# DLT - Distributed Ledger Tech

Péter Garamvölgyi

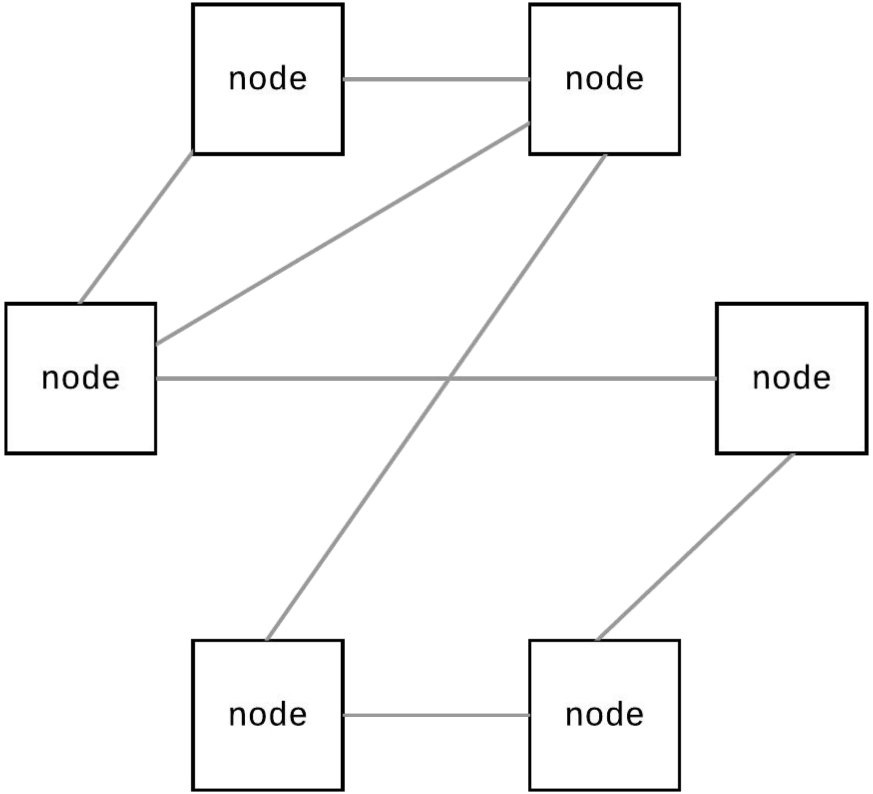
# Outline

- **what problem are we tackling?**
  - what are transactions?
  - what is a ledger?
  - validity and order of transactions
  - comparison with DBMS
  - DLT types

- what is a blockchain?
  - what is a hash function?
  - blockchain data structure
  - Proof-of-Work consensus
  - hard and soft forks
  - adversarial scenarios
  - transaction finality
  - performance
  - key issues

