Solidity Contracts :

- Function Overloading

Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

https://www.udemy.com/rest-api/?couponCode=DKRST1099

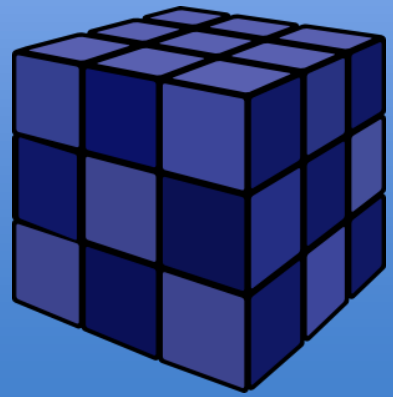
mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com

This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

raj@acloudfan.com

@acloudfan

http://ACloudFan.com



Function Overloading

- Functions with same name but different input parameters

```
function  getOwnerInfo() returns (string name, uint8 age){
  name = ownerName;
  age = ownerAge;
}
```

```
function  getOwnerInfo(uint greaterThan) returns (string name, uint8 age){
  if(ownerAge > greaterThan){
    name = ownerName;
    age = ownerAge;
  }
}
```

Constructor

- Constructor function name = Name of the contract

- Only ONE constructor allowed

➢ To pass parameters to the constructor

```
// Constructor
function  Funcs(string name, uint8 age){
  ownerName = name;
  ownerAge = age;
}
```

2_deploy_contracts.js

```
// Provide the constructor parameters
// string name, uint8 age
deployer.deploy(Funcs,"Nelson",31);
```

Functions Summary

- Functions:
  - May return multiple values
  - Supports return variables
  - Arg names may be skipped
  - Overloading supported
  - Variable scope is function not the block
- Only one constructor allowed
- Tuple is an ordered list of values
  - Declared using keyword **var**
  - Object type need not be specified

Solidity Contracts :

- Complex Datatypes

raj@acloudfan.com

@acloudfan

http://ACloudFan.com

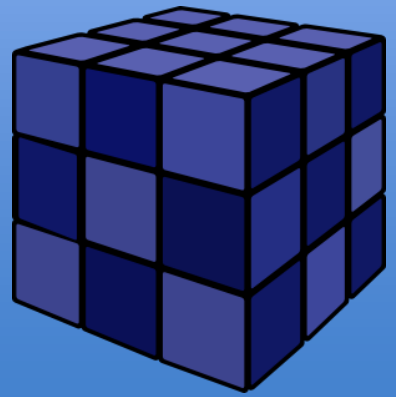
Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

https://www.udemy.com/rest-api/?couponCode=DKRST1099

mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com



This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

Mapping Type

- *Hashtable* like structure
- Allowed only as storage or state variable

```
// storage variable
mapping(address => uint) balances;
```

- Key can be any type except mapping
  - Value type can be Mapping

Mappings Vs Hashtable

Mapping	Hashtable
Value exists for all keys (0x0, 0 ..)	Missing key indicated by undefined/null
keccak256(Key data) hash is stored	Key data is stored
By default NOT iterable	Get all keys and iterate through values
No concept of length	Length can be determined

Enums

- Creates custom types with finite set of values

```
enum  TransferType  {Domestic, Foreign}          // No semicolon
```

- Not part of the ABI definition
- Explicit conversion to/from all integer types

```
uint8    x = TransferType.Domestic;          // Compiler error

uint8    x = uint8(TransferType.Domestic);   // Will work
```

Struct

- Declared using the keyword struct

```
struct BidItem {
  bytes24   name;
  bytes     description;
  address   bidder;
  uint      highBid;
}

// Item instance
BidItem item;
```

- Cannot have member of its own type
- **Can** be contained in arrays and mappings

References

```
function Structs(bytes24 name, bytes description) {
  // constructor
  item = BidItem(name, description,0,0);
}
```

// Copies to storage

- Local reference to the structure instance in storage

```
function setItem(bytes24 name, bytes description){
  BidItem localReference = item;

  localReference.name = name;
  localReference.description = description;
}
```

// Updates Item
// In storage

Memory variables

- Default for struct type local variable is *Storage* type

```
function Structs(bytes24 name, bytes description) {
  // constructor
  BidItem newitem = BidItem(name, description,0,0);

  item = newitem;
}
```

memory

// Compilation error

Mapping, Enum, Struct

- Mapping is like a Hashtable | Associative array
  - Does NOT Provide length function/attribute & a way to iterate
  - Returns 0, empty string for non existent keys
- Enum are not part of the public interface
  - Explicitly converted to/from integers
- Struct are not part of the public interface
  - Cannot contain member of its own type

Solidity Contracts :

- Object orientation

Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

https://www.udemy.com/rest-api/?couponCode=DKRST1099

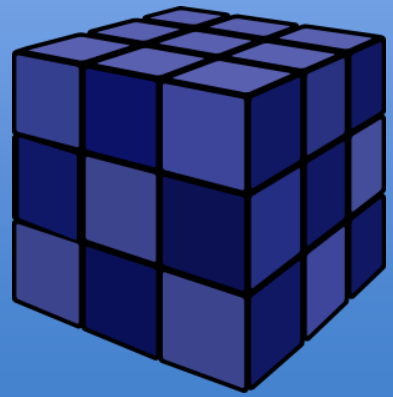
mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com

