A Developer Ecosystem For The Flint Programming Language

Zubair Chowdhury

What is the problem?

- Ethereum is a platform for decentralized trustless execution of programs known as smart contracts
- Solidity is an unsafe language, Flint was designed to replace it
- It is unlikely that Flint will be adopted without a strong developer ecosystem

Ethereum Quirks

- Execution costs money
- Contracts can send and receive money
- Events are used to send data off the blockchain

Testing Framework

Testing Framework - Motivations

- Provide the ability for developers to write unit tests to check the correctness of their contract
- Used widely in industry, developers are used to writing unit tests (JetBrains Survey – 70% of Developers use unit tests regularly)
- Smart contracts cannot be modified once deployed

[2]

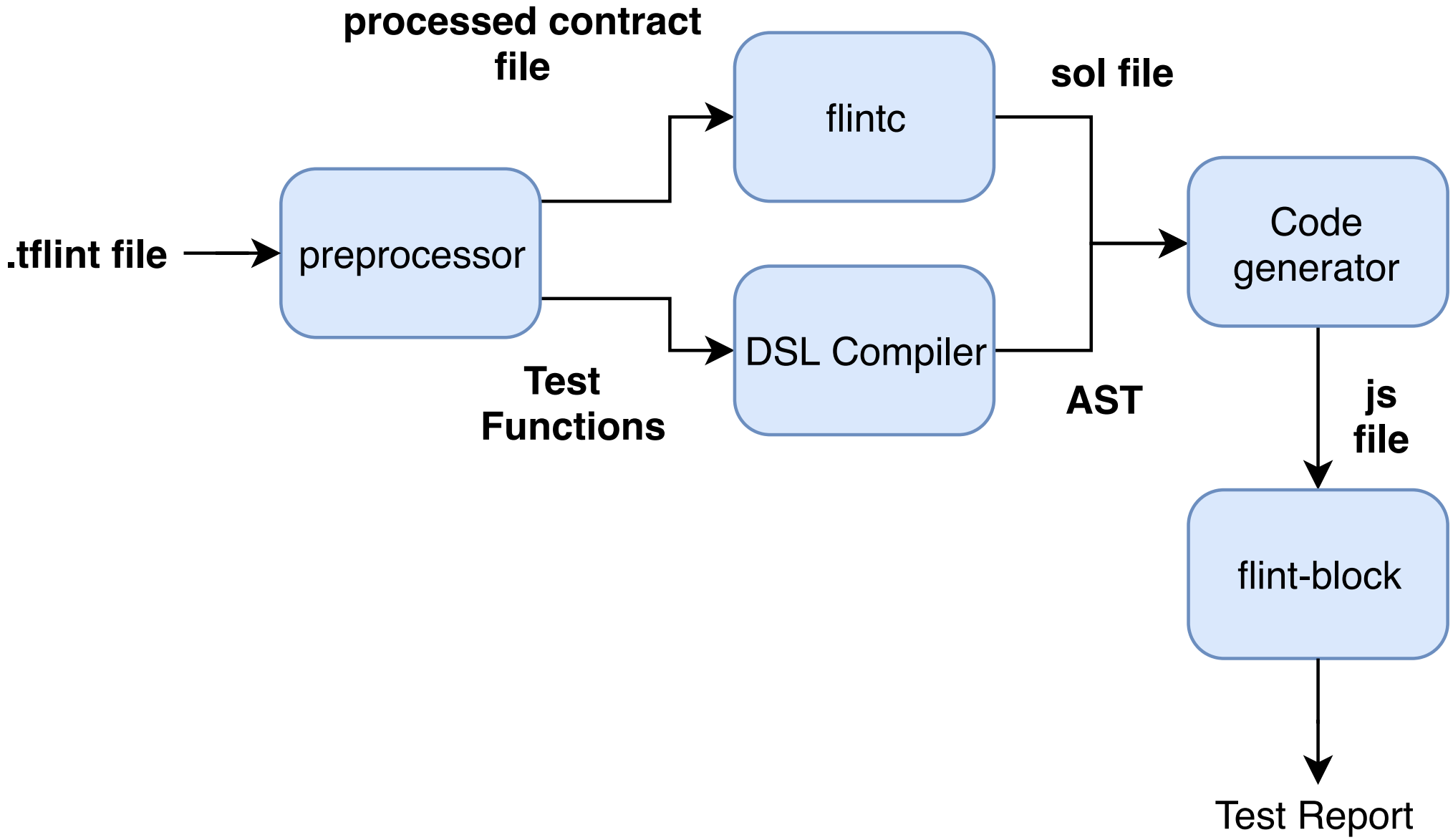
Testing Framework - Challenges

- Smart contracts can only be run on a blockchain
- Need to be able to test all of Flint’s safety features, exceptions, events and also support sending money to a contract during a test
- Choosing how the test framework runs

