

# **INDICE**

## **1. Introduzione alla blockchain**

- Concetti fondamentali
  - Storia della Blockchain
  - Bitcoin
  - Decentralizzazione
  - Principi base di crittografia per la blockchain
    - Crittografia a chiave pubblica
    - Hash
  - Validazione blocchi
    - Proof Of Work
    - Proof Of Stake
- Cosa rende rivoluzionaria la blockchain
  - Problemi risolti: doppia spesa
    - Proof of work di bitcoin
  - Problemi risolti: generali Bizantini
  - Rimozione degli intermediari
  - Gli smart contract
  - Prospettive per il futuro e possibili campi applicativi

## **2. La blockchain di Ethereum**

- Introduzione ad Ethereum
- Ecosistema Ethereum
  - Smart contract di ethereum
  - Token Ether
  - Le DAO
  - NFTs: token non fungibili
  - La DEFI: finanza decentralizzata
- Differenze con Bitcoin
- La rivoluzione di Ethereum

### **3. La tecnologia di Ethereum**

- Macchina virtuale di ethereum
- Il wallet Metamask
- La prima transazione con Metamask
- Chiavi pubbliche e chiavi private
- Linguaggio Solidity
  - il primo smart contract
  - Variabili e tipi solidity
  - Depositare e prelevare eth
  - Smart contract life cycle
  - Mapping e struct
  - Gestione degli errori
  - Fallback function
  - Costruttori
  - View and Pure
  - Ereditarietà
  - Eventi
  - Librerie
- Esempio utilizzo smartcontracts: Automated supply chain

# 1. Introduzione alla blockchain

## Concetti fondamentali

La blockchain (letteralmente "catena di blocchi") è una struttura dati in grado di memorizzare record di dati (detti *transazioni*) in modo sicuro, verificabile, permanente ed e' condivisa e *decentralizzata*: le copie intere dei dati sono distribuite fra I partecipanti alla struttura. È composta da un registro digitale le cui voci sono raggruppate in *blocchi*, concatenati in ordine cronologico, e la cui integrità è garantita dall'uso della *crittografia*. Il suo contenuto una volta scritto non è più né modificabile né eliminabile, a meno di non invalidare l'intera struttura. Per poter aggiungere dei nuovi dati, un blocco alla catena, e' necessario l'utilizzo di un *protocollo del consenso* che garantisce la coerenza fra le varie copie dei dati e la loro validita'. [1]

La blockchain si basa sull'idea delle reti peer-to-peer (P2P), reti informatiche in cui i nodi non sono gerarchizzati. Essa fornisce un set di dati universale di cui ogni attore della rete può fidarsi, anche se potrebbe non conoscersi o fidarsi l'uno dell'altro. Fornisce un registro delle transazioni condiviso e affidabile, in cui le copie immutabili e crittografate delle informazioni sono archiviate su ogni nodo della rete. Possono essere applicati incentivi economici sotto forma di token di rete nativi per rendere la rete tollerante ai guasti e resistente agli attacchi e alla collusione. [2]

## Storia della Blockchain

Il concetto di Blockchain è diventato famoso per la prima volta nell'ottobre 2008, con Bitcoin il cui obiettivo e' creare denaro P2P senza banche. Bitcoin ha introdotto una nuova soluzione al secolare problema della fiducia. La tecnologia blockchain sottostante ad un sistema consente di fidarsi degli output dello stesso senza necessariamente fidarsi di nessun attore al suo interno. Le persone e le istituzioni che non si conoscono o non si fidano l'una dell'altra, risiedono in paesi diversi o sono soggette a giurisdizioni diverse e che non hanno accordi legalmente vincolanti tra loro, possono interagire su Internet senza la necessità di terze parti fidate come le banche, piattaforme Internet o altri tipi di istituti di compensazione.

Tuttavia, il white paper di Bitcoin non è uscito dal nulla e le reti P2P non sono un fenomeno nuovo. Sono radicate nella storia primordiale del computer e di Internet, basandosi su decenni di ricerca su reti di computer, crittografia e teoria dei giochi. Il white paper di Bitcoin ha risolto il problema dell'archiviazione centralizzata dei dati e della gestione delle informazioni. Tutti i computer della rete detengono una copia identica del libro mastro (ledger) delle transazioni, che funge da unico punto di riferimento. L'archiviazione dei dati su una rete P2P elimina i problemi derivanti dalla vulnerabilità dei server centralizzati (one single point failure) mentre si utilizzano diversi metodi crittografici per proteggere la rete, questa ed altre specifiche verranno approfondite nel seguito.

Blockchain fornisce un livello di astrazione di un set di dati universale di cui ogni attore può fidarsi, anche se potrebbe non conoscersi o fidarsi l'uno dell'altro. Questa nuova forma di archiviazione e gestione distribuita dei dati evita anche il *problema della doppia spesa* (*vedi seguito problemi risolti*) del trasferimento di valore esistente su Internet. Le idee sulle reti P2P protette crittograficamente sono state discusse nell'ambiente accademico in diverse fasi evolutive, per lo più in articoli teorici, a partire dagli anni '80. Tuttavia, prima dell'emergere di Bitcoin, non c'è mai stata

una implementazione pratica di una rete P2P che riuscisse ad evitare il problema della doppia spesa, senza la necessità di intermediari fidati che garantissero lo scambio di valore. [5]

Blockchain è un registro delle transazioni condiviso, affidabile e pubblico, che tutti possono ispezionare ma che nessun singolo utente controlla. È un database distribuito che mantiene un elenco in continua crescita di record di dati di transazione, protetto crittograficamente da manomissioni e revisioni.

Il libro mastro viene creato utilizzando un elenco collegato, o catena di blocchi, in cui ogni blocco contiene un certo numero di transazioni convalidate dalla rete (dai partecipanti ad essa in accordo al protocollo del consenso) in un determinato periodo di tempo. Le regole cripto-economiche del protocollo blockchain regolano i comportamenti e i meccanismi di incentivazione di tutte le parti interessate nella rete.

Questo libro mastro viene eseguito su una rete di computer peer-to-peer. Il consenso distribuito basato su meccanismi di incentivazione economica (teoria dei giochi) combinati con la crittografia consente la convalida P2P sicura delle transazioni, aggirando così la necessità di terze parti tradizionali di fiducia. Esso è diventato famoso per la prima volta nell'ottobre 2008 come parte di una proposta per Bitcoin, con l'obiettivo di creare denaro P2P senza banche. Tutte le transazioni di rete vengono archiviate nella blockchain: Immagina Google Docs: ogni persona ha l'ultima versione del documento e tutti possono esaminarla. Per modificare i contenuti del documento, gli utenti devono raggiungere un accordo reciproco (consenso). A differenza di Google Docs, il file non è archiviato centralmente, ma ogni nodo della rete conserva una copia della blockchain, il libro mastro distribuito che registra tutta la cronologia delle transazioni.

Oggi i sistemi sviluppati con la blockchain si possono dividere in tre macrocategorie in base allo stadio di sviluppo delle tecnologie utilizzate:

- la **blockchain 1.0** riguarda il settore finanziario, le sue applicazioni hanno come scopo la gestione delle criptovalute (es. Bitcoin);
- la **blockchain 2.0** estende questi sistemi ad altri settori grazie all'implementazione degli smart contract (es. Ethereum);
- la **blockchain 3.0** riguarda una aspettativa futura il cui adempimento non sembra ancora vicino. Prevede l'utilizzo delle blockchain nella vita di tutti i giorni, anche inconsapevolmente, perché incapsulate in un web sempre più decentralizzato e in cui sia diffuso l'utilizzo delle Dapp (decentralized applications).<sup>1</sup>

## Bitcoin

La rete Bitcoin consente il possesso e il trasferimento pseudo-anonimo delle monete; i dati necessari a utilizzare i propri bitcoin possono essere salvati su uno o più personal computer o dispositivi elettronici quali smartphone, sotto forma di "portafoglio" digitale, o mantenuti presso terze parti che svolgono funzioni simili a una banca. Il wallet bitcoin ha un indirizzo identificato da un codice alfanumerico che possiede tra i 25 e i 36 caratteri tra numeri e lettere; è l'unico dato da comunicare per ricevere un pagamento che godrà di un certo grado di anonimato, ma sarà allo stesso tempo pubblicamente e immutabilmente visibile sulla blockchain per sempre. Occorre fare molta attenzione nella trasmissione del codice alfanumerico in quanto eventuali errori non consentono di annullare l'operazione e causano la perdita del denaro. È possibile ricevere pagamenti più semplicemente attraverso la scansione di codici QR.[6] In ogni caso, i bitcoin possono essere

---

<sup>1</sup> <https://www.zerounoweb.it/cio-innovation/blockchain-cose-come-utilizzarla-e-come-cambiera-il-business/>

trasferiti attraverso Internet verso chiunque disponga di un "indirizzo bitcoin". La struttura peer-to-peer della rete Bitcoin e la mancanza di un ente centrale rende impossibile a qualunque autorità, governativa o meno, il blocco dei trasferimenti, il sequestro di bitcoin senza il possesso delle relative chiavi o la svalutazione dovuta all'immissione di nuova moneta.

Dagli esperti di finanza il Bitcoin non viene classificato come una moneta, ma come una riserva di valore attualmente molto volatile. A differenza della maggior parte delle valute tradizionali, il Bitcoin non fa uso di un ente centrale né di meccanismi finanziari sofisticati, il valore è determinato unicamente dalla leva domanda-offerta: esso utilizza un database distribuito tra i nodi della rete che tengono traccia delle transazioni, ma sfrutta la crittografia per gestire gli aspetti funzionali, come la generazione di nuova moneta e l'attribuzione della proprietà dei bitcoin. [3]

## Decentralizzazione

La blockchain decentralizzata si basa sullo scambio di particolari messaggi su un networking distribuito per memorizzare i dati su tutta la sua rete per evitare di avere un *single point of failure*: un singolo punto di vulnerabilità del sistema software, il cui malfunzionamento puo' portare ad anomalie o addirittura alla cessazione del sistema.[7]

Ogni partecipante alla rete della blockchain e' detto nodo, ogni nodo nel sistema ha una copia della blockchain: difatti la qualità dei dati è mantenuta grazie a una massiva replicazione degli stessi. Non esiste nessuna copia ufficiale centralizzata e nessun utente è più credibile di altri, tutti sono allo stesso livello di credenziali. [4]

I *nodi miner*, validano le nuove transazioni, secondo le regole del *protocollo del consenso*, e le aggiungono al blocco che stanno costruendo dopo aver verificato l'intera blockchain. Una volta completato il blocco, lo trasmettono agli altri nodi della rete. Il numero di transazioni all'interno di ogni blocco varia in base alla dimensione delle transazioni e allo spazio disponibile definito per ogni blocco. Un blocco e' composto da due parti: l'intestazione e il corpo; le transazioni sono inserite nel corpo e nell'intestazione abbiamo vari campi per la gestione del blocco stesso, nella blockchain di Bitcoin ad esempio sono:

- Versione: indica la versione del protocollo utilizzato.
- Hash del blocco precedente: un codice generato tramite crittografia (vedi dopo) per identificare il blocco precedente nella catena.
- Merkel root: hash di tutti gli hash di tutte le transazioni nel blocco.
- Time stamp: rappresenta il time stamp dell'ultima transazione.
- Bits eNonce: sono due campi che vengono utilizzati per rendere il blocco accettabile dalla rete.
- Numero transazione: numero dell'ultima transazione inserita nel blocco. [3]

## Principi base di crittografia per la blockchain

La crittografia è lo studio delle tecniche che permettono di modificare un messaggio affinche' non venga letto da chi non ne è il destinatario, tale messaggio sarà detto *codificato*. Viene utilizzata generalmente per proteggere delle informazioni garantendone confidenzialità (l'informazione deve essere riservata fra mittenti e destinatari), integrità, autenticazione, non ripudiabilità ed è pesantemente utilizzata nella tecnologia BC. [15]

Un classico esempio di algoritmo crittografico è il cfrario di cesare, uno dei più antichi di cui si abbia traccia nella storia. In esso ogni lettera del testo in chiaro è sostituita, nel testo cifrato, dalla lettera che si trova un certo numero di posizioni dopo nell'alfabeto [figura 1].

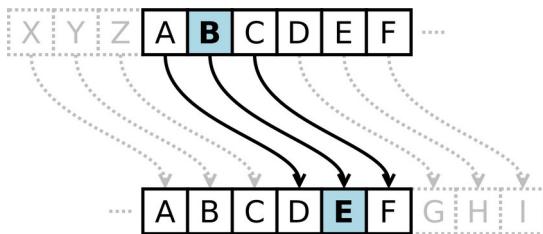


figura 1: Esempio cfrario di Cesare

La catena di blocchi viene chiamata così perché i blocchi, che contengono i dati, sono "legati" fra loro usando algoritmi di crittografia, ogni blocco contiene informazioni crittografate del suo precedente come sarà visto in seguito. [22]

Nella BC vengono usati due principi crittografici: crittografia a chiave asimmetrica e funzione hash.

### Crittografia a chiave asimmetrica

Questo metodo di crittografia utilizza una coppia di chiavi, una chiave di crittografia e una chiave di decripttografia, denominate rispettivamente chiave pubblica e chiave privata (o segreta). La coppia di chiavi è generata dallo stesso algoritmo. È anche chiamata crittografia a chiave pubblica. La chiave usata per crittografare è chiamata "chiave pubblica" e la chiave usata per decripttografare è chiamata "chiave privata" [figura 1].

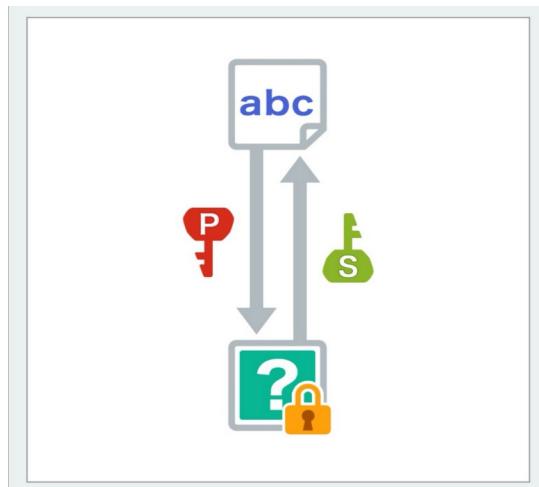


figura 2: Schema crittografia asimmetrica

Si analizza adesso un tipico scambio di dati utilizzante il sistema a chiave pubblica. Si supponga che A voglia mandare dei dati ad B tramite internet: per prima cosa A crea una chiave pubblica ed una chiave privata, la chiave pubblica viene manda a A [figura 3a,3b] che la usa per crittografare I dati che deve mandare, una volta crittografati questi vengono mandati a B che li decripta con la chiave privata che e' rimasta sempre e solo in suo possesso [figura 3c].

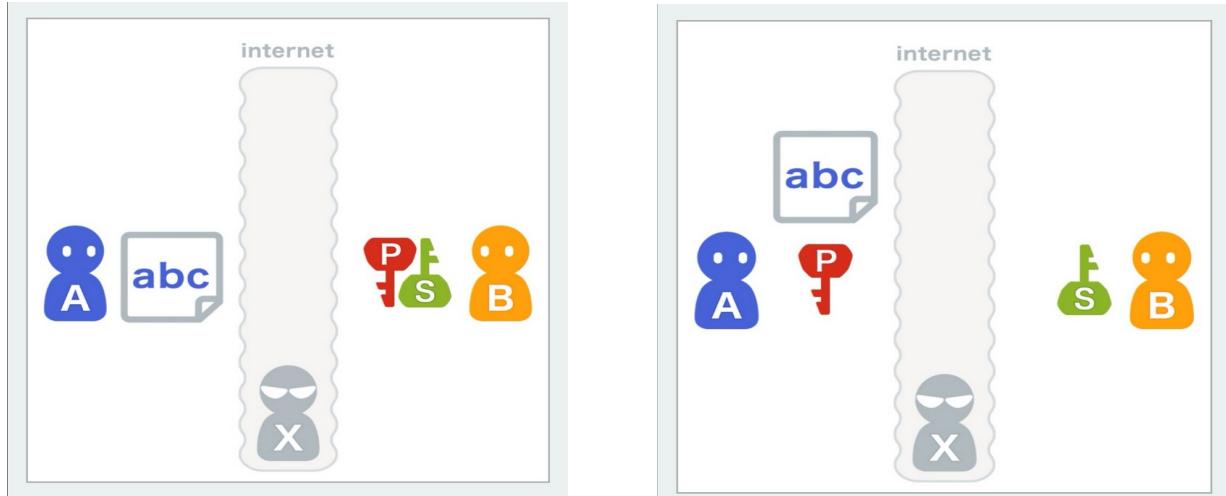


figura 3: a

figura 3: b

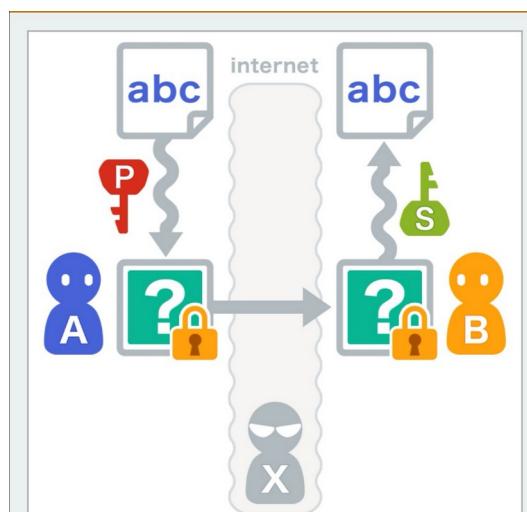


figura 3: c

Visto che la chiave pubblica e il dati crittografati sono stati mandati tramite internet c'e' la possibilita' che essi siano stati intercettati da una entita' malevola X, visto pero' che I dati non possono essere decriptati con la chiave pubblica X non puo' ottenere I dati originali[figura 4].

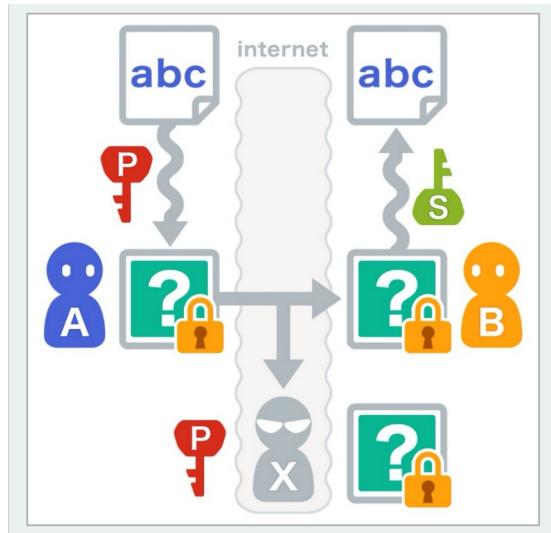


figura 4: Intercettazione, inefficace

Questo tipo di crittografia ha anche il vantaggio di rendere facile lo scambio di informazioni fra un indeterminato numero di parti. B dopo aver preparato le due chiavi non ha problemi nel pubblicare la sua chiave pubblica su internet, se piu' entita' vogliono mandare dati in modo sicuro a B essi possono ottenere la chiave pubblica crittografare I loro dati e mandarli; una volta arrivati B provvede a decriptarli con la chiave privata che deve essere invece cuscodita scrupolosamente [figura 5]. Non vi e' alcun bisogno di preparare una chiave pubblica per ognuno dei mittenti e se la chiave privata e' rimasta tale si ha un alto livello di sicurezza . [20]

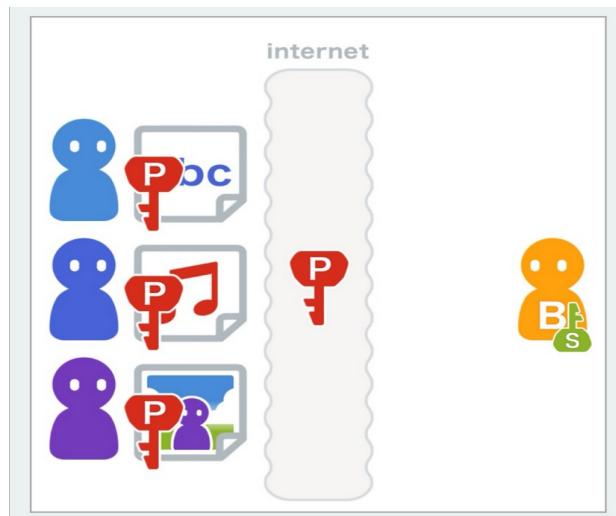


figura 5: Ricezione da molti

Si vede adesso ad esempio come la blockchain di bitcoin utilizza il sopra descritto sistema di crittografia, anche se la crittografia a chiave pubblica è un elemento fondamentale in generale della BC, essa è infatti la tecnologia sottostante per *wallet* e transazioni. Il wallet è un software, dove vengono salvate le chiavi pubbliche e private. Quando un utente crea un wallet (portafoglio) su una blockchain, genera una coppia di chiavi pubblica-privata. L'indirizzo di quel portafoglio, o come viene rappresentato sulla blockchain, è una stringa di numeri e lettere generata dalla chiave pubblica. A causa della natura della tecnologia blockchain, questo indirizzo è pubblico a tutti e può essere utilizzato per controllare il saldo in quel wallet o inviare monete ad esso.

La chiave privata associata a un wallet è come dimostrare la proprietà e permette di controllarlo. Senza la relativa chiave privata non è possibile inviare monete da quel wallet e smarirla significa non potere più autorizzare transazioni // ampliare concetto wallet.

Una transazione sulla blockchain di bitcoin ad esempio non è altro che un messaggio trasmesso che essenzialmente dice: "Prendi X monete dal mio portafoglio e accredita X monete in un altro portafoglio". Una volta confermata, la transazione viene immutabilmente scritta e i saldi vengono aggiornati. Tuttavia, questa richiesta di transazione richiede una firma dalla chiave privata del portafoglio di invio per essere valido. Dopo la trasmissione, chiunque può utilizzare la chiave pubblica di quel portafoglio per assicurarsi che la firma digitale proveniente dalla chiave privata sia autentica. Questo è un ruolo dei *validatori di blocchi* prima di aggiungere qualsiasi transazione alla blockchain. [6]

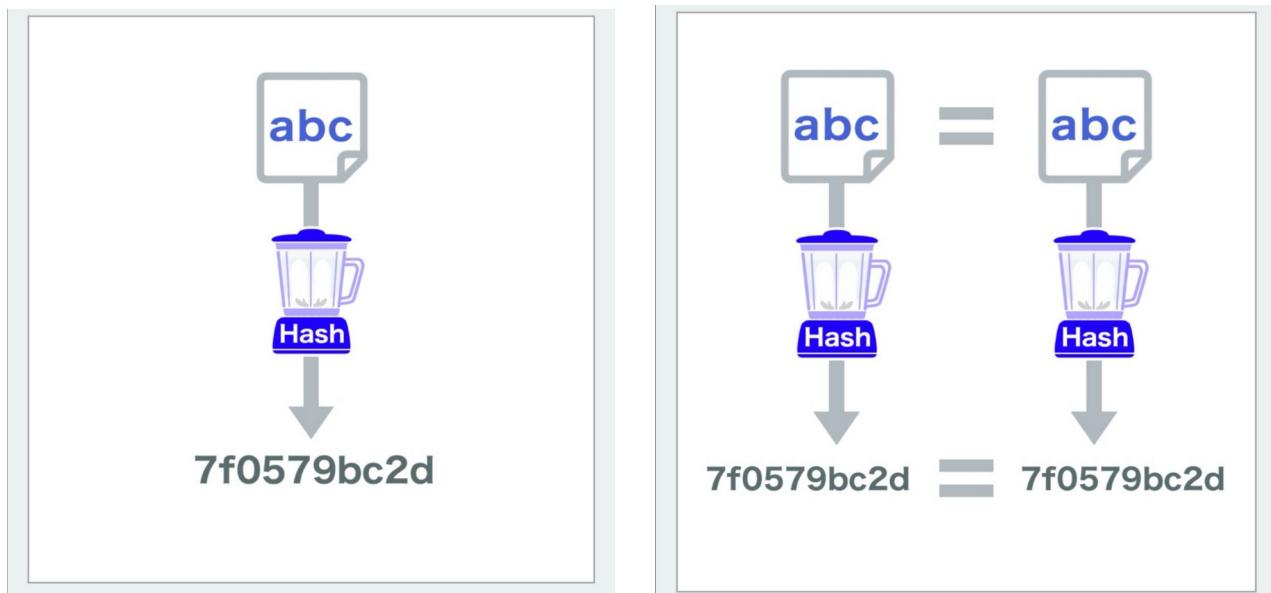
## Funzioni hash

Questo tipo di crittografia non utilizza chiavi. Utilizza un codice per generare una stringa di caratteri detta *valore hash* di una lunghezza fissa da un qualsiasi insieme di dati[figura 6].

Un algoritmo che genera un valore di hash viene detto *funzione di hash*, queste si distinguono dall'algoritmo usato e dal valore che restituiscono, particolarmente importante la lunghezza della stringa restituita che ne va a caratterizzare la “robustezza” come sarà spiegato in seguito.

