

Build on Sui
Weekend Move Workshop
in Tokyo // Day 1

Powered by



自己紹介

2021年にCryptoGamesに入社し、ブロックチェーンエンジニアとして活動。
AstarGames（現MOCHIRON社）、
SuperTeam Japanを経て、現在は**株式会社シーエーシーのブロックチェーン推進グループ**に所属。

シーエーシーは**2017年**からのブロックチェーン事業を開始し、**Corda**を利用したサービスを手がけ、**Sui、Solana、Avalanche** 3基盤の技術選定を行なっています。



KOUKAとは 導入のメリット 主な機能 活用シーン ユーザーの声 ご利用開始までの流れ

ブロックチェーン技術を活用した つながり可視化ツール KOUKA

ブロックチェーン技術によるトークンエコノミーを構築し、社員の見えない価値を可視化。「つながりの可視化」により、社員のモチベーション向上はもちろんのこと、社内・外のコミュニケーション拡大が実現できるサービスです。

資料ダウンロード >

お問合せはこちら >

A diagram illustrating the KOUKA tool's concept. It shows a central figure (a man in a suit) standing on a blue circular platform. Around him are several other people, each on their own platform, connected by arrows. Some people are holding gifts, symbolizing value exchange and community building.

<https://www.cac.co.jp/product/finance-dx/blockchain/>

事前準備

以下の4パターンで進めていきます。

- ローカルにSuiが入っている
- Dockerを準備
- BUTASANのGithubCodeを利用
<https://zenn.dev/bububutasan/articles/65059536febaa1>
- BitsLabを利用
<https://ide.bitslab.xyz/>

事前準備

1 クライアントの作成

Suiとやりとりするためのクライアントを作成します。

sui client -y

これでアドレスを作成できます。

```
root@7f9d1f765bcd:/workspace#  
root@7f9d1f765bcd:/workspace# sui client -y  
Creating config file ["/root/.sui/sui_config/client.yaml"] with default (devnet) Full node server and ed25  
519 key scheme.Generated new keypair and alias for address with scheme "ed25519" [serene-spinel: 0xe604225  
1456015481158b25ac03f8f5c15cc4d160baf436dad1c705a43978e7c]  
Secret Recovery Phrase : [REDACTED]  
Client for interacting with the Sui network  
  
Usage: sui client [OPTIONS] [COMMAND]  
  
Commands:  
  active-address          Default address used for commands when none specified
```


事前準備

2 アクティブアドレスの確認

以下のコマンドで**アクティブアドレスが確認**できます。

```
sui client active-address
```

手持ちのウォレットからテストトークンを送りましょう。(0.02Suiなど少量でOK)

```
-y, --yes
-h, --help          Print help
root@7f9d1f765bcd:/workspace# sui client active-address
[warning] Client/Server api version mismatch, client api version : 1.56.0,
8.2
0xe6042251456015481158b25ac03f8f5c15cc4d160baf436dad1c705a43978e7c
root@7f9d1f765bcd:/workspace#
root@7f9d1f765bcd:/workspace#
root@7f9d1f765bcd:/workspace#
```

事前準備

3 テストネットの指定

以下のコマンドで**テストネットに設定**し、結果を確認しましょう。

```
sui client switch --env testnet
```

```
sui client envs
```

```
root@7f9d1f765bcd:/workspace#  
root@7f9d1f765bcd:/workspace# sui client switch --env testnet  
[warning] Client/Server api version mismatch, client api version : 1.56.0  
8.2  
Active environment switched to [testnet]  
root@7f9d1f765bcd:/workspace# sui client envs  
[warning] Client/Server api version mismatch, client api version : 1.56.0  
8.2
```

alias	url	active
testnet	https://fullnode.testnet.sui.io:443	*

事前準備

4 残高の確認

以下のコマンドで**残高を確認**しましょう。

sui client gas

```
[warning] Client/Server api version mismatch, client api version : 1.56.0, server api version : 1.58.2
Active environment switched to [testnet]
root@7f9d1f765bcd:/workspace# sui client envs
[warning] Client/Server api version mismatch, client api version : 1.56.0, server api version : 1.58.2
```

alias	url	active
testnet	https://fullnode.testnet.sui.io:443	*

```
root@7f9d1f765bcd:/workspace# sui client gas
[warning] Client/Server api version mismatch, client api version : 1.56.0, server api version : 1.58.2
```

gasCoinId	mistBalance (MIST)	suiBalance (SUI)
0xd3656eb90a946c892f16fb32e3559e21cc538118a3d39235dfbecb4535d9cc4b	20000000	0.02

```
root@7f9d1f765bcd:/workspace#
```

Moveとは？

安全かつ柔軟

Moveは、Sui上で安全かつ柔軟に動作するスマートコントラクトのために設計されています。

リソース指向

資産を安全に管理し、リエントランシーのような一般的な脆弱性を防ぎます。

Suiでの利用

Moveはオンチェーンのオブジェクトを制御し、効率的かつ安全なデータ管理を実現します。

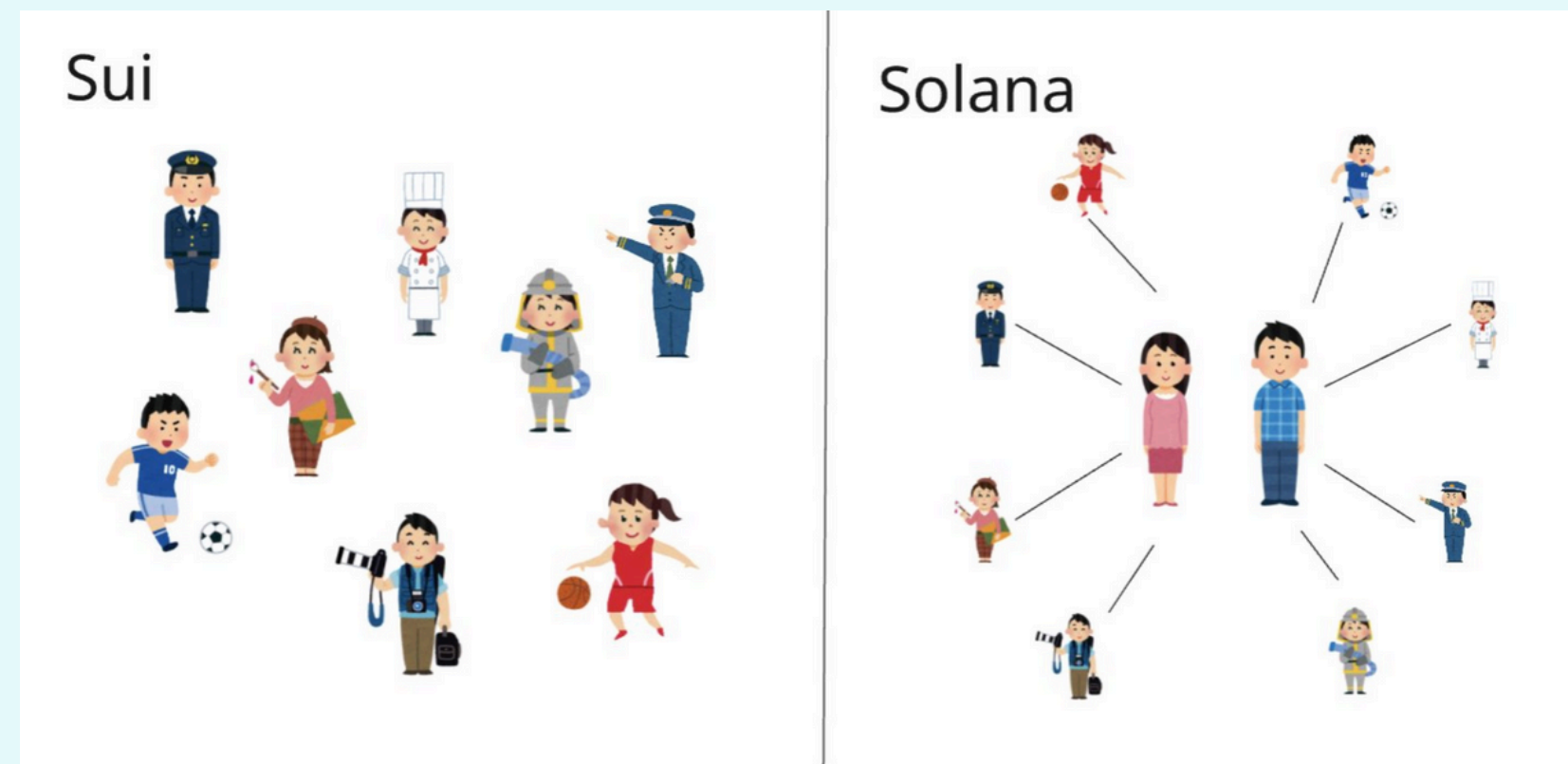
Suiの本質は？ 型を重視した設計

Suiは**型**を非常に重視します。

コックさんの格好をしていれば、コックさんだと確定します。

一方、例えば、**Solana**の場合は、**実行時の挙動**によって変わります。

厨房で料理しているという挙動を持ってコックさんのような人だとわかります。



型からの派生 ①能力 (Ability)

Moveには「Key」「Copy」「Drop」「Store」の4つの能力があります。
型を重視するため、客観的にこのマークがついているかどうかで判断します。

Sui Move 4つのAbility

デジタル資産を安全に扱うMoveの型システム

key 🔑 一意なIDを持つ、 Moveオブジェクトとして使える	store 💾 オブジェクト内に格納 できる
copy 📄 値を複製できる	drop 🗑️ 値を破棄できる

全てなし → 特殊なHot Potato型

作成されたら同一Tx内で必ず消費しないといけない！手に持ったまま放置できないから熱いポテト🍟フラッシュローンで大活躍

#SuiDevTips

💡 Abilityを組み合わせ
て多様なオブジェクトを表現する

key + store → NFTトークン
keyのみ → 共有Pool, 設定オブジェクト
copy + drop + store → データ型
dropのみ → Witness

```
public struct NFT has key, store {  
  id: UID  
}
```

https://x.com/0x_onigiri/status/1964982038962081834

型からの派生 ①能力 (Ability)

能力の例として、カービィのコピー能力とハンターハンターの念能力を考えてみましょう。Suiはどちらだと思えますか。前のページを元に考えてみましょう。



https://www.reddit.com/r/Kirby/comments/ufs0jw/i_made_kirby_and_the_forgotten_land_copy/?tl=ja



https://jumpcs.shueisha.co.jp/shop/g/g4530430522989/?srsltid=AfmBOop7lqDSN_8CqP9zy98iL_up-6kgpKNotRGuAWsJPsy16rClqnpp

型からの派生 ①能力 (Ability) 1 Key

私の何の変哲もないシャーペンは全く同じものが存在しない**唯一のもの**です。

私の千円札も他の千円札と価値は同じですが、**唯一のもの**です。

このような唯一のものに対し、Suiは「**key**」というキーワードをつけます。

世界で唯一の**グローバルID**がつきます。



型からの派生 ①能力 (Ability) 2 Copy

デジタルの世界ではコピペなどは頻繁に行われますが、実際の世界では誰かにものを渡すときにコピーは行われません。

移動するだけです。

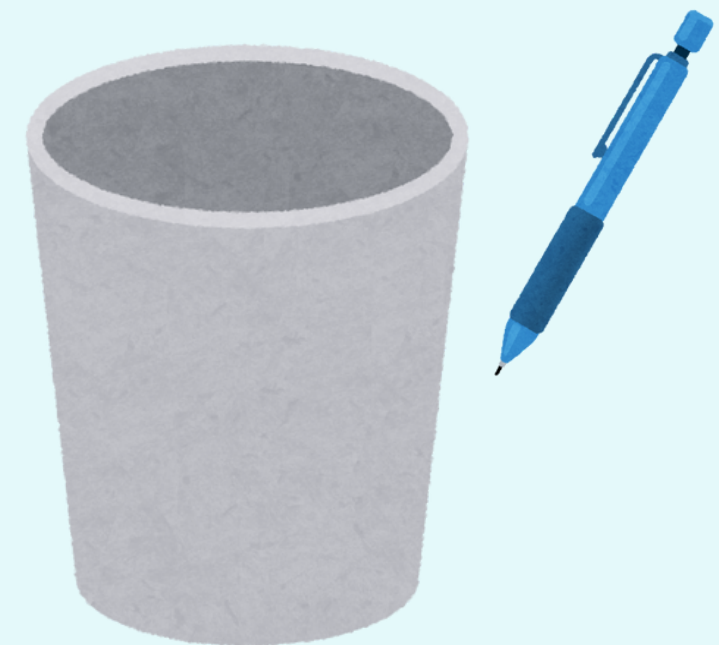
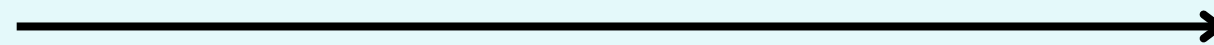
なので、原則コピーはできず、できるようにするには **「Copy」** をつけます。



型からの派生 ①能力 (Ability) 3 Drop

デジタルの世界ではデリート（削除）は簡単に行えますが、現実の世界では捨てたとしても、**存在自体は無くなりません**ね。

なので、原則は破棄ができず、できるようにするには **「drop」** をつけます。

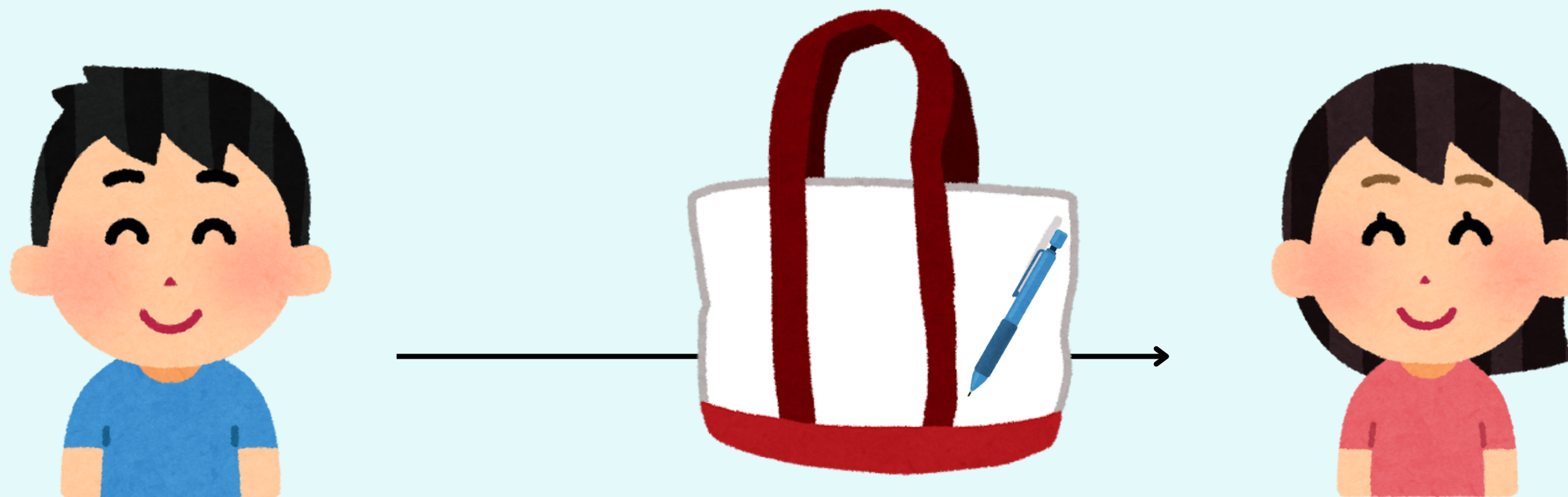


型からの派生 ①能力 (Ability) 4 Store

Moveは「移動」に焦点を当てた言語です。

もしもシャーペンがバッグに入っている（オブジェクトのフィールド）場合、バッグが移動すれば、**その中のシャーペンも移動**します。

そのため、格納できるかどうかで **「store」** をつけます。



補足 The DAO事件とは

再入（リエントランシー）の脆弱性について不正に引き出しを行おうとした事件です。



Photo by ekakinonakagawa

【完全保存版】The DAO事件を通じて、Re-entrancy攻撃やCEIパターンを学ぼう！

♡ 22



ユウキ
2023年7月15日 07:37

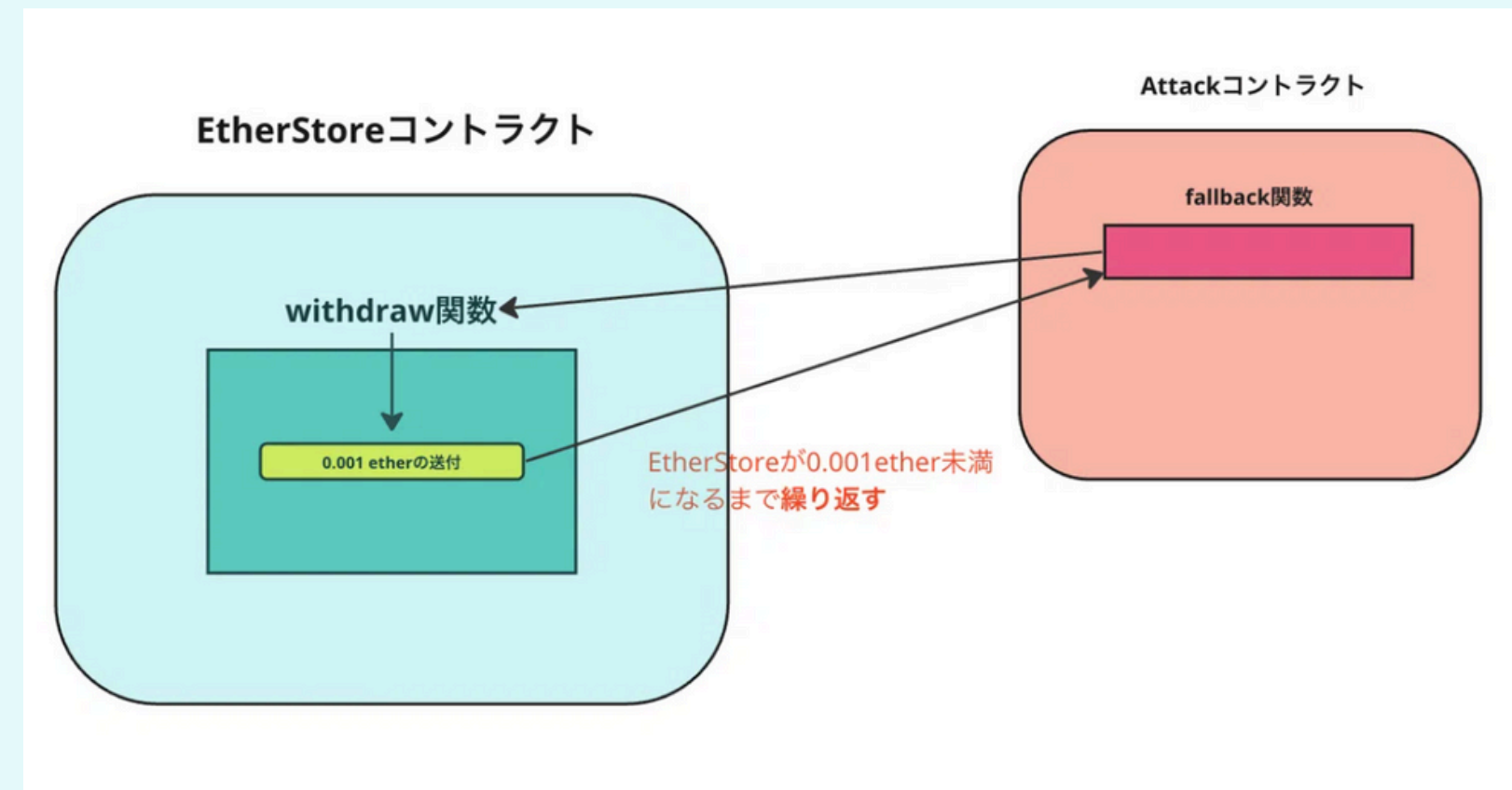
...

<https://note.com/standenglish/n/ncd433393aca3>

補足 The DAO事件とは

下のように、スマートコントラクトの脆弱性（順番が違うなど）によって攻撃を受けてしまった。

```
1  // SPDX-License-Identifier: MIT
2
3  pragma solidity ^0.8.0;
4
5  contract EtherStore {
6      mapping(address => uint) public balances;
7
8      function deposit() public payable {
9          balances[msg.sender] += msg.value;
10     }
11
12     function withdraw() public {
13         uint bal = balances[msg.sender];
14         require(bal > 0); ①残高確認
15
16         (bool sent, ) = msg.sender.call{value: bal}(""); ②送付
17         require(sent, "Failed to send Ether");
18
19         balances[msg.sender] = 0; ③値の反映
20     }
21
22     function getBalance() public view returns (uint) {
23         return address(this).balance;
24     }
25 }
```

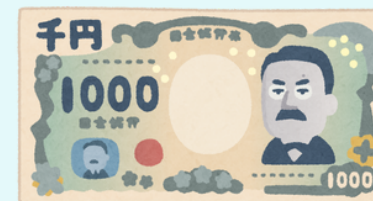


補足 リソース指向とリエントランシー

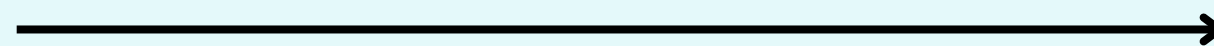
リソース指向であればこのようなケースは起こりえません。
例えば、コインにはそれぞれに一意的な**Key**がついており、**Copy不可**です。

EVMのように金額を数値としてあつかうのではなく、**物（オブジェクト）**として扱っているためです。

（他にもいろいろな理由でリエントランシーを防いでいますがここでは省略）

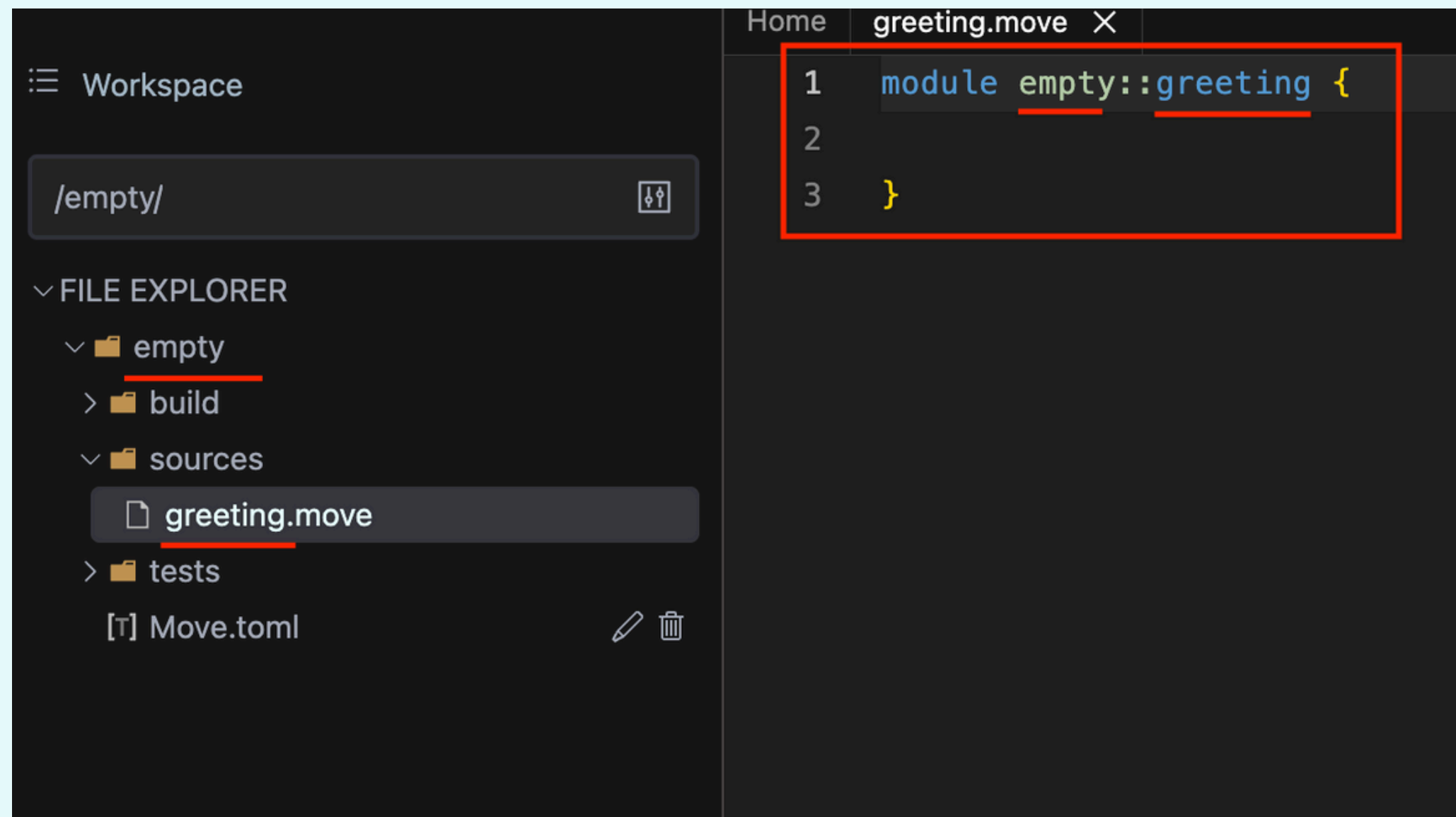


AE0119AE011973AE



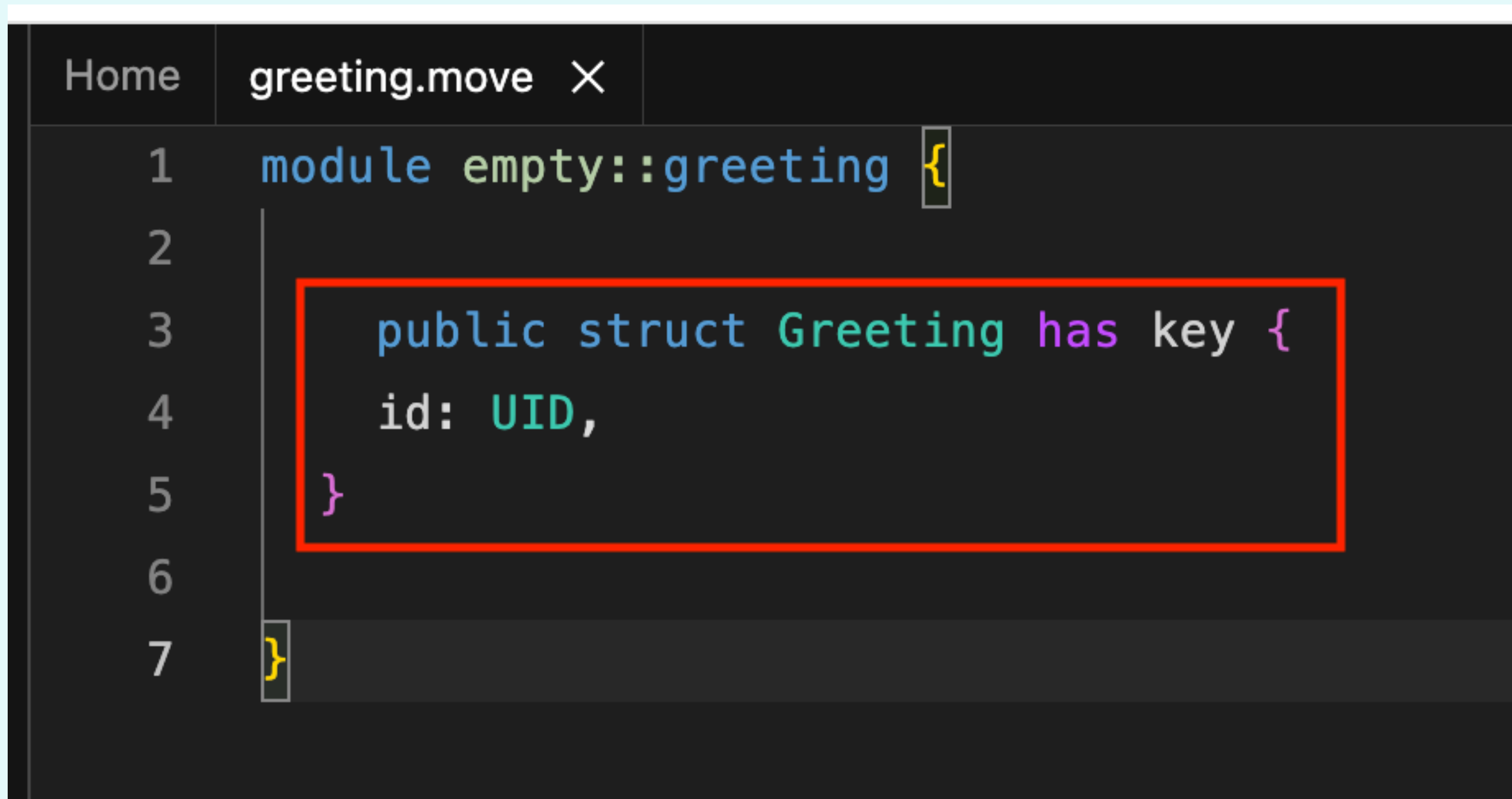
開発① モジュールの作成

sui move new <プロジェクト名>で作成後
下のように、外側を作ってみましょう。



開発② keyアビリティを持つ構造体

グローバル唯一の構造体を持ちます。



```
Home greeting.move ✕  
1 module empty::greeting {  
2  
3     public struct Greeting has key {  
4         id: UID,  
5     }  
6  
7 }
```

