# MADS

Emmanuelle Anceaume

Lesson 1: Bitcoin and its Distributed Ledger Technology

http://people.irisa.fr/Emmanuelle.Anceaume/

# What is Bitcoin ?

- Bitcoin is a distributed cryptocurrency and payment system
- It allows users to anonymously exchange goods against digital currency
- There are no centralized banking authority
- All the valid transactions are recorded in a public distributed ledger, the blockchain
- Blockchain = organizes partially ordered transactions in a totally ordered sequence with high probability

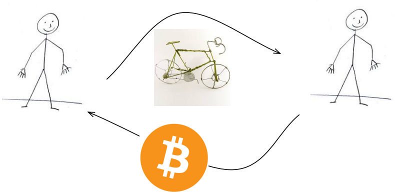
## Ledger

| | | | |
|---|---|---|---|
| Bob | -> | Alice | ฿0.001 |
| Chunk | -> | Sara | ฿0.05 |
| Eva | -> | Alice | ฿0.009 |
| Alice | -> | John | ฿0.02 |
| Bob | -> | Chunk | ฿0.7 |
| Peter | -> | Bob | ฿0.008 |
| Bob | -> | Alice | ฿0.05 |
| Bob | -> | Alice | ฿0.046 |
| Bob | -> | Alice | ฿0.008 |

# What is Bitcoin ?

Ledger

Bob -> Alice Ƀ0.001
Chunk -> Sara Ƀ0.05
Eva -> Alice Ƀ0.009
Alice -> John Ƀ0.02
Bob -> Chunk Ƀ0.7
Peter -> Bob Ƀ0.008
Bob -> Alice Ƀ0.05
Bob -> Alice Ƀ0.046
Bob -> Alice Ƀ0.008



So who maintains this ledger and makes sure no one is cheating ?

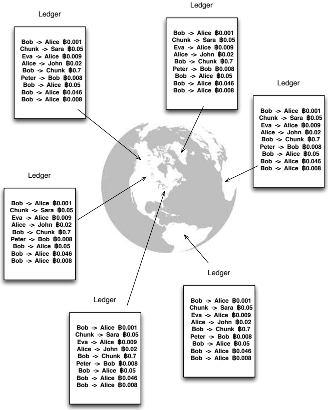
# What is Bitcoin ?

No centralized control

- everyone maintains their own copy of the ledger
- everyone can see all the transactions of the system

How synchronizing money transfers ?

- when Alice spends some money she diffuses that information everywhere
- everyone updates its copy of the ledger




How preventing account thief ?
How preventing double-spending attacks ?
How is money created ?

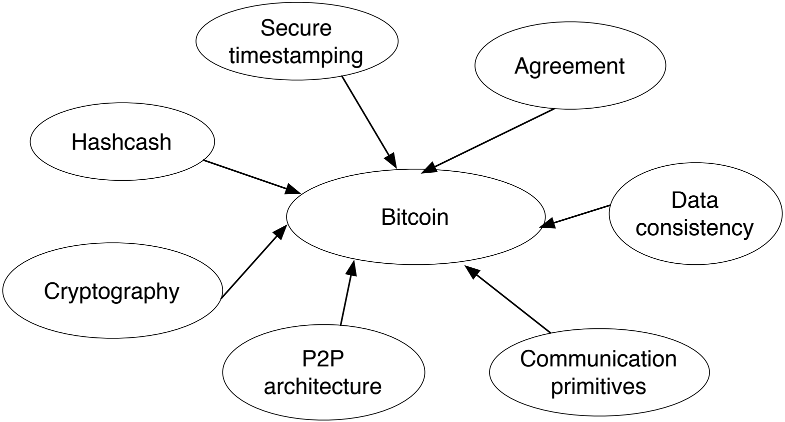
...

# Basic principles

- Crypto currency
  - relies on cryptographic tools
- Decentralized system
  - peer-to-peer architecture
- Trustless model
  - does not require a central server to validate/abort financial transactions but requires participants to be online
- Anonymous users
  - neither sellers nor buyers use their real identities to use Bitcoins but if you are not careful your transactions can be tied together

Satoshi Nakamoto. Bitcoin : A Peer-to-Peer Electroni Cash System. October 2008, http ://nakamotoinstitute.org/bitcoin/

# Bitcoin relies on a set of distributed algorithms



# Content of this lesson

- Crypto background
  - hash functions
  - digital signatures
  - hash pointers
  - Merkle trees
- Bitcoin principles
  - Peer-to-peer networks
  - Transactions
  - Blocks

# Preliminaries on crypto

- cryptographic hash functions
- digital signatures
- Merkle tree

All currencies need some way to control supply and prevent counterfeiting money

- Fiat currencies (Dollar, Euro, Yen, Yuan)
  - central banks mint physical currency
  - integrity of bank notes is guaranteed by anti-counterfeiting features to physical currency
- Digital currencies
  - a string of « 0 » and « 1 »
  - no central bank to prevent double-spending attacks
  - heavy use of cryptography

# Hash functions

A hash function is an algorithm that allows to compute a fingerprint of fixed size from data of arbitrary size

$$\begin{aligned} H : 0,1^* &\rightarrow 0,1^n \\ M &\mapsto H(M) \end{aligned}$$

Applications : make easier the management of databases

- rather than manipulating data of arbitrary size, a fingerprint is associated to each data which makes operation easier
- comparison, membership ...
  - Bloom filters = bit array
  - Count-min = Counting the number of occurrences of elements
  - Protecting data
  - ...

