system:

- State-level information flow control

describes what can be seen by whom after transaction creation and execution. For example, a shielded transaction (as in Zcash) only reveals specific state changes and a proof that they satisfied required invariants, while a transparent transaction reveals all involved data to all observers.

- Intent-level information flow control

describes what can be seen by whom during intent solving, intent composition, and transaction creation. For example, a user may elect to disclose certain information to well-known solvers in order to help those solvers find valid matches quickly, but elect not to disclose that information to other solvers which they do not know.

- Network-level information flow control

describes what metadata going around the network can be seen by whom. For example, a user may elect to disclose certain physical network addresses and transport options - the Bluetooth ID of a phone, for example - only to a few well-known friends, lest the information revealed by the address render the user vulnerable to denial-of-service or deanonymization.

Information flow control is a declarative specification of what should be disclosed to whom under which conditions. Desired information flow control properties constrain, but do not fix, the choice of specific cryptographic primitives such as regular encryption, succinct zero-knowledge proofs, or partially or fully homomorphic encryption - as long as the primitives chosen preserve the desired properties, the choice can be made on the basis of implementation availability, computational efficiency, and cryptographic interoperability. This is not a new concept - it is pretty well-covered in existing computer science research literature. In particular, the framework used by Anoma has been inspired and informed by the Viaduct paper, although Anoma differs in providing a dynamic runtime instead of a one-shot compiler.

Heterogeneous trust

Heterogeneous trust

means that Anoma application developers and users can make their own assumptions about the behavior of other participants on the network and their own choices about who to entrust with specific service-provisioning roles in the operation of their application, while reasoning about these assumptions and choices explicitly and detecting when their assumptions do not match their observations. In particular, most applications need three basic services: reliable ordering

of intents and transactions related to the application, reliable storage

of application data, and efficient compute

of new application states and temporal statistics. Anoma makes all three of these services programmable:

- Using programmable ordering

, users and developers can choose who will order their intents and transactions. These choices may be made independently for each intent, and conditionally delegated to third parties. For example, a user may request their intent to be ordered by A or B, and delegate the choice of whether it is in fact ordered by A or B to a solver, to be made on the basis of relative price.

- Using programmable storage

, users and developers can choose who will store the data needed by their application. These choices may be made independently for each intent and each piece of data, conditionally delegated to third parties, and changed over time. For

example, an application may choose to store recently-used data with the same nodes who will order its transactions, but move stale data to a cheaper long-term storage provider.

- Using programmable compute

, users and developers can choose who will perform the computation required for their intents, transactions, and ongoing application usage. For example, an application may choose to pay low-latency solver nodes for the compute required to match intents, but pay higher-latency but cheaper-per-FLOP server farms for long-term statistical indexing.

Heterogeneous trust is also a pre-existing concept in the academic research literature, from which we've been lucky to draw many ideas, such as Heterogeneous Paxos. A lot of study still remains here, however - and I'm hopeful that blockchain projects (who desperately need this research) can step up a bit more to help organize and fund it.

Mechanisms

How is all of this implemented? This post is already long enough without fully specifying exactly how Anoma works, but I will detail three key mechanisms here: the resource machine

, which implements intents, the heterogeneous trust node architecture

, which supports many networks with one piece of node software, and languages for explicit service commitments

, which match user requests and operator offers for ordering, compute, and storage services.

Resource machine

The Anoma resource machine

implements intents without loss of generality. In relation to the affordances above, the resource machine provides state-level and intent-level information flow control, intent-level composability, and the programming framework for ordering, storage, and compute services required for heterogeneous trust. Heterogeneous trust also requires cross-domain state synchronization and verification, which the resource machine's state architecture is designed to make simple and efficient.

Note: I've chosen to simplify some of the definitions here for conceptual legibility of the key intuitions. For full details, please see the resource machine ART report.

