

東大寄付講座Solidity講義2024\_2実装

# DAOの簡易実装


株式会社 Ecdysis CEO / Solidity House 運営

落合 渉悟

- 事前準備: 開発環境セットアップ
- 開発演習①: シンプルな提案と投票による状態遷移
  - 状態遷移提案（可決で数値インクリメントのみ）
  - 単投票（多数決）
  - 単集計
  - 状態遷移実行

# 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



# 開発演習①

## シンプルな提案と投票を通じて状態遷移するコントラクト

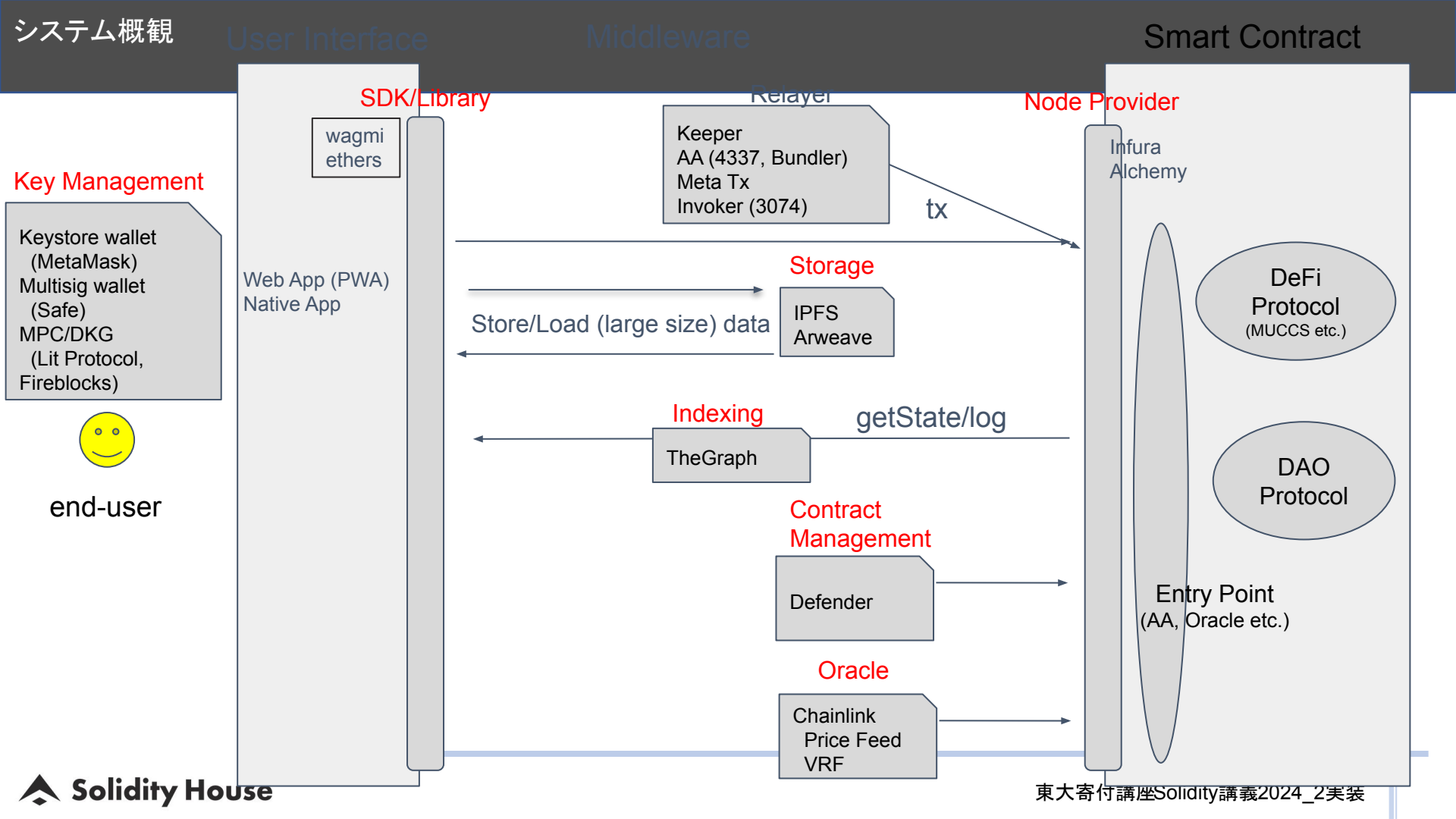
# 作っていくもの

---

シンプルな提案と投票を通じて状態遷移する（DAO）コントラクト  
(e.g., MolochDAO, NounsDAO, etc...)

完成例：

<https://github.com/ecdysisxyz/simple-dao>



Key Management

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



end-user

Web App (PWA)  
Native App

SDK/Library

wagmi  
ethers

Relayer

Keeper  
AA (4337, Bundler)  
Meta Tx  
Invoker (3074)

Node Provider

Infura  
Alchemy

Storage

IPFS  
Arweave

Indexing

TheGraph

Contract Management

Defender

Oracle

Chainlink  
Price Feed  
VRF

DeFi  
Protocol  
(MUCCS etc.)

DAO  
Protocol

Entry Point  
(AA, Oracle etc.)

tx

Store/Load (large size) data

getState/log



# プロジェクトの立ち上げ

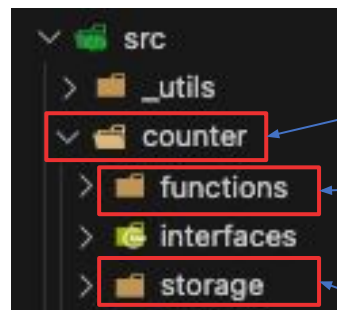
---

```
forge init <project name> -t metacontract/template
```

例) 

```
forge init simple-dao -t metacontract/template
```

# プロジェクトの構成を確認する



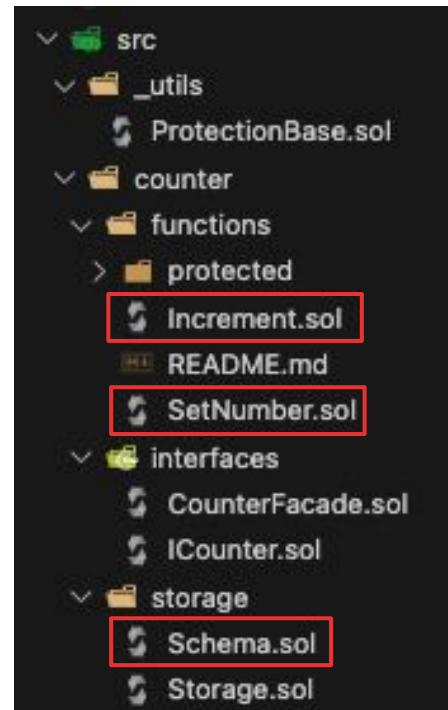
**Bundle**

**Functions**

Bundleの機能(振る舞い)

