Ethereum Blockchain : Developing Smart Contracts

Problem statement

Analyze problem

Use class diagram to represent design

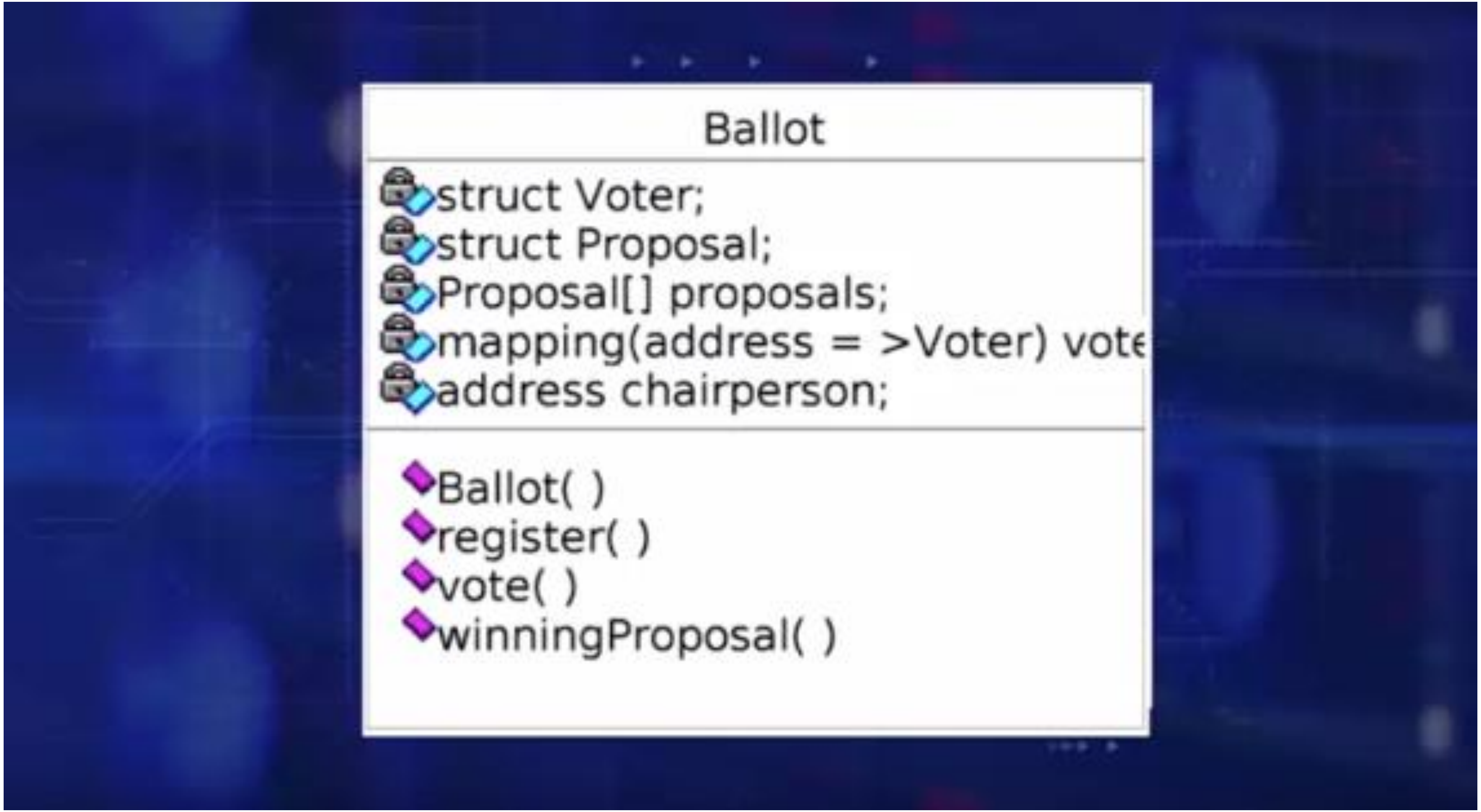
Define the visibility for the state variables and functions

