



**BITS Pilani**  
Pilani Campus

# Blockchain Technology and Systems

(SEZG569/SSZG569)

Dr. Ashutosh Bhatia  
Department of Computer Science and Information Systems



# QUIZ, HYPE & FACTS



# QUIZ

**Q0**

---

## What is BITCOIN

- a) A Cryptocurrency
- b) A decentralized peer-to-peer network
- c) A public transaction ledger
- d) All

# Q1

---

## Who created Bitcoin?

### Satoshi Nakamoto

Satoshi Nakamoto is the name used by the presumed pseudonymous person or persons who developed bitcoin, authored the bitcoin white paper, and created and deployed bitcoin's original reference implementation. As part of the implementation, Nakamoto also devised the first blockchain database. Nakamoto was active in the development of bitcoin up until December 2010. Many people have claimed, or have been claimed, to be Nakamoto.

source: [https://en.wikipedia.org/wiki/Satoshi\\_Nakamoto](https://en.wikipedia.org/wiki/Satoshi_Nakamoto)

---

# Q2

---

## Where do you store your cryptocurrency?

### Crypto Wallets

In the cryptocurrency ecosystem, the term “wallet” refers to software, online or offline, that allows a cryptocurrency owner to access their cryptocurrency holdings.

# Q3

---

## What is a miner?

Computers that validate and process blockchain transactions

# Q4

## Where can you buy cryptocurrency?

- A private transaction
- An exchange
- A Bitcoin ATM

**Q5**

---

## What is a blockchain?

- a) A distributed ledger on a peer to peer network
  - b) A type of cryptocurrency
  - c) An exchange
  - d) A centralized ledger
-

**Q6**

---

## What is a DApp?

A decentralized application that is developed over blockchain platform

---

# What is the term for when a blockchain splits?

## Fork

A bitcoin hard fork refers to a radical change to the protocol of bitcoin's blockchain that effectively results in two branches, one that follows the previous protocol and one that follows the new version. It is through this forking process that various digital currencies with names similar to bitcoin have been created, including bitcoin cash and bitcoin gold. Bitcoin cash remains the most successful hard fork of the primary cryptocurrency; as of June 2021, it is the eleventh-largest digital currency by market cap.

# Q8

---

What incentivizes the miners to give correct validation of transactions?

A block Reward in form of Bitcoins

**Q9**

---

## What is a hash function?

Takes an input of any length and returns a fixed-length string of numbers and letters

**Q10**

---

What does IPFS stand for?

Interplanetary File System

The **InterPlanetary File System (IPFS)** is a protocol and peer-to-peer network for storing and sharing data in a distributed file system. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting all computing devices.<sup>[4]</sup>

[InterPlanetary File System - Wikipedia](#)

# Q11

---

What is the maximum number of bitcoins that can be created?

21 million

# Q12

---

What was the highest the  
Bitcoin price ever reached?

₹ 54,404,923

# Q13

---

## What is Altcoin?

**Altcoins** are cryptocurrencies other than [Bitcoin](#).

[Altcoin - Bitcoin Wiki](#)

- Binance Coin (BNB)
- Cardano (ADA)
- Chainlink (LINK)
- Ether (ETH)
- Litecoin (LTC)

As of today, **over 5000** of these "alternative" currencies have been created worldwide.

# Q14

---

## What is meme coin?

A **meme coin** (also spelled **memecoin**) is a [cryptocurrency](#) that originated from an [Internet meme](#) or has some other humorous characteristic.<sup>[1]</sup> It may be used in the broadest sense as a critique of the cryptocurrency.

In late 2013, [Dogecoin](#) was released after being created as a joke on the [Doge](#) meme by [software engineers](#). This sparked the creation of several subsequent meme coins. In October 2021, there were about 124 meme coins circulating in the market. Notable examples include Dogecoin and [Shiba Inu](#),<sup>[2]</sup>

[Meme coin - Wikipedia](#)

# Q15

---

## What is This?



**Dogecoin** (/dəʊ(d)ʒkɔɪn/ *DOHJ-koyn* or *DOHZH-koyn*<sup>[2]</sup>) code: **DOGE**, symbol: **Ɖ**) is a cryptocurrency created by software engineers Billy Markus and Jackson Palmer, who decided to create a payment system as a "joke", making fun of the wild speculation in cryptocurrencies at the time.<sup>[3]</sup> It is considered both the first "meme coin", and, more specifically, the first "dog coin". Despite its satirical nature, some consider it a legitimate investment prospect.

[Doge \(meme\) - Wikipedia](#)

# Q16

---

## What is Stablecoins?

Cryptocurrency but usually centralized and pegged with some fiat money or asset class

**Stablecoins** are cryptocurrencies where the price is designed to be pegged to a cryptocurrency, fiat money, or to exchange-traded commodities (such as precious metals or industrial metals).<sup>[1]</sup>

[Stablecoin - Wikipedia](#)

# Q17

---

## What is Token?

**Token** is a unit of value issued by a tech or crypto start-up, intended to be a piece in the ecosystem of their technology platform or project. Tokens are supported by blockchains. They only physically exist in the form of registry entries in said blockchain. Initially, most tokens were based on the [ERC20](#) protocol by [Ethereum](#).

Tokens are different from bitcoins and altcoins in that they are not mined by their owners nor primarily meant to be traded (although they may be traded on exchanges if the company that issued them becomes valuable enough in the eyes of the public), but to be sold for fiat or cryptocurrency in order to fund the start-up's tech project.

[Token Definition – Cryptocurrency – BitcoinWiki](#)

# Q18

---

## What is MetaVerse and Omniverse?

Facebook is changing its name to Metaverse to highlight the vision of a future that will be lived in the cyberspace (along with life in the physical space). Augmented Reality and Virtual Reality, along with sensors, displays, artificial intelligence and Digital Twins, are the enabling technologies.

[Metaverse vs Omniverse – IEEE Future Directions](#)

# Q19

---

## What is Sandbox and Dreamland

The Sandbox is a sandbox game for mobile phones and Microsoft Windows, developed by gamestudio Pixowl and released on May 15, 2012. It was released for PC on Steam on 29 June 2015. The brand was acquired by Animoca Brands in 2018, and its name used for a blockchain-based 3D open world game.

**SAND** is an ERC-20 Ethereum-powered utility token that will be the medium of exchange within **The Sandbox**. Facilitates the purchase or sale of LANDs or game ASSETs (LANDs are portions of the Metaverse open to player ownership, while ASSETs are tokens created by players).

[Metaverse vs Omniverse – IEEE Future Directions](#)

---

# Q20

## What is NFT?

A non-fungible token (NFT) is a unique and non-interchangeable unit of data stored on a blockchain, a form of digital ledger. NFTs can be associated with reproducible digital files such as photos, videos, and audio. NFTs use a digital ledger to provide a public certificate of authenticity or proof of ownership, but do not restrict the sharing or copying of the underlying digital files. The lack of interchangeability (fungibility) distinguishes NFTs from blockchain cryptocurrencies, such as Bitcoin.

[Non-fungible token - Wikipedia](#)

# Q21

## What is (DAO) ?

A company or group of like-minded entities that operate based on the rules set forth in a smart contract. DAOs are used to transform business logic into software logic recorded on a blockchain. A company whose funds are locked in a multisignature wallet that is controlled by a smart contract is an example of a DAO. In that same example, board of directors decisions might be voted on, recorded, and effected through a smart contract rather than by holding physical board meetings.

# Q22

## What is ETHEREUM ?

Ethereum is a decentralized Blockchain 2.0 chain. It was the first major smart contract platform and has widespread support from Fortune 500 companies through the Ethereum Enterprise Alliance (EEA).

Ethereum currently uses a Proof-of-Work (PoW) consensus algorithm, but future changes to the protocol will update it to a more scalable algorithm, most likely based on Proof-of-Stake (PoS).

# Q23

---

## What is HASHRATE ?

The rate at which a particular machine can perform a specific hashing function. Hashrate is similar to general CPU speed, but where processor speed is measured based on the number of arbitrary instructions a machine can carry out per second, hashrate is measured based on the number of times a machine can perform that specific function per second, allowing application-specific integrated circuits (ASIC) to have a much higher hashrate than a processor with the same clock speed.

# Q24

## What is MAINNET ?

The largest blockchain network a specific protocol runs, or the most valuable chain as decided by the community. Mainnets are typically where real value is derived and represent the truest intent of the core developers.

**Q25**

---

## What is ORACLE ?

Services that connect real-world data with blockchain applications. Oracles are necessary to provide input that cannot be independently verified, such as temperature measurements. Oracles typically rely on the security of a trusted source rather than the security of trustlessness.

# Q26

---

## What is SOLIDITY ?

A smart contract programming language built for the Ethereum Virtual Machine. Syntactically it resembles C++ and Javascript and compiles to eWASM.

**Q27**

---

## What is SOLIDITY ?

A smart contract programming language built for the Ethereum Virtual Machine. Syntactically it resembles C++ and Javascript and compiles to eWASM.

# Q28

---

## What is TOKENIZATION ?

The concept of translating business strategies, goods, or services into discrete, tradeable units that are recorded on a blockchain or other system.

Physical goods can be tokenized by associating their unique identifiers with on-chain references.

# Q29

---

## What is TOKENIZATION ?

The concept of translating business strategies, goods, or services into discrete, tradeable units that are recorded on a blockchain or other system.

Physical goods can be tokenized by associating their unique identifiers with on-chain references.

**Q30**

---

## **What Is a "51% Attack"?**

A 51% attack refers to a malicious actor (or group acting in concert), controlling over 50% of the total mining power of the blockchain network and disrupting the integrity of the blockchain.

An example of a 51% attack happened in January 2019 on the Ethereum Classic blockchain.

**Q31**

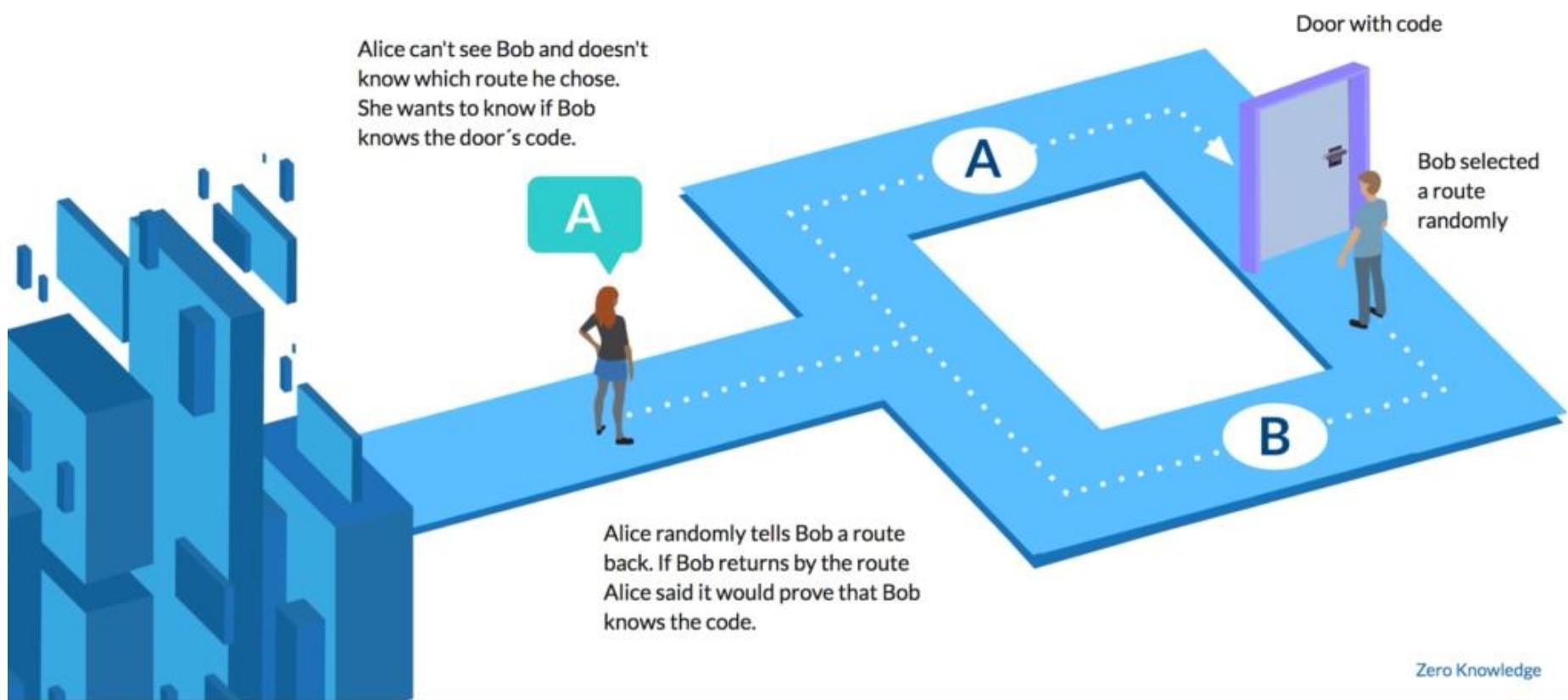
---

# ZERO-KNOWLEDGE (ZK) PROOF

A mathematical representation of an assertion whose output value can be determined without the input information.

Zero-knowledge proofs are used to prove that an actor is in possession of certain information without actually revealing that information. They are especially useful in cryptocurrencies because they can be used to show that a transaction is valid without revealing the sender, recipient, or amount of the transaction. ZK research is still in its infancy.

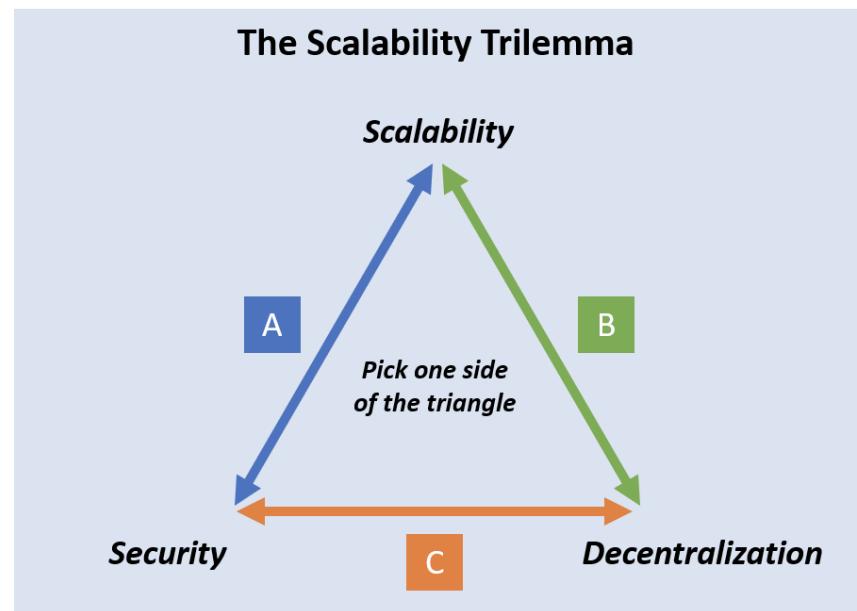
## Zero Knowledge | Intuitive Example



**Q32**

## What Is the Blockchain Trilemma?

The blockchain trilemma is a concept coined by Vitalik Buterin that proposes a set of three main issues — decentralization, security and scalability — that developers encounter when building blockchains, forcing them to ultimately sacrifice one "aspect" for as a trade-off to accommodate the other two.



# Beeple's, *Everydays* "The First 5000 Days" – \$69 Million



Beeple's *Everydays, the First 5000 days* is basically one of the most iconic NFT sales that ever happened in the history of the NFT world. It not only broke the world record but also made. ***Beeple, one of the richest artists in the world.***

**The artwork was auctioned at NFT platform, Christies on March 11, 2021.**  
Bascially, the Everydays 5000 consists of Beeple's entire collection of 5000 artworks that he created since May 1, 2007. The bid on this artwork started with \$100 but it soon rose to millions ultimately settling for \$69 Million dollars.

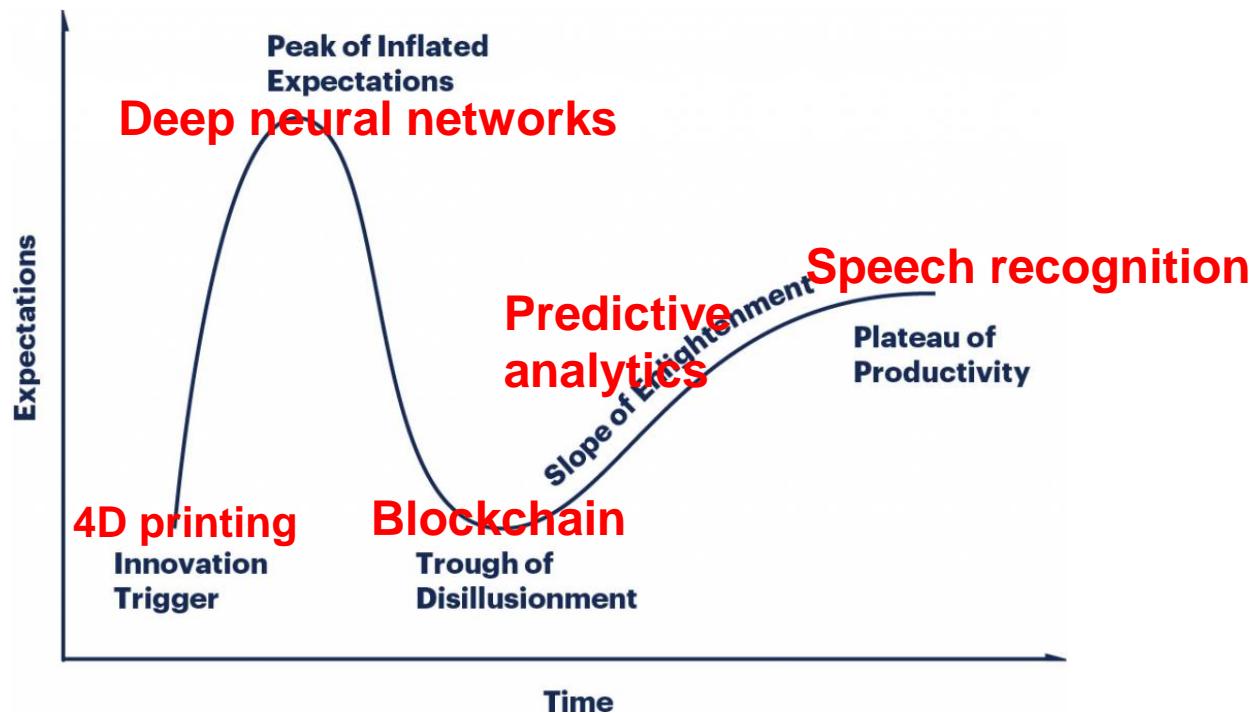
[36 MOST EXPENSIVE NFTs EVER SOLD \(Ranked\) - NFT's Street \(nftsstreet.com\)](https://nftsstreet.com)



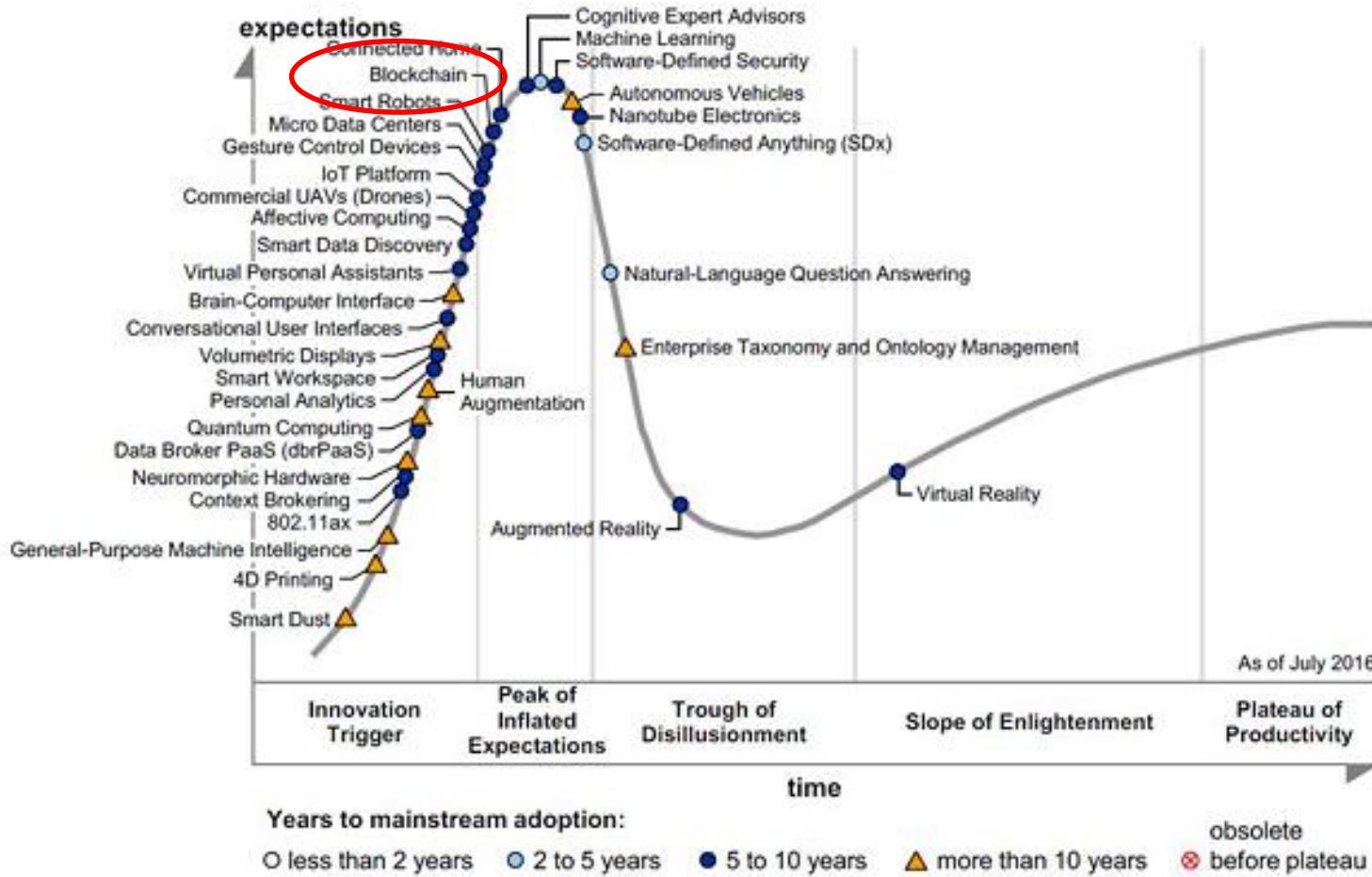
# HYPE

# Gartner Hype Cycle

- The Gartner Hype Cycle is a graphical representation of the perceived value of a technology trend or innovation—and its relative market promotion.
- The cycle can help you understand how the perceived value of a given technology evolves over the course of its maturity lifecycle.



# Hype Cycle for Emerging Technologies, 2016

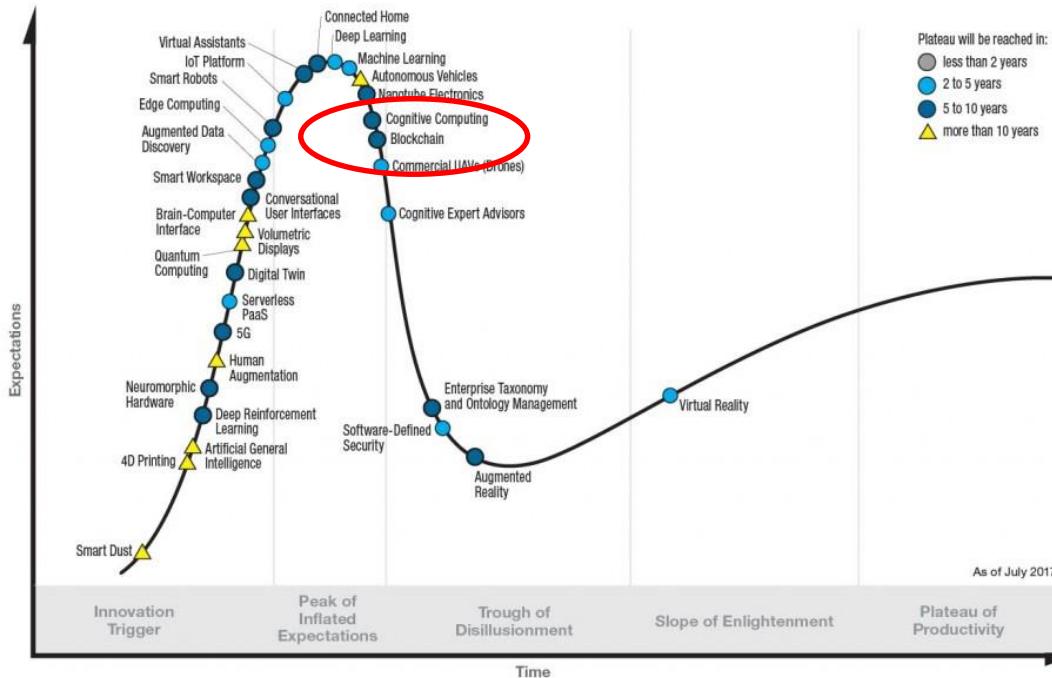


Source: Gartner (July 2016)

# Hype Cycle for Emerging Technologies, 2017



Gartner Hype Cycle for Emerging Technologies, 2017



[gartner.com/SmarterWithGartner](http://gartner.com/SmarterWithGartner)

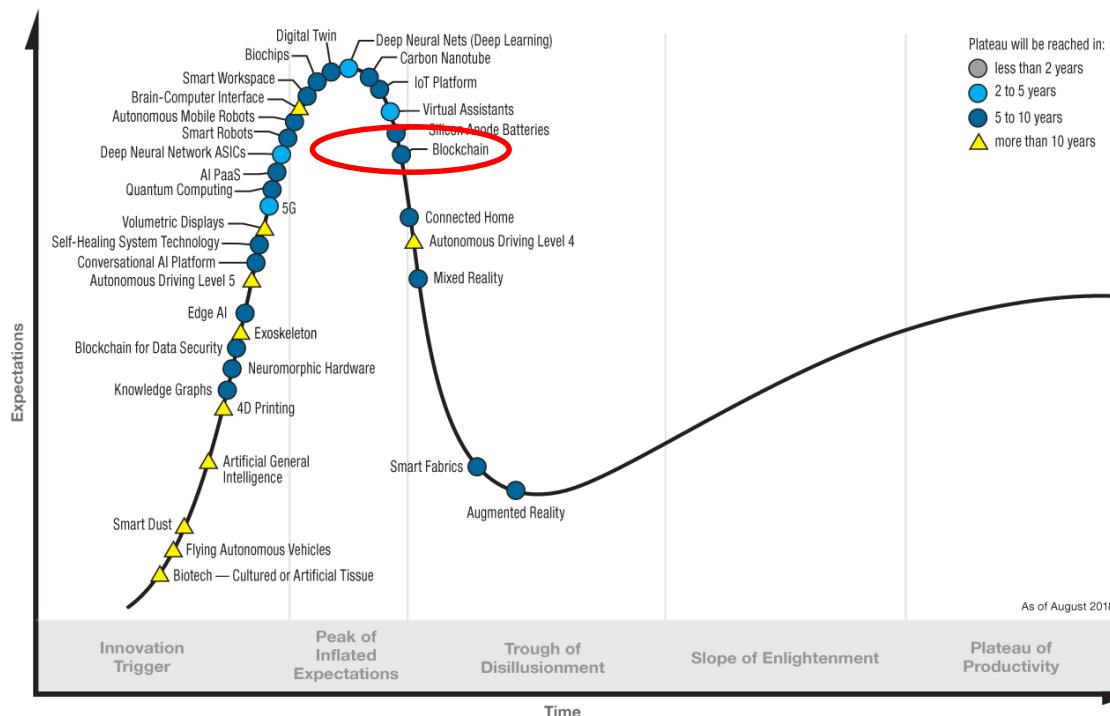
Source: Gartner (July 2017)  
© 2017 Gartner, Inc. and/or its affiliates. All rights reserved.

**Gartner**

# Hype Cycle for Emerging Technologies, 2018



## Hype Cycle for Emerging Technologies, 2018



[gartner.com/SmarterWithGartner](http://gartner.com/SmarterWithGartner)

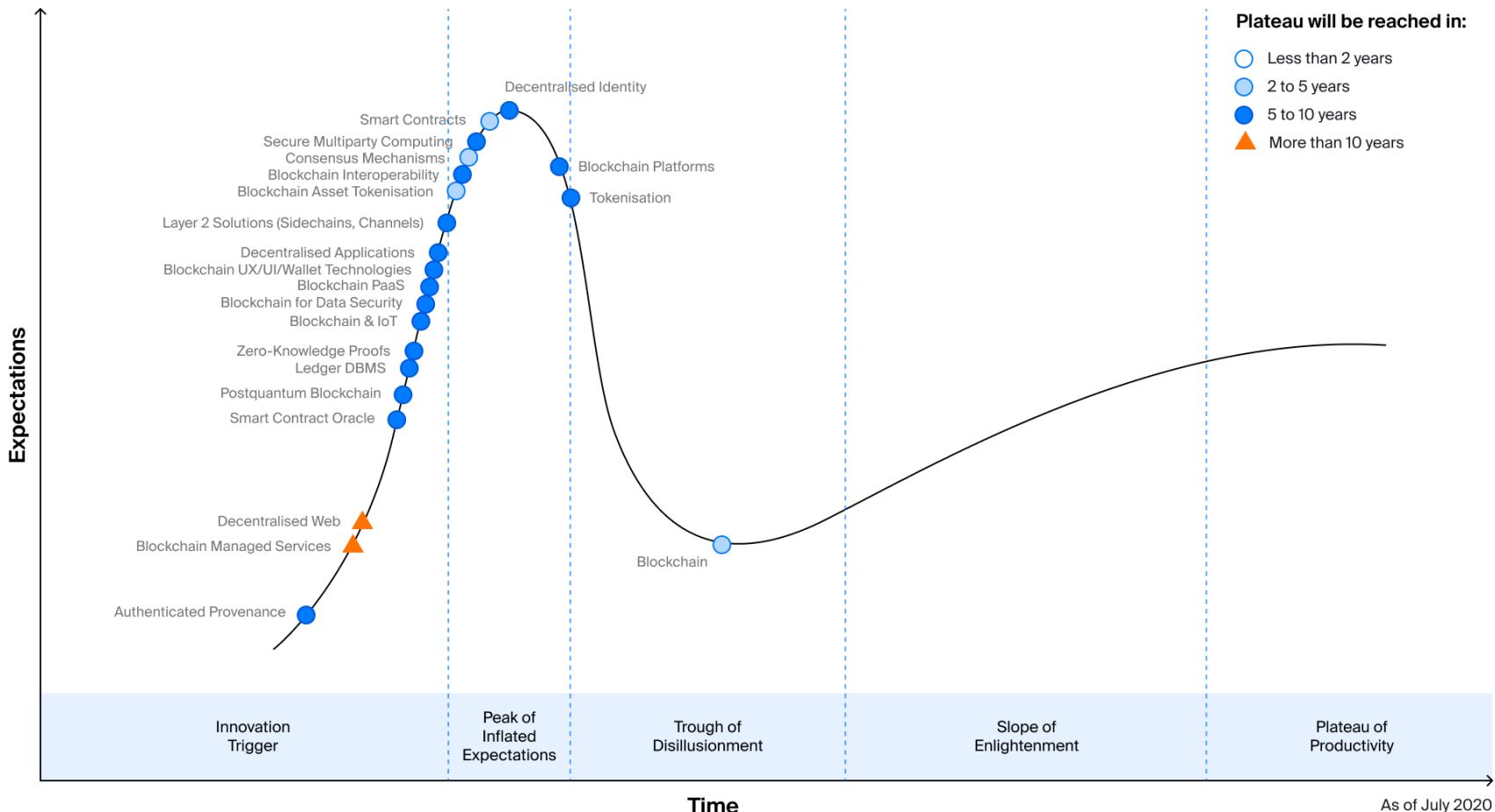
Source: Gartner (August 2018)  
© 2018 Gartner, Inc. and/or its affiliates. All rights reserved.

**Gartner**

# Hype Cycle for Blockchain Technologies 2020

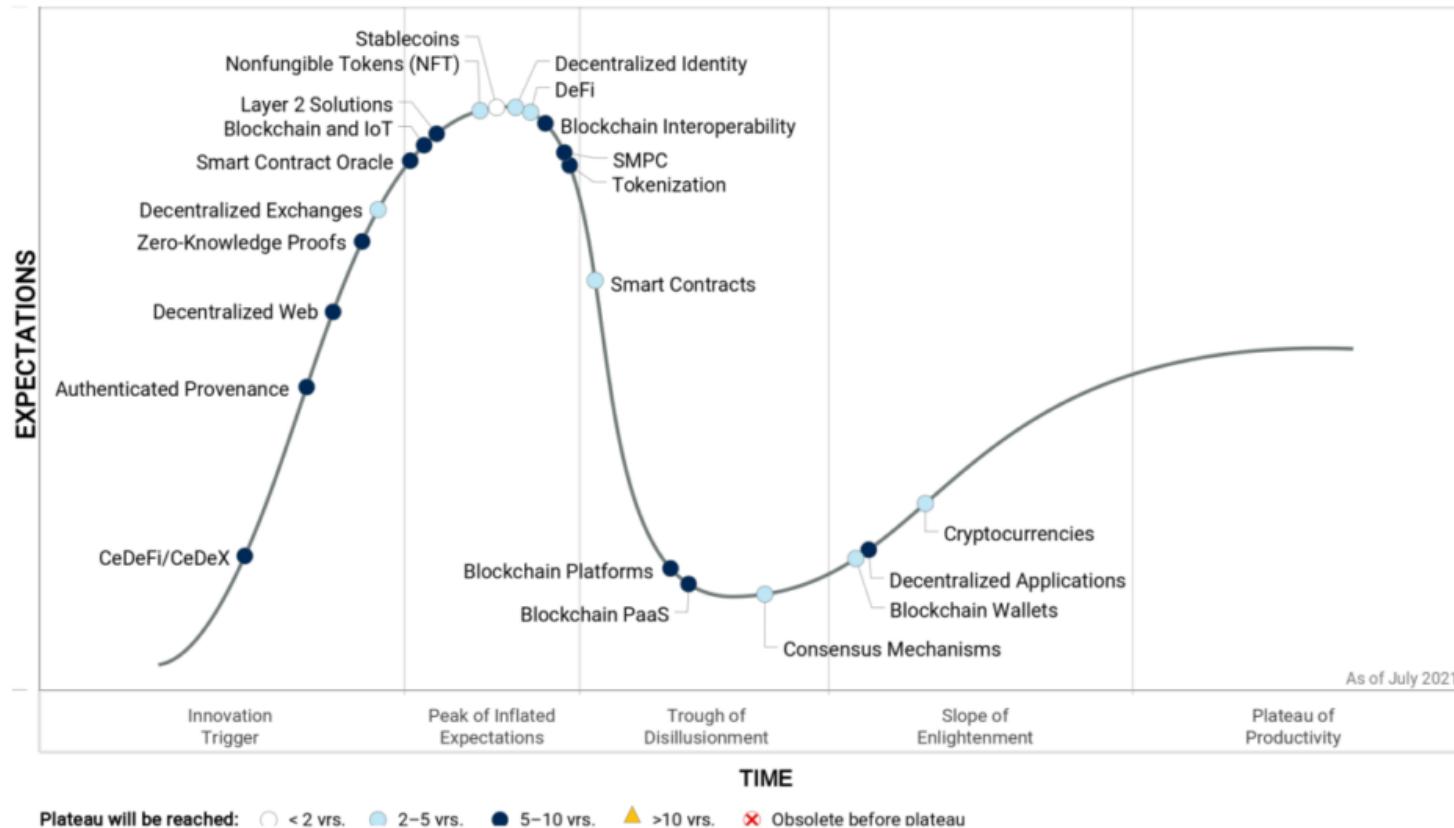


## Hype Cycle for Blockchain Technologies, 2020



# Hype Cycle for Blockchain 2021: More Action than Hype

## Hype Cycle for Blockchain, 2021



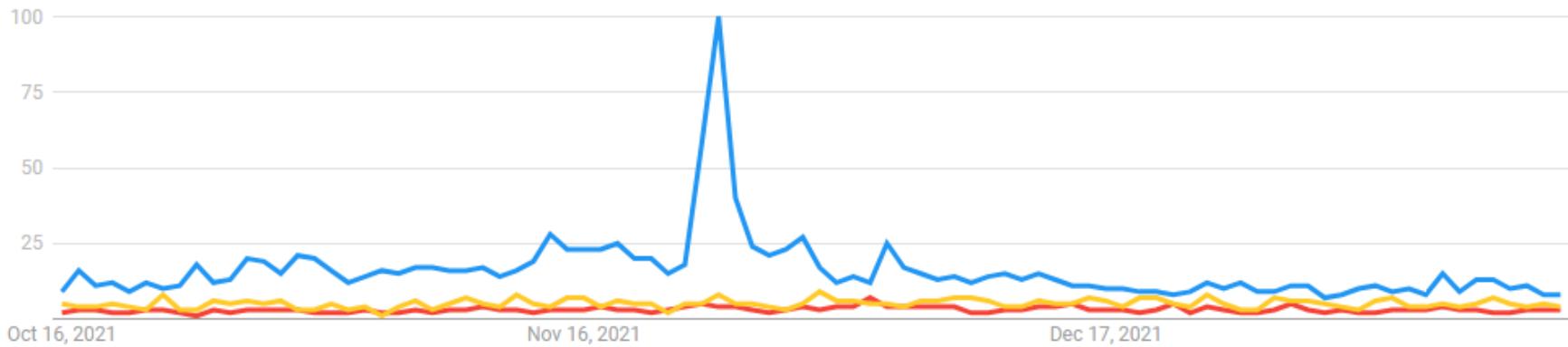
Source: Gartner (July 2021)

747513



# Trends and Facts

# Google Trends: Cryptocurrency, Blockchain ans Machine Learning



# BITCOIN Growth





# Etherium Growth



# Cryptocurrency Market Cap

Zoom 1d 7d 1m 3m 1y YTD **ALL**

Apr 28, 2013 → Jan 16, 2022



# Key adoption drivers

- Mainstream adoption of Bitcoin, including El Salvador's adoption of Bitcoin as legal tender in June 2021.
- Payment network, banking and social network adoption of distributed ledger technologies (DLTs) for money movement, with the expected deployment of central bank digital currencies (CBDCs) being a key influencer.
- Decentralized finance (DeFi) applications offer substantially greater financial rewards than traditional finance. Centralized firms like hedge funds already take advantage of this.
- Tokenization of assets, including explosive growth of NFTs and DeFi tokens, and the promise of tokens linked to physical assets in the future.
- Blockchains such as Binance, Cardano, and Solana offering viable cost-effective alternatives to Ethereum chain transactions.
- Monumental progress in blockchain interoperability, including gateways and abstraction middleware, already used today by DeFi applications.
- Blockchain migration from the proof-of-work (POW) consensus method (still used for Bitcoin) to more energy-efficient consensus methods such as proof of stake (PoS). The ongoing upgrade of Ethereum leads this trend.

# **Still, the picture is not all rosy. There are plenty of challenges**

- Adoption of permissioned blockchains is moving much more slowly. Some use cases — especially around supply chain and authenticated provenance — are benefiting from ledger technology. However, most users are stuck trying to align use cases to the technology.
- Global regulations and accounting standards need clarification before most enterprises adopt cryptocurrency
- China continues to clamp down on crypto activities as they work on making their own CBDC the world's dominant currency.



**BITS** Pilani  
Pilani Campus

# Blockchain Technology and Systems

(SEZG569/SSZG569)

Dr. Ashutosh Bhatia  
Department of Computer Science and Information Systems



# Why a course on Blockchain?

# What is all the excitement about?

---

- 1) **Basic application:** a digital currency (stored value)
    - Current largest: Bitcoin (2009), Ethereum (2015)
    - Global: accessible to anyone with an Internet connection
  - 2) **Beyond stored value:** decentralized applications (DAPPs)
    - **DeFi:** financial instruments managed by public programs (examples: stablecoins, lending, exchanges, ....)
    - **Asset management (NFTs):** art, game assets, domain names.
    - **Decentralized organizations (DAOs):** (decentralized governance) DAOs for investment, for donations, for collecting art, etc.
  - 3) **New programming model:** writing decentralized programs
-

# What is a blockchain?

Abstract answer: a blockchain provides

- coordination between many parties,
- when there is no single trusted party

if trusted party exists ⇒ no need for a blockchain

[financial systems: often no trusted party]

# What is Blockchain?

---

- ❑ A Linked List
  - ❑ Replicated
  - ❑ Distributed
  - ❑ Consistency maintained by Consensus
  - ❑ Cryptographically linked
  - ❑ Cryptographically assured integrity of data
  
- ❑ Used as
  - ❑ Immutable Ledger of events, transactions, time stamped data
  - ❑ Tamper resistant log
  - ❑ Platform to Create and Transact in Cryptocurrency
  - ❑ log of events/transactions unrelated to currency

# What is a blockchain?

LAYER 3

**User Interface** (e.g., web3)

LAYER 2

**Applications** (Solidity, Move, Motoko)

LAYER 1.5

**Compute Layer** (blockchain computer)

LAYER 1

**Consensus Layer**

# Layer 1: Consensus Layer (Informal)

A public append-only data structure:

- **Persistence:** once added, data can never be removed\*
- **Safety:** all honest participants have the same data\*\*
- **Liveness:** honest participants can add new transactions
- **Open(?)**: anyone can add data (no authentication)

achieved by replication



LAYER 1

Consensus Layer

# This Not a New Problem ...

## State machine replication: studied since the 1980s

Google, Amazon, Banks, all have lots of servers:

- need to ensure state is consistent across all servers
- Known # of servers, and all are authorized.