**Storage**

Bundleのデータ構造



Meta ContractのフレームワークではBundleという  
単位でコントラクトを管理/開発します

## Q. なぜそのような概念のもと開発するの？

A. ソフトウェアを複雑にしていくのにも技術が必要で、それをツールの側でやっておいてあげたいから

=> おさらい

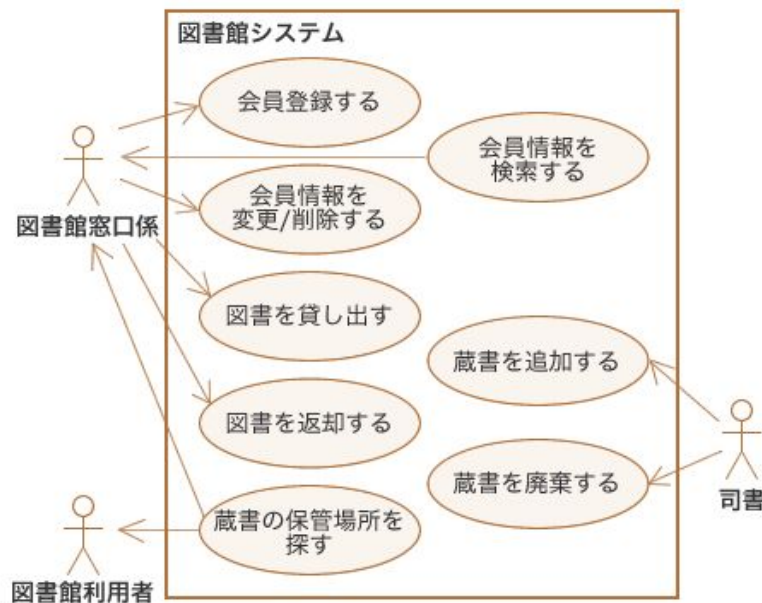
Upgradeability, Clonability, Contract Size Limit, Test Execution Time

# コントラクト (Bundle) の構造を単純化して理解する

- どのような機能が必要か？
  - Meta ContractにおけるFunctions
- どのようなデータ構造（モデル）が必要か？
  - Meta ContractにおけるSchema.sol

# 設計の道へ - 必要な機能の洗い出し

## ユースケース図が便利



[https://www.mamezou.com/sites/default/files/2020-10/modewaza05\\_sp\\_002\\_005\\_f01.gif](https://www.mamezou.com/sites/default/files/2020-10/modewaza05_sp_002_005_f01.gif)

# 設計の道へ - 製図ツールの紹介

- [Mermaid.js](#) / [Mermaid Live Editor](#)

専用の記法を使ってテキストで書くと図を描画してくれる

GitHubが標準で描画対応していたり、LLMの支援を受けやすいが清書できないので複雑な図にはあまり向いていない

- [draw.io](#)

