P2P Systems and Blockchains

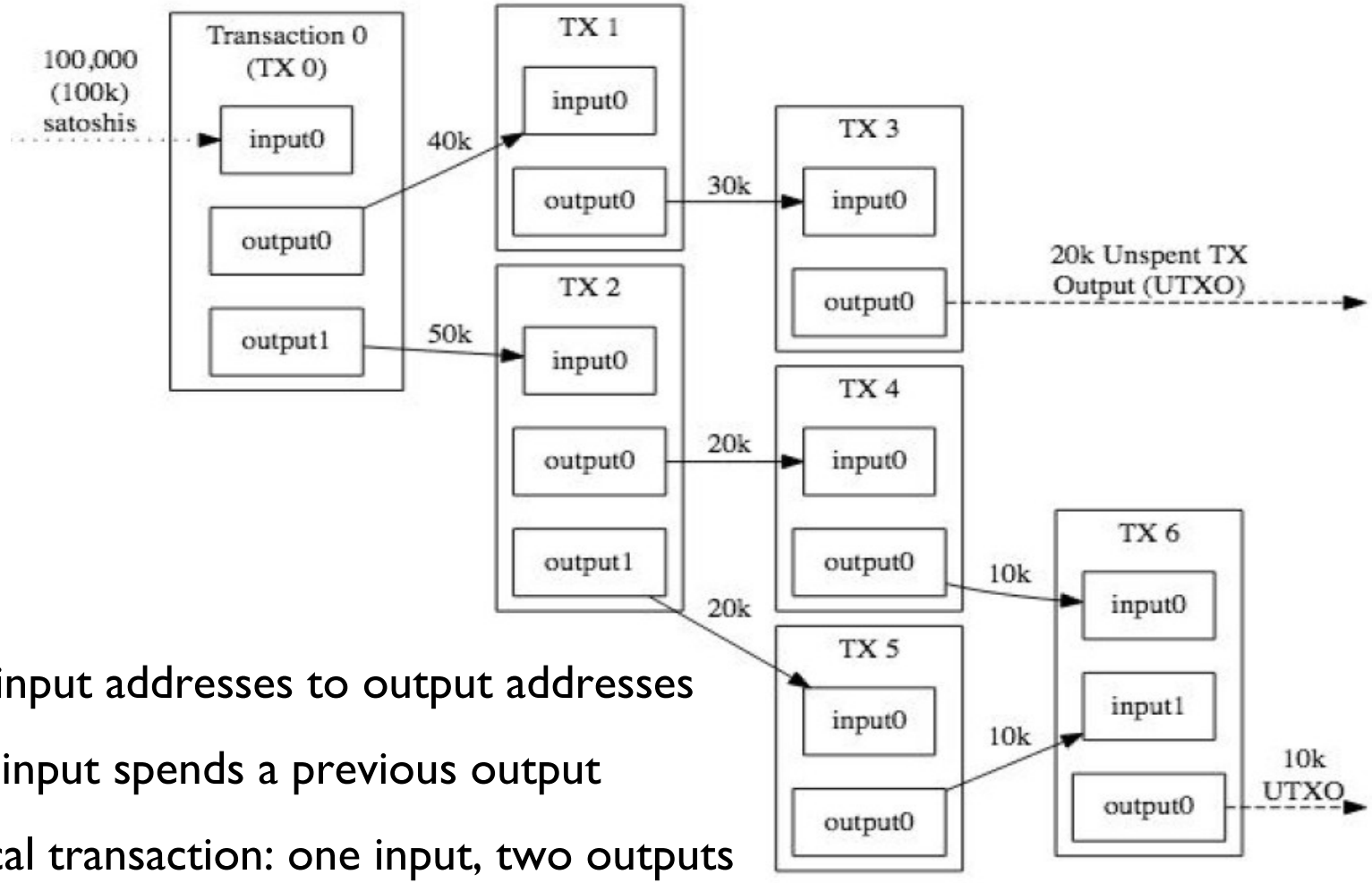
Spring 2018,

instructor: Laura Ricci
laura.ricci@unipi.it

Lesson 14:
BITCOIN:
THE NAKAMOTO CONSENSUS

20/4/2018

TRANSACTIONS: RECAP



- Map input addresses to output addresses
- Each input spends a previous output
- Typical transaction: one input, two outputs
- Contains signature of the owner of the funds

BITCOIN SCRIPTING LANGUAGE: DESIGN GOALS

- Input and output contain scripts
- Bitcoin has its own scripting language for transactions
  - inputs and outputs are scripts written in a **stack based, non-Turing complete** language
  - built for Bitcoin, **FORTH-LIKE**
- not Turing complete: no looping
  - no possibility to create "infinite loops"
  - all Bitocoin "full nodes" (miners and full nodes, notmobiles for instance) have to validate these scripts and they do not have to fall in a loop
  - preventing the transaction validation mechanism from being used as a vulnerability.
  - executable on a range of hardware,

BITCOIN SCRIPTING LANGUAGE: DESIGN GOALS

- stateless: there is no state prior to execution of the script, or state saved after execution of the script.
  - all the information needed to execute a script is contained within the script.
  - deterministic: a script will predictably execute the same way on any system.

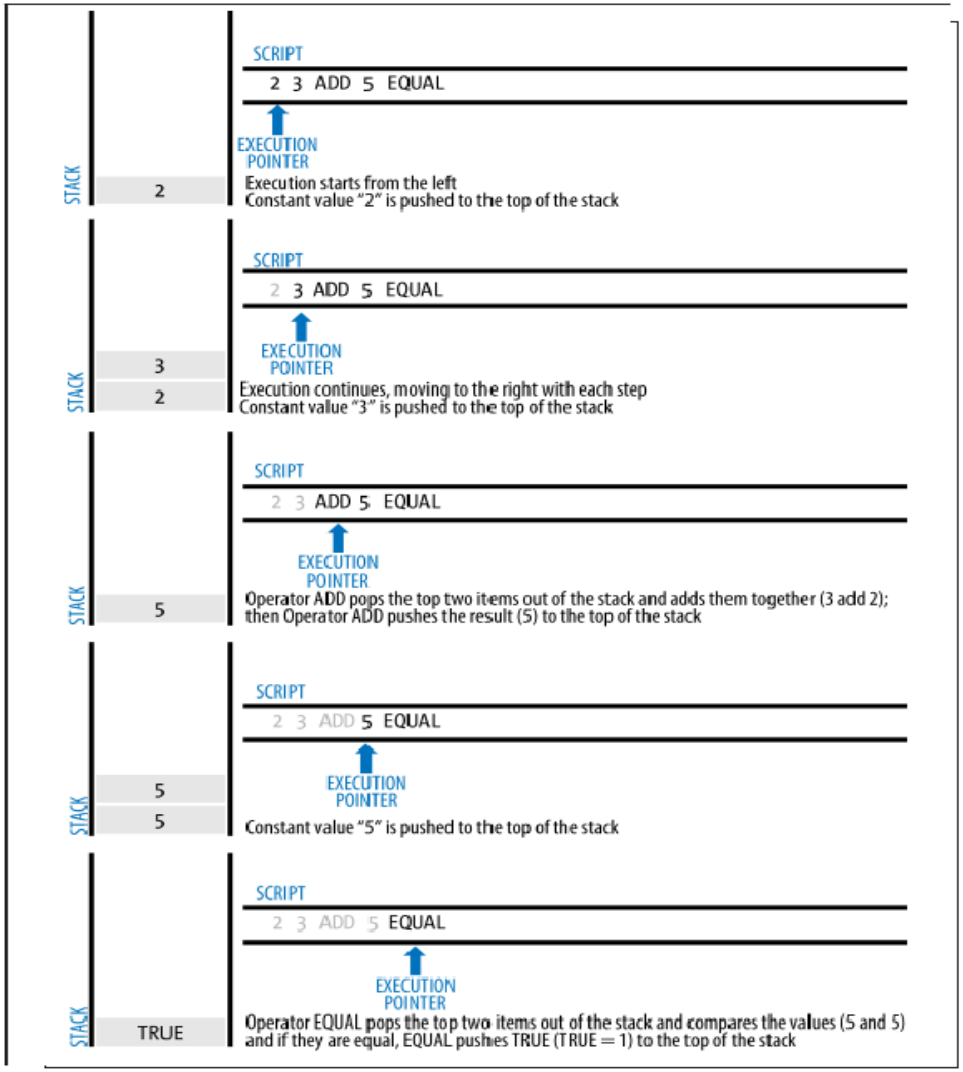
- simple, compact: one byte OPCODE, 256 instructions
  - basic arithmetic, basic logic (IF...THEN...ELSE), special purpose instructions to support cryptography
    - hashes
    - signature verification
    - multisignature verification

BITCOIN SCRIPTING LANGUAGE: DESIGN GOALS

There are 256 opcodes in total: 15 disabled, 75 working; they can be categorised as follows:

- Arithmetic, e.g. OP_ABS, OP_ADD
- Stack, e.g. OP_DROP, OP_SWAP
- Flow control, e.g. OP_IF, OP_ELSE (not while!)
- Bitwise logic, e.g. OP_EQUAL, OP_EQUALVERIFY
- Crypto for:
  - Hashing, e.g. OP_SHA1, OP_SHA256
  - (Multiple) Signature Verification, e.g. OP_CHECKSIG, OP_CHECKMULTISIG
  - Locktime, e.g. OP_CHECKLOCKTIMEVERIFY, OP_CHECKSEQUENCEVERIFY

STACK BASED EVALUATION



Pay-to-Public-Key-Hash (P2PKH)

- for example, Alice's pays 0.00015 bitcoin for a beer to Bob's Bar. Alice makes a payment to the Pub's bitcoin address.

- that transaction output contains a locking script of the form:

  OP_DUP OP_HASH160 <Bar Key Hash> OP_EQUAL OP_CHECKSIG

  where the Bar Key Hash is the bitcoin address of the Bar, the others are FORTH instructions

- The locking script can be satisfied with an unlocking script of the form:

  <Bar Signature> <Bar Public Key>

- The two scripts together would form a combines validation script, which must be executed by all nodes receiving the transaction:

<Bar Signature> <Bar Public Key> OP_DUP OP_HASH160
<Bar Public Key Hash> OP_EQUAL OP_CHECKSIG

P2PKH: SCRIPT EVALUATION

script_to_PubKey_hash (locks output):

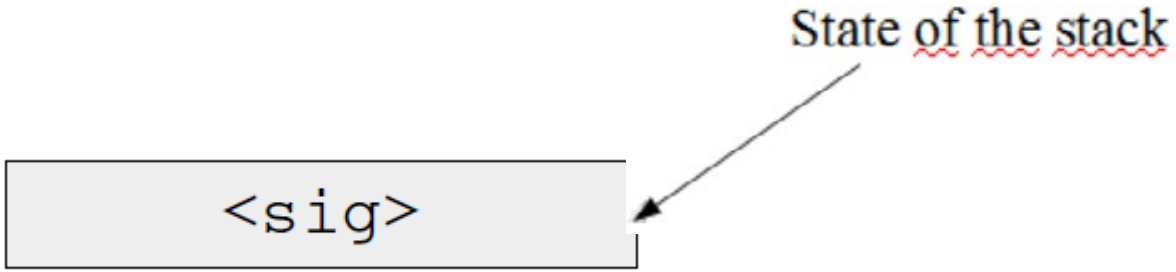
```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

scriptSig (unlocks output i.e. in input):

```
<sig> <pubKey>
```

script concatenation

```
<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG
```



P2PKH: SCRIPT EVALUATION

script_to_PubKey_hash (locks output):

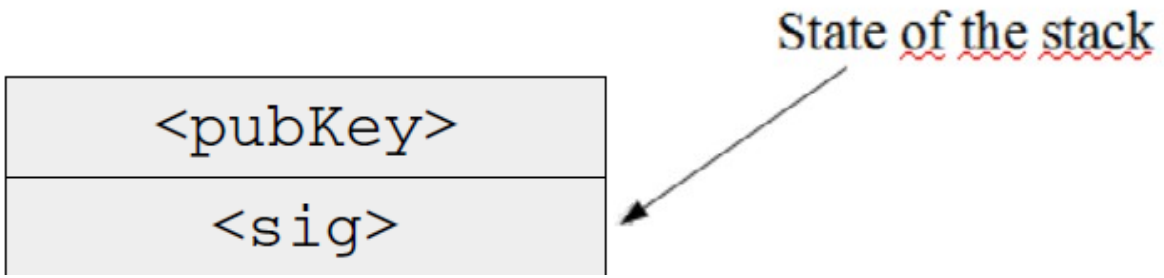
```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

scriptSig (unlocks output i.e. in input):

```
<sig> <pubKey>
```

script concatenation

~~<sig>~~ ~~<pubKey>~~ OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG



P2PKH: SCRIPT EVALUATION

scriptPubKey (locks output):

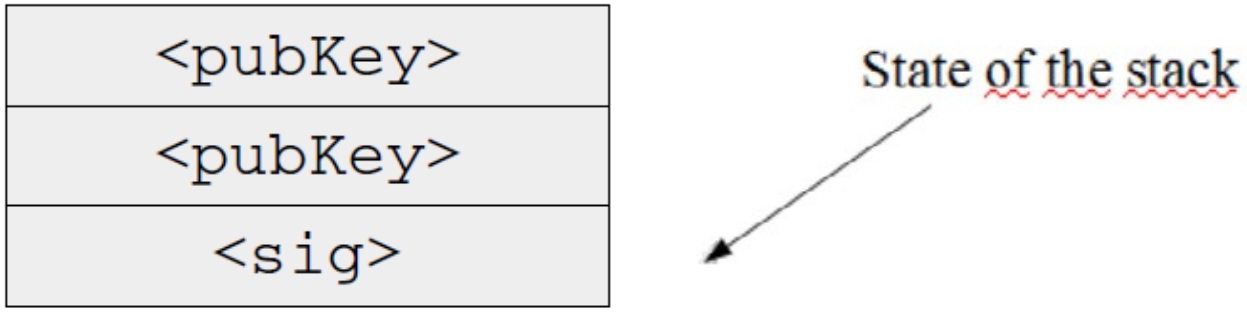
```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

scriptSig (unlocks output i.e. in input):

