18 NOVEMBER 2025

Solidity Summit

{ solx }

An LLVM-based Solidity Toolchain

Phase 1 - Multi-target Support

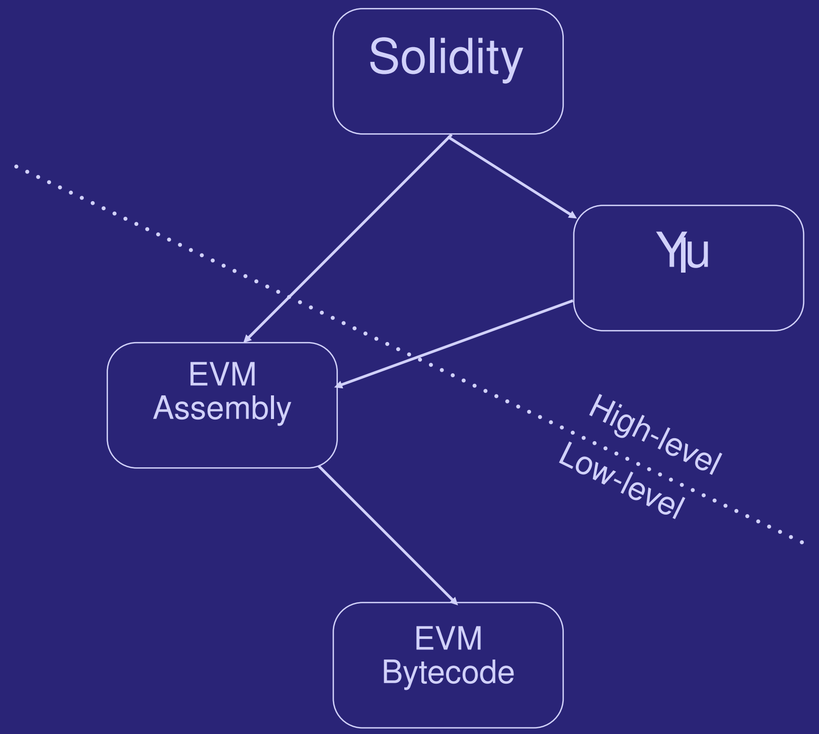
Goal • Problem • Solution • Result

Scope	Solidity compiler for ZKsync VM
Timeframe	~12 months
Headcount	1 Front-End Engineer 1 LLVM Engineer1 Engineer in Test
Baseline	ZKsync VM 25% Ready

Phase 1 - Multi-target Support

Goal • **Problem** • Solution • Result

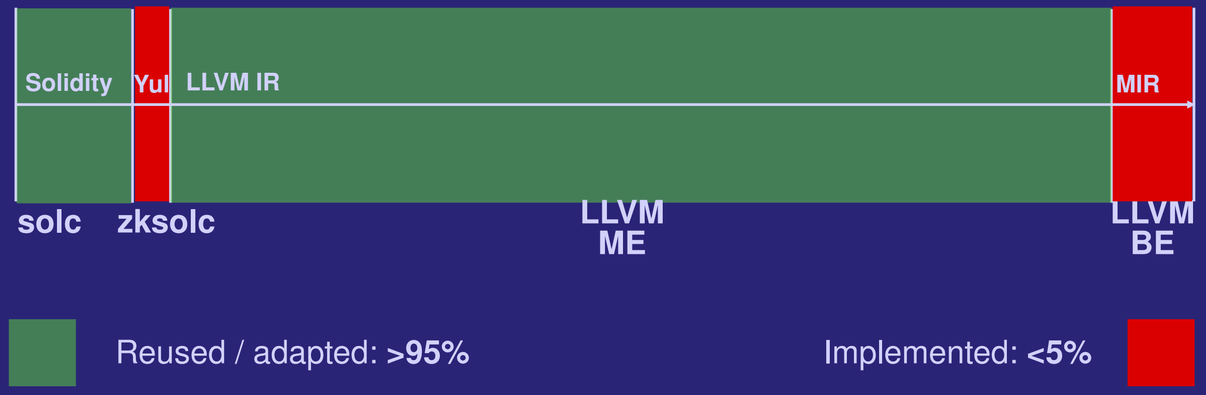
- Only one compiler available: **solc**
- Limited multi-target support
- EVM-centric pipelines and IRs



Phase 1 - Multi-target Support

Goal • Problem • **Solution** • Result

Don’t Implement. Reuse!



