



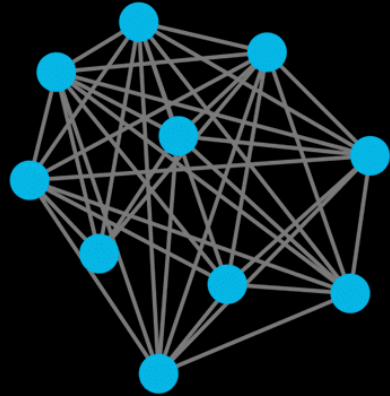
Blockchain Fundamentals: Architecture & Security

Jorden Seet

Lesson structure



**Dapp
Architecture**



**Blockchain
Architecture**



**Dapp
Security**



**Blockchain
Security**

Dapp Architecture



What is a DApp?

- Decentralised Application
- Not very different from a normal web application

**Smart
Contract
code**



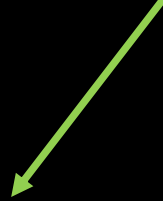
Blockchain



Server-side Scripting



Front-end Interface



Front-end Interface

- Interface component
 - Make the interface pretty and usable
 - HTML / App.js (React)
 - CSS
 - Other libraries like D3.js
 - Save certain arguments to pass into server-side

```
<body background = "../background.png">
```

```
<div class="container">
```

```
<div class="row">
```

```
<div class="col-xs-12 col-sm-8 col-sm-push-2">
```

```
<h1 class="text-center">One-Stop Letter Of Credit platform</h1>
```

```
<hr/>
```

```
<br/>
```

```
</div>
```

```
</div>
```

```
<h1><p id="contractAddress"></p></h1>
```

```
<form>
```

```
<h1>Create a Bill of Exchange</h1>
```

```
<br>
```

```
Exporter address: <input type = "text" id = "exporter">
```

```
<br>
```

```
Importer address: <input type = "text" id = "importer">
```

```
<br>
```

```
Shipper address: <input type = "text" id = "shipper">
```

```
<br>
```

```
Shipment Value: <input type = "number" id = "shipment" min = "0"><br>
```

```
<button class="btn btn-default btn-create" input type = "submit"> Submit </button>
```

```
<p id="boeCreation"></p>
```

```
<p id="boeCreatedDetails"></p>
```

index.html Webpage

One-Stop Letter Of Credit platform

Create a Bill of Exchange

Exporter address:

Importer address:

Shipper address:

Shipment Value:

Change Bill of Exchange contract value

Enter Bill of Exchange contract address:

Set Bill of Exchange value desired

Exercise Bill of Exchange

Enter Bill of Exchange contract address:

Auction for unexercised Bill of Exchanges

Enter Bill of Exchange contract address:

Set Bill of Exchange value desired

End Auction for unexercised Bill of Exchanges

Enter Bill of Exchange contract address:

Indicate Shipment has arrived

Server-side Scripting

- Receives and implement Smart Contract logic
- Interacts with front-end and Blockchain
- Provides the logic to manipulate passed arguments
 - Oracles
 - State Channels

Oracles

Smart
Contract



Customer



Smart
Contract

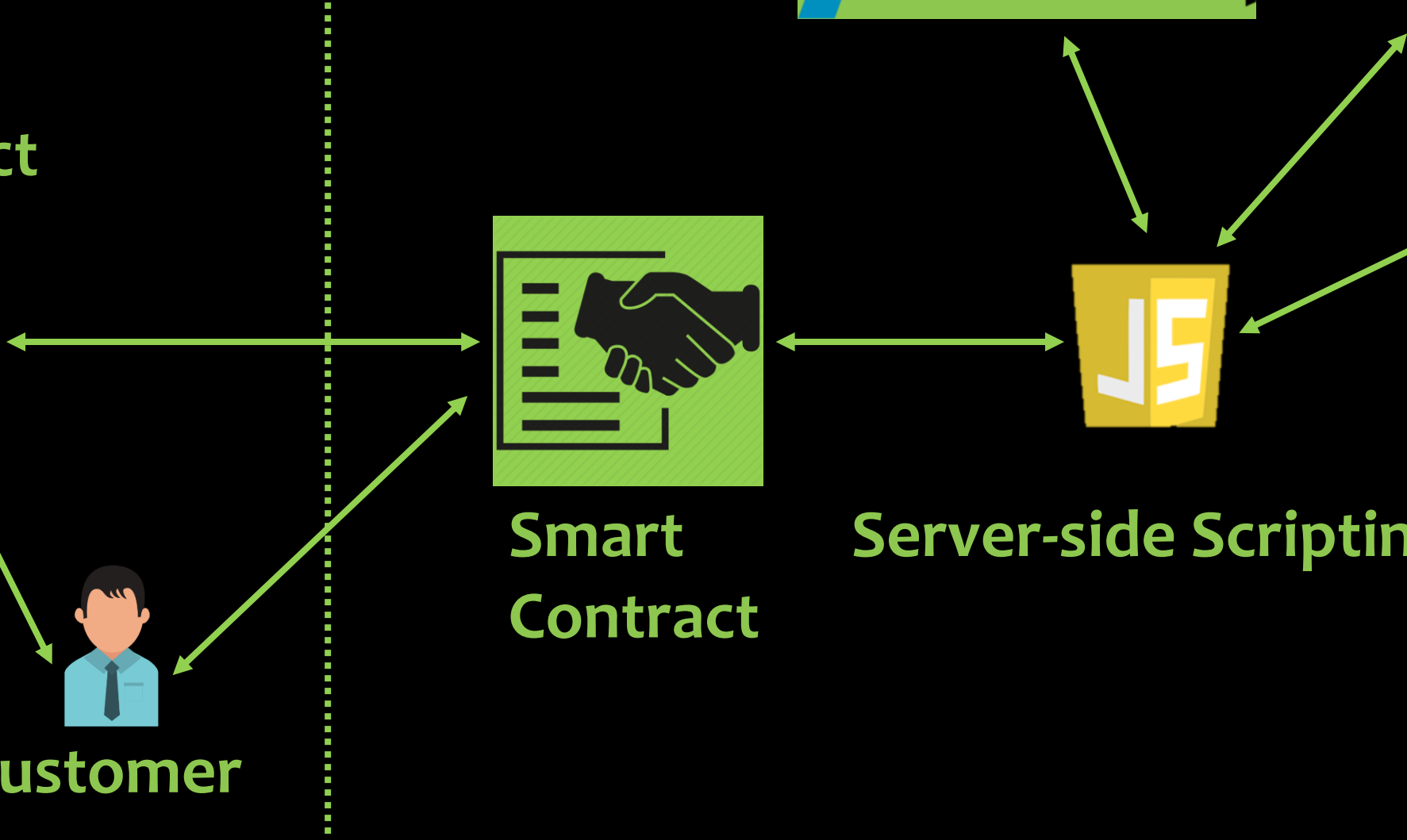
Various APIs



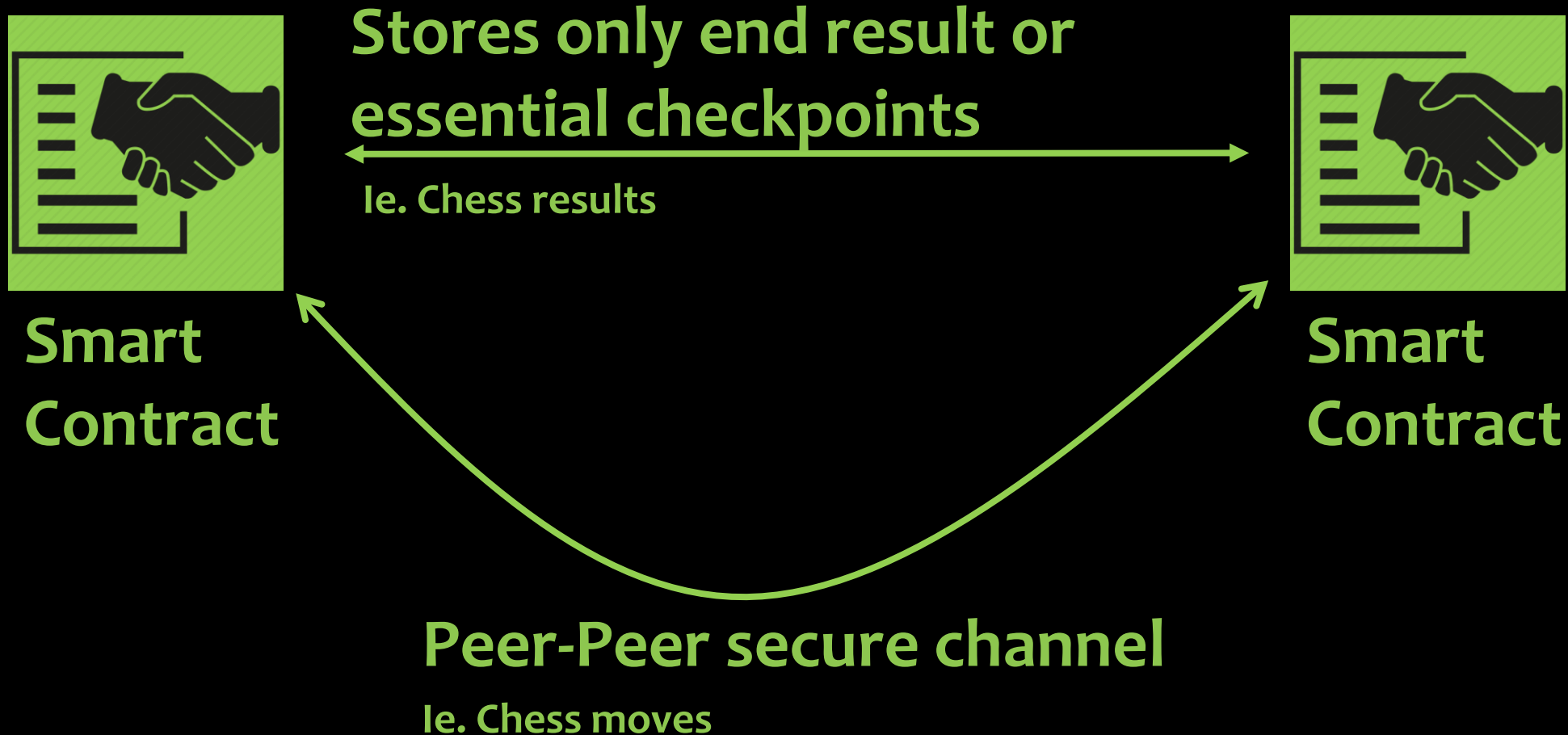
WolframAlpha



Server-side Scripting



State Channels



```
App = {
```

```
  web3Provider: null,  
  contracts: {},
```

```
  init: function () {  
    console.log("App started")  
    return App.initWeb3();  
  },
```

```
  initWeb3: function () {  
    // Is there an injected web3 instance?  
    if (typeof web3 !== 'undefined') {  
      App.web3Provider = web3.currentProvider;  
    } else {  
      // If no injected web3 instance is detected, fall back to Ganache  
      App.web3Provider = new Web3.providers.HttpProvider('http://localhost:8545');  
    }  
    web3 = new Web3(App.web3Provider);  
  
    return App.initContract();  
  },
```

```
  initContract: function () {  
    $.getJSON('LetterOfCredit.json', function (data) {  
      // Get the necessary contract artifact file and instantiate it with truffle-contract  
      var LetterOfCreditArtifact = data;  
      App.contracts.LetterOfCredit = TruffleContract(LetterOfCreditArtifact);  
    });  
  }  
}
```

```
// Set the provider for our contract
App.contracts.LetterOfCredit.setProvider(App.web3Provider);
});
```

App.js

```
return App.bindEvents();

},

bindEvents: function () {
  $(document).on('click', '.btn-create', App.createBoe);
  $(document).on('click', '.btn-setnp', App.setBOE);
  $(document).on('click', '.btn-exercise', App.exerciseBOE);
  $(document).on('click', '.btn-auction', App.auction);
  $(document).on('click', '.btn-auctionEnd', App.auctionEnd);
  $(document).on('click', 'btn-ship', App.setShip);
  $(document).on('click', '.btn-cert', App.certify);
  $(document).on('click', '.btn-details', App.getDetails);
  $(document).on('click', '.btn-stop', App.stop);
  $(document).on('click', '.btn-start', App.start);
},
```

```
createBoe: function(event) {
  event.preventDefault();
```

```
var LetterOfCreditInstance;
var importer = document.getElementById("importer").value;
var exporter = document.getElementById("exporter").value;
var shipper = document.getElementById("shipper").value;
```

```
<body background = "./background.png">
  <div class="container">
    <div class="row">
      <div class="col-xs-12 col-sm-8 col-sm-push-2">
        <h1 class="text-center">One-Stop Letter Of Credit platform</h1>
        <hr/>
        <br/>
      </div>
    </div>
    <h1><p id="contractAddress"></p></h1>
    <form>
      <h1>Create a Bill of Exchange</h1>
      <br>
      Exporter address: <input type = "text" id = "exporter">
      <br>
      Importer address: <input type = "text" id = "importer">
      <br>
      Shipper address: <input type = "text" id = "shipper">
      <br>
      Shipment Value: <input type = "number" id = "shipment" min = "0"><br>
      <button class="btn btn-default btn-create" input type = "submit"> Submit </button>
      <p id="boeCreation"></p>
      <p id="boeCreatedDetails"></p>
```

Smart Contract

- Smart → Automated by code
- Contract → Agreement or Workflow

```
function createBOE(address exporterAddr, address importerAddr, address shipperAddr, uint256 shipmentVal) public {  
    exporter = exporterAddr;  
    importer = importerAddr;  
    shipper = shipperAddr;  
    shipmentValue = shipmentVal;  
    shipmentStatus = "Shipped";  
    boe = BillofExchange({  
        holder: exporterAddr,  
        paymentAmount: shipmentValue  
    });  
  
    coi = CertificateOfInspection({  
        certified: false,  
        date: 0  
    });  
}
```

whatever.sol

Blockchain

- Tamper-resistant Database
- Distributed Ledger
 - Append-only
 - Peer to Peer
 - No single point of failure
 - Every node has the same data
 - Enforced through consensus algorithms

Why do Smart Contracts need the Blockchain?

