

Blockchain

Sem o hype!

Prof. Dr. Marcos A. Simplicio Jr. – mjunior@larc.usp.br

Preâmbulo



NO HYPE
Técnica
3x0

- Empresas de hardware vendem blockchain como uma solução disruptiva, com diversas aplicações possíveis
 - Infelizmente, ~90% desse hype beira a desinformação...
 - ... justificada pelo grande custo em termos de hardware e energia envolvido no uso de blockchain ("yes, there is a catch...")
- A realidade:
 - Blockchain é **útil** em um cenário: **ordenação de eventos** em sistema **totalmente distribuído** e **sem confiança** entre as partes, mas há **incentivo** para sua participação no sistema
 - Blockchain é **pouco (ou nada) interessante** quando:
 - Há uma **entidade confiável** (e.g., um banco); ou
 - É **desnecessário ordenar** eventos (e.g., basta a existência); ou
 - Apenas a ordenação de eventos não é suficiente (e.g., são necessários **instantes exatos de tempo**)



Motivação: ordenação de eventos

□ Problema:

- Em uma rede distribuída, como determinar a ordem em que diversos eventos ocorreram?
 - Ex.: envio de mensagens, para determinar a sequência de uma conversa via WhatsApp (e.g., auditoria de mensagens)

A: Vem pra casa hoje?

B: Umas 19

A: Andou flirtando com alguém no trabalho? 

B: Lógico que não

A: ❤️



A: Vem pra casa hoje?

B: Lógico que não

A: Andou flirtando com alguém no trabalho? 

B: Umas 19

A: 😞

Motivação: ordenação de eventos

□ Problema:

- Em uma rede distribuída, como determinar a ordem que diversos eventos ocorreram?
 - Ex.: transações bancárias, para saber se usuário tem saldo suficiente no momento de uma compra

Saldo: A,B,C = \$0, X = \$400

A ← x: \$400

A → B: \$100

A → C: \$300

B: → A C: → A

Saldo: A,X = \$0,
B=\$100, C=\$300

Saldo: A,B,C = \$0, X = \$400

A → B: \$100

A → C: \$300

A ← x: \$400

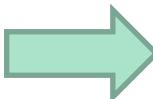
B: → A C: → A

Saldo: A = \$400, B,C,X = \$0

Motivação: ordenação de eventos

- Problema:
 - Em uma rede distribuída, como determinar a ordem que diversos eventos ocorreram?
- **Solução 1:** usuários têm um relógio sincronizado e incluem um carimbo de tempo nas mensagens

18:15 A: Vem pra casa hoje?
18:19 B: Lógico que não
18:18 A: Andou flirtando com alguém no trabalho? 
18:16 B: Umas 19



Analisando os carimbos

18:15 A: Vem pra casa hoje?
18:16 B: Umas 19
18:18 A: Andou flirtando com alguém no trabalho? 
18:19 B: Lógico que não

- Funciona...?

Motivação: ordenação de eventos

- Tente sincronizar dois relógios, com erro menor que 1 s, com duas pessoas em lados opostos de uma sala, passando apenas mensagens entre elas
 - As mensagens podem ser perdidas, atrasadas, ...
 - Uma abordagem possível: usar protocolo de rede para sincronizar sistemas (ex.: *Network Time Protocol* – NTP)
- OK, mas e se **usuários não forem confiáveis...?**



Saldo: A,B,C = \$0, X = \$400

09:00 A ← X: \$400

10:05 A → B: \$400

10:06 B: → A

"10:02" A → C: \$400

"10:03" C: → A



Saldos: A=\$0, B=\$0, C=\$400 ??

15:10 B → Z: \$400

15:10 Z: / → B

15:11 B:

15:12 B: ... → A!!!

15:13 A: tarde demais

15:13 C:

Motivação: ordenação de eventos

- Problema:

- Em uma rede distribuída, como determinar a ordem que diversos eventos ocorreram?

- **Solução 2:** Autoridade Certificadora de Tempo assina eventos (*Timestamp Authority – TSA*)



Saldo: A,B,C = \$0, X = \$400

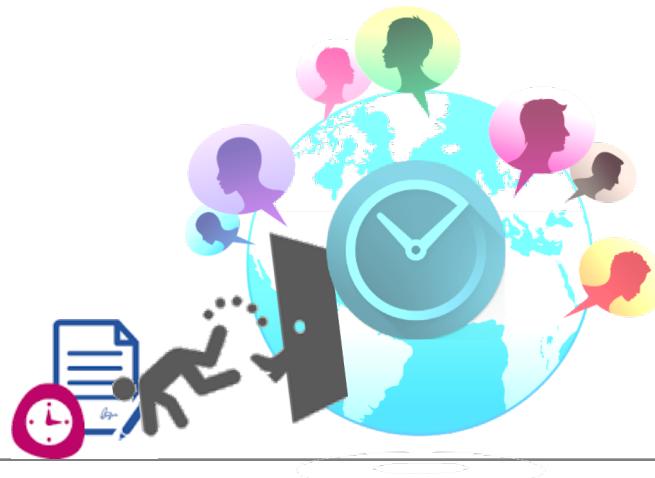
	<u>09:00</u>	A ← X: \$400
	<u>10:05</u>	A → B: \$400
	<u>10:06</u>	B: → A
	<u>"10:02"</u>	A → C: \$400
	<u>"10:03"</u>	C: → A

Saldos: A,C=\$0, B=\$400

	<u>10:10</u>	B → Z: \$400
	<u>10:11</u>	Z: → B
	<u>10:15</u>	C → Z: \$400
	<u>10:16</u>	Z: / → C
	<u>10:17</u>	C:

Motivação: ordenação de eventos

- Problema:
 - Em uma rede distribuída, como determinar a ordem que diversos eventos ocorreram?
- **TSA: simples e efetivo. Mas e se não for possível usar uma TSA no sistema?**
 - Sistema **totalmente distribuído**, sem entidades confiáveis
 - Que tal uma **TSA distribuída (e.g., um blockchain)?**



O que é um blockchain?



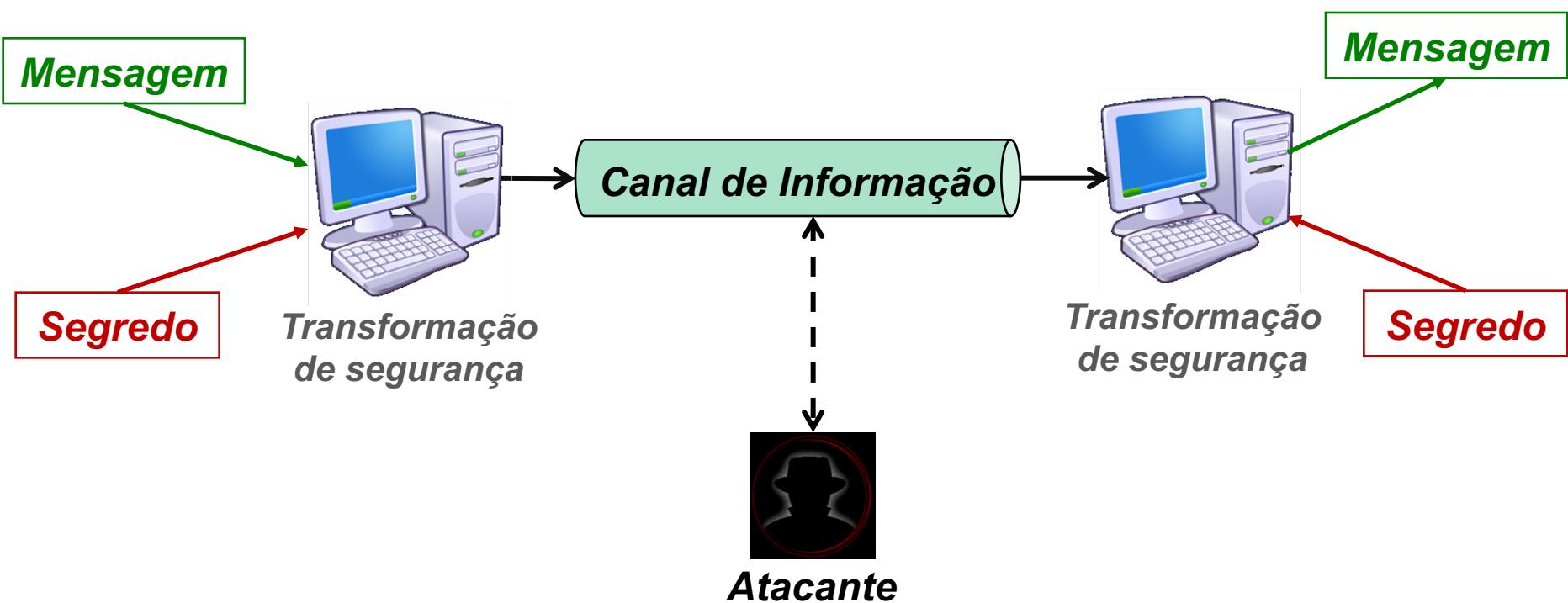


Segurança da Informação

Tudo que você precisa para entender

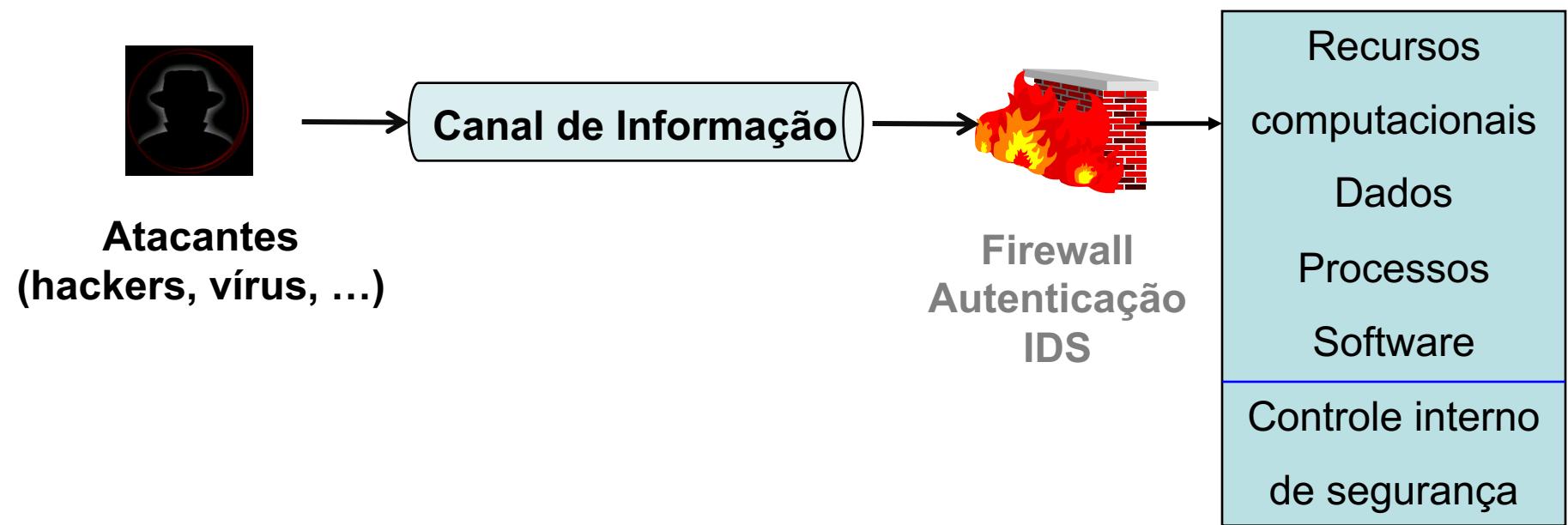
- O que é um Blockchain
- O que não é um Blockchain

Modelo para Segurança de Redes



→ Nosso foco durante o curso ←

Modelo para Segurança de Computadores



Serviços de Segurança

- Há diferentes aspectos que caracterizam a segurança de um sistema de computadores. Estes aspectos são denominados *serviços de segurança*.
- Serviços básicos de segurança:
 - Disponibilidade
 - Confidencialidade
 - Integridade
 - Autenticidade
 - Irretratabilidade



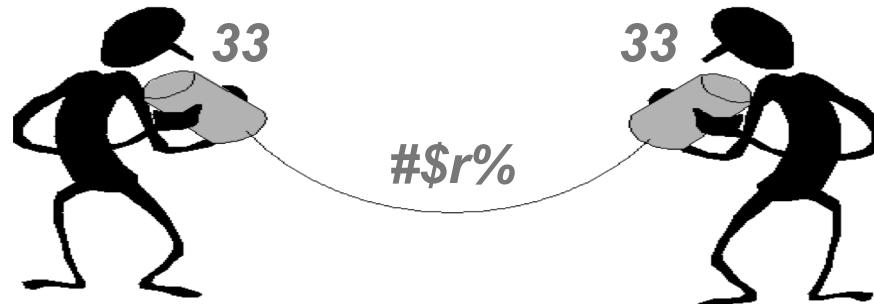
Disponibilidade

- Garantia de que usuários legítimos **não sejam impedidos** indevidamente de acessarem as informações e os recursos do sistema.
- Serviço essencialmente extra-criptográfico (físico), e o mais arquitetural/empírico/ heurístico dentre os serviços básicos da segurança.
 - Exemplos de medidas: **redundância** (amplamente usada em sistemas de blockchain), controle de acesso (físico), etc.



Confidencialidade

- **Confidencialidade de dados:** garantia de que qualquer **informação** armazenada em um sistema de computação ou transmitida via rede seja **revelada somente a usuários devidamente autorizados.**
- Observação: *informação* \neq *dado* (representação da informação).
 - Um dado pode estar acessível a qualquer entidade e mesmo assim não revelar a informação que ele contém.



Confidencialidade

- **Privacidade:** Garantia de que os indivíduos controlem ou influenciem **quais informações sobre eles** podem ser coletadas e armazenadas e **por quem e para quem** tais informações podem ser reveladas
 - Tem relação direta com **confidencialidade de dados** (proteção da informação), mas também envolve políticas de **uso de dados** e **confidencialidade de identidades**.
 - Ex. (blockchain): pseudônimos



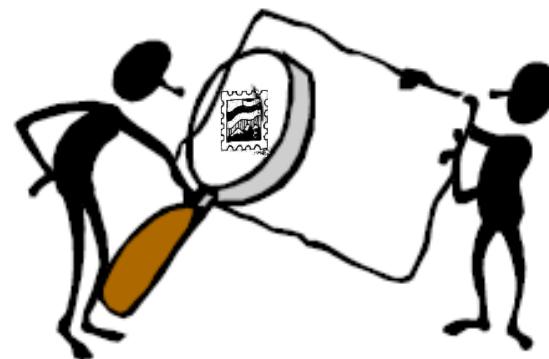
Integridade

- Possibilidade de **verificar a consistência** da informação contida nos dados, impedindo que seja **modificada indevidamente de forma imperceptível**.
- Detalhe: o serviço de integridade **não garante que os dados não sejam alterados**. A garantia efetiva é que, se os dados forem alterados sem autorização, a **alteração será sempre detectada**.



Autenticidade

- Garantia de que a origem ou **o originador** de uma mensagem seja **corretamente identificado** pelo seu destinatário.
- A verificação de autenticidade é necessária após **todo processo de identificação**, seja de um usuário para um sistema, de um sistema para o usuário ou de um sistema para outro sistema.



Irretratabilidade

- O *originador* e o *destinatário* das informações *não podem negar a sua transmissão, recepção ou posse.*
- Relacionado a *assinaturas digitais*.
 - Conceito similar a assinaturas manuais, mas com garantias matemáticas...



Autenticidade vs. Irretratabilidade

- Autenticidade: destinatário **não consegue necessariamente provar para um terceiro** quem é o originador da mensagem
- Analogia com mundo real:
 - Os usuários Alice e Bob têm um mesmo “carimbo”
 - Se Alice recebe mensagem carimbada, então ela deve ter vindo de Bob
 - Porém, Alice não consegue provar para Carlos que foi Bob quem carimbou a mensagem (afinal, a própria Alice pode tê-lo feito!)



Autenticidade vs. Irretratabilidade

- ❑ Irretratabilidade: pode-se **provar a terceiros** quem é o originador da mensagem
- ❑ Analogia com mundo real:
 - Alice envia um documento a Bob usando sua assinatura com firma reconhecida
 - Bob pode apresentar o documento a Carlos, provando que Alice foi a originadora do documento
- ❑ Diferença importante: aparece nos algoritmos usados para prover tais serviços



Serviços de segurança: ilustração

❑ Ataques passivos

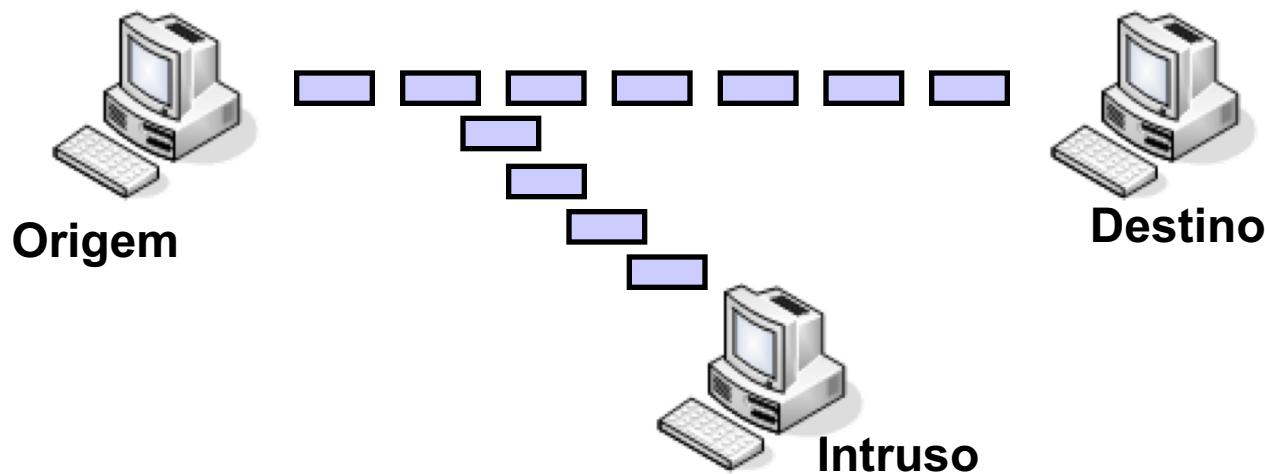
- Aqueles onde a mensagem é apenas observada ou copiada.
- Exemplo: interceptação.

❑ Ataques ativos

- A mensagem sofre alterações ou é desviada
- Exemplos:
 - Interrupção
 - Modificação
 - Fabricação

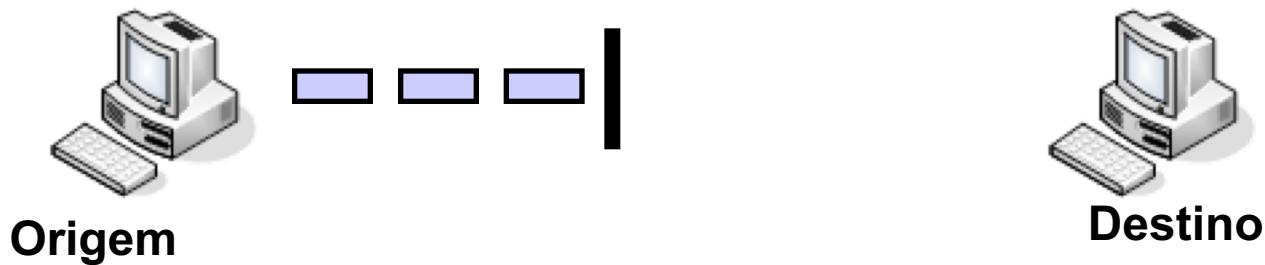
Ataques Passivos: Interceptação

- ❑ Vazamento de informações (ex.: senhas)
- ❑ Para evitar que o intruso entenda o conteúdo das mensagens, é necessário cifrar os dados (*confidencialidade*)



Ataques Ativos: Interrupção

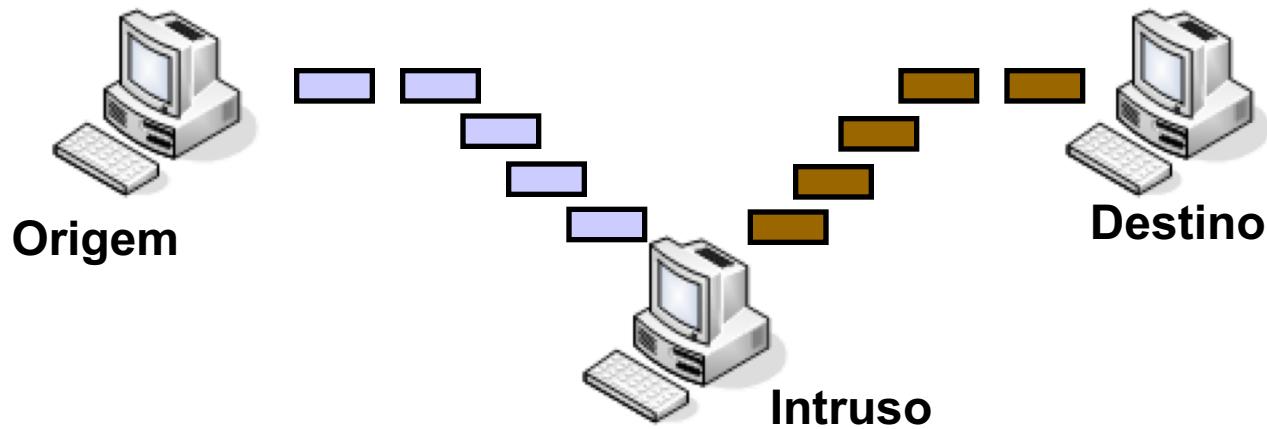
- Dados nunca chegam ao destino
 - Ex.: “derrubar um site”



- É necessária a segurança física dos recursos de processamento e de comunicação de dados!
(disponibilidade)

Ataques Ativos: Modificação

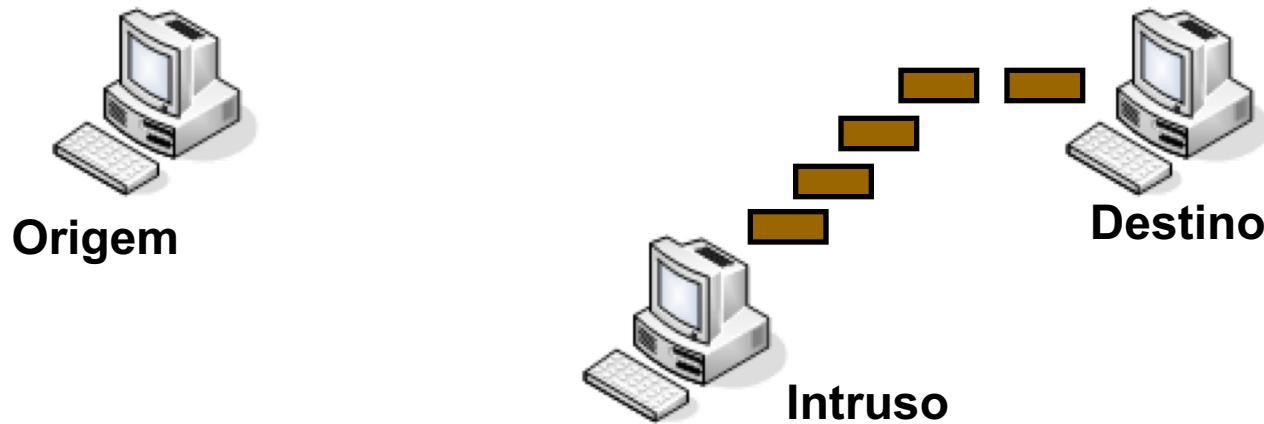
- Informações corrompidas/falsas
 - Ex.: alterar destino de um pagamento bancário



- Para evitar tal ataque, é preciso garantir a *integridade* e a *autenticidade* dos dados

Ataques Ativos: Fabricação

- ❑ Mensagens criadas por atacante
 - Ex.: gerar uma ordem de pagamento falsa



- ❑ Para evitar este tipo de ataque é preciso garantir a *autenticidade* dos dados

Outros Ataques

-  □ **Inferência:** análise de dados permitem deduzir algo sobre a informação neles contida → confidencialidade
-  □ **Exposição:** dados fornecidos diretamente a entidade não autorizada (ex.: enviados para endereço de e-mail errado) → confidencialidade
-  □ **Intrusão:** atacante burla proteções do sistema e acessa dados → confidencialidade
-  □ **Personificação:** atacante se passa por uma entidade autorizada → acessa dados (confidencialidade), altera dados (integridade) ou forja dados (autenticidade)
-  □ **Retratação:** Entidade nega falsamente a responsabilidade por um ato → irretratabilidade



Criptografia

Princípios básicos

Para que serve criptografia?

- Serviços básicos da segurança:
 - Confidencialidade,
 - Integridade,
 - Autenticidade,
 - Irretratabilidade.
- **Não** é possível implementar disponibilidade
 - Não há algoritmo criptográfico que proteja contra ataques físicos...
 - Solução costuma envolver redundâncias



Algoritmos Criptográficos

- ❑ Existem dois tipos básicos de algoritmos criptográficos
 - **Simétricos:** usam a mesma informação secreta (chave) conhecida apenas por remetente e destinatário, mas não por atacantes
 - Esta categoria também inclui algoritmos auxiliares, que não usam chaves secretas
 - **Assimétricos:** usam duas chaves distintas, relacionadas matematicamente. Uma chave é tornada pública (inclusive para atacantes), e a outra é conhecida apenas pelo seu dono.
- ❑ Implementações de algoritmos padronizados amplamente disponíveis (e.g., java.security)



Criptoanálise

- A maneira usual de avaliar a segurança de um sistema é tentar atacá-lo.
- Criptoanálise: sistematização matemática de técnicas gerais de ataque.
 - Até 1990, ataques contra algoritmos cripto-gráficos eram essencialmente *ad hoc*.
 - Conhecem-se hoje publicamente dezenas de abordagens.



Criptoanálise



- ❑ Pode-se quebrar **qualquer** sistema criptográfico baseado em chaves secretas tentando todas as chaves possíveis e verificando o resultado.
 - Isto é conhecido como ataque de força-bruta.
- ❑ Criptografia moderna se baseia em algoritmos **computacionalmente inviáveis** de se quebrar.
 - Algoritmo seguro (forte): não pode ser quebrado com recursos disponíveis atualmente ou em futuro distante.
 - A criptoanálise busca maior eficiência do que um ataque de força-bruta.
 - Mas o que são recursos disponíveis...?



Ataque de força bruta: exemplo

- Exemplo de complexidade vs. recurso:
 - Suponha que ataque a algoritmo envolva testar chaves de 128 bits: 2^{128} possibilidades
 - Suponha também que estejam disponíveis 1 milhão (10^6) de super-computadores, cada um capaz de realizar 1 peta (10^{15}) testes por segundo
 - Ainda assim seriam necessários $\sim 2^{58}$ segundos para recuperar a chave...
 - Idade estimada do universo: $\sim 2^{59}$ segundos



$$\frac{\text{#testes}}{\text{#computadores} \times \text{capacidade}} = \frac{2^{128}}{10^6 \times 10^{15}} \approx \frac{2^{128}}{2^{20} \times 2^{50}} = 2^{58} \text{ segundos}$$

Níveis de Segurança

▫ Recomendação NIST (2015)

Ano	Segurança Mínima	Algoritmos Simétricos	Assimétricos (RSA)	Assimétricos (Curvas Elípticas)	Hash (Assinaturas)
2007-2010	80	Skipjack	1024	160	SHA-1
2011-2030	112	3DES	2048	224	SHA-224
> 2030	128	AES-128	3072	256	SHA-256
>> 2030	192	AES-192	7680	384	SHA-384
>>> 2030	256	AES-256	15360	512	SHA-512

▫ Maiores informações:

- www.keylength.com/
- <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar1.pdf>

Criptoanálise: fundamentos

- O segredo deve residir **totalmente na chave**.
 - Assume-se que criptoanalista conhece completamente os detalhes do algoritmo e sua implementação.
- Considerar que um algoritmo é mais seguro porque sua estrutura interna é desconhecida é **ingenuidade**.
 - Atualmente, descobrir o algoritmo é uma questão de tempo e investimento (engenharia reversa).
- Os **melhores algoritmos** atualmente são aqueles **tornados públicos**:
 - Atacados pelos melhores criptoanalistas do mundo por anos e, mesmo assim, não foram quebrados.



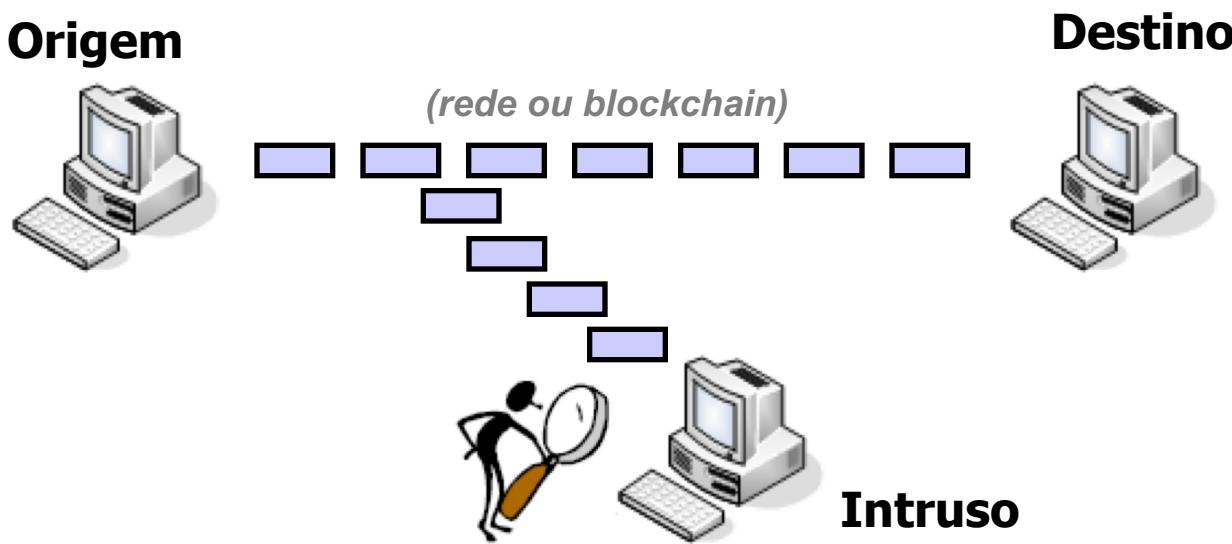


Cifras e Esteganografia

Confidencialidade de dados

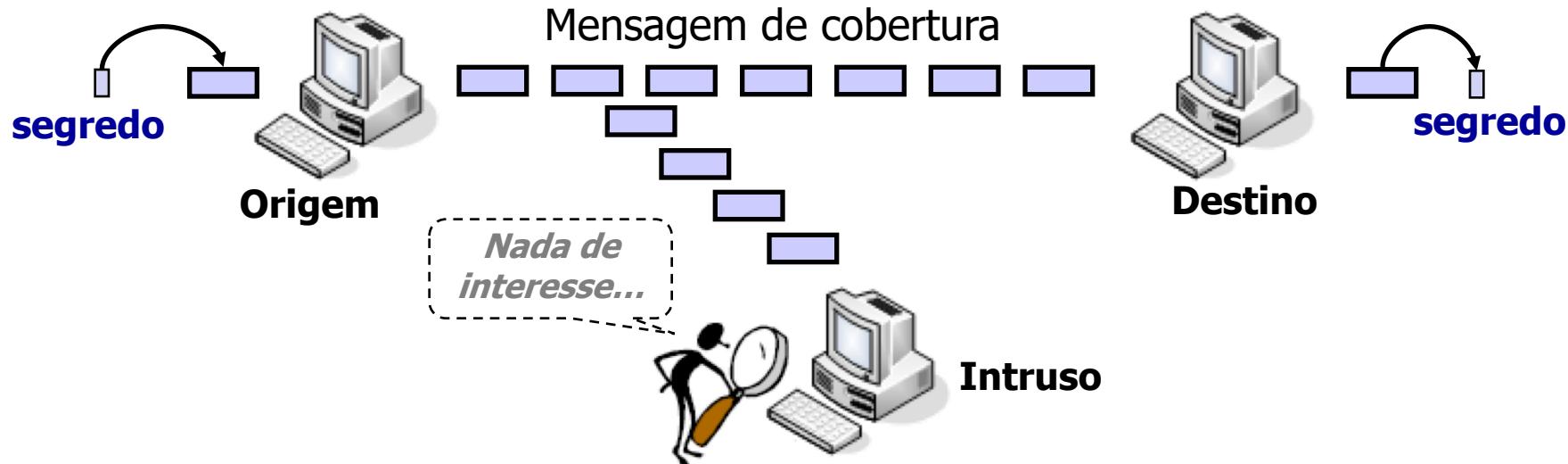
Confidencialidade

- Serviço necessário:
 - Prevenção do vazamento de informações

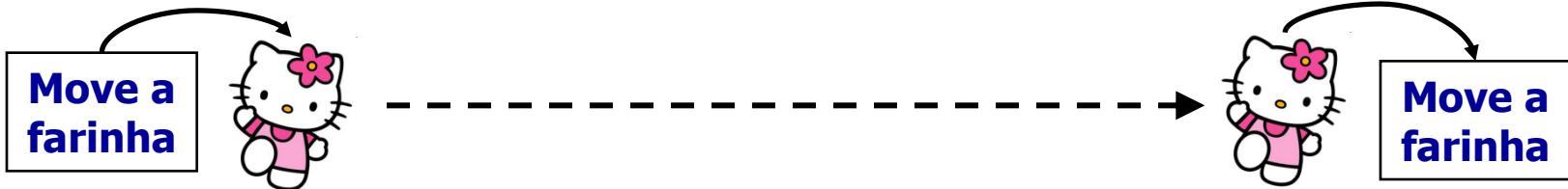


Duas abordagens possíveis: esteganografia

- (1) Disfarçar os dados: **esteganografia**

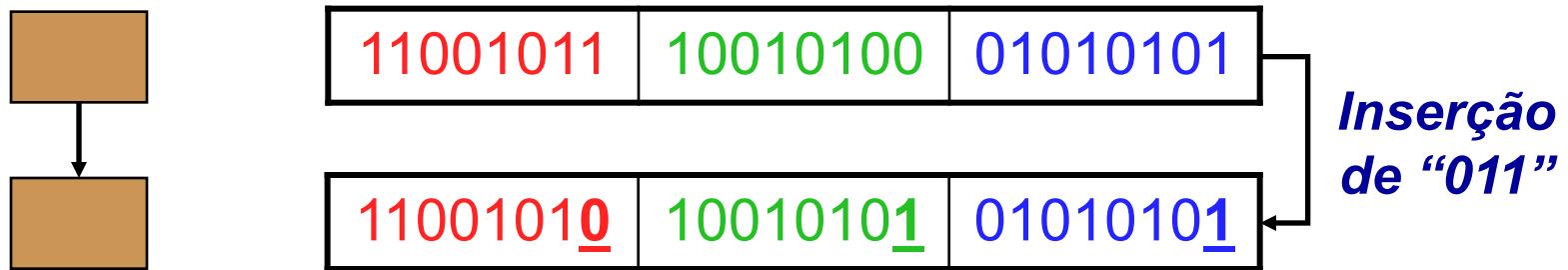


Caso Juan Carlos Abadia (2008)



