



*Lezione 4 –
Programmazione
Smart Contracts*

Prof. Esposito Christian

*Corso di
Sicurezza dei Dati*



::: Sommario

- Programmazione di Smart Contracts:
 - Introduzione agli Smart Contracts;
 - Bitcoin, Etherium, Hyperledger Fabric;
 - Linguaggi per la programmazione di smart contracts: Solidity, Go.

... Letture

- <https://ethereum.org/en/developers/>
- <https://hyperledger-fabric.readthedocs.io/en/release-2.2/>
- V. Y. Kemmoe, W. Stone, J. Kim, D. Kim and J. Son, "Recent Advances in Smart Contracts: A Technical Overview and State of the Art", in IEEE Access, vol. 8, pp. 117782-117801, 2020;
- <https://go.dev/>
- <https://docs.soliditylang.org/en/v0.8.17/>



La Programmazione con gli Smart Contracts

::: Smart Contracts (1/4)



Un contratto è un accordo legalmente vincolante che riconosce e disciplina i diritti e i doveri delle parti del contratto.



Uno smart contract è la “trasposizione” in codice di un contratto, mediante funzioni “if/then” incorporate in software o protocolli informatici, per verificare in automatico l’avverarsi di determinate condizioni e di autoeseguire azioni quando le condizioni sono raggiunte e verificate.

E’ un programma deterministico che elabora le informazioni in una blockchain: a parità di input i risultati restituiti saranno identici. Ciò garantisce alle parti una assoluta “certezza di giudizio oggettivo” escludendo qualsiasi forma di interpretazione.

::: Smart Contracts (2/4)

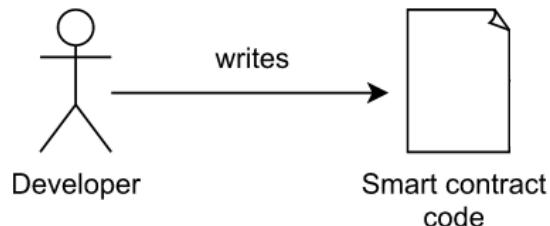
Uno smart contract per blockchain deve soddisfare i seguenti obiettivi: osservabilità, verificabilità, riservatezza e applicabilità. Su una blockchain, uno smart contract non può essere modificato, ma può essere facilmente osservato, verificato, auto-applicato e, a seconda della modalità di accesso della blockchain, è possibile ottenere la privacy.

Ecco un processo graduale di vita di uno smart contract:

::: Smart Contracts (2/4)

Uno smart contract per blockchain deve soddisfare i seguenti obiettivi: osservabilità, verificabilità, riservatezza e applicabilità. Su una blockchain, uno smart contract non può essere modificato né può

Passaggio 1: gli sviluppatori scrivono la logica per lo smart contract in un linguaggio di programmazione supportato dalla piattaforma blockchain che desiderano utilizzare. Utilizzando un compilatore specifico (di solito fornito dalla piattaforma blockchain), compilano il codice sorgente del loro smart contract e ottengono una sua rappresentazione in bytecode.

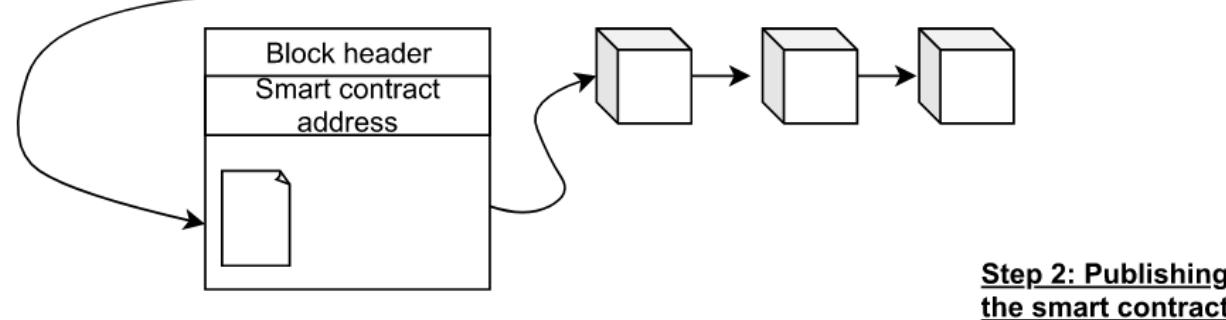
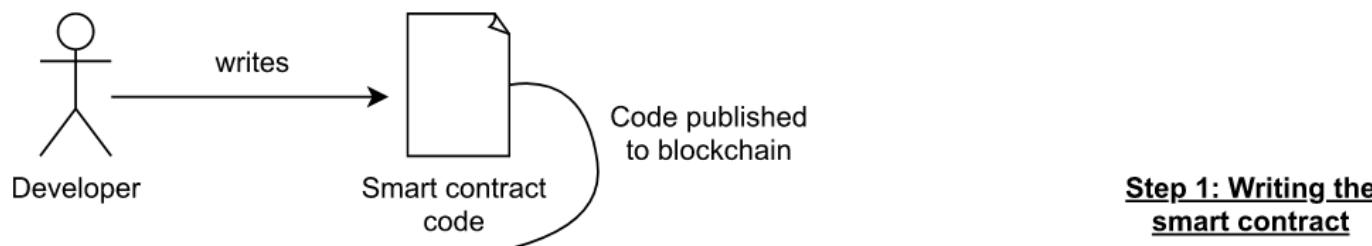


[Step 1: Writing the smart contract](#)

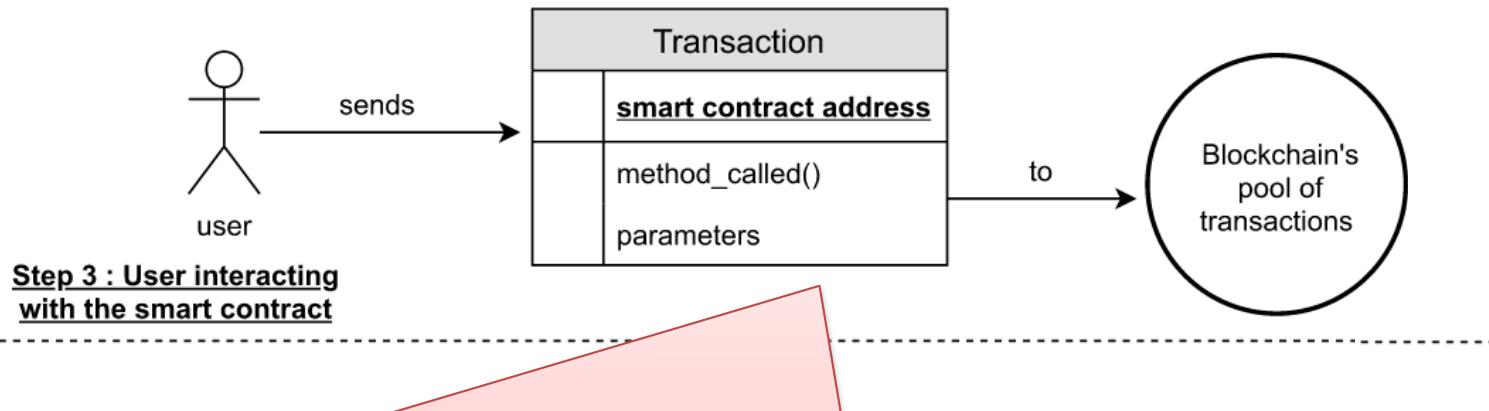
::: Smart Contracts (2/4)

Uno smart contract per blockchain deve soddisfare i seguenti obiettivi: osservabilità, verificabilità, riservatezza e applicabilità. Su una blockchain, uno smart contract non può essere modificato, ma può

Passaggio 2: dopo aver ottenuto il codice byte, lo smart contract è pubblicato sulla piattaforma blockchain ed archiviato. Una volta pubblicato lo smart contract, sarà di sola lettura o modificabile. Nel caso in cui sia di sola lettura, per fornire un aggiornamento, gli sviluppatori dovranno pubblicare una nuova versione dello smart contract e reindirizzare gli utenti ad essa. Una volta caricato, lo smart contract è al suo stato iniziale, pari ai valori iniziali delle variabili interne.



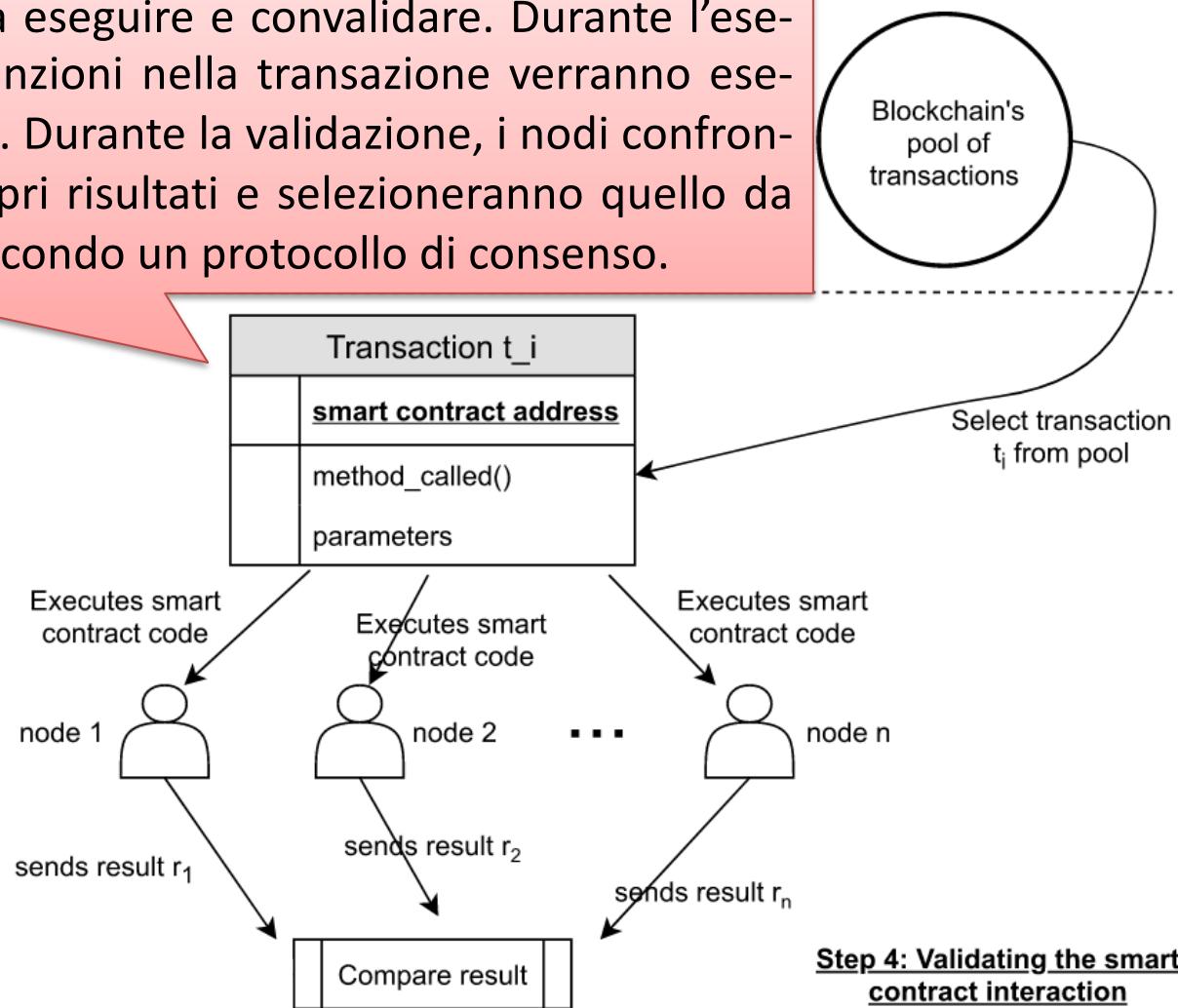
::: Smart Contracts (3/4)



Passaggio 3: l'accesso a un programma di smart contract pubblicato dipende dalla piattaforma blockchain, come un indirizzo restituito quando lo smart contract è caricato nella piattaforma. Tale indirizzo può essere utilizzato per interagire con lo smart contract, inviando le transazioni contenenti la funzione che desiderano utilizzare e gli argomenti della funzione. Se è necessaria una quantità di valuta della piattaforma per avviare l'esecuzione della funzione, tale importo sarà trasferito insieme alla transazione. La transazione verrà archiviata nel pool di transazioni della piattaforma blockchain che attendono di essere eseguite e convalidate.

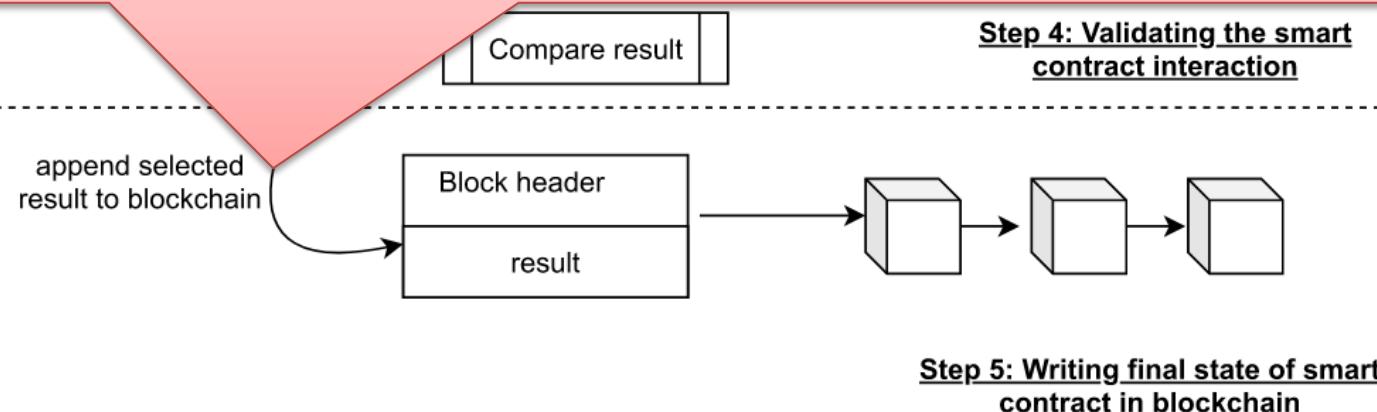
::: Smart Contracts (3/4)

Passaggio 4: la piattaforma blockchain selezionerà le transazioni da eseguire e convalidare. Durante l'esecuzione, le funzioni nella transazione verranno eseguite dai nodi. Durante la validazione, i nodi confronteranno i propri risultati e selezioneranno quello da mantenere secondo un protocollo di consenso.



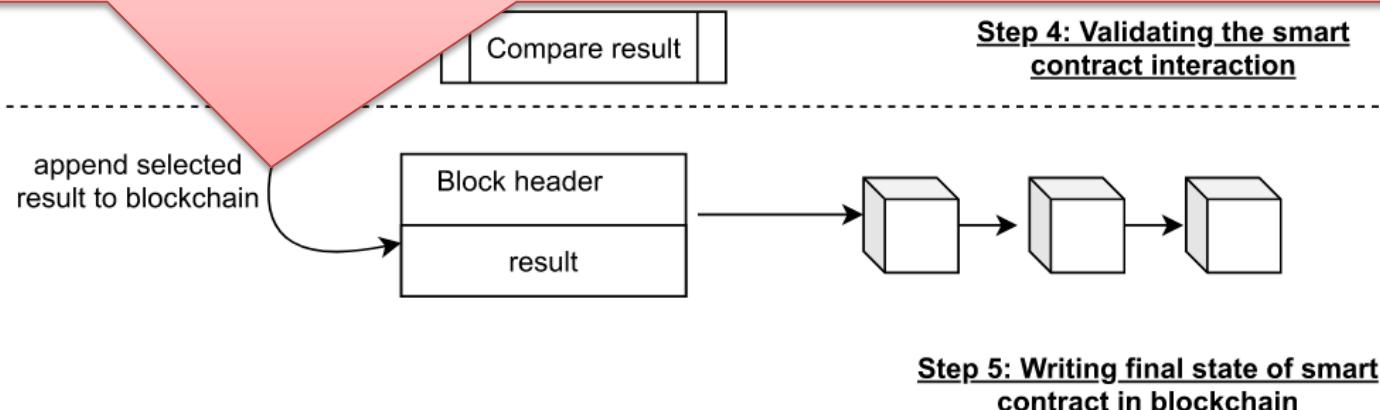
::: Smart Contracts (3/4)

Passaggio 5: Una volta selezionato il risultato valido, verrà inserito in un blocco da aggiungere alla blockchain. Se una transazione validata ha alterato le variabili interne di uno smart contract, i nuovi valori saranno considerati come valori iniziali da transazioni future sullo smart contract.



::: Smart Contracts (3/4)

Passaggio 5: Una volta selezionato il risultato valido, verrà inserito in un blocco da aggiungere alla blockchain. Se una transazione validata ha alterato le variabili interne di uno smart contract, i nuovi valori saranno considerati come valori iniziali da transazioni future sullo smart contract.



L'implementazione di smart contract non è standardizzata e ogni piattaforma blockchain propone una propria soluzione.

Le funzioni di uno smart contract possono essere chiamate direttamente dai client o indirettamente da altri smart contracts. Per garantire che le chiamate terminino, al client viene addebitata una fee ad ogni chiamata e sua durata. Se l'addebito supera quello che il cliente è disposto a pagare, il calcolo viene interrotto e le operazioni annullate.

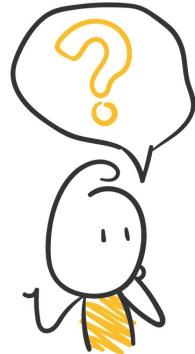
::: Smart Contracts (4/4)

L'attuale approccio di sviluppo di smart contract limita il throughput perché non ammettono concorrenza.

- Quando un miner crea un blocco, assembla una sequenza di transazioni e calcola un nuovo stato provvisorio eseguendo gli smart contract di tali transazioni in serie, nell'ordine in cui si verificano nel blocco.
- Un miner non può eseguire gli smart contracts in parallelo, perché potrebbero sussistere degli accessi in conflitto a dati condivisi e un interleaving arbitrario potrebbe produrre uno stato finale non coerente. Spesso per gli smart contract, non è possibile dire in anticipo se le esecuzioni di contratti siano in conflitto.

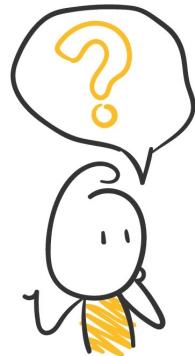
::: Smart Contracts Programming (1/2)

**Dovremmo imparare a
programmare i contratti
intelligenti?**



::: Smart Contracts Programming (1/2)

**Dovremmo imparare a
programmare i contratti
intelligenti?**

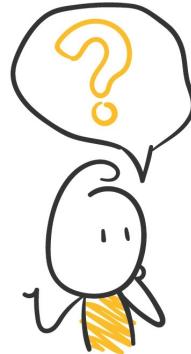


**No Way! Abbiamo
OpenAI e chatGPT!**



::: Smart Contracts Programming (1/2)

**Dovremmo imparare a
programmare i contratti
intelligenti?**



**No Way! Abbiamo
OpenAI e chatGPT!**



ChatGPT è un chatbot di intelligenza artificiale (AI) dove "GPT" sta per Generative Pre-trained Transformer, un tipo di modello linguistico di grandi dimensioni (LLM), ovvero una rete neurale con molti parametri addestrati su grandi quantità di testo senza etichetta.

È stato messo a punto (mediante un approccio definito come trasferimento dell'apprendimento) per applicazioni conversazionali utilizzando una combinazione di tecniche di apprendimento supervisionato e di rinforzo.

::: Smart Contracts Programming (2/2)

The screenshot shows a web browser window with the URL <https://chat.openai.com/?model=text-davinci-002-render-sha>. The chat interface is displayed, with a sidebar on the left showing a list of conversations, including one titled "Identity Management S". The main area shows a message from the AI asking to "Create a smart contract in solidity for identity management". Below this, the AI has provided a sample Solidity code snippet:

```
solidity
pragma solidity ^0.8.0;

contract IdentityManagement {
    struct Identity {
        address owner;
        string name;
        uint256 age;
        string email;
        bool verified;
    }

    mapping(address => Identity) public identities;

    modifier onlyOwner() {
        require(msg.sender == identities[msg.sender].owner, "Only the owner");
    }

    event IdentityCreated(address indexed owner, string name, uint256 age, string email);
    event IdentityUpdated(address indexed owner, string name, uint256 age, string email);
    event IdentityVerified(address indexed owner);
    event IdentityDeleted(address indexed owner);

    function createIdentity(string memory _name, uint256 _age, string memory _email) external {
        require(identities[_owner] == address(0), "Identity already exists");
        identities[_owner] = Identity{owner: msg.sender, name: _name, age: _age, email: _email, verified: false};
        emit IdentityCreated(_owner, _name, _age, _email);
    }

    function updateIdentity(string memory _name, uint256 _age, string memory _email) external onlyOwner {
        identities[msg.sender].name = _name;
        identities[msg.sender].age = _age;
        identities[msg.sender].email = _email;
        emit IdentityUpdated(msg.sender, _name, _age, _email);
    }

    function verifyIdentity(address _owner) external onlyOwner {
        identities[_owner].verified = true;
        emit IdentityVerified(_owner);
    }

    function deleteIdentity(address _owner) external onlyOwner {
        delete identities[_owner];
        emit IdentityDeleted(_owner);
    }
}
```

The code includes comments explaining the structure of the contract, including a struct for Identity, a mapping for identities, and various events and functions for managing identities. A "Copy code" button is visible above the code block.

::: Smart Contracts Programming (2/2)

The screenshot shows a web browser window with the URL <https://chat.openai.com/?model=text-davinci-002-render-sha>. The main content area displays a code snippet:

```
        delete identities[msg.sender];  
  
        emit IdentityDeleted(msg.sender);  
    }  
}
```

In this smart contract, we define an `'Identity'` struct that represents an individual's identity. Each identity contains an owner address, name, age, email, and a verification status.

The `'IdentityManagement'` contract includes the following functions:

- `'createIdentity'`: Allows a user to create their identity by providing their name, age, and email.
- `'updateIdentity'`: Allows the owner of an identity to update their name, age, and email.
- `'verifyIdentity'`: Allows the owner of an identity to verify their identity.
- `'deleteIdentity'`: Allows the owner of an identity to delete their identity.

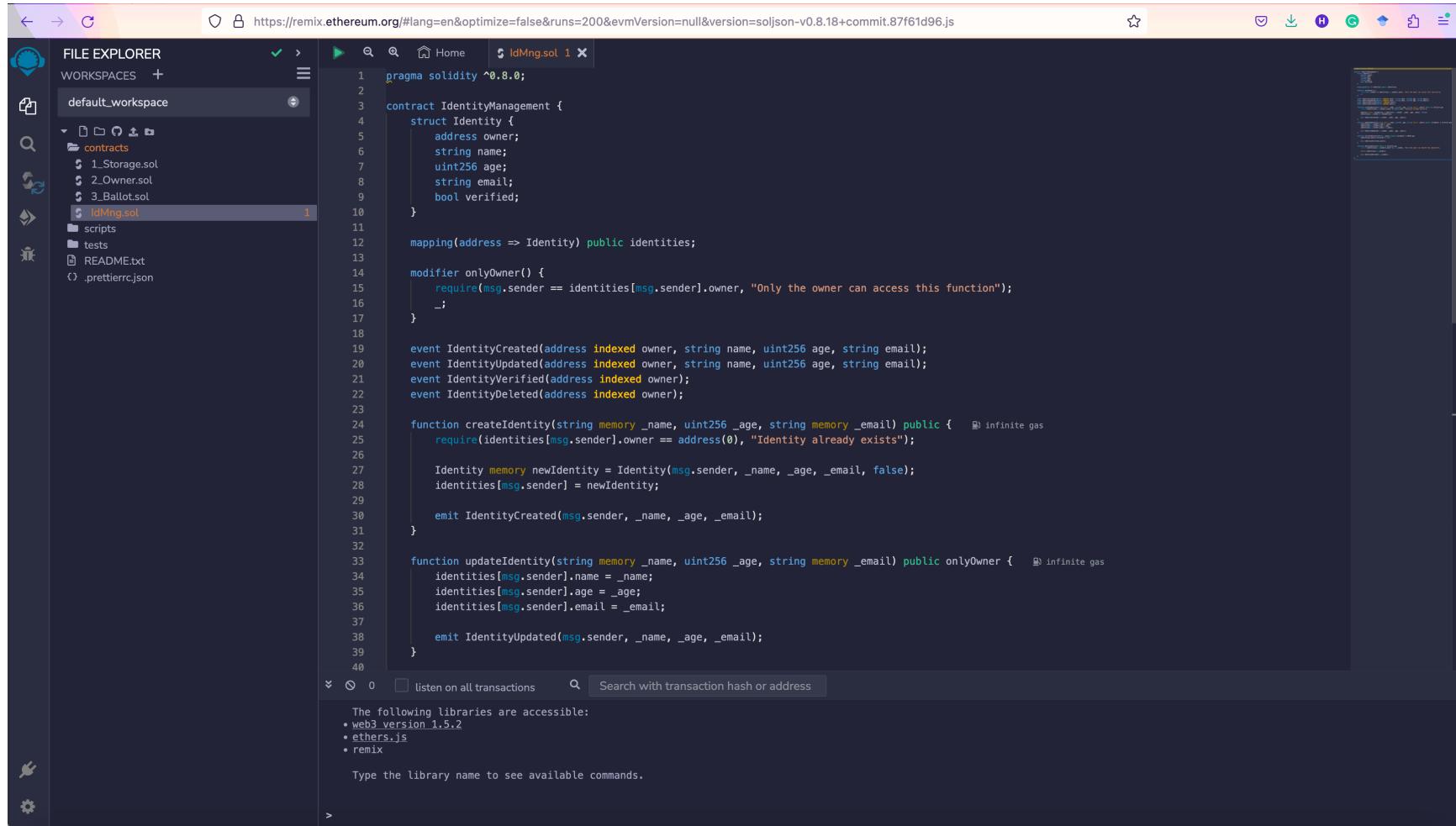
The `'onlyOwner'` modifier ensures that only the owner of an identity can access certain functions.

The contract also emits events for the creation, update, verification, and deletion of identities to provide transparency and enable listeners to react to these events.

Note: This is a simplified example for educational purposes. In real-world scenarios, you may need to consider additional security measures, access control mechanisms, and other requirements depending on your specific use case.

At the bottom of the browser window, there is a message input field with the placeholder "Send a message..." and a "Regenerate response" button.

::: Smart Contracts Programming (2/2)



The screenshot shows the Remix Ethereum IDE interface. The left sidebar is the File Explorer, displaying the workspace structure:

- WORKSPACES +
- default_workspace

 - contracts

 - 1_Storage.sol
 - 2_Owner.sol
 - 3_Ballots.sol
 - IdMng.sol** (selected)

 - scripts
 - tests
 - README.txt
 - .prettierrc.json

The main editor area contains the Solidity code for the **IdMng.sol** contract:

```
pragma solidity ^0.8.0;

contract IdentityManagement {
    struct Identity {
        address owner;
        string name;
        uint256 age;
        string email;
        bool verified;
    }

    mapping(address => Identity) public identities;

    modifier onlyOwner() {
        require(msg.sender == identities[msg.sender].owner, "Only the owner can access this function");
       _;
    }

    event IdentityCreated(address indexed owner, string name, uint256 age, string email);
    event IdentityUpdated(address indexed owner, string name, uint256 age, string email);
    event IdentityVerified(address indexed owner);
    event IdentityDeleted(address indexed owner);

    function createIdentity(string memory _name, uint256 _age, string memory _email) public {
        require(identities[msg.sender].owner == address(0), "Identity already exists");

        Identity memory newIdentity = Identity(msg.sender, _name, _age, _email, false);
        identities[msg.sender] = newIdentity;

        emit IdentityCreated(msg.sender, _name, _age, _email);
    }

    function updateIdentity(string memory _name, uint256 _age, string memory _email) public onlyOwner {
        identities[msg.sender].name = _name;
        identities[msg.sender].age = _age;
        identities[msg.sender].email = _email;

        emit IdentityUpdated(msg.sender, _name, _age, _email);
    }
}
```

The bottom status bar shows:

- listen on all transactions
- Search with transaction hash or address
- The following libraries are accessible:
 - web3 version 1.5.2
 - ethers.js
 - remix
- Type the library name to see available commands.

::: Smart Contracts Programming (2/2)

The screenshot shows the Ethereum Remix IDE interface. The top bar displays the URL <https://remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js>. The left sidebar includes sections for Compiler (version 0.8.18+commit.87f61d96), CONTRACT (IdentityManagement (IdMng.sol)), and various publishing options like Publish on IPFS and Publish on Swarm. A warning message about SPDX license identifier is visible. The main area contains the Solidity code for the IdentityManagement contract, which defines a struct Identity, a mapping of addresses to identities, and several functions for creating, updating, and verifying identities. The bottom right corner shows a terminal window with accessible libraries listed.

```
pragma solidity ^0.8.0;

contract IdentityManagement {
    struct Identity {
        address owner;
        string name;
        uint256 age;
        string email;
        bool verified;
    }

    mapping(address => Identity) public identities;

    modifier onlyOwner() {
        require(msg.sender == identities[msg.sender].owner, "Only the owner can access this function");
       _;
    }

    event IdentityCreated(address indexed owner, string name, uint256 age, string email);
    event IdentityUpdated(address indexed owner, string name, uint256 age, string email);
    event IdentityVerified(address indexed owner);
    event IdentityDeleted(address indexed owner);

    function createIdentity(string memory _name, uint256 _age, string memory _email) public {
        require(identities[msg.sender].owner == address(0), "Identity already exists");

        Identity memory newIdentity = Identity(msg.sender, _name, _age, _email, false);
        identities[msg.sender] = newIdentity;

        emit IdentityCreated(msg.sender, _name, _age, _email);
    }

    function updateIdentity(string memory _name, uint256 _age, string memory _email) public onlyOwner {
        identities[msg.sender].name = _name;
        identities[msg.sender].age = _age;
        identities[msg.sender].email = _email;

        emit IdentityUpdated(msg.sender, _name, _age, _email);
    }
}
```

The following libraries are accessible:

- web3 version 1.5.2
- ethers.js
- remix

Type the library name to see available commands.

::: Smart Contracts Programming (2/2)

The screenshot shows the Solidity IDE interface with the following components:

- FILE EXPLORER**: Shows the workspace structure with contracts and artifacts.
- WORKSPACES**: Default workspace selected.
- COMPILER**: Version 0.8.18+commit.87f61d96.
- CONTRACT**: IdentityManagement (IdMng.) selected.
- PUBLISH**: Options for publishing to IPFS or Swarm.
- COMPILE AND RUN**: Buttons for "Compile IdMng.sol" and "Compile and Run script".
- Warning Message**: A tooltip about SPDX license identifier.
- CODE EDITOR**: The Solidity code for the IdentityManagement contract.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

contract IdentityManagement {
    struct Identity {
        address owner;
        string name;
        uint256 age;
        string email;
        bool verified;
    }

    mapping(address => Identity) put;

    modifier onlyOwner() {
        require(msg.sender == identity.owner);
    }

    function updateIdentity(string memory _name, uint256 _age, string memory _email) public onlyOwner {
        identities[msg.sender].name = _name;
        identities[msg.sender].age = _age;
        identities[msg.sender].email = _email;

        emit IdentityUpdated(msg.sender, _name, _age, _email);
    }
}
```

::: Smart Contracts Programming (2/2)

The screenshot shows the Ethereum Remix IDE interface. On the left, there's a sidebar with various icons for deploying contracts, managing accounts, and interacting with the blockchain. The main area displays the Solidity code for a smart contract named `IdentityManagement`.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

contract IdentityManagement {
    struct Identity {
        address owner;
        string name;
        uint256 age;
        string email;
        bool verified;
    }

    mapping(address => Identity) public identities;

    modifier onlyOwner() {
        require(msg.sender == identities[msg.sender].owner, "Only the owner can access this function");
        ;
    }

    event IdentityCreated(address indexed owner, string name, uint256 age, string email);
    event IdentityUpdated(address indexed owner, string name, uint256 age, string email);
    event IdentityVerified(address indexed owner);
    event IdentityDeleted(address indexed owner);

    function createIdentity(string memory _name, uint256 _age, string memory _email) public {
        require(identities[msg.sender].owner == address(0), "Identity already exists");

        Identity memory newIdentity = Identity(msg.sender, _name, _age, _email, false);
        identities[msg.sender] = newIdentity;

        emit IdentityCreated(msg.sender, _name, _age, _email);
    }

    function updateIdentity(string memory _name, uint256 _age, string memory _email) public onlyOwner {
        identities[msg.sender].name = _name;
        identities[msg.sender].age = _age;
        identities[msg.sender].email = _email;

        emit IdentityUpdated(msg.sender, _name, _age, _email);
    }
}
```

The interface includes fields for setting the environment (Remix VM (Shanghai)), account (0x5B3...eddC4), gas limit (3000000), and value (0 Wei). It also shows a list of deployed contracts, specifically `IDENTITYMANAGEMENT AT 0xD8E`. At the bottom, there are logs and a debug bar.

::: Smart Contracts Programming (2/2)

The screenshot shows the Remix IDE interface with the following details:

- Deploy & Run Transactions**: Environment is set to **Remix VM (Shanghai)**, Account to **0x5B3...eddC4 (99.99999999999999)**, Gas Limit to **3000000**.
- Deployed Contracts**: Contract **IDENTITYMANAGEMENT AT 0xD81** is deployed.
- createIdentity** function parameters:
 - _name**: ChristianEsposito
 - _age**: 41
 - _email**: esposito@unisa.it
- Contract Code (Solidity)**:

```
21 event IdentityCreated
22 event IdentityUpdated
23 event IdentityVerified
24 event IdentityDeleted
25
26 function createIdentity(string memory _name, uint256 _age, string memory _email) public {
27     require(identities[msg.sender].owner == address(0), "Identity already exists");
28
29     Identity memory newIdentity = Identity(msg.sender, _name, _age, _email, false);
30     identities[msg.sender] = newIdentity;
31
32     emit IdentityCreated(msg.sender, _name, _age, _email);
33 }
34
35 function updateIdentity(string memory _name, uint256 _age, string memory _email) public onlyOwner {
36     identities[msg.sender].name = _name;
37     identities[msg.sender].age = _age;
38     identities[msg.sender].email = _email;
39
40     emit IdentityUpdated(msg.sender, _name, _age, _email);
41 }
42
43 event IdentityUpdated(address indexed owner, string name, uint256 age, string email);
44 event IdentityVerified(address indexed owner);
45 event IdentityDeleted(address indexed owner);
```
- Interactions**:
 - createIdentity**: `string _name, uint256 _age, string _email`
 - deleteIdentity**
 - updateIdentity**: `string _name, uint256 _age, string _email`
 - verifyIdentity**: `address _owner`
 - identities**: `address`
- Low level interactions**:
 - CALDATA**: `Transact`
- Logs**:
 - [vm] from: 0x5B3...eddC4 to: IdentityManagement.(constructor) value: 0 wei data: 0x608...20033 logs: 0 hash: 0x5aa...4a00f
- Debug**: `Debug` button.



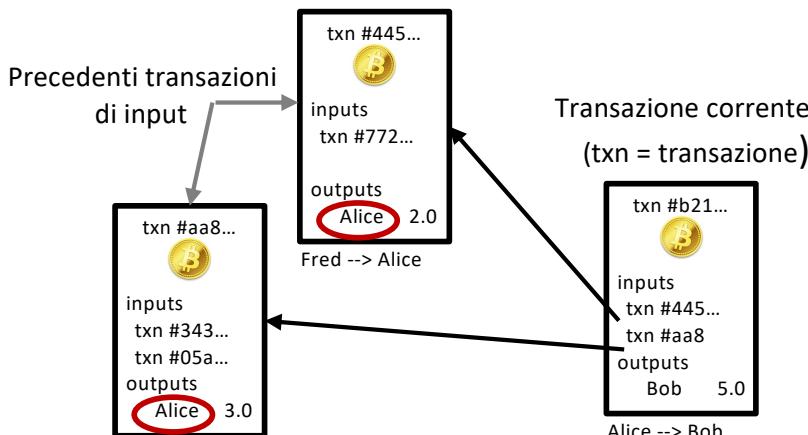
Principali Piattaforme - Bitcoin

::: Bitcoin (1/6)

La blockchain della criptovaluta Bitcoin contiene blocchi di transazioni codificate secondo il modello UTXO (Unspent Transaction Output).

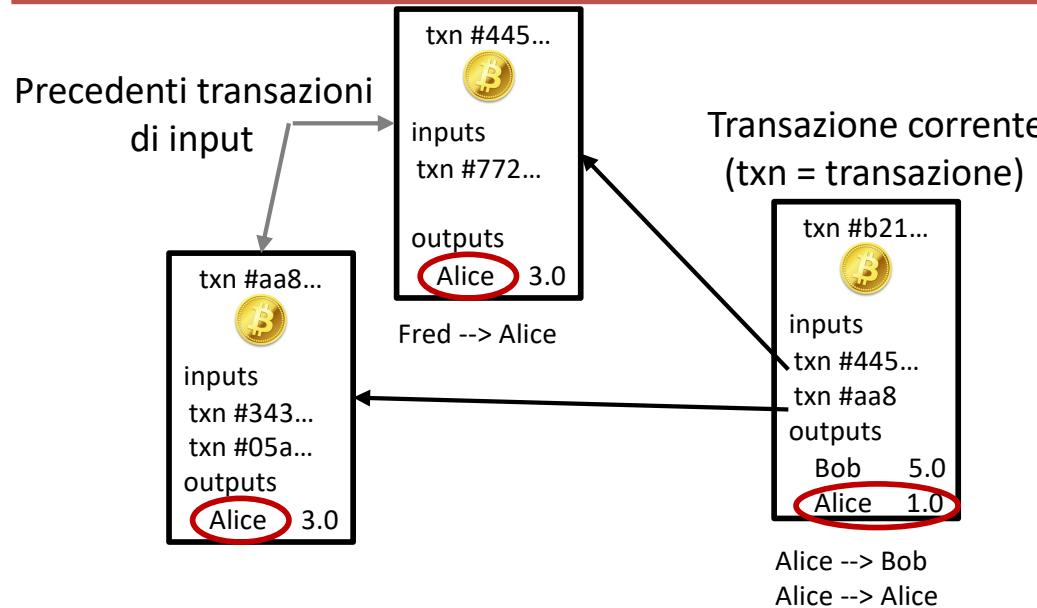
a3e3a13effbb1cd63e8385664a9c60a835e7538dc4c4284f2442fafdd6535851	2018-12-04 16:16:54
12WRaGy2ZGVWo26ViuSbAvaFJ9iewcUb9	34HM3RLF4jsFs2NSqHvWSAGo32aFVqFJNe
	1.9980544 BTC
	1.9980544 BTC
6163aff24a5d9f14db926750016934ad053f0a71039bf2f361fb6ef393b7447	2018-12-04 16:13:58
1GwkXR56uFJaRVkNACXTVJ8LK5hChpXXaE	34Eb81PPtNrchaXVdYpcndvDP6anVhPgEq
	1MD5wKfxfCNwpxc6TcX19gz4GA3chPHcS
	0.5 BTC
	2.99986064 BTC
	3.49986064 BTC

Ogni transazione ha dei Bitcoin in ingresso e in uscita. In ingresso, si ha l'indirizzo delle transazioni mai usate in precedenti transazioni e la cui somma degli output equivale l'output della transazione che le contiene.



I destinatari di una transazione sono identificati da un indirizzo Bitcoin, un identificatore di 26-35 caratteri. Gli indirizzi possono essere generati gratuitamente da qualsiasi utente.

::: Bitcoin (2/6)

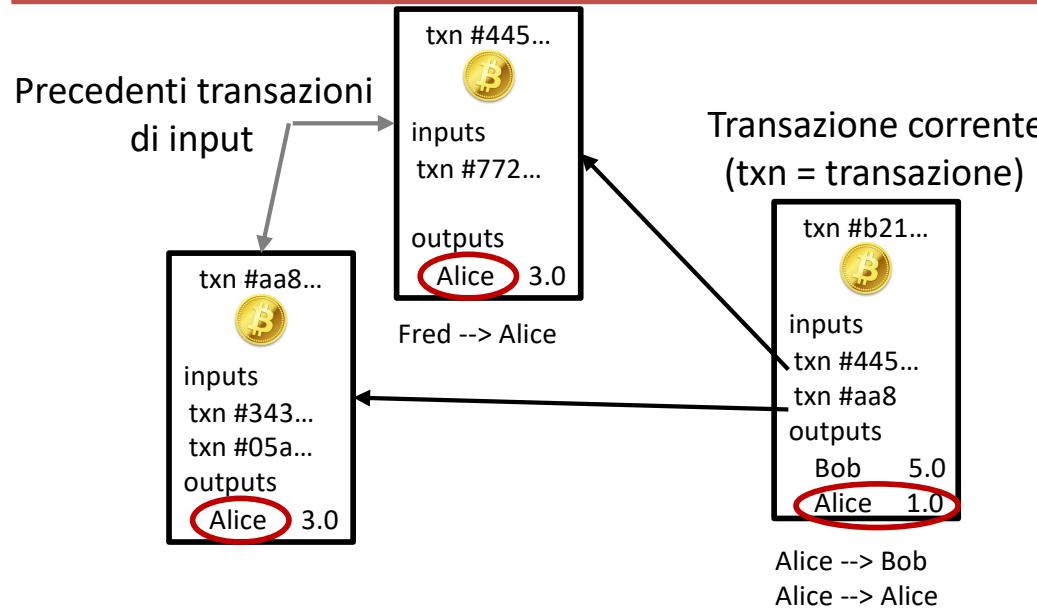


Spesso è difficile avere transazioni che sommate risultano esattamente pari alla quantità da trasferire. L'eccedenza deve essere trasferita all'autore della transazione.

Il modello UTXO non permette di ritrovare sulla blockchain il saldo corrente di un determinato utente, che deve essere ricostruito raccogliendo tutte le transazioni che presentano un determinato utente in output e quelle con tali transazioni in input.

UTXO consente transazioni parallele perché non esiste alcun account, inoltre consente l'anonimato dal momento che un utente può disporre di multipli indirizzi Bitcoin. Essendo stateless rende complesso la realizzazione di applicazione come smart contracts.

::: Bitcoin (2/6)



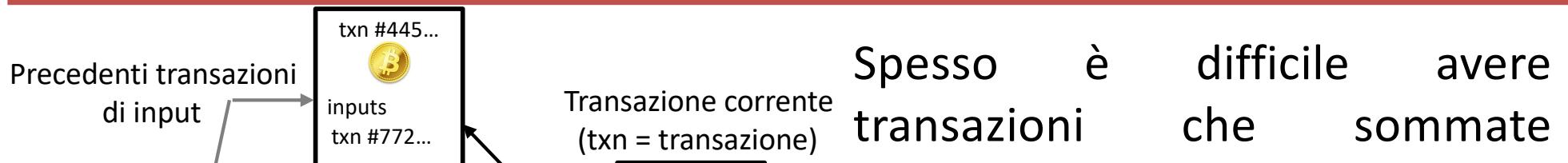
Spesso è difficile avere transazioni che sommate risultano esattamente pari alla quantità da trasferire. L'eccedenza deve essere trasferita all'autore della transazione.

Il modello UTXO non permette di ritrovare sulla blockchain il saldo corrente di un determinato utente, che deve essere ricostruito

La natura stateless della rete consente di ottenere resilienza ed elasticità, oltre che la possibilità per qualsiasi istanza di eseguire qualsiasi attività.

UTXO consente transazioni parallele perché non esiste alcun account, inoltre consente l'anonimato dal momento che un utente può disporre di multipli indirizzi Bitcoin. Essendo **stateless**, rende complesso la realizzazione di applicazione come smart contracts.

::: Bitcoin (2/6)



Gli utenti hanno molti indirizzi Bitcoin, che fungono da pseudonimi per garantire la privacy, e si suole impiegare un indirizzo per ogni transazione:

- Un modo ingenuo per accettare pagamenti in Bitcoin è dire ai propri clienti di inviare denaro a un determinato indirizzo.
- Essendo le transazioni Bitcoin pubbliche, se un cliente Alice invia Bitcoin, un agente dannoso Bob potrebbe vedere tale transazione e inviare un'e-mail affermando che è stato l'autore del pagamento.
- Il destinatario del pagamento non è in grado di distinguere se il trasferimento è stato effettivamente svolto da Bob o Alice.
- Per questo motivo si usa indicare un indirizzo per ogni cliente che deve effettuare un pagamento.

UTXO consente operazioni parallele perché non esiste alcun account, inoltre consente di minimizzare il momento che un utente può disporre di multipli indirizzi Bitcoin. Essendo stateless rende complesso la realizzazione di applicazioni come smart contracts.

::: Bitcoin (3/6)

È possibile impiegare una transazione conoscendo il suo identificativo, che è pubblico. Sorge il problema di evitare che utenti maliziosi impieghino transazioni che non sono proprie, e per formulare dei meccanismi di controllo di autenticazione della legittimità di utilizzo, è stato realizzato un sistema di scripting su Bitcoin, precursore degli smart contract.

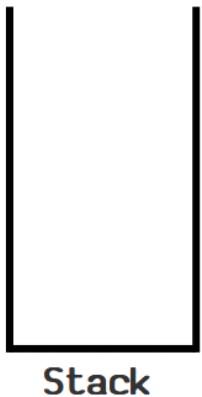
Script è un semplice programma imperativo, basato su stack ed elaborato da sinistra a destra:

- Uno script è essenzialmente un elenco di istruzioni registrate con ogni transazione che descrivono come la prossima persona che desidera spendere i Bitcoin trasferiti può accedervi.
- Una transazione è valida se nulla genera un errore e l'elemento in cima allo stack è True (diverso da zero) alla fine dello script.
- Gli stack contengono vettori di byte.

::: Bitcoin (4/6)

Il linguaggio di scripting ha due tipi di istruzioni:

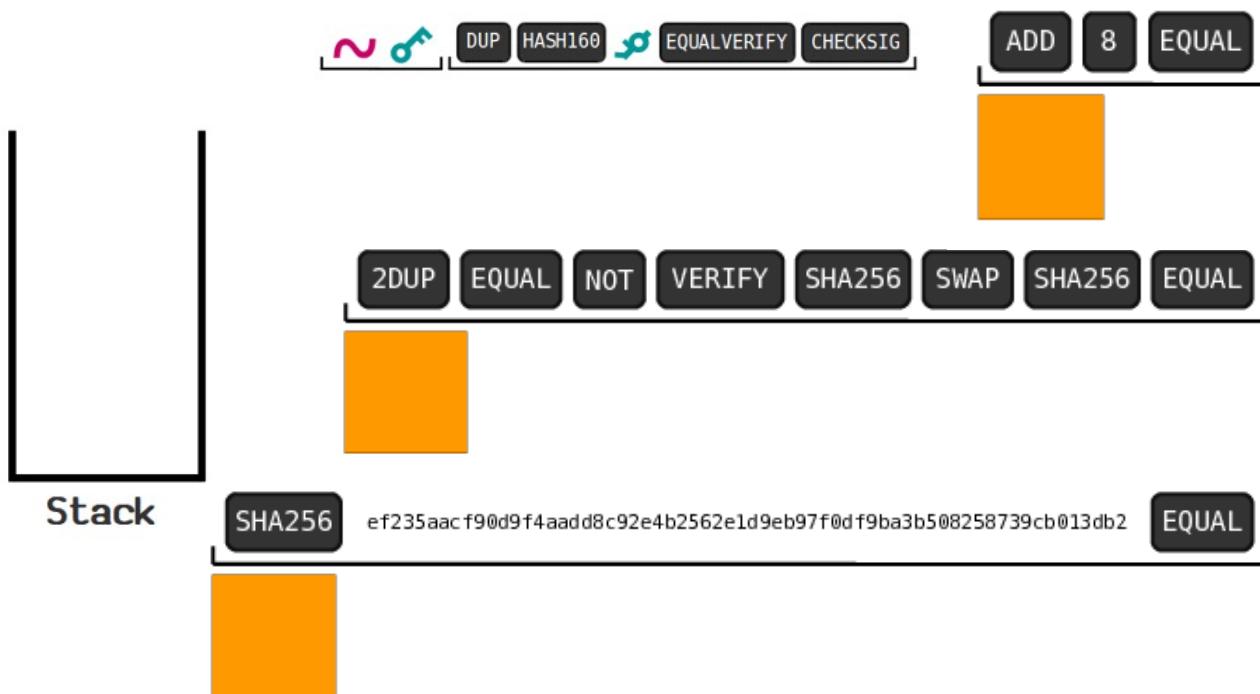
- Le istruzioni di dati contengono semplicemente un valore e sono racchiuse tra parentesi angolari (cioè <data>).
- Gli OP_CODE sono operazioni specifiche appartenenti al linguaggio Bitcoin Scripting che agisce sul valore in cima allo stack e pone il loro risultato anche in cima allo stack.



::: Bitcoin (4/6)

Il linguaggio di scripting ha due tipi di istruzioni:

- Le istruzioni di dati contengono semplicemente un valore e sono racchiuse tra parentesi angolari (cioè <data>).
- Gli OP_CODE sono operazioni specifiche appartenenti al linguaggio Bitcoin Scripting che agiscono sul valore in cima allo stack e pone il loro risultato anche in cima allo stack.



Per spendere questo output, è necessario fornire due numeri la cui somma è 8.

Per spendere questo output, vanno fornite due diverse stringhe di dati che producono lo stesso risultato hash.

Per spendere questo output, va dato qualcosa che ha lo stesso hash di ciò che è all'interno dello script di blocco.

::: Bitcoin (5/6)

```
"vout": [
  {
    "value": 0.10000000,
    "n": 0,
    "scriptPubKey": {
      "asm": "OP_DUP OP_HASH160
7f9bla7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG",
      "hex": "76a9147f9bla7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",
      "reqSigs": 1,
      "type": "pubkeyhash",
      "addresses": [
        "1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK"
      ]
    }
  }
]
```

Ogni codice operativo ha una corrispondente rappresentazione esadecimale, che viene usata per fornire la sua codifica.

Uno locking script viene posizionato su ogni output creato in una transazione, mentre è necessario fornire uno unlocking script per ogni input che si desidera spendere in una transazione.



::: Bitcoin (6/6)

Transazione precedente

```
txn #3be...
Bitcoin icon

"txin": [
  {
    "prev_out": [
      "hash": "...",
      "n": 0
    ],
    "scriptSig": "..."
  },
  ...
],
"txout": [
  {
    "prev_out": [
      "value": "10.12287097",
      "scriptPubKey": "OP_DUP..."
    ],
    ...
  }
]
```

Transazione corrente

```
txn #b21...
Bitcoin icon

"txin": [
  {
    "prev_out": [
      "hash": "3be...",
      "n": 0
    ],
    "scriptSig": "3044..."
  },
  ...
],
"txout": [
  {
    ...
  }
]
```

Anche se l'*unlocking script* è fornito dopo del *locking script* iniziale, in realtà viene messo per primo quando si eseguono entrambi gli script.



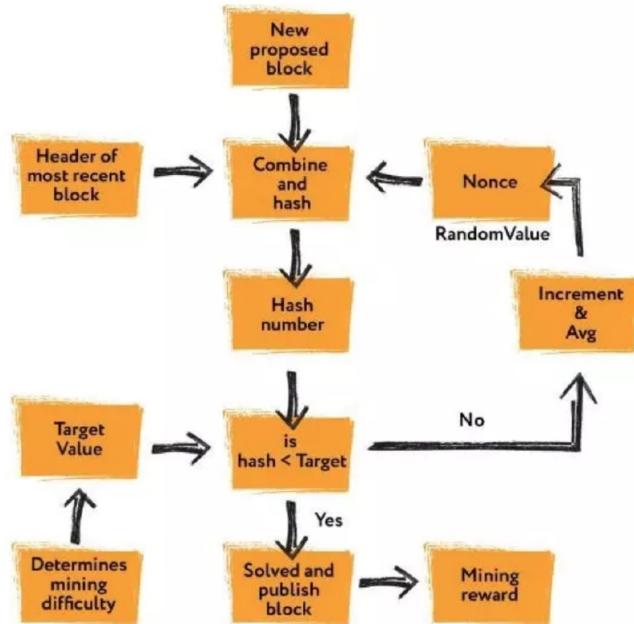
Il linguaggio non è Turing-completo: non ci sono istruzioni condizionali e cicli, perché si vuole predicitività del tempi di esecuzione, che dipende deterministicamente dal numero di istruzioni in esse contenute. Inoltre, è senza stato, non hanno informazione della valuta sbloccata, o non accede alle informazioni del blocco che li contiene.



Principali Piattaforme – oltre Bitcoin

::: Pooled Mining (1/3)

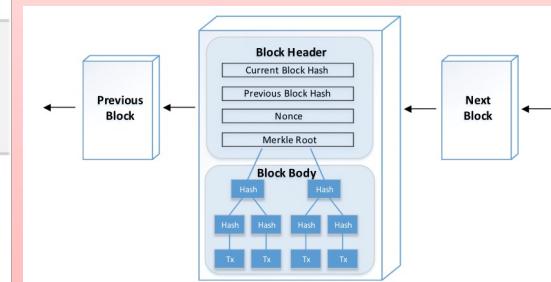
Le criptovalute basate sul consenso Nakamoto utilizzano enormi risorse computazionali per il loro mining. Trovare una soluzione valida a un puzzle PoW è un processo probabilistico, che segue una distribuzione di Poisson. La probabilità di un minatore di trovare una soluzione entro un'epoca è determinata dalla frazione di potenza di calcolo che possiede nella rete. I minatori con una potenza di calcolo modesta possono avere una varianza estremamente elevata.



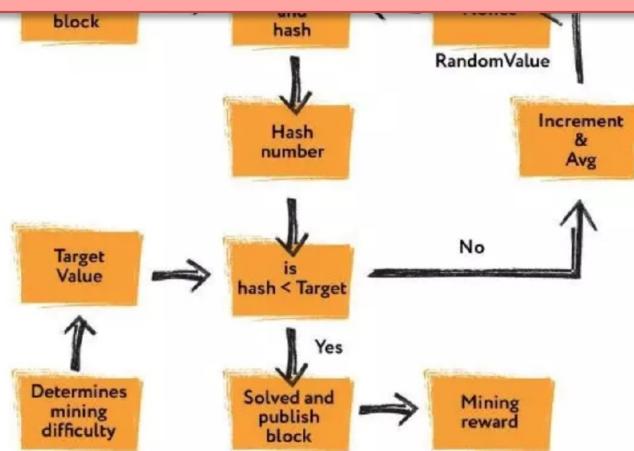
Ogni volta si applica un doppio SHA-256 all'intestazione del blocco e se l'output è maggiore dell'attuale obiettivo di difficoltà Bitcoin, si deve incrementare il valore nonce e ritentare nuovamente. Quando l'output corrisponde a un numero inferiore, il miner ha trovato una soluzione alla Proof-of-Work.

::: Pooled Mining (1/3)

Bytes	Name	Type	Description
4	version	int32_t	The block version number indicates which set of block validation rules to follow. See the list of block versions below.
32	previous block header hash	char[32]	A SHA256(SHA256()) hash in internal byte order of the previous block's header. This ensures no previous block can be changed without also changing this block's header.
32	merkle root hash	char[32]	A SHA256(SHA256()) hash in internal byte order. The merkle root is derived from the hashes of all transactions included in this block, ensuring that none of those transactions can be modified without modifying the header. See the merkle trees section below.
4	time	uint32_t	The block time is a Unix epoch time when the miner started hashing the header (according to the miner). Must be strictly greater than the median time of the previous 11 blocks. Full nodes will not accept blocks with headers more than two hours in the future according to their clock.
4	nBits	uint32_t	An encoded version of the target threshold this block's header hash must be less than or equal to. See the nBits format described below.
4	nonce	uint32_t	An arbitrary number miners change to modify the header hash in order to produce a hash less than or equal to the target threshold. If all 32-bit values are tested, the time can be updated or the coinbase transaction can be changed and the merkle root updated.



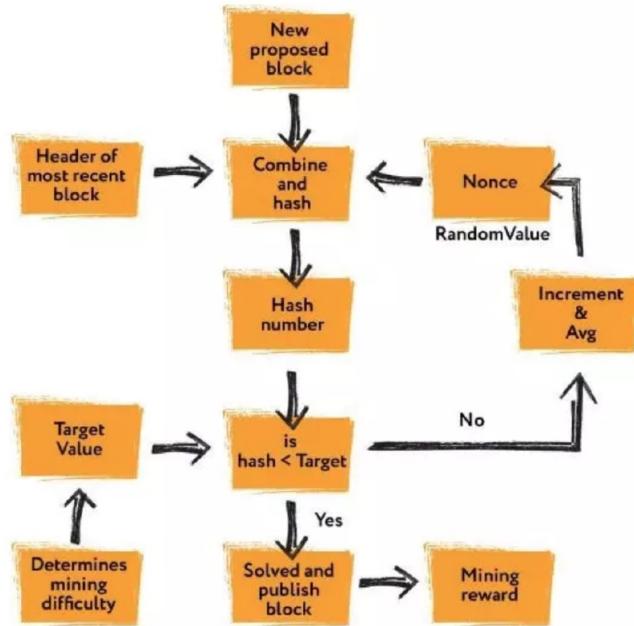
normi risorse
valida a un
distribuzione
na soluzione
calcolo che
olo modesta



all'**intestazione del blocco** e se l'output è maggiore dell'attuale obiettivo di difficoltà Bitcoin, si deve incrementare il valore nonce e ritentare nuovamente. Quando l'output corrisponde a un numero inferiore, il miner ha trovato una soluzione alla Proof-of-Work.