The basic concept of the resource machine is to organize state around resources

. Resources are immutable: they can only be created and consumed exactly once. The current state of the system can be defined as the set of resources which have been created but not yet consumed. Each resource has an associated predicate called a resource logic

that specifies the conditions under which the resource can be created and consumed. These conditions could include, for example, the creation or consumption of other resources, data such as a signature over a specific payload, or a proof that state elsewhere in the system satisfies a certain property. A transaction is considered valid only if the logics associated with all resources created and consumed in the transaction are satisfied, the transaction balances (enforcing e.g. linearity of currency), and no resources consumed in the transaction have been consumed before (no double-spends).

[

Resource logic examples

2001×1072 254 KB

](https://ethresear.ch/uploads/default/original/2X/1/1a560ae66942cc03fa5434d35fa54aa5cb46b2ee.png)

Diagram credit Yulia Khalniyazova

Formal definitions

A resource is a tuple R = (l, label, q, v, nonce)

where:

- Resource = \mathbb{F}*{l} \times *mathbb{F}*{label} \times \mathbb{F}*Q* \times *mathbb{F}*{v} \times \mathbb{F}_{nonce}
- l: \mathbb{F}_{l}

is a succinct representation of the predicate associated with the resource (resource logic)

- label: \mathbb{F}_{label}

specifies the fungibility domain for the resource

- q: \mathbb{F}_Q

is an number representing the quantity of the resource

- v: \mathbb{F}_{v}

is the fungible data of the resource (data which does not affect fungibility)

- nonce: \mathbb{F}_{nonce}

guarantees the uniqueness of the resource computable components

From a resource r

may be computed:

- r.kind = h_{kind}(r.l, r.label)
- r.\Delta = h_{\Delta}(r.kind, r.q)

A transaction is a composite structure TX = (rts, cms, nfs, \Pi, \Delta, extra)

, where:

- rts \subseteq \mathbb{F}_{rt}

is a set of roots of the commitment tree

- cms \subseteq \mathbb{F}_{cm}

is a set of created resources' commitments.

- nfs \subseteq \mathbb{F}_{nf}

is a set of consumed resources' nullifiers.

- \Pi: { \pi: ProofRecord}

is a set of proof records.

- \Delta_{tx}: \mathbb{F}_{\Delta}

is computed from \Delta

parameters of created and consumed resources. It represents the total delta change induced by the transaction.

- extra: {(k, d): k \in \mathbb{F}*{key}, d \subseteq \mathbb{F}*{d}}

contains extra information requested by the logics of created and consumed resources

A transaction is considered valid

with respect to a previous state if and only if:

- rts

contains valid commitment tree roots that are correct inputs for the membership proofs

- nfs

contains valid nullifiers that correspond to consumed resources, and none of these nullifiers have been previously revealed

- input resources have valid resource logic proofs and the compliance proofs associated with them
- output resources have valid resource logic proofs and the compliance proofs associated with them
- \Delta

is computed correctly

A transaction is considered balanced

if and only if \Delta = 0

.

Applying a transaction to the state simply entails adding new commitments and nullifiers to their respective Merkle tree and set. Transaction candidates

- functions which generate transactions - are provided access to the current state and can look up resources by kind, which allows for post-ordering state-dependent updates without conflicts.

Compared to the EVM (and other blockchain VMs such as the SVM and Move), the resource machine unbundles three aspects which these VMs intertwine: the state architecture

, the instruction set

, and the message-passing model

. The EVM, for example, specifies:

- A state architecture

where state is read and written in 256-bit blocks, partitioned by smart contract addresses, and encoded into a Merkle-Patricia Trie.

- An instruction set

for stack-based computations with a 256-bit native word type and an assortment of specially optimized Ethereum-related instructions (e.g. ecrecover

).

- A message-passing model

where one smart contract owns the execution frame at a time, and sending a message to another contract switches control of the execution frame to the message recipient.

The resource machine, by contrast, specifies only

the state architecture. Different instruction sets can be chosen by different operators as long as valid state updates are ultimately produced, and messages can be passed in whatever fashion is most efficient in each particular case. This unbundling is necessary for native intent support, as with intents, much of the computation (expressed in an instruction set) and communication (expressed by a message-passing model) happens before

final transaction execution and state change verification. With the resource machine, the instruction set can be chosen by whomever is performing the computation in question, and the message-passing model can simply reflect the actual intent-matching topology.

