Quartz for Cross-Platform Smart Contracts

What is the Problem?

- Platforms exist for decentralised smart contracts, e.g. Ethereum
- Solidity language was not inherently secure -> Advent of Flint: Implemented in Swift, Designed to be Secure and Easy to use
- Other platforms are now emerging such as Libra Platform
- Flint was heavily tied to Solidity so can not easily target Libra

Quartz Language + Compiler

- Fully featured Language capable of encoding smart contracts that can be compiled down to Solidity (Ethereum) and MoveIR (Libra)
- Working Compiler to compile into cross-platform deployable smart contracts
- New added features such as External Contract Interaction and Asset construct

Quartz Language + Compiler

```
topLevelModule = 1*(topLevelDeclaration CRLF);

topLevelDeclaration = contractDeclaration
                    | contractBehaviourDeclaration
                    | structDeclaration
                    | assetDeclaration
                    | enumDeclaration
                    | traitDeclaration;

; CONTRACTS
contractDeclaration = "contract" SP identifier SP
"{" *(WSP variableDeclaration CRLF) "}";

; VARIABLES
variableDeclaration = [*(modifier SP)] WSP ("var" | "let")
SP identifier typeAnnotation [WSP "=" WSP expression];

; TYPES
typeAnnotation = ":" WSP type;

type = identifier ["<" type *("," WSP type) ">"]
     | basicType
     | solidityType
     | moveType
     | arrayType
     | dictType;

basicType = "Bool"
          | "Int"
          | "String"
          | "Address";

moveType = "bool"
         | "address"
         | "u8"
         | "u64"
         | "vector<u8>";

arrayType     = "[" type "]";
dictType      = "[" type ":" WSP type "]";
```

```
enumDeclaration = "enum" SP identifier SP [typeAnnotation] SP "{" *(WSP enumCase CRLF) "}";
enumCase        = "case" SP identifier
                | "case" SP identifier WSP "=" WSP expression;

traitDeclaration = "(" traitModifier *(WSP traitModifier) ")" "trait" SP identifier
SP "{" *(WSP traitMember CRLF) "}";

traitModifier = "@" identifier*("(" identifer ":" addressLiteral ")");

traitMember = functionSignatureDeclaration;

; EVENTS
eventDeclaration = "event" identifer parameterList

; STRUCTS
structDeclaration = "struct" SP identifier SP "{" *(WSP structMember CRLF) "}";

structMember = variableDeclaration
             | functionDeclaration
             | initializerDeclaration;

;ASSETS
assetDeclaration = "asset" SP identifier SP "{" *(WSP assetMember CRLF) "}";

assetMember = variableDeclaration
            | functionDeclaration
            | initializerDeclaration;

; BEHAVIOUR
contractBehaviourDeclaration = identifier SP "::" WSP [callerBinding] callerProtectionGroup
WSP "{" *(WSP contractBehaviourMember CRLF) "}";

contractBehaviourMember = functionDeclaration
                        | specialDeclaration
                        | initializerSignatureDeclaration
                        | functionSignatureDeclaration;

; ACCESS GROUPS
callerBinding         = identifier WSP "<-";
callerProtectionGroup = identifierGroup;
identifierGroup       = "(" identifierList ")";
identifierList        = identifier *("," WSP identifier)
```

```
functionSignatureDeclaration = functionHead SP identifier
parameterList [returnType]
functionDeclaration  = functionSignatureDeclaration codeBlock;
specialDeclaration   = initializerDeclaration | fallbackDeclaration;
initializerSignatureDeclaration = initializerHead parameterList
initializerDeclaration  = initializerSignatureDeclaration codeBlock;
fallbackDeclaration     = fallbackHead parameterList codeBlock;

functionHead    = [*(attribute SP)] [*(modifier SP)] "func";
initializerHead = [*(attribute SP)] [*(modifier SP)] "init";
fallbackHead    = [*(modifier SP)] "fallback";

modifier  = "public"
          | "mutating"
          | "visible";

returnType = "->" type;

parameterList = "()"
              | "(" parameter *("," parameter) ")";

parameter  = *(parameterModifiers SP) identifier typeAnnotation
[WSP "=" WSP expression];
parameterModifiers = "inout"

; STATEMENTS
codeBlock = "{" [CRLF] *(WSP statement CRLF) WSP
statement [CRLF]"}";
statement = expression
          | returnStatement
          | emitStatement
          | forStatement
          | ifStatement;

returnStatement  = "return" SP expression
emitStatement    = "emit" SP functionCall
forStatement = "for" SP variableDeclaration SP "in" SP expression
SP codeBlock
```

```
expression = identifier
           | inOutExpression
           | binaryExpression
           | functionCall
           | literal
           | arrayLiteral
           | dictionaryLiteral
           | self
           | variableDeclaration
           | bracketedExpression
           | subscriptExpression
           | rangeExpression
           | externalCall;

inOutExpression = "&" expression;

binaryOp = "+" | "-" | "*" | "|" | "**"
         | "&+" | "&-" | "&*"
         | "="
         | "==" | "!="
         | "+=" | "-=" | "*=" | "|="
         | "||" | "&&"
         | ">" | "<" | "<=" | ">="
         | ".";

binaryExpression = expression WSP binaryOp WSP expression;

self = "self"

rangeExpression = "(" expression ( "..<" | "..." ) expression ")"

bracketedExpression = "(" expression ")";

subscriptExpression = subscriptExpression "[" expression "]";
                    | identifier "[" expression "]";

; FUNCTION CALLS
functionCall = identifier "(" [expression] *( "," WSP expression ) ")";

; EXTERNAL FUNCTION CALL
externalCall = "call" WSP functionCall;

; CONDITIONALS
ifStatement = "if" SP expression SP codeBlock [elseClause];
elseClause  = "else" SP codeBlock;

; LITERALS
identifier =  ALPHA  *( ALPHA | DIGIT | "_" );
literal    = numericLiteral
           | stringLiteral
           | booleanLiteral
           | addressLiteral;
```