::: Pooled Mining (1/3)

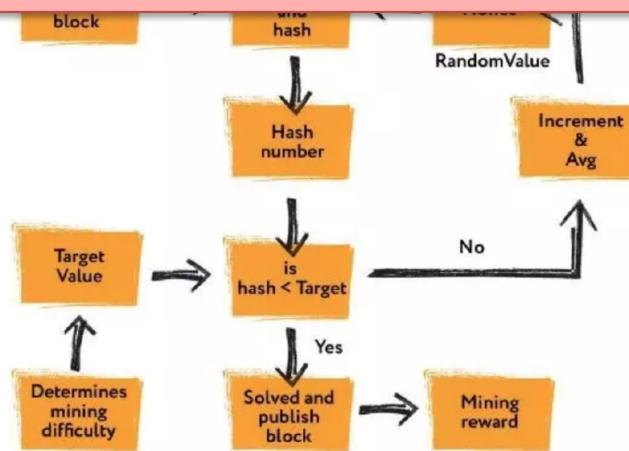
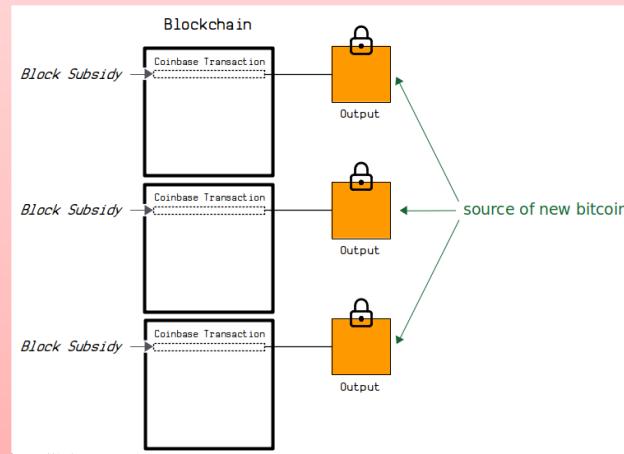
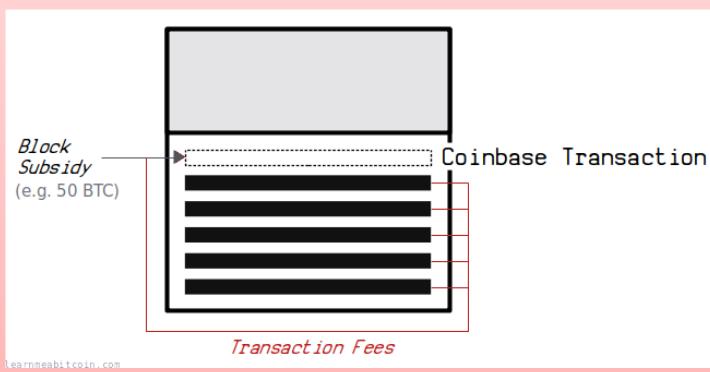
Le criptovalute basate sul consenso Nakamoto utilizzano enormi risorse computazionali per il loro mining. Trovare una soluzione valida a un puzzle PoW è un processo probabilistico, che segue una distribuzione di Poisson, con la probabilità di un minatore di trovare una soluzione entro un'epoca determinata dalla frazione di potenza di calcolo che possiede nella rete. I minatori con una potenza di calcolo modesta possono avere una varianza estremamente elevata.



La reward è la somma del premio del blocco e delle commissioni di tutte le transazioni incluse nel blocco. È codificata nella transazione coinbase, che non utilizza gli UTXO come input, ma essenzialmente genera nuovi Bitcoin dal nulla. coinbase ha un unico output, ovvero l'indirizzo Bitcoin del miner.

::: Pooled Mining (1/3)

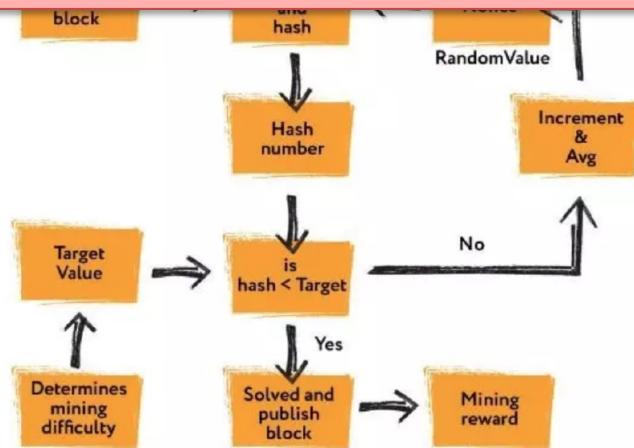
Una transazione coinbase è la prima transazione in un blocco, inserita dal miner quando costruisce il suo blocco candidato in modo che possa reclamare la ricompensa del blocco (premio del blocco + commissioni) se riesce il mining il blocco con successo.



delle commissio... tutte le transazioni incluse nel blocco. È codificata nella transazione **coinbase**, che non utilizza gli UTXO come input, ma essenzialmente genera nuovi Bitcoin dal nulla. coinbase ha un unico output, ovvero l'indirizzo Bitcoin del miner.

normi risorse
valida a un
distribuzione
na soluzione
calcolo che
olo modesta

... Pooled Mining (1/3)

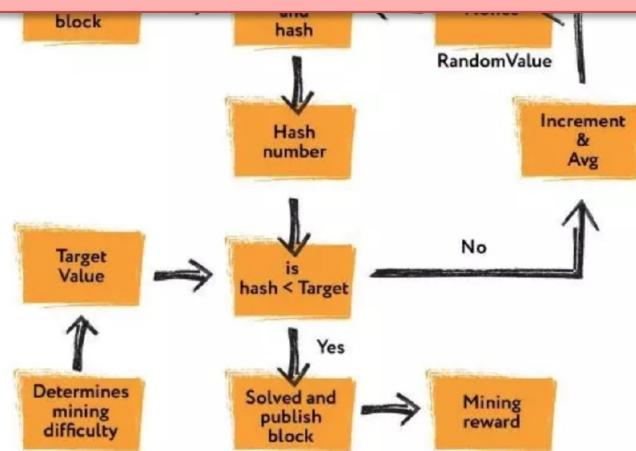


delle commissioni transazioni incluse nel blocco. È indicata nella transazione **coinbase**, che non utilizza gli UTXO come input, ma essenzialmente genera nuovi Bitcoin dal nulla. coinbase ha un unico output, ovvero l'indirizzo Bitcoin del miner.

::: Pooled Mining (1/3)

```
{  
    "version": "01000000",  
    "inputcount": "01",  
    "inputs": [  
        {  
            "txid": "0000000000000000000000000000000000000000000000000000000000000000",  
            "vout": "ffffffff",  
            "scriptsigsize": "08",  
            "scriptsig": "04233fa04e028b12",  
            "sequence": "ffffffff"  
        }  
    ],  
    "outputcount": "01",  
    "outputs": [  
        {  
            "amount": "30490b2a01000000",  
            "scriptpubkeysize": "43",  
            "scriptpubkey": "41047eda6bd04fb27cab6e7c28c99b94977f073e912f25d1ff7165d9c95cd9bbe6da7e7ad7f2acb09e0ced91705f7616af53bee51a2  
38b7dc527f2be0aa60469d140ac"  
        }  
    ],  
    "locktime": "00000000"  
}  
  
Transaction: 18a16d322b235f636ab90e62e79a9f20a0b9c14e8da51e9dc0974f99f82ee444
```

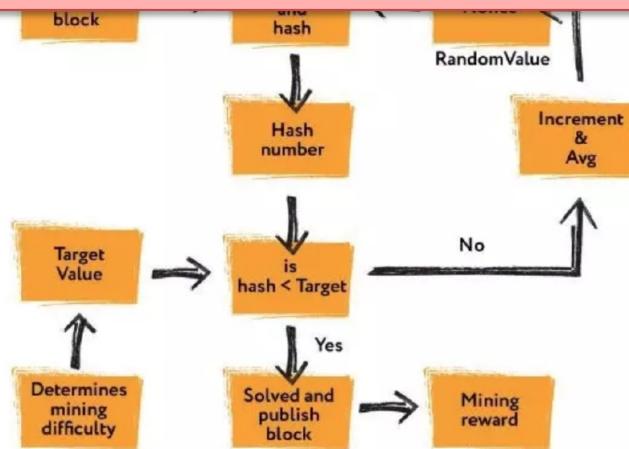
ScriptSig non decodifica in alcun testo ASCII in particolare, perché non c'è nulla da sbloccare. Il miner stava usando questo campo solo come ExtraNonce.



delle commissioni e delle transazioni incluse nel blocco. È codificata nella transazione **coinbase**, che non utilizza gli UTXO come input, ma essenzialmente genera nuovi Bitcoin dal nulla. coinbase ha un unico output, ovvero l'indirizzo Bitcoin del miner.

... Pooled Mining (1/3)

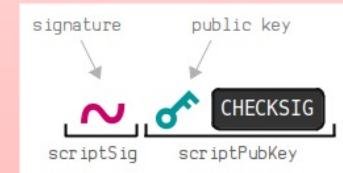
C'è solo un output e ha una quantità di 5000350000 sat. Il premio del blocco era di 5000000000 sat (50 BTC). Gli altri 350000 sat erano le commissioni raccolte dalle altre 10 transazioni incluse nel blocco.



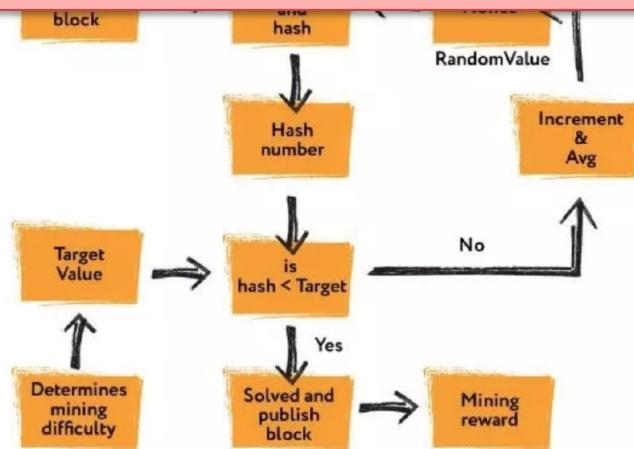
delle commissioni transazioni incluse nel blocco. È indicata nella transazione **coinbase**, che non utilizza gli UTXO come input, ma essenzialmente genera nuovi Bitcoin dal nulla. coinbase ha un unico output, ovvero l'indirizzo Bitcoin del miner.

... Pooled Mining (1/3)

L'output della coinbase è stato bloccato sulla propria chiave pubblica usando uno script di blocco P2PK.



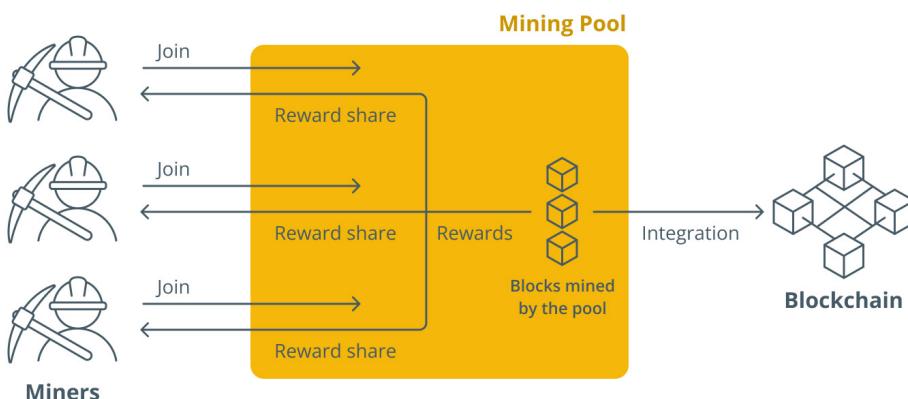
Per sbloccare un P2PK è sufficiente fornire una firma valida.



delle commissioni transazioni
incluse nel blocco. È indicata nella
transazione **coinbase**, che non utilizza gli
UTXO come input, ma essenzialmente genera
nuovi Bitcoin dal nulla. coinbase ha un unico
output, ovvero l'indirizzo Bitcoin del miner.

::: Pooled Mining (1/3)

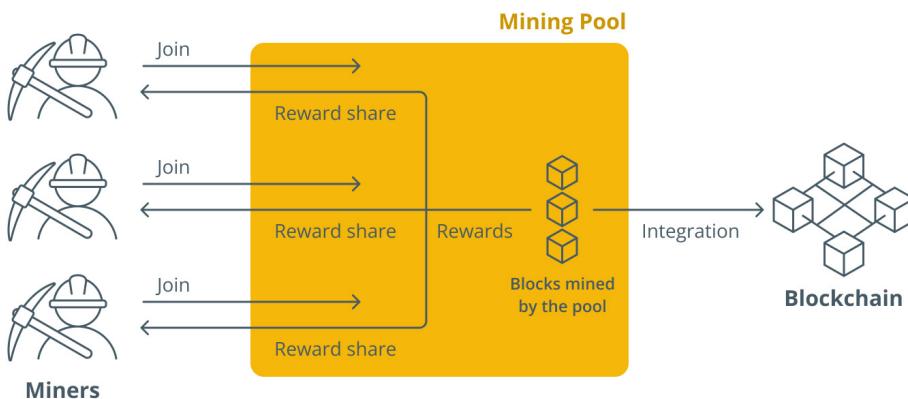
Le criptovalute basate sul consenso Nakamoto utilizzano enormi risorse computazionali per il loro mining. Trovare una soluzione valida a un puzzle PoW è un processo probabilistico, che segue una distribuzione di Poisson, con la probabilità di un minatore di trovare una soluzione entro un'epoca determinata dalla frazione di potenza di calcolo che possiede nella rete. I minatori con una potenza di calcolo modesta possono avere una varianza estremamente elevata.



Per ridurre la varianza, i minatori si uniscono in gruppi o pool per estrarre collaborativamente i blocchi e condividere insieme i premi.

::: Pooled Mining (1/3)

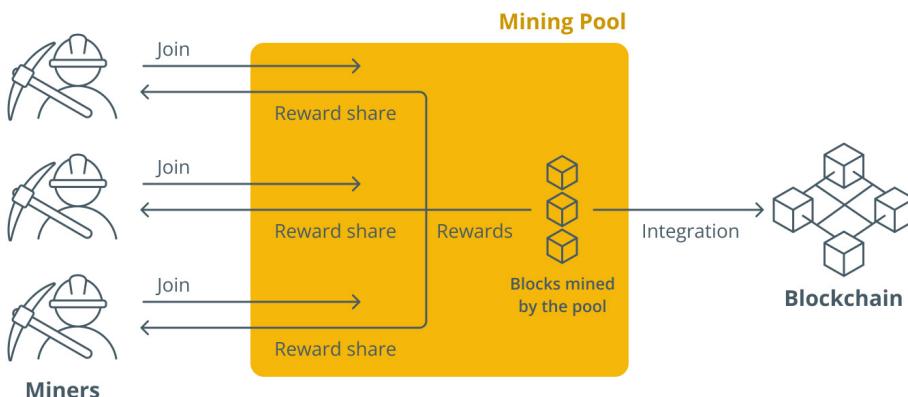
Le criptovalute basate sul consenso Nakamoto utilizzano enormi risorse computazionali per il loro mining. Trovare una soluzione valida a un puzzle PoW è un processo probabilistico, che segue una distribuzione di Poisson, con la probabilità di un minatore di trovare una soluzione entro un'epoca determinata dalla frazione di potenza di calcolo che possiede nella rete. I minatori con una potenza di calcolo modesta possono avere una varianza estremamente elevata.



Un membro del pool è responsabile della distribuzione agli altri membri di un sotto-puzzle di difficoltà inferiore rispetto all'intero puzzle PoW del blocco.

::: Pooled Mining (1/3)

Le criptovalute basate sul consenso Nakamoto utilizzano enormi risorse computazionali per il loro mining. Trovare una soluzione valida a un puzzle PoW è un processo probabilistico, che segue una distribuzione di Poisson, con la probabilità di un minatore di trovare una soluzione entro un'epoca determinata dalla frazione di potenza di calcolo che possiede nella rete. I minatori con una potenza di calcolo modesta possono avere una varianza estremamente elevata.



Quando la soluzione inviata da un minatore produce un blocco valido, il manager del pool lo invia alla rete e ottiene la ricompensa del blocco.

La ricompensa sarà equamente divisa tra tutti i membri del pool in proporzione alle soluzioni fornite.

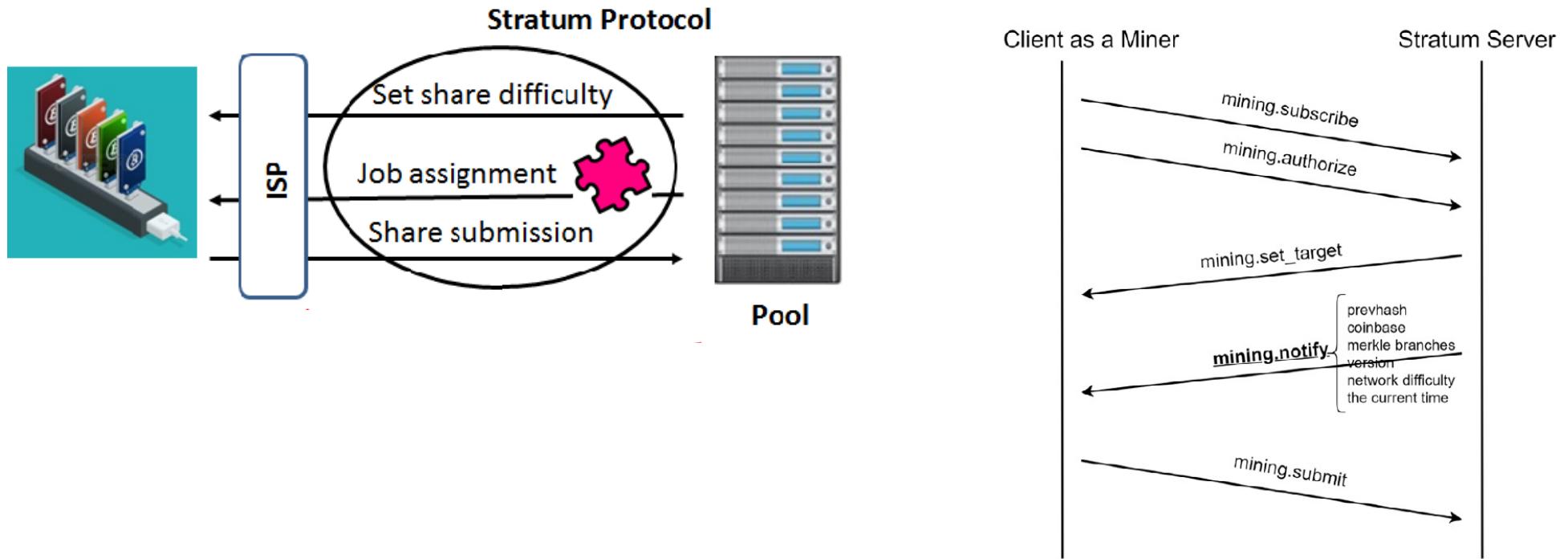
::: Pooled Mining (2/3)

La rete Bitcoin non è perfettamente decentralizzata, vista l'esistenza dei pools, ma finché la distribuzione del potere di hashing è ragionevole e i pool individuali rimangono ben al di sotto del 51% della rete, non si ha una vera situazione di squilibrio.

- **Pool di mining proporzionali:** si consente ai miner di contribuire alla potenza di elaborazione del pool e ricevere quote finché il pool non riesce a trovare un blocco. In questo caso, i miner ricevono ricompense proporzionali al numero di quote bloccate.
- **Pool di mining pay-per-share:** ogni miner riceve quote istantanee per il suo contributo indipendentemente da quando viene trovato il blocco. In questo tipo di pool, un miner può scambiare quote per un pagamento istantaneo.
- **Pool di mining peer-to-peer:** non si ha una centralizzazione della struttura del pool, riducendo il rischio di un singolo fallimento centrale del pool di mining.

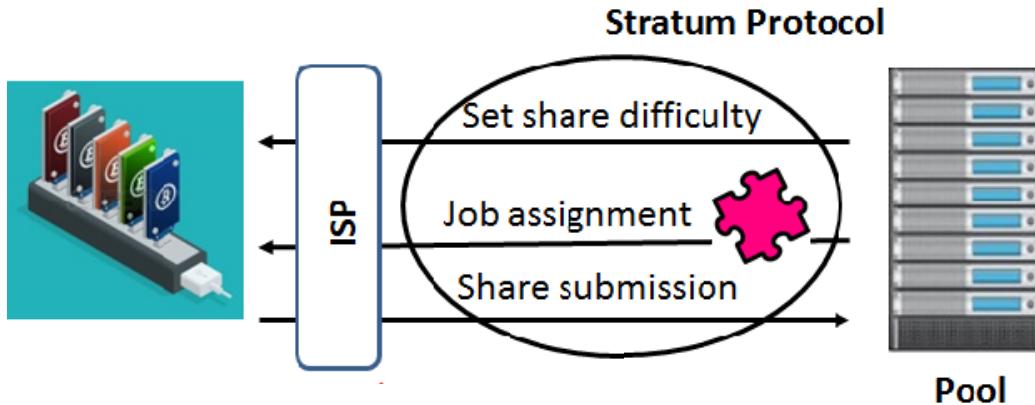
::: Pooled Mining (3/3)

Il protocollo attualmente in uso per la gestione del pooled mining in Bitcoin è Stratum V2, che utilizza una semplice socket TCP, con payload codificato come messaggi JSON-RPC. Il client apre semplicemente una socket TCP e scrive richieste al server sotto forma di messaggi JSON terminati dal carattere di nuova riga \n. Ogni riga ricevuta dal client è di nuovo un frammento JSON-RPC valido, contenente la risposta.



::: Pooled Mining (3/3)

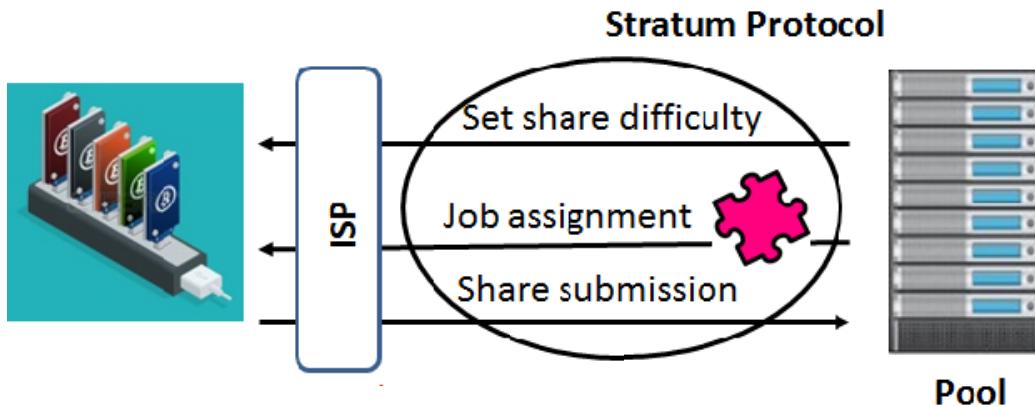
Il protocollo attualmente in uso per la gestione del pooled mining in Bitcoin è Stratum V2, che utilizza una semplice socket TCP, con payload codificato come messaggi JSON-RPC. Il client apre semplicemente una socket TCP e scrive richieste al server sotto forma di messaggi JSON terminati dal carattere di nuova riga \n. Ogni riga ricevuta dal client è di nuovo un frammento JSON-RPC valido, contenente la risposta.



- Con Stratum è possibile controllare la comunicazione in modo più efficiente di HTTP, scambiando i ruoli tra client e server.

::: Pooled Mining (3/3)

Il protocollo attualmente in uso per la gestione del pooled mining in Bitcoin è Stratum V2, che utilizza una semplice socket TCP, con payload codificato come messaggi JSON-RPC. Il client apre semplicemente una socket TCP e scrive richieste al server sotto forma di messaggi JSON terminati dal carattere di nuova riga \n. Ogni riga ricevuta dal client è di nuovo un frammento JSON-RPC valido, contenente la risposta.

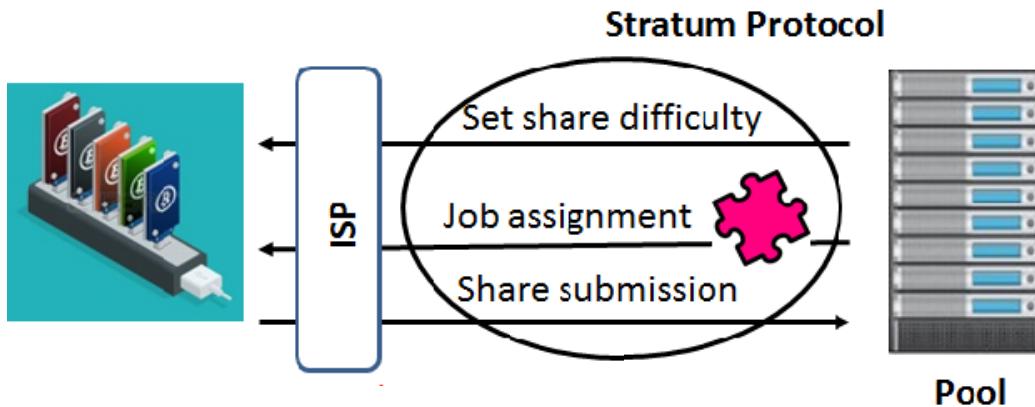


- Due elementi nell'header cambiano tra i tentativi di hash: NTime (cambia ogni sec.), Nonce (cambia ad ogni hash).

Per costruire un'intestazione valida, un minatore deve avere il valore corrente di questi due elementi. Sono necessari anche gli altri elementi dell'intestazione del blocco, ma le modifiche si verificano molto meno frequentemente e sono banali da gestire per il pool.

::: Pooled Mining (3/3)

Il protocollo attualmente in uso per la gestione del pooled mining in Bitcoin è Stratum V2, che utilizza una semplice socket TCP, con payload codificato come messaggi JSON-RPC. Il client apre semplicemente una socket TCP e scrive richieste al server sotto forma di messaggi JSON terminati dal carattere di nuova riga \n. Ogni riga ricevuta dal client è di nuovo un frammento JSON-RPC valido, contenente la risposta.

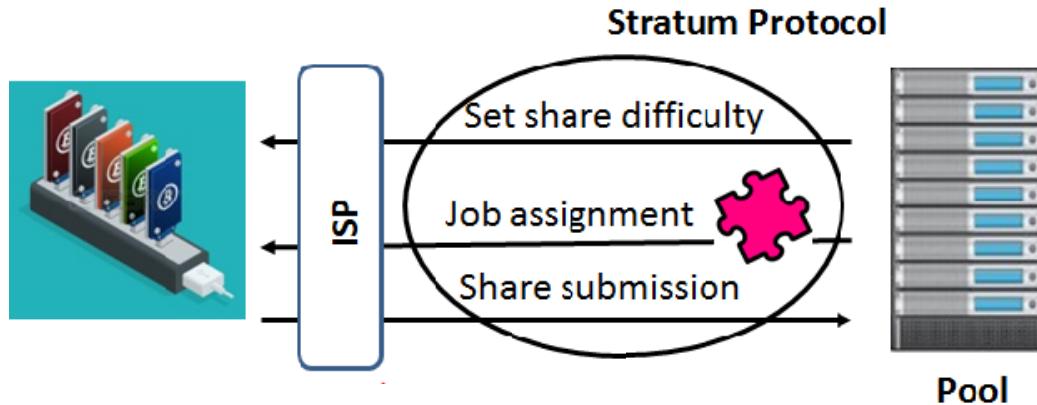


- Il metodo (inefficiente) per gestire queste due variabili è che il pool fornisca entrambi i valori in ogni singola richiesta: un getwork per ogni secondo.

Tuttavia, un minatore con meno di 4,2GH/s troverà in media <1 nonce per getwork. Il numero medio di getworks per azione dipenderà dalla velocità dei singoli minatori; tuttavia, si sta verificando un notevole eccesso di comunicazione.

::: Pooled Mining (3/3)

Il protocollo attualmente in uso per la gestione del pooled mining in Bitcoin è Stratum V2, che utilizza una semplice socket TCP, con payload codificato come messaggi JSON-RPC. Il client apre semplicemente una socket TCP e scrive richieste al server sotto forma di messaggi JSON terminati dal carattere di nuova riga \n. Ogni riga ricevuta dal client è di nuovo un frammento JSON-RPC valido, contenente la risposta.

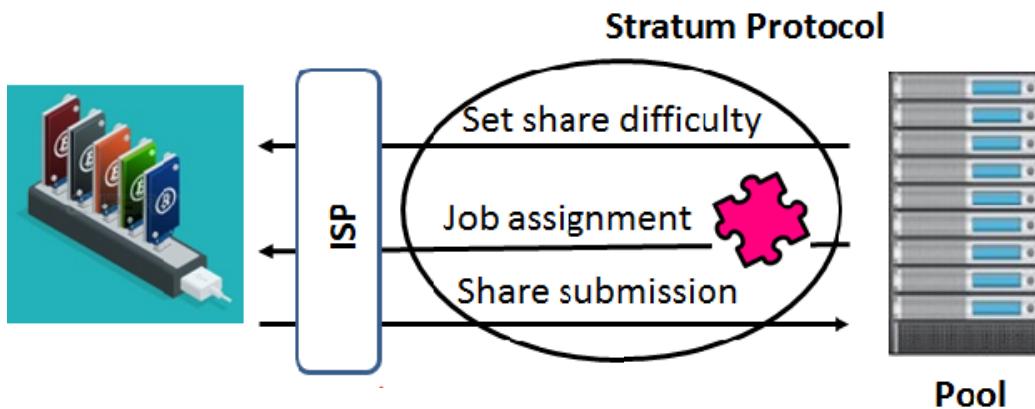


- Sia nonce che NTime hanno modifiche prevedibili, quindi a un miner potrebbe semplicemente essere assegnato un punto di partenza e modificare localmente i valori.

Si riduce significativamente il numero di richieste necessarie. Il valore di rotazione NTime indica al miner di quanto può incrementare NTime (in secondi) prima di dover richiedere nuovo lavoro.

::: Pooled Mining (3/3)

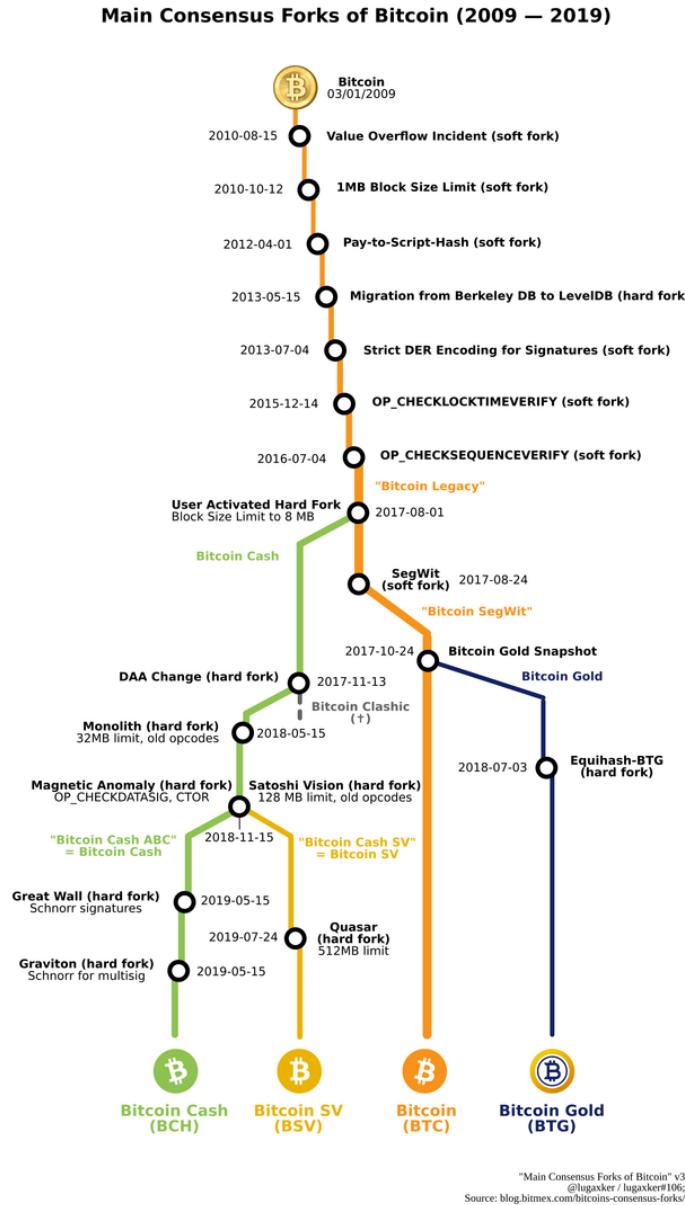
Il protocollo attualmente in uso per la gestione del pooled mining in Bitcoin è Stratum V2, che utilizza una semplice socket TCP, con payload codificato come messaggi JSON-RPC. Il client apre semplicemente una socket TCP e scrive richieste al server sotto forma di messaggi JSON terminati dal carattere di nuova riga \n. Ogni riga ricevuta dal client è di nuovo un frammento JSON-RPC valido, contenente la risposta.



- Stratum evita il Long-Polling di HTTP: si imposta un timeout molto elevato (ad es. 10 sec.) in cui il server ritarda di proposito l'invio delle informazioni finché non ha qualcosa da inviare.

I miner richiedono dati che non ci sono e i server devono mantenere tali connessioni per tutto il tempo in cui avviene l'estrazione.

::: Scalability nel Layer 1 di Bitcoin (1/6)



Il dibattito sulla risoluzione del problema di scalabilità di Bitcoin è arrivato al culmine nel 2017, con una divisione nella community di sviluppo di Bitcoin circa l'implementazione di una caratteristica chiamata SegWit con una soft fork, che esiste ancora oggi, e la mancata adozione generando il branch Bitcoin Cash.

Il Bitcoin Cash aveva un limite di dimensioni di blocco di 8 MB nel 2017, consentendo di elaborare le transazioni molto più velocemente. Questo limite è aumentato da maggio 2018 a 32 MB e potrebbe aumentare ulteriormente sulla base della capacità della rete e velocità di elaborazione delle PoW.

::: Scalability nel Layer 1 di Bitcoin (2/6)

 bitcoin
Block Size: [SegWit] 1MB Maximum, with up to 1.7MB possible for 100% Segwit usage.
Signatures: [SegWit] Transaction signatures are not required to be included in transactions.
Centralized development team and client implementation: Bitcoin Core.
Scaling plan: Limited blocksize settlement layer, off-chain payment channels, Lightning Network hubs that require opening a channel before spending.
Mining Difficulty: Adjusted every 2016 Blocks, vulnerable to hash power drops.

 Bitcoin Cash
Blocksize: [PowerBlocks] 8MB Maximum, with the ability to increase in the future.
Signatures: [SecureSigs] All transaction signatures must be validated and secured on the blockchain.
Multiple independent development teams and client implementations: Bitcoin Unlimited, Bitcoin ABC, Bitcoin XT, and Bitcoin Classic.
Scaling Plan: Large blocksize on-chain digital cash transactions, zero confirmation transaction applications, possibility for off-chain payment channels.
Mining Difficulty: Emergency Difficulty Adjustment. Protects the blockchain from hash power drops.

Bitcoin Cash ha implementato un Emergency Difficulty Adjustment (EDA) per garantire che i blocchi vengano estratti in modo più coerente, fornendo tempi di elaborazione delle transazioni più stabili.

Replace-by-fee (RBF) consente agli utenti di sostituire una transazione non confermata «bloccata» con una nuova che include una commissione più elevata, garantendo un'elaborazione più rapida.

RBF potrebbe facilitare una doppia spesa: un utente malintenzionato potrebbe inizialmente inviare una transazione a basso costo e poi sostituirla con una transazione con una commissione più elevata a un portafoglio che controlla.

::: Scalability nel Layer 1 di Bitcoin (3/6)

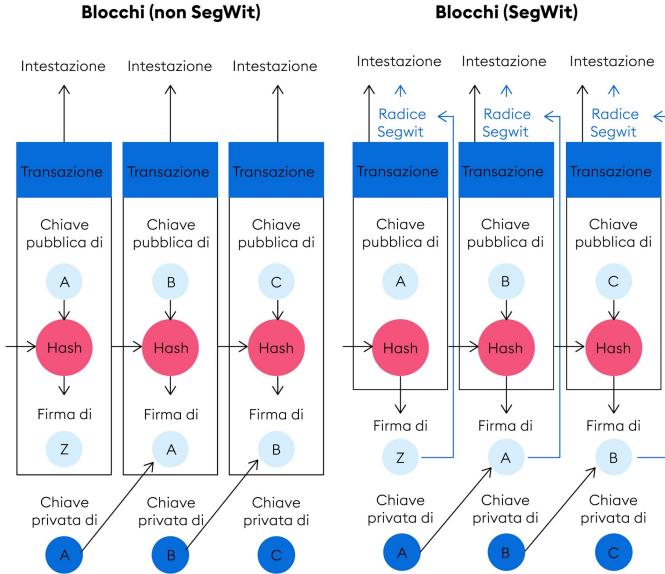
La maggior parte delle implementazioni RBF richiede che la transazione includa gli stessi output. Inoltre, l'attesa di alcune conferme di rete rende RBF inefficace una volta confermata la transazione.

Bitcoin Cash ha rimosso RBF, rendendo irreversibili le transazioni non confermate. La maggiore dimensione dei blocchi e i tempi di transazione più rapidi di BCH riducono la probabilità di doppia spesa.

```
JS blockchainSync.js
examples > JS blockchainSync.js > ...
7   const kth = require("@knuth/bch");
8
9   let running_ = false;
10
11  async function main() {
12    process.on('SIGINT', shutdown);
13    const config = kth.settings.getDefault(kth.network.mainnet);
14    const node = new kth.node.Node(config, false);
15    await node.launch(kth.startModules.all);
16    console.log("Knuth node has been launched.");
17    running_ = true;
18
19    const height = await node.chain.getLastHeight();
20    console.log(`Current height in local copy: ${height}`);
21
22    if (await comeBackAfterTheBCHHardFork(node)) {
23      console.log("Bitcoin Cash has been created!");
24    }
25
26    node.close();
27    console.log("Good bye!");
28  }
29
30  async function comeBackAfterTheBCHHardFork(node) {
31    const hfHeight = 478559;
32    while (running_) {
33      const height = await node.chain.getLastHeight();
34      if (height >= hfHeight) return true;
35      await sleep(10000);
36    }
37    return false;
38 }
```

CashScript [TypeScript](https://cashscript.org/) [SDK](https://cashscript.org/) (<https://cashscript.org/>) semplifica la creazione di transazioni di smart contract,. Offrendo la completa safety dei tipi, gli sviluppatori possono essere certi della qualità e dell'affidabilità delle loro applicazioni. Ha una sintassi familiare, infatti si basa sul linguaggio di smart contract di Ethereum denominato Solidity.

::: Scalability nel Layer 1 di Bitcoin (4/6)



SegWit (Segregated Witness) riduce il peso delle transazioni in un blocco definendo due sezioni.

- La prima parte di una transazione contiene gli indirizzi del mittente e del destinatario;
- La seconda parte contiene i "dati del testimone", ovvero le firme della transazione.

SegWit riduce notevolmente la dimensione delle transazioni, così da metterne di più in un blocco.

SegWit ha permesso di correggere una falla che permetteva agli utenti di cambiare gli hash delle transazioni, che rappresentano anche l'identificativo della transazione stessa (TXID).

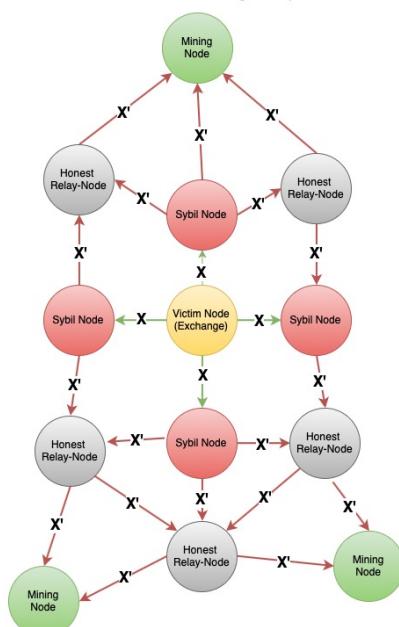
::: Scalability nel Layer 1 di Bitcoin (5/6)

La malleabilità è la capacità di modificare l'identificativo di una transazione (TXID) ma senza invalidarla. Un modo comune per tale azione è attraverso la malleabilità della firma.

Considerando le firme digitali ECDSA, è possibile modificare una firma ma senza invalidarla. Ciò non consente la falsificazione di tali firme, ma di modificare il TXID di una transazione contenente tali firme.

Malleability-Attack

Attacker surrounds an exchange with sybil nodes and then makes a withdrawal X. Sybil nodes malleate X into X' which are eventually mined by honest network. Since exchange can't confirm X, it is tricked into re-issuing new withdrawals to attacker each time losing money.



Bob paghi ad Alice dei Bitcoin tramite una transazione con TXID pari a X. Prima di essere estratta, X venga malleato con un nuovo TXID pari a X'. Alice ha ricevuto il pagamento, ma Bob non lo sa. Alice inganna Bob per riemettere il pagamento e lo ripete finché Bob non si rende conto di cosa sta succedendo, ma potrebbe essere troppo tardi.

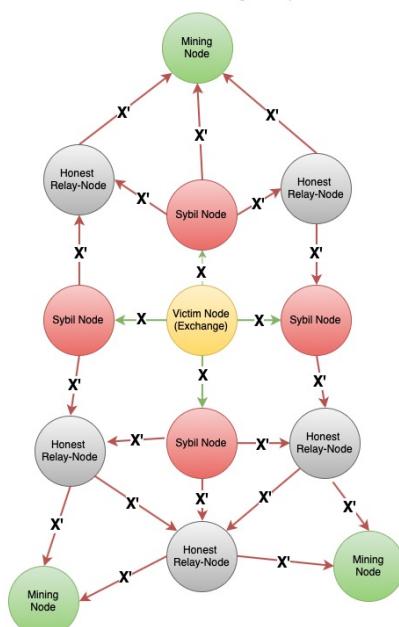
::: Scalability nel Layer 1 di Bitcoin (5/6)

La malleabilità è la capacità di modificare l'identificativo di una transazione (TXID) ma senza invalidarla. Un modo comune per tale azione è attraverso la malleabilità della firma.

Considerando le firme digitali ECDSA, è possibile modificare una firma ma senza invalidarla. Ciò non consente la falsificazione di tali firme, ma di modificare il TXID di una transazione contenente tali firme.

Malleability-Attack

Attacker surrounds an exchange with sybil nodes and then makes a withdrawal X. Sybil nodes malleate X into X' which are eventually mined by honest network. Since exchange can't confirm X, it is tricked into re-issuing new withdrawals to attacker each time losing money.



Un aggressore:

1. Avvia un gruppo di nodi Sybil (rossi)
2. Circonda un nodo di scambio (giallo) con i suoi nodi Sybil.
3. Esegue un prelievo dallo scambio.
4. Non appena il prelievo X lascia lo scambio, i nodi Sybil malleano il prelievo in X'.
5. Il malleato X' viene propagato al resto della rete.

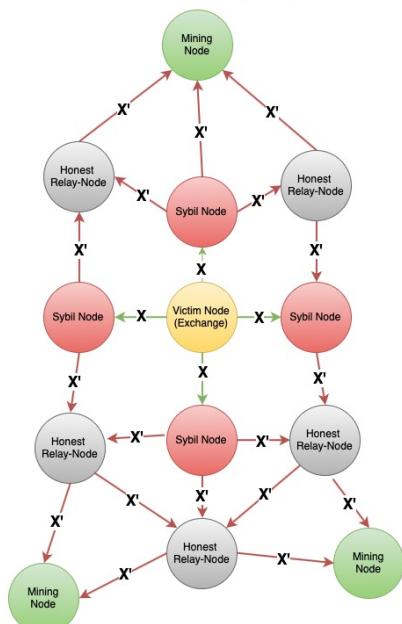
::: Scalability nel Layer 1 di Bitcoin (5/6)

La malleabilità è la capacità di modificare l'identificativo di una transazione (TXID) ma senza invalidarla. Un modo comune per tale azione è attraverso la malleabilità della firma.

Considerando le firme digitali ECDSA, è possibile modificare una firma ma senza invalidarla. Ciò non consente la falsificazione di tali firme, ma di modificare il TXID di una transazione contenente tali firme.

Malleability-Attack

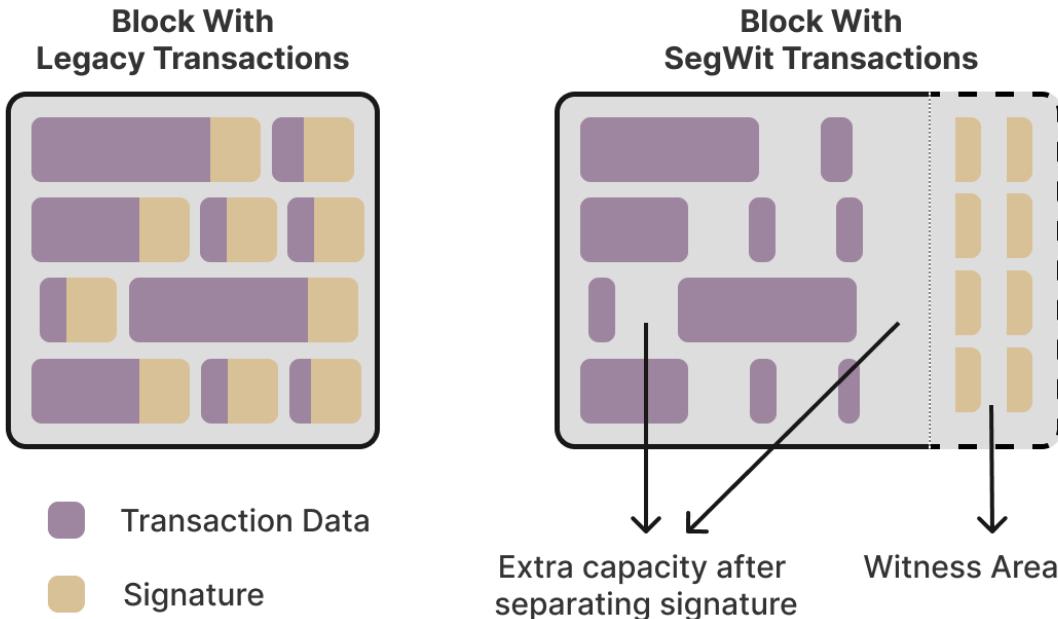
Attacker surrounds an exchange with sybil nodes and then makes a withdrawal X. Sybil nodes malleate X into X' which are eventually mined by honest network. Since exchange can't confirm X, it is tricked into re-issuing new withdrawals to attacker each time losing money.



6. Quando X' viene incluso in un blocco, l'aggressore ha ricevuto il prelievo X' ma il nodo vittima non trova X nella blockchain.
7. L'aggressore chiede di ripetere il prelievo, poiché "non è andato a buon fine".
8. Si ripete l'attacco.

Si può usare una "strategia parassitaria", con piccoli e rari prelievi, oppure una "strategia vampiro" con prelievi di grandi dimensioni ad alta frequenza.

::: Scalability nel Layer 1 di Bitcoin (6/6)



Bitcoin (BTC) ha "risolto" la malleabilità con l'introduzione di Segwit che separa la firma dal calcolo TXID e la sostituisce con un non-malleable hash commitment. Questo hash funge da puntatore alla firma che è memorizzata in un'altra struttura dati.

Per verificare la firma della transazione, il verificatore utilizza l'hash per cercare la firma nell'altra struttura dati, quindi esegue la consueta verifica ECDSA. Ciò elimina le firme come fonte di malleabilità dalla transazione poiché sono memorizzate all'esterno della transazione e il puntatore hash alla firma non può essere malleato.

::: Layer 2 di Bitcoin (1/2)

Nonostante il suo successo, Bitcoin deve risolvere alcune sfide, in particolare i problemi di scalabilità, per cui sono state introdotte le reti Layer 2 di Bitcoin, una classe di protocolli progettati per migliorare la scalabilità, ridurre i costi di transazione e sbloccare nuove possibilità per l'ecosistema Bitcoin.

Sono state costruite sulla blockchain di Bitcoin per elaborare le transazioni al di fuori della blockchain principale, riducendo il carico sul Layer 1, offrendo vantaggi come una migliore scalabilità, una maggiore programmabilità e capacità ampliate per supportare varie applicazioni decentralizzate.

- Il Lightning Network usa i canali di stato per l'invio e la ricezione di pagamenti. Le transazioni all'interno di questi canali avvengono off-chain, e solo i saldi di apertura e chiusura vengono comunicati alla rete principale, riducendo la congestione e migliorando l'efficienza.

::: Layer 2 di Bitcoin (2/2)

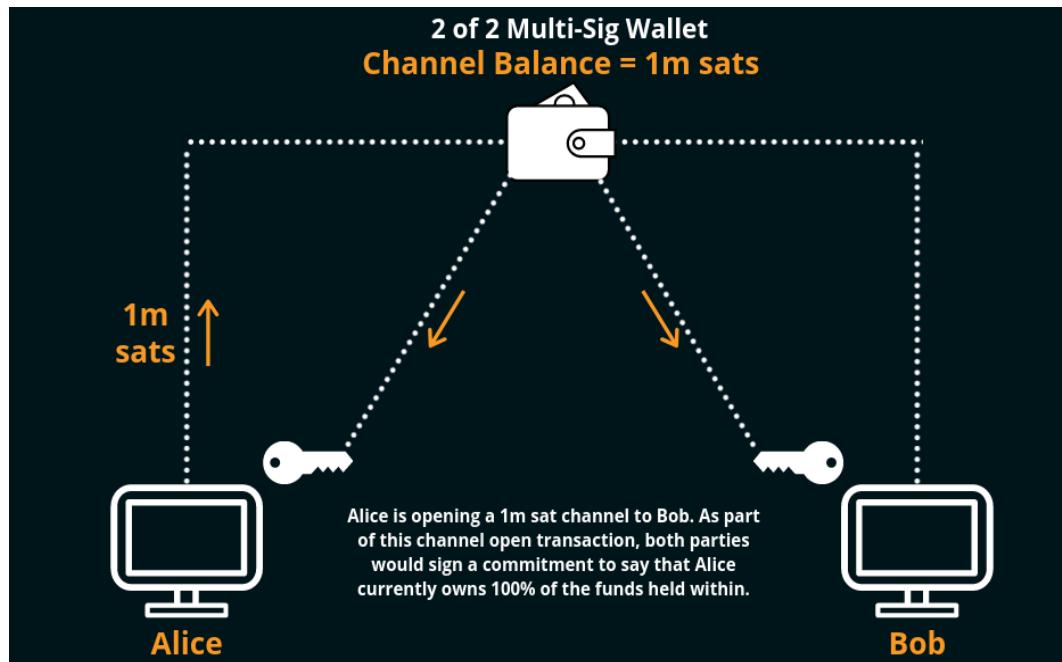
- Operando come sidechain, Rootstock consente agli utenti di inviare Bitcoin alla rete Rootstock, dove diventa uno smart Bitcoin (RBTC) bloccato nel wallet RSK dell'utente, portando a transazioni più rapide ed economiche.
- Stacks Protocol abilita gli smart contract e le applicazioni decentralizzate sulla blockchain di Bitcoin. Stacks utilizza microblocchi per la velocità e un meccanismo Proof-of-Transfer (PoX), legando le transazioni alla blockchain di Bitcoin.
- Liquid Network è una sidechain Layer 2 di Bitcoin che consente ai suoi utenti di trasferire i propri Bitcoin da e verso Bitcoin utilizzando un meccanismo di peg a due vie. Quando BTC viene trasferito su Liquid Network, viene convertito in Liquid BTC (L-BTC) in rapporto 1:1. Supporta anche l'emissione di token e altri asset digitali.

::: Lightning Network (1/6)

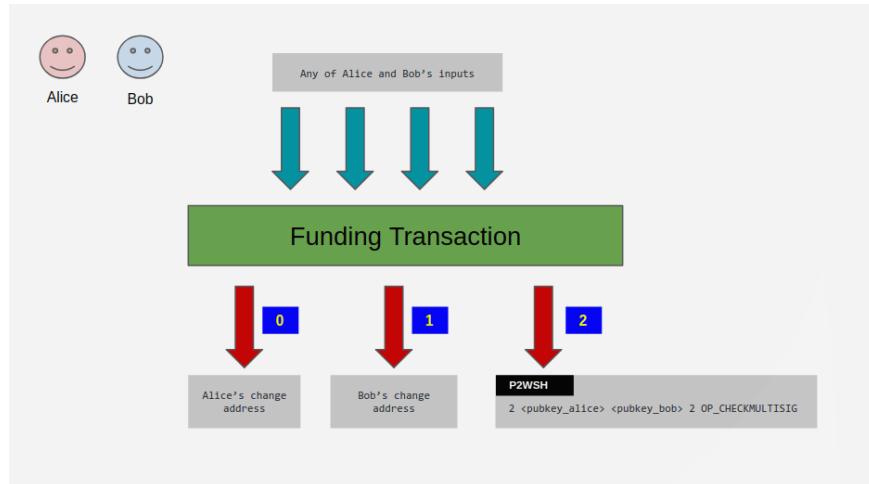
Il Lightning Network è costituito da migliaia di canali di pagamento bipartitici, che consentono a due utenti di scambiarsi reciprocamente criptovalute tutte le volte che desiderano, quasi istantaneamente e senza commissioni blockchain.

Un canale viene aperto da uno o due utenti che bloccano una quantità

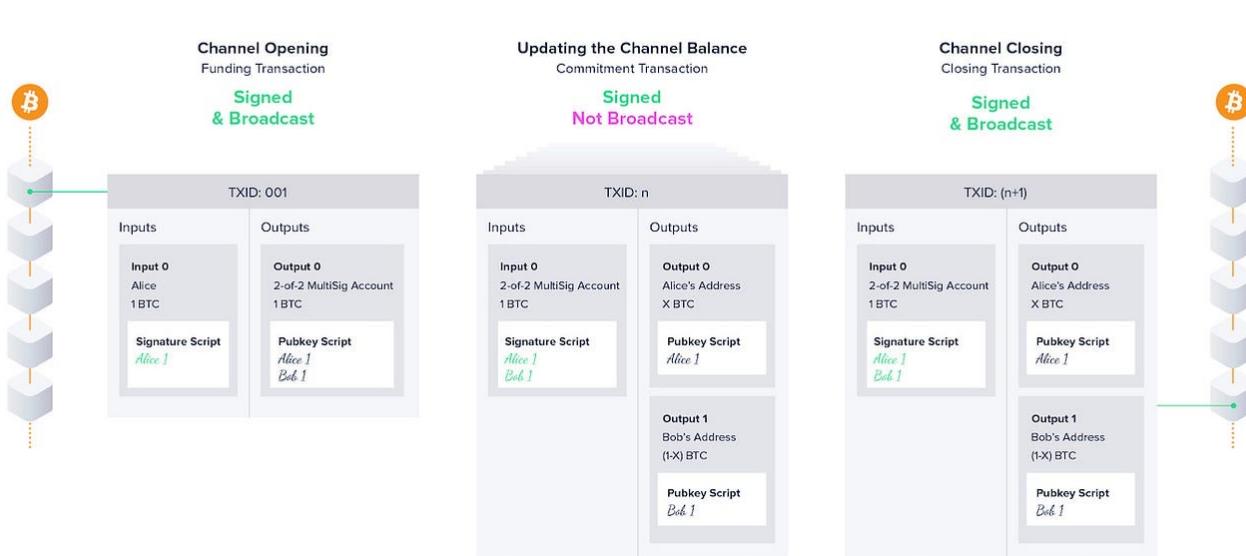
di sat in una "transazione di finanziamento" on-chain che crea un portafoglio multi-firma sulla rete Bitcoin, con ogni utente che riceve una delle chiavi. Lo "stato" del canale di apertura rifletterà l'importo che un utente contribuisce e ogni parte firmerà per dire che accetta che sia corretto.



::: Lightning Network (2/6)

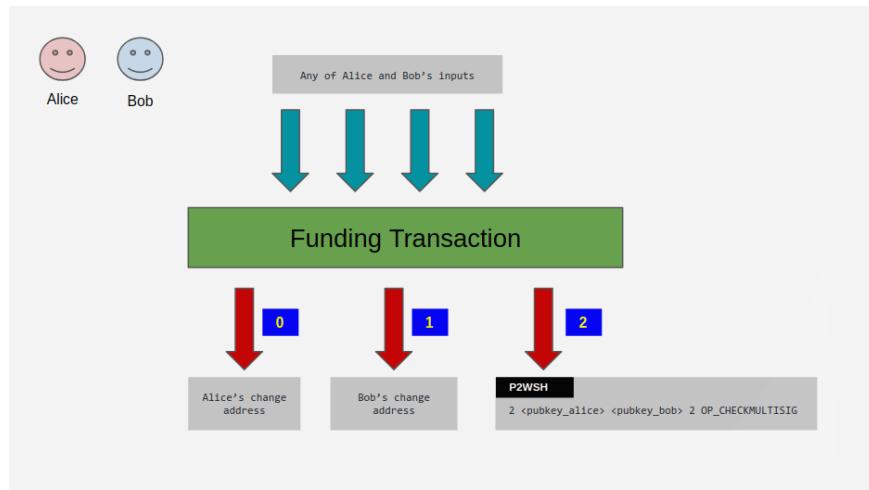


Gli input sono solitamente forniti da Alice e/o Bob. L'output 0 e 1 rappresentano le potenziali modifiche che ritornano agli indirizzi controllati da Alice o Bob. L'output 2 contiene uno script multisig P2WSH che blocca il fondo totale per la durata di questo canale di pagamento tra Alice e Bob.

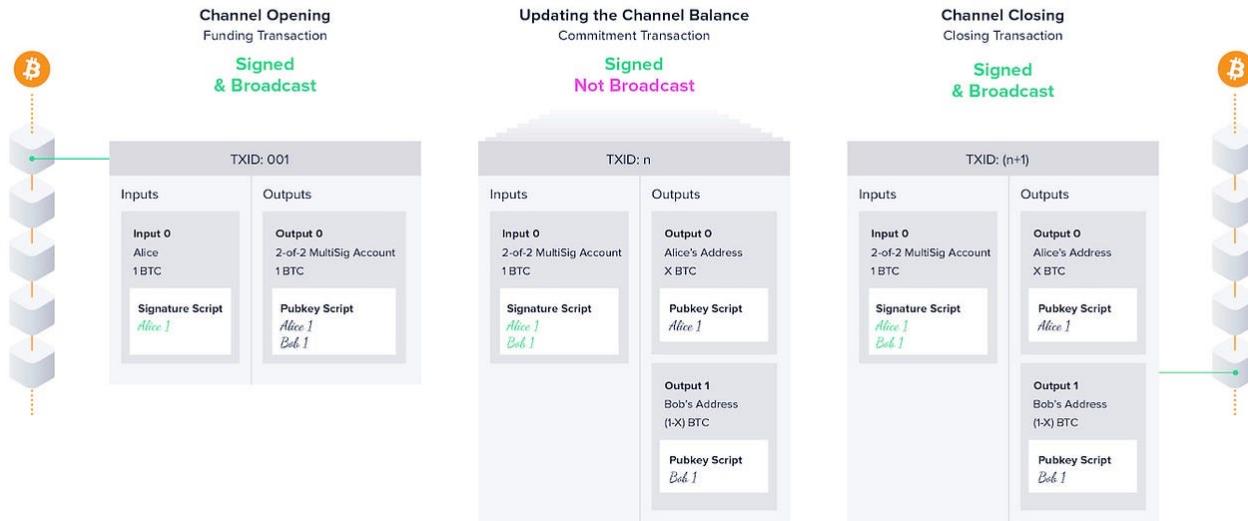


Durante l'uso del canale, le transazioni sono generate ma non trasmesse a tutti i nodi ma solo tra i nodi del canale.

