What kind of layer 3s make sense?

2022 Sep 17 See all posts

Special thanks to Georgios Konstantopoulos, Karl Floersch and the Starkware team for feedback and review.

One topic that often re-emerges in layer-2 scaling discussions is the concept of "layer 3s". If we can build a layer 2 protocol that anchors into layer 1 for security and adds scalability on top, then surely we can scale even more by building a *layer 3* protocol that anchors into layer 2 for security and adds *even more scalability* on top of that?

A simple version of this idea goes: if you have a scheme that can give you quadratic scaling, can you stack the scheme on top of itself and get exponential scaling? Ideas like this include my 2015 scalability paper, the multi-layer scaling ideas in the Plasma paper, and many more. Unfortunately, such simple conceptions of layer 3s rarely quite work out that easily. There's always something in the design that's just not stackable, and can only give you a scalability boost once - limits to data availability, reliance on L1 bandwidth for emergency withdrawals, or many other issues.

Newer ideas around layer 3s, such as the framework proposed by Starkware, are more sophisticated: they aren't just stacking the same thing on top of itself, they're assigning the second layer and the third layer different purposes. Some form of this approach may well be a good idea - if it's done in the right way. This post will get into some of the details of what might and might not make sense to do in a triple-layered architecture.

Why you can't just keep scaling by stacking rollups on top of rollups

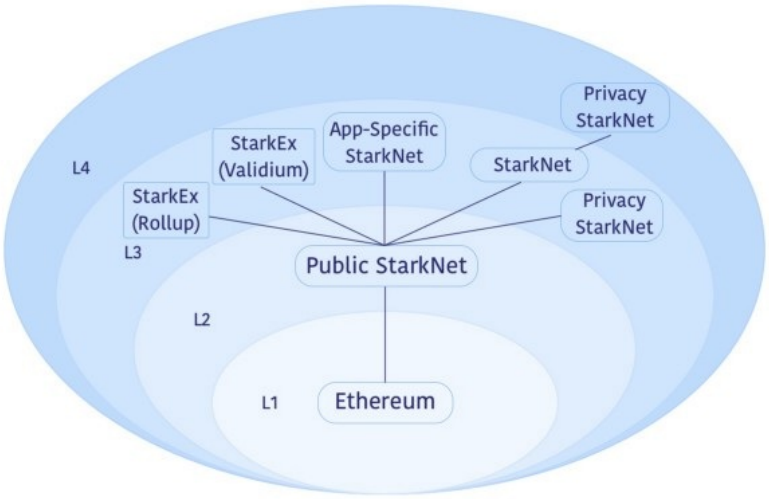
Rollups (see my longer article on them here) are a scaling technology that combines different techniques to address the two main scaling bottlenecks of running a blockchain: *computation* and *data*. Computation is addressed by either fraud proofs or SNARKs, which rely on a very small number of actors to process and verify each block, requiring everyone else to perform only a tiny amount of computation to check that the proving process was done correctly. These schemes, especially SNARKs, can scale almost without limit; you really can just keep making "a SNARK of many SNARKs" to scale *even more* computation down to a single proof.

Data is different. Rollups use a collection of compression tricks to reduce the amount of data that a transaction needs to store on-chain: a simple currency transfer decreases from ~100 to ~16 bytes, an ERC20 transfer in an EVM-compatible chain from ~180 to ~23 bytes, and a privacy-preserving ZK-SNARK transaction could be compressed from ~600 to ~80 bytes. About 8x compression in all cases. But rollups still need to make data available on-chain in a medium that users are guaranteed to be able to access and verify, so that users can independently compute the state of the rollup and join as provers if existing provers go offline. Data can be compressed once, but it cannot be compressed again - if it can, then there's generally a way to put the logic of the second compressor into the first, and get the same benefit by compressing once. Hence, "rollups on top of rollups" are not something that can actually provide large gains in scalability - though, as we will see below, such a pattern can serve other purposes.

So what's the "sane" version of layer 3s?

