Security Preservation through Secure Compilation

Use case: AlgoMove - A Move Embedding for Algorand

Lorenzo Benetollo[1,2] **Michele Bugliesi**[1] **Silvia Crafa**[3] **Sabina Rossi**[1] **Alvise Spanò**[1]

[1]Ca' Foscari University of Venice [2]University of Camerino [3]University of Padua

Meeting PRIN NiRvAna - Udine (UD), June 7, 2024

What can go wrong?

Many attacks have been carried out exploiting smart contract vulnerabilities:

What can go wrong?

Many attacks have been carried out exploiting smart contract vulnerabilities:

- TheDao Attack, Reentrancy, **2016**, ~$60 M

What can go wrong?

Many attacks have been carried out exploiting smart contract vulnerabilities:

- TheDao Attack, Reentrancy, **2016**, ~$60 M
- […] many more

What can go wrong?

Many attacks have been carried out exploiting smart contract vulnerabilities:

- TheDao Attack, Reentrancy, **2016**, ~$60 M
- […] many more
- Rari Capital Exploit, Reentrancy, 2021, ~$10 M

What can go wrong?

Many attacks have been carried out exploiting smart contract vulnerabilities:

- TheDao Attack, Reentrancy, **2016**, ~$60 M
- […] many more
- Rari Capital Exploit, Reentrancy, 2021, ~$10 M
- KyberSwap, Reentrancy, **2023**, ~$50 M

Secure Compilation

What is it?

Preserve the security properties of the source programs in the target programs

Secure Compilation

What is it?

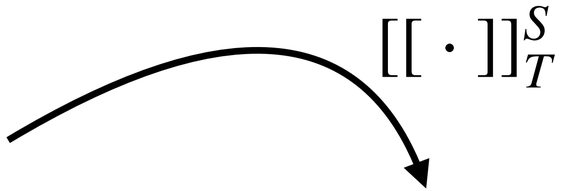
Preserve the security properties of the source programs in the target programs

```
package Bank;

public class Account {
    private int balance = 0;

    public void deposit(int amount) {
        this.balance += amount;
    }
}
```

Secure Compilation

What is it?

Preserve the security properties of the source programs in the target programs



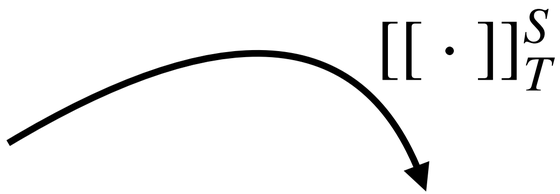

```
package Bank;

public class Account {
    private int balance = 0;

    public void deposit(int amount) {
        this.balance += amount;
    }
}
```

Secure Compilation

What is it?

Preserve the security properties of the source programs in the target programs

```
package Bank;

public class Account {
    private int balance = 0;

    public void deposit(int amount) {
        this.balance += amount;
    }
}
```




```
typedef struct account_t {
    int balance = 0;
    void ( *deposit )( struct Account*, int )
        = deposit_f;
} Account;

void deposit_f( Account* a, int amount ) {
    a->balance += amount;
    return;
}
```

Secure Compilation

How to prove security?

Secure Compilation

How to prove security?

- Fully abstract compilation
  - reflect and preserve observational equivalence
  - requires expensive runtime constructs in compiled code
  - constructs that may have no relevance for security

Secure Compilation

How to prove security?

- Fully abstract compilation
  - reflect and preserve observational equivalence
  - requires expensive runtime constructs in compiled code
  - constructs that may have no relevance for security
- Robust safety
  - preserves relevant safety properties against all adversarial contexts
  - can be proved more easily than fully abstract compilation
  - more efficient code

Secure Compilation

Which security properties?

Secure Compilation

Which security properties?

- Confidentiality
- Integrity
- Memory allocation
- Well-bracketed control flow

Secure Compilation

Which security properties?

- Confidentiality
- Integrity
- Memory allocation
- Well-bracketed control flow
- Linearity preservation?
- Double spending?
- Non-interference?
- Reentrancy?

Secure Compilation

Blockchain context

Secure Compilation

Blockchain context

- What about Smart Contract interaction?

Secure Compilation

Blockchain context

- What about Smart Contract interaction?
- Is the attacker model the same?

Secure Compilation

Blockchain context

- What about Smart Contract interaction?
- Is the attacker model the same?
- Which security properties do we want to preserve?

Secure Compilation

Blockchain context

- What about Smart Contract interaction?
- Is the attacker model the same?
- Which security properties do we want to preserve?
- Which attacks do we want to avoid?

Secure Compilation

Blockchain context

- What about Smart Contract interaction?
- Is the attacker model the same?
- Which security properties do we want to preserve?
- Which attacks do we want to avoid?
- How can we prove secure compilation?

Secure Compilation

Blockchain context

Secure Compilation

Blockchain context

- Solidity vs TEAL vs Move

Secure Compilation

Blockchain context

- Solidity vs TEAL vs Move
- Linearity preservation - reentrancy avoidance - double spending

Secure Compilation

Blockchain context

- Solidity vs TEAL vs Move
- Linearity preservation - reentrancy avoidance - double spending
- Move has been proved robustly safe

AlgoMove

A Move embedding for Algorand Blockchain

Embedding Move on Algorand

Goals

Embedding Move on Algorand

Goals

- Move is really blockchain-agnostic?