Define access modifiers for the functions

Define validations for input variables of the functions

Define conditions that must hold true

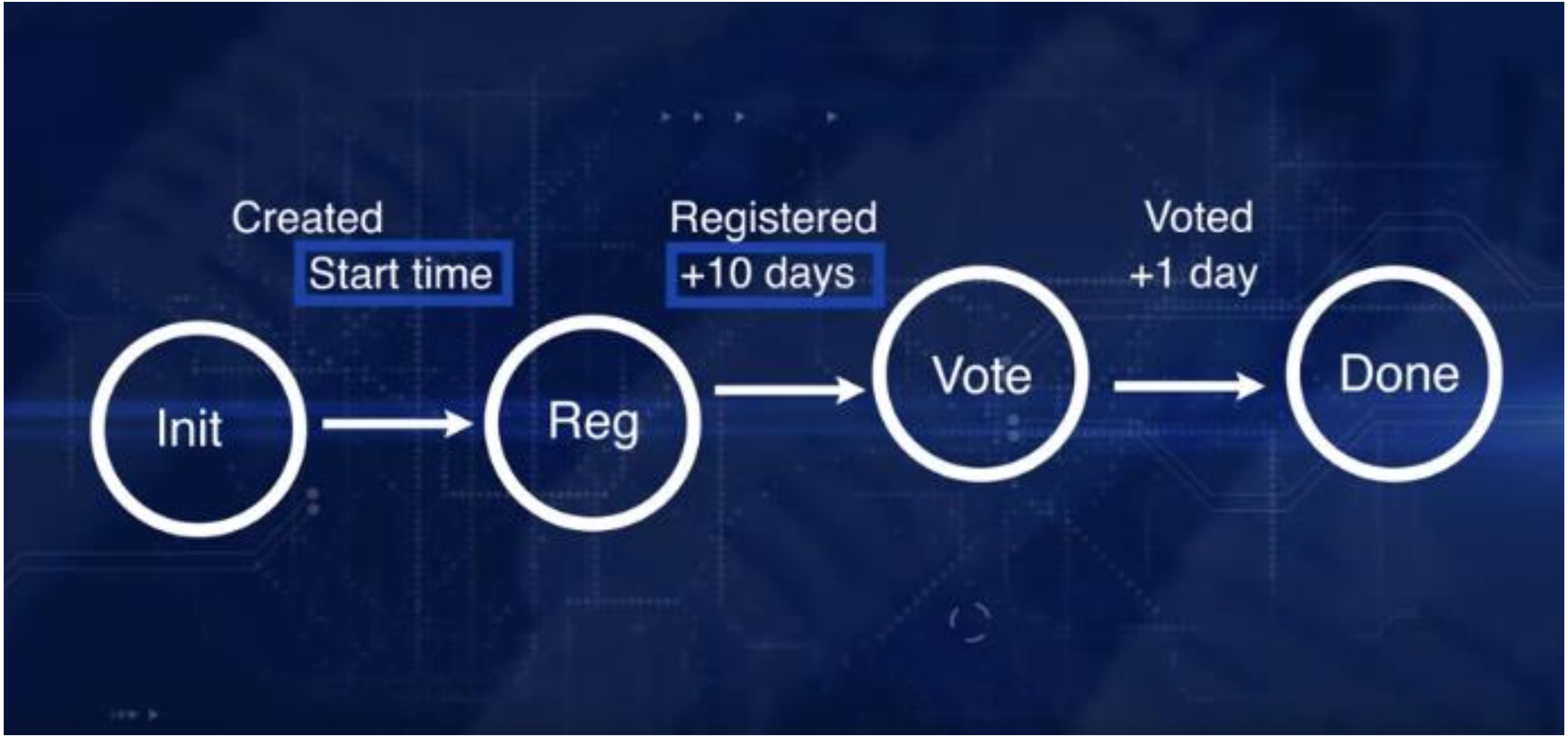
Express conditions that were discovered


Ballot
struct Voter;
struct Proposal;
Proposal[] proposals;
mapping(address = >Voter) vote
address chairperson;
Ballot( )
register( )
vote( )
winningProposal( )