開発③ Contextの理解

コンテキスト：プログラム実行時に利用可能な定義済み変数の集合

```
module sui::tx_context;

/// Information about the transaction currently being executed.
/// This cannot be constructed by a transaction--it is a privileged object created by
/// the VM and passed in to the entrypoint of the transaction as `&mut TxContext`.

public struct TxContext has drop {
    /// The address of the user that signed the current transaction
    sender: address,
    /// Hash of the current transaction
    tx_hash: vector<u8>,
    /// The current epoch number
    epoch: u64,
    /// Timestamp that the epoch started at
    epoch_timestamp_ms: u64,
    /// Counter recording the number of fresh id's created while executing
    /// this transaction. Always 0 at the start of a transaction
    ids_created: u64
}
```

これがポイント
初期値は0でIDを作るたびに1増える

<https://move-book.com/programmability/transaction-context>

開発③ Contextの理解

可変参照か不変参照かで渡し方が異なります。

可変参照

`&mut TxContext`

不変参照

`&TxContext`

開発③ Contextの理解

fresh_object_address関数を見ると、ids_createdの+1が確認できる

```
module sui::tx_context;
```

```
/// Create an `address` that has not been used. As it is an object address, it will never  
/// occur as the address for a user.
```

```
/// In other words, the generated address is a globally unique object ID.
```

```
public fun fresh_object_address(ctx: &mut TxContext): address {
```

```
    let ids_created = ctx.ids_created;
```

```
    let id = derive_id(&ctx.tx_hash, ids_created); tx_hashとids_createdを元にID作成
```

```
    ctx.ids_created = ids_created + 1;
```

```
    id
```

ids_createdは+1されていく

```
}
```

<https://move-book.com/programmability/transaction-context>

開発④ オブジェクトの作成

object::new(ctx)でIDを作成します。
なぜ可変参照かも考えてみましょう。

```
Home greeting.move X
1  module empty::greeting {
2
3      public struct Greeting has key {
4          id: UID,
5      }
6
7      public fun new(ctx: &mut TxContext) {
8          Greeting {
9              id: object::new(ctx),
10         };
11     }
12
13 }
```

開発⑤ 所有者の設定

Suiでは所有者の設定が必要です。
ここでは、**共有**という所有にしましょう。

```
1  module empty::greeting {
2
3      public struct Greeting has key {
4          id: UID,
5      }
6
7      public fun new(ctx: &mut TxContext) {
8          let new_greeting = Greeting {
9              id: object::new(ctx),
10          };
11          transfer::share_object(new_greeting);
12      }
13
14 }
```

開発⑥ 文字の設定

Stringを用いて文字を設定しましょう。

```
Home greeting.move ×
1  module empty::greeting {
2      use std::string;
3
4      public struct Greeting has key {
5          id: UID,
6          text: string::String,
7      }
8
9      public fun new(ctx: &mut TxContext) {
10         let new_greeting = Greeting {
11             id: object::new(ctx),
12             text: b"Hello world!".to_string()
13         };
14         transfer::share_object(new_greeting);
15     }
16
17 }
```

開発⑦ 変更を行う関数

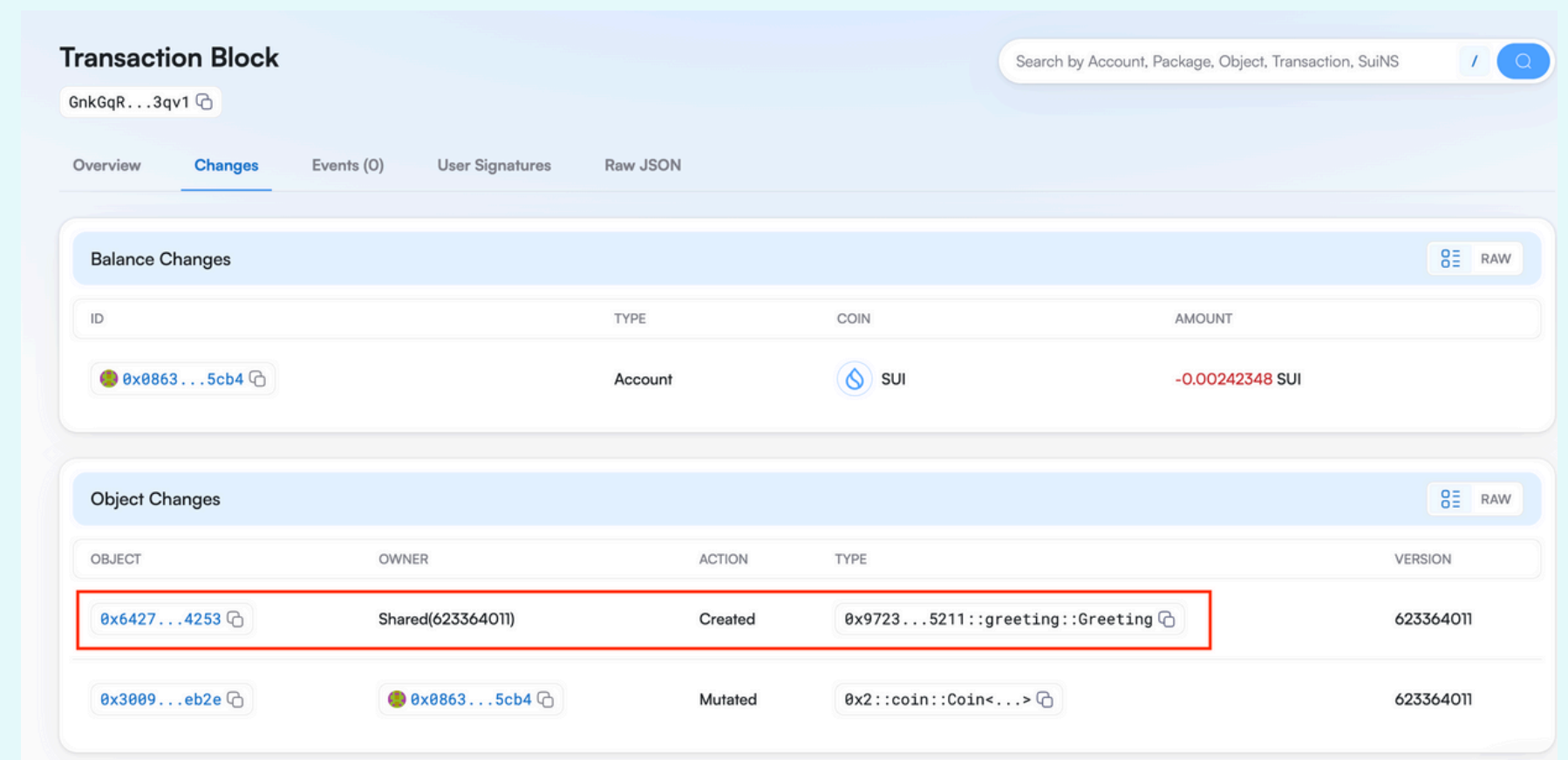
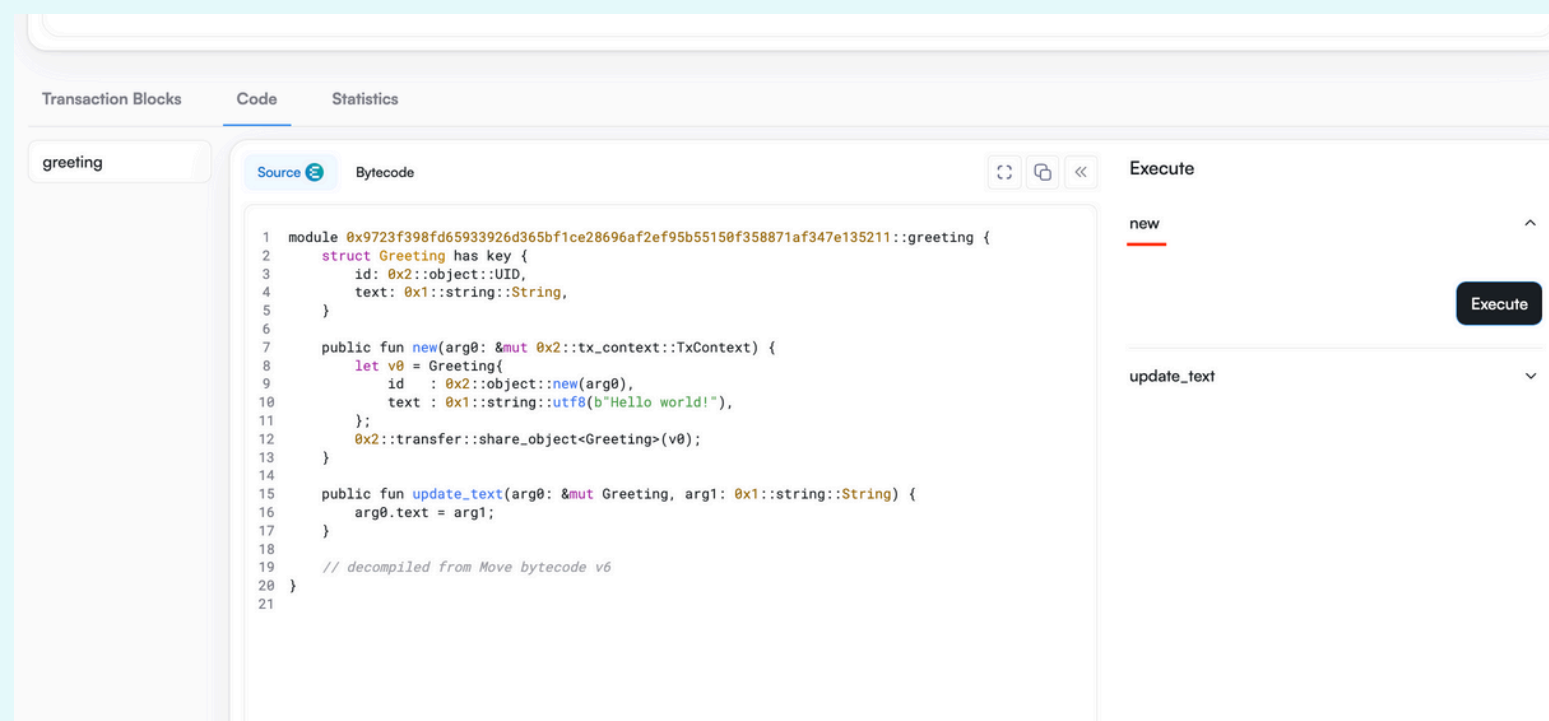
値を変更する関数を作しましょう。

```
Home greeting.move ×
1  module empty::greeting {
2      use std::string;
3
4      public struct Greeting has key {
5          id: UID,
6          text: string::String,
7      }
8
9      public fun new(ctx: &mut TxContext) {
10         let new_greeting = Greeting {
11             id: object::new(ctx),
12             text: b"Hello world!".to_string()
13         };
14         transfer::share_object(new_greeting);
15     }
16     public fun update_text(greeting: &mut Greeting, new_text: string::String) {
17         greeting.text = new_text;
18     }
19 }
```


開発⑧ SuiVisionでの実行

SuiVisionで作成しましょう。

また、実行できた**Object**にIDがついていることも確認しましょう。



開発⑨ 共有オブジェクトの確認

実行アドレスを変えて、うまくいくことを確認してみましょう。

The screenshot displays a web-based interface for interacting with a Move VM. It features three tabs at the top: 'Transaction Blocks', 'Code' (which is active), and 'Statistics'. On the left, a sidebar shows a list of modules, with 'greeting' selected. The main area is divided into two sections: 'Source' and 'Bytecode'. The 'Source' section contains the following code:

```
1 module 0x9723f398fd65933926d365bf1ce28696af2ef95b55150f358871af347e135211::greeting {  
2   struct Greeting has key {  
3     id: 0x2::object::UID,  
4     text: 0x1::string::String,  
5   }  
6  
7   public fun new(arg0: &mut 0x2::tx_context::TxContext) {  
8     let v0 = Greeting{  
9       id : 0x2::object::new(arg0),  
10      text : 0x1::string::utf8(b"Hello world!"),  
11    };  
12    0x2::transfer::share_object<Greeting>(v0);  
13  }  
14  
15  public fun update_text(arg0: &mut Greeting, arg1: 0x1::string::String) {  
16    arg0.text = arg1;  
17  }  
18  
19  // decompiled from Move bytecode v6  
20 }  
21
```

The 'Bytecode' section is currently empty. To the right of the code editor, there are icons for refreshing, copying, and a back arrow. Further right, under the 'Execute' heading, there are two input fields for arguments: 'Arg0' with the value '0x9723f398fd65933926d365bf1ce28696af2ef95b5' and 'Arg1' with the value '0x48656c6c6f'. A red box highlights the 'Execute' button at the bottom right of the interface.

開発10 実行者への所有者の設定

共有ではなく、自分を所有者にしてみましょう。

ctx.sender()が実行者を表します。

```
1  module empty::greeting {
2      use std::string;
3
4      public struct Greeting has key {
5          id: UID,
6          text: string::String,
7      }
8
9      public fun new(ctx: &mut TxContext) {
10         let new_greeting = Greeting {
11             id: object::new(ctx),
12             text: b"Hello world!".to_string()
13         };
14         transfer::public_transfer(new_greeting, ctx.sender());
15     }
16     public fun update_text(greeting: &mut Greeting, new_text: string::String) {
17         greeting.text = new_text;
18     }
19 }
```

開発11 public transferとstore

コンパイルしようとする、エラーになってしまいます。

storeがないためです。

どこかに入れるわけではないのになぜ？

```
Home greeting.move X
6   text: string::String,
7   }
8
9   public fun new(ctx: &mut TxContext) {
10      let new_greeting = Greeting {
11         id: object::new(ctx),
12         text: b"Hello world!".to_string()
13      };
14      transfer::public_transfer(new_greeting, ctx.sender());
15   }
16
17   public fun update_text(greeting: &mut Greeting, new_text: string::String) {
18      greeting.text = new_text;
19   }
```

Output Scan Log

```
10 | let new_greeting = Greeting {
11 | | id: object::new(ctx),
12 | | text: b"Hello world!".to_string()
13 | | };
14 | | transfer::public_transfer(new_greeting, ctx.sender());
    | ~~~~~ The type 'empty::greeting::Greeting' does not have the ability 'store'
    | ~~~~~ 'store' constraint not satisfied

/root/.move/https__github_com_MystenLabs_sui_git_framework__testnet/crates/sui-framework/packages/sui-
framework/sources/transfer.move:63:37
102 | public fun public_transfer(sui: Tx::Coin, recipient: address) {
```

開発11 public transferとstore

ルール上、送れないようです。

私の仮説は右です。これが許されると、すでに何かに格納されているものには送られようとした場合、整合性が取れなくなるからではと考えています。

```
module a::my_module;

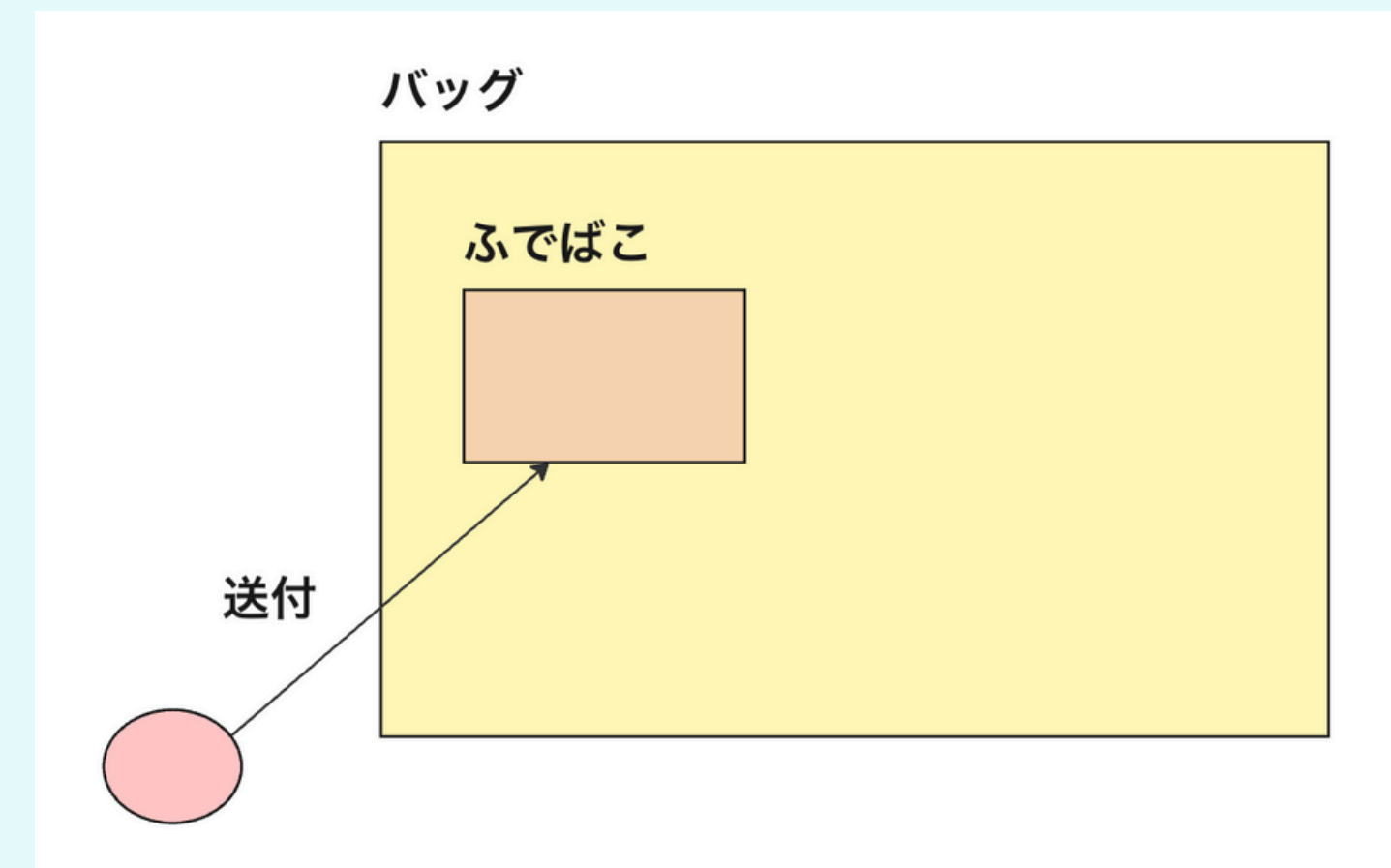
public struct A has key {
  id: sui::object::UID,
}

public struct B has key, store {
  id: sui::object::UID,
}
```

A can only be transferred using the `sui::transfer::transfer` inside of `a::my_module`, while B can be transferred anywhere using `sui::transfer::public_transfer`. These rules are enforced by a custom type system (bytecode verifier) rule in Sui.

A は `a::my_module` 内の `sui::transfer::transfer` を使用してのみ転送できますが、B `sui::transfer::public_transfer` を使用してどこにでも転送できます。これらのルールは、Suiのカスタムタイプシステム(バイトコード検証ツール)ルールによって適用されます。

<https://move-book.com/reference/abilities/object>

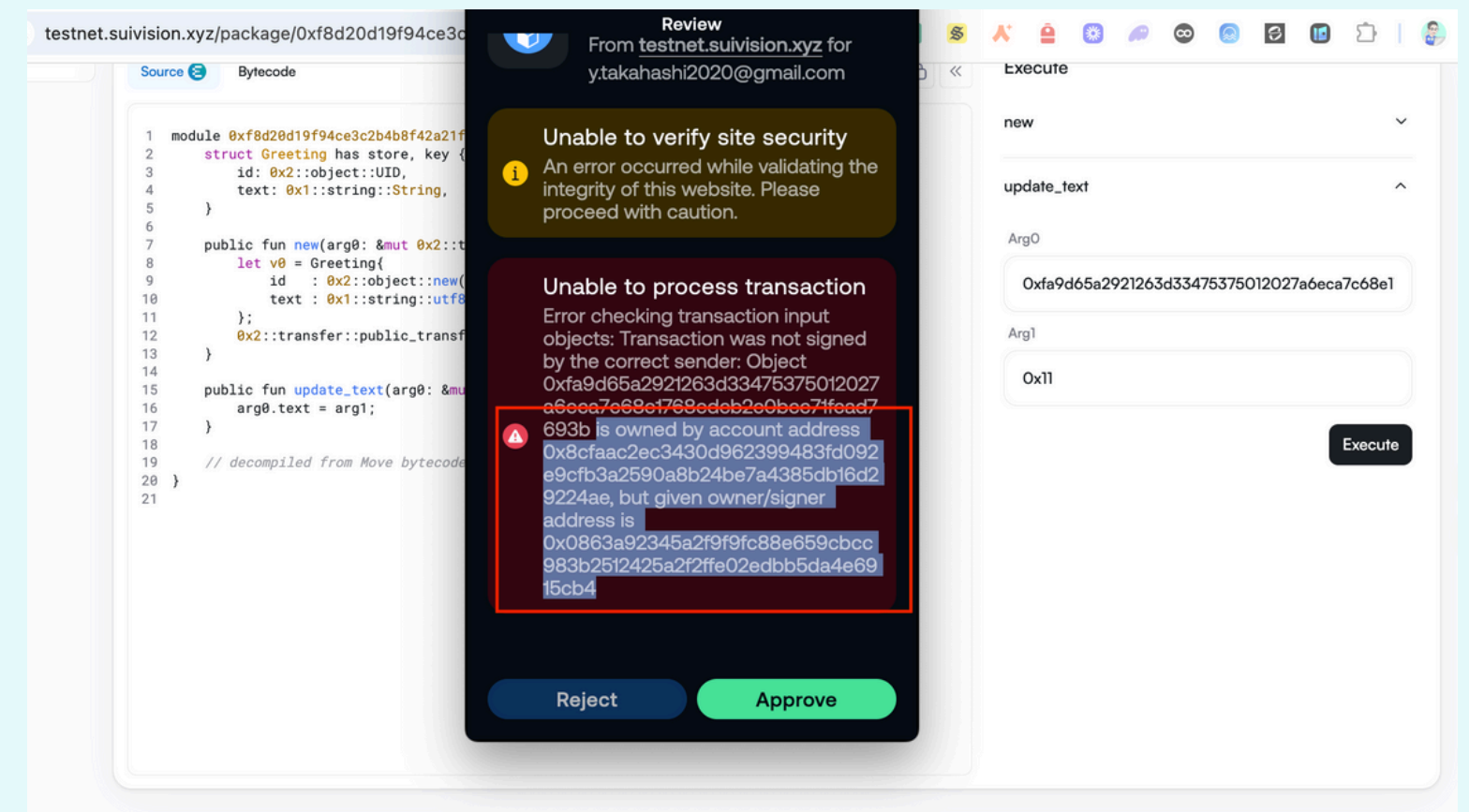


開発12 所有者以外で実行できない確認

storeをつけて、デプロイしましょう。

その後、新しくオブジェクトを作成し、**作成者以外で実行するとエラー**になることが確認できます。

```
Home greeting.move X
1 module empty::greeting {
2   use std::string;
3
4   public struct Greeting has key, store {
5     id: UID,
6     text: string::String,
7   }
8
9   public fun new(ctx: &mut TxContext) {
10     let new_greeting = Greeting {
11       id: object::new(ctx),
12       text: b"Hello world!".to_string()
13     };
14     transfer::public_transfer(new_greeting, ctx.sender());
15   }
16   public fun update_text(greeting: &mut Greeting, new_text: string::String) {
17     greeting.text = new_text;
18   }
19 }
```



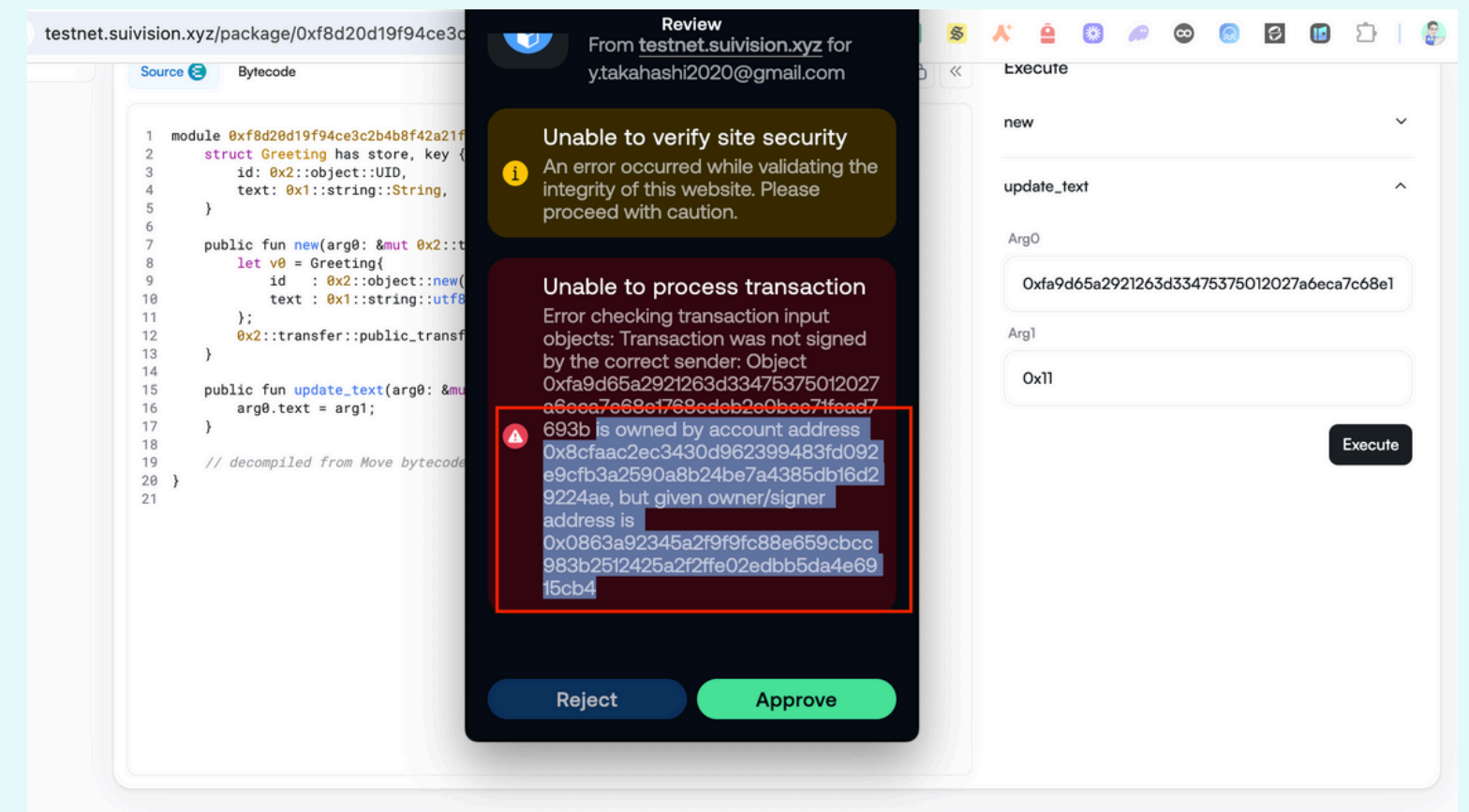
<https://move-book.com/reference/abilities/object>