Phase 1 - Multi-target Support

Goal • Problem • Solution • **Result**

Great	Not so great
Solidity v0.8 on ZKsync VM	Solidity v0.8 only
✅ solc/Yul ported to LLVM*	⚠ Semantic differences with L1
	💀 Compiler crashes
	⏳ Long compilation time

*Yul to RISC-V for PolkaVM
https://github.com/paritytech/revive

Phase 2 - EVM Assembly Support

Goal • Problem • Solution • Result

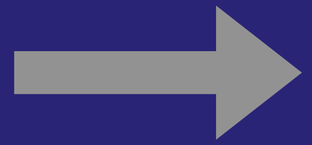
Scope	Solidity <0.8 via EVM Assembly
Timeframe	~6 months
Headcount	1 Front-End Engineer 1 LLVM Engineer1 Engineer in Test
Baseline	Full Yul pipeline in place

Phase 2 - EVM Assembly Support

Goal • **Problem** • Solution • Result

- EVM Assembly is stack-based - LLVM IR is register-based
- Obfuscated control flow & call graph

```
Tag 3
    JUMPDEST
    PUSH [tag]          5
    PUSH [tag]          6
    JUMP
Tag 4
    JUMPDEST
    PUSH [tag]          9
    PUSH [tag]          10
    JUMP
Tag 5
    JUMPDEST
    PUSH                40
    MLOAD
    PUSH [tag]          7
    SWAP2
    SWAP1
    PUSH [tag]          8
    JUMP
```



```
tag3:
  br label %tag6

tag5:
  %addr = getelementptr i256, i256* %mem, i256 40
  %v    = load i256, i256* %addr
  br label %tag8
```

Phase 2 - EVM Assembly Support

Goal • Problem • **Solution** • Result

- Registerization: mapping onto LLVM IR virtual registers
- Stack analysis to restore control flow and call graphs

```
Tag 3
    JUMPDEST
    PUSH [tag]          5
    PUSH [tag]          6
    JUMP
Tag 4
    JUMPDEST
    PUSH [tag]          9
    PUSH [tag]          10
    JUMP
Tag 5
    JUMPDEST
    PUSH                40
    MLOAD
    PUSH [tag]          7
    SWAP2
    SWAP1
    PUSH [tag]          8
    JUMP
```

```
block_rt_3/0: (predecessors: rt_1/0)
    JUMPDEST                    [ V_SHR ]
    PUSH [tag]          5       [ V_SHR ] + [ T_5 ]
    PUSH [tag]          6       [ V_SHR | T_5 ] + [ T_6 ]
    JUMP                        [ V_SHR | T_5 ] - [ T_6 ]
block_rt_4/0: (predecessors: rt_1/0)
    JUMPDEST                    [ V_SHR ]
    PUSH [tag]          9       [ V_SHR ] + [ T_9 ]
    PUSH [tag]          10      [ V_SHR | T_9 ] + [ T_10 ]
    JUMP                        [ V_SHR | T_9 ] - [ T_10 ]
block_rt_5/0: (predecessors: rt_13/0)
    JUMPDEST                    [ V_SHR | 2A ]
    PUSH                40      [ V_SHR | 2A ] + [ 40 ]
    MLOAD                       [ V_SHR | 2A ] - [ 40 ] + [ V_MLOAD ]
    PUSH [tag]          7       [ V_SHR | 2A | V_MLOAD ] + [ T_7 ]
    SWAP2                       [ V_SHR | T_7 | V_MLOAD | 2A ]
    SWAP1                       [ V_SHR | T_7 | 2A | V_MLOAD ]
    PUSH [tag]          8       [ V_SHR | T_7 | 2A | V_MLOAD ] + [ T_8 ]
    JUMP                        [ V_SHR | T_7 | 2A | V_MLOAD ] - [ T_8 ]
```

```
tag3:
  br label %tag6

tag5:
  %addr = getelementptr i256, i256* %mem, i256 40
  %v    = load i256, i256* %addr
  br label %tag8
```

2021 — 2022 Jan — 2023 — 2024 — 2025 — 2026

Phase 2 - EVM Assembly Support

Goal • Problem • Solution • **Result**

Great	Not so great
Solidity ≥0.4.12 on ZKsync VM	Extra optimizations needed
	solc fork required (~80 branches)
	Complex and fragile translation

Phase 3 - EVM

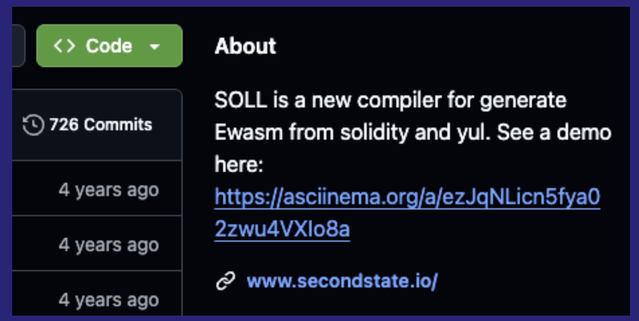
Goal • Problem • Solution • Result

Scope	**EVM Back End in LLVM**	**MLIR Codegen**
Timeframe	~18 months	? months
Headcount	1 LLVM Engineer	1 Front-End Engineer
Baseline	LLVM-based compiler for ZKsync VM	Solidity AST