Le caratteristiche fondamentali delle funzioni di hash sono:

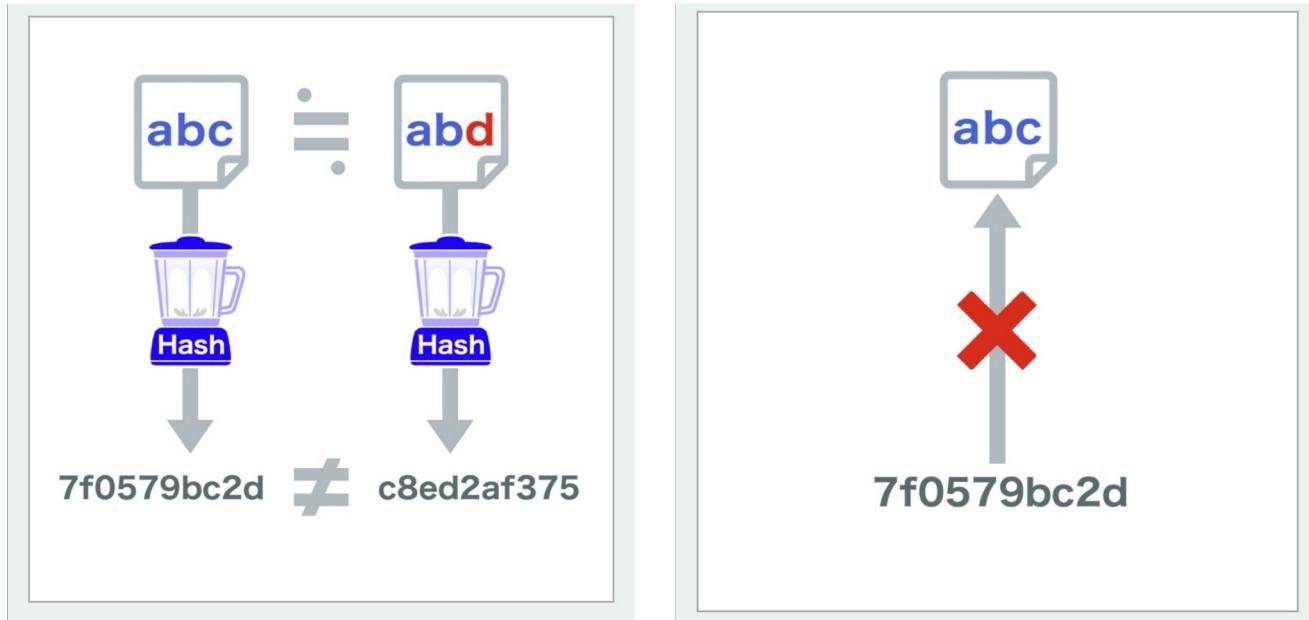
- 1) Lunghezza finita: la stringa generata è di dimensione finita indipendentemente dalla dimensione dei dati in ingresso, di solito viene espressa in esadecimale.
- 2) Due sequenze di dati uguali avranno lo stesso risultato [figura 7].
- 3) Una piccola differenza nei dati di cui fare l'hash risulta in una grande differenza dello stesso [figura 8].



- 4) Può capitare che due diverse sequenze di dati diano origine allo stesso hash, questa viene detta *collistione*. La probabilità con cui si verifichi una collisione è molto bassa e diminuisce con l'aumentare della lunghezza della stringa generata dalla funzione di hash. La probabilità di collisione varia con la tipologia di funzione hash, usualmente è nell'ordine di una su miliardi di miliardi (praticamente impossibile).

5) E' impossibile risalire ai dati originari partendo dall'hash, questo e' dovuto ad una perdita di informazione durante il processo di hashing [figura 9].

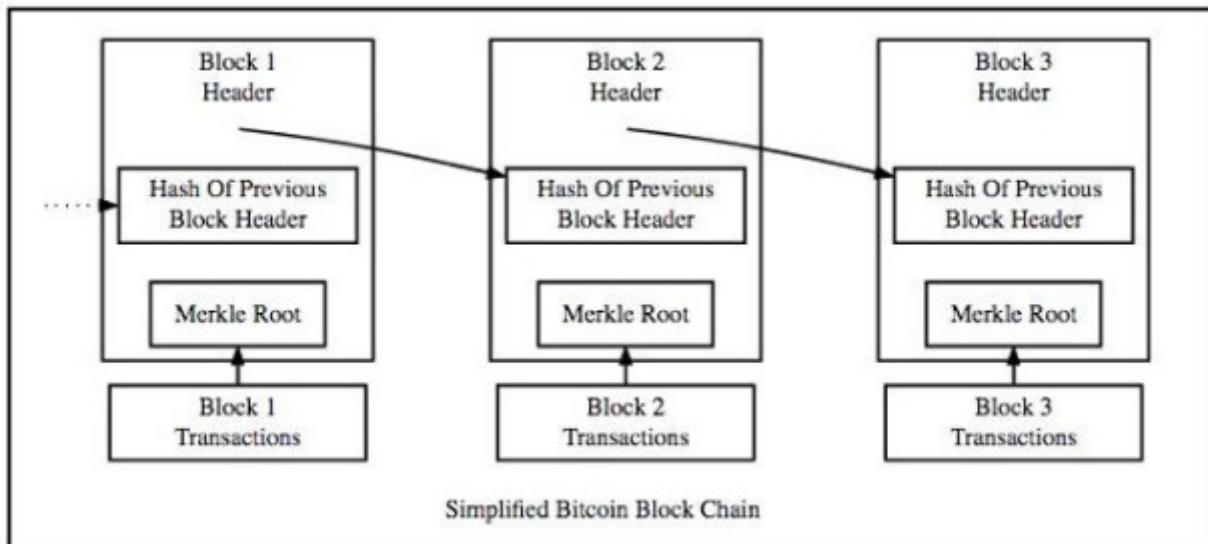
6) Complessita' computazionale relativamente ridotta, non e' necessaria una grossa potenza di calcolo per applicare una funzione di hash.



L'hashing è un elemento fondamentale della tecnologia blockchain ed è grazie ad esso che si implementa una delle caratteristiche più importanti della blockchain, l' immutabilità.

Ogni nuovo blocco aggiunto alla BC contiene al suo interno l'hash di tutti I dati presenti nel blocco precedente, questo e' quello che lega I blocchi (da qui catena di blocchi).

Se alla BC e' stato aggiunto il suo 1000esimo blocco esso conterra' l'hash dell blocco 999 che a sua volta contiene quello del blocco 998 e cosi' via. Attraversando gli hash all'indietro, ogni blocco da 1000 a 1 è collegato. Di seguito viene mostrato un diagramma di questa architettura [figura 10][21]



Questo è in definitiva ciò che rende immutabili i dati in una blockchain. Se qualcuno provasse a modificare solo 1 bit di dati in un blocco passato, non solo altererebbe hash corrispondente di quei dati di blocco, ma ogni blocco successivo, questo viene detto effetto valanga. I miner e i nodi della rete noterebbero immediatamente che gli hash risultanti non corrispondono alla loro versione BC e rifiuterebbero la modifica.

Le funzioni hash, oltre ad avere un ruolo importante nel collegare i blocchi tra loro servono per mantenere l'integrità dei dati memorizzati all'interno di ciascun di essi. Qualsiasi alterazione nei dati del blocco può portare a incoerenze e rompere la blockchain, rendendola non valida, questo requisito è raggiunto grazie all'effetto valanga.

Un esempio, relativo a bitcoin: presi due blocchi, blocco A e blocco B (di cui il blocco A è il primo blocco nella blockchain) per verificare il blocco A, i minatori raccolgono i dati della transazione e gli danno un hash - "hash A".

Per verificare il blocco successivo della catena (B), i minatori dovranno raccogliere un altro set di transazioni e trovare un nuovo hash - "hash B". L'hash B è costruito a partire dall'hash A e sui nuovi dati delle transazioni del blocco B.

Ora, se un hacker malintenzionato volesse modificare i dati nel blocco A, l'hash A cambierebbe, poiché si basa sui dati contenuti nel blocco A. Di conseguenza, anche l'hash B cambierebbe, e anche tutti gli altri hash che seguono l'hash B.

Detto questo, un malintenzionato dovrebbe alterare l'intera BC per modificare i dati archiviati. Ciò, tuttavia, è praticamente impossibile, perché richiede troppa potenza di calcolo.[\[18\]](#)[\[19\]](#)

## Validazione dei blocchi

Ogni volta che una transazione viene condotta su una BC, i dati della transazione verranno archiviati in un nuovo blocco. Questo nuovo blocco verrà quindi aggiunto alla blockchain.

Ma prima che il blocco possa essere aggiunto alla catena, le informazioni in esso contenute devono essere verificate dalla rete, questo avviene attraverso meccanismi ben precisi dettati dal particolare protocollo del consenso di quella BC, in genere ciò avviene creando un cosiddetto "hash".

Un hash è un numero a 256 bit che identifica in modo univoco i dati nel blocco. Per creare questo hash, i nodi della rete devono risolvere un complesso "puzzle matematico". Una volta risolto il puzzle, tutti gli altri nodi sulla rete controllano se i calcoli sono corretti.

Il processo di risoluzione di questo puzzle e di conseguenza la creazione di un nuovo hash si chiama "mining".

Il mining richiede una grande potenza di calcolo, poiché si basa su complesse operazioni matematiche. Richiede anche hardware informatico specializzato. Pertanto, non tutti i nodi della rete saranno in grado di essere un "miner".

L'estrazione mineraria consuma energia e l'energia costa denaro. Quindi, ci deve essere un incentivo per i minatori a estrarre nuovi blocchi.

Questo incentivo è chiamato "mining reward" e di solito viene pagato nella criptovaluta originaria della rete blockchain. Al momento, la reward per la verifica di un nuovo blocco sulla rete Bitcoin è di 6,25 bitcoin. Questi bitcoin sono stati creati di recente, ecco perché il processo si chiama mining. Il mining diventando redditizio genera una competizione, più minatori competono per le reward, permettendo alla BC di funzionare. Senza minatori, non ci sarebbero nuovi blocchi, di conseguenza, la blockchain diventerebbe disfunzionale. [\[17\]](#)

Le reti blockchain applicano un principio di consenso che definisce quale miner riceverà la ricompensa. Ci sono diversi modi per farlo:

## **Proof of work (PoW)**

PoW è il sistema di ricompensa comunemente utilizzato nelle reti di criptovaluta. Sia il Bitcoin che la rete Ethereum si affidano a PoW.

Una volta creato un nuovo blocco, tutti i miner sulla rete inizieranno a lavorare sull'hash puzzle. Il minatore che lo risolve per primo, riceve la ricompensa. Primo arrivato, primo servito.

## **Proof of Stake (PoS)**

PoS ha lo stesso obiettivo di PoW: convalidare le transazioni creando un nuovo hash.

Tuttavia, in un sistema PoS, i nodi non competono per la ricompensa mineraria. Viene invece selezionato un singolo nodo per convalidare l'hash successivo. Il criterio per la selezione è la ricchezza del nodo, o in altre parole, è la posta in gioco nella rete.

Pertanto, in una rete basata su PoS, il consumo di energia sarà molto inferiore, perché solo un nodo sta lavorando per risolvere il problema matematico.

Inoltre, in un sistema PoS, la ricompensa non viene pagata in nuova emissione di moneta. Invece, il nodo selezionato riceverà una commissione di transazione. Tutte la moneta e' già emesse durante la creazione della rete. I nodi che trovano un nuovo hash in un sistema PoS non sono chiamati minatori, ma "falsificatori" (forgers).

Esistono più modi per convalidare le transazioni, ad esempio Proof-of-Authority, Proof-of-Burn, Proof-of-Capacity o Proof-of-Elapsed Time.

In linea di principio, tutti questi sistemi hanno lo stesso obiettivo: convalidare nuovi dati sulla rete. Solo il modo in cui vengono selezionati i minatori sarà diverso.

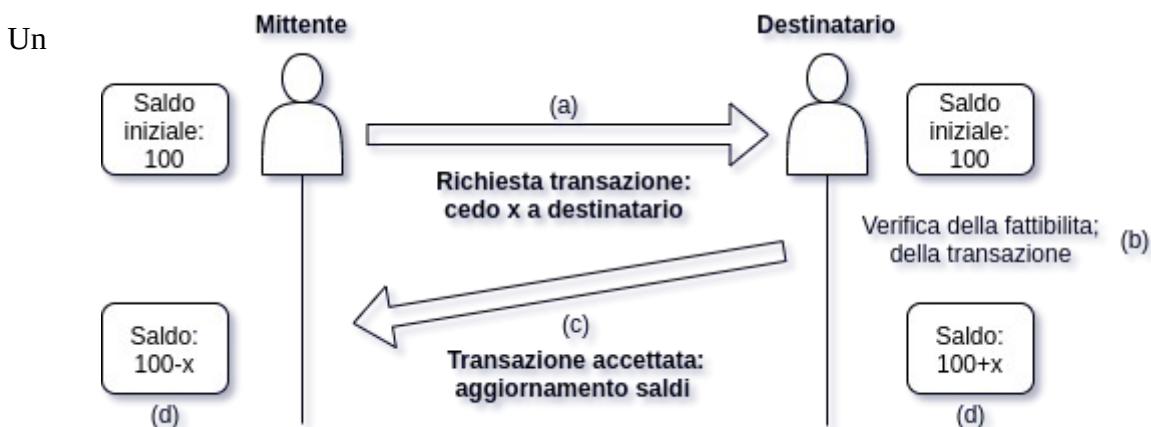
# Cosa rende rivoluzionaria la blockchain

La BC porta con se aspetti molto innovativi, oltre a risolvere vari problemi tecnici, essa puo' permette un utilizzo della rete, e quindi delle operazioni fatte su essa, piu' sicuro e fluido. Analizziamo nel seguito alcuni dei motivi che portano molti a definirla "internet 3.0".

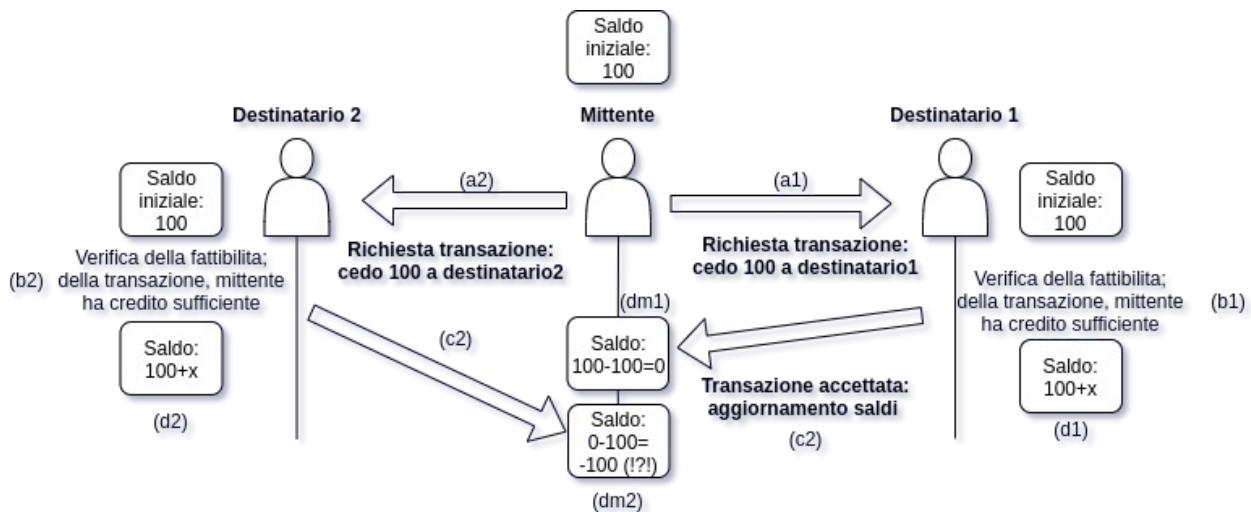
## Problemi risolti: doppia-spesa

Se un acquirente entra in un negozio e paga un qualsiasi bene in denaro contante, una volta avvenuto il "passaggio di mano" delle banconote e delle monete queste, semplicemente, non possono essere piu' spese in un altro negozio. A differenza del denaro contante, quello elettronico non e' altro che una configurazione di bit su un dispositivo hardware, esattamente come un file, che puo' essere copiato e quindi potenzialmente "usato" piu' volte, appunto doppiamente speso, entriamo piu' nel dettaglio.

Quando un pagamento viene effettuato elettronicamente deve avvenire una comunicazione fra il depositario del denaro dell'acquirente, colui che gestisce l'hardware in cui e' memorizzato il saldo, e il venditore; cio' non avviene istantaneamente ma richiede un lasso di tempo, in cui viene verificato se la transazione sia possibile, per esempio se l'acquirente abbia credito sufficiente. In un ipotetico scambio potrebbe essere realizzato, ingenuamente, dalle seguenti azioni a) richiesta della transazione b) verifica della correttezza della transazione c) conferma della transazione d) aggiornamento dei saldi di acquirente e ricevente[figura].



sistema del genere e' presenta diverse falte: non si ha garanzia che il mittente abbia veramente un certo credito, che corrisponda ad una entita' reale, che le entita' in gioco aggiornino correttamente il loro saldo ecc. Inoltre, se il mittente tenta di fare un'altra transazione prima che la procedura di prima si sia esaurita ed abbia portato all'aggiornamento del saldo, potrebbe accadere che un venditore accetti un pagamento con un saldo in verita' insufficiente, quindi con un denaro gia' speso [figura doppia spesa]



Per risolvere questo problema, che e' un problema di fiducia, si fa riferimento ad un ente terzo centralizzato che verifica e ad visione globale di tutte le transazioni (e dei saldi).

Tramite la BC, per prima quella di bitcoin, e' stato risolto questo problema senza la neccessita' di una autorita' garante centralizzata, questo avviene registrando le transazioni sul ledger (pubblico) solo dopo che queste siano state approvate dall'insieme della rete stessa seguendo un protocollo del consenso. Il protocollo del consenso e' un software che determina un insieme di regole a cui tutti I partecipanti della rete devono seguire per farne parte. [39]

Nel seguito viene analizzato il protocollo del consenso Proof of Work di bitcoin in quanto di miliare importanza

### Protocollo del consenso POW bitcoin

Nella BC di Bitcoin vi sono dei nodi, detti miner, addetti alla creazione ed approvazione e quindi aggiunta dei nuovi blocchi, essi prendono le transazioni dalla lista delle transazioni in sospeso, visibili a tutta la rete e ne mettono un numero prestabilito in un blocco, creando un *blocco candidato* che verra' successivamente mandato in broadcast (comunicazione da un punto della rete destinata ad arrivare a tutti I suoi partecipanti) e quindi sottoposto al giudizio degli altri nodi. Le transazioni all'interno di un blocco devono rispettare un set di regole (circa una trentina) imposte dal software (codice) di bitcoin; esistono regole anche per l'intero blocco. La conformita' ad esse viene garantito sia dal miner che crea il blocco candidato, che ha convenienza in questo come si vedra' nel seguito, sia dagli altri nodi della rete.

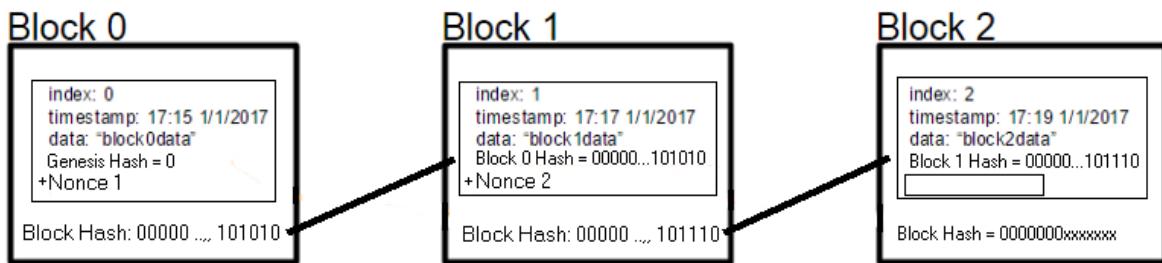
Un blocco candidato costruito sarebbe subito pronto ad essere trasmesso ma al fine di evitare una proliferazione eccessiva di blocchi candidati, che causerebbero intasamento sulla rete, e per garantire affidabilita' e fiducia sui dati nella blockchain e' stato deciso di porre nel protocollo un limite all'invio: prima di essere mandato in broadcast il miner deve risolvere un problema informatico che richiede tempo, capacita' di calcolo, elettricità, *del lavoro (work)* che deve essere testimoniato (proof) a chi ricevera' questo blocco *minato*.

Per rendere conveniente economicamente il mining quando un blocco viene minato gli viene aggiunta una transazione che ricompensa di un valore prefissato in bitcoin il miner. Questo ha portato nel corso degli anni ad un corsa alla competizione nel mining, attualmente questa attivita' viene svolta da grosse imprese che investono capitali nell'acquisto e sviluppo di hardware dedicato, sostengono costi di manutenzione e organizzazione e naturalmente di energia elettrica e collegamento internet. Ogni secondo vengono utilizzati diversi megawatt di potenza da questi miner

che contemporaneamente cercano di aggiudicarsi la ricompensa per il prossimo blocco che verrà aggiunto alla BC. Attualmente viene creato un blocco circa ogni 10 minuti.

Facciamo un passo indietro e vediamo il primo approccio ideato del proof of work per comprendere meglio il seguito. Nel 1997 Adam Back mette a punto una tecnica per evitare lo spam di email indesiderate, consisteva nell'applicare la funzione di hash n volte alla email prima di inviarla ed includere il risultato nel campo oggetto. Per fare questa operazione il mittente impiegava circa mezzo secondo, altrettanto il destinatario per verificarlo, un tempo del tutto accettabile per utilizzare il servizio di posta, cosa che invece non era più per lo spammer che doveva inviare un milione di email alla volta. Questo è il principio del proof of work che bitcoin implementa a suo vantaggio.

In bitcoin, sappiamo che un blocco ha un proprio hash creato utilizzando un algoritmo di hashing (sha 256 per bitcoin), il protocollo prevede che questo hash abbia un formato ben preciso: un numero n definito di zeri iniziale. Visto che l'hash di un blocco dipende dal suo contenuto esso sarebbe già fissato a priori e quindi sarebbe impossibile trovarne uno con una serie di n zeri iniziale, per questo prima di calcolare l'hash viene aggiunto dal miner un valore numeri di dimensione finita detto *nonce*, il miner per soddisfare la condizione è quello di cercare un nonce che aggiunto al blocco crea l'hash con la fila di zeri iniziale. La difficoltà nel trovare il nonce giusto aumenta con l'aumentare della lunghezza della fila di zeri [figura nonce e hashing].

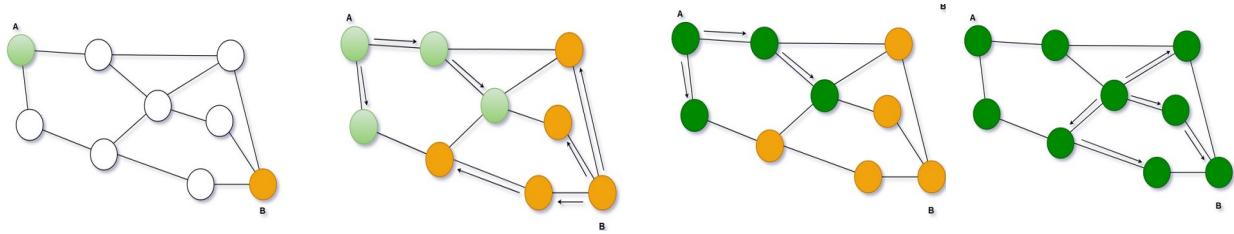


Quando un miner riceve un messaggio che comunica un blocco minato verifica se soddisfa le regole del protocollo e in tal caso aggiunge il nuovo blocco alla sua versione della BC e smette di lavorare sul suo blocco candidato iniziando subito a farlo su uno nuovo. È evidente che lo scopo di un miner è quello trovare il nonce e comunicarlo il più velocemente possibile, inoltre si ha convenienza economica nel creare blocchi conformi alle regole del protocollo (queste regole sono codificate nel software di bitcoin in linguaggio C++).

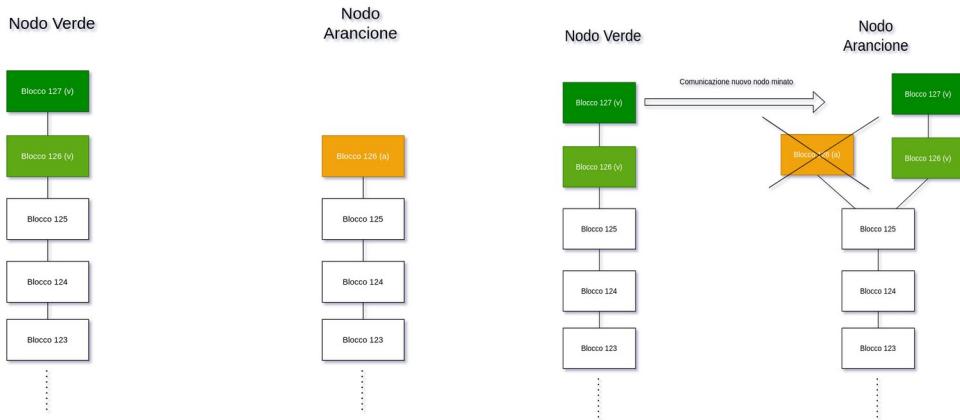
Se un nodo dovesse abbandonare la rete, una volta riconnesso appena riceve la comunicazione di un blocco minato esso può richiedere alla rete i blocchi che gli mancano da dove era rimasta ferma la sua versione della BC fino a questo ultimo blocco.

Potrebbe accadere che più di un miner abbia minato un blocco, in tal caso un nodo potrebbe ricevere diverse comunicazioni per nuovi blocchi da aggiungere alla BC, la scelta del blocco si basa su concetto di lavoro. Su un blocco è possibile calcolare il lavoro impiegato per minarla detto *peso*, ovvero la difficoltà computazionale che è stata necessaria per trovare il suo nonce. Risalendo indietro nella catena tramite il numero del blocco e l'hash è possibile calcolare il peso di ogni blocco e sommando ottenere il *peso totale* della blockchain fino a quel punto. Un nodo che riceve diversi blocchi minati sceglie di aggiungere quello che appartiene alla catena col peso totale maggiore, che corrisponde alla catena più lunga.

