“Centralized” Bitcoins

Prof. Francesco Bergadano

Dipartimento di Informatica
Università di Torino

Summary

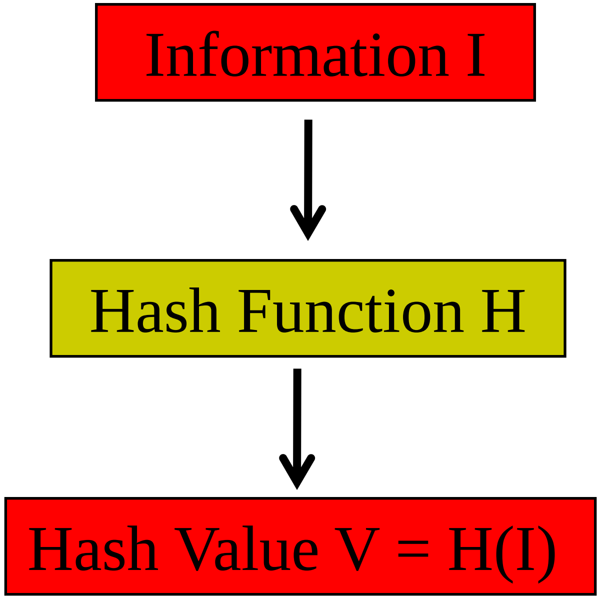
- Basic concepts and tools
- Simple bitcoin scenario
- A centralized Bolockchain

Basic concepts and tools

- One-way hash functions
- Hash pointers & Hash chains
- Public keys as identities (“addresses”)

One-way hash functions

Hash function: we will use collision-resistant hash functions, hence “one-way”



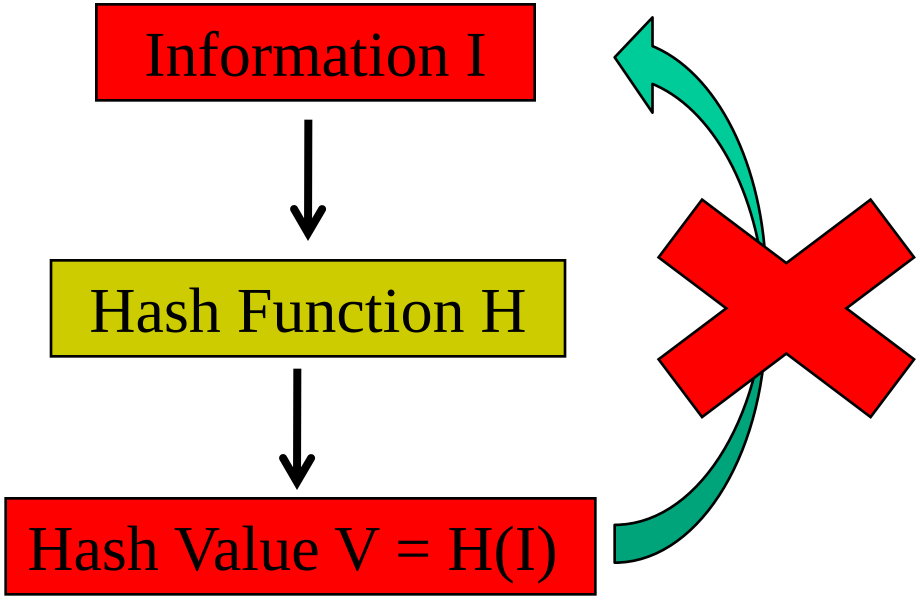
Information I

Hash Function H

Hash Value V = H(I)

One-way hash functions

Hash function: we will use collision-resistant hash functions, hence “one-way”



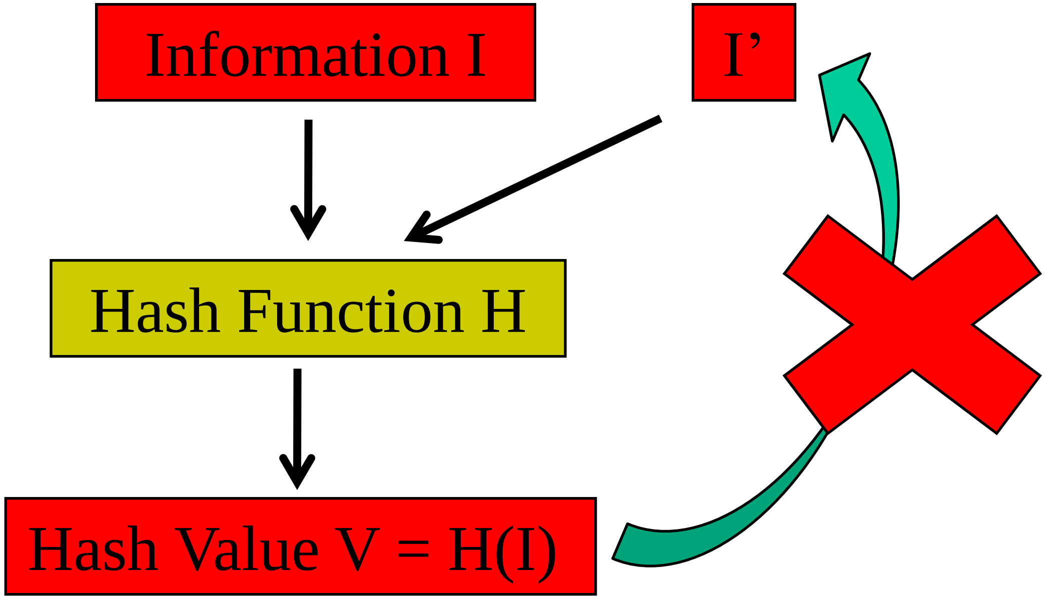
Information I

Hash Function H

Hash Value V = H(I)

One-way hash functions

Hash function: we will use collision-resistant hash functions, hence “one-way”



One-way hash functions

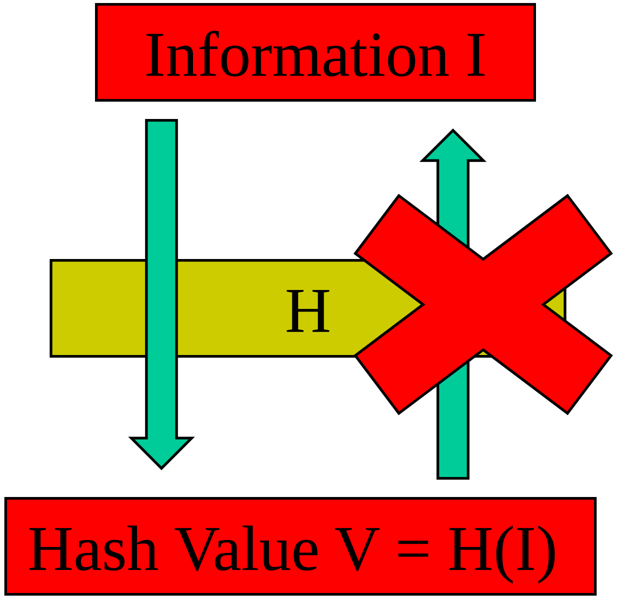
When H is “one-way”, we have that:

*given V, it is practically impossible**
to find I such that H(I) = V

When |V|=k sufficiently large:
a brute force approach will require 2^k hash computations,
for every trial we generate a random input I and compute H(I)

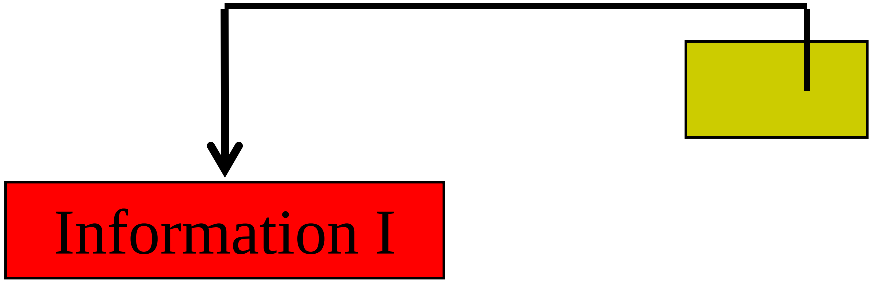
One-way hash functions

Hash function: we will use collision-resistant hash functions, hence “one-way”



