

Mirko Zichichi

Smart contracts

Outline

- Introduzione ai Sistemi Distribuiti e Blockchain
- Introduzione agli Smart Contracts
- Introduzione alla programmazione in Solidity
 - Cryptozombies

Concetti di base necessari

Cryptographic **Hash** function

SHA256

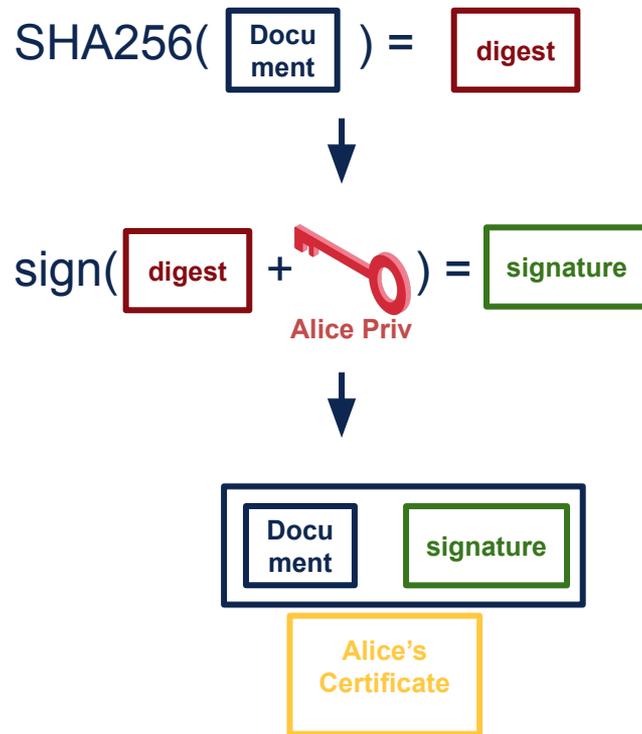
$$f(\textit{string}) = \textit{digest}$$

$$SHA256(\textit{moat}) = 98E2\dots16F3$$

$$SHA256(\textit{m\textcircled{a}ot}) = E671\dots A9C9$$


la lunghezza del digest è
sempre la stessa
(256 bit per SHA256)

Firma Digitale

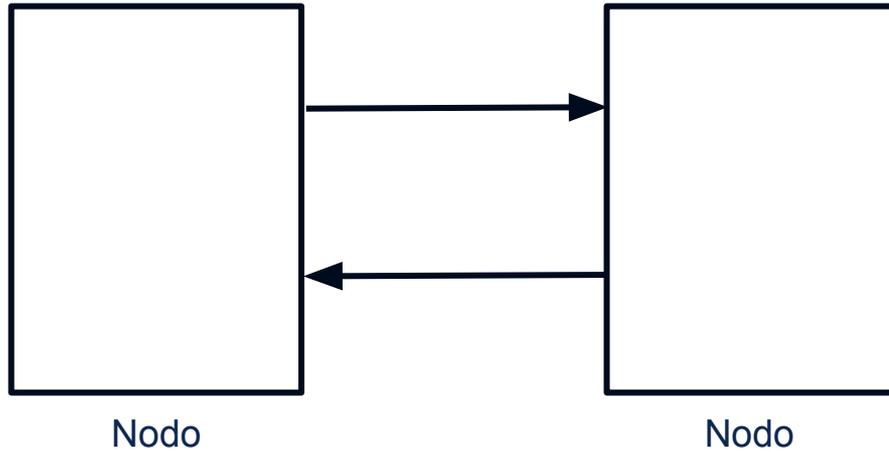


Introduzione ai Sistemi Distribuiti e Blockchain

+ Sistema Distribuito

Sistema informatico costituito da un insieme di **processi** interconnessi tra loro in cui le comunicazioni avvengono solo esclusivamente tramite lo scambio di opportuni messaggi.

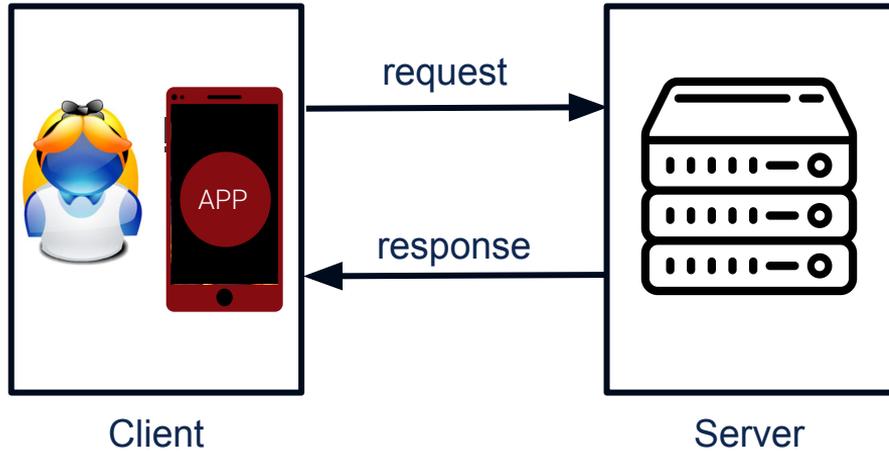
Nodo



Qualsiasi **dispositivo** hardware del sistema in grado di **comunicare** con gli altri dispositivi che fanno parte della rete

Dispongono di una memoria propria, di un proprio sistema operativo e di risorse locali

Architettura Client/Server

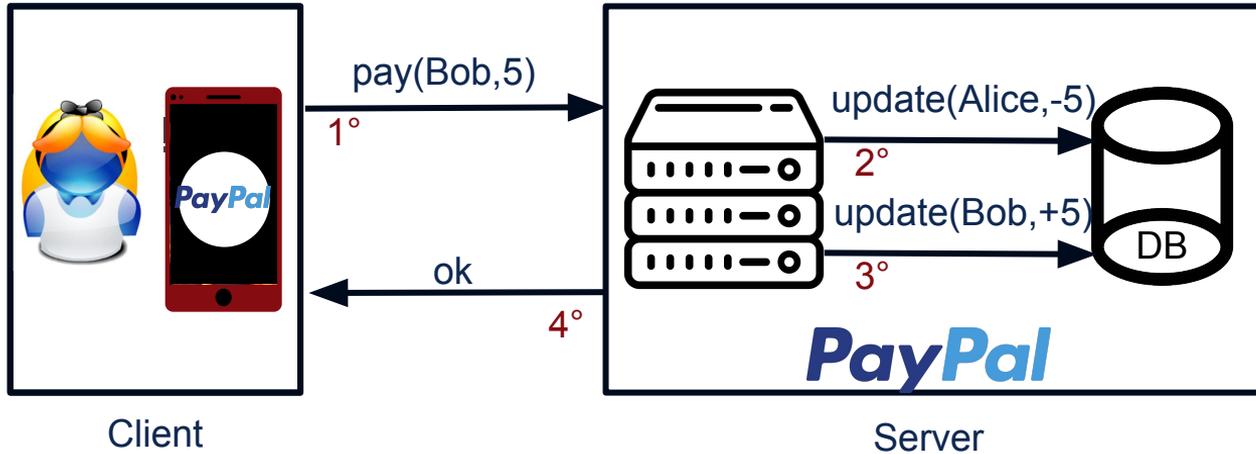


Un'architettura di sistema è il **modello concettuale** che definisce la struttura, il comportamento e più prospettive di un unico sistema

