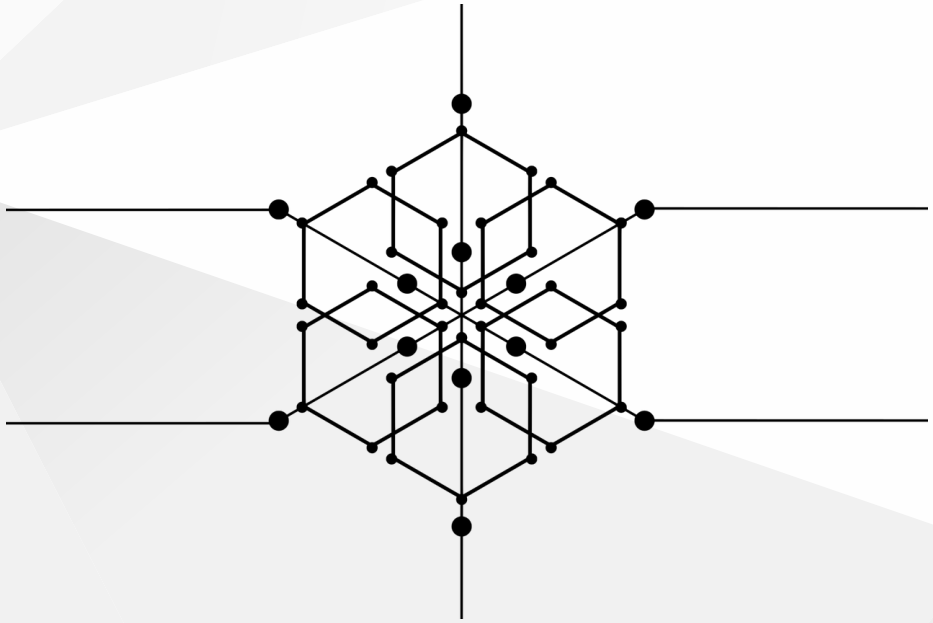


Blockchain 101

Introdução a Aplicações Descentralizadas (Dapp)

Thiago Nóbrega
https://github.com/thiagonobrega

Aula Passada

Blockchain

- Conceitos de um computador global (world computer)

  Permissionaria (permissioned)

  Não permissionaria

- Caracterísiticas

  Decentralizado

  Imutável (*Temper Evident*)

  Transparente/Auditável

  Pode ser utilizado por adversarios (*Distrustful Parties*)

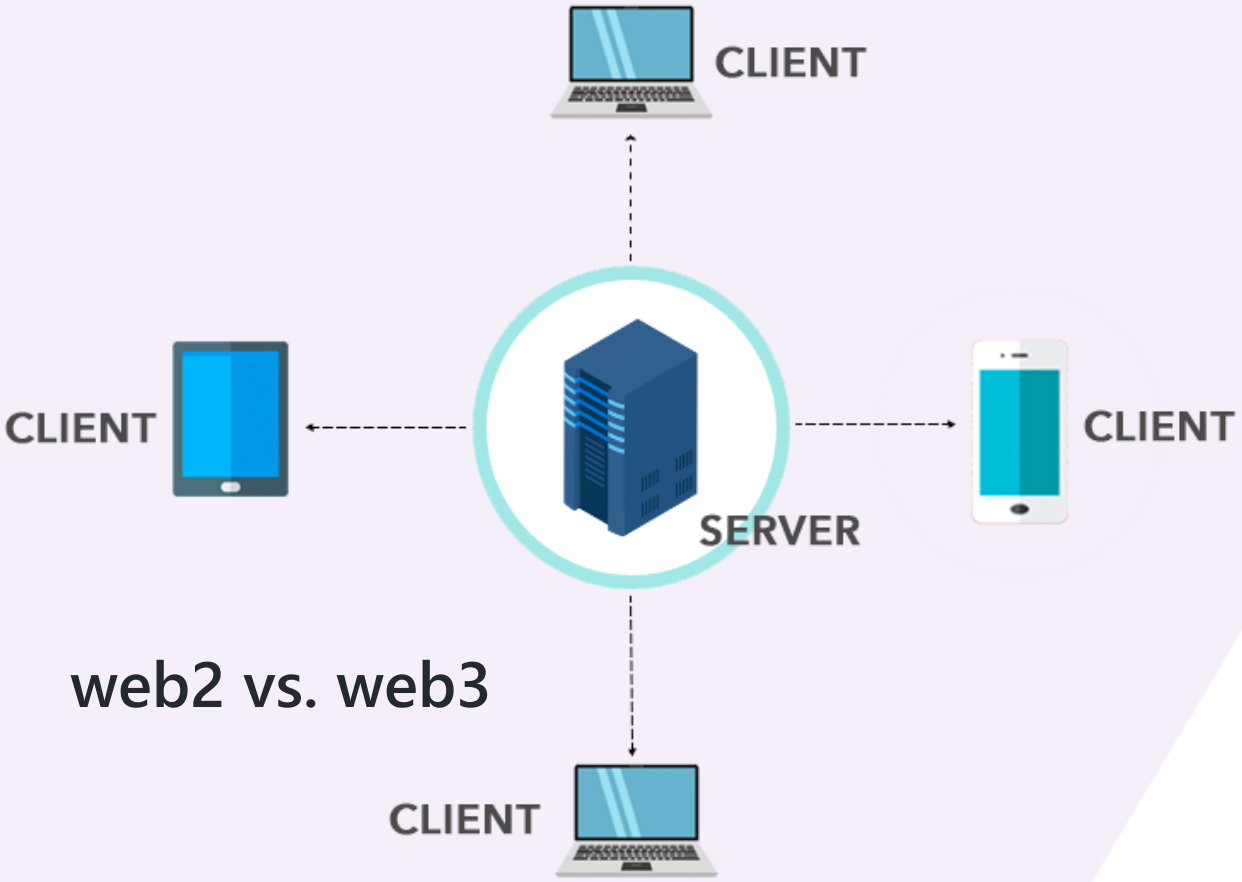
Introdução a Aplicações Descentralizadas (Dapp)