- Smart Contracts require witnesses
 - Honest nodes operating (anonymously) in a distributed network
 - Ensure the veracity of transactions done
 - Verifying signatures, checking for known exploits etc
- Nodes may not necessarily be honest
- Smart Contracts need a Byzantine Fault Tolerant environment
 - Consensus must be reached amidst failure of nodes
 - Ensured (to a higher degree) through the Blockchain

Smart Contract Platform Design

- Privacy
- Turing-Completeness
- Language
- Network Bandwidth

Smart Contract Platform Design

- **Privacy**
- Is it important to keep certain information private?
 - Customer Risk scores, publically identifiable information etc
- Usage of Cryptographic protocols
 - Zero Knowledge Proofs – JP Morgan's Quorum
 - Channels – Hyperledger Fabric
- Uses Cryptographic proofs – computationally expensive
 - Need to ensure that only used when absolutely necessary

Smart Contract Platform Design

- Privacy
- Turing-Completeness
- Language
- Network Bandwidth

Smart Contract Platform Design

- **Turing-Completedness**
- The concept that the program can be used to simulate any known algorithm and derive an answer (assuming infinite runtime & memory)
- Pros: Can be used to program anything
- Cons: Turing-Completedness comes with limitless attack vectors for bad coding

Smart Contract Platform Design

- Privacy
- Turing-Completeness
- Language
- Network Bandwidth

Smart Contract Platform Design

- **Language**
- Compilation speed
 - Java vs Python
- Deterministic
 - Solidity vs C++
- Target Audience
 - Python vs Go
- Available Libraries
 - Solidity vs Rust

Smart Contract Platform Design

- Privacy
- Turing-Completeness
- Language
- Network Bandwidth

Smart Contract Platform Design

- **Network Bandwidth**
- Smart contract states are stored on the blockchain
 - Every change in state utilises network computation
- If bad/inefficient coding is not penalised, could lead to DoS or bottlenecked speeds
- Do we need to penalise such actions?
 - Private vs Public platforms
- Examples: Solidity Gas, EOS RAM

