Intro to Crypto and Cryptocurrencies

Slides by Arvind Narayanan et al.

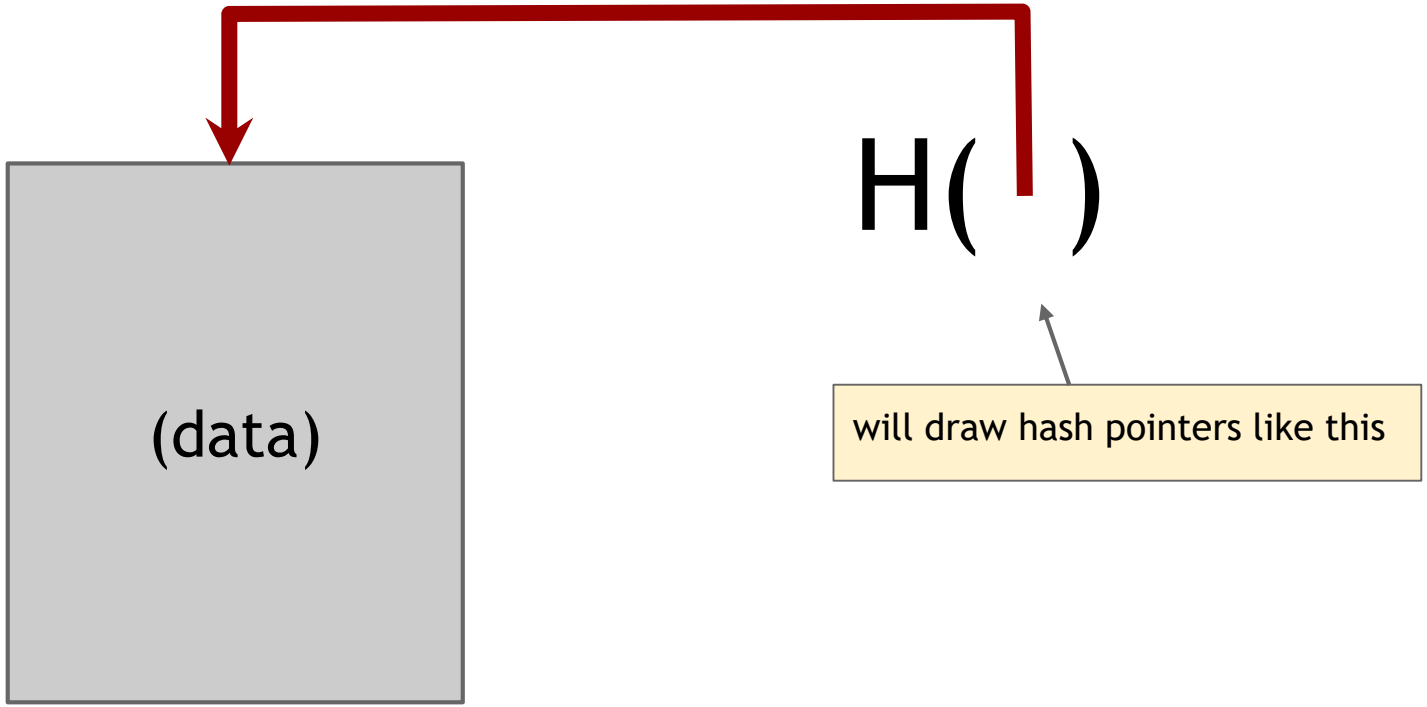
Hash Pointers and Data Structures

hash pointer is:

* pointer to where some info is stored, and
* (cryptographic) hash of the info

if we have a hash pointer, we can

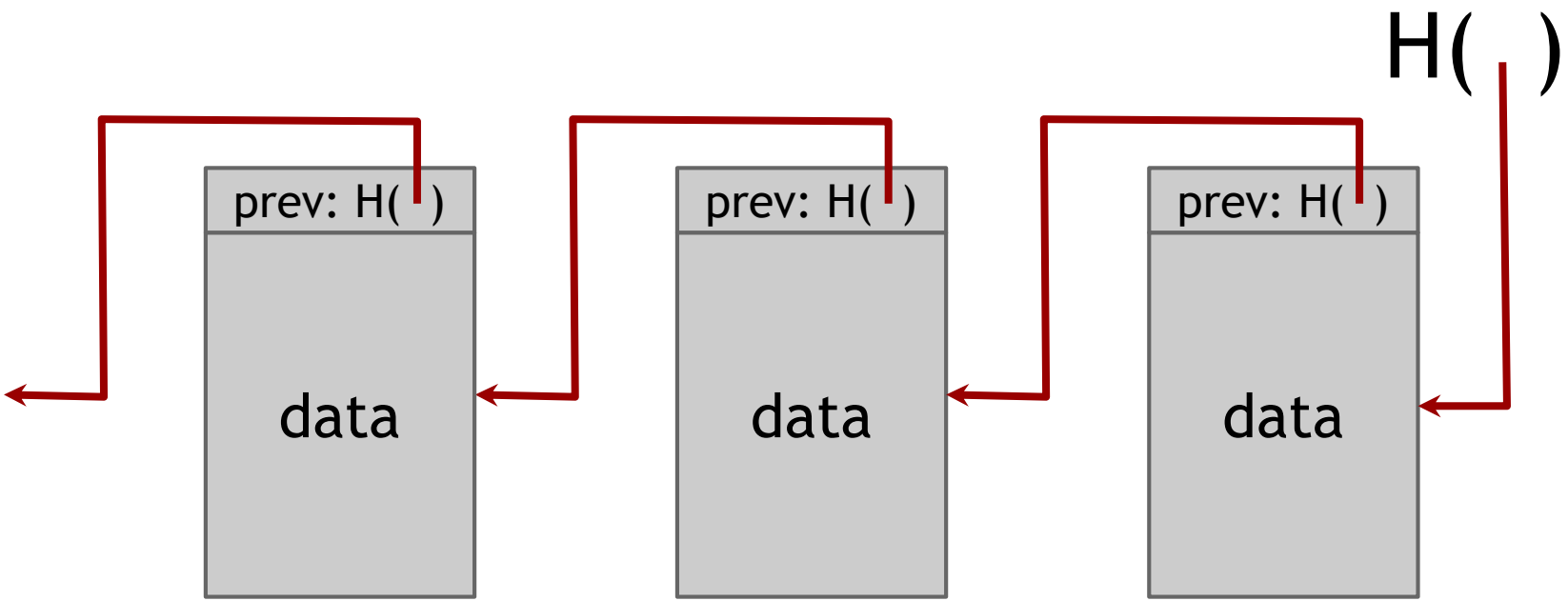
* ask to get the info back, and
* verify that it hasn’t changed


H( )
(data)
will draw hash pointers like this

key idea:

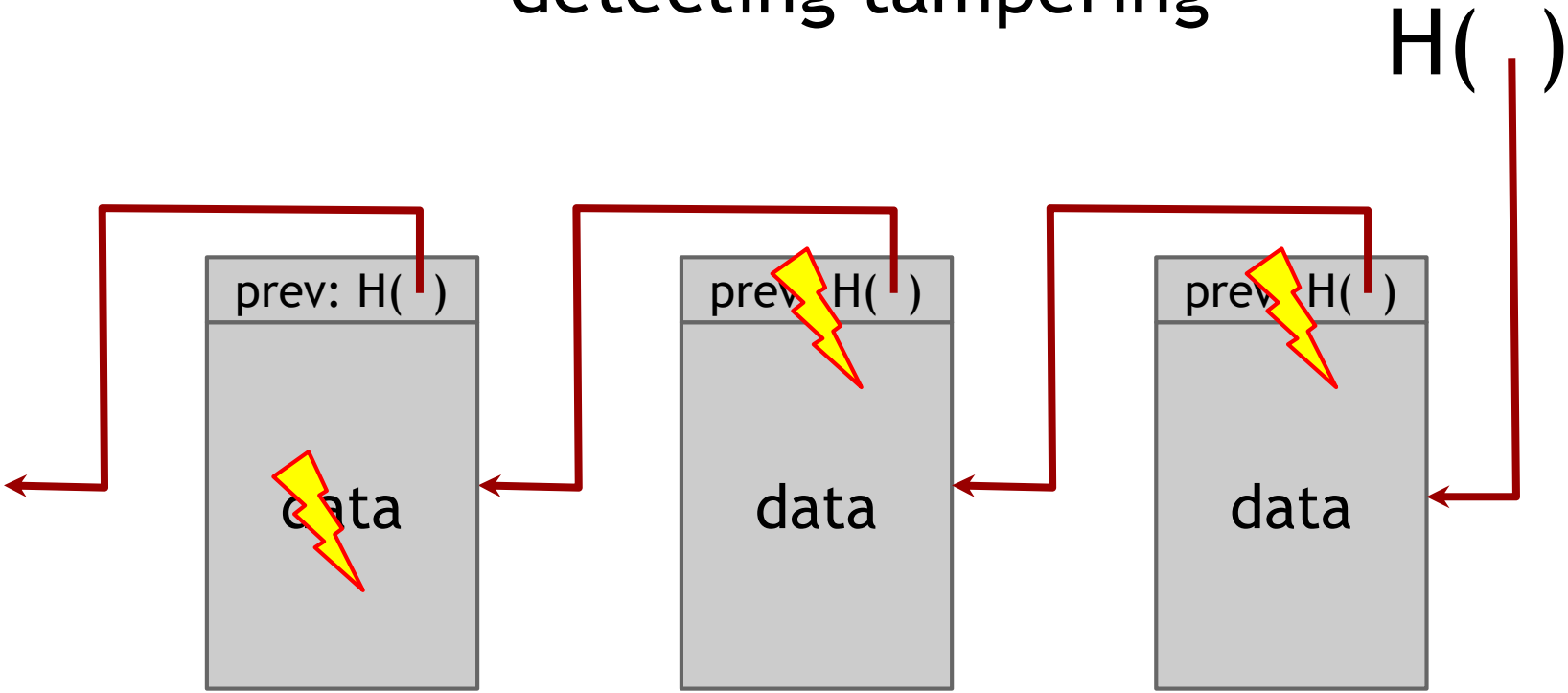
build data structures with hash pointers

