# Loan Service On Blockchain

廖信堯

# Agenda

01 Introduction on lending service and Motivation

02 Industry Survey on blockchainized lending

03 Framework of ETHLend and Implementation

04 Suggestion and Conclusion

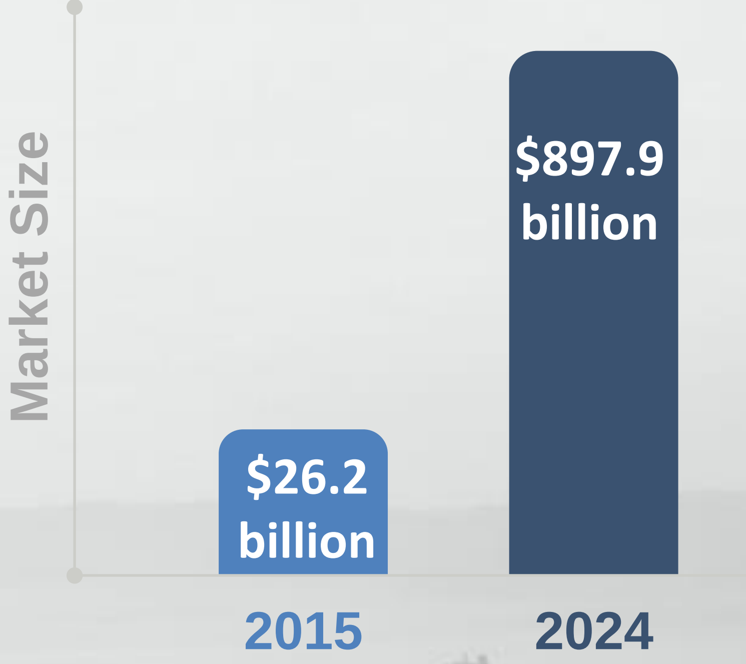
# Introduction on lending service and Motivation

- **Customer lending service**
  - So far, large scale lending service are provided either by centralized banking institution or one-to-one manner.
  - High fee or take worse loan condition and risk.
- **Motivation**
  - Even in opaque environment with high fee and risk, customer lending is appealing and still growing.
  - It seems that the alignment of blockchain and customer lending is an attractive applicability.

# Industry Survey on blockchainized lending

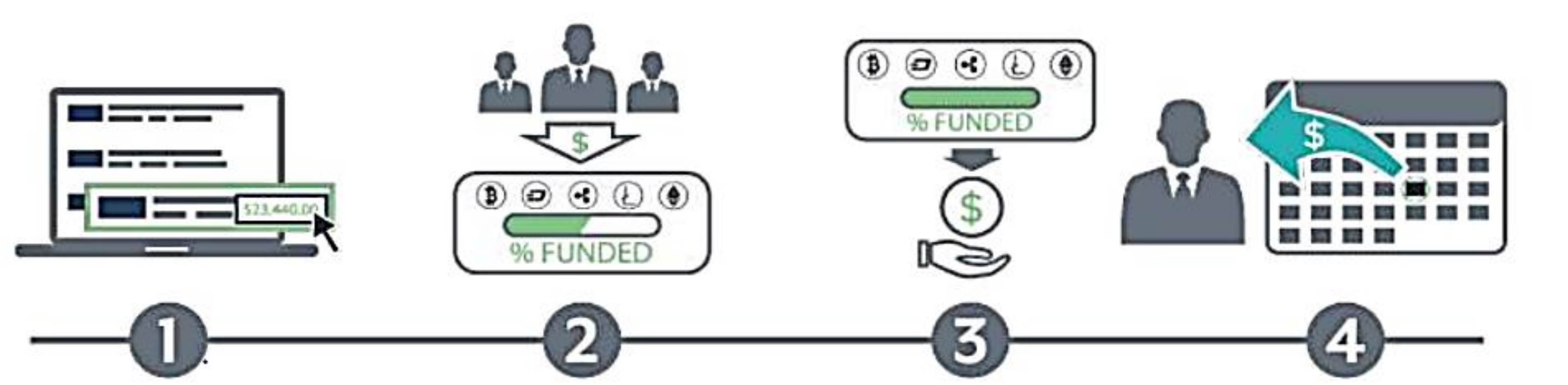
Decentralized lending model replaces the centralized one and its market is expected to reach as much as $897.85 billion by 2024. Considering the advantages for both sides, a number of blockchain lending projects are already being deployed.

- SALT-Lending
- Ripio Credit Network (RCN)
- ETHLend



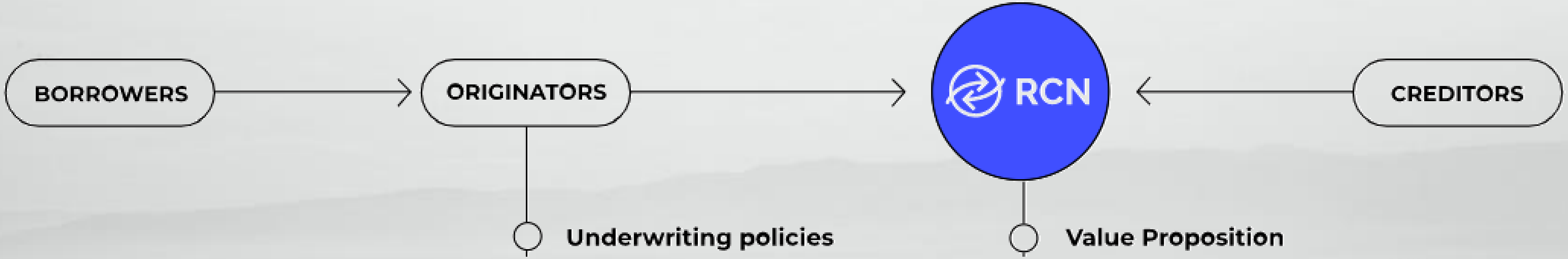
# SALT-Lending

- The largest blockchain lending platform with a market cap of $126 million
- Annual membership business model, which is embedded with Ethereum-based ERC20 smart contract
- Enable their member to leverage their blockchain assets to secure cash loans



# Ripio Credit Network (RCN)

- RCN is a credit network protocol based on cosigned smart contracts, with a market cap of $45 million distributed on Latin America unbanked population
- Besides of collateral, RCN introduces an intermediary cosigner to neutralize lender’s credit risk
- Various type of participants, like scoring agent, ID verifier, etc.