Smart Contract Use-cases



Escrow

Peer to Peer
Car sharing



Financial Workflows

Cross border payments



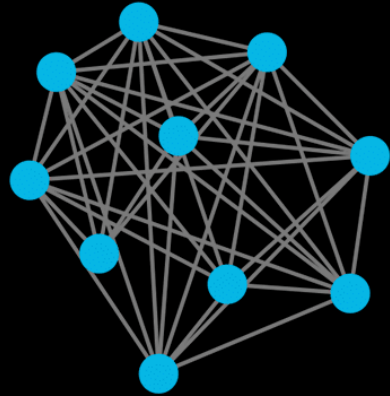
Supply Chain

Automated recording
and tracking

Lesson structure



**Dapp
Architecture**



**Blockchain
Architecture**

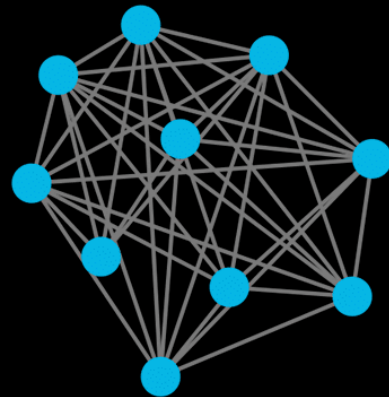


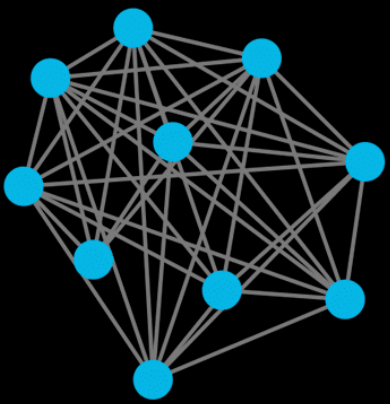
**Dapp
Security**



**Blockchain
Security**

Blockchain Architecture



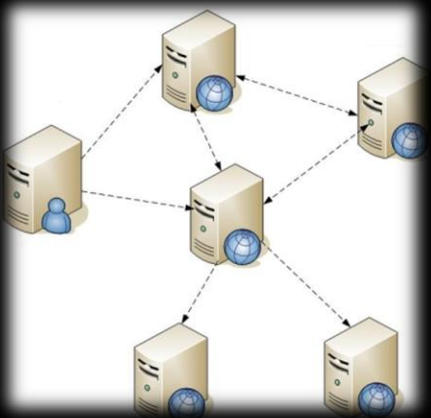


Distributed Ledger

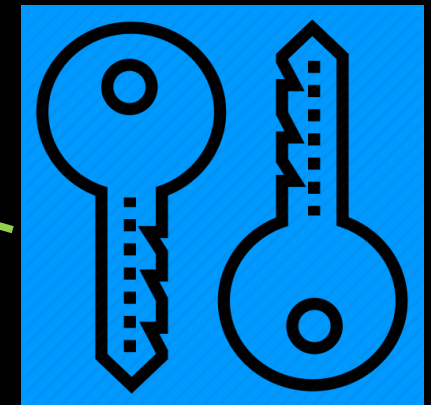


Distributed Messaging

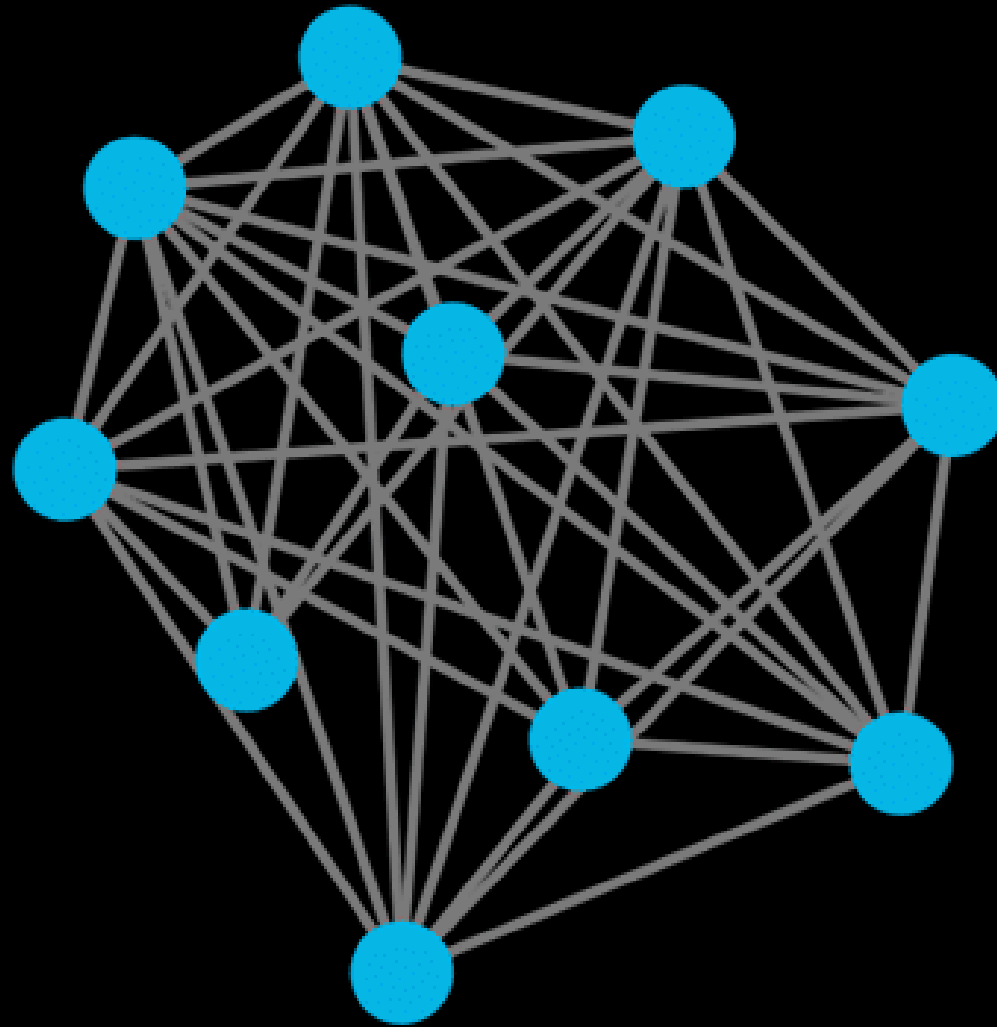
Blockchain



Distributed Computing



Cryptography

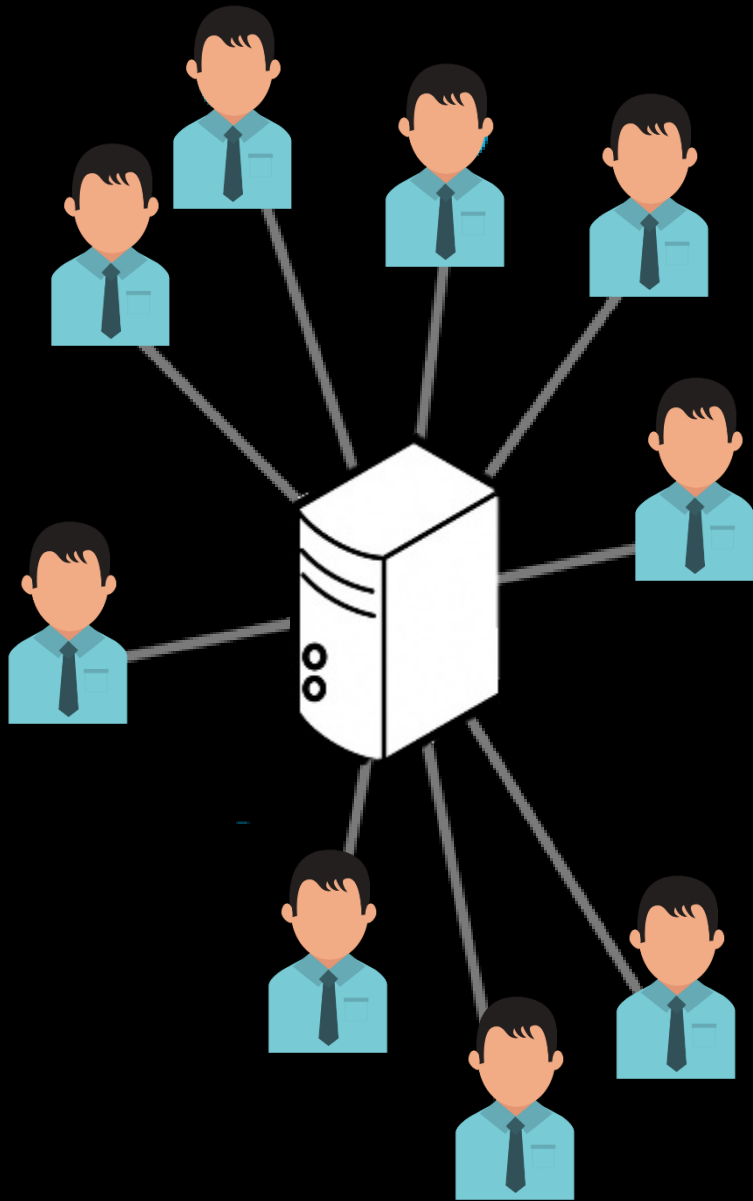


Distributed Ledger

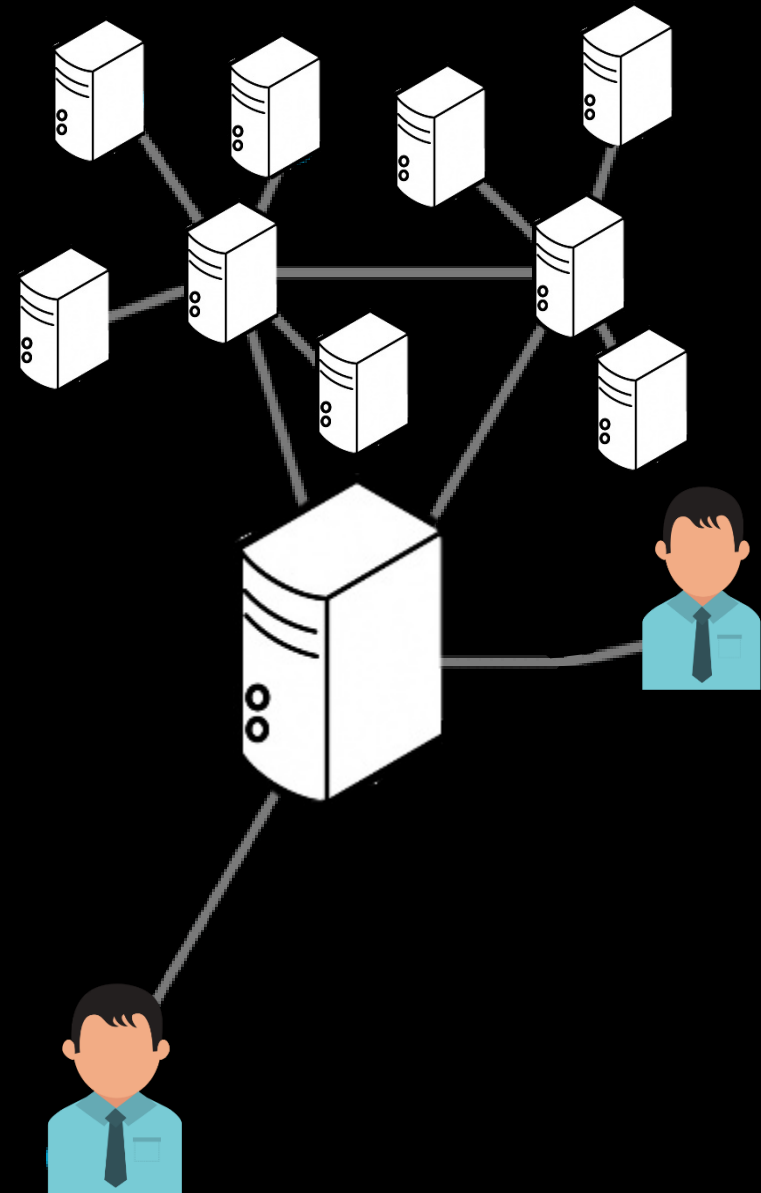
Distributed Ledgers

- “Advanced” Distributed Databases
- What is a Database?
 - A place that stores data
- What is a Distributed Database?