Agenda

- Aplicações Descentralizadas
- Conceito de "world computer"
- Ethereum Virtual Machine (EVM)
- Accounts
- Blockchain properties
- Smart contracts

Aplicações Descentralizadas

web3



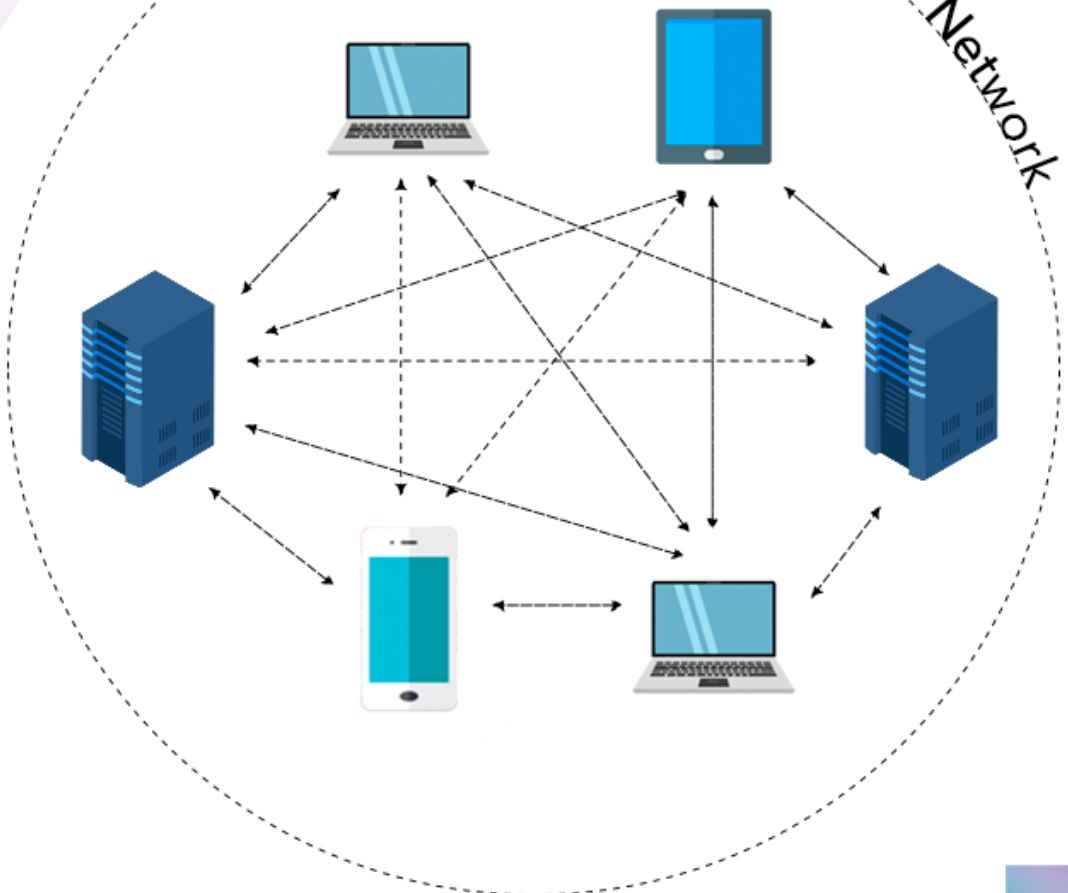
web2 vs. web3

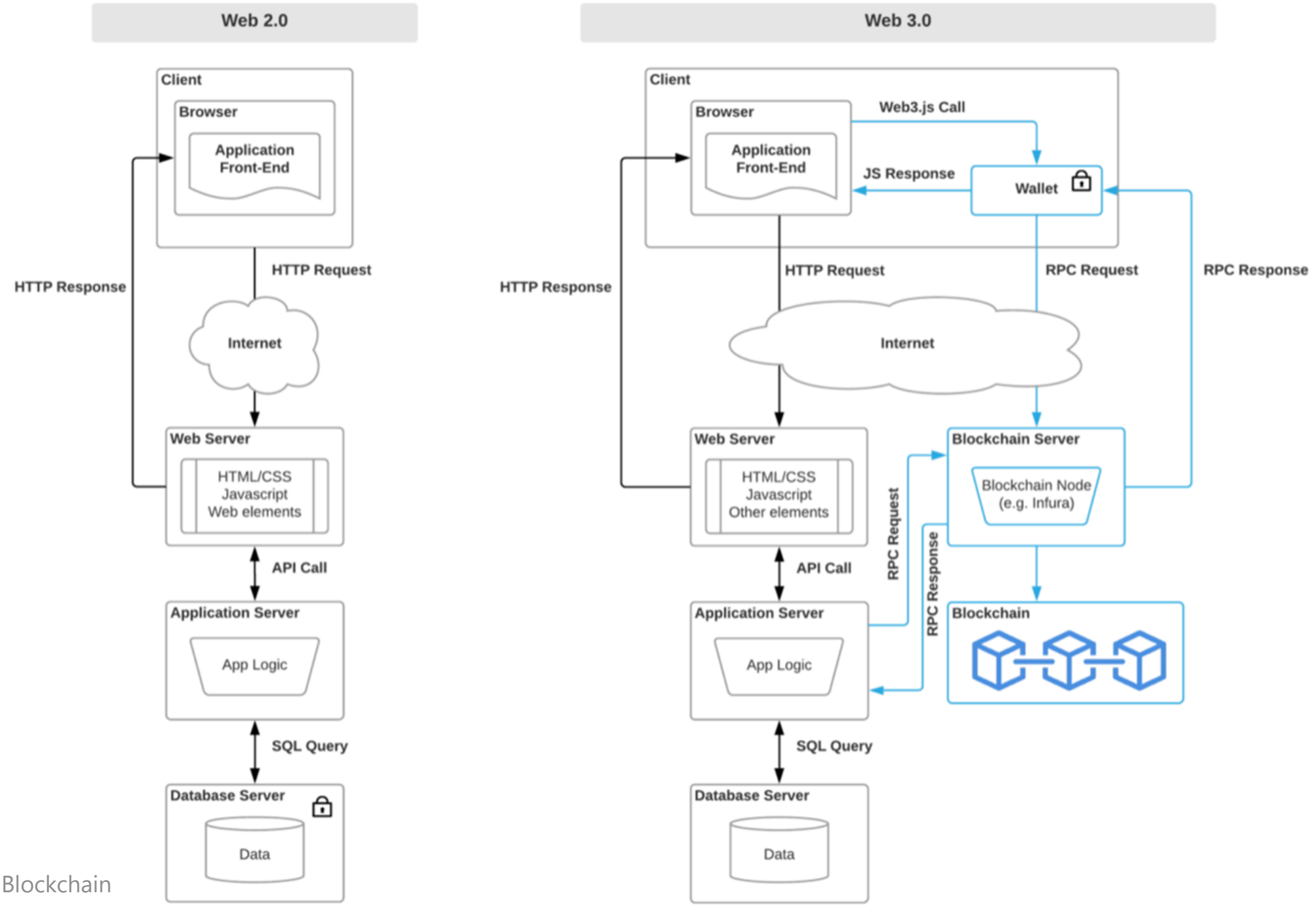
WEB2

Data & apps stored on servers

WEB3

Data & apps owned by many and stored on servers throughout the Web3 network




Web 2.0
Client
Browser
Application
Front-End
HTTP Request
HTTP Response
Internet
Web Server
HTML/CSS
Javascript
Web elements
API Call
Application Server
App Logic
SQL Query
Database Server
Data
Web 3.0
Client
Browser
Application
Front-End
Web3.js Call
JS Response
Wallet
HTTP Request
HTTP Response
RPC Request
RPC Response
Internet
Web Server
HTML/CSS
Javascript
Other elements
Blockchain Server
Blockchain Node
(e.g. Infura)
RPC Request
RPC Response
API Call
Application Server
App Logic
Blockchain
SQL Query
Database Server
Data