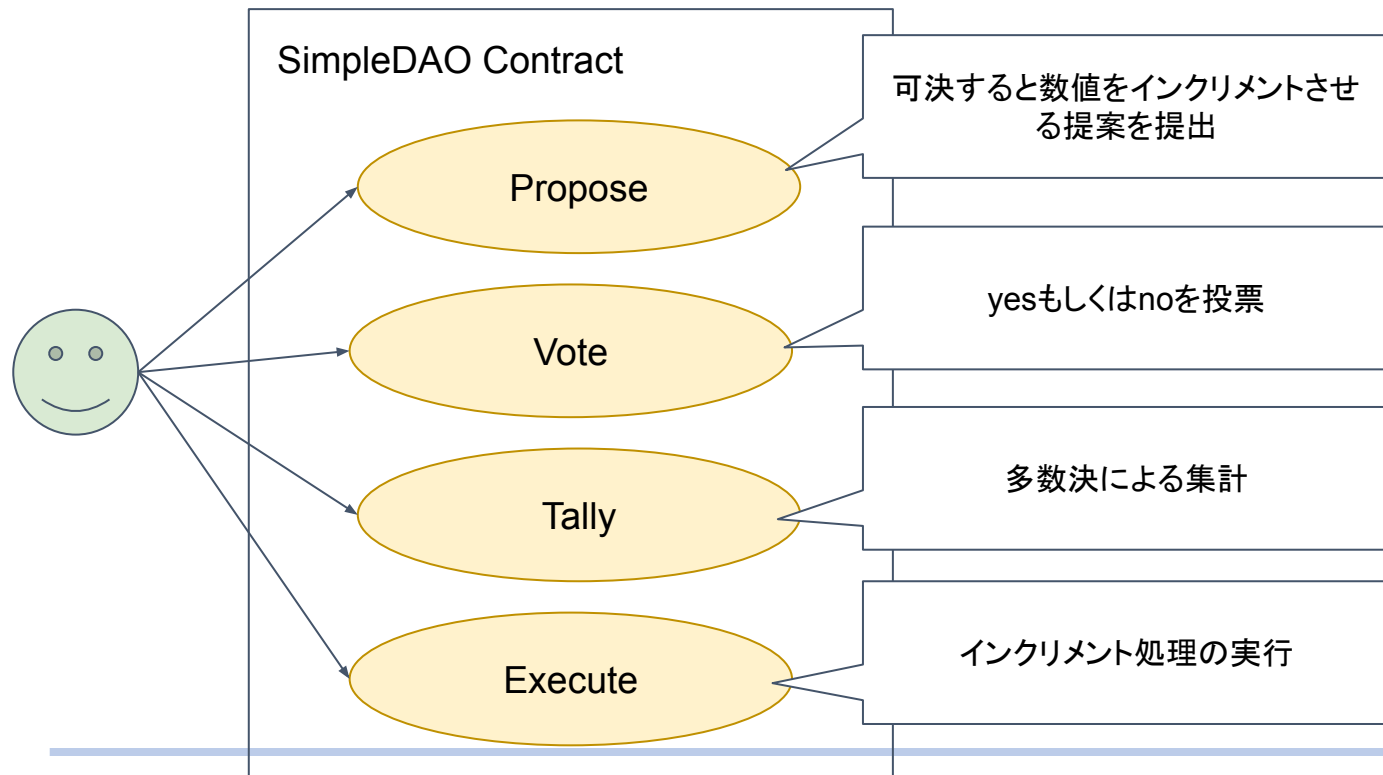
Google Drive等と連携させて共同編集も出来るクラウドサービス

LLMと対話しながらユースケース図を描いてみましょう

以下の機能を有するものとする

- 提案（可決すると数値インクリメントする）
- 投票（yes/noで）
- 集計（多数決）
- 実行（数値インクリメントの状態遷移を実行）

# 設計の道へ - 必要な機能の洗い出し





# 設計の道へ - 必要な機能の洗い出し

必要なFunctionsのファイルを作成する

