Move

the Language for Secure Next Gen Smart Contracts

Mirko Zichichi
mirko.zichichi@iota.org

Mirko Zichichi

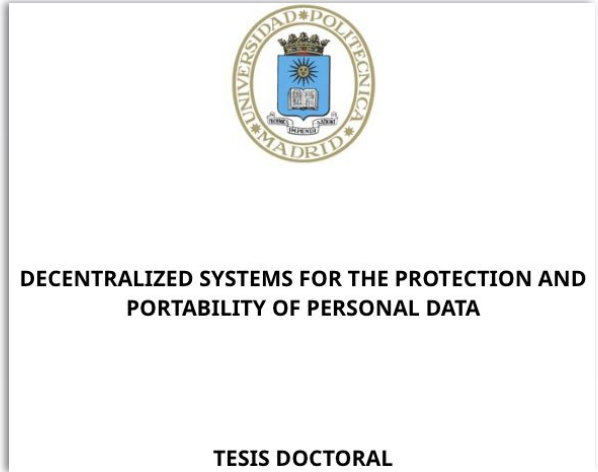
"Lavoro nel mondo crypto ma, purtroppo, dal lato di quelli che si possono permettere una Fiat Panda invece di una Lamborghini"

Mirko Zichichi

"Lavoro nel mondo crypto ma, purtroppo, dal lato di quelli che si possono permettere una Fiat Panda invece di una Lamborghini"

→ *BSc e MSc Informatica, Università degli Studi di Palermo (2017) - Università di Bologna (2019)*

→ *PhD in **Law, Science and Technology**, Universidad Politécnica de Madrid (2023)*



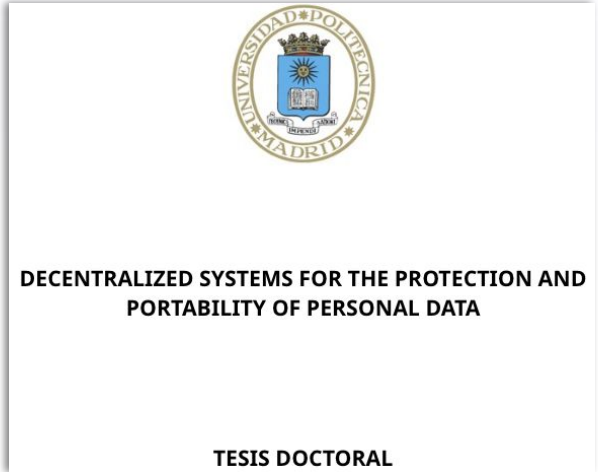
Mirko Zichichi

"Lavoro nel mondo crypto ma, purtroppo, dal lato di quelli che si possono permettere una Fiat Panda invece di una Lamborghini"

→ *BSc e MSc Informatica, Università degli Studi di Palermo (2017) - Università di Bologna (2019)*

→ *PhD in **Law, Science and Technology**, Universidad Politécnica de Madrid (2023)*

→ *Applied Research Engineer alla **IOTA Foundation** (2022)* IOTA Smart Contracts



Arancina

Arancino

the Language for Secure Next Gen Smart Contracts

Risorse

Versatilità

Sicurezza


Risorse

Libra

https://www.theverge.com/2019/6/26/18716326/facebook-libra-cryptocurrency-blockchain-irs-starbucks

Libra → Diem →

https://www.theverge.com/2019/6/26/18716326/facebook-libra-cryptocurrency-blockchain-irs-starbucks

*"Ehi, **Libra** avrà degli **smart contracts**, è importante garantire che la loro programmazione sulla blockchain sia sicura"*

Qualche capoccia di Facebook, 2017 circa.

Blackshear, Sam, et al. "Move: A Language With Programmable Resources"
https://diem-developers-components.netlify.app/papers/diem-move-a-language-with-programmable-resources/2020-05-26.pdf (2020).

"La risorsa più scarsa al mondo non sono il tempo e il denaro, ma sono le risorse mentali dell'uomo.

Quando queste vengono utilizzare per sviluppare software, se si può amplificare la capacità cerebrale, ovvero fare di più per unità di tempo, questa è una delle cose di maggior impatto che si possono realizzare."

Blackshear, Sam, et al. "Move: A Language With Programmable Resources"
https://diem-developers-components.netlify.app/papers/diem-move-a-language-with-programmable-resources/2020-05-26.pdf (2020).

Move → programmazione orientata alle risorse

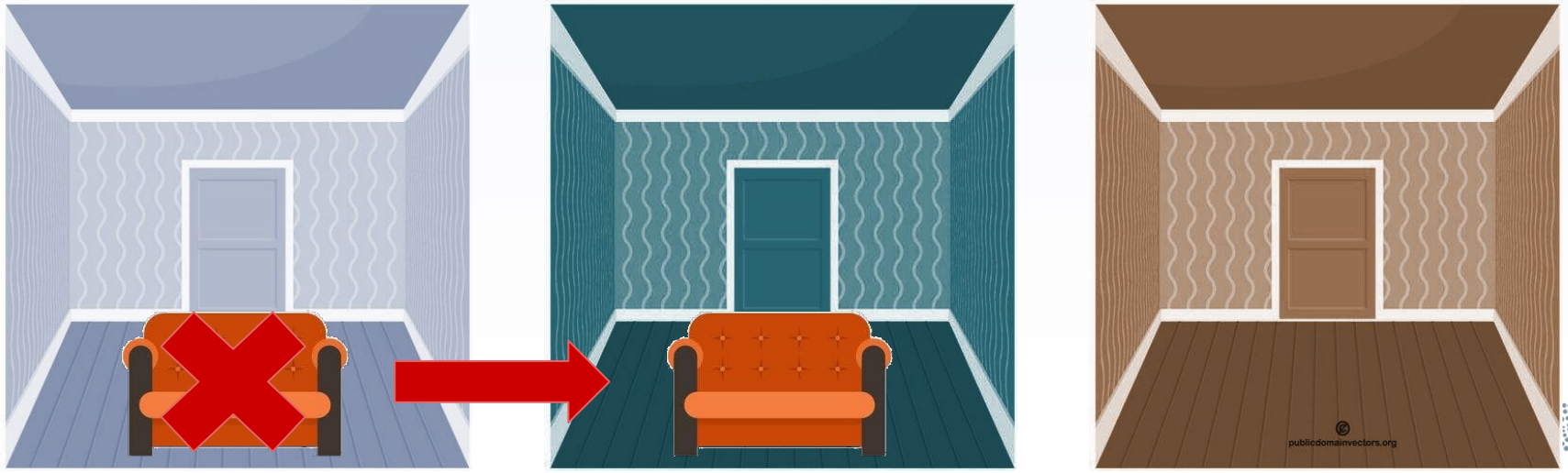
Consente di scrivere programmi che si comportano in linea con le *proprie intuizioni fisiche*.

Move → programmazione orientata alle risorse

Consente di scrivere programmi che si comportano in linea con le *proprie intuizioni fisiche*.

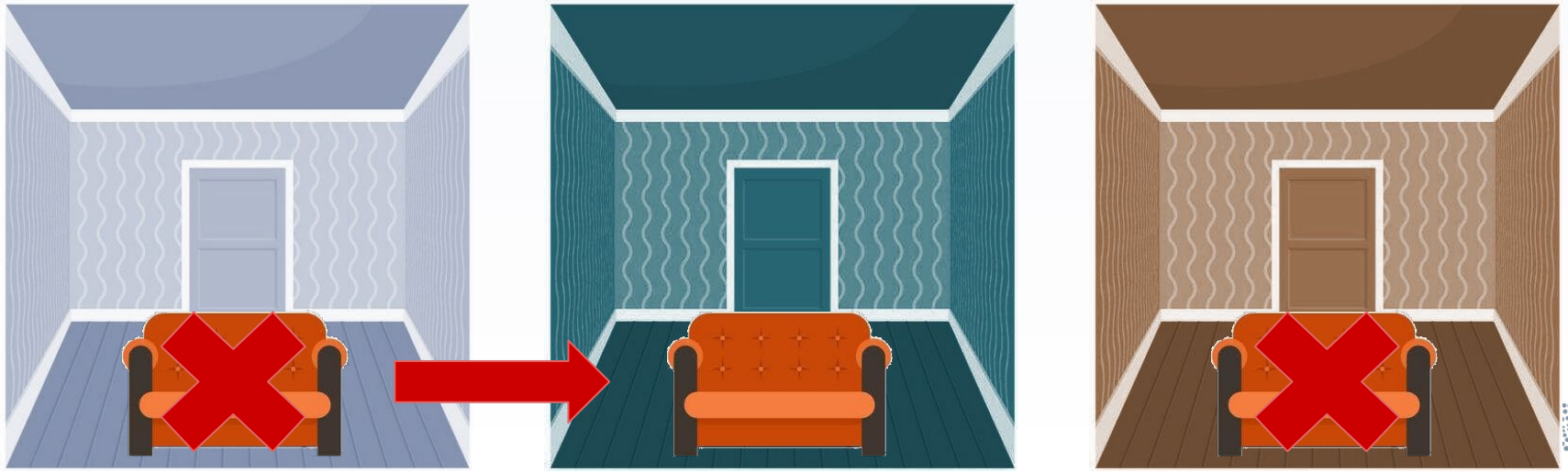
Move → programmazione orientata alle risorse

Consente di scrivere programmi che si comportano in linea con le *proprie intuizioni fisiche*.



