

# Sui Tokyo Developers Workshop Episode 2



Pacific Meta

# 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

**KOUKA**

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ケ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 >

お問い合わせはこちら >



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

# 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。

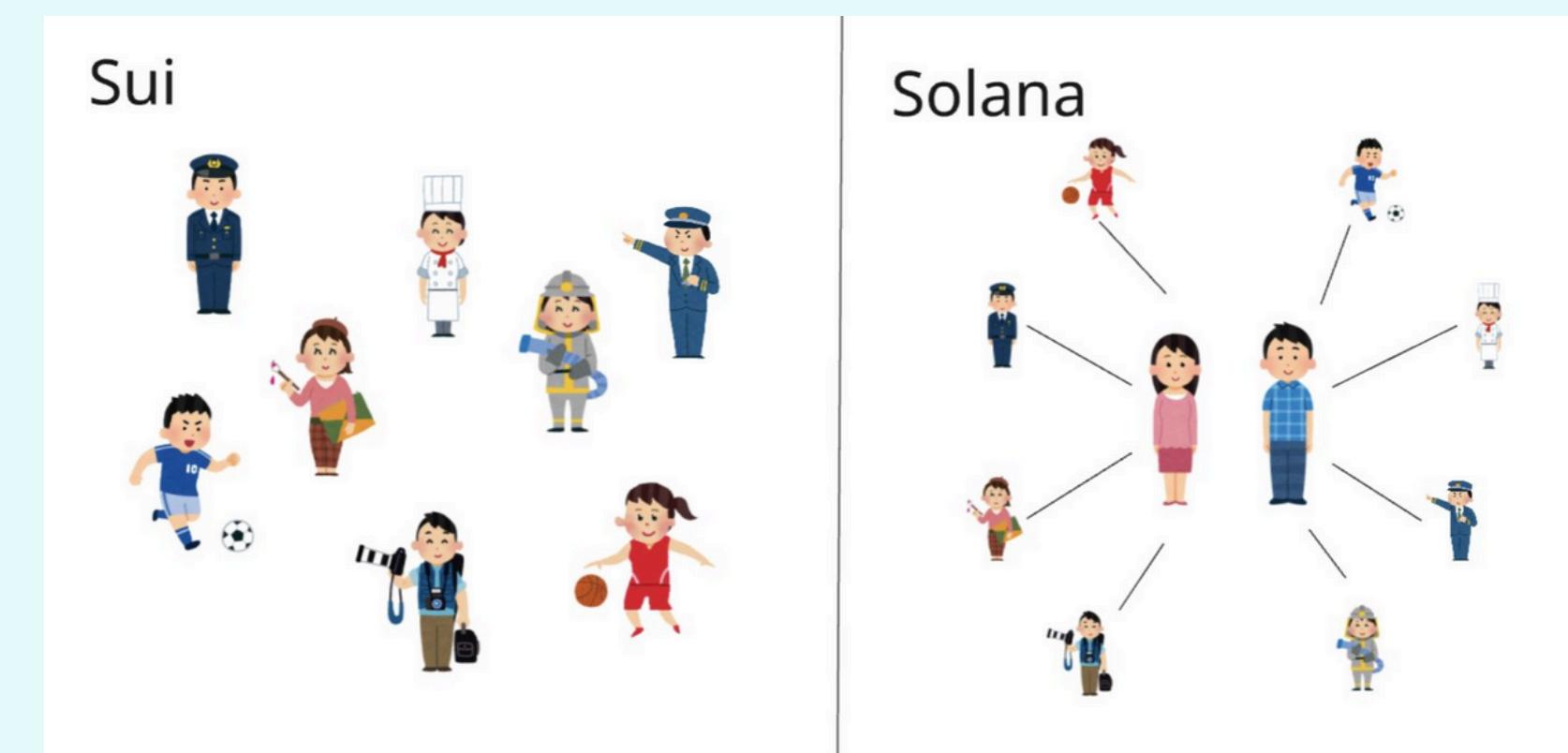
# Suiの本質は？ 型を重視した設計

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

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  
}
```

