# Designing Blockchain Applications on Ethereum

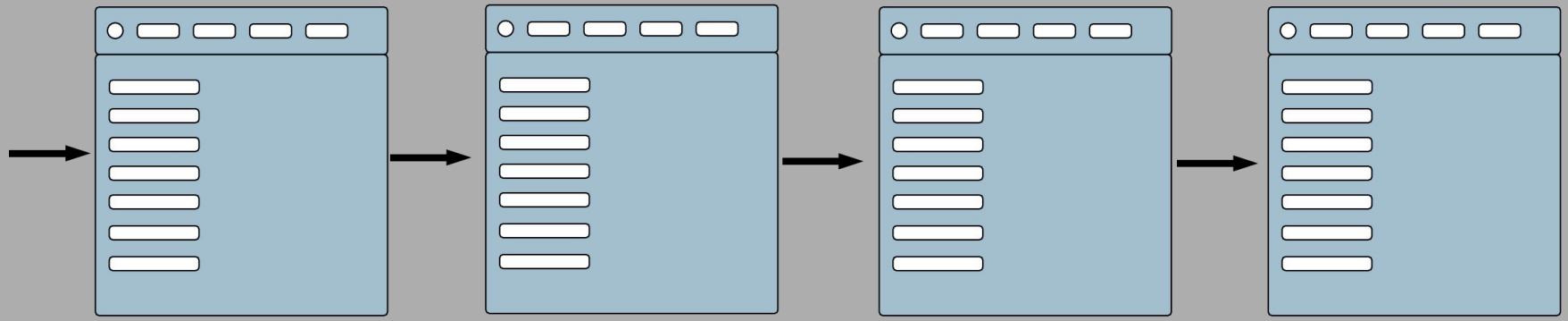
# Review of blockchain basics

# Where blockchain technology comes from (and why we should care about it)

- Born on the internet, and designed for “public use”.  With malicious actors assumed to be part of the system, strong security and non-reliance on trusted parties is a survival requirement.
- Now being adapted for enterprise due to efficiencies that emerge from that approach.
- Real-world high security track record despite attackability and value.
- Track record comes both from the technical foundations of blockchain, and from open source, collaborative projects that allow many experts to examine the security model before it is deployed.

# What is a blockchain?

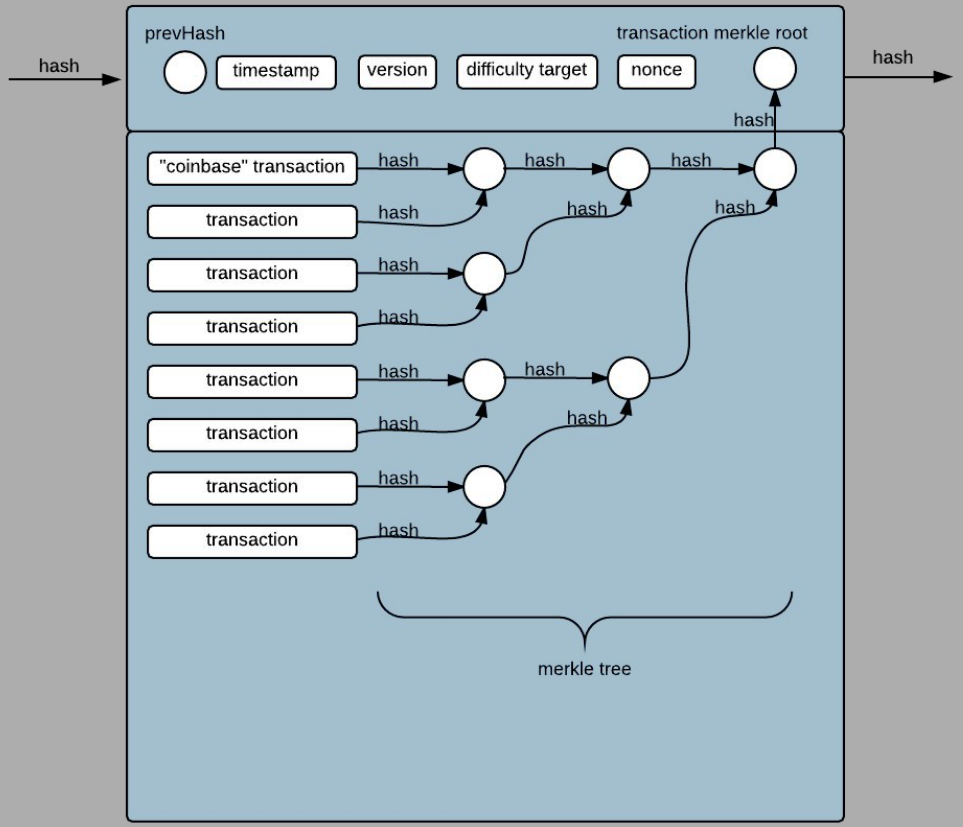
- A blockchain is a series of connected data structures called **blocks**, which contain or track everything that happens on some distributed system
- Each block is linked to and depends upon a previous block, forming a **chain**
- Blocks can only be added to the “end” of the chain, resulting in an **append only** system:  a permanent and irreversible history that can be used as a **real-time audit trail** by any participant to verify the accuracy of the records simply by reviewing the data itself



