MOVE FOR SOLIDITY DEVS

@rahatcodes

AGENDA:

- INTRO TO MOVEMENT
- THE MOVE LANGUAGE
- DEPLOYING A MOVE MODULE
- MORE LEARNING RESOURCES
- OPPORTUNITIES TO BUILD

1

INTRO TO MOVEMENT

WHAT IS MOVEMENT?

L1 Move Blockchain

Move token for Gas and Staking

POS

High Throughput chain with sub-second finality

WHAT IS MOVEMENT?

Parallel Transaction execution

Built on the Aptos Framework

The peoples chain

2

MOVE LANGUAGE

CODE

MOVE IS A RESOURCE ORIENTED PROGRAMMING LANGUAGE

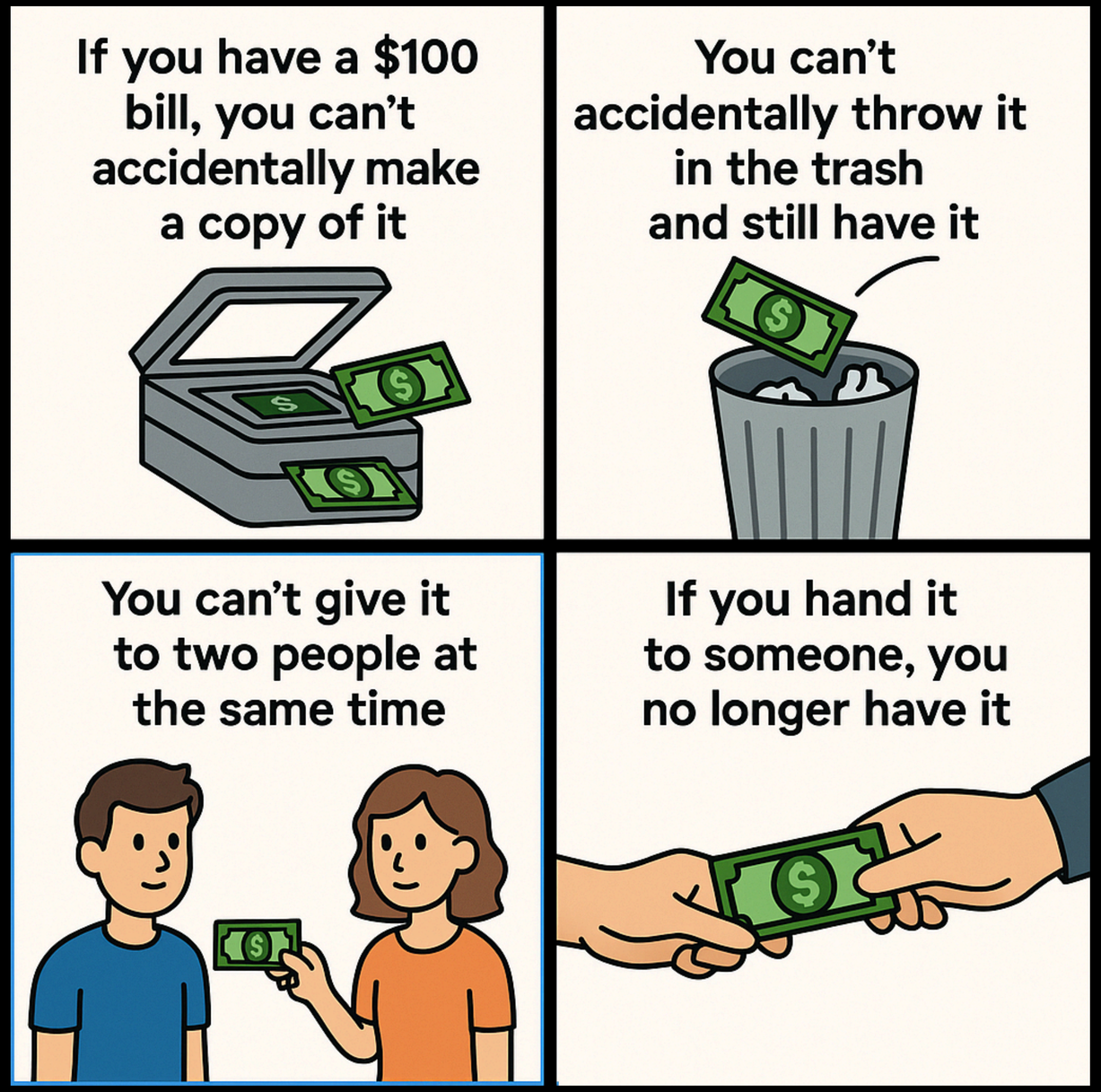
MOVE

Resources can't be copied or accidentally lost

Linear type system prevents double-spending

Bytecode verification before execution

No reentrancy vulnerabilities


If you have a $100 bill, you can't accidentally make a copy of it
You can't accidentally throw it in the trash and still have it
You can't give it to two people at the same time
If you hand it to someone, you no longer have it

