Use Case Avanzati e Implicazioni Pratiche

Settore Finanziario

- **DeFi su Ethereum**: si basa su smart contract aperti che consentono prestiti, exchange di token, assicurazioni e strumenti derivati in un contesto trustless. Chiunque può interagire con questi protocolli semplicemente possedendo un wallet compatibile.
  - **Pro**: massima apertura, innovazione rapida, composability (i protocolli si possono incastrare).
  - **Contro**: volatilità, rischi di hack, complessità normativa, costi di gas a volte elevati.
- **Consorzi bancari**: Molte banche preferiscono reti permissioned per condividere dati di transazioni, regolamenti interbancari, KYC e AML.
  - **Pro**: identità note, compliance legale più semplice, performance elevate.
  - **Contro**: minore trasparenza per il pubblico, dipendenza da accordi tra le parti, possibili problemi di lock-in tecnologico.

Settore Sanitario

Una blockchain **permissionless** non è adatta a memorizzare dati sensibili, se non in forma pseudonimizzata o tramite tecniche crittografiche avanzate (ad es. Zero-Knowledge Proof).

Invece, una blockchain **permissioned** può:

- Gestire i diritti di accesso
- Mantenere un audit trail immutabile e condiviso

Settore Pubblico e Governance

- **Applicazioni permissionless**: es. votazioni pubbliche con smart contract su Ethereum. Il sogno di una “democrazia diretta digitale” spesso si scontra con problemi di privacy, scalabilità e identità.

- **Applicazioni permissioned**: enti governativi che implementano reti private per la gestione sicura di documenti, registri catastali, anagrafe, passaporti. Qui la fiducia nell’autorità è già un presupposto, quindi l’obiettivo principale è l’immutabilità e la condivisione sicura tra uffici diversi.

Approfondimento: Meccanismi di Sicurezza e Teoremi di Consenso

Resistenza a comportamenti bizantini

Teorema CAP → Consistency, Availability, Partition tolerance.

Blockchain Trilemma → Sicurezza, Scalabilità, Decentralizzazione

PBFT richiede che **f < n/3**

- **Permissioned**: l'attaccante deve corrompere più di 1/3 dei nodi validatori.
- **Permissionless** basata su PoW, l'attaccante deve acquisire **>50%** della potenza di calcolo totale (o dell'equivalente stake in PoS).

In entrambi i casi, la sicurezza si regge sul principio che l'attaccante non riesca a dominare la rete con risorse computazionali/finanziarie o corrompere la maggioranza dei nodi noti

Dimostrazione di Sicurezza in PoW

Afferma che:

- L’onestà della catena principale è assicurata quando la maggior parte della potenza di hash è posseduta da nodi onesti.
- Per riorganizzare la catena un avversario deve generare blocchi a una velocità superiore a quella del resto della rete.
- Se l’attaccante ha una frazione **q: q < 0.5** della potenza di calcolo, la probabilità che riesca a prendere il sopravvento diminuisce esponenzialmente con il numero di blocchi di vantaggio che la catena onesta accumula.

Dimostrazione di Sicurezza in PBFT

1. Ogni blocco passa attraverso fasi di richiesta, pre-prepare, prepare, commit.
2. Se i nodi bizantini sono al massimo **f** e **n ≥ 3f+1** allora i nodi onesti possono sempre filtrare i messaggi contraddittori e raggiungere il consenso.

Prestazioni e Metodi di Scalabilità

Permissionless: Layer 2 e Sharding

- **Layer 2**: Ad esempio, in Ethereum, soluzioni come Plasma, Rollup (Optimistic, ZK), e canali di stato (State Channels) spostano parte dell'elaborazione off-chain, mantenendo la sicurezza grazie a periodici controlli su chain.
- **Sharding**: Ethereum 2.0 mira a dividere la blockchain in "shard", ognuna in grado di processare un subset di transazioni, per poi sfruttare un layer di coordinamento comune

  L'obiettivo è aumentare esponenzialmente la capacità di throughput.

Permissioned: Parametrizzazione BFT e Clustering

- La scalabilità è ottenuta facendo sì che non tutti i nodi debbano validare tutto
- Si può configurare un cluster di *orderer* per gestire un gran numero di transazioni.
- L’architettura modulare di Fabric permette di “pluggare” diversi protocolli di consenso o di replicazione a seconda dei requisiti di latenza e throughput.

