

東大寄付講座Solidity講義2024\_1座学

# Solidity開発の概要

株式会社 Ecdysis CEO / Solidity House 運営

落合 渉悟

- スマートコントラクト概要
- スマートコントラクトとセキュリティ
- スマートコントラクトと法制度
- スマコンのDevOps
- Metacontract

# スマートコントラクト概要

# スマートコントラクトとは

「チューリング完全なプログラムをアップロードできる機能」をブロックチェーンに持たせることで、一からネットワークを普及させずとも改ざん耐性のあるシステムを開発できるもの。

**スマートコントラクト**とはこのようなプログラムを開発および公開できる**処理系**(プログラムの実行環境)、およびそれをターゲットとした**高級プログラミング言語**を総称したもの。

なお、個々のプログラムそのものをコントラクト=契約と呼ぶ。

# スマートコントラクトの分類

スマートコントラクトは二種類に分類できる。

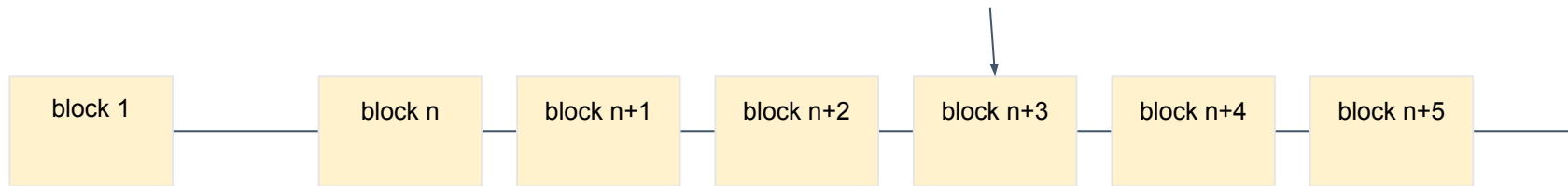
ひとつは末端の使用者の側で作用後の状態がどのようなものになるかまでを含めてメッセージとして署名する**UTXO型**のスマートコントラクトであり、ビットコインなどがそれにあたる。

もうひとつは使用者は実行対象のプログラムとインプットだけを用意して、あと状態はパブリックチェーンの側が計算する**VM型**のスマートコントラクトであり、イーサリアムなどがそれにあたる。

# UTXO型スマートコントラクト

**ビットコイン**は基本的に「BTC」という単位を取引するトランザクションをUTXOで管理。  
取引内容にプログラムコードを含めることも可能。

BTCの取引(トランザクション)を記録

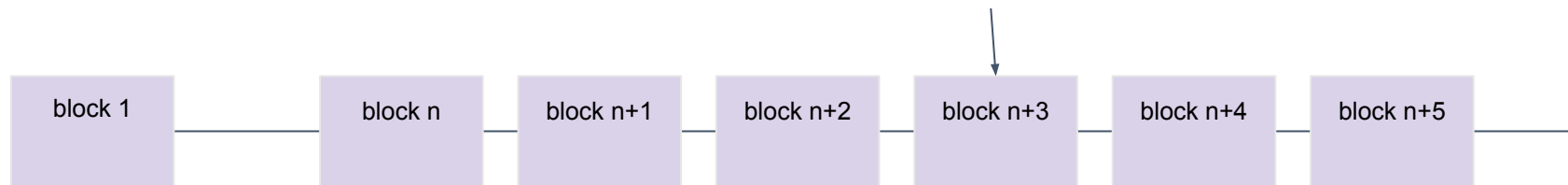


# VM型スマートコントラクト

**イーサリアム**は**Ethereum Virtual Machine (EVM)** というプログラムの実行環境を備えている。

スマートコントラクトアカウント毎に固有の、不変なコードを持つことができ、トランザクションを起点として複雑な処理を実行することが出来る。

- 単純なETHの取引を記録
- スマートコントラクトへのインプットを記録、実行



# VM型スマートコントラクトだけにとできること

1. コントラクトが第一級市民（First-class Citizen）としてアセットを保有できること。（アカウントモデルがUTXO型と異なる）
2. 後状態が矛盾する2つのトランザクションについて、どちらが先になったとしても前状態を特に限定せずに実行可能なので、複雑なプログラムを記述してもスムーズな実行が期待できること。

これらの性質により、「書きかけの契約書」のようなものをインターネット上に放流するような状態が作れる。



## スマートコントラクトの処理系

- opcodesを読み取ることでスマートコントラクトを保存したり、実行したりできる。 (e.g. *ADD*, *SUB*, *CALL*)
- 32bytesずつのバイナリを仮想メモリや仮想ストレージに保存するスタック型のプログラム。
- 1コントラクトの最大プログラムサイズが24.576KBであり、極めて小さい。
- 1トランザクションで3000万ガスまで処理できるが、このガスという単位は例えばストレージへ32bytes保存するだけで21000ガスを消費する。

# EVMのアカウントモデル

EVMには以下2つの種類のアカウントの概念がある。\*アドレスの規格は共通

- **EOA** (Externally Owned Account / 外部所有アカウント)