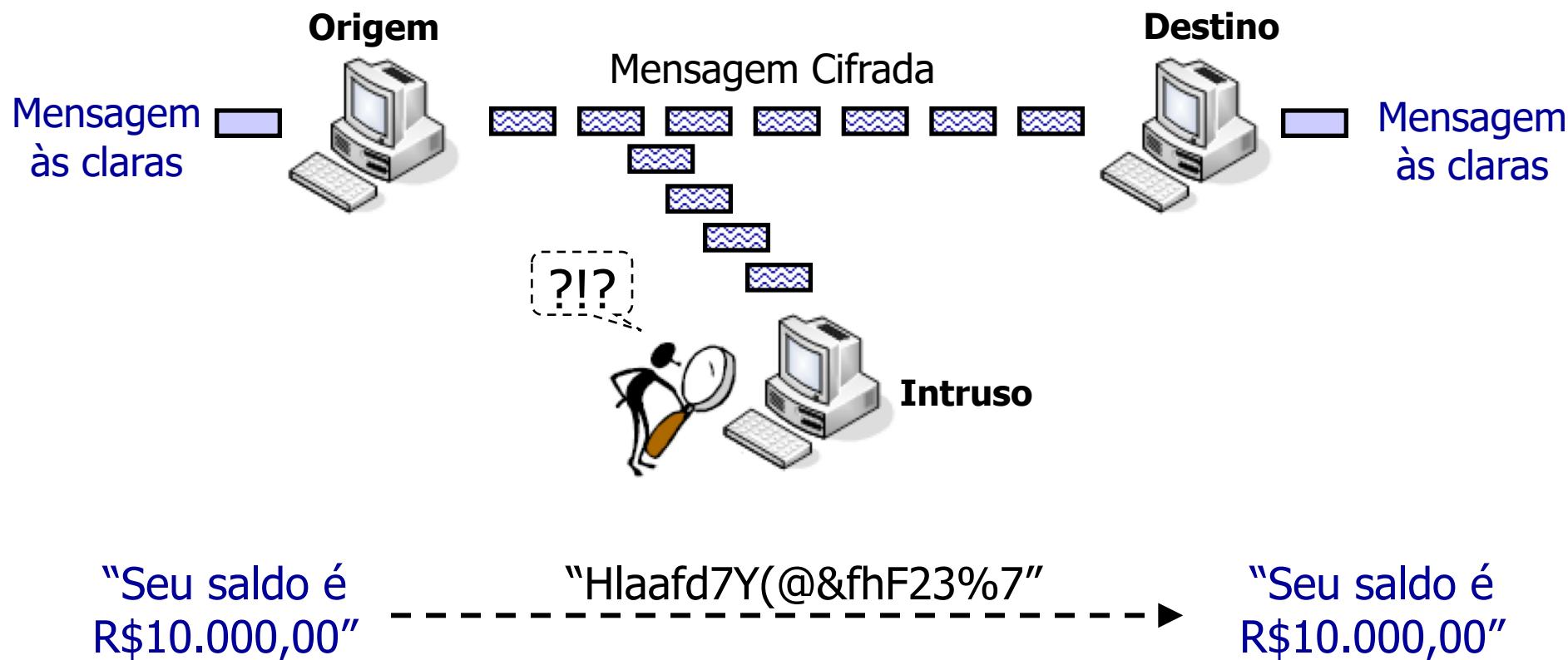
Esteganografia: exemplo

- Bit menos significativo (LSB)
 - Alteração dos pixels da imagem
 - Um dos métodos mais utilizado na área de esteganografia
 - Capacidade: segredo/cobertura $\leq 1/8$



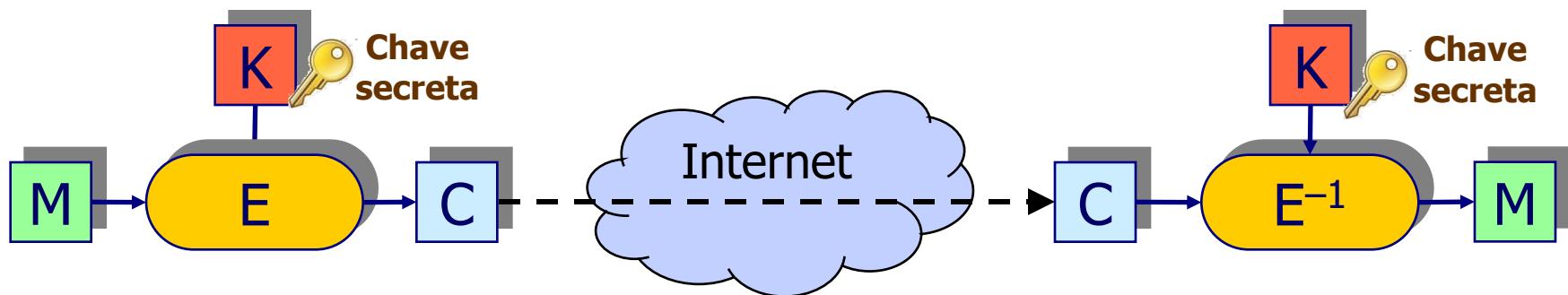
Duas abordagens possíveis: Cifras

- (2) Embaralhar os dados: **cifras**



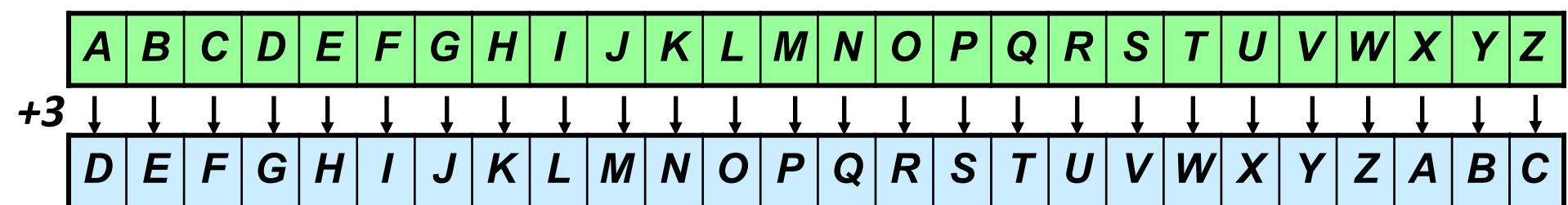
Confidencialidade: Cifra Simétrica

- Definição: transformação matemática reversível cujo cálculo depende, no sentido direto (**cifração**) e no sentido inverso (**decifração**), de uma mesma informação secreta: a **chave K**.
 - Se K for descoberta, a confidencialidade é perdida



Confidencialidade: exemplo didático

- Cifra de César: deslocar cada letra da mensagem de k posições no alfabeto latino (i.e., somar k posições).
 - Decifrar equivale a subtrair a mesma chave k
 - Técnica usada por Júlio César para troca de mensagens com seus generais, com $k=3$.



Confidencialidade: exemplo didático

Exercício

- ❑ Decifrar a seguinte mensagem e identificar o valor de k usado (dica: $k \neq 3$):

- ❑ NBBN ENAVNUQX MN VNDB XUQXB ENV MX ENAMN
MJ WJCDANIJ

Confidencialidade: exemplo didático

Resposta: 9

- ESSE VERMELHO DE MEUS OLHOS VEM DO VERDE DA NATUREZA
- Alguns facilitadores:
 - **NBBN ENAVNUQX MN VNDB XUQXB ENV MX ENAMN MJ WJCDANIJ**



Cifras: Algoritmos Principais



- DES (*Data Encryption Standard*):
 - **Obsoleto** desde 2004 (chaves de 56 bits: muito curtas!)
- 3DES (*DES triplo*): aplicação tripla do DES
 - **Legado**: reaproveita implementação do DES simples
 - Chaves: $3 \times 56 = 168$ bits (mas segurança é de ~ 112 bits)
- RC4 (ArcFour):
 - Chave: tamanho variável (múltiplo de 8 bits, até 2048 bits).
 - **Legado**: uso seguro requer truques (descartar dados iniciais: `RC4_drop[n]`); não recomendado para aplicações futuras
- **AES (Advanced Encryption Standard)**:
 - **Padrão atual** (desde 2001): vencedor de concurso público iniciado em 1997
 - Chaves de 128/192/256 bits.



Exemplo prático: HTTP vs. HTTPS

- HTTP: dados passam em aberto na rede

(Untitled) - Wireshark

No.	Time	Source	Destination	Protocol	Info
22	2.007786	172.20.5.105	64.233.163.104	HTTP	GET /search?hl=pt-BR&source=hp&biw=&bih=&q=au
68	2.772309	172.20.5.105	217.31.57.165	HTTP	POST /v1/rating/ HTTP/1.1 (application/json)

Frame 22 (549 bytes on wire, 549 bytes captured)
Ethernet II, Src: Intel_F5:c4:49 (00:19:d1:f5:c4:49), Dst: JuniperN_86:10:db (00:14:f6:86:10:db)
Internet Protocol, Src: 172.20.5.105 (172.20.5.105), Dst: 64.233.163.104 (64.233.163.104)
Transmission Control Protocol, Src Port: 11188 (11188), Dst Port: http (80), Seq: 1, Ack: 1, Len: 495
Hypertext Transfer Protocol
GET /search?hl=pt-BR&source=hp&biw=&bih=&q=aula+seguranca&btnG=Pesquisa+Google HTTP/1.1\r\nRequest Method: GET
Request URI: /search?hl=pt-BR&source=hp&biw=&bih=&q=aula+seguranca&btnG=Pesquisa+Google
Request Version: HTTP/1.1
Host: www.google.com.br\r\nUser-Agent: Mozilla/5.0 (Windows NT 5.1; rv:2.0.1) Gecko/20100101 Firefox/4.0.1\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\nAccept-Language: pt-br,en-us;q=0.7,en;q=0.3\r\nAccept-Encoding: gzip, deflate\r\nAccept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\nKeep-Alive: 115\r\nReferer: http://www.google.com.br/\r\nDNT: 1\r\nConnection: keep-alive\r\n\r\n



Pesquisa Google:
“aula segurança”

Exemplo prático: HTTP vs. HTTPS

- HTTPS: dados cifrados (túnel TLS)

Screenshot of Wireshark showing network traffic for an HTTPS session.

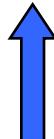
The packet list shows:

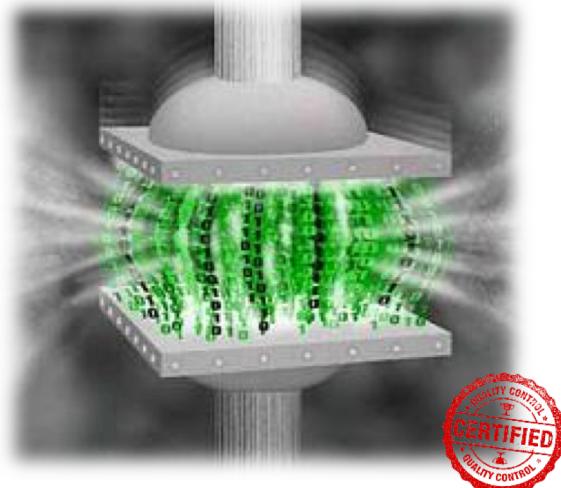
No.	Time	Source	Destination	Protocol	Info
22	3.783756	172.20.5.105	143.107.111.22	TLSv1	Application Data
27	3.921706	172.20.5.105	143.107.111.22	TLSv1	Application Data
33	4.101639	172.20.5.105	143.107.111.22	HTTP	GET /src/webmail.php HTTP/1.1
37	4.131323	172.20.5.105	143.107.111.22	TLSv1	Application Data
40	4.240812	172.20.5.105	143.107.111.22	TLSv1	Application Data
45	4.256434	172.20.5.105	143.107.111.22	SSL	Client Hello
48	4.262424	172.20.5.105	143.107.111.22	TLSv1	Change Cipher Spec, Encrypted Handshake Message,

The details pane shows:

- Frame 37 (651 bytes on wire, 651 bytes captured)
- Ethernet II, Src: Intel_f5:c4:49 (00:19:d1:f5:c4:49), Dst: JuniperN_86:10:db (00:14:f6:86:10:db)
- Internet Protocol, Src: 172.20.5.105 (172.20.5.105), Dst: 143.107.111.22 (143.107.111.22)
- Transmission Control Protocol, Src Port: 11172 (11172), Dst Port: https (443), Seq: 1419, Ack: 3513, Len: 597
- Secure Socket Layer
 - TLSv1 Record Layer: Application Data Protocol: http
Content Type: Application Data (23)
Version: TLS 1.0 (0x0301)
Length: 592
Encrypted Application Data: 2FE8B7CD4A39E582C6F3359D659DD90C84635559F1E357BD...

Dados enviados durante
login em webmail



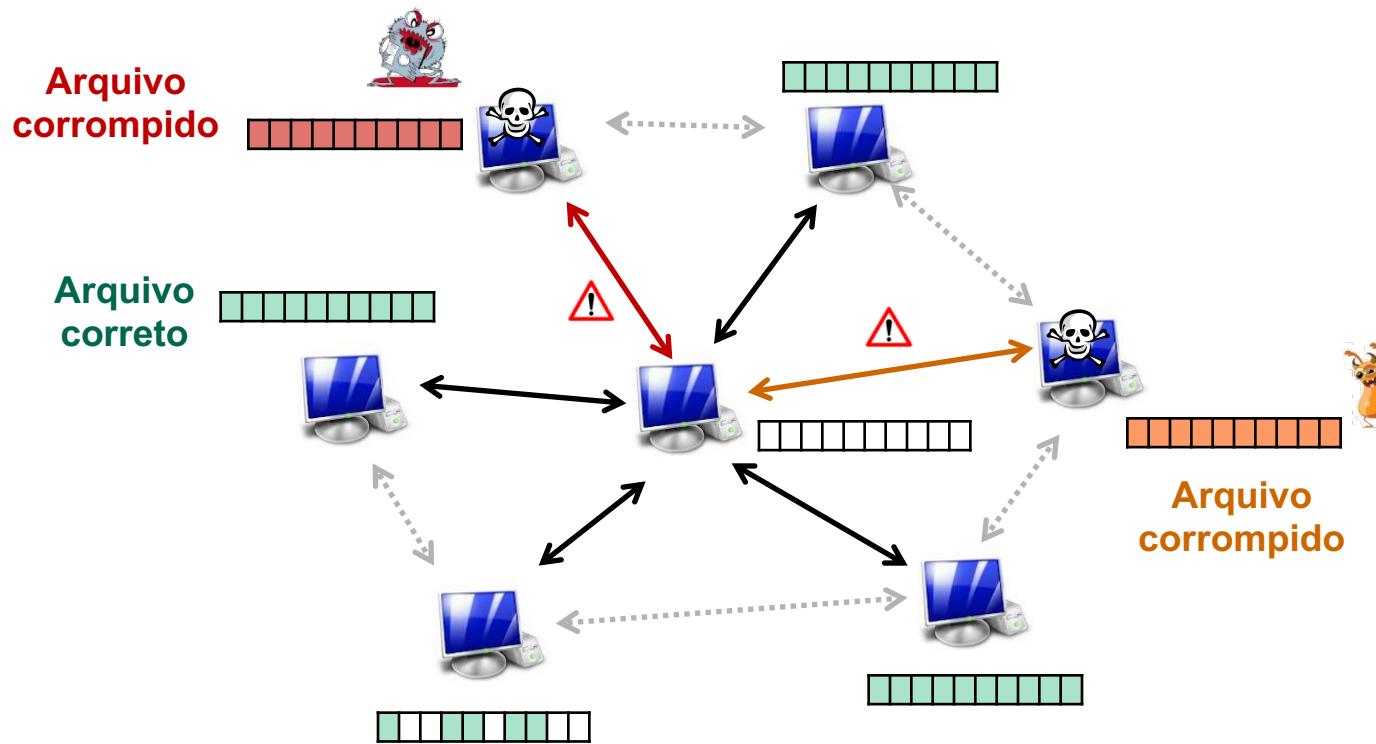


Funções de Hash & Códigos de Autenticação de Mensagens

Integridade e Autenticidade de dados

Integridade

- ❑ Cenário: modificação de arquivos em rede P2P
 - Arquivo vêm de diversos usuários (alguns maliciosos)



Integridade: ideia básica

- Ex. prático (não-criptográfico): modificações acidentais
 - RG/CPF: usa Dígito verificador (DV)
 - Método: “mod 11”
 - Dígito é multiplicado por sua posição, indo do menos significativo (peso 2) até o mais significativo
 - Os resultados são somados
 - DV: resto da divisão desta soma por 11

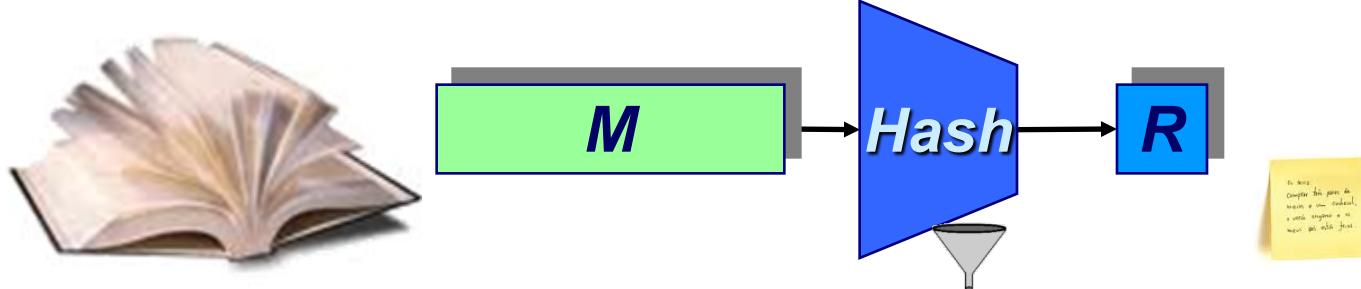
Exemplo simplificado

Entrada:	2	3	5	9	2
Posição:	6	5	4	3	2
Multiplicação:	12	15	20	27	4
Soma:	78				
DV:	$78 \text{ mod } 11 = 1$				

Σ
 $\text{mod } 11$

Integridade: Funções de Hash

- Geram redundâncias que são anexadas a mensagens com o propósito de detectar alterações
 - A redundância é chamada de “**hash**” ou “**resumo criptográfico**” da mensagem
 - O hash tem tamanho fixo, e seu valor depende exclusivamente da mensagem (não existe uma chave secreta envolvida no processo)

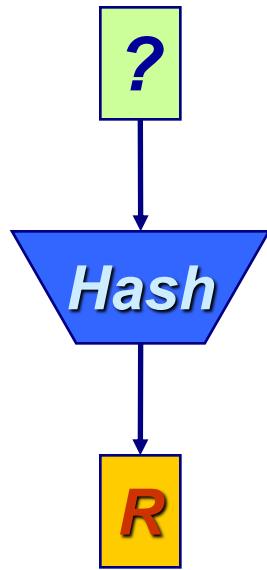


Integridade: Funções de Hash

- **Requisitos criptográficos fundamentais**
- (Resistência a primeira inversão) Dado um resumo R , é inviável encontrar uma mensagem M tal que $R = H(M)$.
- (Resistência a segunda inversão) Dado um resumo R e uma mensagem M_1 tal que $R = H(M_1)$, é inviável encontrar outra mensagem $M_2 \neq M_1$ tal que $R = H(M_2)$.
- (Resistência a colisões) É inviável encontrar duas mensagens M_1 e M_2 tais que $H(M_1) = H(M_2)$.

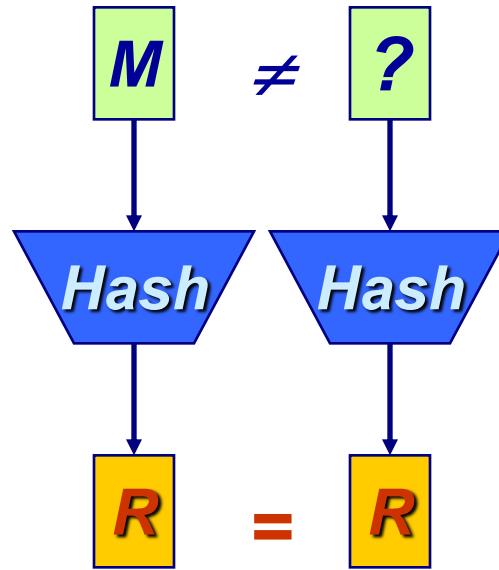
Integridade: Funções de Hash

1^a inversão



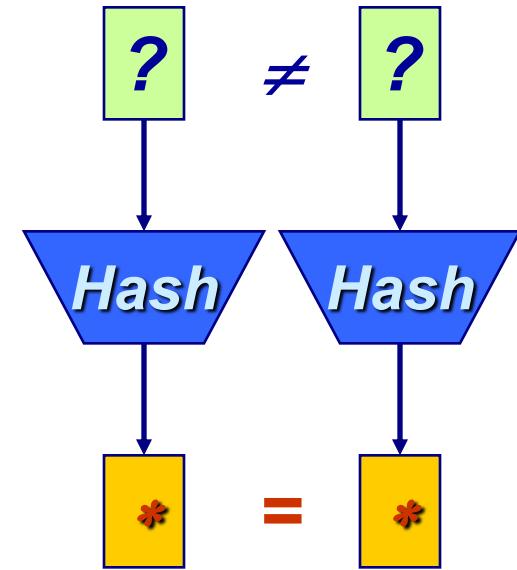
$$2^n$$

2^a inversão



$$2^n$$

colisão



$$2^{n/2}$$

Custo do ataque para hash de n bits

Integridade: Funções de Hash



- Família MD:

- MD2, MD4 e MD5: hashes de 128 bits
 - Completamente quebrada (Wang et al., 2004)



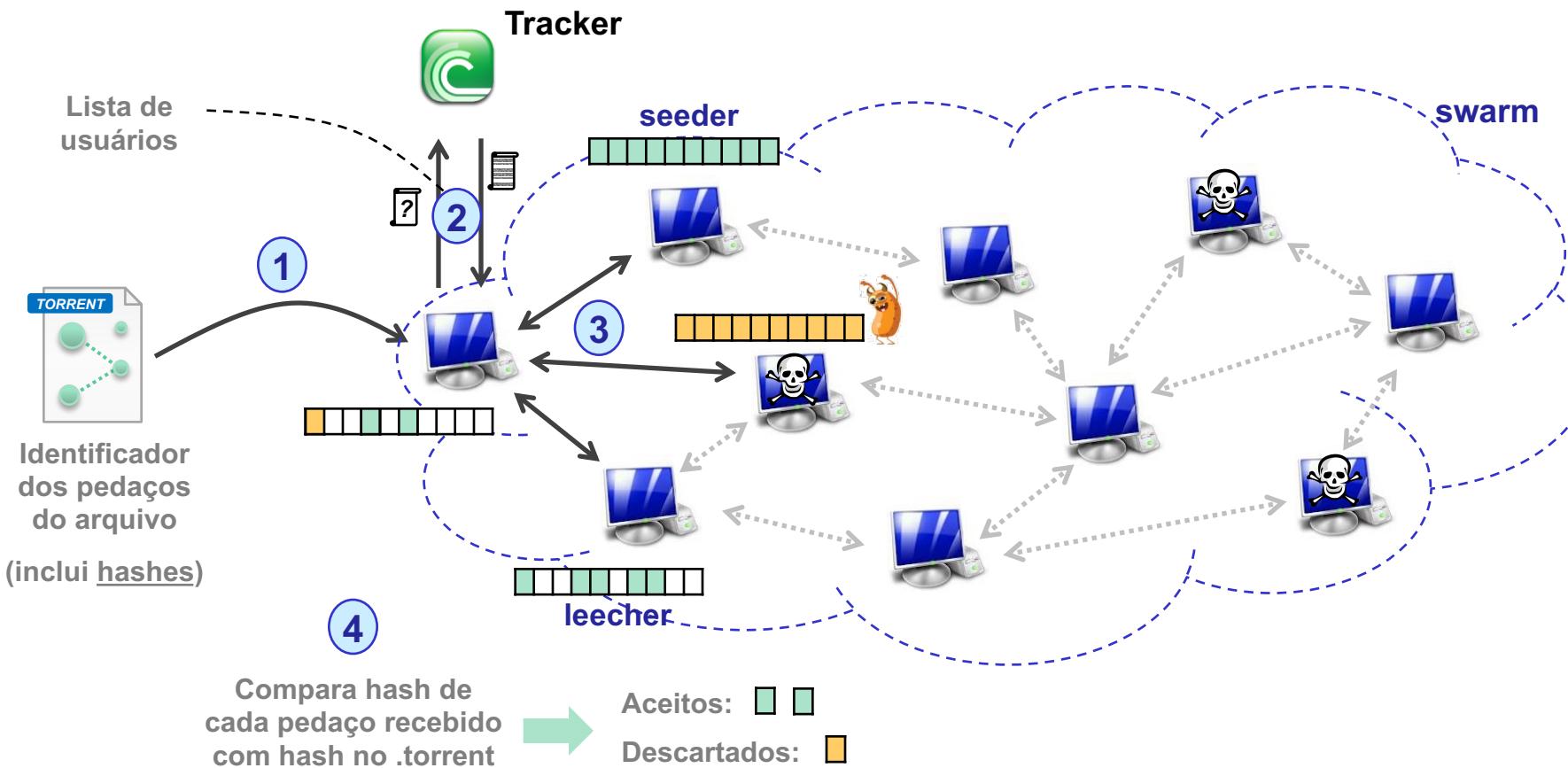
- Família SHA

- SHA-0: hashes de 160 bits
 - Não recomendado: colisão em 2^{39} passos x 2^{80} projetado
 - SHA-1: hashes de 160 bits
 - Não recomendado: desde 2010, para assinaturas
 - Segurança: colisões em 2^{60} passos x 2^{80} projetado
 - **SHA-2**: Hash de X bits, para X=224, 256, 384 ou 512
 - Paliativo atual: baseados no SHA-1, mas hash grande dificulta ataques
 - **SHA-3**: hashes de 224, 256, 384 e 512 bits
 - Concurso público finalizado em 2012: **Keccak**



Integridade: uso no Torrent

- ❑ Cenário: modificação de arquivos em rede P2P
 - Arquivo vem de diversos usuários (alguns maliciosos)



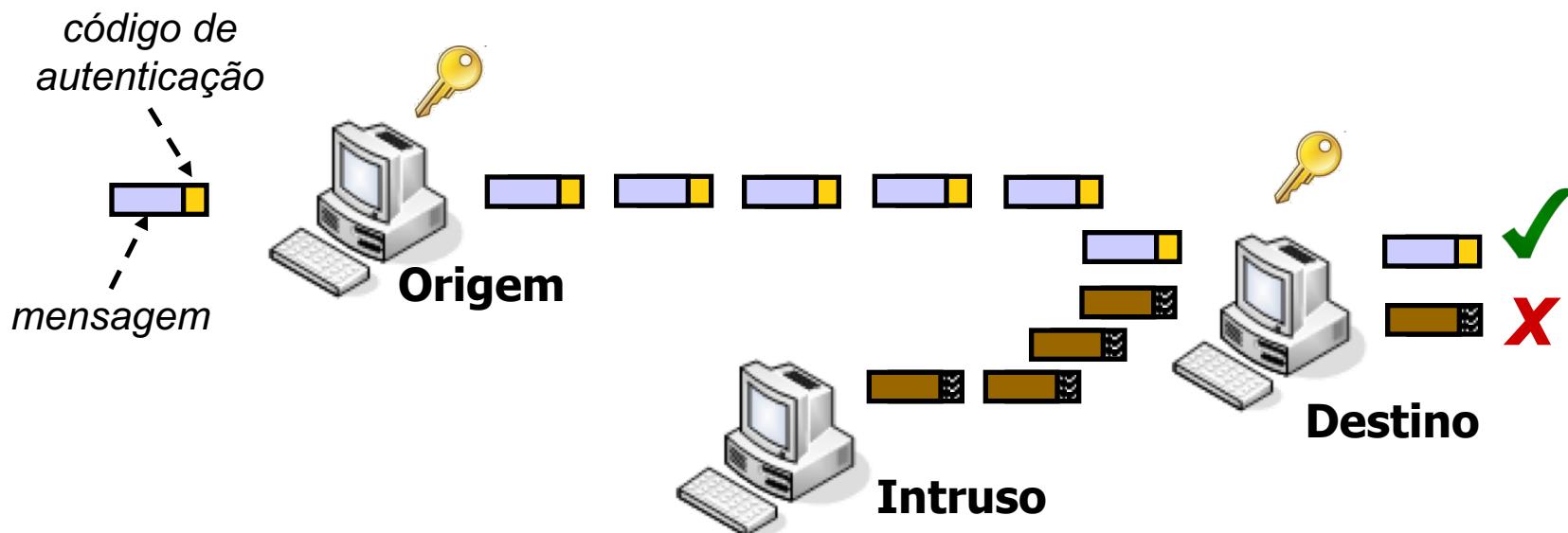
Hash: autenticidade (?)

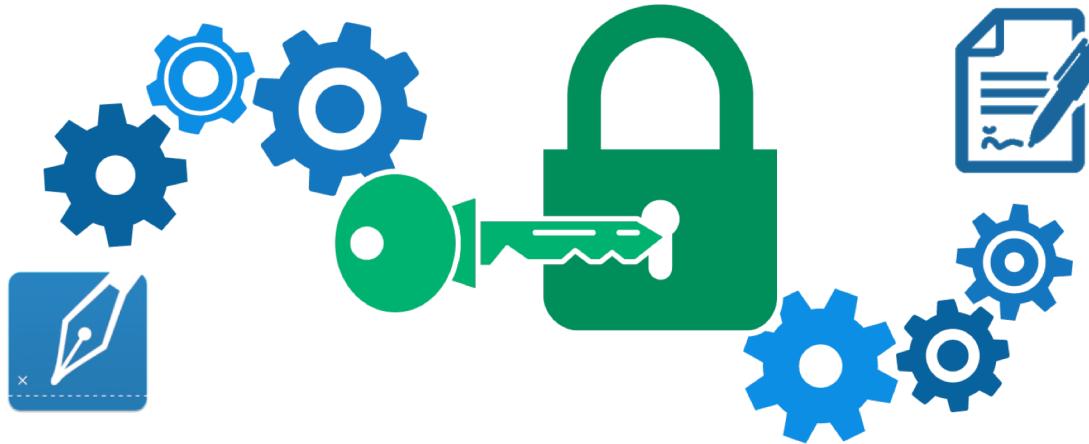
- Hash sozinho **não** provê autenticidade
 - Qualquer pessoa (incluindo intruso) pode calcular o hash da mensagem falsa...
 - O fato da mensagem estar íntegra não significa que foi um usuário legítimo quem a enviou...



Hash com chave: MAC

- Autenticidade obtida combinando hash com chave secreta: Código de Autenticação de Mensagem (MAC)
 - Apenas origem e destino conhecem chave e conseguem calcular códigos de autenticação corretamente
 - Também garante integridade: alteração na mensagem detectada como no caso das funções de hash





Criptografia de Chaves Públicas

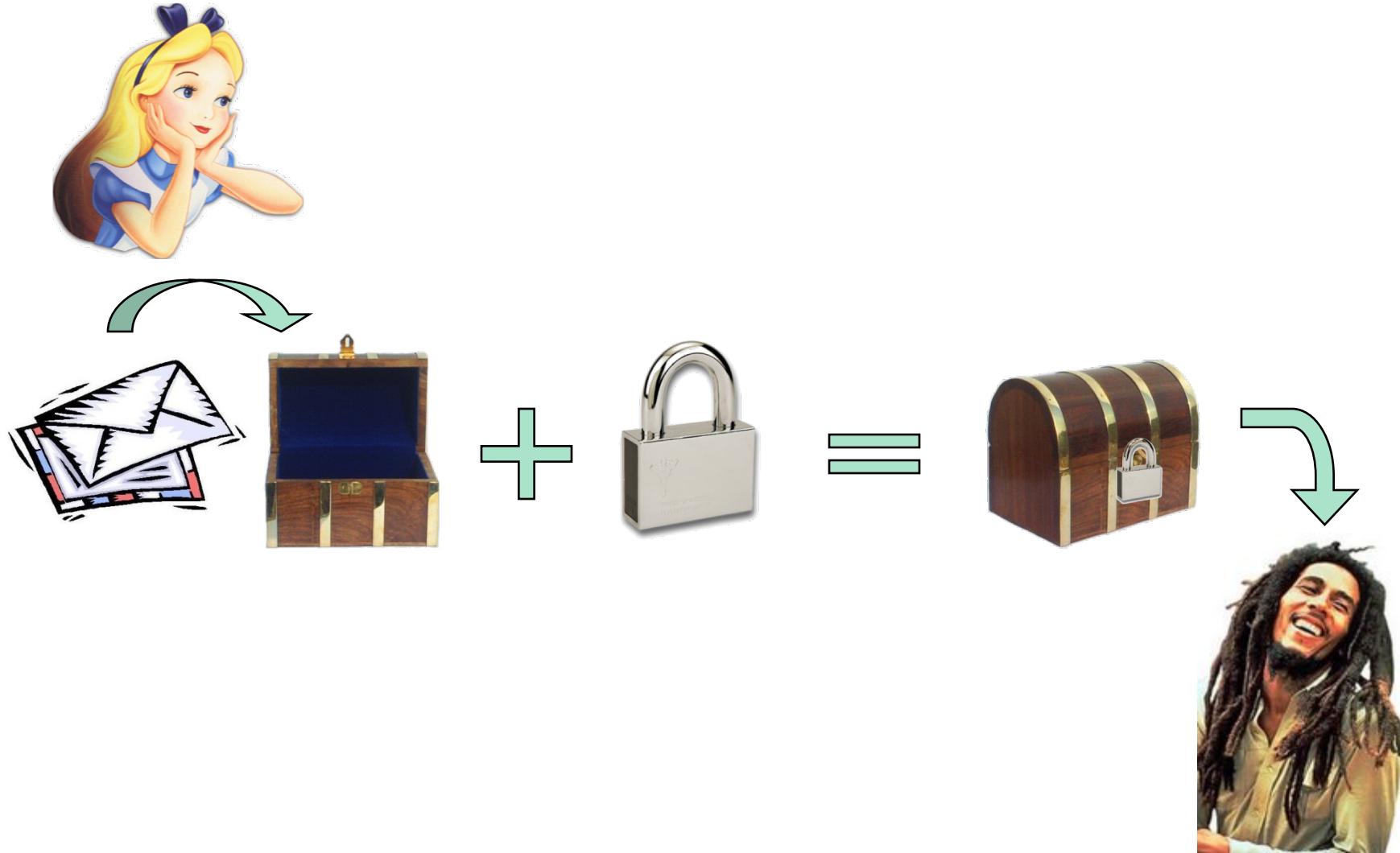
Assinaturas Digitais

Chaves Públicas: exemplo didático

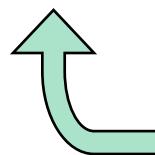
- Problema: Alice deseja enviar uma carta confidencial para Bob, usando apenas um baú com cadeado e suas respectivas chaves.



