This is the loose transcript of a talk by Vitalik Buterin at mev.market, (London 2024). This should provide some insight about the key architectural decisions Ethereum is considering along with whom–which teams will build them.

Introduction

Almost three years ago Vitalik wrote the notable endgame post. In the post he pointed out that various blockchain scaling paths that people were thinking about including Layer 1 scaling and different forms of layer 2 scaling. The post conjectured that if you push all of those paths to their limit and you add the necessary scaffolding to make them decentralized and censorship resistant, they actually end up in a surprisingly similar place. In this place, you get centralized block production, decentralized validation, and strong anti-censorship protection. Since the post was published the ecosystem has moved from theory to practice.

Industrial Organization Problem

How Ethereum scales is an industrial organization problem as much as it is a technical one. Even if the same pieces get built, different choices between who builds the pieces can lead to big differences in outcome.

Analogy: the efficient way to serve food at scale across a country is to have many locations, in each location food gets made in a central kitchen and people can order it. Is each location a branch or a franchise? A branch is like humans in control arranged from the top down. Franchise is like anyone can come in and say, “I have a location, I’ll pay your fee, and I want to serve your food”.

This is a sub-discipline in economics. Try to think about the trade-offs in these terms. Even if the same kinds of pieces get made what are the consequences of different pieces getting made by different parties. Let’s look at a few Ethereum specific examples.

State transition function

If ethereum scales you need to have lots of transactions and these transactions are being processed by a set of rules. These rules talk about how accounts work, how smart contracts work, and so forth. These rules need to be enforced somewhere. Ethereum for the last few years has had a rollup-centric roadmap where ethereum as a blockchain provides shared security and data availability through things like EIP 4844. You have rollups which are layer 2 protocols which handle execution. Theoretically you can achieve the exact same outcome by basically having a big system that has monolithic shards inside the protocol. In the case of Ethereum, there exist three possible futures:

- Each rollup makes their highly customized STF - this is a major basis for people to compete with each other.
- pros: more options for users, faster iterations for VM technology,
- cons: greater software bug risk and more confusing for users
- pros: more options for users, faster iterations for VM technology,
- cons: greater software bug risk and more confusing for users
- Rollups generally reuse the EVM
- pros: medium bug risk (some risk of bugs still), easy to understand for users
- cons: Less room for creativity
- pros: medium bug risk (some risk of bugs still), easy to understand for users
- cons: Less room for creativity
- Rollups use a hypothetical Ethereum L1 EVM precompile
- pros: minimal bug risk, strongest guarantees for users, no need for governance
- rollups upgrade when ethereum upgrades
- the more you enshrine, the less application attack surface there is
- rollups upgrade when ethereum upgrades
- the more you enshrine, the less application attack surface there is
- cons: Even less room for creativity, Enshrined ZK-EVM may involve technical sacrifices
- pros: minimal bug risk, strongest guarantees for users, no need for governance

- rollups upgrade when ethereum upgrades
- the more you enshrine, the less application attack surface there is
- rollups upgrade when ethereum upgrades
- the more you enshrine, the less application attack surface there is
- cons: Even less room for creativity, Enshrined ZK-EVM may involve technical sacrifices

In the enshrined ZK-EVMs post Vitalik wrote, he talks about this idea.

Perhaps Ethereum should include a ZK-EVM precompile as part of its Layer 1 functionality. If we do that then it comes much closer to a world that was called sharding ten years ago, execution sharding. As a rollup you are responsible for spinning up your own shard. If you do a good job of attracting applications to your shard, you get to collect the priority fees for your shard and make money that way. From a technological perspective there are a lot of similarities but from a perspective of who builds it you have a lot of different consequences.

These different options have some interestingly different consequences. A lot of these consequences don't appear on the usual axis like scalability vs. decentralization. They appear on these weird new axis that we don't often think about speed of innovation of different things vs. risk of software bugs vs. the extent to which different pieces of the system need governance.

For example if you are the sort of person who loves the idea of Ethereum ossifying and you are totally afraid of Ethereum governance doing crazy things in 2029 then options 1 and 2 are better for you. But if your the sort of person who believes off-chain governance is best and you are afraid of weird gadgets inside of layer 2s then option 3 is best.

Sequencing

Sequencing is the procedure for determining which txs go in the next rollup block.