::: Lightning Network (2/6)



Gli input sono solitamente forniti da Alice e/o Bob. L'output 0 e 1 rappresentano le potenziali modifiche che ritornano agli indirizzi controllati da Alice o Bob. L'output 2 contiene uno script multisig P2WSH che blocca il fondo totale per la durata di questo canale di pagamento tra Alice e Bob.



Alla chiusura viene creata una transazione che è distribuita con lo stato finale dei partecipanti al canale.

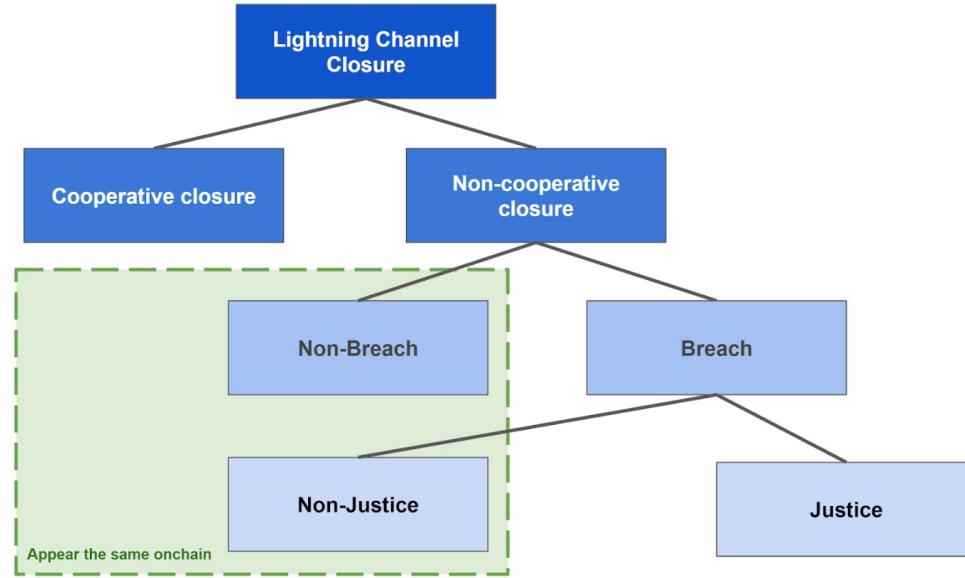
::: Lightning Network (3/6)

- La chiusura collaborativa avviene quando (1) una parte avvia una chiusura del canale comunicandolo al suo partner, (2) entrambe le parti concordano di chiudere il canale e (3) lo stato più recente viene trasmesso alla rete e ogni partecipante riceve i propri sat sul proprio portafoglio on-chain.
- La chiusura forzata si ha quando una parte chiude il canale senza il consenso della controparte, trasmettendo semplicemente lo stato del canale più recente a lui noto. Il richiedente avrà il suo saldo bloccato per un determinato periodo di tempo per consentire alla controparte di vedere la chiusura del canale e di tornare online per confermare lo stato del canale. Se ciò non accade, la parte che ha chiuso i fondi on-chain del canale diventerà spendibile dopo la fine del periodo di blocco (solitamente 2016 blocchi o due settimane).

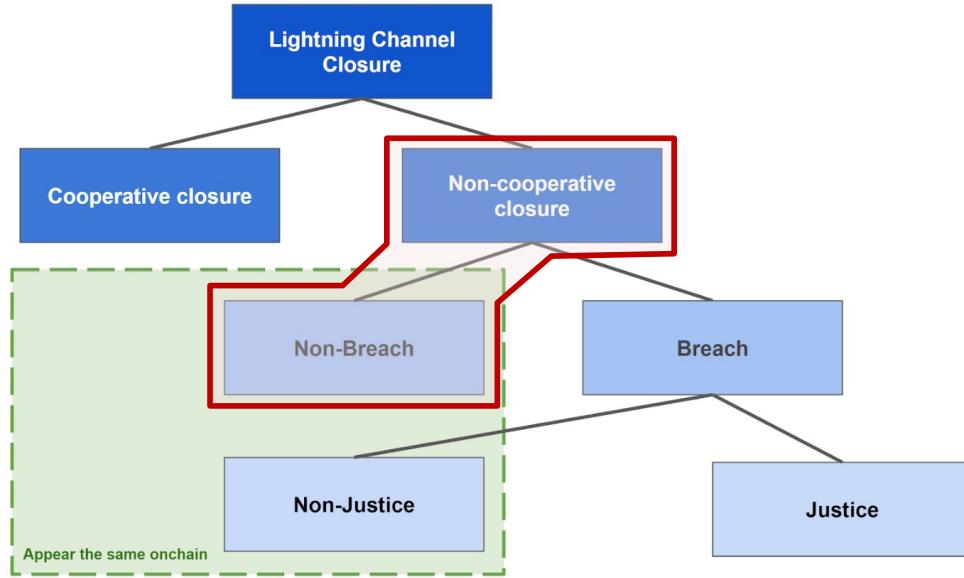
::: Lightning Network (4/6)

- Una chiusura contestata deriva dall'avvio di una chiusura forzata e tra le parti non c'è accordo sull'esito di chiusura. Per innescare questa controversia, le parti riportano semplicemente online entro il periodo di blocco o impiegano un servizio Watchtower che monitorerà i loro canali e agirà per loro conto per una piccola commissione.
- Se la parte che crea la controversia riesce a pubblicare con successo uno stato del canale più recente di quello trasmesso dal proprio partner di canale, il suo nodo sarà in grado di pubblicare una transazione Justice e reclamare l'intero saldo del canale. La minaccia di uno scenario del genere è sufficiente a tenere lontani la maggior parte degli operatori Lightning disonesti per paura di perdere tutti i loro fondi.

::: Lightning Network (5/6)

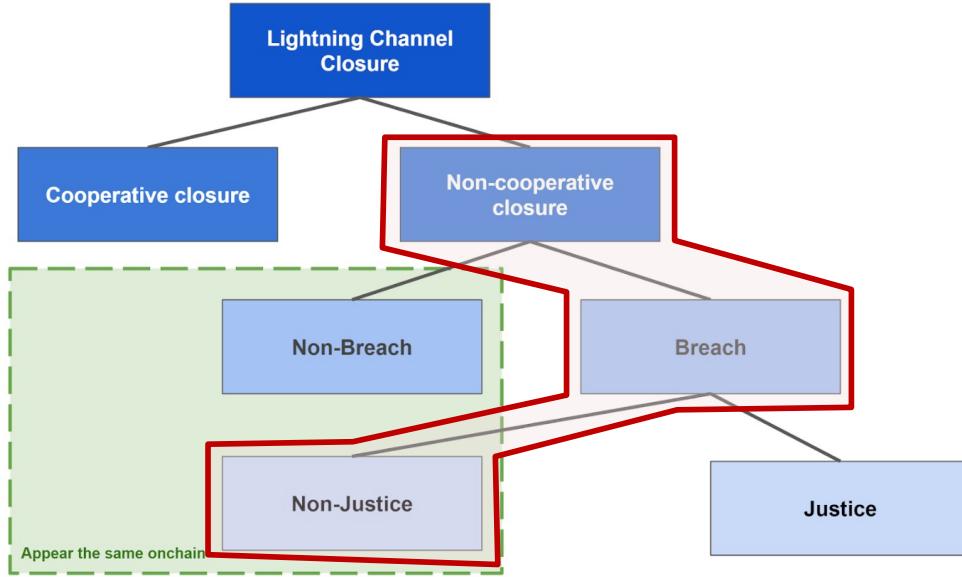


::: Lightning Network (5/6)



Una chiusura non cooperativa senza violazioni si verifica quando un nodo onesto avvia la chiusura, senza comunicare direttamente con la controparte del canale. I fondi vengono distribuiti onchain in base all'ultimo stato del canale.

::: Lightning Network (5/6)

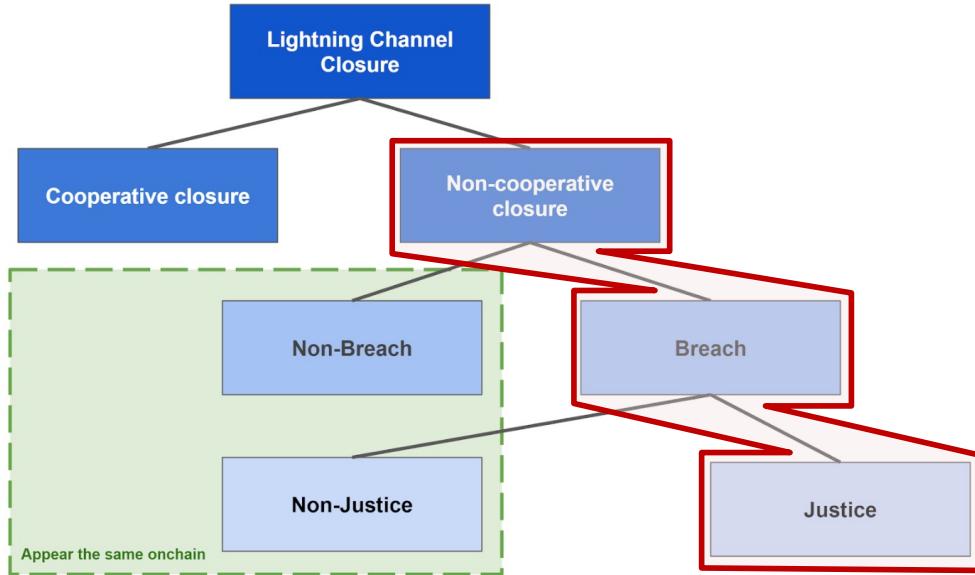


Una chiusura non cooperativa con breach non-justice si verifica quando un nodo disonesto avvia la chiusura del canale, trasmettendo uno stato del canale precedente, tentando di rubare fondi dal nodo dall'altra parte del canale.

Il nodo onesto non controlla la rete entro il periodo di locktime, normalmente 24 ore, e non trasmette una transazione Justice. Pertanto il furto ha successo.

I fondi vengono distribuiti al portafoglio di ciascuna parte in base a uno stato del canale precedente, in modo tale che la parte non-closing perda fondi e la parte disonesta che chiude il canale ruba con successo i fondi.

::: Lightning Network (5/6)

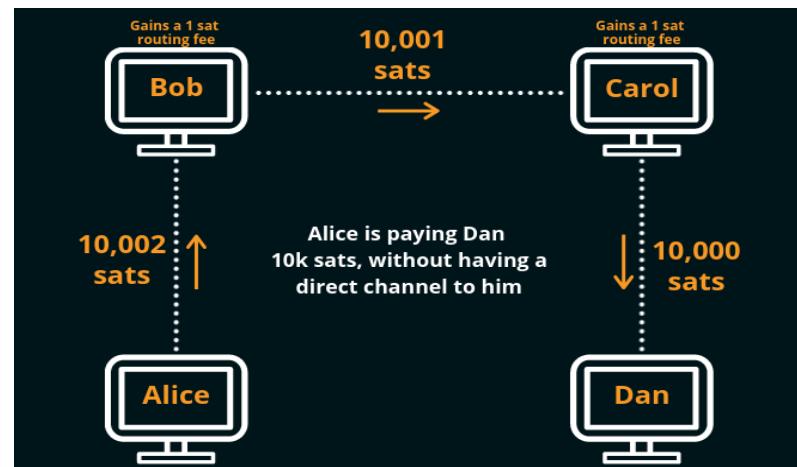


Una chiusura giudiziaria per violazione non cooperativa si verifica quando un nodo disonesto avvia la chiusura del canale, senza comunicare direttamente con il nodo dall'altra parte del canale.

Il nodo non in chiusura controlla la rete entro il periodo di locktime e crea una transazione giudiziaria, in modo che il tentativo di furto fallisca. Il potenziale ladro viene punito e tutti i fondi vanno alla parte onesta non in chiusura.

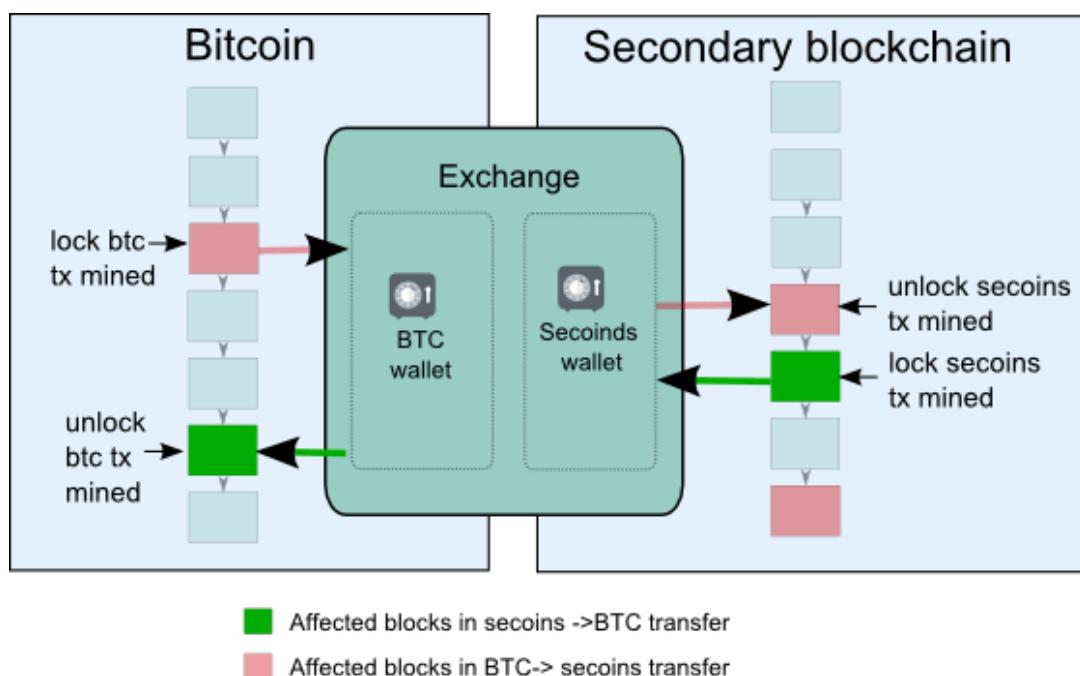
::: Lightning Network (6/6)

Più canali di pagamento a coppie potrebbero essere combinati per formare una rete di canali. Ciò consente a una parte di effettuare transazioni con tutte le altre parti nella rete senza stabilire un canale a coppie per ciascuna di esse. Utilizzando il routing delle transazioni multi-hop all'interno di questa rete, le singole transazioni possono avvenire completamente off-chain. Solo la transazione finale che regola il pagamento per un dato canale o set di canali viene inviata alla blockchain.



::: Rootstock (1/3)

La piattaforma RSK è un protocollo basato sul linguaggio Solidity che funziona come una catena laterale in esecuzione in parallelo alla blockchain di Bitcoin.



Quando i BTC vengono scambiati per RBTC, nessuna valuta viene "trasferita". Quando avviene un trasferimento, alcuni BTC vengono bloccati in Bitcoin e la stessa quantità di RBTC viene sbloccata in RSK.

Quando RBTC deve essere riconvertito in BTC, RBTC viene nuovamente bloccato in RSK e la stessa quantità di BTC viene sbloccata in Bitcoin.

::: Rootstock (2/3)

Gli Smart-contract vengono eseguiti da tutti i nodi completi della rete, mediante la macchina virtuale RSK (RVM). Il risultato dell'esecuzione può essere l'elaborazione di messaggi inter-contratto, la creazione di transazioni monetarie e la modifica dello stato della memoria persistente del contratto.

L'RVM è compatibile con EVM a livello di codice operativo, consentendo ai contratti Ethereum di funzionare senza problemi su RSK.



Questa piattaforma realizza quello che è noto come merged mining, ovvero il mining di due o più criptovalute contemporaneamente, senza sacrificare le prestazioni complessive, se impiegano la stessa funzione di hash.

::: Rootstock (3/3)

In sostanza, un miner può usare la sua potenza di calcolo per estrarre blocchi su più catene contemporaneamente tramite l'uso di ciò che è noto come Auxiliary Proof of Work (AuxPoW).

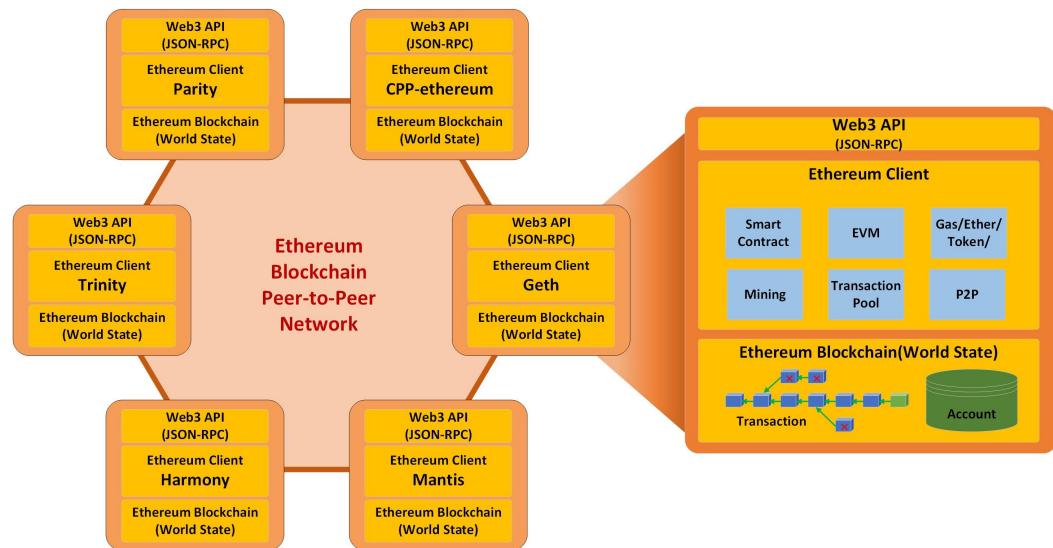
- Il lavoro svolto su una blockchain può essere sfruttato come lavoro valido su un'altra catena. La blockchain che fornisce la proof of work è chiamata blockchain madre, mentre quella che la accetta come valida è la blockchain ausiliaria.



Principali Piattaforme – Ethereum & Solidity

::: Ethereum (1/11)

Ethereum è la più grande piattaforma software decentralizzata ed open-source che consente lo sviluppo di smart contracts e applicazioni implementate al di sopra di una Blockchain permissionless.



- Una rete blockchain (originariamente con PoW e ora con PoS) e una criptovaluta chiamata Ether, usata per pagare gas, un'unità di calcolo utilizzata nelle transazioni e in altre transizioni di stato.

- La blockchain è caratterizzata da accounts, che interagiscono tra loro tramite transazioni (messaggi) e sono caratterizzati da uno stato e un indirizzo a 20 byte che li identifica.



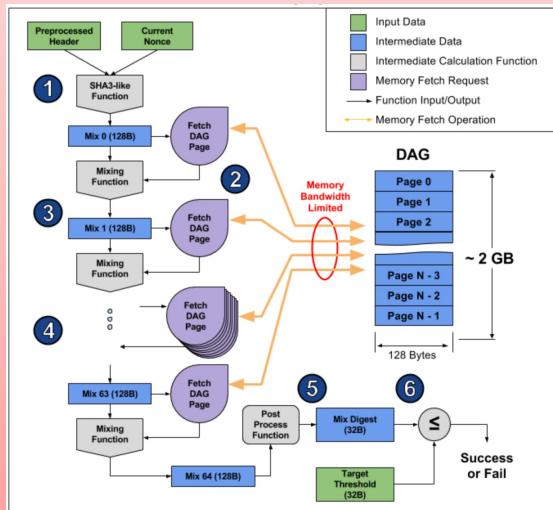
::: Ethereum (1/11)

Ethereum è la più grande piattaforma software decentralizzata ed open-source che consente lo sviluppo di smart contracts e applicazioni implementate al di sopra di una Blockchain permissionless.



- Una rete blockchain (originariamente con PoW e ora con PoS)

Impiega un algoritmo PoW chiamato Ehash che utilizza un algoritmo di hash appartenente alla famiglia Keccak, la stessa delle funzioni hash SHA-3. Utilizza un set di dati di grandi dimensioni che viene periodicamente rigenerato e cresce lentamente nel tempo.



Svolge delle letture intensive ad accesso casuale in memoria a 2 GB di dati strutturati come un grafo aciclico diretto (DAG). La risoluzione si realizza con 64 cicli a partire dall'hash di informazioni di partenza con pagine estratte dal DAG. Il digest viene confrontato con una soglia target: se inferiore o uguale, il calcolo è considerato riuscito, altrimenti, viene rieseguito con un nonce diverso (incrementando quello corrente o scegliendone uno nuovo a caso).

::: Ethereum (1/11)

Ethereum è la più grande piattaforma software decentralizzata ed

Ethereum gestisce due diversi tipi di account:

- Externally owned accounts (EOA), sono in grado di inviare e ricevere Ether, e inviare transazioni agli Smart Contract;
- Contract accounts, che oltre alle funzionalità degli EOAs hanno del codice associato che implementano delle azioni "triggerate" da EOA o altri Contract accounts.

L'esecuzione del codice modifica le informazioni contenute nel proprio spazio di archiviazione (consentendo di utilizzare la Blockchain per scopi diversi dalla criptovalute).



Harmony Mantis
Ethereum Blockchain (World State) Ethereum Blockchain (World State)

transizioni di stato.

- La blockchain è caratterizzata da accounts, che interagiscono tra loro tramite transazioni (messaggi) e sono caratterizzati da uno stato e un indirizzo a 20 byte che li identifica.

address

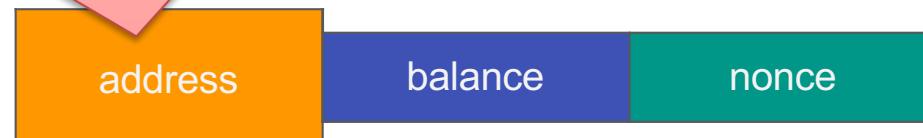
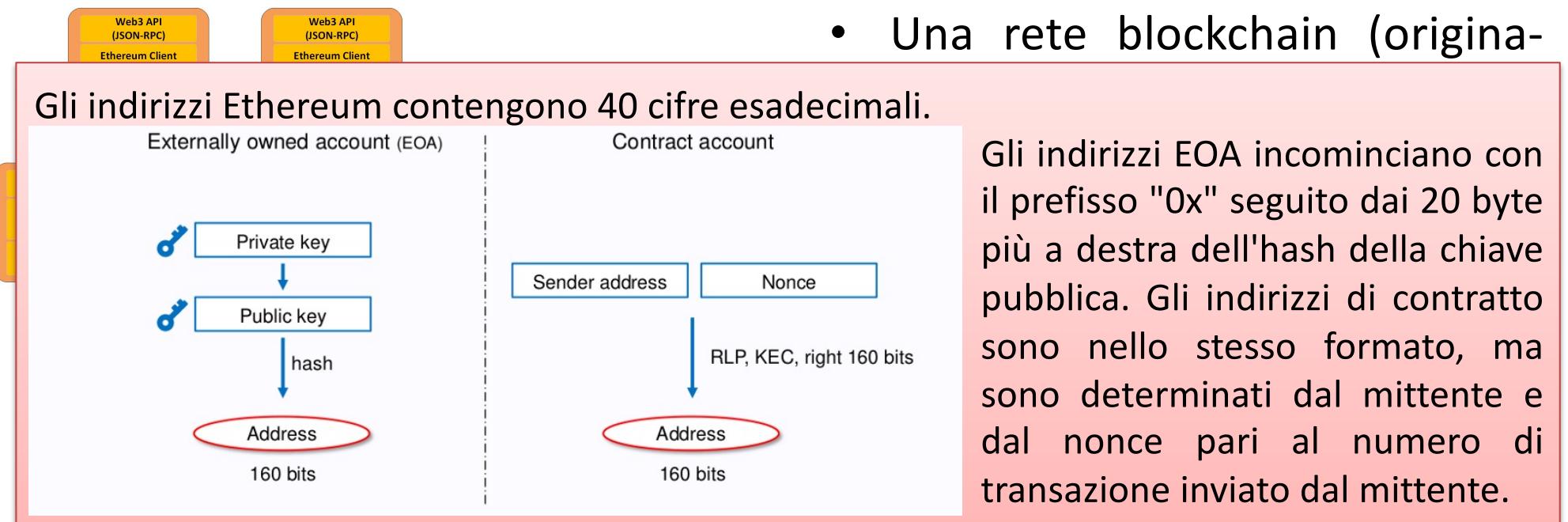
balance

nonce

::: Ethereum (1/11)

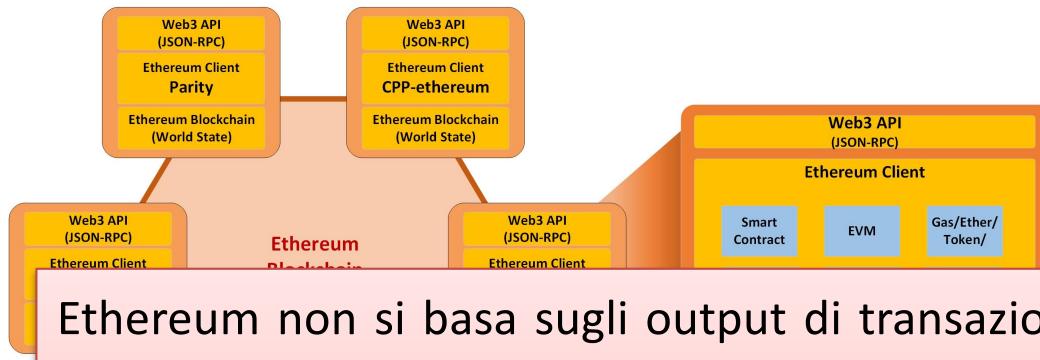
Ethereum è la più grande piattaforma software decentralizzata ed open-source che consente lo sviluppo di smart contracts e applicazioni implementate al di sopra di una Blockchain permissionless.

- Una rete blockchain (origina-



::: Ethereum (1/11)

Ethereum è la più grande piattaforma software decentralizzata ed open-source che consente lo sviluppo di smart contracts e applicazioni implementate al di sopra di una Blockchain permissionless.



- Una rete blockchain (originariamente con PoW e ora con PoS) e una criptovaluta chiamata Ether usata per

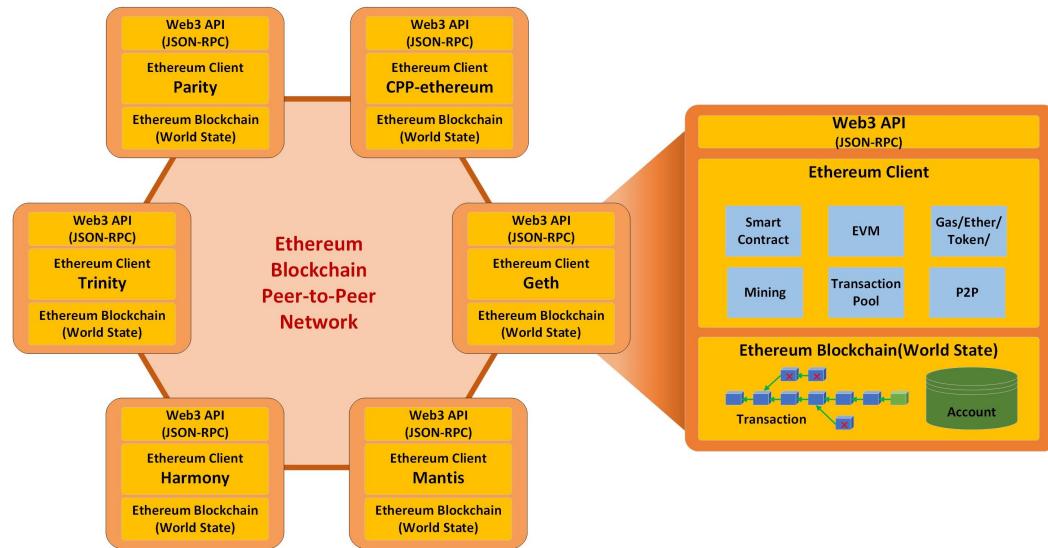
Ethereum non si basa sugli output di transazione non spesi (UTXO), ma gli account hanno uno stato che indica il saldo corrente. Lo stato non è memorizzato sulla blockchain, ma in un albero separato di Merkle Patricia. Un wallet di criptovaluta memorizza le chiavi pubbliche e private (un utente può avere più indirizzi), che possono essere utilizzati per ricevere o spendere ether. I wallet sono come applicazioni che consentono di interagire con un account Ethereum, analogamente alle app di e-banking e un conto bancario.

I loro transaction (messaggi) e sono caratterizzati da uno stato e un indirizzo a cui inviare i dati che li identifica.



::: Ethereum (1/11)

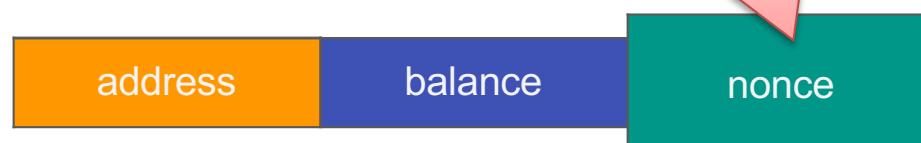
Ethereum è la più grande piattaforma software decentralizzata ed open-source che consente lo sviluppo di smart contracts e applicazioni implementate al di sopra di una Blockchain permissionless.



- Una rete blockchain (originariamente con PoW e ora con PoS) e una criptovaluta chiamata Ether, usata per pagare gas, un'unità di calcolo utilizzata nelle transazioni e in altre transizioni di stato.

- La blockchain è caratterizzata da accounts, che interagiscono tra loro tramite transazioni (messaggi) da uno stato e un indirizzo a 20 byte che li identifica.

Numero di transazioni totali.



::: Ethereum (2/11)

I contratti hanno generalmente 4 scopi:

- Gestire un "data store" che contiene informazioni utili per altri contratti o per il mondo esterno;
- Comportarsi come un EOA con politiche di accesso più avanzate (una sorta di filtro che consente l'inoltro di "messaggi" a determinate EOA solo se determinate condizioni vengono soddisfatte);
- Gestire un contratto o una relazione in corso tra più utenti (un esempio è un contratto che paga automaticamente chi invia una soluzione valida ad alcuni problemi matematici, o dimostra che sta fornendo una risorsa computazionale);
- Fornire funzionalità ad altri contratti (una sorta di libreria).

Il codice viene interpretato dalla Ethereum Virtual Machine (EVM), e rappresentato in un EVM bytecode.

::: Ethereum (3/11)

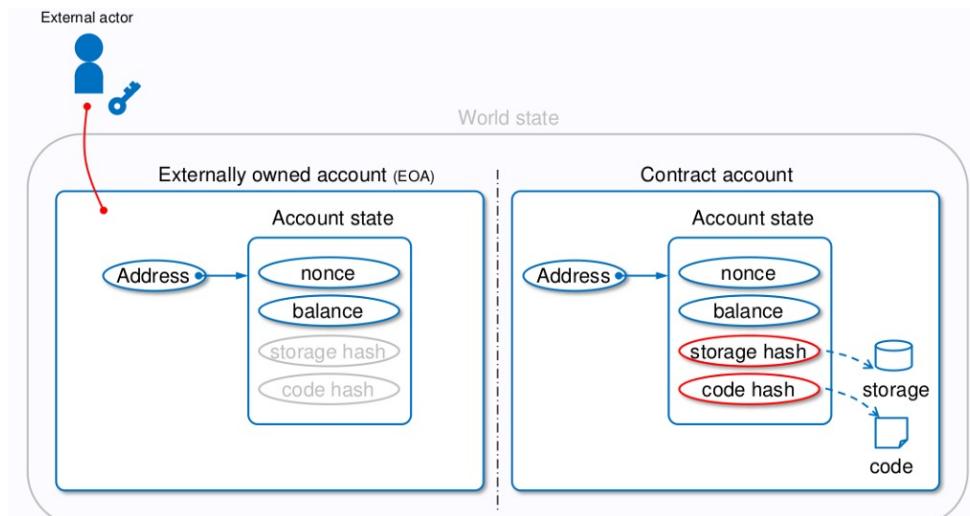
Gli account dei contratti sono diversi:

	Account Personale	Account di Contratto
address	$H(\text{pub_key})$	$H(\text{addr} + \text{nonce del creatore})$
code	\emptyset	Codice
storage	\emptyset	Dati
balance	Saldo ETH (in Wei)	Saldo ETH (in Wei)
nonce	# transazioni inviati	# transazioni inviati

::: Ethereum (3/11)

Gli account dei contratti sono diversi:

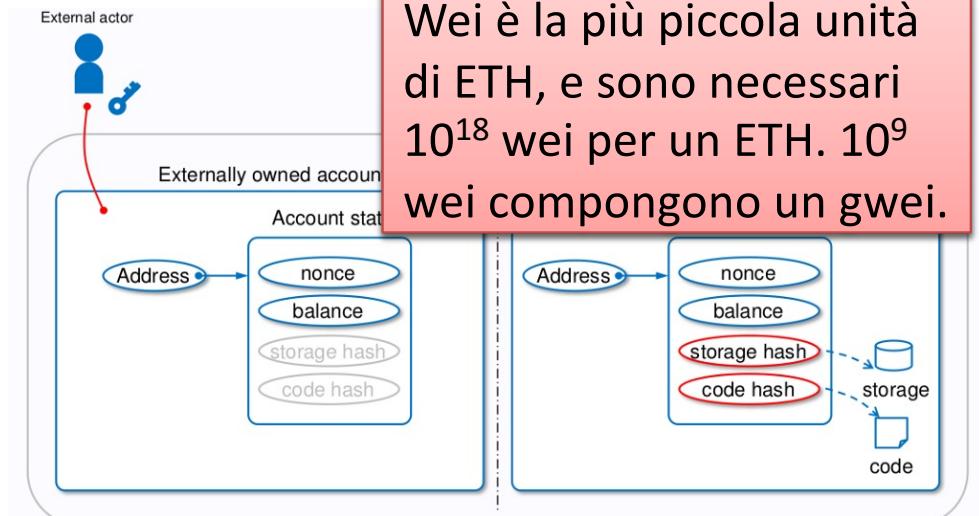
	Account Personale	Account di Contratto
address	H(pub_key)	H(addr + nonce del creatore)
code	\emptyset	Codice
storage	\emptyset	Dati
balance	Saldo ETH (in Wei)	Saldo ETH (in Wei)
nonce	# transazioni inviati	# transazioni inviati



::: Ethereum (3/11)

Gli account dei contratti sono diversi:

	Account Personale	Account di Contratto
address	H(pub_key)	H(addr + nonce del creatore)
code	∅	Codice
storage	∅	Dati
balance	Saldo ETH (in Wei)	Saldo ETH (in Wei)
nonce	# transazioni inviati	# transazioni inviati



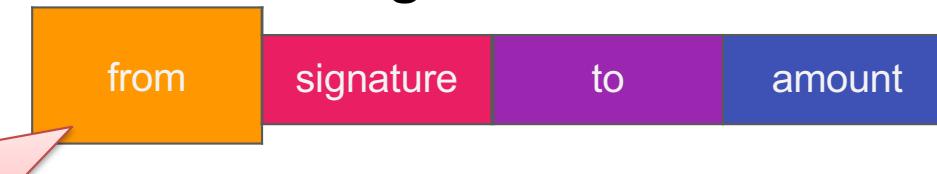
Wei è la più piccola unità di ETH, e sono necessari 10^{18} wei per un ETH. 10^9 wei compongono un gwei.

::: Ethereum (3/11)

Gli account dei contratti sono diversi:

	Account Personale	Account di Contratto
address	H(pub_key)	H(addr + nonce del creatore)
code	\emptyset	Codice
storage	\emptyset	Dati
balance	Saldo ETH (in Wei)	Saldo ETH (in Wei)
nonce	# transazioni inviati	# transazioni inviati

Invece, una transazione ha la seguente struttura dati:



Indirizzo dell'utente di origine della transazione.

::: Ethereum (3/11)

Gli account dei contratti sono diversi:

	Account Personale	Account di Contratto
address	H(pub_key)	H(addr + nonce del creatore)
code	\emptyset	Codice
storage	\emptyset	Dati
balance	Saldo ETH (in Wei)	Saldo ETH (in Wei)
nonce	# transazioni inviati	# transazioni inviati

Invece, una transazione ha la seguente struttura dati:



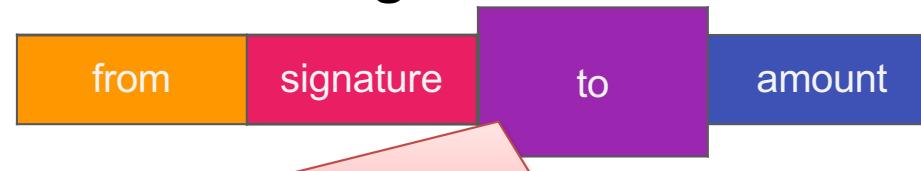
Firma della nuova transazione usando la chiave privata dell'utente creatore.

::: Ethereum (3/11)

Gli account dei contratti sono diversi:

	Account Personale	Account di Contratto
address	H(pub_key)	H(addr + nonce del creatore)
code	\emptyset	Codice
storage	\emptyset	Dati
balance	Saldo ETH (in Wei)	Saldo ETH (in Wei)
nonce	# transazioni inviati	# transazioni inviati

Invece, una transazione ha la seguente struttura dati:



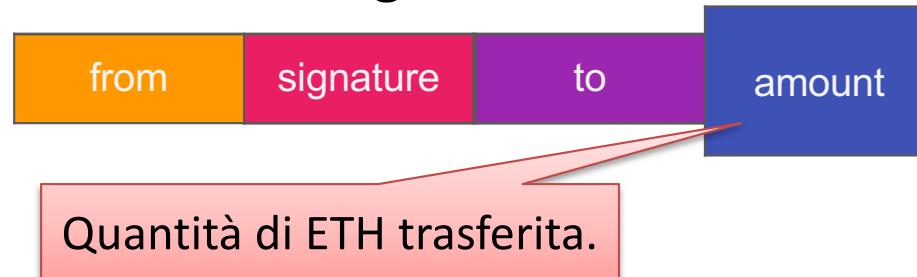
Indirizzo dell'utente di destinazione della transazione.

::: Ethereum (3/11)

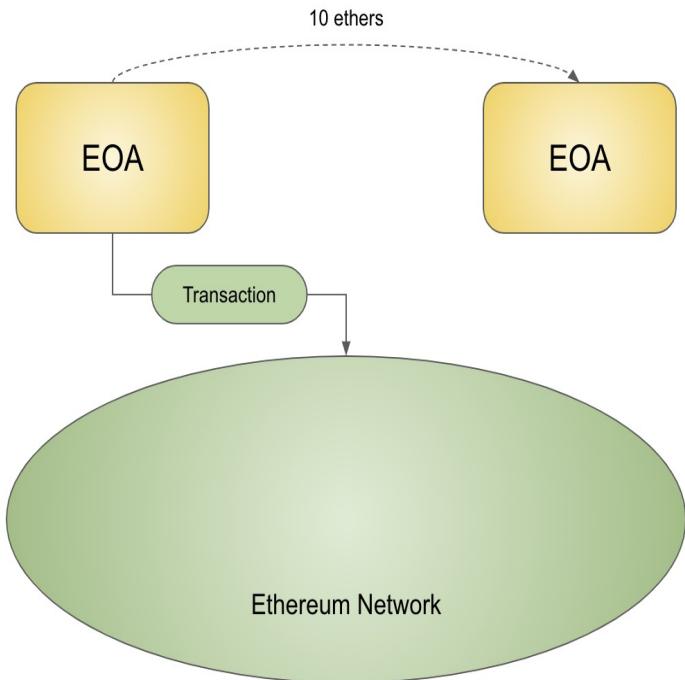
Gli account dei contratti sono diversi:

	Account Personale	Account di Contratto
address	H(pub_key)	H(addr + nonce del creatore)
code	\emptyset	Codice
storage	\emptyset	Dati
balance	Saldo ETH (in Wei)	Saldo ETH (in Wei)
nonce	# transazioni inviati	# transazioni inviati

Invece, una transazione ha la seguente struttura dati:



::: Ethereum (4/11)



- L'avvenuta transazione genera un codice di hash come ricevuta.

- Web3.fromWei converte in Ether un valore espresso in Wei;
- Web3.toWei opera l'inverso;
- La transazione ha richiesto una commissione (in gas), per cui il valore associato all'account[0] è inferiore a 90 Ether.

Terminale Ethereum mediante cui chiamare funzioni della classe principale della libreria web3.js.

```
> web3.fromWei(eth.getBalance(eth.accounts[0]))
100
> web3.fromWei(eth.getBalance(eth.accounts[1]))
100
> eth.sendTransaction({
..... from: eth.accounts[0],
..... to: eth.accounts[1],
..... value: web3.toWei(10)
..... })
"0x497913c178f65613035b22340fcf5bc59c7ed474bfa3c1e798c6dffbeda9da5b"
>
> web3.fromWei(eth.getBalance(eth.accounts[0]))
89.99958
> web3.fromWei(eth.getBalance(eth.accounts[1]))
110
```

::: Ethereum (5/11)

Gli account dei contratti sono diversi:

	Account Personale	Account di Contratto
address	H(pub_key)	H(addr + nonce del creatore)
code	\emptyset	Codice
storage	\emptyset	Dati
balance	Saldo ETH (in Wei)	Saldo ETH (in Wei)
nonce	# transazioni inviati	# transazioni inviati

Invece, una transazione ha la seguente struttura dati:



Nel caso di transazioni per smart contracts la struttura cambia per contenere anche i dati per il contratto:



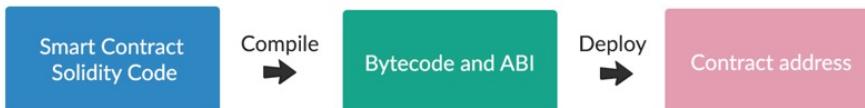
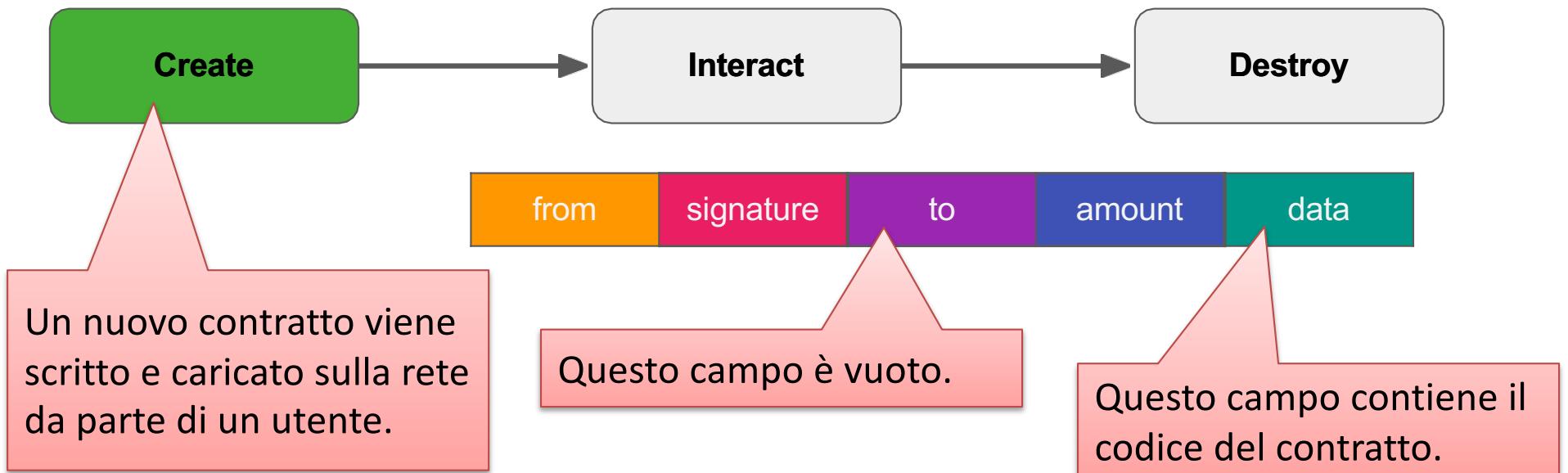
::: Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:



::: Ethereum (6/11)

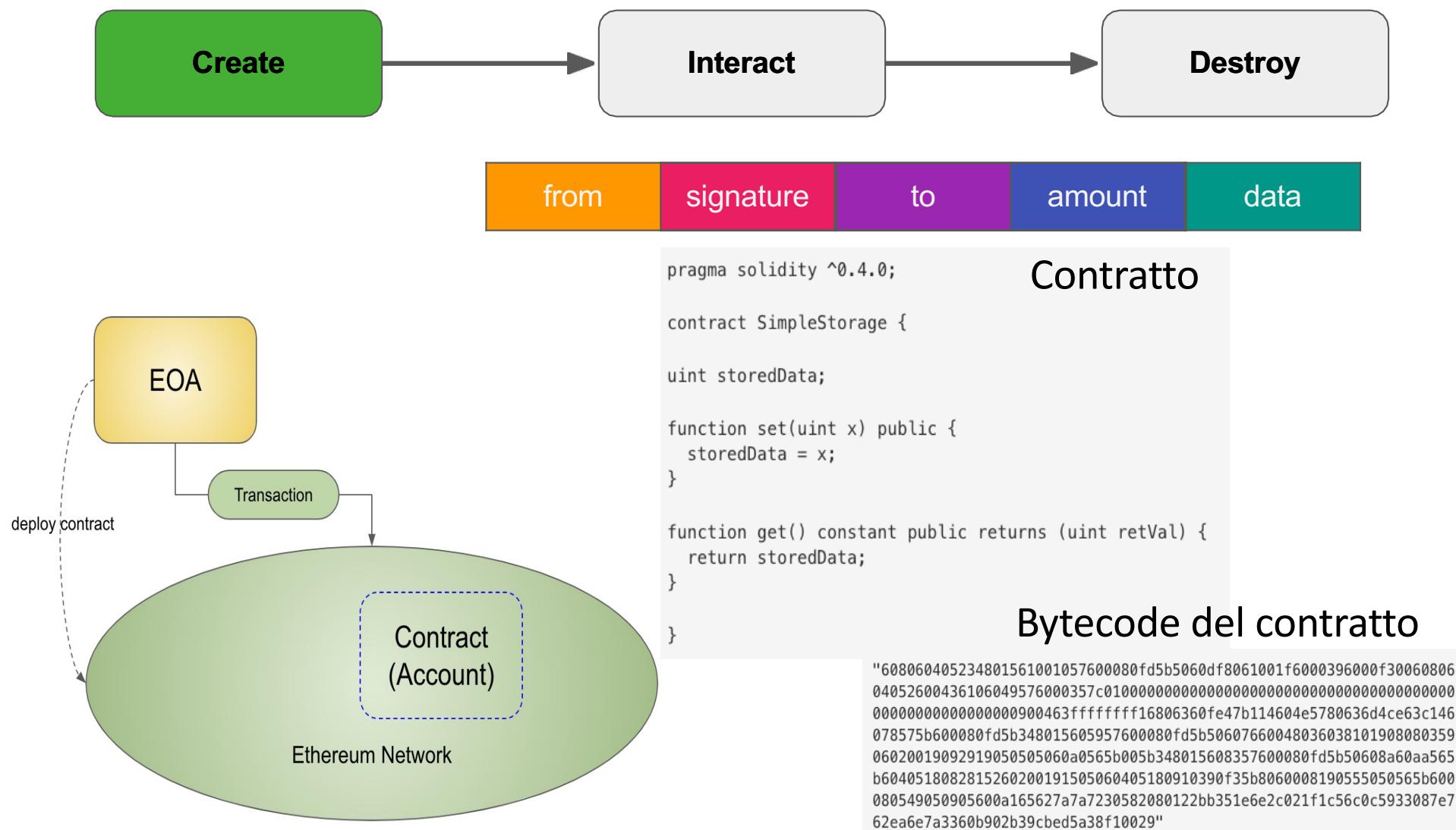
Il ciclo di vita degli smart contract si compone di 3 stati:



Gli smart contracts sono scritti in linguaggi di programmazione di alto livello (e.g. Solidity), tale codice viene compilato dalla EVM e deployato nella Blockchain sottoforma di EVM Bytecode.

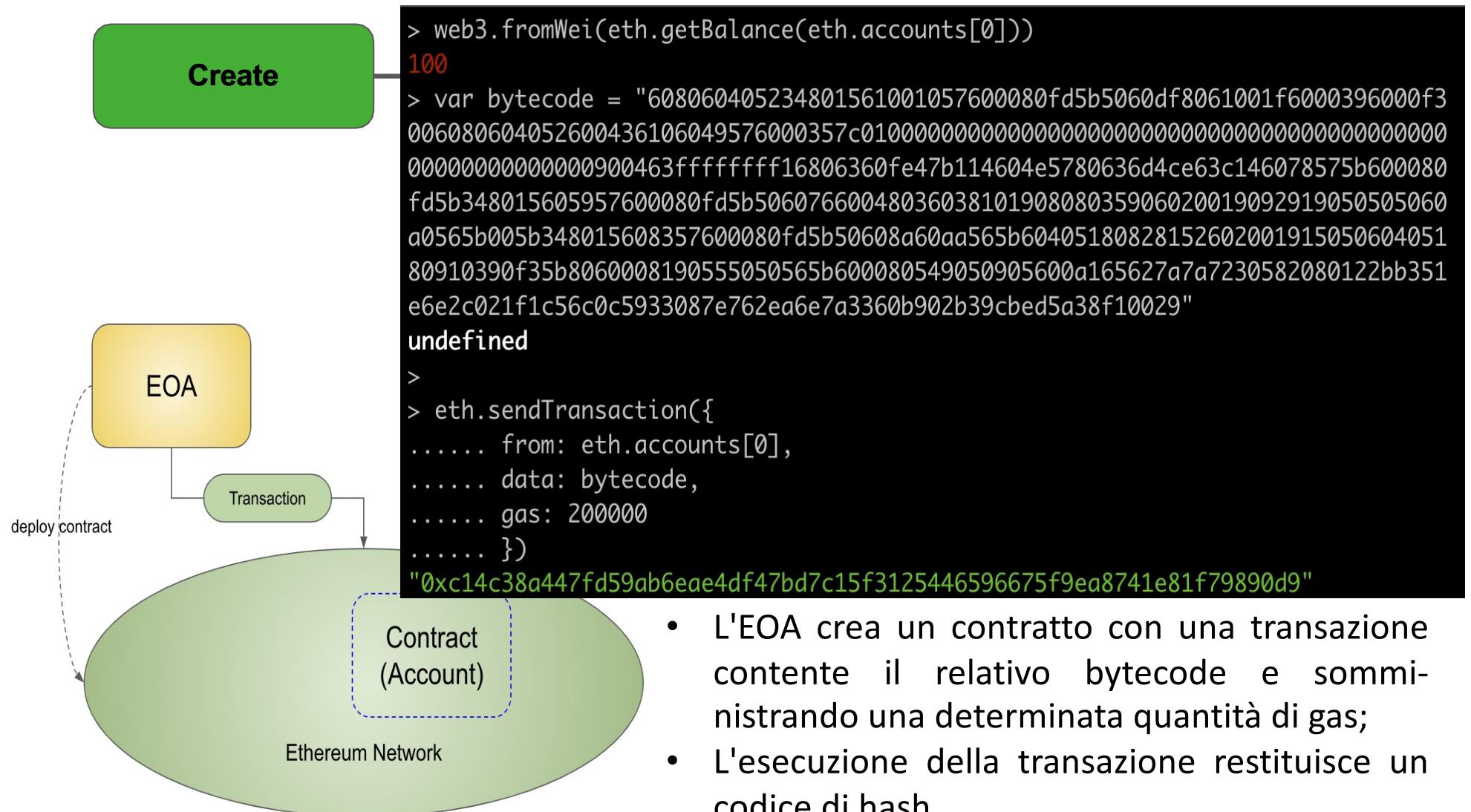
... Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:



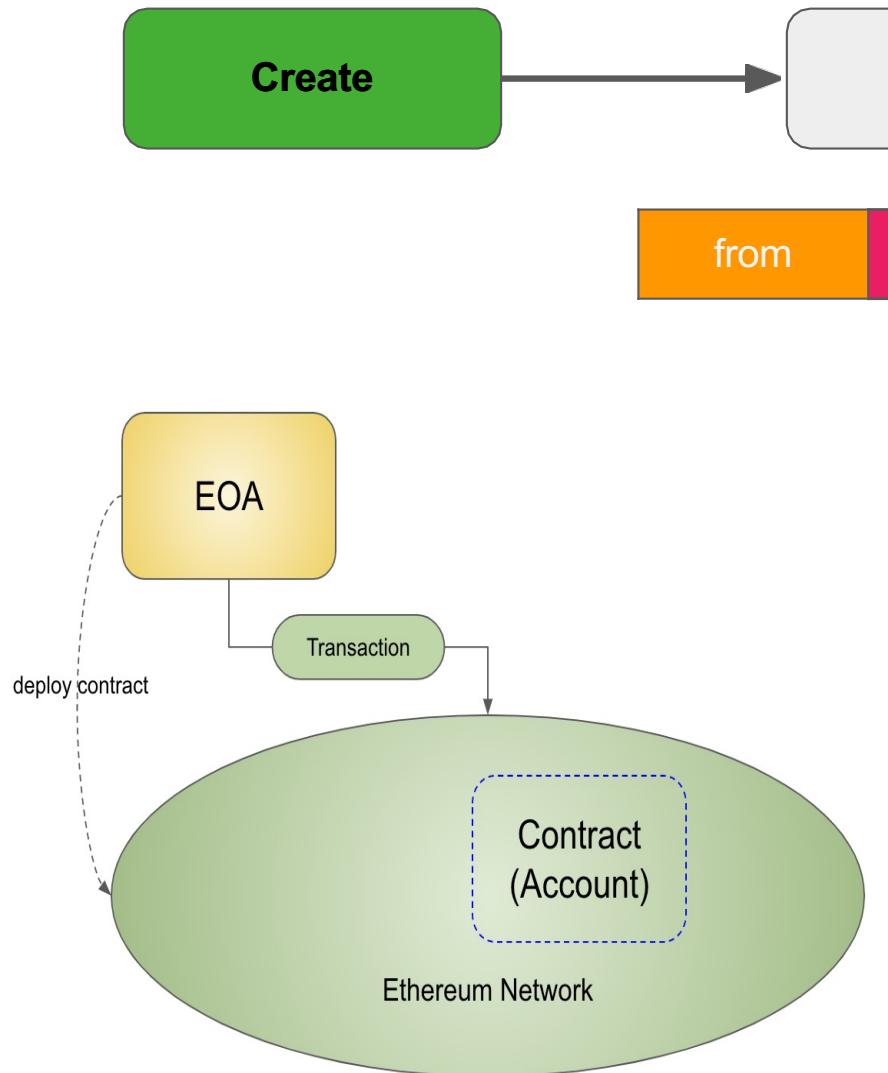
... Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:



::: Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:

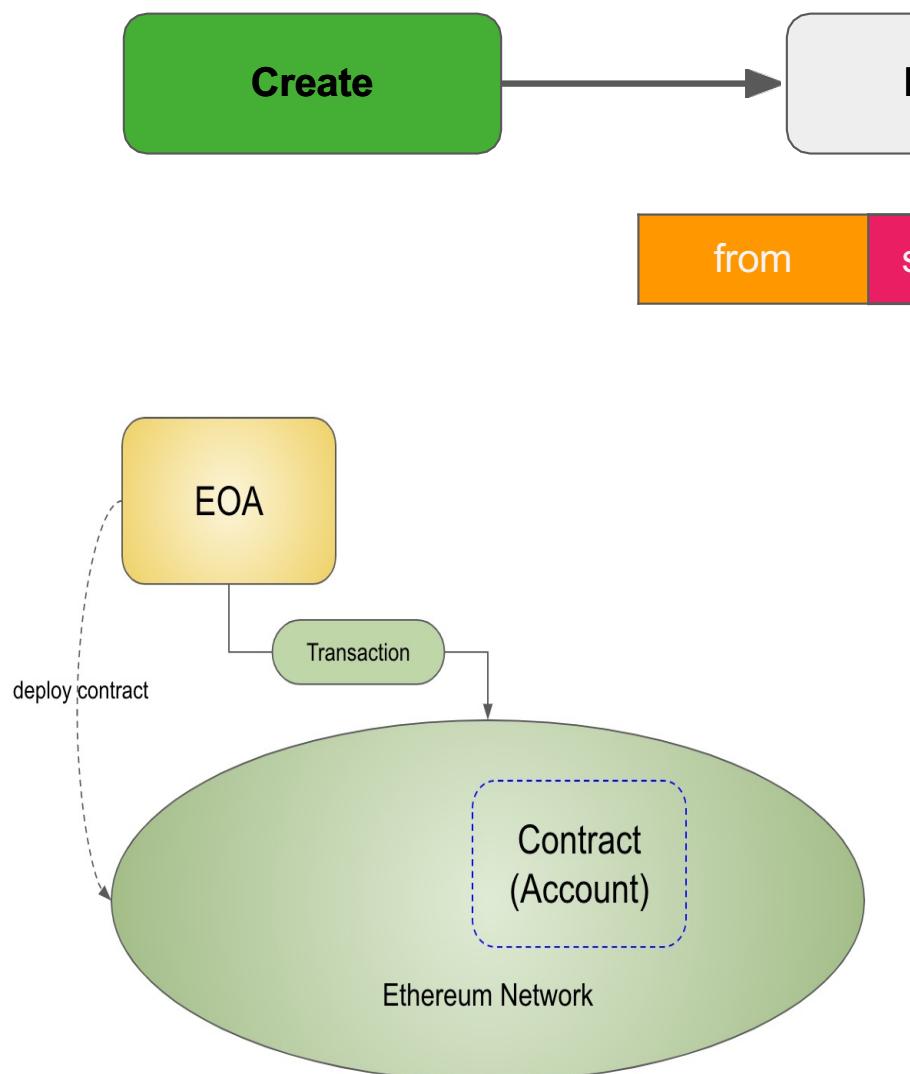


```
> eth.getTransaction("0xc14c38a447fd59ab6eae4df47bd7c15f3125446596675f9ea8741e  
81f79890d9")  
{  
    blockHash: "0xe8a1ed7403baa039f966a22b442cedbf5adb28c9802fea6806e57d75c8ce4  
cf",  
    blockNumber: 1,  
    from: "0x747e967c24abec02b7243e3287cc5ec0f4534a89",  
    gas: 200000,  
    gasPrice: 20000000000,  
    hash: "0xc14c38a447fd59ab6eae4df47bd7c15f3125446596675f9ea8741e81f79890d9",  
    input: "0x608060405234801561001057600080fd5b5060df8061001f6000396000f3006080  
604052600436106049576000357c0100000000000000000000000000000000000000000000000000000000000000  
00000000900463fffffffff16806360fe47b114604e5780636d4ce63c146078575b600080fd5b34  
8015605957600080fd5b5060766004803603810190808035906020019092919050505060a0565b  
005b348015608357600080fd5b50608a60aa565b60405180828152602001915050604051809103  
90f35b8060008190555050565b60008054905090560a165627a7a7230582080122bb351e6e2c0  
21f1c56c0c5933087e762ea6e7a3360b902b39cb5a38f10029",  
    nonce: 0,  
    to: "0x0",  
    transactionIndex: 0,  
    value: 0  
}
```

- Eseguendo `getTransaction(hash_value)` si ottengono le informazioni riguardo la transazione eseguita;
- Il report ottenuto ci informa anche sul blocco creato all'interno della Blockchain.

