- 5 Removals + 4 Additions

```
pragma solidity ^0.5.16;
pragma experimental ABIEncoderV2;

contract Comp {
    /// @notice EIP-20 token name for this token
    string public constant name = "Compound";

    /// @notice EIP-20 token symbol for this token
    string public constant symbol = "COMP";

    /// @notice EIP-20 token decimals for this token
    uint8 public constant decimals = 18;

    /// @notice Total number of tokens in circulation
    uint public constant totalSupply = 10000000e18; // 10 million Comp

    /// @notice Allowance amounts on behalf of others
    mapping (address => mapping (address => uint96)) internal allowances;

    /// @notice Official record of token balances for each account
    mapping (address => uint96) internal balances;

    /// @notice A record of each accounts delegate
    mapping (address => address) public delegates;

    /// @notice A checkpoint for marking number of votes from a given block
    struct Checkpoint {
        uint32 fromBlock;
        uint96 votes;
    }

    /// @notice A record of votes checkpoints for each account, by index
    mapping (address => mapping (uint32 => Checkpoint)) public checkpoints;

    /// @notice The number of checkpoints for each account
    mapping (address => uint32) public numCheckpoints;

    /// @notice The EIP-712 typehash for the contract's domain
    bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");

    /// @notice The EIP-712 typehash for the delegation struct used by the contract
    bytes32 public constant DELEGATION_TYPEHASH = keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");

    /// @notice A record of states for signing / validating signatures
    mapping (address => uint) public nonces;
```

```
pragma solidity ^0.5.16;
pragma experimental ABIEncoderV2;

contract Comp {
    /// @notice EIP-20 token name for this token
    string public constant name = "AGORA DEFI";

    /// @notice EIP-20 token symbol for this token
    string public constant symbol = "AGORA";

    /// @notice EIP-20 token decimals for this token
    uint8 public constant decimals = 18;

    /// @notice Total number of tokens in circulation
    uint public constant totalSupply = 100000000e18; // 100 million

    /// @notice Allowance amounts on behalf of others
    mapping (address => mapping (address => uint96)) internal allowances;

    /// @notice Official record of token balances for each account
    mapping (address => uint96) internal balances;

    /// @notice A record of each accounts delegate
    mapping (address => address) public delegates;

    /// @notice A checkpoint for marking number of votes from a given block
    struct Checkpoint {
        uint32 fromBlock;
        uint96 votes;
    }

    /// @notice A record of votes checkpoints for each account, by index
    mapping (address => mapping (uint32 => Checkpoint)) public checkpoints;

    /// @notice The number of checkpoints for each account
    mapping (address => uint32) public numCheckpoints;

    /// @notice The EIP-712 typehash for the contract's domain
    bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");

    /// @notice The EIP-712 typehash for the delegation struct used by the contract
    bytes32 public constant DELEGATION_TYPEHASH = keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");

    /// @notice A record of states for signing / validating signatures
    mapping (address => uint) public nonces;
```

```
    /// @notice An event thats emitted when an account changes its delegate
    event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);

    /// @notice An event thats emitted when a delegate account's vote balance changes
    event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

    /// @notice The standard EIP-20 transfer event
    event Transfer(address indexed from, address indexed to, uint256 amount);

    /// @notice The standard EIP-20 approval event
    event Approval(address indexed owner, address indexed spender, uint256 amount);

    /**
     * @notice Construct a new Comp token
     * @param account The initial account to grant all the tokens
     */
    constructor(address account) public {
        balances[account] = uint96(totalSupply);
        emit Transfer(address(0), account, totalSupply);
    }

    /**
     * @notice Get the number of tokens `spender` is approved to spend on behalf of `account`
     * @param account The address of the account holding the funds
     * @param spender The address of the account spending the funds
     * @return The number of tokens approved
     */
    function allowance(address account, address spender) external view returns (uint) {
        return allowances[account][spender];
    }

    /**
     * @notice Approve `spender` to transfer up to `amount` from `src`
     * @dev This will overwrite the approval amount for `spender`
