
Hello, Devcon!

I'm Vish!

Developer Advocate at Solidity,
Argot Collective.

🗣️External communications
👉Relaying feedback
📝Documentation
✨Special projects

Solidity:

Then, Now, & the Future!

Table of Contents

How it started

What the compiler used to look like...

Inside the Compiler

Compiler used to have code like

```
// stack: ref new_length current_length ref_value
solAssert(_context.stackHeight() - stackHeightStart == 4 - 2, "Stack");
_context << Instruction::POP << Instruction::DUP3;
CompilerUtils(_context).computeHashStatic();
_context << Instruction::DUP1 << Instruction::SLOAD << Instruction::SWAP1;
// stack: ref new_length current_length first_word data_location
_context << Instruction::DUP3;
ArrayUtils(_context).convertLengthToSize(_type);
```

devcon three

Ethereum Foundation Developers Conference

Nov 1-4, 2017 Cancún, Mexico

...gradually changing to

Inside the Compiler

This is gradually changing to

```
function <functionName>(value, pos) <return> {
  let length := <lengthFun>(value)
  <storeLength> // might update pos
  let headStart := pos
  let tail := add(pos, mul(length, 0x20))
  let srcPtr := <dataAreaFun>(value)
  for { let i := 0 } lt(i, length) { i := add(i, 1) }
  {
    mstore(pos, sub(tail, headStart))
    tail := <encodeToMemoryFun>(<arrayElementAccess>, tail)
    srcPtr := <nextArrayElement>(srcPtr)
    pos := add(pos, 0x20)
  }
  pos := tail
  <assignEnd>
}
```

Alex Beregszaszi talked about Iulia in depth already.

devcon three

Ethereum Foundation Developers Conference

Nov 1-4, 2017 Cancún, Mexico

How it’s going

Meet the Solidity Team

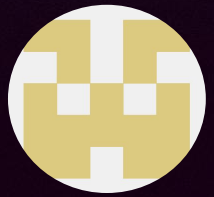
Development


Alex
(aarlt)


Daniel
(ekpyron)


Kamil
(cameel)

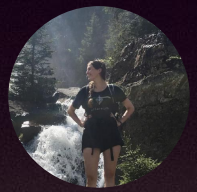

Matheus


Moritz
(clonker)


Nikola


Rodrigo S.
(r0qs)

Community & Ecosystem


Franzi


Vish

Meet the team

ethdebug

Nick
(gnidan)

SMTChecker

Martin
(blishko)

Fuzzing

Bhargava
(bshastry)

Research

Marcin

Rodrigo R.

Fe

Formal
Verification

Recent Language Features & Developments

Transient Storage

- Opcodes in Solidity compiler (0.8.24)
- Parser support for transient storage variables (0.8.27)
- Full support for value types (0.8.28)

Explicit Storage Layout

- Proposal to allow specifying storage locations
- Progress triggered by EIP-7702
- We shared a proposal for explicit storage layout syntax for feedback
- Candidate for implementation: Variant 2 in proposal
- Rationale & Spec under original issue

```
contract FinalContract
    is BaseA, BaseB(42), BaseC
    layout at erc7201("example.main")
{}
```

`require` with Custom Errors

- Gas efficient way to explain to the user why an operation failed
- New overload to support custom errors
- require(bool, error)
- Reverts with custom user-defined errors supplied as second argument
- Released in 0.8.26

```solidity
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.26;

/// Insufficient balance for transfer. Needed `required` but only
/// `available` available.
/// @param available balance available.
/// @param required requested amount to transfer.
error InsufficientBalance(uint256 available, uint256 required);

// This will only compile via IR
contract TestToken {
    mapping(address => uint) balance;
    function transferWithRequireError(address to, uint256 amount) public
        require(
            balance[msg.sender] >= amount,
            InsufficientBalance(balance[msg.sender], amount)
        );
        balance[msg.sender] -= amount;
        balance[to] += amount;
    }
    // ...
}
```


via-ir

It is a new compilation pipeline in Solidity that introduces an intermediate step by first translating the Solidity code into an intermediate representation (Yul) instead of directly compiling Solidity code into EVM bytecode.

Advantages of compiling via IR

- Allows for powerful optimisations
- Transparent & auditable code generation
- Better gas-efficiency
- Eliminating stack-too-deep errors