Embedding Move on Algorand

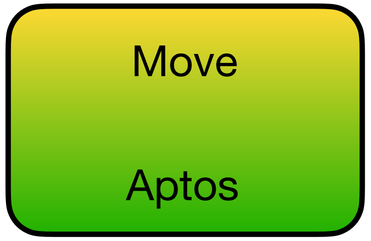
Goals

- Move is really blockchain-agnostic?
- Can Algorand be used with a more high-level language, taking advantage of the safety of static linear types?

Embedding Move on Algorand

Goals

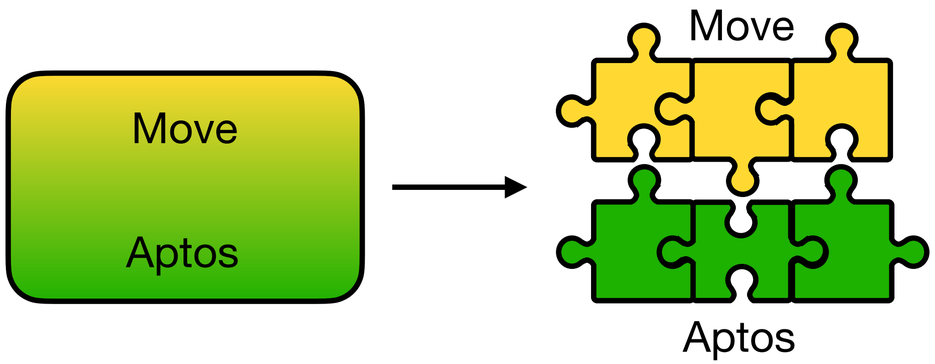
- Move is really blockchain-agnostic?
- Can Algorand be used with a more high-level language, taking advantage of the safety of static linear types?



Embedding Move on Algorand

Goals

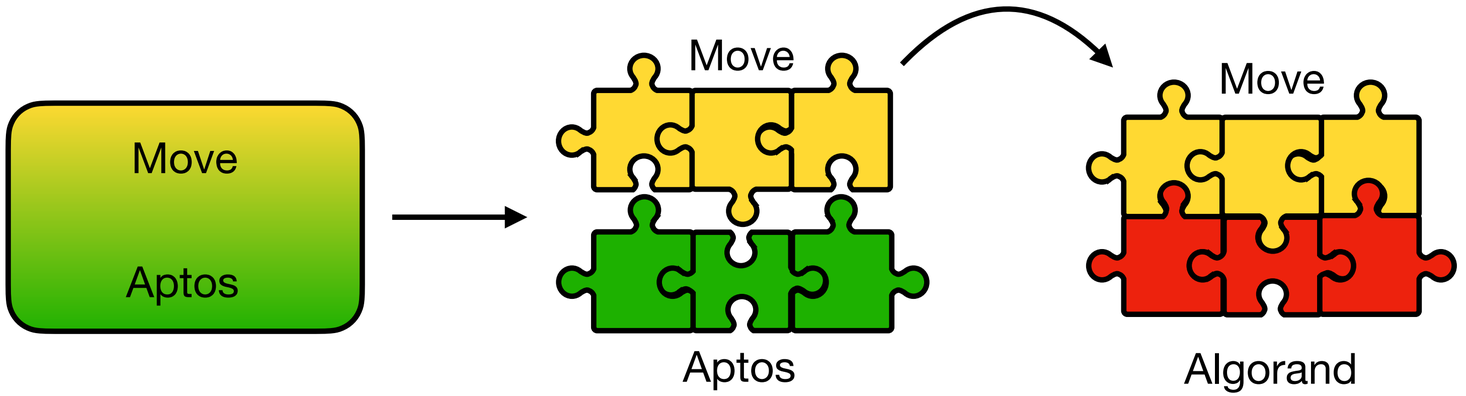
- Move is really blockchain-agnostic?
- Can Algorand be used with a more high-level language, taking advantage of the safety of static linear types?



Embedding Move on Algorand

Goals

- Move is really blockchain-agnostic?
- Can Algorand be used with a more high-level language, taking advantage of the safety of static linear types?



The Move Language

The Move Language

- High-level language compiled into bytecode

The Move Language

- High-level language compiled into bytecode
- Strongly typed with parametric polymorphism

The Move Language

- High-level language compiled into bytecode
- Strongly typed with parametric polymorphism
- Linear types enforce the must-move semantics

The Move Language

- High-level language compiled into bytecode
- Strongly typed with parametric polymorphism
- Linear types enforce the must-move semantics
- Blockchain agnostic (almost!)