Testing Framework - Overview

- Testing DSL is used to write tests
- Compiled down to JavaScript so we can circumvent the limitations of running on the blockchain
- Utilize web3 library to create a set of primitives which enable us to communicate with the blockchain

Testing Framework Demo


processed contract file
flintc
sol file
.tflint file
preprocessor
Code generator
DSL Compiler
Test Functions
AST
js file
flint-block
Test Report

Testing Framework - Preprocess

- Method added to each contract being tested to support initialization of contracts with constructor parameters

- If code coverage is toggled then the contract is also instrumented

Testing Framework – Main Ideas

- Framework differentiates between transactions and calls.
- Manage asynchronous nature by monitoring transactions of interest
- Framework translates between blockchain types and Flint types
- Test Asserts are implemented by checking for reverted transactions and searching event logs.

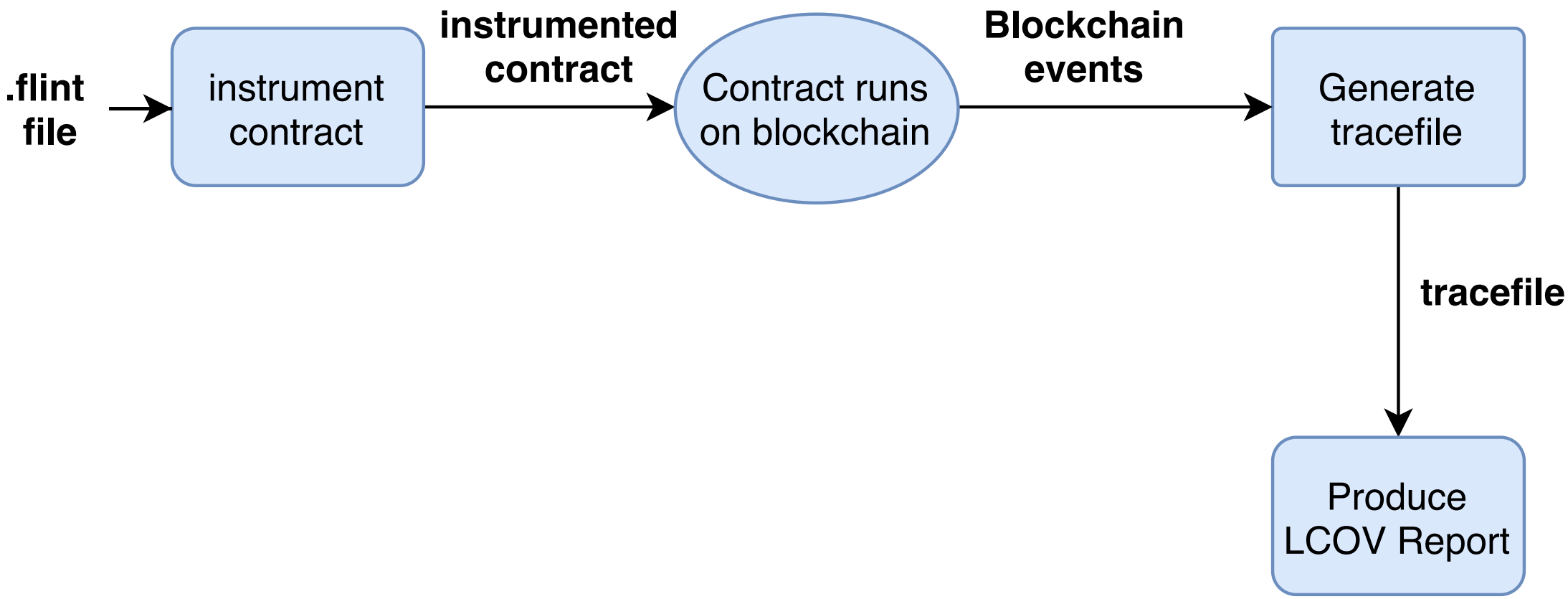
Testing Framework – Example Translation

```
public func test_cant_call_increment_in_print_state() {
    let owner : Address = newAddress()
    let c : Counter = Counter(0, owner)

    setAddr(owner)
    c.printReady()
    unsetAddr()

    assertCantCallInThisState(c.increment)
}
```

```
async function test_cant_call_increment_in_print_state(t_contract) {
   let assertResult012 = {result: true, msg:""}
   console.log(chalk.yellow("Running test_cant_call_increment_in_print_state"))
   let owner = newAddress();
   await transactional_method(t_contract, 'testFrameworkConstructor', [0, owner], {})
   //let c = Counter(0, owner);
   setAddr(owner)
   await transactional_method_void(t_contract, 'printReady', [], {})
   unsetAddr()
   await assertCantCallInThisState(assertResult012, "increment", [], t_contract)
   process_test_result(assertResult012, "test_cant_call_increment_in_print_state")
}
```