Move → programmazione orientata alle risorse

Consente di scrivere programmi che si comportano in linea con le *proprie intuizioni fisiche*.



Move → programmazione orientata alle risorse

- Esperienza di programmazione **tangibile**

- Legata alle intuizioni fisiche di
  - **Scambio** → movement, transfer
  - **Proprietà** → access control, ownership

Critiche ai linguaggi blockchain esistenti

Critiche ai linguaggi blockchain esistenti
→ Ethereum Virtual Machine/Solidity

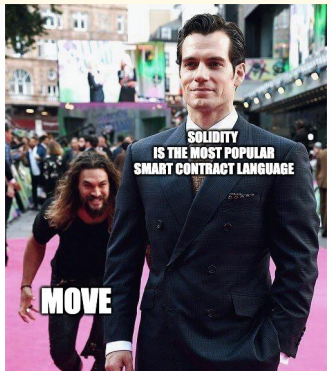
ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER

ISTANBUL VERSION 80085f7 – 2021-07-11

DR. GAVIN WOOD
FOUNDER, ETHEREUM & PARITY
GAVIN@PARITY.IO

ABSTRACT. The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, with Bitcoin being one of the most notable ones. Each such project can be seen as a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state.

Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.



L'ascesa delle Alternative Virtual Machines (AltVMs)

L'ascesa delle Alternative Virtual Machines (AltVMs)

L'ascesa delle Alternative Virtual Machines (AltVMs)

L'ascesa delle Alternative Virtual Machines (AltVMs)

- La **EVM** continuerà a dominare o è arrivato il momento di far emergere un nuovo leader?

- Le altVM possono **coesistere** con Ethereum oppure prenderanno il sopravvento?

Critiche ai linguaggi blockchain esistenti

1. Rappresentazione indiretta degli **asset**

Critiche ai linguaggi blockchain esistenti

1. Rappresentazione indiretta degli **asset**

Codificare utilizzando un numero intero
→ ma un *numero intero* **non equivale ad un asset**.

```
mapping(address => uint) private balance;
```

Critiche ai linguaggi blockchain esistenti

2. Il **controllo della scarsità** di un asset non è integrato nel linguaggio

Critiche ai linguaggi blockchain esistenti

3. Access control non flessibile

Critiche ai linguaggi blockchain esistenti

1. Rappresentazione indiretta degli **asset**

2. Il **controllo della scarsità** di un asset non è integrato nel linguaggio

3. Access control non flessibile

Move

*Rappresentazione di **transizioni di stato** che consente di codificare la proprietà delle **risorse digitali** in un sistema open source*

Move

*Rappresentazione di **transizioni di stato** che consente di codificare la proprietà delle **risorse digitali** in un sistema open source*

First-class Resources

First-class Resources

Fornisce la possibilità di definire tipi di *risorse personalizzate* con una semantica ispirata alla **logica lineare:**

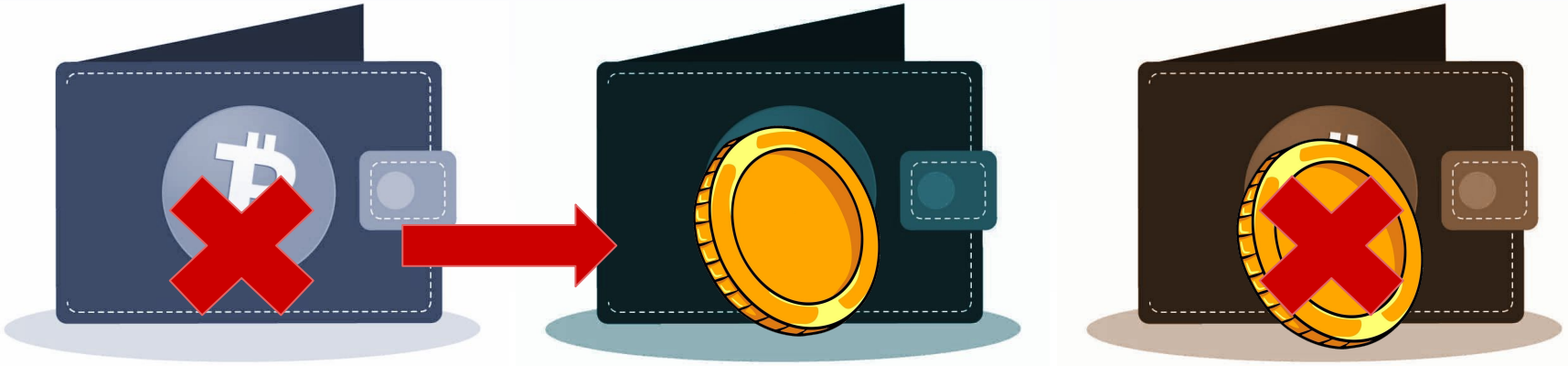
- una risorsa non può mai essere copiata o scartata implicitamente
- solo spostata tra le locazioni di memoria del programma.

Move

First-class Resources

Fornisce la possibilità di definire tipi di *risorse personalizzate* con una semantica ispirata alla **logica lineare:**

- una risorsa non può mai essere copiata o scartata implicitamente
- solo spostata tra le locazioni di memoria del programma.



Move

First-class Resources

I programmatori di Move possono **proteggere l'accesso** alle operazioni critiche sulle risorse tramite i

- ***Moduli***: contengono *tipi di risorse* e *procedure* che codificano le regole per le risorse.

Move

First-class Resources

Solidity

Address	Ether Balance	Data
0x2	3.4	`contract Bank` `mapping (address => uint) credit;`

Move

Blackshear, Sam, et al. "Resources: A safe language abstraction for money." arXiv preprint https://arxiv.org/abs/2004.05106 (2020).

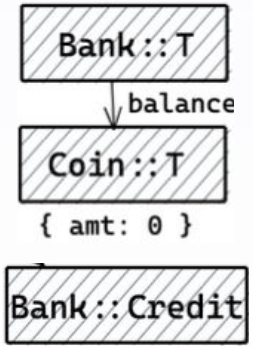
Move

First-class Resources

Solidity

Address	Ether Balance	Data
0x2	3.4	`contract Bank` `mapping (address => uint) credit;`

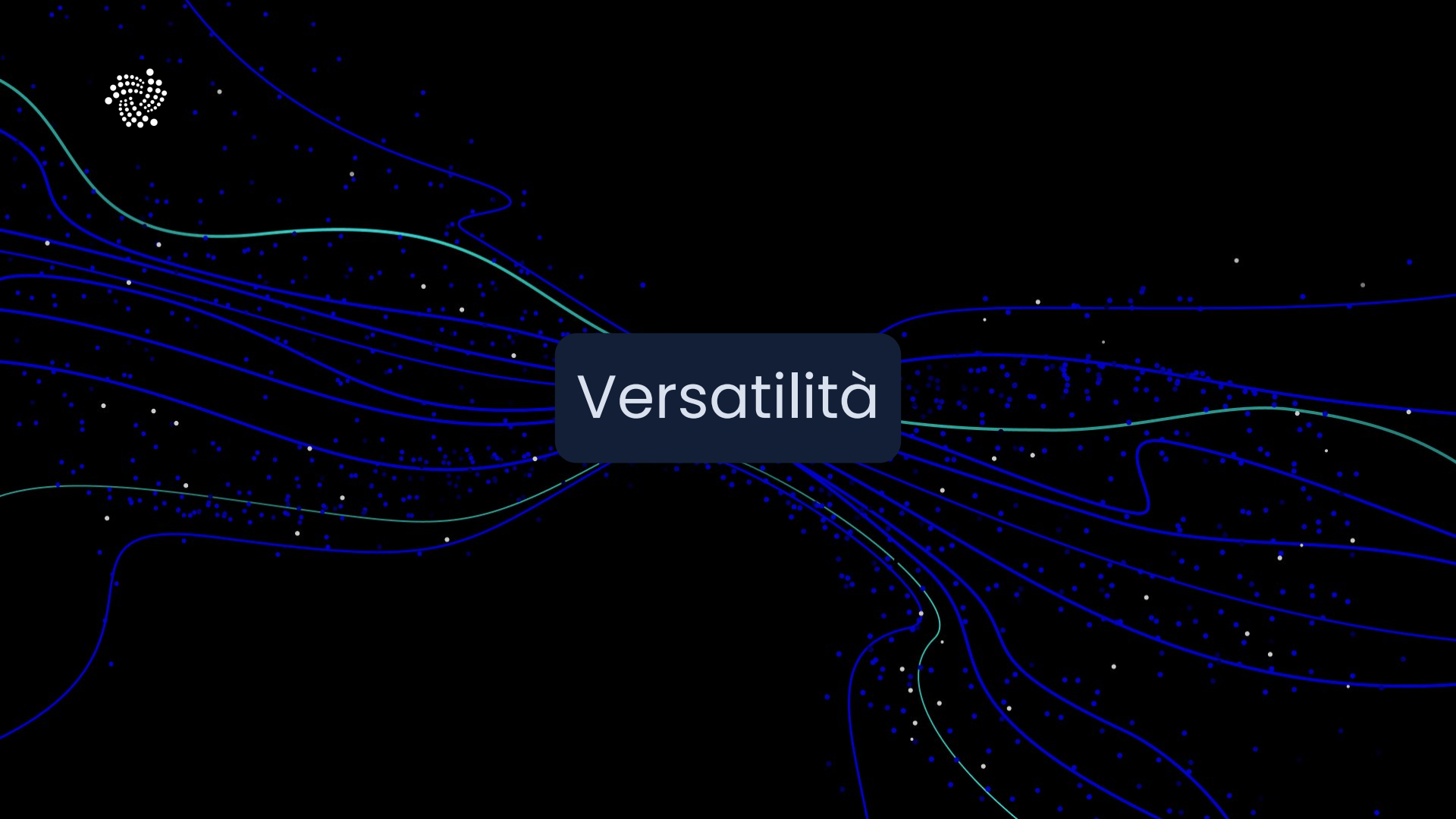
Move




```
module Bank
use 0x0::Coin;
resource T { balance: Coin::T }
resource Credit { amt: u64, bank: address }
```