The Move Language

- High-level language compiled into bytecode
- Strongly typed with parametric polymorphism
- Linear types enforce the must-move semantics
- Blockchain agnostic (almost!)
- Bytecode verifier

Linear Types

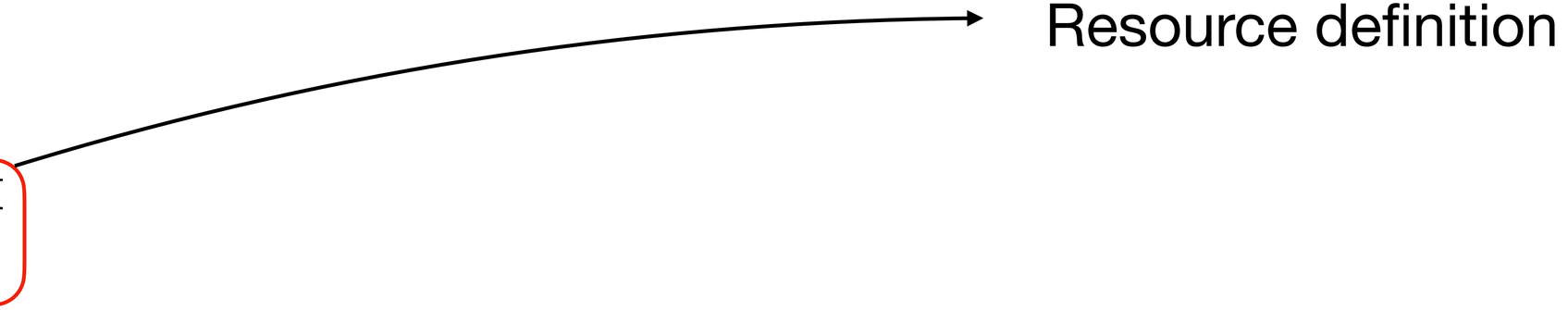
```
module example {

    struct S {
        a: u64
    }

    public fun foo() {
        let s1 = S{ a: 1 };
        let s2 = increment(s1); // Error: s2 can not be dropped!
        s1.a; // Error: s1 hab been moved!
    }

    public fun increment(s: S) : S {
        s.a = s.a + 1;
        s
    }
}
```

Move code example

```
module example {

    use std::signer;

    struct S has key {
        val: u64
    }

    public fun set(acc: &signer, n: u64) {
        let s = S { val: n };
        move_to(acc, s);
    }

    public fun get(acc: address): u64 acquires S {
        let s = borrow_global<S>(acc);
        s.val
    }

    public fun update(acc: &signer, new_val: u64) acquires S {
        let acc_address = signer::address_of(acc);
        let s = borrow_global_mut<S>(acc_address);
        s.val = new_val;
    }
}
```

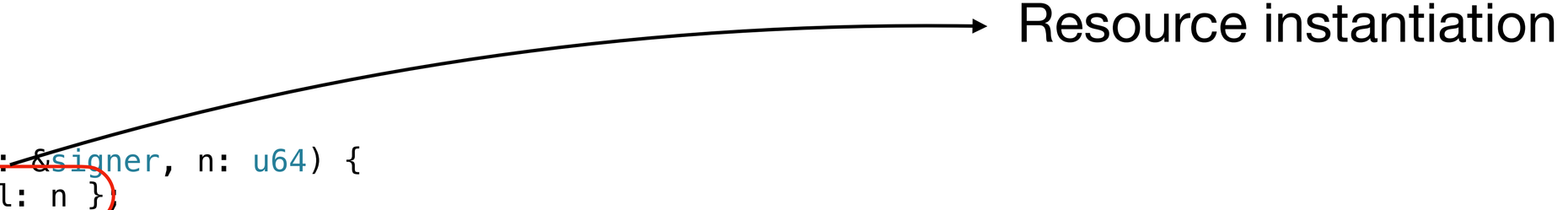
Move code example

```
module example {

    use std::signer;

    struct S has key {
        val: u64
    }

    public fun set(acc: &signer, n: u64) {
        let s = S { val: n };
        move_to(acc, s);
    }

    public fun get(acc: address): u64 acquires S {
        let s = borrow_global<S>(acc);
        s.val
    }

    public fun update(acc: &signer, new_val: u64) acquires S {
        let acc_address = signer::address_of(acc);
        let s = borrow_global_mut<S>(acc_address);
        s.val = new_val;
    }
}
```



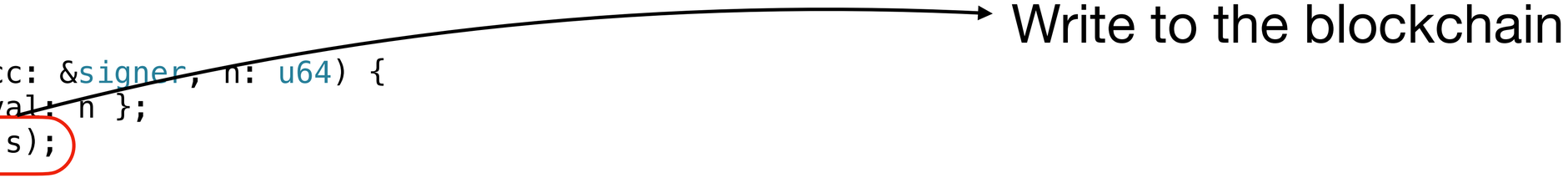
Move code example

```
module example {

    use std::signer;

    struct S has key {
        val: u64
    }

    public fun set(acc: &signer, n: u64) {
        let s = S { val: n };
        move_to(acc, s);
    }

    public fun get(acc: address): u64 acquires S {
        let s = borrow_global<S>(acc);
        s.val
    }

    public fun update(acc: &signer, new_val: u64) acquires S {
        let acc_address = signer::address_of(acc);
        let s = borrow_global_mut<S>(acc_address);
        s.val = new_val;
    }
}
```