linked list with hash pointers = “block chain”



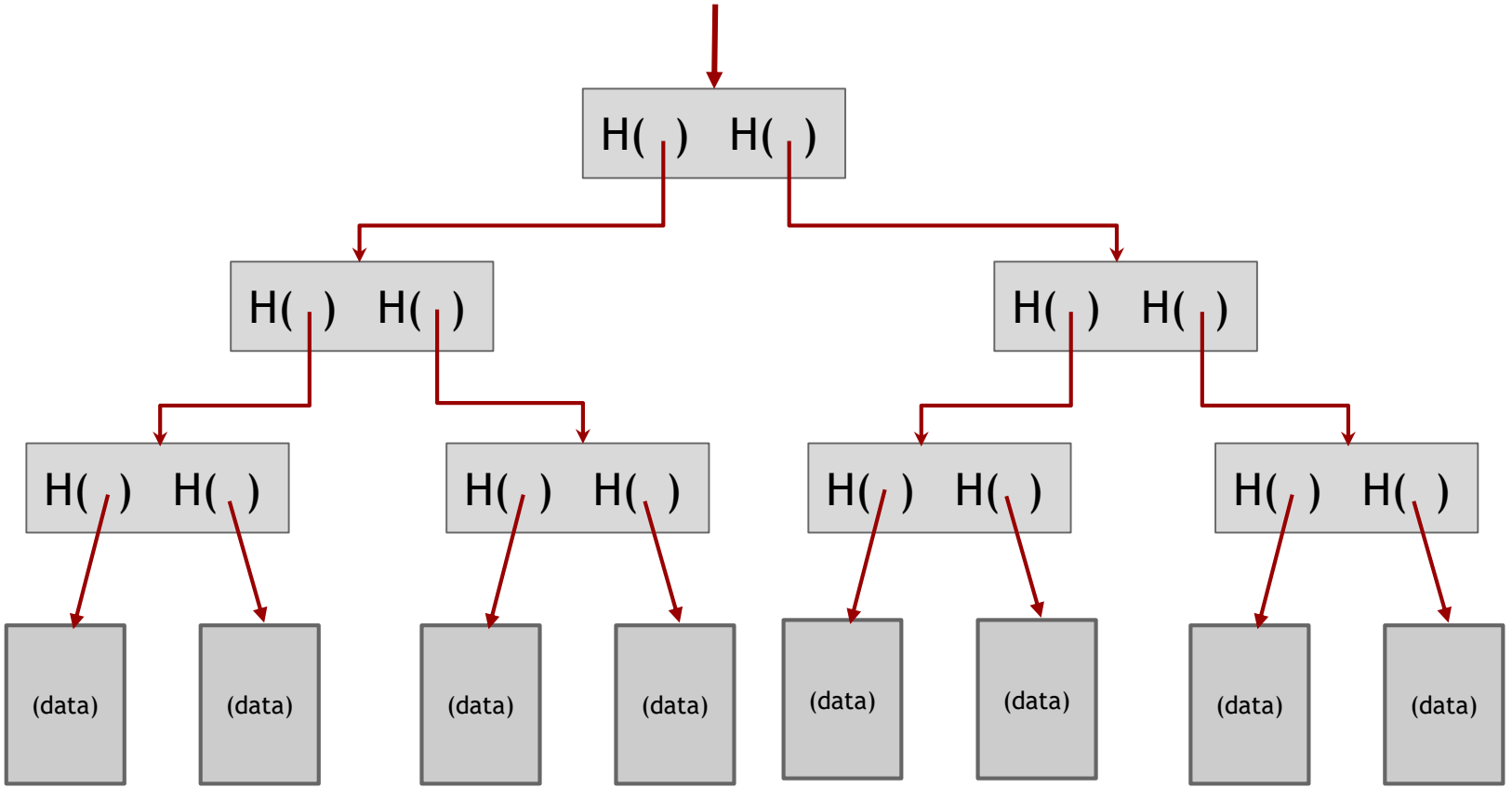
use case: tamper-evident log

detecting tampering

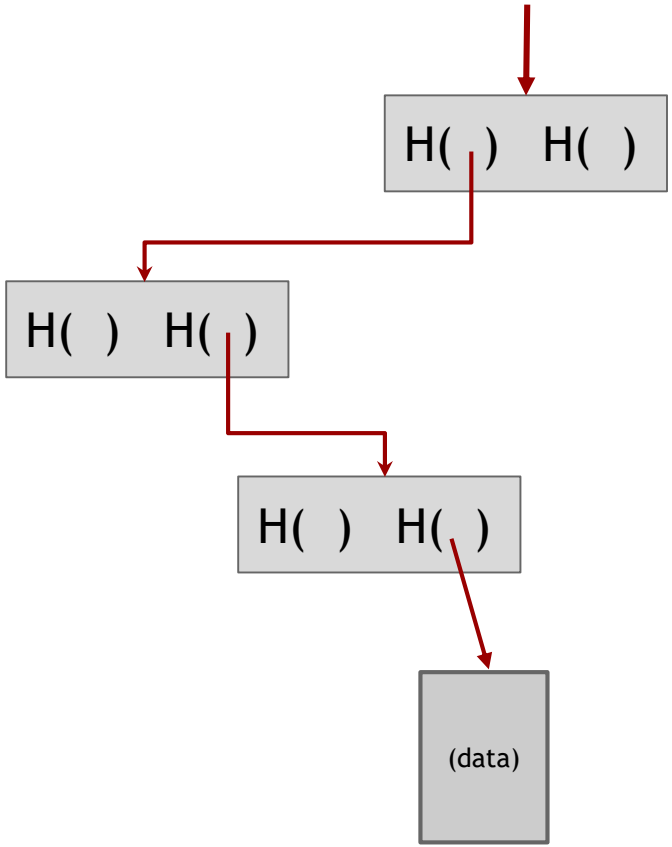


use case: tamper-evident log

binary tree with hash pointers = “Merkle tree”



proving membership in a Merkle tree



show O(log n) items

Advantages of Merkle trees

Tree holds many items

but just need to remember the root hash