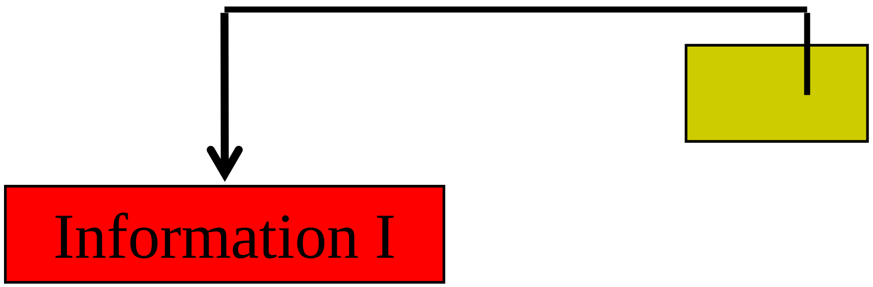
Hash pointers

A pointer:



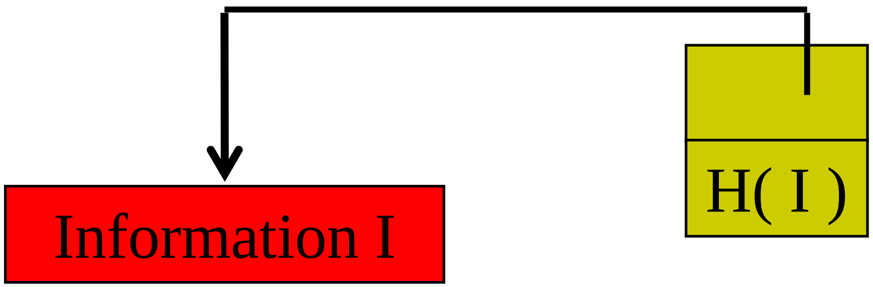
Hash pointers

A pointer:



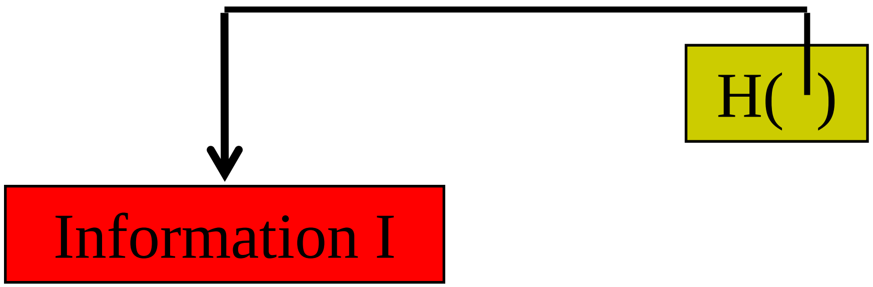
Hash pointers

A hash pointer:



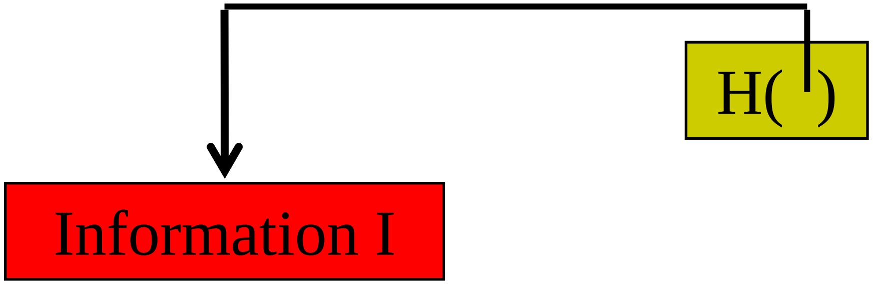
Hash pointer notation

A hash pointer*:



** this is a visualization, equivalent to the previous slide*

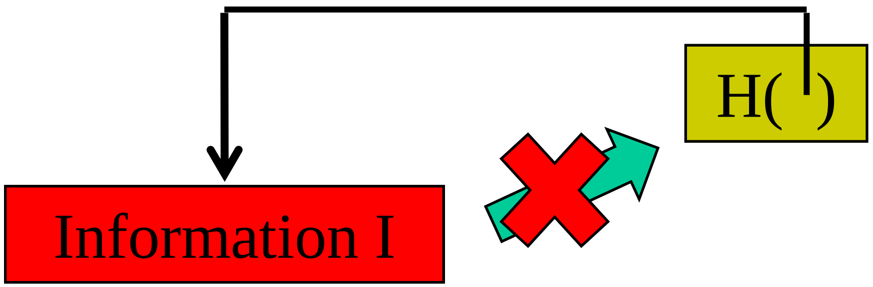
Hash pointer properties



(1) Given the hash pointer, it is practically impossible to generate an information block that makes the pointer valid

(2) If the information I is changed, the hash pointer is no longer valid

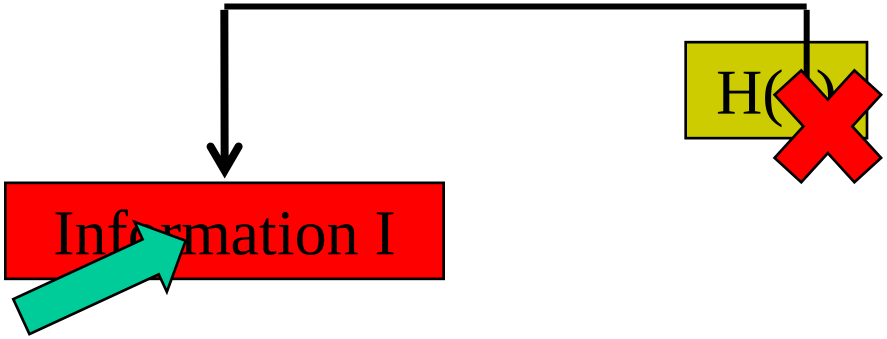
Hash pointer properties



➡ (1) Given the hash pointer, it is practically impossible to generate an information block that makes the pointer valid

(2) If the information I is changed, the hash pointer is no longer valid

Hash pointer properties

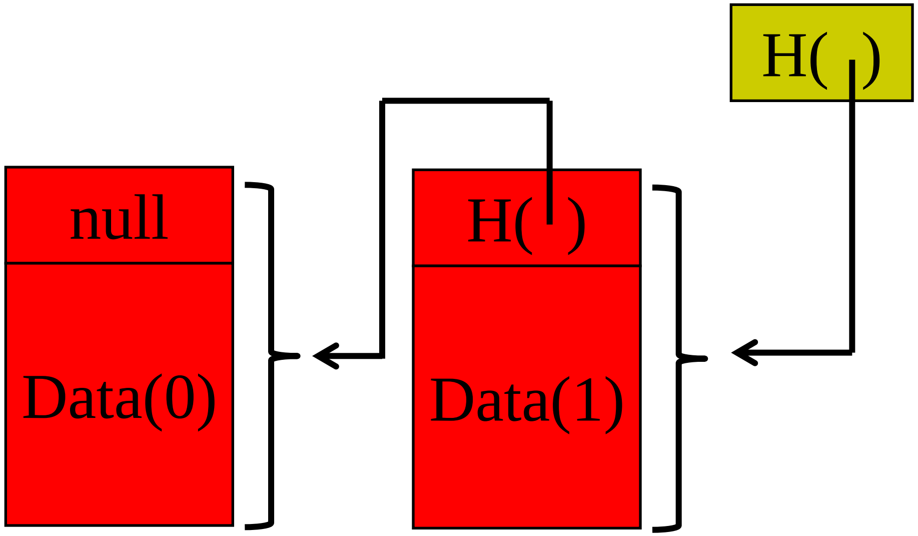


(1) Given the hash pointer, it is practically impossible to generate an information block that makes the pointer valid