Blackshear, Sam, et al. "Resources: A safe language abstraction for money." arXiv preprint https://arxiv.org/abs/2004.05106 (2020).


Risorse


Versatilità

Move

MoveVM→ Macchina Virtuale facilmente estensibile e blockchain-agnostic

Versatilità

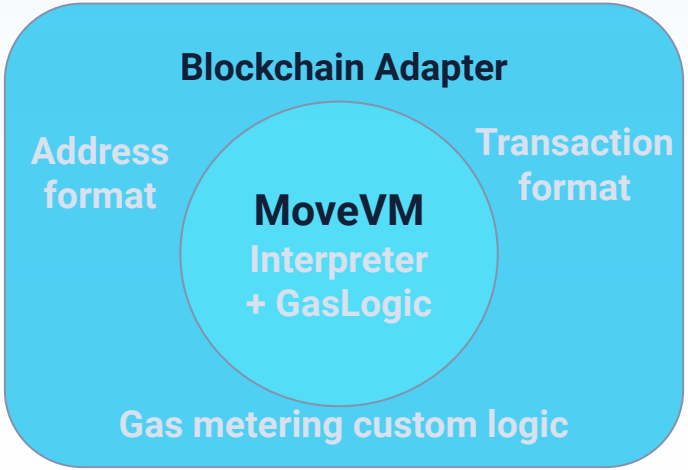
Move

Versatilità

MoveVM
Interpreter
+ GasLogic

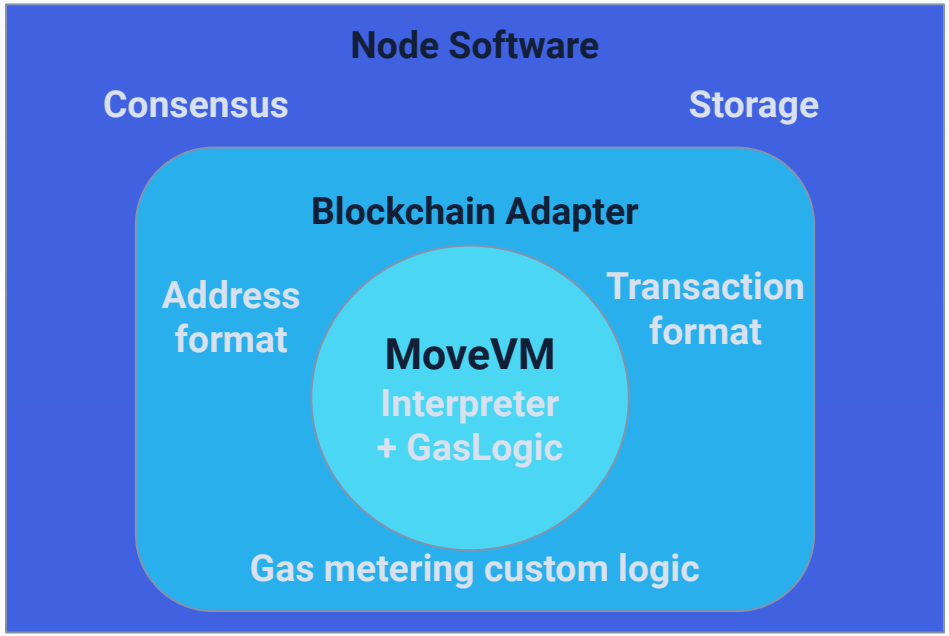
Move

Versatilità



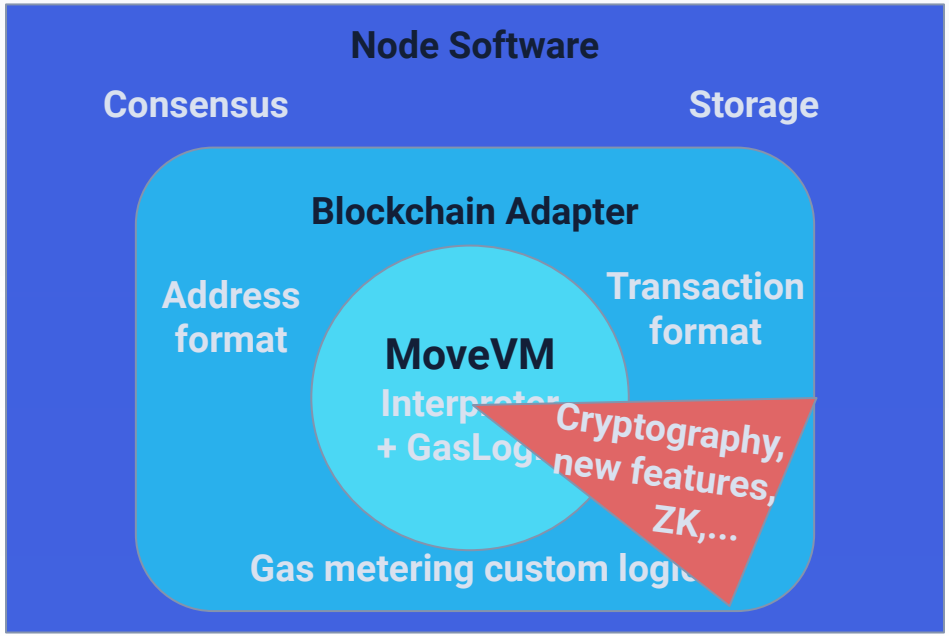
Move

Versatilità



Move

Versatilità



Move

Versatilità

“Move Flavors”: le due scuole di pensiero

Move

Versatilità

"Move Flavors": le due scuole di pensiero

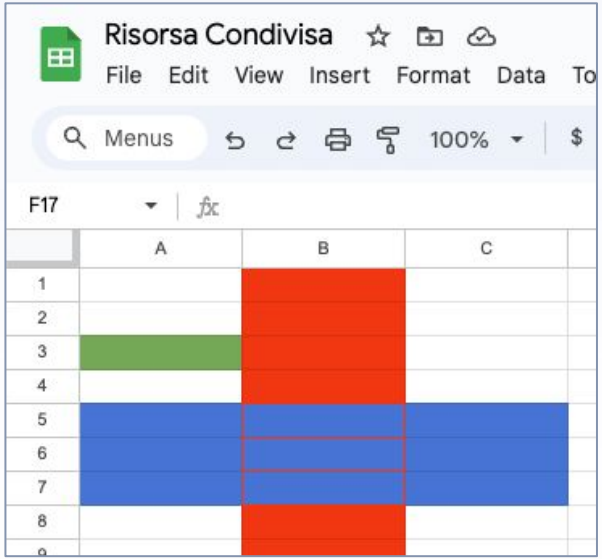
Memoria unificata → Account-based Ledger
(EVM, WASM, ISC, Aptos, Diem, etc.)

Memoria partizionata → Object-based Ledger (UTXO)
(Sui Move, Cardano, Radix, IOTA Stardust, etc.)

Come accedere ad una risorsa condivisa?

Scenario

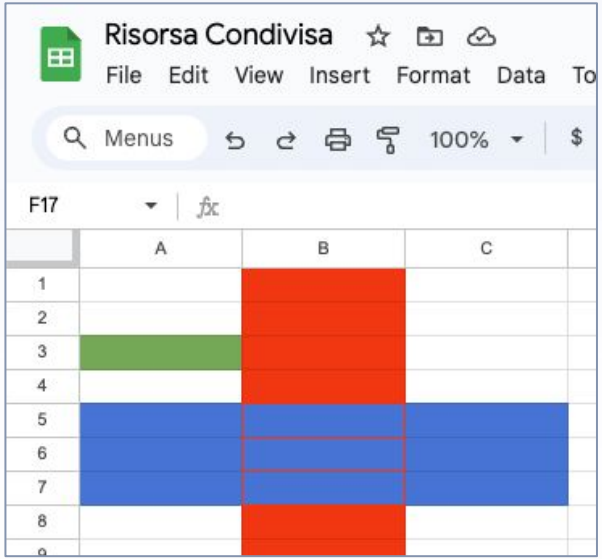
- Tutti vogliono modificare lo stesso foglio Excel
- Una persona ha bisogno di 1 minuto per aggiornare una cella



Come accedere ad una risorsa condivisa?

