Blockchain Applications and Smart Contracts

1. **Introduction to Blockchains and Smart Contracts**
    - **Explain the history of blockchain technology:**

Blockchain: Immutable, unforgeable ledger of assets and transactions

Institutions lower uncertainty allowing two entities to transact without trust, e.g.

- Government issued ID
- Banks and escrows
- Ebay merchant and user reviews

However, these are fragmented with different databases / infrastructure and limited visibility into transactions. Difficult recourse if things go wrong.

Blockchain does not require institutions, instead it is a shared reality across non-trusting entities, and solves some problems of centralized systems:

- Controlled, portable identity
- Transparency
- Public registry, hard if not impossible to tamper with

Blockchain Applications and Smart Contracts

I've been working on a new electronic cash system that's fully peer-to-peer, with no trusted third party.

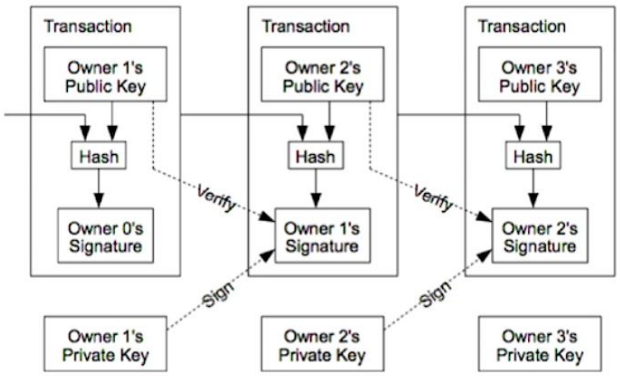
Transactions in a Blockchain

Each transaction digitally signed: More than just electronic signature, a mathematical way to demonstrate authenticity of digital content, e.g. using a public and private key (cryptography)

Electronic coin: Chain of digital signatures

- Hash of previous transaction and public key of next owner
- Anyone can verify the chain of ownership

Order of transactions is determined by a collection of servers, or **nodes.**
"Mining" is an ordered selection mechanism.

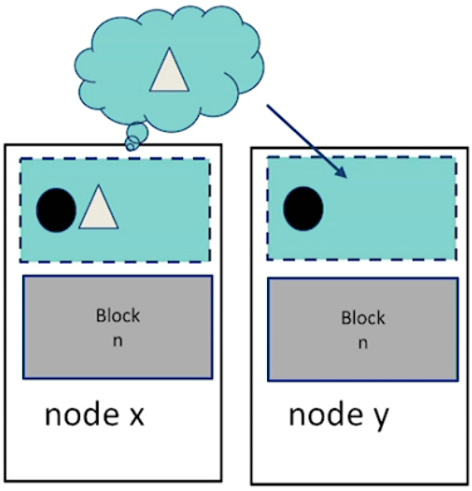


From Satoshi Nakamoto's original bitcoin whitepaper Oct, 2008

- **Understand the consequences of double-spending avoidance**

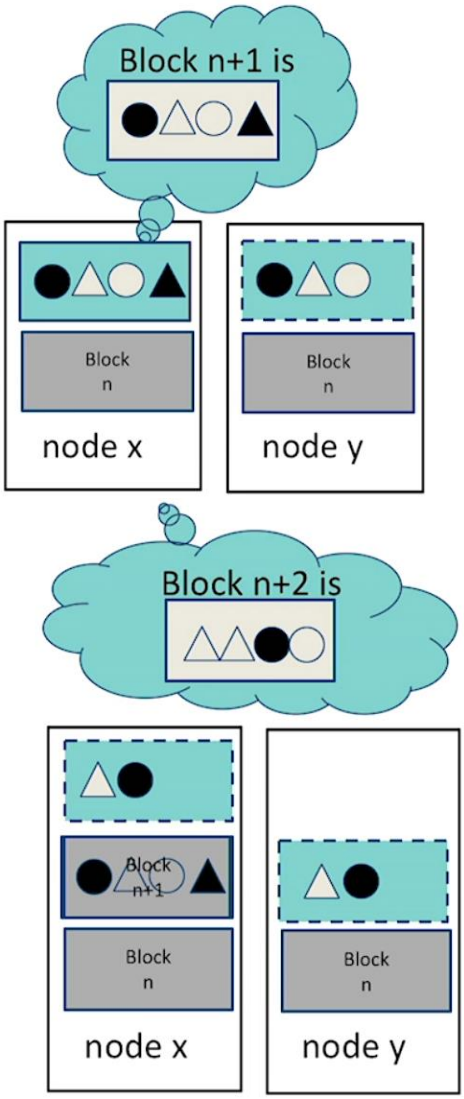
Double-spending avoidance (without a central authority) motivates the need for a **blockchain**:

