Practical session for blockchain

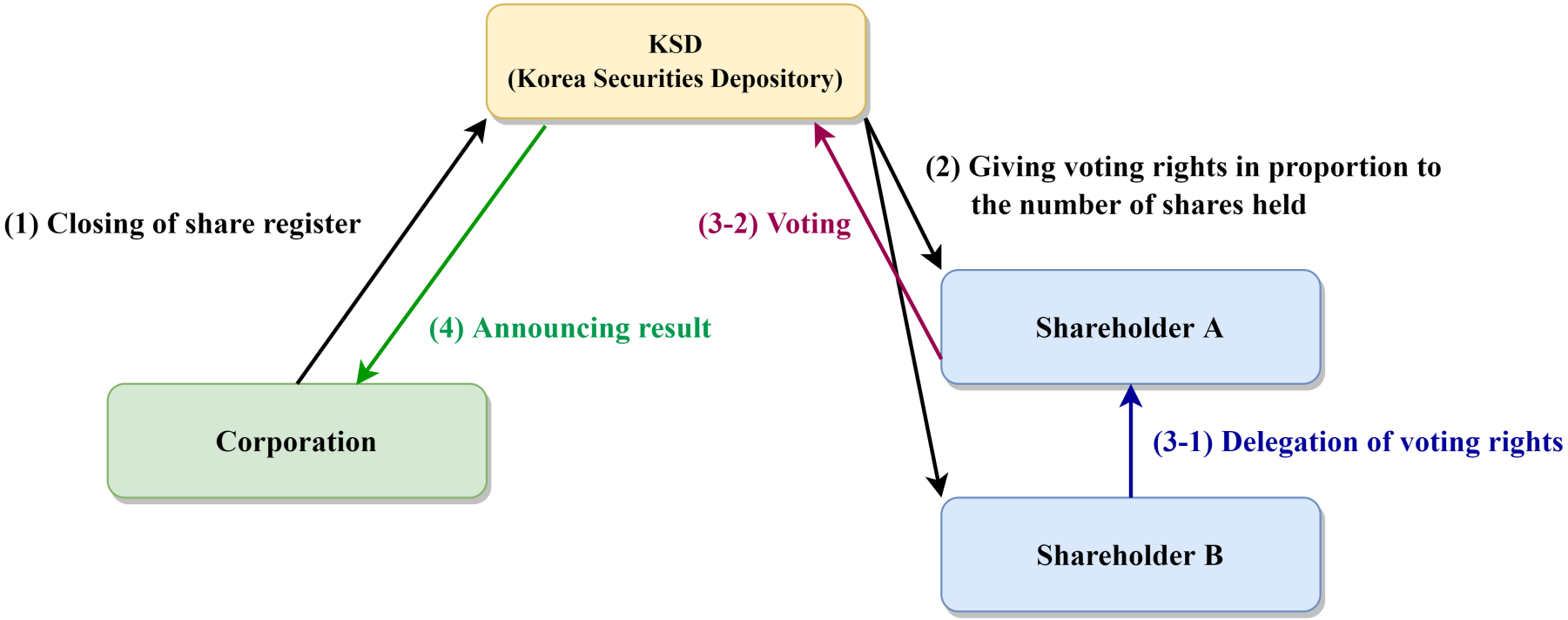
(2) Ballot

Dongkyu Kwak[1] Yeongbong Jin[1]

[1]Financial Risk Engineering Lab.
Seoul National University

April 18, 2022

GMS (General Meeting of Shareholders)



Three State Variables for Ballot

1. Chairperson (address)
   - Those who deploys ballot contract
   - They give voters the right to vote and register candidates.
2. Proposal (struct)
   - Items subject to voting
   - Each proposal has its vote counts.
3. Voter (struct)
   - Those who vote for a candidate or delegate their right to another one.
   - They have owned weight for vote, whether to have voted, proposals they voted, and delegate (optional)

Procedure

1. Chairperson deploys a Ballot contract with a proposals array.
2. Chairperson gives voters (or addresses) the right to vote.
3. A voter can vote on a proposal or delegate the right to another voter.
4. Anyone who has access to the Ballot contract can view the winning proposal and its votes.