Scenario

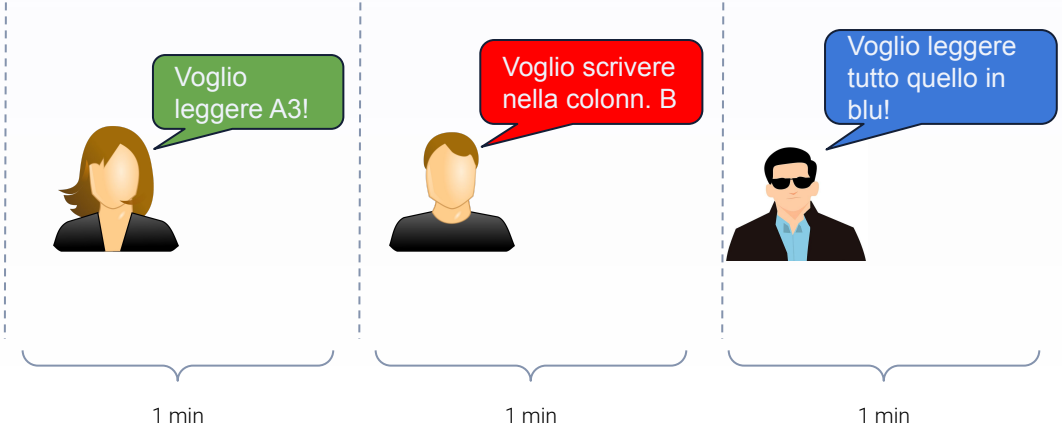
- Tutti vogliono modificare lo stesso foglio Excel
- Una persona ha bisogno di 1 minuto per aggiornare una cella

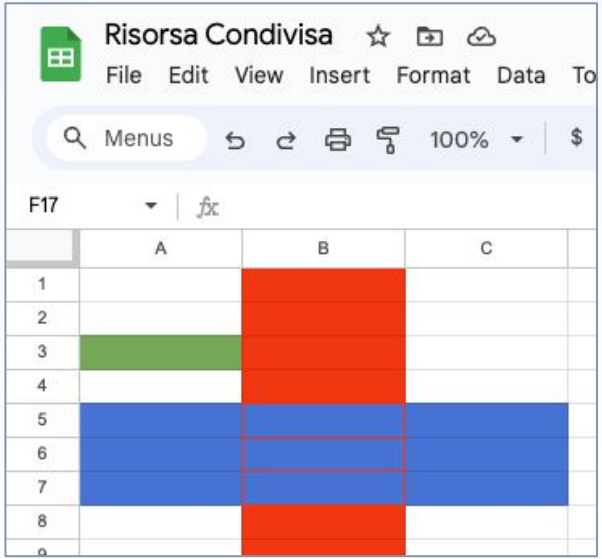


Come accedere ad una risorsa condivisa?

Memoria unificata

- Una persona alla volta può aprire il foglio, apportare le modifiche e poi salvarlo.
- Ci vogliono **1 + 1 + 1 = 3 minuti** prima che tutti finiscano.

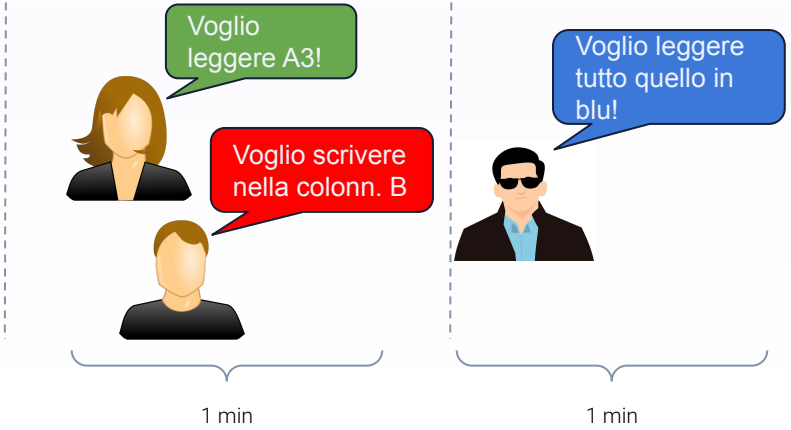


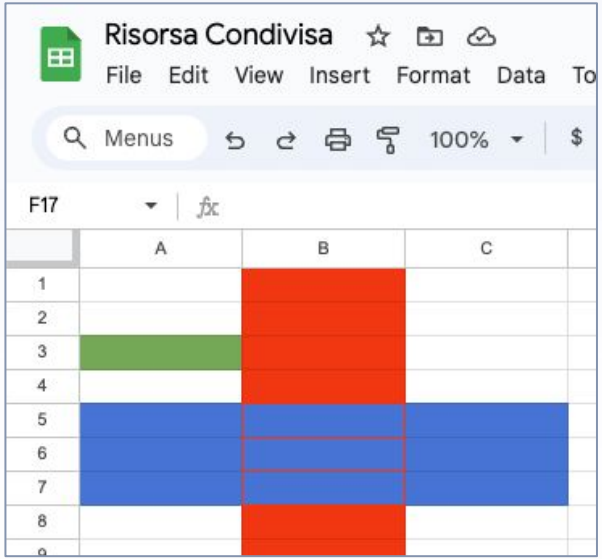


Come accedere ad una risorsa condivisa?

Memoria partizionata

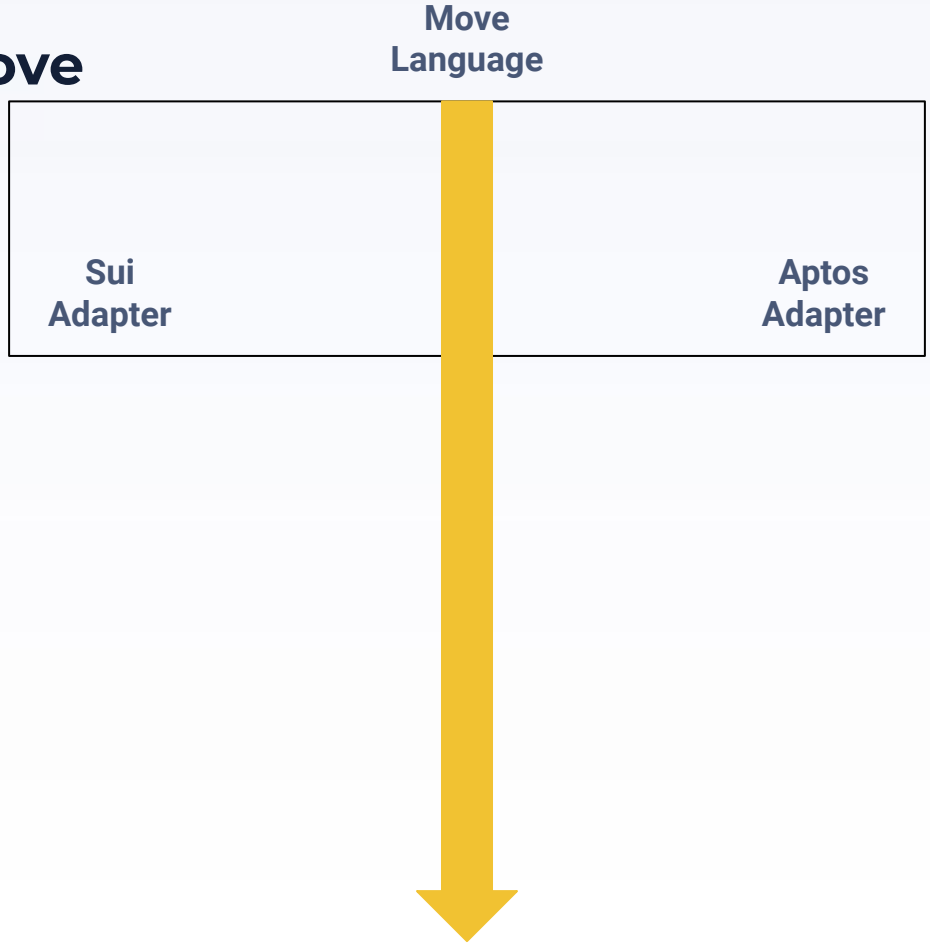
- Dichiara le celle che modificherai: se non sono in uso, vai avanti e modificale!
- Ci vogliono **1 + 1 = 2 minuti** prima che tutti abbiano finito.





