Principles of Distributed Ledgers

Lecture 1: Introduction to blockchains

Lewis Gudgeon, **Daniel Perez**, Paul Pritz, **Sam Werner**

October 9, 2023

Imperial College London

Course logistics

Lecturers

Lewis Gudgeon

Daniel Perez

Paul Pritz

Sam Werner

Outline

Weekly sessions:

- Mondays: 9am - 11am (HXLY 311)
- Fridays: 11am - 1pm (HXLY 145)
- Please refer to the course outline for tutorial dates (part of the weekly sessions)
- There will be 1-2 guest lectures

Reading material:

- Required and recommended reading material will be posted online for each week

Coursework and tutorials

Tutorials:

- Please bring your laptop to the tutorial
- Based on the lecture material of the same week
- Programming-focused
- Similar style to exam questions
- Solutions will be uploaded

Coursework (20%):

- Programming exercise
- End of week 5
- Individual

Exam and questions

Exam (80%):

- 2 hour exam
- Open book (1 A4 sheet)
- No coding required but need to be able to read and understand Solidity code

Questions:

- Speak to us during the tutorials
- Speak to the GTAs
- Post questions on Ed Discussion Forum

Course objectives

1. Understand the fundamental design and workings of blockchains
2. Understand the different consensus mechanisms of blockchains
3. Understand the workings of Ethereum in depth (Ethereum Virtual Machine, consensus mechanism, etc.)
4. Learn how to develop smart contracts on Ethereum using Solidity
5. Understand what different Decentralised Finance (DeFi) applications and concepts exist (e.g., Automated Market Makers, Flash Loans)
6. Be able to identify common attack patterns in DeFi applications
7. Understand the differences between technical and economic security in DeFi

What this course is not

- A cryptography course (see 70009 Cryptography Engineering)
- A non-programming course
- A general course on many different blockchains
- A course on crypto trading
- Financial advice

Number 1 Question of 2022/23

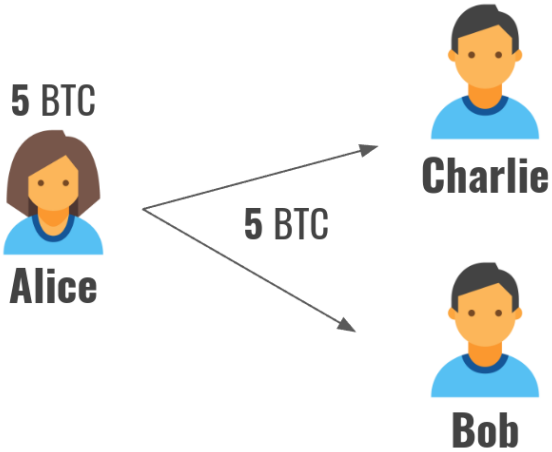
“Is this examinable?”

What is examinable?

- Everything covered in lectures
- Concepts covered in tutorials
- You won't need to write code in the exam
- Not what is covered in guest lectures **unless** explicitly stated
- Required and recommended readings exist to help you get the most out of the concepts introduced in the lectures. There won't be exam questions on required readings where the concepts have not been covered in the lectures.

Introduction to blockchain and Bitcoin

Double-spending problem



What do we want to achieve?

- A global state that can't be tampered with
- Cheap to verify the correctness of the state
- Trustless
- Non-custodial
- Censorship resistance
- Permissionless

What is a blockchain?

- A data structure that stores information, such as transaction data
- Peer-to-peer network
- Data is recorded in multiple identical data stores (ledgers) that are collectively maintained by a distributed network of computers (nodes)
- Consensus algorithm (all nodes see the same data)

2008: Bitcoin whitepaper

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot

Creation of Bitcoin

> *"A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution."*

Why was it novel?

- Trustless
- Permissionless
- Censorship resistant
- No double spending!

