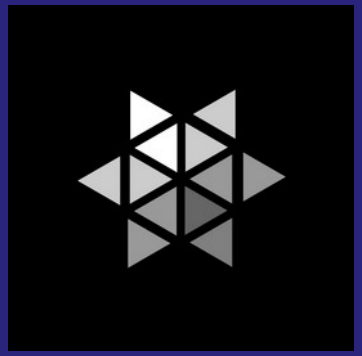
18 NOV 2025

# Solar Invictus

## A new Solidity compiler



# Foundry 100x’d devex

# But...

Limitations of the Solidity compiler soon surfaced, as developers started writing larger and more complex projects using only Solidity

| PROJECT | 0.8.26 | 0.8.27 | SPEEDUP |
|---|---|---|---|
| openzeppelin | 37 s | 37 s | 0% |
| uniswap-v4 | 225 s | 147 s | 42% |
| eigenlayer | 1211 s | 674 s | 44% |

```
[⠒] Solc 0.8.30 finished in 639.24ms
Error: Compiler run failed:
Error: Cannot swap Variable param_5 with Variable param_29: too deep in the stack by 8 slo
ts in [ RET[abi_encode_bytes32] RET[fun_trigger] param param_1 param_2 param_3 param_4 par
am_5 param_6 param_7 param_8 param_9 param_10 param_11 param_12 param_13 param_14 param_15
 param_16 param_17 param_18 param_19 param_20 param_21 param_22 param_23 param_24 param_25
 param_26 param_27 param_28 param_29 ]
memoryguard was present.
 --> test/Counter.t.sol:7:1:
  |
7 | contract CounterTest is Test {
  | ^ (Relevant source part starts here and spans across multiple lines).
```

# Workarounds

1. Smarter caching
2. Recompile as little as possible
3. Request minimum  from the compiler
4. Dynamic linking

# Highlight: dynamic test linking

Eliminates redundant compilation by pre-processing test contracts during the initial build