Can verify membership in O(log n) time/space

Variant: sorted Merkle tree

can verify non-membership in O(log n)

(show items before, after the missing one)

More generally ...

can use hash pointers in any pointer-based data structure that has no cycles

GoofyCoin

Simple Cryptocurrencies

Obvious approach

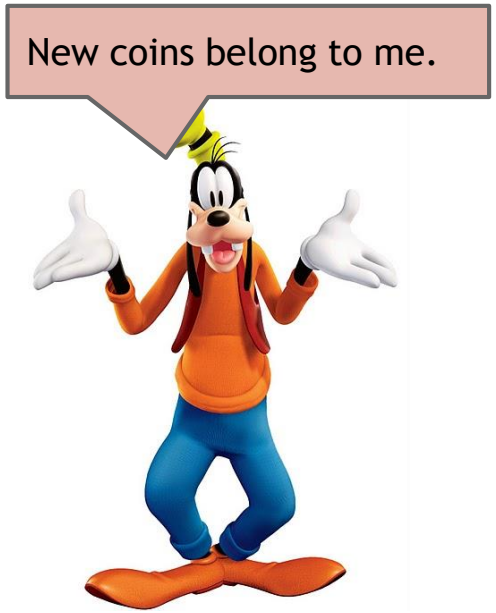
1. Use public keys as addresses
2. Sign to authorize transfer to new address

