There are a number of situations where proposed layer-1 scalability improvements (ie. modifications to the blockchain protocol, or to how clients work) to increase scalability, and proposed layer-2 scalability improvements (using the term broadly; basically anything that is an application-layer design pattern to increase scalability) are in effect doing exactly the same thing. This post will go through some examples and intuitions for how to think about these cases.

Stateless clients

See The Stateless Client Concept for background reading on stateless clients. To summarize, stateless clients work by having full nodes only store root hashes of the state, and use Merkle branches that are sent along with the block to prove that reads/writes from/to the state have been carried out correctly.

But stateless clients could be implemented either as a rework of the blockchain (eg.https://medium.com/@akhounov/data-from-the-ethereum-stateless-prototype-8c69479c8abc), or as a change to a specific contract, writing in code the rule that the contract only has a single hash as its state, and any changes need to come with Merkle proofs.

Notice that in both cases, the behavior used to improve scalability (substituting clients storing data with clients downloading and processing extra Merkle branches) is exactly the same, but it is implemented differently, once as a change to blockchain full node behavior and once as an optional application-layer change.

Fraud proofs

Optimistic rollup works by having a system store a series of historical state roots, and not "finalize" the state until some time (eg. 1 week) after a new state has been added. When a new "package" of transactions is submitted to a rollup contract, the transactions are not verified on-chain (though the availability

of the transactions is implicitly verified as the transactions must be submitted as part of the transaction); instead, the state roots are simply added to the list. However, if an outside observer sees that some package is invalid (ie. it claims a state root that does not match the state that would be generated by honestly executing the block on top of the previous state), they can submit a challenge, and only then the package is executed on chain; if the package turns out to be invalid then it and all successor states are reverted.

Fraud proofs work by having clients not verify the state (though they would still need to download all blocks to verify availability) by default, and instead accepting blocks by default and only rejecting them after the fact if they receive a message from the network containing a Merkle proofs that shows that one particular block was invalid.

Here once again, the same mechanism (download but don't check stuff by default, check only if someone sends an alarm) can be used either inside of a layer-2 scheme or as a client efficiency improvement at layer 1. However, note one important point: for layer-1 fraud proofs to have the same properties as rollup, consensus on data and consensus on state need to be separate processes. Otherwise, a node making a block would need to personally verify all recent blocks (as there might not have been enough time for a fraud proof come through) before publishing their own block, which would limit the scalability gains.

Signature aggregation

Techniques like BLS signature aggregation allow many signatures to be compressed into one, saving heavily on data and somewhat on computation costs (the greater savings on data vs computation make them extremely powerful if paired with fraud proofs). These techniques could be used on-chain, grouping together all transactions in a block into one. They could also be used at application layer, by using a transaction packaging mechanism where many transactions are submitted inside of one package, a single signature checker verifies the signature against the hashes of all the transactions and the pubkeys of their declared senders, and then from then on transactions are executed separately.

SNARK / STARKs

SNARKs and STARKs can be used to generally substitute the need for clients to re-execute a long computation with verification of a simple proof. This can once again be done at layer 1 (eg. see Coda), or at layer 2 (eg. ZK Rollup).

Implementing at layer 1 vs implementing at layer 2

Implementing at layer 1 has the strengths that:

- It "preserves legibility" of the chain because default infrastructure would be able to understand the scalability solutions and interpret what is going on (though standardization of common layer-2 techniques could provide this to some extent)
- It reduces the risk of fragmentation of layer-2 solutions
- It allows the network to organize infrastructure around the solution, eg. automatically updating proofs in response to new blocks, DoS resistance for transactions, etc.

- In cases where tradeoffs exist, it creates more freedom of choice for nodes

to choose their tradeoffs. For example, some clients could store all state and minimize bandwidth whereas other clients could verify blocks statelessly and accept the bandwidth hit of doing so (VPSes vs home computers may choose different options here). Alternatively, some clients could use fraud-proof-based verification to save costs, whereas others would verify everything to maximize their security levels.

Implementing at layer 2 has the strengths that:

- It preserves room for innovation of new schemes over time, without having to hard-fork the existing blockchain
- It minimizes consensus-layer complexity, especially if multiple schemes are needed for different use cases
- It allows users to benefit from applications with stronger assumptions (eg. trusted setup) without making consensus security dependent on them (though sometimes this can be accomplished at layer 1 as well)
- In cases where tradeoffs exist, it creates more freedom of choice for applications

to choose their tradeoffs. Some applications could live on-chain whereas others could live inside a rollup

Other key notes

Scalability gains from layer 1 and layer 2 that rely on the same underlying behavior often do not combine

. For example, scalability gains from using fraud proofs and scalability gains from using rollups don't stack on top of each other, because they're ultimately implementing the same mechanism, and so if using rollups to get 10000 tx/sec on a base layer that gets up to 1000 tx/sec using fraud proofs were safe, just using fraud proofs to get 10000 tx/sec on the same base layer would be safe too.

Doing the same thing at layer 1 and layer 2 leads to unneeded infrastructure bloat, so often choosing one over the other makes the most sense. For example, if stateless contracts at layer 2 are going to be used regardless, then one might as well make layer-1 state extremely expensive to effectively mandate such layer-2 schemes, so as to keep state small to prevent the possibility that stateless clients will ever be also needed to be built at layer 1.

Also, note that data availability

is the unique thing that layer 1 can solve but layer 2 can only provide with strong relaxation of security assumptions (eg. assuming honest majority of another set of actors). This is because data availability proofs, or alternative systems where individual blocks can be reconstructed by erasure-coding from other blocks, crucially rely on client-side randomness that differs from each client, which cannot be replicated on-chain.

Conclusions

The desire to do ongoing innovation at layer 2 is a really big argument that is driving my own preference for a layer-2-heavy design for eth2, where features provided at layer 1 are minimized (eg. the state is kept small so stateless clients will be unnecessary but L2 stateless contracts will be effectively mandatory). However, there are some things that we may want to provide an explicit facility for at layer 1. The biggest thing that must be layer 1 in a general-purpose scalable blockchain is data availability, for the reasons described above, which is a big part of why eth2 is done at all instead of building a layer-2-heavy roadmap for the existing eth1 chain.