Phase 3 - EVM

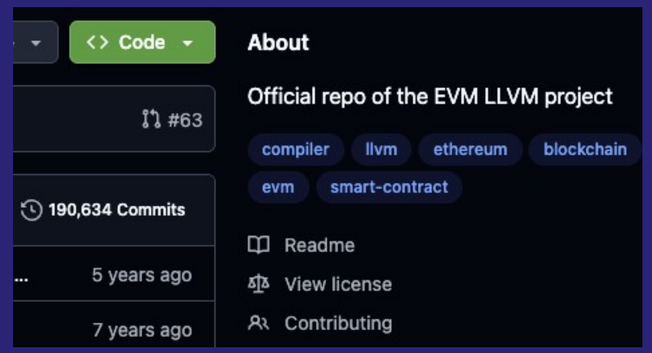
Goal • **Problem** • Solution • Result

SOL L



No EVM?

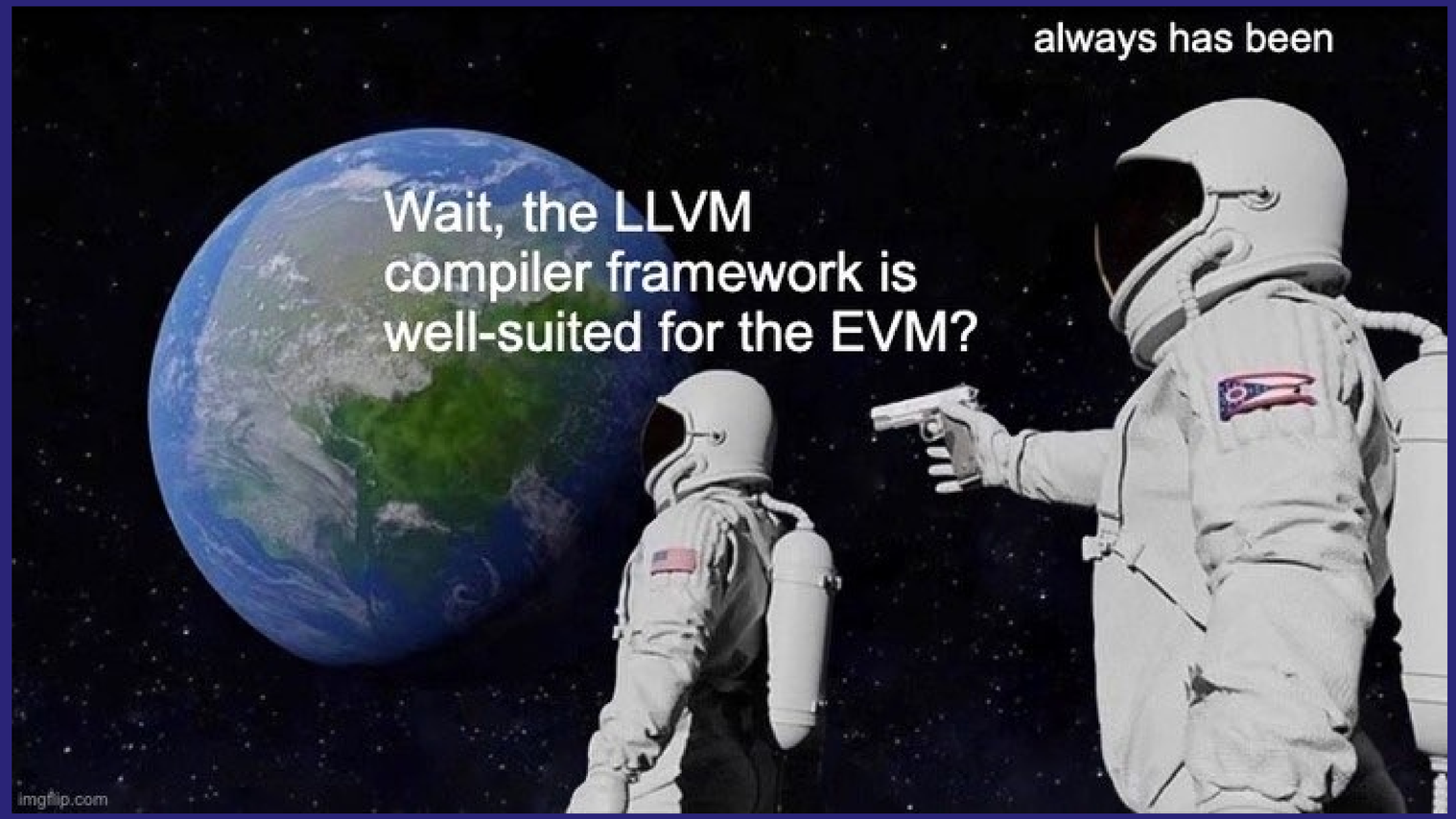
EVM LLVM



2021 2022 2023 May 2024 2025 2026

Phase 3 - EVM

Goal • Problem • **Solution** • Result



2021 2022 2023 May 2024 2025 2026

Phase 3 - EVM

Goal • Problem • **Solution** • Result

```
tag3:
  br label %tag6

tag5:
  %addr = getelementptr i256, i256* %mem, i256 40
  %v    = load i256, i256* %addr
  br label %tag8
```

stackification

```
; %bb.3:                                        ; %"block_rt_15/0.i"
        PUSH1           0x4
        CALLDATALOAD
        PUSH1           0x4
        CALLDATALOAD
        PUSH1           0xF
        SIGNEXTEND
        SUB
        PUSH4           @.BB0_6
        JUMPI
```