... Ethereum (6/11)

Il ciclo di vita degli smart contract

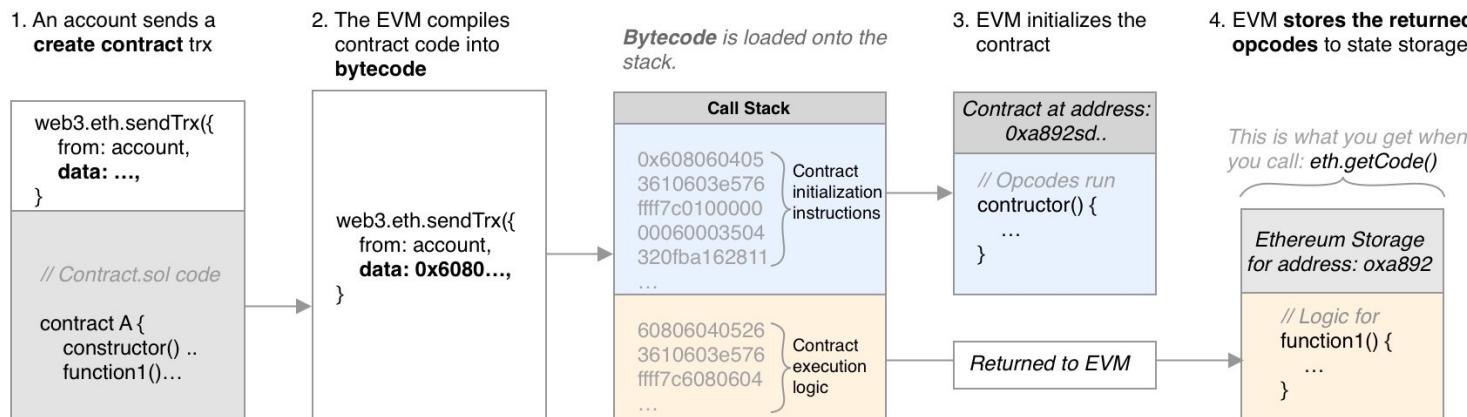
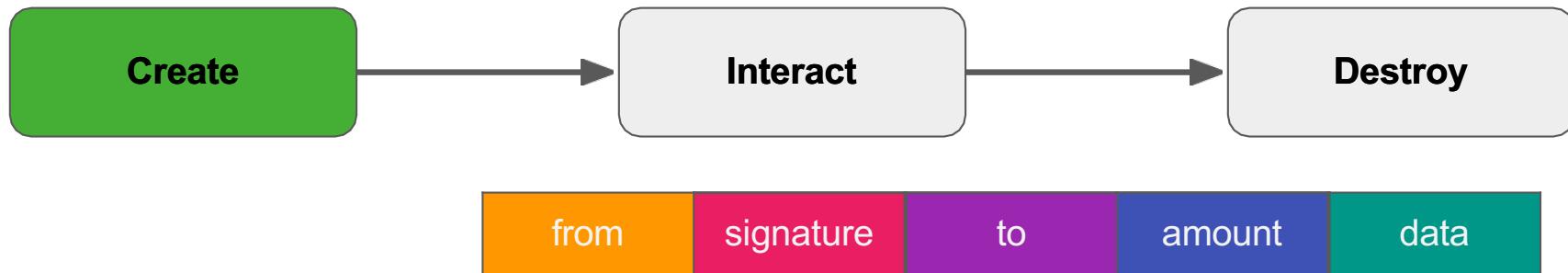


```
> eth.getTransactionReceipt("0xc14c38a447fd59ab6eae4df47bd7c15f3125446596675f9  
ea8741e81f79890d9")  
{  
    blockHash: "0xe8a1ed7403baa039f966a22b442cedbf5adbd28c9802fea6806e57d75c8ce4  
cf",  
    blockNumber: 1,  
    contractAddress: "0xa8e28f1a7031968fb830e5a70c4b246b07f64d2a",  
    cumulativeGasUsed: 112213,  
    gasUsed: 112213,  
    logs: [],  
    logsBloom: "0x0000000000000000000000000000000000000000000000000000000000000000  
00000000000000000000000000000000000000000000000000000000000000000000000000000000  
00000000000000000000000000000000000000000000000000000000000000000000000000000000  
00000000000000000000000000000000000000000000000000000000000000000000000000000000  
00000000000000000000000000000000000000000000000000000000000000000000000000000000  
00000000000000000000000000000000000000000000000000000000000000000000000000000000  
00000000000000000000000000000000000000000000000000000000000000000000000000000000",  
    status: "0x1",  
    transactionHash: "0xc14c38a447fd59ab6eae4df47bd7c15f3125446596675f9ea8741e81  
f79890d9",  
    transactionIndex: 0  
}
```

- Eseguendo `getTransactionReceipt(hash_value)` si ottiene un "ricevuta", con l'indirizzo del contratto e l'informazione sul gas speso
 - In ingresso alla transazione è stato inserito un valore di gas pari a 200000, ma poiché il "gaslimit" è pari a 90000, la parte non spesa viene restituita all'EOA

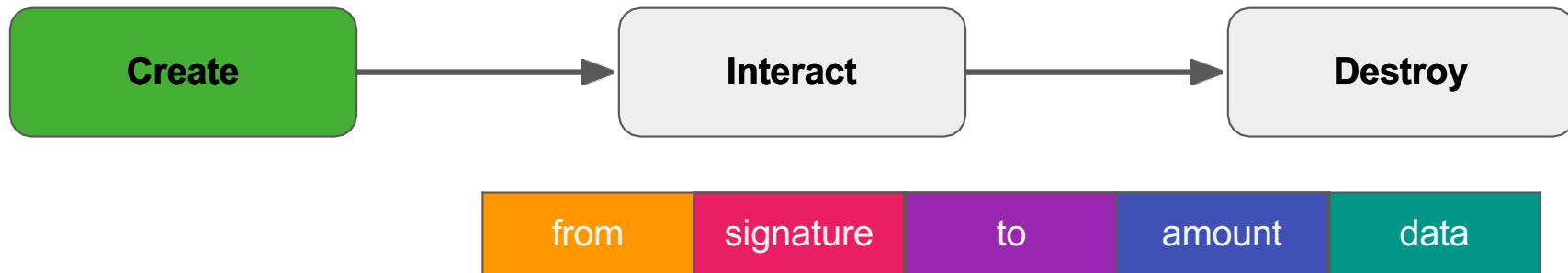
::: Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:

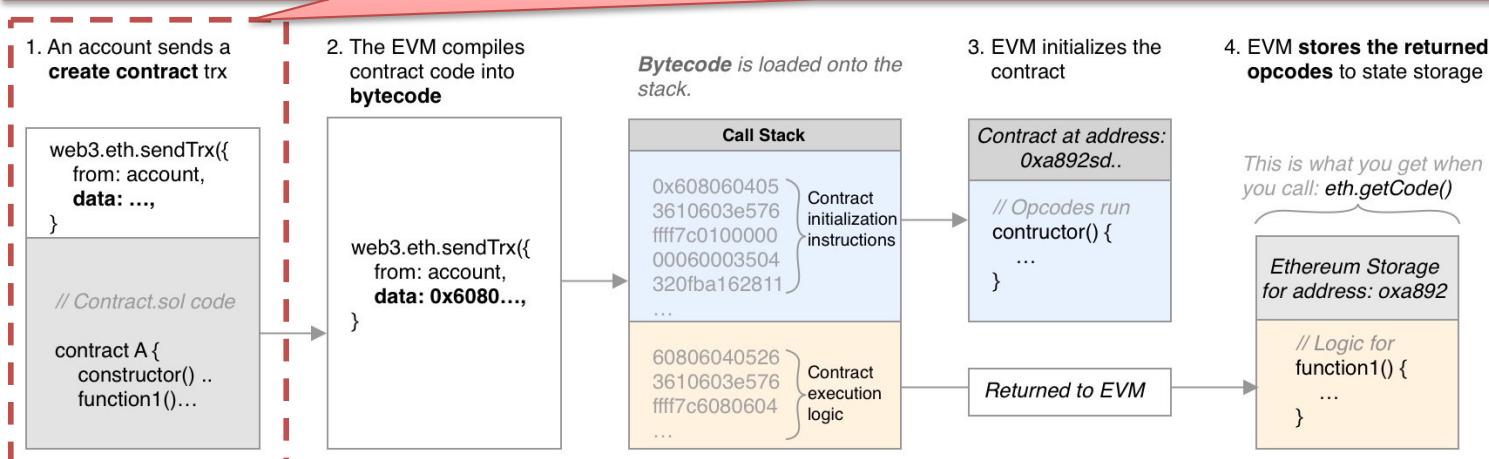


::: Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:

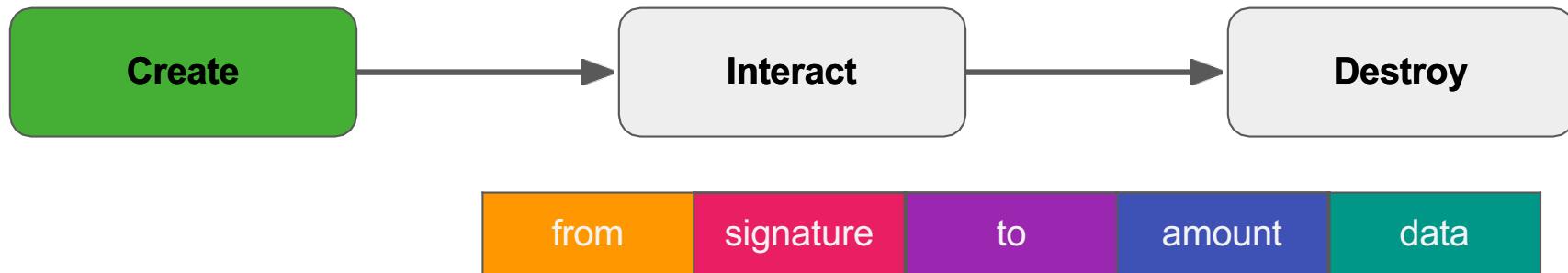


Un EOA o un contratto invia una transazione contenente dati, ma nessun indirizzo di destinatario. Questo formato indica all'EVM che è una creazione di contratto, non una normale transazione di invio / chiamata.

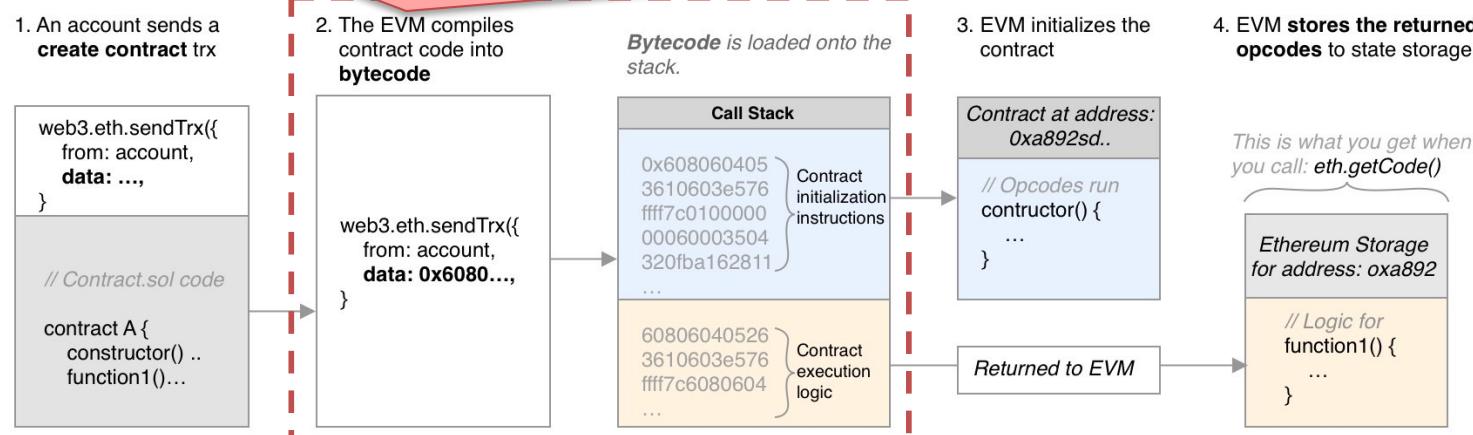


::: Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:

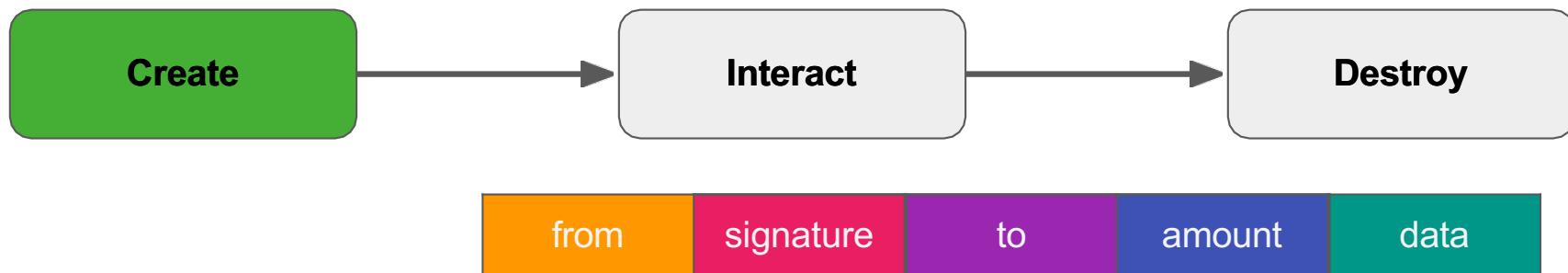


EVM compila il codice del contratto in Solidity ottenendo il bytecode, che traduce direttamente in opcode il codice del contratto, che vengono eseguiti in un singolo stack di chiamate.

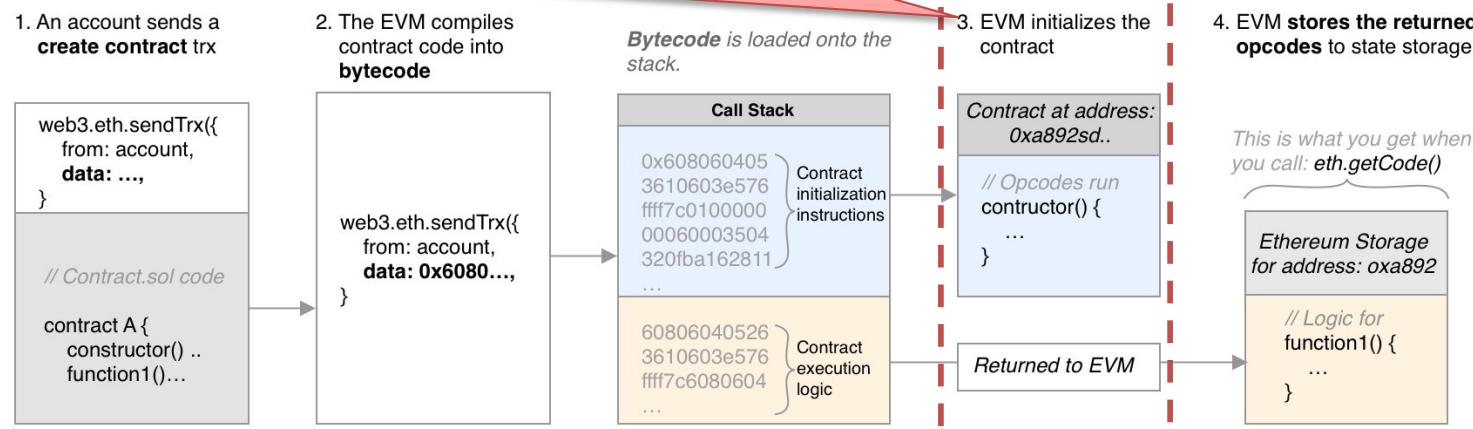


::: Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:

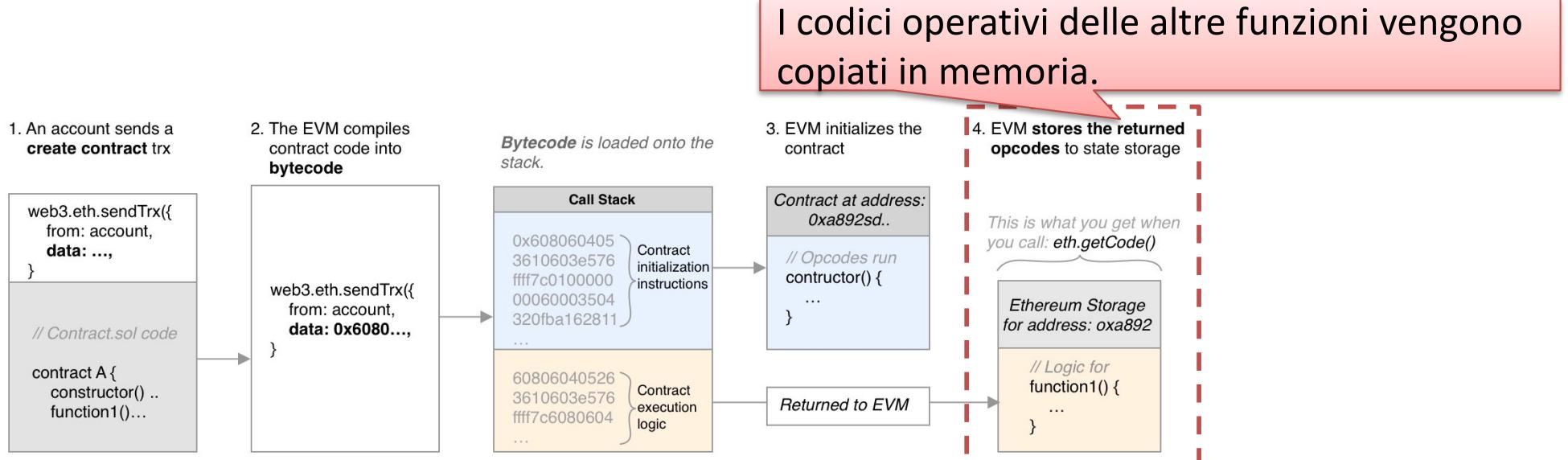
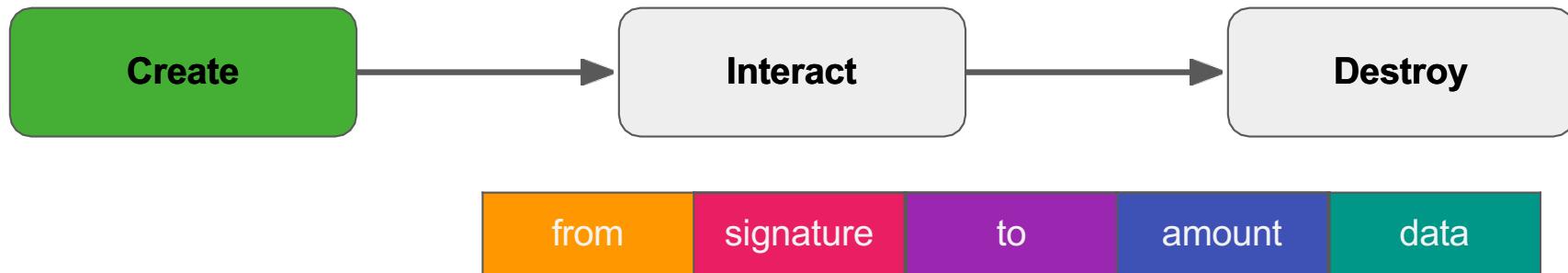


Durante la creazione del contratto, EVM esegue solo il codice di inizializzazione fino a quando non raggiunge la prima istruzione STOP o RETURN nello stack, il contenuto oltre tale punto è ritornato alla EVM e rappresenta le funzioni che si possono invocare. Durante questa fase, viene assegnato un indirizzo al contratto.



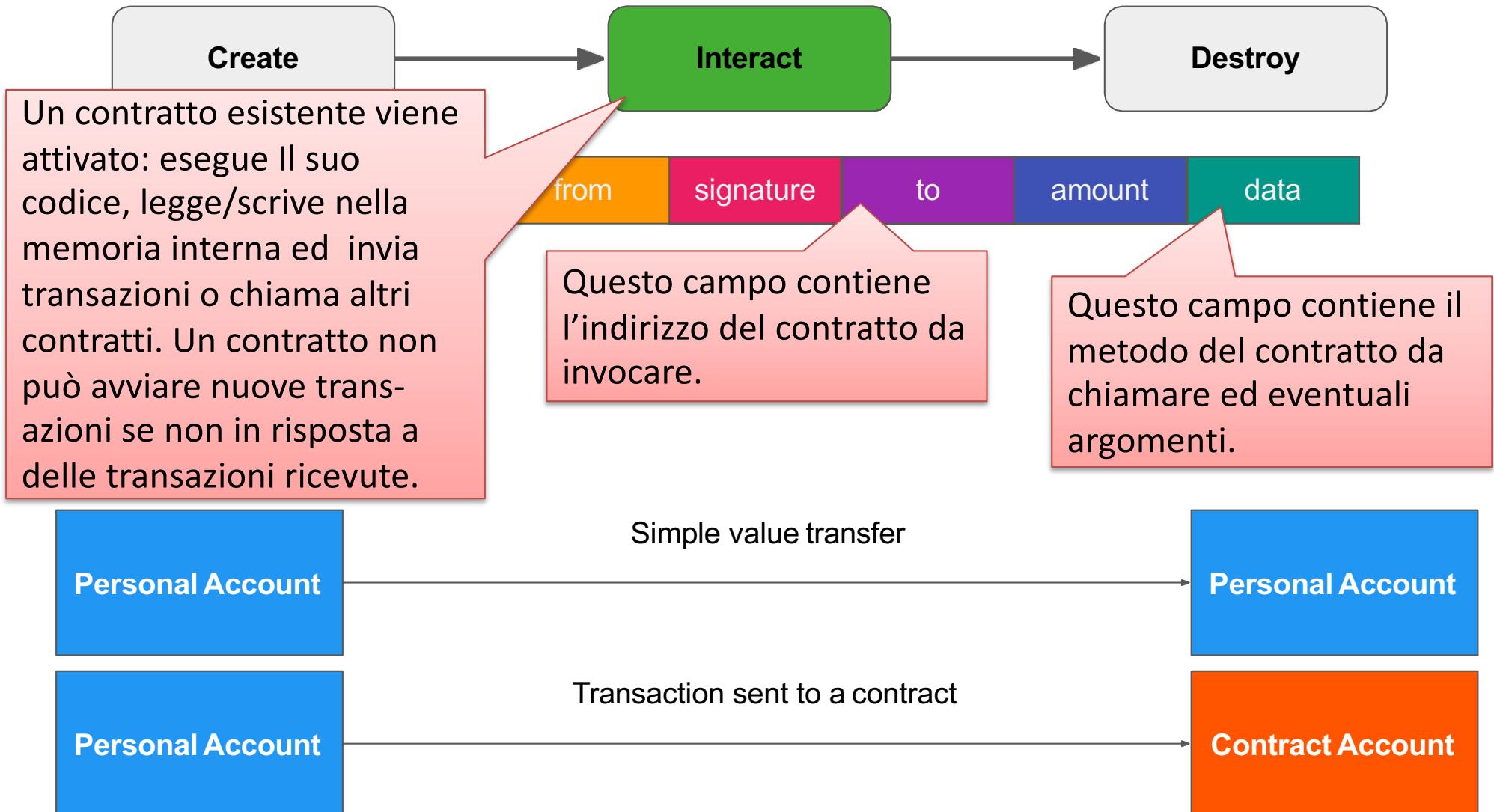
::: Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:



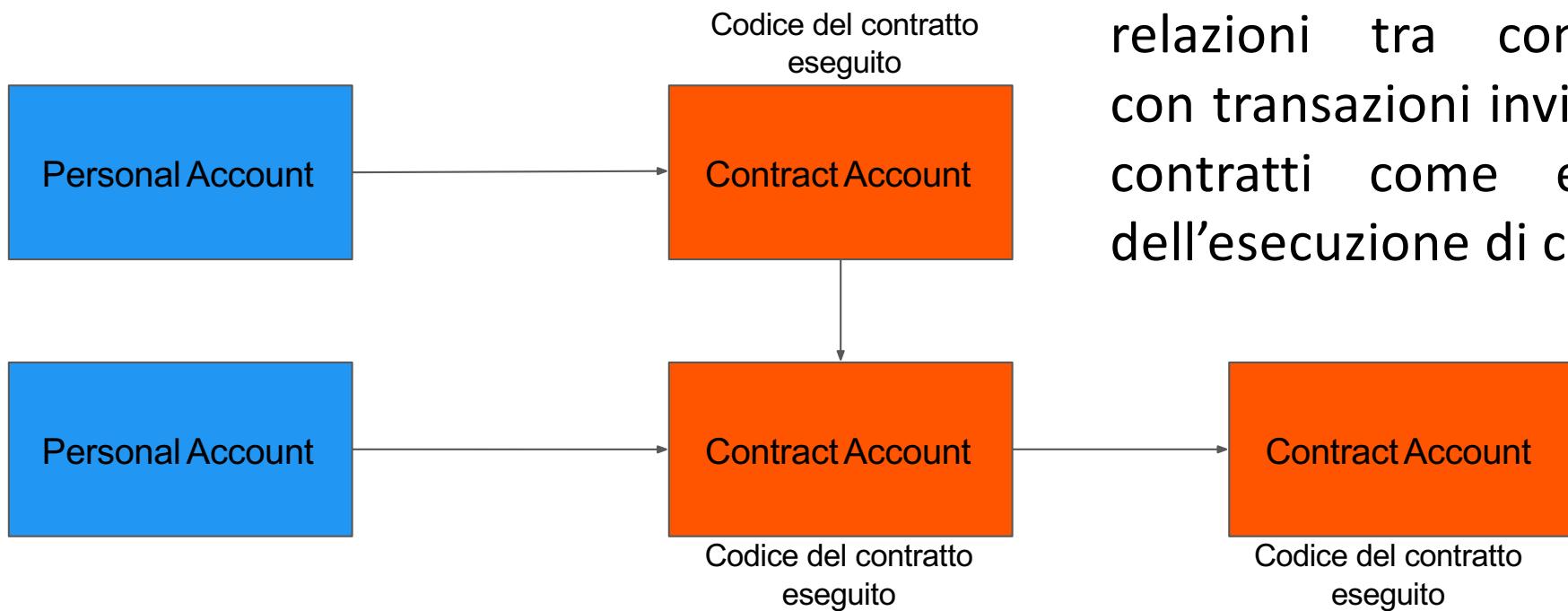
::: Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:



::: Ethereum (6/11)

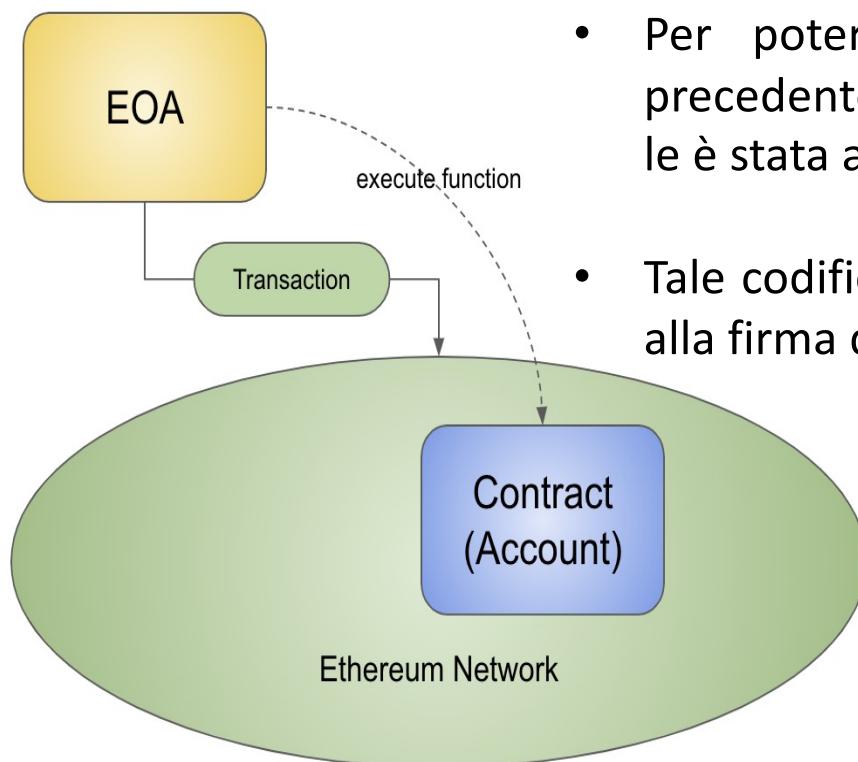
Il ciclo di vita degli smart contract si compone di 3 stati:



Possono sussistere delle relazioni tra contratti, con transazioni inviati tra contratti come effetto dell'esecuzione di codice.

::: Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:



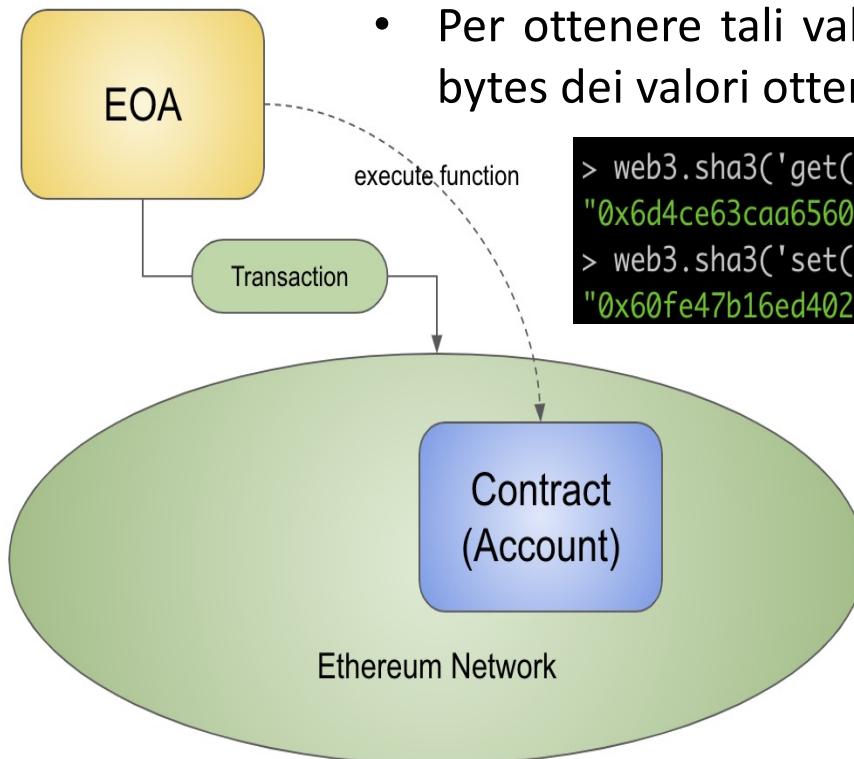
- Per poter accedere alle funzioni implementate nel precedente contratto è necessario ottenere la codifica che le è stata assegnata nel bytecode;
- Tale codifica è realizzata applicando una funzione di hash alla firma del metodo.

... Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:



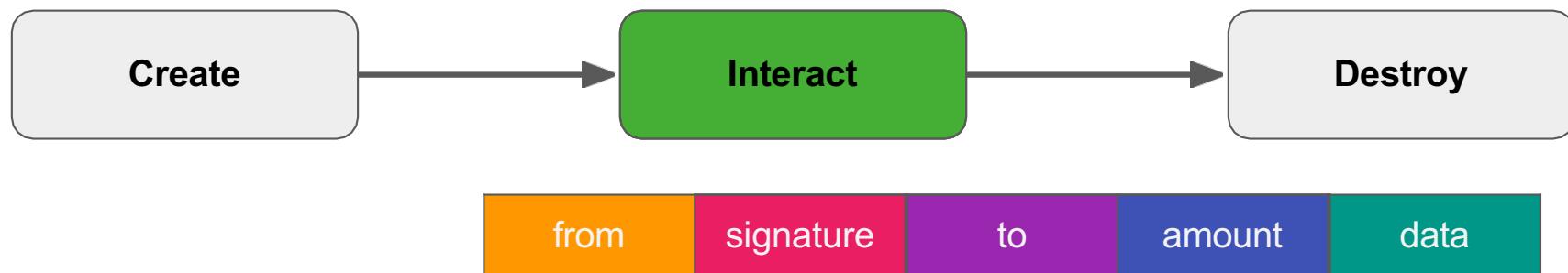
- Per ottenere tali valori è necessario considerare i primi 4 bytes dei valori ottenuti dall'hash delle funzioni:



Bytecode del contratto

... Ethereum (6/11)

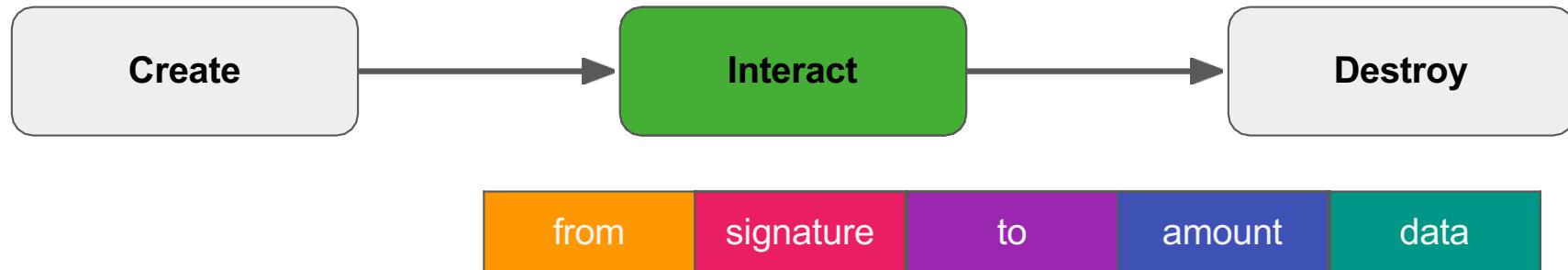
Il ciclo di vita degli smart contract si compone di 3 stati:



- Effettuando una call si verifica che il valore contenuto nella variabile storedData è 0;
 - Viene caricato un nuovo valore tramite la sendTransaction;
 - Rieffettuando la call si ottiene il valore aggiornato.
 - Le call sono transazioni eseguite direttamente dalla VM e non riportate nella Blockchain.

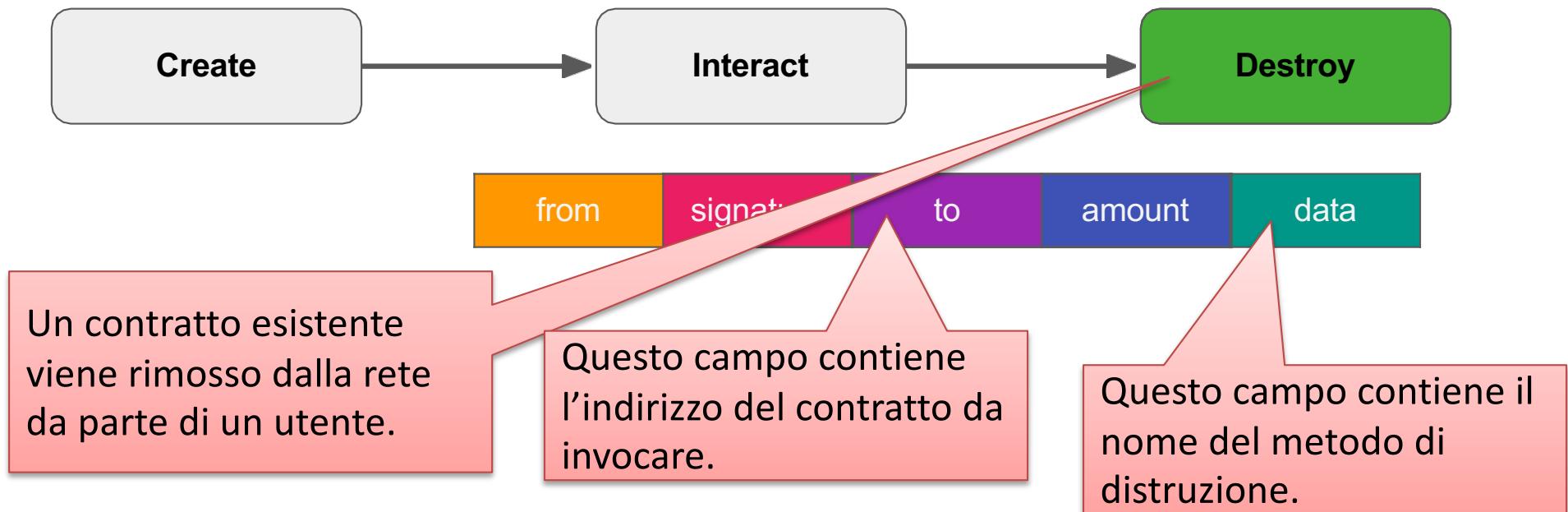
... Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:



::: Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:

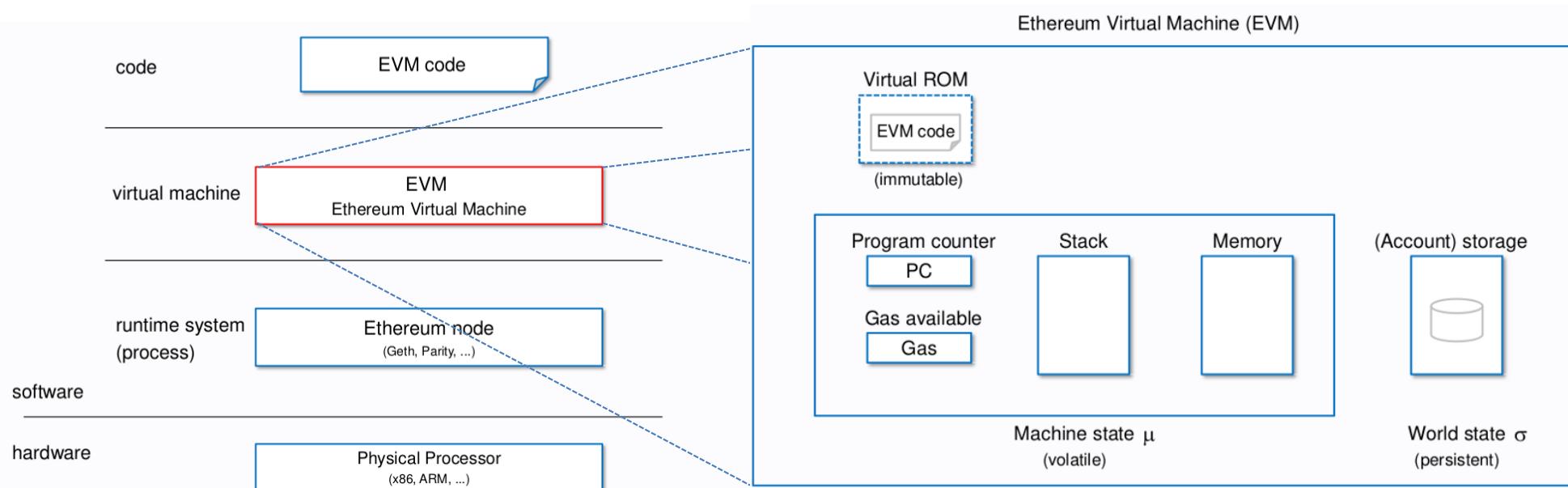


::: Ethereum (6/11)

Il ciclo di vita degli smart contract si compone di 3 stati:



La EVM è un ambiente di esecuzione isolata (sandbox) dei contratti con uno stack, un parallelismo interno a 32-byte, e codici operativi che formano una macchina quasi Turing-completa.



::: Ethereum (7/11)

Solidity è un linguaggio di programmazione con tipizzazione statica ed orientato agli oggetti per la scrittura di applicazioni che implementano la logica di business incorporata in smart contract su Ethereum e altre piattaforme blockchain.

- È progettato attorno alla sintassi ECMAScript ma si differenzia per una tipizzazione statica e tipi di ritorno variadici.
- Supporta inoltre tipi complessi definiti dall'utente, ad esempio struct ed enumerazioni, che consentono di raggruppare i tipi di dati correlati.
- I contratti supportano l'ereditarietà, inclusa l'ereditarietà multipla con linearizzazione C3.
- È stata inoltre introdotta un'interfaccia binaria dell'applicazione (ABI) che facilita più funzioni type-safe all'interno di un singolo contratto.

::: Ethereum (7/11)

Solidity è un linguaggio di programmazione con tipizzazione statica ed orientato agli oggetti per la scrittura di applicazioni che implementano la logica di business incorporata in smart contracts.

Ethereum e

Il controllo del tipo delle strutture dati membro di contratti avviene in fase di compilazione, non in fase di esecuzione come per i linguaggi tipizzati in modo dinamico.

una tipizzazione statica e tipi di ritorno variadici.

- Supporta inoltre tipi complessi definiti dall'utente, ad esempio struct ed enumerazioni, che consentono di raggruppare i tipi di dati correlati.
- I contratti supportano l'ereditarietà, inclusa l'ereditarietà multipla con linearizzazione C3.
- È stata inoltre introdotta un'interfaccia binaria dell'applicazione (ABI) che facilita più funzioni type-safe all'interno di un singolo contratto.

::: Ethereum (7/11)

Solidity è un linguaggio di programmazione con tipizzazione statica ed orientato agli oggetti per la scrittura di applicazioni che implementano la logica di business incorporata in smart contract su Ethereum e altre piattaforme blockchain.

- È progettato attorno alla sintassi ECMAScript ma si differenzia per una tipizzazione statica e tipi di ritorno vari
- Supporta inoltre tipi complessi definiti dall'utente ad esempio

ECMAScript è la specifica tecnica di un linguaggio di scripting, destinato a garantire l'interoperabilità delle pagine Web tra diversi browser Web. L'implementazione più conosciuta di questo linguaggio è JavaScript.

- I contratti supportano l'ereditarietà, inclusa l'ereditarietà multipla con linearizzazione C3.
- È stata inoltre introdotta un'interfaccia binaria dell'applicazione (ABI) che facilita più funzioni type-safe all'interno di un singolo contratto.

::: Ethereum (7/11)

Solidity è un linguaggio di programmazione con tipizzazione statica ed orientato agli oggetti per la scrittura di applicazioni che implementano la logica di business incorporata in smart contract su Ethereum e altre piattaforme blockchain.

- È progettato attorno alla sintassi ECMAScript ma si differenzia per una tipizzazione statica e tipi di ritorno variadici.
- Supporta inoltre tipi complessi definiti dall'utente, ad esempio struct ed enumerazioni, che consentono di definire e comporre i tipi di dati.

Un parametro di ritorno variadico accetta zero o più valori di un tipo specificato, e la funzione restituisce una quantità variabile di valori.

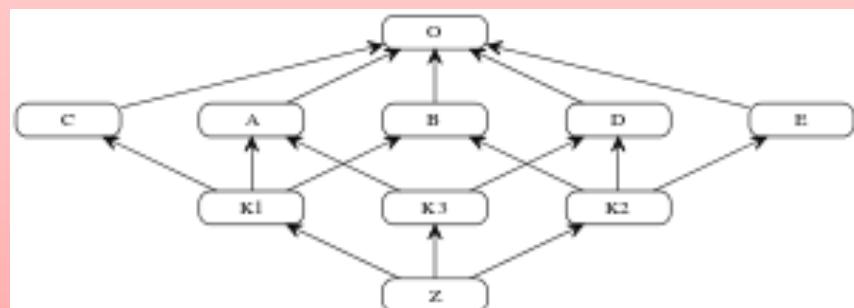
con linearizzazione C3.

- È stata inoltre introdotta un'interfaccia binaria dell'applicazione (ABI) che facilita più funzioni type-safe all'interno di un singolo contratto.

.. Eborum (7/11)

La linearizzazione C3 è un algoritmo utilizzato per ottenere l'ordine in cui i metodi devono essere ereditati in presenza di ereditarietà multipla.

La linearizzazione è la somma della classe più un merge delle linearizzazioni dei suoi genitori e un elenco dei genitori stessi. Il merge viene eseguito selezionando la prima intestazione delle liste che non appare nella coda di nessun'altra. L'elemento selezionato viene rimosso e aggiunto all'elenco di output. Se non è possibile selezionare un'intestazione valida, allora l'unione è impossibile da calcolare a causa di ordinamenti incoerenti delle dipendenze e nessuna linearizzazione è possibile.



$$L(O) := [O]$$

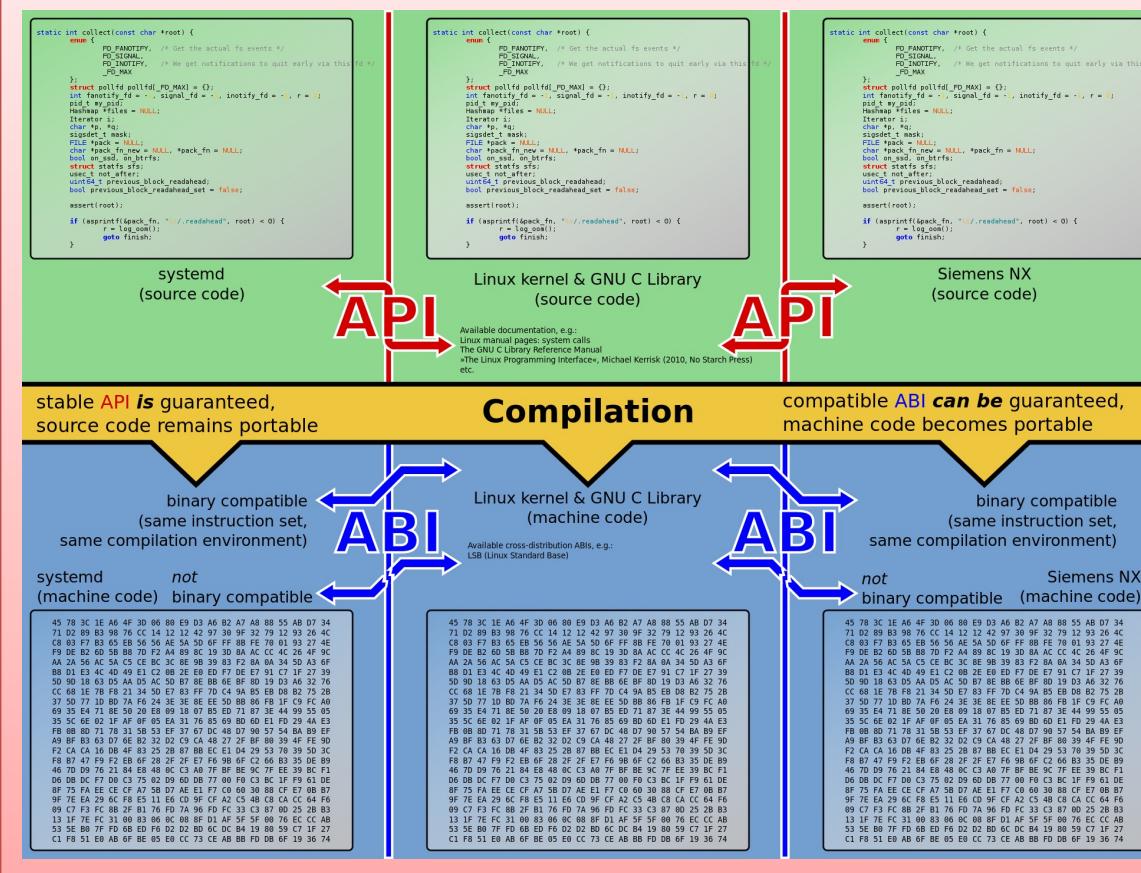
$$L(A) := [A] + \text{merge}(L(O), [O]) = [A] + \text{merge}([O], [O]) = [A, O]$$

$$\begin{aligned} L(K1) &:= [K1] + \text{merge}(L(A), L(B), L(C), [A, B, C]) = \\ &[K1] + \text{merge}([A, O], [B, O], [C, O], [A, B, C]) = \\ &[K1, A] + \text{merge}([O], [B, O], [C, O], [B, C]) = [K1, A, B, C] \\ &+ \text{merge}([O], [O], [O]) = [K1, A, B, C, O] \end{aligned}$$

con linearizzazione C3.

- È stata inoltre introdotta un'interfaccia binaria dell'applicazione (ABI) che facilita più funzioni type-safe all'interno di un singolo contratto.

... Ethereum (7/11)



Un ABI è un'interfaccia tra due moduli di programma binario, e definisce come si accede alle strutture dati o alle routine nel codice macchina. È espresso in un formato di basso livello, dipendente dall'hardware; al contrario, un'API definisce questo accesso nel codice sorgente, in formato di livello relativamente alto, indipendente dall'hardware.

- È stata inoltre introdotta un'interfaccia binaria dell'applicazione (ABI) che facilita più funzioni type-safe all'interno di un singolo contratto.

::: Ethereum (8/11)



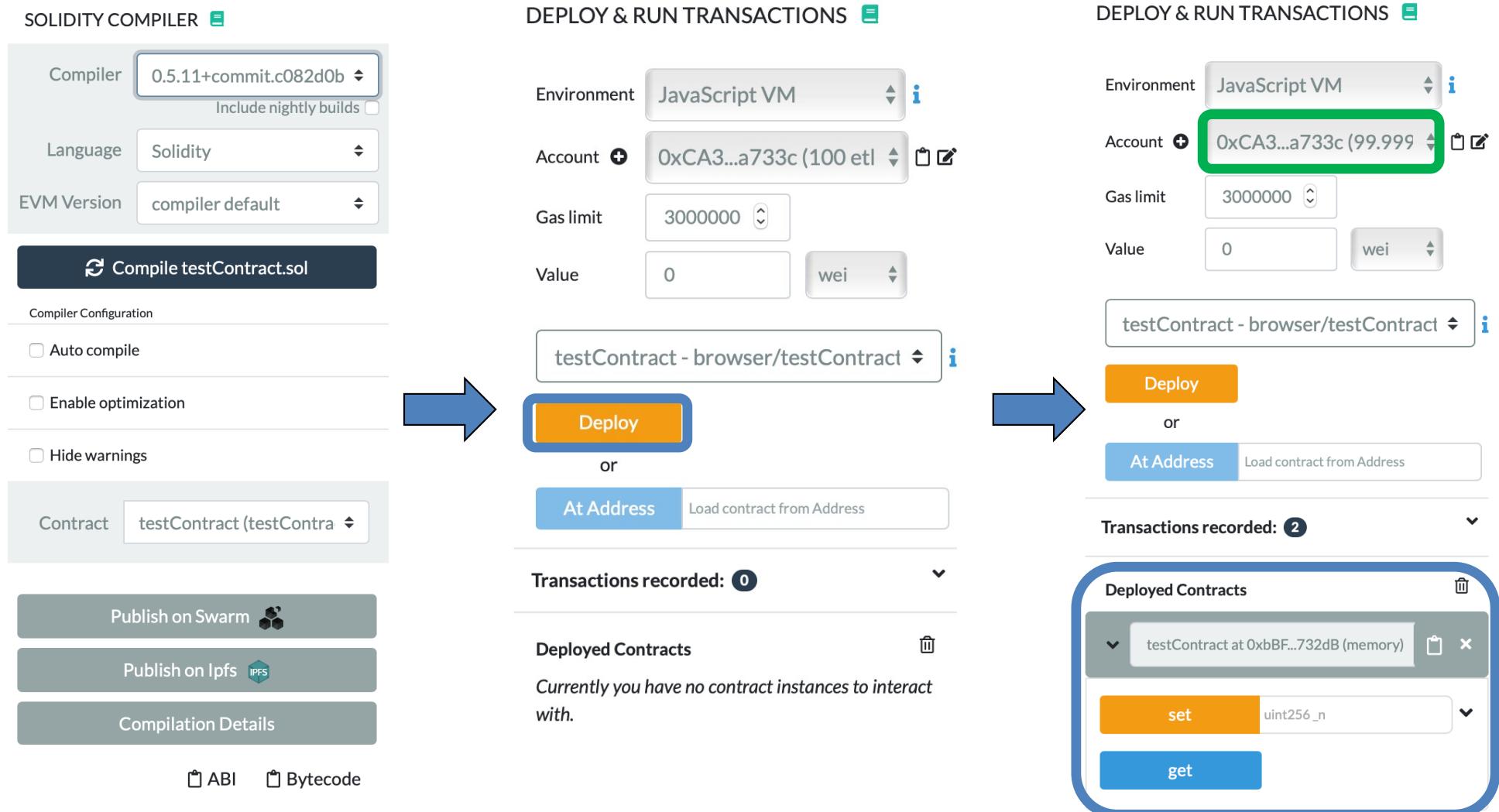
È possibile scrivere applicazioni in Solidity mediante l'IDE Remix.

<https://remix.ethereum.org>

```
testContract.sol ✘
1 pragma solidity ^0.5.1;
2
3 contract testContract {
4
5     uint amount=13;
6
7     function set(uint _n) public {
8         amount = _n;
9     }
10
11    function get() view public returns (uint) {
12        return amount;
13    }
14 }
```

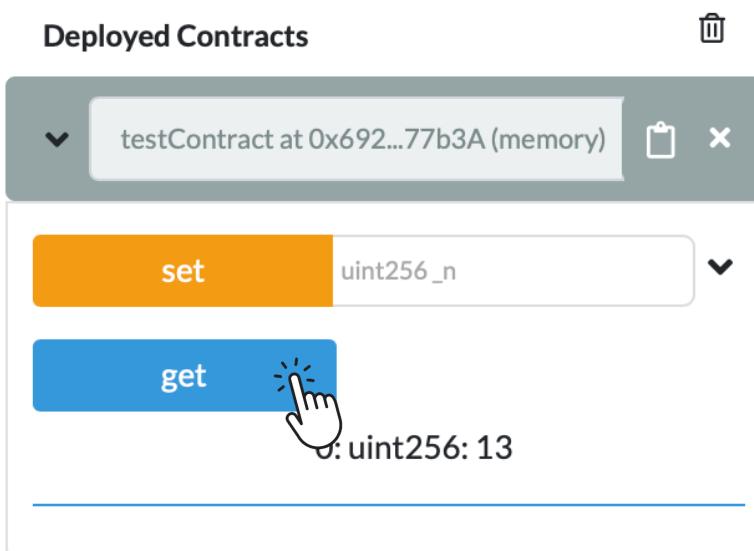
::: Ethereum (8/11)

Deployment dello smart contract:

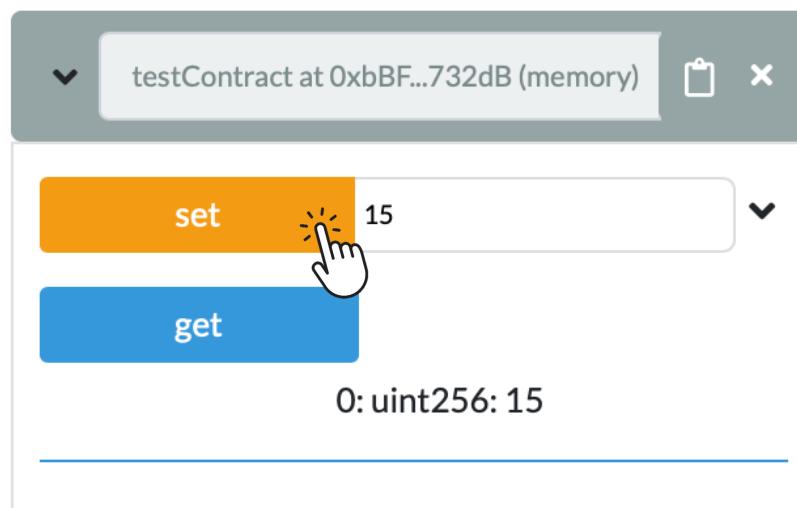


::: Ethereum (9/11)

Esempio di get



Esempio di set e successiva get



::: Ethereum (10/11)

Secondo esempio di smart contract:

The screenshot shows a code editor interface with two tabs at the top: "testContract.sol" and "financialContract.sol". The "financialContract.sol" tab is active, indicated by a blue background and a white "X" icon. Below the tabs is a horizontal line. The main area contains the Solidity code for the "financialContract" contract. The code includes pragmas, a constructor, three functions (receiveFunds, getBalance, withdrawFunds), and an assert statement. Line numbers are visible on the left side of the code.

```
1 pragma solidity ^0.5.1;
2
3 contract financialContract {
4
5     address payable issuer;
6
7     constructor() public{
8         issuer = msg.sender;
9     }
10
11     function receiveFunds() payable public{
12
13     }
14
15     function getBalance() view public returns (uint){
16         return address(this).balance;
17     }
18
19     function withdrawFunds(uint funds) public{
20         assert(issuer == msg.sender);
21         issuer.transfer(funds);
22     }
23
24 }
```

::: Ethereum (11/11)

Deployment dello smart contract:

DEPLOY & RUN TRANSACTIONS

Environment: JavaScript VM

Account: 0xCA3...a733c (99.999)

Gas limit: 3000000

Value: 300 wei

financialContract - browser/financia

Deploy or At Address Load contract from Address

Transactions recorded: 10

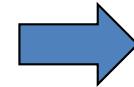
Deployed Contracts

financialContract at 0x8c1...401f5 (memory)

receiveFunds

withdrawFunds uint256 funds

getBalance



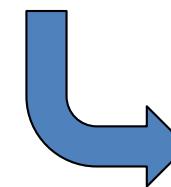
financialContract at 0x8c1...401f5 (memory)

receiveFunds

withdrawFunds uint256 funds

getBalance

int256: 300



financialContract at 0x8c1...401f5 (memory)

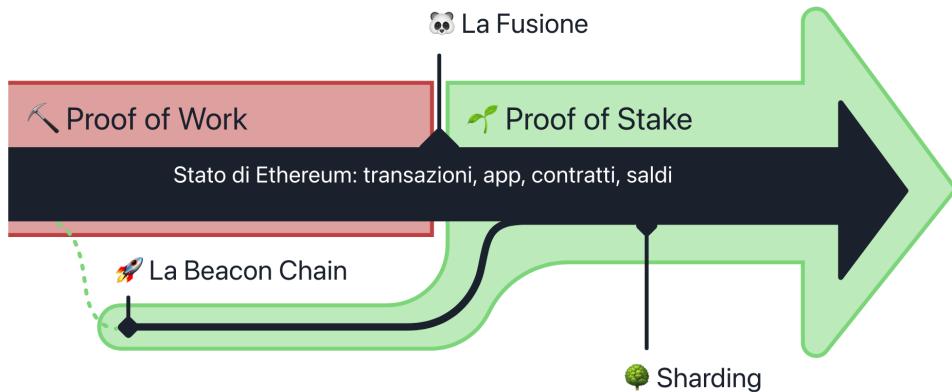
receiveFunds

withdrawFunds uint256 funds

getBalance

0: uint256: 150

::: Ethereum 2.0 (1/5)



Nel dicembre 2020, Ethereum aveva iniziato due blockchain parallele, una legacy con PoW (Mainnet) e una nuova con PoS (Beacon Chain). Il merge di Settembre 2022 le ha unite in un'unica blockchain con PoS.