➡ (2) If the information I is changed, the hash pointer is no longer valid

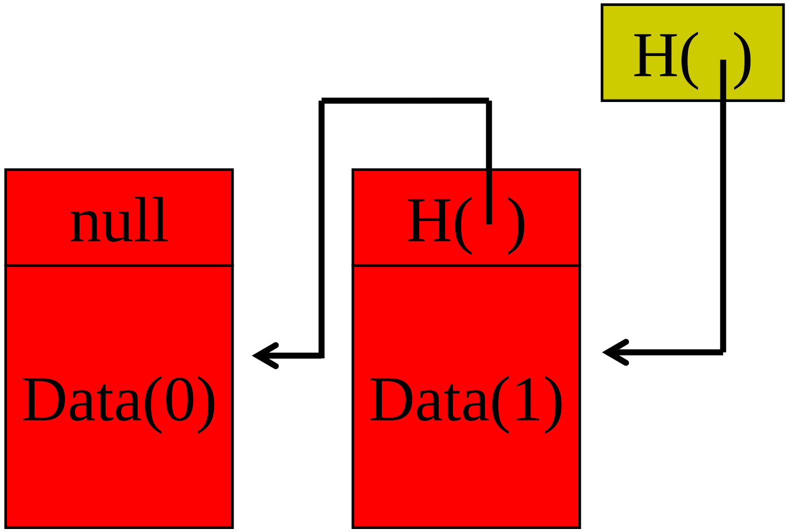
Hash chains

A two-element hash chain:



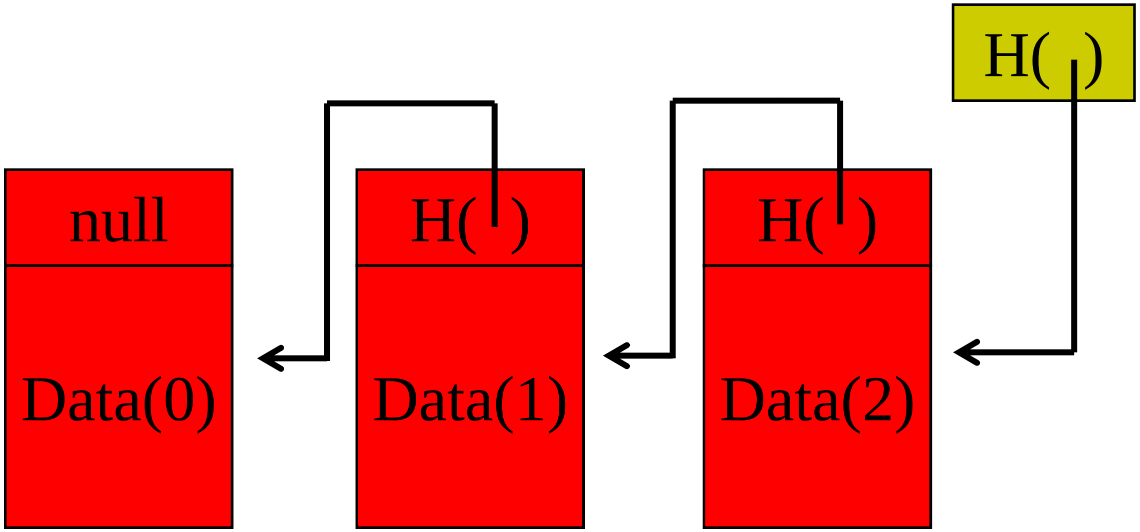
Hash chains

Simplified notation:



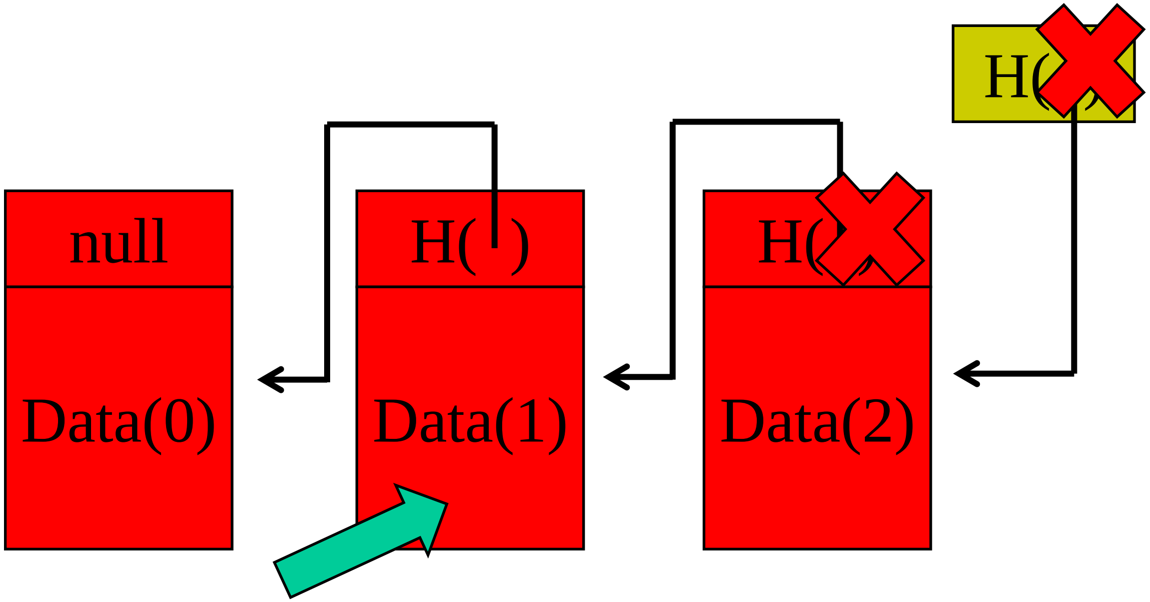
Hash chains

Adding an element:



Hash chains

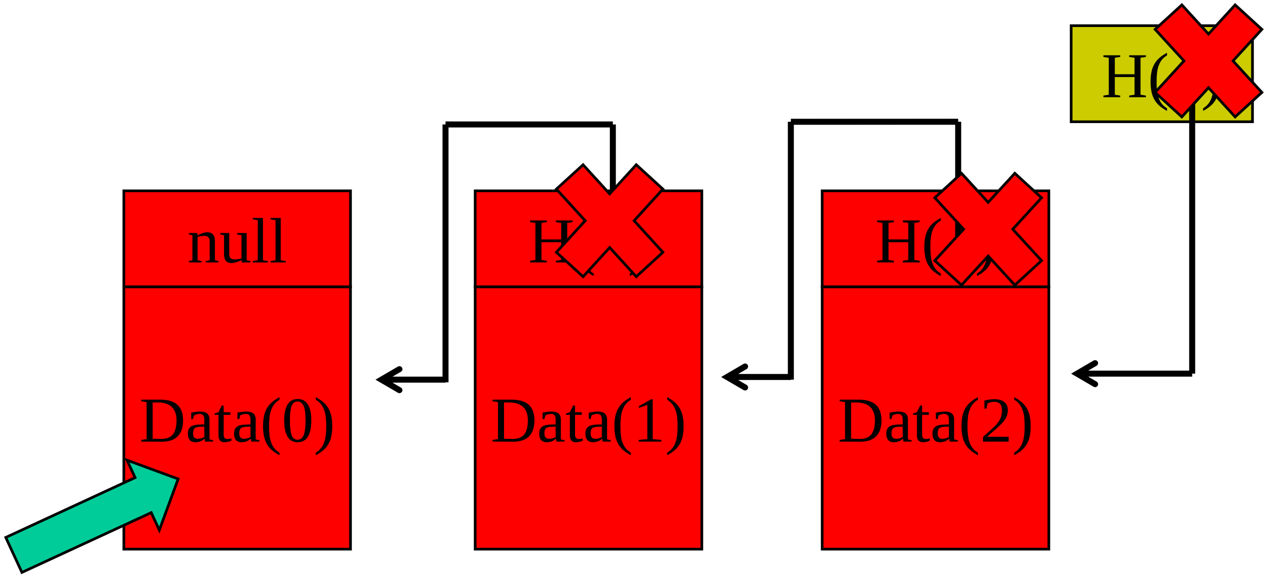
Property:



Any change in Data causes invalid hash pointers

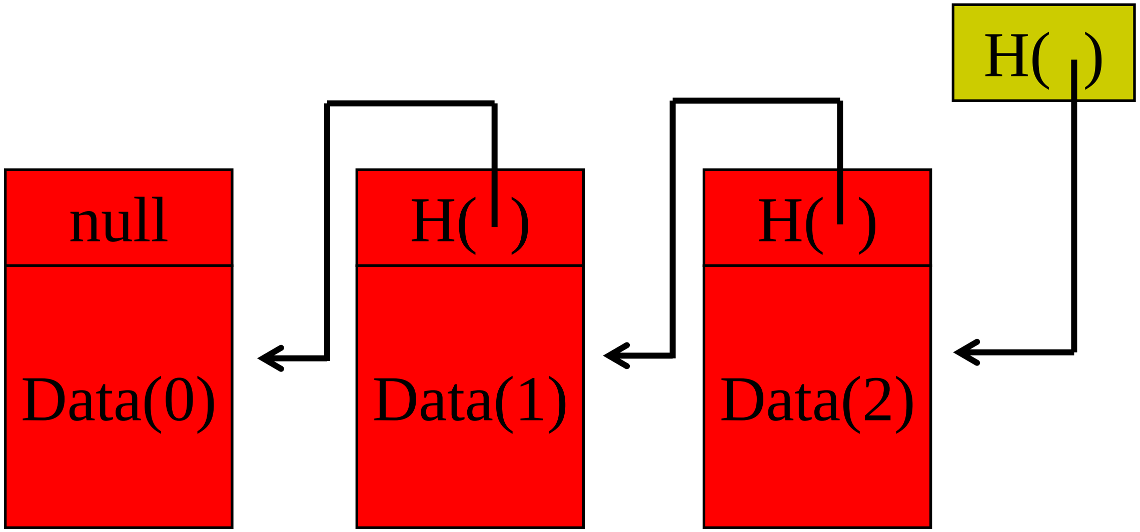
Hash chains

Property:

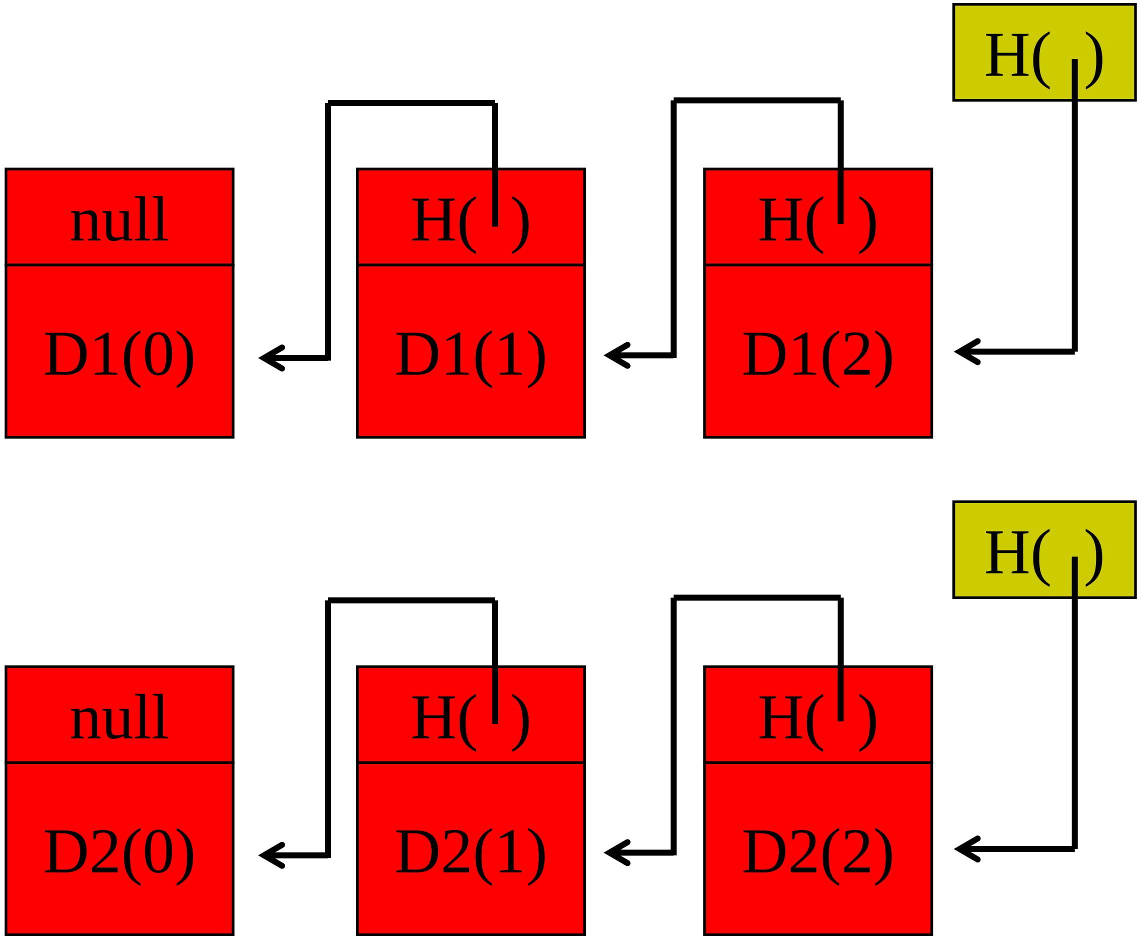


Any change in Data causes invalid hash pointers

Hash chains can be used as a *tamper-evident log*

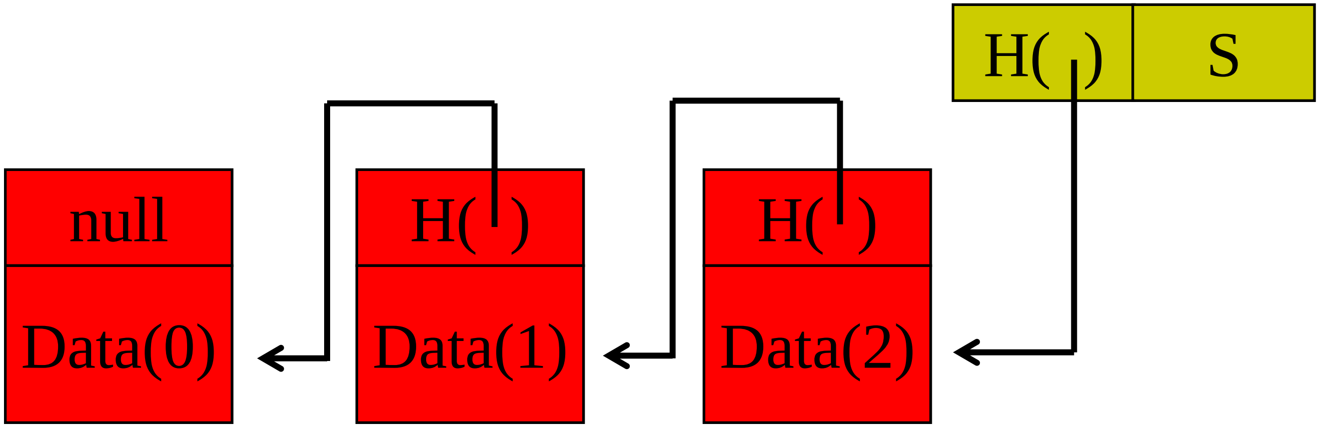


… but one could make a *new* chain



A “signed” chain

In order to make the chain unforgeable …



… the hash pointer is signed, using some private key K^-, and the validity of the chain can be checked using the public key K^+

Public keys as identities

In order to generate an identity:

- Generate a key pair <K^+,K^->
- Publish K^+or H(K^+) in any appropriate way
- Start using K^- to sign any information you like

Public keys as identities

“think” as follows:

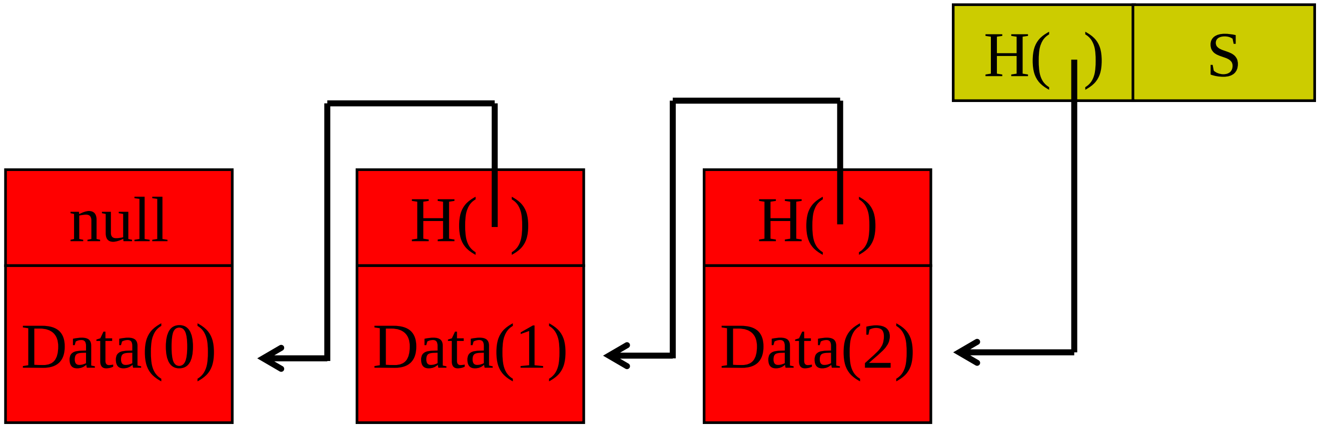
- K^+ is the name of the identity
- Identity K^+ “says” things by signing messages using key K^-
- Hence, <M,Sig(M)> means “K^+ says M”, if Sig(M) is a valid signature for keypair <K^+,K^-> and message M

Public keys as identities

Remarks:

- One can generate as many identities as desired
- No certificates, no third parties needed
- Identities defined as public keys can be anonymous (they are called “addresses” in bitcoin jargon)

Anonymous public keys as identities – sample applications

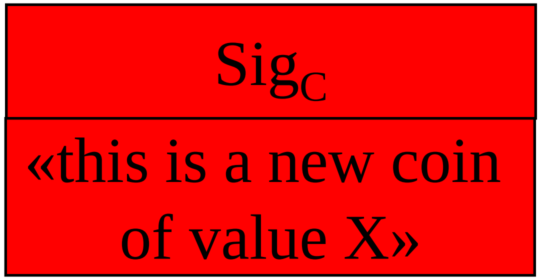


- Start a blog … keep posting (you do not know who I am, but I am the “same person”)
- Maintain an unforgeable log
- Bitcoins (to be discussed)

A simple bitcoin scenario

- Bitcoins as signed data
- A simple, non-secure scenario
- A centrally managed currency

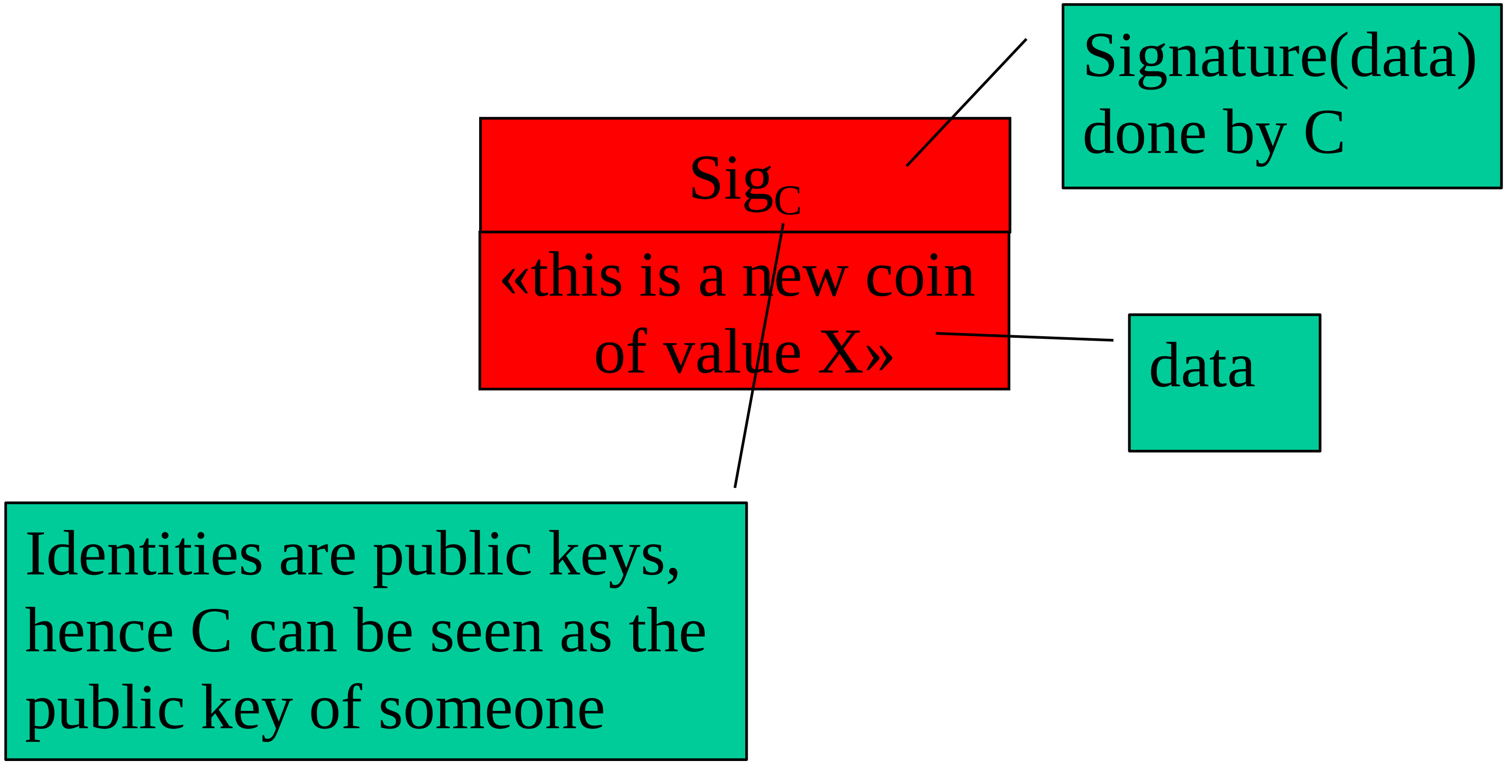
Bitcoins as signed data



Sig_C

«this is a new coin of value X»

Bitcoins as signed data



Signature(data) done by C

Sig$_C$

«this is a new coin of value X»

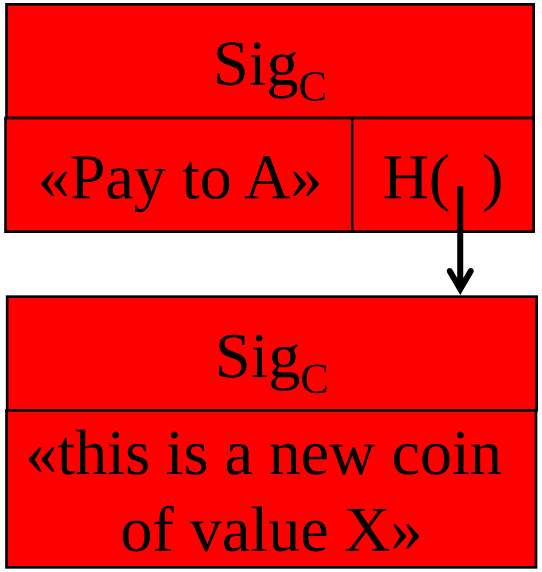
data

Identities are public keys, hence C can be seen as the public key of someone

A simple, non-secure scenario

- Anyone can generate new coins of some type and value
- Coins are “payed” to someone else by signing a transaction
- The recipient can then “pay” the coin to someone else in the same way

“Paying” Bitcoins: C pays A



Sig_C

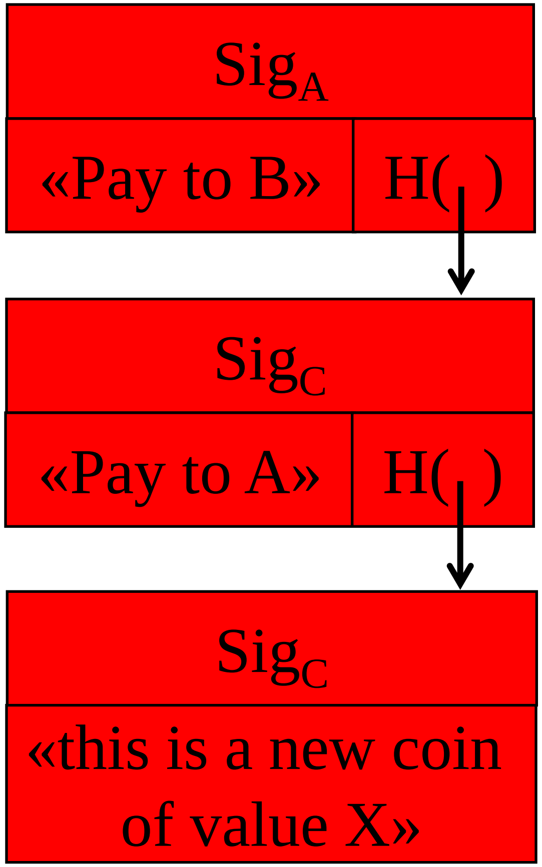
«Pay to A»

