BITCOIN

A PEER-TO-PEER ELECTRONIC CASH SYSTEM

BY SATOSHI NAKAMOTO

NERYIA DAYANZADA

MORIA YEFET

CONTENT

CONTENT

WHY BITCOIN?

ELECTRONIC PAYMENTS

- Reversible ⇒ Payment uncertainties

- Trusted third party required ⇒
  - the information is exposed to another party
  - increases the cost

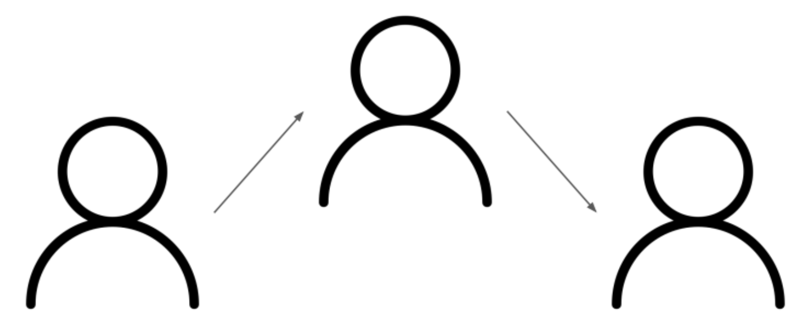
BITCOIN

- An electronic payments system
- Based on **cryptography** instead of trusting a third party
- **Decentralized**
  - i.e., no single entity that controls it ⇒ multiple entities control (a “collective”)
  - fees can be lower (no mediator needed)
- **Solves double-spending** (we will see later)

CONTENT

HOW BITCOIN WORKS?

TRANSACTIONS



- How do we make digital money transfers in common currencies?
  - we use a 3rd party, e.g., a bank
  - the bank deducts money from Alice balance and adds the deducted money to Bob's balance
- But, Bitcoin is decentralized
  - **we need a mechanism that would replace the bank's role**

TRANSACTIONS

- Electronic coin = A chain of digital signatures
- Transference the coin to the next by digitally signing a hash of:
  - the previous transaction
  - the public key of the next owner
- To verify the chain of ownership, the signatures can be verified

WALLETS

- A Bitcoin user has **Wallet**
- The wallet uses keys:
  - the public key is used to identify your wallet
  - the private key is used to sign transactions
  - both stored within the wallet
  - giving you secure access to any crypto coins you own

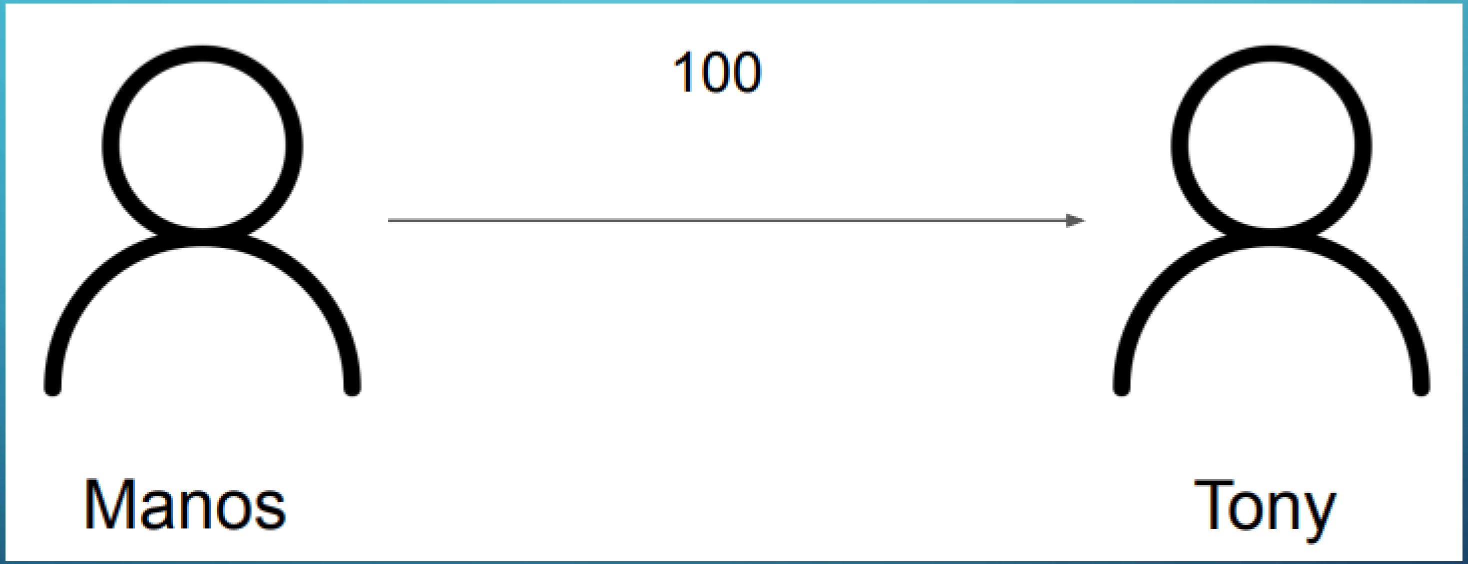
NODES AND MINERS

- Bitcoin is regulated by a network of nodes
- **Node** = computer that runs the Bitcoin software
  - Send and receive transactions with other nodes in the network
  - Verify transactions validity
- **Miner** = node that creates new bitcoins

TRANSACTIONS

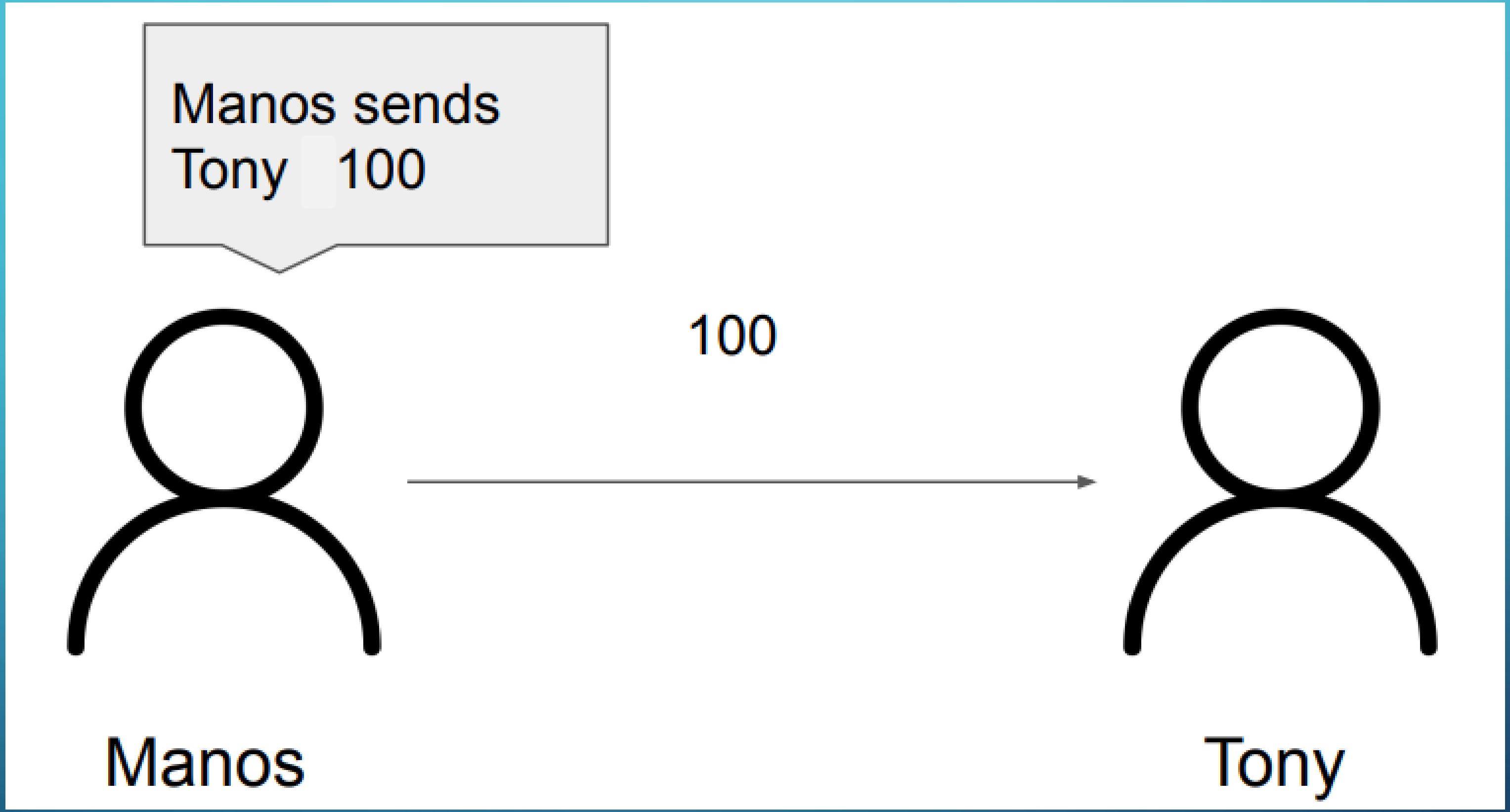
- Transfer ownership of coins between wallets
- Each transaction has input and output (except for block reward which only has an output)
- The entire input must be spent
  - non spent input is considered a transaction fee for the miner

TRANSACTIONS



Who cares? No third Party! Only Manos and Tony knows!