This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

raj@acloudfan.com

@acloudfan

http://ACloudFan.com



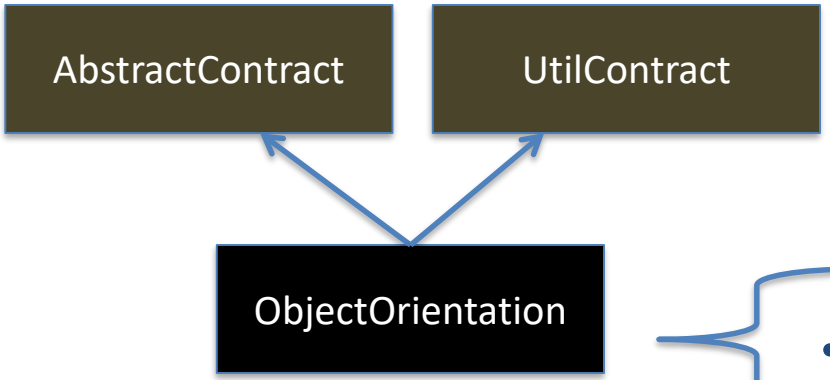
Inheritance >> Abstract Contract

- NO keyword for abstract contracts
  - Functions declared but no body provided

```
contract  AbstractContract  {

  struct agentStruct {
    string  name;
    uint8   commision;
  }

  agentStruct   agent;

  function  calculateAgentCommission(uint16 saleAmount);
}
```

  - Derived class that does not implement functions, itself abstract

Inheritance >> Multiple Inheritance



- Only a single contract created
- By copying
- Name collision lead to errors

- The base & derived class can be in the same file or in multiple files

Inheritance >> Multiple Inheritance

- Inheritance relation created using the keyword **is**

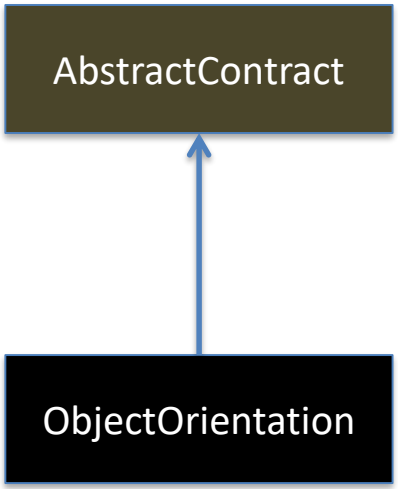
```
pragma solidity ^0.4.4;

import "./AbstractContract.sol";
import "./UtilContract.sol";

// Demonstrates Solidity's support for object orientation
contract ObjectOrientation is AbstractContract, UtilContract {
    // Functions & State variables
}
```

- To be deployable: Implement

```
function  calculateAgentCommission(uint16 saleAmount)
```

Inheritance >> Base Constructor

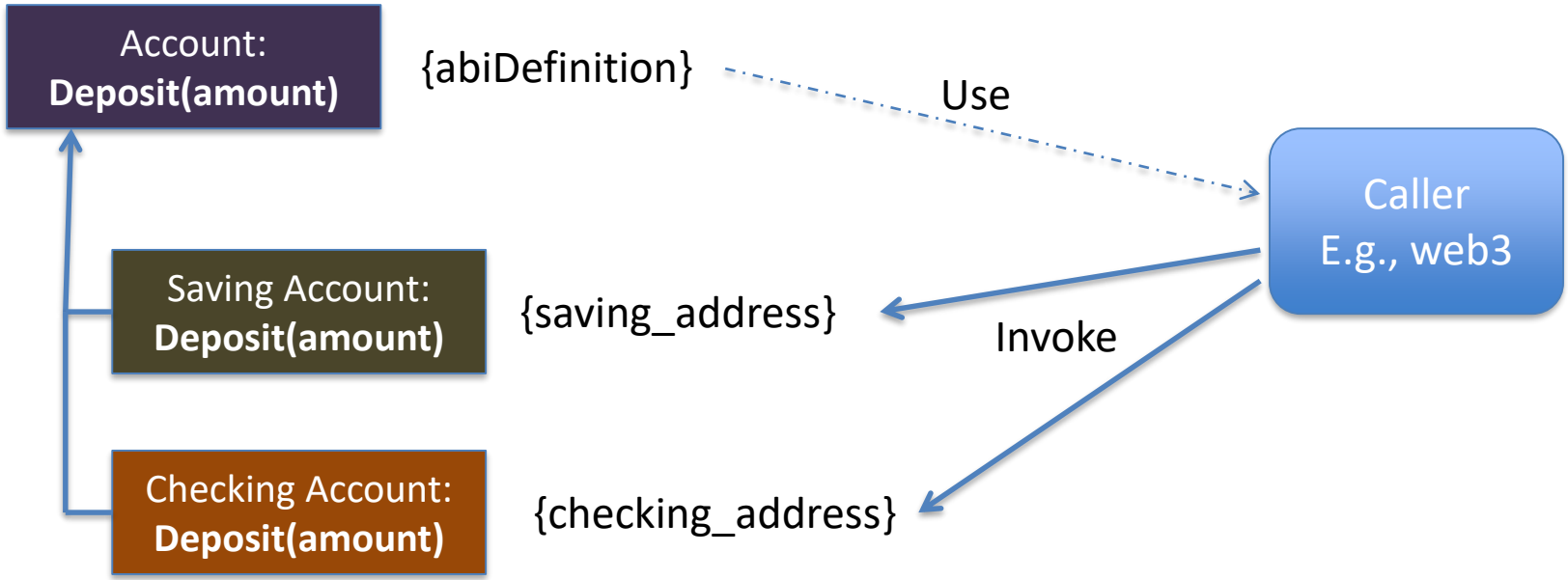
- Derived contract constructor need to provide the arguments for all the base contracts




```
// constructor
function  AbstractContract(string name){
  agent.name = name;
}
```

```
// Constructor inokes the constructor of the base class
function ObjectOrientation(string agentName, uint8 rate) AbstractContract(agentName) {
  commissionRate = rate;
}
```

Polymorphism



Object Orientation

- Solidity supports function overloading
  - No constructor overloading
- Supports multiple inheritance by way of copying
  - Inheritance relation created using the **is** keyword
  - No keyword for abstract contracts
- Supports polymorphism

