Safer Smart Contract Programming with **Scilla**

Ilya Sergey Vaivaswatha Nagaraj Jacob Johannsen

Amrit Kumar Anton Trunov Ken Chan

scilla-lang.org

Yale NUS College

NUS National University of Singapore

zilliqa

Blockchains 101

$$\{tx_1, tx_3, tx_5, tx_4, tx_2\}$$

- transforms a **set** of transactions into a *globally-agreed* **sequence**

- “distributed timestamp server” (Nakamoto 2008)

blockchain consensus protocol

transactions can be *anything*

$$tx_5 \rightarrow tx_3 \rightarrow tx_4 \rightarrow tx_1 \rightarrow tx_2$$

Blockchains 101

$$\{tx_1, tx_3, tx_5, tx_4, tx_2\}$$

$$\downarrow$$

$$[tx_5, tx_3] \rightarrow [tx_4] \rightarrow [tx_1, tx_2]$$

$$\downarrow$$

$$tx_5 \rightarrow tx_3 \rightarrow tx_4 \rightarrow tx_1 \rightarrow tx_2$$

Blockchains 101

$$\{tx_1, tx_3, tx_5, tx_4, tx_2\}$$

$$\downarrow$$

$$[tx_5, tx_3] \leftarrow [tx_4] \leftarrow [tx_1, tx_2]$$

$$\downarrow$$

$$tx_5 \rightarrow tx_3 \rightarrow tx_4 \rightarrow tx_1 \rightarrow tx_2$$

Transactions

- Executed *locally*, alter the *replicated* state.

- Simplest case: *transferring funds* from *A* to *B,*
  consensus: *no* double spending.

- More interesting:
  deploying and executing *replicated computations*

Smart Contracts

Smart Contracts

- *Stateful mutable* objects replicated via a consensus protocol
- State typically involves a stored amount of *funds/currency*
- Main usages:
    - crowdfunding and ICO
    - multi-party accounting
    - voting and arbitration
    - puzzle-solving games with distribution of rewards
- Supporting platforms: Ethereum, Tezos, Concordium, FB Libra,…

A Smart Contract in *Solidity*™

```
contract Accounting {
  /* Define contract fields */
  address owner;
  mapping (address => uint) assets;

  /* This runs when the contract is executed */
  function Accounting(address _owner) {
    owner = _owner;
  }

  /* Sending funds to a contract */
  function invest() returns (string) {
    if (assets[msg.sender].initialized()) { throw; }
    assets[msg.sender] = msg.value;
    return "You have given us your money";
  }
}
```

Mutable fields

Constructor

Entry point

The Givens of Smart Contracts

Deployed in a *low-level language*	Uniform compilation target
Must be *Turing-complete*	Run arbitrary computations
Code is law	What else if not the code?

The Givens of Smart Contracts

Deployed in a *low-level language*	**Difficult** for audit and verification
Must be *Turing-complete*	Complex semantics, **exploits**
Code is law	One should understand the **code** to understand the **contract**

Sending a Message or Calling?

```
contract Accounting {
  /* Other functions */

  /* Sending funds to a contract */
  function invest() returns (string) {
    if (assets[msg.sender].initialized()) { throw; }
    assets[msg.sender] = msg.value;
    return "You have given us your money";
  }

  function withdrawBalance() {
    uint amount = assets[msg.sender];
    if (msg.sender.call.value(amount)() == false) {
      throw;
    }
    assets[msg.sender] = 0;
  }
}
```

Sending a Message or Calling?

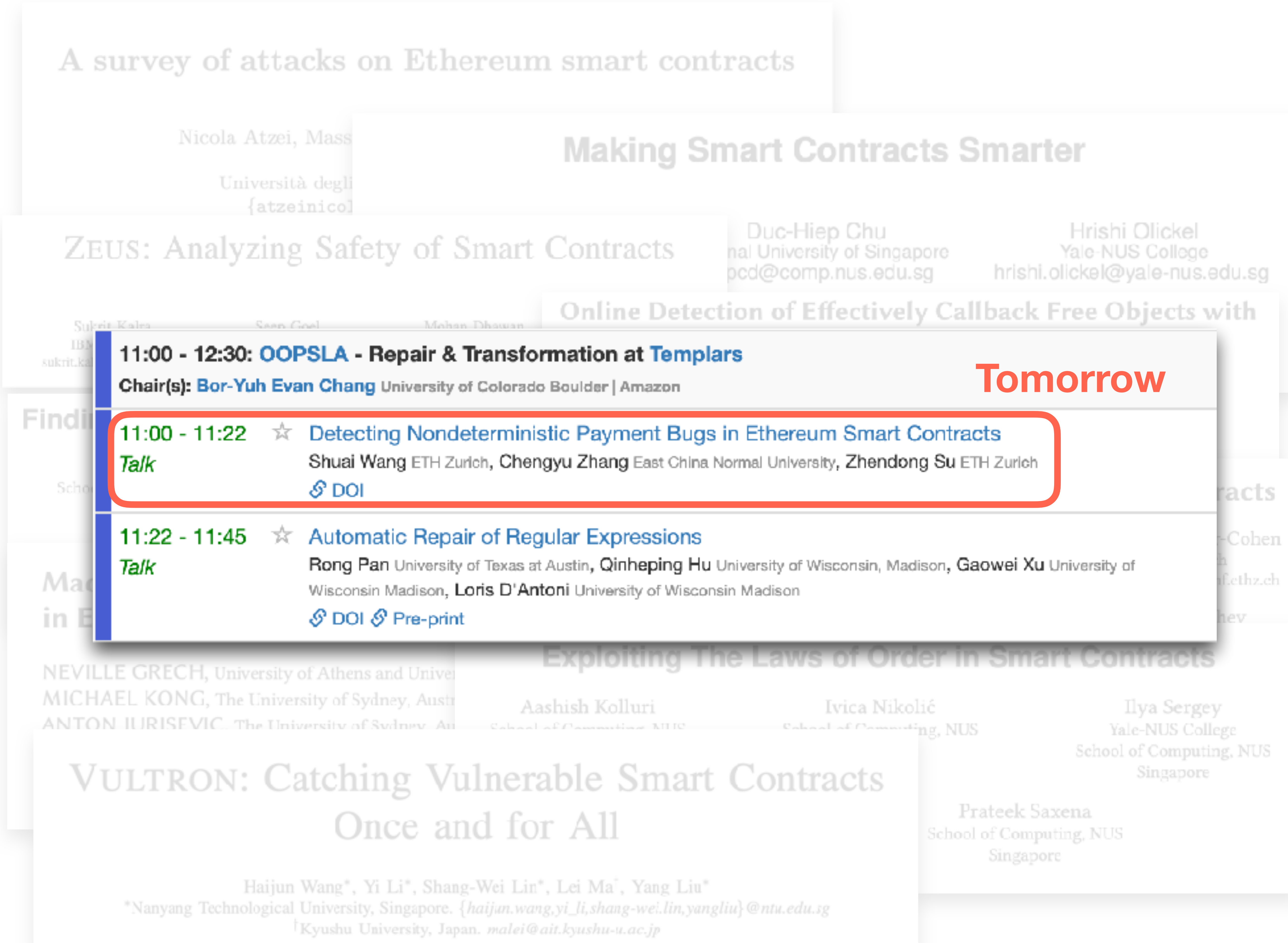
```
contract Accounting {
  /* Other functions */

  /* Sending funds to a contract */
  function invest() returns (string) {
    if (assets[msg.sender].initialized()) { throw; }
    assets[msg.sender] = msg.value;
    return "You have given us your money";
  }

  function withdrawBalance() {
    uint amount = assets[msg.sender];
    if (msg.sender.call.value(amount)() == false) {
      throw;
    }
    assets[msg.sender] = 0;
  }
}
```

Caller can *reenter* and withdraw **again**


A survey of attacks on Ethereum smart contracts
Nicola Atzei, Mass
Università degli
{atzeinicol
Making Smart Contracts Smarter
Duc-Hiep Chu
nal University of Singapore
pcd@comp.nus.edu.sg
Hrishi Olickel
Yale-NUS College
hrishi.olickel@yale-nus.edu.sg
ZEUS: Analyzing Safety of Smart Contracts
Sukrit Kalra
IBM Research
sukrit.kalra@in.ibm.com
Seep Goel
IBM Research
sgoel219@in.ibm.com
Mohan Dhawan
IBM Research
Online Detection of Effectively Callback Free Objects with Applications to Smart Contracts
sity, Israel
USA
rch, USA
Finding The Greedy, Prodigal, and Suicidal Contracts at Scale
Ivica Nikolić
School of Computing, NUS
Singapore
Aashish Kolluri
Singapore
SECURIFY: Practical Security Analysis of Smart Contracts
Andrei Dan
ETH Zurich
andrei.dan@inf.ethz.ch
Dana Drachsler-Cohen
ETH Zurich
dana.drachsler@inf.ethz.ch
Florian Bünzli
Martin Vechev
MadMax: Surviving Out-of-Gas Conditions in Ethereum Smart Contracts
NEVILLE GRECH, University of Athens and Univer
MICHAEL KONG, The University of Sydney, Austr
Exploiting The Laws of Order in Smart Contracts
Aashish Kolluri
Ivica Nikolić
Ilya Sergey
Yale-NUS College
School of Computing, NUS
Singapore
Prateek Saxena
School of Computing, NUS
Singapore
VULTRON: Catching Vulnerable Smart Contracts Once and for All
Haijun Wang*, Yi Li*, Shang-Wei Lin*, Lei Ma†, Yang Liu*
*Nanyang Technological University, Singapore. {haijun.wang,yi_li,shang-wei.lin,yangliu}@ntu.edu.sg
†Kyushu University, Japan. malei@ait.kyushu-u.ac.jp


