OnGrid Systems

Smart Contracts LLVM compiler

FANTOM

www.ongrid.pro

Kirill Varlamov

Systems architect, CTO

kirill@ongrid.pro

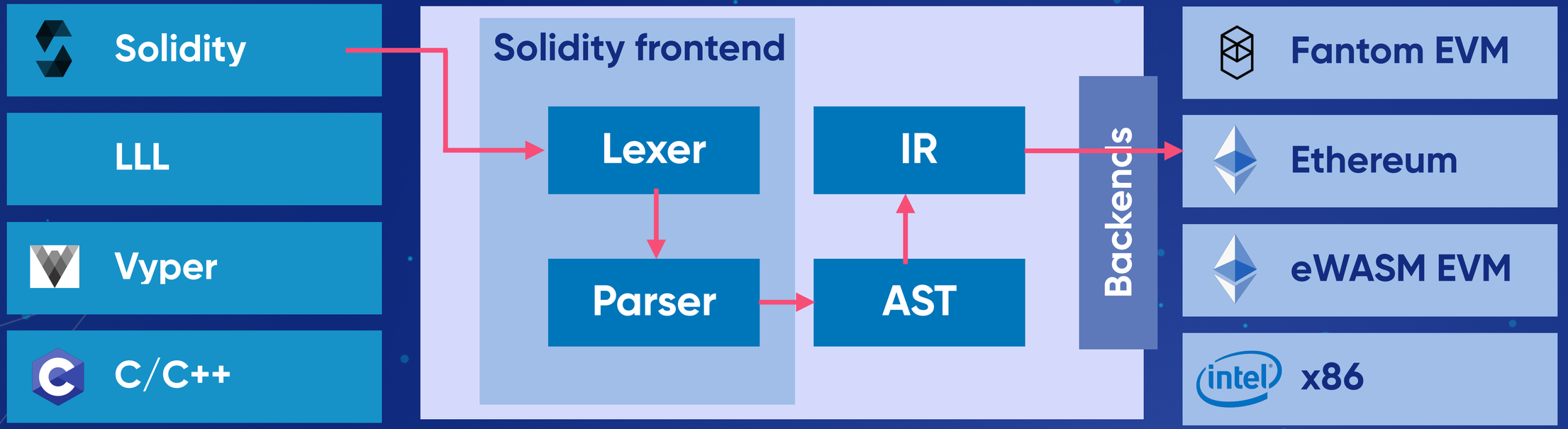
Roman Nesytov

Backend developer, Ops

r.nesytov@ongrid.pro

OnGrid Systems

LLVM compiler framework for smart contracts. Motivation



What does the industry need?

Language and VM independent compiler framework for decentralised state machines

Both frontend-bound and backend-bound components need to be implemented

OnGrid Systems

Solidity is problematic frontend, EVM is painful backend.

Frontend side: Solidity intentionally hides important EVM features to make it simple

it operates with Contracts and make it look like instantiating of the class hiding what is under the hood:

- in EVM constructor returns the contract code. In Solidity it's just internal constructor function
- opcodes are unavailable or hard to use directly (LOG, CREATE, SELFDESTRUCT)
- storage and memory are combined and available as variables

Backend side: Decentralised state machines (incl. EVM) are very uncommon too

- specific opcodes missing on legacy platforms (LOG, SELFDESTRUCT, CREATE)
- specific interfaces: no peripherals, only RLPx serialized data
- very specific native structures: Merkle trees, Bloom filters. No registers.