Code Coverage

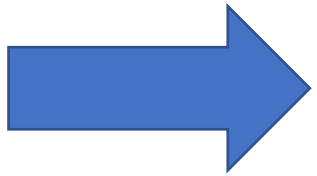
Code Coverage



Code Coverage - Instrumentation

- Instrument With Events

```
11    public func testCoverage() {
12
13      if (true) {
14      }
15
16
17      let z : Int = 0
18    }
```



```
public func testCoverage() {
  emit funcC(line: 11, fName: "testCoverage")

  if (true) {
    emit branchC(line: 13, branch: 0, blockNum: 1)
  } else {
    emit branchC(line: 14, branch: 1, blockNum: 2)
  }

  emit stmC(line: 17)
  let z : Int = 0
}
```

Code Coverage – LCOV Tracefile

- Produce LCOV Tracefile

```
SF: [redacted]
DA:17,1
BRDA: 13, 1, 0, 1
BRDA: 14, 2, 1, 1
FN:11, testCoverage
BRF:2
BRH:1
FNF:1
FNH:1
LF:1
LH:1
end_of_record
```

Contract Analysis

Contract Analysis - Motivations

- Many language ecosystems have tools to help developers understand their code

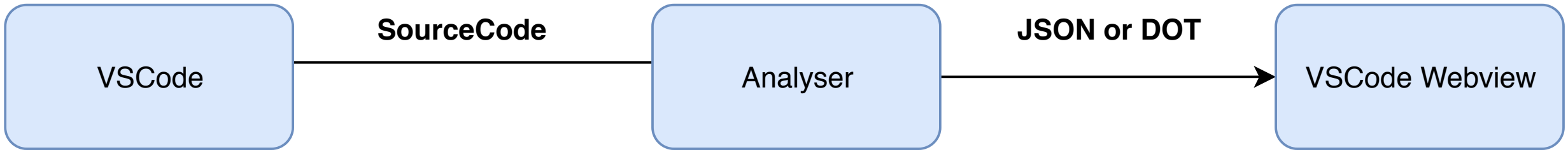
- Helps developer enforce best practices and understand how well they are leveraging Flints safety features

Contract Analysis - Challenges

- Deciding what is important to analyze
- Displaying the results of the analysis in a meaningful way
- Ensuring that the analysis tools are easy to run