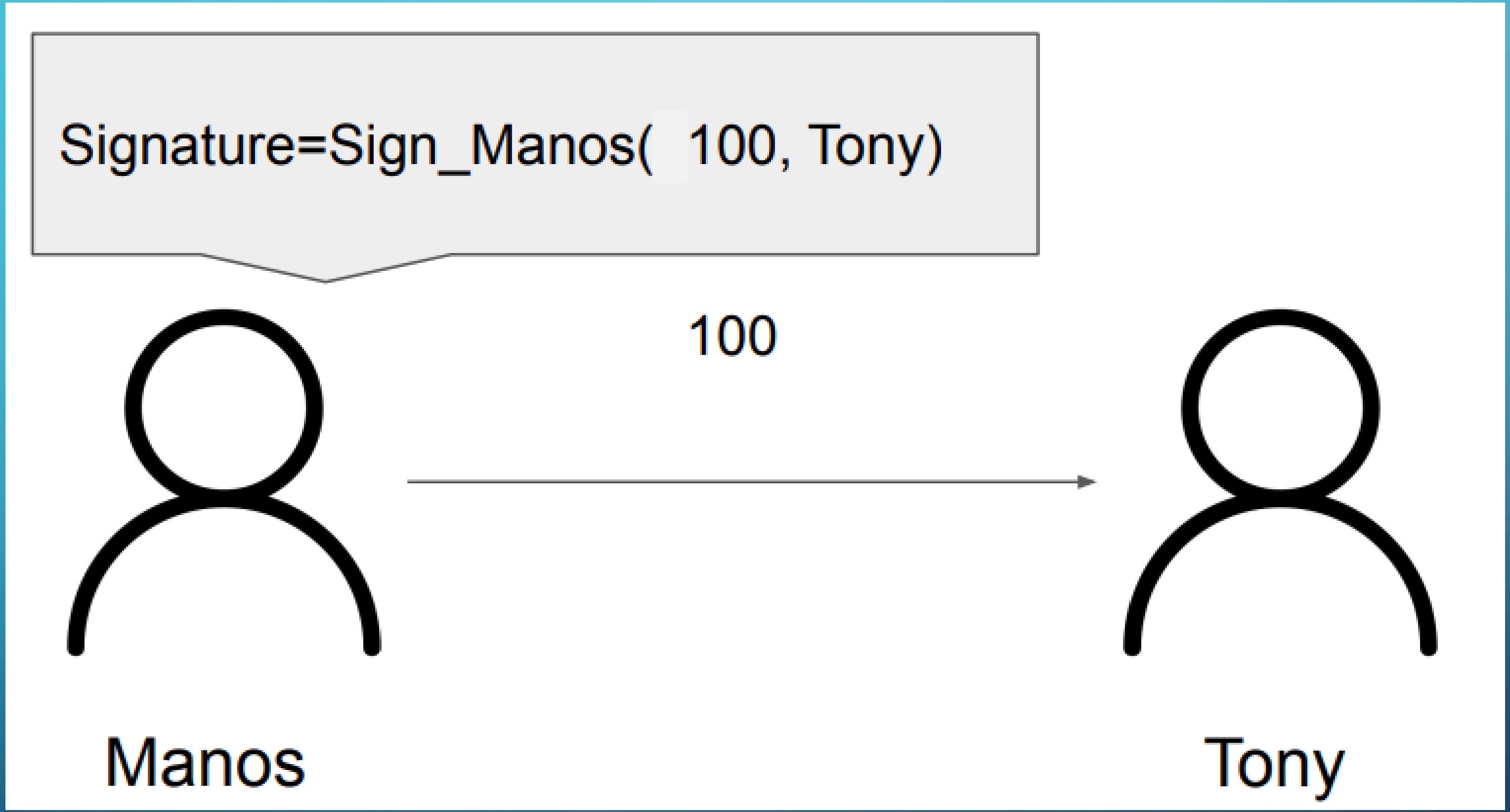
TRANSACTIONS



Manos has to prove that he is the owner of

the "address" where the money is located

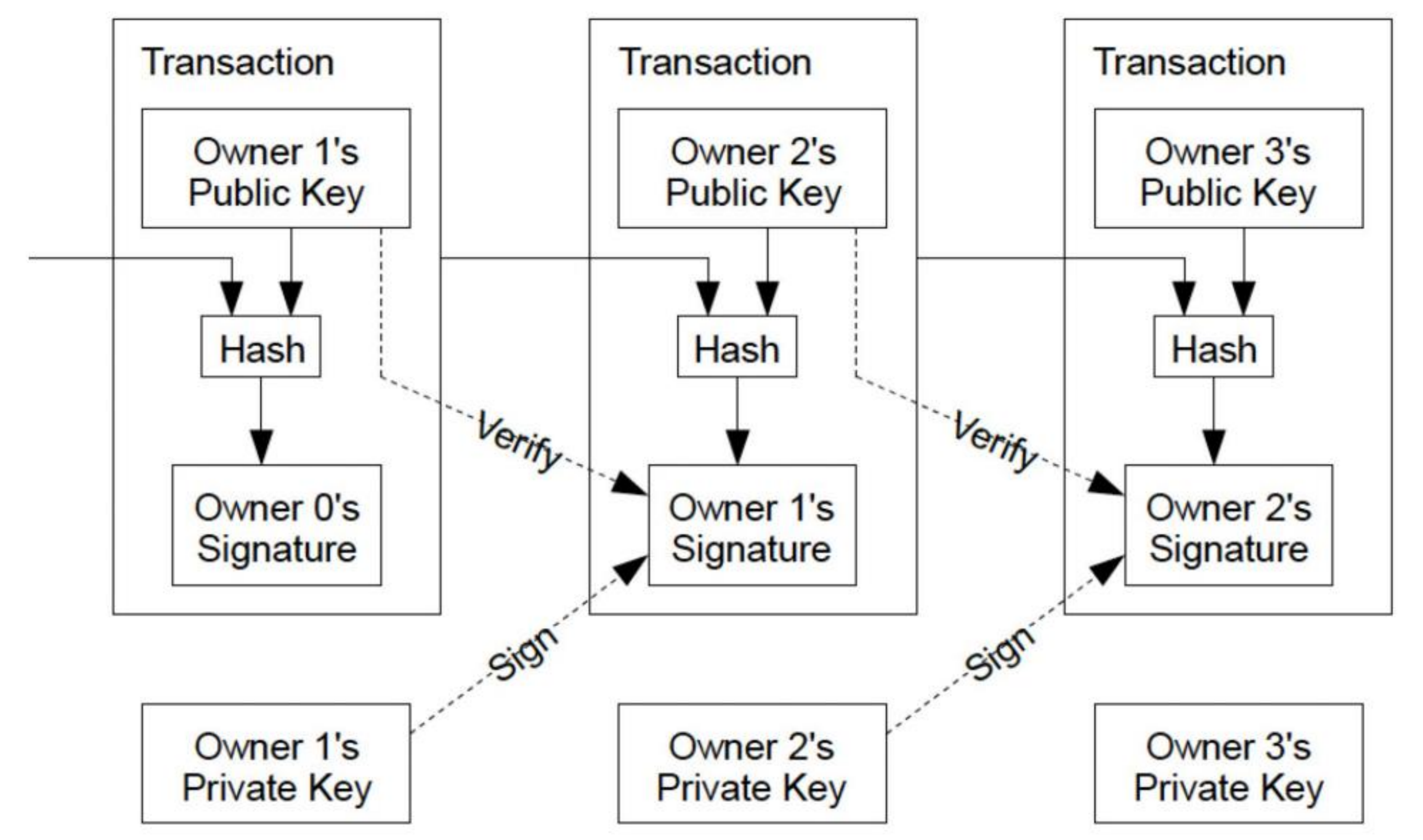
TRANSACTIONS



Manos signs with his **private key** ⇒

which is paired with his **public key** (his "wallet" address)

TRANSACTIONS

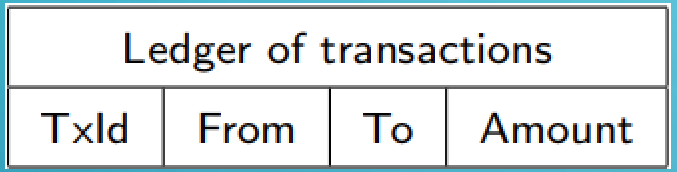


The goal of a digital signature is to prove that you're the owner of a public key

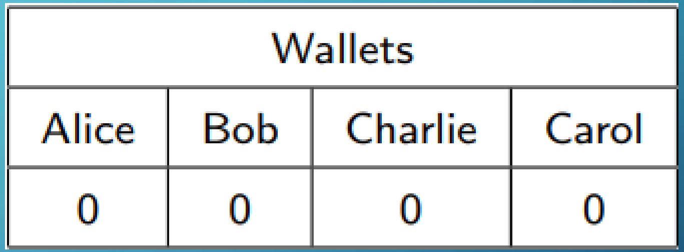
BITCOIN'S LEDGER

- A **ledger** is a collection of **valid** transactions previously made
- The ledger only accepts new transactions that don't conflict with previously recorded transactions
    - a transaction describes that Alice transferred X coins to Bob
    - Alice no longer owns these coins and now Bob owns them
    - Alice can't add another transaction to the ledger where she transfer <u>the same X coins she transferred to Bob</u> to Charlie