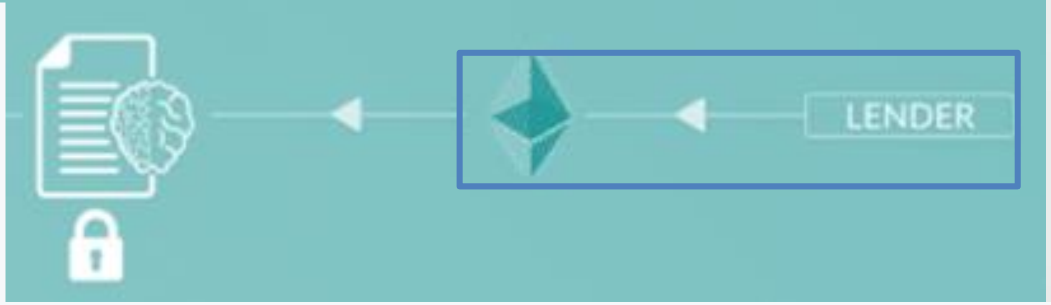


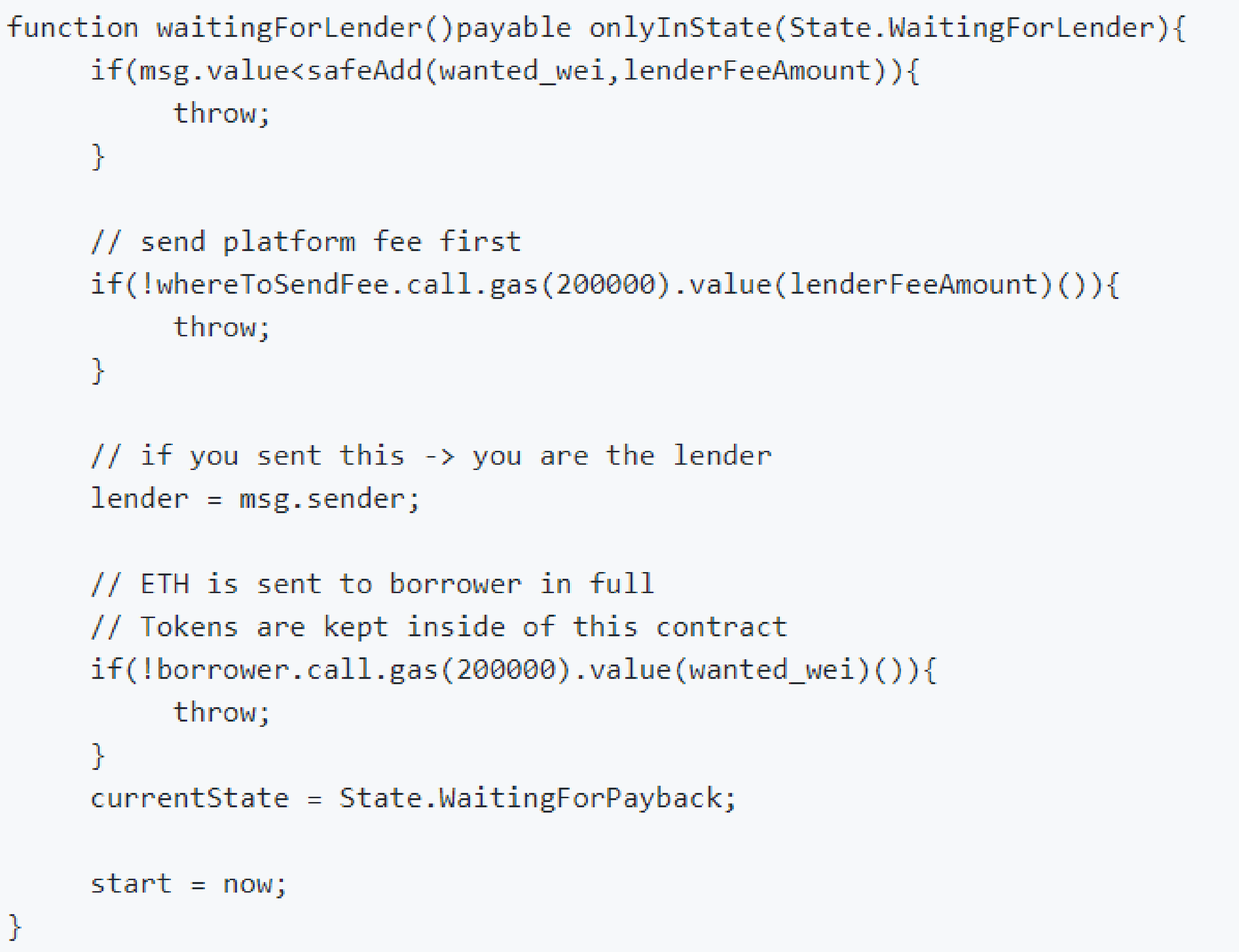
# Framework of ETHLend and Implementation

- ETHLend is an Ethereum-based lending platform, with $45 million market cap
- ETHLend solves the default loss problem by using digital asset, which is ERC-20 compatible tokens as collateral
- Tokenization means that things with value can be issued and represented on Ethereum-based ERC-20 compatible token, like DigixDAO, a gold tokenization

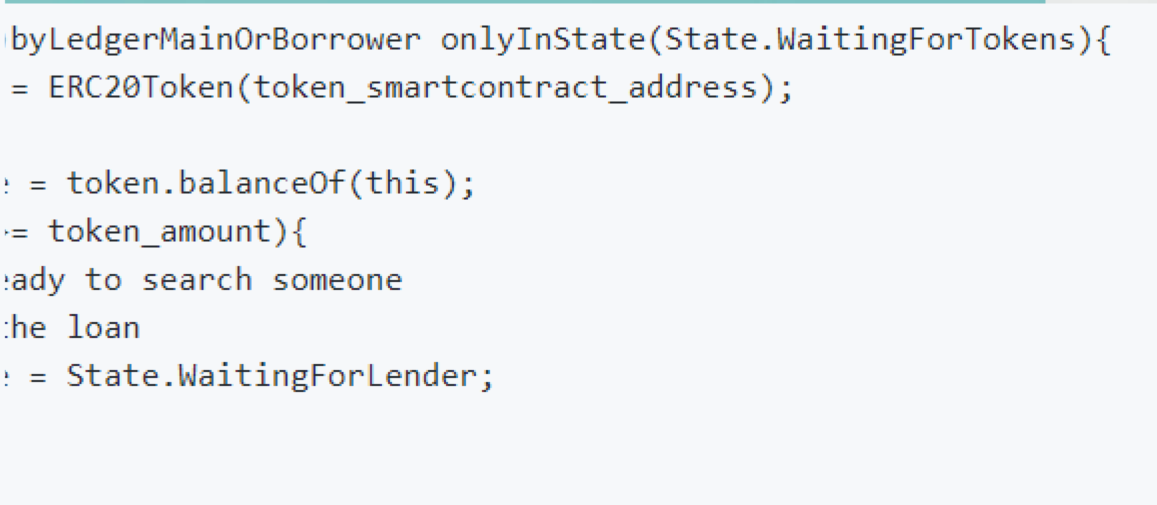
# Framework of ETHLend and Implementation (cont.)