## New aspect: open consensus

- Anyone can write new data to the blockchain
- Was shown to be impossible! [Barak, Canetti et. al 05]
- Nakamoto [2008]
  - A way to bypass the bound using proof of work

# Open Consensus: How?

PROOF-OF-WORK	PROOF-OF-STAKE	PROOF-OF-SPACE	MANY MORE IDEAS
<p>First party to solve puzzle creates next block</p> <ul style="list-style-type: none"> <li>• sybil resistant selection of a random party</li> </ul> <p>Problems:</p> <ul style="list-style-type: none"> <li>• slow, wastes energy</li> </ul> <p><b>bitcoin</b> </p> <p> ethereum</p>	<p>Fast block creation</p> <p>No energy waste</p> <p>But more complex</p> <p> ethereum</p> <p> Tendermint</p> <p> DFINITY</p> <p> celo</p> <p> Algorand™</p>	<p> chia</p> <p> Filecoin</p>	<p> AVAX</p> <p> SOLANA</p>

# Layer 1.5: The blockchain computer



**APP logic is encoded in a program that runs on blockchain**

- Rules are enforced by a public program (public source code)  
⇒ transparency: no single trusted 3<sup>rd</sup> party
- The DAPP program is executed by parties who create new blocks  
⇒ public verifiability: everyone can verify state transitions

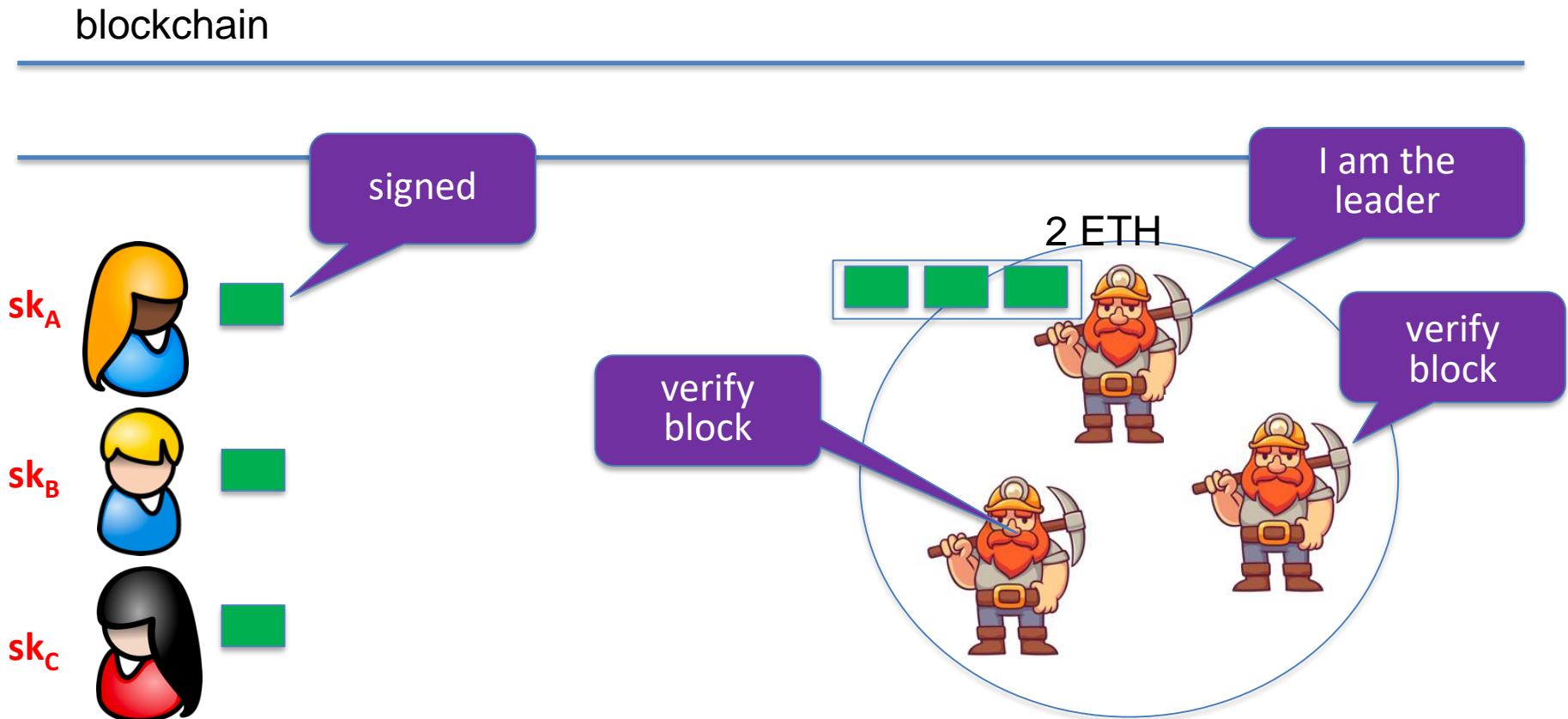
LAYER 1.5

Compute Layer (blockchain computer)

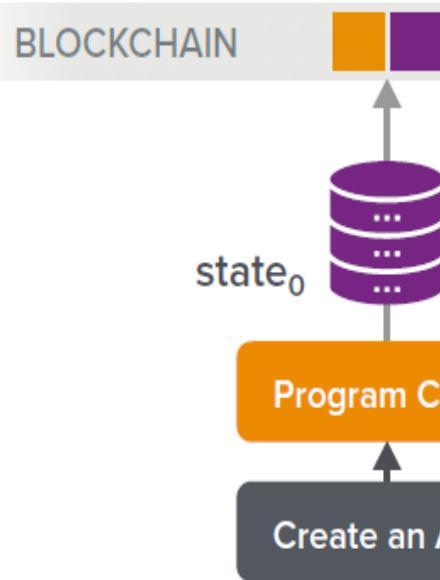
LAYER 1

Consensus Layer

# How are blocks added to chain?



# Running Programs on a Blockchain (APPs)



LAYER 1.5

Compute Layer (blockchain computer)

LAYER 1

Consensus Layer

# Execution Environment

## BITCOIN SCRIPT

### LIMITED COMPUTING ENVIRONMENT

- Limited instruction set (no loops)
- Sufficient for some tasks:
  - atomic swaps
  - payment channels, ...

## ETHEREUM

### GENERAL PROGRAMMING ENVIRONMENT (SOLIDITY)

- EVM is a general purpose computing environment
- APP code updates internal state in response to transactions
- Calling APP costs fees (gas)
  - prevents DoS on miners
  - **storing on-chain state costs fees**

# Decentralized Applications (DAPPs)



Run on  
blockchain  
computer



LAYER 2

**Applications** (Solidity, Move, Motoko)

LAYER 1.5

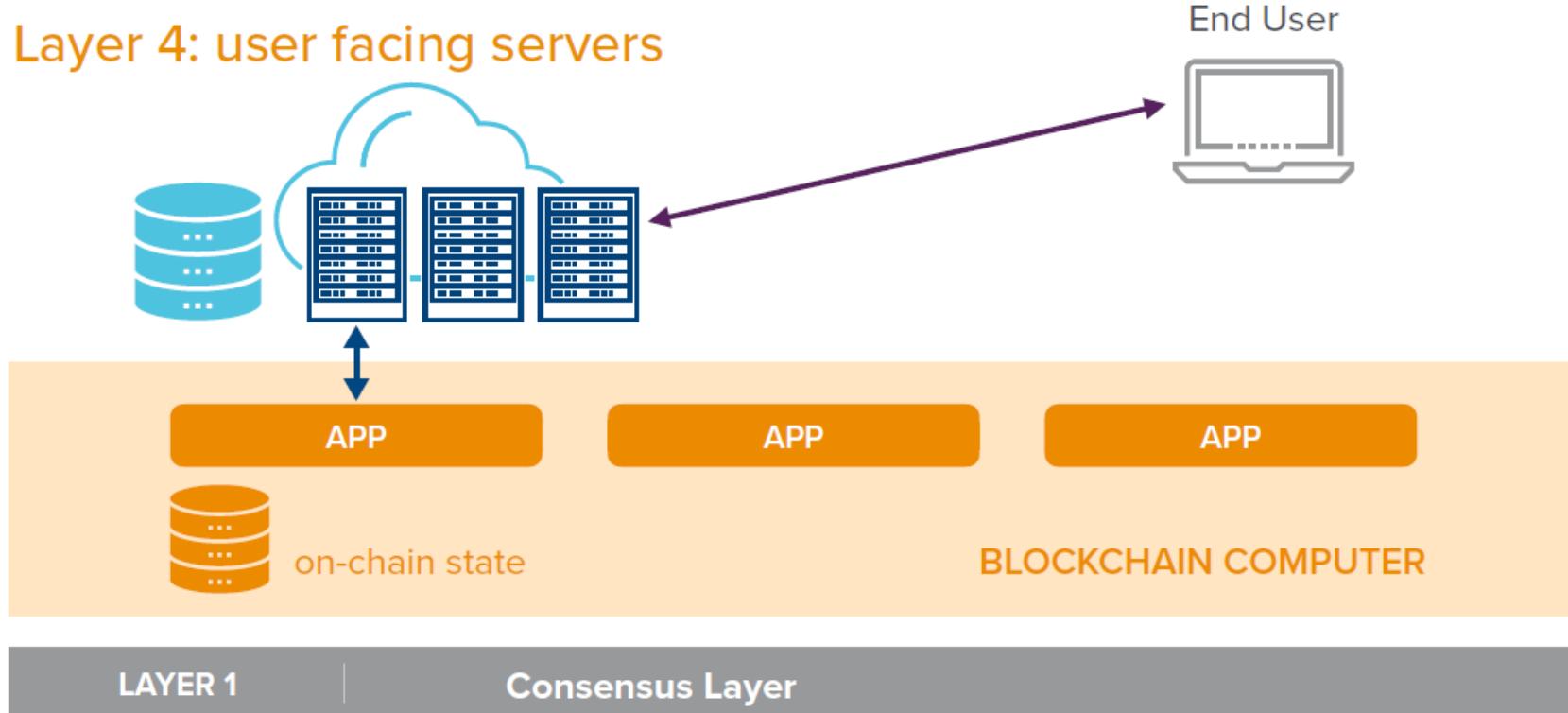
**Compute Layer** (blockchain computer)

LAYER 1

**Consensus Layer**

# Common App Architecture

Layer 4: user facing servers



# Ethereum DeFi

innovate

achieve

lead

## Ethereum's DeFi

### Payments



Protocol  
∞x



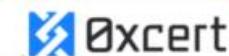
### OPEN PLATFORM



### Custodial Services



### Infrastructure



### Exchanges & Liquidity



### Investing



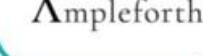
### KYC & Identity



### Derivatives



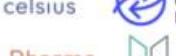
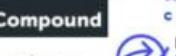
### Stablecoins



### Insurance



### Credit & Lending



### Marketplaces



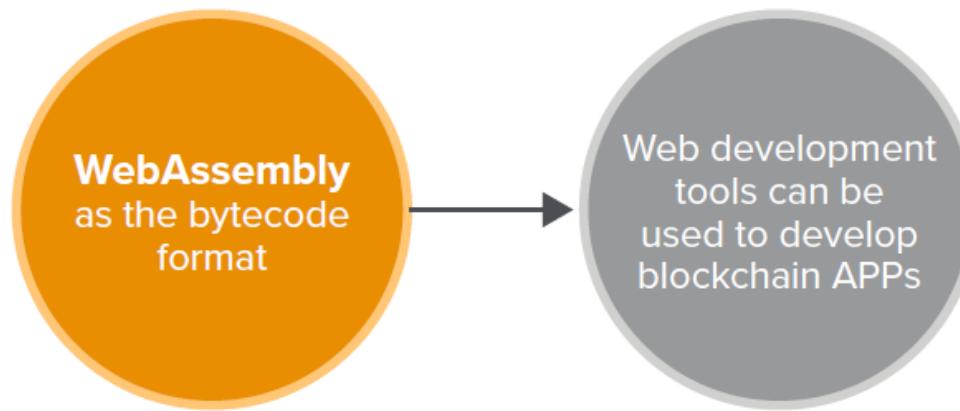
### Prediction Markets



# General Execution Environments



Recent projects

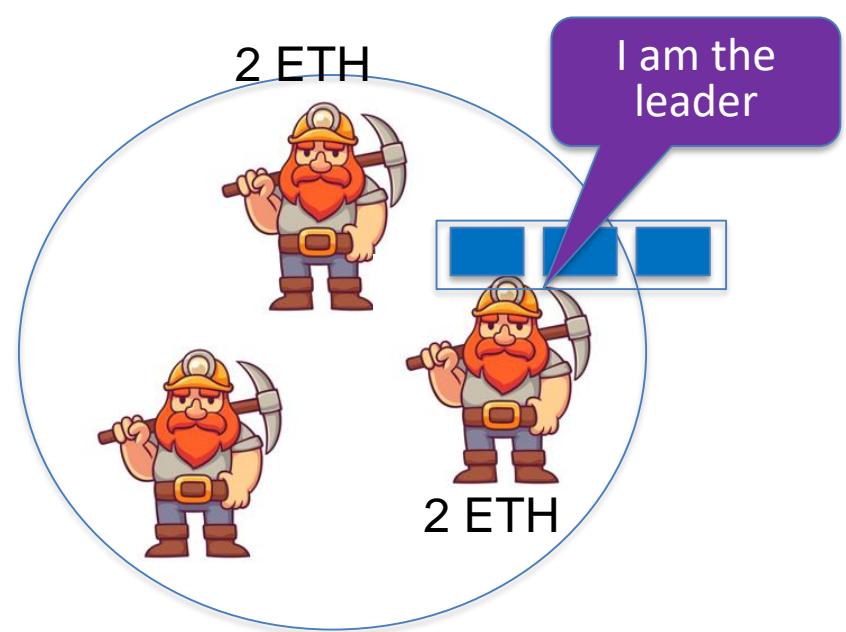
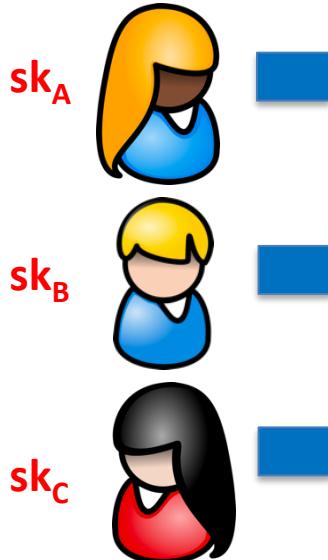


# How are blocks added to chain?

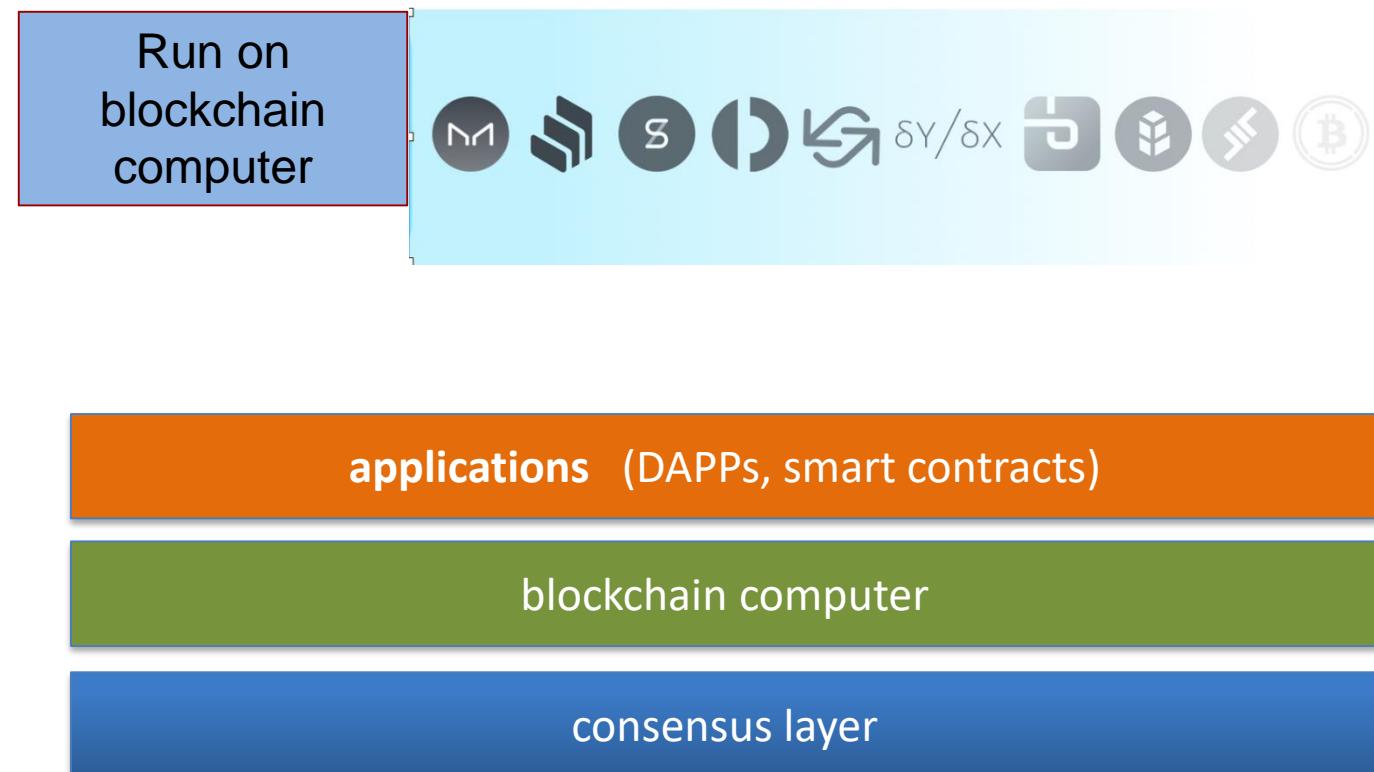
blockchain



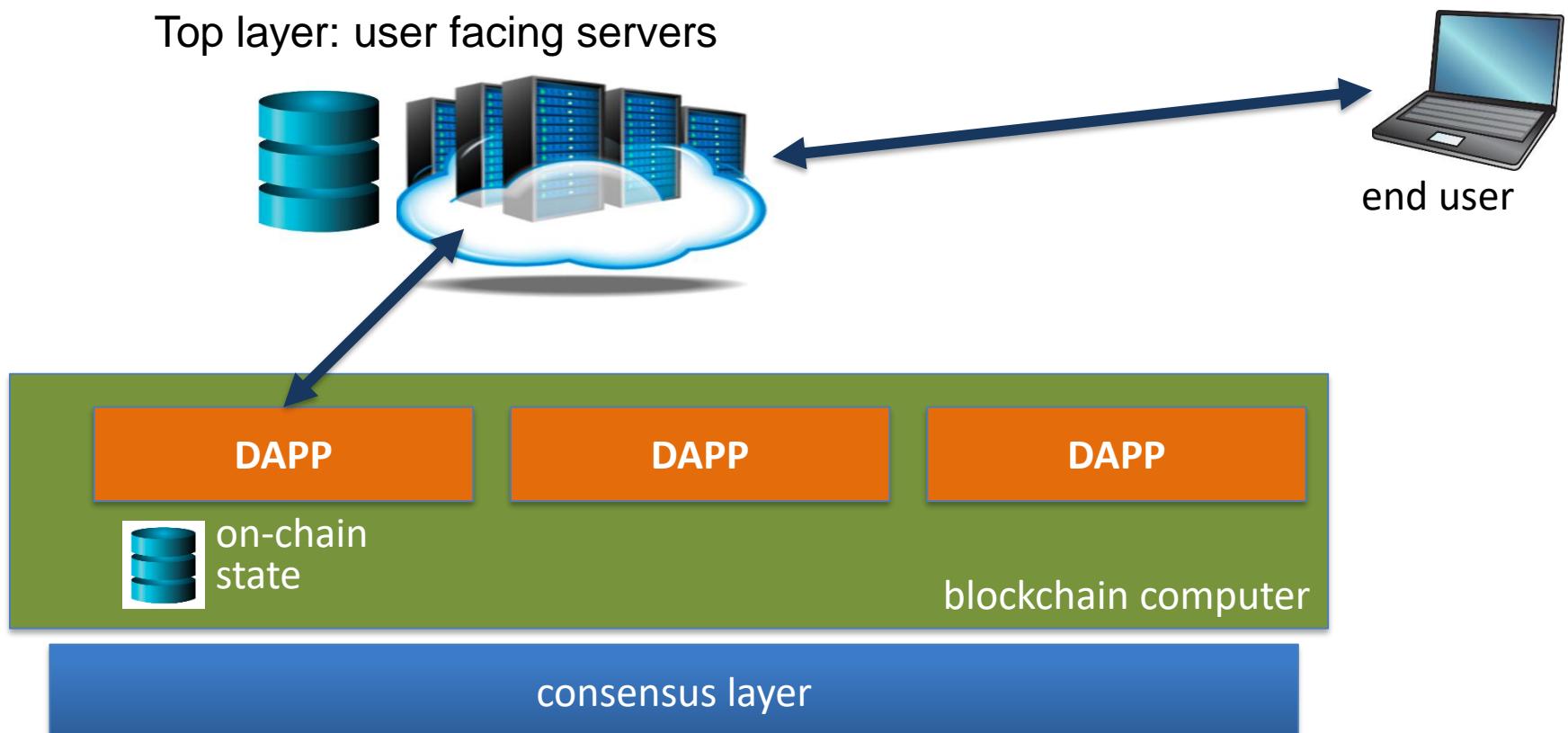
...

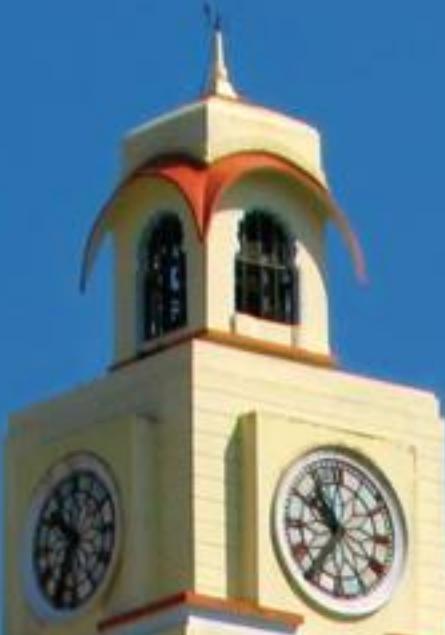


# Decentralized applications (DAPPS)



# Common DAPP architecture





**BITS** Pilani  
Pilani Campus

# Blockchain Technology (BITS F452)

Dr. Ashutosh Bhatia, Dr. Kamlesh Tiwari  
Department of Computer Science and Information Systems



# *Introduction to Crypto and Cryptocurrency*

# LECTURE OUTLINE

---

- Crypto Background
  - Hash Functions
  - Digital Signatures and its Applications
- Introduction to cryptocurrency
  - Basic digital cash

# Hash Functions

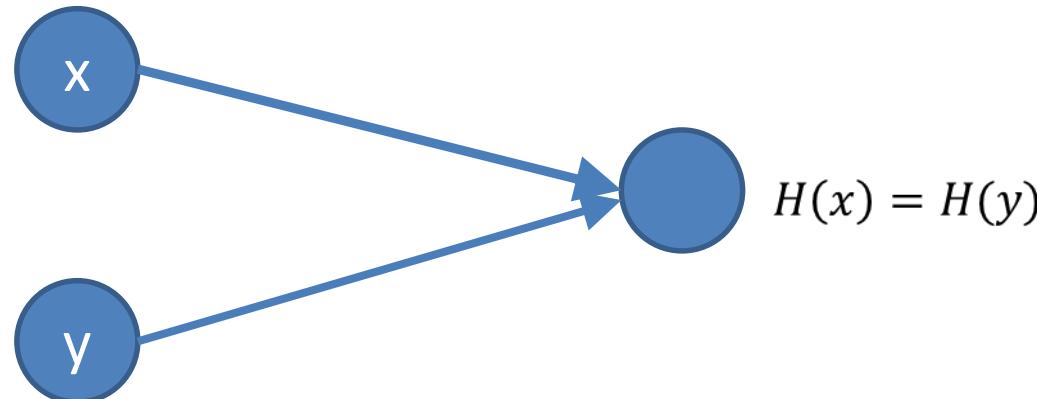
---

- Takes arbitrarily length of string as input
- Produces a fixed sized output
- Efficiently Computable
  
- Security Properties
  - Collision Free
  - Hiding
  - Puzzle friendly

# Hash Properties 1: Collision Resistant



- A collision occurs when two distinct inputs produce the same output
- **Collision-resistance:** A hash function  $H$  is said to be collision resistant if it is infeasible to find two values,  $x$  and  $y$ , such that  $x \neq y$  and  $H(x) = H(y)$



- However, collision do exist

# How to find a collision?

---

- Try  $2^{130}$  randomly chosen inputs and assuming that hash output is 256 bits, 99.8% chance that two of them will collide
  
  - This works, no matter what the hash function is. (**Birthday Paradox**)
  
  - However,  $2^{130}$  is a so large number and any computer ever made by the humanity was trying to find a collision since the beginning of the universe till now, the probability of it finding a collision is infinitesimally small.
-

# How to find a collision?

---

- Try  $2^{130}$  randomly chosen inputs and assuming that hash output is 256 bits, 99.8% chance that two of them will collide
  
- This works, no matter what the hash function is. (**Birthday Paradox**)
  
- However,  $2^{130}$  is a so large number and any computer ever made by the humanity was trying to find a collision since the beginning of the universe till now, the probability of it finding a collision is infinitesimally small.

# Birthday Paradox

**Find the probability that at-least two people in a room have the same birthday**

*Event A: at least two people in the room have the same birthday*

*Event A': No people in the room have the same birthday*

$$\Pr[A] = 1 - \Pr[A']$$

$$\begin{aligned}\Pr[A'] &= 1 \times \left(1 - \frac{1}{365}\right) \times \left(1 - \frac{2}{365}\right) \times \left(1 - \frac{3}{365}\right) \dots \left(1 - \frac{Q-1}{365}\right) \\ &= \prod_{i=1}^{Q-1} \left(1 - \frac{i}{365}\right)\end{aligned}$$

$$\Pr[A] = 1 - \prod_{i=1}^{Q-1} \left(1 - \frac{i}{365}\right)$$

$$\sqrt{Q} \approx 2M \ln \frac{1}{1-\epsilon}$$

M = 365,  $\epsilon$  is the desired

if  $\epsilon = .5$  then  $Q \approx 1.17 \sqrt{M}$

Thus to achieve 128 bit security against collision attacks, hashes of length at-least 256 is required

# Is there a better way?

---

- For some possible Hash functions, YES
  - Example  $H(x) = x \bmod 2^{256}$
- For others we don't know one
- No Hash Function is proven to be collision resistant

# Application: Hash as a message digest

---



- If we know that  $H(x) = H(y)$   
it is safe to assume that  $x = y$
  - To recognize a file that we saw before  
just remember its hash
  - Useful as the hash is small
-

# Hash Property 2: Hiding

- We want something like this  
given  $H(x)$  it is infeasible to find  $x$ .
- The problem is that this property can not be true in the stated form if the number of possible input values is small
- **Hiding:** A hash function  $H$  is hiding if: when a secret value  $r$  is chosen from a probability distribution that has **high min-entropy**, then given  $H(r | x)$  it is infeasible to find  $x$ .
- High min-entropy means that the distribution is very spread out and no particular value is chosen with negligible entropy.

# Application: Commitment

---

We want to "seal a value" in the envelop  
and "open the envelop" later

Commit to a value and reveal it later

# Commitment API

---

```
(com, key) := commit(msg)  
    match := verify(com, key, msg)
```

To seal msg in envelop

(com, key) := commit(msg), then publish com

To open envelop

publish key, msg

Anyone can use verify() to check the message

---

# Commitment API

---

```
(com, key) := commit(msg)
  match := verify(com, key, msg)
```

## Security Properties

**Hiding:** Given com, infeasible to find msg

**Binding:** Infesible to find msg  $\neq$  msg' s.t.

$$\text{verify}(\text{commit}(\text{msg}), \text{msg}') = \text{true}$$

# Commitment API

---

$\text{commit}(\text{msg}) := (\text{H}(\text{key} \mid \text{msg}), \text{key})$

`where key is a random 256 bit value

$\text{verify}(\text{com}, \text{key}, \text{msg}) = (\text{H}(\text{key} \mid \text{msg}) == \text{com})$

## Security Properties

**Hiding:** Given  $\text{H}(\text{key} \mid \text{msg})$ , infeasible to find msg

**Binding:** Infeasible to find  $\text{msg}' \neq \text{msg}$  s.t.

$$\text{H}(\text{key} \mid \text{msg}) == \text{H}(\text{key} \mid \text{msg}')$$

# Hash Property 3: Puzzle friendly

---

For every possible output value  $y$ ,

if  $k$  is chosen randomly from a distribution with high min entropy,

then it is infeasible to find  $x$  such that  $H(k | x) = y$

# Application: Search Puzzle

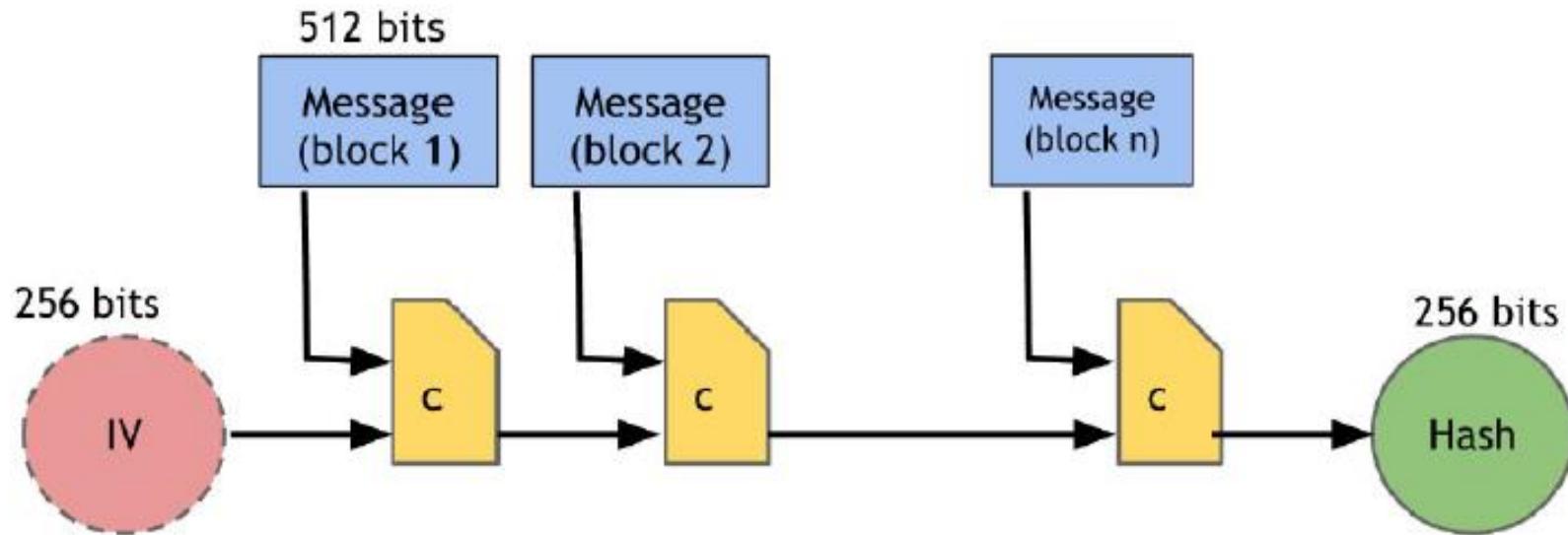
---

Given a puzzle ID,  $\text{id}$  (from high min-entropy dist.)  
and a target set  $Y$

Try to find a solution  $x$ , such that  
 $H(\text{id} | x) \in Y$

Puzzle friendly property implies that no solving strategy is much better than trying random values of  $x$ .

# SHA 256 hash function

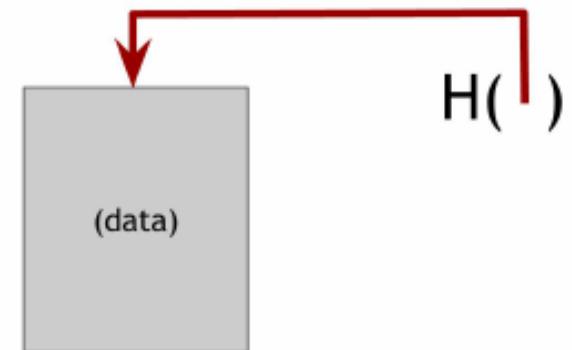


**Theorem:** If  $c$  (the compression function) is collision-free than SHA-256 is collision free

[Blockchain Demo \(andersbrownworth.com\)](http://andersbrownworth.com)

# Hash Pointer

- Hash Pointer is :
  - pointer to where some information is stored
  - cryptographic hash of the information
  
- If we have a hash pointer, we can
  - ask to get the info back
  - verify that it has not changed

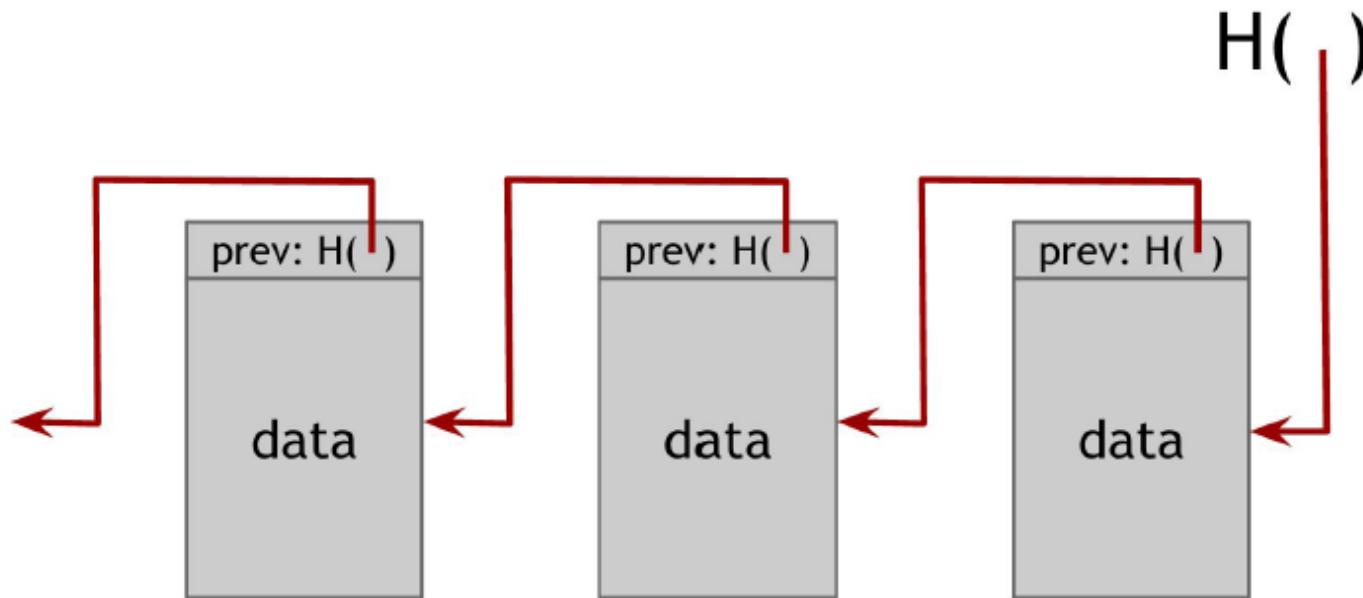


---

Key Idea

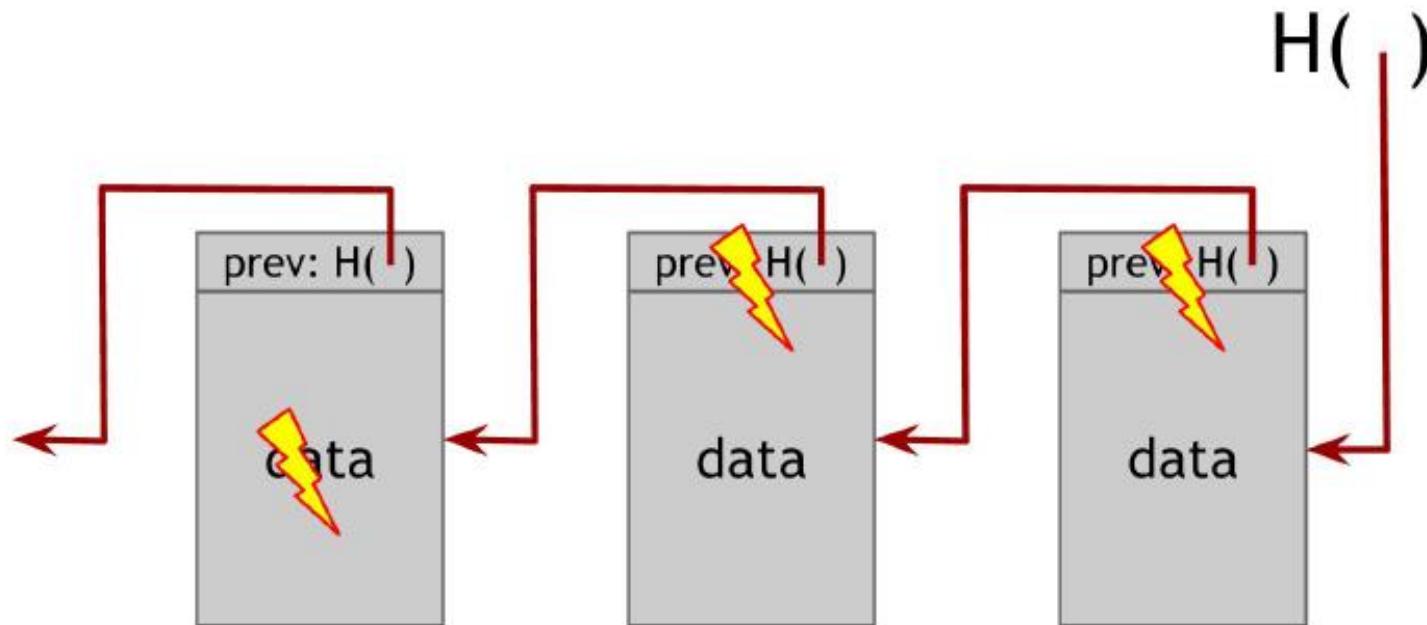
Build Data Structures with Hash Pointers

# Linked List

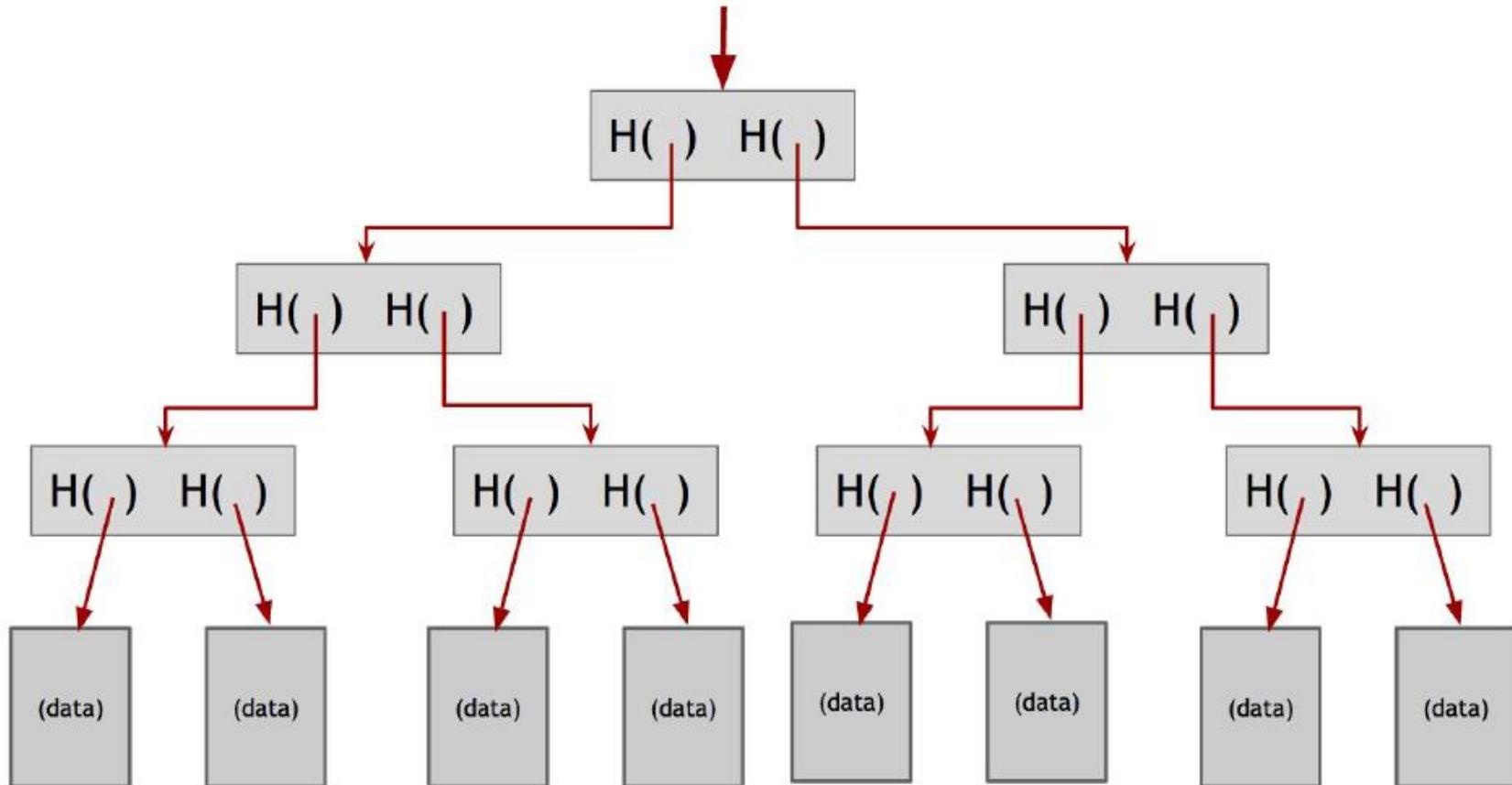


A **blockchain** is a linked list that is built using hash pointers instead of pointers

# Linked List: Tampering Detection

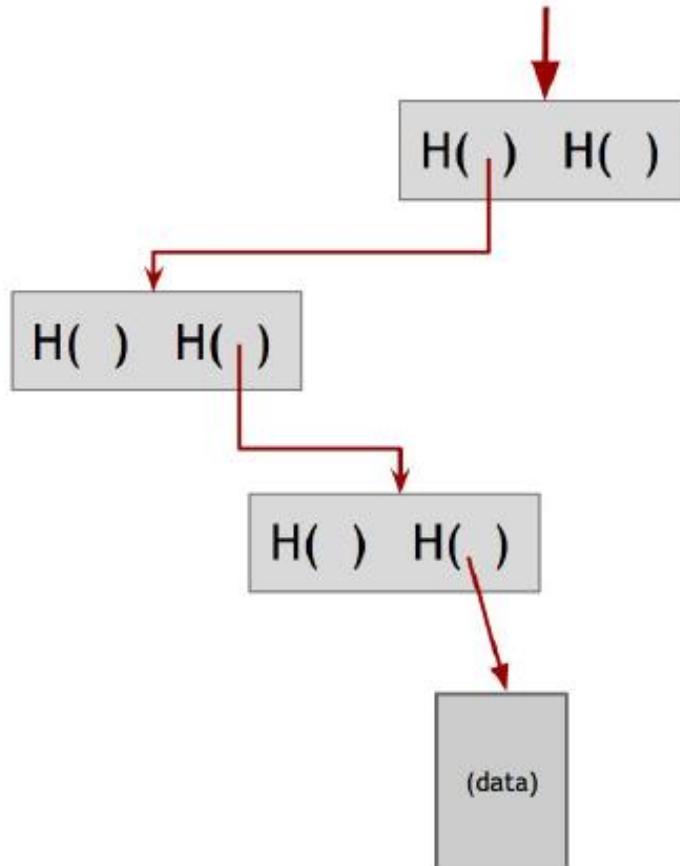


# Binary Tree With Hash Pointers: Merkle Tree



<https://prathamudeshmukh.github.io/merkle-tree-demo/>

# Proof of Membership in a Merkle Tree



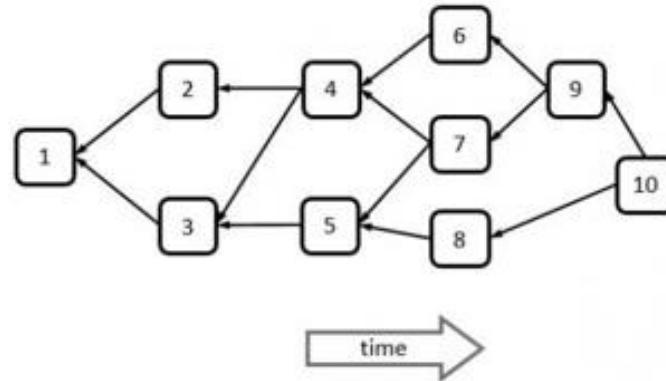
# Advantages of Merkle Tree

Tree holds many items but just need to remember the root hash

Can verify the membership in  $O(\log n)$  time

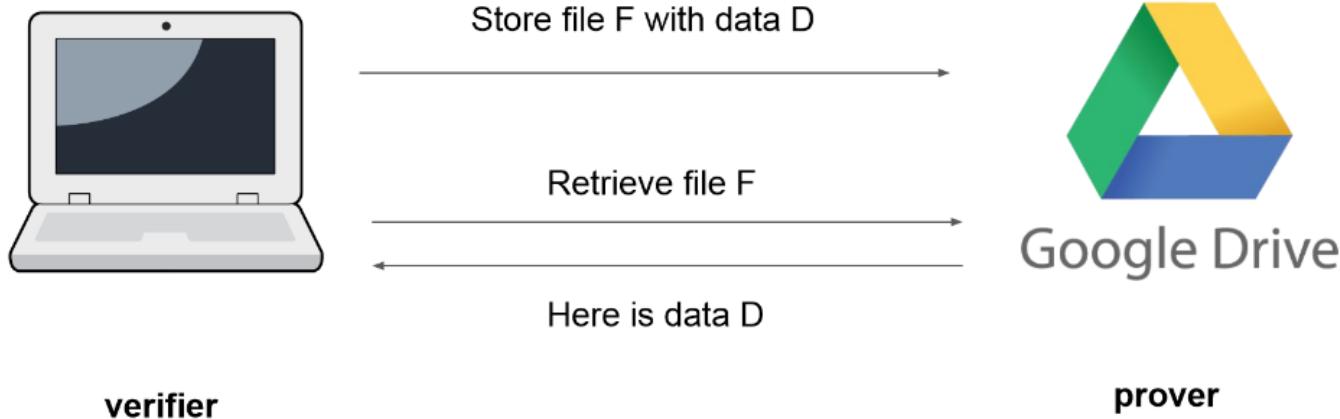
More generally we can use hash pointers in any pointer based data structure that has no cycle.