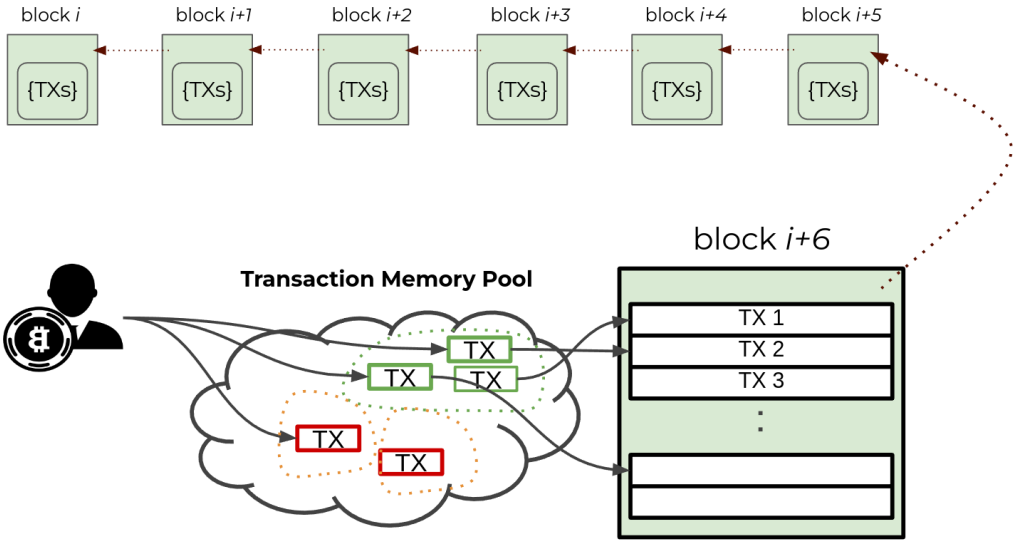
What's a block?

A data structure that stores information, such as transaction data.

In Bitcoin, a block consists of:

1. `block header`:
   - Identifies a particular block of transaction
   - Serialized in 80 byte format
   - Format:
     `<version><previous_block_header_hash><merkle_root_hash><time><...><nonce>`
2. `txn_count`: total number of transactions
3. `txns`: every transaction in the block

A block is not valid unless serialised size is $<=$ 1MB.


block i
block i+1
block i+2
block i+3
block i+4
block i+5
{TXs}
{TXs}
{TXs}
{TXs}
{TXs}
{TXs}
block i+6
Transaction Memory Pool
TX
TX
TX
TX
TX
TX 1
TX 2
TX 3

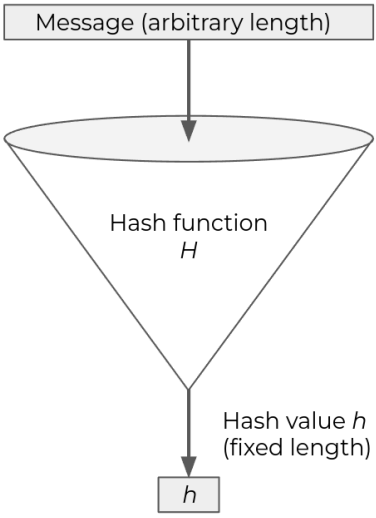
Hash functions

A one-way deterministic function for mapping input data of arbitrary size to a fixed-size bit string.

Characteristics:

- **Pre-image resistance:** For a given hash value h in the output space of hash function H, it is hard to find any input x such that $H(x) = h$.
- **Second pre-image resistance:** For an input value x and a hash function H, it should be hard to find a different input y, such that $H(y) = H(x)$.
- **Collision resistance:** For a hash function H, it is hard to find two different inputs x and y, such that $H(x) = H(y)$.

