Decentralized Systems

Web3 (cnt)

Smart contracts

- Are smart contracts really immutable

Smart contracts

- Are smart contracts really immutable

Yes and No

The **logic of smart contracts is immutable** after deployment, but they **change the state** (balances, storage variables)

It is also possible to **migrate** a smart contract to a different address to change its logic, but we need to convince users to move and be sure we do not lose any state data

Smart contracts

- Are smart contracts really immutable

Yes and No

Upgradable Smart Contracts can be modified after they have been deployed to the blockchain

Upgradable smart contracts

- **Advantages**
  - **Fixing bugs**: if a bug/vulnerability is discovered in a deployed smart contract, it can be fixed by upgrading the contract; of course, this **improves the security** of the smart contract
  - **Adding new features**: new features can be added to a smart contract without having to redeploy the contract from scratch

Upgradable smart contracts

- **Proxy pattern**
  - Uses a **proxy contract** to **delegate calls** to the **implementation contract**. The proxy contract can be upgraded to point to a new implementation contract
  - **Separates the logic** (implementation) **from its address** and **storage** and allows for upgrades without changing the contract's address or losing state data

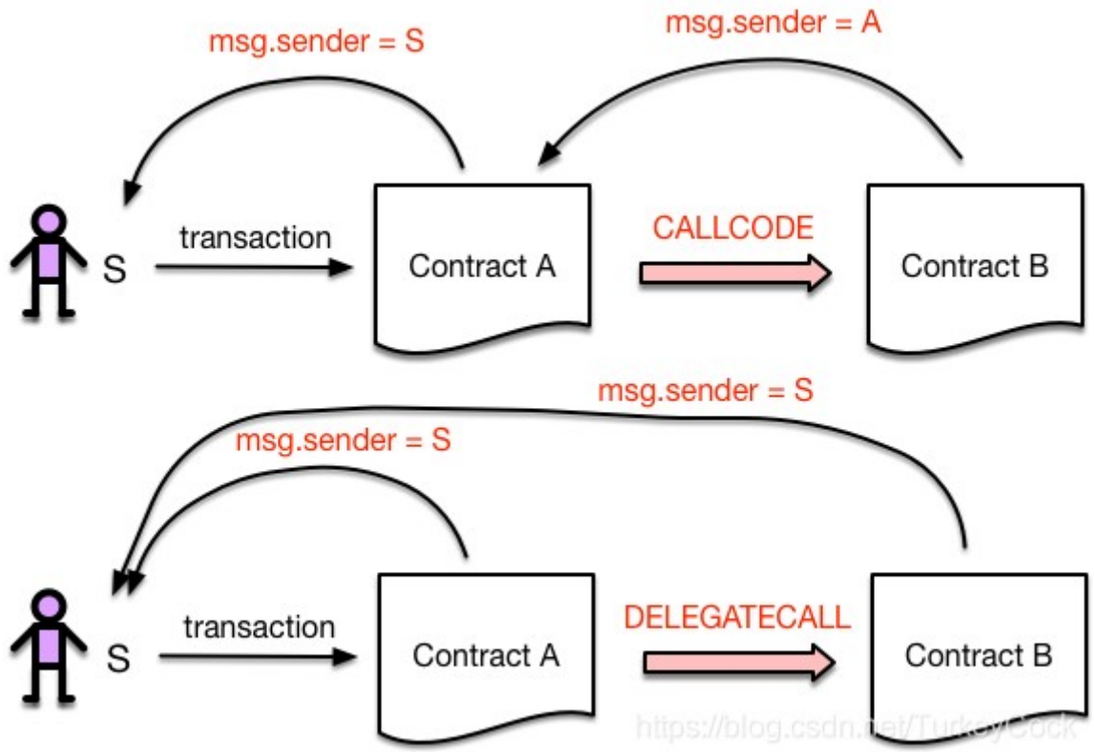
Upgradable smart contracts

- **Implementation** contract
  - Contains all the logic of the contract (v1)
  - To upgrade we deploy a new implementation of the contract (v2)
- **Proxy** contract
  - Points to the “current” implementation (v1)
  - Routes all function calls to that implementation
- **Users** make calls through the proxy
- **Admin** is the only actor who can upgrade to a new implementation contract

