COMP6452 Lecture 2: Existing Blockchain Platforms

Ingo Weber | Principal Research Scientist & Team Leader
Architecture & Analytics Platforms (AAP) team
ingo.weber@data61.csiro.au

Conj. Assoc. Professor, UNSW Australia | Adj. Assoc. Professor, Swinburne University

www.data61.csiro.au

Agenda:

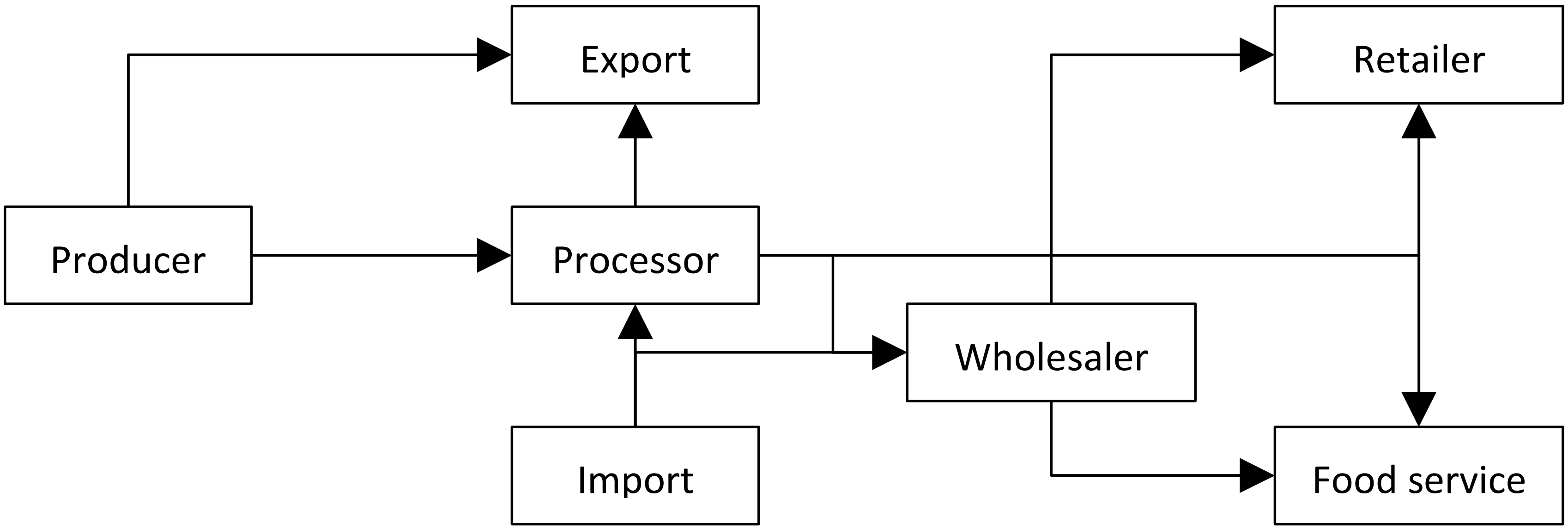
- Use cases
- Cryptography basics
- Bitcoin
- Ethereum
- Smart Contract Development

Agenda:

- Use cases
- Cryptography basics
- Bitcoin
- Ethereum
- Smart Contract Development

Use Case: Supply Chains

Sample Agricultural Supply Chain Network



Using Blockchain for Supply Chains

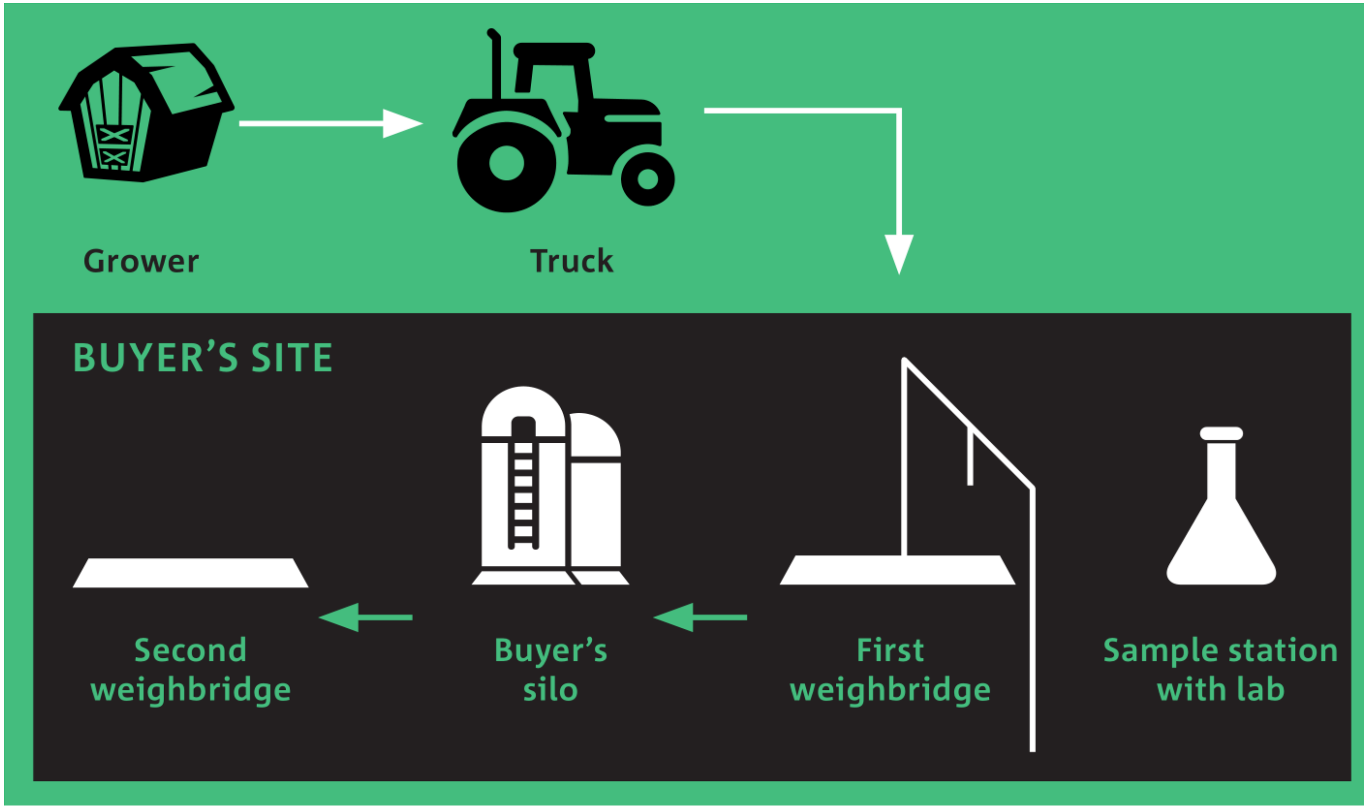
- Why?
  - Irrefutable, tamper-proof data store
    - Prevent or detect counterfeiting
  - Smart contracts can check integrity and authorization / authentication
  - Can solve other problems:
    - Counter-party risks
    - Lack of trust, e.g., in coopetition
    - Supply chain transparency

    ...

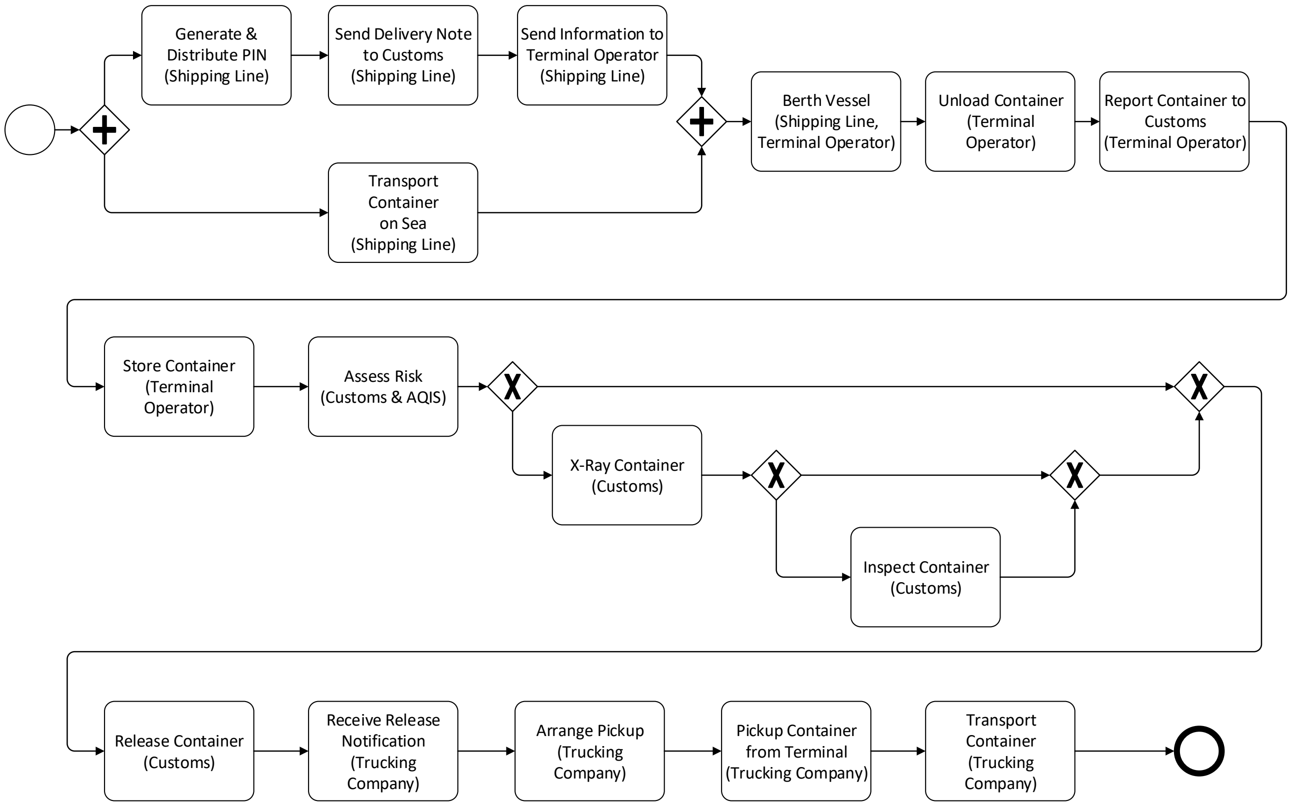
- How?
  - Record supply chain events on blockchain, e.g.:
    - GS1 EPCIS events
    - Other tag scans
    - Phytosanitary certificates
  - Check that event sequences are correct, e.g. through
    1. Process conformance
    2. Business rules adherence
    - Can be on-chain or off-chain
    - Regulatory compliance

