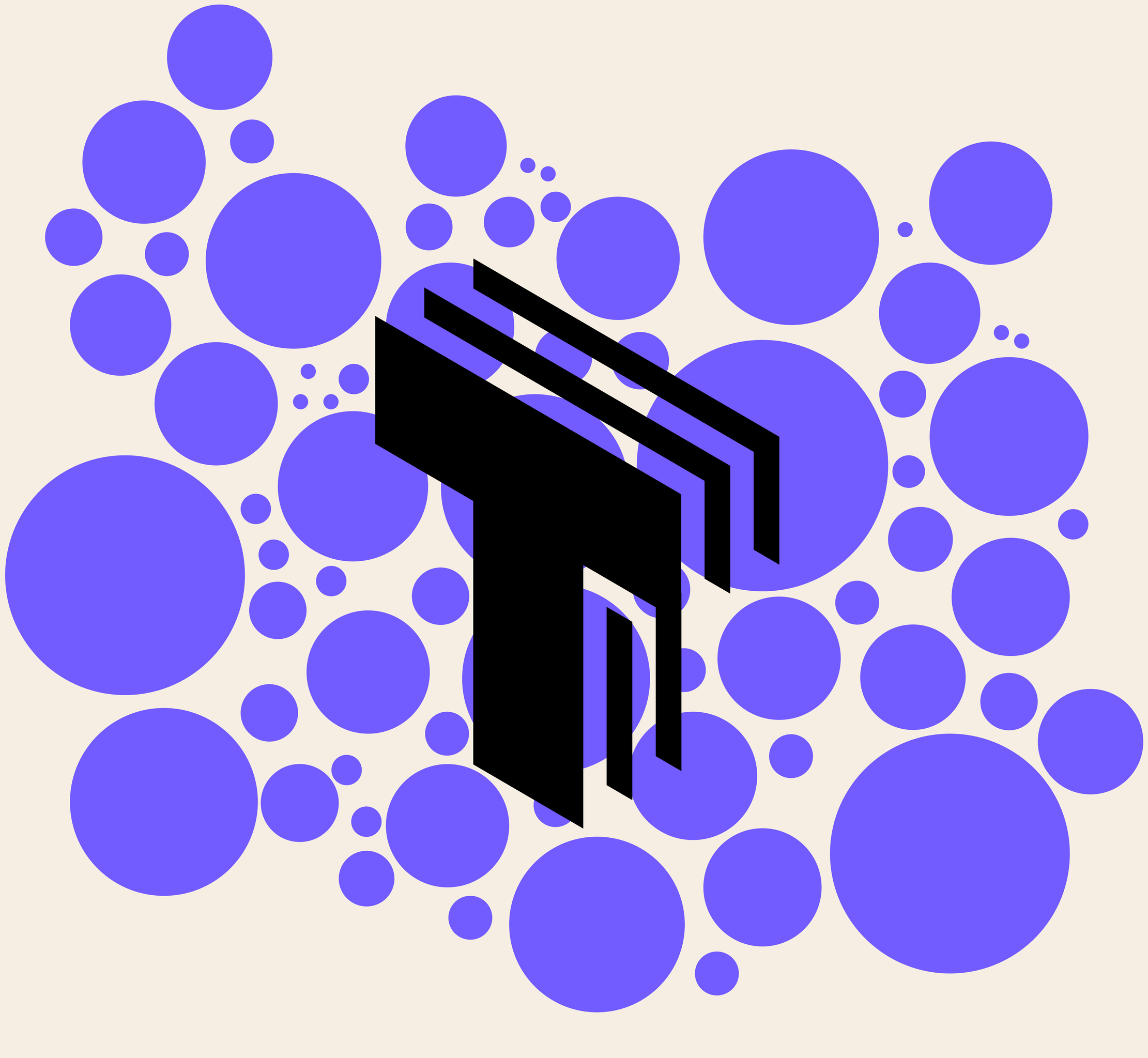


USING GOVERNOR

ERC20Votes SINGLE GOVERNOR WITH TIMELOCK

ABSTRACT

Simple Governor is an example of a basic ERC20 based DAO using OpenZeppelin Governor. This can be thought of as the default secure setup for a ERC20 based 1 token : 1 vote based DAO. This design is the modern implementation of Compound Governor Alpha and should be used in place of Compound Governor Alpha.