# Key concept: **Block structure**

**Different blockchains are different. But every block contains 3 fundamental components:**

1. A list of all new transactions or operations being performed
2. The “state” of the blockchain that results from these operations
3. A record of consensus: the mathematical evidence that the network as a whole agrees on this sequence of events



# Digging deeper: **Smart Blockchains & Contracts**

## What kind of operations can a blockchain contain?

Blockchain transactions are more than just “data entry”:  they are short computer scripts.  Those scripts may be capable of describing:

**1. A single or narrow set of operations**

Early blockchain technologies contained transactions that described specific operations, such as the ownership and transfer of currency tokens, resulting in only an implicit idea of the blockchain’s “state”.

**Example:** Dedicated cryptocurrencies such as Bitcoin

**2. Any arbitrarily complex operation**

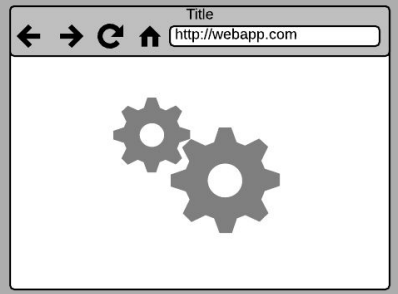
“Smart” chains provide more complete programming capabilities, allowing scripts to store memory or call other scripts.  These “blockchain programs” are called **smart contracts** but are better thought of as incorruptible **smart agents**.

**Example:** Generalised blockchains such as Ethereum

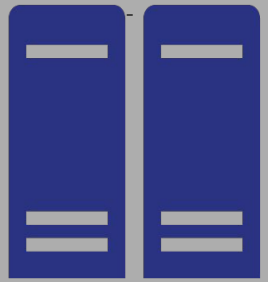
# Web apps vs blockchain apps

# Traditional Web App

Web client

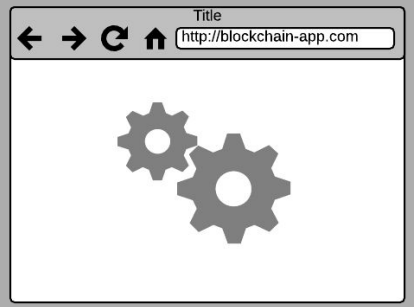


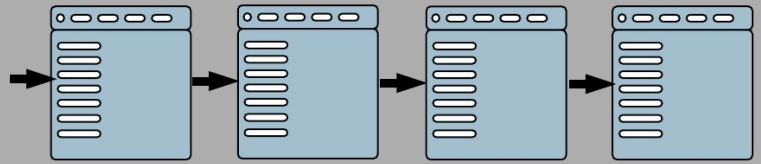

Server



# Blockchain App

Web client



blockchain

# Big Differences

On the plus side:

- The blockchain is fully deterministic (given a particular set of inputs)
- The blockchain cannot be censored or taken offline (i.e. DoS’d)
- The blockchain cannot be hacked or corrupted
- ..even by the application developer!