Web3

- Auditavél
- Não depende de confiança
- Autogerida
- Não depende de permissões
- Distribuída e robusta
- Baseada em estados
- Com pagamentos integrados nativamente

Aplicações Web3

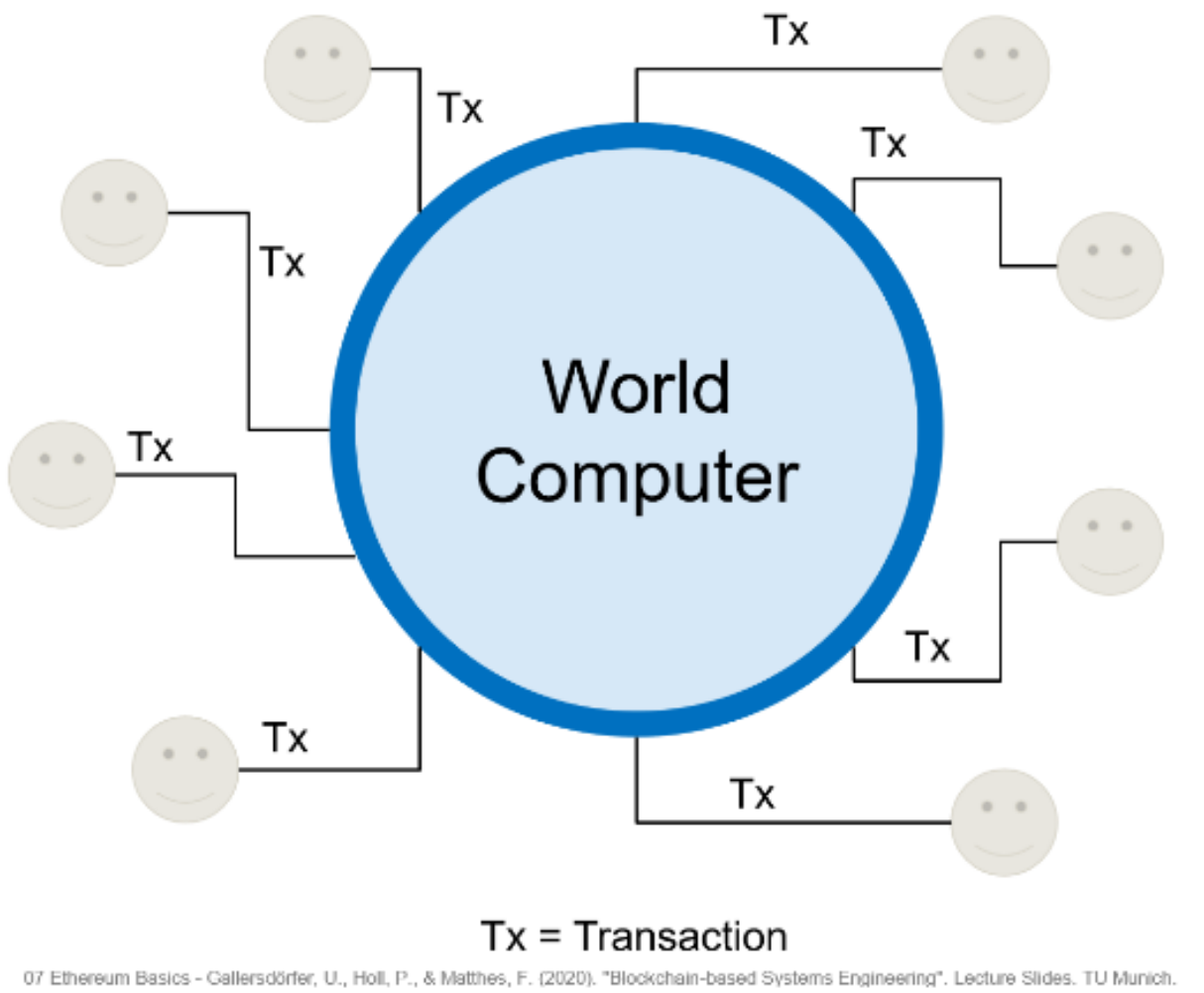
- Games
- NFTs
- DeFi (finanças descentralizadas)

> Ethereum, Fundo de Investimentos, etc.