Cross-Platform Counter Contract Demo

Interacting with External Contracts - Motivations

- Common pattern of smart contracts is interaction with other deployed smart contracts
- Allows Functionality Sharing
- Enables Complex Smart Contract Architectures
- Fundamental component for handling Libra currency

Interacting with External Contracts - Challenges

- Need a means of knowing the interface of contract to enable interaction
- Mapping between Quartz Types and Solidity/Move Types
- Need to facilitate external calls

Interacting with External Contracts - Challenge -> Solution

1. Encoding the interface of contract

Trait:

```
@contract
external trait GlobalDB {
    public func get_product(k: uint64) -> string
    public func insert(k: uint64, v: string)
    public func is_present(k: uint64) -> bool
}
```

Interacting with External Contracts - Challenge -> Solution

2. Mapping between

Internal Quartz and

External Types

Cast Expression:

```
var key: Int;
key = 9;
cast(key to uint64)
```

Interacting with External Contracts - Challenge -> Solution

3. Facilitating External Calls

External Call Expression:

```
call productDatabase.get_product(k: cast key to uint64)
```

Interacting with External Contracts - Solution

```
@contract
external trait GlobalDB {
  public func get_product(k: uint64) -> string
  public func insert(k: uint64, v: string)
  public func is_present(k: uint64) -> bool
}

contract Shop {
  visible var productDatabase: GlobalDB
}

Shop :: sender <- (any) {
  public init() {
    productDatabase = GlobalDB(0x0000000)
  }

  public func get(key: Int) -> Int {
    return cast (call productDatabase.is_present(k: cast key to uint64)) to Int
  }
```

Ethereum External Contract Interaction Demo

Asset - The Problem

- Flint handles currency via manipulating internal raw value and providing safe wrapper

Flint Asset:

```
struct trait Asset {

  init(unsafeRawValue: Int)
…..

  func setRawValue(value: Int) -> Int


  func getRawValue() -> Int
}
```

Asset - The Problem

- The Libra currency can only be interacted through an interface

- There is no access to the internal integer value

Asset - Solution

- A new designated Asset construct that provides a notion of representing and manipulating currency
- Flexible Semantics that generalises across multiple platforms

Asset - Libra Implementation

- Firstly, expose the Libra currency interface within Quartz

```
@resource
external trait Libra_Coin {
  public func getValue() -> uint64
  public func transfer(to: inout LibraCoin, value: uint64)
  public func transfer_value(to: LibraCoin)
}
```

