Brief Introduction to Smart Contracts and Solidity

CS 731: Blockchain Technology and Applications

Instructor : Chavhan Sujeet Yashavant

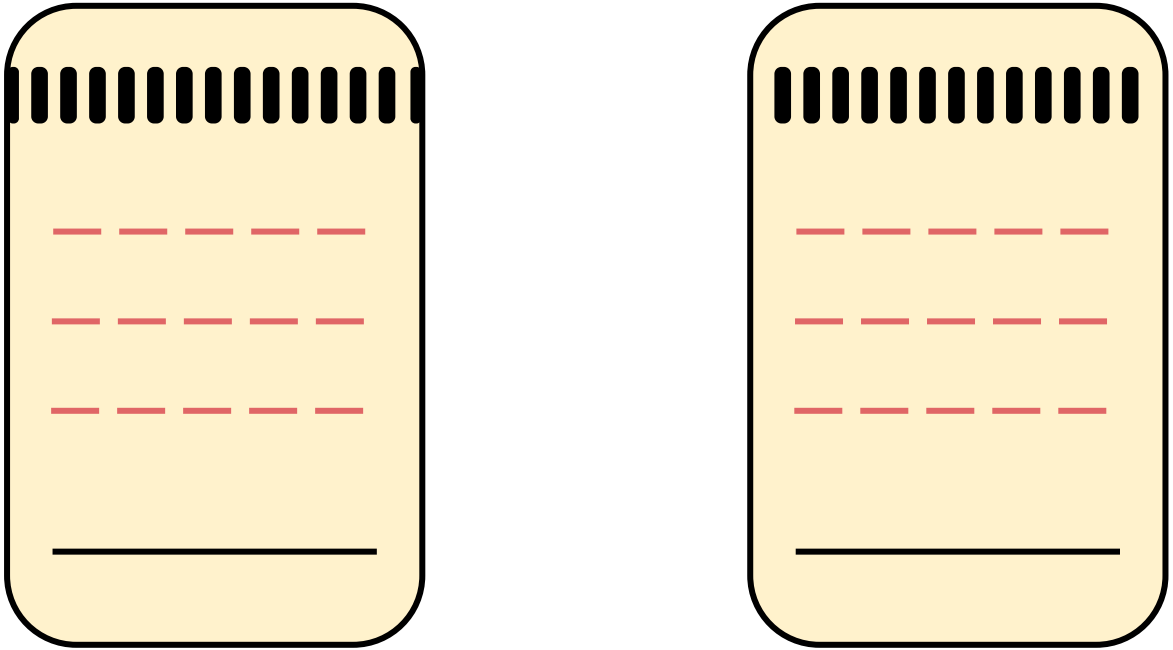
What is Ethereum?

It’s a Blockchain, With following additions

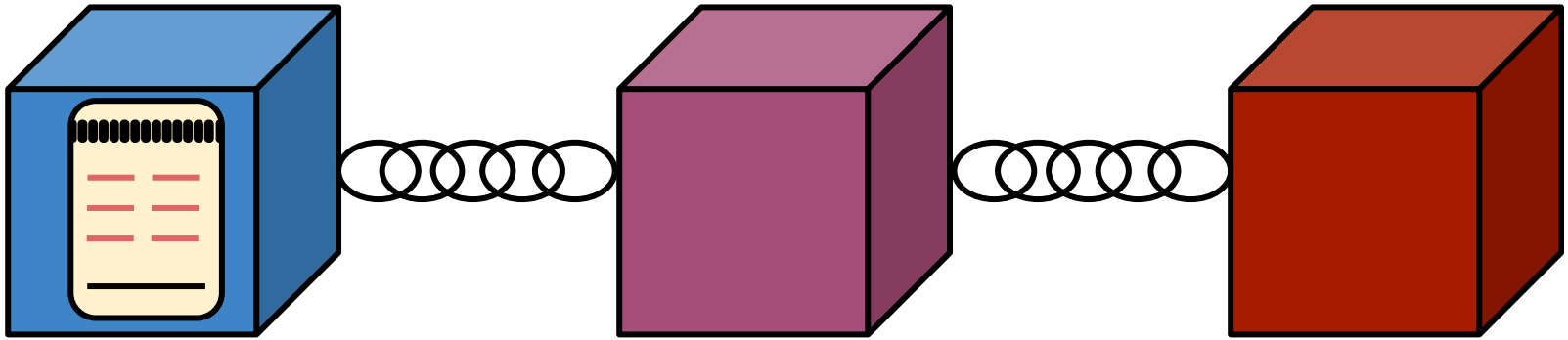
- A built-in programming Language
- Two types of accounts
  - User Accounts (Controlled by Private Keys)
  - Contract Accounts (Controlled by Code)
- Anyone can create an app by defining it as a Contract