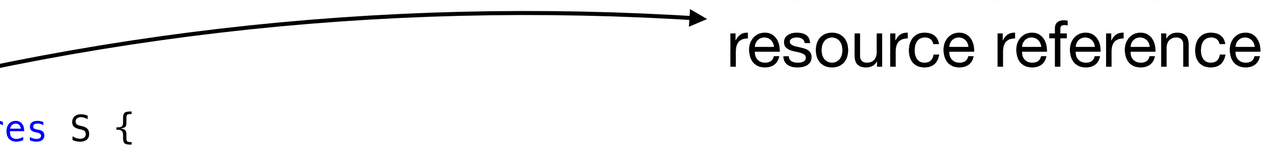
Move code example

```
module example {

    use std::signer;

    struct S has key {
        val: u64
    }

    public fun set(acc: &signer, n: u64) {
        let s = S { val: n };
        move_to(acc, s);
    }

    public fun get(acc: address): u64 acquires S {
        let s = borrow_global<S>(acc);
        s.val
    }

    public fun update(acc: &signer, new_val: u64) acquires S {
        let acc_address = signer::address_of(acc);
        let s = borrow_global_mut<S>(acc_address);
        s.val = new_val;
    }
}
```



Move code example

```
module example {

    use std::signer;

    struct S has key {
        val: u64
    }

    public fun set(acc: &signer, n: u64) {
        let s = S { val: n };
        move_to(acc, s);
    }

    public fun get(acc: address): u64 acquires S {
        let s = borrow_global<S>(acc);
        s.val
    }

    public fun update(acc: &signer, new_val: u64) acquires S {
        let acc_address = signer::address_of(acc);
        let s = borrow_global_mut<S>(acc_address);
        s.val = new_val;
    }
}
```



Move code example

```
module example {

    use std::signer;

    struct S has key {
        val: u64
    }

    public fun set(acc: &signer, n: u64) {
        let s = S { val: n };
        move_to(acc, s);
    }

    public fun get(acc: address): u64 acquires S {
        let s = borrow_global<S>(acc);
        s.val
    }

    public fun update(acc: &signer, new_val: u64) acquires S {
        let acc_address = signer::address_of(acc);
        let s = borrow_global_mut<S>(acc_address);
        s.val = new_val;
    }
}
```

Get **mutable** resource reference

Move code example

```
module example {

    use std::signer;

    struct S has key {
        val: u64
    }

    public fun set(acc: &signer, n: u64) {
        let s = S { val: n };
        move_to(acc, s);
    }

    public fun get(acc: address): u64 acquires S {
        let s = borrow_global<S>(acc);
        s.val
    }

    public fun update(acc: &signer, new_val: u64) acquires S {
        let acc_address = signer::address_of(acc);
        let s = borrow_global_mut<S>(acc_address);
        s.val = new_val;
    }
}
```

TEAL / PyTeal

- Low level assembly-like language
- Approval oriented
- Tailored for Algorand Blockchain
- PyTeal framework to produce TEAL code

PyTeal code example

```python
class S:
    val = LocalStateValue(
        stack_type=pt.TealType.uint64,
    )

app = Application("S", state=S())

@app.external
def set(new_val: pt.abi.Uint64) -> pt.Expr:
    return app.state.val.set(new_val.get())


@app.external
def get(*, output: pt.abi.Uint64) -> pt.Expr:
    return output.set(app.state.val.get())

@app.external
def update(new_val: pt.abi.Uint64) -> pt.Expr:
    return app.state.val.set(new_val.get())

@app.opt_in
def opt_in() -> pt.Expr:
    return app.initialize_local_state()
```

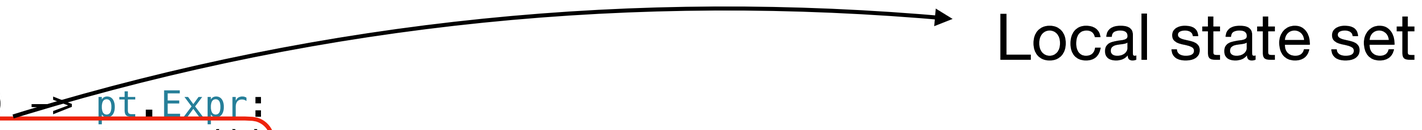
PyTeal code example

```python
class S:
    val = LocalStateValue(
        stack_type=pt.TealType.uint64,
    )

app = Application("S", state=S())

@app.external
def set(new_val: pt.abi.Uint64) -> pt.Expr:
    return app.state.val.set(new_val.get())


@app.external
def get(*, output: pt.abi.Uint64) -> pt.Expr:
    return output.set(app.state.val.get())

@app.external
def update(new_val: pt.abi.Uint64) -> pt.Expr:
    return app.state.val.set(new_val.get())

@app.opt_in
def opt_in() -> pt.Expr:
    return app.initialize_local_state()
```