A survey of attacks on Ethereum smart contracts
Making Smart Contracts Smarter
ZEUS: Analyzing Safety of Smart Contracts
Online Detection of Effectively Callback Free Objects with
11:00 - 12:30: OOPSLA - Repair & Transformation at Templars
Chair(s): Bor-Yuh Evan Chang University of Colorado Boulder | Amazon
Tomorrow
11:00 - 11:22
Talk
Detecting Nondeterministic Payment Bugs in Ethereum Smart Contracts
Shuai Wang ETH Zurich, Chengyu Zhang East China Normal University, Zhendong Su ETH Zurich
DOI
11:22 - 11:45
Talk
Automatic Repair of Regular Expressions
Rong Pan University of Texas at Austin, Qinheping Hu University of Wisconsin, Madison, Gaowei Xu University of Wisconsin Madison, Loris D'Antoni University of Wisconsin Madison
DOI
Pre-print
Exploiting The Laws of Order in Smart Contracts
Aashish Kolluri
Ivica Nikolić
Ilya Sergey
Prateek Saxena
VULTRON: Catching Vulnerable Smart Contracts Once and for All

The Challenge

Preventing smart contract vulnerabilities
with principled Programming Language design

Wishlist

- Explicit interaction: no reentrancy attacks
- Minimalistic
- Explicit control of effects
- Expressive
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

The Givens of Smart Contracts

~~Deployed in a *low level language*~~

~~Must be *Turing-complete*~~

Code is law (so it should be easy to interpret) arXiv, Jan 2018

SCILLA: a Smart Contract Intermediate-Level LAnguage

Automata for Smart Contract Implementation and Verification

Ilya Sergey
University College London
i.sergey@ucl.ac.uk

Amrit Kumar
National University of Singapore
amrit@comp.nus.edu.sg

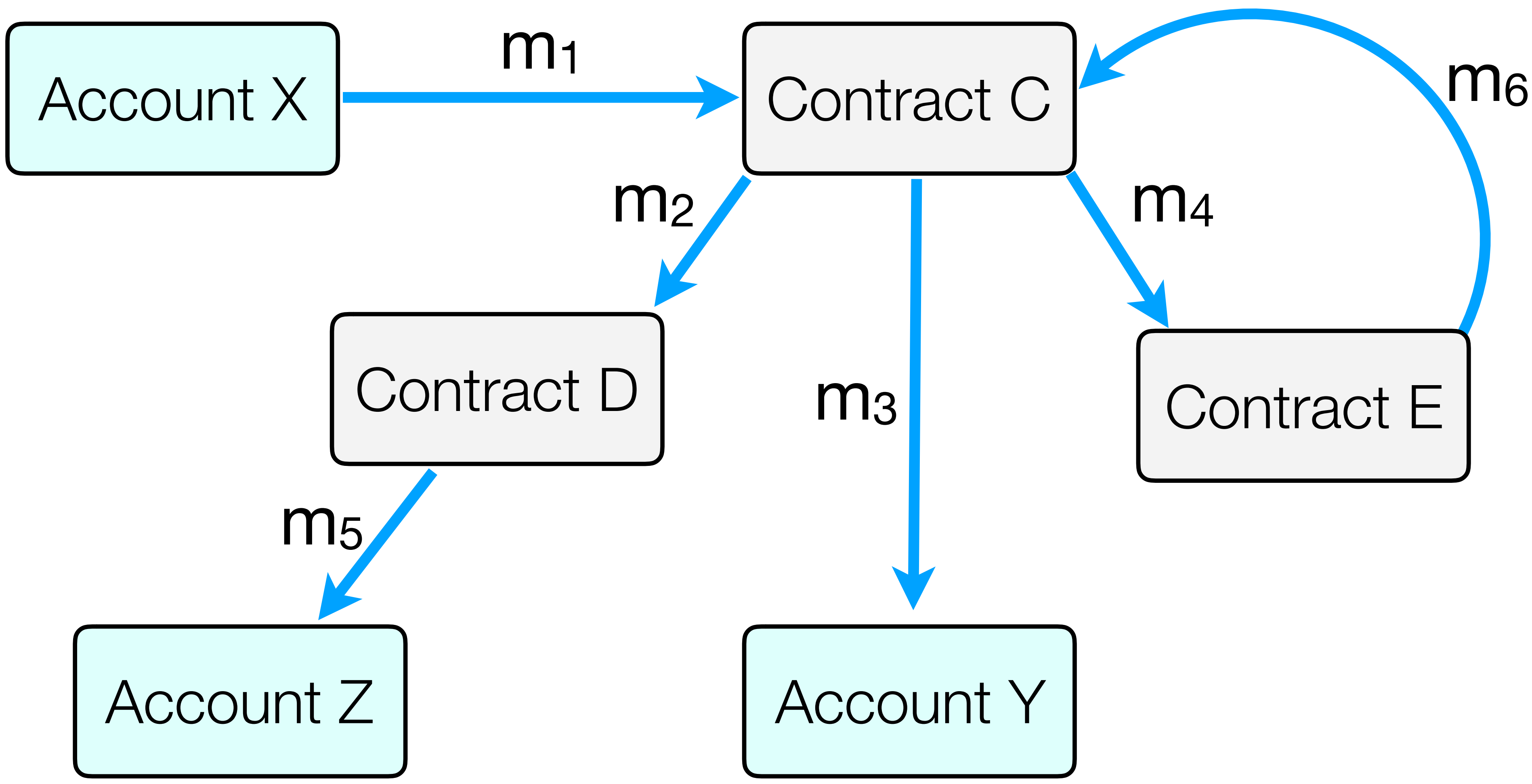
Aquinas Hobor
Yale-NUS College
National University of Singapore
hobor@comp.nus.edu.sg

Simple computation model	**System F** with small extensions
Not Turing-complete	Only *primitive recursion*/iteration
Explicit Effects	*State-transformer* semantics
Communication	Contracts are Autonomous Actors

