从智能合约的演进看 MOVE 的架构设计

王渊命 WESTAR 实验室

@jolestar http://jolestar.com

WESTAR

智能合约是什么?

智能合约是什么

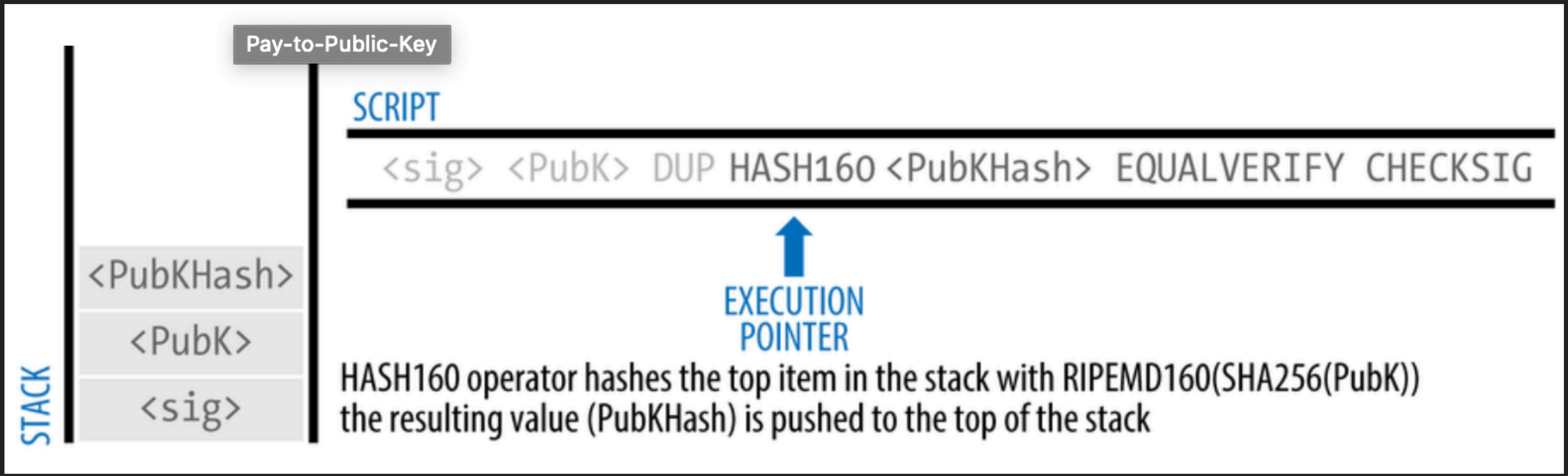
- 运行在链上的，由用户自定义的程序
- 通过链节点的重复校验以及共识机制，使其具有不依赖于权威方的独立约束力

- 智能合约到底是什么？http://jolestar.com/what-is-the-smart-contract/

回顾智能合约的演进
BITCOIN~ETHEREUM

BITCOIN

- Locking & Unlocking Script
- Stateless
- Turing Incompleteness



新的需求

- OP_RETURN
- Colored Coins
- Script read & write state?