Local state definiton

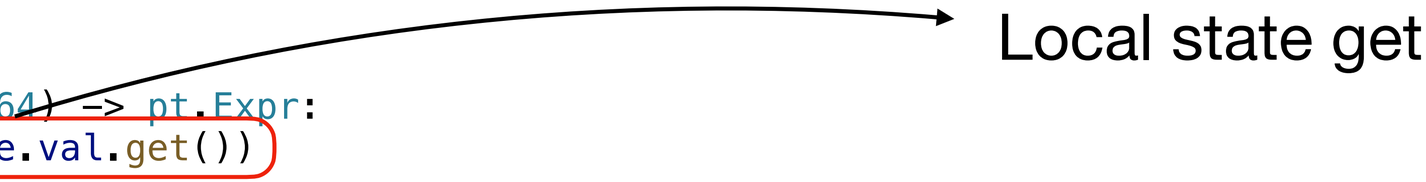
PyTeal code example

```python
class S:
    val = LocalStateValue(
        stack_type=pt.TealType.uint64,
    )

app = Application("S", state=S())

@app.external
def set(new_val: pt.abi.Uint64) -> pt.Expr:
    return app.state.val.set(new_val.get())


@app.external
def get(*, output: pt.abi.Uint64) -> pt.Expr:
    return output.set(app.state.val.get())

@app.external
def update(new_val: pt.abi.Uint64) -> pt.Expr:
    return app.state.val.set(new_val.get())

@app.opt_in
def opt_in() -> pt.Expr:
    return app.initialize_local_state()
```



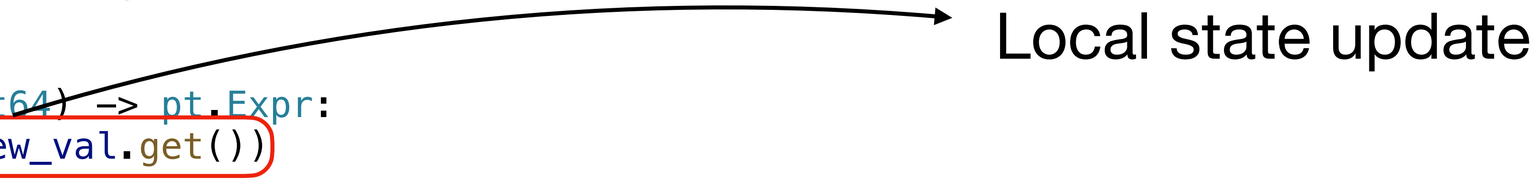
PyTeal code example

```python
class S:
    val = LocalStateValue(
        stack_type=pt.TealType.uint64,
    )

app = Application("S", state=S())

@app.external
def set(new_val: pt.abi.Uint64) -> pt.Expr:
    return app.state.val.set(new_val.get())


@app.external
def get(*, output: pt.abi.Uint64) -> pt.Expr:
    return output.set(app.state.val.get())

@app.external
def update(new_val: pt.abi.Uint64) -> pt.Expr:
    return app.state.val.set(new_val.get())

@app.opt_in
def opt_in() -> pt.Expr:
    return app.initialize_local_state()
```



PyTeal code example

```python
class S:
    val = LocalStateValue(
        stack_type=pt.TealType.uint64,
    )

app = Application("S", state=S())

@app.external
def set(new_val: pt.abi.Uint64) -> pt.Expr:
    return app.state.val.set(new_val.get())


@app.external
def get(*, output: pt.abi.Uint64) -> pt.Expr:
    return output.set(app.state.val.get())

@app.external
def update(new_val: pt.abi.Uint64) -> pt.Expr:
    return app.state.val.set(new_val.get())

@app.opt_in
def opt_in() -> pt.Expr:
    return app.initialize_local_state()
```



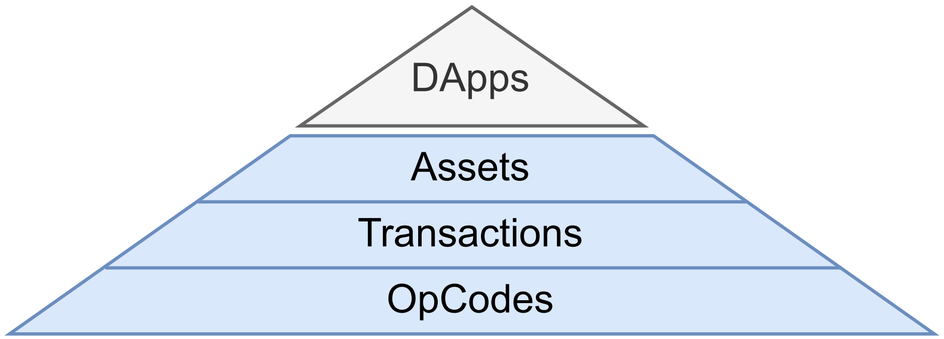
PyTeal code example

```python
class S:
    val = LocalStateValue(
        stack_type=pt.TealType.uint64,
    )

app = Application("S", state=S())

@app.external
def set(new_val: pt.abi.Uint64) -> pt.Expr:
    return app.state.val.set(new_val.get())


@app.external
def get(*, output: pt.abi.Uint64) -> pt.Expr:
    return output.set(app.state.val.get())

@app.external
def update(new_val: pt.abi.Uint64) -> pt.Expr:
    return app.state.val.set(new_val.get())

@app.opt_in
def opt_in() -> pt.Expr:
    return app.initialize_local_state()
```