- “Decentralized Autonomus Organization” (DAOs)

> Financiar Projetos
> Leiolões de obras raras
> contratar proficionar

"world computer"



O conceito de "world computer"

Maquina de Estado

- Todos utilizam a mesma maquina
- Usuários enviam transações para invocar os programas
- Memória e compartilhada

Imagem:Ulrich Gallersdörfer - Technical University of Munich

Programa

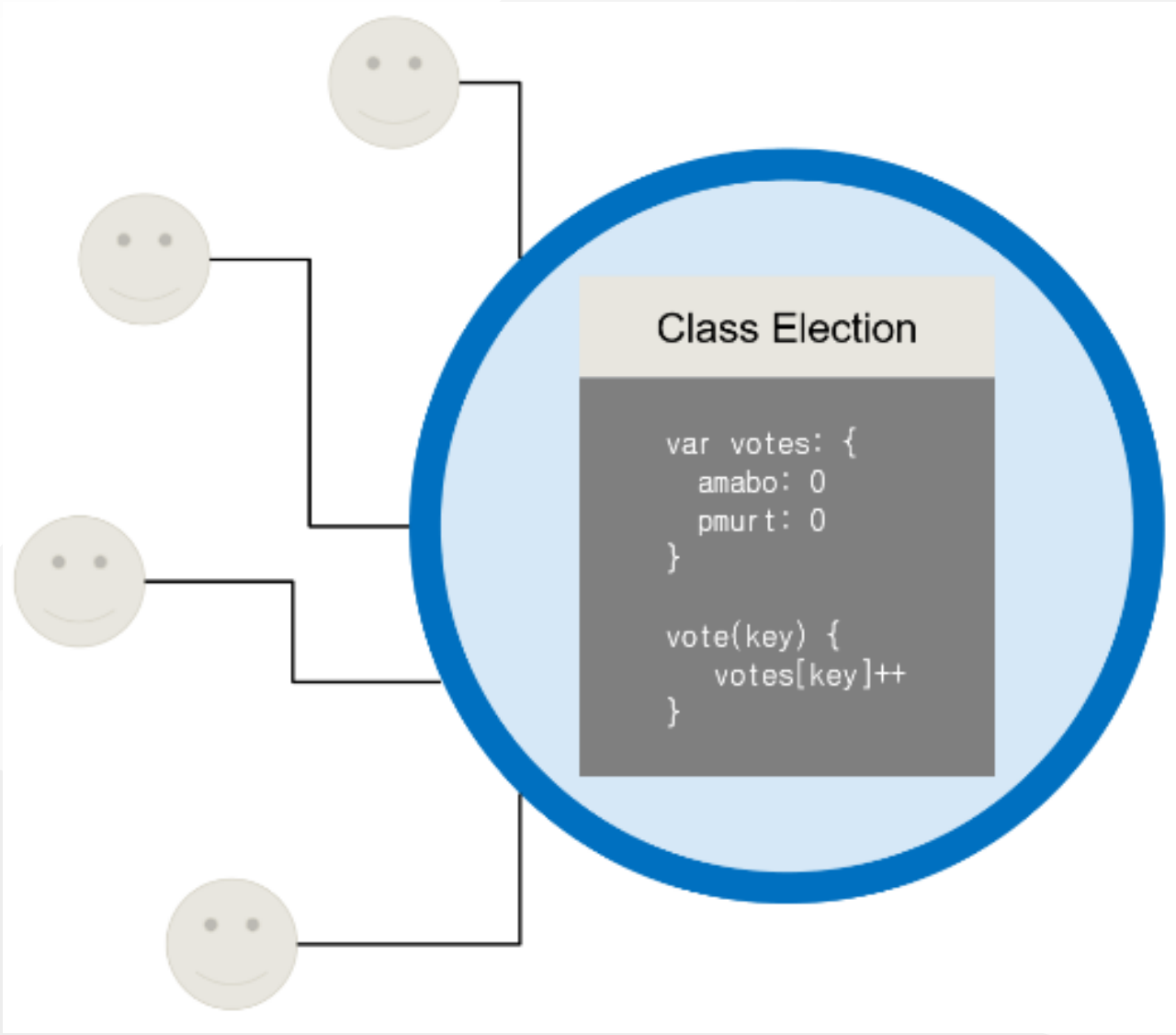


Imagem:Ulrich Gallersdörfer - Technical University of Munich

Máquina de estado na Blockchain

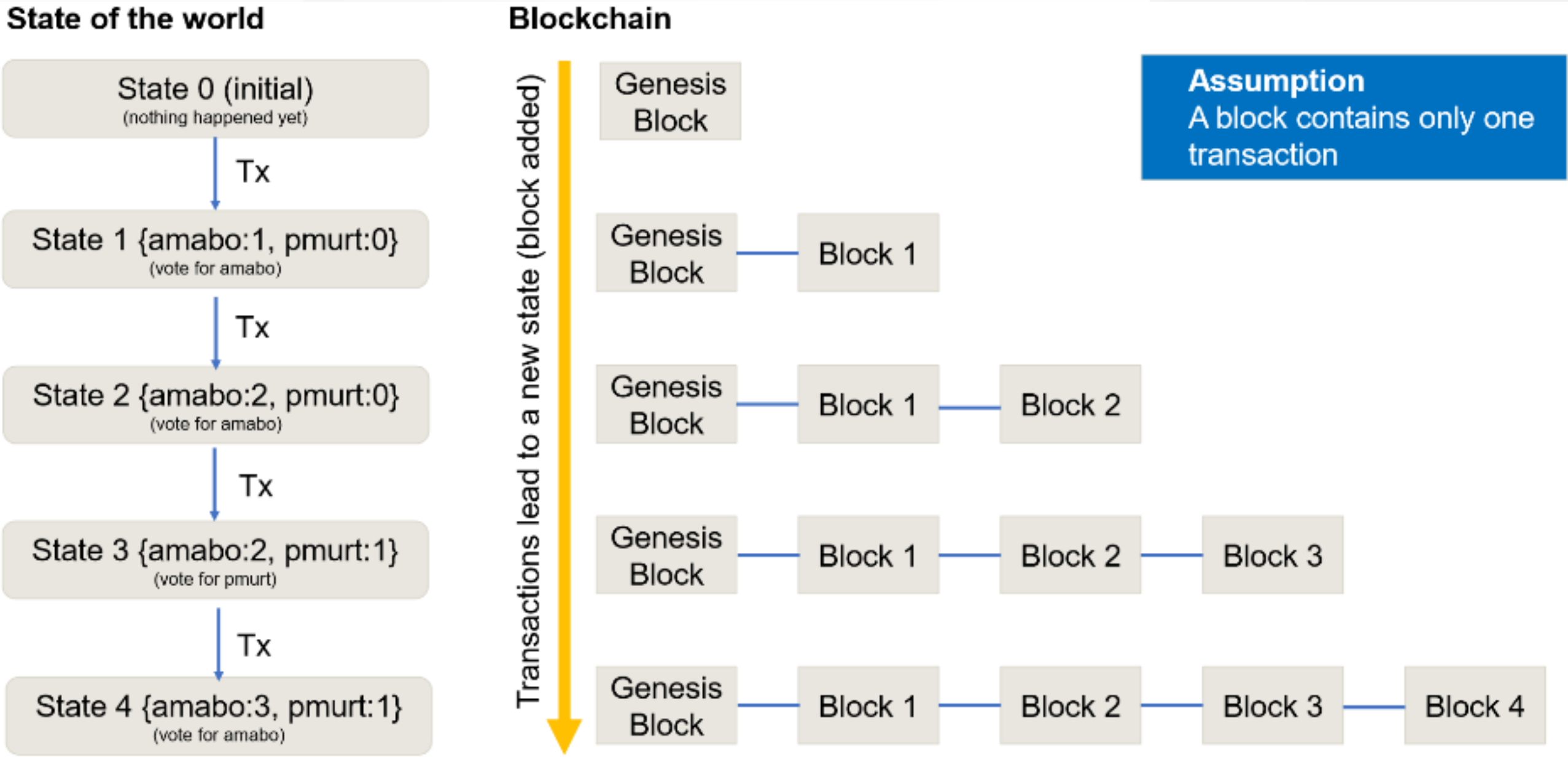
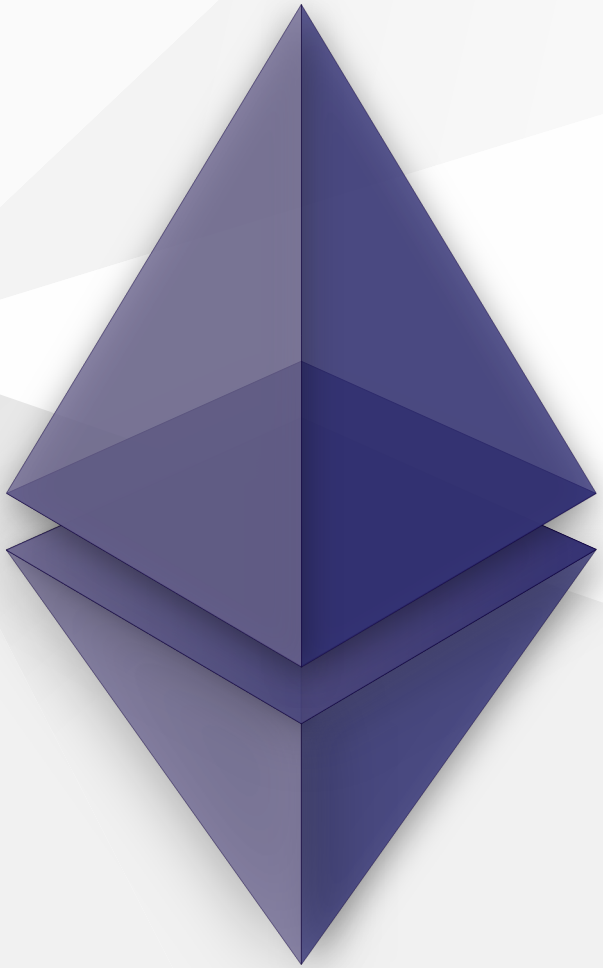