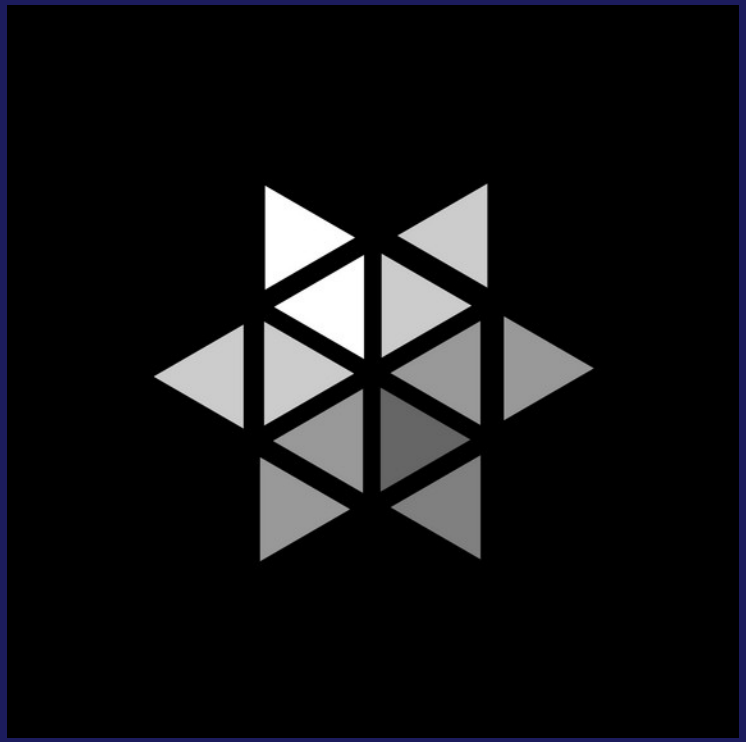
## up to 131x

Faster building and testing

| Project | Change | Files Compiled<br>with/without, after initial compile | Time to Compile<br>with/without, after initial compile |
|---|---|---|---|
| uniswap v4-core | add `Lock.lock()` at `PoolManager.sol#L107` | 1 / 19 | 2.25s / 165.13s |
| spark-psm | change `amountOut < minAmountOut` at `PSM3.sol#L125` | 3 / 28 | 2.14s / 16.15s |
| morpho-blue-bundlers | change `if(assets < 0)` at `MorphoBundler.sol#L106` | 11 / 36 | 16.39s / 251.05s |
| morpho-blue | add `require(assets !=0, ErrorsLib.ZERO_ASSETS)` at `Morpho.sol#L424` | 1 / 23 | 1.01s / 133.73s |
| sablier lockup | change `if(cliffTime < 0)` at `SablierLockup.sol#L480` | 1 / 104 | 781ms / 71.29s |
| solady | add additional `_setOwner(newOwner)` at `Ownable.sol#L182` | 9 / 14 | 6.17s / 6.34s |
| euler evc | change `SET_MAX_ELEMENTS = 11` at `Set.sol#L7` | 28 / 30 | 9.17s / 9.40s |