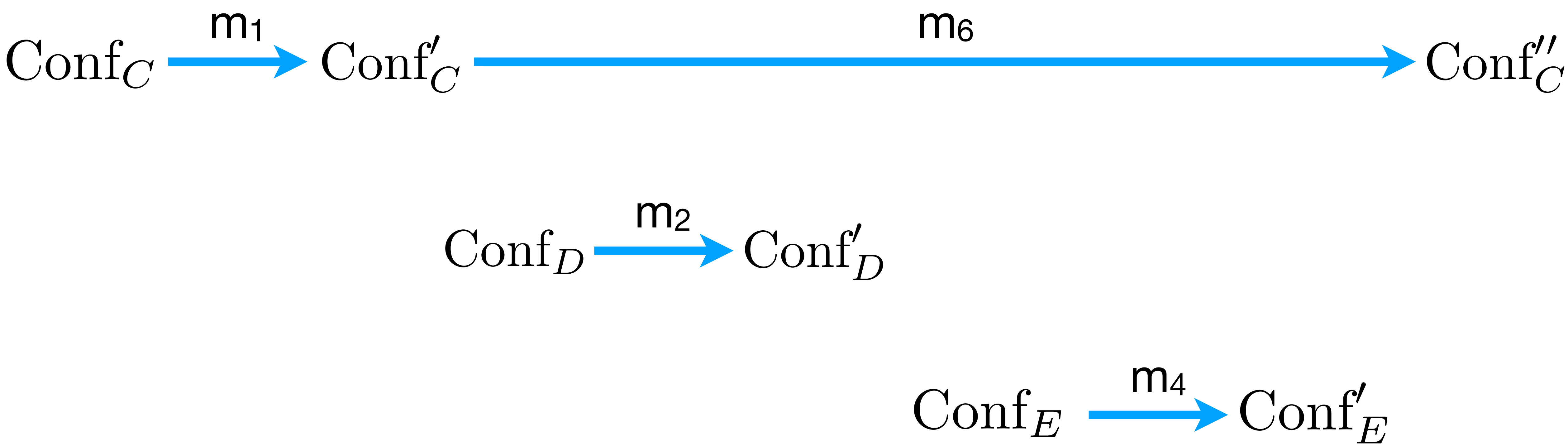
Smart Contracts as Autonomous Actors

Scilla Contract Execution Model

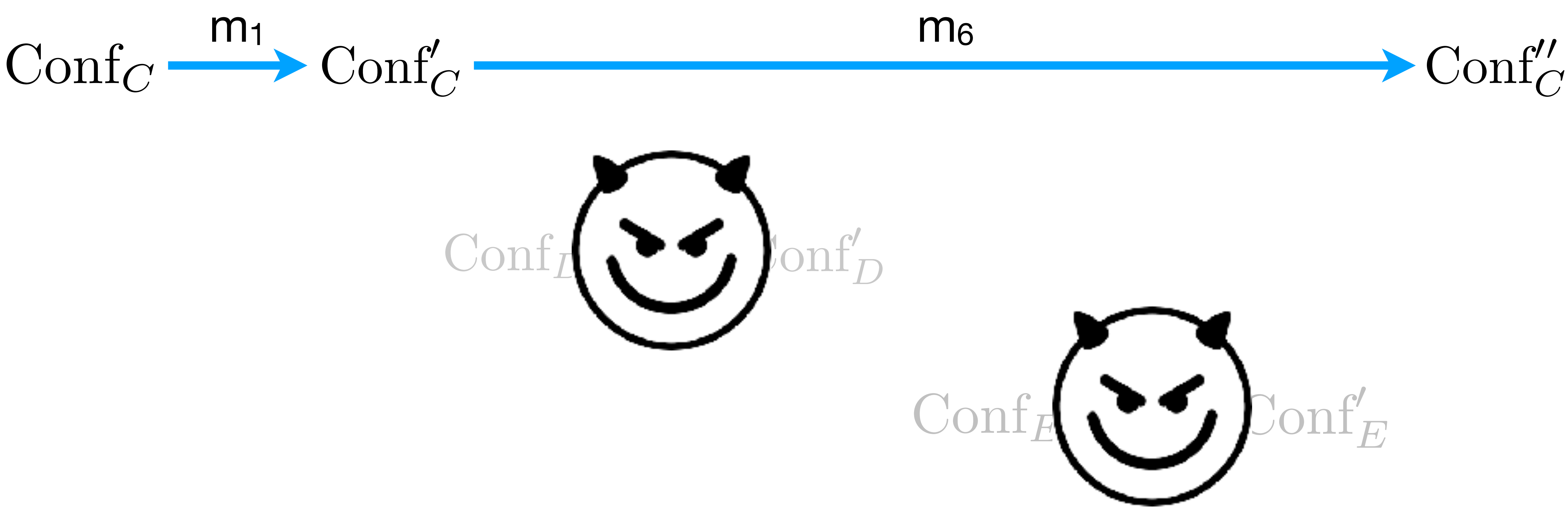
Scilla Contract Execution Model



Scilla Contract Execution Model



Scilla Contract Execution Model



Wishlist

- Explicit interaction: no reentrancy attacks
- Minimalistic
- Explicit control of effects
- Expressive
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- Minimalistic
- Explicit control of effects
- Expressive
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- Minimalistic
- Explicit control of effects
- Expressive
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Types

(signed integers)	int	::=	i32 | i64 | i128 | i256
(unsigned integers)	$uint$	::=	u32 | u64 | u128 | u256
(byte strings)	bst	::=	bystrx n | bystr
(primitive types)	pt	::=	int | $uint$ | bst |
			string | bnum | msg
(algebraic types)	$\mathcal{D}$	::=	unit | bool | nat | option |
			pair | list | U
(general Types)	t	::=	pt | map t t | $t \rightarrow t$ |
			$\mathcal{D}\ \overline{t}$ | α | forall α. t

Expressions (pure)

Expression	e	::=	f	simple expression
			let x ⟨: T⟩ = f **in** e	let-form
Simple expression	f	::=	l	primitive literal
			x	variable
			{ ⟨$entry$⟩$_k$ }	Message
			fun (x : T) => e	function
			builtin b ⟨x_k⟩	built-in application
			x ⟨x_k⟩	application
			tfun α => e	type function
			@x T	type instantiation
			C ⟨ {⟨T_k⟩} ⟩ ⟨x_k⟩	constructor instantiation
			match x **with** ⟨ | sel_k ⟩ **end**	pattern matching
Selector	sel	::=	pat => e	
Pattern	pat	::=	x	variable binding
			C ⟨pat_k⟩	constructor pattern
			(pat)	paranthesized pattern
			_	wildcard pattern
Message entrry	$entry$	::=	b : x	
Name	b			identifier

Structural Recursion in Scilla

Natural numbers (not Ints!)

```
nat_rec : forall α.α -> (nat -> α -> α) -> nat -> α
```

Result type

Value for 0

constructing the next value

number of iterations

final result

Example: Fibonacci Numbers

```
let fib = fun (n : Nat) =>
  let iter_nat = @ nat_rec (Pair Int Int) in
  let iter_fun =
    fun (n: Nat) => fun (res : Pair Int Int) =>
      match res with
      | And x y => let z = builtin add x y in
                   And {Int Int} z x
      end
    in
  let zero = 0 in
  let one = 1 in
  let init_val = And {Int Int} one zero in
  let res = iter_nat init_val iter_fun n in
  fst res
```

Statements (effectful)

`s ::=`	`x <- f`	read from mutable field
	`f := x`	store to a field
	`x = e`	assign a pure expression
	`match` `x` **`with`** `⟨pat => s⟩` **`end`**	pattern matching and branching
	`x <- &B`	read from blockchain state
	`accept`	accept incoming payment
	`event` `m`	create a single event
	`send` `ms`	send list of messages
	`throw`	abort the execution
	in-place map operations	efficient manipulation with maps

Statements (effectful)

`s ::=`	`x <- f`	read from mutable field
	`f := x`	store to a field
	`x = e`	assign a pure expression
	`match x with ⟨pat => s⟩ end`	pattern matching and branching
	`x <- &B`	read from blockchain state
	`accept`	accept incoming payment
	`event m`	create a single event
	`send ms`	send list of messages
	`throw`	abort the execution
	in-place map operations	efficient manipulation with maps

Statements (effectful)

`s ::=`	`x <- f`	read from mutable field
	`f := x`	store to a field
	`x = e`	assign a pure expression
	match x **with** ⟨pat => s⟩ **end**	pattern matching and branching
	`x <- &B`	read from blockchain state
	accept	accept incoming payment
	event m	create a single event
	send ms	send list of messages
	throw	abort the execution
	in-place map operations	efficient manipulation with maps

Statements (effectful)

`s ::=`	`x <- f`	read from mutable field
	`f := x`	store to a field
	`x = e`	assign a pure expression
	`match x with ⟨pat => s⟩ end`	pattern matching and branching
	`x <- &B`	read from blockchain state
	`accept`	accept incoming payment
	`event m`	create a single event
	`send ms`	send list of messages
	`throw`	abort the execution
	in-place map operations	efficient manipulation with maps

Statements (effectful)

`s ::=`	`x <- f`	read from mutable field
	`f := x`	store to a field
	`x = e`	assign a pure expression
	`match` `x` **`with`** `⟨pat => s⟩` **`end`**	pattern matching and branching
	`x <- &B`	read from blockchain state
	`accept`	accept incoming payment
	`event` `m`	create a single event
	`send` `ms`	send list of messages
	`throw`	abort the execution
	`in-place map operations`	efficient manipulation with maps

Statements (effectful)

`s ::=`	`x <- f`	read from mutable field
	`f := x`	store to a field
	`x = e`	assign a pure expression
	`match` `x` **`with`** `⟨pat => s⟩` **`end`**	pattern matching and branching
	`x <- &B`	read from blockchain state
	`accept`	accept incoming payment
	`event` `m`	create a single event
	`send` `ms`	send list of messages
	`throw`	abort the execution
	in-place map operations	efficient manipulation with maps

Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- Minimalistic
- Explicit control of effects
- Expressive
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Wishlist

✔
- Explicit interaction: no reentrancy attacks
- Minimalistic (core interpreter ~200 LOC of OCaml)
- Explicit control of effects
- Expressive
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- Explicit control of effects
- Expressive
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- Explicit control of effects
- Expressive
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Contract Structure

```
library Crowdfunding
(*  Map ByStr20 Uint128 → ByStr20 → Uint128 → *)
(*  Option (Map ByStr20 Uint128)               *)
let check_update = (* ... *)
(*  BNum → BNum → Bool   *)
let blk_leq = (* ... *)

contract Crowdfunding
(* Immutable parameters *)
(owner : ByStr20, max_block : BNum, goal : Uint128)
(* Mutable fields *)
field backers : Map ByStr20 Uint128 = Emp ByStr20 Uint128
field funded : Bool = False
(* Transitions *)
transition Donate (sender : ByStr20, amount : Uint128)
transition GetFunds (sender : ByStr20, amount : Uint128)
transition ClaimBack (sender : ByStr20, amount : Uint128)
```

Contract Structure

```
library Crowdfunding
(*  Map ByStr20 Uint128 ⟶ ByStr20 ⟶ Uint128 ⟶ *)
(*  Option (Map ByStr20 Uint128)              *)
let check_update = (* ... *)
(*  BNum ⟶ BNum ⟶ Bool  *)
let blk_leq = (* ... *)

contract Crowdfunding
(* Immutable parameters *)
(owner : ByStr20, max_block : BNum, goal : Uint128)
(* Mutable fields *)
field backers : Map ByStr20 Uint128 = Emp ByStr20 Uint128
field funded : Bool = False
(* Transitions *)
transition Donate (sender : ByStr20, amount : Uint128)
transition GetFunds (sender : ByStr20, amount : Uint128)
transition ClaimBack (sender : ByStr20, amount : Uint128)
```

Contract Structure

```
library Crowdfunding
(*   Map ByStr20 Uint128 ⟶ ByStr20 ⟶ Uint128 ⟶ *)
(*   Option (Map ByStr20 Uint128)               *)
let check_update = (* ... *)
(*   BNum ⟶ BNum ⟶ Bool   *)
let blk_leq = (* ... *)

contract Crowdfunding
(* Immutable parameters *)
(owner : ByStr20, max_block : BNum, goal : Uint128)
(* Mutable fields *)
field backers : Map ByStr20 Uint128 = Emp ByStr20 Uint128
field funded : Bool = False
(* Transitions *)
transition Donate (sender : ByStr20, amount : Uint128)
transition GetFunds (sender : ByStr20, amount : Uint128)
transition ClaimBack (sender : ByStr20, amount : Uint128)
```

