Some Ewasm Updates

Eth1x/Istanbul meetup Berlin 2019/4/18

Casey Detrio
Pawel Bylica
Alex Beregszaszi
Lane Rettig

Broad scope of work

- EVM maintenance
  - EVMC, aleth, EthereumJS
- Wasm (applied) research
  - Benchmarks, performance, metering, EEI design, ETH2 & sharding
- Wasm engineering
  - Interpreters, EEI implementation, chisel, sentinel

Agenda

- Problems
- Different directions on mainnet
- Metering
- Some proposed EIPs
- Eth2.0 update
- Update on tooling
- Current focus

Problem statement:

Save Ethereum, Scale Ethereum

Better Ethereum

- Better usability would be nice

Better Ethereum

- Better usability would be nice
- If it doesn’t survive, usability won’t matter

Better Ethereum

- Better usability would be nice
- If it doesn’t survive, usability won’t matter
- If it can’t scale, usability won’t matter

Save Ethereum

- State growth is unbounded, and must be stopped.

Save Ethereum

- State growth is unbounded, and must be stopped.

- This rules out “Ewasm 1.0” - EVM 1.0 mirrored in wasm - because the storage model is not compatible with rent.

Save Ethereum

- State growth is unbounded, and must be stopped.
- This rules out “Ewasm 1.0” - EVM 1.0 mirrored in wasm - because the storage model is not incentive compatible with rent.
- Need “Ewasm 1.x” - new storage model designed for rent.

Scale Ethereum

Scale Ethereum: what is the bottleneck?

Scale Ethereum: what is the bottleneck?

- Disk I/O - in most cases (storage opcodes already consume most gas, but still massively underpriced - see SLOAD repricing EIP and EVM benchmarks).

Scale Ethereum: what is the bottleneck?

- Disk I/O - in most cases (storage opcodes already consume most gas, but still massively underpriced - see SLOAD repricing EIP and EVM benchmarks).

- Computation - in some cases (where computation is currently a bottleneck, people propose precompiles).

Eth1.x: Shifting the bottleneck

- Hitting the physical limits of Disk I/O
- But network I/O is under-utilized

Eth1.x: Shifting the bottleneck

- Hitting the physical limits of Disk I/O
- But network I/O is under-utilized
- Solution: utilize more network I/O

Eth1.x: Shifting the bottleneck

- Hitting the physical limits of Disk I/O
- But network I/O is under-utilized
- Solution: utilize more network I/O
- price of storage⬆️

Eth1.x: Shifting the bottleneck

- Hitting the physical limits of Disk I/O
- But network I/O is under-utilized
- Solution: utilize more network I/O
- price of storage⬆
- price of tx data ⬇

Space-time tradeoff: squeeze more gains

Space-time tradeoff: squeeze more gains

- With more computational time, do same task with less space

Space-time tradeoff: squeeze more gains

- With more computational time, do same task with less space
- Faster VM -> more tasks, with same space

Space-time tradeoff: squeeze more gains

- With more computational time, do same task with less space
- Faster VM -> more tasks, with same space

- Will a new VM make it easier to add “precompiles”?