Between independent and shared sequencing you get the risk that shared sequencing happens anyway through side channels. If you do shared sequencing there is greater risk of some ecosystem wide monopolization. If you do it at layer 1 then you have a different risk of things like MEV spilling over to Layer 1.

- Independent Sequencing - every rollup does it for itself
- pros: minimal code dependencies for each rollup, maximum flexibility
- cons: rollups have to build more tech, risk that shared sequencing happens through side channels
- pros: minimal code dependencies for each rollup, maximum flexibility
- cons: rollups have to build more tech, risk that shared sequencing happens through side channels
- Shared Sequencing (as L2 infrastructure)
- pros: more cross-rollup atomic tx options, less side channel risk, protects L1
- cons: greater risk of ecosystem-wide monopolization, shared sequencing protocol as rent extractor
- pros: more cross-rollup atomic tx options, less side channel risk, protects L1
- cons: greater risk of ecosystem-wide monopolization, shared sequencing protocol as rent extractor
- Based sequencing (involves L1) - and additional ideas that try to bring Layer 1 and Layer 2 together
- pros: few code dependencies, more cross-rollup atomic tx options
- cons: who has incentive to build it?, more risk spills over to L1, Greater risk of ecosystem wide …
- pros: few code dependencies, more cross-rollup atomic tx options
- cons: who has incentive to build it?, more risk spills over to L1, Greater risk of ecosystem wide …

In Vitalik's post praising Bitcoin maximalism, he talked about how the thing about bitcoin simplicity that makes sense is not the technical simplicity of the VM. Having applications is bad. The reason having applications is bad is they create MEV and the centralizing effects of MEV spill over to the base layer. If we want Ethereum itself to be maximally protected from this then you have sequencing be very far away so w/e centralization or weirdness happens that gets collected by outside actors and w/e gets pushed onto L1 becomes relatively time independent.

Which of these protocols gets to collect the MEV? This is one of the tradeoffs b/w sequencing being based and sequencing

happening at some higher level. From a technological perspective, you get similar things happening, but a lot of the consequences are economic, they're about code dependency, whether you are leaning on L1 or doing things separately from L1 and these kind of issues.

Proof Aggregation

Proof Aggregation - How does the L1 learn that state roots from multiple L2s have been computed correctly?

Rollups need to publish proofs to the Ethereum. A proof is 500k gas or Millions of gas for STARKs. It would be great if every rollup could publish a proof every block. However at this point you have more than 30 rollups and too many for the gas limit if we want every rollup to publish a proof for every block.

- Independent proof publishing (status quo)
- pros: simplest code, minimal trust dependencies
- cons: high gas costs
- pros: simplest code, minimal trust dependencies
- cons: high gas costs
- Proof Aggregation protocols ("L1.5")
- pros: Near-ideal gas savings (min + 20k per rollup), Medium shared trusted code risk
- cons: opinionated choices (e.g. proof system), who has incentive to build it?
- pros: Near-ideal gas savings (min + 20k per rollup), Medium shared trusted code risk
- cons: opinionated choices (e.g. proof system), who has incentive to build it?
- Aggregation within ecosystems ("L3s")
- pros: already happening: it's clear who builds it, near-ideal gas savings (min + 500k per ecosystem)
- cons: Benefits large rollup ecosystems: harder for small independent rollups, Largest shared trusted code
- pros: already happening: it's clear who builds it, near-ideal gas savings (min + 500k per ecosystem)
- cons: Benefits large rollup ecosystems: harder for small independent rollups, Largest shared trusted code

The status quo is every rollup figures this out for themselves. But what if we had aggregation protocols, instead of 20 rollups publishing 20 snarks to the chain, you somehow publish 1 snark to the chain.

One way is that you could have Ethereum ecosystem wide proof aggregation - you have a protocol where rollups can opt into, and they submit proofs into a mempool and then it's the role of builders to take these proofs and make a proof of the proofs and publish this proof of the proofs with the state roots and then the on-chain verifier contract makes 1 call for each rollup, and you only need one proof. Here you get good gas savings, some level of opinionated choices, some amount of shared and trusted code.

Today we are seeing aggregation within ecosystems. Starknet has its own aggregation. It has layers 3s then the layer 3s commit into layer 2 and layer 2s commit to layer 1. So ultimately, there is one proof. So you get the same effect of proofs inside of proofs. If this is a thing individual rollup ecosystems do separately, this is good for large ecosystems but not small ecosystems, and you have to deal with more trusted code.

