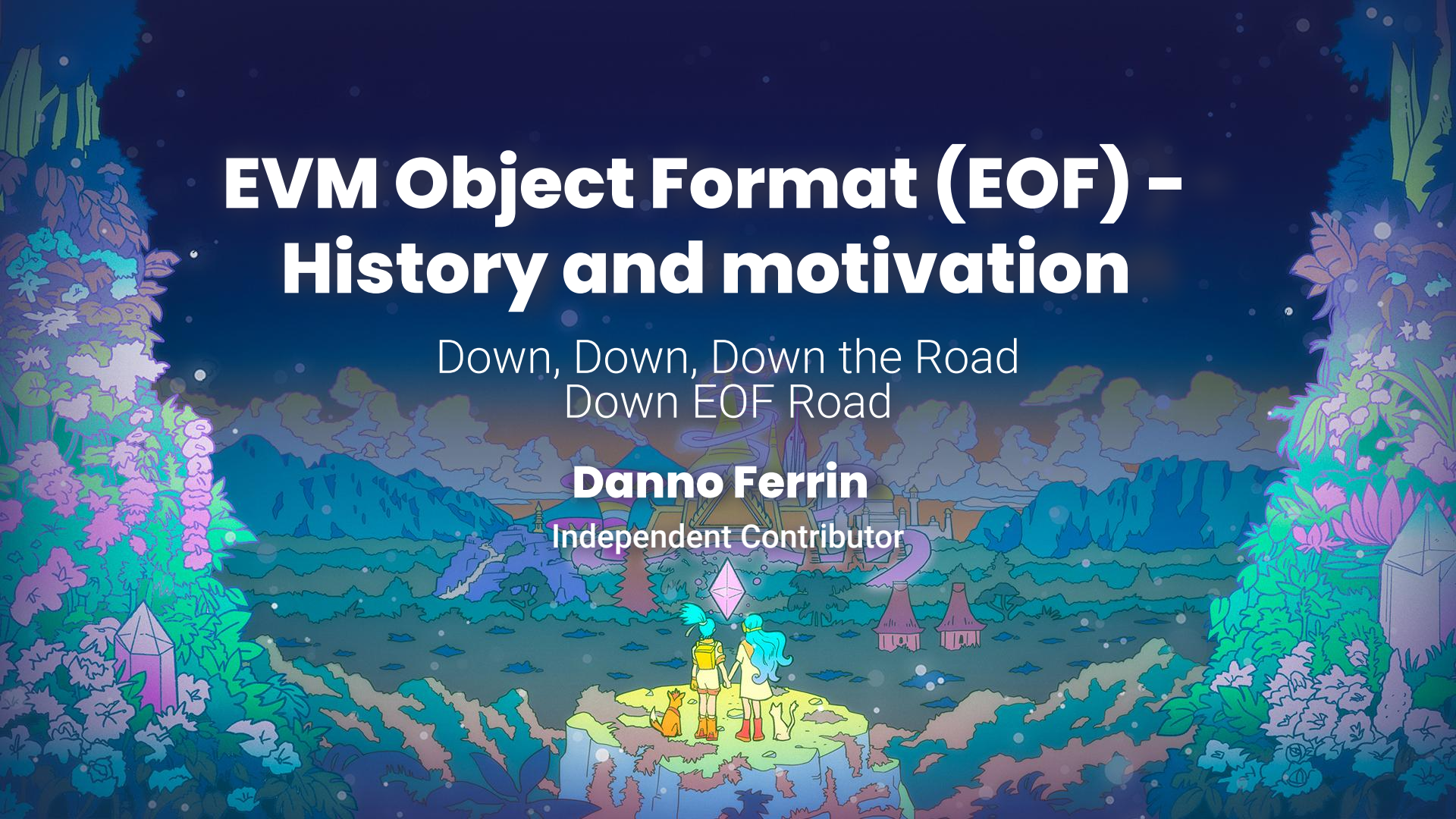
EVM Object Format (EOF) - History and motivation

Down, Down, Down the Road
Down EOF Road

Danno Ferrin

Independent Contributor



Section 1

Building a Legacy Prototypes to EIP-615

Before My Time...