# Solar

https://www.paradigm.xyz/2024/11/solar

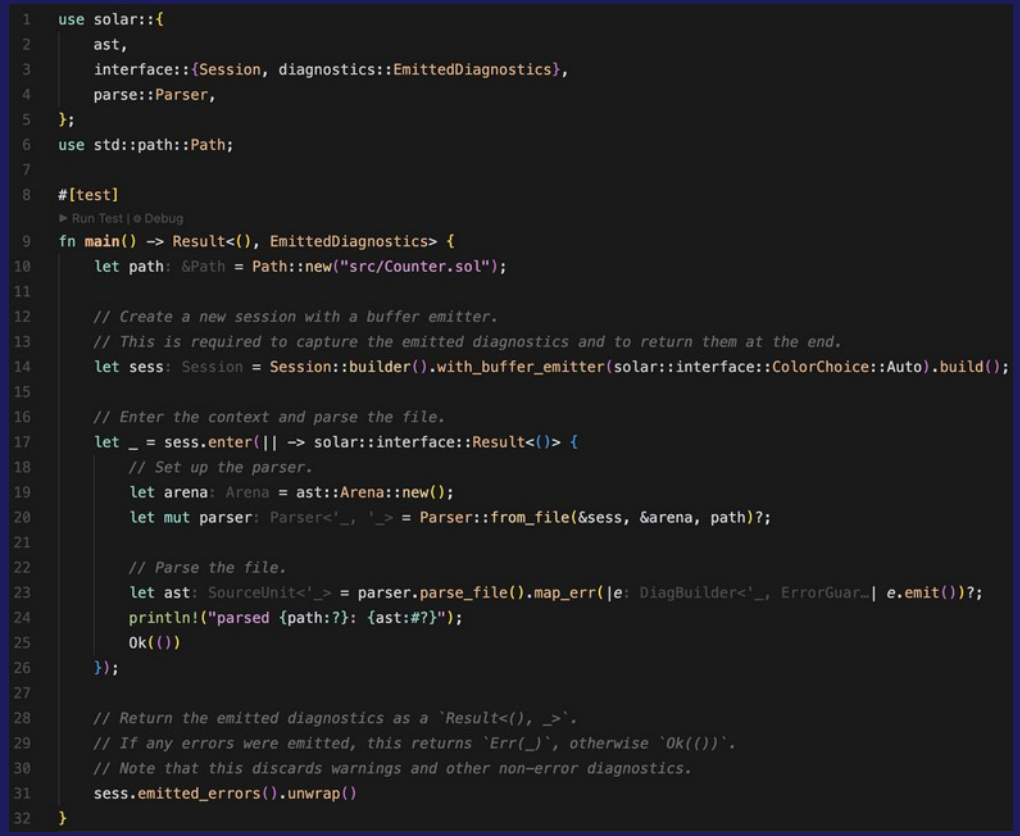


# Written from the ground up with performance in mind

1. Multiple IRs
2. Data Oriented Design
3. Multi-threaded-friendly structure
4. Possibility for incremental compilation

1. Ergonomic design
2. Single entry point (Compiler)
3. Multiple diagnostic formats
4. Multiple backends (soon)

Focus on modularity, extensibility, library usage



```rust
use solar::{
    ast,
    interface::{Session, diagnostics::EmittedDiagnostics},
    parse::Parser,
};
use std::path::Path;

#[test]
▶ Run Test | ⚙ Debug
fn main() -> Result<(), EmittedDiagnostics> {
    let path: &Path = Path::new("src/Counter.sol");

    // Create a new session with a buffer emitter.
    // This is required to capture the emitted diagnostics and to return them at the end.
    let sess: Session = Session::builder().with_buffer_emitter(solar::interface::ColorChoice::Auto).build();

    // Enter the context and parse the file.
    let _ = sess.enter(|| -> solar::interface::Result<()> {
        // Set up the parser.
        let arena: Arena = ast::Arena::new();
        let mut parser: Parser<'_, '_> = Parser::from_file(&sess, &arena, path)?;

        // Parse the file.
        let ast: SourceUnit<'_> = parser.parse_file().map_err(|e: DiagBuilder<'_, ErrorGuar…| e.emit())?;
        println!("parsed {path:?}: {ast:#?}");
        Ok(())
    });

    // Return the emitted diagnostics as a `Result<(), _>`.
    // If any errors were emitted, this returns `Err(_)`, otherwise `Ok(())`.
    // Note that this discards warnings and other non-error diagnostics.
    sess.emitted_errors().unwrap()
}
```

# Early benchmark results

## 5x-10x parsing

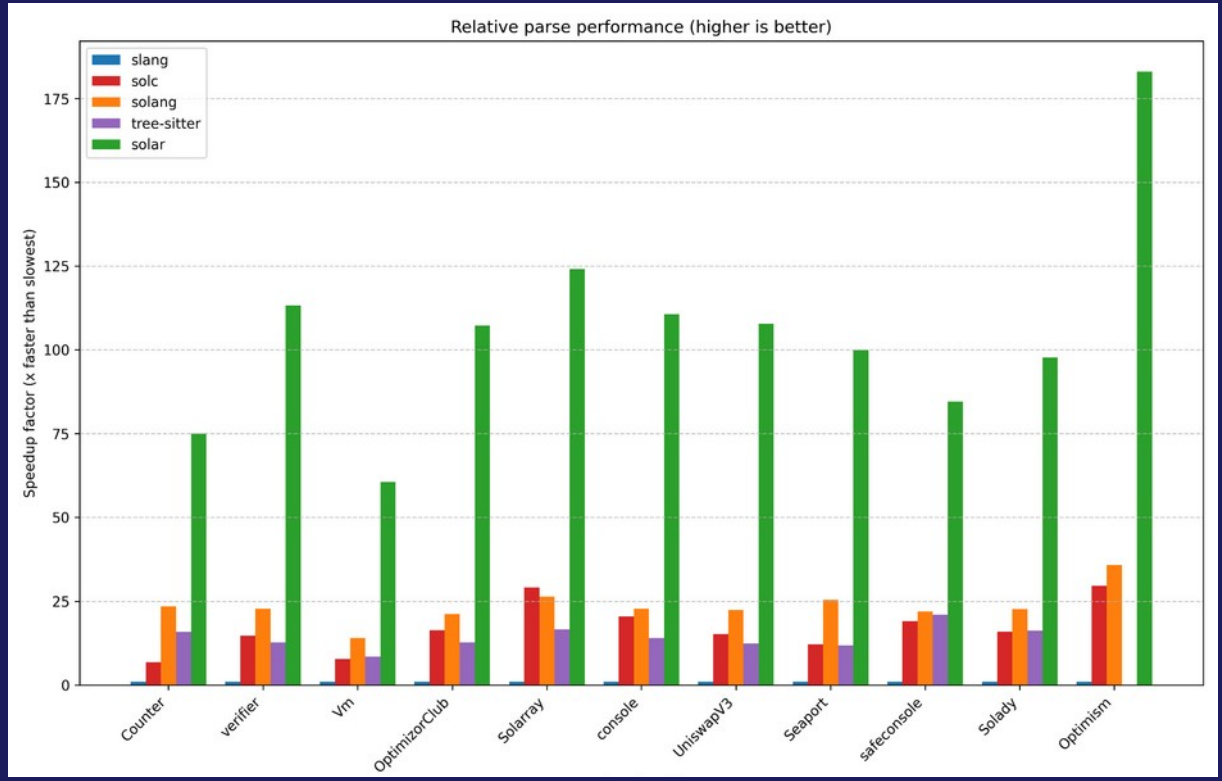
Faster at single-threaded parsing; multithreaded parsing