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

Smart Contracts



Smart contracts



- Tiny computer programs
- Stored inside a blockchain

Ref: https://savjee.be/videos/simply-explained/smart-contracts/

Smart Contract

- A code that resides on blockchain
- Executes when certain predetermined conditions are satisfied

Smart Contract

- agreement between mutually distrusting participants
- automatically enforced by the consensus mechanism of the blockchain
- without relying on a trusted authority.

Ref: Atzei, Nicola, Massimo Bartoletti, and Tiziana Cimoli. "A survey of attacks on ethereum smart contracts (sok)." *International conference on principles of security and trust*. Springer, Berlin, Heidelberg, 2017.

What a Contract can Do?

- Send ETH to other contracts

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

What a Contract can Do?

- Send ETH to other contracts
- Read/write Storage

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

What a Contract can Do?

- Send ETH to other contracts
- Read/write Storage
- Call (i.e. start execution in) other Contracts

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

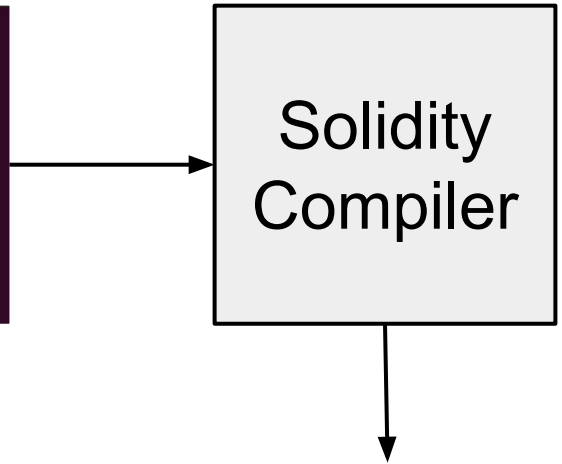
Smart Contract Execution

- Every (full) node on Ethereum network processes every transaction

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

Smart Contract Execution

```
pragma solidity ^0.4.17;
contract Inbox
{
        string public message;
        function Inbox(string initialMessage) public {
                message = initialMessage;
        }
        function SetMessage (string newMessage) public {
                message = newMessage;
        }
}
```



bytecode:

'6060604052341561000f57600080fd5b60405161038e38038061038e8339810160405280805190910190506000818051610003d9291602001906100445565b50506100df565b828054600181600116156101000203166002900480601f0160208091040260200160405190810160405280929190818152602001828054600181600116156101000203166002900480156101d15780601f106101a6576101008083540402835291602001916101d1565b820191906000526020600020905b8154815290600101906020018083116101b457829003601f168201915b505050505081565b828054600181600116156101000203166002900490600052602060002090601f016020900481019282601f1061021a57805160ff1916838001178555610247565b828001600101855582156102475791820158281111561024757825182559160200191906001019061022c565b5061025392915061025756509056b6102739291506102575650905600a165627a7a723058208f0f3e57d60457b58d09389d0da7d2d5375adecd20cfdb52a65ef21ee981d1440029',