Ethereum è passato alla Proof-of-Stake nel Settembre 2022 perché più sicuro, meno energivoro e migliore per l'implementazione di nuove soluzioni di ridimensionamento rispetto alla precedente Proof-of-Work:

- i validatori puntano esplicitamente del capitale sotto forma di ETH in uno smart contract. Questo ETH puntato funge quindi da garanzia che può essere distrutta se il validatore si comporta in modo disonesto o pigro. Il validatore è quindi responsabile del controllo della validità dei nuovi blocchi e, occasionalmente, della creazione e della propagazione di nuovi blocchi stessi.

::: Ethereum 2.0 (2/5)

Per partecipare come validatore, un utente deve depositare 32 ETH in un contratto di deposito ed eseguire tre software:

- un client di esecuzione, un client di consenso e un validatore.

Dopo aver depositato il proprio ETH, l'utente si unisce a una coda di attivazione che limita il tasso di nuovi validatori che si uniscono alla rete.

Una volta attivati, i validatori ricevono nuovi blocchi. Le transazioni nel blocco vengono rieseguite e la firma del blocco viene verificata. Il validatore invia quindi un voto (o attestazione) a favore di quel blocco se valido.

Mentre in proof-of-work la tempistica dei blocchi è determinata dalla difficoltà di mining, in proof-of-stake il tempo è fisso ed è diviso in slot (12 secondi) ed epoche (32 slot).

- Un validatore viene selezionato casualmente per proporre blocchi in ogni slot e creare un nuovo blocco e inviarlo ad altri nodi.
- In ogni slot, viene scelto casualmente un comitato di validatori, i cui voti servono a determinare la validità del blocco proposto.

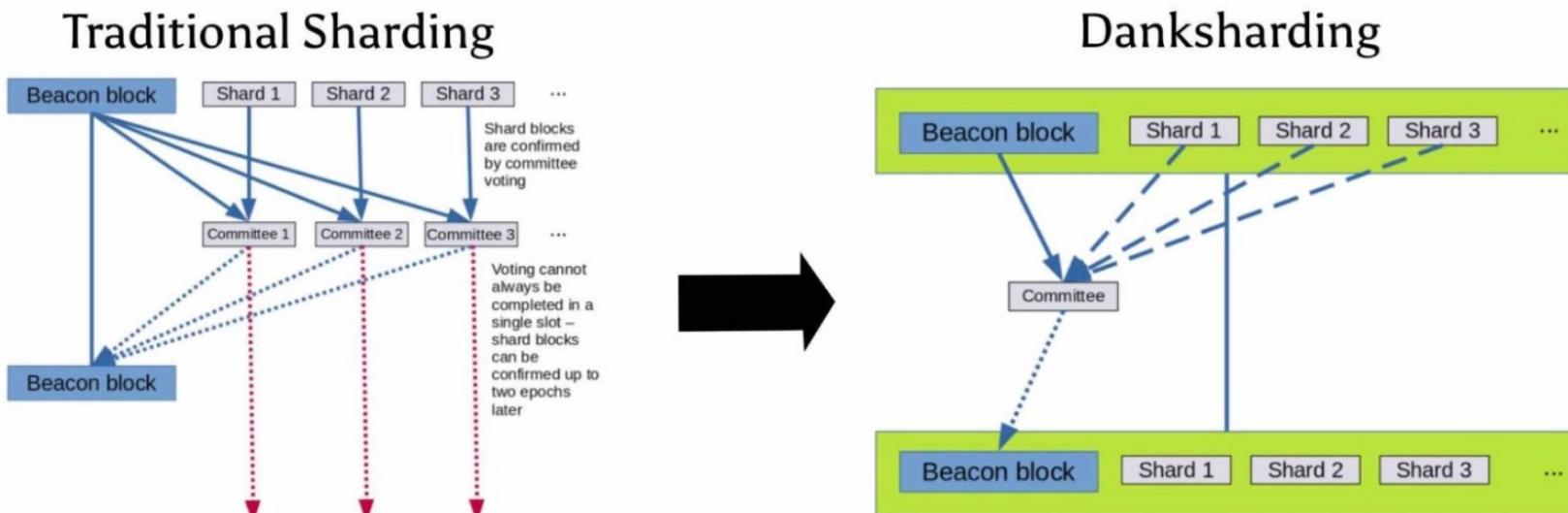
::: Ethereum 2.0 (3/5)

La minaccia di un attacco del 51% esiste ancora su Proof of Stake così come su Proof of Work, ma è ancora più rischiosa per gli aggressori.

- Un utente malintenzionato avrebbe bisogno del 51% degli ETH in stake e utilizzare i propri attestati per assicurarsi che il suo fork preferito fosse quello con il maggior numero di attestati accumulati.
- Tuttavia, un punto di forza della proof-of-stake è che la comunità ha flessibilità nell'organizzare un contrattacco.
 - i validatori onesti potrebbero decidere di continuare a costruire sulla catena di minoranza e ignorare il fork dell'attaccante.
 - Potrebbero anche decidere di rimuovere con la forza l'attaccante dalla rete e distruggere il loro ETH in stake.

::: Ethereum 2.0 (4/5)

L'aggiornamento Cancun-Deneb di Ethereum include EIP-4844, noto anche come Proto-Danksharding, che mira a migliorare la scalabilità e ridurre le tariffe del gas. Danksharding, che utilizza “blob” di dati di grandi dimensioni per aumentare il throughput delle transazioni, è l'attuale proposta di progettazione per lo sharding su Ethereum. Proto-Danksharding è un passo intermedio verso il Danksharding completo, con l'obiettivo di ridurre i costi di transazione e aumentare la produttività.



::: Ethereum 2.0 (5/5)

Proto-danksharding

1. Transazioni blob-carrying: Al centro del proto-danksharding c'è l'introduzione delle "transazioni blob-carrying." In parole povere, queste transazioni comportano la creazione di oggetti binari di grandi dimensioni (blob) che trasportano un payload di dati di 125 kilobyte. A differenza dei calldata tradizionali archiviati in modo permanente sulla Ethereum Virtual Machine (EVM), i blob sono memorizzati a livello di consenso, offrendo un'alternativa economica per l'archiviazione dei dati.
2. Schema di impegno polinomiale KZG: Il proto-danksharding incorpora lo schema di impegno polinomiale KZG (Kate, Zaverucha, Goldberg). Questa tecnica crittografica consente di verificare i dati delle transazioni all'interno dei blob senza rivelarne il contenuto completo. Questo migliora gli aspetti di sicurezza e privacy legati all'archiviazione e al recupero dei dati.

::: Ethereum 2.0 (5/5)

Le prove di frode sono un sistema in cui per accettare il risultato di un calcolo, si richiede a qualcuno con un deposito puntato di firmare un messaggio del modulo "Certifico che se si esegue il calcolo C con l'input X, si ottiene l'output Y". Ti fidi di questi messaggi per impostazione predefinita, ma lasci aperta l'opportunità a qualcun altro con un deposito puntato di effettuare una sfida (un messaggio firmato che dice "Non sono d'accordo, l'output è Z").

Solo quando c'è una sfida, tutti i nodi eseguono il calcolo. Qualunque delle due parti abbia sbagliato perde il proprio deposito e tutti i calcoli che dipendono dal risultato di quel calcolo vengono ricalcolati.

Meccanismi di protezione saranno integrati:

- le **prove di frode** possono essere utilizzate per dimostrare la validità delle transazioni e punire attività disoneste.
- Il secondo meccanismo utilizza il campionamento casuale per prevenire la collusione. Se i validatori non sanno in quale frammento finiranno, diventa più difficile coordinare un attacco al sistema. In questo scenario, un attacco del 51% su una catena di shard diventa impossibile da eseguire.

::: Solidity (1/17)

Solidity è un DSL specifico per la programmazione di Smart Contracts. A differenza dei linguaggi più popolari come Java, Solidity ha delle limitazioni date dalla blockchain.

- Non è possibile memorizzare grandi quantità di dati nelle variabili.
- I cicli for e while sono limitati dal gas e ciò significa che non è possibile creare un ciclo for per un array di migliaia di elementi.
- Non sono disponibili costrutti di programmazione concorrente.

Uno Smart Contract inizia sempre con la versione di Solidity utilizzata in questo modo:

```
pragma solidity 0.5.12;
```

In questo caso si sta usando la versione 0.5.12. È inoltre possibile specificare l'ultima versione con il simbolo ^ per indicare al compilatore che si desidera utilizzare la versione più recente di Solidity o quella indicata.

```
pragma solidity ^0.5.12;
```

::: Solidity (2/17)

```
contract Prova {  
}
```

Segue il preambolo indicando il nome del contratto in maiuscolo come una classe Java. Il file che lo contiene deve essere salvato con estensione .sol, esempio Prova.sol. Il file va nominato in base al nome del contratto scritto all'interno.

Solidity è un linguaggio tipizzato staticamente: bisogna specificare il tipo di variabile prima del suo nome, proprio come in Java.

- uint, uint8, uint16, uint24, uint32, uint64, uint128 e uint256

Uint significa “numero intero senza la virgola” ovvero un numero positivo. Se si tenta di memorizzare un numero negativo, andrà in “overflow”. Il numero dopo uint indica la dimensione della variabile in bit. uint8 memorizza qualsiasi numero intero minore o uguale a $2^8 - 1$ che è 255. Qualsiasi numero superiore a 255 manderà la variabile in “overflow” e significa che supererà la capacità massima consentita.

::: Solidity (3/17)

```
contract Prova {  
    uint256 provaNumero;  
}
```

È possibile convertire il tipo mediante un casting esplicito :

```
uint256 provaNumero = 10;  
uint8 altroNumero = uint8(provaNumero);
```

Il valore di default memorizzato in una variabile inizializzata senza valori è 0.

- Address - Rappresenta l'indirizzo pubblico di ogni utente.

```
contract Prova {  
    uint256 provaValore;  
    address indirizzo_proprietario;  
}
```

Gli indirizzi si dichiarano come stringhe ma senza virgolette. Ad esempio:
address indirizzo_proprietario =
0x055E36b2BB4eF1121F99e3e134c66A8DCb6ef58e; Assicurati di aggiungere
il 0x all'inizio dell'indirizzo. Tutti gli indirizzi hanno il valore di default 0x0.

::: Solidity (4/17)

- Stringhe e Byte memorizzano le stesse informazioni. La differenza è che le stringhe sono utilizzate solo per il testo ed impiegano più gas dei Byte. È inoltre possibile utilizzare byte1, byte2, byte3 fino a byte32, dove i numeri indicano la dimensione in byte.

```
contract Prova {uint256 provaNumero;  
address indirizzo_proprietario;  
bytes32 nome = "Giovanni";  
string articolo = "testo dell'articolo di grandezza variabile";  
}
```

- Boolean è il tipo più semplice di variabile e può contenere solo due valori: vero o falso. Tutti i booleani sono falsi di default.

::: Solidity (5/17)

- Mapping rappresenta il tipo delle memorie associative caratterizzate da una chiave ed un valore.

```
mapping (string = uint256) etaPersone;
```

- Struct indica le strutture, che consentono di creare oggetti con attributi.

```
struct persona {  
    uint256 id;  
    string nome;  
    uint256 anni;  
}
```

```
persona giovanni = persona (1, "Giovanni", 35);
```

::: Solidity (6/17)

- Array di elementi di un tipo predefinito

Esistono 2 tipi di array: quelli di dimensioni fisse e gli array di dimensioni variabili.

- Array di dimensioni fisse e limitate

```
string[5] myNames;
```

Per aggiungere valori a questo array, si utilizza la posizione dell'elemento da modificare e il tipico operatore [].

```
myNames[2] = 'Dario';
```

- Array di dimensioni dinamiche ed illimitate

```
string[] myNames;
```

```
myNames.push('Dario');
```

```
myNames[40] = 'Carlo'; // Non funzionerà a meno che  
quell'elemento non sia già stato modificato con push
```

::: Solidity (7/17)

In ogni Smart Contract ci sono variabili globali:

- now o block.timestamp: entrambe restituiscono lo stesso valore, il valore unix dell'ora corrente, un valore di 10 numeri.
- msg.sender, msg.gas e msg.value: il messaggio contiene l'indirizzo del mittente, il gas rimanente disponibile in quel punto della funzione e la quantità di Ether inviata a quella funzione in wei.
- days, seconds, minutes, hours, weeks e years: sono solo unità per definire valori temporali. Per esempio, 1 minuto equivale a 60 secondi. Tutte queste variabili restituiranno valori espressi in secondi. Vediamo un esempio:
 - uint256 myTime = 25 days;
 - uint256 mySecondTime = 1 minutes;

- this: restituisce l'indirizzo dello Smart Contract corrente. Con this.balance si ottiene quanto ether in wei questo contratto ha.

::: Solidity (7/17)

Sono disponibili tre tipologie di memorizzazione:

- Storage
 - Scritture in blockchain che sono permanenti
 - Costoso, parte del gas viene rimborsato quando lo storage viene eliminato
- Memory
 - Un array di byte con dimensioni dello slot di 32 byte mantenuto da EVM
 - Memorizzato durante l'esecuzione della funzione
 - Economico, ma i costi per operazione si ridimensionano quadraticamente
- Stack
 - Sono accessibili solo 16 variabili sullo stack
 - Più economico, manipolato dall'assembly in-line

Variabili di stato sono sempre nello storage, gli argomenti delle funzioni sono in memoria. Variabili locali di tipo riferimento possono essere in memoria o storage, e la modalità va dichiarata esplicitamente quando si dichiara la variabile. Mentre, le variabili locali di tipi di valore sono sullo stack.

::: Solidity (7/17)

Sono disponibili tre tipologie di memorizzazione:

- Storage
 - Scritture in blockchain che sono permanenti
 - Costoso, parte del gas viene rimborsato quando lo storage viene eliminato
- Memory
 - Un array di byte con dimensioni dello slot di 32 byte
 - Memorizzato durante l'esecuzione della funzione
 - Economico, ma i costi per operazione si ridimensionano quadraticamente

Se la variabile memorizza i propri dati; è un tipo di valore. Se contiene una posizione dei dati; è un tipo di riferimento.

I tipi di riferimento in Solidity sono costituiti da Array, Struct, e Mapping.

— Più economico, manipolato dall'assembly

Variabili di stato sono in memoria, gli argomenti delle funzioni sono in memoria. Variabili locali di tipo riferimento possono essere in memoria o storage, e la modalità va dichiarata esplicitamente quando si dichiara la variabile. Mentre, le variabili locali di tipi di valore sono sullo stack.

::: Solidity (8/17)

Quale delle due è più costosa?

```
Asset[] public assets;
function getUsingStorage(uint _assetIdx) public function getUsingMemory(uint _assetIdx) public
    returns (string memory) {
    Asset stor    returns (string memory) {
        Asset memory a
    Asset stor    return asset.r
    return ass
}
}
27108 gas
}
29398 gas
```

::: Solidity (8/17)

Quale delle due è più costosa?

```
Asset[] public assets;
function getUsingStorage(uint _assetIdx) public
    returns (string memory) {
    Asset storage asset = assets[_assetIdx];
    return asset.name;
}
function getUsingMemory(uint _assetIdx) public
    returns (string memory) {
    Asset memory asset = assets[_assetIdx];
    return asset.name;
}
```

```
function setNameUsingStorage(Asset[] storage asset_to_change) internal {
    asset_to_change[0].name = "new_name_1";
}
```

```
function setNameUsingMemory(Asset[] memory asset_to_change) internal pure {
    asset_to_change[0].name = "new_name_2";
}
```

```
function setName() external {
    setNameUsingStorage(assets);
    setNameUsingMemory(assets);
}
```



Quanto vale asset[0].name?

::: Solidity (8/17)

Quale delle due è più costosa?

```
Asset[] public assets;
function getUsingStorage(uint _assetIdx) public returns (string memory) {
    Asset storage asset = assets[_assetIdx];
    return asset.name;
}

function getUsingMemory(uint _assetIdx) public returns (string memory) {
    Asset memory asset = assets[_assetIdx];
    return asset.name;
}

function setNameUsingStorage(Asset[] storage asset_to_change) internal {
    asset_to_change[0].name = "new_name_1";
}

function setNameUsingMemory(Asset[] memory asset_to_change) internal pure {
    asset_to_change[0].name = "new_name_2";
}

function setName() external {
    setNameUsingStorage(assets);
    setNameUsingMemory(assets);
}
```

→ Quanto vale asset[0].name?

Lo Storage è memorizzato nello stato di uno smart contract ed è persistente tra le chiamate di funzione.

::: Solidity (8/17)

Quale delle due è più costosa?

```
Asset[] public assets;
function getUsingStorage(uint _assetIdx) public
    returns (string memory) {
    Asset storage asset = assets[_assetIdx];
    return asset.name;
}
function getUsingMemory(uint _assetIdx) public
    returns (string memory) {
    Asset memory asset = assets[_assetIdx];
    return asset.name;
}
```

```
function setNameUsingStorage(Asset[] storage asset_to_change) internal {
    asset_to_change[0].name = "new_name_1";
}

function setNameUsingMemory(Asset[] memory asset_to_change) internal pure {
    asset_to_change[0].name = "new_name_2";
}
```

```
function setName() external {
    setNameUsingStorage(assets);
    setNameUsingMemory(assets);
}
```



Quanto vale asset[0].name?

Memory è per dati temporanei accessibili solo all'interno della funzione. Una volta che la funzione viene eseguita, il contenuto di memory viene scartato.

::: Solidity (9/17)

I dati possono essere copiati da una variabile all'altra in due modi. Un modo è copiare l'intero dato (copia per valore) e il secondo modo è copiare per riferimento. Esistono alcuni comportamenti predefiniti di copia in base al tipo di memorizzazione:

- L'assegnazione di una variabile di stato a un'altra variabile di stato crea una nuova copia.
- L'assegnazione alla variabile di stato storage da una variabile in memory crea sempre una nuova copia.
- L'assegnazione a una variabile in memory dalla variabile storage dello stato creerà una copia.
- L'assegnazione da una variabile in memory a un'altra variabile in memory non creerà una copia. Questo è applicabile solo alle variabili di tipo riferimento. La variabile locale crea ancora una nuova copia.

::: Solidity (10/17)

Tipi decimali e relative operazioni non sono pienamente supportati in Solidity. Per gli Ether, non è necessario utilizzare valori frazionari: tutti i valori sono rappresentati in wei, che è la più piccola unità di Ether.

- Se si vuole inviare 0,5 ether, si può invece specificare il tuo valore letterale come "500 finney", che verrà convertito in wei:

`msg.sender.send(500 finney);` o `msg.sender.send(1 etere / 2);`

che sono entrambi esattamente equivalenti a:

`msg.sender.send(500000000000000000000000);`

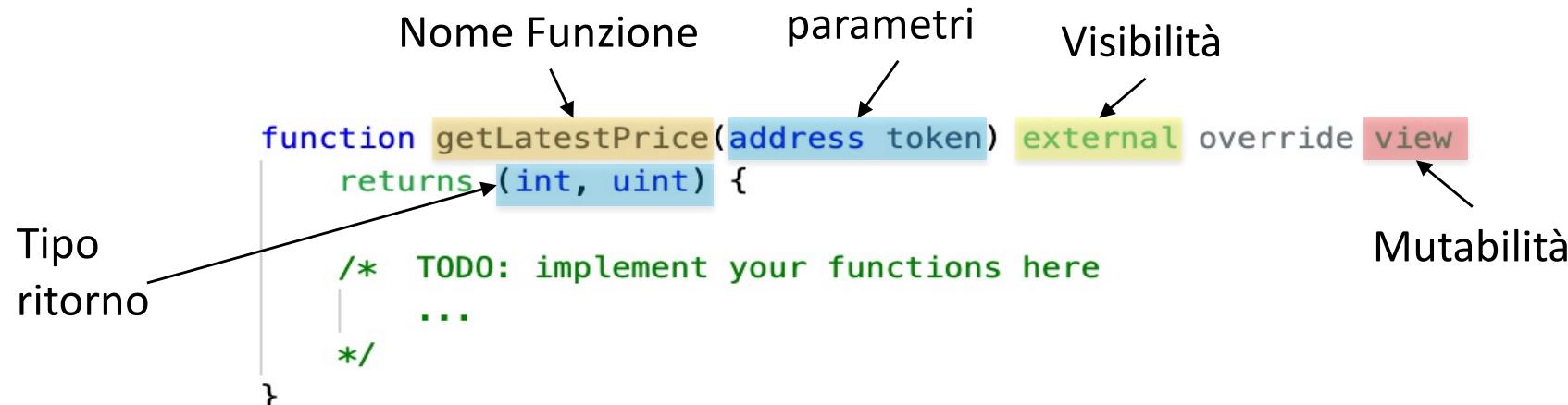
- Se si vuol moltiplicare un valore per una frazione (ad es. 2/3), si moltiplica prima per il numeratore, poi si divide per il denominatore:

`valore = (valore * 2) / 3;`

Vale anche la pena notare che il tipo decimale a virgola mobile per la matematica finanziaria non è ottimale perché introduce errori di arrotondamento che portano facilmente alla perdita di denaro.

::: Solidity (11/17)

Gli smart contract definiscono anche delle funzioni:



Possibili parametri di visibilità:

- Visibilità variabili di stato
 - Pubblico: consente ad altri contratti di leggerne i valori
 - Interno: è possibile accedere solo da contratti definiti/derivati
 - Privato: è possibile accedere solo da contratti definiti
- Visibilità della funzione
 - Esterno: può essere richiamato da altri contratti e tramite transazioni
 - Interno: accessibile solo dal contratto definito/derivato
 - Pubblico: esterno + interno
 - Privato: è possibile accedere solo a contratti definiti

::: Solidity (12/17)

Possibili parametri di mutabilità:

- Funzioni di visualizzazione
 - Impossibile modificare gli stati o chiamare altre funzioni non di visualizzazione

```
Asset[] public assets;
function getUsingStorage(uint _assetIdx) public view returns (string memory) {
    Asset storage asset = assets[_assetIdx];
    return asset.name;
}
```

- Funzioni pure
 - Impossibile leggere o modificare stati o chiamare altre funzioni non pure

```
function setNameUsingMemory(Asset[] memory asset_to_change) internal pure {
    asset_to_change[0].name = "new_name_2";
}
```

::: Solidity (13/17)

Pagabilità:

```
contract Payable {  
    address payable public owner;  
  
    constructor() payable {  
        owner = payable(msg.sender);  
    }  
  
    function deposit() public payable {  
        require(msg.value >= 1.0 ether);  
    }  
}
```

L'indirizzo può ricevere Ether

Alla creazione, il contratto può ricevere Ether

La chiamata implica un trasferimento di Ether

Ottenere la quantità di Ether associata alla transazione

Se qualcuno invia alcuni Ether allo smart contract e non ha una funzione pagabile, lo smart contract non accetterà Ether e la transazione fallirà.

::: Solidity (13/17)

Pagabilità:

```
contract Payable {  
    address payable public owner;  
  
    constructor() payable {  
        owner = payable(msg.sender);  
    }  
  
    function deposit() public payable {  
        require(msg.value >= 1.0 ether);  
    }  
  
    function() external payable{  
        //todo  
    }  
}
```

L'indirizzo può ricevere Ether

Alla creazione, il contratto può ricevere Ether

La chiamata implica un trasferimento di Ether

Ottenere la quantità di Ether associata alla transazione

Solidity supporta una singola funzione anonima esterna pagabile senza parametri chiamata funzione di fallback. Generalmente, una funzione di fallback viene utilizzata per ricevere Ether con un semplice trasferimento, quando qualcuno lo ha chiamato senza fornire alcun dato.

::: Solidity (14/17)

Il fallback si usa anche nel caso in cui il contratto ha ricevuto una transazione di invoke ma nessuna funzione corrisponde alla funzione chiamata. Questo duplice scopo confondeva e causava potenziali problemi di sicurezza, così è stata introdotta esplicitamente sia la funzione di fallback che una nuova di receive, dove la prima è optionalmente pagabile mentre la seconda lo è obbligatoriamente. La prima serve al caso di mancata corrispondenza tra chiamata e funzioni definite, mentre la seconda per il trasferimento di Ether.

```
receive() external payable{  
    // your code here...  
}
```

```
fallback() external [payable] {  
    // your code here...  
}
```

Solidity consente il concetto di sovraccarico di funzioni. Un contratto può avere più funzioni con lo stesso nome, ma il numero di parametri deve essere diverso. Tuttavia, non è possibile sovraccaricare una funzione con il suo tipo restituito, proprio come altri linguaggi di programmazione.

::: Solidity (15/17)

Solidity supporta sia il concetto di interfaccia che di contratto astratto, e anche un'apposita sintassi di implementazione e derivazione tra i contratti.

```
interface IPriceFeed {
    function getLatestPrice() external view returns (int price, uint lastUpdatedTime);
}

import "./interfaces/IPriceFeed.sol";

contract PriceFeed is IPriceFeed {
    function getLatestPrice(address token) external override view
        returns (int, uint) {

        /* TODO: implement your functions here
        ...
        */
    }
}
```

::: Solidity (15/17)

Solidity supporta sia il concetto di interfaccia che di contratto astratto, e anche un'apposita sintassi di implementazione e derivazione tra i contratti.

```
abstract contract Feline {
    function utterance() public pure virtual returns (bytes32);
}

contract Cat is Feline {
    function utterance() public pure override returns (bytes32) { return "meow"; }
}
```

```
abstract contract Feline {
    function utterance() public pure virtual returns (bytes32);
}

contract Cat is Feline {
    function utterance() public pure virtual override returns (bytes32) { return "meow"; }
}

contract Neko is Cat {
    function utterance() public pure override returns (bytes32) { return "nyan"; }
}
```

::: Solidity (16/17)

Solidity supporta anche l'override nelle gerarchie di contratti, come ogni altro linguaggio OO. Una funzione che consente a un contratto derivato di sovrascrivere il suo comportamento verrà contrassegnata come `virtual`. La funzione che esegue l'override di quella funzione di base deve essere contrassegnata come `override`.

- Se si dimentica di aggiungere l'override, il compilatore darà errore
- Se un contratto eredita la stessa funzione da più contratti di base (che non sono correlati), bisogna indicare esplicitamente quali contratti: `override(Base1, Base2)`

L'evento è un membro ereditabile di un contratto. Quando viene emesso un evento, si memorizzano gli argomenti passati nei registri delle transazioni, archiviati su blockchain e sono accessibili utilizzando l'indirizzo del contratto finché il contratto non è presente sulla blockchain.

::: Solidity (17/17)

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.

```
//Dichiara un evento
event Deposito(indirizzo indicizzato _from, bytes32 indicizzato _id, uint
_value);
```

```
//Emette un evento
emit Deposit(msg.sender, _id, msg.value);
```

```
var abi = /* abi as generated using compiler */;
var ClientReceipt = web3.eth.contract(abi);
var clientReceiptContract = ClientReceipt.at("0x1234...ab67" /* address */);

var event = clientReceiptContract.Deposit(function(error, result) {
  if (!error) console.log(result);
});
```

::: Solidity (17/17)

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`.

```
//Dichiara un even
event Deposito(in
    _value);
//Emette un event
emit Deposit(msg.
    )
```

{
 "returnValues": {
 "_from": "0x1111...FFFFCCCC",
 "_id": "0x50...sd5adb20",
 "_value": "0x420042"
 },
 "raw": {
 "data": "0x7f...91385",
 "topics": ["0xfd4...b4ead7", "0x7f...1a91385"]
 }
}

```
var abi = /* abi as generated using compiler */;
var ClientReceipt = web3.eth.contract(abi);
var clientReceiptContract = ClientReceipt.at("0x1234...ab67" /* address */);

var event = clientReceiptContract.Deposit(function(error, result) {
    if (!error) console.log(result);
});
```

::: OpenZeppelin

Open Zeppelin è una libreria per fornire un set di Smart Contracts Open Source da includere nei nostri per introdurre funzionalità di sicurezza e/o strutturare applicazioni ben note come NFT e token.

Consideriamo Ownable, un famoso pattern per gestire permessi e proprietà di uno Smart Contract:

```
import ".../Ownable.sol";  
contract SharedWallet is Ownable {  
    function isOwner() internal view returns(bool) {  
        return owner() == msg.sender;    }  
    function withdrawMoney(address payable _to, uint _amount)  
public onlyOwner {  
        _to.transfer(_amount);    }  
    receive() external payable { }  
}
```

::: OpenZeppelin

Open Zeppelin è una libreria per fornire un set di Smart Contracts Open Source da includere nei nostri per introdurre funzionalità di sicurezza e/o strutturare applicazioni ben note come NFT e token.

Consideriamo Ownable, un famoso pattern per gestire permessi e proprietà di uno Smart Contract:

```
import ".../Ownable.sol";
contract SharedWallet is Ownable {
    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;    }
    function withdrawMoney(address payable _to, uint _amount)
public onlyOwner {
        _to.transfer(_amount);    }
    receive() external payable { }
}
```

**Funzioni implementate nel
contratto base**

**Nuovi modificatori definiti nel
contratto base**

**Variabili di stato definite nel
contratto base**

::: Tokens, NFT e Smart Contracts (1/7)

Le blockchain sono le soluzioni tecnologiche che abilitano i digital assets. Un asset digitale viene creato, o coniato, quando nuove informazioni vengono aggiunte a una particolare blockchain. Attraverso blockchain, gli utenti possono scambiare asset digitali esistenti e/o crearne di nuovi (mint).

I "beni digitali" rappresentano un ampio contenitore che comprende qualsiasi cosa coniata e scambiata su una blockchain.

- Risorse crittografiche - Qualsiasi bene digitale di valore o mezzo di scambio (valuta) memorizzata sulla blockchain.
- Stablecoins - Un tipo di criptovaluta progettato per price stability e collegati a valute fiat, materie prime o altre criptovalute.
- Token non fungibili (NFT) - Un token che rappresenta la proprietà di un oggetto digitale univoco e che certifica che il detentore possiede l'asset digitale sottostante e può venderlo, scambiarlo o riscattarlo.

::: Tokens, NFT e Smart Contracts (2/7)

- Valute digitali della banca centrale (CBDC) - Rappresenta la valuta fiat di una nazione ed è sostenuta dalla sua banca centrale.
- Token di sicurezza - Risorse digitali che soddisfano la definizione di sicurezza o investimento finanziario, come azioni e obbligazioni.

La parola fungibile è definito come "qualcosa" (ad es. denaro o merce) di natura tale che un'altra parte o quantità uguale può sostituire una parte o quantità nel pagamento di un debito o nel saldare un conto".

- Le valute sono fungibili, cioè quando una persona dà a un'altra persona una banconota da cinque dollari, l'altra persona può restituire alla persona originaria una banconota da cinque dollari diversa. Finché le banconote sono autentiche, sono scambiabili e hanno lo stesso valore.

Lo standard ERC-20 sulla piattaforma Ethereum specifica un'interfaccia comune per i token fungibili che sono divisibili e non distinguibili. OpenZeppelin offre tale interfaccia.

::: Tokens, NFT e Smart Contracts (3/7)

```
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns (uint256
balance)
function transfer(address _to, uint256 _value) public returns
(bool success)
function transferFrom(address _from, address _to, uint256 _value)
public returns (bool success)
function approve(address _spender, uint256 _value) public returns
(bool success)
function allowance(address _owner, address _spender) public view
returns (uint256 remaining)
```

<https://eips.ethereum.org/EIPS/eip-20>

::: Tokens, NFT e Smart Contracts (3/7)

```
contract ERC20 is Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private _balances;

    mapping(address => mapping(address => uint256)) private _allowances;

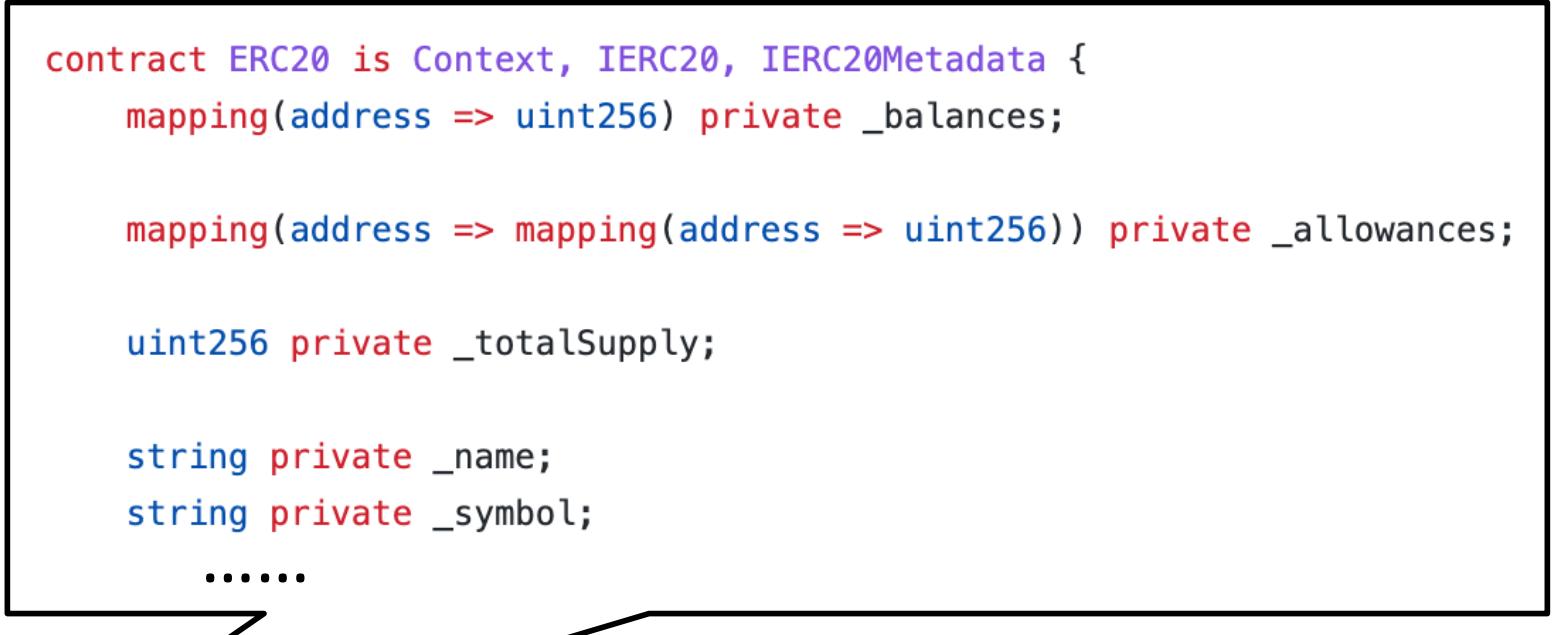
    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
    .....

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract EUSD is ERC20 {

    constructor() ERC20("Ethereum USD", "EUSD") {
        _mint(msg.sender, 100000000 * (10 ** uint256(decimals())));
    }
}
```



The diagram illustrates the inheritance relationship between the EUSD contract and the ERC20 base contract. An arrow points from the import statement in the EUSD code to the callout box containing the ERC20 contract definition.

::: Tokens, NFT e Smart Contracts (3/7)

```
contract ERC20 is Context, IERC20, IERC20Metadata {  
    mapping(address => uint256) private _balances;  
  
    mapping(address => mapping(address => uint256)) private _allowances;  
  
    uint256 private _totalSupply;  
  
    string private _name;  
    string private _symbol;  
    .....  
}
```

```
import "@openzeppelin/cont  
contract EUSD is ERC20 {  
  
    constructor() ERC20("Ethereum USD", "EUSD") {  
        _mint(msg.sender, 100000000 * (10 ** uint256(decimals())));  
    }  
}
```

Invocazione del costruttore nel contratto base.

::: Tokens, NFT e Smart Contracts (3/7)

```
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply += amount;
    _balances[account] += amount;
    emit Transfer(address(0), account, amount);

    _afterTokenTransfer(address(0), account, amount);
}
```

::: Tokens, NFT e Smart Contracts (3/7)

```
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply += amount;
    _balances[account] += amount;
    emit Transfer(address(0), account, amount);

    _afterTokenTransfer(address(0), account, amount);
}
```

```
function _transfer(
    address from,
    address to,
    uint256 amount
) internal virtual {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");

    _beforeTokenTransfer(from, to, amount);

    uint256 fromBalance = _balances[from];
    require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");
    unchecked {
        _balances[from] = fromBalance - amount;
    }
    _balances[to] += amount;

    emit Transfer(from, to, amount);

    _afterTokenTransfer(from, to, amount);
}
```

::: Tokens, NFT e Smart Contracts (3/7)

```
function mint(address account, uint256 amount) internal virtual {
```

Il comportamento "controllato" predefinito costa più gas durante il calcolo, perché quei controlli sono implementati come una serie di codici operativi che, prima di eseguire l'aritmetica effettiva, controllano l'under/overflow e si ripristinano se viene rilevato. Tali controlli consumano gas. Se si ha certezza che la propria aritmetica non possa mai causare under/overflow, si può far a meno di tali controlli e salvare gas, racchiudendo l'aritmetica in un blocco "non controllato".

```
}
```

```
) internal vir
    require(fr
    require(to
        _beforeTokenTransfer(fr
            to, amount);

        uint256 f
            fromBalance = _balances[from];
        require(f
            balance >= amount, "ERC20: transfer amount exceeds balance");
        unchecked {
            _balances[from] = fromBalance - amount;
        }
        _balances[to] += amount;

        emit Transfer(from, to, amount);

        _afterTokenTransfer(from, to, amount);
    }
```

::: Tokens, NFT e Smart Contracts (3/7)

```
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply += amount;
    _balances[account] += amount;
    emit Transfer(address(0), account, amount);

    _afterTokenTransfer(address(0), account, amount);
}
```

```
function _transfer(
    address from,
    address to,
    uint256 amount
) internal virtual {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");

    _beforeTokenTransfer(from, to, amount);
```

```
function _approve(
    address owner,
    address spender,
    uint256 amount
) internal virtual {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}
```

```
    uint256 fromBalance = _balances[from];
    require(balance >= amount, "ERC20: transfer amount exceeds balance");

    [from] = fromBalance - amount;
    += amount;

    [from, to, amount];
    transfer(from, to, amount);
```

::: Tokens, NFT e Smart Contracts (3/7)

```
function _spendAllowance(
    address owner,
    address spender,
    uint256 amount
) internal virtual {
    uint256 currentAllowance = allowance(owner, spender);
    if (currentAllowance != type(uint256).max) {
        require(currentAllowance >= amount, "ERC20: insufficient allowance");
        unchecked {
            _approve(owner, spender, currentAllowance - amount);
        }
    }
}
```

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
    address spender = _msgSender();
    _spendAllowance(from, spender, amount);
    _transfer(from, to, amount);
    return true;
}
```

::: Tokens, NFT e Smart Contracts (4/7)

I token fungibili non possono rappresentare digitalmente l'unicità. Al contrario, i token non fungibili (NFT) differiscono da quelli fungibili (come criptovalute come Bitcoin) in due aspetti importanti: ogni NFT è unico e non può essere diviso o unito. Affinché un token possa essere classificato come NFT, deve seguire quattro regole principali:

- Non può essere replicato.
- Non può essere contraffatto.
- Non può essere creato su richiesta.
- Possiede gli stessi diritti di proprietà e garanzie di permanenza di Bitcoin.

Ogni NFT rappresenta qualcosa di unico, autentico e distintivo come un oggetto da collezione. Hanno valori intrinsecamente diversi e non sono scambiabili. Inoltre, nel caso di risorse digitali, un creatore può anche guadagnare royalties ogni volta che l'NFT viene venduto o scambiato.

::: Tokens, NFT e Smart Contracts (5/7)

Questa nuova forma di token è stata introdotta per la prima volta con lo standard ERC-721 alla fine del 2017 e ulteriormente sviluppata con ERC-1155 come interfaccia standard per i contratti che gestiscono più tipi di token.

- ERC-721 varia notevolmente dallo standard ERC-20 poiché estende l'interfaccia comune per i token con funzioni aggiuntive per garantire che i token basati su di esso siano chiaramente non fungibili e quindi unici.

La negoziabilità completa, la liquidità profonda e l'interoperabilità conveniente consentono a NFT di diventare una soluzione promettente per la protezione della proprietà intellettuale (IP).

::: Tokens, NFT e Smart Contracts (6/7)

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

<https://eips.ethereum.org/EIPS/eip-721>

::: Tokens, NFT e Smart Contracts (7/7)

```
contract GameItem is ERC721URIStorage {  
    using Counters for Counters.Counter;  
    Counters.Counter private _tokenIds;  
  
    constructor() ERC721("GameItem", "ITM") {}  
  
    function awardItem(address player, string memory tokenURI)  
        public  
        returns (uint256)  
    {  
        _tokenIds.increment(); // Genera un id univoco globale  
  
        uint256 newItemId = _tokenIds.current();  
        _mint(player, newItemId);  
        _setTokenURI(newItemId, tokenURI); // Punta a un file JSON che descrive gli attributi del token (nome, descrizione, immagine...)  
        return newItemId;  
    }  
}
```

::: Tokens, NFT e Smart Contracts (7/7)

```
function _transfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual {
    require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer from incorrect owner");
    require(to != address(0), "ERC721: transfer to the zero address");

    _beforeTokenTransfer(from, to, tokenId);

    // Clear approvals from the previous owner
    _approve(address(0), tokenId);

    _balances[from] -= 1;
    _balances[to] += 1;
    _owners[tokenId] = to;

    emit Transfer(from, to, tokenId);

    _afterTokenTransfer(from, to, tokenId);
}
```

::: Tokens, NFT e Smart Contracts (7/7)

```
function _transfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual {
    require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer from incorrect owner");
    require(to != address(0), "ERC721: transfer to the zero address");

    _beforeTokenTransfer(from, to, tokenId);

    // Clear approvals from the previous owner
    _approve(address(0), tokenId);

    _balance[from] = _balance[from] - 1;
    _balance[to] = _balance[to] + 1;
    _owner[tokenId] = to;
    emit Transfer(from, to, tokenId);
}

function _safeTransfer(
    address from,
    address to,
    uint256 tokenId,
    bytes memory _data
) internal virtual {
    _transfer(from, to, tokenId);
    require(_checkOnERC721Received(from, to, tokenId, _data), "ERC721: transfer to non ERC721Receiver implementer");
}
```

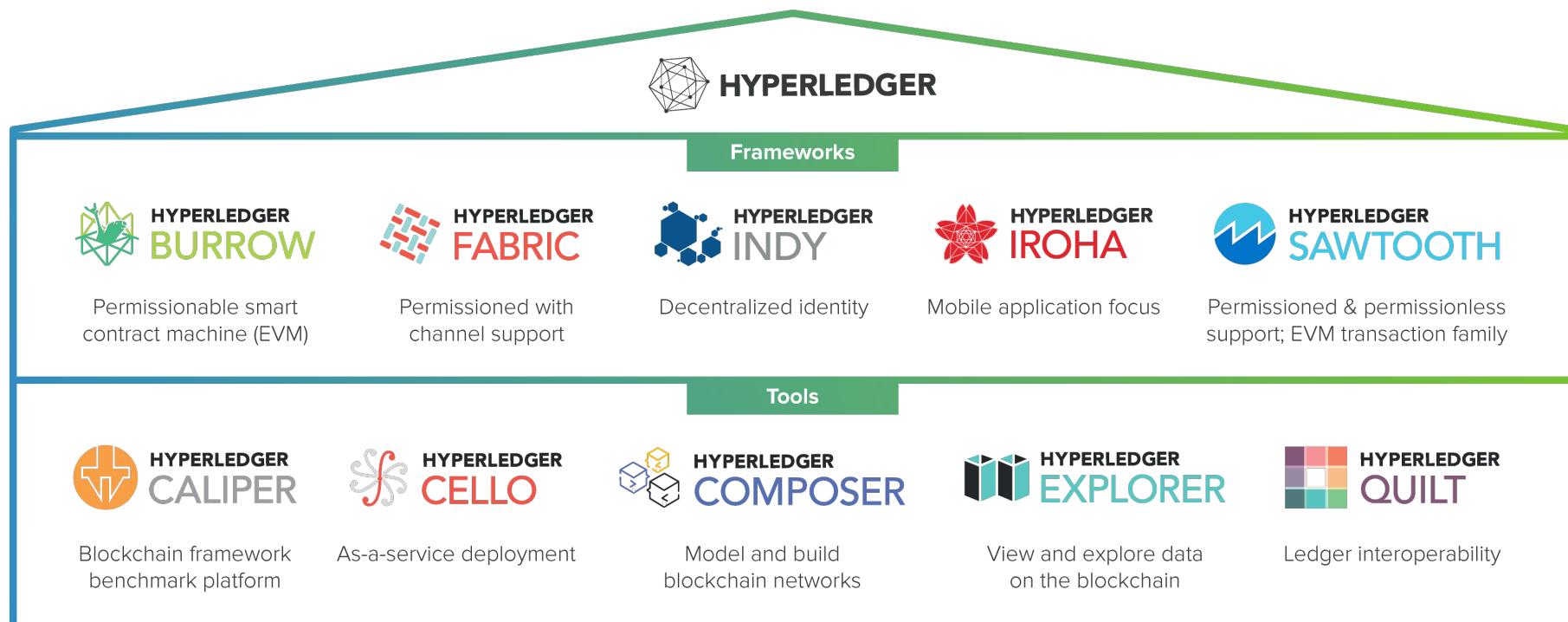
```
if (to.isContract()) {
    try IERC721Receiver(to).onERC721Received(_msgSender(), from, tokenId, _data) returns (bytes4 retval) {
        return retval == IERC721Receiver.onERC721Received.selector;
    } catch (bytes memory reason) {
```



Principali Piattaforme – Hyperledger Fabric & Go

::: Hyperledger Fabric (1/16)

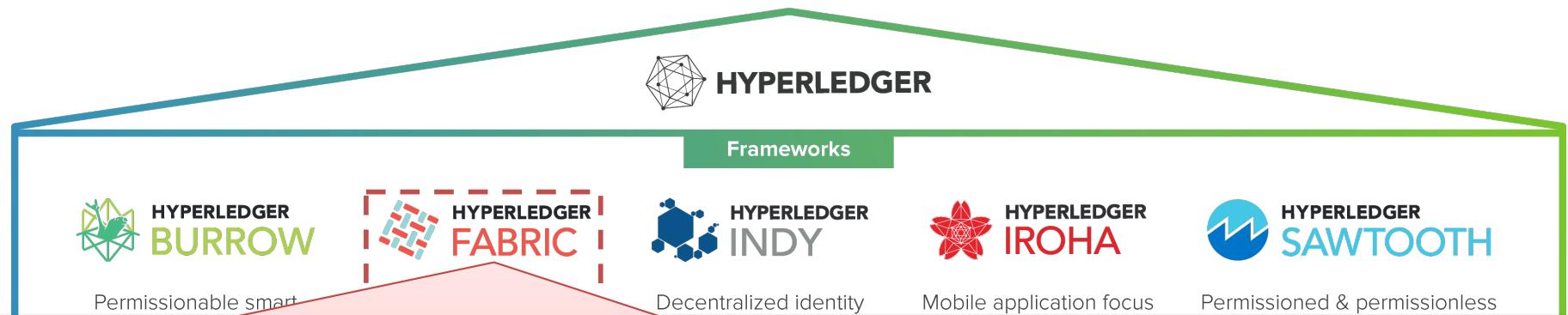
Hyperledger è una famiglia di progetti di blockchain open source avviato a dicembre 2015 dalla Fondazione Linux, per supportare lo sviluppo collaborativo di soluzioni DLT.



Si suddivide in framework per la realizzazione di piattaforme blockchain, e strumenti a supporto di applicazioni e smart contracts.

::: Hyperledger Fabric (1/16)

Hyperledger è una famiglia di progetti di blockchain open source avviato a dicembre 2015 dalla Fondazione Linux, per supportare lo sviluppo collaborativo di soluzioni DLT.



Hyperledger Fabric è un progetto di IBM per imprese, e consiste in una piattaforma di DLT permissioned, con alcune differenze chiave rispetto ad altri popolari soluzioni:

- ha un'architettura altamente modulare e configurabile;
- supporta smart contracts (detto Chaincode) scritti in linguaggi di programmazione comuni, piuttosto che utilizzare linguaggi specifici del dominio;
- i partecipanti sono noti, invece che essere anonimi, adottando un modello di governance costruito sulla fiducia che c'è tra i partecipanti;
- supporto per protocolli di consenso innestabili, da scegliere sulla base delle esigenze, e senza una criptovaluta nativa per incentivare costosi mining o per alimentare l'esecuzione di contratti intelligenti.

::: Hyperledger Fabric (2/16)

Le organizzazioni che prendono parte alla costruzione della rete Hyperledger Fabric sono chiamate «membri», ognuna responsabile di impostare i propri peer per la partecipazione alla rete. Questi peer devono essere configurati con materiali crittografici appropriati per autenticare la propria identità o proteggere i canali di comunicazione.

- I peer ricevono richieste di transazioni dai client mediante un SDK o servizi Web REST per interagire con la rete Hyperledger Fabric, e sono connessi tra loro da canali che consentono l'isolamento dei dati e la riservatezza.
- Tutti i peer mantengono il loro registro unico per canale a cui sono iscritti, ma a differenza di Ethereum, hanno ruoli diversi:
 1. **Endorser peer**: ricevuta la richiesta da un client, convalida la transazione ed esegue il Chaincode simulando l'esito della transazione senza aggiornare la blockchain. Al termine, l'Endorser può approvare o disapprovare la transazione.

::: Hyperledger Fabric (2/16)

Le organizzazioni che prendono parte alla costruzione della rete Hyperledger Fabric sono chiamate «membri», ognuna responsabile di impostare i propri peer per la partecipazione alla rete. Questi peer devono essere configurati con materiali crittografici appropriati per autenticare la propria identità o proteggere i canali di comunicazione.

- I peer ricevono richieste di transazioni dai client mediante un SDK o servizi Web REST per interagire con la rete Hyperledger Fabric, e sono connessi tra loro da canali che consentono l'isolamento dei dati e la riservatezza.
- Tutti i peer mantengono i dati della blockchain, ma non sono tutti iscritti, ma a differenza di Ethereum, i ruoli sono diversi:
 1. **Endorser peer:** ricevuta la richiesta da un client, convalida la transazione ed esegue il Chaincode simulando l'esito della transazione senza aggiornare la blockchain. Al termine, l'Endorser può approvare o disapprovare la transazione.

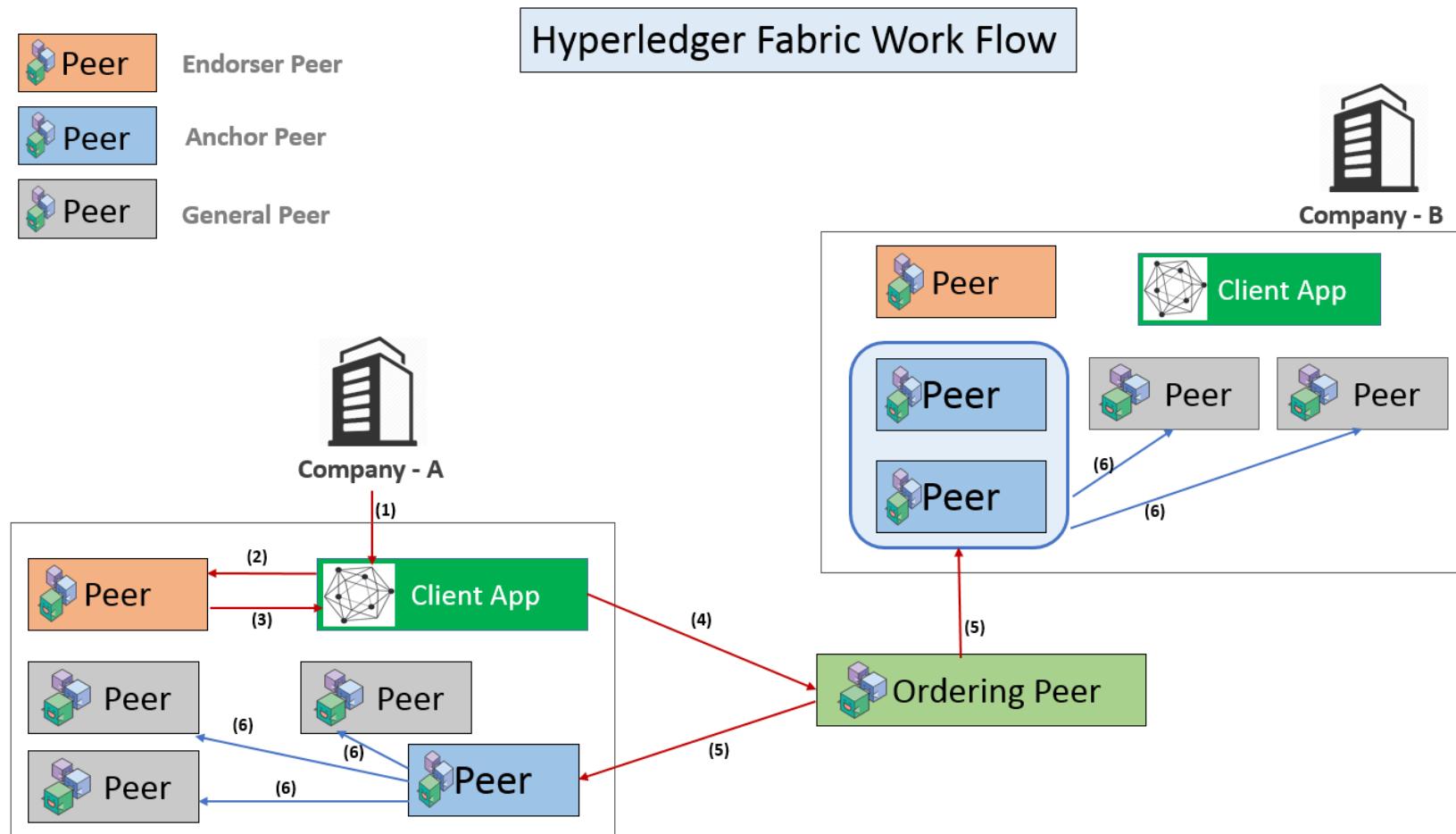
Solo il nodo Endorser esegue il Chaincode, quindi non è necessario installarlo in ogni nodo della rete.

::: Hyperledger Fabric (3/16)

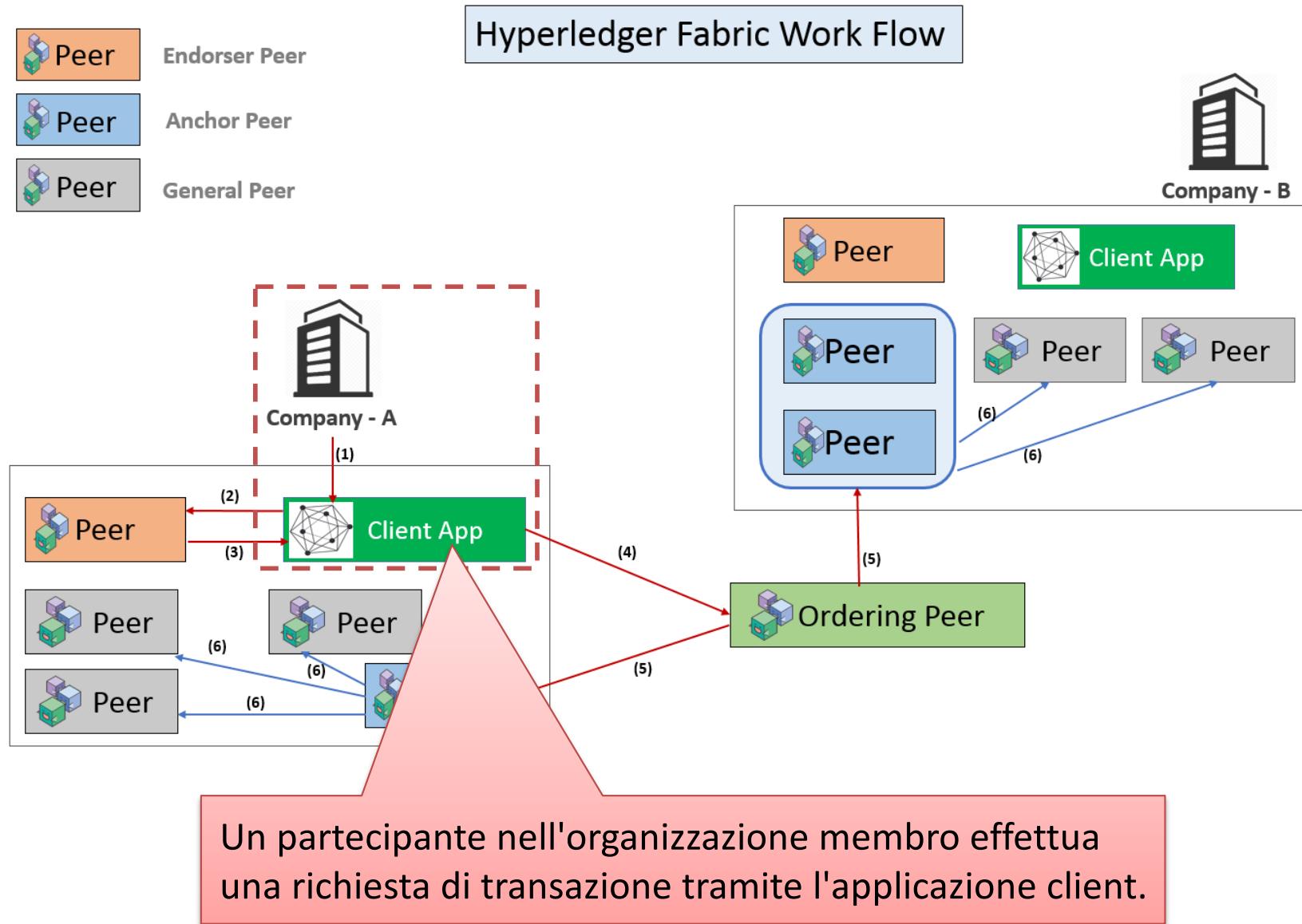
2. **Anchor peer:** riceve gli aggiornamenti e trasmette gli aggiornamenti agli altri peer nell'organizzazione. È possibile configurare canali segreti tra i peer e le transazioni tra i peer di quel canale sono visibili solo ai partecipanti.
3. **Orderer peer:** rappresenta l'elemento centrale di un canale tra peer ed è responsabile dello stato del registro coerente in tutta la rete ordinando le transazioni e collocandole in nuovi blocchi. Sono attualmente disponibili due opzioni:
 - Solo: un singolo orderer da usare per lo sviluppo, non in produzione, perché poco scalabile e resiliente.
 - Kafka: soluzione di streaming processing di Apache che garantisce la ricezione di messaggi nell'ordine di invio.

Si possono avere due tipi di transazioni: Deploy transactions per creare un nuovo chaincode ed installarlo sulla Blockchain, ed Invoke transactions per richiamare la funzioni di un chaincode.

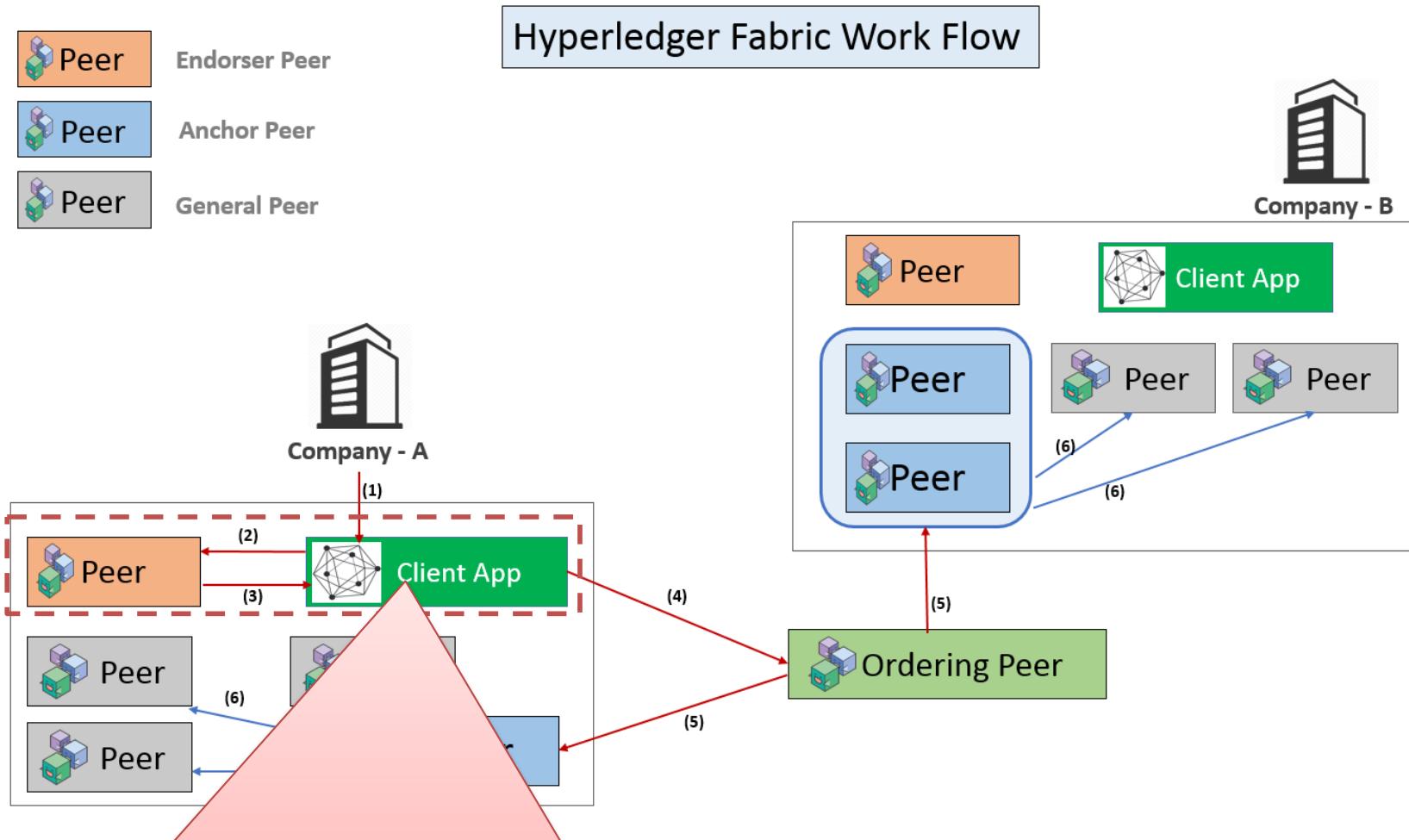
::: Hyperledger Fabric (4/16)



::: Hyperledger Fabric (4/16)

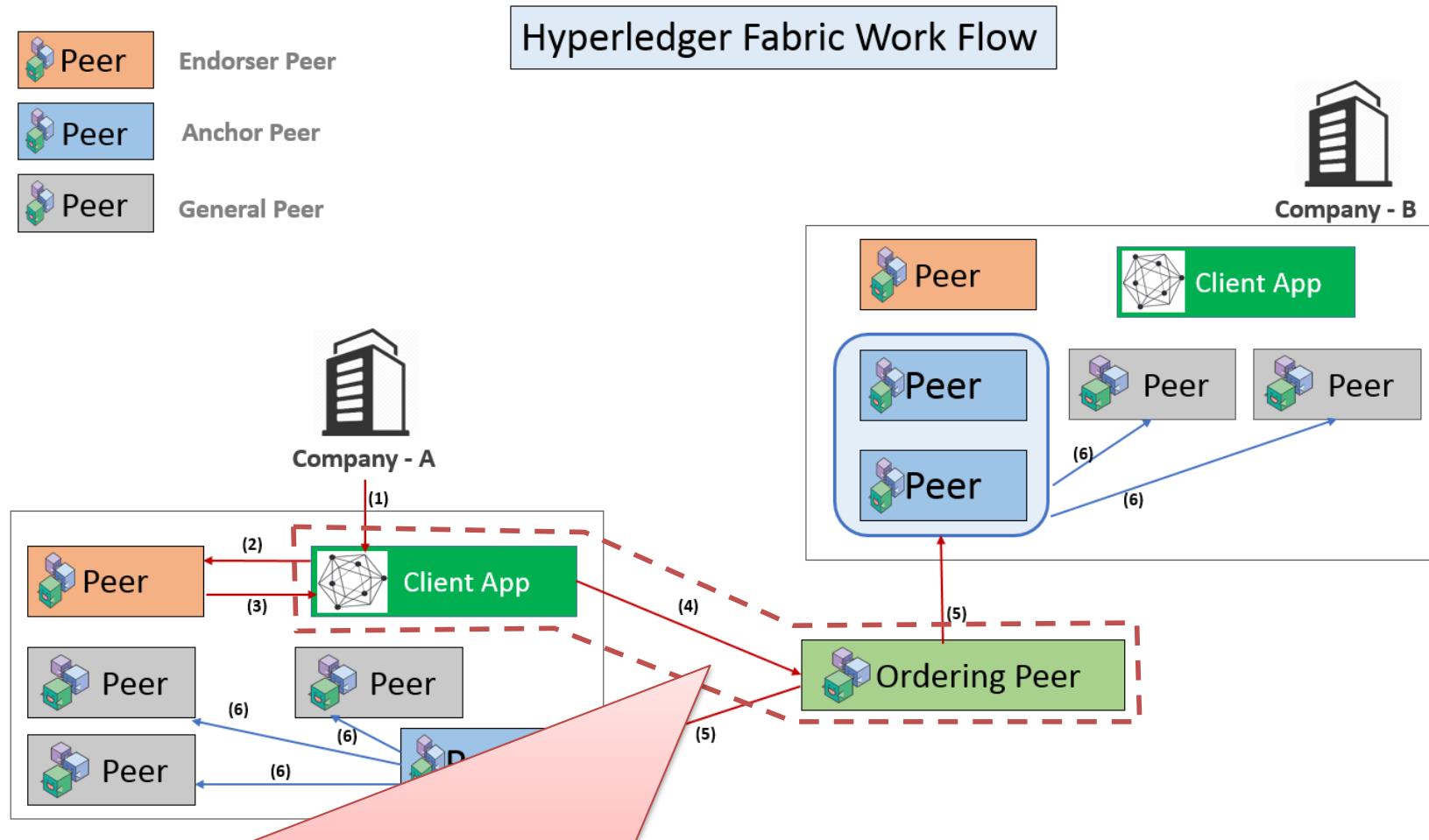


::: Hyperledger Fabric (4/16)



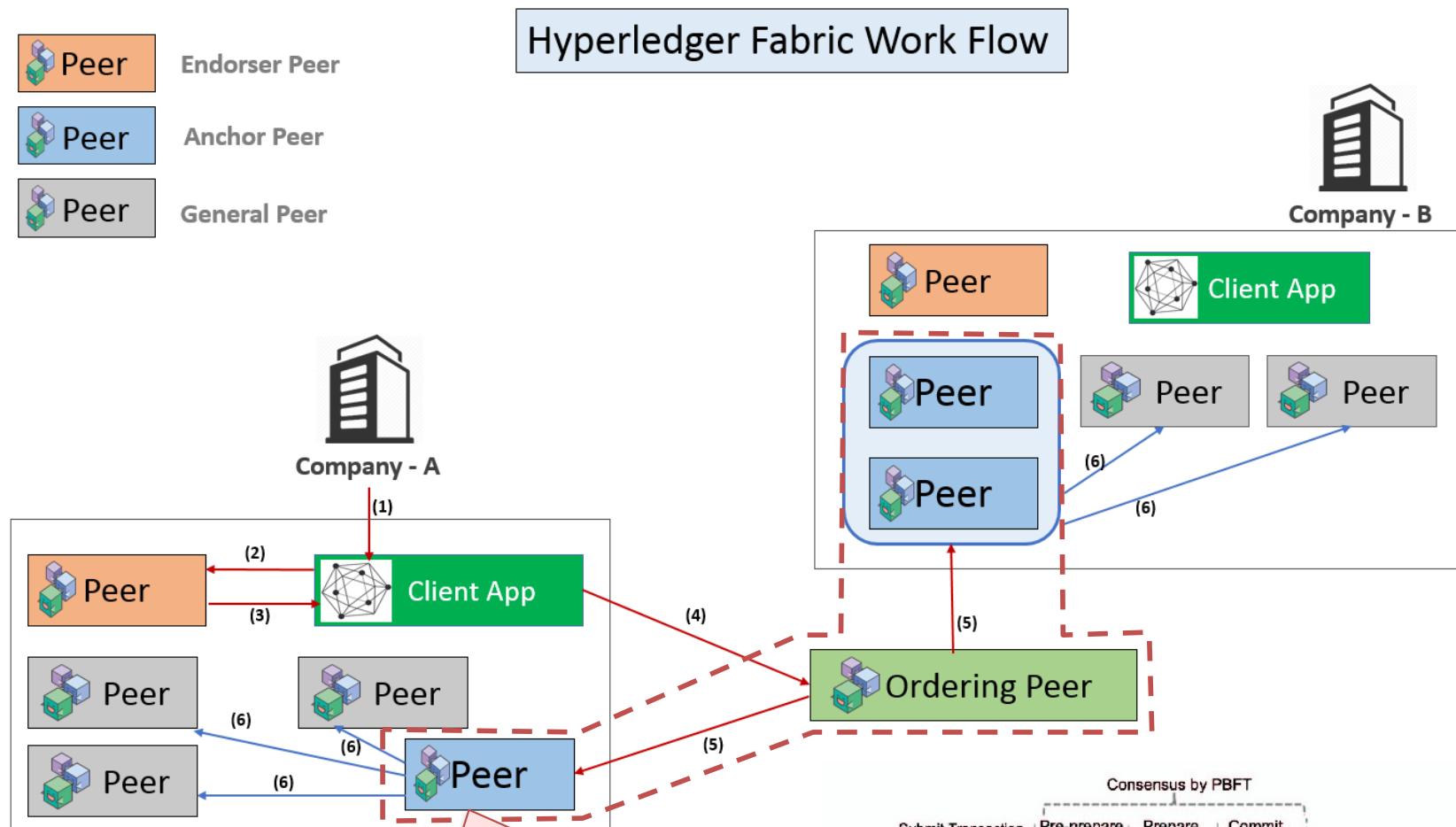
L'applicazione client trasmette la richiesta al Endorser peer, che controlla i dettagli del certificato e altri dettagli per convalidare la transazione ed eventualmente esegue il Chaincode e restituisce le risposte di Endorsement, accettando o rifiutando la transazione.

::: Hyperledger Fabric (4/16)

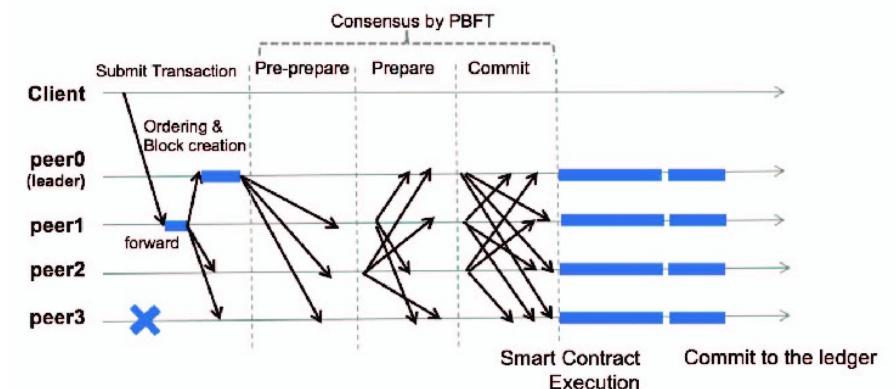


Il client invia la transazione approvata all'Orderer peer, affinché questa venga ordinata correttamente rispetto ad altre transazioni ed inclusa in un blocco.

::: Hyperledger Fabric (4/16)



Il nodo Orderer include la transazione in un blocco, e inoltra il blocco ai nodi Anchor di diverse Organizzazioni membri della rete, che eseguono un consenso distribuito PBFT.



::: Hyperledger Fabric (4/16)

COMPARISON OF CONSENSUS ALGORITHMS USED IN HYPERLEDGER FRAMEWORKS				
	Consensus Algorithm	Consensus Approach	Pros	Cons
Endorsement Peer	Kafka in Hyperledger Fabric Ordering Service	Permissioned voting-based. Leader does ordering. Only in-sync replicas can be voted as leader. ("Kafka," 2017).	Provides crash fault tolerance. Finality happens in a matter of seconds.	While Kafka is crash fault tolerant, it is not Byzantine fault tolerant, which prevents the system from reaching agreement in the case of malicious or faulty nodes.
Anchor Peer	RBFT in Hyperledger Indy	Pluggable election strategy set to a permissioned, voting-based strategy by default ("Plenum," 2016). All instances do ordering, but only the requests ordered by the master instance are actually executed. (Aublin, Mokhtar & Quéma, 2013)	Provides Byzantine fault tolerance. Finality happens in a matter of seconds.	The more nodes that exist on the network, the more time it takes to reach consensus. The nodes in the network are known and must be totally connected.
Genesis Peer	Sumeragi in Hyperledger Iroha	Permissioned server reputation system.	Provides Byzantine fault tolerance. Finality happens in a matter of seconds. Scale to petabytes of data, distributed across many clusters (Struckhoff, 2016).	The more nodes that exist on the network, the more time it takes to reach consensus. The nodes in the network are known and must be totally connected.
	PoET in Hyperledger Sawtooth	Pluggable election strategy set to a permissioned, lottery-based strategy by default.	Provides scalability and Byzantine fault tolerance.	Finality can be delayed due to forks that must be resolved.

3