Legend is the EVM was implemented over a weekend by two engineers

A C++ Client Engineer requested JUMPDEST be added to make LLVM compilations easier

The EVM was “good enough” and proved key concepts early.

A Few Key Misses and Late Additions

- CALLCODE didn't work as intended
  - EIP-7: DELEGATECALL
- Gas schedule was an attack vector
  - EIP-150: Raise cost of I/O heavy ops
  - EIP-158/160/161: State clearing
  - EIP-170: Contract size limit
- Bit math was painful
  - EIP-145: SHL / SHR / SAR - Bit shifting
- Smart contract ergonomic improvements
  - EIP-140: REVERT opcode
  - EIP-211: RETURNDATA SIZE / COPY
  - EIP-214: STATICCALL
- Precompiles
  - EIP-196: alt_bn128 add
  - EIP-197: alt_bn128 pairing
  - EIP-198: Mod Exp for big integers

EIP-615: Subroutines and Static Jumps

“the design of the EVM makes near-linear-time compilation to machine code difficult.”

“We also propose to validate—in near-linear time—that EVM contracts correctly use subroutines, avoid misuse of the stack, and meet other safety conditions before placing them on the blockchain.”

“well-behaved control flow and use of the stack makes life easier for interpreters, compilers, formal analysis, and other tools.”

- Adds static jumps
  - Target is in the code, not on stack
- Adds vector jump
- Adds subroutine jumps and returns
- Adds local variables
- Adds BEGINDATA marker
- Adds “magic” header
- Adds required validation

Sidequest: eWASM and Ethereum 2.0

Serenity is...

- A realization of Casper (pure proof of stake), sharding, EWASM, and other ideas from protocol research between 2014-2018

A Captain's job was simple:

Find a Crew. Get a Job. Keep Flying

Expected Phases

- Phase 0: beacon chain PoS
- Phase 1: shards as data chains
- Phase 2: enable strate transitions (EWASM)
- Phase 3 and beyond: Iterate, improve, add tech

Why was eWasm Abandoned?

- Requires replacing all existing EVM contracts
- Larger bytecode
- Slower execution
- Roadmap Complexity
- “Gas Weaving”
- The Merge

Section 2

The Technical Issues With Legacy EVM Code

Why don’t we do EOF / EIP-615 piecemeal?

Add Static Jumps

Add Code Validation

Enforce Code/Data Separation

Add Simple Subroutines

Drop features: Stack Validation,
Blocking Code and Gas Introspection

Why Not Immediates? A validation story.

0xE0 - RJUMP

“Immediate” bytes forms an int16 (Like PUSH2)

E00080 - Jump forward 128 bytes

E0FF80 - Jump backward 128 bytes

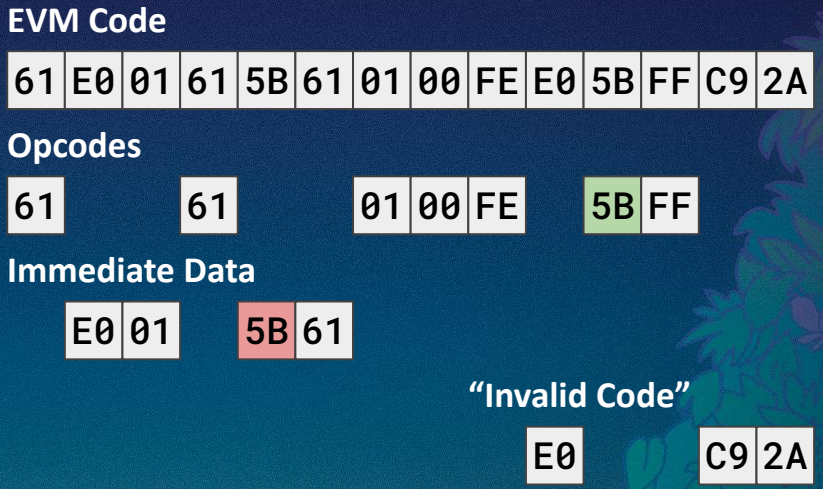
61018065 - PC=0x100 Legacy forward

608065- PC=0x100 Legacy backward

Why Immediates? A validation story.