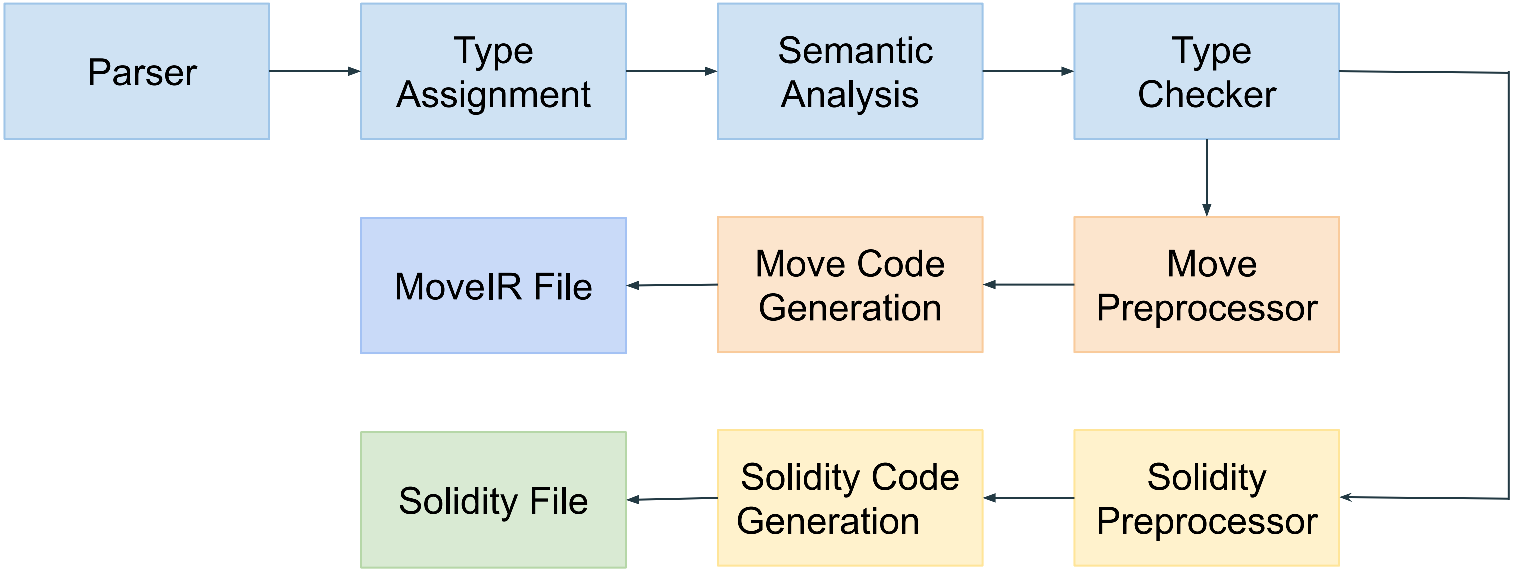
Asset - Libra Implementation

- Secondly, implement representation using the trait

```
asset Libra {
  visible var libra: Libra_Coin

  public init() {
    libra = Libra_Coin(0x0000000000000000000000000000000000000000)
  }

  public func balance() -> Int {
    return cast (call libra.getValue()) to Int
  }

  func transfer(to: inout Libra, amount: Int) mutates (libra) {
    call libra.transfer(to: &to.libra, value: (cast amount to uint64))
  }

  func transfer_value(to: Libra) mutates (libra) {
    call libra.transfer_value(to: to)
  }
}
```