Example: AgriDigital's first pilot

see https://www.data61.csiro.au/blockchain / Chapter 12 in the book



Example: Sea Import to Australia



Some Benefits of using Blockchain in Supply Chains

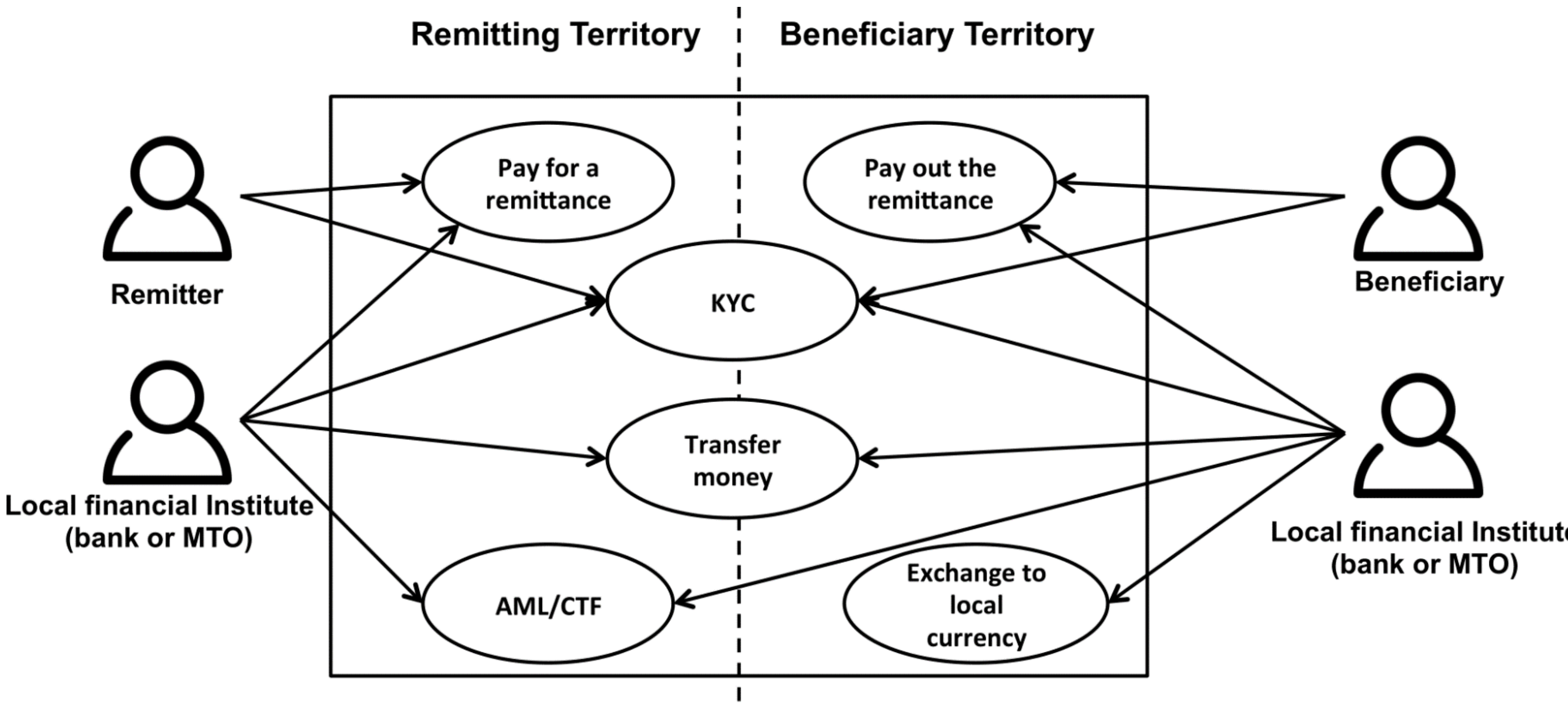
- Electronic titles to supply chain goods
  - Ensure ownership, right to sell, etc are handled correctly
  - Reduce financial risk – e.g.: if a buyer goes bankrupt before paying for the goods, the seller still owns them
- Establish identity and authenticity for:
  - Requester
  - Other relevant supply chain participants
- Check financial record / trustworthiness
- Ensure correctness of specific supply chain documents
  - E.g., invoice, purchase order, ...

Use Case: International Money Transfers

- Many workers in Australia regularly send money back to their families overseas
  - Up to 10% of GDP in some developing countries (and even 27% in Tonga and 20% in Samoa)
  - High remittance costs have serious effects in these countries
- Remittance costs in Pacific Island countries are among the highest in the world
  - For example, to send $200 from Australia to Vanuatu costs $33.20 and $28.60 to Samoa
- Issues:
  - Many parties involved, sometimes little transparency
  - Difficulties of satisfying AML/CTF (Anti-Money Laundering/Counter-Terrorism Financing) regulation, especially where the receiving party may not have a bank account.
  - High latency, with transaction times up to 5days.

