Universidade
Estadual de Londrina

Software baseado em Blockchain

Prof. Bruno Bogaz Zarpelão

Especialização em Engenharia de Software

Fundamentos Básicos

UEL

Introdução

- O que é uma blockchain?
- O que são blocos?
- O que são transações?
- Quais as características dos tipos de blockchain que temos (permissionless e permissioned)?
- O que é o Problema dos Generais Bizantinos?
- O que é um mecanismo de consenso?
- O que é o ataque de 51%?

Blockchain

- Blockchain é uma estrutura de dados que implementa uma distributed virtual ledger.
- Por isso, também usamos o termo DLT (Distributed Ledger Technology).
- Foi criada em 2009 como a estrutura que dá suporte ao Bitcoin.

Bitcoin

- Uma ***cryptocurrency*** ou moeda eletrônica (***e-cash***) criada em 2008 por Satoshi Nakamoto.
- O Bitcoin e a blockchain são uma solução bastante elegante para o problema conhecido como ***double spending***.
- O Bitcoin e sua blockchain são construídos para termos um sistema totalmente distribuído, eliminando qualquer elemento central.

Bitcoin, Blockchain e a Criptografia

- O Bitcoin utiliza ferramentas de criptografia para alcançar seus objetivos:
  - Algoritmos de hash.
  - Criptografia de chave pública.
  - Assinatura digital.

Blockchain e Redes de Computadores

- Sistemas baseados em blockchain são, normalmente, estruturados sobre redes P2P.

Visão Geral

1. Usuários gravam transferências de valores em transações.
2. Usuários enviam transações para seus pares na rede.
3. Mineradores acumulam transações até que possam criar um bloco.
4. Mineradores **competem** pelo direito de publicar o próximo bloco.

Visão Geral

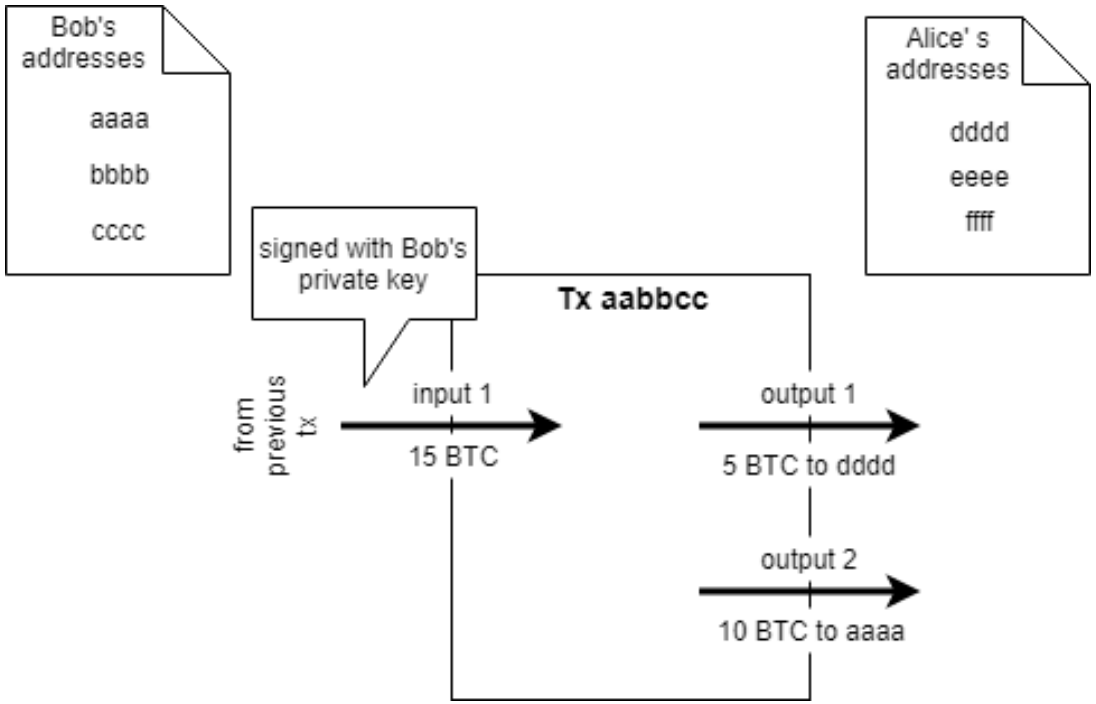
5. Minerador “vencedor” envia seu bloco para **todos** os outros pares na rede.
6. Usuários que recebem esse novo bloco o **verificam** e, caso esteja correto, anexam à sua cópia da blockchain.
7. Minerador recebe recompensa por ter publicado o bloco.
8. Todo bloco publicado na blockchain armazena o valor do código hash do bloco anterior, formando uma cadeia de blocos.

Visão Geral

- **Grande desafio:** a visão que os nós têm da blockchain deve ser consistente.
- O mecanismo de consenso é uma forma de buscar garantir essa consistência (ou consenso).

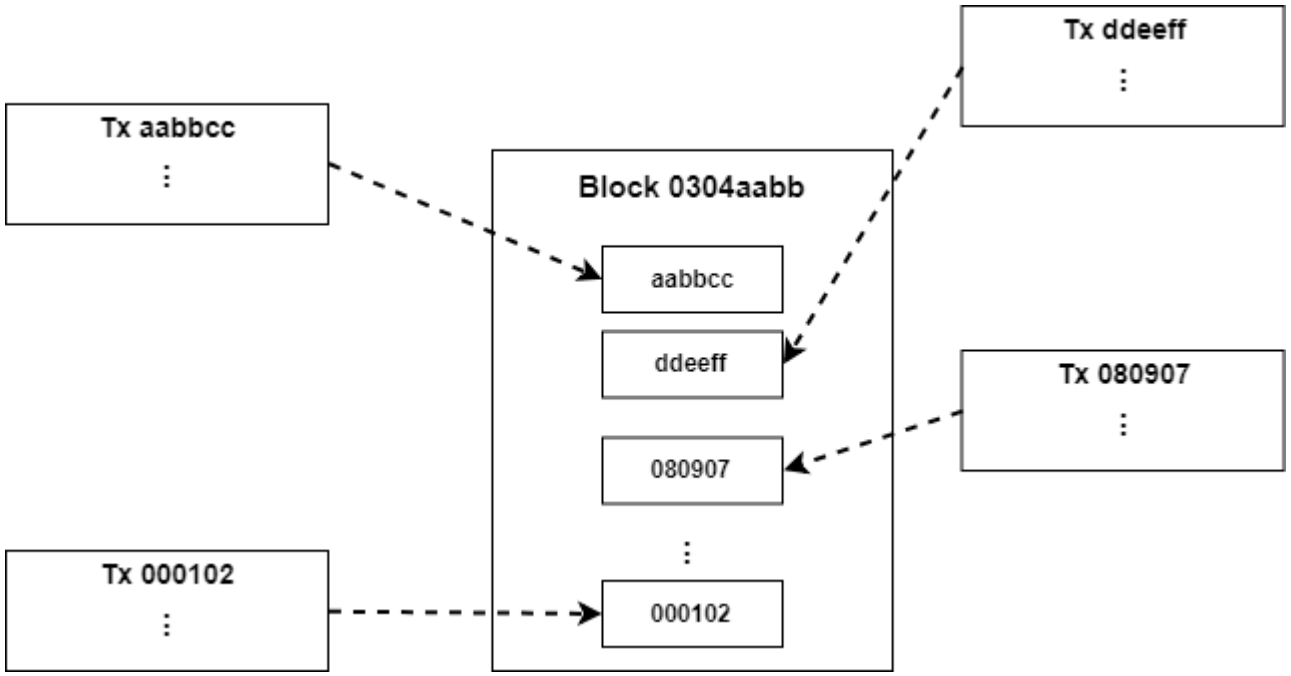
Bitcoin

- Bob tem 15 BTC que recebeu em uma transação anterior e quer transferir 5 BTC para Alice.