Heterogeneous trust node architecture

[

P2P overlay diagram

1215×735 49 KB

](https://ethresear.ch/uploads/default/original/2X/f/f70d2a9b2dae614e776808b0562601a340536cd9.png)

Diagram credit Naqib Zarin

Anoma’s heterogeneous trust node architecture

weaves together the key ordering, storage, and compute functionalities needed to provide permissionless intent infrastructure into a single piece of node software. The base of the node stack is a generalized heterogeneous P2P network stack with support for network-level information flow control. The node architecture can be split into two components: the networking machine

, which implements the heterogeneous P2P network stack and provides interfaces for storage and compute resource provisioning, and the ordering machine

, which implements heterogeneous consensus (total ordering), heterogeneous mempools (partial ordering), and parallel transaction execution.

The networking machine

consists of a set of communicating sub-processes which are responsible for message-passing between nodes. Anoma's network architecture, based on P2P Overlay Domains with Sovereignty, assumes heterogeneous P2P preferences

: different nodes want to broadcast and subscribe to different types of intents, participate in different domains of consensus, storage, and compute, and choose different bandwidth, latency, and resource usage tradeoffs in their network connection choices. The networking machine is designed to abstract this complexity and the ever-changing physical and overlay network topologies behind an interface which simply allows sending a message to any other node or topic in the network - the internal processes within the networking machine are responsible for using network metadata to figure out where it should go and how to route it there efficiently. Generic interfaces are also provided for provisioning and requesting storage and compute.

The ordering machine

consists of a set of communicating sub-processes which are responsible for receiving transactions, partially ordering those transactions in the mempool, coming to consensus over a total order for each logical consensus clock, executing the transactions, updating the state accordingly, and sending updated state to the appropriate recipients. Both the mempool and consensus are heterogeneous - implementing Heterogeneous Narwhal and Heterogeneous Paxos, respectively - and the execution engine based on CalvinDB is capable of n-processor parallel scale-out.

Heterogeneous chains are operationally expensive if you have to run a separate node for each one - so Anoma's node architecture supports many networks with a single node

. The operator simply configures which networks they'd like to participate in and the node software takes care of all the bookkeeping. Processes within the ordering machine, for example, can be spun up or spun down automatically based on which consensi the operator is participating in and how much throughput each needs at the moment.

A full paper on the heterogeneous trust node architecture is still in the works, but the curious reader may be interested in more about multi-chain atomic commits or a brief introduction to Anoma's P2P layer.

Languages for explicit service commitments

Languages for explicit service commitments

provide a way for clients (requesting a service) and servers (providing a service) to automatically negotiate the terms of that service in a way which is legible to the network, so that records can be kept of services performed as promised - and promises broken. For example, service commitments could include

- "I will vote in consensus XYZ"
- "I will store your data D for T weeks"
- "I will spend N units of CPU time searching for an answer to problem P"

Generally, these commitments will be comprised of aspects classifiable into one of two categories: safety

- roughly, promising not to send any messages which violate a certain invariant, and liveness
- roughly, promising to send responses promptly when queried. Initially, I think three kinds of services are particularly important in the context of distributed systems: ordering

services, storage

services, and compute

services. I will expand a bit upon each of these in turn.

Ordering services

are perhaps the most basic historical function of blockchains. Satoshi actuallyuses the term "timestamping server". In order for multiple observers to agree on the history of a particular piece of state, they must agree on a particular party whose local order of observations will determine the order in which events (transactions) are applied to that state. This party must attest to the order in which it has received events, from which we can derive the desired liveness property: signing updates to the event graph (blocks of transactions). Consistency for other observers requires consistency of ordering attestations, from which we can derive the desired safety property: never signing two conflicting updates. Other properties of interest here could include censorship resistance - perhaps promising to construct blocks in a certain fashion (connecting directly to PEPC).