```
<sig> <pubKey>
```

script concatenation

~~<sig> <pubKey> OP_DUP~~ OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG



P2PKH: SCRIPT EVALUATION

scriptPubKey (locks output):

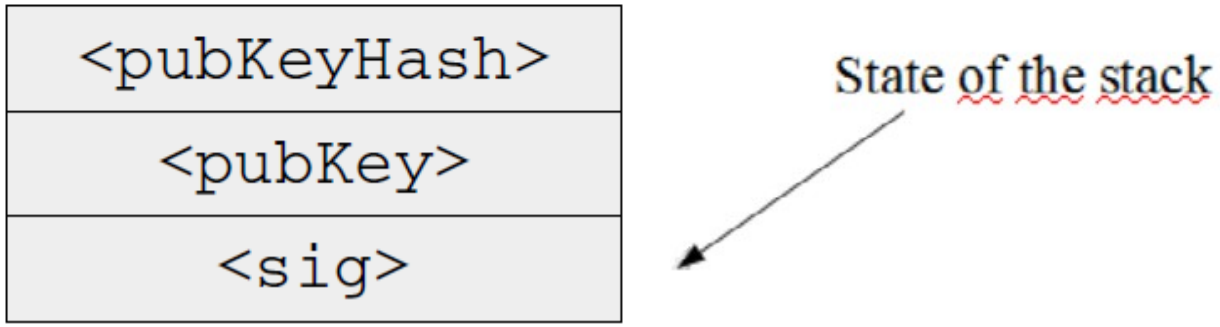
```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

scriptSig (unlocks output i.e. in input):

```
<sig> <pubKey>
```

script concatenation

~~<sig> <pubKey> OP_DUP OP_HASH160~~ <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG



P2PKH: SCRIPT EVALUATION

scriptPubKey (locks output):

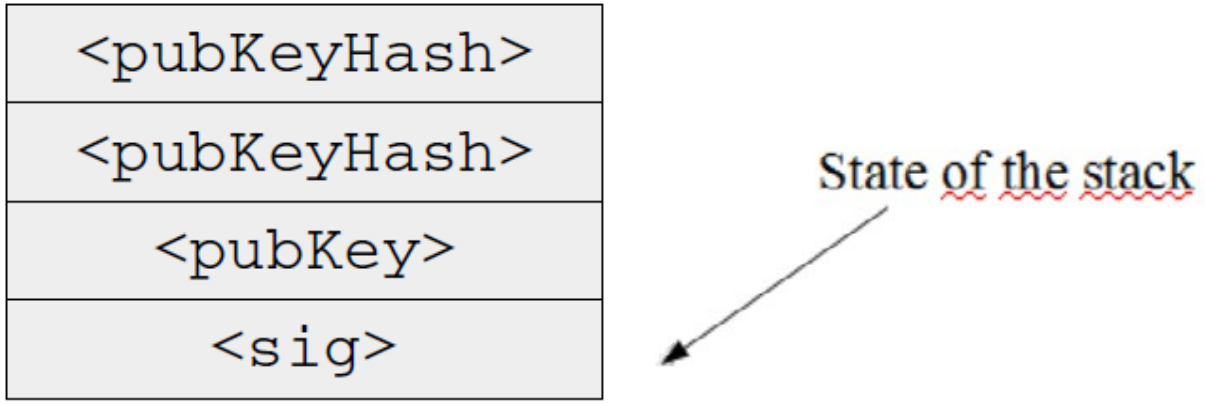
```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

scriptSig (unlocks output i.e. in input):

```
<sig> <pubKey>
```

script concatenation

~~<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash>~~
OP_EQUALVERIFY OP_CHECKSIG



P2PKH: SCRIPT EVALUATION

scriptPubKey (locks output):

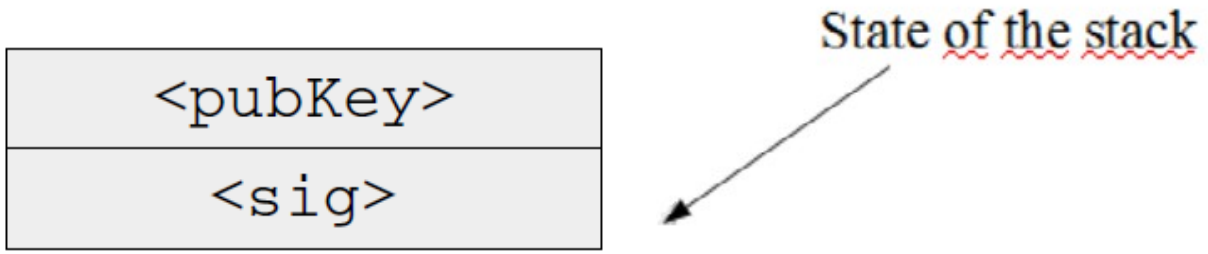
```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

scriptSig (unlocks output i.e. in input):

```
<sig> <pubKey>
```

script concatenation

~~<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash>~~
~~OP_EQUALVERIFY~~ OP_CHECKSIG



P2PKH: SCRIPT EVALUATION

scriptPubKey (locks output):

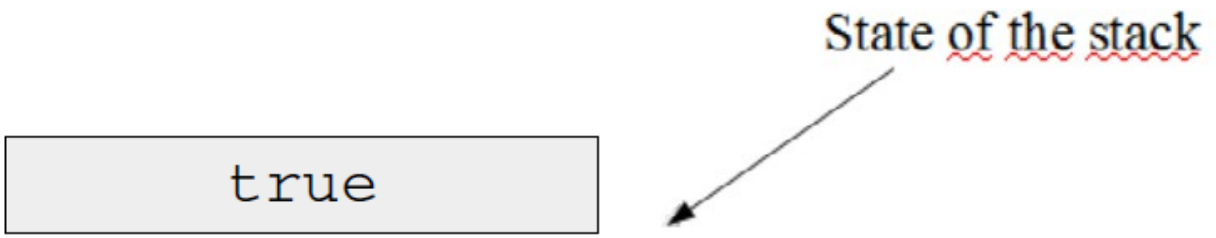
```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

scriptSig (unlocks output i.e. in input):