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

# 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はどちらだと思いますか。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\\_forgetten\\_land\\_copy/?tl=ja](https://www.reddit.com/r/Kirby/comments/ufs0jw/i_made_kirby_and_the_forgetten_land_copy/?tl=ja)



[https://jumpcs.shueisha.co.jp/shop/g/g4530430522989/?srsltid=AfmBOop7lqDSN\\_8CqP9zy98iL\\_up-6kgpKNotRGuAWsJPsY16rClqnpp](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。

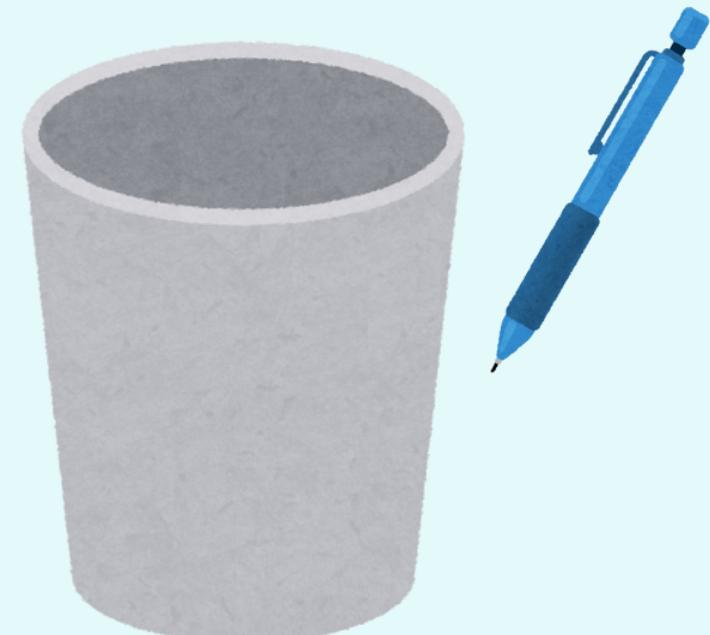