BITCOIN'S LEDGER

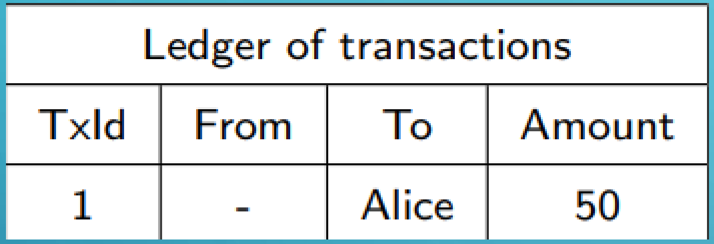


Ledger of transactions			
TxId	From	To	Amount

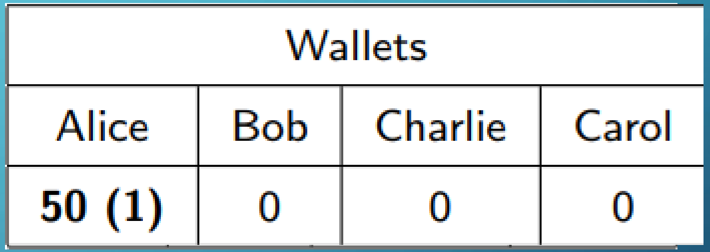


Wallets			
Alice	Bob	Charlie	Carol
0	0	0	0

BITCOIN'S LEDGER

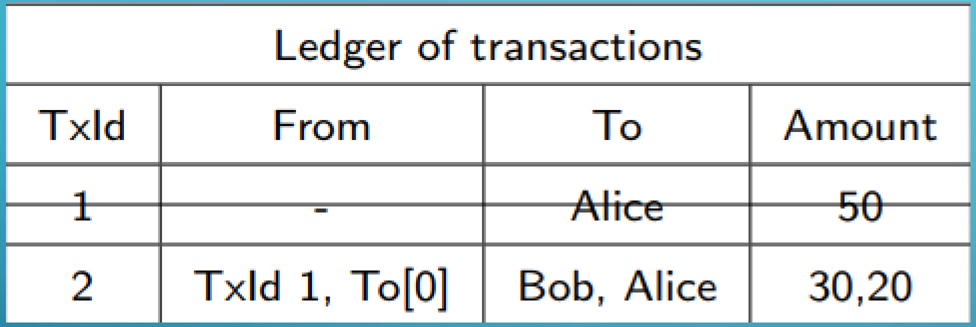


Ledger of transactions			
TxId	From	To	Amount
1	-	Alice	50

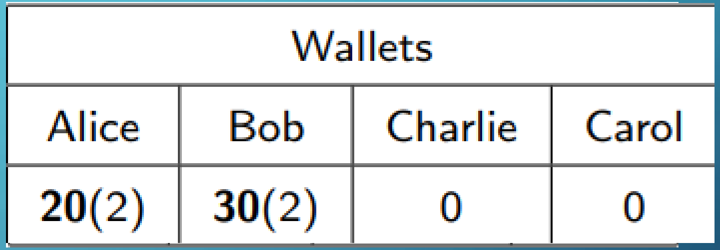


Wallets			
Alice	Bob	Charlie	Carol
50 (1)	0	0	0

BITCOIN'S LEDGER

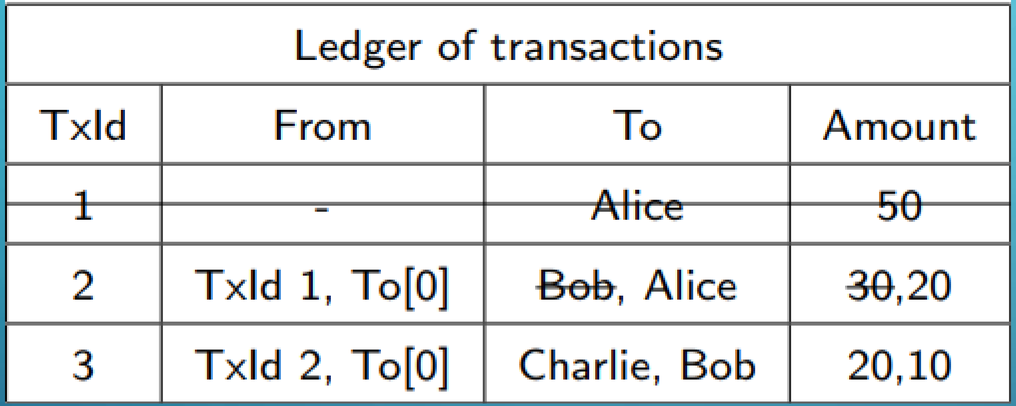


Ledger of transactions			
TxId	From	To	Amount
~~1~~	~~-~~	~~Alice~~	~~50~~
2	TxId 1, To[0]	Bob, Alice	30,20

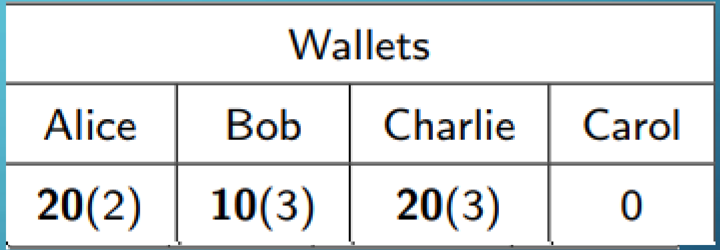


Wallets			
Alice	Bob	Charlie	Carol
20(2)	**30**(2)	0	0

BITCOIN’S LEDGER

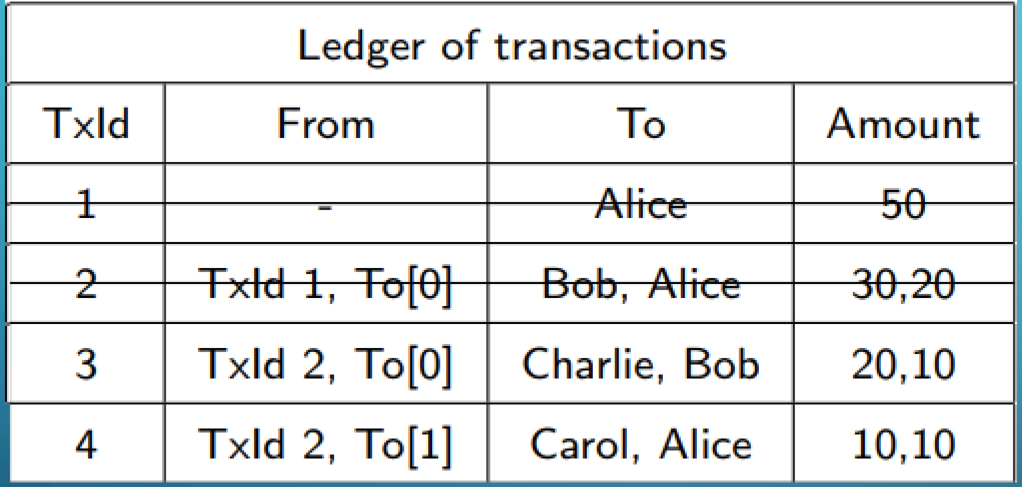


Ledger of transactions			
TxId	From	To	Amount
~~1~~	~~-~~	~~Alice~~	~~50~~
2	TxId 1, To[0]	~~Bob~~, Alice	~~30~~,20
3	TxId 2, To[0]	Charlie, Bob	20,10

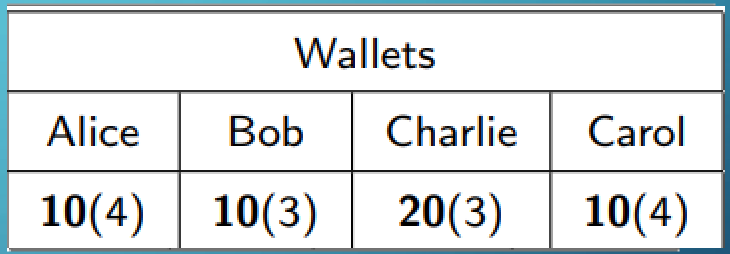


Wallets			
Alice	Bob	Charlie	Carol
20(2)	**10**(3)	**20**(3)	0

BITCOIN'S LEDGER



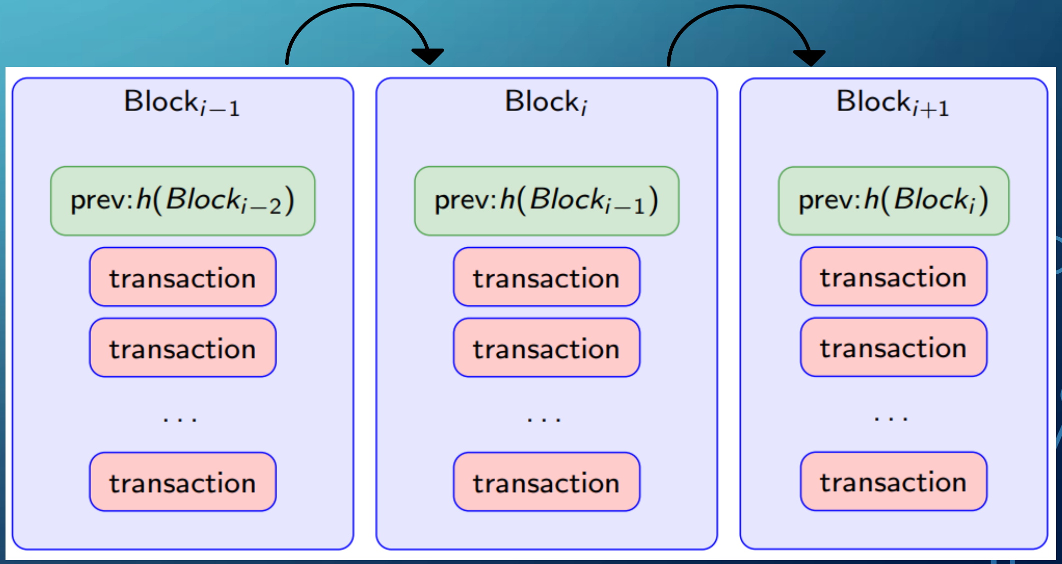
Ledger of transactions			
TxId	From	To	Amount
~~1~~	~~-~~	~~Alice~~	~~50~~
~~2~~	~~TxId 1, To[0]~~	~~Bob, Alice~~	~~30,20~~
3	TxId 2, To[0]	Charlie, Bob	20,10
4	TxId 2, To[1]	Carol, Alice	10,10



Wallets			
Alice	Bob	Charlie	Carol
10(4)	**10**(3)	**20**(3)	**10**(4)

BLOCKCHAIN

- The transactions are arranged in blocks
- The order of the blocks **matters**
  - to ensure that only valid transactions are added
- Each block points to the previous block
  - hence, **blockchain**



Block_{i-1}

prev: $h(Block_{i-2})$

transaction

transaction

$\ldots$

transaction

Block_{i}

prev: $h(Block_{i-1})$

transaction

transaction

$\ldots$

transaction

Block_{i+1}

prev: $h(Block_{i})$

transaction

transaction

$\ldots$

transaction

CONSENSUS

- How is the ledger maintained?
    - Bitcoin is decentralized
    - there is no single entity that maintains the ledger
- Solution: **everyone** maintain a copy of the blockchain!
- But how?
    - we need to have a consensus!