Inspired by:

WASM LLVM BE

Solc EVM BE

Phase 3 - EVM

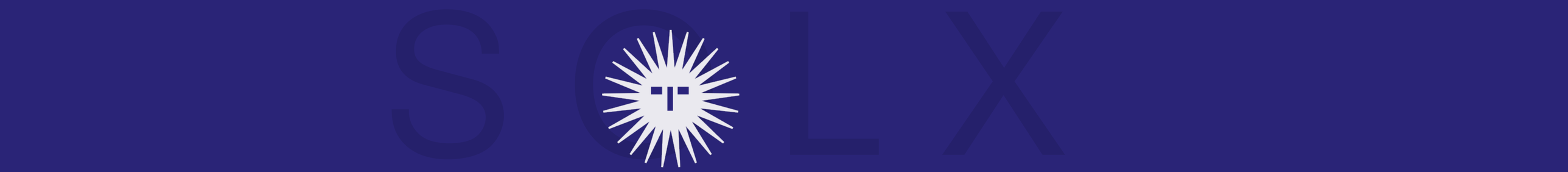
Goal • Problem • Solution • **Result**

Great

✅ EVM Target in LLVM

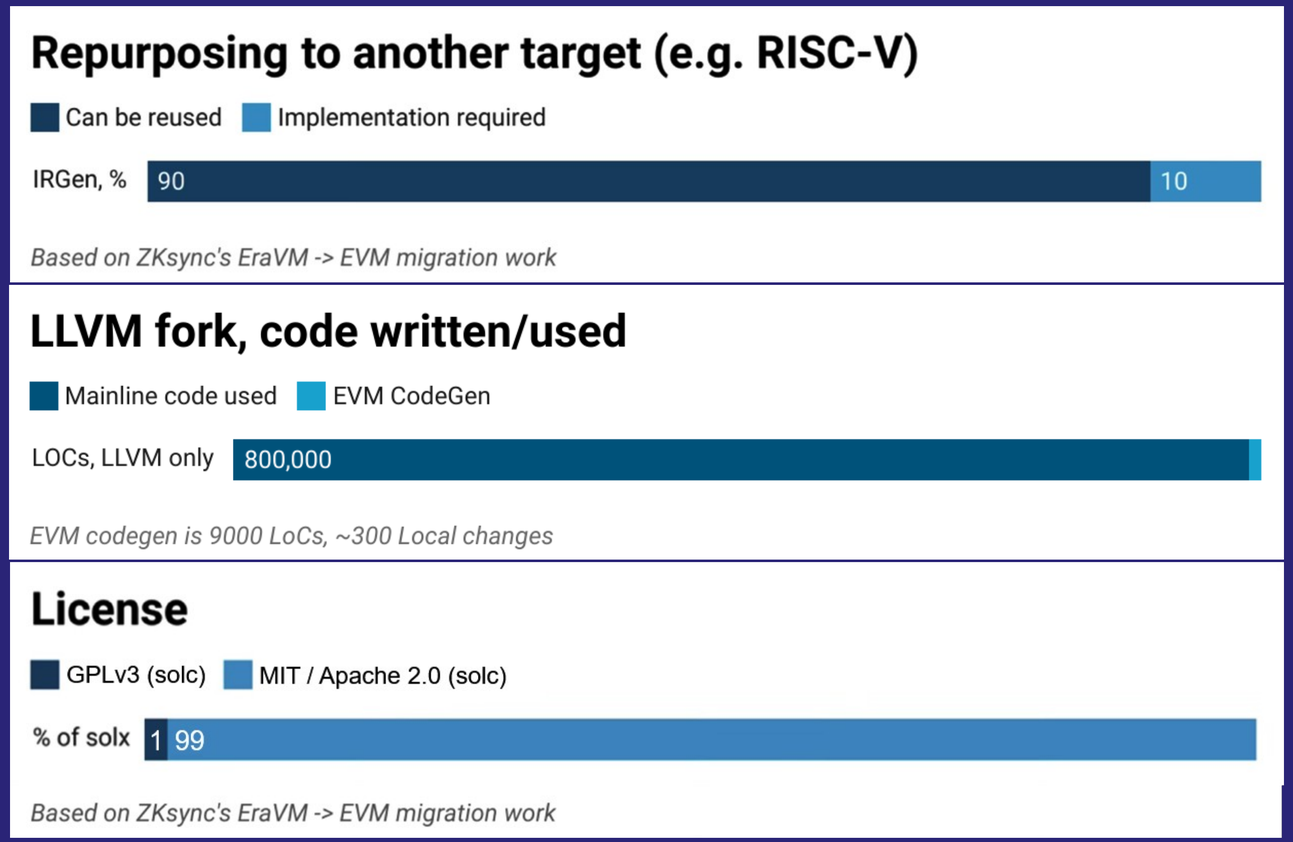
Not so great

❌ Can’t use it yet



Phase 3 - EVM

Goal • Problem • Solution • **Result**



2021 2022 2023 2024 Nov 2025 2026

