

# Che cos'è la blockchain?

La **blockchain**, letteralmente una catena (chain) di blocchi (block) è un **registro, condiviso e immutabile**

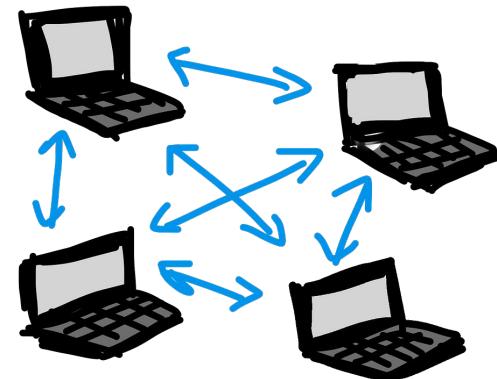
# Registro

- Un **registro** è un archivio nel quale vengono memorizzate informazioni in ordine temporale
- In una blockchain le informazioni sono memorizzate in **blocchi** di dimensioni prestabilite (es. 1 MB)
- Ogni blocco viene memorizzato nella blockchain con una data (timestamp)

No.	Date	Name	Amount		Signature	Secretary
			Debit	Credit		
1	Sept 28 1918	General Bill	1	1.00		
2	" 28 1918	Edith Bell	2	1.00		
3	" 28 1918	Josephine Bell	3	1.00		
4	28 1918	Catherine Courtney	27	1.00		
5	28 1918	Mary Courtney	28	1.00		
6	28 1918	Louis Courtney	29	1.00		
7	28 1918	Theodosius Courtney	30	1.00		
8	28 1918	Tyrone Courtney	31	1.00		
9	28 1918	Thomas Dennis	65	1.00		
10	28 1918	Philip Dennis	66	1.00		
11	28 1918	Conrad Greenwell	77	1.00		
12	28 1918	Albert Grunwell	78	1.00		
13	28 1918	Ottava Haworth	89	1.00		
14	28 1918	James Jones	103	1.00		
15	28 1918	Ernest Lewis	137	1.00		
16	28 1918	Linda Morris	138	1.00		
17	28 1918	Bonnie Morris	150	1.00		
18	28 1918	Gerald Morris	151	1.00		
19	28 1918	Stanley Murphy	162	1.00		
20	28 1918	James Spence	236	1.00		
21	28 1918	Henry Spence	239	1.00		
22	28 1918	Ella Sonneborn	247	1.00		
23	28 1918	Myrtle Steele	283	1.00		
24	7 1918	Estherine Somers	257	1.00		
25	7 1918	Melissa Brown	15	1.00		

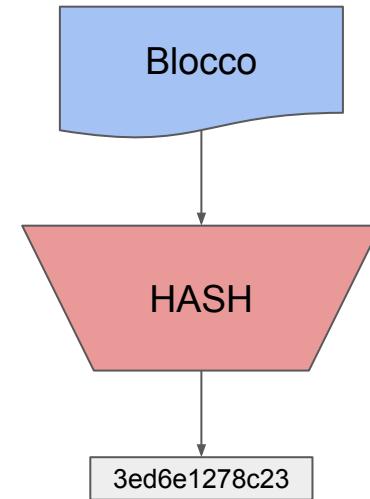
# Condiviso

- Questo registro è **condiviso** pubblicamente tra tutti i nodi della rete blockchain
- Ogni **nodo** detiene la **stessa copia** del registro
- La rete dei nodi è di tipo **peer-to-peer**

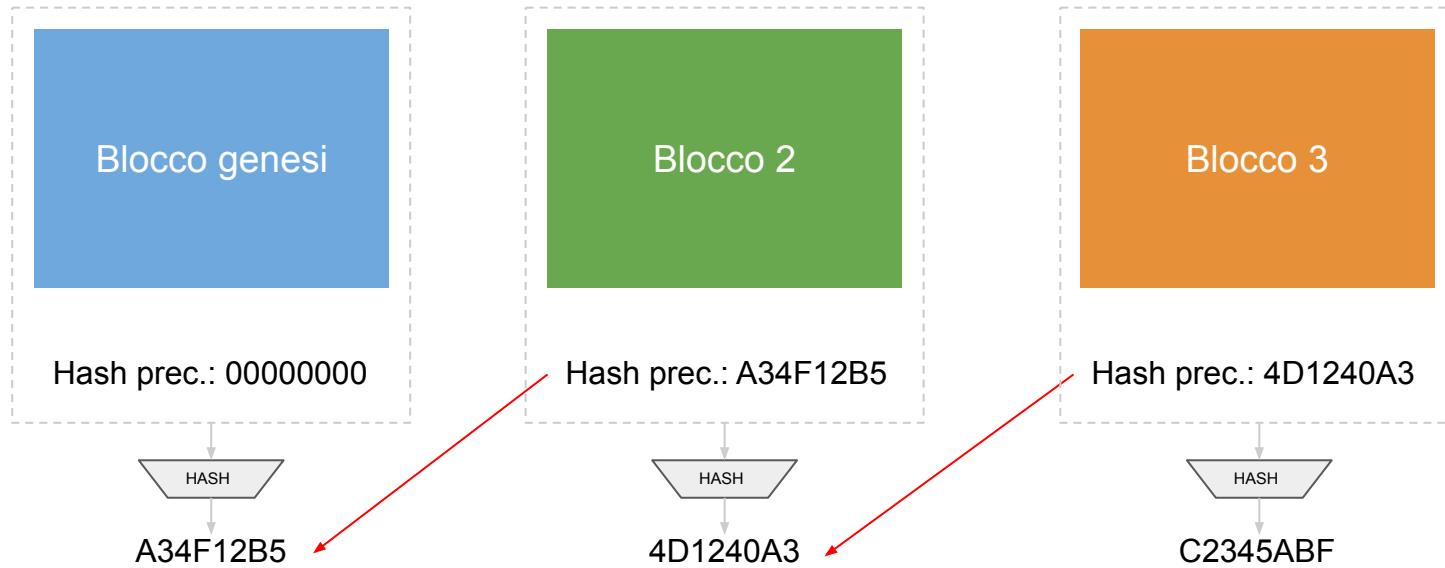


# Immutabilità

- Questo registro condiviso è **immutabile**, una volta che un blocco di dati viene memorizzato, non può essere più modificato
- I blocchi sono collegati tra di loro attraverso delle **funzioni hash**



# Catena di blocchi



# Origini della blockchain

- L'idea della **blockchain** nasce nel 2008 con l'articolo [Bitcoin: A Peer-to-Peer Electronic Cash System](#) di Satoshi Nakamoto (pseudonimo)
- Nakamoto utilizza il termine “block chain” per indicare il registro nel quale vengono memorizzate le transazioni di bitcoin
- L'articolo di Nakamoto è rivoluzionario perché introduce il concetto di **Proof Of Work (PoW)** per risolvere il problema del ***double spending*** su una rete decentralizzata

# Prima di Bitcoin

- L'idea di creare una catena (chain) di informazioni collegate tramite hash non è stata ideata da Nakamoto
- L'idea originale appare per la prima volta nell'articolo [How to Time-Stamp a Digital Document](#) di Stuart Haber e Scott Stornetta del 1991
- Erano state proposte diverse monete elettroniche prima di Bitcoin come [DigiCash](#) (1982), [B-Money](#) (1998), [Bit gold](#) (1998)

# Cyberpunk

- Movimento nato agli inizi degli anni '80 per promuovere l'utilizzo della crittografia come strumento di liberazione, per la protezione della privacy
- Mailing list cyberspace nata nel 1992 per opera di Eric Hughes, Timothy C. May e John Gilmore
- *"Privacy is necessary for an open society in the electronic age. ... We cannot expect governments, corporations, or other large, faceless organizations to grant us privacy ... We must defend our own privacy if we expect to have any. ... Cypherpunks write code. We know that someone has to write software to defend privacy, and ... we're going to write it."*

A Cypherpunk's Manifesto, Eric Hughes, 1993

# bitcoin

- **bitcoin** è la prima criptovaluta ideata nel 2008
- Un bitcoin (฿) vale circa € 19'700\*
- Market cap € 378\* miliardi
- Ad oggi è la criptovaluta più importante; il mercato segue sostanzialmente il comportamento di bitcoin
- Sito ufficiale del progetto: [bitcoin.org](https://www.bitcoin.org)



\* Dato del 15 Settembre 2022

# Come funziona Bitcoin

- In **Bitcoin**, come in tutte le criptovalute, ogni persona può avere uno o più indirizzi
- Questi indirizzi sono **anonimi**, non contengono informazioni sulla persona
- È possibile trasferire bitcoin da un indirizzo a un altro (**transazione**)
- Le transazioni sono pubbliche, chiunque può verificare che un indirizzo A ha trasferito x bitcoin ad un indirizzo B in una certa data (per questo motivo si dice che Bitcoin garantisce un semi-anonimato)

# Come funziona Bitcoin (2)

- I bitcoin vengono generati dai **miner**, ossia i nodi della rete che convalidano la creazione dei blocchi (ogni 10 minuti)
- Il protocollo Bitcoin è progettato in modo tale che nuovi bitcoin sono creati ad una velocità fissa
- E' previsto un numero massimo di bitcoin in circolazione, **21 milioni** (attualmente 19<sup>\*</sup> milioni)

\* Dato del 15 Settembre 2022

# Hash

- La blockchain è una **catena di dati** collegate tra di loro tramite l'utilizzo di una funzione **hash**
- Un hash è una **funzione non invertibile** che mappa una stringa di lunghezza arbitraria in una stringa di lunghezza predefinita (es. 32 bytes)
- Esempio: **SHA256**("Hello World") =  
a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9ad9f146e
- Partendo dal risultato della funzione hash deve essere computazionalmente impossibile determinare la stringa originale

# Proprietà di una funzione hash

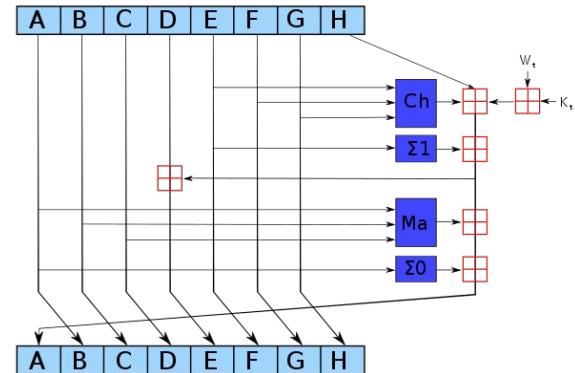
Una funzione hash deve rispettare le seguenti proprietà:

- È facile calcolare l'hash di un messaggio
- È computazionalmente impossibile calcolare il messaggio che ha generato un dato hash (**non invertibilità**)
- È computazionalmente impossibile modificare un messaggio senza modificare il relativo hash (**resistenza debole alle collisioni**)
- Dato un messaggio è computazionalmente impossibile trovarne un altro con lo stesso hash (**resistenza forte alle collisioni**)

# SHA256

- Il messaggio viene suddiviso in blocchi di **512 bit**
- Questi blocchi vengono iterati con una funzione di compressione su un insieme di 8 valori da 32 bit (A, B, C, D, E, F, G, H)
- Il risultato è la concatenazione di questi blocchi ( $8 \times 32 = 256$  bit)
- Bitcoin utilizza **SHA256<sup>2</sup>** ossia SHA256(SHA256(x))

Funzione di compressione SHA-2

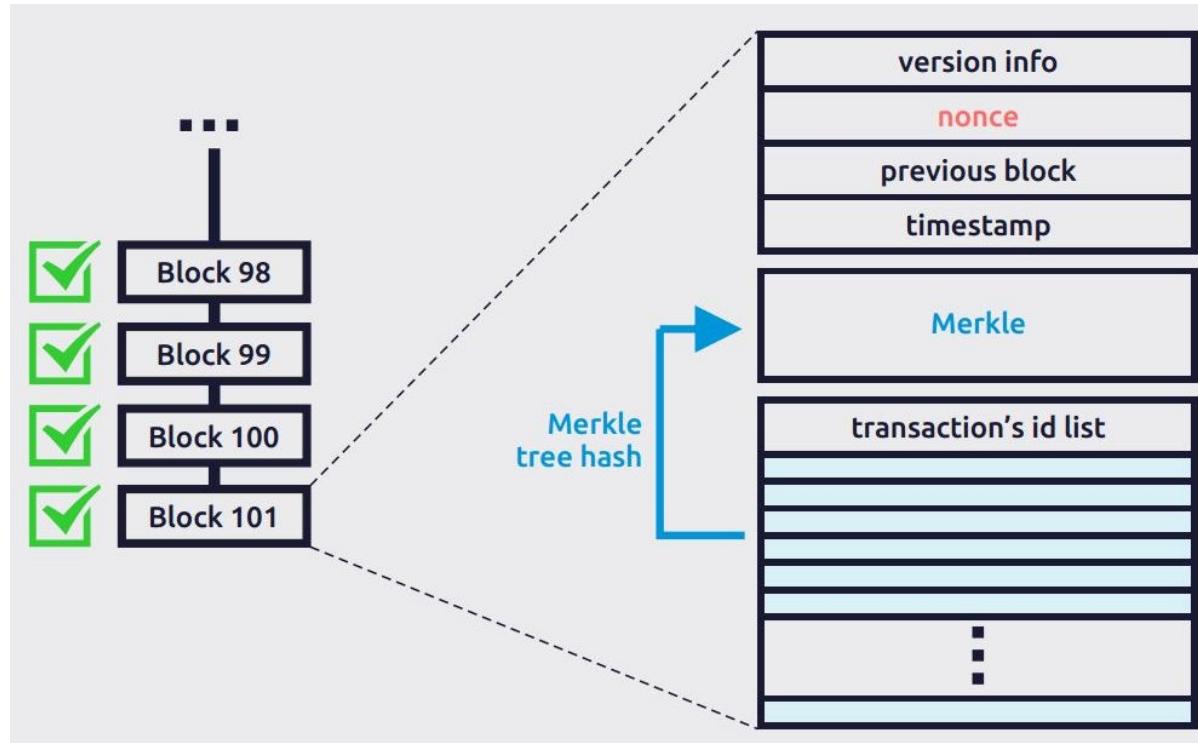


$$\begin{aligned}
 \text{Ch}(E, F, G) &= (E \wedge F) \oplus (\neg E \wedge G) \\
 \text{Ma}(A, B, C) &= (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C) \\
 \Sigma_0(A) &= (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22) \\
 \Sigma_1(E) &= (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)
 \end{aligned}$$

# Proof of Work

- Modificare una blockchain deve essere un'operazione computazionalmente (molto) difficile
- Il **Proof of Work** (PoW) è un'operazione che richiede diverso tempo computazionale (es. un puzzle matematico)
- Il campo **nonce** in un blocco è un esempio di **PoW**, ossia è il valore che deve essere concatenato all'input per ottenere un hash < target
- Bitcoin utilizza l'algoritmo [Hashcash](#)

# Nonce



**Nonce:** è il valore che concatenato alla stringa di partenza da come risultato un hash < target

# Esempio di PoW

- Stringa in input “Hello, world!”
- Trovare il valore di **nonce** tale che **SHA256**(“Hello, world!” + **nonce**) <  $2^{240}$

```
nonce=-1
while (hash >= 2^240)
    nonce++
    hash = SHA256 ("Hello, world!" + nonce)
```

# Esempio di PoW (2)

```
"Hello, world!0" => 1312af178c253f84028d480a6adc1e25e81caa44c749ec81976192e2ec934c64 = 2^252.253458683
"Hello, world!1" => e9afc424b79e4f6ab42d99c81156d3a17228d6e1eef4139be78e948a9332a7d8 = 2^255.868431117
"Hello, world!2" => ae37343a357a8297591625e7134cbea22f5928be8ca2a32aa475cf05fd4266b7 = 2^255.444730341
...
"Hello, world!4248" => 6e110d98b388e77e9c6f042ac6b497cec46660deef75a55ebc7cfdf65cc0b965 = 2^254.782233115
"Hello, world!4249" => c004190b822f1669cac8dc37e761cb73652e7832fb814565702245cf26ebb9e6 = 2^255.585082774
"Hello, world!4250" => 0000c3af42fc31103f1fdc0151fa747ff87349a4714df7cc52ea464e12dcd4e9 = 2^239.61238653
```

- nonce = **4250**
- Il target in questo esempio è  $2^{240}$
- Bitcoin resetta periodicamente il livello di difficoltà, ogni 2016 blocchi (circa 2 settimane), per tenere il tempo di creazione di un blocco ogni 10 minuti