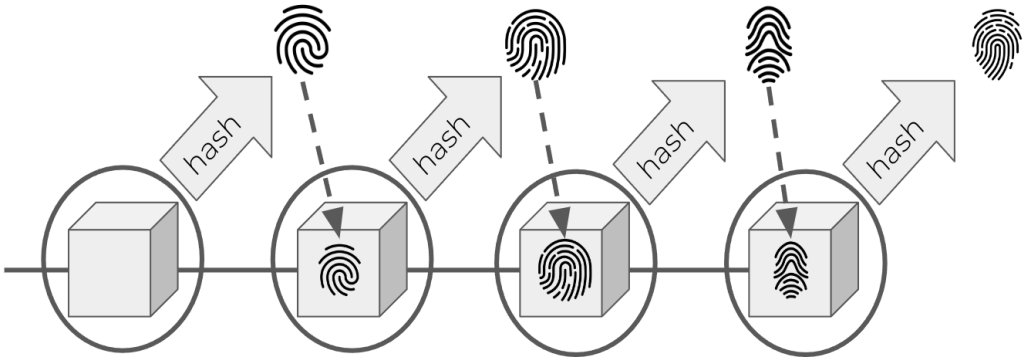
Hash functions



Hash functions

Examples:

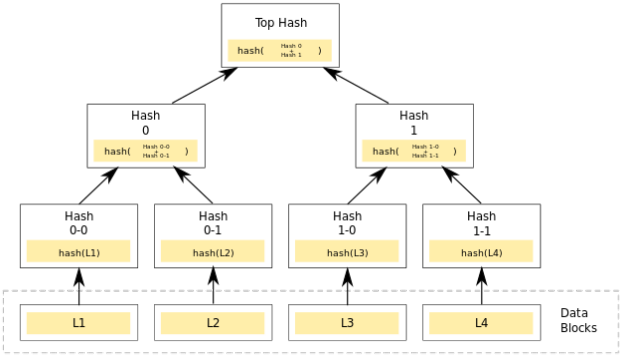
- SHA-256(*"Short input"*) $\rightarrow$
  `0x762c13ac8b2afa3f449e06c3413f703d8e545b095471764c224380e25b14b0e0`
- SHA-256(*"A very long input with all sort of characters: ¨¥f©ø¶§f´f"*) $\rightarrow$
  `0xd0caa87584c3e073fbb8b0986e3593397980568fdeb921d22ad38c8e1039dec1`


hash
hash
hash
hash

Merkle tree

Merkle trees are used to encode a set of values into a single hash: the Merkle root

- Created by hashing nodes together starting from leaves
- Allow for logarithmic time inclusion proof generation
- Allow for logarithmic time/storage inclusion proof verification



Example Merkle tree

Proof of Work (PoW)

- **PoW**: a leader election process in which participating nodes (*miners*) invest computational power in solving cryptographically hard, memoryless puzzles
- The node that generates a valid PoW solution determines the new set of valid transactions that should be added to the next block
- Partial pre-image attack on SHA256 in Bitcoin (generating a valid solution is hard, verifying a hash against a pre-image is easy)

Proof of Work (Bitcoin)

Proof-of-Work ⇒ `SHA-256(SHA-256(<block_header>)) <= Target (a number from 0 to` 2^{256}`-1)`

<version><previous_hash><time><…>**<nonce>**

- A candidate solution to the PoW is a hash of the block header
- Probability of a hash being a solution: $p = \frac{Target}{2^{256}}$ (binomial process)
- Expected number of hashes: $E[NumberOfHashesPerBlock] = \frac{1}{p} = \frac{2^{256}}{Target}$
- Block solve-times follow a Poisson distribution (mean = 10 minutes)
- Important to maintain a stable block solve-time for TX throughput
- *Difficulty* represents how difficult the current target makes it to find a block, relative to how difficult it would be at the highest possible target
- Difficulty algorithms adjust the target (easiness) of the PoW puzzle

Mining

Miners are nodes that try to solve the PoW puzzle.