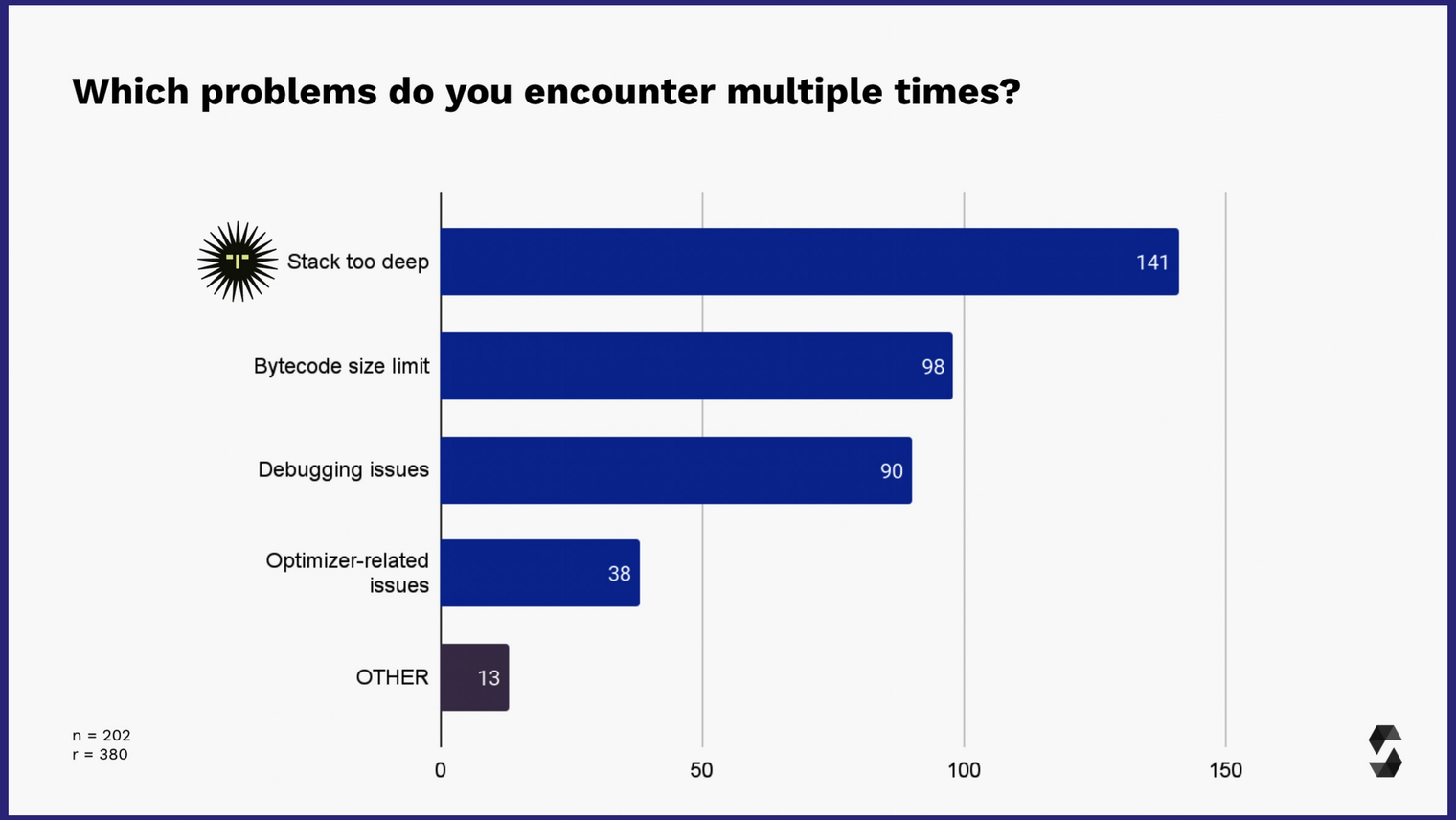
solx v0.1.0: stack-not-so-deep

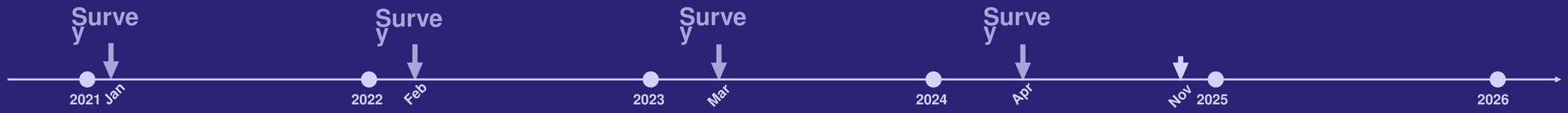
Goal • Problem • Solution • Result

Scope	**solx**: LLVM-based compiler for the EVM
Timeframe	~6 months
Headcount	1 Front-End Engineer 3 LLVM Engineers 1 Engineer in Test
Baseline	EVM Back End in LLVM