Architettura Client/Server

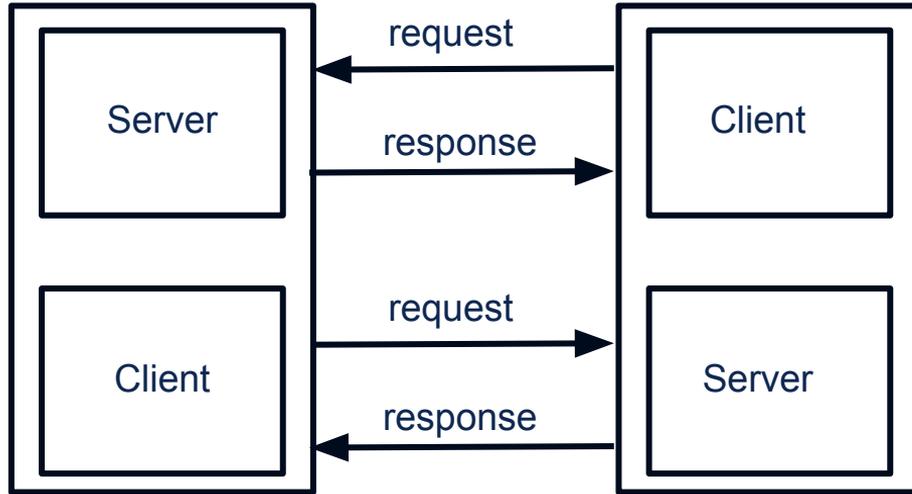
Esempio

Alice
paga
Bob
5 euro



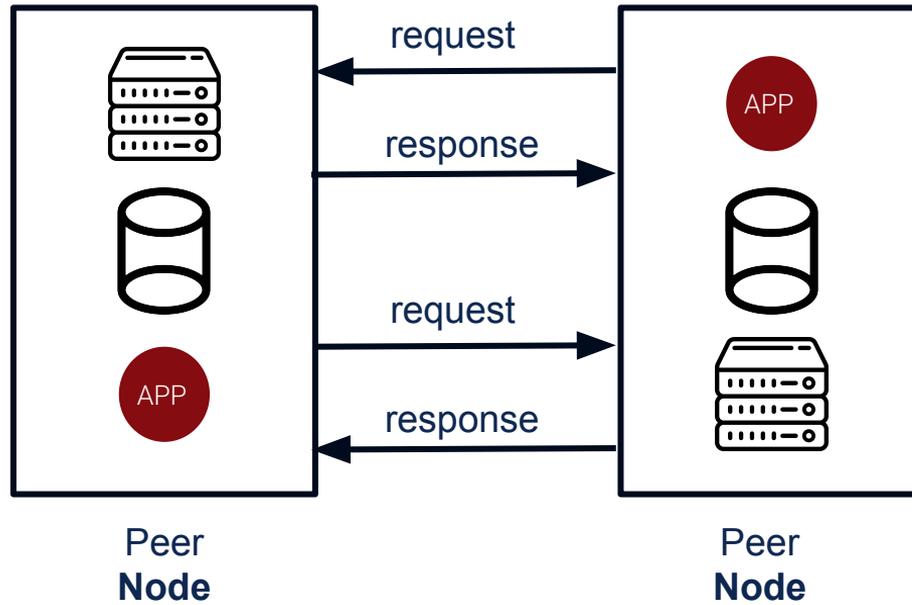
Architettura Client/Server

I nodi Peers sono
Client e Server
simultaneamente



Architettura Peer to Peer (P2P)

I nodi Peers sono
Client e Server
simultaneamente



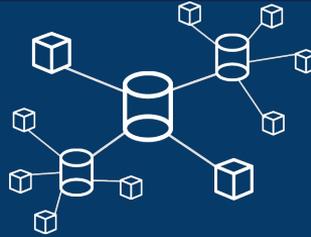
+ Sistema Distribuito

Sistema informatico costituito da un insieme di **processi** interconnessi tra loro in cui le comunicazioni avvengono solo esclusivamente tramite lo scambio di opportuni messaggi.



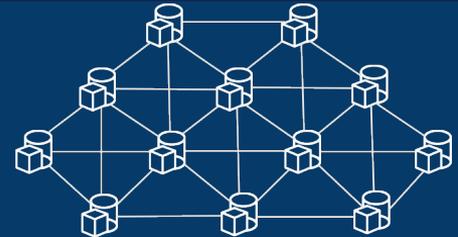
Centralizzato

Un nodo si occupa di tutto

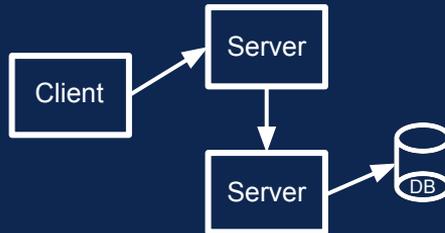


De-centralizzato

Il carico totale dei compiti viene decentralizzato e coinvolge i sotto-nodi



De-centralizzato Distribuito (P2P)





BLOCKCHAIN

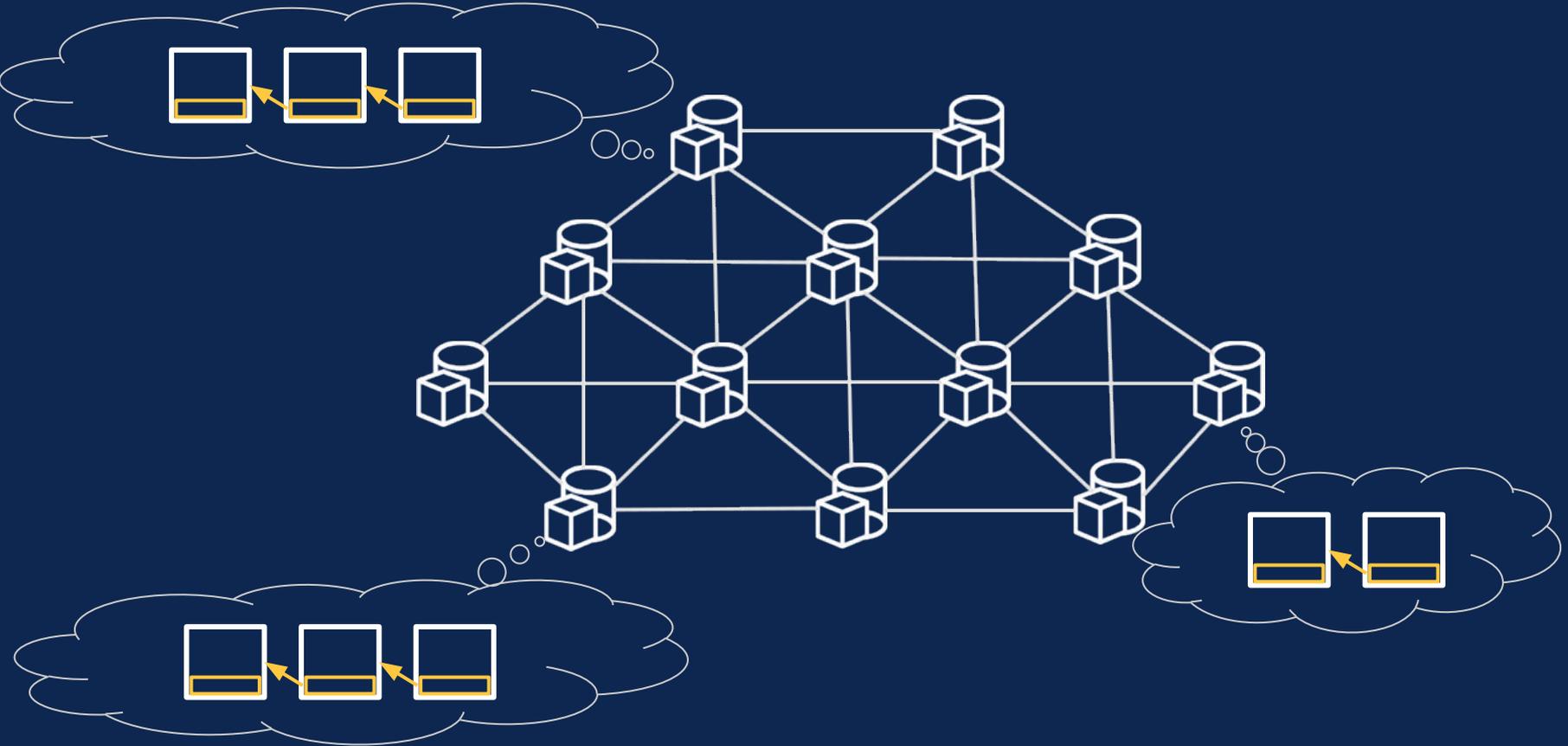
Sistema Distribuito basato su una rete di nodi P2P