# The basic setup

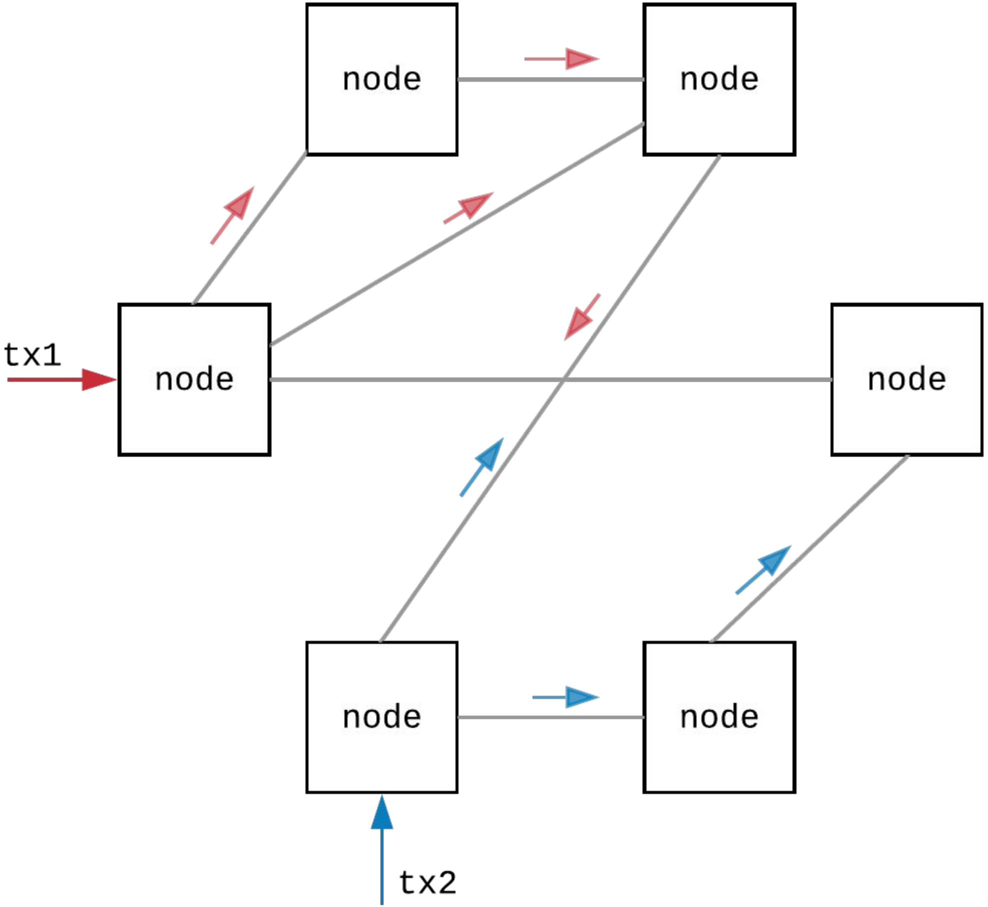
- p2p network of independent nodes



# The basic setup

- p2p network of independent nodes

- async transactions
- gossip protocol

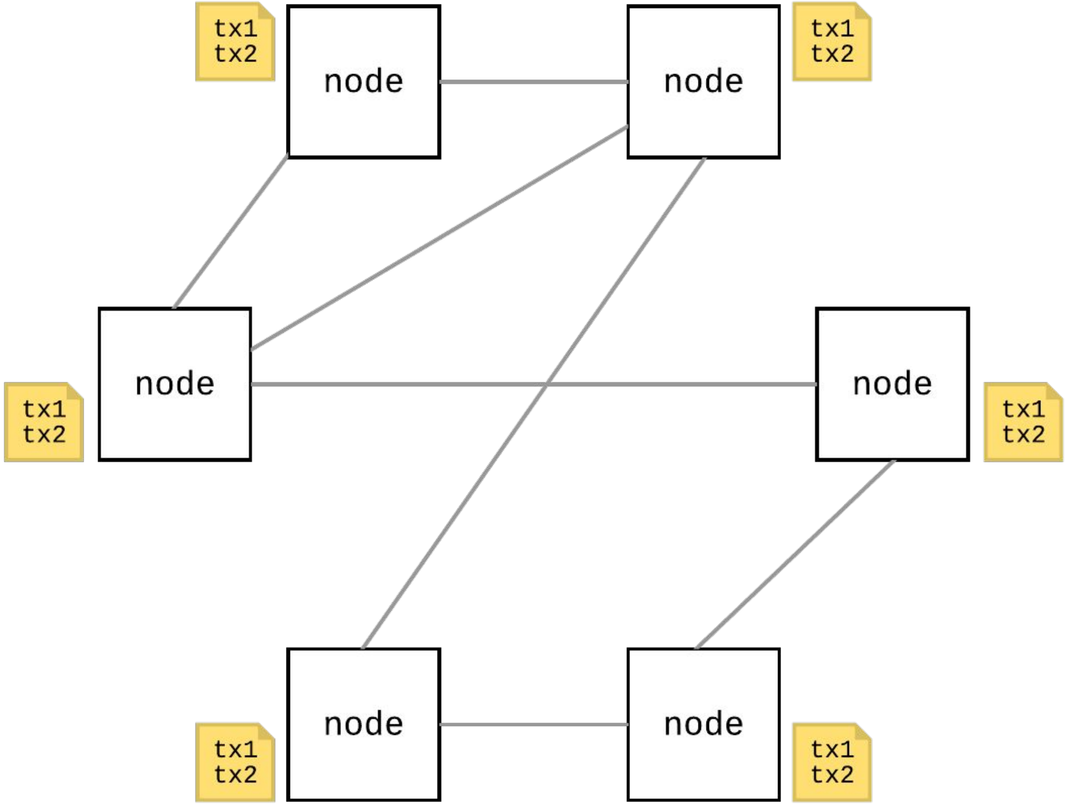


# The basic setup

- p2p network of independent nodes

- async transactions
- gossip protocol

- matching ledgers*



* in practice, this is not always an exact match

# But… what is a transaction*?

- a transaction is a **unit of change**

```
transfer 3 coins from Alice to Bob

change the value of variable x to 5

call the method transfer with the argument 7
```

* transaction will often be abbreviated as tx

# But… what is a ledger?

- a ledger is an (ordered) **append-only collection of transactions**

  ~ immutable database

# The challenge of DLT*

come up with an **algorithm for each node** so that they **reach consensus** on

1. which transactions are **valid**?
2. what is the (partial) **order** of the transactions?

**honest nodes** should end up having the same ledger.

**malicious nodes** should not be able to break the system.

(prevent censorship, spamming, sybil attacks, eclipse attacks, etc.)

* this is an instance of the **Byzantine Generals Problem**

# 1. Transaction validity

- I have the right to send the tx (cannot forge)

    `transfer 3 coins from Alice to Bob` **`(by Bob)`**

    `Alice changes a variable she has no access to`

- the tx does not conflict with other txs (e.g. **double spend**)

    `transfer 3 coins from Alice to Bob` **`and`**

    `transfer 3 coins from Alice to Claire`

```
balance(Alice)  = 3

balance(Bob)    = 0

balance(Claire) = 0
```

# 2. Transaction ordering (1)

- agree on a given order of the txs*

```
tx1 = send 3 coins from Alice to Bob (by Alice)

tx2 = send 3 coins from Bob to Claire (by Bob)
```

```
balance(Alice)  = 3

balance(Bob)    = 0

balance(Claire) = 0
```

`(tx1-tx2)` valid, `(tx2-tx1)` not!