H( )

Sig_C

«this is a new coin of value X»

“Paying” Bitcoins: A pays B



Sig_A

«Pay to B» H( )

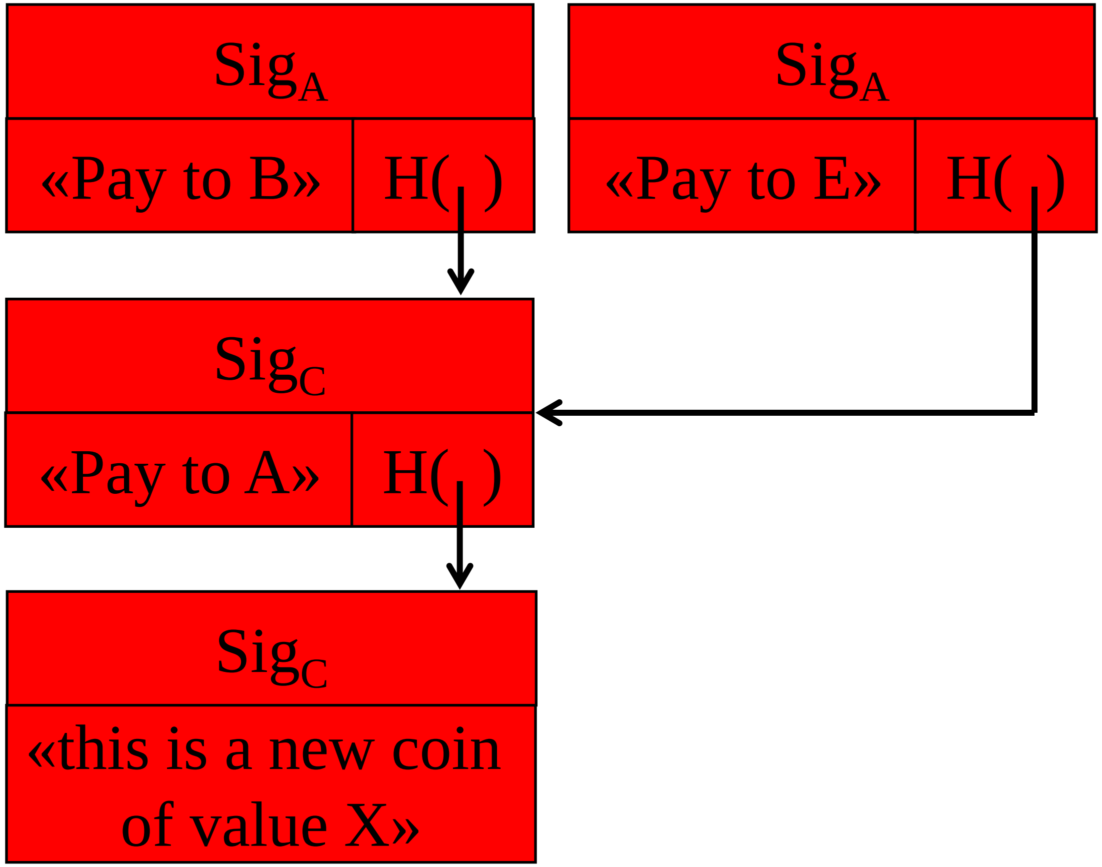
Sig_C

«Pay to A» H( )

Sig_C

«this is a new coin of value X»