Contract Structure

```
library Crowdfunding
(*  Map ByStr20 Uint128 ⟶ ByStr20 ⟶ Uint128 ⟶ *)
(*  Option (Map ByStr20 Uint128)                *)
let check_update = (* ... *)
(*  BNum ⟶ BNum ⟶ Bool   *)
let blk_leq = (* ... *)

contract Crowdfunding
(* Immutable parameters *)
(owner : ByStr20, max_block : BNum, goal : Uint128)
(* Mutable fields *)
field backers : Map ByStr20 Uint128 = Emp ByStr20 Uint128
field funded : Bool = False
(* Transitions *)
transition Donate (sender : ByStr20, amount : Uint128)
transition GetFunds (sender : ByStr20, amount : Uint128)
transition ClaimBack (sender : ByStr20, amount : Uint128)
```

Contract Structure

```
library Crowdfunding
(*  Map ByStr20 Uint128 ⟶ ByStr20 ⟶ Uint128 ⟶ *)
(*  Option (Map ByStr20 Uint128)               *)
let check_update = (* ... *)
(*  BNum ⟶ BNum ⟶ Bool   *)
let blk_leq = (* ... *)

contract Crowdfunding
(* Immutable parameters *)
(owner : ByStr20, max_block : BNum, goal : Uint128)
(* Mutable fields *)
field backers : Map ByStr20 Uint128 = Emp ByStr20 Uint128
field funded : Bool = False
(* Transitions *)
transition Donate (sender : ByStr20, amount : Uint128)
transition GetFunds (sender : ByStr20, amount : Uint128)
transition ClaimBack (sender : ByStr20, amount : Uint128)
```

Contract Structure

```
library Crowdfunding
(*  Map ByStr20 Uint128 ⟶ ByStr20 ⟶ Uint128 ⟶ *)
(*  Option (Map ByStr20 Uint128)              *)
let check_update = (* ... *)
(*  BNum ⟶ BNum ⟶ Bool  *)
let blk_leq = (* ... *)

contract Crowdfunding
(* Immutable parameters *)
(owner : ByStr20, max_block : BNum, goal : Uint128)
(* Mutable fields *)
field backers : Map ByStr20 Uint128 = Emp ByStr20 Uint128
field funded : Bool = False
(* Transitions *)
transition Donate (sender : ByStr20, amount : Uint128)
transition GetFunds (sender : ByStr20, amount : Uint128)
transition ClaimBack (sender : ByStr20, amount : Uint128)
```

```
transition Donate (sender: ByStr20, amount: Uint128)
  blk <- & BLOCKNUMBER;
  in_time = blk_leq blk max_block;
  match in_time with
  | True  =>
    bs  <- backers;
    res = check_update bs sender amount;
    match res with
    | None =>
      msg  = {tag : Main; to : sender; amount : 0; code : already_backed};
      msgs = one_msg msg;
      send msgs
    | Some bs1 =>
      backers := bs1;
      accept;
      msg  = {tag : Main; to : sender; amount : 0; code : accepted_code};
      msgs = one_msg msg;
      send msgs
     end
  | False =>
    msg  = {tag : Main; to : sender; amount : 0; code : missed_dealine};
    msgs = one_msg msg;
    send msgs
  end
end
```

```
transition Donate (sender: ByStr20, amount: Uint128)
  blk <- & BLOCKNUMBER;
  in_time = blk_leq blk max_block;
  match in_time with
  | True  =>
    bs  <- backers;
    res = check_update bs sender amount;
    match res with
    | None =>
      msg  = {tag : Main; to : sender; amount : 0; code : already_backed};
      msgs = one_msg msg;
      send msgs
    | Some bs1 =>
      backers := bs1;
      accept;
      msg  = {tag : Main; to : sender; amount : 0; code : accepted_code};
      msgs = one_msg msg;
      send msgs
     end
  | False =>
    msg  = {tag : Main; to : sender; amount : 0; code : missed_dealine};
    msgs = one_msg msg;
    send msgs
  end
end
```

Structure of the incoming message

```
transition Donate (sender: ByStr20, amount: Uint128)
  blk <- & BLOCKNUMBER;
  in_time = blk_leq blk max_block;
  match in_time with
  | True  =>
    bs  <- backers;
    res = check_update bs sender amount;
    match res with
    | None =>
      msg  = {tag : Main; to : sender; amount : 0; code : already_backed};
      msgs = one_msg msg;
      send msgs
    | Some bs1 =>
      backers := bs1;
      accept;
      msg  = {tag : Main; to : sender; amount : 0; code : accepted_code};
      msgs = one_msg msg;
      send msgs
     end
  | False =>
    msg  = {tag : Main; to : sender; amount : 0; code : missed_dealine};
    msgs = one_msg msg;
    send msgs
  end
end
```

Using pure library functions
(defined above in the contract)

```
transition Donate (sender: ByStr20, amount: Uint128)
  blk <- & BLOCKNUMBER;
  in_time = blk_leq blk max_block;
  match in_time with
  | True  =>
    bs  <- backers;
    res = check_update bs sender amount;
    match res with
    | None =>
      msg  = {tag : Main; to : sender; amount : 0; code : already_backed};
      msgs = one_msg msg;
      send msgs
    | Some bs1 =>
      backers := bs1;
      accept;
      msg  = {tag : Main; to : sender; amount : 0; code : accepted_code};
      msgs = one_msg msg;
      send msgs
     end
  | False =>
    msg  = {tag : Main; to : sender; amount : 0; code : missed_dealine};
    msgs = one_msg msg;
    send msgs
  end
end
```

Reading from blockchain state

Manipulating with fields

```
transition Donate (sender: ByStr20, amount: Uint128)
  blk <- & BLOCKNUMBER;
  in_time = blk_leq blk max_block;
  match in_time with
  | True  =>
    bs  <- backers;
    res = check_update bs sender amount;
    match res with
    | None =>
      msg  = {tag : Main; to : sender; amount : 0; code : already_backed};
      msgs = one_msg msg;
      send msgs
    | Some bs1 =>
      backers := bs1;
      accept;
      msg  = {tag : Main; to : sender; amount : 0; code : accepted_code};
      msgs = one_msg msg;
      send msgs
     end
  | False =>
    msg  = {tag : Main; to : sender; amount : 0; code : missed_dealine};
    msgs = one_msg msg;
    send msgs
  end
end
```

```
transition Donate (sender: ByStr20, amount: Uint128)
  blk <- & BLOCKNUMBER;
  in_time = blk_leq blk max_block;
  match in_time with
  | True  =>
    bs  <- backers;
    res = check_update bs sender amount;
    match res with
    | None =>
      msg  = {tag : Main; to : sender; amount : 0; code : already_backed};
      msgs = one_msg msg;
      send msgs
    | Some bs1 =>
      backers := bs1;
      accept;
      msg  = {tag : Main; to : sender; amount : 0; code : accepted_code};
      msgs = one_msg msg;
      send msgs
     end
  | False =>
    msg  = {tag : Main; to : sender; amount : 0; code : missed_dealine};
    msgs = one_msg msg;
    send msgs
  end
end
```

Explicitly accepting incoming funds

Creating and sending messages

```
transition Donate (sender: ByStr20, amount: Uint128)
  blk <- & BLOCKNUMBER;
  in_time = blk_leq blk max_block;
  match in_time with
  | True  =>
    bs  <- backers;
    res = check_update bs sender amount;
    match res with
    | None =>
      msg  = {tag : Main; to : sender; amount : 0; code : already_backed};
      msgs = one_msg msg;
      send msgs
    | Some bs1 =>
      backers := bs1;
      accept;
      msg  = {tag : Main; to : sender; amount : 0; code : accepted_code};
      msgs = one_msg msg;
      send msgs
     end
  | False =>
    msg  = {tag : Main; to : sender; amount : 0; code : missed_dealine};
    msgs = one_msg msg;
    send msgs
  end
end
```

Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- Explicit control of effects
- Expressive
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- Explicit control of effects (eg, acceptance of funds)
- Expressive
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