# Mining

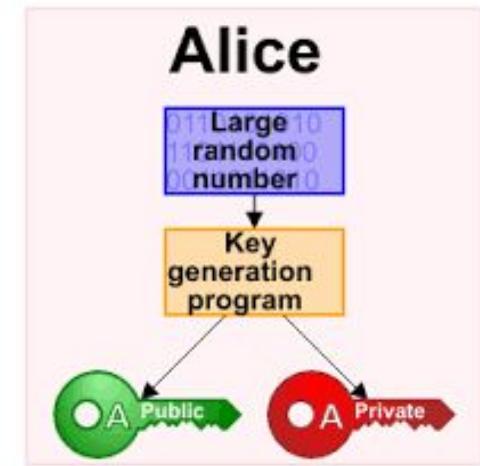
- I **miner** sono i nodi che risolvono i PoW
- Ogni volta che viene risolto un puzzle il nodo viene ricompensato (es. in bitcoin)
- Costo del mining (hardware + energia elettrica)
- Esempio:
  - Hardware: BITMAIN Antminer S17e (64Th), potenza di 2880W, costo ≈ \$1000
  - Costo energia: 0.1 USD/kWH
  - Ricavo: \$ 17.06 USD/giorno



Fonte: Nicehash, profitability calculator

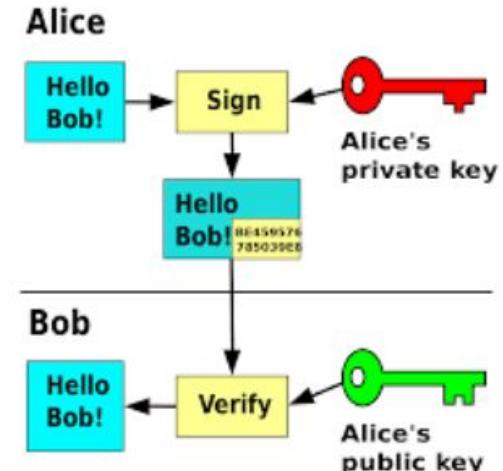
# Crittografia a chiave pubblica

- Per garantire l'autenticità e il non ripudio dei dati memorizzati in una blockchain si utilizza la **crittografia a chiave pubblica**
- Ogni utente genera una coppia di chiavi: **privata** e **pubblica**
- La crittografia a chiave pubblica può essere utilizzata sia per proteggere (**cifrare**) che per autenticare informazioni (**firmare**)



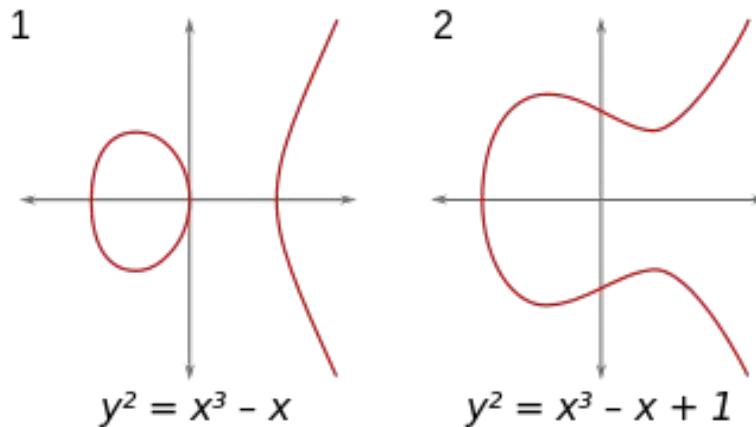
# Firma

- La **chiave privata** viene utilizzata per **firmare** un dato
- La **chiave pubblica** viene utilizzata per **verificare** la firma
- In Bitcoin, un indirizzo è calcolato con l'hash della chiave pubblica, in particolare:  
*ripemd160(SHA256(Public-key))*



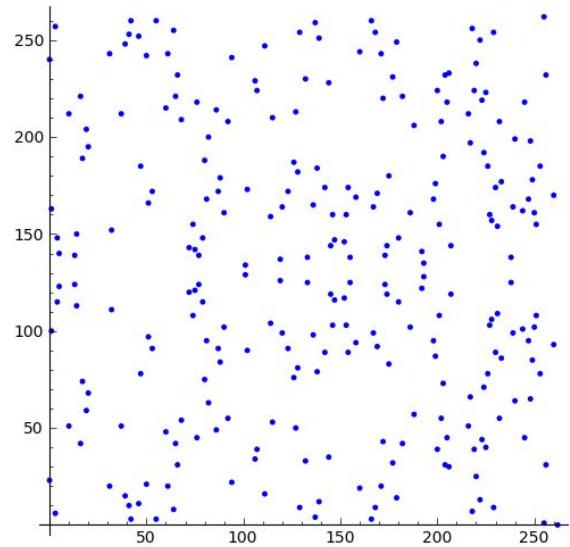
# Curva ellittica

- Una curva ellittica è una funzione matematica particolarmente utilizzata in crittografia
- Una curva ellittica è la funzione  $y^2 = x^3 + ax + b$



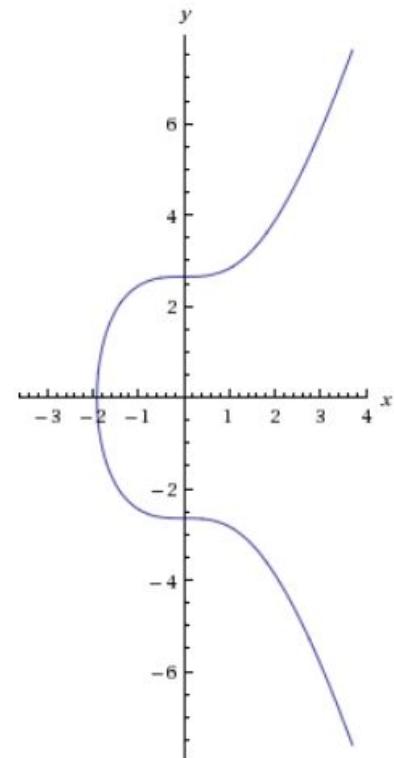
# Crittografia ellittica

- La crittografia ellittica utilizza le curve ellittiche su un campo  $\mathbb{Z}_n$
- Nelle tecnologie blockchain l'algoritmo che viene utilizzato per firmare le transazioni è l'ECDSA, Elliptic Curve Digital Signature Algorithm

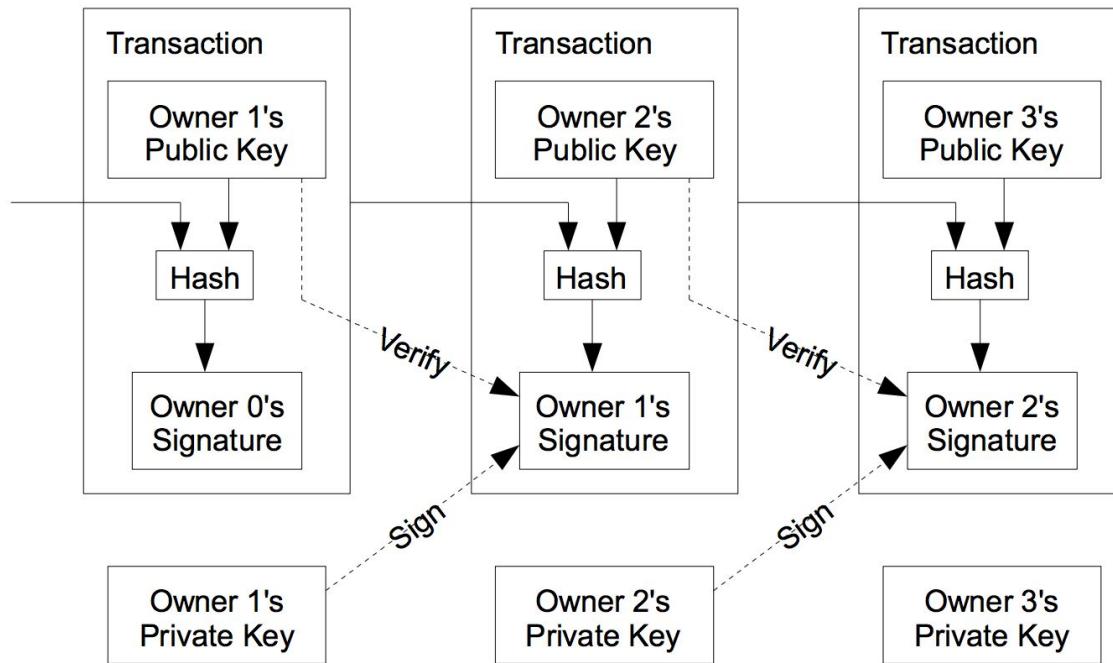


# Secp256k1

- Una delle curve più utilizzate nelle tecnologie bLockchain (es. Bitcoin) è la **Secp256k1**, dove  $a=0$  e  $b=7$ , ossia la curva  $y^2 = x^3 + 7$
- Bitcoin utilizza  $\mathbb{Z}$  definita sul campo  $\mathbb{Z}_{2^{256}-2^{32}-977}$ , nel quale le coordinate  $x$  e  $y$  sono interi a 256-bit modulo un primo molto grande



# Firma delle transazioni



# Generazione di un indirizzo Bitcoin

- Dalla chiave privata, viene generata una chiave pubblica utilizzando la **crittografia ellittica** (Secp256k1)
- Viene calcolato l'hash **SHA-256** della chiave pubblica e successivamente l'hash **RIPEMD-160** (risultato una stringa di 160 bit):  
 $\text{RIPEMD160}(\text{SHA-256}(\text{chiave pubblica}))$
- Infine, il risultato è codificato in **Base58Check** (numeri senza lo zero e lettere maiuscole e minuscole senza i caratteri O, I, e l) con l'aggiunta del checksum
- Esempio di indirizzo Bitcoin: **16JrGhLx5bcBSA34kew9V6Mufa4aXhFe9X**

# Esercizio

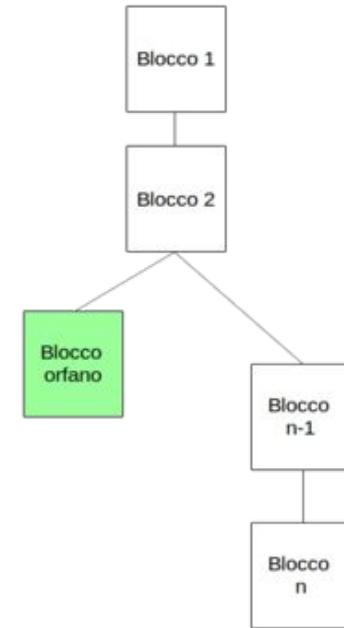
- Scrivere un programma in qualsiasi linguaggio che sia in grado di utilizzare la crittografia a curva ellittica e nello specifico l'algoritmo **ECDSA** con la curva **Secp256k1**
- Provare a generare una coppia di chiavi: privata e pubblica. Da quest'ultima provare a costruire un indirizzo Bitcoin **P2PKH** seguendo l'articolo [How to Generate a Bitcoin Address — Step by Step](#)
- Suggerimento: [web3j](#) (Java), [Secp256k1.Net](#) (.NET)

# Double spending

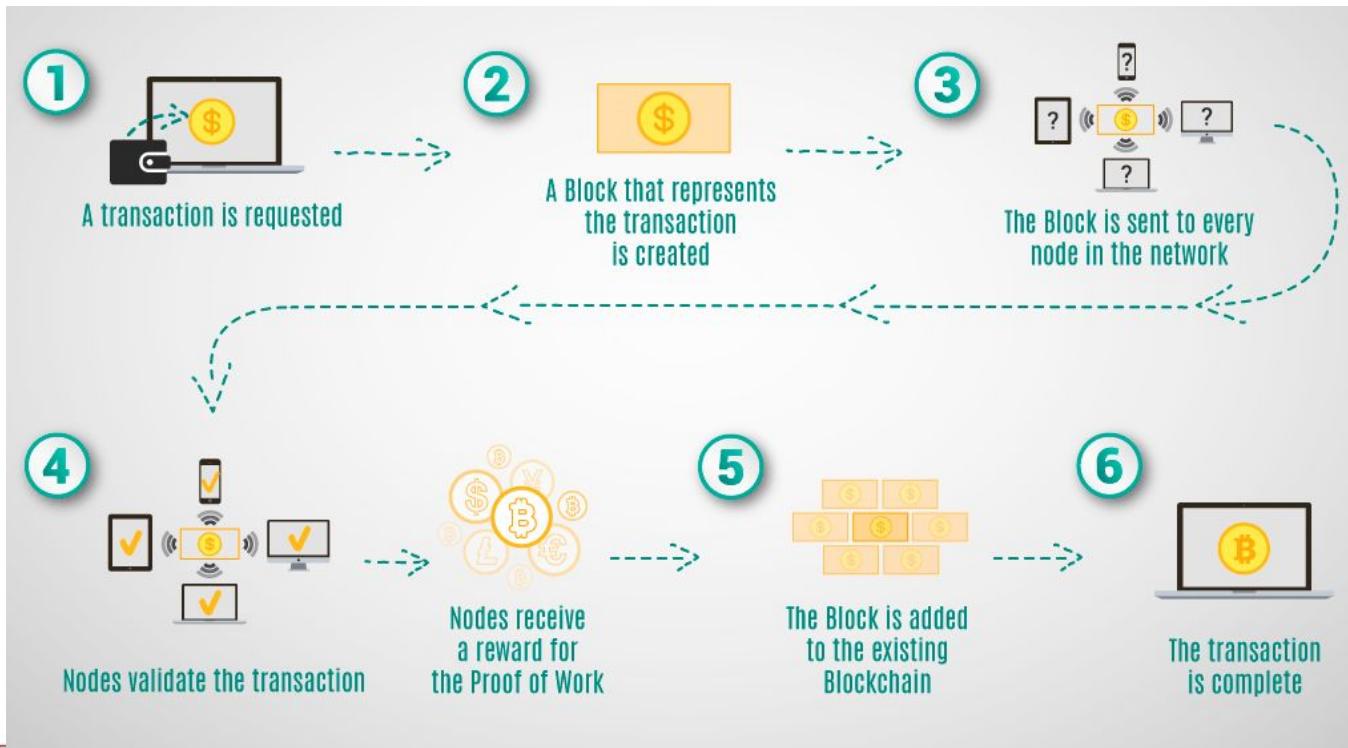
- Utilizzando una moneta elettronica bisogna risolvere il problema del **double spending**
- Dal momento che nel digitale è molto facile copiare una transazione, come faccio ad impedire che A invii a B x bitcoin e contemporaneamente invii gli stessi x bitcoin a C?
- Bitcoin e le altre criptovalute basate su Blockchain utilizzano un **meccanismo di validazione** per prevenire il double spending

# Validazione dei blocchi

- Per poter accettare un blocco è necessaria la validazione del PoW
- I blocchi vengono inseriti in sequenza
- La sequenza più lunga è quella che vince
- In caso di pari lunghezza viene effettuata una scelta casuale
- Se la maggioranza dei nodi (oltre il 50%) non è malevola la sicurezza della blockchain è garantita



# Esempio di transazione Bitcoin



# Bibliografia

- Demers et al. (1987), [Epidemic algorithms for replicated database maintenance](#), PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing
- Satoshi Nakamoto, [Bitcoin: A Peer-to-Peer Electronic Cash System](#), Bitcoin whitepaper
- Narayanan et al. (2016), [Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction](#), Princeton University Press
- Andreas M. Antonopoulos (2017), [Mastering Bitcoin: Programming the Open Blockchain: Unlocking Digital Cryptocurrencies](#), O'Reilly Media, Inc.
- S. Haber, S. Stornetta (1991), [How to Time-Stamp a Digital Document](#), Journal of Cryptology, Vol. 3. Num. 2
- David Chaum, [Blind Signatures for Untraceable Payments](#), Advances in Cryptology Proceedings of Crypto 82
- Jordan Baczuk, [How to Generate a Bitcoin Address — Step by Step](#), Medium blog , 2018
- Beatrice Bertani, [La crittografia nel sistema di moneta digitale Bitcoin](#), Tesi di Laurea in Matematica, Università di Bologna

# Non Fungible

- **Non Fungible** in italiano “non fungibile” o “non riproducibile”
- **Fungibile** agg. [dal lat. mediev. *fungibilis*, der. del lat. *fungi* «fungere»] In diritto, di beni che, non avendo specifica individualità, possono tenere l'uno il posto dell'altro agli effetti giuridici (es. denaro)
- Il termine non fungibile indica quindi un pezzo **unico, non riproducibile**

