EELS
The future of Ethereum Execution Layer Specifications

Peter Davies
EELS Team
What do I mean by “Execution Layer”?  

- We only care about the “state transition function”.  
  - Can a new block be added to the end of a chain?  
  - What happens to the chain state when we add a block?  
- Everything else is out of scope.  
  - Fork choice  
  - Reorgs  
  - Networking  
  - APIs (e.g. JSON RPC)  
  - Performance  
  - Transaction Gossip
Current Sources of Information

- Yellow Paper
- EIPs
- Testsuite
- Client source code

Thus we are able to define the block header validity function $V(H)$:

\[
V(H) \equiv n \leq \frac{2^{256}}{H_d} \land m = H_m \land H_d = D(H) \land H_g \leq H_1 \land H_1 < P(H)_{H_1} + \left[ \frac{P(H)_{H_1}}{1024} \right] \land H_1 > P(H)_{H_1} - \left[ \frac{P(H)_{H_1}}{1024} \right] \land H_1 \geq 5000 \land H_s > P(H)_{H_s} \land H_i = P(H)_{H_i} + 1 \land \|H_x\| \leq 32
\]

where $(n, m) = \text{PoW}(H_W, H_n, d)$.

Noting additionally that extraData must be at most 32 bytes.
Specifications need to be part of standards processes

- Updating standards can’t be an afterthought
- Code that isn’t tested isn’t worth anything

“Beware of bugs in the above code; I have only proved it correct, not tried it.”
– Donald Knuth
Our approach

- Specifications are written in code
  - Python without classes/methods (basically pseudocode)
  - Common language of all programmers
  - Can be executed
- Focus solely on readability
  - Performance is for real clients
- Keep forks separate rather than lots of conditionals
  - Horrendous for code duplication, great for the casual reader
  - Specialist diff tools for comparing hardforks
def sload(evm: Evm) -> None:
    
    Loads to the stack, the value corresponding to a certain key from the
    storage of the current account.

    Parameters
    ----------
    evm:
        The current EVM frame.
    
    # STACK
    key = pop(evm.stack).to_be_bytes32()

    # GAS
    if (evm.message.current_target, key) in evm.accessed_storage_keys:
        charge_gas(evm, GAS_WARM_ACCESS)
    else:
        evm.accessed_storage_keys.add((evm.message.current_target, key))
        charge_gas(evm, GAS_COLD_SLOAD)

    # OPERATION
    value = get_storage(evm.env.state, evm.message.current_target, key)

    push(evm.stack, value)

    # PROGRAM COUNTER
    evm.pc += 1
The two sides of development

- R&D people (e.g. Vitalik Buterin)
  - Interested in theoretical concerns
  - Don’t care about performance complexities
  - Want a flexible playground

- Implementers (e.g. Péter Szilágyi)
  - Care about precise details
  - Want to focus on complicated performance issues (DB structure, etc...)

EELS provides a common framework for these two sides to talk to each other.
Development stages in an EELS world

R&D:
1. Develop an idea to improve the execution layer
2. Prototype the idea in EELS

EELS:
3. Integrate with other proposals to make a hardfork in EELS
4. Fill tests, start ephemeral testnets?

Implementers:
5. Implement in production clients
6. Deploy on testnets and mainnet
Current Status

- All hardforks are implemented (The Merge is still a PR)
- Refactoring complete and code freeze in November (hopefully)
- Shanghai governance shadowing
How you can help!

- We don’t need your help until we’ve finished coding
- Implement your favourite EIP and give us feedback
Questions?

ethereum.github.io/execution-specs

github.com/ethereum/execution-specs
Thanks for listening!

ethereum.github.io/execution-specs

github.com/ethereum/execution-specs