```
<sig> <pubKey>
```

script concatenation

~~<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG~~

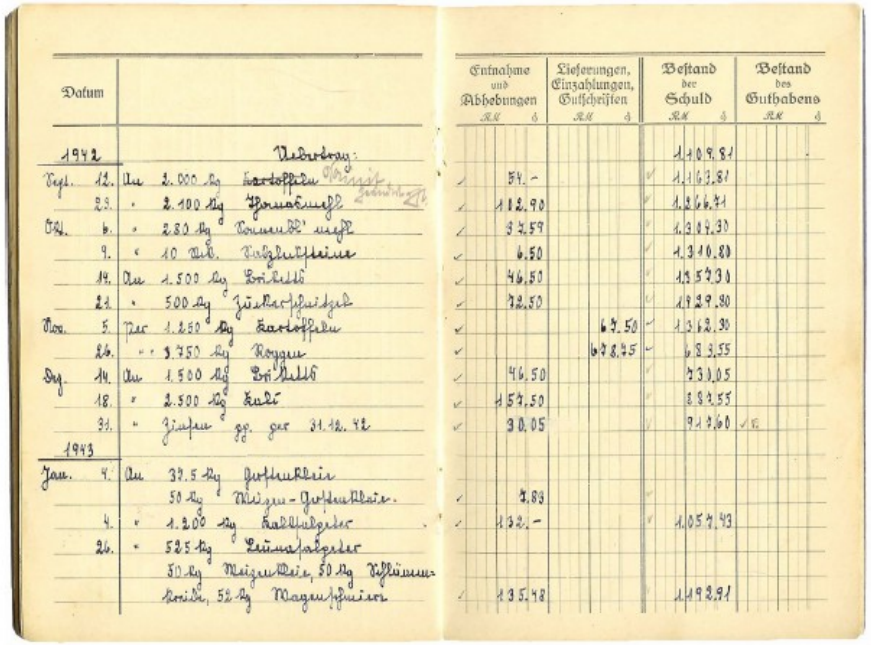


DATA OUTPUT (OP RETURN)

OP_RETURN <data>

- allows developers to add a certain amount bytes (40/80) of nonpayment data to a transaction output
- register the transaction output on the blockchain
- creates an explicitly provably unspendable output, which does not need to be stored in the UTXO set.
- OP_RETURN outputs are recorded on the blockchain, so they consume disk space and contribute to the increase in the blockchain's size
- they are not stored in the UTXO set, therefore do not increase the the RAM cost of full Bitcoin nodes

A DISTRIBUTED LEDGER OF TRANSACTIONS



- Bitcoin goal:
  - Maintain a ledger of transactions, like a bank, but.....
  - making everyone (collectively) the bank
- Define a shared public ledger recording all Bitcoin transactions.
  - everyone using Bitcoin keeps a complete record of which bitcoin belong to which person.
  - replicated on each Bitcoin user
  - eventually all the users have the same, consistent copy of the ledger
- this ledger is stored as a blockchain

A CONSISTENT LEDGER

- A simple (not working!) approach:
  - send each transaction to all the other peer of the P2P network
  - each peer register the transaction in its own ledger.
- Alice sends some bitcoins to Bob, she signs her transaction with her private key and broadcasts it on the P2P network. Bob
  - can use his copy of the ledger to check that, indeed, the bitcoin belongs to Alice and the public key of Alice to verify that the transaction comes from Alice
  - if all checks are ok, Bob will broadcast the transaction on the network
  - every peer updates its copy of the ledger with the new incoming transaction
- Alice can easily cheat by using this protocol
  - double spending her bitcoin
  - this naive protocol does not work!!

YOU CAN NOT DO IT BY YOURSELF!

- A double spending attack:
  - Alice sends a signed transaction to Bob
  - right afterwards sends the same bitcoins to Charlie
  - Bob's and Charlie's ledgers proves that the coin belongs to Alice
  - after checking the validity of the transations they both accept the transaction and broadcast it

- Double spending is detected if Charlie receives the notification from Bob before that the transaction from Alice, but.....
  - network does not maintain message ordering: transaction from Alice may arrive before notification from Bob
  - Alice may
    - use network traffic analysis and find times when there is a lot of latency in communications between Bob and Charlie
    - implement a DOS attack on the channel between Bob and Charlie

NEED OF DISTRIBUTED CONSENSUS

- take another approach: don't try to verify the transaction only by yourself, rather ask help to the network !

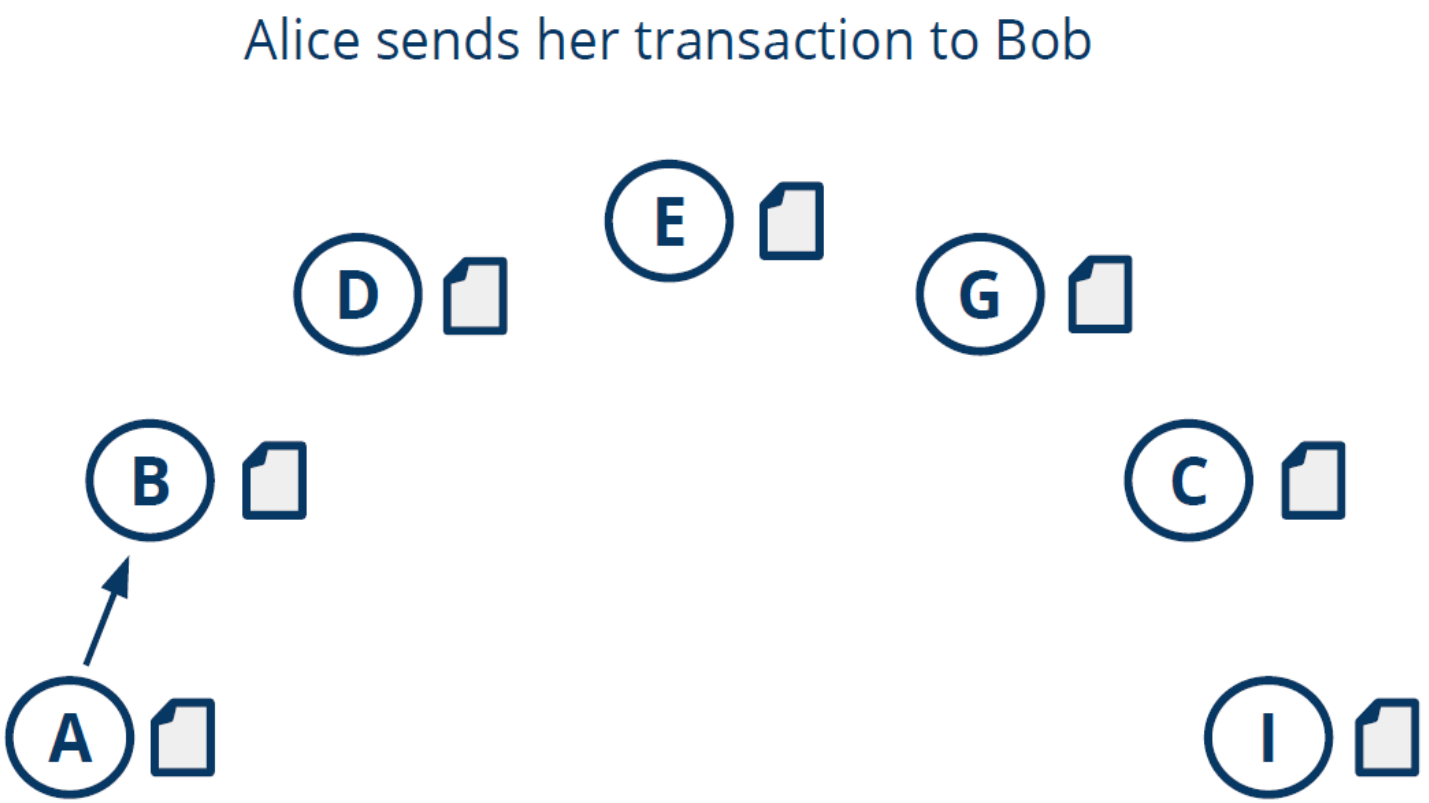
- requires an consensus algorithm, for instance:
  - all the nodes maintain
    - a ledger containing the transactions they have reached consensus on
    - a pool of transactions they have been received but not yet confirmed (not included in the ledger)
  - periodically, each node chooses a block of transactions to be included in the ledger
  - nodes collectively execute a consensus algorithm to decide
    - which block will be included in the ledger or to choose the one that will insert a block in the ledger
    - nodes may be malicious
    - what does majority mean in presence of malicious nodes?

NEED OF DISTRIBUTED CONSENSUS

Making everyone the bank.

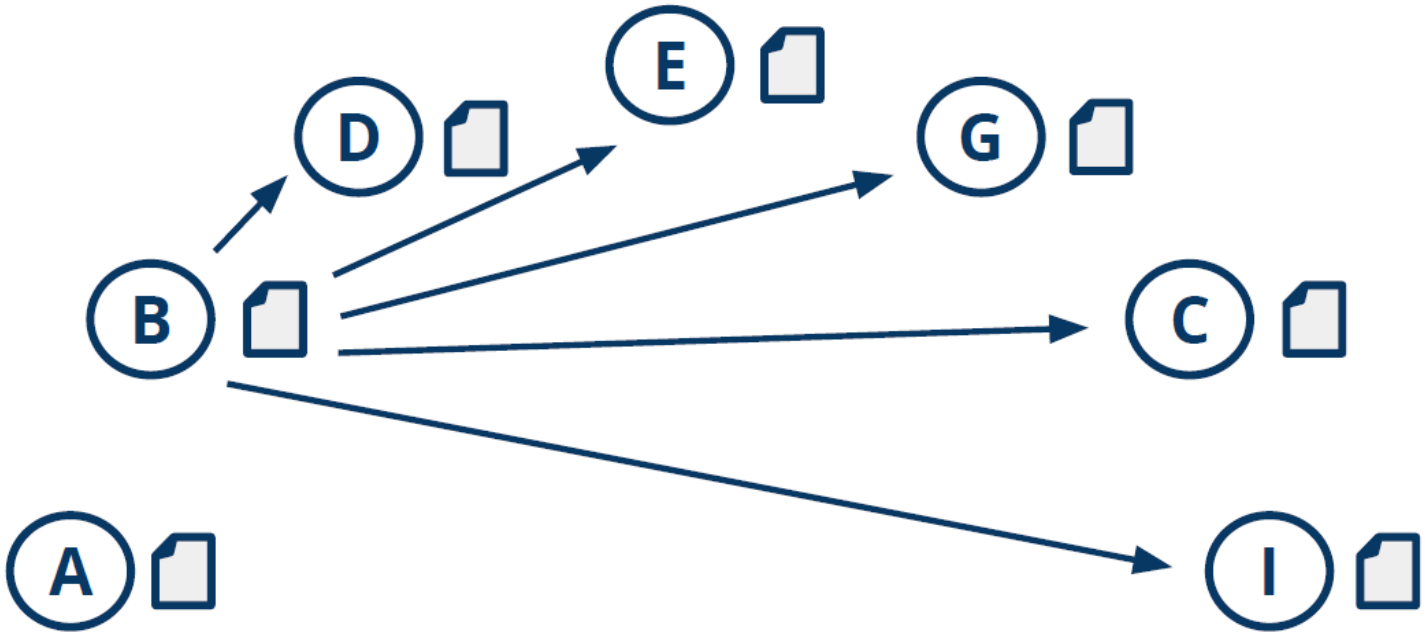