Solidity Contracts :

- Function visibility

Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

https://www.udemy.com/rest-api/?couponCode=DKRST1099

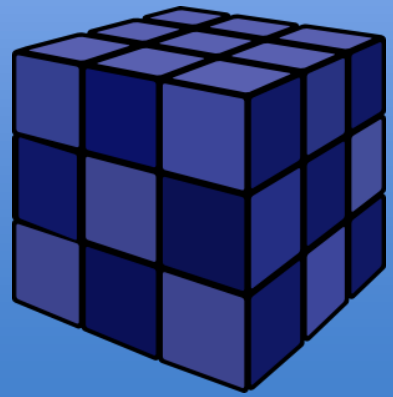
mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com

This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

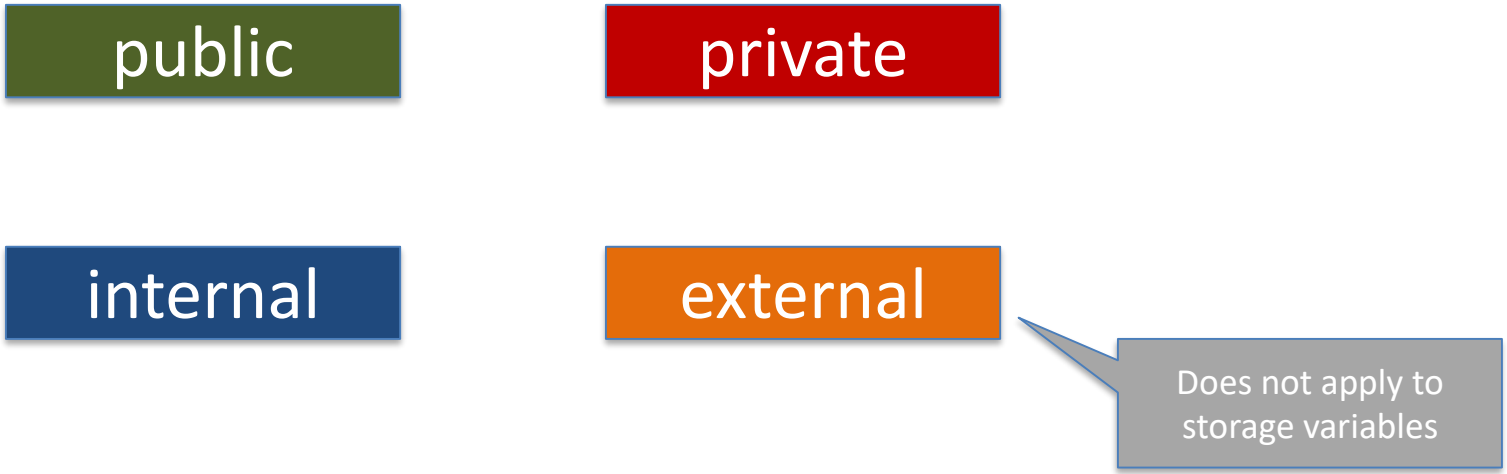
raj@acloudfan.com

@acloudfan

http://ACloudFan.com



Visibility Overview



- Functions can be called from other contracts in a transaction a.k.a. message calls

Public & Private

```
string  public ownerName;
uint8   public ownerAge;

// Constructor
function  Funcs(string name, uint8 age){
  ownerName = name;
  ownerAge = age;
}

// Sets the name
function  setOwnerInfo(string name, uint8 age) public{
  ownerName = name;
  ownerAge = age;
}

function secretFunction() private {
  // Not available outside this contract
}
```

public

Default for functions

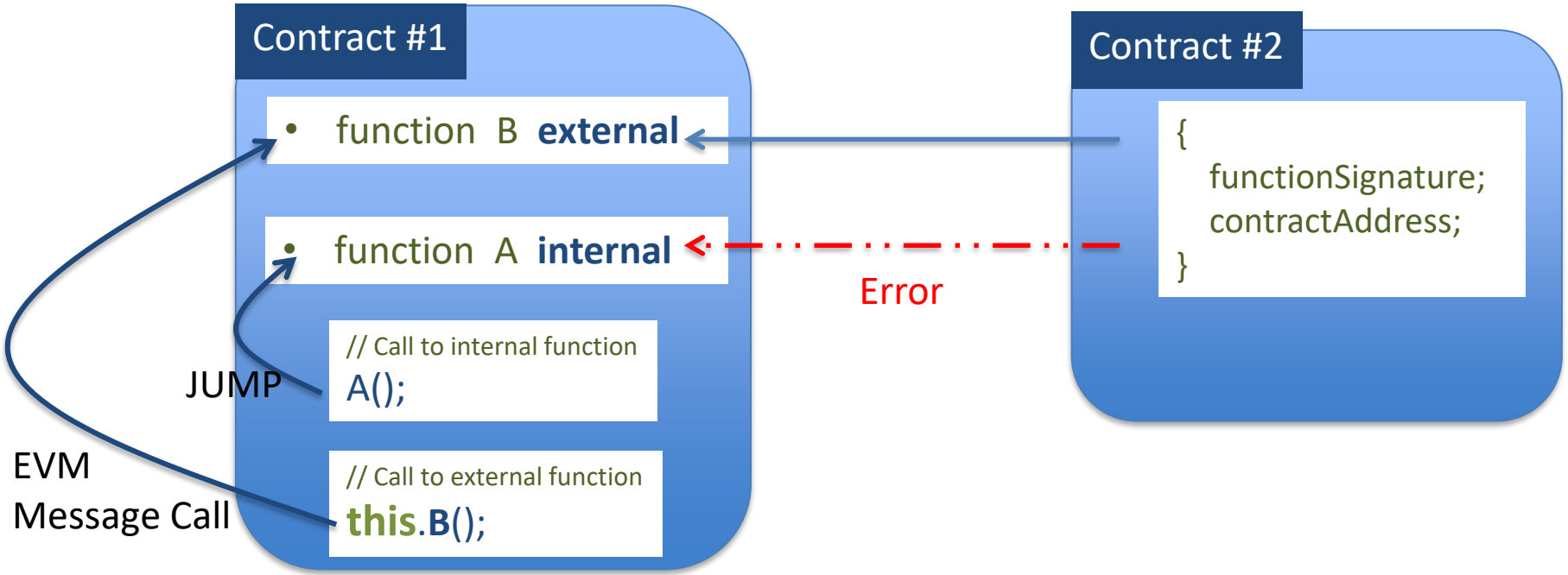
- Part of contract interface
  {abi Definition}
- Automatic *Getter* for state variable
- public Function calls:

  Internal External

private

- Available within the contract only
- Not available even in derived contract

Internal/External Calls



internal

Default for storage variables

- NOT Part of contract interface i.e., *{abi Definition}*
- Function can be invoked from within the contract
- An internal variable/function is available in derived code

external

- NOT applicable to storage variables
- Cannot be invoked from within the contract as a regular function
- Need to use keyword **this.**

Function Type Variables

- Like other variable types
  - Can be a assigned a function
  - Returned from functions
  - Received as parameter

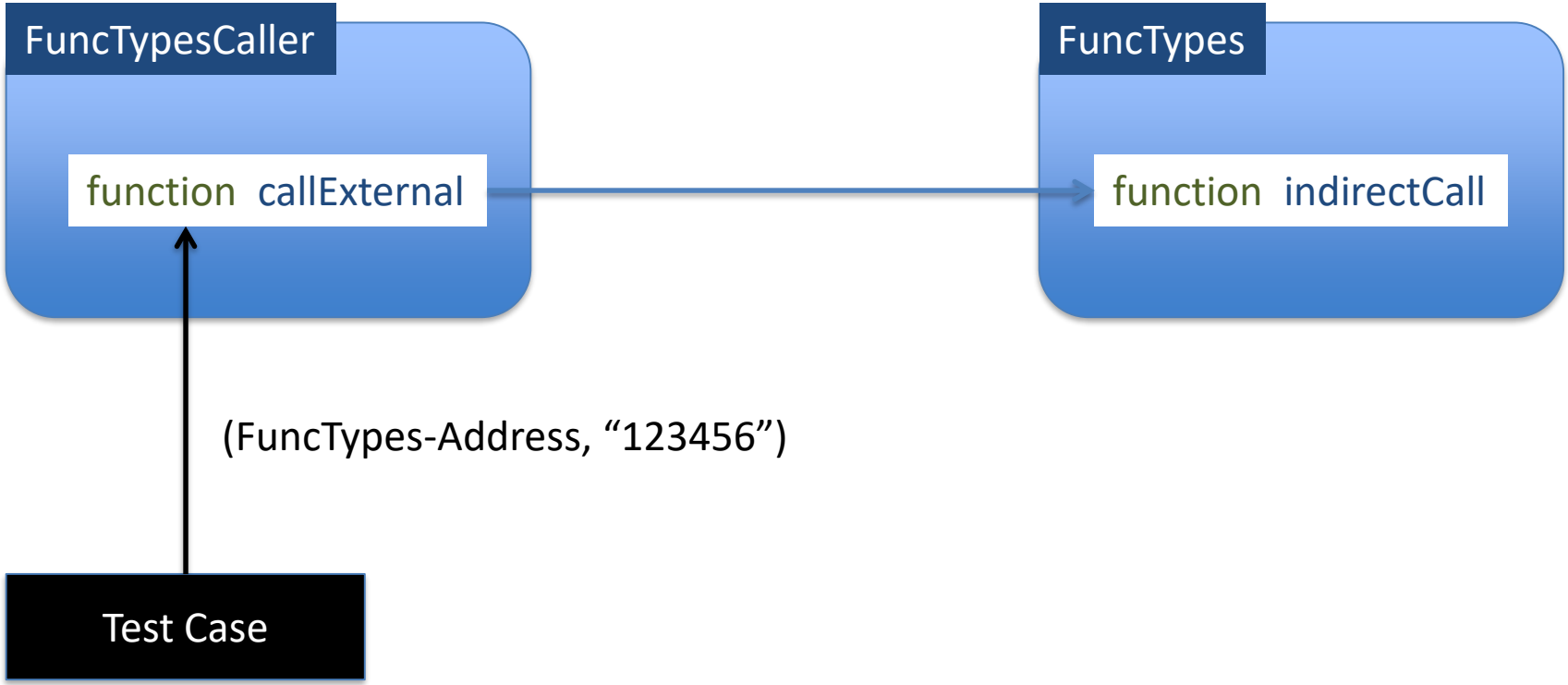
function(<parameter list>) { internal | external }

[constant]

[payable]

returns (<return types>)

Calling external function



Solidity Contracts :

- Datatype conversion
- Global variables
- Global functions

Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

https://www.udemy.com/rest-api/?couponCode=DKRST1099

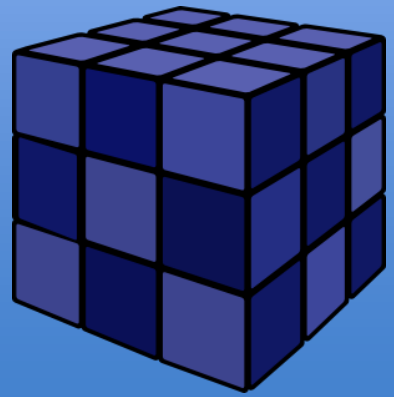
mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com