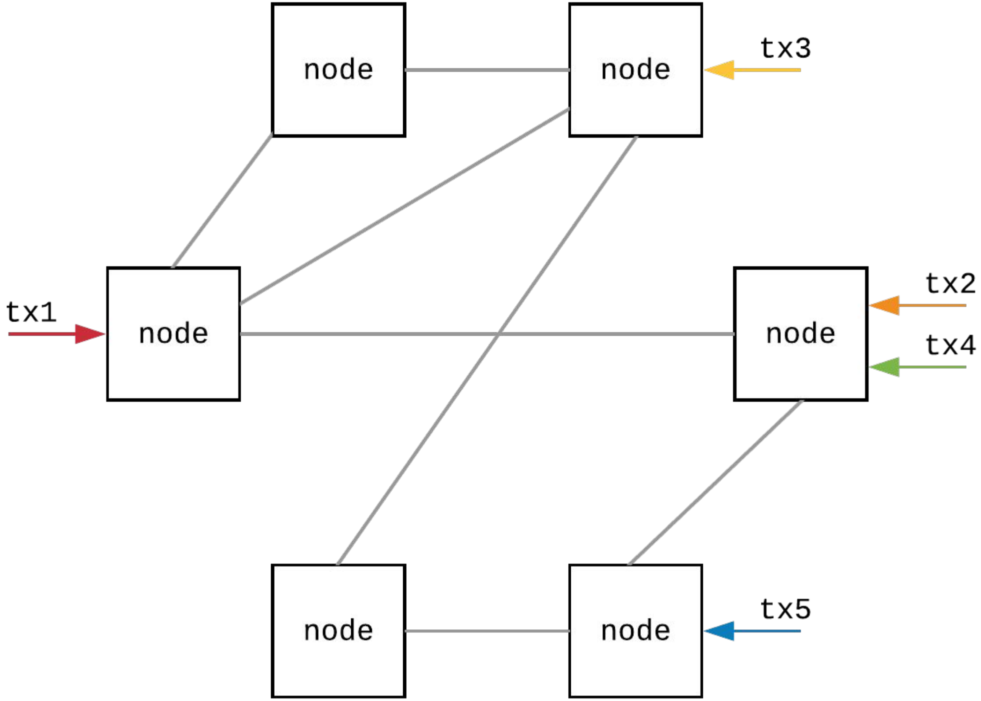
goal: all nodes choose one ordering

* note that timestamps are not reliable here

# 2. Transaction ordering (2)

- goal: all nodes agree on a given ordering, e.g.

```
tx1
tx2
tx3
tx4
tx5
```



# DLT vs DBMS

| | DLT | DBMS |
|---|---|---|
| **main goal** | decentralization, trustless | availability, performance |
| **nodes** | independent, competing | homogeneous/federated |
| **data redundancy** | very high | moderate, configurable |
| **data storage** | immutable* ledger | varies |

* immutability holds under normal circumstances (exceptions include hard forks, etc.)

# Types of DLT

- **blockchain** (Bitcoin, Ethereum, NEO)
- blockDAG (Spectre, Phantom, Conflux)
- DAG/tangle (IOTA, Hashgraph)

- key components / challenges / design decisions
  - ledger structure
  - consensus algorithm
  - network architecture

# Outline

- what problem are we tackling?
  - what are transactions?
  - what is a ledger?
  - validity and order of transactions
  - comparison with DBMS
  - DLT types
- **what is a blockchain?**
  - what is a hash function?
  - blockchain data structure
  - Proof-of-Work consensus
  - hard and soft forks
  - adversarial scenarios
  - transaction finality
  - performance
  - key issues

# First: what is a hash function?

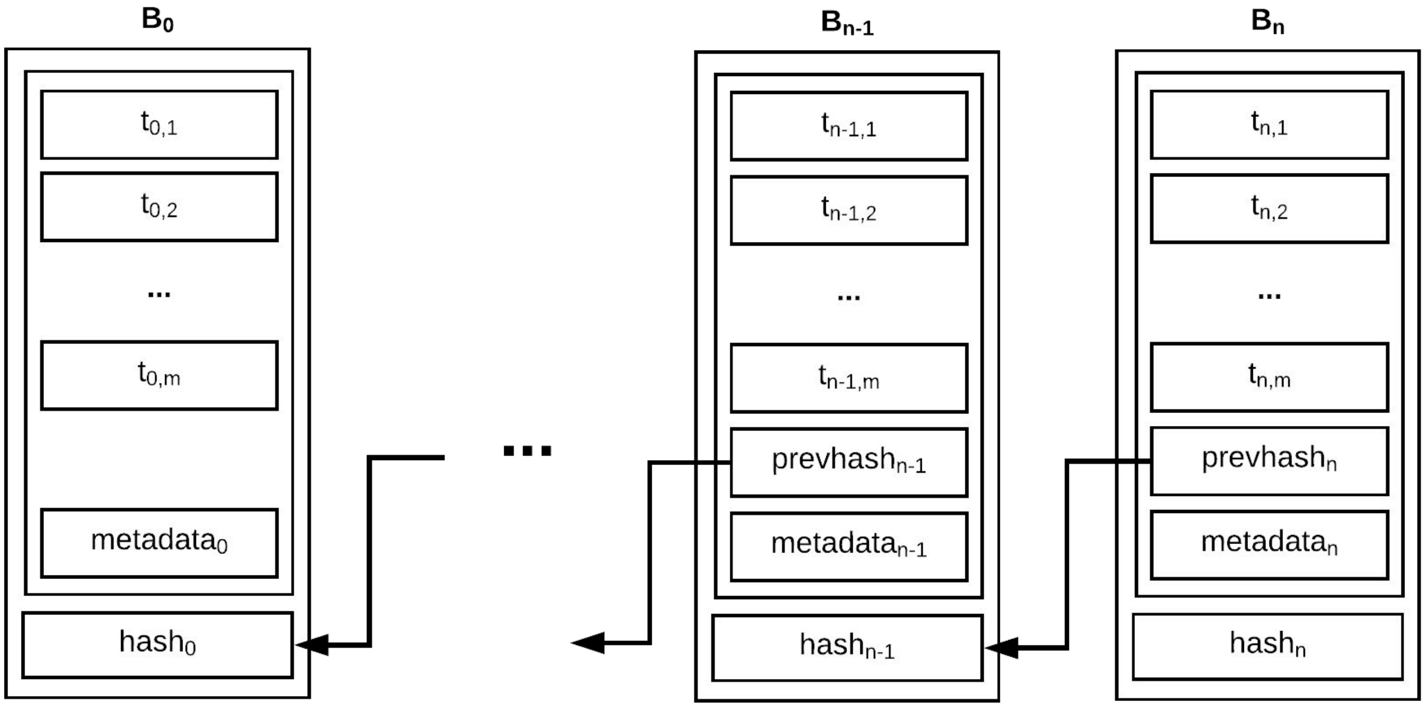
```
            -----------
alpha ---> | hash func | ---> be76331b95dfc399cd776d2fc68021e0db03cc4f
            -----------

            -----------
blpha ---> | hash func | ---> 978db2f4da63d9ed6cf0a8bee17ae9852289780f
            -----------
```