AlgoMove

Framework structure

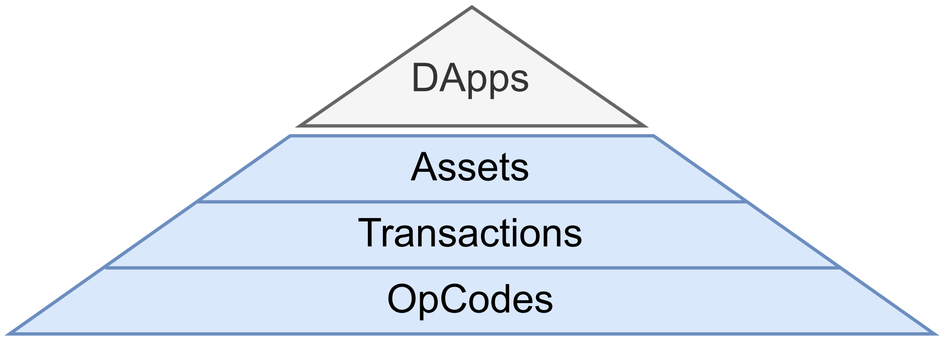
- The ***OpCodes*** Layer: lowest layer consisting of a Move module declaring function prototypes tagged with the *native* keyword
- The ***Transactions*** Layer: middle layer providing an API for performing transactions, inner transactions and transaction groups
- The ***Assets*** Layer: topmost layer provides datatypes and functions for creating and manipulating Algorand assets



AlgoMove

Framework structure

- The ***OpCodes*** Layer: lowest layer consisting of a Move module declaring function prototypes tagged with the *native* keyword
- The ***Transactions*** Layer: middle layer providing an API for performing transactions, inner transactions and transaction groups
- The ***Assets*** Layer: topmost layer provides datatypes and functions for creating and manipulating Algorand assets



AlgoMove
Code Example

```
module algomove::algomove_auction_paper {

  use algomove::opcode;
  use algomove::transaction as txn;
  use algomove::asset::{Asset, deposit}

  struct Auction<phantom AssetType> has key {
    auctioneer: address,
    top_bid: Asset<AssetType>,
    top_bidder: address,
  }

  public fun start_auction<AssetType>(base: Asset<AssetType>) {
    let sender = txn::get_sender();
    let auction = Auction<AssetType> {
      auctioneer: sender,
      top_bid: base,
      top_bidder: sender,
    };
    opcode::app_global_put(txn::bytes_of_address(sender), auction);
  }

  public fun bid<AssetType>(auctioneer: address, assets: Asset<AssetType>) {
    let Auction { auctioneer, top_bid, top_bidder } = opcode::app_global_get<Auction<AssetType>>(txn::bytes_of_address(auctioneer));
    assert!(get_amount(&assets) > get_amount(&top_bid), 1);
    let sender = txn::get_sender();
    deposit(top_bid, top_bidder);
    let new_auction = Auction<AssetType> { auctioneer, top_bid: assets, top_bidder: sender };
    opcode::app_global_put(txn::bytes_of_address(auctioneer), new_auction);
  }

  public fun finalize_auction<AssetType>() {
    let sender = txn::get_sender();
    let auction = opcode::app_global_get<Auction<AssetType>>(txn::bytes_of_address(sender));
    assert!(sender == auction.auctioneer, 2);
    let Auction { auctioneer, top_bid, top_bidder: _ } = auction;
    deposit(top_bid, auctioneer);
  }
}
```

AlgoMove
Code Example

Framework imports

```
module algomove::algomove_auction_paper {

  use algomove::opcode;
  use algomove::transaction as txn;
  use algomove::asset::{Asset, deposit}

  struct Auction<phantom AssetType> has key {
    auctioneer: address,
    top_bid: Asset<AssetType>,
    top_bidder: address,
  }

  public fun start_auction<AssetType>(base: Asset<AssetType>) {
    let sender = txn::get_sender();
    let auction = Auction<AssetType> {
      auctioneer: sender,
      top_bid: base,
      top_bidder: sender,
    };
    opcode::app_global_put(txn::bytes_of_address(sender), auction);
  }

  public fun bid<AssetType>(auctioneer: address, assets: Asset<AssetType>) {
    let Auction { auctioneer, top_bid, top_bidder } = opcode::app_global_get<Auction<AssetType>>(txn::bytes_of_address(auctioneer));
    assert!(get_amount(&assets) > get_amount(&top_bid), 1);
    let sender = txn::get_sender();
    deposit(top_bid, top_bidder);
    let new_auction = Auction<AssetType> { auctioneer, top_bid: assets, top_bidder: sender };
    opcode::app_global_put(txn::bytes_of_address(auctioneer), new_auction);
  }

  public fun finalize_auction<AssetType>() {
    let sender = txn::get_sender();
    let auction = opcode::app_global_get<Auction<AssetType>>(txn::bytes_of_address(sender));
    assert!(sender == auction.auctioneer, 2);
    let Auction { auctioneer, top_bid, top_bidder: _ } = auction;
    deposit(top_bid, auctioneer);
  }
}
```