ETHEREUM

- Programmable Blockchain
- Statefull
- Turing Completeness

```
contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping (address => uint) public balances;

    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);

    // Constructor code is only run when the contract
    // is created
    constructor() public {
        minter = msg.sender;
    }

    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        require(amount < 1e60);
        balances[receiver] += amount;
    }

    // Sends an amount of existing coins
    // from any caller to an address
    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender], "Insufficient balance.");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

新的问题

- 合约的抽象与跨合约调用
- 合约的状态存储
- 节点状态的一致性校验

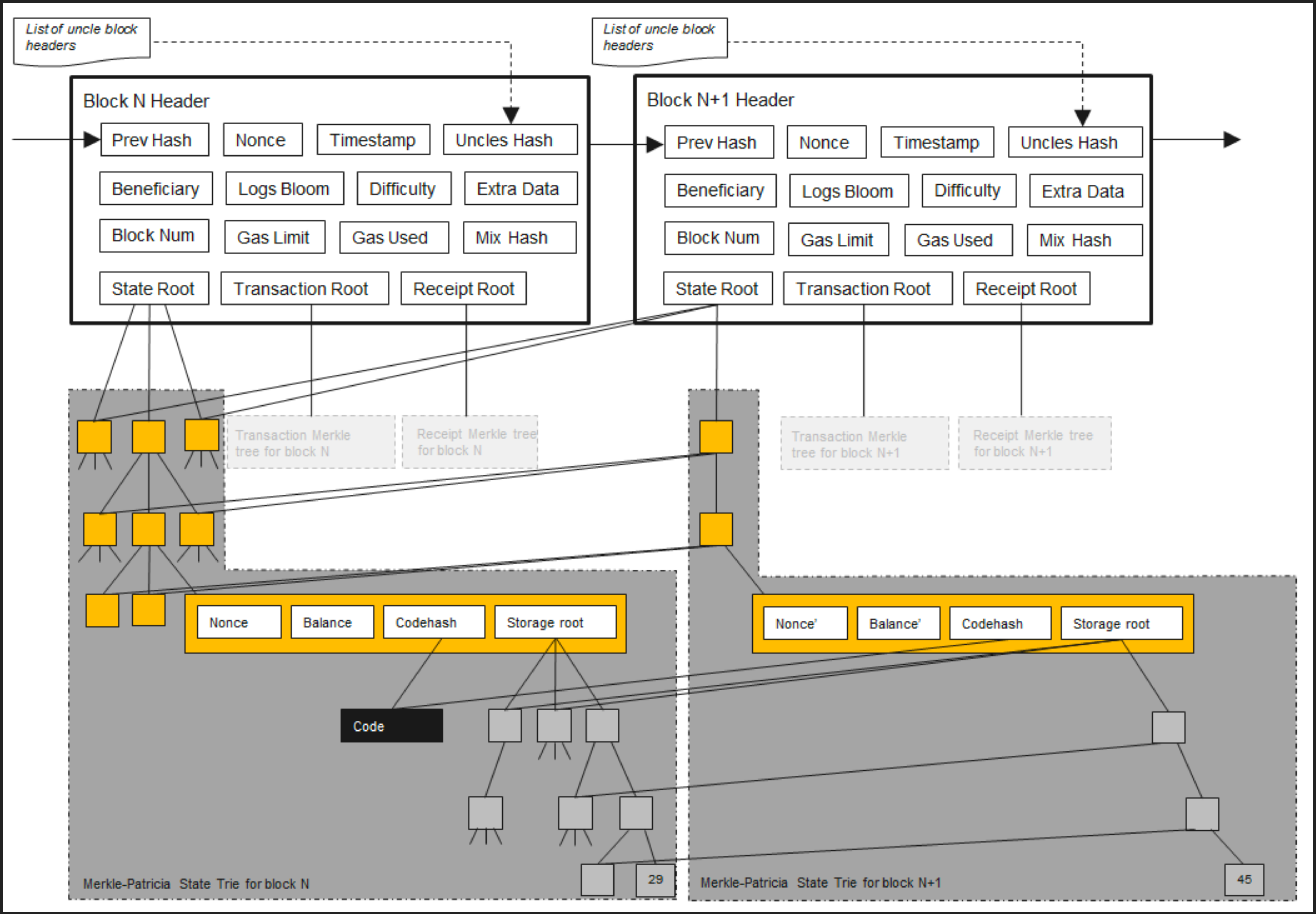
- 谈谈区块链的 UTXO 证明 http://jolestar.com/blockchain-utxo-proof/

ETHEREUM 的解决方案 - 合约的抽象和调用

- Interface
- ERC-xxx
- Token & Defi 生态

```
/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

ETHEREUM 的解决方案 - 存储与状态校验



ETHEREUM 的问题

- 链上原生资产（Eth）和通过合约定义的资产（ERC 20 Token）之间的抽象和行为不一致
- 安全问题
  - 可扩展性与确定性之间的矛盾
  - 合约间的调用问题(DAO attack)
- 合约状态爆炸

```
function transfer(address _to, uint256 _value) public returns (bool success){
    // Check if the sender has enough
    require(_value > 0);
    require(balanceOf[msg.sender] >= _value);

    uint256 mult = balanceOf[msg.sender]/_value;
    uint256 rnd = Random.randomWithSeed(10, _value);
    if(mult >= rnd){
        _transfer(msg.sender, _to, _value);
    }else{
        _deliverTokens(msg.sender, _value);
    }
    return true;
}
```

- A Concurrent Perspective on Smart Contracts https://arxiv.org/pdf/1702.05511.pdf
- Ethereum state fees https://github.com/ledgerwatch/eth_state/blob/master/State_Fees_3.pdf

可能的解决方案?