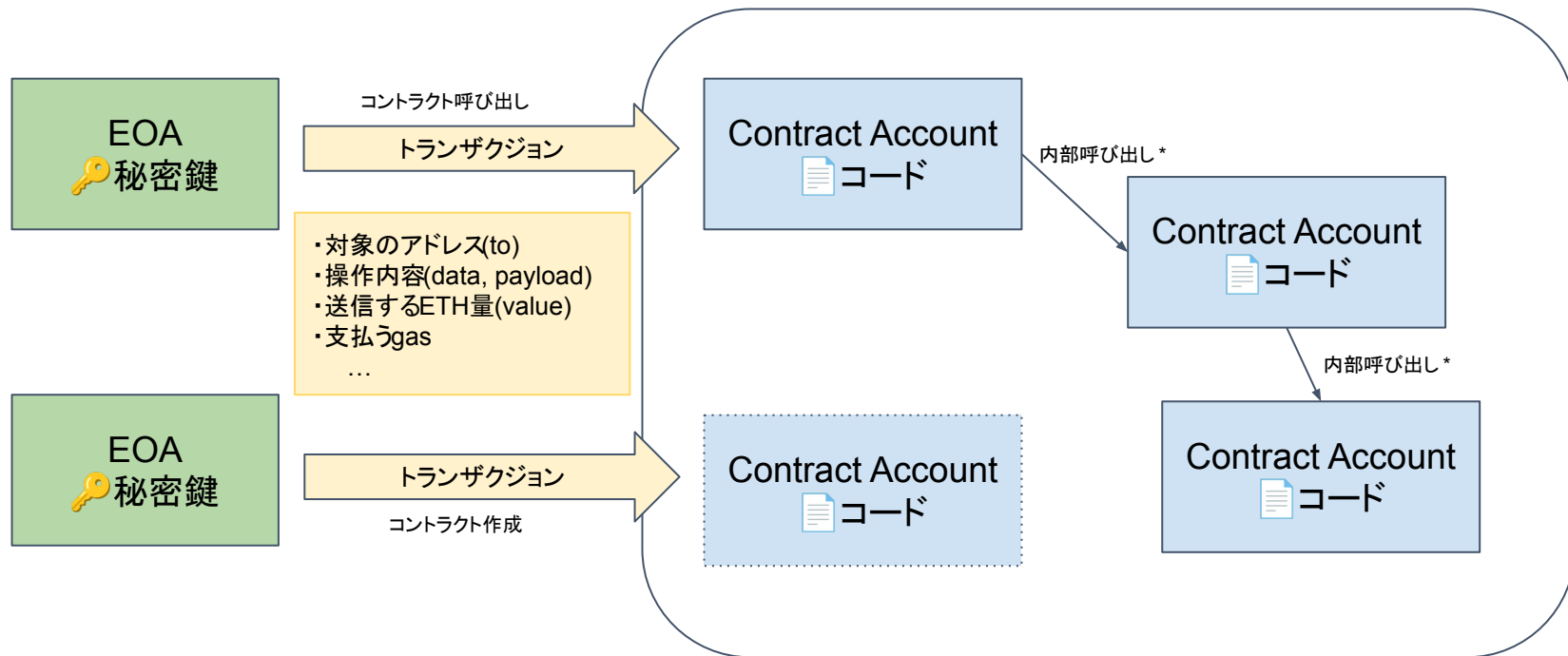
秘密鍵に紐づいたアカウントで、署名を通じてトランザクションを発行できる  
コードは持てない（今後のEthereumアップグレードで変更される可能性はある）

- **CA** (Contract Account)

コードを持つことのできるアカウント

秘密鍵を持たないため、署名は作れないが、EOAからのトランザクションを受け取り、他のコントラクトアカウントと内部トランザクション（メッセージ、コール）を通じてやりとりすることが出来る

# EVMのアカウントモデル



内部呼び出し\* = call, message, internal tx

トランザクションが取り込まれ、Ethereumのグローバルステートを変更するために必要となるETH。

トランザクションからスマートコントラクトを動かしたり、デプロイしたりする際に消費され、バリデータに報酬となったり、Burnされて消失する。

ネットワークダウンを防ぐためにも必要。

相当するETHの単位は主にGweiが使われる。

ETH	1 ether
wei (最小単位)	$10^{(-18)}$ ether = 0.000000000000000001 ether
Gwei (Giga wei)	$10^{(-9)}$ ether = 0.000000001 ether

ここで最新情報を確認できる。

<https://www.blocknative.com/gas-estimator?gasType=ethereum>

実際に消費されるGasはネットワークの混雑具合などによって変動するGas Price (先述のMaxFee) とGas Amountが掛け合わされたものになる  
opcodeごとにMinimum Gasが定められており、処理の複雑さによってGas Amountは変化する  
特に、ストレージへの読み取り・書き込みは多くのGasが必要になる