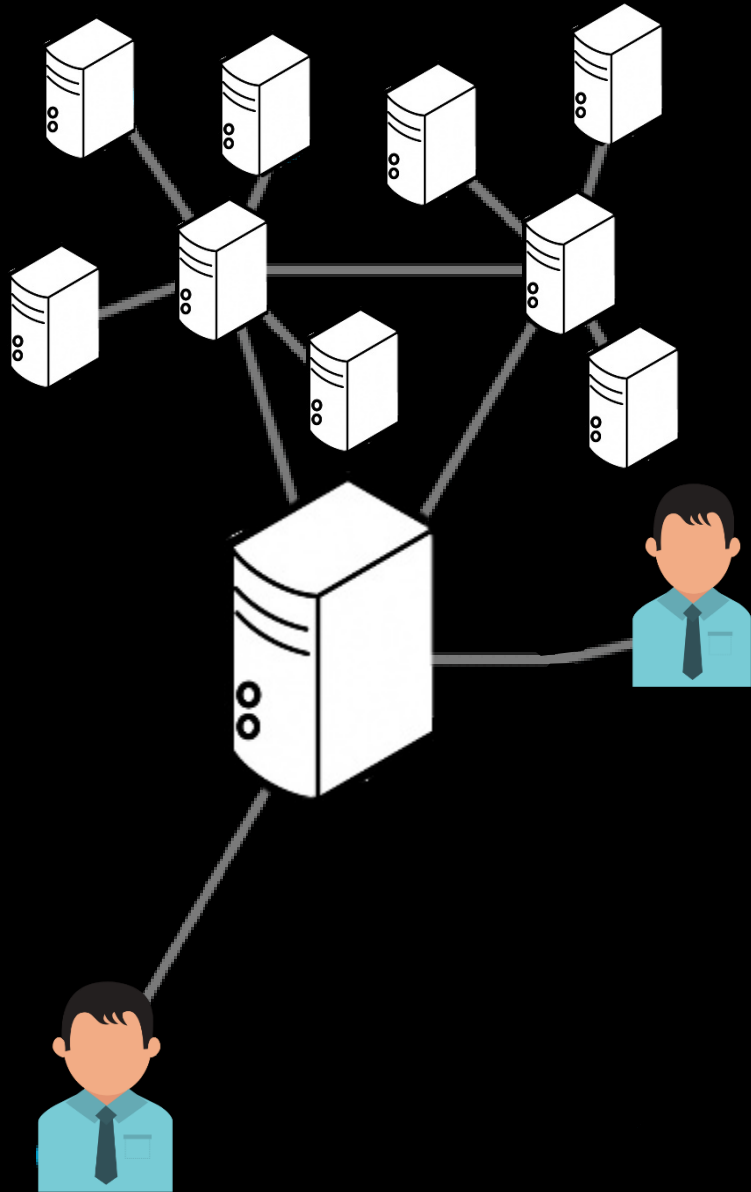
Database



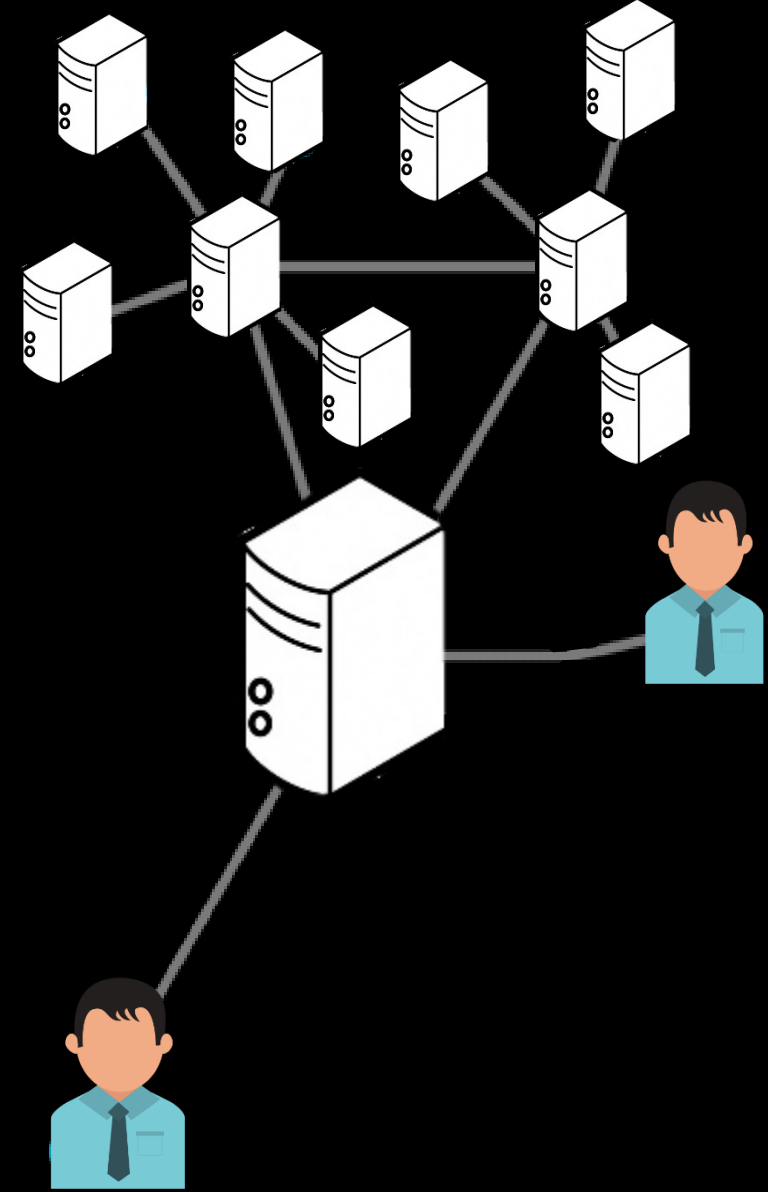
Distributed Database



Distributed Database



Distributed Ledger

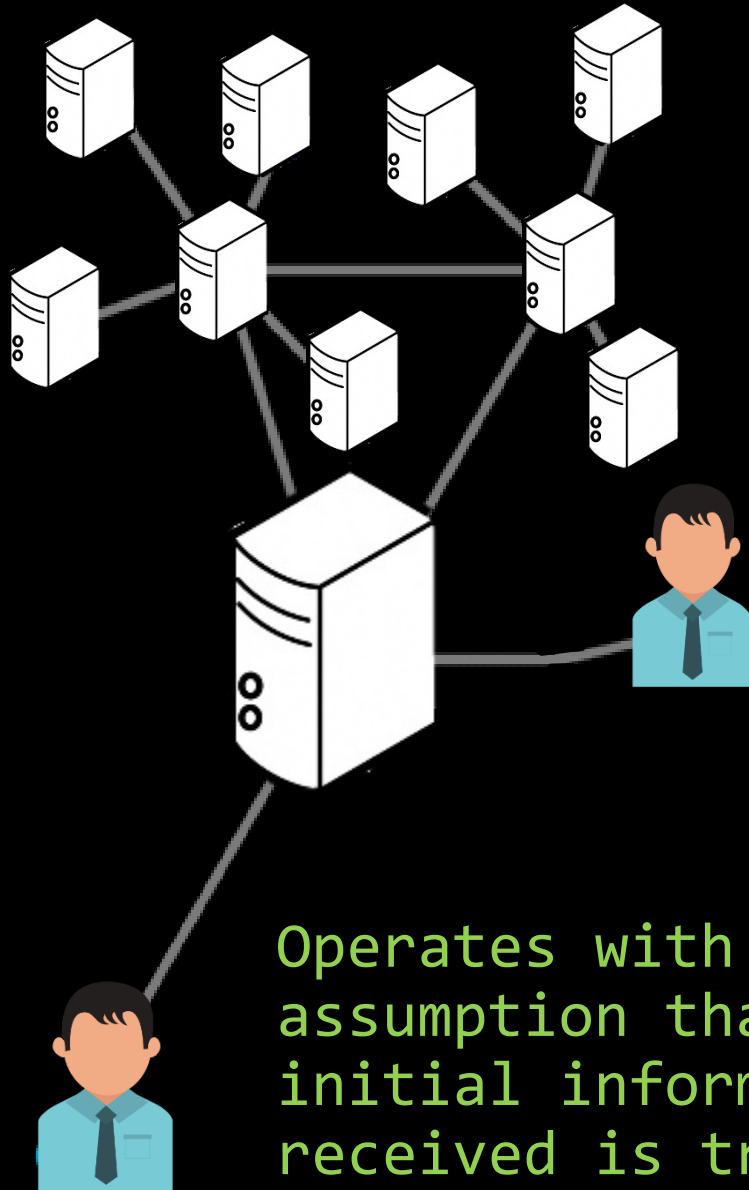




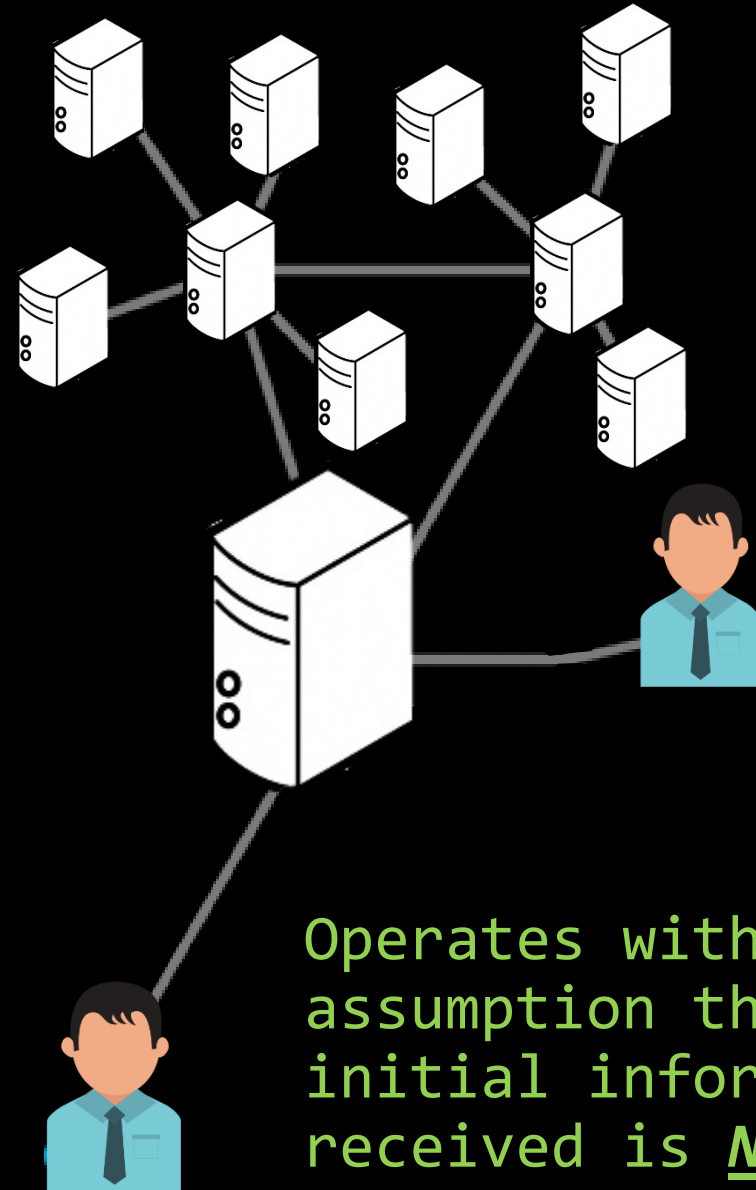
WHATS THE DIFFERENCE?



Distributed Database



Distributed Ledger



How can we operate without trusting the information?

- This is called the Byzantine Generals Problem
- Solving the Byzantine Generals Problem gives us the ability to operate with untrusted information
 - Byzantine Fault Tolerance

Byzantine Generals Problem

- Scenario
- Singapore and Malaysia are at War
 - Malaysia don't want to supply us water anymore
- Singapore war tactic is to slowly conquer parts of Malaysia
 - Now they facing dilemma over one of the states



- Brother, not so easy la. KL is the capital, confirm got hidden weapon one





- This one serious leh, we cannot half-in half-out, half-hearted effort confirm die one



- We must decide to whether to all-in or all-out





- That's what she said



Byzantine Generals Problem

- The important thing is that consensus must be achieved.
- Once the consensus is achieved, the Generals can lead their squad to attack or retreat.

BUT WHAT IF?

There was a traitor?





If I tell Perry and Siow Huang to attack,
then tell Ishak to retreat, huat liao GG
SG lose most of their army



I can join Malaysia then ez game win SG
SG will then come back to Malaysia



Voted Yes



Voted Yes



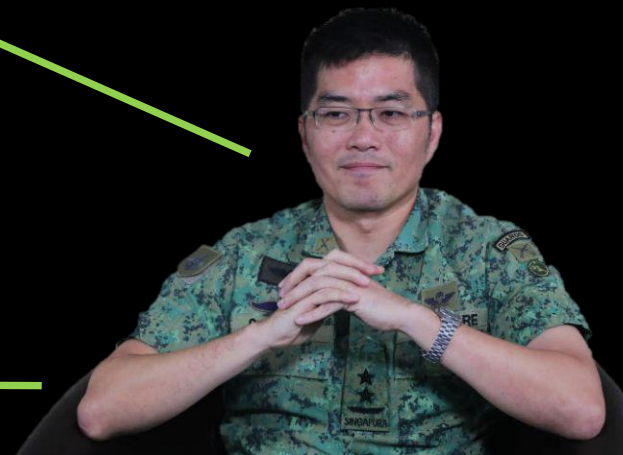
Go attack

Go attack

Voted No



Mai la don't geh siao



Byzantine Generals Problem

- The solution is that non-traitors must have a majority agreement on their strategy
 - Assumption: There are more non-traitors than traitors
- The only way to achieve that is to check with every other General on their decision, rather than relying on 1 General

Voted Yes



Voted Yes-Yes

Voted Yes



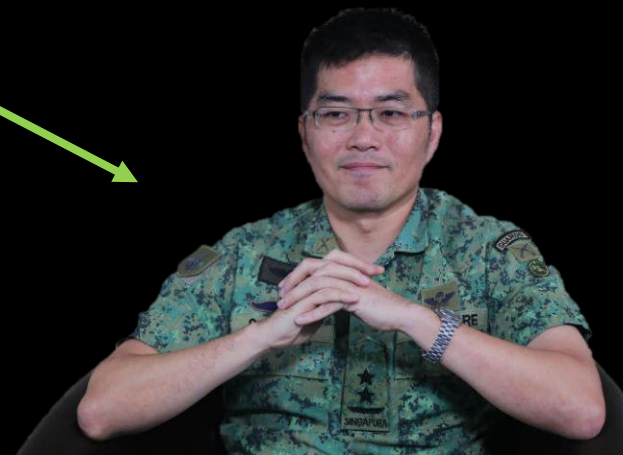
Voted Yes-No

Voted No

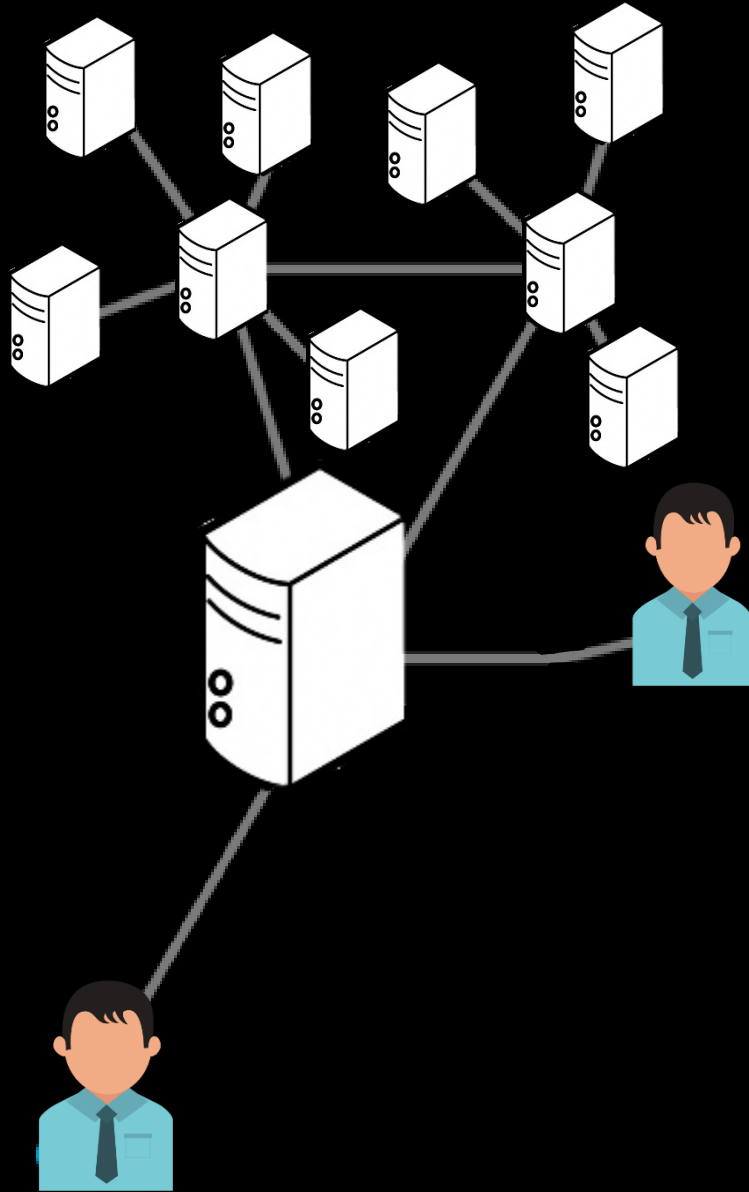


Voted Yes-No

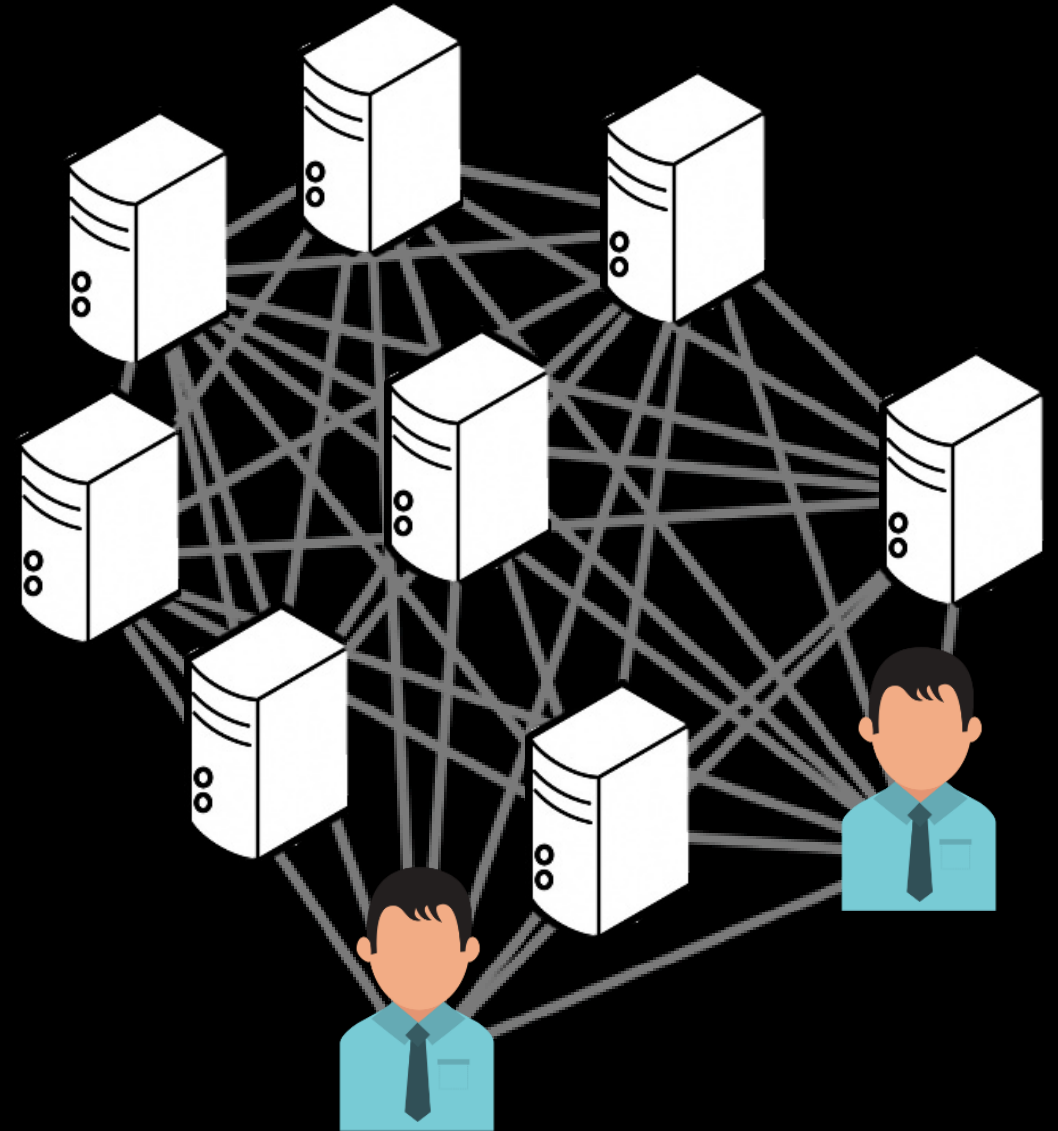
- They all know each other's votes le, can't bluff them le