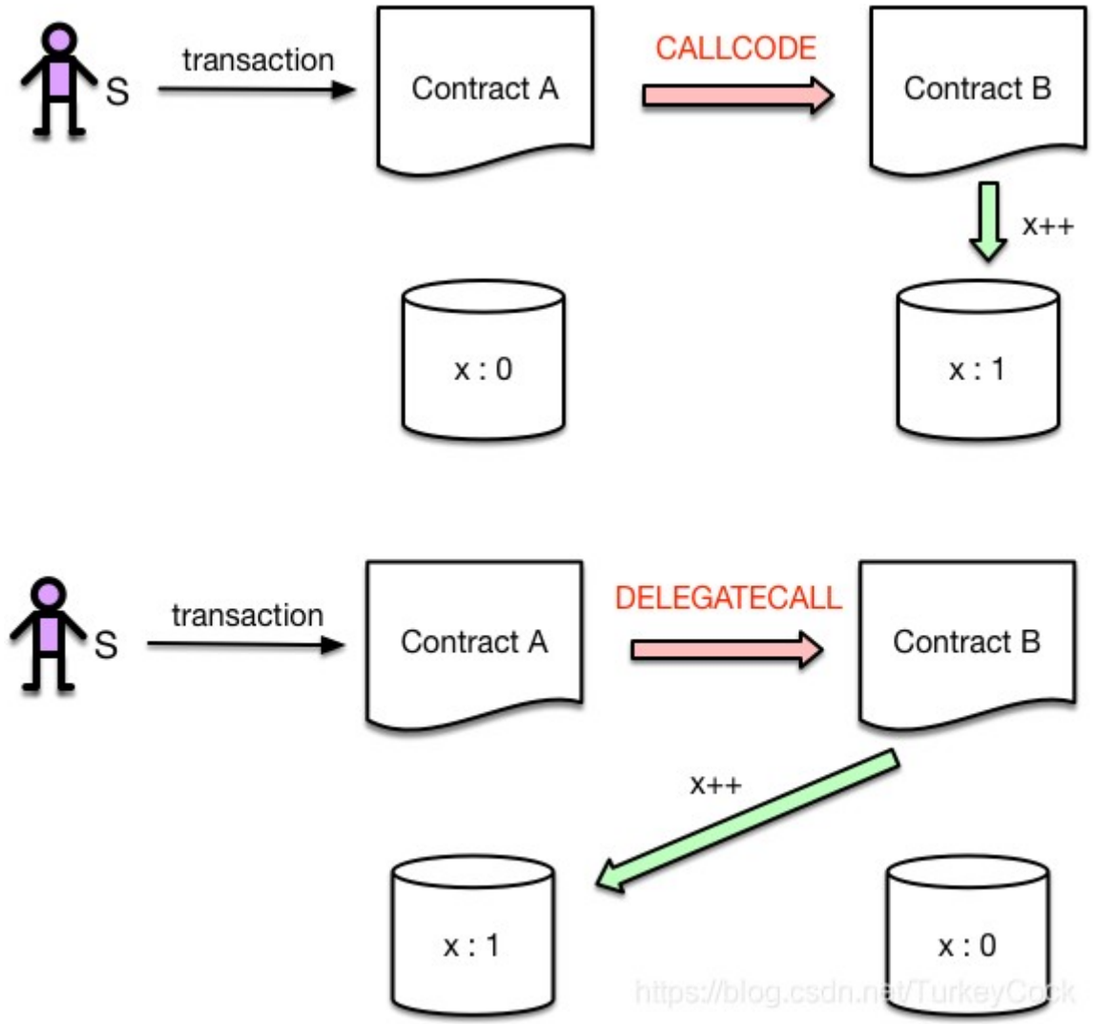
Delegate call

`delegatecall` is a low level function similar to `call`.