BLOCKCHAIN-BASED CONSENSUS

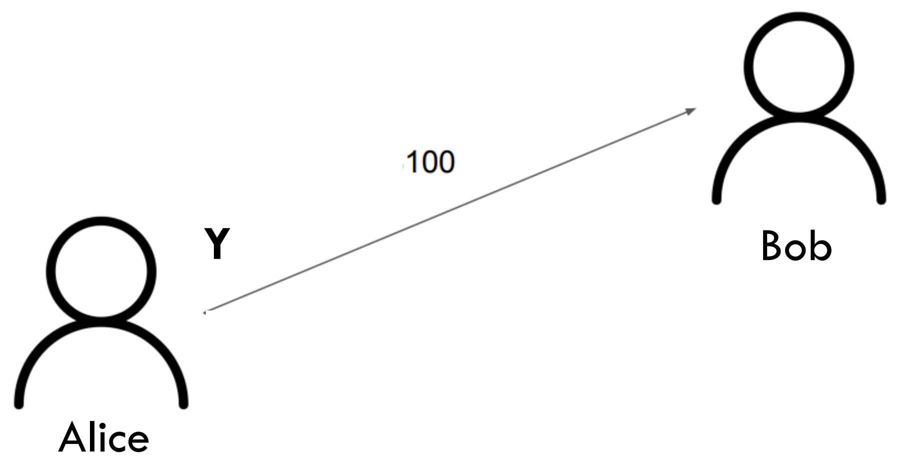
1. New transactions are flooded (to all nodes)
2. Each node (miner) chooses a group of transactions and attempts to generate a valid block
3. When a miner succeeds, it floods the new block
4. Nodes verify and add the new block into their copy of the blockchain
5. Go to step 2

BLOCKCHAIN-BASED CONSENSUS

1. New transactions are flooded (to all nodes)
2. Each node (miner) chooses a group of transactions and attempts to generate a valid block **attempt?**
3. When a miner succeeds, it floods the new block **success?**
4. Nodes verify and add the new block into their copy of the blockchain
5. Go to step 2 **incentive?**

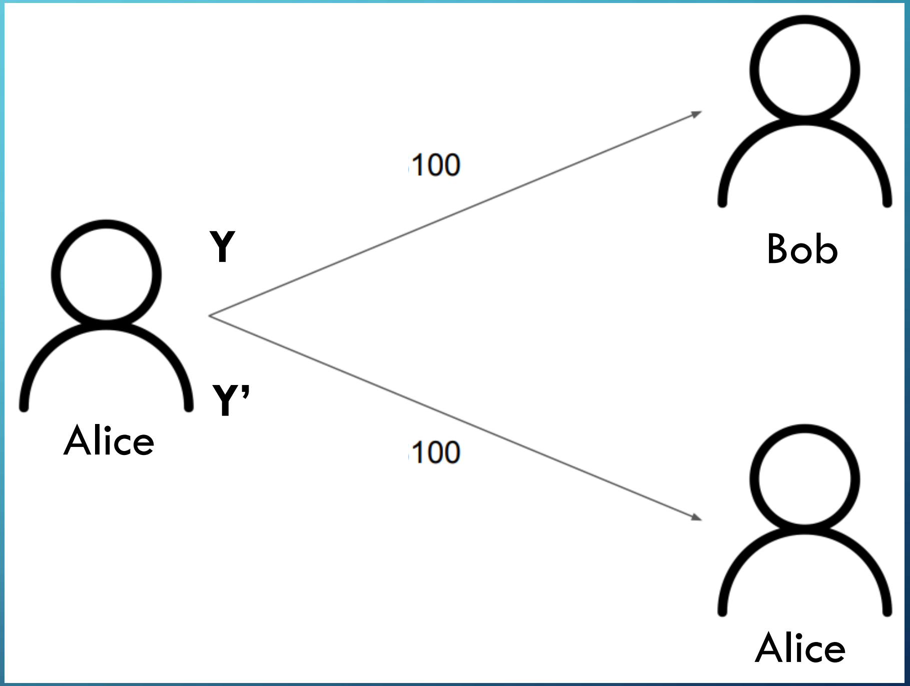
DOUBLE SPENDING

- Alice orders a product, pays with BTC
  - creates transaction **Y** which transfer coins she owns via transaction **X,** to Bob
  - at some point, a new block is added to the blockchain with transaction **Y**



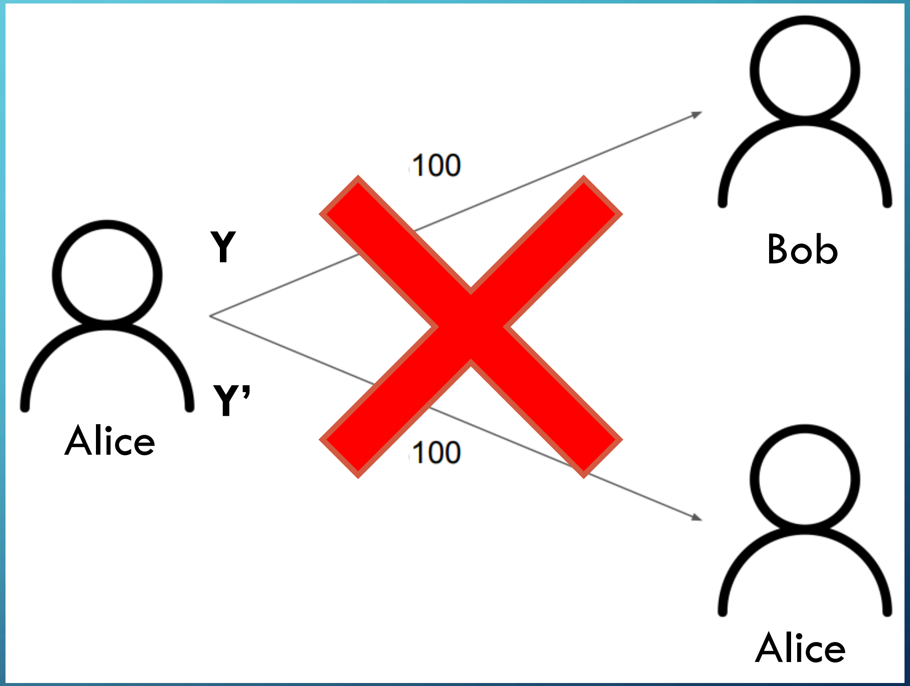
DOUBLE SPENDING

- In parallel, Alice generates a conflicting transaction **Y'**, which takes transaction **X** and transfer the coins to a **different** address, one controlled by Alice



DOUBLE SPENDING

- Two conflicting transactions **can not** co-exist in the blockchain
- Because of the decentralization, different miners can include the transaction from **X** to **Y,** while others from **X** to **Y’**



REQUIREMENTS OF TRANSACTIONS

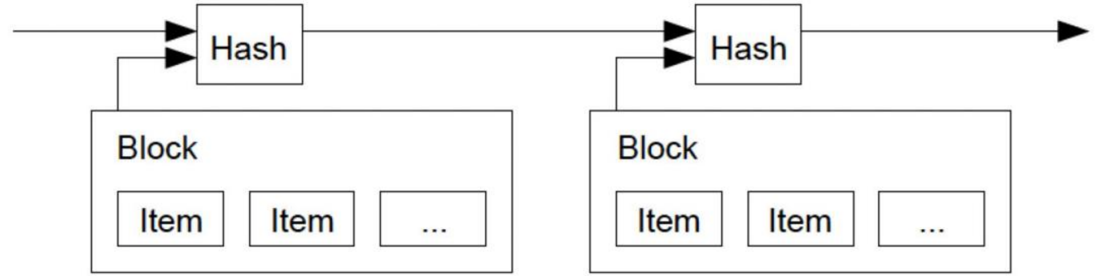
- Transactions must be **publicly** announced
- Need a system to agree on a **single history** of the order
- Need to prove that at the time of each transaction, the **majority nodes agreed** it was the first received

CONTENT

BITCOIN SOLVES THE DOUBLE SPENDING

TIMESTAMP SERVER

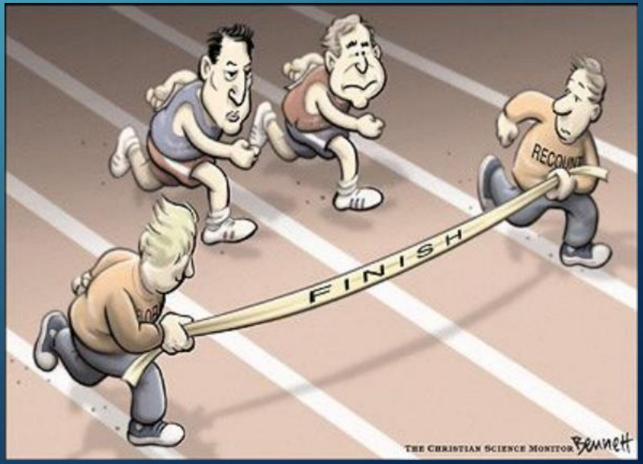
- Taking a hash of a block to be timestamped and widely publishing the hash
- The timestamp **proves that the data must have existed** at the time (obviously, in order to get into the hash)
- Each timestamp **includes the previous timestamp** in its hash, forming a chain
  - each additional timestamp reinforcing the ones before it