solx v0.1.0: stack-not-so-deep

Goal • **Problem** • Solution • Result





solx v0.1.0: stack-not-so-deep

Goal • Problem • **Solution** • Result

solx

- Tries the stackification once
- Records variables that don’t fit into the stack
- Allocates an extra heap area for spills
- Recompiles the translation unit

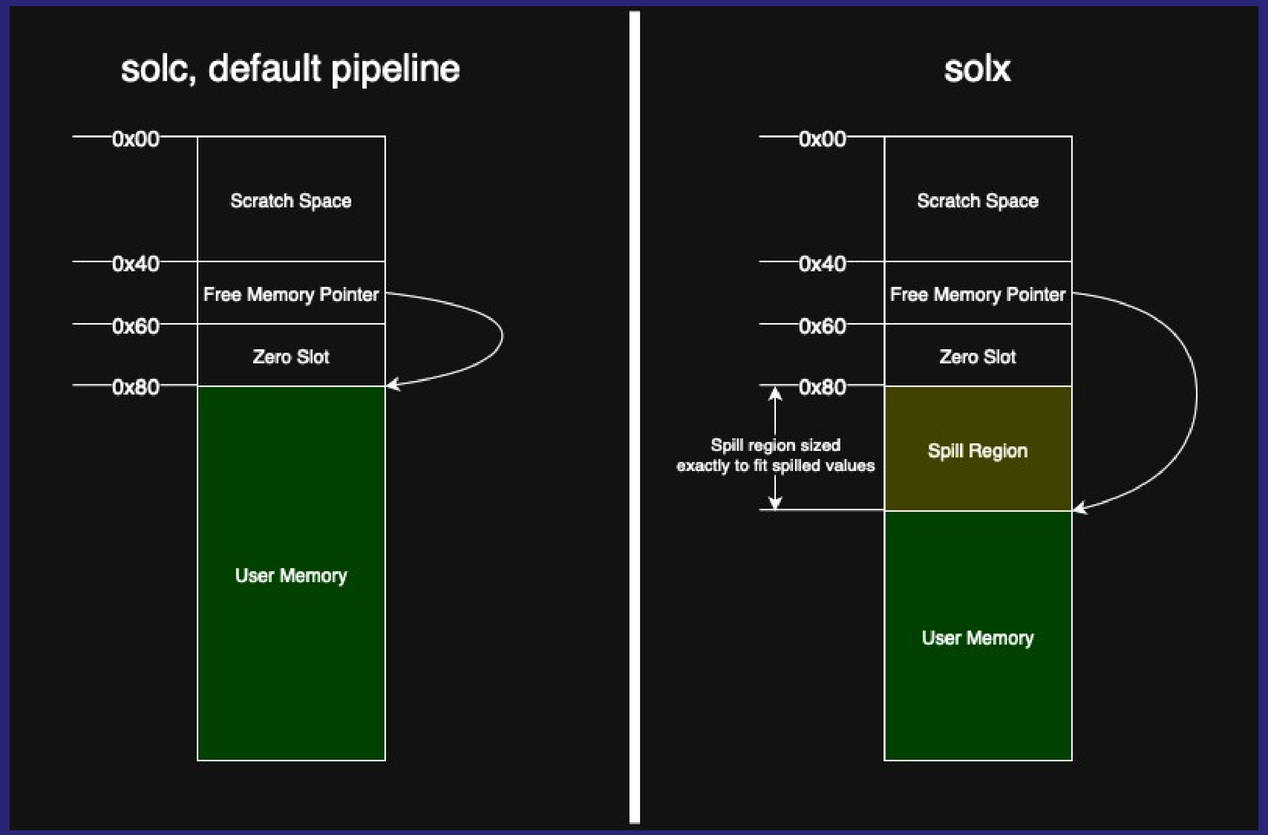
solc / Yul

- Predicts the stack pressure
- Offloads data from the stack
- Does not re-run the compilation

```
Uncaught exception:
./libyul/backends/evm/EVMObjectCompiler.cpp(106): Throw in function void solidity::yul::EVMObjectCompiler::
Dynamic exception type: boost::wrapexcept<solidity::yul::StackTooDeepError>
std::exception::what: Cannot swap Variable _2 with Variable dst: too deep in the stack by 1 slots in [ JUNK _2
memoryguard was present.
[solidity::util::tag_comment*] = Cannot swap Variable _2 with Variable dst: too deep in the stack by 1 slots in
memoryguard was present.
```