開発12 所有者以外で実行できない確認

storeをつけて、デプロイしましょう。

その後、新しくオブジェクトを作成し、**作成者以外で実行するとエラー**になることが確認できます。

```
Home greeting.move X
1 module empty::greeting {
2   use std::string;
3
4   public struct Greeting has key, store {
5     id: UID,
6     text: string::String,
7   }
8
9   public fun new(ctx: &mut TxContext) {
10     let new_greeting = Greeting {
11       id: object::new(ctx),
12       text: b"Hello world!".to_string()
13     };
14     transfer::public_transfer(new_greeting, ctx.sender());
15   }
16   public fun update_text(greeting: &mut Greeting, new_text: string::String) {
17     greeting.text = new_text;
18   }
19 }
```



<https://move-book.com/reference/abilities/object>

開発13 コピーができないことの確認

Copy能力がついていないため、**コピーしようとするときエラーになる**ことを確認しましょう。

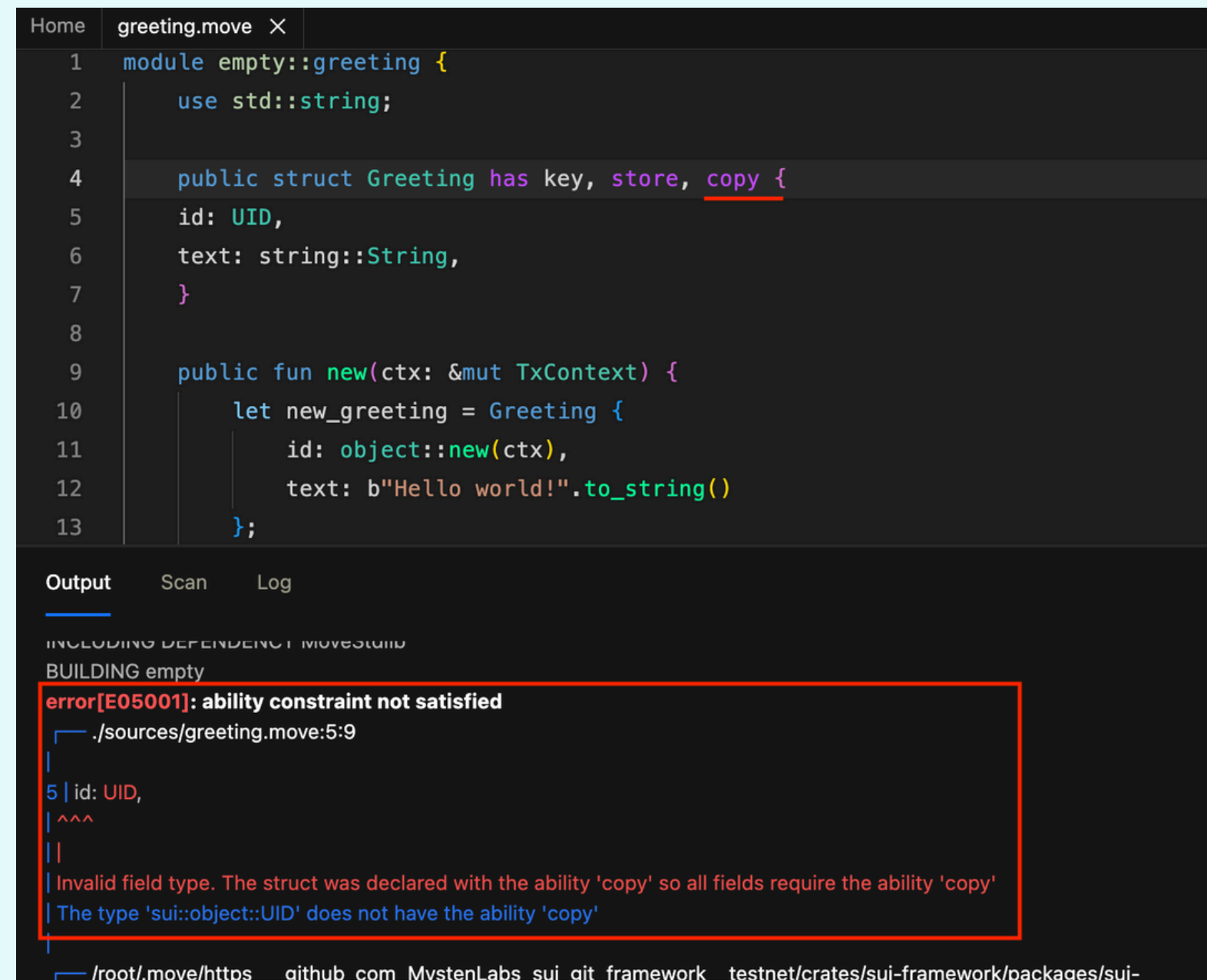
```
6      text: string::String,
7  }
8
9  public fun new(ctx: &mut TxContext) {
10      let new_greeting = Greeting {
11          id: object::new(ctx),
12          text: b"Hello world!".to_string()
13      };
14      let new_greeting2 = copy new_greeting;
15      transfer::public_transfer(new_greeting, ctx.sender());
16  }
17
18  public fun update_text(greeting: &mut Greeting, new_text: string::String) {
19      greeting.text = new_text;
20  }
```

Output Scan Log

```
11 || id: object::new(ctx),
12 || text: b"Hello world!".to_string()
13 || };
14 | _____' The type 'empty::greeting::Greeting' does not have the ability 'copy'
14 | let new_greeting2 = copy new_greeting;
14 | ^^^^^^^^^^^^^^^^^^^^^ Invalid 'copy' of owned value without the 'copy' ability
```

開発14 コピーとキーが両立しない確認

Keyはグローバル上の唯一性です。以下のようにコピーとは両立しません。



```
Home greeting.move X
1 module empty::greeting {
2   use std::string;
3
4   public struct Greeting has key, store, copy {
5     id: UID,
6     text: string::String,
7   }
8
9   public fun new(ctx: &mut TxContext) {
10     let new_greeting = Greeting {
11       id: object::new(ctx),
12       text: b"Hello world!".to_string()
13     };
14   }
15 }
```

Output Scan Log

INCLUDING DEPENDENCY movestd
BUILDING empty

error[E05001]: ability constraint not satisfied
└─ ./sources/greeting.move:5:9
5 | id: UID,
 | ^^^
 |
 | Invalid field type. The struct was declared with the ability 'copy' so all fields require the ability 'copy'
 | The type 'sui::object::UID' does not have the ability 'copy'

└─ /root/.move/https_github_com_MystenLabs_sui_git_framework_testnet/crates/sui-framework/packages/sui-

開発15 dropできないことの確認

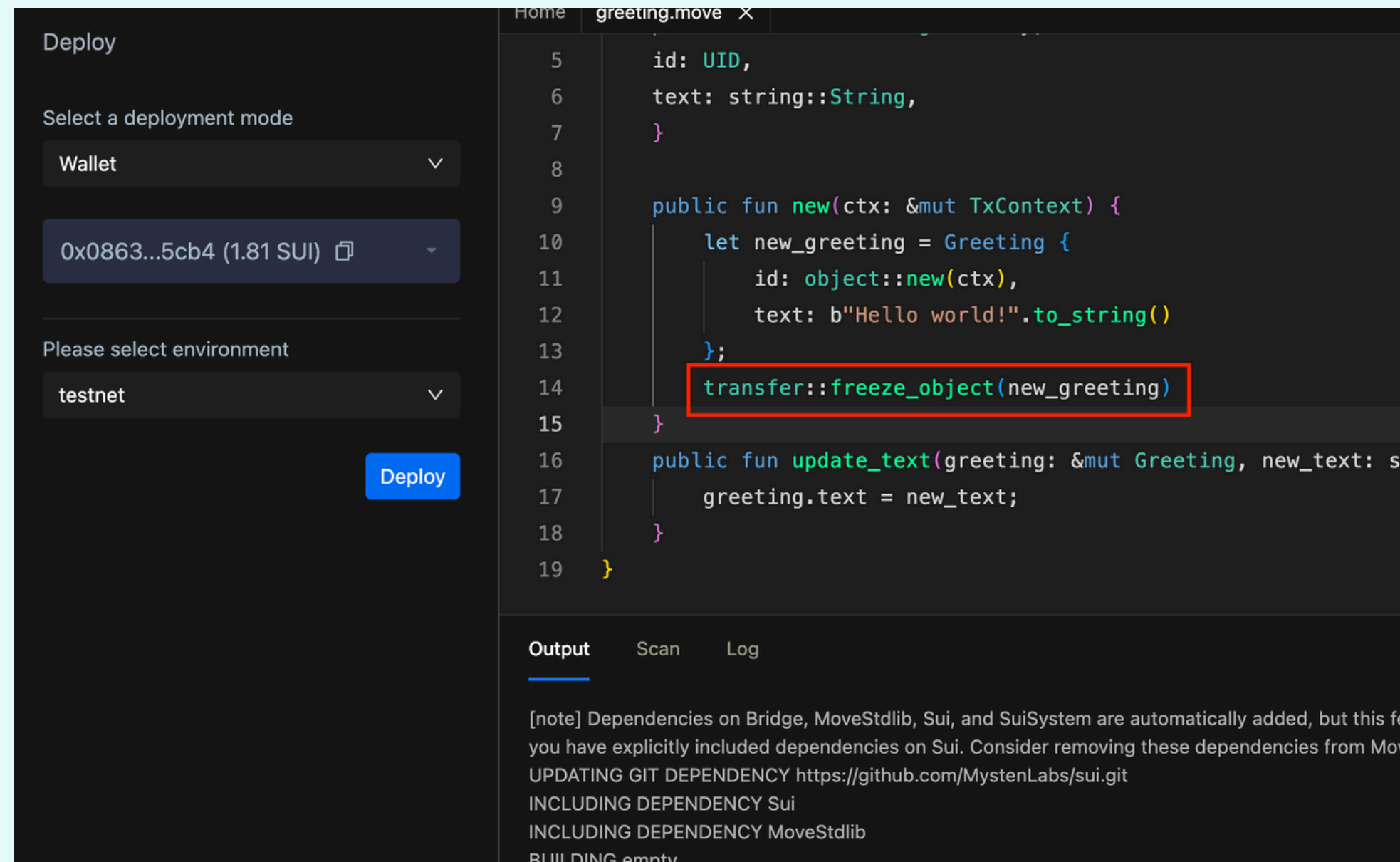
所有者を設定せずに終わろうとすると、**dropのエラー**になります。

```
5     id: UID,  
6     text: string::String,  
7 }  
8  
9 public fun new(ctx: &mut TxContext) {  
10     let new_greeting = Greeting {  
11         id: object::new(ctx),  
12         text: b"Hello world!".to_string()  
13     };  
14     // transfer::public_transfer(new_greeting, ctx.sender());  
15 }  
16 public fun update_text(greeting: &mut Greeting, new_text: string::String) {  
17     greeting.text = new_text;  
18 }  
19 }
```

error[E06001]: unused value without 'drop'
└─ ./sources/greeting.move:13:10
|
4 | public struct Greeting has key, store {
| ----- To satisfy the constraint, the 'drop' ability would need to be added here
.
10 | let new_greeting = Greeting {

開発16 Immutableの設定

Immutableにすることは所有者を設定しないことではありません。
「所有者なし」という所有の状態に設定しています。



開発17 Immutableの確認

オブジェクトがImmutableになることを確認し、**操作しようとするときエラーになることを確認しましょう。**

Transaction Block

34Tid2...5rRx

Overview **Changes** Events (0) User Signatures Raw JSON

Balance Changes

ID	TYPE	COIN	AMOUNT
0x0863...5cb4	Account	SUI	-0.0024

Object Changes

OBJECT	OWNER	ACTION	TYPE
0x2150...b40a	Immutable	Created	0xa7bc...4436::greeting::Greeting
0x3009...eb2e	0x0863...5cb4	Mutated	0x2::coin::Coin<...>

package/0xa7bc2ec9f7def0ca611e2b...

zAHPcZjNLK6SkcYob6qSKS3Vm3ScjoZTGzcJ7u8

ble

statistics

Bytecode

```
0xa7bc2ec9f7def0ca611e2b34b83a30cc884bfbebd9b85917f26ee975a1b4436::greeting {  
  let v0 = Greeting {  
    id : 0x2::object::new(arg0),  
    text : 0x1::string::utf8(b"Hello world!"),  
  };  
  0x2::transfer::freeze_object<Greeting>(v0);  
}  
  
public fun new(arg0: &mut 0x2::tx_context::TxContext) {  
  let v0 = Greeting {  
    id : 0x2::object::new(arg0),  
    text : 0x1::string::utf8(b"Hello world!"),  
  };  
  0x2::transfer::freeze_object<Greeting>(v0);  
}  
  
public fun update_text(arg0: &mut Greeting, arg1: 0x1::string::String) {  
  arg0.text = arg1;  
}
```

decompiled from Move bytecode v6

Review

Transaction request

From testnet.suivision.xyz for y.takahashi2020@gmail.com

Unable to verify site security

An error occurred while validating the integrity of this website. Please proceed with caution.

Unable to process transaction

Dry run failed, could not automatically determine a budget: CommandArgumentError { arg_idx: 0, kind: InvalidObjectByMutRef } in command 0

Reject Approve

リソースとオブジェクト

リソース

希少性を保証し、デジタル資産の複製を防ぎます。

オブジェクト

Suiの中核データ構造。所有、共有、不変のいずれかになり得ます。

所有権

Wrapping（ラップ）により単一所有を強制し、不正な複製を防ぎます。

移転（Transfer）

`transfer::transfer` を用いて、オブジェクトの所有権を安全に移転します。

型からの派生 ②オブジェクトと所有権

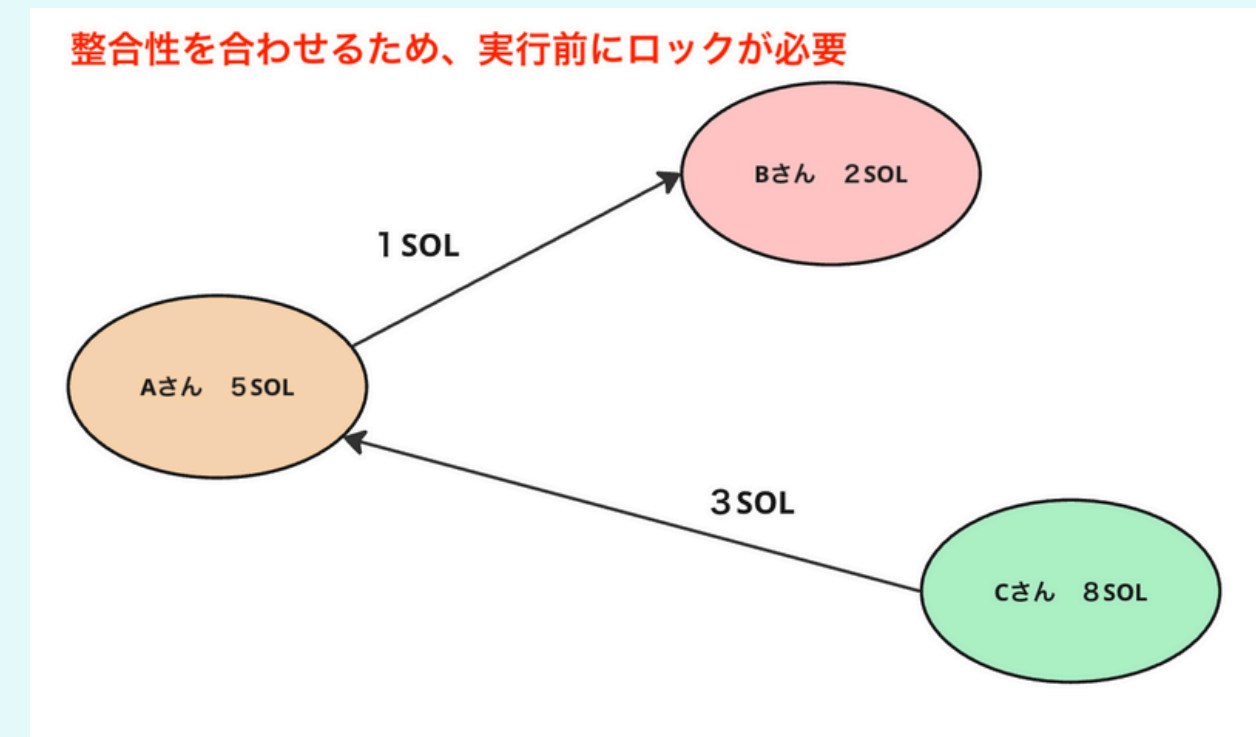
Suiはデータの値ではなく、**オブジェクトという形**で判断します。

また、**所有者**が外から明確に分かるようになっています。

まずは比較のためにSolanaで考えてみましょう。

下のようにAさんのアカウントが同時に利用されようとする場合です。

そのため、**実行前に書き込みが行われるアカウントを確認し、事前時にロック**をすることで整合性を保ちます。

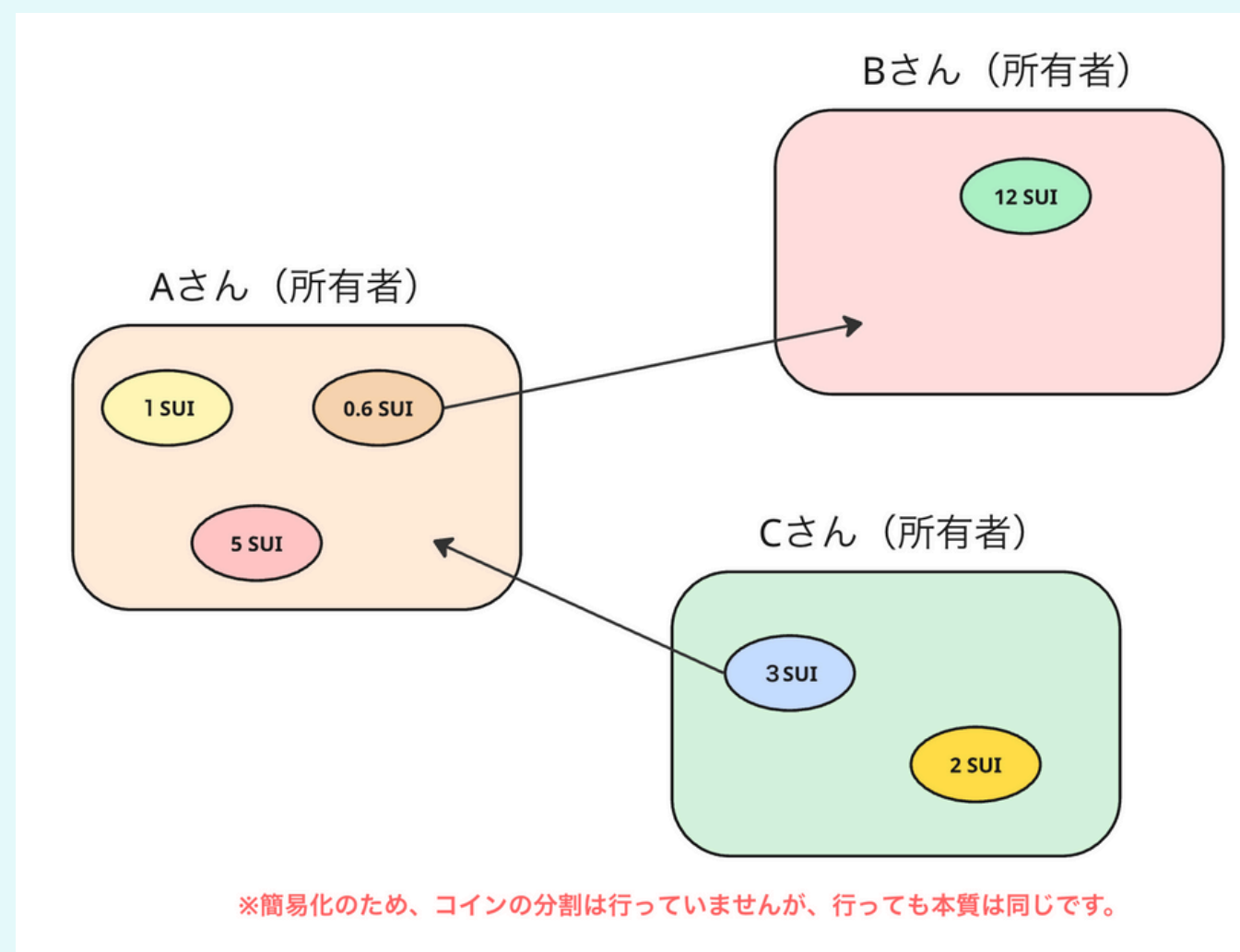


型からの派生 ②オブジェクトと所有権

Suiの場合はオブジェクトを元にした所有権の設計です。

あるコインがアドレス所有である場合、**そのコインを移動できるのは所有者だけ**のため、**整合性の問題がそもそも発生しません。**

そのため**機械的に並列処理**を行うことができます。



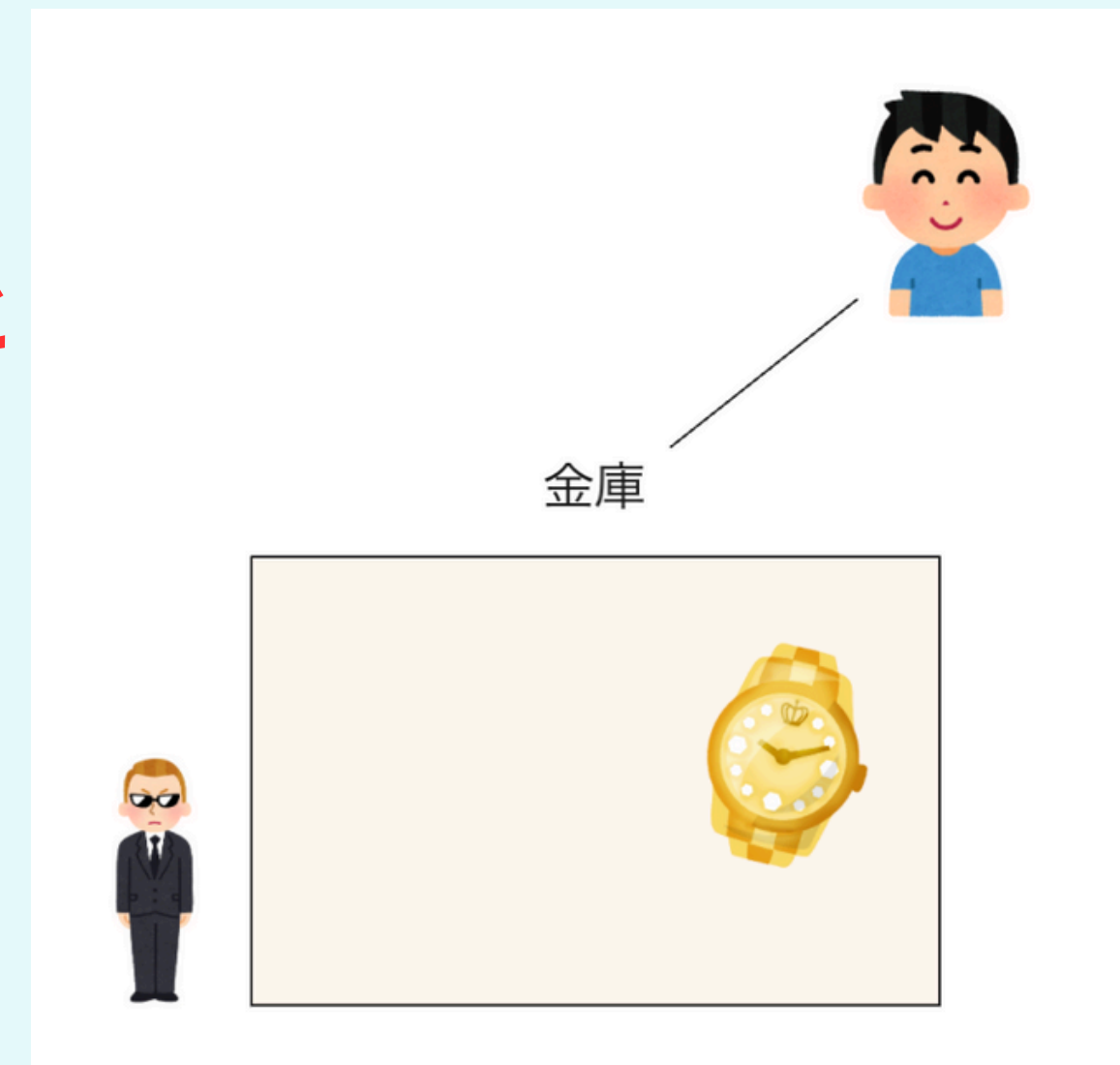
型からの派生 ③ラップと単一所有の強制

現実世界でいうところの所有と占有の話を考えてみましょう。

あなたの高級時計が金庫に保管され、大切に守られています。

この高級時計、動かせるのは誰でしょう。

守っている側からすると、**知らない間にあなたが動かせるとちょっとびっくりしてしまいそう**ですね。



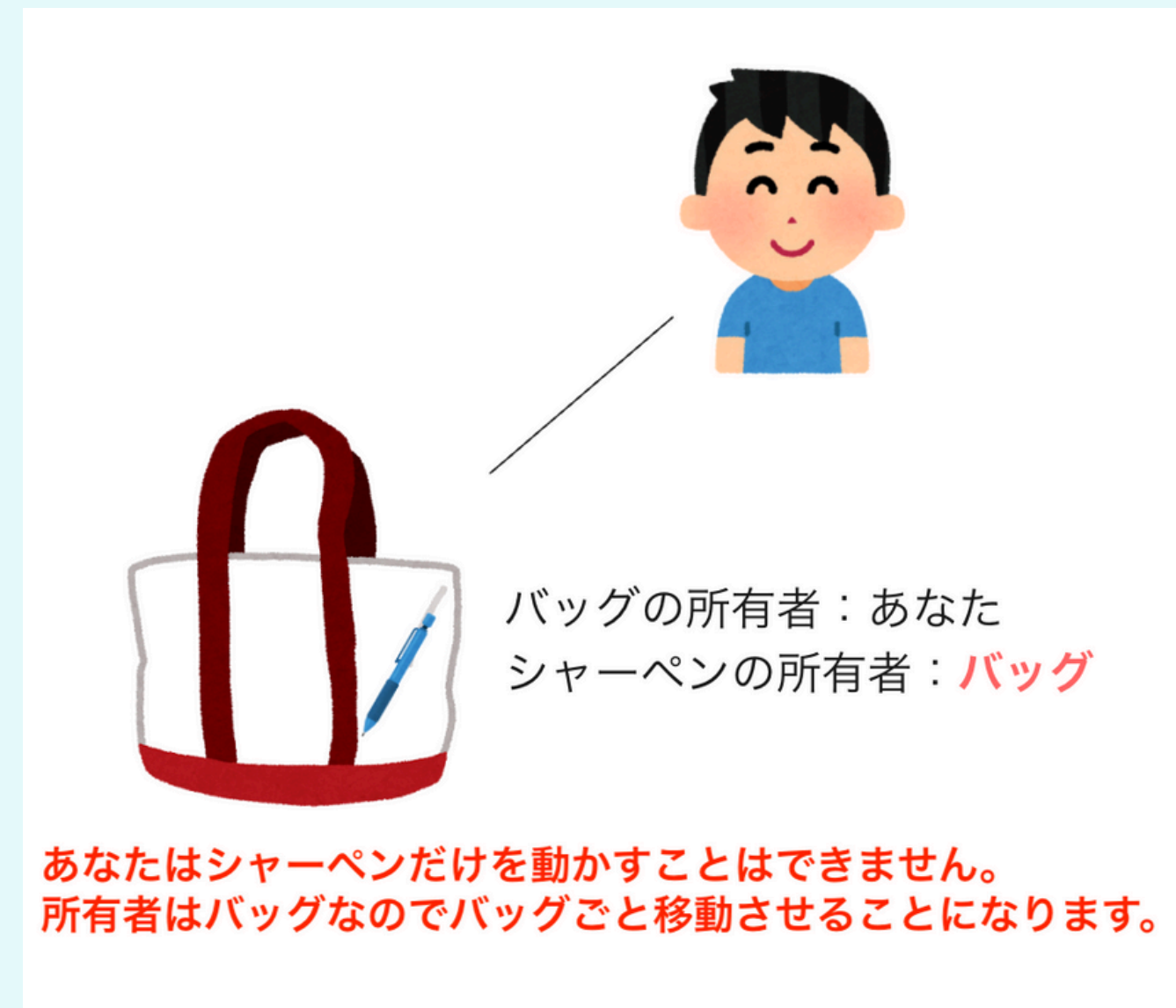
型からの派生 ③ラップと単一所有の強制

移動のイメージにしたいので、金庫でなく、バッグとシャーペンで考えます。

バッグにシャーペンがある場合、シャーペンの所有者は**あなたでなく、バッグのみ**です。

これによって**所有者が一意**になり、明確です。

動かしたいときはバッグの所有者のあなたが**バッグごと動かせば良い**のです。



開発18 Wrapと単一所有の確認

実際に試してみましよう。

```
greeting.move x
empty > sources > greeting.move > sui-move-analyzer > {} wrap_demo
1  module empty::wrap_demo;
2
3  use std::string::String;
4  use sui::object::uid_to_address;
5
6  /// 親
7  public struct Parent has key, store {
8      id: UID,
9      name: String,
10 }
11
12 /// 子
13 public struct Child has key, store {
14     id: UID,
15     note: String,
16 }
17
18 #[allow(lint(self_transfer))]
19 public fun new_parent(ctx: &mut TxContext) {
20     let p: Parent: Parent = Parent {
21         id: UID: object::new(ctx: ctx),
22         name: String: b"parent".to_string(),
23     };
24     transfer::public_transfer( obj: obj: p, recipient: recipient: ctx.sender());
25 }
26
27 #[allow(lint(self_transfer))]
28 public fun new_child(ctx: &mut TxContext) {
29     let c: Child: Child = Child {
30         id: UID: object::new(ctx: ctx),
31         note: String: b"child".to_string(),
32     };
33     transfer::public_transfer( obj: obj: c, recipient: recipient: ctx.sender());
34 }
```

https://github.com/ytakahashi2020/sui_send

代表的な設計パターン

データをオブジェクトとして扱う

Suiのモデルはデータをオブジェクト（所有または共有）として保存します。

SuiのCapability（権限）

Capabilityによってアクセスを制御し、許可された操作のみを許容します。

One-time Witnessパターン

一意に作成された証明オブジェクトにより、操作を一度だけ実行可能にします。

型からの派生 ④Capabilityによる権限

Solanaの場合、権限チェックは**実行時**に行われます。

例えば、「**MintTo**」という関数の場合、**配列の3番目**のアカウントが**ミント権限者**です。

そして、その者の署名があるか確認します。

大事なのは関数ごとに**権限者かが何か異なる**点です。

そのため、**機械的なチェックができず**、実行時に確認します。

```
Transaction {
  // 署名は「message全体」に対するもの (Instructionごとではない)
  signatures: [
    sig_P, // account_keys[0] = P (fee payer)
    sig_M, // account_keys[1] = M (mint authority)
  ],

  message: Message {
    header: MessageHeader {
      // 先頭から num_required_signatures 個のアカウントが署名必須
      num_required_signatures: 2, // P と M の2署名が必要
      num_readonly_signed_accounts: 0, // 読み取り専用の署名アカウント数
      num_readonly_unsigned_accounts: 1, // 読み取り専用の非署名 (例: TOKEN_PROG)
    },

    // このTxで参照するアカウントのリスト (Instruction側は index で参照)
    account_keys: [
      P, // [0] fee payer (署名者) is_signer=true, is_writable=true(手数料)
      M, // [1] mint authority (署名者) is_signer=true, is_writable=false(通常)
      MINT, // [2] Mintアカウント is_signer=false, is_writable=true
      D, // [3] 受取トークン口座 is_signer=false, is_writable=true
      TOKEN_PROG, // [4] SPL Token Program is_signer=false, is_writable=false
    ],

    recent_blockhash: Hash(...),

    instructions: [
      Instruction {
        program_id: TOKEN_PROG, // = account_keys[4]

        // この命令で使うアカウント (上の account_keys の index 参照)
        // mint_to の標準的な並び: [mint, dest_token_account, authority]
        accounts: [2, 3, 1], // [MINT, D, M]

        // 関数+引数をシリアライズしたバイト列 (概念表示)
        data: MintTo { amount: 100 },
      },
    ],
  },
}
```

