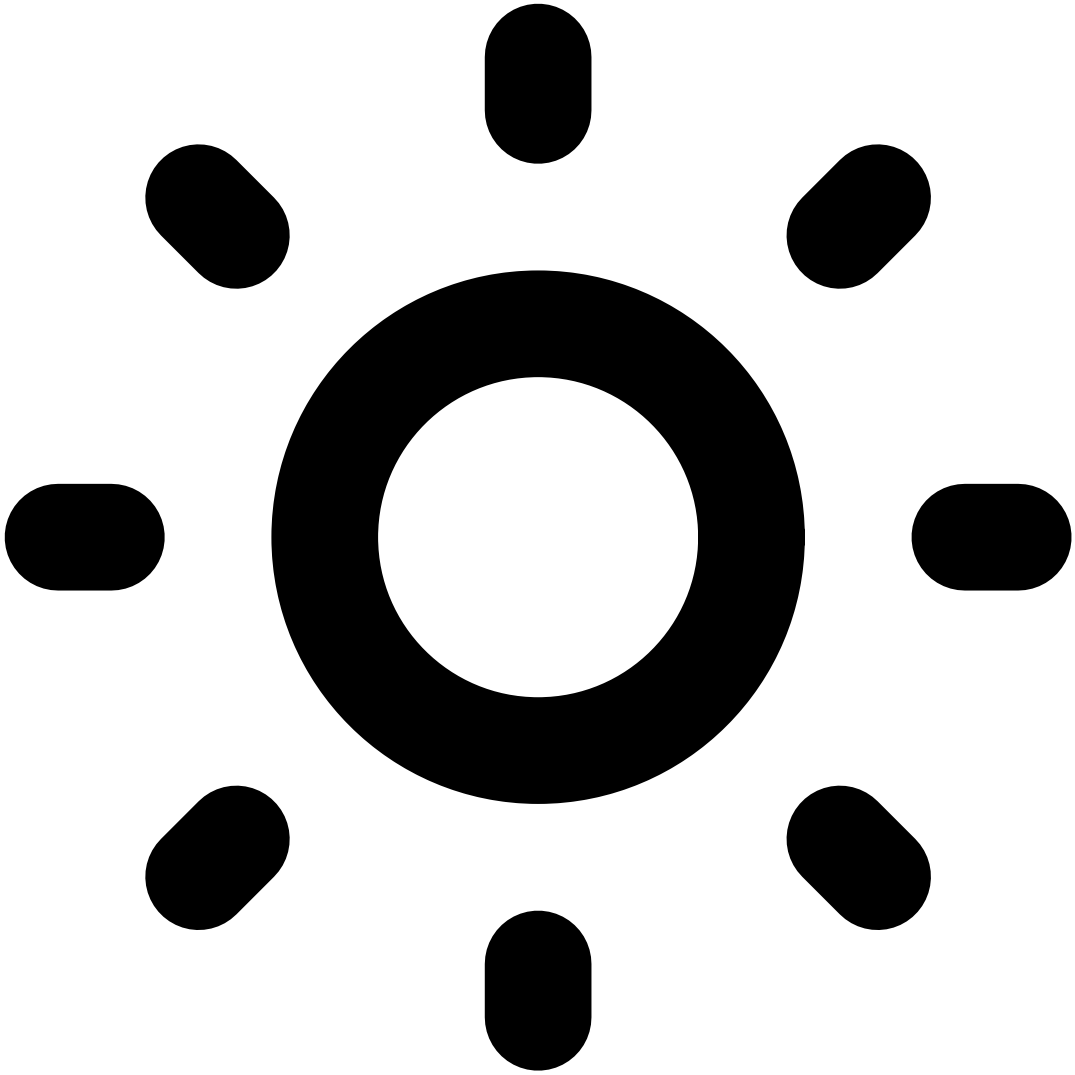
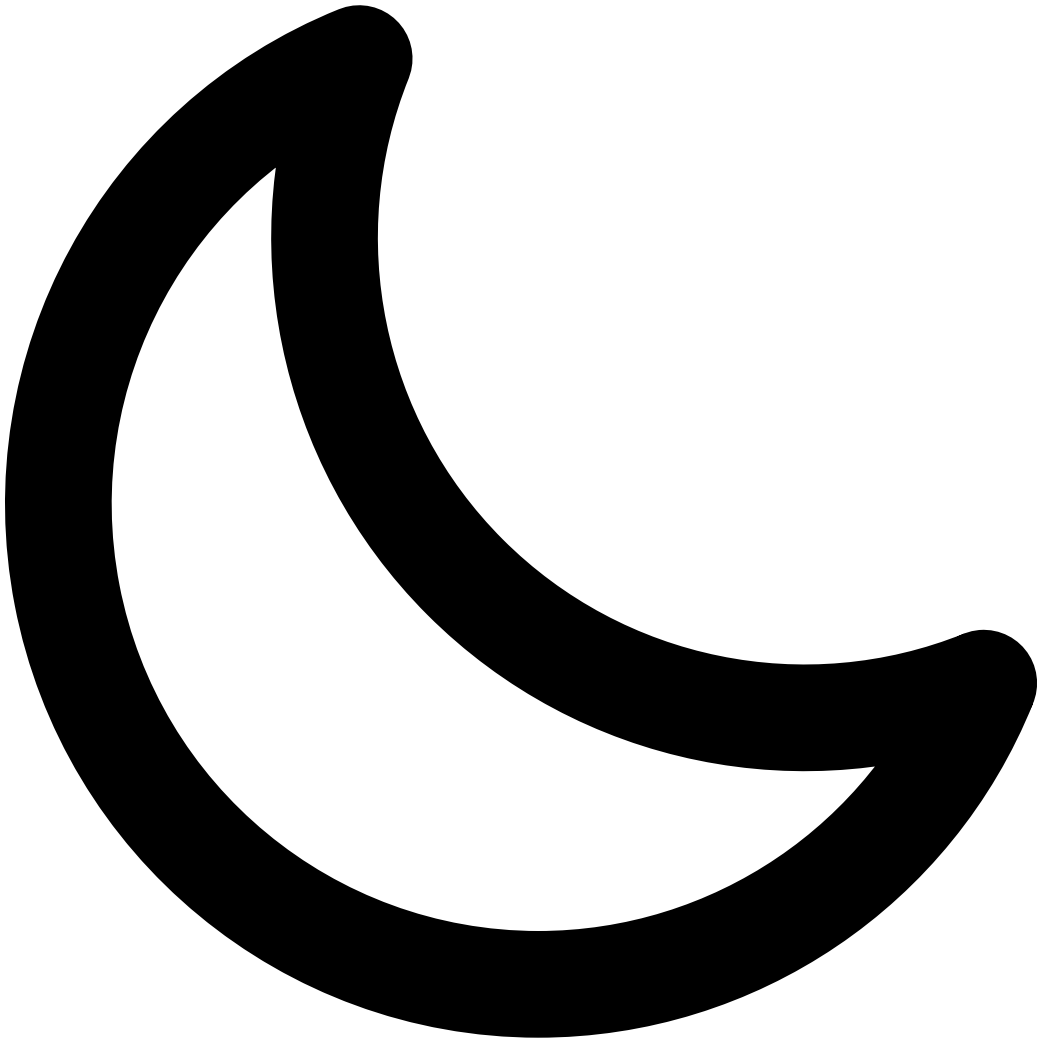