Storia di Move

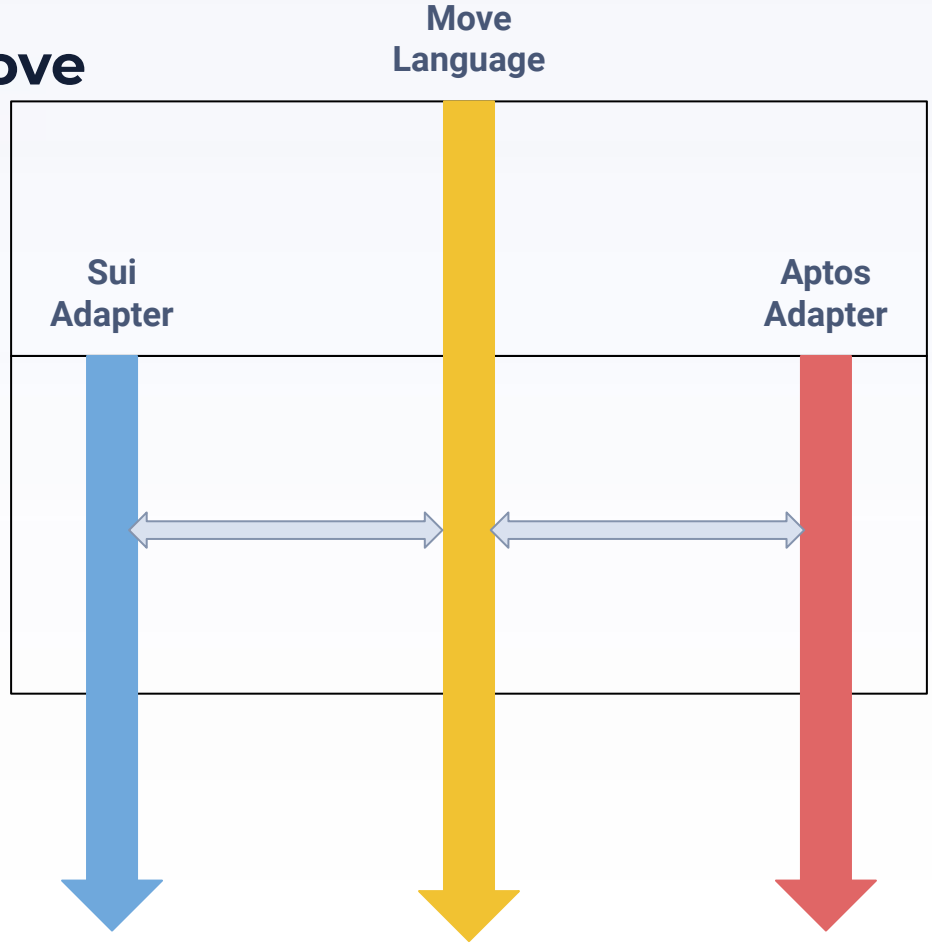
Early Move
Libra/Diem
2018-2021



Storia di Move

Early Move
Libra/Diem
2018-2021

Move Adapters
Sui & Aptos
2022-2023

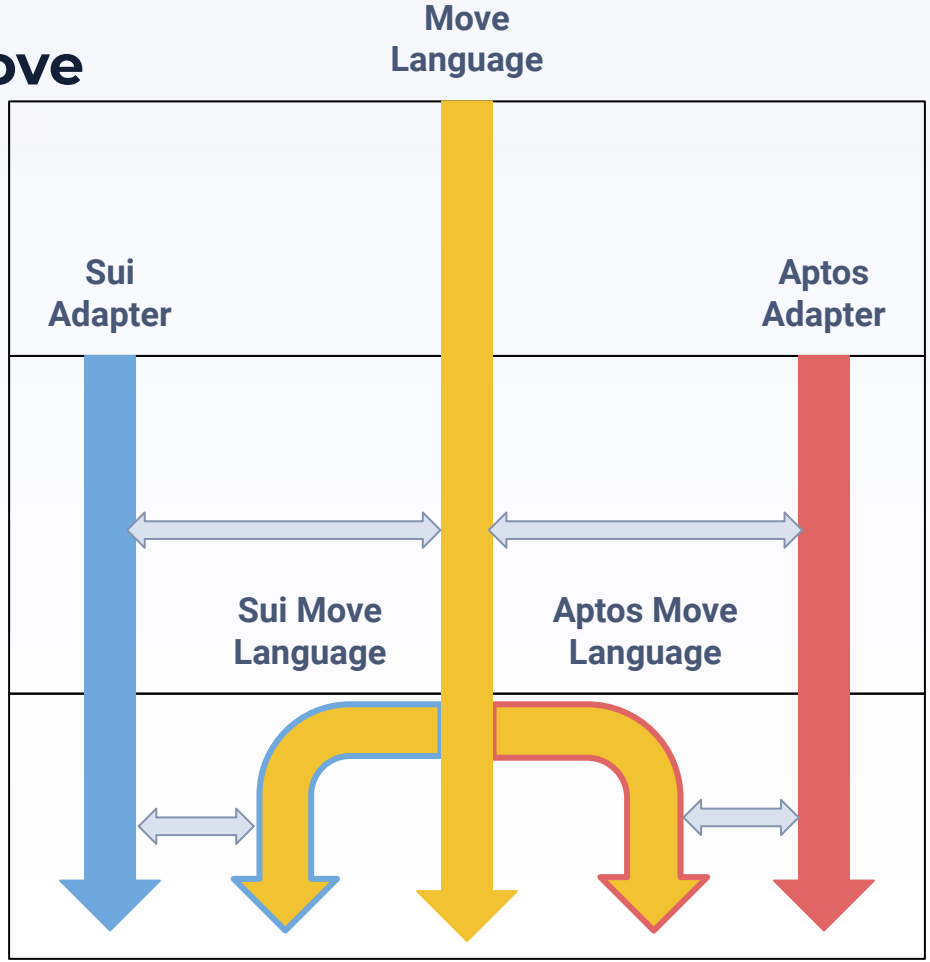


Storia di Move

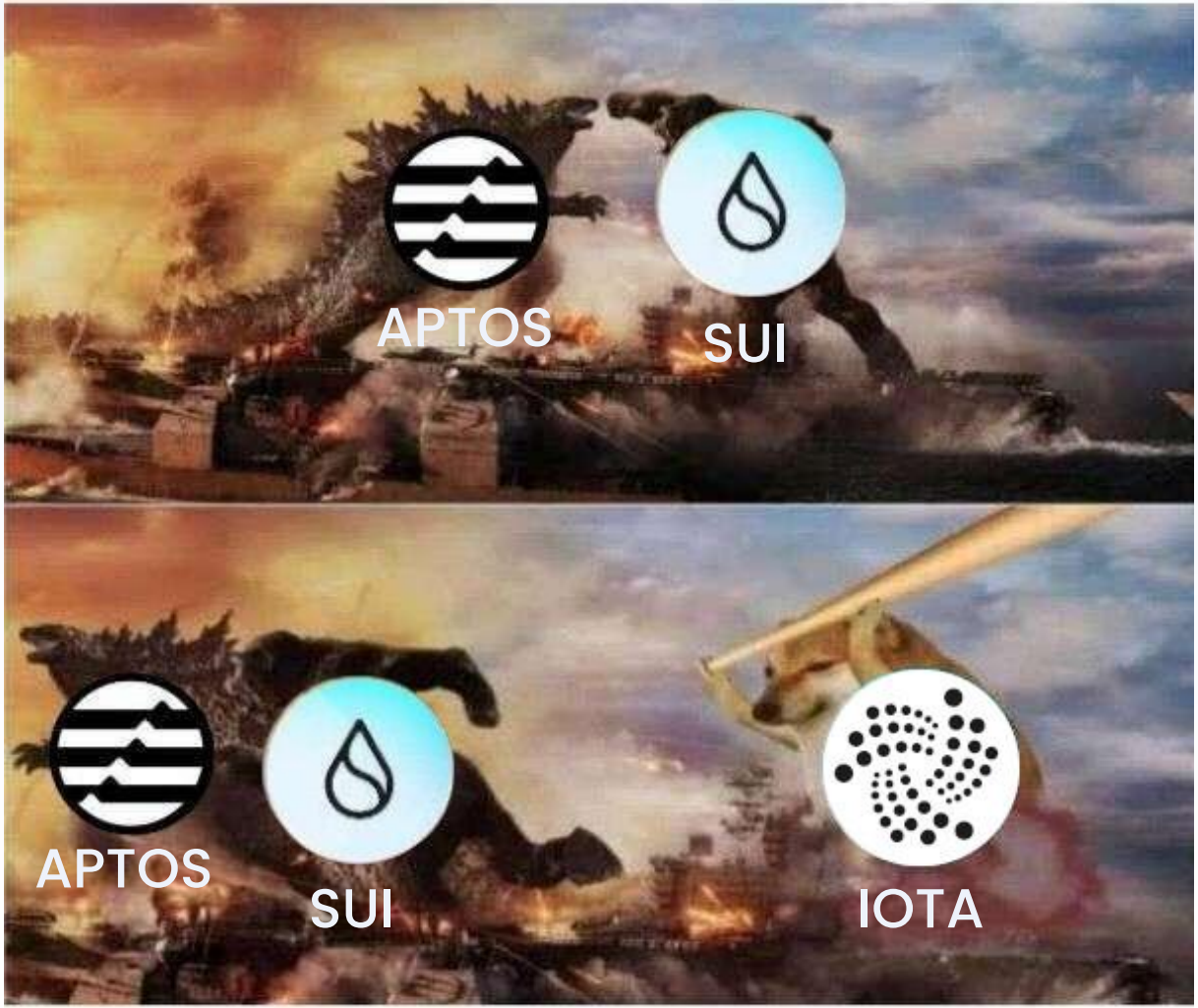
Early Move
Libra/Diem
2018-2021

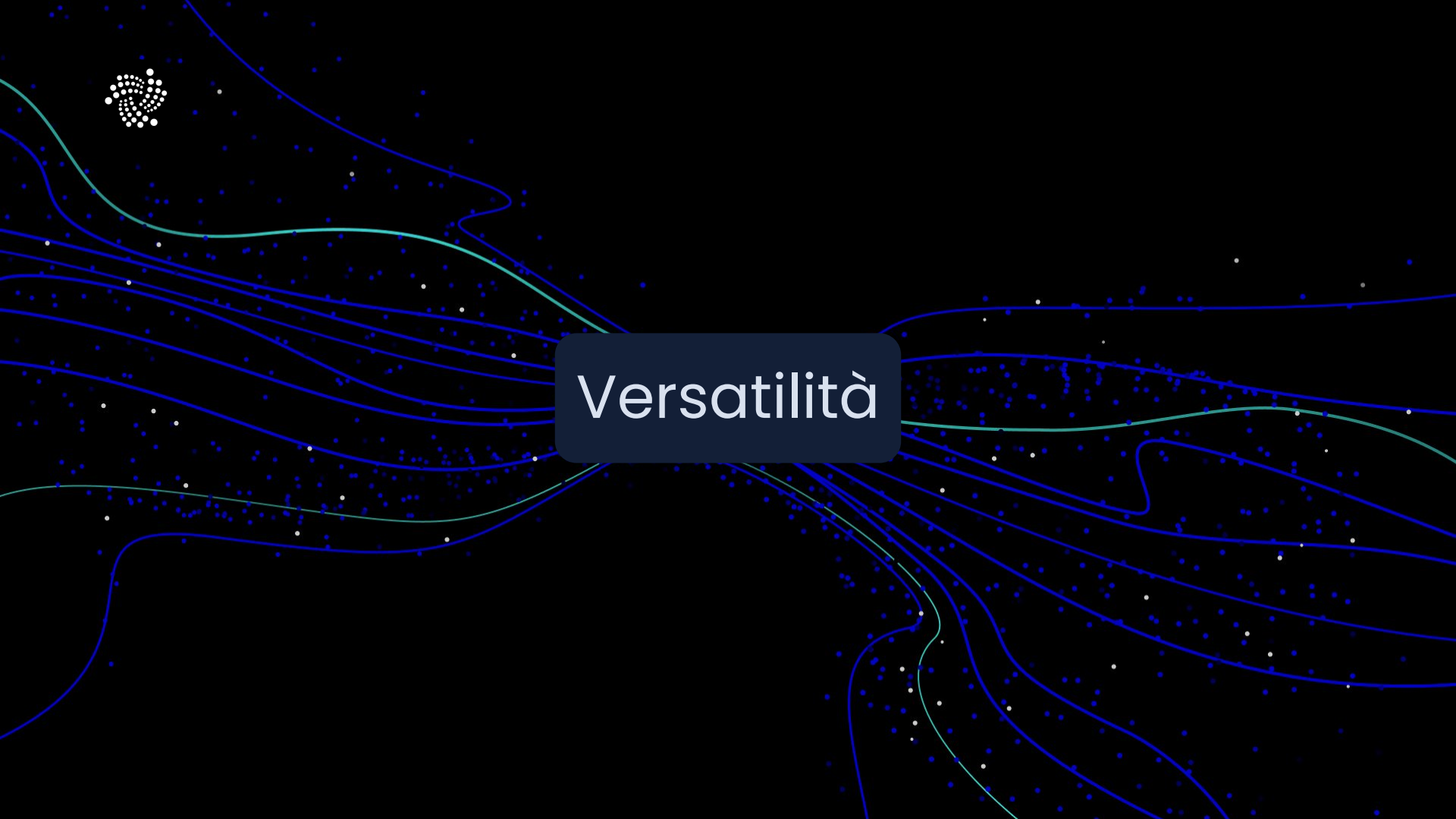