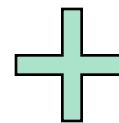
Protocolo Massey-Omura



Protocolo Massey-Omura



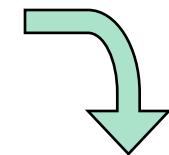
=



Protocolo Massey-Omura



=



Protocolo Massey-Omura



Protocolo Massey-Omura

- ❑ Alice deposita a carta na caixa, aplica o seu cadeado e envia a caixa para Bob.
- ❑ Bob aplica o seu cadeado e devolve a caixa (com dois cadeados!) para Alice.
- ❑ Alice remove o seu cadeado e envia a caixa de novo para Bob.
- ❑ Bob remove o seu cadeado e recupera a carta.

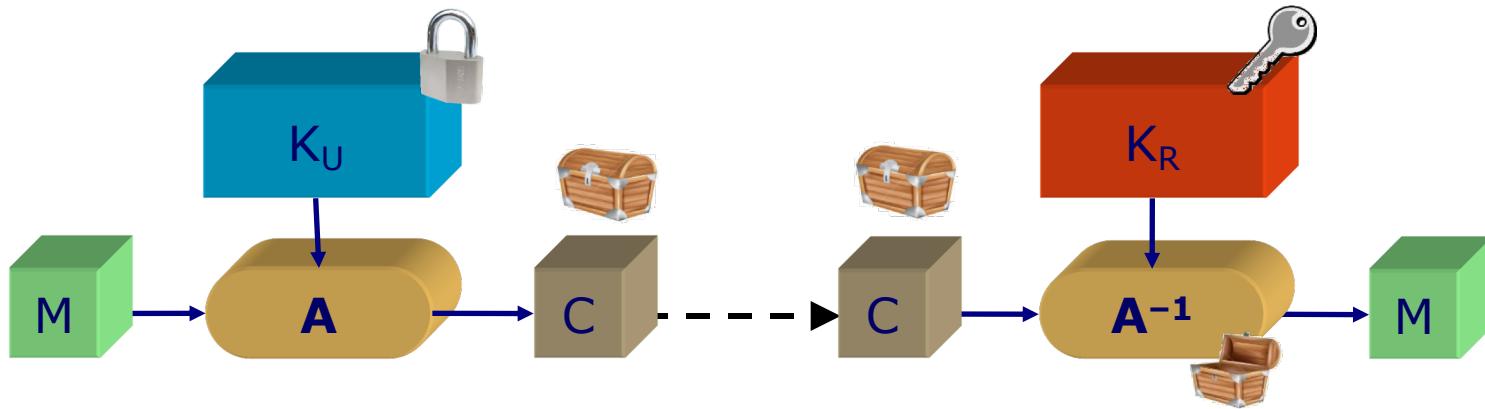
Criptografia Assimétrica

- ❑ Duas chaves distintas
 - Chave **pública** K_U : divulgada abertamente
 - Paralelo: o cadeado
 - Chave **privada** K_R : conhecida apenas pelo seu dono
 - Paralelo: a chave do cadeado
 - Transformações **feitas usando uma chave** somente podem ser **invertidas com a outra chave**.
- ❑ Ambas as chaves são **geradas pelo seu dono**
 - Em um algoritmo seguro, deve ser **inviável calcular a chave privada a partir da chave pública**.
 - Para se comunicar, usuários devem **obter**, de alguma forma, a **chave pública** de seus interlocutores

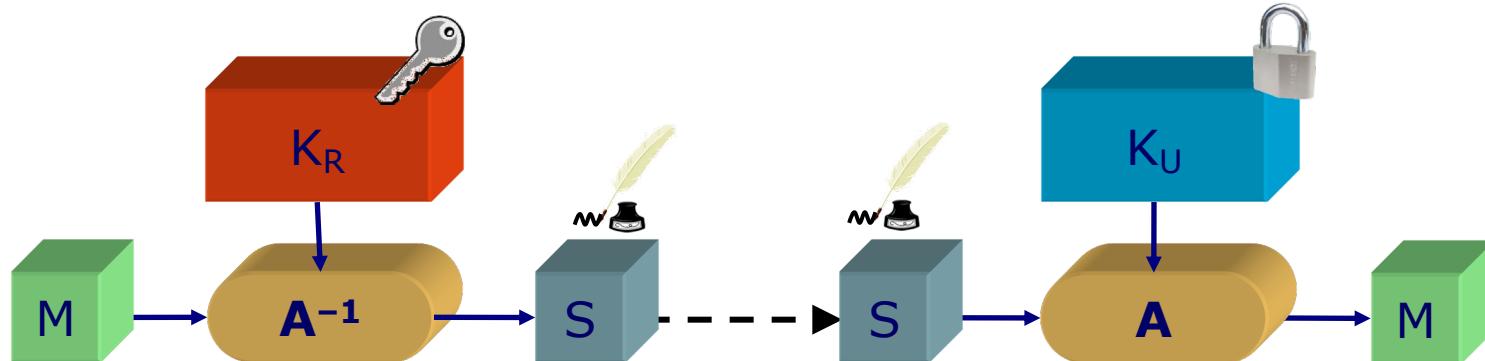


Criptografia Assimétrica

- ❑ Cifração: confidencialidade



- ❑ Assinatura digital: integridade, autenticidade e irretratabilidade

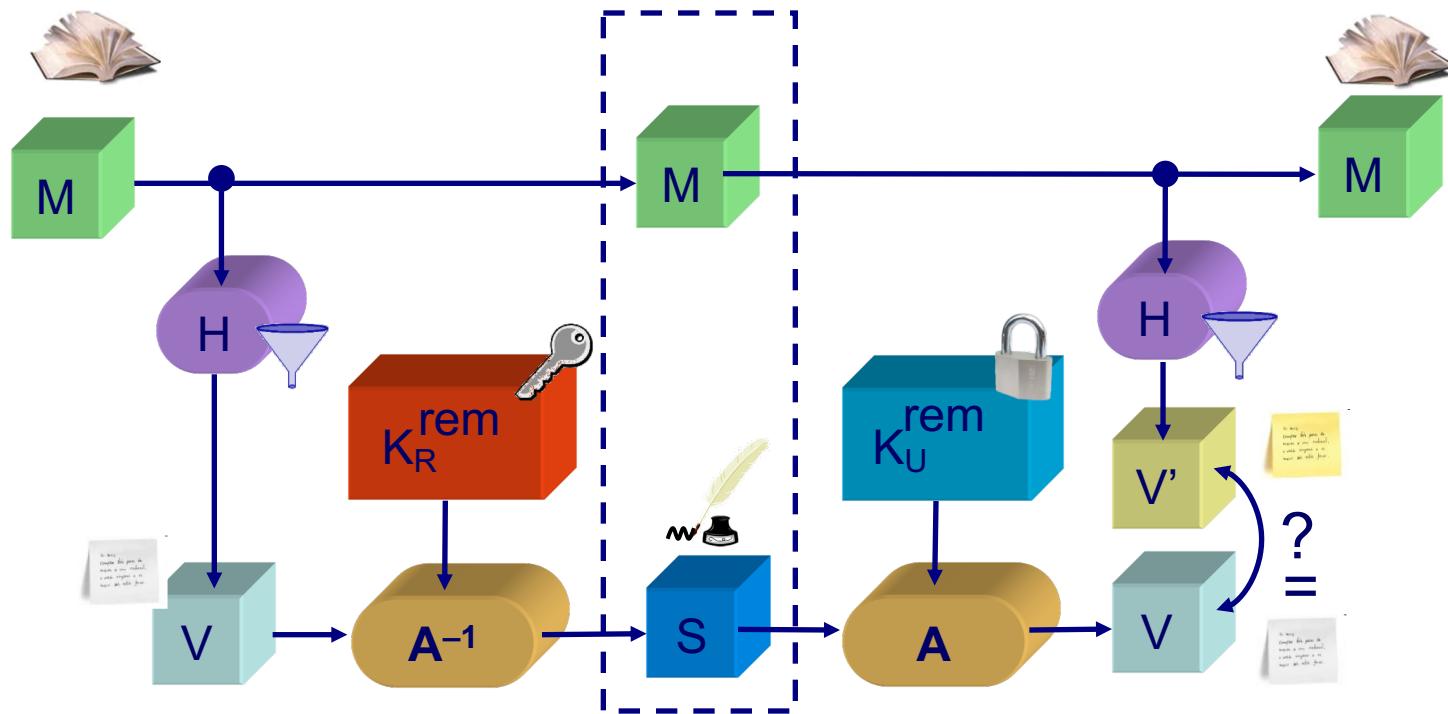


Criptografia assimétrica + simétrica

- ❑ Algoritmos **assimétricos** costumam ser **combinados com simétricos** por razões de **desempenho**:
 - Algoritmos simétricos costumam ser ~1000 vezes mais rápidos do que assimétricos
- ❑ Exemplos comuns:
 - **Estabelecimento de chaves simétricas**: usadas por cifras simétricas (confidencialidade) e algoritmos de MAC (integridade + autenticidade)
 - **Assinatura digital do hash** da mensagem ao invés da mensagem em si: menor quantidade de dados processados pelo algoritmo assimétrico



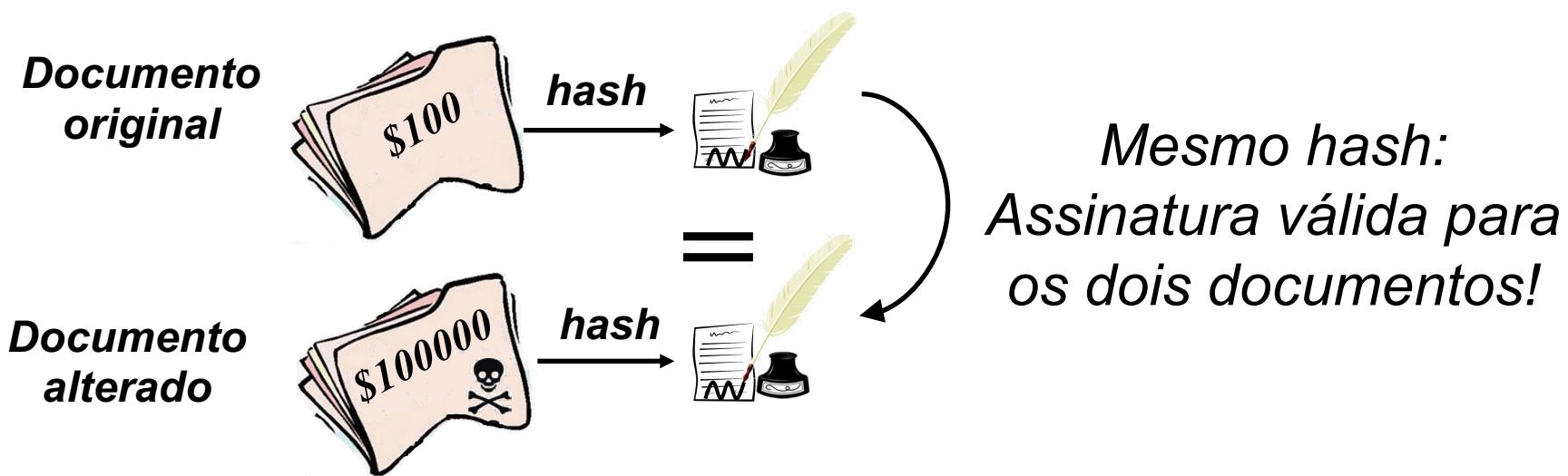
Geração e verificação de assinatura



- ❑ Mensagem **assinada** (S)
 - Serviços: integridade, autenticidade e irretratabilidade
- ❑ **Utilidade:** mais eficiente assinar hash das mensagens

Funções de hash e assinaturas

- Assinaturas digitais aplicadas sobre o hash da mensagem podem ser consideradas seguras se atacante for incapaz de encontrar uma segunda mensagem com o mesmo hash da mensagem assinada
 - Resistência à 2^a inversão e a colisões!!!





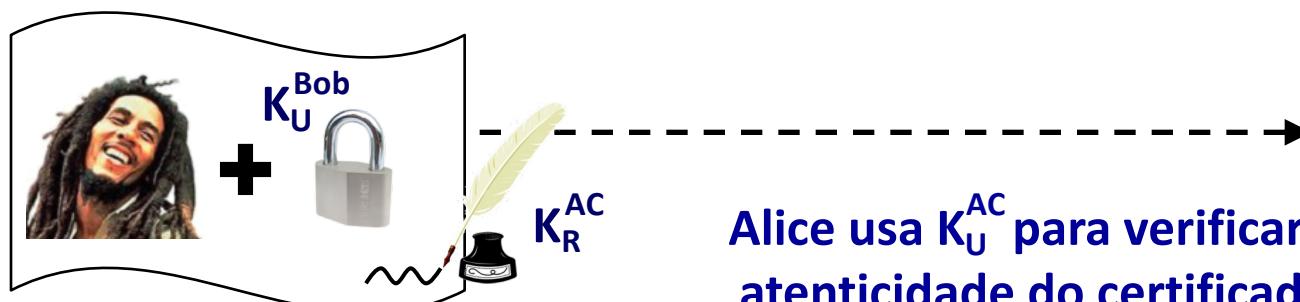
Certificação Digital

Distribuição de chaves públicas



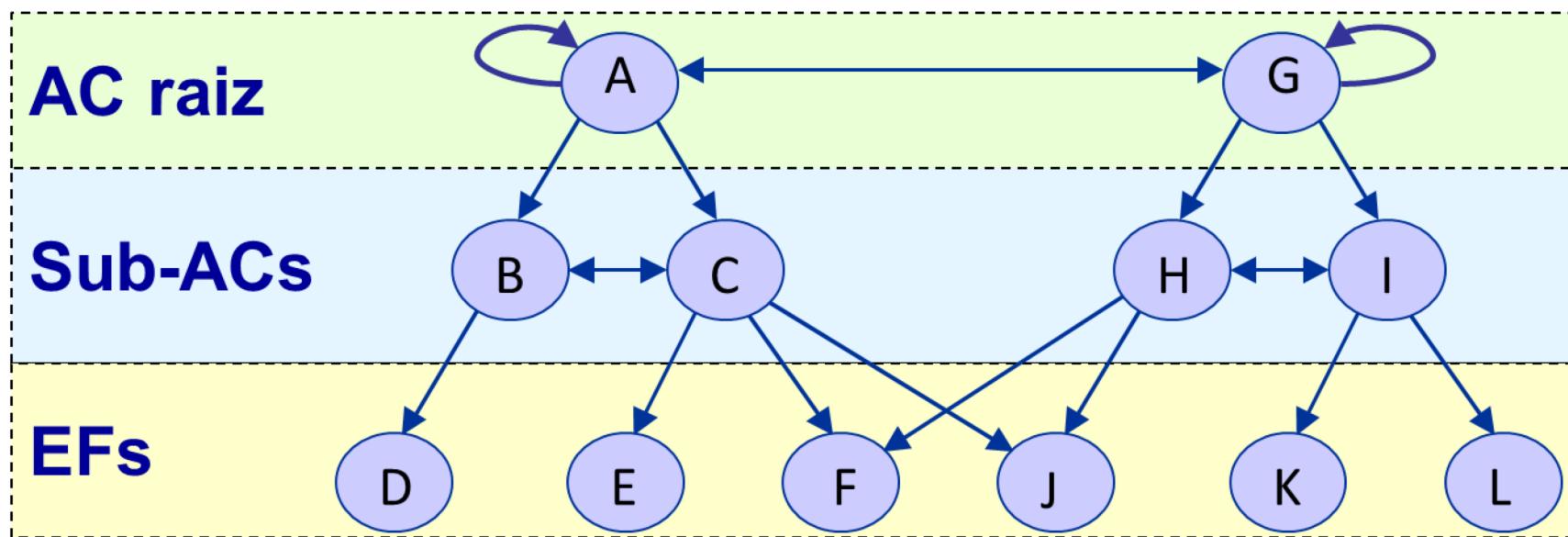
Certificados Digitais

- ❑ Associam uma chave pública ao seu dono.
 - Atestado dizendo qual é a chave pública de Bob
- ❑ Modelo PKI: certificado contém **chave pública** de Bob assinada por uma Autoridade Certificadora (AC)
 - **Premissa:** chave pública da AC é amplamente conhecida.
 - Na prática, **certificados das ACs são pré-instalados** em sistemas computacionais, como navegadores Web.
 - Também podem ser instalados por usuário

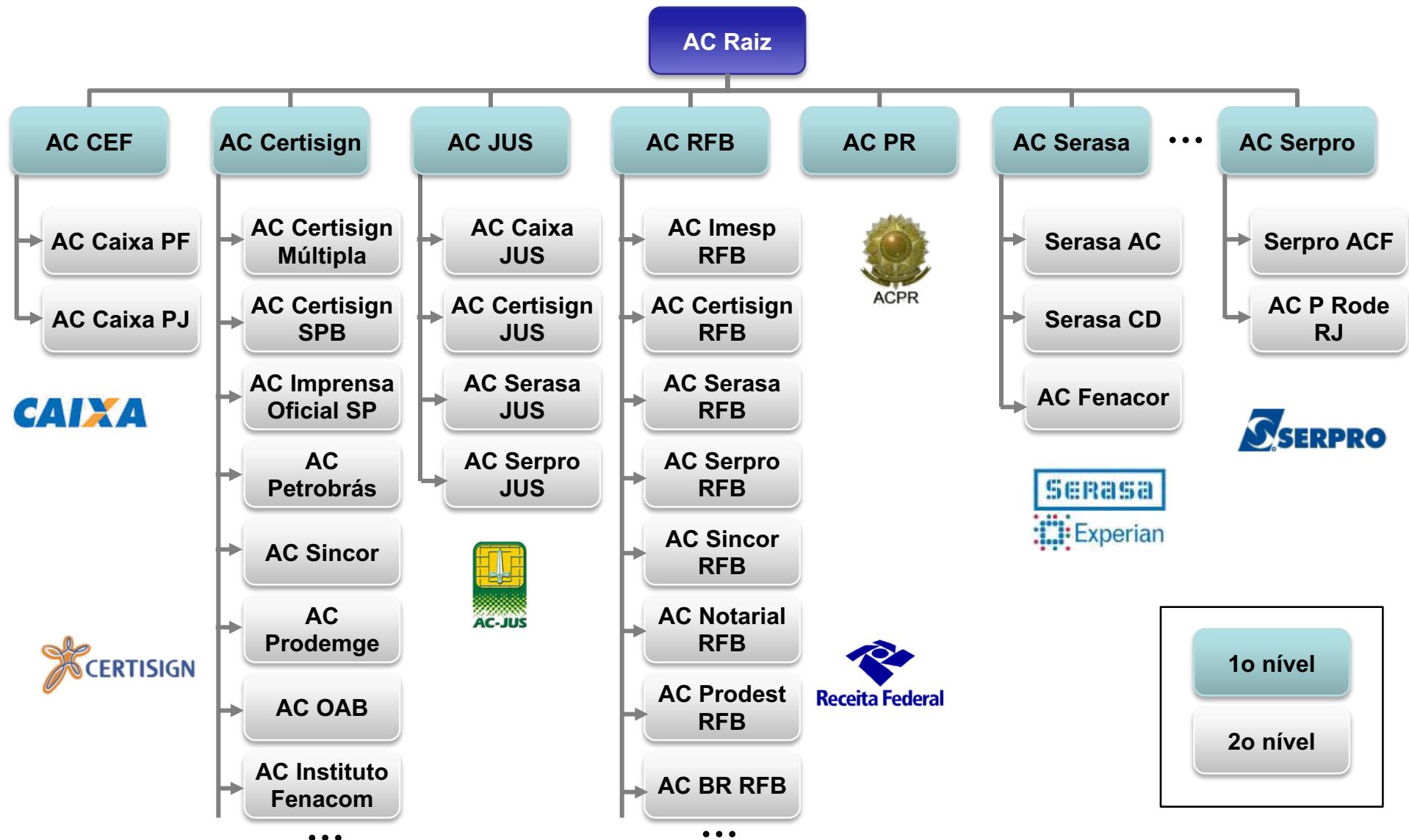


Certificados Digitais: PKI

- Modelo ICP (Infraestrutura de Chaves Públicas), ou PKI (*Public Key Infrastructure*): **cadeias de certificação**
 - Usa chave no certificado da **AC raiz** (auto-assinado) para assinar outras chaves na cadeia, até entidades finais (EFs)
 - Proteção** das chaves mais críticas (mais próximas da raiz)
 - ACs dedicadas a **vários fins**



Exemplo: ICP-Brasil

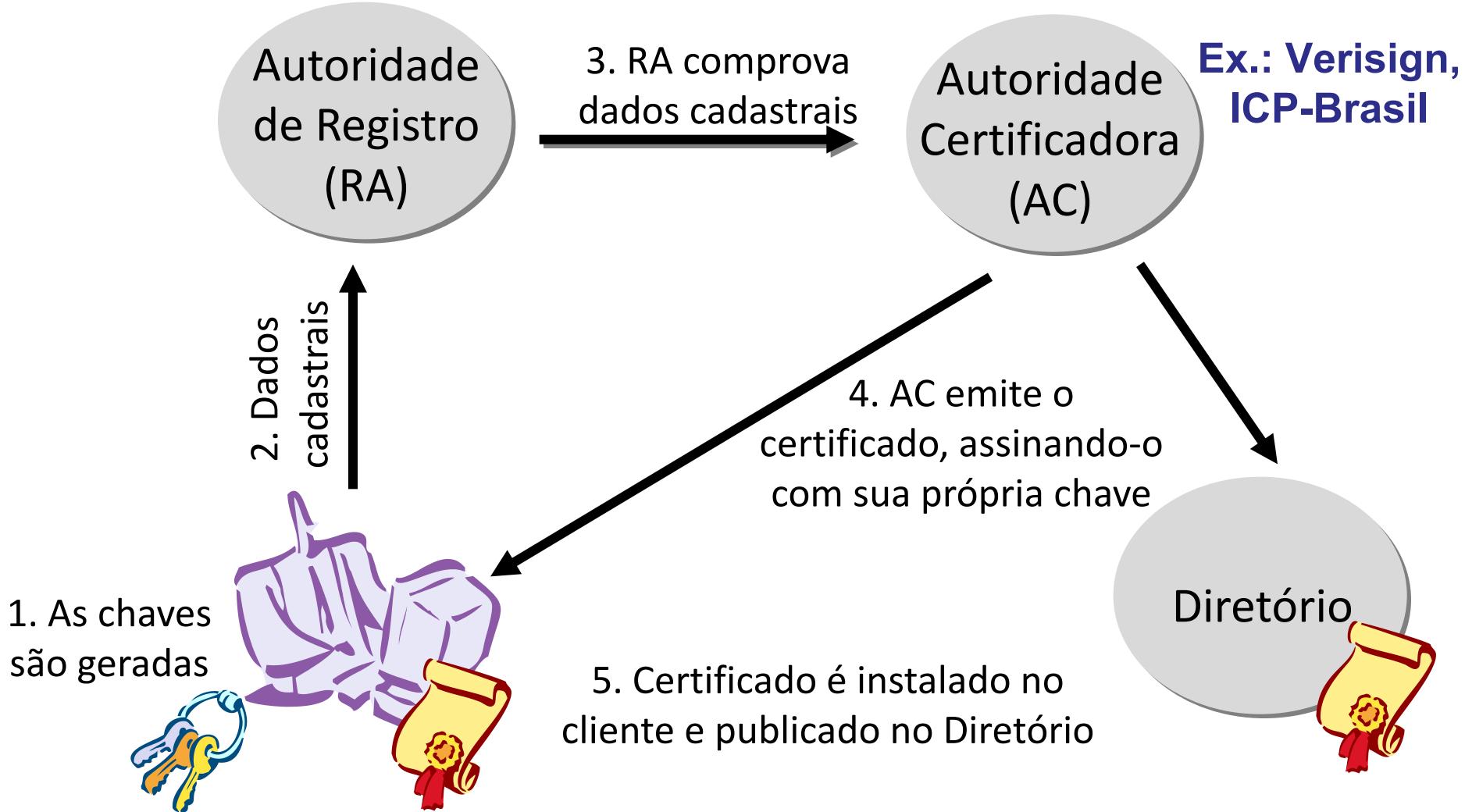


Processo de certificação

- A **verificação de identidades** é comumente delegada pelas ACs a entidades confiáveis: **Autoridades Registradoras (ARs)**.
 - As entidades finais (EFs) apresenta sua **chave pública e demonstram sua identidade**, por meios legais extra-criptográficos, às ARs
 - Dados a serem comprovados: dependem de legislação cabível e políticas da AC
 - Ex.: site web → provar que é dono do domínio
- Serviço descentralizado: por exemplo, distribuído geograficamente.



Processo de certificação



Certificados: Revogação



- ❑ Comprometimento da chave privada do usuário (ou, em um caso extremo, da AC).
- ❑ Alternativas:
 - **Offline:** *Certificate Revocation List* (CRL)
 - “Lista negra” emitida e assinada por AC, distribuída periodicamente.
 - Enumera identificadores (#serial) de certificados revogados não expirados e datas de revogação.
 - **Online:** *Online Certificate Status Protocol* (OSCP)
 - Protocolo web para consulta do status de certificados
 - Resposta assinada: “good”, “revoked” ou “unknown”

Distribuição de chaves públicas

▫ Distribuição de certificados

- Por e-mail (S/MIME), páginas web, etc.
- Durante o estabelecimento de conexão das aplicações (ex.: HTTPS)



▫ Confiança nas chaves públicas

- Certificados com chaves públicas assinadas por entidade de confiança
- Confiança de que os certificados não foram alterados no caminho até o receptor (ex.: certificados auto-assinados)
- “Web of trust”: se A confia em B e B confia em C, então A pode confiar em C (ex.: PGP)





Pretty Good Privacy (PGP)

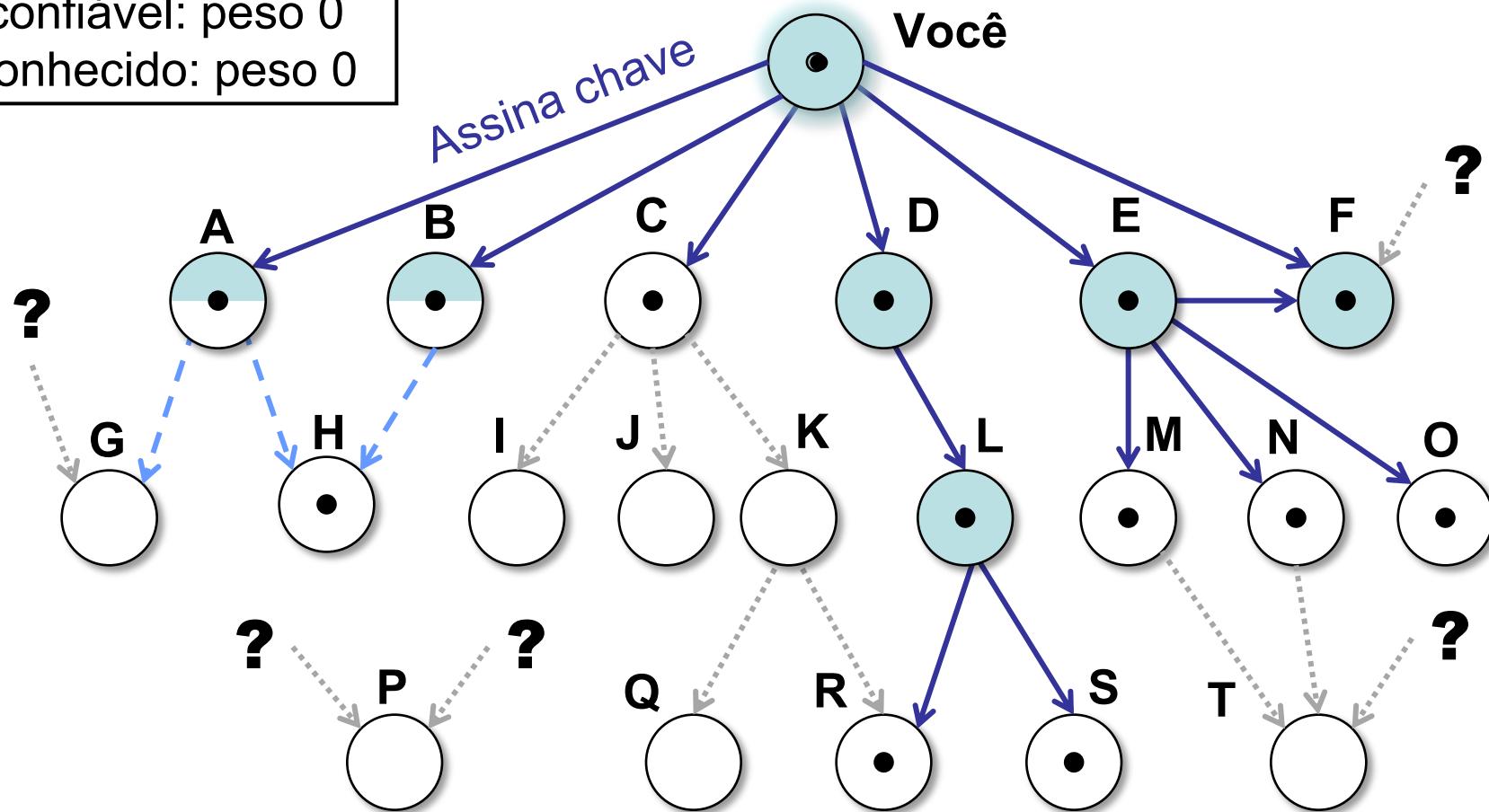
- ❑ Objetivo: prover um mecanismo de cifração forte e de uso fácil para todos
 - Foco: e-mails (cifração e assinatura)
 - Lançado por Phil Zimmermann em 1991
 - Versões abertas: OpenPGP e GNUPG
- ❑ Usa conceito de “molho de chaves”: **descentralização**
 - Usuários usam **certificados auto-assinados**, também disponibilizados em repositórios públicos
 - Usuários podem **assinar certificados de outros usuários** e definir níveis de confiança → **Web of Trust**
 - Cada usuário gerencia sua cópia local de chaves



Web of Trust (PGP)

- Confiável: peso 1
- Semi-confiável: peso $\frac{1}{2}$
- Não confiável: peso 0
- ? Desconhecido: peso 0

● Chave que você considerada legítima



Algoritmos e Protocolos Assimétricos: Exemplos

Nome	Uso	Chaves (128 bits)	Tamanhos
RSA	Assinatura ou cifração	3072	3072 (assinaturas ou dados cifrados*)
DH	Acordo de chaves	3072	3072 (mensagens trocadas)
DSA	Assinatura	3072	512 (assinaturas)
ECDSA	Assinatura	256	512 (assinaturas)
EdDSA	Assinatura	256	512 (assinaturas)
ECIES	Cifração	256	256 + 64** (dados cifrados*)

* Tamanho adicional ao da mensagem cifrada em si, usando cifra simétrica

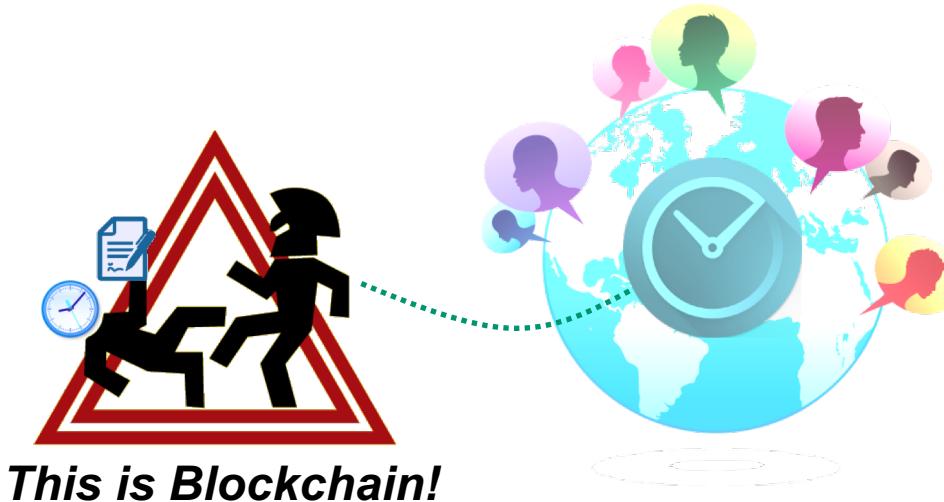
** ECIES usa MAC para calcular um tag de autenticação (64 bits ou maior)

Voltando à pergunta original: O que é um blockchain?