Smart Contract setup with ERC20 Token Collateral



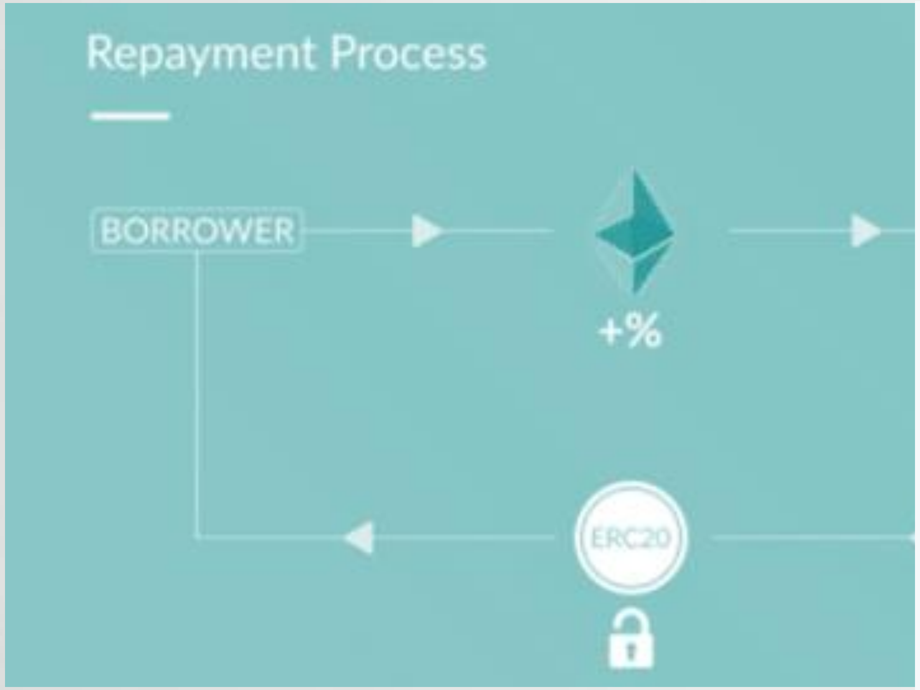


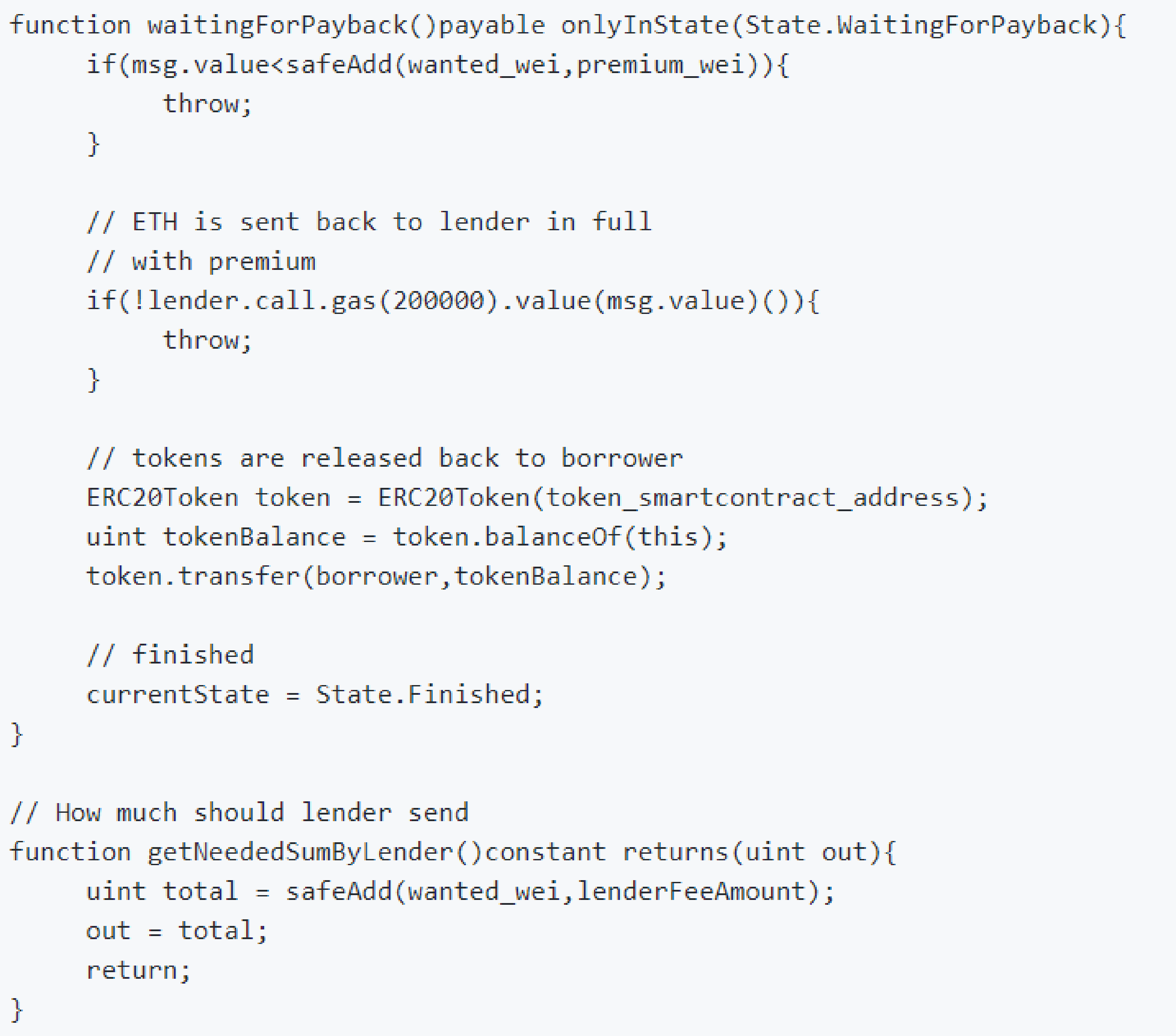
```
function waitingForLender()payable onlyInState(State.WaitingForLender){
     if(msg.value<safeAdd(wanted_wei,lenderFeeAmount)){
          throw;
     }

     // send platform fee first
     if(!whereToSendFee.call.gas(200000).value(lenderFeeAmount)()){
          throw;
     }

     // if you sent this -> you are the lender
     lender = msg.sender;

     // ETH is sent to borrower in full
     // Tokens are kept inside of this contract
     if(!borrower.call.gas(200000).value(wanted_wei)()){
          throw;
     }
     currentState = State.WaitingForPayback;

     start = now;
}
```



```
byLedgerMainOrBorrower onlyInState(State.WaitingForTokens){
= ERC20Token(token_smartcontract_address);

 = token.balanceOf(this);
= token_amount){
ady to search someone
he loan
 = State.WaitingForLender;
```