Storage services

are recently in the spotlight thanks to the concept of "data availability", which illuminates an important nuance: it is not

enough for observers that data be stored

, it must be available

- they must be able to retrieve it. In order to make data available, though, one must store it. I think storage services need only a liveness property: responding with the requested data, perhaps only within a certain span of time from the initial request. Thanks to content-addressing, the recipients of that data can quickly check that the correct bits were provided. Storage markets are very heterogeneous - different data needs to be retrieved by different parties, at different frequencies, and stored with different level of redundancy and distribution.

Compute services

are currently split across a few different areas: searching

performed in the Ethereum MEV supply chain, indexing

still typically performed by web2-style services, and historical statistics

calculated on the backend by explorers or specialized providers. I think that these various specific services are variants of one basic unit: the provision of computational infrastructure to search for a solution satisfying a particular relation, which may include network history. Safety for compute means providing the correct result, and liveness means getting a result quickly and with high likelihood.

We're still in the design process for these service commitment languages, and I see a lot of potential for collaboration. In particular, several folks in the Ethereum research community have been exploring relevant concepts: protocol-enforced proposer commitments, whereby Ethereum block proposers can make commitments which are enforced by the core protocol, and rainbow staking, where staking services are split into "heavy" and "light" categories (the names follow from expected node operator resource requirements). I expect that clear specification of various flavors of heavy and light services as described in the post, and negotiation of who is to perform those services, how they are to be compensated, and how other network actors can tell whether or not they are performing as promised will require some form of service commitment language, and PEPC would allow these commitments to be legible to and enforceable by the core protocol. Promise Theory may also provide helpful theoretical ammunition. I hope to develop a better understanding of the Ethereum ecosystem's requirements here and see whether it is possible to develop a common standard.

Example applications

What kinds of applications can be built with these affordances and mechanisms? I'll save a comprehensive survey for a separate discussion, but I want to list a few here which I'm particularly excited about.

Multichat

Multichat is a distributed chat network without any specially designated server operators. Like Slack or Discord, multichat supports permissioned channels in complex topologies. Like Signal, multichat encrypts messages and metadata. Unlike Signal, Slack, or Discord, multichat can run offline, or in an isolated physical subnetwork, using direct connections between chat participants. Unlike Slack or Discord, multichat cleanly separates protocol, operators, and interfaces, allowing different interfaces to be designed for different user needs while retaining protocol compatibility, and allowing users to choose which operators they want to trust with what roles.

For more details on this application concept, seethis forum thread.

Public Signal

Public Signal is a double-sided version of Kickstarter implemented with intents. Kickstarter is a supply-side driven platform: a group or individual wishing to produce a particular good or service and in need of funding posts a description of their product or service and organizes a campaign to raise small donations/payments in order to cover their capital expenditure requirements and labor costs. Public Signal, by contrast, is demand-side driven

: potential purchasers of a good or beneficiaries of a service describe their preferences and are gradually matched together, generating a demand signal which potential suppliers can use to decide what to produce. The demand-side approach is particularly appealing because it can generate an otherwise difficult-to-observe signal for public

or hybrid

goods.

For more details on this application concept, see thePublic Signal blog post.

Scale-Free Money

Scale-free money is a hypothetical monetary system design which allows anyone to create arbitrary denominations of

money at any time for any purpose and provides the necessary infrastructure to unbundle the three key functions of money (store of value, means of exchange, and unit of account) and allow for the choice of what money to use to be made differently by different network participants without compromising their ability to enact voluntary economic interchange. To me, scale-free money is the natural synthesis of the experimental practice of decentralized cryptocurrencies

, the anthropological record of heterogeneous credit

, and the cybernetic approach to coordination systems design

.

For more details on this application concept, seethis blog post.

Promise Graph

Promise Graph, inspired in part by the Collaborative Web model of Galois, and Informal Systems' Workflow, is a language and structured accounting logic for making and managing promises in order to effectively synchronize causally-interwoven workstreams across a distributed organization and demonstrate to customers that the organization can keep its promises. Heliax has been dogfooding the promise graph system internally for awhile now, but it is designed in particular to facilitate coordination irrespective of, and even without knowledge of, organizational boundaries. Similar to how Anoma splits the architecture of an intent-centric system from the particular topology users choose, Promise Graph splits the architecture of promise-based coordination from the particular topology of promises necessary to achieve a particular aim.

