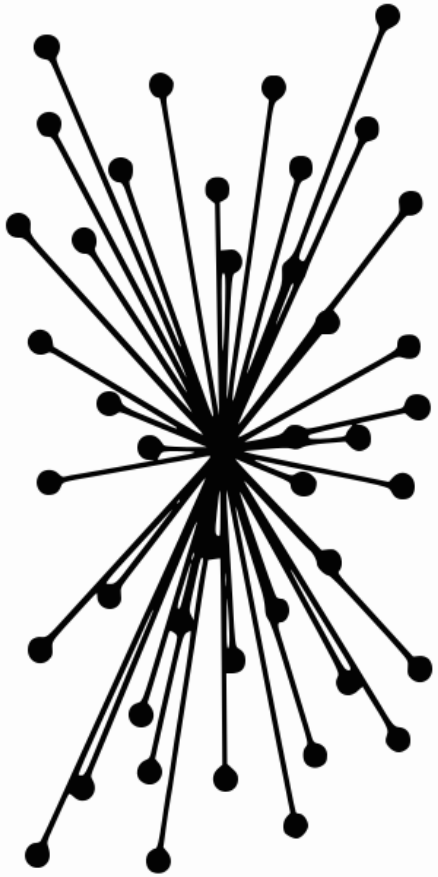
BLOCKCHAIN

Consenso distribuido

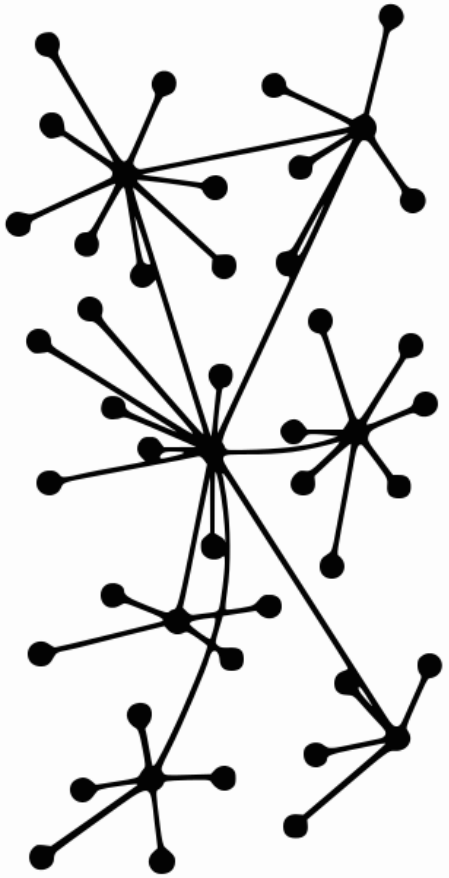
David Cortés
david@oruka.lat
@dacocp

giz Deutsche Gesellschaft für Internationale Zusammenarbeit (GIZ) GmbH

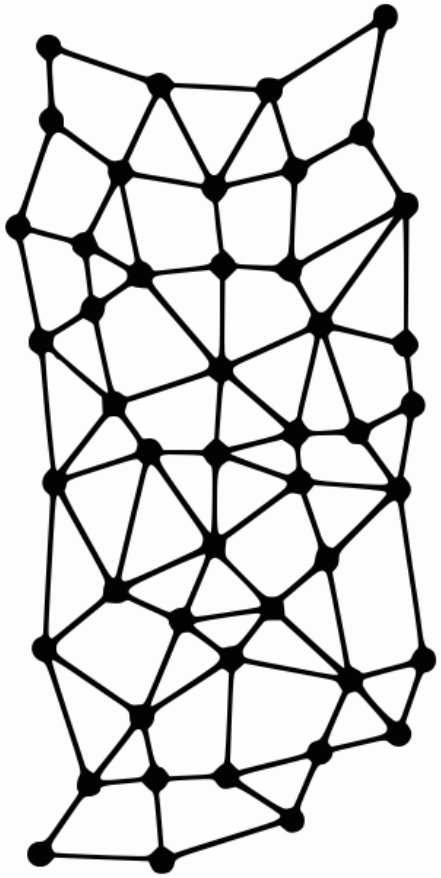
ITAM



Centralized

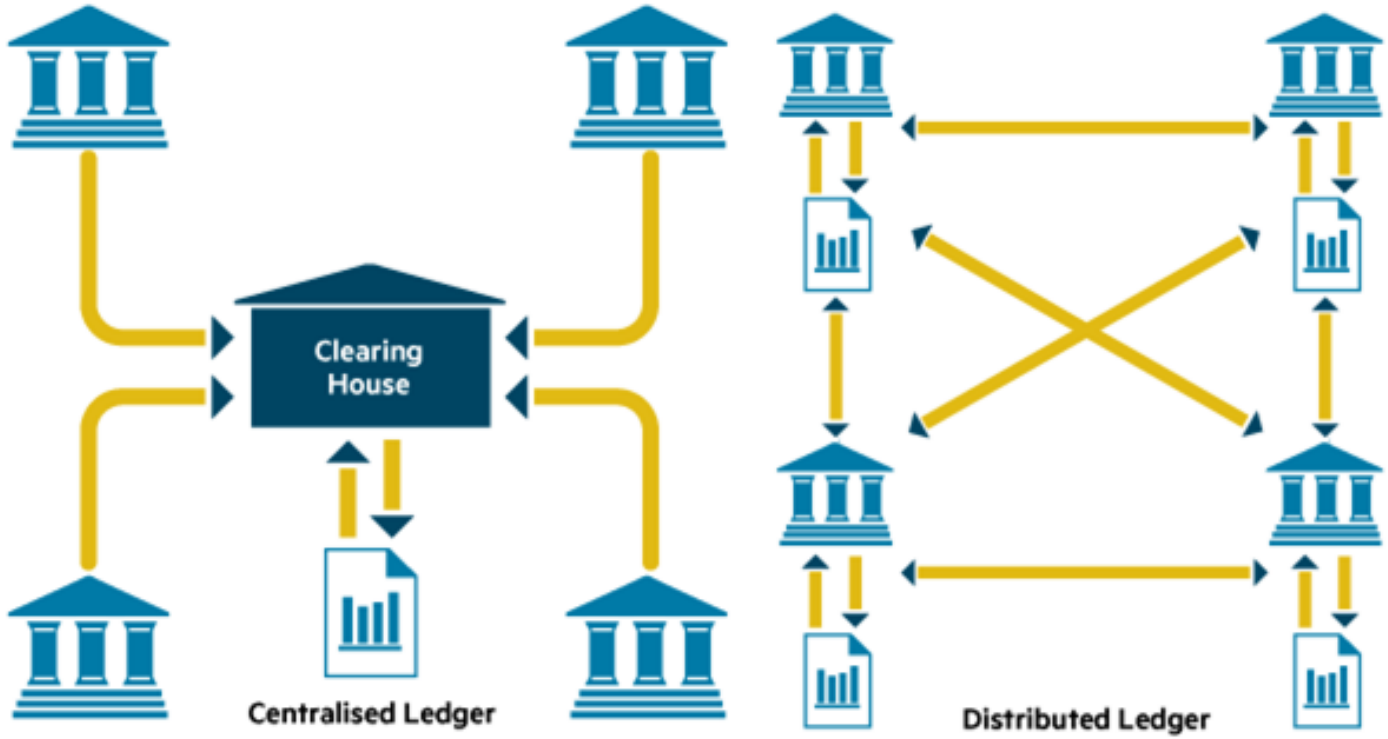


Decentralized



Distributed

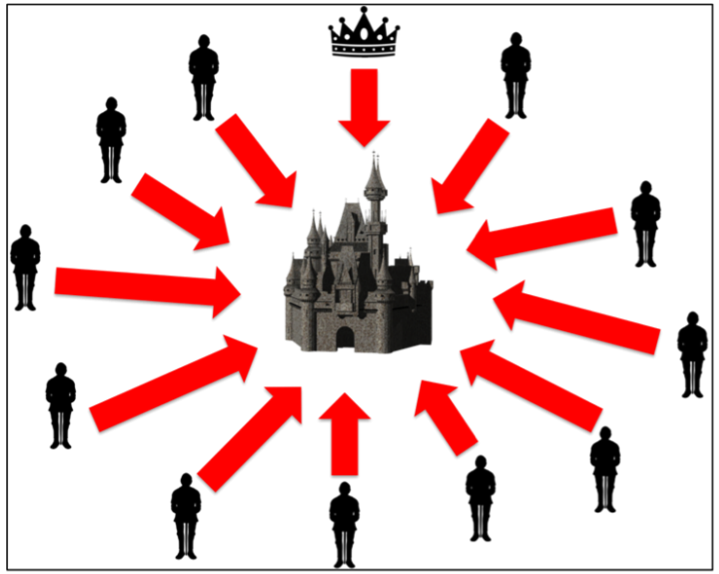

Clearing
House
Centralised Ledger


Clearing House
Centralised Ledger
Distributed Ledger

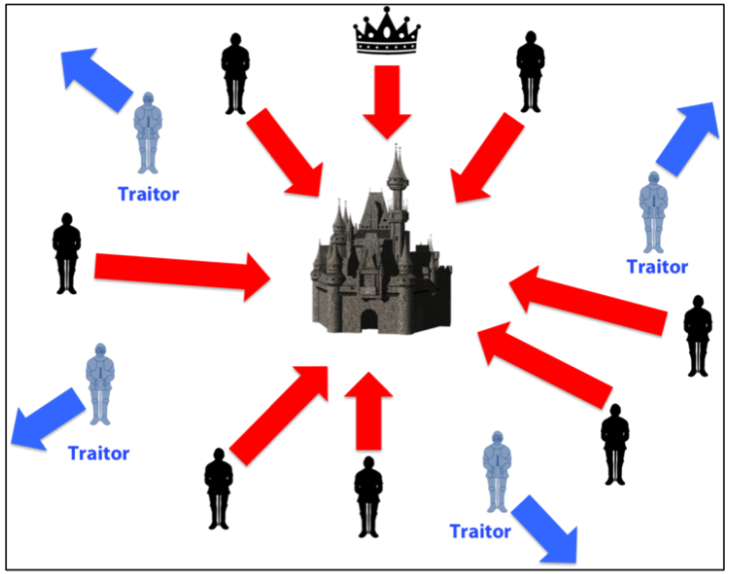
Problema de los generales bizantinos (1982)

- Alcanzar consenso entre los generales leales (i.e. si uno ataca todos atacan / si uno se retira todos se retiran).
- Un pequeño número de traidores no pueden convencer a generales leales de adoptar un plan equivocado.

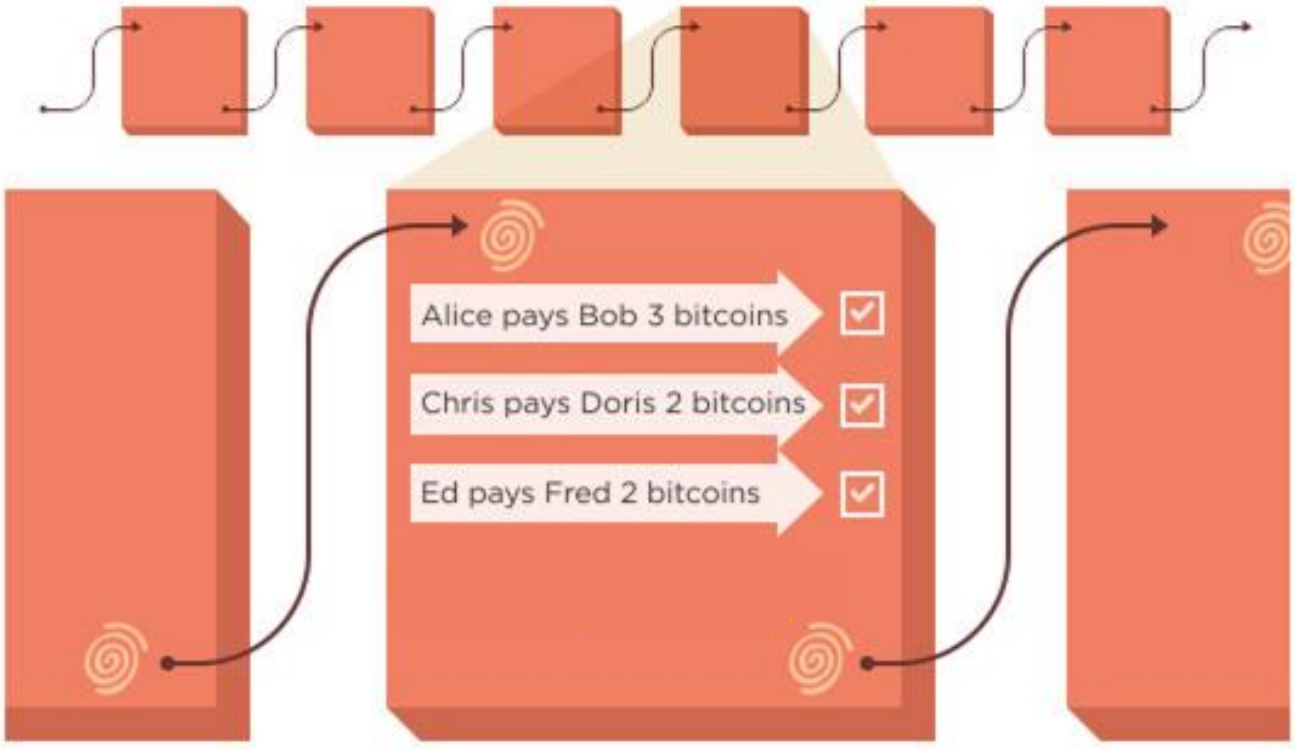
N generales bizantinos asedian una ciudad desde diferentes lugares y quieren llegar a un consenso sobre la decisión de atacar o retirarse.



Ataque coordinado que lleva a la victoria

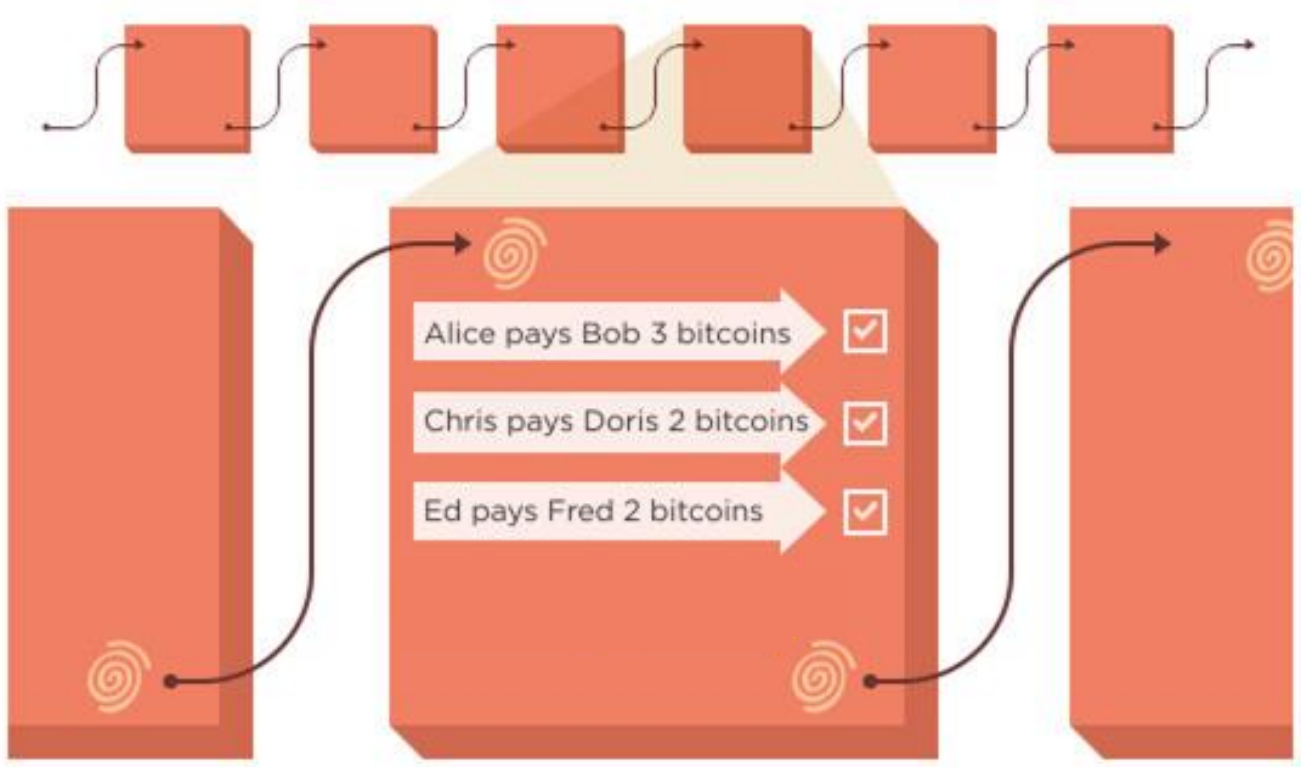


Ataque no coordinado que asegura la derrota



Blockchain

- Se registran transacciones validas (firmadas) en bloques.
- Hay un periodo entre bloques (no es tiempo real)
- Se incluye el hash del bloque anterior
- Criptográficamente auditables



Bitcoin

Bitcoin fue la primera blockchain.

- Es pública
- No es necesario conocer la identidad de los participantes
- No es necesario registrarse ni sacar una cuenta
- Incentiva con *tokens* de bitcoin a los participantes honestos
- Permite a usuarios que no se conocen y no confían el uno en el otro transaccionar sin necesidad de terceros.

Blockchains públicas (Bitcoin, Ethereum, etc.)

- *Trustless*

- No es necesario conocer la identidad de los participantes (nodos / *mineros*)
- Requieren de un *token*
- El mecanismo de consenso recompensa a los participantes honestos que validan transacciones con un *token* (bitcoin, ether, etc.)
- Se penaliza a atacantes a través de un gasto en equipo y consumo eléctrico (*minería / Proof of Work*) o confiscando sus *tokens (Proof of Stake)*.
- Escalables en cuanto a número de participantes (*mineros /* validadores de bloques)

Blockchains privadas

- Se conoce la identidad de los participantes
- No requieren de un *token*
- Existen muchos algoritmos de consenso para blockchains privadas
- Escalables en cuanto a capacidad de procesar transacciones

¿Públicas o Privadas?

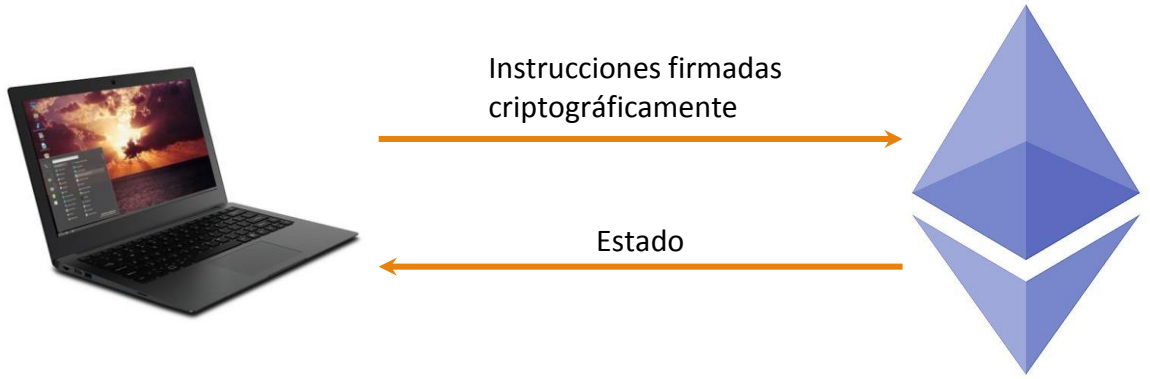
Es cuestion de confianza...

Públicas

Innovación en las orillas
Innovación sin permiso

Ethereum: la computadora del mundo

Tres tipos de instrucciones



- Crear un nuevo contrato (programa)
- Ejecutar una función de un contrato específico
- Transferir ether

Contratos inteligentes

- Programa que puede manejar fondos y moverlos de un lugar a otro si se cumplen ciertas condiciones.
- Son “contratos autoejecutables”, que eliminan la necesidad de notarios, abogados, jueces y otros intermediarios.
- Fungen como marco de **confianza** para que individuos que no confían el uno en el otro, puedan transaccionar.
- Una vez lanzados, la red de Ethereum quedan a la escucha de usuarios que interactúan con los contratos a través sus funciones que ofrecen. Estas definen la “API” del contrato.

Aplicaciones descentralizadas

- Uno (o más) **contratos inteligentes** que se ejecutan en una blockchain y se comunican con una aplicación o sitio web para permitir a los usuarios interactuar.

- Una vez lanzadas no le pertenecen a nadie, convirtiéndose en una plataforma pública para que interactúen los usuarios de manera *peer-to-peer*.

```
pragma solidity ^0.4.18;
// La versión del compilador que se utilizará

contract Votacion {

  /* Mapea los nombres de los candidatos (representados como variables de
  32 bytes) a un entero sin signo.*/
  mapping (bytes32 => uint8) public votosRecibidos;

  /* Solidity no permite utilizar un arreglo de cadenas en el constructor,
  así que se utiliza un arreglo de datos de 32 bytes para guardar la lista
  de candidatos */
  bytes32[] public listaCandidatos;

  /* Este es el constructor que se ejecuta sólo una vez, cuando se lanza el
  contrato inteligente en la blockchain. Requiere como argumento la lista
  de nombres de los candidatos */
  function Votacion(bytes32[] nombresCandidatos) public {
    listaCandidatos = nombresCandidatos;
  }

  // Regresa el conteo total de votos para un candidato hasta el momento
  function totalVotesFor(bytes32 candidato) view public returns (uint8) {
    require(candidatoValido(candidato));
    return votosRecibidos[candidato];
  }

  // Incrementa la cuenta de votos de un candidato específico (emitir voto)
  function votarPorCandidato(bytes32 candidato) public {
    require(candidatoValido(candidato));
    votosRecibidos[candidato] += 1;
  }

  function candidatoValido(bytes32 candidato) view public returns (bool) {
    for(uint i = 0; i < listaCandidatos.length; i++) {
      if (listaCandidatos[i] == candidato) {
        return true;
      }
    }
    return false;
  }
}
```

¡Gracias!

David Cortés
david@oruka.lat
@dacocp