

Il protocollo Bitcoin

Corso Fintech Software Developer
Modulo **Tecnologie Blockchain**

Docente: Dott. Enrico Zimuel

in collaborazione con:



REGIONE
PIEMONTE

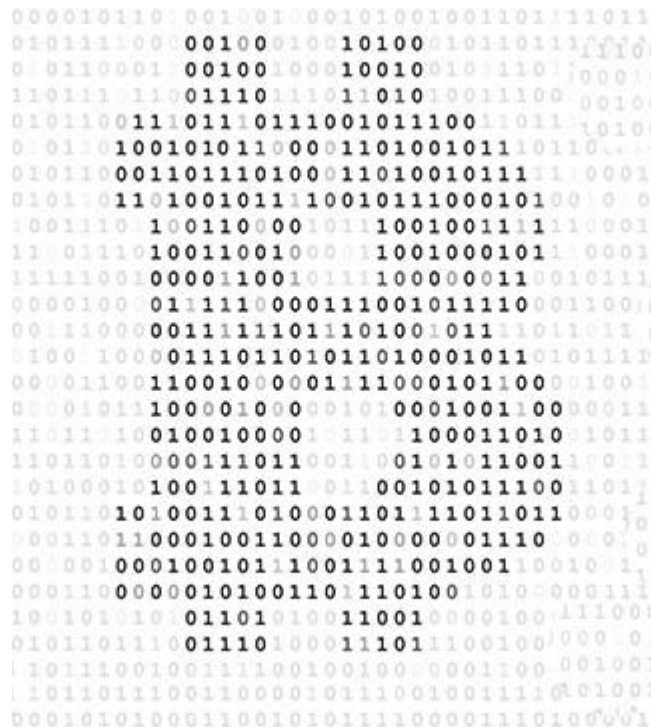
per una crescita intelligente,
sostenibile ed inclusiva

www.regione.piemonte.it/europa2020

INIZIATIVA CO-FINANZIATA CON FSE

Sommario

- Il protocollo Bitcoin
- Transazioni
- Bitcoin Script
- Alcune applicazioni:
 - Deposito a garanzia
 - Abbonamento
 - Pagamenti posticipati
- Blocchi, coinbase
- Limiti del protocollo
- Hard/Soft Fork




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 ₿ → Alice]	
2	IN: 1 [0] OUT: 17 ₿ → Bob , 8 → Alice	
3	IN: 2 [0] OUT: 8 ₿ → Carol , 9 → Bob	
4	IN: 3 [0] OUT: 5 ₿ → Alice , 3 → Carol	
5	IN: 4 [0], 2 [1] OUT: 15 ₿ → David , -2 → Alice	

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

m	IN: x [y] OUT: 8฿ → Carol
k	IN: z [j] OUT: 10฿ → Bob
n	IN: m [0], k [0] OUT: 10฿ → Alice , 5฿ → Bob , 3฿ → Carol



Firme digitali:



Alice



Bob



Carol

Dettaglio di una transazione

Metadati

Input

Output

```
{
  "hash": "b6f6991d03df0e2e04daffcd6bc418aac66049e2cd74b80f14ac86db1e3f0da",
  "ver": 1,
  "vin_sz": 1,
  "vout_sz": 2,
  "lock_time": "0",
  "size": 258,
  . . .
  "in": [
    {
      "prev_out": {
        "hash": "a3e2bcc9a5f776112497a32b05f4b9e5b2405ed9",
        "n": "2"
      },
      "scriptSig": "76a914641ad5051edd97029a003fe9efb29359fcee409d88ac"
    }
  ],
  "out": [
    {
      "value": "9.8000000",
      "scriptPubKey": "76a914641ad5051edd97029a003fe9efb29359fcee409d88ac"
    },
    {
      "value": "1.2000000",
      "scriptPubKey": "76a914641ad5051edd97029a003fe9efb29359fcee409d88ac"
    }
  ]
}
```