solx v0.1.0: stack-not-so-deep

Goal • Problem • **Solution** • Result



solx v0.1.0: stack-not-so-deep

Goal • Problem • Solution • **Result**

Great

✅ **solx v0.1.0**: stack-too-deep solved

🚀 Any codegen, no semantic changes

First LLVM-based EVM compiler

☺ Moderate gas usage reduction

Not so great

⌛ Long compilation times

⚠ Bytecode size issues

solx v0.1.1: Compilation Time

Goal • Problem • Solution • Result

Scope	Don’t be slower than **solc**
Timeframe	~1 month
Headcount	1 FE Engineer
Baseline	**solx** is up to 3x slower than **solc**

solx v0.1.1: Compilation Time

Goal • **Problem** • Solution • Result

1. **solc** is single-threaded
2. Both **solc** and LLVM optimizers enabled

solx v0.1.1: Compilation Time

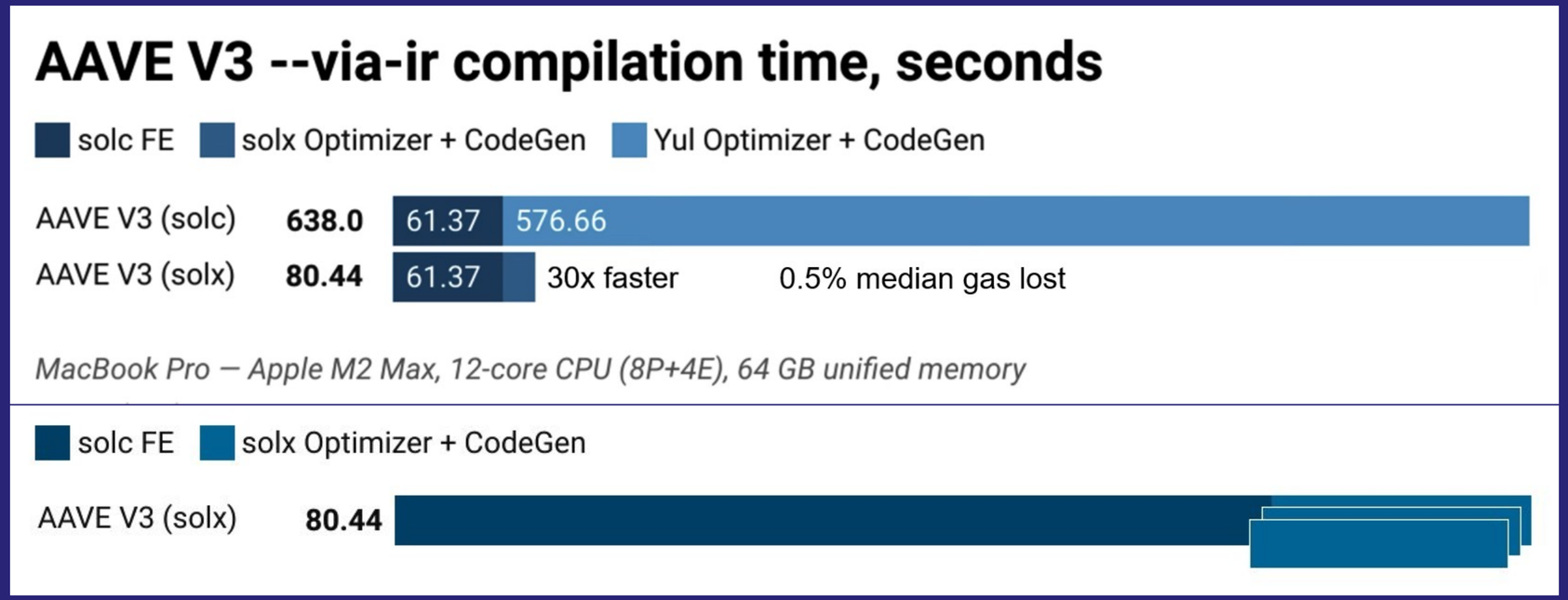
Goal • Problem • **Solution** • Result

❌ Add multi-threading to solc

✅ Turn off the solc optimizer

solx v0.1.1: Compilation Time

Goal • Problem • Solution • **Result**



2021 2022 2023 2024 2025 Jul 2026

solx v0.1.2: Bytecode Size

Goal • Problem • Solution • Result

Scope	Fit into the EVM bytecode size limit
Timeframe	~1 month
Headcount	1 LLVM Engineer
Baseline	Bytecode is **6%** larger (median)

solx v0.1.2: Bytecode Size

Goal • **Problem** • Solution • Result

- LLVM passes not tuned for EVM
- Strong preference for gas reduction

```
Binary:
5b608060405234601957630000000e480630000001e5f395ff35b5f80fdfe5b608060405234607a576003361115607a
57601f6004360313636eac1c0b60e11b63ffffffff60e01b5f3516141615607a576004356004356f0b03607a5760
04357ffffffffffffffffffffffffffffffff800000000000000000000000000000003607e57634e487b7160e01b
5f526011600452602245ffd5b5f80fd5b6004355f03600f0b60805260206080f3fea2646970667358221220 4f3d4f1e
13a85eff7ef16ecfea4359db2d01f519ca9b921c4fd7aeb97a03762564736f6c637821736f6c783a302e312e303b73
6f6c633a302e382e33303b6c6c766d3a312e302e320052
```

solx v0.1.2: Bytecode Size

Goal • Problem • **Solution** • Result

```
PUSH32 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