なので、原則コピーはできず、できるようになるには「Copy」をつきます。



# 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**」をつきます。

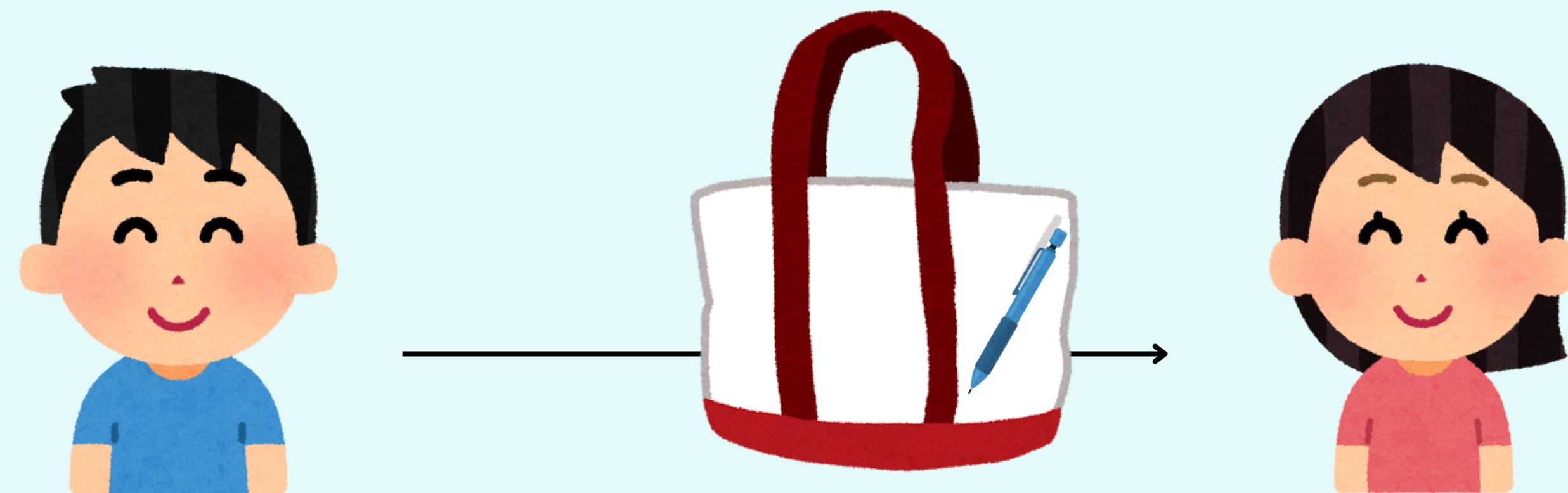


# 型からの派生 ①能力 (Ability) 4 Store

Moveは「移動」に焦点を当てた言語です。

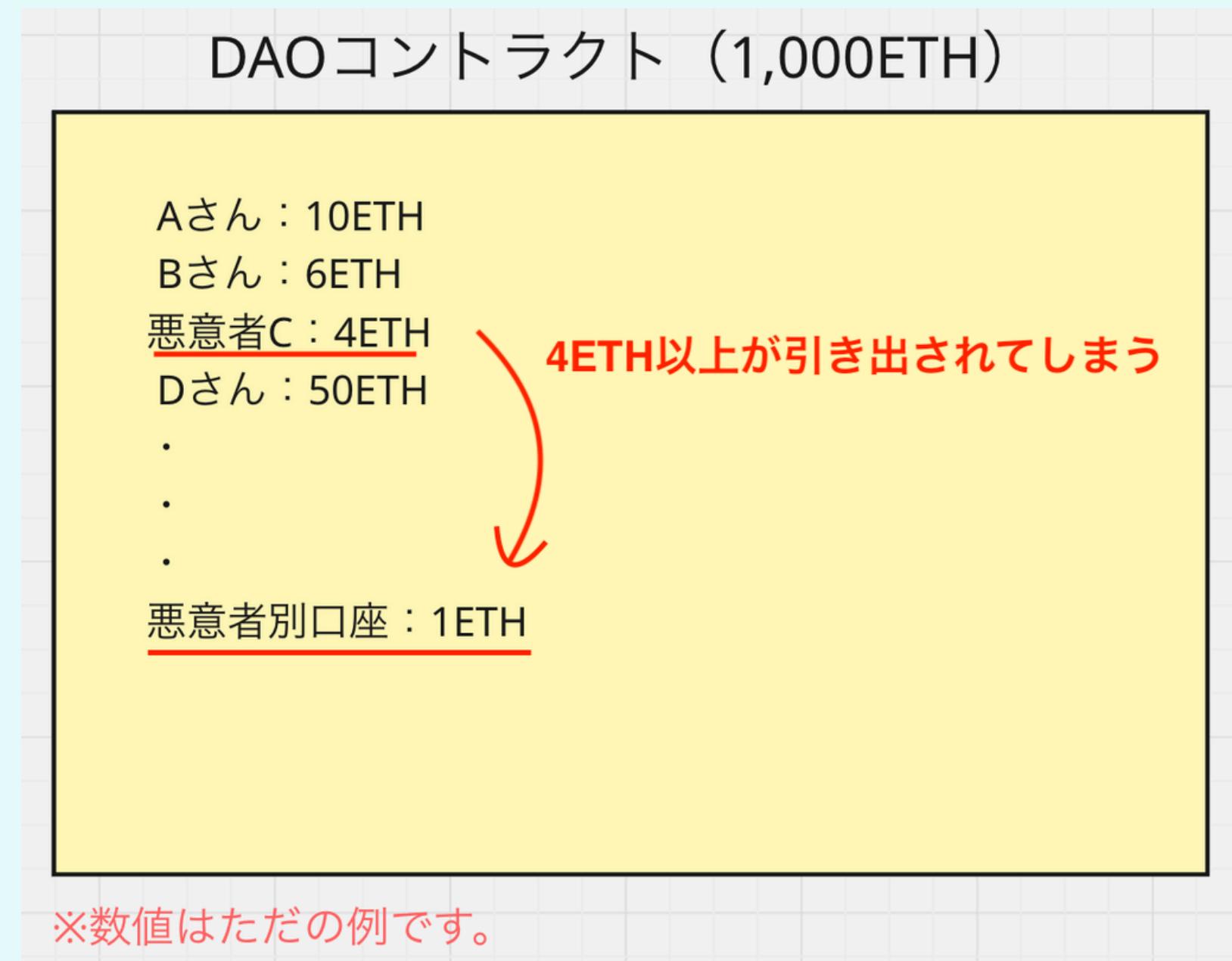
もしもシャーペンがバッグに入っている（オブジェクトのフィールド）場合、  
バッグが移動すれば、その中のシャーペンも移動します。

そのため、格納できるかどうかで **「store」** をつけます。



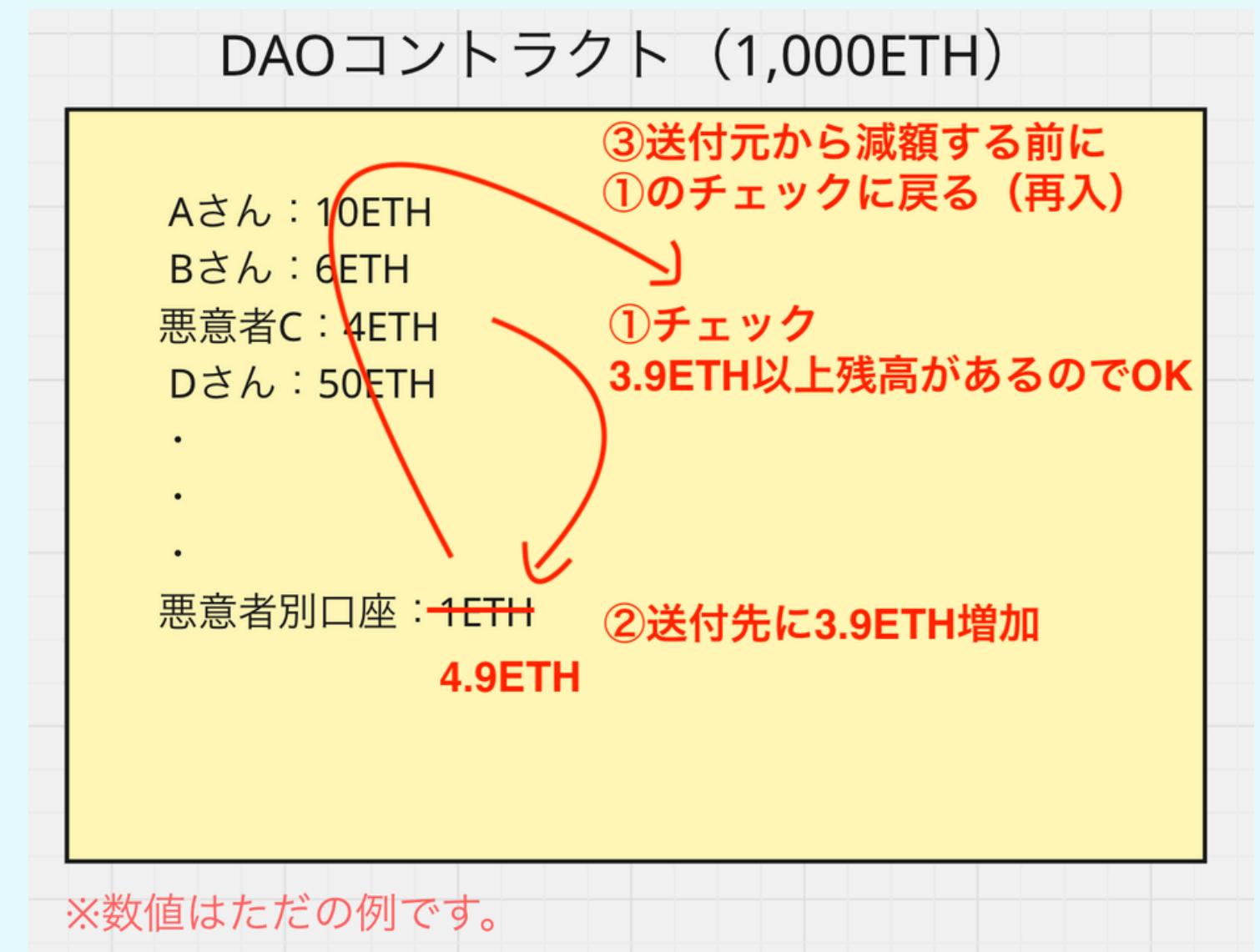
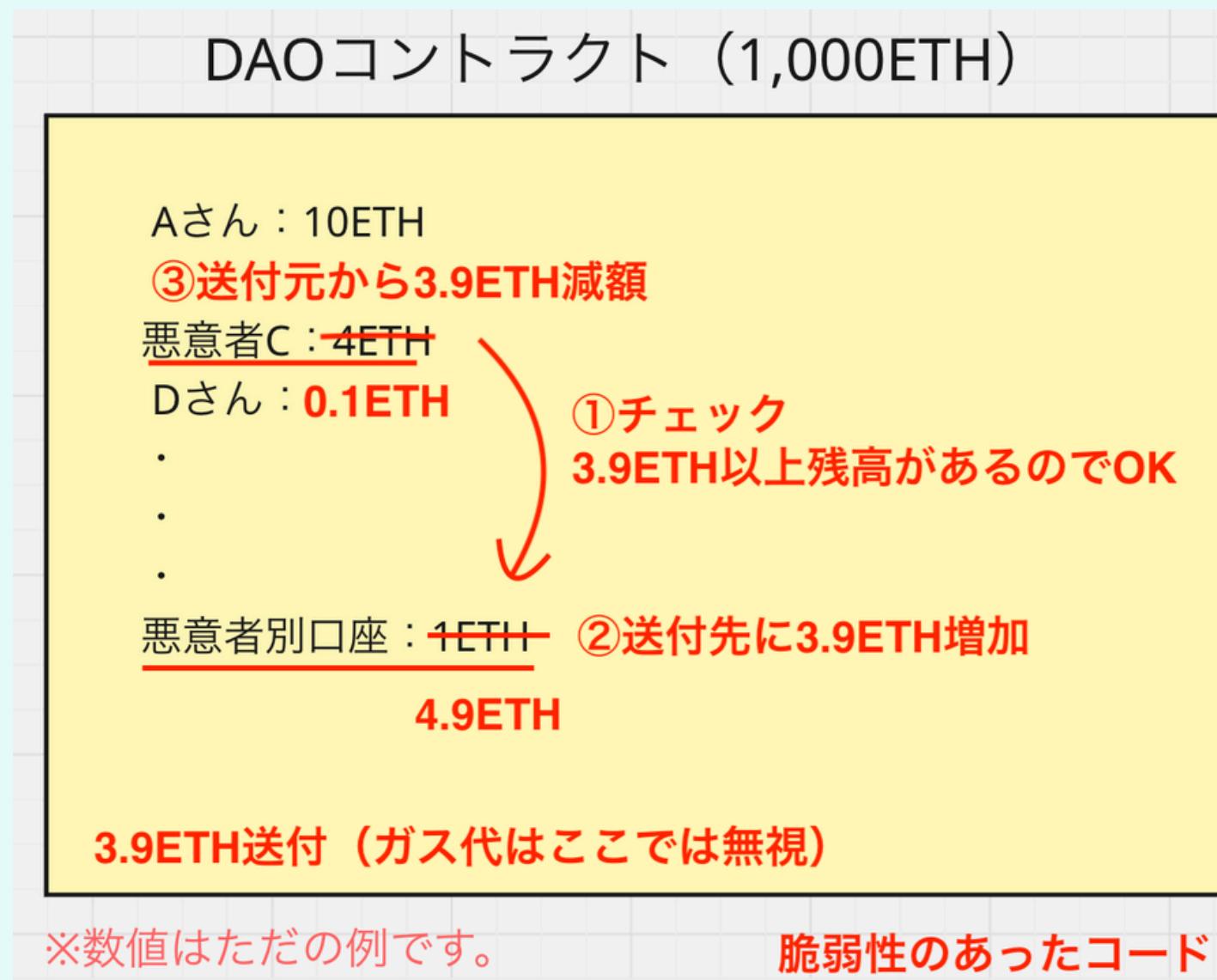
# 補足 The DAO事件とは

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