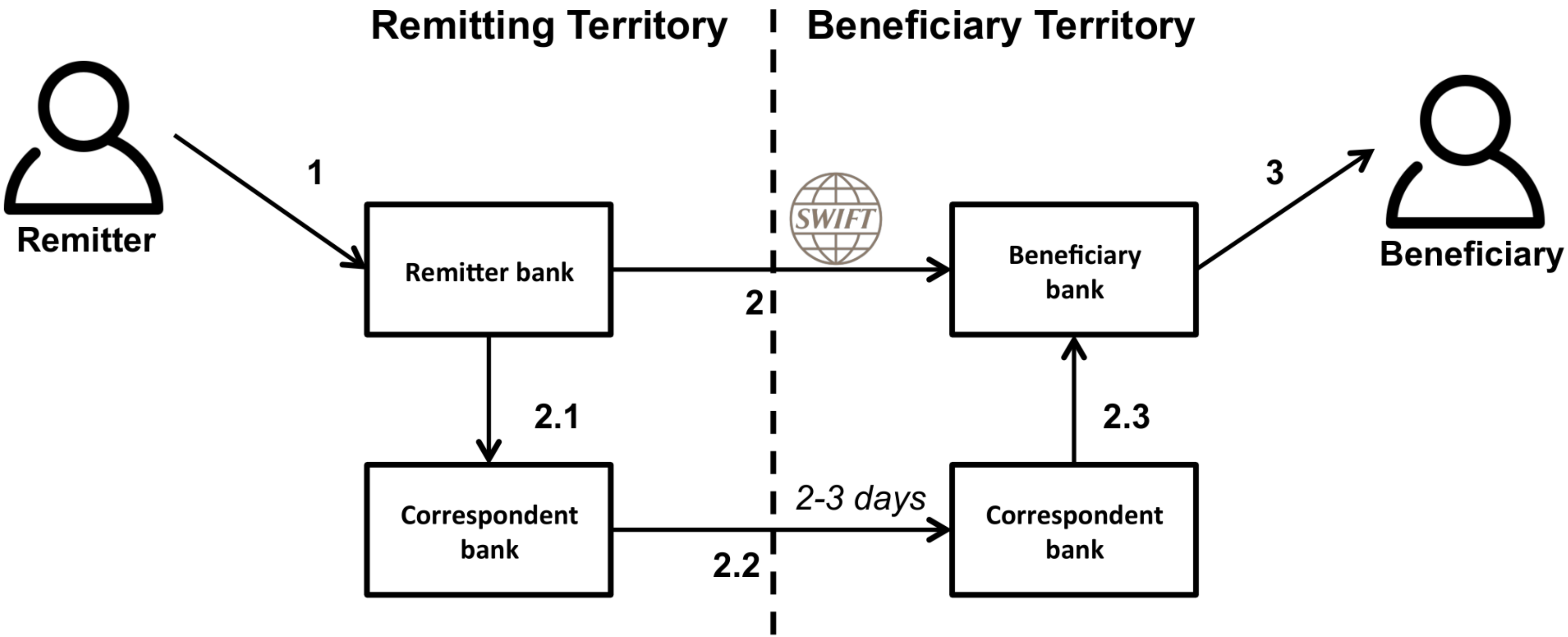
Use Case: International Money Transfers

Stakeholders and Functions



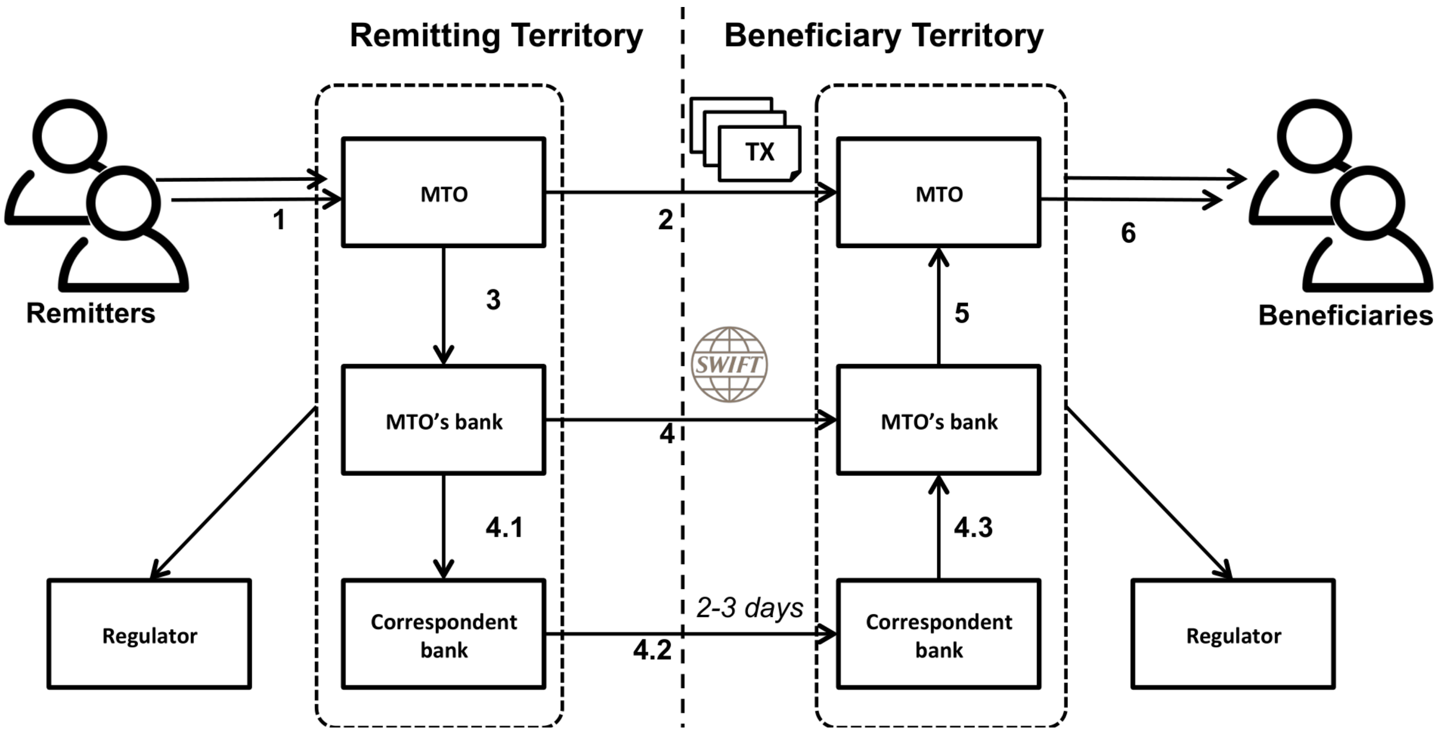
Use Case: International Money Transfers

Remittance through banks



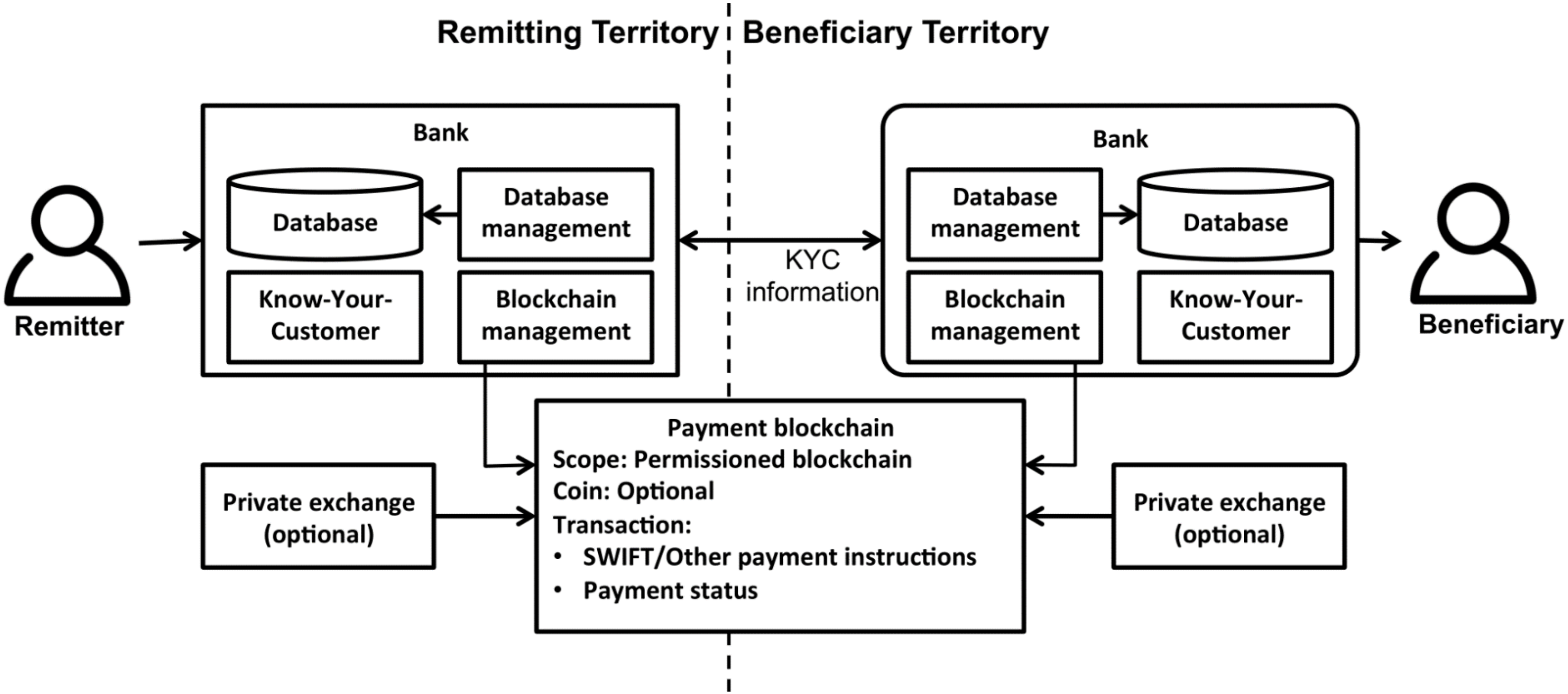
Use Case: International Money Transfers

Remittance through a Money Transfer Operator (MTO)



Use Case: International Money Transfers

Remittance through Blockchain



Alternatively, use existing cryptocurrencies like Bitcoin.

Intl. Money Transfers Non-functional Properties

- Transaction Latency:
  - From days (conventional) to hours or minutes (with blockchain)
- Cost:
  - Depends on the fees charged by various parties – but more parties are involved in the conventional designs
- Transparency:
  - Greater in the blockchain setting; foreign exchange rates might still be unfavourable for the customers
- Barriers to Entry
  - Conventional design requires participants to have banking / financial services licenses, and business relationships with correspondent banks
  - Public blockchains have low barriers to entry, but local regulation still applies to end-points within countries

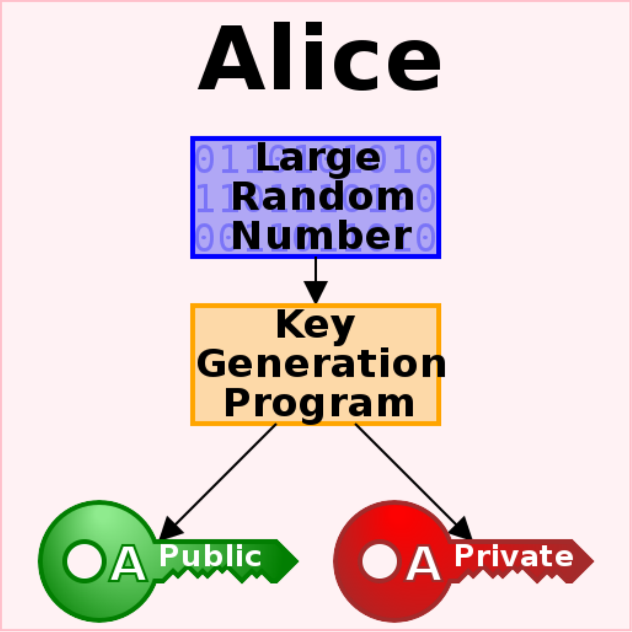
Agenda:

- Use cases
- Cryptography basics
- Bitcoin
- Ethereum
- Smart Contract Development

Cryptography basics: public-key cryptography (1)

Overview

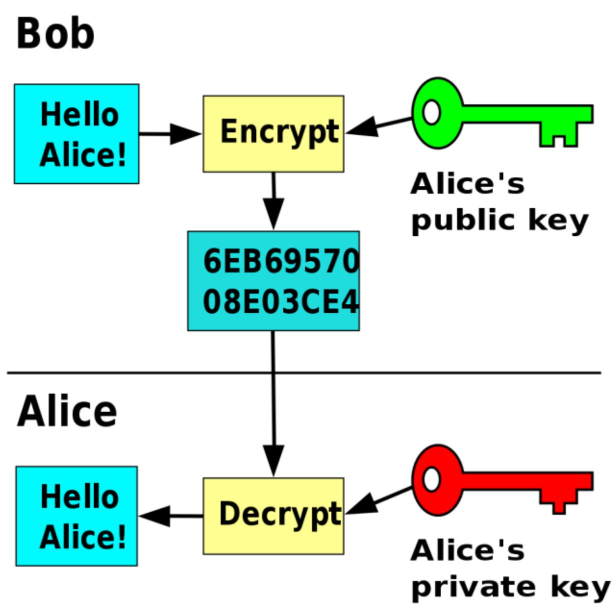
- Public-key cryptography, or asymmetric cryptography, is a cryptographic system that uses *pairs of keys*:
  - public keys which may be disseminated widely;
  - private keys which are known only to the owner.
- Effective security only requires keeping the private key private
- Easy to create new key pairs
- Used heavily in blockchain
  - Losing your private key can mean loss of assets
  - If hackers can get your private key, they can steal your assets

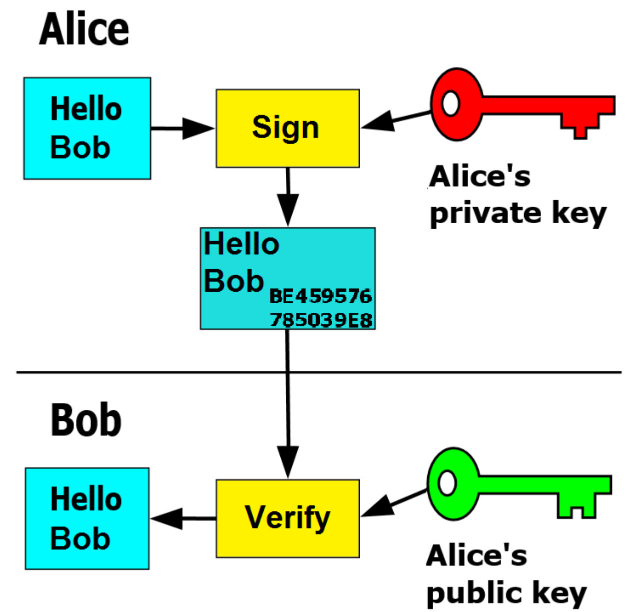


Some content from https://en.wikipedia.org/wiki/Public-key_cryptography

Cryptography basics: public-key cryptography (2)

Encryption and digital signatures





Only Alice can decrypt the message

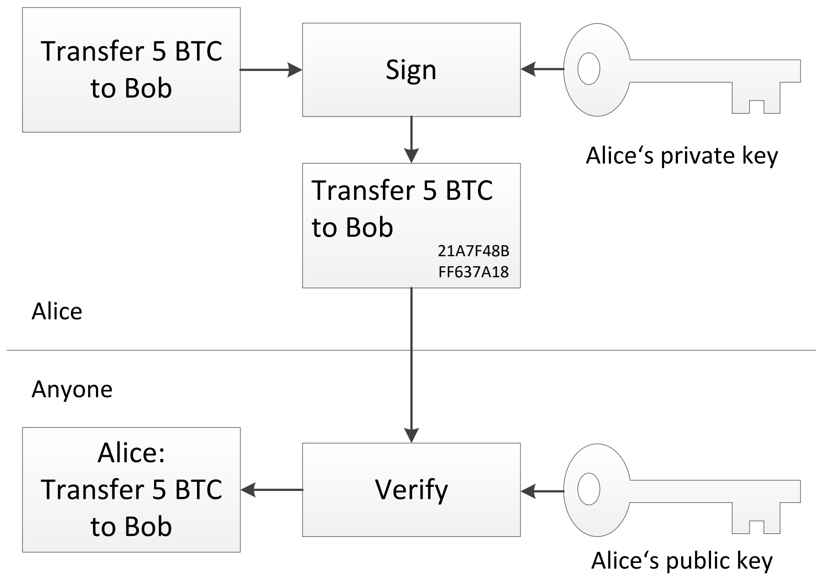
No-one can change the message without breaking Alice‘s signature

Some content from https://en.wikipedia.org/wiki/Public-key_cryptography

Cryptography basics: public-key cryptography (3)

Use in Blockchain

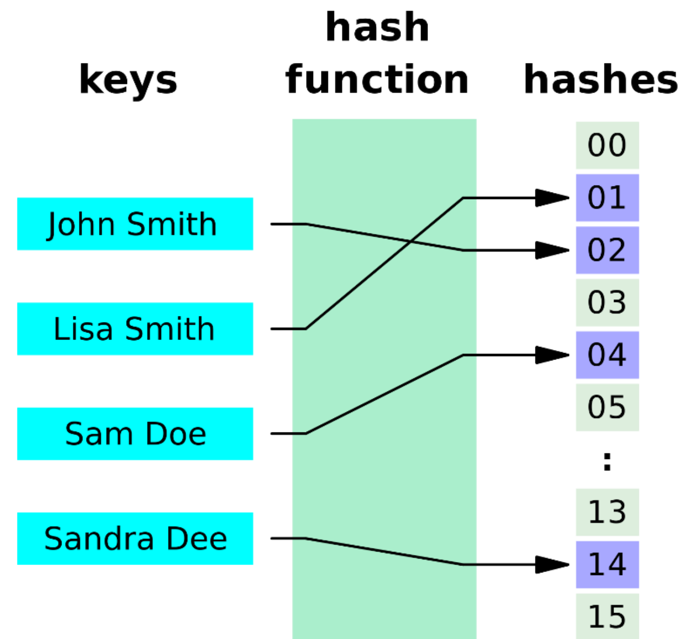
- Control over accounts: by public key
  - Control means the ability to act on behalf of the account, like spending the assets it owns
- In fact: each account is known by its public key
  - "Alice" here is really 0x7a2f16dab8b5c2cf99c35e4c6a5beb45c7df8f87
  - For some accounts, we may know the person / organization owning it
  - But by default, we don't