Blockchain: ordenação de eventos

- Objetivo do blockchain: determinar a **ordem** em que diversos eventos ocorreram em **rede distribuída**
 - Substitui uma autoridade de carimbo de tempo (**Timestamp Authority – TSA**) centralizada
 - Similar a AC, mas certifica horário em vez de chaves públicas
 - Pressupõe **ausência de confiança** entre os usuários



Bitcoin

- Para entender o funcionamento do blockchain, é interessante analisá-lo no contexto do Bitcoin
 - Afinal, essa é uma de suas principais aplicações!
- O que é o Bitcoin:
 - Livro de contabilidade (*ledger*) digital: permite verificar saldos ao analisar a **ordem de eventos** no sistema
 - Evento = transação assinada por usuário
 - Permite transações **sem intermediários**: totalmente **descentralizado**
 - Embora **plataformas de gerenciamento** ("exchanges") possam atuar como intermediários, facilitando o acesso
 - **Previne fraudes**, como duplicação de moedas: embora usuários **não sejam confiáveis**



Bitcoin (cont.)

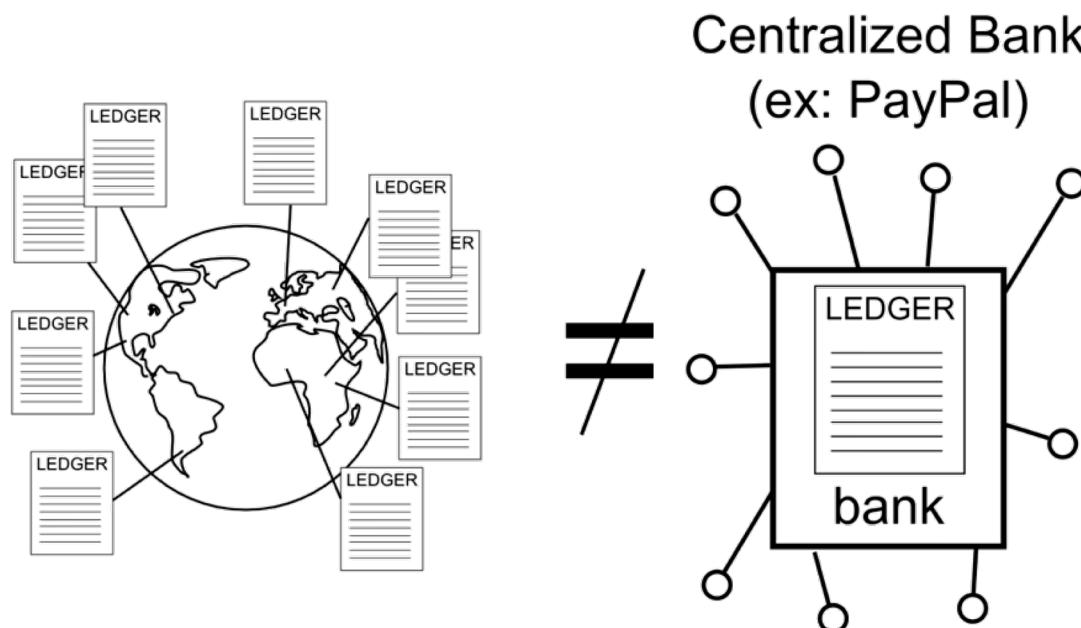
- ❑ Para entender o funcionamento do blockchain, é interessante analisá-lo no contexto do Bitcoin
 - Afinal, essa é uma de suas principais aplicações!
- ❑ O que é o Bitcoin:
 - Há **incentivos** para participação: *mineração* de moedas e *taxis* pagas pelas transações
 - Permite **pseudoanonimato**: usuários são representados por suas chaves públicas (pseudônimos)
 - Chaves públicas são sequência de bits sem qualquer relação com a identidade de seus donos!
 - Uso de diferentes chaves permite algum grau de privacidade
 - Embora existam várias técnicas para ligar um usuário a suas transações (e.g., análise estatística, rastreamento de IPs, etc.)



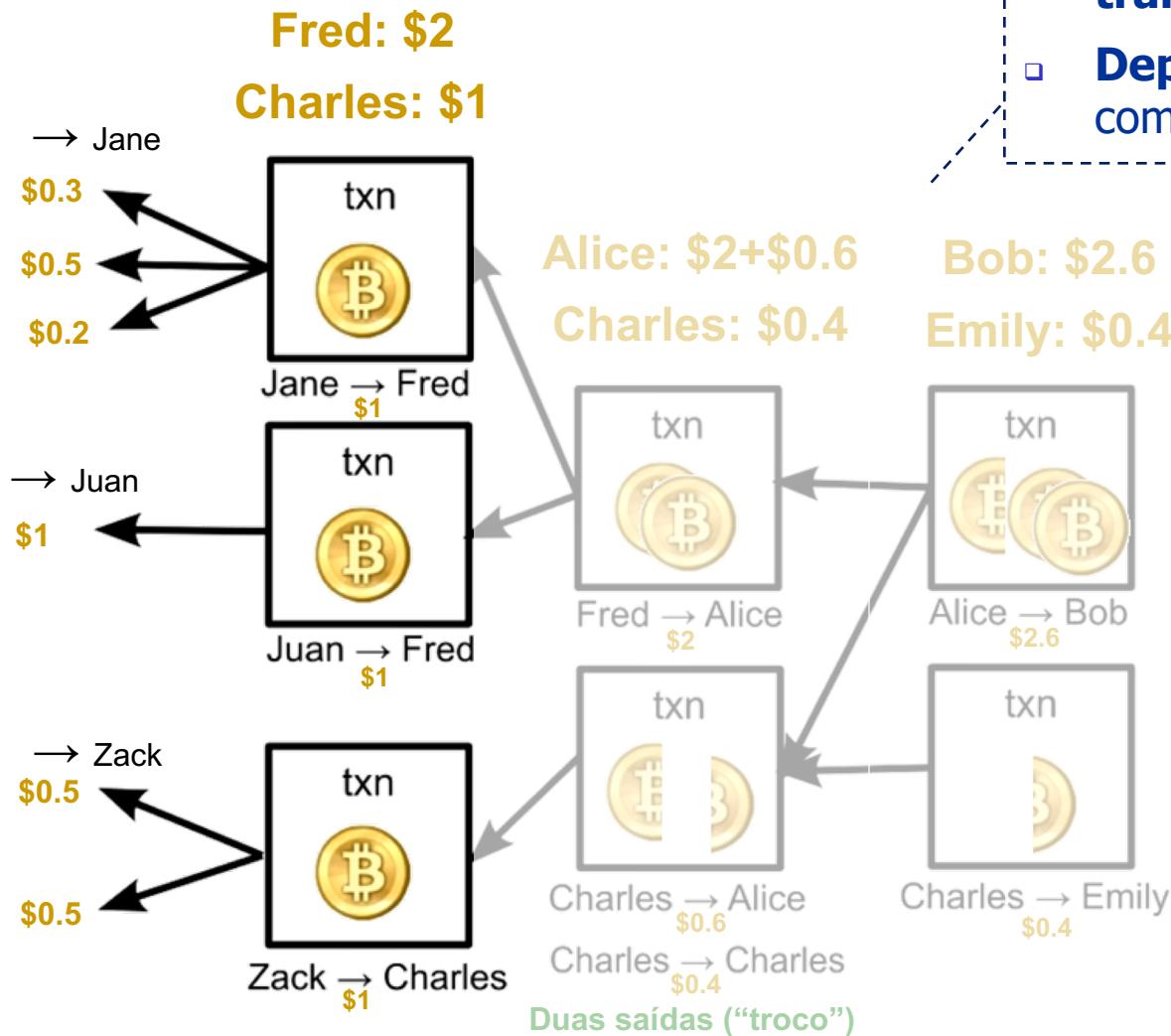
Bitcoin: explicações didáticas

▫ Vídeos recomendados:

- How Bitcoin Works in 5 Minutes (Technical):
<https://youtu.be/l9jOJk30eQs> (5 min)
- How Bitcoin Works Under the Hood:
<https://youtu.be/Lx9zgZCMqXE> (22 min)



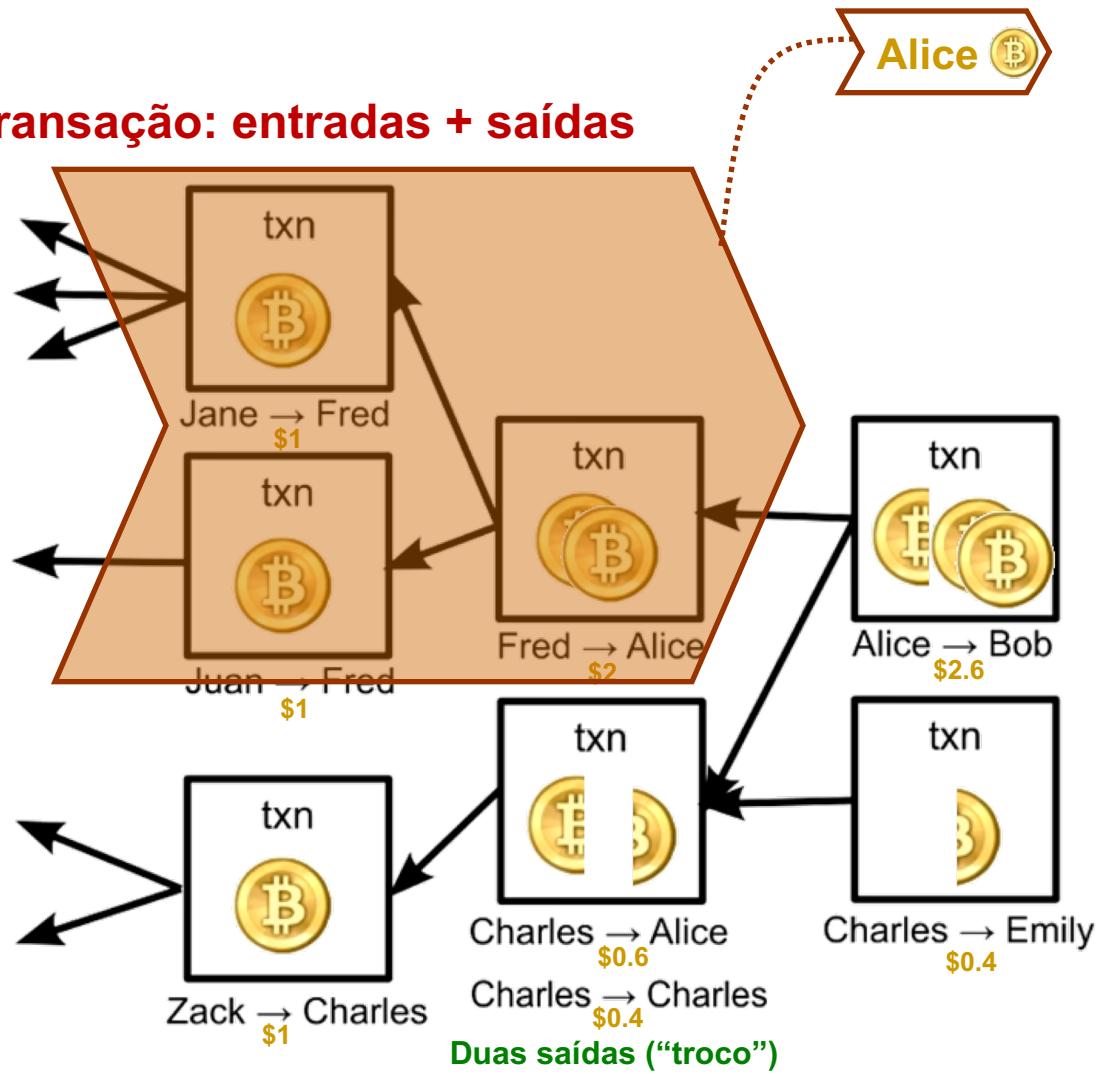
Bitcoin: transações



- **Início:** validação de **todas as transações** desde o 1º Bitcoin
- **Depois:** manutenção de base com **moedas não gastas**

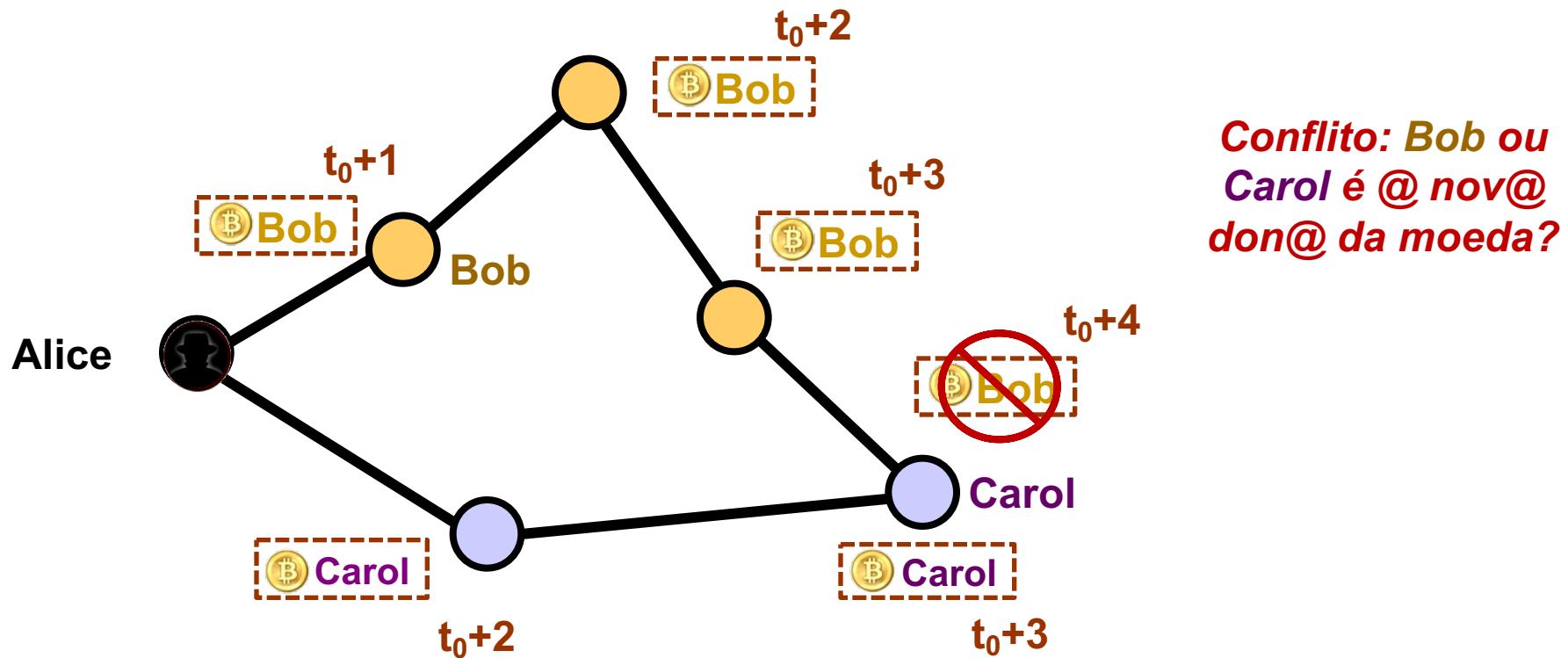
Bitcoin: transações

Transação: entradas + saídas



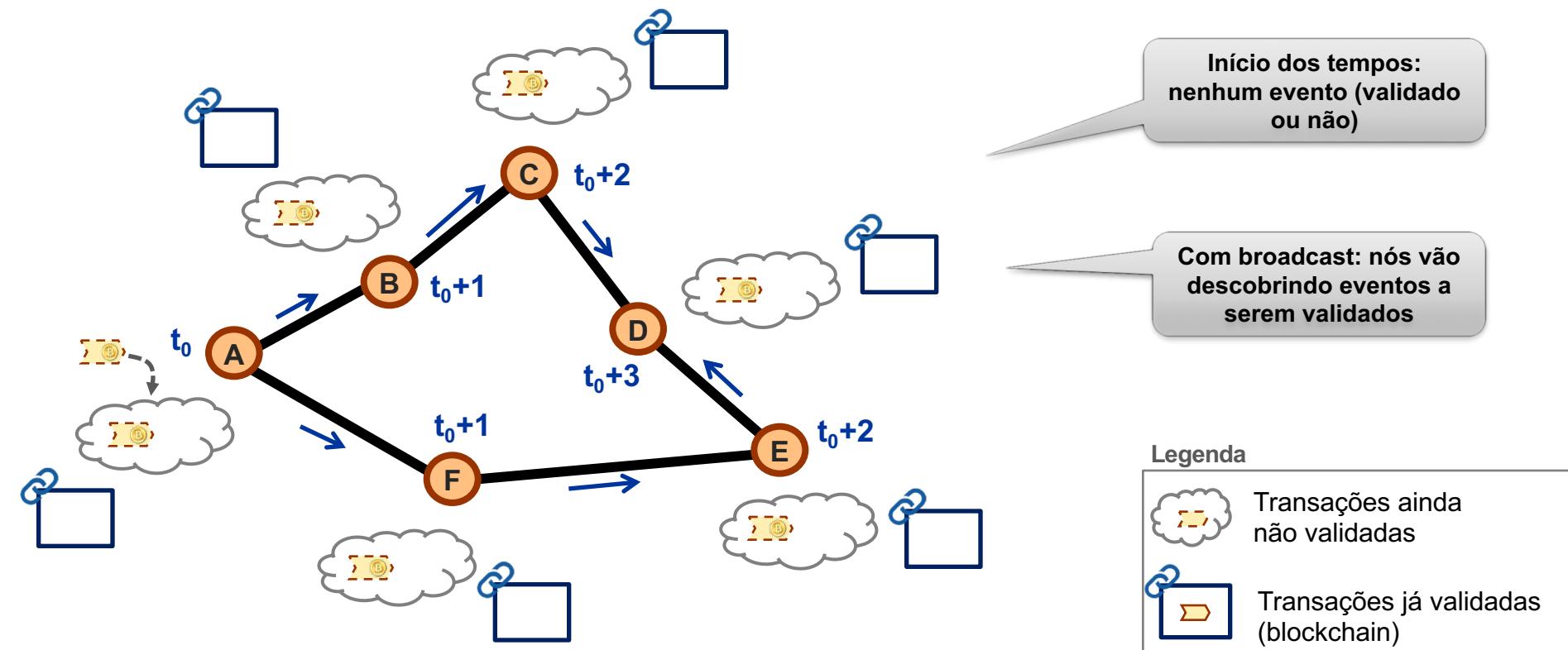
Bitcoin: transações

- **Problema alvo:** *Double spending* (ou “gasto duplo”)
 - Rede deve entrar em acordo sobre ordem de eventos!!!
 - Apenas após consenso, Carol/Bob entrega produto a Alice



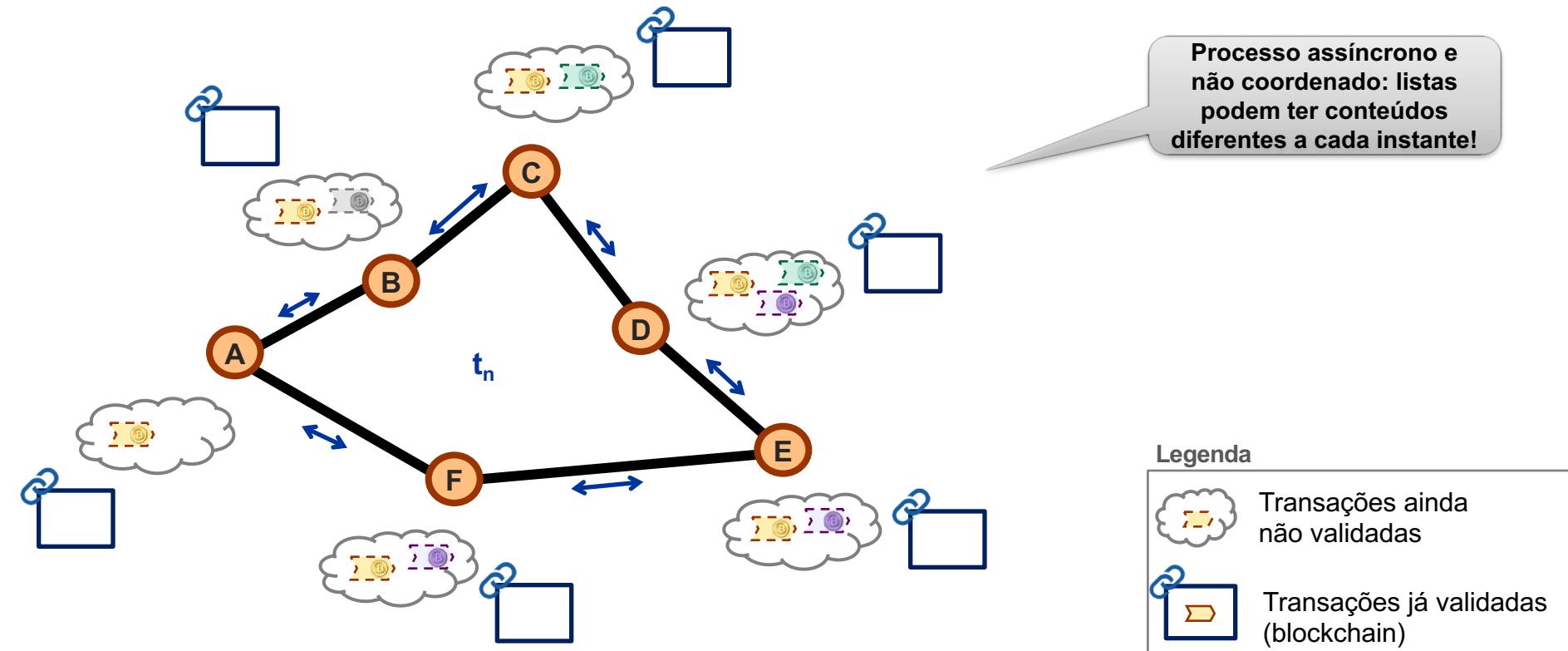
Blockchain: processando eventos

- Eventos processados em **duas fases**:
 - Fase 1:** nós informam sobre evento (broadcast)
 - Nós receptores adicionam evento a sua lista de não validados



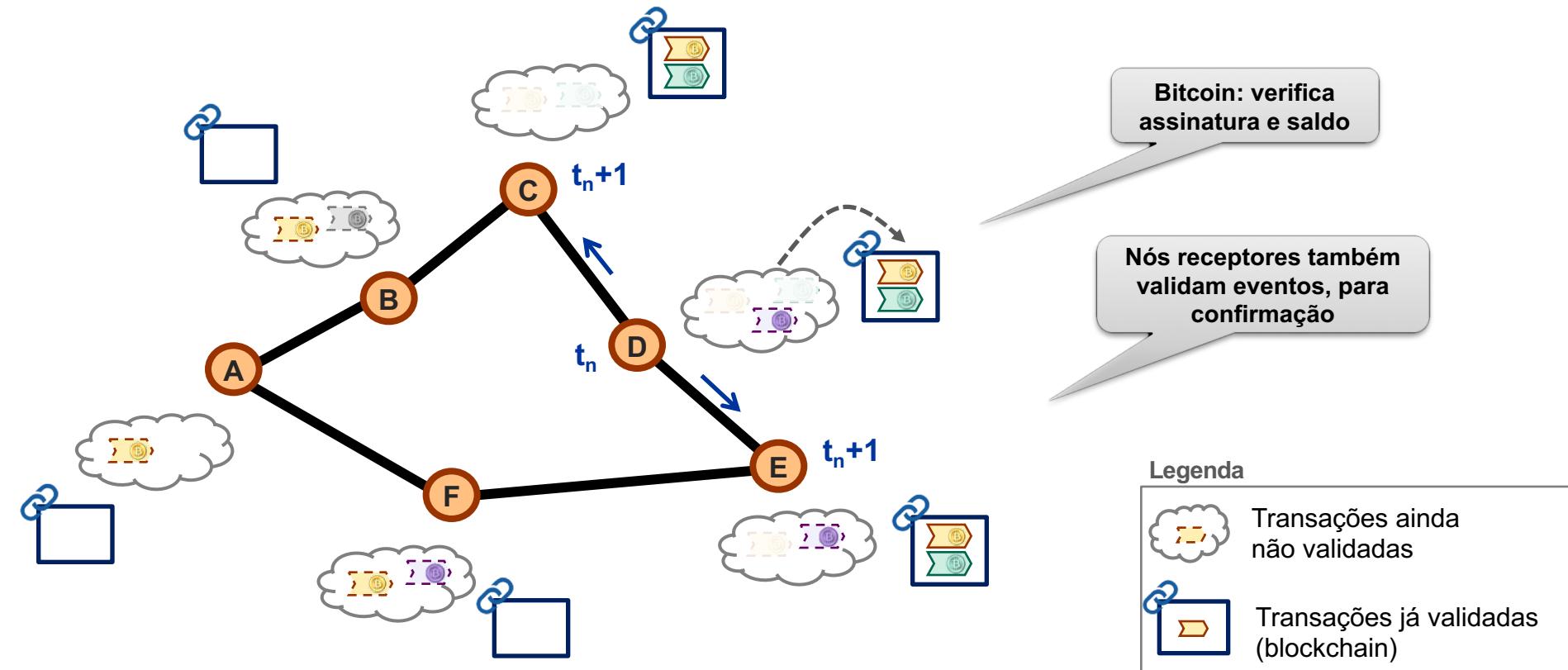
Blockchain: processando eventos

- Eventos processados em **duas fases**:
 - Fase 1:** nós informam sobre evento (broadcast)
 - Nós receptores adicionam evento a sua lista de não validados



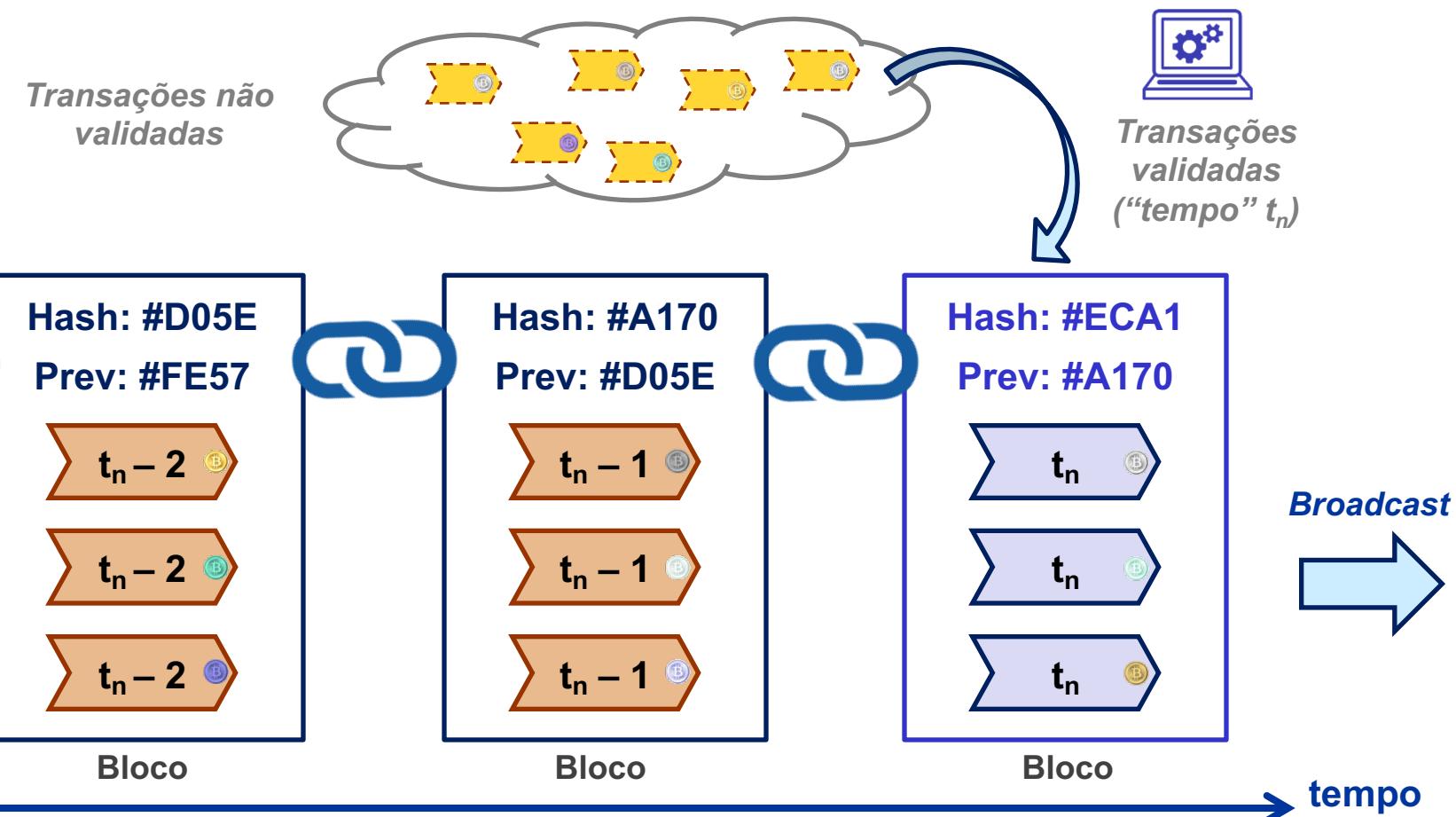
Blockchain: processando eventos

- Eventos processados em **duas fases**:
 - Fase 2:** nós validam eventos e informam rede (broadcast)
 - Eventos validados movidos para blockchain... mas como...?



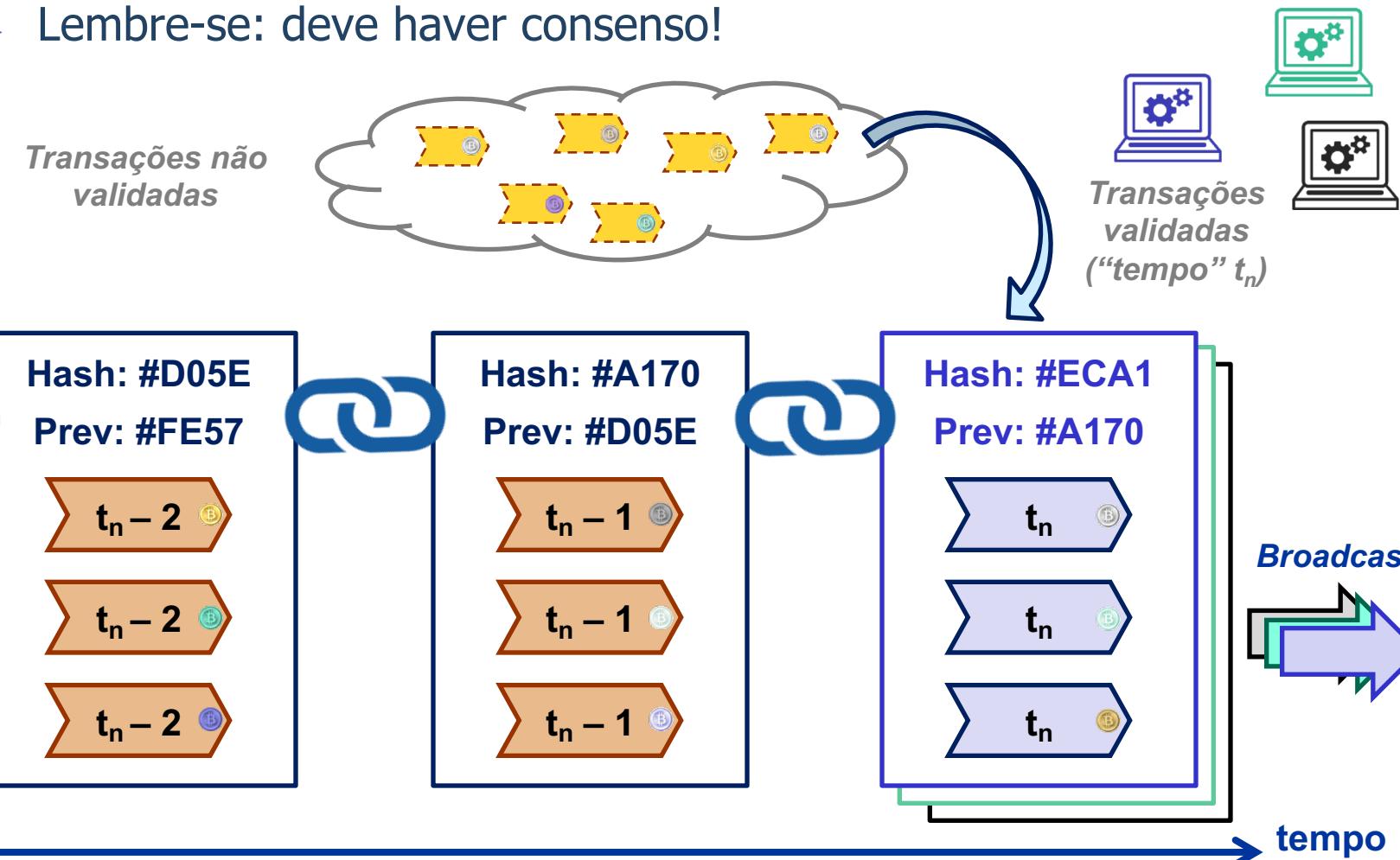
Blockchain: encadeamento de eventos

- Objetivo: definir a ordem em que eventos ocorrem
 - Blocos**: contêm conjuntos de transações



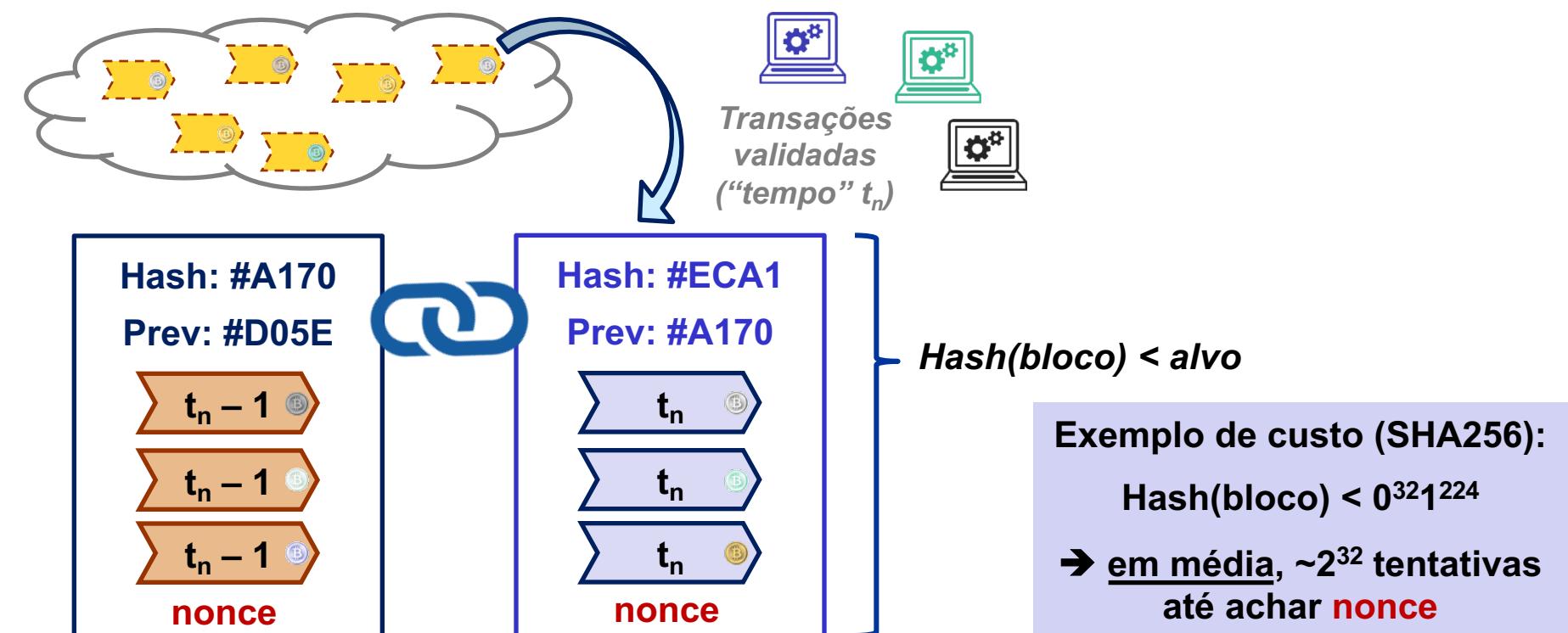
Blockchain: encadeamento de eventos

- Mas como decidir qual o próximo bloco “correto”?
 - Lembre-se: deve haver consenso!