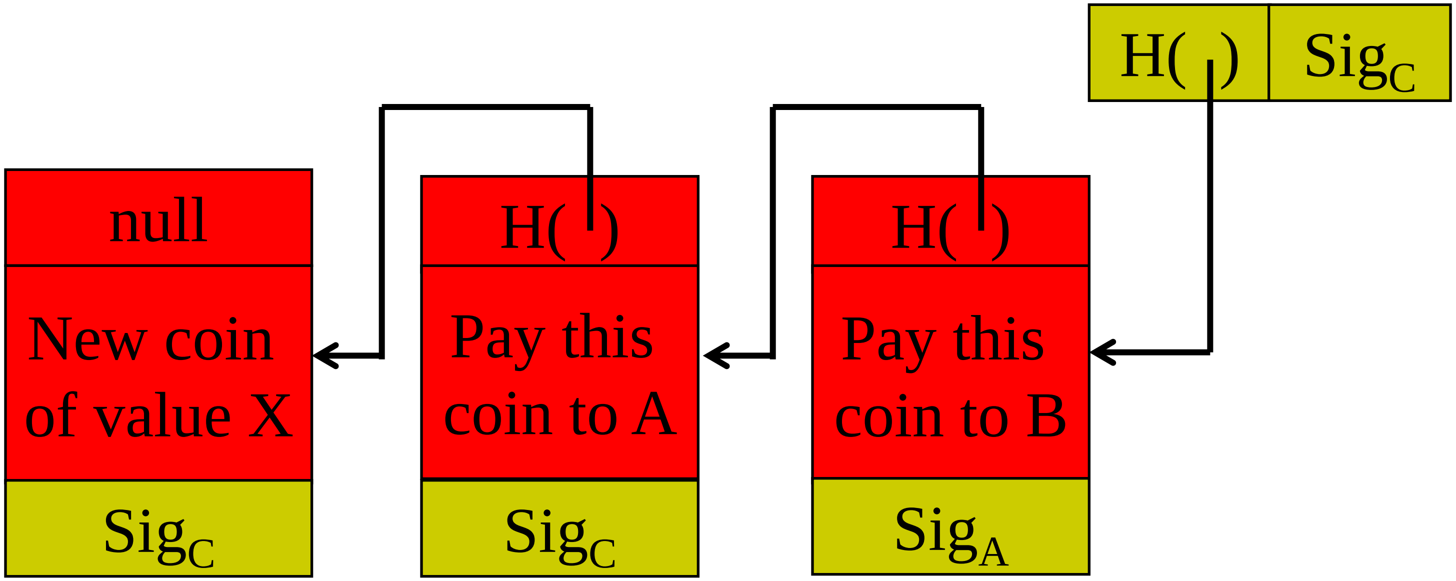
Double spending



A centrally managed bitcoin system, avoiding double spending

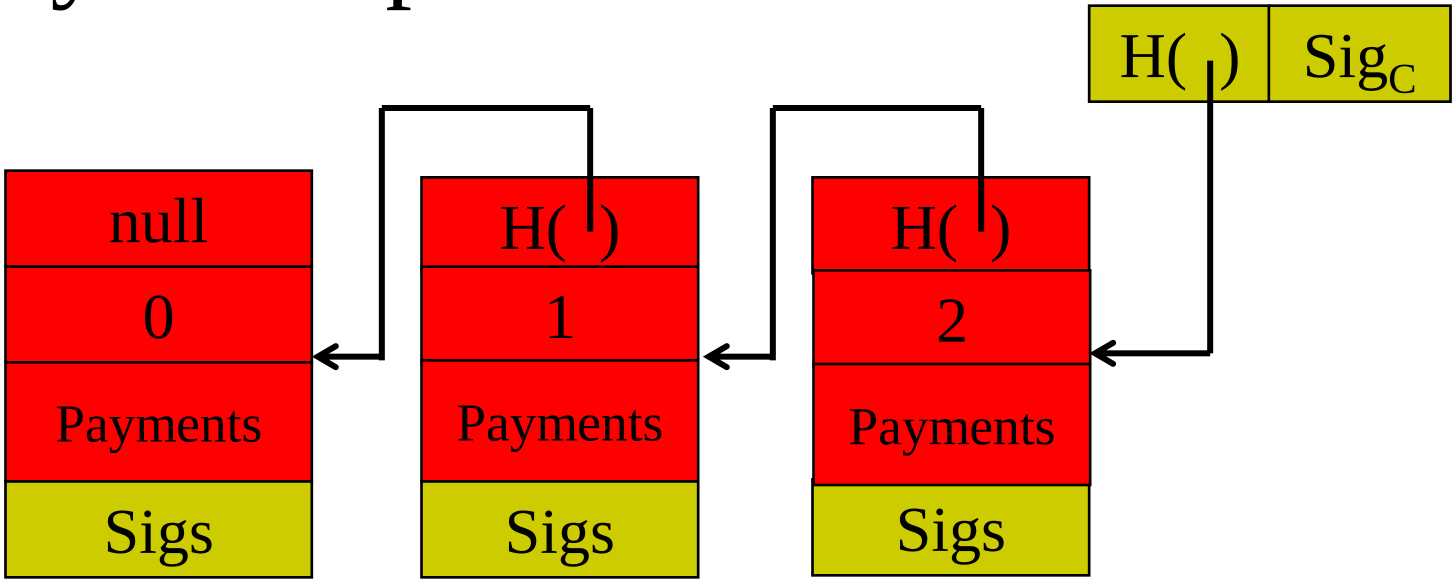
- The chain of transactions are signed by a central authority
- The central authority can be the same identity that generates new coins
- Everyone must trust the central authority

A “signed” chain of payments



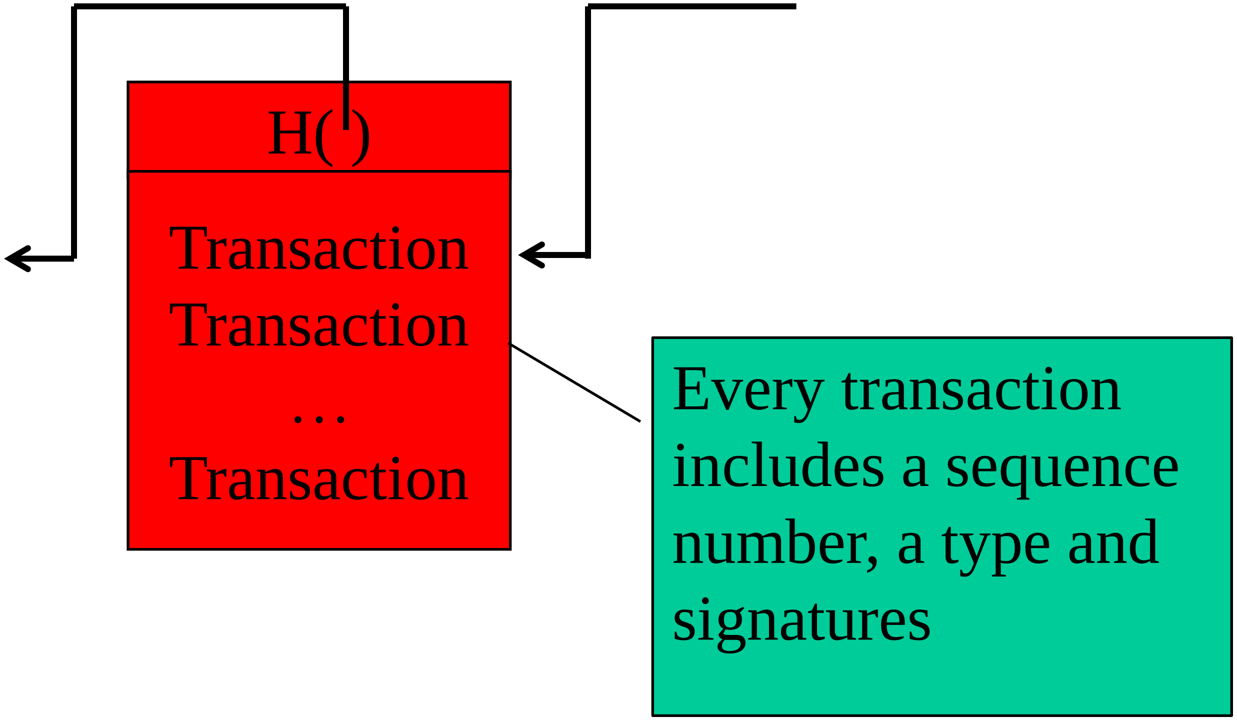
- C creates new coins and maintains the chain
- To pay B, A signs the newest block, submits to C
- C checks the signatures and absence of double spending, then adds the block to the chain

Optimization example: multiple payments per block



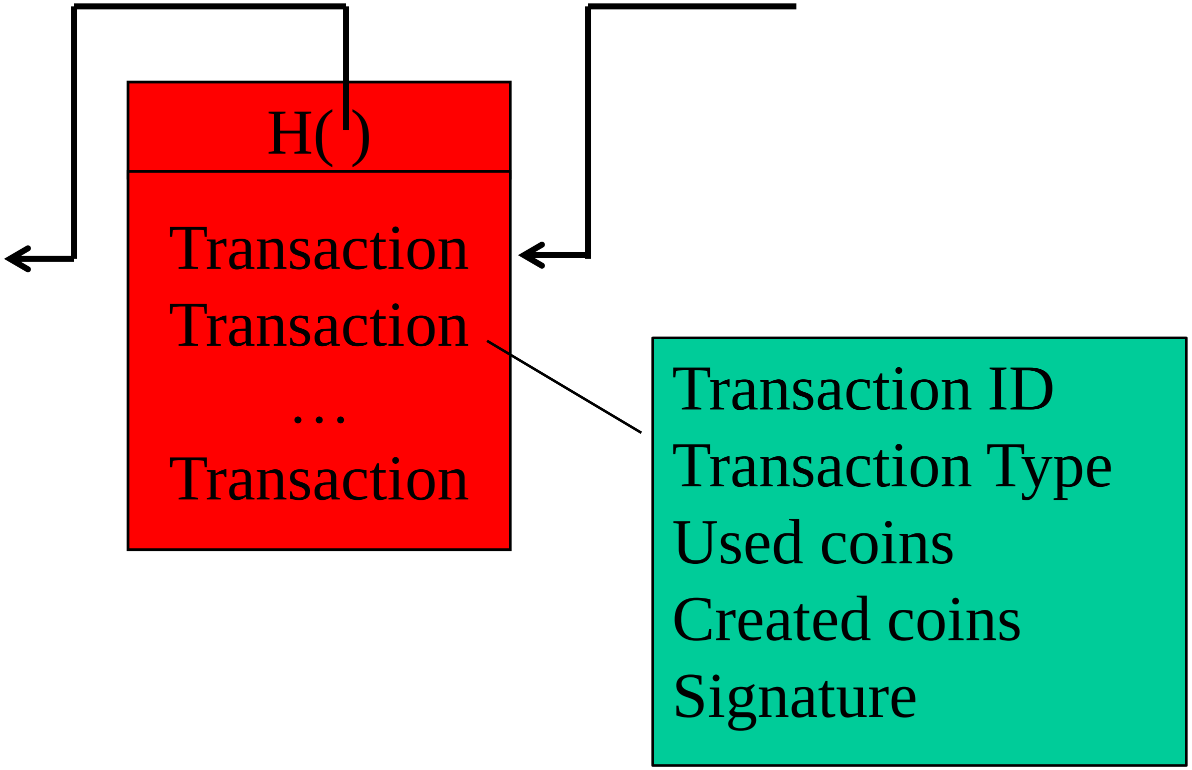
- Every block has a sequence number
- There can be many payments in each block, hence many signatures if done by different identities

Further optimization: transactions with IDs, types and signatures



Every transaction includes a sequence number, a type and signatures

A number of coins are used and created in each transaction



H( )