FEATURES

Updatable Parameters (Via Governance Proposal)
Quorum based on supply
Delegation

CONTRACTS

ERC20 Standard Token with `Votes` Extension
OpenZeppelin Governor
OpenZeppelin Timelock

INTRODUCTION

OpenZeppelin Governor, ERC20Votes, and the OpenZeppelin Timelock Controller are key components of OpenZeppelin's governance framework, designed to facilitate the creation and management of Decentralized Autonomous Organizations (DAOs).

In simple terms, the OpenZeppelin Governor is a smart contract that enables decentralized decision-making for a DAO. It allows token holders to propose, vote on, and execute actions in a secure and transparent manner.

INTRODUCTION (cont.)

The Governor relies on the ERC20Votes extension, which enhances a standard ERC20 token with voting capabilities. This extension assigns voting power to token holders based on their token balance, enabling them to participate in the governance process.

The Timelock Controller is a separate smart contract that adds a layer of security and control to the execution of proposals. When a proposal is approved by the Governor, it's not executed immediately. Instead, it's scheduled for execution after a predetermined delay. This delay allows stakeholders to review and potentially intervene in case of malicious proposals or unintended consequences.

These three contracts, used together, provide a robust, secure, and flexible governance framework that developers can use to build and manage their DAOs.

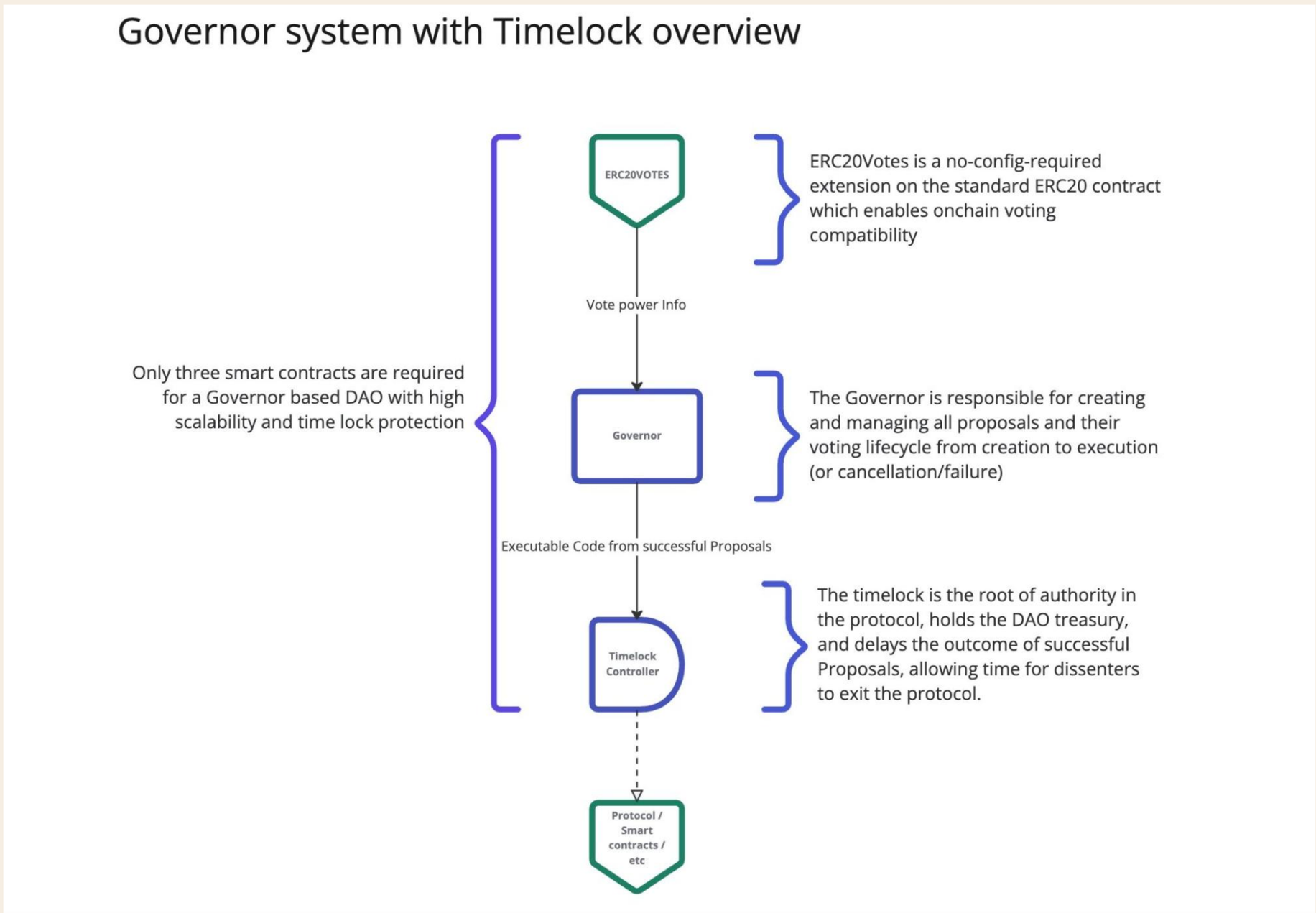
GOVERNOR SYSTEM WITH TIMELOCK OVERVIEW

A basic Governor DAO system consists of three smart contracts and provides a secure, scalable and modular framework for an onchain DAO. This design is suitable for the majority of onchain DAOs, and can be updated without smart contract upgrades to accommodate changes in parameters or administrative roles.

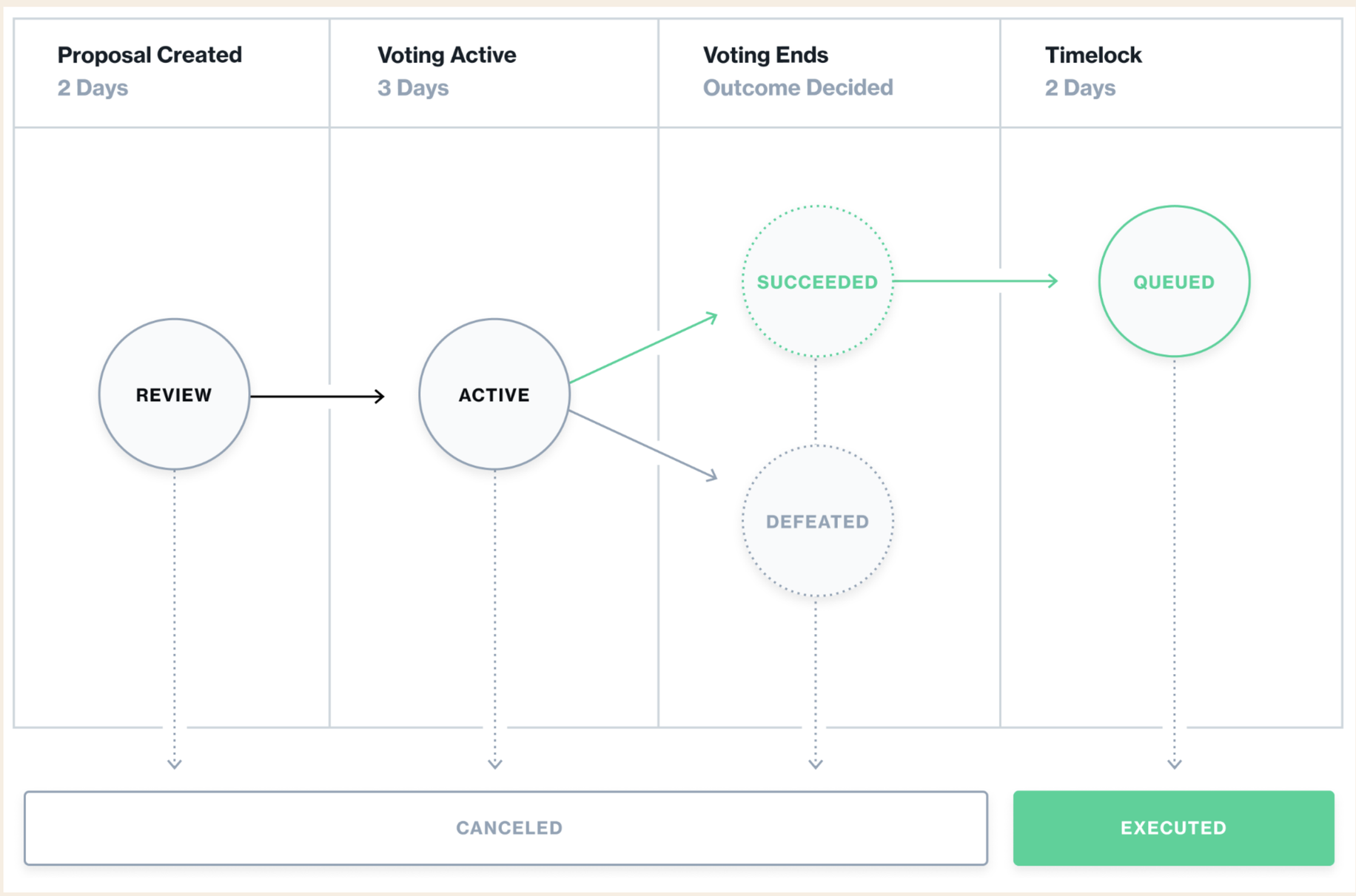
GOVERNOR SYSTEM WITH TIMELOCK OVERVIEW (cont.)