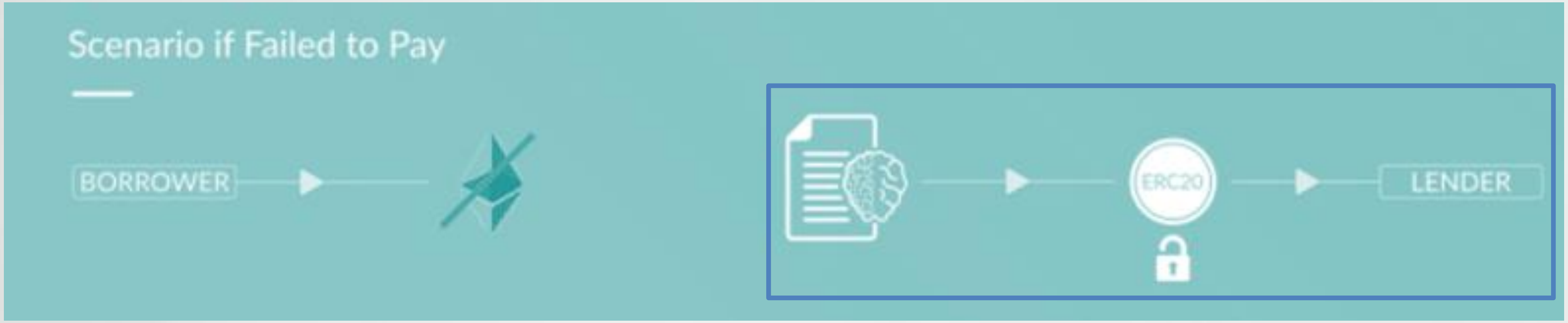
# Framework of ETHLend and Implementation (cont.)

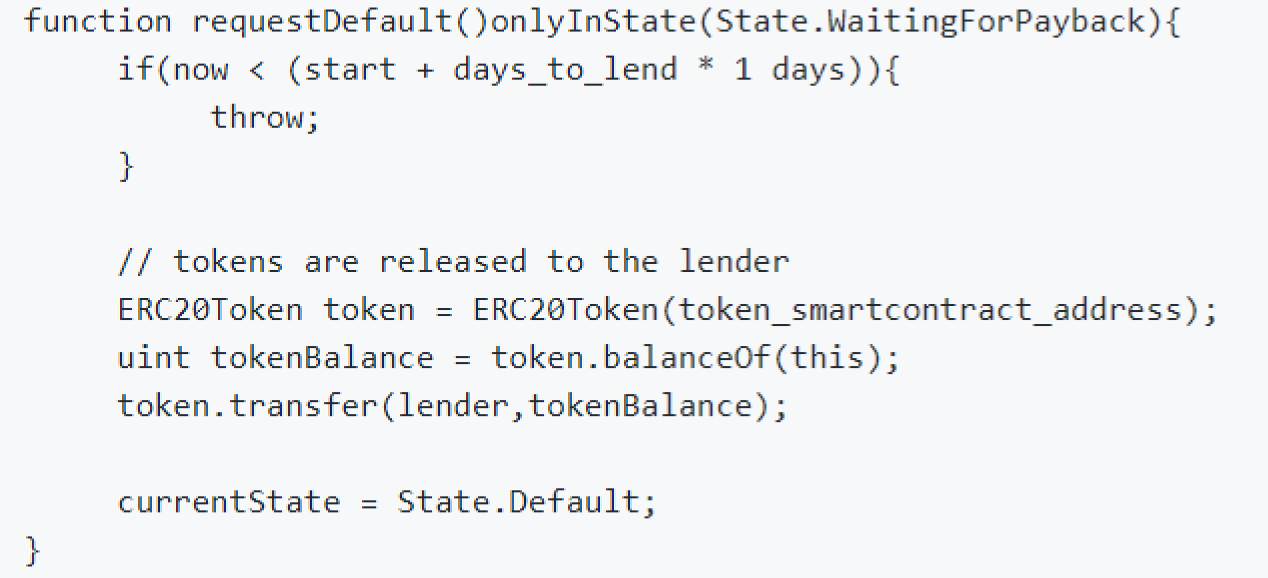




```
function waitingForPayback()payable onlyInState(State.WaitingForPayback){
     if(msg.value<safeAdd(wanted_wei,premium_wei)){
          throw;
     }

     // ETH is sent back to lender in full
     // with premium
     if(!lender.call.gas(200000).value(msg.value)()){
          throw;
     }

     // tokens are released back to borrower
     ERC20Token token = ERC20Token(token_smartcontract_address);
     uint tokenBalance = token.balanceOf(this);
     token.transfer(borrower,tokenBalance);

     // finished
     currentState = State.Finished;
}

// How much should lender send
function getNeededSumByLender()constant returns(uint out){
     uint total = safeAdd(wanted_wei,lenderFeeAmount);
     out = total;
     return;
}
```

# Framework of ETHLend and Implementation (cont.)





```
function requestDefault()onlyInState(State.WaitingForPayback){
     if(now < (start + days_to_lend * 1 days)){
          throw;
     }

     // tokens are released to the lender
     ERC20Token token = ERC20Token(token_smartcontract_address);
     uint tokenBalance = token.balanceOf(this);
     token.transfer(lender,tokenBalance);

     currentState = State.Default;
}
```