Smart Contract Execution

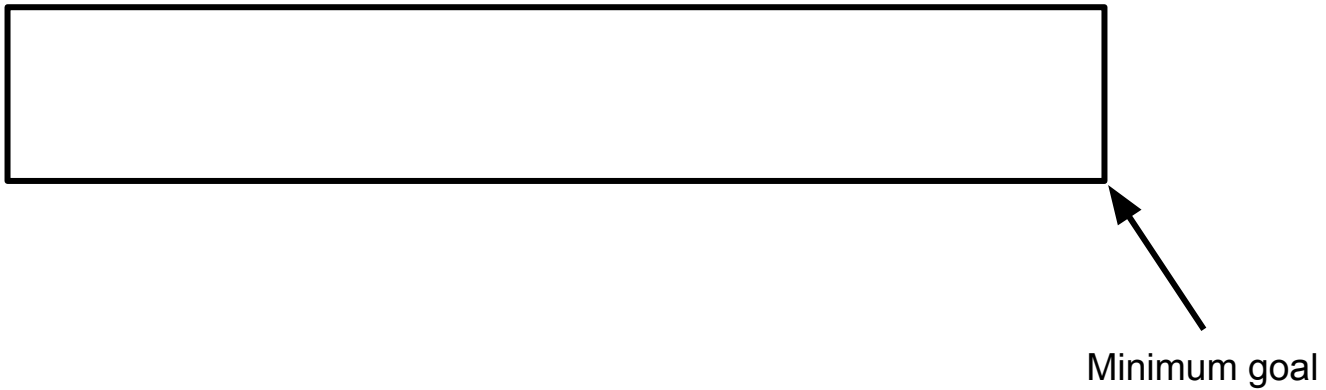
```
bytecode:
  '6060604052341561000f57600080fd5b60405161038e38038061038e83398101604052808051909101905060008180516100 3d9291602001906100445 65b50506100df565b828054600181600116156101000203166002900480601f0160208091040260200160405190810160405280929190818152602001828054600181600116156101000203166002900480156101d15780601f106101a657610100808354040283529160200191610 1d1565b820191906000526020600020905b81548152906001019060200180831161 01b457829003601f168201915b505050505081565b828054600181600116156101000203166002900490600052602060002090601f016020900481019282601f1061021a57805160ff1916838001178555610247565b828001600101855582156102475792182015b8281115610247578251825591602001919060010190610 1c565b506102539291506102575 65b5090565b61027191905b8082111561025357600081556001016102 5d565b905600a165627a7a723058208f0f3e57d60457b58d09389d0da7d2d5375adecd20cfdb52a65ef21ee981d1440029',
```

EVM (Ethereum Virtual Machine)

Ethereum Virtual Machine (EVM)

- Global Singleton Computing Machine with a shared ledger of data

Crowdfunding platform



Ref: https://savjee.be/videos/simply-explained/smart-contracts/

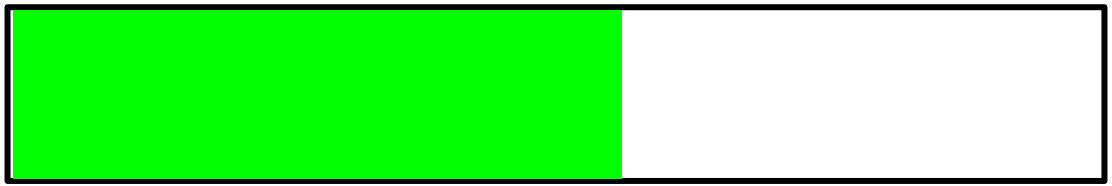
Crowdfunding platform

Ref: https://savjee.be/videos/simply-explained/smart-contracts/

Crowdfunding platform

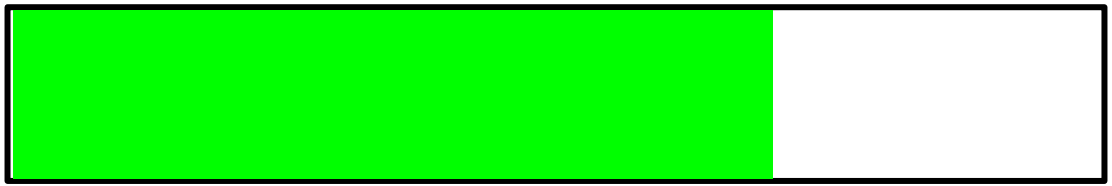
Ref: https://savjee.be/videos/simply-explained/smart-contracts/

Crowdfunding platform



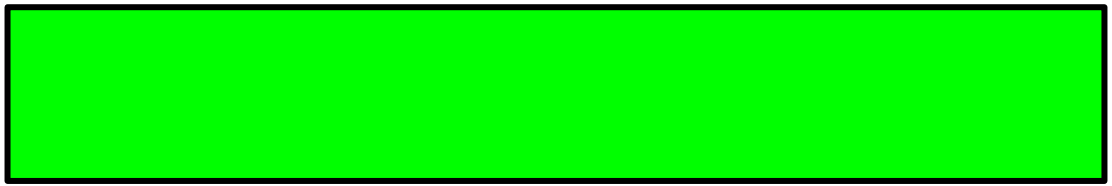
Ref: https://savjee.be/videos/simply-explained/smart-contracts/