参考

ETHの送信:21000Gas、ガス代:30Gweiだと0.0006ETHかかる

## ブロックガスリミット

ブロックに取り込むことのできるトランザクションのガス量の総和にはリミットが設けられている  
これは、無限ループ実行によるネットワークの遅延・破壊を防ぐための機構である  
現在は3000万Gasに設定されている

# スマートコントラクトのクセ

1. EOAからしかTxを発生させられない  
→スマコン系内のスケジューラーは不可
2. 決定論性を破るプログラムが書けない  
→外部への通信は不可、オラクルが必要になる
3. ログインという概念がない（署名検証による認証）
4. 32bytesのデータを基本単位としたバイナリプログラミング  
→計算機資源の制約、Stack too deep、Gas golfing

# スマートコントラクトとセキュリティ

ラテン語のsē-cūra（気にかけること無しで）が語源。

**仕様に穴がない前提で、仕様通りに実装されていること。**

しかし、完全あるいは十分な仕様が事前にシステムに与えられることは実際は多くなく、かつ攻撃者は常に仕様の穴を探す努力をしている。

そのモデルが成立するためのassumption (前提)を利用者に明示することで安全にシステムを利用してもらうことが可能であり、その説明の中には攻撃者モデルも含まれているべきである。



例えばNakamoto Consensusを採用するパブリックチェーンはシビル耐性アルゴリズム(PoW, PoS, dPoS等)において51%攻撃が継続することによってネットワークが麻痺し、この状況で利用したユーザーはreorgや二重支払いによって資産(状態)のsafetyが守られないことがある。

例えばEthereumのスケーリングソリューションであるOptimistic Rollupにおいては1週間のchallenge periodの内にサーバー管理者の不正の証拠の提出が必要であるし、分散ファイルシステムであるIPFSもGarbage Collectionにデータを破棄されないようにheart beat処置を実施しなければならない。(SafetyとLiveness)

## コントラクトの脆弱性を検出するサービス

サービス形態	概要	料金	監査人数
ファーム (OpenZeppelin)	専門事業者へ依頼	低～高	数人
コンテスト (Code4rena)	不特定多数の監査人	中～高	数十～数百人
バウンティ (Immunefi)	深刻度に応じた賞金	0～高	0～数百人



# スマートコントラクトと法制度

- **銀行**

現金をお預かりして、銀行口座にその数値を記録する。銀行が残高を更新する権利を持ち、現金を運用することも許可されている。

- **Fintechスタートアップ**

サーバーで残高管理をし、銀行口座で決済を完了させる。サーバーの値を運営が更新する権利を持つが、預かり金の運用は制限されている。

- **スマートコントラクト**

システムの残高すなわち保有現金であり、自分以外が残高を更新することも運用することもできない。

# 利益を約束すること

第一種金融商品取引業（金融商品取引法第28条第一項）

1. 取得勧誘の相手方が50名以上
2. 取得勧誘＝予定された資金調達に先立ってなされる情報及び声明の公表並びに広報活動であって、公衆の心理を調整し、又は発行者若しくはその証券に対する公衆の関心を喚起する効果を有するもの（米国SEC）
3. NFTも利益の約束をすると証券扱い

# 賭博及び富くじに関する罪：刑法185条-187条

簡単に言うとギャンブルは規制業種であり、限られた主体しか営業できない。

オンラインカジノは取締りにくい海外のギャンブル運営主体に日本人が能動的にアクセスしにしているという状況。

日本在住の開発者がギャンブル系のスマートコントラクトを公開することは違法行為なので注意が必要。

# スマートコントラクトと課税

都度スマートコントラクトの設計ごとに実質的な税解釈を与えなければならない。

税務的に不確実な要素は「税率が高い方」を想定して活動せねばならないので不利益になる。

税務的に「枯れた」統一規格を制定するとやりやすい。

# 匿名性と制裁：Virgil Griffithの事例

2019年、北朝鮮にEthereumの基本的な仕様を教えに渡航したEthereum FoundationのSpecial Project担当だったVirgil Griffith (当時39歳)は懲役5年の刑を言い渡された。



# 匿名性と制裁：Tornado Cashの事例

2022年、匿名送金プロトコルであるTornado Cashは数千億円にもものぼる犯罪被害を資金洗浄することを幫助(特に北朝鮮のハッカーグループ Lazarusについて)したとして、開発者のAlex Pertsev (29)をアムステルダム市警が逮捕した。

Tornado CashはOFAC(米国外国資産管理局)のブラックリストをプロトコル内で参照し利用できないようにしていたが、そのリストに入っていない Lazarusおよび他の犯罪収益を幫助したことにより、強権的な対応となった。

Tornadoのデプロイは分散的に行われているが、それでも開発者は逮捕された。

# 匿名性と制裁：Tornado Cashの事例

その後、米国民を雇用しているプロトコルはそのメンバーの身の潔白を示すためにOFACのBlacklistをUIに埋め込んだり、プロトコルに埋め込んだりしているが、Tornado Cash自身もOFACのBlacklistはもともとプロトコルレベルで参照しており、そのリストにまだ追加されていないLazarusと思わしきアカウントに利した結果、開発者が制裁を受け、逮捕となっている。