Può verificarsi che due blocchi siano minati a breve distanza, essi saranno leggermente diversi visto che l'ordine delle transazioni può cambiare ma il peso totale della catena alle loro spalle sarà lo stesso, si entra quindi in una condizione di indecisione chiamata in informatica *race condition* (condizione di concorrenza). I nodi che ricevono uno di questi due blocchi e lo accettano iniziano a minare aggiungendogli blocchi in serie, nel contempo i nodi che riveno l'altro blocco fanno lo stesso, si ha quella che viene detta *fork (biforcazione)*: una parte di nodi “costurisce” su una linea e una parte su un’altra. Nell’esempio in figura è stato ipotizzato che il nodo A e B minino un nuovo blocco contemporaneamente e lo comunichino nella rete [fig fork a]. I nodi verdi sono quelli che ricevono per primo il nodo minato da A e quelli arancioni quello di B [fig fork b]. Successivamente il nodo A mina un altro nodo e lo comunica nella rete, i nodi che lo accettano diventano verde scuro [figura fork c].



Il messaggio del nuovo blocco arriva anche a quelli arancioni che riconoscendo un peso maggiore eliminano la parte di catena che avevano costruito e la rimpiazzano con la nuova[figura fork d-e1-e2].



I nodi che ricevono comunicazione dell’altra “linea” con peso maggiore, eliminano la loro linea fino all’ultimo blocco in comune con l’altra e ricostruiscono la catena con lo stesso procedimento di un nodo che aveva abbandonato la rete. Si ha quindi una stratificazione dipendente dal lavoro che si è fatto su una linea, più e’ grande il lavoro più e’ difficile creare una linea malevola con informazioni alterate. Se ne volessi creare una dovrei avere il controllo sulla maggioranza della potenza di calcolo della rete per costruire una linea più velocemente di come fa il resto della rete, il che richiederebbe un impegno economico astronomico.

Se invece volessi prendere il controllo della blockchain creandone una parallela dovrei in 10 minuti (tempo medio mining nuovo blocco) costruire una serie di blocchi con peso maggiore della blockchain esistente il che e’ praticamente impossibile. In tutto questo modello e’ imperante la

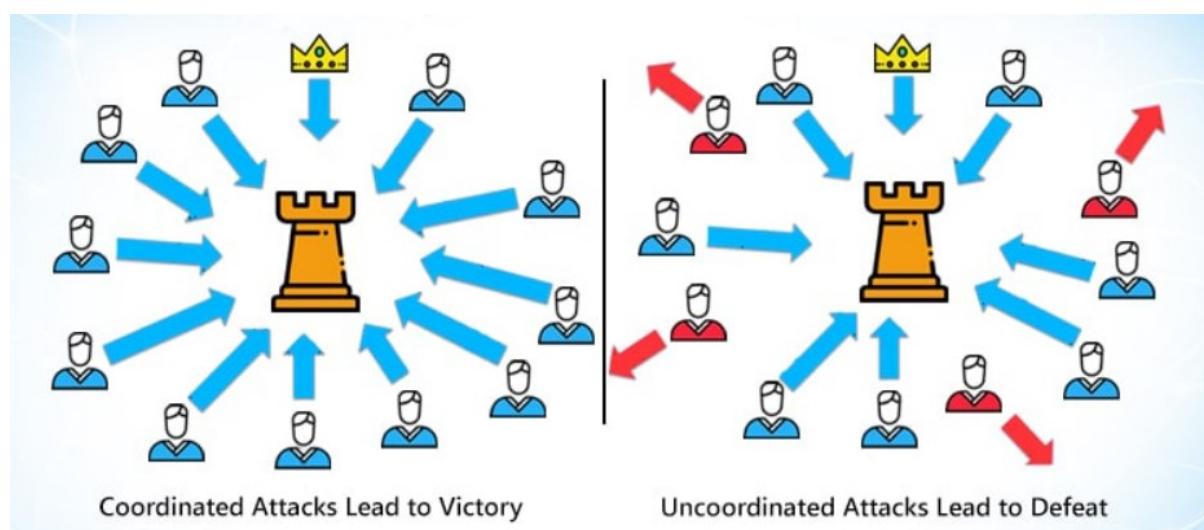
teoria dei giochi, ad ogni agente del sistema conviene operare secondo le regole e andarne contro e' molto faticoso e dispendioso. Il risultato finale di tutto questo processo e' la capacita' globale della rete di bilanciarsi ed essere affidabile. [40]

I detrattori del proof of work criticano il grandissimo consumo di elettricità, quindi impatto ambientale, dovuto alla rete bitcoin; I sostenitori dicono che questo e' il prezzo da pagare per la sicurezza e l'affidabilità della rete.

## Problemi risolti: generali-bizantini

Il problema della doppia spesa e' uno, dei principali, problemi di cui soffrono I sistemi distribuiti, rientra nella categoria piu' generale dei *byzantine faults* (fallimenti bizantini). Questa categoria nasce nel 1982 quando in un research paper di Leslie Lamport viene coniata la metafora dei *generali bizantini* per concettualizzare e rendere comprensibili le difficolta' che stavano affrontanto I nascenti sistemi distribuiti. La metafora e' la seguente.

Dei generali bizantini, ognuno a capo di una armata stanno accerchiando una fortezza nemica; questa fortezza puo' resistere all'attacco di una parte dei generali con le rispettive armate ma non sarebbe in grado di resistere ad un attacco congiunto. I generali sono seperati e presidiano diversi punti d'attacco, possono comunicare fra di loro tramite messaggieri a cavallo, il loro scopo e' raggiungere un accordo sull'ora e il giorno in cui attaccare per poter espugnare la fortezza. Le difficolta' in cui incorrono questi generali sono molteplici: un messaggero potrebbe essere catturato e addirittura sostituito con uno falso dai nemici, potrebbero inoltre esserci fra I generali dei traditori che potrebbero dare comunicazioni false per far fallire l'attacco. Come possono I generali raggiungere un accordo che gli permetta portare l'attacco vincente con sicurezza? Ovvero come puo' esserre raggiunto un *consenso*?[figura generali bizantini]



Nella metafora basta sostituire ai generali un sistema di computer e, I problemi legati all'affidabilita' delle comunicazioni e dell'onesta' di un partecipante , sono riconducibili a sistemi informatici. Questi problemi sono appunto I *byzantine faults*, piu' un sistema riesce ad evitare e mitigare questi problemi maggiore e' la sua *byzantine failure tollerance* (tolleranza ai fallimenti bizantini).

Le BC risolvono questi problemi tramite un protocollo del consenso, questo protocollo permette il raggiungimento di un accordo fra parti che non si fidano reciprocamente. [41]

## Rimozione degli intermediari

Invece di un intermediario terzo fidato che convalida le transazioni tramite i propri server con autorità (voto singolo), una rete peer-to-peer di computer che eseguono il protocollo blockchain convalida le transazioni per consenso (voto a maggioranza). Il protocollo blockchain, formalizza regole di consenso predefinite per la convalida delle transazioni sulla rete P2P, come *regole di governance* imposte dal codice del software stesso, che permette una gestione e applicazione automatica delle transazioni di tutti i partecipanti alla rete.

Nel caso di Bitcoin ad esempio, invece della convalida delle transazioni finanziarie da parte della banca controllando il saldo digitale di chi partecipa alla transazione sul proprio server, una rete P2P di computer che eseguono il protocollo bitcoin, I nodi miner, convalida le transazioni con il consenso della maggioranza. Le regole di consenso della rete Bitcoin governano il modo in cui i partecipanti alla rete interagiscono tra loro. Definiscono:

- A quali condizioni è valida una transazione.
- Costi di transazione relativi all'invio di denaro.
- Meccanismo di incentivazione, involvente la teoria dei giochi, per la convalida delle transazioni con un token crittografico.
- Regole su come modificare le attuali regole di consenso.

## Gli smart contract

La BC e' stata inizialmente progettata solo per denaro P2P. Ma presto ha mostrato il potenziale per essere utilizzata per qualsiasi tipo di transazione di valore P2P su Internet. Il progetto Ethereum ha quindi introdotto l'idea di disaccoppiare il *livello del contratto* dal *livello blockchain*, nel livello del contratto il ledger (libro mastro) viene utilizzato da *contratti intelligenti* che attivano automaticamente le transazioni, che sono a livello blockchain, quando vengono soddisfatte determinate condizioni predefinite. Disaccoppiando I due livelli, blockchain come Ethereum mirano a fornire un ambiente di sviluppo più flessibile rispetto alla blockchain di Bitcoin.

Questi contratti intelligenti (smart contract) sono codici in esecuzione su una rete blockchain, dove le risorse digitali sono controllate dal codice del contratto che implementa regole arbitrarie. Si possono implementare accordi contrattuali simili in tutto e per tutto a contratti legali.

Se e quando tutte le parti dello smart contract soddisfano le regole arbitrarie predefinite, lo smart contract eseguirà automaticamente la/le transazione. Questi contratti intelligenti mirano a fornire una sicurezza delle transazioni superiore al diritto contrattuale tradizionale e ridurre i costi di transazione, di coordinamento e applicazione. [23]

I contratti intelligenti possono essere utilizzati per semplici transazioni economiche come l'invio di denaro. Possono essere utilizzati anche per registrare qualsiasi tipo di proprietà e diritti di proprietà come i registri immobiliari e la proprietà intellettuale, o per gestire il controllo intelligente degli accessi nelle applicazioni dell'economia della condivisione ,es. car sharing ecc. Inoltre, i contratti intelligenti possono essere utilizzati per transazioni più complesse come governare un gruppo di persone che condividono gli stessi interessi e obiettivi. Le organizzazioni autonome decentralizzate, DAO, sono un esempio di contratti intelligenti più complessi.

Con blockchain e contratti intelligenti, si puo' immaginare un mondo in cui i contratti sono incorporati nel codice digitale e archiviati in database condivisi e trasparenti, dove sono protetti da

eliminazioni, manomissioni e revisioni. In questo mondo, ogni accordo, ogni processo, attività e pagamento avrebbe una registrazione e una firma digitali che potrebbero essere identificate, convalidate, archiviate e condivise.

Intermediari come avvocati, broker, banchieri e amministratori pubblici potrebbero non essere più necessari. Individui, organizzazioni, macchine e algoritmi effettuerebbero transazioni e interagirebbero liberamente tra loro con pochi attriti e una frazione dei costi di transazione correnti. Pertanto, blockchain e contratti intelligenti:

- Ridurrebbe radicalmente i costi di transazione (burocrazia) grazie al consenso della condiviso e al codice autoapplicabile.

- Aggirerebbe i tradizionali dilemmi principale-agente delle organizzazioni, fornendo così un sistema operativo per ciò che alcuni chiamano "fiducia senza fiducia", ovvero organizzazioni con gerarchia liquida. Ciò significa che non ci si dovrebbe fidare di persone e organizzazioni, ma del codice, che è open source e fornisce processi trasparenti.

## Prospettive per il futuro e possibili campi applicativi

Blockchain può risolvere molti problemi del mondo reale e offrire un modello di business e una struttura economica migliori per tutti noi. Esploriamo questi problemi e le possibili soluzioni offerte dalla blockchain.

Siamo più abituati a sentire parlare di blockchain insieme a Bitcoin e Cryptocurrency, ma grazie proprio agli smart contract visti precedentemente sono possibili tante nuove applicazioni che vanno ben oltre le risorse digitali. In effetti, la blockchain è di gran lunga una delle tecnologie più dirompenti e rivoluzionarie del nostro tempo.

Secondo i rapporti, si stima che la spesa globale per blockchain aumenterà e si prevede che supererà i 15,9 miliardi di dollari entro il 2023.

Inoltre, quasi ogni settore industriale sta cercando di adottare la tecnologia cogliendone i benefici. Questo inizia ad accadere per le aziende finanziarie, persino per le industrie dei media e l'assistenza sanitaria. Data la sua struttura unica che la rende super sicura e resistente a qualsiasi tipo di modifica dei dati, è proprio nei settori che la richiedono maggiormente che si stanno sviluppando applicazioni.

Innanzitutto, bisogna ricordare che non esiste un'autorità centrale che governa la blockchain. Questo lo rende un sistema completamente democratizzato. Inoltre, poiché le informazioni fornite sono immutabili, possono essere facilmente condivise e rese aperte al pubblico consentendo livelli di trasparenza senza precedenti.

Questi due fattori rendono la blockchain letteralmente la prossima rivoluzione di Internet, internet 3.0. Vediamo quali sono i principali problemi risolvibili dalla blockchain.

### Pagamenti internazionali

Allo stato attuale i pagamenti internazionali attraverso i canali bancari sono problematici. È un processo a più fasi che coinvolge molti intermediari. Inoltre, ogni fase del processo richiede molto tempo e richiede anche una notevole quantità di denaro.

Infatti, secondo la Banca Mondiale, la commissione media di transazione per i pagamenti mondiali è di circa il 7%.

La Blockchain aiuta a snellire l'intero processo eliminando tutti gli intermediari e le procedure lunghe, quindi l'onere di ritardi non necessari.

Utilizzando il suo registro distribuito sicuro e sofisticato, una volta che una transazione viene registrata, il pagamento viene trasferito quasi istantaneamente alla parte ricevente. Poiché la

transazione non può essere annullata o modificata, garantisce anche una migliore responsabilità e sicurezza rispetto al sistema attualmente utilizzato.

Secondo Deloitte, per le transazioni effettuate utilizzando blockchain, le parti coinvolte sostengono dal 40% all'80% in meno di commissioni rispetto ai metodi tradizionali. Non solo, ma tutte le transazioni vengono elaborate e completate nel giro di 4-6 secondi invece di dover attendere per giorni.

Quando si includono anche i vantaggi di una maggiore sicurezza e trasparenza, la blockchain è sicuramente una soluzione migliore per le transazioni monetarie globali.

Attualmente, un gruppo di società finanziarie utilizza la blockchain per risolvere questo problema. I nomi importanti nel campo includono:

- *Circle*: consente agli utenti di trasferire denaro in sicurezza (più valute supportate) in oltre 29 paesi.
- *Ripple*: aiuta le banche, i fornitori di servizi di pagamento ad effettuare gli scambi di risorse digitali, a inviare e ricevere denaro a livello globale in modo veloce e con commissioni minime.
- *WeTrade*: WeTrade offre una piattaforma digitale innovativa che consente alle banche e alle aziende di lavorare in modo collaborativo in tutta Europa, offrendo opportunità sicure, trasparenti.

#### Responsabilità con contratti e accordi tradizionali

Fondamentalmente creiamo contratti e accordi perché c'è una mancanza di fiducia tra le parti che effettuano transazioni. Di conseguenza, seguiamo questa procedura legale, di creazione di contratti, un processo a più fasi che coinvolge ancora più parti e aumenta notevolmente la burrocrazia.

Di conseguenza, molto tempo e denaro vengono spesi per gli intermediari.

Anche se lo consideriamo la norma e parte della creazione di contratti e accordi, tutti questi inconvenienti possono essere evitati sfruttando la tecnologia blockchain, in particolare i contratti intelligenti. Ebbene tramite gli Smart Contracts, una soluzione ingegnosa, potrebbero essere sostituiti in futuro I contratti e gli accordi tradizionali.

Poiché i contratti intelligenti sono archiviati sulla blockchain, i termini e le condizioni non possono essere modificati o manipolati. Inoltre, tutto è aperto e disponibile al pubblico, consentendo una maggiore trasparenza.

Ci sono tantissime startup e aziende che hanno già adottato smart contract in cambio di accordi tradizionali. Ecco alcune menzioni degne di nota:

- *Tradelanse*: aiuta a digitalizzare la catena di approvvigionamento facilitando accordi e contratti tradizionali.
- *Tradeix*: fornisce valore a finanziatori alternativi, banche, fornitori di valore aggiunto

#### Gestione e protezione dei dati dei pazienti nelle organizzazioni sanitarie

Il problema: gli attuali sistemi sanitari utilizzano sistemi centralizzati per registrare e gestire i dati dei pazienti. Questi possono essere meno sicuri e soggetti a fallo, ridondanze e inconsistenze.

Ci sono stati numerosi casi di violazioni dei dati negli ospedali in cui sono stati rubati i dati dei pazienti come le informazioni della carta di credito e le registrazioni dei test genomici. In effetti, tra il 2009 e il 2017, oltre 176 milioni di cartelle cliniche dei pazienti sono state compromesse a causa di violazioni dei dati.

Inoltre, lo stato attuale della comunicazione interospedaliera dei dati dei pazienti è un completo disastro. Molte volte i pazienti scelgono di cambiare ospedale. A volte, a causa delle emergenze, vengono ricoverati in ospedali in una rete diversa.

Poiché l'attuale sistema non facilita lo scambio rapido e semplice dei record dei pazienti dal loro fornitore di cure primarie, sono costretti a creare una nuova cartella clinica, che è una perdita di tempo e denaro.

Blockchain può aiutare a creare un registro decentralizzato dei dati dei pazienti che è trasparente e pubblico per l'accesso a tutti gli ospedali. Inoltre, non c'è modo di corrompere i dati, quindi l'intero problema della comunicazione interospedaliera potrebbe essere risolto.

Poi viene la sicurezza, nonostante sia trasparente e pubblica, la blockchain può anche aiutare a mantenere privati i tuoi dati nascondendo la tua identità con codici sicuri che proteggeranno tutti i tuoi dati sensibili. Ricordiamo che è necessario fornire una chiave privata e solo allora le informazioni saranno accessibili.

Molte aziende adottano la blockchain nel settore sanitario per fornire una migliore gestione e sicurezza dei dati dei pazienti. Ecco un paio di nomi importanti:

- *BurstIQ*: L'azienda fornisce soluzioni Blockchain per aiutare le aziende sanitarie a gestire enormi quantità di dati dei pazienti in modo sicuro e protetto.
- *MedicalChain*: aiuta a mantenere l'integrità delle cartelle cliniche. Medici, ospedali e laboratori possono richiedere informazioni sui pazienti, ma l'identità del paziente rimane al sicuro e protetta da fonti esterne.

### Supply Chain

La *Supply Chain Management* si riferisce alla pianificazione ed esecuzione di tutti i processi correlati che portano alla distribuzione di un prodotto finito. In generale, costituisce una rete di entità, individui o aziende che partono dai fornitori di materie prime, passando ai produttori di prodotti, e poi fino ai distributori.

Una catena di fornitura ben ottimizzata garantisce la massima produttività con attività fraudolente e costi generali ridotti. Le implementazioni nel mondo reale sono un lavoro complicato e frenetico. Sono stati compiuti sforzi di automazione utilizzando AI e Machine Learning, ma la blockchain promette di essere un punto di svolta in questo settore.

Uno dei principali vantaggi che la blockchain apporta alla gestione della catena di approvvigionamento sarebbe l'interoperabilità.

La trasparenza nella condivisione dei dati garantisce che tutti siano sincronizzati, a partire dai produttori ai rivenditori, ai fornitori e persino agli appaltatori. A sua volta, questo aiuta a ridurre eventuali conflitti e ritardi nelle operazioni.

Inoltre, poiché tutti i prodotti possono essere tracciati in tempo reale, riduce i rischi di smarrimento o di merci che si bloccano nella catena di approvvigionamento.

Blockchain inoltre offre un altro livello di scalabilità, con database di grandi dimensioni disponibili da più località in tutto il mondo. Ciò significa che non importa quanto sia grande una attività o quanto sia diffusa in tutto il mondo, utilizzando la blockchain si è in grado di gestire la catena di approvvigionamento senza sforzo.

Alcuni esempi di aziende che utilizzano blockchain per una gestione efficiente della supply chain sono:

- *Origintrail*: offrono un protocollo di scambio dati che può essere utilizzato da catene di approvvigionamento interconnesse.
- *Openport*: una piattaforma Blockchain collega direttamente spedizionieri e corrieri, riducendo i costi e ottimizzando la gestione della catena di approvvigionamento.
- *Walmart*: uno dei più grandi negozi al dettaglio, utilizza la blockchain per migliorare la propria catena di approvvigionamento.

## 2. LA BLOCKCHAIN DI ETHEREUM

Ethereum è una BC pubblica per applicazioni decentralizzate che opera su scala globale, viene spesso accostata a Bitcoin, tuttavia, oltre alla sua criptovaluta Ether (ETH), ci sono pochissime somiglianze.

Ethereum fa un uso più ampio della tecnologia blockchain. Consente agli utenti di sviluppare ed eseguire ogni tipo di codice, memorizzarlo nel ledger e di renderlo disponibile sotto forma di smart contract a tutti i partecipanti alla piattaforma etherium.

Ethereum consente a persone di tutto il mondo di interagire senza la necessità di intermediari. Di conseguenza, le transazioni monetarie e lo scambio di servizi, informazioni, conoscenze e beni possono avvenire a un costo inferiore e senza le tradizionali imposizioni delle istituzioni centralizzate. [25]



### ECOSISTEMA DI ETHEREUM

Ethereum è una rete di computer open source, affidabile e decentralizzata che utilizza un linguaggio di codifica unico e una tecnologia blockchain per scambiare servizi e applicazioni tramite un token (criptovaluta) chiamato "Ether". Al momento ci sono oltre 8.000 computer (operatori) in tutto il mondo impegnati in questo processo.

L'intera rete Ethereum funziona come una blockchain programmabile. Utilizza un protocollo peer-to-peer per consentire agli utenti di creare applicazioni o "contratti intelligenti", che possono aggiungere alla blockchain ed eseguire inviando loro Ether.

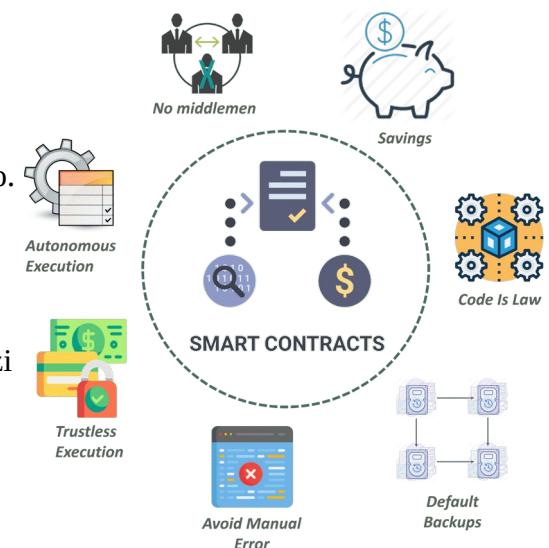
Come afferma il sito web della piattaforma: "su Ethereum, puoi scrivere codice che controlla il valore digitale". I codici sono chiamati "contratti intelligenti" e poiché la rete è pubblica e globale, chiunque può creare ed eseguire un contratto intelligente pagando Ether.

Una piattaforma decentralizzata significa che ogni contratto intelligente su di essa è archiviato su computer/server in più posizioni. La veridicità ed affidabilità del codice di questi contratti è dimostrata dall'intera rete che segue un protocollo del consenso.

Ethereum non è sotto il controllo di alcuna autorità governativa o istituto bancario. La piattaforma Ethereum è completamente autonoma e la sua autenticità è data dalle migliaia di computer e volontari che ci lavorano in tutto il mondo. [27]

#### Smart Contracts di Ethereum

Gli Smart Contracts di Ethereum sono codici autoeseguibili predefiniti che gli utenti aggiungono alla blockchain e segnalano un accordo tra due o più parti. Funzionano sulla Ethereum Virtual Machine (EVM), la sulla totalità dei nodi (computer) partecipanti al progetto Ethereum in tutto il mondo. Quando vengono soddisfatti i requisiti dello smart contract, il codice viene eseguito senza l'approvazione di una terza entità. Ad esempio, è possibile costruire una piattaforma per freelancer su Ethereum, come Ethlance. Lì, professionisti e imprese indipendenti possono incontrarsi per scambiare servizi tramite contratti intelligenti. Si può scrivere un contratto per



dire che quando un creatore di contenuti incontra un determinato servizio, il pagamento concordato dall'azienda contraente viene trasferito all'account del libero professionista senza la necessità dell'approvazione della piattaforma e, cosa più importante, senza che la piattaforma prenda una commissione dal contratto come fanno tutte le piattaforme di freelance nella realtà'.

Questo è solo un esempio, ma ci sono infinite possibilità e scopi per l'utilizzo di contratti intelligenti. Sempre più aziende utilizzano Ethereum e la sua interpretazione della tecnologia blockchain per sviluppare app, giochi e software di programmazione.

Il protocollo Ethereum consente quindi agli utenti di creare ed eseguire qualsiasi contratto intelligente o tipo di transazione. Esso è soggetto ad aggiornamenti periodici che ne migliorano l'architettura e la sicurezza. [28]

## Token Ether

Ether è una valuta digitale, un token o una proprietà virtuale associata a Ethereum.

Gli sviluppatori che desiderano creare app su Ethereum devono pagare per renderla operativa, ai nodi della rete una certa quantità di Ether. Le persone che in seguito vorranno utilizzare quelle app potrebbero anche doverle pagare in Ether. Si può pagare i servizi in Ether o richiederli come pagamento al di fuori della piattaforma.

Come altre forme di criptovaluta, puoi scambiare Ether con denaro o altre criptovalute (es. Bitcoin). Alcuni servizi e siti di scambio di criptovaluta tendono ad alternare il nome di questi token tra Ether ed Ethereum, sebbene rappresentino la stessa cosa.

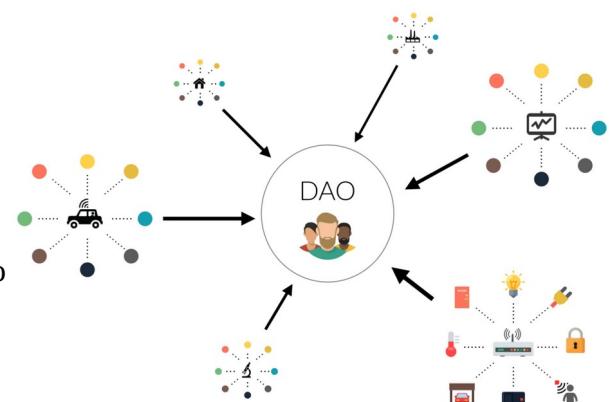
Nel corso del tempo si è assistito ad un aumento vertiginoso del valore degli eth, questo dimostra la popolarità crescente e la solidità del progetto in cui sempre più aziende e privati investono capitale. [29]



## Le DAO

Organizzazioni autonome decentralizzate, sono un modello di società efficace e sicuro per far lavorare persone che la pensano allo stesso modo ed hanno comuni obiettivi insieme da tutto il mondo.