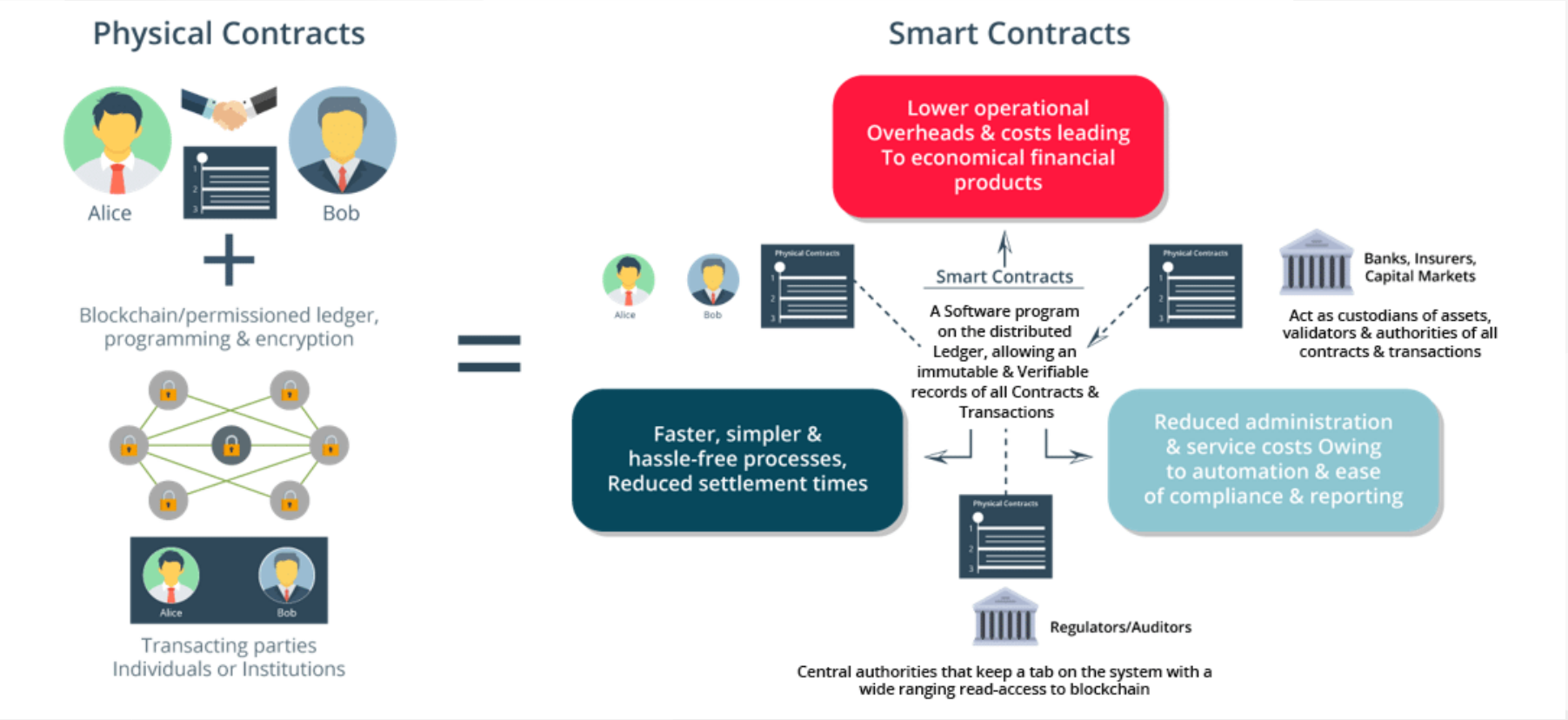
Imagem:Ulrich Gallersdörfer - Technical University of Munich



Ethereum Virtual Machine (EVM)

- Proof-of-Work (main net) ---> Proof-of-Stake
- Dificuldade é ajustada a cada bloco
- Gas limit*

Contratos Inteligentes



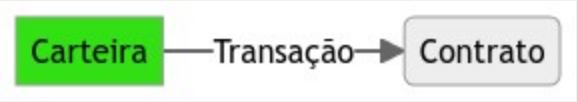
Contratos Inteligentes

Caso de uso

- Token systems (USD, Binance + 1026 outros tokens)
- Identity and reputation systems
- Decentralized Autonomous Organization (DAO)
- Election and voting systems