We took simpler LLL language and reduced EVM opcodes set

Added LLVM-IR backend to Solidity.LLL

It's able to compile both:

- EVM bytecode suitable for deployment via web3
- LLVM-IR executable with iLLL interpreter

OnGrid Systems

Live demo (build and verify):

1. **Build: Write LLL code**

   Only primitives supported - math and operators nesting.
   (+ 1 (- 5 2))

2. **Build: Compile LLL to LLVM-IR**

   we implemented LLVM-IR backend specified with -l option. It **outputs standard LLVM-IR**

3. **Verify: Run LLVM-IR on x86 with standard lli**

   returns the result of LLVM-IR execution confirming the correctness of LLVM-IR code

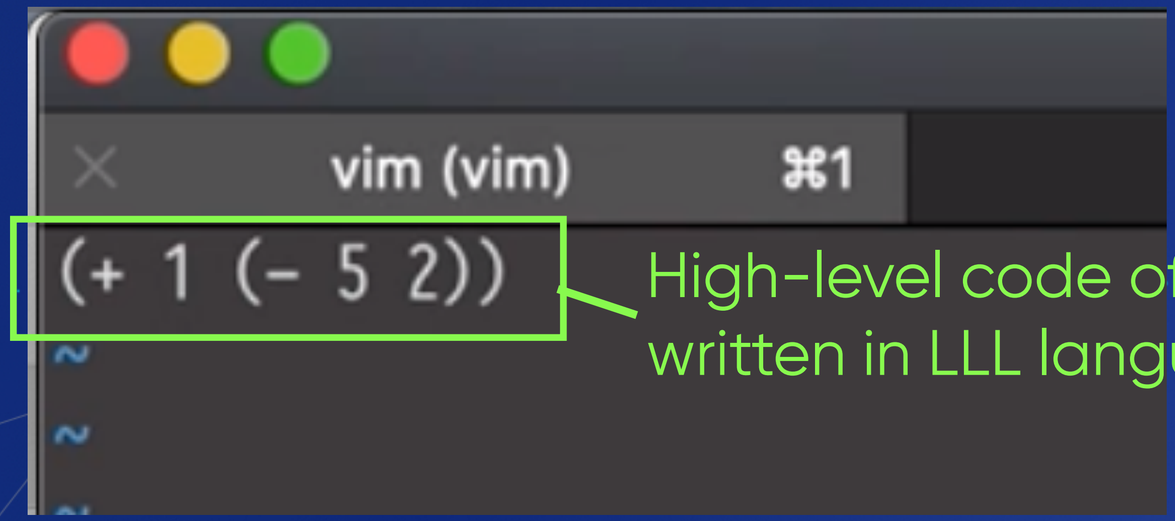
4. **Verify: Compile LLL to EVM bytecode**

5. **Verify: Deploy bytecode to Ethereum EVM**

   web3.getCode() returns the same value as p. 3

OnGrid Systems

1. Build: Write LLL code



OnGrid Systems

2. Build: Compile LLL to LLVM-IR

newly introduced flag -l (LLVM-IR output)

```
~/dev/present (zsh)  ⌘1
~/dev/present  vim simple.lll
~/dev/present  ./lllc -l simple.lll > simple.ll
~/dev/present  cat simple.ll
; ModuleID = 'test'
source_filename = "test"

define i32 @main() {
EntryBlock:
  ret i32 4
}

~/dev/present
```