根本的な回避方法は発明されていない。

# アップグレードビリティとパブリックコメント

meta contractをはじめとするアップグレードビリティ手法は担当者が資産を引き抜く関数を追加できる能力を持っていることから、担当者はカスタディ業者に近い性格があるが・・・

2020年4月3日 パブコメ 11番

10	<p>秘密鍵の一部を預かり、利用者の指示により、利用者の暗号資産の移転を補完するために、署名を追加する業務は、資金決済法第2条第7項第4号に規定する「他人のために暗号資産の管理をすること」に該当しないとの理解でよい。</p> <p>また、秘密鍵を全て預かるものの、暗号資産の移動の指示は利用者のみが行え、個々の移動指示に署名を加えるだけしかできない装置をもって業務を行えば、当該業務は資金決済法第2条第7項第4号に規定する「他人のために暗号資産の管理をすること」に該当しないか。</p>	<p>個別事例ごとに事態に即して実質的に判断されるべきものではありませんが、事業者が利用者の暗号資産を移転するために必要な秘密鍵の一部を保有するにとどまり、事業者の保有する秘密鍵のみでは利用者の暗号資産を移転することができない場合には、当該事業者は、主体的に利用者の暗号資産の移転を行い得る状態にないと考えられますので、<b>基本的には、資金決済法第2条第7項第4号に規定する「他人のために暗号資産の管理をすること」に該当しない</b>と考えられます。</p>
11	<p>事業者、又は事業者と委託先の両者をあわせて、マルチシグアドレス(又は同等の機能を持ったスマートコントラクトで実現されたコントラクトウォレット)の暗号資産の移転に必要な数に満たない数の秘密鍵のみを保管し、暗号資産の移転のために必要な署名のうち一部のみの署名を行う場合、事業者は、主体的に利用者の暗号資産の移転を行い得る状態になく、「他人のために暗号資産の管理をすること」には該当しないとの理解でよい。</p>	<p>また、事業者が利用者の暗号資産を移転することができ得る数の秘密鍵を保有する場合であっても、その保有する秘密鍵が暗号化されており、事業者は当該暗号化された秘密鍵を復号するために必要な情報を保有していないなど、当該事業者の保有する秘密鍵のみでは利用者の暗号資産を移転することができない場合には、当該事業者は、主体的に利用者の暗号資産の移転を行い得る状態にないと考えられますので、基本的には、同号に規定する「他人のために暗号資産の管理をすること」に該当しないと考えられます。</p>
	<p>事業者が保管する秘密鍵が暗号化されており、事業者は復号・使用することができず、署名時</p>	

1. スマートコントラクトは強力であるがゆえに、簡単に法に抵触する
2. システムを運用するだけでなく、利益を約束したり、特定の主体に知識を教えたり、プログラムを開発するだけで刑罰の対象になりうる
3. 独断でシステムを設計したりビジネスを設計してはいけない。必ず先人に相談しよう。



休憩（5分）



# スマートコントラクトのDevOps

改ざん不可能なプログラムが動き続けるのがブロックチェーンの良いところ

しかし、セキュリティの向上や法令改正に従うためには、アップグレード  
ビリティが必要

アップグレード権限を適切に管理しながら、コントラクトの質を向上させていくのがのぞましい（最終的に放棄できる）

Solidityにおいてもアップグレード手法の標準化が試みられている

あるテンプレートが存在し、それを基に複数のコントラクトが立ち上がるシチュエーションが想定される

テンプレートの子（Clone）に共通の振る舞いが期待されるとき、アップグレードがそれらすべてに反映されることがのぞましい

基の機能を保持しつつも、カスタムのために親から抜ける（Duplicate）機構もあるとよい

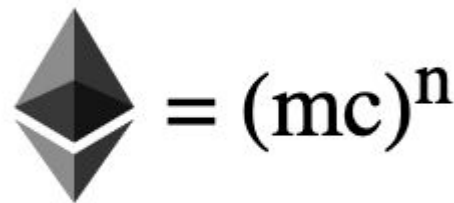


# Meta Contractとは

# What is "Meta Contract"?

"metacontract/mc" is a UCS-based framework for creating highly maintainable and cost-effective smart contracts.

- $Eth = (mc)^n$
- $mc = (bundle)^m$
- $bundle = schema * (function)^q$
- $schema = (vertical\ domain)^p$
- $vertical\ domain = A\ struct\ tree\ in\ a\ storage$



- 制限された計算資源
  - 24,576 bytesのコントラクトサイズ上限
  - 1024までのスタック
  - 30Mのブロックガスリミット
- 不変性
  - バグがあっても基本的には修正できない
- 外部通信不可→Oracleどうする？

and more and more...