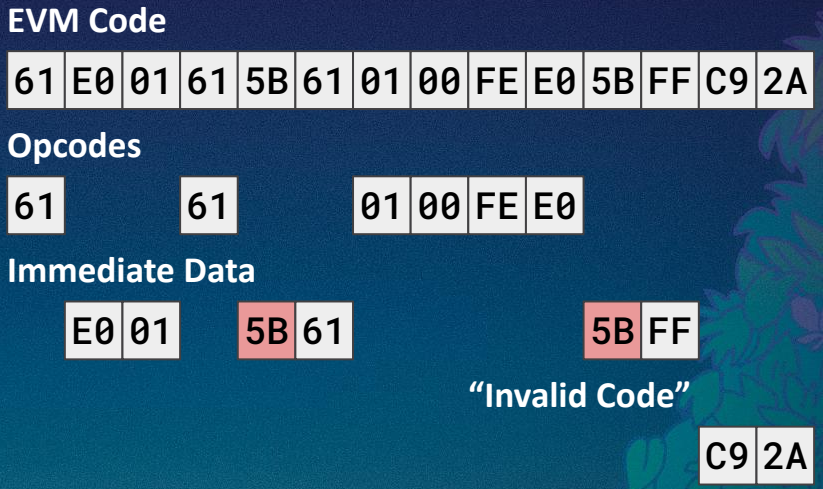
JUMPDEST instructions cannot be in “immediate data”

i.e. the contents of PUSH instructions



Why Immediates? A validation story.

Adding opcodes with new immediate data changes the interpretation of existing code



Just validate all the code

Compiler Nightmare Code: Vector jumps from Calldata

```
60003561FF1656

PUSH1 0x00
CALLDATALOAD
PUSH2 0xFFFF
AND
JUMP
```

Calldata determines where the PC jump destination is.

Just version the contracts

EIP-1702: Generalized Account Versioning Scheme

- Was Initially in Istanbul release

- Required new behavior on new accounts / contracts
- Preserved old behavior in old accounts
- Would have kept all current and old behaviors forever
- Also included facilities to introduce new account fields

- Removed from Istanbul because of the utility:complexity ratio

EIP-2315 Simple Subroutines

- Skip the immediate value drama, just add subroutines!
- Was added to Berlin
- Implemented in all clients too!
- Removed at the last moment as Solidity wouldn't use it