# Token

- **Token** in italiano “gettone” o “pedina”
- Il termine **token** degli NFT è un **certificato di proprietà e autenticità** memorizzato in una blockchain
- **NFT** è dunque un certificato memorizzato in una blockchain per un bene unico (non fungibile) sia fisico che digitale

# Proprietà di un NFT

- Gli NFT **non possono essere scambiati** tra di loro
- Un NFT è **indivisibile**, non può essere suddiviso in parti più piccole
- Gli NFT sono **immutabili** poichè sono memorizzati in una blockchain
- Un NFT **può essere venduto** (cambio di proprietà)
- Tutti possono **verificare la proprietà** di un NFT e consultare anche la storia dei passaggi di proprietà (compreso il prezzo di vendita) fino al creatore originale (es. [CryptoPunk 7136](#))

# NFT e beni digitali

- Se io possiedo un bene digitale (es. un'immagine originale) posso registrarne la proprietà e l'autenticità
- La proprietà è garantita grazie all'utilizzo della blockchain e dal fatto di aver registrato per primo l'oggetto
- L'autenticità è garantita dalla firma digitale dell'hash del bene digitale

# NFT e beni fisici

- Anche un bene fisico può essere registrato con un NFT?
- Si, ma qui le cose si fanno più complicate perché non esiste un metodo standard per legare il certificato NFT all'oggetto fisico (non possiamo calcolare l'hash)
- E' necessario creare una versione digitale dell'oggetto fisico (es. foto in alta risoluzione)
- Ci sono alcuni soluzioni proposte (es. [Virtual Equivalents of Real Objects, VEROs](#)) ma questo è un argomento tuttora di ricerca

# Come creare un NFT?

- **ERC-721** è uno standard Ethereum per realizzare NFT tramite smart contract
- Lo standard contiene un'interfaccia da implementare (in Solidity)

```
interface ERC721 {
    event Transfer(address indexed _from, address indexed _to, uint256 indexed _tokenId);
    event Approval(address indexed _owner, address indexed _approved, uint256 indexed _tokenId);
    event ApprovalForAll(address indexed _owner, address indexed _operator, bool _approved);

    function balanceOf(address _owner) external view returns (uint256);
    function ownerOf(uint256 _tokenId) external view returns (address);
    function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) external payable;
    function safeTransferFrom(address _from, address _to, uint256 _tokenId) external payable;
    function transferFrom(address _from, address _to, uint256 _tokenId) external payable;
    function approve(address _approved, uint256 _tokenId) external payable;
    function setApprovalForAll(address _operator, bool _approved) external;
    function getApproved(uint256 _tokenId) external view returns (address);
    function isApprovedForAll(address _owner, address _operator) external view returns (bool);
}
```

# Ambiti d'utilizzo

- Mercato dell'arte digitale
- Collezionismo, aste
- Verifica dell'autenticità di un prodotto
- Mercato immobiliare
- Brevetti e proprietà intellettuale
- Votazioni
- Videogiochi
- Supply chain
- Biglietteria automatica



# Riferimenti

- Debarshi Chaudhury, [NFTs Are Fueling Authenticity For Digital Assets And Have The Potential To Create New Use Cases](#), Forbes, 29 Giugno 2021
- Larva Labs, [CryptoPunks](#)
- Mitchell Clark, [Coinbase launches social NFT marketplace in limited beta, just as NFT sales dive](#), The Verge, 20 Aprile 2022
- Axios, [Report: NFT sales exceeded \\$17B in 2021](#)
- Melanie Kramer, Stephen Graves e Daniel Phillips, [Beginner's Guide to NFTs: What Are Non-Fungible Tokens?](#), Decrypt, 2022
- Ethereum, [Non-fungible tokens \(NFT\)](#)
- Jacob Kastrenakes, [Beeple sold an NFT for \\$69 million](#), The Verge

# Il protocollo Bitcoin

- E' un insieme di regole che governano la rete Bitcoin
- In particolare contiene:
  - Protocollo di rete basato su TCP per la comunicazione tra i nodi
  - Definizione del formato dei blocchi e delle transazioni
  - Algoritmo di mining tramite Proof of Work
  - Algoritmo di verifica dei nodi
  - Meccanismo di gestione delle transazioni in attesa di essere processate (mempool)
  - Gestione delle chiavi private (wallet)
  - Strumenti crittografici come SHA256, RIPEMD160, ECDSA, etc
- Software open source disponibile all'indirizzo <https://github.com/bitcoin/bitcoin>
- Risorsa per sviluppatori Bitcoin: <https://developer.bitcoin.org/index.html>

# Transazioni basate su conto (account based)

Creo 25 ₩ e li assegno ad **Alice**

**Alice** trasferisce 17 ₩ a **Bob**

**Bob** trasferisce 8 ₩ a **Carol**

**Carol** trasferisce 5 ₩ a **Alice**

**Alice** trasferisce 15 ₩ a **David**

Alice: ~~25~~ ~~8~~ 13

Bob: ~~17~~ 9

Carol: ~~8~~ 3

David:

Firme digitali:  Alice

 Bob

 Carol

 David

# Transazioni basate su registro (ledger)

Creo 25 ₶ e li assegno ad **Alice**

**Alice** trasferisce 17 ₶ a **Bob**

**Bob** trasferisce 8 ₶ a **Carol**

**Carol** trasferisce 5 ₶ a **Alice**

**Alice** trasferisce 15 ₶ a **David**

1	IN: OUT: [ 25 ₶ → <b>Alice</b> ]
2	IN: 1 [ 0 ] OUT: 17 ₶ → <b>Bob</b> , 8 → <b>Alice</b>
3	IN: 2 [ 0 ] OUT: 8 ₶ → <b>Carol</b> , 9 → <b>Bob</b>
4	IN: 3 [ 0 ] OUT: 5 ₶ → <b>Alice</b> , 3 → <b>Carol</b>
5	IN: 4 [ 0 ], 2 [ 1 ] OUT: 15 ₶ → <b>David</b> , -2 → <b>Alice</b>

15 > 13 la transazione 5 non è valida

Firme digitali:  Alice

 Bob

 Carol

 David

# Transazioni: cambio indirizzo

- Alice può creare più di un indirizzo Bitcoin
- In una transazione Alice può inviare il residuo di una transazione a un altro indirizzo (sempre gestito da Alice)
- Questa operazione è chiamata **change address**
- Può essere utile per questioni di privacy, esempio:
  - Alice genera 2 indirizzi x e y
  - Alice riceve 20฿ da Bob sull'indirizzo x
  - Alice invia 10฿ a Carol e invia il restante a x (questa transazione, essendo legata alla precedente, identifica un'operazione di recupero)
  - Alice invia 10฿ a Carol e invia il restante a y (non è chiaro se y è un indirizzo gestito da Alice oppure un altro indirizzo gestito da terzi)

# Transazioni: verifica efficiente

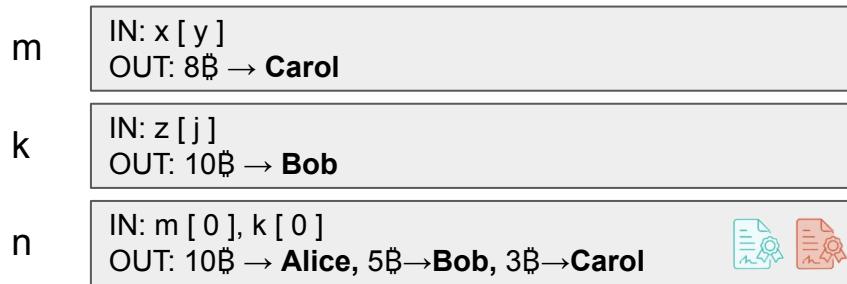
- Come è possibile verificare che una transazione sia valida?
- E' necessario verificare la catena degli input delle transazione rispetto al blocco corrente ed eseguire il calcolo dell'ammontare a disposizione
- Non è necessario verificare tutto lo storico delle transazioni della blockchain

# Transazioni: consolidamento fondi

- Ogni transazione può avere più input e più output
- Ciò vuol dire che è possibile dividere dei fondi in più indirizzi o consolidare dei fondi su un unico indirizzo

# Transazioni: pagamenti congiunti

- E' possibile effettuare un pagamento dividendolo tra due o più indirizzi Bitcoin
- E' necessario creare una transazione con 2 input e 1 output
- Esempio: Carol e Bob vogliono inviare 5฿ a testa ad Alice



Firme digitali:  Alice  Bob  Carol

# Dettaglio di una transazione

Metadati

Input

Output

```
{
  "hash": "b6f6991d03df0e2e04daafffc6bc418aac66049e2cd74b80f14ac86db1e3f0da",
  "ver": 1,
  "vin_sz": 1,
  "vout_sz": 2,
  "lock_time": "0",
  "size": 258,
  ...
  "in": [
    {
      "prev_out": {
        "hash": "a3e2bcc9a5f776112497a32b05f4b9e5b2405ed9",
        "n": "2"
      },
      "scriptSig": "76a914641ad5051edd97029a003fe9efb29359fceee409d88ac"
    }
  ],
  "out": [
    {
      "value": "9.800000",
      "scriptPubKey": "76a914641ad5051edd97029a003fe9efb29359fceee409d88ac"
    },
    {
      "value": "1.200000",
      "scriptPubKey": "76a914641ad5051edd97029a003fe9efb29359fceee409d88ac"
    }
  ]
}
```

# Medati, Input

- Un blocco ha una dimensione di **1 MB** ( $\approx$ 2000 transazioni per blocco)
- Metadati (80 byte):
  - **vin\_sz** = numero di input, **vout\_sz** = numero di output
  - **lock\_time** = numero del blocco da attendere prima di inserire la transazione
  - **size** = dimensione del blocco
  - ...
- Input:
  - transazione precedente
  - indice della transazione
  - **scriptSig** = le operazioni da eseguire sull'input

# Output

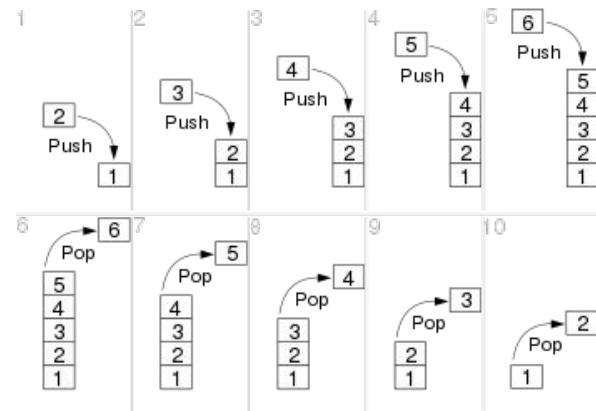
- Output:
  - valore della transazione
  - $\sum \text{output} \leq \sum \text{input}$
  - se la somma dell'output è minore dell'input la differenza è detta **transaction fee**, che viene assegnata al miner che pubblica la transazione
  - **scriptPubKey** = le operazioni da eseguire sull'output

# Validazione di una transazione

- Per verificare la validità di una transazione vengono eseguiti in sequenza gli script **scriptSig** degli input e gli script **scriptPubKey** degli output
- Se il risultato non genera nessun errore allora la transazione risulta essere valida, altrimenti la transazione viene rifiutata
- In un certo senso la validazione è programmabile attraverso questi script che possono di fatto essere considerati degli “smart contract”

# Bitcoin Script

- E' il linguaggio di scripting utilizzato nel protocollo Bitcoin per il processo di validazione di un blocco
- E' stato ideato basandosi sul linguaggio [Forth](#)
- E' un linguaggio di programmazione basato su stack (**stack-based**)
- Lo **stack** (pila in Italiano) è una struttura dati con due operazioni: push, pop



# Bitcoin Script (2)

- Ogni istruzione in Bitcoin Script è eseguita in maniera lineare, non ci sono istruzioni di loop
- Il numero di istruzioni fornisce dunque un limite superiore sul tempo di esecuzione di uno script e della memoria utilizzata
- Il linguaggio Bitcoint Script non è Turing completo
- E' stato progettato senza la presenza di loop per impedire l'esecuzione di potenziali loop infiniti
- Il linguaggio Script ha solo **256** istruzioni, espresse tramite 1 byte. Di queste 15 istruzioni sono disabilitate e 75 sono riservate (vuote per sviluppi futuri)
- Quindi il set di istruzioni si riduce a **166**

# Bitcoin Script (3)

- Ci sono istruzioni comuni in ogni linguaggio di programmazione come if-then, operatori logici, istruzioni matematiche, istruzioni di return, gestione degli errori
- Sono presenti istruzioni specifiche per eseguire funzioni crittografiche: hash, verifica della firma e istruzioni speciali come **CHEKMULTISIG** che consente di verificare firme multiple in un'unica istruzione
- L'istruzione CHECKMULTISIG richiede  $n$  chiavi pubbliche e un parametro di soglia  $t$ . Questa istruzione viene eseguita correttamente se ci sono almeno  $t$  firme valide

# Alcune istruzioni comuni del linguaggio Script

<b>OP_DUP</b>	Duplicates the top item on the stack
<b>OP_HASH160</b>	Hashes twice: first using SHA-256 and then RIPEMD-160
<b>OP_EQUALVERIFY</b>	Returns true if the inputs are equal. Returns false and marks the transaction as invalid if they are unequal
<b>OP_CHECKSIG</b>	Checks that the input signature is a valid signature using the input public key for the hash of the current transaction
<b>OP_CHECKMULTISIG</b>	Checks that the $k$ signatures on the transaction are valid signatures from $k$ of the specified public keys.

# Esecuzione di uno Script

- Per eseguire uno Script si utilizza uno stack dove vengono inserite (push) o prelevate (pop) istruzioni o dati
- Non è presente nessuna memoria al di fuori dello stack; ciò rende il linguaggio Script molto semplice dal punto di vista computazionale
- Ci sono due tipi di istruzioni: **dati** e **opcode**
- **Opcode** sono le istruzioni del linguaggio Script che di solito prendono in input i dati presenti in cima allo stack

# Esempio: Pay-to-PubkeyHash script

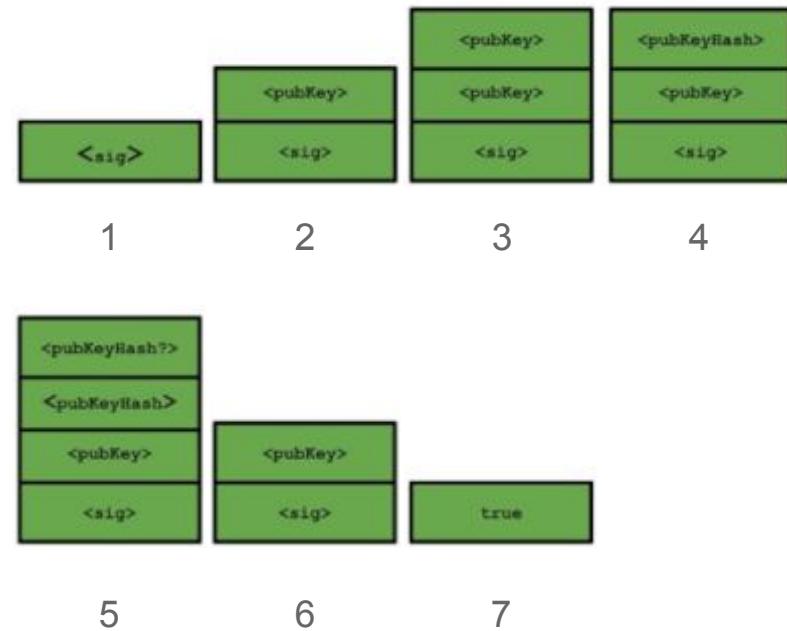
```

1 <sig>
2 <pubKey>
-----
3 OP_DUP
4 OP_HASH160
5 <pubKeyHash?>
6 OP_EQUALVERIFY
7 OP_CHECKSIG
  
```

scriptSig

scriptPubKey

Stack di esecuzione:



# Tipologie di Script

- La stragrande maggioranza degli script nelle transazioni Bitcoin è del tipo precedente, ossia **Pay-to-PubkeyHash**
- Questo script utilizza una chiave pubblica e richiede la firma della stessa per effettuare una transazione
- Altre tipologie di script sono:
- **Proof of burn**: uno script che “distrugge” una transazione. L’implementazione prevede l’utilizzo di un OP\_RETURN. Tutte le istruzioni dopo quella di ritorno non vengono eseguite e quindi possibile scrivere qualsiasi cosa nel ledger
- **Pay-to-script-hash**: transazione basata sull’hash di uno script al posto di un indirizzo Bitcoin