# コントラクトを開発する上で対処すべきこと

- コードを高品質なものとするために自動テストが必要
  - 長くなりがちなテスト実行時間への対処
- 機能/非機能要件の追加によって複雑になるコードベース
  - 複雑化するファイルやディレクトリ構成への対処

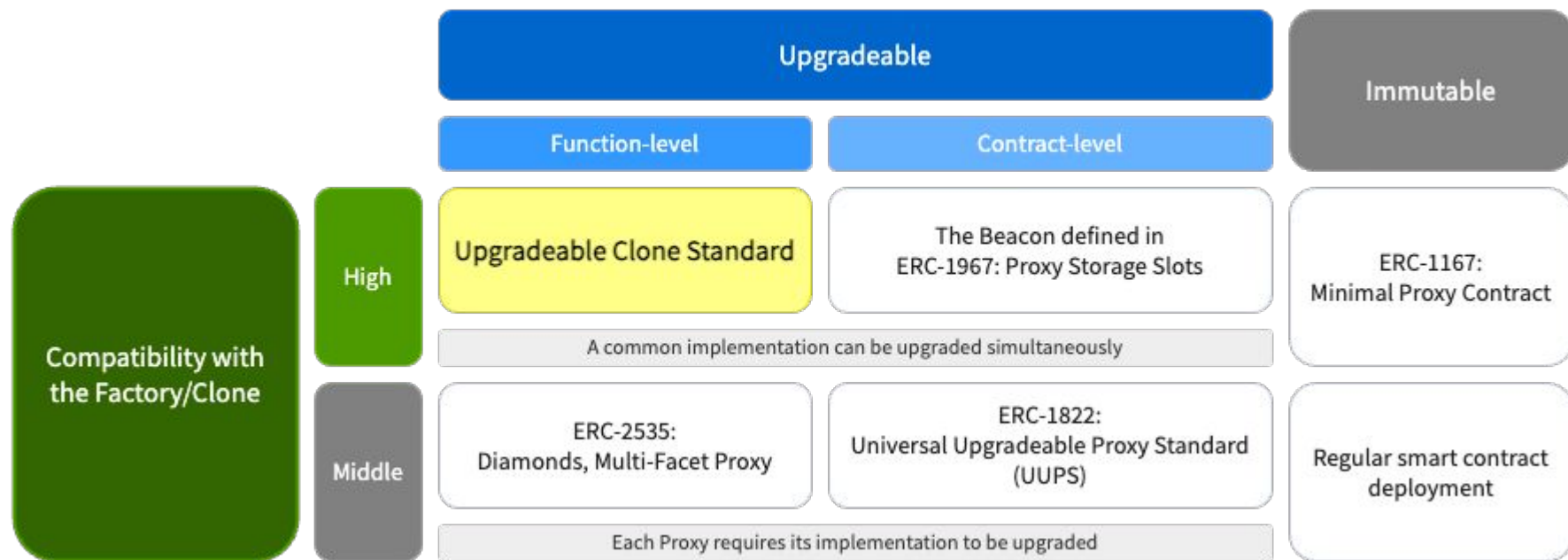
and more and more...

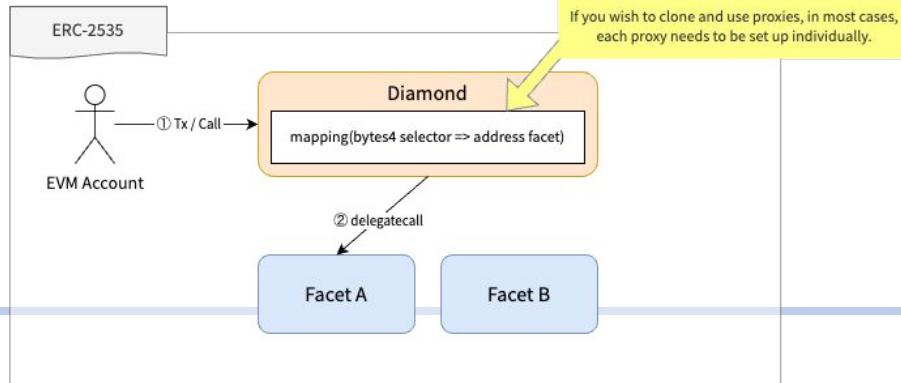
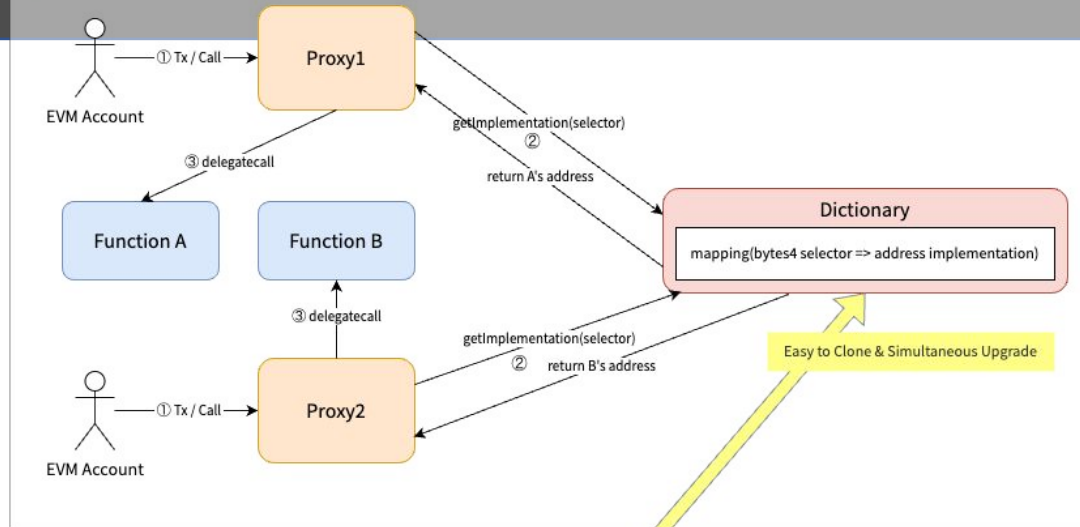
- Function-Level Upgradeability 