This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

raj@acloudfan.com

@acloudfan

http://ACloudFan.com



Ether units

- Conversion by suffixing literal with the *Ether* sub-denomination

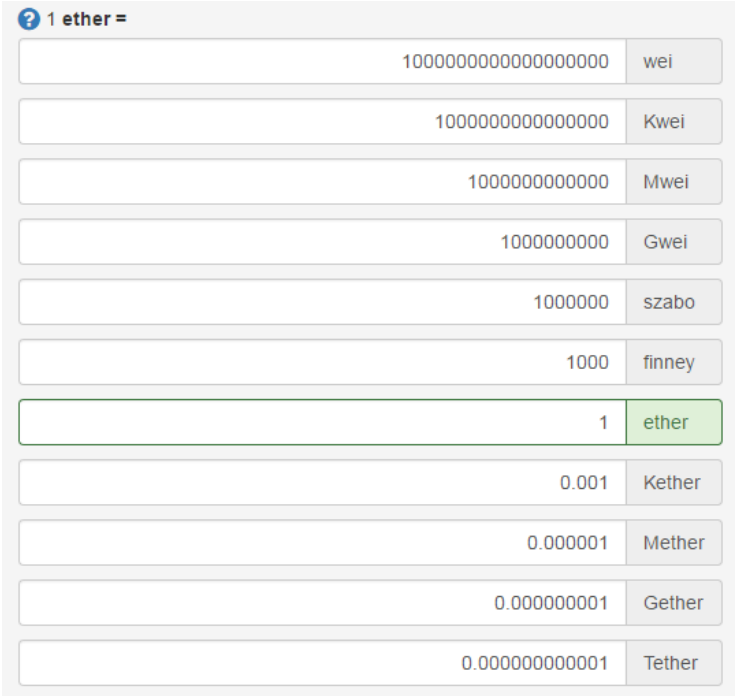
wei (default)

ether

finney

szabo

```
// True
flag = (1 wei == 0.000000000000000001 ether);

// True
uint amount = 10 ether;
flag = (amount*2 == 20000 finney);
```



http://ether.fund/tool/converter

Time

now

- Returns block time in seconds (from 1970)

- Conversion by suffixing literal with the *time* units

default

seconds

minutes

hours

days

weeks

years

block

- Current block information

.number

.coinbase

.timestamp

.difficulty

.gaslimit

.blockhash(uint blkNum) returns (bytes32)

Hashes of most recent 256 blocks
Excludes current

msg

.data

- Call data in *bytes*

.sender

- Caller's *address*

.sig

- Function identifier i.e., first 4 *bytes* of call data

.value

- Number of *wei* sent in the message
- Available only in functions that are *payable*

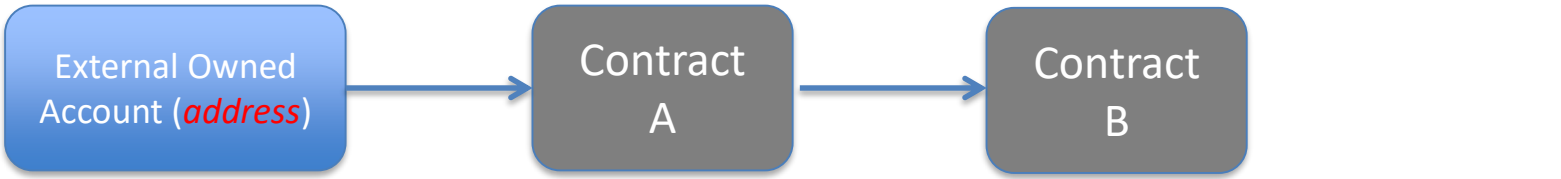
tx

.gasprice

- Gas price for the transaction

.origin

- Address that originated transaction
- DO NOT use this for future compatibility reasons*



msg.sender = address
tx.origin = address

msg.sender = Address of contract A
tx.origin = address

* http://ethereum.stackexchange.com/questions/196/how-do-i-make-my-dapp-serenity-proof?noredirect=1&lq=1

Cryptography Hash Functions

- Special types of hash functions

- **Deterministic** : message hash is always the same for the same message
- **Quick**: to compute the hash
- **Infeasible**: to recreate the message from hash
- **Any change**: to message will change the hash
- **Collision resistant**: different messages will never get the same hash

Crypto Functions

- Takes multiple **bytes** parameters and produces **bytes32**

keccak256(...)

sha3(...)

- Alias to **keccak256(...)**

sha256(...)

- Takes multiple **bytes** parameters and produces **bytes20**

ripemd160(...)

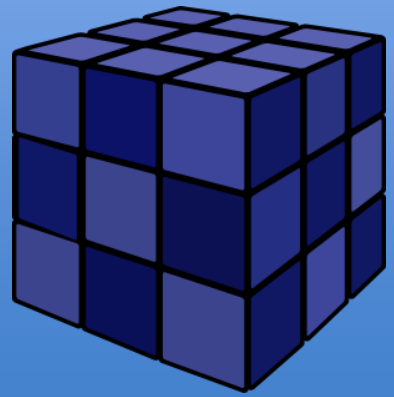
Solidity Contracts :

- Exceptions

raj@acloudfan.com

@acloudfan

http://ACloudFan.com



Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

https://www.udemy.com/rest-api/?couponCode=DKRST1099

mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com

This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

throw;

DEPRECATED
Use require() or revert()

- Aborts the transaction execution
  - All state changes are reverted
  - No ethers are sent out
  - Ethers received in transaction returned
  - Gas is spent i.e., there is a cost
  - Transaction is recorded on the chain; nonce is valid & recorded
  - No catch

revert()

- Behaves like throw;
  - *throw*; uses up all of the gas
  - *revert()* refunds the unused gas

Throw; is deprecated….start using revert()

August 2017

assert(condition)

- Throws if condition is NOT met

```
string  public lastCaller = "not-set";

function  throwBehavior(string name) returns (bool){
  lastCaller = name;

  throw ;

  return true;
}
```

require(condition)