Medati, Input

- Un blocco ha una dimensione di **1 MB** (≈ 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

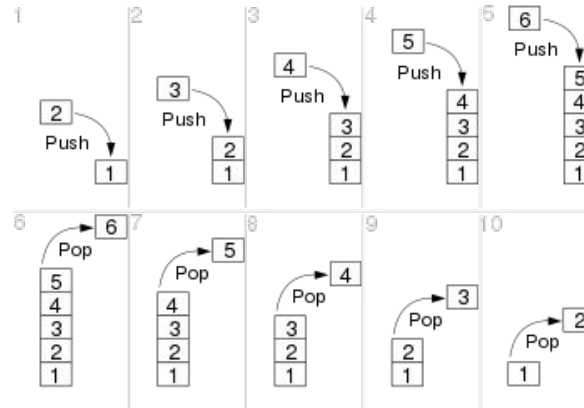
- Output:
 - valore della transazione
 - $\sum \text{output} \leq \sum \text{input}$
 - se la somma dell'output è minore dell'input la differenza è detta **transaction fee**, che viene assegnata al miner che pubblica la transazione
 - **scriptPubKey** = le operazioni da eseguire sull'output

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 Bitcoin 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 **CHECKMULTISIG** 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

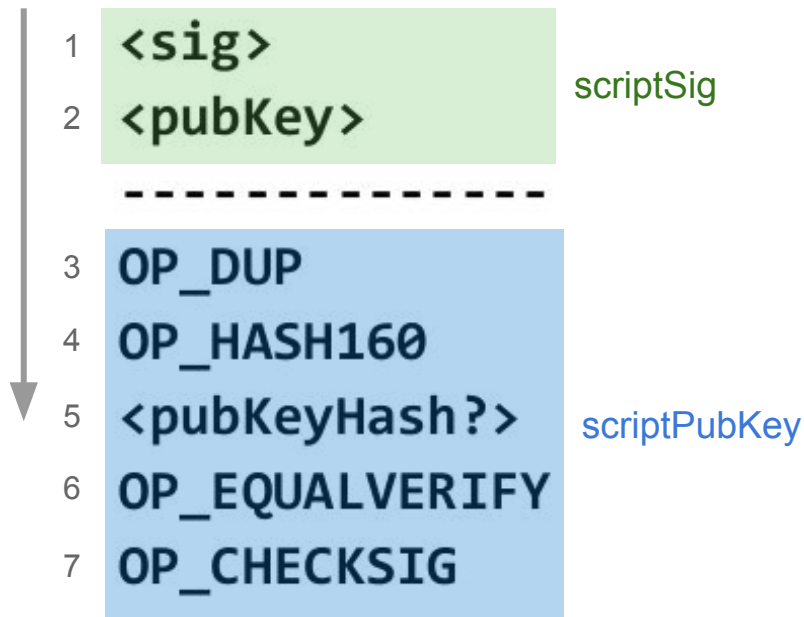
Alcune istruzioni comuni del linguaggio Script