☐ Dark Mode Toggle



Layer 1 Should Be Innovative in the Short Term but Less in the Long Term

2018 Aug 26

See all posts

See update 2018-08-29

One of the key tradeoffs in blockchain design is whether to build more functionality into base-layer blockchains themselves ("layer 1"), or to build it into protocols that live on top of the blockchain, and can be created and modified without changing the blockchain itself ("layer 2"). The tradeoff has so far shown itself most in the scaling debates, with block size increases (and sharding) on one side and layer-2 solutions like Plasma and channels on the other, and to some extent blockchain governance, with loss and theft recovery being solvable by either the DAO fork or generalizations thereof such as EIP 867, or by layer-2 solutions such as Reversible Ether (RETH). So which approach is ultimately better? Those who know me well, or have seen me out myself as a dirty centrist, know that I will inevitably say "some of both". However, in the longer term, I do think that as blockchains become more and more mature, layer 1 will necessarily stabilize, and layer 2 will take on more and more of the burden of ongoing innovation and change.

There are several reasons why. The first is that layer 1 solutions require ongoing protocol change to happen at the base protocol layer, base layer protocol change requires governance, and **it has still not been shown that, in the long term, highly "activist" blockchain governance can continue without causing ongoing political uncertainty or collapsing into centralization**.

To take an example from another sphere, consider Moxie Marlinspike's defense of Signal's centralized and non-federated nature. A document by a company defending its right to maintain control over an ecosystem it depends on for its key business should of course be viewed with massive grains of salt, but one can still benefit from the arguments. Quoting:

> One of the controversial things we did with Signal early on was to build it as an unfederated service. Nothing about any of the protocols we've developed requires centralization; it's entirely possible to build a federated Signal Protocol-based messenger, but

> I no longer believe that it is possible to build a competitive federated messenger at all.