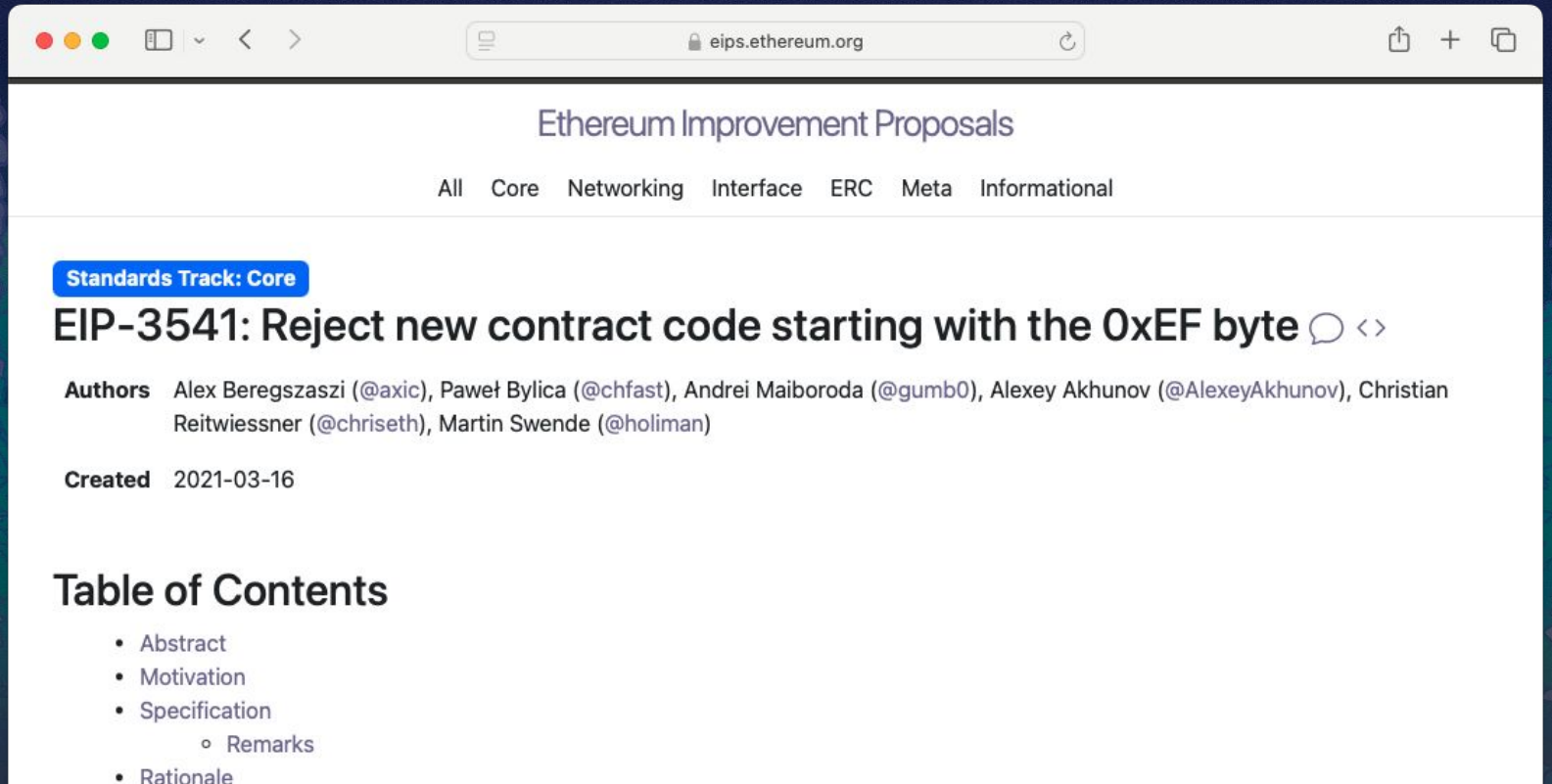
`JUMPSUB`

Transfers control to a subroutine.

1. Pop the `location` off the `data stack`.
2. If the opcode at `location` is not a `BEGINSUB` _`abort`_.
3. If the `return stack` already has `1024` items _`abort`_.
4. Push the current `pc + 1` to the `return stack`.
5. Set `pc` to `location + 1`.