New coins created [somehow]

Goofy can create new coins

signed by sk_{Goofy}

CreateCoin [uniqueCoinID]



A coin's owner can spend it.

signed by sk_{Goofy}

Pay to pk_{Alice} : H( )

signed by sk_{Goofy}

CreateCoin [uniqueCoinID]

The recipient can pass on the coin again.

signed by sk_{Alice}

Pay to pk_{Bob} : H( )

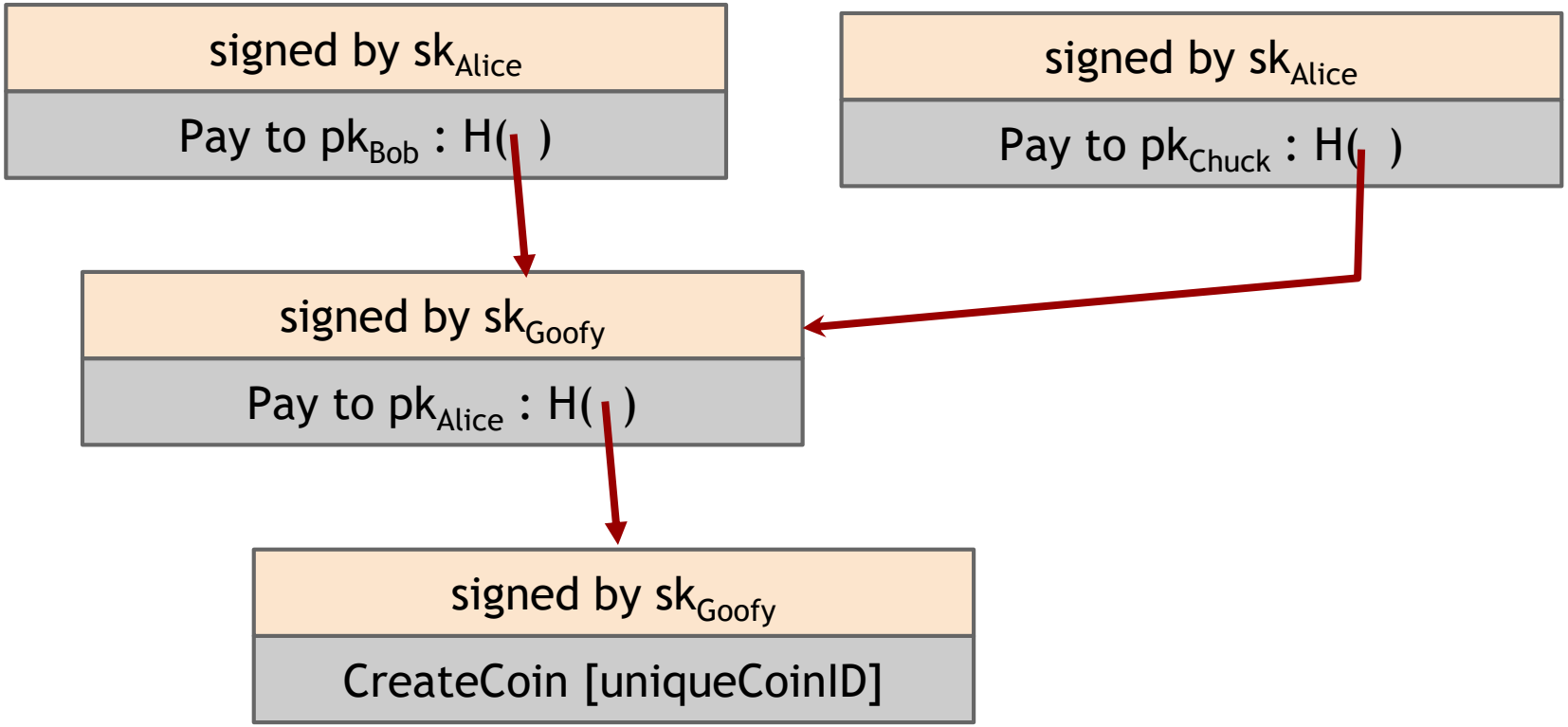
signed by sk_{Goofy}

Pay to pk_{Alice} : H( )

signed by sk_{Goofy}

CreateCoin [uniqueCoinID]