CONTENT

PROOF OF WORK

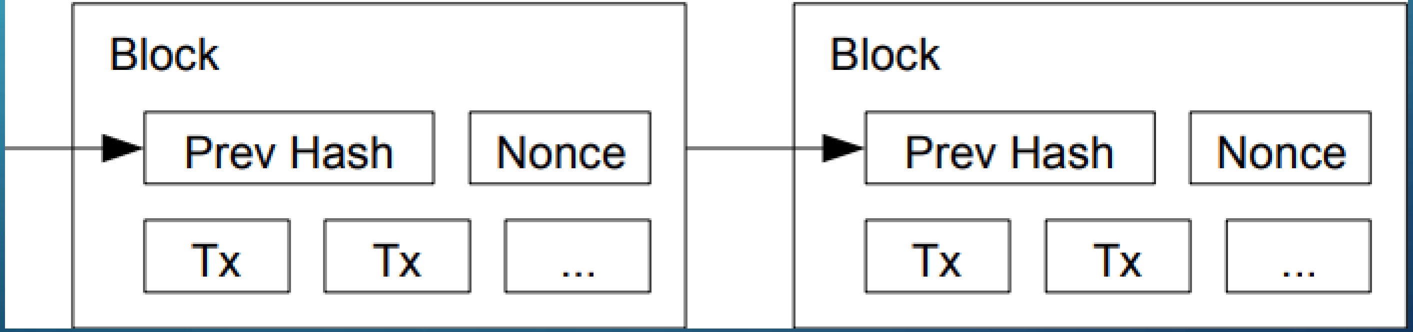
- ➢ Creating a new valid block requires computational effort
- ➢ The more computational power you have - the bigger your chances to win
- ➢ All miners need to find x that hash(x#txs) < ϵ
  - ➢ whoever finds it – wins and gets to create a block
- ➢ Searching for **X**: mining
- ➢ A node that searches for **X**: miner



PROOF OF WORK

- Makes the blockchain difficult to be changed
  - To do it, need to:

1. recalculate the hash value of **the target block**
2. recalculate the hash value of **all blocks after**



PROOF OF WORK

- Solves the problem of determining representation in majority decision making
  - The majority decision is represented by the longest chain ⇒ has the greatest proof-of-work effort invested in it
  - Problem: If the majority is based on number of IP addresses ⇒ Attacker with many IP addresses can break the system
  - Solution: CPU based majority makes the honest chain to grow fastest ⇒ a lot of efforts required to attack

PROOF OF WORK

- The blockchain with **largest proof-of-work** is determined
- To validate an alternative chain, **need to achieve a majority of CPU**
- The probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added
- Hardware speed develops fast, maybe attackers can catch up in future?
  - The proof-of-work difficulty depends on the average number of blocks per hour
  - If they are generated too fast, the difficulty increases

CONTENT

NETWORK

1. New transactions are broadcast to all nodes
2. Each node collects new transactions into a block
3. Each node works on finding a difficult proof-of-work for its block
4. When a node finds a proof-of-work, it broadcasts the block to all nodes
5. Nodes accept the block only if all transactions in it are valid and not already spent
6. Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash

NETWORK

- Nodes always consider **the longest chain to be the correct** one and will keep working on extending it
- In case of a fork, temporarily work on the first one they received
- The tie will be broken when the next proof-of-work is found and one branch becomes longer
- The nodes that were working on the other branch will then switch to the longer one

CONTENT

- ✓Introduction
- ✓Transactions
- ✓Timestamp Server
- ✓Proof-of-Work
- ✓Network
- ➢**Incentive**
- ➢Reclaiming Disk Space
- ➢Simplified Payment Verification
- ➢Combining and Splitting Value
- ➢Privacy
- ➢Calculations
- ➢Conclusion

INCENTIVE TO PARTICIPATE IN BITCOIN

- Bitcoin should have **many** nodes that maintain the blockchain
  - otherwise, Bitcoin won’t exist
- How to make nodes invest computational power, i.e., money, in maintaining the blockchain?
  - reward them by giving them coins
- Bitcoin is a currency, thus its coins have value
  - this is the incentive

INCENTIVE TO PARTICIPATE IN BITCOIN

- Nodes are incentivized to maintain the blockchain
  - what does it mean?
    1. group new transaction into blocks
    2. add new blocks to the chain
    3. enforce validity rules on new blocks
- Rewarded for every new block they add to the chain
  - the first transaction in a block ⇒ A new coin to the miner

REWARD FOR CREATING A NEW BLOCK

- Block Creation Fee
  - the node that create a new block adds a transaction that prints new money to it
- Transaction fee
  - any input value that isn’t spent in the transaction output is collected by the block creator
  - it incentivizes miners to pick the transaction over other transactions

INCENTIVE

- ➢ The incentive may help encourage nodes to stay honest
- ➢ If a greedy attacker is able to get more CPU power than all the honest nodes, he would have to choose between:
  1. using it to defraud people by stealing back his payments
  2. using it to generate new coins

  It’s more profitable to play by the rules, than to undermine the system and the validity of his own wealth

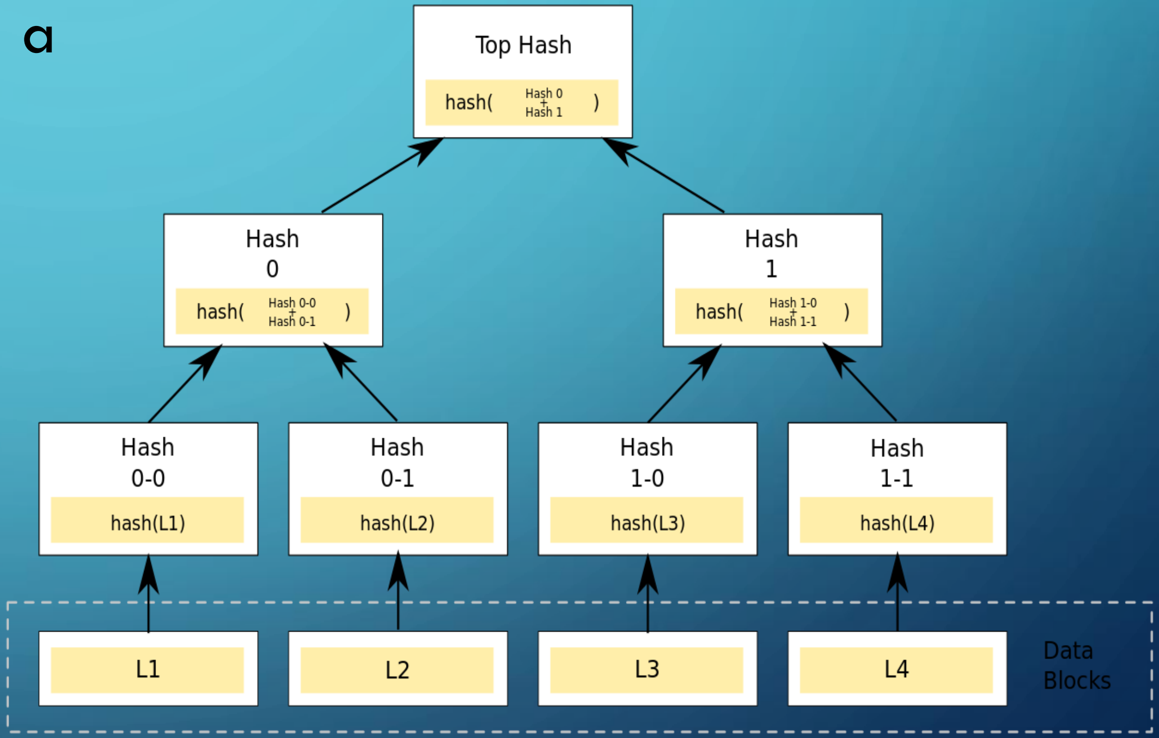
CONTENT

RECLAIMING DISK SPACE

- Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space
- Every modification of a block will result in a change in the content of the block
  - accordingly, a change in the number chosen to maintain Proof-of-Work
- The solution is to use Merkel trees!!

MERKLE TREES

- Every "leaf" is labelled with the hash of a transaction’s data
- Every other nodes is labelled with the hash of the labels of its childs nodes
- In our case in order to verify a transaction, we only need to check whether its ID exists in the given block
    - since we have a prof-of-work



Top Hash
hash(Hash 0 + Hash 1)

Hash 0
hash(Hash 0-0 + Hash 0-1)

Hash 1
hash(Hash 1-0 + Hash 1-1)

Hash 0-0
hash(L1)

Hash 0-1
hash(L2)

Hash 1-0
hash(L3)

Hash 1-1
hash(L4)