- Like assert it throws an exception
  - assert() & require() are style exception

```
function guess(uint8 num) returns (bool){

  assert(num < 10;)

  for(uint8 i = 0 ; i < numArray.length ; i++){
    if(numArray[i] == num) {
      // Increase the winner count
      winnerCount++;

      require(winnerCount > 0);

      return true;
    }
  }
  loserCount++;
  return false;
}
```

Solidity Contracts :

- Constant Functions | Variables
- Fallback functions
- Payable functions

Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

https://www.udemy.com/rest-api/?couponCode=DKRST1099

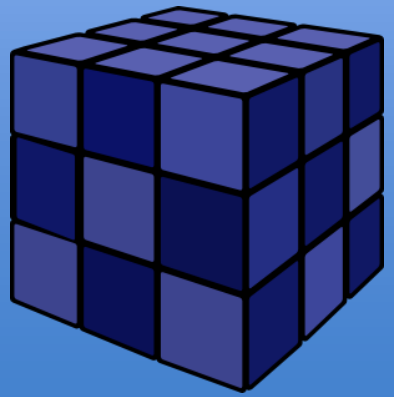
mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com

This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

raj@acloudfan.com

@acloudfan

http://ACloudFan.com



Constant Variable

- **Constant** variable MUST be initialized at compile time
  - No storage allocated by compiler; copies the literal for references
  - Built in functions may be called e.g., *keccak256(..)*
  - Unlike JAVA initialization in constructor NOT allowed

Compiler error >>>

```
3   contract ConstantsPayable {
4
5     // MUST initialize storage here
6     uint    public constant creationTime;
7
8     function   ConstantsPayable() {
9       // Storage constant initialization NOT allowed
10      // in constructor
11      creationTime = now;
12    }
13
```

Constant Variable

- Not allowed for function variables
- Allowed for value types & string

Compiler error >>>

```
struct agentStruct {
  string  name;
  uint    commission;
}

// Not allowed
agentStruct  constant  agent;

uint constant  interest = 10;
```

Constant Functions

- **Constant** functions Promises not to change state of the contract

```
contract Pricer {

  // Current price
  uint  price;

  function setPrice(uint newPrice){
    price = newPrice;
  }

  function  evaluatePricing(uint newPrice) constant {
    // Some logic
    price = newPrice;
  }
}
```

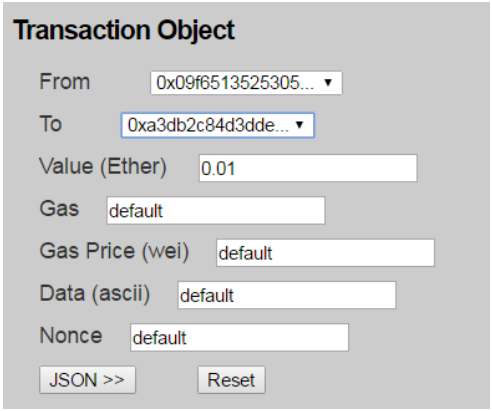
<<< Not enforced by compiler

Fallback Function

- An un-named function in the contract
  - Invoked without the data i.e., the function signature
  - Restrictions:
    - No arguments
    - Cannot return anything
    - Maximum gas spend = 2300 gas

Receiving Ethers

- A **Contract** like EOA can receive Ethers
- A **Function** invocation can receive Ethers



Receiving Ethers

Contract

- Contract can receive ethers by way of *payable* fallback function
- Invoked when ethers are received (msg.value) without data
  - Exceeding the gas if tries to update storage or call a function

    Throws exception

    Sends back the ethers

    Best Practice: Just log an event in the fallback function

Receiving Ethers

Function

- A function MUST be marked *payable* to receive ethers
  - Amount sent available in msg.value

```
function receiveEthers(string name)      {
  lastSender = msg.sender;
  lastReceived = msg.value;
  lastCaller = name;
}
```

payable

<<< Compiler enforced

  - Unlike fallback function, NO restriction on gas usage
  - Ethers held in the contract

Solidity Contracts :

- Function modifiers

Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

https://www.udemy.com/rest-api/?couponCode=DKRST1099

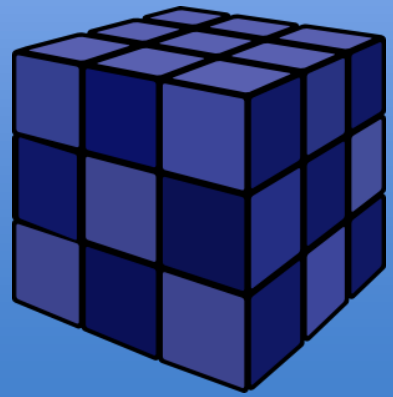
mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com

This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

raj@acloudfan.com

@acloudfan

http://ACloudFan.com



Modifiers?

- Changes the behavior of a function

Defines a modifier

Condition executed before function body

Function body

Anyone can execute
Modifier not applied

Only owner can execute

```
contract Modifiers {
  // state variable
  address   owner;
  // modifier
  modifier ownerOnly {
    if(owner == msg.sender){
      _ ;
    } else {
      throw;
    }
  }

  // Anyone can call this
  function checkOwnership() returns (bool){
    // ...
    return true;
  }
  // Othe than owner - will throw an exception
  function transferOwnership()   ownerOnly   {
    // ...
  }
}
```

Return Vs Throw

- A revert() (earlier throw) in modifier halts the execution
- A return in modifier returns from modifier body