src/simple-dao/functions/

- Propose.sol
- Vote.sol
- Tally.sol
- Execute.sol

# 設計の道へ - 必要な機能の洗い出し

コントラクト全体がどのような振る舞いをするかを記述するFacadeコントラクトファイルを作成する

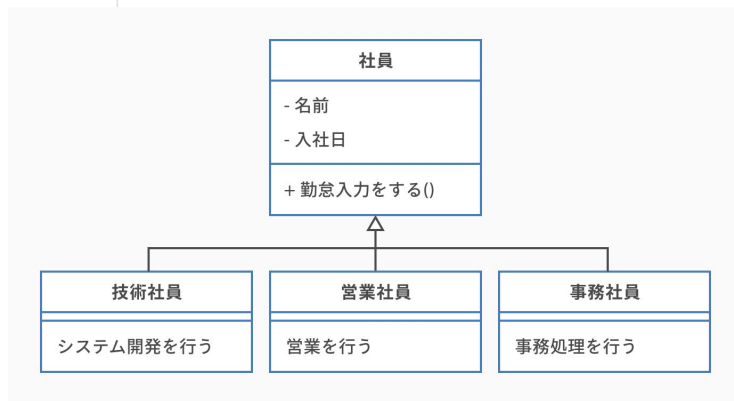
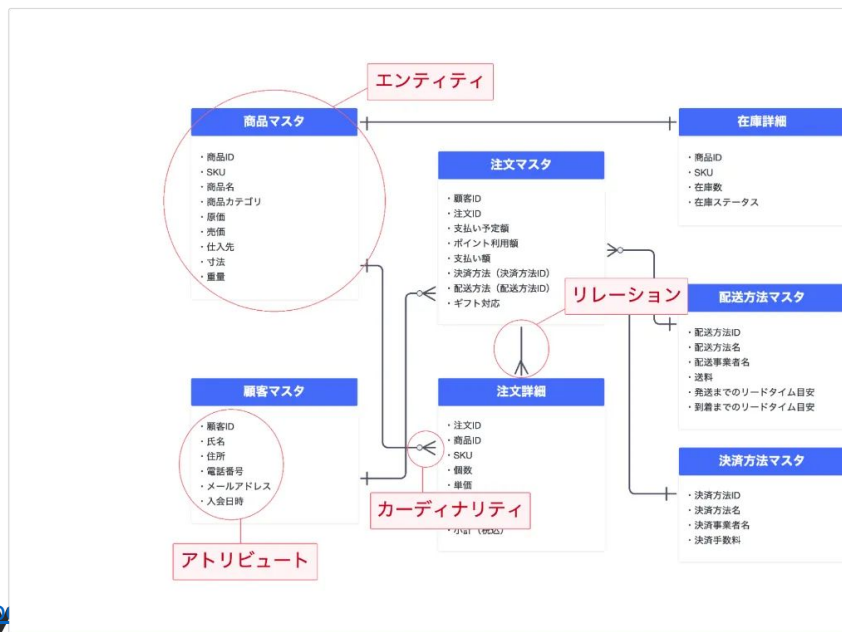
```
src/simple-dao/interfaces/Facade.sol
```