Before



After



Byzantine Generals Problem

- Problem:
- This solution works in a permissioned setting, where voters are known and controlled.
- What happens in a public setting, where voters are unknown and uncontrolled?
- We could have someone create 10000000000 accounts and vote just to gain majority number of votes
 - This is called a Sybil attack

Byzantine Generals Problem

- Solution?
- Use probability
- Choose a random person inside the voter list to validate the votes
 - Assumption, more honest voters than dishonest voters
 - Hence, probability of successful validation of messages $\geq 51\%$
- The more random the selection, the better

Honest



Honest



75% Chance of picking
honest voter to validate

Honest



Malicious



Honest



Honest



Honest



Honest



85% Chance of picking
honest voter to validate

Honest



Honest



Malicious



Honest



Honest



Malicious



Honest



71% Chance of picking
honest voter to validate

Honest



Honest



Malicious



Byzantine Generals Problem

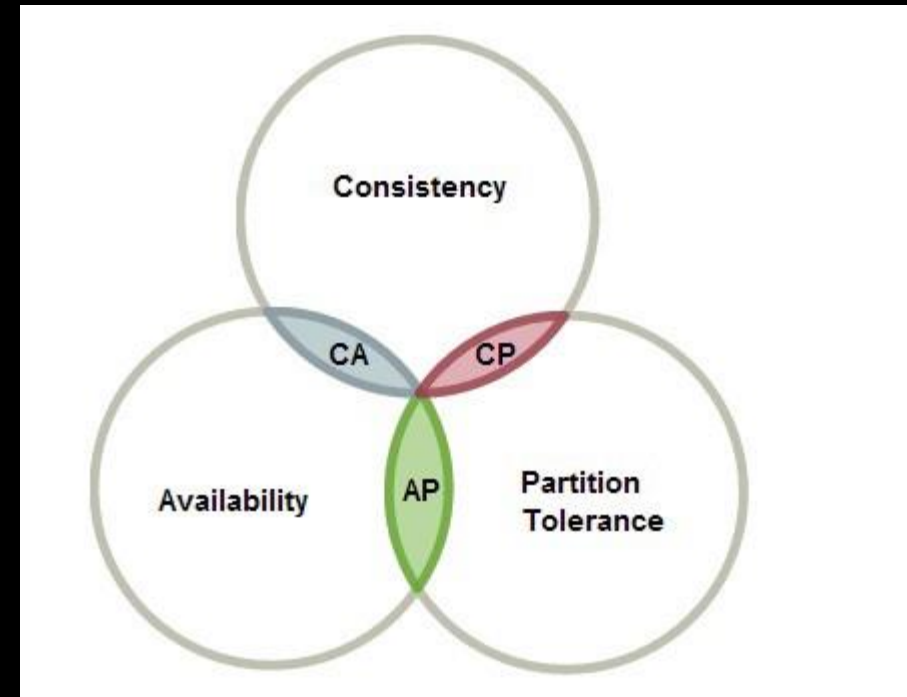
- How to deter malicious actors from creating accounts?
- Incur a cost when creating an account, or when making transactions
 - Proof of Work
- Punitive measures if found out that malicious
 - Proof of Stake

Distributed Ledgers Summary

- A Distributed Database with Consensus Mechanisms
- A Consensus Mechanism is derived from the Consensus Algorithm
- Consensus Algorithms are designed based on a variety of factors
 - Permissioned or Public network?
 - Nature of participants?
 - How to achieve randomness?
 - How scalable should it be?
 - How secure should it be?
 - Many others...

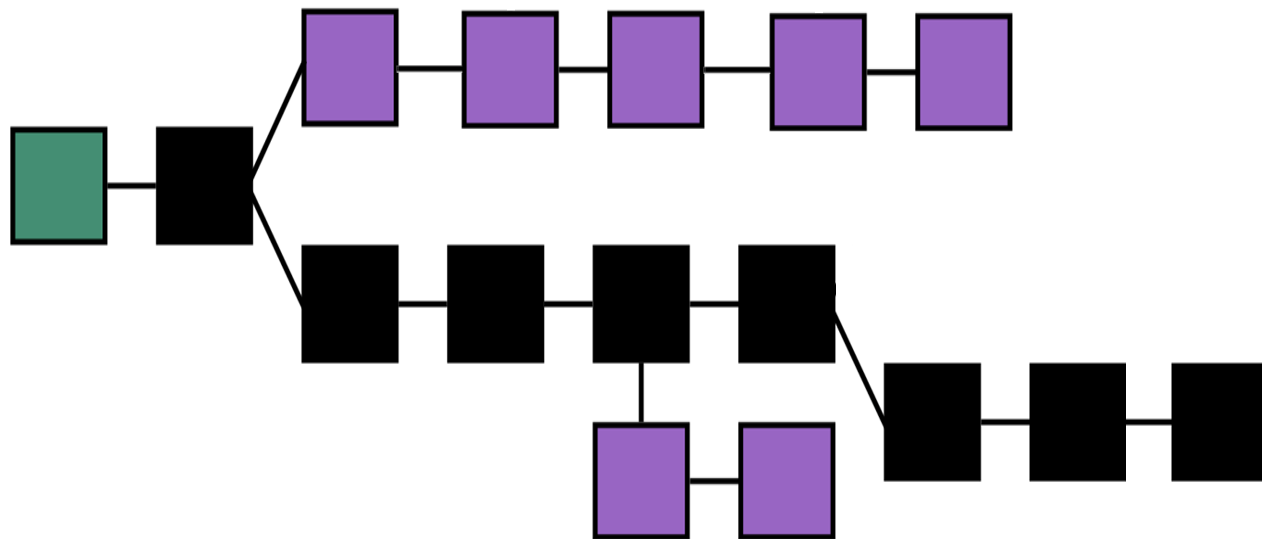
Distributed Ledgers Summary

- Consensus Algorithms are designed based on a variety of factors
- CAP Theorem
- Partition Tolerance is a necessity
 - Ability of system to run despite message delay
- Tradeoffs usually occur between Consistency and Availability
- Consistency
 - All data are consistent at all times
- Availability
 - Ability to get response on request



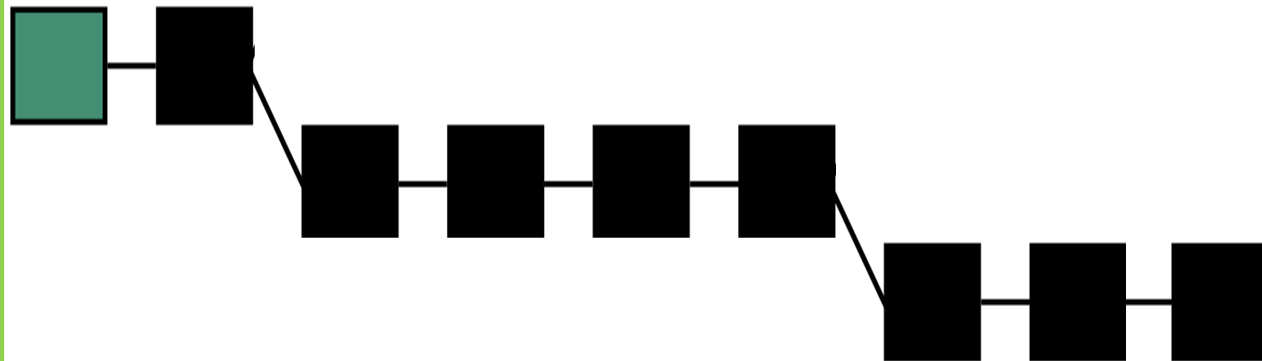
Probabilistic Algorithms

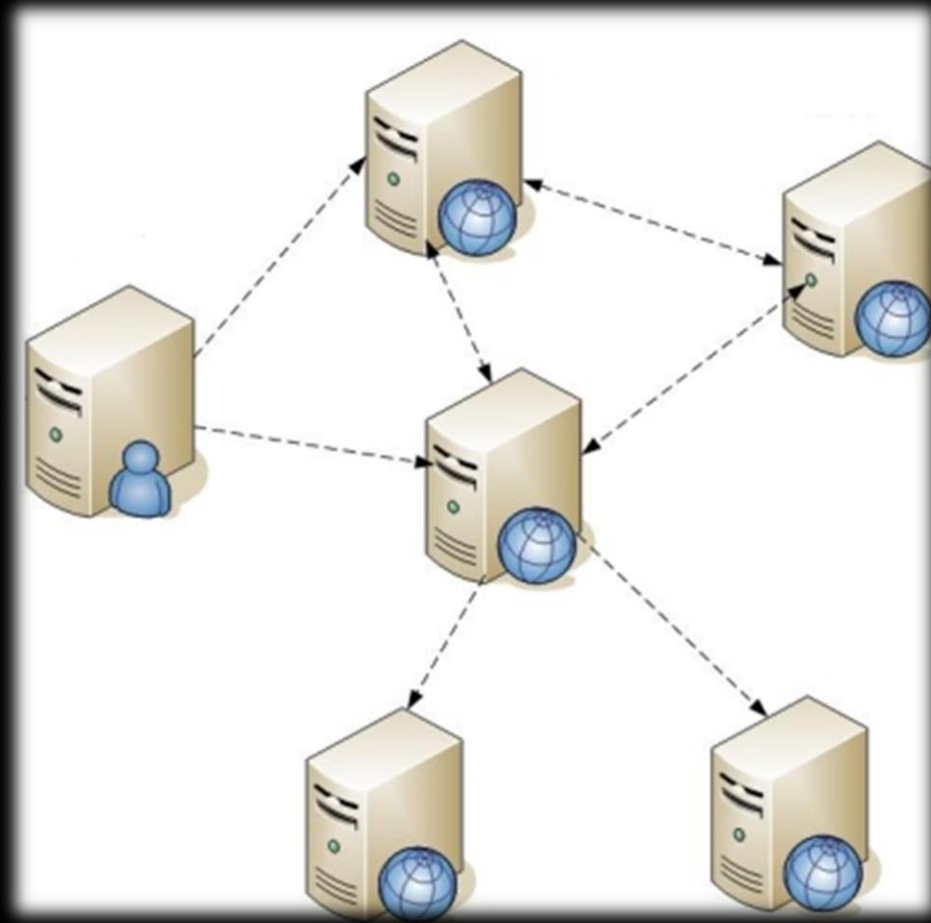
- Proof of xxxx
- Ability to split into multiple chains
- Consensus mechanism determined by longest chain
- Value Availability over Consistency



Byzantine Fault Tolerant Algorithms

- PBFT, IBFT, HBBFT etc
- Every block must have unanimous decision on validity
- If >33% different votes, vote again
 - System can retrieve new txns, but wont validate new blocks
- Value Consistency over Availability