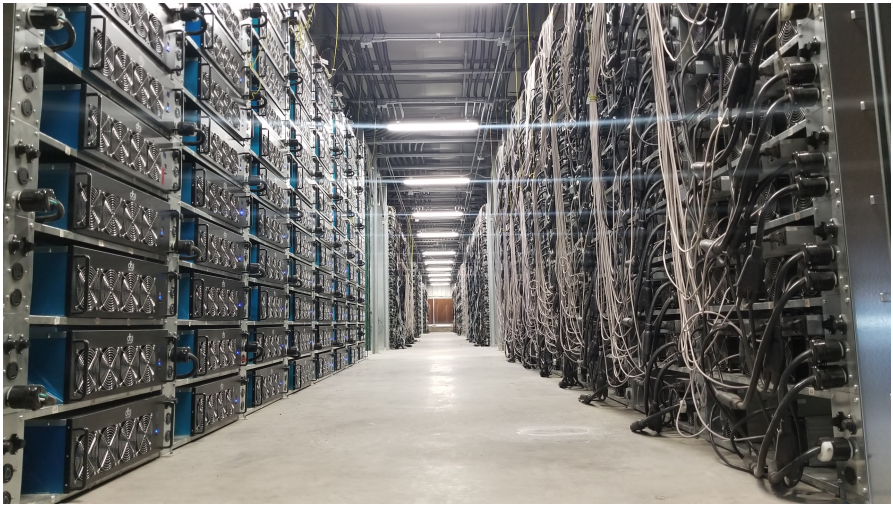
Nodes are incentivised to select transactions because of:

- Transaction fees (difference between TX outputs and TX inputs)
- Block reward (newly minted coins per new block)

In Bitcoin:

- Finite supply of 21 million BTC
- Reward halves approx. every 4 years (next is mid April 2024)
- Current reward is 6.25 BTC per block

Mining in practice



Source: https://www.americanbanker.com/payments/news/a-greener-approach-to-crypto-mining

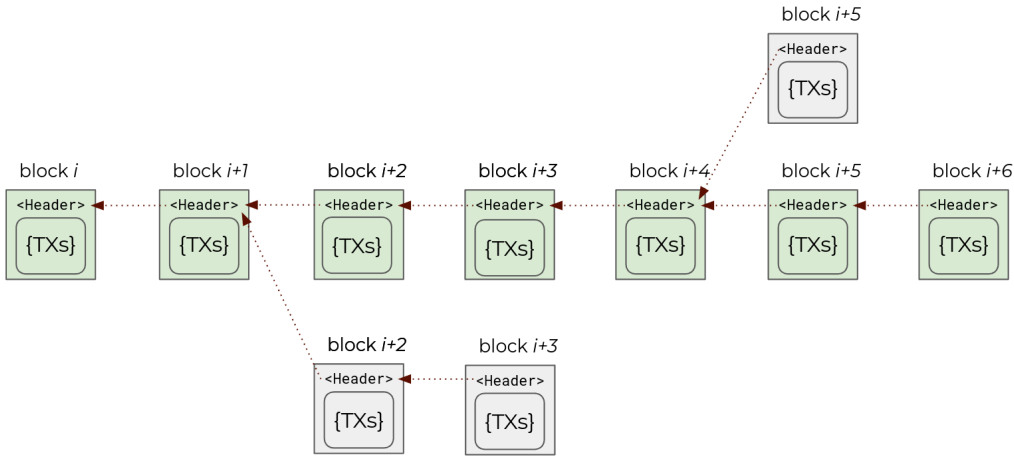
“Longest chain” rule

Longest chain rule == accept the “correct” version of the transaction history (blocks)

- Allows nodes to reach agreement on which history is the right one
- How many hashes one would have needed to perform to mine each block

Block tree: all valid blocks whose previous linked blocks are known (up to genesis block).