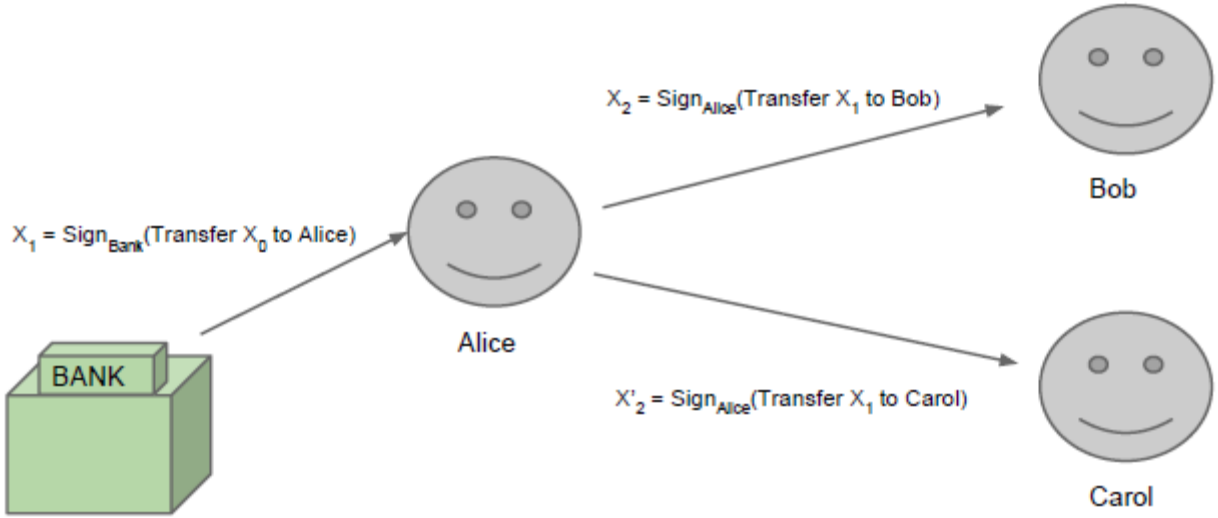
double-spending attack



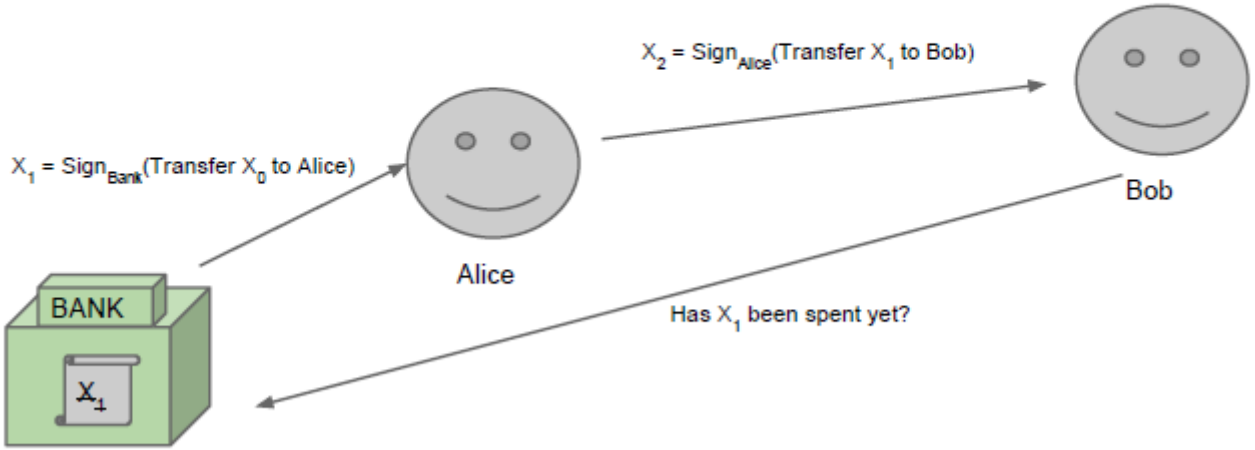
double-spending attack

the main design challenge in digital currency

Double-spends must be prevented



Traditional approach: talk to the issuer



ScroogeCoin

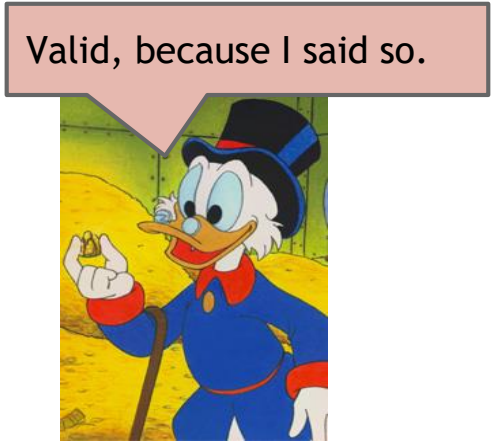
CreateCoins transaction creates new coins

transID: 73	type:CreateCoins	
coins created		
num	*value*	*recipient*
0	3.2	0x...
1	1.4	0x...
2	7.1	0x...

coinID 73(0)

coinID 73(1)

coinID 73(2)



PayCoins transaction consumes (and destroys) some coins, and creates new coins of the same total value

transID: 73 type:PayCoins		