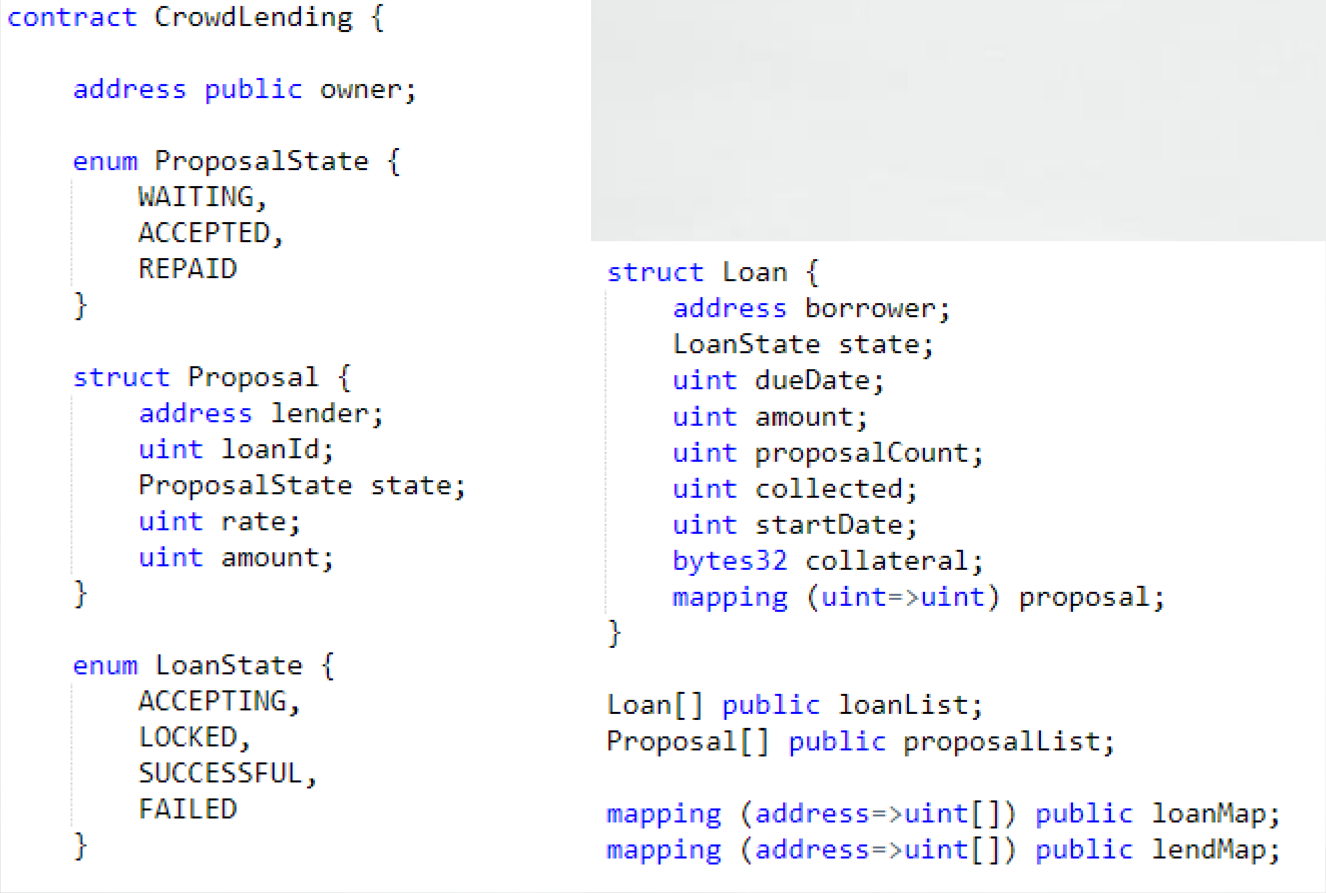
# Suggestion and Conclusion

- **Suggestion**
  - Unsecured loans? Borrowers can build their reputation by amount of Credit tokens (CRE) over time as they successfully pay back loans
  - Many-to-many model? Like crowdfunding, collecting funds from multiple lenders to increase loan capability and distributing lender’s risk of default to multiple bids
- **Conclusion**
  - Blockchain brings lower fees, better transparency, saved time and money, and
  - market expansion to the lending market.
  - The alignment of blockchain and lending is appealing, but it’s still in its infancy and
  - needs more mechanisms to complete the lending service

# Toy Smart Contract

- **Many-to-many model**
  - The system structure is similar to ETHLend, but there are 3 differences:
    1. The loan terms, including amount and interest rate, are determined by lenders, not set by borrowers.
    2. Borrowers can accept multiple proposals within requested amount, which gives them rights to choose based on lenders’ proposals.
    3. The collateral is not restricted to ERC-20 only.

# Toy Smart Contract (cont.)

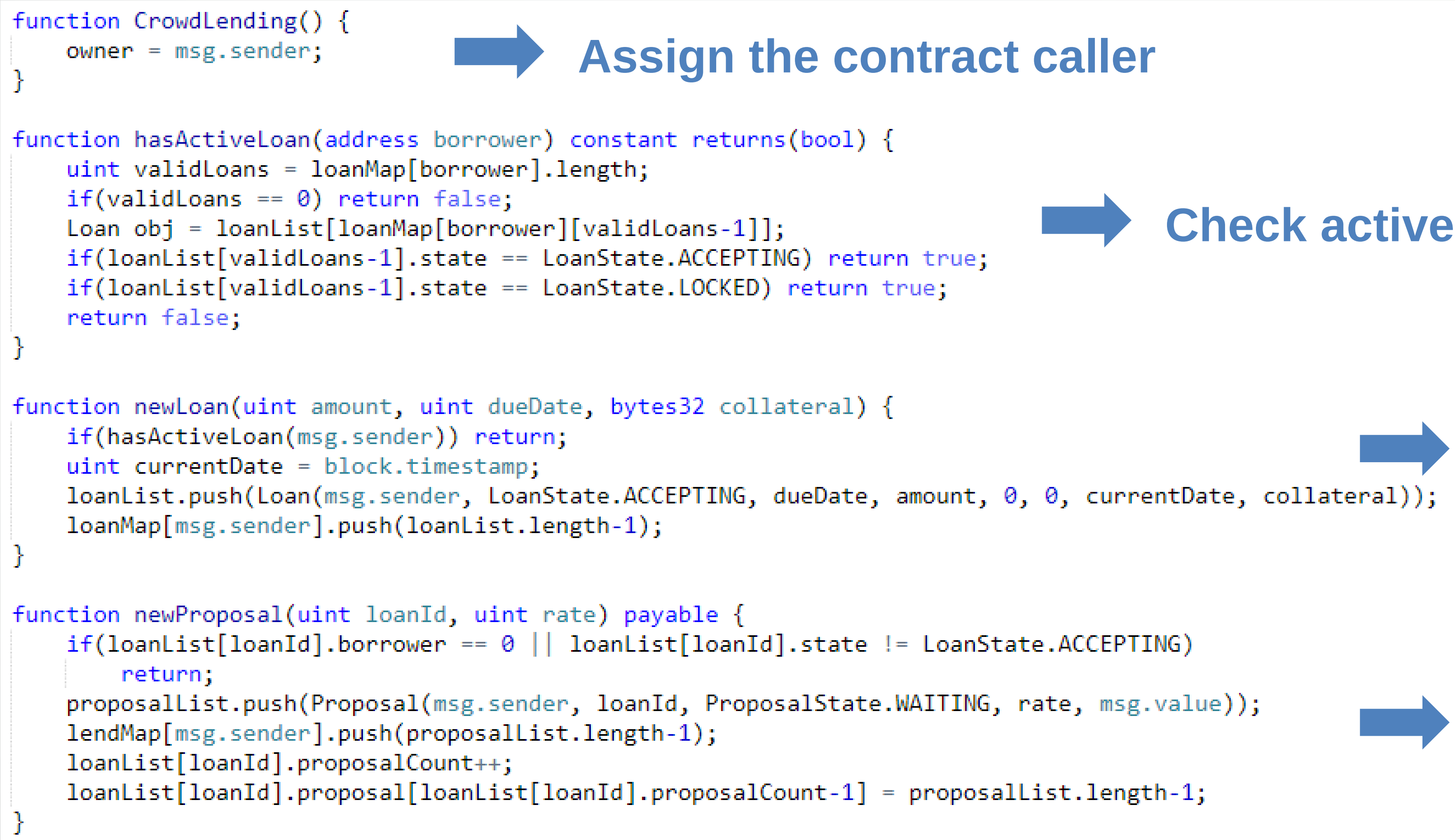


```
contract CrowdLending {

    address public owner;

    enum ProposalState {
        WAITING,
        ACCEPTED,
        REPAID
    }

    struct Proposal {
        address lender;
        uint loanId;
        ProposalState state;
        uint rate;
        uint amount;
    }

    enum LoanState {
        ACCEPTING,
        LOCKED,
        SUCCESSFUL,
        FAILED
    }
```

```
struct Loan {
    address borrower;
    LoanState state;
    uint dueDate;
    uint amount;
    uint proposalCount;
    uint collected;
    uint startDate;
    bytes32 collateral;
    mapping (uint=>uint) proposal;
}

Loan[] public loanList;
Proposal[] public proposalList;

mapping (address=>uint[]) public loanMap;
mapping (address=>uint[]) public lendMap;
```