Si possono pensare come ad aziende native di Internet di proprietà e gestite collettivamente dai suoi membri. Hanno bilanci incorporati a cui nessuno ha l'autorità di accedere senza l'approvazione del gruppo. Le decisioni sono governate da proposte e votazioni per garantire che tutti nell'organizzazione abbiano voce.



Non esiste un CEO che possa autorizzare le spese in base ai propri capricci e nessuna possibilità che un dirigente manipoli le informazioni. È tutto allo scoperto e le regole sulla spesa sono inserite nella DAO tramite il suo codice.

Avviare un'organizzazione con qualcuno che coinvolge fondi e denaro richiede molta fiducia nelle persone con cui lavori. Ma è difficile fidarsi di qualcuno con cui hai interagito solo su Internet. Con i DAO non è necessario fidarsi di nessun altro nel gruppo, ma solo del codice della DAO, che è trasparente al 100% e verificabile da chiunque.<sup>[31]</sup>

Questo apre così tante nuove opportunità per la collaborazione e il coordinamento globali.

DAO	Organizzazione tradizionale
Di solito piatto e completamente democratizzato.	Di solito gerarchico.
Votazione richiesta dai membri per eventuali modifiche da attuare.	A seconda della struttura, le modifiche possono essere richieste da un unico partito o può essere offerto il voto.
Voti conteggiati e risultato implementato automaticamente senza intermediario fidato.	Se la votazione è consentita, i voti vengono conteggiati internamente e il risultato della votazione deve essere gestito manualmente.
I servizi offerti sono gestiti automaticamente in modo decentralizzato	Richiede la manipolazione umana o l'automazione controllata centralmente, incline alla manipolazione.
Tutte le attività sono trasparenti e completamente pubbliche.	L'attività è tipicamente privata e limitata al pubblico.

## Funzionamento DAO

La spina dorsale di una DAO è il suo smart contract. Il contratto definisce le regole dell'organizzazione e detiene la tesoreria del gruppo. Una volta che il contratto è attivo su Ethereum, nessuno può modificare le regole se non con un voto. Se qualcuno cerca di fare qualcosa che non è coperto dalle regole e dalla logica del codice, fallirà. E poiché la gestione delle risorse è definita anche dallo smart contract, ciò significa che nessuno può spendere i soldi senza l'approvazione del gruppo. Ciò significa che le DAO non necessitano di un'autorità centrale. Il gruppo prende le decisioni collettivamente e i pagamenti vengono autorizzati automaticamente tramite votazioni.

Ecco alcuni possibili esempi di DAO:

Un ente di beneficenza: puoi accettare l'adesione e le donazioni da chiunque nel mondo e il gruppo può decidere come spendere le donazioni.

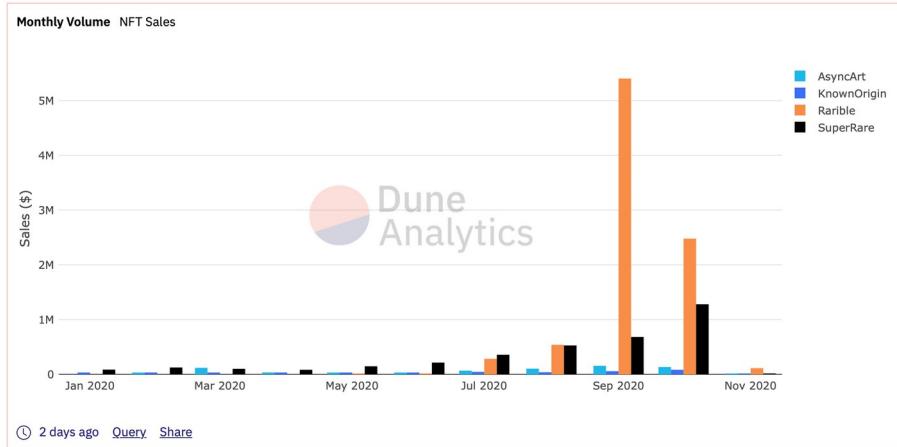
Una rete di liberi professionisti: potresti creare una rete di appaltatori che mettono insieme i loro fondi per gli spazi per uffici e gli abbonamenti software.

Ventures e sovvenzioni: potresti creare un fondo di venture capital che riunisce il capitale di investimento e vota sulle iniziative a sostegno. Il denaro rimborsato potrebbe essere successivamente ridistribuito tra i membri di DAO.

Ciò è possibile perché gli smart contract sono a prova di manomissione una volta che vengono pubblicati su Ethereum. Non si può semplicemente modificare il codice (le regole DAO) senza che le persone se ne accorgano perché tutto è pubblico.

## NFTs

Gli NFT stanno attualmente prendendo d'assalto il mondo dell'arte digitale e degli oggetti da collezione. Gli artisti digitali stanno vedendo le loro vite cambiare grazie alle enormi vendite a un nuovo cripto-pubblico. E le celebrità si stanno unendo a loro quando individuano una nuova opportunità per entrare in contatto con i fan. Ma l'arte digitale è solo un modo per utilizzare gli NFT. In realtà possono essere utilizzati per rappresentare la proprietà di qualsiasi risorsa unica, come un atto per un oggetto nel regno digitale o fisico.



### Cos'è un NFT?

Gli NFT sono token (oggetto digitale) che possiamo utilizzare per rappresentare oggetti unici, permettono di rappresentare cose come arte, oggetti da collezione e persino immobili. Possono avere un solo proprietario ufficiale alla volta e sono protetti dalla blockchain di Ethereum: nessuno può modificare la proprietà oppure copiare e incollare un NFT.

NFT sta per token non fungibile. Non fungibile è un termine economico che potresti usare per descrivere cose come i tuoi mobili, un file di una canzone o il tuo computer. Queste cose non sono intercambiabili per altri articoli perché hanno proprietà uniche.

Gli oggetti fungibili, d'altra parte, possono essere scambiati perché il loro valore li definisce piuttosto che le loro proprietà uniche. Ad esempio, ETH o dollari sono fungibili perché 1 ETH / \$ 1 USD è scambiabile con un altro 1 ETH / \$ 1 USD.

### L'Internet degli asset

NFT ed Ethereum risolvono alcuni dei problemi che esistono oggi in Internet. Man mano che tutto diventa più digitale, è necessario replicare le proprietà di oggetti fisici come la scarsità, l'unicità e la prova di proprietà. Per non parlare del fatto che gli articoli digitali spesso funzionano solo nel contesto del loro prodotto. Ad esempio, non puoi rivendere un mp3 di iTunes che hai acquistato o non puoi scambiare i punti fedeltà di una società con il credito di un'altra piattaforma anche se esiste un mercato.

Gli NFT hanno alcune proprietà speciali:

- Ogni gettone coniato ha un identificatore univoco.
- Non sono direttamente intercambiabili con altri token 1: 1. Ad esempio 1 ETH è esattamente uguale a un altro ETH. Questo non è il caso degli NFT.
- Ogni token ha un proprietario e queste informazioni sono facilmente verificabili.
- Vivono su Ethereum e possono essere acquistati e venduti su qualsiasi mercato NFT basato su Ethereum.

- Puoi facilmente provare che lo possiedi.
- Nessuno può manipolarlo in alcun modo.
- Puoi venderlo e se e' programmato il creatore originare guadagnerà i diritti di rivendita

E se crei un NFT:

- Puoi facilmente dimostrare di essere il creatore.
- Sei tu a determinare la scarsità.
- Puoi guadagnare royalty ogni volta che viene venduto.
- Puoi venderlo su qualsiasi mercato NFT o peer-to-peer. Non sei bloccato su nessuna piattaforma e non hai bisogno di nessuno che faccia da intermediario.

### Scarsità

Il creatore di una NFT deve decidere la scarsità del proprio asset. Ad esempio, un biglietto per un evento sportivo. Proprio come l'organizzatore di un evento può scegliere quanti biglietti vendere, il creatore di un NFT può decidere quante repliche esistono. A volte si tratta di repliche esatte, come 5000 biglietti di ingresso generale. A volte ne vengono coniate diverse molto simili, ma ognuna leggermente diversa, come un biglietto con un posto assegnato. In un altro caso, il creatore potrebbe voler creare un NFT in cui solo uno è coniato come oggetto da collezione raro speciale. In questi casi, ogni NFT avrebbe ancora un identificatore univoco (come un codice a barre su un "biglietto" tradizionale), con un solo proprietario. La scarsità prevista dell'NFT è importante e dipende dal creatore. Un creatore può voler rendere ogni NFT completamente unico per creare scarsità o avere motivi per produrre diverse migliaia di repliche, queste informazioni sono tutte pubbliche.

### Royalties

Gli NFT possono essere programmati per pagare automaticamente le royalty ai loro creatori quando vengono venduti. Questo è ancora un concetto in via di sviluppo ma è uno dei più potenti, già alcune piattaforme, come Foundation e Zora, supportano i diritti d'autore per i loro artisti.

Questo è completamente automatico, quindi i creatori possono semplicemente sedersi e guadagnare royalty mentre il loro lavoro viene venduto. Al momento con i sistemi attuali, capire le royalty è molto difficile e poco accurato: molti creatori non vengono pagati per ciò che si meritano questo non avviene con un NFT che ha una royalty programmata.

Gli NFT alimentano una nuova economia dei creatori in cui i creatori non cedono la proprietà dei loro contenuti alle piattaforme che usano per pubblicizzarli. La proprietà è incorporata nel contenuto stesso. Quando vendono i loro contenuti, i fondi vanno direttamente a loro. Se il nuovo proprietario poi vende l'NFT, il creatore originale può persino ricevere automaticamente le royalty. Ciò è garantito ogni volta che viene venduto perché l'indirizzo del creatore fa parte dei metadati del token, metadati che non possono essere modificati.

Gli NFT hanno riscontrato molto interesse da parte degli sviluppatori di videogiochi. Gli NFT possono fornire registrazioni di proprietà per oggetti di gioco, alimentare le economie di gioco e portare una serie di vantaggi ai giocatori. In molti giochi normali puoi acquistare oggetti da utilizzare nel tuo gioco, ma se quell'oggetto fosse un NFT potresti recuperare i soldi vendendolo. Si potrebbe anche realizzare un profitto se quell'oggetto diventa più desiderabile.

Gli sviluppatori di giochi, in quanto emittenti dell'NFT, potrebbero guadagnare una royalty ogni volta che un oggetto viene rivenduto sul mercato aperto. Questo crea un modello di business reciprocamente vantaggioso in cui sia i giocatori che gli sviluppatori guadagnano dagli NFT. Ciò significa anche che se un gioco non è più mantenuto dagli sviluppatori, gli oggetti raccolti rimangono di proprietà del giocatore che li ha trovati: gli oggetti nel gioco possono sopravvivere ai

giochi stessi. Anche se un gioco non viene più mantenuto, i tuoi oggetti saranno sempre sotto il tuo controllo. Ciò significa che gli oggetti di gioco diventano cimeli digitali e hanno un valore al di fuori del gioco.

Gli NFT possono anche rappresentare oggetti reali tangibili, un giorno si potrebbe acquistare un'auto o una casa utilizzando ETH e ricevere in cambio l'atto come NFT (nella stessa transazione). Il portafoglio Ethereum potrebbe addirittura diventare la chiave della tua auto o della tua casa: la tua porta viene sbloccata dalla prova crittografica della proprietà. Con beni preziosi come automobili e proprietà rappresentabili su Ethereum, puoi utilizzare gli NFT come garanzia nei prestiti decentralizzati (DEFI vedi dopo). Ciò è particolarmente utile se non sei ricco di contanti o di criptovalute ma possiedi oggetti fisici di valore.[32]

## DEFI

DeFi è un sistema finanziario aperto e globale costruito per l'era di Internet, un'alternativa a un sistema opaco, strettamente controllato e tenuto insieme da infrastrutture e processi vecchi di decenni, dà controllo e visibilità sui soldi, dà esposizione ai mercati globali e alternative alla tua valuta locale o alle opzioni bancarie. I prodotti DeFi offrono servizi finanziari a chiunque disponga



di una connessione Internet e sono in gran parte di proprietà e gestiti dai loro utenti. Finora decine di miliardi di dollari di criptovalute sono fluite attraverso le applicazioni DeFi e crescono ogni giorno. Con la DeFi, i mercati sono sempre aperti e non ci sono autorità centralizzate che possono bloccare i pagamenti o negarti l'accesso a un certo prodotto finanziario. I servizi che prima erano lenti e a rischio di errore umano sono automatici e più sicuri visto che sono gestiti da un codice che chiunque può ispezionare e controllare.

Uno dei modi migliori per vedere il potenziale della DeFi è capire i problemi che esistono oggi nel mondo della finanza:

- Ad alcune persone non è concesso l'accesso per creare un conto bancario o utilizzare servizi finanziari
- I servizi finanziari hanno un costo nascosto: i tuoi dati personali.
- I governi e le istituzioni centralizzate possono chiudere i mercati a piacimento e gli orari di negoziazione sono spesso limitati agli orari di lavoro di un fuso orario specifico
- I trasferimenti di denaro possono richiedere giorni a causa di processi umani interni.
- Ci sono premi per i servizi finanziari su cui istituti intermediari speculano, cercando di massimizzare il numero di sottoscrittori e non la validità del prodotto finanziario in sé
- Le istituzioni finanziarie sono chiuse e private: non puoi chiedere di vedere la cronologia dei loro prestiti, un registro dei loro beni gestiti e così via.

Con la DeFi invece:

- Controllo sui tuoi soldi, non sono detenuti dalle aziende.
- Sei tu a controllare dove vanno i tuoi soldi e come vengono spesi.
- Non devi fidarti delle società che potrebbe gestire male i tuoi soldi, come prestare a mutuatari rischiosi.
- I trasferimenti di fondi avvengono in pochi minuti.
- L'attività di transazione è anonima invece di essere strettamente collegata alla tua identità.
- DeFi è aperto a tutti, basta fare domanda per utilizzare i servizi finanziari.

- I mercati sono sempre aperti.
- Si basa sulla trasparenza: chiunque può esaminare i dati di un prodotto e controllare come funziona il sistema.

### Denaro programmabile

Una delle caratteristiche predefinite dei token su Ethereum è che chiunque può programmare la logica nei pagamenti. In questo modo si può ottenere il controllo e la sicurezza insieme ai servizi finanziari. Ciò consente di fare cose con le criptovalute come prestare e prendere in prestito, programmare pagamenti, investire in fondi indicizzati e altro ancora.

Applicando il principio del Peer-to-peer alla finanza significa che un mutuatario prenderà in prestito direttamente da un prestatore specifico, scelto da una lista di prestatori che forniscono fondi (liquidità) da cui i mutuatari possono prendere in prestito.

Ci sono molti vantaggi nell'usare un istituto di credito decentralizzato. Il prestito decentralizzato funziona senza che nessuna delle parti debba identificarsi. Il mutuatario deve invece fornire garanzie che il mutuante riceverà automaticamente se il prestito non viene rimborsato. Alcuni istituti di credito accettano persino NFT come garanzia, è possibile prendere in prestito denaro senza consegnare informazioni private. [30]

## DIFFERENZE CON BITCOIN

Mentre entrambe le reti Bitcoin ed Ethereum sono alimentate dal principio del ledger distribuito e della crittografia, le due differiscono tecnicamente in molti modi. Ad esempio, le transazioni sulla rete Ethereum possono contenere codice eseguibile, mentre i dati apposti alle transazioni di rete Bitcoin sono generalmente solo per scrivere note. Altre differenze includono il tempo di blocco (una transazione ether viene confermata in secondi rispetto ai minuti per bitcoin) e gli algoritmi di hashing su cui vengono eseguiti (Ethereum utilizza ethash mentre Bitcoin utilizza SHA-256). Le reti Bitcoin ed Ethereum inoltre sono diverse rispetto ai loro obiettivi generali. Mentre il bitcoin è stato creato come alternativa alle valute nazionali e quindi aspira a essere un mezzo di scambio e una riserva di valore, Ethereum è inteso come una piattaforma per facilitare contratti e applicazioni programmabili e immutabili pagabili tramite la propria valuta.

BTC ed ETH sono entrambe valute digitali, ma lo scopo principale di Ether non è quello di affermarsi come un sistema monetario alternativo, ma piuttosto quello di facilitare e monetizzare il funzionamento degli smart contract di Ethereum e della piattaforma dell'applicazione decentralizzata (dapp).

Ethereum è un altro caso d'uso per una blockchain e teoricamente non dovrebbe competere con Bitcoin.

### Consenso

Bitcoin utilizza l'algoritmo SHA-256. Questa equazione matematica richiede che i minatori dimostrino il loro lavoro attraverso calcoli avanzati. La rete regola automaticamente la sua difficoltà per garantire che i blocchi di transazioni vengano approvati solo a intervalli di dieci minuti. Questo approccio garantisce una strategia di emissione monetaria predittiva fino a quando l'ultimo Bitcoin non verrà estratto nel 2140.

Ethereum, come Bitcoin, attualmente utilizza un protocollo di consenso proof-of-work (PoW). Tuttavia, Ethereum utilizza l'algoritmo Ethash. Il fondatore Vitalik Buterin ha deciso di utilizzare questo meccanismo per ridurre il vantaggio delle grandi mining farm ASIC (Application Specific

Integrated Circuit) specializzate. I mining rig ASIC sono costruiti da zero per risolvere l'algoritmo SHA-256 utilizzato da Bitcoin. I critici dei minatori ASIC sostengono che questi rig costosi causano la centralizzazione nella rete Bitcoin.

Quando si confronta la transazione tramite il put delle reti, Ethereum risulta molto più avanti di Bitcoin. Bitcoin approva i blocchi ogni 10 minuti. Questi blocchi non contengono più di 1 MB di dati. Di conseguenza, Bitcoin è in grado di eseguire solo circa 7 transazioni al secondo. Questa bassa velocità di trasmissione dati è stata integrata nella codifica principale di Bitcoin per garantire che chiunque potesse utilizzare la rete.

La rete Ethereum è in grado di effettuare circa 15 transazioni al secondo. Queste funzionalità sono destinate a migliorare in modo significativo a seguito del prossimo aggiornamento di Ethereum 2.0. Questo aggiornamento spingerebbe le capacità di Ethereum più vicino a 100.000 transazioni al secondo secondo gli sviluppatori.

#### Premi per I miner

Ci sono diversi premi di mining pagati ai nodi su ciascuna rete. I minatori di Bitcoin ricevono una ricompensa di 6,5 BTC se sono il nodo che completa per primo l'equazione SHA-256 e aggiunge il blocco successivo alla blockchain. In confronto, i minatori di Ethereum ricevono una ricompensa di 2 ETH per la loro partecipazione alla convalida di blocchi di transazioni.

#### Limiti di emissione

Bitcoin limita la sua offerta a 21.000.000 di coin. Questa strategia garantisce che Bitcoin mantenga la scarsità nel mercato. Al contrario, non esiste un limite alla quantità di Ether (ETH). La rete deve continuare a produrre ETH a tempo indeterminato per coprire le tariffe del gas create dagli sviluppatori che eseguono la ethereum virtual machine (EVM). Attualmente, ci sono 114.467.625,91 ETH in circolazione.

#### Ethereum passaggio a PoS

È interessante notare che Ethereum dovrebbe fare un importante aggiornamento quest'anno a Ethereum 2.0. Questo hard fork collocherebbe ETH su una nuova blockchain che gira su un algoritmo Proof-of-Stake (PoS). Le reti PoS rimuovono i minatori e si affidano ai possessori di monete che puntano i loro token per convalidare lo stato della rete.

Le reti PoS sono molto più efficienti dal punto di vista energetico e più economiche da mantenere. Forniscono inoltre tempi di transazione più rapidi rispetto alle reti PoW. Soprattutto, non è necessario acquistare costose attrezature per il mining perché tutto ciò di cui si ha bisogno è un portafoglio (wallet) per puntare le tue monete su un sistema PoS.

#### Il lancio di ETH ICO vs Bitcoin

Bitcoin ha avuto un lancio tranquillo che è stato celebrato solo da pochi eletti nella comunità cypherpunk e di sviluppo che hanno preso nota di questa monumentale invenzione. È interessante notare che il viaggio di Bitcoin è iniziato ufficialmente con il suo primo blocco, il blocco genesi. Questo è il primo blocco della blockchain di Bitcoin. Anche se nessuno sa con certezza quanti Bitcoin ha estratto Nakamoto, le stime parlano di 1 milione di Bitcoin.

In confronto, Ethereum è entrato nel mercato con molto più clamore. L'ICO (Initial Coin Offering) di Ethereum ha raccolto 18 milioni di dollari. Ethereum ha continuato col lancio della prima DAO (Decentralized Autonomous Organization). Questo evento ha avuto luogo nell'aprile 2016. Il lancio

delle DAO ha rafforzato lo status di Ethereum e ha aiutato la rete a garantire \$ 150 milioni nella sua ICO pubblica. [24]

### Una breve storia di Ethereum

Nel novembre del 2013, un programmatore russo-canadese di 19 anni di nome Vitalik Buterin ha pubblicato il white paper di Ethereum. La sua proposta era quella di una piattaforma che potesse utilizzare la tecnologia blockchain per archiviare e implementare programmi informatici attraverso una rete internazionale di nodi distribuiti.

Nel gennaio 2014, Buterin ha annunciato pubblicamente lo sviluppo della piattaforma Ethereum. Allo stesso tempo, il team ha avviato un progetto ICO che ha accumulato \$ 18,4 milioni in meno di 7 mesi.

Nel maggio 2015 è stato rilasciato il primo testnet Ethereum "Olympic", seguito da "Frontier" come prima fase di sviluppo nel luglio dello stesso anno.

Il 14 marzo 2016 ha segnato l'uscita ufficiale di Homestead, la prima versione di Ethereum stabile. A giugno, a meno di tre mesi dal suo rilascio,

Dal 2016 al 2019, Buterin e il suo team hanno annunciato diversi aggiornamenti per rafforzare e proteggere il protocollo Ethereum originale: The Classic fork (25 ottobre 2016), The Metropolis Byzantium hard fork (16 ottobre 2017) e The Metropolis Constantinople hard fork ( 28 febbraio 2019).

### Come possedere Ether

Si possono acquisire Ether minandoli o acquistandoli con valuta tradizionale o altre criptovalute. In entrambi i casi, e' necessario un portafoglio elettronico per archiviare i token.

Se si desidera estrarre Ether, bisogna configurare un'unità di elaborazione grafica (GPU) e convenientemente unirsi a un pool di mining Ethereum. Il processo è simile al mining di Bitcoin, tranne per il fatto che la blockchain di Ethereum memorizza sia la cronologia delle transazioni che lo stato attuale della rete.

### I vantaggi dell'utilizzo di Ethereum

Ethereum apporta cambiamenti significativi alla tecnologia blockchain e fornisce vantaggi essenziali ai suoi utenti, come ad esempio:

- La rete Ethereum non può essere disattivata, il che significa che le app e le aziende che la utilizzano non potranno mai spegnersi
- È impenetrabile a modifiche da parte di terzi
- La sicurezza della rete è data dal consenso degli utenti sulla prova di lavoro e dal protocollo crittografico che è alla base di tutte le transazioni
- Ethereum non può essere manomesso da attori intermediari come governi o istituti bancari

### Gli svantaggi dell'utilizzo di Ethereum

Ethereum rimane una rete informatica altamente affidabile e sicura. D'altra parte, l'ossessione di mantenere i protocolli di sicurezza in esecuzione alla massima capacità danneggia gli utenti tradizionali che cercano di eseguire nuove applicazioni sulla blockchain. Di conseguenza, il costo di Ether rimane elevato e inaccessibile a molti sviluppatori.

Ethereum sta rapidamente sviluppando una fedele comunità di sviluppatori che abbracciano la possibilità di eseguire qualsiasi codice desiderino sulla sua blockchain.

Allo stato attuale, Ethereum è una delle rivelazioni più importanti della tecnologia blockchain e, molto probabilmente, la piattaforma numero 1 per le applicazioni decentralizzate del futuro.

## LA RIVOLUZIONE DI ETHEREUM

Al momento, quando vengono fornite delle informazioni personali (come nome e indirizzo) a un'azienda online, tali informazioni vengono memorizzate sul suo server.

Tutte le tue informazioni personali sono archiviate su centinaia di diversi server aziendali e governativi: nomi, numeri di telefono, saldi bancari, record di carte di credito, messaggi di testo, e-mail, cartelle cliniche e persino le foto.

C'è un'enorme rete invisibile di server informatici, piena fino all'orlo di tutte le nostre informazioni. La maggior parte di questi sistemi informatici sono gestiti da grandi aziende tecnologiche come Microsoft e Google, che li gestiscono per conto delle aziende con cui intrattieni rapporti d'affari. Questo sistema ha un grosso svantaggio: le informazioni personali in questi sistemi sono vulnerabili all'hacking. L'hacking si verifica sempre, perché ogni server agisce come un grande bersaglio pieno di preziose informazioni sui clienti. Tutte le aziende affrontano questo problema. Sono in competizione senza fine con gli hacker.

Esempi recenti di grandi hack che hanno portato al furto di molte informazioni sui clienti includono Dropbox, Walmart, Starbucks, Facebook, Uber, Equifax e Google, anche le aziende più grandi non possono proteggersi per sempre dall'hacking.

Ethereum è rivoluzionario perché, per la prima volta, consente ai sistemi informatici online di funzionare senza l'utilizzo di NESSUNA terza parte (come Google).

Non aver bisogno di una terza parte (come Google) per archiviare e trasferire le informazioni ha molti vantaggi. Senza intermediari, i sistemi informatici diventano più economici da gestire e anche più difficili da chiudere. Inoltre, le informazioni personali possono diventare più private perché le aziende non le memorizzano più (per sempre) sui loro server.