In case of cycles there is no node to start



# The storage problem

- Client wants to store a file on the server
- File has a name **F** and data **D**
- Client wants to retrieve **F** later



# The storage: Basic Protocol

---

- Client sends F with Data D to server
- Server stored (F, D)
- Client deletes D
- Client requests F from server
- Server returns D
- Client has recovered D

# The storage protocol Against Adversaries

---

- What if server is adversarial and returns  $D' \neq D$
- Trivial solution
  - Client does not delete  $D$
  - Whenever server returns  $D'$  client can compare  $D$  and  $D'$

What is client does not have memory to store data for a long time?

---

# The storage : Hash based protocol

---



- Client send file F with Data D to the server
- Server stores (F,D)
- Client stored H(D), deletes D
- Client requests F from server
- Server returns D'
- Client compares  $H(D) = H(D')$

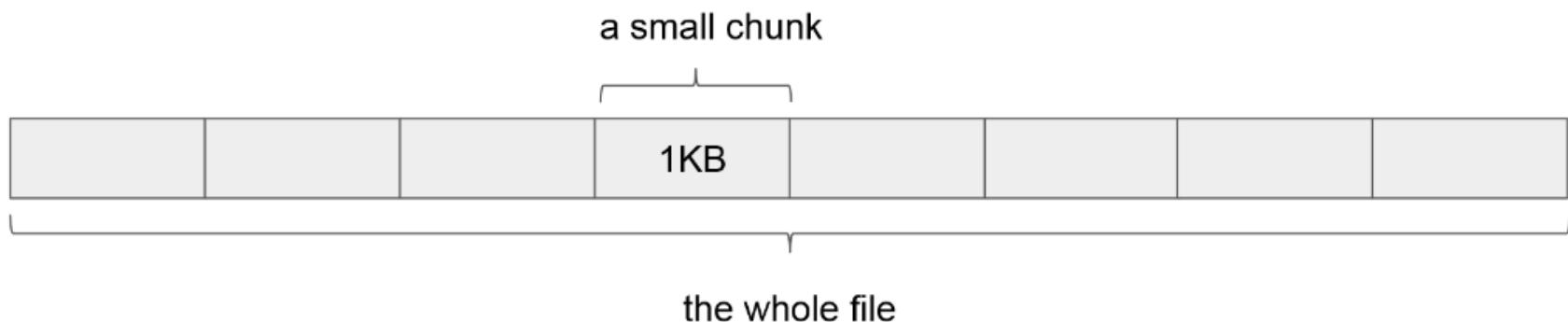
# The storage : File chunks

---

- What if client wants to retrieve the 19007<sup>th</sup> byte of the file
- Must download the whole file
- Merkle tree to rescue.

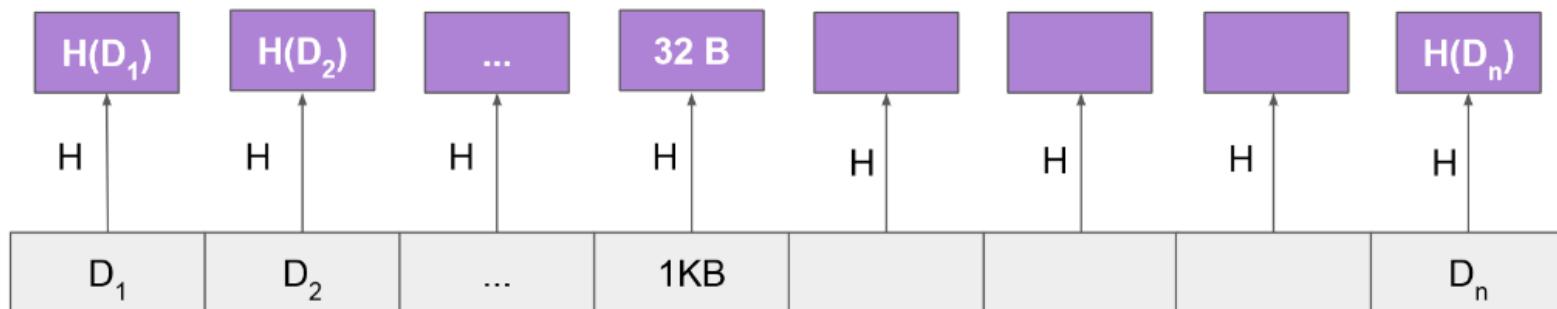
# Merkle Tree:

- Splits file into chunks (say 1 KB)



# Merkle Tree:

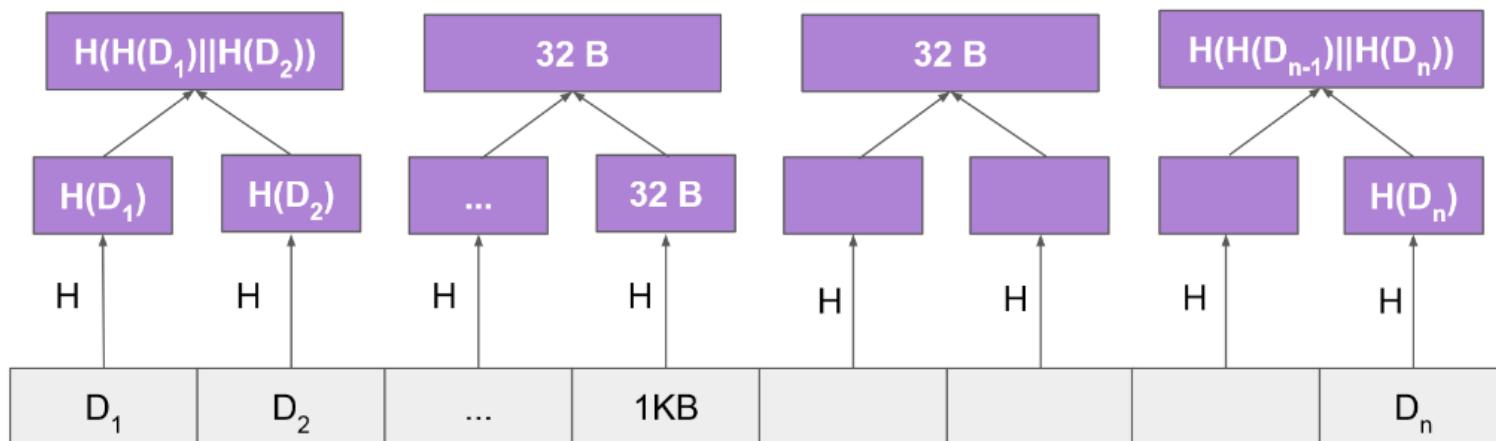
- Hash each chunk using cryptographic hash function



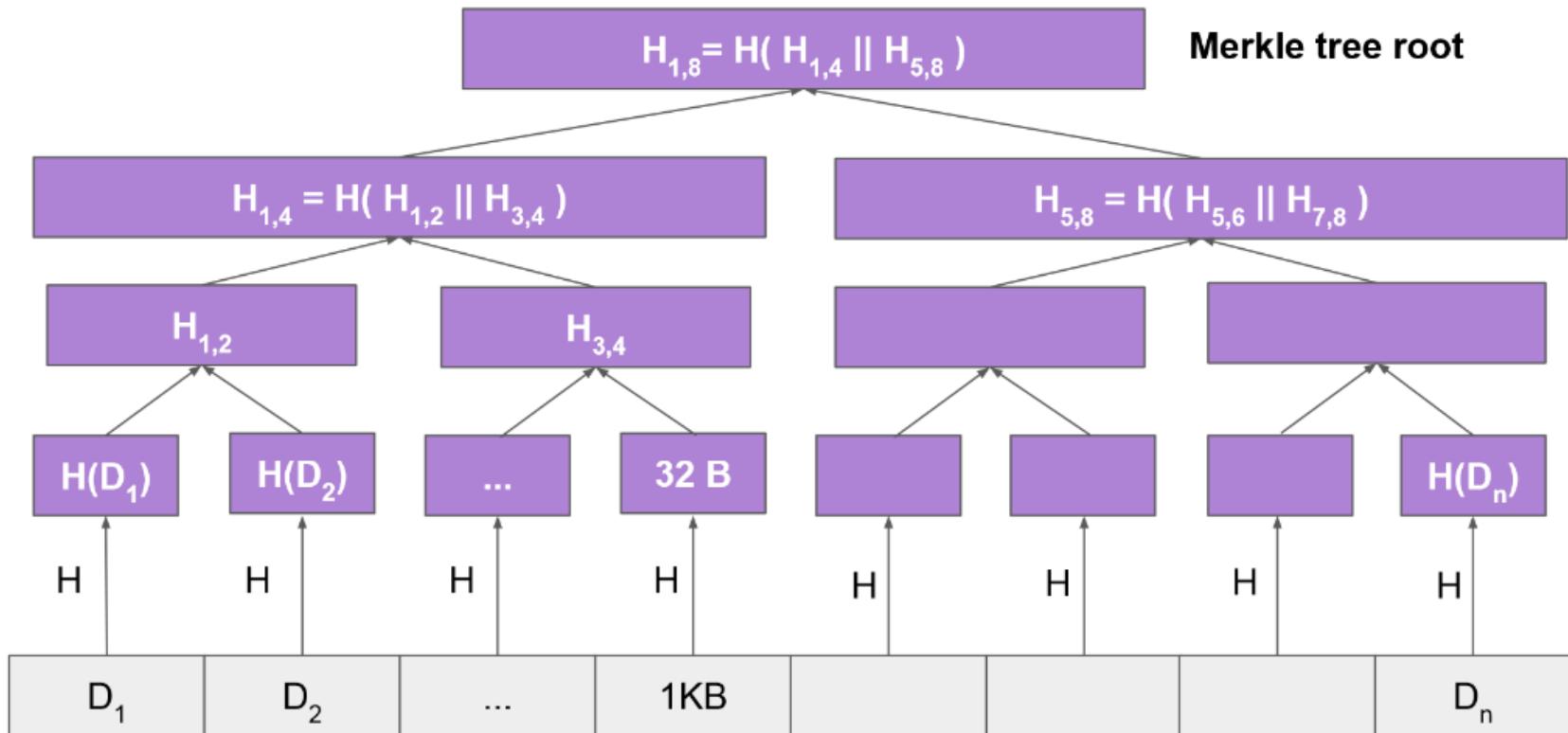
Arrows show direction of hash function application

# Merkle Tree:

- Combine them to create a binary tree
- Each node stores the hash of the concatenation of their children



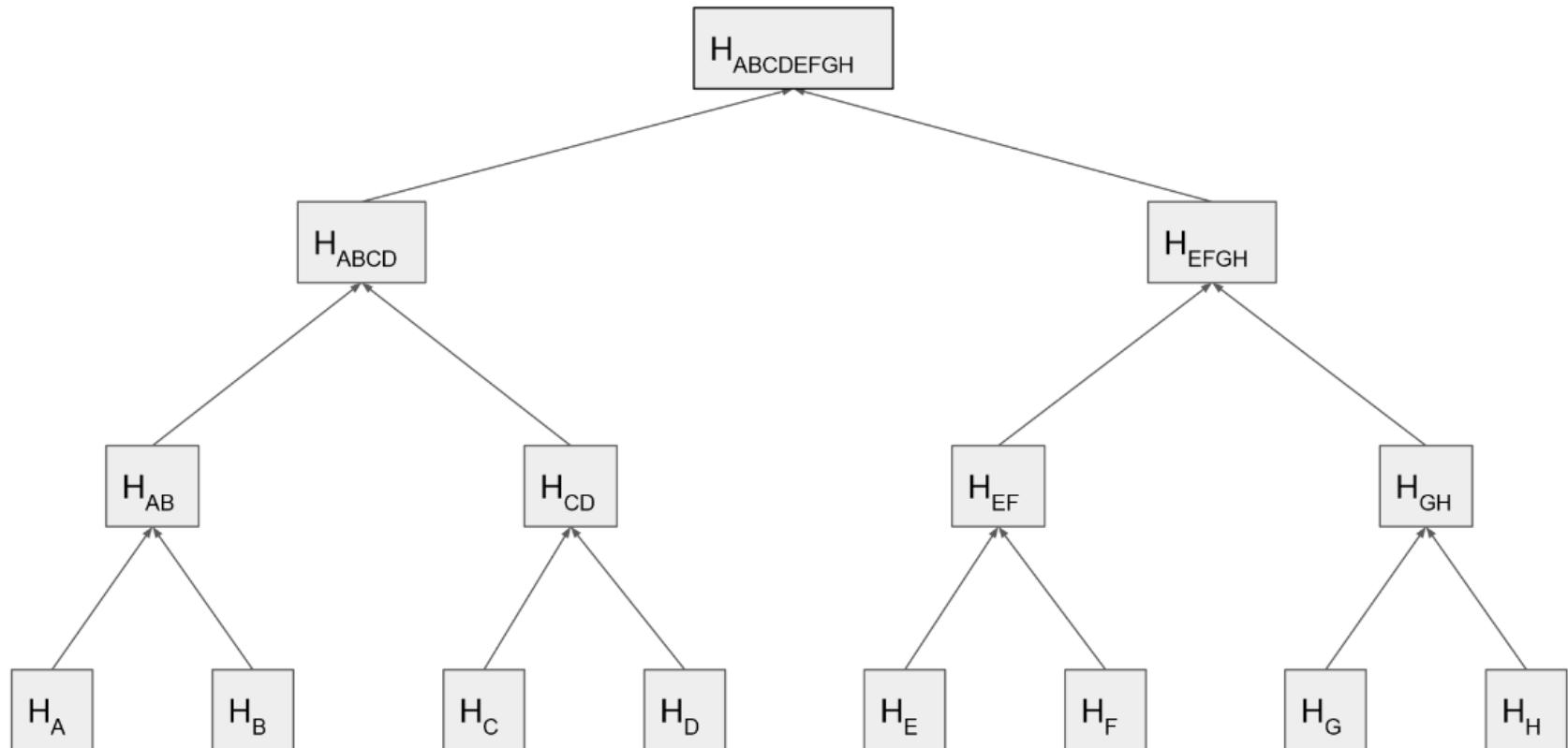
# Merkle Tree:



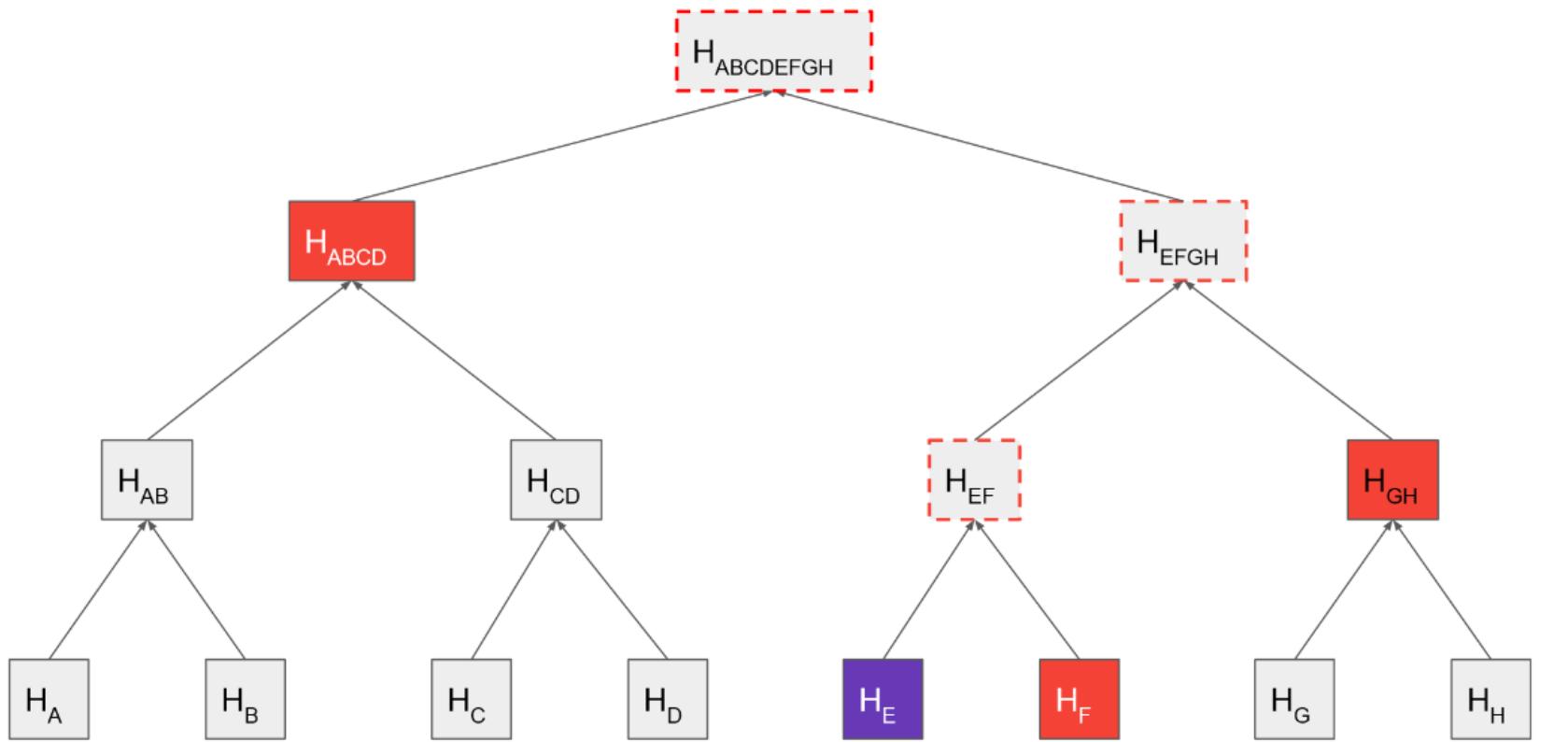
# Proof of inclusion

- Client creates Merkle Tree with root MRT from file data D.
- Client send file data D to server.
- Client deletes data D but stores MTR
- Client request chunk X from the server
- Server returns chunk X and a short proof of inclusion  $\pi$
- Client checks that chunk X is include in MTR using proof  $\pi$

# Proof of inclusion



# Proof of inclusion



# Proof of inclusion

---

- Prover sends chunks
  - Prover sends **siblings** along path connecting leaf to MTR
  - Verifier computes hashes along the path connecting leaf to MTR
  - Verifier checks that computer root is = MTR
  - The proof of inclusion is  $O(\log n)$
  - If adversary can present proof-of-inclusion for incorrect leaf then we can break the hash function
-

# Merkle Tree Protocol (Optional)



## MT-Construct(D)

```
// Constructs a Merkle Tree with given Data D  
//Return the Merkle tree root
```

**If**  $|D| = \text{chunk size}$  **then**

    MT-Construct(D) = H(D)

**Else**

    MT-Construct(D) = H(MT-Construct(D1) || MT-Construct(D2)), where D = D1 || D2

# Merkle Tree Protocol (Optional)



MT-Prove( $D, x$ )

- Given Data  $D$  and element  $x$  in  $D$ , construct proof of inclusion
- Return the proof of inclusion  $\pi$  to be used with MT-construct
- Proof contains:
  - Siblings on path connecting  $x$  to root
  - A bit for each sibling indication whether the path we are taking is left or right.

# Merkle Tree Protocol (Optional)



MT-Verify( $r, \pi, x$ )

- Given Merkle root  $r$ , element  $x$  and proof-of-inclusion  $\pi$
- Output True/False based on whether the verification was successful

Correctness

For all  $D, x :$

$\text{MT-Verify}(\text{MT-Construct}(D), \text{MT-prove}(D, x), x) = \text{True}$

# Merkle Tree Applications

---

- Bitcoin uses Merkle Tree to store the transactions
- Bit-Torrent uses Merkle tree to exchange file
- Etheriun Blockchain uses Merkle-Patricia tries for storage and transactions



# Digital Signatures

# What we want from Digital Signatures?

---



Only you can sign but any one can verify.

Signature is tied to a particular document

Can't be cut and paste to another document.

---

# API for digital signatures

(sk ; pk ) := generateKeys(keySize)

sk : secret signing key

pk : Public verification key

sig := sign(sk ; message)

isValid : = verify(pk ; message; sig)

# Requirements for Signatures

---

Valid Signatures Verify

$$\text{verify}(\text{pk} ; \text{message}; \text{sign}(\text{sk} ; \text{message})) == \text{true}$$

Can't forge signatures

Adversary who, knows  $\text{pk}$  , gets to see the signature of his own choice, can't produce a verifiable signature on another message.

# Practical Stuff ...

---

Algorithms to generate keys need to be randomized  
So, we need a good source of randomness

Limit of message size  
fix: use Hash(message) rather than message.

Fun Trick: Sign a hash pointer  
Signature covers the whole structure

BITCOIN uses ECDSA standard for Digital Signatures

---

# Useful trick: Public key == Identity

---

If you see sig such a `verify(pk; msg; sig) == true`

Think of it as

pk says “[msg]”

To speak for pk you must know sk

# Decentralized Identity Management



Anybody can make a new identity at anytime  
make as many as you want

No central point of coordination

These identities are called “addresses” in Bitcoin

# Privacy

---

Addresses not directly connected to real world identity

But observer can link together an address's activity over time



**BITS** Pilani  
Pilani Campus

# Blockchain Technology (BITS F452)

Dr. Ashutosh Bhatia, Dr. Kamlesh Tiwari  
Department of Computer Science and Information Systems



*A simple Cryptocurrency*

# Useful trick: Public key == Identity

---

If you see sig such a `verify(pk; msg; sig) == true`

Think of it as

pk says “[msg]”

To speak for pk you must know sk

# Decentralized Identity Management



Anybody can make a new identity at anytime  
make as many as you want

No central point of coordination

These identities are called “addresses” in Bitcoin

# Privacy

---

Addresses not directly connected to real world identity

But observer can link together an address's activity over time

# GoofyCoin

Goofy, can create new coins

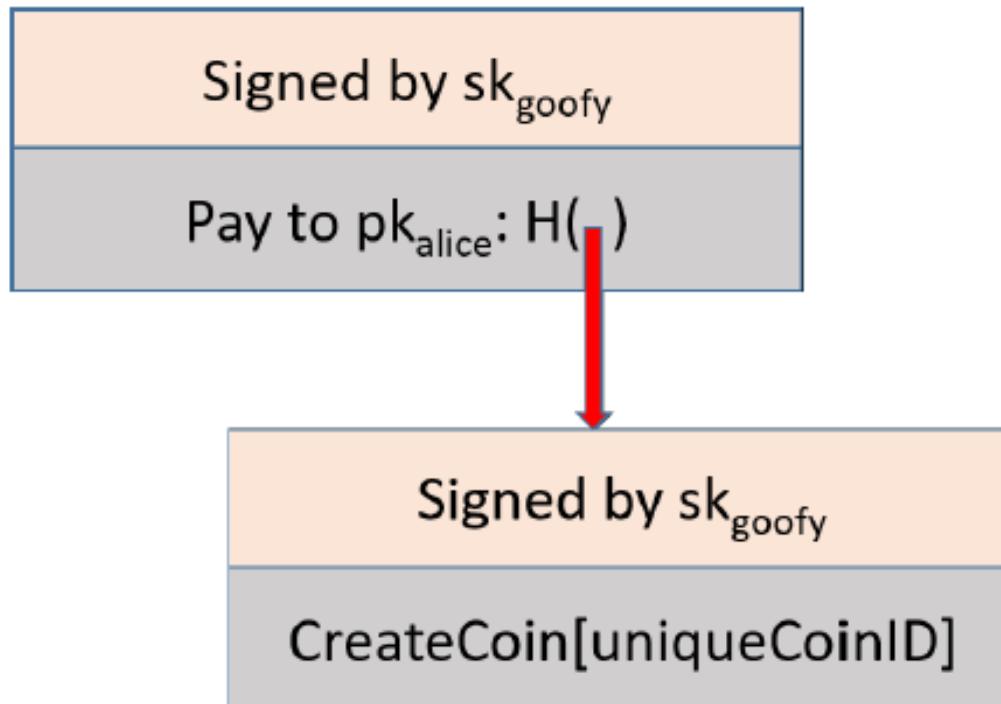
Signed by  $sk_{goofy}$

CreateCoin[uniqueCoinID]



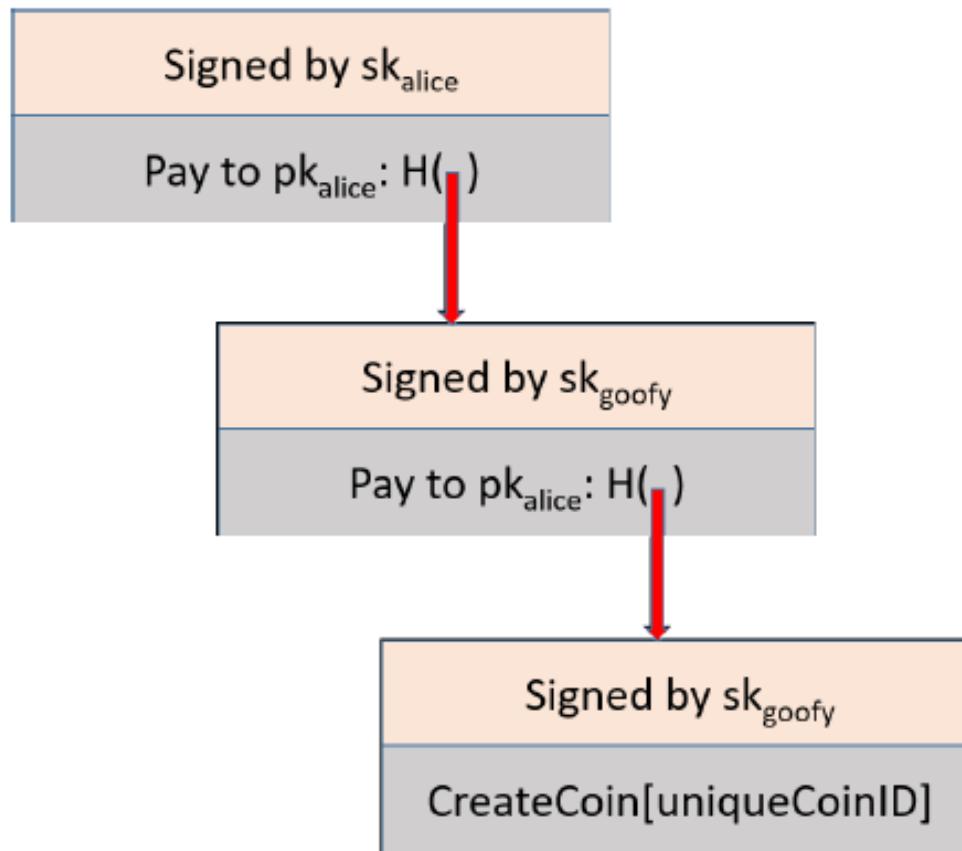
# GoofyCoin

A coin's owner can spend it

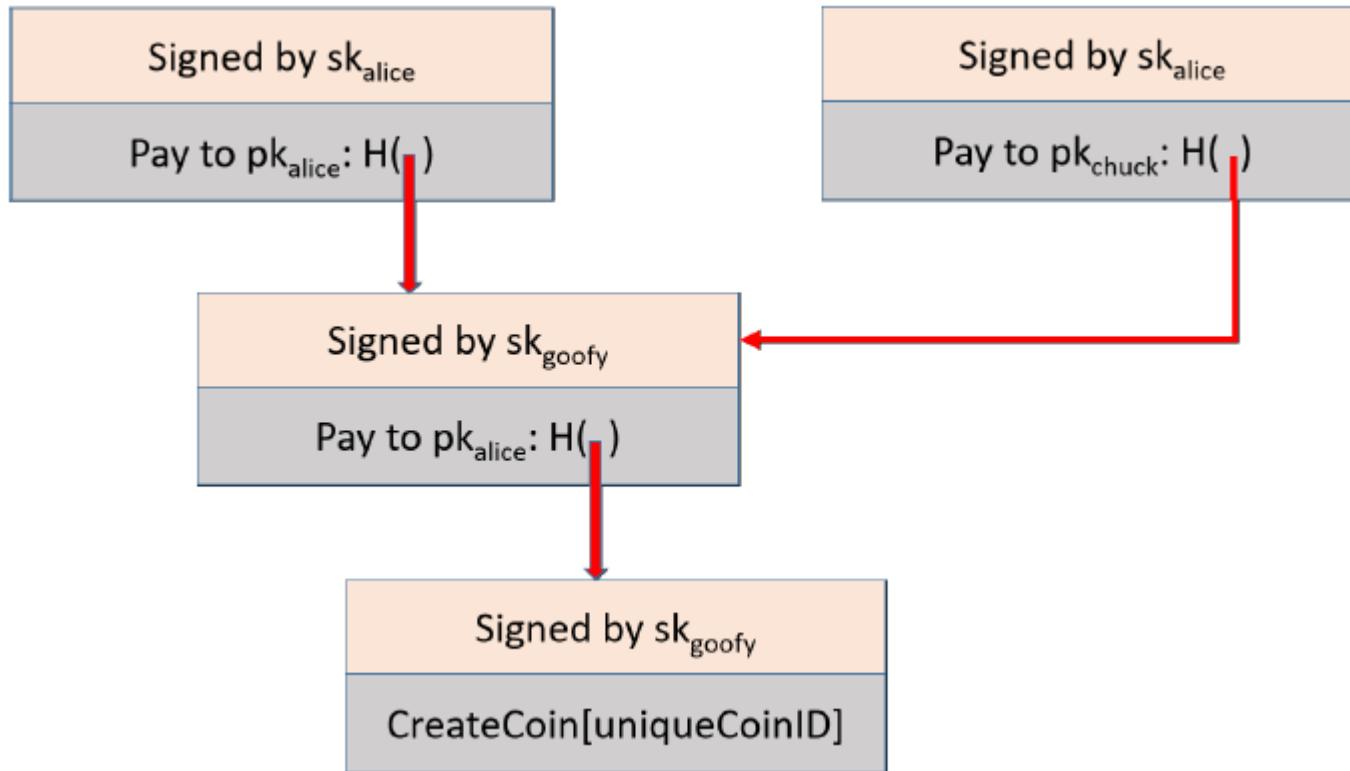


# GoofyCoin

A recipient can pass the coin again



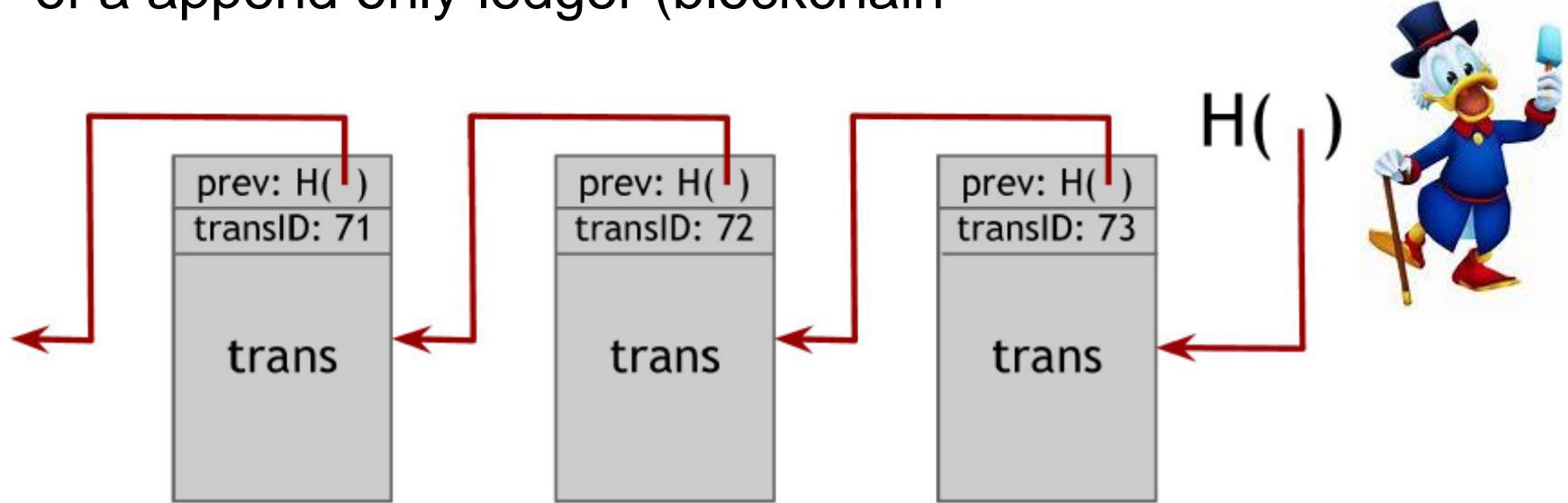
## Double Spending Problem



# ScroogeCoin

## ScroogeCoin: Solving Double Spending Problem

Scrooge publishes a history of all the transactions in form of a append only ledger (blockchain)



Optimization: put multiple transactions in the same block

# ScroogeCoin

## CreateCoin Transaction create a new coin

transID: 73	type:CreateCoins	
coins created		
<i>num</i>	<i>value</i>	<i>recipient</i>
0	3.2	0x...
1	1.4	0x...
2	7.1	0x...

coinID 73(0) ←

coinID 73(1) ←

coinID 73(2) ←

# ScroogeCoin

A Paycoin transaction consumes some coins and creates new coins of the same value

transID: 73	type:PayCoins			
consumed coinIDs: 68(1), 42(0), 72(3)				
coins created				
num	value	recipient		
0	3.2	0x...		
1	1.4	0x...		
2	7.1	0x...		
signatures				

Valid if

- ✓ Consumed coins are valid
- ✓ Not already consumed
- ✓ total value out = total value in
- ✓ signed by owners of all consumed coins

# ScroogeCoin

---

Problem with the scrooge coin

Coins can't be transferred, subdivided or combined

but you can get the same effect by using transactions to sub divide:

create a new transaction, consume your coin and pay out two new coins to yourself.



## Crucial Question

Can we de-scoogify the currency and operate without a trusted third party

We need to figure out:

How every one agree upon a single public block chain

How every one agree upon which transactions are valid

How to assign IDs to coins in a decentralized manner.



**BITS** Pilani  
Pilani Campus

# Blockchain Technology (BITS F452)

Dr. Ashutosh Bhatia, Dr. Kamlesh Tiwari  
Department of Computer Science and Information Systems



# *Decentralized Cryptocurrency*



## Crucial Question

Can we de-scoogify the currency and operate without a trusted third party

We need to figure out:

How every one agree upon a single public block chain

How every one agree upon which transactions are valid

How to assign IDs to coins in a decentralized manner.



# Decentralization is not all-or-nothing

Email: Decentralized protocol but dominated by centralized webmail services

# Aspects of Decentralization in BITCOIN

---

- Who maintains the ledger?
- Who has authority over which transactions are valid?
- Who creates new Bitcoins?
- Who determines how the rules of the system change?
- How do Bitcoins acquire exchange value?

**Beyond the protocol:** Exchange, wallet, software and service providers

---

# Aspects of Decentralization in BITCOIN

---

- **Peer to Peer Network**
  - Open to anyone, low barrier to entry
  - Currently there are several thousands of bitcoin nodes
- **Mining**
  - open to anyone but inevitable concentration of power often seem as undesirable.
- **Updates to Software**
  - Core developers trusted by the community, have great power

# BITCOIN's Key Challenge

Key technical challenge of decentralized ecash : **Distributed Consensus**

or: How to decentralize ScroogeCoin

# Why Consensus Protocol ?

---

Traditional Motivation

Reliability in Distributed Systems

Distributed Key-Value Store enables various applications  
DNS, Public-Key Dictionary, Stock Trades

---

# Defining Distributed Consensus

---

There is a fix number of nodes or processes and each of these has some input value

Protocol terminates and all correct nodes decide on the same value

This value must have been proposed by some correct node

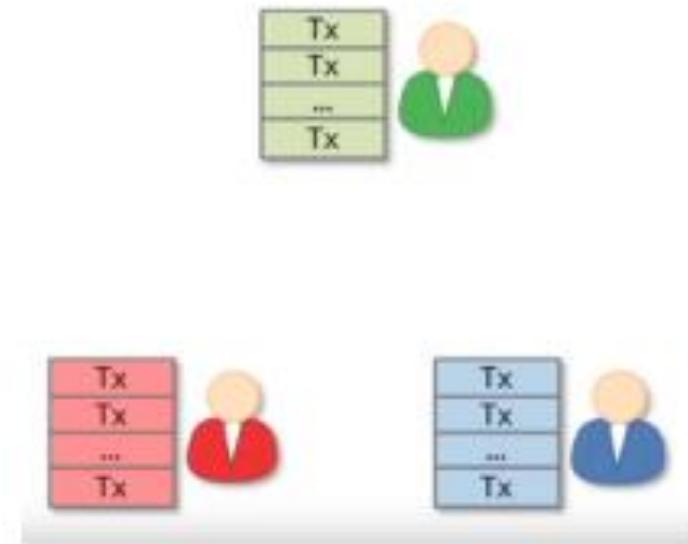
# BITCOIN is a P2P system

---

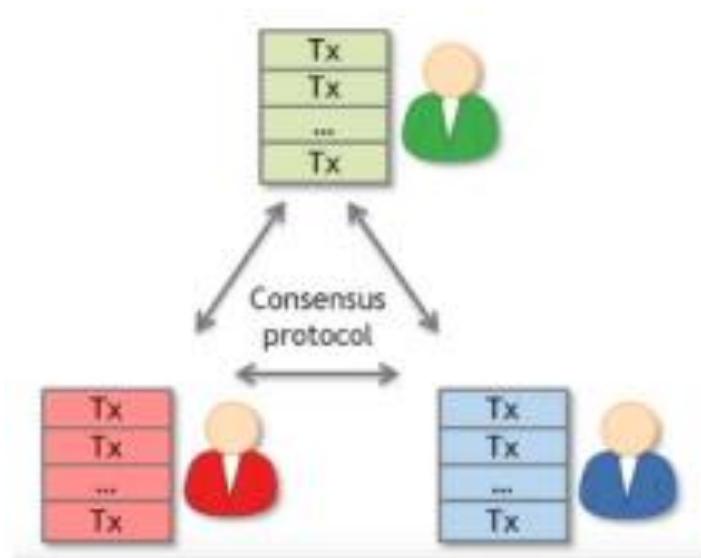
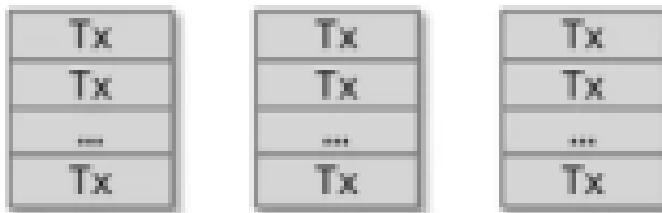
At any given time

- All nodes have sequence of blocks of transactions that they have consensus on
- Each node has a set of outstanding transactions that they have heard about

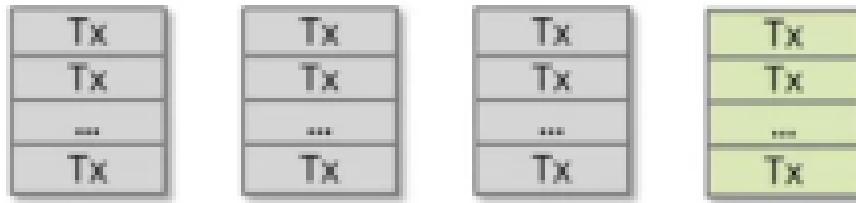
# How Consensus could work in BITCOIN



# How Consensus could work in BITCOIN



# How Consensus could work in BITCOIN



OK to select any valid block, even of proposed by only one node

# Why Consensus is hard

---

Nodes may crash

Nodes may be malicious

Network is imperfect

- Not all pair of nodes connected
- Faults in network
- Latency



No notion of Global Time

# Many impossibility results

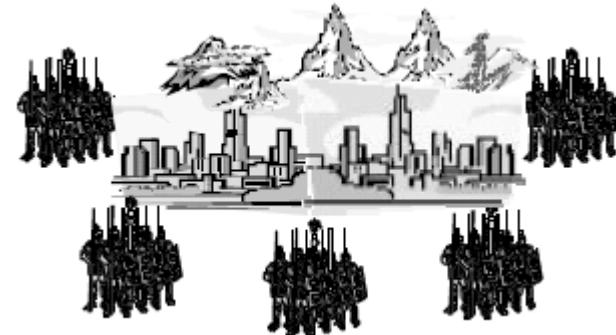
---

- Byzantine generals problem
- Fischer-Lynch-Paterson (FLP) result says that you can't do agreement in an Asynchronous Message Passing system if even one crash failure is allowed, unless you augment the basic model in some way, e.g. by adding randomization or failure detectors.