And:

> Their retort was "that's dumb, how far would the internet have gotten without interoperable protocols defined by 3rd parties?" I thought about it. We got to the first production version of IP, and have been trying for the past 20 years to switch to a second production version of IP with limited success. We got to HTTP version 1.1 in 1997, and have been stuck there until now. Likewise, SMTP, IRC, DNS, XMPP, are all similarly frozen in time circa the late 1990s. To answer his question, that's how far the internet got. It got to the late 90s. That has taken us pretty far, but it's undeniable that once you federate your protocol, it becomes very difficult to make changes. And right now, at the application level, things that stand still don't fare very well in a world where the ecosystem is moving ... So long as federation means stasis while centralization means movement, federated protocols are going to have trouble existing in a software climate that demands movement as it does today.

At this point in time, and in the medium term going forward, it seems clear that decentralized application platforms, cryptocurrency payments, identity systems, reputation systems, decentralized exchange mechanisms, auctions, privacy solutions, programming languages that support privacy solutions, and most other interesting things that can be done on blockchains are spheres where there will continue to be significant and ongoing innovation. Decentralized application platforms often need continued reductions in confirmation time, payments need fast confirmations, low transaction costs, privacy, and many other built-in features, exchanges are appearing in many shapes and sizes including on-chain automated market makers, frequent batch auctions, combinatorial auctions and more. Hence, "building in" any of these into a base layer blockchain would be a bad idea, as it would create a high level of governance overhead as the platform would have to continually discuss, implement and coordinate newly discovered technical improvements. For the same reason federated messengers have a hard time getting off the ground without re-centralizing, blockchains would also need to choose between adopting activist governance, with the perils that entails, and falling behind newly appearing alternatives.

Even Ethereum's limited level of application-specific functionality, precompiles, has seen some of this effect. Less than a year ago, Ethereum adopted the Byzantium hard fork, including operations to facilitate elliptic curve operations needed for ring signatures, ZK-SNARKs and other applications, using the alt-bn128 curve. Now, Zcash and other blockchains are moving toward BLS-12-381, and Ethereum would need to fork again to catch up. In part to avoid having similar problems in the future, the Ethereum community is looking to upgrade the EVM to E-WASM, a virtual machine that is sufficiently more efficient that there is far less need to incorporate application-specific precompiles.

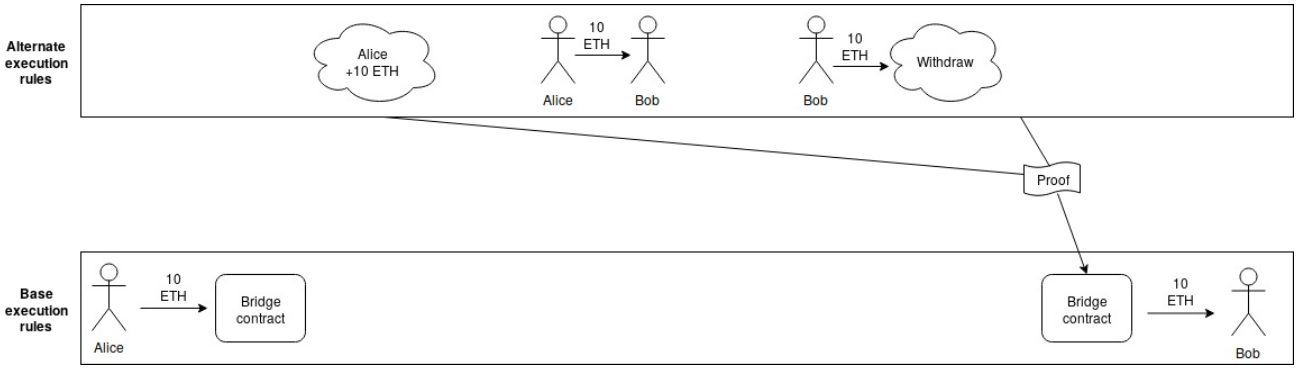
But there is also a second argument in favor of layer 2 solutions, one that does not depend on speed of anticipated technical development: *sometimes there are inevitable tradeoffs, with no single globally optimal solution*. This is less easily visible in Ethereum 1.0-style blockchains, where there are certain models that are reasonably universal (eg. Ethereum's account-based model is one). In *sharded* blockchains, however, one type of question that does *not* exist in Ethereum today crops up: how to do cross-shard transactions? That is, suppose that the blockchain state has regions A and B, where few or no nodes are processing both A and B. How does the system handle transactions that affect both A and B?