Everyone has a complete record of transactions

NEED OF DISTRIBUTED CONSENSUS

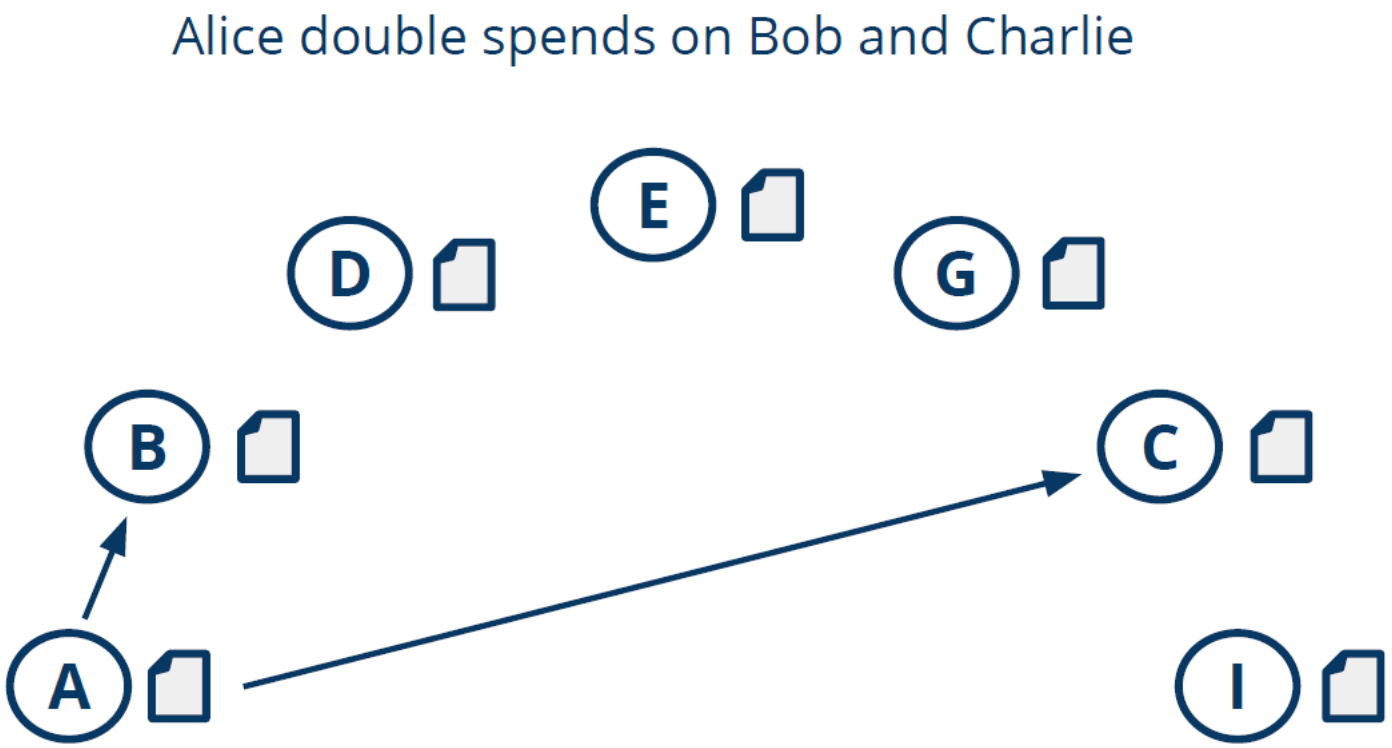


NEED OF DISTRIBUTED CONSENSUS

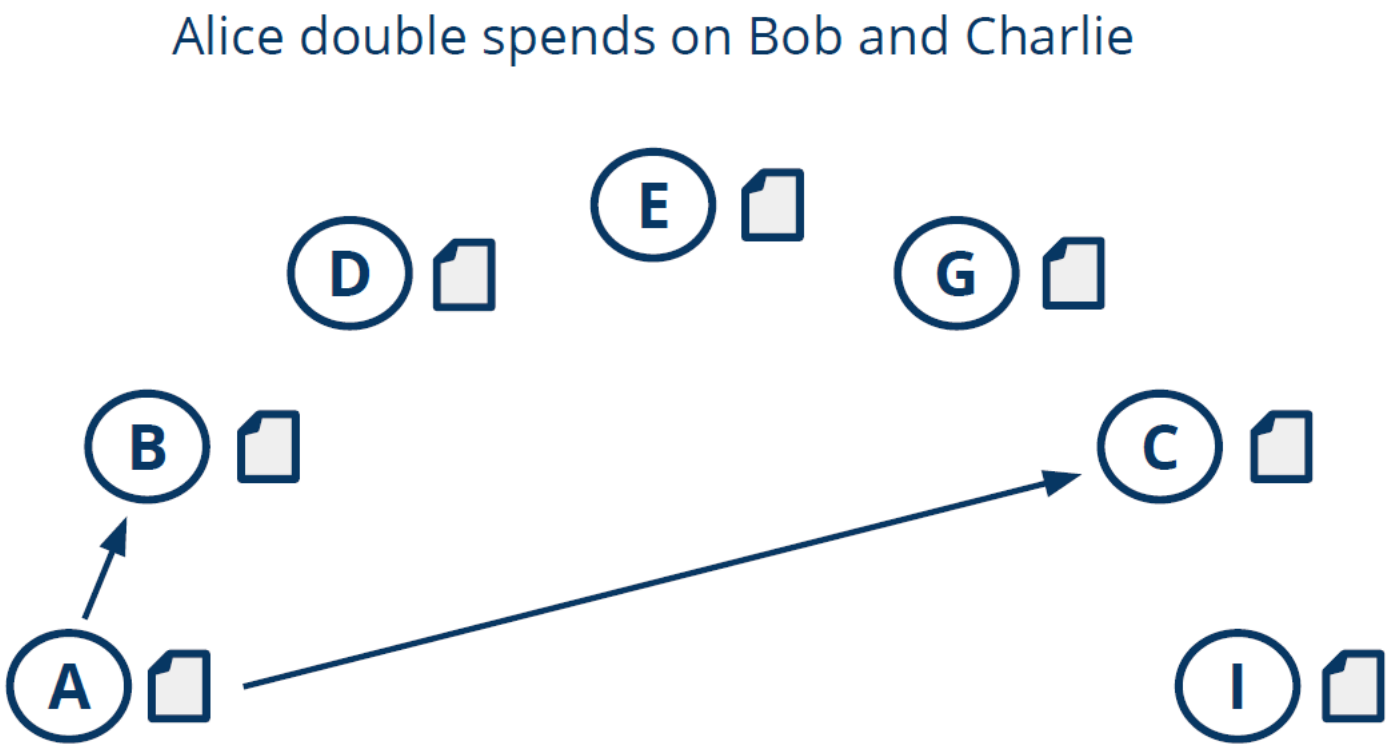
Bob announces the transaction to the world



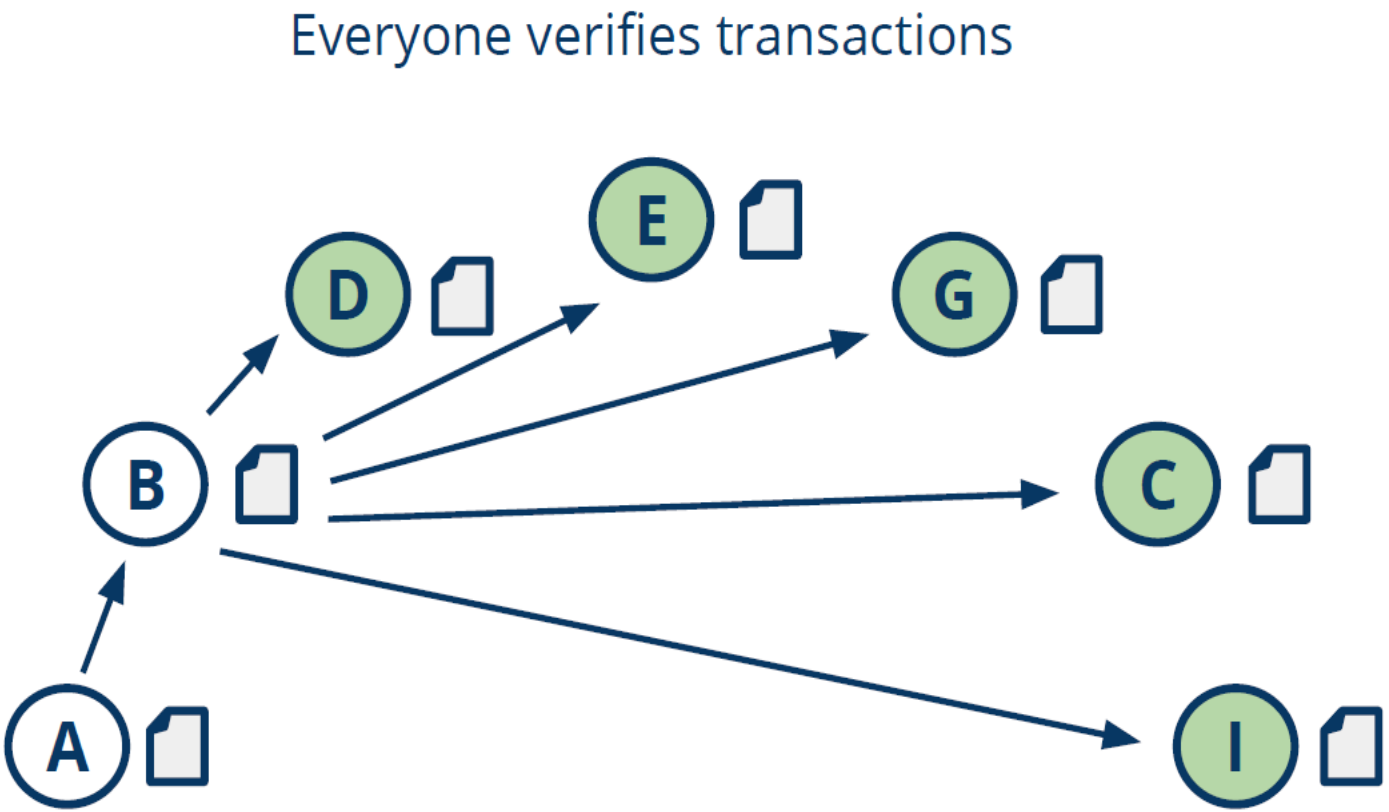
NEED OF DISTRIBUTED CONSENSUS



NEED OF DISTRIBUTED CONSENSUS

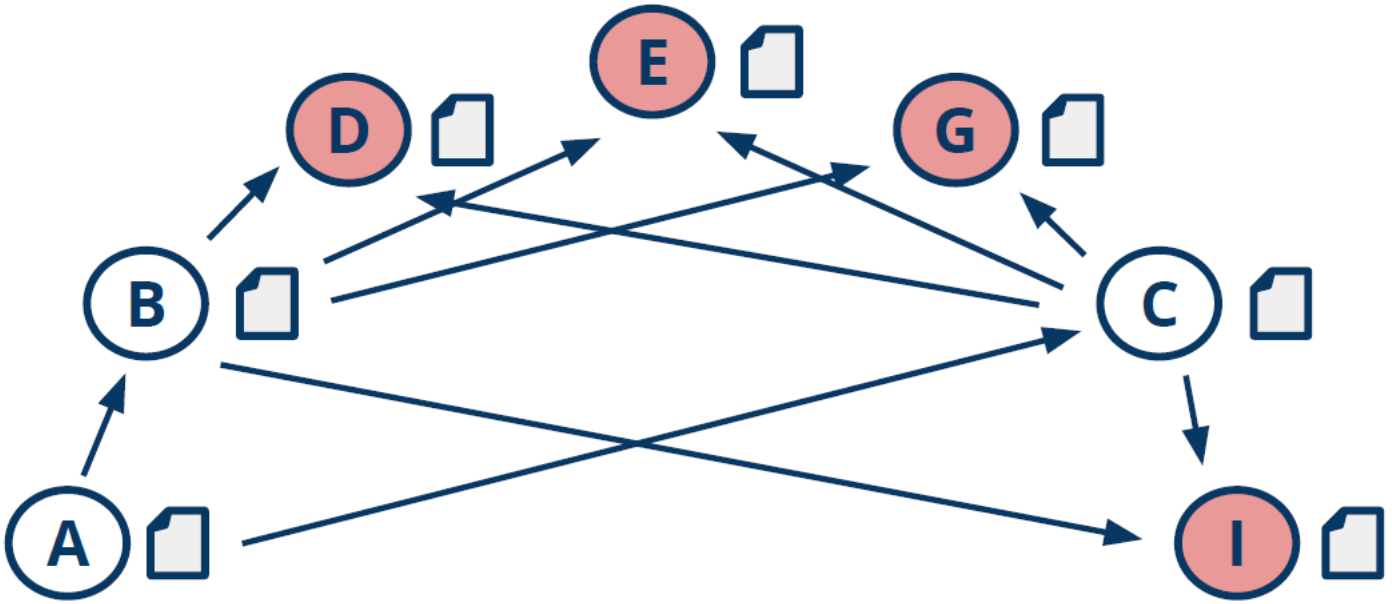


NEED OF DISTRIBUTED CONSENSUS



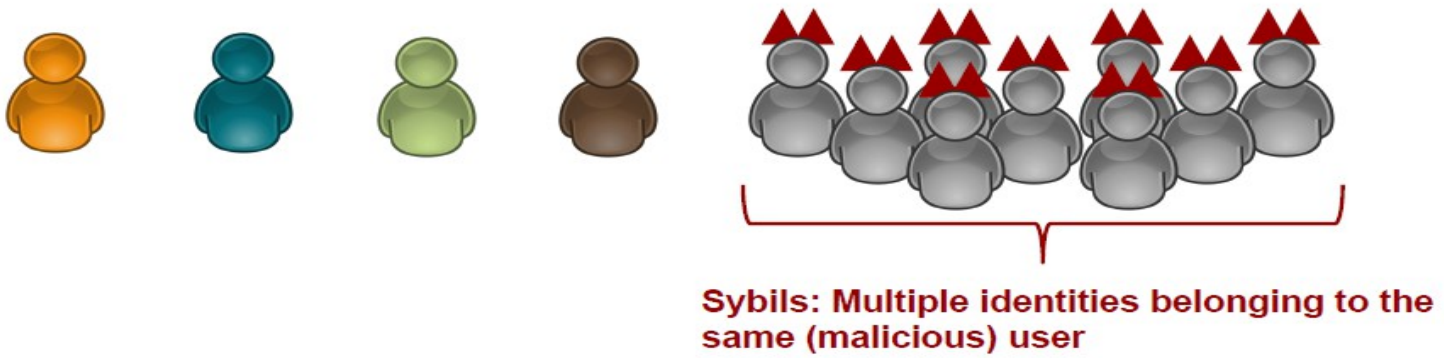
NEED OF DISTRIBUTED CONSENSUS

Alice is prevented from double spending



SYBIL ATTACK

- Sybil attack: in this attack a node in a P2P network claims multiple identities
- is avoided in systems where a central authority assigns identities to partecipants

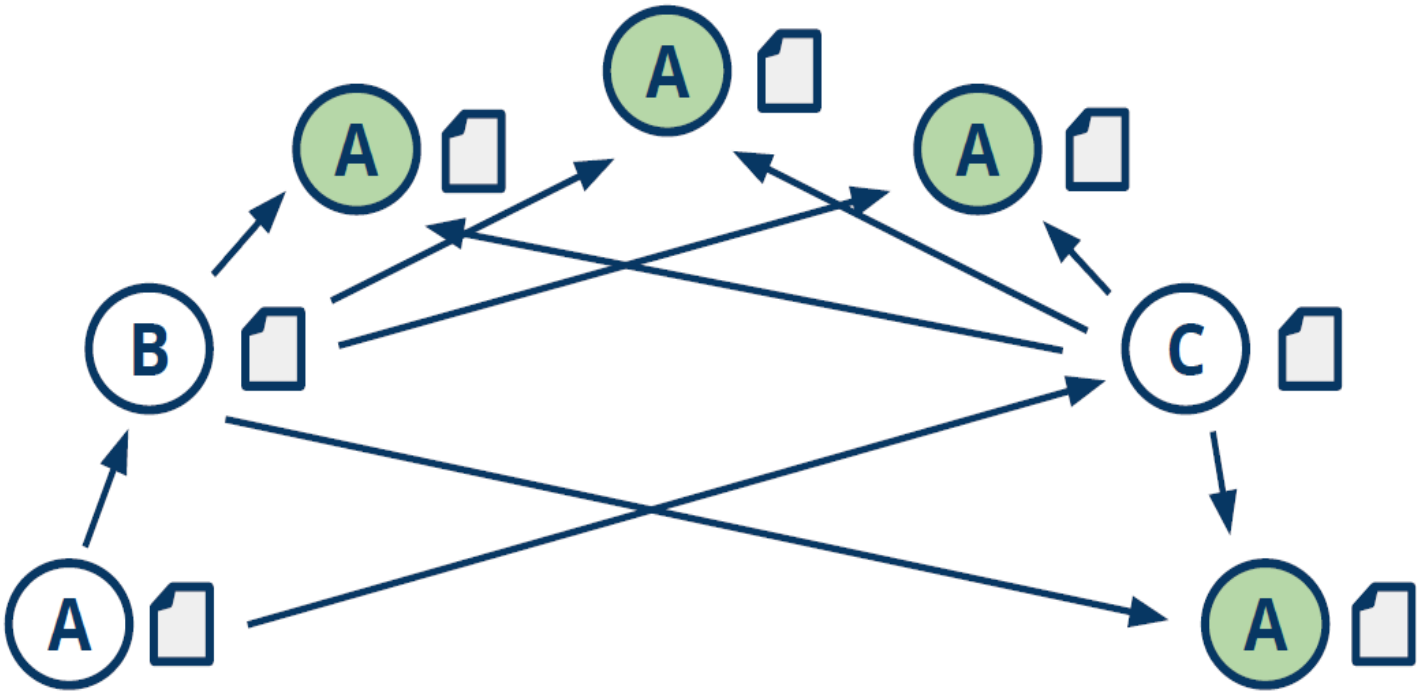


- consensus is even more hard in Bitcoin:
  - everyone is free to participate
  - because of pseudo-anonimity, nodes do not have persistent, long term identities,
  - a node can autonomously generate as many addresses as he/she wants

NEED OF DISTRIBUTED CONSENSUS

Alice double spends with her multiple identities

Sybil Attack: Creating many fake identities to subvert a system



CONSENSUS APPROACHES

Distributed consenus has been widely studied in the last 30 years

- Byzantine agreement
- Paxos
- Zab
- Raft
- Viewstamped Replication
- ……

EXPLICIT CONSENSUS AND BITCOIN

- The approaches in the previous slide exploit forms of explicit consensus: nodes execute a distributed algoritm to reach consensus
  - ...but they do not work for Bitcoin, why?

- in peer to peer networks, due to the high churn
  - nodes do not know each other a-priori
    - nodes come and go
    - anyone can join
  - no authenticated channels
  - no network synchronization

- even if we consensus has been studied for more than 30 years, relative little knowledge about this new model
  - classical algorithms, like Paxos, do not work.

A NEW SOLUTION: NAKAMOTO CONSENSUS

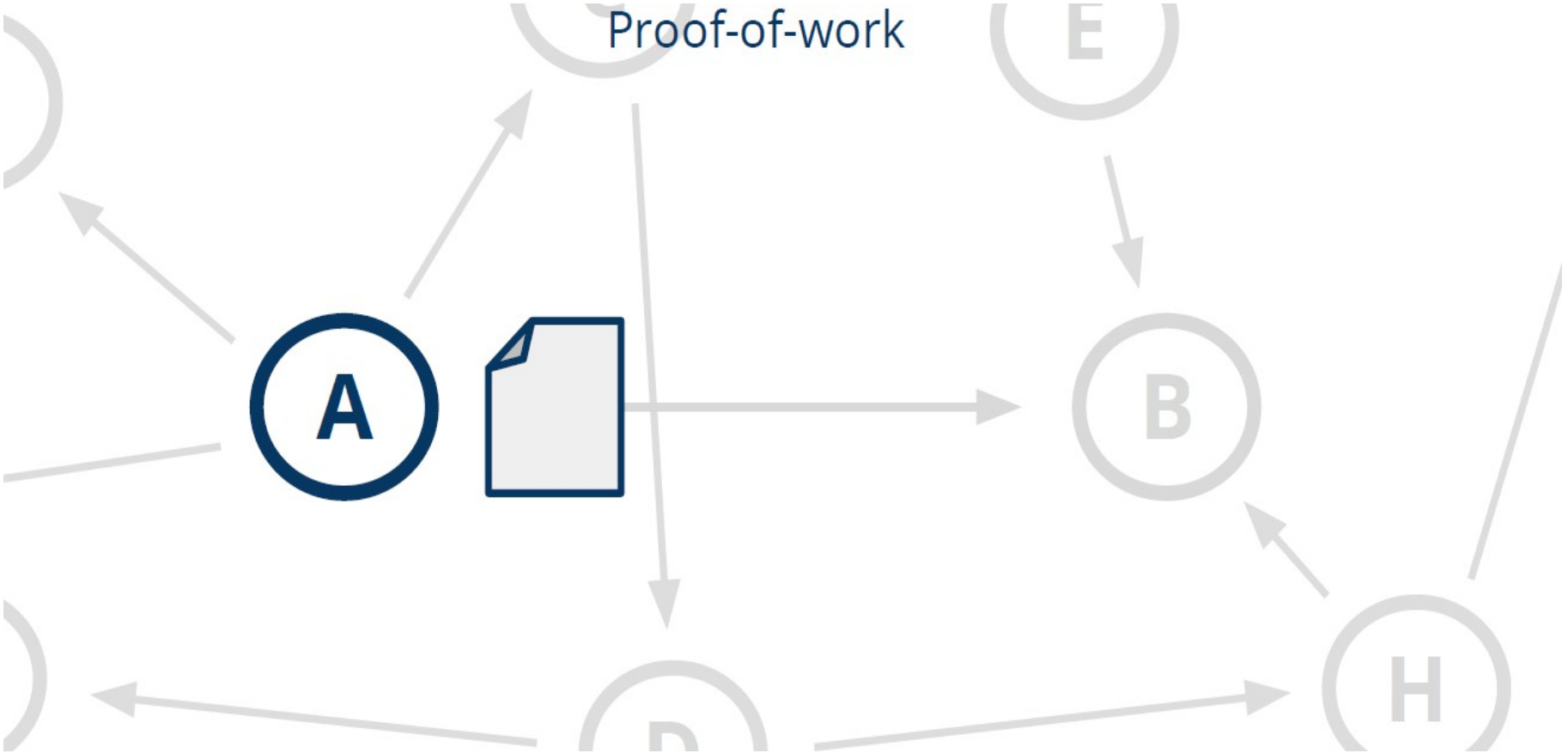
- prevents Sybil attacks with Proof-of-Work Puzzles
- claims blockchain mantains "public ledger" assuming "a particular notion of majority of honest nondes":
  - consistency: eventually everyone sees the same history
  - everyone can add new transactions
- it works in practice, difficult to make theoretical proofs (even if some work have been recently presented).
  - overcomes several impossibility result for unathenticated model

"*...the mechanism through which Bitcoin achieves decentralization is* ***not purely technical,*** *but it's a combination of technical methods and* ***clever incentive engineering***."

NAKAMOTO CONSENSUS

- let us suppose, for the moment, that:
  - it is possible to pick a random node in the network
    - like picking a random token in a lottery
  - at least 51% of the time, this process will pick an honest node.
- the consensus protocol:
  - at each round: select a node n at random
  - n unilaterally proposes, without contacting other nodes, the next block of transactions to be inserted in the ledger (from the unconfimed transactions)
    - can be viewed as a random leader election on every block
  - n broadcasts it in the network
  - all nodes check the validity of the block and update their blockchains with the new chosen block

NAKAMOTO CONSENSUS



The next transaction (block of transaction) is chosen by a single node, which should be, with high probability, a honest node

NAKAMOTO CONSENSUS

- An implicit consensus approach:
  - no collective distributed algorithm executed by the nodes
  - no voting
  - selection of malicious nodes is also implicitily handled by the system