# Hash functions

A hash function satisfies the following properties

- The input space is the set of string of arbitrarily length
  - « hello world » and « hellohellohello world » are perfectly fine inputs
- The output space is a set of strings of fixed length
  - H(« hello world ») = 000223
  - H(« hellohellohello world ») = 130554
- H is deterministic
- H is efficiently computable
  - Given a string $s$ of length $n$ the complexity to compute $H(s)$ is $O(n)$

In addition to these properties, crypto-hash functions have additional requirements

# Properties of cryptographic hash functions

- Collision resistance

  It must be difficult to find two inputs $x$ and $x'$ such that $H(x) = H(x')$

- Second pre-image resistance

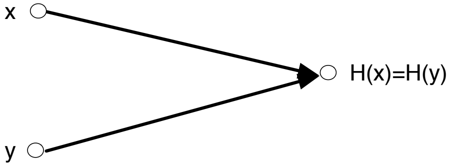
  Given an input $x$, it must be difficult to find an input value $x' \neq x$ such that $H(x') = H(x)$

- Pre-image resistance

  Given $z$, it must be difficult to find an input value $x$ such that $H(x) = z$

# Collision resistance

Find two inputs $x$ and $x'$ such that $H(x) = H(x')$



collisions do exist

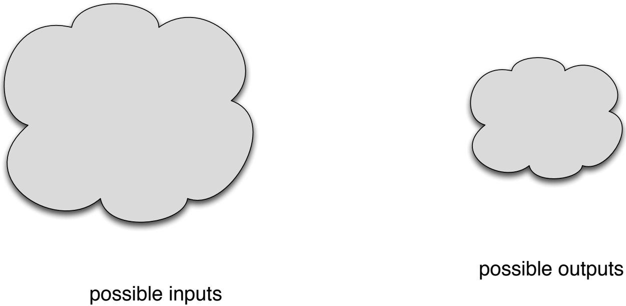



Image source: *Bitcoin and Cryptocurrency Technologies*.

# Collision resistance

collisions do exist

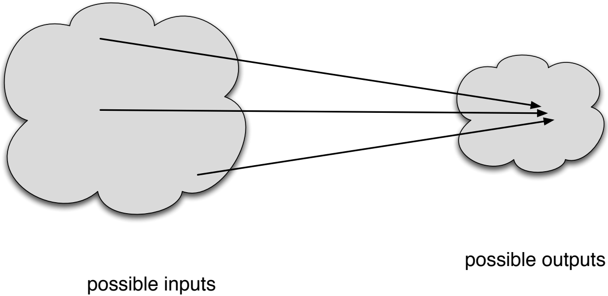



Image source: *Bitcoin and Cryptocurrency Technologies*.

but can anyone find them ?

# Collision resistance property

Find two inputs $x$ and $x'$ such that $H(x) = H(x')$

Generic attack (i.e., a technique capable of attacking any $n$-bit hash function )

- Choose $2^{n/2}$ random messages (birthday paradox)
- Compute the hashed values and store them
- Find one pair (x,x') such that $H(x) = H(x')$

# Birthday paradox

Birthday paradox is about the probability that, in a set of $m$ randomly chosen people, some pair of them will have the same birthday.

- if $m = 23$ the probability to have collision is 50%
- if $m = 70$ then $p$ is equal to 99.9%

# Birthday paradox

Let us first compute the probability that no two persons have the same birthday. Let $p'(m$ be this probability

$$
\begin{aligned}
p'(m) &= \frac{365}{365}\frac{364}{365}\cdots\frac{365-(m-1)}{365} \\
&= \frac{365!}{(365-m)!}\frac{1}{365^m}
\end{aligned}
$$

Thus the probability $p(m)$ that there exists two persons having the same birthday is

$$
\begin{aligned}
p(m) &= 1 - p'(m) = 1 - \frac{365!}{(365-m)!}\frac{1}{365^m} \\
&\simeq 1 - e^{-\frac{m(m-1)}{2\times 365}}
\end{aligned}
$$

Thus

$$
m(p) \simeq \sqrt{2 \times 365 \times ln\frac{1}{1-p}}
$$

# Birthday paradox

$$m(p) \simeq \sqrt{2 \times 365 \times ln \frac{1}{1-p}}$$

we get

$$m(0.5) \quad = \quad 23$$

In our case, the set of possible values is equal to $2^n$ with $n$ the length of the binary string of the fingerprint
Thus

$$\begin{aligned} m(0.5) &\simeq \sqrt{2 \ln 2}\, 2^{N/2} \\ &\simeq 2^{N/2} \end{aligned}$$

Find two inputs $x$ and $x'$ such that $H(x) = H(x')$

Generic attack (i.e., a technique capable of attacking any hash function)

- Choose $2^{n/2}$ random messages
- Compute the hashed values and store them
- Find one pair $(x, x')$ such that $H(x) = H(x')$

If a computer calculates $10,000$ hashes/s

- it would take $10^{27}$ years to output $2^{128}$ hashes, and
- thus $10^{27}$ years to produce a collision with probability $1/2$

Astronomical number of computations!!

So far no hash functions have been proven to be collision resistant

# Collision resistance property

To summarize :

Collision resistant hash functions allows us

- to identify data by its hashed value (i.e digest, fingerprint)
  - if $H(x) = H(y)$ then it is safe to assume that $x = y$
- Bitcoin :
  - to identify blocks in the blockchain
  - to make blocks resistant to tampering (modifying a single bit changes the fingerprint)

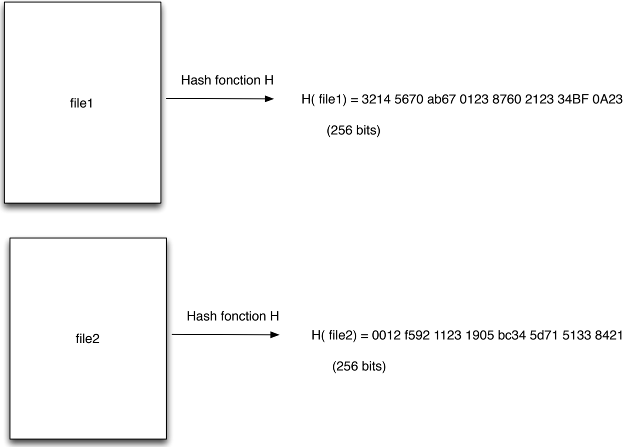
# Second-preimage resistance

Given an input $x$, it is difficult to find an input value $x' \neq x$ such that $H(x') = H(x)$

Generic Attack : probabilistic search

- Given $x$ and its hashed value $H(x)$ ($n$ bits value)
- Randomly choose $x_i$ and compute $z_i = H(x_i)$
- $\text{Proba}(z_i = H(x)) = 1/2^n$
- Thus after having chosen $2^n$ inputs it is likely that one can find a pre-image $x_i \neq x$ such that $H(x_i) = H(x)$