# Deposito a garanzia

- Alice vuole acquistare un bene materiale da Bob
- Alice vuole pagare Bob in Bitcoin ma solo dopo aver ricevuto la merce e Bob vuole essere sicuro di essere pagato dopo aver consegnato la merce
- Questo scenario può essere implementato in Bitcoin script con il MULTISIG
- E' necessario introdurre una terza persona fidata: Judy
- Alice crea una transazione MULTISIG con Bob e Judy in modo che per riscattare la transazione siano necessarie almeno 2 firme delle tre (Alice, Bob e Judy).
- Se Bob invia la merce e non si riscontrano problemi, Alice e Bob possono firmare la transazione e Bob riceverà il compenso (Judy non entra in gioco)

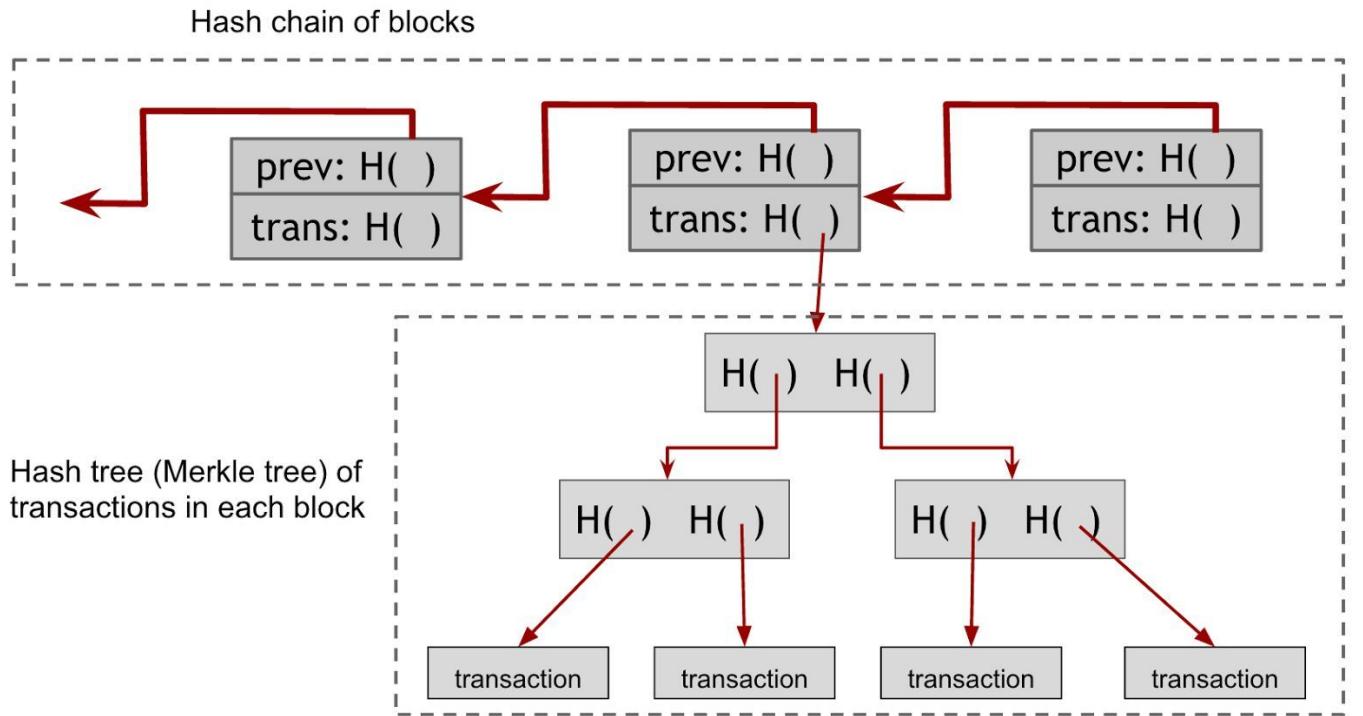
# Deposito a garanzia (2)

- In caso di problemi Judy può intervenire e decidere se trasferire la transazione ad Alice (es. la merce è danneggiata o non rispondente agli accordi) oppure confermare il pagamento verso Bob (es. Alice si rifiuta di pagare dopo aver ricevuto la merce)

# Lock time

- Alice vuole creare una transazione verso Bob con un tempo di esecuzione  $t$  (parametro lock\_time)
- Questo parametro può essere espresso in termini temporali (timestamp) o tramite il numero del blocco
- La transazione verrà inserita nel registro distribuito solo dopo la scadenza del tempo
- In questo modo si possono implementare diversi casi d'uso: pagamenti a rate, abbonamenti, pagamenti con rimborso (utilizzando il MULTISIGN), etc

# Blocchi in Bitcoin



# Coinbase

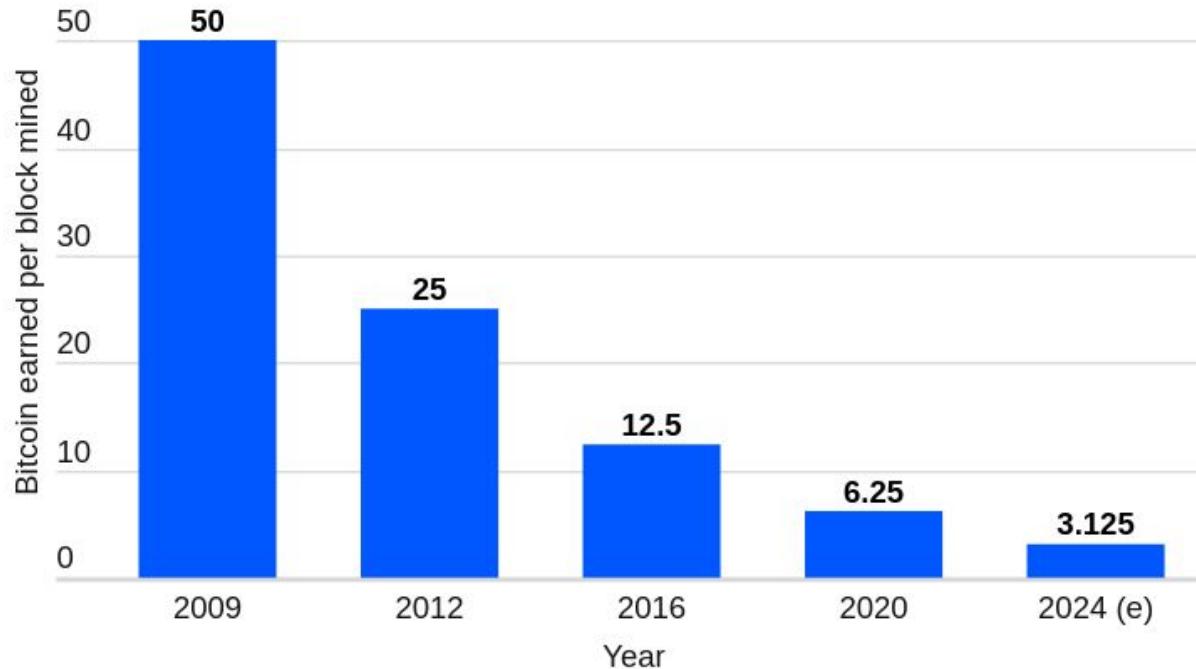
- La prima transazione in ogni blocco è chiamata **coinbase**
- Questa è una transazione speciale poiché contiene la creazione di bitcoin (mining) da trasferire verso l'indirizzo del miner
- Ha sempre un solo input e un solo output
- L'input non riscatta nessuna transazione precedente, l'hash precedente è 0
- Il valore dell'output è di circa **6,5 B** (fino al 2024) + le commissioni (**fees**) raccolte da ogni transazione inclusa nel blocco
- Contiene un campo denominato “coinbase” che può contenere del testo arbitrario (spesso il nome del mining pool)

# Blocco genesi

- Il campo coinbase del blocco genesi, ossia il primo blocco del registro Bitcoin è stato creato il **3 gennaio 2009** da Satoshi Nakamoto e conteneva la stringa:  
 “The Times 03/Jan/2009 Chancellor on brink of second bailout for banks.”



# Ricompensa per i miner



# La rete Bitcoin

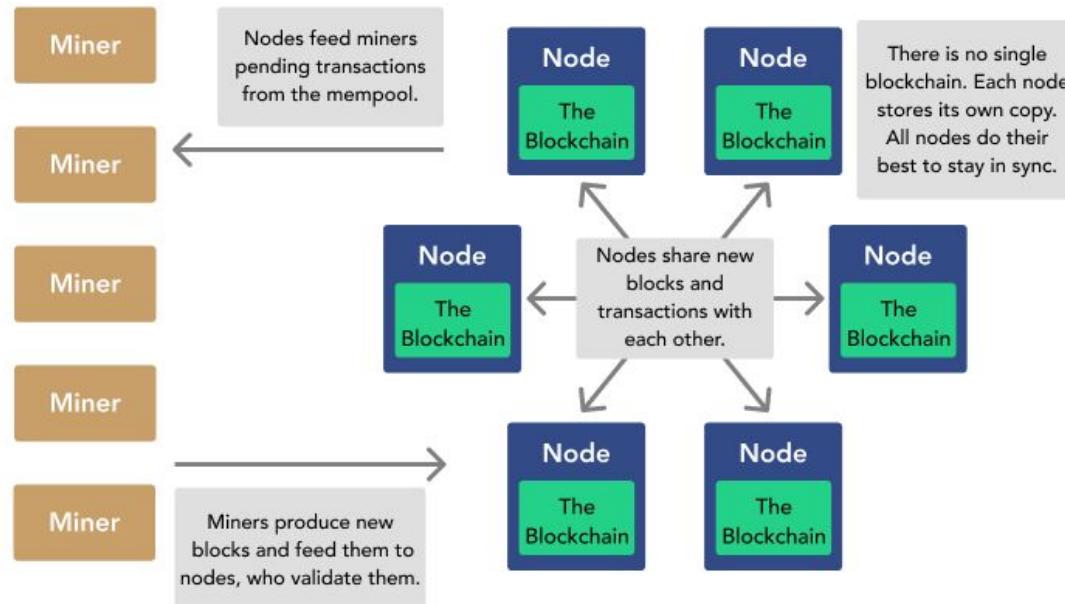
- E' una rete decentralizzata di tipo **peer-to-peer**
- Tutti i nodi sono equivalenti, non c'è una gerarchia
- La topologia della rete è casuale, nel senso che un nodo si collega agli altri in modo casuale
- Quando un nodo si aggancia alla rete manda un messaggio ad un seed node che conosce già. Questo nodo risponde con un elenco di nodi a cui lui è connesso. Con una scelta casuale tra questi nodi il nuovo nodo inizia a comunicare con gli altri
- Quando c'è un nodo ha un nuovo messaggio lo propaga agli altri nodi di sua conoscenza (algoritmo di flooding o ossip)

# Esempio di transazione

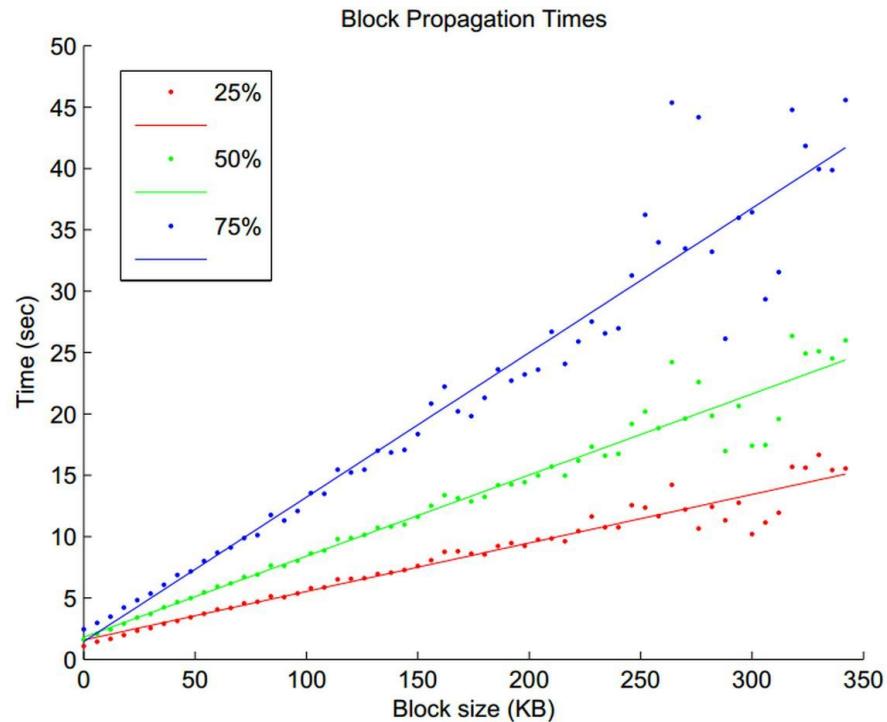
- Alice vuole inviare  $x \text{ \AA}$  a Bob
- Il client di Alice invia la transazione verso un nodo della rete
- Questo client memorizza la transazione in un memoria temporanea (**mempool**) e propaga la transazione verso gli altri nodi; se un nodo ha già la transazione nel suo pool non viene più propagata verso altri
- Il mempool contiene dunque le transazioni in attesa di risoluzione del PoW da parte di un miner

# Nodi e Miner

## Bitcoin Network Interactions



# Tempi di propagazione di un blocco



Source: Yonatan Sompolinsky and Aviv Zohar: "Accelerating Bitcoin's Transaction Processing" 2014

# Nodi leggeri (lightweight node)

- Un **lightweight node** è un nodo che non memorizza l'intera blockchain ma soltanto una porzione utile ad eseguire le operazioni correnti
- Questi nodo sono anche chiamati **thin client** o **Simple Payment Verification (SPV)**
- Di solito la dimensione del registro memorizzato in un thin client è di circa 1/1000 del totale della blockchain

# Limiti della rete Bitcoin

- Un grande limite della rete Bitcoin è legato al numero di transazioni al secondo
- Considerando che un blocco è di 1 MB e ogni transazione è almeno di 250 byte, in un blocco ci possono essere al massimo 4000 transazioni
- Un blocco è inserito nel registro distribuito ogni 10 minuti, quindi circa 7 transazioni al secondo (tps)
- Il circuito delle carte di credito Visa gestisce in media 2000 tps con picchi di 24'000 tps. Paypal gestisce in media 100 tps

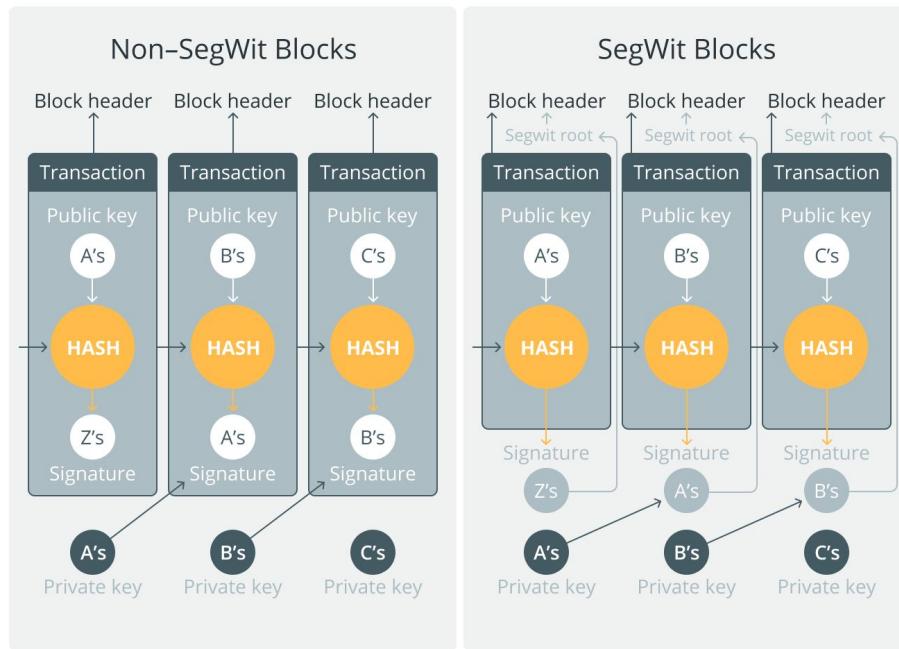
# Come evolvere la rete Bitcoin

- **Hard fork**, ossia delle modifiche sostanziali del codice che fanno deviare la blockchain verso una nuova versione del protocollo non compatibile con la precedente
- **Soft fork**, introdurre delle modifiche sulla validazione dei blocchi che siano più restrittive. In questo modo i vecchi nodi continuano ad accettare tutti i blocchi mentre quelli nuovi accetteranno solo i nuovi blocchi. Propagando il nuovo software nella rete si può ottenere un effetto di migrazione verso il nuovo
- Ad esempio, l'introduzione del **Pay-to-script-hash** è stata eseguita grazie ad un soft fork