L1
L2
L3
L4

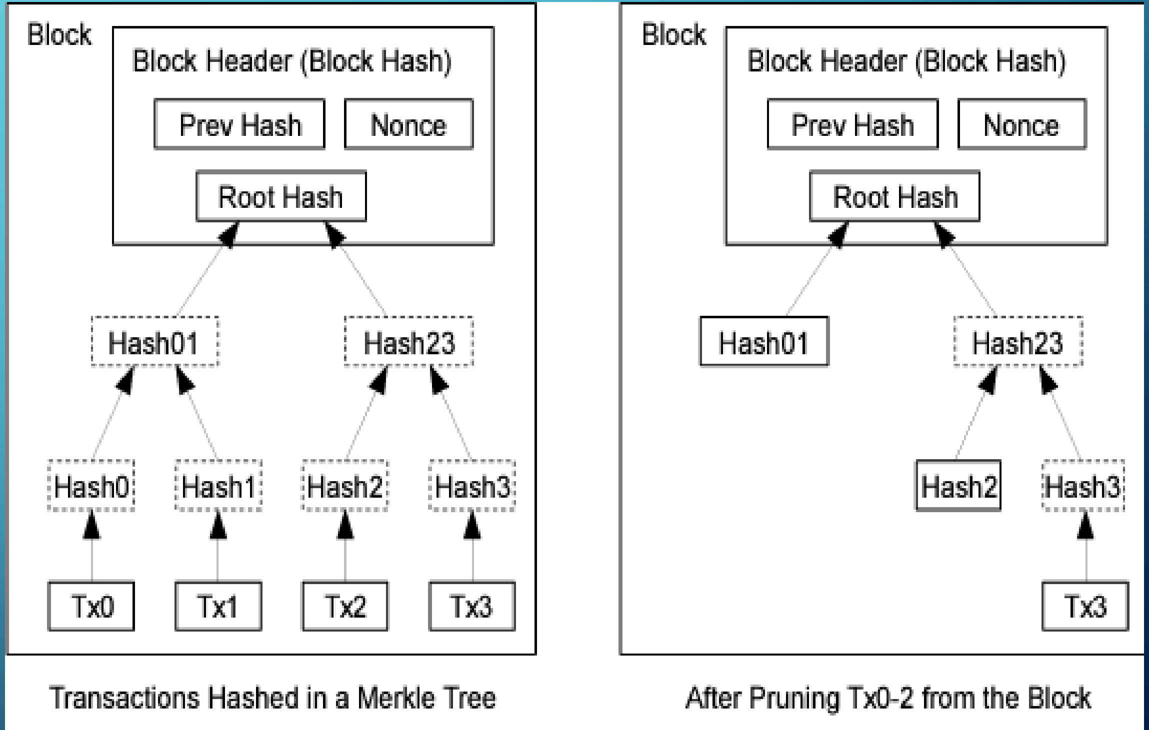
Data Blocks

MERKLE TREES

- Therefore, we can use Merkle trees and when we want to check the integrity of a transaction, we will go to the given block and search in Merkle tree
- If it turns out that the transaction exists, then it is necessarily correct
- A block header with no transaction's data and Merkle tree would be about 80 bytes
- If we suppose blocks are generated every 10 minutes,

80 bytes * 6 * 24 * 365 = 4.2MB

per year



Transactions Hashed in a Merkle Tree

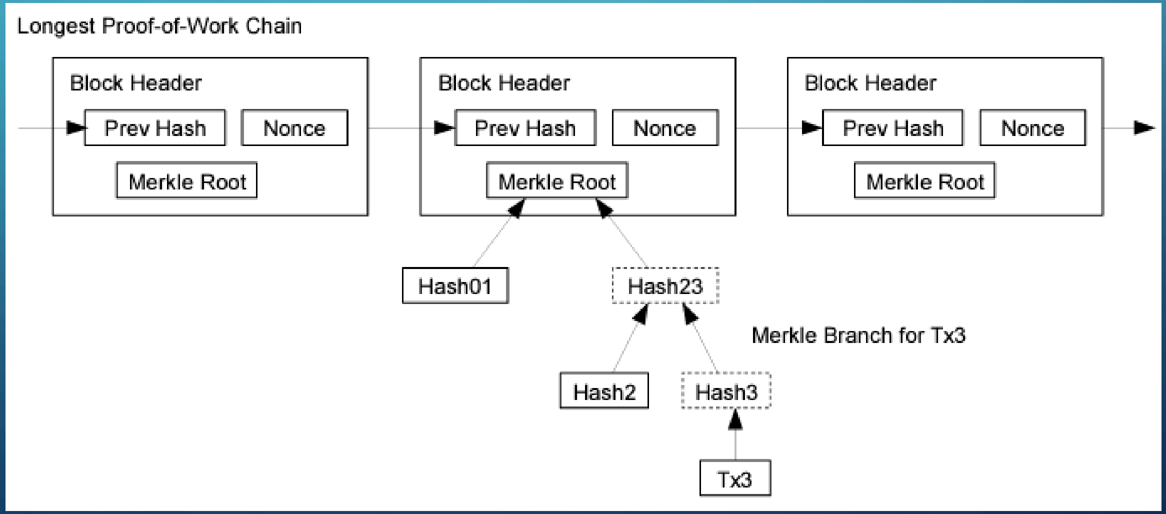
After Pruning Tx0-2 from the Block

CONTENT

SIMPLIFIED PAYMENT VERIFICATION

How to verify transactions?

- ➢ It is not necessary to own and run a node in the network to verify transactions
- ➢ We only need the Merkel trees of each block in the longest chain and connect them
- ➢ To get them you can simply run a query on each block
- ➢ Therefore, each miner will have a chain of headers of all the blocks
  - ➢ when we need to check the reliability of a transaction, we will check whether it appears in the Merkle tree



51% ATTACK

- Mining pool is a group of miners mine together
- If a mining pool grows too much, it could have more than 51% of the CPU power.
- Multiple miners might be (secretly) colluding to avoid becoming a single (too big) entity

BACK TO BITCOIN….

- The verification is reliable as long as honest nodes control the network
  - but it's vulnerable if the network is overpowered by an attacker
- Bitcoin Vulnerable to "51% attack“
  - there isn't really solution to this problem
- One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block
  - each node download the full block and alerted transactions to confirm the inconsistency

GHash.IO 2014

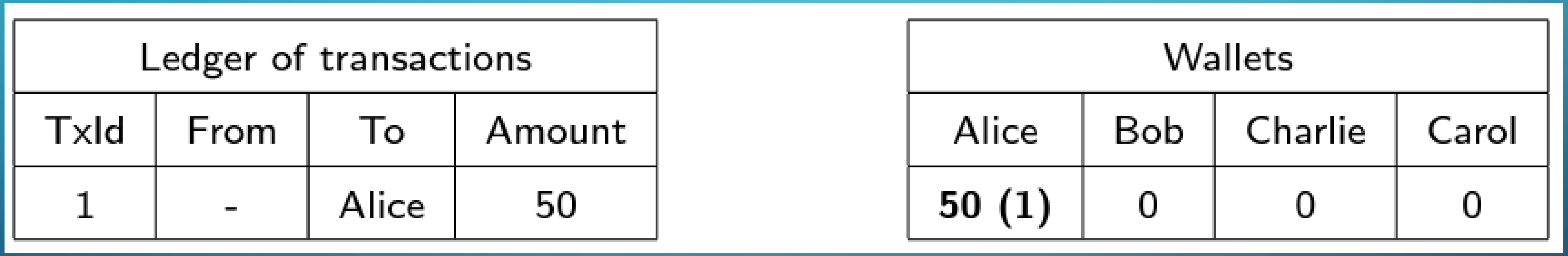
CONTENT

- ✓ Introduction
- ✓ Transactions
- ✓ Timestamp Server
- ✓ Proof-of-Work
- ✓ Network
- ✓ Incentive
- ✓ Reclaiming Disk Space
- ✓ Simplified Payment Verification
- ➢ **Combining and Splitting Value**
- ➢ Privacy
- ➢ Calculations
- ➢ Conclusion

COMBINING AND SPLITTING VALUE

- Until now, we supported only in one input and one output transaction.

- Consider the next senerio:



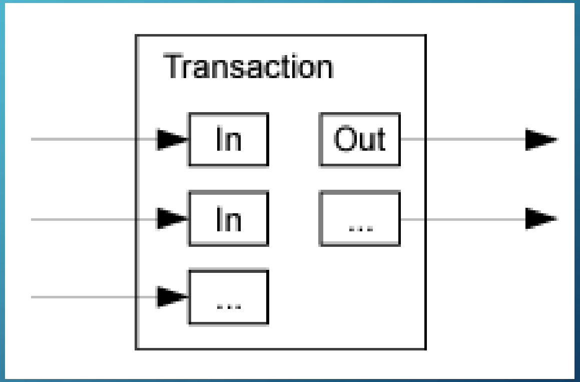
Ledger of transactions			
TxId	From	To	Amount
1	-	Alice	50

Wallets			
Alice	Bob	Charlie	Carol
50 (1)	0	0	0

- **How do Alice transfer 30 bitcoin to Bob without pay transaction fee?**

THE SOLUTION…

- Bitcoin will be either support a single input from a larger previous transaction or multiple inputs combining from smaller amounts
- Bitcoin will be either support at most two outputs: one for the payment, and one returning the change, if any, back to the sender.
- Now we get into problem, when a transaction depends on several transactions, and those transactions depend on many more….
- This is not problem for us, since we got proof-of-work, we need to check only the previous and not their previous…



CONTENT

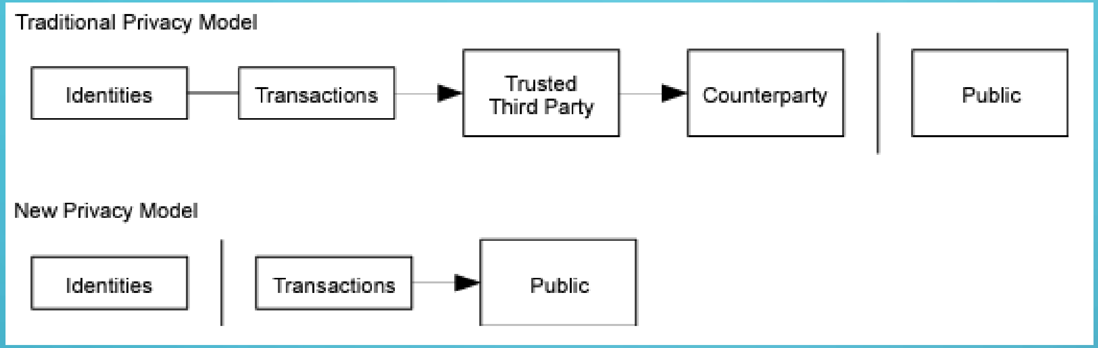
- ✓ Introduction
- ✓ Transactions
- ✓ Timestamp Server
- ✓ Proof-of-Work
- ✓ Network
- ✓ Incentive
- ✓ Reclaiming Disk Space
- ✓ Simplified Payment Verification
- ✓ Combining and Splitting Value
- ➢ **Privacy**
- ➢ Calculations
- ➢ Conclusion

PRIVACY

- Traditional banks ensure the privacy of transactions by limiting information on transactions to the parties involved, including the third-party intermediary.