Some content from https://en.wikipedia.org/wiki/Public-key_cryptography

Cryptography basics: hash functions (1)

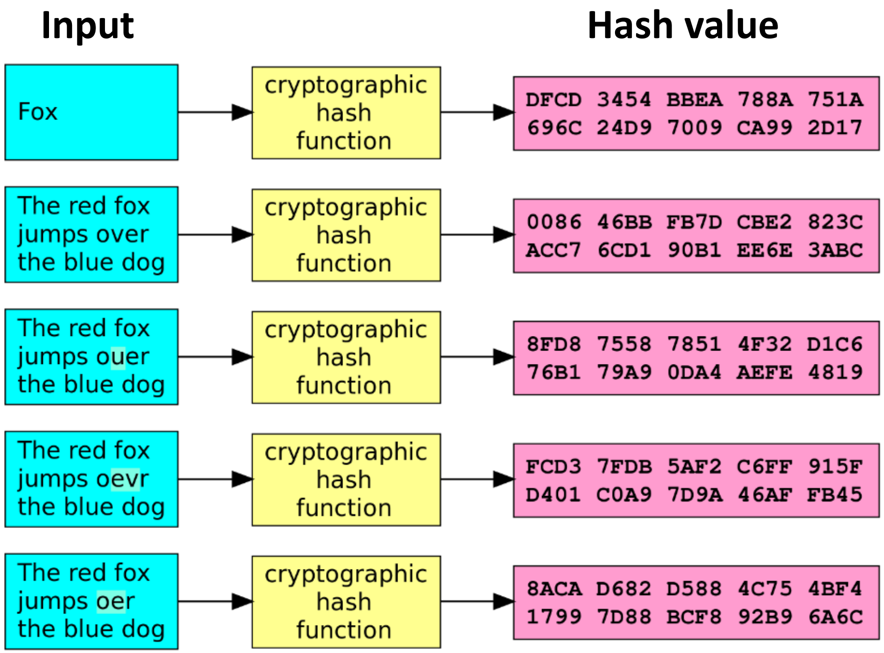
- A hash function is any function that can be used to map data of arbitrary size to data of a fixed size.
  - Values returned by a hash function are called hash values, hash codes, digests, or simply hashes.



Some content from https://en.wikipedia.org/wiki/Hash_function and https://en.wikipedia.org/wiki/Cryptographic_hash_function

Cryptography basics: hash functions (2)

- A **cryptographic hash function** is a special class of hash function with certain properties:
  - a one-way function: a function which is infeasible to invert.
  - deterministic: the same message always results in the same hash
  - quick to compute hash value for any message
  - it is infeasible to generate a message from its hash value except by trying all possible messages
  - a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value
  - it is infeasible to find two different messages with the same hash value



Some content from https://en.wikipedia.org/wiki/Hash_function and https://en.wikipedia.org/wiki/Cryptographic_hash_function

Agenda:

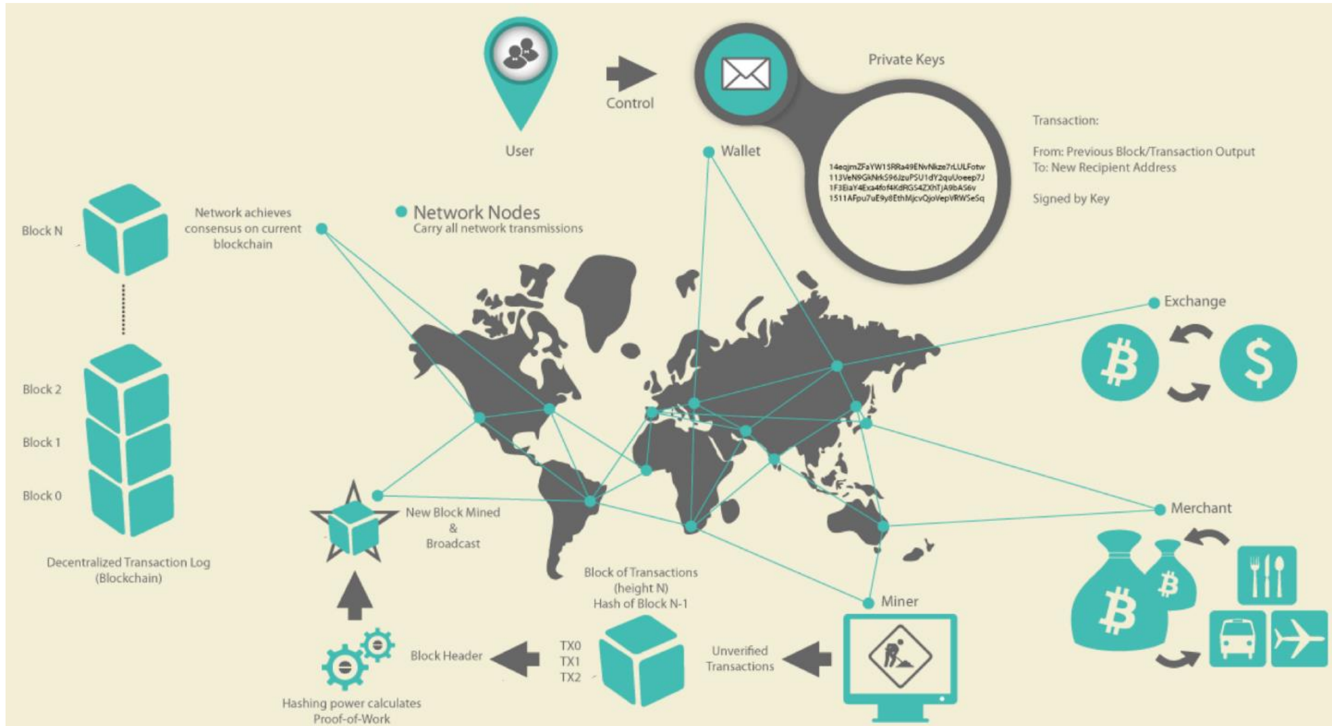
- Use cases
- Cryptography basics
- Bitcoin
- Ethereum
- Smart Contract Development

Bitcoin

First Cryptocurrency

Inventor: Satoshi Nakamoto (American? European? Denis Wright? Hal Finney?)

- A cryptocurrency operated on a p2p blockchain network
- 3 main types of participants
  - Users with wallets which contain keys to authenticate the transaction sent by the user using digital signature
  - Miners for producing blocks which store the validated transactions
  - Exchanges where users can trade bitcoin with other currencies



Source: Andreas M.Antonopoulos, Mastering Bitcoin-Unlocking Digital Cryptocurrencies

Bitcoin

Network Distribution

GLOBAL BITCOIN NODES DISTRIBUTION

Reachable nodes as of Thu Nov 22 2018 11:36:15 GMT+1100 (Australian Eastern Daylight Time).

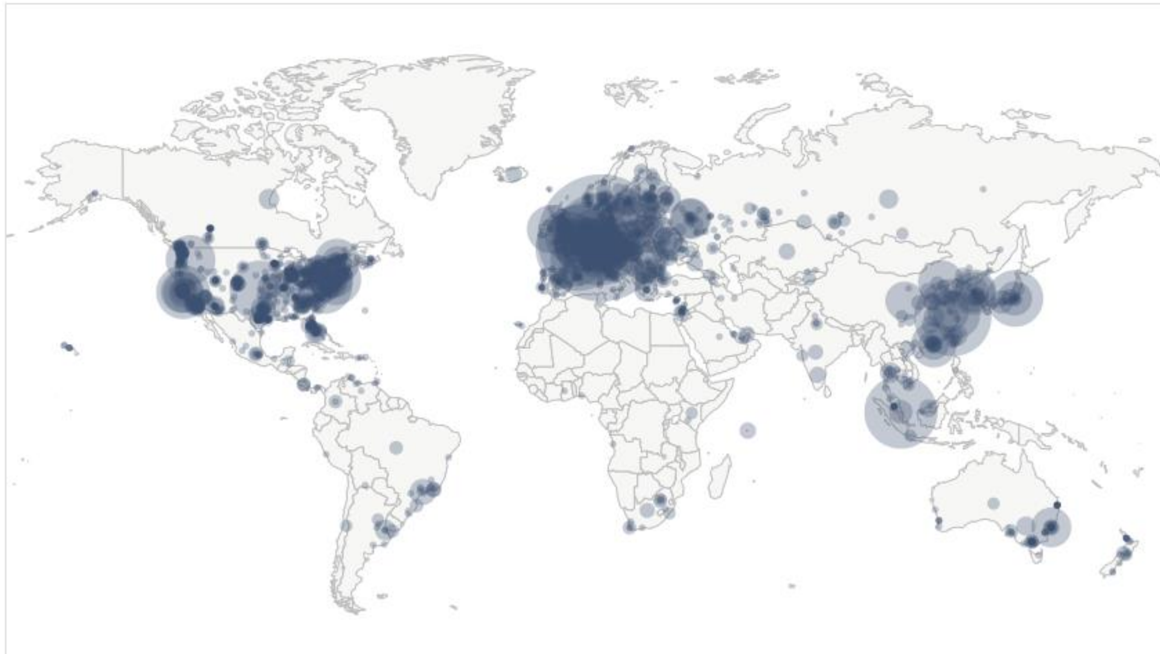
10168 NODES

24-hour charts »

Top 10 countries with their respective number of reachable nodes are as follow.

RANK	COUNTRY	NODES
1	United States	2375 (23.36%)
2	Germany	1936 (19.04%)
3	France	667 (6.56%)
4	China	630 (6.20%)
5	Netherlands	493 (4.85%)
6	n/a	482 (4.74%)
7	Canada	367 (3.61%)
8	United Kingdom	331 (3.26%)
9	Singapore	266 (2.62%)
10	Russian Federation	259 (2.55%)

More (102) »



Source: bitnodes.21.co/

Bitcoin

Exchange rate to US$

- Lowest: $0.05
- Highest: ~$20,000
- Now: below ~$4000
- Not stable

Source: 99bitcoins.com/price-chart-history/

Bitcoin

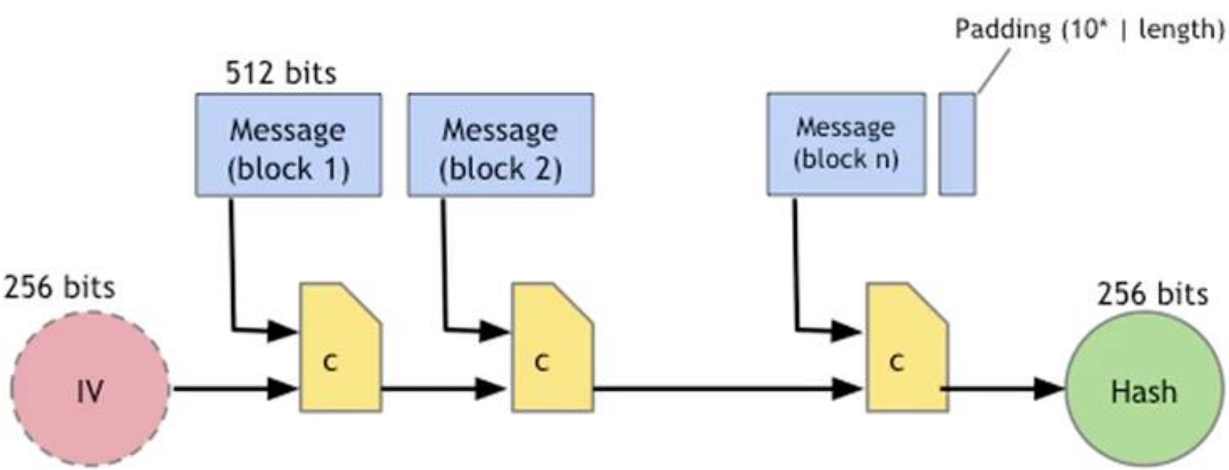
Deflationary Cryptocurrency

- Total supply: 21 million
- New BTCs are issued during the mining process
  - New BTCs are rewarded to the miner who create the valid block.
  - The reward is halving every 210,000 blocks.
  - Reward (since 2016): 12.5 BTC (initially, 50 BTC)
  - Minting in Bitcoin will run out in 2140, when no more new BTC being issued.