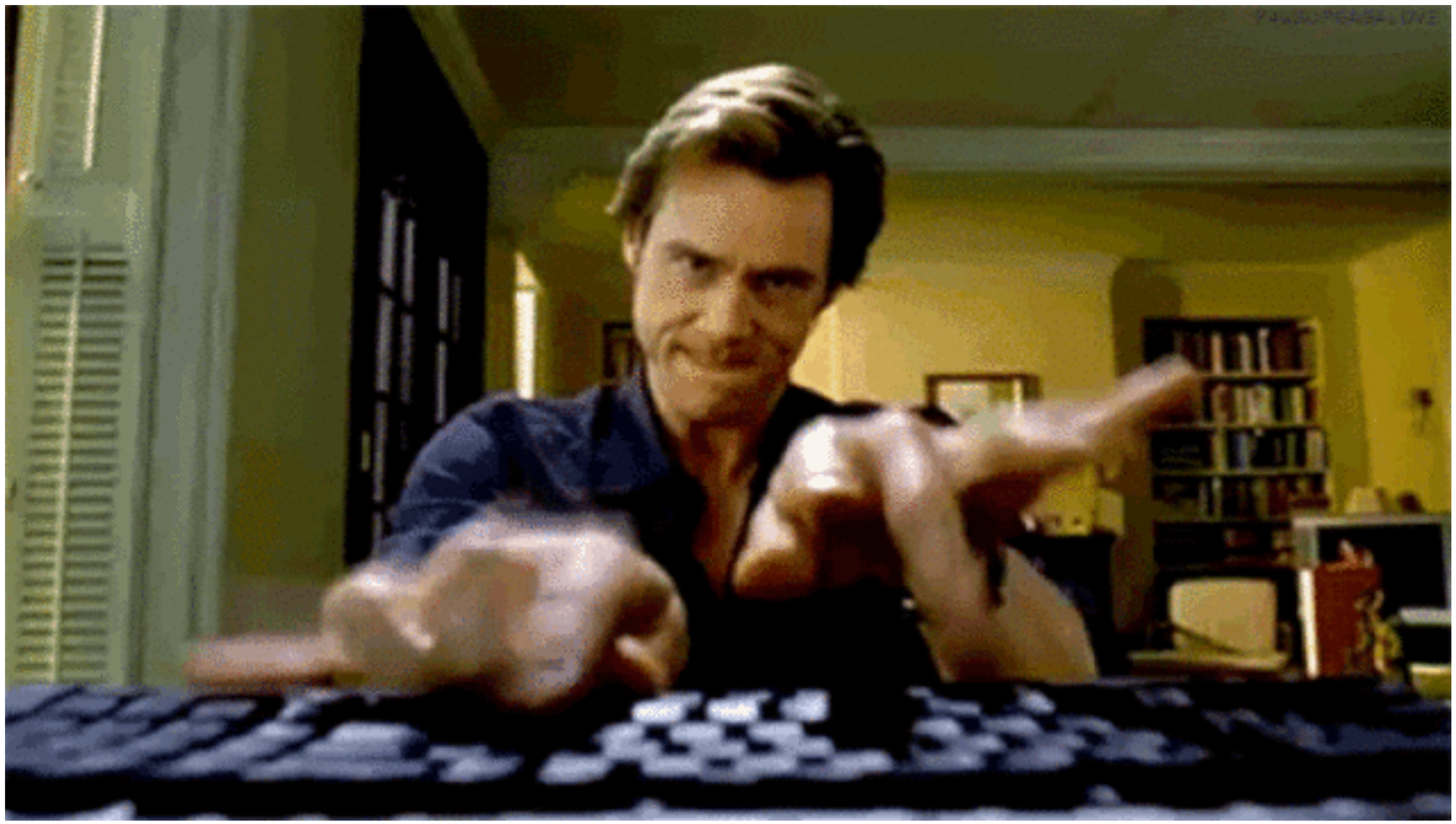
Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- ✔ Explicit control of effects (eg, acceptance of funds)
- Expressive
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Wishlist

- Explicit interaction: no reentrancy attacks
- Minimalistic (core interpreter ~200 LOC of OCaml)
- Explicit control of effects (eg, acceptance of funds)
- Expressive
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Expressivity



Expressivity

- Standard Library: ~1 kLOC

Contract	LOC	#Lib	#Trans
HelloWorld	31	3	2
Crowdfunding	127	13	3
Auction	140	11	3
ERC20	158	2	6
ERC721	270	15	6
Wallet	363	28	9
Bookstore	123	6	3
HashGame	209	16	3
Schnorr	71	2	3

Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- ✔ Explicit control of effects (eg, acceptance of funds)
- Expressive
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Wishlist

- Explicit interaction: no reentrancy attacks
- Minimalistic (core interpreter ~200 LOC of OCaml)
- Explicit control of effects (eg, acceptance of funds)
- Expressive (suitable for all scenarios of interest)
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- ✔ Explicit control of effects (eg, acceptance of funds)
- ✔ Expressive (suitable for all scenarios of interest)
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- ✔ Explicit control of effects (eg, acceptance of funds)
- ✔ Expressive (suitable for all scenarios of interest)
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Verification-Friendliness

- A framework for staged static analyses (optional)
- Two instances:
  - Gas-Usage Analysis
  - Cash-Flow Analysis

Verification-Friendliness

- A framework for staged static analyses (optional)
- Two instances:
  - Gas-Usage Analysis (resources)
  - Cash-Flow Analysis (data flow)

Verification-Friendliness

- A framework for staged static analyses (optional)
- Two instances:
  - Gas-Usage Analysis (resources)
  - Cash-Flow Analysis (data flow)

Cash-Flow Analysis

```
contract Crowdfunding
(* Immutable parameters *)
(owner : ByStr20, max_block : BNum  goal : Uint128)
(* Mutable fields *)
field backers : Map ByStr20 Uint128 = Emp ByStr20 Uint128
field funded : Bool = False
(* Transitions *)
transition Donate (sender : ByStr20, amount : Uint128)
transition GetFunds (sender : ByStr20, amount : Uint128)
transition ClaimBack (sender : ByStr20, amount : Uint128)
```

Which of those correspond to currency?

Cash-Flow Analysis

- Soundly infers what fields *represent money*
- Based on simple abstract interpretation
- Takes user annotations for *custom tokens*

Lattice of Cash Tags

$$\begin{array}{lcl} \tau & ::= & \textbf{Money} \mid \textbf{NotMoney} \mid \textbf{Map}\ \tau \mid t\ \overline{\tau} \mid \top \mid \bot \\ t & ::= & \textbf{Option} \mid \textbf{Pair} \mid \textbf{List} \mid \ \ldots \end{array}$$

$$\begin{array}{lrcll} \text{(maps)} & \textbf{Map}\ \tau & \sqsubseteq & \textbf{Map}\ \tau' & \text{iff } \tau \sqsubseteq \tau' \\ \text{(algebraic types)} & t\ \overline{\tau} & \sqsubseteq & t'\ \overline{\tau'} & \text{iff } t = t' \text{ and } \tau_i \sqsubseteq \tau'_i \text{ for all } i \\ \text{(bottom)} & \bot & \sqsubseteq & \tau & \text{for all } \tau \\ \text{(top)} & \tau & \sqsubseteq & \top & \text{for all } \tau \end{array}$$

Cash-Flow Analysis

- Soundly infers what fields *represent money*
- Based on simple abstract interpretation
- Takes user annotations for *custom tokens*

```
contract Crowdfunding
(* Immutable parameters *)
(owner : ByStr20, max_block : BNum, goal : Uint128)
(* Mutable fields *)
field backers : Map ByStr20 Uint128 = Emp ByStr20 Uint128
field funded : Bool = False
(* Transitions *)
transition Donate (sender : ByStr20, amount : Uint128)
transition GetFunds (sender : ByStr20, amount : Uint128)
transition ClaimBack (sender : ByStr20, amount : Uint128)
```

Cash-Flow Analysis

Contract	LOC	#Lib	#Trans
HelloWorld	31	3	2
Crowdfunding	127	13	3
Auction	140	11	3
ERC20	158	2	6
ERC721	270	15	6
Wallet	363	28	9
Bookstore	123	6	3
HashGame	209	16	3
Schnorr	71	2	3

Cash-Flow Analysis

Contract	LOC	#Lib	#Trans	$-Flow
HelloWorld	31	3	2	✓
Crowdfunding	127	13	3	✓
Auction	140	11	3	✓
ERC20	158	2	6	✓*
ERC721	270	15	6	$✓_\perp$
Wallet	363	28	9	✓
Bookstore	123	6	3	✓
HashGame	209	16	3	✓
Schnorr	71	2	3	✓

Cash-Flow Analysis

Contract	LOC	#Lib	#Trans	$-Flow
HelloWorld	31	3	2	✓
Crowdfunding	127	13	3	✓
Auction	140	11	3	✓
ERC20	158	2	6	$\checkmark^*$
ERC721	270	15	6	$\checkmark_\perp$
Wallet	363	28	9	✓
Bookstore	123	6	3	✓
HashGame	209	16	3	✓
Schnorr	71	2	3	✓

non-native tokens

Cash-Flow Analysis

Contract	LOC	#Lib	#Trans	$-Flow
HelloWorld	31	3	2	✓
Crowdfunding	127	13	3	✓
Auction	140	11	3	✓
ERC20	158	2	6	✓*
ERC721	270	15	6	$\checkmark_{\perp}$
Wallet	363	28	9	✓
Bookstore	123	6	3	✓
HashGame	209	16	3	✓
Schnorr	71	2	3	✓

non-fungible tokens

Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- ✔ Explicit control of effects (eg, acceptance of funds)
- ✔ Expressive (suitable for all scenarios of interest)
- Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- ✔ Explicit control of effects (eg, acceptance of funds)
- ✔ Expressive (suitable for all scenarios of interest)
- ✔ Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- ✔ Explicit control of effects (eg, acceptance of funds)
- ✔ Expressive (suitable for all scenarios of interest)
- ✔ Analysis/Verification friendly
- Predictable resource (gas) consumption
- Reasonable performance