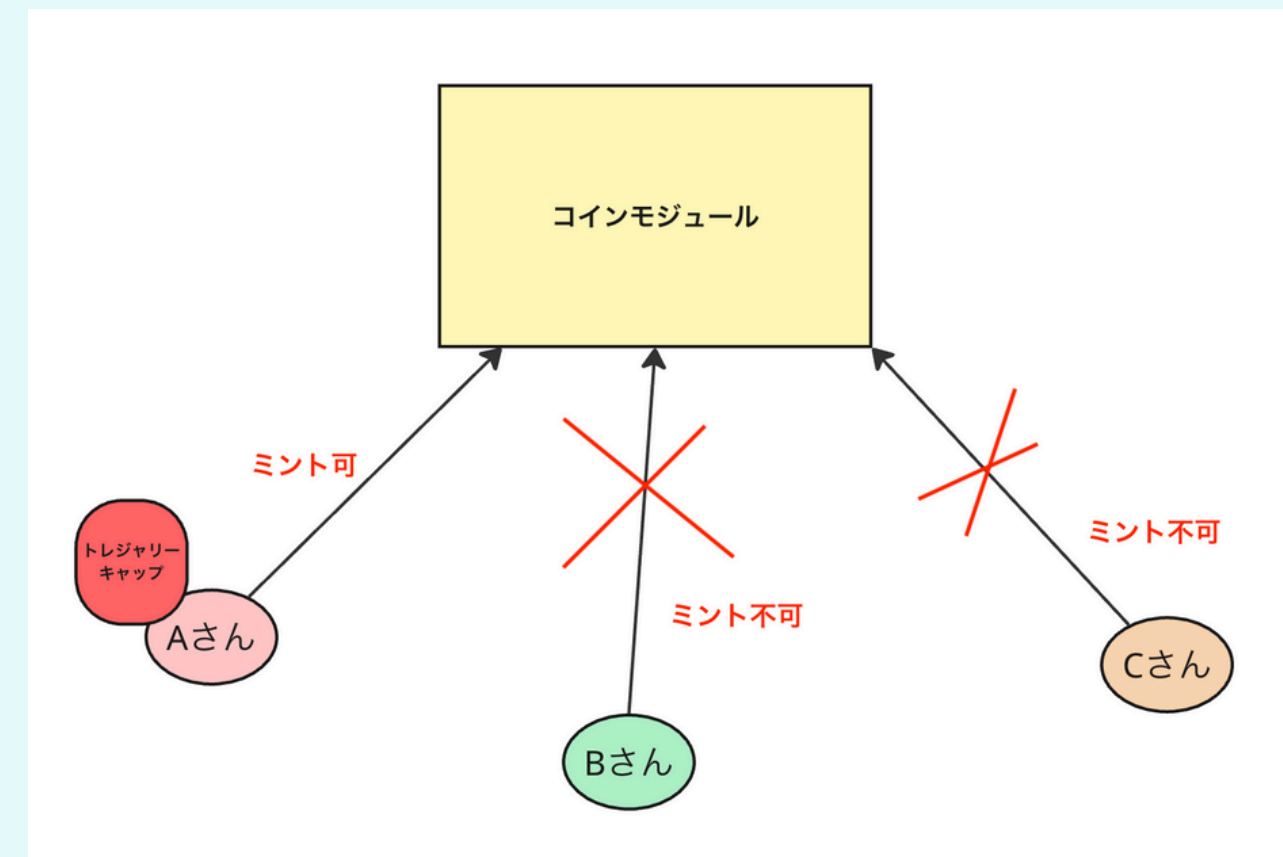

型からの派生 ④Capabilityによる権限

Suiの場合、権限チェックは**コンパイル時**に行われます。

Suiの場合は**権限オブジェクトを持っている人が権限者**というシンプルな構造です。

そのため、実行時に形式的に判断ができます。

また、コンパイル時に「**ミント関数に権限オブジェクトが引数に設定されているか**」の形式的なチェックで済みます。



型からの派生 ⑤OTWによる一度実行

一度きりの処理には**OTW (One-Time Witness)** を引数として渡します。

これにより、一度きりであることが客観的にわかります。

```
module examples::my_coin_new;

use sui::coin_registry;

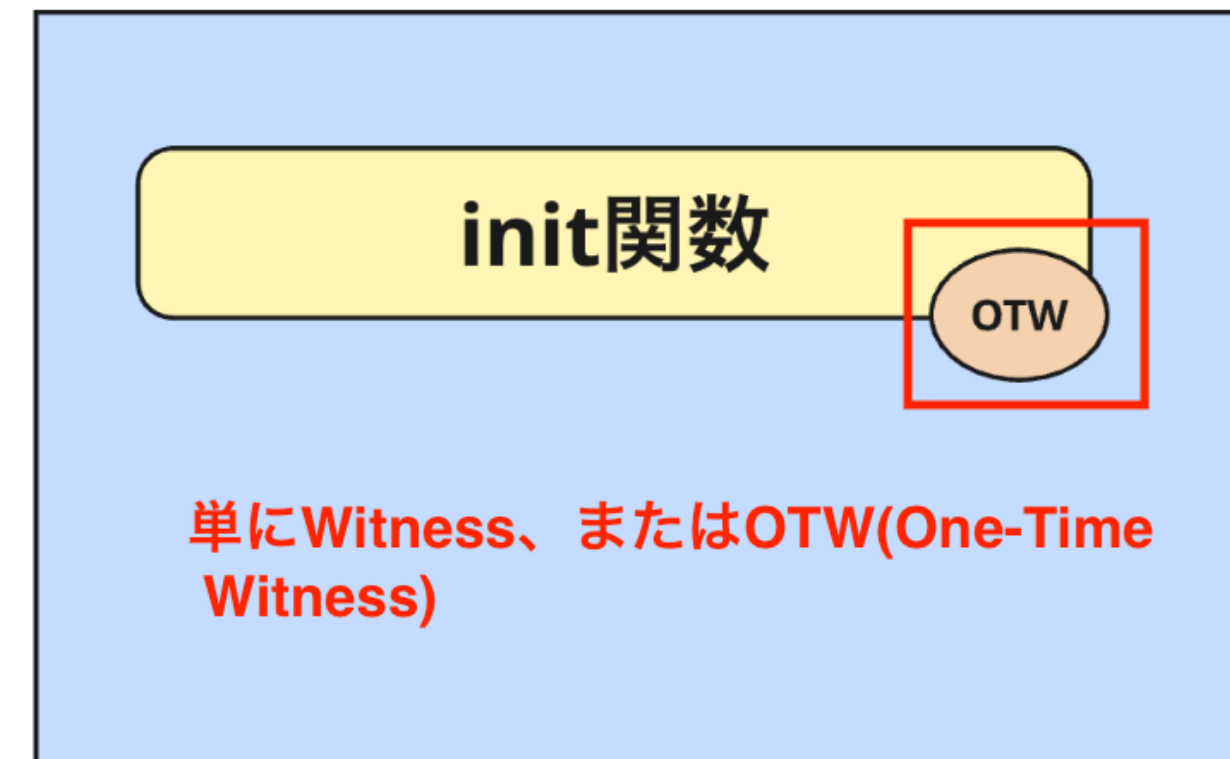
// The type identifier of coin. The coin will have a type
// tag of kind: `Coin<package_object::mycoin::MYCOIN>`
// Make sure that the name of the type matches the module's name.
public struct MY_COIN_NEW has drop {}

// Module initializer is called once on module publish. A `TreasuryCap` is sent
// to the publisher, who then controls minting and burning. `MetadataCap` is also
// sent to the Publisher
fun init(witness: MY_COIN_NEW, ctx: &mut TxContext) {
  let (builder, treasury_cap) = coin_registry::new_currency_with_otw(
    witness,
    6, // Decimals
    b"MY_COIN".to_string(), // Symbol
    b"My Coin".to_string(), // Name
    b"Standard Unregulated Coin".to_string(), // Description
    b"https://example.com/my_coin.png".to_string(), // Icon URL
    ctx,
  );

  let metadata_cap = builder.finalize(ctx);

  // Freezing this object makes the metadata immutable, including the title, name, and icon
  // If you want to allow mutability, share it with public_share_object instead.
  transfer::public_transfer(treasury_cap, ctx.sender());
  transfer::public_transfer(metadata_cap, ctx.sender());
}
```

my_coin_newモジュール



<https://docs.sui.io/guides/developer/currency>

開発19 OTWの作成

コイン作成の例で先ほどの内容を学びましょう。

まずは、otwを作成します。 **dropアビリティ**がついています。

```
Home my_coin_new.move ×  
1  module empty::my_coin_new;  
2  
3  
4  public struct MY_COIN_NEW has drop {}  
5  
6
```

開発20 Structと可視性

現在、**Structは必ずpublicが必要**です。
将来の可視性に備えているためです。

Struct declarations 構造体宣言

Currently, struct declarations can only be `public`, so it's not necessary to include the visibility modifier. To make room for a future where struct types have visibility other than `public`, the 2024 edition of Move will require `public` on all struct declarations.

現在、構造体宣言は パブリック にしかできないため、可視性修飾子を含める必要はありません。構造体型がパブリック以外の可視性を持つ将来に備える余地を確保するために、2024 年版の Move では、すべての構造体宣言で パブリック が義務付けられます。

<https://blog.sui.io/move-edition-2024-update>

開発21 init関数

公開時に一度だけ実行されるのが**init関数**です。
witnessを引数に取ることで**一度のみの実行**であることを示します。

```
Home my_coin_new.move ✕  
1  module empty::my_coin_new;  
2  
3  use sui::coin_registry;  
4  
5  public struct MY_COIN_NEW has drop {}  
6  
7  fun init(witness: MY_COIN_NEW, ctx: &mut TxContext) {  
8  
9  }
```

<https://docs.sui.io/guides/developer/currency>

開発22 new_currency_with_otw

コインを作成してみましょう。

「**treasury_cap**」が権限である、**Capability**です。

```
Home my_coin_new.move X
1 module empty::my_coin_new;
2
3 use sui::coin_registry;
4
5 public struct MY_COIN_NEW has drop {}
6
7 fun init(witness: MY_COIN_NEW, ctx: &mut TxContext) {
8     let (builder, treasury_cap) = coin_registry::new_currency_with_otw(
9         witness,
10         6, // Decimals
11         b"MY_COIN".to_string(), // Symbol
12         b"My Coin".to_string(), // Name
13         b"Standard Unregulated Coin".to_string(), // Description
14         b"https://example.com/my_coin.png".to_string(), // Icon URL
15         ctx,
16     );
17
18 }
```

開発23 metadata_capの取得

finalize関数によって、メタデータの修正権限である、「**metadata_cap**」を取得します。

```
Home my_coin_new.move ×
1 module empty::my_coin_new;
/home
3 use sui::coin_registry;
4
5 public struct MY_COIN_NEW has drop {}
6
7 fun init(witness: MY_COIN_NEW, ctx: &mut TxContext) {
8     let (builder, treasury_cap) = coin_registry::new_currency_with_otw(
9         witness,
10         6, // Decimals
11         b"MY_COIN".to_string(), // Symbol
12         b"My Coin".to_string(), // Name
13         b"Standard Unregulated Coin".to_string(), // Description
14         b"https://example.com/my_coin.png".to_string(), // Icon URL
15         ctx,
16     );
17     let metadata_cap = builder.finalize(ctx);
18
19 }
```

開発24 Capabilityの送付

発行権限、メタデータの修正権限を実行者に送ります。

```
Home my_coin_new.move X
1  module empty::my_coin_new;
2
3  use sui::coin_registry;
4
5  public struct MY_COIN_NEW has drop {}
6
7  fun init(witness: MY_COIN_NEW, ctx: &mut TxContext) {
8      let (builder, treasury_cap) = coin_registry::new_currency_with_otw(
9          witness,
10         6, // Decimals
11         b"MY_COIN".to_string(), // Symbol
12         b"My Coin".to_string(), // Name
13         b"Standard Unregulated Coin".to_string(), // Description
14         b"https://example.com/my_coin.png".to_string(), // Icon URL
15         ctx,
16     );
17     let metadata_cap = builder.finalize(ctx);
18
19     transfer::public_transfer(treasury_cap, ctx.sender());
20     transfer::public_transfer(metadata_cap, ctx.sender());
21
22 }
```


開発25 public, entry関数

でもこんなふうにやればできます。
これってSuiVisionでできないのでしょうか？

```
base ~ (5.455s)
sui client call \
  --package 0x2 \
  --module coin \
  --function mint_and_transfer \
  --type-args 0x4dfcb9827820316b9e81a685de76921e13ebf71c093baaafbcea273241cb945c::my_coin_new::MY_COIN_NEW \
  --args 0x402ebf3fab8c8a64bf1c2faa1db9818d179e9bab3dc8fe2897ba20d1ea405f45 10000 0x917bb0d388ae4557da41a3d8fa4b04eb31f68e01325770f8fe10f35bc0c9bb2f

[warning] Client/Server api version mismatch, client api version : 1.57.2, server api version : 1.59.0
Transaction Digest: CcccL7wsRq41VjGPQViPVzXXXzxc7z3WgG2uG1WqgK99

Transaction Data

Sender: 0x917bb0d388ae4557da41a3d8fa4b04eb31f68e01325770f8fe10f35bc0c9bb2f
Gas Owner: 0x917bb0d388ae4557da41a3d8fa4b04eb31f68e01325770f8fe10f35bc0c9bb2f
Gas Budget: 4434888 MIST
Gas Price: 1000 MIST
Gas Payment:
```


開発26 0x2パッケージ

0x2パッケージで実行してみましょう。

The screenshot displays the Suivision testnet interface. On the left, a sidebar lists various transaction blocks, with 'coin' highlighted. The main area shows the source code for the 'coin' package, which defines a 'Coin' type and includes functions like 'mint' and 'mint_and_transfer'. On the right, the 'Execute' tab is active, showing the execution results for the 'mint_and_transfer' function. The results include the function name, type arguments, and three positional arguments: 'TreasuryCap', 'amount', and 'recipient'. The 'TreasuryCap' argument is highlighted in blue.

```
1 // Copyright (c) Mysten Labs, Inc.
2 // SPDX-License-Identifier: Apache-2.0
3
4 /// Defines the `Coin` type - platform wide representation of fungible
5 /// tokens and coins. `Coin` can be described as a secure wrapper around
6 /// `Balance` type.
7 module sui::coin {
8     use std::string;
9     use std::ascii;
10    use std::option::{Self, Option};
11    use sui::balance::{Self, Balance, Supply};
12    use sui::tx_context::TxContext;
13    use sui::object::{Self, UID};
14    use sui::transfer;
15    use sui::url::{Self, Url};
16    use std::vector;
17
18    /// A type passed to create_supply is not a one-time witness.
19    const EBadWitness: u64 = 0;
20    /// Invalid arguments are passed to a function.
21    const EInvalidArg: u64 = 1;
22    /// Trying to split a coin more times than its balance allows.
23    const ENotEnough: u64 = 2;
24
25    /// A coin of type `T` worth `value`. Transferable and storable
26    struct Coin<phantom T> has key, store {
27        id: UID,
28        balance: Balance<T>
29    }
30
31    /// Each Coin type T created through `create_currency` function will have a
32    /// unique instance of CoinMetadata<T> that stores the metadata for this coin type.
33    struct CoinMetadata<phantom T> has key, store {
34        id: UID,
35        /// Number of decimal places the coin uses
36    }
```

Function: mint_and_transfer
Type Arguments:
0xf0bcb4e4c92f1e966918e793c85d07ce604939b3101421ae43c2e3f4d9385f1b::my_coin_new::MY_COIN_NEW

Arguments:
Arg0 (TreasuryCap): 0x28456cb129952ada3de7ac59ff388f904e32485a03bd995c1408b83d81676e4a
Arg1 (amount): 111 (好きな数字に変更可能)
Arg2 (recipient): 0x0863a92345a2f9f9fc88e659cbcc983b2512425a2f2ffe02edbb5da4e6915cb4

<https://testnet.suivision.xyz/package/0x2?tab=Code>

開発27 entry関数の作成

以下のようにmint関数を作ります。

<MY_COIN_NEW>で型を設定しています。

```
Home my_coin_new.move X Object Explorer
1  module empty::my_coin_new;
2
3  use sui::coin;
4  use sui::coin_registry;
5
6  public struct MY_COIN_NEW has drop {}
7
8  > fun init(witness: MY_COIN_NEW, ctx: &mut TxContext) { ...
22 }
23
24 ∨ entry fun mint(
25     cap: &mut coin::TreasuryCap<MY_COIN_NEW>,
26     amount: u64,
27     recipient: address,
28     ctx: &mut TxContext
29 ∨ ) {
30     let new_coin = coin::mint<MY_COIN_NEW>(cap, amount, ctx);
31     transfer::public_transfer(new_coin, recipient);
32 }
```

開発28 SuiVisionでの実行

SuiVisionで実行ができることを確認しましょう。

Publisher:

0x0863...5cb4

Publish Time:

Oct 24, 2025 02:22:05 +UTC

Last Transaction ID:

FfmB8pvnpKDAHGMfu3PSHarMTWALjULdzV1HGbfMgtLe

Version:

1

Owner:

Immutable

Transaction Blocks

Code

Statistics

my_coin_new

Source

Bytecode

```
1 module 0xbde479c784c71eb37fed2005cfbdb9f555adff2469ac9791524a94a0ae0e7865::my_coin_new {
2   struct MY_COIN_NEW has drop {
3     dummy_field: bool,
4   }
5
6   entry fun mint(arg0: &mut 0x2::coin::TreasuryCap<MY_COIN_NEW>, arg1: u64, arg2: address, arg3:
7     0x2::transfer::public_transfer<0x2::coin::Coin<MY_COIN_NEW>>(0x2::coin::mint<MY_COIN_NEW>{
8   }
9
10  fun init(arg0: MY_COIN_NEW, arg1: &mut 0x2::tx_context::TxContext) {
11    let (v0, v1) = 0x2::coin_registry::new_currency_with_otw<MY_COIN_NEW>(arg0, 6, 0x1::string
12      0x2::transfer::public_transfer<0x2::coin::TreasuryCap<MY_COIN_NEW>>(v1, 0x2::tx_context::s
13      0x2::transfer::public_transfer<0x2::coin_registry::MetadataCap<MY_COIN_NEW>>(0x2::coin_reg
14    )
15  }
16  // decompiled from Move bytecode v6
17 }
18 }
```

Execute

mint

Arg0

Ox6af242069e8718fa4fbd54f78527d64c4937193b:

Arg1

1111

Arg2

Ox884344b72b6d2d20684b91ce84beaf8b8cbce8:

Execute

SuiVision

Blockchain

Validators

Ox0863...5cb4

Object

0x96dd...5dfb

Search by Account, Package, Object, Transaction, SuiNS

Details

Owner:

0x8843...d33b

Publisher:

0x0863...5cb4

Type:

0x2::coin::Coin<0xbde4...7865::my_coin_new>

Version:

623364027

Transaction Block Digest:

G1wKwP...ePp8

Fields

```
{
  balance: "1111"
  id: {
    id: "0x96dd0504599d940dab5d783bf6121ffb60dc9057685732d39becaece5fdf5dfb"
  }
}
```

Dynamic Fields

CHILD OBJECT ID	VERSION	OBJECT TYPE	LAST TX ID
-----------------	---------	-------------	------------

セキュリティの考慮事項

ガス最適化

高コストなトランザクションによるサービス拒否を避けるため、コストを最小化します。

リエントランシー対策

リソースモデルによりリエントランシー攻撃を効率的に阻止します。

形式検証

重要なコントラクトロジックの正しさを数学的に証明します。

型からの派生 ⑥ガス最適化

Ethereumの場合、計算コストとストレージコストが合算してガスPriceにかけていますが、**Solana・Suiは単価を分けて**います。

Solanaの場合は、事前にレンタル代として確保するので、書き方を変えています。

Suiでは**オブジェクトで所有者が決まっている**ので、**ストレージの計算が明瞭**です。

Ethereum		
ガス使用量 × ガスPrice		
Solana		
署名数 × 基本手数料 + (CUPrice × 設定CU)	+	(data_length × rent_exemption_rate)
Sui		
computation_units × reference_gas_price	+	storage_units × storage_price

型からの派生 ⑦形式的検証

Suiは客観的に数学的にロジックが正しいことを検証できます。

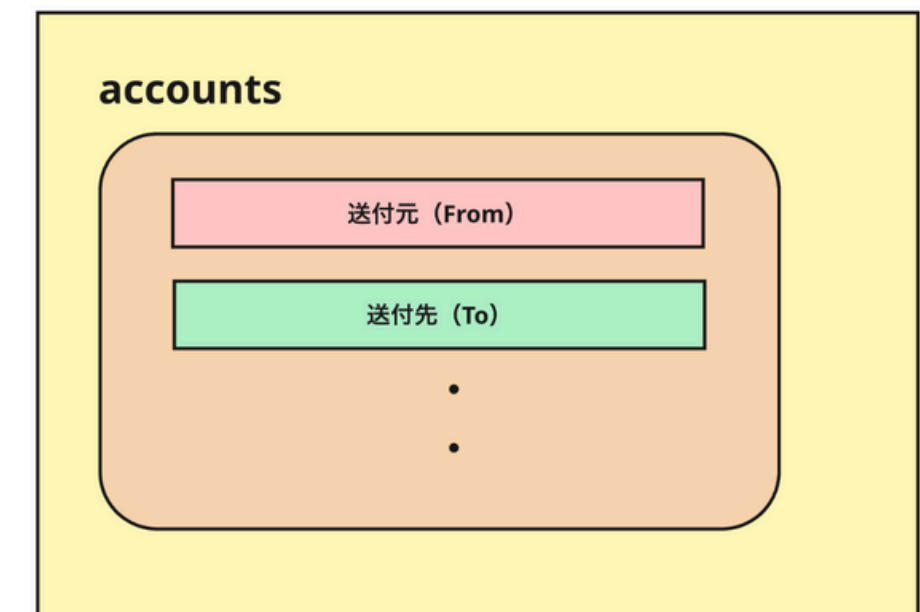
例えば、Solanaで送付時に総量が変わらないことを客観的に証明してみましょう。

動的に中身が変わったり、他のトランザクションの影響もありうるので、証明すべきパターンが多くなり過ぎてしまいます。

```
fn transfer(ctx: Context<Transfer>, amount: u64) -> Result<()> {  
    let from = &mut ctx.accounts.from;  
    let to = &mut ctx.accounts.to;  
    from.amount -= amount;  
    to.amount += amount;  
    Ok()  
}
```