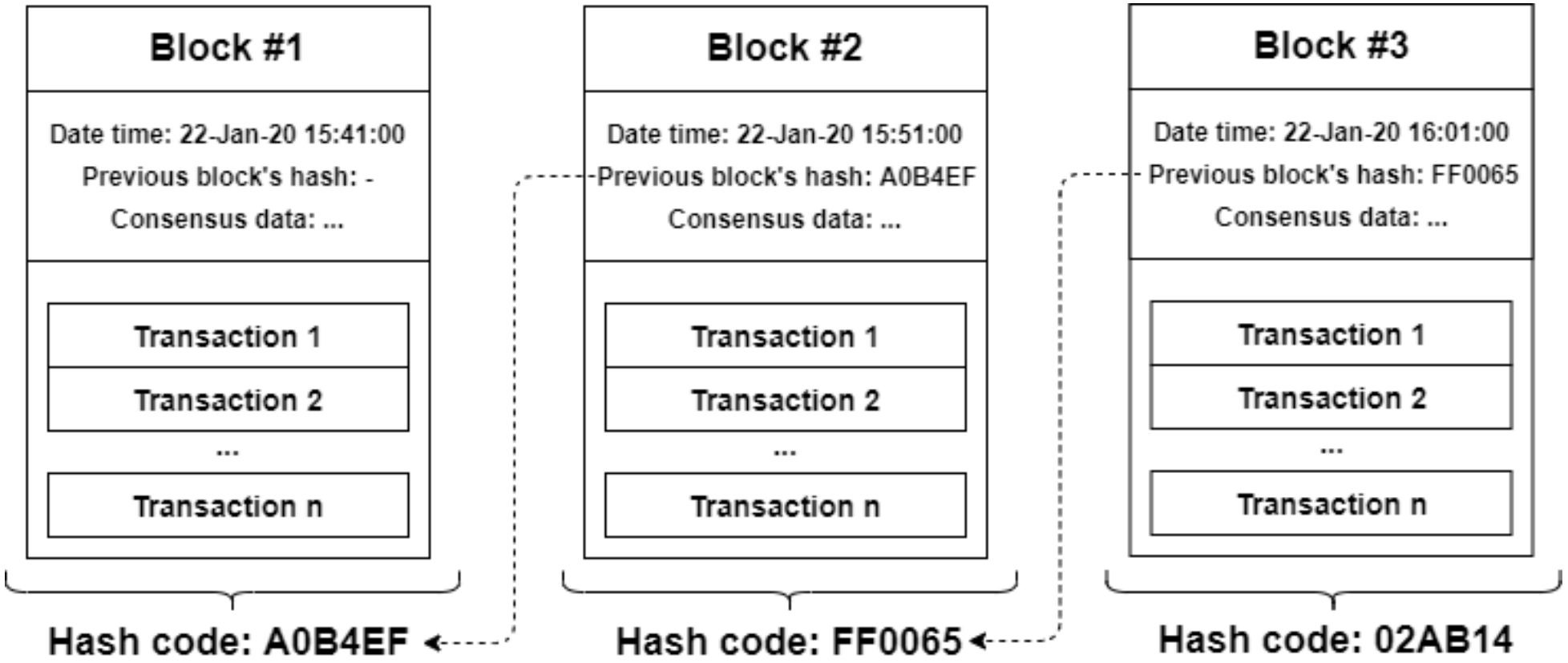
Bitcoin

- Bob envia esta transação como um broadcast para os outros nós da rede Bitcoin que ele conhece.
- Cada minerador reúne as transações recebidas e coloca em um bloco.



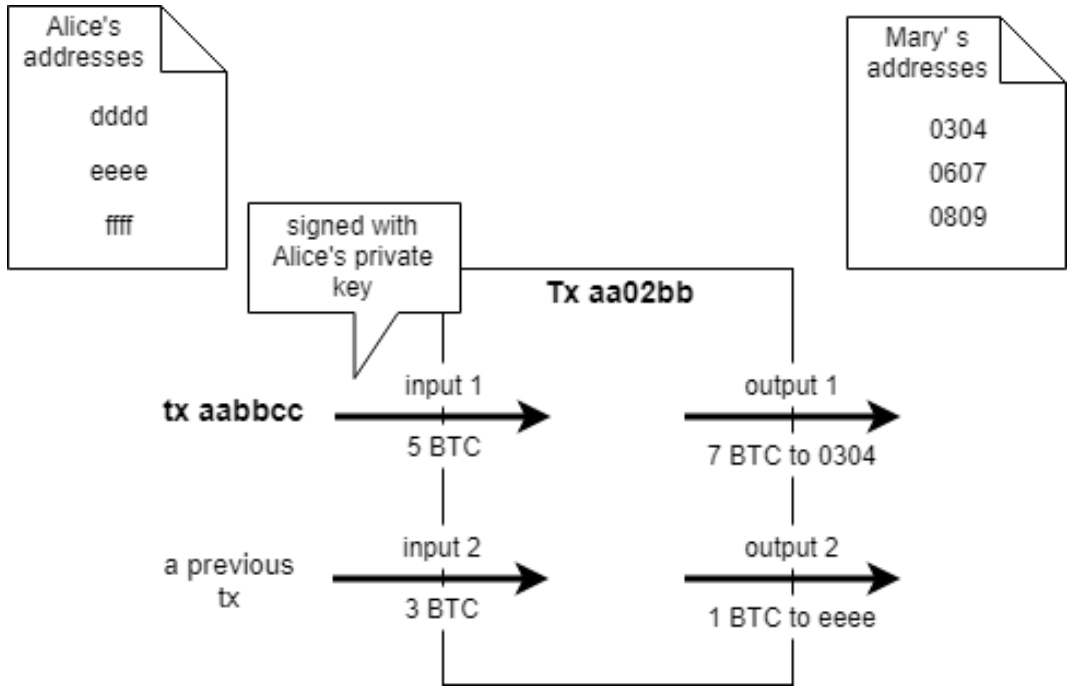
Bitcoin

- Cada bloco novo é associado ao bloco anterior por meio de seu código *hash.*



Bitcoin

- Quando o bloco é publicado e validado, Alice pode gastar este valor. Ela então transfere 7 BTC para Mary.



Exercício

- Como o Bitcoin garante que ninguém gere uma transação falsa para transferir um valor que a conta origem não tem?

Permissionless blockchains

- Qualquer indivíduo pode instalar um software cliente e se juntar à rede.
- Qualquer indivíduo pode se tornar um minerador, mas a publicação de novos blocos depende do mecanismo de consenso.
- Ambiente aberto, sem controle centralizado, com transações sendo disseminadas em redes P2P.
- Exemplos: Bitcoin e Ethereum.

Redes P2P

- Surgiram em 1999 para compartilhamento de arquivos entre usuários da Web.
- Nós podem agir como clientes ou servidores conforme a demanda.
- Não é construída uma infraestrutura específica. Cada nó usa a sua própria infraestrutura.
- Não há controle central e qualquer indivíduo com um computador pode se juntar à rede.

Redes P2P

- Diferentes maneiras de disseminar/localizar conteúdo e encontrar nós próximos:
  - DHT (Distributed Hash Table).
  - Listas pré-definidas de nós mais robustos.
  - Disseminação de conteúdo por inundação.
- Bitcoin é implementado como uma rede P2P: https://developer.bitcoin.org/devguide/p2p_network.html

Permissioned blockchains

- Sistemas que pertencem a um grupo fechado de organizações ou indivíduos.
- A atribuição de papeis dentro do sistema pode depender de certos requisitos.
- Solução normalmente utilizada em ambientes corporativos.
- Exemplo: sistemas desenvolvidos a partir do framework HyperLedger.

Problema dos Generais Bizantinos

- Problema discutido inicialmente no artigo “The Byzantine Generals Problem” de Lamport et al., publicado em 1982 no periódico ACM Transactions on Programming Language and Systems.
- A história dos generais bizantinos é uma alegoria para discutir o comportamento de sistemas distribuídos na presença de falhas (ou nós maliciosos).

Problema dos Generais Bizantinos

- Suponha que temos 6 divisões do exército Bizantino cercando uma cidade inimiga.
- Cada divisão tem seu general e todos devem entrar em acordo com relação ao plano de ação.
- As divisões estão separadas por uma longa distância, então a ordem será transmitida entre as divisões por um mensageiro.

Problema dos Generais Bizantinos

- Entre os generais, podem haver traidores.
- Há, portanto, o risco de que eles modifiquem maliciosamente a mensagem para sabotar a ação planejada.

Cenário 1: os traidores vencem!

- O general 1 decide atacar às 22h00. Ele envia então o mensageiro com esta mensagem para o general 2.
- O general 2 recebe a mensagem, registra e devolve ao mensageiro, que a leva ao general 3 (traidor).
- O general 3 recebe a mensagem, rasga e prepara outra mensagem falsa: “Atacar às 21h00”. Ele a repassa ao mensageiro.
- Os outros 3 generais (4, 5 e 6) recebem a mensagem falsa.

Cenário 1: os traidores vencem!

- As divisões 4, 5 e 6 atacam às 21h00, mas são poucos e acabam dominados e derrotados.
- As divisões 1 e 2 atacam às 22h00, mas também são derrotados por estarem em larga desvantagem numérica.

Cenário 2: os traidores não têm chance!

- O general 1 envia a mesma mensagem (“Atacar às 22h00”), mas agora temos mais duas regras:
  - É usada uma máquina que demora 10 minutos para escrever uma nova ordem dentro da mensagem.
  - A mensagem deve incluir todo o histórico de ordens dadas pelos generais anteriores.

Cenário 2: os traidores não têm chance!

- O general 2 recebe a mensagem e, utilizando a máquina escreve a ordem (“Atacar às 22h00”) na mesma mensagem. Após 10 minutos, o mensageiro parte com a mensagem para o general 3 (traidor).

Cenário 2: os traidores não têm chance!

- O capitão 3 tem duas opções:
  - Gerar uma mensagem falsa. Para isso, ele precisa gerar uma nova mensagem com 3 ordens falsas, sendo que cada ordem leva 10 minutos para ser escrita. O porém é que o mensageiro parte em 10 minutos.
  - Aceitar que não tem como corromper o sistema e repassar a mensagem correta.

Mecanismo de Consenso

- O mecanismo de consenso é utilizado para garantir a consistência entre as cópias de blockchain armazenadas pelos diferentes nós.
- Busca evitar também fraudes na realização de transações.
- Mecanismo de consenso do Bitcoin: Proof-of-Work.

Proof-of-Work

- **A ideia é que o nó deva gastar recurso computacional para ter o direito de publicar um novo bloco.**
- Bitcoin usa um sistema chamado de hashcash:
  - O minerador deve adivinhar uma entrada para o algoritmo SHA256 que produza uma saída menor que um dado alvo.
  - O alvo é um número de 256 bits.
  - O alvo vai sendo modificado pra tornar a tarefa mais difícil.

Proof-of-Work

- Quando um minerador encontra a resposta para o desafio, ele a insere no novo bloco e envia para todos os outros nós na rede.
- Os nós, antes de adicionar o bloco à sua cópia da blockchain, verificam a resposta.
- Em tese, o minerador evita tentar publicar blocos fraudados ou incorretos, pois ele não receberá a recompensa.

Pergunta

- Conhecendo o conceito de hash criptográfico, assinale a alternative correta no que diz respeito ao desafio que compõe a prova de trabalho do Bitcoin:

a) Existe um “atalho” pra encontrar esta entrada, então o minerador deve ser perspicaz.
b) O único caminho possível é a força bruta, então tem maior possibilidade de vencer quem pode tentar mais vezes.