# Byzantine Generals Problem (Optional)



- Generals = Computer Components
- The abstract problem...
  - Each division of Byzantine army is directed by its own general.
  - There are  $n$  Generals, some of which are traitors.
  - All armies are camped outside enemy castle, observing enemy.
  - Communicate with each other by messengers.
  - Requirements:
    - G1: All loyal generals decide upon the same plan of action
    - G2: A small number of traitors cannot cause the loyal generals to adopt a bad plan
  - Note: We **do not** have to identify the traitors.



# Some well known protocols

---

Example: Paxos

Never produces inconsistent result but can get stuck.

# **BITCOIN consensus theory and practice**

---



**BITCOIN consensus works better in practice than in theory**

Theory is still catching up

BUT theory is important, can help predict unforeseen attacks.

---

# Some things BITCOIN does differently



- Introduces incentives
  - Possible only because it's a currency
- Embraces randomness
  - Does away with the notion of specific end point
  - Consensus happen over a long time scale

# BITCOIN consensus algorithm

---

Keep in mind that BITCOIN does this without having any long term identities which is different from classical distributed system.

Why don't BITCOIN node have identities

Identity is hard in P2P system – **Sybil Attack**

Pseudoanonymity is a goal of BITCOIN

# Key Idea: Implicit Consensus

---

In each round a random node is picked

This node proposes a next block in the chain

Other nodes implicitly accept/reject this block

- By either extending it
- Or ignoring it and extending the chain from the earlier block

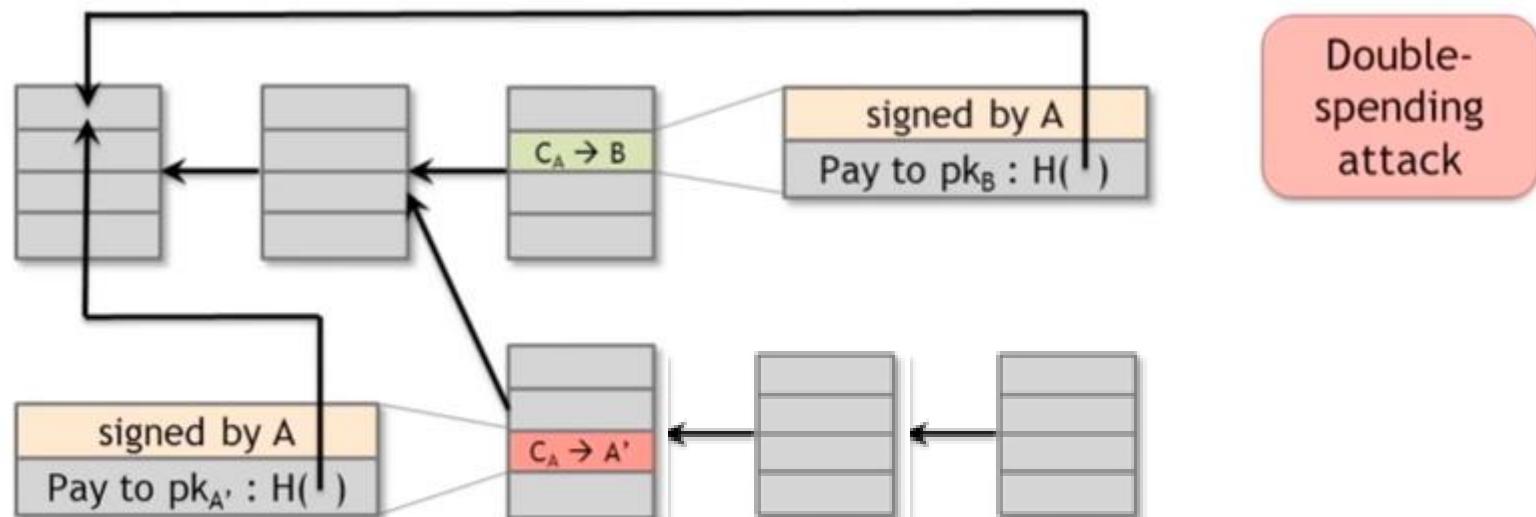
Every block contains the hash of the block it extends

# Consensus algorithm simplified

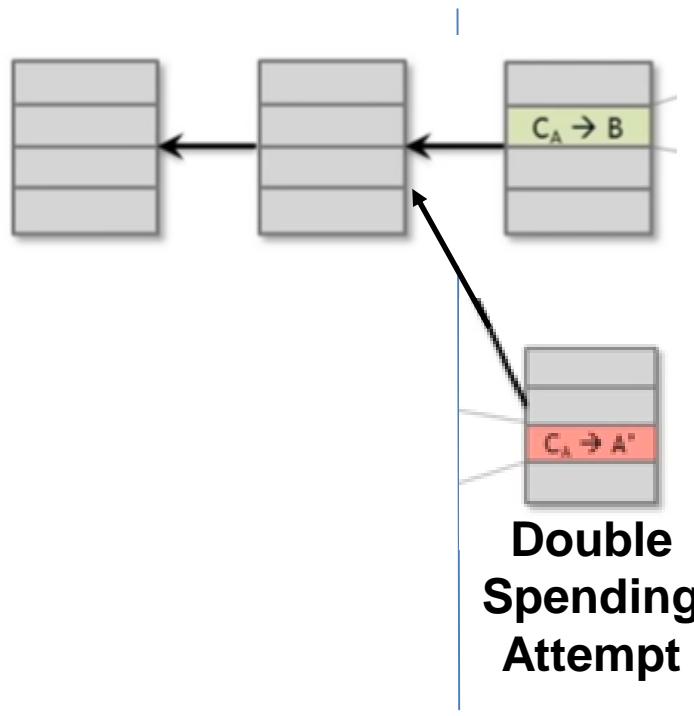


1. New transactions are broadcast to all nodes
2. Each node collects new transactions into a block
3. In each round a random node gets to broadcast its block
4. Other nodes accept the block only if all the transaction in the block are valid (unspent, valid signatures)
5. Nodes express their acceptance of the block by including its hash in the next block they create.

# What can a malicious node do



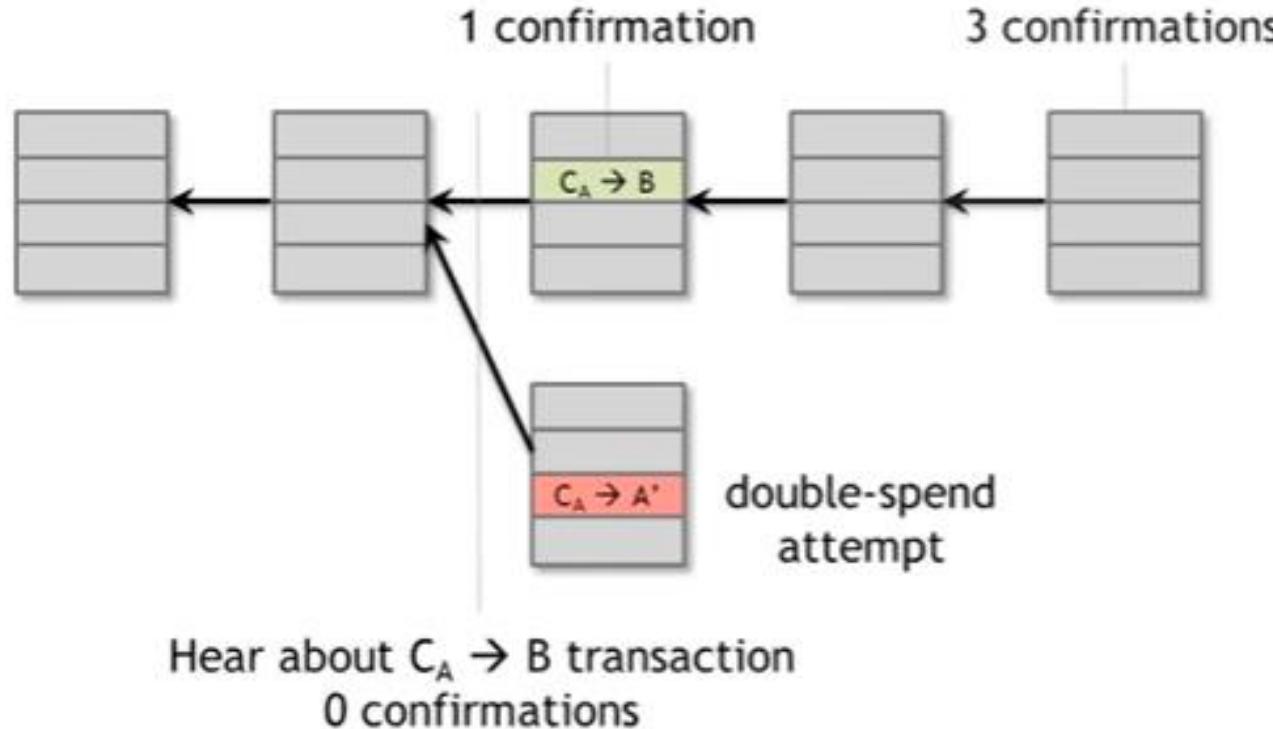
# From Bob the merchants point of View



Hear About  $C_A \rightarrow B$   
transaction over P2P  
network  
**(0 confirmation)**

Hear About  $C_A \rightarrow B$   
transaction in the  
blockchain first time  
**(1 confirmation)**

# From Bob the merchants point of View



- Double spending probability decreases exponentially with number of confirmation
- Most common heuristic is wait for 6 confirmations

# Recap

---

Protection against invalid transactions is cryptographic but enforced by consensus

Protection against double spending is purely by consensus

You are never 100% sure that a transaction is in consensus branch

Guarantee is probabilistic

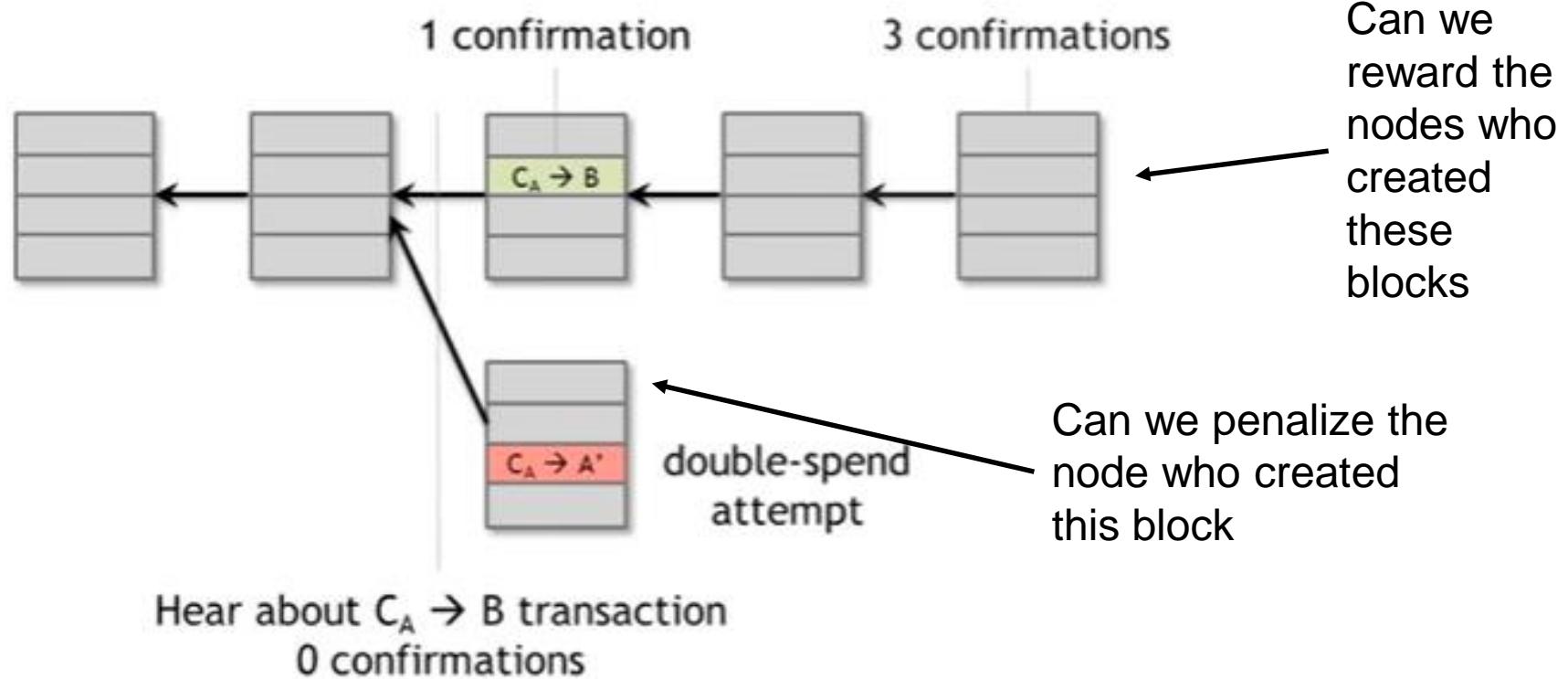
---



# Incentives and Proof of Work

# Assumption of honesty is problematic

Can we give nodes incentives for behaving honestly.



# Incentive 1 : Block Reward

---

Creator of block get to

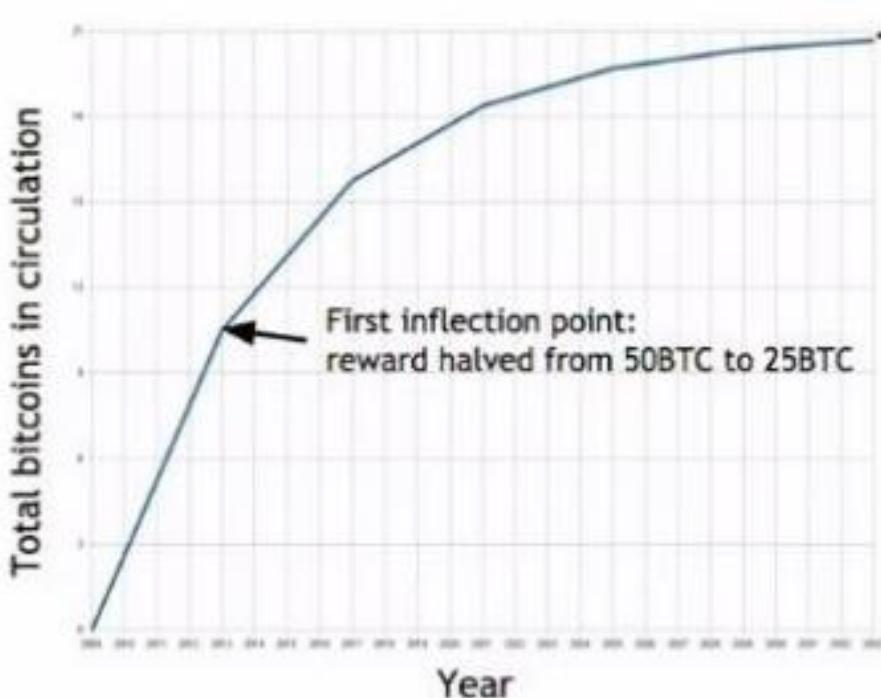
Special coin-creation transaction in the block

Choose receipt address of this transaction

Value is fixed currently : 6.25 BTC halves every 4 year

If the block end up on the long term consensus branch

# Finite Supply of BITCOINS



Total supply: 21 million

Block reward is how new bitcoins are created

Runs out in 2140. No new bitcoins unless rules change

# Incentive 2: Transaction Fee

---

Creator of a transaction can make its output value less than to its input value

Remainder is the transaction fee and it goes to the block creator

Purely voluntary like a tip

# Remaining Problems

---

How to pick a random node?

How to avoid a free-for-all due to rewards?

How to prevent the Sybil attack?



**BITS** Pilani  
Pilani Campus

# Blockchain Technology (BITS F452)

Dr. Ashutosh Bhatia, Dr. Kamlesh Tiwari  
Department of Computer Science and Information Systems



# *Proof of Work*

# Remaining Problems

---

How to pick a random node?

How to avoid a free-for-all due to rewards?

How to prevent the Sybil attack?

Are these problems related and have same solution :  
**Proof of Work**

# Proof of Work

To approximate selecting a random node  
select node in proportion to a resource that no one can monopolize (we hope)

- In proportion to computing power : Proof-of-Work
- In proportion to ownership: Proof-of-stake

**Idea:** allow nodes to compete with each other using their computing power that implies the nodes automatically being picked in that proportion

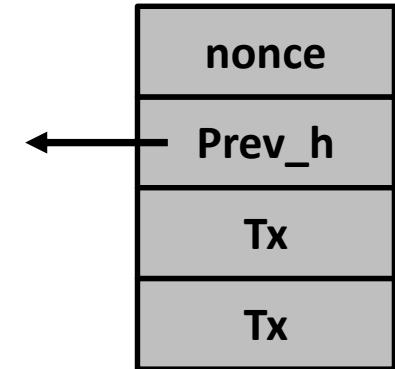
# Equivalent views of POW

---

1. Select nodes in proportion to computing power
2. Let nodes compete for right to create blocks
3. Make it moderately hard to create new identities  
protection against Sybill attack

# Hash Puzzles

To create block, find nonce (a random value) such that

$$H(\text{nonce} \parallel \text{prev\_hash} \parallel \text{tx} \parallel \text{tx} \parallel \dots \parallel \text{tx}) < \text{target}$$


Output space of hash



← →

Target Space

**If hash function is secure:  
Only way to succeed is to try enough nonces until  
you get lucky**

# POW property 1: difficult to compute

---

As of Feb 2022 the bitcoin difficulty is  **$26.69 \times 10^{12}$**  hashes

It requires approximately  **$2.7 \times 10^{15}$**  hashes to create one BITCOIN

Only some nodes bother to compete - minors

# POW property 2: parameterizable cost

---

Nodes automatically re-calculate the target every two weeks

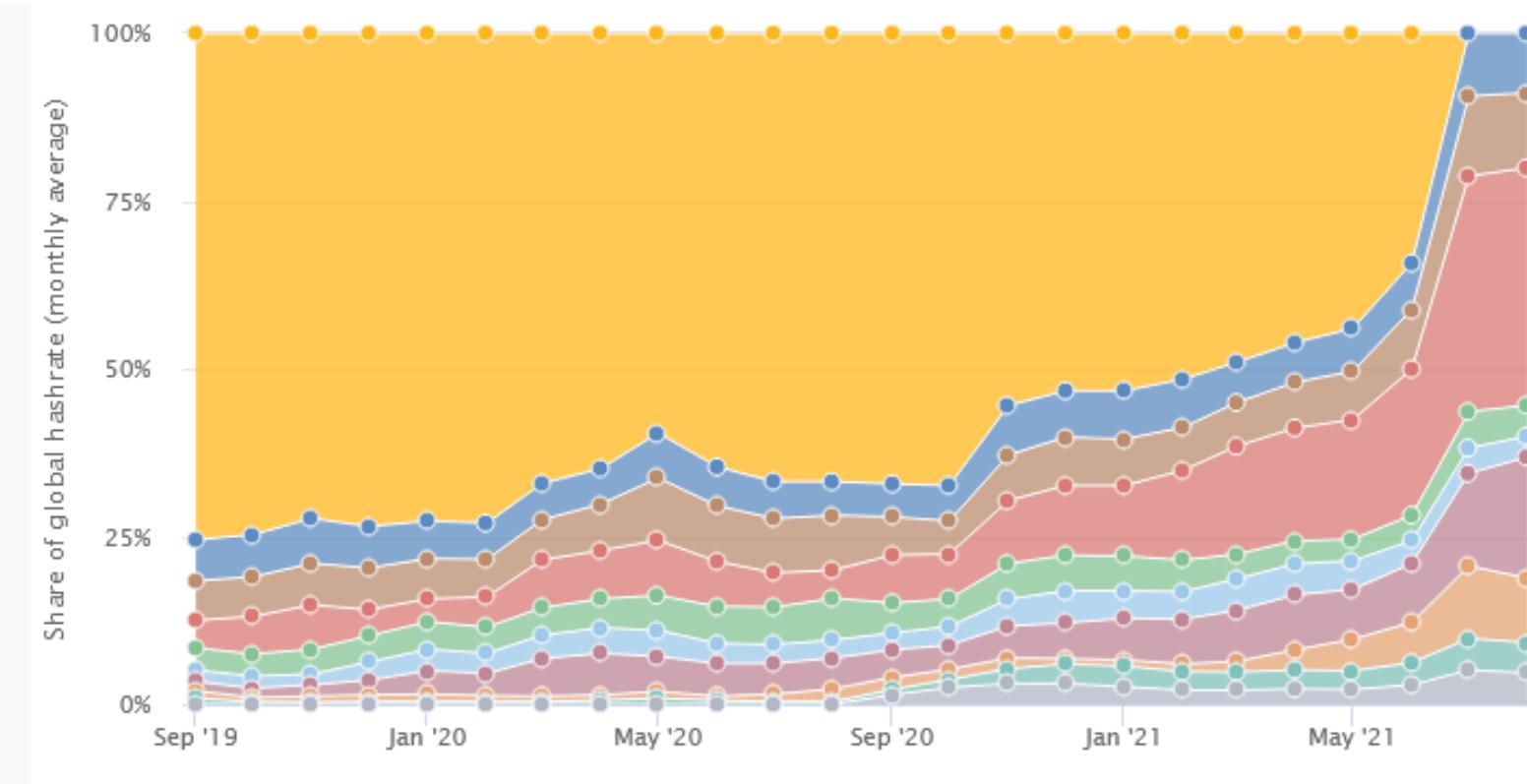
Goal average time between blocks = 10 minutes

Each 2016-block interval is known as a ***difficulty epoch***. At the beginning of every epoch the Bitcoin network recalculates the Current Target.

If you put a fixed amount of H/W for mining the rate at which you find the block depends upon the total computer power available with others

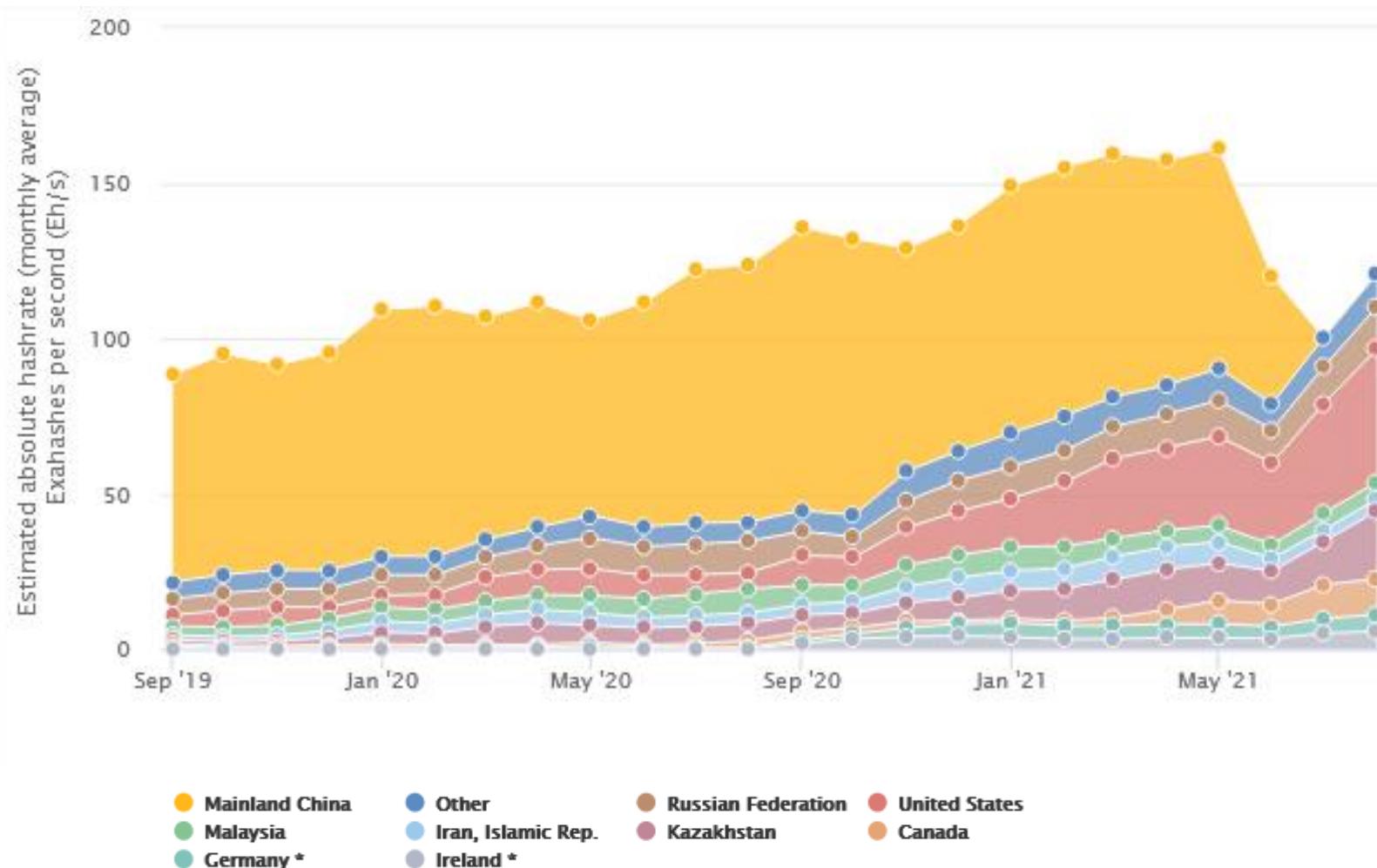
Prob (Alice wins the next block) = fraction of global hash power she controls

- Distribution of Bitcoin mining by country



- Mainland China
- Other
- Russian Federation
- United States
- Malaysia
- Iran, Islamic Rep.
- Kazakhstan
- Canada
- Germany \*
- Ireland \*

# BITCOIN Global Hashrate



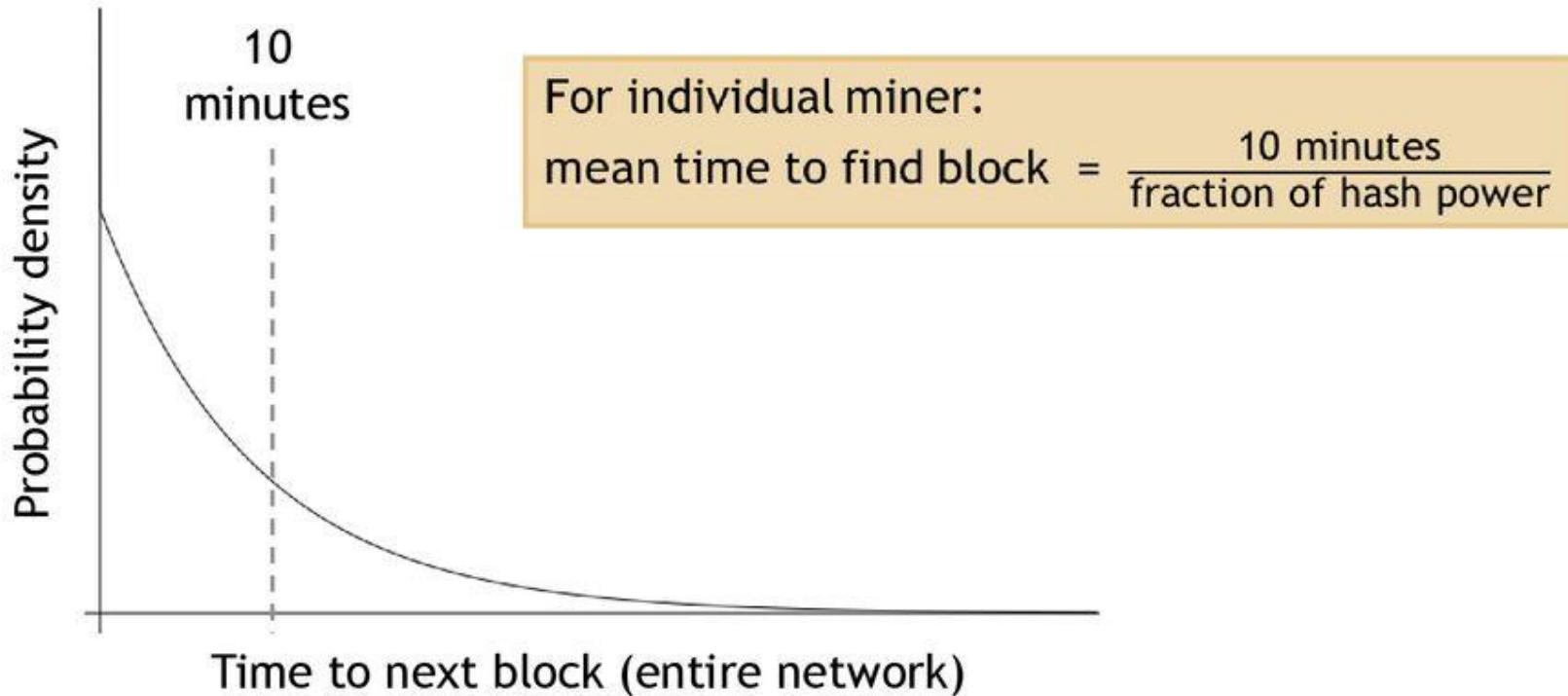
# Key Security Assumptions

---

Attacks infeasible if majority of minors  
weighted by hash power follow the protocol

This will ensure a more than 50% chance  
that the next node is proposed by a honest  
node

# Solving hash puzzles is probabilistic



## PoW property 3: trivial to verify

Nonce must be published as part of block

Other miners simply verify that

$$H(\text{nonce } \text{prev\_hash } \text{tx } \dots \text{ tx}) < \text{target}$$

Advantage?

No centralized verifier needed! Any node or miner can verify that the block was correctly mined

# Mining economics

If mining reward  
(block reward + Tx fees) > mining cost  
(hardware + electricity cost) → Profit

## Complications:

- Fixed (hardware) vs. variable (electricity) costs
- Reward depends on rate at which miners propose blocks (ratio of their hash rate to the global hash rate)
- Cost in dollars, but reward in BTC → profit depends on exchange rate

## Recap

Identities

Block chain & consensus

Transactions

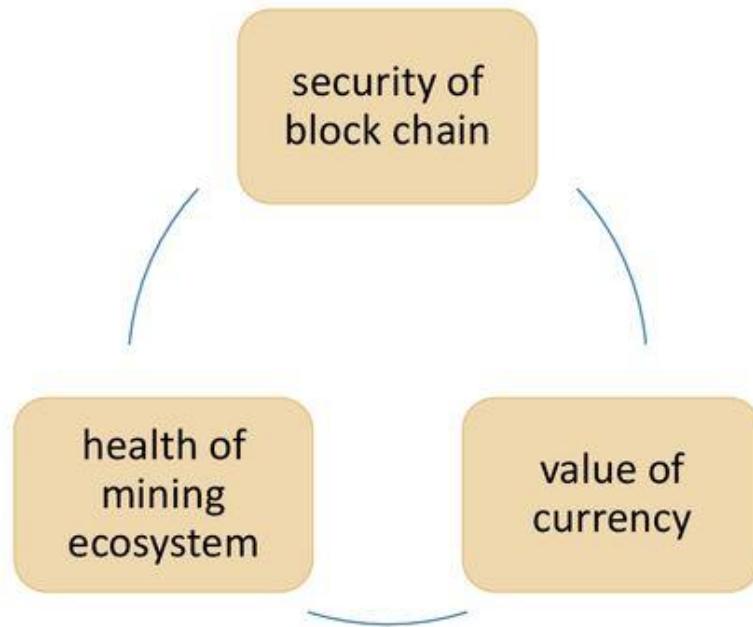
Hash puzzles & mining

P2P network

Bitcoin has three types of consensus

- Value
- State
- Rules

# Bitcoin is bootstrapped



# Blockchain Technology

## Introduction to Ethereum

Ashutosh Bhatia

BITS Pilani

[ashutosh.bhatia@pilani.bits-pilani.ac.in](mailto:ashutosh.bhatia@pilani.bits-pilani.ac.in)

# Overview

---

- What is Ethereum
- Compared to Bitcoin
- Components of a public Blockchain
- Ethereum: A general purpose blockchain
- Ethereum components
- Ethereum and Turing Completeness

# What is Ethereum

---

- Often described as “World wide computer operating under consensus”
- Computer Science Perspective
  - Ethereum is a deterministic, but practically unbounded state machine, consisting of a globally accessible singleton state and a virtual machine that applies changes to that state according to consensus rules.
- Practical Perspective
  - Open source, globally decentralized computing infrastructure that executes programs called smart contracts and uses blockchain to synchronize and store the system’s state change along with a cryptocurrency called ether, to meter and constrain resource costs
- Developer Perspective
  - A platform that enables developers to built decentralized applications with build-in business logic, providing availability, auditability, transparency and neutrality, and reducing certain counterparty risks.

# Ethereum Vs BITCOIN (Commonalities)

---

- **Open Blockchains:** Trustless, Immutable, Uncensorable, No central point of failure
- **Decentralized Identify:** Pseudo anonymous identify using public key
- **Consensus algorithm:** POW based Byzantine Fault tolerant consensus algorithm (mining power proportional to the computer power)
- **Cryptographic primitives:** use of cryptographic primitives such as digital signatures and hashes, and a digital currency (ether).
- **P2P Network:** peer-to-peer network connecting participants

# Ethereum Vs BITCOIN (Differences)

---

<b>BITCOIN</b>	<b>Ethereum</b>
<ol style="list-style-type: none"><li>1. A blockchain for cryptocurrency</li><li>2. Asset: Bitcoins as currency<ul style="list-style-type: none"><li>• Primary purpose of the blockchain</li></ul></li><li>3. Simple and robust</li><li>4. primitive scriptive language, not Turing complete</li><li>5. UTXO-based</li></ol>	<ol style="list-style-type: none"><li>1. General purpose programmable blockcahin</li><li>2. Asset: Ether as utility currency<ul style="list-style-type: none"><li>• Fund computations</li></ul></li><li>3. Complex and feature rich</li><li>4. Turing complete scripting language</li><li>5. Account based</li></ol>

# Birth of Ethereum

- Conceived at a time when people recognized the power of Bitcoin model
- Trying to move beyond cryptocurrency:
  - a similar model with more generalized applications
- Developers Confusion:
  - Either build on top of BITCOIN by trying to find workarounds and live with the constraints imposed by the transaction type, data types and size of the data storage. Design anything else needed as off-chain, which could completely negate the very reason of using the blockchains
- In December 2013, a young programmer and BITCOIN enthusiast, **Vitalik Buterin** started sharing a white paper outline the idea behind **Ethereum**



<https://ethereum.github.io/yellowpaper/paper.pdf>

<https://ethereum.org/en/whitepaper/>

# Birth of Ethereum

- A few dozen people saw the early draft and offered feedback helping Vitalik to evolve the proposal.
- Gavin Wood, a computer programmer was one of the first people to reach out Vitalik and offered his C++ programming skills in the creation of Ethereum.
- Vitalik on Galvin contribution to Ethereum

This was the time when the Ethereum protocol was entirely my own creation. From here on, however, new participants started to join the fold. By far the most prominent on the protocol side was Gavin Wood, who reached out to me in an about.me message in December 2013:

---

**Gav Wood sent you a message on about.me**

1 message

i@gavwood.com <i@gavwood.com>  
Reply-To: i@gavwood.com  
To: vbuterin@gmail.com

Thu, Dec 19, 2013 at 11:53 AM



**Gavin Wood**

Computer scientist

Gavin Wood is a British computer programmer, co-founder of Ethereum and creator of Polkadot. He invented Solidity and wrote the Yellow Paper specifying the Ethereum Virtual Machine. Wood served as the Ethereum Foundation's first chief technology officer. [Wikipedia](#)

# Ethereum's Four Stages of Development

---

- **Block #0 : Frontier**—The initial stage of Ethereum, lasting from July 30, 2015, to March 2016.
  - **Block #200,000: Ice Age**—A hard fork to introduce an exponential difficulty increase, to motivate a transition to PoS when ready.
- **Block #1,150,000 Homestead**—The second stage of Ethereum, launched in March 2016.
  - **Block #1,192,000 DAO**—A hard fork that reimbursed victims of the hacked DAO contract and caused Ethereum and Ethereum Classic to split into two competing systems.
  - **Block #2,463,000 Tangerine Whistle**—A hard fork to change the gas calculation for certain I/O heavy operations
  - **Block #2,675,000: Spurious Dragon**— A hard fork to address more DoS attack vectors, and another state clearing. Also, a replay attack protection mechanism.
- **Block #4,370,000 Metropolis Byzantium**—Metropolis is the third stage of Ethereum
- **Serenity** Ethereum 2.0, also known as **Serenity** or ETH 2.0, is an upgrade to Ethereum on a number of levels. Its primary objective is to increase Ethereum's capacity for transactions, reduce fees and make the network more sustainable.

# Ethereum: A general-purpose Blockchain

---

- Unlike BITCOIN which tracks the state transitions for “Currency Ownership”, the Ethereum tracks the state transition of **general purpose data expressible as key-value pair**
- Similar to RAM model of a computer it stored both code and data and uses blockchain to track the changes in this in stored data.
- Like a general purpose computer, etherum can load the code in the state machine, run that code and storing the resulting state change in the blockchain
- Two critical differences with general purpose computers
  - Governed by rules of consensus.
  - State is distributed globally.

key	value
firstName	Bugs
lastName	Bunny
location	Earth

# Ethereum is Turing Complete

---

- In 1936 Alan Turing created a mathematical model for the computer consisting of a state machine that manipulates symbols by reading and writing on a sequential memory (Tape)
- Using this model he provided a proof to the question “Whether all problems are solvable” (universal computability)
- He proved that there are classes of problems that are uncomputable, specially the **halting problem**
- A system is said to be Turing complete if it can be used to simulate a Turing machine.

Ethereum groundbreaking contribution is to combine a general purpose computing architecture of a stored program computer with a decentralized blockchain thereby creating a distributed single state world computer.

# Ethereum's Components

---

- P2P Network: Etherreum runs on Ethereum main network, which is addressable on TCP port 30303 and runs a protocol called DEVp2p

**devP2P**

Cross platform peer-to-peer client library, for desktops and mobiles!

The C++ standalone library for establishing direct and secured P2P VPN connections. It can perform file transfers, port forwardings, full network redirection and more...

Once connected, use it for whatever purpose you need - share you screens using favorite application, open remote files, transfer data, send messages....



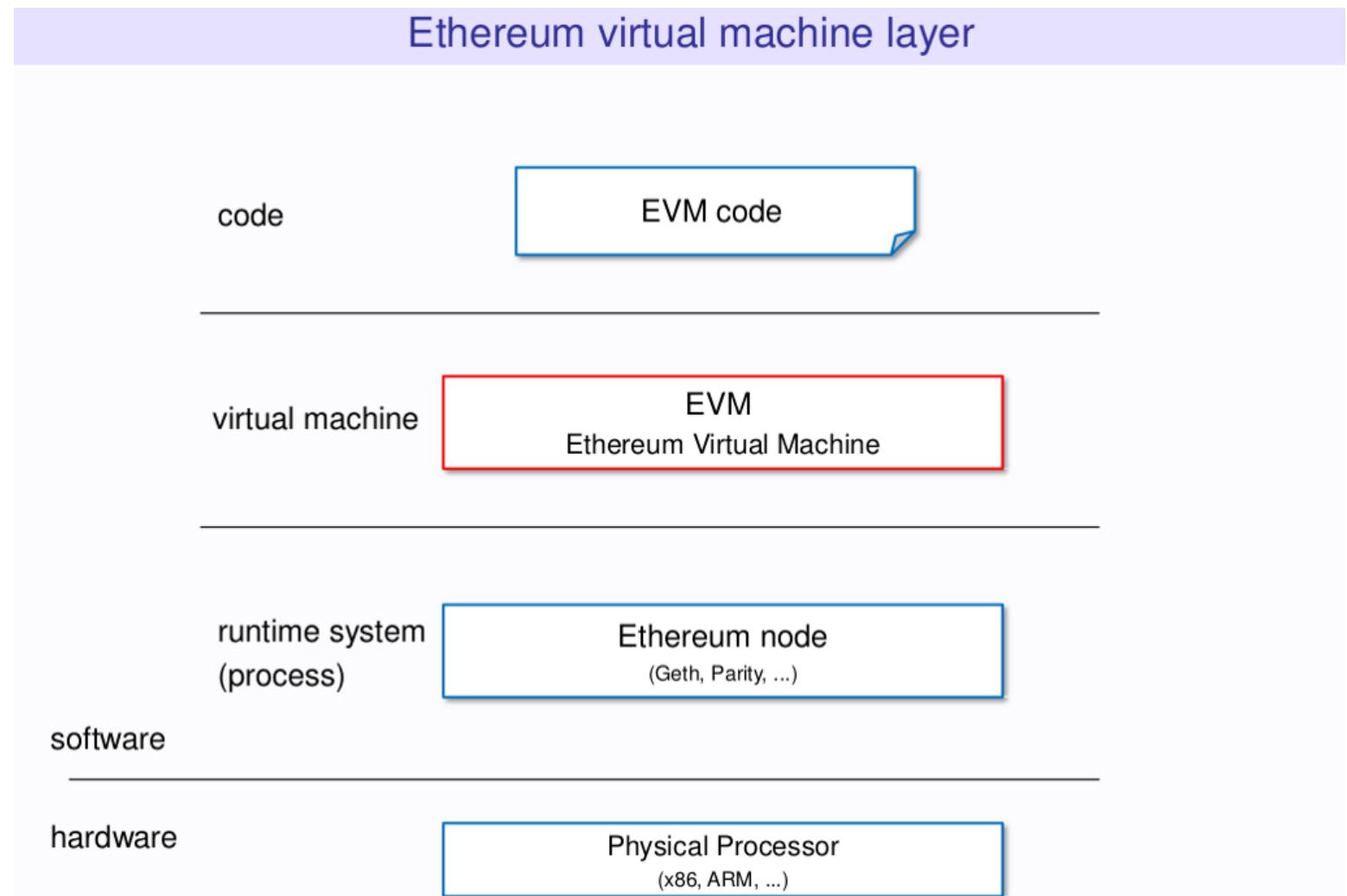
# DevP2P General Features

---

- Establishes secure P2P VPN over internet
  - Connection is established through UDP or TCP protocol with remote devP2P, virtually anywhere
- Provides LAN over internet
  - When network redirect is used, local network packets for outgoing peers are captured, transferred through devP2P to remote peer, and there they are provided to the system just as they have arrived from network cable.
- Share screen, data, files
  - You can fire up tools to view remote desktop through firewalls, transfer files and data.
- Forwards ports.. Sends files and messages.. Redirects network..
  - After connection is established, there are 1024 channels to use.  
Use them to SendMessage, SendFile, StartForwarding.