1. **Publicly announced spending transactions**



Blockchain Applications and Smart Contracts

2. Each node keeps track of a chain of blocks of transactions (mining)
   a. Competes for completed block of transactions (proof of work)
   b. Broadcasts each completed block to all other nodes
   c. Accepts broadcast block only if all transactions are not already spent
   d. Starts building the next block based on this



3. If any discrepancies exist between nodes, the longest chain wins, and invalid blocks are reverted, abandoning the later duplicate transactions

Overall, the underlying mechanism does not need to be understood, but it provides a motivation to learn many aspects of the technology.

A conventional blockchain is:

Public…by default: A decentralized, peer-to-peer ledger without trust. But private blockchains can be set up in a similar manner.

Immutable…over time: Consensus is built through mining. Need 6 confirmations to be 99.9% sure of the transaction, so this takes an hour for Bitcoin and 1.5 minutes for Ethereum.

Blockchain Applications and Smart Contracts

- **Appreciate the objective of different blockchains:**

Coin	Ethereum Classic (ETC)
Description	Added Turing-complete smart contracts
Details	Exploited by hackers, but supporters still keep it alive.

Coin	Ethereum (ETH)
Description	Fork of Ethereum classic to remove exploitation by hackers, uses Proof-of-Work but migrating to Proof-of-Stake (maybe in 2018?)
Details	"Digital dollar". Unlimited Ethereum. 15 seconds for each block (size dictated by gas, approx 25 tx/sec), many altcoins based on Ethereum

Coin	Litecoin (LTC)
Description	Faster transactions, built on BTC, developers test ideas here since it does not alter BTC.
Details	"Digital silver". 2.5 minutes for 1MB block (25 tx/sec), more will move here if any BTC turbulence.

Blockchain Applications and Smart Contracts

Coin	Bitcoin Cash (BCH)
Description	Longer blocks allow more transactions per second.
Details	10 minutes for 8MB block size (48 tx/sec) and more room for extensions like Omni (altcoin on BTC).

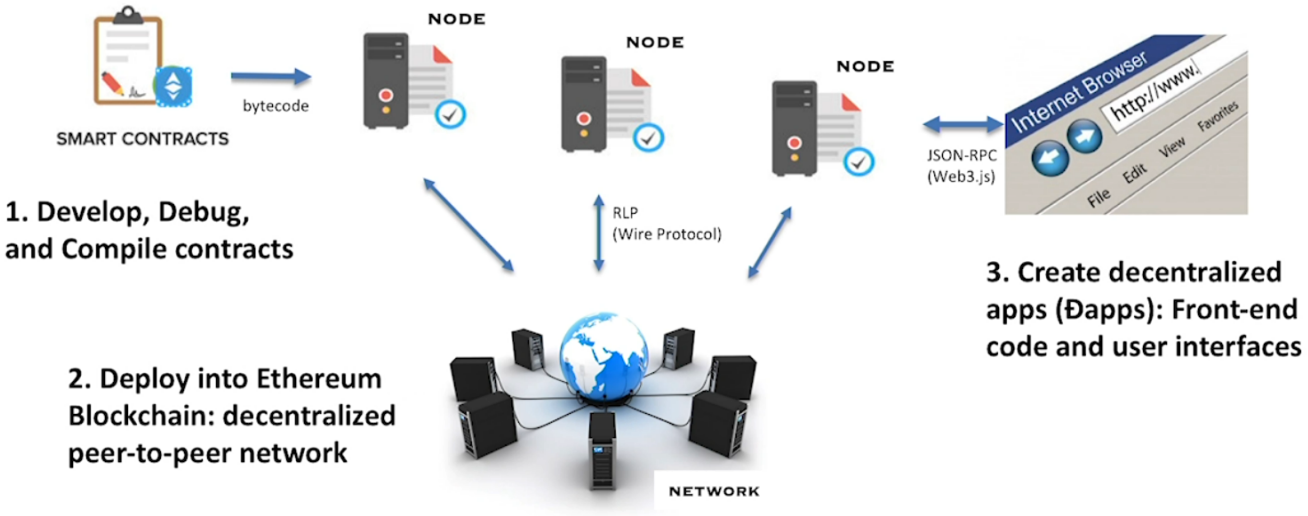
Coin	Ripple (XRP)
Description	Real-time gross settlement system backed by banks.
Details	Not a blockchain: Truly immutable once ledger closes, 3 second ledger update, 10,000 tx/sec.