Active chain: a single path from the genesis block to a leaf node of the block tree (every path is a valid path), however, nodes pick the one path with the most “work” performed (sum of the difficulties).


block i+5
<Header>
{TXs}
block i
<Header>
{TXs}
block i+1
<Header>
{TXs}
block i+2
<Header>
{TXs}
block i+3
<Header>
{TXs}
block i+4
<Header>
{TXs}
block i+5
<Header>
{TXs}
block i+6
<Header>
{TXs}
block i+2
<Header>
{TXs}
block i+3
<Header>
{TXs}

Forks

Soft fork:

- Protocol changes that remain backward compatible with older protocol versions (clients adhering to the previous protocol rules)
- Example: a stricter limit on the block size (e.g. 0.5 MB instead of 4MB)
- Generally just requires the majority of miners to upgrade

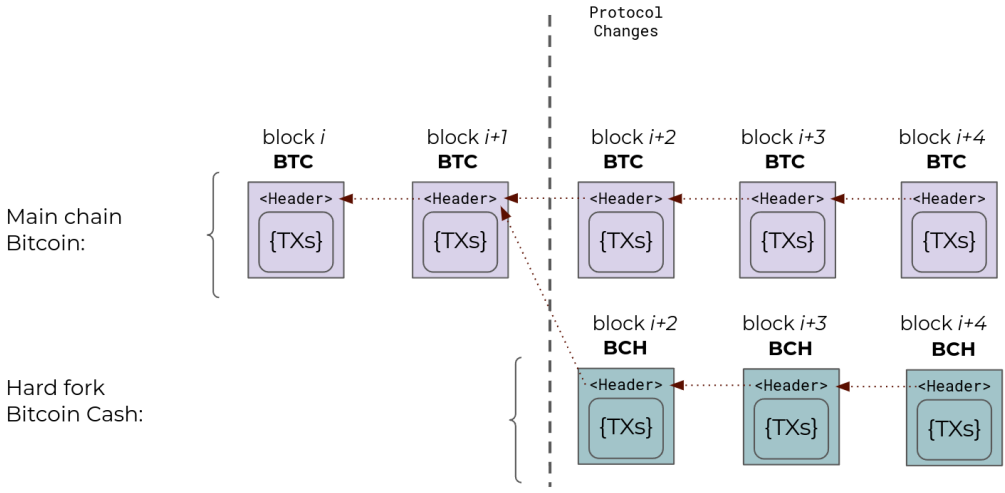
Hard fork:

- Protocol changes which can incur a permanent split of the blockchain (not backward compatible) – blocks considered invalid under previous protocol rules
- A subset of network participants will reject branches that build on blocks that are invalid to them
- Examples: increased block size

Case study: Bitcoin Cash (BCH)

- On 1 August 2017 (block number 478559), Bitcoin hard forked into Bitcoin (BTC) and Bitcoin Cash (BCH)
- Bitcoin Cash implemented a larger block size (from 1 MB to 32 MB), new difficulty algorithm, etc.
- Holders of BTC received an equal amount of BCH on the "new" chain