* _pops one item off the `data stack`_
* _pushes one item on the `return stack`_

One Minor Change in London



The Merge

All EVM Proposals were paused

Section 3

Little and Big EOF

A container format for all the code and all the data in a smart contract

EIP-3550: EOF - EVM Object Format V1

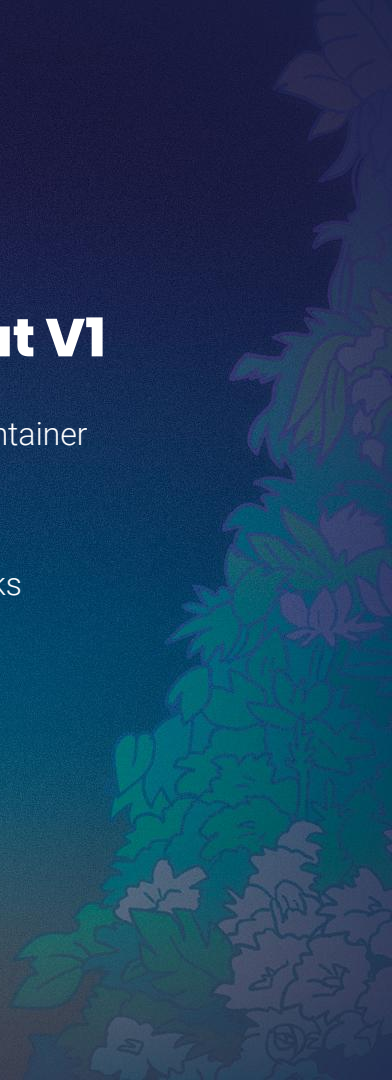
Other major JIT VMs have a full container

Java, .NET, LLVM

Full container avoids truncation risks

Versionable and extensible

Full code and data separation



EIP-3670: EOF - Code Validation

Prevents unallocated opcodes from appearing

Bans some older obsolete and problematic opcodes

Requires all operands with immedates to be complete

Makes future new opcodes easier to deploy

Only valid programs need apply

Fixing two problems

- **Dynamic Targets**
- **Absolute Targets**

EIP-4200: EOF - Static relative jumps

Adds JUMP and JUMPI to the banned opcodes list

Two bytes of range: up to +/- 32k range for jump

Relative targets makes code easily relocatable

EIP-4750: EOF - Functions

Splits code into multiple sections

Call and return from sections within the same frame

Metadata in container header for stack inputs, outputs, and maximum stack height of the function

No “dead” code sections: all must be reachable

Simple subroutines by another name

Validate the stack once
Validate the stack early

EIP-5450: EOF - Stack Validation

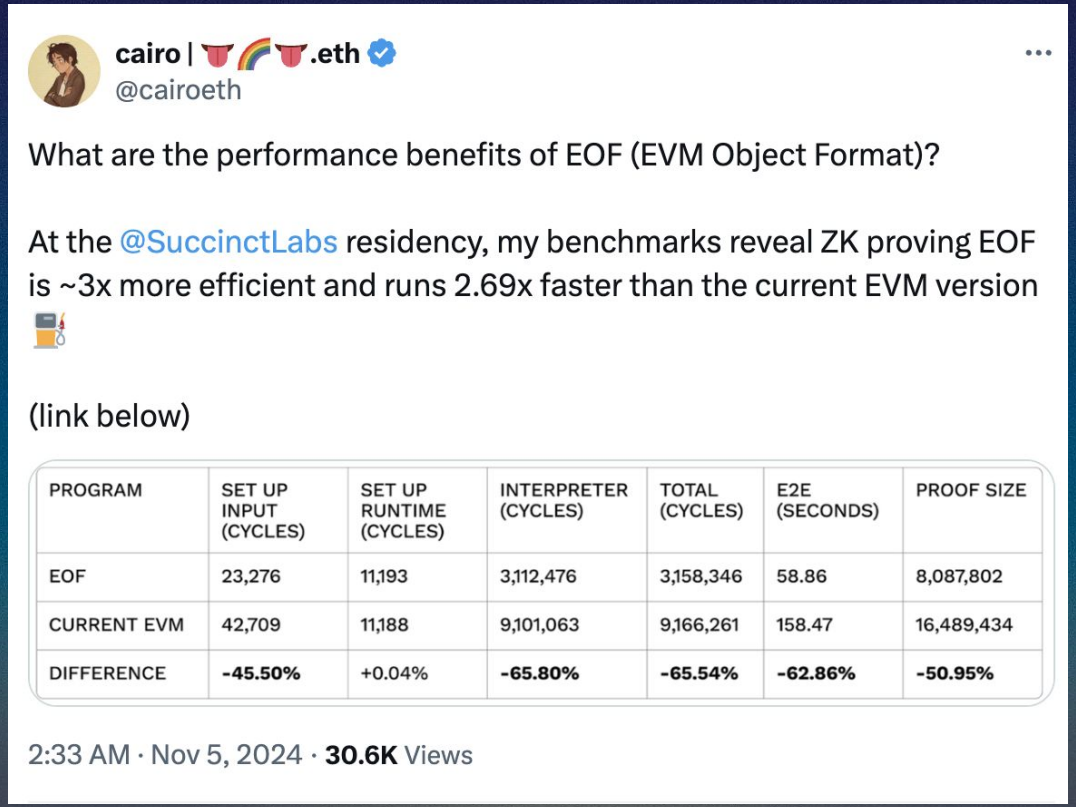
Rules about stack height aimed at preventing runtime overflow and runtime underflows