関数にContextという箱を渡し、
その中に入っているアカウントを操作する

Context<Transfer>

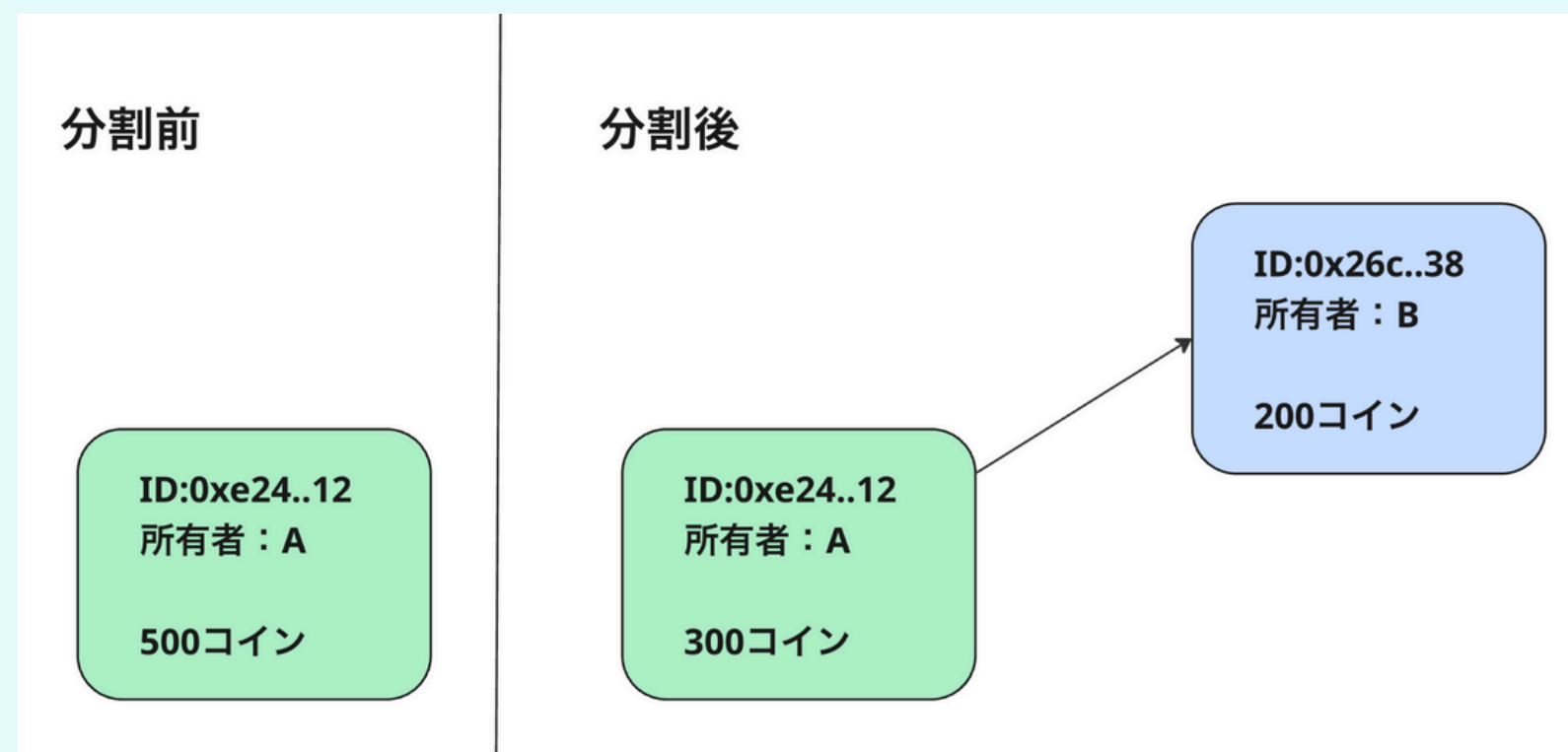


型からの派生 ⑦形式的検証

Suiの場合は「何を」「いくら」「誰に」渡すかを具体的に渡すため、**証明すべき範囲が明確**です。

それだけでなくコインの場合、**複製・破棄がルール上不可能**であり、**ミント権限 (Capability)** を持っていないことが客観的に判断できるので、形式上問題ないかを検証することができます。

```
module transfer_demo::utils {  
  use sui::coin::{Self as coin, Coin};  
  use sui::tx_context::TxContext;  
  
  public entry fun transfer<T>(  
    c: &mut Coin<T>, 何を (どのコインを)  
    amount: u64,       いくら  
    to: address,       誰に  
    ctx: &mut TxContext  
  ) {  
    // 新しいコインを生成するので ctx が必要  
    let part = coin::split(c, amount, ctx);  
    // 所有権の移動は ctx 不要  
    coin::transfer(part, to);  
  }  
}
```



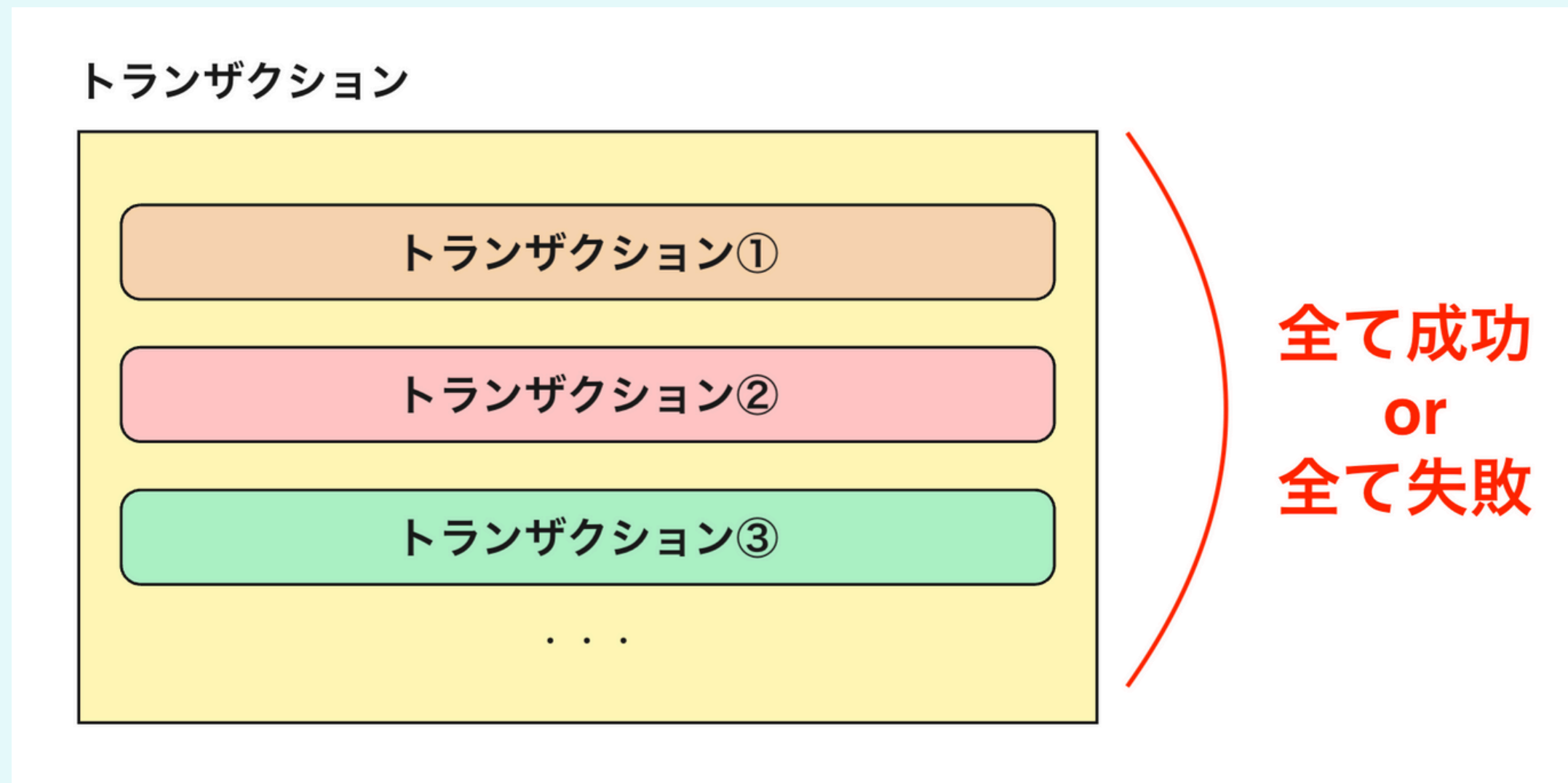
Build on Sui
Weekend Move Workshop
in Tokyo // Day 1

Powered by



1 PTBとは

PTB (programmable transaction block) はプログラム可能なトランザクションのブロックです。
まとめることで**原子性 (全て成功か失敗)** を確保できます。



1 PTBとは

また、大きな特徴として、**前の結果をそのまま次に渡せ**ます。

トランザクション

1 コインの分割



2 分割したコインの送付

1 PTBとは

こちらが使用できるコマンドです。

今回は「**TransferObjects**」と「**SplitCoins**」を使います。

The screenshot shows the Sui documentation page for Transaction type. The left sidebar contains a navigation menu with items like Overview, Sui Components, App Developers, Object Model, Move Overview, Transactions, Programmable Transaction Blocks, Sponsored Transactions, Gas Smashing, gRPC Overview (Beta), Gaming on Sui, GraphQL and Indexer Framework, Cryptography, Sui Architecture, Tokenomics, and Research Papers. The main content area is titled 'Transaction type' and explains that there are two parts of a PTB relevant to execution semantics. It shows a JSON structure for a PTB: { inputs: [Input], commands: [Command], }. Below this, it lists two main components: inputs and commands. The commands section is highlighted with a red box and lists several commands: TransferObjects, SplitCoins, MergeCoins, MakeMoveVec, MoveCall, Publish, and Upgrade. The description for SplitCoins mentions it can be any sui::coin::Coin<_> object.

Transaction type

There are two parts of a PTB that are relevant to execution semantics. Other transaction information, such as the transaction sender or the gas limit, might be referenced but are out of scope. The structure of a PTB is:

```
{
  inputs: [Input],
  commands: [Command],
}
```

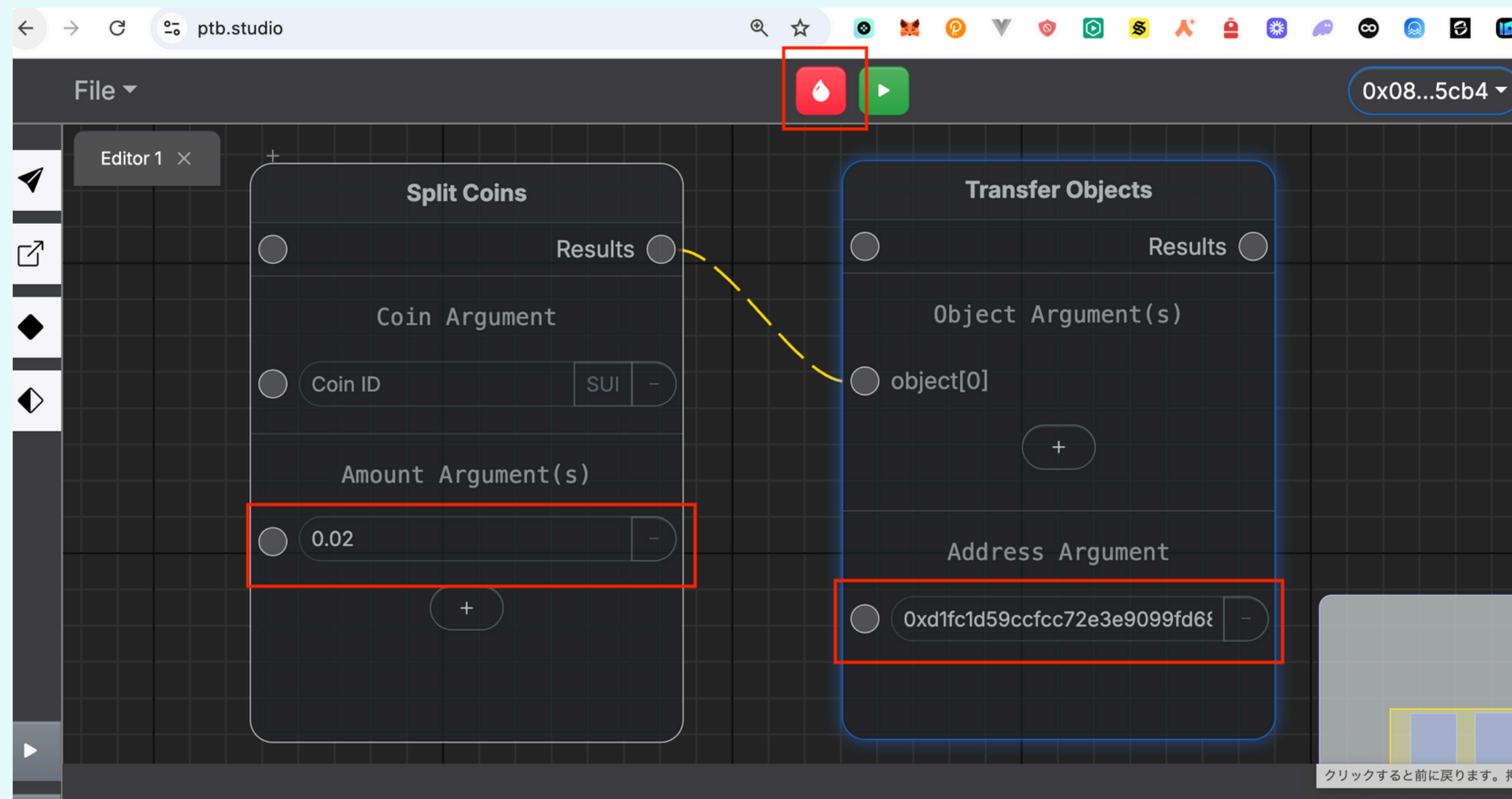
Looking closer at the two main components:

- The `inputs` value is a vector of arguments, `[Input]`. These arguments are either objects or pure values that you can use in the commands. The objects are either owned by the sender or are shared/immutable objects. The pure values represent simple Move values, such as `u64` or `String` values, which you can construct purely from their bytes. For historical reasons, `Input` is `CallArg` in the Rust implementation.
- The `commands` value is a vector of commands, `[Command]`. The possible commands are:
 - `TransferObjects` sends multiple (one or more) objects to a specified address.
 - `SplitCoins` splits off multiple (one or more) coins from a single coin. It can be any `sui::coin::Coin<_>` object.
 - `MergeCoins` merges multiple (one or more) coins into a single coin. Any `sui::coin::Coin<_>` objects can be merged, as long as they are all of the same type.
 - `MakeMoveVec` creates a vector (potentially empty) of Move values. This is used primarily to construct vectors of Move values to be used as arguments to `MoveCall`.
 - `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 upgrade is gated by the `sui::package::UpgradeCap` for that package.

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

1 PTBとは

ptb.studioで体験してみましょう。

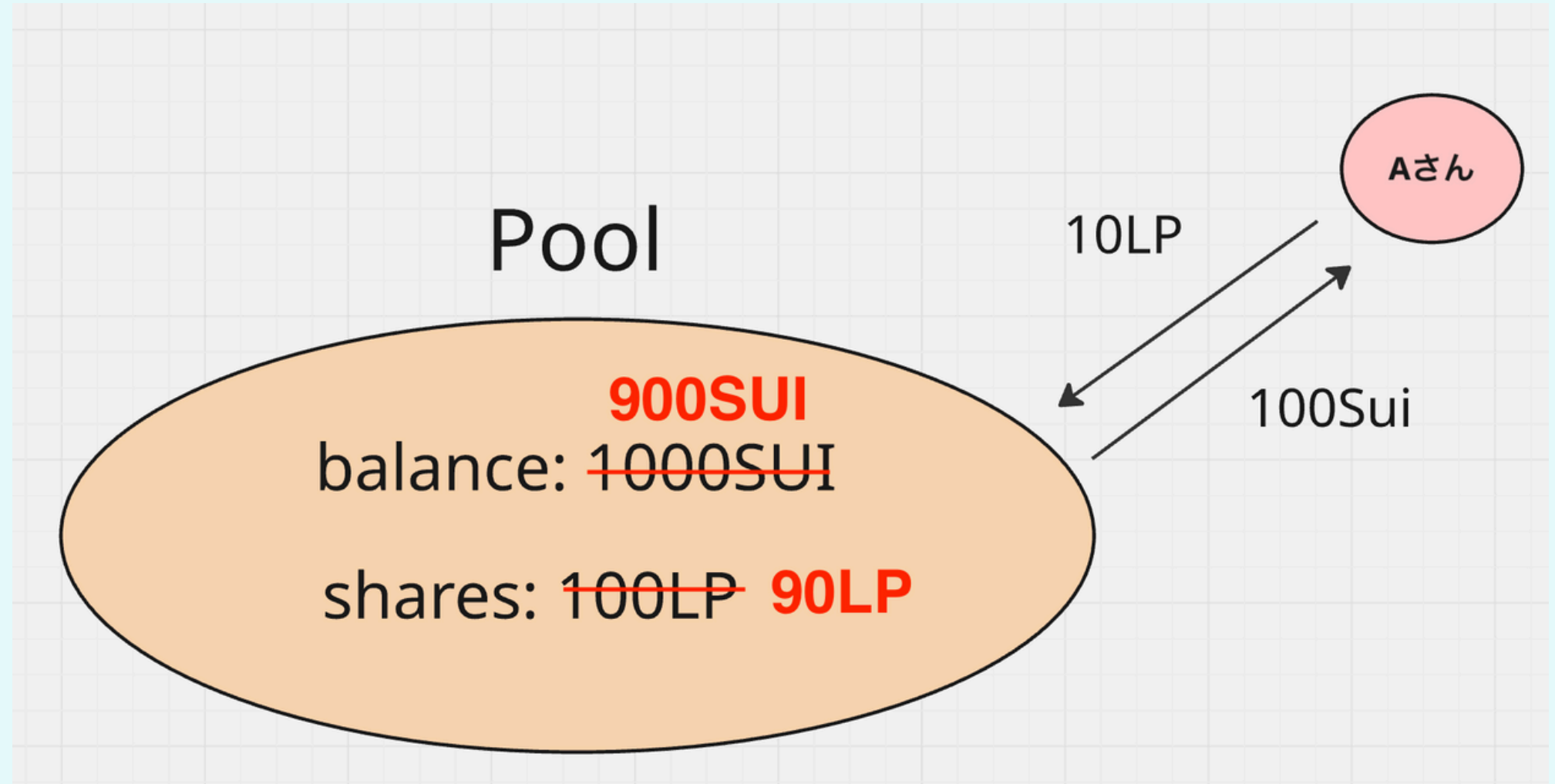


<https://ptb.studio/>

2 形式的検証 Sui Prover

プールからの引き出し（withdraw）について考えてみます。

例えば、所有している10LPを使用することで、その割合分である100SUIが取得できるとします。

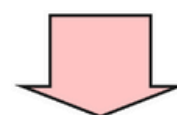


2 形式的検証 Sui Prover

今回は、引き出し後のLPの価値が引き出し前の価値より下がらないことを証明します。

割り算は使いたくないので、下のような掛け算の内容を証明します。

$$\frac{\text{引き出し前の総額 (1000SUI)}}{\text{引き出し前のシェア (100LP)}} \leq \frac{\text{引き出し後の総額 (900SUI)}}{\text{引き出し後のシェア (90LP)}}$$



$$\begin{aligned} & \text{引き出し前の総額 (1000SUI)} \times \text{引き出し後のシェア (90LP)} \\ & \leq \text{引き出し後の総額 (900SUI)} \times \text{引き出し前のシェア (100LP)} \end{aligned}$$

2 形式的検証 Sui Prover

以下のように、実行後に数学的に検証します。

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with folders like .vscode, chapter1, chapter2, chapter3, sui-prover/amm, and sources. The code editor shows the file simple_lp.move. The code is divided into two main sections: the implementation of the withdraw function and its formal verification. The implementation section (lines 12-26) is highlighted with a red box. The verification section (lines 31-46) is also highlighted with a red box. Within the verification section, the line `let result = withdraw(pool, shares_in);` is highlighted with a red box and labeled "処理の実行" (Execution of the process). The line `ensures(old_balance.mul(new_shares).lte(new_balance.mul(old_shares)));` is highlighted with a red box and labeled "形式的に検証" (Formal verification).

```
11
12 public fun withdraw<T>(pool: &mut Pool<T>, shares_in: Balance<LP<T>>): Balance<T> {
13     if (shares_in.value() == 0) {
14         shares_in.destroy_zero();
15         return zero()
16     };
17
18     let balance = pool.balance.value();
19     let shares = pool.shares.supply_value();
20
21     let balance_to_withdraw =
22         (((shares_in.value() as u128) * (balance as u128)) / (shares as u128)) as u64;
23
24     pool.shares.decrease_supply(shares_in);
25     pool.balance.split(balance_to_withdraw)
26 }
27
28 #[spec_only]
29 use prover::prover::{ensures, requires, old};
30
31 ...
32 #[spec(prove)]
33 fun withdraw_spec<T>(pool: &mut Pool<T>, shares_in: Balance<LP<T>>): Balance<T> {
34     // 前提
35     requires(shares_in.value() <= pool.shares.supply_value());
36     let old_pool = old!(pool);
37     let result = withdraw(pool, shares_in);
38     // 検証
39     let old_balance = old_pool.balance.value().to_int();
40     let new_balance = pool.balance.value().to_int();
41     let old_shares = old_pool.shares.supply_value().to_int();
42     let new_shares = pool.shares.supply_value().to_int();
43     ensures(old_balance.mul(new_shares).lte(new_balance.mul(old_shares)));
44
45     result
46 }
```

https://github.com/SuiJapan/core-workshop/blob/main/chapter3/sui-prover/amm/sources/simple_lp.move#L31

2 形式的検証 Sui Prover

興味がありましたら、ぜひやってみてください



https://www.youtube.com/watch?v=wE_fu-nUdDE

3 Clockオブジェクト

ポイント

- ①引数のClockは不変で渡す
- ②時間の取得はclock.timestamp_ms()
- ③Clockのアドレスは0x6

examples/move/basics/sources/clock.move

```
module basics::clock;

use sui::clock::Clock;
use sui::event;

public struct TimeEvent has copy, drop, store {
    timestamp_ms: u64,
}

entry fun access(clock: &Clock) {
    event::emit(TimeEvent { timestamp_ms: clock.timestamp_ms() });
}
```

A call to the previous entry function takes the following form, passing 0x6 as the address for the Clock parameter:

<https://docs.sui.io/guides/developer/sui-101/access-time>

3 Clockオブジェクト

実際に処理を行ってみましょう。

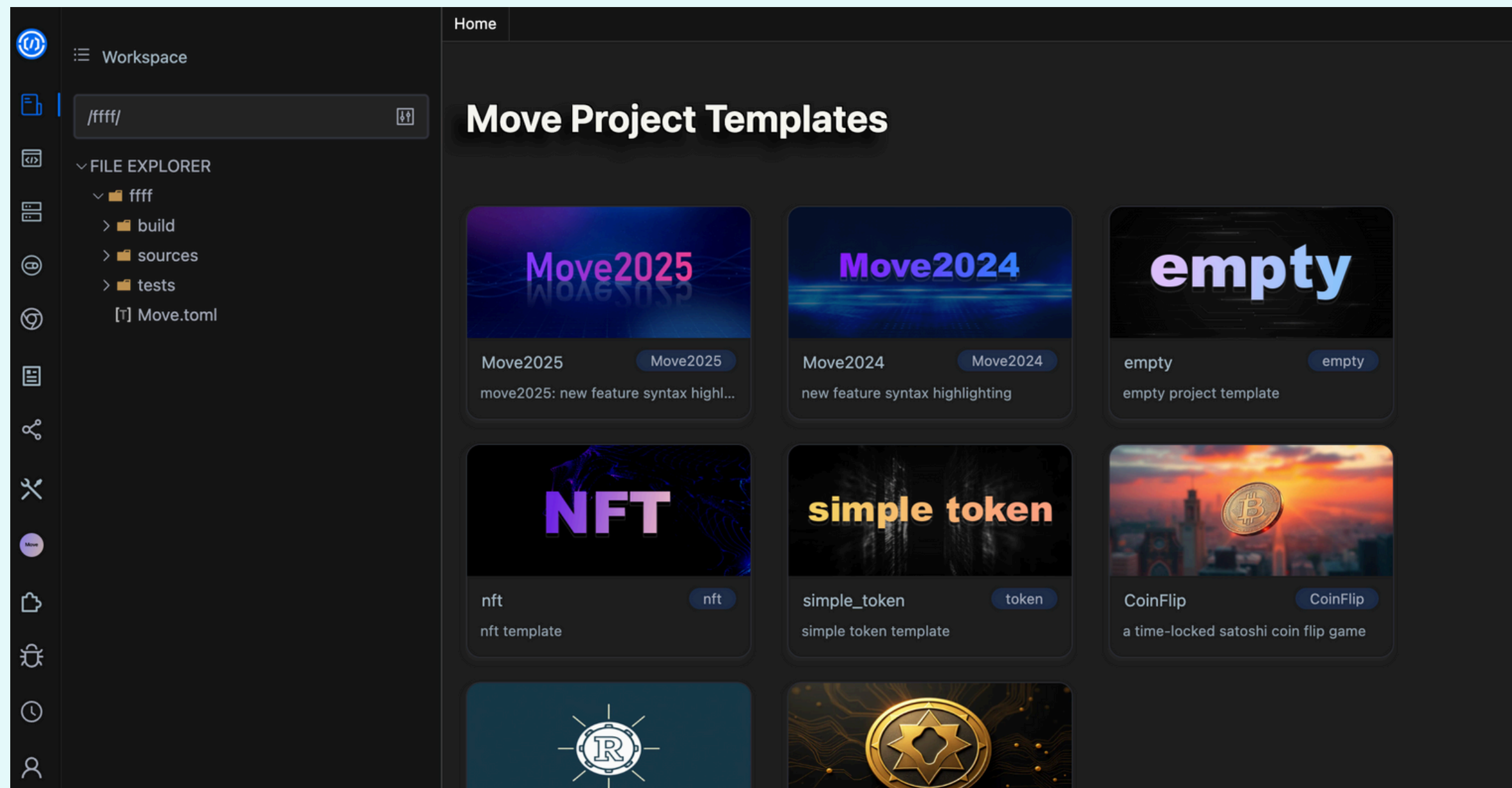
The screenshot displays the Suivision testnet interface. At the top, the 'Details' tab shows package information: Publisher (0x917b...bb2f), Publish Time (Oct 18, 2025 23:20:52 +UTC), Last Transaction ID (EGEhyDtreauoPRVZKj65agQ3eZGrdXgbtzw15BFBch71), Version (1), and Owner (Immutable). Below this, the 'Code' tab is active, showing the source code for a package named 'clock_example'. The code defines a 'TimeEvent' struct and an 'access' function. A red oval highlights the 'access' function definition. To the right of the code, the 'Execute' section shows the function signature 'access' with an argument 'Arg0' set to '0xd'. An 'Execute' button is located at the bottom right of the code editor.

```
1 module 0x5279c3e5d5bc5d1050d4c4ed469e734d47dbf1f0da4086f8c872c8bb1c0eb0f4::clock_example {
2   struct TimeEvent has copy, drop, store {
3     timestamp_ms: u64,
4   }
5
6   entry fun access(arg0: &0x2::clock::Clock) {
7     let v0 = TimeEvent{timestamp_ms: 0x2::clock::timestamp_ms(arg0)};
8     0x2::event::emit<TimeEvent>(v0);
9   }
10
11   // decompiled from Move bytecode v6
12 }
13
```

<https://testnet.suivision.xyz/package/0x5279c3e5d5bc5d1050d4c4ed469e734d47dbf1f0da4086f8c872c8bb1c0eb0f4?tab=Code>