1. **ERC20 (Or ERC721) with the `VOTES` extension**
    a. The Token is responsible for checkpointing the voting power of addresses
    b. Delegation is done on the token contract
2. **Governor**
    a. Responsible for managing the entire proposal lifecycle
    b. Hold the parameters of the DAO (Quorum, proposal threshold)
    c. Sends successful proposals to the Timelock for execution
3. **Timelock Controller**
    a. Is the root of authority in a protocol. When a smart contract requires the DAO as an admin, it is the Timelock controller address which is used.
    b. Delays the execution of proposals to allow user to exit the system before a proposal is enacted
    c. Customizable Permissions for DAO designs where rolls such as PROPOSE, CANCEL, EXECUTE are not necessarily all controlled by the Governor (such as a security council for canceling malicious proposals)

SYSTEM LAYOUT



Governor system with Timelock layout



Governor enabled onchain goverance

Tally

Reference Documentation

@tallyxyz

OPEN ZEPPELIN WIZARD

It is possible to generate the smart contract code for this system and various variations to it, using the OpenZeppelins Wizard, a no code tool for generating secure OpenZeppelin smart contract systems. The resulting code can be deployed using Remix, or downloaded to be used in traditional smart contract development workflows. The code can be considered secure, but Tally recommends audits for all production systems of high value.

To get started creating the system described in this document with the OpenZeppelin Wizard, use the following settings:

ERC20/ERC721 Token

The only required setting is to implement the VOTES extension. All other extensions are optional.

Governor

Voting Period, Voting Delay, Proposal Threshold, and Quorum should all be filled in with settings appropriate for your DAO. When testing, it helps to use low numbers to be sure proposals can be created, voted on, and passed within a reasonable time. Do not set the voting period to less than 8 minutes as the Tally indexer can take up to 8 minutes to reflect state changes on the blockchain.

**Quorum is the number of YES votes a proposal requires to be considered valid. A proposal that has a majority of yes votes, but has not reached quorum, will not pass.*

OPEN ZEPPELIN WIZARD (cont.)

Updatable Settings

Required if the DAO will be able to change the Voting Period, Voting Delay, Proposal Threshold and Quorum parameters. Highly recommended.

Votes

Either ERC20Votes or ERC721Votes. The COMP-like option is for backward compatibility reasons and should not be used for modern deployments.

Timelock

Timelock Controller should be selected. Compound is for only legacy deployments and should not be used.