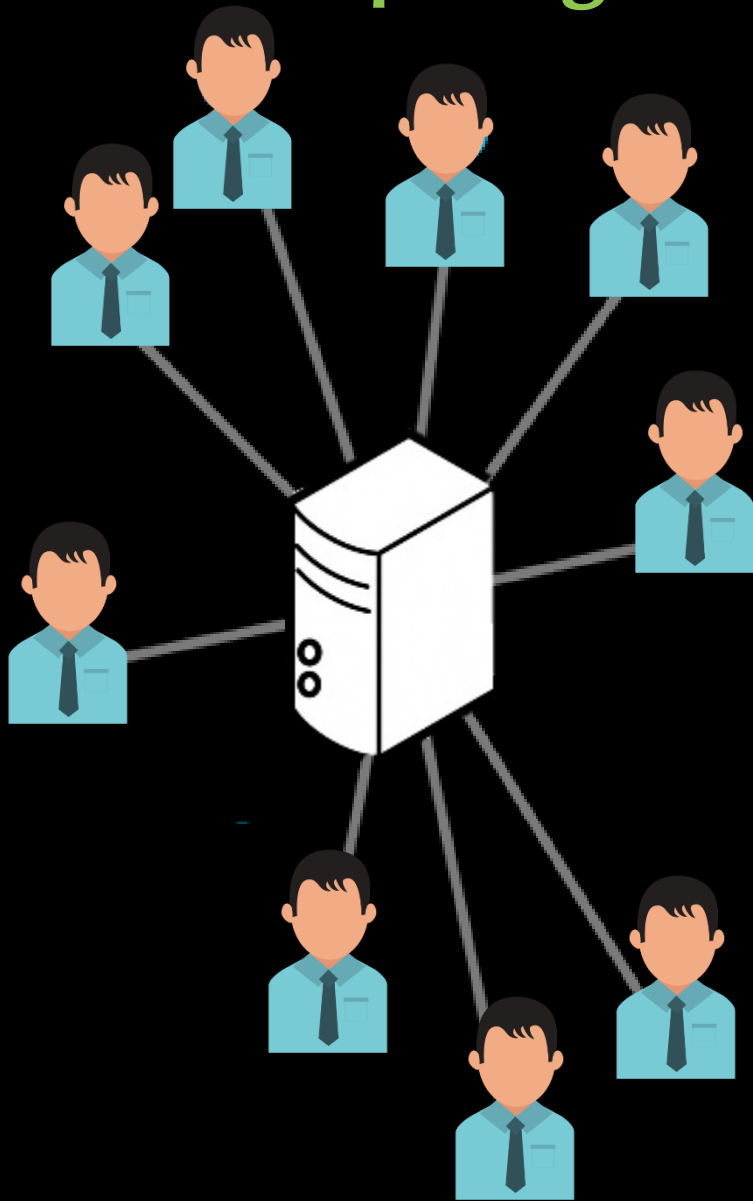


Distributed Computing

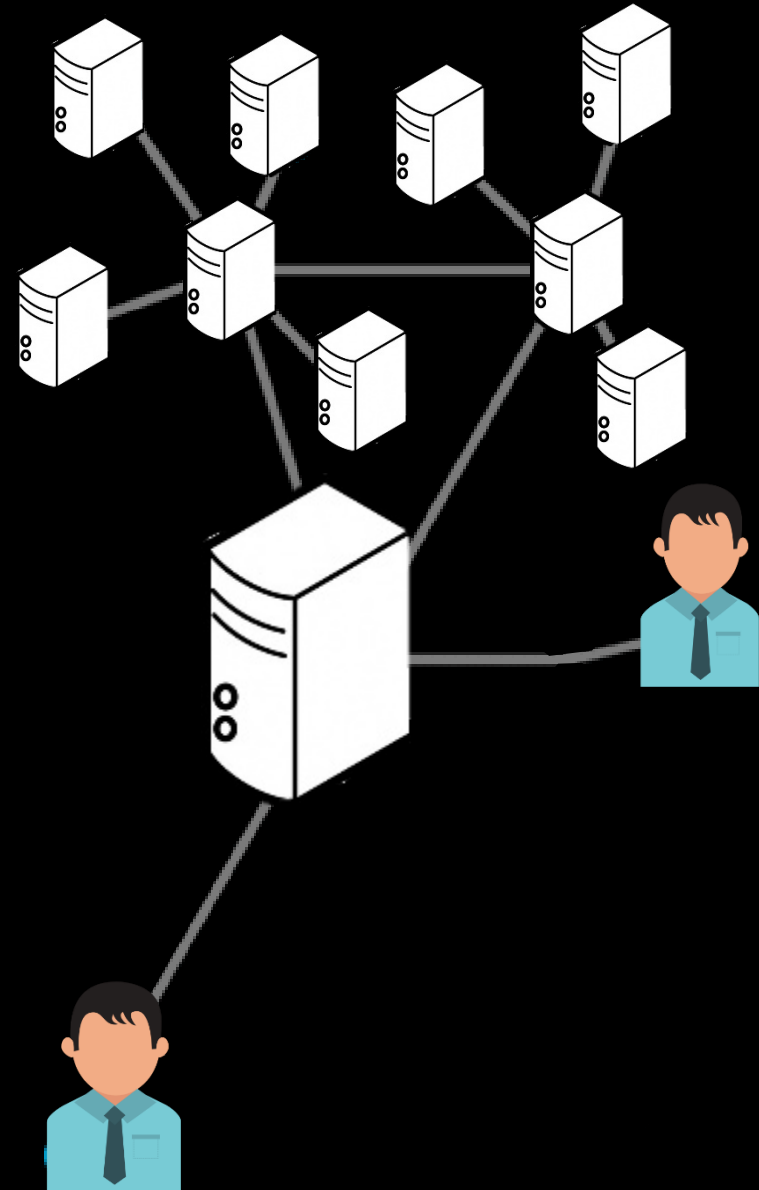
Distributed Computing

- Computing is spread out to multiple computers instead of one
 - Not just limited to MapReduce
 - Intranet, World Wide Web, MMORPGs etc
- Distributed Databases talks about storage
- Distributed Computing talks about computing

Computing



Distributed Computing

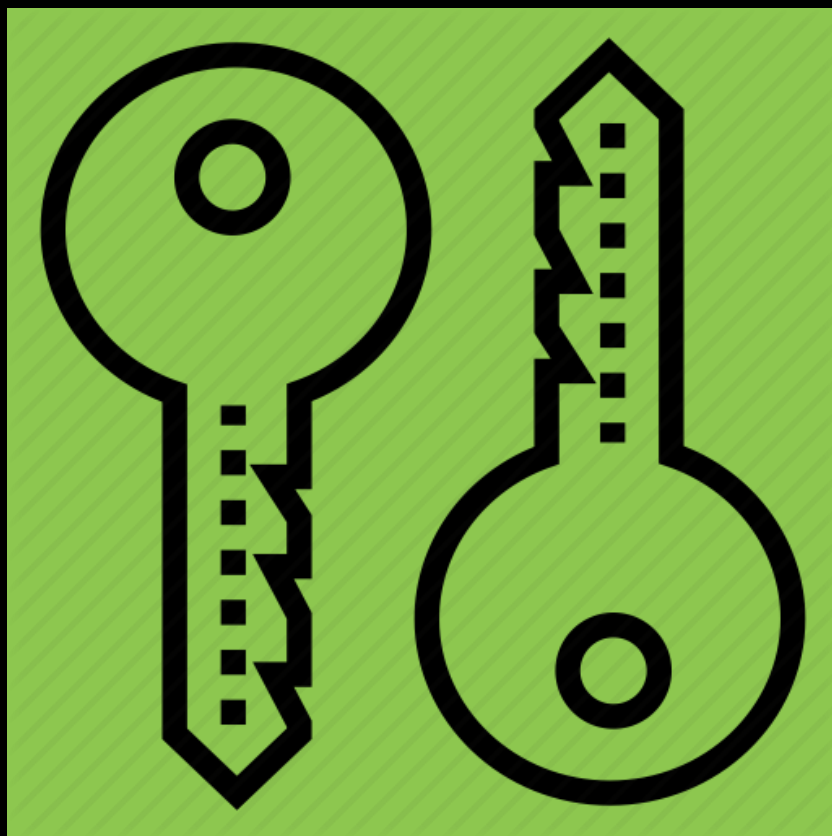


Distributed Computing

- Blockchain (Or Distributed Ledger Technologies) do not rely on just one computer
- Validation of messages/transactions, connection and synchronizing with other nodes is spread across nodes
 - Different nodes may validate different transactions
 - Different nodes may connect to different nodes
- Necessary for different computations to be processed by different computers
 - May be too much for one/every computer to handle

Validation of messages/transactions

- Question:
 - How do messages/transactions get validated?
- Answer:
 - Through Cryptography



Cryptography

Cryptography

- The first half of Crypto-currency
- Primarily consists of two parts:
 - Hashes
 - Encryption

Hashes

- A one-way function
- Takes in an input value, and outputs a value of fixed-length
- Deterministic
 - Same input value → Same output value
- Used to make data unreadable



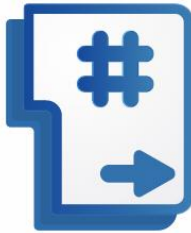
Hashes

Hashing

Plain Text



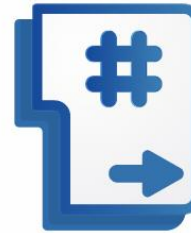
Hash Algorithm



Plain Text



Hash Algorithm

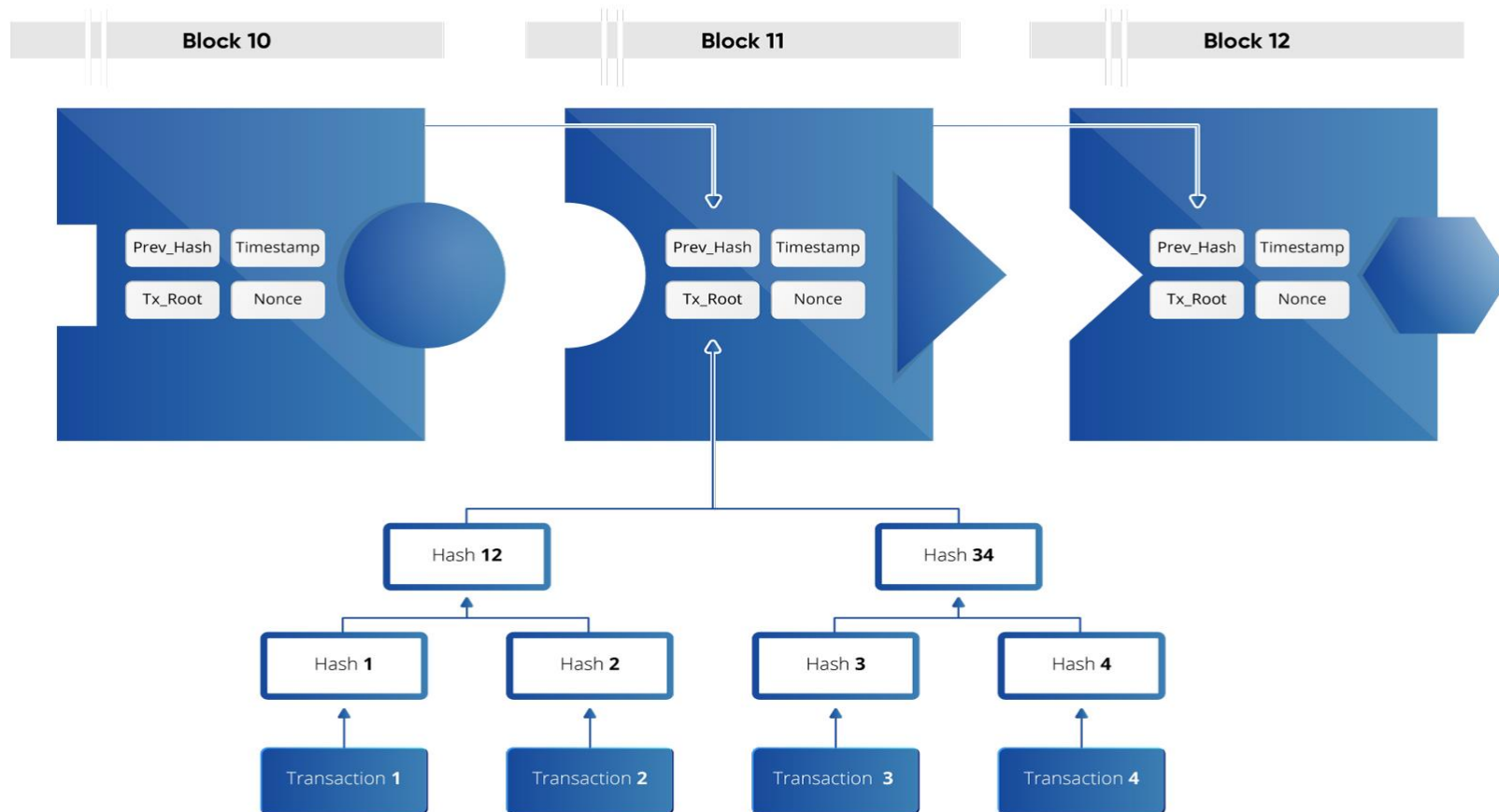


Hashes

- How are Hashes used in Blockchain?
- Unique Identifiers
 - Transaction Hash, Block Hash
- Used to prove data integrity

Hashes

Structure of the blockchain



Encryption

- Signing Transactions
 - Single Signatures
 - Multi Signatures
- Cryptographic Proofs
 - Atomic Swaps
- Anonymity
 - Ring Signatures
 - Blind Signatures

Encryption

I'm dying
already

Encryption

sf43rr3fe

sf43rr3fe

Decryption

I'm dying
already

Encryption

- Public & Private Keys
- Private keys are proofs of identity
 - Encryption – If a message is encrypted with a private key, it proves that the person who owns the private key signed the message
 - Decryption – If a message can only be decrypted by a private key, it is only readable by the person who owns the private key
- Public keys complement private keys
 - Mathematically prove non-repudiation of ownership
 - Non-repudiation → Cannot Deny

Blockchain/DLT Use-cases



Data Integrity
Military records,
information and
communications



IoT/Signal Attestation
Consensus against
fraudulent signals for
Autonomous Vehicles