3 Clockオブジェクト

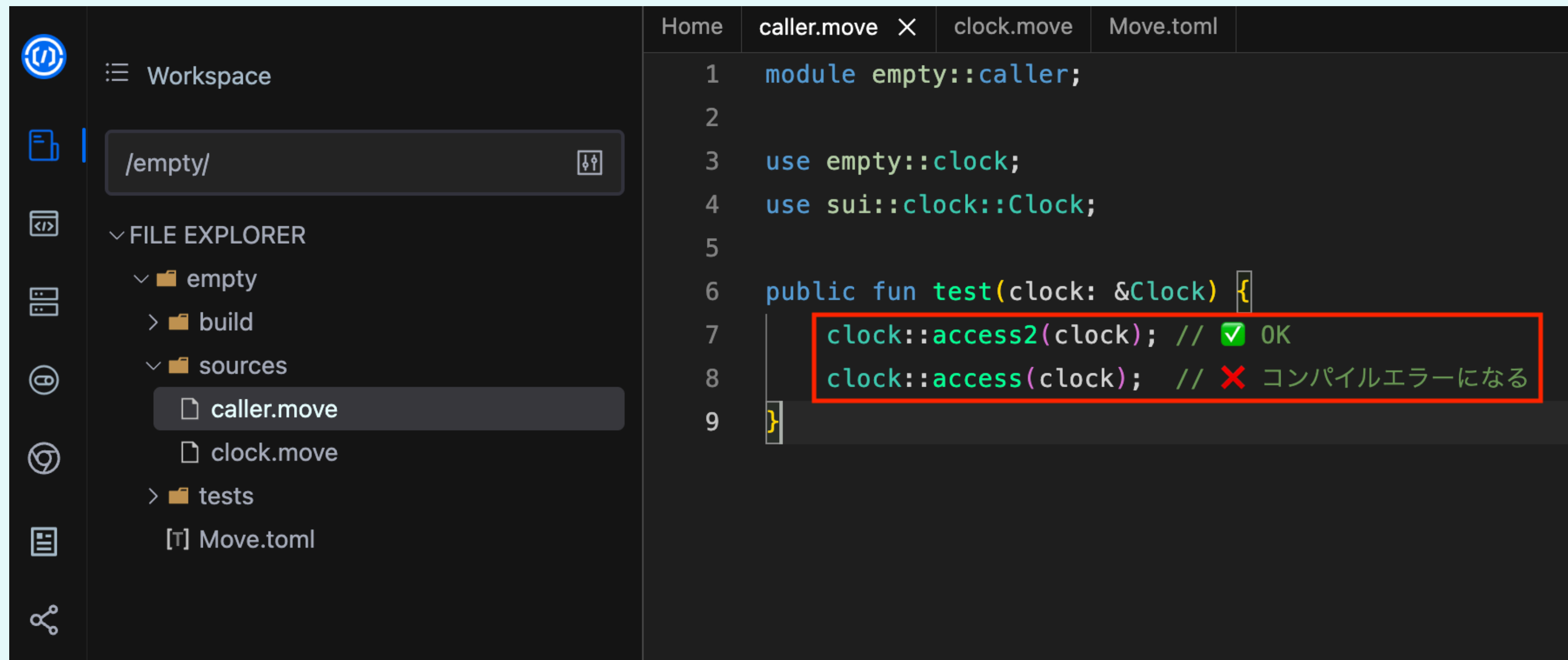
Bits Labを使ってみましょう。



<https://ide.bitslab.xyz/>

3 Clockオブジェクト

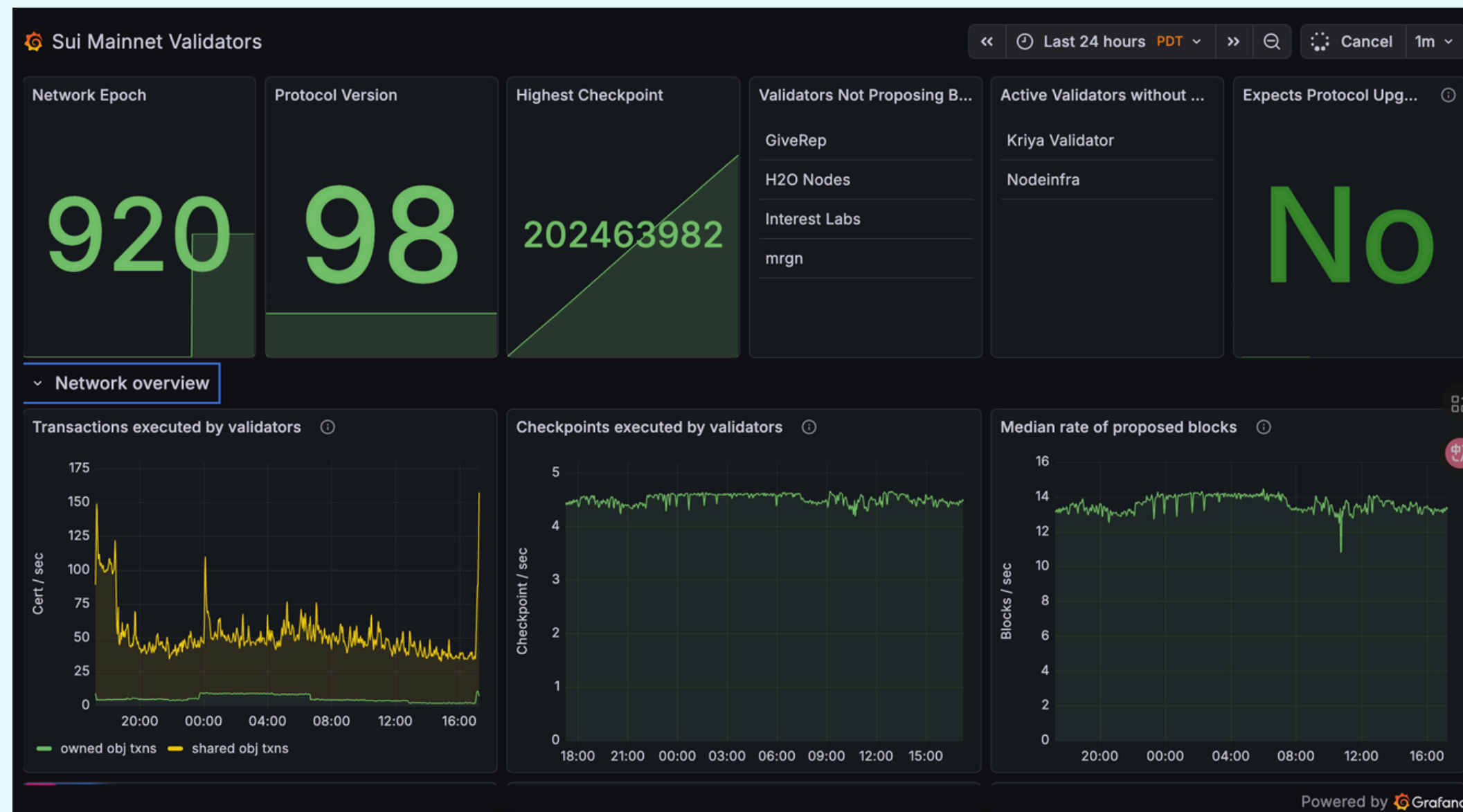
entryとpublicの違いを体験しましょう。



```
1 module empty::clock;
2
3 use empty::clock;
4 use sui::clock::Clock;
5
6 public fun test(clock: &Clock) {
7     clock::access2(clock); // ✓ OK
8     clock::access(clock);  // ✗ コンパイルエラーになる
9 }
```


3 Clockオブジェクト

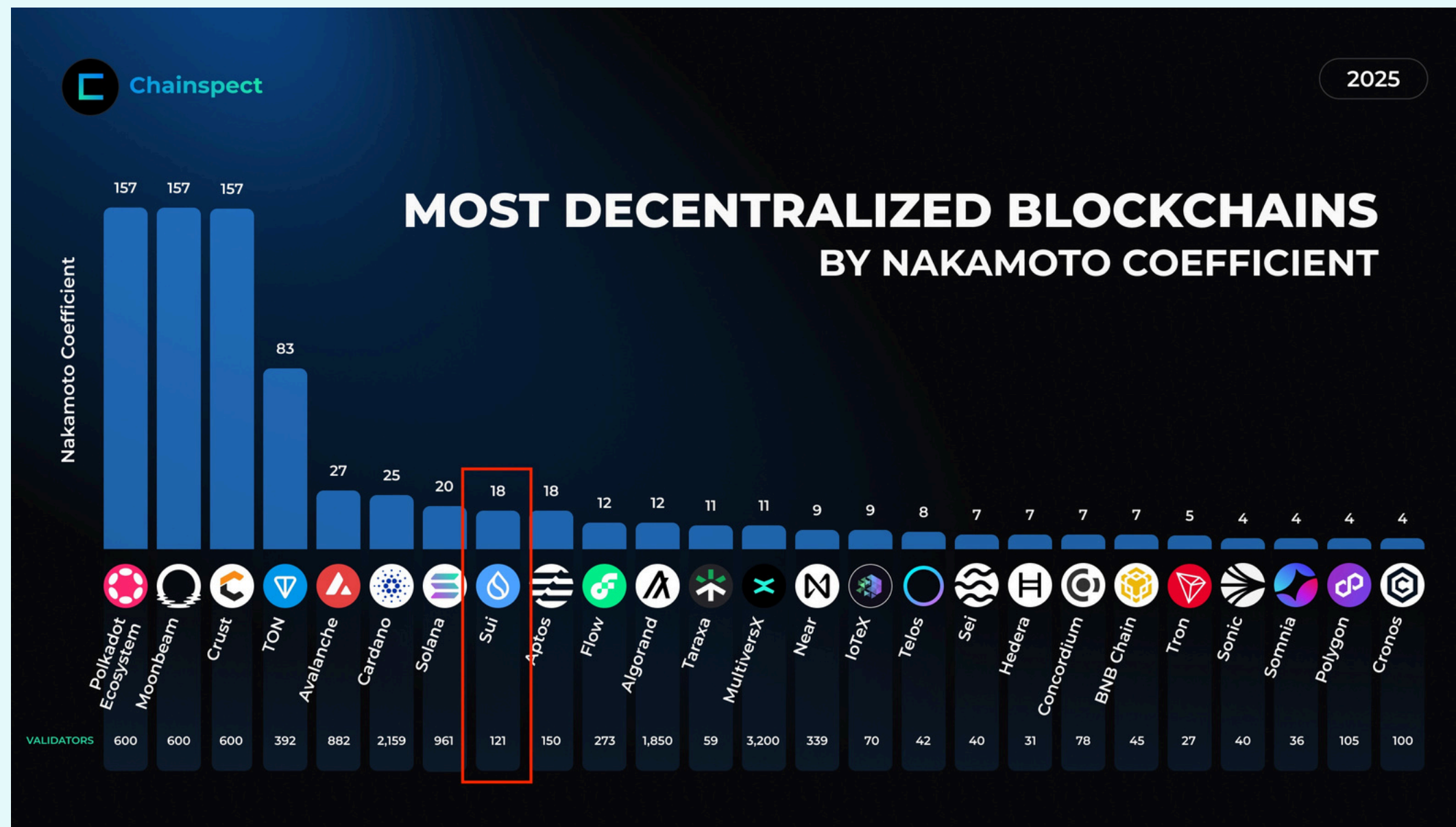
せっかくなので色々な指標を見てみましょう。



https://metrics.sui.io/public-dashboards/4ceb11cc210d4025b122294586961169?from=now-24h&to=now&timezone=America%2FLos_Angeles

余談 ナカモト指数について

何ノードが協力すれば、ネットワークを支配（50%）できるか？を示す数字です。



https://x.com/chainspect_app/status/1978096780580364414/photo/1

余談 ナカモト指数について

Suiの分散状況も見てください。



[https://metrics.sui.io/public-dashboards/4ceb11cc210d4025b122294586961169?
from=now-24h&to=now&timezone=America%2FLos_Angeles](https://metrics.sui.io/public-dashboards/4ceb11cc210d4025b122294586961169?from=now-24h&to=now&timezone=America%2FLos_Angeles)

余談 ナカモト指数について

ちなみに、Etehreumは2です。

これは「Lido Finance」と「Coinbase」が大きなシェアを持つためです。

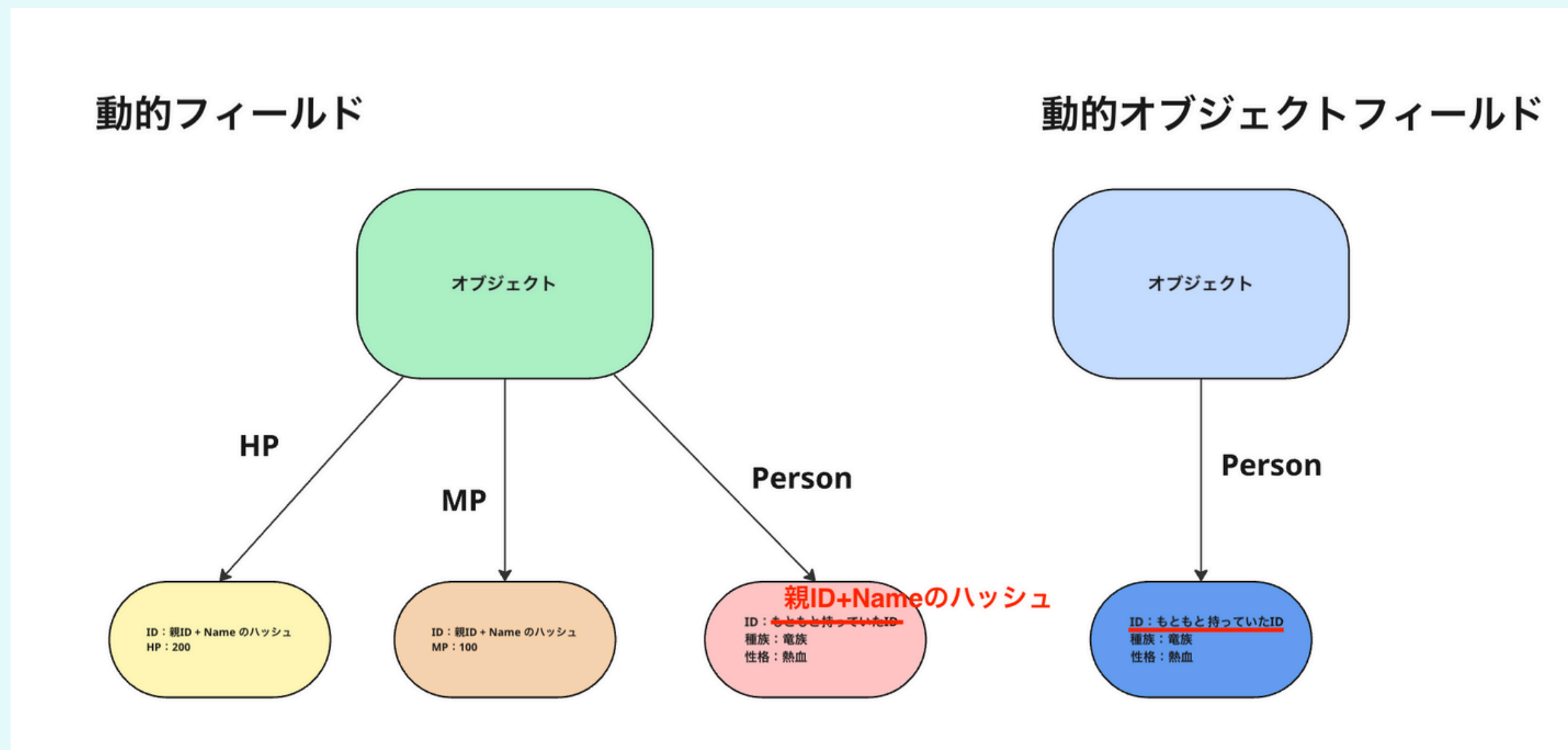
Data from [Chainspect](#) | Oct 19, 2025, 00:29 UTC

#	Name	Nakamoto Coefficient	Validators/Miners	Stake/Hashrate	Consensus Mechanism
30	Layer 1		↓ 2.86%	↓ 16.38%	Proof of Work (PoW)
31	Wanchain Layer 1	2	32	\$1.52M ↑ 0.88%	Proof of Stake (PoS)
32	Kaia Layer 1	2	40	\$249M ↑ 1.48%	Proof of Stake (PoS)
33	Ethereum Layer 1	2	1,021,000 ↓ 0.15%	\$138.6B ↑ 3.27%	Proof of Stake (PoS)
34	eCash Layer 1	2	16	0.0782 EH/s ↑ 2.76%	Proof of Work (PoW)
35	Starknet Layer 2	1	1	—	Rollup (ZK)
36	Scroll Layer 2	1	1	—	Rollup (ZK)
37	Optimism Layer 2	1	1	—	Rollup (Optimistic)

<https://chainspect.app/dashboard/decentralization>

4 動的（オブジェクト）フィールド

Suiではオブジェクトに動的にフィールドの設定ができます。
2種類がありますが、どちらもオブジェクトとして付属させます。



https://github.com/ytakahashi2020/dynamic_field/blob/main/my/sources/dynamic_field_demo.move

4 動的（オブジェクト） フィールド

例えば、下のオブジェクトは「HP」,「MP」というフィールドを持ち、それぞれがオブジェクトです。

Object: 0xd66d...d385

Search by Account, Package, Object, Transaction, SuiNS

Details

Owner: 0x0863...5cb4
Publisher: 0x0863...5cb4
Type: 0xe051...fb8b::simple...demo::Parent
Version: 619109403
Transaction Block Digest: 46wts7...QGp7

Fields

```
{  
  id: {  
    id: "0xd66dbd412bbefef054549ca2bc650a23782ec0ed552948de515f983c2aacd385"  
  }  
}
```

Dynamic Fields

CHILD OBJECT ID	VERSION	OBJECT TYPE	LAST TX ID	
0x214f...9e1f	619109403	u64	46wts7...QGp7	HPフィールド
0x8587...9828	619109402	u64	DetrjF...axQP	MPフィールド

Object: 0x214f...9e1f

Search by Account, Package, Object, Transaction, SuiNS

Details

Owner: 0xd66d...d385
Publisher: 0x0863...5cb4
Type: 0x2::dynamic_field::Field<vector<u8>, u64>
Version: 619109403
Transaction Block Digest: 46wts7...QGp7

Fields

```
{  
  id: {  
    id: "0x214fba91545b10bdd1a860dfb386475c475af57b99a699d6ee151328b8cb9e1f"  
  }  
  name: {  
    0: 104 H  
    1: 112 P  
  }  
  value: "200"  
}
```

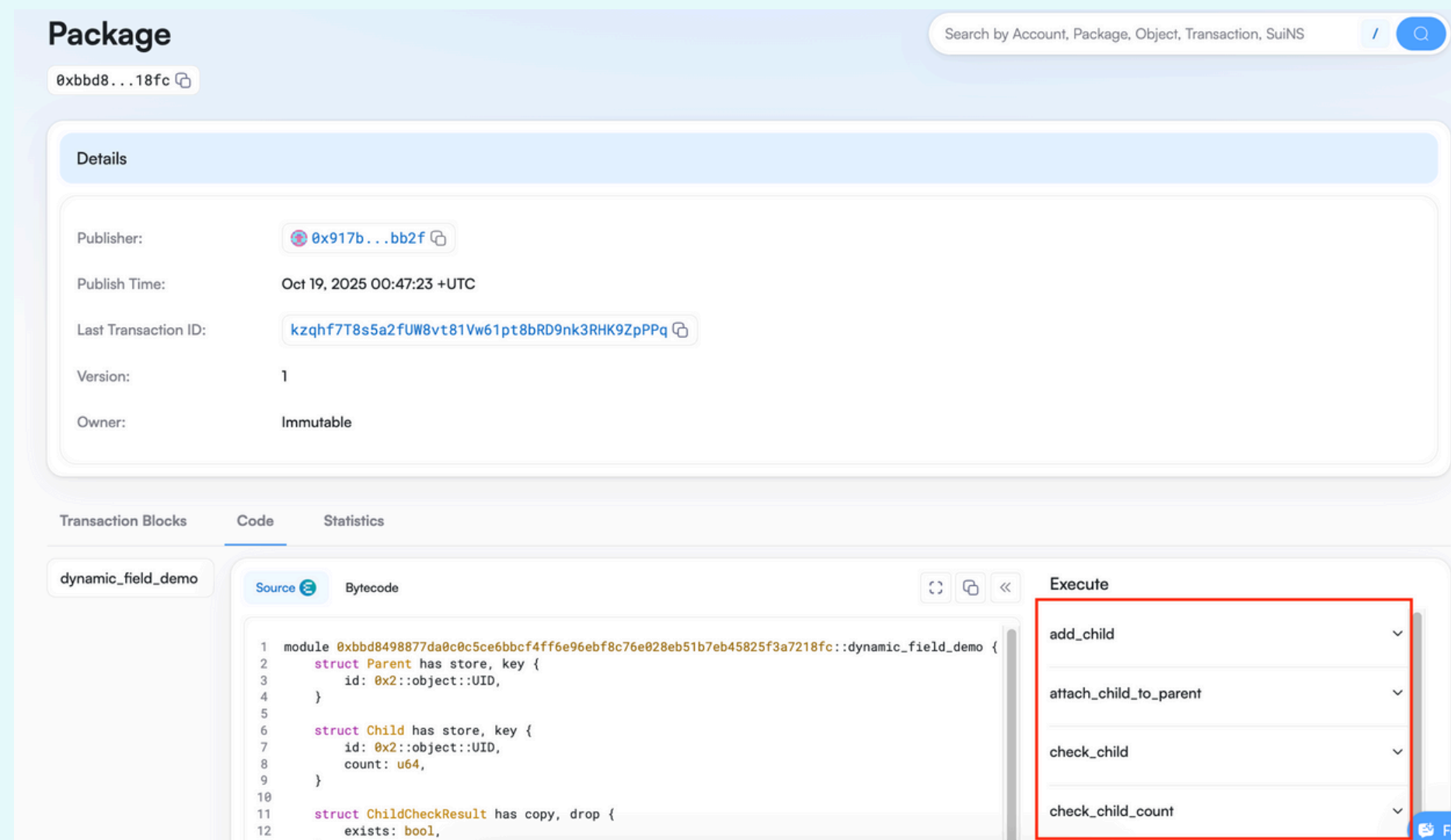
Dynamic Fields

CHILD OBJECT ID	VERSION	OBJECT TYPE	LAST TX ID
-----------------	---------	-------------	------------

<https://testnet.suivision.xyz/object/0xa8fc841714fef0a9dcabdca7ba7c98a678359180a6984bfcc4a552913d2d93e0f>

4-1 動的フィールド

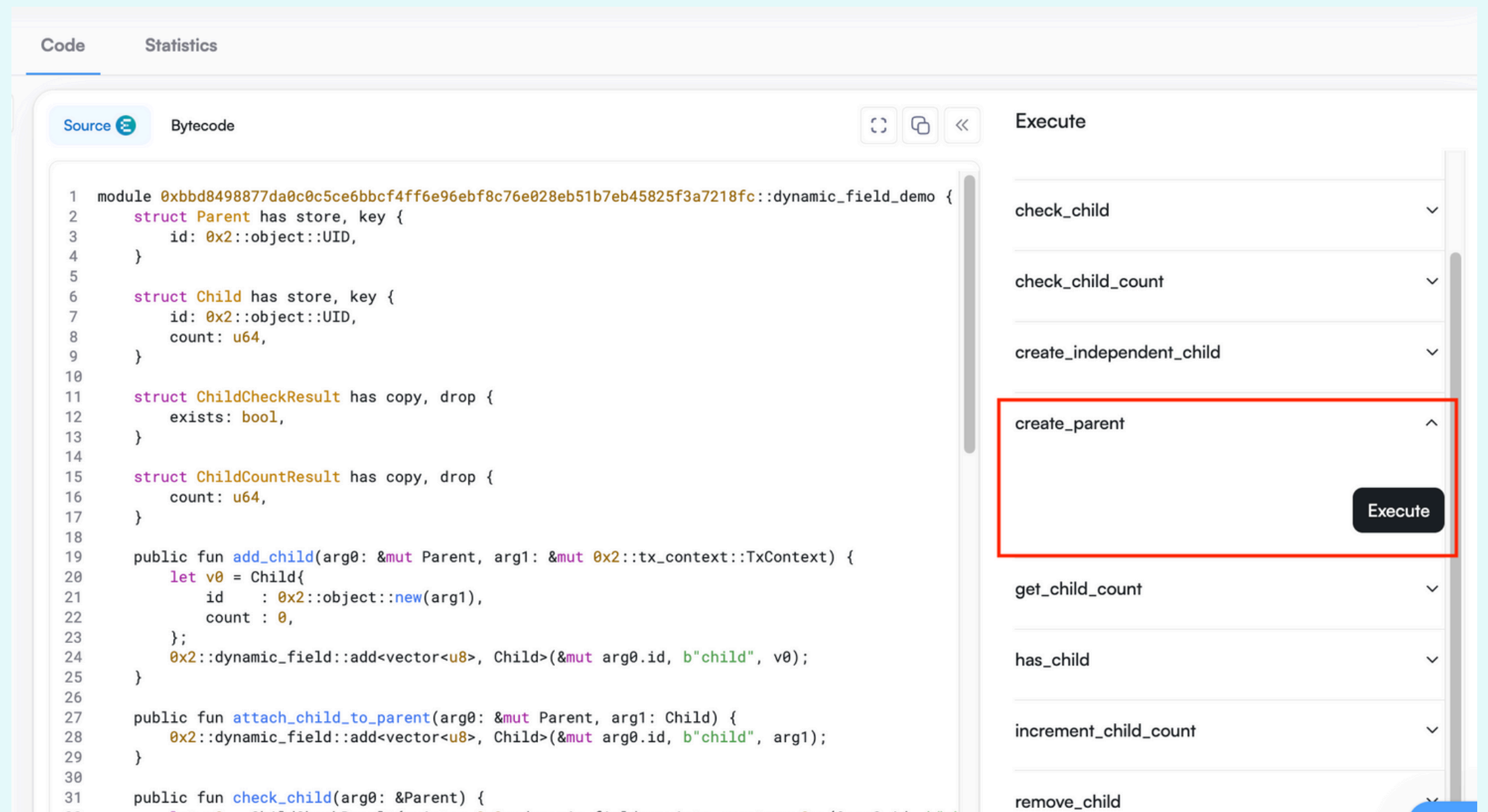
なお、動的フィールドの場合、新しくIDが生成されます。
そのため、すでにあるオブジェクトを子にした場合、元のIDからアクセスできず、親を通じてのみのアクセスとなります。



4-1 動的フィールド

まずは親のオブジェクトを作成します。

0x9a...fd

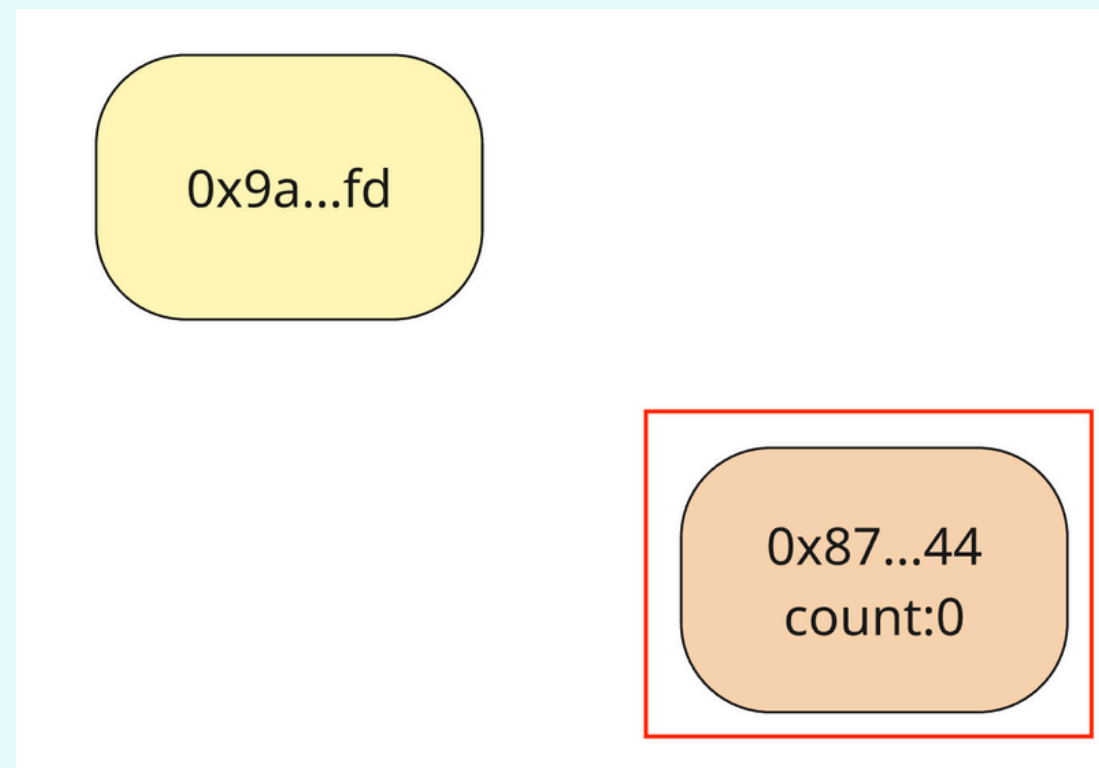


```
1 module 0xbbd8498877da0c0c5ce6bbcf4ff6e96ebf8c76e028eb51b7eb45825f3a7218fc::dynamic_field_demo {
2     struct Parent has store, key {
3         id: 0x2::object::UID,
4     }
5
6     struct Child has store, key {
7         id: 0x2::object::UID,
8         count: u64,
9     }
10
11     struct ChildCheckResult has copy, drop {
12         exists: bool,
13     }
14
15     struct ChildCountResult has copy, drop {
16         count: u64,
17     }
18
19     public fun add_child(arg0: &mut Parent, arg1: &mut 0x2::tx_context::TxContext) {
20         let v0 = Child{
21             id: 0x2::object::new(arg1),
22             count: 0,
23         };
24         0x2::dynamic_field::add<vector<u8>, Child>(&mut arg0.id, b"child", v0);
25     }
26
27     public fun attach_child_to_parent(arg0: &mut Parent, arg1: Child) {
28         0x2::dynamic_field::add<vector<u8>, Child>(&mut arg0.id, b"child", arg1);
29     }
30
31     public fun check_child(arg0: &Parent) {
32         let v0 = ChildCheckResult{exists: 0x2::dynamic_field::exists_vector_u8(&arg0.id, b"child")};
33     }
34 }
```

<https://testnet.suivision.xyz/package/0xbbd8498877da0c0c5ce6bbcf4ff6e96ebf8c76e028eb51b7eb45825f3a7218fc?tab=Code>