Well, let's look at what Starkware, in their post on layer 3s, advocates. Starkware is made up of very smart cryptographers who are actually sane, and so if they are advocating for layer 3s, their version will be much more sophisticated than "if rollups compress data 8x, then *obviously* rollups on top of rollups will compress data 64x".

Here's a diagram from Starkware's post:



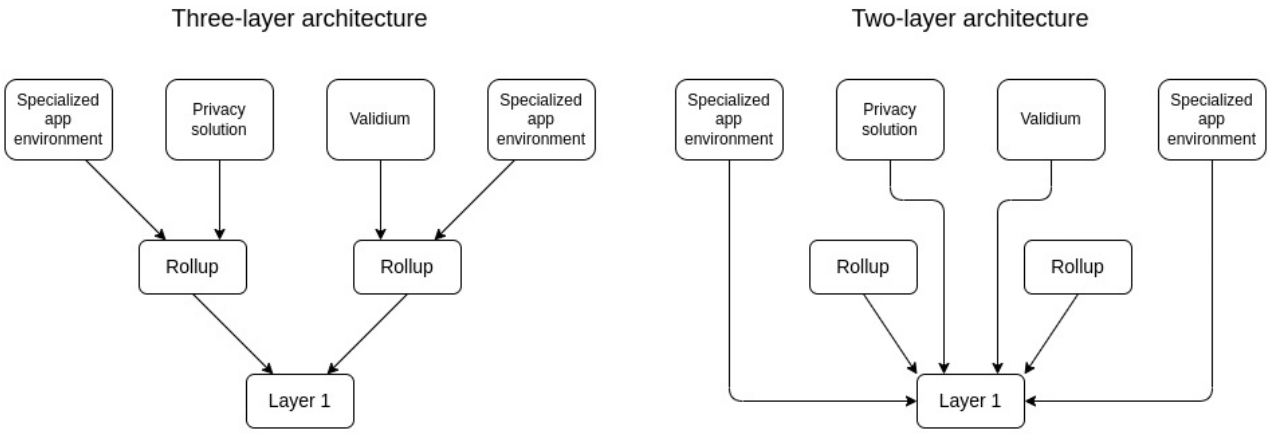
A few quotes:

> An example of such an ecosystem is depicted in Diagram 1. Its L3s include:
>
> - A StarkNet with Validium data availability, e.g., for general use by applications with extreme sensitivity to pricing.
> - App-specific StarkNet systems customized for better application performance, e.g., by employing designated storage structures or data availability compression.
> - StarkEx systems (such as those serving dYdX, Sorare, Immutable, and DeversiFi) with Validium or Rollup data availability, immediately bringing battle-tested scalability benefits to StarkNet.
> - Privacy StarkNet instances (in this example also as an L4) to allow privacy-preserving transactions without including them in public StarkNets.

We can compress the article down into three visions of what "L3s" are for:

1. **L2 is for scaling, L3 is for customized functionality, for example privacy.** In this vision there is no attempt to provide "scalability squared"; rather, there is one layer of the stack that helps applications scale, and then separate layers for customized functionality needs of different use cases.
2. **L2 is for general-purpose scaling, L3 is for customized scaling**. Customized scaling might come in different forms: specialized applications that use something other than the EVM to do their computation, rollups whose data compression is optimized around data formats for specific applications (including separating "data" from "proofs" and replacing proofs with a single SNARK per block entirely), etc.
3. **L2 is for trustless scaling (rollups), L3 is for weakly-trusted scaling (validiums)**. Validiums are systems that use SNARKs to verify computation, but leave data availability up to a trusted third party or committee. Validiums are in my view highly underrated: in particular, many "enterprise blockchain" applications may well actually be best served by a centralized server that runs a validium prover and regularly commits hashes to chain. Validiums have a lower grade of security than rollups, but can be vastly cheaper.