SOLIDITY

Managing Balances

```
mapping(address => uint256) public balances;
    uint256 public totalSupply;

    function mint(address to, uint256 amount) public {
        // Manual updates - easy to mess up
        balances[to] += amount;
        totalSupply += amount;

        // Problems:
        // ❌ Could duplicate: balances[to] += amount;
(again) // ❌ Could forget: totalSupply += amount;
        // ❌ Compiler doesn't protect you
    }
```

MOVE

Managing Balances

```
        let user_address = signer::address_of(account);
        let management = borrow_global<Management>(fusd_address());

        // Creates a resource that MUST be used
        let tokens = fungible_asset::mint(&management.mint_ref,
amount);
        // MUST deposit - compiler enforces
        primary_fungible_store::deposit(user_address, tokens);

        // Guarantees:
        // ✅ Can't duplicate tokens (no copy ability)
        // ✅ Can't forget deposit (compile error)
        // ✅ Total supply updated automatically
```

SOLIDITY

Transferring Funds

```
function transfer(address to, uint256 amount) public returns (bool)
{       require(balances[msg.sender] >= amount, "Insufficient");
        require(to != address(0), "Invalid recipient");

        // Manual arithmetic - must be in correct order
        balances[msg.sender] -= amount;  // Deduct from sender
        balances[to] += amount;          // Add to recipient

        emit Transfer(msg.sender, to, amount);
        return true;

        // Problems:
        // ❌ Could deduct but forget to add (lose tokens)
        // ❌ Could add but forget to deduct (create tokens)
        // ❌ External calls between steps = reentrancy
        // ❌ Overflow/underflow (pre-0.8)
    }
```

MOVE

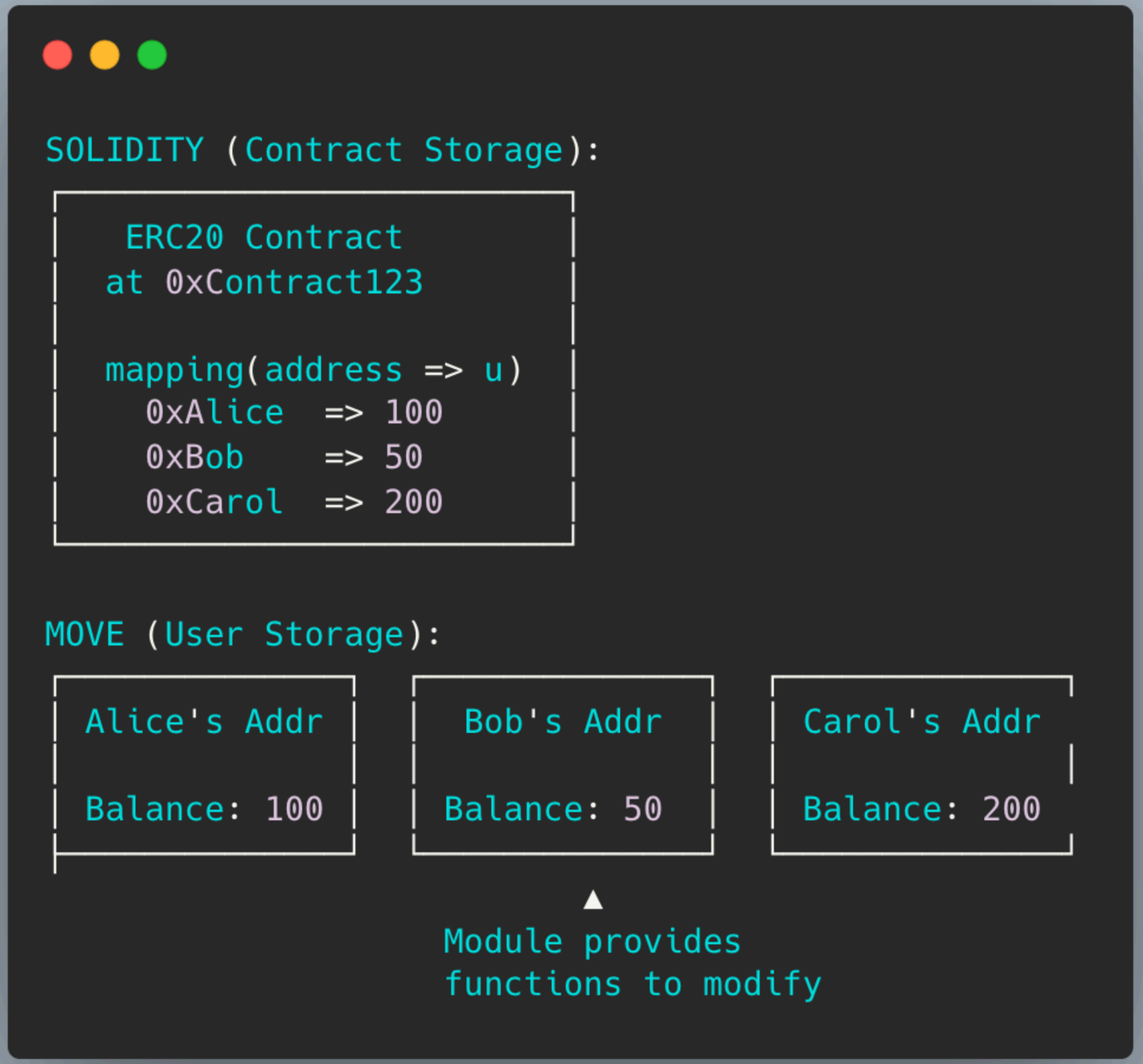
Transferring Funds

```
public fun transfer(
    from: &signer,
    to: address,
    metadata: Object<Metadata>,
    amount: u64
) {
    // Withdraw returns resource
    let tokens = withdraw(from, metadata, amount);

    // Deposit consumes resource
    deposit(to, tokens);

    // Guarantees:
    // ✅ Can't withdraw without depositing (compile error)
    // ✅ Can't deposit twice (resource moved)
    // ✅ Cant lose tokens between steps
    // ✅ Balance invariant guaranteed
    // ✅ No reentrancy possible
}
```