```

graph TD
    Peer1[Peer] --> OS[Kafka in Hyperledger Fabric Ordering Service]
    Peer2[Peer] --> OS
    Peer3[Peer] --> OS
    OS --> Ledger[Commit to the ledger]
    subgraph "Execution"
        direction TB
        P1[Peer] --> S1[Peer]
        S1 --> S2[Peer]
        S2 --> S3[Peer]
    end

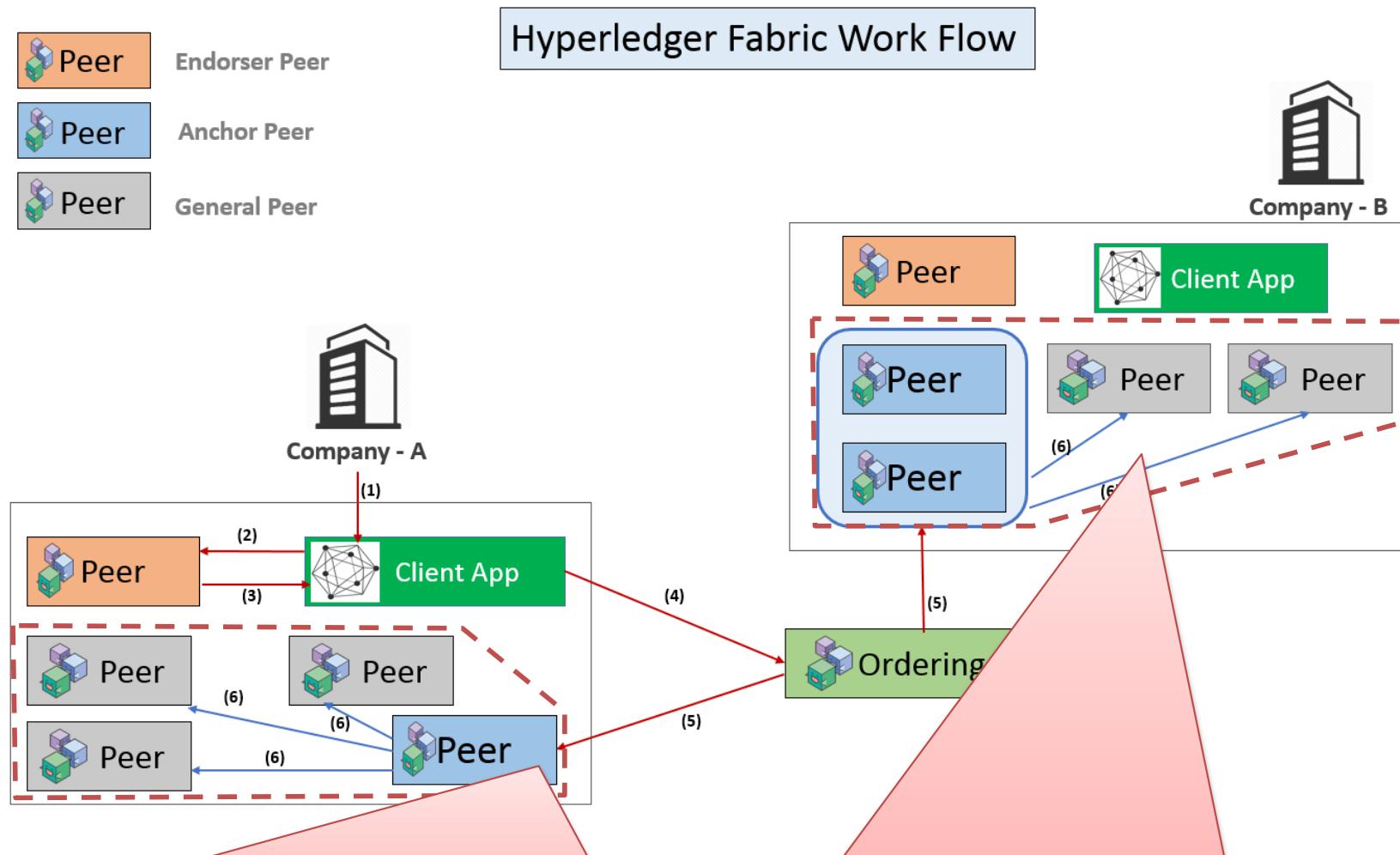
```

Il nodo Ordine blocco, e inoltre diverse Organizzazioni eseguono

Execution

Commit to the ledger

::: Hyperledger Fabric (4/16)



Al raggiungimento del consenso, gli Anchor peer trasmettono il blocco agli altri peer all'interno della propria organizzazione, così che questi aggiornano il proprio registro locale con l'ultimo blocco. Così tutta la rete ottiene il registro sincronizzato.

::: Hyperledger Fabric (5/16)

Una Certificate Authority (CA) emette i certificati per consentire

- alle organizzazioni di autenticarsi sulla rete;
- alle applicazioni client di autenticare le proposte di transazione ;
- ai peer di approvare le proposte e caricare le transazioni nel libro mastro se valide.

Possono esserci una o più CA sulla rete e le organizzazioni possono scegliere di utilizzare una propria CA.

::: Hyperledger Fabric (5/16)

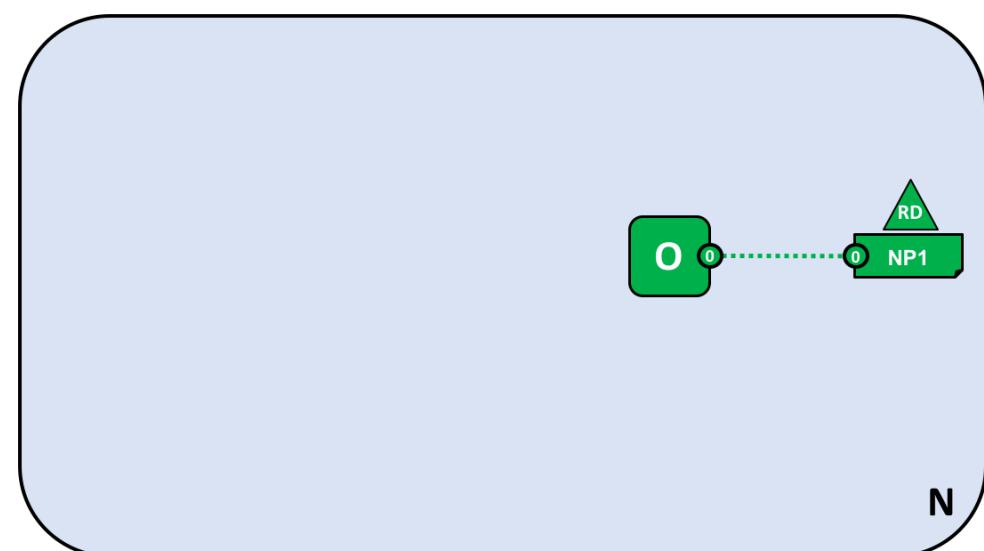
Una Certificate Authority (CA) emette i certificati per consentire

- alle organizzazioni di autenticarsi sulla rete;
- alle applicazioni client di autenticare le proposte di transazione ;
- ai peer di approvare le proposte e caricare le transazioni nel libro mastro se valide.

Possono esserci una o più CA sulla rete e le organizzazioni possono scegliere di utilizzare una propria CA.

La rete viene creata dalla definizione del servizio di ordinazione con la configurazione per i canali all'interno della rete, includendo le politiche di accesso e le informazioni sull'appartenenza (come certificati radice X509) per ogni membro del canale.

CA4



N

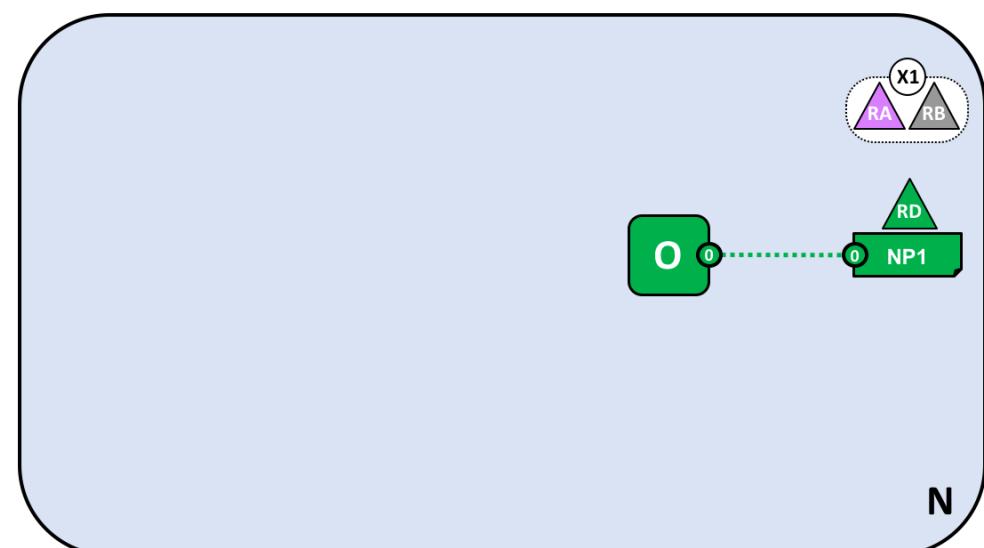
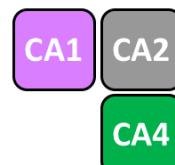
::: Hyperledger Fabric (5/16)

Una Certificate Authority (CA) emette i certificati per consentire

- alle organizzazioni di autenticarsi sulla rete;
- alle applicazioni client di autenticare le proposte di transazione ;
- ai peer di approvare le proposte e caricare le transazioni nel libro mastro se valide.

Possono esserci una o più CA sulla rete e le organizzazioni possono scegliere di utilizzare una propria CA.

Il consorzio alla base della rete viene istaurato definendo i certificati delle organizzazioni membro e le relative CA



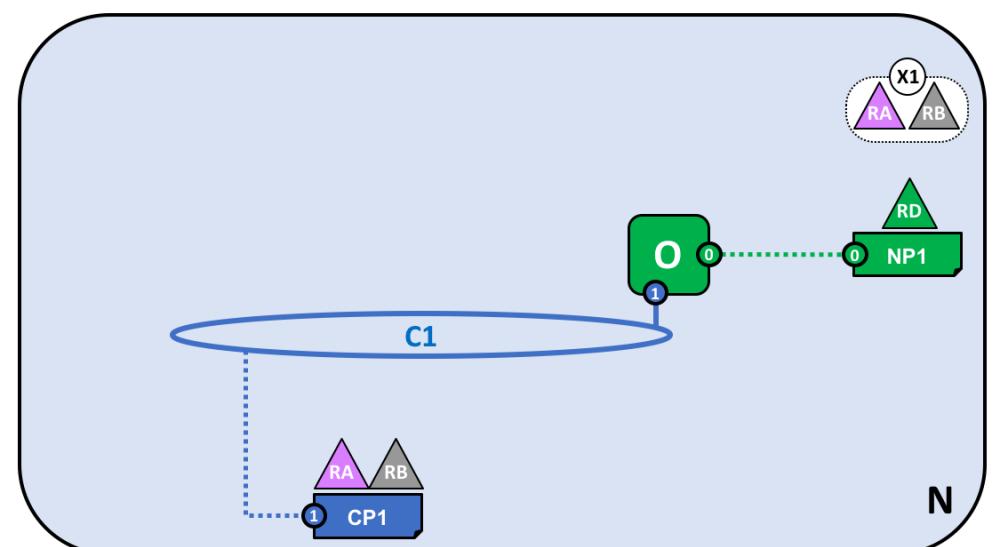
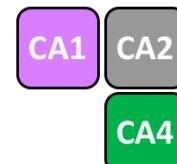
::: Hyperledger Fabric (5/16)

Una Certificate Authority (CA) emette i certificati per consentire

- alle organizzazioni di autenticarsi sulla rete;
- alle applicazioni client di autenticare le proposte di transazione ;
- ai peer di approvare le proposte e caricare le transazioni nel libro mastro se valide.

Possono esserci una o più CA sulla rete e le organizzazioni possono scegliere di utilizzare una propria CA.

Un canale C1 è creato generando il blocco di configurazione sul servizio di ordinazione, che valuta la validità della configurazione del canale. L'uso dei canali sono regolati dai criteri con cui sono configurati.



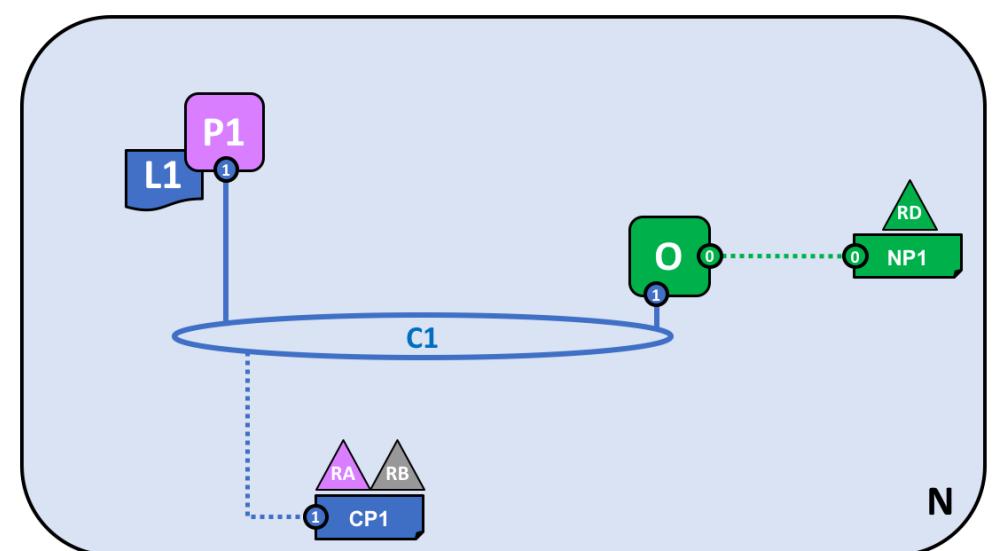
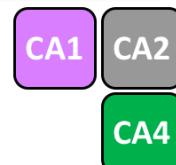
::: Hyperledger Fabric (5/16)

Una Certificate Authority (CA) emette i certificati per consentire

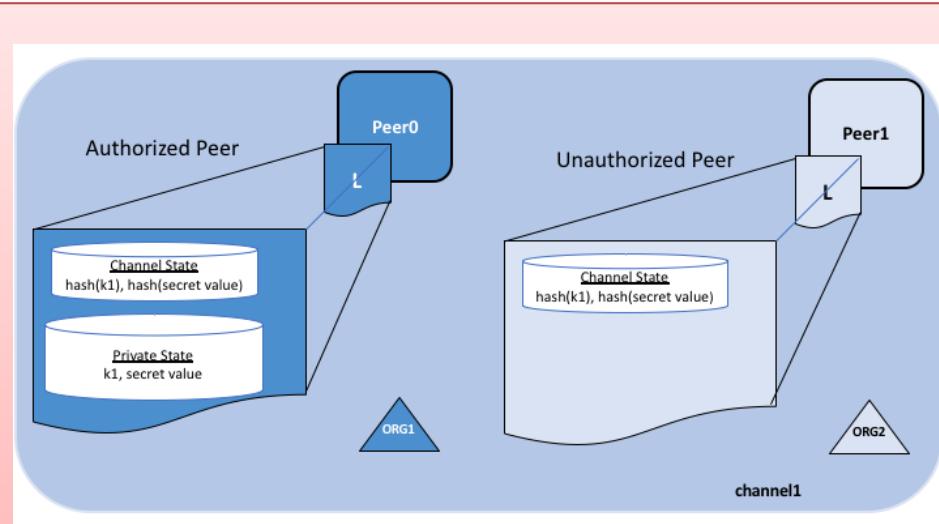
- alle organizzazioni di autenticarsi sulla rete;
- alle applicazioni client di autenticare le proposte di transazione ;
- ai peer di approvare le proposte e caricare le transazioni nel libro mastro se valide.

Possono esserci una o più CA sulla rete e le organizzazioni possono scegliere di utilizzare una propria CA.

I peer vengono uniti ai canali dalle organizzazioni che li possiedono e possono esserci più nodi peer sui canali all'interno della rete. P1 è il peer che mantiene copia dello stato della blockchain L1 per le transazioni su C1



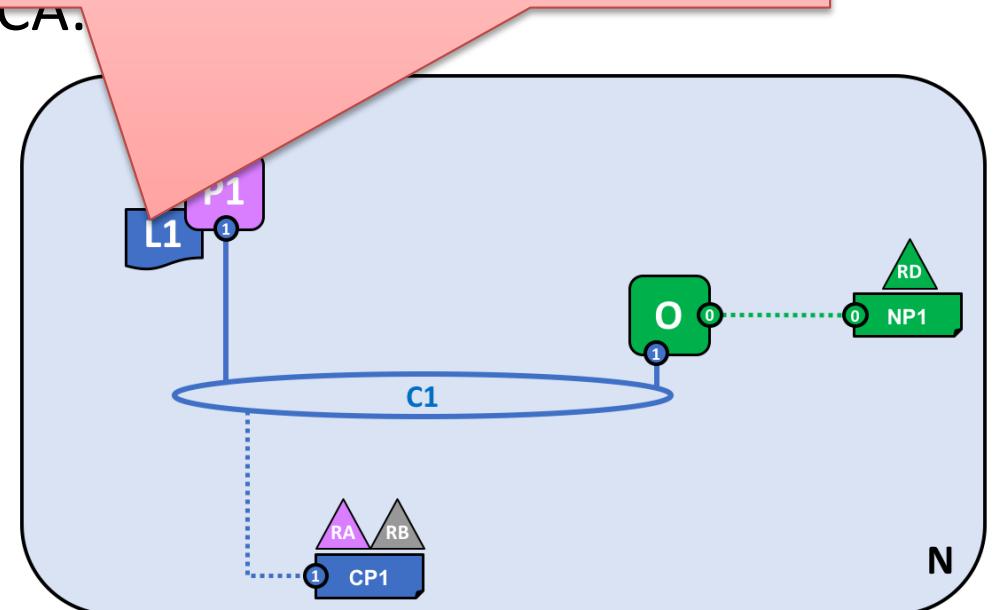
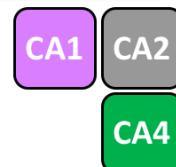
::: Hyperledger Fabric (5/16)



Il libro mastro contiene anche delle informazioni private, accessibili sono ad alcune delle organizzazioni sul canale. Sono contenute in un SideDB e seguono lo stesso processo di endorsement e commit dei dati pubblici, ma la blockchain contiene solo il suo hash.

scegliere di utilizzare una propria CA.

I peer vengono uniti ai canali dalle organizzazioni che li possiedono e possono esserci più nodi peer sui canali all'interno della rete. P1 è il peer che mantiene copia dello stato della blockchain L1 per le transazioni su C1



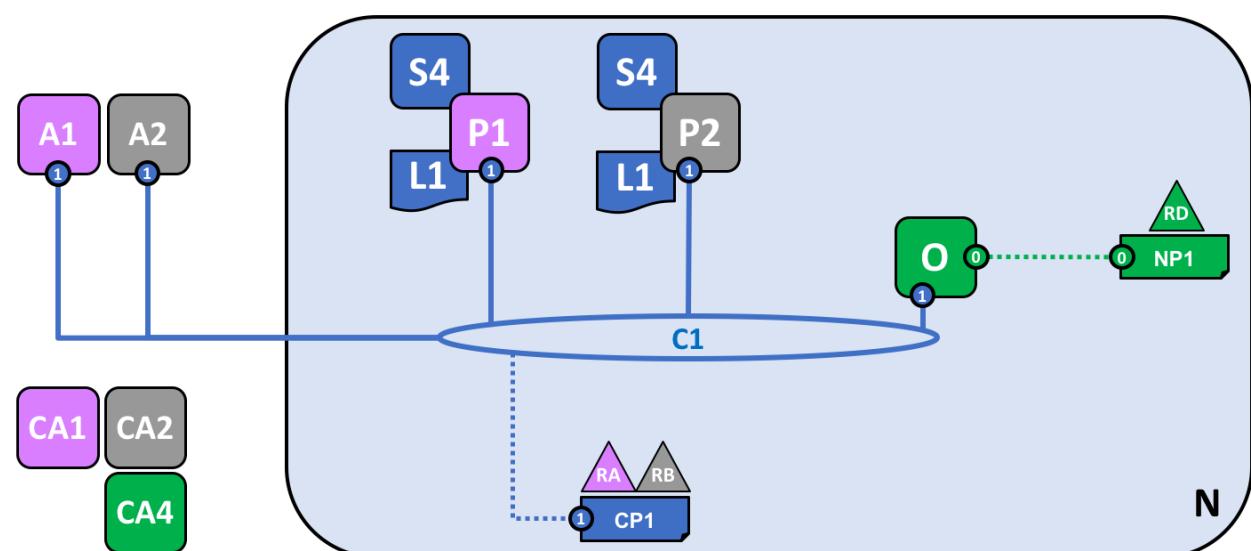
::: Hyperledger Fabric (5/16)

Una Certificate Authority (CA) emette i certificati per consentire

- alle organizzazioni di autenticarsi sulla rete;
- alle applicazioni client di autenticare le proposte di transazione ;
- ai peer di approvare le proposte e caricare le transazioni nel libro mastro se valide.

Possono esserci una o più CA sulla rete e le organizzazioni possono scegliere di utilizzare una propria CA.

Non vi siano limiti teorici alla dimensione di una rete, e il protocollo gossip è usato per accogliere un gran numero di nodi peer sulla rete.



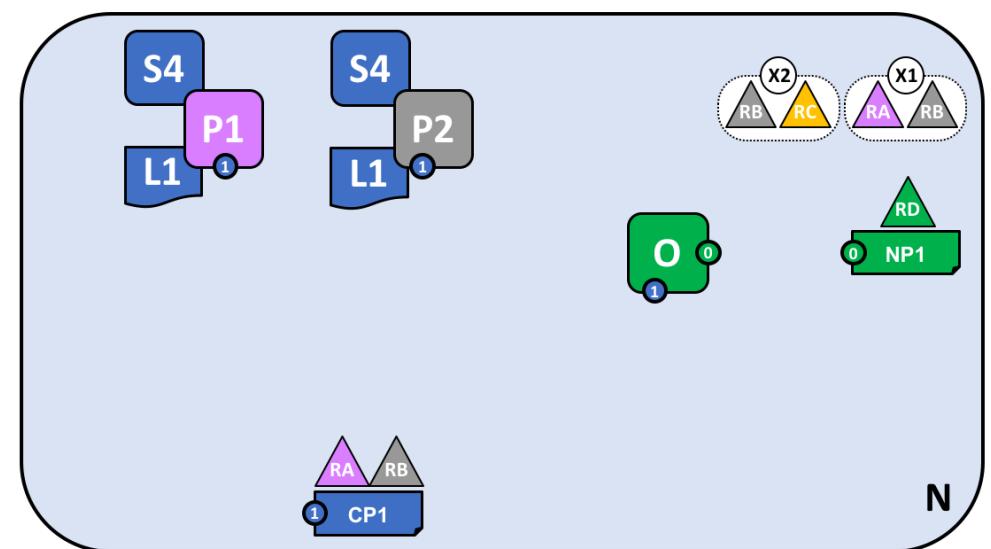
::: Hyperledger Fabric (5/16)

Una Certificate Authority (CA) emette i certificati per consentire

- alle organizzazioni di autenticarsi sulla rete;
- alle applicazioni client di autenticare le proposte di transazione ;
- ai peer di approvare le proposte e caricare le transazioni nel libro mastro se valide.

Possono esserci una o più CA sulla rete e le organizzazioni possono scegliere di utilizzare una propria CA.

È possibile definire un altro consorzio, dove una terza organizzazione interagisce con la seconda.



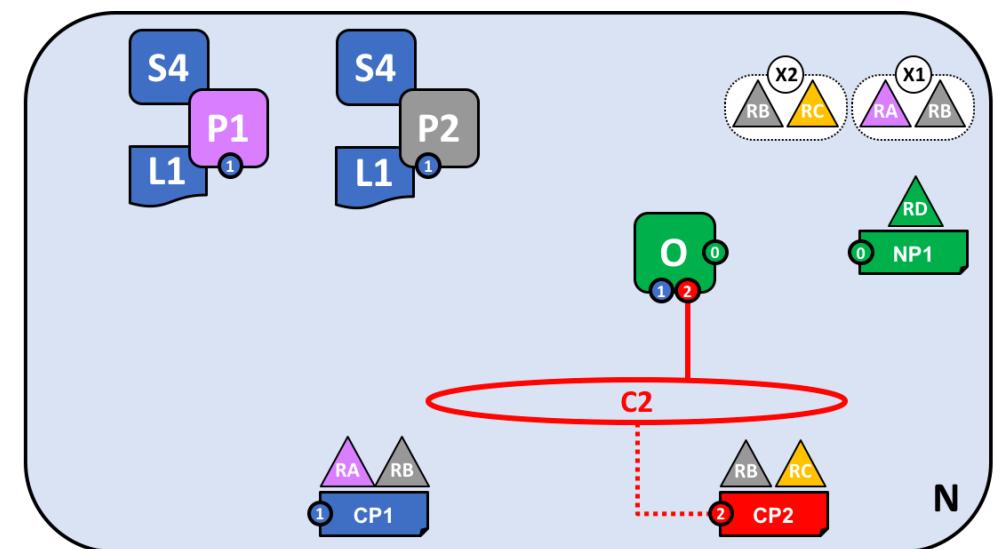
::: Hyperledger Fabric (5/16)

Una Certificate Authority (CA) emette i certificati per consentire

- alle organizzazioni di autenticarsi sulla rete;
- alle applicazioni client di autenticare le proposte di transazione ;
- ai peer di approvare le proposte e caricare le transazioni nel libro mastro se valide.

Possono esserci una o più CA sulla rete e le organizzazioni possono scegliere di utilizzare una propria CA.

Un secondo canale nell'ambito del secondo consorzio è definibile, con proprie politiche di accesso.



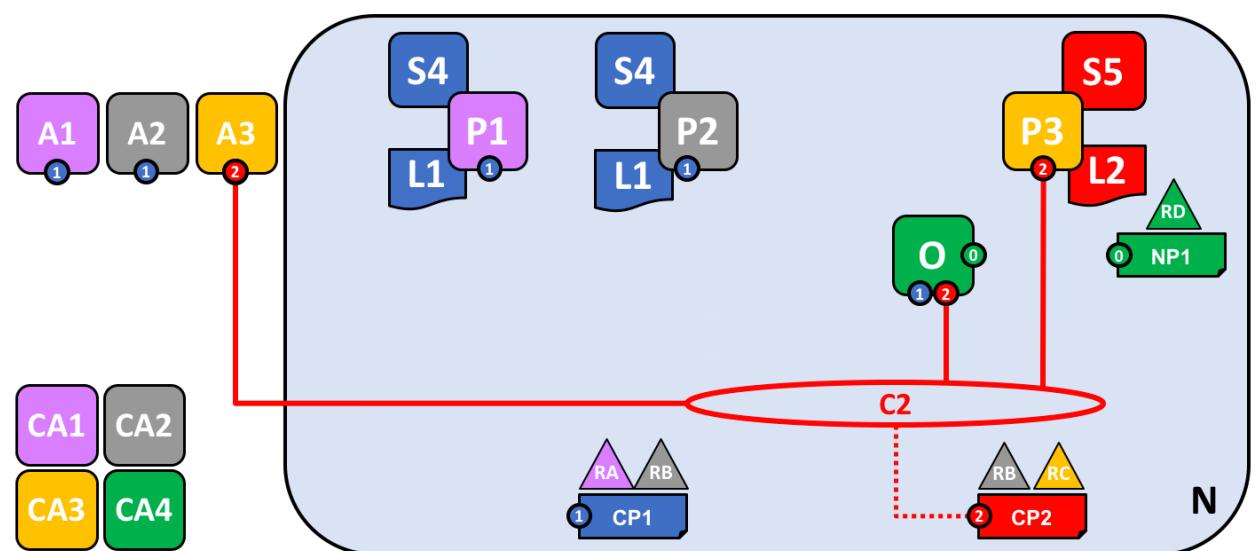
::: Hyperledger Fabric (5/16)

Una Certificate Authority (CA) emette i certificati per consentire

- alle organizzazioni di autenticarsi sulla rete;
- alle applicazioni client di autenticare le proposte di transazione ;
- ai peer di approvare le proposte e caricare le transazioni nel libro mastro se valide.

Possono esserci una o più CA sulla rete e le organizzazioni possono scegliere di utilizzare una propria CA.

Un peer per la terza organizzazione può essere istanziato e collegato al secondo canale, dispiegando il chaincode dello smart contract S5.

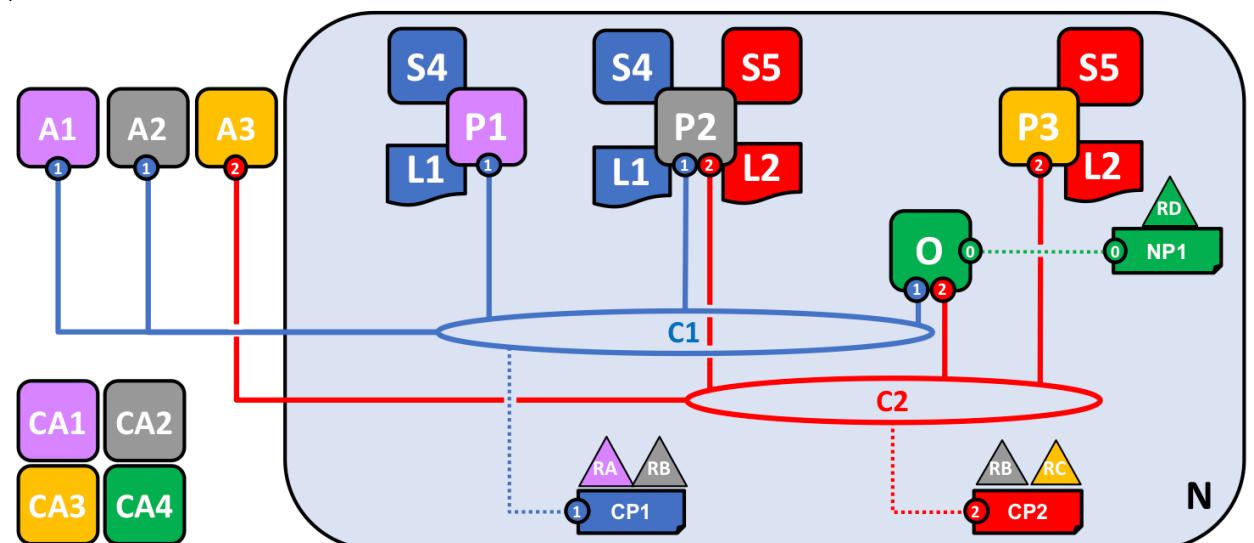


::: Hyperledger Fabric (5/16)

Una Certificate Authority (CA) emette i certificati per consentire

- alle organizzazioni di autenticarsi sulla rete;
- alle applicazioni client di autenticare le proposte di transazione ;
- ai peer di approvare le proposte e caricare le transazioni nel libro mastro se valide.

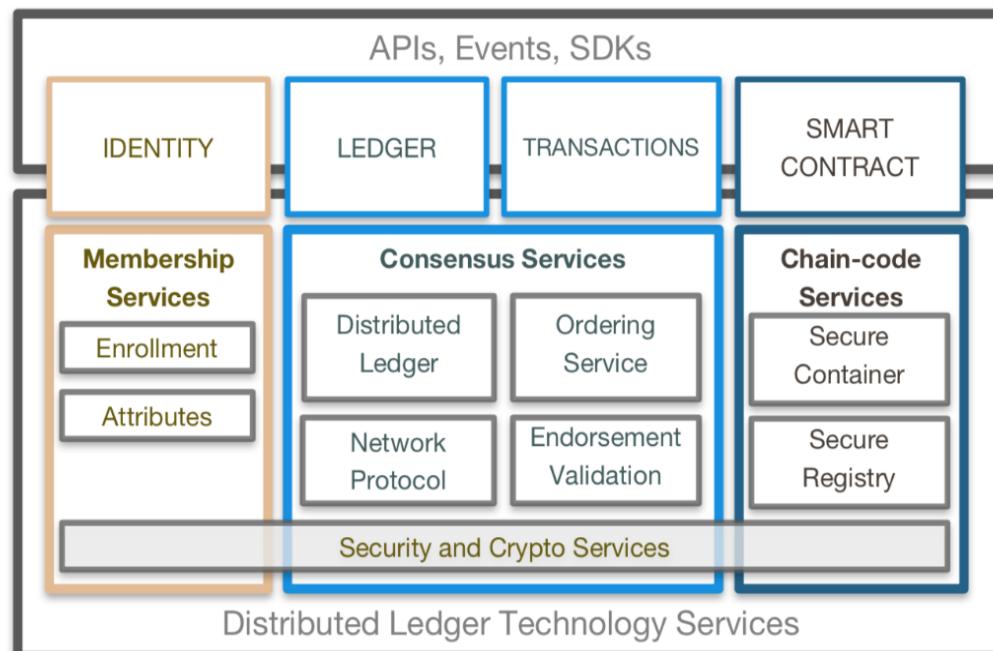
Possono esserci una o più CA sulla rete e le organizzazioni possono scegliere di utilizzare una propria CA.



::: Hyperledger Fabric (6/16)

Nel complesso l'architettura di Hyperledger è composta da tre macro-aree:

1. Una per la gestione delle identità, dell'appartenenza di peer ad associazioni e l'iscrizione del peer ai canali;
2. Un libro mastro con lo stato corrente dei dati e uno storico delle transazioni;

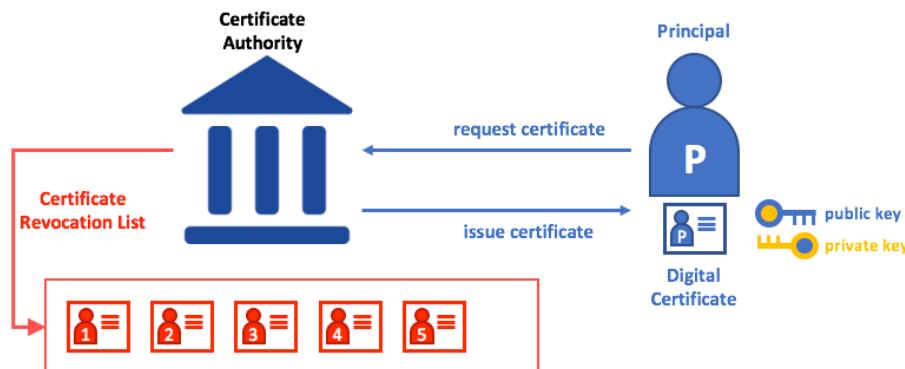


3. L'ambiente di esecuzione dei chaincode per gli smart contract.

Trasversalmente sono disponibili delle primitive crittografiche a supporto di questi tre strati.

::: Hyperledger Fabric (7/16)

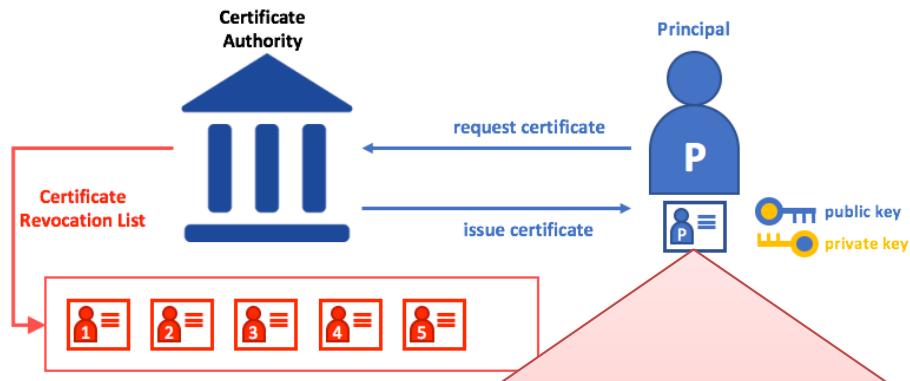
Ogni elemento ha un'identità digitale incapsulata in un certificato digitale X.509, esprimendo anche le autorizzazioni necessarie per le risorse e l'accesso alle funzionalità nella rete blockchain.



Affinché un'identità sia verificabile, deve provenire da un'autorità fidata, come la CA con un modello gerarchico PKI.

::: Hyperledger Fabric (7/16)

Ogni elemento ha un'identità digitale encapsulata in un certificato digitale X.509, esprimendo anche le autorizzazioni necessarie per le risorse e l'accesso alle funzionalità nella rete blockchain.



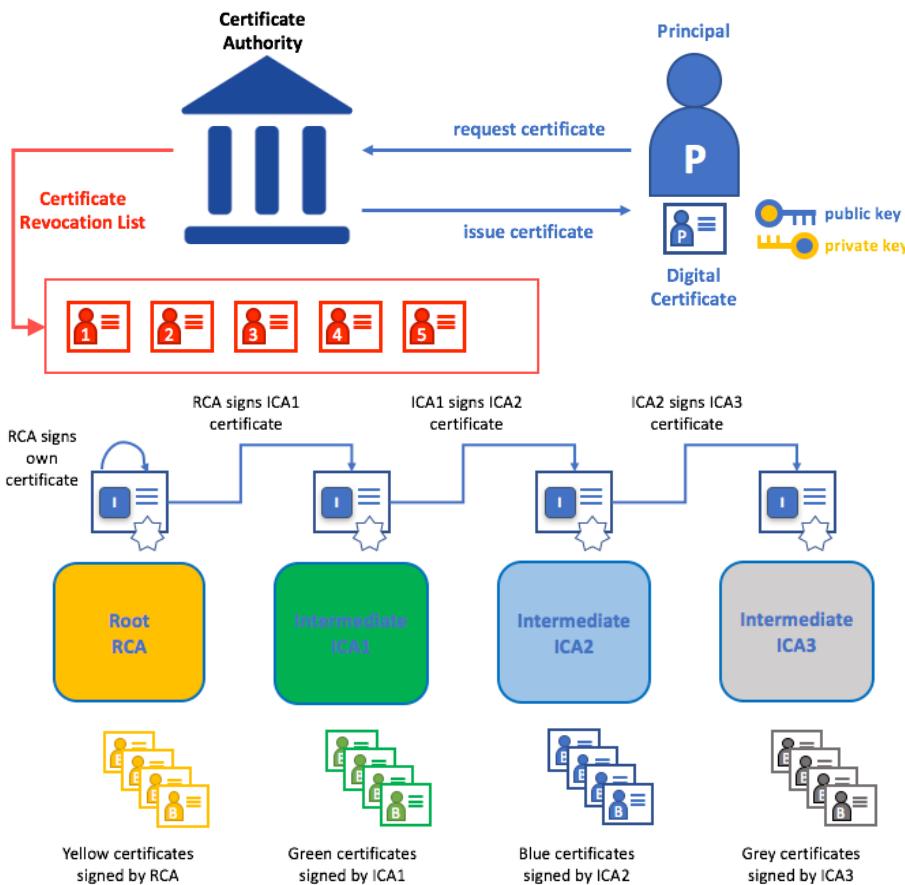
Affinché un'identità sia verificabile, deve provenire da un'autorità fidata, come la CA con un modello gerarchico PKI.



Il certificato digitale contiene molte informazioni in merito ad un'identità in SUBJECT, ma anche la chiave pubblica legata all'identità, mentre la sua chiave di firma non lo è e viene mantenuta privata. Il certificato è controfirmato dalla CA che lo emette e lo rende impossibile da modificare.

::: Hyperledger Fabric (7/16)

Ogni elemento ha un'identità digitale incapsulata in un certificato digitale X.509, esprimendo anche le autorizzazioni necessarie per le risorse e l'accesso alle funzionalità nella rete blockchain.



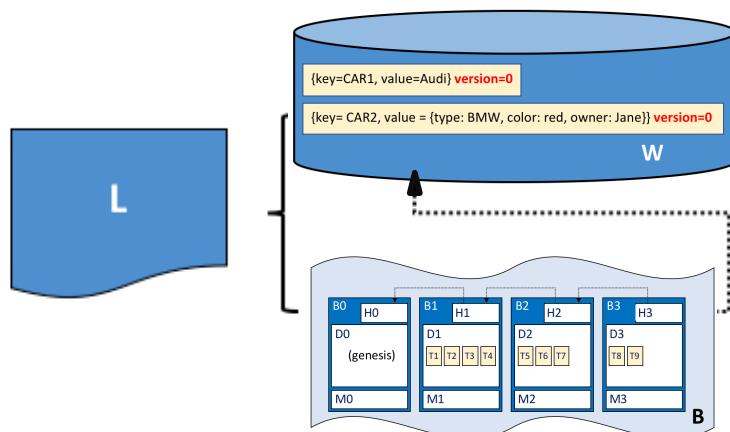
Affinché un'identità sia verificabile, deve provenire da un'autorità fidata, come la CA con un modello gerarchico PKI.

Esiste una gerarchia di CA, con una Root CA come radice e una serie di CA intermediari i cui certificati sono firmati dalle CA di livello superiore. Fabric offre una propria CA radice.

::: Hyperledger Fabric (8/16)

Il libro mastro blockchain è costituito da due parti distinte:

- uno stato globale in un database che contiene i valori correnti di un insieme di dati indirizzabili per mezzo di una chiave.
- Uno storico delle transazioni come blockchain, con tutti i cambiamenti dello stato globale.

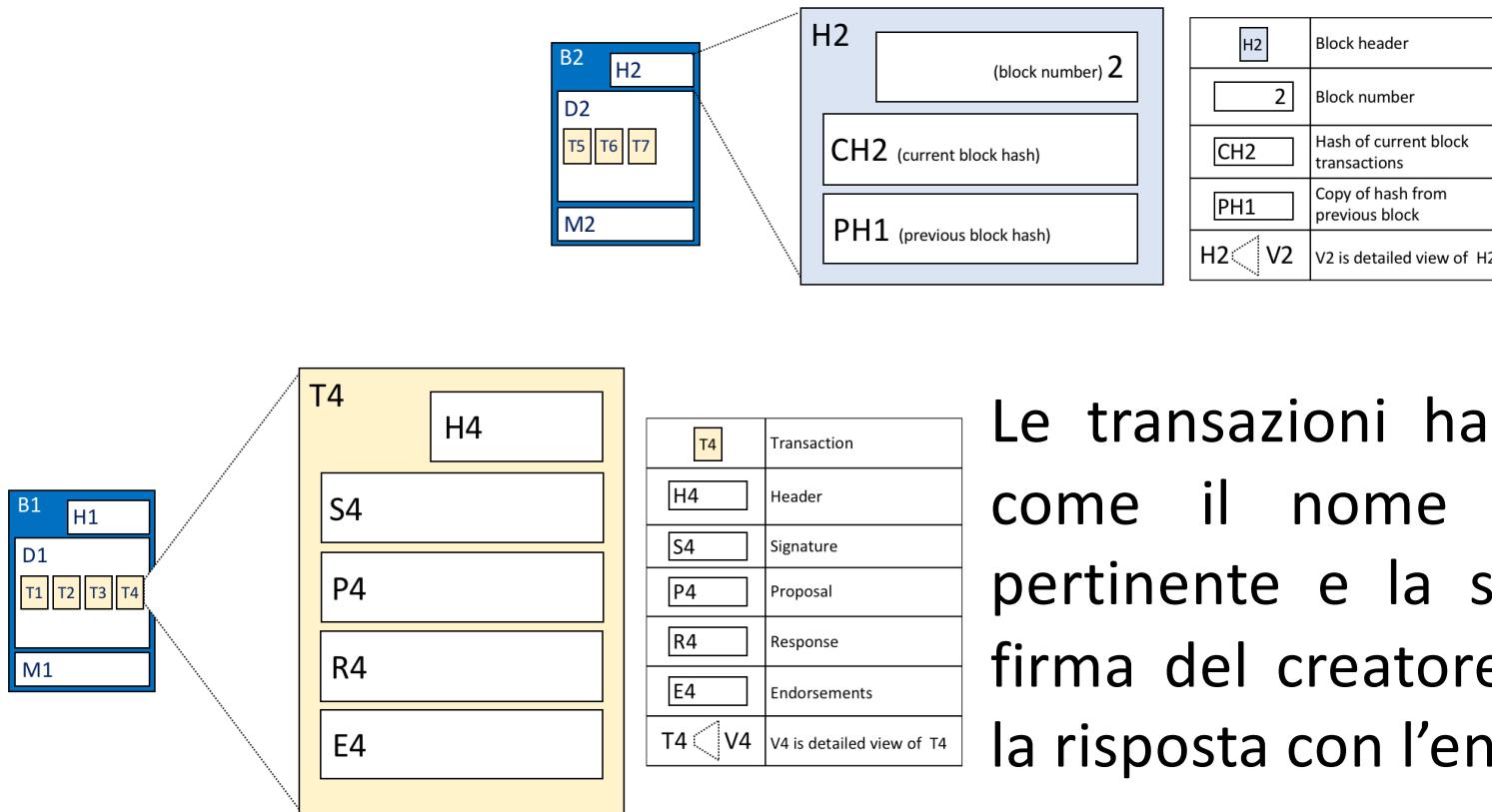


L	Ledger
W	World State
B	Blockchain
L { W	L comprises B and W
W { B	B determines W
B	H2 is chained to H1
B	Blockchain
B1	Block
H3	Block header
D1	Block data
T5	Transaction
M3	Block metadata
H1	
H2	
W	Ledger world state
{key=K, value = V } version=0	A ledger state with key=K. It contains a set of facts expressed as a simple value, V. The state is at version 0.
{key=K, value = {KV} } version=0	A ledger state with key=K. It contains a set of facts expressed as a set of key-value pairs (KV). The state is at version 0.

Il numero di versione viene incrementato ogni volta che lo stato cambia, e anche verificato ogni volta che lo stato viene aggiornato.

::: Hyperledger Fabric (9/16)

Un blocco ha un header con un numero di sequenza, l'hash del blocco e del blocco precedente, il corpo con l'insieme ordinato delle transazioni e dei meta-dati con l'ora di creazione del blocco, il certificato, la chiave pubblica e la firma del creatore del blocco.

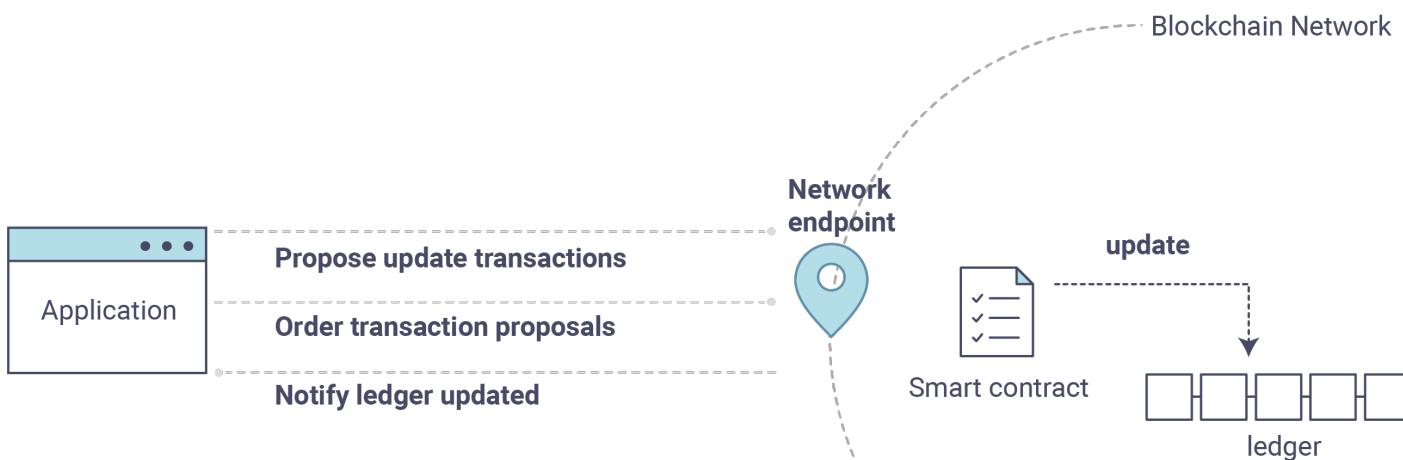


Le transazioni hanno un header come il nome del chaincode pertinente e la sua versione, la firma del creatore, la proposta e la risposta con l'endorsement.

::: Hyperledger Fabric (10/16)

Chaincode è un programma scritto in Go, node.js o Java che implementa un'apposita interfaccia e viene eseguito in un contenitore Docker isolato dal processo dell'Endorser peer.

- Lo stato creato da un chaincode è limitato esclusivamente a quel chaincode e non è possibile accedervi direttamente da un altro chaincode. Tuttavia, un chaincode può invocare un altro chaincode per accedere al suo stato.



::: Hyperledger Fabric (11/16)

L'interfaccia Chaincode specifica le funzioni da implementare:

- Init() viene chiamato quando un chaincode riceve un'istanza o una transazione di aggiornamento in modo da eseguire qualsiasi inizializzazione necessaria, inclusa l'inizializzazione dello stato dell'applicazione;
- Invoke() viene chiamato in risposta alla ricezione di una transazione invoke per elaborare le proposte di transazione.

L'altra interfaccia è ChaincodeStubInterface, che viene utilizzato per accedere e modificare la blockchain e per effettuare invocazioni tra i chaincode.

Dato che il supporto Java è stato introdotto di recente, le API Node.js e Go lang sono a un livello più maturo. JavaScript non si presta bene nel caso di calcoli numerici.

::: Hyperledger Fabric (12/16)

Il linguaggio più usato per realizzazione chaincode è Go. Questo linguaggio non presenta il concetto di classe, ma Go offre struct, una versione leggera delle classi, a cui è possibile aggiungere metodi.

Realizziamo uno smart contract in Go per la gestione delle auto e dei loro proprietari. Implementiamo le struct per questi dati.