Transaction
Transaction
…
Transaction

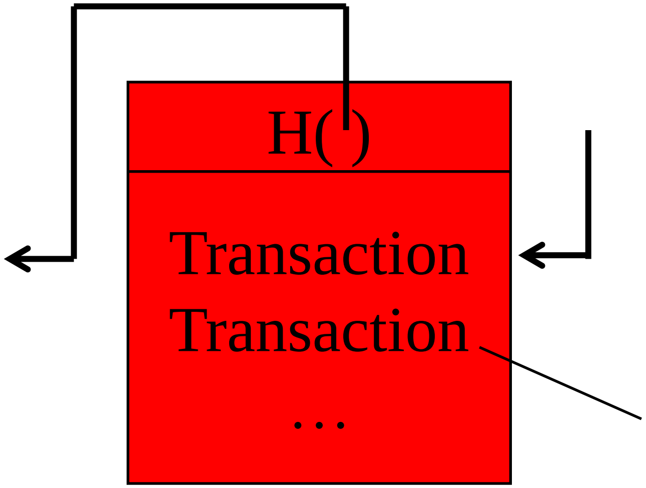
Transaction ID
Transaction Type
Used coins
Created coins
Signature

Types of transactions

We will use two types of transactions for the time being:

- “create coins”: new coins are created (signed by C)
- “pay coins”: some identity A uses coins that she owns to create and pay new coins to other identities (signed by A)

“Create coins” transaction



H( )

Transaction

Transaction

…

Transaction ID: 432

Transaction Type: create_coins

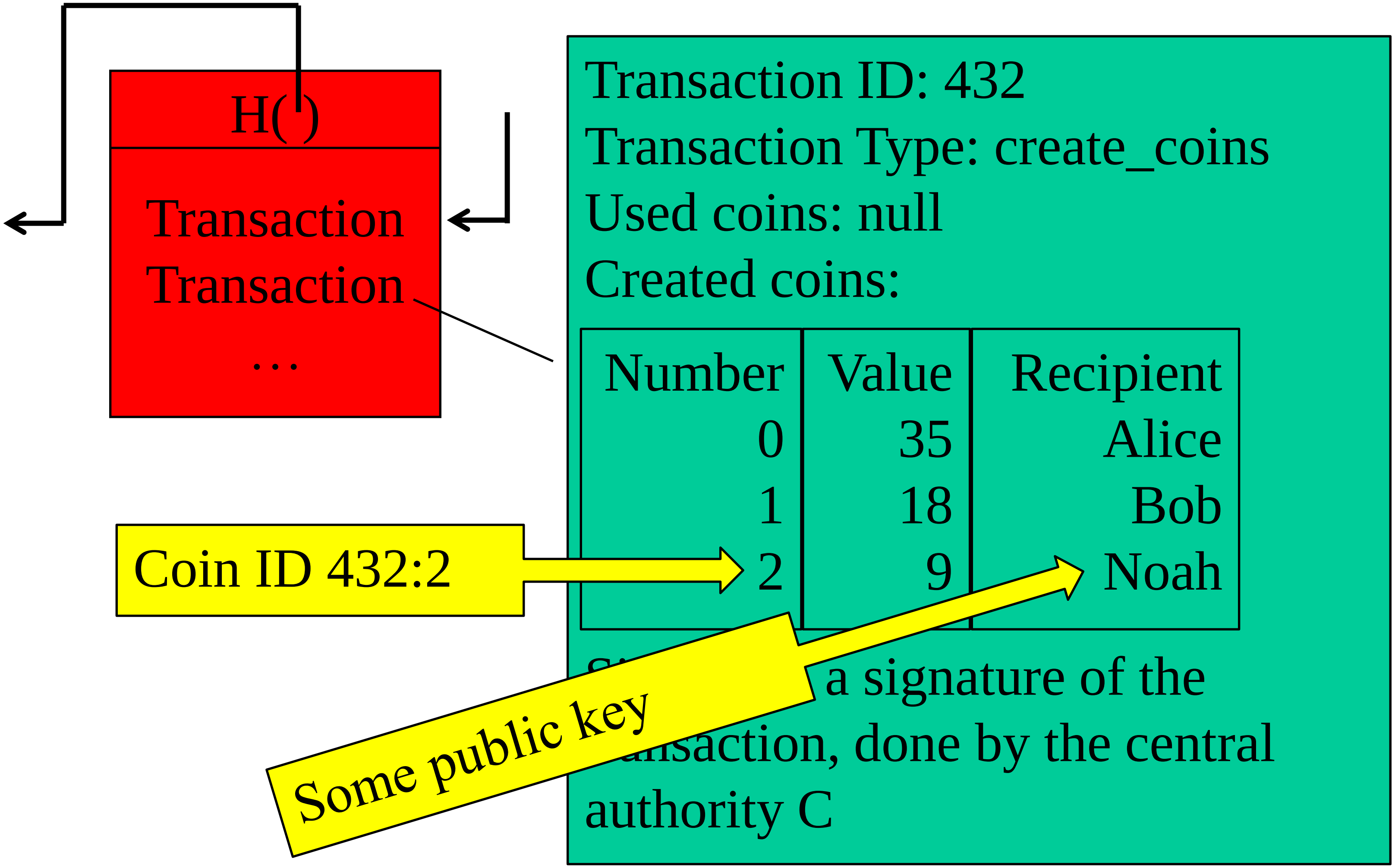
Used coins: null

Created coins:

Number	Value	Recipient
0	35	Alice
1	18	Bob
2	9	Noah

Signature: a signature of the transaction, done by the central authority C

“Create coins” transaction



Transaction ID: 432

Transaction Type: create_coins

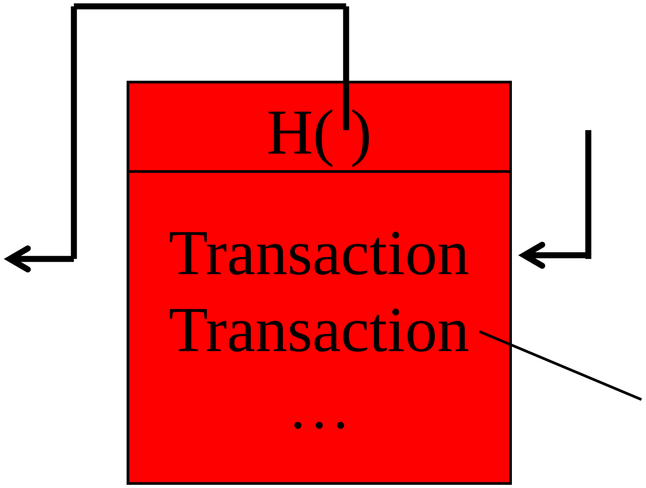
Used coins: null

Created coins:

Number	Value	Recipient
0	35	Alice
1	18	Bob
2	9	Noah

Si[illegible] a signature of the [illegible]ansaction, done by the central authority C

“Pay coins” transaction



Transaction ID: 1732

Transaction Type: pay_coins

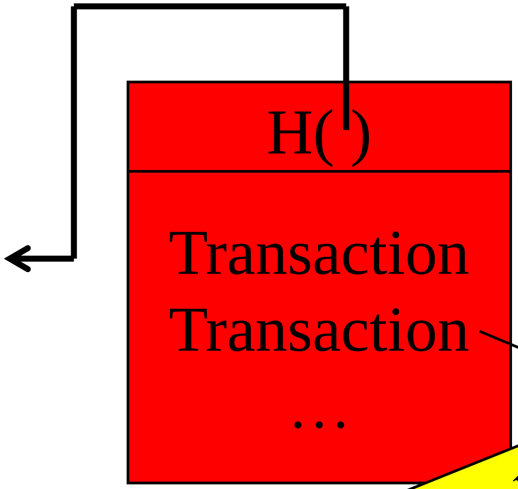
Used coins: 3:82,432:0, 1729:6

Created coins:

Number	Value	Recipient
0	350	Noah
1	43	Sam

Signature: a signature of the transaction, done by A. the owner of the used coins

“Pay coins” transaction



Transaction ID: 1732
Transaction Type: pay_coins
Used coins: 3:82,432:0, 1729:6
Created coins:

Number	Value	Recipient
	350	Noah
1	43	Sam

Signature: a signature of the transaction, done by A. the owner of the used coins

The sum of the values of the created coins must not exceed the sum of the values of the used coins

The centralized bitcoin system

- Coins as correctly signed information
- Only C can create coins
- C manages a tamper-evident blockchain
- Double spending impossible
- Users are anonymous (one could create a new ID and pay coins to his new ID)

Conclusions

- Bitcoins = signed data
- Payments = signed transactions
- To prevent double spending, a central authority was introduced, that is also responsible for creating coins

Next, we remove such a central authority