# EVM

- state transitions are processed by the EVM a stack based virtual machine that executes byte code.
- EVM programs called “smart contracts” are written in high level programming language and compiled to byte code for execution in EVM.



# Stack based computer

- A stack based computer do not use address field in instruction.

To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

- Example
  - Expression:  $X = (A+B)*(C+D)$
  - Postfixed :  $X = AB+CD+*$

PUSH A      TOP = A

PUSH B      TOP = B

ADD      TOP = A+B

PUSH C      TOP = C

PUSH D      TOP = D

ADD      TOP = C+D

MUL      TOP = (C+D)\*(A+B)

POP X      M[X] = TOP

# Why Stack based Machine

---

- Traditionally, virtual machine implementers have favored stack-based architectures over register-based due to 'simplicity of VM implementation'
- Ease of writing a compiler back-end
- executables for stack architecture are invariably smaller than executables for register architectures.

# Why EVM and not a JVM

---

- **complex and voluminous** : a single java method can have a size up to 64KB so such VM or language isn't space saving.
- **useless features and security concerns**: Network access, I/O stream, File W/R etc => big security issues.
  - think of it, you can write a code which ping (of death) another machine or access protected files or even steal the miner's private keys.
  - file "write/read" feature could break the whole system's security. so we'll need to get rid of all these features, which is a hard task to achieve for a licensed VM.
  - We need to remember that a Blockchain VM should be **isolated** without the capacity to communicate with the external environment.
- Imagine you have a Java bytecode with a rand(), what would be the result and how to reach the consensus then?.
- **Weak DDoS resistance** : how to set a gas-like system in a complex VM like Java VM? .
- **JAVA VM is a licensed Sun product**, so you can't customize it to integrate it to the Ethereum's environment
  - (for example how would you calculate gas cost to avoid Dos attacks?)? to overcome this problem you need to write your own Java VM which is a complex task read

# Ethereum's Components: Data Structures

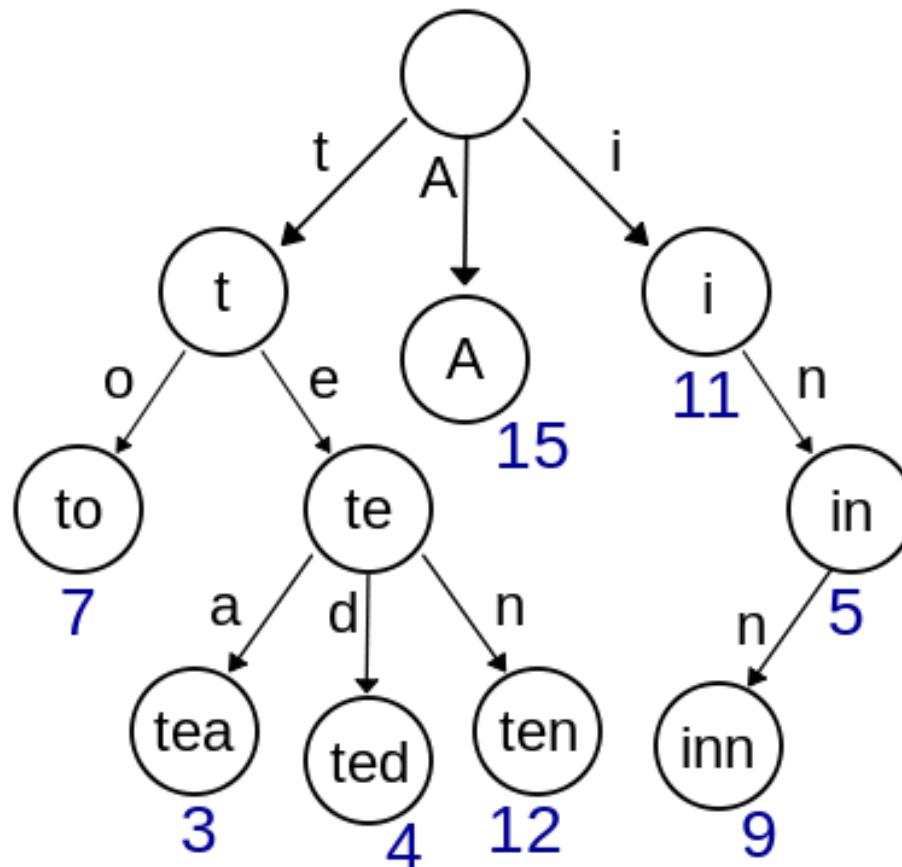
---

- Ethereum database is stored locally on each node as a database (usually Google's LevelDB), which contains transactions and system state in a serialized based data structure called **Merkle Patricia Trie (MPT)**.
- Basically, MPT is a combination of Patricia trie and Merkle tree, with few additional optimizations that fit the characteristics of Ethereum.
- Patricia trie is a data structure which is also called Prefix tree, radix tree or trie.
- Trie uses a key as a path so the nodes that share the same prefix can also share the same path.
- This structure is fastest at finding common prefixes, simple to implement, and requires small memory.
- Thereby, it is commonly used for implementing routing tables, systems that are used in low specification machines like the router.

# Patricia Trie: Example

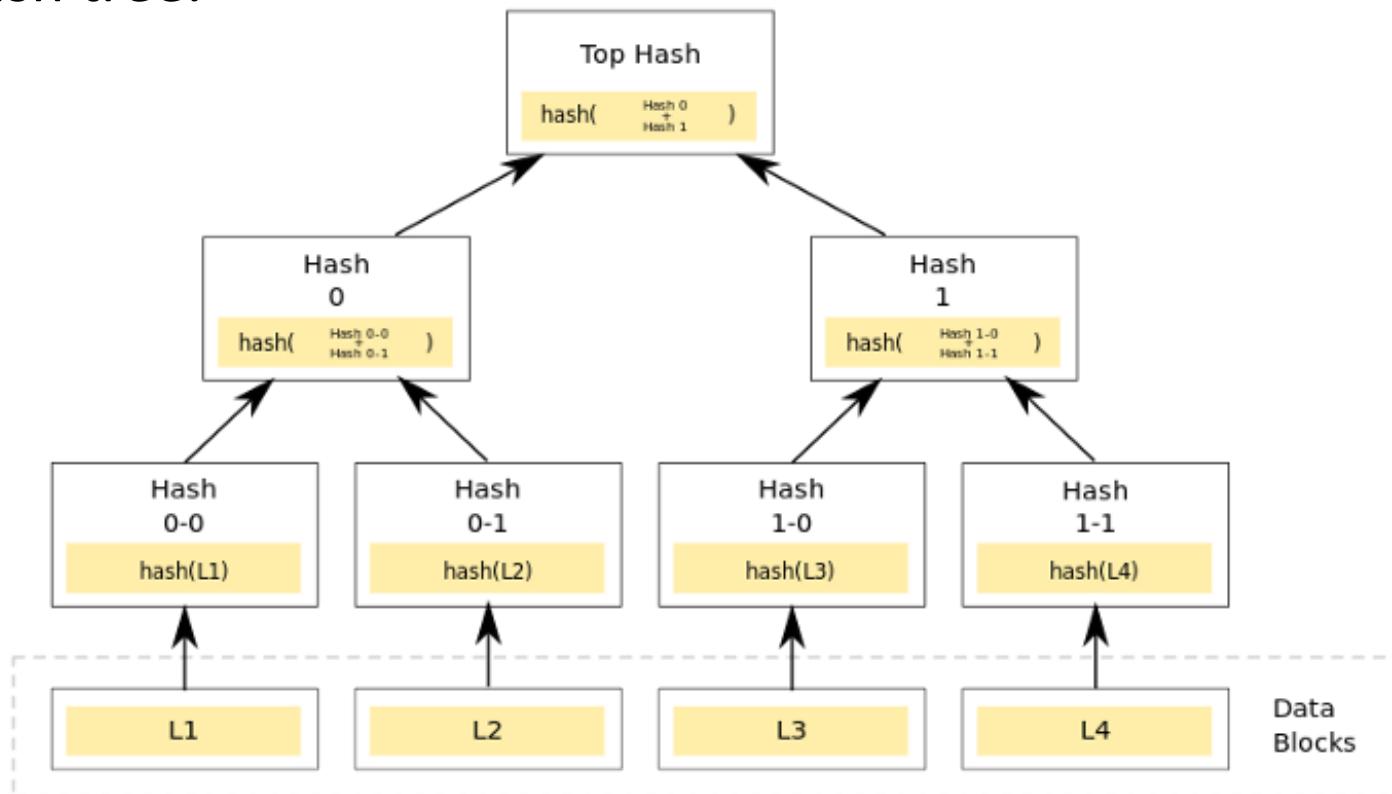
---

- A trie for keys "A", "to", "tea", "ted", "ten", "i", "in", and "inn". Each complete English word has an arbitrary integer value associated with it.

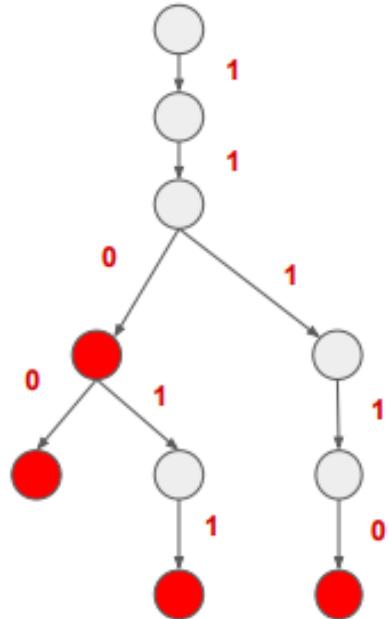


# Merkle Tree

Merkle tree is a tree of hashes. Leaf nodes store data. Parent nodes contain their children's hash as well as the hashed value of the sum of their children's hashes. Since all the nodes except for leaf nodes contain a hash, the Merkle tree is also known as a hash tree.

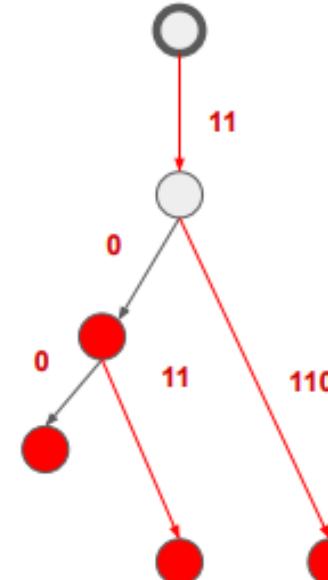
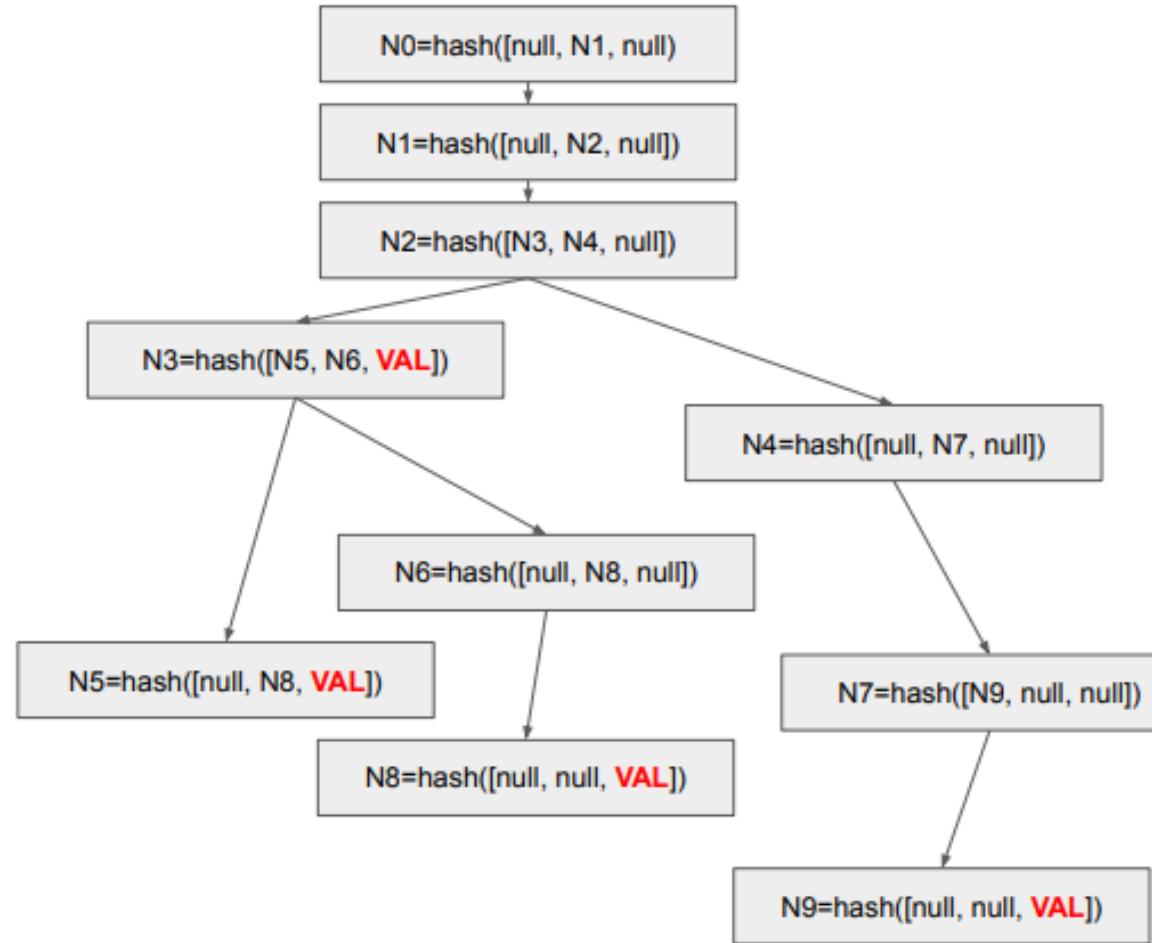


# Prefix, Merkle and Patricia Tree



Keys:

110  
1100  
11011  
11110

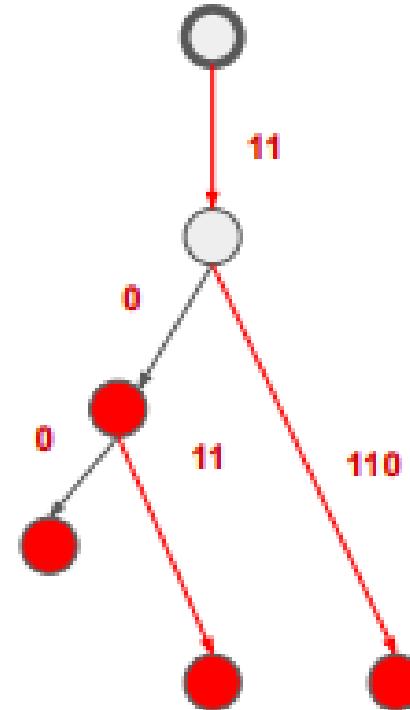


# Merkle Patricia Trie

---

The Merkle Patricia Trie defines three types of nodes

- Branch – a 17-item node [ $i_0, i_1, \dots, i_{15}, value$ ]
- Extension – A 2-item node [path, value]
- Leaf – A 2-item node [path, value]



Keys:

110

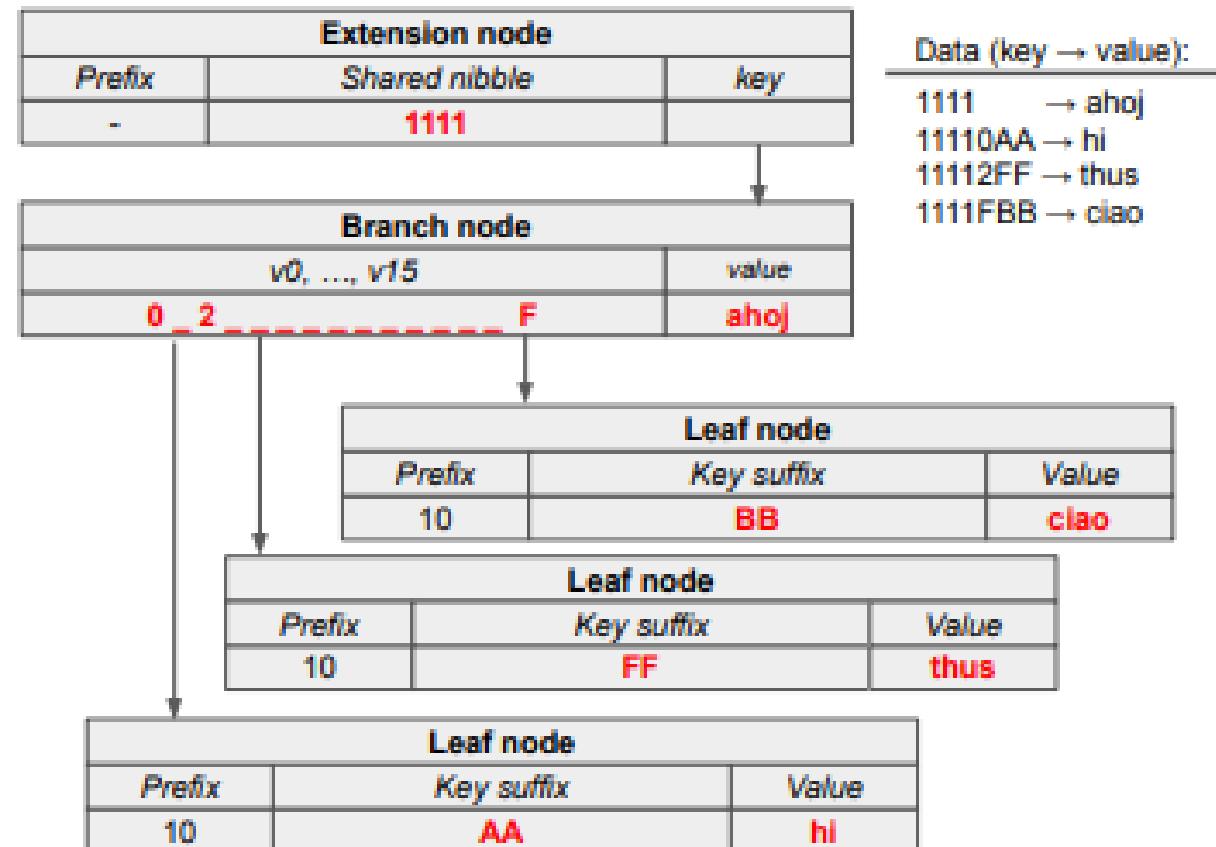
1100

11011

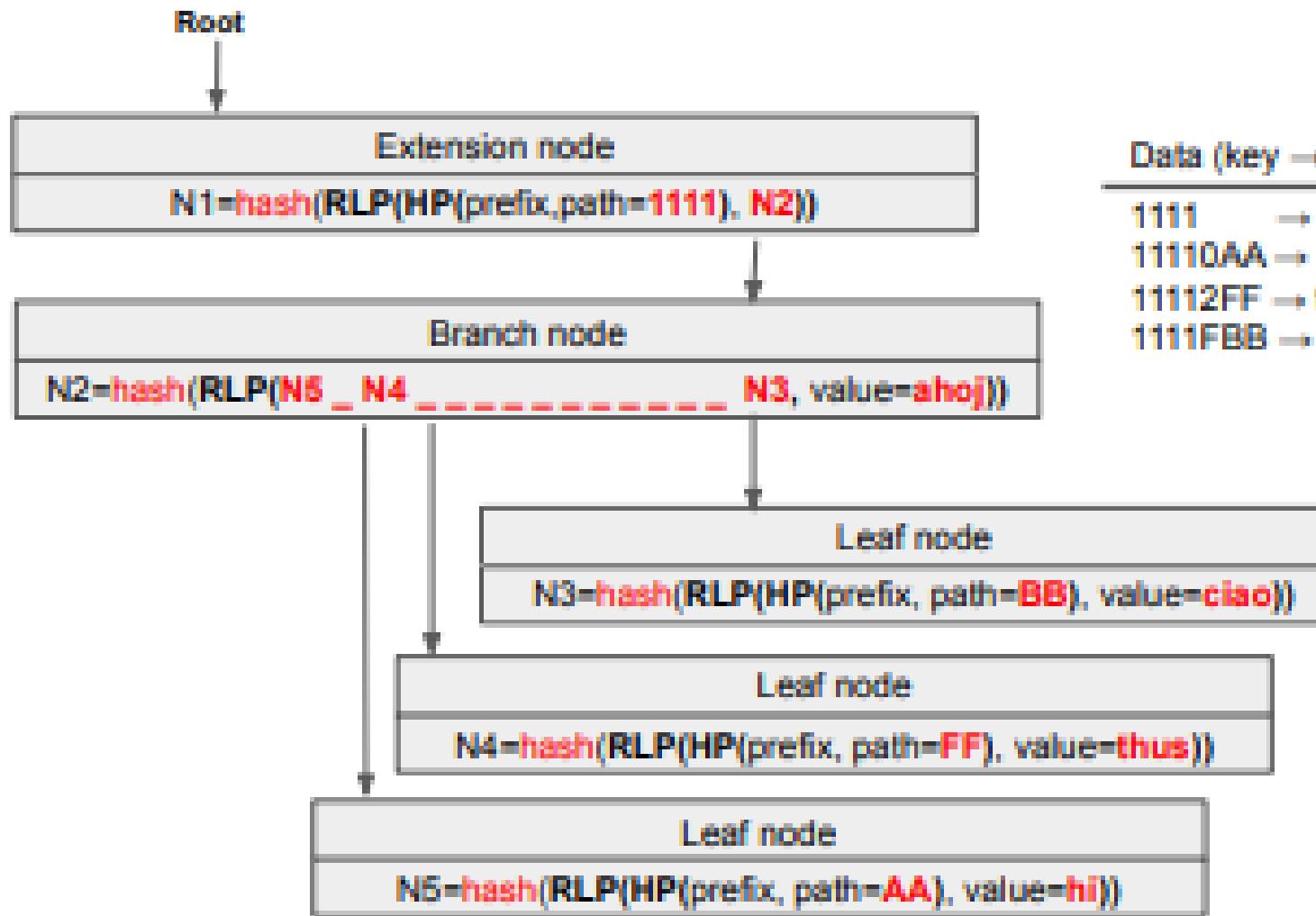
1110

# Merkle Patricia Trie

1. All the keys have the same prefix 1111 and for this reason the extension node is created as a root
2. Then the keys start to differ, which is captured by creating the branch node. This node branches for characters at the same position of all keys. In particular, branches are created for characters '0', '2', and 'F'.
3. The key having solely '1111' terminates here, which means that the value for this key is stored in this branch node.
4. Rest of the keys form leaf nodes as there is no more branching. The leaf nodes store suffixes of each key and store values for the keys as well.



# Ethereum encoded merkle petricia Tree



Data (key → value):

1111	→ ahoj
11110AA	→ hi
11112FF	→ thus
1111FBB	→ ciao

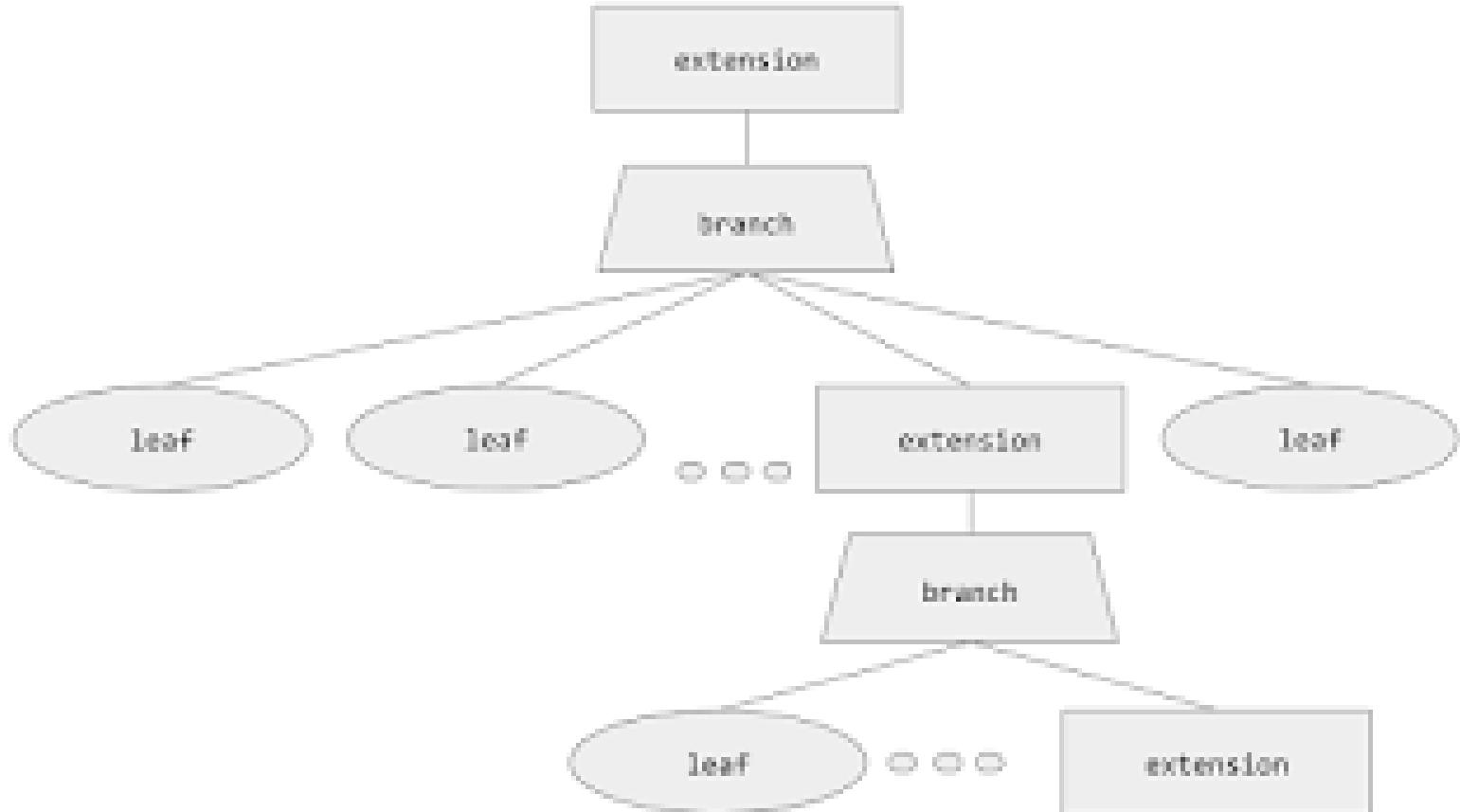
Recursive Length Prefix  
HP: Hex Prefix Encoding

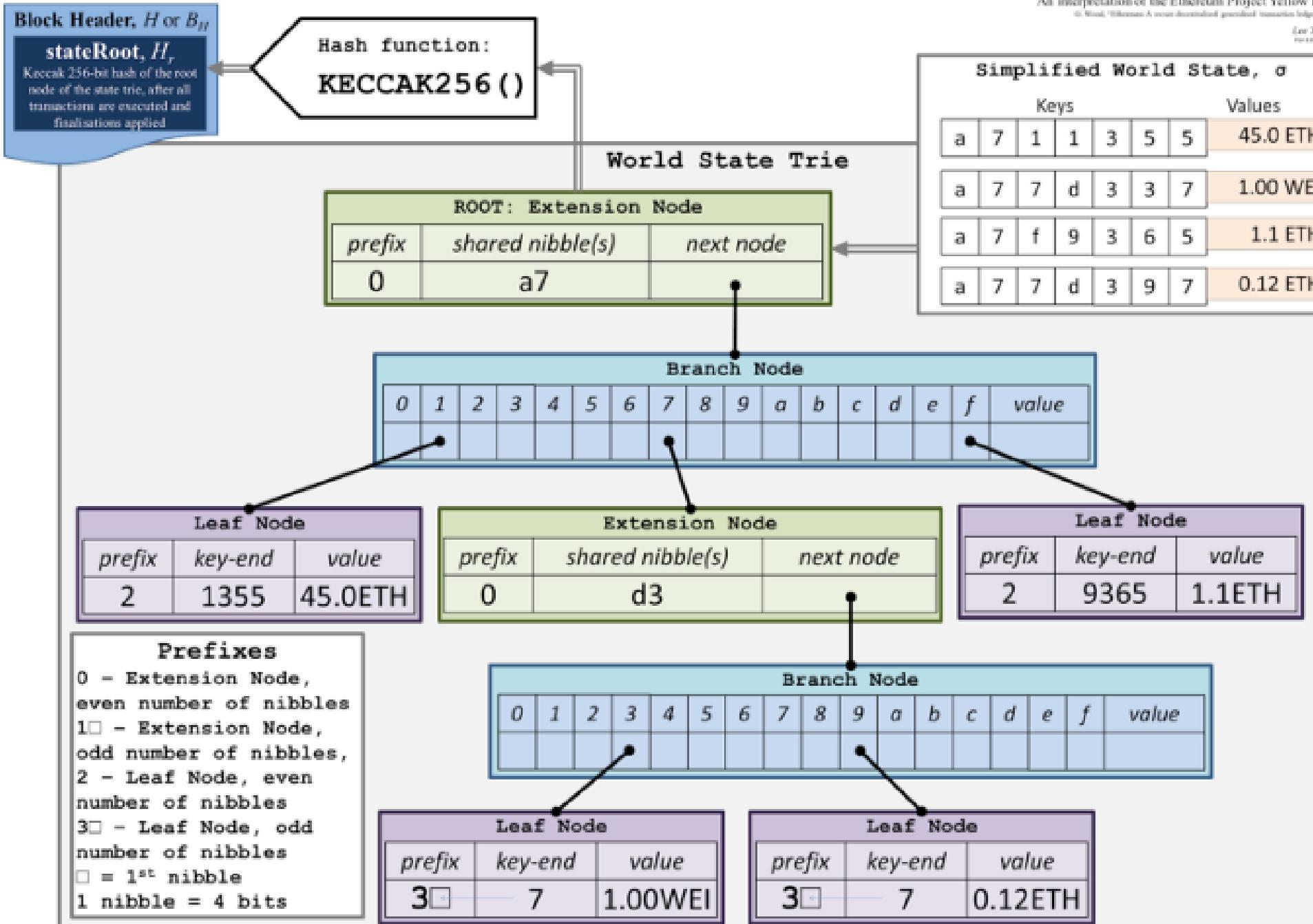
Prefix	Payload	Node Type	Path Length
[0, 0]	$[x_1x_2][x_3x_4]$	Extension	Even
[1, $x_1$ ]	$[x_2x_3][x_4x_5]$	Extension	Odd
[2, 0]	$[x_1x_2][x_3x_4]$	Leaf	Even
[3, $x_1$ ]	$[x_2x_3][x_4, x_5]$	Leaf	Odd

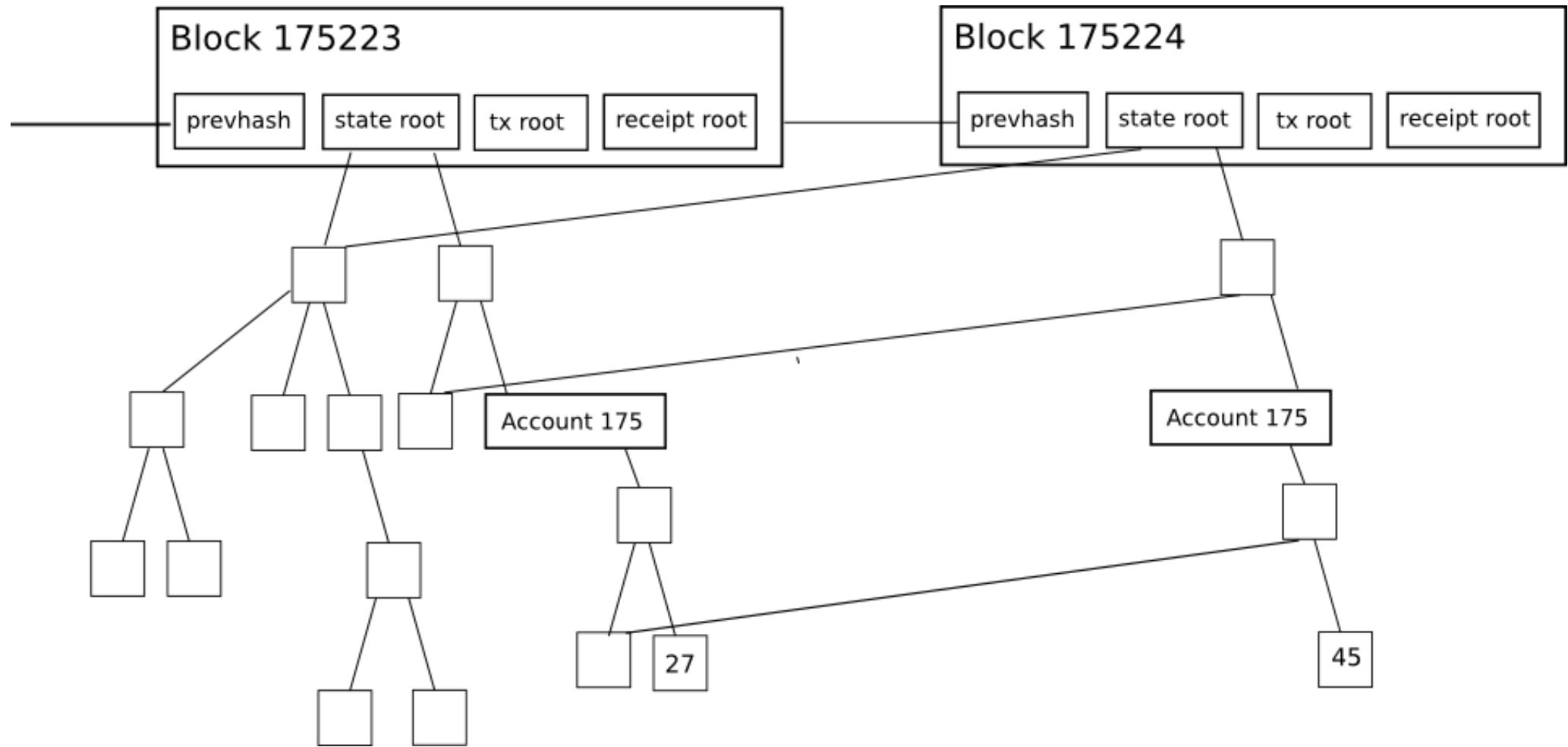
# Merkle Patricia Trie

---

- In the MPT, there is one more type of nodes apart from the branch nodes and the leaf nodes. They are extension nodes.
- An extension node is an optimized node of the branch node.







# State Trie Architecture

---

The Ethereum State Trie has four types:

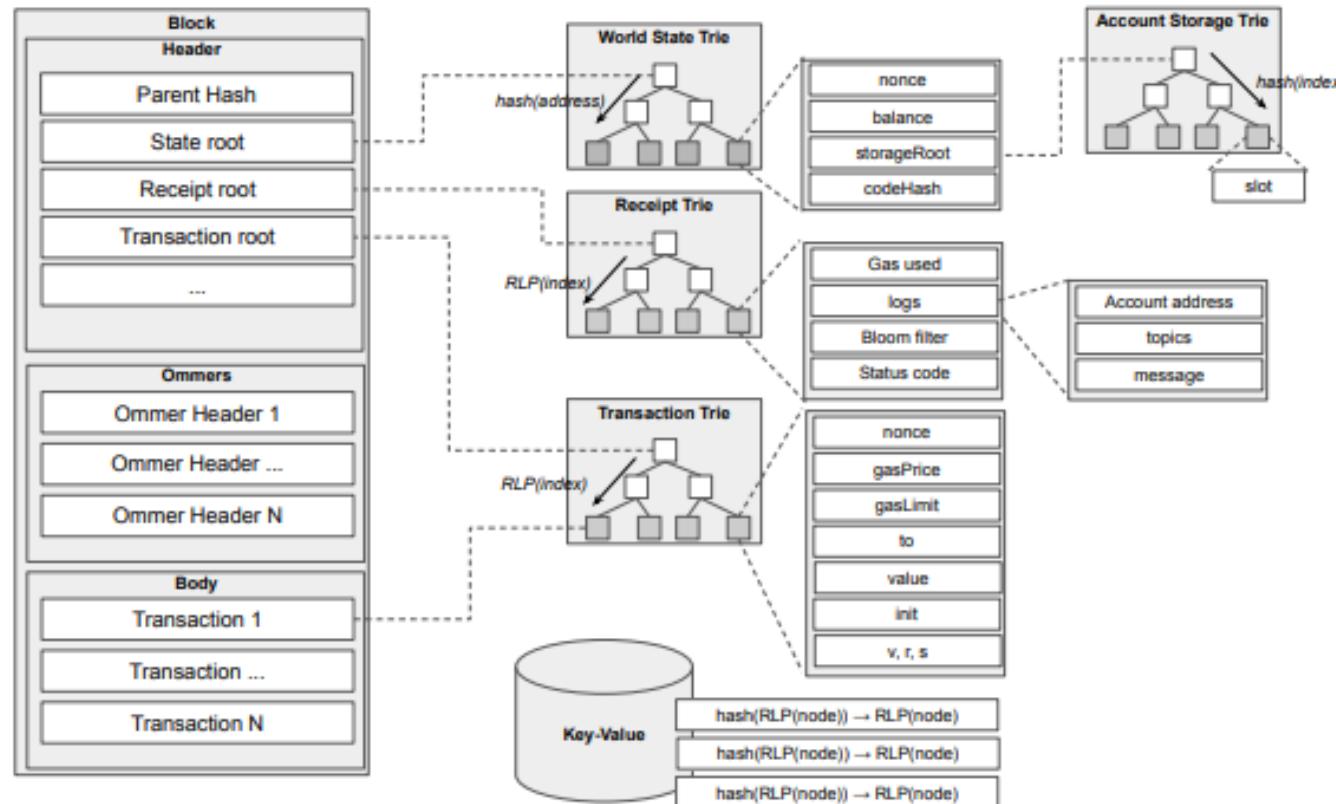
1. world state trie,
2. transaction trie
3. transaction receipt trie
4. account storage trie.

Each state trie is constructed with Merkle Patricia Trie and only root node (top node of state trie) is stored in block to spare storage.

# Blocks in Ethereum

- We can find that a block  $B$  contains a header  $H$ , transactions  $T$  and ommers, sometimes referred to as uncles  $U$  :

$$B = (H, T, U)$$



# Block Header

---

- **parentHash** The Keccak 256 bit hash of the parent block. This field connects blocks in a chain.
- **ommersHash** The Keccak 256 bit hash of list of ommers.
- **beneficiary** An account address of a user who mined this block and receives reward.
- **stateRoot** The Keccak 256 bit hash of a root of the World State Trie.
- **receiptsRoot** The Keccak 256 bit hash of a root of the Transaction Receipt Trie.
- **transactionsRoot** The Keccak 256 bit hash of a root of the Transaction Trie.
- **logsBloom** A filter relating logs hashes with the log records.
- **difficulty** A scalar value corresponding to an effort that has to be undertaken to mine this block.
- **number** A scalar value equivalent to an ordinal number of this block. Every new block in the chain gets a number increased by one.
- **gasLimit** A scalar value containing an accumulated gas limit required to process all transactions in this block.
- **gasUsed** A scalar value containing an accumulated real consumed gas to process all transactions in this block.
- **extraData** A free form, 32 bytes or less long byte array, containing any additional data.
- **mixHash** The Keccak 256 bit hash confirming that a sufficient computation has been spent on mining this block.
- **nonce** A 64 bit value. This value combined with mixHash also proves that the computation has been spend for mining this block.

# Transactions Trie

---

A transaction is mapped in the trie so that the key is a transaction *index* and the value is the transaction  $T$ . Both the transaction index and the transaction itself are *RLP* encoded. It compose a key-value pair, stored in the trie:

$$RLP(index) \rightarrow RLP(T)$$

- **Nonce**: an ordinal number of a transaction. For every new transaction submitted by the same sender, the nonce is increased. Prevents sending the same transaction twice.
- **gasPrice**: A value indicating current price of gas.
- **gasLimit**: A value indicating maximal amount of gas the sender is able to spend on executing this transaction.
- **to**: Address of an account to receive funds, or zero for contract creation.
- **value**: Amount of funds to transfer between external accounts, or initial balance of an account for a new contract.
- **init**: EVM initialization code for new contract creation.
- **data**: Input data for a message call together with the message (i.e. a method) signature.
- **v, r, s**: Values encoding signature of a sender.