# File integrity




Property : It is difficult to build two files with same fingerprint

# Preimage resistance

Given $z$, find an input value $x$ such that $H(x) = z$

Generic Attack : probabilistic search

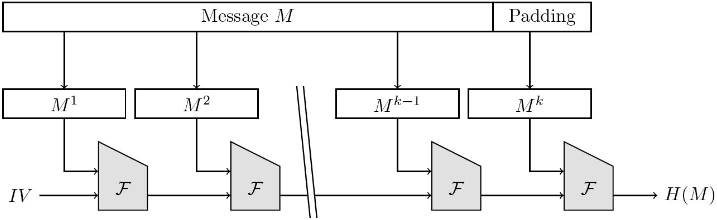
- Given a hashed value $z$
- Randomly choose $x_i$ and compute $z_i = H(x_i)$
- Proba$(z_i = z) = 1/2^n$
- Thus after having chosen $2^n$ inputs it is likely that one can find a pre-image $x_i$ such that $H(x_i) = z$

# Passwords storage

- In your machine, passwords are not stored. Only their hashed value is stored
- When you want to authenticate, the login pg computes the hashed value, which is compared with the one stored in /etc/passwd

Property : Given the hashed value $y$ it must be difficult to find $x$ such that $H(x) = H(password) = y$

# Merkle-Damgard construction



# Additional Properties (Bitcoin)

- Hiding

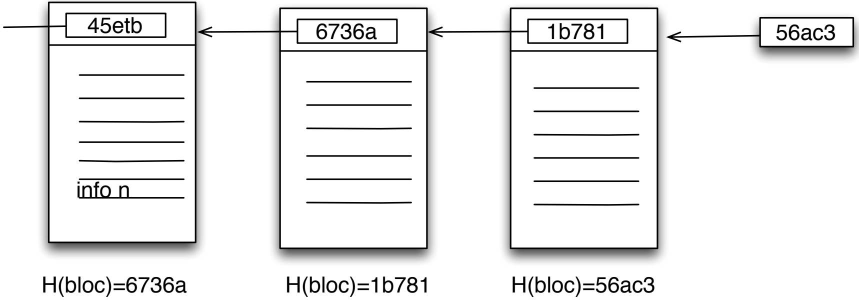
  Given $z$, find the input value $x$ such that $H(x) = z$

- Puzzle-friendliness

  Given $z$, find an input value $x'$ that $H(rx') = z$ with $r$ some random number

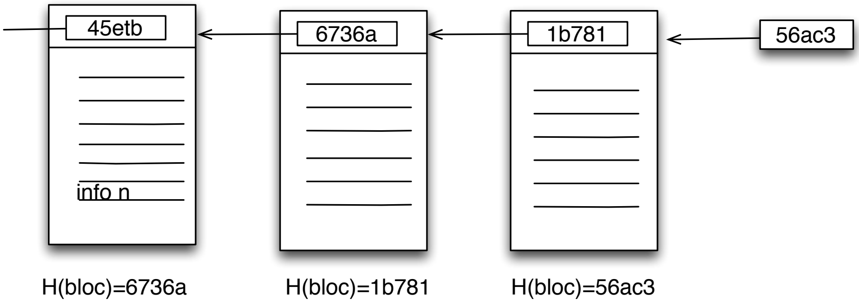
# Hash pointers

A hash pointer is a pointer to where the information is stored together with a cryptographic hash value of the information



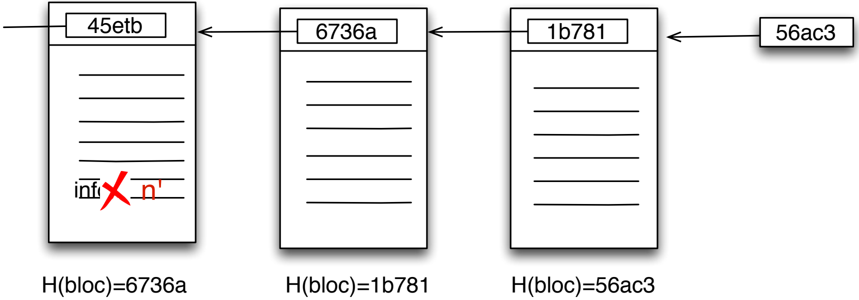
# Hash pointers

Hash pointers allows the construction of a log data structure that allows the detection of any manipulation



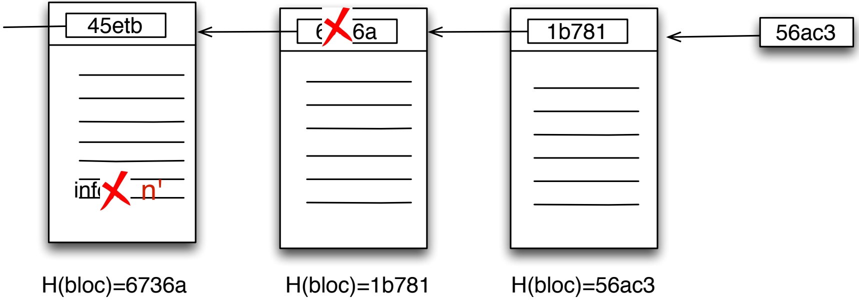
# Hash pointers

Hash pointers allows the construction of a log data structure that allows the detection of any manipulations



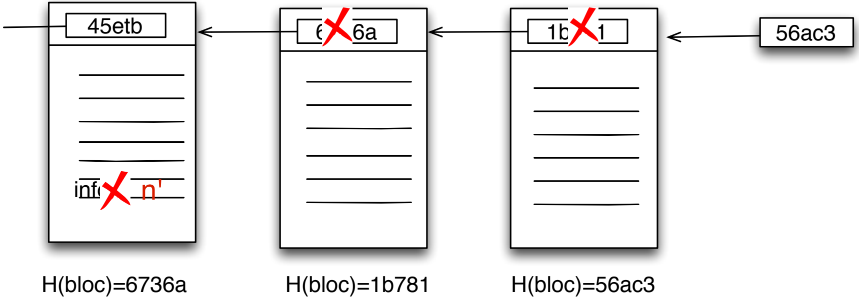
# Hash pointers

Hash pointers allows the construction of a log data structure that allows the detection of any manipulations



# Hash pointers

Hash pointers allows the construction of a log data structure that allows the detection of any manipulations




✓ By only keeping the hash pointer of the head of the data structure, we have a tamper-evident hash of a possibly very long list