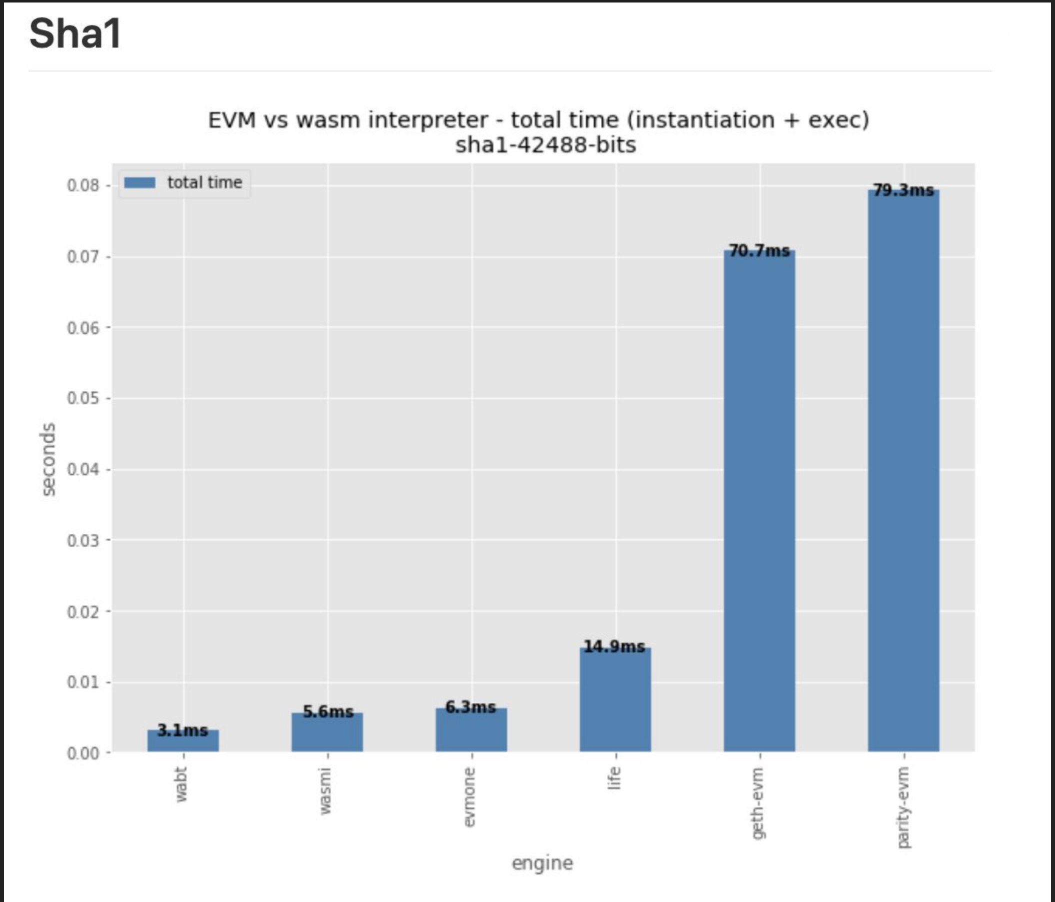
Space-time tradeoff: squeeze more gains

- With more computational time, do same task with less space
- Faster VM -> more tasks, with same space

- Will a new VM make it easier to add “precompiles”?
- Will a new VM be faster?