State Variables, Constructor

State Variables

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;
contract Ballot {
    struct Voter {
        uint weight; // weight is accumulated by delegation
        bool voted;  // if true, that person already voted
        address delegate; // person delegated to
        uint vote;   // index of the voted proposal
    }
    struct Proposal {
        // If you can limit the length to a certain number of bytes,
        // always use one of bytes1 to bytes32 because they are much cheaper
        bytes32 name;   // short name (up to 32 bytes)
        uint voteCount; // number of accumulated votes
    }
    address public chairperson;
    mapping(address => Voter) public voters;
    Proposal[] public proposals;
```

State Variables, Constructor (cont.)

Iteration

- *for* iteration statement **for (initial val.; end cond.; step)** sequentially assigns the initial variable by increasing or decreasing as much as the step.

Constructor function

```
constructor(bytes32[] memory proposalNames) {
    chairperson = msg.sender;
    voters[chairperson].weight = 1;

    for (uint i = 0; i < proposalNames.length; i++) {
        // Input proposals are sequentially assigned to proposals (an struct Proposal)
        proposals.push(Proposal({
            name: proposalNames[i],
            voteCount: 0
        }));
    }
}
```

Function - giveRightToVote

giveRightToVote

```
function giveRightToVote(address voter) public {
    require(
        msg.sender == chairperson,
        "Only chairperson can give right to vote."
    );
    require(
        !voters[voter].voted,
        "The voter already voted."
    );
    require(voters[voter].weight == 0);
    voters[voter].weight = 1;
}
```

Function - delegate

delegate

```
function delegate(address to) public {
    Voter storage sender = voters[msg.sender];
    require(!sender.voted, "You already voted.");
    require(to != msg.sender, "Self-delegation is disallowed.");

    while (voters[to].delegate != address(0)) {
        to = voters[to].delegate;

        // We found a loop in the delegation, not allowed.
        require(to != msg.sender, "Found loop in delegation.");
    }
    sender.voted = true;
    sender.delegate = to;
    Voter storage delegate_ = voters[to];
    if (delegate_.voted) {
        // If the delegate already voted,
        // directly add to the number of votes
        proposals[delegate_.vote].voteCount += sender.weight;
    } else {
        // If the delegate did not vote yet,
        // add to her weight.
        delegate_.weight += sender.weight;
    }
}
```

Function - vote

vote

```
function vote(uint proposal) public {
    Voter storage sender = voters[msg.sender];
    require(sender.weight != 0, "Has no right to vote");
    require(!sender.voted, "Already voted.");
    sender.voted = true;
    sender.vote = proposal;