STORAGE


SOLIDITY (Contract Storage):
ERC20 Contract
at 0xContract123
mapping(address => u)
0xAlice => 100
0xBob => 50
0xCarol => 200
MOVE (User Storage):
Alice's Addr
Balance: 100
Bob's Addr
Balance: 50
Carol's Addr
Balance: 200
Module provides
functions to modify

SOLIDITY ACCESS CONTROL

```
address public owner;
    mapping(address => uint256) public balances;

    constructor() {
        owner = msg.sender;  // Store deployer's address
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Not owner");
        _;
    }

    // Only owner can mint
    function mint(address to, uint256 amount) public onlyOwner
{       balances[to] += amount;
        totalSupply += amount;
    }
```

CODE

MOVE

Global Storage Operations

```
// 1. STORE: Put a resource at an address
move_to<T>(&signer, resource)

// 2. REMOVE: Take a resource from an address
move_from<T>(address): T

// 3. BORROW (immutable): Read a resource
borrow_global<T>(address): &T

// 4. BORROW (mutable): Modify a resource
borrow_global_mut<T>(address): &mut T

// 5. CHECK: Does resource exist?
exists<T>(address): bool
```

COMPARISON

Feature	Solidity ERC20	Move FUSD
Storage	Contract mapping	User account resources
Balance Location	`balances[addr]`	`primary_fungible_store` at user addr
Minting	Manual `balances[to] += amount`	`fungible_asset::mint()` + deposit
Burning	Manual `balances[from] -= amount`	`withdraw()` + `burn()`
Total Supply	Manual tracking	Framework automatic
Access Control	`onlyOwner` modifier	Capability refs (`MintRef`, `BurnRef`)
Transfer Safety	Manual checks	Linear types (can't lose tokens)
Reentrancy	ReentrancyGuard needed	`acquires` prevents it
Upgrades	Proxy pattern	Native module upgrade
Parallel TXs	Sequential (shared storage)	Parallel (user storage)


LETS DEPLOY

CODE

MORE RESOURCES

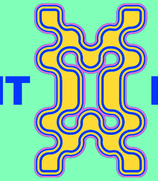
RESOURCES:

DOCS
FORUM
DISCORD
MOVE BUILDERS CHAT


5
OPPORTUNITIES TO BUILD


Movement
M1
HACKATHON

THANK YOU

THE MOVEMENT  MOVE INDUSTRIES