# 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）によって攻撃を受けてしまった。



# 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の**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



# 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、不变のいずれかになり得ます。

## 所有権

Wrapping (ラップ) により单一所有を強制し、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

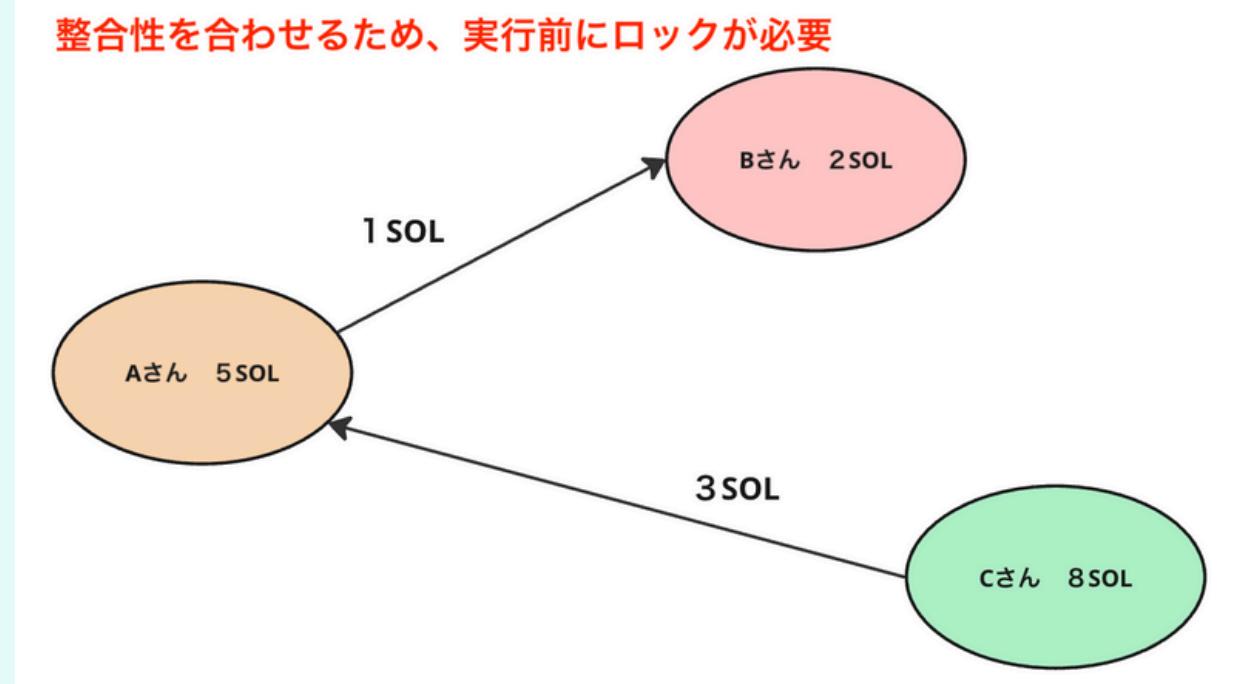
Suiはデータの値ではなく、**オブジェクト**という形で判断します。

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**することで整合性を保ちます。

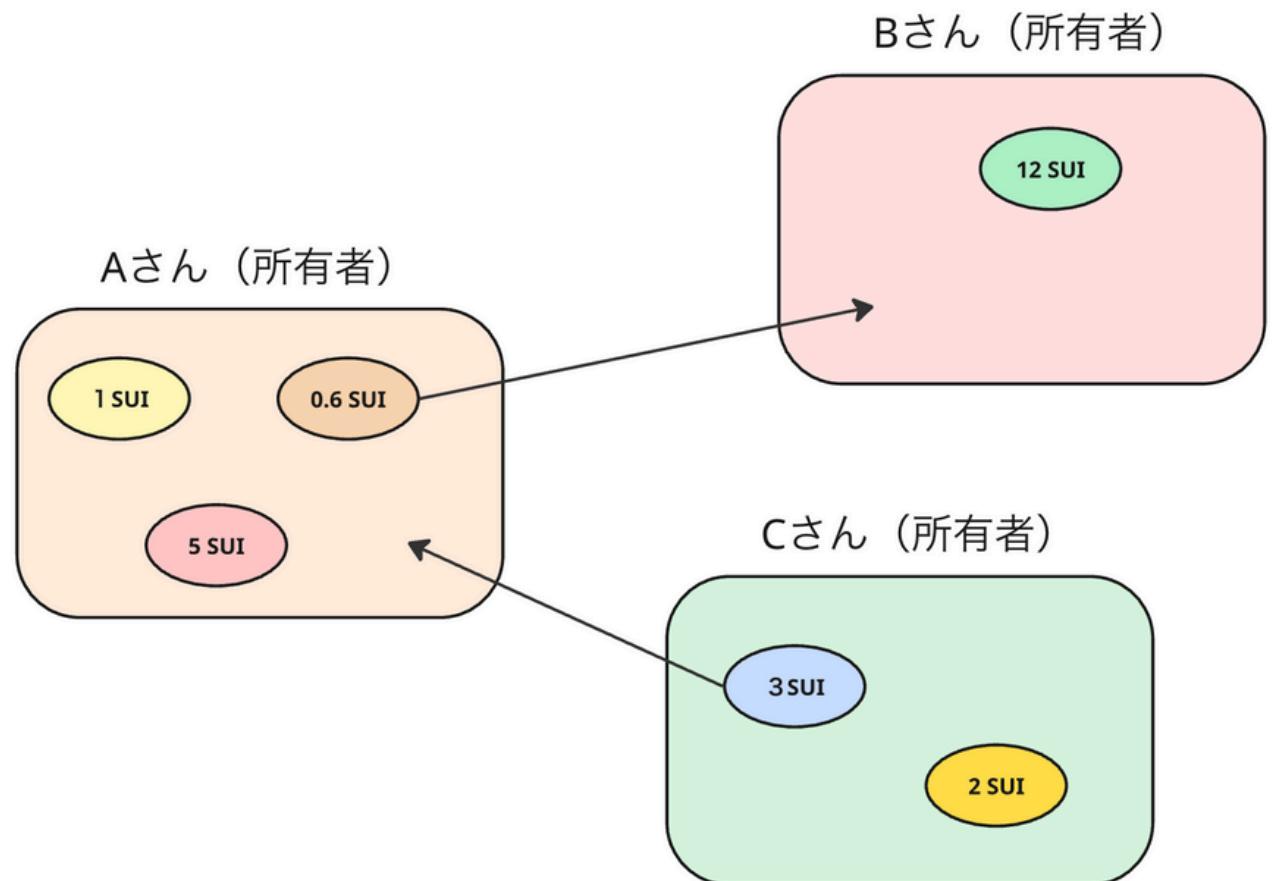


# 型からの派生 ② オブジェクトと所有権

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。