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

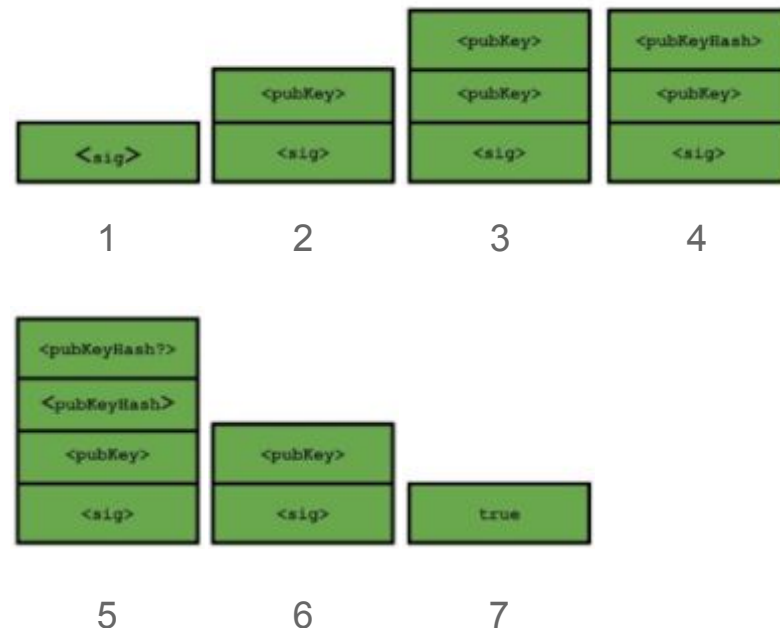
Esecuzione di uno Script

- 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



Stack di esecuzione:



Esercizio

- Utilizzare l'ambiente di sviluppo per Bitcoin script riportato a questo indirizzo <https://siminchen.github.io/bitcoinIDE/build/editor.html>
- Realizzare i seguenti script:
 - Dato un raggio r , calcolare l'area di un cerchio
 - Scrivere un ScriptPubKey per una transazione che può essere riscossa da qualcuno che è in grado di calcolare la radice quadrata di 1764

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)

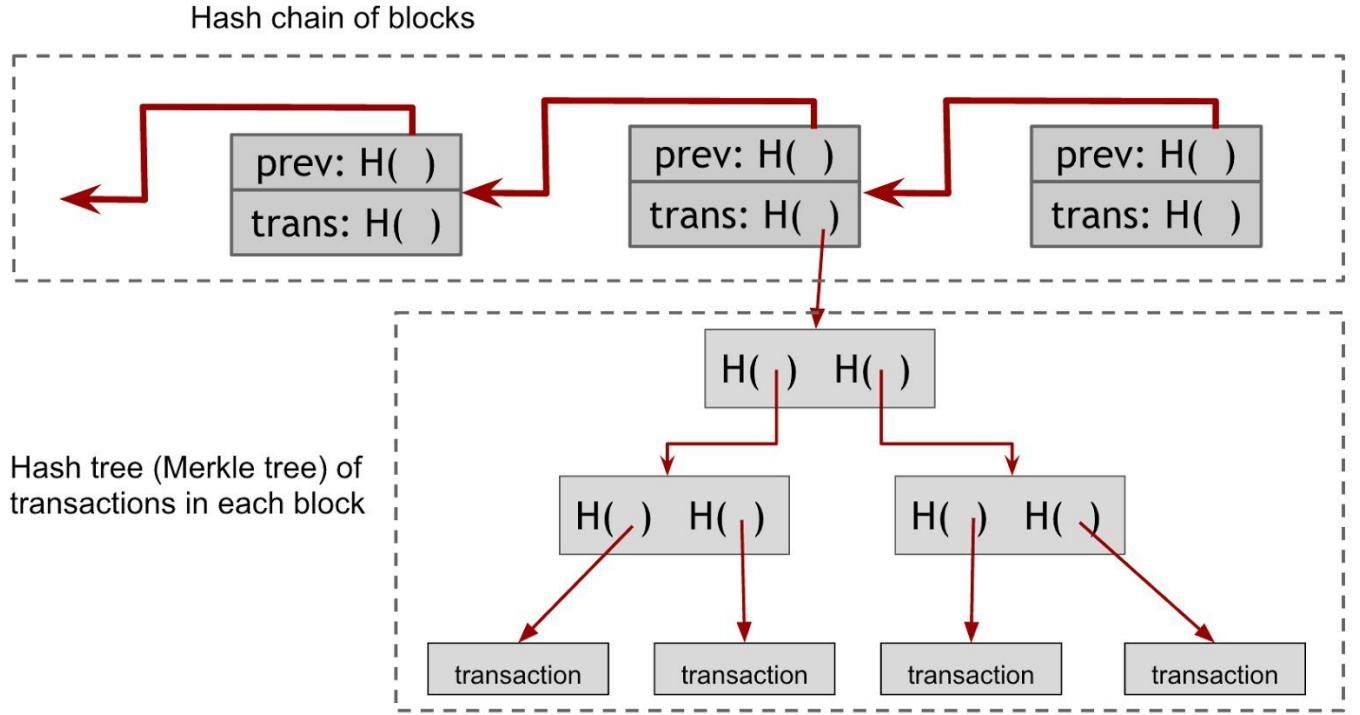
Deposito a garanzia (2)