     *  and is subject to issues noted [here](https://eips.ethereum.org/EIPS/eip-20#approve)
     * @param spender The address of the account which may transfer tokens
     * @param rawAmount The number of tokens that are approved (2^256-1 means infinite)
     * @return Whether or not the approval succeeded
     */
    function approve(address spender, uint rawAmount) external returns (bool) {
        uint96 amount;
        if (rawAmount == uint(-1)) {
            amount = uint96(-1);
        } else {
            amount = safe96(rawAmount, "Comp::approve: amount exceeds 96 bits");
```


```
    /// @notice An event thats emitted when an account changes its delegate
    event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);

    /// @notice An event thats emitted when a delegate account's vote balance changes
    event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

    /// @notice The standard EIP-20 transfer event
    event Transfer(address indexed from, address indexed to, uint256 amount);

    /// @notice The standard EIP-20 approval event
    event Approval(address indexed owner, address indexed spender, uint256 amount);

    /**
     * @notice Construct a new Comp token
     * @param account The initial account to grant all the tokens
     */
    constructor(address account) public {
        balances[account] = uint96(totalSupply);
        emit Transfer(address(0), account, totalSupply);
    }

    /**
     * @notice Get the number of tokens `spender` is approved to spend on behalf of `account`
     * @param account The address of the account holding the funds
     * @param spender The address of the account spending the funds
     * @return The number of tokens approved
     */
    function allowance(address account, address spender) external view returns (uint) {
        return allowances[account][spender];
    }

    /**
     * @notice Approve `spender` to transfer up to `amount` from `src`
     * @dev This will overwrite the approval amount for `spender`
     *  and is subject to issues noted [here](https://eips.ethereum.org/EIPS/eip-20#approve)
     * @param spender The address of the account which may transfer tokens
     * @param rawAmount The number of tokens that are approved (2^256-1 means infinite)
     * @return Whether or not the approval succeeded
     */
    function approve(address spender, uint rawAmount) external returns (bool) {
        uint96 amount;
        if (rawAmount == uint(-1)) {
            amount = uint96(-1);
        } else {
            amount = safe96(rawAmount, "Comp::approve: amount exceeds 96 bits");
```

```
        }

        allowances[msg.sender][spender] = amount;

        emit Approval(msg.sender, spender, amount);
        return true;
    }

    /**
     * @notice Get the number of tokens held by the `account`
     * @param account The address of the account to get the balance of
     * @return The number of tokens held
     */
    function balanceOf(address account) external view returns (uint) {
        return balances[account];
    }

    /**
     * @notice Transfer `amount` tokens from `msg.sender` to `dst`
     * @param dst The address of the destination account
     * @param rawAmount The number of tokens to transfer
     * @return Whether or not the transfer succeeded
     */
    function transfer(address dst, uint rawAmount) external returns (bool) {
        uint96 amount = safe96(rawAmount, "Comp::transfer: amount exceeds 96 bits");
        _transferTokens(msg.sender, dst, amount);
        return true;
    }

    /**
     * @notice Transfer `amount` tokens from `src` to `dst`
     * @param src The address of the source account
     * @param dst The address of the destination account
     * @param rawAmount The number of tokens to transfer
     * @return Whether or not the transfer succeeded
     */
    function transferFrom(address src, address dst, uint rawAmount) external returns (bool) {
        address spender = msg.sender;
        uint96 spenderAllowance = allowances[src][spender];
        uint96 amount = safe96(rawAmount, "Comp::approve: amount exceeds 96 bits");

        if (spender != src && spenderAllowance != uint96(-1)) {
            uint96 newAllowance = sub96(spenderAllowance, amount, "Comp::transferFrom: transfer amount exceeds spender allowance");
            allowances[src][spender] = newAllowance;

            emit Approval(src, spender, newAllowance);
```

```
        }

        _transferTokens(src, dst, amount);
        return true;
    }

    /**
     * @notice Delegate votes from `msg.sender` to `delegatee`
     * @param delegatee The address to delegate votes to
     */
    function delegate(address delegatee) public {
        return _delegate(msg.sender, delegatee);
    }

    /**
     * @notice Delegates votes from signatory to `delegatee`