Move Adapters
Sui & Aptos
2022-2023

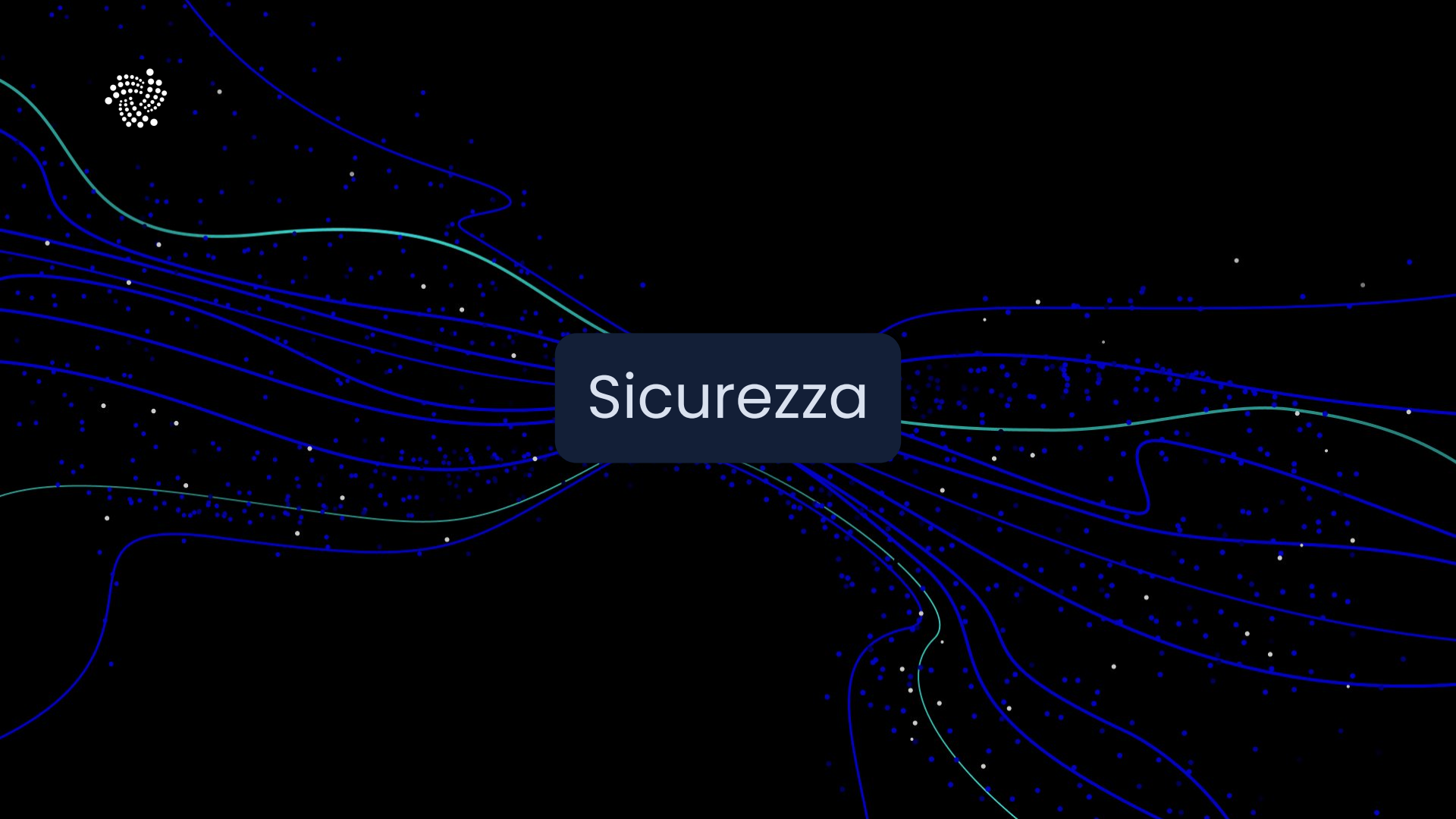
Move 2024
Sui & Aptos Forks




APTOS
VS
sui


APTOS
SUI
APTOS
SUI
IOTA


Versatilità


Sicurezza

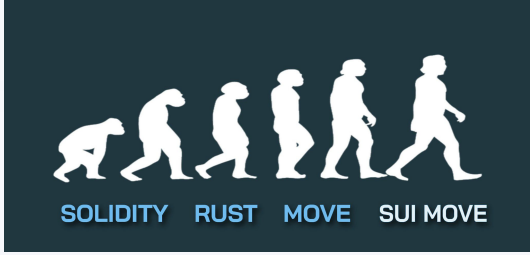
Move

“Come possiamo scrivere codice che siamo sicuri al 100% che sarà sicuro, perché sappiamo che dovrà gestire denaro?”