A more modern compiler design

Feasibility of multiple backends & frontends

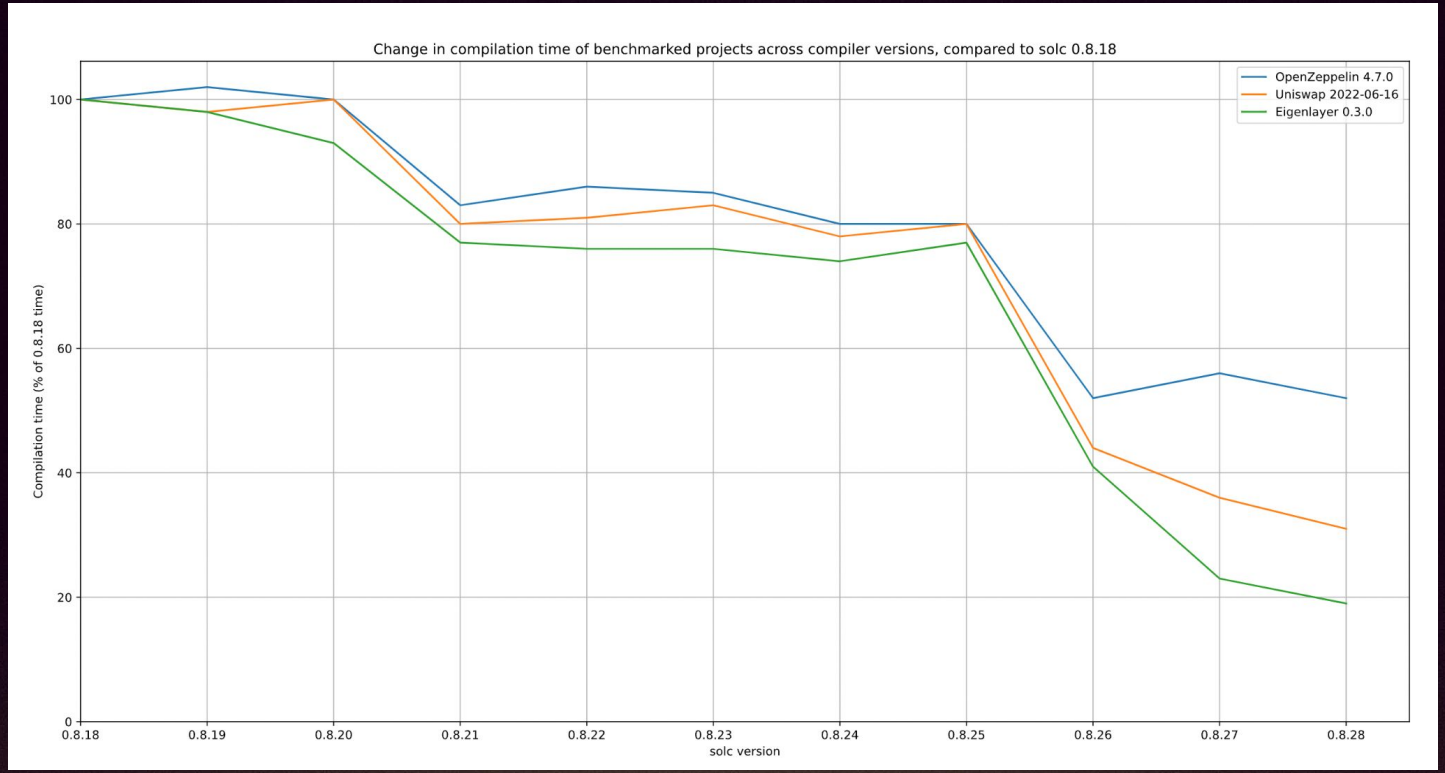
- Designed with the goal to serve as a backend for multiple compilers (e.g. Fe Lang).
- Targets multiple backends, initially EVM and EWASM and now EOF.
- Intended to be a backend for Experimental Solidity.

Recent Performance Optimizations

Making compilation via IR faster

- New, faster Yul optimiser sequence in 0.8.26
- Caching of optimized IR in 0.8.27
- Generating JSON artifacts of Yul ASTs only on demand in 0.8.28
- And various smaller changes…

Improvements in compilation times via IR



Compared to Solidity **v0.8.18,** compilation times via IR were reduced by

50%

for OpenZeppelin,

69%

for Uniswap, and…

...81%

for EigenLayer.

