Introduction to Move

Objectives:

- Build smart contracts on Sui with strong foundations and patterns
- Learn Move language and the unique Sui object-centric model

Agenda

1. What is Move?
2. Toolchain & Environment Setup
3. Variables, Data Types, and Mutability
4. Sui’s Smart Contract Patterns
5. Capabilities in Sui
6. Error Handling and Security Practices

Learning Objectives

- Key takeaways: Move ensures safety, Sui's object model and capabilities enable advanced patterns.
- Explore resources and exercises for deeper learning.
- Sui Developer Portal for further resources. (https://sui.io/developers)

What is Move?

Safe & Flexible

Move is designed for secure and adaptable smart contracts on Sui.

Resource-oriented

It manages assets safely, preventing common vulnerabilities like reentrancy.

Sui Usage

Move controls on-chain objects, ensuring efficient and secure data management.

Toolchain & Environment Setup

Install Sui CLI
Ensure scarcity and prevent duplication of digital assets.

Initialize a new package
sui move new <project>

Variables, Data Types, and Mutability

Ownership model

Variables follow Rust-like ownership rules for memory safety.

Data types

- bool: true/false
- u8, u64: unsigned integers
- address: account identifiers
- vector: dynamic arrays

Mutability

let creates immutable variables, let mut makes them mutable.

```
module sui_move::sui_move{
    let Variable : Type
    let Variable = Expression
    let Variable : Type = Expression

    //example
    let a;
    let b : u8;
    let c = true;
    let d : u8 = 10;
}
```

(https://move-book.com/reference/primitive-types)

Resources and Objects

Resources

Ensure scarcity and prevent duplication of digital assets.

Objects

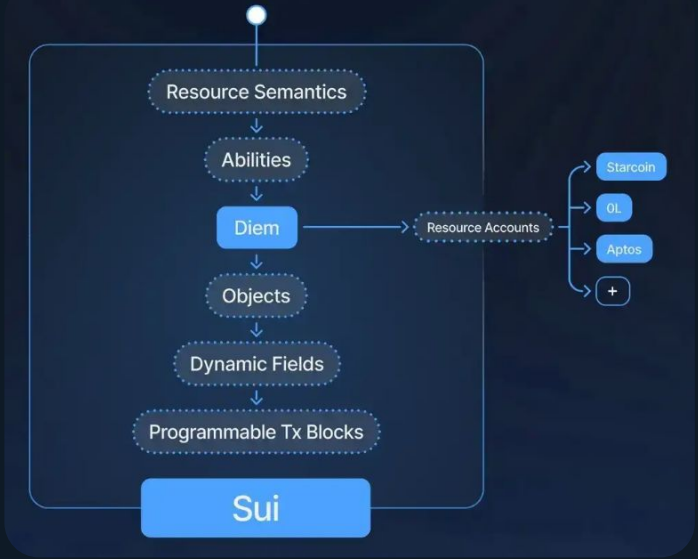
Sui's core data structures; can be owned, shared, or immutable.

Ownerships

Wrapping enforces single ownership to avoid unauthorized copies.

Transfer

Use transfer::transfer to safely move object ownership.



Models and Functions

Modules

Contain code defining structs and functions for organization.

Functions

Serve as entry points implementing on-chain business logic.

Visibility

Public: accessible within modules

Private: restricted access

Entry: callable from transactions

Example

A module creating and managing fungible tokens on Sui.

Basic Structure of a Move Project

Move.toml

- The main configuration file of the project
- Contains information about the project name, dependencies, and published addresses
- Similiar to package.json in NodeJS or Cargo.toml in Rust

sources/

- Directory containing the main source code of the project
- .move files contain smart contract code
- Each Move module is defined in a separate file

tests/

- Directory containing test files
- Test files usually have the suffix _test.move.
- Used to write unit tests for smart contracts

Common Design Patterns

Data as objects

Sui's model stores data as objects (owned or shared)

Capabilities in Sui

Capabilities control access, ensuring only authorized actions

One-time Witness Pattern

Ensures an action can only be performed once by a uniquely created object as proof.

```
module sui_move::sui_move{
    public struct ONE_TIME has drop {}

    fun init(otw: ONE_TIME, ctx: &mut TxContext) {
        // do something with the OTW
    }
}
```

Security Considerations

Gas Optimization

Minimize costs to avoid denial-of-service from expensive transactions.

Reentrancy Prevention

Resource model helps block reentrancy attacks efficiently.

Formal Verification

Mathematically prove critical contract logic correctness.

Practical Exercises

- Sui Documentation: Writing Your First Smart Contract for guidance. (https://docs.sui.io/guides/developer/writing-your-first-smart-contract)
- Build a smart contract: creating a token, transferring ownership, managing resources with capabilities.

Thank You.