

# Introduzione a Hyperledger

Corso Fintech Software Developer  
Modulo **Tecnologie Blockchain**

Docente: Dott. Enrico Zimuel

in collaborazione con:



REGIONE  
PIEMONTE

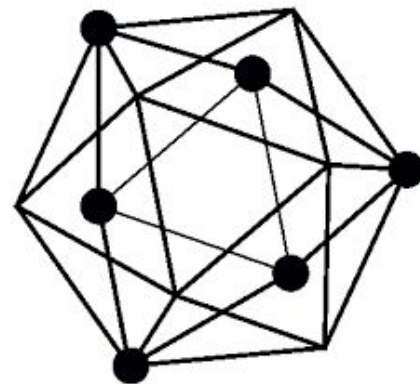
per una crescita intelligente,  
sostenibile ed inclusiva

[www.regione.piemonte.it/europa2020](http://www.regione.piemonte.it/europa2020)

INIZIATIVA CO-FINANZIATA CON FSE

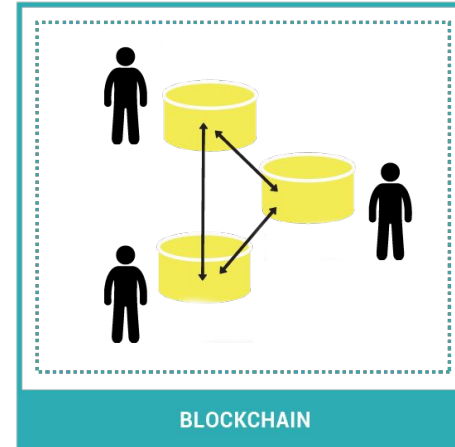
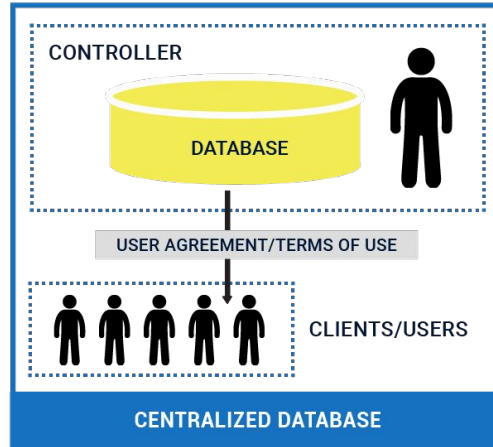
# Sommario

- Database vs. Blockchain
- Tipi di blockchain
- Blockchain in ambito business
- Hyperledger
- I progetti di Hyperledger
- Fabric