# 振る舞いとデータ構造 - どこでデータ構造を定義するか

- どのような機能が必要か？
  - Meta ContractにおけるFunctions
- どのようなデータ構造（モデル）が必要か？
  - Meta ContractにおけるSchema.sol

# 構造のモデリング - ER図

どのような状態（データモデル）が必要かを考える  
→ER(Entity Relationship)図やクラス図が便利

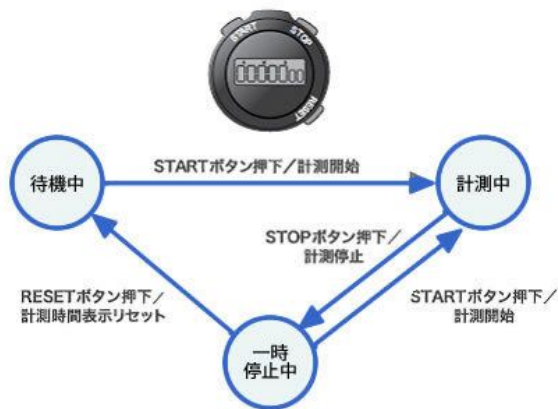


<https://cacoo.com/ja/blog/how-to-write-class-diagram/>

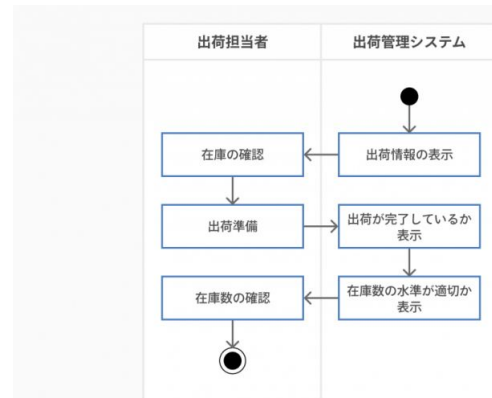
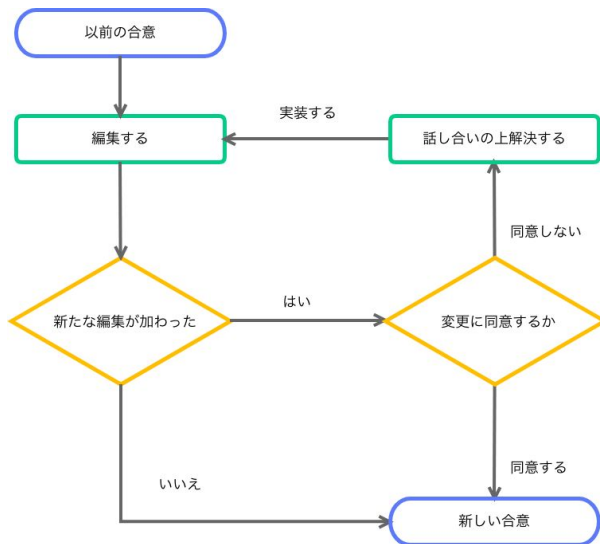
# 振る舞いのモデリング - ステートマシン図等