got LLVM-IR output

OnGrid Systems

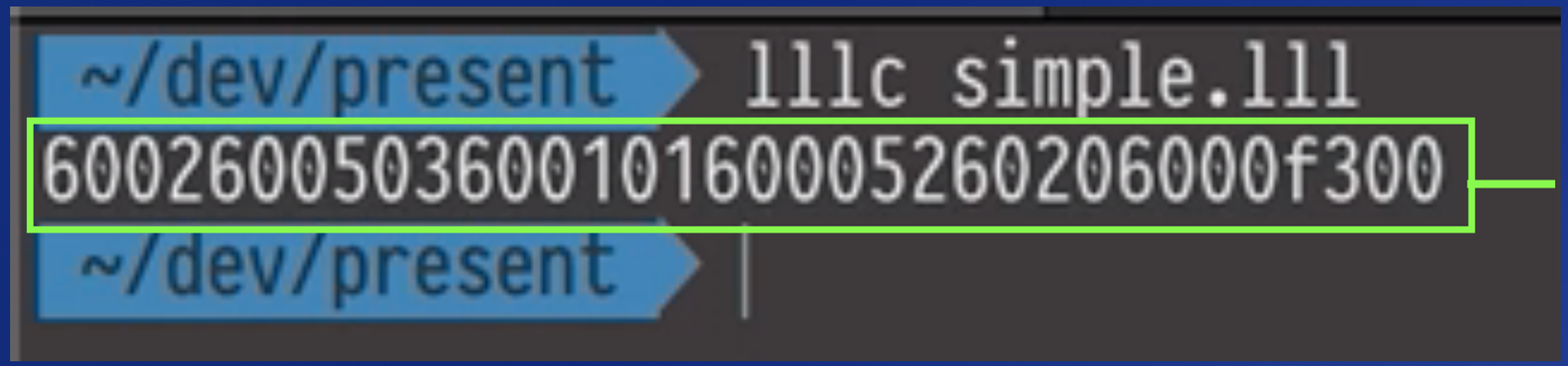
3. Verify: Run LLVM-IR on x86 with standard lli interpreter

```
~/dev/present  /usr/local/Cellar/llvm/7.0.0/bin/lli simple.ll
~/dev/present  echo $?
4
~/dev/present
```

result of LLVM-IR interpretation on x86.
Remember it

OnGrid Systems

4. Verify: Compile LLL to EVM bytecode



take EVM bytecode

OnGrid Systems

5. Verify: Deploy bytecode to Ethereum EVM

code from LLLc (prev. step)

```
In [63]: contract = w3.eth.contract(abi=[], bytecode='600260050360010160005260206000f300')

In [64]: contract.constructor().transact()
Out[64]: HexBytes('0x162e774a8adb7907477f496dd14cc1076f2677c7b93aab6abfcad10d50656671')

In [65]: w3.eth.getTransactionReceipt('0x162e774a8adb7907477f496dd14cc1076f2677c7b93aab6abfcad10d50656671')
Out[65]:
AttributeDict({'transactionHash': HexBytes('0x162e774a8adb7907477f496dd14cc1076f2677c7b93aab6abfcad10d50656671'),
 'transactionIndex': 0,
 'blockNumber': 11,
 'blockHash': HexBytes('0xb30315d3855d210127cb1454c38ea21462286fd2b171711a31c5186d17d0fbe9'),
 'cumulativeGasUsed': 60394,
 'gasUsed': 60394,
 'contractAddress': '0x00De48310d77A4d56aa400248b0B1613508f5B73',
 'logs': [],
 'state_root': b'\x01'})

In [66]: w3.eth.getCode('0x00De48310d77A4d56aa400248b0B1613508f5B73')
Out[66]: HexBytes('0x0000000000000000000000000000000000000000000000000000000000000004')
```

web3.getCode() returns the same value as p. 3

OnGrid Systems

Live demo (build and verify):

1. Build: Write LLL code

Only primitives supported - math and operators nesting.
(+ 1 (- 5 2))

2. Build: Compile LLL to LLVM-IR

we implemented LLVM-IR backend specified with -l option. It **outputs standard LLVM-IR**

3. Verify: Run LLVM-IR on x86 with standard lli

returns the result of LLVM-IR execution confirming the correctness of LLVM-IR code

4. Verify: Compile LLL to EVM bytecode

5. Verify: Deploy bytecode to Ethereum EVM

web3.getCode() returns the same value as p. 3

OnGrid Systems

Questions?

Links and related work

- https://github.com/OnGridSystems/SolidityToLLVM – Github Repository
- https://media.consensys.net/an-introduction-to-lll-for-ethereum-smart-contract-development-e26e38ea6c23 – LLL intro by Consensys team

OnGrid Systems

Blockchain platforms and cryptocurrency services development, audit of cloud systems, consulting.

www.ongrid.pro