In my opinion, the current sharding spec as described herehttps://github.com/ethereum/sharding/blob/develop/docs/doc.md is basically already good enough to get us to thousands of transactions per second with reasonable security properties, as well as the ability to add cross-shard transactions as a not-that-difficult second step [do people agree?]. The demand for scaling is urgent, so it seems reasonable to just build this possibly with minor changes, launch it as fast as safely possible and go from there.

That said, I'd like to make sure that the design decisions we make also give us freedom to upgrade the design over time, and don't end up hamstringing us in unexpected ways. Here are some ways in which we've been hamstrung so far:

- The lack of a chain ID in transactions made replay protection hard
- The lack of a version number in transactions made upgrading harder
- The fact that a transaction can access arbitrary parts of the state makes upgrades that facilitate parallelization hard
- The presence of synchronous cross-contract calls makes parallelization harder and makes re-entrancy attacks an issue
- The fact that code can be so large, and code is not in a Merkle tree-like structure, makes it hard to make reasonable bounds on the size of Merkle proofs

In sharded systems, there are many more tradeoffs. Here are ways that we could get hypothetically hamstrung:

- We agree to the 2L in 1L design (which commenters here seem to be positive on), but then decide that we want to switch to a design where contract storage is abstracted away entirely (eg. see Justin Drake's proposals). Grandfathering in old-style contracts becomes a challenge.
- We start off with the stateless model, and make assumptions around it (eg. instant shuffling), but then decide that we want to switch to a model where the top level of the tree (ie. accounts but not storage or other accumulators) is guaranteed to be stored by clients/miners.
- We start off with the current "N shards" model, with synchronous calls within each shard, but then decide that we want to switch to a "one contract per shard" model. However, cross-shard calls can only be asynchronous, so we'd have to make synchronous calls retroactively not work.

Are there any other such gotchas we should be worried about? Are there quick tweaks we can make to the current proposal to limit them? I'd be willing to consider switching to the actor model for v1 (it's not hard; basically, CALLs get processed only after all other CALLs, including the context that created them, end).