All three of these visions are, in my view, fundamentally reasonable. The idea that specialized data compression requires its own platform is probably the weakest of the claims - it's quite easy to design a layer 2 with a general-purpose base-layer compression scheme that users can automatically extend with application-specific sub-compressors - but otherwise the use cases are all sound. But this still leaves open one large question: *is a three-layer structure the right way to accomplish these goals?* What's the point of validiums, and privacy systems, and customized environments, anchoring into layer 2 instead of just anchoring into layer 1? The answer to this question turns out to be quite complicated.



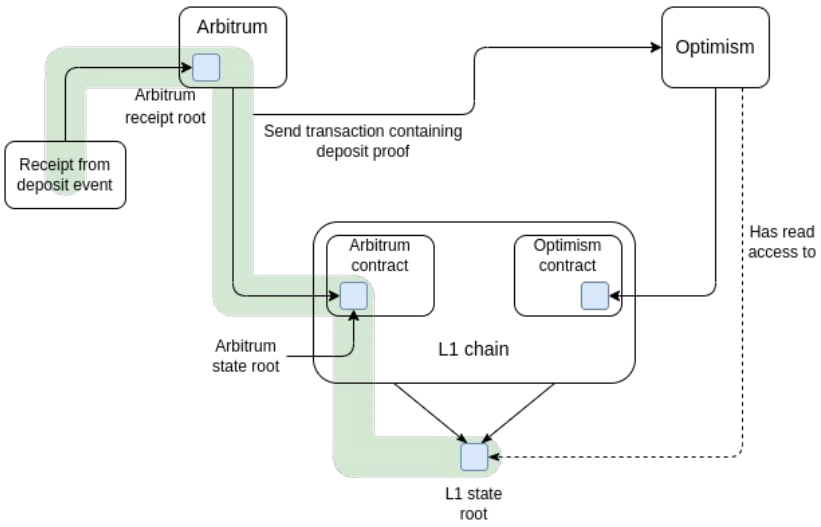
Which one is actually better?

Does depositing and withdrawing become cheaper and easier within a layer 2's sub-tree?

One possible argument for the three-layer model over the two-layer model is: a three-layer model allows an entire sub-ecosystem to exist within a single rollup, which allows cross-domain operations within that ecosystem to happen very cheaply, without needing to go through the expensive layer 1.

But as it turns out, you can do deposits and withdrawals cheaply even between two layer 2s (or even layer 3s) that commit to the same layer 1! The key realization is that **tokens and other assets do not have to be issued in the root chain**. That is, you can have an ERC20 token on Arbitrum, create a wrapper of it on Optimism, and move back and forth between the two without any L1 transactions!

Let us examine how such a system works. There are two smart contracts: the *base contract* on Arbitrum, and the *wrapper token contract* on Optimism. To move from Arbitrum to Optimism, you would send your tokens to the base contract, which would generate a receipt. Once Arbitrum finalizes, you can take a Merkle proof of that receipt, rooted in L1 state, and send it into the wrapper token contract on Optimism, which verifies it and issues you a wrapper token. To move tokens back, you do the same thing in reverse.



Even though the Merkle path needed to prove the deposit on Arbitrum goes through the L1 state, Optimism only needs to read the L1 state root to process the deposit - no L1 transactions required. Note that because data on rollups is the scarcest resource, a practical implementation of such a scheme would use a SNARK or a KZG proof, rather than a Merkle proof directly, to save space.

Such a scheme has one key weakness compared to tokens rooted on L1, at least on optimistic rollups: *depositing also requires waiting the fraud proof window*. If a token is rooted on L1, withdrawing from Arbitrum or Optimism back to L1 takes a week delay, but depositing is instant. In this scheme, however, both depositing and withdrawing take a week delay. That said, it's not clear that a three-layer architecture on optimistic rollups is better: there's a lot

of technical complexity in ensuring that a fraud proof game happening inside a system that itself runs on a fraud proof game is safe.