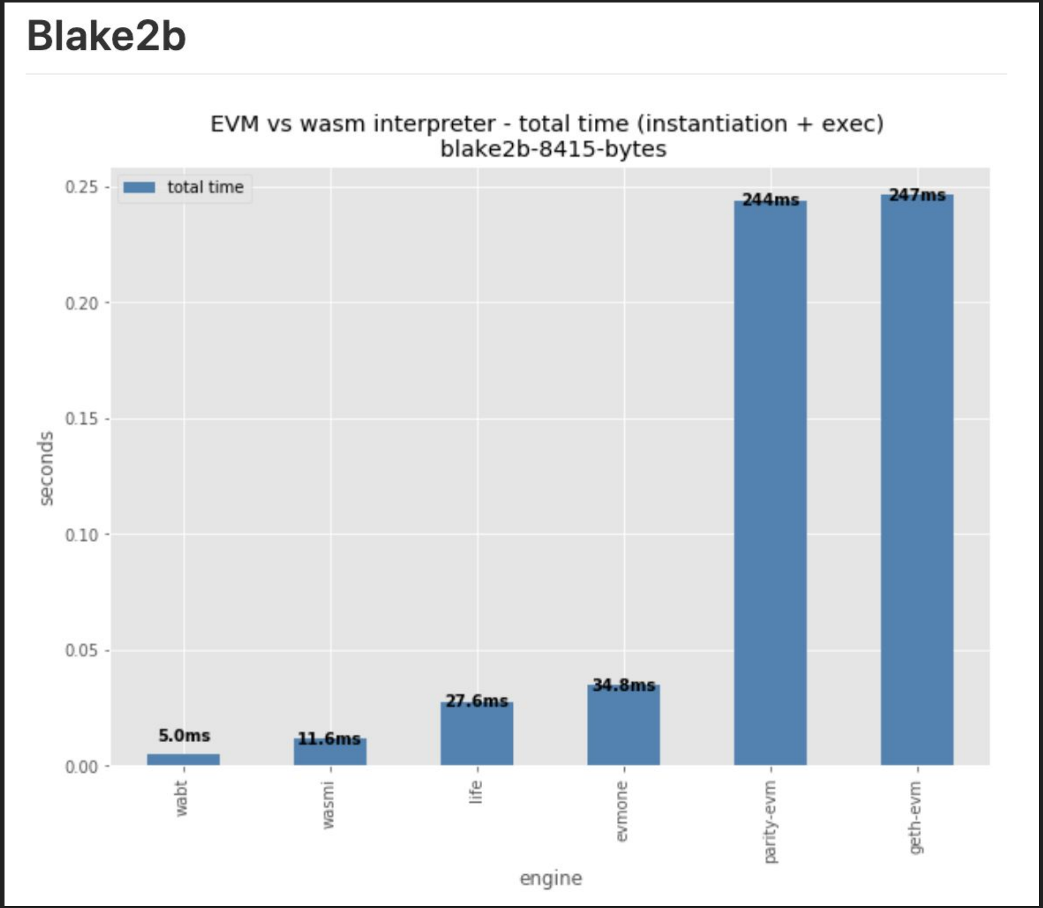
EVM vs Ewasm shootout - Interpreters

EVM vs Ewasm shootout - Interpreters



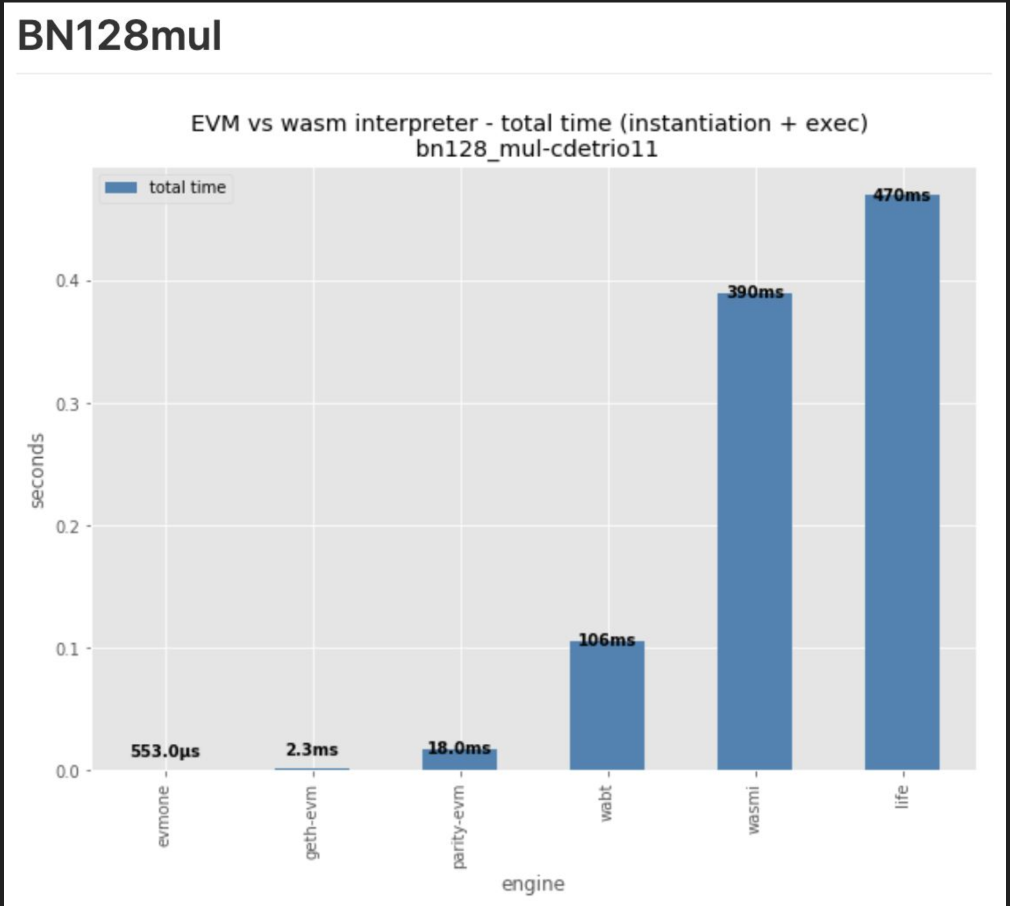
Sha1

EVM vs Ewasm shootout - Interpreters



Blake2b

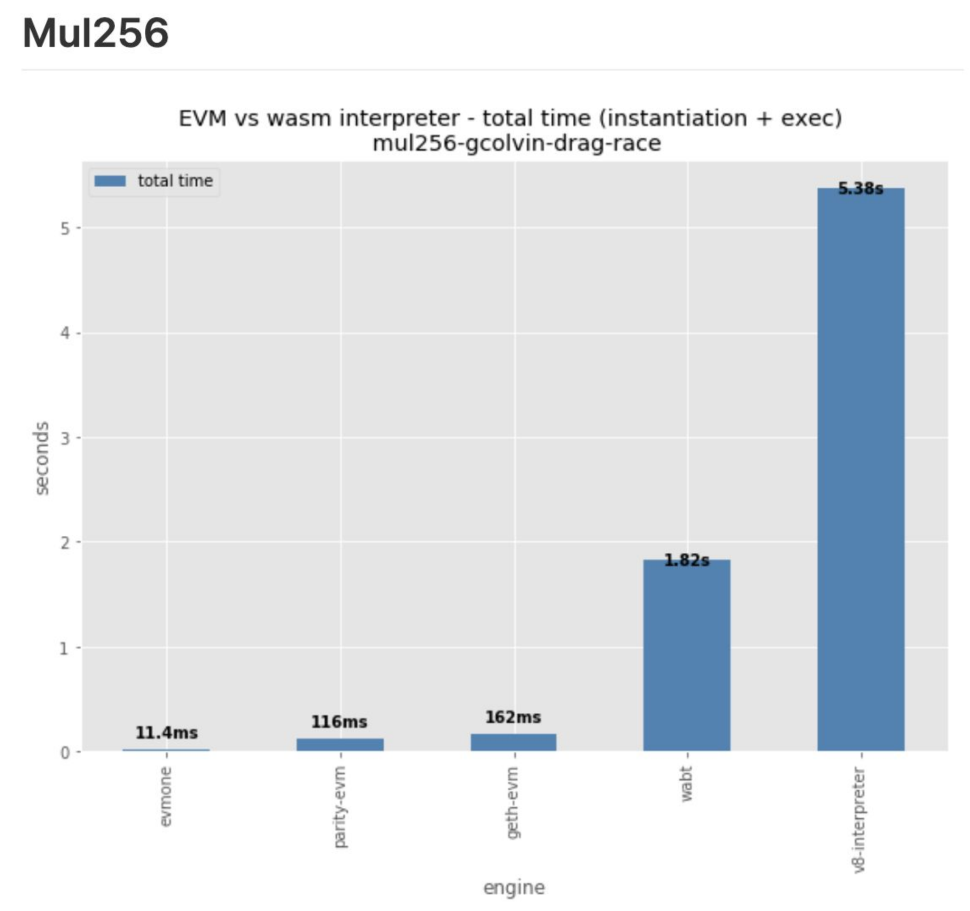
EVM vs Ewasm shootout - Interpreters



BN128mul

EVM vs wasm interpreter - total time (instantiation + exec)