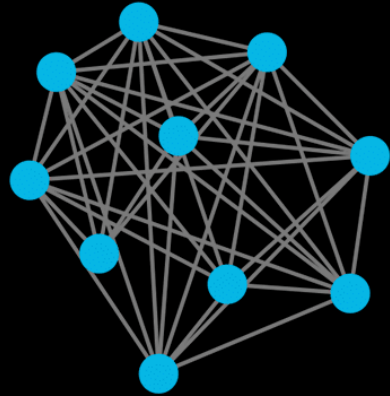


Digital Identity
Prevent citizenship
record and identity
fraud

Lesson structure



**Dapp
Architecture**



**Blockchain
Architecture**



**Dapp
Security**



**Blockchain
Security**




**Dapp
Security**

Integer Underflow/Overflow

- When one puts in a value that exceeds the limit, the value becomes something else.
- Uints have a max value of $2^{256} - 1$, in Solidity it wraps after max value to 0

```
CBlock(hash=000000000790ab3, ver=1, hashPrevBlock=000000000606865, hashMerkleRoot=618eba, pool) {
  nTime=1281891957, nBits=1c00800e, nNonce=28192719, vtx=2)
  CTransaction(hash=012cd8, ver=1, vin.size=1, vout.size=1, nLockTime=0)
    CTxIn(COutPoint(000000, -1), coinbase 040e80001c028f00)
    CTxOut(nValue=50.51000000, scriptPubKey=0x4F4BA55D1580F8C3A8A2C7)
  CTransaction(hash=1d5e51, ver=1, vin.size=1, vout.size=2, nLockTime=0)
    CTxIn(COutPoint(237fe8, 0), scriptSig=0xA87C02384E1F184B79C6AC)
    CTxOut(nValue=92233720368.54275808, scriptPubKey=OP_DUP OP_HASH160 0xB7A7)
    CTxOut(nValue=92233720368.54275808, scriptPubKey=OP_DUP OP_HASH160 0x1512)
  vMerkleTree: 012cd8 1d5e51 618eba
```

 92.2 Billion BTC each!

Block hash: 000000000790ab3f22ec756ad43b6ab569abf0bddeb97c67a6f7b1470a7ec1c

Transaction hash: 1d5e512a9723cbef373b970eb52f1e9598ad67e7408077a82fdac194b65333c9

Timestamp Dependence

- Dangerous to attempt pseudo-random number generation via block timestamps
 - Deterministic, attackers can determine the “random” variable

```
contract CoinFlip {
    uint256 public consecutiveWins;
    uint256 lastHash;
    uint256 FACTOR = 57896044618658097711785492504343953926634992332820282019728792003956564819968;

    function CoinFlip() public {
        consecutiveWins = 0;
    }

    function flip(bool _guess) public returns (bool) {
        uint256 blockValue = uint256(block.blockhash(block.number-1));

        if (lastHash == blockValue) {
            revert();
        }

        lastHash = blockValue;
        uint256 coinFlip = blockValue / FACTOR;
        bool side = coinFlip == 1 ? true : false;
```

Denial of Service – Smart Contract level

- Smart contracts can indicate if they are “payable”
 - Non-payable functions/contracts cannot receive cryptocurrency

```
3 contract Auction {
4     address highestBidder;
5     uint highestBid;
6
7     function bid() {
8         if (msg.value < highestBid) throw;
9
10        if (highestBidder != 0) {
11            highestBidder.transfer(highestBid);
12        }
13
14        highestBidder = msg.sender;
15        highestBid = msg.value;
16    }
17 }
```

Reentrancy Attack

```
function payout() public payable {  
  
    checkPermissions(msg.sender);  
  
    if (game.originator.status == STATUS_TIE && game.taker.status == STATUS_TIE) {  
        game.originator.addr.transfer(game.betAmount);  
        game.taker.addr.transfer(game.betAmount);  
    } else {  
        if (game.originator.status == STATUS_WINNER) {  
            game.originator.addr.transfer(game.betAmount*2);  
        } else if (game.taker.status == STATUS_WINNER) {  
            game.taker.addr.transfer(game.betAmount*2);  
        } else {  
            game.originator.addr.transfer(game.betAmount);  
            game.taker.addr.transfer(game.betAmount);  
        }  
    }  
}  
  
resetGame();  
getBetOutcome();  
}
```

When a payment is made, it calls the fallback function of the payee contract

The fallback function can call the payment function again recursively

Phishing using tx.origin

```
interface Wallet {  
    function transferTo(address to, uint amount);  
}  
contract Exploit {  
    address owner;  
    constructor() public {  
        owner = msg.sender; {  
    }  
    function getOwner() public returns (address) {  
        return owner;  
    }  
    function() payable public {  
        Wallet(msg.sender).transferTo(owner, msg.sender.balance);  
    }  
}
```

Parity Wallet attacks

- Bad coding left them vulnerable to attacks on two occasions

```
contract WalletLibrary is WalletEvents {
    function initWallet(address[] _owners, uint _required, uint _daylimit) only_uninitialized {
        initDaylimit(_daylimit);
        initMultiowned(_owners, _required);
    }
    function execute(address _to, uint _value, bytes _data) external onlyowner returns (bytes32 o_hash) {
        //sends money to the address
    }
}

contract Wallet is WalletEvents {

    function Wallet(address[] _owners, uint _required, uint _daylimit) {
        function() payable {
            // just being sent some cash?
            if (msg.value > 0)
                Deposit(msg.sender, msg.value);
            else if (msg.data.length > 0)
                _walletLibrary.delegatecall(msg.data);
        }
    }
}
```


Short Address Attack

- The fault here lies in the way EVM handles underflows, it pads it with 0s

[illegible]

```
contract MyToken {
    mapping (address => uint) balances;

    event Transfer(address indexed _from, address indexed _to, uint256 _value);

    function MyToken() {
        balances[tx.origin] = 10000;
    }
    // 0x62bec9abe373123b9b635b75608f94eb86441630
    function sendCoin(address to, uint amount) returns(bool sufficient) {
        if (balances[msg.sender] < amount) return false;
        balances[msg.sender] -= amount;
        balances[to] += amount;
        Transfer(msg.sender, to, amount);
        return true;
    }

    function getBalance(address addr) constant returns(uint) {
        return balances[addr];
    }
}
```

2<<4 = 32 coins
= 30 new coins

Honeypots

```
contract MultiplierX3 {
    address public Owner = msg.sender;

    function() public payable{}

    function withdraw() payable public{
        require(msg.sender == Owner);
        Owner.transfer(this.balance);
    }

    function Command(address adr,bytes data) payable public{
        require(msg.sender == Owner);
        adr.call.value(msg.value)(data);
    }

    function multiply(address adr) public payable{
        if(msg.value>=this.balance){
            adr.transfer(this.balance+msg.value);
        }
    }
}
```

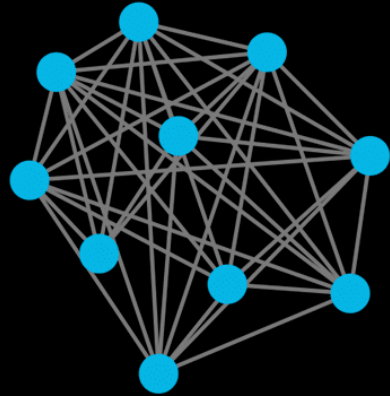
this.balance is updated before the multiply method is called

Hence, (msg.value>=this.balance) will always be false and transfer will not run (unless initial balance is 0)

Lesson structure



**Dapp
Architecture**



**Blockchain
Architecture**



**Dapp
Security**



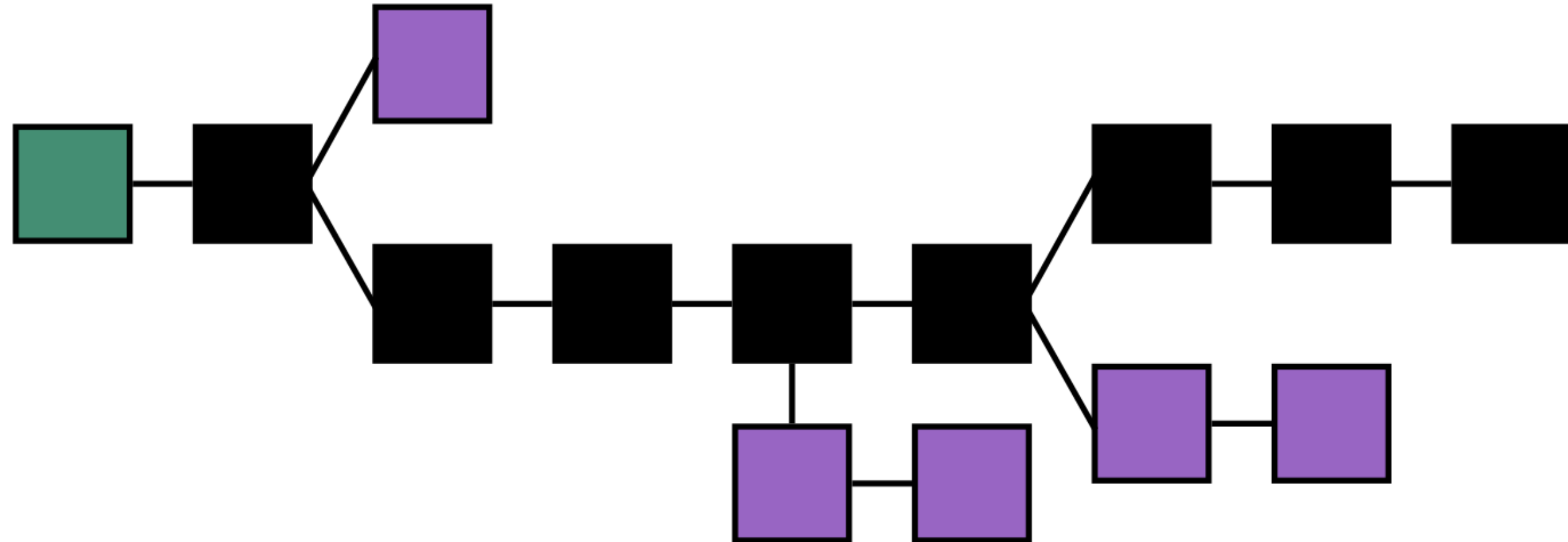
**Blockchain
Security**



**Blockchain
Security**

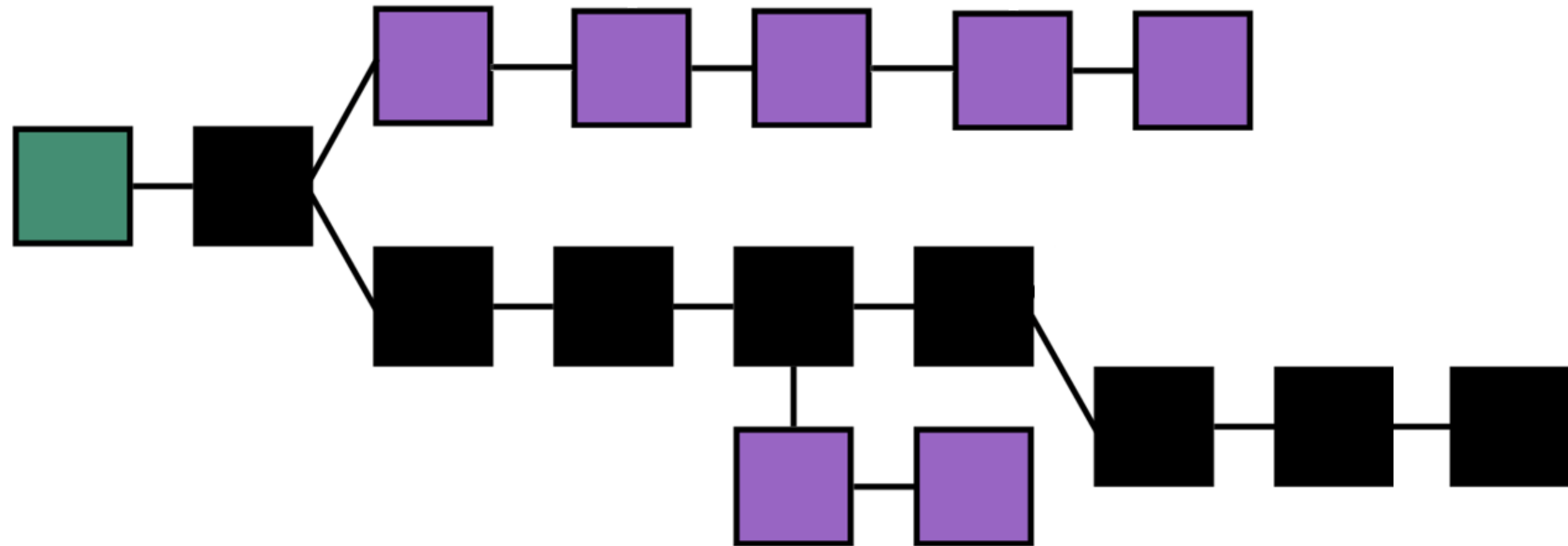
51% Attack (Proof of Work)