Coin	Nem (XEM)
Description	Private/public blockchain. Proof-of-importance. Take what bitcoin had and apply to all technological infrastructure. Smart assets.
Details	Recognized by some Japanese banks, very scalable and low-cost, 1 min/block

Blockchain Applications and Smart Contracts

- **Add smart contracts to blockchains**

Ethereum: How to Run a Decentralized Computer?



1. Develop, Debug, and Compile contracts

2. Deploy into Ethereum Blockchain: decentralized peer-to-peer network

3. Create decentralized apps (Đapps): Front-end code and user interfaces

- **Determine relevant smart contract use-cases**

Smart Contract Use-Cases

Not good:

- Complex programs like machine learning, graphical output, etc. Only put business logic and data crucial for consensus
- Interacts with external service such as the Weather station: every node contacts at different times. Instead use Oracle to enter data into the blockchain
- Relies on confidential information
- Relies on low latency

Good:

- Tokenize all valuable assets, and trade these tokens for other tokens or fiat (refinance house without interest)
- Data store representing something which is useful to either other contracts or to the outside world (contract that records membership in an organization)
- Forward incoming messages to some desired destination only if certain conditions are met (withdrawal limit that is over-rideable via some more complicated access procedure)
- Manage an ongoing contract or relationship between multiple users (escrow with some set of mediators)
- Open contract for any other party to engage with at any time (pay prize to first valid solution to some problem)

Blockchain Applications and Smart Contracts

Some Interesting Ethereum Projects

Augur, Gnosis: Decentralized prediction market

BoardRoom: Blockchain governance platform

Colony: Platform for autonomous blockchain organizations

BlockApps: Tools to build decentralized apps

Airlock: Keyless access protocol for smart property

Provenance: Gather and share information & stories behind products

Slock.it: Smart locking and billing for the sharing economy

DigixGlobal: Technology to own gold assets

WeiFund: Crowdfunding platform

Maker: Autonomous bank & market maker

HitFin: OTC derivatives settlement

Solidity: Online compiler

Etherparty: Smart contract deployment tools

DappLib: library of math functions

Blockchain Applications and Smart Contracts

2. Ethereum: A Smart Contract Blockchain

- **Introduce Ethereum as a blockchain for smart contracts**

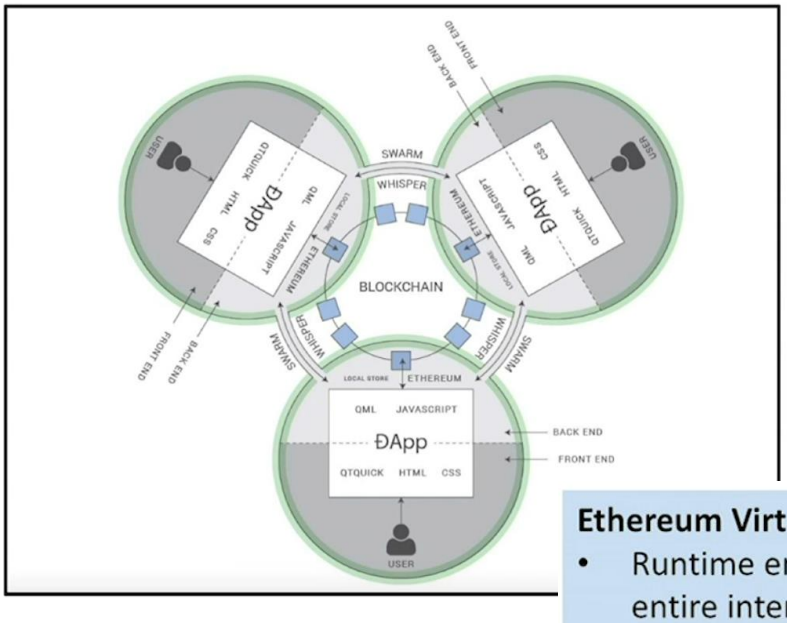
Called the "new web" or **web3.0** with no web servers or middleman.