- even if the nodes may have occasionally an inconsistent view of the ledger (blockchain forks) consensus will eventually occur
  - the consistent ledger will eventually be the longest chain
  - This is true if the majority of the nodes are honest

RANDOM NODE SELECTION

- how to select a random node at each round?
- the key idea: node are selected in proportion to a resource that it is hard to monopolize
- in Bitcoin this resource is computational power and selection is done on the basis of the Proof of work

Proof of work

- nodes which try to solve the proof of work are called miners and the whole validation process is called mining

PROOF OF WORK

- A proof of work (PoW) is a mechanism that allows a party to prove to another party that a certain amount of computational resources has been utilized for a period of time.
- Based on puzzles that can be solved, but require a considerable effort which cannot be short-circuited
- it must be possible to verify the effort made to solve a PoW in a easy way
  - verification requires less time with respect to the time needed to conduct the PoW
- many applications of the this technique:
  - deterring DOS attacks
  - deterring e-mail SPAM
  - Bitcoin mining

PROOF OF WORK AND DOS ATTACKS

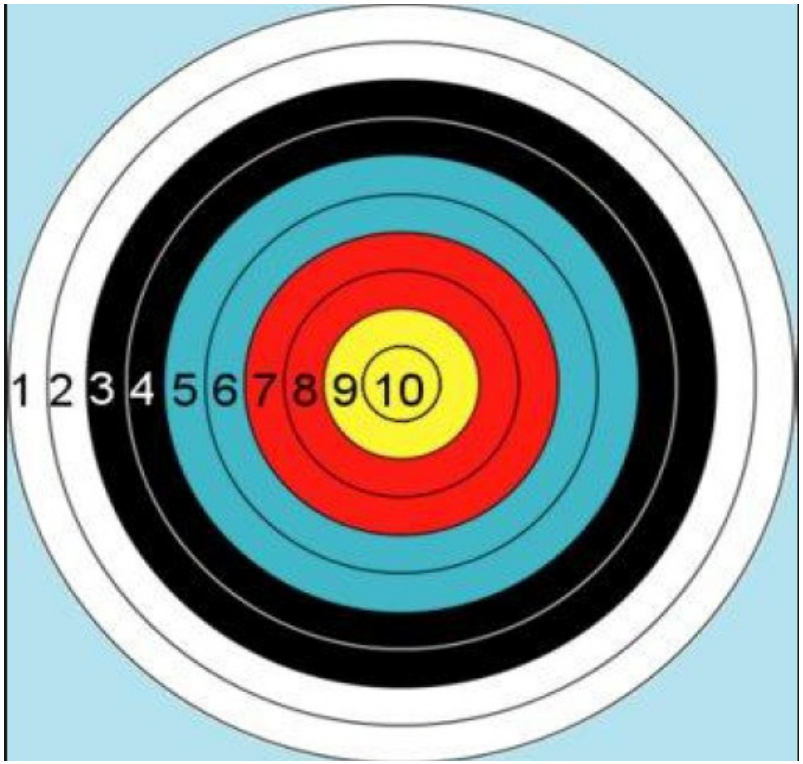
deterring of denial-of-service attacks:

- allow users access to a particular service only if they solve a highly computationally expensive problem
- the computational effort required is effectively a way to throttle the requesters
- it must be easy for the service provider to check if the requester carried out the required work
  - only if that work was carried out the service is given.

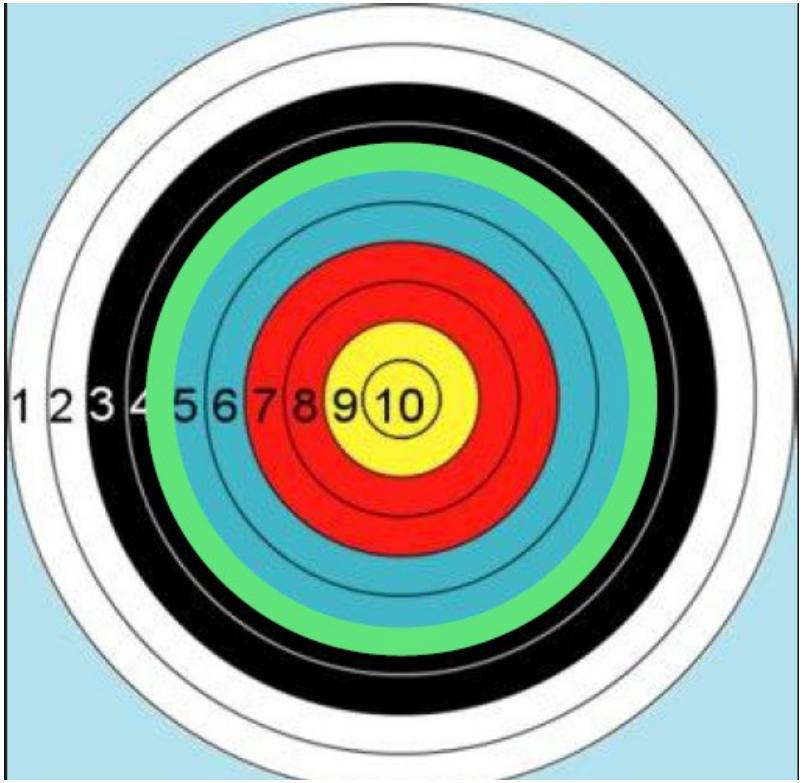
PROOF OF WORK AND EMAIL SPAMS

- a proof of work protocol may be tied to an email message
- like affixing a post stamp to a message
  - rather than paying for that stamp using money, pay for the stamp through CPU cycles
- if someone is sending out a small number of messages
  - the proof of work will not amount to a huge amount of work, because it is only executed a very small number of times
- for a spammer, who might be sending hundreds of thousands or millions of messages
  - it might be prohibitively expensive to use so many CPU cycles for each message it sends

PROOF OF WORK: A METAPHOR



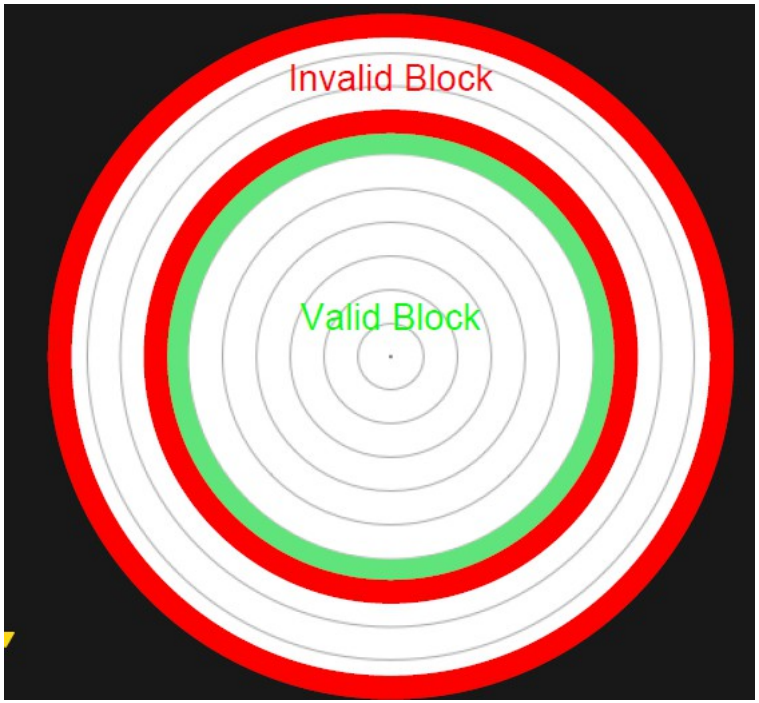
POW: TUNING THE DIFFICULTY



PROOF OF WORK: A METAPHOR

Proof of work is like throwing darts at a target while blindfolded:

- equal likelihood of hitting any ring
- faster throwers: more hit per seconds
- target: within the green ring
- difficulty: inversional proportional to green ring size
  - green ring adjuts depending on average time to produce valid results
  - If people get better at throwing darts, green circle needs to get smaller
  - It can be tuned to obtain the desired difficulty.



POW: A FORMAL DEFINITION

- Consider following notation:
  - d: difficulty, is a positive number which is used to adjust the time to execute the proof
  - c: challange is a string
  - x: nonce is a string

- a Proof-of-Work is a function:

$$F_d(c,x) \rightarrow \{true,false\}$$

which satisfies the following properties:

  - d and c are fixed
  - find x such that $F_d(c,x)$ = true is computationally difficult, but feasible.
  - $F_d(c,x)$ is fast to compute, if d, c, and x are fixed

PROOF OF WORK: HOW DOES IT WORK?

- given challenge string, c, define a proof which is tied to this string
- the challange string may be:
  - for spams, the whole message.