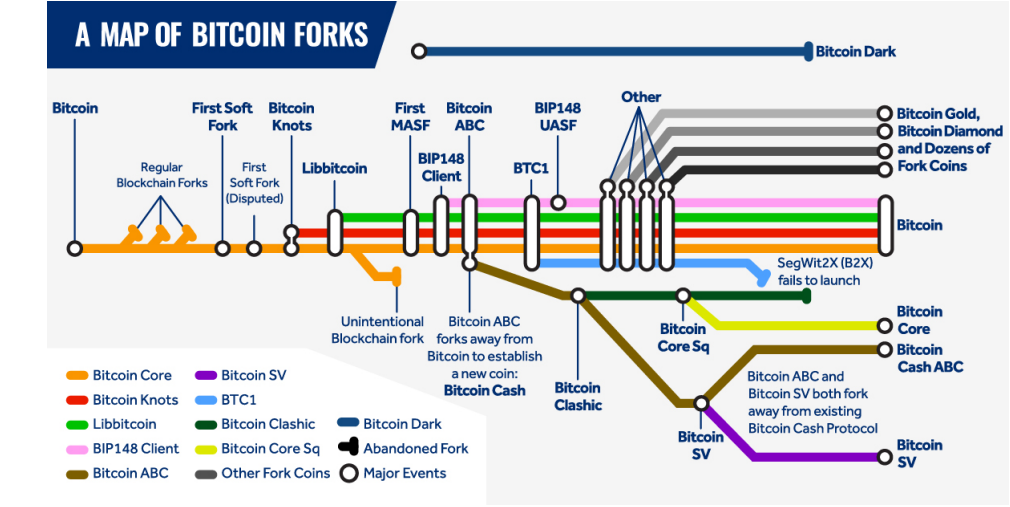
BTC and BCH fork



Post-fork price: BTC

Post-fork price: BCH

Bitcoin forks



Source: https://www.visualcapitalist.com/major-bitcoin-forks-subway-map/

Keys and Wallets

- Ownership of Bitcoins is proven using digital keys and digital signatures
- A digital key has an associated unique address
- Digital keys are generated and stored privately in a wallet
- Transactions require a digital signature to prove ownership of the keys
- Anyone with access to the keys can spend the money owned by the address

Public Key Cryptography

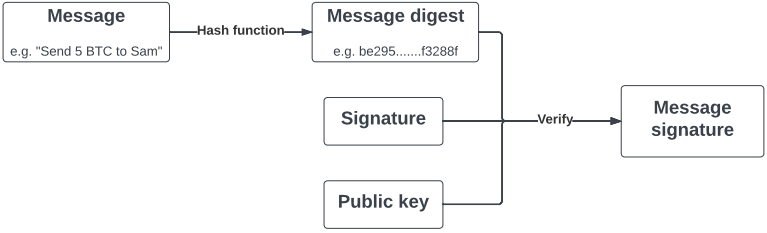
Public key cryptography uses a pair of keys, private and public, to securely allow operations such as message encryption or digital signatures.

- Bitcoin uses Elliptic Curves (ECDSA) as its basis for public key cryptography
- A Bitcoin private key is a random number between 1 and 2^{256}
- A Bitcoin address is derived from a public key, that is derived from a private key
- Using a private key, a user can sign any arbitrary message
- With the message and a public key, anyone can verify the message authenticity

Message signing

Signing a message:

Verifying a signature:



Pseudorandom number generators

Psudorandom number generators (PRNG) allow to generate a series of numbers that "look" random.

Characteristics

- PRNGs require a seed value, that is typically a random number
- PRNGs are not "really" random because they will always give the same series of numbers given the same seed

Generating an address

1. Generate a random number, the seed. This number is typically generated from a seed phrase, a list of random words.
2. Use the seed and a PRNG to generate another random number: the private key
3. Generate the unique public key associated to the generated private key
4. Hash the public key to obtain the address

State and transitions

In Bitcoin, state refers to the set of all coins (who owns what)

How can we transition between states?

- State transitions are triggered via *transactions*
- A transaction will typically take coins from one owner and send it to one or more other owners
- State transitions actually occur when the transaction is included in a block and the block added to the chain

Transactions

- Created $\rightarrow$ signed $\rightarrow$ propagated $\rightarrow$ validated $\rightarrow$ added to the blockchain
- Publically broadcasted (no confidential information)
- Data structures that encodes the transfer of funds from an input source to an output destination
- 300 - 400 bytes of data

Structure:

`<version><input_counter><inputs><output_counter><outputs><lock_time>`

Unspent Transaction Output (UTXO)