A smart contract platform for decentralized apps **(Đapps)**.

Large decentralized computer that knows each user. No logins needed!

- **Ethereum** = computer
- **IPFS** = storage (replaces Swarm)

Original Ethereum Concept



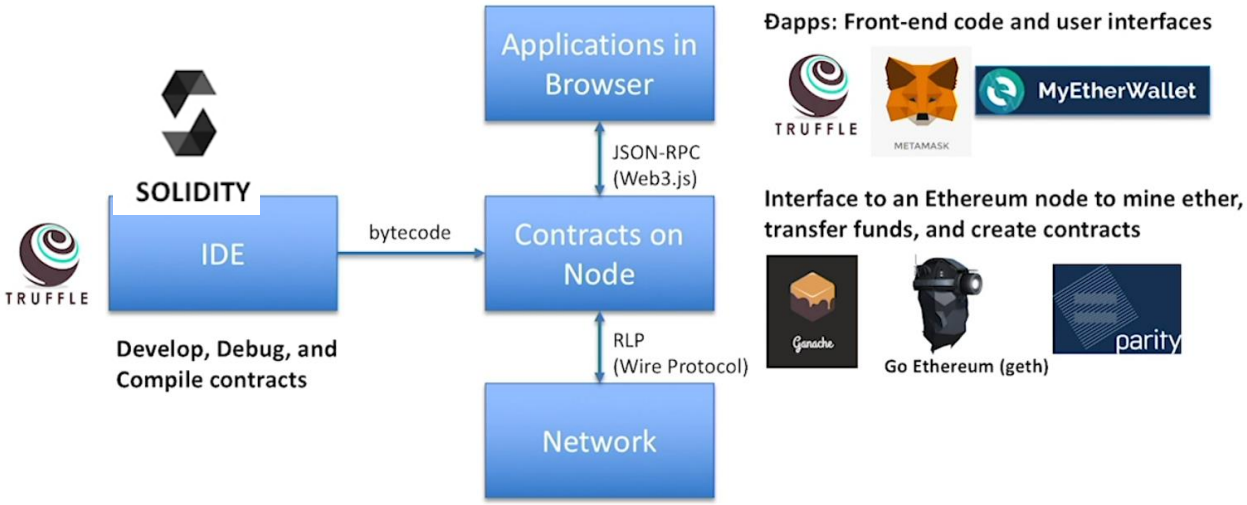
Ethereum Virtual Machine (EVM):

- Runtime environment handling the entire internal state and computation. It is sandboxed and completely isolated.
- Every operation is executed on every node in the network.
- Not a register machine, but a stack machine. Stack maximum size of 1024 elements of 256-bits each.

Blockchain Applications and Smart Contracts

Ethereum Hard Fork	Time
Ice Age	September 2015
Homestead (1st production rel)	March 2016
Tangerine Whistle (DAO split)	October 2016
Spurious Dragon	November 2016
Byzantium	October 2017
Serenity (POS)	TBD