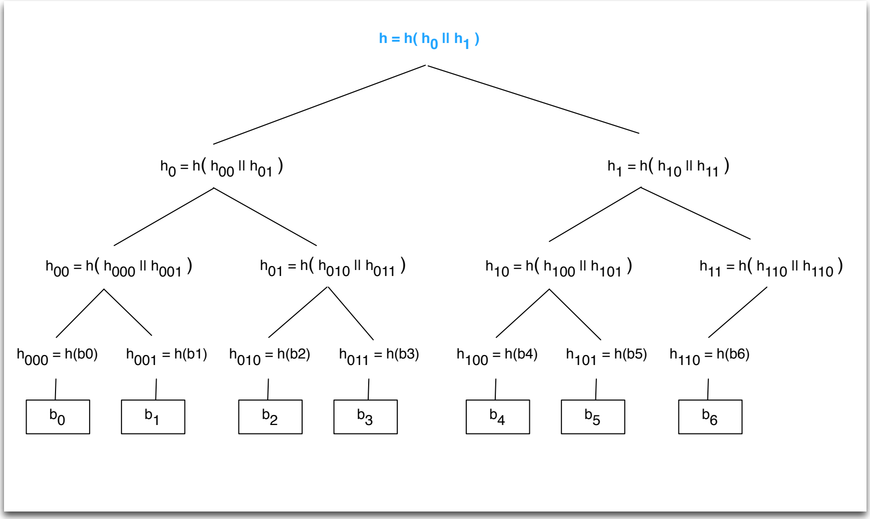
# Hash tree : Merkle Tree

A Merkle tree[1] is a tree of hashes

- Leaves of the tree are data blocks
- Nodes are the hashes of their children
- Root of tree is the fingerprint of the tree

1. Merkle, R. C. (1988). "A Digital Signature Based on a Conventional Encryption Function". Advances in Cryptology - CRYPTO '87.

# Hash tree : Merkle Tree



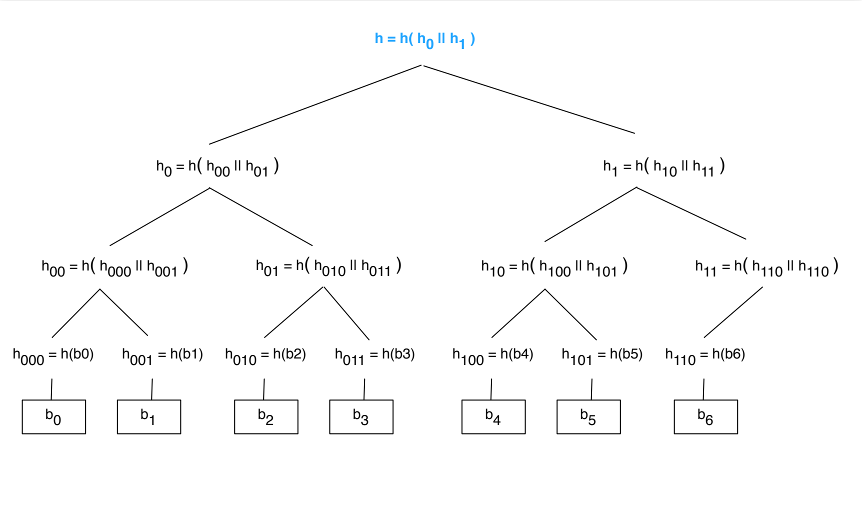
# Hash tree : Merkle Tree

✓ Checking the integrity of the $n$ data blocks of the tree

- easy due to collision resistance property of crypto. hash functions

- Data blocks membership
  - checked with $\log n$ pieces of information and in $\log n$ operations

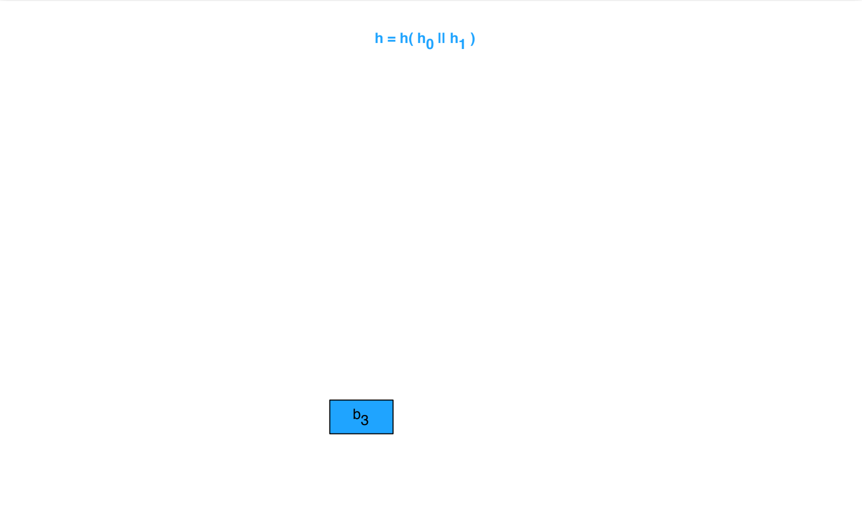
# Hash tree : Merkle Tree



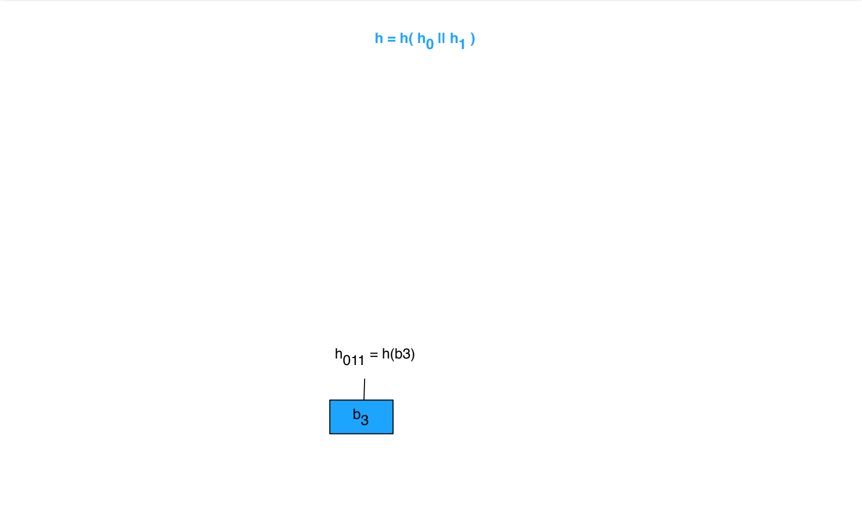
I know the root of the Merkle tree, and I would like to know whether data block $b_3$ belongs to the tree ?