Pergunta

- Considerando que um usuário alega que encontrou a entrada que produza uma saída menor que o alvo, assinale a alternative correta:

a) É fácil para outro usuário verificar se a resposta dada está correta.
b) A verificação dessa resposta também exige um grande poder computacional.

Dificuldade da prova de trabalho

- No Bitcoin, a dificuldade é ajustada pela rede para que um bloco seja gerado a cada 10 minutos aproximadamente.

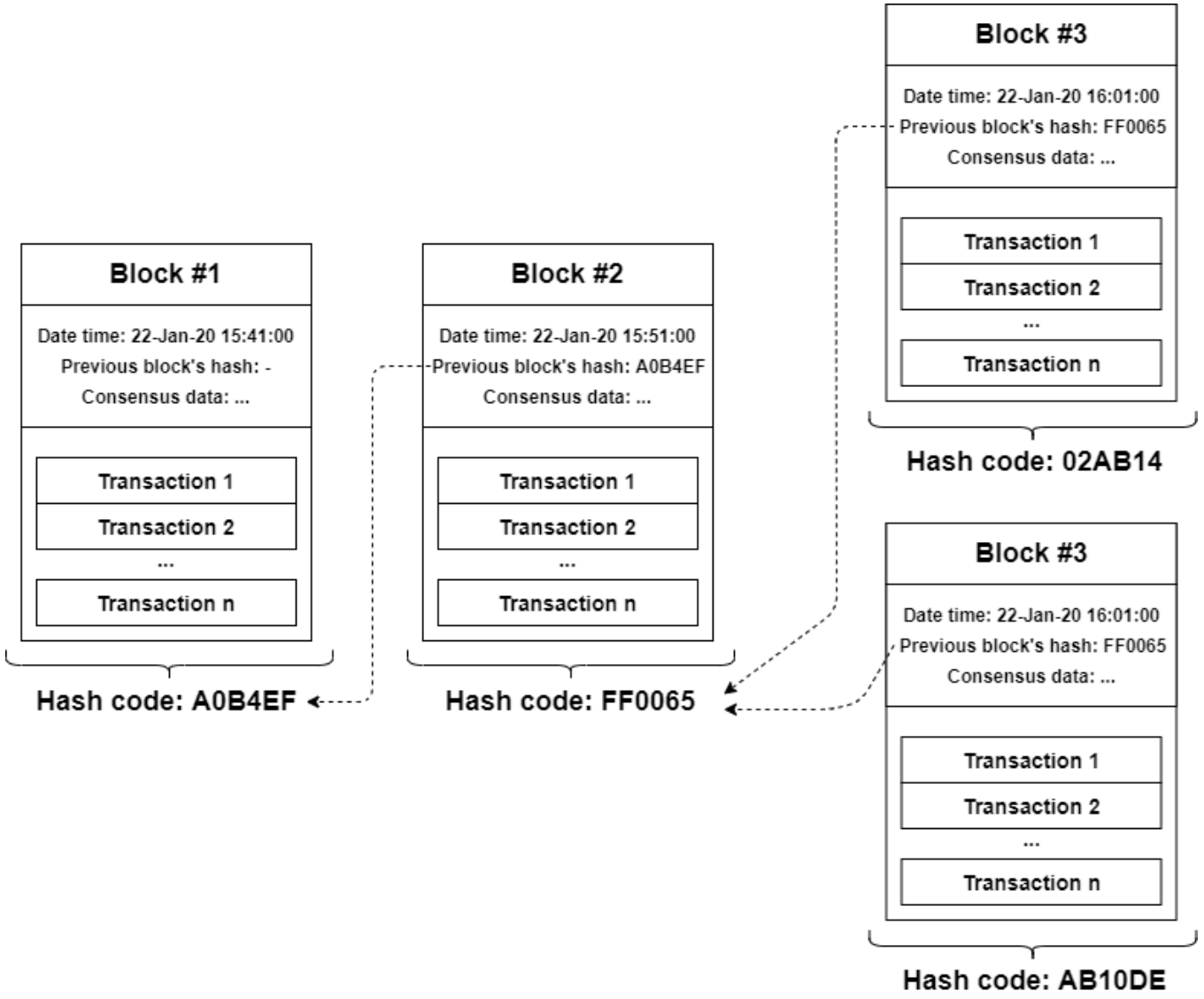
Exercício

- Escolha um dos mecanismos de consenso a seguir, faça uma busca na Web sobre ele, e explique como ele funciona para o restante da classe:
  - PoS (Proof of Stake)
  - PoET (Proof of Elapsed Time)
  - PBFT (Practical Byzantine Fault Tolerance)

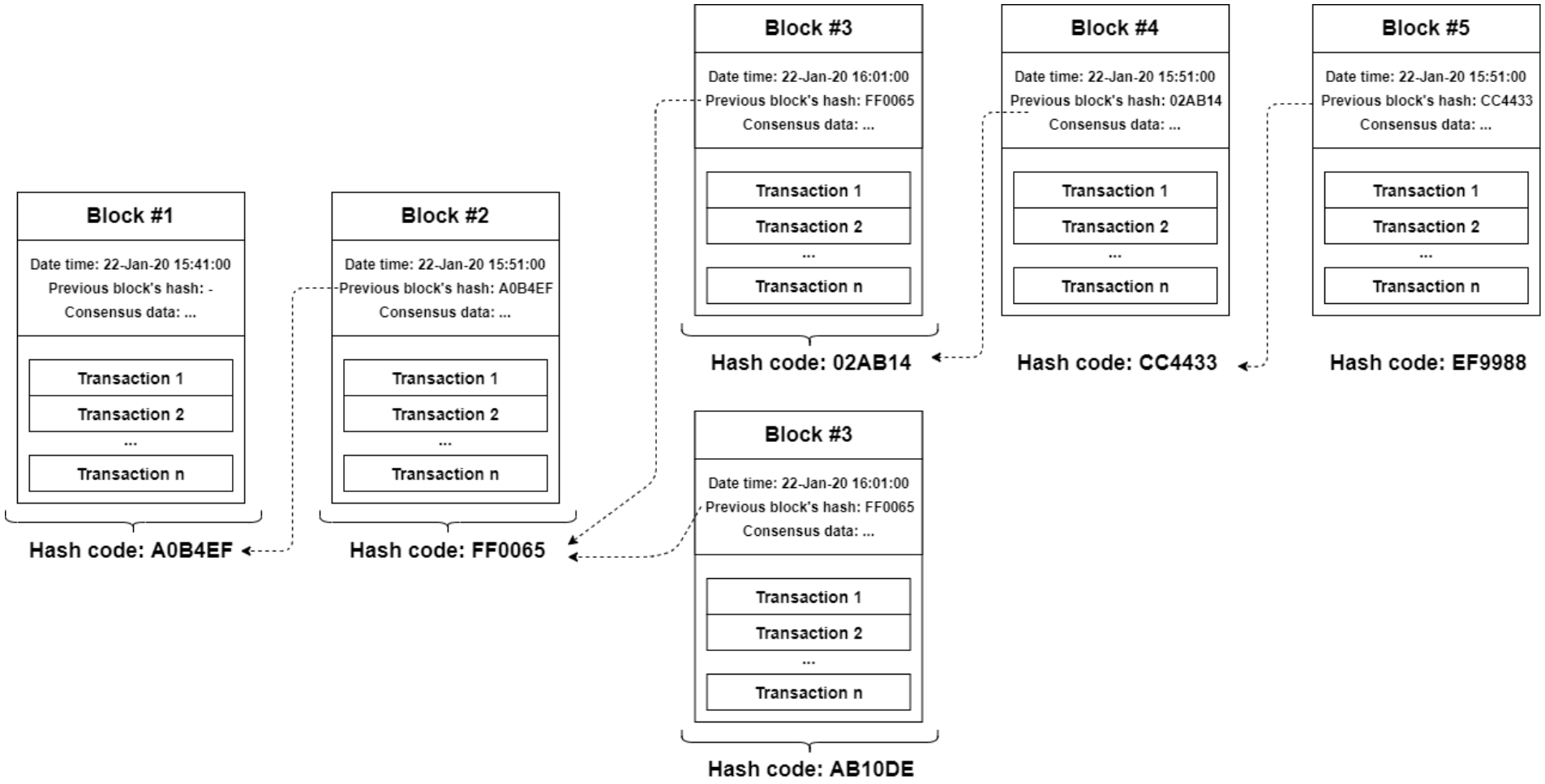
Blocos minerados ao mesmo tempo

- É possível que dois nós diferentes cumpram a prova de trabalho ao mesmo tempo (ou em momentos muito próximos) e enviem blocos válidos diferentes para outros nós.
- Neste caso, teremos dois novos blocos apontando para o mesmo bloco anterior.
- Também teremos nós armazenando blockchains diferentes (pelo menos por um tempo).