- **Use Truffle as a smart contract development tool:**



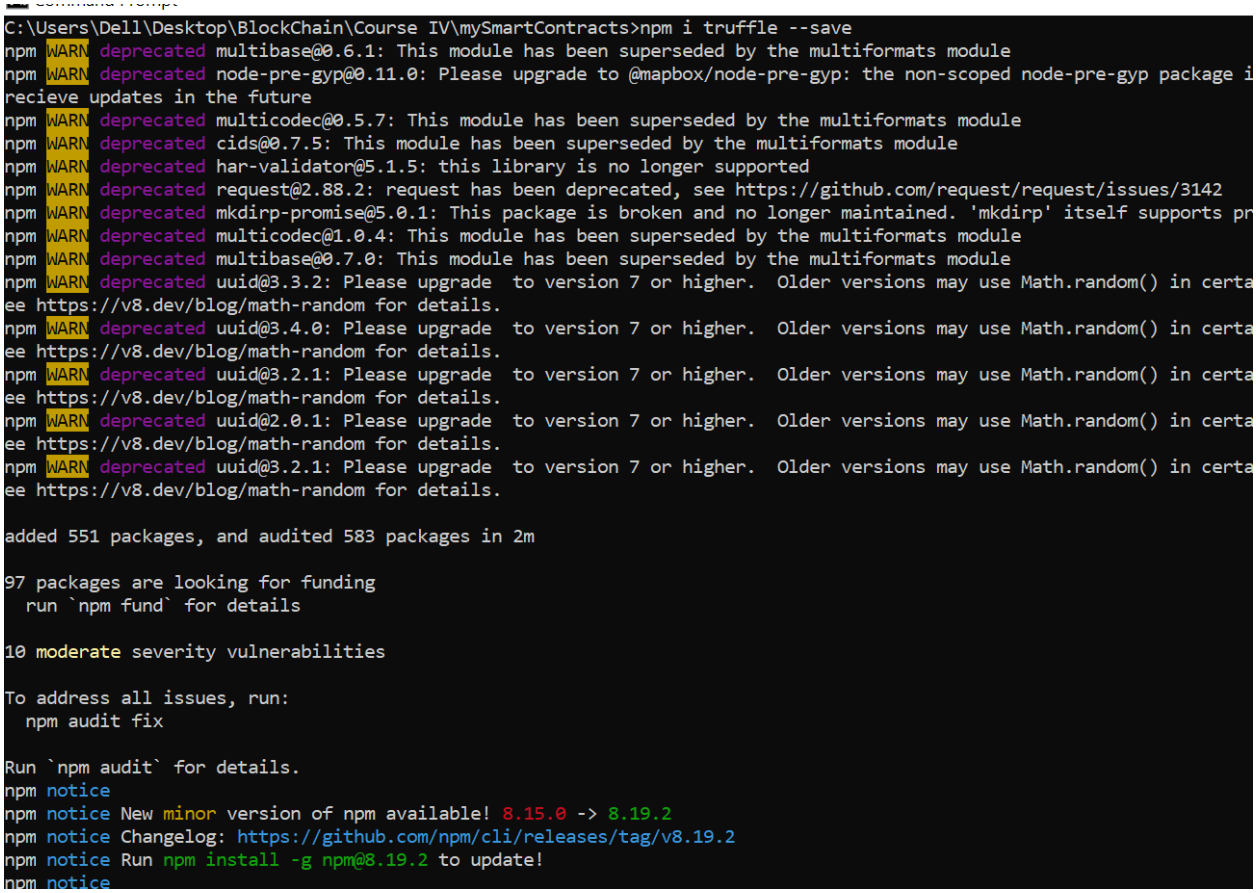
- **Personal blockchain: Ganache**
  - Access through standalone client (e.g. ganache-cli previously testrpc)
  - Instantly processes transactions for quick development

- **Public blockchain: geth, parity**
  - Access through Ethereum client
  - Different networks to access (see right)

Ethereum Networks

- 0: Olympic, Ethereum public pre-release testnet
- 1: Frontier, Homestead, Metropolis, the Ethereum public main network
- 1: Classic, the (un)forked public Ethereum Classic main network, *chain ID 61*
- 1: Expanse, an alternative Ethereum implementation, *chain ID 2*
- 2: Morden, the public Ethereum testnet, now Ethereum Classic testnet
- 3: Ropsten, the public cross-client Ethereum testnet
- 4: Rinkeby, the public Geth Ethereum testnet
- 42: Kovan, the public Parity Ethereum testnet
- 77: Sokol, the public POA testnet
- 99: POA, the public Proof of Authority Ethereum network

Blockchain Applications and Smart Contracts



- **Explain Ethereum addresses and transactions**

Two types of accounts share the same address space:

- **Externally owned accounts** (EOAs) controlled by public-private key pairs (i.e. humans), have balance and storage
- **Contract accounts**: have code storage, controlled by that code

How an EOA address is created:

- **Private key**: 64 hex chars randomly generated
- **Public key**: 128 hex chars generated from private key using the Elliptic Curve Digital Signature Algorithm
- **Public address**: last 40 chars of 64 char hash of public key

Blockchain Applications and Smart Contracts

How an EOA address is created:

- **Private key**: 64 hex chars randomly generated
- **Public key**: 128 hex chars generated from private key using the Elliptic Curve Digital Signature Algorithm
- **Public address**: last 40 chars of 64 char hash of public key

```
var address = '0x' + util.bufferToHex(util.sha3(publicKey)).slice(26);
web3._extend.utils.isAddress(address)
```

How a Contract account address is created:

- Address generated from creator address and **nonce** (number of transactions sent from that address)

Message from one account to another

- Can include binary data (payload) and ether
- Can go between EOA and contract accounts
  - EOA -> EOA: transfer ether
  - EOA -> 0: create contract from payload
  - EOA -> contract: run code with data from payload
  - contract -> EOA: transfer ether
  - contract -> contract: run code with data from payload