Invece di utilizzare il sistema informatico di una grande azienda come Google (un sistema centralizzato), Ethereum consente alle applicazioni software di funzionare su una rete di molti computer privati, un sistema decentralizzato.

I computer aziendali ei server cloud vengono sostituiti da una grande rete decentralizzata di molti piccoli computer gestiti da volontari di tutto il mondo. Scorrendo qualsiasi app store possiamo vedere un lungo elenco di applicazioni che si affidano a grandi aziende per eseguirle e memorizzare le nostre informazioni personali. Dalle banche alla salute ai giochi: tutte queste app utilizzano server centrali di grandi dimensioni. Anche la scelta di una app è controllata da terze parti come Apple e Google. Se a queste aziende non piace un'app, non sarà consentita nei loro app store.

Ethereum sta facendo l'opposto: vuole rimuovere terze parti dal modo in cui accedi e utilizzi le applicazioni, mettendo al comando la community degli utenti e sviluppatori. La grande idea alla base di Ethereum è che chiunque può utilizzare questa nuova rete decentralizzata per creare ed eseguire applicazioni decentralizzate. Non è necessaria alcuna autorizzazione perché non sono più necessarie terze parti. Ethereum come detto precedentemente e' una piattaforma tecnologica che consente a chiunque di eseguire applicazioni sulla sua rete globale.

Poiché queste applicazioni non utilizzano più un server centrale, sono note come applicazioni decentralizzate (o dApp). Ethereum non è controllato da Vitalik e dal suo team o da qualsiasi altra persona, azienda o governo, e' gestito dalla comunità dei suoi utenti da tutto il mondo e offrono volontariamente i loro computer per far funzionare la rete Ethereum. Le informazioni su questa rete vengono inviate direttamente da persona a persona, invece che da persona a azienda a persona.

Questo è noto come sistema peer-to-peer (P2P): non esiste un controllo centrale e nessuno degli utenti della comunità di Ethereum ha bisogno di conoscersi affinché il sistema funzioni. Puoi utilizzare Ethereum online senza bisogno di incontrare o fidarti di nessun'altra persona. [35]

La visione di Ethereum è quella di creare un "World Computer", un'enorme rete di molti computer privati che eseguono tutte le future applicazioni Internet senza terze parti (come Google e

Microsoft). Ethereum sta reinventando il funzionamento di Internet. Sta togliendo potere e controllo alle grandi aziende tecnologiche e li sta mettendo nelle mani degli utenti.

La rimozione di terze parti ha molti vantaggi, ad esempio: poiché Ethereum è una rete di computer gestita dalla comunità dei suoi utenti, le informazioni personali non vengono più archiviate sui server centrali delle grandi aziende, in attesa di essere hackerate, vendute o corrotte.

La decentralizzazione significa anche che nessuno può essere escluso dall'utilizzo delle applicazioni Ethereum. Persone di tutti i paesi e background possono utilizzarli senza approvazione o autorizzazione. Anche le stesse applicazioni Ethereum non hanno bisogno del permesso per esistere. Non possono essere rimossi o censurati da alcun app store: un sistema aperto e chiunque abbia una connessione Internet. Due persone che si scambiano messaggi sui loro smartphone, tradizionalmente, userebbero app di messaggistica come WhatsApp, Facebook o Skype, che sono gestite da grandi aziende. Una volta inviato, il messaggio passa effettivamente dal mittente a un server aziendale e solo successivamente al destinatario.

Poiché le aziende controllano le app di messaggistica tradizionali, controllano e archiviano anche tutti i messaggi inviati, oltre a qualsiasi altra informazione personale, come le foto che allega. Tutto ciò viene conservato per sempre. Questo può diventare un problema se i server aziendali vengono violati, se i servizi vengono interrotti o se le aziende vendono le tue informazioni agli operatori di marketing. Se le due persone utilizzassero invece un'app di messaggistica Ethereum decentralizzata, quando qualcuno preme "Invia", il messaggio va dal mittente alla rete Ethereum e da lì al destinatario, il messaggio è nascosto dalla rete e dai suoi algoritmi di crittografia. Anche se Ethereum utilizza una rete globale e pubblica di computer, solo il mittente e il destinatario possono vedere i loro messaggi. Le due persone che inviano messaggi non noterebbero alcuna differenza tra le app tradizionali centralizzate e le nuove app decentralizzate. Dal loro punto di vista, l'app Ethereum funziona esattamente come un'app di messaggistica tradizionale. Le app decentralizzate funzionano automaticamente, senza intermediari, perché utilizzano gli Smart Contracts, costituiti da regole concordate inserite nel codice e quindi le applicano automaticamente. Dopo che le regole sono state concordate, nessuno può modificarle o manipolarle, garantendo l'equità a tutte le parti. Lo Smart Contract agisce da solo e fa esattamente ciò per cui è programmato, qualunque cosa accada. Senza che siano possibili interferenze, tutti i partecipanti possono fidarsi dello Smart Contract, anche se potrebbero non fidarsi l'uno dell'altro.

Molte applicazioni Ethereum e Smart Contract richiedono uno scambio di denaro. Poiché Ethereum è completamente digitale, ha bisogno di un metodo di pagamento integrato per farlo.

Nel sistema Ethereum, tutti i pagamenti vengono effettuati utilizzando la propria valuta digitale: Ether. Ecco come funzionano insieme Ethereum e la valuta Ether: ogni volta che il denaro viene trasferito da un Ethereum Smart Contract, Ether viene utilizzato per quella transazione.

A differenza delle valute tradizionali come il dollaro USA, Ether è digitale al 100%. Per questo motivo, può essere "memorizzato" direttamente nel codice dello Smart Contract. Ciò consente allo Smart Contract di inviare e ricevere denaro automaticamente.

Oltre agli Smart Contracts, Ether viene utilizzato anche per pagare tutti i volontari che aiutano a far funzionare il sistema Ethereum decentralizzato con i loro computer. Ogni partecipante riceve Ether, poiché paga l'elettricità e usa il proprio hardware per aiutare a far funzionare il sistema. Ether ha anche un'importante funzione di sicurezza per la piattaforma Ethereum.

Le app decentralizzate devono pagare una piccola tariffa Ether ogni volta che vengono eseguite e utilizzano la rete. Ciò impedisce alle persone di creare app dannose progettate per essere eseguite ripetutamente e rallentare il sistema. A causa della tariffa Ether, l'esecuzione di programmi scadenti diventa troppo costosa. Questa commissione è giustamente chiamata "gas" perché è il carburante che gestisce le applicazioni di Ethereum. [34]

Molte nuove applicazioni sono possibili con Ethereum, potrebbero essere necessari alcuni anni prima che le applicazioni Blockchain decentralizzate siano performanti e convenienti come le app centralizzate che usi oggi, ma Ethereum sta crescendo molto rapidamente

La rimozione degli intermediari cambierà molti settori nei prossimi anni e potrebbe comportare la perdita di posti di lavoro, tuttavia c'è da considerare che si potrebbero creare nuovi lavori che prima non esistevano. Di seguito vengono esposti alcuni esempi di applicazioni decentralizzate (Dapp) possibili su Ethereum.

#### Esempio 1 - Marketplace P2P

Con Ethereum, è possibile creare mercati in cui gli acquirenti possono pagare direttamente i venditori, senza la necessità di intermediari. Ad esempio, artisti che vendono musica direttamente ai loro fan, invece di utilizzare intermediari come Apple e Amazon. La musica potrebbe diventare più economica e gli artisti avrebbero comunque un ritorno più ampio.

Uno Smart Contracts può automaticamente garantire che tutte le parti facciano ciò che hanno concordato, che i pagamenti vengano effettuati e che i prodotti vengano ricevuti. Una volta terminati i download o consegnati i pacchetti, Ethereum può pagare automaticamente il venditore.

In questo esempio Ethereum rimuove effettivamente due intermediari tipici: le piattaforme (come eBay ed Etsy) che addebitano le commissioni, più le carte di credito (come Visa e MasterCard) che addebitano le commissioni di transazione.

#### Esempio 2 - Sharing Economy

Con Ethereum, è possibile trasformare le cose che le persone possiedono ma che usano raramente in proprietà che producono reddito affittandole. L'assicurazione e la protezione contro il furto potrebbero essere automaticamente integrate in questi sistemi per la massima tranquillità.

Gli Smart Contracts possono essere creati per addebitare automaticamente a qualcuno che prende in prestito, sistemi di sharing economy come Uber e AirBnb potrebbero funzionare attraverso sistemi decentralizzati peer-to-peer, senza che nessuna azienda addebiti commissioni.

### 3. La tecnologia di Ethereum

Per comprendere a pieno i possibili utilizzi della tecnologia Ethereum sono necessarie oltre alle conoscenze precentemente trattate (*crittografia a chiave pubblica, funzione di hash, protocolli del consenso, mining*) anche quelle sul software.

Prima di procedere con l'esposizione del caso di studio affrontato, accennero' alle tecnologie chiave necessarie per lavorare alla creazione di una applicazione sulla BC di ethereum, ed approfondiro' il linguaggio *Solidity*, alla base dello sviluppo degli smart contract ethereum.

#### Macchina virtuale Ethereum

La macchina virtuale Ethereum è la principale innovazione del progetto Ethereum. Questa è una macchina virtuale progettata per essere eseguita da tutti i partecipanti nella rete peer to peer, può leggere e scrivere su una blockchain sia codice eseguibile che dati, verificare le firme digitali ed è in grado di eseguire qualsiasi codice (completezza di Turing) . Esegirà il codice solo quando riceve un messaggio verificato da una firma digitale (tramite chiave privata) e le informazioni memorizzate sulla blockchain indicano che è stato appropriato farlo.

Ethereum è una rete peer to peer in cui ogni peer (nodo) archivia la stessa copia di un database (la blockchain) ed esegue una macchina virtuale Ethereum per mantenere e modificare il proprio stato, cio' rende la blockchain etherium comparabile anche ad un sistema operativo distribuito, in grado di eseguire programmi, gli smart contract, sul hardware in questo caso distribuito. [33]

#### Metamask

Per utilizzare i servizi riguardanti la blockchain Ethereum e' necessario un *wallet*, questo viene usato per memorizzare e proteggere le chiavi private che identificano gli "attori" che agiscono attivando gli smart contract, ricevendo o mandando fondi sulla BC. Tramite il wallet e' possibile avere un'interazione diretta con la blockchain, mantenendo le proprie chiavi al sicuro in modo crittografico.

Affinché le transazioni blockchain di criptovaluta siano correttamente verificate, devono essere firmate con una chiave privata e verificate utilizzando una chiave pubblica. Solo quando ciò si verifica la blockchain viene aggiornata con un nuovo blocco di informazioni. Queste chiavi private sono uniche per un individuo e, come suggeriscono i loro nomi, devono rimanere private nello stesso modo in cui mantieni private le tue password. Per proteggere le tue chiavi private da altri, devi archiviare le tue chiavi private in portafogli crittografici.

Proprio come le chiavi private sono vitali per le transazioni, i portafogli crittografici sono vitali per garantire la sicurezza delle tue chiavi private e, quindi, della tua criptovaluta.

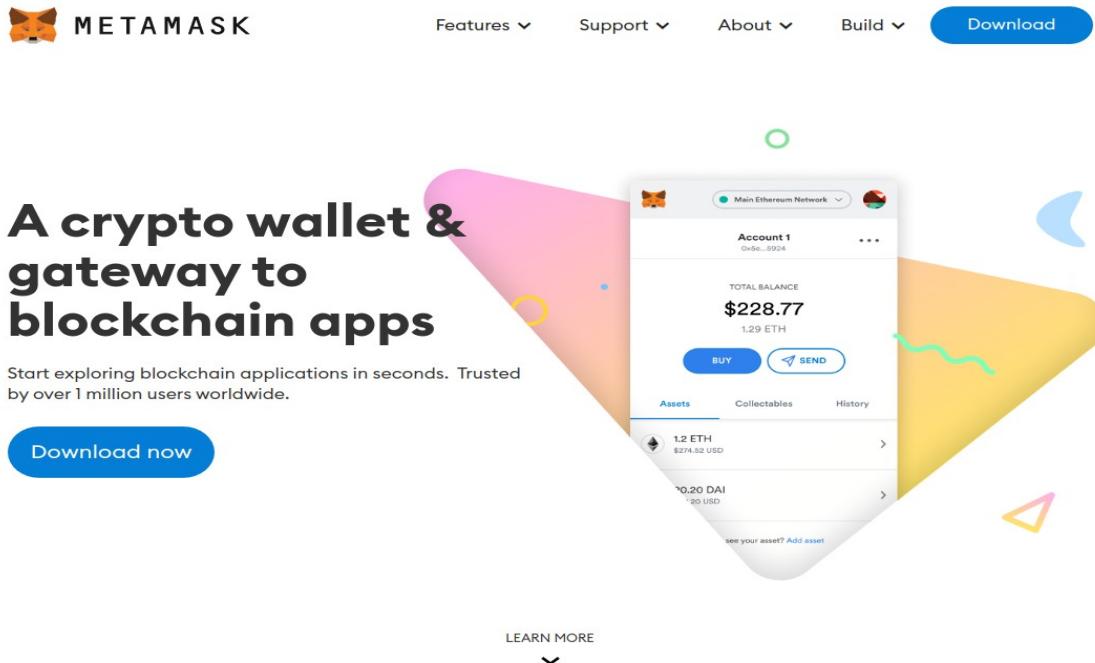
Il wallet MetaMask è un "ponte" che ti connette a tutti i servizi forniti dalla rete Ethereum, come l'accesso ad applicazioni decentralizzate (DApps), servizi di finanza decentralizzata (DeFi) e le borse di scambio crypto decentralizzate (Decentralize Exchange (DEX)).

MetaMask è uno dei portafogli di criptovalute più popolari oggi sul mercato, è open source (software a cui tutti possono avere accesso). Funziona e supporta la blockchain di Ethereum e tutti i token ERC-20 che girano sulla sua rete.

MetaMask funziona come un'applicazione se stai usando uno smartphone o come un'estensione se stai usando un browser web. Chrome, Firefox e Brave sono i browser attualmente supportati.

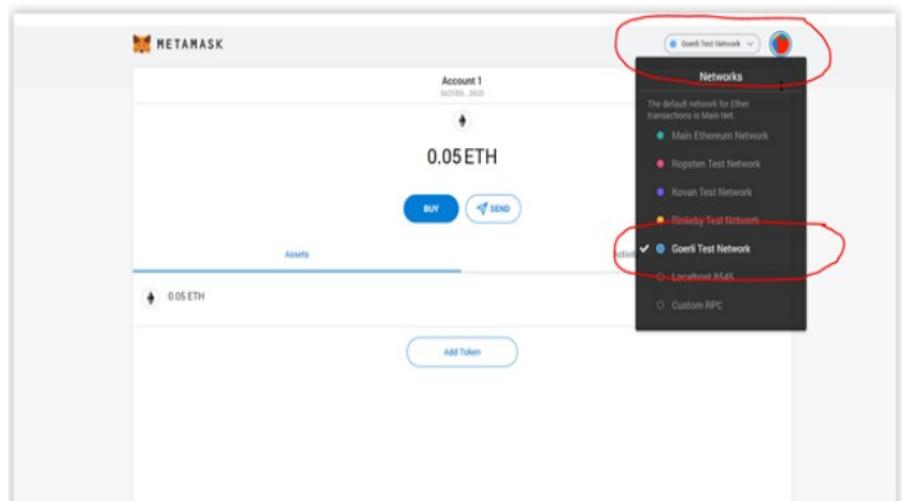
MetaMask non solo archivia le tue criptovalute, ma funge anche da mezzo per interagire con DApp nella finanza decentralizzata (DeFi).

Una volta arrivati sulla sua pagina ufficiale e' molto intuitivo riuscire ad installarlo come plugin al nostro browser.



Una volta installato bisogna settare la password e la frase di recupero, quest'ultima serve per recuperare il controllo sui nostri account in caso di compromissione degli stessi.

Una volta avviato notiamo che abbiamo a disposizione diverse blockchain a cui accedere e non una sola[Fig.], oltre alla blockchain vera e propria di etherium, detta *mainnet*, ci sono diverse blockchain non ufficiali che replicano in modo piu' o meno fedele il funzionamento della blockchain di ethereum, queste sono state create a scopo di test e vengono dette *testnet*.



Le testnet servono per non intasare di transazioni la blockchain principale e per evitare di dover pagare *gas* per testare la propria applicazione. Il *gas*, e' un costo che viene pagato per eseguire una transazione sulla BC, nel corso della trattazione del linguaggio solidity verrà trattato maggiormente nel dettaglio. Sulle testnet e' possibile acquisire eth (moneta della BC di ethereum) senza costi in modo illimitato, e con gli eth e' possibile pagare il *gas*.

## Traking con etherscan

Etherscan

Home Blockchain Tokens Resources More Sign In

## The Ethereum Blockchain Explorer

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

ETHER PRICE \$2,423.62 @ 0.0685 BTC (-2.97%)

MARKET CAP \$281,310,046,685.00

TRANSACTIONS 1,152.17 M (15.5 TPS)

DIFFICULTY 7,603.44 TH

MED GAS PRICE 23 Gwei (\$1.17)

HASH RATE 594,161.68 GH/s

ETHEREUM TRANSACTION HISTORY IN 14 DAYS

May 14 May 21 May 28

### Latest Blocks

Bk	12528352 26 secs ago	Miner F2Pool Old 285 txns in 1 sec	2,37343 Eth
Bk	12528351 27 secs ago	Miner Oxf541c3cd1d2df40... 135 txns in 5 secs	2,34121 Eth
Bk	12528350 31 secs ago	Miner Oxbcc817f057950b... 103 txns in 6 secs	2,35474 Eth

### Latest Transactions

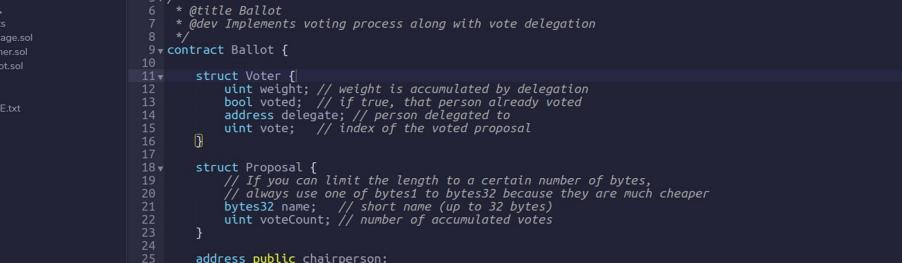
Tx	0x12e8e6d5c... 26 secs ago	From 0x6be7c3d3be71c2... To 0xa090e606e30bd...	0.00508 Eth
Tx	0x2a188c16d... 26 secs ago	From 0x36219b8182bf64... To 0xa090e606e30bd...	0.00508 Eth
Tx	0x96b5d7b6c... 26 secs ago	From 0xf8dedcb589cf6b7... To 0xa090e606e30bd...	0.00508 Eth

Etherscan è un dei principali *BlockExplorer* per la BC di Ethereum. Un BlockExplorer è un motore di ricerca che consente agli utenti di cercare, confermare e convalidare facilmente le transazioni che hanno avuto luogo sulla Blockchain di Ethereum.

La Blockchain di Ethereum ha un registro pubblico (come un database decentralizzato) che Etherscan.io indicizza e rende disponibili queste informazioni attraverso il sito. Etherscan facilita la trasparenza della Blockchain indicizzando e rendendo ricercabili tutte le transazioni sulla Blockchain di Ethereum nel modo più trasparente e accessibile possibile. Tramite etherscan sarà possibile tenere d'occhio tutte le transazioni eseguite dalla nostra applicazione e controllare il loro stato ed esito. È uno strumento fondamentale per ogni sviluppatore BC, inoltre offre le sue funzionalità anche per le testnet. [37]

# Remix

Remix è un editor, compilatore, tester e deployer online di Smart Contract. Si tratta di un progetto open source ed è gratuito.



The screenshot shows the Remix IDE interface with the Solidity file `3.Ballot.sol` open. The code defines a `Ballot` contract with a `Voter` struct and a `Proposal` struct. It includes mappings for voters and proposals, and an array for `proposals`. A constructor is present with parameters for `proposalNames` and `voteCounts`. The code is annotated with SPDX license information and JSDoc-style comments explaining the purpose of each part.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

/**
 * @title Ballot
 * @dev Implements voting process along with vote delegation
 */
contract Ballot {
    struct Voter {
        uint weight; // weight is accumulated by delegation
        bool voted; // if true, that person already voted
        address delegate; // person delegated to
        uint vote; // index of the voted proposal
    }
    struct Proposal {
        // If you can limit the length to a certain number of bytes,
        // always use one of bytes1 to bytes32 because they are much cheaper
        bytes32 name; // short name (up to 32 bytes)
        uint voteCount; // number of accumulated votes
    }
    address public chairperson;
    mapping(address => Voter) public voters;
    Proposal[] public proposals;
}

/**
 * @dev Create a new ballot to choose one of 'proposalNames'.
 * @param proposalNames names of proposals
 */

```

La barra di navigazione sul lato sinistro, contiene diverse icone, dall'alto verso il basso:

- Logo di Remix.
- File explorer: memorizza e consente di navigare attraverso i file degli smart contract, che terminano con l'estensione “.sol”.
- Compilatore: qui puoi trasformare i tuoi codici in Bytecode, leggibile dalla macchina virtuale di Ethereum (EVM).
- Deployer: consente di inserire il tuo smart contract su una blockchain (pubblica, di test, privata o simulata sul browser).
- Solidity Analysis: consente di verificare se lo smart contract contiene errori o comporta potenziali rischi.
- Debugger: strumento utilizzato per controllare l'esecuzione, passo dopo passo, di uno smart contract
- Sourcify: usato per dimostrare al pubblico che il lo smart contract sulla BC è lo stesso di quello che hai scritto.
- Unità di prova: consente di preparare dei test per I propri smart contract.
- Gestore plugins: per aggiungere nuove funzionalità o personalizzare uno smart contract utilizzando il software aggiuntivo di Remix.

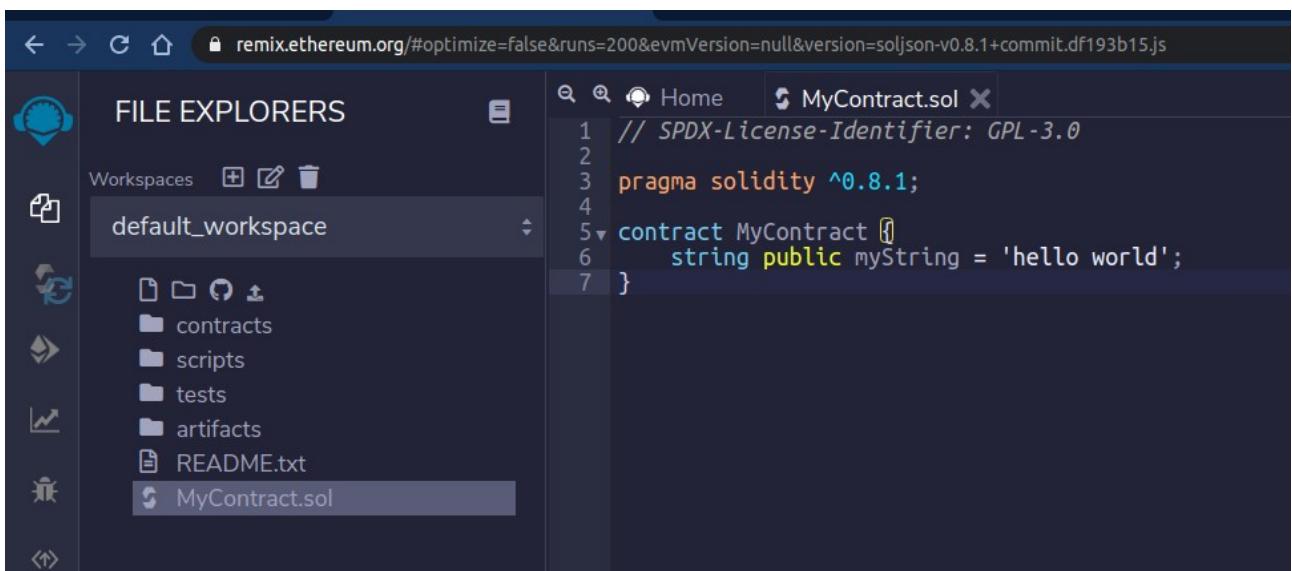
Verra' spiegato nel dettaglio man mano che si utilizzeranno le sue funzionalita' durante la descrizione del linguaggio solidity.

# Il linguaggio Solidity

Solidity è un linguaggio di alto livello orientato agli oggetti per l'implementazione di smart contract, e' influenzato da C ++, Python e JavaScript ed è progettato per la Ethereum Virtual Machine (EVM). Solidity è tipizzato staticamente, supporta l'ereditarietà, librerie e tipi complessi definiti dall'utente tra le altre funzionalità. Con Solidity puoi creare contratti per usi come votazioni, crowdfunding, aste alla cieca e portafogli multi-firma.

## Il primo smart contract

Creiamo un primo smart contract molto semplice ed analizziamo le basi della sintassi di Solidity, aggiungiamo un nuovo file *MyContract.sol* [Fig]



The screenshot shows the Remix IDE interface. On the left, there's a sidebar titled "FILE EXPLORERS" with icons for Workspaces, contracts, scripts, tests, artifacts, README.txt, and MyContract.sol. The "default\_workspace" is selected. In the main area, the title bar says "MyContract.sol". The code editor contains the following Solidity code:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract MyContract {
    string public myString = 'hello world';
}
```

La prima riga di codice serve per specificare sotto quale licenza sarà disponibile il file, e' facoltativa ed e' stata inserita solo per non ricevere un warning in fase di compilazione.

La riga 3 e' una direttiva al precompilatore che informa la versione di Solidity utilizzata.