Bitcoin

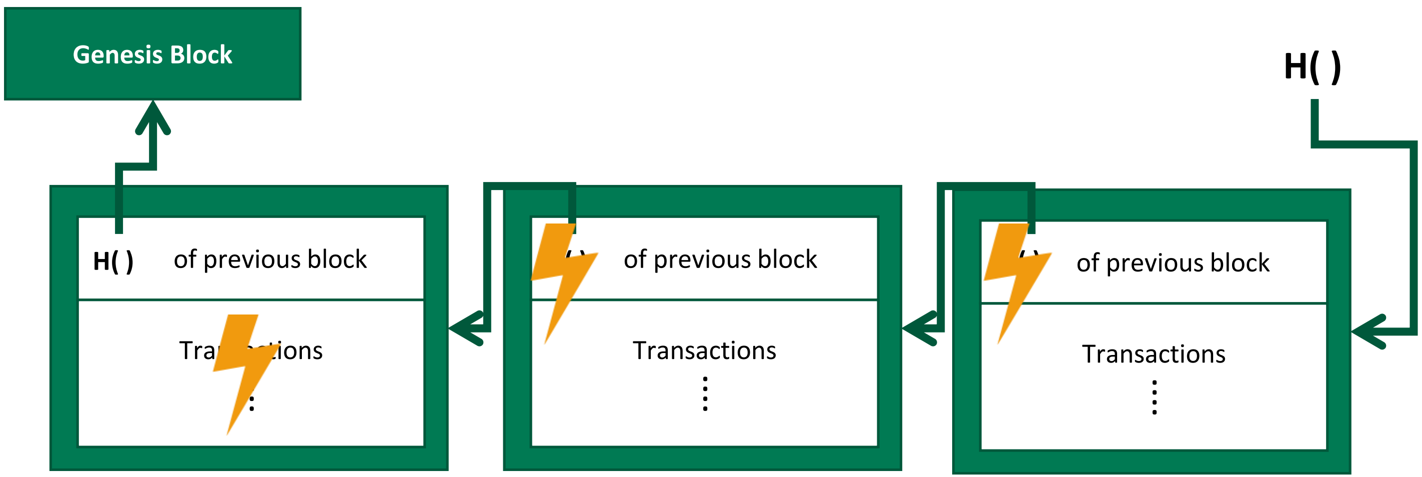
SHA-256 Hash Function

- Hash function
  - Map data of arbitrary size to data of fixed size
  - One-way function –easy to compute but hard to invert –collision free



Bitcoin

Linked list with hash pointer

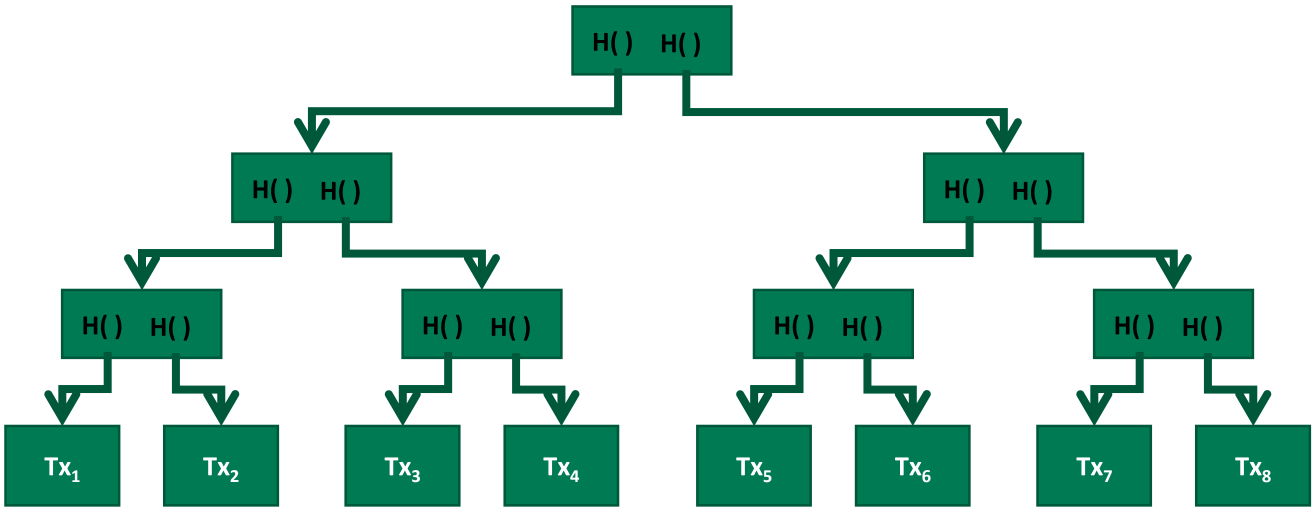


- Time between blocks, called *inter-block time*, is on average 10 minutes
  - But variation of times is high

Bitcoin

Block – Container of Transactions

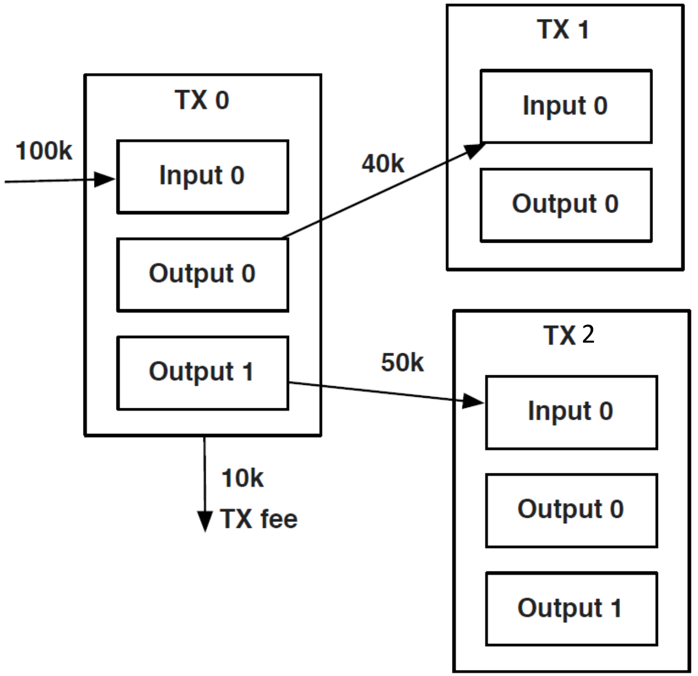
- Merkle Tree: Binary tree built using the hash function



Bitcoin

Transactions

- Transfer currency from source addresses to destination addresses
- Contains one or more inputs and one or more outputs
  - If the sum of the outputs is less than the sum of the inputs, the difference is a fee to the miner
    - Transaction fee is an incentive for miners to contribute computing power
    - The only variable that client software ask users to choose when create a transaction
- Contains proof of ownership for each input, in the form of a digital signature from the owner

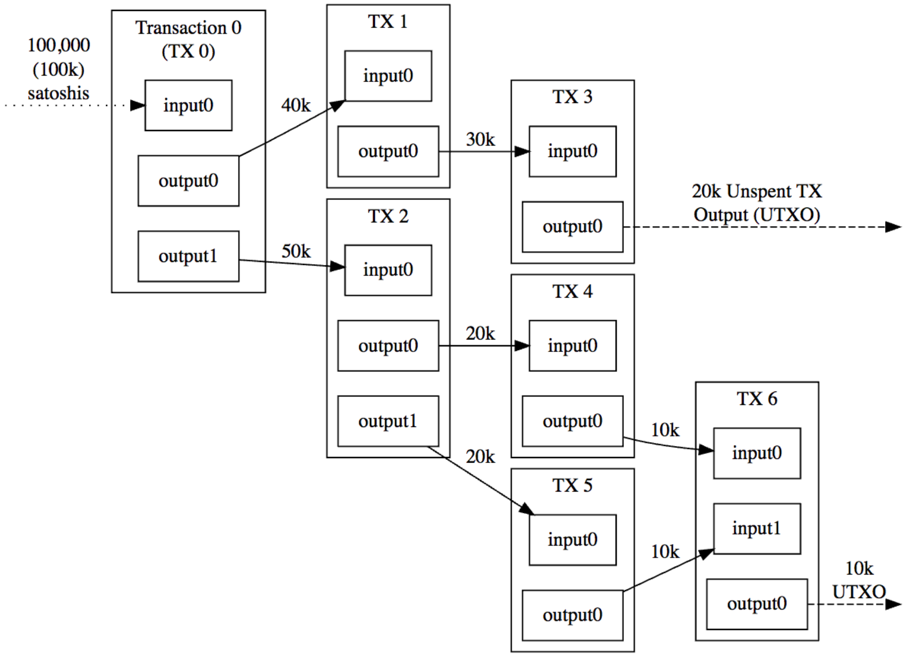


How transactions are linked

Bitcoin

Transactions

- Linked Transactions
  - Outputs of transactions become inputs of new transactions
  - Bitcoin addresses don't contain "coins"
    - Store unspent transaction outputs (UTXO)
  - "Address balance": the sum of all of the values of UTXOs associated with the address
    - Bit"coin" is misleading, because fractional ownership (e.g., 1.64 BTC) is the norm



Bitcoin

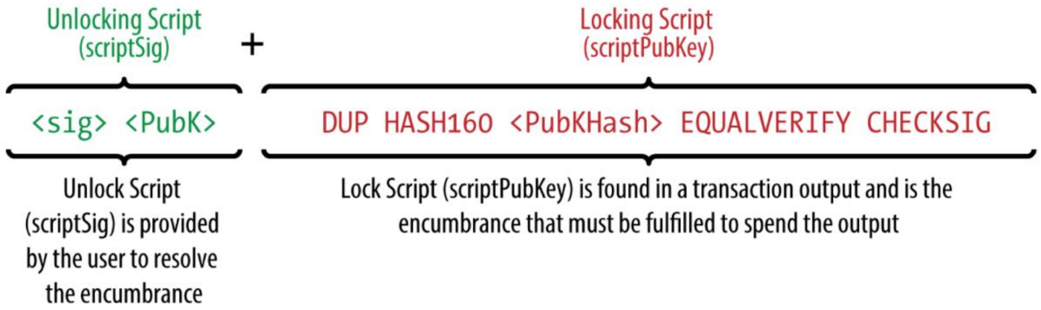
Transactions

- Transactions delay factors
  - Transaction fee
    - Miners tend to optimize block creation by preferring transactions with higher fees
  - Transaction announcement happens during mining of next block
    - Mining will not be restarted, so new transactions can only be included in the block after the next one → theoretical inclusion delay is 1.5x inter-block time
  - Orphan
    - Transactions must arrive in order for a miner to process them fast
    - If the referenced input transactions, called parents, are as-yet unknown, a miner will delay the inclusion of the new transaction — it is then a so-called orphan
    - Miners may choose to keep orphans in the mempool while waiting for the parent transactions to arrive, but they also remove orphans after a time-out they choose
  - Locktimes
    - A transaction can contain a parameter declaring it invalid until the block with a certain sequence number has been mined