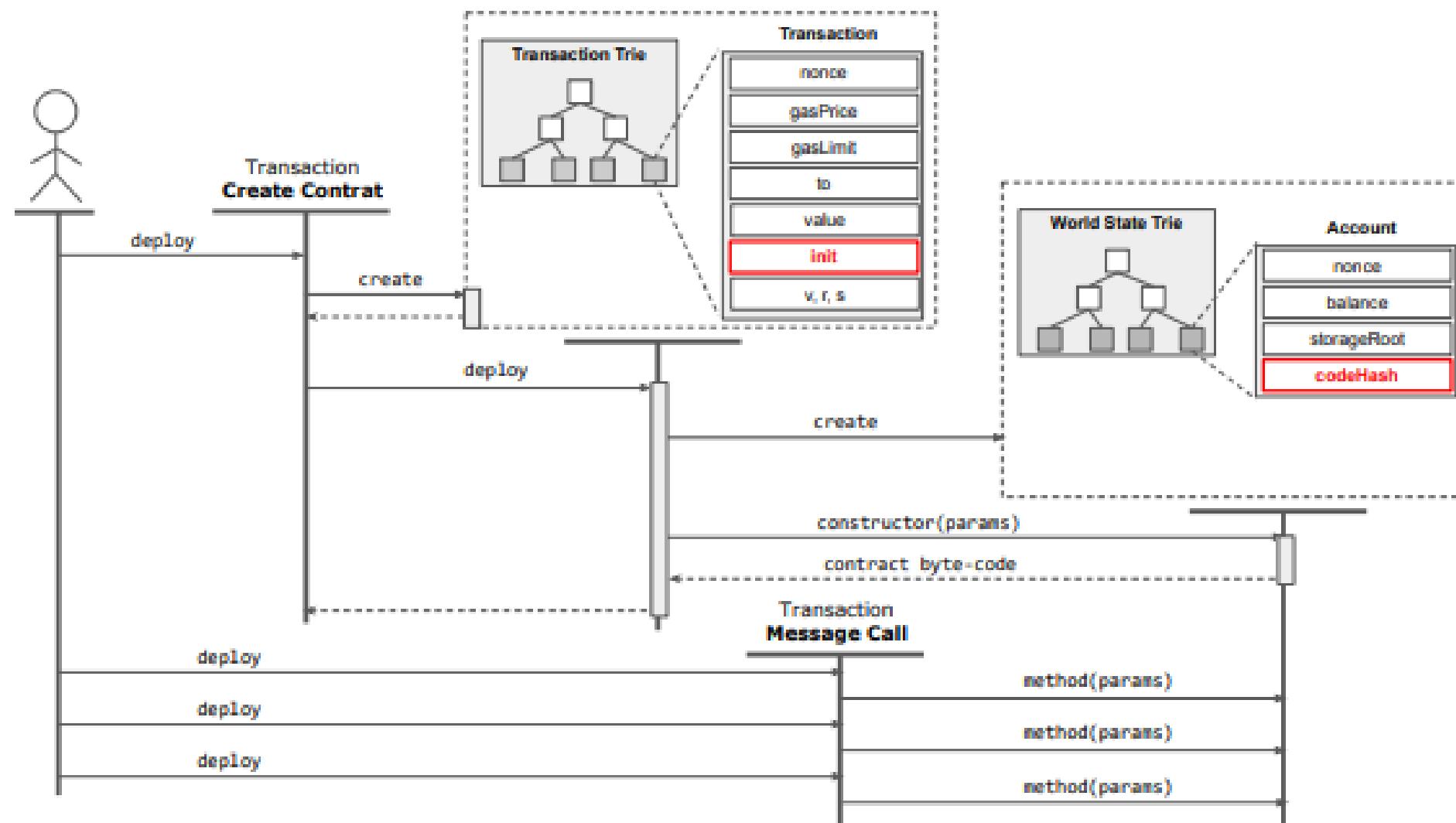
# Transactions Trie

---

Field	to	data	init
External account	recipient address		
Contract creation			byte-code + data
Message call	contract address.	signature + data	

Three types of transactions and the mapping of respective fields

# Contract Creation



# Message Calls

---

- Message call execution of a method triggered by a special transaction which has non-empty field data
- The transaction that calls a contract combines both the method signature and the input data into the **data field**
- The method signature is stored in hashed form followed by the input data
- The data field is encoded in standardized binary format **Abstract Binary Interface**
- Contracts may call methods of other contracts via internal transactions.
- Internal transactions are **not recorded in the blockchain**
- They do not have the gas limit and consumed gas is billed against the source caller
- This allows creating libraries which are not priced themselves

# Transaction Receipt

---

- When a transaction is submitted to a blockchain, it is not executed immediately.
- First executes the transaction the miner, who potentially put it in the block.
- Then, the transaction becomes part of the blockchain and all participating nodes will sooner or later synchronize, i.e. execute the transaction.
- **Because of this process, a user submitting originally the transaction cannot have its results immediately.**

# Transaction Receipt Trie

---

- As a solution, result of each transaction execution is captured in the **Transaction Receipt Trie**.
- One entry in this trie is created for each transaction
- Optionally, the trie may contain log messages generated by smart contracts.
- By analyzing the Receipts Trie, the user can study log messages to get deeper insight.
- Ethereum nodes provide an API that allows the user to register for events and get notifications when a transaction receipt has arrived.

# Transaction Receipt Trie

---

- The Receipt Trie has following structure. The **key is a transaction index** and the value is a **receipt  $R$** . They compose a key-value pair:

$$RLP(index) \rightarrow RLP(R)$$

**The receipt  $R$  has following fields:**

- Cumulative gas used in a respective block after execution of current transaction.
- Set of logs printed by this transaction
- A Bloom Filter containing hashes of log entries
- A status code of the transaction result, expressed as a number. Its semantics is application dependent.

# Blooms Filter

---

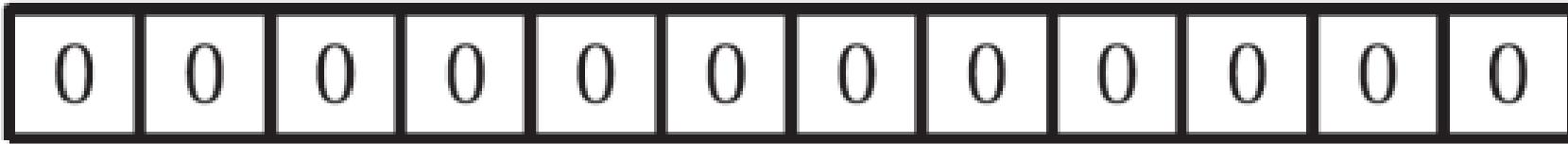
- Randomized data structure introduced by Burton Bloom [CACM 1970]
  - It represents a set for membership queries, with false positives
  - Probability of false positive can be controlled by design parameters
  - When space efficiency is important, a Bloom filter may be used if the effect of false positives can be mitigated.
  
- First applications in dictionaries and databases

# Standard Bloom Filter

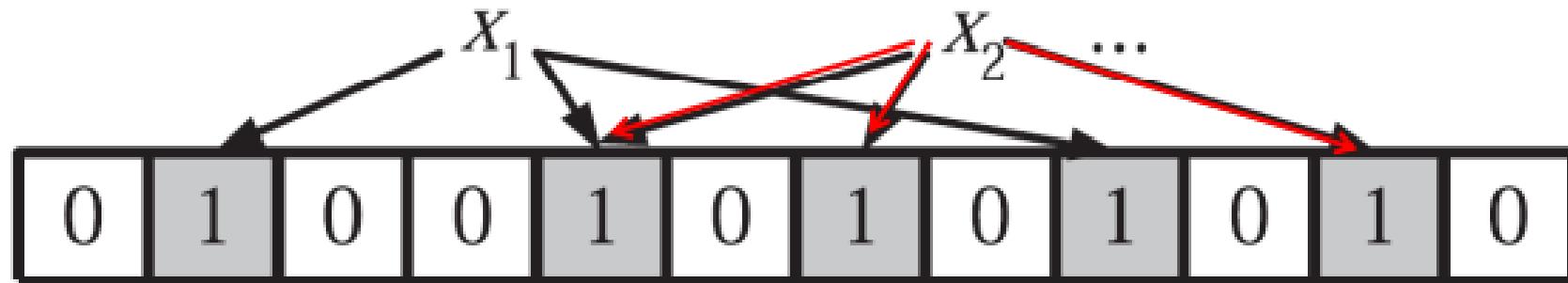
---

- A Bloom filter is an array of  $m$  bits representing a set  $S = \{x_1, x_2, \dots, x_n\}$  of  $n$  elements
  - Array set to 0 initially
- $k$  independent hash functions  $h_1, \dots, h_k$  with range  $\{1, 2, \dots, m\}$ 
  - Assume that each hash function maps each item in the universe to a random number *uniformly* over the range  $\{1, 2, \dots, m\}$
- For each element  $x$  in  $S$ , the bit  $h_i(x)$  in the array is set to 1, for  $1 \leq i \leq k$ ,
  - A bit in the array may be set to 1 multiple times for different elements

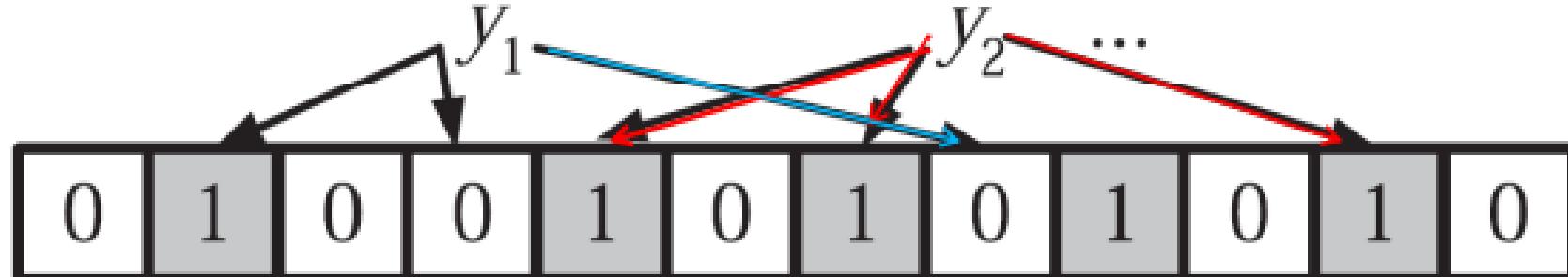
# Bloom filter example



Insert  $X_1$  and  $X_2$



Check  $Y_1$  and  $Y_2$



# Standard Bloom Filter (cont.)

---

- To check membership of  $y$  in  $S$ , check whether  $h_i(y)$ ,  $1 \leq i \leq k$ , are all set to 1
  - If not,  $y$  is definitely not in  $S$
  - Else, we conclude that  $y$  is in  $S$ , but sometimes this conclusion is wrong (false positive)
- For many applications, false positives are acceptable as long as the probability of a false positive is small enough
- We will assume that  $kn < m$

# False positive probability

---

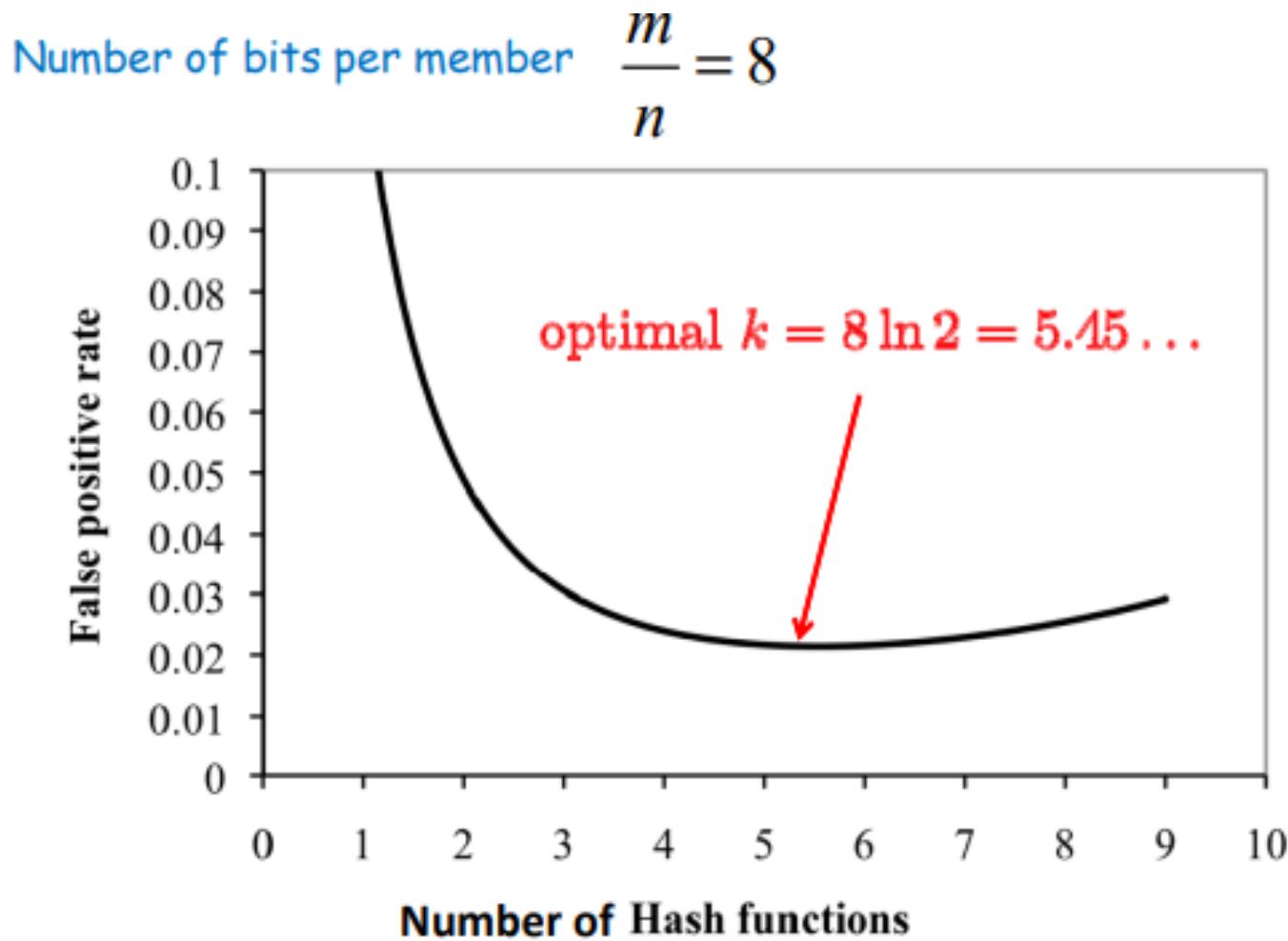
- After all members of  $S$  have been hashed to a Bloom filter, the probability that a specific bit is still 0 is

$$p' = \left(1 - \frac{1}{m}\right)^{kn} \simeq e^{-kn/m} = p$$

- For a non member, it may be found to be a member of  $S$  (all of its  $k$  bits are nonzero) with *false positive probability*

$$(1 - p')^k \simeq (1 - p)^k$$

# False positive rate vs. k



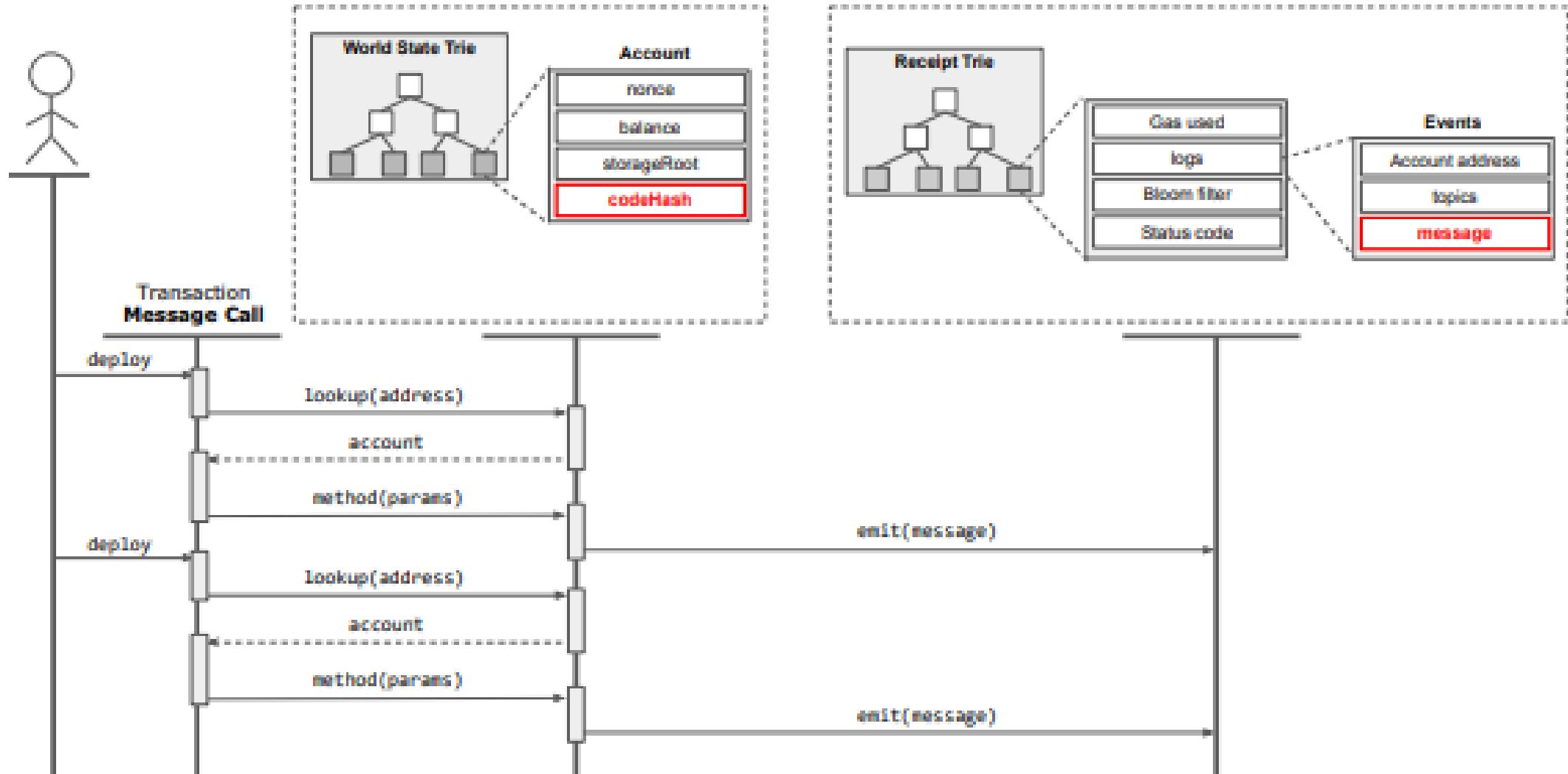
# Message Logs

---

- The Receipt Trie can contain a set of log messages, produced by smart contract.
- In the case of Solidity, the log message is emitted via a keyword **emit**.
- The set of logs contains tuples with following items:
  - A contract account address, who triggered the log, i.e. executed this Smart Contract.
  - A sequence of 32 bytes long containing event topics.
  - A sequence of bytes containing the log message itself
- The log messages may contain any arbitrary data and every message may be assigned to a topic. This way clients may listen only to messages from certain topics if they are not interested in all log messages.

# Log Event Updates Transaction Receipt

```
1 contract ExampleContract {  
2     uint storedData  
3     event Sent(msg);  
4     constructor() public {}  
5  
6     function method(uint x) public {  
7         storedData = x  
8         emit Sent('Success');  
9     }  
10 }
```



# World State Trie

---

- Also known as **State Trie or Global State Trie**
- In contrast to Transaction Trie it is mutable data structure
- World state trie is a mapping between addresses and account states.
  - Keccak(address) -> RLP(A)
- It can be seen as a global state that is constantly updated by transaction executions.
- The Ethereum network is a decentralized computer and state trie is considered as hard drive.
- All the information about accounts are stored in world state trie and you can retrieve information by querying it.
- World state trie is closely related to account storage trie because it has “storageRoot” field that points the root node in account storage trie.

# World state trie node fields

- **nonce**

- Number of transactions sent from this address (if this is an External Owned Account - EOA) or the number of contract-creations made by this account

- **balance**

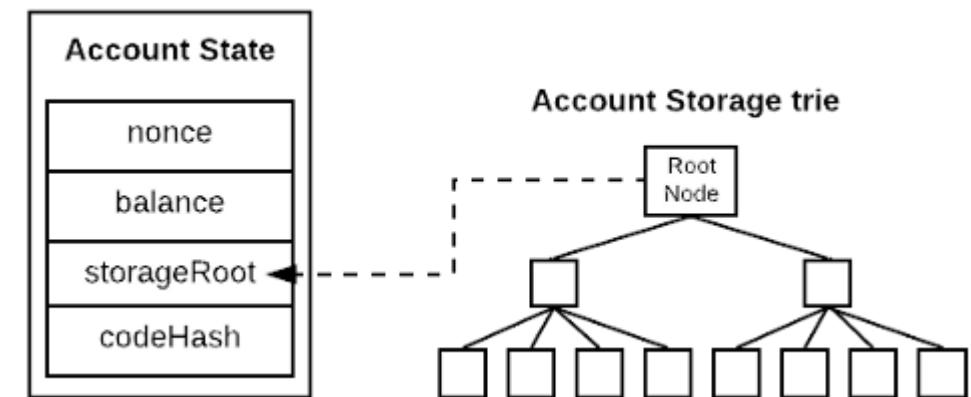
- Total Ether (in Wei) owned by this account.

- **storageRoot**

- A 256-bit hash root of the account storage trie. Empty for EOAs

- **codeHash**

- Hash of EVM code. The bytecode is stored in underlying database under this hash key. Empty for EOA



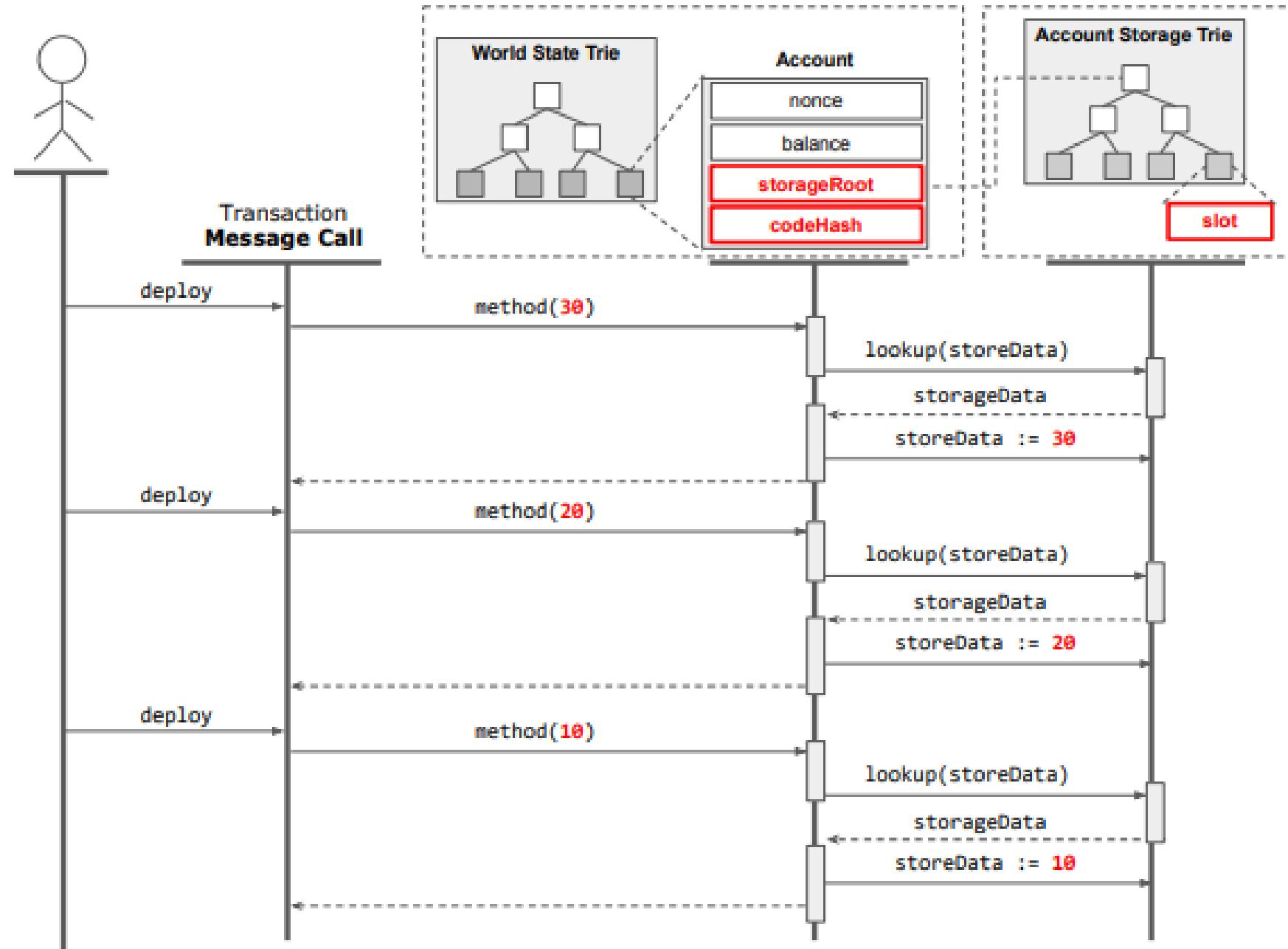
One important details about the account state is that **all fields (except the codeHash) are mutable**.

# Account Storage Trie

---

- Account Storage Trie is where data associated with an account is stored.
- Accessed via storageRoot field in world state trie account
- This try is directly modified using smart contract byte code SLOAD and SSTORE
- Every key in this trie is an index of a slot stored in the leaf node
  - Keccak(index) -> RLP(slot)
- Smart contract data is persisted in the account storage trie as a mapping between 32-bytes integers.

# Storage Trie Updates



# Smart Contract Programming

- Solidity (javascript based), most popular
  - Not yet as functional as other, more mature, programming languages
- Serpent (python based)
- LLL (lisp based)

# Smart Contract Programming

## Solidity

Solidity is a language similar to JavaScript which allows you to develop contracts and compile to EVM bytecode. It is currently the flagship language of Ethereum and the most popular.

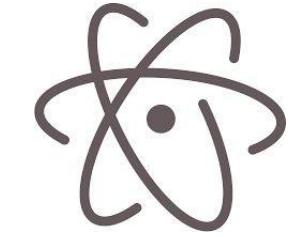
- [Solidity Documentation](#) - Solidity is the flagship Ethereum high level language that is used to write contracts.
- [Solidity online realtime compiler](#)

## Serpent

Serpent is a language similar to Python which can be used to develop contracts and compile to EVM bytecode. It is intended to be maximally clean and simple, combining many of the efficiency benefits of a low-level language with ease-of-use in programming style, and at the same time adding special domain-specific features for contract programming. Serpent is compiled using LLL.

- [Serpent on the ethereum wiki](#)
- [Serpent EVM compiler](#)

# Smart Contract Programming



[\*\*Atom Ethereum interface\*\*](#) - Plugin for the Atom editor that features syntax highlighting, compilation and a runtime environment (requires backend node).

[\*\*Atom Solidity Linter\*\*](#) - Plugin for the Atom editor that provides Solidity linting.



[\*\*Vim Solidity\*\*](#) - Plugin for the Vim editor providing syntax highlighting.

[\*\*Vim Syntastic\*\*](#) - Plugin for the Vim editor providing compile checking.

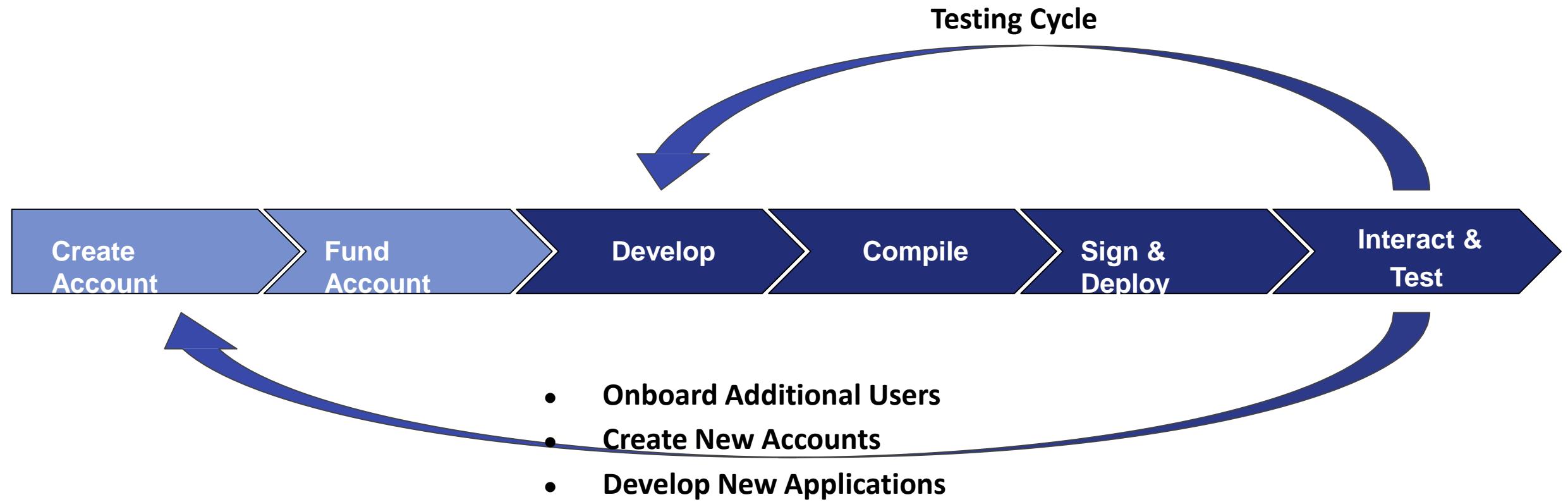
# Smart Contract Programming: Solidity

```
contract Example {  
  
    uint value;  
  
    function setValue(uint pValue)  
    {    value = pValue;  
    }  
  
    function getValue() returns (uint)  
    {    return value;  
    }  
}
```

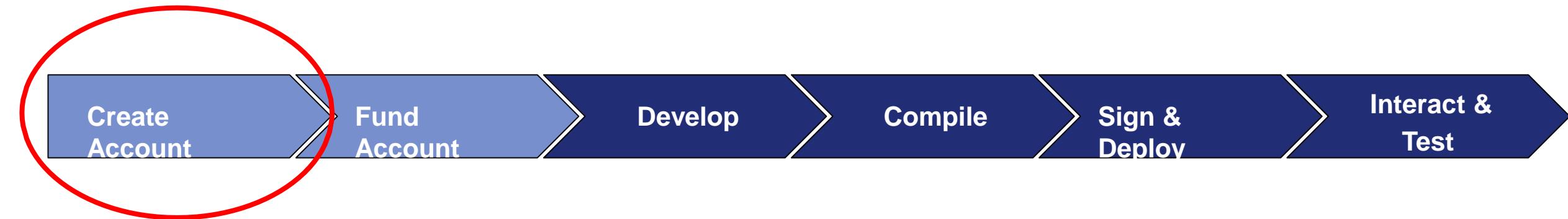
# Smart Contract Programming: Solidity

```
var logIncrement =  
    OtherExample.LogIncrement({sender: userAddress,  
    uint value});  
  
logIncrement.watch(function(err, result) {  
    // do something with result  
})
```

# Development Workflow

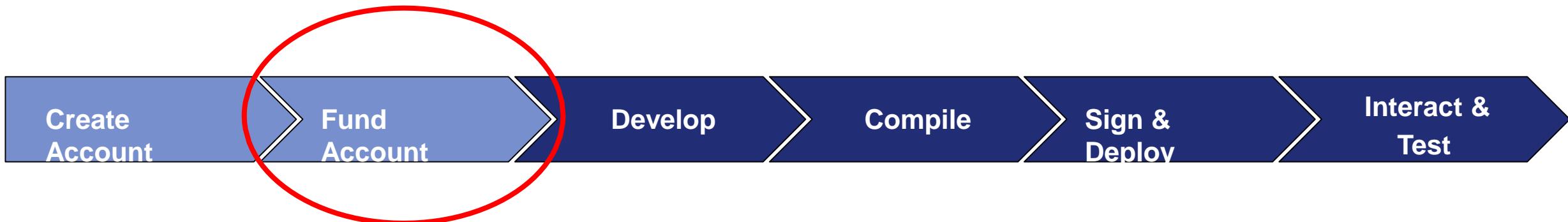


# Development Workflow: Create Account



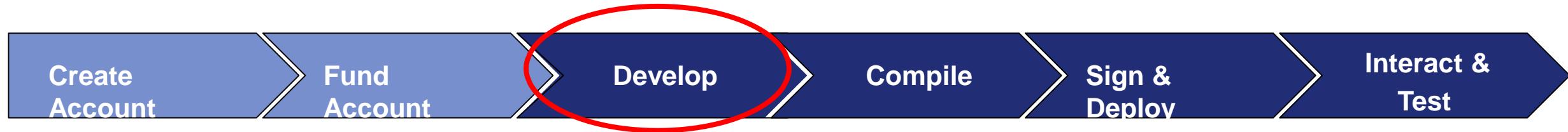
- Programmatically: Go, Python, C++, JavaScript, Haskell
- Tools
  - MyEtherWallet.com
  - MetaMask
  - TestRPC
  - Many other websites

# Development Workflow: Fund Account



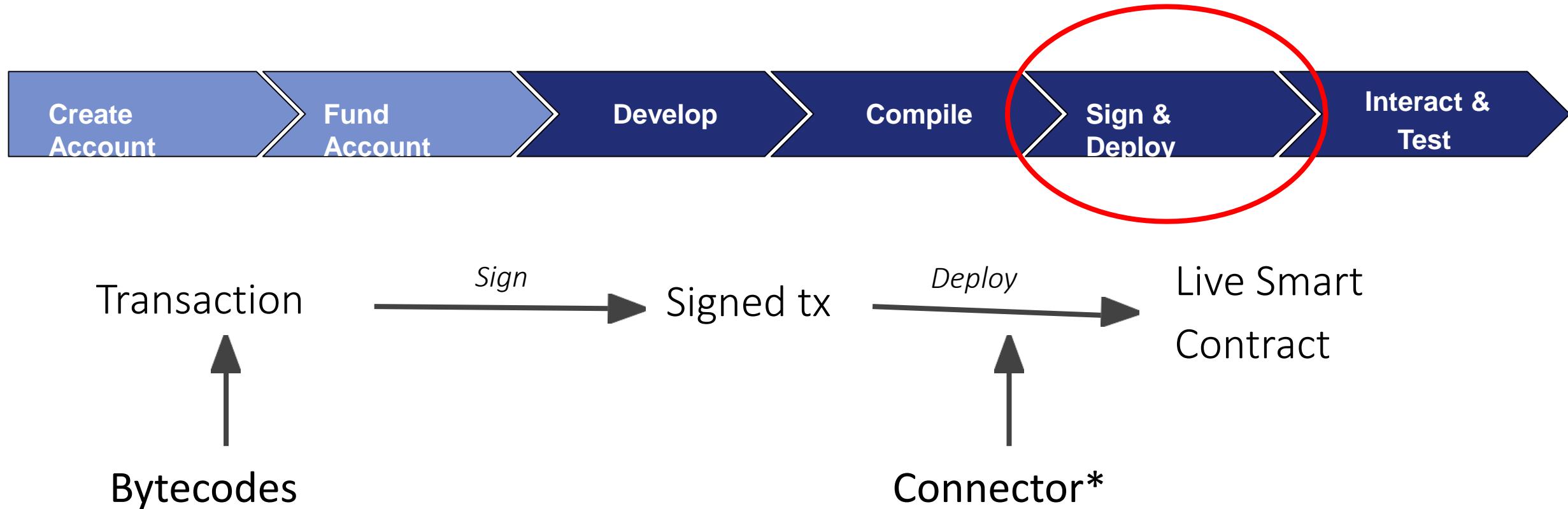
- From friends
- Faucet
- Exchanges (for public blockchain)

# Development Workflow: Develop



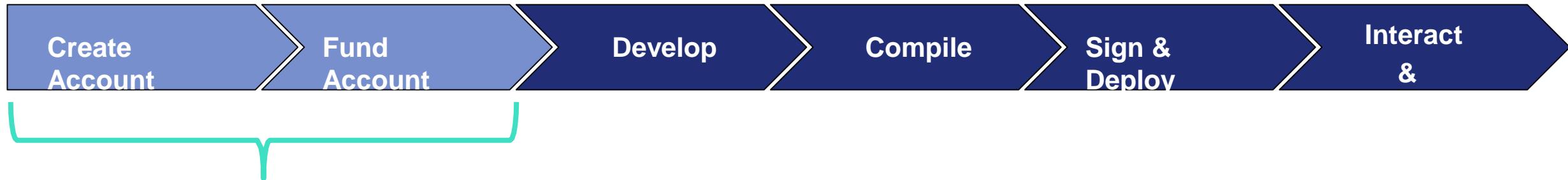
- **Ethereum Application Components:**
  - **Base application:** can be developed in **any** language
  - **Smart contract:** developed in Solidity or one of the other contract compatible languages
  - **Connector library:** facilitates communication between base application and smart contracts (Metamask)

# Development Workflow: Sign and Deploy



\*Library that facilitates communication and connection with Blockchain; Connects your code to a running node.

# Development Workflow: TestRPC



## TestRPC/TestChain

- Local development or Test Blockchain
- <https://github.com/ethereumjs/testrpc>

# Development Workflow: TestRPC

- EthereumJS TestRPC: <https://github.com/ethereumjs/testrpc> is suited for development and testing
- It's a complete blockchain-in-memory that runs only on your development machine
- It processes transactions instantly instead of waiting for the default block time – so you can test that your code works quickly – and it tells you immediately when your smart contracts run into errors
- It also makes a great client for automated testing
- Truffle knows how to use its special features to speed up test runtime by almost 90%.

# Blockchain Technology

## Introduction to Ethereum Tokens

Ashutosh Bhatia

BITS Pilani

[ashutosh.bhatia@pilani.bits-pilani.ac.in](mailto:ashutosh.bhatia@pilani.bits-pilani.ac.in)

# Token on Blockchain

---

- Block chain based abstractions (logical entities) that can be owned and that represent asset, currency access rights etc.
- Unlike physical token that are not really exchangeable and often restricted to specific businesses, organization and locations

# How token are used in Blockchain

---

- **Currency** : A token can serve as a form of currency, with a value determined through private trade.
- **Resource** : A token can represent a resource earned or produced in a sharing economy. Ex. storage or CPU token representing resources
- **Asset**: A token can represent ownership of an intrinsic or extrinsic, tangible or intangible asset; Ex. gold, real estate, a car, oil, energy, etc.
- **Access**: A token can represent access rights and grant access to a digital or physical property, such as a discussion forum, an exclusive website, a hotel room, or a rental car.
- **Voting**: A token can represent voting rights in a digital or legal system.
- **Collectible**: A token can represent a digital collectible (e.g., CryptoPunks) or physical collectible (e.g., a painting).
- **Identity**: A token can represent a digital identity (e.g., avatar) or legal identity (e.g., national ID).

# Tokens and Fungibility

---

- In economics, fungibility is the property of a good or a commodity whose individual units are essentially interchangeable.
- Tokens are fungible when we can substitute any single unit of the token for another without any difference in its value or function.
- Non-fungible tokens are tokens that each represent a unique tangible or intangible item and therefore are not interchangeable.
  - Digital Collectables: In CryptoKitties you can breed and adopt Kitties of all colours and shapes. Create Collections of your favourite cats and share them with our breeding community.

# Counterparty Risks

---

- Counterparty risk is the risk that the *other* party in a transaction will fail to meet their obligations.
- when an asset is traded indirectly through the exchange of a token of ownership, there is additional counterparty risk from the custodian of the asset.
  - Do they have the asset?
  - Will they recognize (or allow) the transfer of ownership based on the transfer of a token (such as a certificate, deed, title, or digital token)?
  - In the world of digital tokens representing assets, as in the non-digital world, it is important to understand who holds the asset that is represented by the token and what rules apply to that underlying asset.

# Tokens on Ethereum

---

- Blockchain tokens existed before Ethereum. In some ways, the first blockchain currency, Bitcoin, is a token itself.
- Many token platforms were also developed on Bitcoin and other cryptocurrencies before Ethereum.
- However, the introduction of the first token standard on Ethereum led to an explosion of tokens.
- Vitalik Buterin suggested tokens as one of the most obvious and useful applications of a generalized programmable blockchain such as Ethereum.

# Tokens on Ethereum

---

- Tokens are different from ether because the Ethereum protocol does not know anything about them.
- Sending ether is an intrinsic action of the Ethereum platform, but sending or even owning tokens is not.
- The ether balance of Ethereum accounts is handled at the protocol level, whereas the token balance of Ethereum accounts is handled at the smart contract level.
- In order to create a new token on Ethereum, you must create a new smart contract.
- Once deployed, the smart contract handles everything, including ownership, transfers, and access rights.
- You can write your smart contract to perform all the necessary actions any way you want, but it is probably wisest to follow an existing standard.
- We will look at such standards next. We discuss the pros and cons of the following standards

# The ERC20 Token Standard

---

- The first standard was introduced in November 2015 by Fabian Vogelsteller as an Ethereum Request for Comments (ERC).
- It was automatically assigned GitHub issue number 20, giving rise to the name “ERC20 token.”
- The vast majority of tokens are currently based on the ERC20 standard.
- ERC20 is a standard for *fungible tokens*
  - different units of an ERC20 token are interchangeable
- The ERC20 standard defines a common interface for contracts implementing a token, such that any compatible token can be accessed and used in the same way.
  - The interface consists of a number of functions that must be present in every implementation of the standard, as well as some optional functions and attributes that may be added by developers.

# Using Tokens: Utility or Equity

---

- Almost all projects in Ethereum today launch with some kind of token. But do all these projects really need tokens?
- The majority of projects are using tokens in one of two ways: either as “**utility tokens**” or as “**equity tokens**.”
- Utility tokens is required to gain access to a service, application, or resource.
  - Examples of utility tokens include tokens that represent resources such as shared storage, or access to services such as social media networks.
- Equity tokens represent shares in the control or ownership of something.
  - Equity tokens can be as limited as nonvoting shares for distribution of dividends and profits, or as expansive as voting shares in a decentralized autonomous organization

# Utility Tokens: Who Needs Them?

---

- The real problem is that utility tokens introduce significant risks and adoption barriers for startups.
- Perhaps in a distant future “tokenize all the things” will become reality, but at present the set of people who have an understanding of and desire to use a token is a subset of the already small cryptocurrency market.
- For a startup, each innovation represents a risk that works as the barrier for the adoption of the technology by the users
  - Still a very small portion of the world believe in blockchain technology
  - Only a subset of those people would be ready to use the service offered by your innovation over this technology
  - Adding utility tokens to that may further reduce the degree of adoption of the project.
- Nevertheless, some of the most innovative business ideas are indeed taking place in the crypto realm.
  - If regulators are not quick enough to adopt laws and support new business models, entrepreneurs and associated talent will seek to operate in other jurisdictions that are more crypto-friendly. This is already happening.