- In bitcoin, transactions are publicly announced and as such, since everyone sees them, there is no privacy.

THE SOLUTION (FIRST TRY…)



- keeping public keys (from and to) anonymous.
- The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone.

TxId	From	To	Amount
1	Alice	Bob	50

In the new model the transaction look like:

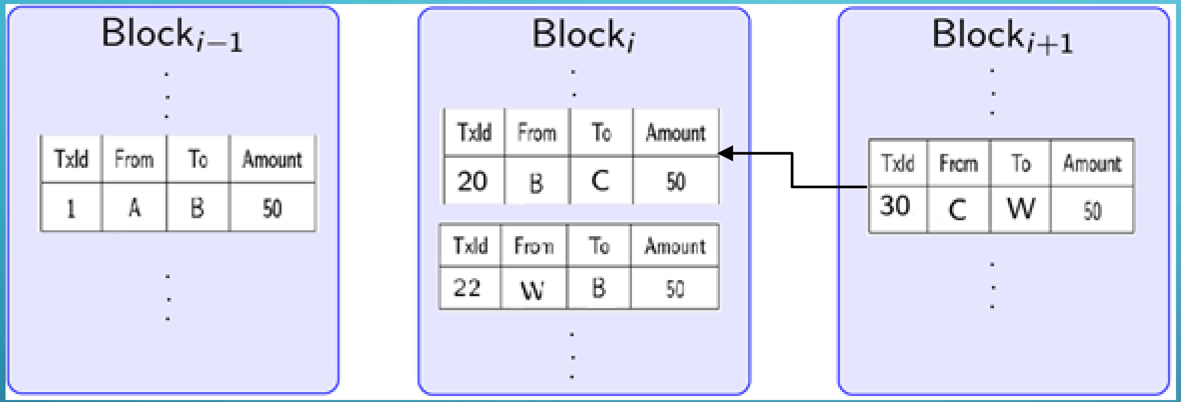
TxId	From	To	Amount
1	A	B	50

THE PROBLEM

Node 1

Node 2

Node 3



TxId	From	To	Amount
1	A	B	50

TxId	From	To	Amount
20	B	C	50

TxId	From	To	Amount
22	W	B	50

TxId	From	To	Amount
30	C	W	50

A = Alice

B = Bob

C = Carol

D = Carol

W = Neryia

L = Moria

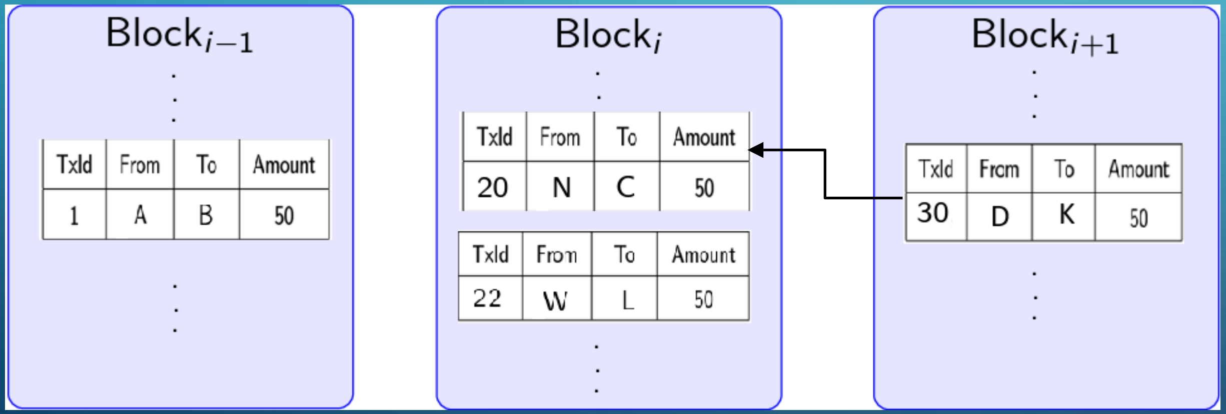
THE SOLUTION

- A new key pair should be used for each transaction to keep them from being linked to a common owner
- Back to the previous example:

Node 1

Node 2

Node 3



Block_{i-1}

TxId	From	To	Amount
1	A	B	50

Block_i

TxId	From	To	Amount
20	N	C	50

TxId	From	To	Amount
22	W	L	50

Block_{i+1}

TxId	From	To	Amount
30	D	K	50

A = Alice

B = Bob

C = Carol

D = Carol

N = Bob

W = Neryia

L = Moria

K = Neryia

WHEN CAN I BE SURE ABOUT THE MONEY?

- Let's explore the duration that a recipient of a new transaction must wait until he can be confident enough that the sender is unable to modify the transaction.
- We're assuming the sender is a malicious actor who aims to trick the recipient into thinking they've been paid and then revert the transaction later on to redirect the funds back to themselves.
- The sender hopes the recipient won't be able to react in time to prevent this.
- Let's see an example:

THE PROBLEM

- We will describe the following scenario:
  - Alice wants to buy a pizza at 8 pm from "Pizza Shemesh" pizzeria.
  - Alice generates two different keys, one for her and one for the pizzeria (let's say A=Alice, S=pizzeria)
  - Alice uploads the transaction already in the morning.
- **What could be the problem in this scenario?**

THE PROBLEM

- We will describe the following scenario:
  - Alice wants to buy a pizza at 8 pm from "Pizza Shemesh" pizzeria.
  - Alice generates two different keys, one for her and one for the pizzeria (let's say A=Alice, S=pizzeria)
  - Alice uploads the transaction already in the morning.
- **What could be the problem in this scenario?**
  - **If Alice will send the transaction before and will track the block and the next blocks, Eventually She will be able to replace the transaction with another transaction that bring back the money to her.**

THE SOLUTION

- Only when Alice arrives at the pizzeria, the seller will give her the pizzeria's public key and only then she will be able to upload a transaction to the network.

- Since the public key will only be transferred when the pizza is received, Alice will not be able to prepare.

- That is, she won't be able to take advance and prepare a duplicate block in which she replaced the payment transaction with a self-transaction

CONTENT

CALCULATIONS

- We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain.

- This race can be characterized as a Binomial Random Walk.

- The success event is the honest chain being extended by one block, increasing its lead by +1, and the failure event is the attacker's chain being extended by one block, reducing the gap by -1.

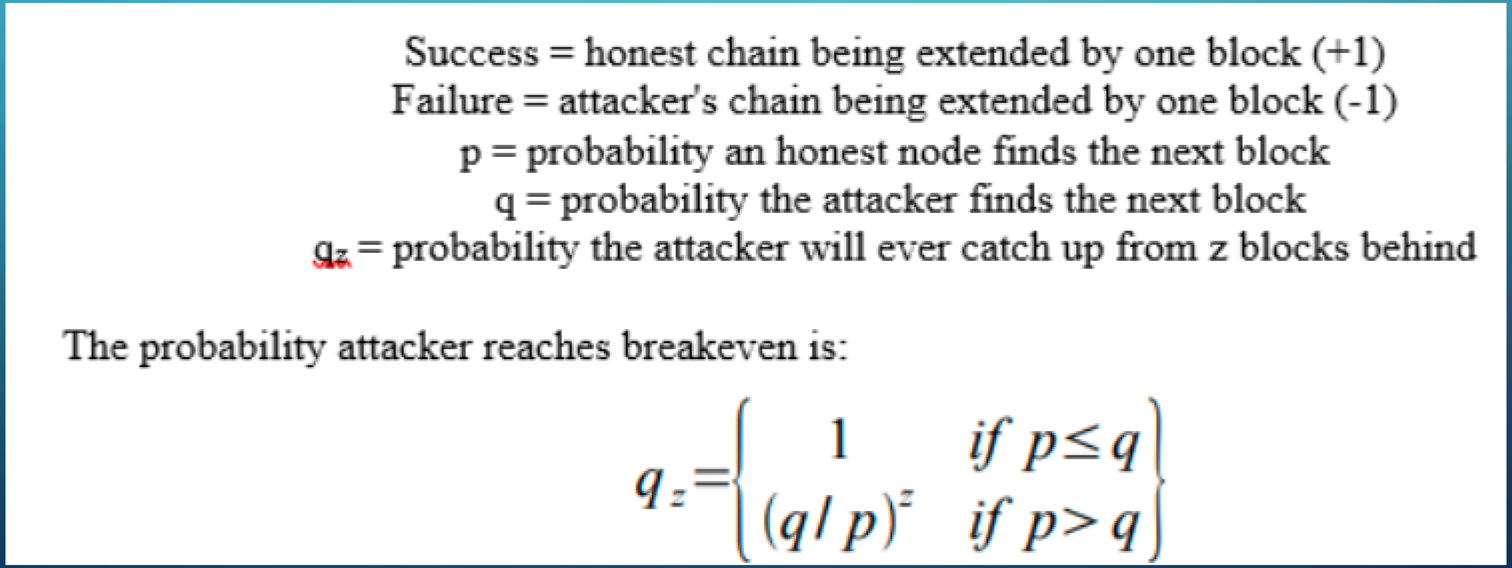
CALCULATIONS

- A random walk is a random process that describes a path that consists of a succession of random steps on some mathematical space.