Requirement for forward referenced code permits single-pass verification

Relaxed in some instances to allow multiple heights targeting some code paths (such as REVERTs and RETURNs)

Simplifies stack/register conversions in compilation

Stack Validation Improves zk Proving



cairo | 👅🌈👅.eth
@cairoeth

What are the performance benefits of EOF (EVM Object Format)?

At the @SuccinctLabs residency, my benchmarks reveal ZK proving EOF is ~3x more efficient and runs 2.69x faster than the current EVM version ⛽

(link below)

PROGRAM	SET UP INPUT (CYCLES)	SET UP RUNTIME (CYCLES)	INTERPRETER (CYCLES)	TOTAL (CYCLES)	E2E (SECONDS)	PROOF SIZE
EOF	23,276	11,193	3,112,476	3,158,346	58.86	8,087,802
CURRENT EVM	42,709	11,188	9,101,063	9,166,261	158.47	16,489,434
DIFFERENCE	**-45.50%**	+0.04%	**-65.80%**	**-65.54%**	**-62.86%**	**-50.95%**

2:33 AM · Nov 5, 2024 · **30.6K** Views

Shanghai and Austria Interop

Two problems prevented “Big EOF” from shipping in Shanghai

- Solidity Constructor Customizations
- Code Introspection Concerns
  - “EOF proposal: ban code introspection of EOF accounts” (Eth Magicians Forum, December 2022)

If we are breaking backwards compatibility,
let’s do it once and break all the things we want to.

Section 4

Mega EOF

EIP–7620:
EOF Contract Creation

Depricates all old CREATE paths in EOF

Adds sub-containers to the EOF container:

- Entire EOF contracts within the EOF container
- All sub-containers are deeply validated

EOF create deploys those sub-container container

- Extra data may be appended to the data section

Most code introspection issues are contract creation issues

How EOF contracts are introduced into the blockchain

EIP-7698: EOF - Creation transaction

Replacement for proposed TXCREATE opcode

Leverages existing contract creation patterns

- Load the contract into the transaction
- Copy constructor args after the contract

EOF contract validations only need to be performed when the EOF creation transaction is used, or when used in a genesis allocation

Validations are one-and-done for the life of the contract

EIP-7480: EOF – Data section access instructions

Replacements for CODESIZE and CODECOPY opcodes banned in the container EIP

Allows access to the data section in the same ways as CODESIZE and CODECOPY

DATALOADN provides an equally efficient way to access static and constant values like Solidity's rewriting methods in contract init code.

Accessing immutable values and contract payloads efficiently

Addressing Gas introspection and preserving space for Address Space Expansion

EIP-7069: Revamped CALL instructions

EXTCALL, EXTDELEGATECALL, EXTSTAICCALL opcodes

Just like call series, but

- No Gas (all is passed in)
- No output memory (use return buffer)

GAS opcode is banned in EIP-3540

Call address is not 160-bit trimmed

BALANCE still trims

EIP-663: SWAPN, DUPN and EXCHANGE instructions

Increases the size of SWAP and DUP from 16 to 256

- Stack depth is a one-byte immediate

Introduces EXCHANGE instruction

- Swaps an item up to 17 items deep with another item up to 16 stack entries deeper