For more details on this application concept, seethis description of the system.

Topology

Now, let's talk topology. How does Anoma relate to the topology of the Ethereum network today? As a refresher, by topology

, I mean specific network connections, specific security relations, and specific service providers performing specific network roles. My understanding is that the Ethereum network is still figuring out what topology it wants to have, as evidenced by the fervent discussions around shared sequencing for rollups/L2s, EigenLayer, and rainbow staking. I think the existence of such discussions is evidence of a healthy network - a network with a static topology is a network which can no longer adapt to changes in the world outside.

There are infinitely many possible topologies, and I don't know of any finite framework to classify them - so here I will simply list a few topologies under discussion in the Ethereum ecosystem and explain at a high level how Anoma could be used with them. I will also discuss how Anoma could interface with specific specialized ordering, compute, and storage service providers which already exist, and how service commitments could be made using existing assets. Finally, I will detail two components which I believe sufficient to connect Anoma to any and all of these topologies.

Rollups and L2s

For several years now, the Ethereum ecosystem has been following the rollup-centric roadmap, where different sequencers order transactions for different rollup chains, which then post proofs and some data to Ethereum. Most rollups use the EVM, although a few have developed distinct VMs with different properties designed for particular kinds of applications. Recent proposals have brought forth the idea that many rollups may wish to share a sequencer, which is subsequently referred to as a "shared sequencer". The Ethereum main chain could in fact play this role of the shared sequencer.

Anoma could be useful for rollups in two ways:

- The resource machine could be used as a rollup execution environment instead of or in addition to the EVM, providing the rollup's users with intent-level and transaction-level information flow control, intent-level composability, and parallel transaction execution.
- The heterogeneous node architecture could be used to simply run the EVM, which would allow rollup operators to participate in a shared heterogeneous P2P network on which they could connect to other rollups, Ethereum main chain validators, searchers, and other intent infrastructure operators. Heterogeneous Paxos, in particular, may be interesting for rollups with frequent cross-traffic, because it generalizes shared sequencers.

Plasma

Plasma is the name for a family of Ethereum scaling solutions which require only posting deposit transactions, withdrawal transactions, and updated state Merkle roots to the main chain, while keeping all data and compute happening "within Plasma" off-chain. The basic concept assumes a block-producing operator who publishes state roots for each new block to the main chain and sends to individual users any state updates affecting them (e.g. tokens they've received). Should the operator misbehave, users can themselves publish proof of state they own to the main chain in order to withdraw it. In the original Plasma design, this forced withdrawal requires a 7-day challenge period, but the addition of validity proofs (where the operator proves that each new state root is the valid result of block execution) allows for withdrawals referring to the

latest state root to be processed instantly. For a comprehensive summary of Plasma research and recent work, I recommend Vitalik's blog post on the topic.

As discussed in that blog post, even with validity proofs, several challenges remain for Plasma:

- Dependency tracking

: The EVM does not attempt to limit or explicitly track state change dependencies, so (for example) ETH held in an account in block n + 1 could have come from anywhere block n. Exiting may then require publication of the entire history, which would incur prohibitive gas costs - and this worst-case exit game would likely entail limitations on the compute and storage bandwidth of Plasma, since it must be possible to replay everything on the main chain.

- Incentive-compatible generalization

: Plasma works well with state objects which have a clear economic owner, such as fungible or non-fungible tokens, but it's not clear how to adapt the incentive model to objects without a clear economic owner such as a CDP, which a user might want to pretend to have forgotten the data for if the price of ETH goes below the DAI they've withdrawn.

- Developer-facing complexity

: Plasma application developers must reason about state ownership graphs and data storage/publication incentives, adding mental overhead and introducing whole new classes of potential bugs.