Contract Analysis Demo

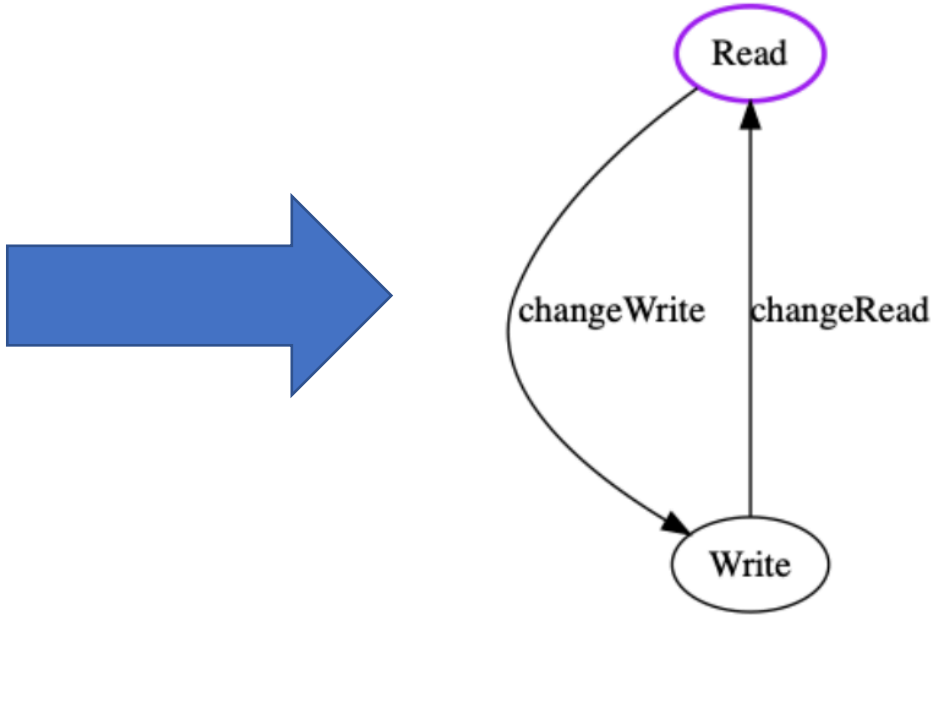
Contract Analysis Overview



Contract Analysis – Type state visualization

- Walk the Flint AST, trace all the state changes by looking for instances of the `become` keyword
- Produce a dot file representing the state diagram

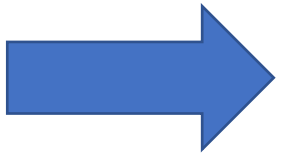
```
Counter@(Write) :: (any) {

  mutating public func changeRead() {
    become Read
  }

}

Counter@(Read) :: (any) {
  public init() {
    become Read
  }

  mutating public func changeWrite() {
    become Write
  }

}
```

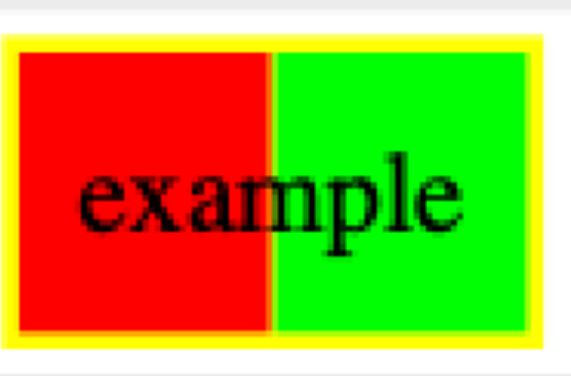


Contract Analysis – Function Visualization

- Walk the contract AST, find all contract functions and infer properties.

```
@payable
mutating public func example(implicit w : Wei) {
    send(address: owner, amount: &w)
    self.value += 1
}
```





Contract Analysis – Protection Analysis

- AST walk – Infer caller protections and state protections from the AST

- Information is collected and converted into JSON and passed back to VSCode for display

```
Counter @(Read) :: (owner) {
    public func example() {
    }
}
```

Language Server

Language Server - Motivations

- Common to most language ecosystems (82% of developers use an IDE, 69% use light desktop editors e.g. VSCode)
- Makes the process of writing code convenient
- Flint currently has limited supported for this

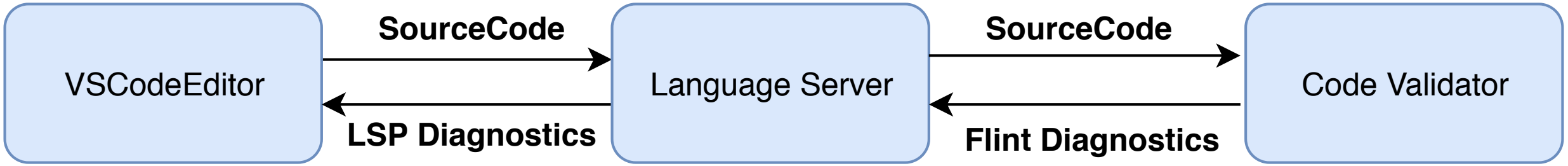
[2]