Sicurezza

Move

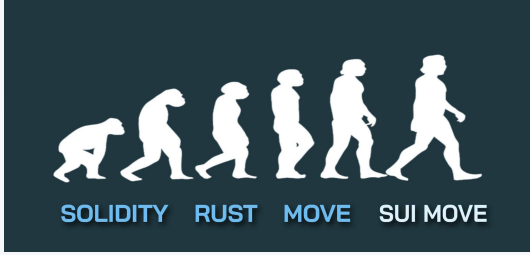
Sicurezza



- Eredita i concetti di **sicurezza della memoria e dei tipi** da *Rust*
    - Il compilatore cattura gli errori che normalmente non verrebbero rilevati in altri compilatori (es. Solidity)

Move

Sicurezza



- Eredita i concetti di **sicurezza della memoria e dei tipi** da *Rust*
  - Il compilatore cattura gli errori che normalmente non verrebbero rilevati in altri compilatori (es. Solidity)
- **Resource safety**
  - *I tipi semplici* come gli interi e address → possono essere copiati
    le risorse → possono essere solo **spostate**.
  - uso della **logica lineare** impedisce il *"double spending"* (spostare una risorsa due volte).

Sicurezza

- Access Control by default
    - Forzato dal linguaggio anche se il programmatore potrebbe dimenticare di implementarlo.

Move

Sicurezza

- Access Control by default
    - Forzato dal linguaggio anche se il programmatore potrebbe dimenticare di implementarlo.
- Mutabilità limitata
    - Ogni mutazione di un valore in Move avviene attraverso un ***“riferimento”*** come in *Rust*.
        - by-value → *value*
        - mutable → *&mut value*
        - read-only → *&value*

Passare un valore *by-value*

Passare un valore *by-value*

Passare un valore *by-value*

“Borrow” un valore *mutable ref (&mut)*

“Borrow” un valore *mutable ref (&mut)*

“Borrow” un valore *mutable ref (&mut)*

“Borrow” un valore *read-only ref (&)*

Sicurezza

Verifica doppia:

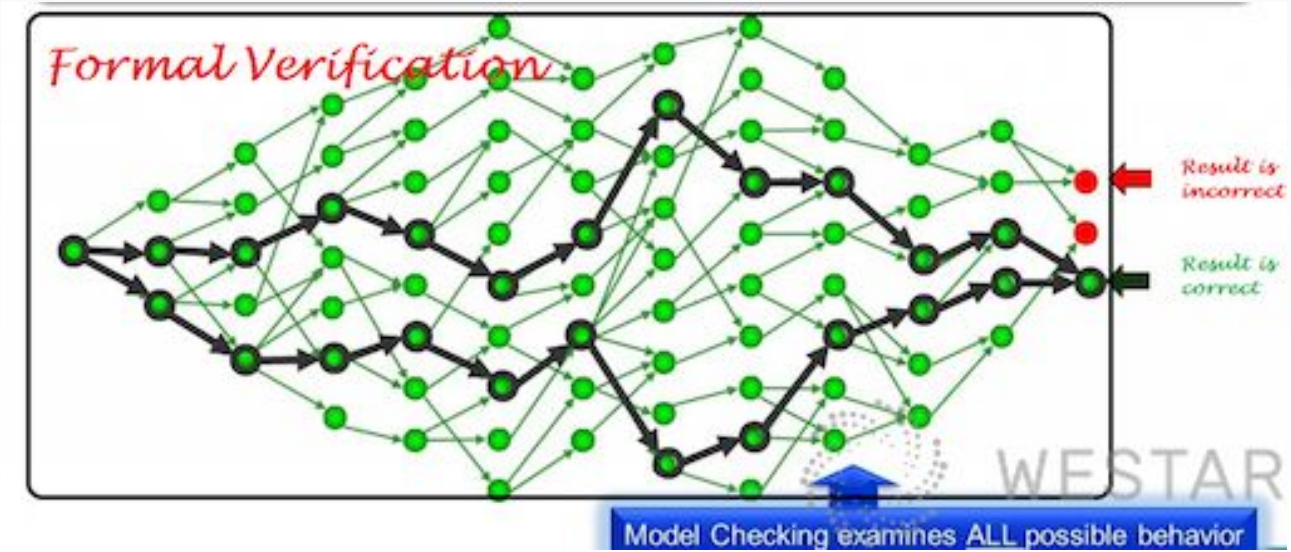
- il linguaggio di programmazione di alto livello
  - viene compilato tramite un compilatore che *verifica le proprietà di sicurezza*

- il linguaggio di programmazione di basso livello non tipizzato
  - esegue i controlli di sicurezza **in fase di esecuzione**

Move

Sicurezza

I programmi in Move possono essere **Formalmente Verificati**



Sicurezza

NO reentrancy.

Sicurezza

NO reentrancy.

Solidity

```
function withdraw() {
  uint amt = credit[msg.sender];
  msg.sender.transfer(amt);
  credit[msg.sender] = 0;
}
```

Move

Blackshear, Sam, et al. "Resources: A safe language abstraction for money." arXiv preprint https://arxiv.org/abs/2004.05106 (2020).

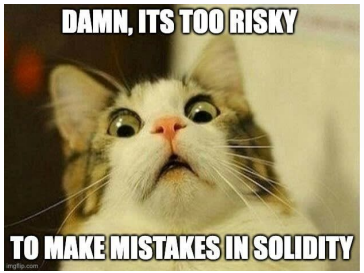
Move

Sicurezza

NO reentrancy.

Solidity

```
function withdraw() {
  uint amt = credit[msg.sender];
  msg.sender.transfer(amt);
  credit[msg.sender] = 0;
}
```



Move

```
fun withdraw(credit: Credit): Coin::T {
  Credit { amt, bank } = move credit;
  let t = borrow_global<T>(move bank);
  return Coin::withdraw(
    &mut t.balance, move amt
  );
}
```

Blackshear, Sam, et al. "Resources: A safe language abstraction for money." arXiv preprint https://arxiv.org/abs/2004.05106 (2020).

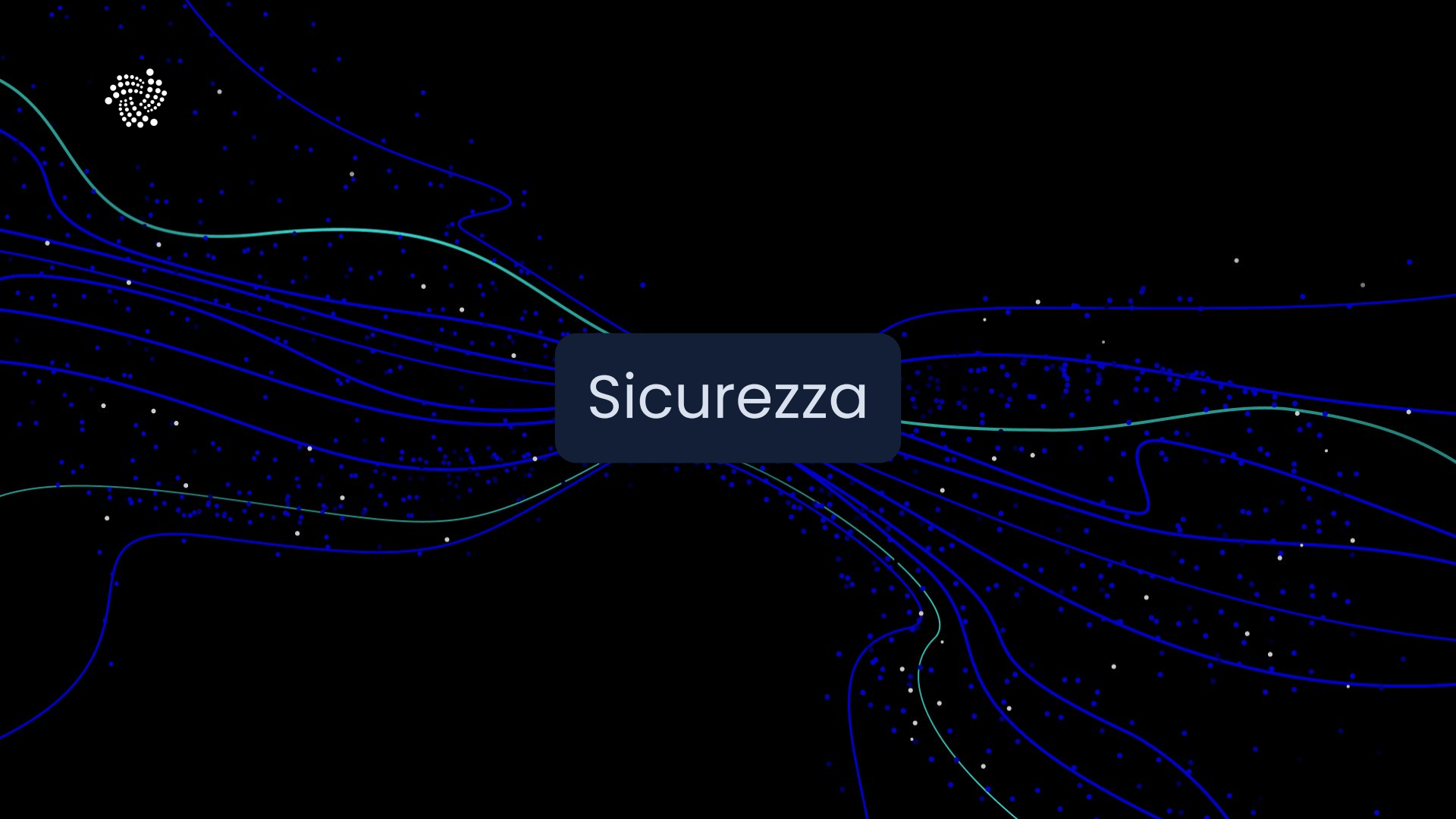
Move

Sicurezza

NO reentrancy.