    // If 'proposal' is out of the range of the array,
    // this will throw automatically and revert all
    // changes.
    proposals[proposal].voteCount += sender.weight;
}
```

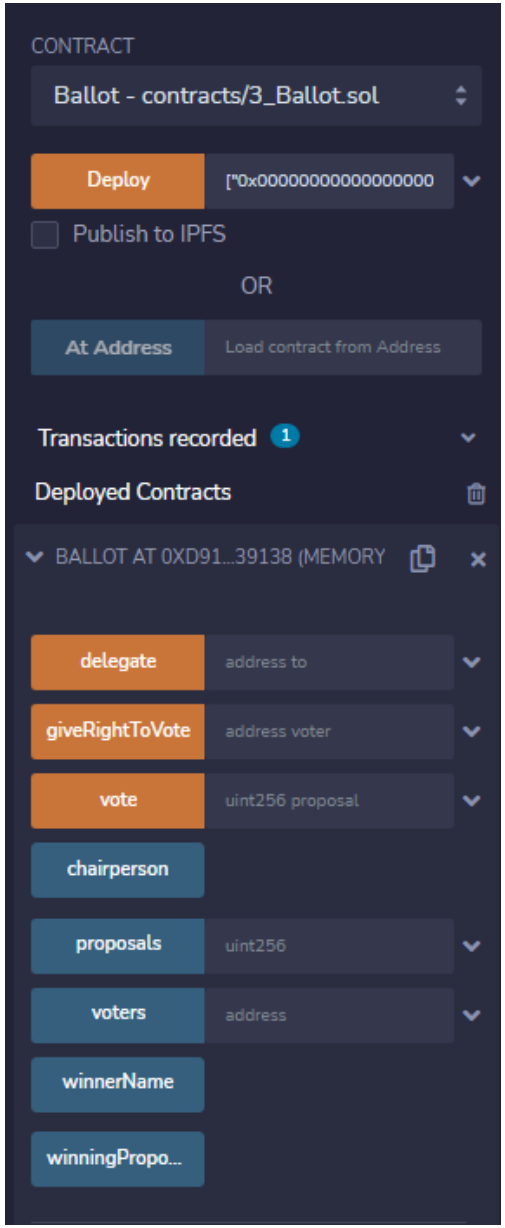
Function - winningProposal, winnerName

winningProposal, winnerName

```
    function winningProposal() public view
            returns (uint winningProposal_)
    {
        uint winningVoteCount = 0;
        for (uint p = 0; p < proposals.length; p++) {
            if (proposals[p].voteCount > winningVoteCount) {
                winningVoteCount = proposals[p].voteCount;
                winningProposal_ = p;
            }
        }
    }

    function winnerName() public view
            returns (bytes32 winnerName_)
    {
        winnerName_ = proposals[winningProposal()].name;
    }
}
```

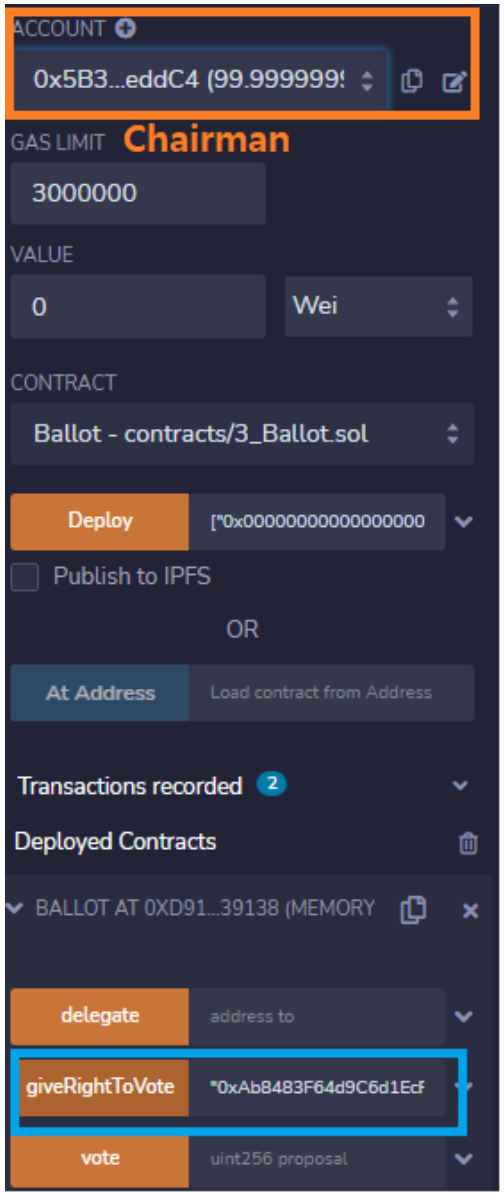
Deploy Ballot With Proposals bytes32 Array



When the Ballot contract is deployed, a **bytes32 array** with at least one item must be entered.

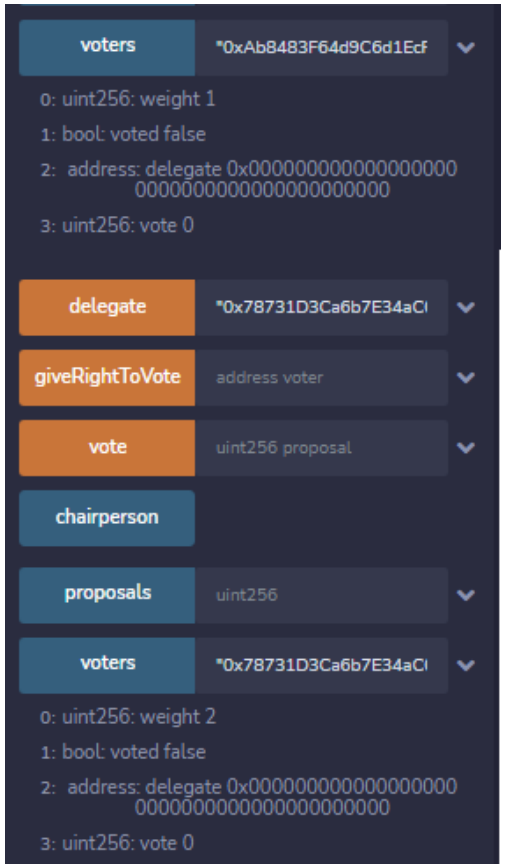
bytes32 is constructed from 64 digits of hexadecimal numbers.
(for example,
["0x0000000000000000000000000000000000000000000000000000000000000001",
"0x0000000000000000000000000000000000000000000000000000000000000002",
"0x0000000000000000000000000000000000000000000000000000000000000003"])

Give other addresses right to vote



After msg.sender == chairperson set, run transaction *giveRightToVote* with an address of other accounts (at least 3 accounts)

Check Voter struct and delegate

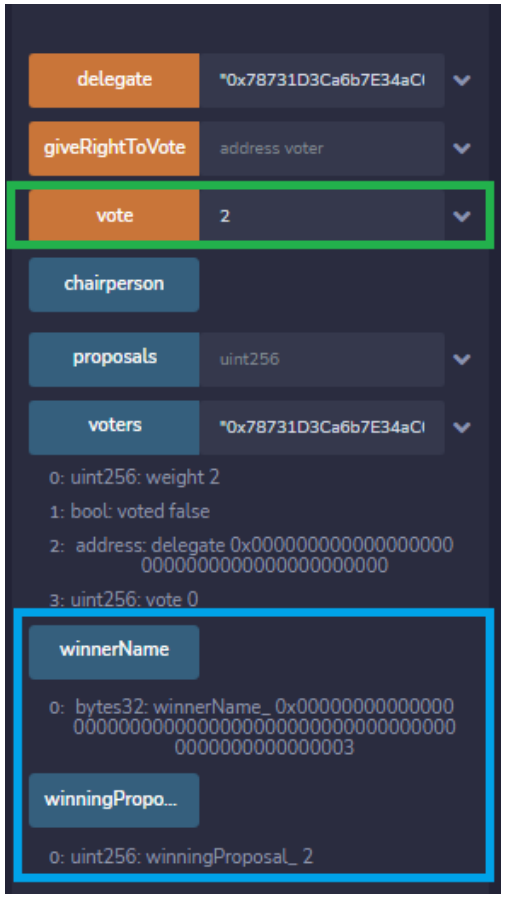


Check *Voter* struct of a voter through the voter's address.

Then, delegate the voter's right to another voter's address.

Check the *Voter* struct of the delegated address again.

Vote and Check Winning Proposal



Vote on a proposal according to the order of each proposal.

Check the winning proposal and the name of winning proposal via winnerName.

Improvements

1. After vote, anyone can easily check which candidate the voter voted for.
   How can it be switched to the secret vote?
2. It seems to be unfair to be able to preview the majority of the votes before the voting ends.
   How can the chairperson adjust manually or automatically (by specific time) whether the result is shown?

References

- Ballot, https://remix.ethereum.org/