**The use of a Timelock is optional. If no Timelock is selected, the Governor contract will be the root authority of the DAO. The OpenZeppelin Wizard will not generated the code for deploying a Timelock Controller as there is no configuration required. To learn more see Timelock Controller*

Upgradability

Upgrades are optional. Learn more about Governor upgrades here

EXAMPLE CODE

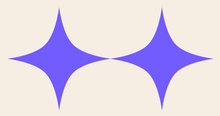
Timelock Controller

It is highly recommended to deploy the Timelock Controller as-is without changes. To read more: https://docs.openzeppelin.com/contracts/4.x/api/governance#TimelockController

EXAMPLE CODE

ERC20 Token

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol";

contract MyToken is ERC20, Ownable, ERC20Permit, ERC20Votes {
    constructor() ERC20("MyToken", "MTK") ERC20Permit("MyToken") {}

    function mint(address to, uint256 amount) public onlyOwner {
        _mint(to, amount);
    }

    // The following functions are overrides required by Solidity.

    function _afterTokenTransfer(address from, address to, uint256 amount)
        internal
        override(ERC20, ERC20Votes)
    {
        super._afterTokenTransfer(from, to, amount);
    }

    function _mint(address to, uint256 amount)
        internal
        override(ERC20, ERC20Votes)
    {
        super._mint(to, amount);
    }

    function _burn(address account, uint256 amount)
        internal
        override(ERC20, ERC20Votes)
    {
        super._burn(account, amount);
    }
}
```



EXAMPLE CODE

Governor

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/governance/Governor.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorSettings.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorCountingSimple.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorVotes.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorVotesQuorumFraction.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorTimelockControl.sol";

contract MyGovernor is Governor, GovernorSettings, GovernorCountingSimple, GovernorVotes, GovernorVotesQuorumFraction, GovernorTimelockControl {
    constructor(IVotes _token, TimelockController _timelock)
        Governor("MyGovernor")
        GovernorSettings(1 /* 1 block */, 50400 /* 1 week */, 0)
        GovernorVotes(_token)
        GovernorVotesQuorumFraction(4)
        GovernorTimelockControl(_timelock)
    {}

    // The following functions are overrides required by Solidity.

    function votingDelay()
        public
        view
        override(IGovernor, GovernorSettings)
        returns (uint256)
    {
        return super.votingDelay();
    }

    function votingPeriod()
        public
        view
        override(IGovernor, GovernorSettings)
        returns (uint256)
    {
        return super.votingPeriod();
    }

    function quorum(uint256 blockNumber)
        public
        view
        override(IGovernor, GovernorVotesQuorumFraction)
        returns (uint256)
    {
        return super.quorum(blockNumber);
    }

    function state(uint256 proposalId)
        public
        view
        override(Governor, GovernorTimelockControl)
        returns (ProposalState)
    {
        return super.state(proposalId);
    }

    function propose(address[] memory targets, uint256[] memory values, bytes[] memory calldatas, string memory description)
        public
        override(Governor, IGovernor)
        returns (uint256)
    {
        return super.propose(targets, values, calldatas, description);
    }

    function proposalThreshold()
        public
        view
        override(Governor, GovernorSettings)
        returns (uint256)
    {
        return super.proposalThreshold();
    }

    function _execute(uint256 proposalId, address[] memory targets, uint256[] memory values, bytes[] memory calldatas, bytes32 descriptionHash)
        internal
        override(Governor, GovernorTimelockControl)
    {
        super._execute(proposalId, targets, values, calldatas, descriptionHash);
    }

    function _cancel(address[] memory targets, uint256[] memory values, bytes[] memory calldatas, bytes32 descriptionHash)
        internal
        override(Governor, GovernorTimelockControl)
        returns (uint256)
    {
        return super._cancel(targets, values, calldatas, descriptionHash);
    }

    function _executor()
        internal
        view
        override(Governor, GovernorTimelockControl)
        returns (address)
    {
        return super._executor();
    }

    function supportsInterface(bytes4 interfaceId)
        public
        view
        override(Governor, GovernorTimelockControl)
        returns (bool)
    {
        return super.supportsInterface(interfaceId);
    }
}
```