On the minus side:

- The blockchain has no secrets
- Blockchain operations are very expensive

# Structure of a blockchain application

Blockchain (“server”)

- Your code stored on blockchain
- Handles “trusted” logic where blockchain-consensus is necessary

# Structure of a blockchain application

Off-chain webapp (“client”)

- Handles off-chain operations that do not require high levels of trust or consensus
- Handles user interface
- Includes or is connected to a blockchain client like geth over RPC or etc

# Key Ethereum concepts

# Ethereum - the “Consensus as a Service” platform

- **Ethereum** is a blockchain with a built-in programming language
  - Alternatively: a consensus-based globally executed virtual machine
  - Or a decentralized computer for running code
  - It lets developers focus on building their app, without worrying about decentralization
- The Ethereum Virtual Machine (EVM) will run the code you submit
- Less of a currency, and more of a platform for building decentralized apps and smart contracts
- Open Source (all network infrastructure code and specs on github)
- No servers required
- No need to create user accounts or log-ins; it’s like Open-ID by default

# Ethereum Contracts

Ethereum “smart contracts” or “smart agents” have two parts:

**Code**

- Consists of **Ethereum Virtual Machine (EVM) bytecode**, submitted into the blockchain via a regular transaction and stored there publicly.
- For development purposes, smart contract code is usually written in a high level object-oriented language called **Solidity**, then compiled to EVM bytecode (like Java!) before being submitted.
- Like most software “objects”, the smart contract exposes a set of functions which can be called by either users or other contracts.  Each function can take arguments, modify object variables, and return results.

Solidity realtime compiler and runtime - Google Chrome

Solidity realtime com

https://chriseth.github.io/browser-solidity/

```
contract Ballot {
    struct Voter {
        uint weight;
        bool voted;
        uint8 vote;
        address delegate;
    }
    struct Proposal {
        uint voteCount;
    }

    address chairperson;
    mapping(address => Voter) voters;
    Proposal[] proposals;

    // Create a new ballot with $(_numProposals) different proposals.
    function Ballot(uint8 _numProposals) {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;
        proposals.length = _numProposals;
    }

    // Give $(voter) the right to vote on this ballot.
    // May only be called by $(chairperson).
    function giveRightToVote(address voter) {
        if (msg.sender != chairperson || voters[voter].voted) return;
        voters[voter].weight = 1;
    }

    // Delegate your vote to the voter $(to).
    function delegate(address to) {
        Voter sender = voters[msg.sender]; // assigns reference
        if (sender.voted) return;
        while (voters[to].delegate != address(0) && voters[to].delegate != msg.sender)
            to = voters[to].delegate;
        if (to == msg.sender) return;
        sender.voted = true;
        sender.delegate = to;
        Voter delegate = voters[to];
        if (delegate.voted)
            proposals[delegate.vote].voteCount += sender.weight;
        else
            delegate.weight += sender.weight;
    }

    // Give a single vote to proposal $(proposal).
    function vote(uint8 proposal) {
        Voter sender = voters[msg.sender];
        if (sender.voted || proposal >= proposals.length) return;
        sender.voted = true;
        sender.vote = proposal;
        proposals[proposal].voteCount += sender.weight;
    }

    function winningProposal() constant returns (uint8 winningProposal) {
        uint256 winningVoteCount = 0;
        for (uint8 proposal = 0; proposal < proposals.length; proposal++) {
            if (proposals[proposal].voteCount > winningVoteCount) {
                winningVoteCount = proposals[proposal].voteCount;
                winningProposal = proposal;
            }
            ++proposal;
        }
    }
}
```

# Solidity realtime compiler and runtime

Version: 0.1.3-4457170b/.-Emscripten/clang/int linked to libethereum-

Change to: v0.1.3-2015-09-28-4457170

Execution environment does not connect to any node, everyhing is local and in memory only.

`tx.origin = 0xca35b7d915458ef540ade6068dfe2f44e8fa733c`

Text Wrap  Enable Optimization

**Ballot** 1796 bytes

Create uint8 _numProposals

Bytecode: 6060604052604051602080610704833901604052606080519060200150 5b336000606006101000a81

Interface: [{"constant":false,"inputs":[{"name":"to","type":"address"}],"name":"delegate","outputs":[],"type":"functi

Web3 deploy: `var _numProposals = /* var of type uint8 here */ ;`

uDApp: `[{"name":"Ballot","interface":"[{\"constant\":false,\"inputs\": [{\"name\":\"to\",\"type\":\"address\"}],\"name\":\"delegate\",\"outputs\`

Details

A fully functional voting system on Ethereum using Solidity Realtime Compiler

# Ethereum Contracts (continued)

In addition to code, Ethereum contracts also have:

**State**

- An ether (ETH) balance that the object itself owns.  This balance can be sent by code executed within the contract's functions but not by third parties (unless so implemented).
- Whichever local variables you create will also be tracked, in keeping with the object oriented paradigm.  These act as the contract's "memory" and can only be changed by calling the contract's functions.
- The contract's bytecode is state too!  For example, you can "delete" a contract's code if it has been designed with "suicide" functionality.