➡ **Initialization and mapping**

# Toy Smart Contract (cont.)



```
function CrowdLending() {
    owner = msg.sender;
}

function hasActiveLoan(address borrower) constant returns(bool) {
    uint validLoans = loanMap[borrower].length;
    if(validLoans == 0) return false;
    Loan obj = loanList[loanMap[borrower][validLoans-1]];
    if(loanList[validLoans-1].state == LoanState.ACCEPTING) return true;
    if(loanList[validLoans-1].state == LoanState.LOCKED) return true;
    return false;
}

function newLoan(uint amount, uint dueDate, bytes32 collateral) {
    if(hasActiveLoan(msg.sender)) return;
    uint currentDate = block.timestamp;
    loanList.push(Loan(msg.sender, LoanState.ACCEPTING, dueDate, amount, 0, 0, currentDate, collateral));
    loanMap[msg.sender].push(loanList.length-1);
}

function newProposal(uint loanId, uint rate) payable {
    if(loanList[loanId].borrower == 0 || loanList[loanId].state != LoanState.ACCEPTING)
        return;
    proposalList.push(Proposal(msg.sender, loanId, ProposalState.WAITING, rate, msg.value));
    lendMap[msg.sender].push(proposalList.length-1);
    loanList[loanId].proposalCount++;
    loanList[loanId].proposal[loanList[loanId].proposalCount-1] = proposalList.length-1;
}
```