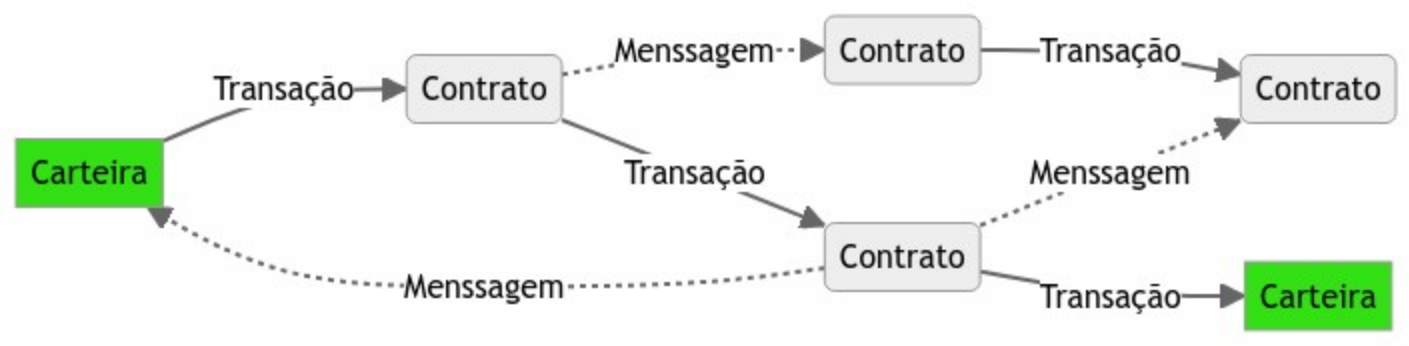
Ethereum Virtual Machine (EVM)

Transaction



Ethereum Virtual Machine (EVM)

Transactions



Solidity

Propriedades da Solidity

- Sintaxe similar ao JavaScript
- Estaticamente tipada
- Orientadas a Objeto
  - Suporte a herança
- Dynamic binding
  - compilada em EVM opcode

Característica da Solidity

Tipos nativos

> inteiros (int, uint)
> ~~ponto flutuante~~
> bytes
> string*
> boleanos (bool)
> array
> struct
> enum
> mapping

Característica da Solidity

Funções Built-in

- **Tratamento de erro**: assert(), require(), revert()
- **Math & Crypto**: addmod(), mulmod(), sha3(), keccak256(), sha256(), ripemd160(), ecrecover(), (+,-,mod,/,*)
- **Block info**: gasleft(), blockhash()
- **Contrato**: selfdestruct()

Flow control

- if, else, do, while, break, continue, for, return, ? ... : ... (ternary operator)

Característica da Solidity

Constructor

- Executado quando o contrato e criado por uma transação;
- Não pode ser executado apos a criação;
- Utilizado para configurar o contrator;
- Instruções complexas podem aumentar o custo (gas) de deployment do contrato;

Característica da Solidity

Funções

<table style="width:100%;"> <tr> <th style="width:60%;text-align:left">

```
function <name>([<parameter types>])
{internal|external|public|private}
[pure|constant|view|payable]
[(modifiers)]
[returns (<return types>)]
```

</th> <th style="text-align:left">

- Functions are used to change the state of a contract.
- Can also be used to read the state of the contract.
- Consist of a name, a signature, a visibility,

a type, a list of modifiers, and a return

type.

</th> </tr> </table>

Visibilidade de Funções

<table style="width:100%;"> <tr> <th style="width:50%;text-align:left">

```
function <name>([<parameter types>])
**{internal|external|public|private}**
[pure|constant|view|payable]
[(modifiers)]
[returns (<return types>)]
```

</th> <th style="text-align:left">

Visibilidade

- **Public**
- **External** : não pode ser acessado pelo prorio contrato
- **Internal** : ~External + contrados derivados deles
- **Private** : Apenas pelo contrato

</th> </tr> </table>

Special function types

<table style="width:100%;"> <tr> <th style="width:50%;text-align:left">

```
function <name>([<parameter types>])
{internal|external|public|private}
**[pure|constant|view|payable]**
[(modifiers)]
[returns (<return types>)]
```

</th> <th style="text-align:left">

Visibilidade

Leitura (não aletram a chain)

- **view** : pode ler estado de variaveis
- **pure** : não pode ler estado de variaveis

Escrita (a transação precisa ser inserida)

- **playable** :

</th> </tr> </table>

```
uint state = 5;

function add(uint a, uint b)
public view
returns (uint sum) {
    return a + b + state
}
```

</th> <th style="width:33%;text-align:left"> pure

```
function add(uint a, uint b)
public view
returns (uint sum) {
    sum = a + b;
}
```

</th> <th style="width:33%;text-align:left"> playable

```
unit = total;

function add(uint a, uint b)
public {
```

Laboratório

https://remix.ethereum.org

https://github.com/thiagonobrega/bc101