# Gas

- Computation performed by the ethereum network must be compensated
- There must be a cost to operations on ethereum to prevent spam
- Gas is the mechanism used to pay for operations
- Every operation has a “gas cost” that must be paid by the transaction that initiates an action
    - e.g. a transaction that creates a contract must contain sufficient ether to pay the gas cost to store that contracts data
    - E.g. a transaction that triggers a contract to operate must contain enough ether to pay the gas cost for all the operations triggered by it
- Gas =/= ether. Instead, there is a floating conversion rate

# Transactions

There are 5 steps to interacting with a “smart contract” or “smart agent” on the Ethereum blockchain:

1. On the local client, compose a blockchain transaction. This transaction has a “from” field and a “to” field, with the from address paying the fees (gas) for anything the transaction causes to happen. It optionally attaches some ether which will be transferred along with the message, and optionally calls some functions of the “to” address, possibly with arguments. The destination contract may in turn call more contracts as a result of your call: you will be responsible for all the gas.
2. Sign this transaction cryptographically, proving that you have the authority to initiate transactions from the “from” address.
3. Send this signed transaction to one or more Ethereum nodes. You may use an API, connect to a local node, or connect to the Ethereum p2p network.
4. Wait 12-15 seconds.
5. Once the transaction is included in the Ethereum blockchain, the operation is complete!

# Levels in the Ethereum stack

- **Low-level:** Ethereum Clients such as Go-Ethereum, Cpp-Ethereum, Py-Ethereum, EthereumJ, ethereumjs, ethereumH, Parity (Rust), ruby-ethereum

- **Middle-level**: Smart-Contracts (high level Solidity, Serpent etc. compile to contract bytecode that runs in the EVM)

- **High-level:** Web frameworks (HTML, JS, CSS, AngularJS, MeteorJS, ReactJS, NPM etc.) that interact with the above

Let’s make one!

# First, we need a smart agent on the blockchain!

(All links are on my twitter feed @technocrypto)

Open this window:

https://ethereum.github.io/browser-solidity/

And paste in this code:

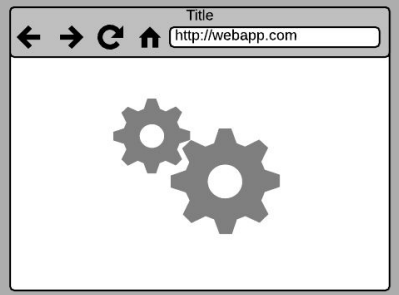
https://github.com/ledgerlabs/dapp-testrpc/blob/master/SimpleStore.sol