Bitcoin

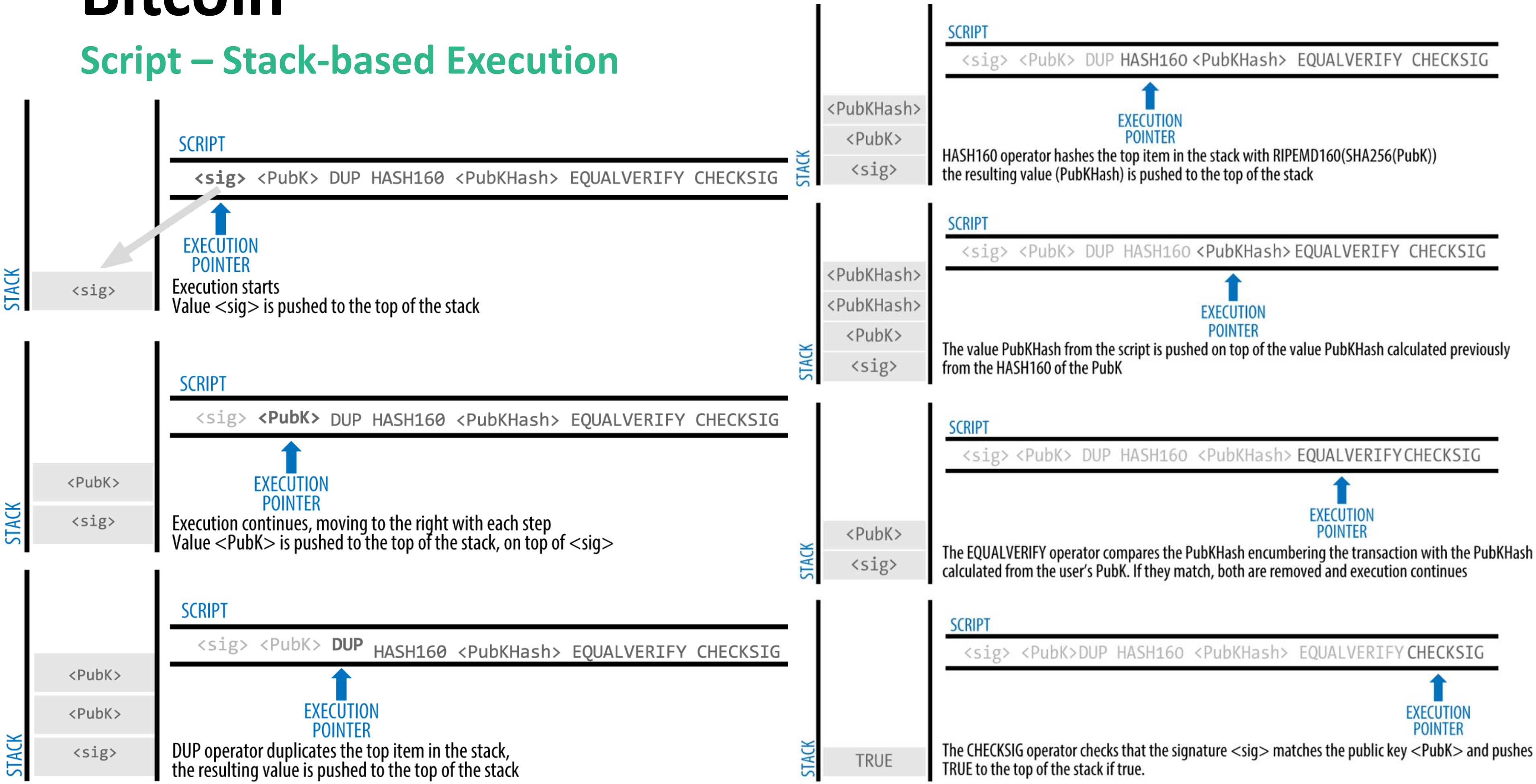
Script

- Bitcoin uses a scripting system for transactions
  - Transaction validation relies on two types of script
    - A locking script in an output specifies the conditions for spending the BTC
    - An unlocking script in an input satisfies the conditions of the locking script
    - The unlocking script and the locking script are combined and executed
      - If the result is true, the transaction is valid
    - Script keywords are called *opcode*
    - Pay-to-PubKey-Hash(P2PKH): the most common opcode implements a simple transfer



Bitcoin

Script – Stack-based Execution



Bitcoin

Script

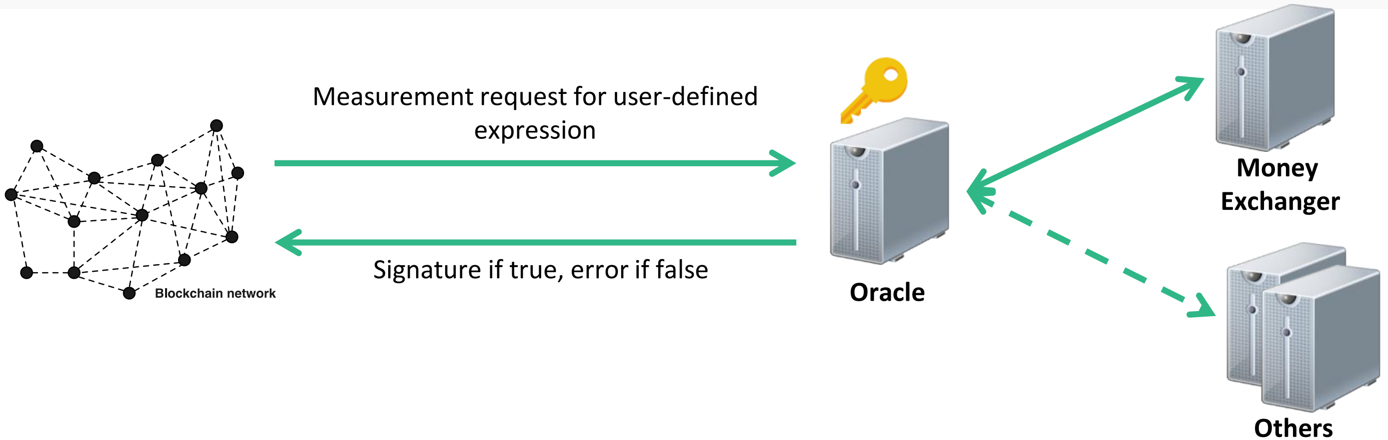
- Provides certain flexibility to change the parameters of the conditions to spend the transferred BTC
  - Multi-signature: A transaction can require multiple keys and signatures
  - OP_RETURN
    - An *opcode* used to mark a transaction output as invalid
    - Has been used as a standard way to embed arbitrary data to the Bitcoin blockchain for various purposes, like representing assets, e.g. diamonds

Bitcoin

Script

- Script programs are pure functions, which cannot import any external state
- An oracle can be used to include external state into the blockchain execution environment

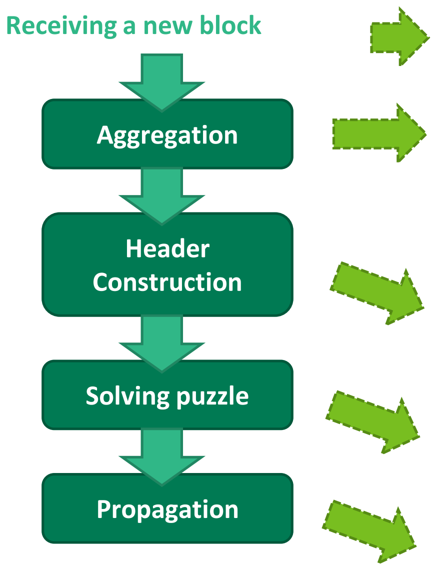
```
today() == 2011/09/25 && exchange_rate(mtgoxUSD) >= 12.5 && exchange_rate(mtgoxUSD) <= 13.5
Require exchange rate to be between two values on a given date
```



Bitcoin

Mining

- Miners compete to create new blocks by solving hash puzzles
- Bitcoin proof-of-work: hashcash



- End of one round of a competition is the beginning of the next round

- Remove the transactions of the newly announced block from transaction pool
- Aggregate remaining valid transactions; select subset that can be included
- Add coinbase transaction as the first transaction to the transaction list for the next block

- Calculate the hash of the previous block
- Construct a merkle tree to summarize all the included transactions

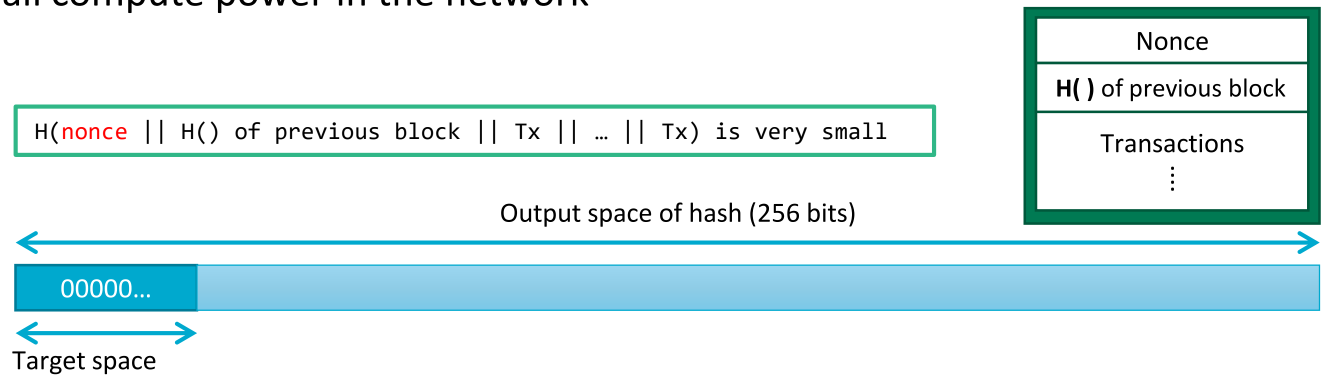
- Find solution to the proof-of-work algorithm
- The result will be inserted to the block header if successful

- If found, the new block is immediately propagated

Bitcoin

Mining – Proof-of-Work

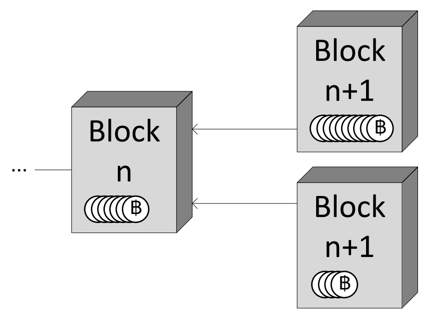
- Solve a hash puzzle - finding a value for a field in the block header, the nonce, which leads to the block hash being smaller than a given threshold
  - Requires a lot trials and errors – try values until you get luck
  - The threshold is adjusted over time to ensure that the average inter-block time is 10 minutes
  - The likelihood to solve the puzzle is proportional to the compute power invested relative to all compute power in the network