bn128_mul-cdetrio11

EVM vs Ewasm shootout - Interpreters



Mul256

EVM vs wasm interpreter - total time (instantiation + exec)
mul256-gcolvin-drag-race

EVM vs Ewasm shootout: what's next?

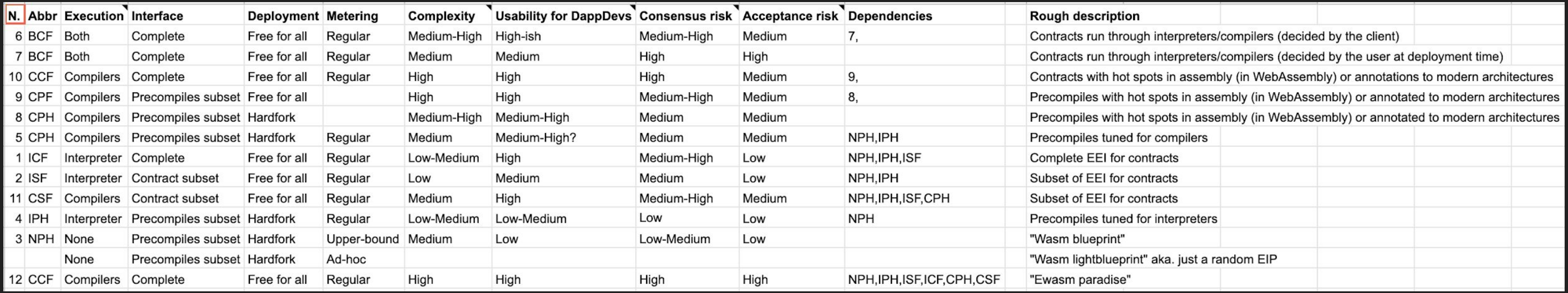
- Hack wasm interpreters to bring 256-bit performance up to par with EVM
- More benchmarks (realistic use cases)
- Keep working on wasm compiler engines