Privacy e Crittografia Avanzata

Privacy e Crittografia Avanzata

1. **Permissioned con canali privati**
2. **Permissionless con Zero-Knowledge Proof (ZKP)**:
    - possiamo effettuare transazioni cifrate, verificabili da tutti, senza divulgare importi e destinatari in chiaro.
    - Ethereum sta introducendo un crescente supporto per ZKP per realizzare contratti in cui la logica di calcolo è privata, ma la validità è pubblicamente verificabile.

3. **Off-chain**: molte soluzioni archiviano i dati fuori dalla blockchain e lasciano on-chain solo le “prove di integrità”, riducendo l’esposizione di informazioni e problemi di scalabilità.

Approfondimenti su Governance e Aggiornamenti di Protocollo

Hard Fork in Ethereum

- Ethereum ha subito un hard fork dopo l’evento “The DAO” (2016), quando un grave bug in uno smart contract portò al furto di milioni di ETH.

- La community decise una sorta di rollback del ledger, ma non tutti furono d’accordo, generando la nascita di Ethereum Classic (ETC).

Hard Fork in Ethereum (not so smart contract)

```
mapping(address => uint) public balances; // Registra i saldi degli utenti

function withdraw(uint amount) public {
    require(balances[msg.sender] >= amount);
    msg.sender.call{value: amount}("");
    balances[msg.sender] -= amount;
}
```

← Se è un **contratto**, esso può **eseguire codice automaticamente** quando riceve ETH

Hard Fork in Ethereum (not so smart contract)

```
contract Attacker {
    DAO public dao;
    constructor(address _daoAddress) {
        dao = DAO(_daoAddress);
    }
    function attack() public payable {
        require(msg.value >= 1 ether, "Invia almeno 1 ETH");
        dao.deposit{value: msg.value}();  // Deposita ETH nella DAO
        dao.withdraw(1 ether);  // Chiede di prelevare 1 ETH
    }
    fallback() external payable {
        if (address(dao).balance >= 1 ether) {
            dao.withdraw(1 ether);
        }
    }
}
```

Strumenti di Verifica Formale e Best Practice

Negli ultimi anni, per mitigare i rischi legati a bug negli smart contract, si sono sviluppati strumenti di **verifica formale**, che tentano di dimostrare matematicamente la correttezza di un programma.

- **Solidity**: ci sono tool come *Oyente*, *Mythril*, *Slither*, *CertiK*, *Echidna* e altri, che analizzano il bytecode o il codice sorgente alla ricerca di vulnerabilità comuni (overflow, reentrancy, ecc.).
- **Fabric**: la sicurezza del chaincode dipende anche dalla logica di endorsement e dall'architettura del channel. Sebbene la programmazione in Go o Node.js sia più familiare per molti sviluppatori, serve comunque grande attenzione nella gestione delle policy e degli accessi.

Le **best practice**:

1. **Audit esterni** del codice.
2. **Test**
3. **Limitare le funzioni**
4. **Modularità**
5. **Gestione attenta**

Vantaggi e Svantaggi Conclusivi

Caratteristica	Permissionless	Permissioned	Ibrido
Partecipazione	Aperta (chiunque)	Basata su autorizzazioni e identità note	Misto: reti private con ancoraggio pubblico
Consenso	PoW/PoS (alto dispendio o stake)	BFT/CFT (alto TPS, bassa latenza)	Scelta modulare + rely su PoW/PoS per sicurezza
Scalabilità	Limitata (Layer 1), migliorabile con Layer 2	Alta (ma rete più piccola)	Dipende dall'architettura
Governance	Comunitaria, potenziali fork	Consortile, basata su accordi	Duale: parte aperta, parte chiusa
Privacy	Tutto pubblico (salvo ZKP o soluzioni off-chain)	Canali privati, maggiore controllo	Dati sensibili on private, prove su public
Sicurezza	Basata su criptoeconomia, costi d'attacco elevati	Basata su trust parziale e protocolli BFT	Mix, con validazione incrociata
Use Case Tipici	Criptovalute, DeFi, NFT, micropagamenti globali	Supply chain, finanza aziendale	Progetti di tracciabilità con trasparenza pubblica