Language Server - Challenges

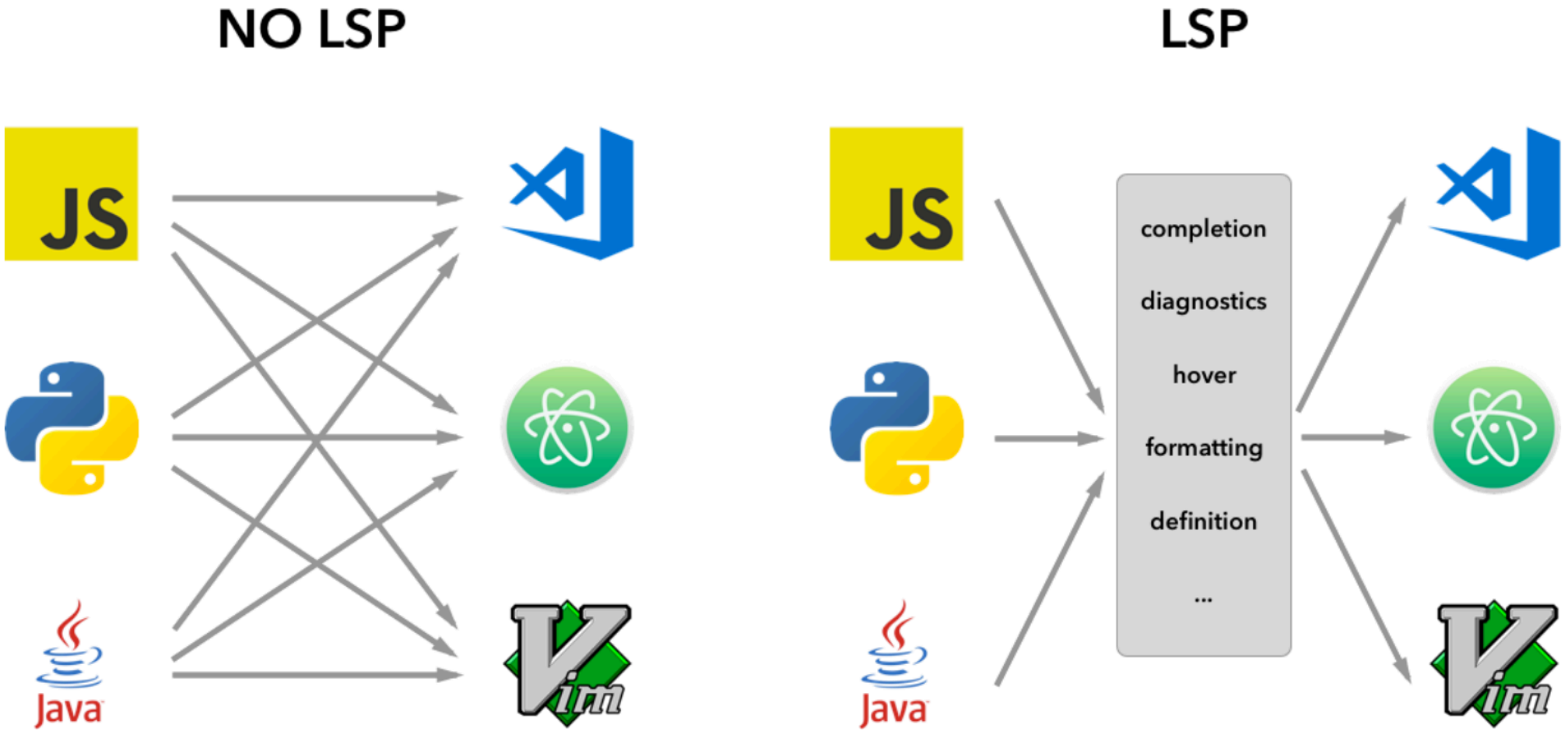
- Code which is being written is generally syntactically incorrect
- The tool which provides Flint language support should not be tied to any particular editor

Language Server Demo

Language Server Overview



LSP Protocol Overview



[1]

LSP Protocol

- Language agnostic protocol, centers around the structure of the document instead of the language
- Server and Client run in different processes
- Over 15 editors currently support the LSP protocol

Wirth’s Follow Set Method

```
var value : Int
```

```
var value Int
```

- Extension to panic recovery
- On detection of error, parser discards tokens until it hits a synchronization token
- Synchronization tokens are decided based on grammar of language

Interactive Console (REPL)