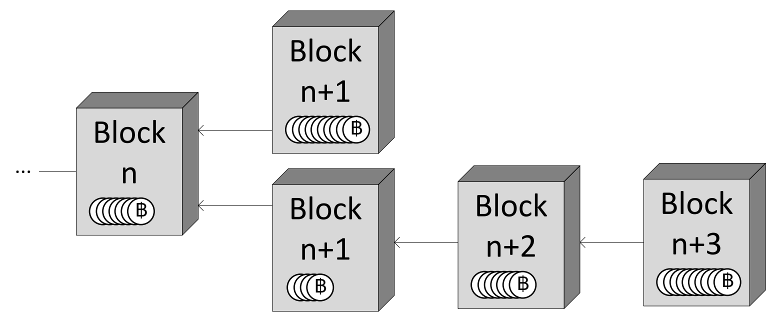
Bitcoin

Nakamoto Consensus

- Miners might find and announce the next block at the same time
- Treat the longest history of blocks as the *main chain*
  - The one that received most computation



Fork in the blockchain



Fork decided: longer chain wins

Bitcoin

Nakamoto Consensus

- To determine with high probability that a transaction is permanently included:
  - wait for several blocks (5 blocks by default) to be added after first inclusion of the transaction
  - Each of these subsequent blocks is called a *confirmation block*
  - Once sufficiently many confirmations occurred after the transaction block inclusion, then the transaction is considered *committed*
- Unlike many traditional transaction commit semantics:
  - Commit only has a probabilistic guarantee
  - A longer chain *could* appear – although it may be very, very unlikely

Bitcoin

Accounts and States

- An account is associated with a cryptographic key pair
  - Public key is used to create the account address
  - Private key: sign transactions sent from the account
- The account balance is the sum of UTXO that an account has control over
- The state of the blockchain, and the account balances of all users, result from the genesis block (very first block of the blockchain) and the set of transactions included since
  - Some accounts might be pre-loaded with an initial account balance from the beginning
- As transactions are grouped into blocks, the entire system moves from one discrete state to another through adding a new block

Bitcoin

Wallets and Exchanges

- Wallets
  - Software wallet: Manage a collection of private keys corresponding to their accounts, and to create and sign transactions
  - Hardware wallets are devices that store private keys in chips
  - Cold storage backups: to avoid key loss/stores a representation of the keys independent of the user's current hardware wallets
- Exchanges
  - Places to trade Bitcoin with other currencies
  - Holds currency on behalf of users
  - Clients may choose to ask the exchange to transfer purchased Bitcoin to an address under their control
  - If the exchange's system fails, their users may lose control of "their" Bitcoin.
  - Key stakeholders for public blockchain
    - Provide liquidity for cryptocurrency, which supports its real-world value
    - Underpin the incentive mechanism

Bitcoin

A bit of fun: Bitcoin Obituaries

- Declared dead over 300 times, and yet… still alive and operating!
- https://99bitcoins.com/bitcoin-obituaries/

Bitcoin Obituaries

Bitcoin has died 346 times

Submit an Obituary

Bitcoin Obituary Stats

Most Recent Death:

February 14, 2019 - JPMorgan Just Killed the Bitcoin Dream

Oldest Death:

December 15, 2010 - Why Bitcoin can't be a currency

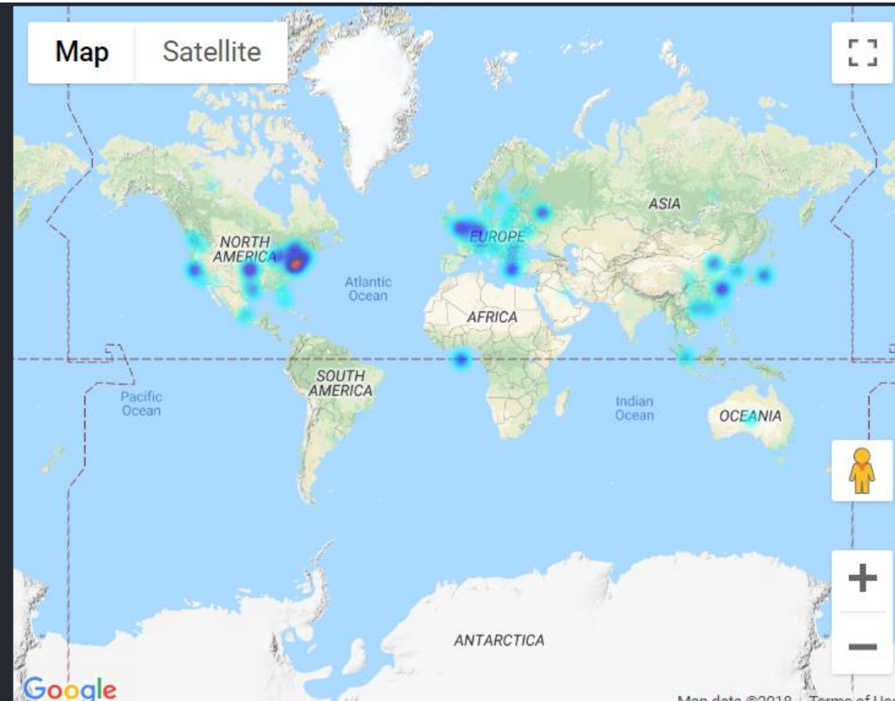
Agenda:

- Use cases
- Cryptography basics
- Bitcoin
- Ethereum
- Smart Contract Development

Ethereum

Network distribution

Total	13355 (100%)
United States	5612 (42.02%)
China	1752 (13.12%)
Canada	974 (7.29%)
Germany	557 (4.17%)
Russian Federation	462 (3.46%)
United Kingdom	436 (3.26%)
Netherlands	305 (2.28%)
Korea, Republic of	271 (2.03%)
France	240 (1.80%)
Ukraine	228 (1.71%)



Ethereum

Exchange rate

Lowest price: $0.05
Highest price: $1400

Source: etherscan.io/chart/etherprice

Ethereum

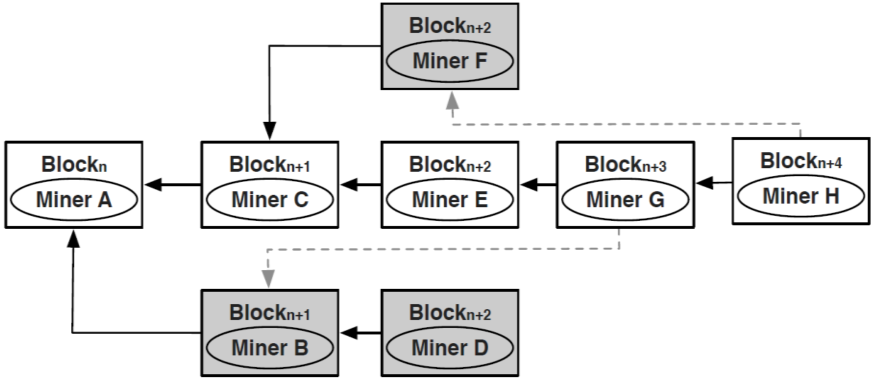
Proof of Work - Ethash

- Shorter inter-block time: 13-15 seconds (Bitcoin: 10 mins)
  - The Ethereum inter-block time is significantly shorter than Bitcoin's, typically meaning transactions are recorded and executed a lot faster on the Ethereum blockchain
  - According to median values, Ethereum appends a new block to its data structure every 13-15 seconds
- Smaller blocks
  - Gas limit per block - block size limits complexity of transactions
  - At most 380 transactions in a block (Bitcoin: 1,500 txs/block)
    - Currently maximum block size is around 8,000,000 gas
    - Basic transactions have a complexity of 21,000 gas
  - Most blocks are under 2KB (Bitcoin: 1 MB)

Ethereum

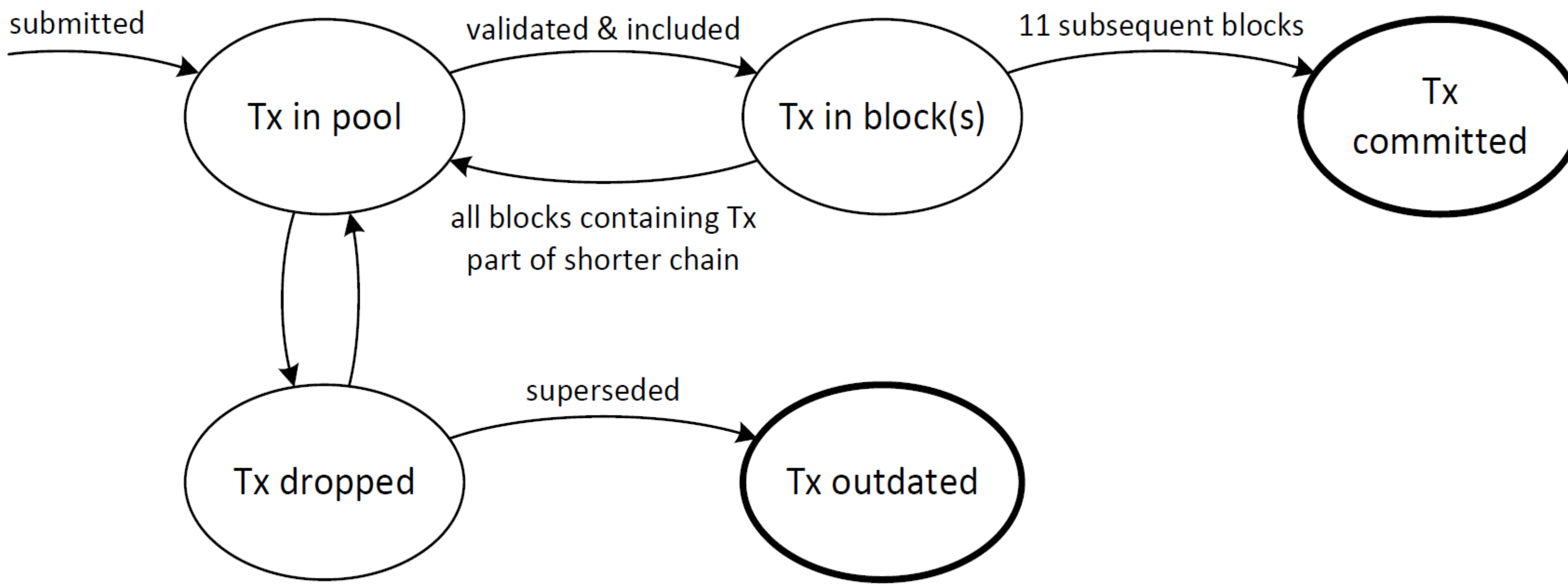
Ethereum Protocol

- Ethereum's inter-block time is not hugely longer than block propagation time around the globe
  - Much more likely that multiple competing blocks are created at a similar time
- GHOST protocol (Greedy Heaviest Observed Subtree)
  - The heaviest chain wins and uncles contribute to the weight
  - Miners of uncle blocks receive 87.5% of a standard block reward
  - For every uncle included in the block, the miner gains an additional 3.125% and increases the weight of the chain incl