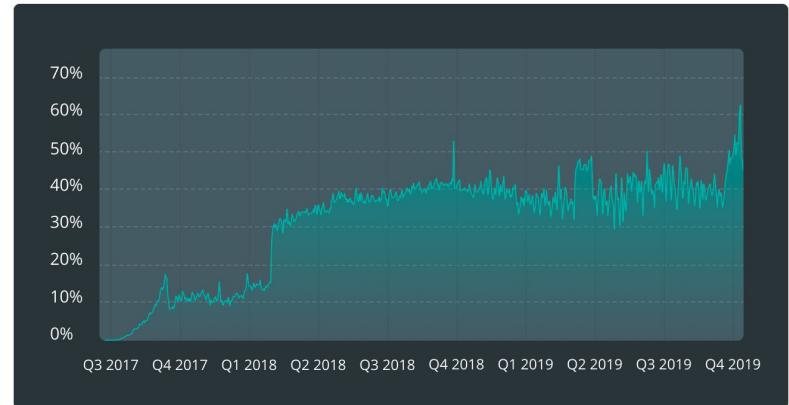
# Storia dei fork di Bitcoin

- **2014: Bitcoin XT**, blocchi di 8 MB con 24 tps. Progetto di Mike Hearn, dopo un inizio promettente non è mai decollato ed è stato abbandonato
- **2015:** Pieter Wuille, sviluppatore del core di Bitcoin, ha presentato l'idea del Segregated Witness (**SegWit**) come soft fork. L'idea è quella di ridurre la dimensione di una transazione, spostando la firma della transazione nell'header.
- **2016: Bitcoin Classic**, alcuni programmatori entusiasti delle modifiche introdotte con Bitcoin XT hanno ridotto il blocco a 2 MB
- **2017: Bitcoin Cash**, hard fork di Bitcoin con blocchi di 8 MB

# SegWit



## SegWit BTC transactions

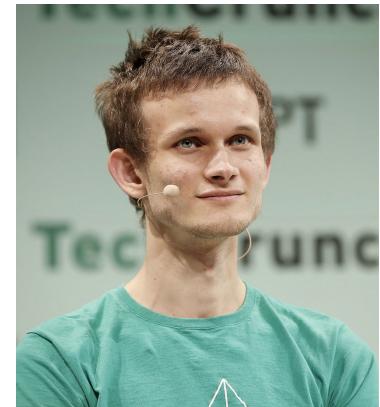


# Riferimenti

- Capitolo 3 del libro [Bitcoin and Cryptocurrency Technologies](#), Princeton Press
- Axel Hodler, [Dissecting a Bitcoin Script](#), Medium article, 2020
- Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y. (2017). [The first collision for full SHA-1](#), In: Katz, J., Shacham, H. (eds) Advances in Cryptology – CRYPTO 2017. CRYPTO 2017. Lecture Notes in Computer Science(), vol 10401. Springer
- Andreas M. Antonopoulos, [Mastering Bitcoin: Programming the Open Blockchain](#), O'Reilly, 2017
- Katelyn Peters, [A History of Bitcoin Hard Forks](#), Investopedia, 2021
- [List of bitcoin forks](#), Wikipedia
- Georgia Weston, [A Complete Guide on Segregated Witness \(SegWit\)](#), 2022

# Ethereum

- Ethereum è una blockchain pubblica con la possibilità di eseguire applicazioni che girano all'interno della blockchain
- **Ethereum Virtual Machine** (EVM) è il “computer” che esegue queste applicazioni
- Le richieste di esecuzione di un'applicazione sono chiamate transazioni
- Ogni transazione modifica lo stato della EVM che viene memorizzato nel registro distribuito (blockchain)
- Ethereum è stata creata da Vitalik Buterin nel 2013

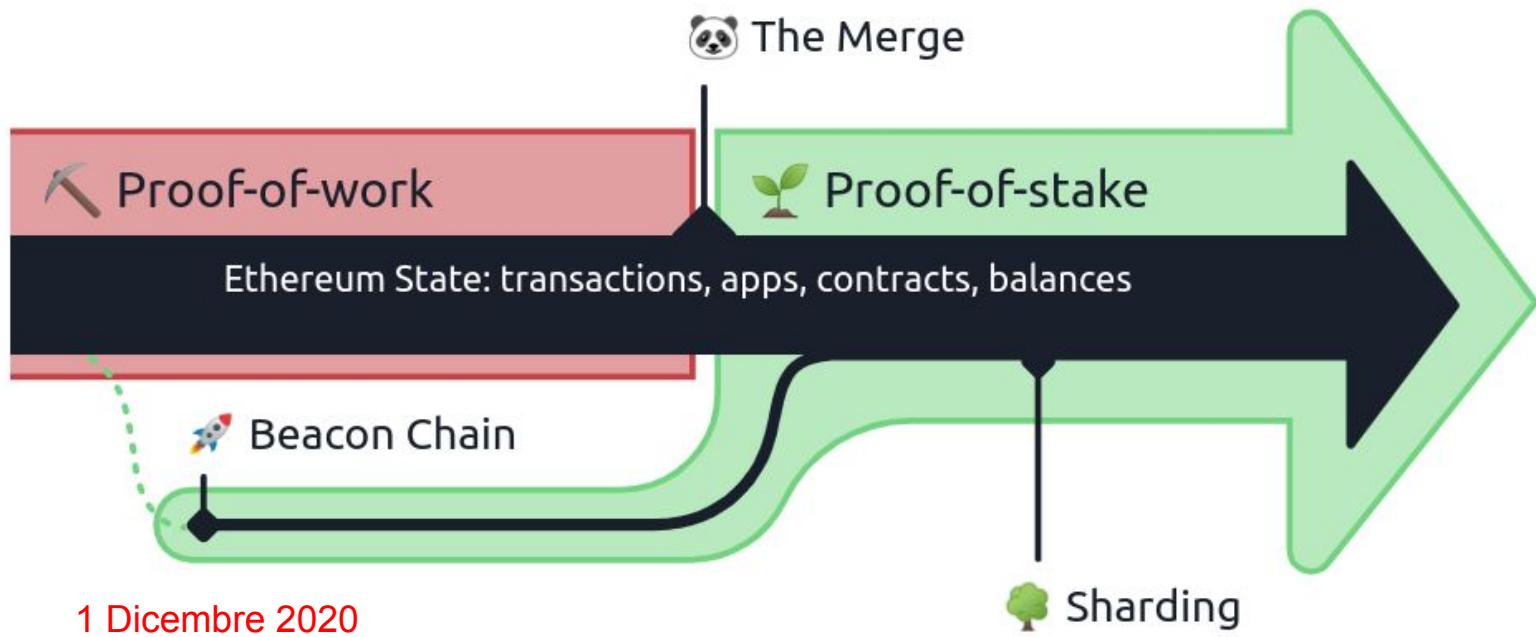


# Proof of Stake

- **Proof of Stake** (PoS) è il nuovo algoritmo di consenso utilizzato da Ethereum a partire dal 2022
- Ha sostituito il **Proof of Work** con un soft fork denominato [The Merge](#)
- Il Proof of Stake ha ridotto il consumo energetico necessario per far girare la rete Ethereum del 99,95%
- L'idea del Proof of Stake è riservare temporaneamente un deposito di almeno **32 ETH** (Ether, la moneta di Ethereum) ed essere inseriti in una coda di attivazione
- Quando un nodo validatore è attivato in questa coda esegue la validazione del blocco e invia un voto (attestato) in favore del blocco

# The Merge

15 Settembre 2022



# Slot ed epoches

- Con il PoW il tempo di risoluzione dettava la frequenza di mining (ogni 10 minuti)
- Con il PoS il tempo è fisso e diviso in **slot** (ogni 12 secondi) ed **epoches** (contenenti 32 slot)
- Un nodo validatore **è scelto a caso** tra quelli in attesa per ogni slot. Validatori con depositi maggiori di riserva avranno più possibilità di essere eletti. Il validatore prescelto è responsabile per la creazione di nuovo blocco e per la trasmissione verso gli altri nodi
- In ogni slot è anche presente una **commissione di validatori** preposta a votare la validità del blocco appena creato

# Ricompensa al validatore

- Quando un validatore viene selezionato ed esegue il suo compito viene ricompensato (con degli ETH coniati) + eventuali manci presenti nel blocco
- Se il comitato di verifica del blocco rileva delle manomissioni sulle transazioni presenti in un blocco il nodo validatore viene penalizzato, prelevando una percentuale di ETH dal suo deposito. Inoltre il validatore sarà escluso, per un periodo di tempo, dalla possibilità di far parte nuovamente del processo di validazione
- Ogni validatore è quindi incentivato economicamente a comportarsi correttamente

# Ether

- **Ether** (ETH, simbolo  $\Xi$ ) è la criptovaluta utilizzata per diversi scopi in Ethereum
- È l'unica forma di pagamento per la tassa (fee) delle transazioni
- È la moneta utilizzata nel mercato **DeFi** (Decentralized finance)
- È lo strumento utilizzato come **gas fee** per l'esecuzione di transazioni
- Gli ETH vengono coniati per ogni blocco proposto e per ogni epoca verificata dal comitato di validatori
- Circa  $\frac{1}{8}$  degli ETH coniati va al validatore scelto a caso e gli altri sono equamente distribuiti tra tutti i partecipanti al processo di verifica

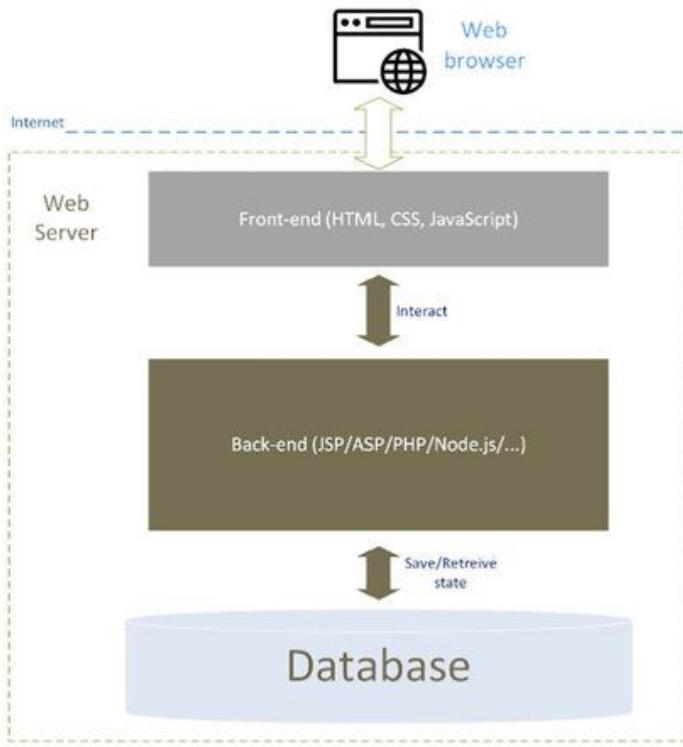
# Burning ETH

- Ethereum oltre a coniare (**minting**) prevede anche di bruciare (**burning**) ETH
- Bruciare ether equivale a rimuoverlo permanentemente dalla circolazione
- L'operazione di **burning** avviene in ogni transazione Ethereum
- Quando un utente paga il **gas fee** per una transazione, questo viene bruciato
- Questo per evitare che i gas fee vengano utilizzati in qualche modo dai validatori
- Quando il traffico delle transazioni è elevato, i blocchi possono bruciare più ETH di quelli che verranno coniati
- Questo meccanismo tende a compensare il numero di ETH coniati, evitando possibili svalutazioni

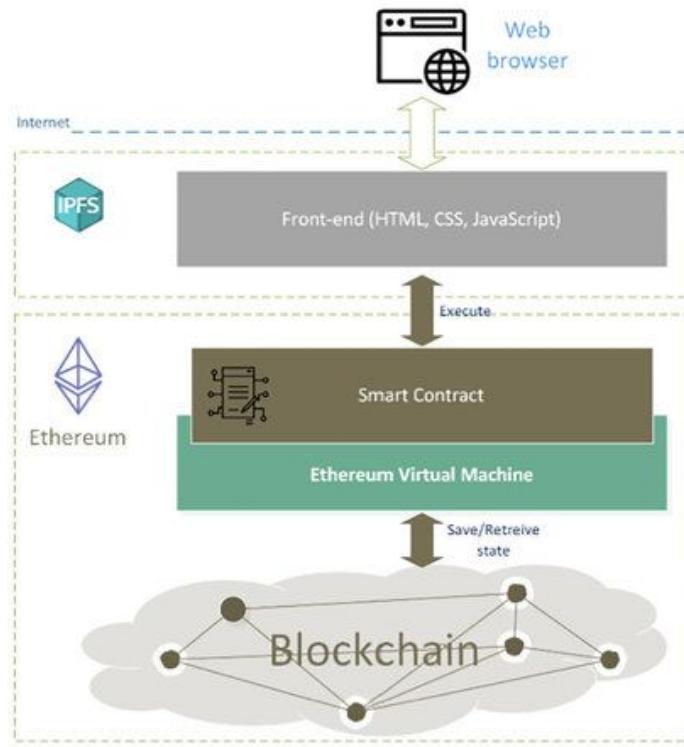
# dapp

- Un'applicazione decentralizzata (**dapp**) è un'applicazione creata su una rete distribuita che utilizza uno o più smart contract e un frontend, con un interfaccia utente
- Una dapp ha le seguenti caratteristiche:
  - **Decentralizzata**, vengono eseguite nella rete Ethereum
  - **Deterministica**, il risultato dell'esecuzione di una dapp deve essere lo stesso a prescindere dall'ambiente nel quale viene eseguito
  - **Turing completa**, le dapp possono eseguire qualsiasi azione date le adeguate risorse
  - **Isolata**, l'esecuzione di una dapp avviene in un ambiente virtuale denominato Ethereum Virtual Machine

### Traditional Web Application



### Decentralized Application (DApp)



# Smart contract

- Gli **smart contract** rappresentano la parte backend di una dapp
- Uno smart contract è un programma che viene eseguito dalla blockchain Ethereum
- Una volta che uno smart contract è inserito nella blockchain non può essere modificato
- Pro:
  - **Zero downtime**, gli smart contract vengono eseguiti su una rete distribuita per cui saranno sempre disponibili in qualche nodo
  - **Privacy**, gli smart contract possono essere eseguiti in maniera pseudo-anonima (basta un l'indirizzo Ethereum)
  - **Resistenza alla censura**, uno smart contract non può essere censurato, bloccato da uno o più partecipanti alla rete

# Smart contract (2)

- Pro:
  - **Data integrity**, gli smart contract non possono essere alterati grazie alle proprietà crittografiche della blockchain
  - **Trustless computation/verifiable behavior**, gli smart contract possono essere analizzati e la loro esecuzione è totalmente prevedibile, non c'è bisogno di fidarsi di un'autorità centrale
- Contro:
  - **Manutenzione**, difficile da eseguire poiché il codice non può essere modificato in una blockchain, è possibile solo rilasciare versioni più aggiornate
  - **Performance**, gli smart contract non possono essere troppo performanti perché la loro esecuzione è soggetta a tutti i meccanismi di verifica di una blockchain
  - **Congestione della rete**, se una dapp utilizza troppo risorsa computazionale può influenzare il traffico dell'intera rete

# Smart contract (3)

- Contro:
  - **Esperienza utente**, è difficile progettare esperienze utenti user-friendly perché all'utente finale può risultare difficile mettere in piedi tutta l'infrastruttura per interagire con la blockchain
  - **Centralizzazione**, le interfacce utenti delle dapp possono girare su server centralizzati e richiamare gli smart contract sulla blockchain. In questo modo una dapp non è più decentralizzata, perdendo di fatto tutti i vantaggi della blockchain

# Account

- In Ethereum ci sono due tipologie di Account:
  - **Externally-owned account** (EOA), gestiti da qualcuno attraverso chiavi private;
  - **Contract account**, uno smart contract pubblicato nella rete Ethereum.
- Tutti gli account possono:
  - ricevere, tenere, e inviare ETH e token;
  - interagire con smart contract pubblicati nella rete Ethereum.