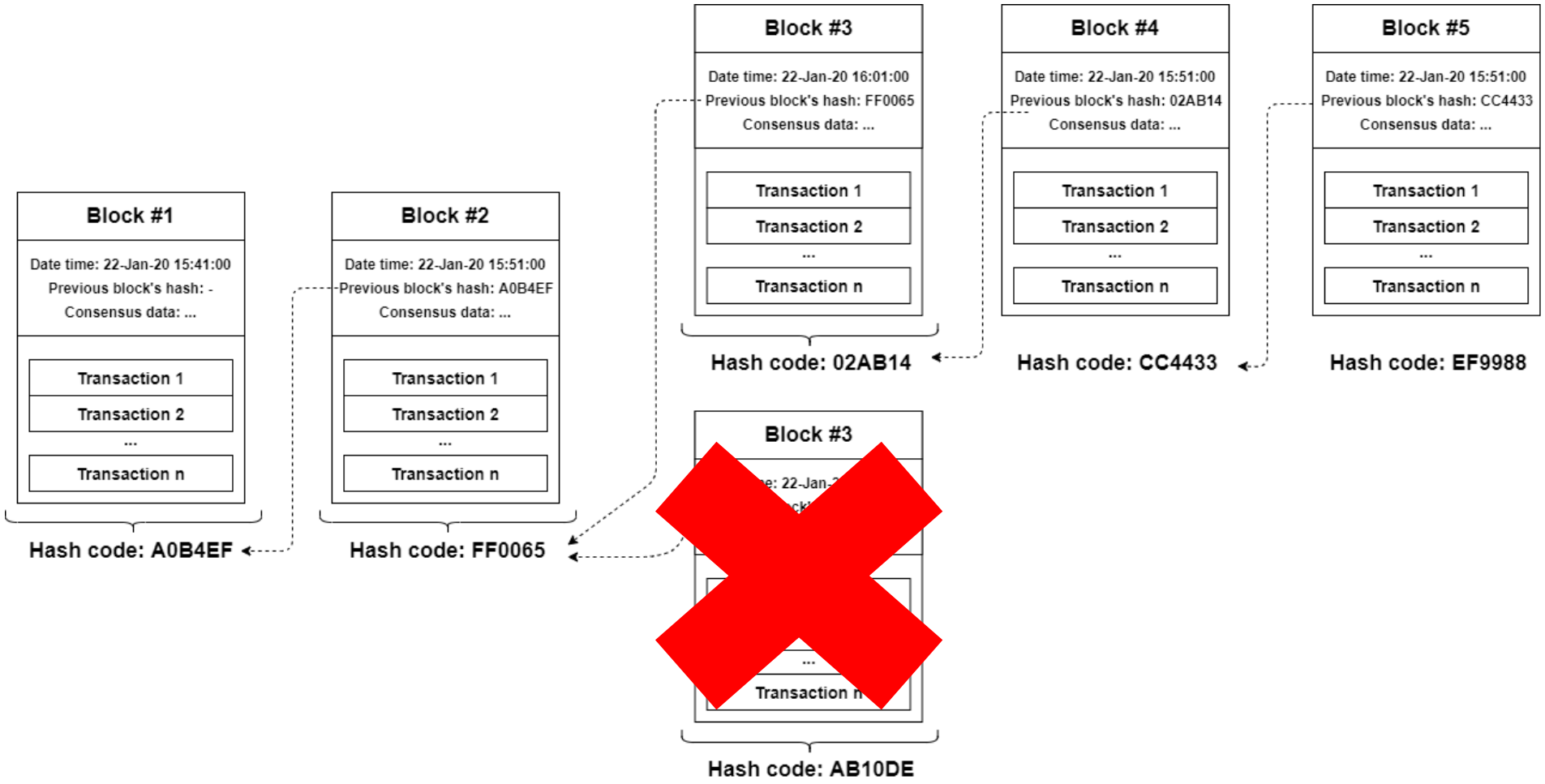
Blocos minerados ao mesmo tempo



Blocos minerados ao mesmo tempo



Blocos minerados ao mesmo tempo



Exercício

- Explique o que é o ataque dos 51%.

Bitcoin

Introdução

- Quais tipos de nós temos na rede Bitcoin?
- Qual é o método de assinatura digital utilizado no Bitcoin?
- Quais informações podem ser encontradas no cabeçalho de um bloco do Bitcoin?
- O que é uma Árvore de Merkle?

Características gerais

- Permissionless blockchain:
  - Basta instalar um aplicativo compatível e criar uma conta (gerar par de chaves) e você fará parte da rede.
- Tipos de nós:
  - Full node: valida blocos e transações, e dissemina novas transações.
  - SPV (Simplified Payment Verification): mantém uma cópia apenas dos cabeçalhos dos blocos e requisita cópias das transações a full nodes quando necessário.

Transação e assinatura digital

- Toda transação é assinada digitalmente pelo seu emissor.
- No caso do Bitcoin:
  - Todo *input* está ligado ao *output* de uma transação anterior (exceto no caso de *coinbase*).
  - O emissor da transação deve assinar a transação com a chave privada que faz par com a chave pública que identifica a "conta" de onde o valor será retirado.
- No Bitcoin, a assinatura digital é realizada por meio do ECDSA (Elliptic Curve Digital Signature Algorithm).

ECDSA no Bitcoin

- O Bitcoin segue o padrão secp256k1 para curvas elípticas, especificado pela Standards for Efficient Cryptography Group (SEC).
- A função hash usada no algoritmo de assinatura digital é o SHA256.
- A chave privada é um número de 256 bits.

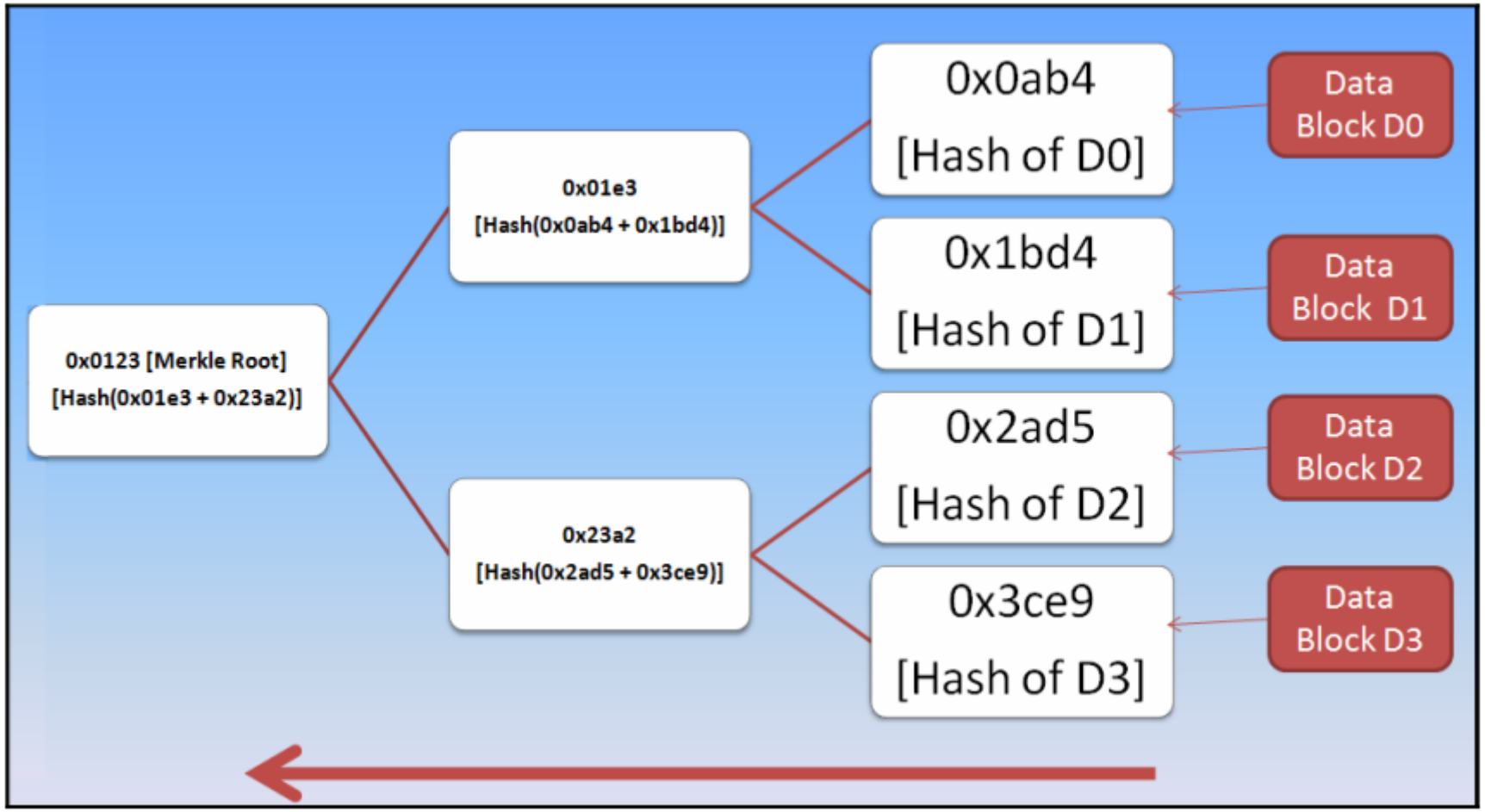
Bloco

- O cabeçalho de um bloco contém:
  - Bloco anterior: hash do bloco anterior.
  - Merkle root: código hash que representa todas as transações deste bloco.
  - Timestamp: registra quando o bloco foi criado.
  - Dificuldade: o alvo utilizado para geração deste bloco na prova de trabalho.
  - Nonce: valor resposta ao desafio da prova de trabalho.

Árvore de Merkle

- A árvore de Merkle é uma estrutura de dados onde cada nó é o hash da combinação dos códigos hash dos seus nós filhos.
- Geralmente, trata-se de uma árvore binária, i.e., cada nó tem no máximo dois filhos.
- Por meio desta estrutura de dados, conseguimos verificar a integridade de um conjunto de dados.
- Mais importante, conseguimos identificar onde a integridade foi quebrada.