- 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 ₿** (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)

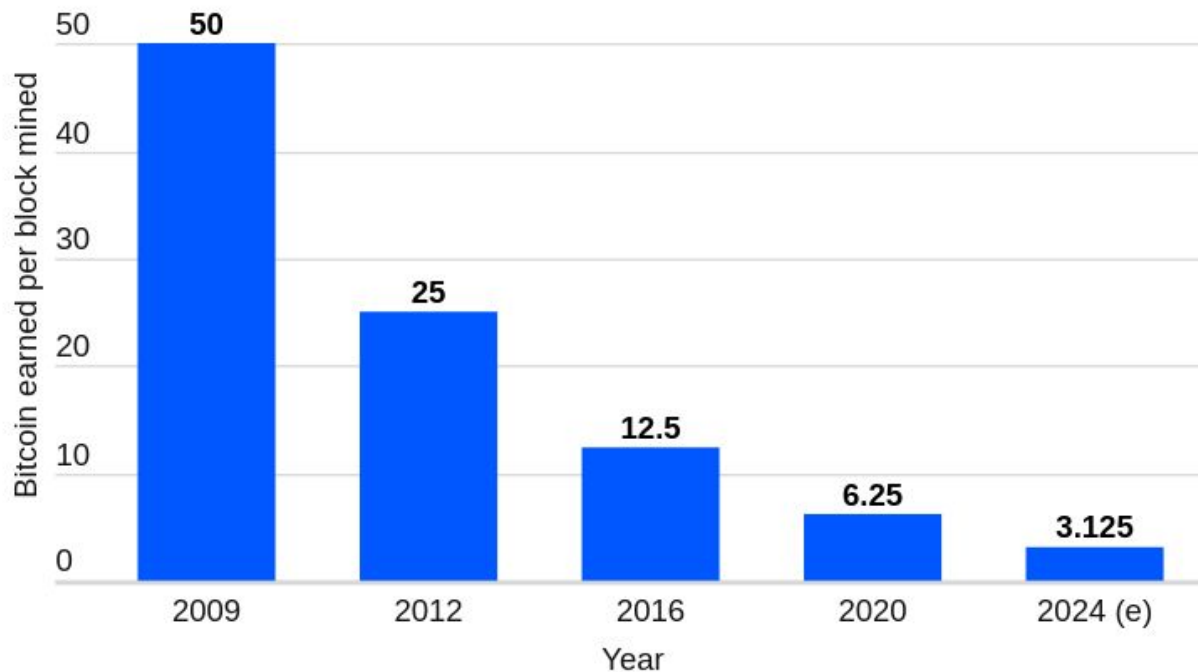
Blocco genesis

- Il campo coinbase del blocco genesis, 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



Esercizio

- Tramite uno dei vari **Bitcoin explorer** online (es. blockchain.com) cercare la prima transazione **coinbase** della storia
- Negli blocchi Bitcoin più recenti, verificare l'ammontare delle commissioni (Fees Reward) e della ricompensa (Block Reward)
- In media, quante transazioni ci sono in ogni blocco?