# Live dapp creation

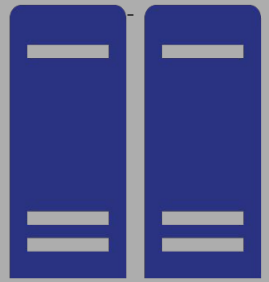
# Technique and best practices

# Web App

Web client



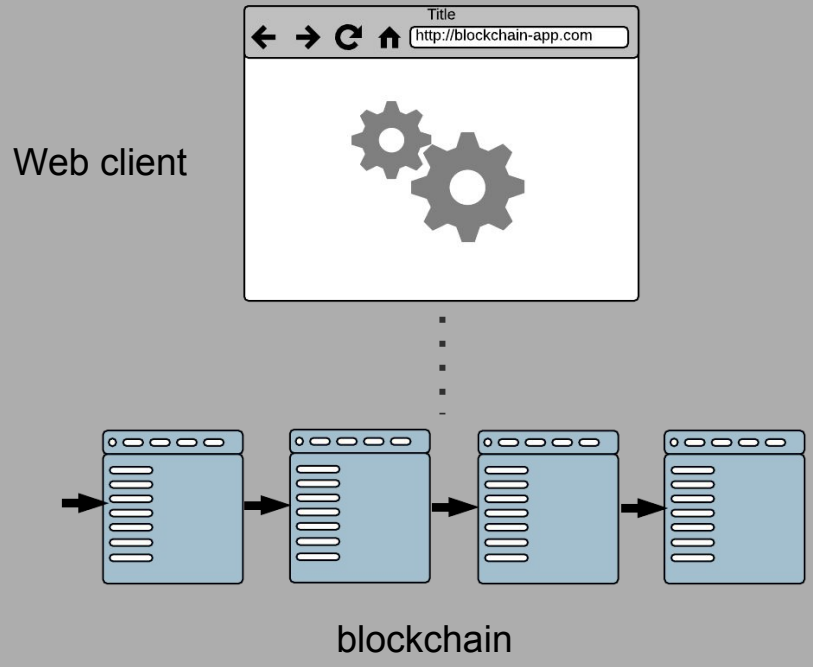

Server



# Traditional Web Application

- Two parts:
  - HTML5 client running in a browser
  - [insert server language here] server
- Good design consists of:
  - Maximising performance by letting the client handle user interface as well as most computation
  - Maximising security by letting the server handle authentication, maintenance of secure state, and anything the client can’t be trusted to do
  - Building secure cryptographic links (TLS, session tokens, etc.) between the two that both protect the users and prevent the server from being DoS’d.
  - Keep both parts reliable, but especially the server, which ideally has 100% uptime

# Blockchain App



# Blockchain-based Application (aka “Dapp”)

- Two parts:
  - HTML5 client/UI running anywhere [or native/embedded, or whatever, HTML5 is just easiest]
  - Blockchain-based “smart contract” or “smart agent” with code, state, etc.
- Good design consists of:
  - Maximising performance by letting the client handle user interface as well as most computation
  - Maximising security by letting the blockchain handle authentication, maintenance of secure state, and anything the client can’t be trusted to do
  - Building secure cryptographic links (asymmetric signatures, etc.) between the two to protect the user while minimising on-chain operations.
  - Keep both parts reliable, but especially on-blockchain code, which is not easily changed and provides the application users with strong security guarantees

# Ethereum Design Paradigms

- **Blockchain operations are expensive!** Therefore anything that can be done securely outside of the chain should be. Data should be stored in adjunct integrity-guaranteeing systems like IPFS. Advanced approaches to communications and computation include state channels and counterfactual verification. When on-chain watch gas levels closely. An effective implementation should not use very much storage or gas!

- **You are not smart enough to secure others' money!** Therefore use modular, widely vetted components and contracts instead of rolling your own. When you do write your own, publish the code, and obtain third party reviews and security audits before accepting others' funds.

# Ethereum Design Paradigms (continued)

- **Test, test, test.  Mistakes can be irreversible!**  Unlike in most software paradigms, blockchain consequences can be extremely permanent.  Every serious contract should go through at least three stages of testing:  local testing, live TestNet deployment, and then **pre-value** livenet deployment with small, sample amounts of resources.  All stages should be heavily reviewed.

- **Use simple, modular design!**  This will make it substantially easier to secure, debug, and re-use your applications.  Individual solidity objects/contracts should be on the much smaller side in terms of lines of code compared to objects in a typical programming language.  Many traditional design patterns like factories work well in ethereum, however.

# Ethereum Limitations

- **The blockchain can't access external resources** In order to get external information into the blockchain, someone must submit it. Even widely known information like market prices may be nearly impossible to integrate without relying on "trusted oracles" which can create a serious security hole.
- Expensive computation cycles (paid in "gas")
- Blocktimes only 15 seconds (bitcoin is 10 minutes)
- Scalability (solutions in the pipeline)

# Resources

- Ethereum Tutorial: https://github.com/ledgerlabs/ethereum-getting-started/wiki
- Ethereum Wiki: https://github.com/ethereum/wiki/wiki
- Github Repo: https://github.org/ethereum
- Ethereum Forum: http://forum.ethereum.org
- “Homestead” docs: https://ethereum-homestead.readthedocs.io/en/latest/
- Dapps for Beginners: https://dappsforbeginners.wordpress.com
- Solidity documentation: https://solidity.readthedocs.io/en/latest/
- Browser solidity: https://ethereum.github.io/browser-solidity/
- TestRPC: https://github.com/ethereumjs/testrpc
- Go-Ethereum Installation Guide: https://github.com/ethereum/go-ethereum/wiki/Building-Ethereum

# Q&A

# Thanks!

LedgerLabs

Jeff Coleman
Head of Technology
ledgerlabs.com
info@ledgerlabs.com