# Account: EOA

- Nessun costo per la creazione di un account
- Può avviare transazioni
- Le transazioni possono essere di ETH o token
- Costruito tramite una coppia di chiavi: pubblica e privata

# Account: smart contract

- La creazione di uno smart contrat ha un costo perché si utilizzano risorse della rete per la memorizzazione
- Possono eseguire transazioni solo in risposta a un'altra transazione
- Le transazioni da un account external possono attivare (trigger) del codice di uno smart contract che può eseguire diverse azioni, come trasferire dei token o creare un altro smart contract
- Gli account degli smart contract non hanno una chiave privata. Sono controllati dalla logica del codice

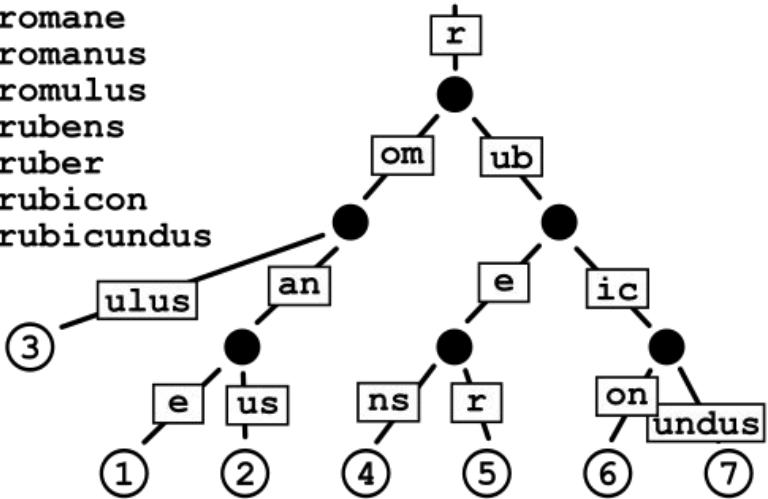
# Formato degli account

- Un account Ethereum è composto da 4 elementi:
  - **nonce**, un contatore del numero di transazioni inviate dall'account. Questo numero viene utilizzato per verificare che le transazioni vengano eseguito soltanto una volta. In un account contract identifica il numero di smart contract creati;
  - **balance**, il numero di **wei** ( $1 \text{ ETH} = 10^{18} \text{ wei}$ ) posseduti dall'account;
  - **codeHash**, hash del codice che verrà eseguito sulla Ethereum Virtual Machine (EVM). Per gli account external questo hash è vuoto;
  - **storageRoot**, conosciuto anche come **storage hash** è un hash di 256 bit del root node di un Modified Merkle Patricia Trie (MPT) contenente i dati dello smart contract sotto forma di coppie chiave, valore.

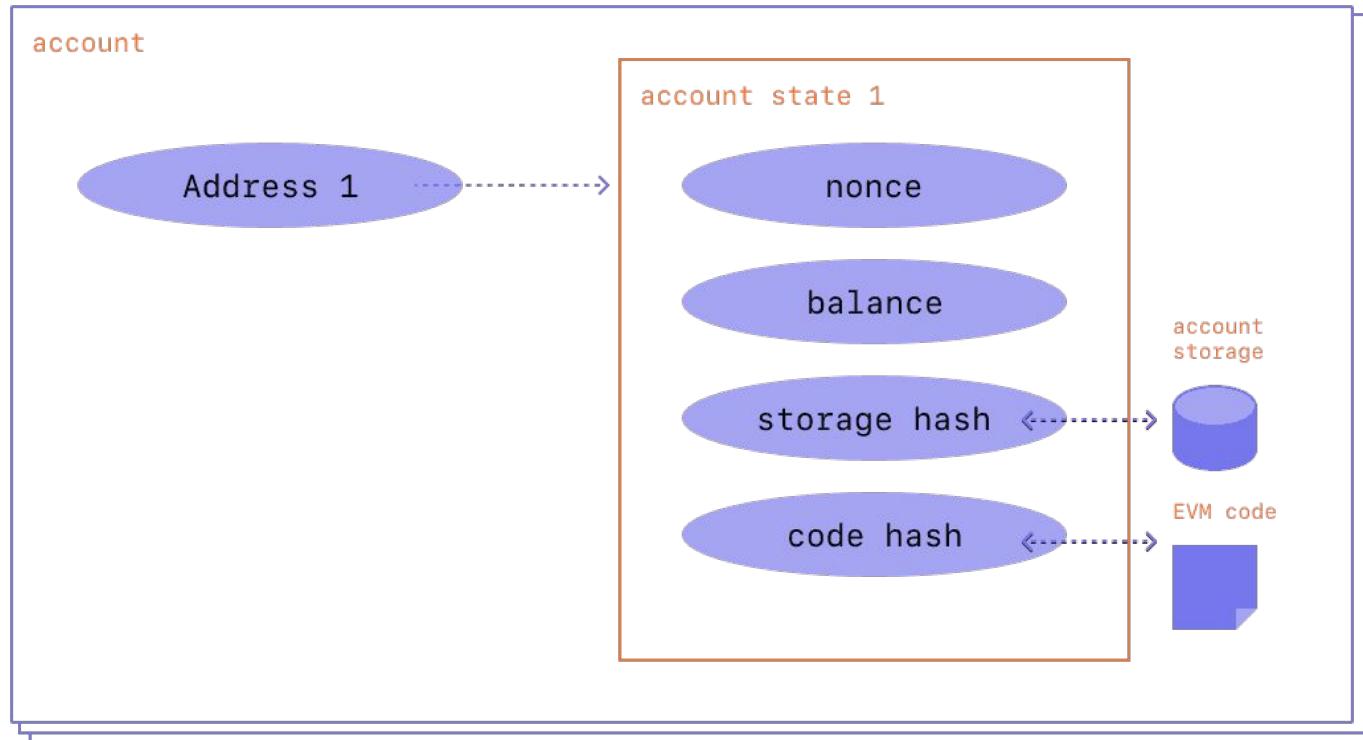
# Patricia Trie

- **Patricia trie o radix trie o radix tree**  
è una struttura dati che consente di memorizzare delle stringhe raggruppandole per caratteri comuni tramite un albero
- Curiosità: la parola inglese “trie” deriva da **retrieval**

1 romane  
 2 romanus  
 3 romulus  
 4 rubens  
 5 ruber  
 6 rubicon  
 7 rubicundus

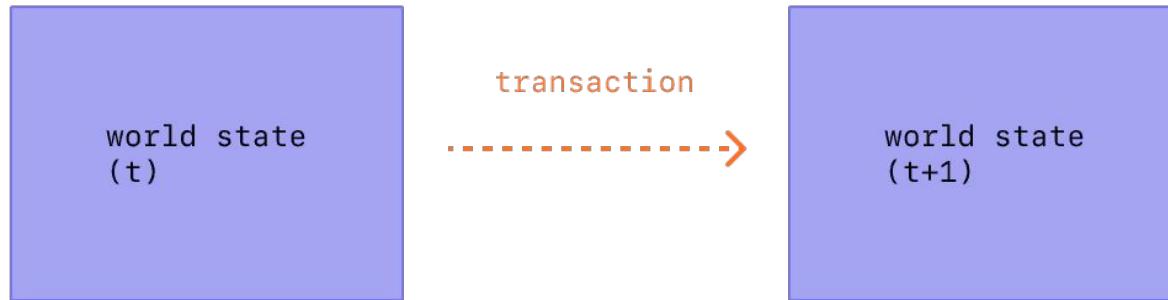


# Ethereum account



# Transazioni

- Una transazione Ethereum si riferisce a un'azione compiuta da un account externally-owned, quindi un account gestito da una persona
- Esempio, Bob invia ad Alice 1 ETH; il saldo di Bob sarà -1 e quello di Alice +1
- Una transazione produce un cambio di stato



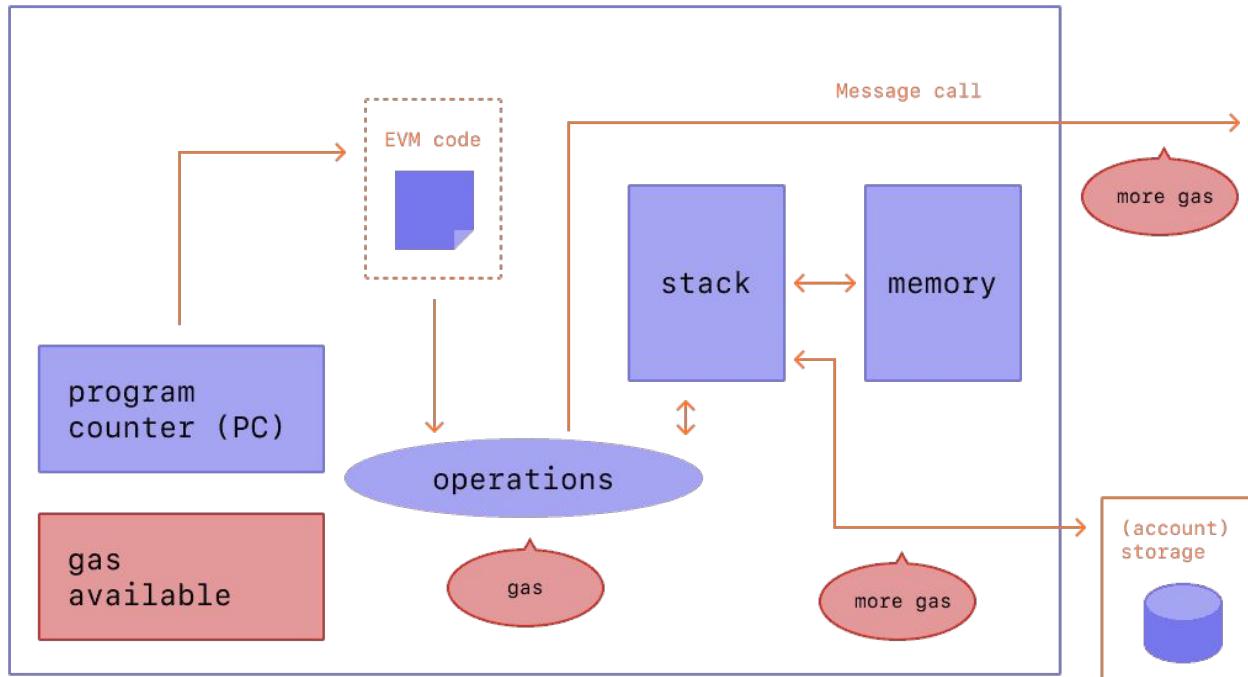
# Transazioni (2)

- Un cambio di stato deve essere propagato in tutta la rete
- Tutte le transazioni richiedono una tassa (fee) per essere eseguite
- Le transazioni contengono le seguenti informazioni:
  - **recipient**, indirizzo del destinatario
  - **signature**, firma della transazione da parte del mittente (con la sua chiave privata)
  - **nonce**, un numero incrementale che identifica la transazione per l'account
  - **value**, l'ammontare di ETH oggetto della transazione (espresso in WEI)
  - **data**, qualsiasi dato si voglia specificare (opzionale)
  - **gasLimit**, il massimo numero di gas unit che la transazione può consumare
  - **maxPriorityFeePerGas**, il massimo numero di gas come mancia da dare al validatore
  - **maxFeePerGas**, la quantità massimo di gas disponibile per pagare la transazione (inclusa la maxPriorityFeePerGas)

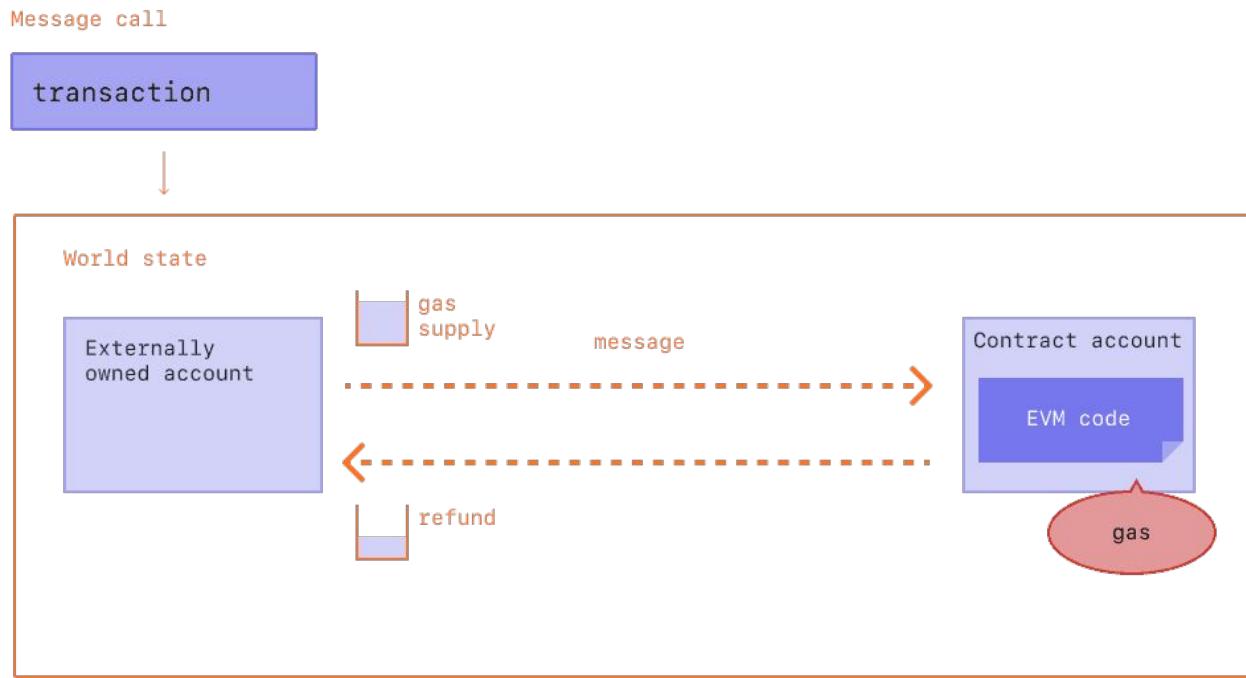
# Gas

- Il **gas** è un'unità di misura relativa alla computazione necessaria per processare una transazione da parte di un nodo validatore
- Il **gasLimit** e il **maxPriorityFeePerGas** determinano il valore massimo pagato al validatore
- Se il gas non viene tutto consumato la parte eccedente viene recuperata
- Calcolo del costo di una transazione:
  - **units of gas used \* (base fee + priority fee)**
  - Es: Alice paga 1 ETH a Bob, gas limit = 21'000 unità, base fee = 10 gwei, priority fee (mancia) di 2 gewi (1 gwei =  $10^{-9}$  ETH = 0.000000001 ETH),
    - $21'000 * (10 + 2) = 252'000$  gwei, 0.000252 ETH  $\approx$  0.38 \$
- Stima dei costi delle transazioni su Ethereum: [Ethereum Gas Tracker](#)

# Consumo del gas

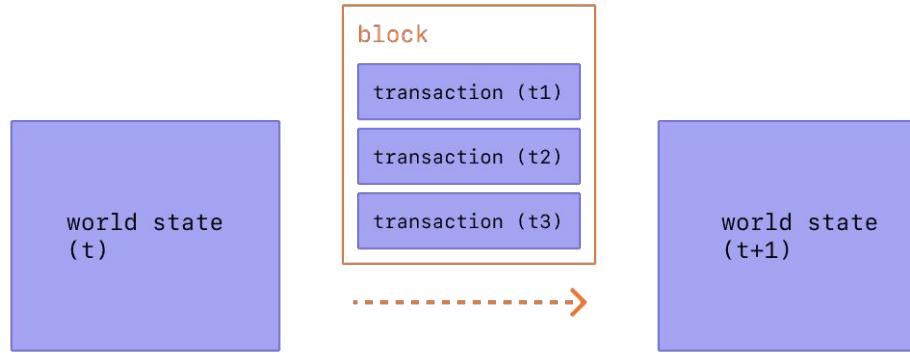


# Gas refund



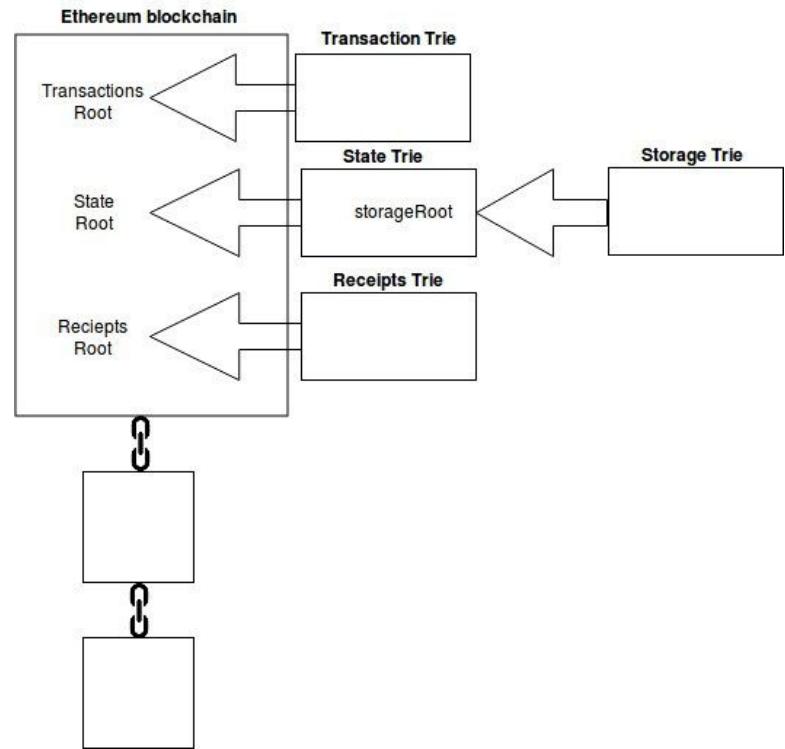
# Blocchi

- Per garantire che tutti i partecipanti alla rete Ethereum mantengano una copia sincronizzata dello stato, le transazioni vengono raggruppate in blocchi (decine o centinaia)
- Anche se le transazioni possono essere inserite decine di volte al secondo, i blocchi vengono ogni **12 secondi** per consentire la sincronizzazione dei dati tra i nodi