4-1 動的フィールド

次に別のオブジェクトを作ります。この時点では関係性はありません。



Transaction Blocks Code Statistics

dynamic_field_demo

Source Bytecode

```
1 module 0xbdd8498877da0c0c5ce6bbcf4ff6e96ebf8c76e028eb51b7eb45825f3a7218fc::dynamic_field_demo {
2   struct Parent has store, key {
3     id: 0x2::object::UID,
4   }
5
6   struct Child has store, key {
7     id: 0x2::object::UID,
8     count: u64,
9   }
10
11  struct ChildCheckResult has copy, drop {
12    exists: bool,
13  }
14
15  struct ChildCountResult has copy, drop {
16    count: u64,
17  }
18
19  public fun add_child(arg0: &mut Parent, arg1: &mut 0x2::tx_context::TxContext) {
20    let v0 = Child{
21      id : 0x2::object::new(arg1),
22      count : 0,
23    };
24    0x2::dynamic_field::add<vector<u8>, Child>(&mut arg0.id, b"child", v0);
25  }
26
27  public fun attach_child_to_parent(arg0: &mut Parent, arg1: Child) {
28    0x2::dynamic_field::add<vector<u8>, Child>(&mut arg0.id, b"child", arg1);
29  }
30
```

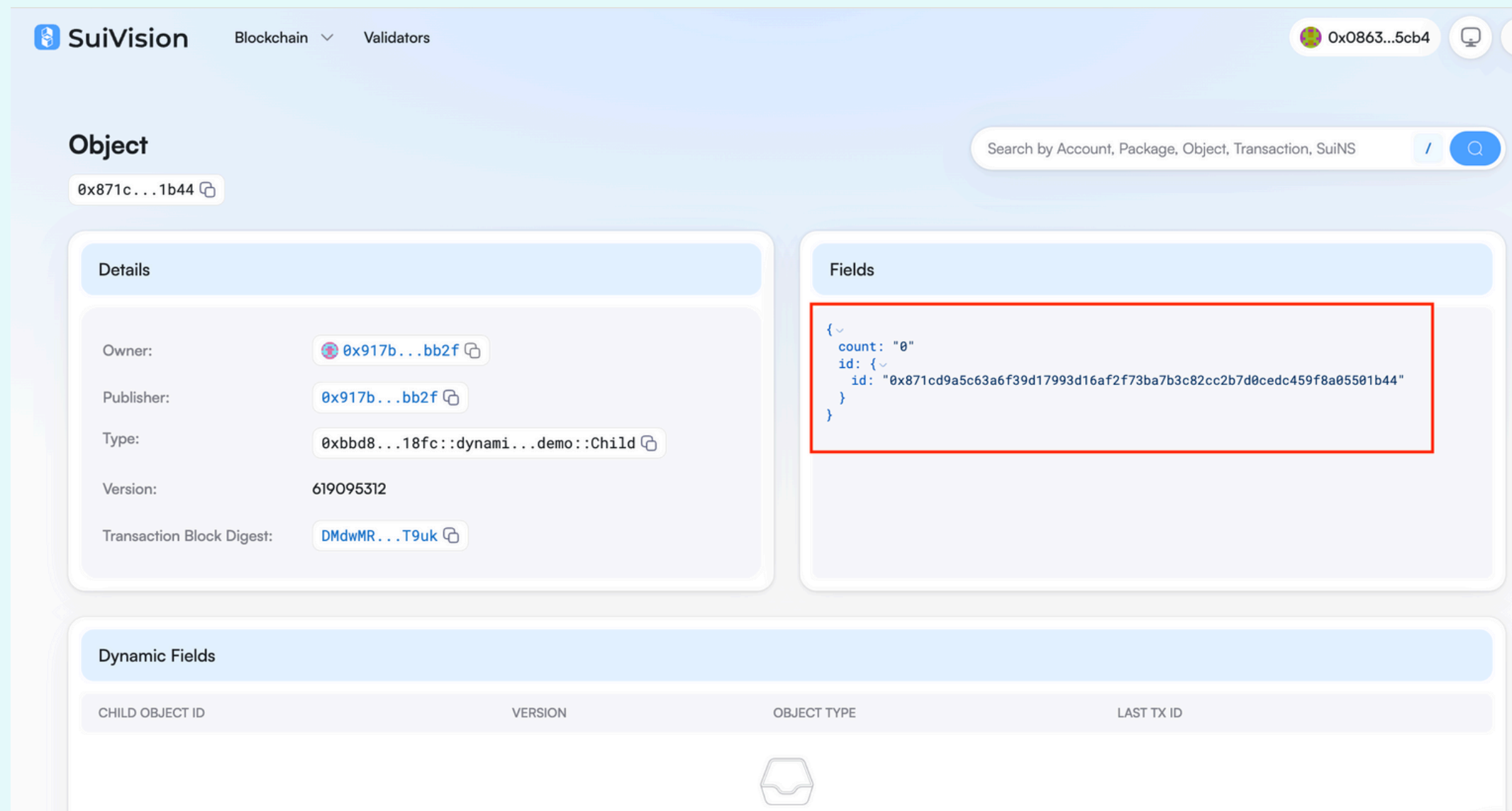
Execute

- add_child
- attach_child_to_parent
- check_child
- check_child_count
- create_independent_child
- create_parent
- get_child_count
- has_child

Execute

4-1 動的フィールド

作成したオブジェクトにアクセスできることを確認します。



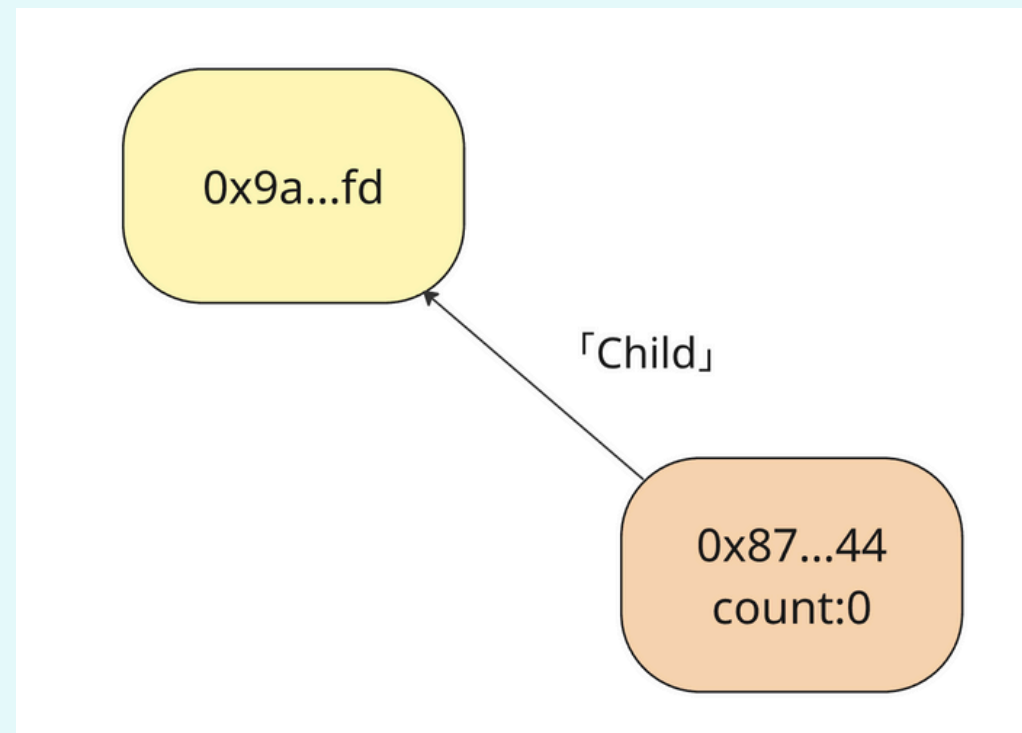
The screenshot displays the SuiVision web application interface. At the top, there's a navigation bar with 'SuiVision', 'Blockchain', and 'Validators' tabs. A search bar on the right allows searching by Account, Package, Object, Transaction, or SuiNS. The main content area is titled 'Object' and shows details for a specific object ID: 0x871c...1b44. The 'Details' section lists fields such as Owner (0x917b...bb2f), Publisher (0x917b...bb2f), Type (0xbbd8...18fc::dynam...demo::Child), Version (619095312), and Transaction Block Digest (DMdwMR...T9uk). The 'Fields' section shows a JSON object with 'count' set to 0 and an 'id' field containing the object's full ID. A red box highlights this JSON object. Below the details, there's a 'Dynamic Fields' section with a table header: CHILD OBJECT ID, VERSION, OBJECT TYPE, and LAST TX ID. The bottom of the page features a Sui logo.

CHILD OBJECT ID	VERSION	OBJECT TYPE	LAST TX ID
-----------------	---------	-------------	------------

<https://testnet.suivision.xyz/object/0x871cd9a5c63a6f39d17993d16af2f73ba7b3c82ccb7d0cedc459f8a05501b44>

4-1 動的フィールド

では子オブジェクトを「child」という名前で紐づけましょう。



Transaction details:

- Publisher: 0x917b...bb2f
- Publish Time: Oct 19, 2025 00:47:23 +UTC
- Last Transaction ID: kzqhf7T8s5a2fUW8vt81Vw61pt8bRD9nk3RHK9ZpPPq
- Version: 1
- Owner: Immutable

Transaction Blocks | Code | Statistics

dynamic_field_demo

Source | Bytecode

```
1 module 0xbbd8498877da0c0c5ce6bbcf4ff6e96ebf8c76e028eb51b7eb45825f3a7218fc::dynamic_field_demo {
2   struct Parent has store, key {
3     id: 0x2::object::UID,
4   }
5   struct Child has store, key {
6     id: 0x2::object::UID,
7     count: u64,
8   }
9   struct ChildCheckResult has copy, drop {
10    exists: bool,
11  }
12  struct ChildCountResult has copy, drop {
13    count: u64,
14  }
15  public fun add_child(arg0: &mut Parent, arg1: &mut 0x2::tx_context::TxContext) {
16    let v0 = Child{
17      id: 0x2::object::UID,
18      count: 0,
19    }
20    attach_child_to_parent(arg0, v0)
21  }
```

Execute

add_child

attach_child_to_parent

Arg0 親

0x9efd67b460d83c228c8e2a4f8cde76ef8fd9dd

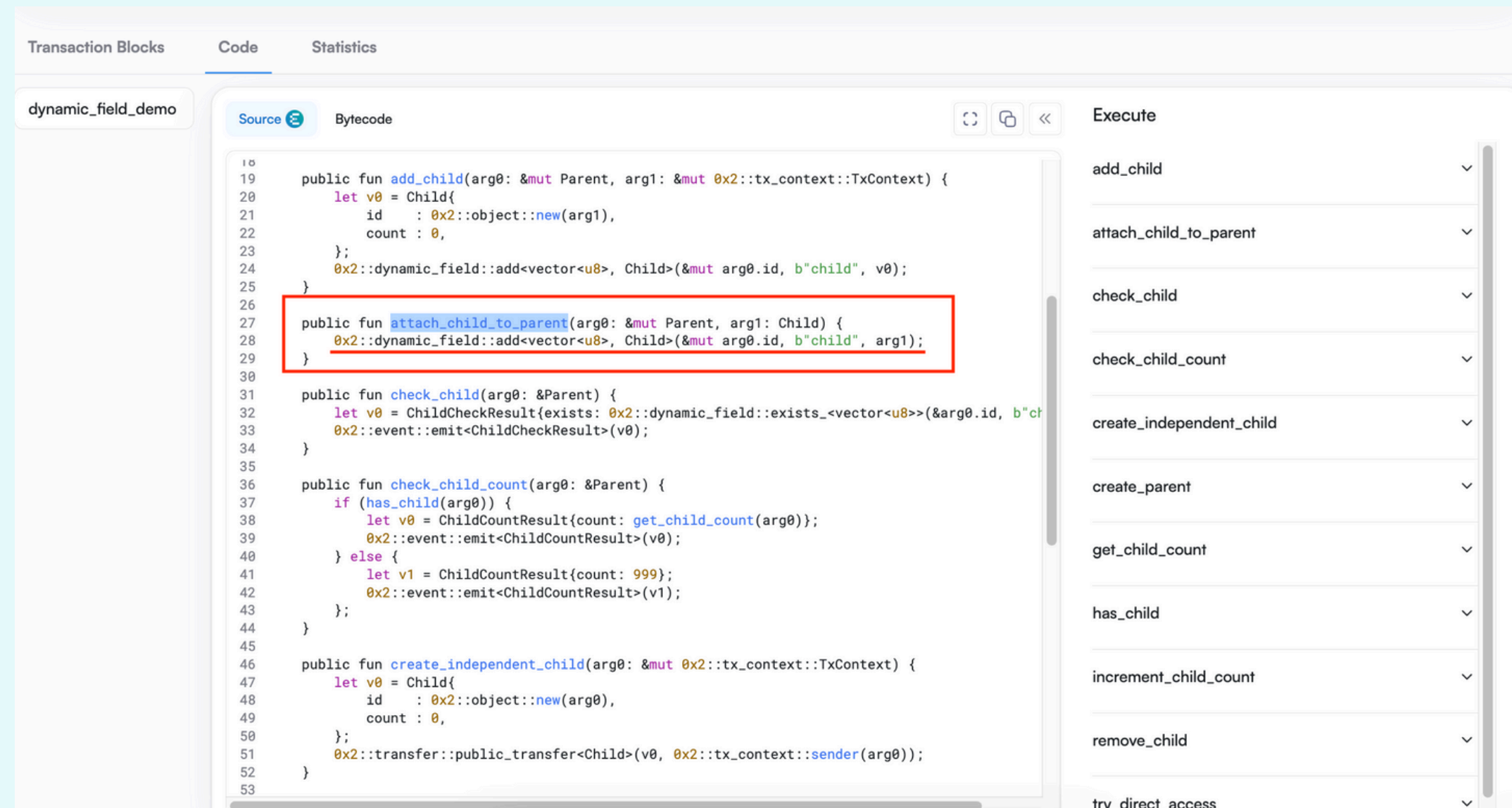
Arg1 子

0x325a49d4702b7f76f72f2f3bf2d161907ac1e48c

Execute

4-1 動的フィールド

動的フィールドを作成し、子のオブジェクトが直接参照できないことを確認しましょう。



```
18
19 public fun add_child(arg0: &mut Parent, arg1: &mut 0x2::tx_context::TxContext) {
20     let v0 = Child{
21         id : 0x2::object::new(arg1),
22         count : 0,
23     };
24     0x2::dynamic_field::add<vector<u8>, Child>(&mut arg0.id, b"child", v0);
25 }
26
27 public fun attach_child_to_parent(arg0: &mut Parent, arg1: Child) {
28     0x2::dynamic_field::add<vector<u8>, Child>(&mut arg0.id, b"child", arg1);
29 }
30
31 public fun check_child(arg0: &Parent) {
32     let v0 = ChildCheckResult{exists: 0x2::dynamic_field::exists<vector<u8>>(&arg0.id, b"ch
33     0x2::event::emit<ChildCheckResult>(v0);
34 }
35
36 public fun check_child_count(arg0: &Parent) {
37     if (has_child(arg0)) {
38         let v0 = ChildCountResult{count: get_child_count(arg0)};
39         0x2::event::emit<ChildCountResult>(v0);
40     } else {
41         let v1 = ChildCountResult{count: 999};
42         0x2::event::emit<ChildCountResult>(v1);
43     };
44 }
45
46 public fun create_independent_child(arg0: &mut 0x2::tx_context::TxContext) {
47     let v0 = Child{
48         id : 0x2::object::new(arg0),
49         count : 0,
50     };
51     0x2::transfer::public_transfer<Child>(v0, 0x2::tx_context::sender(arg0));
52 }
53 }
```

<https://testnet.suivision.xyz/package/0xbbd8498877da0c0c5ce6bbcf4ff6e96ebf8c76e028eb51b7eb45825f3a7218fc?tab=Code>

4-1 動的フィールド

子オブジェクトにアクセスしようとすると、できないことがわかります。

The screenshot shows the SuiVision interface. At the top, there's a header with the SuiVision logo, 'Blockchain' dropdown, 'Validators' dropdown, and a user profile '0x0863...5cb4'. Below the header, a yellow warning box with a red border contains the message: 'This object was deleted. You can view its history here.' Below this, there's a section titled 'Transaction Blocks'. Under this section, there are two tabs: 'Updated Objects' (selected) and 'Input Objects'. Below the tabs is a table with the following columns: DIGEST, PAYLOAD, SENDER, TXNS, STATUS, AGE, and GAS. The table contains one row of data:

DIGEST	PAYLOAD	SENDER	TXNS	STATUS	AGE	GAS
DMdwMR...T9uk	0xbbd8...18fc::dynami...demo::create...hild	0x917b...bb2f	1	✓	11 mins 54 secs ago	0.00243868 SUI

At the bottom of the interface, there is an 'End' button.

<https://testnet.suivision.xyz/object/0x871cd9a5c63a6f39d17993d16af2f73ba7b3c82cc2b7d0cedc459f8a05501b44>

4-1 動的フィールド

親オブジェクトを確認すると動的フィールドがあることが確認できます。

Version: 619095309

Transaction Block Digest: GnDYEh...yTx1

Dynamic Fields

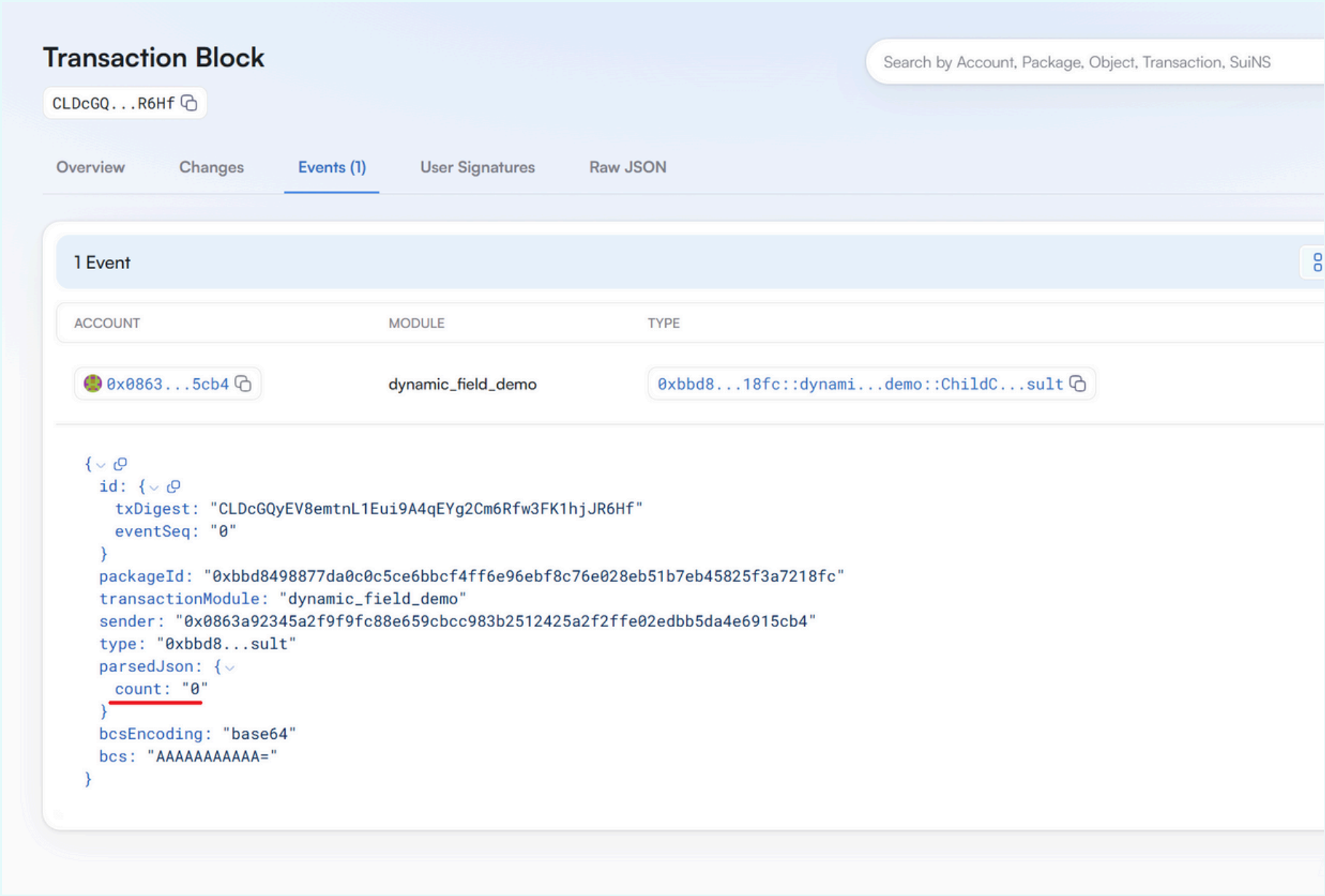
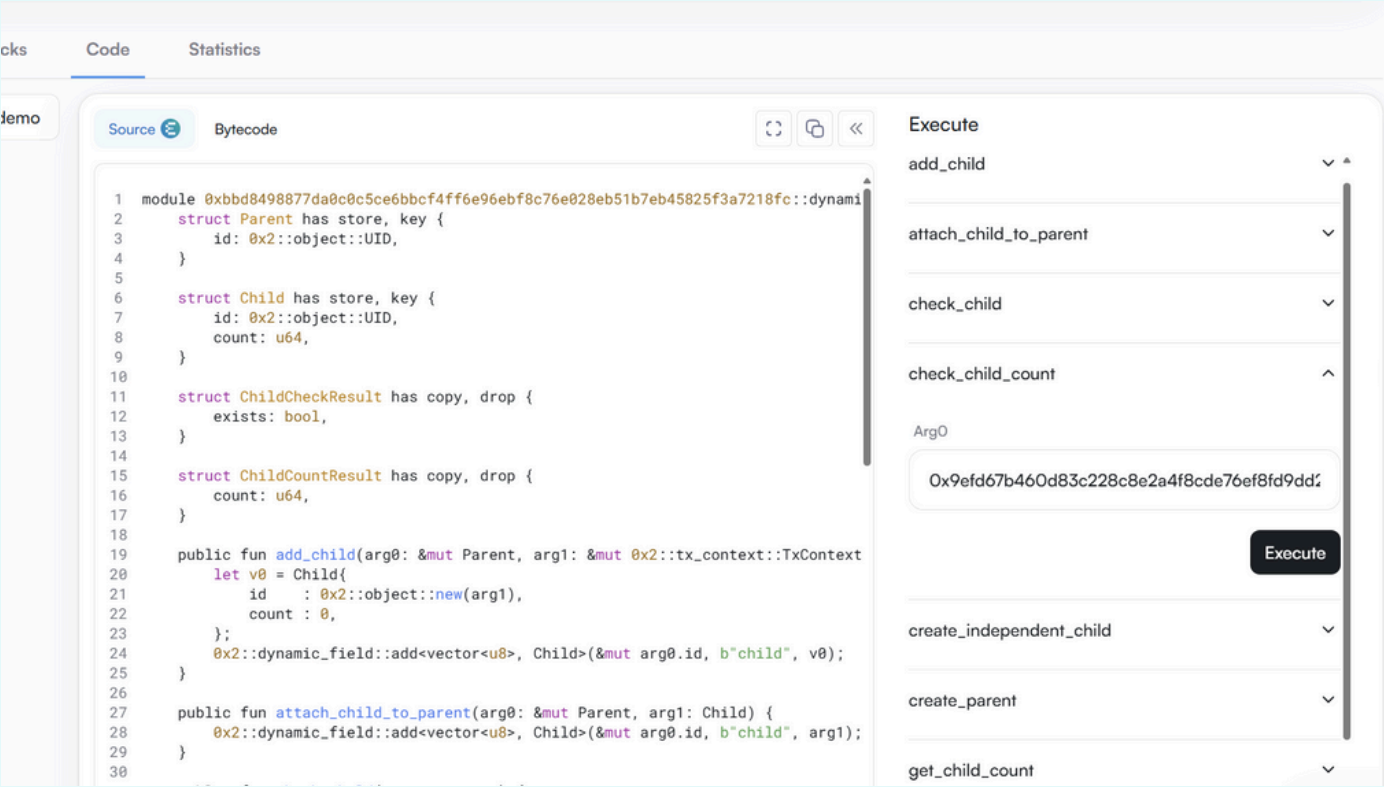
CHILD OBJECT ID	VERSION	OBJECT TYPE	LAST TX ID
0x09e8...3412	619,095,309	0xbbd8...18fc::dynami...demo::Child	GnDYEh...yTx1

```
{
  name: {
    type: "vector<u8>"
    value: [
      0: 99 C
      1: 104 h
      2: 105 i
      3: 108 l
      4: 100 d
    ]
  }
  bcsEncoding: "base64"
  bcsName: "BWNNoaWxk"
  type: "DynamicField"
  objectType: "0xbbd8498877da0c0c5ce6bbcf4ff6e96ebf8c76e028eb51b7eb45825f3a7218fc::dynamic_field_demo::Child"
  objectId: "0x09e873def42ae80ceff8a7af82e5c33a5bc793c2742d38fa6d62180e0b5b3412"
  version: 619095309
  digest: "GxM1GEKhtMnNLfLbzZwfWNCED5pV9ZtxPE8KAFkQNhtj"
}
```

<https://testnet.suivision.xyz/object/0x9aafb73cbde8227798ae87661a39f91d61035e88d77993cfd40f6fd6cbbf5dfd>

4-1 動的フィールド

次に親オブジェクトから値を確認してみましょう。子IDはもはや使いません。



4-1 動的フィールド

子オブジェクトの値を操作してみましょう。+1を行なっています。

Transaction Blocks

Code

Statistics

dynamic_field_demo

Source

Bytecode

```
1 module 0xbbd8498877da0c0c5ce6bbcf4ff6e96ebf8c76e028eb51b7eb45
2     struct Parent has store, key {
3         id: 0x2::object::UID,
4     }
5
6     struct Child has store, key {
7         id: 0x2::object::UID,
8         count: u64,
9     }
10
11     struct ChildCheckResult has copy, drop {
12         exists: bool,
13     }
14
15     struct ChildCountResult has copy, drop {
16         count: u64,
17     }
18
19     public fun add_child(arg0: &mut Parent, arg1: &mut 0x2::t
20         let v0 = Child{
21             id : 0x2::object::new(arg1),
22             count : 0,
23         };
24         0x2::dynamic_field::add<vector<u8>, Child>(&mut arg0.
25     }
26
27     public fun attach_child_to_parent(arg0: &mut Parent, arg1
28         0x2::dynamic_field::add<vector<u8>, Child>(&mut arg0.
29     }
30
31     public fun check_child(arg0: &Parent) {
32         let v0 = ChildCheckResult{exists: 0x2::dynamic_field:
33         0x2::event::emit<ChildCheckResult>(v0);
34     }
35
```

Execute

Execute

create_independent_child

create_parent

get_child_count

has_child

increment_child_count

Arg0

0x9efd67b460d83c228c8e2a4f8cde76ef8fd9dd2

Execute

remove_child

4-1 動的フィールド

Deleting an object with dynamic fields

動的フィールドを持つオブジェクトの削除

It is possible to delete an object that has (potentially non-`drop`) dynamic fields still defined on it. Because field values can be accessed only via the dynamic field's associated object and field name, deleting an object that has dynamic fields still defined on it renders them all inaccessible to future transactions. This is true regardless of whether the field's value has the `drop` ability. This might not be a concern when adding a small number of statically known additional fields to an object, but is particularly undesirable for on-chain collection types that could be holding unboundedly many key-value pairs as dynamic fields.

動的フィールドがまだ定義されている (ドロップされない可能性がある) オブジェクトを削除することができます。フィールド値には、動的フィールドに関連付けられたオブジェクトとフィールド名を介してのみアクセスできるため、動的フィールドがまだ定義されているオブジェクトを削除すると、今後のトランザクションですべてのオブジェクトにアクセスできなくなります。これは、フィールドの値にドロップ能力があるかどうかに関係なく当てはまります。これは、静的に知られている少数の追加フィールドをオブジェクトに追加する場合には問題にならないかもしれませんが、動的フィールドとして無制限に多くのキーと値のペアを保持する可能性のあるオンチェーンコレクションタイプには特に望ましくありません。

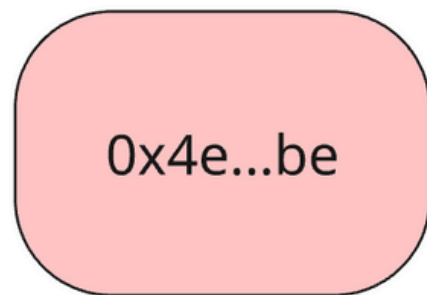
Sui provides `Table` and `Bag` collections built using dynamic fields, but with additional support to count the number of entries they contain to protect against accidental deletion when non-empty. To learn more, see [Tables and Bags](#).