When contract `A` executes `delegatecall` to contract `B`, `B`'s code is executed with contract `A`'s storage, `msg.sender` and `msg.value`.



Delegate call



https://solidity-by-example.org/delegatecall/

Proxy contract structure

- **State variables**: those of the implementation plus at least an **address** to store the address of the currently active implementation contract
- **Constructor**: takes the address of the initial implementation contract as an argument and sets the address accordingly
- **Upgrade function**

```
modifier onlyAdmin() {
        require(msg.sender == admin, "Only admin can call this function");
        _;
    }
function upgradeContract(address _newAddress) external onlyAdmin {
      implementation = _newAddress;
}
```

Proxy contract structure

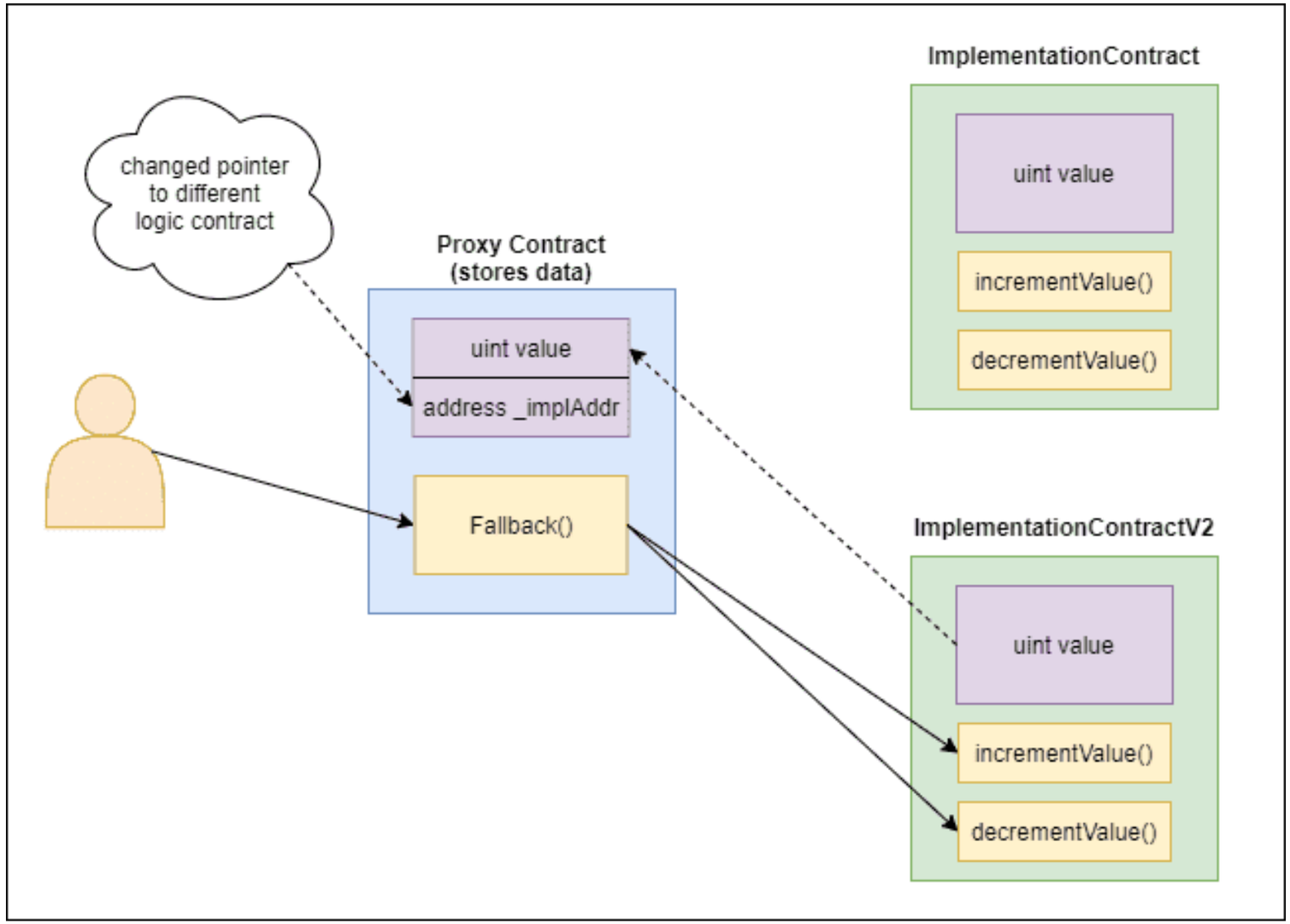
- **Fallback** functions handles calls to functions that do not exist in the contract, can receive Ethers (if defined as payable)
- In a proxy contract act as the **default entry point** for all external calls
- Intercept incoming transactions, extract the function selector and calldata, and delegate the call to the implementation contract using the delegatecall opcode