Da riga 5 abbiamo il codice dello smart contract vero e proprio, indentificato dalla parola chiave *contract* e seguita dal nome. Successivamente fra parentesi graffe abbiamo la dichiarazione di una variabile di tipo *string* e con accessibilità *public* con un valore assegnato. Una delle particolarità di solidity e' che quando viene creato un attributo *public* viene automaticamente generato un getter per quella variabile, tramite il getter sarà possibile interagire con essa, in generale tutti i metodi all'interno di uno SC (smart contract) sono dei *punti di ingresso* tramite i quali e' possibile interagire con lo stesso. [36]

Una volta compilato lo SC può essere rilasciato su una BC tramite una apposita transazione, le transazioni sono il modo con cui dialoghiamo con la BC. Abbiamo a disposizione diverse opzioni su dove rilasciare il contratto:

- mainnet: usata solo in fase di effettivo rilascio di un prodotto
- testnet: usata per testare un prodotto in fase di sviluppo, simulando il suo funzionamento
- su una BC locale: nel caso in dovesimo fare test particolari e avere esigenze di personalizzazione lato blockchain.
- su una BC simulata all'interno del Browser tramite javascript: questa simulazione non e' molto accurata ma permette di testare con effetto immediato il proprio codice senza la necessita' di

attendere I tempi di mining delle transazione di rilascio e comunicazione con uno SC, questa' sara' l'opzione utilizzata da qui in avanti [Fig].

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract MyContract {
    string public myString = 'hello world';
}
```

Accedendo alla scheda apposita *Deploy & run transaction* e cliccando su deploy vediamo apparire il contratto che e' stato rilasciato e il suo relativo indirizzo. In etherium ogni SC ha un indirizzo esattamente come un account detenuto da una persona, tramite questo indirizzo e' appunto possibile inviare le transazioni a quel contratto, nella transazione possono venire specificate:

- l'interazione (metodo) col quale interagire
- Un quantitativo di ether da inviare
- Dei parametri da passare (campo data)

status	true Transaction mined and execution succeed
transaction hash	0x922fcf4e93d1ba51bb8d46cd5d4be413932102af9b24d839da4b63309e1faad1
contract address	0xd9145CCE2D3867254917e481e4449943F3193
from	0x5B30a6a701c568545dC7d803Fc8875f56bed0d4
to	MyContract.(constructor)
gas	3000000 gas
transaction cost	240397 gas
execution cost	133049 gas
hash	0x922fcf4e93d1ba51bb8d46cd5d4be413932102af9b24d839da4b63309e1faad1
input	0x600...10033
decoded input	[empty]

Remix ci mette a disposizione dei tasti con cui interagire direttamente con lo SM [Fig] e inoltre in basso ci da tutte le informazioni sulla transazione avvenuta. In caso di transazione non avvenuta con successo si ha un *revert* cioe' il ripristino allo stato precedente della BC a prima della transazione.

Se clicchiamo il tasto myString verrà fatta una richiesta (call) al getter della variabile myString. Questa chiamata (sempre una transazione) verrà registrata e tramite il suo contenuto potremmo capire il comportamento dello SC.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract MyContract {
    string public myString = 'hello world';
}
```

Quando uno SC viene creato esso è creato da una transazione, questa deve essere inviata da un account (indirizzo), questo si deve fare carico del costo (gas) della transazione. Come si può vedere nella successiva figura [Fig] il primo account non ha più 100 eth (assegnati di default ad ogni aggiornamento della pagina) ma un numero inferiore, diminuito dal costo della transazione che ha creato lo SC sulla BC.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract MyContract {
    uint public myUint;
    bool public myBool;
    address public myAddress;
    function setMyUint(uint _myUint) public {
        myUint = _myUint;
    }
    function setMyBool(bool _myBool) public {
        myBool = _myBool;
    }
    function setAddress(address _address) public {
        myAddress = _address;
    }
    function getBalanceOfAccount() public view returns(uint) {
        return myAddress.balance;
    }
    string public myString = 'hello world!';
    function setMyString(string memory _myString) public {
        myString = _myString;
    }
}
```

## Variabili

Nel seguente SC abbiamo aggiunto tutti i tipi elementari di variabili presenti in Solidity.

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with icons for deploying, running, and managing contracts. The main area shows a deployed contract named "MYCONTRACT AT 0xD8B...33FAB". The "Low level interactions" section on the left lists variables with their initial values:

- setAddress: address \_address (0: uint256: 0)
- setMyBool: bool \_myBool (0: bool: false)
- setMyString: string \_myString (0: string: hello world!)
- setMyUint: uint256 \_myUint (0: uint256: 0)

The right side displays the Solidity code for the contract:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract MyContract {
    uint public myUint;
    function setMyUint(uint _myUint) public {
        myUint = _myUint;
    }
    bool public myBool;
    function setMyBool(bool _myBool) public {
        myBool = _myBool;
    }
    address public myAddress;
    function setAddress(address _address) public {
        myAddress = _address;
    }
    function getBalanceOfAccount() public view returns(uint) {
        return myAddress.balance;
    }
    string public myString = 'hello world!';
    function setMyString(string memory _myString) public {
        myString = _myString;
    }
}
```

A teal box highlights the line "Inizializzazioni automatiche" (Automatic initializations) in the code.

Le variabili vengono inizializzate automaticamente a seconda del tipo. I tipi fondamentali sono:

- (u)int: descrive numeri interi con segno (Unsigned, senza segno). Il numero che segue e' il numero di bit utilizzati per descrivere il numero.
- bool: valori booleani
- address: descrive un indirizzo sulla BC, questo tipo contiene anche dei metodi e attributi come per esempio `address.balance` che restituisce i fondi (eth) dell'indirizzo.
- string: sono di fatto degli array, quindi non ci sono delle funzioni native per il confronto e la manipolazione delle stringhe, le stringhe ed in generale gli array sono costosi da memorizzare e lavorare a causa dei cicli necessari per la loro iterazione.

Ogni istruzione ha un relativo costo (gas), questo verrà approfondito in seguito. [36]

## SC esempio: depositare e prelevare eth

Consolidiamo quanto detto finora e aggiungiamo altri aspetti dello sviluppo degli SC con il seguente esempio[Fig].

```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity ^0.8.1;
4
5 contract DepositWithdraw {
6
7     uint public balanceReceived;
8     uint public lockedUntil;
9
10    function depositMoney() public payable {
11        balanceReceived += msg.value;
12        lockedUntil = block.timestamp + 1 minutes;
13    }
14
15    function getBalance() public view returns(uint) {
16        return address(this).balance;
17    }
18
19    function withdrawMoney() public {
20        if(lockedUntil < block.timestamp) {
21            address payable to = payable(msg.sender);
22            to.transfer(getBalance());
23        }
24    }
25
26    function withdrawMoneyTo(address payable _to) public {
27        if(lockedUntil < block.timestamp) {
28            _to.transfer(getBalance());
29        }
30    }
31 }
```

Questo SC ha la funzione di depositare fondi *depositMoney()*, di ritirarli *withdrawMoney()*, e di spostarli verso un altro indirizzo *withdrawMoneyTo()*. La parola chiave *payable* viene utilizzata per identificare un indirizzo che può ricevere fondi, senza questa specifica avremmo un errore in fase di compilazione; infatti il metodo *address.transfer()*, che si occupa del trasferimento di fondi dall'indirizzo attuale a quello che chiama il metodo, deve essere riferito ad un *payable*.

Nella variabile *balanceReceived* viene conservato il numero totale di eth depositati nello SC dal momento della sua creazione.

La variabile *lockedUntil* viene utilizzata per registrare un timestamp, ovvero una data e ora, che verrà utilizzato come controllo per accedere ai fondi, questi non potranno essere prelevati o spostati fino a quando quel timestamp non sarà passato. Nelle funzioni che permettono di prelevare e spostare i fondi infatti viene fatto un controllo con un *if* proprio fra la variabile sopracitata e *block.timestamp*, l'oggetto *block* è un oggetto globale che contiene informazioni riguardanti il blocco in cui è minata la transazione, una di queste informazioni è appunto il timestamp di quando il blocco viene minato. Nota: questo valore potrebbe non essere accurato, nella documentazione ethereum viene indicato un possibile errore di +/- 15sec.

Nota: in Solidity abbiamo a disposizione delle parole chiave che ci permettono di evitare di scrivere valori numerici, a volte anche molto lunghi, una di queste è ad esempio *minutes*.

Un altro oggetto globale e' *msg* questo contiene, invece, informazioni sul messaggio che viene inviato insieme alla transazione, esso contiene informazioni come i parametri da passare ad una funzione e in questo caso *value* che contiene il valore in eth che si e' voluto mandare col messaggio, questo valore viene inserito dall'account (indirizzo) che chiama la funzione *depositMoney()* che e' , da notare, contrassegnata come payable.

Ultima nota su questo SC, la parola chiave *view* nella signature della funzione *getBalance()*, sta ad indicare che questa funzione non richiede nessuna scrittura sulla BC ma e' solo una lettura, tale lettura viene fatta localmente sul proprio nodo ed e' una operazione virtualmente senza costi che non ha bisogno di mining.

Testiamo adesso le funzionalità del contratto per spiegare il passaggio di fondi fra account e SC .

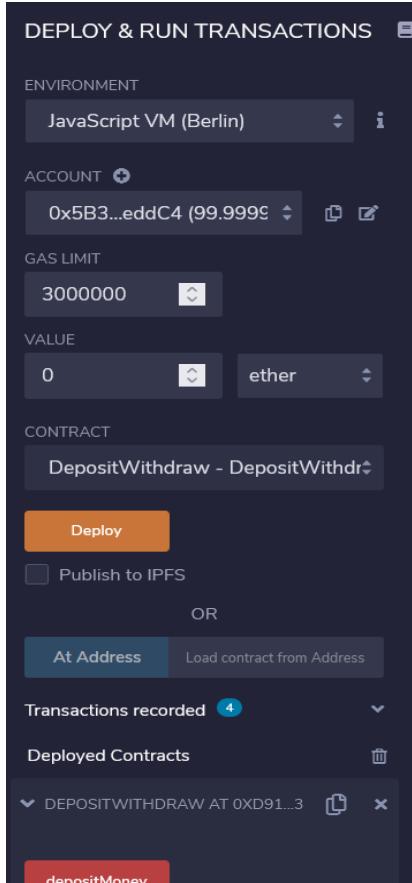


Fig.1

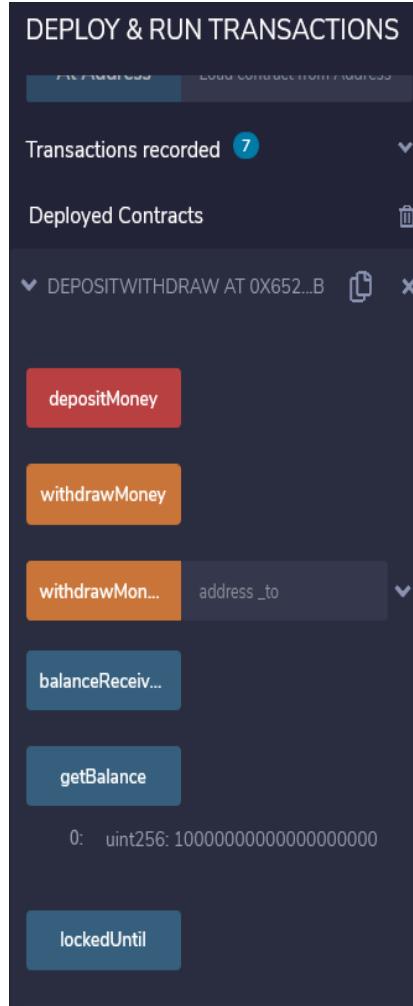


Fig.2

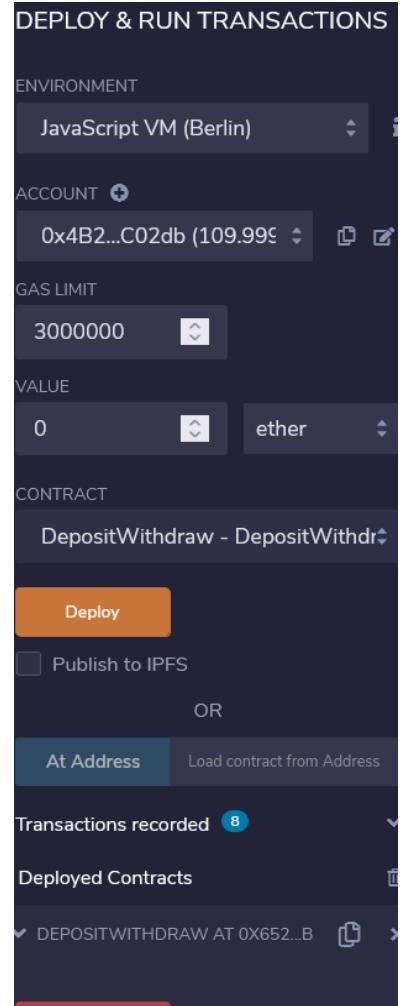


Fig.3

In figura 1 abbiamo già rilasciato il contratto tramite l'account 1 a cui e' stato dedotto il costo relativo a alla transazione del rilascio. In figura 2 abbiamo depositato 10 eth dall'account 2 sul contratto, a questo account viene addebitato il costo della transazione. Come si può vedere dalla figura 3 abbiamo trasferito i fondi presenti nel contratto all'account 3.

## Smart contract life cycle

Analizziamo adesso nel dettaglio le fasi di vita di uno SC.

### 1. START

Quando il nostro codice Solidity viene compilato otteniamo il suo *byte code*, equivalente ad un codice binario per il nostro sistema operativo, che puo' essere eseguito dalla EVM (Etherium Virtual Machine). Questo codice viene inviato nel campo *data* di una transazione, l'invio di una transazione per la creazione di uno SC ha schematicamente questa forma:

**sendTransaction({from:<senderAddress>, to: , data: byte code, ...})**

Viene chiamata la funzione di routine all'interno del nodo per inviare una transazione, il contenuto ha tre campi importanti:

- from: in cui viene indicato l'indirizzo del mittente, questo mittente deve essere un account, uno SC infatti nonostante abbia anche'esso un indirizzo non puo' creare altri SC.
- to: destinatario della transazione, nel caso di creazione di uno SC questo campo deve essere vuoto.
- data: qua va inserito il codice compilato dello SC

Dopo la compilazione e la transazione di creazione, se e' andata a buon fine, lo SC puo' ricevere chiamate (transazioni indirizzate a lui) ai suoi metodi.

### 2. RUNNING

Una volta creato lo SC esso riceve calls tramite transazioni, vediamo schematicamente come e' composta:

**sendTransaction(..., to:<contractAddress>, data:<metodo e parametri>, value:<eth passati>,...})**

- to: questa volta contiene l'indirizzo dello SC
- data: qua va indicato il nome del metodo(funzione) dello SC col quale si vuole interagire e I suoi eventuali parametri
- value: qui indichiamo I fondi da passare allo SC

Lo SC non puo' ritornare direttamente valori a chi lo ha chiamato, per questo vengono usati gli *eventi* che verranno trattati nel seguito.

### 3. STOP

Uno SC puo' venire fermato e "cancellato", virgolette d'obbligo perche' sappiamo che sulla BC nulla puo' essere cancellato in quanto I blocchi(i quali contengono le transazioni) una volta aggiunti alla catena non possono piu' essere modificati. Tramite la funzione *selfdestruct()* chiamata all'interno dello SC possiamo decidere di rimuoverlo dallo stato della BC, ovvero esiste ancora ma nessuno puo' piu' interagire con esso. [36]

## Mapping and Struct

Miglioriamo l'ultimo contratto visto in modo tale da permette a qualsiasi utente di depositare e prelevare e spostare fondi (come se fosse una banca presso cui si ha un conto) , nel farlo introduciamo due strutture dati tipiche di Solidity: *mapping* e *struct*. Lo SC e' nella figura[Fig] e verra' analizzato nel seguito.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;
contract Bank {
    struct Payment {
        uint amount;
        uint timestamp;
    }
    struct Balance {
        uint totalBalance;
        uint numPayments;
        mapping(uint => Payment) payments;
    }
    mapping(address => Balance) public balanceReceived;
    function getBalance() public view returns(uint) {
        return address(this).balance;
    }
    function sendMoney() public payable {
        balanceReceived[msg.sender].totalBalance += msg.value;
        Payment memory payment = Payment(msg.value, block.timestamp);
        balanceReceived[msg.sender].payments[balanceReceived[msg.sender].numPayments] = payment;
        balanceReceived[msg.sender].numPayments++;
    }
    function withdrawMoney(uint _amount) public {
        require(_amount <= balanceReceived[msg.sender].totalBalance, "not enough funds");
        balanceReceived[msg.sender].totalBalance -= _amount;
        assert(balanceReceived[msg.sender].totalBalance <= balanceReceived[msg.sender].totalBalance - _amount);
        payable(msg.sender).transfer(_amount);
    }
}
```

Le struct sono insiemi di dati erogenei, molto simili alle struct di C/C++, possono contenere al loro interno altre struct e qualsiasi altro tipo di dato. I valori all'interno di una struct sono inizializzati automaticamente seguendo le stesse regole delle variabili.

Le mapping (mappe) sono un tipo di dati interessante in Solidity. Vi si accede come array, ma hanno un grande vantaggio: tutte le coppie chiave/valore vengono inizializzate con il loro valore predefinito. Le mappe sono un insieme omogeneo di coppie chiave:valore.

Nel codice con mappe e struct e' possibile tenere memoria per ogni indirizzo che abbia interagito con lo SC del suo bilancio e dei pagamenti/prelievi eseguiti.

La parola chiave *Memory* viene utilizzata per memorizzare valori temporanei, questi valori sono cancellati quando termina la chiamata al metodo.

Nella funzione withdrawMoney() sono presenti un *require()* e un *assert()*, questa sintassi si riferisce alla gestione degli errori che verrà trattata nel paragrafo seguente. [36]

## Gestione degli errori

In solidity è possibile gestire gli errori con tre funzioni diverse: *require*, *assert*, *revert*.

La funzione require ha la seguente sintassi:

*require(<condition>, "message");*

Se la condizione risulta soddisfatta all'ora l'esecuzione dello SC procede normalmente, se invece non viene soddisfatta allora l'esecuzione viene bloccata e quanto fatto fino a quel momento viene

annullato e la BC viene ripristinata allo stato precedente, gli errori sono detti “state reverting” cioè fanno ritornare le cose allo stato in cui erano prima che l’errore si verificasse.

La funzione require puo’ essere chiamata tramite codice o automaticamente al verificarsi di certi problemi come per esempio: chiamata ad un contratto non esistente, fallimento della funzione .transfer(), la chiama ad una funzione non termina correttamente. Require viene utilizzata maggiormente per fare dei controlli su dei valori passati allo SC. Require restituisce I costi sostenuti per l’esecuzione fino a quel momento al chiamante.

La funzione assert ha esattamente la stessa sintassi di require ma non ritorna un messaggio, la differenza sostanziale e’ che essa non ritorna il costo sostenuto per l’esecuzione e che viene utilizzata per verificare lo stato interno del contratto, come il valore delle variabili. Assert puo’ essere chiamata da codice o viene automaticamente lanciata se si verificano errori di base come per esempio una divisione per zero.

La funzione revert semplicemente annulla tutti I cambiamenti e ritorna il costo al chiamante, viene usata raramente in quanto require e’ una sua versione piu’ evoluta.

## Fallback function

Le funzione di fallback di solidity vengono eseguite se nessuna delle altre funzioni corrisponde all’identificatore della funzione chiamata o se non sono stati forniti dati con la chiamata. A un contratto può essere assegnata solo una funzione di questo tipo, essa viene anche eseguita ogni volta che il contratto riceve Ether senza dati. Per ricevere Ether e aggiungerli al saldo totale del contratto, la funzione di fallback deve essere contrassegnata come payable. Se tale funzione non esiste, il contratto non può ricevere Ether tramite transazioni regolari e genererà un’eccezione. [38]

Proprietà di una funzione di fallback:

- Non ha nome o argomenti.
- Se non è contrassegnato come pagabile, il contratto genererà un’eccezione se riceve ether semplice senza dati.
- Non è possibile restituire nulla.
- Può essere definito una volta per contratto.
- Viene eseguita anche se il chiamante intendeva chiamare una funzione che non è disponibile
- È obbligatorio contrassegnarla con la parola chiave *external*.
- Ha un limite a 2300 gas quando viene richiamata da un’altra funzione.

Per dichiarare una funzione di fallback la sintassi e’ quella di una funzione anonima:

```
function () external { }
```

## Costruttori

Il costruttore è una funzione speciale dichiarata utilizzando la parola chiave *constructor()*. È una funzione opzionale e viene utilizzata per inizializzare le variabili di stato di un contratto. Di seguito sono riportate le caratteristiche chiave di un costruttore.

- Un contratto può avere un solo costruttore.
- Il codice del costruttore viene eseguito una volta quando viene creato un contratto e viene utilizzato per inizializzare lo stato del contratto.

Il costruttore puo' includere funzioni pubbliche e codice raggiungibile tramite funzioni pubbliche. Il codice del costruttore o qualsiasi metodo interno utilizzato solo dal costruttore non sono inclusi nel codice finale. Un costruttore può essere pubblico o interno (vedi modificatori di accesso nel seguito). Nel caso in cui non sia definito alcun costruttore, nel contratto è presente un costruttore predefinito. Di seguito il codice di un costruttore di esempio

```
contract Test {  
    constructor() public {}  
}
```

## View and Pure

Le funzioni di visualizzazione (view) assicurano che non modificheranno lo stato del contratto, ovvero non scriveranno su nessuna variabile. Se una funzione e' dichiarata come view le seguenti istruzioni, se presenti nella funzione, sono considerate modificanti lo stato e il compilatore lancerà un avviso in questi casi:

- Modificare le variabili di stato.
- Emissione di eventi.
- Creazione di altri contratti.
- Uso di selfdestruct().
- Invio di Ether tramite chiamate.
- Chiamare qualsiasi funzione che non sia contrassegnata come vista o pura.
- Utilizzo di chiamate di basso livello.

Il metodo getter è in modo predefinito una funzione di visualizzazione, esempio di seguito.

```
contract Test {  
    function getResult() public view returns(uint product, uint sum){  
        uint a = 1; // local variable  
        uint b = 2;  
        product = a * b;  
        sum = a + b;  
    }  
}
```

Nonostante sembri che avvengano delle scritture manipolando delle variabili, questa funzione e' effettivamente una funzione di vista in quanto non modifica nessuna variabile dello SC. [43][44]

## Ereditarietà

L'ereditarietà fra contratti e' consentita ed e' simile a quella di molti altri linguaggi di programmazione orientati ad oggetti (java, C++). L'ereditarietà in solidity è un modo per estendere la funzionalità di un contratto.[45] Solidity supporta sia l'ereditarietà singola che multipla. Di seguito sono riportati i punti salienti principali:

- Un contratto derivato può accedere a tutti i membri non privati, inclusi i metodi interni e le variabili di stato. Ma l'utilizzo di questo non è consentito.
- L'override della funzione è consentito a condizione che la signature della funzione rimanga la stessa. In caso di differenza dei parametri di output, la compilazione fallirà.

- Possiamo chiamare la funzione di un contratto “padre” usando la parola chiave *super* o usando il suo nome.

In caso di ereditarietà multipla, la chiamata di funzione tramite super dà la precedenza al contratto da cui si e’ maggiormente ereditato in termini di codice (funzioni, attributi). Di seguito un esempio.

```

1 pragma solidity ^0.8.1;
2
3 contract C {
4     uint private data;
5
6     uint public info;
7
8     //constructor
9     constructor(){
10         info = 10;
11     }
12     //private function
13     function increment(uint a) private pure returns(uint) { return a + 1; }
14
15     //public function
16     function updateData(uint a) public { data = a; }
17     function getData() public view virtual returns(uint) { return data; }
18     function compute(uint a, uint b) internal pure returns (uint) { return a + b; }
19 }
20
21 //Contratto derivato
22 contract E is C {
23     uint private result;
24     C private c;
25     constructor(){
26         c = new C();
27     }
28     function getComputedResult() public {
29         result = compute(3, 5);
30     }
31     function getResult() public view returns(uint) { return result; }
32     function getData() public view virtual override returns(uint) { return c.info(); }
33 }

```

Per ereditare si usa la parola chiave *is*, e’ necessaria inoltre la parola chiave *override* quando sovrascriviamo una funzione dal contratto padre. La parola chiave *virtual* va usata quando una funzione si vuole rendere sovrascrivibile da chi eredita quel contratto.

## Eventi

L’evento è un membro ereditabile di un contratto. Quando viene *emesso* un evento, memorizza gli argomenti passati nei log delle transazioni. Questi registri sono memorizzati su blockchain e sono accessibili utilizzando l’indirizzo del contratto fino a quando il contratto è presente sulla blockchain. **[46]** Un evento generato non è accessibile dall’interno dei contratti, nemmeno quello che li ha creati ed emessi.

Un evento può essere dichiarato utilizzando la parola chiave *event*, ed emesso tramite *emit*

```

event Deposit(address indexed _from, bytes32 indexed _id, uint _value);
emit Deposit(msg.sender, _id, msg.value);

```

Di seguito un esempio sugli eventi.