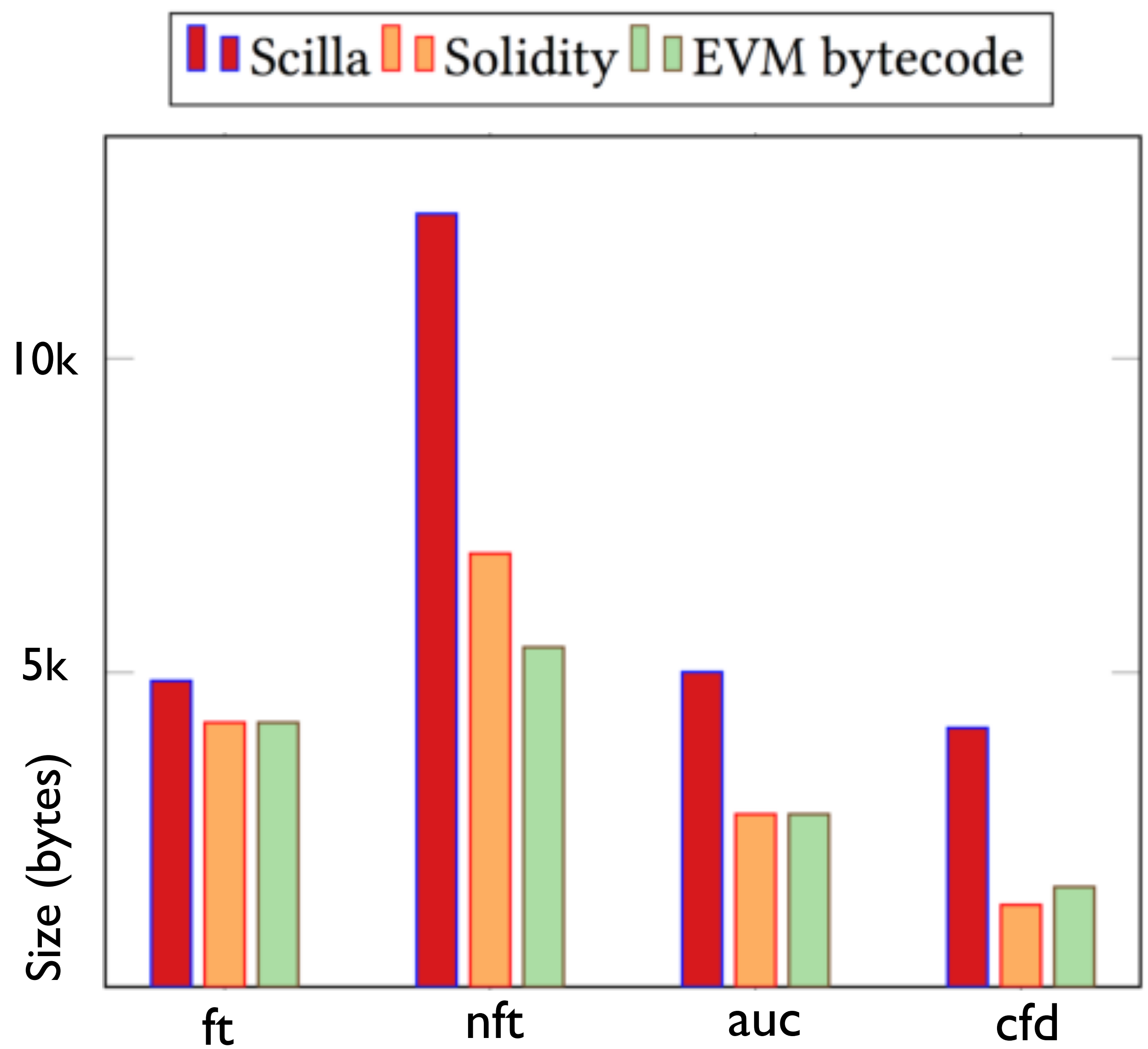
Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- ✔ Explicit control of effects (eg, acceptance of funds)
- ✔ Expressive (suitable for all scenarios of interest)
- ✔ Analysis/Verification friendly
- ✔ Predictable resource (gas) consumption
- Reasonable performance

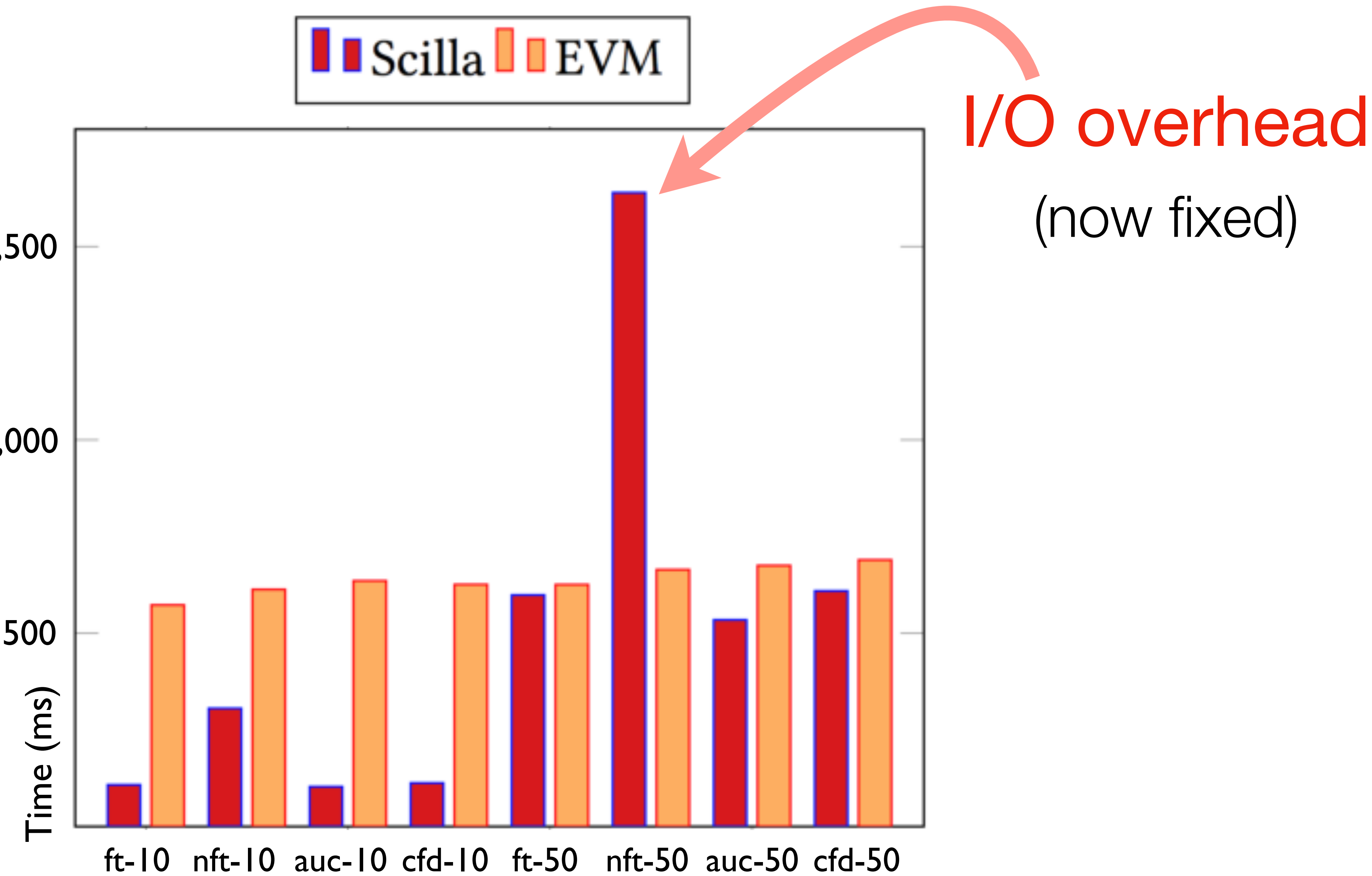
Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- ✔ Explicit control of effects (eg, acceptance of funds)
- ✔ Expressive (suitable for all scenarios of interest)
- ✔ Analysis/Verification friendly
- ✔ Predictable resource (gas) consumption
- Reasonable performance

Relative Code Size



Performance



Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- ✔ Explicit control of effects (eg, acceptance of funds)
- ✔ Expressive (suitable for all scenarios of interest)
- ✔ Analysis/Verification friendly
- ✔ Predictable resource (gas) consumption
- Reasonable performance

Wishlist

- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- ✔ Explicit control of effects (eg, acceptance of funds)
- ✔ Expressive (suitable for all scenarios of interest)
- ✔ Analysis/Verification friendly
- ✔ Predictable resource (gas) consumption
- Reasonable performance (in a ballpark of EVM)