- Fundamental building block of a transaction
- Amounts of BTC that are linked to some address
- Whenever someone receives BTC, this amount is recorded as a UTXO
- Transactions consume one or more UTXOs
- UTXOs must be consumed entirely
- Wallet applications typically combine many small UTXOs into one that is equal or larger than the desired transfer amount
- Sending BTC == creating a UTXO that belongs to the recipient

Unspent Transaction Output (UTXO)

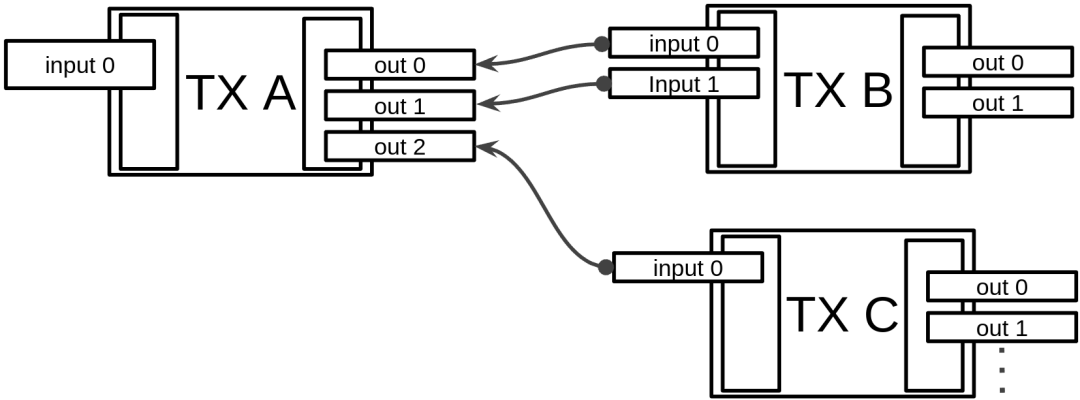
Transaction **inputs**:

- UTXOs consumed by a transaction (reference to tx hash + sequence number)
- Unlocking scripts that meet spending conditions (e.g. signature proving ownership)

Transaction **outputs**:

- UTXOs created by a transaction (amount + locking script)

Transaction fees: inputs - outputs


input 0
TX A
out 0
out 1
out 2
input 0
Input 1
TX B
out 0
out 1
input 0
TX C
out 0
out 1

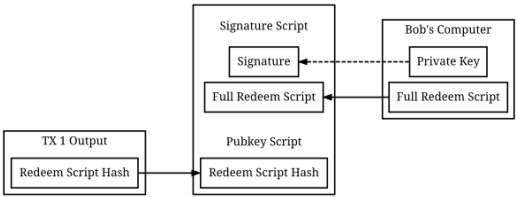
Bitcoin scripting

- Bitcoin uses a scripting language
- Stack-based
- Has conditional jump
- Not Turing-complete (no loop)
- Mostly used to define how to spend Bitcoins
- Transaction is invalid if script fails

```
OP_2
41 ; push 65 bytes
PUBKEY1
41
PUBKEY2
41
PUBKEY3
OP_3
OP_CHECKMULTISIG
```

Bitcoin Scripting and UTXO

- Bitcoin follows an UTXO model
- Bitcoin uses “redeem scripts” to enable scripting with UTXO
- Only a hash of the script is published beforehand
- Full scripts are published when spending



Using a Pay to Script Hash (P2SH)

Flow of a Bitcoin redeem script

1. Write script
2. Hash script to create address
3. Receive Bitcoins
4. Publish script and required data (usually signature) using a transaction

Block explorer

Let's check out what UTXOs look like in the wild!

Reading Material

Reading Material

Required reading:

- Bitcoin whitepaper
- Bitcoin SoK

Outlook

Friday's lecture

The next lecture will be about other blockchains, and specifically about Ethereum.

Tutorial: Implementing a simple blockchain node (Oct. 16)

- Load blockchain state from file
- Produce new blocks (load transactions from file) using proof-of-work
- Write new state to file

Can use any language, but we recommend Python.