Question : How can I do that without looking for the full tree ?

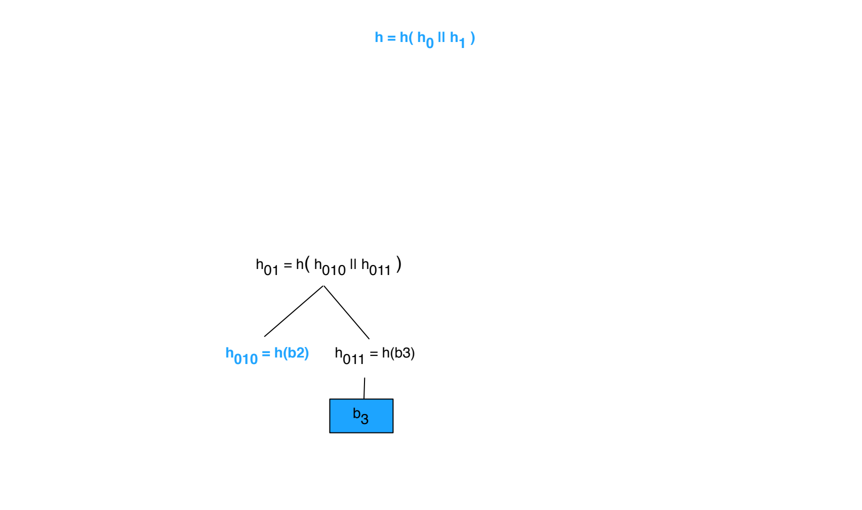
# Hash tree : Merkle Tree



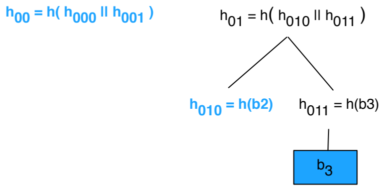
# Hash tree : Merkle Tree



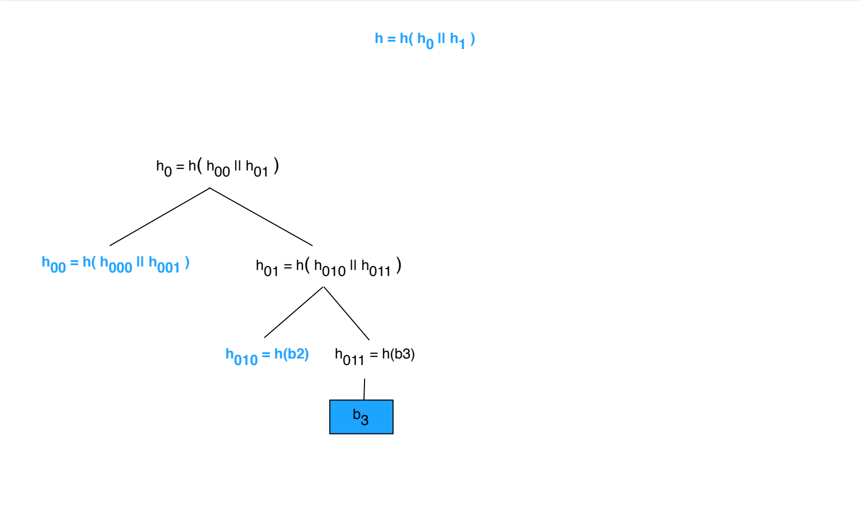
# Hash tree : Merkle Tree



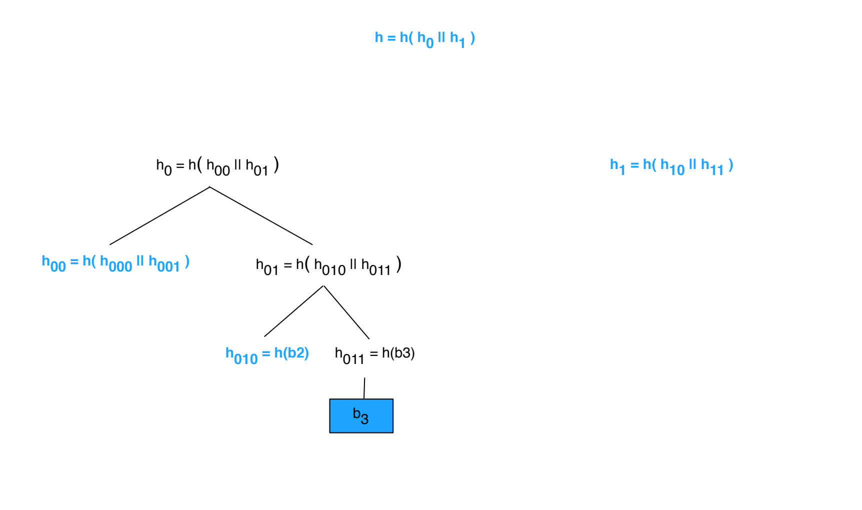
# Hash tree : Merkle Tree

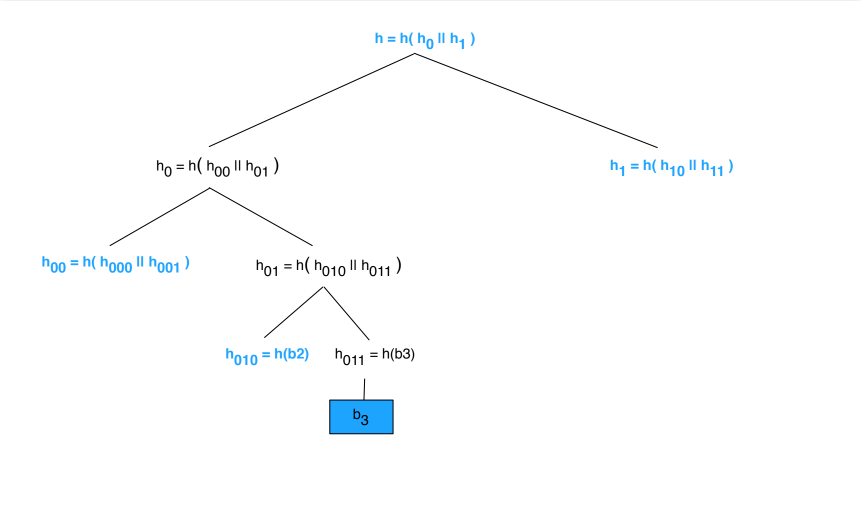


# Hash tree : Merkle Tree



# Hash tree : Merkle Tree




h = h( h0 ll h1 )
h0 = h( h00 ll h01 )
h1 = h( h10 ll h11 )
h00 = h( h000 ll h001 )
h01 = h( h010 ll h011 )
h010 = h(b2)
h011 = h(b3)
b3

# Hash tree : Merkle Tree

I know the root of the Merkle tree, and I would like to know whether data block $b_3$ belongs to the tree ?