Blockchain: encadeamento de eventos

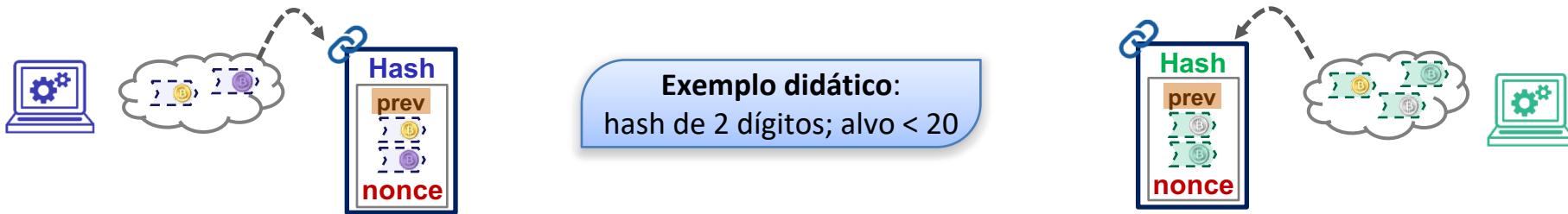
- Consenso: proof-of-work
 - O **primeiro que achar “nonce”**, faz broadcast para a rede toda
 - Nós da rede sempre incluem blocos sobre **maior cadeia** recebida



Blockchain: encadeamento de eventos

▫ Consenso: proof-of-work

- O **primeiro que achar “nonce”**, faz broadcast para a rede toda
- Nós da rede sempre incluem blocos sobre **maior cadeia** recebida



tempo		
t_n	0	88
t_n+1	1	17
t_n+2	2	29
t_n+3	4	33
t_n+4	8	13
t_n+5	16	02
	...	

← broadcast

propagando...

propagando...

propagando...

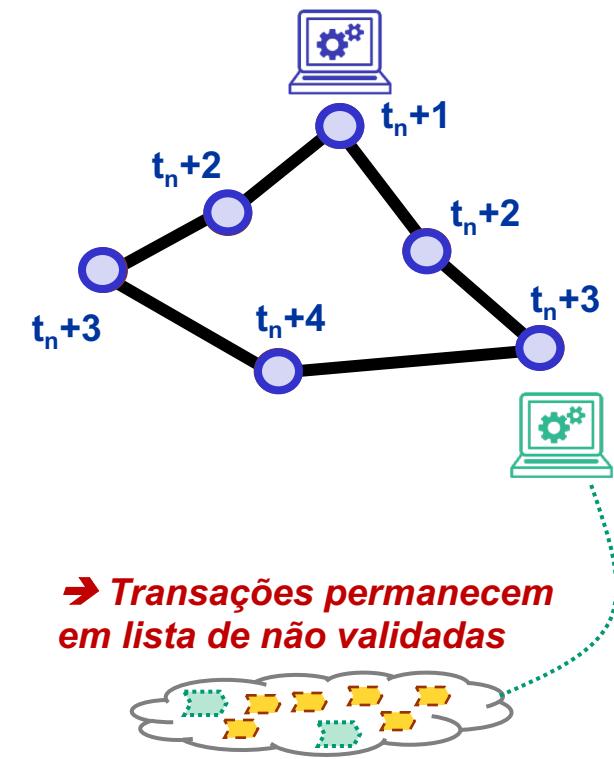
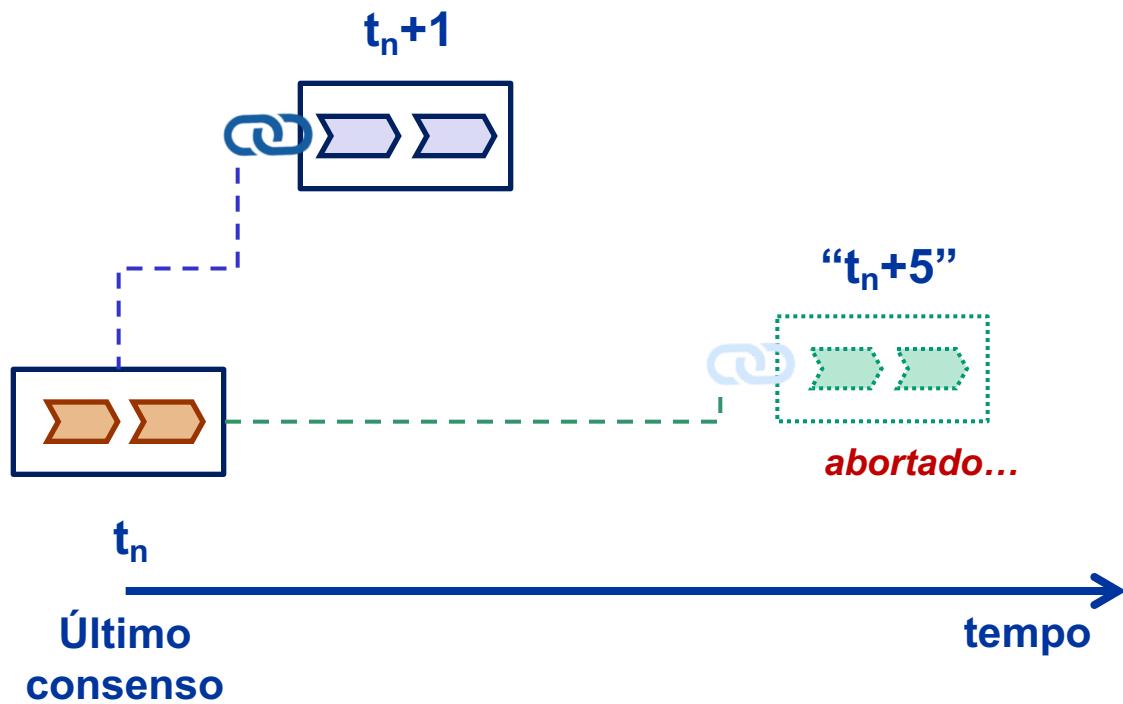
propagando...

broadcast →

0	29
1000	54
2000	91
3000	38
4000	54
5000	12
...	

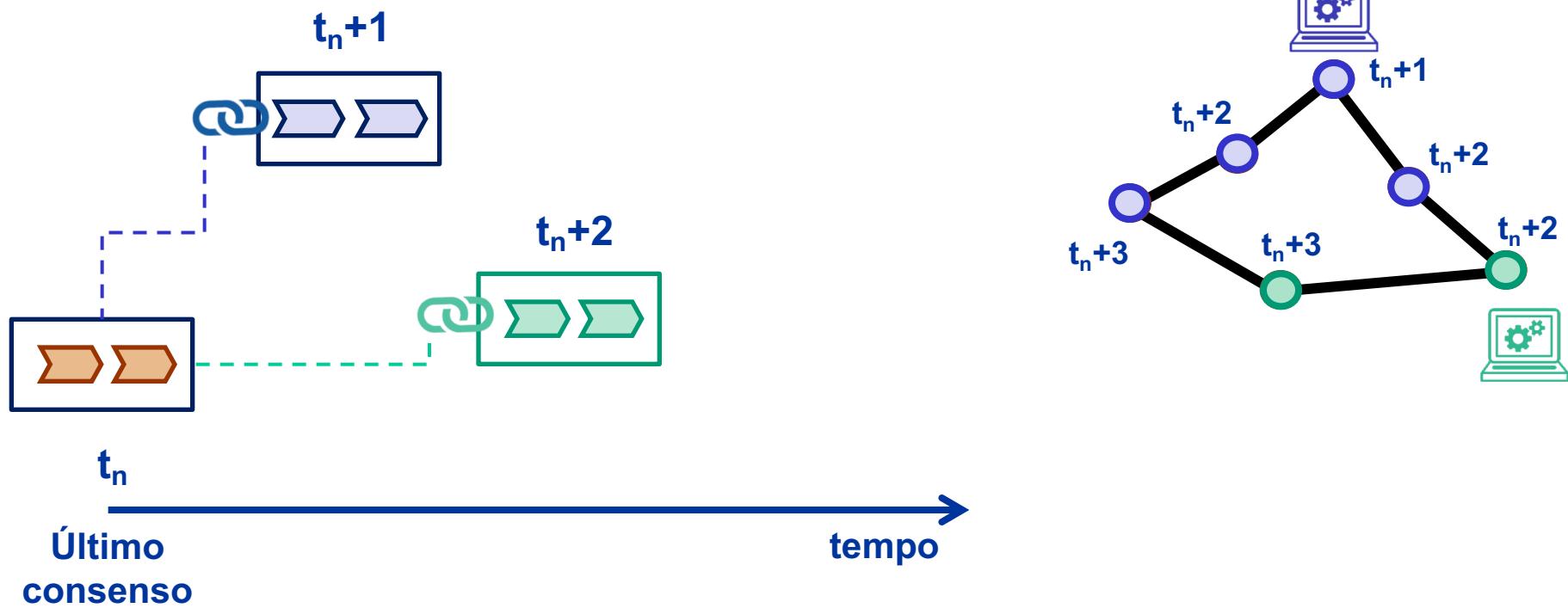
Blockchain: encadeamento de eventos

- Consenso: proof-of-work
 - O **primeiro que achar nonce**, faz broadcast para a rede toda
 - **Maior cadeia** recebida → consenso...



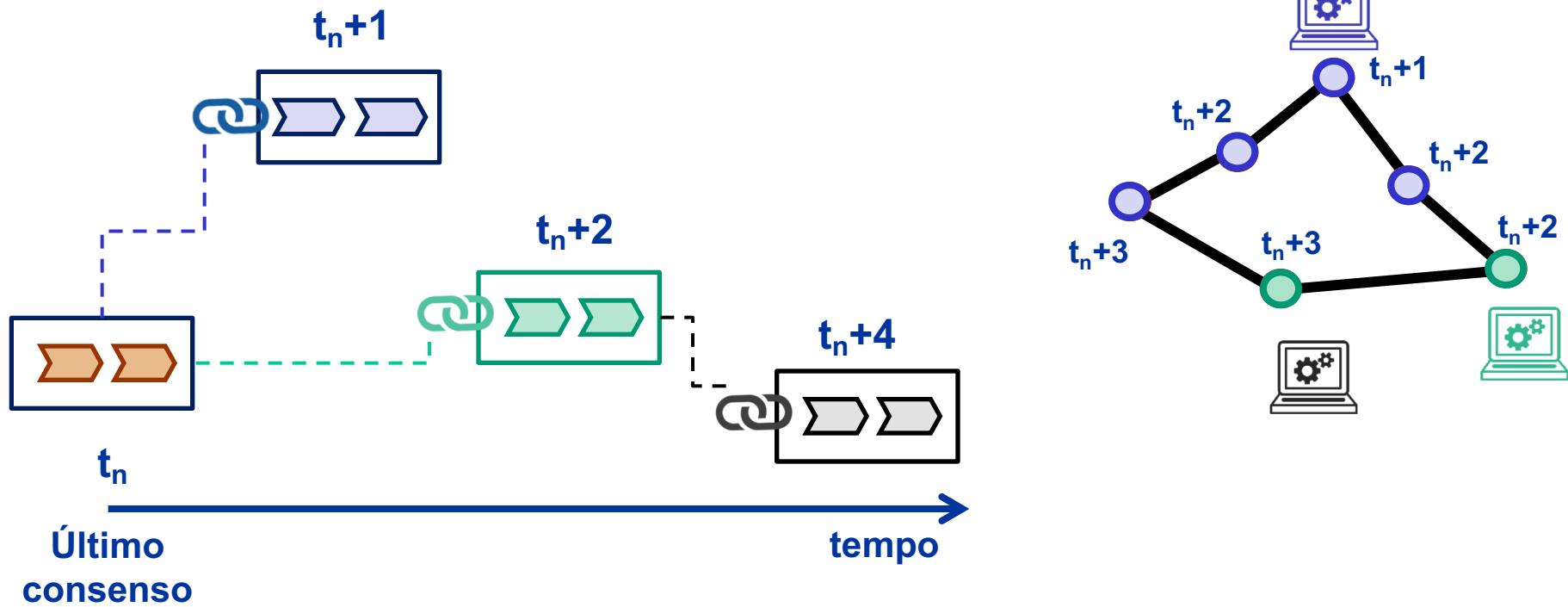
Blockchain: encadeamento de eventos

- Consenso: proof-of-work
 - O **primeiro que achar nonce**, faz broadcast para a rede toda
 - **Maior cadeia** recebida → consenso... nem sempre!!!



Blockchain: encadeamento de eventos

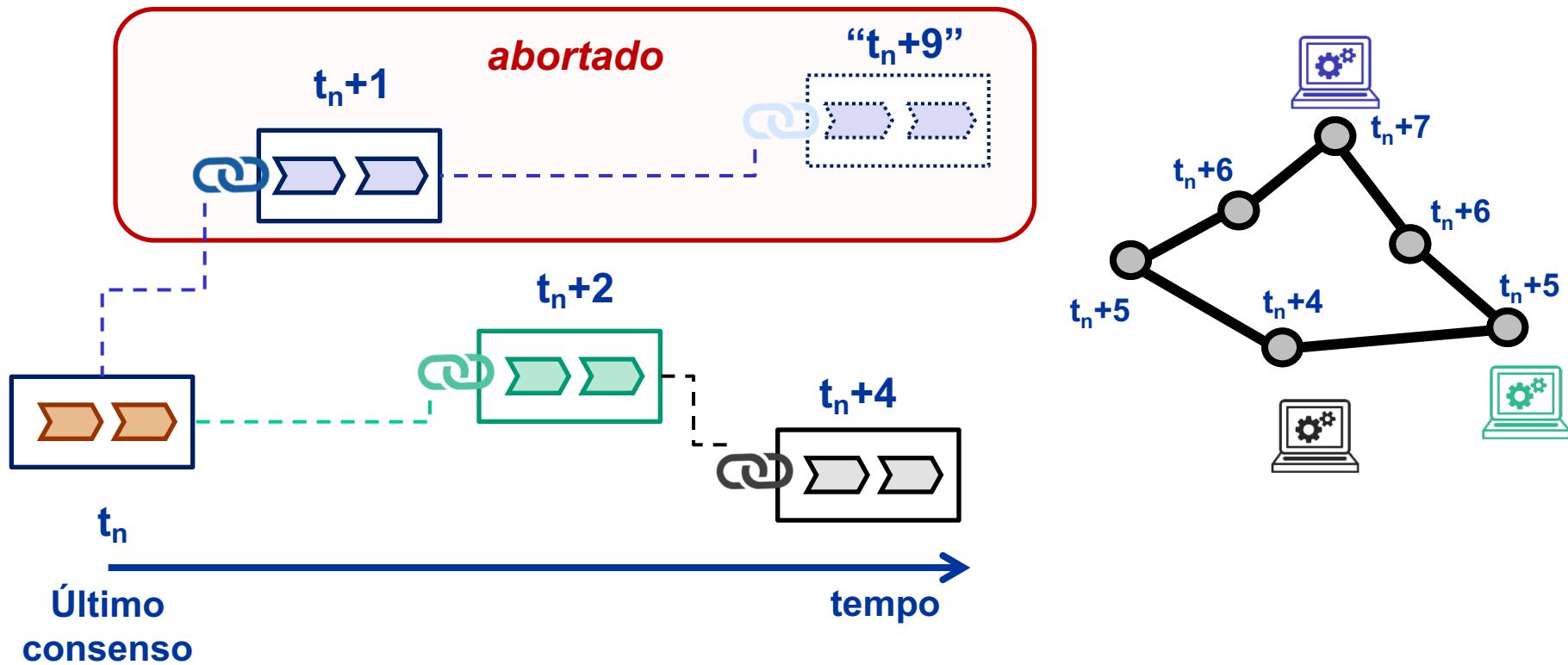
- Consenso: proof-of-work
 - Cadeia que **cresce mais rápido** ganha a “corrida do consenso”
 - Forks: duas visões da realidade (pré-consenso)



Blockchain: encadeamento de eventos

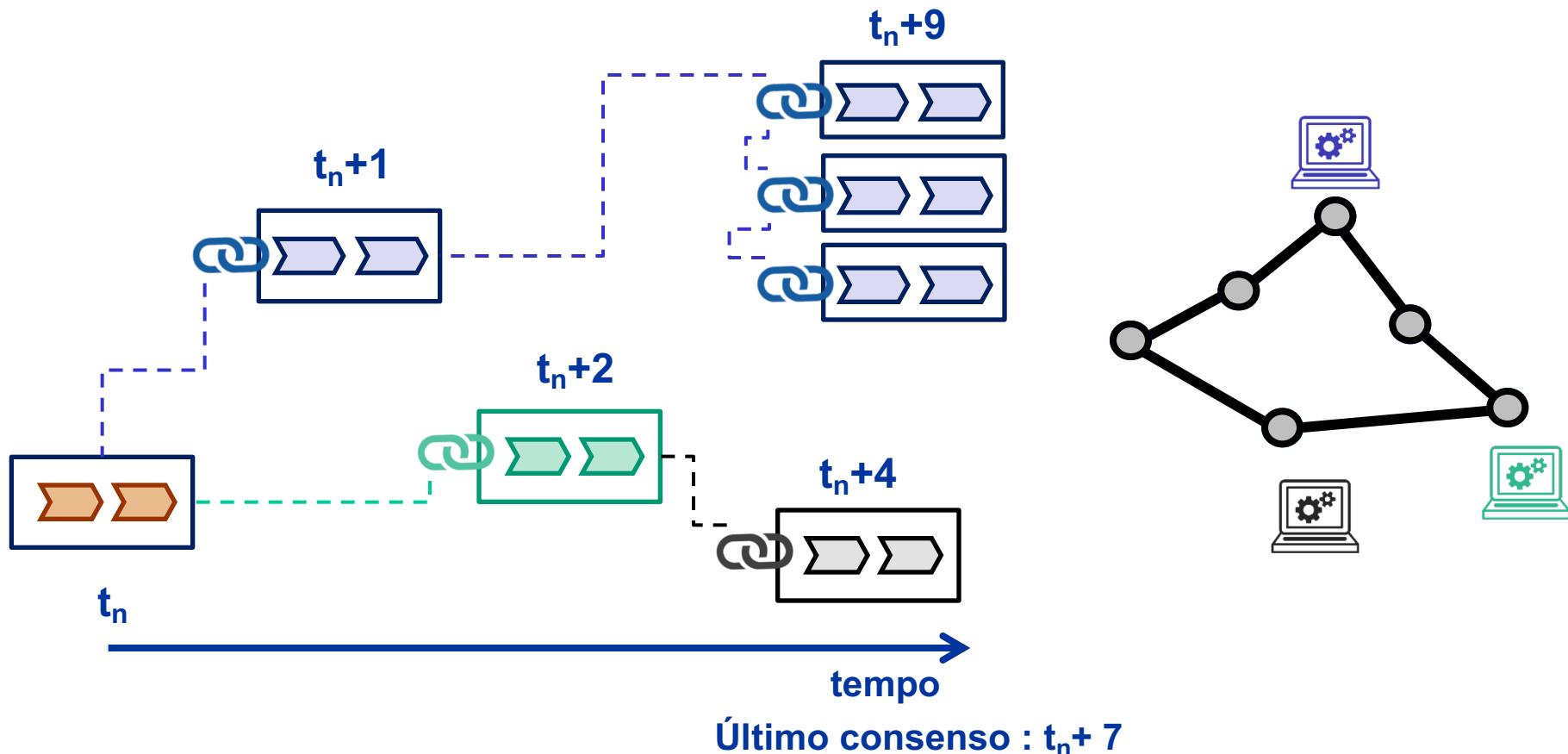
▫ Consenso: proof-of-work

- Cadeia que **cresce mais rápido** ganha a “corrida do consenso”
 - Forks: desaparecem à medida que consenso é atingido



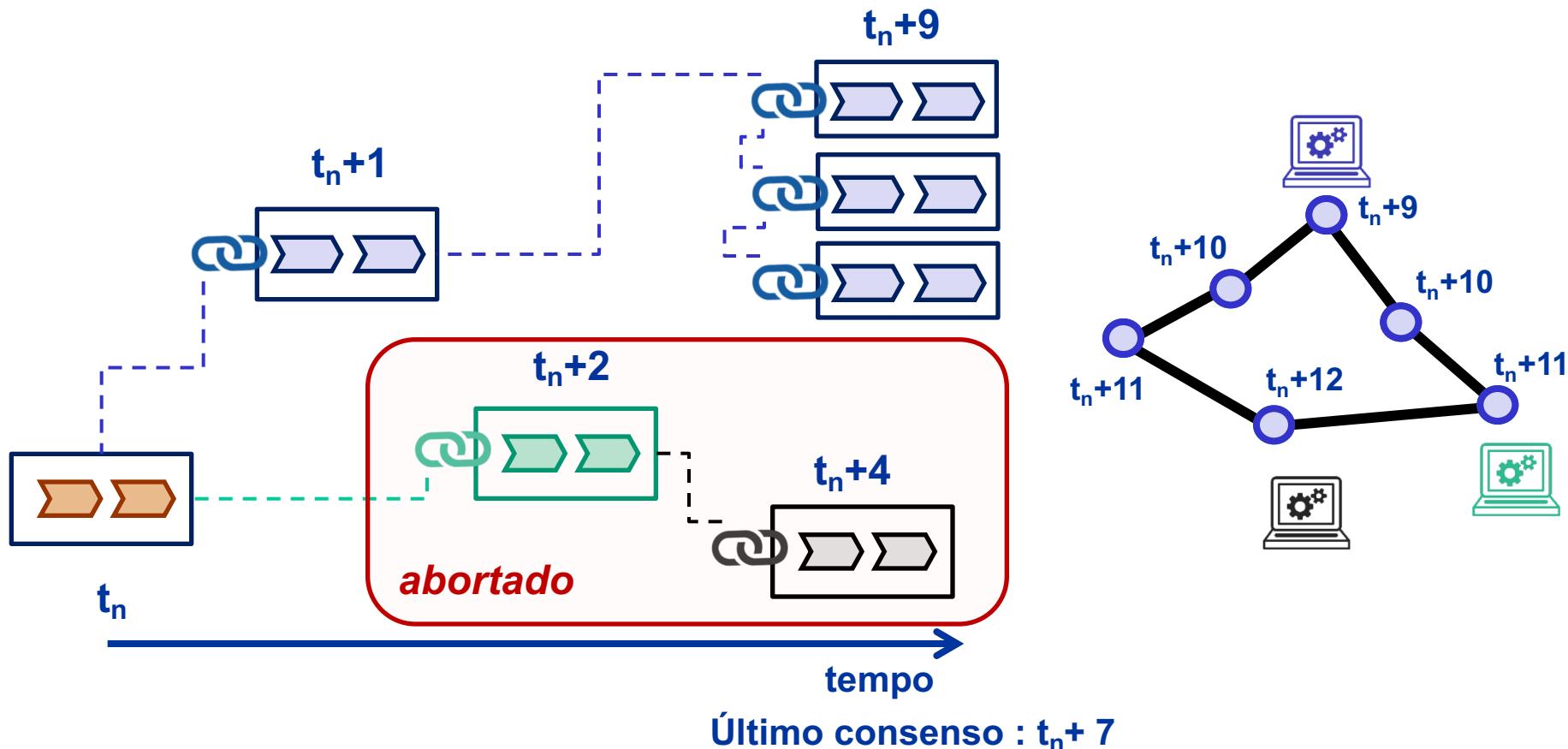
Blockchain: encadeamento de eventos

- ❑ Possível trapacear o consenso?
 - Se minha cadeia crescer muito rápido, posso apagar eventos!!!



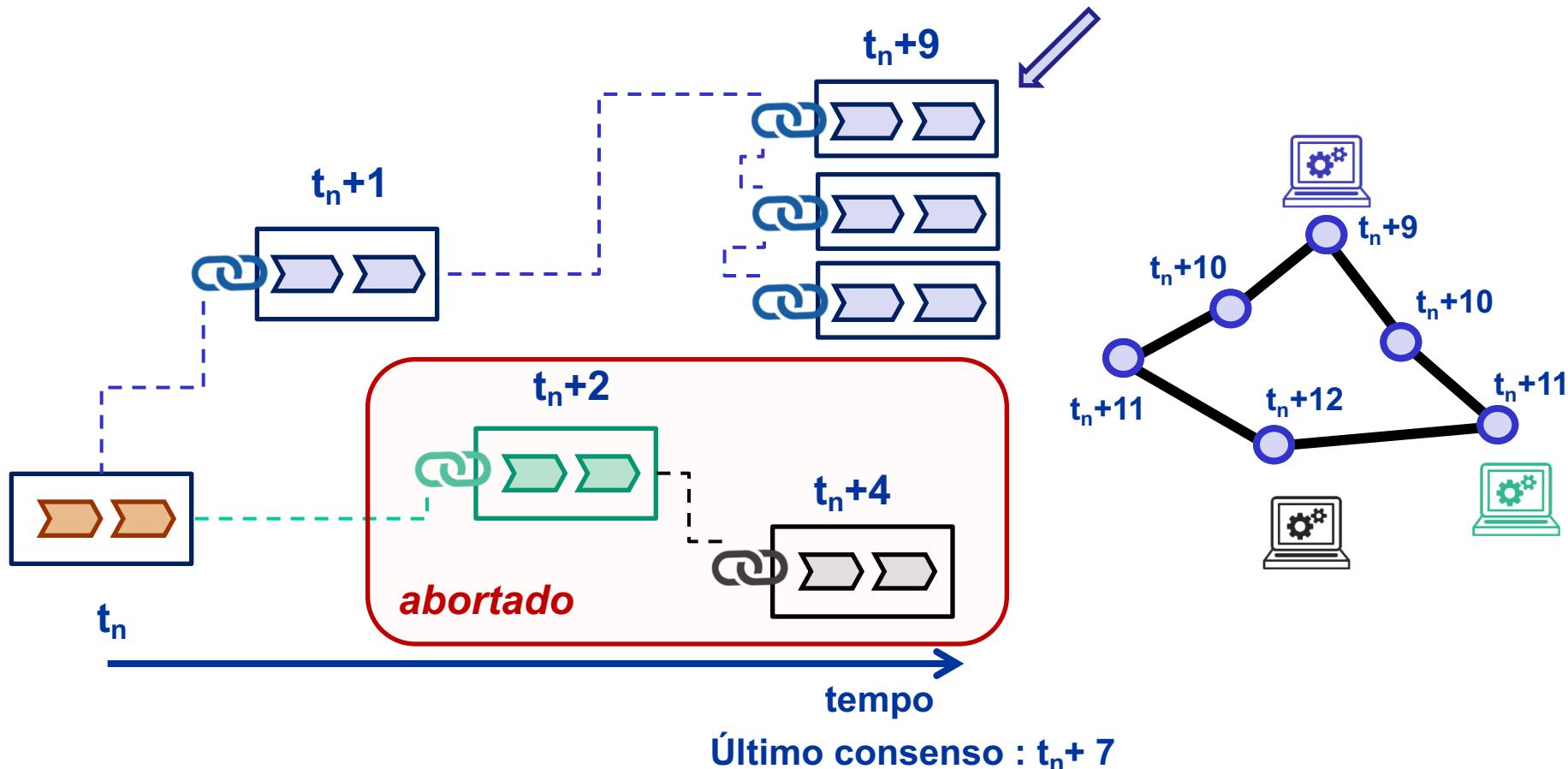
Blockchain: encadeamento de eventos

- ❑ Possível trapacear o consenso?
 - Se minha cadeia crescer muito rápido, posso apagar eventos!!!



Blockchain: encadeamento de eventos

- ❑ Possível trapacear o consenso?
 - Só com **poder computacional superior ao da rede toda...**



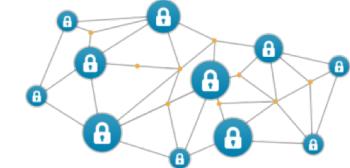


Blockchain: que tipo de eventos?

- Blockchain não faz validação dos eventos em si
 - Isso fica a cargo da camada de aplicação!
- Verificabilidade: depende do sistema...
 - No Bitcoin: evento = transferência de moeda de **A → B**
 - Verificar assinatura de **A** sobre evento
 - Verificar saldo de **A**: (1) **A** recebeu as moedas referenciadas na transferência?; (2) **A** ainda não usou essas moedas?
 - Obs.: nó mantém base separada com “moedas não gastos”
 - Outros cenários: pode ser complexo...
 - Assinatura dos envolvidos no evento costuma ser comum
 - Interação com o mundo real (e.g., execução de um serviço, ou entrega de um produto) pode exigir auditoria no mundo real



Blockchain: resumo

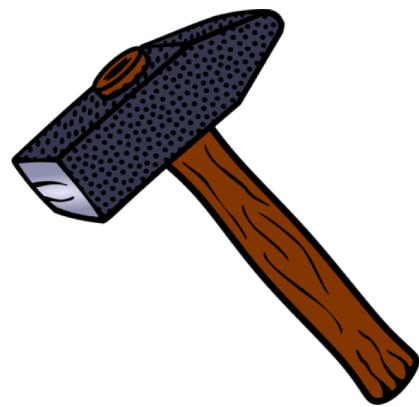


- Basicamente: mecanismo distribuído para ordenação de eventos com consenso entre as partes
 - **Ordenação:** um evento faz referência ao anterior
 - **Consenso:** proof-of-work
 - Obs.: existem outras técnicas do tipo “proof-of-something”
 - Processo **custoso computacionalmente e demorado:**
 - No Bitcoin, recomenda-se aguardar a validação de alguns (e.g., 8) blocos antes de aceitar as transações nele contidas como “parte do consenso”
 - Segurança requer resiliência a conluio
 - “Ataque dos 51%”: se um nó (ou um grupo de nós em conluio) for muito mais poderoso que o restante da rede



Considerações

“Se a única ferramenta que você tem é um martelo, tudo começa a parecer com um prego.” Abraham Maslow, 1966



Blockchain



**Problemas em
sistemas distribuídos**

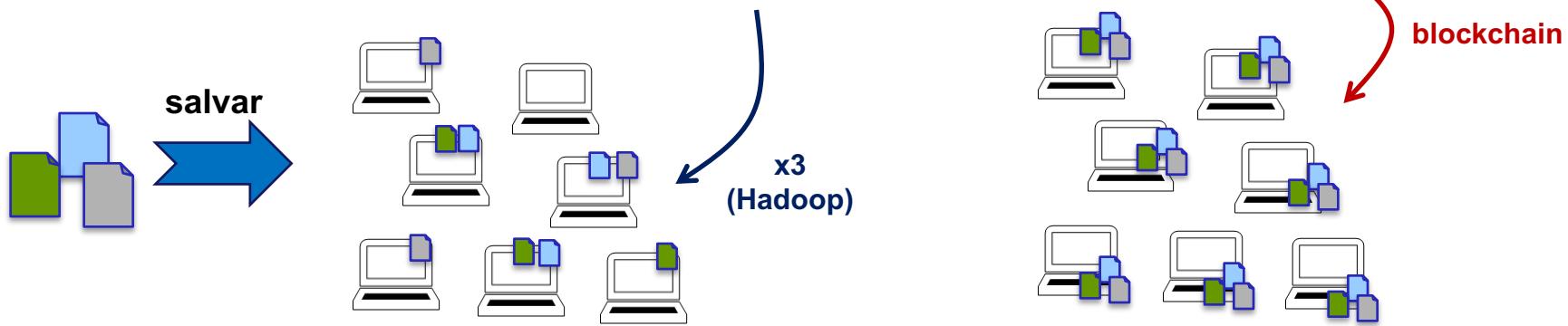
Considerações

- “Mas blockchain não é um jeito de armazenar dados de forma imutável?”
 - Não: isso é o que faz um hash seguro, pois não se pode alterar dados sem que se altere seu hash
 - Logo, embora um blockchain possa ser usado, é possível que usar apenas **hashes** seja suficiente (e mais eficiente...)
- “Mas blockchain não garante irretratabilidade de dados?”
 - Não: quem faz isso é uma **assinatura digital** (que pode, obviamente, ser associada a um blockchain)
- “Mas blockchain não garante veracidade dos dados colocados nele?”
 - Não, blockchain não faz mágica... verificação fica a cargo da **camada de aplicação** sobre o blockchain



Considerações

- “Mas blockchain não é uma espécie de sistema de arquivos distribuído, como na nuvem?”
 - Não: no blockchain, os dados são **100% replicados**, não **distribuídos com redundância**



- Se você precisa armazenar dados de forma distribuída, você está procurando outras tecnologias, não voltadas a ordenação em si

- P2P: Distributed Hash Table (DHT – ex: Kademlia), BitTorrent, Freenet, InterPlanetary File System (IPFS)
- Com controlador centralizado (“Big Data”): Hadoop

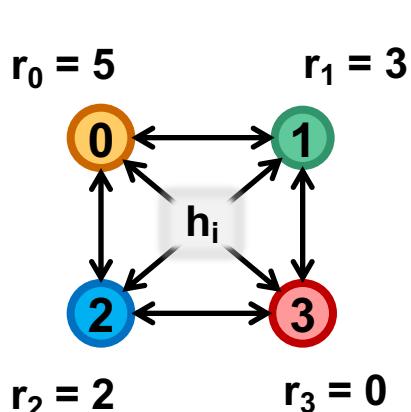
Considerações

- “Mas blockchain não é uma solução leve e eficiente?”
 - Não: protocolos de consenso distribuído dificilmente são computacionalmente eficientes...
 - E quando o são, geram muitos forks: diversas visões da realidade geradas rapidamente dificultam consenso...
 - **Nota 1:** pode ser mais eficiente que sistemas físicos alternativos
 - Razão: cartórios e transferência entre bancos **são ineficientes.**
 - **Nota 2:** certas **simplificações** podem criar consenso eficiente
 - Ex.: “commit-and-reveal” permite **sorteio justo** se nós são conhecidos e confia-se na sua participação (e.g., incentivo é claro)

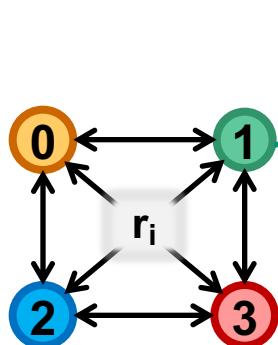


Considerações

- “Mas blockchain não é uma solução leve e eficiente?”
 - Ex.: “Commit-and-reveal” para **sorteio justo** entre **n = 4** nós
 - Requisito:** todos os nós da rede são conhecidos!
 - 1. Commit:** nó N_i gera aleatório r_i e faz broadcast de $h_i = \text{Hash}(r_i)$
 - 2. Reveal:** após receber todos os $h_{j \neq i}$, N_i faz broadcast de r_i
 - 3. Sorteio:** nó sorteado é N_k , onde $k = \text{Soma}(r_i) \bmod n$



1. Commit: $h_i = \text{Hash}(r_i)$



2. Reveal: broadcast de r_i

$r_1 = 3$
 $r_0 = 5 \rightarrow \text{Hash}(5) = h_0?$
 $r_2 = 2 \rightarrow \text{Hash}(2) = h_2?$
 $r_3 = 0 \rightarrow \text{Hash}(0) = h_3?$

Valor de r_i não pode ser alterado: invalidaria hash!