To a contract account: smart contract activates and can

- Return data (like a function call)
- Call other contracts
- Only complete when all pieces complete (cannot move on until that happens)

- Message can contain: source, target, data payload, ether, gas, and return data
- **Special type of message**: A **delegatecall** is executed in the context of the calling function (like a library or subroutine)
- **Special operation**: **selfdestruct** call removes contract from the blockchain

Blockchain Applications and Smart Contracts

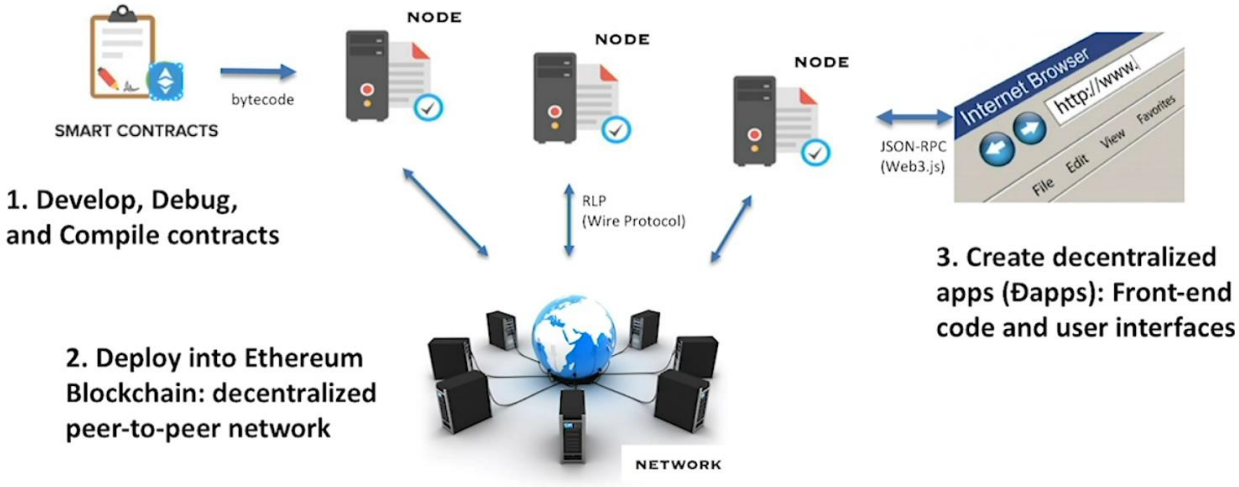
Storage:

- **Permanent**
- Persistent memory inside a contract
- Costly to read or modify
- Key/value store (each 32B) in sparse format

Memory:

- **Temporary**
- Fresh every contract transaction
- Expanded by 32B chunks as it is accessed
- Cost quadratically as access grows

- **Understand the relationship between Ether and Gas**



1. Develop, Debug, and Compile contracts

2. Deploy into Ethereum Blockchain: decentralized peer-to-peer network

3. Create decentralized apps (Đapps): Front-end code and user interfaces

How to avoid malicious or poorly written code taking whole system down?

Blockchain Applications and Smart Contracts

Operation Name	Gas Cost	Remark
STEP	1	Default amount per execution cycle
STOP	0	Free
SUICIDE	5000	Permanently kill contract
SLOAD	200	Load word from permanent storage
SSTORE	5000	Put word into permanent storage
MLOAD	3	Load word from memory
MSTORE	3	Put word into memory
BALANCE	400	Balance of given account
CREATE	53000	Changed in homestead from 21000
LOG0	375+8*b	Log b bytes of data
SHA3	30+6*w	Encode w words of input

Every account has a balance in **Ether** (1 Ether = 10^{18} Wei)
Every transaction requires **Gas** to execute

maxTransactionCost = gasLimit * gasPrice

gasLimit: Amount of gas purchased by sender, default value 4712388 (at least need 21000 for the transaction fee)

gasPrice: Specified by transactor, a criteria for selection by miner, typically 0.5 GWei and max 50 GWei (1 GWei = 10^9 Wei)

Transaction execution depletes gas according to specific rules:

- If gas runs out, then all processing reverts, but account still charged
- If any gas left, then refunded to the sender's account