Proxy contract structure

- The **delegatecall** function is available on the **address type**
  - One parameter, e.g., a bytes array containing the encoded function call data
  - This data is typically generated using the abi.encodeWithSelector or abi.encodeWithSignature functions, used to encode function calls into the low-level bytes format required for interacting with other contracts

- The **data** include
  - The function selector (the first 4 bytes of the Keccak256 hash of the function signature)
  - The arguments encoded in ABI format

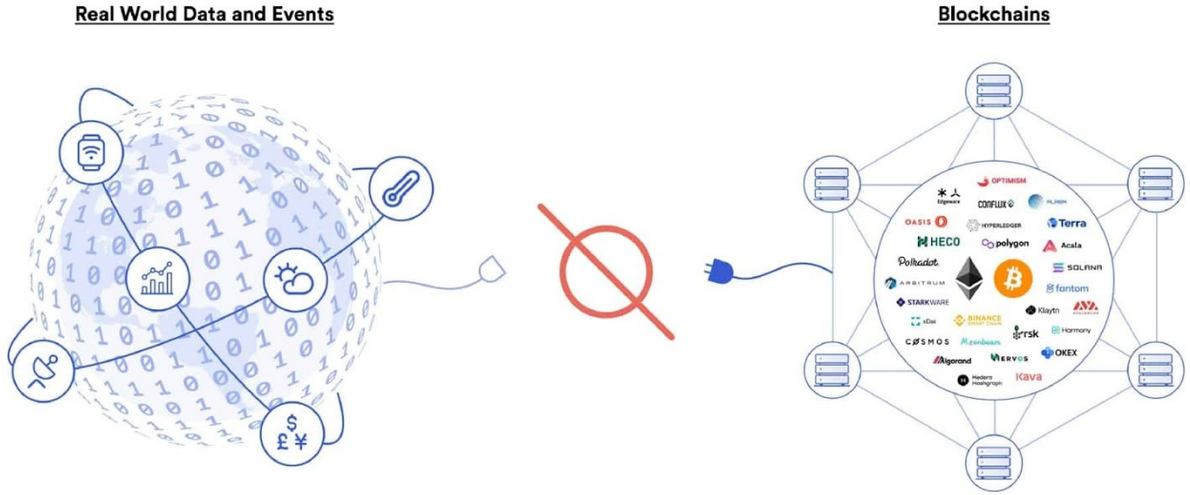
Proxy contract structure



Reading data from the outside world: Oracles

Thanks to Ali Haider and Giacomo Pedemonte

The Oracle problem

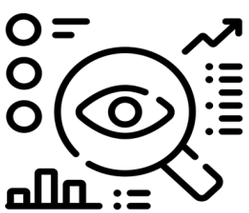


- By being purposely isolated, blockchains obtain
  - **strong consensus** on the validity of user transactions
  - **prevention of double-spending** attacks
  - **mitigation** of network **downtime**