- unique
- irreversible
- avalanche-effect

- MD5
- SHA-256
- BLAKE2

# The blockchain data structure



metadata: timestamp, block size, **nonce**, tx id, state root, etc.

# Blockchain as a distributed ledger

1. **tx validity**: txs inside blocks are valid (others: pending/discarded)
2. **tx ordering**: order of blocks, order of txs inside each block

```
 -------         -------
|  tx1  |   .---|  tx4  |
|  tx2  |   |   |  tx5  |        (tx1, tx2, tx3), (tx4, tx5, tx6)
|  tx3  |<--.   |  tx6  |
 -------         -------
```

# Who creates the block? Proof-of-Work (Bitcoin)

- nodes (**miners**) compete for creating next block
- each node solves a computationally complex **puzzle**:

*find nonce s.t. block hash will start with a certain number of zeroes*

e.g. Bitcoin blockchain **block #544881**:

```
nonce = 1172287561
hash  = 0000000000000000000b7fbec02765fc8ddffa7645c9a5da3b5c3091307c0dc2
```

- brute-force (try many possibilities)
- 1 block / 10 mins on average (dynamic difficulty)

# Proof-of-Work consensus (Bitcoin)

1. miner **M** creates a new block **B** by finding the right nonce (PoW)
2. broadcast block to the network (gossip)
3. other nodes validate **B** (nonce, txs, etc.)
4. if accepted:
   - nodes will try create next block on top of **B**
   - **M** gets block reward and transaction fees*

(financial incentives make it more rewarding to play by the rules**)

* caveat: the block might still be dropped

** the study of incentives in DLT systems is often referred to as **cryptoeconomics**

# Proof-of-Work consensus (Bitcoin)

“*The only way to confirm the absence of a transaction is to* ***be aware of all transactions****. […] we need a system for participants to* ***agree on a single history*** *of the order in which [the transactions] were received.*” *

“*To modify a past block, an attacker would have to* ***redo the proof-of-work of the block and all blocks after it*** *and then* ***catch up*** *with and surpass the work of the honest nodes.*” *

* Bitcoin: A Peer-to-Peer Electronic Cash System

# Incentives

“*The incentive may help* ***encourage nodes to stay honest****. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins.*” *

possible incentives: block reward, tx fees, loss aversion, reputation, etc.