Interactive Console - Motivations

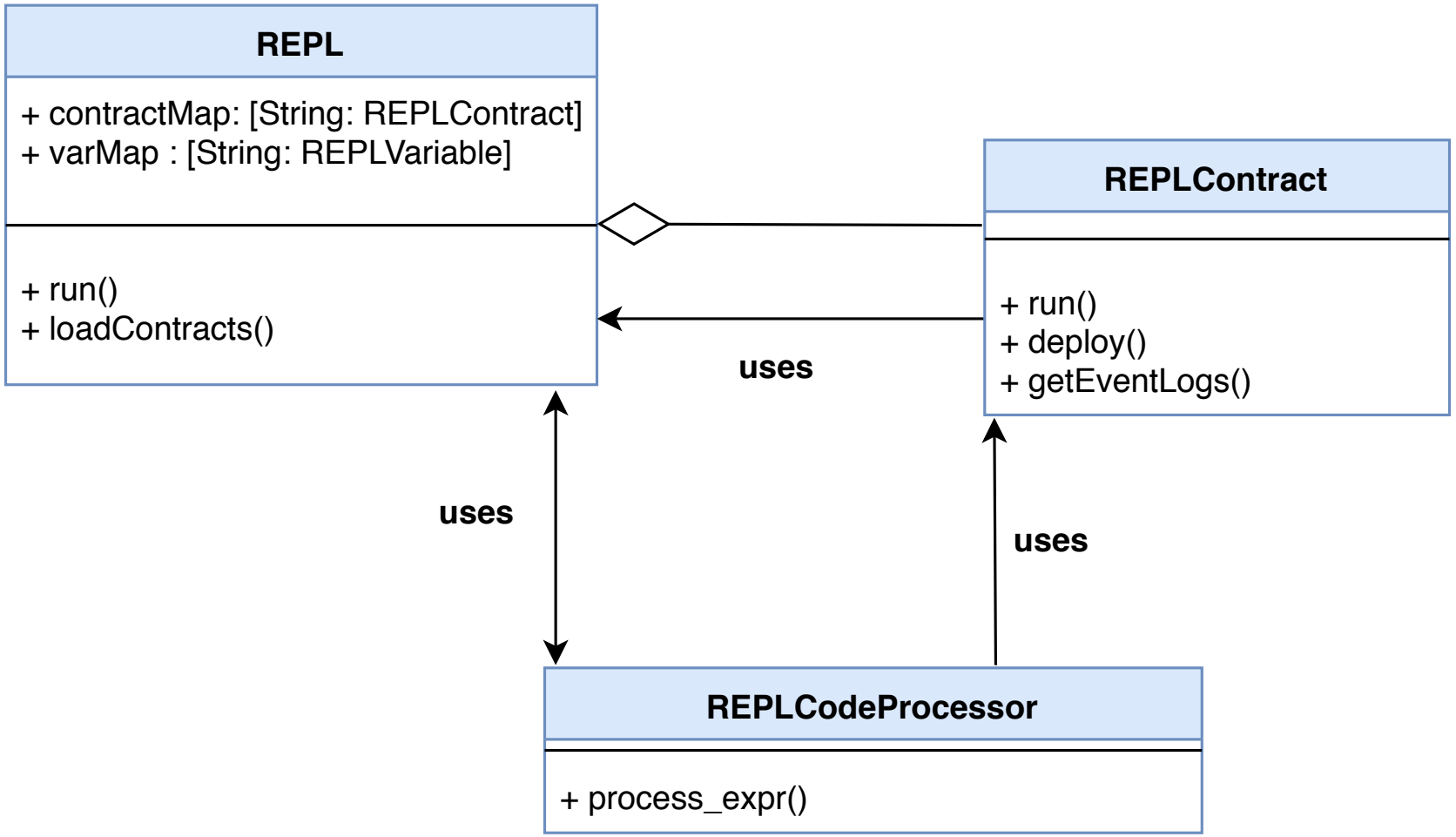
- Developers run their code whilst they are developing to get rapid feedback on their code.
- Manually deploying and interacting with your contract is cumbersome and requires a deep understanding of the internals of the blockchain.
- Provide a convenient way for developers to interact with their contracts on their local machine

Interactive Console - Challenges

- Console should allow developers to use Flint syntax
- Designing a REPL which can deploy and manage interaction with multiple contracts
- REPL should support users querying event logs and sending money to contracts