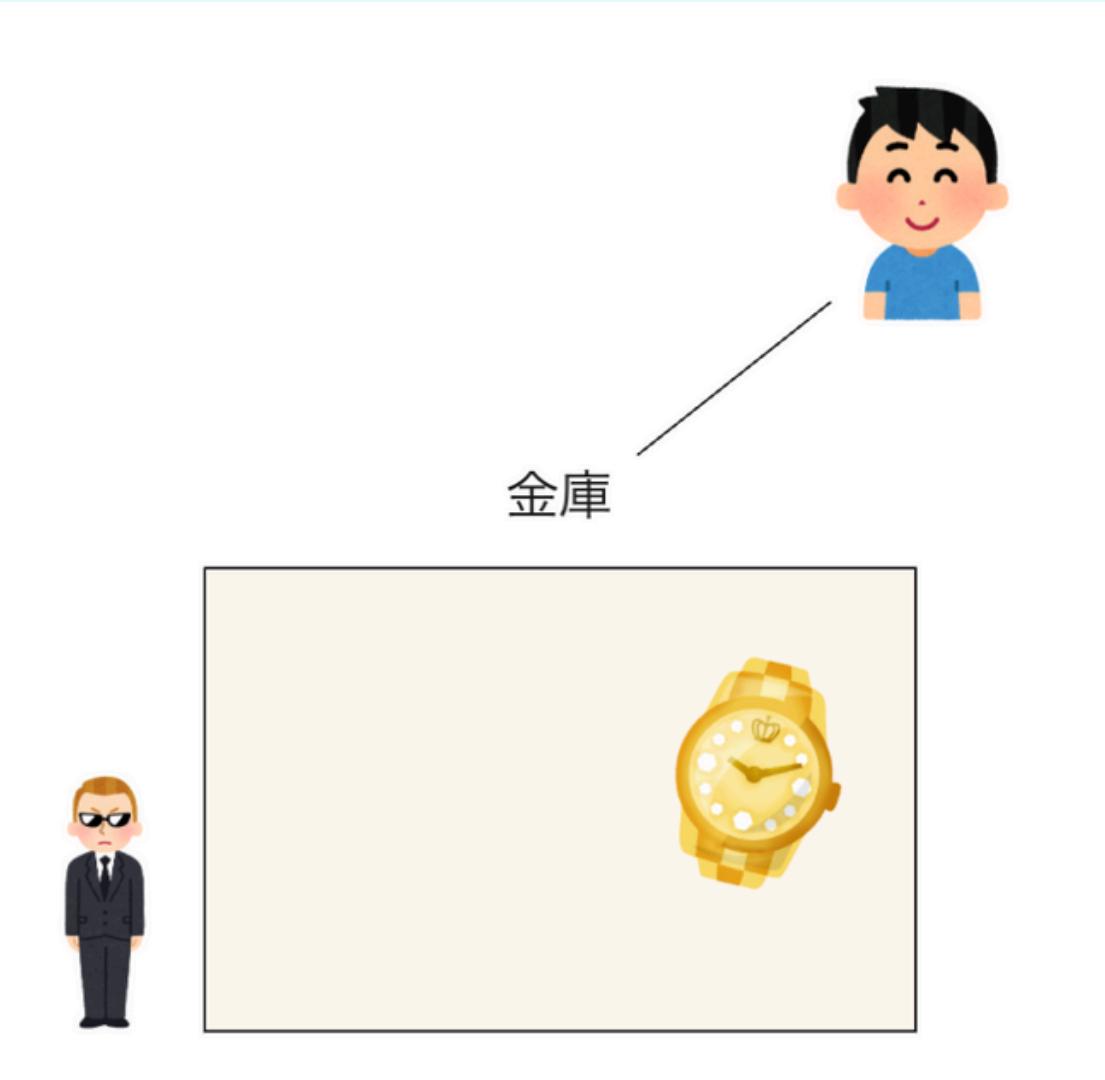
# 型からの派生 ③ラップと単一所有の強制

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。



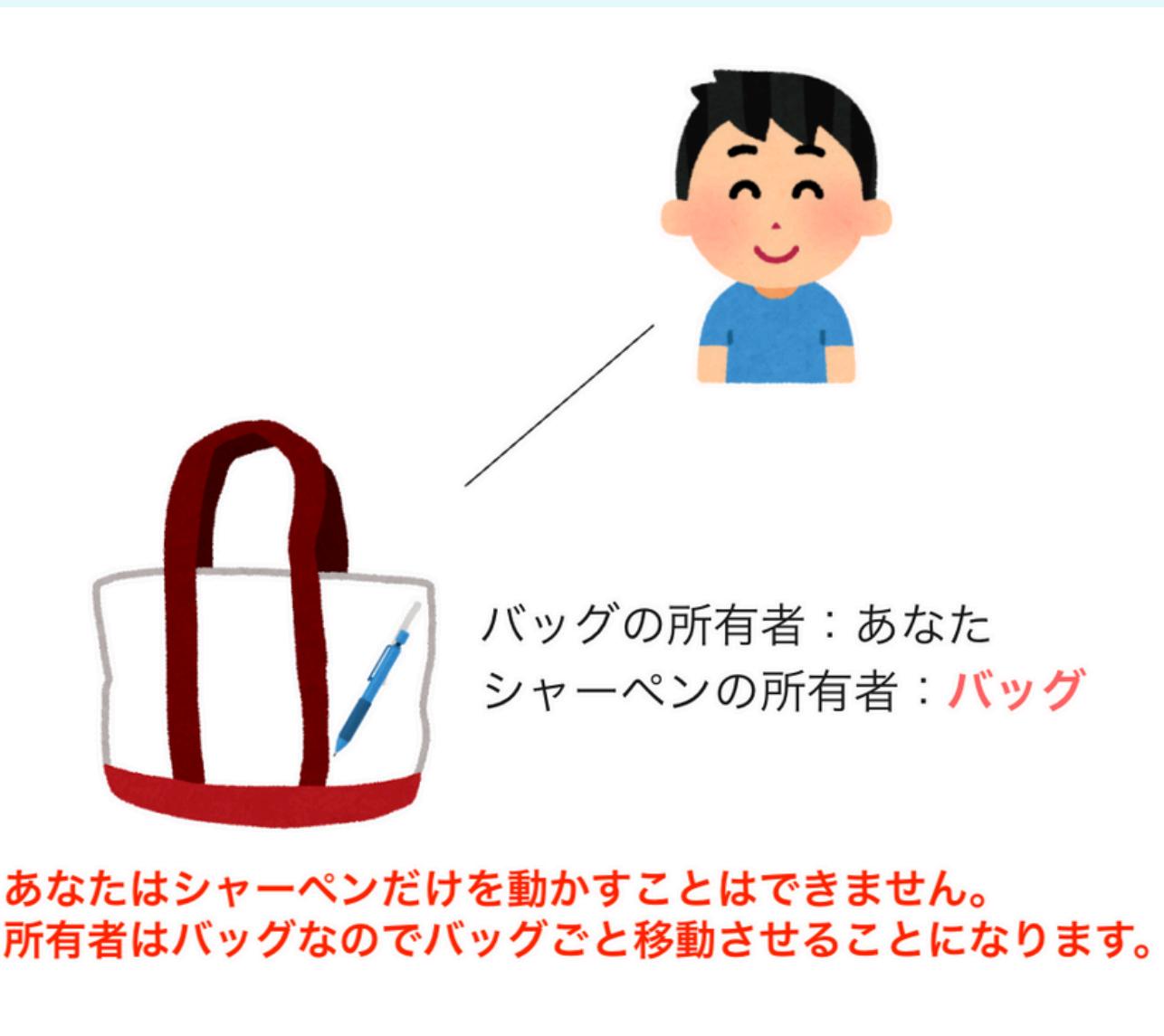
# 型からの派生 ③ラップと単一所有の強制

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。



バッグの所有者：あなた  
シャーペンの所有者：バッグ

あなたはシャーペンだけを動かすことはできません。  
所有者はバッグなのでバッグごと移動させることになります。

# 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 ④ Capability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**が異なる点です。

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。

コードをコピーする