* Bitcoin: A Peer-to-Peer Electronic Cash System

# Overview

blockchain is …

… an **immutable** ledger …

… of transactions …

… ordered and validated …

… by nodes in a p2p network …

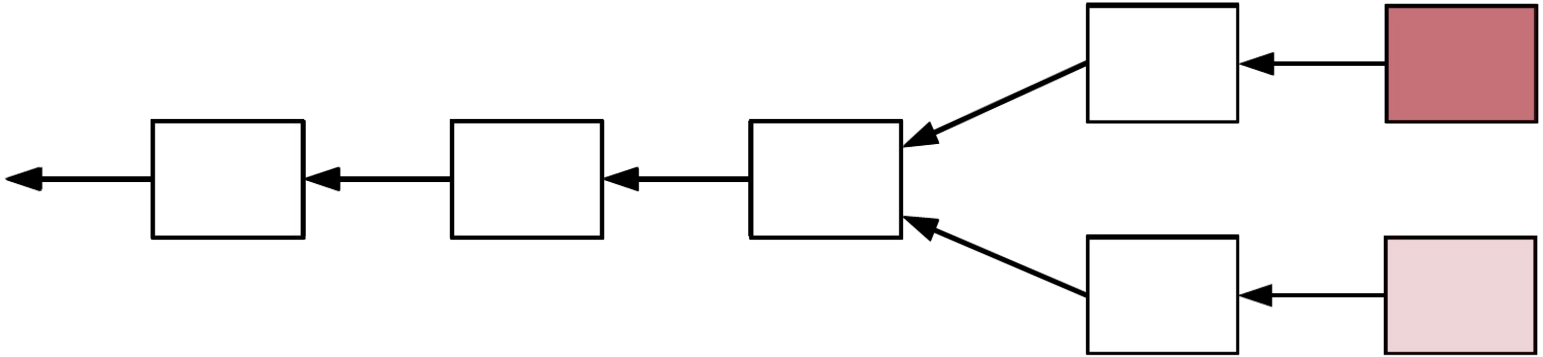
… who **do not trust each other**.

# Outline

- what problem are we tackling?
  - what are transactions?
  - what is a ledger?
  - validity and order of transactions
  - comparison with DBMS
  - DLT types
- **what is a blockchain?**
  - what is a hash function?
  - blockchain data structure
  - Proof-of-Work consensus
  - **hard and soft forks**
  - adversarial scenarios
  - transaction finality
  - performance
  - key issues

# Forks

> *“If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. [...] Nodes always **consider the longest chain to be the correct** one and will keep working on extending it.”**



* Bitcoin: A Peer-to-Peer Electronic Cash System

# Forks

**soft fork**

- happens spontaneously all the time
- **compatible** protocol changes

e.g. change block size to 0.5M from 1M.

**hard fork**

- under special circumstances
- **incompatible** protocol changes

e.g. the DAO hard fork (Ethereum)