0 1
2 3
Consenso: nó 2

3. Sorteio: $10 \bmod 4 = 2$



Considerações

- ❑ “Isso não é um olhar um tanto pessimista/limitado?”
- ❑ Não exatamente: é somente um **olhar crítico!**
 - Existem aplicações nas quais Blockchain é muito útil...
 - ... assim como existem diversos “vendedores de blockchain” que vendem ilusões
- ❑ Toda tecnologia tem um período de “**hype**”
 - Nem tudo é resolvido movendo seus dados/aplicações para uma **nuvem pública** (e.g., **segurança de aplicações**)
 - Nem tudo se resume a **Big Data** sobre dados não estruturados (e.g., sistemas transacionais se beneficiam muito de **dados estruturados** em bancos relacionais)
- ❑ **Evite armadilhas: tenha um olhar crítico!**



Blockchain: quando usar?

- Lembre-se do que foi colocado no início:
 - Blockchain é **útil** em um cenário: **ordenação de eventos** em sistema **totalmente distribuído** e **sem confiança** entre as partes, mas há **incentivo** para sua participação no sistema
 - Obs.: “Incentivo” nem sempre é essencial, mas costuma ser interessante -- por que usuários irão participar do processo de consenso, essencial para o funcionamento do sistema?
 - Blockchain é **pouco (ou nada) interessante** quando:
 - Há uma **entidade confiável** (e.g., um banco); ou
 - É **desnecessário ordenar** eventos (e.g., basta a existência); ou
 - Apenas a ordenação de eventos não é suficiente (e.g., são necessários **instantes exatos de tempo**)

Blockchain: quando usar?



- ❑ Blockchain substitui **TSA**, com abordagem distribuída
 - Logo: se uma solução para o problema for uma TSA, então Blockchain também é uma potencial solução
- ❑ Sugestão de procedimento:
 - Defina bem o seu **problema**
 - Ex.: “preciso de uma base ordenada de eventos para verificar se houve duplicação ou supressão de algum evento”
 - Identifique os **requisitos** que justificam um blockchain
 - Modele uma solução **usando TSA**:
 - Ex.: TSA pode assinar ID dos eventos com seu timestamp, e publicar resultados em banco de dados aberto -- todos os interessados podem replicar (parte do) banco
 - Substitua a TSA pelo **Blockchain**
 - Preferencialmente, defina **incentivo** para participação no consenso (“pelo bem do sistema” pode ou não ser suficiente)

Blockchain: aplicações potenciais



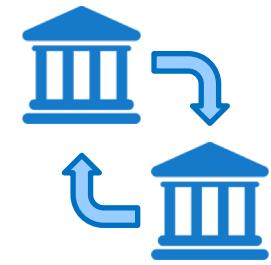
▫ **Killer application:** transferência de ativos

- **Ordenação de eventos:** necessário para saber quem é o dono atual do ativo
 - Não são necessários **instantes exatos de tempo**: não importa se o ativo foi recebido há anos ou há segundos, mas apenas quem é o dono atual
- Sistema **totalmente distribuído**: sem intermediários
 - Eliminar **entidade confiável** permite eliminar taxas correspondentes
- **Sem confiança**: usuários podem tentar cometer fraudes
 - Consenso previne que a “visão de mundo” fraudulenta prevaleça: apenas uma instância do ativo no sistema
- Ideia por trás do **Bitcoin** e outras **criptomoedas**
 - **Incentivo**: moedas e “pelo bem da economia”

Blockchain: aplicações potenciais



- Variantes de **transferência de ativos**:
 - **Contratos** diversos, como posse ou promessa de dívida
 - Substituição de cartório físico, que tem estrutura (semi-)centralizada
 - **Importante:** eventos no blockchain não podem ser confrontados com eventos registrados fora do blockchain (e.g., em cartórios físicos) → ordenação exigiria **instantes exatos de tempo**
 - Troca de **itens colecionáveis**, físicos ou reais
 - Ex.: figurinhas, cartas em jogos de cartas colecionáveis
 - Comumente implementado com arquiteturas centralizadas controladas pelo gerador dos itens (**entidade confiável**)
 - Ex.: empresa responsável pelo jogo ou pela marca
 - Mas pode haver **redução de custos** com arquitetura distribuída
- **Desafio:** como convencer usuários a investir recursos computacionais no processo de consenso ("**incentivo**")?



Blockchain: aplicações potenciais

- Variantes de transferência de ativos: **Câmara Interbancária de Pagamentos (CIP)**
 - Verifica transferência entre bancos, entre outras transações
 - Requer **ordenação de eventos**: como qualquer ativo
 - CIP é centralizada, mas acionistas são os bancos: pode ser feita de forma **totalmente distribuída**, pelos bancos
 - Mas informação passa a ser replicada para todos: requer solução para **confidencialidade** de transações
 - Mas bancos não são considerados **entidades confiáveis?**
 - Em 2010, o Banco PanAmericano foi indiciado por vender ativos duplicados... é um sistema **sem confiança**
 - Mas o Banco Central não seria uma **entidade confiável?**
 - Sim, mas em princípio não tem interesse de intermediar todas as transações → atuação como **auditor** apenas
 - **Incentivo**: “pelo bem do sistema”

Blockchain: aplicações potenciais



❑ Auditoria: cotação/histórico de preços

- Ao registrar a compra de produto da empresa X, esse era o preço mais baixo?
- A “Black Friday” da empresa X foi uma “Black Fraude”?
- **Sem confiança:** nós poderiam omitir cotações
 - “Quero comprar mais caro de quem me paga propina”
 - “Tomara que ninguém tenha notado que os preços atuais são basicamente iguais ao do meu histórico”
- **Ordenação de eventos:** registros de preços
 - **Instantes exatos de tempo**, embora potencialmente úteis, não são essenciais:
 - Cotação: basta saber o preço no momento do registro da compra (nó tem que ter preço mais baixo nesse ponto...)
 - Histórico: “já teve preços mais baixos” no passado



Blockchain: aplicações revolucionárias*

(só que não)



- Coleta de assinaturas para projetos de lei de iniciativa popular
 - Não tem “**ordem de eventos**” nesse cenário → Solução: bastam as assinaturas digitais correspondentes
- Votação eletrônica
 - Normalmente também não há “**ordem de eventos**” nesse cenário → Solução: bastam as assinaturas digitais correspondentes.
 - Obs.: se forem urnas físicas, elas podem gerar as assinaturas; se for votação online, há processos para garantir assinaturas anônimas, preservando o sigilo do voto (e.g.: Helios).



*obs.: propostas reais...

Blockchain: aplicações revolucionárias*

(só que não)



- Armazenamento de contratos em servidor com carimbo de tempo, criando “cartório digital publicamente verificável”
 - O carimbo de tempo feito por **entidade confiável** dispensa o blockchain → Solução: assinaturas digitais com carimbos de tempo do cartório e replicação de dados **ou** blockchain
- Permitir que banco reduza custos: deixa de guardar transações de seus clientes, mas permanece como intermediário (entidade central confiável)
 - Blockchain é útil só se banco deixar de ser **entidade confiável** no sistema (e, potencialmente, de cobrar taxas pelas transações...), como no próprio Bitcoin



*obs.: propostas reais...

Blockchain: aplicações revolucionárias*

(só que não)



- Criar “universidade Blockchain com um token nativo”
 - Não que seja fácil entender o que isso quer dizer, mas...
 - Característica 1: “blockchains e smart contracts para garantir as relações entre estudantes e educadores”
 - Se for uma relação temporal, pode fazer sentido.
 - Se for apenas “estudante X fez curso Y”, bastam assinaturas digitais considerando os educadores **entidades confiáveis**.
 - Se educadores não forem confiáveis para assinar, blockchain não gera essa confiança...
 - Característica 2: “blockchains com smart contracts podem automatizar processos administrativos usando o token da universidade, reduzindo os custos”
 - Reduzir **custos** usando consenso distribuído...? Difícil...

*obs.: propostas reais...

Blockchain: aplicações revolucionárias*

(só que não)



- Manter arquivos de log das ações de um controlador SDN, para posterior auditoria dessas ações:
 - SDN: *Software Defined Network*, ou Rede Definida por Software
 - Tecnologia que permite maior programabilidade de equipamentos de rede, como switches e roteadores
 - Controlador é **entidade confiável** central em redes SDN: pode realizar ações e não registrar no blockchain → Solução: switches armazenam requisições assinadas pelo controlador
 - Obs.: não previne ações indevidas pelos switches se elas forem realizadas localmente, mas ao menos permite verificar comandos do controlador



*obs.: propostas reais...

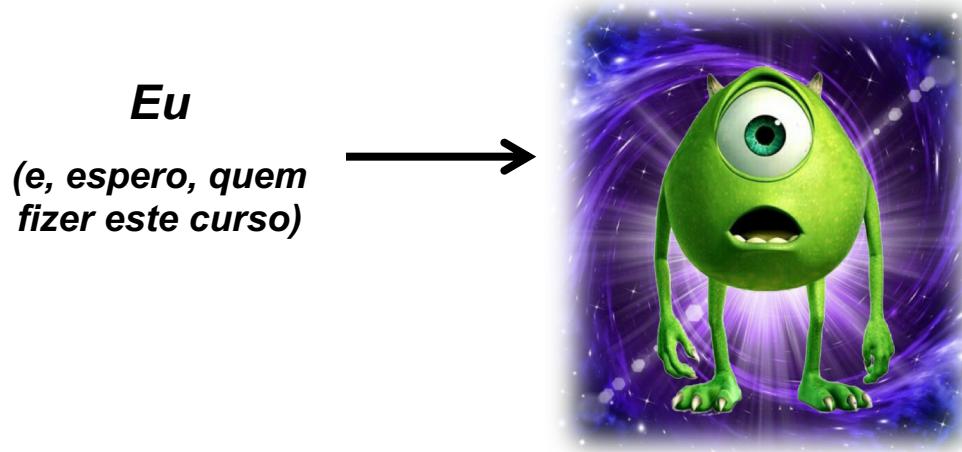
Blockchain: aplicações revolucionárias*

(só que não)



- Alguma que vocês já tenham visto?
 - Agradeço contribuições para expandir esta lista!
- Se nunca viram algo assim, recomendo ler:
 - <https://medium.com/@mehmettoral/blockchain-as-the-cure-for-cancer-or-how-a-hammer-was-mistaken-for-a-painting-643741abf972>

Conclusão: “sem o hype”



Entusiastas de Blockchain

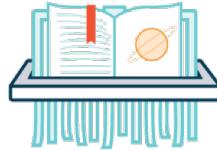


Sistemas de Arquivos

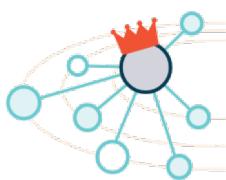
IPFS

GFS (Hadoop)

Problemas com o HTTP



- ❑ Centralizada em servidores, que podem ser desligados
 - Conteúdo acaba sendo **perdido**, proposital ou accidentalmente
- ❑ Cria dependência de alguns serviços essenciais
 - Buscas: Google pode controlar o que usuários encontram
 - Hosting: Facebook/Amazon pode controlar o que usuários armazenam/veem
 - Resultado: censura, espionagem, direcionamento de opiniões, possibilidade de interrupção de serviços...
- ❑ Ineficiente: servidor de conteúdo popular torna-se gargalo
 - Escolha entre lentidão ou contratação serviços de caching...
- ❑ Voltado a locais, não a conteúdos:
 - Links quebram se local for alterado (404 Not Found)



IPFS: InterPlanetary File System



- ❑ Projeto criado em 2015, por Juan Benet
 - Um sistema de arquivos versionado, distribuído globalmente
 - “Uma web permanente e distribuída”: similar a um enorme swarm bittorrent para troca de objetos versionados
 - Permite a construção de sites sem um servidor correspondente: “servidor” distribuído na rede!
- ❑ Links:
 - Página oficial: <https://ipfs.io/>
 - White paper:
<https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf>
 - Um pequeno experimento: <https://linuxroot1.github.io/IPFS/>

IPFS: Arquitetura

- Combina diferentes tecnologias



Fonte: <https://github.com/ipfs/specs/tree/master/architecture>

IPFS: Arquitetura (roteamento)



- ❑ Roteamento descentralizado, via **DHT**
 - Se dados pequenos (<1KB): dados armazenados na DHT;
 - Caso contrário, DHT armazena referência para dados (IPs de nós que podem fornecer dados)
- ❑ Algoritmos:
 - **S/Kademlia:** estrutura em árvore para busca eficiente
 - Vide **apêndice**
 - **Coral:** considera **localidade** dos dados para melhorar eficiência das buscas
 - Clusters organizados por região e tamanho
 - “Busca(chave)” retorna subconjunto de IPs que têm o conteúdo em vez de lista completa (“Sloppy DHT”)

IPFS: Arquitetura (troca de arquivos)

- Baseado no **BitSwap**

- similar a **BitTorrent**, mas com **um único swarm** com todos os conteúdos

- Mecanismos de **incentivo**:

- **Tit-for-tat**: pode obrigar nó A a buscar pedaços que nó B deseja para que então possa receber pedaços vindos de B
 - Nó A também lista de “**credores**” (nós que mandaram mais dados para A do que receberam de A) e tenta pagar débito
 - Também pode usar ***distributed ledger*** para gerenciar créditos



IPFS: Arquitetura (versionamento – GIT)

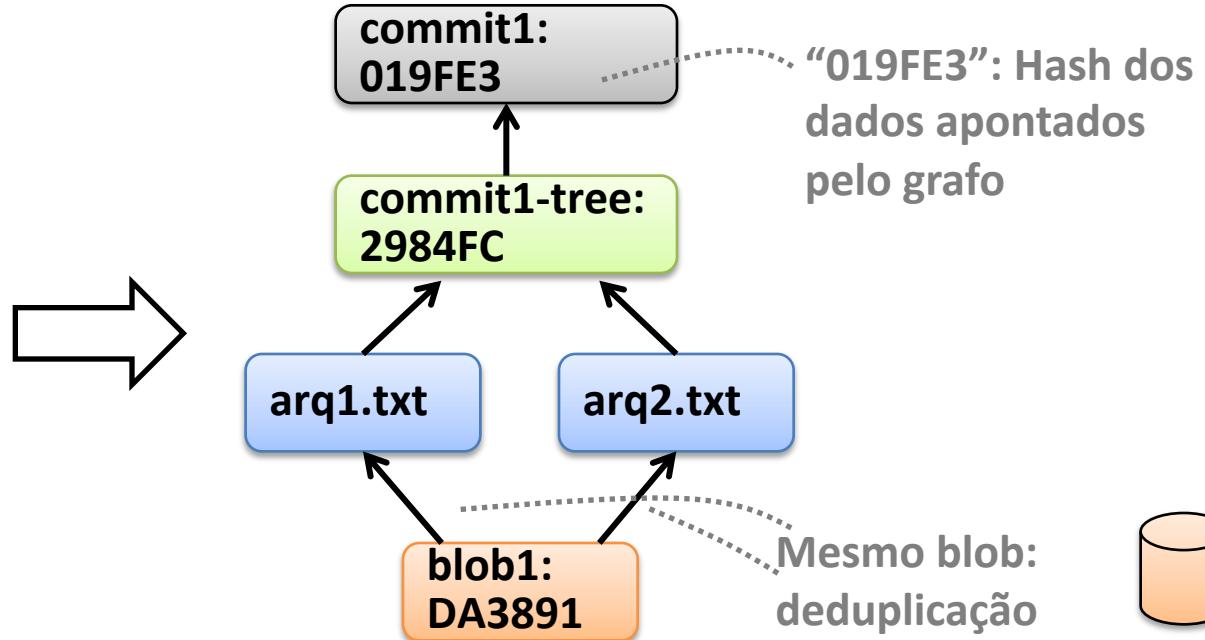
- Usa Merkle ~~tree~~ DAG (Grafo Direcionado Acíclico)



IPFS: Arquitetura (versionamento – GIT)

- ❑ Usa Merkle DAG (Grafo Direcionado Acíclico)
 - Commit: autor, mensagem, ponteiro (hash) para uma árvore
 - Tree: ponteiro para árvores e arquivos (estrutura de pastas)
 - Blob: dados

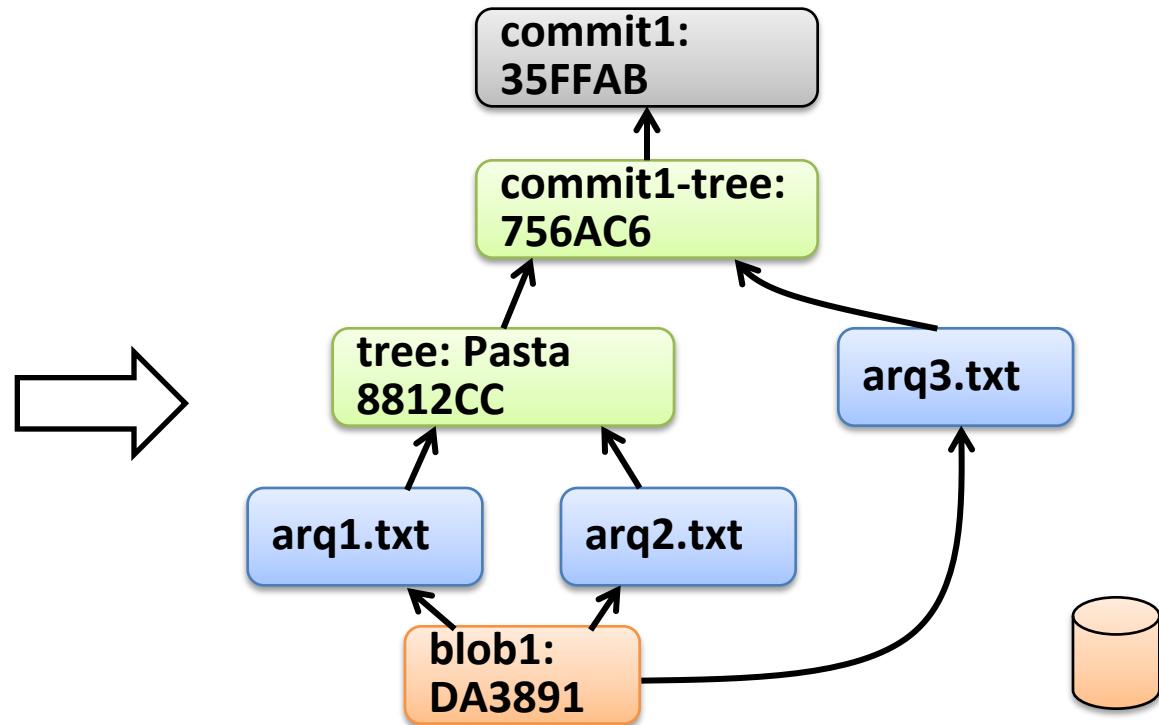
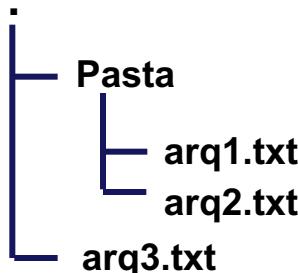
Ex.: dois arquivos idênticos na raíz



IPFS: Arquitetura (versionamento – GIT)

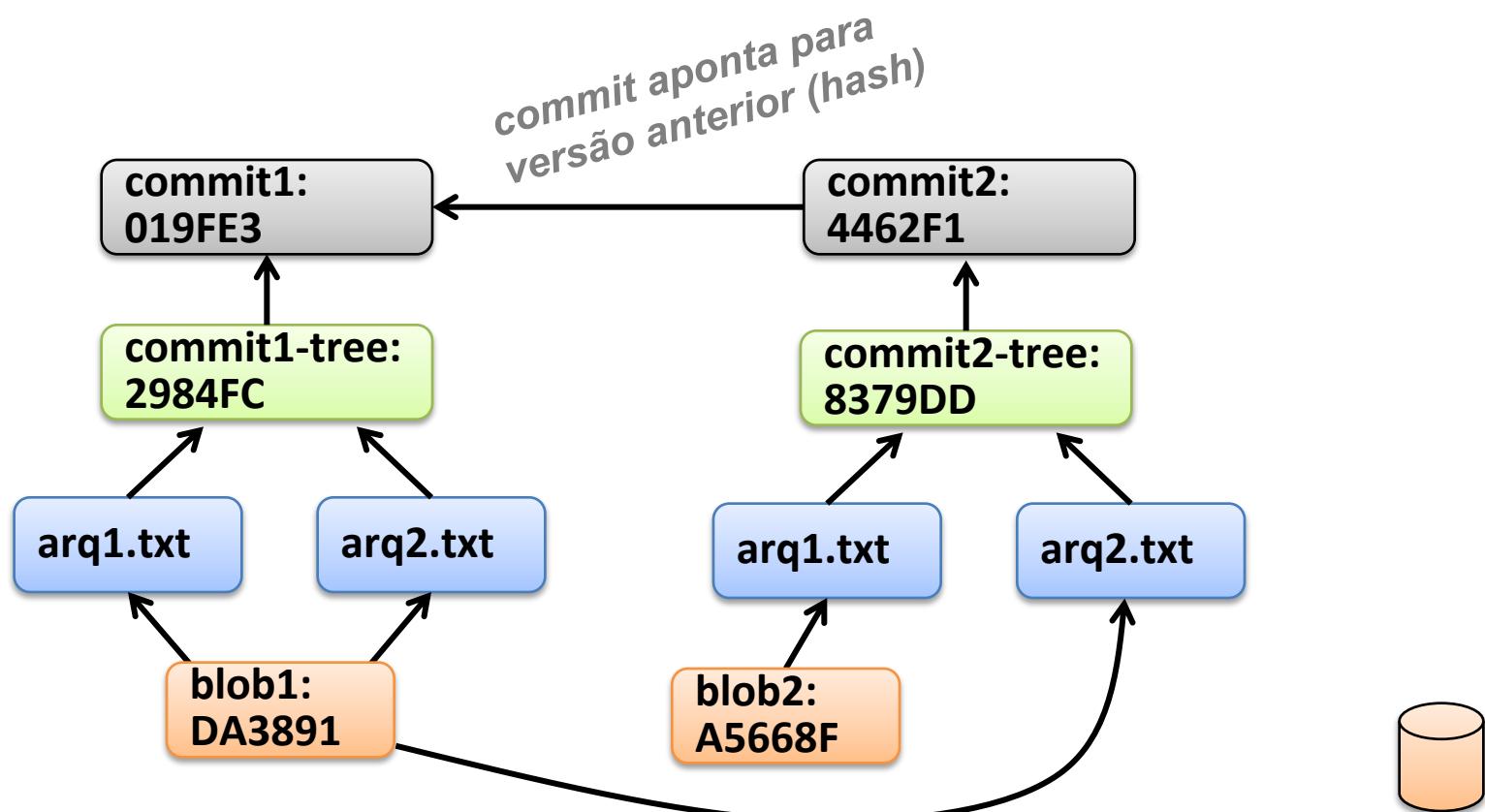
- ❑ Usa Merkle DAG (Grafo Direcionado Acíclico)
 - Commit: autor, mensagem, ponteiro (hash) para uma árvore
 - Tree: ponteiro para árvores e arquivos (estrutura de pastas)
 - Blob: dados

Ex.: três arquivos idênticos, em pastas



IPFS: Arquitetura (versionamento – GIT)

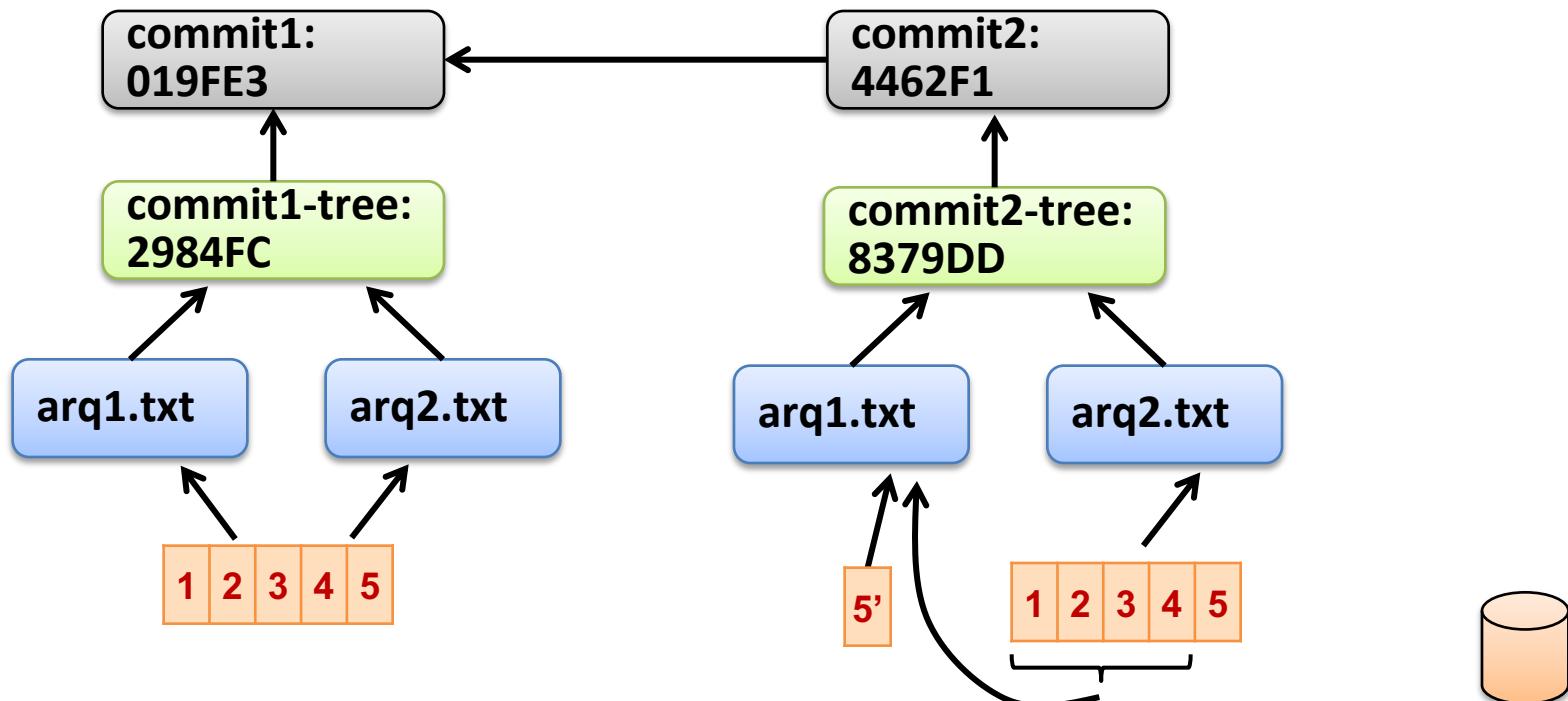
- Controle de versão via hashes
 - Permite acompanhar o caminho das alterações dos arquivos



IPFS: Arquitetura (versionamento – GIT)

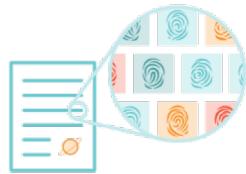
Controle de versão via hashes

- Permite acompanhar o caminho das alterações dos arquivos
- Se blobs quebrados em blocos: **deduplicação** mais efetiva



IPFS: Arquitetura (versionamento – GIT)

- **Objetos** no IPFS: todos identificados pelo seu hash, tanto se forem **arquivos** ou **links**
 - **Multihash** para suporte a diferentes algoritmos
 - Formato: <Algoritmo><Tamanho_Hash><Bytes_Hash>
 - Navegação entre links de um domínio = navegação no Merkle DAG, usando o hash de cada link



Formato: /ipfs/<hash-of-object>/<name-path-to-object>
Ex.: /ipfs/XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x/arq.txt

- Para acessar o arquivo “fig.png” localizado no caminho “<dominio>/pasta/fig.png, pode-se usar qualquer das opções:
 - 1) Domínio: /ipfs/<hash-de-dominio>/pasta/fig.png
 - 2) Pasta: /ipfs/<hash-de-pasta>/fig.png
 - 3) Arquivo: /ipfs/<hash-de-fig.png>

IPFS: Arquitetura (versionamento – GIT)

- Qualquer usuário pode publicar objetos na rede
 - Basta incluir hash do objeto na DHT, se declarar como um peer para objeto e publicar caminho para objeto
- Uso de Merkle DAGs: hash não pode ser alterado, logo **objetos são permanentes!**
 - Redução do consumo de banda: **caching**
 - Conteúdo servido por nós **sem confiança**: análogo ao que ocorre no BitTorrent
 - **Links são permanentes**: sem “links quebrados” desde que alguém tenha arquivo
 - Usuários podem escolher fazer backups de **dados específicos** para garantir sua **longevidade**



IPFS: Arquitetura (sistema de nomes)

▫ SFS (Self Certified Filesystem)



- Identificadores dos nós (NodeId) correspondem ao hash de suas chaves públicas
 - NodeID usado para roteamento de buscas
 - Dificulta “escolha do ID” para eventuais ataques
- Totalmente descentralizado: basta servidor provar posse da chave privada para verificar corretude
 - Permite troca de chaves para estabelecer canais seguros
 - Similar a HTTPS, mas sem depender de certificados digitais emitidos por Autoridade Certificadora



Formato: /ipns/<NodeID>
Ex.: /ipns/hj17rsy89MnOo

IPFS: Arquitetura (sistema de nomes)

- IPNS (Name Space): **mutabilidade** de objetos
 - Nó pode associar seu domínio /ipns/<NodeId> a um objeto
 - Busca na DHT por domínio retorna versão atual do objeto (seu hash), que pode ser modificado quando desejado
 - Assinatura do servidor sobre objeto garante autenticidade
 - Objeto pode ser commit de página web completa: carrega histórico de versões!



mesma
chave



NodeId	Valor
/ipns/hj17rsy89MnOo	98sdfHjyu87q
/ipns/hj17rsy89MnOo	3dfvJ3KdgYr

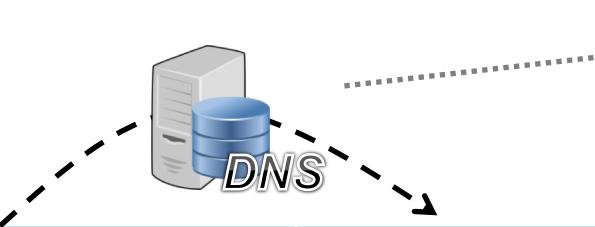


conteúdos
distintos



IPFS: Arquitetura (sistema de nomes)

- IPNS (Name Space): mutabilidade de objetos
 - Facilita nomes amigáveis a humanos, no lugar de hashes
 - Ex.: criar registro DNS (e.g., blockchaindev.blog) p/ nodeID
 - blockchaindev.blog: "dnslink= /ipns/hj17rsy89MnOo"



DNS	NodeID	Valor
/ipns/blockchaindev.blog	/ipns/hj17rsy89MnOo	98sdfHjyu87q 
/ipns/blockchaindev.blog	/ipns/hj17rsy89MnOo	3dfvJ3KdgYr 

Leitura: <https://decentralized.blog/ten-terrible-attempts-to-make-ipfs-human-friendly.html>

IPFS: Teste você mesmo!

