This is a (probably stupid) idea that I thought of. There are probably a million reasons why this is a really bad idea, but I can't think of any. Hence, I'm posting it here.

UPDATE: This proposal is extremely similar toEIP-86. In fact, it's nearly identical. My original proposal is below for posterity, but if it's a good idea then EIP-86 should be resurrected.

Assumptionless Ethereum

As I see it, there are four main invariants of Ethereum that don't have to do with the EVM. The first is the existence of ether (i.e. the coin), which is the basis for economic incentives that power the EVM. As such, it's necessary. The other essential invariant is the per-block gas limit, which ensures that Ethereum doesn't grow too large.

I argue that the other two invariants, however, are not necessary and can be phased out safely. Those are hardcoded gas costs and authentication. Gas costs (as opposed to gas limits), for the purposes of this discussion, are the hardcoded burning of ether by using opcodes and

the hardcoded priority fee of EIP-1559. Authentication is the invariant that all transactions originate from EOA accounts that use ECDSA key pairs, and that any transaction that is unsigned is invalid.

Economic incentives to create a perfect generalized MEV extractor mean that, at some point in the near future, an open-source generalized MEV extractor will appear. If this occurs, I show that a mechanism can be designed to replicate the existing functionalities of gas costs and EOAs. Note that this entire proposal hinges on this. This solution WILL NOT WORK until

this idealized MEV extractor exists. As such, this is a very long-term proposal.

Turning EOAs Into Contract Accounts

The Issue With EOAs

You know why EOAs are bad. If you're looking for a discussion of why EOAs are bad, look elsewhere. If you really want me to provide sources, then you're lucky that you have never lost a seed phrase or had it stolen. There are plenty of people that have. Ask them about it.

Two New Opcodes: APPROVE

and DENY

In order to make this system work, two opcodes are introduced: APPROVE

and DENY

. If a transaction reverts, any state changes made before the APPROVE

or DENY

are NOT reverted.

APPROVE

sets msg.sender

to the current contract, or reverts if msg.sender

is already set. DENY

reverts with an error stating a lack of authorization.

Using Ethereum Without An EOA Entrypoint

Since Ethereum is turing-complete, one could simulate an EOA with functions that either DENY

on an invalid signature or APPROVE

and CALL

the respective function. All EOAs could be replaced with that implementation, and new EOAs are forbidden.

Running Transactions Without A Guarantee Of Sucess

If you're a validator, you might be worried about DoS attacks. What if someone spams a bunch of transactions that use up a lot of processing power before they end up reverting? An idealized MEV bot still can't solve the halting problem!

Well, the answer is simple. Any state changes executed before the APPROVE

or DENY

stay in effect even if the transition ends up reverting. As long as the amount that's paid in MEV is greater than the cost of computation, the computation continues. Otherwise, give up and find a new transaction.

One might make the argument that checking if you've been paid enough is nontrivial. However, it's as trivial as gas costs, except not fixed to certain amounts. And a small amount of compute time (not much more than is already needed to validate ECDSA signatures) can safely be allotted without the risk of a DoS attack, as evidenced by the fact that submitting many invalid ECDSA signatures has never resulted in a DoS attack.

Using Ethereum Without An Account

The fact that you have to have an account to create an account might appear to be an issue: anybody should be able to receive Ethereum without first having some. So how can you solve this problem?

Very easily. Just have a payable

contract that stores ether to a designated ECDSA "account" (not actual account), and authenticate with any ECDSA key with enough balance to either pay a real account, ECDSA account, or make a new contract account. This allows anybody without an account to accept and use Ethereum.

Turning Gas Costs Into MEV'

Note that this proposes removing gas costs, not

per-block gas limits. Per-block gas limits, for the foreseeable future, are essential.

The Priority Fee

The EIP-1559 priority fee is nice from the point of view of a MEV extractor: built into the protocol is a near guarantee that you will receive a certain amount of Ethereum.

I argue that the only opcode that is necessary to replace EIP-1559 priority fees is an opcode that unconditionally transfers an amount of ether to a given account (I'll call it PAY

)–similar to CALL

, but without actually calling a function. To functionally replace EIP-1559, all that is needed is to add the following to the entry point described in replacing EOAs with Contract Accounts.

contract BasicContractAccount { // Some other functions

function someEntrypointFunction(bytes someParameter, uint256 priorityFee) {

. // Do Authentication Stuff block.coinbase.pay(pritorityFee); // No more EIP-1559 // APPROVE or DENY // Do the entrypoint stuff } }

The Opcode Costs

Most opcodes consume gas in the form of ether. This is because it is assumed that all validators have to run the code, and computation costs money!

However, as discussed in the "Running Transactions Without A Guarantee Of Sucess" section, one can replace this with MEV to convince validators that it's worth including their transaction.

Note again that the per-block opcode limit will remain untouched.

Conclusion

With three new opcodes, gas costs and EOAs can be removed.