```
type Car struct{  
    modelName string  
    color string  
    serialNo string  
    manufacturer string  
    owner Owner //composition  
}  
  
type Owner struct{  
    name string  
    nationalIdentity string  
    gender string  
    address string  
}
```

Il metodo è definito esternamente, indicando nel primo caso se le modifiche nel corpo del metodo si riflettono sul chiamante.

```
//attached by reference ==> called as pointer receiver  
func (c *Car) changeOwner(newOwner Owner) {  
    c.owner = newOwner  
}  
  
/** attached by value ==> called as value receiver  
func (c Car) changeOwner(newOwner Owner) {  
    c.owner = newOwner  
}  
*/
```

::: Hyperledger Fabric (13/16)

L'interfaccia da implementare è la seguente con le due principali funzioni delle 36 attualmente specificate:

```
type Chaincode interface {
    // Init method accepts stub of type ChaincodeStubInterface as
    // argument and returns peer.Response type object
    Init(stub ChaincodeStubInterface) peer.Response

    // Invoke method takes stub of type ChaincodeStubInterface as
    // argument and returns peer.Response type object
    Invoke(stub ChaincodeStubInterface) peer.Response
}
```

Non sussiste una sintassi di derivazione, ma basta solo implementare i metodi di interesse.

```
type CarChaincode struct{
}

//Init implemented by CarChaincode
func (t *CarChaincode) Init(stub shim.ChaincodeStubInterface)
pb.Response {

}

//Invoke implemented by CarChaincode
func (t *CarChaincode) Invoke(stub shim.ChaincodeStubInterface)
pb.Response {

}
```

::: Hyperledger Fabric (14/16)

Inseriamo la logica di inizializzazione:

```
func (t *CarChaincode) Init(stub shim.ChaincodeStubInterface)
pb.Response {

    //Declare owners from Owner struct
    tom := Owner{name: "Tom H", nationalIdentity: "ABCD33457", gender: "M", address: "1, Tumbbad"}
    bob := Owner{name: "Bob M", nationalIdentity: "QWER33457", gender: "M", address: "2, Tumbbad"}

    //Decleree carfrom Car struct
    car := Car{modelName: "Land Rover", color: "white", serialNo: "334712531234", manufacturer: "Tata Motors", owner: tom}

    // convert tom Owner to []byte
    tomAsJSONBytes, _ := json.Marshal(tom)
    //Add Tom to ledger
    err := stub.PutState(tom.nationalIdentity, tomAsJSONBytes)
    if err != nil {
        return shim.Error("Failed to create asset " + tom.name)
    }

    //Add Bob to ledger
    bobAsJSONBytes, _ := json.Marshal(bob)
    err = stub.PutState(bob.nationalIdentity, bobAsJSONBytes)
    if err != nil {
        return shim.Error("Failed to create asset " + bob.name)
    }

    //Add car to ledger
    carAsJSONBytes, _ := json.Marshal(car)
    err = stub.PutState(car.serialNo, carAsJSONBytes)
    if err != nil {
        return shim.Error("Failed to create asset " + car.serialNo)
    }

    return shim.Success([]byte("Assets created successfully."))
}
```

::: Hyperledger Fabric (15/16)

```
func (c *CarChaincode) Invoke(stub shim.ChaincodeStubInterface)
pb.Response {

    // Read args from the transaction proposal.
    // fc=> method to invoke
    fc, args := stub.GetFunctionAndParameters()
    if fc == "TransferOwnership" {
        return c.TransferOwnership(stub, args)
    }
    return shim.Error("Called function is'nt defined in the chaincode")
}

func (c *CarChaincode) TransferOwnership(stub
shim.ChaincodeStubInterface, args []string) pb.Response {
    // args[0]=> car serial no
    // args[1]==> new owner national identity
    // Read existing car asset
    carAsBytes, _ := stub.GetState(args[0])
    if carAsBytes == nil {
        return shim.Error("car asset not found")
    }

    // Construct the struct Car
    car := Car{}
    _ = json.Unmarshal(carAsBytes, &car)
```

Inseriamo la logica di invoke:

```
//Read newOwnerDetails
ownerAsBytes, _ := stub.GetState(args[1])
if ownerAsBytes == nil {
    return shim.Error("owner asset not found")
}

// Construct the struct Owner
newOwner := Owner{}
_ = json.Unmarshal(ownerAsBytes, &newOwner)

// Update owner
car.changeOwner(newOwner)

carAsJSONBytes, _ := json.Marshal(car)

// Update car ownership in the
err := stub.PutState(car.serialNo, carAsJSONBytes)
if err != nil {
    return shim.Error("Failed to create asset " + car.serialNo)
}
return shim.Success([]byte("Asset modified."))
}
```

::: Hyperledger Fabric (16/16)

Per completare lo smart contract è necessario specificare il preambolo:

```
package main

import (
    "encoding/json"

    "github.com/hyperledger/fabric/core/chaincode/shim"
    pb "github.com/hyperledger/fabric/protos/peer"
)
```

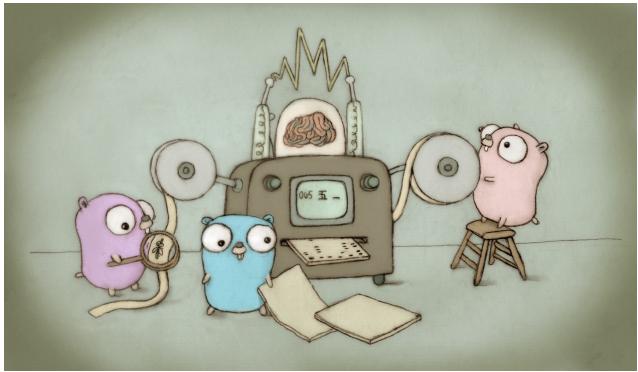
E la funzione principale:

```
func main() {

    logger.SetLevel(shim.LogInfo)

    // Start chaincode process
    err := shim.Start(new(CarChaincode))
    if err != nil {
        logger.Error("Error starting PhantomChaincode - ", err.Error())
    }
}
```

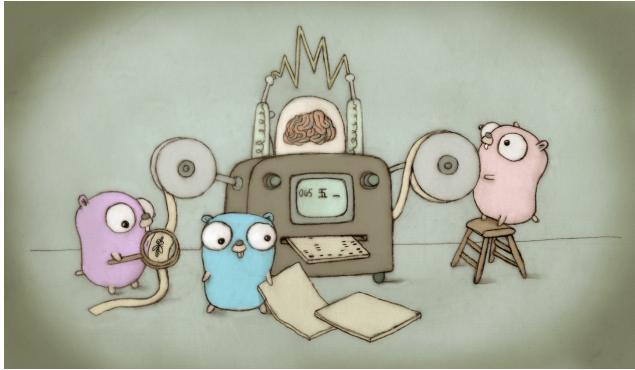
::: Golang (1/28)



Go è un linguaggio di programmazione open source offerto da Google che soddisfa le esigenze di una compilazione efficiente, di un'esecuzione veloce e di una facilità di programmazione.

È sintatticamente simile a C con tipizzazione statica, ma con memory safety, garbage collection, tipizzazione strutturale, e concorrenza in stile CSP.

::: Golang (1/28)

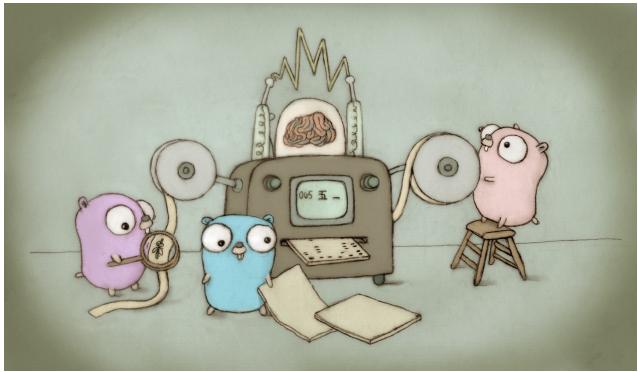


Go è un linguaggio di programmazione open source offerto da Google che soddisfi le esigenze di una compilazione efficiente, di un'esecuzione veloce e di una facilità di programmazione.

È sintatticamente simile a C con tipizzazione statica, ma con **memory safety**, garbage collection, tipizzazione strutturale, e concorrenza in stile CSP.

Implica l'essere protetti da vari bug del software e vulnerabilità di sicurezza quando si ha a che fare con l'accesso alla memoria, come buffer overflow e puntatori penzolanti. Java è sicuro per la memoria perché il rilevamento degli errori a runtime controlla i limiti dell'array e le dereferenze dei puntatori. Al contrario, C e C++ consentono l'aritmetica dei puntatori arbitrari con i puntatori aventi indirizzi di memoria diretti senza alcuna previsione per il controllo dei limiti, e quindi sono potenzialmente non sicuri per la memoria.

::: Golang (1/28)

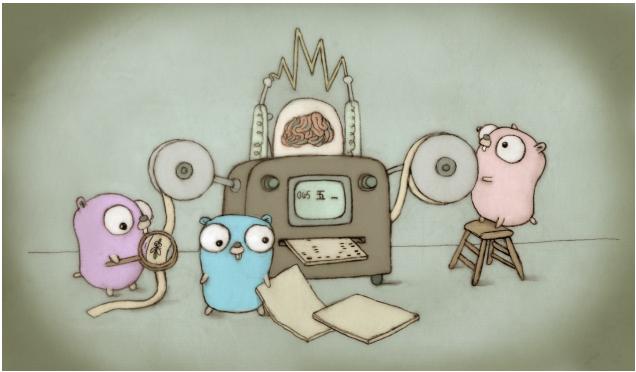


Go è un linguaggio di programmazione open source offerto da Google che soddisfi le esigenze di una compilazione efficiente, di un'esecuzione veloce e di una facilità di programmazione.

È sintatticamente simile a C con tipizzazione statica, ma con memory safety, garbage collection, **tipizzazione strutturale**, e concorrenza in stile CSP.

La compatibilità e l'equivalenza del tipo sono determinate dalla struttura o dalla definizione effettiva del tipo e non da altre caratteristiche come il suo nome o il luogo della dichiarazione.

::: Golang (1/28)

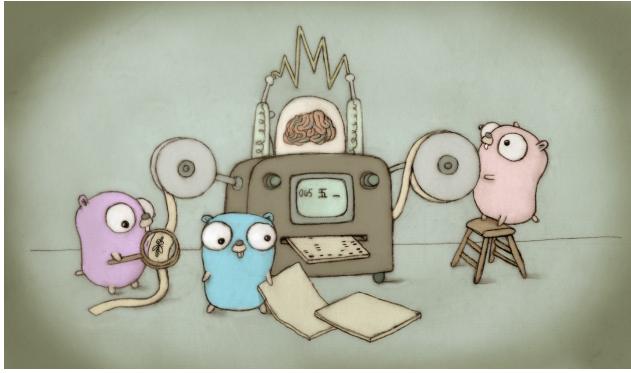


Go è un linguaggio di programmazione open source offerto da Google che soddisfi le esigenze di una compilazione efficiente, di un'esecuzione veloce e di una facilità di programmazione.

È sintatticamente simile a C con tipizzazione statica, ma con memory safety, garbage collection, tipizzazione strutturale, e **concorrenza in stile CSP**.

Communicating Sequential Processes (CSP) è un linguaggio formale per descrivere modelli di interazione in sistemi concorrenti, basato sul passaggio di messaggi attraverso i canali.

::: Golang (1/28)



Go è un linguaggio di programmazione open source offerto da Google che soddisfi le esigenze di una compilazione efficiente, di un'esecuzione veloce e di una facilità di programmazione.

È sintatticamente simile a C con tipizzazione statica, ma con memory safety, garbage collection, tipizzazione strutturale, e concorrenza in stile CSP.

```
hello.go
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello, 世界")
7 }
```

Per esercitarsi è possibile installare direttamente la toolchain di programmazione o mediante Docker. È anche possibile servirsi di un ambiente playground:

<https://go.dev/play/>

::: Golang (2/28)

Ha una sintassi fondamentalmente simile a C utilizzando la forma estesa di Backus-Naur (EBNF), con tipi e dichiarazioni invertiti, più parole chiave per introdurre ogni tipo di dichiarazione.

```
var a int  
var b, c *int  
var d []int  
type S struct { a, b int }
```

Le strutture di controllo sono familiari, ma senza parentesi tonde:

```
if a == b { return true } else { return false }  
for i = 0; i < 10; i++ { ... }
```

Un punto e virgola viene automaticamente inserito alla fine di una riga non vuota se il token finale della riga è

- un identificatore; un valore intero, a virgola mobile, immaginario, carattere o stringa; una delle parole chiave break, continue, fallthrough, o return; uno degli operatori e delimitatori ++ , -- ,) ,] , o }.

Un punto e virgola può essere omesso prima di una chiusura ")" o "}".

::: Golang (3/28)

int	uint		
int8	uint8 = byte		
int16	uint16		
int32	uint32	float32	complex64
int64	uint64	float64	complex128

Un valore intero è una sequenza di cifre che rappresenta una costante intera. Un prefisso facoltativo fissa una base non decimale: 0 per ottale, 0x o 0X per esadecimale.

Int non è int32, perché la sua dimensione è specifica dell'implementazione, come uint. uintptr rappresenta un numero intero abbastanza grande da memorizzare un puntatore.

Il solito tipo booleano, bool, con valori true e false.

string rappresenta come array di byte immutabile: una volta creati, è impossibile modificare il contenuto di una stringa. Gli elementi di stringhe hanno tipo byte e si possono accedere utilizzando le consuete operazioni di indicizzazione, senza riportarne l'indirizzo.

::: Golang (4/28)

La conversione di un valore numerico da un tipo a un altro è realizzabile mediante una chiamata ad apposita funzione:

`uint8(intVar) // tronca alla dimensione`

`int(float64Var) // tronca la frazione`

`float64(intVar) //converte in float64`

Le costanti numeriche sono "numeri ideali": nessuna dimensione o segno. La sintassi del letterale determina il tipo:

`1.234e5 // floating-point`

`1e2 // floating-point`

`3.2i // imaginary floating-point`

`100 // integer`

Le costanti possono essere combinati in espressioni, dette costanti, con il tipo della risultato determinato dal tipo delle costanti.

::: Golang (5/28)

Il linguaggio consente l'uso di costanti senza conversione esplicita se il valore può essere rappresentato:

```
uint8(^0)          // bad: -1 can't be represented  
^uint8(0)          // OK  
uint8(350)         // bad: 350 can't be represented  
uint8(35.0)        // OK: 35  
uint8(3.5)         // bad: 3.5 can't be represented
```

Le dichiarazioni sono introdotte da una parola chiave (var, const, type, func) e sono invertite rispetto al C:

```
var i int  
const PI = 22./7.  
type Point struct { x, y int }  
func sum(a, b int) int { return a + b }
```

::: Golang (6/28)

Le dichiarazioni di variabili sono introdotte da var. Possono avere un tipo o un'espressione di inizializzazione.

```
var i int
```

```
var j = 365.245
```

```
var k int = 0
```

Può essere noioso, quindi le dichiarazioni a const, type, var ma non a func possono essere raggruppate.

```
var (
```

```
    i int
```

```
    j = 356.245
```

```
    k int = 0
```

```
    l, m uint64 = 1, 2
```

```
    inter, floater, stringer = 1, 2.0, "hi"
```

```
)
```

::: Golang (7/28)

Nelle funzioni (solo), dichiarazioni della forma var v = valore possono essere abbreviate in v := valore. Il tipo è quello del valore

a, b, c, d, e := 1, 2.0, "tre", FOUR, 5e0i

Queste abbreviazioni sono usati molto in punti come gli inizializzatori di loop. Le dichiarazioni di costanti sono introdotte da const, hanno una "espressione costante", valutata in fase di compilazione, come inizializzatore e l'indicazione del tipo è facoltativo.

const Pi = 22./7.

const AccuratePi float64 = 355./113

const manzo, due, pastinaca = "carne", 2, "veg"

const (

lunedì, martedì, mercoledì = 1, 2, 3

giovedì, venerdì, sabato = 4, 5, 6

)

::: Golang (8/28)

Le dichiarazioni di costanti possono utilizzare il contatore iota, che inizia da 0 in ogni blocco const e aumenta a ogni punto e virgola implicito (fine riga).

```
const (
    Monday = iota // 0
    Tuesday = iota // 1
)
```

La funzione built-in new alloca la memoria. La sintassi è come una chiamata di funzione, con tipo come argomento, simile a C++. Restituisce un puntatore all'oggetto assegnato.

```
var p *Punto = new(Punto)
v := new(int) // v ha il tipo *int
```

Non ci sono funzioni di delete, perché Go ha un Garbage Collector.

::: Golang (9/28)

Le strutture di controllo sono simili a C, ma con differenze.

- L'istruzione if ha una forma con 1 elemento pari alla condizione, che può essere preceduta da una istruzione di inizializzazione che termina con il punto e virgola.

```
if v := f(); v < 10 {  
    fmt.Printf("%d less than 10\n", v)  
} else {  
    fmt.Printf("%d not less than 10\n", v)  
}
```

- Lo switch ha una forma a 2 elementi, è simile a quello del C ma con importanti differenze:
 - le espressioni non devono essere costanti o addirittura int;
 - Nessun fallthrough automatico, ma lessicamente l'ultimo case può essere fallthrough oppure bisogna usare un apposita parola chiave;
 - più casi possono essere separati da virgolette.

::: Golang (9/28)

Le strutture di controllo sono simili a C, ma con differenze.

- L'istruzione if ha una forma con 1 elemento pari alla condizione, che può essere preceduta da una istruzione di inizializzazione che termina con il punto e virgola.

```
if v := f(); v < 10 {  
    fmt.Printf("%d less than 10\n", v)  
} else {  
    fmt.Printf("%d not less than 10\n", v)
```

Il controllo dovrebbe fluire dalla fine di un blocco case alla prima istruzione del blocco successivo. In caso contrario, il controllo scorre fino alla fine dello "switch".

- LO SWITCH ha una struttura a 2 elementi, e simile a quello del C ma con importanti differenze:
 - le espressioni non devono essere costanti o addirittura int;
 - Nessun **fallthrough** automatico, ma lessicamente l'ultimo case può essere fallthrough oppure bisogna usare un apposita parola chiave;
 - più casi possono essere separati da virgolette.

::: Golang (10/28)

```
switch count%7 {  
    case 4,5,6: error()  
    case 3: a *= v; fallthrough  
    case 2: a *= v; fallthrough  
    case 1: a *= v; fallthrough  
    case 0: return a*v  
    default: fmt.Printf("non-zero")  
}
```

Senza il primo elemento dello switch si ha una catena if-else:

a, b := x[i], y[j]

```
switch {  
    case a < b: return -1  
    case a == b: return 0  
    case a > b: return 1  
}
```



```
switch a, b := x[i], y[j]; { ... }
```

- Il ciclo for ha forme a 1 e 3 elementi:
 - La forma con un singolo elemento è il while in C: for a {}
 - La forma con tre elementi è il for in C: for a;b;c {}

::: Golang (11/28)

Le funzioni sono introdotte dalla parola chiave func. Il tipo restituito, se presente, viene dopo i parametri. L'istruzione return è come quella nel C.

```
func square(f float64) float64 { return f*f }
```

Una funzione può restituire più valori. In tal caso, i tipi restituiti sono un elenco tra parentesi.

```
func MySqrt(f float64) (float64, bool) {
    if f >= 0 { return math.Sqrt(f), true }
    return 0, false
}
```

L'identificatore vuoto, `_`, è predichiarato e gli si può sempre essere assegnato qualsiasi valore, che viene scartato. Lo si impiega per ignorare uno dei ritorni di una funzione che restituisce più valori.

::: Golang (12/28)

I "parametri" del risultato sono variabili effettive che è possibile utilizzare se le dichiarate nella clausola di ritorno a seguito della firma della funzione.

```
func MySqrt(f float64) (v float64, ok bool) {  
    if f >= 0 { v,ok = math.Sqrt(f), true }  
    else { v,ok = 0,false }  
    return v,ok  
}
```

Infine, un ritorno senza espressioni restituisce il valore corrente delle variabili di risultato, che sono inizializzate secondo il loro tipo. L'istruzione defer esegue una funzione (o metodo) quando il contesto o funzione che la contiene ritorna. Gli argomenti sono valutati al momento del defer, non della sua definizione.

::: Golang (13/28)

```
func data(fileName string) string {  
    f := os.Open(fileName)  
    defer f.Close()  
    contents := io.ReadAll(f)  
    return contents  
}
```

Ogni defer eseguito mette in coda una chiamata di funzione da eseguire, in ordine LIFO:

```
func f() {  
    for i := 0; i < 5; i++ {  
        defer fmt.Printf("%d ", i)  
    }  
}
```

::: Golang (14/28)

Come in C, le funzioni non possono essere dichiarate all'interno delle funzioni, ma i valori letterali di funzione possono essere assegnati alle variabili. Rappresentano una forma di espressioni lambda:

```
func f() {  
    for i := 0; i < 10; i++ {  
        g := func(i int) { fmt.Printf("%d",i) }  
        g(i)  
    }  
}
```

::: Golang (14/28)

Come in C, le funzioni non possono essere dichiarate all'interno delle funzioni, ma i valori letterali di funzione possono essere assegnati alle variabili. Rappresentano una forma di espressioni lambda:

```
func f() {  
    for i := 0; i < 10; i++ {  
        g := func(i int) { fmt.Printf("%d",i) }  
        g(i)  
    }  
}
```

Qual è il risultato di questa esecuzione?

```
func adder() (func(int) int) {  
    var x int  
    return func(delta int) int {  
        x += delta  
        return x  
    }  
}  
f := adder()  
fmt.Println(f(1))  
fmt.Println(f(20))  
fmt.Println(f(300))
```

::: Golang (14/28)

Come in C, le funzioni non possono essere dichiarate all'interno delle funzioni, ma i valori letterali di funzione possono essere assegnati alle variabili. Rappresentano una forma di espressioni lambda:

```
func f() {  
    for i := 0; i < 10; i++ {  
        g := func(i int) { fmt.Printf("%d",i) }  
        g(i)  
    }  
}
```

Stampa 1 21 321 – accumulando nella
Variabile x di f.

```
func adder() (func(int) int) {  
    var x int  
    return func(delta int) int {  
        x += delta  
        return x  
    }  
}  
f := adder()  
fmt.Println(f(1))  
fmt.Println(f(20))  
fmt.Println(f(300))
```

::: Golang (15/28)

All'interno di un package, tutte le variabili globali, le funzioni, i tipi e le costanti sono visibili da tutti i file sorgente del pacchetto.

Per i client (importatori) del pacchetto, i nomi devono essere in maiuscolo per essere visibili: variabili globali, funzioni, tipi, costanti, più metodi e campi struttura per variabili globali e tipi.

```
const hello = "you smell"          // package visible
const Hello = "you smell nice"    // globally visible
const _Bye = "stinko!"            // _ is not upper
```

Due modi per inizializzare le variabili globali prima dell'esecuzione di main.main:

1. Una dichiarazione globale con un iniziatore;
2. All'interno di una funzione init(), di cui può esserci un numero qualsiasi in ogni file sorgente.

::: Golang (16/28)

Go non ha classi, ma si può collegare metodi a qualsiasi tipo. I metodi sono dichiarati, separati dalla dichiarazione del tipo, come funzioni con un destinatario esplicito.:

```
type Point struct { x, y float64 }

// A method on *Point

func (p *Point) Abs() float64 {
    return math.Sqrt(p.x*p.x + p.y*p.y)
}
```

È possibile anche avere un destinatario esplicito non come puntatore, per avere il passaggio per valore. Il metodo è invocabile con l'operatore ':':

```
p := &Point{ 3, 4 }
fmt.Println(p.Abs())           // stamperà 5
```

::: Golang (17/28)

Le funzioni possono avere destinatari impliciti anche non struct:

```
type IntVector []int
func (v IntVector) Sum() (s int) {
    for _, x := range v { // blank identifier!
        s += x
    }
    return
}
fmt.Println(IntVector{1, 2, 3}.Sum())
```

Go introduce anche una semplificazione sintattica per i vari tipi di destinatari espliciti.

```
p1 := Point{ 3, 4 }
fmt.Println(p1.Abs()) // semplificazione per (&p1).Abs()
```

o

```
p3 := &Point3{ 3, 4, 5 }
fmt.Println(p3.Abs()) // semplificazione per (*p3).Abs()
```

::: Golang (18/28)

Quando un campo anonimo è incorporato in una struttura, anche i metodi di quel tipo sono incorporati - in effetti, eredita i metodi:

```
type Point struct { x, y float64 }
func (p *Point) Abs() float64 { ... }

type NamedPoint struct {
    Point
    name string
}
n := &NamedPoint{Point{3, 4}, "Pythagoras"}
fmt.Println(n.Abs()) // stampa 5
```

Questo meccanismo sintattico consente anche l'overidding di una funzione nel tipo con il campo anonimo.

::: Golang (19/28)

```
type NamedPoint struct {  
    Point  
    name string  
}  
  
func (n *NamedPoint) Abs() float64 {  
    return n.Point.Abs() * 100.  
}  
  
n := &NamedPoint{Point{3, 4}, "Pythagoras"}  
fmt.Println(n.Abs())                  // stampa 500
```

fmt.Print e funzioni similari possono identificare i valori da stampare invocando il metodo String() del tipo della variabile passata. Questo è un meccanismo similare al `toString()` in Java.

::: Golang (20/28)

Go è molto diverso da C++ rispetto alla visibilità.

1. Go ha uno scope di package (C++ lo ha rispetto ai file).
2. L'ortografia determina una vista globale/locale (pub/priv).
3. I tipi struct nello stesso package hanno pieno accesso a campi e metodi altrui.
4. Il tipo locale può esportare i propri campi e metodi.
5. Nessuna vera sottoclasse, quindi nessuna nozione di "protected".

Un'interfaccia è un insieme di metodi, che vengono solo specificati ma non implementati né ha la specifica del destinatario esplicito.

```
type AbsInterface interface {  
    Abs() float64    // il destinatario è desunto  
}
```

::: Golang (21/28)

```
type MyFloat float64
func (f MyFloat) Abs() float64 {
    if f < 0 { return float64(-f) }
    return f
}
```

MyFloat implementa AbsInterface anche se non lo fa float64. Bisogna comprendere che MyFloat non è un contenitore di float64; le loro rappresentazioni sono identiche.

AbsInterface è implementato da qualsiasi tipo che ha un metodo con firma Abs() float64, indipendentemente da quali altri metodi quel tipo può avere.

Una volta che una variabile è stata dichiarata con il tipo di interfaccia, potrebbe memorizzare qualsiasi valore che implementa quell'interfaccia.

::: Golang (22/28)

Dopo che un valore di un tipo concreto è inserito in una variabile di interfaccia, è necessario un "unbox" per recuperare l'originale, utilizzando un "tipo di asserzione".

Sintassi: `interfaceValue.(typeToExtract)`

Un tipo non ha bisogno di dichiarare le interfacce che implementa, né ha bisogno di ereditare da un tipo di interfaccia. Se ne implementa i metodi, implementa l'interfaccia.

Le funzioni variadiche hanno elenchi di parametri di lunghezza variabile dichiarati con la sintassi `...T`, dove `T` è il tipo dei singoli argomenti. Tali argomenti devono essere gli ultimi nell'elenco degli argomenti. All'interno della funzione l'argomento variadico ha implicitamente il tipo `[]T`.

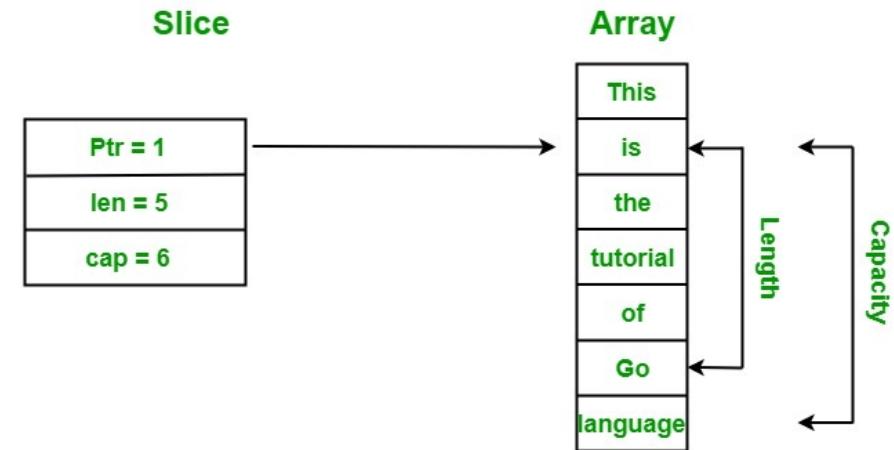
::: Golang (23/28)

Go dispone sia del tipo array e mapping, che troviamo in altri linguaggi di programmazione.

- Gli array hanno sempre una dimensione finita, e deve essere un'espressione costante specificata in dichiarazione che restituisce un valore intero non negativo. La lunghezza dell'array a può essere scoperta usando la funzione built-in `len(a)`.
- Una slice è un riferimento a un segmento contiguo di un array e contiene una sequenza numerata di elementi di tale array. Una slice viene dichiarata proprio come un array, ma non contiene la sua dimensione, che quindi può crescere o ridursi in base alle esigenze. La slice ha anche un valore di capacità che rappresenta la massima estensione che può assumere.

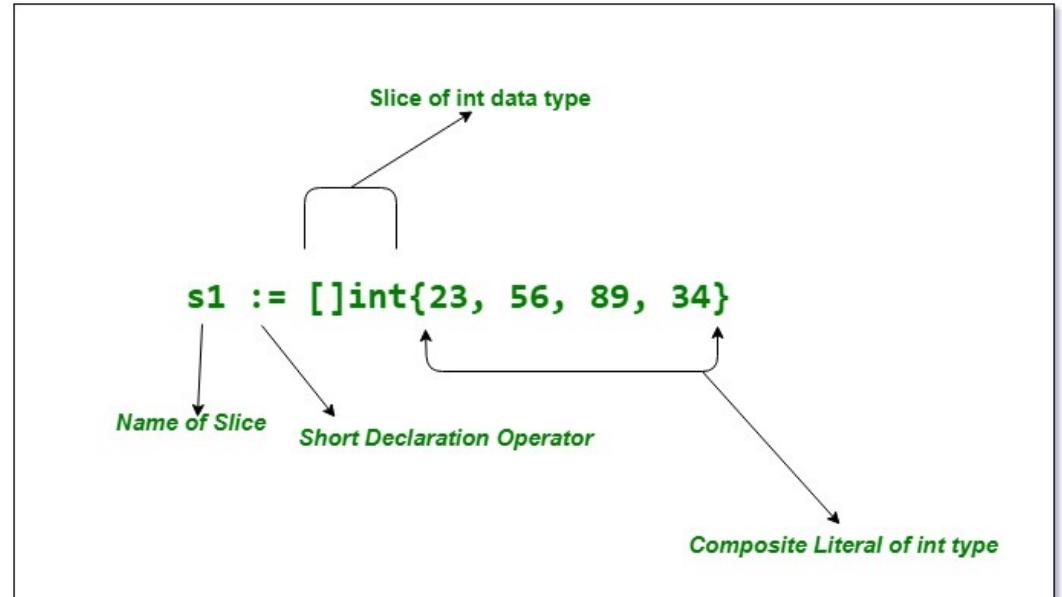
::: Golang (24/28)

```
func main() {  
  
    // Creating an array  
    arr := [7]string{"This", "is", "the", "tutorial", "of", "Go", "language"}  
  
    // Display array  
    fmt.Println("Array:", arr)  
  
    // Creating a slice  
    myslice := arr[1:6]  
  
    // Display slice  
    fmt.Println("Slice:", myslice)  
  
    // Display length of the slice  
    fmt.Printf("Length of the slice: %d", len(myslice))  
  
    // Display the capacity of the slice  
    fmt.Printf("\nCapacity of the slice: %d", cap(myslice))  
}
```



::: Golang (25/28)

```
func main() {  
  
    // Creating s slice  
    oRignAl_slice := []int{90, 60, 40, 50,  
                          34, 49, 30}  
  
    // Creating slices from the given slice  
    var my_slice_1 = oRignAl_slice[1:5]  
    my_slice_2 := oRignAl_slice[0:]  
    my_slice_3 := oRignAl_slice[:6]  
    my_slice_4 := oRignAl_slice[:]  
    my_slice_5 := my_slice_3[2:4]  
  
    // Display the result  
    fmt.Println("Original Slice:", oRignAl_slice)  
    fmt.Println("New Slice 1:", my_slice_1)  
    fmt.Println("New Slice 2:", my_slice_2)  
    fmt.Println("New Slice 3:", my_slice_3)  
    fmt.Println("New Slice 4:", my_slice_4)  
    fmt.Println("New Slice 5:", my_slice_5)  
}
```



::: Golang (26/28)

```
func main() {
    // Using make function
    var my_slice_1 = make([]int, 4, 7)
    fmt.Printf("Slice 1 = %v, \nlength = %d, \ncapacity = %d\n",
        my_slice_1, len(my_slice_1), cap(my_slice_1))

    var my_slice_2 = make([]int, 7)
    fmt.Printf("Slice 2 = %v, \nlength = %d, \ncapacity = %d\n",
        my_slice_2, len(my_slice_2), cap(my_slice_2))

    var s []int
    printSlice(s)
    s = append(s, 0)
    printSlice(s)
    s = append(s, 1)
    printSlice(s)
    s = append(s, 2, 3, 4)
    printSlice(s)

}

func printSlice(s []int) {
    fmt.Printf("len=%d cap=%d %v\n", len(s), cap(s), s)
}
```

::: Golang (26/28)

```
func main() {  
  
    // Creating a slice  
    myslice := []string{"This", "is", "the", "tutorial",  
        "of", "Go", "language"}  
  
    // Iterate using for loop  
    for e := 0; e < len(myslice); e++ {  
        fmt.Println(myslice[e])  
    }  
  
}  
  
func main() {  
  
    // Creating a slice  
    myslice := []string{"This", "is", "the", "tutorial",  
        "of", "Go", "language"}  
  
    // Iterate slice  
    // using range in for loop  
    for index, ele := range myslice {  
        fmt.Printf("Index = %d and element = %s\n", index+3, ele)  
    }  
  
}
```

```
func main() {  
  
    // Creating a slice  
    myslice := []string{"This", "is", "the",  
        "tutorial", "of", "Go", "language"}  
  
    // Iterate slice  
    // using range in for loop  
    // without index  
    for _, ele := range myslice {  
        fmt.Printf("Element = %s\n", ele)  
    }  
  
}
```

::: Golang (27/28)

- Una mappa è una memoria associativa ed è caratterizzata da un tipo per la chiave e uno per il valore.

```
var m map[string]int
```

map è un tipo di riferimento, come puntatori o slice, quindi il valore di m nella precedente dichiarazione è nil (valore di default per un riferimento), e non punta a una mappa inizializzata. Una mappa nil si comporta come una mappa vuota durante la lettura, ma i tentativi di scrivere su una mappa nil causeranno un errore a runtime. Per inizializzare una mappa, si può utilizzare la funzione make:

```
m = make(map[string]int)
```

È possibile accedere agli elementi di una mappa mediante l'operatore [] e un valore di chiave. La funzione delete consente di rimuovere un elemento.

::: Golang (28/28)

```
func main() {  
    m := make(map[string]int)  
  
    m["Answer"] = 42  
    fmt.Println("The value:", m["Answer"])  
  
    m["Answer"] = 48  
    fmt.Println("The value:", m["Answer"])  
  
    delete(m, "Answer")  
    fmt.Println("The value:", m["Answer"])  
  
    v, ok := m["Answer"]  
    fmt.Println("The value:", v, "Present?", ok)  
}
```

::: Prog. Concorrente in Go (1/10)

Una goroutine consiste nell'invocare una funzione con la parola chiave go, e realizza il fork di un thread.

```
package main

import (
    "fmt"
    "time"
    "math/rand"
)

func f(n int) {
    for i := 0; i < 10; i++ {
        fmt.Println(n, ":", i)
        amt := time.Duration(rand.Intn(250))
        time.Sleep(time.Millisecond * amt)
    }
}

func main() {
    for i := 0; i < 10; i++ {
        go f(i)
    }
    var input string
    fmt.Scanln(&input)
}
```

::: Prog. Concorrente in Go (1/10)

Una goroutine consiste nell'invocare una funzione con la parola chiave go, e realizza il fork di un thread.

```
package main

import (
    "fmt"
    "time"
    "math/rand"
)

func f(n int) {
    for i := 0; i < 10; i++ {
        fmt.Println(n, ":", i)
        amt := time.Duration(rand.Intn(250))
        time.Sleep(time.Millisecond * amt)
    }
}

func main() {
    for i := 0; i < 10; i++ {
        go f(i)
    }
    var input string
    fmt.Scanln(&input)
}
```

Un canale fa da passacarte tra threads ed è indicato con chan seguita dal tipo di dati scambiabili.

L'operatore <- è utilizzato per inviare/ricevere dati sul canale.

```
package main

import (
    "fmt"
    "time"
)

func pinger(c chan string) {
    for i := 0; ; i++ {
        c <- "ping"
    }
}

func printer(c chan string) {
    for {
        msg := <- c
        fmt.Println(msg)
        time.Sleep(time.Second * 1)
    }
}

func main() {
    var c chan string = make(chan string)

    go pinger(c)
    go printer(c)

    var input string
    fmt.Scanln(&input)
}
```

::: Prog. Concorrente in Go (1/10)

Una goroutine consiste nell'invocare una funzione con la parola chiave go, e realizza il fork di un thread.

```
package main

import (
    "fmt"
    "time"
    "math/rand"
)

func f(c chan string) {
    for i := 0; i < 10; i++ {
        amt := rand.Intn(100)
        time.Sleep(time.Millisecond * amt)
    }
}

func main() {
    for i := 0; i < 10; i++ {
        go f(i)
    }
    var input string
    fmt.Scanln(&input)
}
```

I canali sono un tipo di riferimento, e se si assegna una variabile chan a un'altra, entrambe le variabili accedono allo stesso canale. Significa anche che bisogna impiegare la funzione make per allocare uno.

Un canale fa da passacarte tra threads ed è indi-

L'operatore <- è utilizzato per inviare/ricevere dati sul canale.

```
package main

import (
    "fmt"
    "time"
)

func pinger(c chan string) {
    for i := 0; i < 10; i++ {
        c <- "ping"
    }
}

func main() {
    var c chan string = make(chan string)
    go pinger(c)
    go printer(c)

    var input string
    fmt.Scanln(&input)
}
```

::: Prog. Concorrente in Go (1/10)

Un canale

La freccia punta nella direzione del flusso di dati. Per impostazione predefinita, la comunicazione è sincrona: due goroutine che scambiano dati attraverso un canale si sincronizzano al momento della comunicazione.

```
package main

import (
    "fmt"
    "time"
    "math/rand"
)

func f(n int) {
    for i := 0; i < 10; i++ {
        fmt.Println(n, ":", i)
        amt := time.Duration(rand.Intn(250))
        time.Sleep(time.Millisecond * amt)
    }
}

func main() {
    for i := 0; i < 10; i++ {
        go f(i)
    }
    var input string
    fmt.Scanln(&input)
}
```

Un canale fa da passacarte tra threads ed è indicato con chan seguita dal tipo di dati scambiabili.

L'operatore <- è utilizzato per inviare/ricevere dati sul canale.

```
package main

chan string) {
    i++ {
    }

func printer(c chan string) {
    for {
        msg := <- c
        fmt.Println(msg)
        time.Sleep(time.Second * 1)
    }
}

func main() {
    var c chan string = make(chan string)

    go pinger(c)
    go printer(c)

    var input string
    fmt.Scanln(&input)
}
```

::: Prog. Concorrente in Go (2/10)

```
func pump() chan int {  
    ch := make(chan int)  
    go func() {  
        for i := 0; ; i++ { ch <- i }  
    }()  
    return ch  
}  
  
stream := pump()  
fmt.Println(<-stream) // prints 0
```

"Function returning channel» è un idioma di programmazione molto comune in Go.

::: Prog. Concorrente in Go (2/10)

```
func pump() chan int {  
    ch := make(chan int)  
    go func() {  
        for i := 0; ; i++ { ch <- i }  
    }()  
    return ch  
}  
  
stream := pump()  
  
fmt.Println(<-stream) // prints 0
```

```
func suck(ch chan int) {  
    go func() {  
        for v := range ch { fmt.Println(v) }  
    }()  
}  
  
suck(pump())
```

"Function returning channel" è un idioma di programmazione molto comune in Go. La clausola di intervallo sui cicli for accetta un canale come operando, nel qual caso i cicli for vengono eseguiti sui valori ricevuti dal canale.

::: Prog. Concorrente in Go (3/10)

Come fa range a sapere quando il canale ha finito di fornire i dati?

Il mittente chiama la funzione built-in close:

close(ch)

Il ricevitore verifica se il mittente ha chiuso il canale usando "virgola ok":

val, ok := <-ch

Il risultato è (valore, vero) mentre i valori sono disponibili; una volta che il canale è chiuso e svuotato, il risultato è (zero, falso).

```
for value := range <-ch {  
    use(value)  
}
```



```
for {  
    value, ok := <-ch  
    if !ok { break }  
    use(value)  
}
```

::: Prog. Concorrente in Go (3/10)

Come fa range a sapere quando il canale ha finito di fornire i dati?

Il mittente chiama la funzione built-in close:

```
close(ch)
```

Il ricevitore verifica se il mittente ha chiuso il canale usando "virgola ok":

```
val, ok := <-ch
```

Il risultato è (valore, vero) mentre i valori sono disponibili; una volta che il canale è chiuso e svuotato, il risultato è (zero, falso).

Usare close solo quando è necessario per segnalare al ricevitore che non arriveranno più valori. Il più delle volte, non è necessario un esplicito close; non è analogo alla chiusura di un file. I canali vengono chiusi a prescindere dal garbage collector.

::: Prog. Concorrente in Go (4/10)

Un tipo di canale può essere annotato per specificare che può solo inviare o solo ricevere. Tutti i canali vengono creati bidirezionali, ma possiamo assegnarli a variabili di canale direzionali.

```
func sink(ch <-chan int) {
    for { <-ch }
}

func source(ch chan<- int) {
    for { ch <- 1 }
}

c := make(chan int) // bidirectional
go source(c)
go sink(c)
```

::: Prog. Concorrente in Go (4/10)

I canali sincroni sono senza buffer. Gli invii non vengono completati finché un destinatario non ha accettato il valore.

```
c := make(chan int)
go func() {
    time.Sleep(60*1e9)
    x := <-c
    fmt.Println("received", x)
}()
fmt.Println("sending", 10)
c <- 10
fmt.Println("sent", 10)
Output:
sending 10 (happens immediately)
sent 10 (60s later, these 2 lines appear)
received 10
```

::: Prog. Concorrente in Go (4/10)

I canali sincroni sono senza buffer. Gli invii non vengono completati finché un destinatario non ha accettato il valore.

```
c := make(chan int, 50)
go func() {
    time.Sleep(60*1e9)
    x := <-c
    fmt.Println("received", x)
}()
fmt.Println("sending", 10)
c <- 10
fmt.Println("sent", 10)
Output:
sending 10 (happens immediately)
sent 10 (now)
received 10
```

Un canale asincrono bufferizzato viene creato indicando a make il numero di elementi nel buffer.

Si noti che la dimensione del buffer, o anche la sua esistenza, non fa parte del tipo del canale, ma solo del valore. Questo codice è quindi consentito, sebbene pericoloso:

```
buf = make(chan int, 1)
unbuf = make(chan int)
buf = unbuf
unbuf = buf
```

::: Prog. Concorrente in Go (5/10)

Select è una struttura di controllo in Go analoga a un'istruzione switch di comunicazione. Ogni case deve essere una comunicazione, inviata o ricevuta.

```
ci, cs := make(chan int), make(chan string)
select {
    case v := <-ci:
        fmt.Printf("received %d from ci\n", v)
    case v := <-cs:
        fmt.Printf("received %s from cs\n", v)
}
```

Select esegue un case eseguibile a caso. Se nessun case è eseguibile, si blocca finché uno non lo è. Una clausola default è sempre eseguibile.

::: Prog. Concorrente in Go (6/10)

È possibile implementare un generatore random di numeri con un select:

```
c := make(chan int)
go func() {
    for {
        fmt.Println(<-c)
    }
}()
for {
    select {
        case c <- 0: // no statement, no fall through
        case c <- 1:
    }
}
```

::: Prog. Concorrente in Go (6/10)

È possibile implementare un generatore random di numeri con un select:

```
c := make(chan int)
go func() {
    for {
        fmt.Println(<-c)
    }
}()
for {
    select {
        case c <- 0: // no statement, no fall through
        case c <- 1:
    }
}
```

Stampa a video 0 1 1 0 0 1 1 1 0 1 ...

::: Prog. Concorrente in Go (7/10)

Può una comunicazione riuscire in un dato intervallo?

Il tempo del pacchetto contiene la funzione After:

```
func After(ns int64) <-chan int64
```

Fornisce un valore (l'ora corrente) sul canale restituito dopo l'intervallo specificato. Usalo in select per implementare il timeout:

```
select {
    case v := <-ch:
        fmt.Println("received", v)
    case <-time.After(30*1e9):
        fmt.Println("timed out after 30 seconds")
}
```

I canali sono valori di prima classe, il che significa che possono essere inviati tramite canali. Questa proprietà rende facile scrivere un multiplexer di servizio poiché il cliente può fornire, insieme alla sua richiesta, il canale su cui rispondere.

```
chanOfChans := make(chan chan int)
```

::: Prog. Concorrente in Go (8/10)

```
type request struct {
    a, b int
    replyc chan int
}

type binOp func(a, b int) int

func run(op binOp, req *request) {
    req.replyc <- op(req.a, req.b)
}

func server(op binOp, service <-chan *request) {
    for {
        req := <-service // requests arrive here
        go run(op, req) // don't wait for op
    }
}
```

::: Prog. Concorrente in Go (8/10)

```
type request struct {
    a, b int
    replyc chan int
}
type binOp func(a,
func run(op binOp,
    req.replyc <- op(r)
}
func server(op binOp) {
    for {
        req := <-service
        go run(op, req)
    }
}
```

```
func startServer(op binOp) chan<- *request {
    service := make(chan *request)
    go server(op, req)
    return service
}
adderChan := startServer(
    func(a, b int) int { return a + b }
)
req1 := &request{7, 8, make(chan int)}
req2 := &request{17, 18, make(chan int)}
Le richieste sono pronte, vanno inviate:
adderChan <- req1
adderChan <- req2
```

::: Prog. Concorrente in Go (9/10)

Ipotizziamo un programma che deve eseguire un compito computazionale pesante e restituire una risposta a un richiedente in modo sincrono, come una sequenza di due funzioni:

- `getSingleHash()`, che prende dei dati come input e calcola l'output come `crc32(data) + "~" + crc32(md5 (data))`, dove `+` significa concatenazione di stringhe;
- `getMultiHash()`, che restituisce l'output come `crc32 (i + data))`, dove `i` è un numero di iterazione compreso tra 0 e 6.