必要な情報の精度を上げる

→状態遷移図やフローチャート、アクティビティ図などが便利



[https://gihyo.jp/dev/serial/01/test\\_up/0004](https://gihyo.jp/dev/serial/01/test_up/0004)



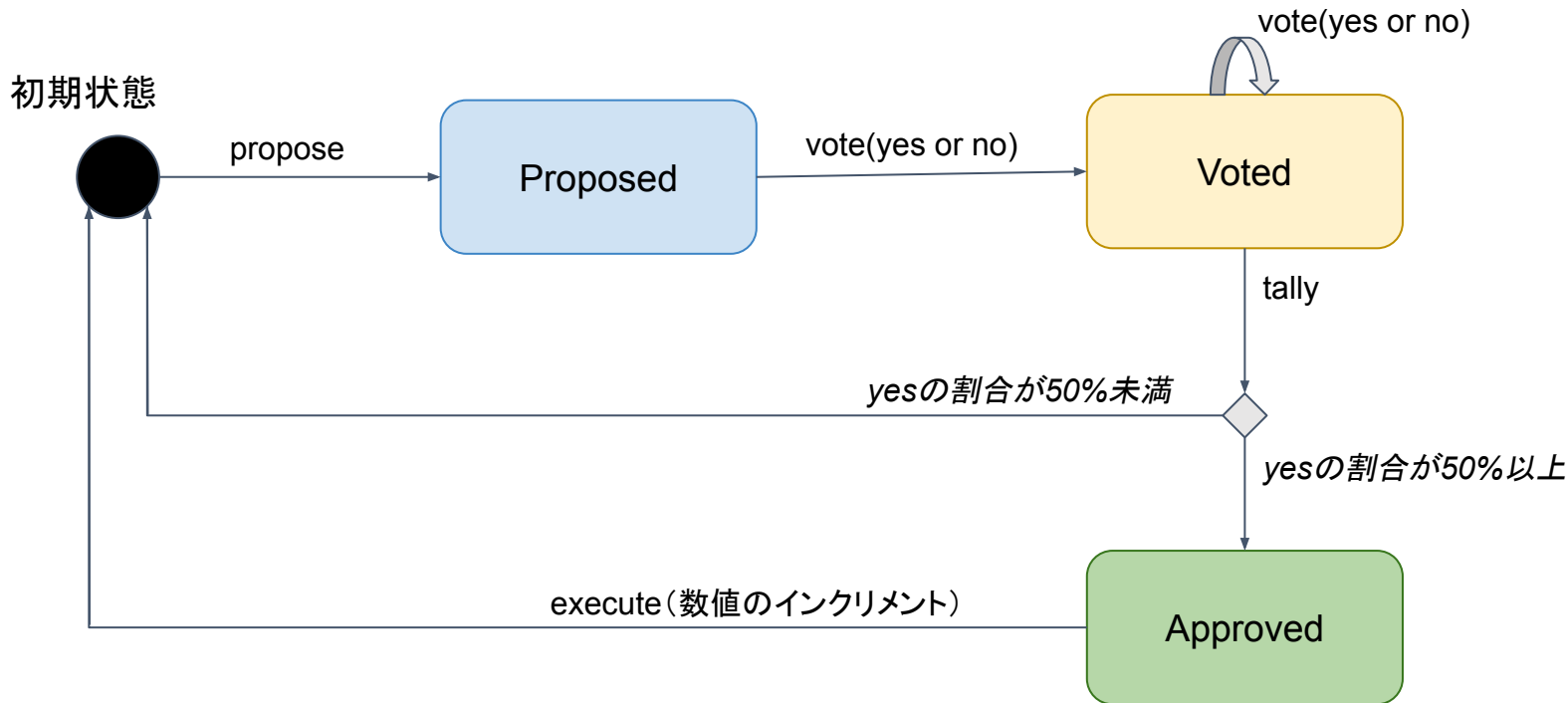
<https://cacao.com/ja/blog/what-is-activity-diagram/>