Árvore de Merkle



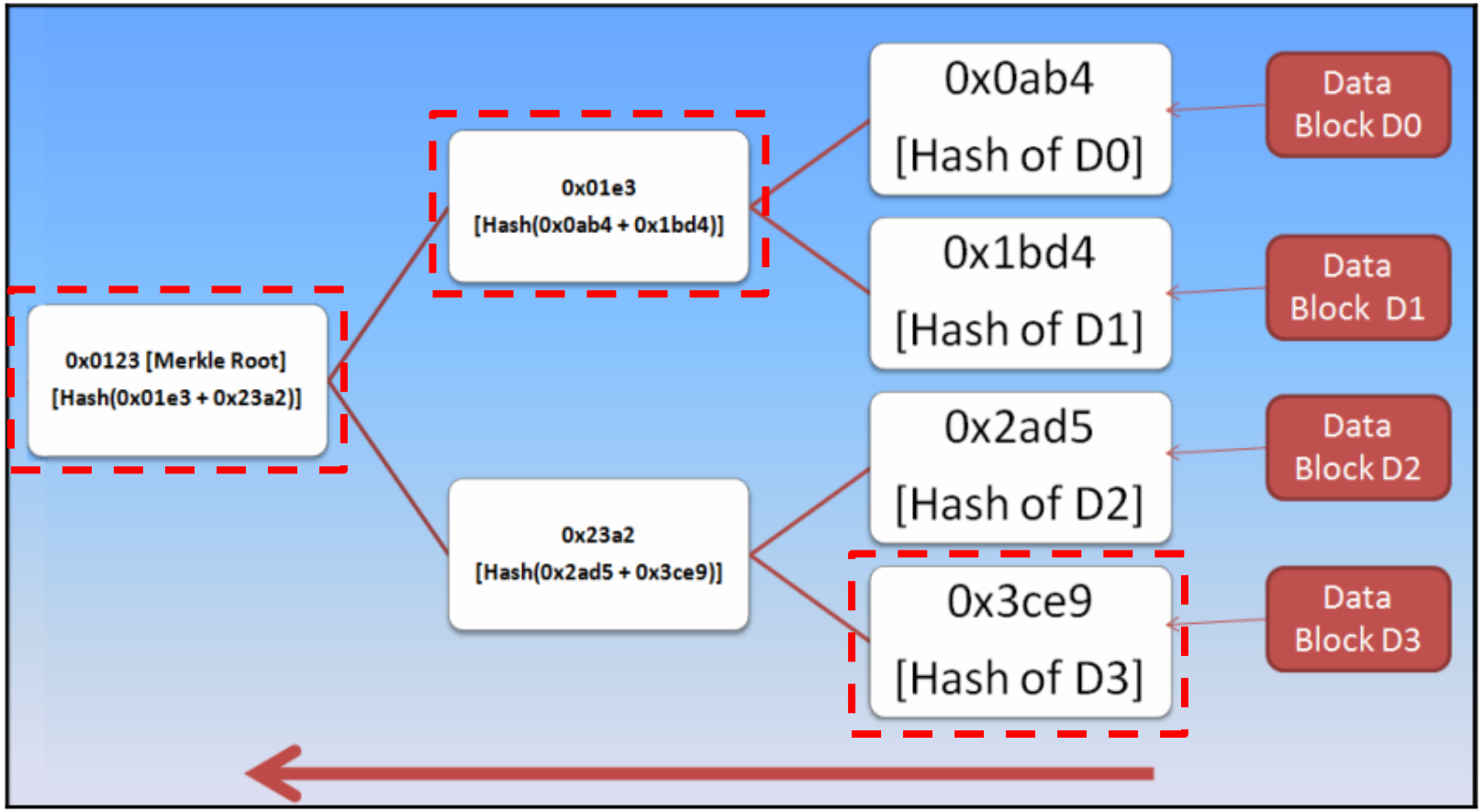
Fonte: Mukhopadhyay, M., “Ethereum Smart Contract Development”, Packt Publishing, Birmingham, UK, 2018.

Árvore de Merkle

- Vantagens:
  - Permite que se use apenas a raiz da árvore na hora de computar o hash de todas as transações do bloco.
  - Facilita o processo de verificar se uma dada transação está em um determinado bloco.

Árvore de Merkle

Para verificar se D2 está nesta árvore, preciso apenas dos nós marcados.



UEL

Ethereum

Introdução

- O que é o Ethereum?
- O que é Turing-completude?
- O que são DApps?

Introdução

- Pessoas começaram a buscar desenvolver outras aplicações com base na estrutura que suportava Bitcoin (blockchain, mecanismo de consenso, etc.).
- O Bitcoin não permitia isso, ficando mais restrito à questão monetária.
- Toda nova aplicação parecia demandar uma nova blockchain ou algum complemento (“gambiarra”) ao Bitcoin.

Ethereum

- No final de 2013, Vitalik Buterin propôs a ideia de uma blockchain com:
  - Suporte a linguagem Turing-completa.
  - Possibilidade de desenvolvimento de aplicações de propósito geral.
- De fato, o Ethereum foi pensado mais como uma espécie de framework para desenvolvimento de aplicações baseadas em blockchain:
  - Programador não precisa se preocupar com a rede P2P e mecanismo de consenso, por exemplo.

Ethereum

- Bitcoin:
  - Máquina de estados.
  - Cada estado registra quais donos tem qual montante de todo as moedas disponíveis.
  - Transações causam uma transição de estado.

Ethereum

- O Ethereum também é uma máquina de estados:
  - O estado é composto por dados de propósito geral.
  - Propósito geral: quaisquer dados que possam ser expressos no formato *chave:valor*.
  - Transições de estado são disparadas por transações ou mensagens.
  - O Ethereum permite a execução de código que altere o seu estado, e a evolução do estado é toda guardada na blockchain.

Ethereum

- Contratos inteligentes: programas que podem ser invocados por meio de transações emitidas por um usuário ou mensagens emitidas por outro contrato.
- Os contratos inteligentes são os códigos que alteram o estado do Ethereum.

Turing-completude

- Alan Turing propôs a máquina de Turing, que define os limites teóricos da computação.
- Um sistema é Turing completo se ele pode ser utilizado para simular qualquer máquina de Turing.
- O Ethereum tem esta capacidade e, portanto, é Turing-completo.
- Programas do Ethereum são executados por nós descentralizados, mas é mantido apenas um estado para este programa de acordo com o mecanismo de consenso.

Implicações da Turing-completude

- Turing demonstrou que precisamos executar um programa para saber se e como ele irá terminar.
- Existe o potencial de loops infinitos.
- Clientes do Ethereum, quando validam uma transação, executam os contratos relacionados a ela.
- Podemos imaginar o estrago de um programa que não para.

Implicações da Turing-completude

- Solução: mecanismo denominado *gas.*
- Quando um código é executado, cada instrução “gasta” uma determinada quantidade de *gas*.
- Antes de um contrato ser executado, é determinado qual é o limite máximo de *gas* a ser gasto na execução.
- Para requisitar a execução de um contrato, é necessário comprar *gas* utilizando *ether*.

DApp

- DApp (Descentralized Application):
  - Aplicação web construída sobre serviços peer-to-peer descentralizados.
  - Uma DApp é composta por um front-end web e um contrato inteligente em uma blockchain.
  - Também pode incluir um mecanismo de armazenamento descentralizado (P2P) e um mecanismo de troca de mensagens.

Distribuição e descentralização

- O Ethereum é uma plataforma de computação distribuída.
- Quanto à descentralização:
  - Sob a perspectiva de arquitetura, o Ethereum é sistema descentralizado.
  - Sob a perspectiva da lógica de execução, o Ethereum trabalha como um único computador.

Exercício

- Baseado no que vimos sobre o Bitcoin, faça **suposições** sobre o funcionamento do Ethereum com relação aos seguintes pontos:
  - Como um usuário cria um programa e o implanta na blockchain?
  - Como um usuário pode disparar um programa armazenado na blockchain?
  - Quando um usuário requisita a execução de um programa na blockchain, quem realiza esta execução?
  - Como podemos validar a execução de um programa?

Ethereum – Fundamentos Básicos

UEL

Introdução

- Qual é a moeda básica do Ethereum?
- O que é uma carteira?
- O que é a EVM (Ethereum Virtual Machine)?
- O que é a EOA (External Owned Account)?
- Quais são os tipos de cliente Ethereum?
- Quais técnicas de criptografia o Ethereum utiliza?
- Como é alcançado o consenso no Ethereum?

Moeda

Value (in wei)	Exponent	Common name	SI name
1	1	wei	Wei
1,000	10^3	Babbage	Kilowei or femtoether
1,000,000	10^6	Lovelace	Megawei or picoether
1,000,000,000	10^9	Shannon	Gigawei or nanoether
1,000,000,000,000	10^{12}	Szabo	Microether or micro
1,000,000,000,000,000	10^{15}	Finney	Milliether or milli
1,000,000,000,000,000,000	*10^{18}*	*Ether*	*Ether*
1,000,000,000,000,000,000,000	10^{21}	Grand	Kiloether
1,000,000,000,000,000,000,000,000	10^{24}		Megaether

Fonte: Antonopoulos, A.M. e Wood, G., “Mastering Ethereum”, O’Reilly, 2018.

UEL

Carteira (*wallet*)

- É uma aplicação que permite a um usuário gerenciar a sua conta no Ethereum.
- Permite:
  - Gerenciar chaves.
  - Criar e disseminar transações.
- MetaMask: extensão para navegadores que permite se conectar a diferentes redes baseadas em Ethereum. Muito boa para testes.

Carteira (*wallet*)

- Uma conta possui um par de chave privada e chave pública.
- Deve-se tomar muito cuidado ao armazenar a chave privada:
  - Com posse dela, qualquer um pode manipular todos os seus fundos.
- Interessante: carteira possui chaves, não fundos. Os fundos estão gravados na blockchain e podem ser manipulados por meio da posse das chaves.

Carteira (*wallet*)

- Recurso para recuperação de carteira:
  - Chaves podem ser geradas a partir de uma semente.
  - Sementes são codificadas em *mnemonic code words* (uma sequência de palavras).
  - Essas carteiras são conhecidas como determinísticas.