```
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+引数をシリアル化したバイナリ列 (概念表示)
            data: MintTo { amount: 100 },
        },
    ],
}
```

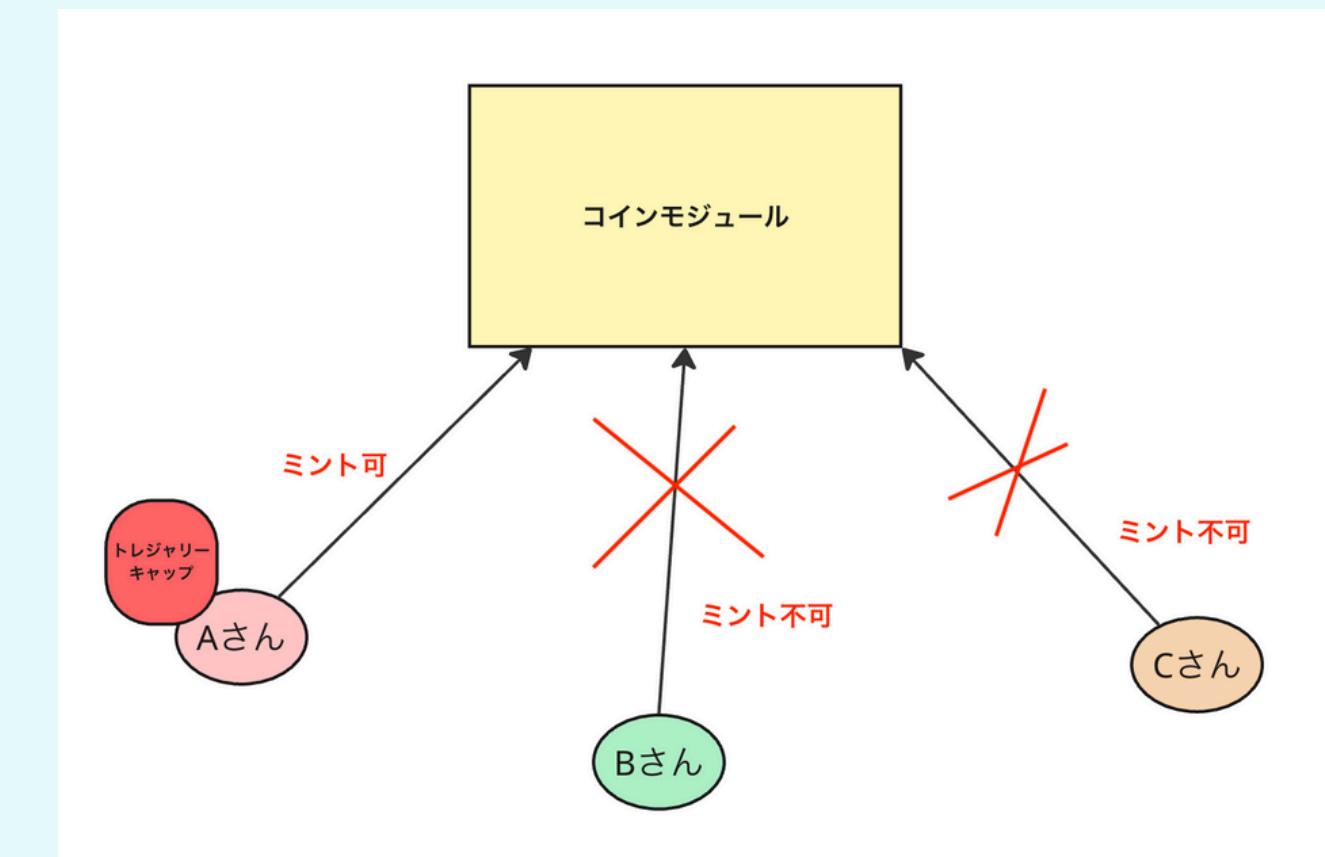
# 型からの派生 ④ Capabilityによる権限

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 ico
    // 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



# 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

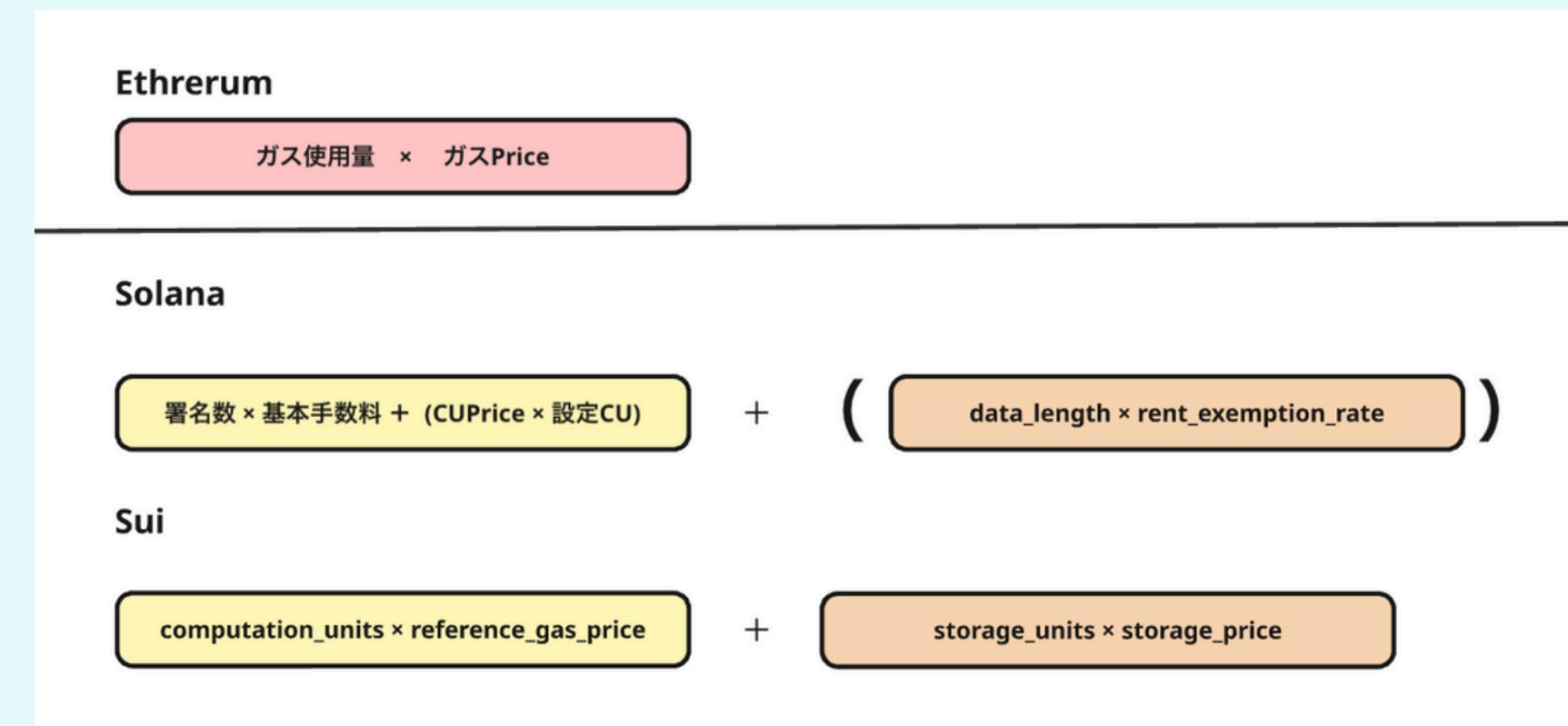