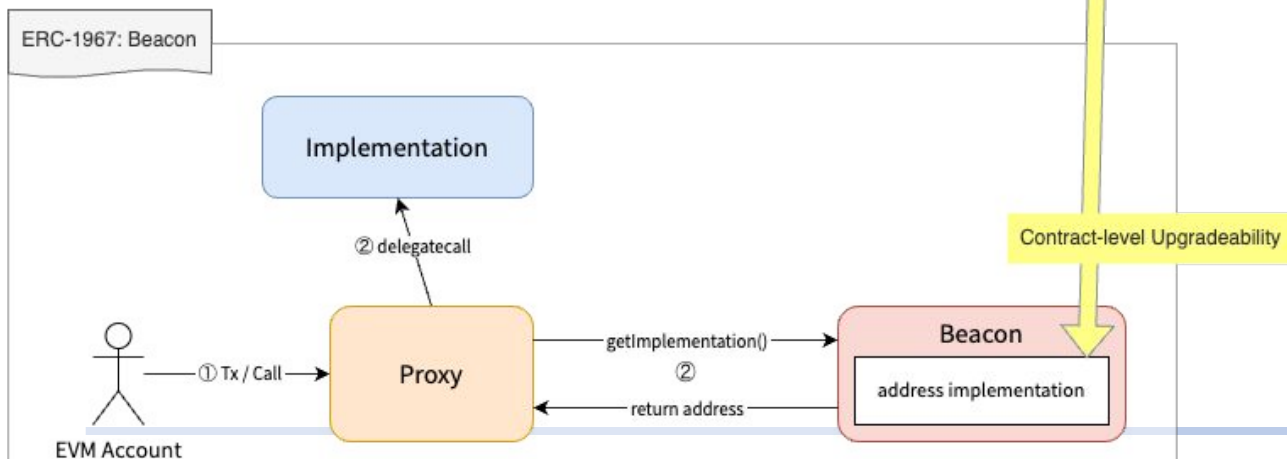
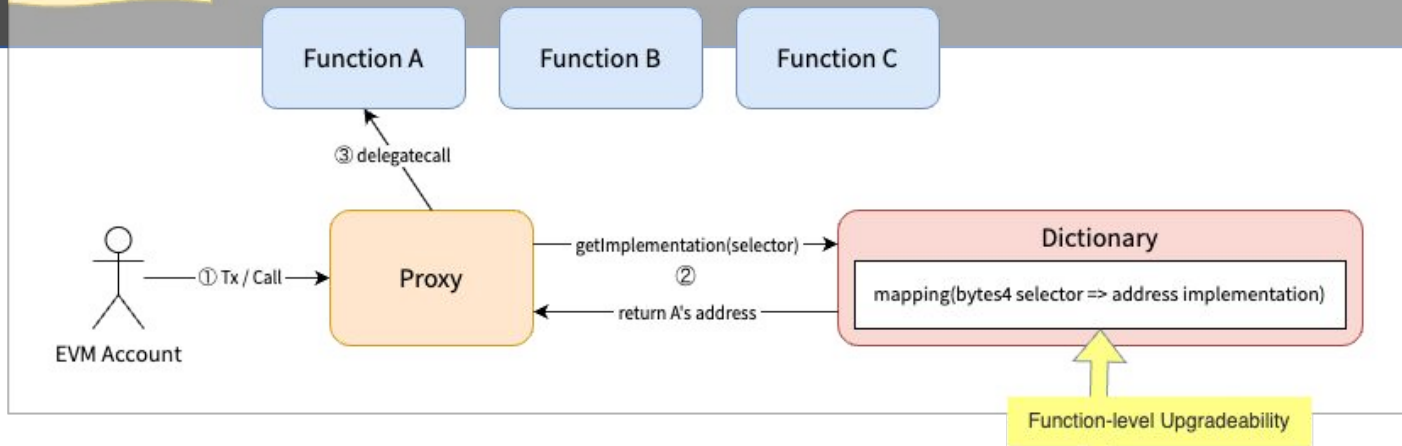
Improve development & maintenance flexibility

- Mass Production & Unified Management 

Easily handle use cases like Contract Wallets and DAOs  
where non-devs need individualized contract states









# Getting Started

- prerequisite: *foundry*
- フレームワークのインストール

```
forge init <Project Name> -t metacontract/template
```