     * @param delegatee The address to delegate votes to
     * @param nonce The contract state required to match the signature
     * @param expiry The time at which to expire the signature
     * @param v The recovery byte of the signature
     * @param r Half of the ECDSA signature pair
     * @param s Half of the ECDSA signature pair
     */
    function delegateBySig(address delegatee, uint nonce, uint expiry, uint8 v, bytes32 r, bytes32 s) public {
        bytes32 domainSeparator = keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(name)), getChainId(), address(this)));
        bytes32 structHash = keccak256(abi.encode(DELEGATION_TYPEHASH, delegatee, nonce, expiry));
        bytes32 digest = keccak256(abi.encodePacked("\x19\x01", domainSeparator, structHash));
        address signatory = ecrecover(digest, v, r, s);
        require(signatory != address(0), "Comp::delegateBySig: invalid signature");
        require(nonce == nonces[signatory]++, "Comp::delegateBySig: invalid nonce");
        require(now <= expiry, "Comp::delegateBySig: signature expired");
        return _delegate(signatory, delegatee);
    }

    /**
     * @notice Gets the current votes balance for `account`
     * @param account The address to get votes balance
     * @return The number of current votes for `account`
     */
    function getCurrentVotes(address account) external view returns (uint96) {
        uint32 nCheckpoints = numCheckpoints[account];
        return nCheckpoints > 0 ? checkpoints[account][nCheckpoints - 1].votes : 0;
    }

```

```
    /**
     * @notice Determine the prior number of votes for an account as of a block number
     * @dev Block number must be a finalized block or else this function will revert to prevent misinformation.
     * @param account The address of the account to check
     * @param blockNumber The block number to get the vote balance at
     * @return The number of votes the account had as of the given block
     */
    function getPriorVotes(address account, uint blockNumber) public view returns (uint96) {
        require(blockNumber < block.number, "Comp::getPriorVotes: not yet determined");

        uint32 nCheckpoints = numCheckpoints[account];
        if (nCheckpoints == 0) {
            return 0;
        }

        // First check most recent balance
        if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
            return checkpoints[account][nCheckpoints - 1].votes;
        }

        // Next check implicit zero balance
        if (checkpoints[account][0].fromBlock > blockNumber) {
            return 0;
        }

        uint32 lower = 0;
        uint32 upper = nCheckpoints - 1;
        while (upper > lower) {
            uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
            Checkpoint memory cp = checkpoints[account][center];
            if (cp.fromBlock == blockNumber) {
                return cp.votes;
            } else if (cp.fromBlock < blockNumber) {
                lower = center;
            } else {
                upper = center - 1;
            }
        }
        return checkpoints[account][lower].votes;
    }

    function _delegate(address delegator, address delegatee) internal {
        address currentDelegate = delegates[delegator];
        uint96 delegatorBalance = balances[delegator];
        delegates[delegator] = delegatee;

        emit DelegateChanged(delegator, currentDelegate, delegatee);
```

```
        _moveDelegates(currentDelegate, delegatee, delegatorBalance);
    }

    function _transferTokens(address src, address dst, uint96 amount) internal {
        require(src != address(0), "Comp::_transferTokens: cannot transfer from the zero address");
        require(dst != address(0), "Comp::_transferTokens: cannot transfer to the zero address");

        balances[src] = sub96(balances[src], amount, "Comp::_transferTokens: transfer amount exceeds balance");
        balances[dst] = add96(balances[dst], amount, "Comp::_transferTokens: transfer amount overflows");
        emit Transfer(src, dst, amount);

        _moveDelegates(delegates[src], delegates[dst], amount);
    }

    function _moveDelegates(address srcRep, address dstRep, uint96 amount) internal {
        if (srcRep != dstRep && amount > 0) {
            if (srcRep != address(0)) {
                uint32 srcRepNum = numCheckpoints[srcRep];
                uint96 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;
                uint96 srcRepNew = sub96(srcRepOld, amount, "Comp::_moveVotes: vote amount underflows");
                _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
            }

            if (dstRep != address(0)) {
                uint32 dstRepNum = numCheckpoints[dstRep];
                uint96 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].votes : 0;