What happens if we try to build a Plasma-like construction on top of the Resource Machine, instead of the EVM? Let's call it Resource Plasma

. In Resource Plasma, the operator executes transactions and publishes only the Merkle root of the commitment tree, the Merkle root of the nullifier set, and a proof of correct transaction execution to the main chain. Withdrawals function as follows:

- Withdrawals with a proof made against the latest nullifier set Merkle root can be processed immediately. They also reveal the nullifier of the resource being withdrawn (so that it can no longer be withdrawn again).
- Withdrawals against older Merkle roots require a delay period. When the withdrawal is requested, the Plasma contract starts a "double spend" challenge game, where anyone can submit a proof that the nullifier for the resource being withdrawn was included in a later nullifier set than the Merkle root used for the withdrawal request. After the challenge period, if no conflicting nullifier has been found, the withdrawal is processed.

Thanks to the dual commitment-nullifier system used by the resource machine, client data storage requirements are minimal - the client need only store their resources, proofs that the commitments corresponding to those resources were included in the commitment Merkle tree root, and proofs that the nullifiers corresponding to those resources were not revealed at whatever height the client last synced at. Clients wanting to withdraw instantly must update these latter proofs, but have the option to exit if the operator withholds data (in this case, the latest nullifier set). Resource Plasma thus removes the need for separate dependency tracking

- clients need not track dependencies, only the latest state relevant to them and proofs associated with it. Developer-facing complexity

is also reduced because this state architecture, including commitments and nullifiers, is built into the resource machine and requires no additional effort on the part of developers.

Incentive-compatible generalization

is a trickier beast. One option is to have each resource designate a particular highly-available party whose signature over an attestation that they have stored the resource data is required in order to construct a valid transaction. For example, in the CDP example, perhaps it would be suitable for a quorum of operators elected by MKR holders to store data associated with all CDPs. If the CDP is to be liquidated and the owner pretends not to have the data, this quorum could then reveal the data, allowing for the liquidation of the CDP. MKR holders have an incentive to store and publish this data in order to keep the system on which their value flows depend credible. In a version of MakerDAO without governance, it is less clear what to do in this scenario. Perhaps the data could also be sent to a party who might want to liquidate the CDP, or more generally one with the opposite economic incentive. Perhaps systems such as Maker really do need a designated party (which could be a dynamic, distributed operator set) whose job it is to store the data. The resource machine makes it easy to designate who should store what state, but it doesn't change the game design problem.

Resource Plasma is no panacea. Application developers must still reason about storage handoff, and users who want to be able to receive payments while offline must pick a highly available operator who they trust to temporarily store data on their behalf (or pay the main chain itself to do so). These limitations, however, are fundamental - they could not be alleviated by a different virtual machine - and other scaling solutions such as L2s or rollups will be faced with the same constraints. This is just a sketch of stirring the Plasma and Resource Machine models together, and seeing what emerges from the resulting conceptual melange. I think there may be some compelling options here, and I plan to explore this design space further in a separate post.

EigenLayer and service providers

Many blockchains today - including, typically, those who self-identify as data availability layers, in the modular conceptual framework - provide both ordering

and storage

services. Ordering services are mostly distinguished by the validator set and consensus mechanism, the combination of which determine latency, costs, and security. Storage services are often distinguished by how long the storage is to be provided for - blob storage on Ethereum, Celestia, and EigenDA, for example, is short-term, while file storage on Filecoin or Arweave is (at least nominally) permanent. Storage services are also distinguished by operators, redundancy, and optimization for specific patterns of retrieval. Many applications will likely want to use multiple storage providers with different specialties with different purposes. Compute

services today are mostly provided by off-chain actors, as it is not typically necessary to replicate compute (since the results can be verified). I understand Ethereum ecosystem searchers/builders, indexers, and statistical data providers to all be providing compute services of various specialized flavors.

EigenLayer provides a set of smart contracts and coordination infrastructure with which Ethereum validators can make commitments to operate additional services (AVSs), and "restake" their existing ETH staking bonds to provide security for those services (in the sense that these bonds can be slashed if the services are not performed as promised). The services performed by these validators could include ordering, storage, and compute as described here.

Anoma's service commitment languages can help both users and operators - the demand and supply sides of this service provisioning market - detail what services they are willing to provide, navigate through the services on offer, bargain for a mutually acceptable price, detect if service commitment properties have been violated, and publish evidence of defection in a standard way that can be embedded into network history and preserved as a reputation signal for future participants.

