Introduction

Client Server paradigm

Client:	Server:
- Runs on end-hosts	- Runs on dedicated host
- On/off behavior	- Always on
- Issue requests	- Receives requests
- Do not communicate to other clients	- Satisfies clients requests
- Needs to know the server's IP address	- Needs a fixed IP / DNS name
- Uses the service	- Provides the service

Peer To Peer paradigm

Each peer

- Runs on end-hosts
- On/off behavior **(churn)**
- Needs to join the "crowd" and to discover other peers
- Both uses and provides the service
- Communicates directly with other peers

Due to this, we need to define *communication rules* in order to prevent free riding.

Note: servers are needed for bootstrap (but not for resource sharing!)

Def: A peer to peer system is a set of **autonomous entities** (peers) able to **auto-organize** and sharing a set of distributed resources (CPU cycles, memory, bandwidth, ..) in a computer network. The system exploits such resources to give a service in a **partial/complete decentralized way** and is able to adapt to a **continuos churn** of the nodes, maintaining connectivity and reasonable performances without a centralized entity.

Consequences: exploit "in excess" computational resources, scalability, decrease the cost to setup up new apps, fault tolerance

Resource sharing:

The peers' connection is transient (frequent [dis/]connections). The resources offered by the peers are dynamically added and removed. Each peer is paired with a different IP address for each connection (a resource cannot be located by a static IP -> *define at IP level new addressing mechanisms*

File sharing:

Persistence and security are **not** the main goal, while anonymity is important. The peer stores the shared file in a directory and pairs each file with a set of keys able to identify it *(song? Title, author, date, ..)*. Another user that is interested in finding that file sends a query to the system, finds some peers and chooses some of them (with some criteria) and the file is copied through a download.

Blockchain

Def: a replicated, consistent, immutable, append-only database stored in multiple copies on computer throughout the world, maintained without the need for a central authority.

Cannot be shutdown, censored or delete its content *(through digital signatures [authN], cryptographic hash functions [immutability], replication [availability], distributed consesus amongst mutually trusting or distrusting replica [integrity, decentralized control]*

- ➔ **Bitcoin:** A distributed ledger of transactions.

  A wants to send money to B == the transaction is inserted in a block with other transactions, the block is broadcasted to every part of the network, those in the network approve that the transaction is valid so that the block is added to the chain which provides indelible and transparent record of transactions.

  There are different actors involved, miners, exchangers, …

- ➔ **Ethereum:** A grand vision of “generalized” cryptocurrency, uses **smart contracts**. Programmable through Turing complete languages *(Solidity, Serpent, ..)* and executed by all nodes: consensus as agreement on the results of computation

- ➔ **Etc.**.

Permissionless blockchains *(Bitcoin, ..)*

- Everyone allowed to use
- Everyone untrusted (can be malicious)
- No central authority

Permissioned blockchains *(Hyperledger, Corda, ..)*

- Selected and authN users can participate
- Supports immutability, info sharing, privacy, ..
- 1° pt. through consortium or central authority

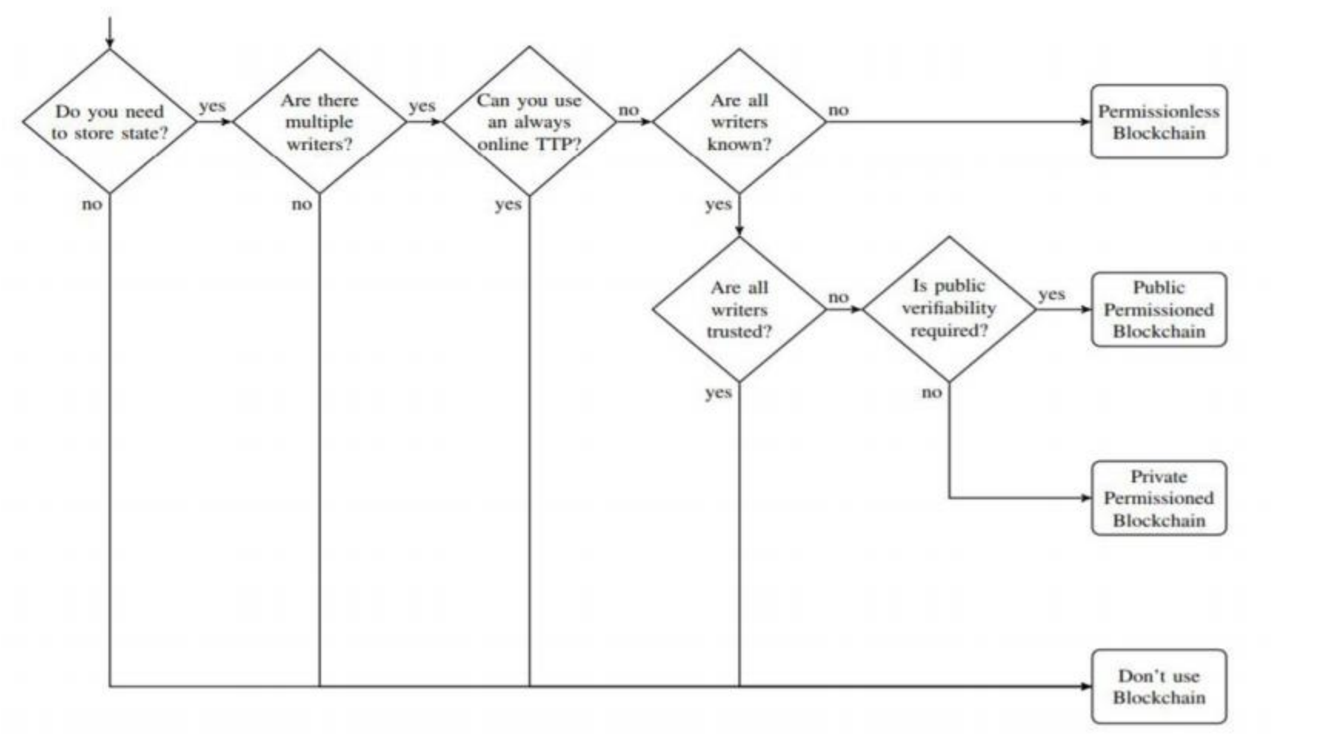
<u>See paper on “Do you need a blockchain”</u>

Use it in apps:

- *that require shared common append-only DB with limited capacity*
- *with multiple participants with varying degrees of trust amongst them*
- *must run in a distributed manner*
- *that require a complex settlement process with a trusted third party*

- *needing integrity, authN, non-repudiation*
- *governed by precise rules that do not change and are simple to encode*
- *requiring transparency*

WHEN DO YOU NEED A BLOCKCHAIN?


Do you need to store state?
yes
no
Are there multiple writers?
yes
no
Can you use an always online TTP?
no
yes
Are all writers known?
no
yes
Permissionless Blockchain
Are all writers trusted?
no
yes
Is public verifiability required?
yes
no
Public Permissioned Blockchain
Private Permissioned Blockchain
Don't use Blockchain

Currency asset transfer: in general, sending money between 2 banks can involve a 3rd one *(many hops)*. This is solved via **distributed ledger** in which only banks are writers *(permissioned BC),* where transactions happen only between the bank and the ledger.