Ethereum

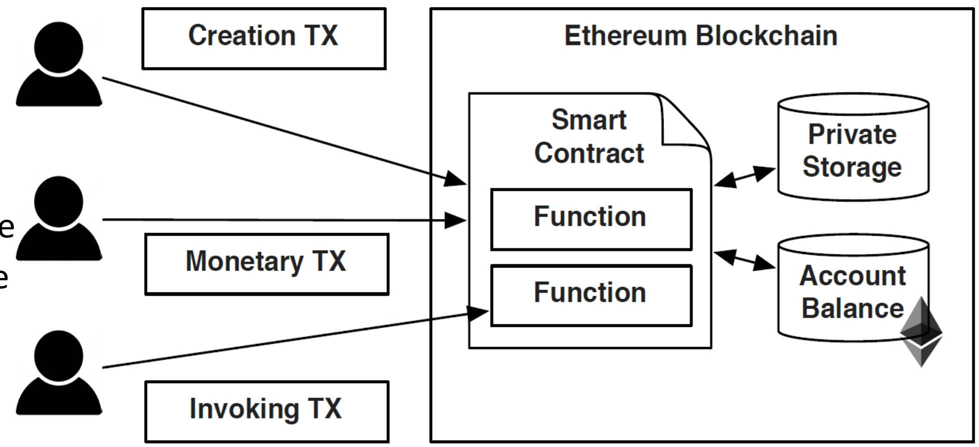
Transactions Lifecycle



Ethereum

Smart Contract (SC) in Solidity

- SC creation TX: deployment
  - After included, identified by a SC address
  - A smart contract account contains
    - A piece of executable code
    - An internal storage to store internal state
    - An amount of Ether, i.e. contract balance
- Monetary TX
  - Users can transfer Ether to the SC
- Invoking TX contains
  - The interface of the function being invoked and its parameters in the data payload
  - An amount of Ether as gas for the TX execution



Signature of the TX sender authorizes the TX

Ethereum

Paying Fees in “Gas”

- Gas: a fee to limit the resource usage
- Gas cost
  - A fixed gas cost per transaction (base cost)
  - Variable gas cost for data (dependent on its size) and execution of a smart contract method (charged per bytecode instruction)
  - Additional gas cost for the deployment of new contracts
- Gas cost is converted to Ether according to a user defined gas price –how much ether per gas the TX creator is willing to pay
  - By default, clients set the gas price to a market price
  - Users set higher gas price if inclusion of transaction is urgent

Ethereum

Data structure of a transaction

- An Ethereum transaction has the following fields
  - From – source address (regular account or contract address)
  - To – destination address (regular account or contract address)
  - Value – Ether (in unit “wei”) to transfer to the destination (can be 0)
  - Nonce – transaction sequence number for the sending account
  - Gas price – price you are offering to pay (Ether per gas)
  - Gas limit – maximum amount of gas allowed for the transaction (called “startgas”)
  - Data – payload data, e.g., description of transaction, binary code to create smart contract, or function invocation
  - Digital signature (field names: v, r, s)
- Once included, additional information (block number, actual gas used, actual fee, etc.) is available
- See e.g. https://etherscan.io/tx/0x5c504ed432cb51138bcf09aa5e8a410dd4a1e204ef84bfed1be16dfba1b22060

Agenda:

- Use cases
- Cryptography basics
- Bitcoin
- Ethereum
- Smart Contract Development

Recap: Dapps and Smart Contracts

- Smart contracts
  - Programs deployed as data and executed in transactions on the blockchain
  - Blockchain can be a computational platform (more than a simple distributed database)
  - Code is deterministic and immutable once deployed
  - Can invoke other smart contracts
  - Can hold and transfer digital assets
- Decentralized applications or dapps
  - Main functionality is implemented through smart contracts
  - Backend is executed in a decentralized environment
  - Frontend can be hosted as a web site on a centralized server
    - Interact with its backend through an API
  - Could use decentralized data storage such as IPFS
  - "State of the dapps" is a directory recorded on blockchain: https://www.stateofthedapps.com/

Smart Contracts in general

- Analogy: Java program
  - Smart contract code: a class
  - Deployed contract: an object
- Many software design patterns still apply, e.g., Factory
  - Code is deterministic, i.e., same state/inputs result in the same state changes/outputs
    - However, state may not be deterministic from viewpoint of the caller (e.g. block number)
- Why can smart contract execution be trustworthy?
  - A contract is deployed as data in a transaction and thus immutable
  - All their inputs are through transactions and the current state
  - Their code is deterministic
  - The results of transactions (including function calls) are captured in the state and receipt trees, which are part of the consensus
    - If other nodes get a different result, they would reject a proposed block

Smart Contract Development in Ethereum

- Smart contracts in Ethereum are deployed and executed as byte code
  - Runs on the Ethereum Virtual Machine (EVM)
- Typically, byte code results from compiling code in a high-level language
- Most popular language for Ethereum: Solidity
  - High-level, object-oriented language; syntax is similar to that of JavaScript
  - Statically typed, supports inheritance, libraries and complex user-defined types
- Useful links:
  - In-browser IDE: https://remix.ethereum.org/
  - Solidity official documentation page: https://solidity.readthedocs.io/en/latest/
  - Official tutorials / examples: https://solidity.readthedocs.io/en/latest/solidity-by-example.html
  - Other tutorials, e.g.: https://ethereumbuilders.gitbooks.io/guide/content/en/solidity_tutorials.html
  - Smart contract best practices from Consensys (a company) https://consensys.github.io/smart-contract-best-practices/

Solidity

Example

```
pragma solidity >=0.4.0 <0.6.0;  // compiler instruction: language and version

contract SimpleStorage {
        uint storedData;       // variable declaration

        // getter, callable by anyone on the network
        function get() constant returns (uint retVal) {
                return storedData;
        }

        // setter, callable by anyone on the network
        function set(uint x) {
                storedData = x;
        }
}
```

Solidity

Features

- Object-oriented, high-level language
- Statically typed, supports inheritance, libraries and complex user-defined types
  - Multi-inheritance is supported
- Designed for smart contracts to run on the Ethereum Virtual Machine (EVM)
- Syntax is closest to JavaScript, with some similarity to Java
  - But no support for Lambda expressions (anonymous methods without a declaration)
  - Also influenced by C++, Python
- Each deployed contract is assigned an address (like a regular account)
  - Can receive, hold, and spend assets, like Ether
  - Assets held by the account address are controlled by the program code
    - Bugs / vulnerabilities / omissions can lead to permanent loss of assets
    - But: untrusting parties can use smart contract as neutral, trustworthy middle ground
- Follow best practices, especially around security – there is no “safety net”

Solidity

Features

- Deploy contract by sending a transaction with recipient “to” set to “null”
- Can specify interfaces, by having at least one un-implemented function

```
contract base { function foo(); }                       // an interface
contract derived is base { function foo() {} }     // implements the interface
```

- Can overload functions (same function name but different parameters)
  - But might not be a good idea – hard to understand the code
- Arrays work in the usual, Java/JavaScript-like way
- Constructor is executed when a contract is created is executed once
  - Function name: `constructor`

```
uint[3] public data;  data[0] = 0; …
contract Base {
    constructor(uint i) public {…}
}
```

Solidity

Remix example

```
pragma solidity >=0.4.22 <0.6.0;
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.
    constructor(uint8 _numProposals) public {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;
        proposals.length = _numProposals;
    }
//...
}

// From https://remix.ethereum.org/
```

Solidity

Remix test example

```
import "remix_tests.sol"; // this import is automatically injected by Remix.
import "./ballot.sol";    // the file to test

contract test3 {

    Ballot ballotToTest;
    function beforeAll () public {
       ballotToTest = new Ballot(2);
    }

    function checkWinningProposal () public {
        ballotToTest.vote(1);
        Assert.equal(ballotToTest.winningProposal(), uint(1),
                 "1 should be the winning proposal");
    }

    function checkWinninProposalWithReturnValue () public view returns (bool) {
        return ballotToTest.winningProposal() == 1;
    }
}


// From https://remix.ethereum.org/
```

Solidity

Visibility of data / functions

- Solidity provides two types of function calls: internal and external
  - Internal call: local call from the same contract
  - External: by message call
- Functions can be specified to be *external*, *public*, *internal* or *private*
  - External: only from outside (transactions or other contracts); part of the contract interface
    - If function **f** is external, calling it internally is impossible (for example, **f()** will not work, however, **this.f()** works).
  - Public: can be called both by message calls and internally; part of the contract interface
    - Used to be the default, but compiler now enforces explicit declaration
  - Internal: only from other code in the same contract (including by inheritance)
  - Private: only be visible for the contract that they are declared in only, and not even in derived contracts
    - No data is ever really private in smart contracts – anyone on the network can extract it
- For state variables, external is not possible and the default is internal
  - Compiler automatically creates getter functions for public variables

Solidity

Visibility of data / functions - example

```
pragma solidity ^0.5.1;

contract cont1 {
    uint private data;
    function func(uint x) private returns(uint y) { return x + 1; }
    function dataSet(uint x) public { data = x; }
    function dataGet() public returns(uint) { return data; }
    function compute(uint x, uint y) internal returns (uint) { return x+y; }
}

contract cont2 {
    function dataRead() public {
        cont1 z = new cont1();
        uint local = z.func(7); // error: member "func" is not visible
        z.dataSet(3);
        local = z.dataGet();
        local = z.compute(3, 5); // error: member "compute" is not visible
    }
}

contract cont3 is cont1 {
    function g() public {
        cont1 z = new cont1();
        uint val = compute(3, 5); // access to internal member (from derived to parent contract)
    }
}

// Adapted from https://www.bitdegree.org/learn/solidity-visibility-and-getters
```

General principles for smart contracts

- Follow the KISS principle: keep it simple, stupid
  - ... and readable / understandable, so other developers/technical users can start trusting into your code
- Follow best practices, especially around security – there is no "safety net"
- Interface of a smart contract is NOT visible from the deployed code
  - No requirement of the binary code to have a particular structure
    - But Solidity compiler establishes a particular structure
    - This structure needs to be known to the caller – i.e., the transaction invoking a smart contract method must be aware of this structure and be constructed accordingly
  - Signatures can be guess-extracted (but not necessarily the function names), if the binary code was written in Solidity and compiled with the standard compiler
    - Cannot rely on it not becoming known
- Make the code available to the potential users (e.g., open source), or at least the interface
  - Many interface standards proposed, e.g., ERC-20 for fungible tokens
- Always look at the online documentation: Solidity is under very active development and changes are frequent

Summary

- Use cases
- Cryptography basics
- Bitcoin
- Ethereum
- Smart Contract Development

Course Outline – next two weeks

Week	Date	Lecturer	Lecture Topic	Relevant Book Chapters	Notes
2nd	25 Feb	Ingo Weber	Existing Blockchain Platforms	4. Example use cases 2. Existing Blockchain Platforms (1h on smart contract dev)	Assignment 1 out (Monday before lecture)
3rd	4 Mar	Sherry Xu	Blockchain in Software Architecture 1	3. Varieties of blockchain 5. Blockchain in Software Architecture (including software architecture basics) 1/2	
4th	11 Mar	Mark Staples	Blockchain in Software Architecture 2	5. Blockchain in Software Architecture (Non-functional properties and trade-offs) 2/2	Pitching session Assignment 1 due (Wednesday)

End of Lecture / Consultation

Ingo Weber | Principal Research Scientist & Team Leader

Architecture & Analytics Platforms (AAP) team

ingo.weber@data61.csiro.au

Conj. Assoc. Professor, UNSW Australia | Adj. Assoc. Professor, Swinburne University

www.data61.csiro.au