- È una tecnologia che fa parte del regno delle DLTs:
Distributed Ledger Technologies
- Nelle DLTs un registro viene distribuito tra i nodi di una rete P2P, che aggiornano la loro **copia locale** secondo un unico meccanismo di consenso
- Una blockchain è una DLT in cui il registro assume la forma di un **insieme di blocchi di dati (relativamente) ordinati cronologicamente**

+

BLOCKCHAIN

Sistema Distribuito basato su una rete di nodi P2P



+ Blockchain

- Cosa scrivere sul registro → **transazioni**

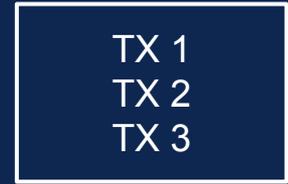


- Struttura del registro → **chain of blocks**



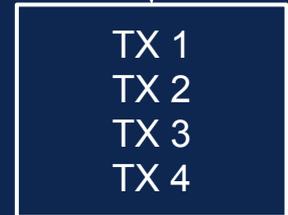
+ Transazioni

- Se il registro mantiene lo stato del sistema → allora una transazione è l'operazione che modifica questo stato
- Lo stato del sistema in un certo momento (snapshot) è un elenco di transazioni
- Una nuova transazione si riferisce ad una precedente e aggiorna lo stato del sistema
- Una transazione valida viene firmata utilizzando la firma digitale dell'account a cui fa riferimento la transazione precedente