LLMと対話しながら状態遷移図を描いてみましょう

以下の状態を遷移するものとする

- 初期状態（何も提案されていない状態）
- 提案が提出された状態
- 提案に対して投票が行われた状態
- 投票が可決し、実行可能になった状態

# 状態遷移図の例



# 粗いモデリングを通して見えてくること

以下のようなデータが必要そうなことがわかる

- （提案と投票によって管理される）数値
- 提案
- 提案に対する投票状況があり
- 提案に対する投票結果がある

>> LLMとの対話を通じてER図を描いてみましょう



```
library Schema {  
    /// @custom:storage-location erc7201:SimpleDAO.SimpleDAOState  
    struct $SimpleDAOState {  
        uint256 number;  
        Proposal[] proposals;  
    }  
  
    struct Proposal {  
        uint256 totalVoteCount;  
        uint256 approvalCount;  
        bool isApproved;  
    }  
}
```

# Schemaファイルの作成

---

1. Schema.solファイルを作成する

src/simple-dao/storage/Schema.sol

# 関数からデータへアクセスする (Storage.sol)

1. データモデルに従ってSchema.solファイルを実装する
2. FunctionsからSchemaを利用するためのStorage.solを実装する
3. 状態遷移図やユースケース図を元に各Functionを実装する  
(必要に応じてそれらの図も更新する)
  - a. Propose.sol
  - b. Vote.sol
  - c. Tally.sol
  - d. Execute.sol

# Storageファイルの作成

1. Storage.solファイルを作成する  
src/simple-dao/storage/Storage.sol
2. ERC7201のスポットを決める  
sh calc-storage-location.sh SimpleDAO.SimpleDAOState

# 動作確認とその自動化 - 単体テスト (a.k.a. 自動テスト)

ソフトウェアテストには粒度の概念がある

1. ユニット（単体）テスト

→ Function毎のState Fuzzing Test

```
import {MCTest} from "@mc/devkit/Flattened.sol";
```

2. インテグレーション（統合）テスト

→ デプロイスクリプトやMockを利用する

# テストを通した設計（テスト駆動開発・TDD）

ソフトウェアテストは仕様としての役割もある

- どのような前提条件で、
- どのような操作を行なったとき、
- どのような結果になるか

1. 正常系テスト
2. 異常系テスト

# テストとリリースプロセス（安全なデプロイのために）

作成したコントラクトを実際にネットワークにデプロイ（展開して利用可能な状態に）する流れの一例：

1. ローカルネットワーク環境での自動テスト
2. テストネット環境にデプロイし、[システム全体](#)を統合したテスト
3. 既にリリース済みの場合や、外部サービスとの連携がある場合などにはFork環境でのテスト
4. ローンチ前テスト
5. システムモニタリング

ネットワークにデプロイするためのスクリプトを用意する

- **script/DeployLib.sol**

デプロイするコントラクトの内容を記述するファイル

- **script/Deploy.s.sol**

デプロイするためのネットワーク情報やデプロイヤー情報を記述するファイル



# Meta Contract DevKitの使い方

## 基本的な操作

- `mc.init()`

新しいBundleを用意する

- `mc.use()`

BundleにFunctionを登録する

- `mc.useFacade()`

BundleにFacadeを登録する

- `mc.deploy()`

Bundle情報を元にコントラクトをデプロイする