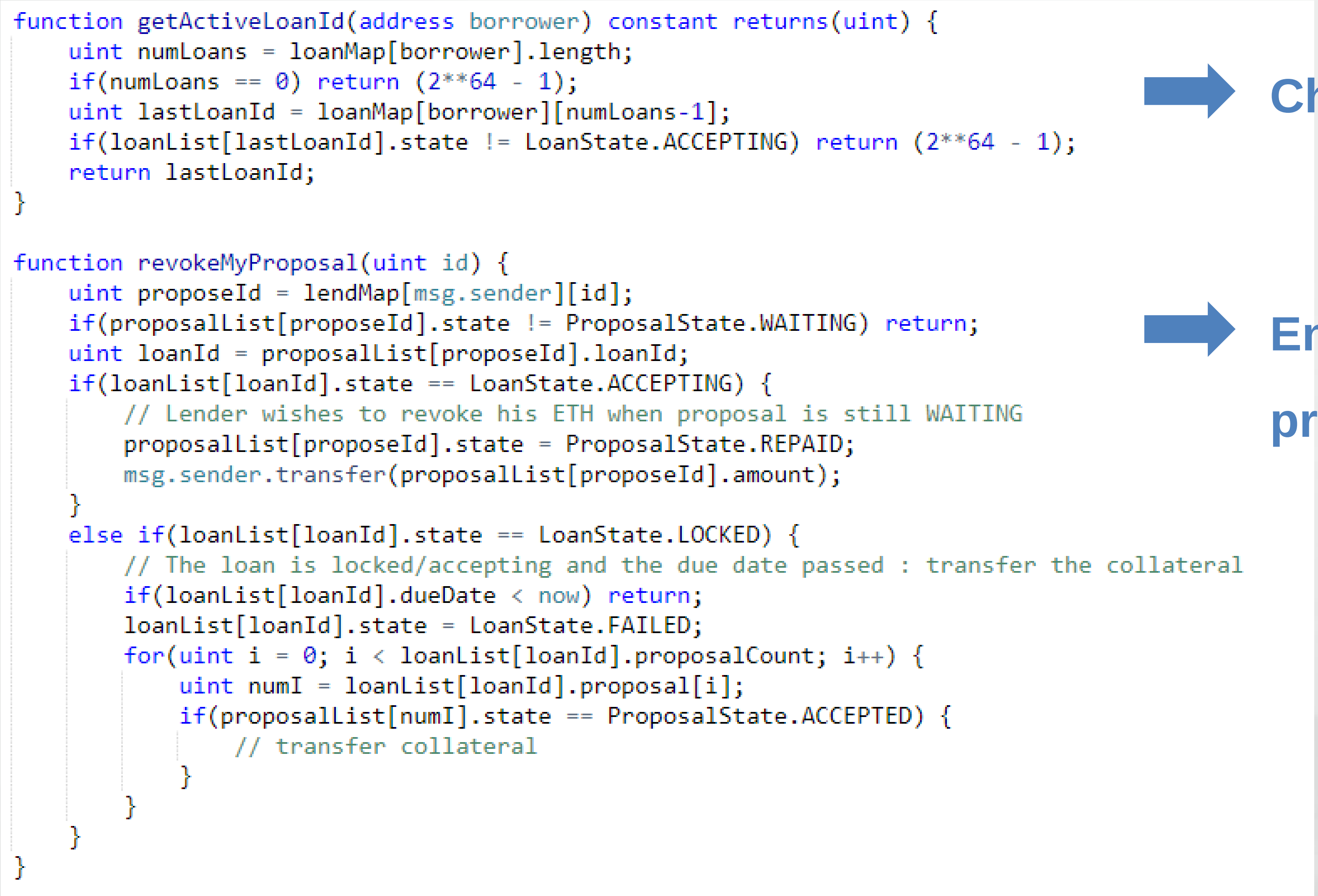
Assign the contract caller

Check active loan

Activate new loan

Create new proposal

# Toy Smart Contract (cont.)

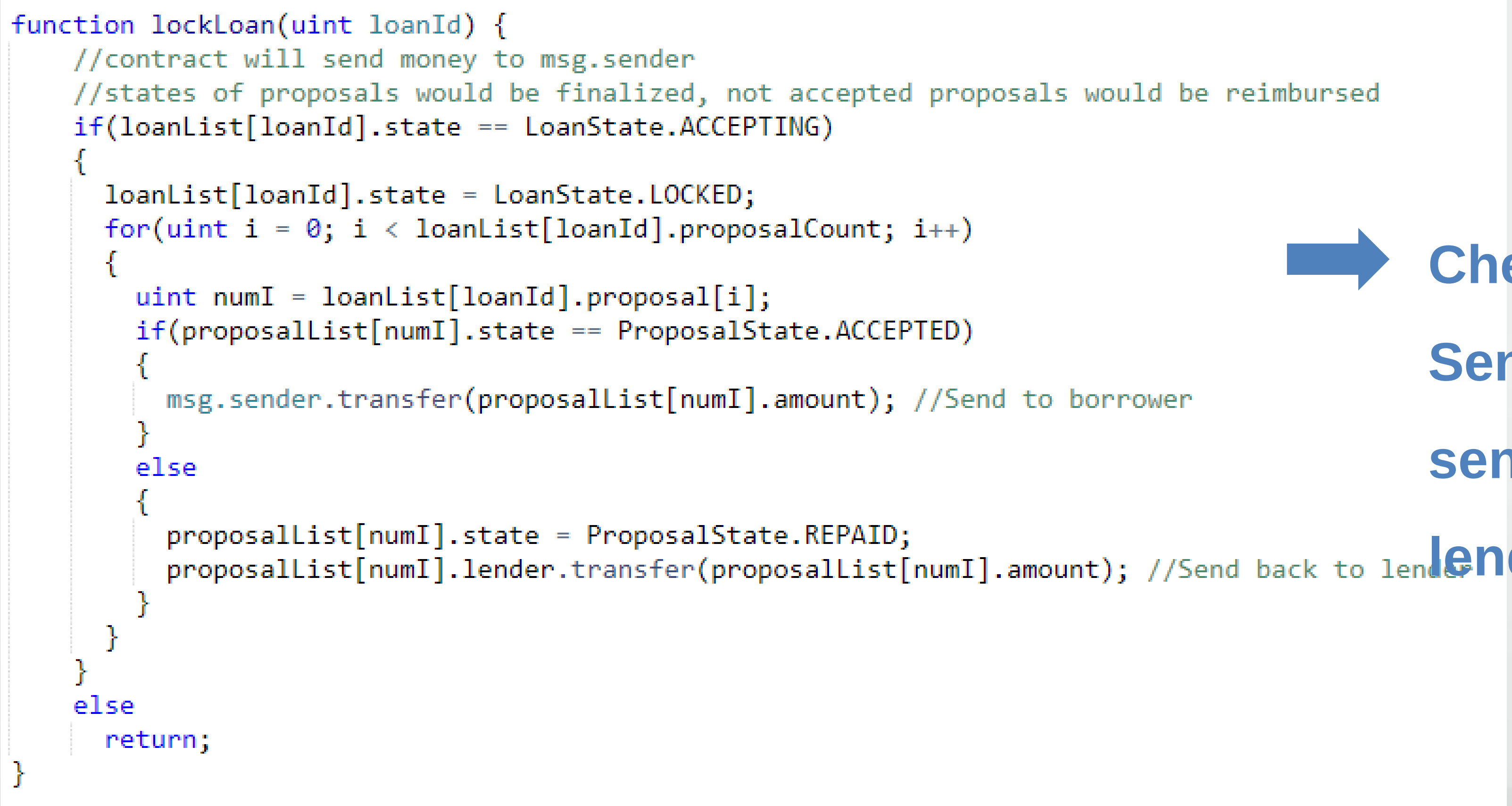


```
function getActiveLoanId(address borrower) constant returns(uint) {
    uint numLoans = loanMap[borrower].length;
    if(numLoans == 0) return (2**64 - 1);
    uint lastLoanId = loanMap[borrower][numLoans-1];
    if(loanList[lastLoanId].state != LoanState.ACCEPTING) return (2**64 - 1);
    return lastLoanId;
}

function revokeMyProposal(uint id) {
    uint proposeId = lendMap[msg.sender][id];
    if(proposalList[proposeId].state != ProposalState.WAITING) return;
    uint loanId = proposalList[proposeId].loanId;
    if(loanList[loanId].state == LoanState.ACCEPTING) {
        // Lender wishes to revoke his ETH when proposal is still WAITING
        proposalList[proposeId].state = ProposalState.REPAID;
        msg.sender.transfer(proposalList[proposeId].amount);
    }
    else if(loanList[loanId].state == LoanState.LOCKED) {
        // The loan is locked/accepting and the due date passed : transfer the collateral
        if(loanList[loanId].dueDate < now) return;
        loanList[loanId].state = LoanState.FAILED;
        for(uint i = 0; i < loanList[loanId].proposalCount; i++) {
            uint numI = loanList[loanId].proposal[i];
            if(proposalList[numI].state == ProposalState.ACCEPTED) {
                // transfer collateral
            }
        }
    }
}
```