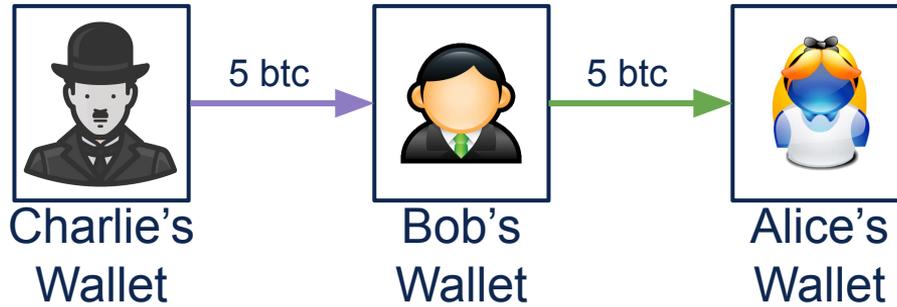


+

TX 4



Transazioni: Esempio



TX 1 : 5 btc |--> Charlie
TX 2 : Charlie |--> Bob
TX 3 : Bob |--> Alice

+ Blockchain

- Cosa scrivere sul registro → **transazioni**

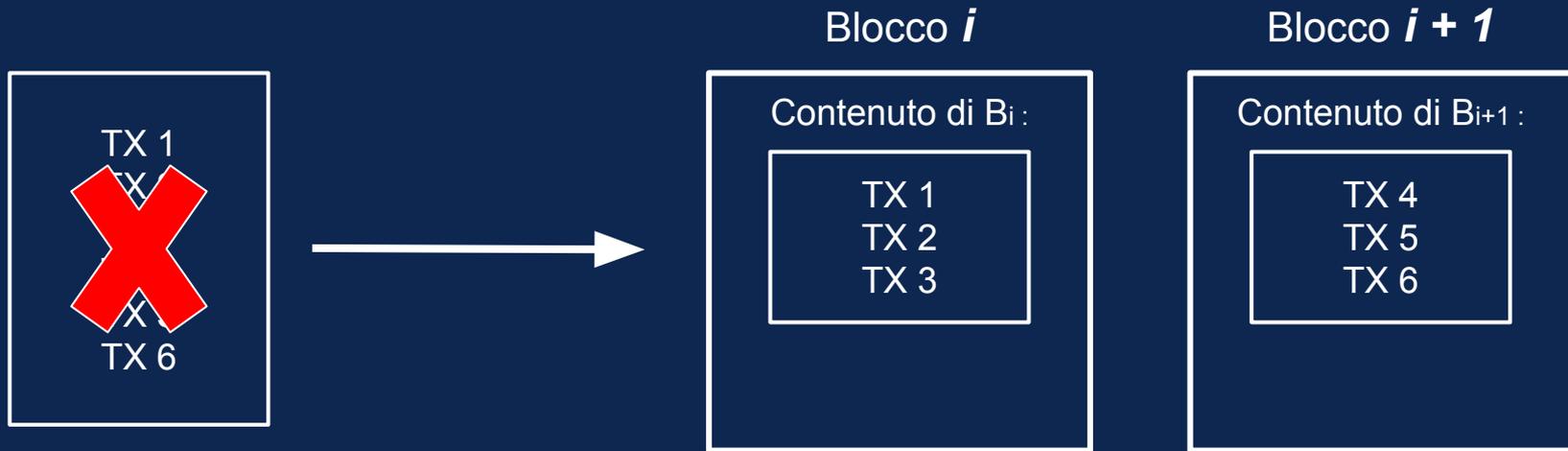


- Struttura del registro → **chain of blocks**



+ Registro a Blocchi

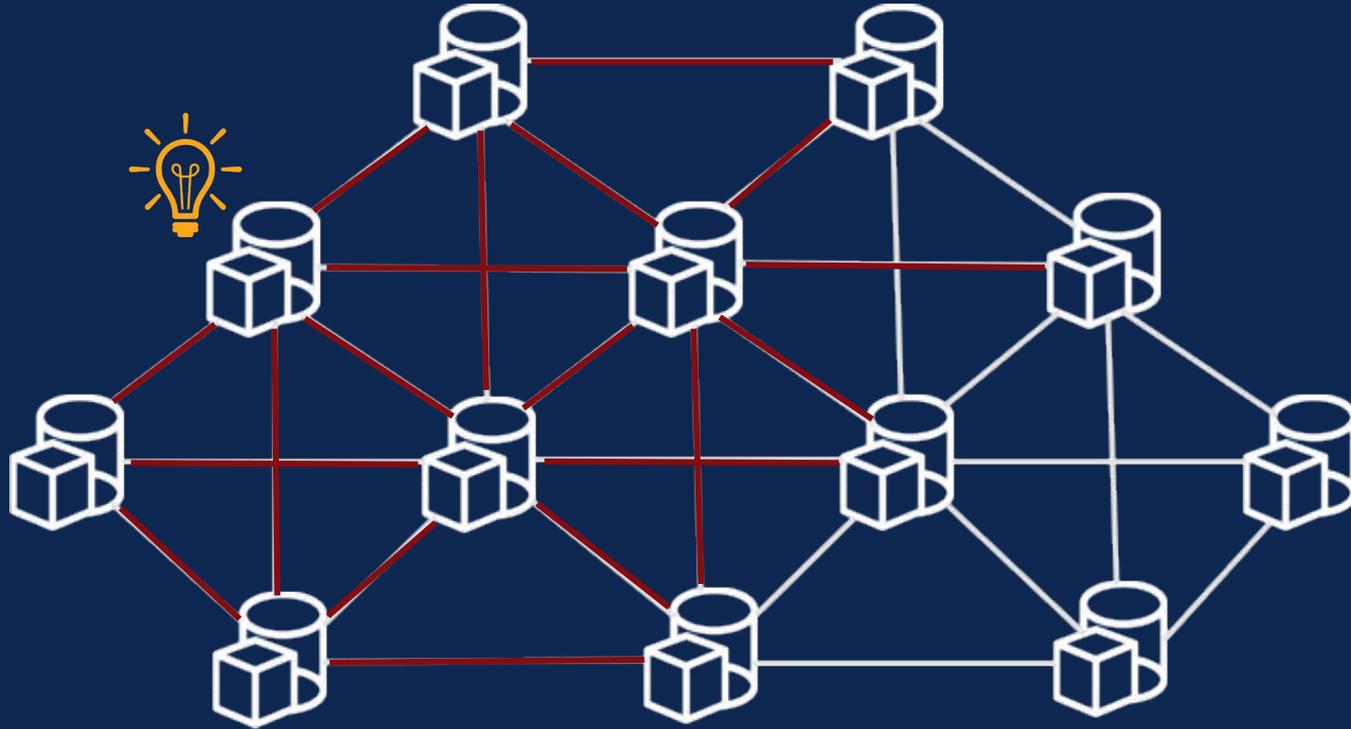
invece di avere un unico documento contenente tutto il registro di transazioni, la blockchain lo divide in BLOCCHI:



+ Registro distribuito: creazione di un nuovo blocco

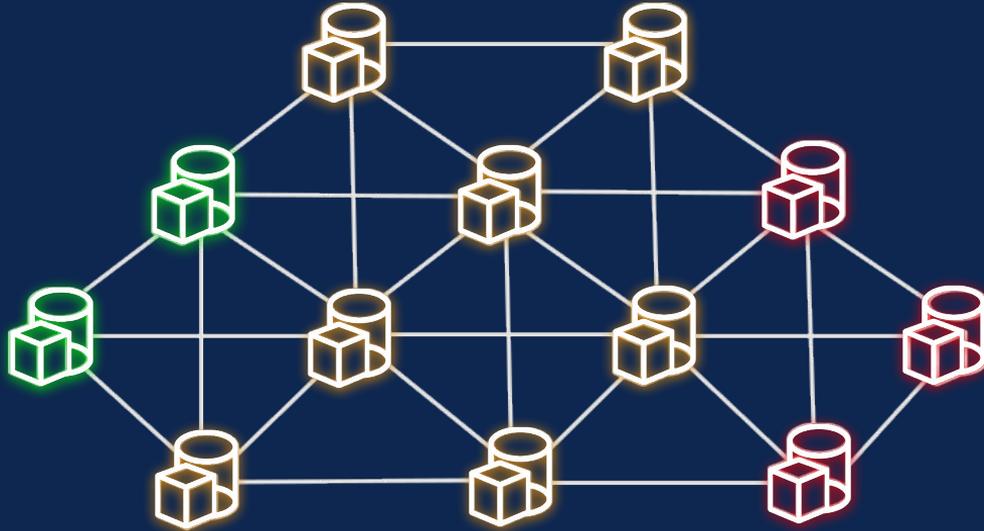
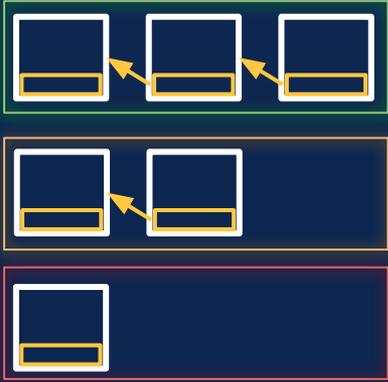