Crowdfunding platform



Ref: https://savjee.be/videos/simply-explained/smart-contracts/

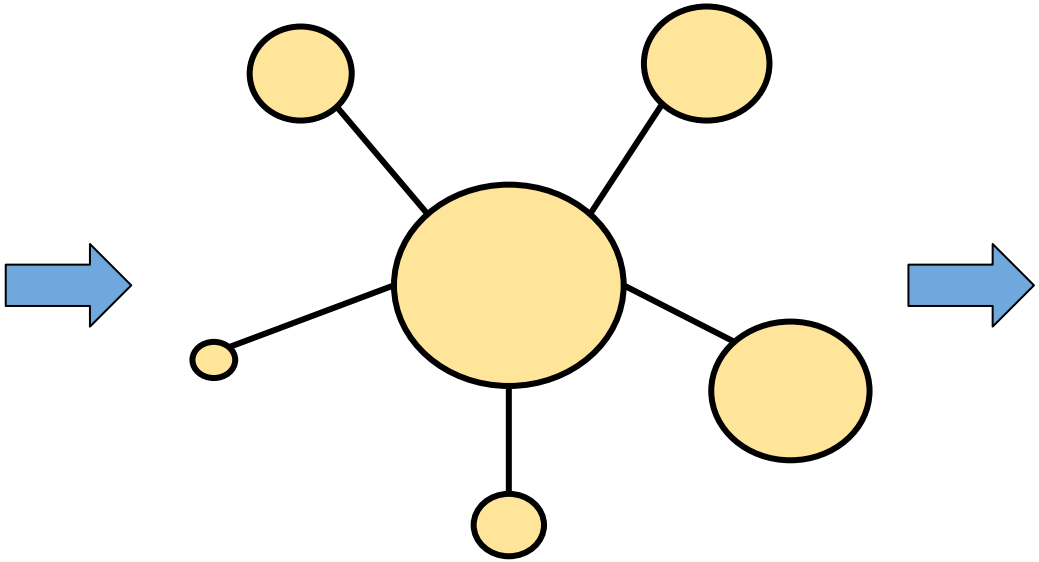
Crowdfunding platform



Funded!

Ref: https://savjee.be/videos/simply-explained/smart-contracts/

Kickstarter for Crowdfunding platform



Supporters

Product team

Ref: https://savjee.be/videos/simply-explained/smart-contracts/

Kickstarter for Crowdfunding platform

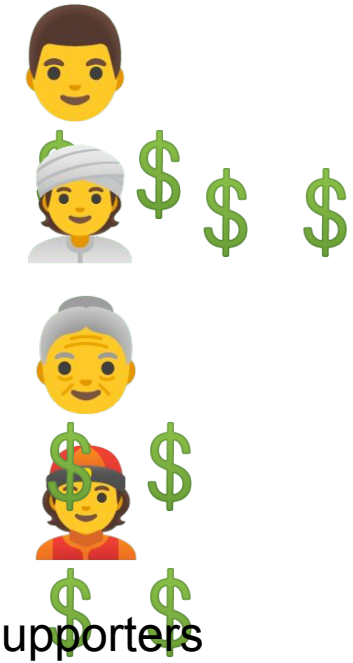
Trusting a third-party is required

Ref: https://savjee.be/videos/simply-explained/smart-contracts, https://pngimg.com

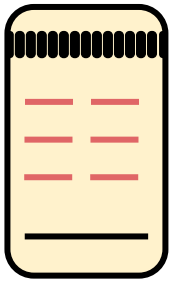
Smart contracts

We can build a similar system with a Smart Contracts without the requirement of any third party