Libra Handling Currency Demo

Quartz Compiler - Architecture

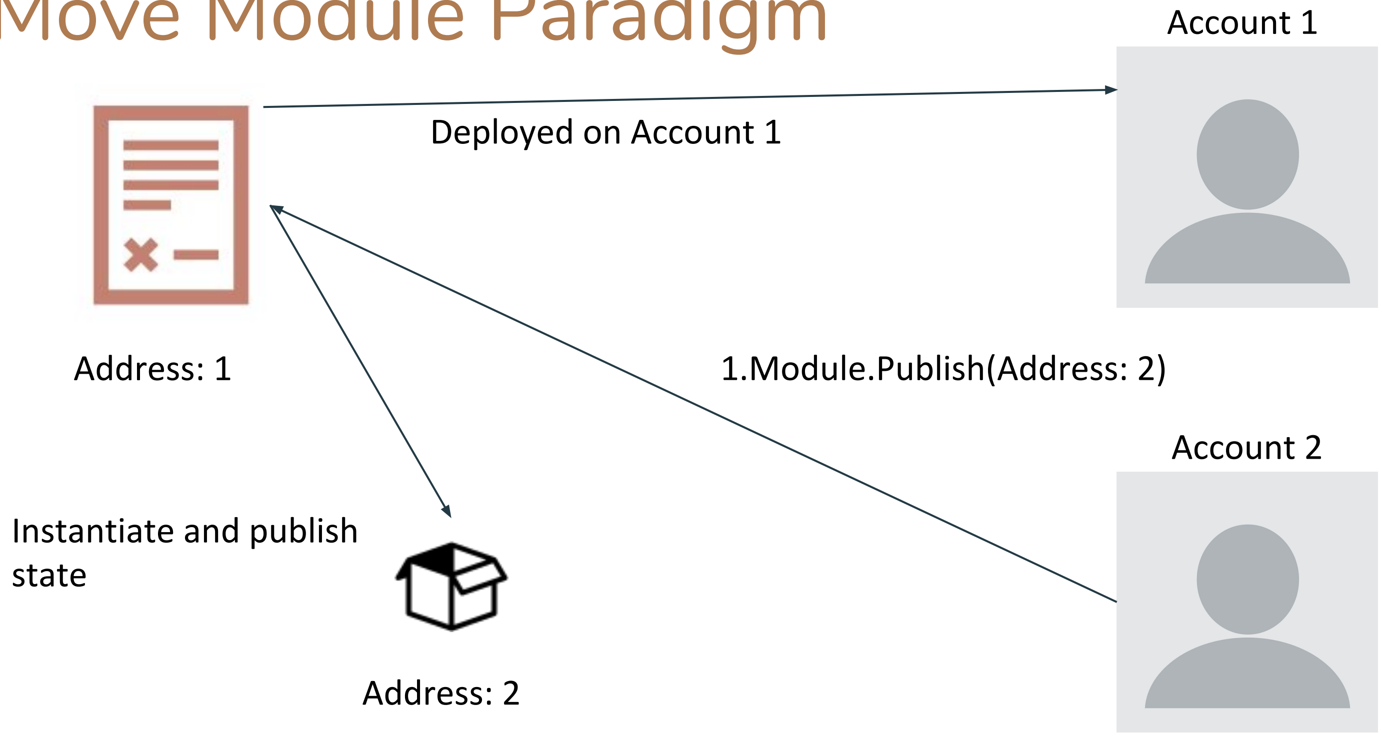


Move Code Generation - Challenges

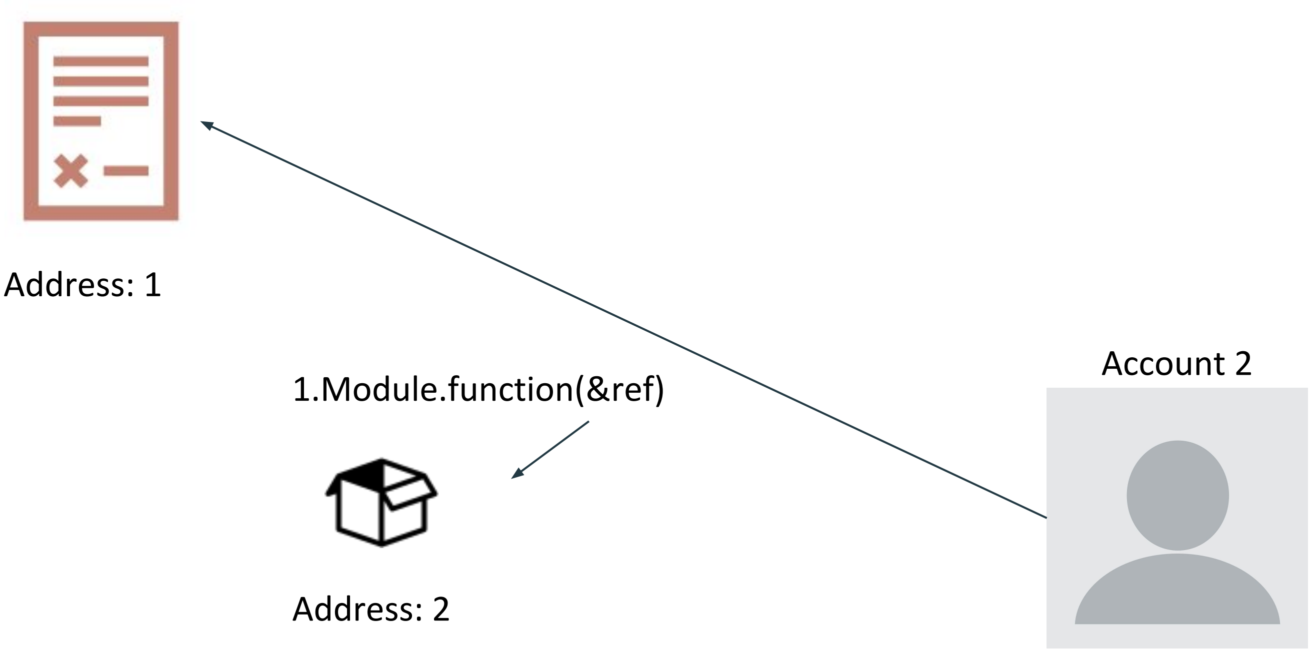
- Requires mapping between the object oriented Contract Paradigm to the Module Imperative Paradigm