+ Registro distribuito: propagazione di un blocco



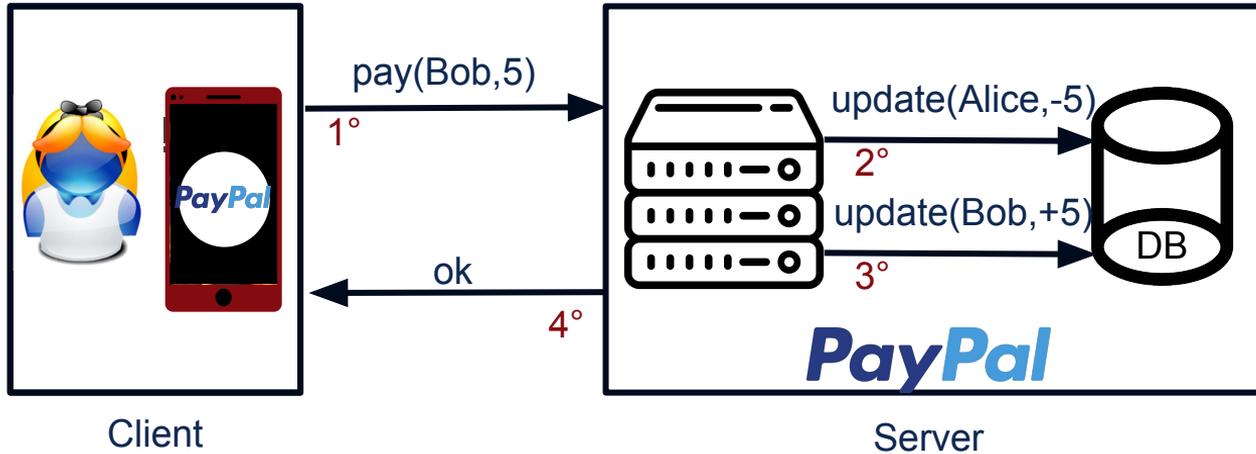
Registro distribuito: sincronizzazione

+



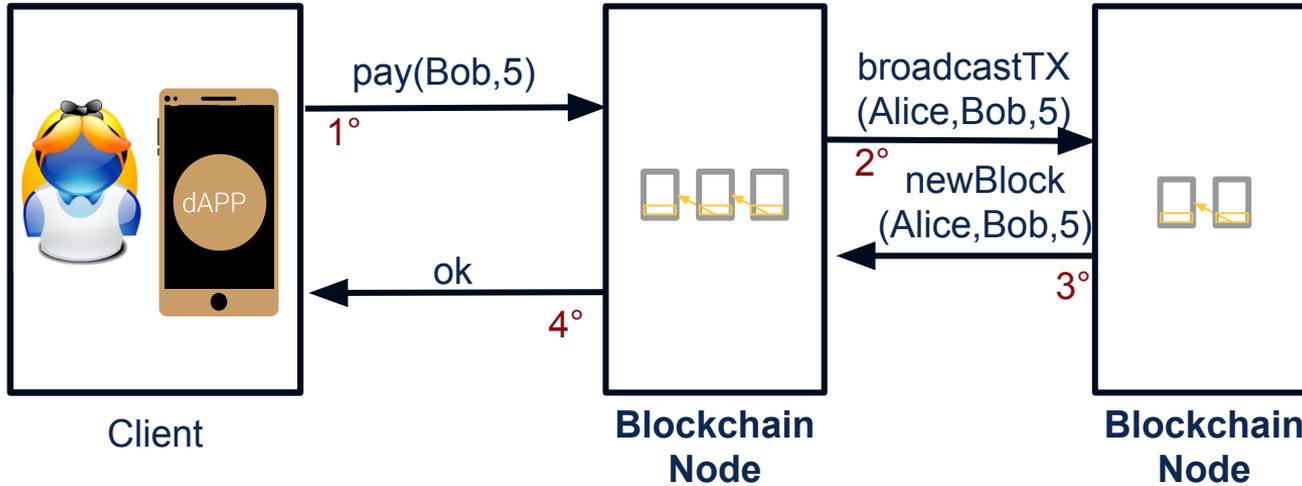
Comparazione Blockchain e Client/Server

Alice
paga
Bob
5 euro

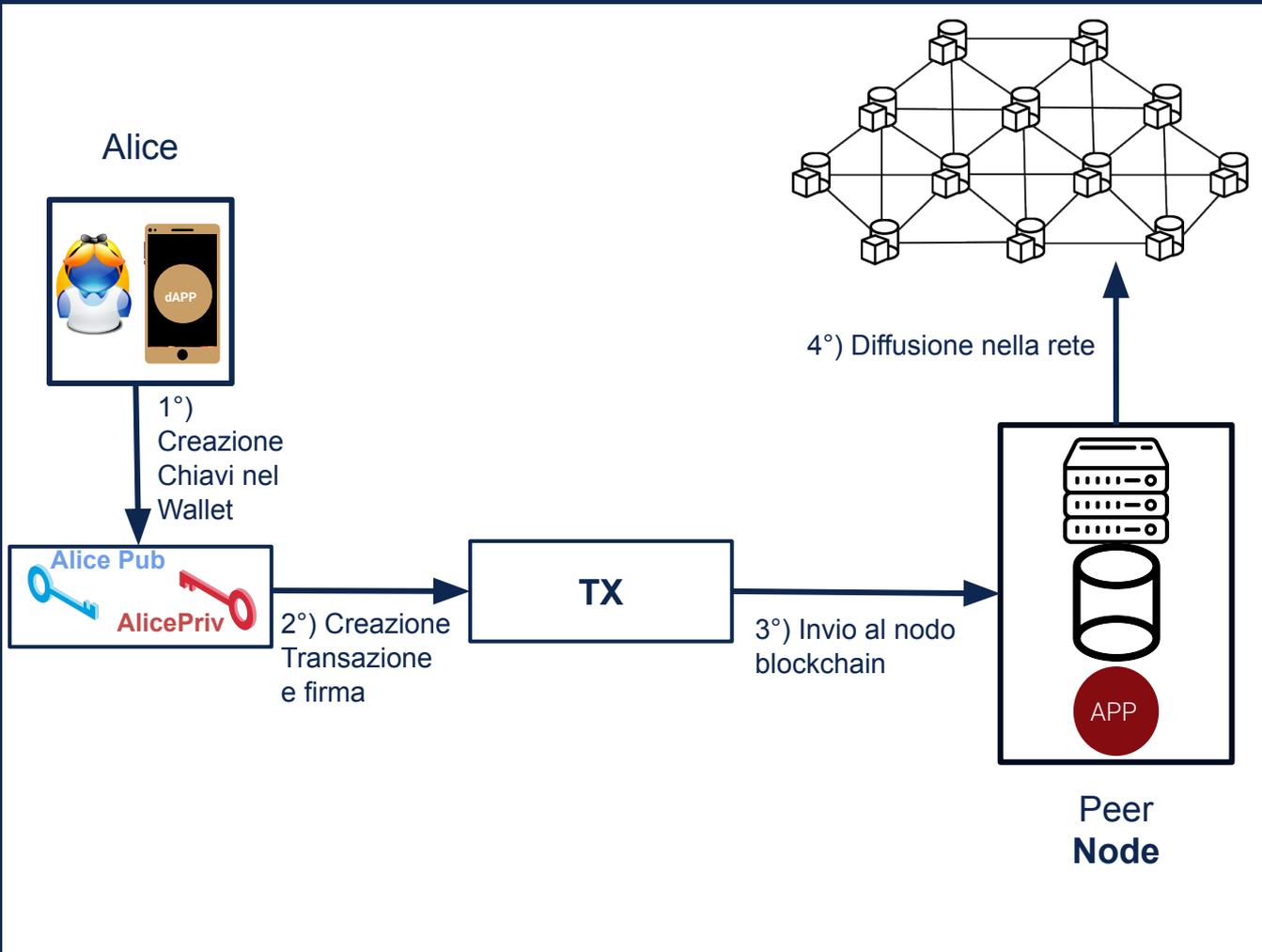


Comparazione Blockchain e Client/Server

Alice
paga
Bob
5 bitcoin



Esempio creazione transazione



Interfaccia nodi blockchain



Interfaccia utenti



1°) Creazione Chiavi nel Wallet

- Per inserire transazioni nella blockchain bisogna autenticarsi (firma digitale)
 - Se non si possiede già, serve una coppia di chiavi asimmetriche
 - Chiave Pubblica + Chiave Privata
- In Ethereum -> **Elliptic Curve Digital Signature Algorithm (ECDSA)**
 - La chiave privata consiste in un valore segreto generato da un processo di randomizzazione
 - La chiave pubblica deriva da quella privata
- L'Indirizzo (Address) di un Account si ottiene dalla chiave pubblica

1°) Creazione Chiavi nel Wallet



Private Key (256 bits):

```
'0xdc47ca238ffb638cf76658fb02a351bc7c1c6b  
bb32fa40db9bb43fee47c9dfbd'
```

↓
ECDSA secp256k1

Public Key (x,y point – 512 bits - Two 32-bit integers):

```
'b9f5d91099422bcfa991abe2866f3dc39bba8da  
50c52b77179eca74ecdaefd06cebbb2987f223c  
e8d1585d899999948b969b0039e3c36f14b297  
3cf20ed96330'
```

↓
Keccak-256
(first 40 bytes - 160 bits)

Ethereum address:

```
'0x39532829E35c3238cd0bc1613F7e586Cb106  
46CC'
```

<https://asecuritysite.com/encryption/ethadd>

Firma Digitale

$$\text{SHA256}(\text{TX}) = \text{digest}$$



$$\text{sign}(\text{digest} + \text{Alice Priv}) = \text{signature}$$



2°) Creazione Transazione e firma



2°) Creazione Transazione e firma

```
transaction = {  
  nonce: web3.toHex(0),  
  gasPrice: web3.toHex(200000000000),  
  gasLimit: web3.toHex(100000),  
  to: '0x687422eEA2cB73B5d3e242bA5456b782919AFc85',  
  value: web3.toHex(1000),  
  data: '0xc0de'  
}
```

rlp + hash



0x6a74f15f29c3227c5d1d2e27894da58d417a484ef53bc7aa57ee323b42ded656

sign with privateKey



```
v: '0x1c'  
r: '0x668ed6500efd75df7cb9c9b9d8152292a75453ec2d11030b0eec42f6a7ace602'  
s: '0x3efcbbf4d53e0dfa4fde5c6d9a73221418652abc66dff7fddd78b81cc28b9fbf'  
signature
```

https://miro.medium.com/max/1142/1*4Ta0bUfCEmgH4kOrNI8mKg.png



Parte interattiva

- 1. Installazione wallet (Metamask)**

<https://metamask.io/download.html>

- 2. Acquire Ether (Faucet) IOTA Testnet**

<https://wiki.iota.org/tutorials/shimmerevm-testnet-setup/>

- 3. Leggere dalla Blockchain**

<https://explorer.evm.testnet.shimmer.network/>

Introduzione agli Smart Contracts

+ Uno Smart Contract è (semplicemente) un programma che viene eseguito da tutti i nodi di una rete di una DLT

```
/**
 * @dev Log a vote for a challenge
 * @param challengeID The challenge position in the list
 * @param inFavour Boolean value indicating if in favour or not
 */
function vote(uint256 challengeID, bool inFavour)
    public
    onlyEligible(challengeID)
    notExecuted(challengeID)
{
    Challenge storage chall = _challenges[challengeID];
    Vote storage v = chall.votes[msg.sender];
    require(!v.voted, "Voting: already voted");

    _vote(challengeID, inFavour);

    emit Voted(challengeID, msg.sender, inFavour);
}
```