Wishlist

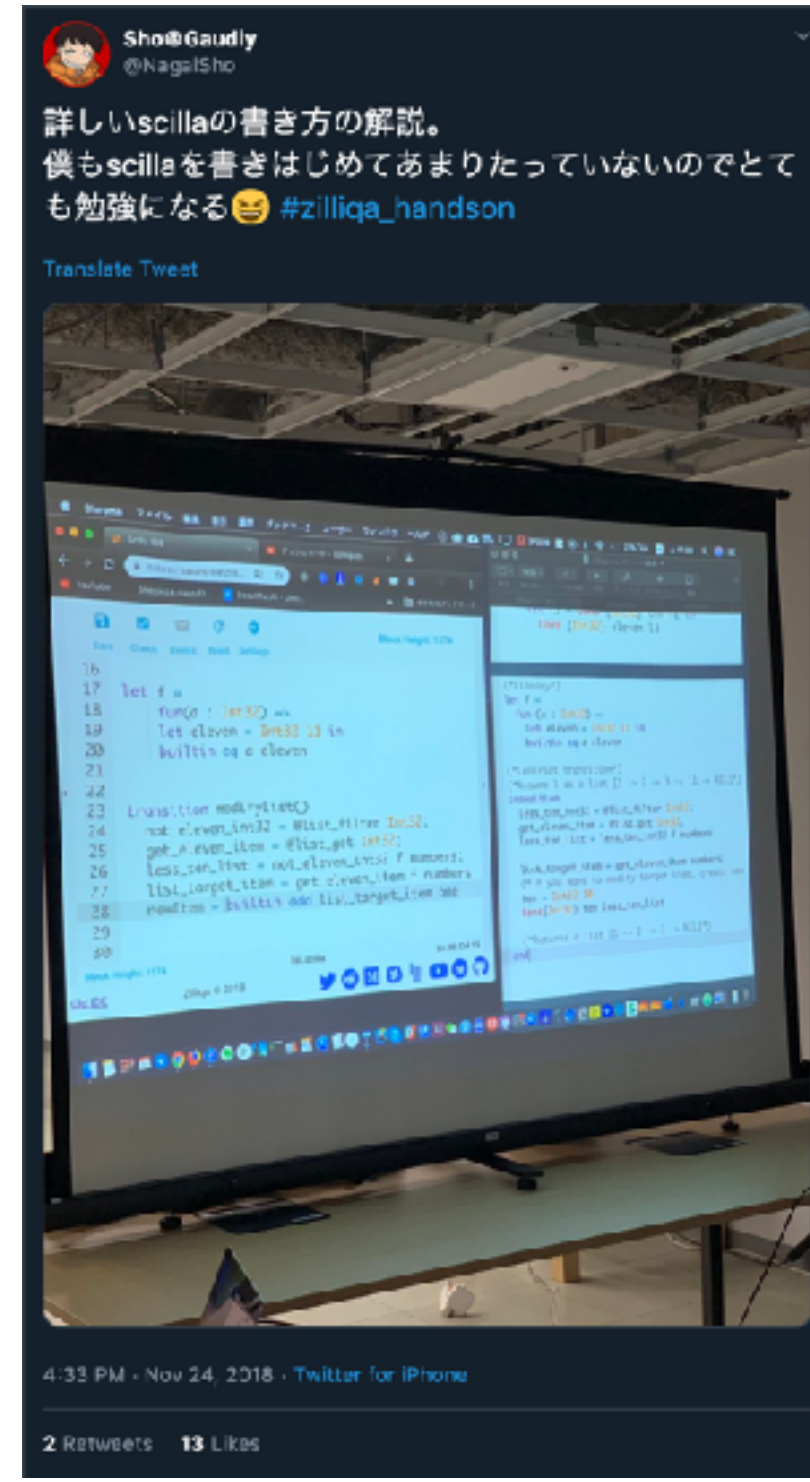
- ✔ Explicit interaction: no reentrancy attacks
- ✔ Minimalistic (core interpreter ~200 LOC of OCaml)
- ✔ Explicit control of effects (eg, acceptance of funds)
- ✔ Expressive (suitable for all scenarios of interest)
- ✔ Analysis/Verification friendly
- ✔ Predictable resource (gas) consumption
- ✔ Reasonable performance (in a ballpark of EVM)

Wishlist

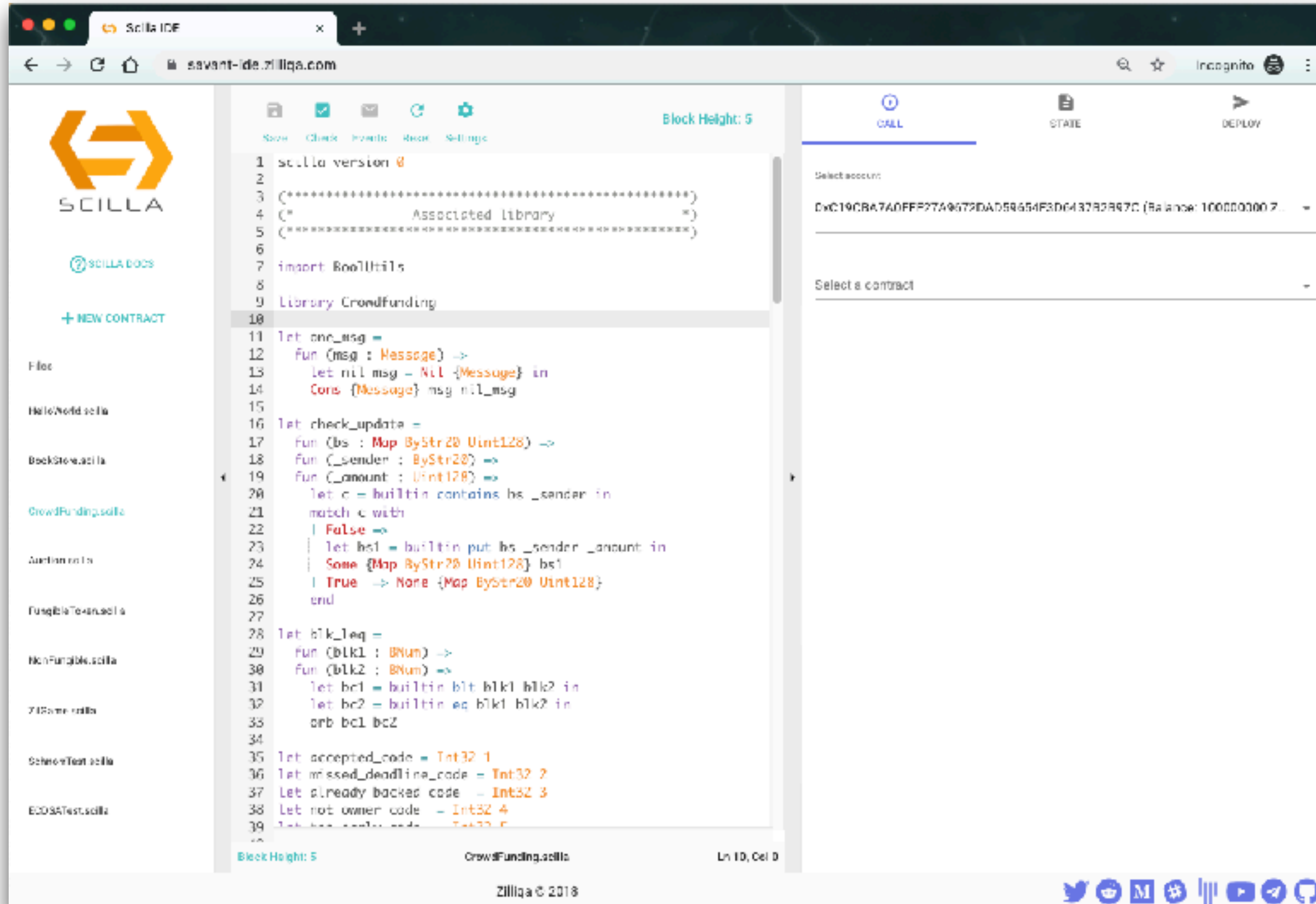
- Explicit interaction: no reentrancy attacks
- Minimalisti … C of OCaml)
- Explicit co … nce of funds)
- Expressive … f interest)
- Analysis/Ve …
- Predictable … on
- Reasonable performance (in a ballpark of EVM)