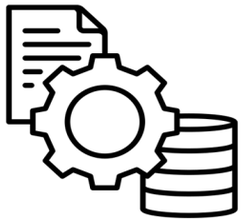
Blockchain Oracle

- Securely interoperating with **off-chain systems** from a blockchain requires an additional piece of infrastructure known as an “oracle” to bridge the two environments
- The vast majority of smart contract use cases like DeFi (Decentralized Finance) require **knowledge of real-world data and events** happening off-chain

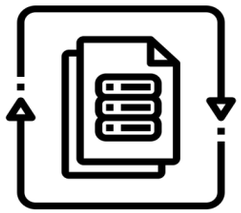
Blockchain Oracle



LISTEN

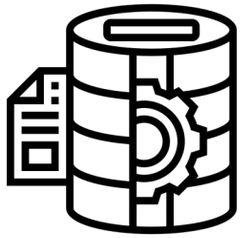


EXTRACT

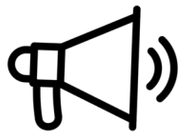


FORMAT

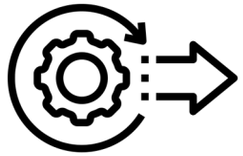
VALIDATE



COMPUTE

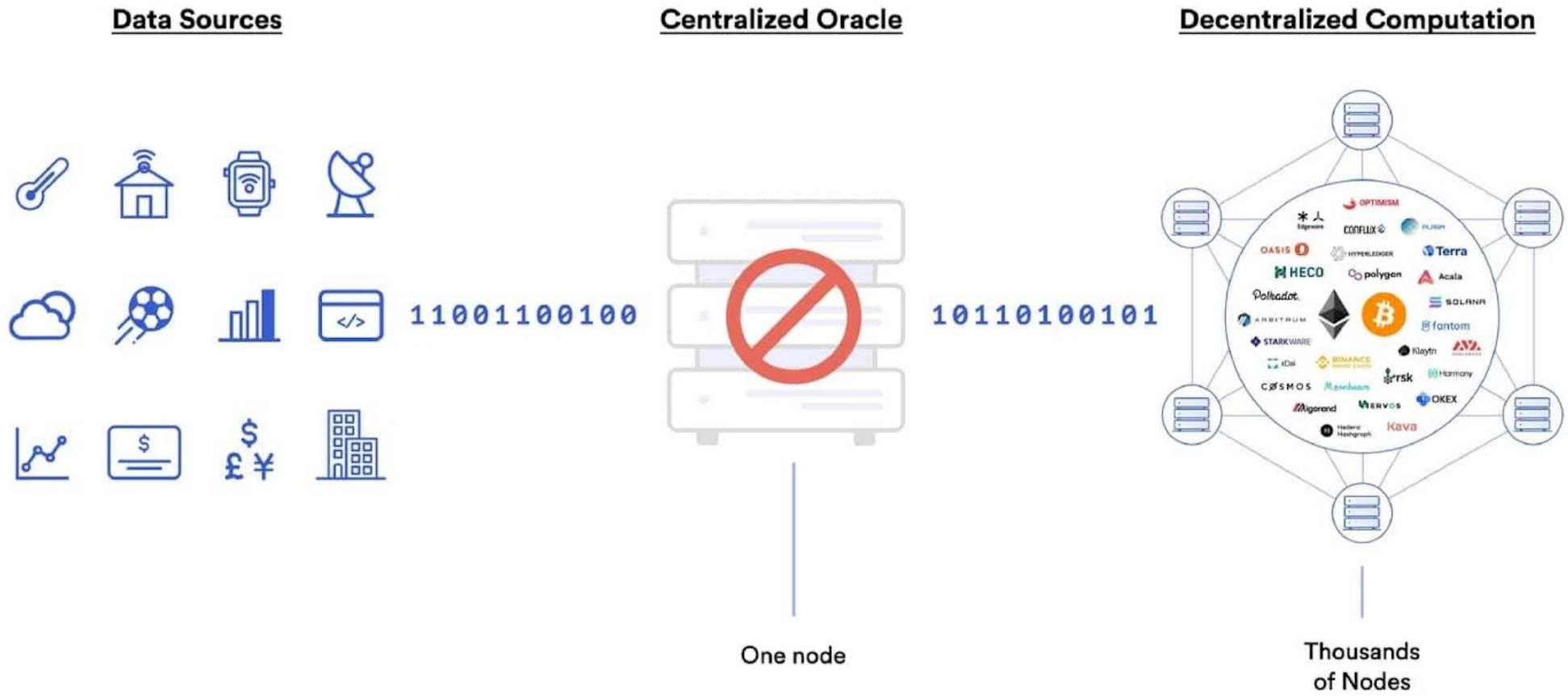


BROADCAST



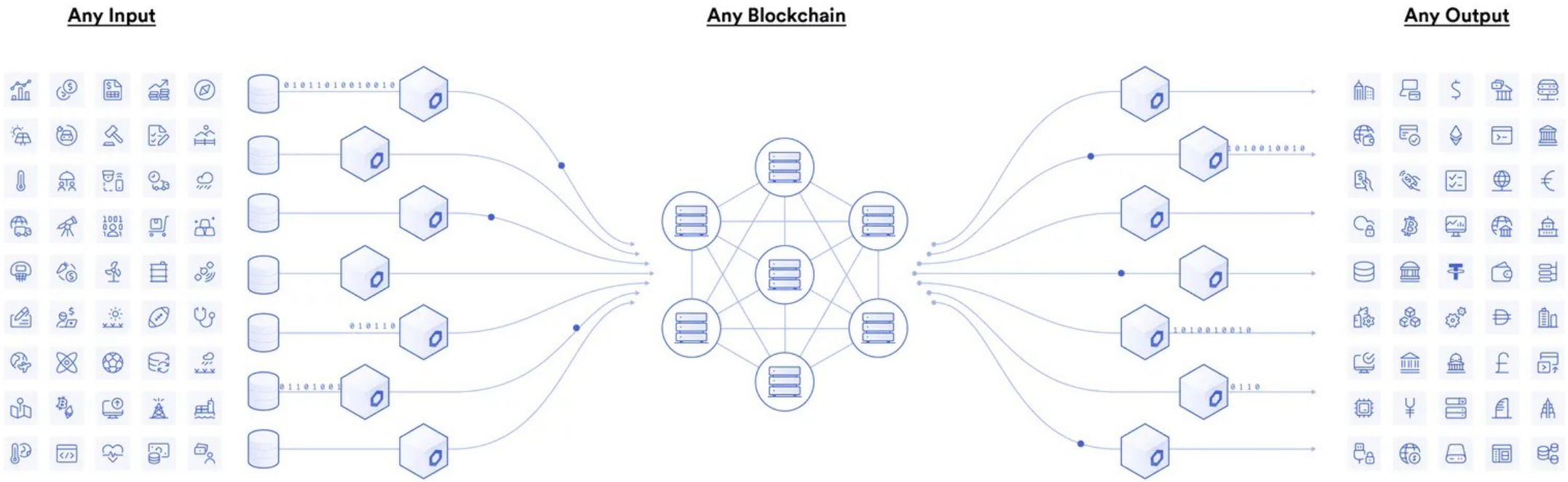
OUTPUT

Centralized Oracle



Centralized oracles are a **single point of failure**

Decentralized Oracle



Combines **multiple independent oracle** node operators and multiple reliable data sources to establish end-to-end **decentralization**

Take away points on Smart Contracts and Web3

Take away points

- Smart contracts are **as good as people writing them**
  - a smart contract that receives bad or incorrect information from the network will still execute, possibly propagating errors
- **Testing is fundamental** before deployment since it is critical to check for every possible way things could go wrong (and it happened)
- In case of misbehavior, **who is responsible**? The legal validity of smart contracts is still an open problem

Take away points

- Smart contracts allow for the development of dApps and the **overall ecosystem is complex**
- Also, remember that there is no way to guarantee that the current value of cryptocurrencies and other tokens will be maintained in the future
- There is much more over there, **we scraped only the surface**…

Take away points

Now it is your turn, enjoy!


Start
DEMO