## 50x ABI

Faster at emitting ABI information

## ??x bytecode

Faster and better bytecode than Solc (SOON)


Relative parse performance (higher is better)
slang
solc
solang
tree-sitter
solar
Speedup factor (x faster than slowest)
0
25
50
75
100
125
150
175
Counter
verifier
Vm
OptimizorClub
Solarray
console
UniswapV3
Seaport
safeconsole
Solady
Optimism

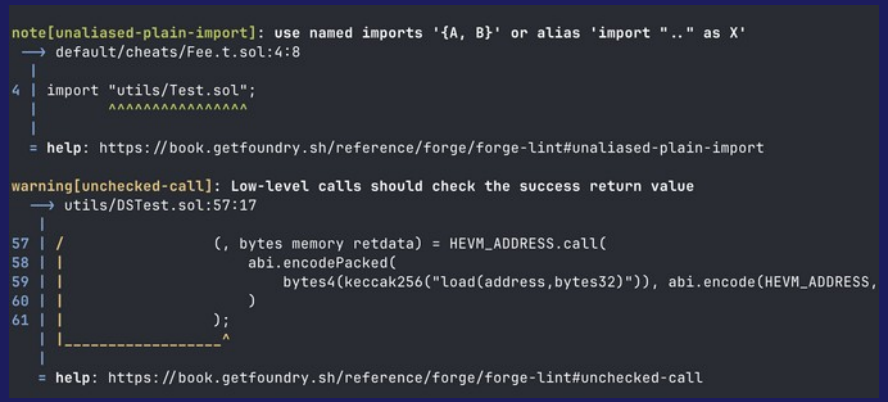
# SOLAR

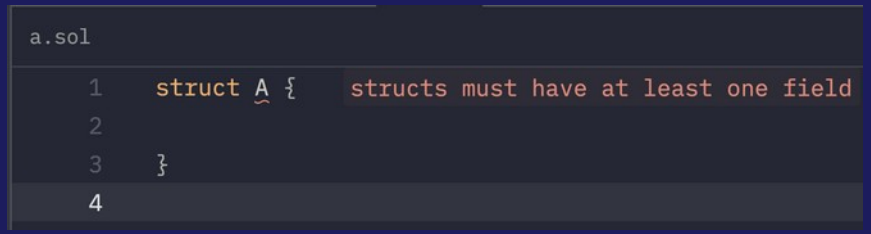
```
~/solady main ❯ hyperfine -w1 --shell=zsh 'solar $(forge re) {src,test}/**/*.sol --emit abi' 'solc $(forge re) {src,test}/**/*.sol --combined-json abi'
Benchmark 1: solar $(forge re) {src,test}/**/*.sol --emit abi
  Time (mean ± σ):      38.1 ms ±   1.2 ms    [User: 91.6 ms, System: 23.0 ms]
  Range (min … max):    35.4 ms …  40.9 ms    78 runs

Benchmark 2: solc $(forge re) {src,test}/**/*.sol --combined-json abi
  Time (mean ± σ):      1.589 s ±  0.023 s    [User: 1.299 s, System: 0.287 s]
  Range (min … max):    1.554 s …  1.615 s    10 runs

Summary
  solar $(forge re) {src,test}/**/*.sol --emit abi ran
   41.70 ± 1.47 times faster than solc $(forge re) {src,test}/**/*.sol --combined-json abi
```

SOLAR

# Forge Lint

Powerful static analysis built-in to Forge, powered by Solar






# Solar LSP (soon)

Powering Foundry IDE extensions (soon)

Diagnostics, code actions, go to definition, etc.

# Q&A