- Blocks are validated through mining, which is dependent on hashing power



Long Range Attack (Proof of Stake)

- Blocks are validated through validators, which is dependent on wealth (amount staked)



Transaction (sig) Malleability

- Broadcasting the same transaction signature with a modified transaction hash
 - Sometimes, transaction signatures do not encompass all data in the transaction
 - Possible to mangle transaction hash without invalidating transaction signature
 - Requires broadcast before confirmation of transaction into a valid block
- Possible to mangle transaction hashes by
 - Modifying minor details (whitespaces or paddings)
 - Using complementary signatures of certain cryptographic signature schemes (ECDSA)
 - Major transaction details like recipients and value remains unchanged
- This transaction, if confirmed over the other original, will trick the sender to think the transaction failed when it actually succeeded
 - Senders query using their transaction hashes
 - Which is invalidated due to the modified, fraudulent transaction
- Can lead to double-spending

Denial of Service – System level

- EOS RAM Hijack
- EOS uses RAM to execute and store state of smart contracts
- Smart contracts can notify others about specific events, such as token transfer

```
void apply_context::update_db_usage( const account_name& payer, int64_t delta ) {  
    if( delta > 0 ) {  
        if( !(privileged || payer == account_name(receiver)) ) {  
            require_authorization( payer );  
        }  
    }  
    trx_context.add_ram_usage(payer, delta);  
}  
  
void apply_context::require_authorization( const account_name& account ) {  
    for( uint32_t i=0; i < act.authorization.size(); i++ ) {  
        if( act.authorization[i].actor == account ) {  
            used_authorizations[i] = true;  
            return;  
        }  
    }  
    EOS_ASSERT( false, missing_auth_exception, "missing authority of ${account}", ("account",account)  
}
```

If recipient is a contract, does not check if action is authorized by action handler → RAM consumed

Can potentially fill action handler's RAM with garbage notifications, thus using up all bandwidth to process contracts

BFT Consensus algorithms forking

- In BFT type algorithms, Blocks require 66% consensus on transaction validity
- NEO Blockchain
 - Initially used a 2-phase protocol for efficiency and lower complexity
 - Missed the “commit” phase of traditional BFT algos
- Uniformity of message “Proposed blocks” is not agreed upon
 - Led to different blocks formed on the blockchain → Fork
- Caused ensuing blocks to record different previous hashes
 - Consensus cannot be agreed upon and stalled

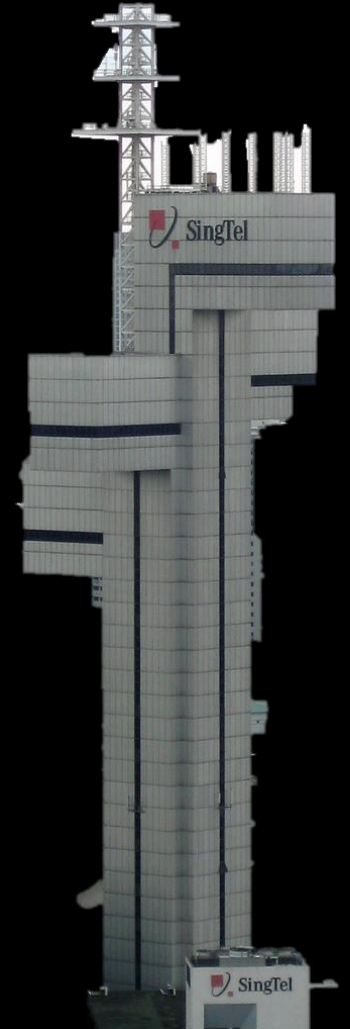
TEE-based consensus

- Proof of Elapsed Time (Intel's Hyperledger Sawtooth)
- Based on Intel Software Guard eXtensions (Intel SGX)
- Utilises Trusted Execution Environments to generate randomness
 - Data is protected from even malicious or hacked kernels
- Speculative Execution
 - Microarchitecturally, processor might speculatively guess values from memory
 - Faulty guesses disturbs other parts of the processor like the cache contents
 - Speculative Execution detects and measures such disturbances to infer in-memory values
- Meltdown, Spectre, Foreshadow

- If the Smart contract and Blockchain infrastructure are secure
 - Is there no other way we can attack the blockchain?



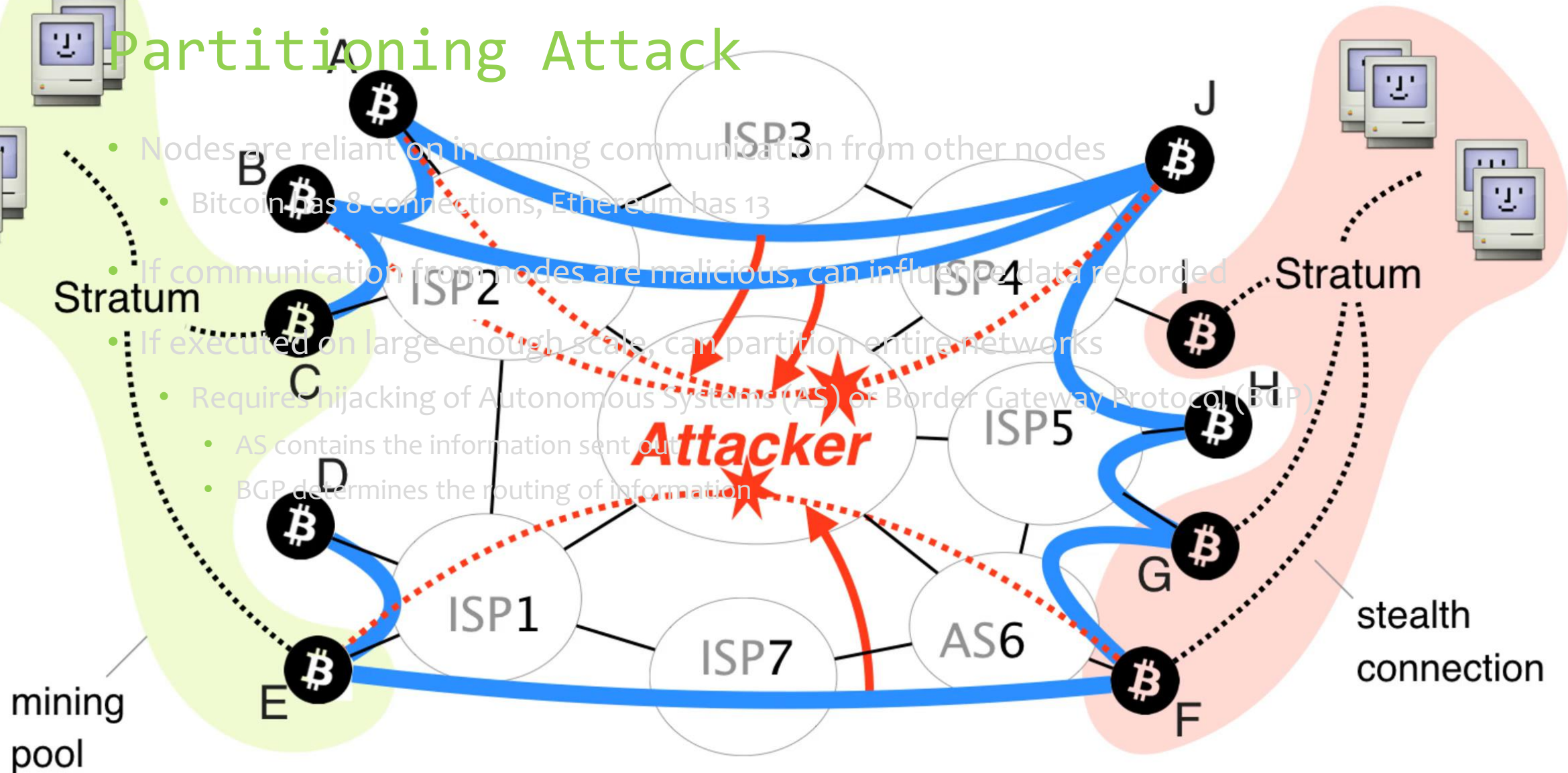
- Answer: Blockchain ultimately relies on networks to communicate and gain consensus
 - That is our attack vector

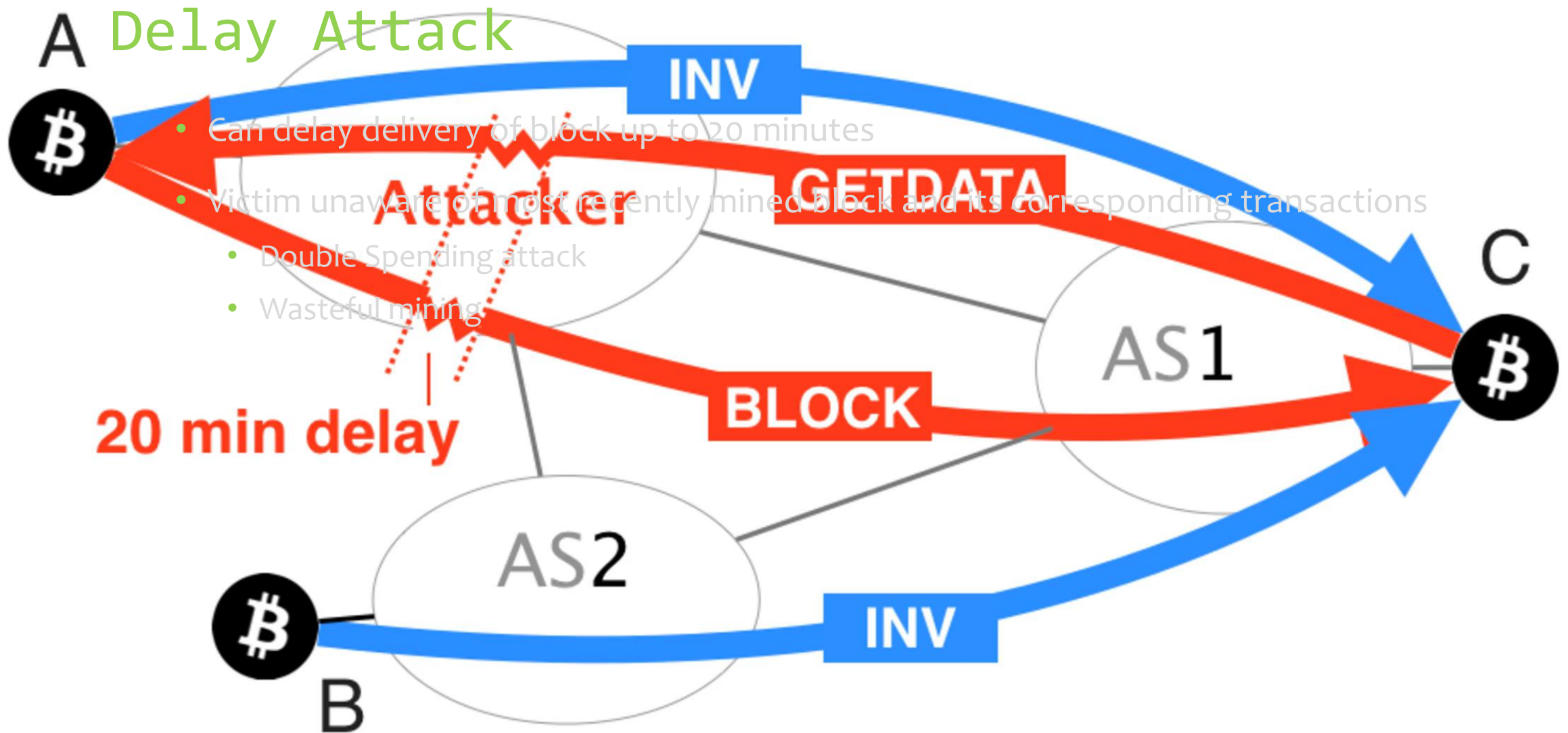


miners

Partitioning Attack

- Nodes are reliant on incoming communication from other nodes
- Bitcoin has 8 connections, Ethereum has 13
- If communication from nodes are malicious, can influence data recorded
- If executed on large enough scale, can partition entire networks
- Requires hijacking of Autonomous Systems (AS) or Border Gateway Protocol (BGP)
- AS contains the information sent out
- BGP determines the routing of information







Questions?

Just ask don't shy