Origine Smart Contract

- **Nick Szabo, 1994** → "un protocollo computerizzato che esegue i termini di un contratto, implementato con programmi su una rete di computer, 'più intelligente' dei suoi antenati su carta"
 - *smart* ← *intelligere* ← *scegliere tra*
 - gli smart contract **automatizzano la scelta** in base a **condizioni predefinite**
- Esempio: **distributore automatico**
 - il *produttore* predetermina le **condizioni** (inserire una moneta X)
 - la macchina può **eseguire** il compito (consegnare il prodotto) *se queste condizioni sono soddisfatte*



Vending Machine



Alice



Manufacturer

La certezza di una corretta esecuzione è limitata alla fiducia che si ha nel produttore

Se il produttore dovesse cambiare i termini del distributore automatico, potrebbe ingannare il potenziale acquirente



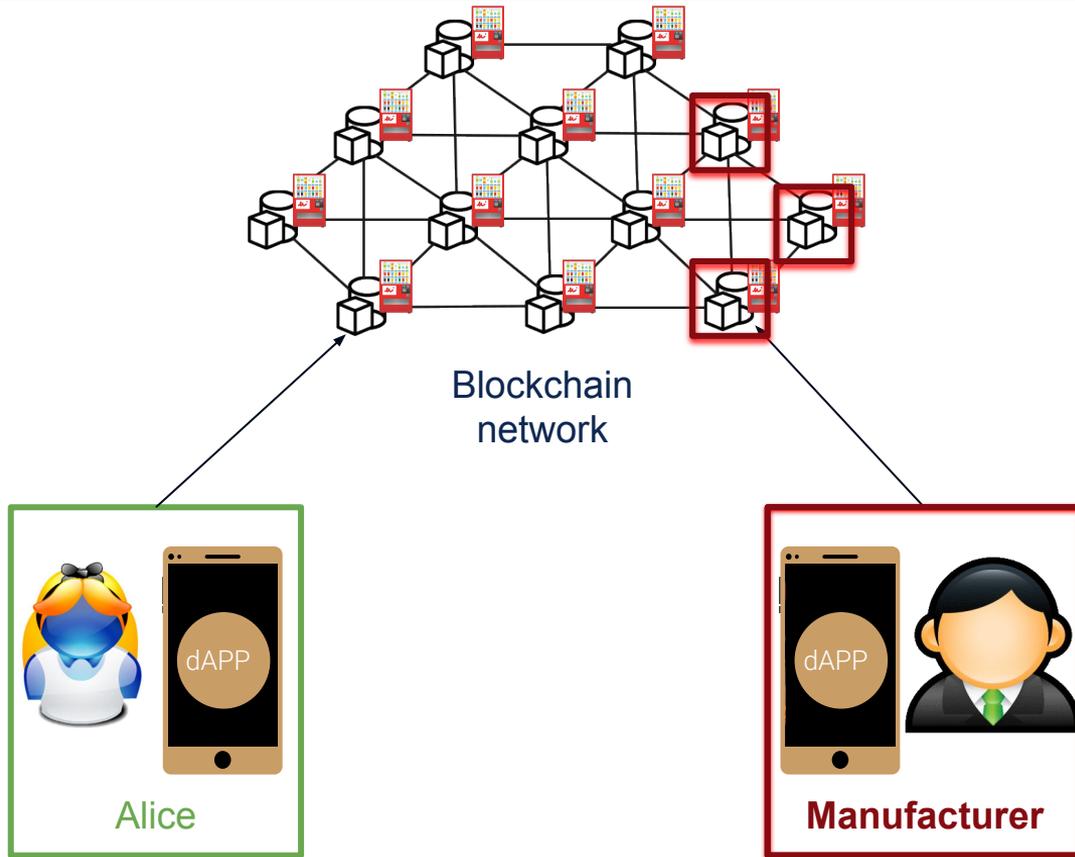
Fiducia contrattuale senza una terza parte fidata?

1. Se la **maggioranza** ($\frac{2}{3} + 1$) dei nodi in una rete è d'accordo ("*meccanismo di consenso*")
2. Se
 - a. si conosce il **codice sorgente** su cui è costruito
→ *open source*
 - b. se il **codice sorgente dello smart contract** è verificabile
→ *immutabilità dei dati*

allora

gli smart contracts

impediscono le deviazioni di natura tecnologica dall'accordo iniziale



Esempio pratico
del perché
fidarsi di uno
smart contract

Gli smart contract non
possono essere
cambiati o rimossi
unilateralmente.

Questo limite alle
azioni unilaterali
rafforza la fiducia.



Ethereum

- Ethereum può essere considerato come il tentativo di **decentralizzare l'accesso alle informazioni su Internet**
 - ma soprattutto come il riconoscimento dei limiti della rete **Bitcoin**, la prima ad utilizzare la blockchain.
 - L'intento è quello di creare un protocollo alternativo per lo sviluppo di **applicazioni decentralizzate** attraverso una Blockchain.
- Progetto open-source introdotto da → *Vitalik Buterin*
 - Nessuno “controlla” Ethereum, tuttavia esiste un'entità, la Ethereum Foundation, che controlla lo sviluppo del software progetto.



Ethereum

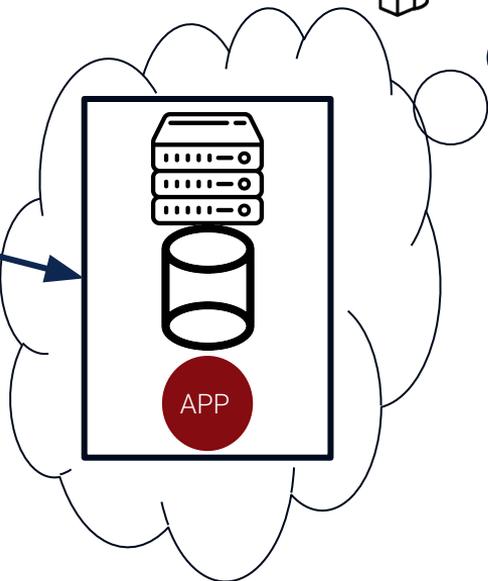
- Ethereum è un sistema composto da:
 - un network di **validatori** (nodi della blockchain)
 - un **algoritmo di consenso** (PoW, PoS)
 - un **registro condiviso** (blockchain)
 - un sistema di **indirizzi** (address e wallets)
 - una **computer decentralizzato** (macchina virtuale)
 - un insieme di **linguaggi di programmazione**
 - una **struttura economica** complessa (cryptocurrency e tokens)