                uint96 dstRepNew = add96(dstRepOld, amount, "Comp::_moveVotes: vote amount overflows");
                _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
            }
        }
    }

    function _writeCheckpoint(address delegatee, uint32 nCheckpoints, uint96 oldVotes, uint96 newVotes) internal {
      uint32 blockNumber = safe32(block.number, "Comp::_writeCheckpoint: block number exceeds 32 bits");

      if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
          checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
      } else {
          checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
          numCheckpoints[delegatee] = nCheckpoints + 1;
```


```
        _moveDelegates(currentDelegate, delegatee, delegatorBalance);
    }

    function _transferTokens(address src, address dst, uint96 amount) internal {
        require(src != address(0), "Comp::_transferTokens: cannot transfer from the zero address");
        require(dst != address(0), "Comp::_transferTokens: cannot transfer to the zero address");

        balances[src] = sub96(balances[src], amount, "Comp::_transferTokens: transfer amount exceeds balance");
        balances[dst] = add96(balances[dst], amount, "Comp::_transferTokens: transfer amount overflows");
        emit Transfer(src, dst, amount);

        _moveDelegates(delegates[src], delegates[dst], amount);
    }

    function _moveDelegates(address srcRep, address dstRep, uint96 amount) internal {
        if (srcRep != dstRep && amount > 0) {
            if (srcRep != address(0)) {
                uint32 srcRepNum = numCheckpoints[srcRep];
                uint96 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;
                uint96 srcRepNew = sub96(srcRepOld, amount, "Comp::_moveVotes: vote amount underflows");
                _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
            }

            if (dstRep != address(0)) {
                uint32 dstRepNum = numCheckpoints[dstRep];
                uint96 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].votes : 0;
                uint96 dstRepNew = add96(dstRepOld, amount, "Comp::_moveVotes: vote amount overflows");
                _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
            }
        }
    }

    function _writeCheckpoint(address delegatee, uint32 nCheckpoints, uint96 oldVotes, uint96 newVotes) internal {
      uint32 blockNumber = safe32(block.number, "Comp::_writeCheckpoint: block number exceeds 32 bits");

      if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
          checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
      } else {
          checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
          numCheckpoints[delegatee] = nCheckpoints + 1;
```

```
270       }
271
272       emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
273     }
274
275     function safe32(uint n, string memory errorMessage) internal pure returns (uint32) {
276         require(n < 2**32, errorMessage);
277         return uint32(n);
278     }
279
280     function safe96(uint n, string memory errorMessage) internal pure returns (uint96) {
281         require(n < 2**96, errorMessage);
282         return uint96(n);
283     }
284
285     function add96(uint96 a, uint96 b, string memory errorMessage) internal pure returns (uint96) {
286         uint96 c = a + b;
287         require(c >= a, errorMessage);
288         return c;
289     }
290
291     function sub96(uint96 a, uint96 b, string memory errorMessage) internal pure returns (uint96) {
292         require(b <= a, errorMessage);
293         return a - b;
294     }
295
296     function getChainId() internal pure returns (uint) {
297         uint256 chainId;
298         assembly { chainId := chainid() }
299         return chainId;
300     }
301 }
302
```


```
270       }
271
272       emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
273     }
274
275     function safe32(uint n, string memory errorMessage) internal pure returns (uint32) {
276         require(n < 2**32, errorMessage);
277         return uint32(n);
278     }
279
280     function safe96(uint n, string memory errorMessage) internal pure returns (uint96) {
281         require(n < 2**96, errorMessage);
282         return uint96(n);
283     }
284
285     function add96(uint96 a, uint96 b, string memory errorMessage) internal pure returns (uint96) {
286         uint96 c = a + b;
287         require(c >= a, errorMessage);
288         return c;
289     }
290
291     function sub96(uint96 a, uint96 b, string memory errorMessage) internal pure returns (uint96) {
292         require(b <= a, errorMessage);
293         return a - b;
294     }
295
296     function getChainId() internal pure returns (uint) {
297         uint256 chainId;
298         assembly { chainId := chainid() }
299         return chainId;
300     }
301 }
302
```