Added to EOF at the request of compiler teams

Helping compilers avoid “Stack too deep” errors

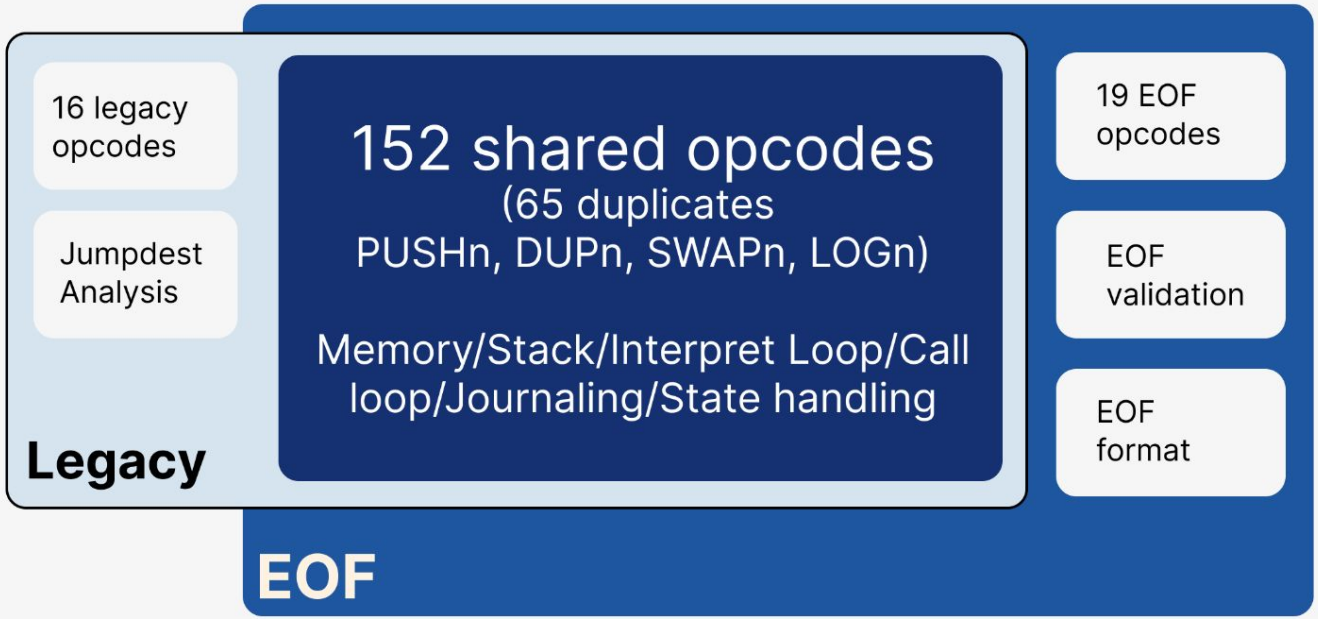
Deferred and Removed EOF Features

To reduce “surface area” of testing a number of features were removed from EOF in Osaka.

These four features may arrive in future hard forks, as they do not break any backwards compatibility with EOF v1.

- Remove TXCREATE
  - Removed to avoid another TX type
- Variable Length Quantities
  - Useful for contracts larger than 32k/64k
  - Current code size limit is 24k/48k
- HASCODE / Legacy Introspection
  - Under discussion for Osaka
  - EIP-721/EIP-1155 needs to detect EOAs
  - EOF Proxies need to detect EOF
- EXTDATACOPY / EXTDATASIZE
  - Contracts can share data via functions