Functions

- Constructor
- Register
- Vote
- Client Application

Time Elements

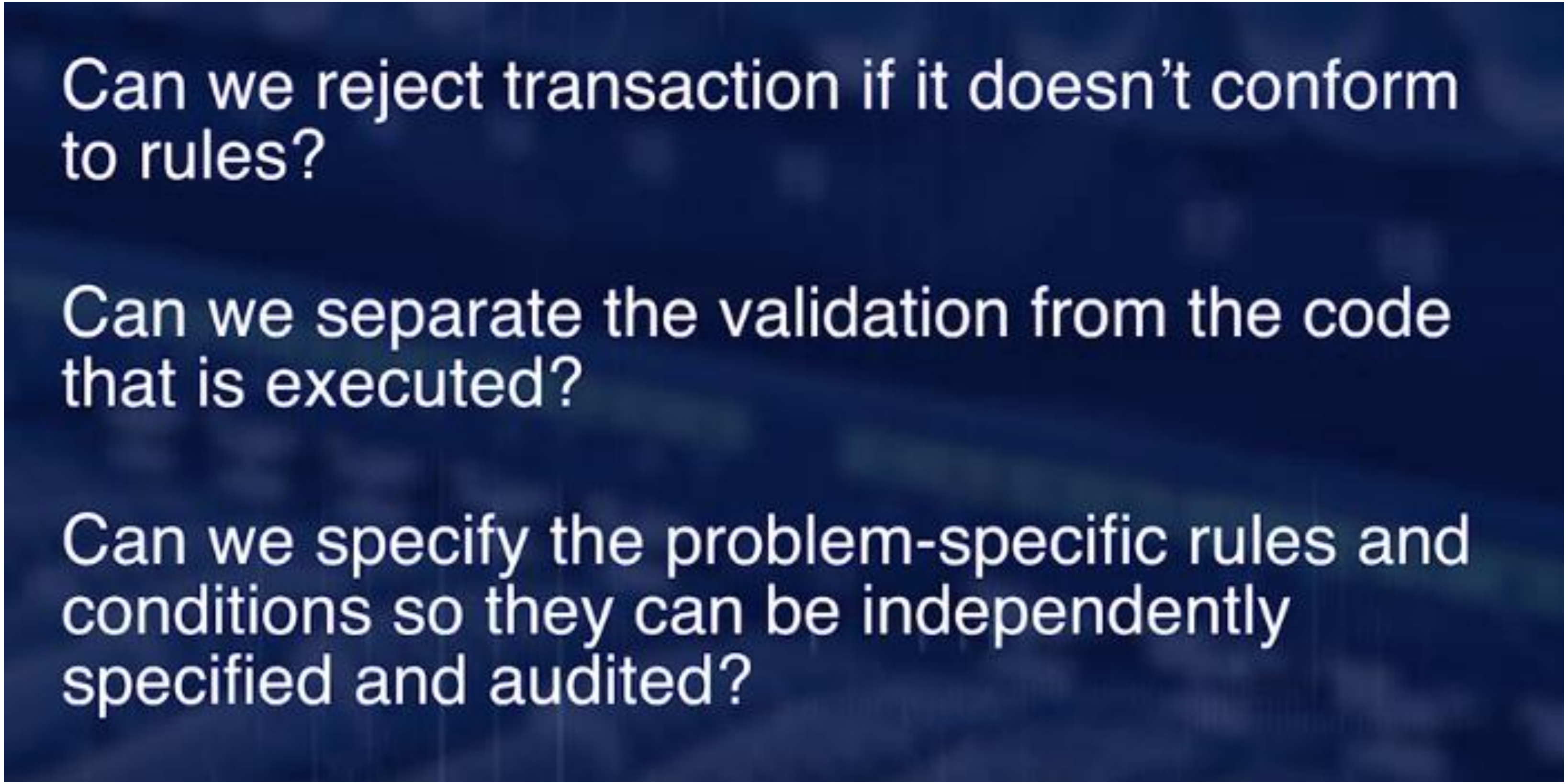


```
//Ballot with time elements: See Resources for full code
enum Stage {Init,Reg, Vote, Done}
Stage public stage = Stage.Init;

function register(address toVoter) public {
        if (stage != Stage.Reg) {return;}
....
}

function vote(uint8 toProposal) public  {
        if (stage != Stage.Vote) {return;}
...
}
```

Validation and Testing



Function Modifier

modifier validStage(Stage requiredStage)...;

Function Header

function vote(uint8 toProposal) public voteStage {
///////////////function code

Stage requiredStage

modifier validStage(Stage requiredStage)
{ require(stage == requiredStage);
_; }

Demo : BallotWithModifier.sol

Client Applications

Event Definition

```
event nameOfEvent(parameters);
```

Example:

```
event votingCompleted();
```

The application can listen to the events pushed, using a listener code, to:

Track transaction

Receive results

Initiate a pull request to receive information from a smart contract

In summary we developed the Ballot smart contract incrementally to illustrate:

Time dependencies

Validation outside the function code

Asserts and require declarations

Event logging

Capturing Smart Contract Events in our User Interface (Solidity)


Best practices for designing blockchain-based applications
Best practices for designing solutions with smart contracts using Solidity & Remix IDE

When to use a blockchain

Make sure your application requirements need blockchain features


Blockchain is most suitable for:
Decentralized problems
Peer-to-peer transactions
Beyond boundaries of trust among unknown peers
Require validation, verification & recording of timestamped, immutable ledger
Autonomous operations guided by rules & policies