- Functionsの設定
  - Standard Functionsを組み合わせて使う
  - オリジナルのCustom Functionを開発して使う

# Standard Functions

cloneやsetAdminなど、汎用的に使われるであろう関数を metacontract/mcの標準関数群として提供し、それらを組み合わせるだけでシンプルなコントラクトであれば作成可能にする (OZ Contractsのようなイメージ)

# Custom Function / Bundle

Standard Functionsに含まれていない機能を利用したい場合、開発者は独自の関数を簡単に追加することが出来る。命名規則など、いくつかの条件さえ守れば、基本的にはアップグレードやそれに伴うストレージコンフリクトなどの危険性もほとんど意識せず機能追加することが可能。

また、それらのFunctionsをVertical DomainごとにまとめたものをBundleと呼ぶ。

# Custom Functionを開発する

State Fuzzingを利用することで、TDDフレンドリーに最小限の  
Unit Testで開発イテレーションを回すことが可能

## 「Is this a proxy?」の仕組み

ERC-1967互換のコントラクトにおいて

- Logic→そのアドレスのABI
  - Beacon→BeaconからgetImplしたアドレスのABI
- をread/write as Proxyとして表示させる

DictionaryがBeaconなので、参照したい関数をまとめる  
**Facadeコントラクト**をgetImplに設置する

# Etherscan Compatibility

```
contract CounterFacade is ICounter {  
    function increment() external {}  
    function initialize(uint256 initialNumber) external {}  
    function setNumber(uint256 newNumber) external {}  
}
```

```
function deployCounter(MCDevKit storage mc, uint256 initialNumber) internal returns(MCDevKit storage) {  
    mc.init(bundleName());  
    mc.use("Increment", Increment.increment.selector, address(new Increment()));  
    mc.use("Initialize", Initialize.initialize.selector, address(new Initialize()));  
    mc.use("SetNumber", SetNumber.setNumber.selector, address(new SetNumber()));  
    mc.set(address(new CounterFacade()));  
    mc.deploy(abi.encodeCall(Initialize.initialize, initialNumber));  
    return mc;  
}
```

# Etherscan Compatibility

✔ **Contract Source Code Verified** (Exact Match)

Contract Name: **ERC7546ProxyEtherscan**

Optimization Enabled: **Yes with 200 runs**

Compiler Version: **v0.8.24+commit.e11b9ed9**

Other Settings: **paris EvmVersion**

 **Contract Source Code** (Solidity [Standard](#) [Json-Input](#) format)

Open In ▾

More Options ▾

File 1 of 12 : ERC7546ProxyEtherscan.sol

 Is this a proxy?

Code Read Contract Write Contract Read as Proxy **Write as Proxy**



 **ABI for the implementation contract at 0x745daca792e4d28e1487ea50e45979323406cc6, using the EIP-1967 Beacon Proxy pattern.**

Learn more about proxy contracts in our [Knowledge Base](#) 


 Connect to Web3

[\[Expand all\]](#) [\[Reset\]](#)

1. increment (0xd09de08a)

  →

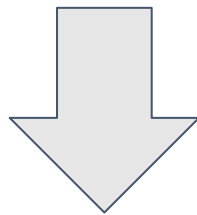
2. initialize (0xfe4b84df)

  →

3. setNumber (0x3fb5c1cb)

  →

- Reth ExEx で state diffを取得する
- storage layoutのうちのどこが更新されたかを同定する
- GraphAPI Serverに差分を適用する
- GraphAPIを用いて検索等を行えるようになる



TheGraphハンドラの実装が不要



metacontract.sol ▾

## StorageScheme.sol

The ``StorageScheme.sol`` will define the storage layout that supports the operations described in the scoring facet. This includes storing logs, protocol data, and configurations related to the logging and scoring mechanism. We'll use the ERC-7201 Namespaced Storage Layout for organization and compatibility.

solidity

Copy code

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

library StorageScheme {
    struct LogConfig {
        uint checkpointDuration;
        address veAddr; // Voting escrow address for checking balances
        address tokenAddr; // Token address for minting rewards
    }

    struct LogStorage {
        mapping(uint => mapping(address => uint256)) scores; // Mapping of epoch to u
        mapping(uint => uint256) sum; // Mapping of epoch to sum of scores
        mapping(uint => uint256) mintables; // Mapping of epoch to mintable amount
        LogConfig config;
    }
}
```

metacontract.sol ▾

## Folder Structure

The folder structure is designed to encapsulate the functionalities and storage scheme described in the Yamato meta contract's scoring facet. It adheres to the guidelines of isolating logic in functions and structuring tests accordingly.

scss

Copy code