The current answer involves asynchronous cross-shard communication, which is sufficient for transferring assets and some other applications, but insufficient for many others. Synchronous operations (eg. to solve the train and hotel problem) can be bolted on top with cross-shard yanking, but this requires multiple rounds of cross-shard interaction, leading to significant delays. We can solve these problems with a synchronous execution scheme, but this comes with several tradeoffs:

- The system cannot process more than one transaction for the same account per block
- Transactions must declare in advance what shards and addresses they affect
- There is a high risk of any given transaction failing (and still being required to pay fees!) if the transaction is only accepted in some of the shards that it affects but not others

It seems very likely that a better scheme can be developed, but it would be more complex, and may well have limitations that this scheme does not. There are known results preventing perfection; at the very least, Amdahl's law puts a hard limit on the ability of some applications and some types of interaction to process more transactions per second through parallelization.

So how do we create an environment where better schemes can be tested and deployed? The answer is an idea that can be credited to Justin Drake: layer 2 execution engines. Users would be able to send assets into a "bridge contract", which would calculate (using some indirect technique such as interactive verification or ZK-SNARKs) state roots using some alternative set of rules for processing the blockchain (think of this as equivalent to layer-two "meta-protocols" like Mastercoin/OMNI and Counterparty on top of Bitcoin, except because of the bridge contract these protocols would be able to handle assets whose "base ledger" is defined on the underlying protocol), and which would process withdrawals if and only if the alternative ruleset generates a withdrawal request.



Note that anyone can create a layer 2 execution engine at any time, different users can use different execution engines, and one can switch from one execution engine to any other, or to the base protocol, fairly quickly. The base blockchain no longer has to worry about being an optimal smart contract processing engine; it need only be a data availability layer with execution rules that are quasi-Turing-complete so that any layer 2 bridge contract can be built on top, and that allow basic operations to carry state between shards (in fact, only ETH transfers being fungible across shards is sufficient, but it takes very little effort to also allow cross-shard calls, so we may as well support them), but does not require complexity beyond that. Note also that layer 2 execution engines can have different state management rules than layer 1, eg. not having storage rent; anything goes, as it's the responsibility of the users of that specific execution engine to make sure that it is sustainable, and if they fail to do so the consequences are contained to within the users of that particular execution engine.

In the long run, layer 1 would not be actively competing on all of these improvements; it would simply provide a stable platform for the layer 2 innovation to happen on top. **Does this mean that, say, sharding is a bad idea, and we should keep the blockchain size and state small so that even 10 year old computers can process everyone's transactions? Absolutely not.** Even if execution engines are

something that gets partially or fully moved to layer 2, consensus on data ordering and availability is still a highly generalizable and necessary function; to see how difficult layer 2 execution engines are without layer 1 scalable data availability consensus, see the difficulties in Plasma research, and its difficulty of naturally extending to fully general purpose blockchains, for an example. And if people want to throw a hundred megabytes per second of data into a system where they need consensus on availability, then we need a hundred megabytes per second of data availability consensus.

Additionally, layer 1 can still improve on reducing latency; if layer 1 is slow, the only strategy for achieving very low latency is state channels, which often have high capital requirements and can be difficult to generalize. State channels will always beat layer 1 blockchains in latency as state channels require only a single network message, but in those cases where state channels do not work well, layer 1 blockchains can still come closer than they do today.

Hence, the other extreme position, that blockchain base layers can be truly absolutely minimal, and not bother with either a quasi-Turing-complete execution engine or scalability to beyond the capacity of a single node, is also clearly false; there is a certain minimal level of complexity that is required for base layers to be powerful enough for applications to build on top of them, and we have not yet reached that level. Additional complexity is needed, though it should be chosen very carefully to make sure that it is maximally general purpose, and not targeted toward specific applications or technologies that will go out of fashion in two years due to loss of interest or better alternatives.

And even in the future base layers will need to continue to make some upgrades, especially if new technologies (eg. STARKs reaching higher levels of maturity) allow them to achieve stronger properties than they could before, though developers today can take care to make base layer platforms maximally forward-compatible with such potential improvements. So it will continue to be true that a balance between layer 1 and layer 2 improvements is needed to continue improving scalability, privacy and versatility, though layer 2 will continue to take up a larger and larger share of the innovation over time.

Update 2018.08.29: Justin Drake pointed out to me another good reason why some features may be best implemented on layer 1: those features are public goods, and so could not be efficiently or reliably funded with feature-specific use fees, and hence are best paid for by subsidies paid out of issuance or burned transaction fees. One possible example of this is secure random number generation, and another is generation of zero knowledge proofs for more efficient client validation of correctness of various claims about blockchain contents or state.