  - for Bitcoin: the header of a block containing a set of transactions
- the prover must find and return a proof nonce string
  - the prover must give a response that has a specific mathematical property in relations to the challenge
- for instance: given the challange c find a nonce p such that

  the output of a cryptographic hash (HASH-256) applied to the concatenation of c and p satisfies some property
- example of property: the output must have a prefix of 0 of predefined length

BITCOIN PROOF OF WORK

$$\mathcal{F}_d(c, x) \rightarrow \mathrm{SHA256}(\mathrm{SHA256}(c|x)) < \frac{2^{224}}{d}.$$

- double hash of the concatenation of the challenge c and the nonce x, using SHA256
- find the nounce such that the hash of the entire block (including the nonce) starts with a given number of zero bits.
  - by incresing d the number of zeros in 2^{224}/d decreaeses
- the output of SHA256 is a cryptographic hash with a numeric value in {0.. 2^{256-1}} which is compared to a target value 2^{224}/d , which gets smaller, when increasing the difficulty.
- In Bitcoin

SHA256(SHA256(prev_hash || BlockHeader || nonce))<target

PROOF OF WORK: HOW DOES IT WORK?

- In a good cryptographic hash function the only known way to find the response string is to try a huge number of distinct alternatives.
  - brute force: trying a lot different proposed strings until you find one that works
  - If we require that the output has a prefix of x consecutive zeros this would require, on the average, to try 2^x possibuilities
    - but if you are lucky, you will resolve it faster
  - a lot of computational power !

- But... it is easy to be verified: if someone proposes a nounce, it is very easy to validate it is a correct proof of work
  - take the challange and the nounce and hash them together
  - if the output produced by the PoW has the required number of leading zeros, the PoW is valid

PROOF OF WORK: HOW DOES IT WORK?

Why brute force is the only possible approach?

- what makes proof of work hard to solve?
  - the output from a cryptographic hash function behaves like a random number
  - each output bit looks like coin flips (0/1)
  - change the input a bit and the output from the hash function changes completely
- so no better way of finding the correct output than trying by brute force
  - the probability to find out the right output is very small!

PROOF OF WORK: HOW TO SOLVE IT?

An example of a simplified PoW

- c = "Hello, world!" and nonce x = 0 then (output is in hexadecimal)

h("Hello, world! 0") = 1312Af178c253f84028d480a6adc1e25e81caa44c749ec81976192e2ec934c64

- first with x = 0 is a failure, since the output does not begin with any zeroes at all, try nounce x = 1

h("Hello, world! 1") = e9afc424b79e4f6ab42d99c81156d3a17228d6e1eef4139be78e948a9332a7d8

- keep trying different values for the nonce, x = 2, 3,.... finally, at x = 4250 obtain:

h("Hello, world! 4250") = 0000c3af42fc31103f1fdc0151fa747ff87349a4714df7cc52ea464e12dcd4e9

success if the proof of work requires 4 leading zeros !

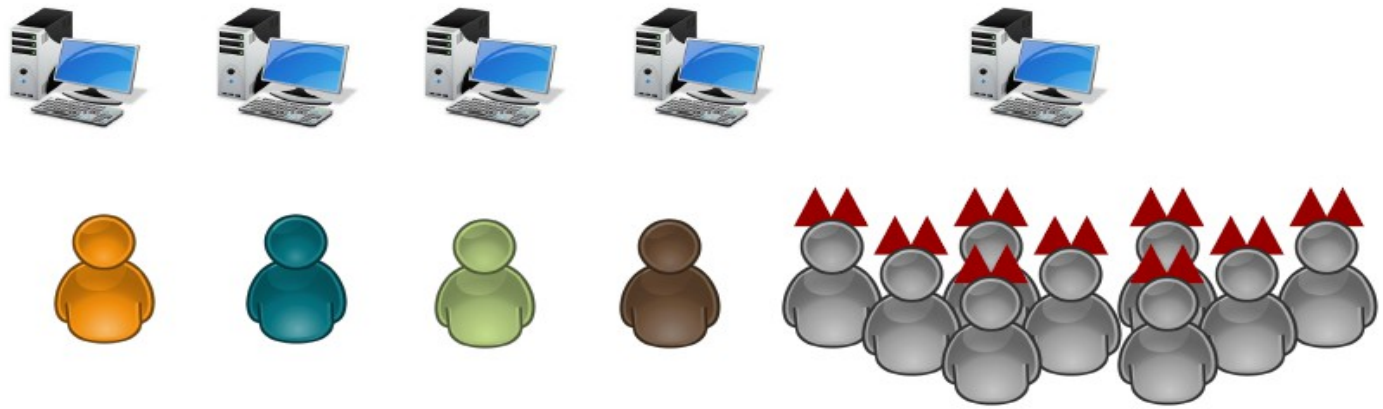
MINING A BLOCK

- But who is solving the PoW?
- Bitcoin miner must:
  - verify incoming transactions by checking signatures
  - create a block of transactions collected valid transactions
  - find the nounce, starting from the block header (the core mining process)
  - hope that your block is accepted by other nodes and not defeated by competitor nodes
  - if its is accepted, reward is yours!

NAKAMOTO CONSENSUS AND SYBIL ATTACKS

- Benefit of making it costly to propose new blocs of validated transactions: validation