Fortunately, neither of these issues will be a problem on ZK rollups. ZK rollups do not require a week-long waiting window for security reasons, but they do still require a shorter window (perhaps 12 hours with first-generation technology) for two other reasons. First, particularly the more complex general-purpose ZK-EVM rollups need a longer amount of time to cover non-parallelizable compute time of proving a block. Second, there is the economic consideration of needing to submit proofs rarely to minimize the fixed costs associated with proof transactions. Next-gen ZK-EVM technology, including specialized hardware, will solve the first problem, and better-architected batch verification can solve the second problem. And it's precisely the issue of optimizing and batching proof submission that we will get into next.

Rollups and validiums have a confirmation time vs fixed cost tradeoff. Layer 3s can help fix this. But what else can?

The cost of a rollup *per transaction* is cheap: it's just 16-60 bytes of data, depending on the application. But rollups also have to pay a high *fixed cost* every time they submit a batch of transactions to chain: 21000 L1 gas per batch for optimistic rollups, and more than 400,000 gas for ZK rollups (millions of gas if you want something quantum-safe that only uses STARKs).

Of course, rollups could simply choose to wait until there's 10 million gas worth of L2 transactions to submit a batch, but this would give them very long batch intervals, forcing users to wait much longer until they get a high-security confirmation. Hence, they have a tradeoff: long batch intervals and optimum costs, or shorter batch intervals and greatly increased costs.

To give us some concrete numbers, let us consider a ZK rollup that has 600,000 gas per-batch costs and processes fully optimized ERC20 transfers (23 bytes), which cost 368 gas per transaction. Suppose that this rollup is in early to mid stages of adoption, and is averaging 5 TPS. We can compute gas per transaction vs batch intervals:

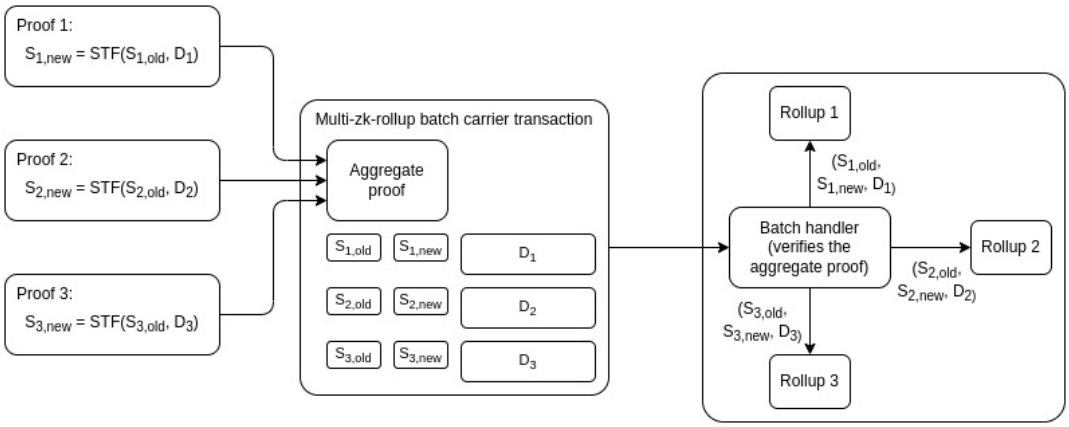
Batch interval	Gas per tx (= tx cost + batch cost / (TPS * batch interval))
12s (one per Ethereum block)	10368
1 min	2368
10 min	568
1 h	401

If we're entering a world with lots of customized validiums and application-specific environments, then many of them will do much less than 5 TPS. Hence, tradeoffs between confirmation time and cost start to become a very big deal. And indeed, the "layer 3" paradigm does solve this! A ZK rollup inside a ZK rollup, even implemented naively, would have fixed costs of only ~8,000 layer-1 gas (500 bytes for the proof). This changes the table above to:

Batch interval	Gas per tx (= tx cost + batch cost / (TPS * batch interval))
12s (one per Ethereum block)	501
1 min	394
10 min	370
1 h	368

Problem basically solved. So are layer 3s good? Maybe. But it's worth noticing that there is a different approach to solving this problem, inspired by ERC 4337 aggregate verification.