# ERC20: functions and events

---

- An ERC20-compliant token contract must provide at least the following functions and events:
- `totalSupply` : Returns the total units of this token that currently exist. ERC20 tokens can have a fixed or a variable supply.
- `balanceOf`: Given an address, returns the token balance of that address.
- `transfer`: Given an address and amount, transfers that amount of tokens to that address, from the balance of the address that executed the transfer.
- `transferFrom` : Given a sender, recipient, and amount, transfers tokens from one account to another. Used in combination with `approve`.
- `Approve` : Given a recipient address and amount, authorizes that address to execute several transfers up to that amount, from the account that issued the approval.
- `Allowance` : Given an owner address and a spender address, returns the remaining amount that the spender is approved to withdraw from the owner.
- `Transfer`: Event triggered upon a successful transfer (call to `transfer` or `transferFrom`) (even for zero-value transfers).
- `Approval`: Event logged upon a successful call to `approve`.



**BITS** Pilani  
Pilani Campus

# Blockchain Technology (BITS F452)

Dr. Ashutosh Bhatia, Dr. Kamlesh Tiwari  
Department of Computer Science and Information Systems





# ***BITCOIN: Blocks and P2P Network***

Source: Bitcoin and Cryptocurrency Technologies Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, Steven Goldfeder

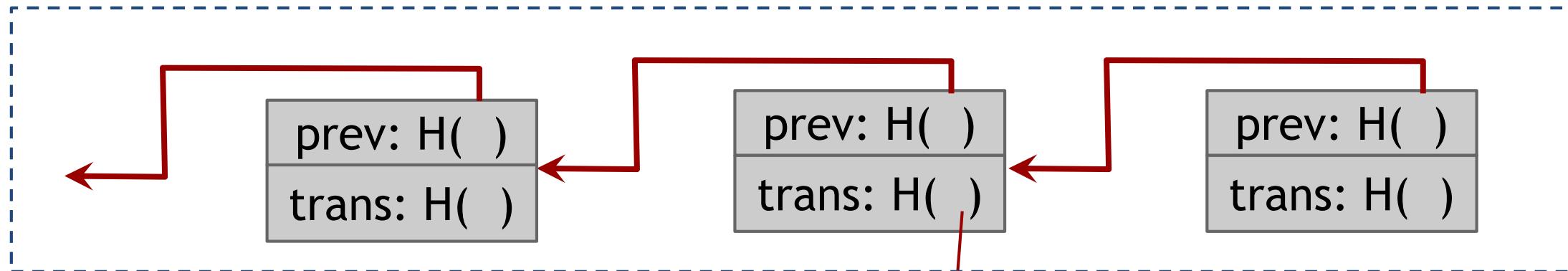
# Bitcoin blocks

Why bundle transactions together?

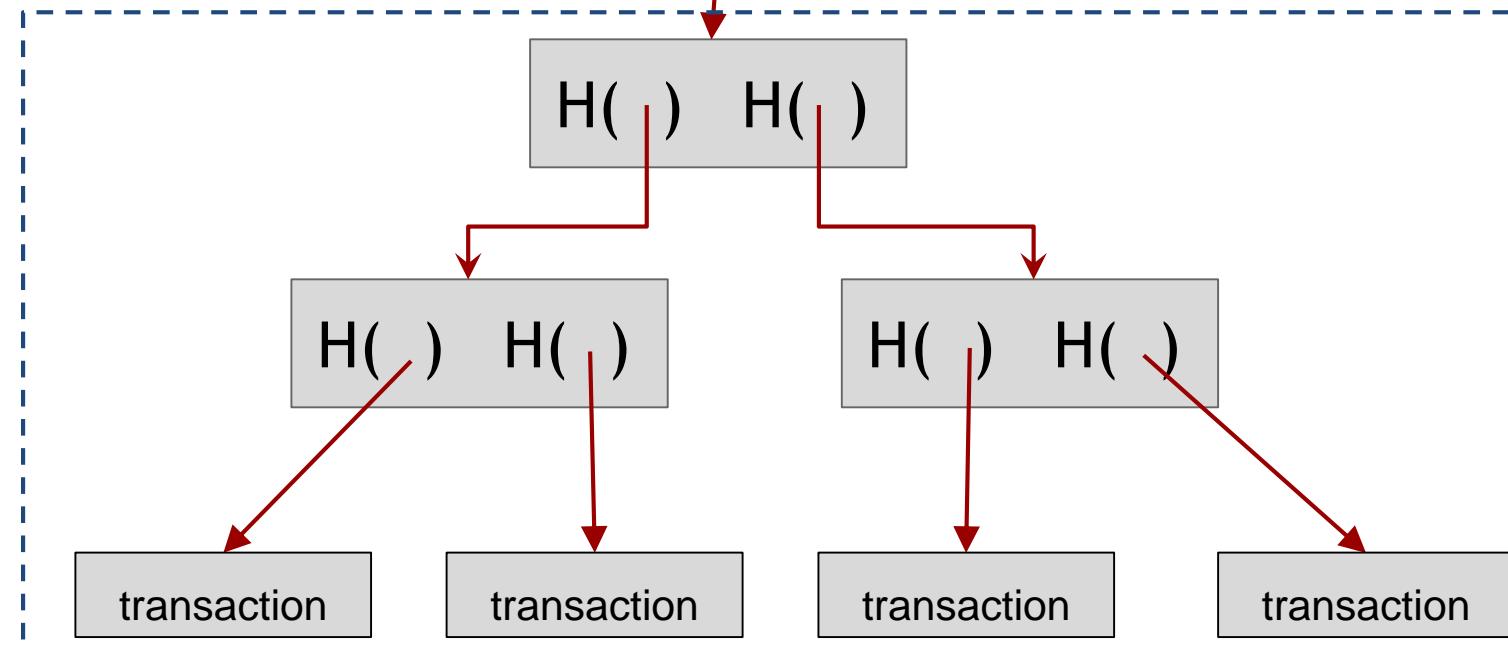
- Single unit of work for miners
- Limit length of hash-chain of blocks
  - Faster to verify history

# Bitcoin block structure

Hash chain of blocks



Hash tree (Merkle tree)  
of transactions in each  
block



# The real deal: a Bitcoin block

block  
header

transaction  
data

```
{  
    "hash":"00000000000000001aad2...",  
    "ver":2,  
    "prev_block":"00000000000000003043...",  
    "time":1391279636,  
    "bits":419558700,  
    "nonce":459459841,  
    "mrkl_root":"89776...",  
    "n_tx":354,  
    "size":181520,  
    "tx": [  
        ...  
    ],  
    "mrkl_tree": [  
        "6bd5eb25...",  
        ...  
        "89776cdb..."  
    ]  
}
```

# The real deal: a Bitcoin block header

```
{  
  "hash":"00000000000000001aad2...",  
  "ver":2,  
  "prev_block":"00000000000000003043...",  
  "time":1391279636,  
  "bits":419558700,  
  "nonce":459459841,  
  "mrkl_root":"89776...",  
  ...  
}
```

mining  
puzzle  
information

hashed  
during  
mining

not hashed

The diagram illustrates the structure of a Bitcoin block header. It shows a JSON-like object with various fields. A bracket on the right side groups the fields into two categories: 'hashed during mining' (which includes 'hash', 'prev\_block', 'time', 'bits', 'nonce', and 'mrkl\_root') and 'not hashed' (which includes the ellipsis '...'). Another bracket on the left side groups the first six fields ('hash', 'ver', 'prev\_block', 'time', 'bits', and 'nonce') under the label 'mining puzzle information'. Arrows point from the labels to their respective brackets.

# The real deal: coinbase transaction

redeeming  
nothing

arbitrary

```
"in":[]  
{  
    "prev_out":{  
        "hash":"000000....000000",  
        "n":4294967295  
    },  
    "coinbase": "..."  
},  
"out":[]  
{  
    "value":"25.03371419",  
    "scriptPubKey":"OPDUP OPHASH160 ... "  
}
```

Null hash pointer

First ever coinbase parameter:  
“The Times 03/Jan/2009 Chancellor  
on brink of second bailout for banks”

block  
reward  
transaction fees

# See for yourself!

## Transaction

View information about a bitcoin transaction

151b750d1f13e76d84e82b34b12688811b23a8e3119a1cba4b4810f9b0ef408d



1KryFUt9tXHvaoCYTNPbqpWPJKQ717YmL5

1KvrdrQ3oGqMAiDTMEYCcdDSnVaGNW2YZh  
1KryFUt9tXHvaoCYTNPbqpWPJKQ717YmL5

1.0194 BTC  
3.458 BTC

9 Confirmations

4.4774 BTC

### Summary

Size 257 (bytes)

Received Time 2014-08-05 01:55:25

Included In Blocks 314018 (2014-08-05 02:00:40 +5 minutes)

Confirmations 9 Confirmations

Relayed by IP ⓘ Blockchain.info

Visualize [View Tree Chart](#)

### Inputs and Outputs

Total Input 4.4775 BTC

Total Output 4.4774 BTC

Fees 0.0001 BTC

Estimated BTC Transacted 1.0194 BTC

Scripts [Show scripts & coinbase](#)

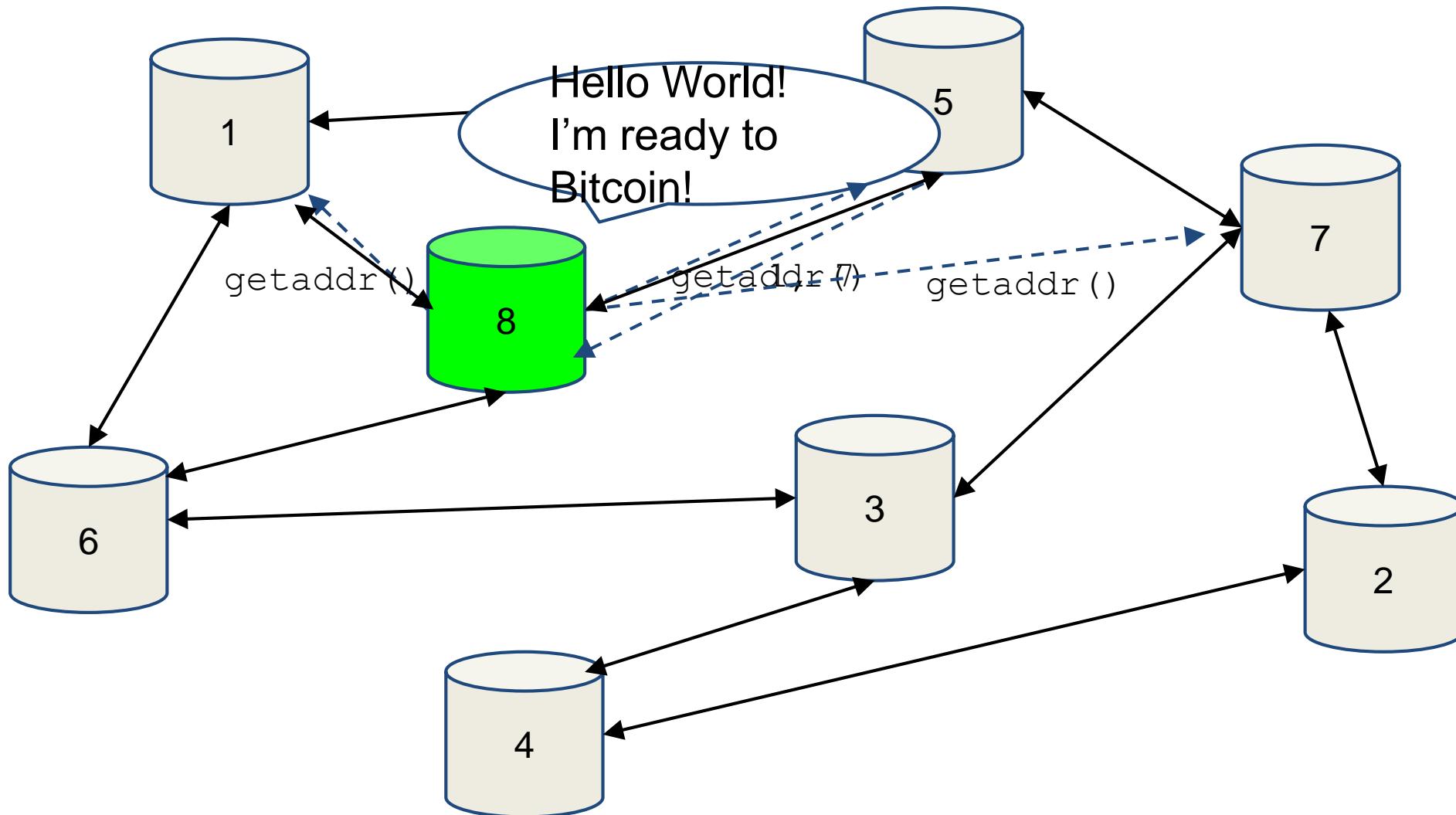
blockchain.info (and many other sites)

# The Bitcoin network

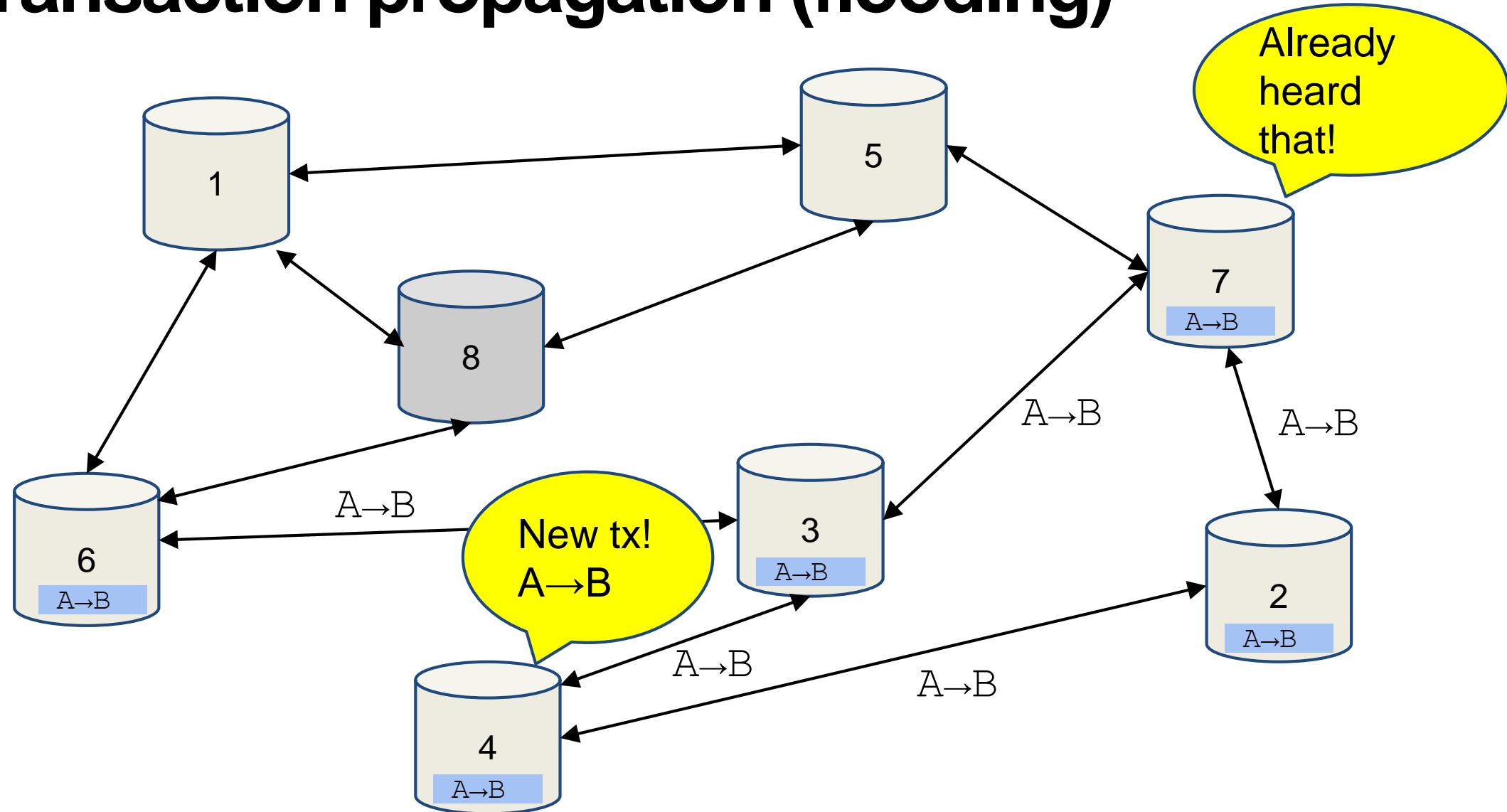
# Bitcoin P2P network

- Ad-hoc protocol (runs on TCP port 8333)
- Ad-hoc network with random topology
- All nodes are equal
- New nodes can join at any time
- Forget non-responding nodes after 3 hr

# Joining the Bitcoin P2P network



# Transaction propagation (flooding)



# Should I relay a proposed transaction?

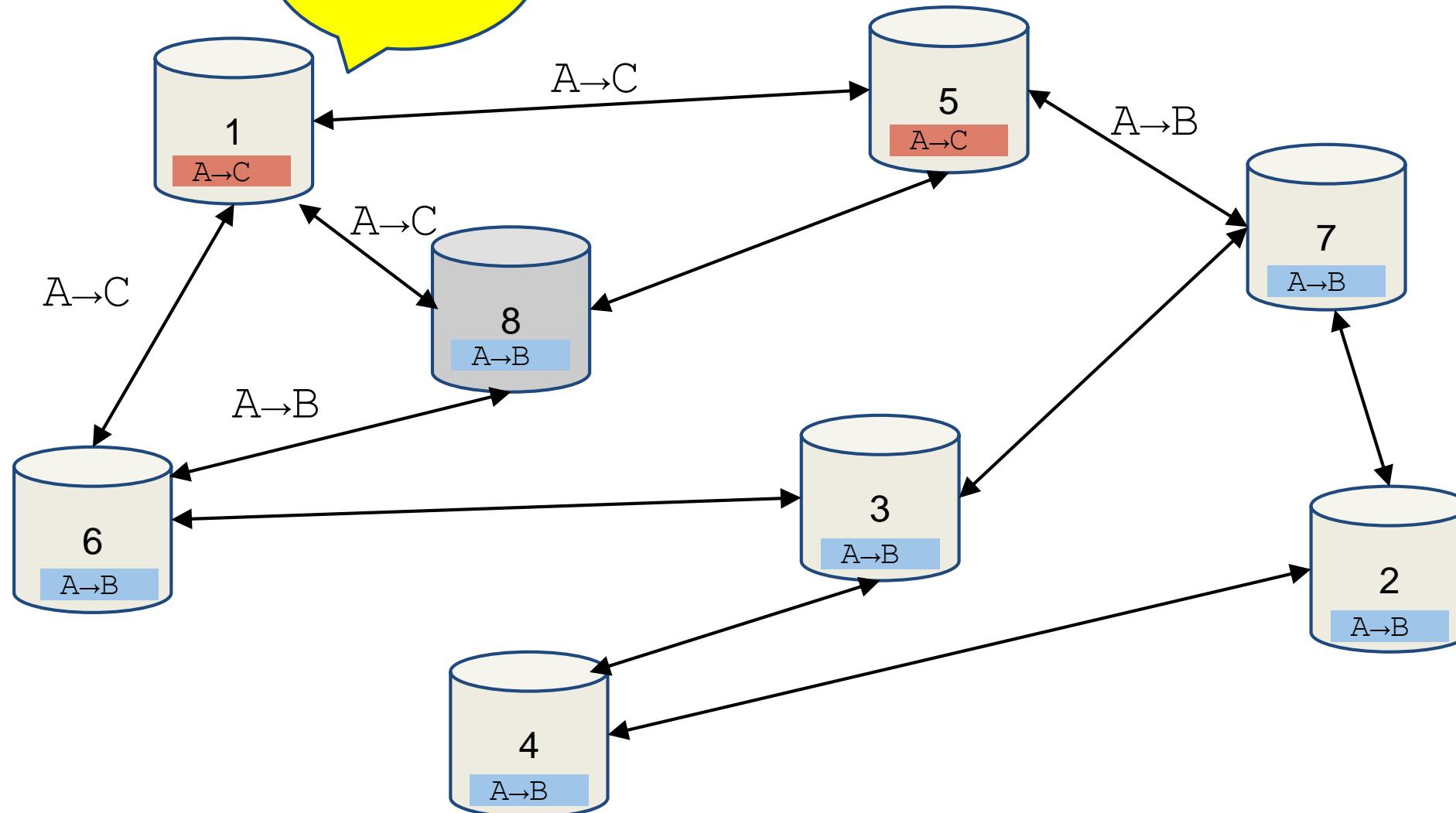
- Transaction valid with current block chain (default)
  - Run script for each previous output being redeemed and ensure that script returns true!
- Script matches a whitelist
  - Avoid unusual scripts
- Haven't seen before
  - Avoid infinite loops
- Doesn't conflict with others I've relayed
  - Avoid double-spends

Sanity checks only...

Well-behaving nodes implement them!

Some nodes may ignore them!

# Nodes make a peer-to-peer on transaction pool



# Race conditions

Transactions or blocks may *conflict*

- Default behavior: accept what you hear first
- Network position matters
- Miners may implement other logic!

Stay tune for the lecture on mining!

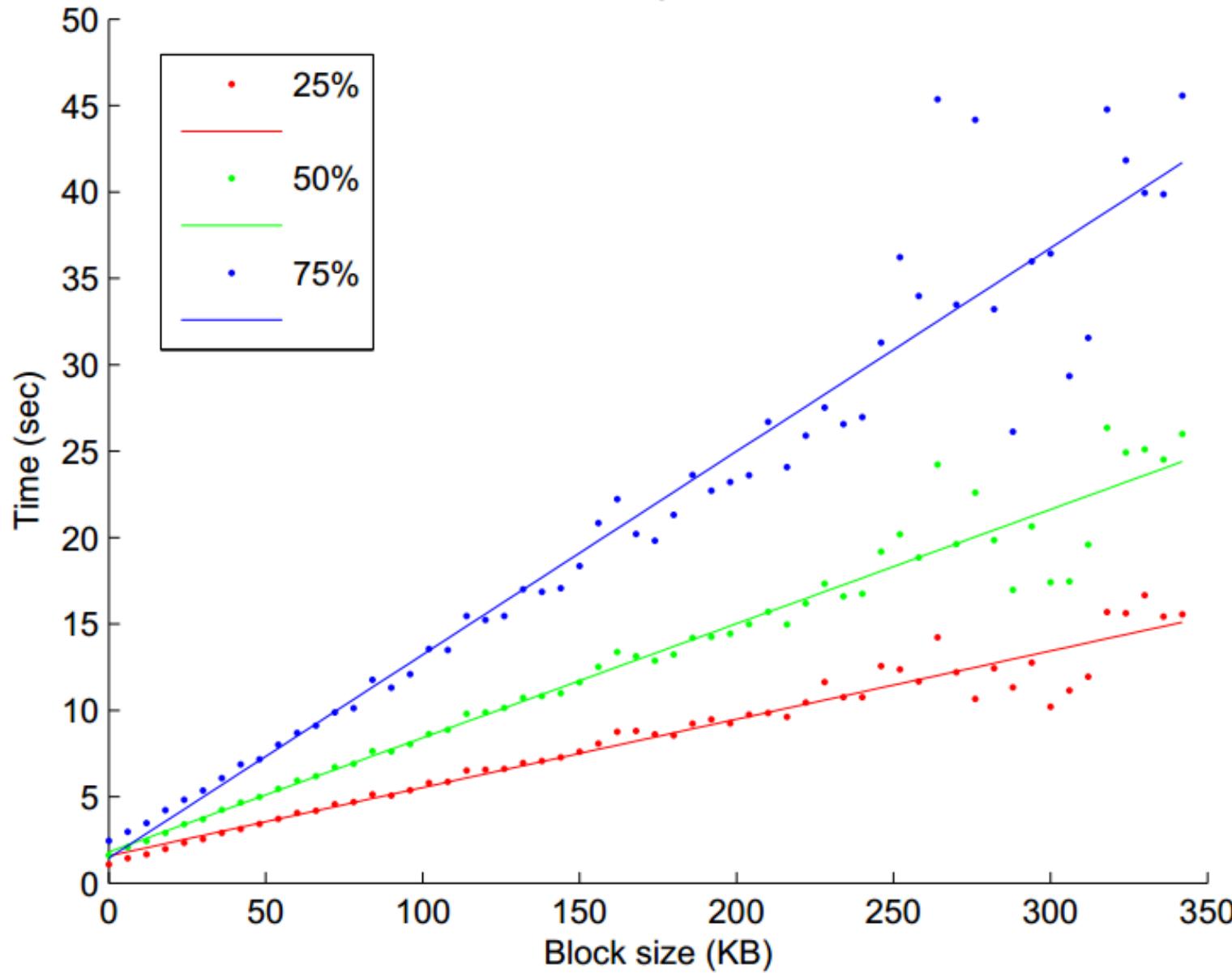
# Block propagation nearly identical

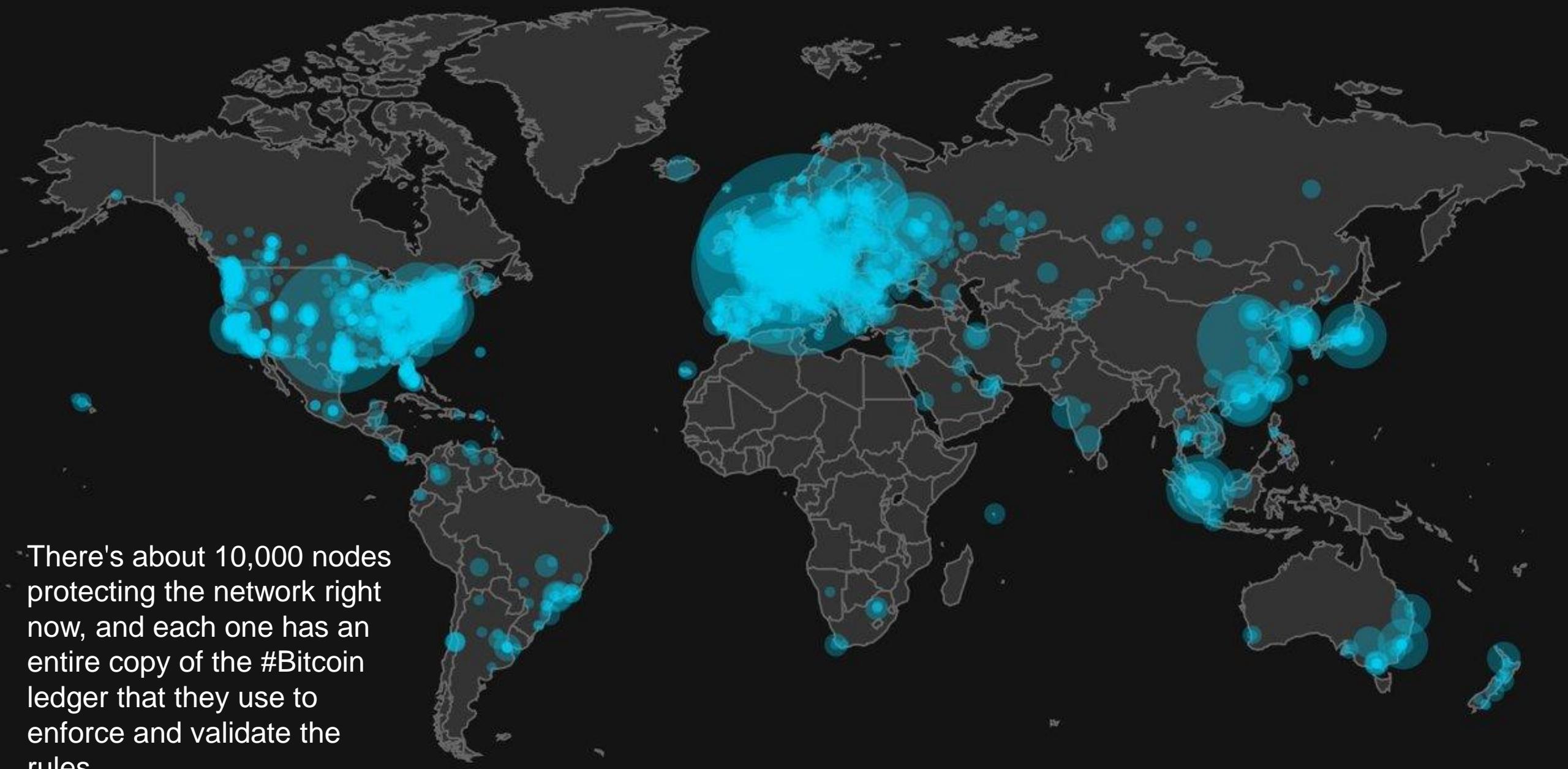
Relay a new block when you hear it if:

- Block meets the hash target
- Block has all valid transactions
  - Run *all* scripts, even if you wouldn't relay
- Block builds on current longest chain
  - Avoid forks

Sanity check  
Also may be ignored...

### Block Propagation Times





There's about 10,000 nodes protecting the network right now, and each one has an entire copy of the #Bitcoin ledger that they use to enforce and validate the rules.

# How big is the network?

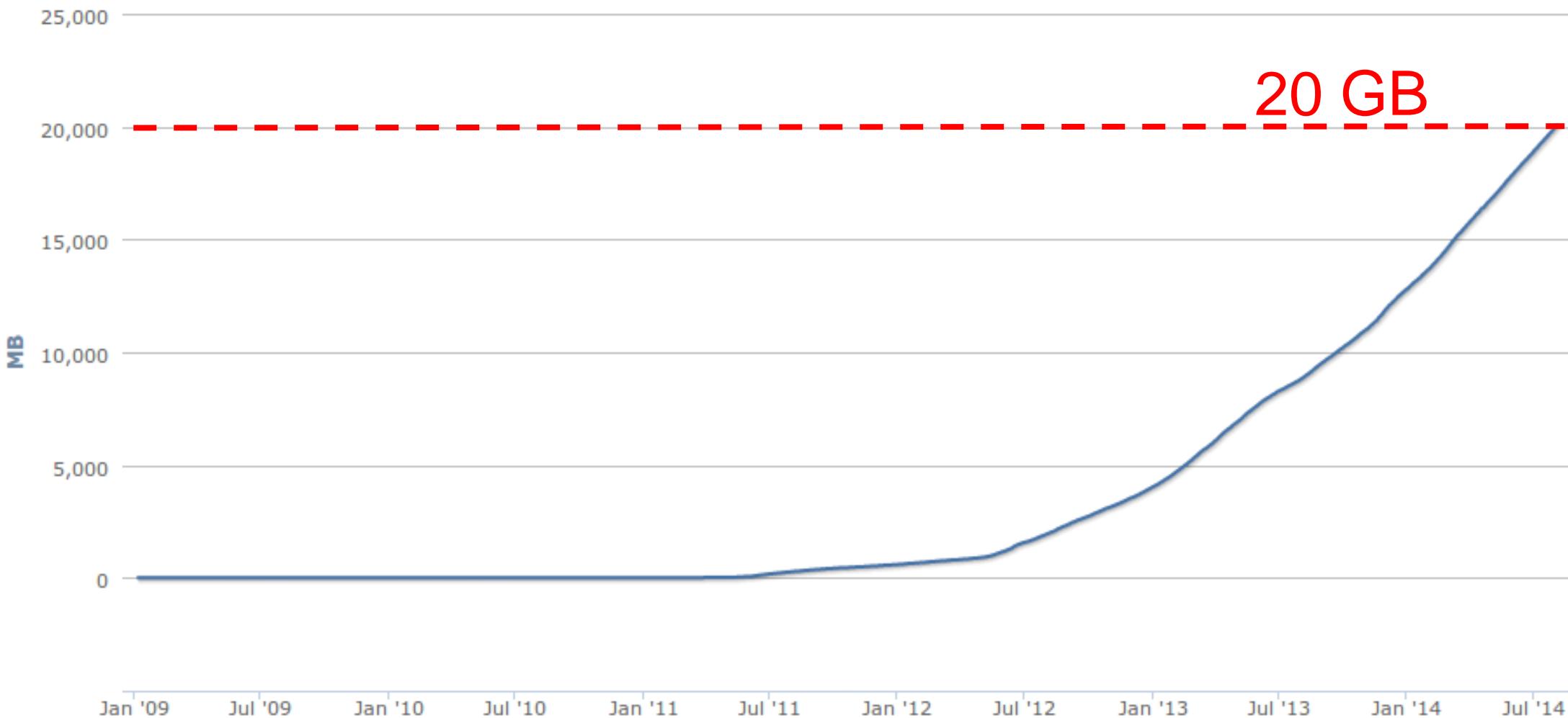
- Impossible to measure exactly
- Estimates-up to 1M IP addresses/month
- Only about 5-10k “full nodes”
  - Permanently connected
  - Fully-validate
- This number may be dropping!

# Fully-validating nodes

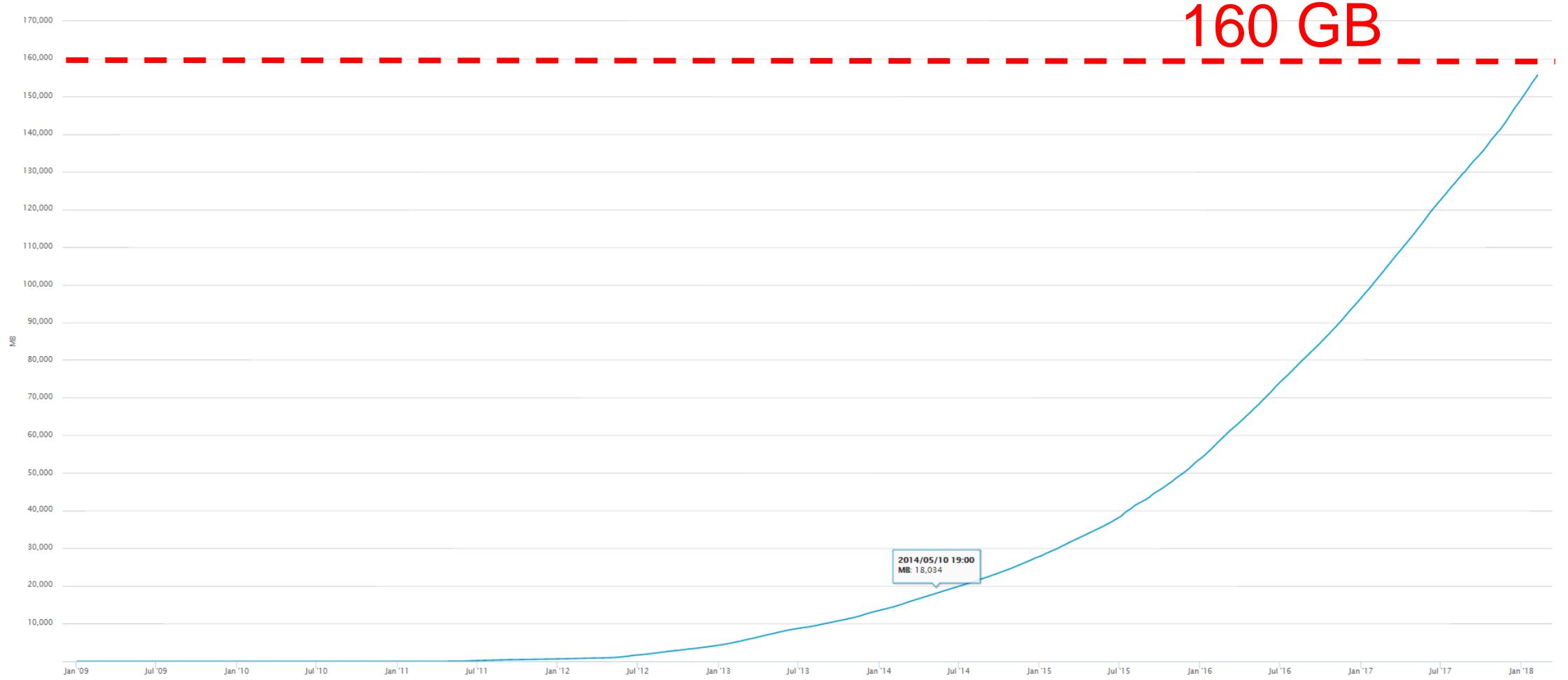
- Permanently connected
- Store entire block chain
- Hear and forward every node/transaction

# Storage costs (in 2014)

Blockchain Size  
Source: blockchain.info

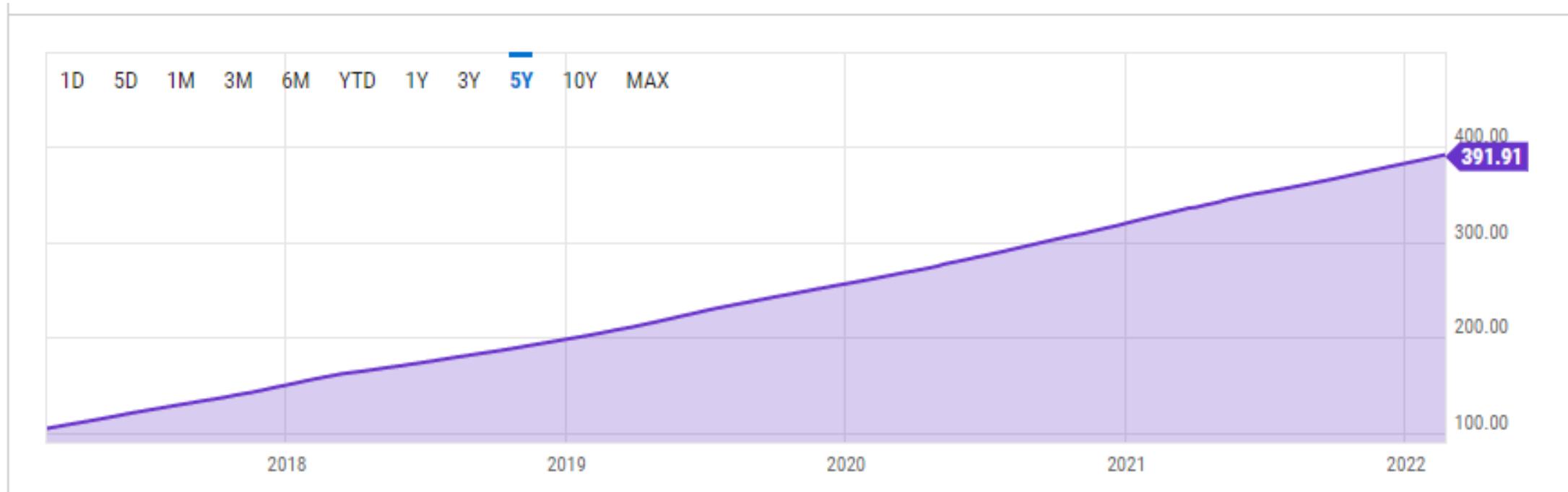


# Storage costs (in 2018)



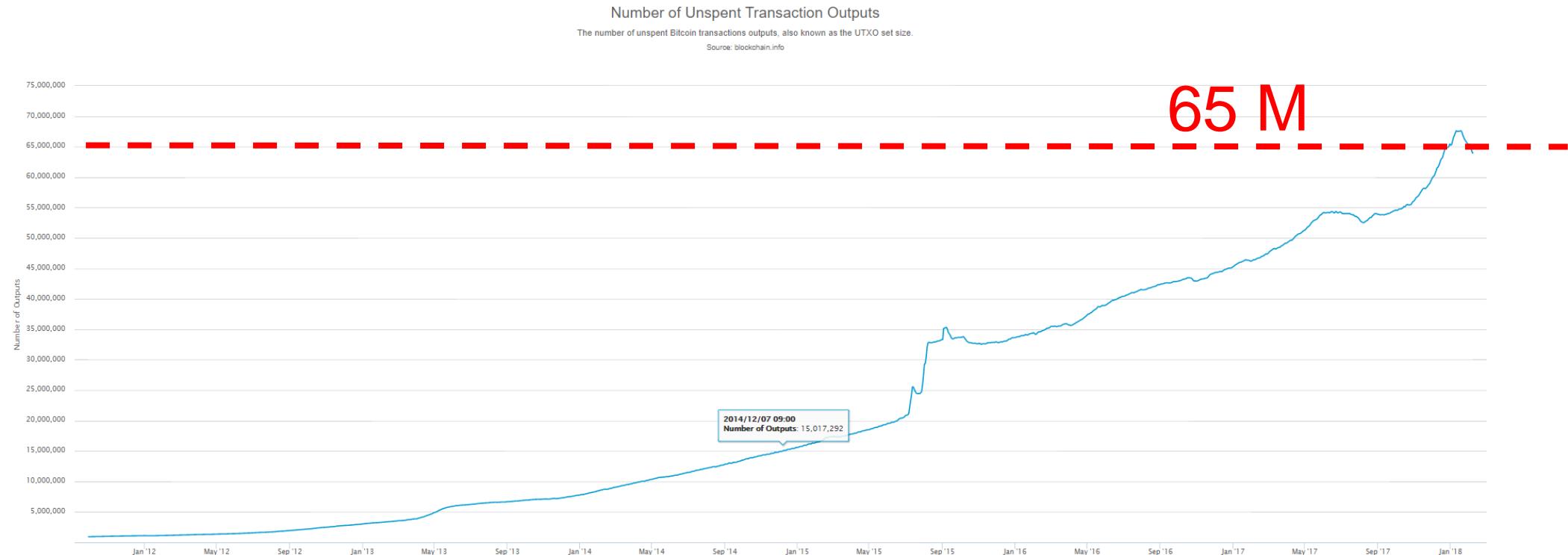
Source: blockchain.info

# Storage costs (in 2022)



# Tracking the UTXO set

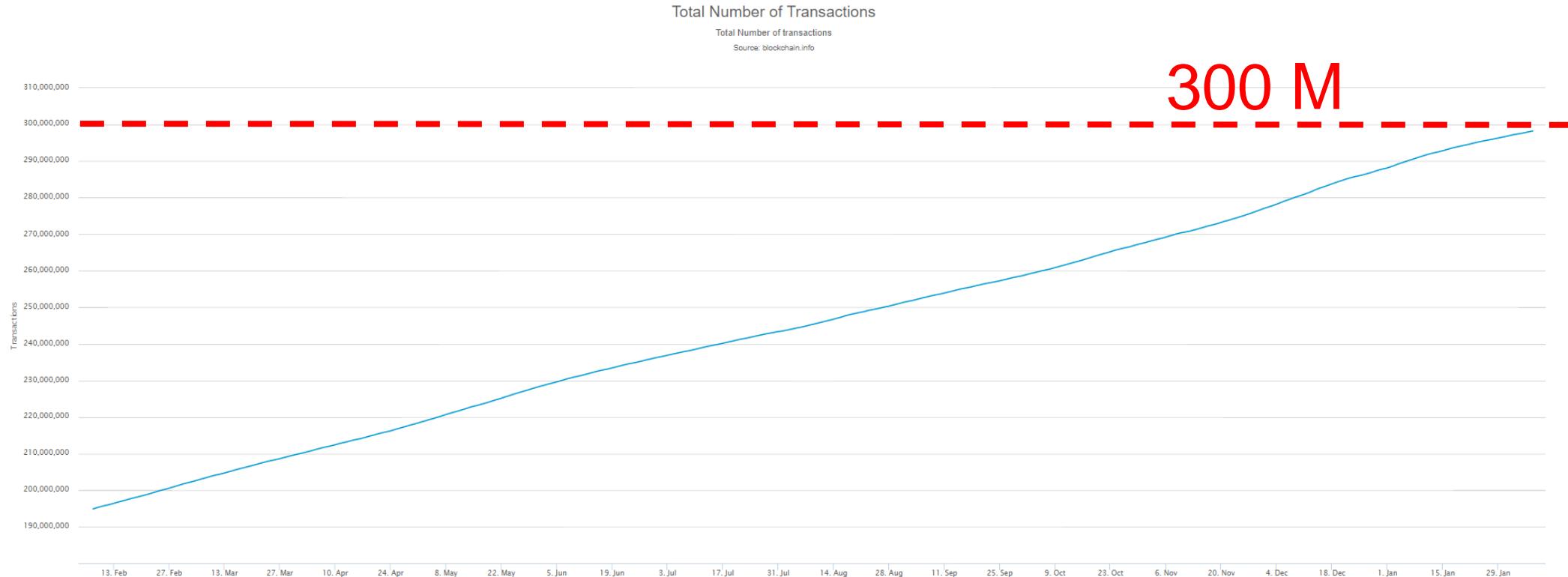
- Unspent Transaction Output
  - Should be stored in memory - everything else can be stored on disk, why?