  - is no longer be influenced by the number of network identities the attacker controls
  - influenced instead by the total computational power the attacker brings to bear on validation.
- a cheater would need enormous computational resources to cheat, making it impractical
  - sybil attack becomes difficult to implement
- reformulate the initial hypothesis:
  - require that the majority of miners, weighted by hash power, are honest, i.e. follow the protocol
  - further, give incentives to miners for being honest!

NAKAMATO CONSENSUS AND SYBIL ATTACKS



- majority of honest nodes refer to majority of miners
- majority defined as the majority of computational power
- note that Sybil creation does not increase attackers computational power !

MINING PROCESS: RECAP

A simplified proof of work algorithm

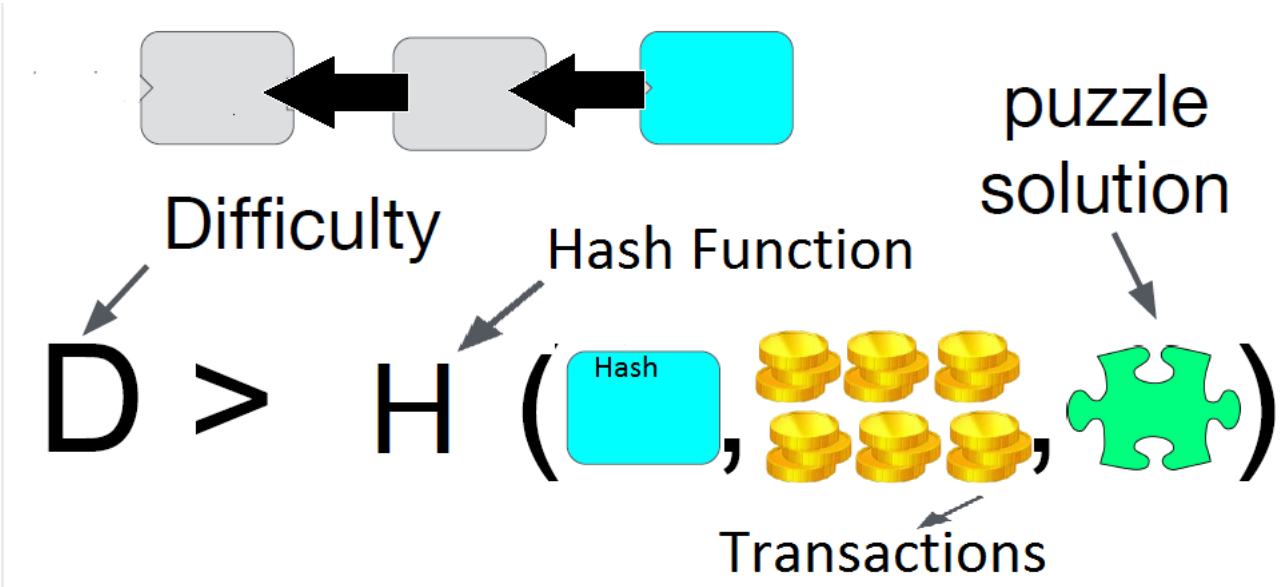
Nonce x = 0, challenge c, difficulty d, previous block b_{t-1}

repeat

x = x + 1

until $F_d(c, x)$ = true

broadcast block b_t = (memory-pool, b_{t-1}, x)



MINING PROCESS: RECAP

A simplified proof of work algorithm

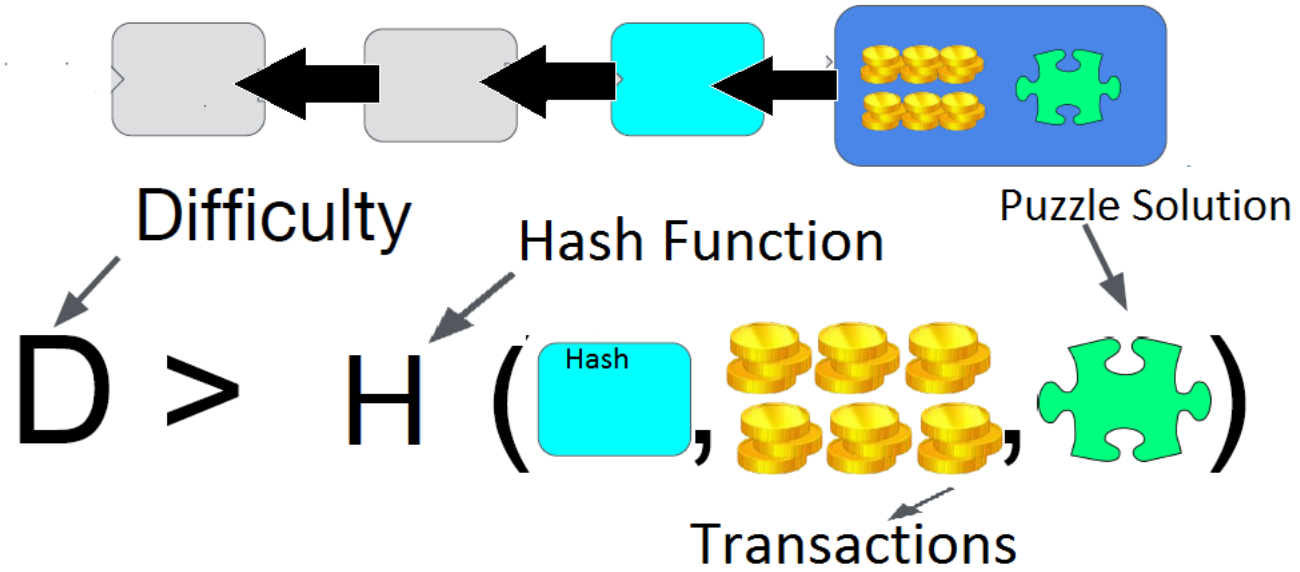
Nonce x = 0, challenge c, difficulty d, previous block b_{t-1}

repeat

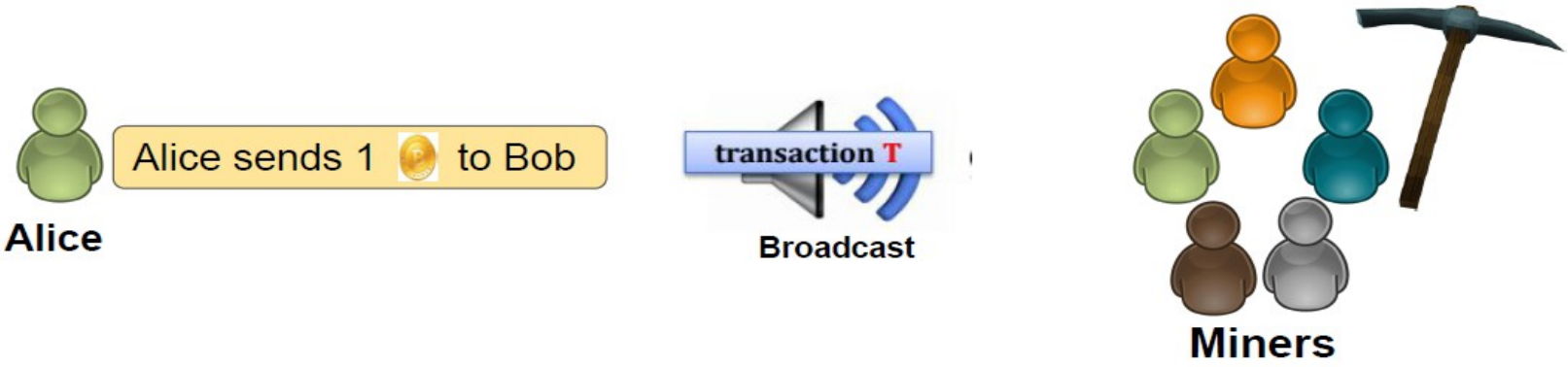
x = x + 1

until $F_d(c, x)$ = true

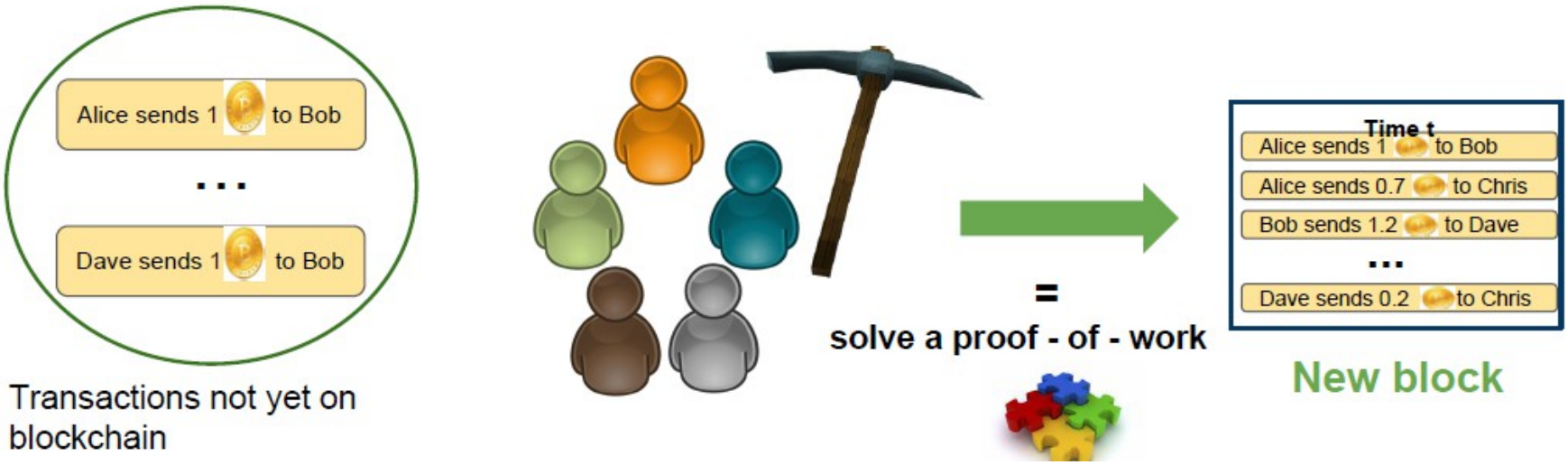
broadcast block b_t = (memory-pool, b_{t-1}, x)



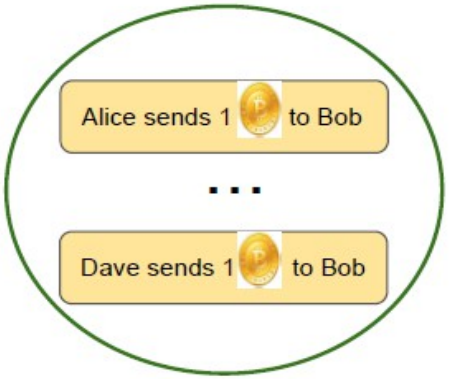
MINING PROCESS: RECAP



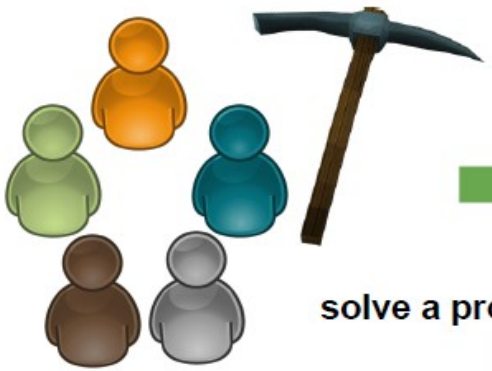
Miners compete on who will be able to propose the next block.



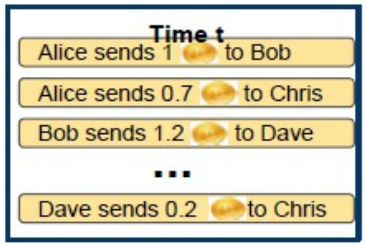
MINING PROCESS: RECAP



Transactions not yet on blockchain

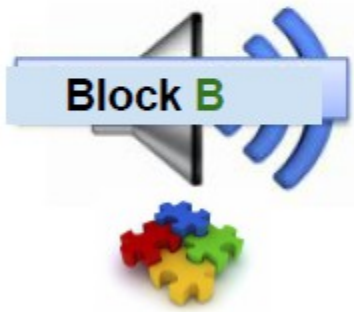


= solve a proof - of - work



New block

I found a new block!

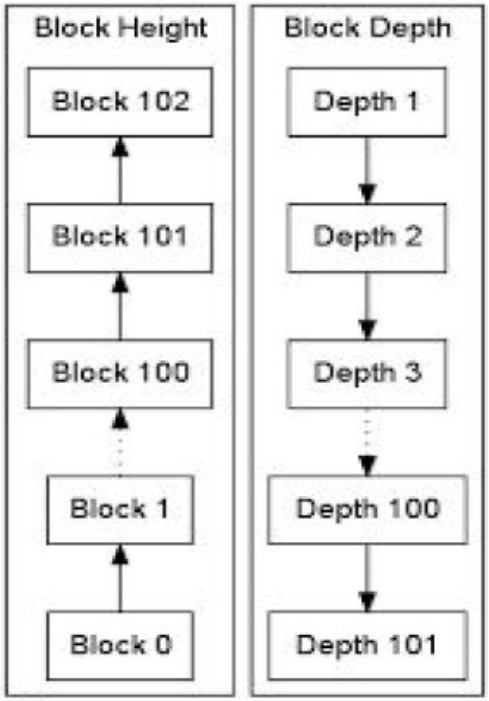


And to other nodes

CREATINGA BLOCKCHAIN

- A miner which can to solve the PoW, for a block, links the block to the previous one
  - The block contains the hash of the previous one

- The result is a chain of blocks, the blockchain
  - block: contains a bunch of transactions
  - the entire set of blocks chained together

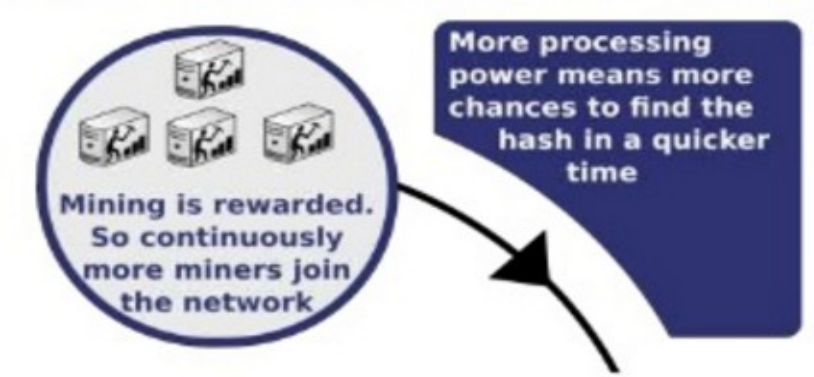


ADJUSTING THE POW DIFFICULTY

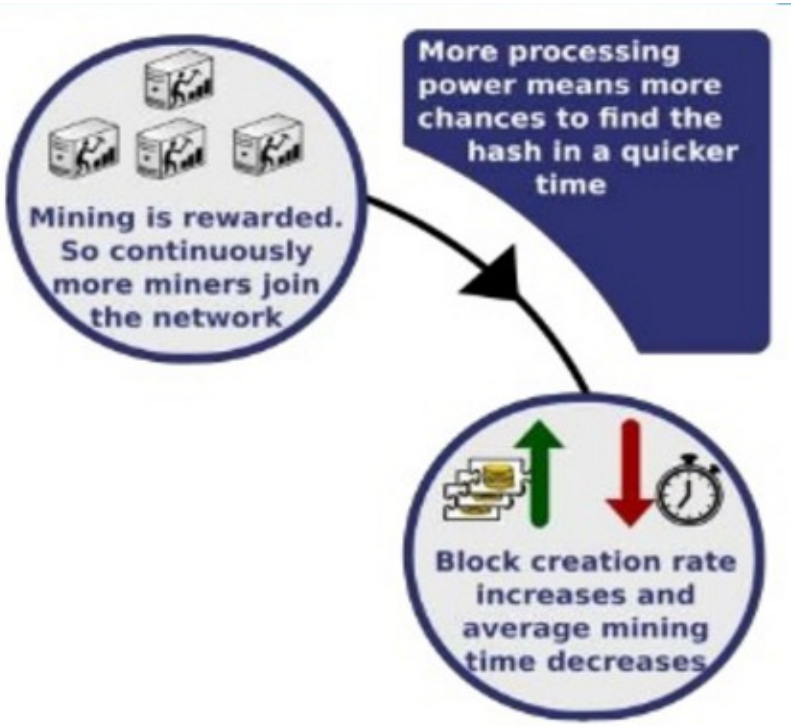
- one of the goals of the Bitcoin protocol is to have a block mined every 10 minutes
- mining frequency depends on the difficulty of the PoW and on the number of miners power n the network
- mining power is continously added to the network
  - miners can enter the network whenever they want
  - miners are incentivates to enter, becuse of
    - the fees they receive
    - a reward for having mined a block
  - no centralized control
  - the more miner, the more computational power, the more it is easy to find a block
- the time to find a new block decreases

ADJUSTING THE POW DIFFICULTY

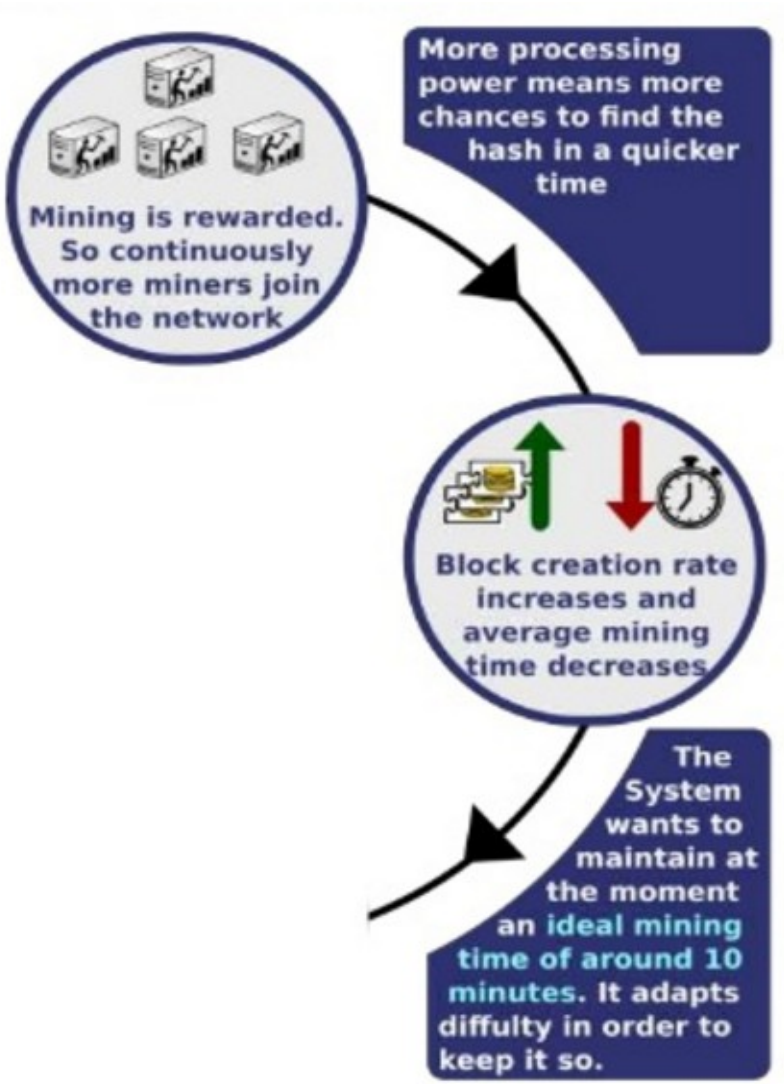
ADJUSTING THE POW DIFFICULTY



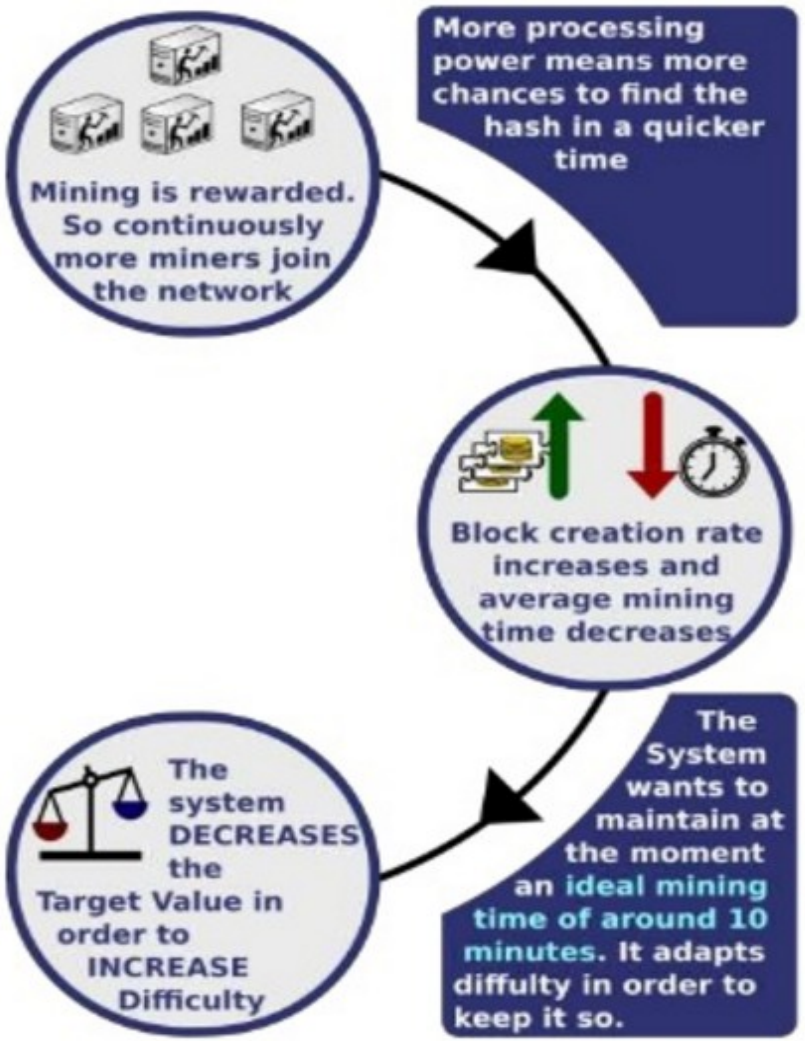
ADJUSTING THE POW DIFFICULTY



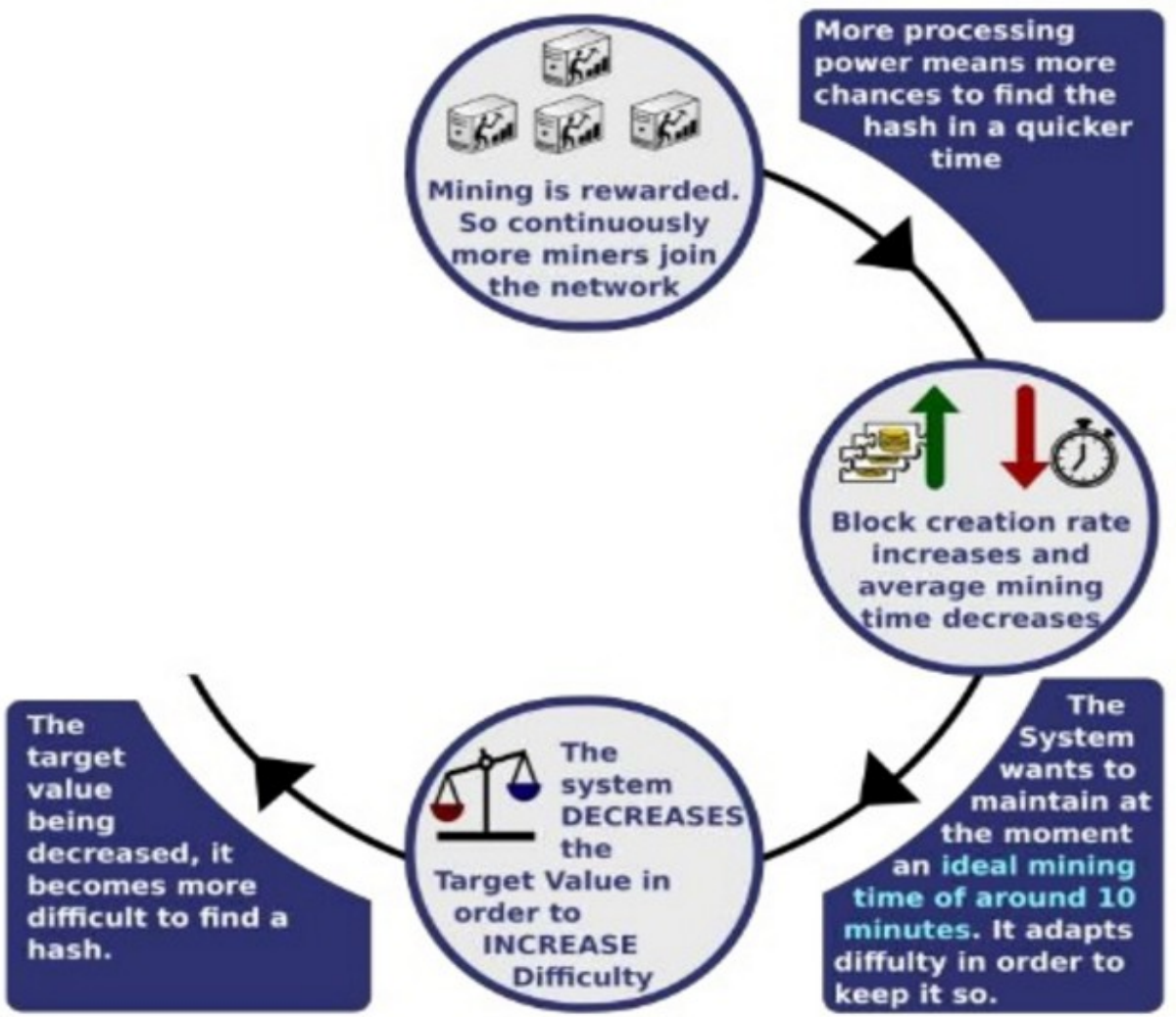
ADJUSTING THE POW DIFFICULTY



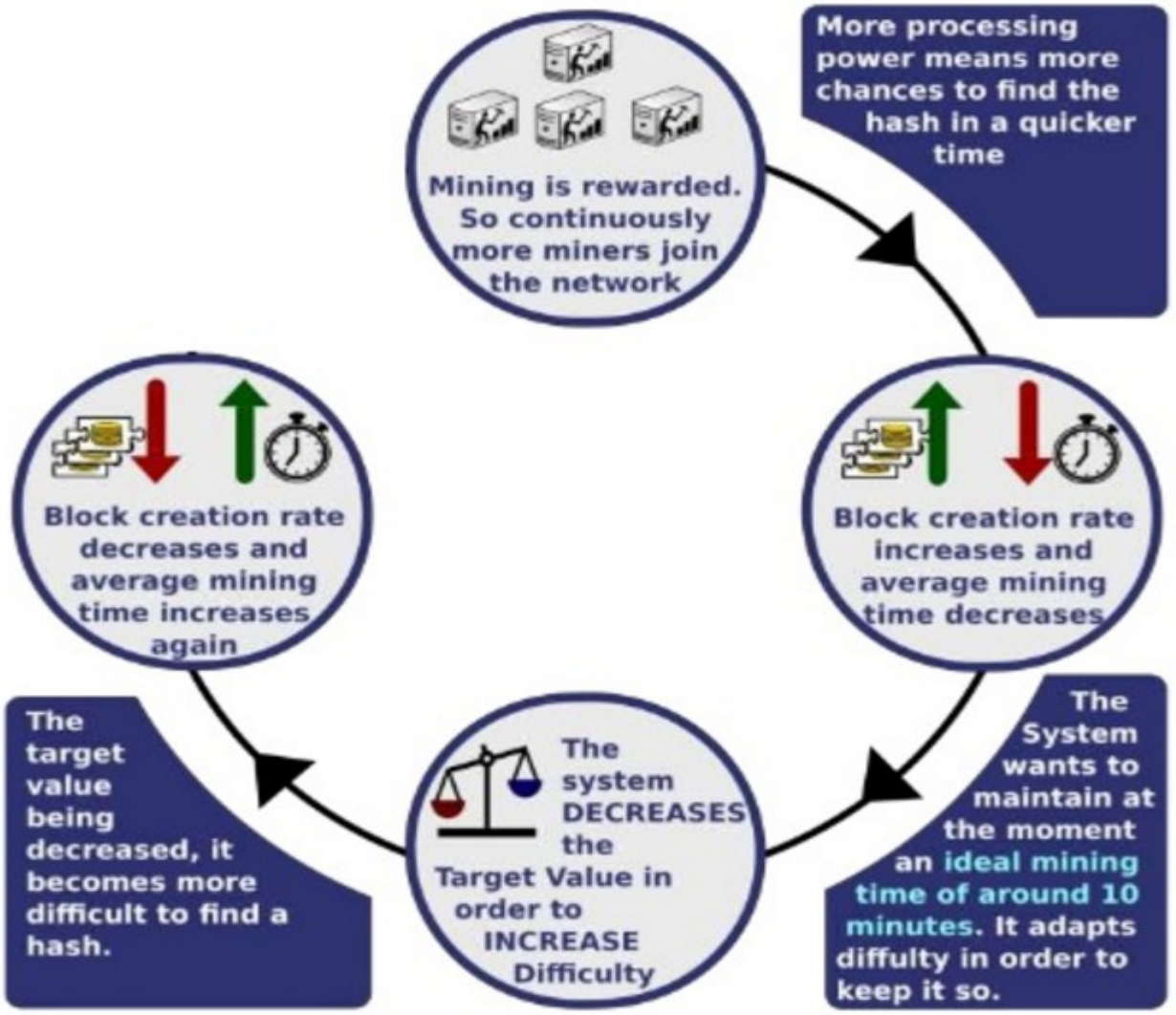
ADJUSTING THE POW DIFFICULTY



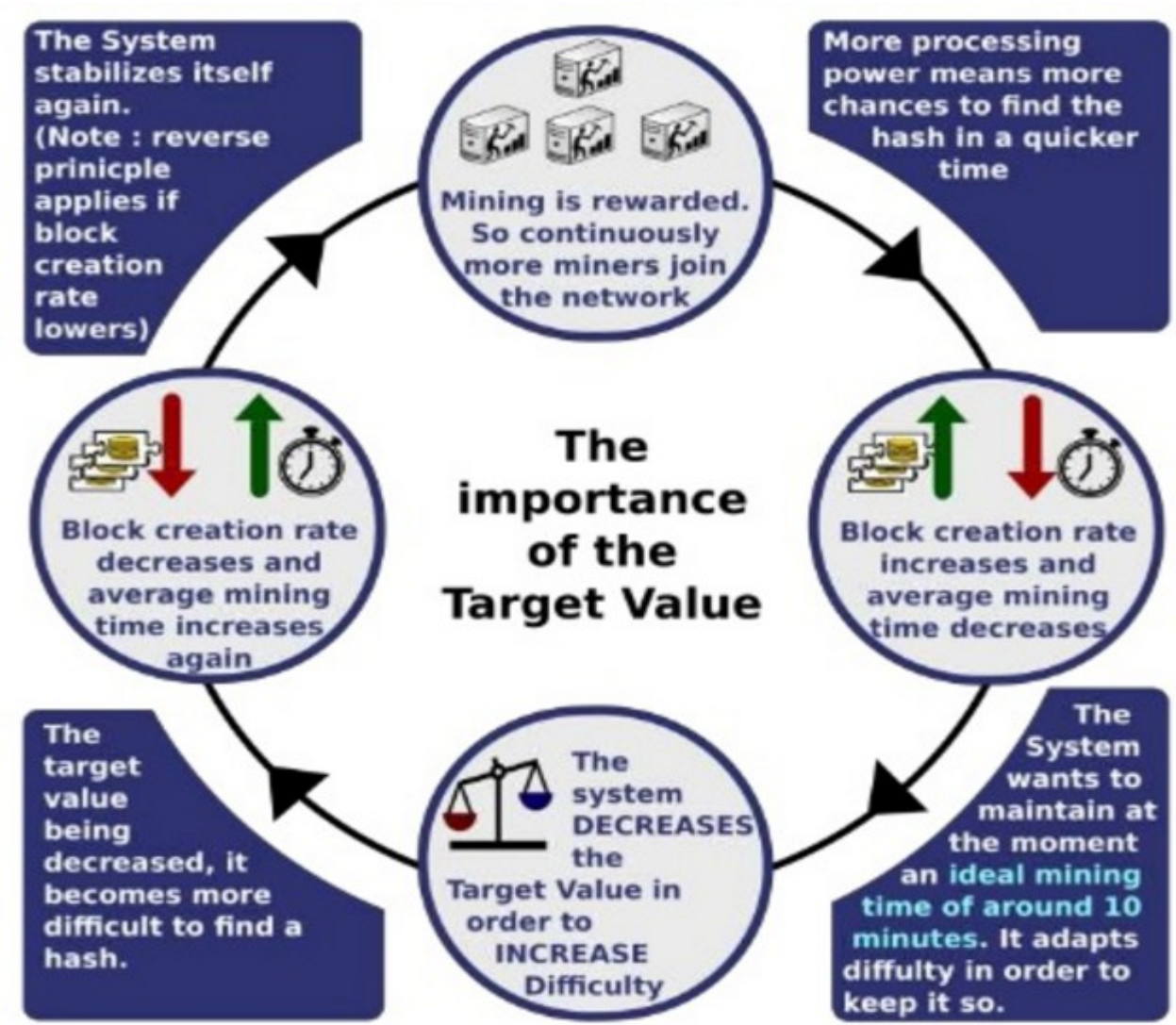
ADJUSTING THE POW DIFFICULTY



ADJUSTING THE POW DIFFICULTY



ADJUSTING THE POW DIFFICULTY



ADJUSTING THE POW DIFFICULTY

- to maintain the 10 minutes frequency: adjust block difficulty !
  - less leading zeros correspond to less difficulty
  - difficulty adjusted every 2,016 blocks or roughly every 2 weeks
- the adjustment takes into account the change in the computational power of the whole network since the last adjustment