Blockchain Applications and Smart Contracts

- **Accurate:** Executes as intended
- **Efficient:** Minimize cost upon deployment and when invoked
- **Secure:** Immune to attacks

Blockchain Applications and Smart Contracts

3. Solidity: A Contract-Oriented Language

- **Explain the structure of a Solidity smart contract**

Solidity: Probably the first "Contract-Oriented Language" and reworks the well-established Object-Oriented Languages (similar to JavaScript or C++)

Some Similarities to Object-Oriented Languages:

- Contains persistent data in state variables and functions that can modify these variables
- Includes common object-orientated concepts such as inheritance, abstract contracts, and interfaces
- Calling a function on a different contract (instance) will perform an EVM function call and thus switch the context such that the calling contract state variables are inaccessible from the called contract

Solidity contracts run solely on a blockchain and have important differences:

- Access blockchain data (such as **this.balance** and **block.number**)
- Execution depends on mining latency, use events to see results, utilizing logging
- Costs money; must be robust against reentrant issues
- Immutable; be sure no bugs or erratic behavior that uses gas. Utilize error checking and importance of unit testing

Blockchain Applications and Smart Contracts

```
pragma solidity ^0.4.4;

/// @title Simple example of setting and getting storage data
/// @author Dev Name
/// @dev Storage is costly. Only use for critical data.
contract GetAndSet {
   uint16[3] storedData;

   function setStoredData(uint8 n, uint16 x) {
      storedData[n] = x;
   }

   function getStoredData(uint8 n) public view returns (uint16) {
      return storedData[n];
   }
}
```

Note:

- Pragma
- Comments and Natspec
- Contract like a class
- State variables located in storage, function parameters located in memory
- Functions are executable units in a contract
- New keywords such as “view”

- **Use Solidity declarations**

Types:

- **bool:** Boolean can be either true or false
- **int / uint:** Signed and unsigned integers from 8 to 256 bits (int and uint are aliases for int256 and uint256)
- **fixed / ufixed:** Signed and unsigned fixed point numbers of various sizes (fixedMxN has M bits and N decimal points) (fixed is alias for fixed128x19)
- **address:** 160-bit non-arithmetic value. Has member functions “balance”, “transfer”, (“send”, “call”, “delegatecall”)
- **bytesN:** N bytes (byte is alias for bytes1).
- **bytes:** Is a dynamically-sized array (more costly)
- **string:** Dynamically-sized UTF-8 encoded string

Blockchain Applications and Smart Contracts

Modifiers:

- **constant:** Deprecated for functions, means variable is constant at compile time
- **pure:** Does not read or change the storage state
- **view:** Like "constant" only reads the storage state
- **payable:** Can receive Ether through send or transfer, needs fallback function

Visibility:

- **external:** Can be accessed from other contracts, efficient for large arrays of data
- **internal:** Can only be accessed from inside the contract, default
- **public:** Can be accessed outside the contract, default
- **private:** Not in derived contracts, but information still visible to all external observers

Declaring state variable as public basically adds a getter. For example:

```
uint256 public value1;
```

Makes an implicit function:

```
function value1() returns (uint256) {
    return value1;
}
```

Accessed internal manner: value1

Blockchain Applications and Smart Contracts

- **Utilize Solidity function modifiers**

Function Modifiers

```
enum State { Created, Locked, Inactive }

modifier isOwner() {
   if (msg.sender != owner) { throw; }
   _; // continue executing rest of method body
}

modifier isState(State _state) {
   require(state == _state);
   _;
}

modifier cleanUp() {
  _;  // finish running method body
  // clean up code
}

doSomething() isOwner isState(State.Created) cleanUp {
  // code
}
```

- Function modifiers (like asserts or decorate patterns) are inherited and can be overridden
- Use "_;" to refer to main body of code in the method being modified
- Some examples of function modifier
- How would you write a modifier to protect a contract from reentrant calls?