Ewasm 1.x Design Space

Original Design

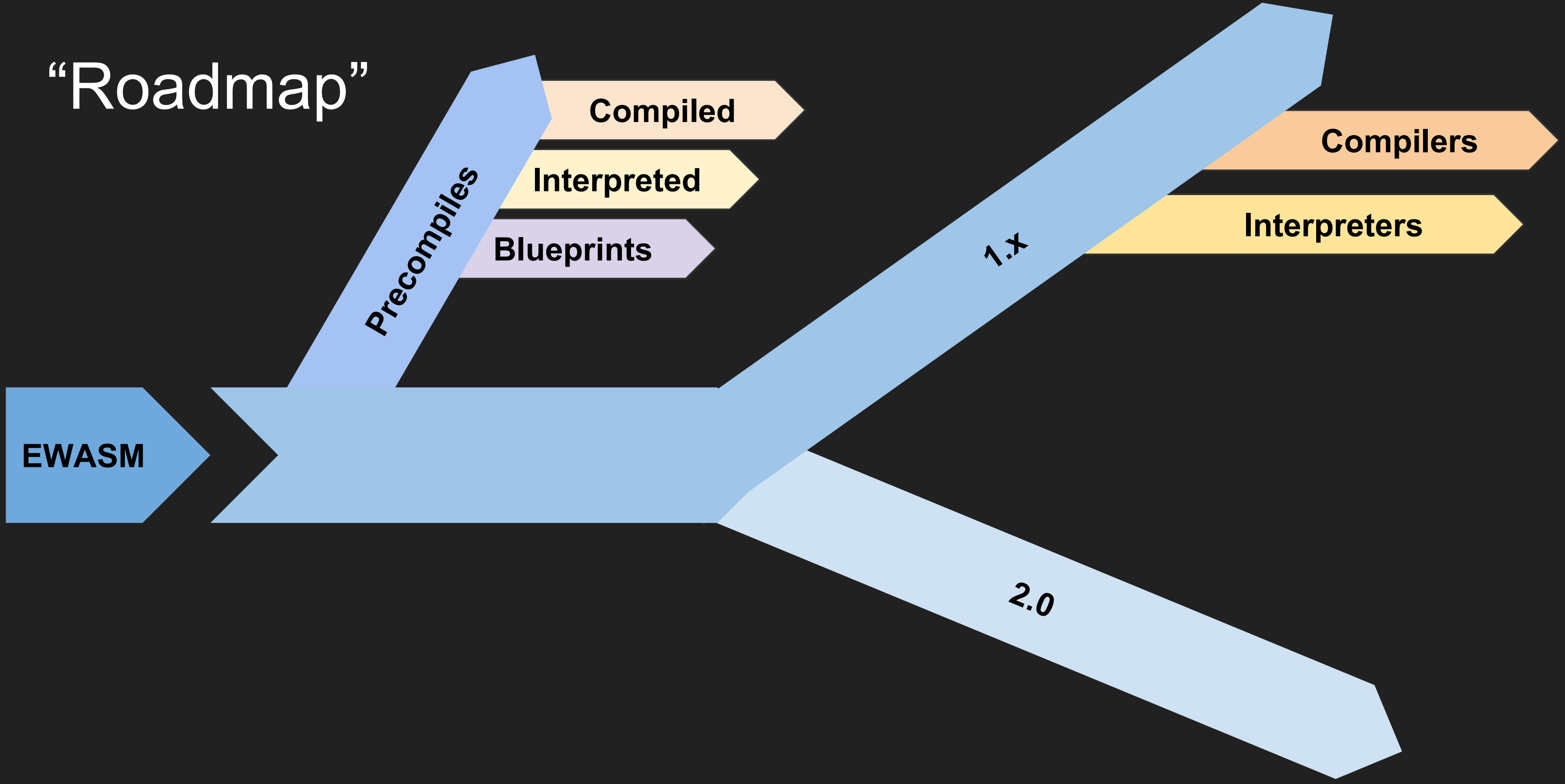
New generation of EVM for Ethereum 1.0 contracts

Lots of options



N.	Abbr	Execution	Interface	Deployment	Metering	Complexity	Usability for DappDevs	Consensus risk	Acceptance risk	Dependencies	Rough description
6	BCF	Both	Complete	Free for all	Regular	Medium-High	High-ish	Medium-High	Medium	7,	Contracts run through interpreters/compilers (decided by the client)
7	BCF	Both	Complete	Free for all	Regular	Medium	Medium	High	High		Contracts run through interpreters/compilers (decided by the user at deployment time)
10	CCF	Compilers	Complete	Free for all	Regular	High	High	High	Medium	9,	Contracts with hot spots in assembly (in WebAssembly) or annotations to modern architectures
9	CPF	Compilers	Precompiles subset	Free for all		High	High	Medium-High	Medium	8,	Precompiles with hot spots in assembly (in WebAssembly) or annotated to modern architectures
8	CPH	Compilers	Precompiles subset	Hardfork		Medium-High	Medium-High	Medium	Medium		Precompiles with hot spots in assembly (in WebAssembly) or annotated to modern architectures
5	CPH	Compilers	Precompiles subset	Hardfork	Regular	Medium	Medium-High?	Medium	Medium	NPH,IPH	Precompiles tuned for compilers
1	ICF	Interpreter	Complete	Free for all	Regular	Low-Medium	High	Medium-High	Low	NPH,IPH,ISF	Complete EEI for contracts
2	ISF	Interpreter	Contract subset	Free for all	Regular	Low	Medium	Medium	Low	NPH,IPH	Subset of EEI for contracts
11	CSF	Compilers	Contract subset	Free for all	Regular	Medium	High	Medium-High	Medium	NPH,IPH,ISF,CPH	Subset of EEI for contracts
4	IPH	Interpreter	Precompiles subset	Hardfork	Regular	Low-Medium	Low-Medium	Low	Low	NPH	Precompiles tuned for interpreters
3	NPH	None	Precompiles subset	Hardfork	Upper-bound	Medium	Low	Low-Medium	Low		"Wasm blueprint"
		None	Precompiles subset	Hardfork	Ad-hoc						"Wasm lightblueprint" aka. just a random EIP
12	CCF	Compilers	Complete	Free for all	Regular	High	High	High	High	NPH,IPH,ISF,ICF,CPH,CSF	"Ewasm paradise"

“Roadmap”


EWASM
Precompiles
Compiled
Interpreted
Blueprints
1.x
Compilers
Interpreters
2.0

Decision points

- Interpreter or compiler?
- Method of deployment?
- Method of metering?
- Precompiles only or any contract?
- Precompiles tuned for interpreters or for compilers?
- Complete EEI or subset of EEI?
- Does client or user choose compilation/interpretation?