AlgoMove
Code Example

Auction definition

```
module algomove::algomove_auction_paper {

  use algomove::opcode;
  use algomove::transaction as txn;
  use algomove::asset::{Asset, deposit}

  struct Auction<phantom AssetType> has key {
    auctioneer: address,
    top_bid: Asset<AssetType>,
    top_bidder: address,
  }

  public fun start_auction<AssetType>(base: Asset<AssetType>) {
    let sender = txn::get_sender();
    let auction = Auction<AssetType> {
      auctioneer: sender,
      top_bid: base,
      top_bidder: sender,
    };
    opcode::app_global_put(txn::bytes_of_address(sender), auction);
  }

  public fun bid<AssetType>(auctioneer: address, assets: Asset<AssetType>) {
    let Auction { auctioneer, top_bid, top_bidder } = opcode::app_global_get<Auction<AssetType>>(txn::bytes_of_address(auctioneer));
    assert!(get_amount(&assets) > get_amount(&top_bid), 1);
    let sender = txn::get_sender();
    deposit(top_bid, top_bidder);
    let new_auction = Auction<AssetType> { auctioneer, top_bid: assets, top_bidder: sender };
    opcode::app_global_put(txn::bytes_of_address(auctioneer), new_auction);
  }

  public fun finalize_auction<AssetType>() {
    let sender = txn::get_sender();
    let auction = opcode::app_global_get<Auction<AssetType>>(txn::bytes_of_address(sender));
    assert!(sender == auction.auctioneer, 2);
    let Auction { auctioneer, top_bid, top_bidder: _ } = auction;
    deposit(top_bid, auctioneer);
  }
}
```

AlgoMove
Code Example

```
module algomove::algomove_auction_paper {

  use algomove::opcode;
  use algomove::transaction as txn;
  use algomove::asset::{Asset, deposit}

  struct Auction<phantom AssetType> has key {
    auctioneer: address,
    top_bid: Asset<AssetType>,
    top_bidder: address,
  }

  public fun start_auction<AssetType>(base: Asset<AssetType>) {
    let sender = txn::get_sender();
    let auction = Auction<AssetType> {
      auctioneer: sender,
      top_bid: base,
      top_bidder: sender,
    };
    opcode::app_global_put(txn::bytes_of_address(sender), auction);
  }

  public fun bid<AssetType>(auctioneer: address, assets: Asset<AssetType>) {
    let Auction { auctioneer, top_bid, top_bidder } = opcode::app_global_get<Auction<AssetType>>(txn::bytes_of_address(auctioneer));
    assert!(get_amount(&assets) > get_amount(&top_bid), 1);
    let sender = txn::get_sender();
    deposit(top_bid, top_bidder);
    let new_auction = Auction<AssetType> { auctioneer, top_bid: assets, top_bidder: sender };
    opcode::app_global_put(txn::bytes_of_address(auctioneer), new_auction);
  }

  public fun finalize_auction<AssetType>() {
    let sender = txn::get_sender();
    let auction = opcode::app_global_get<Auction<AssetType>>(txn::bytes_of_address(sender));
    assert!(sender == auction.auctioneer, 2);
    let Auction { auctioneer, top_bid, top_bidder: _ } = auction;
    deposit(top_bid, auctioneer);
  }
}
```

AlgoMove
Code Example

```
module algomove::algomove_auction_paper {

  use algomove::opcode;
  use algomove::transaction as txn;
  use algomove::asset::{Asset, deposit}

  struct Auction<phantom AssetType> has key {
    auctioneer: address,
    top_bid: Asset<AssetType>,
    top_bidder: address,
  }

  public fun start_auction<AssetType>(base: Asset<AssetType>) {
    let sender = txn::get_sender();
    let auction = Auction<AssetType> {
      auctioneer: sender,
      top_bid: base,
      top_bidder: sender,
    };
    opcode::app_global_put(txn::bytes_of_address(sender), auction);
  }

  public fun bid<AssetType>(auctioneer: address, assets: Asset<AssetType>) {
    let Auction { auctioneer, top_bid, top_bidder } = opcode::app_global_get<Auction<AssetType>>(txn::bytes_of_address(auctioneer));
    assert!(get_amount(&assets) > get_amount(&top_bid), 1);
    let sender = txn::get_sender();
    deposit(top_bid, top_bidder);
    let new_auction = Auction<AssetType> { auctioneer, top_bid: assets, top_bidder: sender };
    opcode::app_global_put(txn::bytes_of_address(auctioneer), new_auction);
  }

  public fun finalize_auction<AssetType>() {
    let sender = txn::get_sender();
    let auction = opcode::app_global_get<Auction<AssetType>>(txn::bytes_of_address(sender));
    assert!(sender == auction.auctioneer, 2);
    let Auction { auctioneer, top_bid, top_bidder: _ } = auction;
    deposit(top_bid, auctioneer);
  }
}
```