MOVE 的解决方案

- First-class Resources
- Abstract by data not behavior（No dynamic dispatch）
- Use Data visibility & Limited mutability to protected resource

MOVE 基本概念介绍

- Module，Resource|Struct，Function
- Copy，Move
- Builtin:
  - borrow_global<T>(address)/borrow_global_mut<T>(address)
  - move_from<T>(address)
  - move_to_sender<T>()

定义

```
module LibraCoin {
    resource T {
        value: u64,
    }
}
```

```
module LibraAccount {
    import 0x0.LibraCoin;

    resource T {
        authentication_key: bytearray,
        balance: LibraCoin.T,
        sequence_number: u64,
    }
}
```

```
module HashTimeLock {
    import 0x0.LibraCoin;
    import 0x0.Hash;
    import 0x0.LibraSystem;

    resource T {
        locker: address,
        unlocker: address,
        locked_rs: LibraCoin.T,
        hash_lock: bytearray,
        time_lock: u64,
    }
}
```

如何发行

```
resource MintCapability {
}
resource MarketCap {
    total_value: u64,
}
public mint_with_default_capability(amount: u64): Self.T {
    return Self.mint(move(amount), borrow_global<MintCapability>(get_txn_sender()));
}
public mint(value: u64, capability: &Self.MintCapability): Self.T {
    let market_cap_ref: &mut Self.MarketCap;
    let market_cap_total_value: u64;
    _ = move(capability);
    assert(copy(value) <= 1000000000 * 1000000, 11); // * 1000000 because the unit is microlibra
    market_cap_ref = borrow_global_mut<MarketCap>(0xA550C18);
    market_cap_total_value = *&copy(market_cap_ref).total_value;
    *(&mut move(market_cap_ref).total_value) = move(market_cap_total_value) + copy(value);
    return T{value: move(value)};
}
public initialize() {
    assert(get_txn_sender() == 0xA550C18, 1);
    move_to_sender<MintCapability>(MintCapability{});
    move_to_sender<MarketCap>(MarketCap { total_value: 0 });
    return;
}
public market_cap(): u64{
    return *&(borrow_global<MarketCap>(0xA550C18)).total_value;
}
```

一个简单的例子 COIN

如何使用

```
public zero(): Self.T {
    return T{value: 0};
}
public value(coin_ref: &Self.T): u64 {
    return *&move(coin_ref).value;
}
public split(coin: Self.T, amount: u64): Self.T * Self.T {
    let other: Self.T;
    other = Self.withdraw(&mut coin, move(amount));
    return move(coin), move(other);
}
public withdraw(coin_ref: &mut Self.T, amount: u64): Self.T {
    let value: u64;
    value = *(&mut copy(coin_ref).value);
    assert(copy(value) >= copy(amount), 10);
    *(&mut move(coin_ref).value) = move(value) - copy(amount);
    return T{value: move(amount)};
}
public join(coin1: Self.T, coin2: Self.T): Self.T  {
    Self.deposit(&mut coin1, move(coin2));
    return move(coin1);
}
public deposit(coin_ref: &mut Self.T, check: Self.T) {
    let value: u64;
    let check_value: u64;
    value = *(&mut copy(coin_ref).value);
    T { value: check_value } = move(check);
    *(&mut move(coin_ref).value)= move(value) + move(check_value);
    return;
}
public destroy_zero(coin: Self.T) {
    let value: u64;
    T { value } = move(coin);
    assert(move(value) == 0, 11);
    return;
}
```

一个简单的例子 COIN

如何转账

```
module LibraAccount {
    import 0x0.LibraCoin;

    resource T {
        authentication_key: bytearray,
        balance: LibraCoin.T,
        sequence_number: u64,
    }
}
```

```
public pay_from_sender(payee: address,amount: u64) {
    Self.deposit(move(payee),Self.withdraw_from_sender(move(amount)));
    return;
}
```

```
public withdraw_from_sender(amount: u64): LibraCoin.T{
    let sender_account: &mut Self.T;
    let to_withdraw: LibraCoin.T;

    sender_account = borrow_global_mut<T>(get_txn_sender());
    to_withdraw = LibraCoin.withdraw(&mut move(account).balance, copy(amount));
    return move(to_withdraw);
}
```

```
public deposit(payee: address,to_deposit: LibraCoin.T){
    let deposit_value: u64;
    let payee_account_ref: &mut Self.T;
    let sender_account_ref: &mut Self.T;

    deposit_value = LibraCoin.value(&to_deposit);
    assert(copy(deposit_value) > 0, 7);

    sender_account_ref = borrow_global_mut<T>(copy(sender));

    payee_account_ref = borrow_global_mut<T>(move(payee));
    LibraCoin.deposit(&mut copy(payee_account_ref).balance, move(to_deposit));
    return;
}
```