- Enabling compliance to MoveIR’s strict semantics

Move Module Paradigm



Move Module Paradigm



Code Generation - Contract vs Module

- Quartz’s perspective of data and code being together

Vs

- Move’s Strict separation of Data and Code

Contract vs Module

```
contract Counter {
  var value: Int = 0
}


Counter :: (any) {
  public init() {}

  public func getValue() -> Int {
    return value
  }
```

```
module Counter {

  resource T {
    value: u64
  }

  new(): Self.T {
    let __this_value: u64;
    __this_value = 0;
    return T {
      value: move(__this_value) };
  }

  public publish() {
    move_to_sender<T>(Self.new());
    return;
  }

  Counter_getValue (this: &mut Self.T): u64  {
    let ret: u64;
    ret = *&mut copy(this).value;
    _ = move(this);
    return move(ret);
  }

  public getValue (__address_this: address): u64 acquires T {
    let ret: u64;
    let this: &mut Self.T;
    this = borrow_global_mut<T>(move(__address_this));
    ret = Self.Counter_getValue(copy(this));
    _ = move(this);
    return move(ret);
  }
```

Code Generation - Wrapping Functions

- Strict separation of state and code requires function interface to be transformed

Code Generation - Wrapping Functions

```
public func getValue() -> Int {
  return value
}
```

```
Counter_getValue (this: &mut Self.T): u64  {
  let ret: u64;
  ret = *&mut copy(this).value;
  _ = move(this);
  return move(ret);
}

public getValue (__address_this: address): u64 acquires T {
  let ret: u64;
  let this: &mut Self.T;
  this = borrow_global_mut<T>(move(__address_this));
  ret = Self.Counter_getValue(copy(this));
  _ = move(this);
  return move(ret);
}
```

Code Generation - Ownership & Reference Handling

- Values can only have 1 unique owner
- Ownership is transferred or borrowed via references
- Manual reference handling has to be correct
- Property Access restricted to 1 level
- No support for higher level data structures

Code Generation - Example

```
public func setBxy(y: Bool) mutates (A.y) {
    b.x.y = y
}
```

1. Variable Declarations
2. Correct referencing and dereferencing
3. Performing the actual state change
4. Releasing references

```
C_setBxy (this: &mut Self.T, _y: bool) {

    let _temp__4:     &mut Self.B;
    let _temp__6:     &mut Self.A;

    _temp__4 =     &mut copy(this).b;
    _temp__6 =     &mut copy(_temp__4).x;
    *&mut copy(_temp__6).y =     copy(_y);

    _ = move(_temp__4);
    _ = move(_temp__6);
    _ = move(this);
    return;  }
```

Evaluation

- We have successfully verified correctness of code generation
- The Quartz compiler outperforms the Flint compiler quantitatively

Evaluation - Compiler Performance

Table 7.1: Compilation Time Comparison

Compiler	Debug Compilation	Release Compilation (optimised)
Quartz	38.9s	96.5s
Flint	109.1s	297s

Table 7.2: Runtime Comparison

Compiler	Contract	Target	Runtime
Quartz	Counter	Ethereum	0.029s
		Libra	0.027s
	External GlobalDB	Ethereum	0.034s
		Libra	0.033s
	Moneypot	Ethereum	0.049s
		Libra	0.044s
Flint	Counter	Ethereum	0.112s
	Moneypot	Ethereum	0.146s

Challenges

- Working with the changing nature of the Move language
- Significant amount of investigation and design work for Quartz Language
- Significant amount of Code design and Implementation in the compiler (18,000 lines of my Rust code)

Questions?