Interpreter or compiler?

Does client or user choose compilation/interpretation?

Method of deployment?

Method of metering?

Precompiles only or any contract?

Precompiles tuned for interpreters or for compilers?

Complete EEI or subset of EEI?

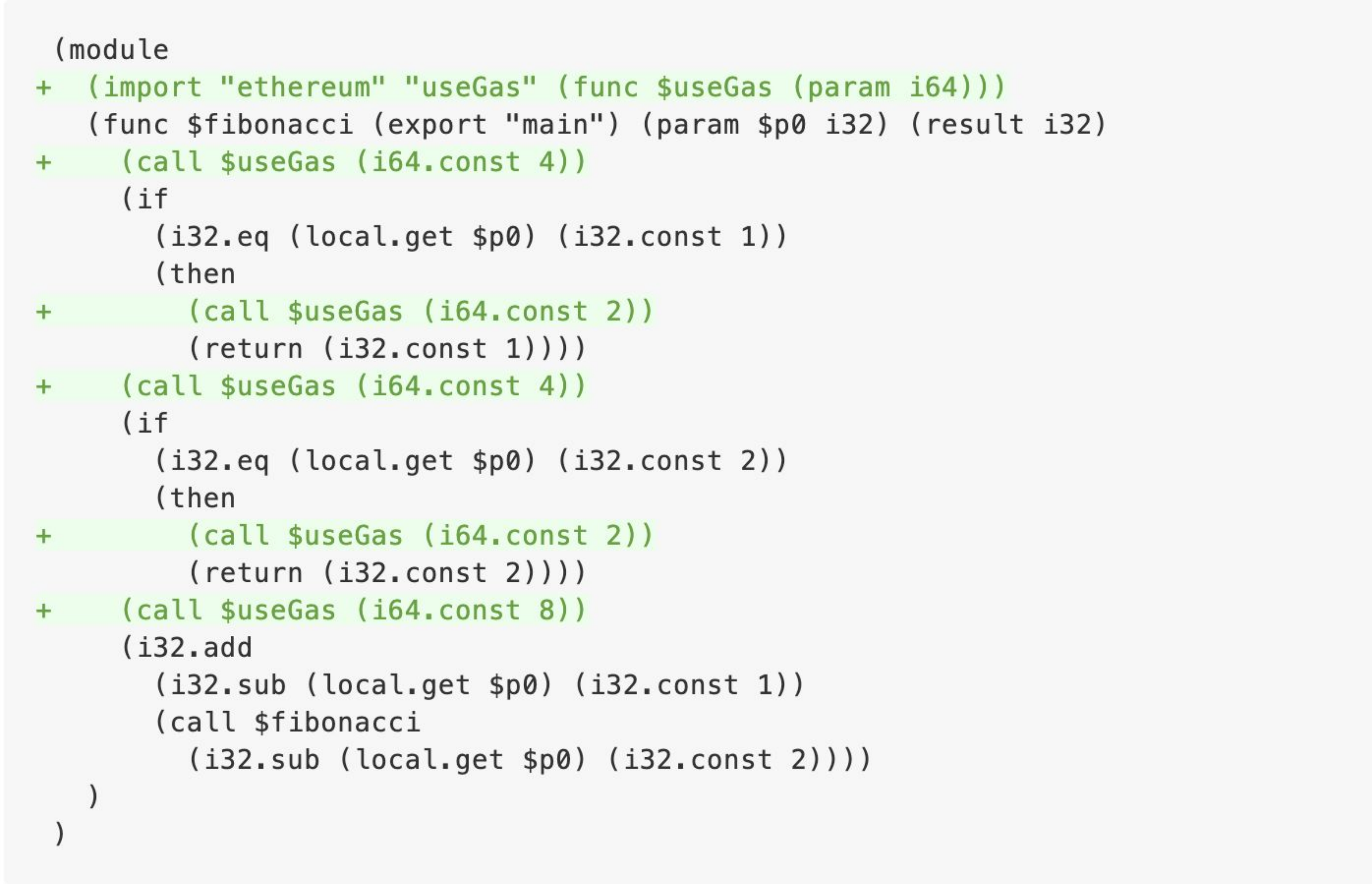
Precompiles evaluated

- blake2: nay
- sha1: nay
- ed25519: **yay**
- bls12 pairings: *yay-ish*
- bls12-381 pubkey aggregation and sig verification:
    - this includes pairing computation
- bls12 signature recovery: nay
- secp256k1: nay
- “generic elliptic curve utils”: **yay**
- keccak256: *nay-ish* unless ewasm contracts are added
- sha3: nay
- parameterized keccak: *yay-ish*
- sha2: nay
- unlimited bignums: nay
- 256-bit bignums: *nay-ish*
- primality testing: nay
- multisig: nay