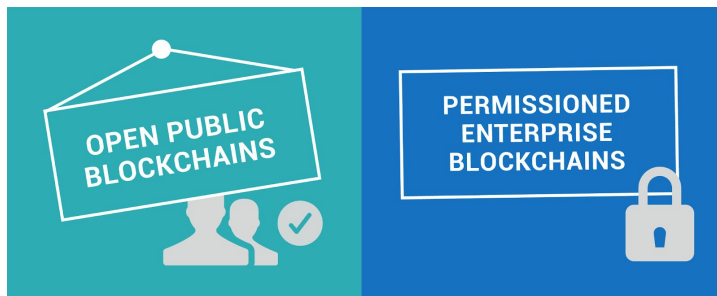
# Database e Blockchain

## CENTRALIZED DATABASES VS. BLOCKCHAIN



# Tipi di blockchain

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



# Blockchain in ambito business

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

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

# Permissioned

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

# Hyperledger

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



**HYPERLEDGER**  
FOUNDATION

# Obiettivi di Hyperledger



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



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



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



Educate the public about the market opportunity for blockchain technology



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











# Progetti Hyperledger

## Graduated Hyperledger Projects (6)

 <p><b>HYPERLEDGER ARIES</b></p> <p>Hyperledger Aries Hyperledger</p> <p>★ 1,421</p>	 <p><b>HYPERLEDGER BESU</b></p> <p>Hyperledger Besu Hyperledger</p> <p>★ 904</p>	 <p><b>HYPERLEDGER FABRIC</b></p> <p>Hyperledger Fabric Hyperledger</p> <p>★ 20,723</p>	 <p><b>HYPERLEDGER INDY</b></p> <p>Hyperledger Indy Hyperledger</p> <p>★ 1,599</p>	 <p><b>HYPERLEDGER IROHA</b></p> <p>Hyperledger Iroha Hyperledger</p> <p>★ 669</p>	 <p><b>HYPERLEDGER SAWTOOTH</b></p> <p>Hyperledger Sawtooth Hyperledger</p> <p>★ 1,750</p>
---	---	--	--	---	---

## Incubating Hyperledger Projects (8)

 <p><b>HYPERLEDGER BEVEL</b></p> <p>Hyperledger Bevel Hyperledger</p> <p>★ 276</p>	 <p><b>HYPERLEDGER CACTUS</b></p> <p>Hyperledger Cactus Hyperledger</p> <p>★ 230</p>	 <p><b>HYPERLEDGER CALIPER</b></p> <p>Hyperledger Caliper Hyperledger</p> <p>★ 596</p>	 <p><b>HYPERLEDGER CELLO</b></p> <p>Hyperledger Cello Hyperledger</p> <p>★ 814</p>	 <p><b>HYPERLEDGER FIREFLY</b></p> <p>Hyperledger Firefly Hyperledger</p> <p>★ 397</p>	 <p><b>HYPERLEDGER GRID</b></p> <p>Hyperledger Grid Hyperledger</p> <p>★ 224</p>
 <p><b>HYPERLEDGER TRANSACTION</b></p> <p>Hyperledger Transact Hyperledger</p> <p>★ 68</p>	 <p><b>HYPERLEDGER URSA</b></p> <p>Hyperledger Ursa Hyperledger</p> <p>★ 314</p>				

# Fabric

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



# Pluggable consensus protocol

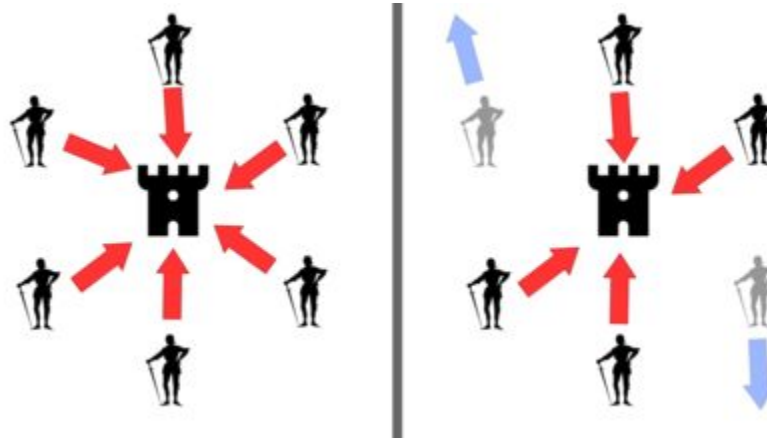
- Una delle caratteristiche più interessanti di Fabric è la capacità di poter configurare l'algoritmo di consenso per la scrittura nella blockchain
- Ad esempio, nel caso di utilizzo di un registro distribuito in un'unica realtà aziendale non ha molto senso utilizzare un algoritmo **Byzantine fault-tolerant**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

# Problema dei generali bizantini

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

# Problema dei generali bizantini

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



# Smart contract (chaincode)

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

# Order-execute

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

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

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

# Execute-order-validate

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

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



# Hyperledger Fabric Model

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

# Esempio: eseguire un chaincode in Node.js

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

try {

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

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

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

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

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

# Esempio: Contract in Node.js

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

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

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

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

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

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

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

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

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

};

module.exports = UpdateValuesContract
```

# Esempio: Chaincode in Node.js

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

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

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

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

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

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

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

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

# Riferimenti

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

# Grazie dell'attenzione!

Per informazioni:

[enrico.zimuel@its-ictpiemonte.it](mailto:enrico.zimuel@its-ictpiemonte.it)