La rete Bitcoin

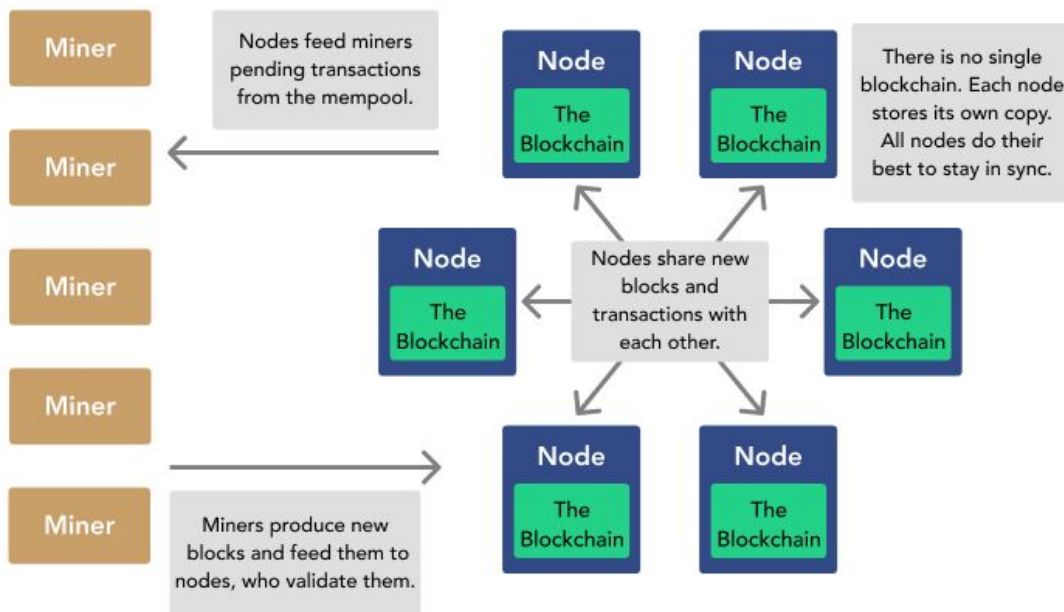
- E' una rete decentralizzata di tipo **peer-to-peer**
- Tutti i nodi sono equivalenti, non c'è una gerarchia
- La topologia della rete è casuale, nel senso che un nodo si collega agli altri in modo casuale
- Quando un nodo si aggancia alla rete manda un messaggio ad un seed node che conosce già. Questo 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

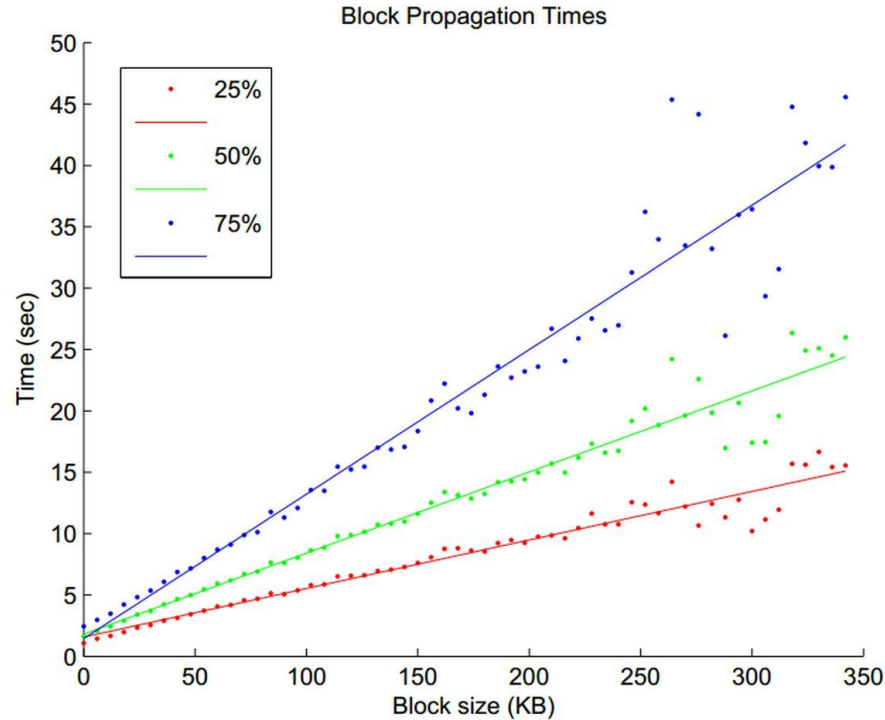
- Alice vuole inviare x ₿ a Bob
- Il client di Alice invia la transazione verso un nodo della rete
- Questo client memorizza la transazione in un memoria temporanea (**mempool**) e propaga la transazione verso gli altri nodi; se un nodo ha già la transazione nel suo pool non viene più propagata verso 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 memorizzata 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