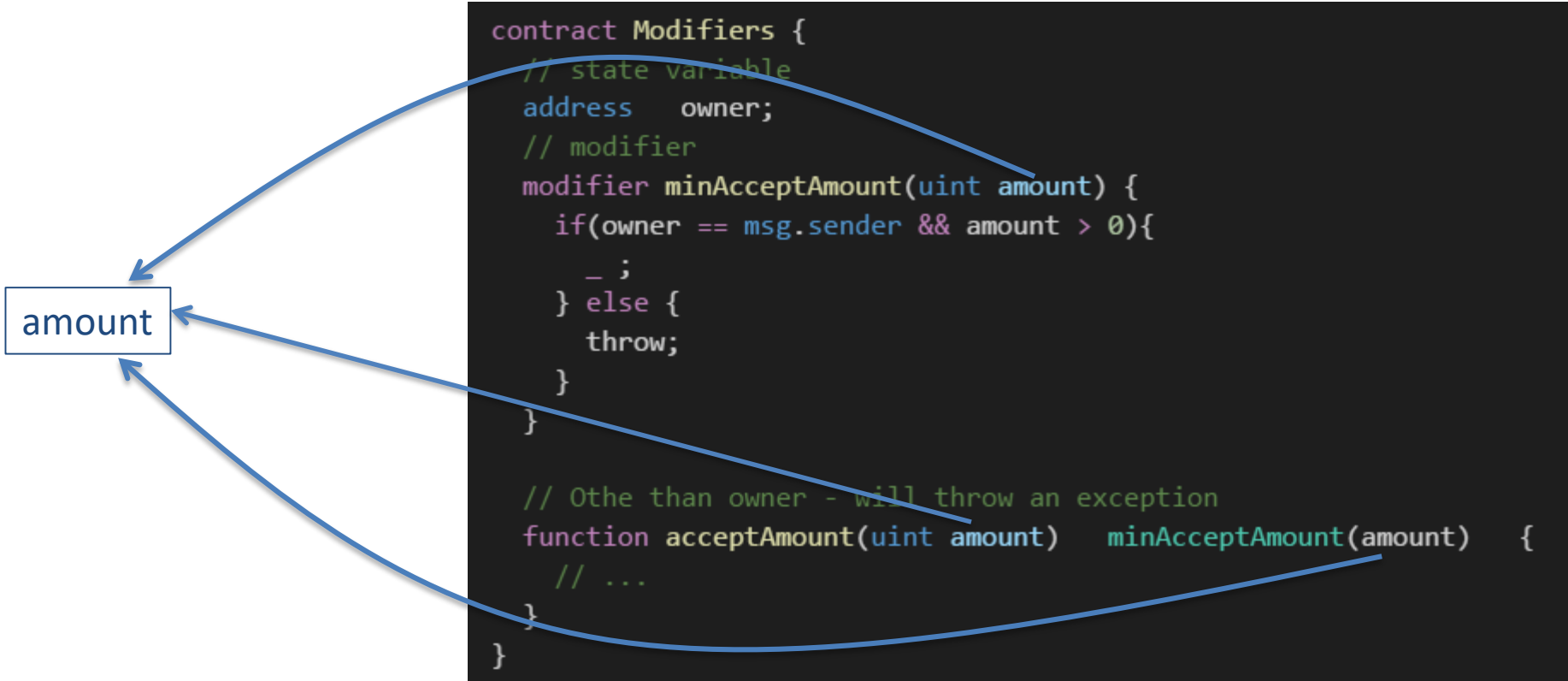
```
modifier  ownerOnly {
  if(msg.sender == owner){
    _;
    return;
  } else {
    throw;
  }
}
```

```
modifier  ownerOnly {

  return; // Function body n

  if(msg.sender == owner){
    _;
  } else {
    throw;
  }
}
```

- Function not executed

Arguments

- Modifiers can take arguments



Local Variables

- Local variables from within modifiers NOT available in functions

```
modifier minAcceptAmount(uint amount) {

  uint someVar = 89;

  if(owner == msg.sender && amount > 0){
    _ ;
  } else {
    throw;
  }
}
```

// Compilation Error

```
function acceptAmount(uint amount)    minAcceptAmount(amount)   {
  // ...
  amount += someVar;
}
```

Applying Modifiers

- Multiple modifiers may be applied to functions
  - Order is important

```
// (a) Check if owner called (b) Check if amount greater than min
function acceptAmount(uint amount)  owned  minAcceptAmount(amount)   {
  // ...
}
```

- Inheritable & may be overridden by child contract

Solidity Contracts :

- Events
- Logs

raj@acloudfan.com

@acloudfan

http://ACloudFan.com

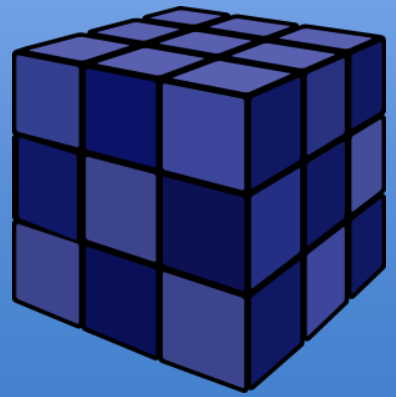
Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