Ongoing work on compilation via IR

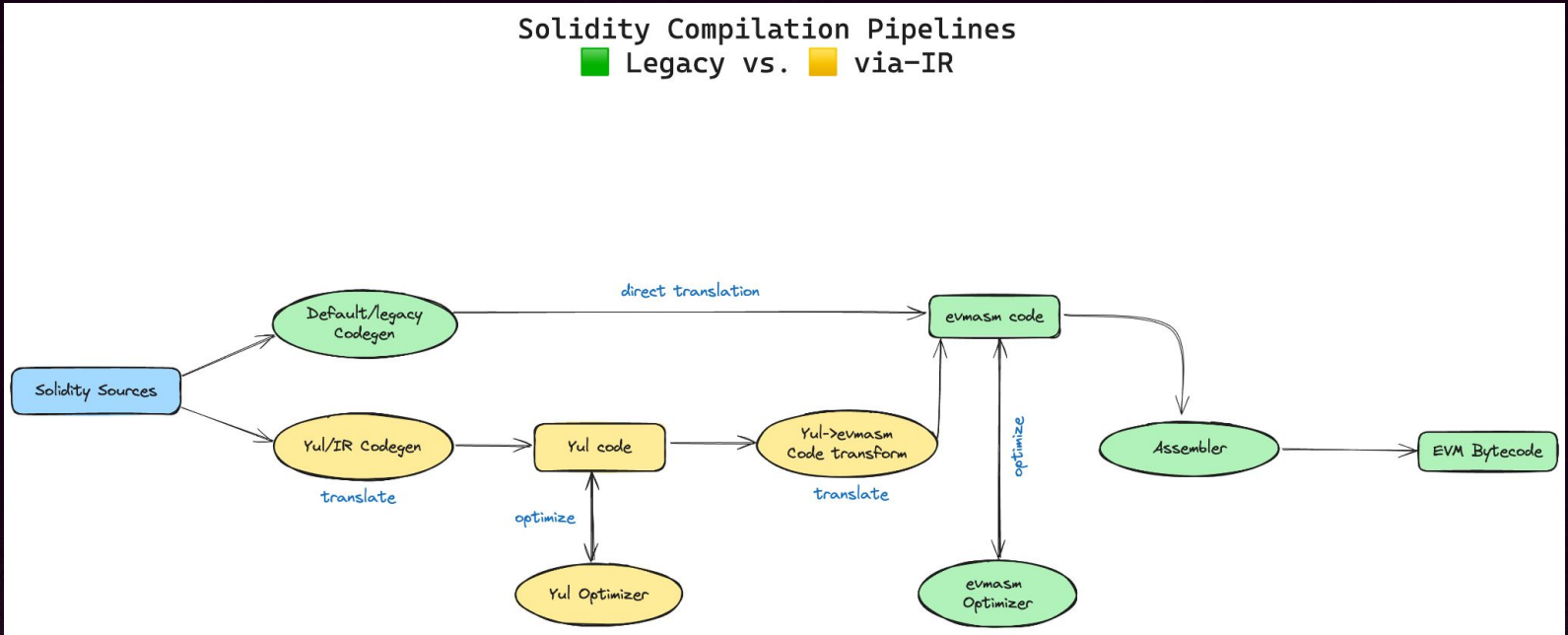
- ethdebug: better debugging for the IR pipeline
- Project GreY (University of Madrid): improve Yul IR -> EVM bytecode transformations
- SSA CFG optimisations to speed up compilation times

What's next? 🤔

`via-ir` 🤝 EOF

We wrote an explainer about it!

You can read it on our blog.🔗



Solidity in the Future

Current state of the language

- Compiler hard-coded built in functionality
- Inferior extensibility
- Introducing new features comes at a high cost and risk

What's missing?

- Generics
- Formally defined semantics
- Highly requested features by the user community

Path A

Current Solidity

- Extending the language with compiler-hardcoded features.
- Further increase language complexity.
- Fast paced in the short term; high risks.

Path forward

- Moving to a lean core language.
- Reduce language complexity by moving feature development to a standard library
- Making safe extensibility a first-class priority.
- Faster implementation of important features.

Path B

Path forward ✅

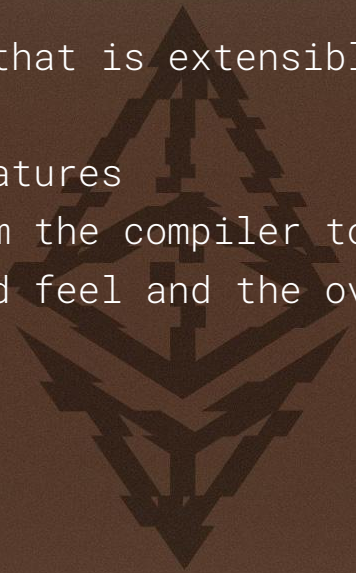
- Moving to a lean core language.
- Reduce language complexity by moving feature development to a standard library
- Making safe extensibility a first-class priority.
- Faster implementation of important features.
- Long-term future of Solidity.

Path B

Path B

Experimental Solidity

Experimental Solidity: Design Goals



- Simplified core language that is extensible
- Allows formal semantics
- No hard-coded language features
- Move features & types from the compiler to a Standard Library
- Maintain the same look and feel and the overall usability of the current language

Advantages

- Core language changes are rare and features can be added without changing the core language drastically
- Lean core language -> ease of formal verification and auditing -> Improved security
- Move faster by delivering community feature requests collectively
- Community stewardship over the standard library

Current timeline.

Ongoing

Existing prototypes that are able to reconstruct simple contracts.

2025

- A more complete baseline standard library.
- Fully functional prototypes.
- Initial syntactic sugar.
- Feedback on the above.

2026

- Stabilisation of the process.
- Moving from prototypes to production capacity.

Resources

- Blog: solidity.org/blog
- Docs: docs.solidity.org
- Forum: forum.solidity.org
- Follow us on Twitter @solidity_lang!
- Learn more about Argot Collective: argot.org

Thank you!

Vish

Developer Advocate, Solidity

vishwa.mehta@ethereum.org

@vishwamehta30