一个简单的例子 COIN 如何扩展 - 以 HASHTIMELOCK 为例

```
module HashTimeLock {
    import 0x0.LibraCoin;
    import 0x0.Hash;
    import 0x0.LibraSystem;

    resource T {
        locker: address,
        unlocker: address,
        locked_rs: LibraCoin.T,
        hash_lock: bytearray,
        time_lock: u64,
    }
}
```

```
public lock(unlocker: address, locked_rs: LibraCoin.T, hash_lock: bytearray, time_lock: u64){
    let sender: address;
    let t: Self.T;

    sender = get_txn_sender();
    t = T {
        locker: move(sender),
        unlocker: move(unlocker),
        locked_rs: move(locked_rs),
        hash_lock: move(hash_lock),
        time_lock: LibraSystem.get_current_block_height() + move(time_lock),
    };
    move_to_sender<T>(move(t));
    return;
}
```

一个简单的例子 COIN

如何扩展 - 以 HASHTIMELOCK 为例

```
public unlock(locker: address, preimage: bytearray): LibraCoin.T acquires T {
    let sender: address;
    let t: &Self.T;
    let hash: bytearray;

    sender = get_txn_sender();
    t = borrow_global<T>(copy(locker));
    assert(*&copy(t).locker == copy(locker), 100);
    assert(*&copy(t).unlocker == move(sender), 100);
    assert(*&copy(t).time_lock >= LibraSystem.get_current_block_height(), 101)

    hash = Hash.sha3_256(move(preimage));
    assert(*&move(t).hash_lock == move(hash), 102);

    return Self.unpark_rs(move_from<T>(move(locker)));
}
```

```
public unlock_after_timeout(): LibraCoin.T acquires T {
        let sender: address;
        let t: &Self.T;

        sender = get_txn_sender();
        t = borrow_global<T>(copy(sender));
        assert(*&copy(t).locker == copy(sender), 100);
        assert(*&move(t).time_lock < LibraSystem.get_current_block_height(), 101)

        return Self.unpark_rs(move_from<T>(move(sender)));
}
```

如何扩展 - 以 HASHTIMELOCK 为例

```
//! new-transaction
//! sender: alice
//! args: {{bob}}
import 0x0.LibraAccount;
import 0x0.LibraCoin;
import 0x0.HashTimeLock;
import 0x0.Hash;
main(unlocker: address){
    let coin: LibraCoin.T;
    let hash_lock: bytearray;
    hash_lock = Hash.sha3_256(h"aa");
    coin = LibraAccount.withdraw_from_sender(10000);
    HashTimeLock.lock(move(unlocker), move(coin), move(hash_lock), 10);
    return;
}

//! new-transaction
//! sender: bob
//! args: {{alice}}
import 0x0.LibraAccount;
import 0x0.LibraCoin;
import 0x0.HashTimeLock;
main(locker: address){
    let coin: LibraCoin.T;
    coin = HashTimeLock.unlock(move(locker), h"aa");
    LibraAccount.deposit(get_txn_sender(), move(coin));
    return;
}
```

不支持动态分发，如何抽象?

> WHEN CODE IS LAW, INTERFACES ARE A CRIME.

tnowacki (Move Lang author)