Causa principale della reentrancy
→ *dispatch dinamico*:
all'interno di uno smart contract si ha una funzione la cui definizione non è nota in anticipo allo sviluppatore.

In Move ogni volta che si chiama una funzione il codice che viene chiamato è *staticamente noto (static dispatch)*.


Sicurezza

Risorse

Versatilità

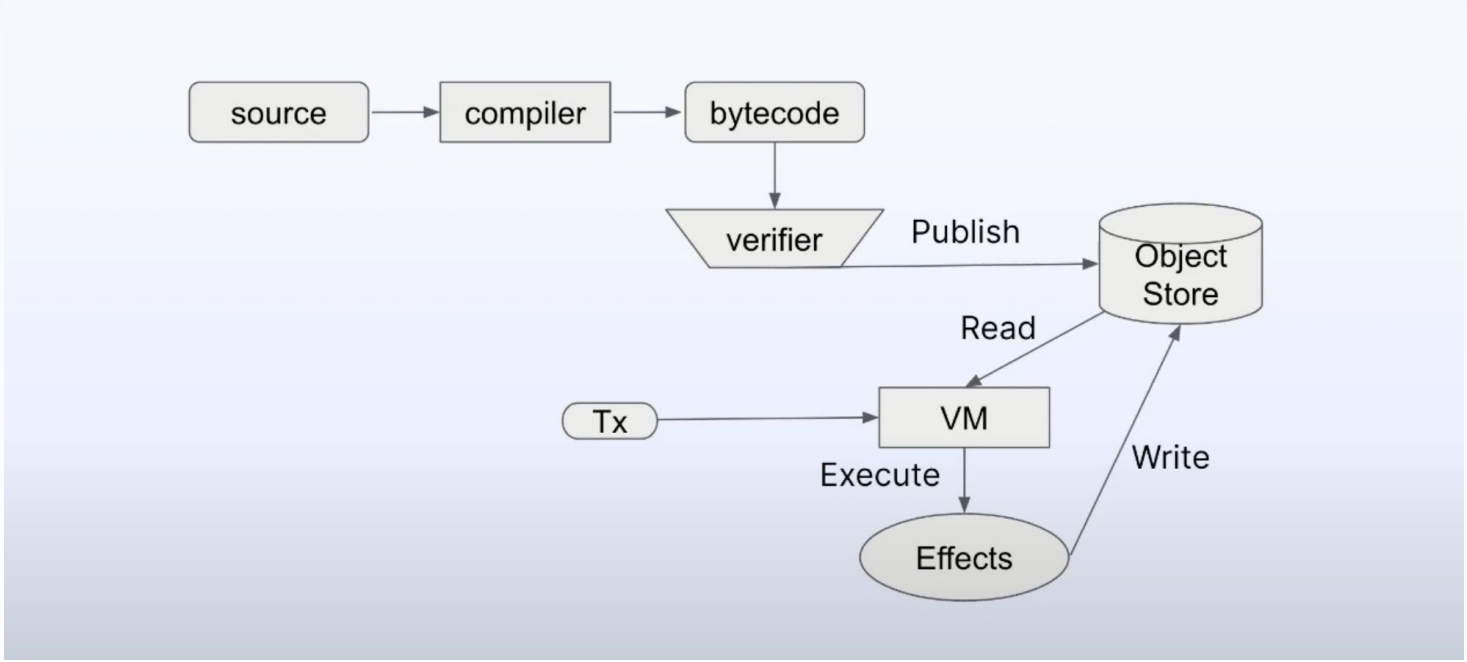
Sicurezza

Domande?

Mirko Zichichi
mirko.zichichi@iota.org

IOTA

Move compile/publish/run toolchain



1. Object Basics

- The first field of the **struct** must be the id of the object with type **UID**

Struct

```
struct Color {
    red: u8,
    green: u8,
    blue: u8,
}
```

Object

```
use sui::object::UID;

struct ColorObject has key {
    id: UID,
    red: u8,
    green: u8,
    blue: u8,
}
```

https://docs.sui.io/build/programming-with-objects/ch1-object-basics

2. Owned, Shared and Immutable Objects

- Objects in Sui can have different types of **ownership**, with three categories:
  - **Owned mutable** object -> is owned by an address/object
  - **Shared mutable** object -> anyone can use it in a transaction
  - **Immutable** object -> an object that can't be mutated, transferred or deleted.

- In other blockchains, ***every object is shared***
  - In Sui Move programmers have the choice to implement a particular use-case using **shared objects, owned objects, or a combination**.

- In Sui, a transaction that touches a shared object needs to pass through the consensus mechanism. Whilst, a transaction that touches only owned objects does not need it.

https://docs.sui.io/learn/objects

3. Programmable Transaction Blocks

- The **inputs value** of a PTB is value is a vector of arguments, either *objects* or *pure values*
- The **commands value** of a PTB is a vector of commands using *inputs* or *results* to execute code
    - *TransferObjects* sends (one or more) objects to a specified address
    - *SplitCoins* splits off (one or more) coins from a single coin. It can be any sui::coin::Coin<_>
    - *MergeCoins* merges (one or more) coins into a single coin
    - *MakeMoveVec* creates a vector of Move values
    - ***MoveCall*** invokes either an *entry* or a *public* Move function in a published package.
    - *Publish* creates a new package and calls the init function of each module in the package.
    - *Upgrade* upgrades an existing package.
- The **result values** is a vector of values tha can be produced by each command; the type of the value can be any arbitrary Move type, not limited to objects or pure values.
- A PTB can perform up to 1,024 unique operations in a single execution.

https://docs.sui.io/concepts/transactions/prog-txn-blocks

3. Programmable Transaction Blocks

```
$ sui-ctf client ptb \
--move-call 0xd95b4510206e13fbe9413bc61183ac3b8375c8971adc54c81eeb9c96d61b5ff1::pkg::func
"<0xd95b4510206e13fbe9413bc61183ac3b8375c8971adc54c81eeb9c96d61b5ff1::pkg1::TYPE1,0xd95b451
0206e13fbe9413bc61183ac3b8375c8971adc54c81eeb9c96d61b5ff1::pkg2::TYPE2>"
@0x0b72fb4d8106699c773bf58fd0a49ffe3a08bdd58f245946d160ed5463f7ba47 99 true \
--assign result_variable \
--move-call sui::tx_context::sender \
--assign sender \
--transfer-objects "[result_variable.2]" sender \
--move-call 0xd95b4510206e13fbe9413bc61183ac3b8375c8971adc54c81eeb9c96d61b5ff1::pkg::func2
"<0xd95b4510206e13fbe9413bc61183ac3b8375c8971adc54c81eeb9c96d61b5ff1::pkg1::TYPE1"
@0x0b72fb4d8106699c773bf58fd0a49ffe3a08bdd58f245946d160ed5463f7ba47 result_variable.0 \
--gas-budget 50000000
```

https://docs.sui.io/references/cli/ptb

Move “Resource”

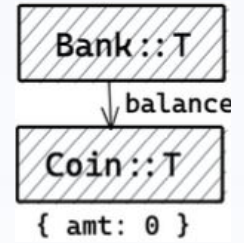
Address	Ether Balance
0x2	3.4

Solidity

```
contract Bank
mapping (address => uint) credit;

function deposit() payable {
  amt =
    credit[msg.sender] + msg.value
  credit[msg.sender] = amt
}

function withdraw() {
  uint amt = credit[msg.sender];
  msg.sender.transfer(amt);
  credit[msg.sender] = 0;
}
```

Move

```
module Bank
use 0x0::Coin;
resource T { balance: Coin::T }
resource Credit { amt: u64, bank: address }

fun deposit(
  coin: Coin::T,
  bank: address
): Credit {
  let amt = Coin::value(&coin);
  let t = borrow_global<T>(copy bank);
  Coin::deposit(&mut t.balance, move coin);
  return Credit {
    amt: move amt, bank: move bank
  };
}

fun withdraw(credit: Credit): Coin::T {
  Credit { amt, bank } = move credit;
  let t = borrow_global<T>(move bank);
  return Coin::withdraw(
    &mut t.balance, move amt
  );
}
```



Blackshear, Sam, et al. "Resources: A safe language abstraction for money." arXiv preprint https://arxiv.org/abs/2004.05106 (2020).