AlgoMove
Code Example

```
module algomove::algomove_auction_paper {

  use algomove::opcode;
  use algomove::transaction as txn;
  use algomove::asset::{Asset, deposit}

  struct Auction<phantom AssetType> has key {
    auctioneer: address,
    top_bid: Asset<AssetType>,
    top_bidder: address,
  }

  public fun start_auction<AssetType>(base: Asset<AssetType>) {
    let sender = txn::get_sender();
    let auction = Auction<AssetType> {
      auctioneer: sender,
      top_bid: base,
      top_bidder: sender,
    };
    opcode::app_global_put(txn::bytes_of_address(sender), auction);
  }

  public fun bid<AssetType>(auctioneer: address, assets: Asset<AssetType>) {
    let Auction { auctioneer, top_bid, top_bidder } = opcode::app_global_get<Auction<AssetType>>(txn::bytes_of_address(auctioneer));
    assert!(get_amount(&assets) > get_amount(&top_bid), 1);
    let sender = txn::get_sender();
    deposit(top_bid, top_bidder);
    let new_auction = Auction<AssetType> { auctioneer, top_bid: assets, top_bidder: sender };
    opcode::app_global_put(txn::bytes_of_address(auctioneer), new_auction);
  }

  public fun finalize_auction<AssetType>() {
    let sender = txn::get_sender();
    let auction = opcode::app_global_get<Auction<AssetType>>(txn::bytes_of_address(sender));
    assert!(sender == auction.auctioneer, 2);
    let Auction { auctioneer, top_bid, top_bidder: _ } = auction;
    deposit(top_bid, auctioneer);
  }
}
```

AlgoMove
Code Example

```
module algomove::algomove_auction_paper {

  use algomove::opcode;
  use algomove::transaction as txn;
  use algomove::asset::{Asset, deposit}

  struct Auction<phantom AssetType> has key {
    auctioneer: address,
    top_bid: Asset<AssetType>,
    top_bidder: address,
  }

  public fun start_auction<AssetType>(base: Asset<AssetType>) {
    let sender = txn::get_sender();
    let auction = Auction<AssetType> {
      auctioneer: sender,
      top_bid: base,
      top_bidder: sender,
    };
    opcode::app_global_put(txn::bytes_of_address(sender), auction);
  }

  public fun bid<AssetType>(auctioneer: address, assets: Asset<AssetType>) {
    let Auction { auctioneer, top_bid, top_bidder } = opcode::app_global_get<Auction<AssetType>>(txn::bytes_of_address(auctioneer));
    assert!(get_amount(&assets) > get_amount(&top_bid), 1);
    let sender = txn::get_sender();
    deposit(top_bid, top_bidder);
    let new_auction = Auction<AssetType> { auctioneer, top_bid: assets, top_bidder: sender };
    opcode::app_global_put(txn::bytes_of_address(auctioneer), new_auction);
  }

  public fun finalize_auction<AssetType>() {
    let sender = txn::get_sender();
    let auction = opcode::app_global_get<Auction<AssetType>>(txn::bytes_of_address(sender));
    assert!(sender == auction.auctioneer, 2);
    let Auction { auctioneer, top_bid, top_bidder: _ } = auction;
    deposit(top_bid, auctioneer);
  }
}
```

Future Works

- Produce extended version with technical details
- Formalize the translation and prove correctness/equivalence
- Translate Move to other blockchains (EVM?)

Thank you

References

- Patrignani Marco, Amal Ahmed, and Dave Clarke. "**Formal approaches to secure compilation: A survey of fully abstract compilation and related work.**" ACM Computing Surveys (CSUR) 51.6 (2019): 1-36.

- Patrignani Marco and Deepak Garg. "**Robustly safe compilation.**" Programming Languages and Systems: 28th European Symposium on Programming, ESOP 2019

- Patrignani Marco and Sam Blackshear. "**Robust safety for move.**" 2023 IEEE 36th Computer Security Foundations Symposium (CSF). IEEE, 2023.

- Benetollo Lorenzo et al. "**ALGOMOVE–A Move Embedding for Algorand.**" 2023 IEEE International Conference on Blockchain (Blockchain). IEEE, 2023.

- Blackshear Sam et al. "**Move: A language with programmable resources.**" Libra Assoc (2019): 1.

- Gilad Yossi et al. "**Algorand: Scaling byzantine agreements for cryptocurrencies.**" Proceedings of the 26th symposium on operating systems principles. 2017.

- Atzei, Nicola, Massimo Bartoletti, and Tiziana Cimoli. "**A survey of attacks on ethereum smart contracts (sok).**" Principles of Security and Trust: 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017