Question : How can I do that without looking for the full tree ?

I need $\log n$ pieces of information and $\log n$ hash operations

# Digital signature primitive

A digital signature is just like a signature on a document

- Only the creator of the document can sign, but anyone can verify it
- Signature is tied to a particular document

How can we build such a digital signature ?

# Digital signature

Three functions :

- $(s_k, p_k)$ := generateKeys(keysize)
  - $s_k$ : private signing key
  - $p_k$ : public verification key
- sig := sign($s_k$, message)
- isValid := verify($p_k$, message, sig)

# Digital signature

Requirements :

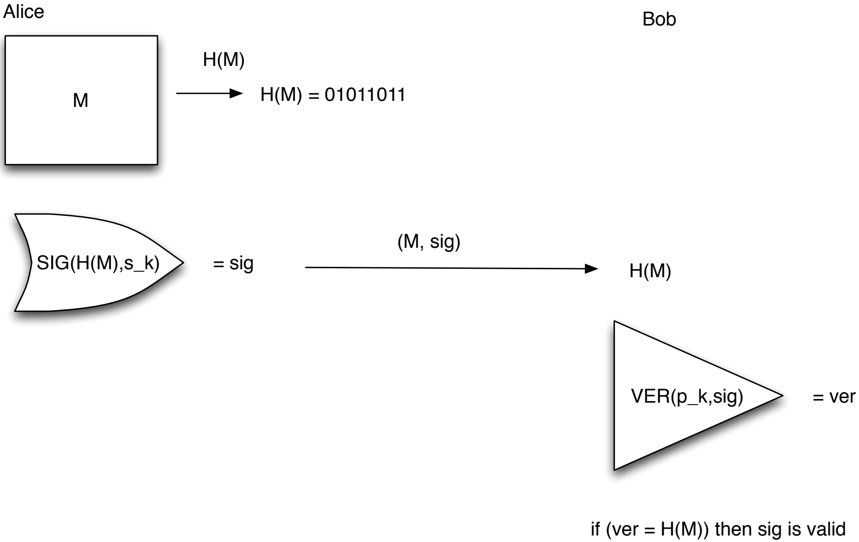
- The verify operation must return true when fed with valid signatures

$$\text{verify}(p_k,\text{message}, \text{sign}(s_k, \text{message}))=\text{true}$$

- The signature scheme is unforgeable

  An adversary that knows $p_k$ and can choose any messages to be signed cannot produce a verifiable signature for another message

# Digital signature



# Digital signature

- The algorithms to generate keys and sign must have access to a good source of randomness
- Signing the hash of a message is as safe as signing the message itself

In Bitcoin, the signature scheme is ECDSA (Elliptic Curve Digital Signature Algorithm) [2]

- private key = 256 bits
- Public key = 512 bits
- Message = 256 bits
- signature = 512 bits

2. Johnson, Don, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ECDSA) . International Journal of Information Security 1.1 (2001) : 36âĂŘ63

# Using verification public key as an identity

Idea : use the verification key of a signature as an identity

- If you see a msg such that the signature verifies under $p_k$ (i.e. *verify*($p_k$, *msg*, *sig*)= *true*) then on can see $p_k$ as a party saying statements by signing them
- To speak on behalf of $p_k$ one must know $s_k$
- So there is an identity in the system such that only a single one can speak for it which is what we want for a signature

✓ By looking at public keys as identities you can generate as many identities as you want !

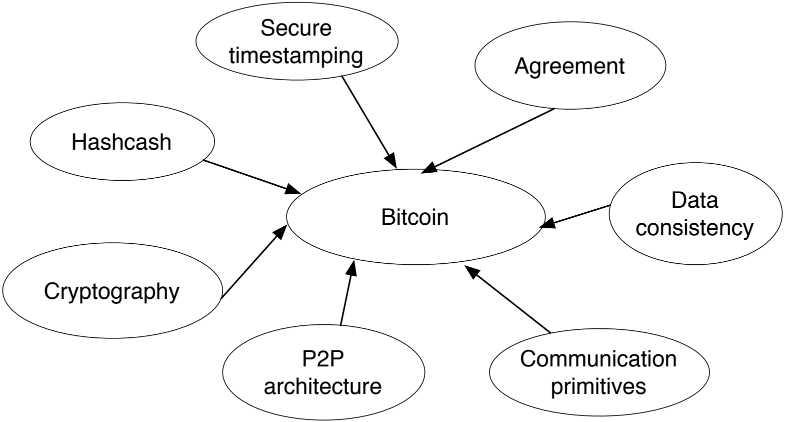
# Using verification public key as an identity

- Create new identities :
  - Eric creates a new pair ($s_k$,$p_k$)
  - $p_k$ is the public name Eric uses
  - Eric is the only person that can speak on behalf of $p_k$ because he knows $s_k$
  - $p_k$ is sufficient ! nobody needs to know that Eric created it
- Creation of identities as often as you want !
  - no central authority in charge of registering new identities !
  - this is the way Bitcoin creates identities (called addresses)
    - address = Hash(public key)

Some words on privacy

- no relationships between $p_k$ based identities and real identities
- by using the same $p_k$ (identity) an adversary can infer some relationships based on the activity of $p_k$

# What is Bitcoin ?



# Bitcoin ingredients : Computational puzzles

Bitcoin are created (minted) and valued independently of any other currencies

- To acquire value a digital currency must be scarce by design
- Minting money requires solving a computational problem
- This is not a new idea : Dwork and Naor in 1992[3] proposed pricing functions

3. C. Dwork and M. Naor, « Pricing via Processing or Combatting Junk Mail », Proceedings of the 12th Annual International Cryptology (Crypto 92), pp 138-147

# Bitcoin ingredients : Computational puzzles

Main principles

- Sending an email requires solving a computation problem
- Absence of proof = no delivery
- Moderate effort if unfrequent email, prohibitive otherwise

Computational puzzle are helpful if

- each puzzle unique (e.g. email depends on both sender, recipient, time)
- the solution of a puzzle should be easy to verify
- solving a puzzle should not decrease the time for solving another one
- difficulty of puzzles should vary according to hardware/environment features

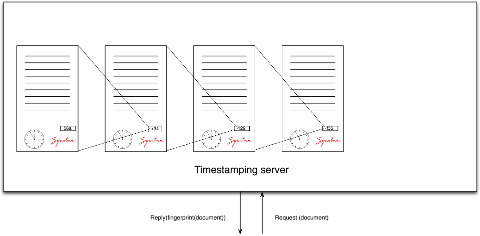
The blockchain : a ledger in which all Bitcoin transactions are securely recorded.