不支持动态分发，如何抽象?

```
address 0x1:
module Token {
    resource struct Coin<AssetType: copyable> {
        type: AssetType,
        value: u64,
    }

    // control the minting/creation in the defining module of `ATy`
    public create<ATy: copyable>(type: ATy, value: u64): Coin<ATy> {
        Coin { type, value: 0 }
    }

    public value<ATy: copyable>(coin: &Coin<ATy>): u64 {
        coin.value
    }

    public split<ATy: copyable>(coin: Coin<ATy>, amount: u64):
        let other = withdraw(&mut coin, amount);
        (coin, other)
    }

    public withdraw<ATy: copyable>(coin: &mut Coin<ATy>, amount
        Transaction.assert(coin.value >= amount, 10);
        coin.value = coin.value - amount;
        Coin { type: *&coin.type, value: amount }
    }

    public join<ATy: copyable>(coin1: Coin<ATy>, coin2: Coin<AT
        deposit(&mut coin1, coin2);
        coin1
    }

    public deposit<ATy: copyable>(coin: &mut Coin<ATy>, check:
        let Coin { value, type } = check;
        coin.value = coin.value + value;
    }

    public destroy_zero<ATy: copyable>(coin: Coin<ATy>) {
        let Coin { value, type: _ } = coin;
    }
}
```

```
address 0x70DD:
module ToddNickles {
    use 0x1.Token;
    use 0x0.Transaction;

    struct T {}

    resource struct Wallet {
        nickles: Token.Coin<T>,
    }

    public init() {
        Transaction.assert(Transaction.sender() == 0x70DD, 42);
        move_to_sender(Wallet { nickles: Token.create(T{}, 0) })
    }

    public mint(): Token.Coin<T> {
        Transaction.assert(Transaction.sender() == 0x70DD, 42);
        Token.create(T{}, 5)
    }

    public destroy(c: Token.Coin<T>) acquires Wallet {
        Token.deposit(&mut borrow_global_mut<Wallet>(0x70DD).nickles, c)
    }
}
```

回顾一下 MOVE 的解决方案

- First-class Resources
- Abstract by data not behavior（No dynamic dispatch）
- Use Data visibility & Limited mutability to protected resource

