**Ethereum**

*Recall: Bitcoin scripting:*
- non Turing-complete
- limited/limiting

**Solutions:**
- Add *application-specific functionality* in scripting of altcoin
- Create altcoin with *Turing-complete scripting*

---

**Smart Contracts Programming Model**

**Smart Contract** (Nick Szabo 1994):
“... is a computerized transaction protocol that executes the terms of a contract.”

Classical example of mechanized contract:
- *Vending Machine*

In Ethereum, contracts are stored in the blockchain.
Written in *Solidity* programming language
Users can make *procedure calls* to the contract.
Example: Namecoin in Ethereum

```solidity
contract NameRegistry {
    mapping(bytes32 => address) public registryTable;
    function claimName(bytes32 name) {
        if (msg.value < 10) {
            throw;
        }
        if (registryTable[name] == 0) {
            registryTable[name] = msg.sender;
        }
    }
}
```

Namecoin in Ethereum – Improvements

1. Add more data with mapping.
2. Require periodic re-registration
   - “last-updated” field
3. Add function to withdraw funds
   - Ethereum supports security features
Example: Simple Currency

```solidity
contract MyToken {  
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;

    /* Initializes contract with initial supply tokens to the creator of the contract */
    function MyToken(  
        uint256 initialSupply
    ) {  
        balanceOf[msg.sender] = initialSupply; // Give the creator all initial tokens
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) {  
        if (balanceOf[msg.sender] < _value) throw; // Check if sender has enough
        if (balanceOf[_to] + _value < balanceOf[_to]) throw; // Check for overflows
        balanceOf[msg.sender] -= _value; // Subtract from the sender
        balanceOf[_to] += _value; // Add the same to the recipient
    }
}
```

Some more basic Information

```solidity
/* Initializes contract with initial supply tokens to the creator of the contract */
function MyToken(uint256 initialSupply,
    string tokenName,
    uint8 decimalUnits,
    string tokenSymbol) {  
    balanceOf[msg.sender] = initialSupply; // Give the creator all initial tokens
    name = tokenName; // Set the name for display purposes
    symbol = tokenSymbol; // Set the symbol for display purposes
    decimals = decimalUnits; // Amount of decimals for display purposes
}
```
Events

/* This generates a public event on the blockchain that will notify clients */
event Transfer(address indexed from, address indexed to, uint256 value);

/* Send coins */
function transfer(address _to, uint256 _value) {
    if (_to == 0x0) throw;            // Prevent transfer to 0x0 address
    if (balanceOf[msg.sender] < _value) throw; // Check if sender has enough
    if (balanceOf[_to] + _value < balanceOf[_to]) throw; // Check for overflows
    balanceOf[msg.sender] -= _value;  // Subtract from the sender
    balanceOf[_to] += _value;         // Add the same to the recipient
    Transfer(msg.sender, _to, _value); // Notify anyone listening that this
    transfer took place
}

Ethereum Wallet
Deploying a Contract

[Image of a computer screen showing Ethereum wallet with contract deployment options]

Deploying a Contract

[Image of a computer screen showing the source code of a Ethereum contract]

Deploying a Contract

Using the Contract
Watch Token...

Improvement: Add Central Administrator

```solidity
contract owned {
    address public owner;

    function owned() {
        owner = msg.sender;
    }

    modifier onlyOwner {
        if (msg.sender != owner) throw;
        _;
    }

    function transferOwnership(address newOwner) onlyOwner {
        owner = newOwner;
    }
}
```
Adding Central Administrator

contract MyToken is owned {
    /* Rest of the contract as usual */
}

/* This creates an array with all balances */
mapping (address => uint256) public balanceOf;

/* Initializes contract with initial supply tokens to the creator of the contract */
function MyToken(
    uint256 initialSupply
) {
    balanceOf[msg.sender] = initialSupply; // Give the creator all initial tokens
}

/* Send coins */
function transfer(address _to, uint256 _value) {
    if (balanceOf[msg.sender] < _value) throw; // Check if the sender has enough
    if (balanceOf[_to] + _value < balanceOf[_to]) throw; // Check for overflows
    balanceOf[msg.sender] -= _value; // Subtract from the sender
    balanceOf[_to] += _value; // Add the same to the recipient
}
Cryptocurrency Technologies

Ethereum

Central Mint

```solidity
function mintToken(address target, uint256 mintedAmount) onlyOwner {
    balanceOf[target] += mintedAmount;
    totalSupply += mintedAmount;
    Transfer(0, owner, mintedAmount);
    Transfer(owner, target, mintedAmount);
}
```

Improvement: Freezing Assets

```solidity
mapping (address => bool) public frozenAccount;

event FrozenFunds(address target, bool frozen);

function freezeAccount(address target, bool freeze) onlyOwner {
    frozenAccount[target] = freeze;
    FrozenFunds(target, freeze);
}

function transfer(address _to, uint256 _value) {
    if (frozenAccount[msg.sender]) throw;
    ...
}
```
**Improvement: Automatic Selling and Buying**

```solidity
uint256 public sellPrice;
uint256 public buyPrice;

function setPrices(uint256 newSellPrice, uint256 newBuyPrice) onlyOwner {
    sellPrice = newSellPrice;
    buyPrice = newBuyPrice;
}
```