Kickstarter with Smart Contract



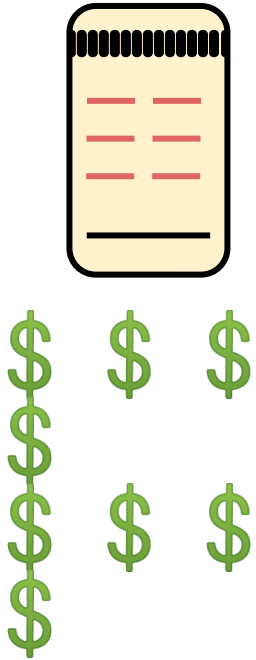
Supporters



Product team

Kickstarter with Smart Contract

Supporters

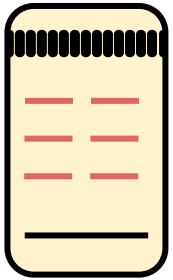


Product team

Kickstarter with Smart Contract

Supporters

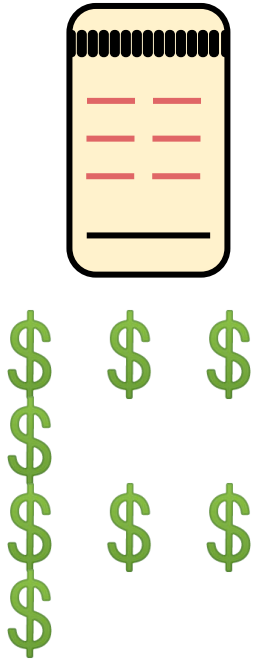
Funded!



Product team

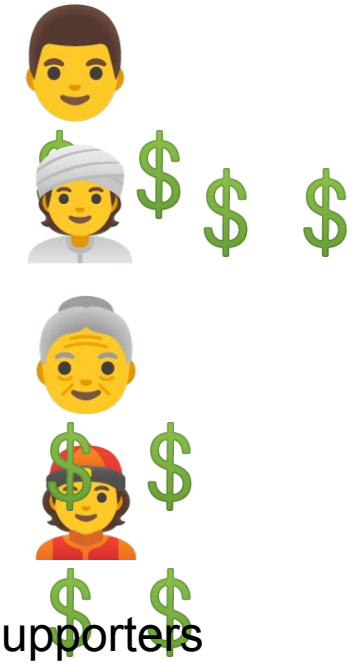
Kickstarter with Smart Contract

Supporters



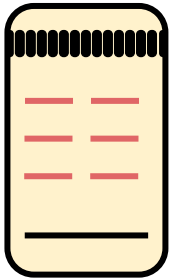
Product team

Kickstarter with Smart Contract



Supporters

Failed!



Product team

Introduction to Solidity

```
1  pragma solidity ^0.4.17;
2  contract Inbox {
3      string public message;
4      function Inbox(string initialMessage) public {
5          message = initialMessage;
6      }
7      function setMessage(string newMessage) public {
8          message = newMessage;
9      }
10     function getMessage(
11     ) public view returns (string) {
12         return message;
13     }
14 }
```

Introduction to Solidity: Version Pragma

```
1 pragma solidity ^0.4.17;
```

- Instructions to the compiler on how to treat the code.
- All solidity source code should start with a “version pragma” which is a declaration of the version of the solidity compiler this code should use.
- This helps the code from being incompatible with the future versions of the compiler which may bring changes.

Ref: https://www.geeksforgeeks.org/introduction-to-solidity

Introduction to Solidity: Contract keyword

```
1  pragma solidity ^0.4.17;
2  contract Inbox {
3      string public message;
4      function Inbox(string initialMessage) public {
5          message = initialMessage;
6      }
7      function setMessage(string newMessage) public {
8          message = newMessage;
9      }
10     function getMessage(
11     ) public view returns (string) {
12         return message;
13     }
14 }
```

It declares a contract under which is the code encapsulated.

Ref: https://www.geeksforgeeks.org/introduction-to-solidity

Introduction to Solidity: State Variables

```
1  pragma solidity ^0.4.17;
2  contract Inbox {
3      string public message;
4      function Inbox(string initialMessage) public {
5          message = initialMessage;
6      }
7      function setMessage(string newMessage) public {
8          message = newMessage;
9      }
10     function getMessage(
11     ) public view returns (string) {
12         return message;
13     }
14 }
```

Permanently stored in contract storage → written to Ethereum Blockchain.