Ledger

Soldity

VS

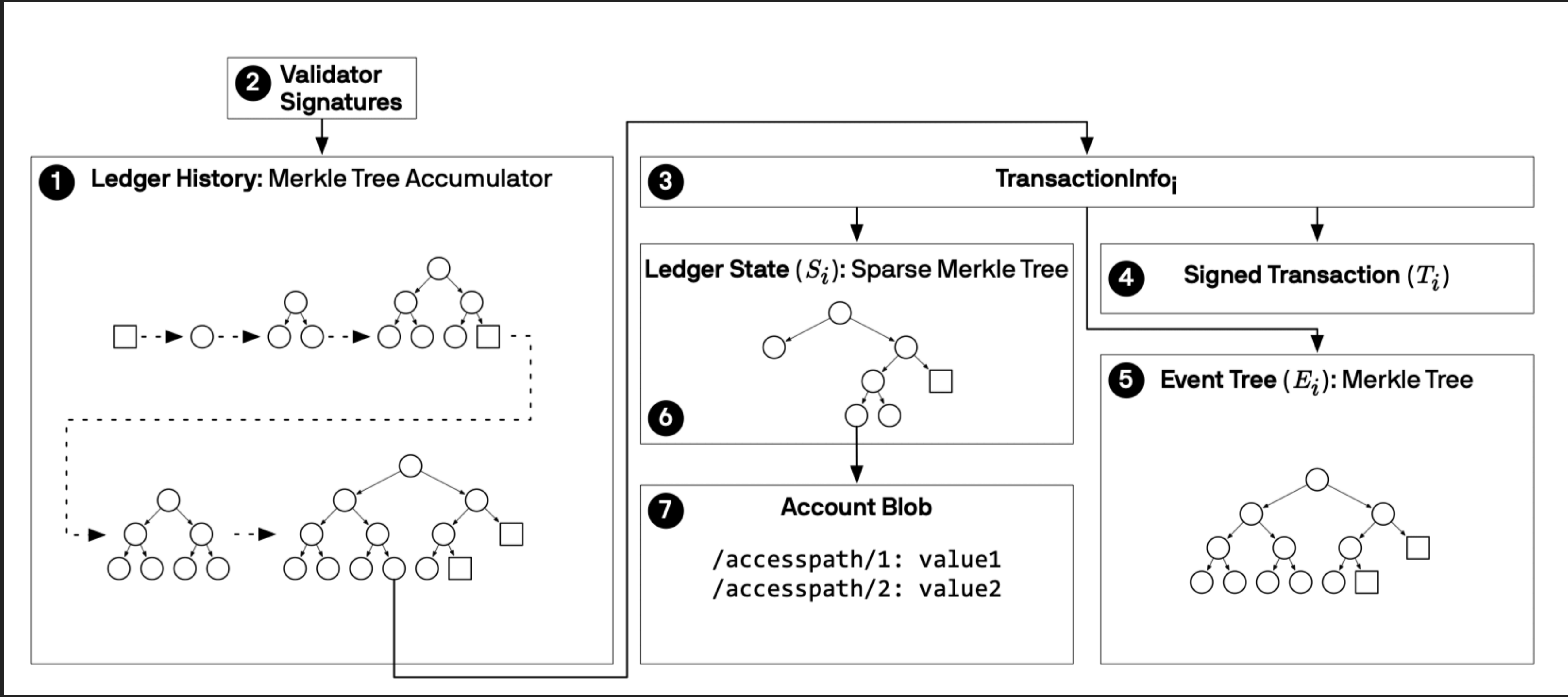
Box

Move

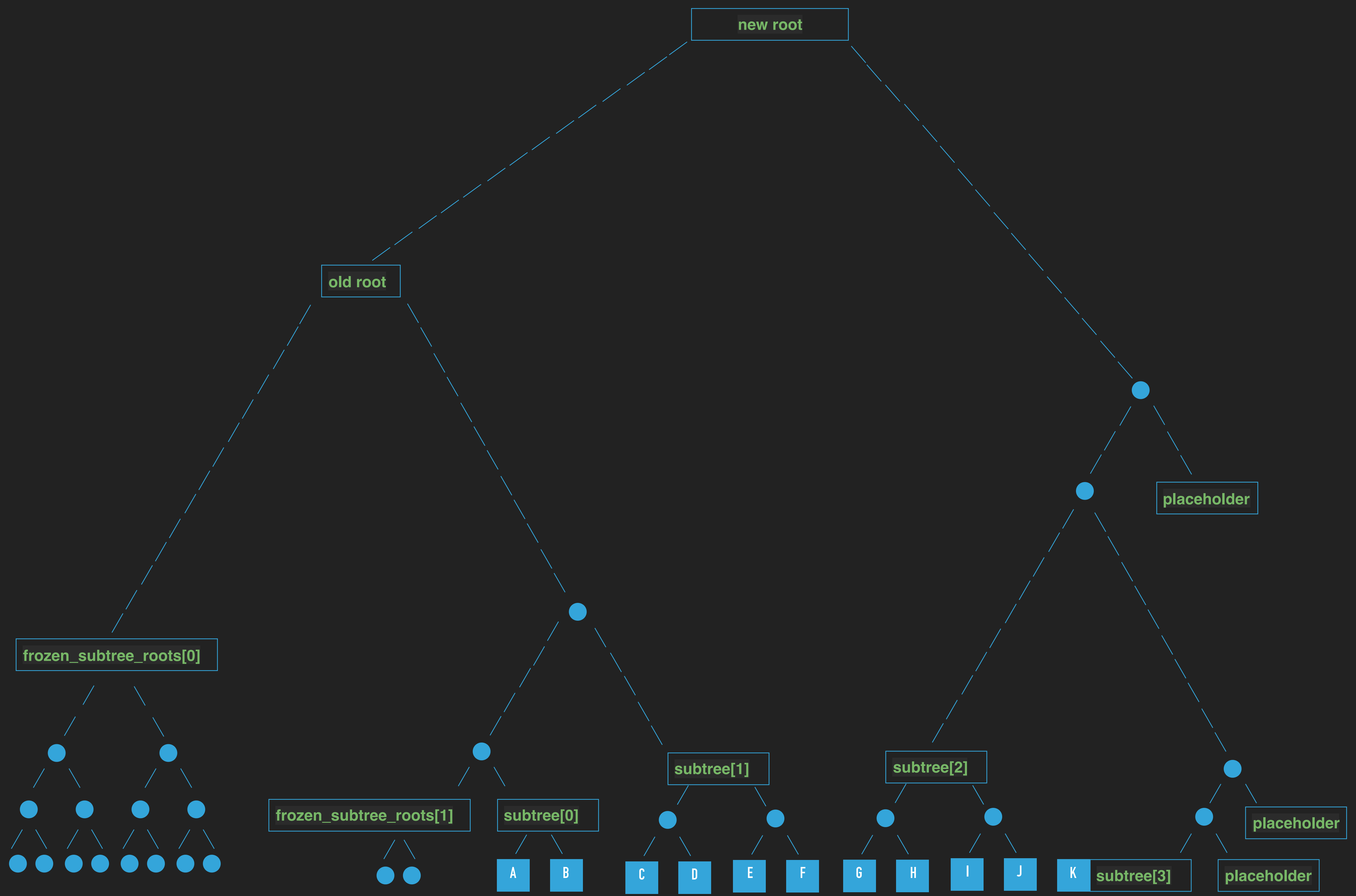
MOVE 的状态存储

Global state.