```

FILE EXPLORERS
Workspaces default_workspace
contracts
scripts
tests
artifacts
MyContract_metadata.json
MyContract.json
SendMoneyExample_metadata.json
SendMoneyExample.json
MappingsStructExample_metadata.json
MappingsStructExample.json
Bank_metadata.json
Bank.json
Bank_metadata.json
Bank.json
C_metadata.json

pragma solidity ^0.8.1;
contract EventExample {
    mapping(address => uint) public tokenBalance;
    event TokensSent(address _from, address _to, uint _amount);
    constructor() {
        tokenBalance[msg.sender] = 100;
    }
    function sendToken(address _to, uint _amount) public returns(bool) {
        require(tokenBalance[msg.sender] >= _amount, "Not enough tokens");
        assert(tokenBalance[_to] + _amount >= tokenBalance[_to]);
        assert(tokenBalance[msg.sender] - _amount <= tokenBalance[msg.sender]);
        tokenBalance[msg.sender] -= _amount;
        tokenBalance[_to] += _amount;
        emit TokensSent(msg.sender, _to, _amount);
        return true;
    }
}

```

E' un semplice contratto, di "beneficenza", che al momento della sua creazione prende un certo deposito da creatore e successivamente chiunque puo' spostare questi fondi.

A riga 6 abbiamo dichiarato un tipo di evento che prende in ingresso tre parametri, a riga 21 "emettiamo" quell'evento con le informazioni sulla transazione eseguita da sendToken().

Se dopo aver creato il contratto spostiamo I fondi e analizziamo la trasformazione troveremo nei file di log le informazioni "emesse" dall'evento insieme ad altre informazioni standard fra le quali ad esempio chi ha avviato la transazione.

gas	3000000 gas
transaction cost	34010 gas
execution cost	11138 gas
hash	0xb8519f7bac79cb40eaac7ce4043b957eb2d75f4ee0f28cbb47ab7e170dc16324
input	0xee4...00014
decoded input	{ "address _to": "0xAb8483F64d9C6d1EcF9b849Ae677d03315835cb2", "uint256 _amount": "20" }
decoded output	{ "0": "bool: true" }
logs	[ { "from": "0x7EF2e0048f5bae0e046f6BF797943daF4E8CB47", "topic": "0xe07861bafffd292b19188affe88c1a72bdbcb69d3015f18bb2cd0bf5349cc3e1", "event": "TokensSent", "args": { "0": "0x5B30Da6a701c568545dCfcB03FcB875f56beddC4", "1": "0xAb8483F64d9C6d1EcF9b849Ae677d03315835cb2", "2": "20", "_from": "0x5B30Da6a701c568545dCfcB03FcB875f56beddC4", "_to": "0xAb8483F64d9C6d1EcF9b849Ae677d03315835cb2", "_amount": "20" } } ]
value	0 wei

## Librerie

Le librerie sono simili ai contratti ma sono principalmente destinate al riutilizzo. Una libreria contiene funzioni che possono essere richiamate da altri contratti. Solidity ha alcune restrizioni sull'uso di una libreria. Di seguito sono riportate le caratteristiche chiave di una libreria in Solidity.

- Le funzioni di libreria possono essere chiamate direttamente se non modificano lo stato. Ciò significa che solo le funzioni pure o di visualizzazione possono essere chiamate dall'esterno della libreria.
- Una libreria non può avere variabili di stato.
- Una libreria non può ereditare alcun elemento.
- Una libreria non può essere ereditata.

Di seguito un esempio sull'utilizzo di una funzione di libreria.



The screenshot shows the Solidity IDE interface. On the left, the 'FILE EXPLORERS' panel displays a workspace named 'default\_workspace' containing contracts, scripts, tests, and artifacts. Artifacts include JSON files for MyContract, SendMoneyExample, MappingsStructExample, and Bank. On the right, the code editor shows two files: 'Events.sol' and 'Libraries.sol'. The 'Libraries.sol' file contains a library named 'Search' with a function 'indexOf' that returns the index of a value in an array. The 'Test' contract imports this library and uses its 'indexOf' function to find the index of a specific value ('4') in an array of uints.

```
pragma solidity ^0.8.1;

library Search {
    function indexOf(uint[] storage self, uint value) public view returns (uint) {
        for (uint i = 0; i < self.length; i++) if (self[i] == value) return i+1;
        return 0;
    }
}

contract Test {
    uint[] data;
    constructor() {
        data.push(1);
        data.push(2);
        data.push(3);
        data.push(4);
        data.push(5);
    }
    function isValuePresent() external view returns(uint){
        uint value = 4;
        // utilizzo della funzione di libreria
        uint index = Search.indexOf(data, value);
        return index;
    }
}
```

Di solito le librerie vanno messe in file separati e poi importate.

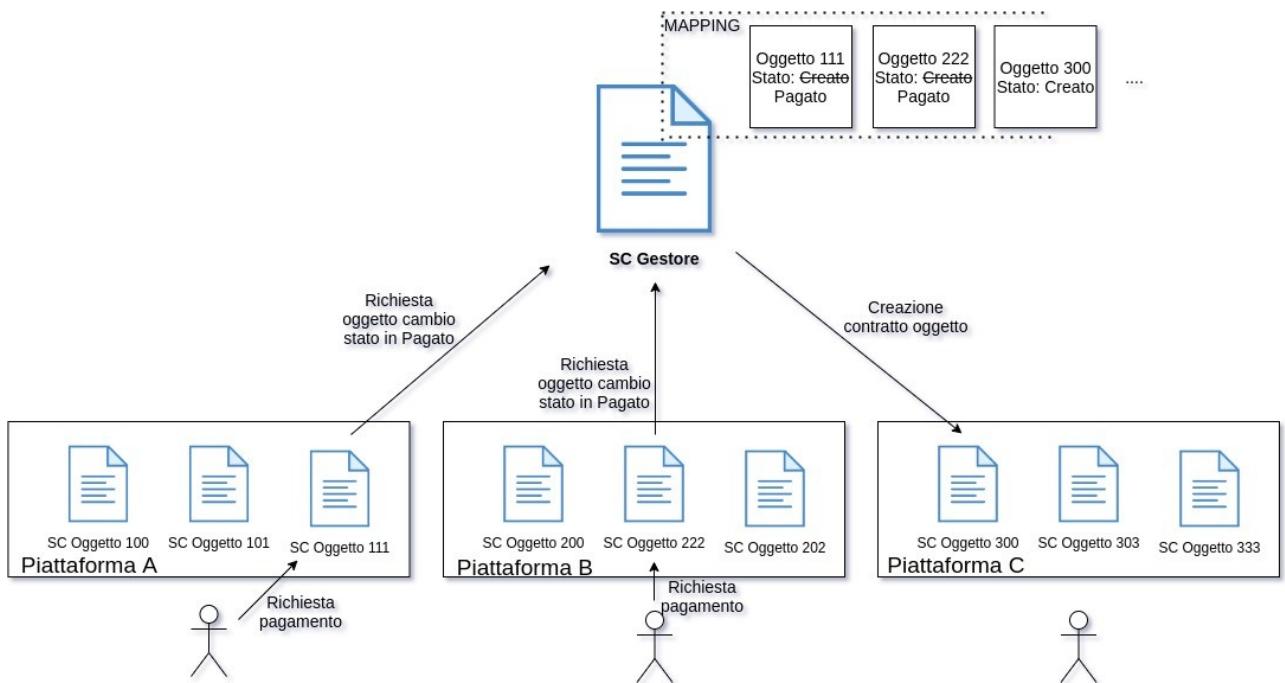
## Esempio utilizzo smart contracts: Automated supply chain

Con questo progetto si vuole dare una semplice dimostrazione su come la tecnologia degli smart contract su BC può essere utilizzata nella gestione di una supply chain (catena di approvvigionamento). I casi reali in cui potrebbe essere inserito sono l'automatizzazione dei sistemi di spedizione al pagamento di un oggetto e la raccolta dei pagamenti senza intermediari.

Nel seguito si darà dimostrazione:

- del sistema degli eventi di solidity
- l'utilizzo della funzione di basso livello address.call.value()
- integrazione di più smartcontract
- utilizzo dei modificatori

Lo schema di funzionamento è riassunto nella seguente figura:



Abbiamo un gestore che si occupa di mettere degli oggetti in vendita, il pagamento viene fatto sul singolo oggetto mentre le operazioni che riguardano la supply chain vengono svolte tutte dal gestore. Uno schema del genere potrebbe essere utile se per esempio avessimo un gestore che mette i suoi oggetti in vendita su molte piattaforme diverse, le piattaforme mettono a disposizione la possibilità di interagire con l'oggetto (pagamento) mentre il gestore si occupa della logistica in modo indipendente rispetto a dove sia avvenuto l'acquisto.

Pur nella sua semplicità questo semplice progetto ci offre già un'ottica di quello che è possibile fare per una supply chain reale utilizzando gli smart contracts di Ethereum.

Analizziamo adesso il codice degli smart contract e spieghiamo le parti più importanti.

## SC Gestore

Questo contratto ha lo scopo di definire il gestore della supply chain, di riconoscerlo e di dare, ai metodi che ne necessitano, la possibilita' di essere eseguiti solo dal gestore stesso.

Vediamo il codice e commentiamone le caratteristiche:

```
1 pragma solidity ^0.8.0;
2
3 ▼ abstract contract Gestore {
4     address public id_gestore;
5
6
7 ▼     constructor () {
8         id_gestore = msg.sender;
9     }
10
11 ▼     /**
12      * @dev lancia un errore se il metodo e' chiamato da chi non e' il proprietario
13      */
14 ▼     modifier soloGestore() {
15         require(isGestore(), "Non sei il gestore");
16         _;
17     }
18
19 ▼     /**
20      * @dev Ritorna vero se lo chiama il proprietario
21      */
22 ▼     function isGestore() public view returns (bool) {
23         return (msg.sender == id_gestore);
24     }
25 }]
```

Lo SC è abstract, non può essere istanziato direttamente, perchè di vogliono poi estendere le suoi funzionalità nel dettaglio nel contratto GestoreOggetti che erediterà da esso.

Il costruttore di occupa di memorizzare l'indirizzo del gestore che viene memorizzato nell'apposita variabile.

Successivamente abbiamo un modificatore, contraddistinto dalla parola chiave *modifier*. Viene utilizzato per aggiungere funzionalità ad un metodo, il codice dentro il modificatore viene eseguito nei metodi a cui viene aggiunto, in questo caso tutti I metodi a cui sarà aggiunto eseguiranno il require che controlla se si è il gestore. Il “\_” simboleggia il codice del metodo a cui verrà aggiunto il modificatore, mettendo questo carattere dopo il require significa che priva verrà eseguito quest'ultimo e poi il codice del metodo.

Il metodo *isGestore()* viene usato per fare il controllo se il chiamante è il gestore.

## SC Gestore oggetti

Questo contratto eredita da quello precedente tutte I metodi, le variabili, e il costruttore. Si occupa di fornire I metodi che saranno necessari alla gestione vera e propria deglio oggetti nella supply chain: creazione di un oggetto, pagamento, spedizione, consegna.

Codice:

```
1 pragma solidity ^0.8.0;
2
3 import "./Oggetto.sol";
4 import "./Gestore.sol";
5
6 contract GestoreOggetti is Gestore{
7
8     enum StatoOggetto{Creato, Pagato, InConsegna, Consegnato}
9
10    struct Oggetto_gestito {
11        Oggetto oggetto;
12        GestoreOggetti.StatoOggetto stato;
13    }
14
15    mapping (uint => Oggetto_gestito) public oggetti;
16    uint indice;
17
18    event AvanzamentoStato(uint _indiceOggetto, uint _stato, address _address);
19
20    function creaOggetto(string memory _name, uint _prezzoInWei) public soloGestore {
21        Oggetto o = new Oggetto(this, _prezzoInWei, indice, _name );
22        oggetti[indice].oggetto = o;
23        oggetti[indice].stato = StatoOggetto.Creato;
24        emit AvanzamentoStato(indice, uint(oggetti[indice].stato), address(o));
25        indice++;
26    }
27
28    function avviaPagamento(uint _indice) public payable {
29        Oggetto o = oggetti[_indice].oggetto;
30        require(address(o) == msg.sender, "Solo l'oggetto puo' aggiornare il suo stato");
31        require(oggetti[_indice].stato == StatoOggetto.Creato, "Oggetto non esistente nella supply chain");
32        oggetti[_indice].stato = StatoOggetto.Pagato;
33        emit AvanzamentoStato(indice, uint(oggetti[_indice].stato), address(o));
34    }
35
36    function avviaSpedizione(uint _indice) public payable soloGestore{
37        require(oggetti[_indice].stato == StatoOggetto.Pagato, "Oggetto non pagato");
38        oggetti[_indice].stato = StatoOggetto.InConsegna;
39        emit AvanzamentoStato(indice, uint(oggetti[_indice].stato), address(oggetti[_indice].oggetto));
40    }
41
42    function consegnaEffettuata(uint _indice) public payable soloGestore {
43        require(oggetti[_indice].stato == StatoOggetto.InConsegna, "Oggetto non spedito");
44        oggetti[_indice].stato = StatoOggetto.Consegnato;
45        emit AvanzamentoStato(indice, uint(oggetti[_indice].stato), address(oggetti[_indice].oggetto));
46    }
47 }
```

I due import sono necessari per poter accedere al contenuto dei file degli altri SC.

Abbiamo una enum che definisce I possibili stati di un particolare oggetto: Creato, Pagato, InConsegna, Consegnato.

Con una struct rappresentiamo l'oggetto e il suo stato, questa sarà utilizzata per fare una mapping degli oggetti nella supply chain identificati da un indice (numero intero).

L'evento *AvanzamentoStato()* verrà lanciato ogni volta che l'oggetto cambia il suo stato, per esempio da Pagato a InConsegna.

La funzione *creaoOggetto()* prende in ingresso come parametri il nome dell'oggetto e il suo prezzo il Wei (10^18 eth) e lo crea, lo aggiunge alla mapping degli oggetti gestiti con un identificati indice che si incrementa ogni volta. Viene settato lo stato del nuovo oggetto a “Creato” e viene emesso l'evento che notifica questo.

La funzione `avviaPagamento()` prende in ingresso l'indice del particolare oggetto ed è payable visto che deve ricevere fondi. Questa funzione può essere chiamata solo dall'oggetto stesso ovvero dallo smart contract associato all'oggetto, trova l'oggetto con quel particolare indice altrimenti viene lanciato un errore, aggiorna lo stato dell'oggetto ed emette l'evento che lo notifica. All'interno dello SC associato all'oggetto avremo un metodo apposito che permette il suo pagamento che automaticamente chiamerà quello nel gestore per aggiornare il suo stato: solo l'oggetto è in grado di aggiornare il suo stato da Creato a Pagato.

Le funzioni `avviaSpedizione()` e `consegnaEffettuate()` sono del tutto simili, sono chiamabili solo dal gestore e aggiornano il relativo stato dell'oggetto sono se lo stato precedente è compatibile, un oggetto non può passare da Creato a InConsegna ad esempio, l'oggetto deve passare per ogni suo stato in modo ordinato.

## SC Oggetto

Per ogni oggetto viene creato un SC di questo tipo, esso contiene tutte le caratteristiche dell'oggetto e un riferimento al gestore da cui è gestito. Un costruttore inizializza tutti i suoi valori, oltre questo abbiamo una funzione di fallback che viene chiamata automaticamente ogni volta che viene inviato denaro all'oggetto, viene segnata external perchè viene acceduta solo dall'esterno del contratto.

```

1 pragma solidity ^0.8.0;
2
3 import "./GestoreOggetti.sol";
4
5 contract Oggetto{
6
7     uint public prezzoInWei;
8     uint public pagatoInWei;
9     uint public id;
10    string public name;
11
12    GestoreOggetti gestore;
13
14    constructor(GestoreOggetti _gestore, uint _prezzoInWei, uint _id, string memory _name){
15        gestore = _gestore;
16        prezzoInWei = _prezzoInWei;
17        id = _id;
18        name = _name;
19    }
20
21    receive() external payable{
22        require(prezzoInWei == msg.value, "Prezzo non corretto");
23        require(pagatoInWei == 0, "Oggetto già pagato");
24        pagatoInWei += msg.value;
25        (bool successo, ) = address(gestore).call{value:msg.value}(abi.encodeWithSignature("avviaPagamento(uint _indice)",id));
26        require(successo, "Pagamento non avvenuto con successo");
27    }
}

```

La variabile `pagatoInWei` memorizza la somma pagata per l'oggetto, questa viene utilizzata per controllare al momento del pagamento se l'oggetto è già stato pagato (acquistato). Oltre questo controllo nella funzione di fallback `receive()` controlliamo anche se il prezzo pagato è corretto, in tal caso aggiorniamo la variabile `pagatoInWei` ed effettuiamo la chiamata alla funzione apposita del gestore per aggiornare lo stato dell'oggetto. Questa chiamata viene effettuata con una funzione di basso livello, `call` che ha la seguente sintassi:

`.call{value:<fondi da passare>}{abi.encodeWithSignature("signaturefunzione",<parametri>)}`

Nel nostro caso passiamo I fondi che il pagante ha inviato a questo contratto, questo a sua volta li passa al contratto gestore che esegue la funzione per aggiornare lo stato. La signature è il nome della funzione seguito dal tipo e dal nome dei parametri passati. Nel nostro caso inseriamo quella della funzione che vogliamo chiamare seguita dal valore del parametro.

## Supply chain miele

Nella seguente applicazione, basata sulla BC di ethereum viene implementata la gestione della supply chain di un consorzio di produttori e distributori di miele, organizzato in lotti; chi vi aderisce fornisce i propri dati (fra cui per esempio la locazione geografica dell'apicoltore che produce il miele) e viene registrato come una delle seguenti figure:

- *apicoltore*: potrà registrare i lotti di miele prodotti, con le relative informazioni, metterli in vendita oppure acquistarne.

- *distributore*: potrà acquistare lotti, mettere in vendita e nel caso delle messa in vendita verso il consumatore finale registrare la fuoriuscita del lotto dalla supply chain.

Tutte queste operazioni sono svolte all'interno del consorzio, con lo scopo di una maggiore trasparenza e fiducia nel consorzio, automatizzazione dei processi gestionali, e una informazione affidabile per il consumatore che può consultare le informazioni riguardanti il singolo lotto: provenienza geografica, apicoltore che ha prodotto il lotto, data (time stamp transazioni ethereum) e storia di tutti i passaggi di mano del lotto.

### Panoramica contratti e registrazione degli attori

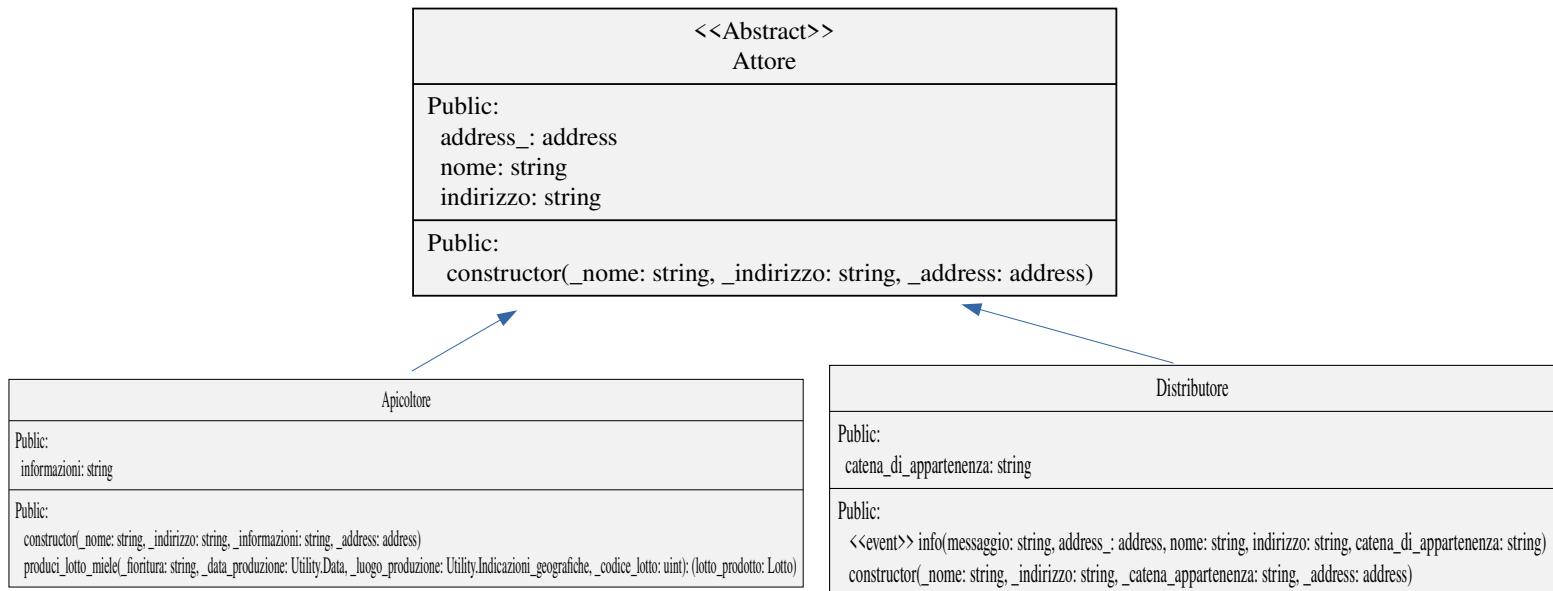
Per attore viene inteso un generico partecipante alla supply chain, che ha uno dei ruoli sopra citati e potenzialmente anche altri che possono essere creati. Questa figura generica viene descritta con uno smart contract astratto, ovvero non instanziabile, *Attore.sol* che contiene le caratteristiche comuni che devono avere tutti i partecipanti alla supply chain.

```
1 // SPDX-License-Identifier: UNLICENSED
2 pragma solidity >=0.7.0 <0.9.0;
3
4 abstract contract Attore {
5
6     address public address_;
7     string public nome;
8     string public indirizzo;
9
10    constructor(string memory _nome, string memory _indirizzo, address _address) {
11
12        address_ = _address;
13        nome = _nome;
14        indirizzo = _indirizzo;
15    }
16
17 }
```

Le caratteristiche comuni ad ogni attore sono:

- un address *address\_* che contiene l'indirizzo ethereum di chi verrà registrato come attore
- una stringa per il *nome*
- una stringa per l'*indirizzo*

Viene definito l'apposito costruttore che verrà invocato nei costruttori dei contratti figli *distributore.sol* e *apicoltore.sol*.



*Distributore.sol* eredita da *Attore* e in più contiene l’informazione sulla catena di distribuzione di appartenenza. Nel suo costruttore viene richiamato il costruttore di *Attore*, questo permette di assegnare i parametri dati in ingresso al costruttore di *Distributore* passandoli al costruttore di *Attore*.

```

1 // SPDX-License-Identifier: UNLICENSED
2 pragma solidity >=0.7.0 <0.9.0;
3
4 import "./Attore.sol";
5
6 contract Distributore is Attore {
7
8     string public catena_di_appartenenza;
9
10    constructor(string memory _nome, string memory _indirizzo, string memory _catena_appartenenza, address _address)
11    Attore(_nome, _indirizzo, _address) {
12        catena_di_appartenenza = _catena_appartenenza;
13    }
14
15 }

```

*Apicoltore.sol* eredita da *Attore*, ha una stringa extra per eventuali ulteriori informazioni, il suo costruttore, ed il metodo *producি\_lotto\_miele* che verrà illustrato nel seguito.

```

1 // SPDX-License-Identifier: UNLICENSED
2 pragma solidity >=0.7.0 <0.9.0;
3
4 import "./Lotto.sol";
5 import "./Attore.sol";
6 import "./Utility.sol";
7
8 contract Apicoltore is Attore{
9
10     string public informazioni;
11     //si possono aggiungere tutte le informazioni necessarie
12
13     |
14     constructor(string memory _nome, string memory _indirizzo, string memory _informazioni, address _address)
15     Attore(_nome, _indirizzo, _address) {
16         informazioni = _informazioni;
17     }
18
19
20     function produci_lotto_miele(
21         string memory _fioritura,
22         Utility.Data memory _data_produzione,
23         Utility.Indicazioni_geografiche memory _luogo_produzione,
24         uint _codice_lotto
25     )
26     public
27     returns (Lotto lotto_prodotto)
28     {
29         return Lotto(new Lotto(this, _fioritura, _data_produzione, _luogo_produzione, _codice_lotto));
30     }
31
32
33 }

```

Nel contratto *Supply\_chain\_miele* abbiamo le strutture dati per tenere traccia dei lotti (mapping *lotti*) degli apicoltori (mapping *apicoltori*) e dei distributori (mapping *distributori*) e diverse funzioni, illustrate tutte nel corso della trattazione, fra cui quelle che permettono di registrare un nuovo attore (apicoltore o distributore) all'interno del consorzio.