```
PUSH1 0x01
PUSH0
SUB
```

Constant unfolding

Adaptive gas/size tradeoff

solx
v0.1.2

Goal • Problem • Solution • **Result**

Bytecode size	solx 0.1.1	solx 0.1.1 --via-ir	solc 0.8.30	solc 0.8.30 --via-ir
% Better	95,5 %	98,5 %	89,0 %	43,5 %
% Worse	0,3 %	0,4 %	11,0 %	56,4 %
% No Change	4,2 %	1,1 %	0,0 %	0,1 %
Total (sum of all contract sizes)	-8.54%	-11.30%	-11.67%	-2.76%
Median	-6.13%	-13.10%	-28.80%	-21.96%

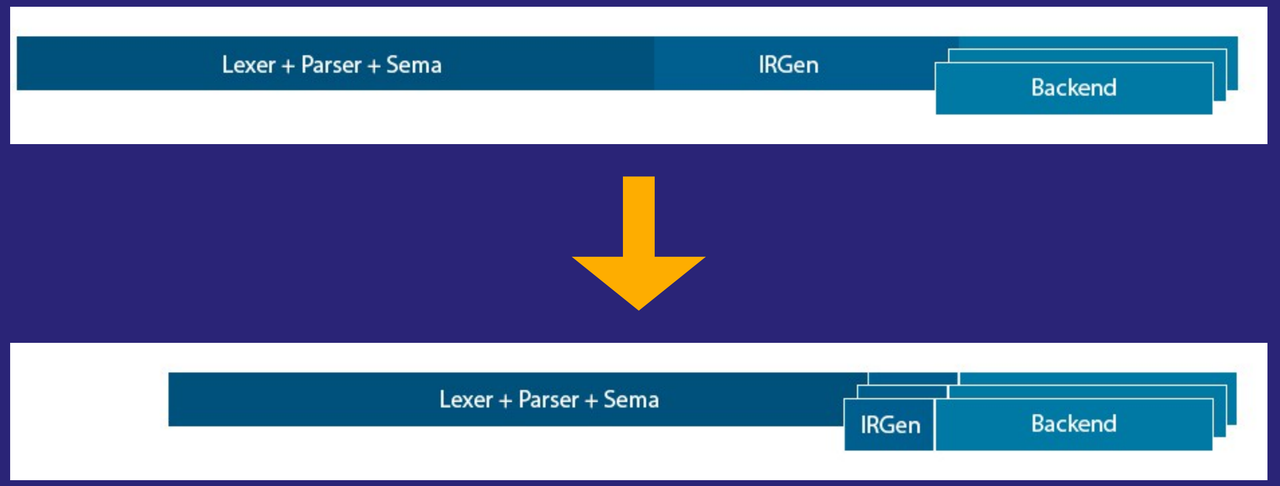
solx
v0.1.2

Goal • Problem • Solution • **Result**

Gas	solx 0.1.1	solx 0.1.1 --via-ir	solc 0.8.30	solc 0.8.30 --via-ir
% Better	16,9 %	37,3 %	94,2 %	89,4 %
% Worse	56,4 %	59,4 %	4,6 %	9,4 %
% No Change	26,8 %	3,4 %	1,3 %	1,1 %
Total gas spent by all tests	-0.35%	-0.50%	-1.80%	-1.37%
Median	-0.15%	-0.01%	-12.90%	-12.82%

Coming soon: MLIR Codegen

- Debuggers & Tools
- 3-6x compilation time



Testing & Security

✓ Integrated

- 24 Foundry projects
- 4 Hardhat projects
- Upstream Solidity tests
- ZKsync VM tests
- LLVM unit tests

WIP || Coming soon

- Fuzzing / bug mining
- >90% Code Coverage
- Contract Sanctuary
- Front-end & back-end audit

Wen solx?

✅ Recommended

- Fast development (compilation time)
- Stack-too-deep annihilation

Production

- Non-mission-critical projects
- Projects with ~100% test coverage

Share your experience

Website: https://solx.zksync.io/

GitHub: **NomicFoundation/solx**

Blog: https://zksync.mirror.xyz/

Telegram: **@solx_devs**

X: **solx_compiler**

Q&A

TRY
{ solx }
NOW