- Solução ainda razoavelmente experimental
 - Mas “mão na massa” ajuda a entender funcionamento!



<https://ipfs.io/docs/getting-started/>

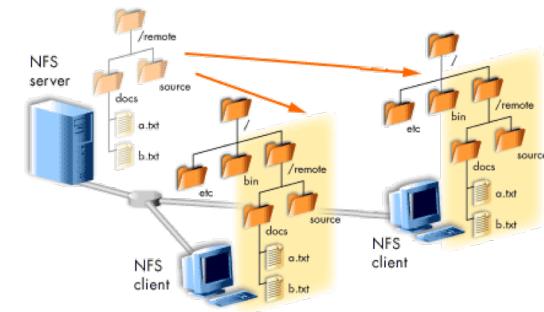
“Big Data”: Google File System (GFS)

- ❑ **Big Data** requer **sistema de arquivos** adequado
 - Distribuição de dados, tolerância a falhas, redundância ...
- ❑ Para atacar problema, o Google desenvolveu o **GFS**
 - Projetado para necessidades e cargas de trabalho do Google
 - Linguagens de programação principais: C++ (base) ou Sawzall (otimizada para MapReduce)
 - Sawzall: código MapReduce ~10x menor que equivalente em C++
 - **Gerenciamento** automático de **tarefas MapReduce** por ferramenta Workqueue



“Big Data”: Google File System (GFS)

- ❑ Por que não usar sistemas de arquivos existentes?
 - “**Big Data**”: problemas neste cenário **são diferentes** dos encontrados no compartilhamento de arquivos por usuários
 - Logo, bom **desempenho** requer algumas “personalizações”
- ❑ Por exemplo: Network File System (**NFS**)
 - Sistema de arquivos de **propósito geral**
 - Cenário de uso comum: compartilhar arquivos entre vários usuários
 - **Leituras/escritas em pontos aleatórios**
 - **Muitos arquivos, em geral pequenos**
 - **Sem suporte a redundância** (provida externamente, se preciso)



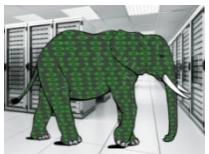
“Big Data”: Cenário do Google/GFS

- Alta taxa de **falhas**



- Grande número de componentes de hardware baratos (ex.: discos magnéticos), que comumente acabam falhando

- **Número modesto de arquivos gigantescos**



- Alguns milhões de arquivos tendo 100 MB ou mais (arquivos de GB são comuns)

- **Acesso** aos arquivos é “**atípico**”



- Múltiplas **leituras sequenciais**, seguidas de operações de **anexar** ao final do arquivo (ex.: MapReduce!)

- **Leituras curtas e aleatórias** ocorrem, mas **são raras**

- **Alta vazão** é mais importante do que baixa latência



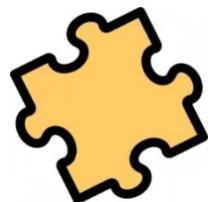
“Big Data”: Projeto do GFS

- Arquivos armazenados em “**chunks**” grandes

- Tamanho fixo: **64 MB**

- 64TB de dados → 1 milhão de chunks
 - Isso **facilita leitura sequencial**

- No NFS: o tamanho típico é **2k-8k**
 - **Tabela de arquivos grande**, armazenada em disco: menor desempenho para ler grandes volumes de dados



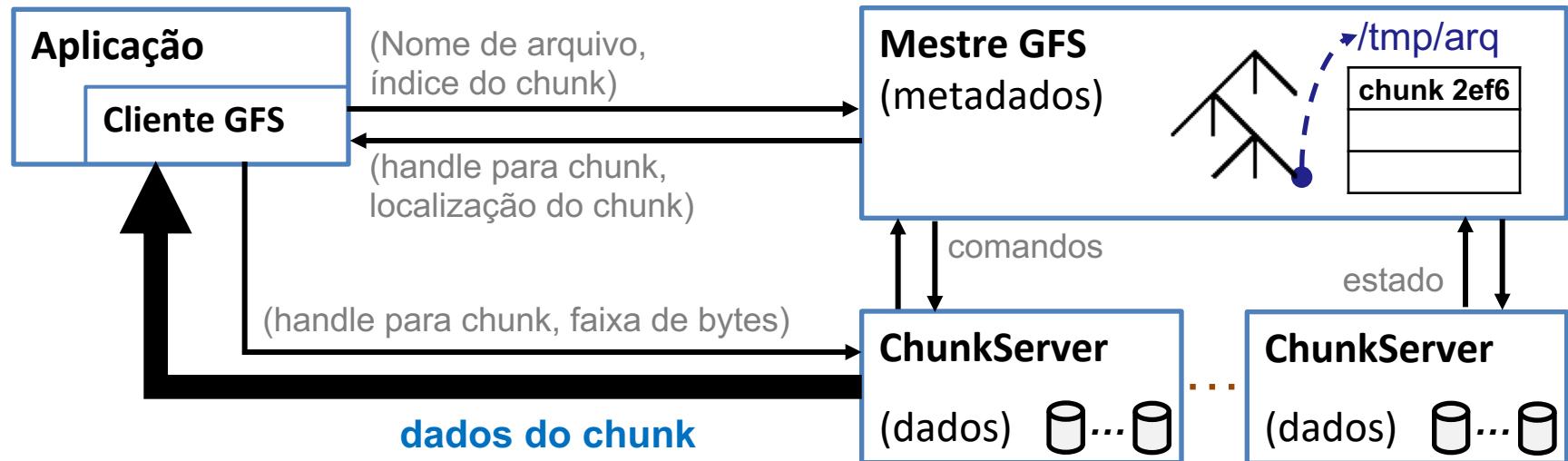
- **Sem caching** no cliente

- Traria poucas vantagens dado o tamanho dos dados e leitura sequencial
 - No NFS: sincronização entre caches locais nos clientes aumenta congestionamento na rede.



“Big Data”: Projeto do GFS (cont.)

- Confiabilidade de dados via **replicação de chunks**
 - Cada chunk armazenado em **3 ou mais ChunkServers**
 - Um único **mestre mantém metadados com localização dos chunks**: controla acesso e permite visão global da rede
 - **Tabela de metadados é pequena** (poucos chunks): pode ficar na **memória principal**, acelerando acesso



“Big Data”: Projeto do GFS (cont.)

- ❑ Confiabilidade de dados via replicação de chunks (cont.)
 - Mestre deve garantir número de chunks replicados:
 - Distribuir **nova réplica** quando um ChunkServer **falha**
 - **Remover réplicas** excedentes quando ChunkServer é restabelecido
 - Mestre define ChunkServer “primário” para interagir com clientes
 - Em caso de **alterações** em chunks: cliente contacta diretamente os ChunkServers com as réplicas, mas **primário controla quando alterações são aplicadas.**



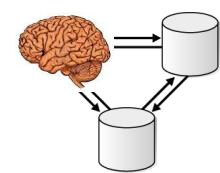
“Big Data”: Projeto do GFS (cont.)

- Um único mestre: simples, mas



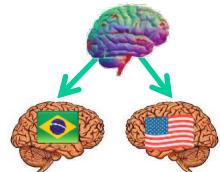
- **Ponto único de falha**
 - **Gargalo** para escalabilidade do sistema

- Soluções do GFS:



- **Replicação** remota de log de operações do mestre (“shadows”): **restabelecimento rápido em caso de falha**
 - **Envolvimento mínimo do mestre**: após obter metadados, cliente não acessa mestre enquanto lê chunk (grande)

- Obs.: o Google vem trabalhando em solução com **mestre distribuído (“Colossus”)**



- Poucas informações públicas, exceto pela estrutura da base de dados “Spanner”: hierarquia por zonas
(<http://research.google.com/archive/spanner-osdi2012.pdf>)

“Big Data”: Uso do GFS



- ❑ Basicamente todos os sistemas do Google...
 - Mais de 50 clusters, gerenciando petabytes de dados
- ❑ Dentre eles: **BigTable**
 - Armazenamento distribuído de dados da ordem de terabytes
 - **Dados não estruturados:** armazenamento na forma de mapa indexado por <linha, coluna, timestamp>
 - Ordenação das chaves por linhas
 - Cada célula da tabela pode conter diversas instâncias de um mesmo arquivo, diferenciadas pelo timestamp
 - Admite uso de locks: serviço provido por ferramenta “Chubby”

“Big Data”: Hadoop



- ❑ **GFS** é sistema proprietário, de **código fechado**
- ❑ **Hadoop Distributed File System (HDFS)**: versão de **código aberto** do GFS
 - Obtido por **engenharia reversa** do GFS, feita pelo Yahoo!
 - Distribuído pela Apache
 - Linguagens de programação principais: **Java** (base) ou **Pig** (otimizada para **MapReduce**)
 - Pig: código MapReduce 20x menor, requer 16x menos tempo para programar e executa pouco mais lentamente do que Java.
 - Usa JobTracker para agendar tarefas de MapReduce e TaskTracker para executá-las



Apêndice: DHT

Aplicações P2P & Anonimato: DHT



▫ Hash Table (“Tabela de Dispersão”)

- Estrutura de dados que mapeia “chaves” em “valores”
- Interface:
 - `put(chave, valor)`: insere valor na tabela
 - `get(chave)`: recupera valor da tabela

▫ Distributed Hash Table (DHT)

- Similar, mas espalhada pela Internet
- Objetivo: permitir a qualquer nó localizar conteúdo

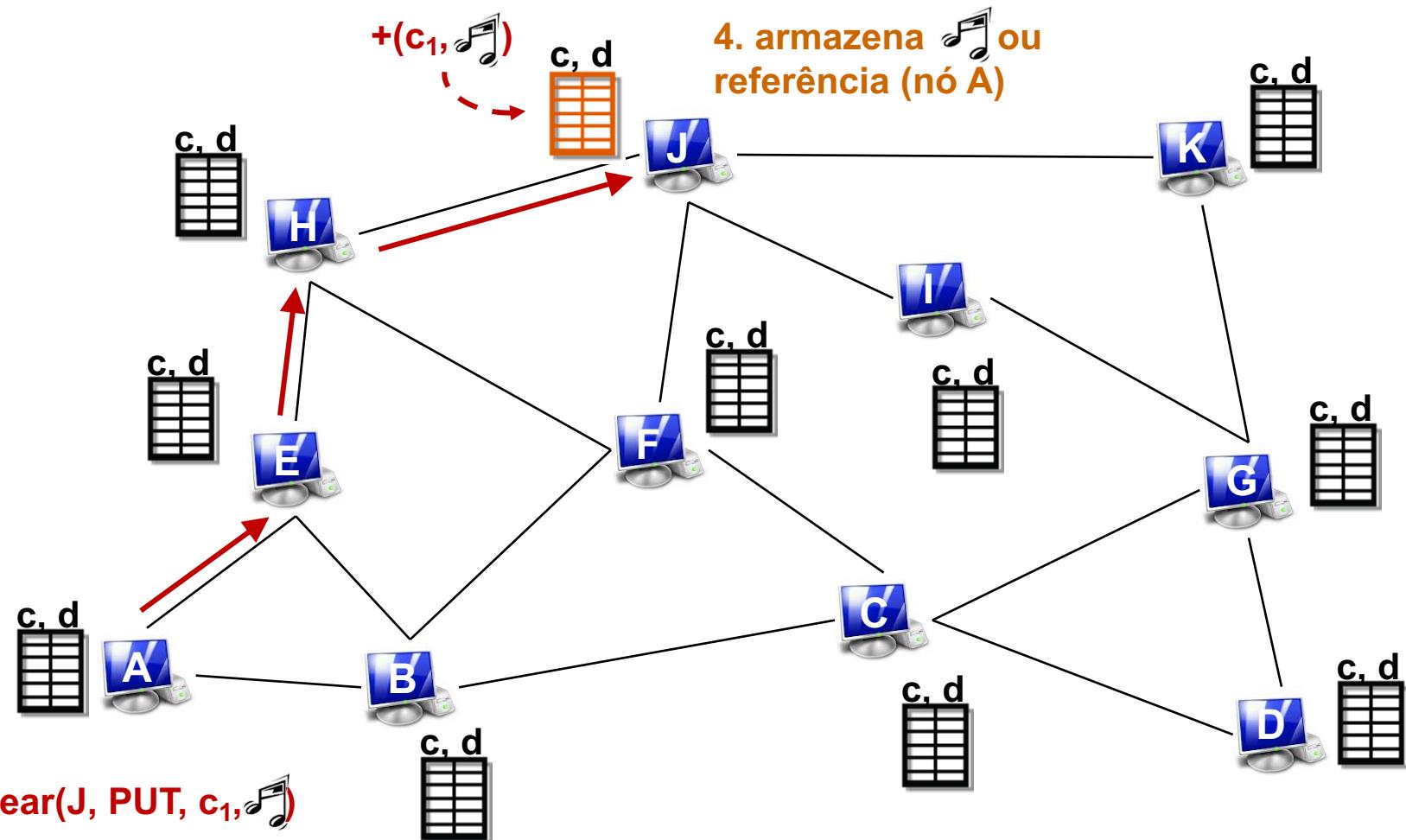
Aplicações P2P & Anonimato: DHT

- ❑ Hash table em um único nó:
 - chave = hash (dado)
 - put(chave, dado)
 - get(chave): dado
- ❑ Distributed Hash Table (DHT):
 - chave = hash (dado)
 - Busca (chave) : IP_nó
 - Rotear (IP_nó, PUT, chave, dado)
 - Rotear (IP_nó, GET, chave) : dado
- ❑ Ideia:
 - Nó específico armazena (referência para) conteúdo específico
 - Todos os nós têm capacidade de roteamento: dada uma chave, eles roteiam mensagens para o nó que armazena a chave

Hash( M) = 2442

chave	dado
0140	 E
1515	 Z
2442	 M
3910	 P
4441	 X
...	...
8731	 Y

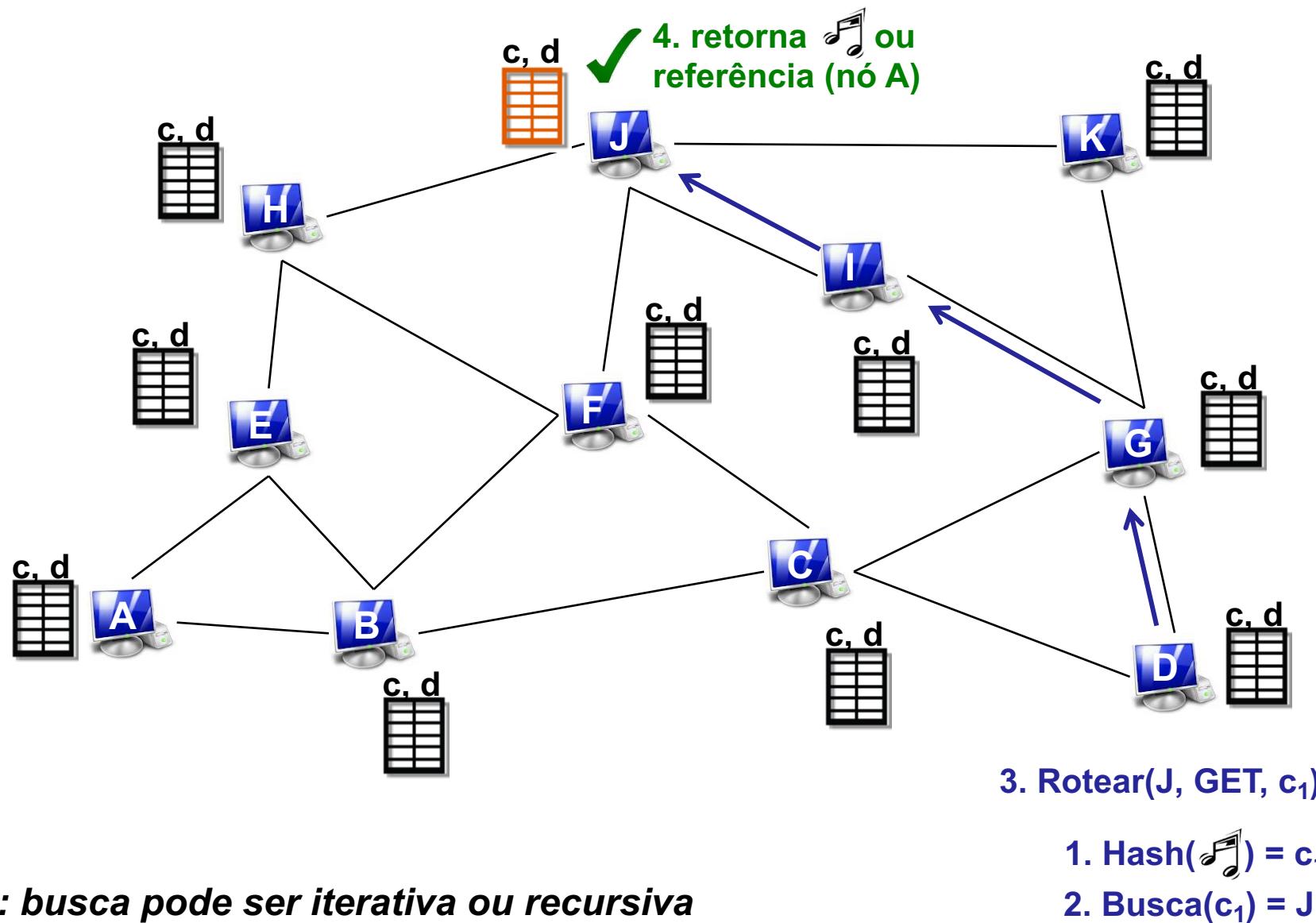
DHT: inserindo arquivos (“PUT”)



1. Hash(🎵) = c_1

2. Busca(c_1) = J

DHT: buscando arquivos (“GET”)



Obs.: busca pode ser iterativa ou recursiva

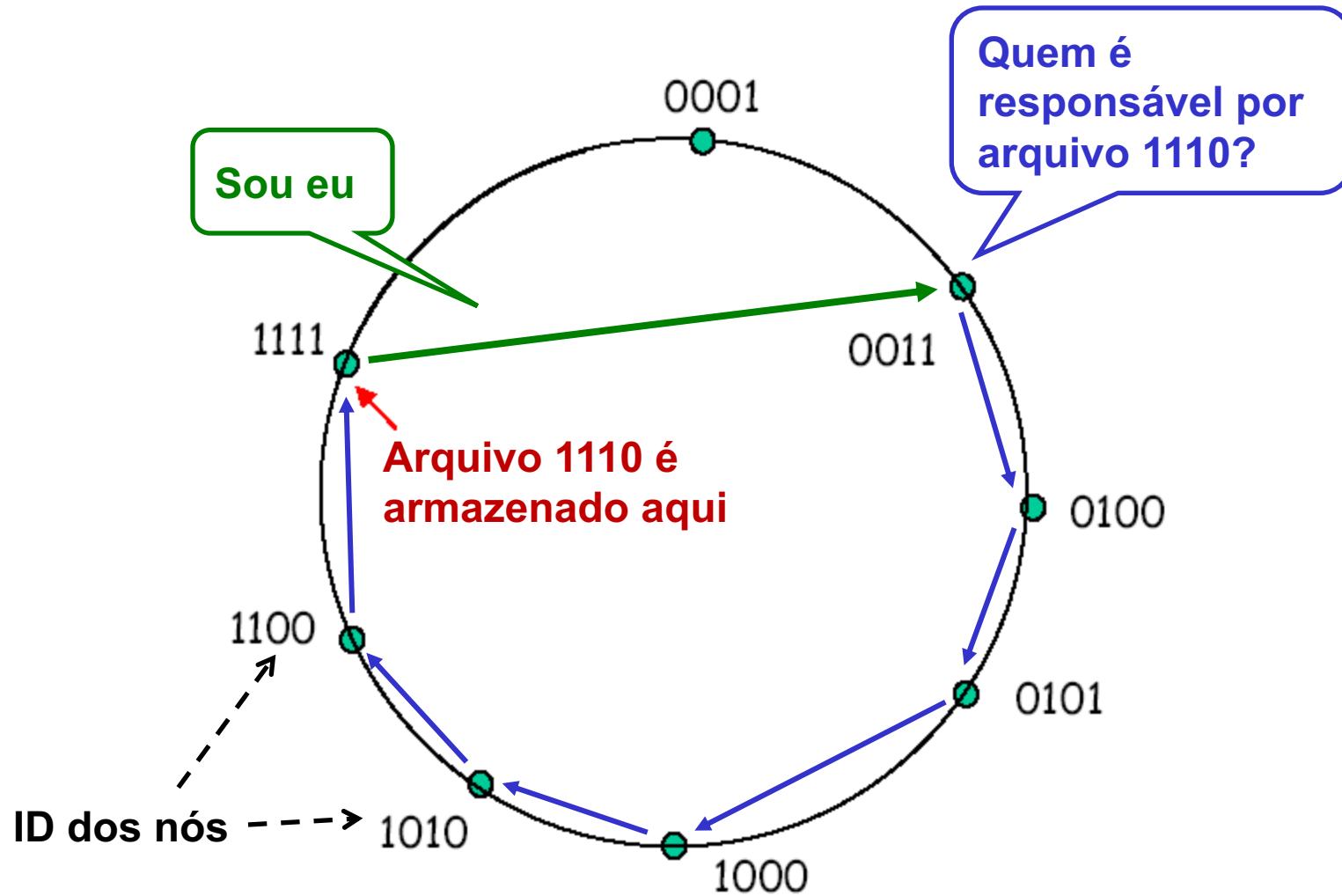
Abordagens de DHT

- ❑ Hashing consistente
- ❑ Chord
- ❑ CAN (Content Addressable Network)
- ❑ Pastry
- ❑ Tapestry
- ❑ Kademlia

Hashing consistente (cont.)

- ❑ Problema: associar arquivo a nó na rede
- ❑ Ideia: calcular hash **h** do arquivo (chave) e associá-lo ao nó “mais próximo” de **h**, dada uma certa métrica de proximidade
- ❑ Uma abordagem possível: rede overlay é um círculo
 - Cada nó tem um ID escolhido aleatoriamente: hash (end. IP)
 - Chaves no mesmo espaço de IDs
 - Sucessor de nó A: nó cujo ID seja imediatamente maior que ID_A
 - Cada nó sabe IP de seu sucessor

Hashing consistente (cont.)



Hashing consistente (cont.)

▫ Funcionamento básico

Saída de um nó	Entrada de um nó
<ul style="list-style-type: none">• Se o sucessor de A deixa a rede, A precisa escolher próximo sucessor• Logo: cada nó mantém referência para $s \geq 2$ sucessores• Quando o sucessor de A deixa a rede, A pede a seu novo sucessor pela sua lista de sucessores; A então atualiza sua própria lista de s sucessores	<ul style="list-style-type: none">• Você é um novo nó, e seu ID é k• Peça a qualquer nó N para encontrar o nó N' que é sucessor de k• Obtenha sua lista de sucessores de N'• Diga a seus predecessores que atualizem suas listas de sucessores• Logo: cada nó deve saber quem é seu predecessor

▫ $\#vizinhos = s+1 = O(1)$

➤ Tabela de roteamento: (ID_vizinho, IP_vizinho)

▫ Número médio de mensagens para encontrar chave: $O(n)$

**Podemos
fazer melhor?**

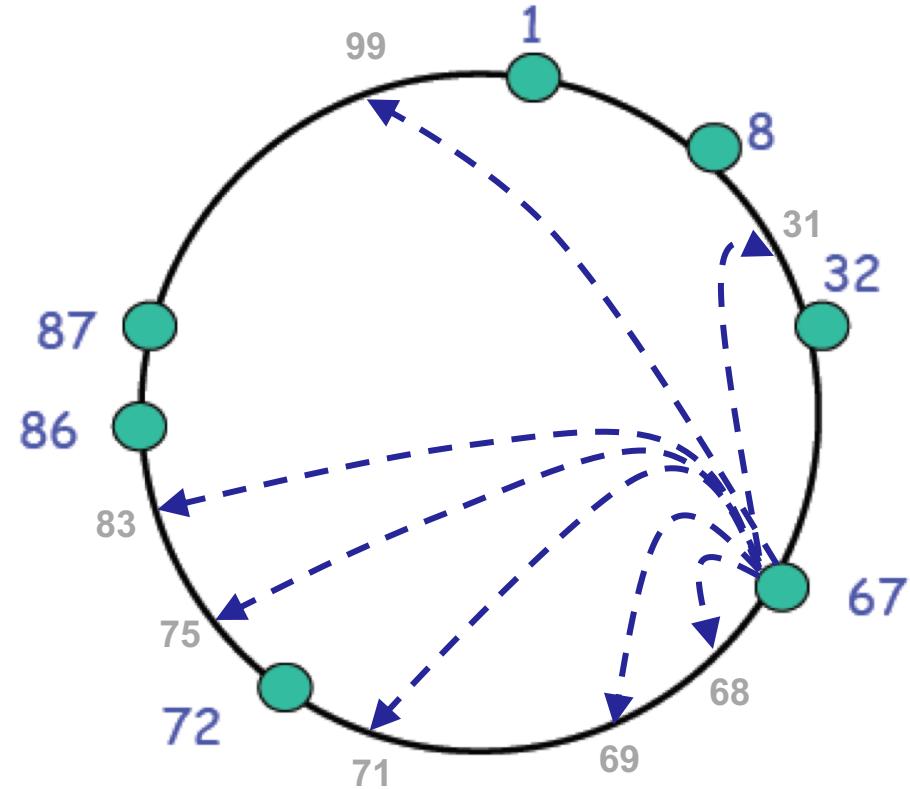
Chord

- Baseado em hashing consistente
 - Overlay circular
 - ID aleatório unidimensional no espaço de hashes
 - Domínio: $] \text{ID}_{\text{anterior}}, \text{ID}_{\text{próprio}] \pmod {|\text{espaço de IDs}|}$
- Tabela de derivação (*finger table*)
 - Conjunto de vizinhos conhecidos
 - O i -ésimo vizinho (sentido horário) do nó de ID \mathbf{n} tem o ID mais próximo de (e é maior que) $\mathbf{n+2^i} \pmod {|\text{espaço de IDs}|}$, $i \geq 0$
- Roteamento
 - Para alcançar o nó responsável pelo ID $\mathbf{n'}$, envie a mensagem para o vizinho número $\mathbf{\log_2(n' - n)}$

Chord (cont.)

- ❑ Finger table para nó 67 → $n=67$
- ❑ Espaço de IDs de $[0, 99]$ → $N = 100$

i	$(n+2^i) \bmod N$	Finger table
0	68	
1	69	
2	71	
3	75	
4	83	
5	99	
6	31	
7	95	x



Chord (cont.)

- ❑ Roteamento: busca de $K = 25$

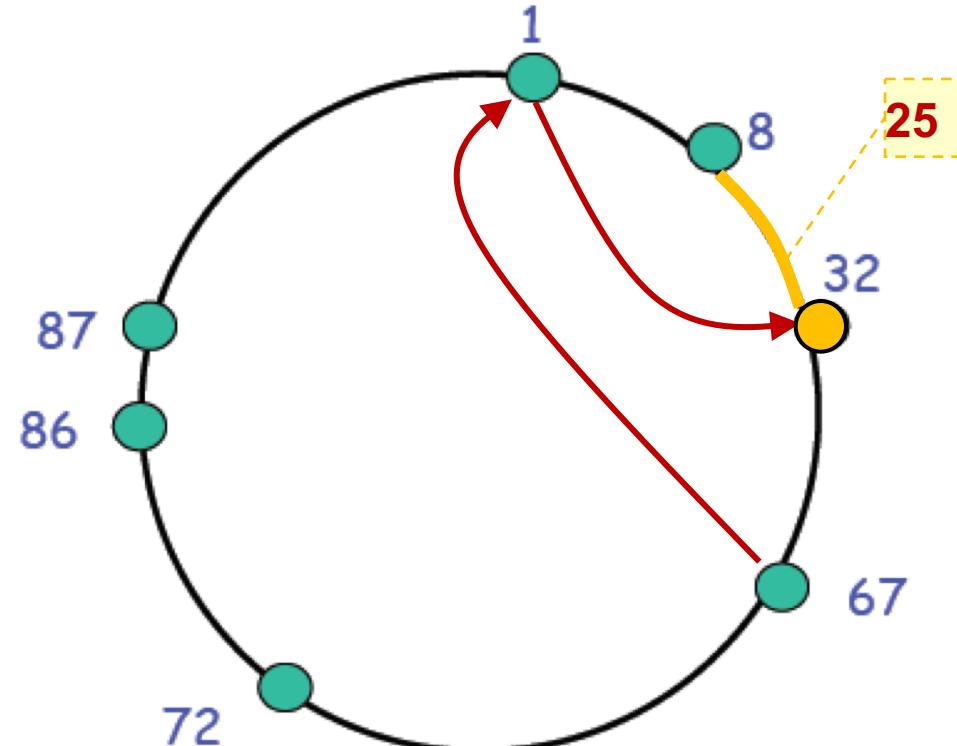
- **Nó 67:** destino = $\log_2((25 - 67) \bmod 100) = \log_2(58) \approx 5$
- **Nó 1:** destino = $\log_2((25 - 1) \bmod 100) = \log_2(24) \approx 4$

$n = 67$

i	Finger table
0	72
1	72
2	72
3	86
4	86
5	1
6	32

$n = 1$

i	Finger table
0	8
1	8
2	8
3	32
4	32
5	67
6	67



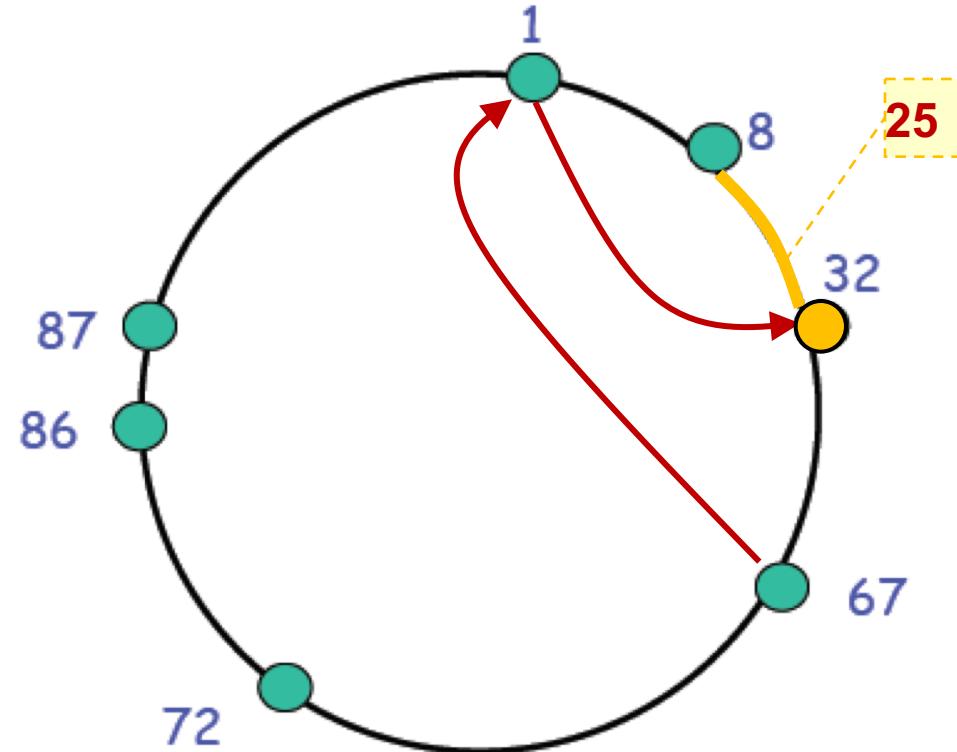
Chord (cont.)

- ❑ Roteamento: qualquer nó pode ser alcançado a partir de qualquer outro nó em até $\log_2(N)$

- N: número máximo de nós da rede

- Em nosso exemplo:

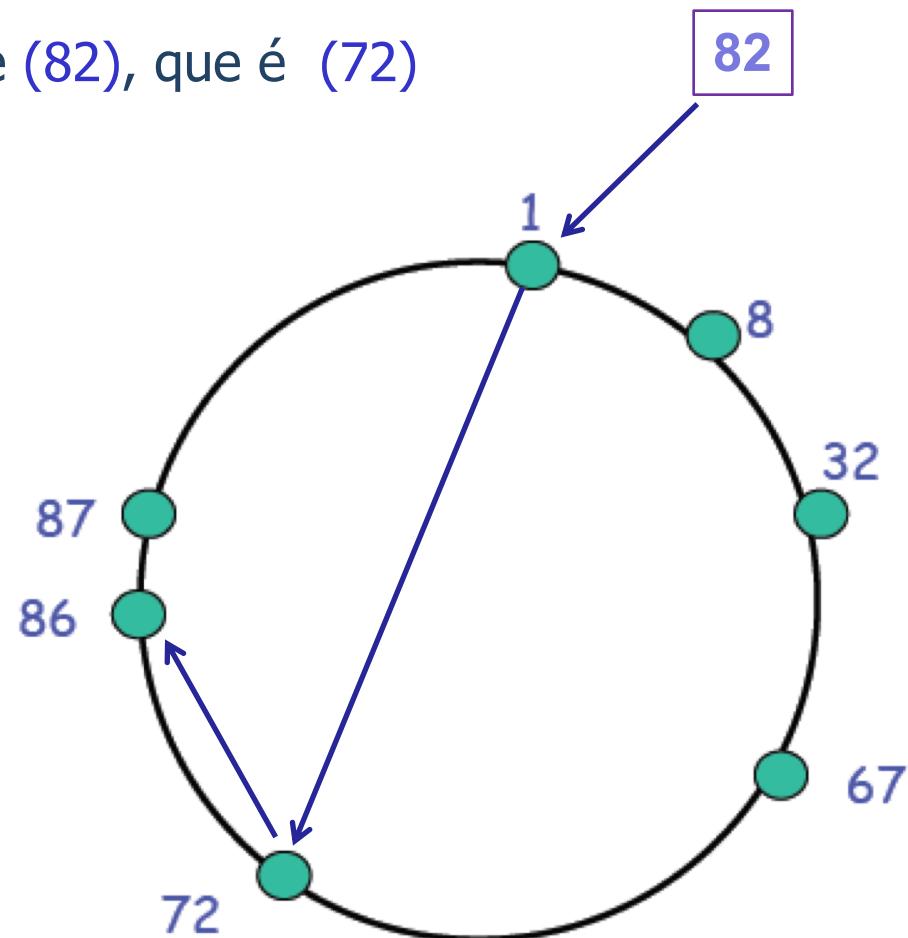
- $N = 7 \rightarrow \log_2(N) \approx 3$
 - Busca: custo de 2



Chord (cont.)