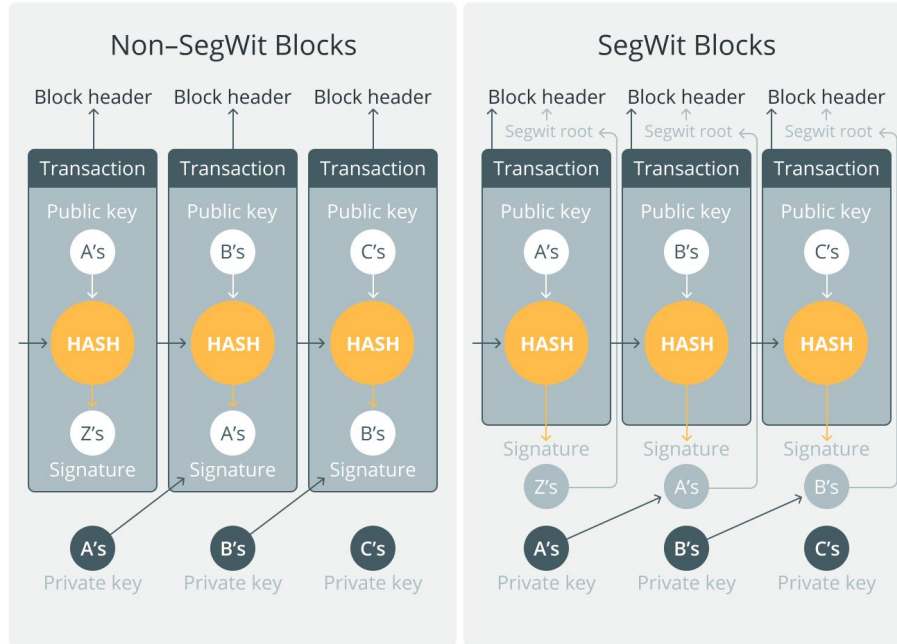
Come evolvere la rete Bitcoin

- **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

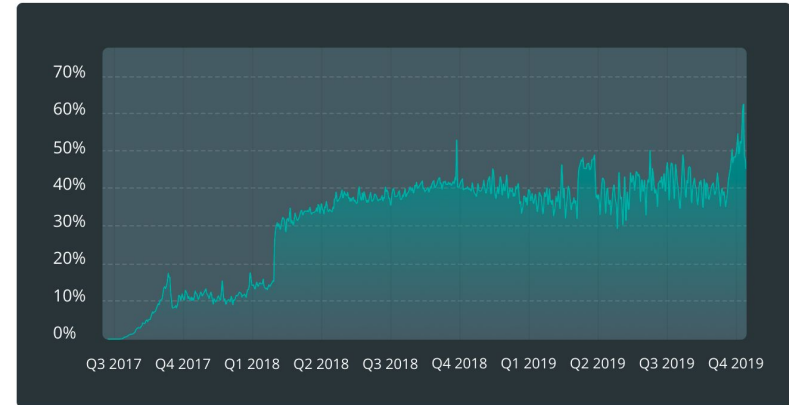
Storia dei fork di Bitcoin

- **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



Esercizio

- Leggere l'articolo [Dissecting a Bitcoin Script](#) e studiare lo script Bitcoin presentato
- Rispondere alla seguente domanda: come hanno fatto a far restituire 1 (true) allo script riportato nell'articolo?

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

Grazie dell'attenzione!

Per informazioni:

enrico.zimuel@its-ictpiemonte.it