# Adversary model

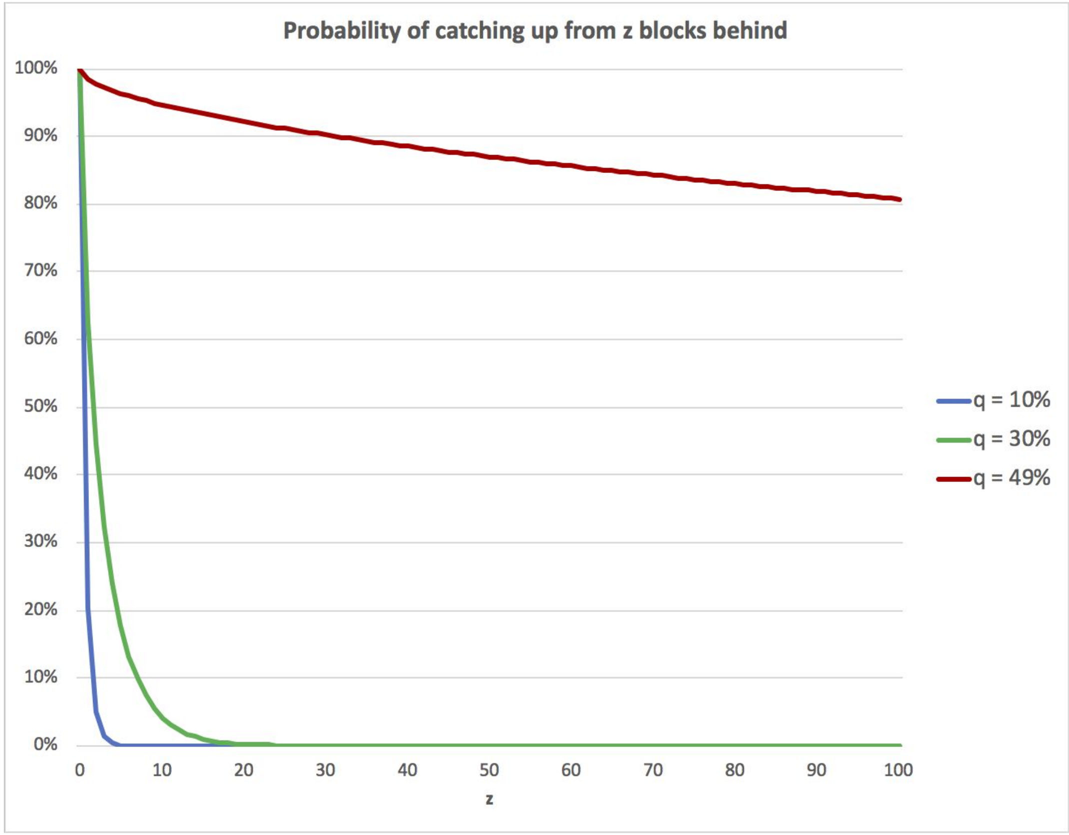
An attacker can

- ~~create new coins~~
- ~~take other user’s coins~~
- **try to change its earlier transaction**
- **take back recently spent money (double spend)**
- prevent tx delivery (censorship) or isolate node from network (eclipse attack)

# Attack probability

- probability of successful attack falls exponentially

- **51% attack**
  - 50%+1 attack
  - due to longest chain rule



# Transaction finality

- are transactions reversible?
  - yes (to a certain extent)

- probabilistic finality model
  - after 6 blocks you can be pretty sure

- can I?
  - the DAO attack hard fork

# Performance of PoW blockchains

- lots of unnecessary work
- conflicting blocks
  - GHOST/uncles
- everyone has to validate
- block propagation is constrained by network latency