  - compare the time-stamps of two blocks 2,016 positions apart
  - increase the number of starting zero bits in the target
- difficulty adjustment is part of the rules of Bitcoin protocol and it is coded into every Bitcoin client.

VERIFYING THE POW

- it takes, a lot of tries to find a nonce that makes the block hash fall below the target
- instead, verifying the Pow must be easy
- nonce is published as part of the block to allow PoW verification by the other nodes
- any node which receives the mined block may:
  - hash all the block contents together, and verify that the PoW has been carried on, i.e. the output is less than the target.
- another property allowing to get rid of centralization:
  - no centralized authority to verify that miners are doing their work properly

INCENTIVES: BLOCK REWARDS

- Why I should be a miner?
- Incentive miners to be honest: further prevents sybil attacks
- There are two kind of incentive:
- Block reward:
  - the miner is rewarded for confirming transactions by allowing it to mint new coins.
  - like a payment to the miner in exchange for the service of creating a block on the consensus chain.
  - the only way in which new bitcoins are allowed to be created in Bitcoin !

INCENTIVES: BLOCK REWARDS

- a miner solving the PoW can include in the block a special transaction:
  - reward transaction: the first transaction in a block:
    - a dummy input
    - sum of output values:

      fixed subsidy + sum of the fees of transactions confirmed in the block
    - reward dynamically varies in time
  - the recipients of the reward are typically one or more addresses of the miner

INCENTIVES: TRANSACTION FEES

- Transaction fee:
  - difference between the total value of the transaction inputs and the outputs
  - voluntarily inserted to obtain a certain degree of quality of service from the miners
- the miner that first puts that transaction into the blockchain
  - collects the fee for all the transactions in the block

ADJUSTING BLOCK REWARD

- 1 BTC = 10^8 Satoshi
- rule: half the mining reward after all 210.000 blocks (roughly every 4 years)

  Era 1: 50/1 50 BTC

  Era 2: 50/2 = 25 BTC

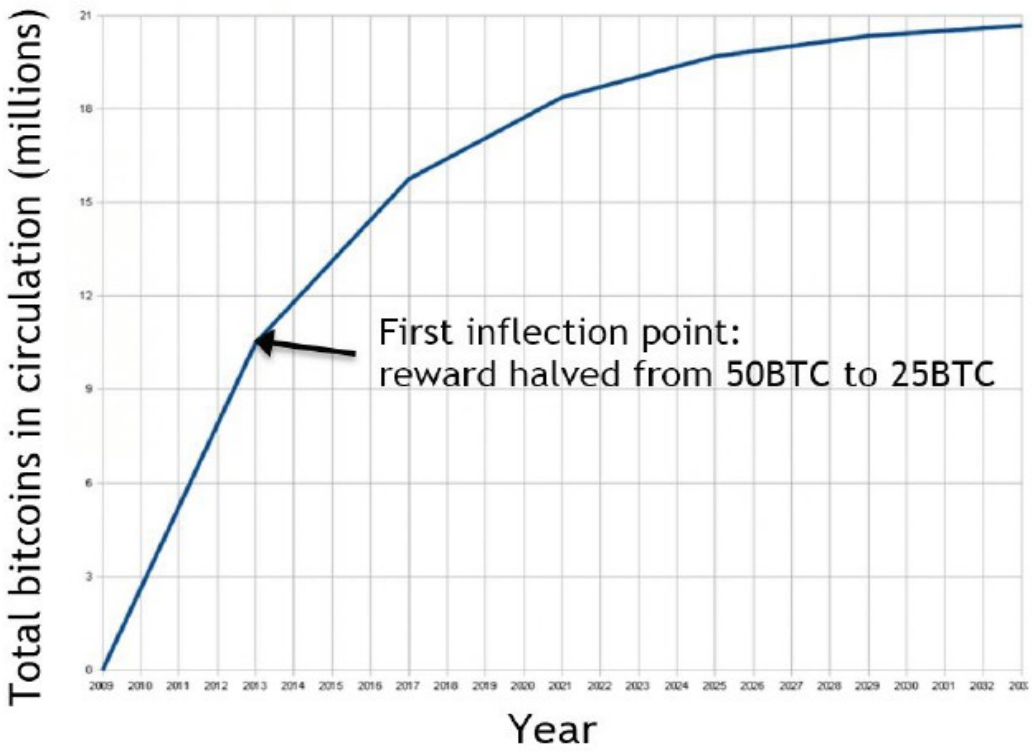
  Era 3: 25/2 = 12.5 BTC

  ...

  Era 33: 0.00000001 "Satoshis" (smallest unit)

- currently, the value of the block reward is fixed at 12.5 Bitcoins
  - last split in July 2016
  - we are in the third period of the "bitcoin era"
- based on the rate of block creation

ADJUSTING BLOCK REWARD



ADJUSTING BLOCK REWARD

- Historically, most of the compensation of miners has come from the block reward, and only a small percentage of their retribution from transaction fees
- it is expected that over time a larger percentage of the retribution will be due to transaction fees.
- new entrances try to capture block rewards, lowering the reward of all miners already in the network.
- so miners have to keep increasing their hashing rate in order to obtain the same reward.

NAKAMOTO CONSENSUS: CONCLUSIONS

Differences between the Bitcoin model of consensus and the others:

- randomized or probabilistic consensus (also called stabilizing consensus) i.e. the probability to disagree, decreases over time
- it uses a non-deterministic consensus protocol
- it uses a network of unknown size, i.e. supports anonymity
- users can join and leave the network (also called permission-less). This actually makes the problem harder
- it relies on some form of synchrony (bounded delay)
- it introduces the idea of incentives