consumed coinIDs: 68(1), 42(0), 72(3)		
coins created		
num	*value*	*recipient*
0	3.2	0x...
1	1.4	0x...
2	7.1	0x...
signatures		

Valid if:
-- consumed coins valid,
-- not already consumed,
-- total value out = total value in, and
-- signed by owners of all consumed coins

Immutable coins

Coins can’t be transferred, subdivided, or combined.

But: you can get the same effect by using transactions

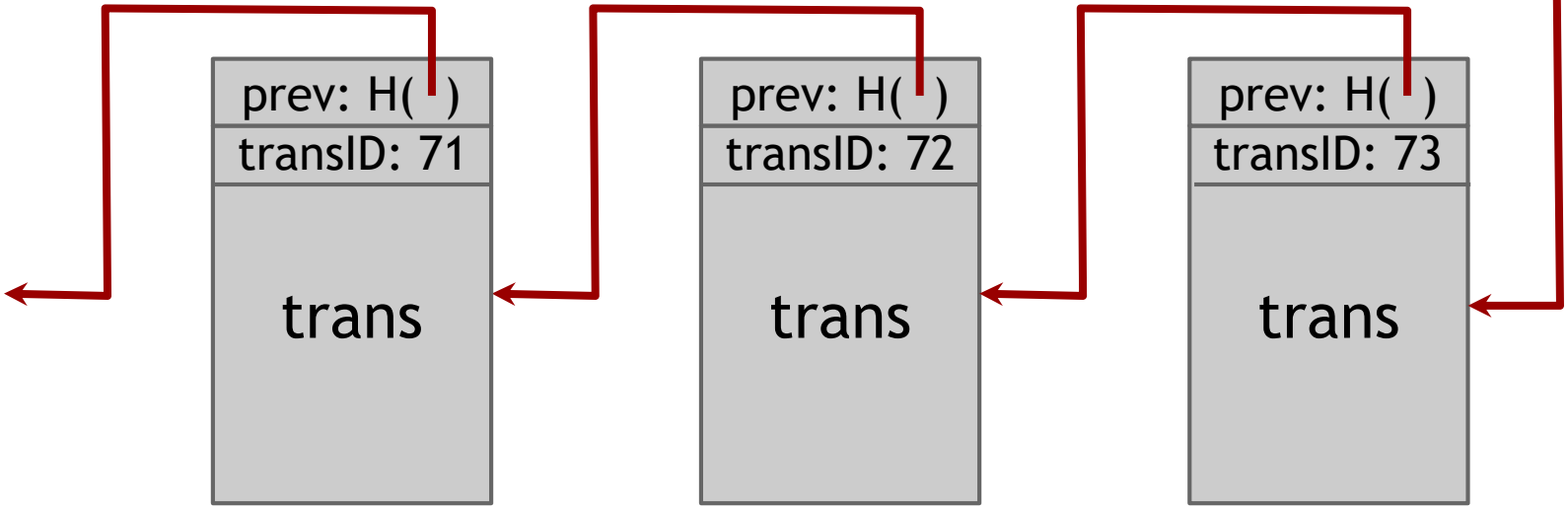
to subdivide: create new trans

consume your coin

pay out two new coins to yourself

Scrooge publishes a history of all transactions
(a block chain, signed by Scrooge)