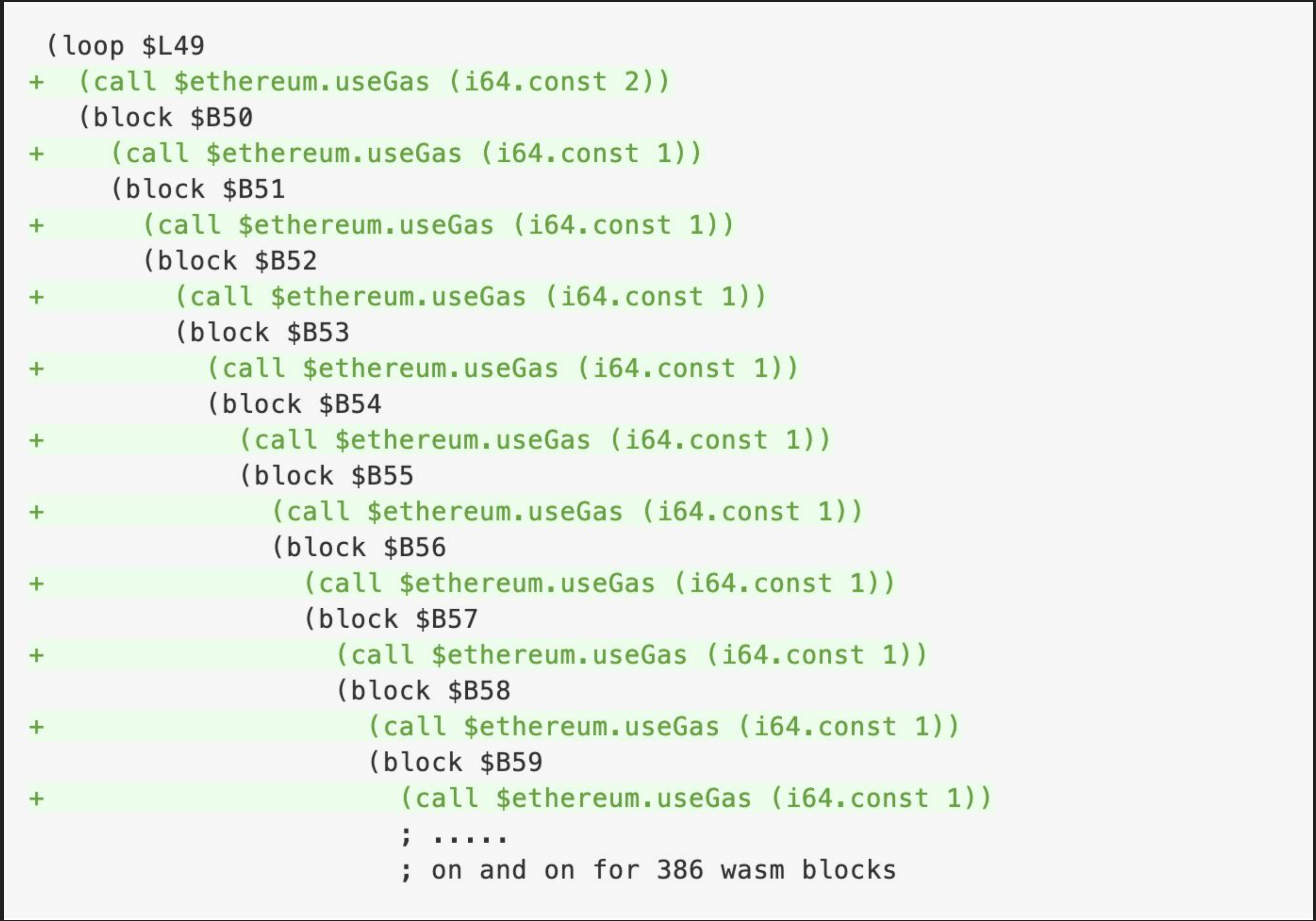
Metering?