Ethereum como um computador

- Na verdade, a ideia do *ether* é utilizá-lo para pagar a execução de programas.
- A criptomoeda no Ethereum é só parte do seu propósito de trabalhar como um computador descentralizado.
- Programas são executados pela Ethereum Virtual Machine (EVM).
- EVM opera como se fosse um computador global:
  - Cada nó tem uma cópia local da EVM para validar as execuções.
  - A blockchain guarda os estados dos programas.

EOAs e contratos

- Externally Owned Account (EOA): são contas que tem uma chave privada, proporcionando controle de fundos e contratos.
- Um contrato não tem uma chave privada.
- Contratos e EOAs têm endereços.
- Contratos e EOAs podem receber dinheiro.
- Quando o destino de uma transação é um contrato, isso causa a execução dele.
- Um contrato não pode iniciar uma transação.

Exemplo de código em Solidity

```
// Version of Solidity compiler this program was written for
pragma solidity 0.6.4;

// Our first contract is a faucet!
contract Faucet {
    // Accept any incoming amount
    receive() external payable {}

    // Give out ether to anyone who asks
    function withdraw(uint withdraw_amount) public {
        // Limit withdrawal amount
        require(withdraw_amount <= 10000000000000000);

        // Send the amount to the address that requested it
        msg.sender.transfer(withdraw_amount);
    }
}
```

Implementação e utilização do contrato

- Agora vamos conferir como podemos usar uma certeira e um IDE para criar, implantar e executar um contrato em uma blockchain Ethereum.

Clientes Ethereum

- Um cliente Ethereum é um software que implementa as especificações do Ethereum e se comunica por meio de uma rede P2P com outros clientes.
- Uma especificação denominada *Yellow Paper* define o protocolo do Ethereum.
- Temos diferentes clientes e diferentes redes Ethereum:
  - Apesar de todos alegarem seguir os protocolos, nem sempre é possível utilizar qualquer cliente em qualquer rede.

Clientes Ethereum

- *Full node*: cliente que armazena uma cópia completa da blockchain da rede onde está conectado. No caso da *mainnet*, a blockchain completa pode ter algumas centenas de GB.
- *Light client*: semelhante ao SPV do Bitcoin, valida cabeçalhos de blocos e verifica a inclusão de transações.
- *Remote client*: é o mesmo que carteira. Se conecta em um full node e é capaz de criar e disseminar transações (mas não as valida).

Redes Ethereum

- *mainnet:*
  - Lida com ”dinheiro real”.
  - Em fevereiro de 2024, blockchain toda tem mais de 1TB.
- *testnet*:
  - Menos dados para sincronizar que a *mainnet.*
  - Podemos trabalhar com contratos e transações obtendo fundos de faucets.
  - São blockchains públicas com muitos contratos disponíveis e várias opções de experimentação.
  - Não lida com “dinheiro de verdade”. O *gas*, por exemplo, é gratuito.

Redes Ethereum

- Rede simulada (Ganache):
  - Ocupa pouquíssimo espaço em disco.
  - Não demanda interação com faucets para obter fundos.
  - Não tem outros contratos, apenas aqueles que você implementar.
  - Cenários de testes não são tão diversos como em uma blockchain pública.
- É possível desenvolver utilizando as *testnets* e redes simuladas.

Ethereum e a criptografia

- Chaves privadas das EOAs:
  - Comprimento de 256 bits.
  - Nunca são transmitidas ou armazenadas na blockchain.
  - Valor gerado com a utilização de PRNG.
  - Devem ser guardadas com muito cuidado.
- Chaves públicas das EOAs:
  - Derivadas a partir das chaves privadas.
  - Comprimento de 64 bytes.
- Fundos de EOAs só são transferidos por meio da assinatura digital.

Ethereum e a criptografia

- Algoritmo de criptografia de chave pública:
  - ECC (Criptografia de Curvas Elípticas).
  - Padrão secp256k1 (mesmo do Bitcoin).
- Algoritmo de hash:
  - Keccak-256 (base da especificação do padrão SHA3 do NIST)
- Assinatura digital: ECDSA (como no Bitcoin)

Ethereum e a criptografia

- Endereço das contas corresponde aos últimos 20 bytes do resultado da aplicação do Keccac-256 sobre a chave pública da conta.
- Contratos não tem chaves privadas.

Mecanismo de consenso

- O Ethereum utilizou PoW em sua *mainnet* até Setembro de 2022.
- Ocorreu então uma mudança onde uma rede que era mantida por Proof of Stake foi combinada à *mainnet: The Merge*.
- A partir deste evento (*The Merge*) o Ethereum passou a usar o Proof of Stake.
- https://ethereum.org/en/upgrades/merge/

Proof of Stake

- Validador:
  - Precisa disponibilizar/bloquear 32 ETH.
  - Depois de disponibilizar esse valor, entra em uma fila de ativação. Ela é usada para controlar a quantidade de validadores na rede.
  - Quando o validador é finalmente ativado, ele passa a receber blocos para realizar a validação.
  - Quando o bloco é considerado válido, o validador envia seu voto (*attestation*) para a rede.

Proof of Stake

- O tempo é organizado em *slots* e *epochs*.
- Cada slot tem 12 segundos e cada epoch tem 32 slots.
  - Ou seja, não temos mais o tempo sendo controlado pela dificuldade da PoW.
- Um validador é aleatoriamente escolhido em cada slot para propor um novo bloco.
- Em cada slot, um comitê de validadores é escolhido para validar o bloco criado.
- O início de cada *epoch* é um checkpoint, quando os blocos são confirmados caso tenham 2/3 de aprovações dentro do comitê.

Proof of Stake

- Validadores que se comportam mal são punidos com a perda de alguns ethers que estavam bloqueados.
- Validadores que se comportam bem vão recebendo recompensas.
- Mais detalhes em: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/

Ethereum – Transações

Introdução

- Quais campos compõem uma transação no Ethereum?
- Como uma transação é disseminada no Ethereum após a sua criação?

Transações

- No Ethereum, uma transação é uma mensagem assinada pela chave privada de uma EOA.
- As transações, após serem mineradas, são todas gravadas em blocos.
- As transações são responsáveis por registrar as mudanças de estado na grande máquina de estados que é o Ethereum.

Estrutura da transação

- Uma transação contém os seguintes dados:
  - Nonce
  - Gas price
  - Gas limit
  - Recipiente
  - Valor
  - Dados
  - *v, r, s* (componentes da assinatura digital)
- Vamos ver exemplos de transações no Etherscan: https://etherscan.io

Estrutura da transação

- Nonce:
  - Um valor escalar que representa o número de transações já enviadas por aquele endereço.
  - É utilizado para ordenar o processamento das transações de uma dada conta.
    - A ordem de processamento das transações importa porque elas podem representar chamadas a programas.
  - Ajuda a evitar o processamento de transações duplicadas.

Estrutura da transação

- Gas:
  - É cotado em ethers.
  - A cotação pode variar para evitar que o custo das transações sofra com a flutuação do ether.
- Gas price:
  - Determina a cotação que o emissor da transação pagará para o custo de processamento da transação.
  - Se define um gas price baixo, pode ser mais difícil para encontrar mineradores.

Estrutura da transação

- Gas limit:
  - máximo que emissor irá pagar pelo processamento da transação.
  - define uma condição de parada no processamento de contratos.
  - Transações que envolvem apenas transferência de fundos têm preço fixo de 21000 gas units.

Estrutura da transação

- Recipiente:
  - EOA ou contrato.
  - O campo recipiente contém o endereço público de 20 bytes.
  - O endereço não é validado. Se for inexistente, o fundo transferido é “queimado”, ou seja, jogado fora.

Estrutura da transação

- Valor:
  - Valor de fundos transferidos de acordo com a transação.
  - Pode ser deixado em branco.
- Dados:
  - Usuário pode incluir qualquer informação neste campo.
  - Invocações de funções de contratos são colocadas neste campo.

Estrutura da transação

- Uma transação que contém um valor é um pagamento.
- Se o recipiente da transação é um EOA, o saldo desta conta será atualizado com a adição deste valor.

Estrutura da transação

- Se o recipiente é um contrato:
  - A EVM executa o contrato e busca invocações de função no campo de dados.
  - Se não há qualquer função invocada, a EVM vai chamar uma função *fallback* do contrato. Se ela for *payable*, o valor vai ser manipulado de acordo com esta função. Se a função *fallback* não tem conteúdo, o valor é adicionado ao saldo do contrato.
  - Se não há função *fallback* ou ela não é *payable*, a transação é revertida.

Estrutura da transação

- Transação especial de criação de contrato:
  - O recipiente é preenchido com um endereço especial: `0x0`.
  - O campo ”dados” é preenchido com o *bytecode* compilado do contrato.
  - Se o campo valor for preenchido, estes fundos serão adicionados ao saldo do contrato quando criado.
  - Quando minerada, esta transação permitirá a gravação do código fonte do contrato na blockchain.

Estrutura da transação

- Passo a passo da assinatura digital:
  - Criar uma estrutura de dados com os campos nonce, gasPrice, gasLimit, to, value, dados, e chainID.
  - Produzir uma mensagem serializada a partir desta estrutura.
  - Computar o Keccak-256 da mensagem serializada.
  - Computar a assinatura digital, assinando o hash produzido no passo anterior com a chave privada da EOA.
  - Anexar a assinatura (valores *v*, *r* e *s*) à transação.

Disseminação da transação

- Após a criação, a transação é disseminada em um esquema denominado *flood routing:*
  - Nós da rede recebem a transação, validam, armazenam e repassam para os outros nós conectados a eles.
- Para ser gravada em um bloco, ela deve ser selecionada por um minerador e incluída no bloco que ele publicar.