```
/src
  /scorebundle
    /functions
      ScoreLib.sol
      PointMathLib.sol
      SetLogConfigProtected.sol
      PopulateMintableProtected.sol
      Mint.sol
    /storages
      StorageScheme.sol
    /_internal
      // Shared state mutation logics that are used across different functions
      // This could include libraries for handling specific storage manipulation or c
      // that are common across the yamato meta contract's facets
  /tests
```

solidity

Copy code

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import "../storages/StorageScheme.sol";
import "../_internal/PointMathLib.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

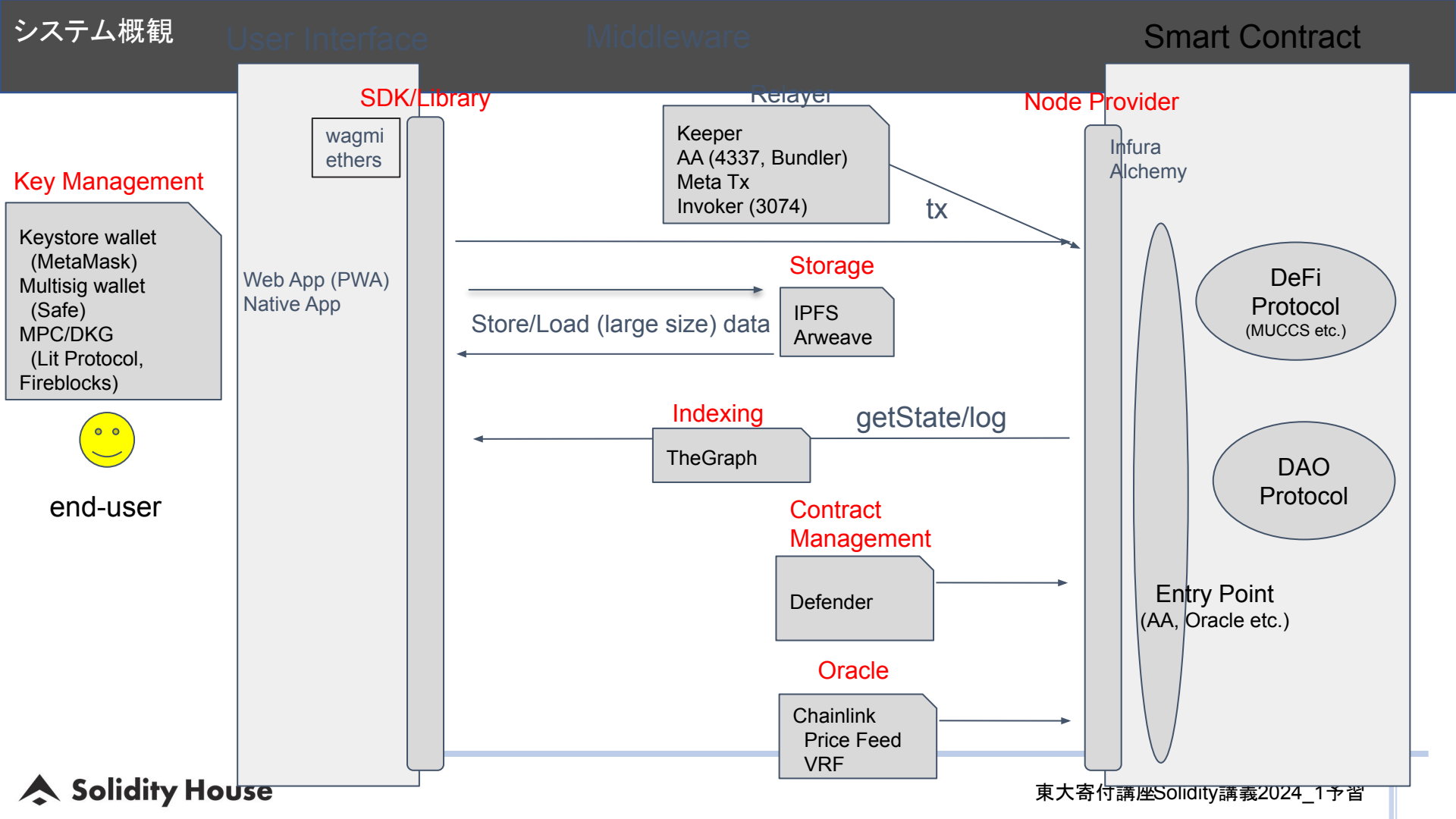
library ScoreLib {
    using StorageScheme for StorageScheme.LogStorage;

    /// @notice Logs an action with the associated points, updates scores and sums ac
    /// @param action The name of the action performed.
    /// @param abiParams The parameters of the action encoded in ABI format.
    function log(string memory action, bytes memory abiParams) external {
        StorageScheme.LogStorage storage log = StorageLib.log();
        uint epoch = block.timestamp / log.config.checkpointDuration;
        uint point;

        if (keccak256(abi.encodePacked(action)) == keccak256(abi.encodePacked("borrow
            point = borrow(abiParams);
```

# 開発環境セットアップ

- 推奨エディタ
  - [Cursor](#)
  - [Visual Studio Code](#) (VSCode)
- スマートコントラクト開発環境
  - [Foundry](#)
- スマートコントラクト開発フレームワーク
  - [Meta Contract](#)
  - GPTs: MC GPT / Schema



Key Management

- Keystore wallet (MetaMask)
- Multisig wallet (Safe)
- MPC/DKG (Lit Protocol, Fireblocks)



end-user