重要なコントラクトロジックの正しさを数学的に証明します。

# 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。



# 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。

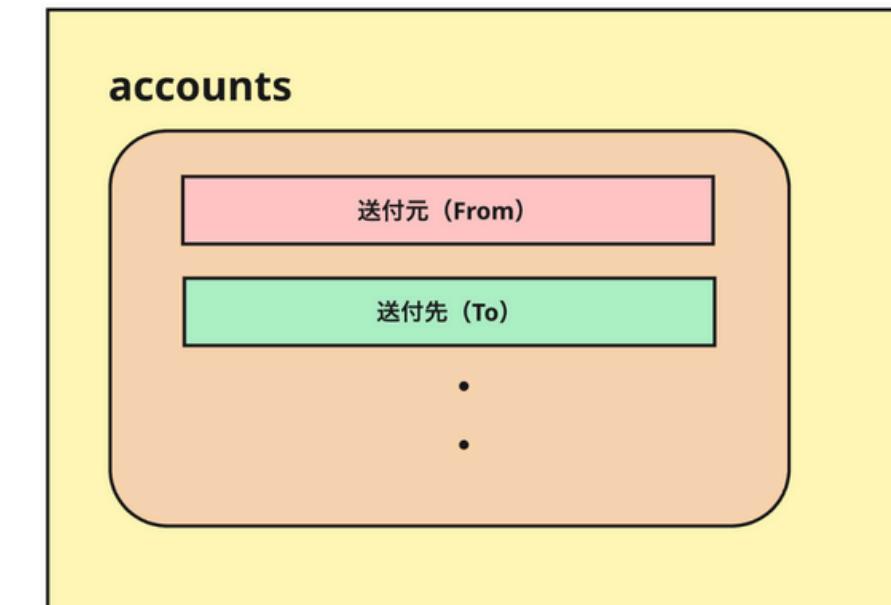
例えば、Solanaで送付時に総量が変わらないことを客観的に証明してみましょう。

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

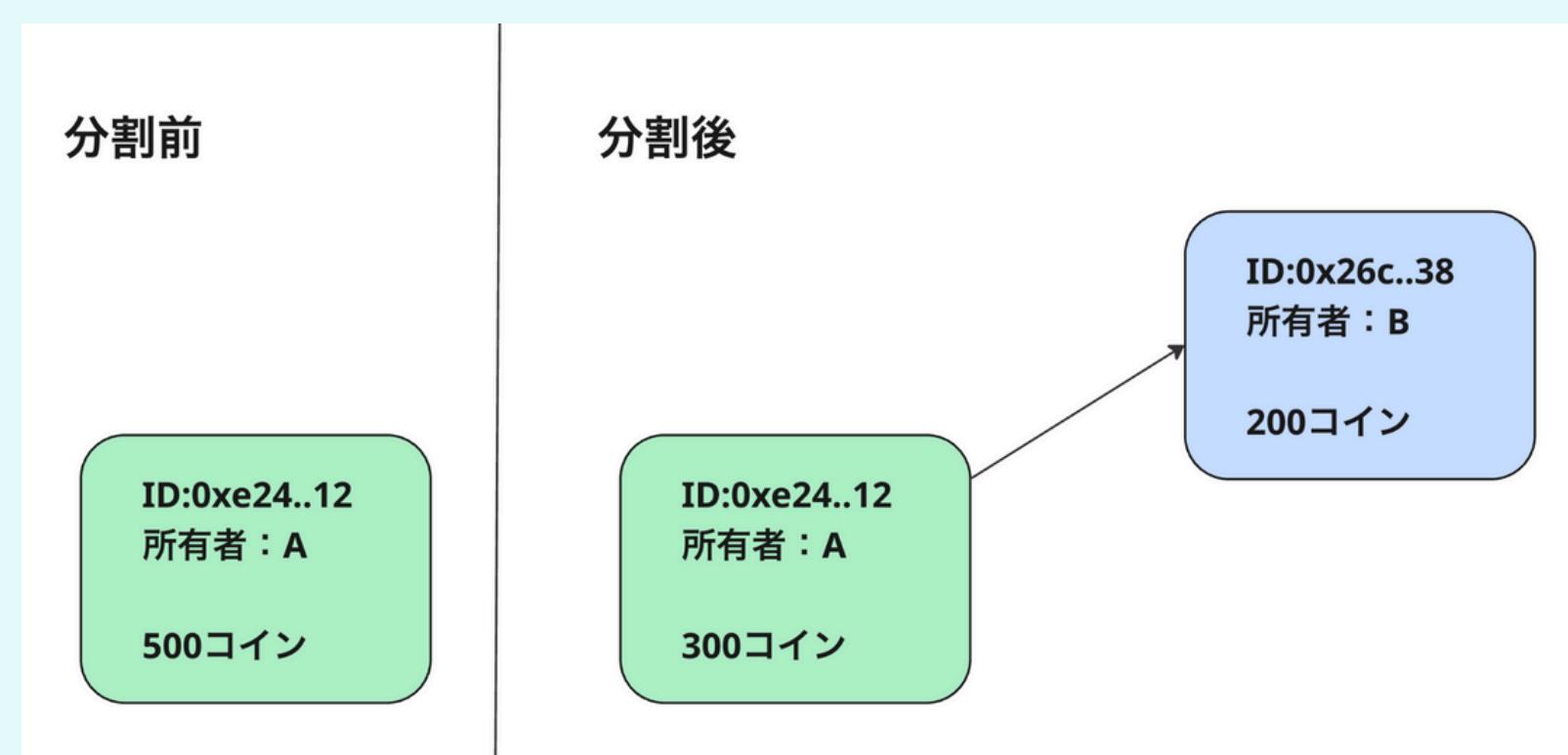
## ⑦形式的検証

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所以 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);
    }
}
```