Network di Validatori

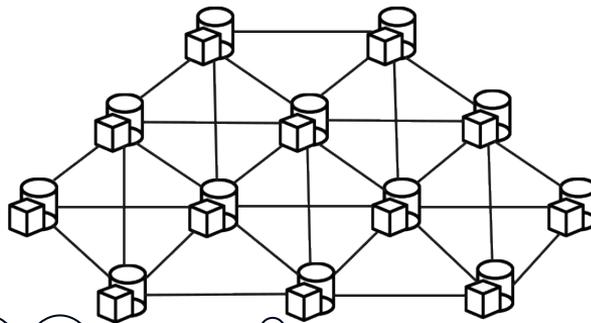


User

= **Wallet**



Peer Node = **Miner**

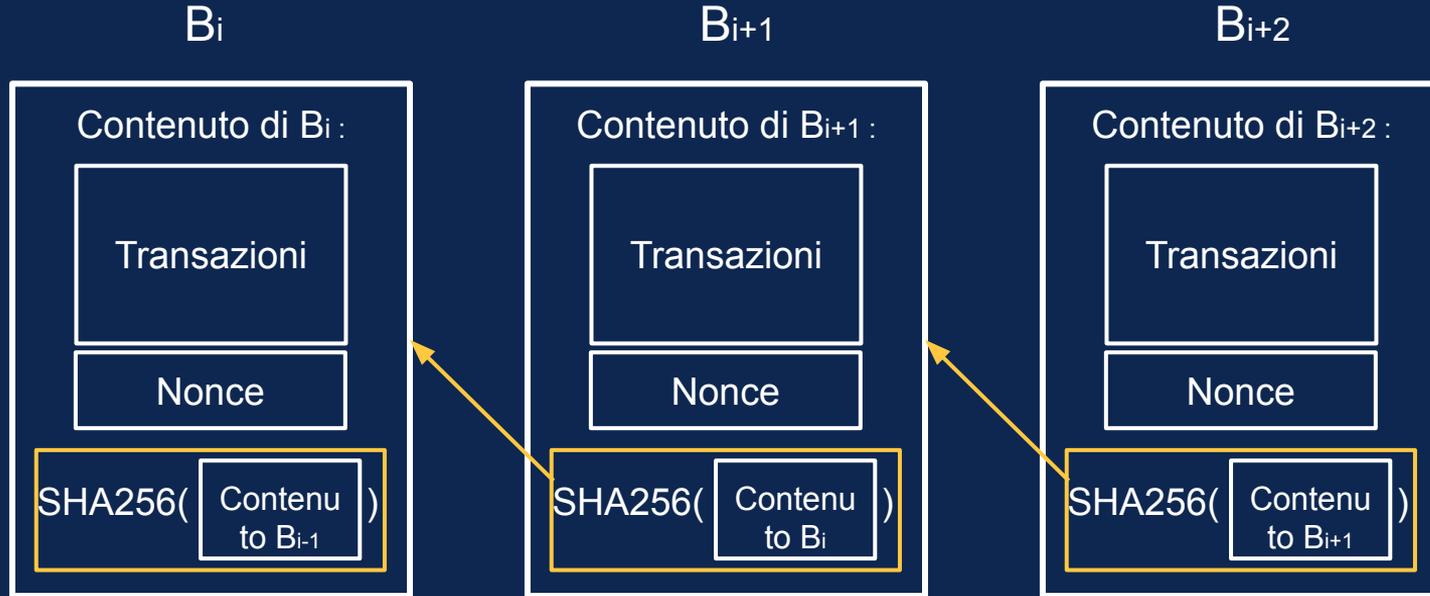


Interfaccia **nodi** blockchain



Interfaccia **utenti**

+ Registro condiviso (Blockchain)





Ethereum

- Ethereum è un sistema composto da:
 - un network di **validatori** (nodi della blockchain)
 - un **algoritmo di consenso** (PoW, PoS)
 - un **registro condiviso** (blockchain)
 - un sistema di **indirizzi** (address e wallets)
 - una **computer decentralizzato** (macchina virtuale)
 - un insieme di **linguaggi di programmazione**
 - una **struttura economica** complessa (cryptocurrency e tokens)

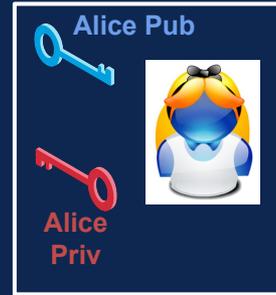




Wallet

applicazione che contiene uno o più Accounts e che permette di inviare o ricevere informazioni al/dal sistema

- **Account** -> modifica lo stato del sistema
 - **Externally Owned Account (EOAs)** ----->
 - **Contract Account (CAs)**



- **Messaggi e Transazioni** -> permettono di scambiare dati
 - Le transazioni sono definite come **pacchetti di dati firmati e inviati da un EOA**
 - I messaggi vengono scambiati solo **internamente al sistema tra CAs**

Introduzione alla programmazione in Solidity

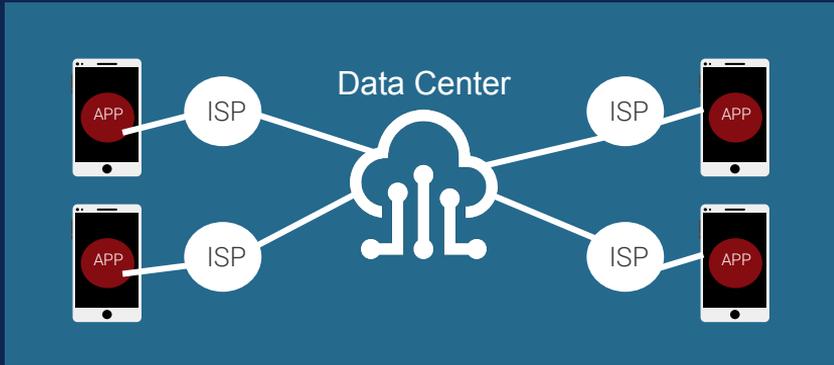


Ethereum

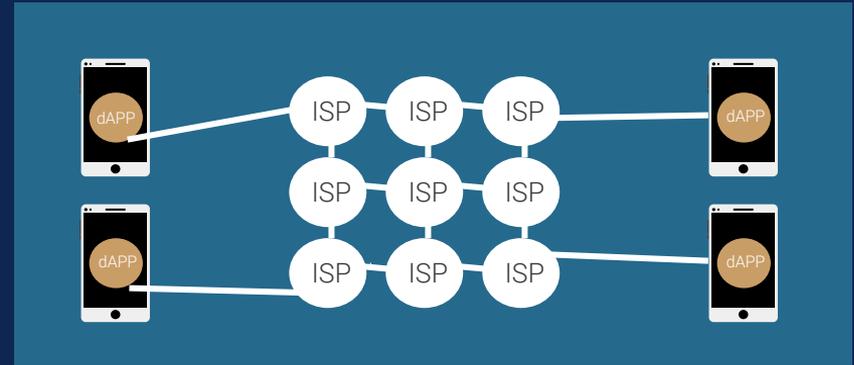
- Ethereum è un sistema composto da:
 - un network di **validatori** (nodi della blockchain)
 - un **algoritmo di consenso** (PoW, PoS)
 - un **registro condiviso** (blockchain)
 - un sistema di **indirizzi** (address e wallets)
 - una **computer decentralizzato** (macchina virtuale)
 - un insieme di **linguaggi di programmazione**
 - una **struttura economica** complessa (cryptocurrency e tokens)



+ Computazione Decentralizzata



Le app tradizionali fanno richieste che vengono elaborate da uno o “pochi” server



Le dApp (decentralized App) fanno richieste che vengono elaborate da tutti i nodi della rete blockchain

+ Decentralized Applications

Sono interfacce rivolte all'utente finale che lo collegano alla tecnologia blockchain attraverso una combinazione di Smart Contracts sottostanti



Il rapporto tra dApp, Smart Contracts e Blockchain è simile alle applicazioni web tradizionali. Le app client/server interagiscono con un particolare server per accedere al suo database.