Ref: https://www.geeksforgeeks.org/introduction-to-solidity

Introduction to Solidity: Function declaration

```
1  pragma solidity ^0.4.17;
2  contract Inbox {
3      string public message;
4      function Inbox(string initialMessage) public {
5          message = initialMessage;
6      }
7      function setMessage(string newMessage) public {
8          message = newMessage;
9      }
10     function getMessage(
11     ) public view returns (string) {
12         return message;
13     }
14 }
```

Constructor

Ref: https://www.geeksforgeeks.org/introduction-to-solidity

Introduction to Solidity: Function Visibility

Introduction to Solidity: Function Visibility

- Public - any contract and account can call
- Private - only inside the contract that defines the function
- External - only other contracts and accounts can call
- Internal - only inside contract that inherits an internal function

Ref: https://solidity-by-example.org/visibility/

Introduction to Solidity: View and Pure functions

- View function declares that no state will be changed.
- Pure function declares that no state variable will be changed or read.

Ref: https://solidity-by-example.org/visibility/

Introduction to Solidity: Code Execution on Real Blockchain (Try this after success on Local Blockchain)

- Testnet (most of the course projects will do it):
  - Can use **Remix** and **Metamask**
  - Can use **hardhat** to deploy on **Goerli Testnet**
- Mainnet
  - Require real money
  - Do not try unless you become expert

Ref: https://www.geeksforgeeks.org/introduction-to-solidity

Introduction to Solidity: Code Execution on Local Blockchain (Try this first)

- Offline (Blockchain inside local machine): I will post a video link on Discord about how to do it. It takes time.
  - Can use **Remix** and **Ganache**
- Online (Blockchain inside browser): Remix IDE
  - Simple one, first try this
  - Let’s see a Demo

Ref: https://www.geeksforgeeks.org/introduction-to-solidity

Crowdfunding Smart Contract

```
pragma solidity ^0.4.19;
contract Crowdfunding {
    address owner;
    uint256 deadline;
    uint256 goal;
    mapping(address => uint256) public pledgeOf;
    function Crowdfunding(uint256 numberOfDays, uint256 _goal) public {
        owner = msg.sender;
        deadline = now + (numberOfDays * 1 days);
        goal = _goal;
    }
    function pledge(uint256 amount) public payable {
        require(now < deadline);                // in the fundraising period
        require(msg.value == amount);
        pledgeOf[msg.sender] += amount;
    }
    function claimFunds() public {
        require(address(this).balance >= goal); // funding goal met
        require(now >= deadline);               // in the withdrawal period
        require(msg.sender == owner);

        msg.sender.transfer(address(this).balance);
    }
    function getRefund() public {
        require(address(this).balance < goal);  // funding goal not met
        require(now >= deadline);               // in the withdrawal period
        uint256 amount = pledgeOf[msg.sender];
        pledgeOf[msg.sender] = 0;
        msg.sender.transfer(amount);
    }
}
```