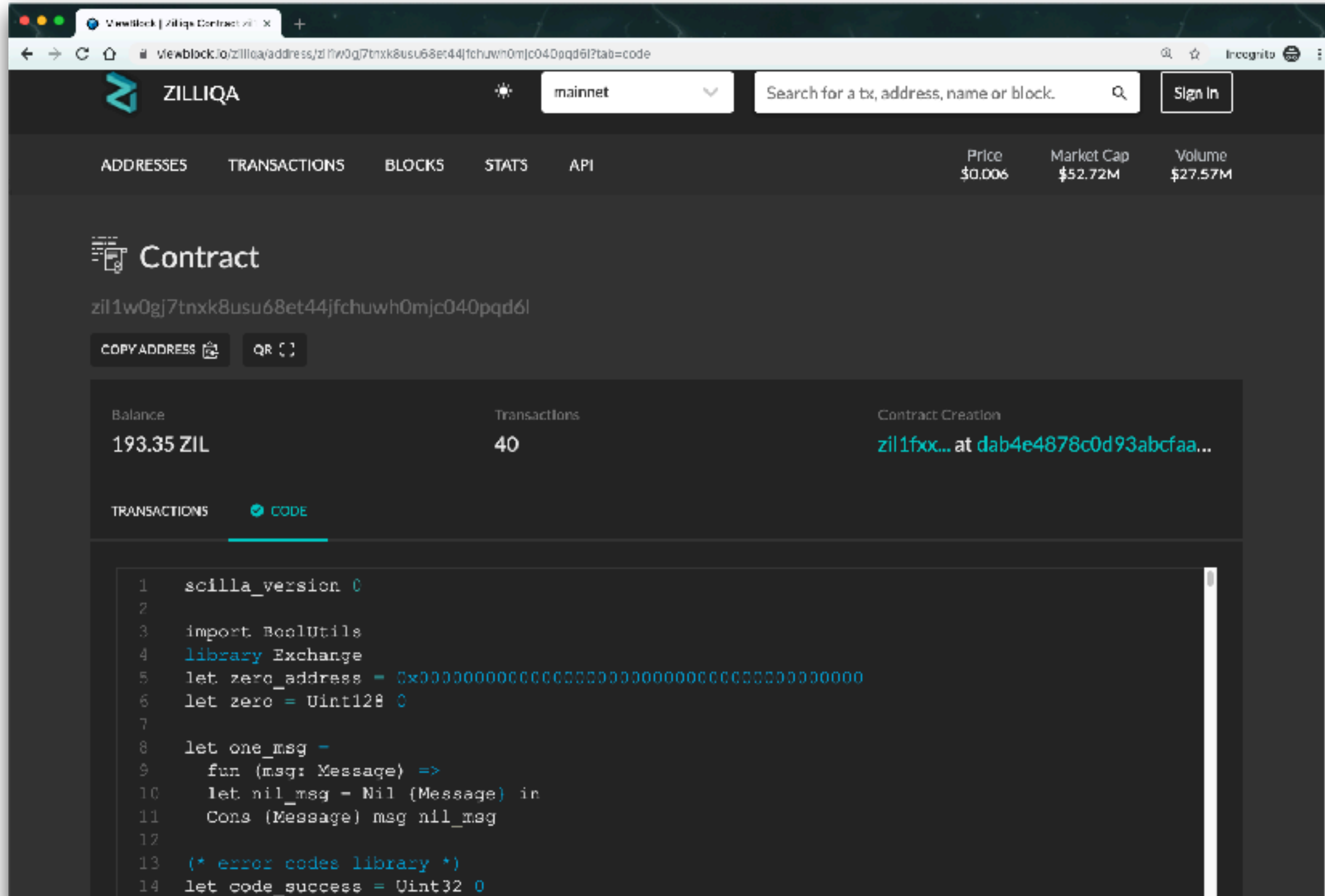
SCILLA

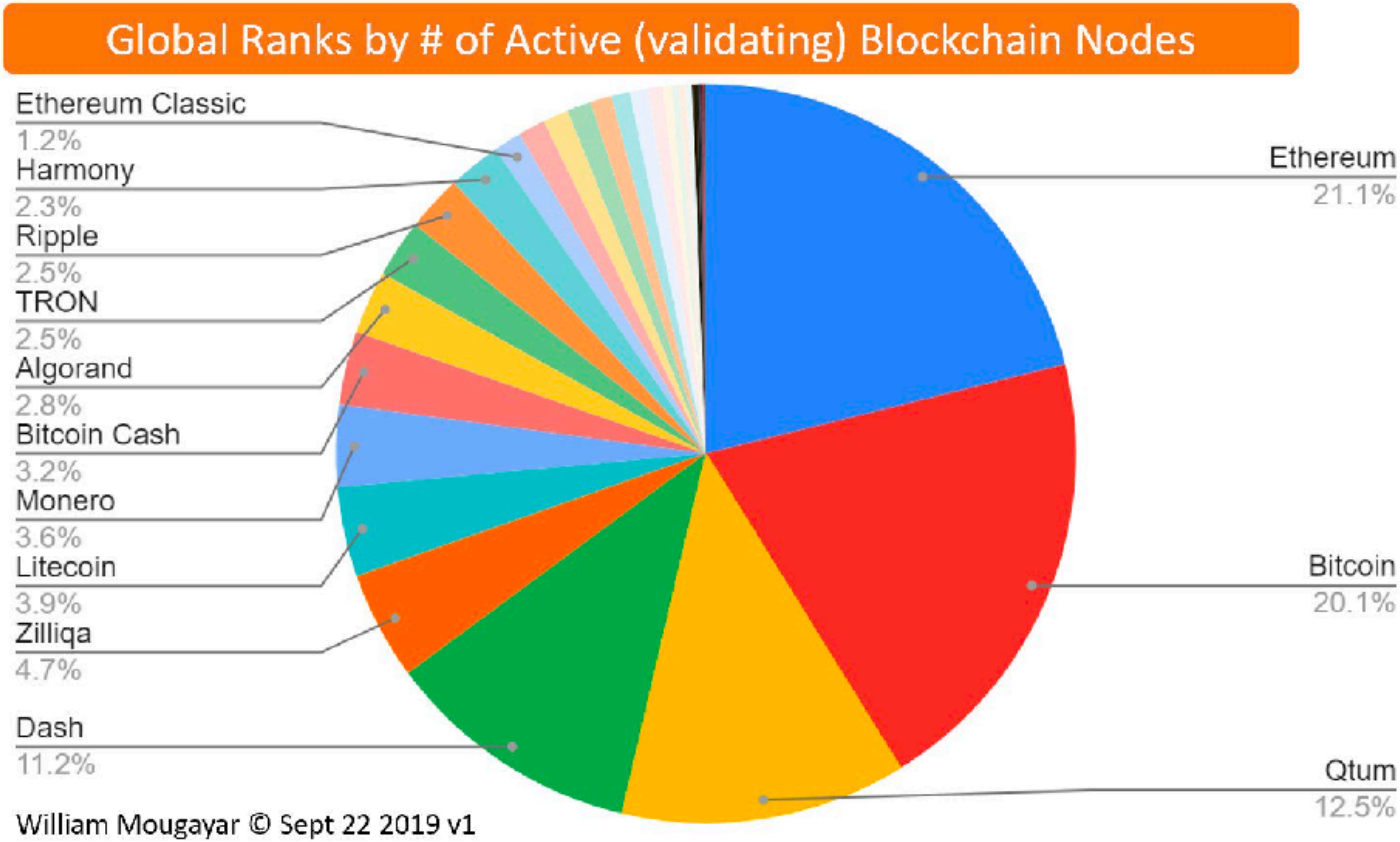
Adoption



- Scilla launched on Zilliqa test-net in June 2018, on main-net since June 2019
- Dozens of community-contributed contracts:
  - ERC223, ERC777
  - contracts for crowdsales, escrows
  - contracts for access control
  - upcoming standard ERC1404 for security tokens
- Language-Server Protocol Support
- Emacs and VSCode plugins (w/ semantic highlighting)
- Workshops, tutorials, developer sessions


Scilla IDE
savant-ide.zilliqa.com
Incognito
SCILLA
SCILLA DOCS
+ NEW CONTRACT
Files
HelloWorld.scilla
BookStore.scilla
CrowdFunding.scilla
Auction.scilla
FungibleToken.scilla
NonFungible.scilla
ZilGame.scilla
SchnorrTest.scilla
ECDSATest.scilla
Save
Check
Events
Reset
Settings
Block Height: 5
scilla_version 0
(* Associated library *)
import BoolUtils
library CrowdFunding
let one_msg =
fun (msg : Message) =>
let nil_msg = Nil {Message} in
Cons {Message} msg nil_msg
let check_update =
fun (bs : Map ByStr20 Uint128) =>
fun (_sender : ByStr20) =>
fun (_amount : Uint128) =>
let c = builtin contains bs _sender in
match c with
| False =>
let bs1 = builtin put bs _sender _amount in
Some {Map ByStr20 Uint128} bs1
| True => None {Map ByStr20 Uint128}
end
let blk_leq =
fun (blk1 : BNum) =>
fun (blk2 : BNum) =>
let bc1 = builtin blt blk1 blk2 in
let bc2 = builtin eq blk1 blk2 in
orb bc1 bc2
let accepted_code = Int32 1
let missed_deadline_code = Int32 2
let already_backed_code = Int32 3
let not_owner_code = Int32 4
Block Height: 5
CrowdFunding.scilla
Ln 10, Col 0
CALL
STATE
DEPLOY
Select account
Select a contract
Zilliqa © 2018


ViewBlock | Zilliqa Contract
viewblock.io/zilliqa/address/zil1w0gj7tnxk8usu68et44jfchuwh0mjc040pqd6l?tab=code
Incognito
ZILLIQA
mainnet
Search for a tx, address, name or block.
Sign In
ADDRESSES
TRANSACTIONS
BLOCKS
STATS
API
Price $0.006
Market Cap $52.72M
Volume $27.57M
Contract
zil1w0gj7tnxk8usu68et44jfchuwh0mjc040pqd6l
COPY ADDRESS
QR
Balance
193.35 ZIL
Transactions
40
Contract Creation
zil1fxx... at dab4e4878c0d93abcfaa...
TRANSACTIONS
CODE
scilla_version 0
import BoolUtils
library Exchange
let zero_address = 0x0000000000000000000000000000000000000000
let zero = Uint128 0
let one_msg =
fun (msg: Message) =>
let nil_msg = Nil {Message} in
Cons {Message} msg nil_msg
(* error codes library *)
let code_success = Uint32 0


Global Ranks by # of Active (validating) Blockchain Nodes
Ethereum Classic
1.2%
Harmony
2.3%
Ripple
2.5%
TRON
2.5%
Algorand
2.8%
Bitcoin Cash
3.2%
Monero
3.6%
Litecoin
3.9%
Zilliqa
4.7%
Dash
11.2%
Ethereum
21.1%
Bitcoin
20.1%
Qtum
12.5%
William Mougayar © Sept 22 2019 v1

To Take Away

- Adopting a foundational calculus is a great way to keep a new language *minimalistic* and *expressive.*
- **Lots of ideas from PL research** can be reused with *very low overhead* on implementation and adoption.
- Yet the language will be forced to *grow* and *change*.
- It pays off to build an enthusiastic developer community: more feedback — more *informed design choices.*

scilla-lang.org

Thanks!