ETHWORKS

ETHEREUM MIMUW ’19

EVENTS

Ether splitter

```
contract Splitter {

    address benficary1;

    address benficary2;

    function Splitter(address _benficary1, address _benficary2) public
{
        benficary1 = _benficary1;
        benficary2 = _benficary2;
    }

    function () payable public {
        uint amount = msg.value / 2;
        benficary1.transfer(amount);
        benficary2.transfer(msg.value - amount);
    }

}
```

From web3

```
splitter = ...
splitter.events.MyEvent({
    filter: {from: '0x123...456'},
    fromBlock: 0
})
.on('data', (event) => {
    console.log(event);
})
.on('error', console.error);
```

EVENTS

- Allow notifying the external world that something happened
- Cheaper than storage
- Allow filtering by a specific type
- No guarantee of support

ETHWORKS

EVENT COSTS

Instruction	Cost
Arithmetics	1-10
SHA3	30
sstore	20000 gas - first write to a new position 5000 gas - subsequent writes to an existing
log	~ 375 gas + 8gas per byte + 375 per indexed parameter

SAFE MATH

Underflow

```
contract EtherSale {

  int balance = 0;

  function withdraw(int amount) public constant returns (int) {
    // ...
    return balance - amount;
  }

}


EtherSale.new().withdraw(1) == -1
```

Overflow

```
contract EtherSale {

  uint balance = 0;

  function withdraw(uint amount) public constant returns (uint) {
    // ...
    return balance - amount;
  }

}



EtherSale.new().withdraw(1) ==
115792089237316195423570985008687907853269984665640564039457584007913129639935
```

SafeMath

```
library SafeMath {
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by
0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this
doesn't hold
    return c;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}
```

TOKENS

Basic Token

```
pragma solidity ^0.4.23;


contract ERC20Basic {
  uint256 public totalSupply;
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}
```

Basic Token implementation

```
import './ERC20Basic.sol';
import '../math/SafeMath.sol';


contract BasicToken is ERC20Basic {
  using SafeMath for uint256;

  mapping(address => uint256) balances;

 function transfer(address _to, uint256 _value) public returns (bool) {
    require(_to != address(0));
    require(_value <= balances[msg.sender]);

    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    Transfer(msg.sender, _to, _value);
    return true;
  }

 function balanceOf(address _owner) public view returns (uint256 balance) {
    return balances[_owner];
  }

}
```

TOKENS AND CONTRACTS

Token Sale

```
contract TokenSale {
  using SafeMath for uint256;

  Token public token = Token(0x0);

  address public wallet = …;

  function buyToken() public payable {
    uint256 weiAmount = msg.value;
    uint256 tokens = weiAmount.mul(10);
    token.transfer(msg.sender, tokens);
    wallet.transfer(msg.value);
  }

}
```

MORE COMPLETE TOKEN SALE

- Dates (start, end)
- Sales limit (i.e. hard cap)
- Configurable treasury and token
- Rate
- Preico
- Allocations

Token Sale

```
contract TokenSale {
  using SafeMath for uint256;

  Token public token = Token(…);

  address public wallet = …;

  function buyToken() public payable {
    uint256 weiAmount = msg.value;
    uint256 tokens = weiAmount.mul(10);
    token.transfer(msg.sender, tokens);
    wallet.transfer(msg.value);
  }

}
```

ΞTHWORKS

Ether Sale

```
contract EtherSale {
  using SafeMath for uint256;

  Token public token = Token(…);

  address public wallet = …;

  function buyEther() public {
    uint256 weiAmount = ?; //Eeeee…
    uint256 etherAmount = weiAmount.div(10);
    wallet.transfer(msg.sender, etherAmount); //Wait what?
  }

}
```

?