# Key Issues

- **scalability**

```
Bitcoin ~   7 TPS
VISA    ~ 50k TPS
```

ledger structures, consensus algorithms

sharding, payment channels

- **privacy**

everything is public

zk-proofs, homomorphic encryption

off-chain data storage

- **energy-efficiency**

energy usage comparable to smaller countries

alternative consensus algorithms

useful work for mining

# Not mentioned here

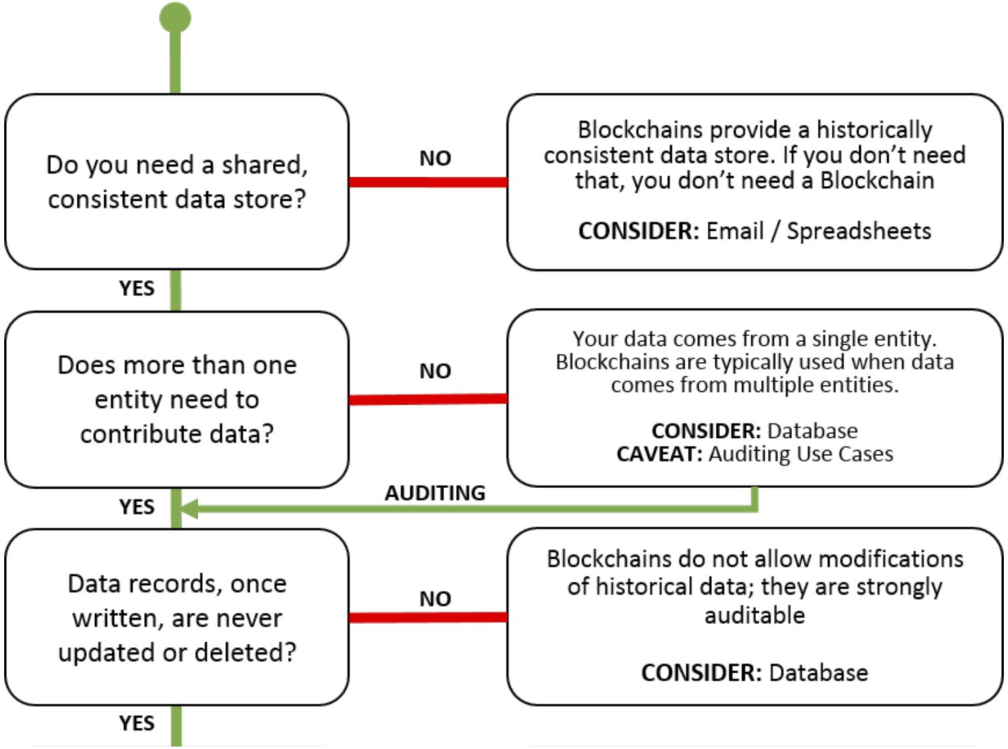
- **other consensus algorithms**

  Proof-of-Stake, BFT, dPoS, dBFT, Proof-of-Authority, Proof-of-Space, etc.

- **smart contracts**

- **...**

# When to use blockchain? (1) *



Do you need a shared, consistent data store?

Blockchains provide a historically consistent data store. If you don’t need that, you don’t need a Blockchain

**CONSIDER:** Email / Spreadsheets

Does more than one entity need to contribute data?

Your data comes from a single entity. Blockchains are typically used when data comes from multiple entities.

**CONSIDER:** Database
**CAVEAT:** Auditing Use Cases

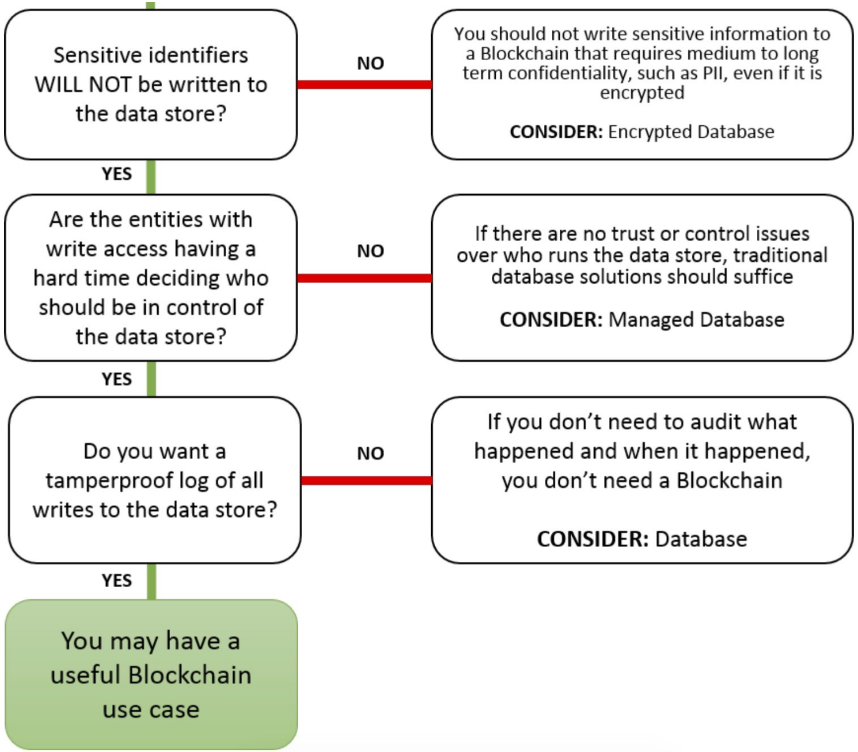
Data records, once written, are never updated or deleted?

Blockchains do not allow modifications of historical data; they are strongly auditable

**CONSIDER:** Database

* NISTIR 8202 Blockchain Technology Overview https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf

# When to use blockchain? (2) *



Sensitive identifiers WILL NOT be written to the data store?

NO

You should not write sensitive information to a Blockchain that requires medium to long term confidentiality, such as PII, even if it is encrypted

**CONSIDER:** Encrypted Database

YES

Are the entities with write access having a hard time deciding who should be in control of the data store?

NO

If there are no trust or control issues over who runs the data store, traditional database solutions should suffice

**CONSIDER:** Managed Database

YES

Do you want a tamperproof log of all writes to the data store?

NO

If you don't need to audit what happened and when it happened, you don't need a Blockchain

**CONSIDER:** Database

YES

You may have a useful Blockchain use case

* NISTIR 8202 Blockchain Technology Overview https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf