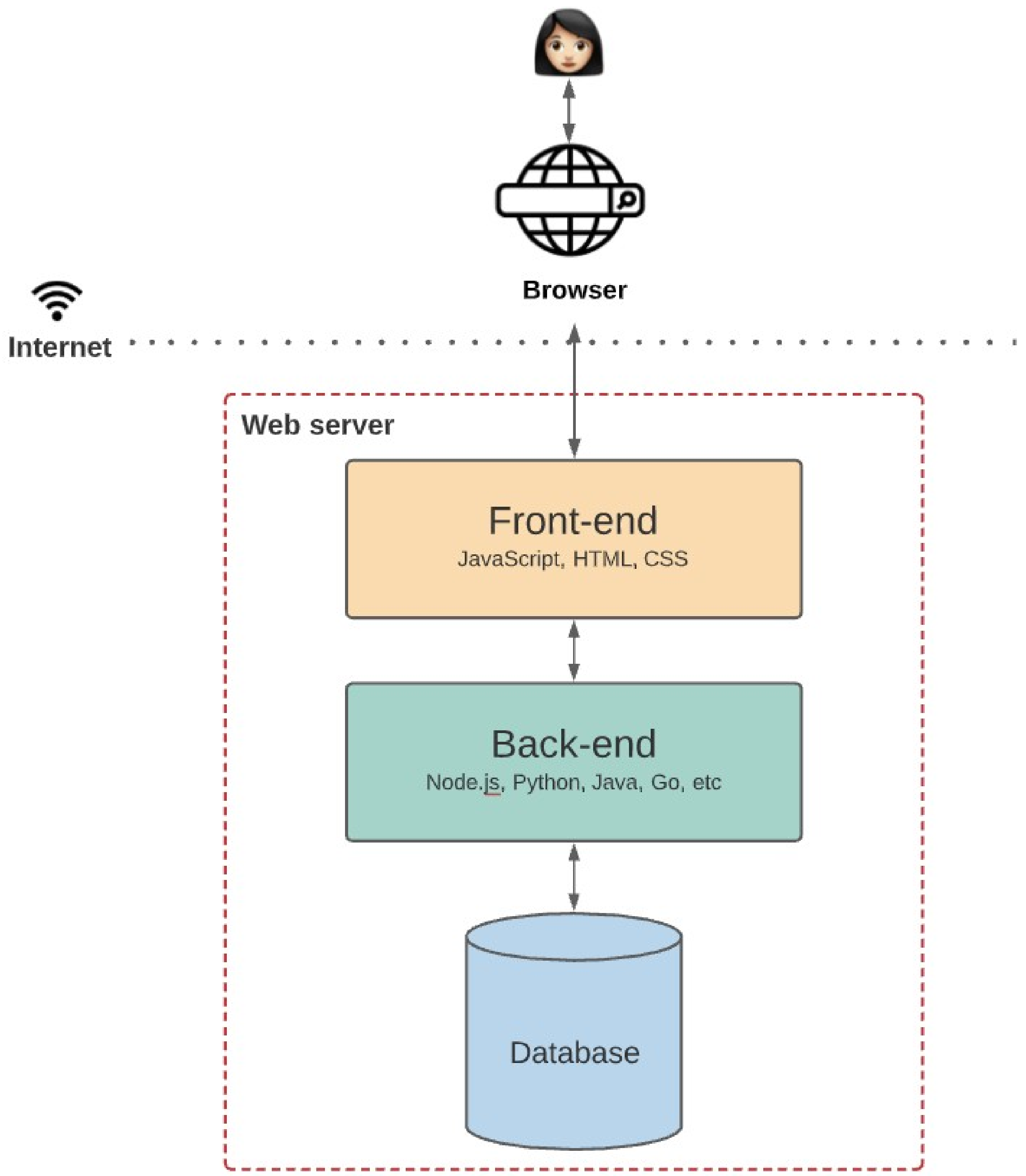
Decentralized Systems

Web3

The Web3

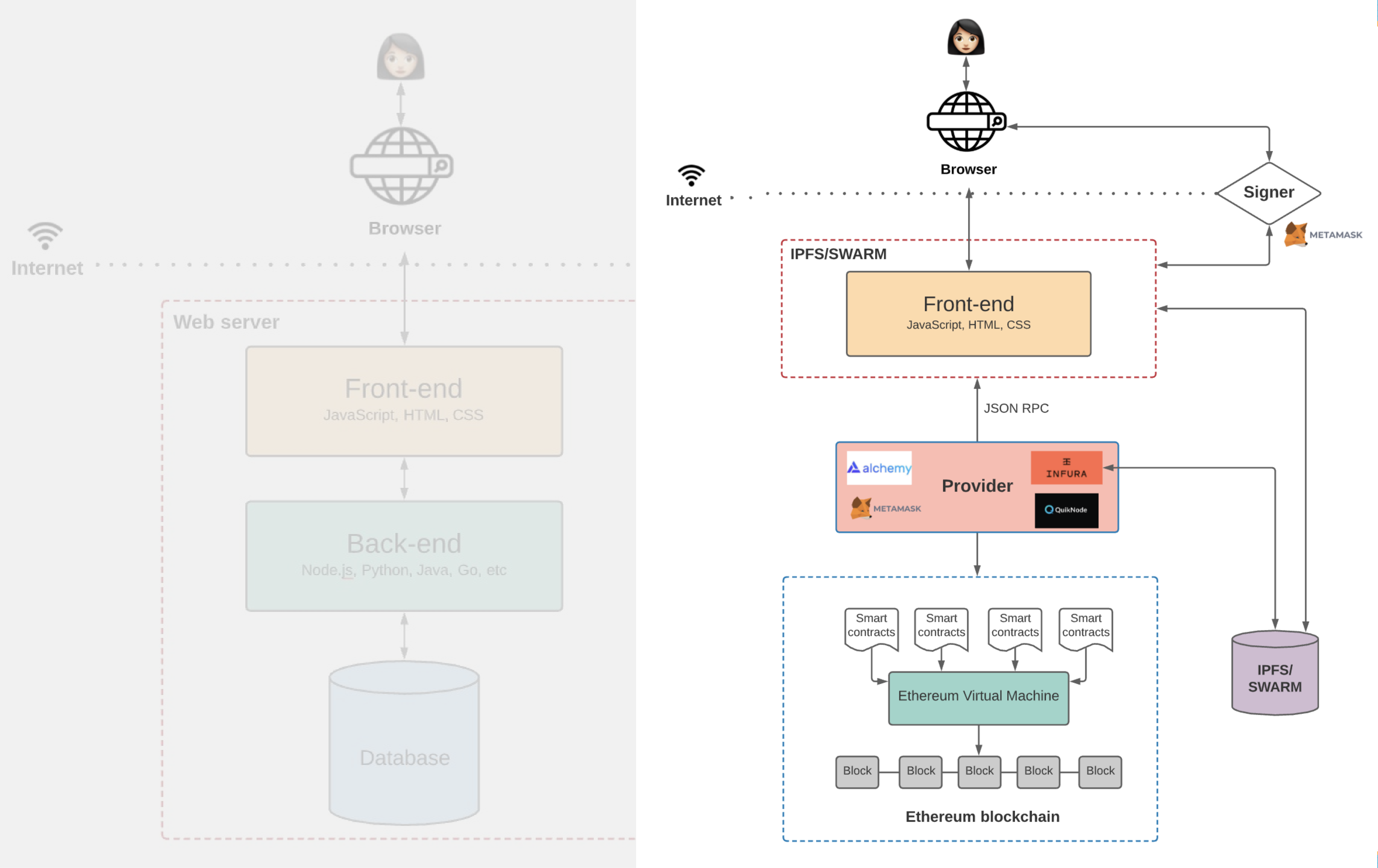
- The Internet we have today is “broken”
  - We do not control our data
  - Every time we interact, online copies of our data are sent to the servers of some (few) “trusted” tech companies
- While the **Web2** was a **front-end revolution**, the **Web3** is a **back-end revolution** proposing a **decentralized state layer** built **on top of blockchain** technologies



Suggested reading:
https://www.preethikasireddy.com/post/the-architecture-of-a-web-3-0-application

Web server for the back-end logic

“Centralized” database storing the application state


Browser
Internet
Web server
Front-end
JavaScript, HTML, CSS
Back-end
Node.js, Python, Java, Go, etc
Database
Browser
Internet
Signer
METAMASK
IPFS/SWARM
Front-end
JavaScript, HTML, CSS
JSON RPC
alchemy
INFURA
Provider
METAMASK
QuikNode
Smart contracts
Smart contracts
Smart contracts
Smart contracts
Ethereum Virtual Machine
Block
Block
Block
Block
Block
Ethereum blockchain
IPFS/ SWARM

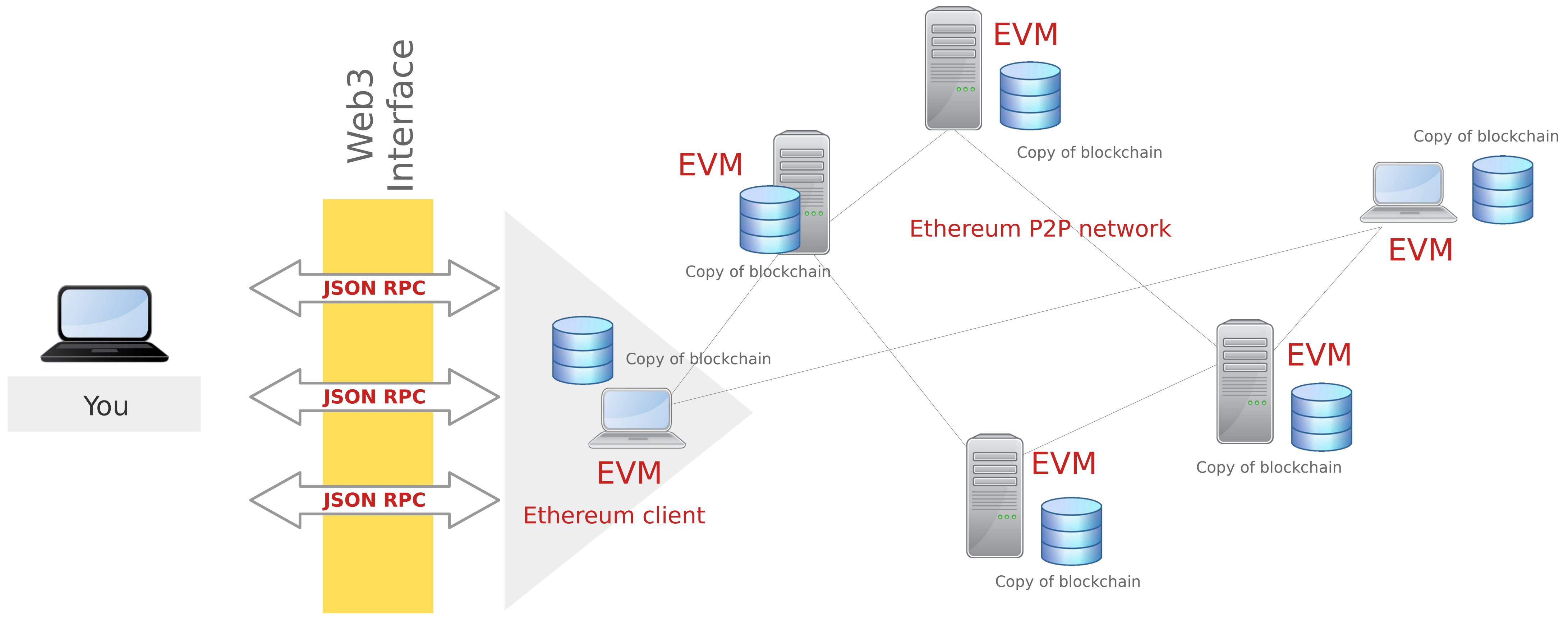
The Web3

- **Decentralization** is at the core of Web3, which is
  - Verifiable
  - Self-governing
  - Permissionless
  - Provides native built-in payments
- Producers and consumers of **Web3 services** (computing, storage, bandwidth, identity, hosting) **pay**, similarly to what happens today for cloud services. But in Web3 **rewards go directly to the network participants** who keep the overall ecosystem up, running, and secure

Ethereum stack

- LEVEL 1: EVM ✓
- LEVEL 2: Smart Contracts ✓
- LEVEL 3: Ethereum nodes
  Computers running software (e.g., an Ethereum client implementing the JSON-RPC standard) they **collectively store the state of the Ethereum blockchain** and **reach consensus** on transactions to mutate the blockchain state
- LEVEL 4: Ethereum Client APIs ← **Today**
  See: https://ethereum.org/en/developers/docs/apis/javascript/
- LEVEL 5: End-User Applications ✓
  Standard web and mobile apps

Ethereum ecosystem



web3.js

- **web3.js** the the first **JavaScript library** developed to **interact** with the **Ethereum** blockchain
- It supports different APIs
  - **web3-eth** for the Ethereum blockchain and smart contracts
  - **web3-shh** for the whisper protocol, p2p communication and broadcast
  - **web3-bzz** for the swarm protocol, the decentralized file storage
  - **web3-utils** contains useful helper functions for dApp developers
  - See https://docs.web3js.org/

ethers.js

- JavaScript library designed to interact with the Ethereum blockchain. It has many classes:
  - **1. Provider**: class for a **connection** to the Ethereum blockchain
  - **Read-only access** to the blockchain
    - provider.**getBalance**(address)
    - provider.**getBlockNumber**()
    - provider.**getTransaction**(txhash)

ethers.js

- JavaScript library designed to interact with the Ethereum blockchain. It has many classes:
  - **2. Contract**: class to **interact with a specific deployed contract**, accessible like a JavaScript object
    - ethers.**Contract**(address, abi, provider)
  - Given the contract instance, it is possible to call its **methods** as if they were local **JavaScript functions**

ethers.js

- JavaScript library designed to interact with the Ethereum blockchain. It has many classes:
  - **3. Signer**: can **sign messages and transactions** to perform **write operations** that have a cost
    - Usually connected to a Provider
    - ethers.**Wallet**, for non custodial accounts
    - ethers.**JsonRpcSigner**, for custodial accounts with private keys managed by other services

ethers.js

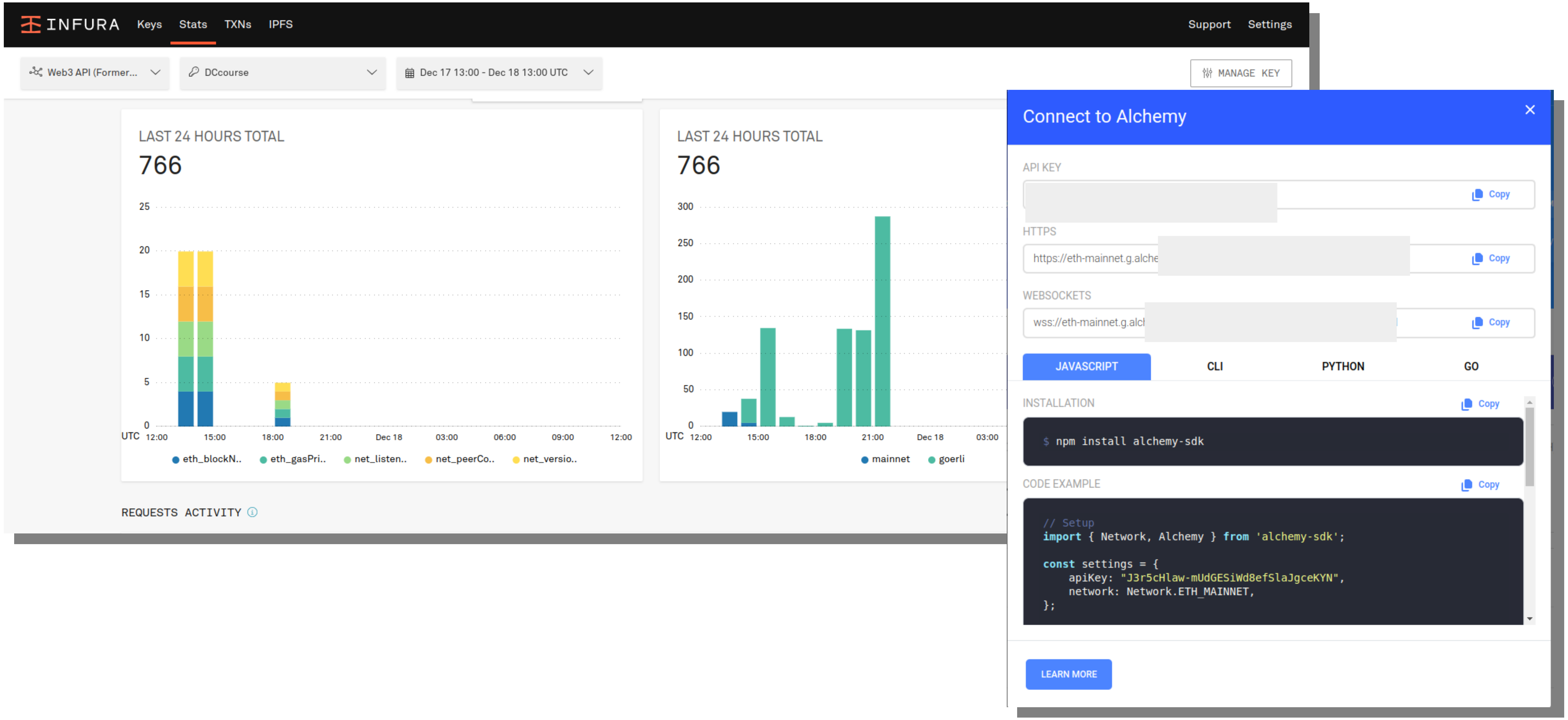
- JavaScript library that also includes numerous **utility functions** for common operations:
  - ethers.utils.formatEther / ethers.utils.parseEther
  - ethers.utils.keccak256 / ethers.utils.sha256
- **BigNumber** class (to handle **large numbers**, especially for Ethereum balances in wei)
  - ethers.BigNumber.from (creates a BigNumber from a number, string, or hex value)

How to

- To use web3.js or ether.js you need
  - **node.js**
  - the **JS library of your choice** (npm install ...)
  - an **Ethereum node** which provides access to the **Ethereum JSON-RPC API method library** that interacts with the Ethereum blockchain
    see
    https://www.alchemy.com/overviews/blockchain-node-providers

Connect to a node

- You need to create an account and get an **API Key**



API KEY

[redacted]

HTTPS

https://eth-mainnet.g.alche[redacted]

WEBSOCKETS

wss://eth-mainnet.g.alc[redacted]

INSTALLATION

```
$ npm install alchemy-sdk
```

CODE EXAMPLE

```
// Setup
import { Network, Alchemy } from 'alchemy-sdk';

const settings = {
    apiKey: "J3r5cHlaw-mUdGESiWd8efSlaJgceKYN",
    network: Network.ETH_MAINNET,
};
```

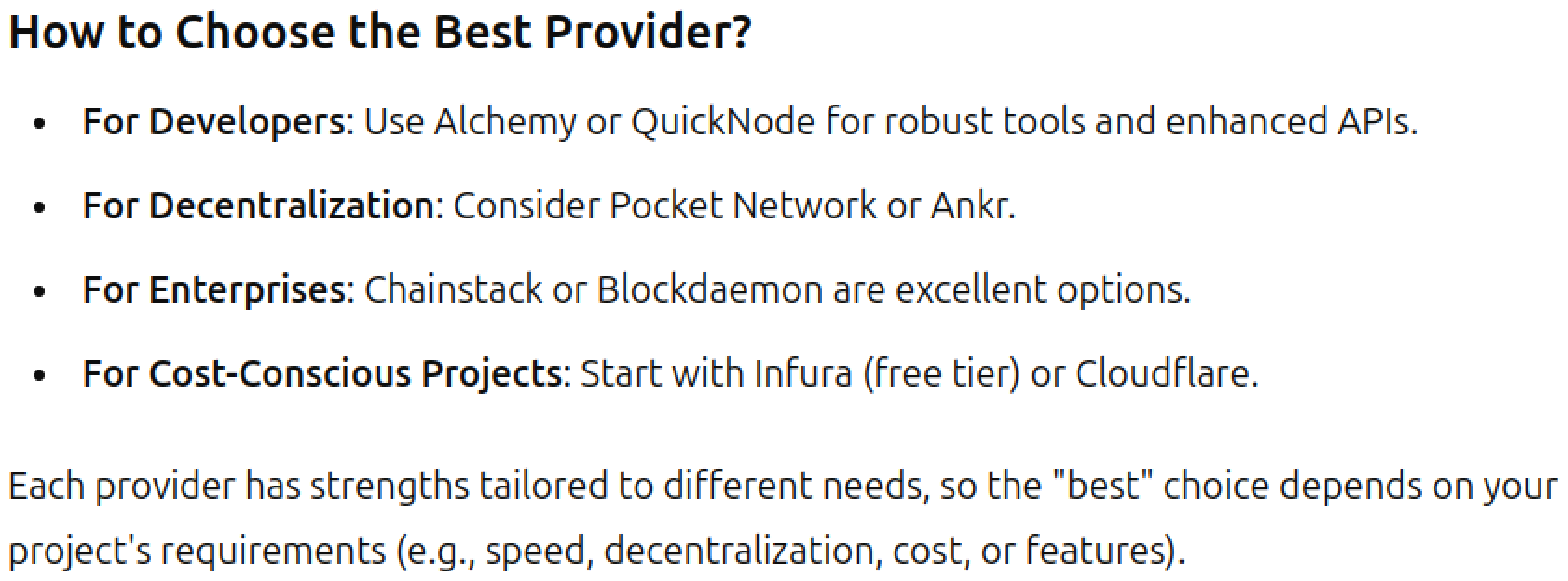
Connect to a node

- With the **API Key** you can connect to Ethereum
- I will use **INFURA API Key** (but others seems more popular today)
- Hint: for private and API keys consider the **dotenv module** to store your private information outside the code and load them from a **.env** file

  See: https://www.npmjs.com/package/dotenv

Connect to a node

- According to ChatGPT ;)



How to Choose the Best Provider?

- **For Developers**: Use Alchemy or QuickNode for robust tools and enhanced APIs.
- **For Decentralization**: Consider Pocket Network or Ankr.
- **For Enterprises**: Chainstack or Blockdaemon are excellent options.
- **For Cost-Conscious Projects**: Start with Infura (free tier) or Cloudflare.

Each provider has strengths tailored to different needs, so the "best" choice depends on your project's requirements (e.g., speed, decentralization, cost, or features).

And interact :)

- There are **two ways** to interact with accounts and smart contracts: **reading** state or **writing** state
  - **Reading** from the blockchain **does not cost**: the function call is carried out by the connected node and it is free
  - **Writing** into the blockchain **has a cost**, paid in gas, which determines **the fee** required to successfully conduct a transaction or execute a contract to **update the state** of Ethereum

Start
DEMO