- This is not a new idea : Haber and Stornetta (1991) [4] proposed a method for secure timestamping of digital documents (rather than digital money)

4. S. Haber and W.S Stornetta, « How to Time-Stamp a Digital Document », Journal on Cryptology (1991) 3(2) pages 99–111

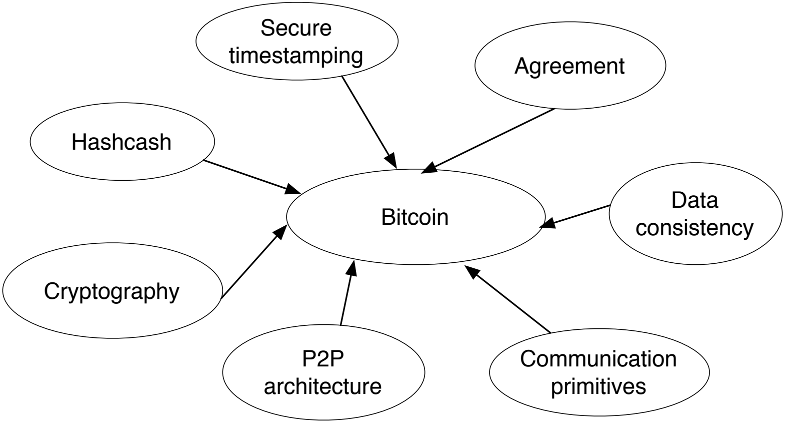
# Bitcoin ingredients : ledger

- Give an idea of when a document has been created
- Provide the order of creation of documents
- Integrity of each (previous) document
- Total ordering relies on the trusted server




- Bitcoin : get ride of central authority while guaranteeing a total ordering of the transactions

# Bitcoin relies on a set of distributed algorithms



# Bitcoin ingredients

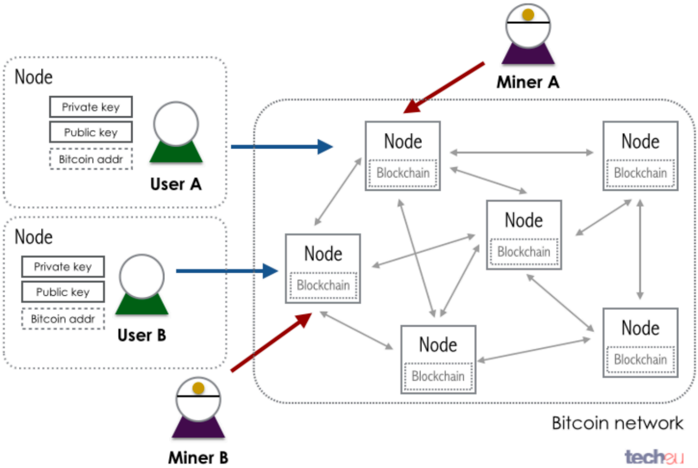
- Participating entities
  - Users, Miners and Bitcoin nodes
- Data structures
  - Addresses
  - Transactions
  - Blockchain

# The Bitcoin Network

- A P2P network of a large number of nodes
- Each node implements different functions
    - routing, keeping the blockchain, verifying the transactions, mining
- The Bitcoin runs over TCP
- Nodes can join and leave the system at any time
- The network is not structured

✓ The main purpose of the P2P network is to maintain and verify the distributed ledger

# The Bitcoin Network



# The wallet

In Bitcoin, each user uses a wallet

- A wallet stores all the keys generated by the user
- Keys : $(s_k, p_k)$
  - $s_k$ must be a random number (flip a coin)
  - $p_k$ is generated from $s_k$
- In a transaction, the recipient of a payment is represented by a bitcoin address which is the fingerprint of a public key
- Each time a user wishes to create a transaction, it generates a new address

# Bitcoin transaction

- A transaction is the data structure that allows a user A to transfer bitcoins to user B (bitcoin address of B)
- A transaction consists in 300 to 400 bytes
- A transaction does not contain any confidential information

# Input and output lists

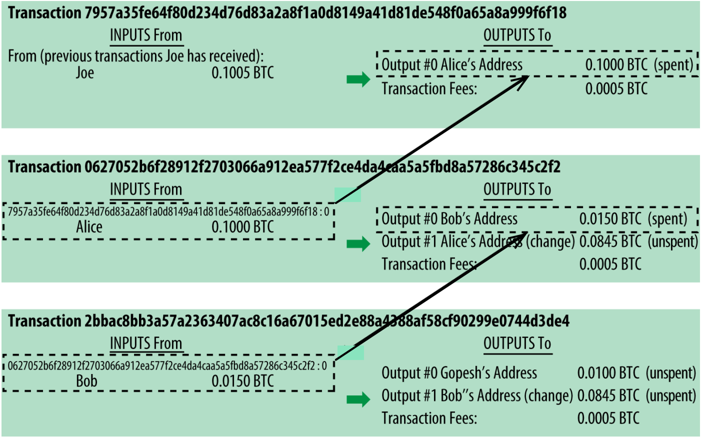
- A transaction contains two types of information
  - The input list
  - The output list