Account Abstraction / Key Stores

AA/Key-stores - If the authentication / authorization logic for an account can change, where does the mutable data live?

We have wallets. One basic security property which is considered standard in regular cybersecurity is you want to be able to expire keys. You want keys that controlled an account before to no longer control the account later. This is something that EOAs do not do and one of the 5 fundamental reasons EOAs must die. If you have some AA-based setup then basically you have to store the information somewhere like which keys currently have the right to process transactions from an account. The question is–where does this data get stored?

- Separately on each L2 (status quo)
- pros: simplest code, minimal trust dependencies
- cons: high gas cost per key change, risk a key change will "miss an L2"

- pros: simplest code, minimal trust dependencies
- cons: high gas cost per key change, risk a key change will “miss an L2”
- “Minimal key-store rollup”
- pros: medium trust dependencies, low gas cost
- cons: requires more standardization (e.g. L2s reading L1s), requires more cryptography, who builds it?
- pros: medium trust dependencies, low gas cost
- cons: requires more standardization (e.g. L2s reading L1s), requires more cryptography, who builds it?
- Each user chooses a “home rollup” puts their key-store there
- pros: maximum trust dependencies
- cons: requires more standardization (e.g. L2s reading L1), Requires more cryptography, N^2 tech blowup (N state proof types in N rollups)
- pros: maximum trust dependencies
- cons: requires more standardization (e.g. L2s reading L1), Requires more cryptography, N^2 tech blowup (N state proof types in N rollups)

Status Quo: If you have a smart contract wallet, you have a copy of that smart contract wallet on L2. This is fairly simple code with minimal trust dependencies, but there are obvious costs. If you trade the keys in one place, how do you change the keys in other places? You have high gas costs of changing the keys around in different places, and there is a risk you will just forget to change the keys somewhere. Let’s say you have a Gnosis safe, and you change your keys in 3 different places, and you forgot you have a copy of the safe somewhere else, then you forget to change the keys. Then 2 years later, one at a time, the keyholders get hacked, or someone goes after them.

A natural approach is that instead of having that data live in every place, you can have the data live in one place. Either use L1 which maybe a bit too expensive, use some L2 to store this information on which keys currently have the right to access some account. One idea is this minimal key-store rollup. You create this minimalistic rollup whose only function is to store this account related logic and the rules for updating it. Trying to keep the rollup simple and trust dependencies low, requires some standardization. If you want to update the account you only need to do it in one place, then basically every time you sign your transaction, that transaction would include a SNARK which proves the current status of your account. This is going to become cheap in the future with proof aggregation, and then there is still the question of who builds it?

Another option is to let users choose. Users can base their accounts on whatever chain they want. You put their key store there. You store the logic of who can access your account on whatever chain you want. Different users make different choices. There is tech blowup risk here - an n^2

blowup because you have n

different proofs for n

different rollups. You have more trust dependencies, but on the other hand it requires less shared infrastructure, and it is this thing that maybe happens anyway by default. Again, here you have approaches which are very similar but differences in practice, depending on who builds it.

Conclusion

Once you start getting into the weeds, there are many choices about which actors are responsible for which pieces. If you go back to the endgame post, from a technological perspective, the different scaling solutions feel like they are lining up toward the same conclusion. But what is the big difference between what I call an ideal L2 scaling future and an ideal Solana where you actually have consensus nodes being able to run on laptops, and have censorship inclusion channels, and have STARKs for everything? The answer is basically you have a boundary of which actors are responsible for building and maintaining which parts of the ecosystem and what their incentives are. Across all of these different ares, the difference between these things matters a lot.

You have two separate questions:

- the tech arrangement
- who is responsible for building and maintaining what

The biggest actors we have are independent layer 2 teams. Another is standards groups - RIPs, EIPs, ERCs, and potentially more standardization. You also have L1 core protocol teams. Depending on which of these pieces is responsible for building

the different things you start to have some different consequences. The main axis on which the consequences are different on all of these is

- who has the incentive to build the thing?
- is there incentive for people to try to get some monopoly and start extracting rent?
- where does the code bug risk lie, and how much code bug risk is there?
- other questions…?

A lot of these questions we have in the future are similar. Even if the outcome is the same, the path for getting there defines these boundaries of what kind of actor is responsible for what. These choices will decide a lot of these details.