Interactive Console Demo

Interactive Console Overview (Information Omitted)

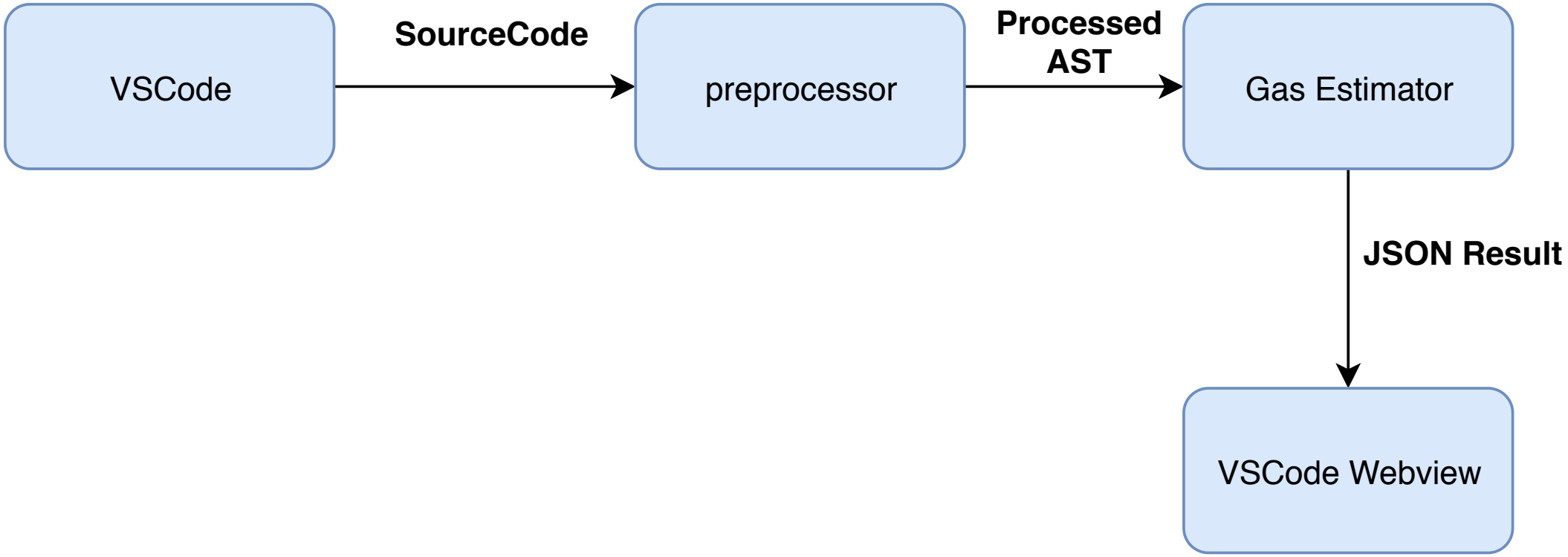


Gas Estimator

Gas Estimator - Motivations

- Contracts require money to operate on real blockchains
- Gas estimation will help the programmer understand the economics behind their contracts
- Help guide developer time – costly functions can be optimized first.