▫ Entrando na rede:

- Nó (82) conhece nó (1), obtido fora de banda
- (1) encontra predecessor de (82), que é (72)

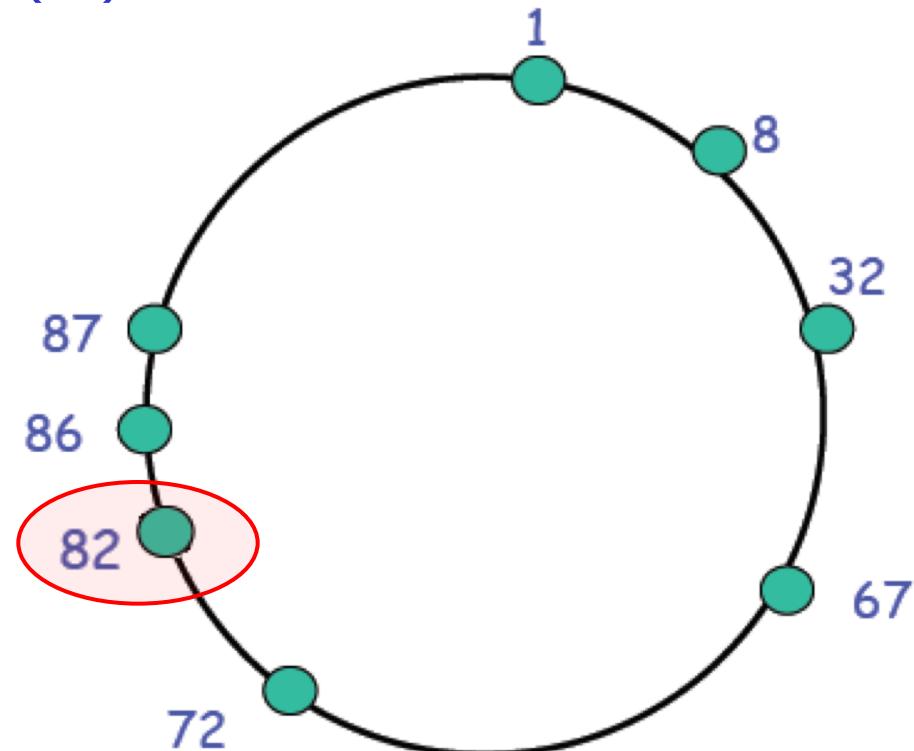


Chord (cont.)

Entrando na rede:

- › Nó (82) conhece nó (1), obtido fora de banda
- › (1) encontra predecessor de (82), que é (72)
- › (72) constrói finger table de (82)

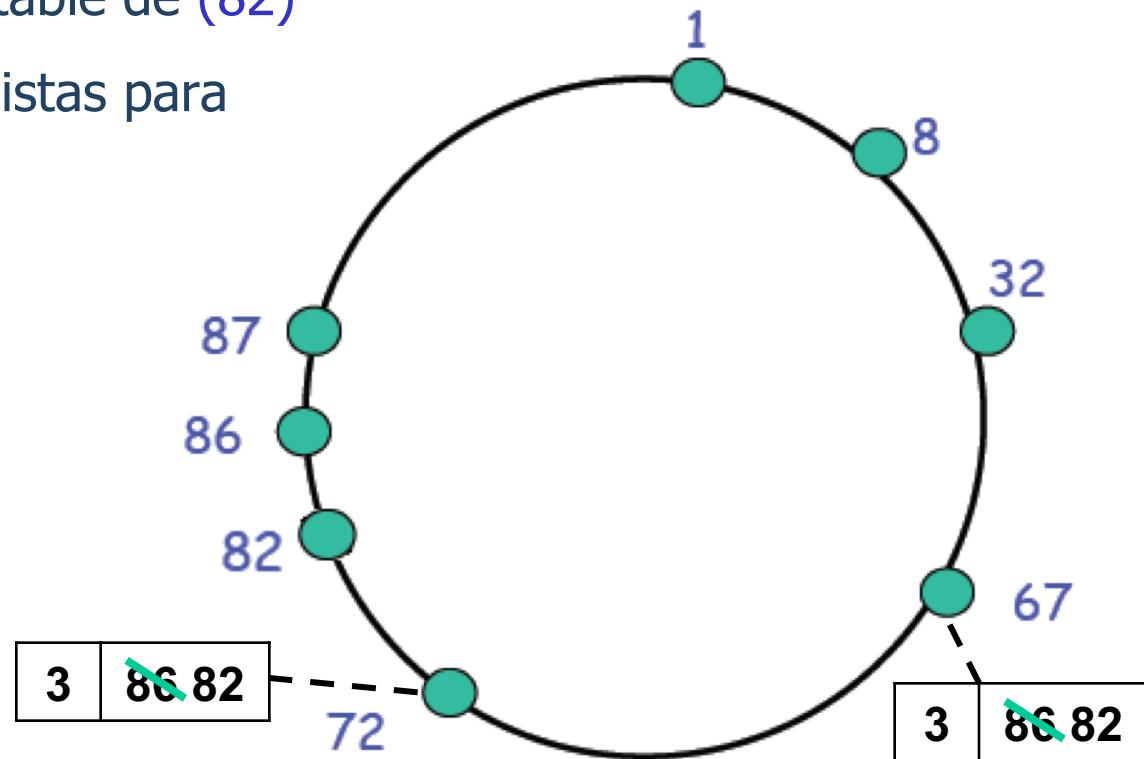
i	$(n+2^i)\text{mod } N$	Finger table
0	68	72
1	69	72
2	71	72
3	75	86
4	83	86
5	99	1
6	3	8



Chord (cont.)

Entrando na rede:

- › Nó (82) conhece nó (1), obtido fora de banda
- › (1) encontra predecessor de (82), que é (72)
- › (72) constrói finger table de (82)
- › Nós atualizam suas listas para considerar (82)
- › Atualização: $\log_2(N)$



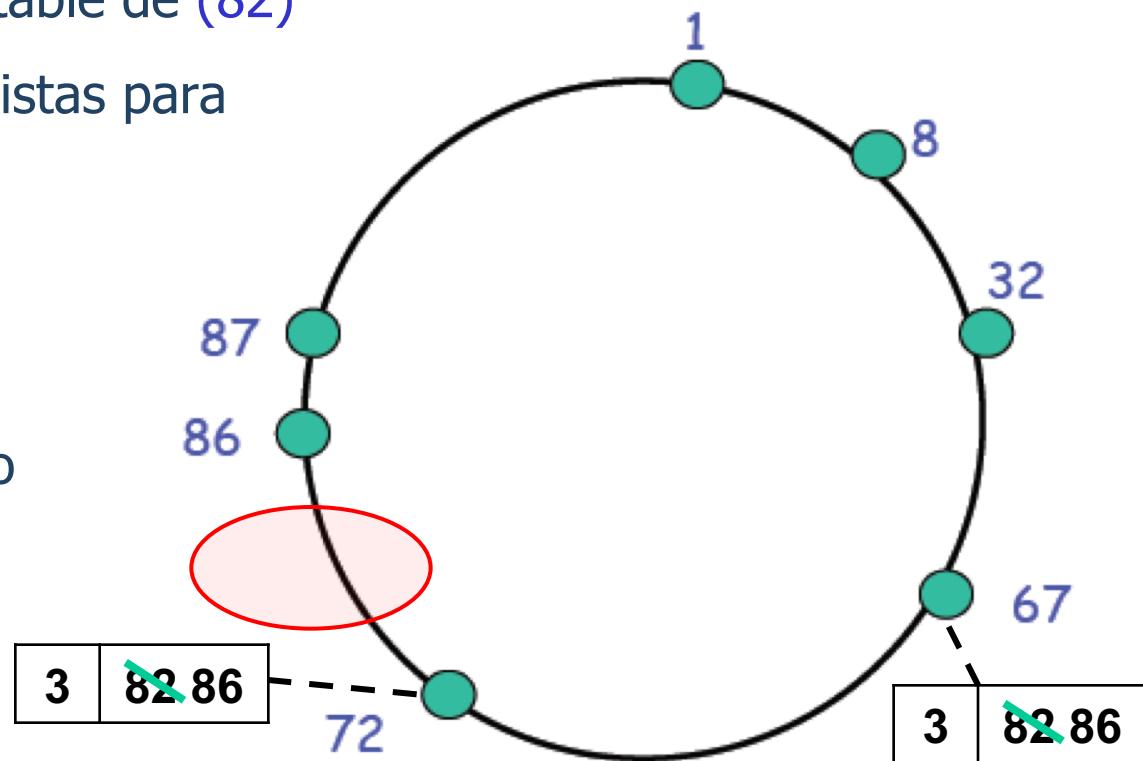
Chord (cont.)

Entrando na rede:

- › Nó (82) conhece nó (1), obtido fora de banda
- › (1) encontra predecessor de (82), que é (72)
- › (72) constrói finger table de (82)
- › Nós atualizam suas listas para considerar (82)
- › Atualização: $\log_2(N)$

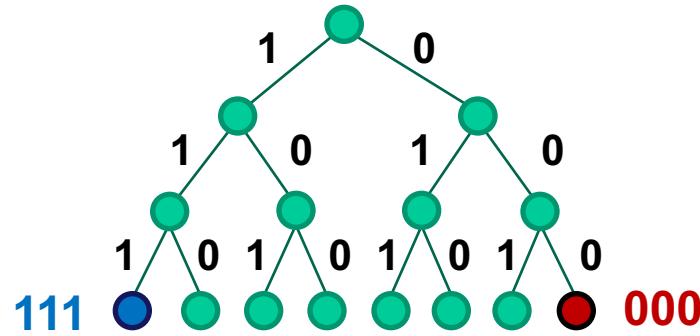
Saindo da rede:

- › Atualização após não receber resposta a mensagem de “keep-alive”



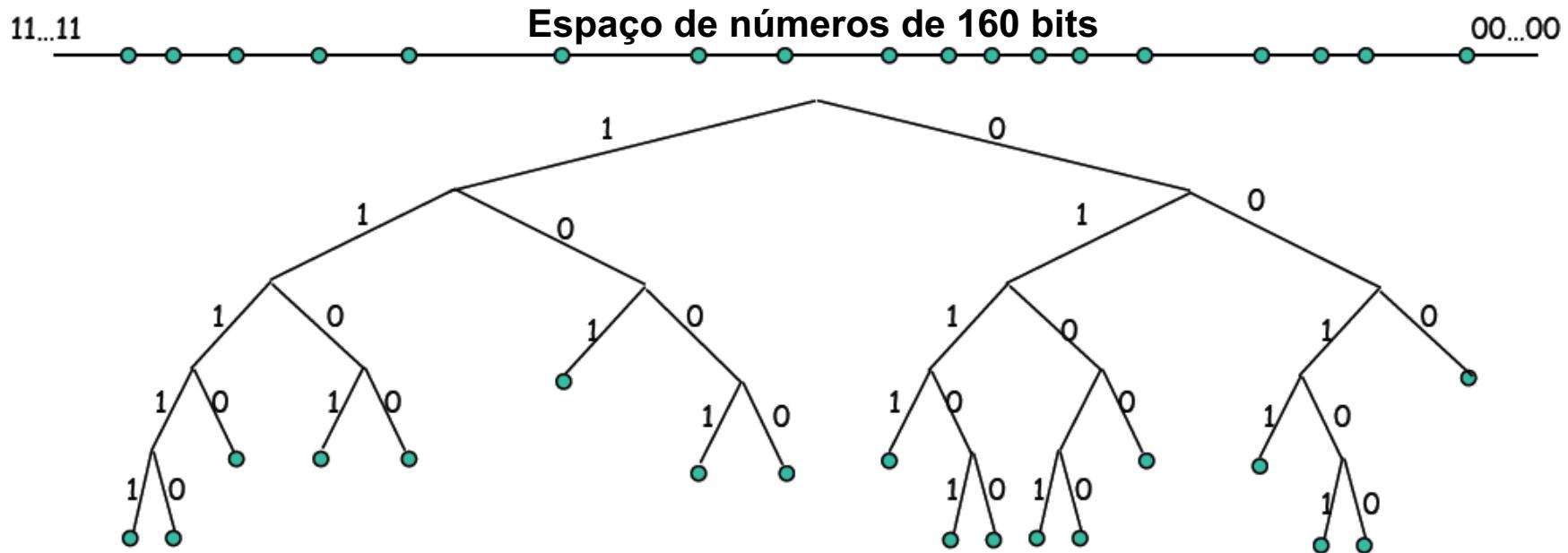
Kademlia

- Algoritmo de DHT criado em 2002
 - É atualmente um dos algoritmos de DHT mais utilizados por algoritmos P2P, como links magnéticos
- Estrutura baseada em uma árvore binária
 - Organização dos nós e chaves dos arquivos



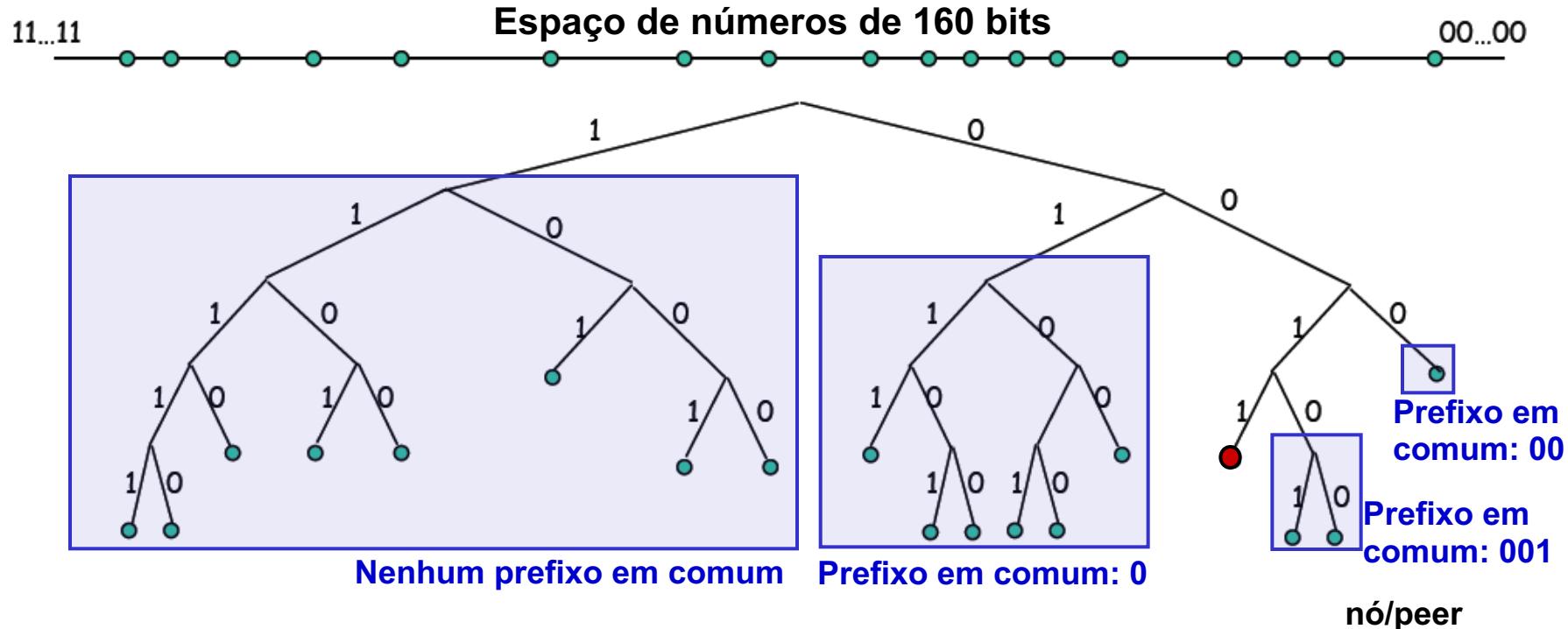
- S/Kademlia: ID é a chave pública do nó

Kademlia



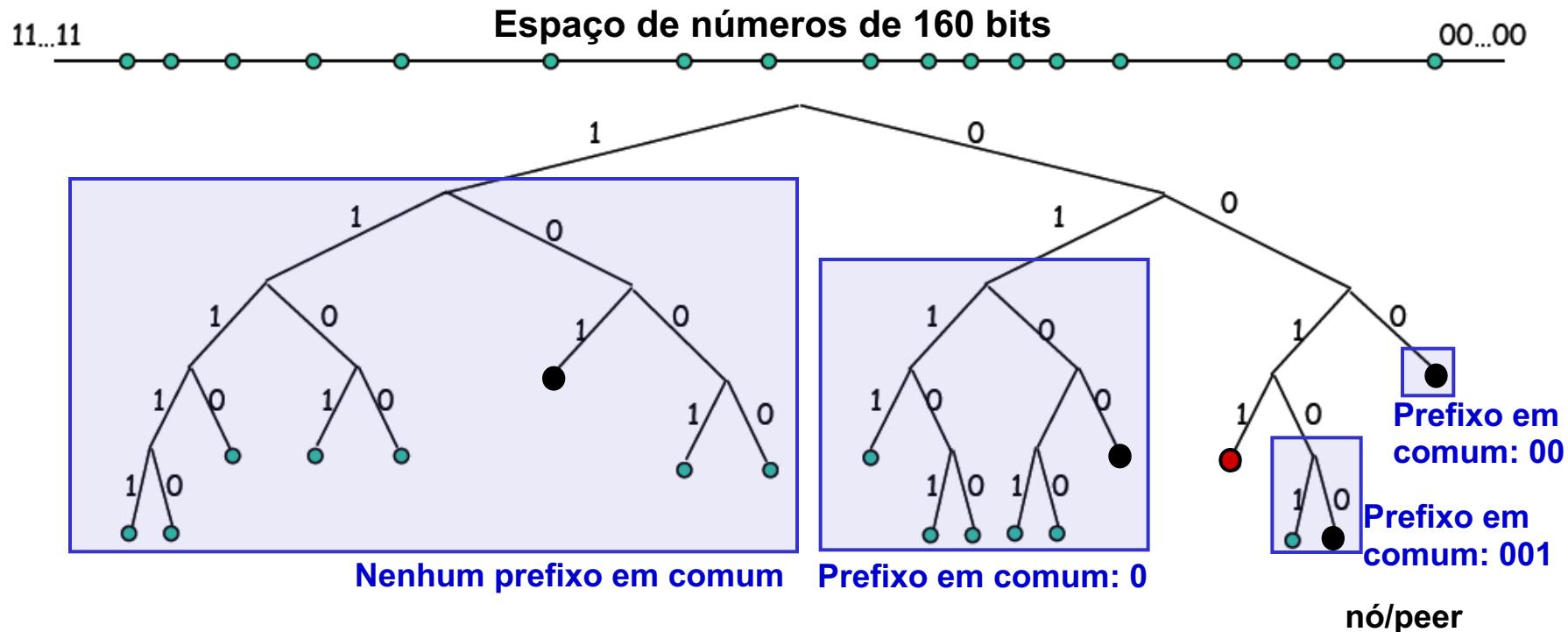
- ❑ Nós: folhas em uma árvore binária ● nó/peer
 - Posição determinada pelo menor prefixo exclusivo de seu ID
 - Nó responsável por dados com IDs mais “próximos” a ele
 - Distância entre ID x e y: $d(x,y) = x \oplus y \rightarrow d(0101, 0011)=0110$
 - Nós/IDs na mesma sub-árvore têm maior prefixo em comum → estão mais próximos

Kademlia: distribuição de IDs



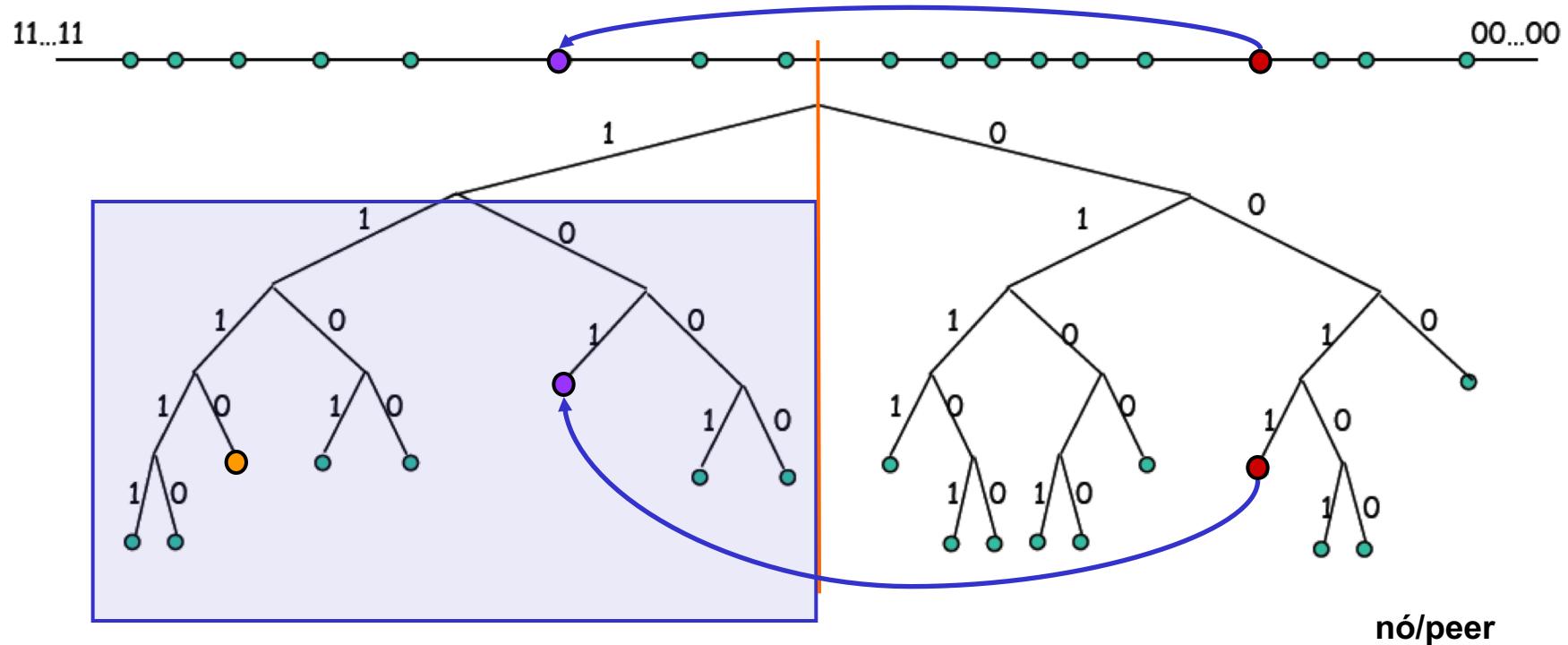
- Para um nó qualquer (ex.: nó 0011):
 - A árvore binária é dividida em sub-árvores de tamanho máximo que não contêm o nó

Kademlia: distribuição de IDs



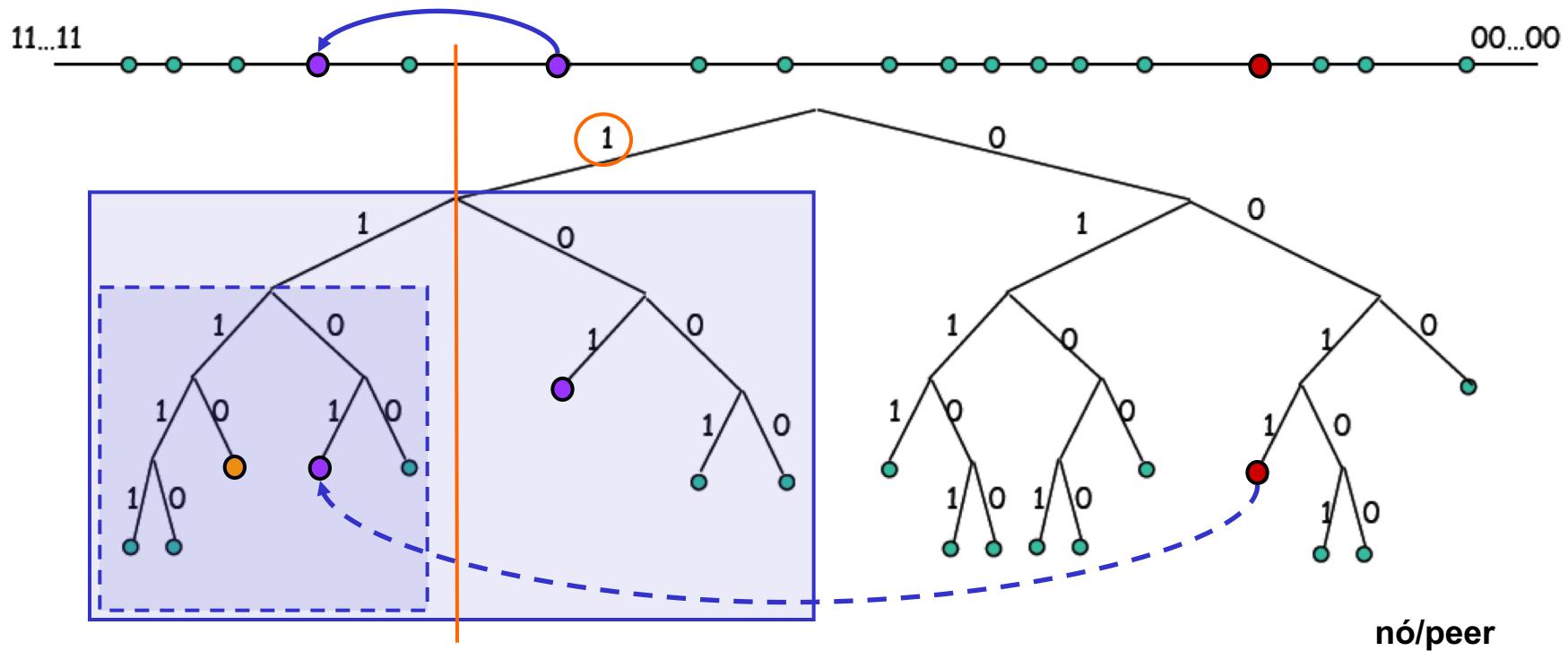
- Para um nó qualquer (ex.: nó 0011):
 - Um nó deve conhecer pelo menos um outro nó em cada uma destas sub-árvore (ex.: nó 0011 conhece nós em **preto**).
 - S/Kademlia: escolha de nós maximiza conectividade mesmo na presença de vários nós maliciosos

Kademlia: roteamento de busca



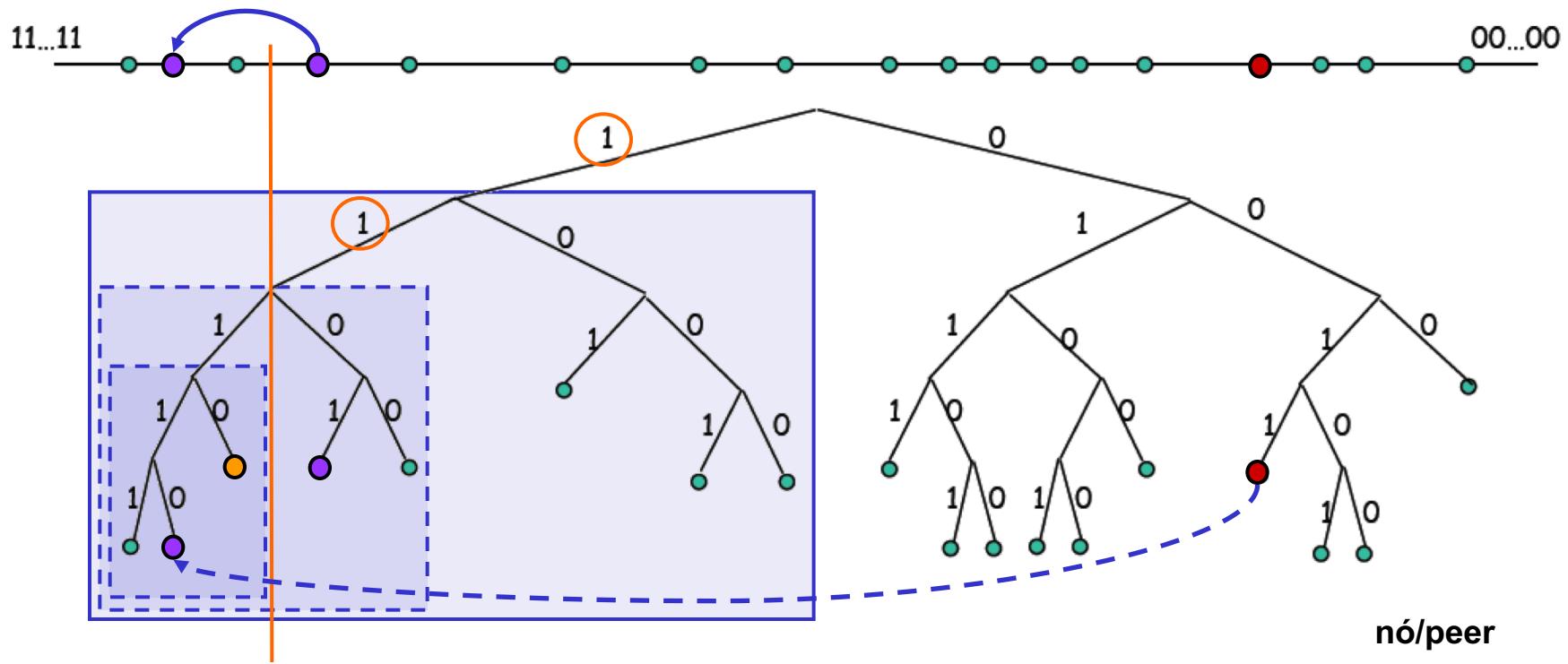
- Considere que o nó **0011100...** está procurando um dado cujo ID é **111010...**
 - Nenhum prefixo em comum: ele contacta nó na árvore correspondente (com prefixo 1: **101...**) que roteia a busca

Kademlia: roteamento de busca (cont.)



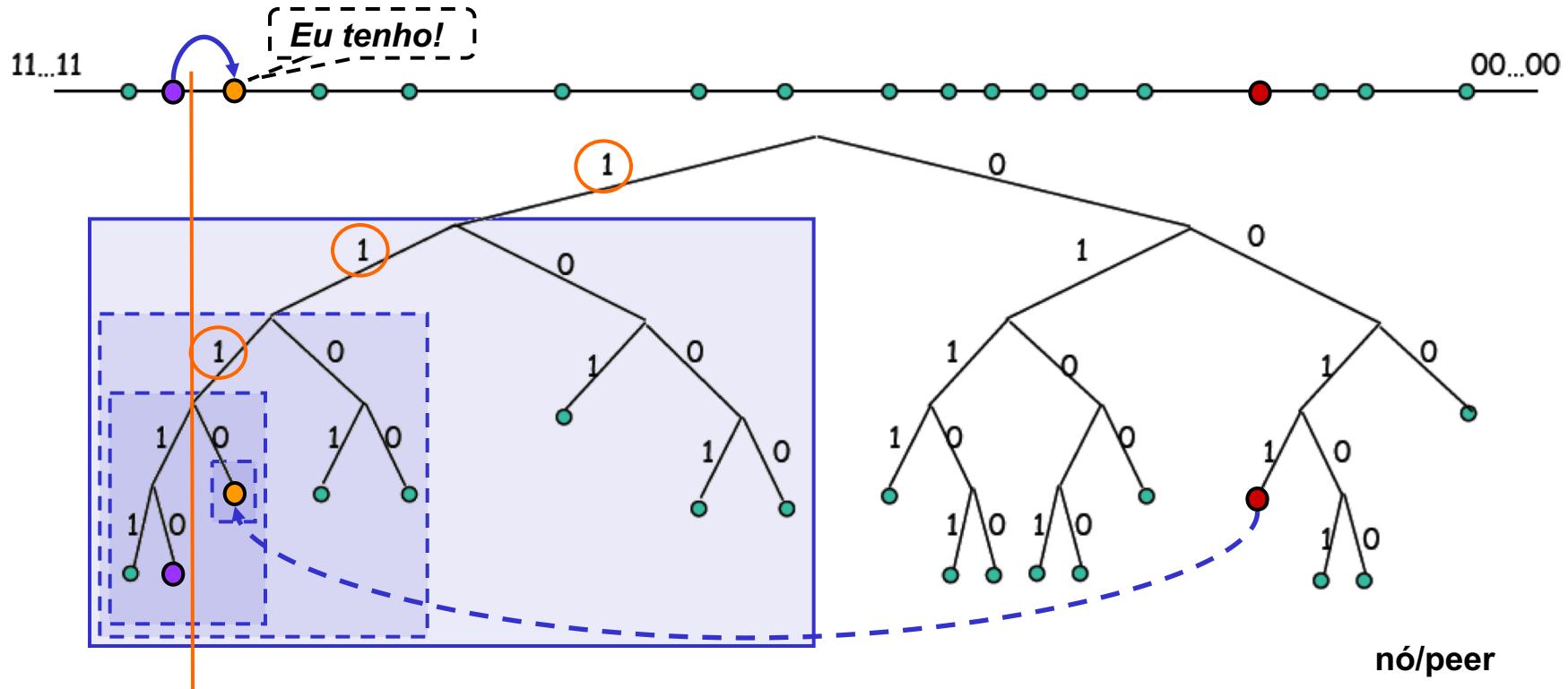
- Considere que o nó **0011100...** está procurando um dado cujo ID é **111010...**
 - Processo se repete até nó correto ser encontrado: nó **1101...**

Kademlia: roteamento de busca (cont.)



- Considere que o nó **0011100...** está procurando um dado cujo ID é **111010...**
 - Processo se repete até nó correto ser encontrado: nó **11110...**

Kademlia: roteamento de busca (cont.)



- Considere que o nó **0011100...** está procurando um dado cujo ID é **111010...**
 - Processo se repete até nó correto ser encontrado: nó **1110...**
 - Custo total da busca: **O(log₂(N))**