Ethereum – Solidity

UEL

Introdução

- Qual é o ciclo de vida de um contrato no Ethereum?
- Quais são os tipos possíveis para as variáveis em Solidity?
- Quais são as funções e variáveis pré-definidas?
- Como podemos construir as nossas próprias funções?

Contrato Inteligente

- Definição mais formal de Nick Szabo:
  - "*a set of promises, specified in digital form, including protocols within which the parties perform on the other promises*"
- No Ethereum, temos uma definição mais aberta:
  - Necessariamente, não há elementos inteligentes ou obrigações legais.
  - Programas **imutáveis** que são executados de maneira **determinística** no ambiente descentralizado do Ethereum.

Ciclo de vida do contrato

1. Contrato é escrito em linguagem de alto nível (Solidity)
2. Contrato é compilado em código de baixo nível (bytecode).
3. Contrato compilado é implantado na blockchain por meio de uma transação especial.
4. É atribuído um endereço ao contrato:
   - Envio de fundos ao contrato.
   - Invocação de funções do contrato.
   - Contrato não tem chaves privadas ou públicas.

Ciclo de vida do contrato

- Contratos só são executados quando invocados por meio de uma transação ou mensagem.
- Transações são atômicas e podem ser confirmadas ou revertidas.
- Modificações no estado realizadas a partir de transações revertidas são também todas revertidas.
- Um contrato não pode ser modificado depois de implantado.
- Um contrato pode ser apagado:
  - Possível apenas quando o dono previu esta hipótese na programação do contrato.

Solidity

- Linguagem de alto nível para programação de contratos.
- Lembra um pouco Java e C++.
- Existem outras linguagens, mas a Solidity é a mais popular.
- Desenvolvida e mantida como um projeto independente no GitHub (https://github.com/ethereum/solidity)

Solidity

- Versão da linguagem denotada por
  - MAJOR.MINOR.PATCH
  - Versão atual, por exemplo, é 0.8.17
- IDE mais utilizada: Remix.

Tipos de dados em Solidity

Boolean	Declarado como bool, tem os valores true ou false
Integer	Declarado em incrementos de 8 bits (`int8` até `uint256`). Se declarado só como `int` ou `uint`, assume tamanho de 256 bits.
Fixed point	Valores com ponto fixo declarados como (u)fixedMxN, onde M é o tamanho em bits (8 até 256) e N é o número de decimais depois do ponto (máx 18). Exemplo: `ufixed32x2`.
Address	Objeto que representa endereço de 20 bytes e tem funções úteis como `balance` e `transfer`
Byte array (fixed)	Vetor de bytes que varia de 1 a 32 bytes (fixo). Exemplo: `bytes1`, `bytes32`
Byte array (dynamic)	Vetores dinâmicos de bytes ou string
Enum	Tipos que definem faixas de valores discretos: `enum NAME {LABEL1, LABEL2, …}`
Arrays	Vetor de qualquer tamanho, fixo ou dinâmico: `uint32[][5]`
Struct	Containers criados para agrupar variáveis: `struct NAME{TYPE1 VARIABLE 1; TYPE2 VARIABLE2;…}`
Mapping	Tabelas no formato chave:valor: `mapping(KEY_TYPE => VALUE_TYPE) NAME`
Time units	Seconds, minutes, hours, e days podem ser usados como sufixos, sendo convertidos para a base de segundos
Ether units	Wei, finney, szabo, e ether podem ser usados como sufixos, convertidos para a base de wei.

UEL

Variáveis e funções pré-definidas

`msg.sender`	Endereço que originou a mensagem (EOA ou outro contrato)
`msg.value`	Valor dos fundos enviados com a chamada (em wei)
`msg.gas`	Quantidade de gas ainda disponível para execução desta chamada
`msg.data`	Dados preenchidos no campo de dados da chamada
`msg.sig`	4 primeiros bytes dos dados da mensagem (indica função a ser invocada)
`tx.gasprice`	Cotação do gás para esta transação
`tx.origin`	Endereço da EOA que originou a transação (Cuidado, função insegura)
`block.blockhash(_blocknumber_)`	Retorna o hash do bloco cujo numero foi passado como parâmetro
`block.coinbase`	Endereço do recipiente da recompensa paga neste bloco
`block.difficulty`	Dificuldade da PoW deste bloco
`block.gaslimit`	Máximo de gas gasto em todas as transações do bloco.
`block.number`	Número do bloco, também conhecido como altura do bloco
`block.timestamp`	Data e horário determinados pelo minerador no bloco

UEL

Variáveis e funções pré-definidas

`address.balance`	Retorna o saldo da conta neste endereço em wei
`address.transfer(_amount_)`	Transfere os fundos expressos em _amount_ para este endereço
`address.send(_amount_)`	Similar ao transfer. Diferença é que retorna falso ao invés de lançar exceção em caso de erros. Por isso, é importante sempre checar o retorno.
`address.call(_payload_)`	Chamada de baixo nível que invoca a função presente em _payload_. É uma função insegura.
`address.delegatecall(_payload_)`	Também é uma função de baixo nível que demanda conhecimento avançado para uso.

Variáveis e funções pré-definidas

`ecrecover`	Recupera o endereço usado para assinar a mensagem
`keccak256, sha256, sha3, ripemd160`	Calcula função hash usando o respectivo algoritmo
`selfdestruct(_recipient_address_)`	Apaga o contrato, enviando os fundos remanescentes para o endereço recipiente.
`this`	O endereço do contrato sendo executado

UEL

Construindo funções

```
function FunctionName([parameters])
{public|private|internal|external}
[pure|view|payable] [modifiers] [returns
(return types)]
```

- FunctionName:
  - Nome que será usado para invocar a função.
  - Todo contrato pode ter uma função *fallback* ou uma função *receive*, definidas com as palavras reservadas `fallback` ou `receive`.

Construindo funções

- *public*: é o padrão. Estas funções podem ser chamadas por outras EOAs ou contratos.
- *external*: semelhante a public, mas não pode ser chamada de dentro do próprio contrato, a menos que a palavra reservada *this* seja usada.
- *internal*: só são acessíveis de dentro do contrato ou por contratos filhos (herança)
- *private*: semelhante a *internal*, mas não pode ser invocada por contratos filhos.

Construindo funções

- *constant ou view:* funções que não modificam estado.
- *pure:* não lê ou escreve nenhuma variável no espaço de armazenamento. Só manipula os argumentos e retorna dados.
- *payable:* função que pode aceitar pagamentos.

Construtor e `selfdestruct`

- Construtor:
  - Função opcional.
  - Executada apenas uma vez, quando o contrato é criado.
  - Inicializa estado do contrato.
  - Pode ser definida tendo o mesmo nome do contrato (propenso a erros).
  - Alternativa é o uso da palavra reservada `constructor` em uma função sem nome: `constructor() { ... }`

Construtor e `selfdestruct`

- Para destruir o contrato:
  - É necessário adicionar função que invoque `selfdestruct(address recipiente);`
  - Exemplo de função para destruir contrato:

```
function destroy() public {
    require (msg.sender == owner);
    selfdestruct(owner);
}
```

Modificadores de funções

- O modificador é um tipo de função especial que podemos declarar para depois aplicar em outras funções. Exemplo:

```
modifier onlyOwner{
     require(msg.sender == owner);
     _; //placeholder
}

function destroy() public onlyOwner{
     selfdestruct(owner);
}
```

Herança

- O Solidity suporta herança:

```
contract Child is Parent {
    ...
}

contract Child is Parent1, Parent2{
    ...
}
```

Tratamento de exceções

- Função require:
  - Testa se a entrada da função é falsa ou verdadeira.
  - Quando o teste retorna falso, uma exceção é lançada e o código não é mais executado.

```
require(msg.sender == owner);

require(msg.sender == owner, “Only the
contract owner can call this function”);
```

Chamando outros contratos

- Pode ser perigoso, pois nem sempre conhecemos completamente o funcionamento deste outro contrato.
- Maneira mais segura:
  - Você mesmo cria uma nova instância do contrato:

```
import “Faucet.sol”
contract Token is Mortal{
    Faucet _faucet;
    constructor(){
            _faucet = new Faucet();
    }
    function destroy() ownerOnly{
            _faucet.destroy();
    }
}
```

Chamando outros contratos

- Podemos também fazer um *casting* de um objeto retornado a partir do endereço do contrato.

```
import “Faucet.sol”
contract Token is Mortal{
   Faucet _faucet;
   constructor(address _f){
        _faucet = Faucet(_f);
        _faucet.withdraw(0.1 ether);
   }
}
```

Chamando outros contratos

- Podemos ainda usar a função de “baixo-nível”. Opção mais flexível e mais arriscada:

```
import “Faucet.sol”
contract Token is Mortal{
   constructor(address _faucet){
        if !(_faucet.call(“withdraw”, 0.1 ether)){
             revert(“Withdraw from faucet failed”);
        }
   }
}
```

Ethereum – Tokens

Introdução

- O que são tokens?
- O que é fungibilidade?
- Como podemos usar o padrão ERC20 para desenvolver os nossos tokens no Ethereum?
- Existe um padrão para NFTs no ambiente Ethereum?

Tokens

- Na vida real, tokens são objetos que tem um valor dentro de um dado contexto:
  - Pense nas fichas de fliperama, de jogos de sinuca ou das quermesses/festas juninas.
- Na blockchain, tokens definem uma abstração que pode representar uma moeda, ativos ou direito de acesso a um determinado sistema.
  - Tokens na blockchain também denotam propriedade/direito.

Como os tokens são utilizados?