Metering options

1. Basic block metering
2. Super block metering
3. Basic block with inline metering
4. Super block with inline metering
5. Lifting useGas statements e.g. to outside of constant-time loops



```
 (module
+  (import "ethereum" "useGas" (func $useGas (param i64)))
   (func $fibonacci (export "main") (param $p0 i32) (result i32)
+    (call $useGas (i64.const 4))
     (if
       (i32.eq (local.get $p0) (i32.const 1))
       (then
+        (call $useGas (i64.const 2))
         (return (i32.const 1))))
+    (call $useGas (i64.const 4))
     (if
       (i32.eq (local.get $p0) (i32.const 2))
       (then
+        (call $useGas (i64.const 2))
         (return (i32.const 2))))
+    (call $useGas (i64.const 8))
     (i32.add
       (i32.sub (local.get $p0) (i32.const 1))
       (call $fibonacci
         (i32.sub (local.get $p0) (i32.const 2))))
   )
 )
```



```
 (loop $L49
+  (call $ethereum.useGas (i64.const 2))
   (block $B50
+    (call $ethereum.useGas (i64.const 1))
     (block $B51
+      (call $ethereum.useGas (i64.const 1))
       (block $B52
+        (call $ethereum.useGas (i64.const 1))
         (block $B53
+          (call $ethereum.useGas (i64.const 1))
           (block $B54
+            (call $ethereum.useGas (i64.const 1))
             (block $B55
+              (call $ethereum.useGas (i64.const 1))
               (block $B56
+                (call $ethereum.useGas (i64.const 1))
                 (block $B57
+                  (call $ethereum.useGas (i64.const 1))
                   (block $B58
+                    (call $ethereum.useGas (i64.const 1))
                     (block $B59
+                      (call $ethereum.useGas (i64.const 1))
                       ; .....
                       ; on and on for 386 wasm blocks
 (loop $L49
+  (call $ethereum.useGas (i64.const 387))
   (block $B50
     (block $B51
       (block $B52
         (block $B53
           (block $B54
             (block $B55
               (block $B56
                 (block $B57
                   (block $B58
                     (block $B59
                     ; 386 blocks total...
```

(Some) Proposed EIPs

Account versioning

- Needed for ewasm
- Talked within the team for a while
- Discussions with the Go Ethereum team
- Great work by Wei Tang, in discussion with him

Init code vs. deploy code

- Deploying contracts is a two-step process
- Split the initialisation/deployment code in the transaction
- Key benefits:
    - Simplification
    - Verification
    - Metering
- (EIP to be pushed soon)

Sane limits of (some) properties

- Certain parameters have no explicit limits
  - block gas limit
  - timestamp
  - (buffer/memory) sizes
- Certain parameters have implicit limits only
  - gas

Sane limits of (some) properties (cont.)

- Some benefits:
  - EVM optimisations
  - Better design on non-EVM machines
- Clients already do some of these optimisations
  - (unlikely) consensus risk!
- (EIP to be pushed soon)

Unlimited SWAPn/DUPn

- EIP-663
- Benefits:
  - Can address the entire stack
  - Removes the “Stack too deep” error from Solidity □
  - A good first step to EIP-615
- Still need to decide on encoding the value
  - PUSH before SWAPn/DUPn
  - Immediate value after the opcode (Fixed 8 or 16-bit? Two opcodes?)
- After EIP-615:
  - Changed to only access the local stack frame

Restricted address range for system contracts/precompiles

- EIP-1352
- Precompiles are at a specific address
- Benefits:
  - Reduces consensus risk (very low)
  - Simplifies other opcodes (which have exceptions for “precompiles”)

Eth2.0 Update

Eth 2.0 focus points

- Shard state size
  - Bytecode size
  - Code reuse (merklization of code)
  - “Contract linking”
- Cross-shard communication
- More opportunity for (radical) changes
  - Trying to share most of the results with Eth 1.x

Updates on tooling (languages)

Tools

- Sentinel
  - *Wasm verification and metering*
  - Compiled as system contract on Ewasm
  - github.com/ewasm/sentinel-rs
- Chisel
  - *Wasm toolkit for transforming, optimising and verifying binaries*
  - github.com/wasmx/wasm-chisel

Solidity

- Yul: *(intermediate) assembly language*
- Solidity to Yul
  - Second prototype merged
- Yul to Wasm
  - Second prototype being worked on

EVM support

- RunEVM/RuneVM
  - *EVM interpreter in Rust compiled to wasm*
  - github.com/axic/runevm
- yevm
  - *EVM to WebAssembly compiler written in Rust*
  - github.com/axic/yevm
  - github.com/axic/solc-rust
  - github.com/axic/yultsur

Rust support

- ewasm_api
  - *Low/Mid-level Ewasm API for Rust*
  - github.com/ewasm/ewasm-rust-api
- ewasm-precompiles
  - github.com/ewasm/ewasm-precompiles
  - Implementation of all Byzantium precompiles + blake2, bls12, ed25519, sha1

Current focus

Current focus

- Metering
- Compiler engines
- EIPs
- State repricing prototype
- EVMC + EVM

Thanks!

- https://github.com/ewasm/design
- https://gitter.im/ewasm/Lobby