Transactions in Ethereum do not afford the sender a lot of flexibility in customizing execution: they allow “data” (say, function arguments) to be sent to a single contract call, and nothing else. Complementary to this rigidity of form is the highly permissive execution model of the EVM, which provides inter-contract calls as a first-class language feature alongside more ordinary programming constructs that occur intra-contract. This, in turn, leads to enormous difficulties: not only such prosaic failures as reentrancy bugs, but also difficulties in moving Ethereum forward, because it makes the effect of transactions on EVM state undecidable

and so sabotages the task of separating transactions into parallel shards for scaling. Rather than taking my own crack at that problem, here I propose redefining it so that it is no longer hard.

Sharding would be easy if the stateful effects of each transaction were visible in the transaction message: this would self-organize transactions into a dependency graph that yields actually a much greater granularity of sharding that any of the current proposals or even fantasies of the future (in fact, it provides the maximum granularity). It seems worthwhile to acknowledge that, while Ethereum with computed contract calls is Turing complete both within a contract and as an aggregate of all interoperating contracts, it also places an enormous burden on scalability theorists; by contrast, an Ethereum with static contract calls would still have great computational expressiveness (and would not necessarily preclude any known useful contract schemas), and would also make scaling a reality immediately.

Why is this post entitled “smart transactions”? Because in a programming model that declares contract calls in the transaction message, the actual activity of the transaction would be to merge their return values into some useful activity. This kind of orchestration is, in the current EVM, performed by each contract as it calls others, and so what I’m describing is an inversion of control that makes transactions into one-time smart contracts. The transaction message would contain actual executable code to combine contract return values. Contract return values, correspondingly, would become the primary product of a contract call, and not its various internal side-effects. This gives transactions a functional-language aspect, though it does not necessitate a functional language for writing transaction or contract code

, because it actually makes the language of contract calls and transactions into an entirely separate machine from the language of contracts themselves.

Do I have such a model in mind? Yes, of course; I have spent a while developing it and if you’ve seen my name before it’s probably in connection with it. But I am not bringing it up here because I’d like this post to be about what a new perspective on transactions can bring to the Ethereum scalability table: in other words, about ideas, and not advertisement.

I’ve kept this prose intentionally free of technical details: no discussion of data structures or tunable parameters or cryptoeconomic games. I don’t think they are necessary at this altitude, and would refer to a specific realization of this execution model anyway, which is not the point.

In summary: invert inter-contract dependencies by resolving them all at the top level of the transaction for ease of static analysis, facilitating sharding. Thoughts?