- The probability of an attacker catching up from a given deficit is analogous to a Gambler's Ruin problem.

THE GAMBLER'S RUIN PROBLEM

- Suppose a gambler with unlimited credit starts at a deficit and plays potentially an infinite number of trials to try to reach breakeven.
- We can calculate the probability he ever reaches breakeven, or (in our case) that an attacker ever catches up with the honest chain, as follows:



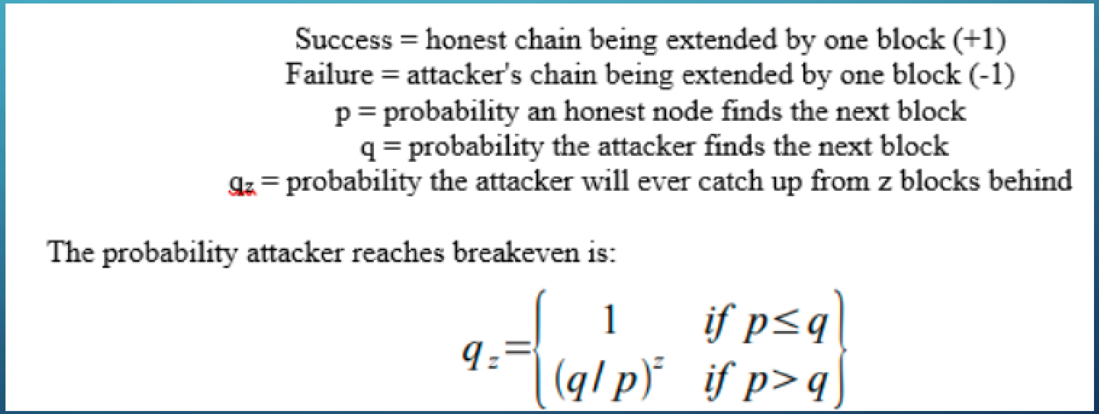
Success = honest chain being extended by one block (+1)
Failure = attacker's chain being extended by one block (-1)
p = probability an honest node finds the next block
q = probability the attacker finds the next block
q_z = probability the attacker will ever catch up from z blocks behind

The probability attacker reaches breakeven is:

$$q_z = \left\{ \begin{matrix} 1 & \text{if } p \le q \\ (q/p)^z & \text{if } p > q \end{matrix} \right\}$$

NOW, BACK TO OUR QUESTION: WHEN CAN I BE SURE ABOUT THE MONEY?

- The probability of an attacker obtaining one block is $\frac{q}{p}$
- So, the probability of an attacker obtaining Z block is $\left(\frac{q}{p}\right)^{Z}$



Success = honest chain being extended by one block (+1)
Failure = attacker's chain being extended by one block (-1)
p = probability an honest node finds the next block
q = probability the attacker finds the next block
q_z = probability the attacker will ever catch up from z blocks behind

The probability attacker reaches breakeven is:

$$q_z = \begin{Bmatrix} 1 & \text{if } p \leq q \\ (q/p)^z & \text{if } p > q \end{Bmatrix}$$

CALCULATIONS

- $p + q = 1$
- $(p + q) * Q_z = Q_z$
- $p * Q_z + q * Q_z = p * Q_{z+1} + q * Q_{z-1}$
- $p * Q_{(z+1)} - p * Q_z = \ q * Q_z - \ q * Q_{z-1}$
- $p(Q_{z+1} - Q_z) = q(Q_z - Q_{z-1})$
- $(Q_{z+1} - Q_z) = \frac{q}{p} * (Q_z - Q_{z-1})$

CALCULATIONS

- After a transaction has been added to a block, the recipient will wait until z blocks have been linked to it.
- The recipient is uncertain about the precise progress that the attacker has made.
- However, the recipient makes an assumption that the honest blocks took the average expected time per block.
- the attacker's potential progress will be a Poisson distribution with expected
- value: $\lambda = z \frac{q}{p}$

CALCULATIONS

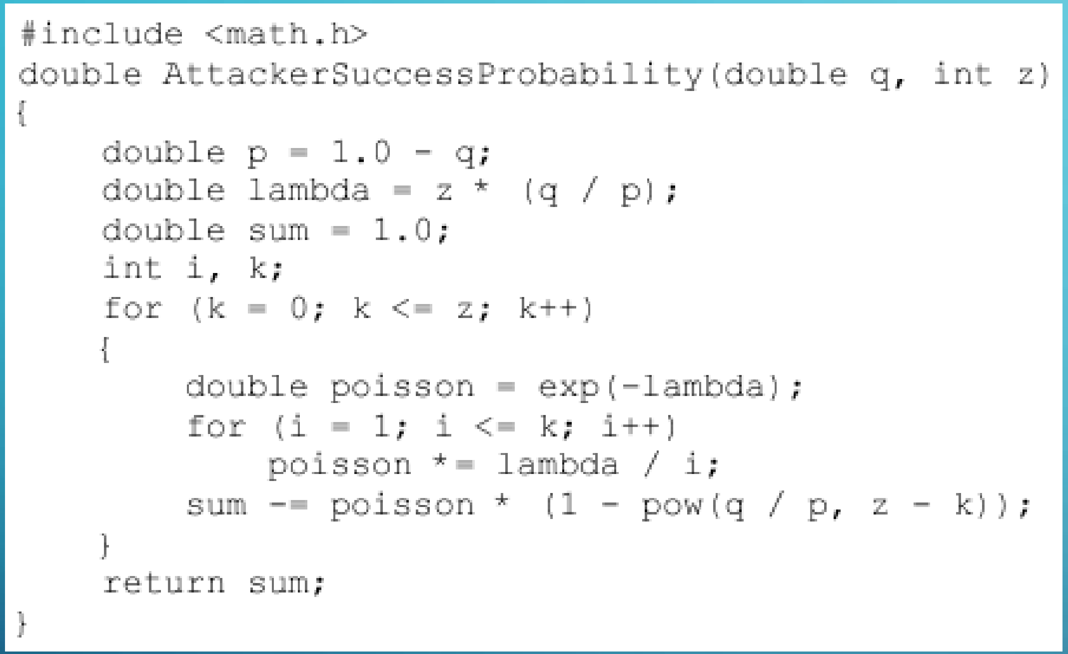
- To know the probability that the attacker could still catch up at this point, we need to multiply the Poisson density for each possible amount of progress the attacker have made by the probability that he could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \left\{ \begin{array}{cc} (q/p)^{(z-k)} & if\ k \le z \\ 1 & if\ k > z \end{array} \right\}$$

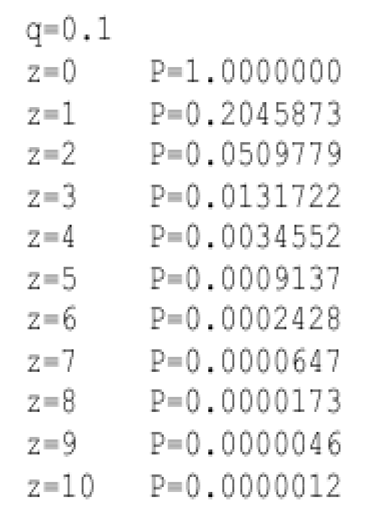
- Rearranging to avoid summing the infinite tail of the distribution…

$$1-\sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1-(q/p)^{(z-k)}\right)$$

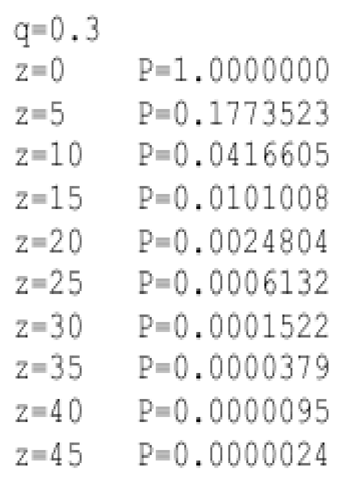
IMPLEMENTATION IN C



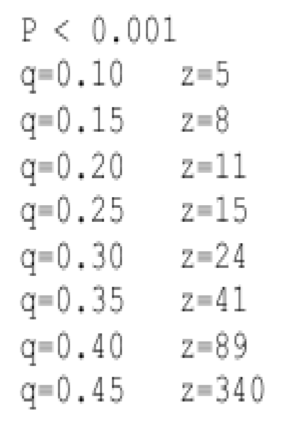
```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```



```
q=0.1
z=0     P=1.0000000
z=1     P=0.2045873
z=2     P=0.0509779
z=3     P=0.0131722
z=4     P=0.0034552
z=5     P=0.0009137
z=6     P=0.0002428
z=7     P=0.0000647
z=8     P=0.0000173
z=9     P=0.0000046
z=10    P=0.0000012
```



```
q=0.3
z=0     P=1.0000000
z=5     P=0.1773523
z=10    P=0.0416605
z=15    P=0.0101008
z=20    P=0.0024804
z=25    P=0.0006132
z=30    P=0.0001522
z=35    P=0.0000379
z=40    P=0.0000095
z=45    P=0.0000024
z=50    P=0.0000006
```



```
P < 0.001
q=0.10   z=5
q=0.15   z=8
q=0.20   z=11
q=0.25   z=15
q=0.30   z=24
q=0.35   z=41
q=0.40   z=89
q=0.45   z=340
```

CONTENT

CONCLUSION

- We talk about the design of a peer-to-peer electronic cash system.
- The system does not rely on trust and provides a solution for the double-spending problem.
- The system is based on digital signatures and a proof-of-work consensus mechanism.
- The proof-of-work mechanism is used to create a public history of transactions that is computationally impractical for attackers to change.

CONCLUSION

- The network is unstructured and simple, with nodes working together without coordination.
- Nodes can leave and rejoin the network at will, and their CPU power is used to accept or reject valid and invalid blocks.
- Any needed rules and incentives can be enforced with this consensus mechanism.