- Moeda: uma moeda que pertence a uma aplicação ou contexto particular.
- Recurso: um token pode representar o direito de acesso/uso a um recurso compartilhado (armazenamento em disco, horas de CPU, etc.).
- Acesso: token pode representar direito de acesso a ativo físico (e.g., quarto de hotel, carro alugado) ou virtual (e.g., site exclusivo, fórum de discussão)

Como os tokens são utilizados?

- Ativo: pode representar a propriedade de diferentes ativos como ouro, uma propriedade, um carro, energia, etc.
- Ação em companhias: um token pode representar a participação em uma organização digital ou uma empresa comum.
- Colecionável: um token pode representar um item digital (e.g., um meme ou um gif) ou físico (um quadro de Van Gogh).

Como os tokens são utilizados?

- Identificação: um token pode representar um documento de identidade com valor legal (um RG) ou uma identidade virtual (um avatar).
- Certidão: representa uma certidão emitida por órgão oficial (e.g., certidão de casamento, diploma) ou baseada em reputação mediante um sistema descentralizado.

Fungibilidade

- Tokens fungíveis podem ser trocados uns pelos outros, já que cada token não é único. Exemplo: ficha de festa junina.
- Tokens não fungíveis: cada token tem características únicas, não podendo ser substituído por outro token.

Ponto a observar

- Quando o token representa a propriedade de um ativo físico que depende de uma terceira parte, este risco deve ser levado em conta.
  - É importante envolver esta terceira parte no processo de alguma forma.
  - É importante notar que, em alguns casos, a blockchain vai facilitar partes do processo, mas não automatizá-lo por completo.

Tokens no Ethereum

- Tokens são diferentes de ether:
  - Ether é parte intrínseca da rede.
  - Transações com ether e saldo de ether são controlados no nível de protocolo.
  - Transações com tokens e saldo de tokens são controlados no nível de contrato inteligente.
- É importante seguir algumas especificações no desenvolvimento deste contrato inteligente.

ERC20

- ERC20 é um padrão para desenvolvimento de contratos que implementam tokens.
- Prevê uma série de funções que precisam ser desenvolvidas.

ERC20

totalSupply	Retorna a quantidade total de tokens existentes
balanceOf	Retorna a quantidade de tokens de um determinado endereço
transfer	Transfere uma determinada quantidade de tokens do endereço que chamou a função para um endereço passado como destinatário
transferFrom	Passados uma origem e um recipiente, transfere tokens entre eles
approve	Usado em conjunto com transferFrom para transferir tokens entre dois endereços específicos
allowance	Retorna quantos tokens ainda podem ser transferidos de acordo com uma aprovação dada anteriormente
Transfer	Um evento disparado após uma transferência concluída com sucesso
Approval	Um evento disparado para denotar sucesso em uma solicitação de aprovação
name	Retorna o nome do token em formato amigável (ex., reais brasileiros)
symbol	Retorna o símbolo do token em formato amigável (ex., R$ ou BRL)
decimals	Número de decimais utilizados para dividir os tokens. Por exemplo, se o valor deste parâmetro é 2, o total dividido por 100 é usado para representar um valor.

*funções em vermelho são opcionais quando o contrato é implementado

UEL

ERC20

```
contract ERC20 {
   function totalSupply() constant returns (uint theTotalSupply);
   function balanceOf(address _owner) constant returns (uint balance);
   function transfer(address _to, uint _value) returns (bool success);
   function transferFrom(address _from, address _to, uint _value) returns
      (bool success);
   function approve(address _spender, uint _value) returns (bool success);
   function allowance(address _owner, address _spender) constant returns
      (uint remaining);
   event Transfer(address indexed _from, address indexed _to, uint _value);
   event Approval(address indexed _owner, address indexed _spender, uint _value);
}
```


UEL

ERC20

- Para evitar problemas de segurança, existem bibliotecas que podem ser usadas para implementar estes tokens, como é o caso do OpenZeppelin.
- Vamos fazer um tutorial para criar o nosso próprio token.

ERC721

- ERC721 é um padrão para criação de NFTs (non fungible tokens).
- Em NFTs, cada token é unicamente identificado e possui um proprietário.
- O token pode se referir a um item digital, como um meme, um gif ou um item em um jogo.
- O token pode se referir a um item físico, como um carro, uma casa, ou uma obra de arte.

ERC721

- Exigência do padrão é que a “coisa” representada pelo token seja unicamente identificada por um número de 256 bits.
- Há uma relação de propriedade. Um determinado endereço público é dono de um item representado por um código de 256 bits.

ERC721

```
// Mapping from deed ID to owner
mapping (uint256 => address) private deedOwner;
```

```
interface ERC721 /* is ERC165 */ {
    event Transfer(address indexed _from, address indexed _to, uint256 _deedId);
    event Approval(address indexed _owner, address indexed _approved,
                   uint256 _deedId);
    event ApprovalForAll(address indexed _owner, address indexed _operator,
                         bool _approved);

    function balanceOf(address _owner) external view returns (uint256 _balance);
    function ownerOf(uint256 _deedId) external view returns (address _owner);
    function transfer(address _to, uint256 _deedId) external payable;
    function transferFrom(address _from, address _to, uint256 _deedId)
        external payable;
    function approve(address _approved, uint256 _deedId) external payable;
    function setApprovalForAll(address _operator, boolean _approved) payable;
    function supportsInterface(bytes4 interfaceID) external view returns (bool);
}
```


UEL

Importância dos padrões

- Os padrões são uma especificação mínima para tokens.
- A ideia deles é promover a interoperabilidade entre contratos:
  - Facilita, por exemplo, que se desenvolva carteiras genéricas que podem ser usadas para diferentes tokens.
- Permite ainda usar implementações bem conhecidas (e.g., OpenZeppelin) como base.

DApps

Introdução

- O que é a Web3?
- O que são DApps?
- Como implementamos um DApp?

Web3

- Criadores do Ethereum sempre vislumbraram uma nova Web:
  - Funções de pagamentos e regras de negócio das aplicações seriam descentralizadas na forma de contratos inteligentes.
  - Outros aspectos como armazenamento, troca de mensagens, e resolução de nomes também poderiam ser descentralizados.
  - Web dos DApps, denominada Web3.

Web3

- Na prática, atualmente:
  - DApps são aplicações com *front-end* web e *back-end* implementado como contrato inteligente.

DApp

- Definição: aplicação totalmente ou quase totalmente descentralizada.
- O que pode ser descentralizado:
  - Lógica da aplicação (*back-end*)
  - *Front-end*
  - Armazenamento de dados
  - Troca de mensagens
  - Resolução de nomes

DApp

- Vantagens:
  - Resiliência
  - Transparência
  - Resistência a censura

DApp

- Back-end:
  - Baseado principalmente em contratos inteligentes (Solidity e outras linguagens relacionadas).
  - Também pode usar algumas fontes de dados externas e fazer algumas computações *off chain* para economizar gas.

DApp

- Front-end:
  - Utiliza tecnologias típicas de Web como HTML, CSS e JavaScript.
  - Interação com Ethereum é intermediada por ferramentas como MetaMask.
  - Biblioteca JavaScript *web3.js*.
    - MetaMask funciona como um provedor Web3.
  - Pode ser um aplicativo para *smartphone* também.

DApp

- Armazenamento de dados:
  - Contratos inteligentes **não** são apropriados para armazenar grandes volumes de dados.
  - Utilização de soluções P2P de armazenamento descentralizado como IPFS (Inter-Planetary File System) e Swarm.

Oráculos

- A implementação de aplicações mais complexas evidencia algumas limitações da EVM e seus contratos inteligentes.
  - Dificuldade em armazenar volumes maiores de dados.
  - Funções que dependem de fontes de aleatoriedade.
- A solução é implementar algumas funções *off-chain*:
  - Oráculos.

Oráculos

- Oráculos proveem formas de obter informações externas à blockchain como:
  - Preço do ouro, resultados de jogos de futebol, números aleatórios, etc.
- Ajudam a construir aplicações mais abrangentes e diversificadas.
- Oferecem um risco de segurança.

Oráculos

- Exemplos de informações que podem ser oferecidos por oráculos:
  - Números aleatórios.
  - Informações sobre eventos como desastres naturais, que podem disparar a utilização de seguros, etc.
  - Cotação de moedas “físicas”.
  - Preços de ações e outros ativos de mercado financeiro.
  - Dados estáticos como códigos telefônicos de países.
  - Informações sobre o clima.

Oráculos

- Exemplos de informações que podem ser oferecidos por oráculos:
  - Dados de geolocalização.
  - Informações sobre danos a um patrimônio (útil para contratos de seguros).
- Importante: cuidado com a autenticação destas fontes para que elas sejam confiáveis.

Quando utilizar blockchain?

UEL

Vantagens, desvantagens e indicações

- A blockchain é indicada para situações onde:
  - Necessita-se de um registro de informações/ações que seja imutável.
  - Necessita-se de alta capacidade de resiliência.
  - Ambiente onde se queira remover qualquer tipo de controle centralizado.
  - Ambiente onde podemos ter controle centralizado, mas ele envolve diferentes partes com interesses diferentes (talvez conflitantes).

Vantagens, desvantagens e indicações

- Desvantagens:
  - O sistema de consolidação de transações pode ser lento (depende do mecanismo de consenso).
  - Não é adequado para armazenar grandes volumes de informação.
  - Ausência de controle central pode abrir espaço para outros tipos de fraudes e cibercrimes.

Referências bibliográficas

- Nakamoto S. Bitcoin: A peer-to-peer electronic cash system.(2008). Disponível em: https://bitcoin.org/bitcoin.pdf
- Antonopoulos A.M. e Wood. G. Mastering Ethereum. O'Reilly Media, 2018. Disponível em: https://github.com/ethereumbook/ethereumbook
- Mukhopadhyay, M., “Ethereum Smart Contract Development”, Packt Publishing, 2018.

UEL