The strategy is as follows. Today, each ZK rollup or validium accepts a state root if it receives a proof proving that \(S_{new} = STF(S_{old}, D)\): the new state root must be the result of correctly processing the transaction data or state deltas on top of the old state root. In this new scheme, the ZK rollup would accept a message from a *batch verifier contract* that says that it has verified a proof of a batch of statements, where each of those statements is of the form \(S_{new} = STF(S_{old}, D)\). This batch proof could be constructed via a recursive SNARK scheme or Halo aggregation.



This would be an open protocol: any ZK-rollup could join, and any batch prover could aggregate proofs from any compatible ZK-rollup, and would get compensated by the aggregator with a transaction fee. The batch handler contract would verify the proof once, and then pass off a message to each rollup with the \((S_{old}, S_{new}, D)\) triple for that rollup; the fact that the triple came from the batch handler contract would be evidence that the transition is valid.

The cost per rollup in this scheme could be close to 8000 if it's well-optimized: 5000 for a state write adding the new update, 1280 for the old and new root, and an extra 1720 for miscellaneous data juggling. Hence, it would give us the same savings. Starkware actually has something like this already, called SHARP, though it is not (yet) a permissionless open protocol.

One response to this style of approach might be: *but isn't this actually just another layer 3 scheme?* Instead of `base layer <- rollup <- validium`, you have `base layer <- batch mechanism <- rollup or validium`. From some philosophical architectural standpoint, this may be true. But there is an important difference: instead of the middle layer being a complicated full EVM system, the middle layer is a simplified and highly specialized object, and so it is more likely to be secure, it is more likely to be built at all without needing yet another specialized token, and it is more likely to be governance-minimized and not change over time.

Conclusion: what even is a "layer"?

A three-layer scaling architecture that consists of stacking *the same* scaling scheme on top of itself generally does not work well. Rollups on top of rollups, where the two layers of rollups use the same technology, certainly do not. A three-layer architecture where the second layer and third layer have *different* purposes, however, can work. Validiums on top of rollups do make sense, even if they're not certain to be the long-term best way of doing things.

Once we start getting into the details of *what kind* of architecture makes sense, however, we get into the philosophical question: what is a "layer" and what is not? The `base layer <- batch mechanism <- rollup or validium` pattern does the same job as a `base layer <- rollup <- rollup or validium` pattern. But in terms of *how it works*, a proof aggregation layer looks more like ERC-4337 than like a rollup. Typically, we don't refer to ERC-4337 as a "layer 2". Similarly, we don't refer to Tornado Cash as a "layer 2" - and so if we were to be consistent, we would not refer to a privacy-focused sub-system that lives on top of a layer 2 as a layer 3. So there is an unresolved semantics debate of what deserves the title of "layer" in the first place.

There are many possible schools of thought on this. My personal preference would be to keep the term "layer 2" restricted to things with the following properties:

- Their purpose is to increase scalability
- They follow the "blockchain within a blockchain" pattern: they have their own mechanism for processing transactions and their own internal state
- They inherit the full security of the Ethereum chain

So, optimistic rollups and ZK rollups are layer 2s, but validiums, proof aggregation schemes, ERC 4337, on-chain privacy systems and Solidity are something else. It may make sense to call some of them layer 3s, but probably not all of them; in any case, it seems premature to settle definitions while the architecture of the multi-rollup ecosystem is far from set in stone and most of the discussion is happening only in theory.

That said, the language debate is less important than the technical question of which constructions actually make the most sense. There is clearly an important role to be played by "layers" of some kind that serve non-scaling needs like privacy, and there is clearly is an important function of proof aggregation that needs to be filled *somehow*, and preferably by an open protocol. But at the same time, there are good technical reasons to make the intermediary layers that connect user-facing environments to the layer 1 as simple as possible; the "glue layer" being an EVM rollup is probably not the right approach in many cases. I suspect more sophisticated (and simpler) constructions such as those described in this post will start to have a bigger role to play as the layer 2 scaling ecosystem matures.