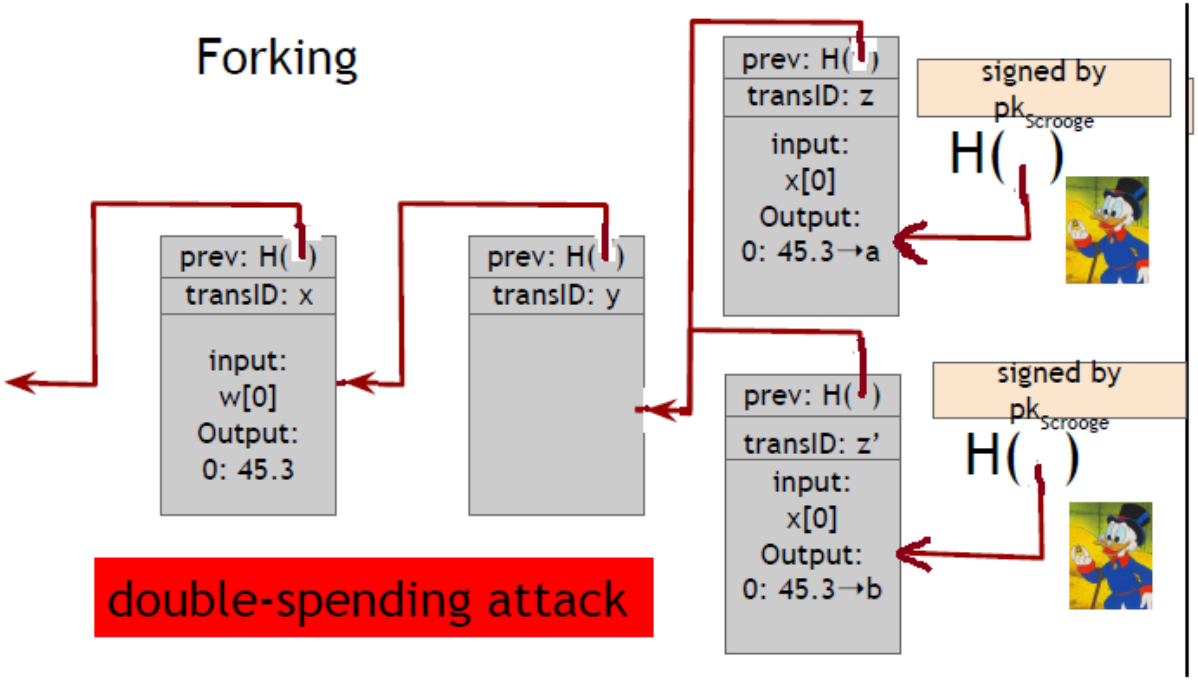
H( )



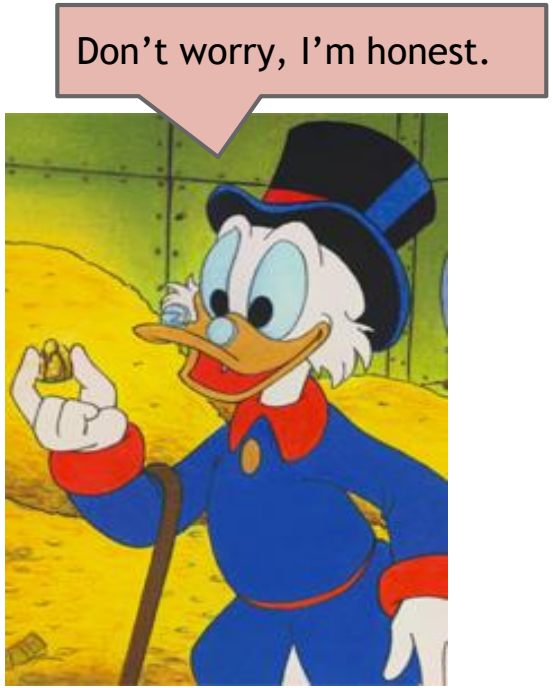
optimization: put multiple transactions in the same block

Don’t worry, I’m honest.

Forking



What if Scrooge is malicious?



Crucial question:

Can we descroogify the currency, and operate without any central, trusted party?

Bitcoin's approach: global ledger