Sui は、動的フィールドを使用して構築された `Table` コレクションと `Bag` コレクションを提供しますが、空でない場合の誤った削除を防ぐために、含まれるエントリの数のカウントするための追加のサポートが備わっています。詳細については、「[テーブルとバッグ](#)」を参照してください。

<https://docs.sui.io/concepts/dynamic-fields>

4-2 動的オブジェクトフィールド

まずは親のオブジェクトを作成します。



dynamic_object_field_demo

Transaction Blocks Code Statistics

Source Bytecode

```
3      id: 0x2::object::UID,  
4    }  
5  
6    struct Child has store, key {  
7      id: 0x2::object::UID,  
8      count: u64,  
9    }  
10  
11    public fun add_child(arg0: &mut Parent, arg1: &mut 0x2::t  
12      let v0 = Child{  
13        id : 0x2::object::new(arg1),  
14        count : 0,  
15      };  
16      0x2::dynamic_object_field::add<vector<u8>, Child>(&mu  
17    }  
18  
19    public fun attach_child_to_parent(arg0: &mut Parent, arg1  
20      0x2::dynamic_object_field::add<vector<u8>, Child>(&mu  
21    }  
22  
23    public fun create_independent_child(arg0: &mut 0x2::tx_co  
24      let v0 = Child{  
25        id : 0x2::object::new(arg0),  
26        count : 0,  
27      };  
28      0x2::transfer::public_transfer<Child>(v0, 0x2::tx_con  
29    }  
30  
31    public fun create_parent(arg0: &mut 0x2::tx_context::TxCo
```

Execute

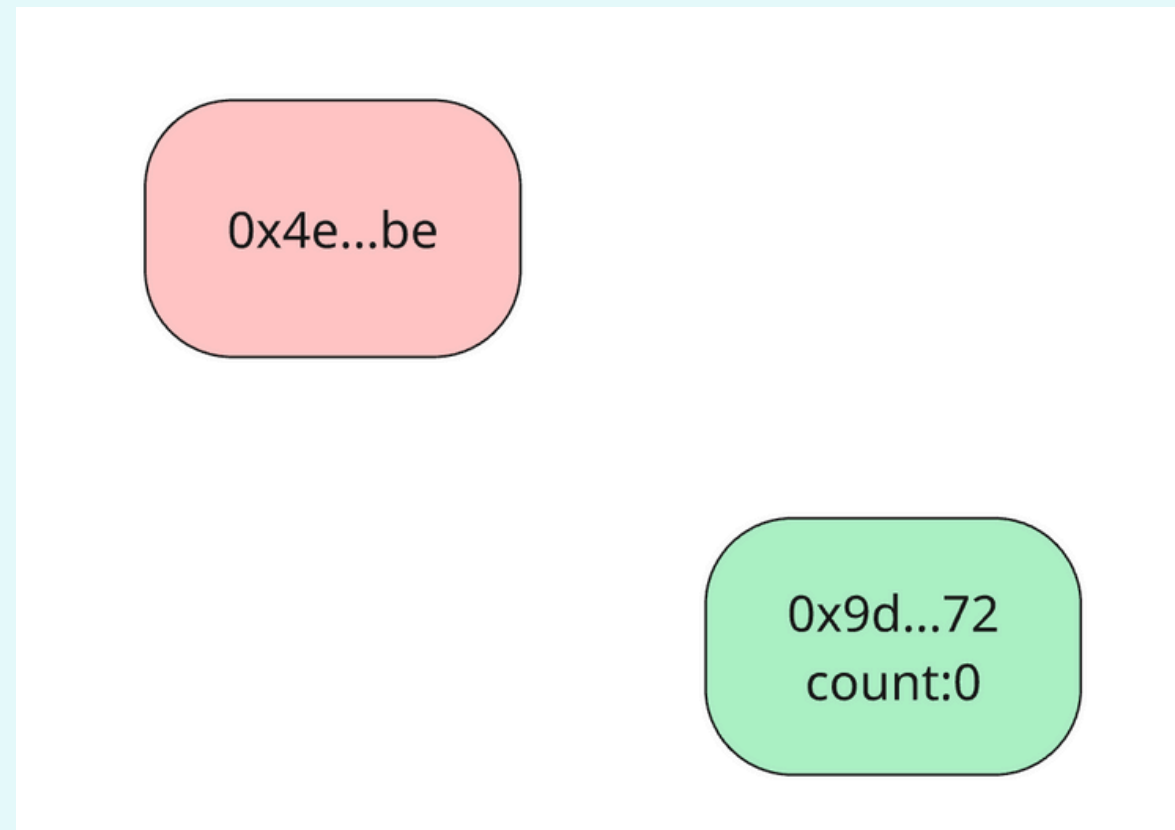
add_child
attach_child_to_parent
create_independent_child
create_parent

Execute

<https://testnet.suivision.xyz/package/0x1b9c2b3af8aeba360d50fce544a57cfcc9d4670ad01ae7356a9ff9a9b975ba7a?tab=Code>

4-2 動的オブジェクトフィールド

次に別のオブジェクトを作ります。この時点では関係性はありません。



Version: 1
Owner: Immutable

Transaction Blocks Code Statistics

dynamic_object_fiel
d_demo

Source Bytecode

```
2 struct Parent has store, key {  
3   id: 0x2::object::UID,  
4 }  
5  
6 struct Child has store, key {  
7   id: 0x2::object::UID,  
8   count: u64,  
9 }  
10  
11 public fun add_child(arg0: &mut Parent, arg1: &mut 0x2::t  
12   let v0 = Child{  
13     id : 0x2::object::new(arg1),  
14     count : 0,  
15   };  
16   0x2::dynamic_object_field::add<vector<u8>, Child>(&mu  
17 }  
18  
19 public fun attach_child_to_parent(arg0: &mut Parent, arg1  
20   0x2::dynamic_object_field::add<vector<u8>, Child>(&mu  
21 }  
22  
23 public fun create_independent_child(arg0: &mut 0x2::tx_co  
24   let v0 = Child{
```

Execute

add_child

attach_child_to_parent

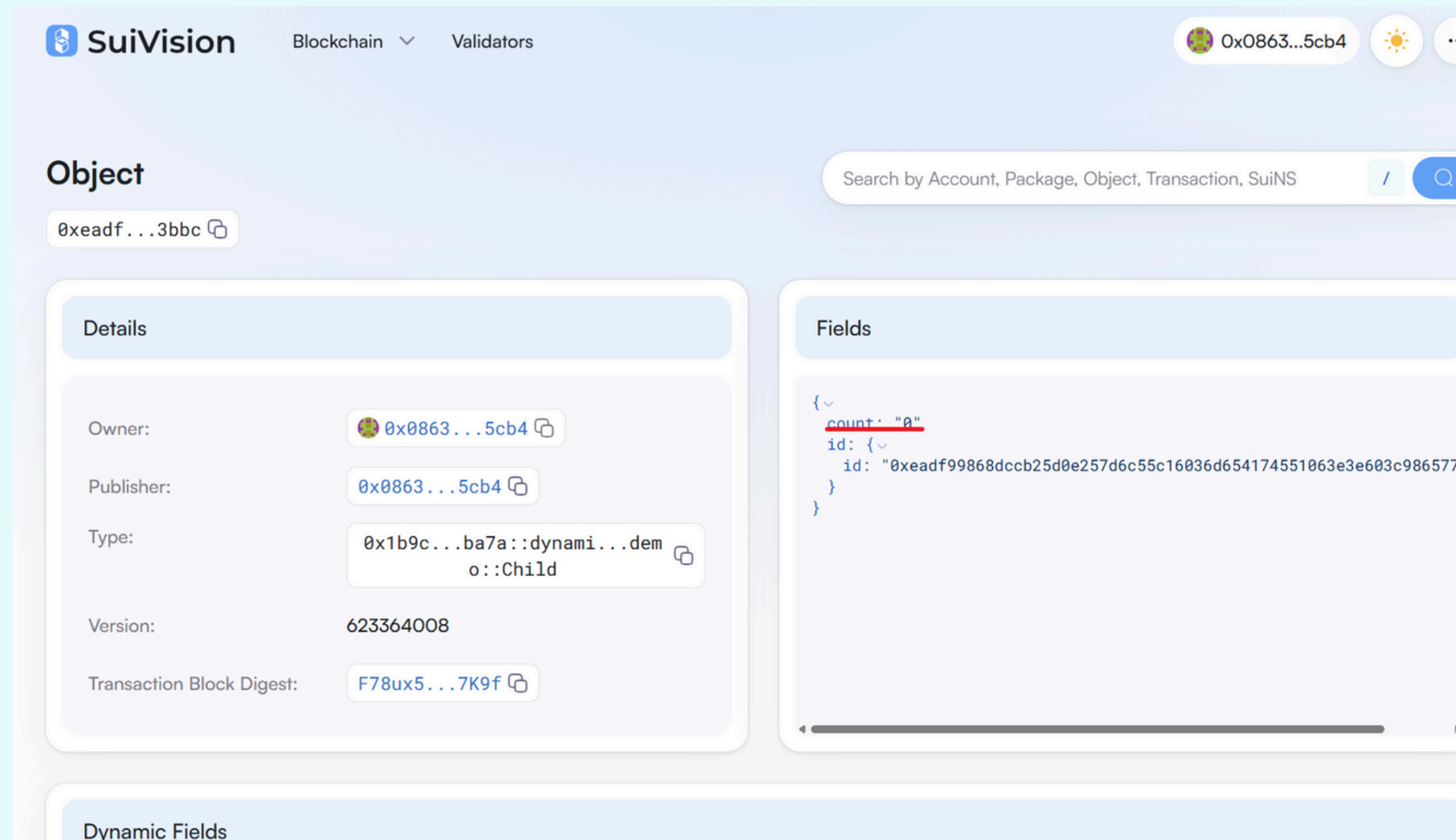
create_independent_child

create_parent

Execute

4-2 動的オブジェクトフィールド

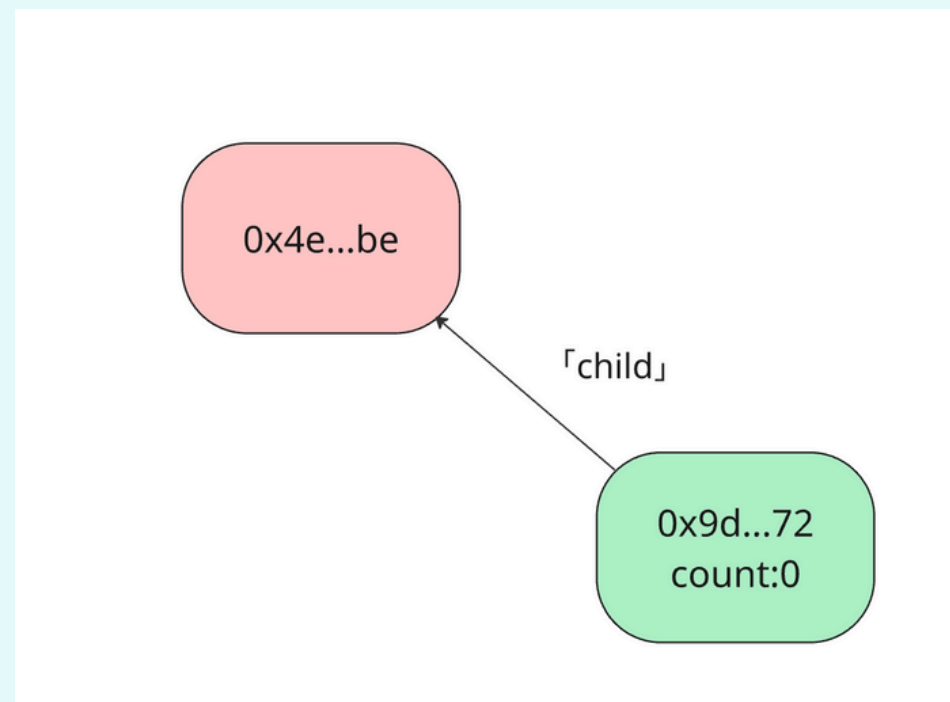
作成したオブジェクトにアクセスできることを確認します。



<https://testnet.suivision.xyz/object/0xeadf99868dcc25d0e257d6c55c16036d654174551063e3e603c986577c03bbc>

4-2 動的オブジェクトフィールド

では子オブジェクトを「child」という名前で紐づけましょう。



Transaction ID: C293sSZV2bEZUAWao8hAdUVqLdzohsg4DqqntfRTsXLe

Version: 1

Owner: Immutable

Transaction Blocks | Code | Statistics

dynamic_object_field_demo

Source | Bytecode

```
1 module 0x1b9c2b3af8aeba360d50fce544a57cfc9d4670ad01ae7356a9f
2     struct Parent has store, key {
3         id: 0x2::object::UID,
4     }
5
6     struct Child has store, key {
7         id: 0x2::object::UID,
8         count: u64,
9     }
10
11 public fun add_child(arg0: &mut Parent, arg1: &mut 0x2::t
12     let v0 = Child{
13         id : 0x2::object::new(arg1),
14         count : 0,
15     };
16     0x2::dynamic_object_field::add<vector<u8>, Child>(&mu
17 }
18
19 public fun attach_child_to_parent(arg0: &mut Parent, arg1
20     0x2::dynamic_object_field::add<vector<u8>, Child>(&mu
21 }
22
```

Execute

add_child

attach_child_to_parent

Arg0

0x033ae2fafc2a56adfcc346c62250eae043e9691f

Arg1

0xeadf99868dcc25d0e257d6c55c16036d654174

Execute

create_independent_child

4-2 動的オブジェクトフィールド

親オブジェクトからだけでなく、子オブジェクトからも確認できることがわかります。

Version: 619095322

Transaction Block Digest: 7WjW7o...uHSQ

Dynamic Fields

CHILD OBJECT ID	VERSION	OBJECT TYPE	LAST TX ID
0x9d22...9272	619,095,322	0x1b9c...ba7a::dynam...demo::Child	7WjW7o...uHSQ

```
{
  name: {
    type: "vector<u8>"
    value: [
      0: 99
      1: 184
      2: 185
      3: 188
      4: 188
    ]
  }
  bcsEncoding: "base64"
  bcsName: "DYNObJxx"
  type: "DynamicObject"
  objectType: "0x1b9c...child"
  objectId: "0x9d22a386878b1d6ab82a5433422663b9438cbde6ec868e361888ecb8dbf99272"
  version: 619095322
  digest: "5mPVam36vHGg1V1nRKqG58pwwAFPzCtJHvgjYgQjmVR7"
}
```

SuiVision Blockchain Validators OX0863...Scb4

Object 0x9d22...9272

Details

Owner: 0x8212...bf54

Publisher: 0x917b...bb2f

Type: 0x1b9c...ba7a::dynam...demo::Child

Version: 619095322

Transaction Block Digest: 7WjW7o...uHSQ

Fields

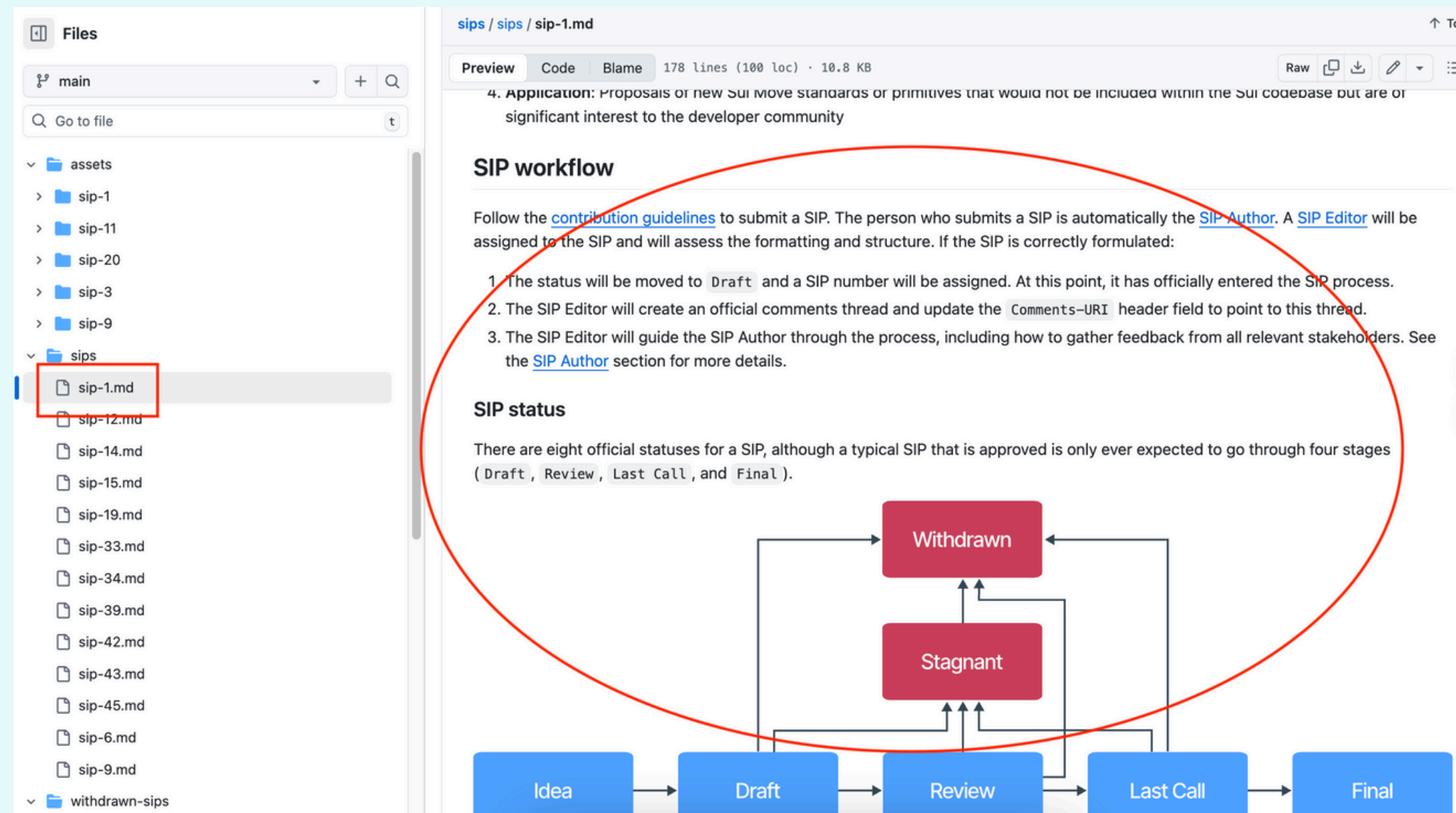
```
{
  count: "8"
  id: {
    id: "0x9d22a386878b1d6ab82a5433422663b9438cbde6ec868e361888ecb8dbf99272"
  }
}
```

Dynamic Fields

CHILD OBJECT ID	VERSION	OBJECT TYPE	LAST TX ID
-----------------	---------	-------------	------------

5 SIPとは

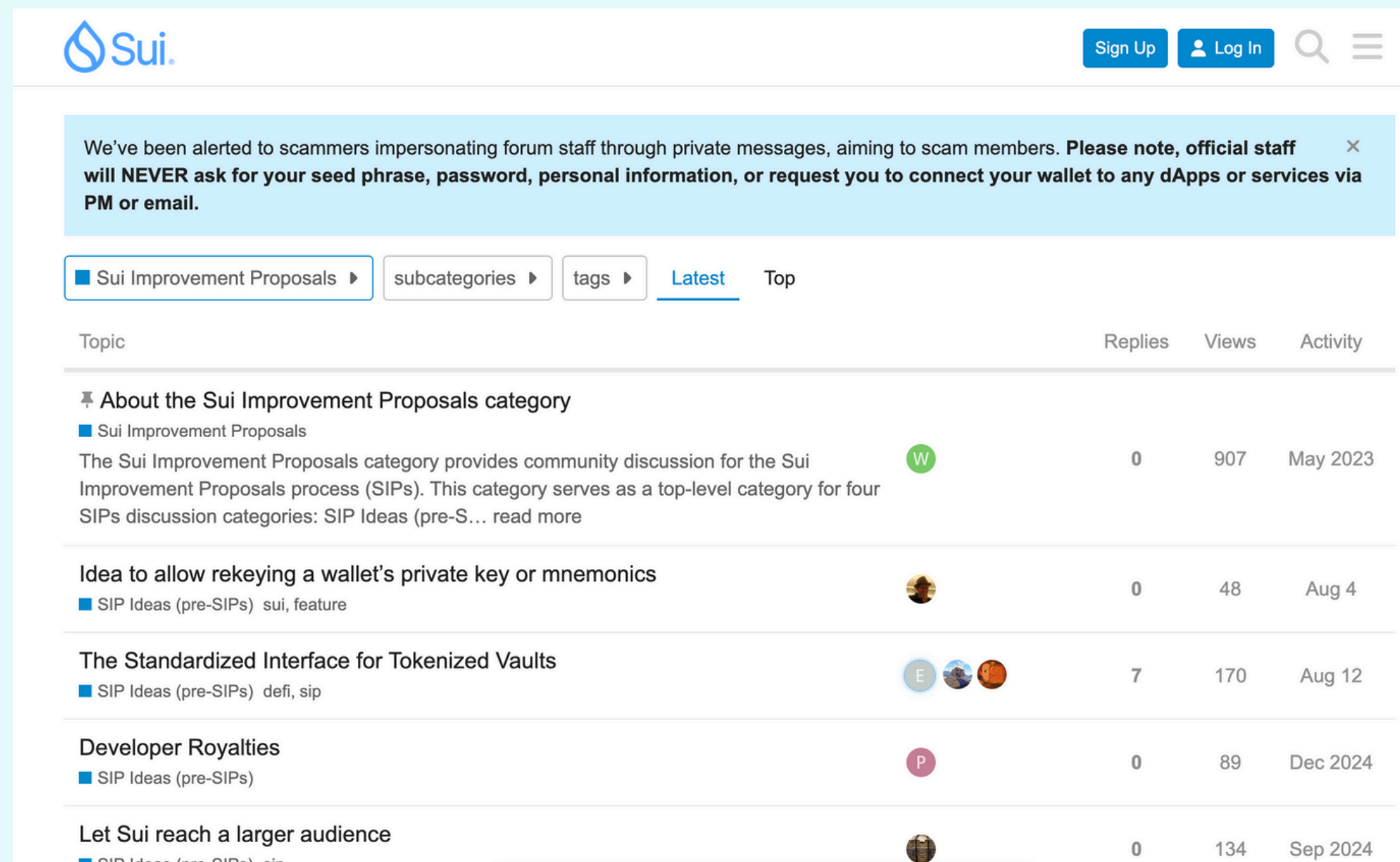
Sui Improvement Proposalの略で、Suiについての新しい提案を行うことができます。



<https://github.com/sui-foundation/sips/blob/main/sips/sip-1.md>

5 SIPとは

Sui Improvement Proposalの略で、Suiについての新しい提案を行うことができます。



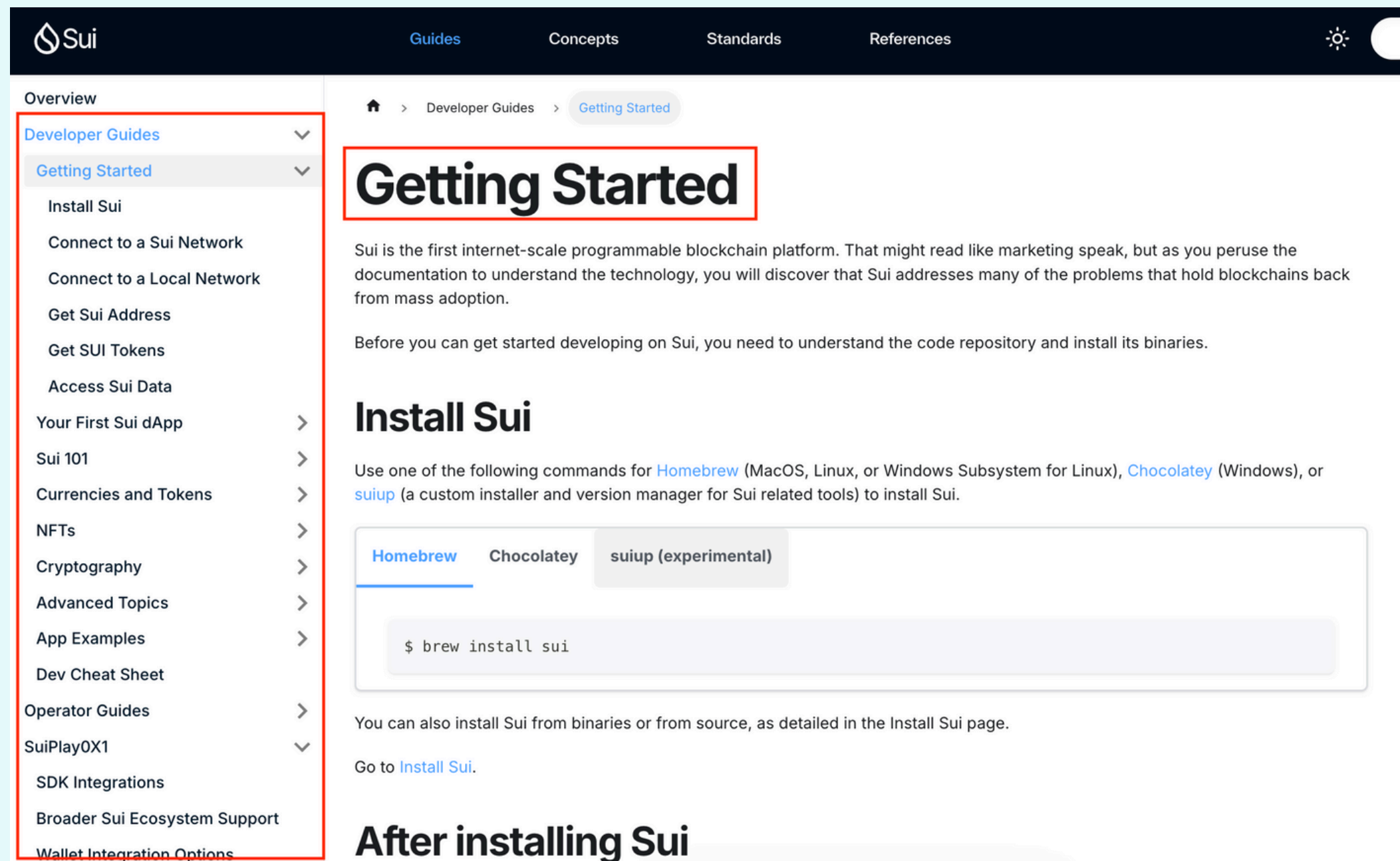
<https://github.com/sui-foundation/sips/blob/main/sips/sip-1.md>

開発について ①開発コミュニティ



<https://discord.com/invite/NM8jWvmQ6V>

開発について ②公式ドキュメント



<https://docs.sui.io/guides/developer/getting-started>

開発について ③ハンズオン動画

Suiハンズオン

1から学ぶSui入門
インストール後〜
パッケージの公開

Suiハンズオン

作成者: yuki

再生リスト・6本の動画・0回視聴

すべて再生

1 Suiハンズオン 1から学ぶSui入門 インストール後〜パッケージの公開 #Sui
yuki・188回視聴・1か月前

2 Suiハンズオン フロントエンドでのウォレット接続実装 #Sui
yuki・49回視聴・1か月前

3 Suiハンズオン フロントエンド徹底ハンズオン #sui
yuki・81回視聴・1か月前

4 Suiハンズオン Dockerを利用したPackage〜フロントエンドの作成
yuki・97回視聴・3週間前

5 Suiハンズオン AIエージェントキットの使用 (Nimbus) #sui
yuki・50回視聴・2週間前

6 Suiハンズオン コインの作成 〜DenyListの操作まで
yuki・39回視聴・9日前

[https://www.youtube.com/playlist?
list=PL1rRUVBEqXMnlkv4B5FZAllt76kwe6Q_7](https://www.youtube.com/playlist?list=PL1rRUVBEqXMnlkv4B5FZAllt76kwe6Q_7)

免責事項

本資料は、ブロックチェーン技術の学習・情報共有を目的として作成したものであり、正確性や完全性を保証するものではありません。内容には誤りや不十分な点が含まれる可能性があります。

また、本資料は投資を勧誘・推奨する意図は一切なく、投資判断に利用することを目的としたものではありません。

実際に投資や取引を行う場合は、ご自身の責任と判断において行ってください。