ERC20 Token

```
contract ERC20 is ERC20Basic {
  function allowance(address owner, address spender) public view returns (uint256);
  function transferFrom(address from, address to, uint256 value) public returns (bool);
  function approve(address spender, uint256 value) public returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

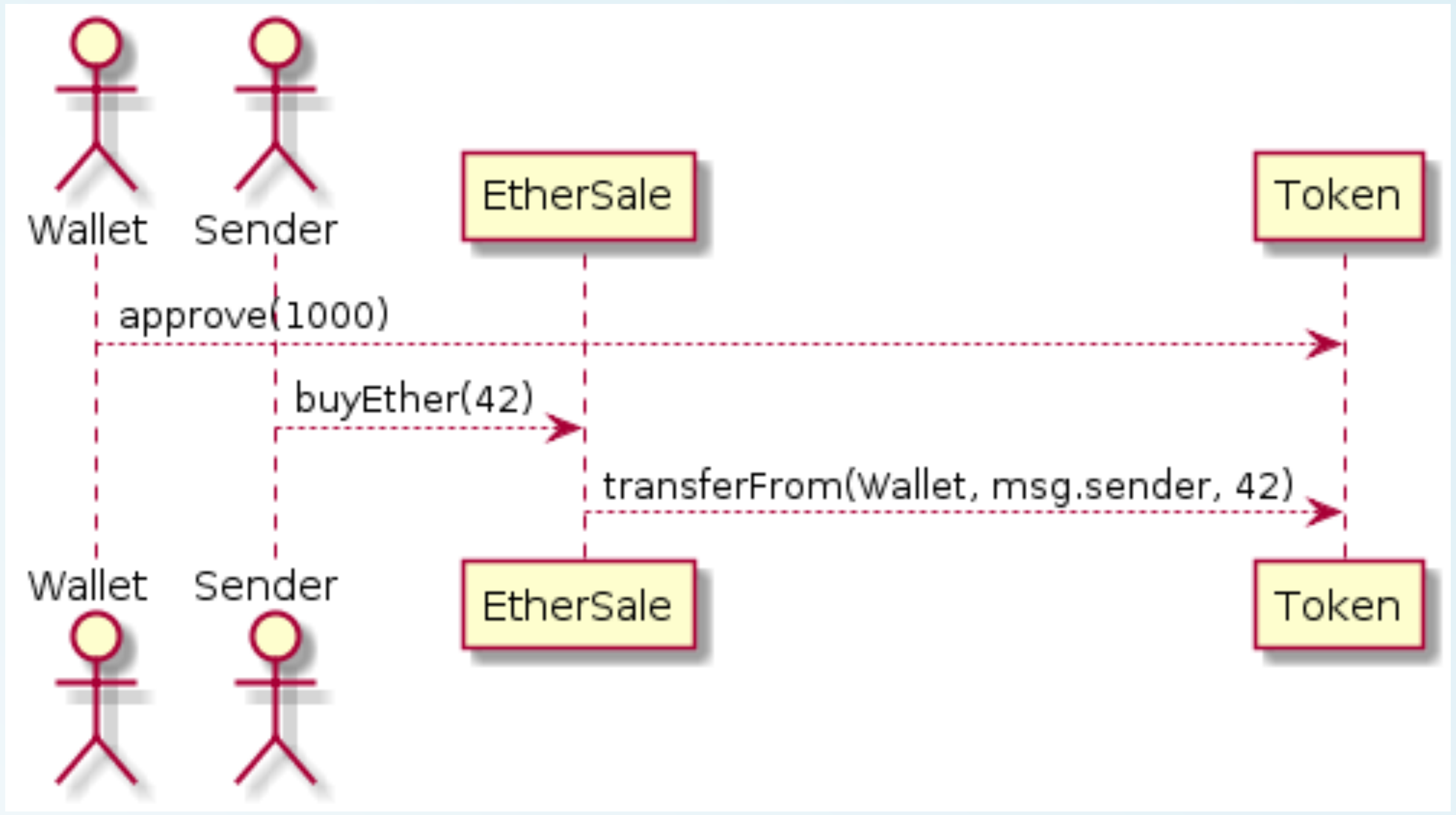
ERC20 Token implementation

```
contract StandardToken is ERC20, BasicToken {

  mapping (address => mapping (address => uint256)) internal allowed;

  function transferFrom(address _from, address _to, uint256 _value) public returns (bool) {
    require(_to != address(0));
    require(_value <= balances[_from]);
    require(_value <= allowed[_from][msg.sender]);

    balances[_from] = balances[_from].sub(_value);
    balances[_to] = balances[_to].add(_value);
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
    Transfer(_from, _to, _value);
    return true;
  }

  function approve(address _spender, uint256 _value) public returns (bool) {
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);
    return true;
  }

  function allowance(address _owner, address _spender) public view returns (uint256) {
    return allowed[_owner][_spender];
  }

}
```

Ether Sale

```
contract EtherSale {
  using SafeMath for uint256;

  Token public token = Token(…);

  address public wallet = …;

  function buyEther(uint weiAmount) public {
    uint256 etherAmount = weiAmount.div(10);
    wallet.transferFrom(wallet, msg.sender, etherAmount);
  }

}



//Token holder:
Token.approve(etherSaleAddress, 20);

//Sender:
EtherSale.buyEther(20);
```

Ether Sale


Wallet
Sender
EtherSale
Token
approve(1000)
buyEther(42)
transferFrom(Wallet, msg.sender, 42)
Wallet
Sender
EtherSale
Token

HOMEWORK

Ether splitter

```
contract Splitter {

    address benficary1;

    address benficary2;

    function Splitter(address _benficary1, address _benficary2) public {
        benficary1 = _benficary1;
        benficary2 = _benficary2;
    }

    function () payable public {
        uint amount = msg.value / 2;
        benficary1.transfer(amount);
        benficary2.transfer(msg.value - amount);
    }

}
```

Token splitter

```
contract Splitter {

    function Splitter(address [] beneficiaries, address feeCollector) public;

    function split(…) public;

    …
}
```

WRITE A TOKEN SPLITTER

- Write a contract doing a token split
- Take constant in tokens

- Cover with tests
- Use approval method
- No unauthorized withdraws!
- No out of gas exceptions
  (up to reasonable beneficiaries number)

- To be checked at next lab

Thank you.

ETHWORKS