I am proud to present Manaflow, a new approach to smart contract scaling, representing the result of 6+ years of EVM know-how and 3+ years of scaling research.

This post is a short ‘pre-whitepaper’ intended to give enough information for developers (read: potential hires) to understand our approach, without giving away so much info that some other team could impelment it first. As with most smart contract architecture problems, the devil is in the details.

First, a short list of insights:

- Atomic composability in a single shared global runtime is the secret sauce that makes ethereum and defi so successful.
- Most approaches to smart contract scaling sacrifice atomic composability in some way. Even ETH2’s sharding breaks this critical property.
- UTXO is nearly-trivially-parallel. The double-spend (more generally, ‘double-use’) critical path can handle millions of state changes per second. (As an aside, UTXO is designed for big blocks. Regardless of your perspective on whether bitcoin “should” have big blocks, the fact is that it was designed

to have big blocks.)

- Existing UTXO systems have either limited, local statefulness, or they have global statefulness in a way that sacrifices the parallel validation nature of UTXO.
- If a system claims to have a model as good as EVM, it should be able to emulate the EVM. Thus our solution starts with EVM compatibilty and later could be extended to use LLVM or WASM.
- Emulating a single isolated EVM contract is not “EVM compatibility”! If contracts can’t atomically call other contracts, it’s not much better than traditional platforms.

Our approach unifies the global, synchronous, atomically composable logical view of EVM with the trivially parallel nature of UTXO validation. Only the state “double-use” is in the critical path. This critical path is blazing fast when there is no other validation that needs to be done.

The trick is that transactions declare not just the slots they will access, but the actual before and after values of those slots. This goes well beyond approaches like Vitalik’s ‘address access list’ proposal. The result is that each transaction can be validated by looking only at the one transaction. This is a big-block-UTXO-style scaling solution. In fact, it is ‘even more parallel’ than bitcoin-style UTXO, because transactions fully declare the ‘before’ states they are consuming. The entire blockchain could be validated in reverse!

More concretely, transactions are extended with a set of “moves”:

move : (mask // signature bitmask mark // short type tag, e.g. 's' for 'state slot' intx // transaction in which this state was last used addr // address this state belongs to slot // for 's' moves, the state slot ("key") prev // mark-specific value (e.g. 'before' value for 's' mark) next // mark-specific value (e.g. 'after' value for 's' mark))

(Here we refer to just ‘state’ moves, but proper EVM compatibility requires a few more ‘bookkeeping’ move types, which we will keep secret for now.)

The tradeoff here is that the problem of sequencing/coordinating transactions (“stitching together” valid transitions) is offloaded to secondary nodes and end users. State that is not highly contested can be filled in by the end user’s wallet. State that is highly contested is filled in by various ‘coordinators’, much like the sequencers of L2s like Optimism. Here we encounter the theme that often the best ‘solution’ is to expose the tradeoff for users (developers) to solve as they see fit. The important thing is that validators can accept batches of transactions from any number of coordinators and end users in parallel.

Notice the ‘signature bitmask’ field (mask

). The purpose of this field is to allow end users to leave some parts of the move undeclared, like a generalized form of Bitcoin’s SIGHASH variants. This allows coordinator nodes to fill in the values dynamically. A simple example is the resulting balance after a uniswap transaction. If the pool is highly contested, the exact value won’t be known until transaction fill time in the coordinator. Note that these value can still be constrained in the contract code! The design space for synchronization primitives that work ‘automagically’ from the point of view of the contract code is fascinating.

Validators maintain just a single index, the “utxo set” keyed by intx,mark,addr,slot

. Coordinators and other nodes that implement the web3 API (i.e., what is the value of a given state slot) maintain the secondary index keyed by mark,addr,slot

. This secondary index is not necessary for validation.

The reason we are going for an L2/sidechain instead of advocating for adding this directly to Ethereum L1 is that we completely remove the merkle-patricia state trie, a compromise that Ethereum is unlikely to be willing to make. Thin client proofs for Manaflow require a combination of simple SPV-style proofs of transaction combined with in-contract logic for cases where it is critical to know that the value in the transaction's moveset is the 'latest'.

The good news is that this transaction structure is very well-suited for fraud proofs, making optimistic rollups easy. In fact, it is so clean that it could even be verified with recursive ZK proofs, though we are not taking this approach for performance reasons.

We are currently in the sweet spot where the hardest architecture problems have been solved, but implementation has just begun and founder-sized stake is available for top hires.

If you're an Ethereum client dev looking for some fresh well-compensated and challenging work, please shoot me an email to hello@manaflow.io.

Happy to answer questions in this thread.