Topology implementation requirements

What would be required to support these varied possible topologies? Just two main ingredients, I think: an Anoma protocol adapter

and an Anoma node sidecar

. I will describe each of these in turn.

The Anoma protocol adapter

is a smart contract or set of smart contracts - primarily written for the EVM, but variants could be written for other VMs or execution environments - which emulate the resource machine and thus allow for execution of Anoma-formatted transactions, synchronization of Anoma-formatted state, and verification of state changes executed elsewhere on the Anoma network.

There may be some efficiency loss in execution emulation, but there shouldn't be too much: since the resource machine does not fix a particular instruction set or message-passing architecture, these transactions can simply use EVM bytecode to compute new resource values and the EVM's message-passing architecture to pass messages between different contracts required for the computation. Verification with succinct ZKPs is constant-cost anyways, so no problems there. Should the Anoma application model prove safe and popular, the EVM could easily enshrine it with a few precompiles for native performance.

Slight aside: solvers will get to have some fun in this model in a pro-public-good way: now they get to compete on optimizing JIT EVM compilers, the software created for which could probably aid other languages and systems trying to target the EVM.

The Anoma node sidecar

is a process which would run alongside and communicate bidirectionally with Ethereum execution & consensus clients, rollup sequencer processes, solver/searcher algorithms, etc. The node sidecar process runs all the Anoma network protocols, allowing nodes running the sidecar to connect to other Anoma nodes, subscribe to intents which are of interest to them, and solve, order, and execute as desired. This sidecar would be fully opt-in and configurable per the node operator's preferences in terms of who they want to connect to (or not), what services they want to offer, what intents they want to receive (or even send), and how they want to process or forward them. If the node operator solves two intents to create an EVM transaction with the protocol adapter, they can submit it directly to their local web3 HTTP API - minimum latency, maximum throughput!

What would this intent dataflow look like visually? Something like this:

[

Intent dataflow

1290×907 147 KB

](https://ethresear.ch/uploads/default/original/2X/3/3294ba703a672efe0a0f167477ede160368535de.jpeg)

Diagram credit @0xapriori

Challenges and trade-offs

Nothing new is without challenges or trade-offs. I don't want to paint with too rosy a brush here - plenty remains to be figured out, and plenty could go wrong. I see three main challenges (and I'm sure there are many which I miss):

- Implementation risk

: Anoma, although mathematically characterized at a high-level, is not yet fully implemented. We're working to reduce uncertainty and risk here by conducting and publishing more research, writing a formal model of the protocol stack in Isabelle/HOL, and developing an initial node implementation in Elixir (to be released soon) - but there could still be mistakes in our current understanding, and there are plenty of details we need to get right (which will always be the case with any new architecture)

- Hardware requirements

: compared to just running a vanilla Ethereum full node, running an Ethereum full node plus the Anoma sidecar will increase resource usage somewhat. Different node operators can pick and choose which kinds of extra messages they want to receive and process, which should help, but perhaps full node operators and solo stakers are already strained to the limit. An interesting option could be to enmesh the two protocols more closely together in order to support more points in the spectrum in between light nodes and full nodes (related to the rainbow staking concept).

- Developer education

: writing applications for Anoma is very different not only from writing applications for the EVM or other blockchains, but even from writing applications in imperative, computation-ordering-oriented languages at all - in a certain sense, it's more akin to a kind of declarative object-oriented programming, where you define which kinds of objects exist (resource kinds), how they can change (resource logics), and what actions users can take with them (application actions). Personally, I find this model much easier to work with after getting used to it - it provides a lot of expressive power - but it will take developers some time to learn.

Request for comment

Thank you for reading this far! These are my initial thoughts, and I'd love to know what you think. I have four specific questions - or respond with anything you like.

1. Of what I describe here, what makes sense to you? What doesn't make sense?
2. What challenges, risks, or trade-offs do you see that I didn't cover?
3. What do you think would be most valuable to the Ethereum community for us to focus on?
4. What would you - or a team or project you know - like to collaborate on? Who should we talk to?

Cheers!