```

1  // SPDX-License-Identifier: UNLICENSED
2  pragma solidity >=0.7.0 <0.9.0;
3
4  import "./Lotto.sol";
5  import "./Apicoltore.sol";
6  import "./Distributore.sol";
7  import "./Utility.sol";
8
9  ▼ contract Supply_chain_miele{
10      uint progressivo_lotti = 0;
11      mapping (uint => Lotto) lotti;
12      mapping (address => Apicoltore) apicoltori;
13      mapping (address => Distributore) distributori;
14
15      function aggiungi_apicoltore(string memory _nome, string memory _indirizzo, string memory _informazioni) public {
16          Apicoltore apicoltore = new Apicoltore(_nome, _indirizzo, _informazioni, msg.sender);
17          require(!apicoltori[msg.sender] == apicoltore), "Indirizzo già registrato come apicoltore";
18          apicoltori[msg.sender] = apicoltore;
19      }
20
21      function aggiungi_distributore(string memory _nome, string memory _indirizzo, string memory _catena_appartenenza) public {
22          Distributore distributore = new Distributore(_nome, _indirizzo, _catena_appartenenza, msg.sender);
23          require(!distributori[msg.sender] == distributore), "Indirizzo già registrato come distributore";
24          distributori[msg.sender] = distributore;
25      }
26
27  }
```

*aggiungi\_apicoltore* e *aggiungi\_distributore* hanno la stessa struttura, prendono in ingresso i dati dell'attore da aggiungere, creano un nuovo oggetto di quel tipo e controllano tramite un *require* che quell'attore non sia già registrato, se il controllo viene superato allora il nuovo attore viene memorizzato nell'apposita mapping.

## Creazione del lotto: ingresso nella supply chain

In *Supply\_chain\_miele* abbiamo anche la funzionalità che permette di registrare un nuovo lotto:

```

32      function registra_lotto_miele_prodotto(
33          string memory _fioritura,
34          Utility.Data memory _data_produzione,
35          Utility.Indicazioni_geografiche memory _luogo_produzione
36      ) public
37  {
38      Apicoltore a = apicoltori[msg.sender];
39      require(address(a) != address(0x0), 'Indirizzo non registrato come apicoltore');
40      Lotto nuovo_lotto = a.produci_lotto_miele(_fioritura, _data_produzione, _luogo_produzione, progressivo_lotti);
41      lotti[progressivo_lotti] = nuovo_lotto;
42      progressivo_lotti++;
43  }
```

Questa funzione:

- prende in ingresso le informazioni sul lotto creato: tipo fioritura, data produzione, luogo produzione
- Acquisisce nella variabile *a* di tipo *Apicoltore* l'apicoltore che ha l'indirizzo uguale a chi sta invocando la funzione
- Se l'indirizzo dell'apicoltore trovato è diverso dall'indirizzo nullo, ovvero il valore di default con cui vengono inizializzati i contenuti di tipo indirizzo delle mapping, allora vuol dire che l'indirizzo di chi sta invocando la funzione è registrato come apicoltore e gli è quindi permesso creare un nuovo lotto.
- Nella variabile *nuovo\_lotto* di tipo *Lotto* verrà contenuto l'oggetto lotto creato dal metodo *producil\_lotto\_miele* di quell'apicoltore

- Se le funzioni chiamate sugli oggetti vanno a buon fine allora il lotto viene aggiunto nella relativa mapping e il numero progressivo del lotto (identificatore univoco dei lotti) viene incrementato.

Il metodo *produci\_lotto\_miele* dell’apicoltore chiama a sua volta il costruttore del contratto *Lotto* che crea un nuovo oggetto lotto passando oltre ai parametri ricevuti da *registra\_lotto\_miele* l’apicoltore produttore con la parola chiave *this*.

Il contratto *Lotto.sol* contiene tutti gli attributi e le strutture necessarie per descriverne le informazioni, lo stato e la storia dei passaggi di proprietà nella mapping *storia\_lotto*.

```
1 // SPDX-License-Identifier: UNLICENSED
2 pragma solidity >=0.7.0 <0.9.0;
3
4 import "./Apicoltore.sol";
5 import "./Utility.sol";
6
7 contract Lotto {
8
9     enum Stato_lotto{
10
11         Prodotto_da_apicoltore,      // 0
12         In_vendita,                // 1
13         Comprato,                  // 2
14         fuori_supply_chain         // 3
15     }
16
17     struct Trasferimento {
18         address da;
19         address a;
20         Utility.Data data;
21     }
22
23     uint public codice_lotto;          // codice lotto
24     address public address_proprietario_corrente; // Address del proprietario corrente
25     Apicoltore public apicoltore_originario; // apicoltore che lo ha prodotto
26     string public fioritura;          // tipo fioritura
27     Utility.Data public data_produzione; // data produzione
28     uint public prezzo;              // prezzo
29     Stato_lotto public stato_lotto;   // Stato corrente del lotto
30     Utility.Indicazioni_geografiche public luogo_produzione; // Indicazioni_geografiche
31
32     mapping (uint => Trasferimento) storia_lotto;
33
34     //definizione eventi
35     event Comprato(string messaggio, uint codice_lotto);
36     event In_vendita_per_consumatore_finale(string messaggio, uint codice_lotto);
37     event Prodotto_da_apicoltore(string messaggio, uint codice_lotto);
38     event In_vendita(string messaggio, uint codice_lotto);
```

Per ogni cambiamento di stato del lotto è definito un apposito evento da lanciare.

Il costruttore è il metodo che viene chiamato automaticamente quando viene creato un oggetto di una certa classe (contratto), per il lotto il costruttore prende in ingresso tutti i parametri su detti e l’apicoltore che lo produce: quello che viene passato da metodo di apicoltore con la parola chiave *this*. Dopo aver assegnato tutti i parametri passati al costruttore ai relativi attributi viene inizializzato il prezzo a zero, lo stato iniziale a “prodotto da apicoltore”, e aggiunto alla storia del lotto la data di produzione con un primo trasferimento vuoto (da indirizzo nullo a indirizzo nullo, da nessuno a nessuno). In fine viene emesso l’evento che regista nel log della transazione l’avvenuta produzione del lotto.

```

40     constructor(
41         Apicoltore _apicoltore_originario,
42         string memory _fioritura,
43         Utility.Data memory _data_produzione,
44         Utility.Indicazioni_geografiche memory _luogo_produzione,
45         uint _codice_lotto
46     )
47     {
48
49         address proprietario_corrente = _apicoltore_originario.address_();
50         codice_lotto = _codice_lotto;
51         apicoltore_originario = _apicoltore_originario;
52         fioritura = _fioritura;
53         data_produzione = _data_produzione;
54         prezzo = 0;
55         luogo_produzione = _luogo_produzione;
56         stato_lotto = Stato_lotto.Prodotto_da_apicoltore;
57
58         Trasferimento memory t;
59         t.da = address(0x0);
60         t.a = address(0x0);
61         t.data = _data_produzione;
62         storia_lotto[0] = t;
63
64         // Emisso evento relativo al cambio di stato
65         emit Prodotto_da_apicoltore("Prodotto lotto ",codice_lotto);
66     }

```

## Vendita e acquisto

Il contratto *Supply\_chain\_miele.sol* mette a disposizione la funzione *registra\_messa\_in\_vendita*, che prende in ingresso il codice del lotto che si vuole mettere in vendita e il prezzo.

```

46     function registra_messa_in_vendita(uint _codice_lotto, uint _prezzo) public {
47         Attore a = apicoltori[msg.sender];
48         if (address(a) == address(0x0)){
49             a = distributori[msg.sender];
50         }
51         Lotto l = lotti[_codice_lotto];
52         require((address(l) != address(0x0)), "Lotto non esistente");
53         require((address(a) != address(0x0)), "Non sei registrato come apicoltore o distributore");
54         require(_prezzo >0, "Prezzo non valido");
55         lotti[_codice_lotto].metti_in_vendita(_prezzo, a.address_());
56     }

```

Per prima cosa viene reperito, se esiste, l'attore: se l'indirizzo del chiamante non è registrato la transazione viene annullata. Poi viene preso il lotto, se esiste, con il codice passato alla funzione e si verifica se il prezzo di vendita è un prezzo valido (maggiore di zero). Se questi controllo vengono passati allora si richiama il metodo *metti\_in\_vendita* di quel lotto passandogli il prezzo e l'indirizzo del chiamante.

```

68     function metti_in_vendita(uint _prezzo, address address_proprietario) public{
69         require((stato_lotto != Stato_lotto.In_vendita) && (stato_lotto != Stato_lotto.fuori_supply_chain) , "Lotto gia' in vendita");
70         require(address_proprietario == address_proprietario_corrente, "Non sei il proprietario del lotto");
71         stato_lotto = Stato_lotto.In_vendita;
72         prezzo = _prezzo;
73         emit In_vendita("Lotto messo in vendita", codice_lotto);
74     }

```

In questo metodo viene controllato se il lotto è in uno stato che permette la messa in vendita (se non è già in vendita oppure fuori dalla supply chain) e se l'indirizzo del chiamante corrisponde all'indirizzo del proprietario attuale del lotto. Se i controlli vengono passati viene aggiornato lo stato del lotto e il prezzo ed emesso il relativo evento.

Per l'acquisto si procede in maniera simile alla messa in vendita.

Dal contratto *Supply\_chain\_miele.sol* la funzione *registra\_acquisto* svolge sempre gli stessi controlli sull'attore e sul lotto, inoltre viene fatto un controllo per verificare che il chiamante non sia il proprietario stesso del lotto, infine viene controllato se valore mandato con la transazione è sufficiente a coprire il costo di quel lotto. Nella variabile *vecchio\_proprietario* viene memorizzato l'indirizzo di ritorno del metodo *imposta\_acquistato* del lotto che restituisce proprio l'indirizzo del vecchio proprietario, questo verrà usato per trasferire il prezzo pagato al vecchio proprietario, inoltre al nuovo proprietario corrente, viene trasferito un eventuale resto.

```

58▼   function registra_acquisto(uint _codice_lotto) public payable {
59     Attore a = apicoltori[msg.sender];
60▼   if (address(a) == address(0x0)){
61     a = distributori[msg.sender];
62   }
63   Lotto l = lotti[_codice_lotto];
64   require((address(l) != address(0x0)), "Lotto non esistente");
65   require((address(a) != address(0x0)), "Non sei registrato come apicoltore o distributore");
66   require(msg.sender != l.address_proprietario_corrente(), "Non puoi comprare tu stesso");
67   require(msg.value >= l.prezzo(), "Fondi non sufficienti");
68   uint resto = msg.value - l.prezzo();
69   address payable vecchio_proprietario = payable(l.imposta_acquistato(msg.sender));
70▼   if (resto > 0){
71     (payable(l.address_proprietario_corrente())).transfer(resto);
72   }
73   vecchio_proprietario.transfer(msg.value - resto);
74 }
```

Il metodo *imposta\_acquistato* in *Lotto* controlla se il lotto è acquistabile (sia in vendita) e in tal caso aggiorna lo stato del lotto, reimposta il prezzo a zero, aggiorna il proprietario, emette il relativo evento e restituisce l'indirizzo del vecchio proprietario che verrà usato come su detto.

```

76▼   function imposta_acquistato(address _address_nuovo_proprietario) public returns (address vecchio_proprietario){
77     require(stato_lotto == Stato_lotto.In_vendita, "Lotto non in vendita");
78     stato_lotto = Stato_lotto.Comprato;
79     prezzo = 0;
80     vecchio_proprietario = address_proprietario_corrente;
81     address_proprietario_corrente = _address_nuovo_proprietario;
82     emit Comprato("Lotto comprato",codice_lotto);
83     return vecchio_proprietario;
84 }
```

## Uscita dalla supply chain

Quando un lotto viene acquistato da un distributore e questo lo mette in vendita per il consumatore finale questo “esce” dalla supply chain, ovvero non è più necessario tenerne traccia. Per questa funzionalità abbiamo *registra\_messa\_in\_vendita\_finale* chiamabile da *Supply\_chain\_miele*

```
76  function registra_messa_in_vendita_finale(uint _codice_lotto) public{
77      Distributore d = distributori[msg.sender];
78      Lotto l = lotti[_codice_lotto];
79      require((address(l) != address(0x0)), "Lotto non esistente");
80      require((address(d) != address(0x0)), "Non sei registrato come distributore");
81      lotti[_codice_lotto].messa_in_vendita_finale(d.address_());
82  }
```

Questa funzione verifica che il chiamante sia un distributore registrato (solo i distributori possono impostare la messa in vendita finale verso il consumatore) e se il lotto con il codice passato esiste. Se i controlli vengono passati allora viene chiamato il metodo *messa\_in\_vendita\_finale* del lotto.

Quest’ultimo controlla se il distributore è il proprietario del lotto e se questo non è già fuori dalla supply chain, se i controllo vengono passati il prezzo viene resettato a zero e lo stato del lotto aggiornato, infine viene emesso il relativo evento.

```
86  function messa_in_vendita_finale(address address_proprietario) public{
87      require(address_proprietario == address_proprietario_corrente, "Non sei il proprietario del lotto");
88      require(stato_lotto != Stato_lotto.fuori_supply_chain, "Lotto già' fuori supply chain");
89      Stato_lotto = Stato_lotto.fuori_supply_chain;
90      prezzo = 0;
91      emit In_vendita_per_consumatore_finale("Lotto in vendita per consumatore finare (fuori supply chain)",codice_lotto);
92  }
93 }
```

## Ottenerne informazioni dalla supply chain

A scopo di testing e di utilizzo sono stati implementati dei metodi che permettono di interrogare la supply chain per ottenere informazioni relative ad un attore o ad un particolare lotto.

Il metodo *ottieni\_apicoltore* e *ottieni\_distributore* prendono in ingresso un address e ritornano le informazioni relative a quell’attore se esiste sulla supply chain.

Il metodo *ottieni\_lotto* restituisce le informazioni relative al lotto, se esiste, con il codice passato.

```
84 // FUNZIONI PER OTTENERE I DATI
85
86     function ottieni_apicoltore(address address_) public view returns(
87         string memory nome,
88         string memory indirizzo,
89         string memory informazioni
90     )|
91 {
92     Apicoltore a = apicoltori[address_];
93     require(address(a) != address(0x0),"Apicoltore inesistente");
94     return (
95         a.nome(),
96         a.indirizzo(),
97         a.informazioni()
98     );
99 }
```

```

101     function ottieni_distributore(address address_) public view returns(
102         string memory nome,
103         string memory indirizzo,
104         string memory catena_appartenenza
105     )
106    {
107        Distributore d = distributori[address_];
108        require(address(d) != address(0x0) , "Distributore inesistente");
109        return (
110            d.nome(),
111            d.indirizzo(),
112            d.catena_di_appartenenza()
113        );
114    }

```

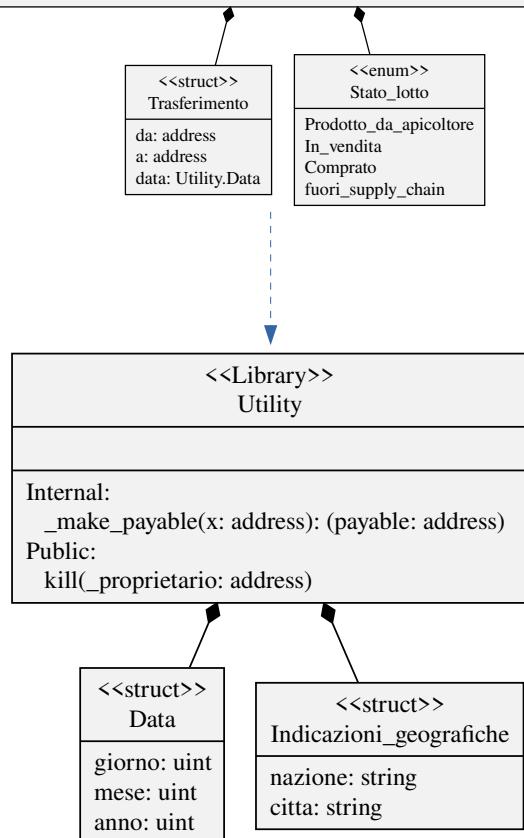
```

116     function ottieni_lotto(uint _id_lotto) public view returns(
117         uint codice_lotto,
118         address address_proprietario_corrente,
119         address address_apicoltore_originario,
120         string memory fioritura,
121         uint prezzo,
122         Lotto.Stato_lotto stato_lotto,
123         uint giorno_produzione,
124         uint mese_produzione,
125         uint anno_produzione,
126         string memory citta_di_produzione
127     )
128    {
129        Lotto l = lotti[_id_lotto];
130        require(address(l) != address(0x0) , "Lotto inesistente");
131        (uint giorno, uint mese, uint anno) = l.data_produzione();
132        (, string memory cit) = l.luogo_produzione();
133        return(
134            l.codice_lotto(),
135            l.address_proprietario_corrente(),
136            l.address_proprietario_corrente(),
137            l.fioritura(),
138            l.prezzo(),
139            l.stato_lotto(),
140            giorno,
141            mese,
142            anno,
143            cit
144        );
145    }
146

```

## Diagrammi contratti

Lotto
<pre>Public: codice_lotto: uint address_proprietario_corrente: address apicoltore_originario: Apicoltore fioritura: string data_produzione: Utility.Data prezzo: uint stato_lotto: Stato_lotto luogo_produzione: Utility.Indicazioni_geografiche stato: Stato_lotto storia_lotto: mapping(uint=&gt;Trasferimento)</pre>
<pre>Public: &lt;&lt;event&gt;&gt; Comprato(messaggio: string, codice_lotto: uint) &lt;&lt;event&gt;&gt; In_vendita_per_consumatore_finale(messaggio: string, codice_lotto: uint) &lt;&lt;event&gt;&gt; Prodotto_da_apicoltore(messaggio: string, codice_lotto: uint) &lt;&lt;event&gt;&gt; In_vendita(messaggio: string, codice_lotto: uint) constructor(_apicoltore_originario: Apicoltore, _fioritura: string, _data_produzione: Utility.Data, _luogo_produzione: Utility.Indicazioni_geografiche, _codice_lotto: uint) metti_in_vendita(_prezzo: uint, address_proprietario: address) imposta_acquistato(_address_nuovo_proprietario: address): (vecchio_proprietario: address) messi_in_vendita_finale(address_proprietario: address)</pre>



Supply_chain_miele
<pre>Public: progressivo_lotti: uint lotti: mapping(uint=&gt;Lotto) apicoltori: mapping(address=&gt;Apicoltore) distributori: mapping(address=&gt;Distributore)</pre>
<pre>Public: &lt;&lt;payable&gt;&gt; registra_acquisto(_codice_lotto: uint) aggiungi_apicoltore(_nome: string, _indirizzo: string, _informazioni: string) aggiungi_distributore(_nome: string, _indirizzo: string, _catena_appartenenza: string) registra_lotto_miele_prodotto(_fioritura: string, _data_produzione: Utility.Data, _luogo_produzione: Utility.Indicazioni_geografiche) registra_messa_in_vendita(_codice_lotto: uint, _prezzo: uint) registra_messa_in_vendita_finale(_codice_lotto: uint) ottieni_apicoltore(address: address): (nome: string, indirizzo: string, informazioni: string) ottieni_distributore(address: address): (nome: string, indirizzo: string, catena_appartenenza: string) ottieni_lotto(_id_lotto: uint): (codice_lotto: uint, address_proprietario_corrente: address, address_apicoltore_originario: address, fioritura: string, prezzo: uint, stato_lotto: Lotto.Stato_lotto, giorno_produzione: uint, mese_produzione: uint, anno_produzione: uint, citta_di_produzione: string)</pre>

## Bibliografia

1. <https://docs.microsoft.com/it-it/learn/modules/intro-to-blockchain/2-what-is-blockchain> [Documentazione introduttiva alla Blockchain di Microsoft]
2. [https://it.wikipedia.org/wiki/Single\\_point\\_of\\_failure](https://it.wikipedia.org/wiki/Single_point_of_failure) [Definizione su Wikipedia di “Single point of failure”]
3. <https://en.wikipedia.org/wiki/Bitcoin#Blockchain> [Definizione Bitcoin Wikipedia]
4. <https://it.wikipedia.org/wiki/Blockchain#Decentramento> [Definizione decentramento Wikipedia]
5. <https://www.youtube.com/watch?v=EH6vE97qIP4&list=PLUl4u3cNGP63UUkfL0onkxF6MYgVa04Fn> [Lezione 1 del corso “Introduction fro Blockchain and money” tenuto al MIT del 2018 liberamente accessibile attraverso la piattaforma MIT OpenCourseWare]
6. <https://emergetech.org/cryptography-in-blockchain/> [Articolo sull’uso della crittografia nella Blockchain della fondazione “The Emerging Tech Foundation”]
7. <https://atomicwallet.io/what-is-decentralization> [Articolo sulla decentralizzazione]
8. [https://www.youtube.com/watch?v=87WWHgbtjBA&ab\\_channel=EthereumFoundation](https://www.youtube.com/watch?v=87WWHgbtjBA&ab_channel=EthereumFoundation) [Intervento al DevCon3 (evento più importante nel panorama Ethereum) di Giuseppe Bertone, IT solution architect in Almaviva, sull’iniziativa del ministero dell’agricoltura italiano per il tracciamento della supply chain dei vini]
9. [https://www.youtube.com/watch?v=tihpYQCWnCw&ab\\_channel=IBMBLOCKCHAIN](https://www.youtube.com/watch?v=tihpYQCWnCw&ab_channel=IBMBLOCKCHAIN) [Video introduttivo alla soluzione per supply chain internazionali di IBM sul loro canale ufficiale Youtube]
10. [https://www.youtube.com/watch?v=Z6HS1bSzzHk&ab\\_channel=SirajRaval](https://www.youtube.com/watch?v=Z6HS1bSzzHk&ab_channel=SirajRaval) [Video introduttivo all’utilizzo della Blockchain per la gestione della supplychain]
11. [https://www.youtube.com/watch?v=2RFxO52Go0M&ab\\_channel=CoinBureau](https://www.youtube.com/watch?v=2RFxO52Go0M&ab_channel=CoinBureau) [Video sulle prospettive della Blockchain per la gestione della supplychain dal canale ufficiale Youtube della rivista specialistica Coin Bureau]
12. [https://www.youtube.com/watch?v=HSIfqobiYBQ&ab\\_channel=TheCryptonomist](https://www.youtube.com/watch?v=HSIfqobiYBQ&ab_channel=TheCryptonomist) [Video sulla Blockchain per la gestione della supplychain dal canale ufficiale Youtube della rivista specialistica The Cryptonomist]
13. [https://www.youtube.com/watch?v=Xwqo\\_fwPEJo&list=WL&index=249&ab\\_channel=TradeLens](https://www.youtube.com/watch?v=Xwqo_fwPEJo&list=WL&index=249&ab_channel=TradeLens) [Video sulla supplychain gestita da IBM]
14. [https://www.youtube.com/watch?v=YOr9A\\_TygBk&list=WL&index=258&ab\\_channel=AbhishekSingh](https://www.youtube.com/watch?v=YOr9A_TygBk&list=WL&index=258&ab_channel=AbhishekSingh) [Journey of Coffee - Blockchain Supply Chain - Provenance to Consumption]

15. <https://en.wikipedia.org/wiki/Cryptography> [Definizione Crittografia su Wikipedia]
16. <https://bitmind.it/web/riservezza-autenticita-integrita-disponibilita-e-non-ripudio> [Articolo sulla sicurezza informatica]
17. <https://ico.li/blockchain-validate-data> [Articolo sul mining]
18. <https://blockgeeks.com/guides/what-is-hashing> [Articolo sull'hashing]
19. <https://crushcrypto.com/cryptography-in-blockchain> [Articolo sulla crittografia utilizzata nella tecnologia Blockchain]
20. Algorithms: Explained and Animated (app playstore) - Public-Key Cryptosystem
21. Algorithms: Explained and Animated (app playstore) - Hash function
22. [https://it.wikipedia.org/wiki/Cifrario\\_di\\_Cesare](https://it.wikipedia.org/wiki/Cifrario_di_Cesare) [Cifrario di Cesare Wikipedia]
23. <https://cryptoadventure.org/the-beginners-guide-to-ethereum/> [Guida introduttiva ad Ethereum]
24. <https://www.investopedia.com/articles/investing/031416/bitcoin-vs-ethereum-driven-different-purposes.asp> [Articolo differenze fra Ethereum e Bitcoin]
25. <https://www.upfolio.com/ultimate-ethereum-guide> [Guida completa al mondo Ethereum]
26. <https://crypttheory.org/ethereum-nft-growth-should-be-met-with-caution-says-ceo/> [Articolo sulla crescita del mercato degli nft]
27. <https://ethereum.org/en/> [Sito ufficiale Ethereum]
28. <https://ethereum.org/en/what-is-ethereum/> [Sito ufficiale Ethereum: spiegazione generale]
29. <https://ethereum.org/en/eth/> [Sito ufficiale Ethereum: la valuta eth]
30. <https://ethereum.org/en/defi/> [Sito ufficiale Ethereum: descrizione DeFi]
31. <https://ethereum.org/en/dao/> [Sito ufficiale Ethereum: definizione Dao]
32. <https://ethereum.org/en/nft/> [Sito ufficiale Ethereum: definizione nft]
33. <https://ethereum.org/en/developers/docs/evm/> [Sito ufficiale Ethereum: descrizione Ethereum Virtual Machine]
34. <https://ethereum.org/en/whitepaper/> [Sito ufficiale Ethereum: Ethereum white paper]
35. <http://gavwood.com/paper.pdf> [Paper che tratta nel dettaglio il funzionamento della compilazione degli smartcontract su Ethereum]

36. <https://docs.soliditylang.org/en/latest/> [Documentazione ufficiale solidity]
37. <https://etherscan.io/> [Sito etherscan]
38. <https://ethereum-blockchain-developer.com/028-fallback-view-constructor/02-receive-fallback-function/> [Guida Solidity sulle fallback function]
39. <https://www.investopedia.com/terms/d/doublespending.asp> [Aricolo sul problema della doppia spesa]
40. [https://www.youtube.com/watch?v=rsLrJp6cLf4&ab\\_channel=aantonopaantonop](https://www.youtube.com/watch?v=rsLrJp6cLf4&ab_channel=aantonopaantonop) [Intervento all'università di New York di Andreas Antonopoulos, autore del libro best seller “Mastering Bitcoin” sul proof of work]
41. [https://www.youtube.com/watch?v=LoGx\\_1dRBU0&ab\\_channel=MartinKleppmannMartinKleppmann](https://www.youtube.com/watch?v=LoGx_1dRBU0&ab_channel=MartinKleppmannMartinKleppmann) [Video sul problema dei generali bizantini]
42. [https://www.youtube.com/watch?v=ipwxYa-F1uY&t=2s&ab\\_channel=freeCodeCamp.orgfreeCodeCamp.org](https://www.youtube.com/watch?v=ipwxYa-F1uY&t=2s&ab_channel=freeCodeCamp.orgfreeCodeCamp.org) [Video tutorial Solidity]
43. [https://www.tutorialspoint.com/solidity/solidity\\_view\\_functions.htm](https://www.tutorialspoint.com/solidity/solidity_view_functions.htm) [Tutorial Solidity: View Functions]
44. [https://www.tutorialspoint.com/solidity/solidity\\_pure\\_functions.htm](https://www.tutorialspoint.com/solidity/solidity_pure_functions.htm) [Tutorial Solidity: Pure Functions]
45. [https://www.tutorialspoint.com/solidity/solidity\\_interfaces.htm](https://www.tutorialspoint.com/solidity/solidity_interfaces.htm) [Tutorial Solidity: interfacce]
46. [https://www.tutorialspoint.com/solidity/solidity\\_events.htm](https://www.tutorialspoint.com/solidity/solidity_events.htm) [Tutorial Solidity: eventi]