**Automated Selling and Buying**

```solidity
function buy() payable returns (uint amount) {
    amount = msg.value / buyPrice;  // calculates the amount
    if (balanceOf[msg.sender] < amount) throw;  // checks if it has enough to sell
    balanceOf[msg.sender] += amount;  // adds the amount to buyer's balance
    balanceOf[this] -= amount;  // subtracts same from seller's bal.
    Transfer(this, msg.sender, amount);  // execute an event about the change
    return amount;  // ends function and returns
}
```
Automated Selling and Buying

```Solidity
function sell(uint amount) returns (uint revenue) {
    if (balanceOf[msg.sender] < amount) throw; // checks if the sender has enough to sell
    balanceOf[this] += amount; // adds the amount to owner's balance
    balanceOf[msg.sender] -= amount; // subtracts amount from seller's balance
    revenue = amount * sellPrice;
    if (!msg.sender.send(revenue)) { // sends ether to the seller: it's important
        throw; // to do this last
        // to prevent recursion attacks
    } else {
        Transfer(msg.sender, this, amount); // executes event reflecting on the change
        return revenue; // ends function and returns
    }
}
```

Improvement: Autorefill

```Solidity
uint minBalanceForAccounts;

function setMinBalance(uint minimumBalanceInFinney) onlyOwner {
    minBalanceForAccounts = minimumBalanceInFinney * 1 finney;
}

/* Send coins */
function transfer(address _to, uint256 _value) {
    ...
    if(_to.balance<minBalanceForAccounts)
        _to.send(sell((minBalanceForAccounts-_to.balance)/sellPrice));
}
```
**Improvement: Proof-of-Work**

```solidity
def function giveBlockReward() {
    balanceOf[block.coinbase] += 1;
}
```

**Example: Crowdfunding**

```solidity
class token {
    function transfer(address receiver, uint amount) {
    }
}

class Crowdsale {
    address public beneficiary;
    uint public fundingGoal; uint public amountRaised; uint public deadline;
    uint public price;
    token public tokenReward;
    mapping(address => uint256) public balanceOf;
    bool fundingGoalReached = false;
    event GoalReached(address beneficiary, uint amountRaised);
    event FundTransfer(address backer, uint amount, bool isContribution);
    bool crowdsaleClosed = false;

    /* data structure to hold information about campaign contributors */
    ...
}
```
Crowdfunding

/* at initialization, setup the owner */
function Crowdsale(
    address ifSuccessfulSendTo,
    uint fundingGoalInEthers,
    uint durationInMinutes,
    uint etherCostOfEachToken,
    token addressOfTokenUsedAsReward
) {
    beneficiary = ifSuccessfulSendTo;
    fundingGoal = fundingGoalInEthers * 1 ether;
    deadline = now + durationInMinutes * 1 minutes;
    price = etherCostOfEachToken * 1 ether;
    tokenReward = token(addressOfTokenUsedAsReward);
}

Crowdfunding: Raise Funds

/* The function without name is the default function
that is called whenever anyone sends funds to a contract */
function () payable {
    if (crowdsaleClosed) throw;
    uint amount = msg.value;
    balanceOf[msg.sender] = amount;
    amountRaised += amount;
    tokenReward.transfer(msg.sender, amount / price);
    FundTransfer(msg.sender, amount, true);
}
Crowdfunding: Deadline

```solidity
modifier afterDeadline() { if (now >= deadline) _; }

/* checks if goal / time limit has been reached; ends the campaign */
function checkGoalReached() afterDeadline {
    if (amountRaised >= fundingGoal) {
        fundingGoalReached = true;
        GoalReached(beneficiary, amountRaised);
    }
    crowdsaleClosed = true;
}
```

Closing Crowdfunding Round (1)

```solidity
function safeWithdrawal() afterDeadline {
    if (!fundingGoalReached) {
        uint amount = balanceOf[msg.sender];
        balanceOf[msg.sender] = 0;
        if (amount > 0) {
            if (msg.sender.send(amount)) {
                FundTransfer(msg.sender, amount, false);
            } else {
                balanceOf[msg.sender] = amount;
            }
        }
    }

    if (fundingGoalReached && beneficiary == msg.sender) {
        ...
    }
}
```
Closing Crowdfunding Round (2)

```solidity
function safeWithdrawal() afterDeadline {
    if (!fundingGoalReached) {
        ...
    }
    if (fundingGoalReached && beneficiary == msg.sender) {
        if (beneficiary.send(amountRaised)) {
            FundTransfer(beneficiary, amountRaised, false);
        } else {
            // If we fail to send the funds to beneficiary, unlock funders balance
            fundingGoalReached = false;
        }
    }
}
```