# 開発について ①開発コミュニティ



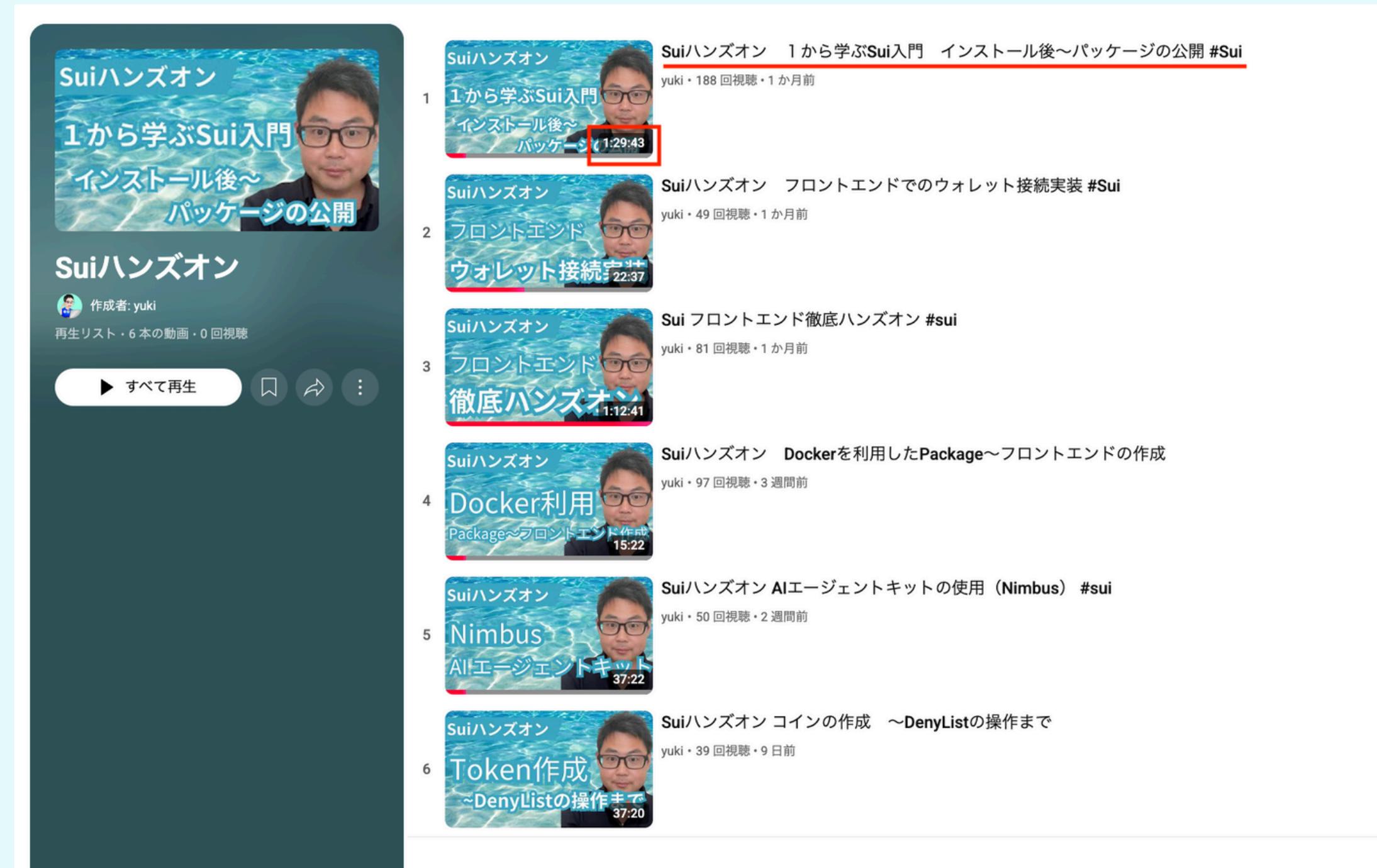
<https://discord.com/invite/NM8jWvmQ6V>

# 開発について ②公式ドキュメント

The screenshot shows the Sui Developer Guides website. The navigation bar at the top includes links for Guides, Concepts, Standards, References, and a light/dark mode switch. The main content area has a breadcrumb trail: Home > Developer Guides > Getting Started. A large red box highlights the 'Getting Started' section in the sidebar menu, which lists various developer guides. The main article title is 'Getting Started'. It describes Sui as the first internet-scale programmable blockchain platform and explains that before starting development, one needs to understand the code repository and install its binaries. It provides instructions for Homebrew, Chocolatey, and suiup (experimental) and includes a command-line example: '\$ brew install sui'. Below this, it states that Sui can also be installed from binaries or source. A link to the 'Install Sui' page is provided. The article concludes with the heading 'After installing Sui'.

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

# 開発について ③ハンズオン動画



[https://www.youtube.com/playlist?  
list=PL1rRUVBEqXMnlkv4B5FZAllt76kwe6Q\\_7](https://www.youtube.com/playlist?list=PL1rRUVBEqXMnlkv4B5FZAllt76kwe6Q_7)

# 開発について

## ④事前準備



Photo by *ibaraki\_nakai*

### 【完全保存版】Sui勉強会の事前準備

❤ 10



ユウキ

2025年10月1日 08:14

...

利用するPCは個人所有のものを想定しています。

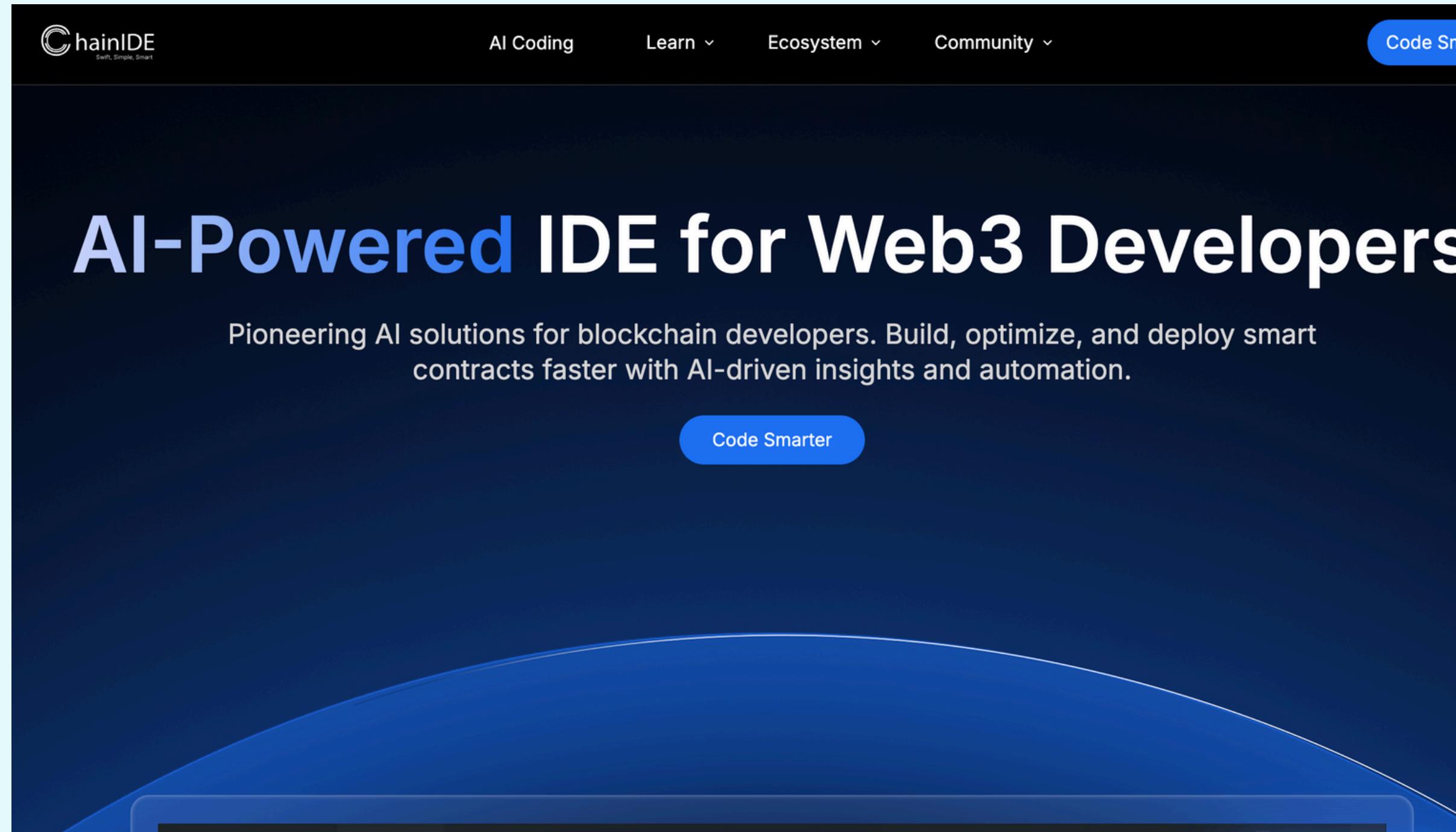
会社のPCなどを利用する場合は、ご自身の責任の下、会社のルールに則って実施してください。

ハンズオン勉強会は環境構築や準備に手間取ることが少なくありません。

しっかりと事前準備を行い、身のある勉強会にしていきましょう。

<https://note.com/standenglish/n/ndd97f7417504>

# 開発について ⑤IDEツール



<https://chainide.com/>

ハルキさんに教えていただきました。

# 開発について ⑥Dockerを使った開発



[https://www.youtube.com/watch?v=qpt\\_hECr8F8](https://www.youtube.com/watch?v=qpt_hECr8F8)

資料

<https://www.canva.com/design/DAGy6pZPE3c/FFSX5ZuKdZcl0T25W1uNdQ/edit>

# 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。