Source: blockchain.info

# Tracking the UTXO set

- Currently ~65 M UTXOs
  - Out of 300 M transactions
- Can require several Gigabytes to store – can it fit in the RAM of a standard computer?



Source: [blockchain.info](https://blockchain.info)

# Thin/SPV clients (not fully-validating)

SPV – Simplified Payment Verification (e.g., Wallet nodes)

## Idea: Don't store everything

- Store block headers – verify the puzzle was solved correctly, but cannot verify every transaction in each block!
- Validate only those transactions that affect them → By requesting transactions as needed
  - To verify incoming payment
  - Trust fully-validating nodes

1000x cost savings! Requires only a few tens of Megabytes (compare to tens of Gigabytes needed for fully validating nodes)

# Software diversity

- About 90% of nodes run “Core Bitcoin” (C++)
  - Some are out of date versions
- Other implementations running successfully
  - BitcoinJ (Java)
  - Libbitcoin (C++)
  - btcd (Go)
- “Original Satoshi client”

# Limitations & Improvements

# Hard-coded limits in Bitcoin

- 10 min. average creation time per block
  - 1 M bytes in a block
  - 20,000 signature operations per block
  - 100 M *satoshis* per bitcoin
  - 23M total bitcoins maximum
  - 50,25,12.5... bitcoin mining reward
- 
- These affect economic balance of power too much to change now

# Throughput limits in Bitcoin

- 1 M bytes/block (10 min)
- >250 bytes/transaction
- 7 transactions/sec 😞

Compare to:

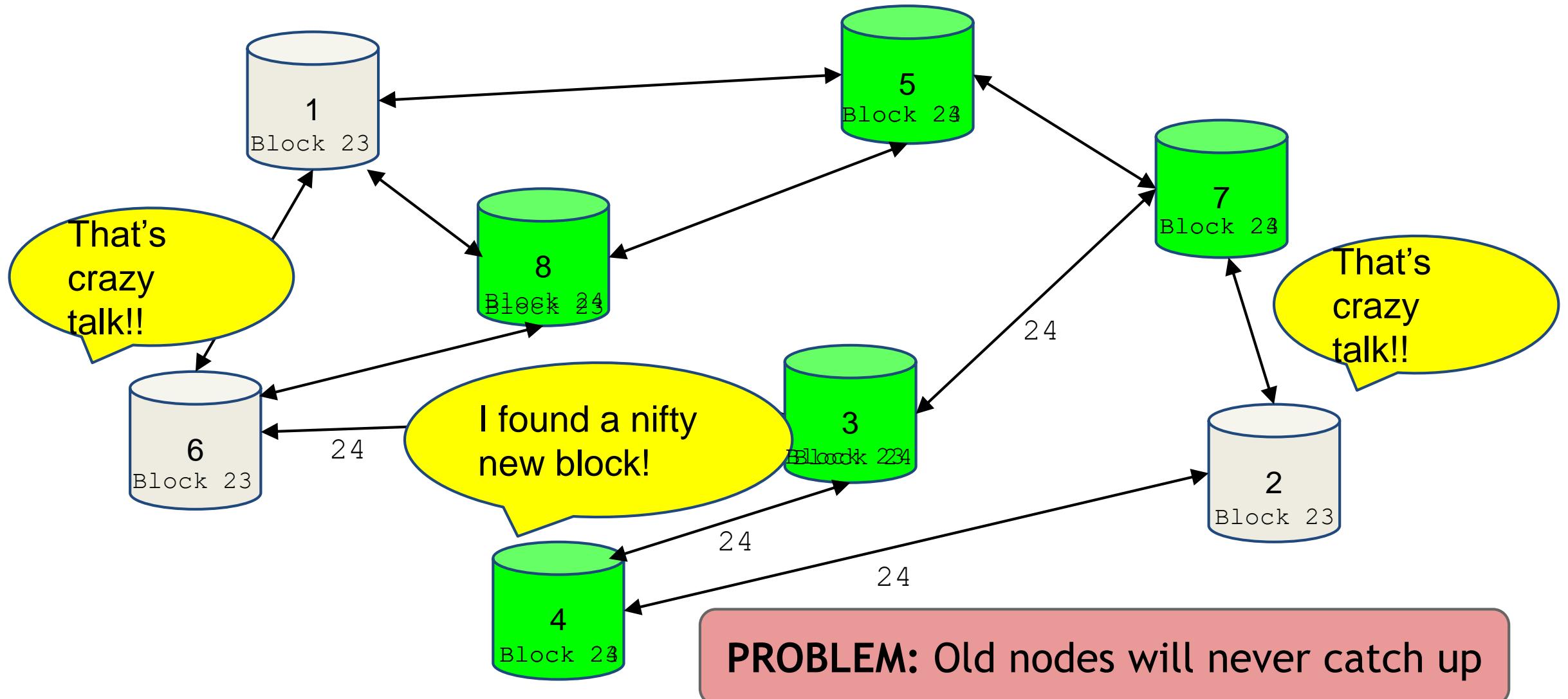
- VISA: 2,000-10,000 transactions/sec
- PayPal: 50-100 transaction/sec

# Cryptographic limits in Bitcoin

- Only 1 signature algorithm (ECDSA/P256)
- Hard-coded hash functions

Crypto primitives might break by 2040...

# “Hard-forking” changes to Bitcoin



# Soft forks

Observation: we can add new features which only  
*limit* the set of valid transactions

Need majority of nodes to enforce new rules

Old nodes will approve

RISK: Old nodes might mine now-invalid blocks

# Soft fork example: pay to script hash

<Redeem Script> 2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 CHECKMULTISIG

Old nodes      <Redeem Script> OP\_HASH160 <hash of redemption script> EQUAL

New Nodes      <Redeem Script> OP\_HASH160 <hash of redemption script> EQUAL  
                  2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 CHECKMULTISIG

} Two executions

Old nodes will just approve the hash, not run the embedded script

# Soft fork possibilities

- New signature schemes
- Extra per-block metadata
  - Shove in the coinbase parameter
  - Commit to UTXO tree in each block

# Hard forks

- New op codes
- Changes to size limits
- Changes to mining rate
- Many small bug fixes

Currently seem very unlikely to happen

Stay tuned for the lecture on altcoins!



**BITS** Pilani  
Pilani Campus

# Blockchain Technology (BITS F452)

Dr. Ashutosh Bhatia, Dr. Kamlesh Tiwari  
Department of Computer Science and Information Systems





# ***BITCOIN: Transaction***

Source: Bitcoin and Cryptocurrency Technologies Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, Steven Goldfeder

# Recap: Bitcoin consensus

Bitcoin consensus gives us:

- Append-only ledger
- Decentralized consensus protocol
- Miners to validate transactions

*Assuming a currency exists to motivate miners!*

*In this chapter we will see how such a currency can be engineered*

# An account-based ledger (*not* Bitcoin)

time

Create 25 coins and credit to Alice ASSERTED BY MINERS

Transfer 17 coins from Alice to Bob SIGNED(Alice)

Transfer 8 coins from Bob to Carol SIGNED(Bob)

Transfer 5 coins from Carol to Alice SIGNED(Carol)

Transfer 15 coins from Alice to David SIGNED(Alice)

..

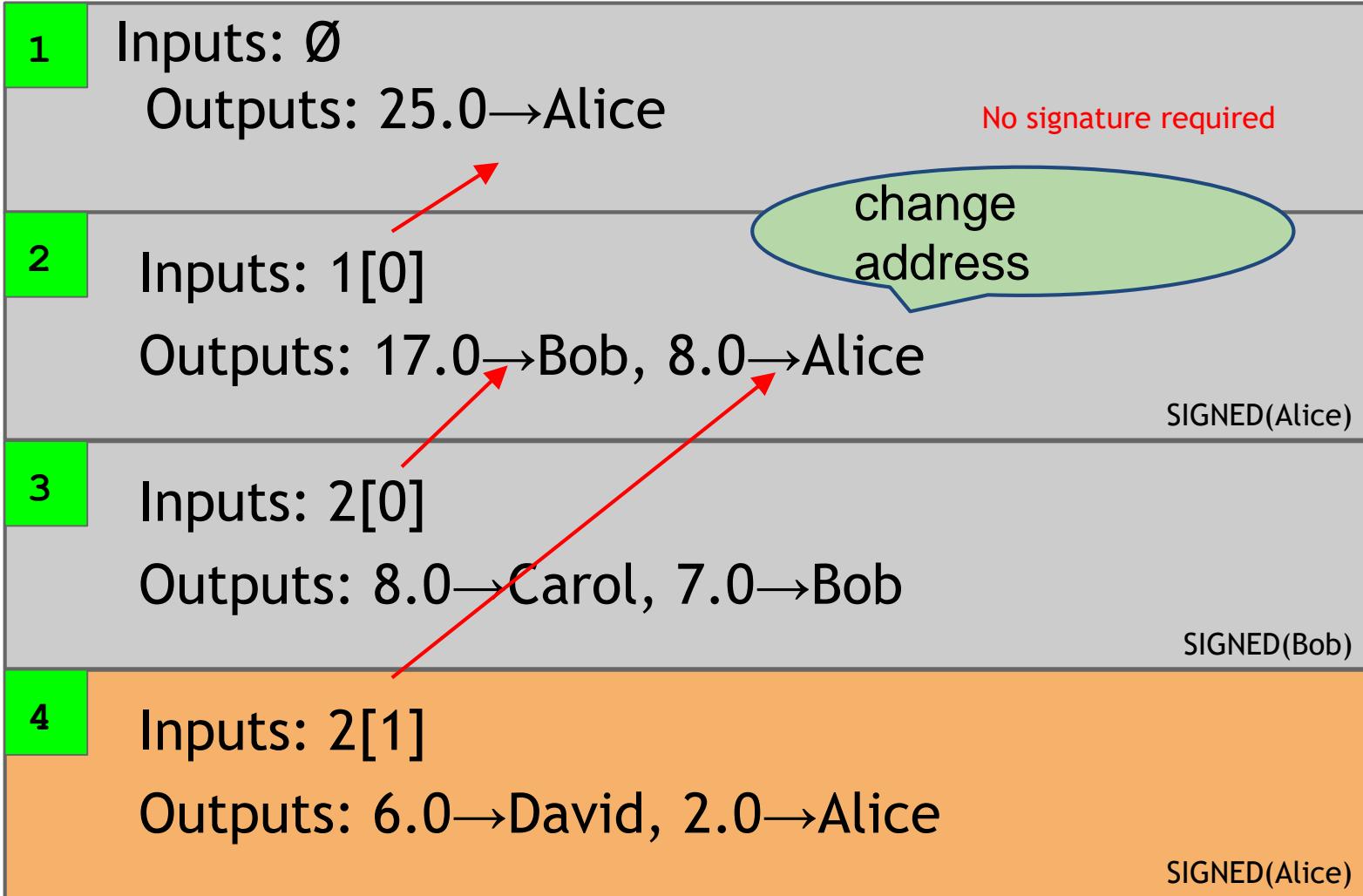
might need to  
scan  
backwards  
until genesis!

is this valid?

SIMPLIFICATION: only one transaction per block

# A transaction-based ledger (Bitcoin)

time ↓



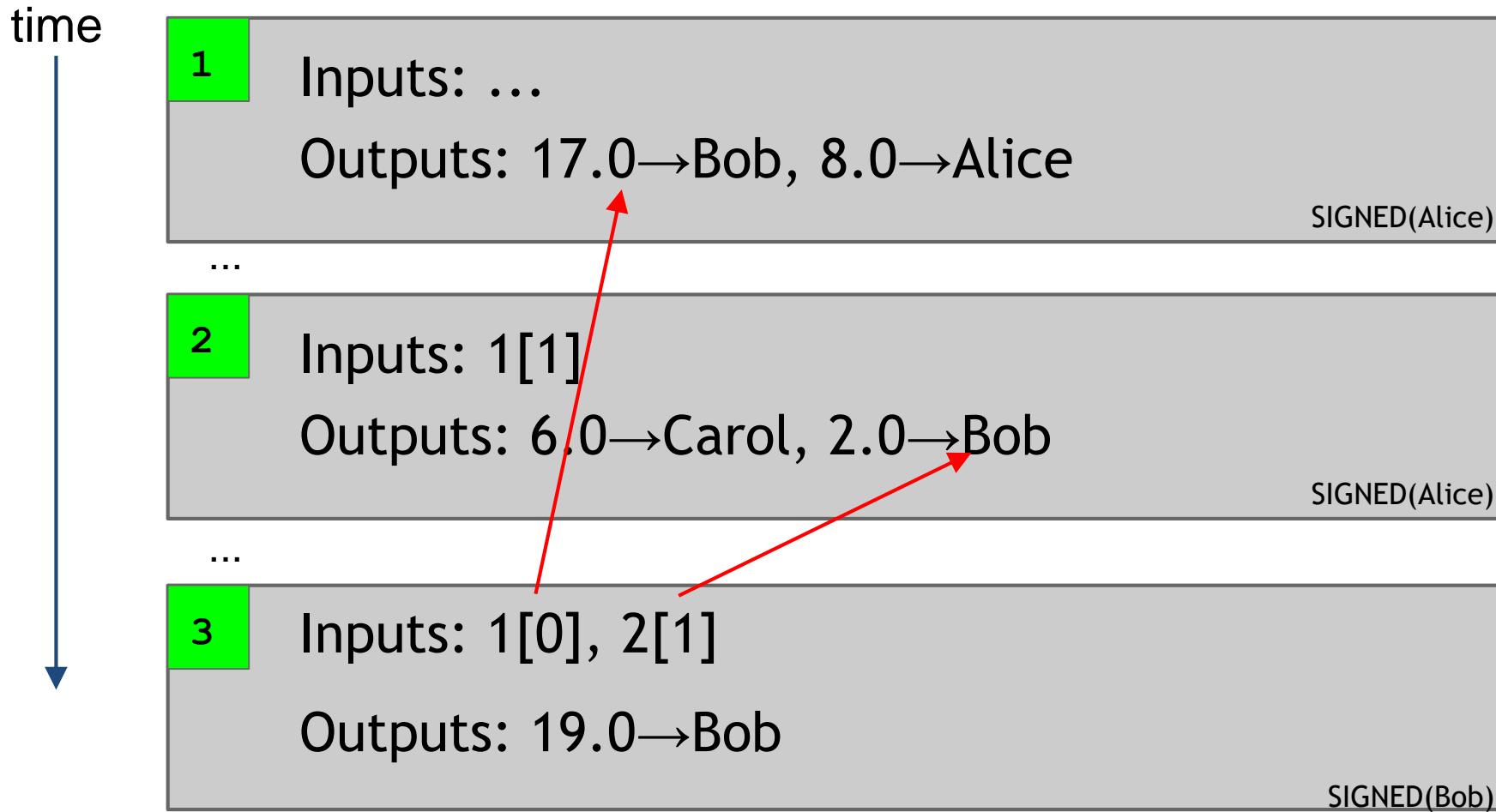
we implement this  
with hash pointers

finite scan to  
check for validity

is this valid?

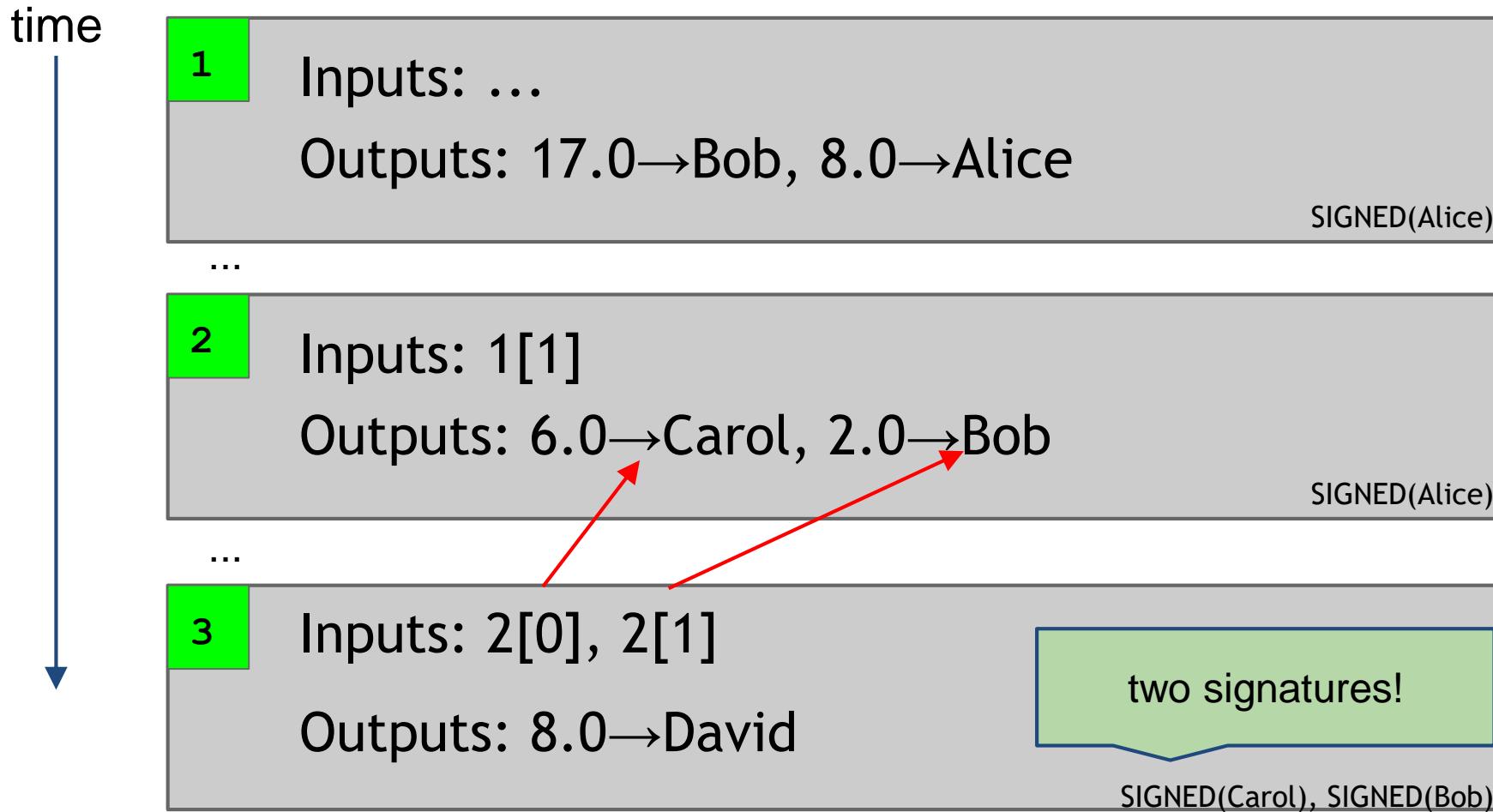
SIMPLIFICATION: only one transaction per block

# Merging value



SIMPLIFICATION: only one transaction per block

# Joint payments



SIMPLIFICATION: only one transaction per block

# The real deal: a Bitcoin transaction

```
{  
  "hash":"5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",  
  "ver":1,  
  "vin_sz":2,  
  "vout_sz":1,  
  "lock_time":0,  
  "size":404,  
  "in": [  
    {  
      "prev_out": {  
        "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",  
        "n": 0  
      },  
      "scriptSig": "30440..."  
    },  
    {  
      "prev_out": {  
        "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f81afc5c3f52f91ff6b34e",  
        "n": 0  
      },  
      "scriptSig": "3f3a4ce81...."  
    }  
  ],  
  "out": [  
    {  
      "value": "10.12287097",  
      "scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_  
      EQUALVERIFY OP_CHECKSIG"  
    }  
  ]  
}
```

# The real deal: transaction metadata

```
{  
transaction hash:  
  "hash":"5a42590...b8b6b",  
    "ver":1,  
    "vin_sz":2,  
    "vout_sz":1,  
    "lock_time":0,  
    "size":404,  
  ...  
}  
  
also serves as a unique ID  
more on this later...
```

# The real deal: transaction inputs

```
"in": [  
  {  
    "prev_out": {  
      "hash": "3be4...80260",  
      "n": 0  
    },  
    "scriptSig": "30440....3f3a4ce81"  
  },  
  ...  
],
```

previous transaction

signature

(more inputs)

# The real deal: transaction outputs

```
output value      "out": [ {  
recipient address??    "value": "10.12287097",  
                           "scriptPubKey": "OP_DUP OP_HASH160  
                           69e...3d42e OP_EQUALVERIFY OP_CHECKSIG"  
                           },  
                           ...  
                           ]  
(more outputs)           more on this  
                           soon...
```

**Sum of all output values less than or equal to sum of all input values!**

If sum of all output values less than sum of all input values, then difference goes to miner as a transaction fee

# Bitcoin scripts

*Bitcoin transaction validation is not based on a static pattern, but instead is achieved through the execution of a scripting language. This language allows for a nearly infinite variety of conditions to be expressed. This is how bitcoin gets the power of ‘programmable money’*

Source: Mastering Bitcoin

# Output “addresses” are really *scripts*

```
OP_DUP  
OP_HASH160  
69e02e18...  
OP_EQUALVERIFY OP_CHECKSIG
```

# Input “addresses” are *also* scripts



TO VERIFY: Concatenated script must execute completely with no errors

# Bitcoin scripting language (“Script”)

## Design goals

- Built for Bitcoin (inspired by Forth)
- Simple, compact
- Support for cryptography
- Stack-based (linear)
- Limits on time/memory
- No looping
  - **Result:** Bitcoin script is not Turing Complete  
i.e., cannot compute arbitrarily powerful functions
  - **Advantage:** No infinite looping problem!

I am not impressed

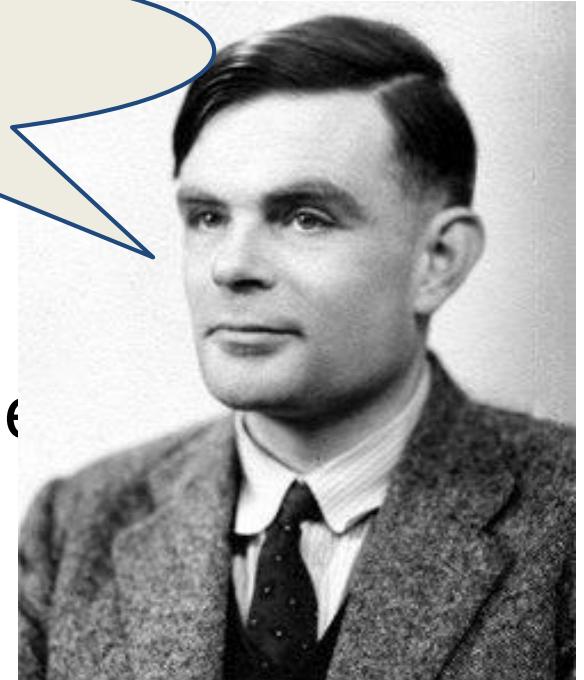


image via Jessie St. Amand

# **Bitcoin scripting language (“Script”)**

- 256 instructions (each represented by 1 byte)
  - 75 reserved, 15 disabled
  - Basic arithmetic, basic logic (“if” → “then”), throwing errors, returning early, crypto instructions (hash computations, signature verifications), etc.
- Only two possible outcomes of a Bitcoin script
  - Executes successfully with no errors → transaction is valid OR
  - Error while execution → transaction invalid and should not be accepted in the block chain

# Common script instructions

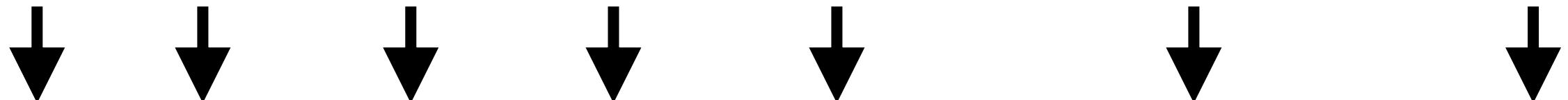
Name	Functions
<b>OP_DUP</b>	Duplicates top item on the stack
<b>OP_HASH160</b>	Hashes twice: first using SHA-256, then using RIPEMD-160
<b>OP_EQUALVERIFY</b>	Returns true if inputs are equal, false (marks transaction invalid) otherwise
<b>OP_CHECKSIG</b>	Checks that the input signature is valid using input public key for the hash of the current transaction
<b>OP_CHECKMULTISIG</b>	Checks that $t$ signatures on the transaction are valid from $t$ ( <i>out of</i> $n$ ) of the specified public keys

# **OP\_CHECKMULTISIG**

- Built-in support for joint signatures
- Specify  $n$  public keys
- Specify  $t$
- Verification requires  $t$  signatures are valid

# Bitcoin script execution example

<pubKeyHash?>
<pubKeyHash>
<pubKey>
true



```
<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash?> OP_EQUALVERIFY OP_CHECKSIG
```

# Bitcoin scripts in practice (as of 2014)

- Most nodes whitelist known scripts
- 99.9% are simple signature checks
- ~0.01% are MULTISIG More on this soon
- ~0.01% are Pay-to-Script-Hash
- Remainder are errors, proof-of-burn

# Proof-of-burn

nothing's going to redeem that 😞

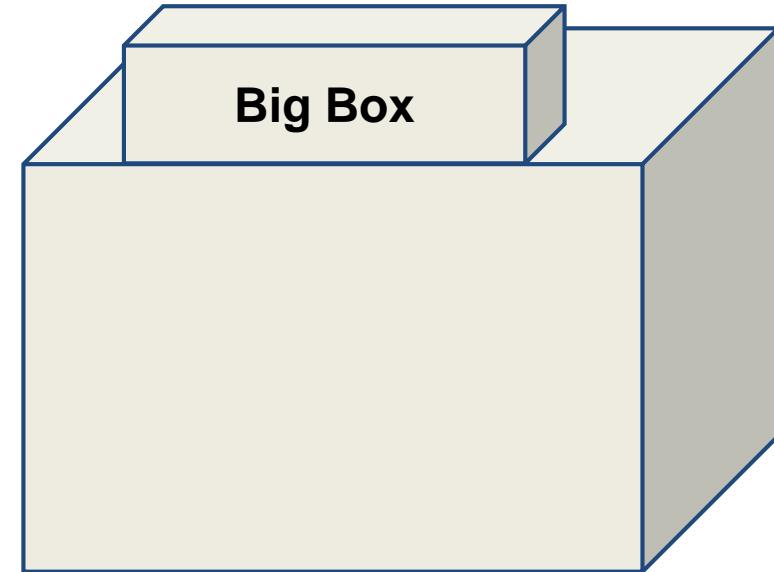
OP\_RETURN  
<arbitrary data>

# Should senders specify scripts?

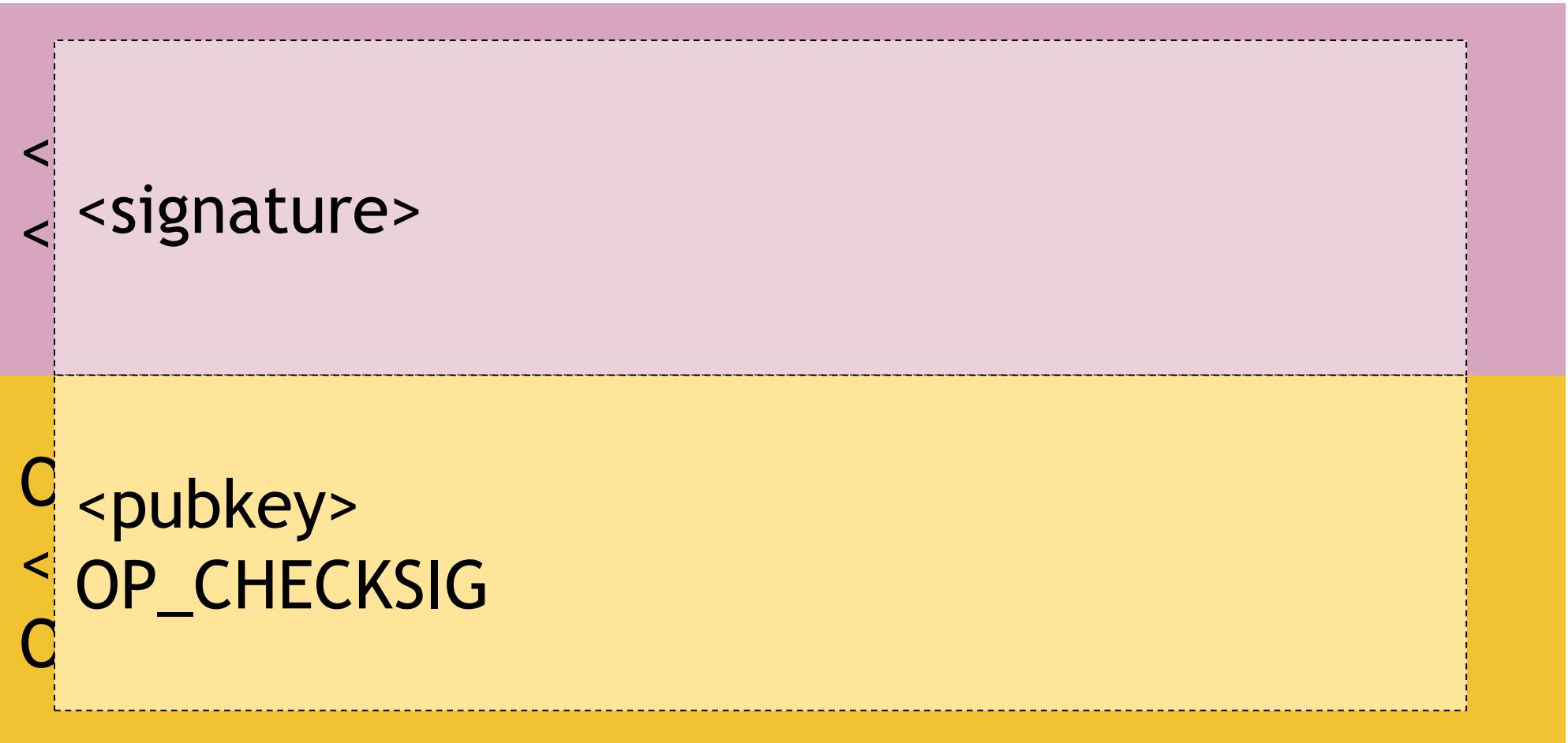


I'm ready to pay for my purchases!

Cool! Well we're using MULTISIG now, so include a script requiring 2 of our 3 account managers to approve. Don't get any of those details wrong. Thanks for shopping at Big Box!

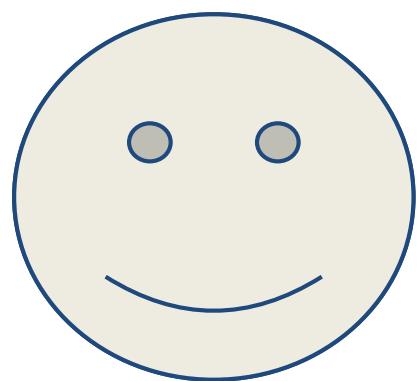


# Idea: use the hash of redemption script



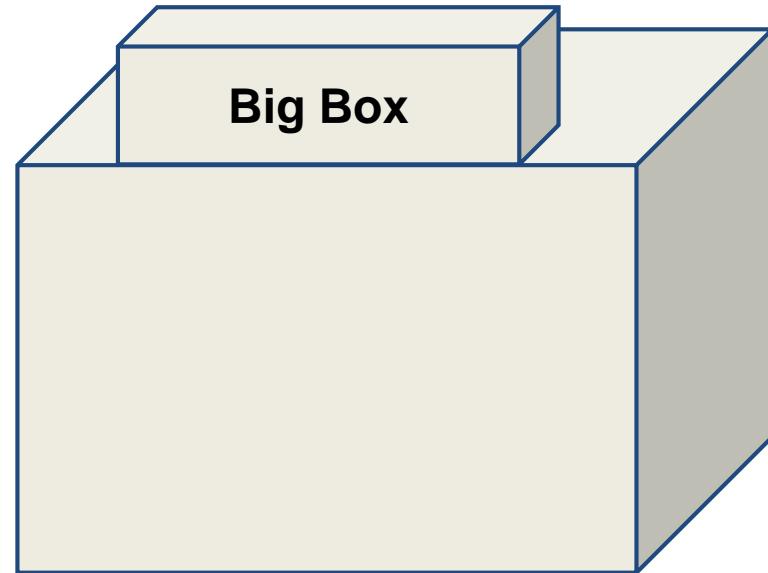
“Pay to Script Hash”

# Pay to script hash



I'm ready to pay for my purchases!

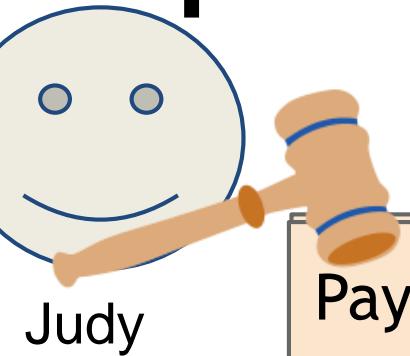
Great! Here's our address:  
0x3454



# Applications of Bitcoin scripts

# Example Escrow transactions

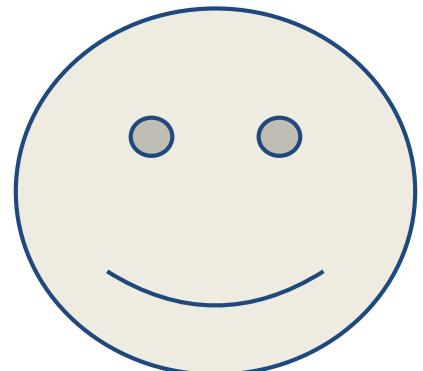
(disputed case)



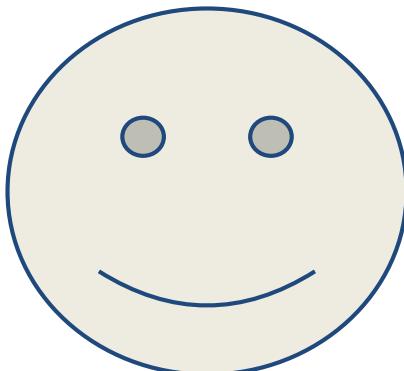
Judy

Pay x to Alice

SIGNED(ALICE, JUDY)



Alice



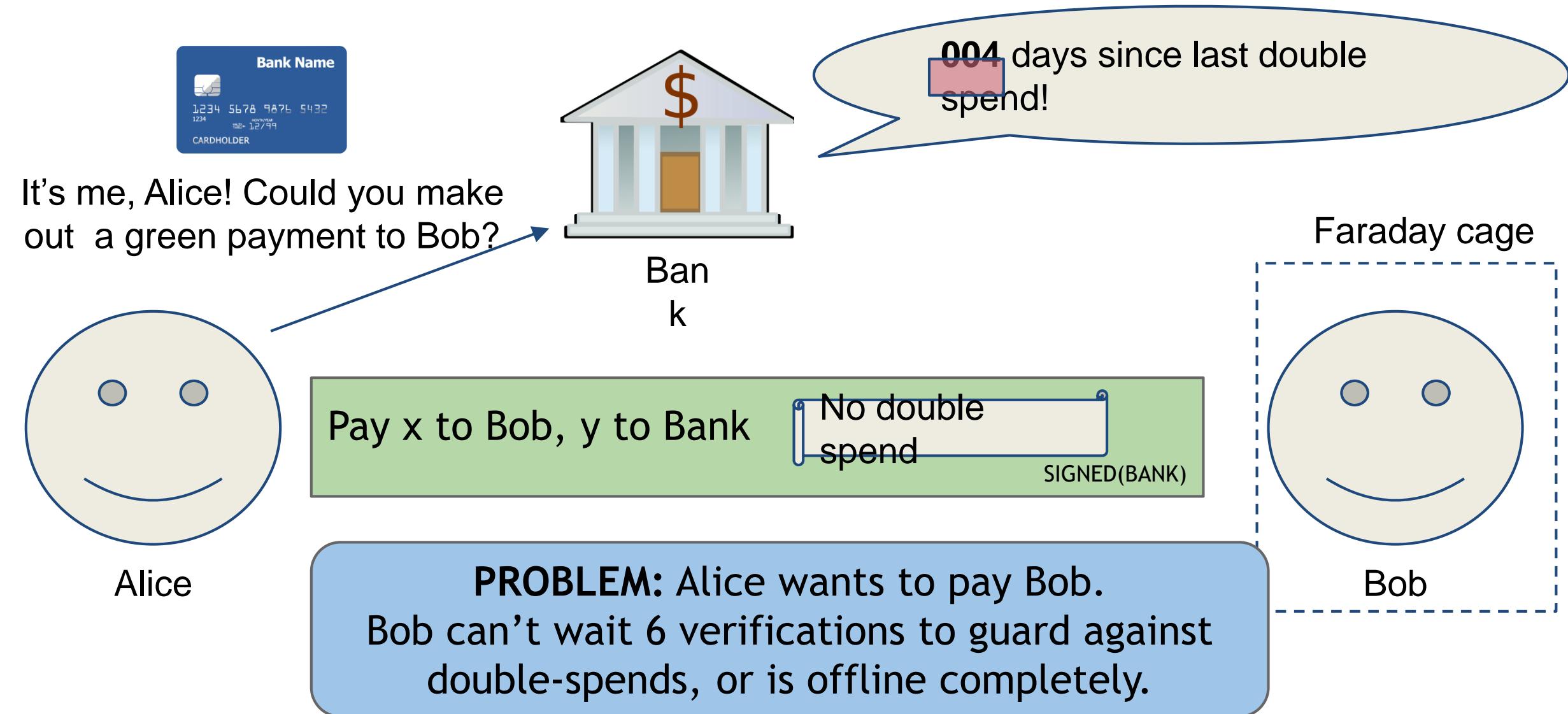
Bob

Pay x to 2-of-3 of Alice, Bob, Judy (MULTISIG)

SIGNED(ALICE)

Bob doesn't want to ship until after Alice pays.

# Example 2: Green addresses



# Example 3: Efficient micro-payments

all of these  
could be  
double-  
spends!

What if Bob never signs??

Input:  $x$ ; Pay 42 to Bob, 58 to Alice

SIGNED(ALICE) SIGNED(BOB)

...  
Alice demands a timed refund transaction before

Inputs:  $x$ ; Pay 100 to Alice, LOCK until time  $t$

SIGNED(ALICE) SIGNED(BOB)

I'm done!

Inputs:  $x$ ; Pay 03 to Bob, 97 to Alice

SIGNED(ALICE)

Input:  $x$ ; Pay 02 to Bob, 98 to Alice

SIGNED(ALICE)

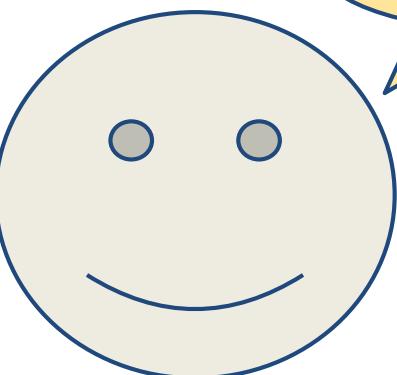
Input:  $x$ ; Pay 01 to Bob, 99 to Alice

SIGNED(ALICE)

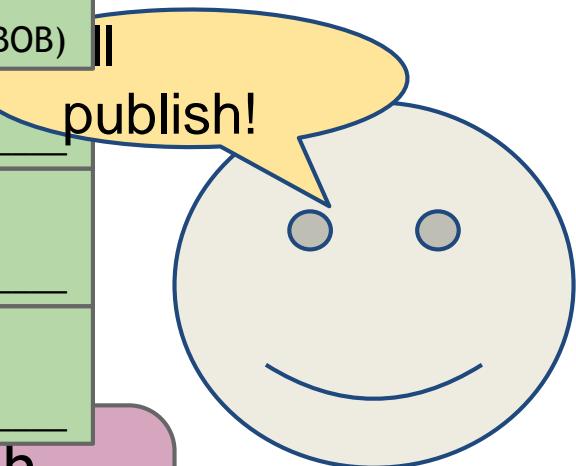
**PROBLEM:** Alice wants to pay Bob for each

Input:  $y$ ; Pay 100 to Bob/Alice (MULTISIG)

SIGNED(ALICE)



Alice



Bob

I'm done!

publish!

↓

↓

↓

↓

↓

↓