Check active loanID

Enable lenders to withdraw proposal before accepted

# Toy Smart Contract (cont.)

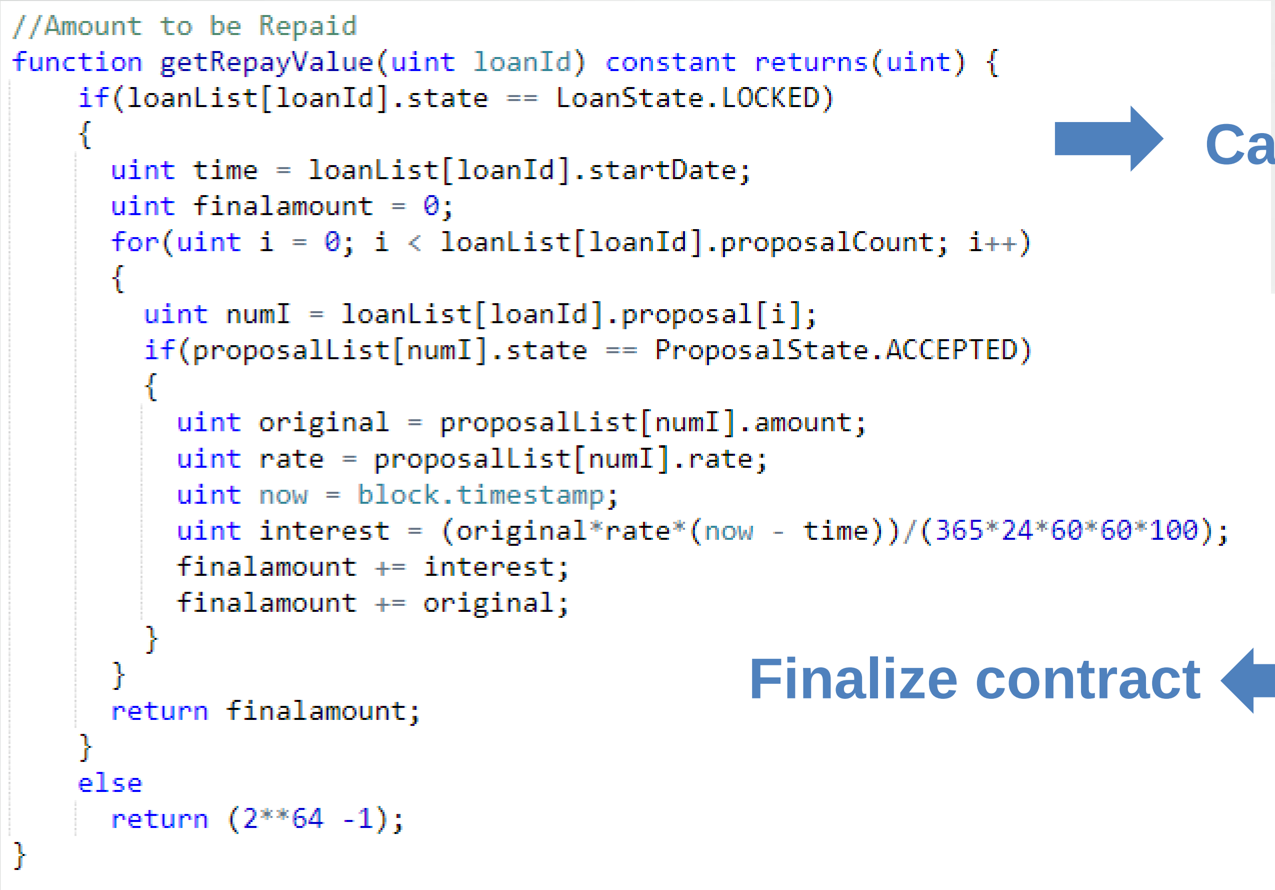


```
function lockLoan(uint loanId) {
    //contract will send money to msg.sender
    //states of proposals would be finalized, not accepted proposals would be reimbursed
    if(loanList[loanId].state == LoanState.ACCEPTING)
    {
      loanList[loanId].state = LoanState.LOCKED;
      for(uint i = 0; i < loanList[loanId].proposalCount; i++)
      {
        uint numI = loanList[loanId].proposal[i];
        if(proposalList[numI].state == ProposalState.ACCEPTED)
        {
          msg.sender.transfer(proposalList[numI].amount); //Send to borrower
        }
        else
        {
          proposalList[numI].state = ProposalState.REPAID;
          proposalList[numI].lender.transfer(proposalList[numI].amount); //Send back to lender
        }
      }
    }
    else
      return;
}
```

Check proposal status

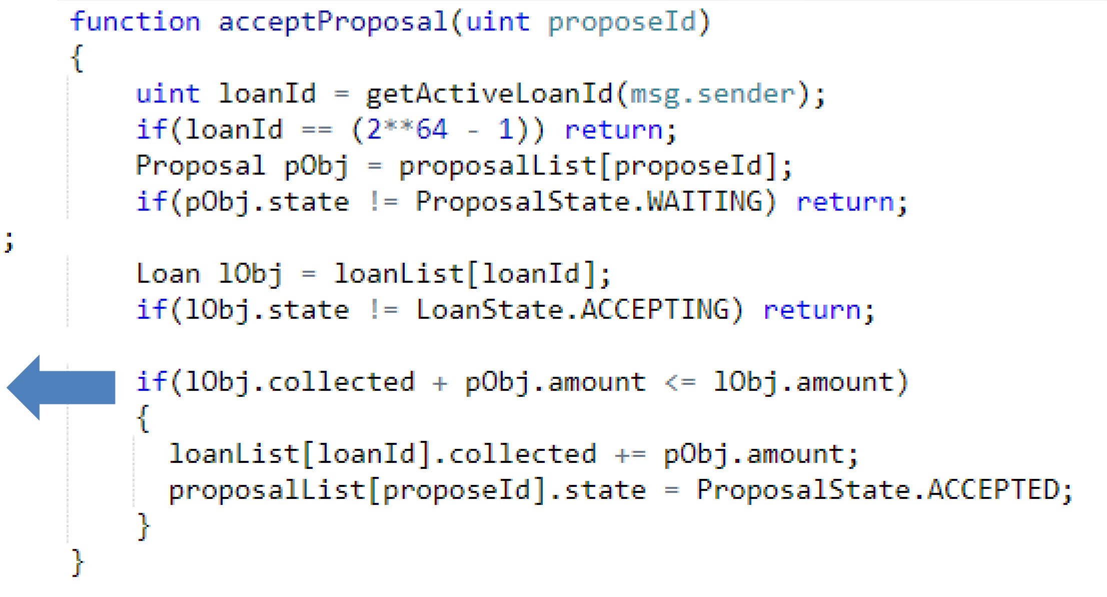
Send ETH to the borrower and send ETH back to unaccepted lenders

# Toy Smart Contract (cont.)



```
//Amount to be Repaid
function getRepayValue(uint loanId) constant returns(uint) {
    if(loanList[loanId].state == LoanState.LOCKED)
    {
      uint time = loanList[loanId].startDate;
      uint finalamount = 0;
      for(uint i = 0; i < loanList[loanId].proposalCount; i++)
      {
        uint numI = loanList[loanId].proposal[i];
        if(proposalList[numI].state == ProposalState.ACCEPTED)
        {
          uint original = proposalList[numI].amount;
          uint rate = proposalList[numI].rate;
          uint now = block.timestamp;
          uint interest = (original*rate*(now - time))/(365*24*60*60*100);
          finalamount += interest;
          finalamount += original;
        }
      }
      return finalamount;
    }
    else
      return (2**64 -1);
}
```

Calculate repayment



```
function acceptProposal(uint proposeId)
{
    uint loanId = getActiveLoanId(msg.sender);
    if(loanId == (2**64 - 1)) return;
    Proposal pObj = proposalList[proposeId];
    if(pObj.state != ProposalState.WAITING) return;

    Loan lObj = loanList[loanId];
    if(lObj.state != LoanState.ACCEPTING) return;

    if(lObj.collected + pObj.amount <= lObj.amount)
    {
      loanList[loanId].collected += pObj.amount;
      proposalList[proposeId].state = ProposalState.ACCEPTED;
    }
}
```

Finalize contract

# Reference

1. MinGi Kweon. (2018). A Study of Blockchain Technology in Facilitating Lending Services with Distributed Risk Aversion. Review of Computational Science and Engineering, 4(1), 91-101.
2. FLATICON website. https://www.flaticon.com/
3. FLATICON website. https://www.flaticon.com/
4. Data is retrieved from https://www.transparencymarketresearch.com/
5. White Paper – SALT Blockchain-backed Loans. https://whitepaper.io/document/85/salt-whitepaper
6. Ripio Credit Network website. https://ripiocredit.network/
7. White Paper - Democratizing Lending. http://bit.ly/2MCaIya
8. What is ETHLend? | Beginner’s Guide. https://coincentral.com/ethlend-beginner-guide/
9. The Loan Markets and Opportunities in Blockchain. https://iolite.io/static/pdf/iOlite-LoanPink.pdf

# Thanks for your patience