# Dove sono memorizzati i dati?

- Gli smart contract, o meglio il codice EVM opcode è memorizzato nella blockchain Ethereum
- L'archivio degli account è memorizzato nello **Storage trie** (database di tipo chiave, valore)
- Lo **storageRoot** è memorizzato nella blockchain Ethereum



# Etherscan.io

The Ethereum Blockchain Explorer

All Filters ▼ Search by Address / Txn Hash / Block / Token / Ens Q

Featured: Build Precise & Reliable Apps with [FTMScan APIs](#). [Learn More!](#)

 ETHER PRICE  
\$1,280.87 @ 0.06733 BTC (-1.05%)

 MARKET CAP  
\$154,377,363,398.00

 TRANSACTIONS  
1,751.65 M (13.4 TPS)

 LAST FINALIZED BLOCK  
15791592

 MED GAS PRICE  
32 Gwei (\$0.86)

 LAST SAFE BLOCK  
15791624

ETHERUM TRANSACTION HISTORY IN 14 DAYS



**Latest Blocks**

Bk	15791676	Fee Recipient <a href="#">Eden Network: Builder</a>	0.19698 Eth
	15 secs ago	161 txns in 12 secs	
Bk	15791675	Fee Recipient <a href="#">builder0x69</a>	0.19138 Eth
	27 secs ago	250 txns in 12 secs	

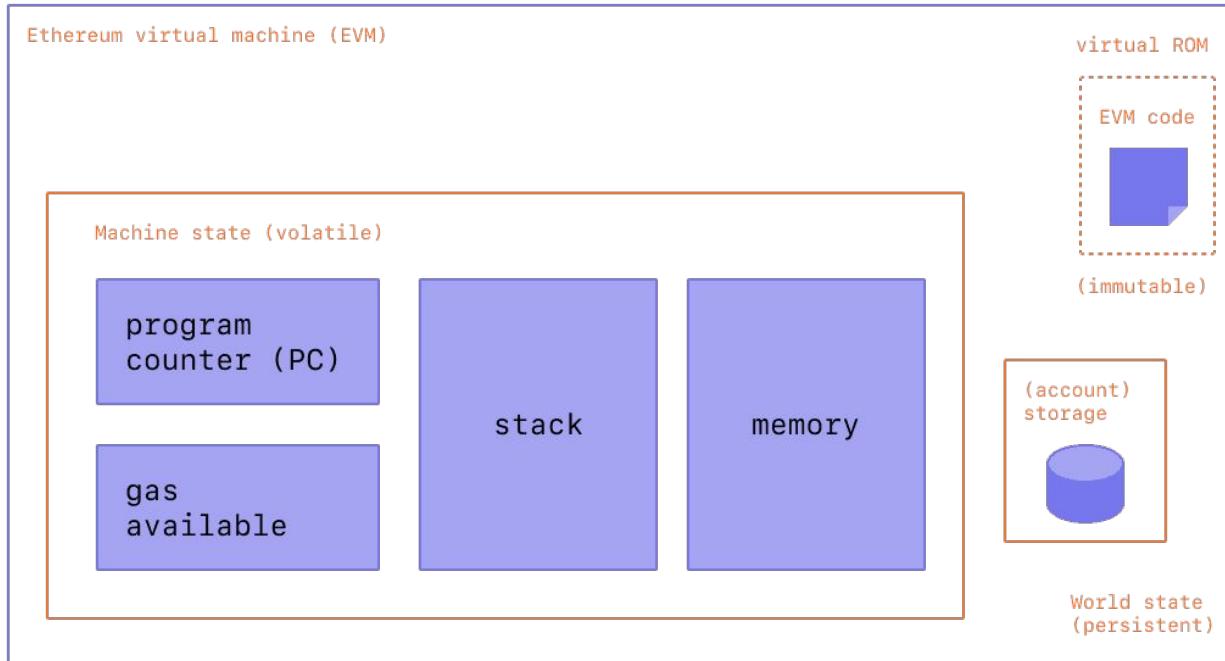
**Latest Transactions**

Tx	0x44a433c34285...	From <a href="#">0xaab27b150451726ec...</a>	0.21241 Eth
	15 secs ago	To <a href="#">0x388c818ca8b9251b39...</a>	
Tx	0x87925c50c15e...	From <a href="#">0x16cd1d708bb08d5f19f...</a>	0.02 Eth
	15 secs ago	To <a href="#">0x68b3465833fb72a70e...</a>	

# Ethereum Virtual Machine

- La Ethereum Virtual Machine (EVM) è il computer virtuale basato su stack per l'esecuzione degli smart contract
- Gli smart contract vengono scritti in un linguaggio ad alto livello, come Solidity e compilati in opcode
- L'opcode viene inserito nella blockchain Ethereum per poter essere eseguito da chiunque
- Il codice inserito nella blockchain è dunque immutabile (non può essere corretto)

# EVM



# Nodi

- Un nodo è un'istanza di un client Ethereum che è connesso ad altre istanze che girano su computer differenti, formando una rete
- Dopo The Merge il client Ethereum svolge due compiti:
  - **Execution Engine**, gestisce le nuove transazioni sulla rete, esegue il codice nella EVM, e sincronizza lo stato e i database di tutti i dati Ethereum
  - **Consensus client** (conosciuto anche come Beacon Node), implementa l'algoritmo di consenso proof-of-stake
- Il tutto utilizzando una sola rete (prima del The Merge si utilizzavano due reti diverse)

# Tipi di nodi

- Full node
  - memorizza i dati di tutta la blockchain
  - partecipa alla validazione dei blocchi
  - tutti gli stati possono essere ricostruiti da un nodo full
  - è al servizio della rete, fornisce dati su richiesta
- Light node
  - memorizza soltanto gli header dei blocchi
  - non partecipano al meccanismo di consenso
  - non sono così diffusi come i full node
- Archive node
  - memorizza tutti i dati di un full node compreso lo storico degli stati
  - tipicamente terabyte di dati, non adatti a tutti gli utenti ma a coloro che sono interessati a realizzare servizi come block explorer, wallet e analisi dei dati

# Network

- **Mainnet**, è la rete pubblica di Ethereum, quella dove vengono utilizzati ETH “reali”
- **Testnets**, sono delle reti pubbliche di test, dove gli ETH non hanno un valore economico
  - [Goerli](#), rete di test basata su proof-of-stake
  - [Sepolia](#), rete di test basata su proof-of-stake
  - [Ropsten](#), rete di test deprecata

# Bitcoin VS. Ethereum

	Bitcoin	Ethereum
<b>Creator(s)</b>	Satoshi Nakamoto	Vitalik Buterin, Charles Hoskinson, Gavin Wood, Jeffrey Wilcke, Mihai Alisie, Anthony Di Iorio and Amir Chetrit
<b>Launch date</b>	January 2009	July 2015
<b>Currency vs. platform</b>	A credible alternative to traditional fiat currencies (medium of exchange, store of value)	A platform to run programmatic contracts and applications via Ether
<b>Consensus algorithm</b>	Proof-of-Work	Proof-of-Stake
<b>Block time</b>	10 minutes on average	15 seconds on average
<b>Transaction throughput</b>	7 transactions per second	30 transactions per second
<b>Supply</b>	Finite supply-capped at 21 million BTC	Infinite supply

# Riferimenti

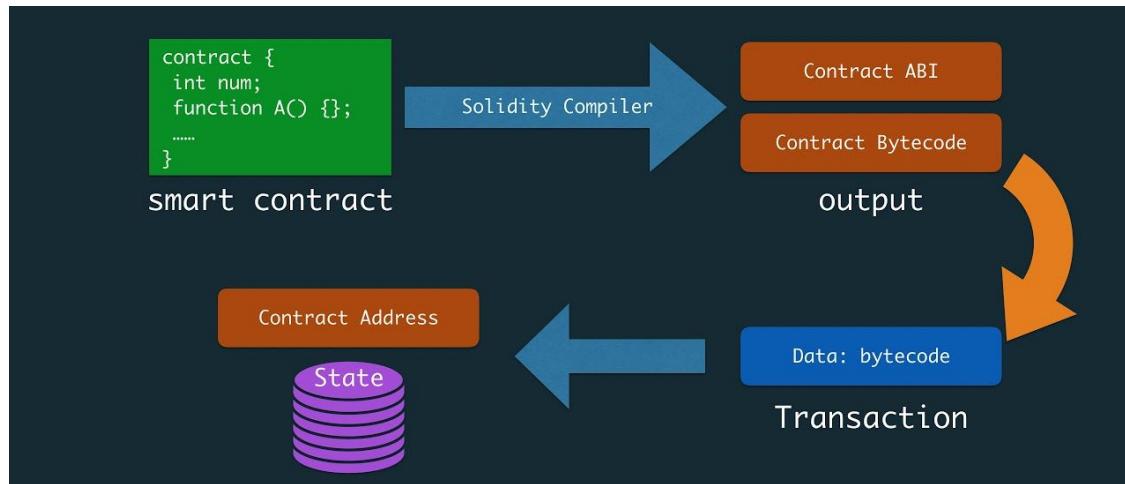
- Andreas Antonopoulos, Gavin Wood, [Mastering Ethereum: Building Smart Contracts and Dapps](#), O'Reilly, 2018
- Binance Academy, [What is Proof of Stake?](#) Youtube video
- Finematics, [DEFI - From Inception To 2021 And Beyond \(History Of Decentralized Finance Explained\)](#), Youtube video
- Laurent Senta, [Where and how application data is stored in Ethereum?](#), Blog, 2021
- David Eisler, [Trie and Patricia Trie Overview](#), Carleton University
- Vasa, [Getting Deep Into Ethereum: How Data Is Stored In Ethereum?](#), HackerNoon, 2018
- Bernard Peh, [Solidity Bytecode and Opcode Basics](#), Medium, 2017

# Solidity

- Solidity è un linguaggio a oggetti con **tipizzazione statica** e una sintassi con **parentesi graffe** (curly-braces) ideato per sviluppare smart contract per Ethereum
- Nato nel 2014 da Gavin Wood
- Quando si sviluppa in Solidity è necessario utilizzare sempre l'ultima versione, poiché soltanto questa riceve security fixes (tranne casi eccezionali).
- Attualmente, l'ultima versione è la 0.8.17

# Solidity e smart contract

- Per Solidity un contratto (smart contract) è una collezione di **codice** (funzioni) e **dati** (stati) che risiedono ad uno specifico indirizzo della blockchain Ethereum



# Esempio: storage

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.16 <0.9.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

variabile di stato  
unsigned integer (256 bits)

# Esempio: coin

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.4;

contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping (address => uint) public balances;

    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);

    // Constructor code is only run when the contract
    // is created
    constructor() {
        minter = msg.sender;
    }

    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }
}
```

```
// Errors allow you to provide information about
// why an operation failed. They are returned
// to the caller of the function.
error InsufficientBalance(uint requested, uint available);

// Sends an amount of existing coins
// from any caller to an address
function send(address receiver, uint amount) public {
    if (amount > balances[msg.sender])
        revert InsufficientBalance({
            requested: amount,
            available: balances[msg.sender]
        });

    balances[msg.sender] -= amount;
    balances[receiver] += amount;
    emit Sent(msg.sender, receiver, amount);
}
```

# Costrutti di base del linguaggio

- In Solidity ci sono i costrutti **if**, **else**, **while**, **do**, **for**, **break**, **continue**, **return** così come in C o Javascript
- Le variabili di stato possono essere:
  - **public**, visibili da altri contratti esterni
  - **internal**, visibili solo all'interno del contratto e da tutti quelli che lo estendono
  - **private**, visibili solo all'interno del contratto e non da contratti derivati
- Anche le funzioni possono essere public, internal o private. Inoltre possono essere anche **external** ossia eseguibili da altri smart contract
- Gestione delle eccezioni con **try/catch** ma solo per funzioni esterne

# Funzioni modifier

- I **modifier** sono delle funzioni speciali di Solidity che consentono di eseguire delle condizioni prima della loro esecuzione
- Tecnicamente modificano il comportamento di una funzione con una modalità dichiarativa

# Esempio di modifier

```
contract owned {  
    constructor() { owner = payable(msg.sender); }  
    address payable owner;  
  
    modifier onlyOwner {  
        require(  
            msg.sender == owner,  
            "Only owner can call this function."  
        );  
        _;  
    }  
}
```

```
contract destructible is owned {  
    function destroy() public onlyOwner {  
        selfdestruct(owner);  
    }  
}
```

body della funzione

# Struct

- E' possibile raggruppare più variabili in una stessa struttura (struct)
- Le **struct** sono presenti in altri linguaggi, es. C

```
contract test {
    struct Book {
        string title;
        string author;
        uint book_id;
    }
    Book book;