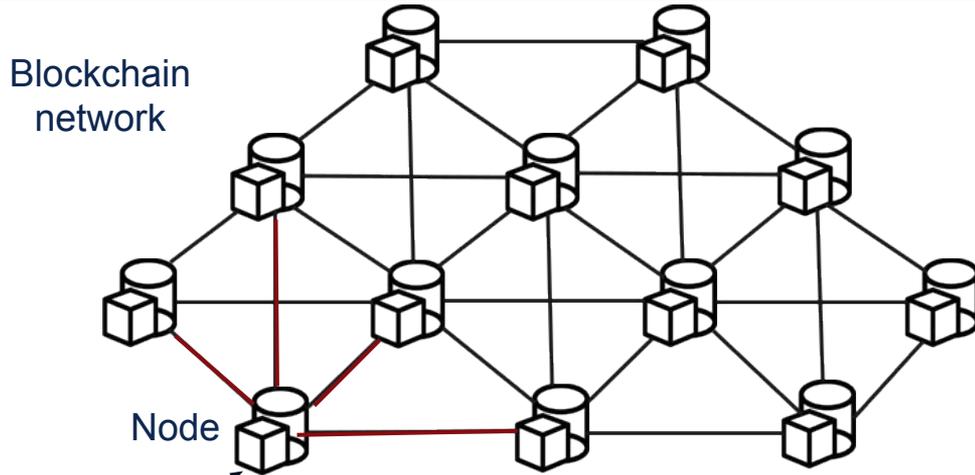
Analogamente, le dApp utilizzano gli Smart Contracts per connettersi alla particolare Blockchain su cui si basano (ad es. Ethereum).



Ethereum Smart Contracts

- Permette di mantenere facilmente delle strutture dati nella blockchain
- Una nuova transazione si riferisce ad una precedente e aggiorna lo stato del sistema
 - In questo caso lo stato del sistema considera non solo le transazioni monetarie, ma anche le strutture dei dati negli smart contracts
 - La transazione precedente si riferisce ad una che mantiene il codice e lo stato dello smart contract
 - La nuova transazione indica un insieme di istruzioni da eseguire nel contratto

Esempio: Un'operazione di voto in uno Smart Contract



```
execute(  
  SmartContractAddress: VotingContract,  
  methodToExecute: vote,  
  parameters: challengeID, true  
)
```

Esempio: Un'operazione di voto in uno Smart Contract

```
function vote(Challenge challenge, bool inFavour) public {  
    if (inFavour) {  
        challenge.inFavour.add(msg.sender); // Alice  
    } else {  
        challenge.against.add(msg.sender); // Alice  
    }  
}
```

+ Ethereum Virtual Machine (EVM)

maggiore differenza con Bitcoin

- **Linguaggio Turing Completo**

- Ogni sistema o linguaggio di programmazione in grado di calcolare qualsiasi cosa calcolabile, date sufficienti risorse, è detto Turing completo
- La EVM permette di **scrivere regole ed eseguire programmi *quasi-Turing-completi*** -> Smart Contracts

- **GAS**

- Il *quasi* si riferisce al fatto che ogni passo di computazione nella EVM ha un costo -> il GAS è l'unità di misura
- Ogni transazione deve includere un valore di GAS che indichi il limite di gas che l'esecuzione può raggiungere
- Il GAS utilizzato viene moltiplicato per un certo GASPRICE ed il risultato viene trasferito ai Miners come tassa per aver eseguito la computazione



Ethereum

- Ethereum è un sistema composto da:
 - un network di **validatori** (nodi della blockchain) ✓
 - un **algoritmo di consenso** (PoW, PoS) ✓
 - un **registro condiviso** (blockchain) ✓
 - un sistema di **indirizzi** (address e wallets) ✓
 - una **computer decentralizzato** (macchina virtuale) ✓
 - un insieme di **linguaggi di programmazione**
 - una **struttura economica** complessa (cryptocurrency e tokens)



Codice

1. Il codice è, nella sua forma più semplice, **un linguaggio usato per dare istruzioni** ai computer
2. Un **programma** per computer è un insieme di istruzioni scritte in codice ed eseguite da un computer.
3. Il processo di creazione di un programma si divide in:
 - a. **codice sorgente** → scrittura del codice in un linguaggio di programmazione di **"alto livello"**
 - b. **codice macchina** → la conversione finale del codice sorgente in un linguaggio di programmazione a **"basso livello"**



Linguaggi di programmazione di alto livello

Python, C++, Java, Javascript

Gli smart contract, come qualsiasi altro software, sono scritti in un linguaggio di programmazione di alto livello -> **Solidity***

Utilizzato dai programmatori per scrivere le istruzioni degli smart contracts, che poi vengono tradotte (*compile*) in linguaggio macchina -> **EVM**



Breve introduzione alla programmazione

1. Variabili

- a. Le variabili sono contenitore di dati situati in una porzione di memoria destinata a contenere valori.
- b. Sono identificate da un nome arbitrario (scelto dal programmatore).
- c. Sono suscettibili di modifica nel corso dell'esecuzione di un programma.
- d. Sono caratterizzate da un tipo.

2. Esempio

- a. **str** nome = "Moat"
- b. **int** anni = 24
- c. **float** peso = 54.3



Breve introduzione alla programmazione

3. Operatori

- a. Gli operatori sono simboli che specificano quale legge applicare a uno o più operandi, per generare un risultato.
- b. Ne esistono di vari tipi: aritmetici, booleani, assegnamento, decremento/incremento, condizionali, bitwise, di relazione.
- c. Funzionano... più o meno come ci si aspetta!

4. Esempio

- a.

```
int anni = 24
int anni_doppi = anni * 2
bool risultato = anni > anni_doppi
```
- b.

```
print(anni_doppi) >>> 48
```
- c.

```
print(risultato) >>> False
```



Breve introduzione alla programmazione

5. Strutture di controllo

- a. Le strutture di controllo sono dei costrutti sintattici che controllano il flusso di esecuzione di un programma.
- b. Specificano se, quando, in quale ordine e quante volte devono essere seguite determinate istruzioni.
- c. Le più note sono: if-then-else, while, for

6. Esempio

- a.

```
If (anni > 80) then {  
  bool vecchio = True  
} else {  
  bool vecchio = False  
}
```



Breve introduzione alla programmazione

7. Funzioni

- a. Le funzioni sono costrutti sintattici che permettono di raggruppare insieme una sequenza di istruzioni.
- b. Prendono uno o più valori in input e restituiscono uno o più valori in output.
- c. Sono identificate da un nome arbitrario scelto dal programmatore.
- d. Per essere eseguite, vanno “chiamate” o “invocate”.

8. Esempio

```
a. function somma(int a, int b) {  
    int c = a + b;  
    return c;  
}
```



Parte interattiva 2

1. **Lezioni crypto zombies**

<https://cryptozombies.io>



Parte interattiva 3

3. KLEROS

- a. https://www.youtube.com/watch?v=NuSps_2wMQ4&t=7s
- b. <https://www.ibanet.org/lex-cryptographia-due-process-blockchain-based-arbitration>
- c. <https://kleros.io/>
- d. <https://kleros.gitbook.io/docs/products/court/kleros-juror-tutorial>
- e. <https://ropsten.etherscan.io/address/0x9AdCEAa6CFd7182b838Beb085e97729EB1Da681E#writeContract>
- f. <https://court.kleros.io/>
- g. <https://court.kleros.io/cases/19>