Currency Example

```
pragma solidity ^0.8.4;
contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping(address => uint) public balances;
    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);
    // Constructor code is only run when the contract
    // is created
    constructor() {
        minter = msg.sender;
    }
    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }
    // Errors allow you to provide information about
    // why an operation failed. They are returned
    // to the caller of the function.
    error InsufficientBalance(uint requested, uint available);
    // Sends an amount of existing coins
    // from any caller to an address
    function send(address receiver, uint amount) public {
        if (amount > balances[msg.sender])
            revert InsufficientBalance({
                requested: amount,
                available: balances[msg.sender]
            });
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Gas

- Halting problem
  - Can’t tell whether a program will halt or run infinitely

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

Gas

- Halting problem
  - Can’t tell whether a program will halt or run infinitely
- Solution: Gas Limit

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

Gas Limit

- Each opcode has a fixed amount of gas assigned and is a measure of computational effort
- Gas is the execution fee, paid by the sender of the transaction that triggered the computation

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

Gas Limit

- User sets max amount of Gas for a transaction
- Gas is lost if a user run out of Gas Limit, all changes are reversed
- If a transaction uses less gas than gas limit, then user gets remaining Gas

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

Gas Limit

- Total fees = Total amount of Gas used ❌ gas Price
- The gas price is not fixed

Ref: DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin

Why Optimization?

- Caller needs to pay Gas according to Computational Steps

Ref: Brandstätter, Tamara, et al. "Characterizing Efficiency Optimizations in Solidity Smart Contracts." *2020 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2020.

Why Optimization?

- Caller needs to pay Gas according to Computational Steps
- Optimization ⇒ Gas Saving ⇒ Money Saving

Ref: Brandstätter, Tamara, et al. "Characterizing Efficiency Optimizations in Solidity Smart Contracts." *2020 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2020.

Why Optimization?

- Caller needs to pay Gas according to Computational Steps
- Optimization ⇒ Gas Saving ⇒ Money Saving
- A smart contract gets invoked many times, a small optimization can result in huge saving

Ref: Brandstätter, Tamara, et al. "Characterizing Efficiency Optimizations in Solidity Smart Contracts." *2020 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2020.

Gas Costly Pattern 1: Dead Code

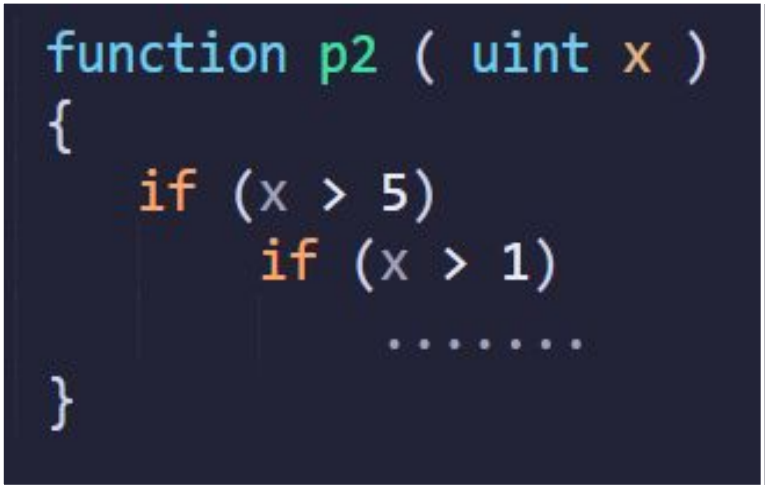
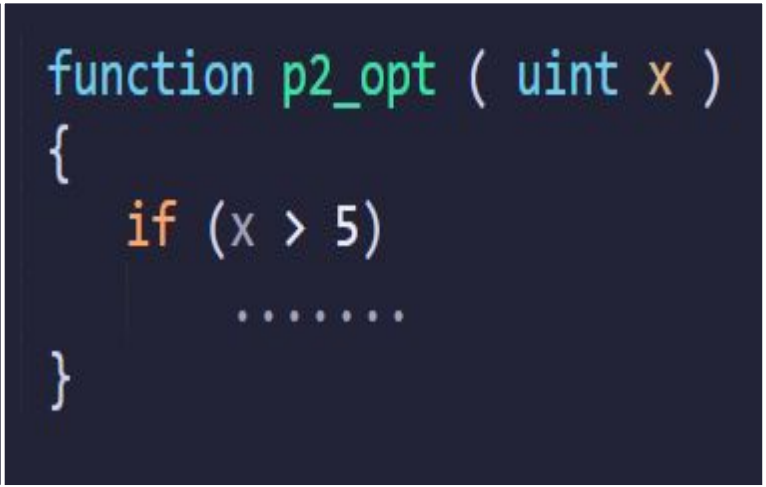
Unoptimized Code	Optimized Code

```
function p1 ( uint x )
{
    if (x > 5)
    {
        if (x*x < 20)
            Statement 1
        Statement 2
    }
    Statement 3
}
```

```
function p1_opt ( uint x )
{
    if (x > 5)
    {
        Statement 2
    }
    Statement 3
}
```

Ref: Chen, Ting, et al. "Under-optimized smart contracts devour your money." *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017.

Gas Costly Pattern 2: Opaque Predicate

Unoptimized Code	Optimized Code
	

Ref: Chen, Ting, et al. "Under-optimized smart contracts devour your money." *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017.

Gas Costly Pattern 3: Constant outcome of a Loop

```
1 function p4 returns (uint) {
2         uint sum = 0;
3         for (uint i = 1; i <= 100; i++)
4                 sum += i;
5         return sum;
6 }
```

```
1 function p4_opt returns (uint) {
2         return 5050;
3 }
```

Ref: Chen, Ting, et al. "Under-optimized smart contracts devour your money." *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017.

Gas Costly Pattern 4: Comparison with Unilateral Outcome in a Loop

```
1 function p7 (uint x, uint y) returns (uint)
2 {
3         for (int i = 0; i < 100; i++)
4                 if (x > 0)
5                         y += x;
6         return y;
7 }
```