Blockchain Applications and Smart Contracts

```
contract ExampleWithMutex {
        bool locked;
        modifier noReentrancy() {
                require(!locked);
                locked = true;
                _;
                locked = false;
        }

        /// This function is protected by a mutex, which means that
        /// reentrant calls from within `msg.sender.call` cannot call `f` again.
        /// The `return 7` statement assigns 7 to the return value but still
        /// executes the statement `locked = false` in the modifier.
        function f() public noReentrancy returns (uint) {
                require(msg.sender.call());
                return 7;
        }
}
```

- **Understand Solidity error checking**

Handling of errors is very important in Solidity. The old way could sacrifice gas (used prior to Solidity 0.4.10):

```
if(msg.sender != owner) { throw; }     //do not use
```

Three ways to check for errors:

1. **require**
   - Refunds remaining gas to caller
   - Use to validate inputs or state conditions
   - Can return a value
   - Most often used towards beginning of function

```
require(msg.sender == owner);
```

Blockchain Applications and Smart Contracts

2. **revert**
   - Refunds remaining gas to caller
   - Use same as require, but for more complex logic
   - Can return a value

   if(msg.sender != owner) { revert(); }

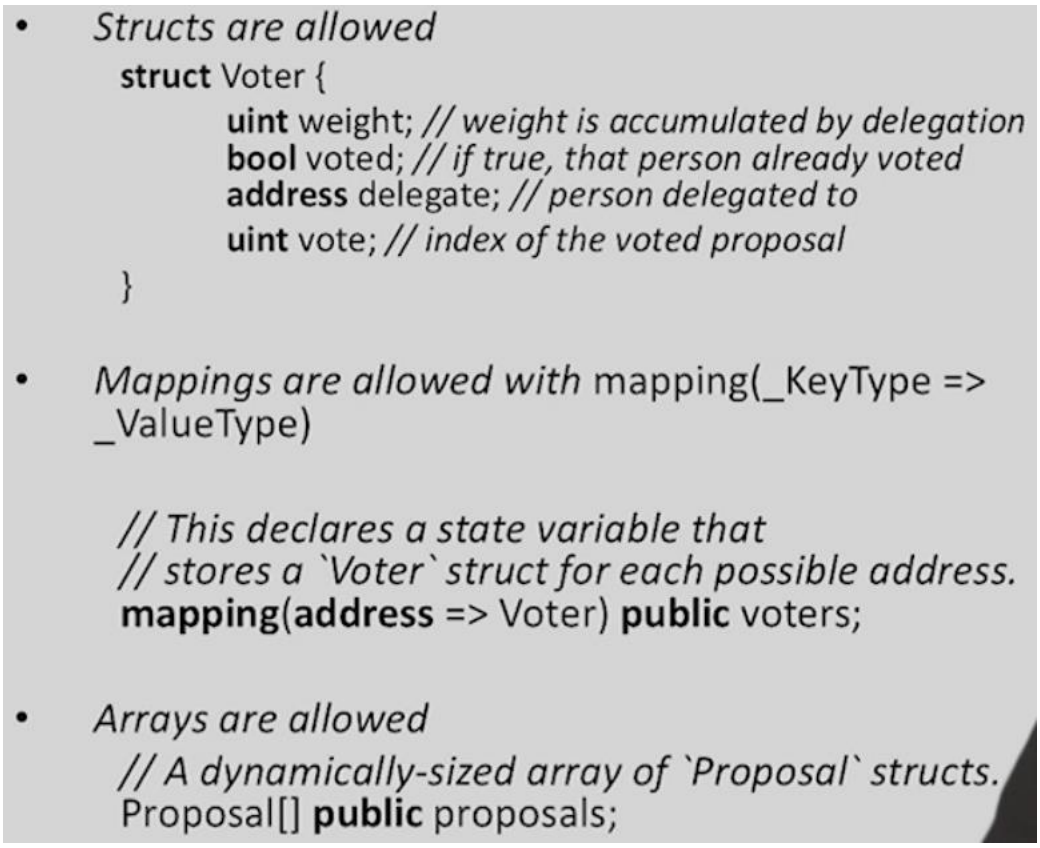
3. **assert**
   - Gas sacrificed to the asserted transaction
   - Use to check overflow/underflow
   - Should never be reached…an error in the code
   - Most often used towards end of function

   assert(this.balance > totalSupply);

- Fallback function: **`function()`**
  - Cannot take arguments but can access msg.data
  - Used when just Ether is sent to the contract. Need this function to be payable to receive the Ether!
  - Also used when called function does not exist. Avoids the default behavior of throwing an exception

- Blockchain data accessible from the contract:
  - now
  - this.balance, <address>.balance
  - block.number, block.timestamp, etc.
  - msg.value, msg.sig, msg.data
  - tx.gasprice, tx.origin

Blockchain Applications and Smart Contracts



4. **Testing, Debugging, and Deploying Smart Contracts**
5. **Smart Contracts Example: a Custom Token in Ethereum**