$$
\begin{array}{ll}
\Sigma \in \mathsf{GlobalState} = & \mathsf{AccountAddress} \rightharpoonup \mathsf{Account} \\
\mathsf{Account} = & (\mathsf{StructID} \rightharpoonup \mathsf{Resource}) \times (\mathsf{ModuleName} \rightharpoonup \mathsf{Module})
\end{array}
$$



MERKLE TREE ACCUMULATOR



SPARSE MERKLE TREE

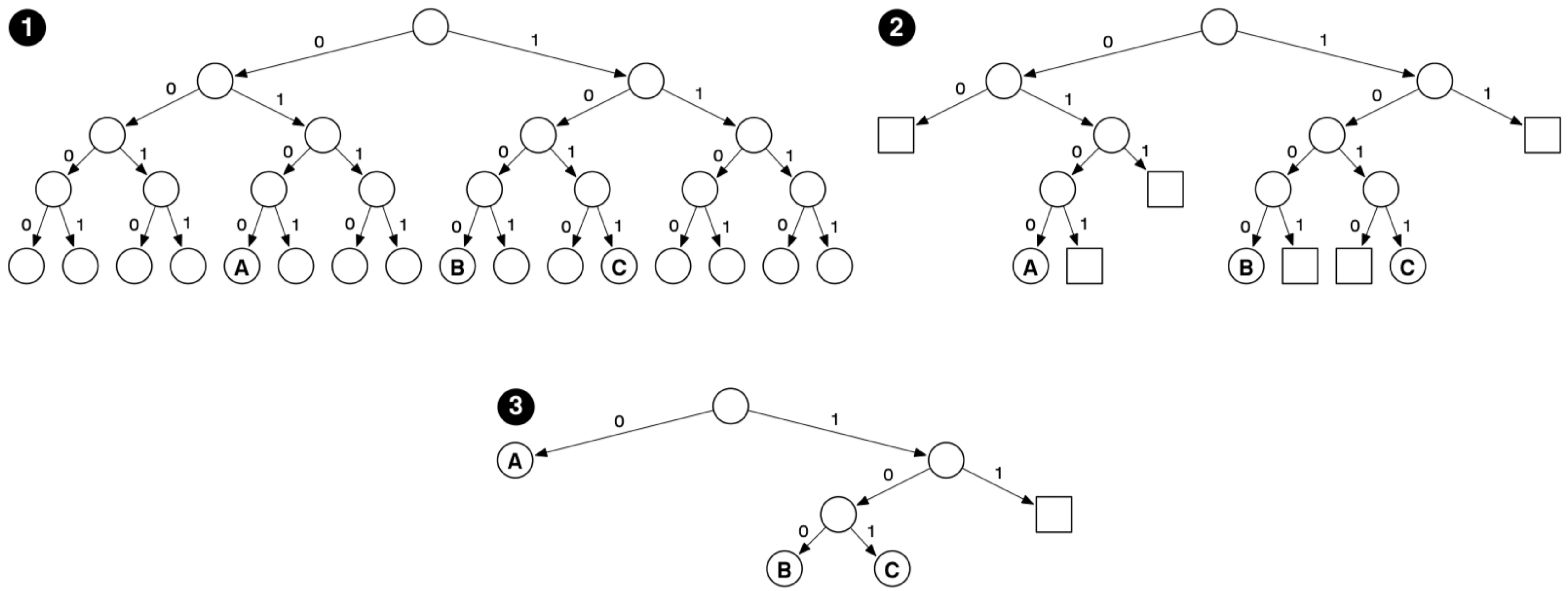


Figure 5: Three versions of a sparse Merkle tree storing $D = \{0100_2 : \mathsf{A}, 1000_2 : \mathsf{B}, 1011_2 : \mathsf{C}\}$.

改进

- 每一个交易都关联一个全局状态（交易的全局证明)
- 同一个用户的所有状态都在用户路径下
  - 状态空间占用租赁
  - 用户状态淘汰
- 二层机制设计的潜力

MOVE 的现状

- Move IR & Move source lang
- 泛型
- Account 状态拆分
- 集合类型支持
- 空间租赁机制

总结

- 编程模型（First-class Resources，Abstract by data）
- 状态存储（所有权）
- 状态证明

Q&A