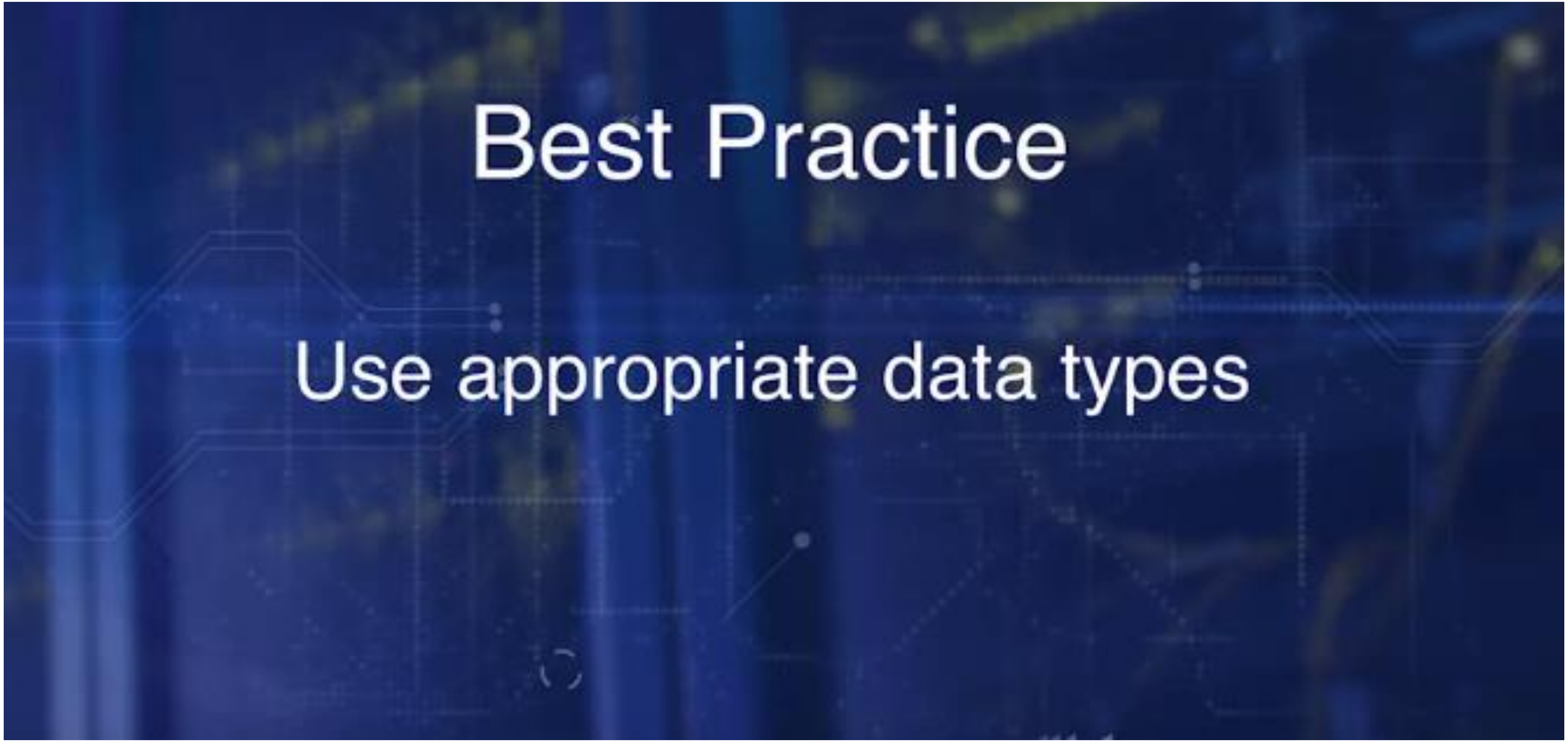

Blockchain is most suitable for:
Make sure you need a smart contract
on blockchain for your application


Best Practice
Keep the smart contract code simple, coherent, and auditable


Best Practice
Make smart contract functions auditable by using custom function modifiers


Best Practice
Keep only the necessary data in the smart contract


On-Chain Data
Off-Chain Data


Best Practice
Use appropriate data types


Best Practice
Understand the public visibility modifier for data


Best Practice
Maintain a standard order for the different function types within a smart contract


Best Practice
Functions can have many different modifiers

Best Practice

```
function buy(..) payable enoughMoney itemAvail returns (..)
```

Best Practice

Use Solidity-defined “payable” modifier when sending value

```
function deposit() payable {};
function register(address sender) payable {};
```


Best Practice
Use modifier declarations for implementing rules

Use function access modifiers for:

- Implementing rules, policies and regulations
- Implementing common rules for all who may access a function
- Declaratively validating application-specific conditions
- Providing auditable elements to allow verification of the correctness of a smart contract


Best Practice
Using events for notification

Best Practice

Beware of “now” time variable

Best Practice

Use secure hashing for protecting data

keccak256(...) returns (bytes32):

That computes the **Keccak**-256 hash of the parameters

sha256(...) returns (bytes32):

compute the SHA-256 hash of the parameters

ripemd160(...) returns (bytes20):

compute RIPEMD-160 hash of the parameter: used for address calculation


Best Practice
Pay attention to Remix static analysis

Reference:

Bina Ramamurthy, Smart Contracts [MOOC]. Coursera.
https://www.coursera.org/learn/smarter-contracts