```
// gets a crc32(data) + "~" + crc32(md5(data)) value
func getSingleHash(data string) string {
    var result string
    crc32Hash1 := getCrc32(data)
    md5Hash := getMd5(data)
    crc32Hash2 := getCrc32(md5Hash)
    result = crc32Hash1 + "~" + crc32Hash2
    return result
}
```

```
// gets a crc32(i + data) where i = 0..5
func getMultiHash(data string) string {
    var result string
    for i := 0; i < 6; i++ {
        sI := strconv.Itoa(i)
        result += getCrc32(sI + data)
    }
    return result
}
```

::: Prog. Concorrente in Go (10/10)

Consideriamo una implementazione sequenziale:



Il codice sarà racchiuso in un metodo invocabile come segue:

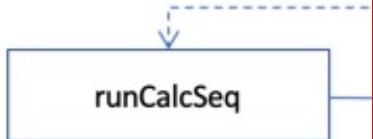
```
func runCalcSeq(stub shim.ChaincodeStubInterface, args []string) peer.Response {
    var resAsBytes []byte
    var hash1, hash2 string

    for _, val := range args {
        hash1 = getSingleHash(val)
        hash2 = getMultiHash(hash1)
    }
    resAsBytes = []byte(hash2)

    return shim.Success(resAsBytes)
}
```

::: Prog. Concorrente in Go (10/10)

Consideriamo una implementazione sequenziale:



```
==== RUN TestSeq
Call: runCalcSeq ( 0 )
Code execution time for 'runCalcSeq' is 8.027733723s
RetCode: 200
RetMsg:
Payload: 29568666068035183841425683795340791879727309630931025356555
Received result:
29568666068035183841425683795340791879727309630931025356555
--- PASS: TestSeq (8.03s)
    main_test.go:20: The code execution time is: [8.027805802s]
PASS
ok      GoBasicHLPipelines/src  8.810s
```

The output of a test run. It shows the command "TestSeq", the call to "runCalcSeq", the execution time (8.027733723s), the return code (200), the payload (a large hex string), the received result (the same hex string), a pass message, the execution time (8.027805802s highlighted with a red box), a PASS message, and finally "ok" and the file path "GoBasicHLPipelines/src" with a timestamp of 8.810s.

Il codice sarà racchiuso in

```
func runCalcSeq(stub shim.ChaincodeStubInterface, args []string) peer.Response {
    var resAsBytes []byte
    var hash1, hash2 string

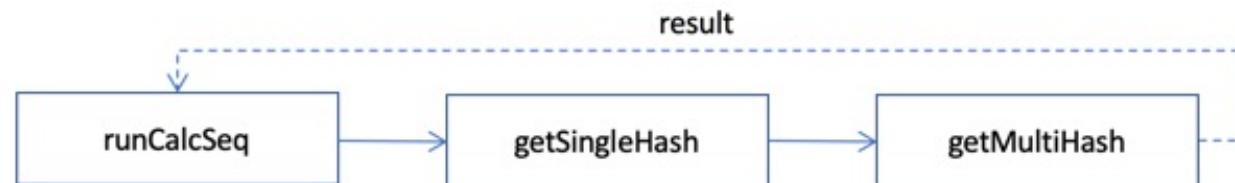
    for _, val := range args {
        hash1 = getSingleHash(val)
        hash2 = getMultiHash(hash1)
    }
    resAsBytes = []byte(hash2)

    return shim.Success(resAsBytes)
}
```

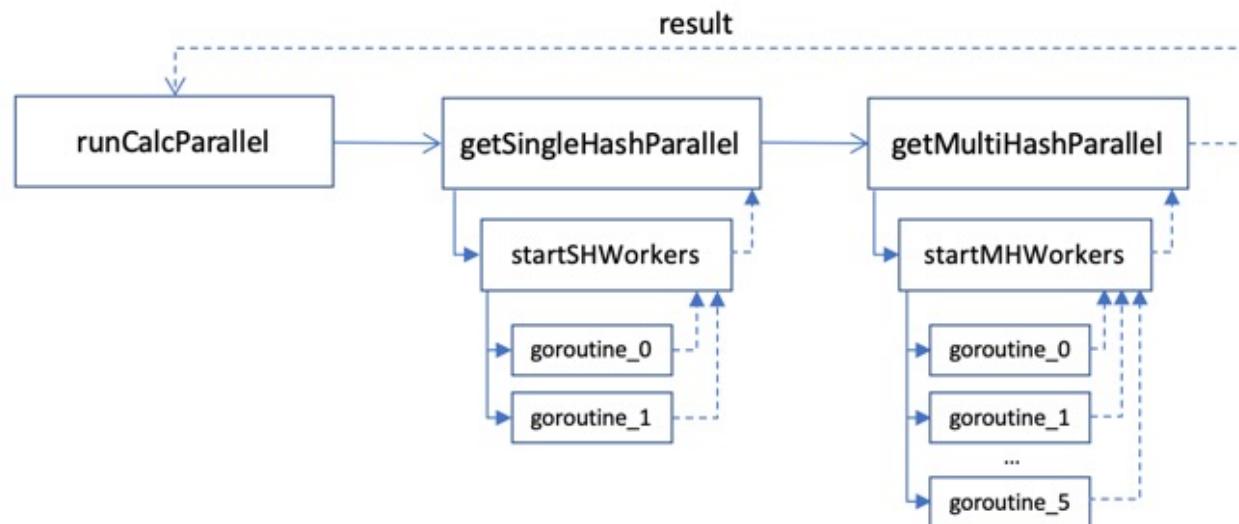
A Go code block enclosed in a red box. It defines a function "runCalcSeq" that takes a "shim.ChaincodeStubInterface" and a slice of strings "args" as parameters. It initializes two variables, "resAsBytes" and "hash1", and then iterates over "args" to calculate "hash1" and "hash2" using "getSingleHash" and "getMultiHash" functions respectively. Finally, it returns a "shim.Success" response with "resAsBytes". A red arrow points from the end of the code block to the execution time in the terminal output above.

::: Prog. Concorrente in Go (10/10)

Consideriamo una implementazione sequenziale:



Consideriamo una implementazione parallela:



::: Prog. Concorrente in Go (10/10)

```
func startSingleHashWorkers(data string, out chan<- string) {
    wg := &sync.WaitGroup{}
    wg.Add(2)

    var left, right string

    go func() {
        defer wg.Done()
        left = getCrc32(data)
   }()

    go func() {
        defer wg.Done()
        hash := getMd5(data)
        right = getCrc32(hash)
   }()

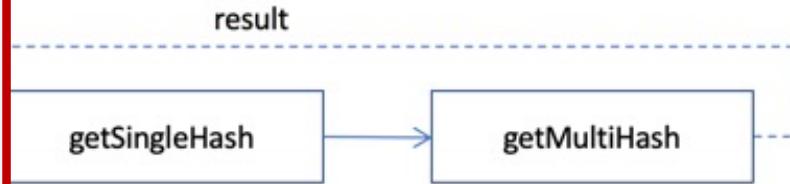
    wg.Wait()
    result := left + "~" + right
    out <- result
}

// gets a crc32(data) + "~" + crc32(md5(data)) value in parallel mode
func getSingleHashParallel(in string) string {
    out := make(chan string)

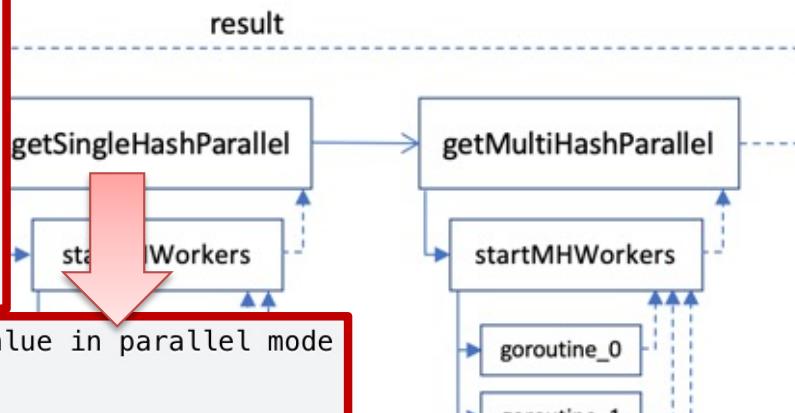
    go func(data string) {
        startSingleHashWorkers(data, out)
   }(in)

    return <-out
}
```

ntazione sequenziale:



ntazione parallela:



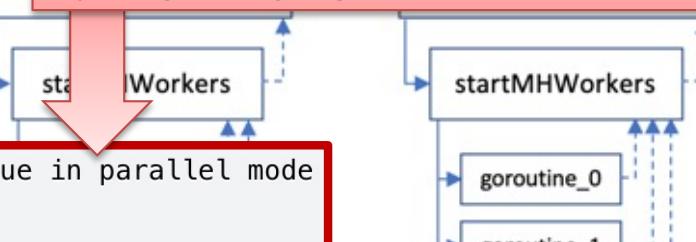
Un canale chiamato "out" viene creato e passato alla funzione scheduler, che istanzia due goroutine per il calcolo delle due porzioni del risultato. Sull'ultima stringa restituiamo solo un valore fornito dal canale.

::: Prog. Concorrente in Go (10/10)

```
func startSingleHashWorkers(data string, out chan<- string) {  
    wg := &sync.WaitGroup{  
        wg.Add(2)  
  
    var left, right string  
  
    go func() {  
        defer wg.Done()  
        left = getCrc32(data)  
    }()  
  
    go func() {  
        defer wg.Done()  
        hash := getMd5(data)  
        right = getCrc32(hash)  
    }()  
  
    wg.Wait()  
    result := left + "~" + right  
    out <- result  
}  
  
// gets a crc32(data) + "~" + crc32(md5(data)) value in parallel mode  
func getSingleHashParallel(in string) string {  
    out := make(chan string)  
  
    go func(data string) {  
        startSingleHashWorkers(data, out)  
    }(in)  
  
    return <-out  
}
```

Introduzione sequenziale:

Si usa un oggetto di sincronizzazione WaitGroup per attendere che entrambe le goroutine finiscano attendendo su `wg.Wait()`. Ogni istanza di WaitGroup ha un contatore interno, che viene aumentato dal metodo `wg.Add()` e diminuito dal metodo `wg.Done()`. La parola chiave "defer" ritardare l'esecuzione della funzione o del metodo o di un metodo anonimo fino al ritorno delle funzioni vicine.



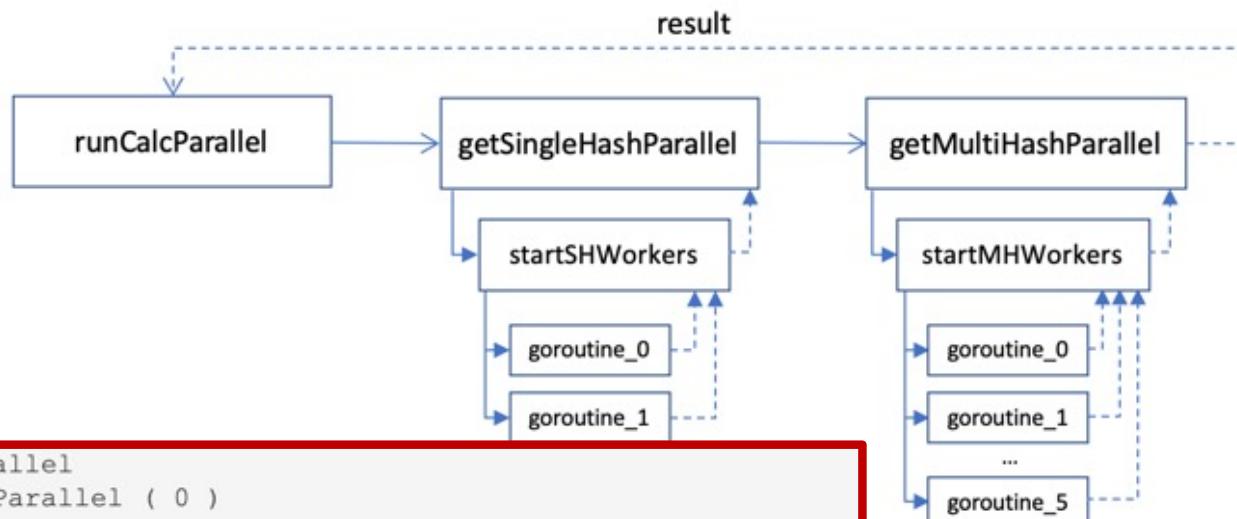
Un canale chiamato "out" viene creato e passato alla funzione scheduler, che istanzia due goroutine per il calcolo delle due porzioni del risultato. Sull'ultima stringa restituiamo solo un valore fornito dal canale.

::: Prog. Concorrente in Go (10/10)

Consideriamo una implementazione sequenziale:



Consideriamo una implementazione parallela:

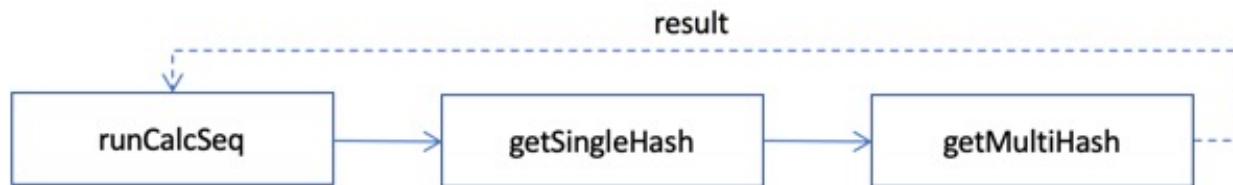


```
==== RUN TestParallel
Call: runCalcParallel ( 0 )
RetCode: 200
RetMsg:
Payload: 29568666068035183841425683795340791879727309630931025356555
Received result:
29568666068035183841425683795340791879727309630931025356555
--- PASS: TestParallel (2.02s)
    main_test.go:35: The code execution time is: 2.021541811s
PASS
```

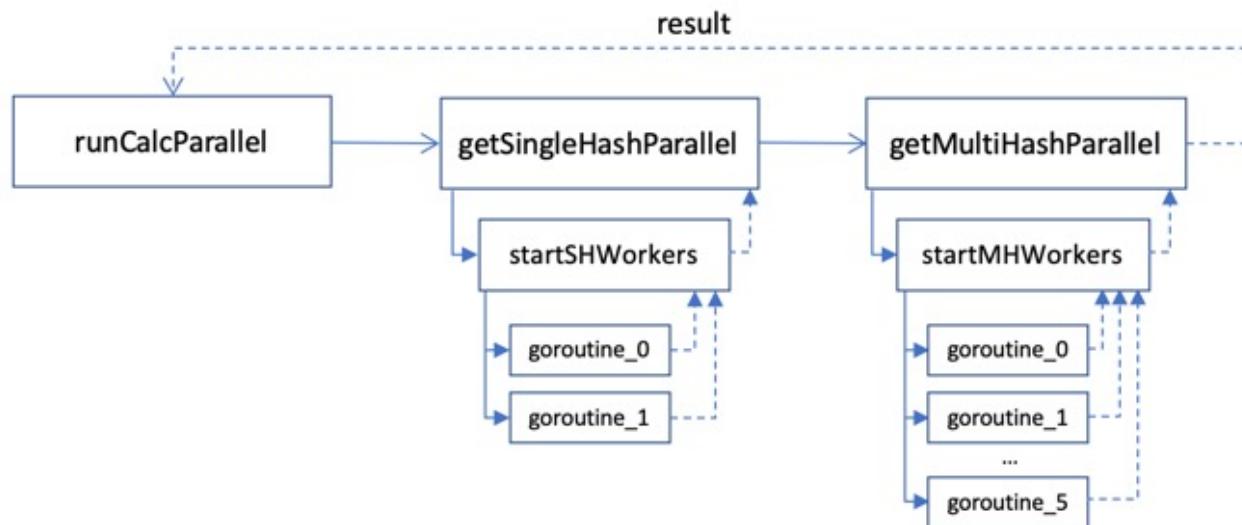


::: Prog. Concorrente in Go (10/10)

Consideriamo una implementazione sequenziale:



Consideriamo una implementazione parallela:



Se un programma concorrente non viene gestito in modo appropriato, si verificherà facilmente un problema di race condition.

::: Prog. Asynch in Go (1/6)

Golang è un linguaggio di programmazione che ha potenti funzionalità come goroutine e channel in grado di gestire molto bene le attività sia sincrone che asincrone. Le goroutine non sono thread del sistema operativo, ed è per questo che è possibile crearne in gran numero senza molto sovraccarico, con uno stack di soli 2 KB.

Async/Await è un pattern di programmazione concorrente che troviamo in vari linguaggi di programmazione come C# e Java, ma non in Go perché non è difficile replicare lo schema con goroutine e canali.

Prendiamo un esempio di attendere un risultato da una funzione asincrona.

::: Prog. Asynch in Go (2/6)

In JavaScript

```
const longRunningTask = async () => {
    // simulate a workload
    sleep(3000);
    return Math.floor(Math.random() * Math.floor(100));
};
```

```
const r = await longRunningTask();
console.log(r);
```

Vediamo un qualcosa di simile in Go.

::: Prog. Asynch in Go (3/6)

```
func longRunningTask() <-chan int32 {
    r := make(chan int32)

    go func() {
        defer close(r)

        // simulate a workload
        time.Sleep(time.Second * 3)
        r <- rand.Int31n(100)
    }()

    return r
}

func main() {
    r := <-longRunningTask()
    fmt.Println(r)
}
```

::: Prog. Asynch in Go (4/6)

Per dichiarare una funzione "asincrona" in Go:

- Il tipo restituito è <-chan ReturnType.
- All'interno della funzione, si crea un canale con make(chan ReturnType) e si restituisce il canale creato alla fine della funzione.
- Si avvia una goroutine anonima con go func() {...} e si implementa la logica della funzione all'interno di quella funzione anonima.
- Si restituisce il risultato inviando il valore al canale.
- All'inizio della funzione anonima, si aggiunge defer close(r) per chiudere il canale una volta terminato.
- Per "attendere" il risultato, basta leggere il valore dal canale tramite v := <- fn().

::: Prog. Asynch in Go (5/6)

Promise.all() – è molto comune attendere i risultati da molti task asincroni.

```
func longRunningTask() <-chan int32 {  
    r := make(chan int32)  
    go func() {  
        defer close(r)  
        // simulate a workload  
        time.Sleep(time.Second * 3)  
        r <- rand.Int31n(100)  
    }()  
    return r  
}  
func main() {  
    aCh, bCh, cCh := longRunningTask(), longRunningTask(), longRunningTask()  
    a, b, c := <-aCh, <-bCh, <-cCh  
    fmt.Println(a, b, c)  
}
```

::: Prog. Asynch in Go (5/6)

Promise.all() – è molto comune attendere i risultati da molti task asincroni.

```
func longRunningTask() <-chan int32 {  
    r := make(chan int32)  
    go func() {  
        defer close(r)  
        // simulate a workload  
        time.Sleep(time.Second * 3)  
        r <- rand.Int31n(100)  
    }()  
    return r  
}  
func main() {  
    aCh, bCh, cCh := longRunningTask(), longRunningTask(), longRunningTask()  
    a, b, c := <-aCh, <-bCh, <-cCh  
    fmt.Println(a, b, c)  
}
```

Non possiamo fare <-longRun(), <-longRun(), <-longRun(), perchè longRun() verrebbe eseguito in serie uno per uno invece che in parallelo in maniera asincrona.

aCh, bCh, cCh := longRunningTask(), longRunningTask(), longRunningTask()

a, b, c := <-aCh, <-bCh, <-cCh

::: Prog. Asynch in Go (6/6)

Promise.race() – A volte, dei dati possono essere ricevuti da diverse fonti per evitare latenze elevate, oppure ci sono casi in cui vengono generati più risultati ma sono equivalenti e viene consumata solo la prima risposta.

```
func one() <-chan int32 {  
    r := make(chan int32)  
    go func() {  
        defer close(r)  
        // simulate a workload  
        time.Sleep(time.Millisecond *  
            time.Duration(rand.Int63n(2000)))  
        r <- 1  
    }()  
    return r  
}
```

```
func two() <-chan int32 {  
    r := make(chan int32)  
    go func() {  
        defer close(r)  
        // simulate a workload  
        time.Sleep(time.Millisecond *  
            time.Duration(rand.Int63n(1000)))  
        time.Sleep(time.Millisecond *  
            time.Duration(rand.Int63n(1000)))  
        r <- 2  
    }()  
    return r  
}
```

::: Prog. Asynch in Go (6/6)

Promise.race() – A volte, dei dati possono essere ricevuti da diverse fonti per evitare latenze elevate, oppure ci sono casi in cui vengono generati più risultati ma sono equivalenti e viene consumata solo la prima risposta.

```
func main() {  
    var r int32  
    select {  
        case r = <-one():  
        case r = <-two():  
    }  
  
    fmt.Println(r)  
}
```

::: Prog. Asynch in Go (6/6)

Promise.race() – A volte, dei dati possono essere ricevuti da diverse fonti per evitare latenze elevate, oppure ci sono casi in cui vengono generati più risultati ma sono equivalenti e viene consumata solo la prima risposta.

```
func main() {  
    var r int32  
    select {  
        case r = <-one():  
        case r = <-two():  
    }  
  
    fmt.Println(r)  
}
```

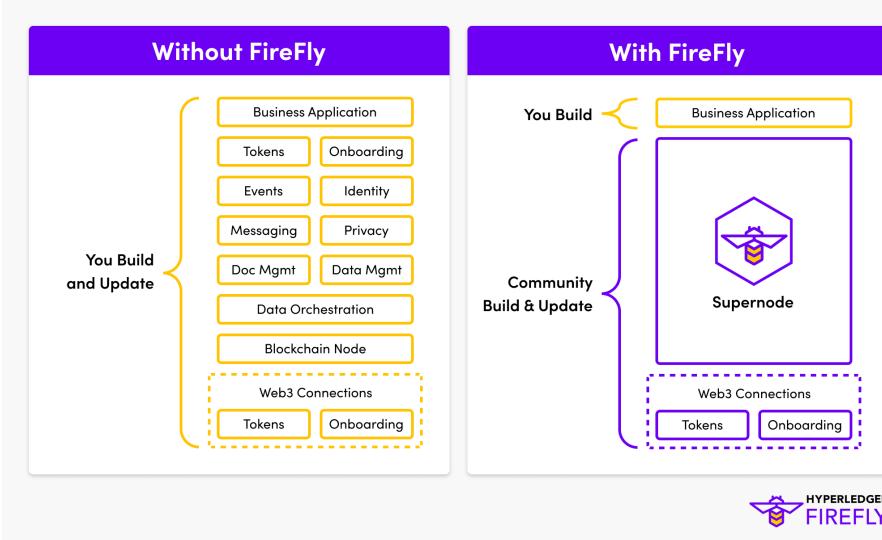
Poiché il modello di propagazione degli errori di Go è molto diverso da altri linguaggi, non esistono modi chiari per replicare Promise.then() e Promise.catch(). In Go, gli errori vengono restituiti come valori, non ci sono eccezioni. Pertanto, se una funzione può fallire, + meglio considerare di cambiare il return <-chan ReturnType in <-chan ReturnAndErrorType, che è una struttura che contiene sia il risultato che l'errore.



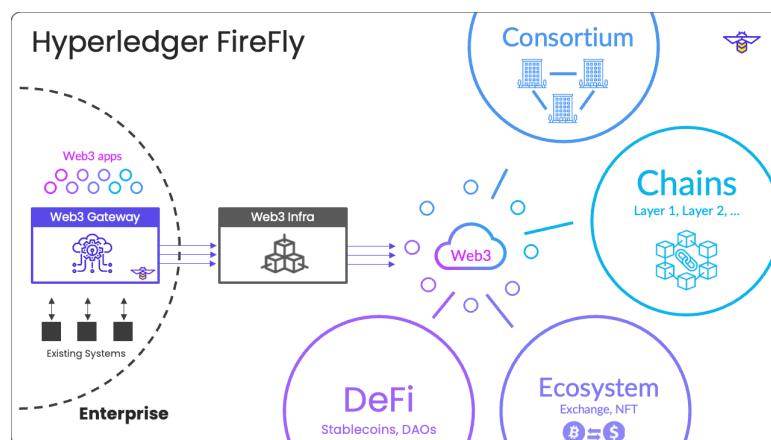
Principali Piattaforme – Hyperledger FireFly

::: Hyperledger FireFly (1/13)

Nell'ultimo decennio di progetti blockchain aziendali, gli sviluppatori si sono resi conto di aver bisogno di molto più di un nodo blockchain affinché i loro progetti abbiano successo.

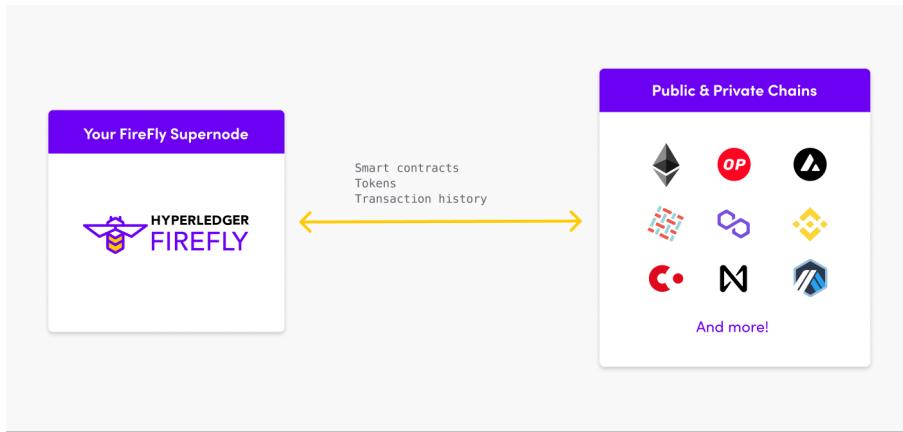


Lo stack di sviluppo necessario per un'applicazione Web3 di livello aziendale è sofisticato quanto lo stack richiesto per le applicazioni Web 2.0 precedenti.

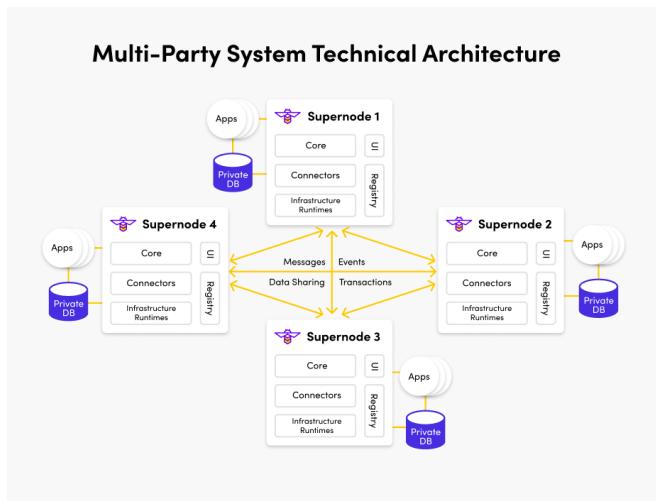


Hyperledger FireFly è un gateway per Web3, non è un'altra implementazione blockchain, piuttosto integra tutti i diversi tipi di tecnologie decentralizzate esistenti in Web3.

::: Hyperledger FireFly (2/13)

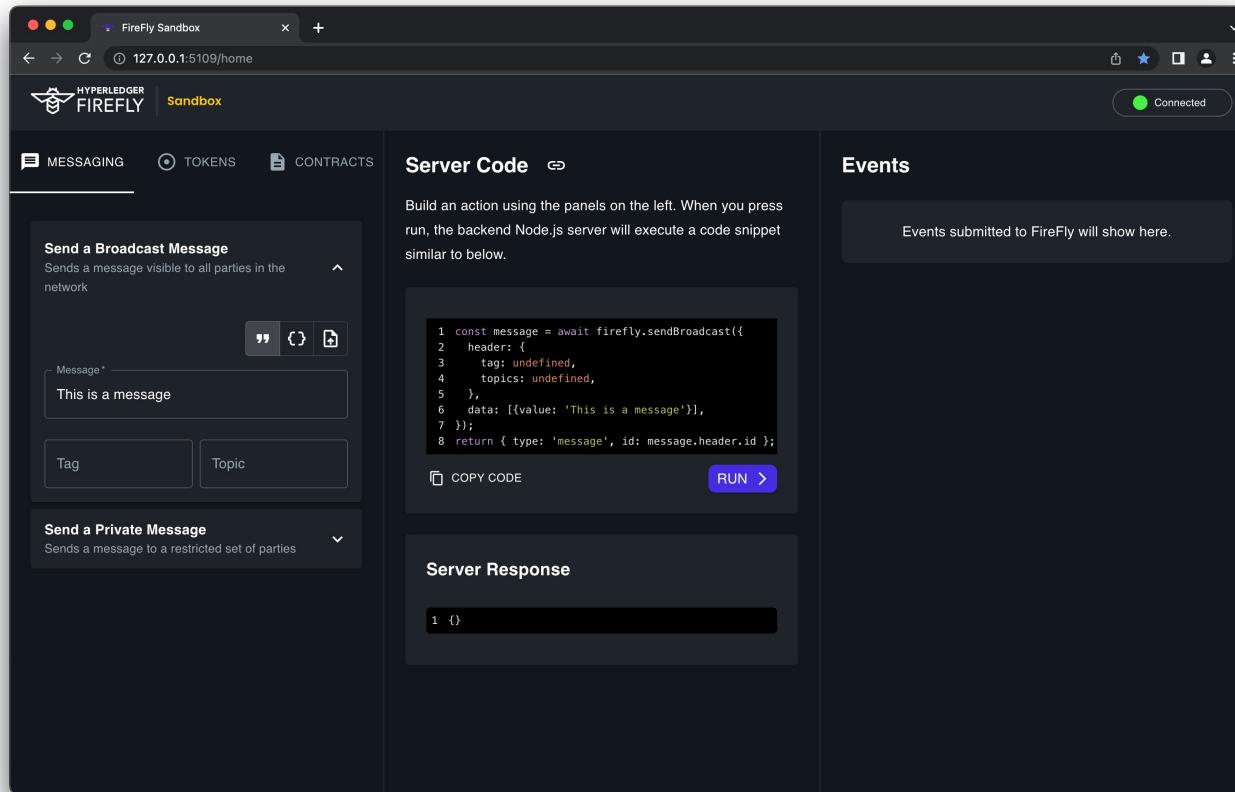


La modalità Web3 Gateway consente di interagire con qualsiasi applicazione Web3, indipendentemente dal fatto che Hyperledger FireFly sia utilizzato da altri membri.

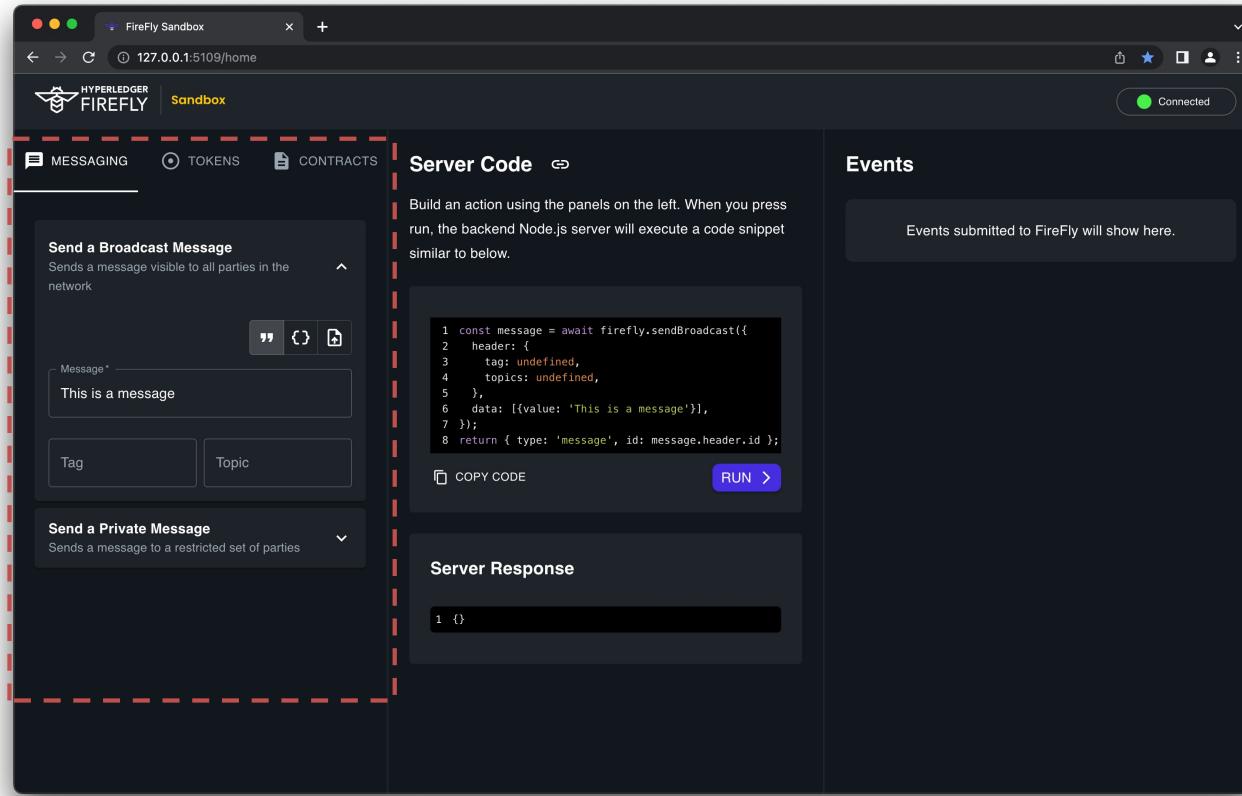


La modalità multiparty viene utilizzata per creare un runtime di applicazione comune, distribuito da più partecipanti aziendali. Successivamente sono necessari protocolli adeguati per supportare l'interazione dei supernodi in ciascun partecipante.

::: Hyperledger FireFly (3/13)

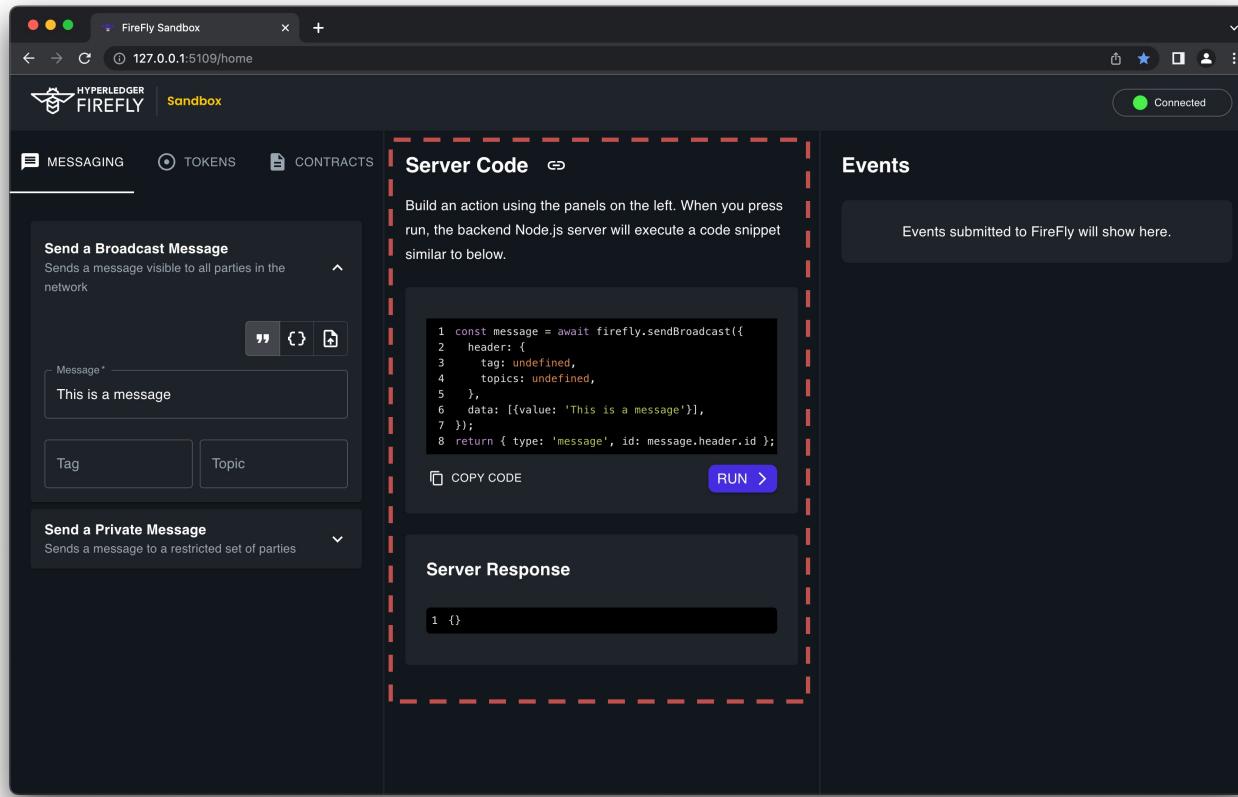


::: Hyperledger FireFly (3/13)



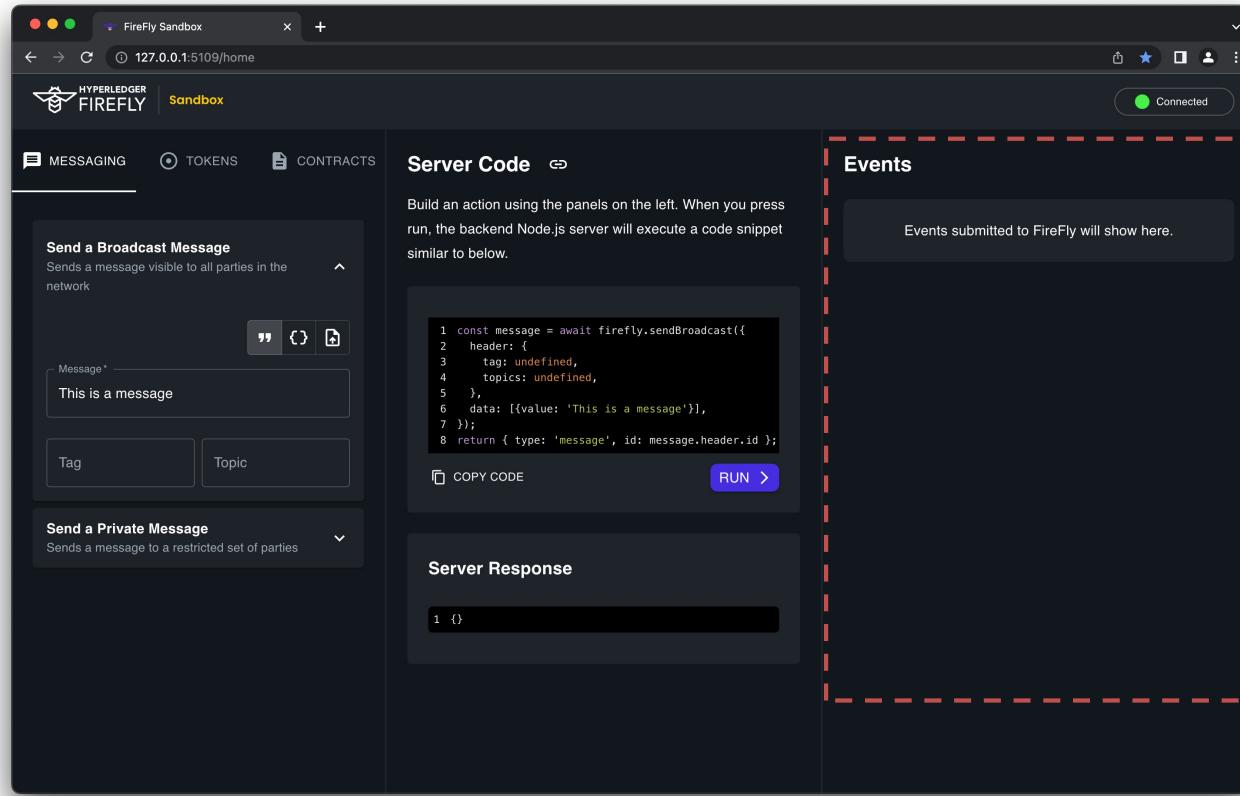
Sul lato sinistro della pagina si può compilare semplici campi del modulo per creare messaggi e altro ancora.

::: Hyperledger FireFly (3/13)



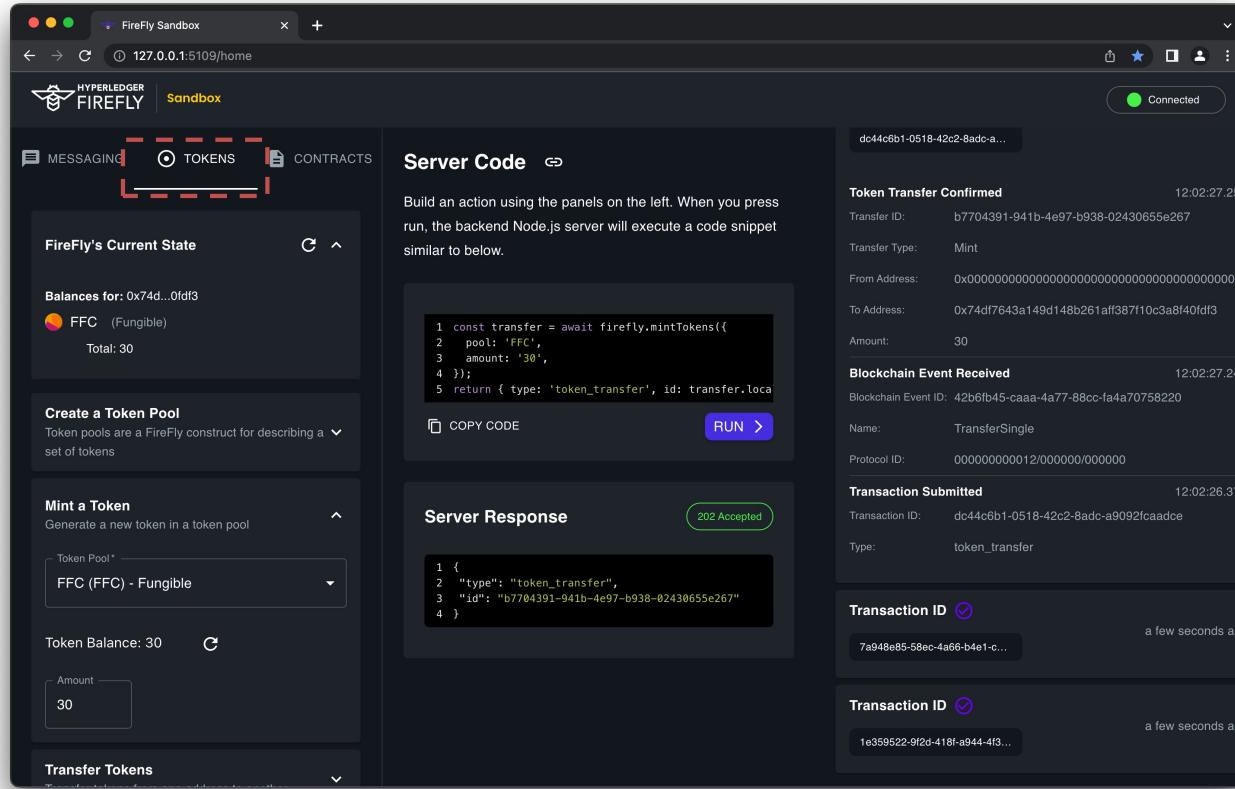
La sezione centrale è un'anteprima del codice generato e dei risultati della sua esecuzione.

::: Hyperledger FireFly (3/13)



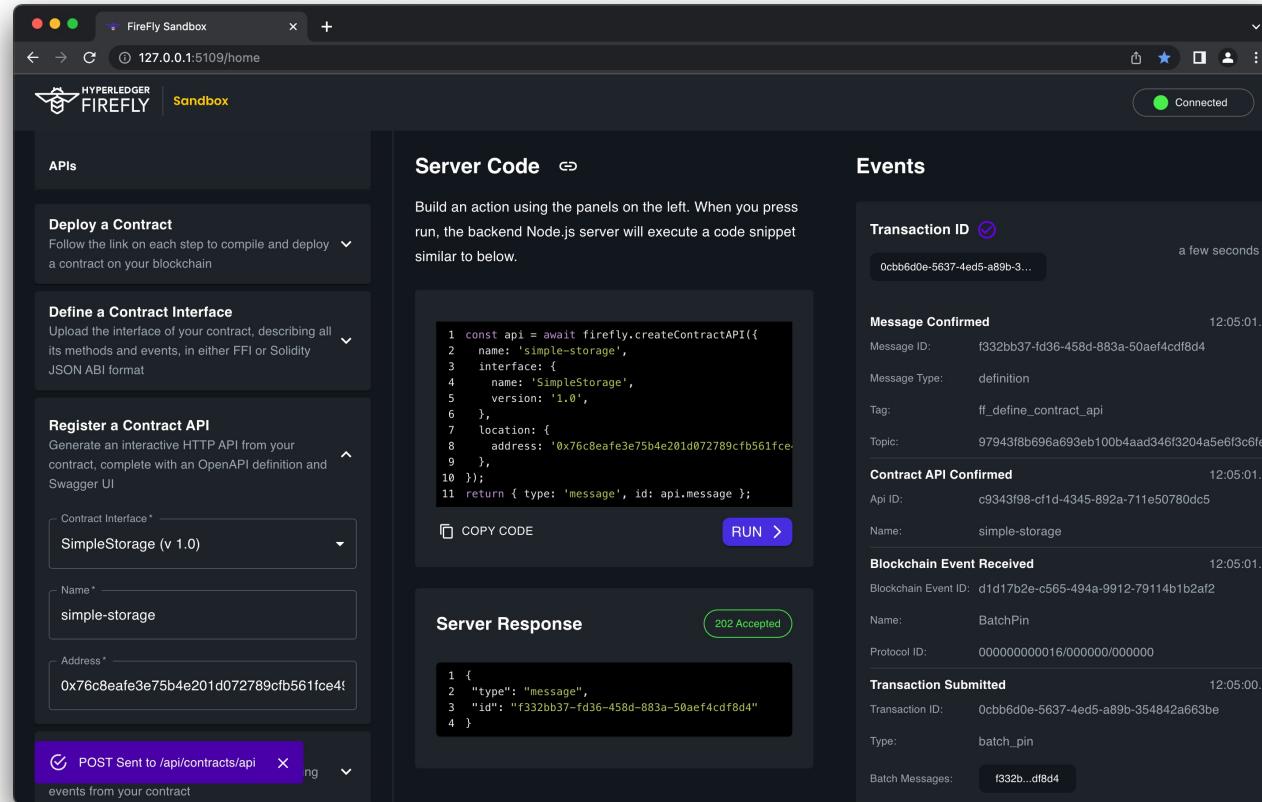
Sul lato destro c'è un flusso di eventi ricevuti su una connessione WebSocket che il backend ha aperto su FireFly.

::: Hyperledger FireFly (3/13)



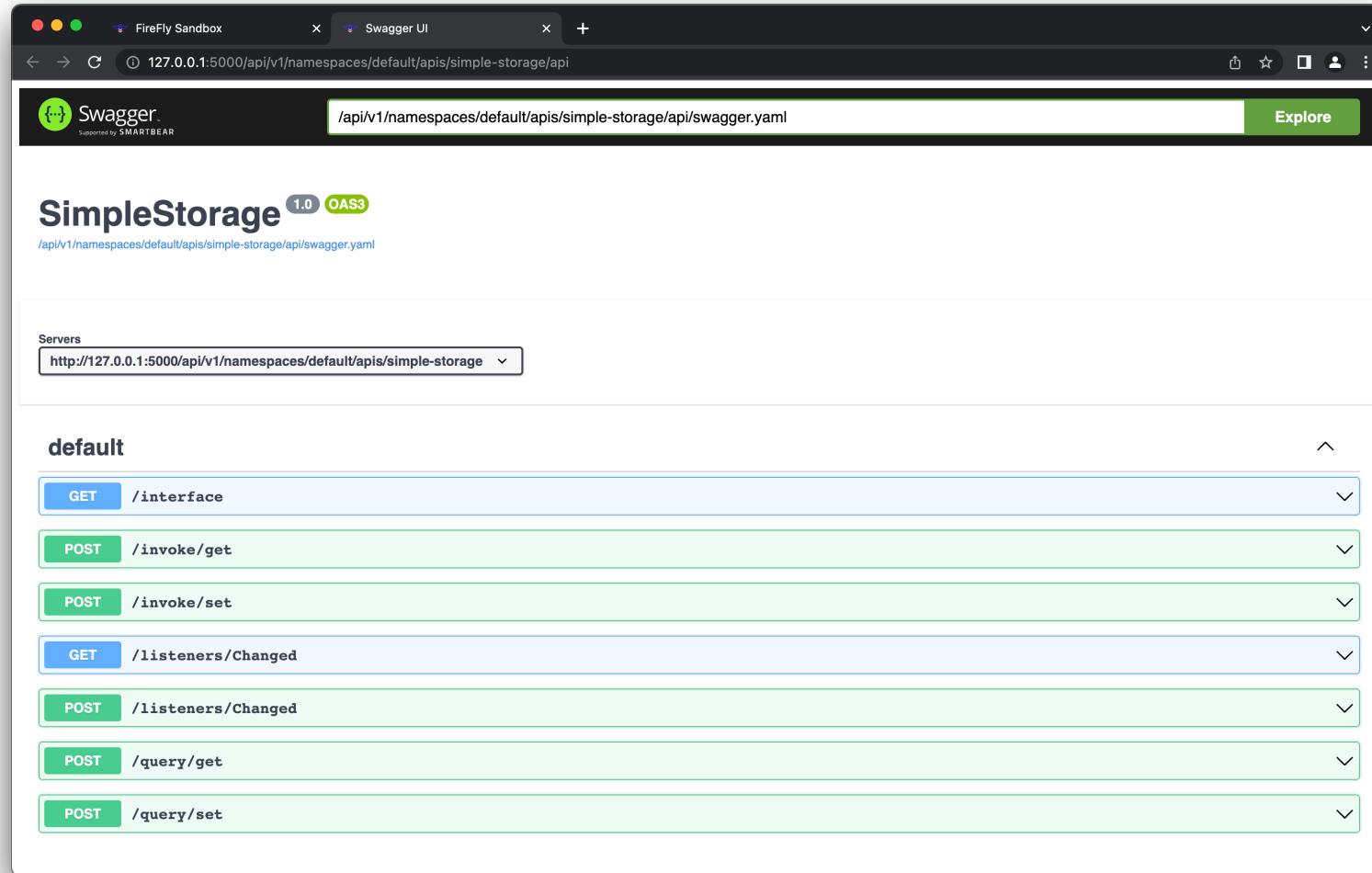
La scheda Token serve a creare pool di token e coniare, masterizzare o trasferire token, sia token fungibili che non fungibili (NFT).

::: Hyperledger FireFly (3/13)



La sezione Contratti della Sandbox consente di interagire con contratti intelligenti personalizzati, direttamente dal browser web.

::: Hyperledger FireFly (3/13)



::: Hyperledger FireFly (4/13)

L'API unificata di FireFly crea una programmazione applicativa coerente, indipendentemente dalla specifica blockchain sottostante. FireFly definisce i seguenti costrutti:

- **Interfaccia contrattuale:** FireFly definisce un modo comune e indipendente dalla blockchain per descrivere i contratti intelligenti, denominato interfaccia contrattuale. Un'interfaccia di contratto è scritta nel formato FireFly Interface (FFI). È un semplice documento JSON che ha un nome, uno spazio dei nomi, una versione, un elenco di metodi e un elenco di eventi.

Per le blockchain che offrono un DSL che descrive l'interfaccia del contratto intelligente, come l'ABI di Ethereum, FireFly offre uno strumento pratico per convertire il DSL nel formato FFI.

::: Hyperledger FireFly (5/13)

- API HTTP: sulla base di un'interfaccia di contratto, FireFly definisce ulteriormente un'API HTTP per il contratto intelligente, che è completo di una specifica OpenAPI e dell'interfaccia utente Swagger. Un'API HTTP definisce un percorso root /invoke per inviare transazioni e un percorso root /query per inviare richieste di query per leggere nuovamente lo stato.

Il modo in cui le richieste di invocazione e di query vengono interpretate nelle richieste blockchain native è specifico del connettore della blockchain. Il connettore Ethereum traduce le chiamate /invoke in richieste JSON-RPC eth_sendTransaction, mentre le chiamate /query vengono tradotte in richieste JSON-RPC eth_call.

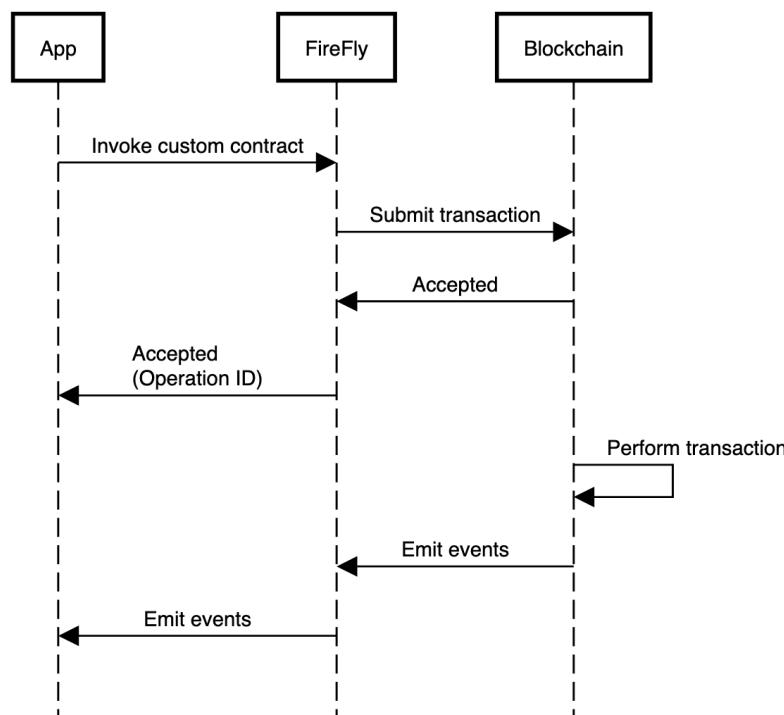
::: Hyperledger FireFly (6/13)

- Ascoltatore di eventi Blockchain: indipendentemente dal design specifico di una blockchain, l'elaborazione delle transazioni è sempre asincrona: una transazione viene inviata e il client che la invia riceve una conferma che è stata accettata per un'ulteriore elaborazione. Il client ascolta quindi le notifiche della blockchain quando la transazione viene impegnata nel registro della blockchain.

FireFly definisce ascoltatori di eventi per consentire all'applicazione client di specificare gli eventi blockchain rilevanti di cui tenere traccia. Un'applicazione client può quindi ricevere le notifiche da FireFly tramite un abbonamento all'evento.

::: Hyperledger FireFly (7/13)

- Sottoscrizione agli eventi: mentre un ascoltatore di eventi dice a FireFly di tenere traccia di determinati eventi dalla blockchain, una sottoscrizione agli eventi dice a FireFly di passare tali eventi al client.



Ogni abbonamento rappresenta un flusso di eventi che può essere consegnato a un client in ascolto con varie modalità di consegna con una garanzia di consegna almeno una volta.

::: Hyperledger FireFly (8/13)

Dato un contratto intelligente in Solidity, deve essere compilato utilizzando solc o qualche altro strumento. Dopo aver compilato il contratto, cerca nel file di output JSON (ABI) i campi per creare la richiesta di distribuzione di seguito.

POST <http://localhost:5000/api/v1/namespaces/default/contracts/deploy>

```
{
  "contract": "608060405234801561001057600080fd5b5061019e806100206000396000f3fe608060405234801",
  "definition": [
    {
      "anonymous": false,
      "inputs": [
        {
          "indexed": true,
          "internalType": "address",
          "name": "from",
          "type": "address"
        },
        {
          "indexed": true,
          "internalType": "address",
          "name": "to",
          "type": "address"
        }
      ],
      "name": "Transfer",
      "type": "event"
    }
  ]
}
```

::: Hyperledger FireFly (8/13)

Dato un contratto intelligente in Solidity, deve essere compilato utilizzando solc o qualche altro strumento. Dopo aver compilato il contratto, cerca nel file di output JSON (ABI) i campi per creare la richiesta di distribuzione di seguito.

POST <http://localhost:5000/api/v1/namespaces/default/contracts/deploy>

```
"output": {  
    "headers": {  
        "requestId": "default:aa155a3c-2591-410e-bc9d-68ae7de34689",  
        "type": "TransactionSuccess"  
    },  
    "contractLocation": {  
        "address": "0xa5ea5d0a6b2eaf194716f0cc73981939dca26da1"  
    },  
    "protocolId": "000000000024/000000",  
    "transactionHash": "0x32d1144091877266d7f0426e48db157e7d1a857c62e6f488319bb09243f0f851"  
},  
    "created": "2023-02-03T15:42:52.750277Z",  
    "updated": "2023-02-03T15:42:52.750277Z"  
}
```

Restituirà una risposta in cui nella sezione di output il nuovo indirizzo del contratto è indicato come 0xa5ea5d0a6b2eaf194716f0cc73981939dca26da1.

::: Hyperledger FireFly (9/13)

Esiste un endpoint HTTP sull'API FireFly che prenderà come input l'ABI di un contratto esistente e genererà automaticamente l'interfaccia FireFly.

POST <http://localhost:5000/api/v1/namespaces/default/contracts/interfaces/generate>

```
{  
  "namespace": "default",  
  "name": "",  
  "description": "",  
  "version": "",  
  "methods": [  
    {  
      "name": "get",  
      "pathname": "",  
      "description": "",  
      "params": [],  
      "returns": [  
        {  
          "name": "",  
          "schema": {  
            "type": "integer",  
            "details": {  
              "type": "uint256",  
              "internalType": "uint256"  
            }  
          }  
        }  
      ]  
    }  
  ]  
}
```

FireFly genera e restituisce l'intera interfaccia FireFly per il contratto di input nel corpo della risposta. L'FFI ottenuto può essere trasmesso all'intera rete utilizzando l'endpoint API /contracts/interfaces.

POST <http://localhost:5000/api/v1/namespaces/default/contracts/interfaces>

::: Hyperledger FireFly (10/13)

FireFly ora deve creare un'API HTTP per questo contratto intelligente, completa di specifica OpenAPI e interfaccia utente Swagger, indicando dove si trova il contratto sulla blockchain.

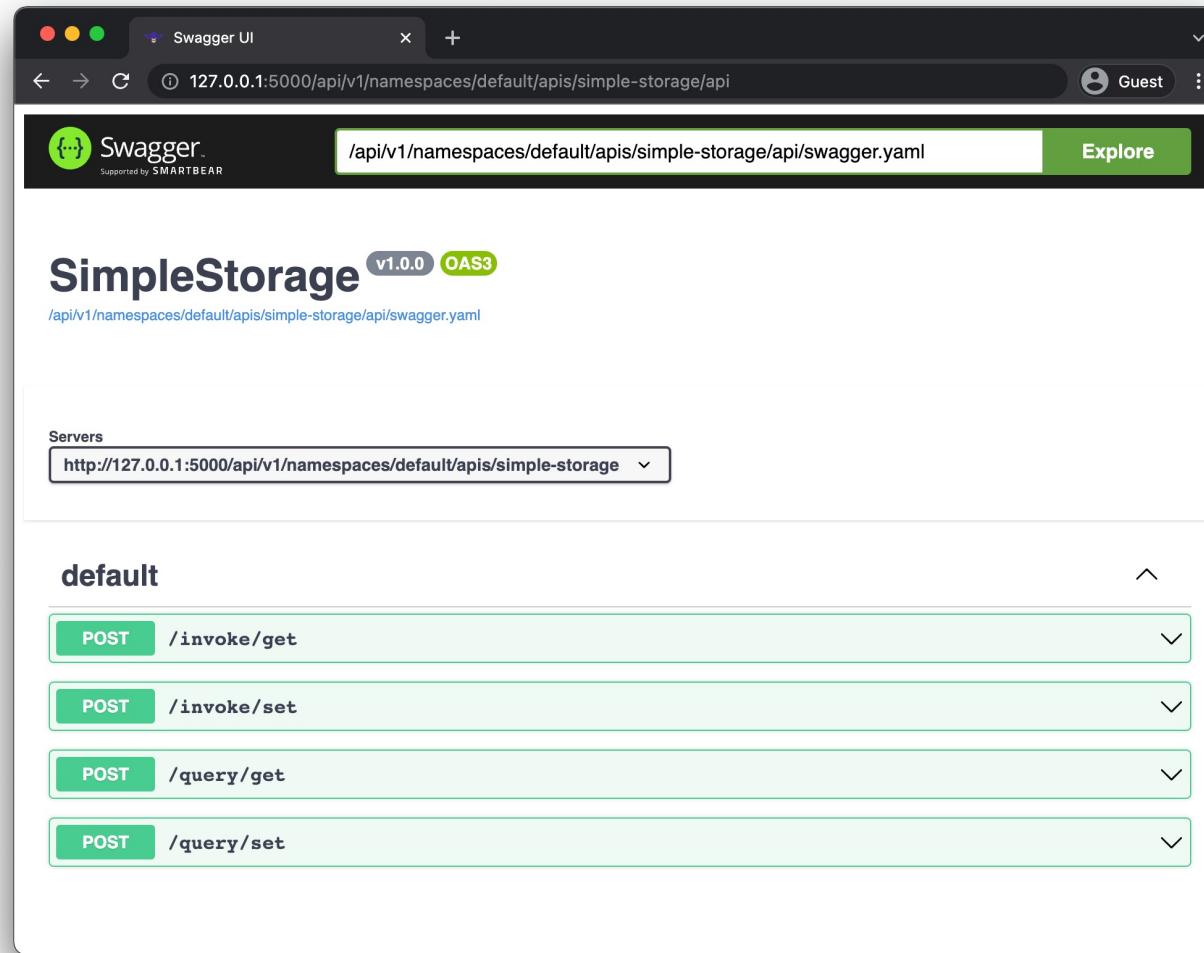
```
POST http://localhost:5000/api/v1/namespaces/default/apis
```

```
{
  "name": "simple-storage",
  "interface": {
    "id": "8bdd27a5-67c1-4960-8d1e-7aa31b9084d3"
  },
  "location": {
    "address": "0xa5ea5d0a6b2eaf194716f0cc73981939dca26da1"
  }
}
```

::: Hyperledger FireFly (11/13)

```
{  
  "id": "9a681ec6-1dee-42a0-b91b-61d23a814b0f",  
  "namespace": "default",  
  "interface": {  
    "id": "8bdd27a5-67c1-4960-8d1e-7aa31b9084d3"  
  },  
  "location": {  
    "address": "0xa5ea5d0a6b2eaf194716f0cc73981939dca26da1"  
  },  
  "name": "simple-storage",  
  "message": "d90d0386-8874-43fb-b7d3-485c22f35f47",  
  "urls": {  
    "openapi": "http://127.0.0.1:5000/api/v1/namespaces/default/apis/simple-storage/api/swagger.json"  
    "ui": "http://127.0.0.1:5000/api/v1/namespaces/default/apis/simple-storage/api"  
  }  
}
```

::: Hyperledger FireFly (12/13)



::: Hyperledger FireFly (13/13)

È ora di utilizzare il contratto intelligente effettuando una richiesta POST all'endpoint di invocazione/impostazione per impostare il valore intero sulla catena. Impostiamolo sul valore 3 adesso.

```
POST http://localhost:5000/api/v1/namespaces/default/apis/simple-storage/invoke/set
```

```
{
  "input": {
    "newValue": 3
  }
}
```

::: Hyperledger FireFly (13/13)

È ora di utilizzare il contratto intelligente effettuando una richiesta POST all'endpoint di invocazione/impostazione per impostare il valore intero sulla catena. Impostiamolo sul valore 3 adesso.

```
POST http://localhost:5000/api/v1/namespaces/default/apis/simple-storage/invoke/set
```

```
{  
  "input": {  
    "newValue": 3  
  }  
}
```

```
{  
  "id": "41c67c63-52cf-47ce-8a59-895fe2ffdc86"  
}
```

::: Hyperledger FireFly (13/13)

È ora di utilizzare il contratto intelligente effettuando una richiesta POST all'endpoint di invocazione/impostazione per impostare il valore intero sulla catena. Impostiamolo sul valore 3 adesso.

POST <http://localhost:5000/api/v1/namespaces/default/apis/simple-storage/invoke/set>

```
{  
  "input": {  
    "newValue": 3  
  }  
}
```

{
 "id": "41c67c63-52cf-47ce-8a59-895fe2ffdc86"
}

POST <http://localhost:5000/api/v1/namespaces/default/apis/simple-storage/query/get>

```
{}
```

::: Hyperledger FireFly (13/13)

È ora di utilizzare il contratto intelligente effettuando una richiesta POST all'endpoint di invocazione/impostazione per impostare il valore intero sulla catena. Impostiamolo sul valore 3 adesso.

POST <http://localhost:5000/api/v1/namespaces/default/apis/simple-storage/invoke/set>

```
{  
  "input": {  
    "newValue": 3  
  }  
}
```

{
 "id": "41c67c63-52cf-47ce-8a59-895fe2ffdc86"
}

POST <http://localhost:5000/api/v1/namespaces/default/apis/simple-storage/query/get>

```
{}
```

{
 "output": "3"
}