EVM (EOF) taxonomy



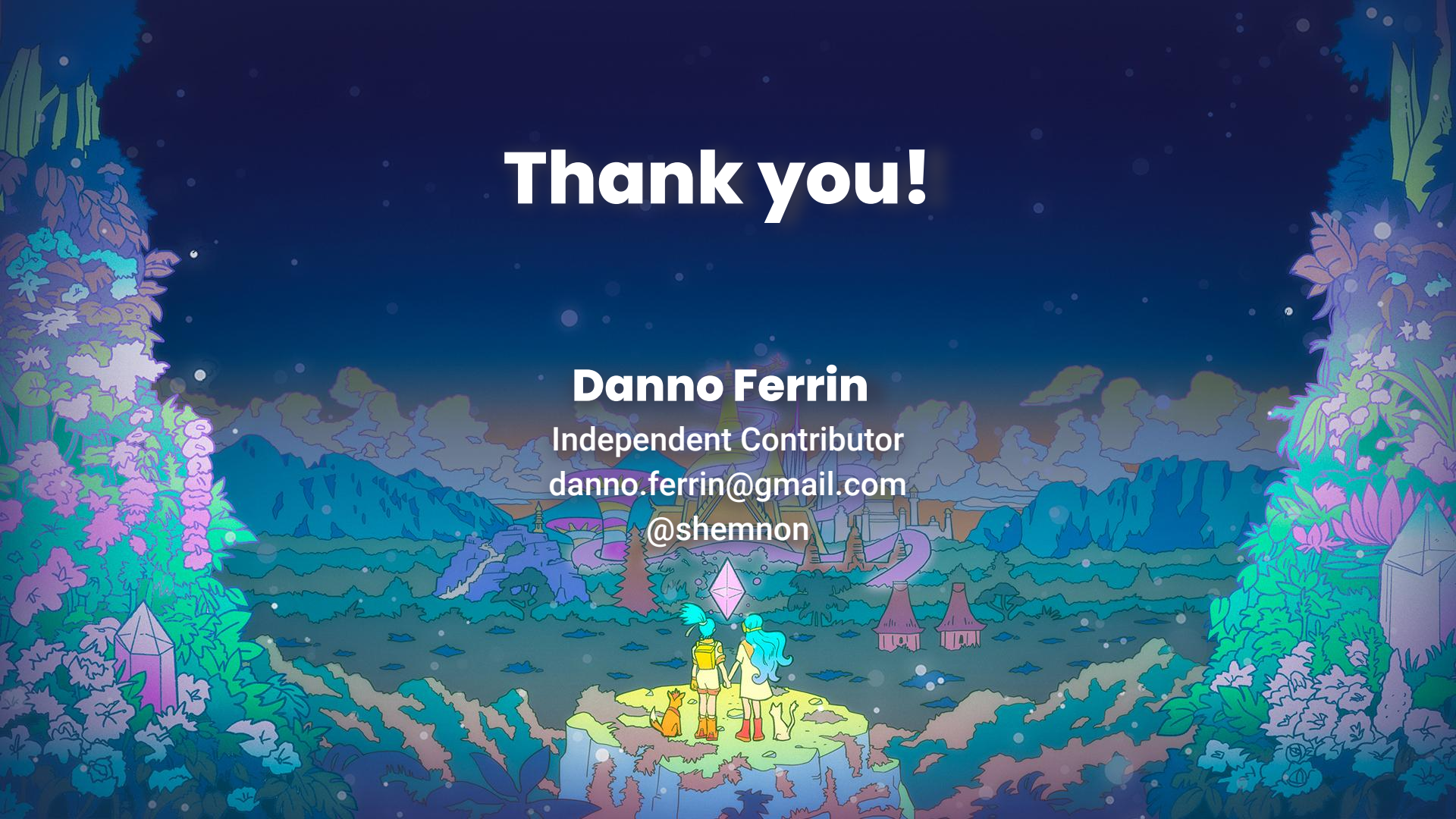
@rakitadragan

https://x.com/rakitadragan/status/1825174503548178843

EOF is the proper solution for these issues.

So I would say EOF is in that sense for us and for the stack below us the superior design by orders of magnitude.

Daniel, Solidity Tech Lead - ACDE#192


Thank you!
Danno Ferrin
Independent Contributor
danno.ferrin@gmail.com
@shemnon