Gas Estimator Overview



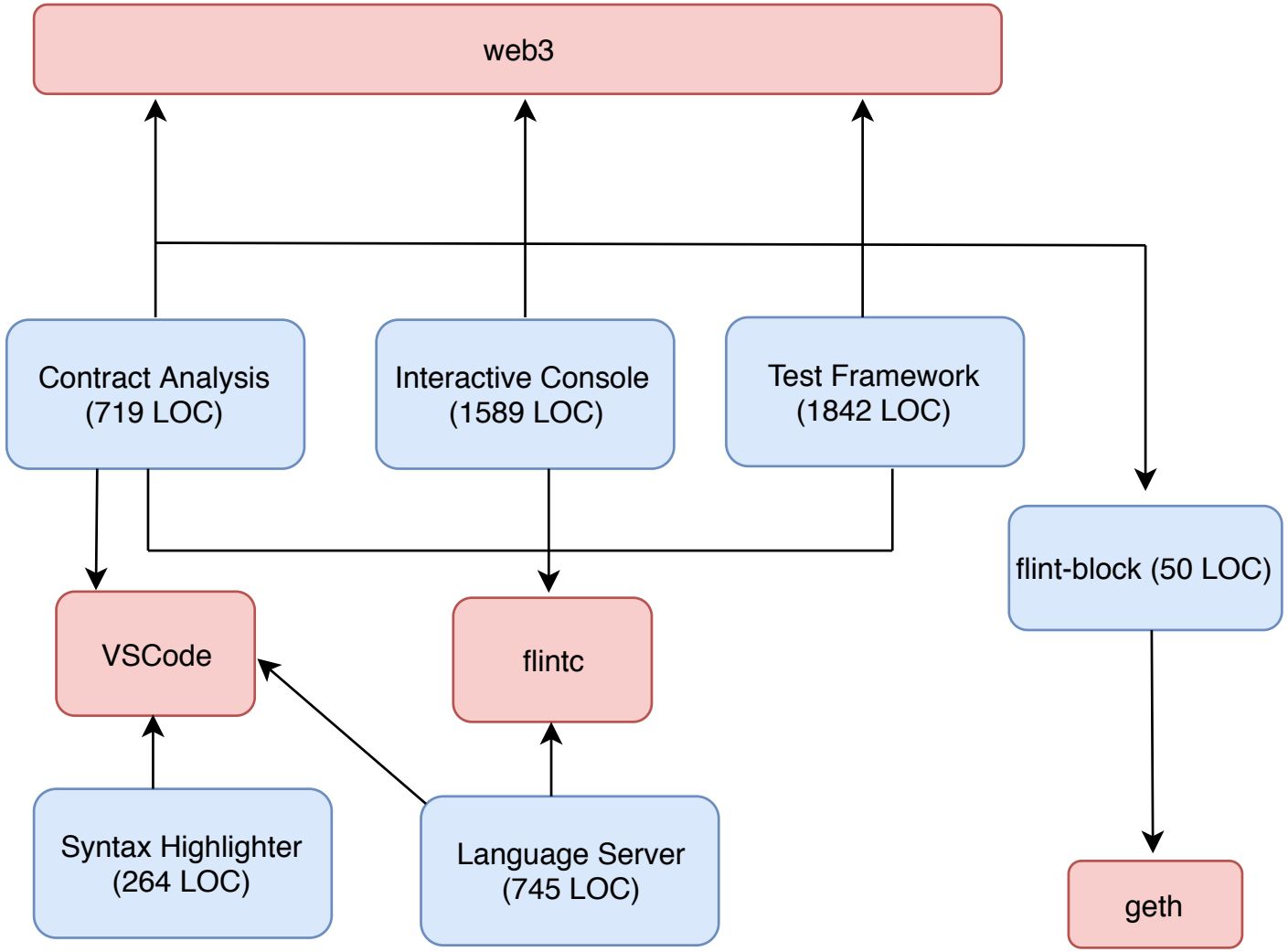
Gas Estimator - Challenges

- Simulating transactions requires calling functions with parameters. We only fill constructor parameters.

- Each of the functions in the contract need to be called by anyone, Flint safety features need to be removed.

Gas Estimator Demo

Final Ecosystem



Evaluation

- We have created a set of tools to help developers write safer Flint code. The ecosystem has no debugging or profiling tools.
- We show in the report how the ecosystem adds value by demonstrating in detail how it can be used to detect bugs.
- Compared to the Solidity ecosystem, we are smaller but more coherent and we do not have an issue of multiplicity.
- Speed of the testing framework is limited by our backing node.

Challenges

- Working with the Ethereum ecosystem. It is new, the provided tools have bugs and the documentation is not great.
- Blockchain provides little visibility when things go wrong and debugging tools are limited
- Significant amount of design and implementation (5000 lines of code – mix of Swift, bash & JavaScript)

Any Questions?

References

1. Language Server Extension Guide | Visual Studio Code Extension API. (n.d.). Retrieved January 22, 2019, from https://code.visualstudio.com/api/language-extensions/language-server-extension-guide
2. The State of Developer Ecosystem 2018 - Infographic | JetBrains. (n.d.). Retrieved January 19, 2019, from https://www.jetbrains.com/research/devecosystem-2018/

Performance Comparisons

Feature	Truffle Test	Flint-Test
Verifying Events	10.52	18.99
Testing State Protections	10.64	22
Testing Caller Protections	10.53	26.14
Testing For Exceptions	8.11	23.93

Table 8.1: Execution Speed Of Testing Framework For Different Features