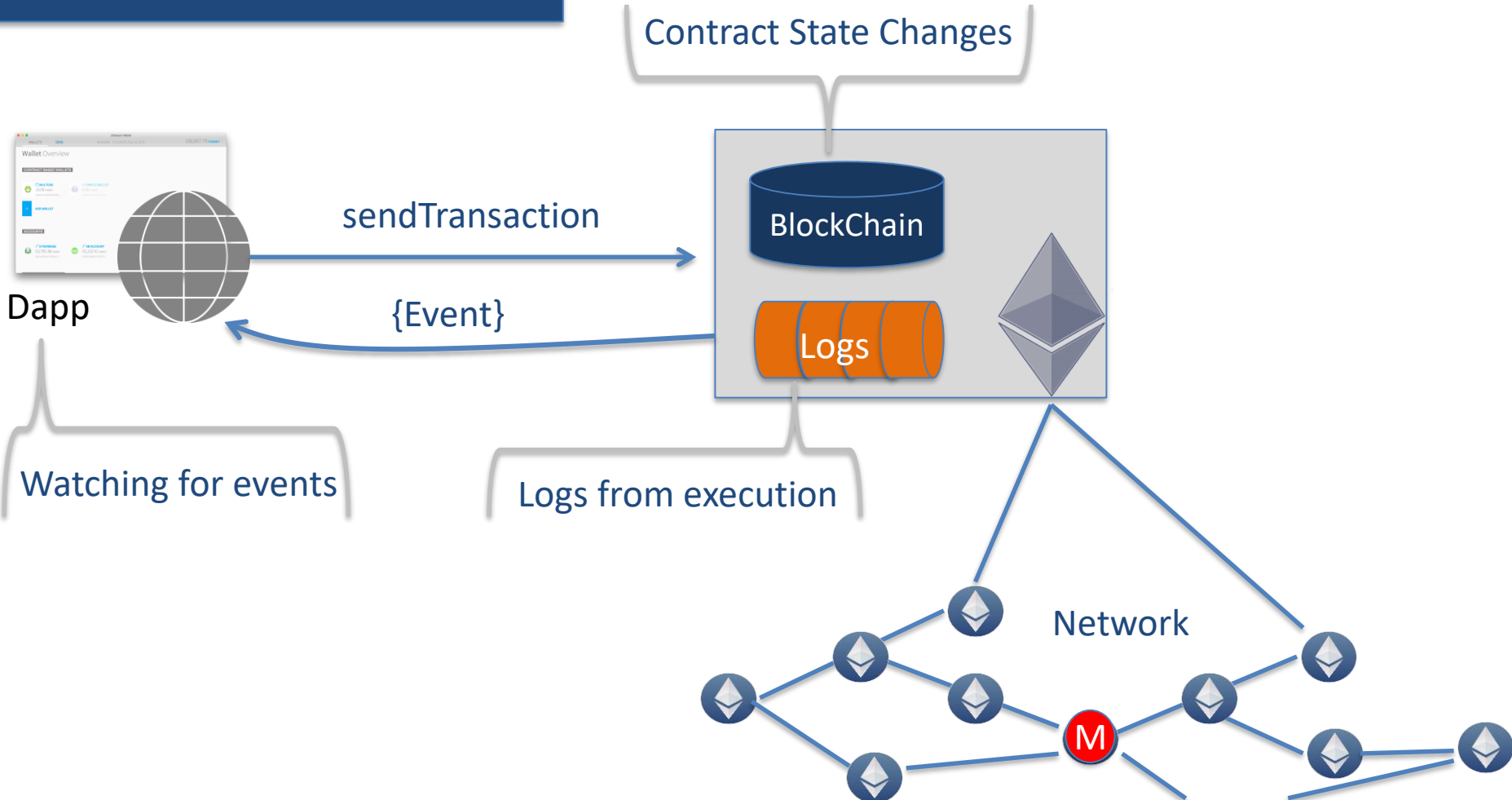
https://www.udemy.com/rest-api/?couponCode=DKRST1099

mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com



This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

RECAP: Logs & Events



RECAP: Events

- Events are part of abi definition
- Event arguments are stored in the logs
- Logs can be read using topic filters
  - Event arguments marked as indexed can be used as criteria/filter
  - A maximum of 3 indexed arguments allowed

Coding

- Declared like a function without body

```
// Whenever a high bid is received
event NewHighBid(address indexed who, string name, uint howmuch);
// High bid preceded by this event
event BidFailed(address indexed who, string name, uint howmuch);
```

Invokes bid(…)

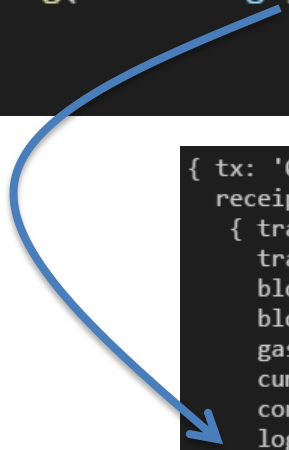
Receives events

- Invoked like functions

```
function  bid(string name) payable timed {
  // Bids allowed in increments of 10 wei
  if(msg.value > (highBidder.bid + 10)){
    //...
    // Received a high bid - emit event
    NewHighBid(msg.sender, name, msg.value);
  } else {
    // Received bid less than high bid emit event
    BidFailed(msg.sender, name, msg.value);
  }
```

Testing: Log Access

```
function  dumpEvents(result){
  for(var i=0; i<result.logs.length;i++){
        console.log(result.logs[i].event,'>>', result.logs[i].args)
  }
}
```



```
{ tx: '0x2ca02447dec41d90ed062a292fcb2ceda36e8ab3e13beb56e5516a0693222634',
  receipt:
   { transactionHash: '0x2ca02447dec41d90ed062a292fcb2ceda36e8ab3e13beb56e5516a0693222634',
     transactionIndex: 0,
     blockHash: '0x22401247fc9cd3dcbc82eefd7bbdc8b087c805381dd8298d7d346ccce17919d6',
     blockNumber: 1693,
     gasUsed: 24971,
     cumulativeGasUsed: 24971,
     contractAddress: null,
     logs: [ [Object] ] },
  logs:
   [ { logIndex: 0,
       transactionIndex: 0,
       transactionHash: '0x2ca02447dec41d90ed062a292fcb2ceda36e8ab3e13beb56e5516a0693222634',
       blockHash: '0x22401247fc9cd3dcbc82eefd7bbdc8b087c805381dd8298d7d346ccce17919d6',
       blockNumber: 1693,
       address: '0xb56e5f52060991a030cf2739d4bbc5582ac983d5',
       type: 'mined',
       event: 'BidFailed',
       args: [Object] } ] }
```