**Transaction as Double-Entry Bookkeeping**

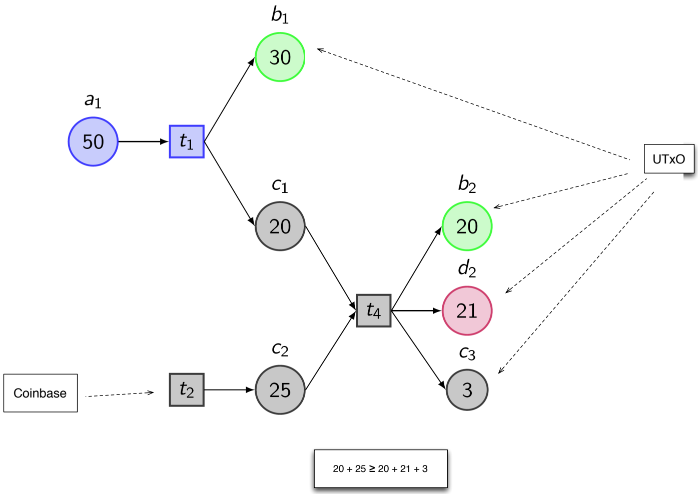
| Inputs | Value | Outputs | Value |
|---|---|---|---|
| Input 1 | 0.10 BTC | Output 1 | 0.10 BTC |
| Input 2 | 0.20 BTC | Output 2 | 0.20 BTC |
| Input 3 | 0.10 BTC | Output 3 | 0.20 BTC |
| Input 4 | 0.15 BTC | | |
| Total Inputs: | 0.55 BTC | Total Outputs: | 0.50 BTC |

| | | |
|---|---|---|
| | *Inputs* | *0.55 BTC* |
| - | *Outputs* | *0.50 BTC* |
| | *Difference* | *0.05 BTC (implied transaction fee)* |

Validity checked by anyone → presence of a trusted third party superflous



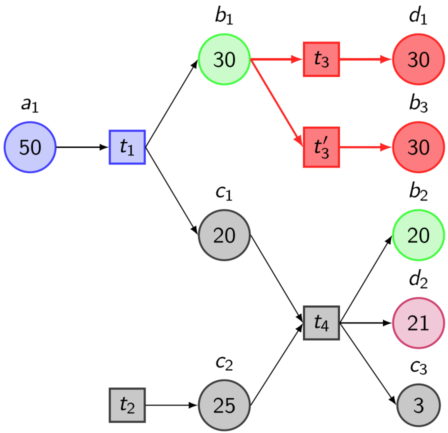
# Bitcoin Transaction



# Unspent Transactions Output (UTXO)

- In Bitcoin there are no accounts (as maintained in a bank)
- There are only UTXOs
- In a transaction
  - an input refers to an UTXO
  - an output creates an UTXO

# Bitcoin Transaction



Transaction 08f794a8a28d8ba58daef1337ce4a88171f931dd858477db3889df adef5b917a from block 438070:

0100000001ba54aa54af3ca6247589210f47e3c617ba4219f84b61c8b8724381
cd2c448349010000006a47304402201e0b0555330b9ba6dc689aeebecf04643
91a882c41a6650ab66f803179860a1802207d1b46c45d37e8a88fee49c3e02ad
b9cd3ccb4bb96ca313135e00a5c01f71a6b012102374f390070a14763707fe93
10a73eaf2b2221734d0ff0a0684078571e2a12e9efeffffff0237b9190000000000
1976a9143d5b9da23ff21a211f101ee2adec37d6b797db7c88ac40420f000000
00001976a9144ce03f31d4bdbc2932f14cea99f4d96edcdbef0c88ac35af0600

Decoded:

- Header: ver=1, vin.size=1, vout.size=2, nLockTime=438069
- Inputs:
  - ID: 4983442ccd814372b8c8614bf81942ba17c6e3470f21897524a63caf54aa54ba
  - Index: 1 (input value: 0.030 980 35 BTC)
  - scriptSig:
    - Signature: 304402201e0b0555330b9ba6dc689aeebecf0464391a882c41a6650ab66f803179860a1802207d1b46c45d37e8a88fee49c3e02adb9cd3ccb4bb96ca313135e00a5c01f71a6b[ALL]
    - Key: 02374f390070a14763707fe9310a73eaf2b2221734d0ff0a0684078571e2a12e9e
  - nSeq: 4294967294

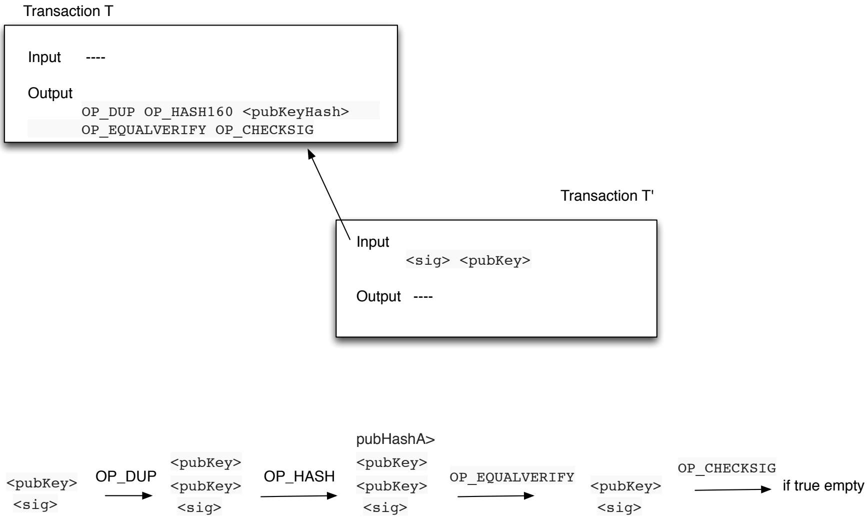
Decoded (ctd):

- Outputs:
  - n: 0
    - value: 0.016 858 15 BTC
    - ScriptPubKey: `OP_DUP OP_HASH160` 3d5b9da23ff21a211f101ee2adec37d6b797db7c `OP_EQUALVERIFY OP_CHECKSIG`
  - n: 1
    - value: 0.010 000 00 BTC
    - ScriptPubKey: `OP_DUP OP_HASH160` 4ce03f31d4bdbc2932f14cea99f4d96edcdbef0c `OP_EQUALVERIFY OP_CHECKSIG`

# Bitcoin Transactions

- Bitcoin relies on a (limited) script language to lock inputs and to unlock outputs
- To lock an output, the script provides all the conditions to spend the output
  - fingerprint of the public key $H(p_k)$
  - conditions for a miner to spend its output
- To unlock an input, the script provides all the conditions to spend the output
  - public key $H(p'_k)$ together with the signature of the $s'_k$

# Bitcoin Transactions



# Validation of transactions

- Each node validates the transactions it receives
  - For each input,
    - the node checks that the script returns true
    - the UTXO has not been already spent
- If the input is not valid, the node does not propagate the transaction
- Node stores the validated transactions in the « Transactions pool »

Any questions ?