    function setBook() public {
        book = Book('Learn Java', 'TP', 1);
    }
    function getBookId() public view returns (uint) {
        return book.book_id;
    }
}
```

# Laboratorio

- Installazione di un wallet, es. MetaMask
- Utilizzo della rete di test Goerli
- Richiesta di ETH su Goerli
- Utilizzo dell'IDE Remix
- Esercizi

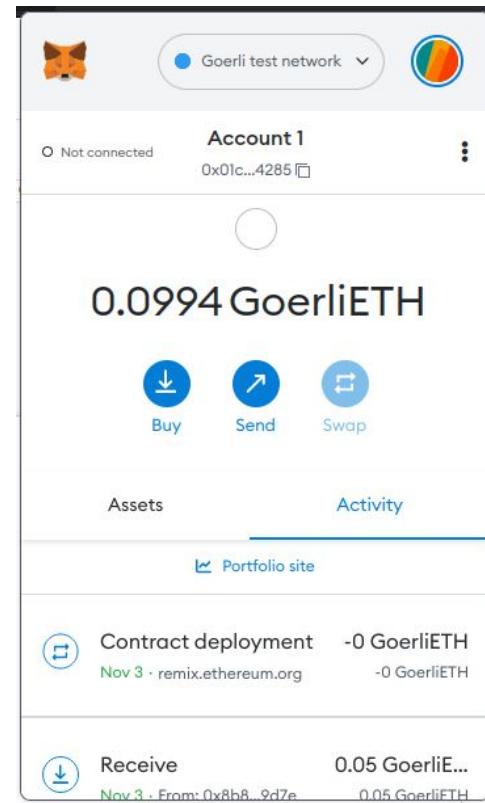
# MetaMask

- **MetaMask** è un wallet che gira come estensione di un browser (eg. Chrome, Firefox, etc) o come app iOS o Android
- Siti web o dapp possono collegarsi, autenticarsi e/o eseguire funzionalità di smart contract utilizzando il wallet di MetaMask
- Installazione su <https://metamask.io/download/>



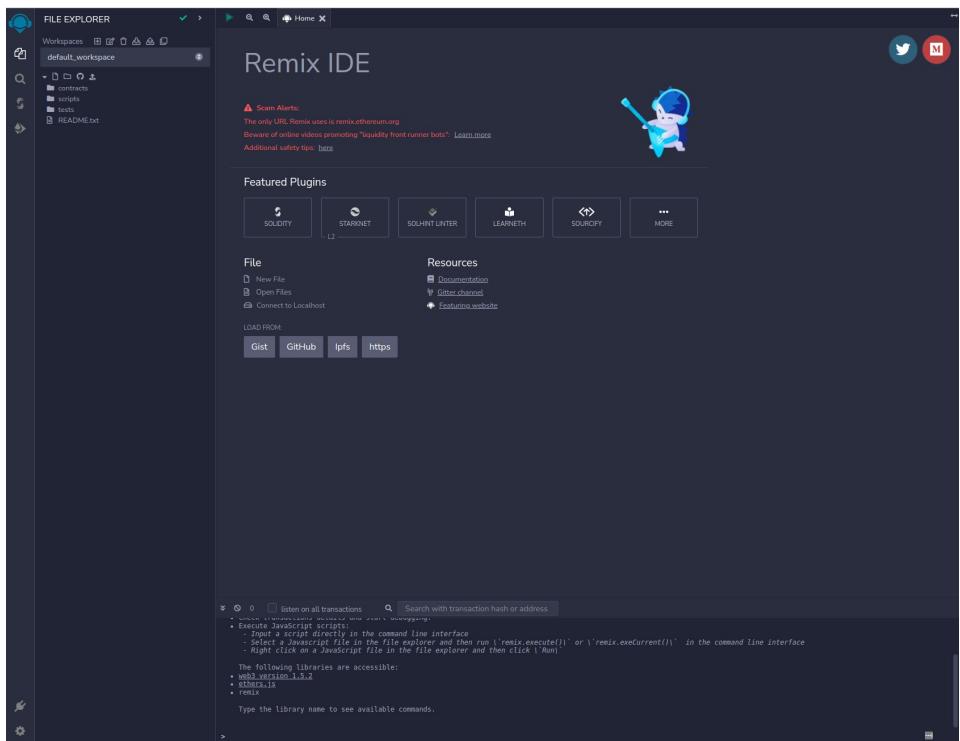
# Test network Goerli

- Per poter eseguire degli smart contract è necessario prima testarli su un network di test
- Esistono diversi network test come Goerli ETH
- E' possibile richiedere ETH di test utilizzando un servizio [Goerli Faucet](#)



# IDE Remix

- Remix è un progetto contenente diversi tool (IDE, CLI, VS code plugin) per lo sviluppo di smart contract su Ethereum
- E' disponibile un IDE online all'indirizzo  
<https://remix.ethereum.org/>

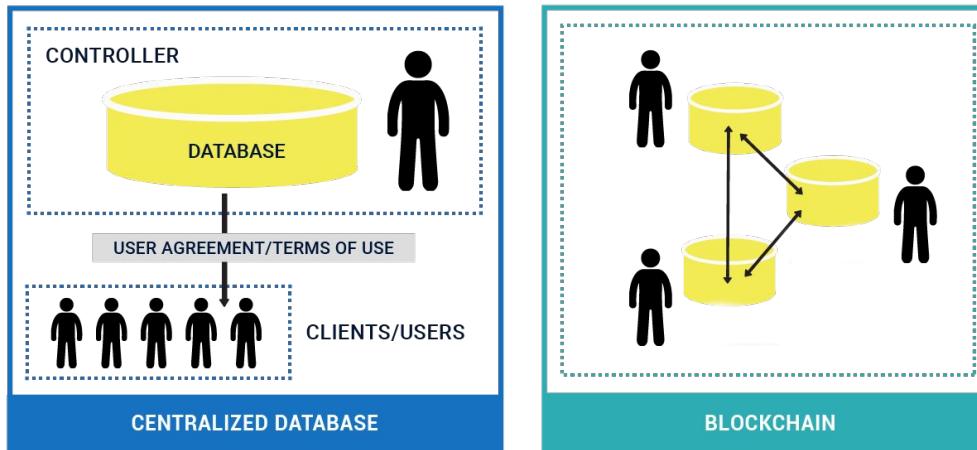


# Riferimenti

- Andreas Antonopoulos, Gavin Wood, [Mastering Ethereum: Building Smart Contracts and Dapps](#), O'Reilly, 2018
- [Solidity](#) documentazione ufficiale
- Kevin Solorio, Randall Kanna, David H. Hoover, [Hands-On Smart Contract Development with Solidity and Ethereum: From Fundamentals to Deployment](#), O'Reilly, 2019
- Ritesh Modi, [Solidity Programming Essentials: A beginner's guide to build smart contracts for Ethereum and blockchain](#), Packt, 2018
- Jitendra Chittoda, [Mastering Blockchain Programming with Solidity](#), Packt, 2019
- [Solidity Tutorial - A Full Course on Ethereum, Blockchain Development, Smart Contracts, and the EVM](#), Youtube Video

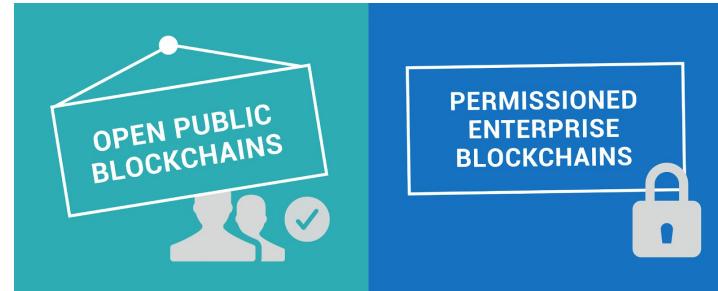
# Database e Blockchain

## CENTRALIZED DATABASES VS. BLOCKCHAIN



# Tipi di blockchain

- **Permissionless** (senza permessi), sono quelle pubbliche  
Bitcoin, Ethereum, Solana, etc
- **Permissioned** (necessitano di permessi), sono quelle private  
utilizzate internamente in una LAN aziendale



# Blockchain in ambito business

Un Distributed Ledger Technology (DLT) in ambito business deve avere le seguenti caratteristiche:

- I partecipanti devono essere identificati/identificabili
- La rete deve essere di tipo **permissioned**
- Supportare un elevato numero di transazioni
- **Bassa latenza** nella conferma di una transazione
- **Sicurezza e privacy** delle transazioni

# Permissioned

- Una blockchain **permissioned** prevede l'utilizzo di un sistema di autenticazione per poter interagire con la blockchain
- Essendo gli utenti tutti autorizzati il rischio di avere malintenzionati è, teoricamente, minore rispetto a una blockchain pubblica
- Le proprietà di una permissioned sono le stesse di una permissionless: registro temporale, immutabilità, condivisione
- Le permissioned offrono **migliori prestazioni** in termini di performance, dal momento che l'algoritmo di consenso è più semplice

# Hyperledger

- Hyperledger è un progetto open source della Linux Foundation, creato nel 2015 per proporre soluzioni basate su blockchain in ambito aziendale
- È la tecnologia blockchain più utilizzata in ambito business
- Aziende membri del progetto: Accenture, IBM, JP Morgan, Hitachi, Fujitsu, etc
- Le tecnologie blockchain utilizzate in Hyperledger sono principalmente **permissioned**



**HYPERLEDGER**  
FOUNDATION

# Obiettivi di Hyperledger



Create enterprise grade, open source, distributed ledger frameworks and code bases to support business transactions



Provide neutral, open, and community-driven infrastructure supported by technical and business governance



Build technical communities to develop blockchain and shared ledger POCs, use cases, field trials and deployments



Educate the public about the market opportunity for blockchain technology



Promote our community of communities taking a toolkit approach with many platforms and frameworks

# Progetti Hyperledger

Graduated Hyperledger Projects (6)

 <b>HYPERLEDGER ARIES</b> Hyperledger Aries Hyperledger	 <b>HYPERLEDGER BESU</b> Hyperledger Besu Hyperledger	 <b>HYPERLEDGER FABRIC</b> Hyperledger Fabric Hyperledger	 <b>HYPERLEDGER INDY</b> Hyperledger Indy Hyperledger	 <b>HYPERLEDGER IROHA</b> Hyperledger Iroha Hyperledger	 <b>HYPERLEDGER SAWTOOTH</b> Hyperledger Sawtooth Hyperledger
--	--	--	---	--	--

Incubating Hyperledger Projects (8)

 <b>HYPERLEDGER BEVEL</b> Hyperledger Bevel Hyperledger	 <b>HYPERLEDGER CACTUS</b> Hyperledger Cactus Hyperledger	 <b>HYPERLEDGER CALIPER</b> Hyperledger Caliper Hyperledger	 <b>HYPERLEDGER CELLO</b> Hyperledger Cello Hyperledger	 <b>HYPERLEDGER FIREFLY</b> Hyperledger Firefly Hyperledger	 <b>HYPERLEDGER GRID</b> Hyperledger Grid Hyperledger
 <b>HYPERLEDGER TRANSACT</b> Hyperledger Transact Hyperledger	 <b>HYPERLEDGER URSA</b> Hyperledger Ursa Hyperledger				

# Fabric

- Hyperledger Fabric è un framework blockchain modulare diventato uno standard de facto per le piattaforme blockchain aziendali
- È un distributed ledger permissioned scritto in Go
- Sviluppato appositamente per soluzioni aziendali
- Nato nel 2016, ha interessato più di 35 aziende con più di 200 sviluppatori
- Ultima release stabile: 2.4.4 (Giugno 2022)



# Pluggable consensus protocol

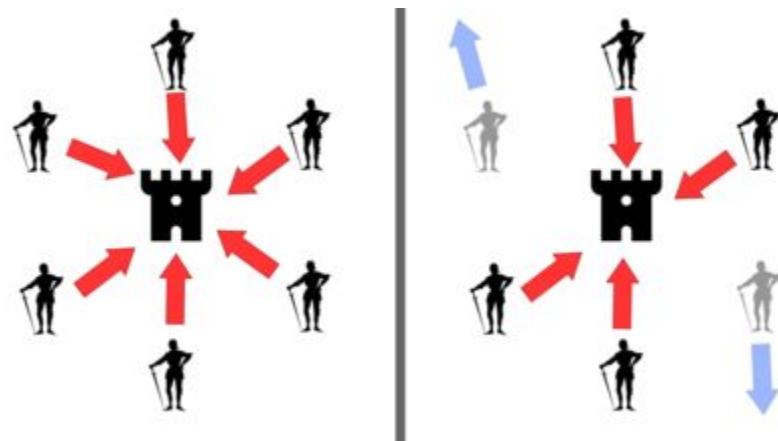
- Una delle caratteristiche più interessanti di Fabric è la capacità di poter configurare l'algoritmo di consenso per la scrittura nella blockchain
- Ad esempio, nel caso di utilizzo di un registro distribuito in un'unica realtà aziendale non ha molto senso utilizzare un algoritmo **Byzantine fault-tolerant** (BFT) ma è meglio utilizzare un algoritmo come **Crash fault-tolerant** (CFT) che offre prestazioni migliori (tempi di scrittura più rapidi)

# Problema dei generali bizantini

- Tre o più generali bizantini devono decidere se attaccare o ritirarsi dato un ordine da un comandante superiore. Uno o più dei generali potrebbero essere dei traditori con l'intenzione di confondere gli altri, quindi potrebbe verificarsi il caso in cui il comandante dia ordini discordanti ai generali oppure il caso in cui uno dei generali comunichi ai propri colleghi un ordine differente da quello impartito dal comandante
- In informatica il problema è legato al raggiungimento di un **consenso in presenza di errori**. Il problema consiste nel trovare un accordo, comunicando solo tramite messaggi, tra componenti diversi nel caso in cui siano presenti informazioni discordanti

# Problema dei generali bizantini

- Si può dimostrare che non esiste soluzione al problema se il numero di processi non corretti è maggiore o uguale a un terzo del numero totale di processi



# Smart contract (chaincode)

- Hyperledger Fabric è la prima piattaforma DLT a supportare smart contract (detti **chaincode**) scritti in linguaggi general-purpose come Java, Go e Node.js
- I chaincode in Fabric utilizzano il principio **execute-order-validate** al posto del principio **order-execute** utilizzato dalla maggior parte delle blockchain come Ethereum, Tendermint, etc.
- La validazione dello smart contract non viene effettuata verificando che l'output dell'esecuzione su tutti i nodi sia lo stesso. Quindi è possibile che i programmi siano anche non deterministici.

# Order-execute

L'architettura **order-execute** prevede di:

1. validare e ordinare le transazioni;
2. ogni peer esegue le transazioni in maniera sequenziale

Per arrivare ad un consenso il risultato dell'esecuzione dello smart contract deve essere lo stesso per ogni peer, quindi deve essere deterministica (questo è il motivo per il quale si utilizzano linguaggi DSL (eg. Solidity)

# Execute-order-validate

L'architettura **execute-order-execute** prevede invece di:

1. eseguire una transazione e verificare la sua correttezza (avallandola);
2. ordinare le transazioni utilizzando un algoritmo di consenso (configurabile);
3. validare le transazioni secondo una policy specifica dell'applicazione prima di inserirla nel registro distribuito

# Hyperledger Fabric Model

- **Assets**, gli oggetti, le entità che si vogliono gestire nella blockchain
- **Chaincode**, gli smart contract
- **Ledger**, il registro distribuito con la possibilità di eseguire interrogazioni SQL-like
- **Privacy**, il sistema di protezione dei dati
- **Security & Membership**, il servizio di autenticazione
- **Consensus**, gli algoritmi di consenso configurabili (pluggable)

# Esempio: eseguire un chaincode in Node.js

```
// Connect to a gateway peer
const connectionProfileJson = (await fs.promises.readFile(connectionProfileFileName)).toString();
const connectionProfile = JSON.parse(connectionProfileJson);
const wallet = await Wallets.newFileSystemWallet(walletDirectoryPath);
const gatewayOptions: GatewayOptions = {
    identity: 'user@example.org', // Previously imported identity
    wallet,
};
const gateway = new Gateway();
await gateway.connect(connectionProfile, gatewayOptions);

try {

    // Obtain the smart contract with which our application wants to interact
    const network = await gateway.getNetwork(channelName);
    const contract = network.getContract(chaincodeId);

    // Submit transactions for the smart contract
    const args = [arg1, arg2];
    const submitResult = await contract.submitTransaction('transactionName', ...args);

    // Evaluate queries for the smart contract
    const evalResult = await contract.evaluateTransaction('transactionName', ...args);

    // Create and submit transactions for the smart contract with transient data
    const transientResult = await contract.createTransaction(transactionName)
        .setTransient(privateData)
        .submit(arg1, arg2);

} finally {
    // Disconnect from the gateway peer when all work for this client identity is complete
    gateway.disconnect();
}
```

# Esempio: Contract in Node.js

```
// updatevalues.js
'use strict';

// SDK Library to asset with writing the logic
const { Contract } = require('fabric-contract-api');

// Business logic (well just util but still it's general purpose logic)
const util = require('util');

/**
 * Support the Updating of values within the SmartContract
 */
class UpdateValuesContract extends Contract

    constructor(){
        super('UpdateValuesContract');
    }

    async transactionA(ctx, newValue) {
        // retrieve existing chaincode states
        let oldValue = await ctx.stub.getState(key);

        await ctx.stub.putState(key, Buffer.from(newValue));

        return Buffer.from(newValue.toString());
    }

    async transactionB(ctx) {
        // ....
    }
};

module.exports = UpdateValuesContract
```

# Esempio: Chaincode in Node.js

```
const shim = require('fabric-shim');

const Chaincode = class {
    async Init(stub) {
        // use the instantiate input arguments to decide initial chaincode state values

        // save the initial states
        await stub.putState(key, Buffer.from(aStringValue));

        return shim.success(Buffer.from('Initialized Successfully!'));
    }

    async Invoke(stub) {
        // use the invoke input arguments to decide intended changes

        // retrieve existing chaincode states
        let oldValue = await stub.getState(key);

        // calculate new state values and saves them
        let newValue = oldValue + delta;
        await stub.putState(key, Buffer.from(newValue));

        return shim.success(Buffer.from(newValue.toString()));
    }
};
```

# Riferimenti

- Elena Mesropyan, [22 Companies Leveraging Blockchain for Identity Management and Authentication](#), 2017
- Daniel Palmer, [7 Cool Decentralized Apps Being Built on Ethereum](#), 2016
- Chris Berg, Sinclair Davidson and Jason Potts, [The Blockchain Economy: A beginner's guide to institutional cryptoeconomics](#), 2017
- [Hyperledger Fabric: Open, Proven, Enterprise-grade DLT](#), IBM white paper
- L.Lamport, R. Shostak, M. Pease, [The Byzantine Generals Problem](#), ACM Transactions on Programming Languages and Systems, Vol.4, No. 3, July 1982
- Tapasweni Pathak, [Hyperledger Fabric: Develop permissioned blockchain smart contracts](#), 2022
- [Hyperledger Architecture, Volume 1](#), Hyperledger foundation
- John Tucker, [Hyperledger Fabric By Example](#), 2019