```
1 function p7_opt (uint x, uint y) returns (uint)
2 {
3         if (x > 0)
4                 for (int i = 0; i < 100; i++)
5                         y += x;
6         return y;
7 }
```

Ref: Chen, Ting, et al. "Under-optimized smart contracts devour your money." *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017.

Smart Contract Security

- Correctness is ensured by the consensus mechanism
- Unfortunately, correctness is not sufficient to make Smart Contracts secure.

Ref: Atzei, Nicola, Massimo Bartoletti, and Tiziana Cimoli. "A survey of attacks on ethereum smart contracts (sok)." *International conference on principles of security and trust*. Springer, Berlin, Heidelberg, 2017.

Smart Contract Vuln 1: Overflow and Underflow

```
1  mapping (address => uint256) public balanceOf;
2
3  // INSECURE
4  function transfer(address _to, uint256 _value) {
5      /* Check if sender has balance */
6      require(balanceOf[msg.sender] >= _value);
7      /* Add and subtract new balances */
8      balanceOf[msg.sender] -= _value;
9      balanceOf[_to] += _value;
10 }
11
12 // SECURE
13 function transfer(address _to, uint256 _value) {
14     /* Check if sender has balance and for overflows */
15     require(balanceOf[msg.sender] >= _value &&
16       balanceOf[_to] + _value >= balanceOf[_to]);
17     /* Add and subtract new balances */
18     balanceOf[msg.sender] -= _value;
19     balanceOf[_to] += _value;
20 }
```

Ref: Sayeed, Sarwar, Hector Marco-Gisbert, and Tom Caira. "Smart contract: Attacks and protections." IEEE Access 8 (2020): 24416-24427.

Smart Contract Vuln 2: Default Visibilities

```
1  contract Puzzle {
2    uint256 amount = 0.5; //0.5 ethers
3    function submit_answer (string answer) {
4      //Logic to check submitted answer
5      if (/*answer is correct*/)
6        _sendEther();
7    }
8    function _sendEther(){
9      msg.sender.tranfer(amount);
10   }
11 }
```

Ref: Sayeed, Sarwar, Hector Marco-Gisbert, and Tom Caira. "Smart contract: Attacks and protections." IEEE Access 8 (2020): 24416-24427.

Smart Contract Vuln 3: Timestamp Dependence

- A smart contract that utilizes a current timestamp to produce random numbers in order to determine lottery results
- Miners can put a timestamp within 30 seconds of block validation
- Miners can alter outcome of random number generator

Ref: Sayeed, Sarwar, Hector Marco-Gisbert, and Tom Caira. "Smart contract: Attacks and protections." IEEE Access 8 (2020): 24416-24427.

Smart Contract Vuln 3: Timestamp Dependence

```
1  pragma solidity ^0.5.0;
2  contract TimedCrowdsale {
3    event Finished();
4    event notFinished();
5    // Sale should finish exactly at January 1, 2019
6    function isSaleFinished() private returns (bool) {
7      return block.timestamp >= 1546300800;
8    }
9    function run() public {
10     if (isSaleFinished()) {
11         emit Finished();
12     } else {
13         emit notFinished();
14     }
15   }
16 }
```

Ref: https://swcregistry.io/docs/SWC-116

THE END

Backup Slides

Gas assigned per Opcode

Operation	Gas	Description
ADD/SUB	3	Arithmetic Operation
MUL/DIV	5	
ADDMOD/MULMOD	8	
AND/OR/XOR	3	Comparison Operation
LT/GT/SLT/SGT/EQ	2	Stack Operation
POP	3	

Ref: Wood, Gavin. "Ethereum: A secure decentralised generalised transaction ledger." *Ethereum project yellow paper* (2014)

Gas assigned per Opcode

Operation	Gas	Description
BALANCE	400	Get balance of an account
CREATE	32000	Create a new account using CREATE
CALL	25000	Message-call into an account

Ref: Wood, Gavin. "Ethereum: A secure decentralised generalised transaction ledger." *Ethereum project yellow paper* (2014)

Gas assigned per Opcode

Operation	Gas	Description
MLOAD/MSTORE	3	Memory Operation
JUMP	8	Unconditional Jump
JUMPI	10	Conditional Jump
SLOAD	200	Storage Operation
SSTORE	5000/20000	

Ref: Wood, Gavin. "Ethereum: A secure decentralised generalised transaction ledger." *Ethereum project yellow paper* (2014)