# Introduction to blockchains

Abdelkader LAHMADI

# References

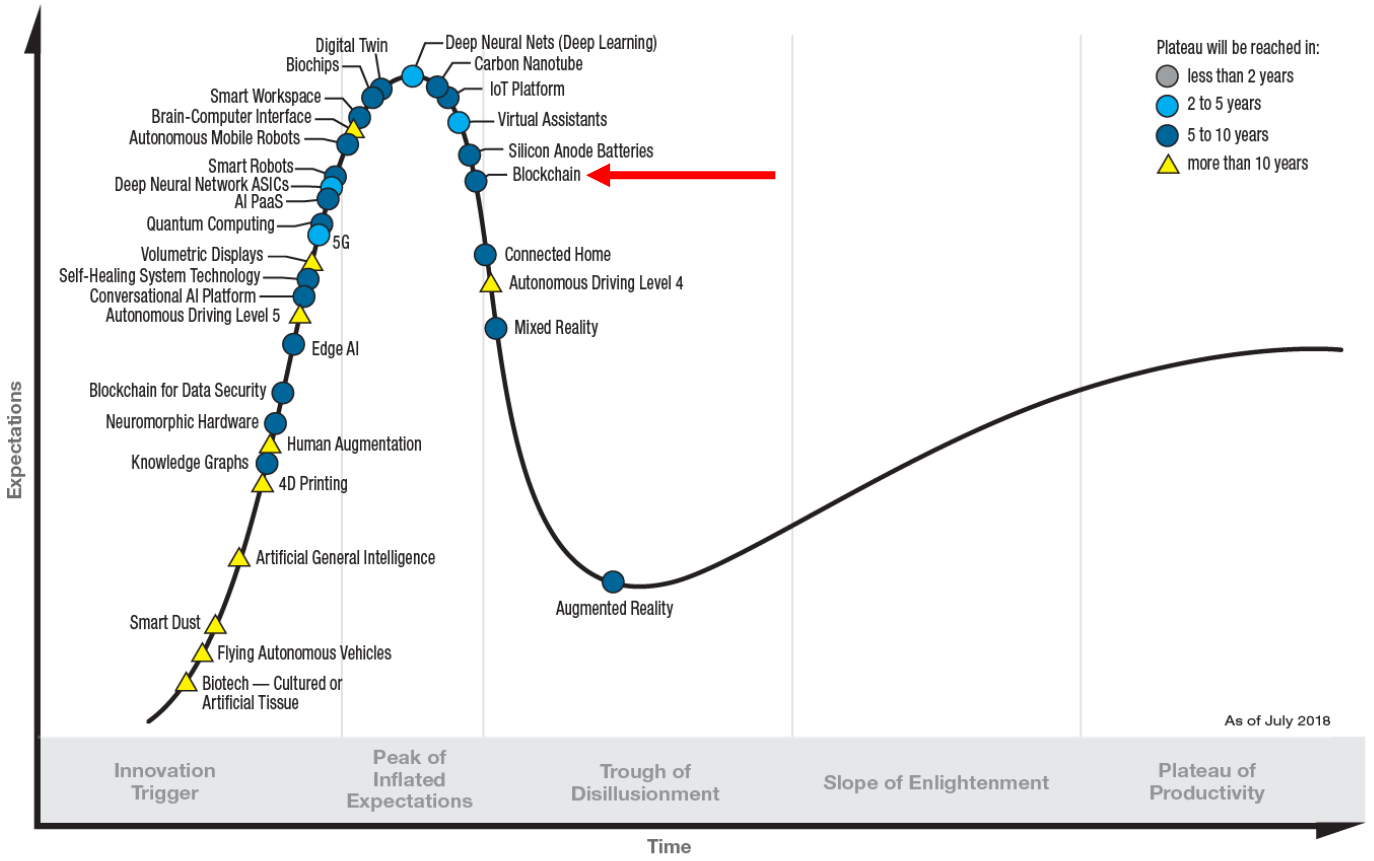

- Mastering Bitcoin 2nd Edition - Programming the Open Blockchain, by by Andreas M. Antonopoulos : https://github.com/bitcoinbook/bitcoinbook
- Mastering Ethereum, by Andreas M. Antonopoulos, Gavin Wood https://ethereumbook.info/: https://github.com/ethereumbook/ethereumbook
- Imran Bashir, Mastering Blockchain - Second Edition, March 2018

# Content

- Blockchain concepts
  - Structure and components
- Achieving consensus
  - Proof of Work and Proof of Stake


I THINK WE SHOULD BUILD A BLOCKCHAIN
UH-OH
DOES HE UNDERSTAND WHAT HE SAID OR IS IT SOMETHING HE SAW IN A TRADE MAGAZINE AD?
WHAT COLOR DO YOU WANT THAT BLOCKCHAIN?
I THINK MAUVE HAS THE MOST RAM.
S. Adams
E-mail: SCOTTADAMS@AOL.COM
© 1995 United Feature Syndicate, Inc.(NYC)

# Blockchain as emerging technology




gartner.com/SmarterWithGartner

Source: Gartner (August 2018)



Gartner

# The history

- 1980: Electronic cash (David Chaum)
  - Blind Signature and secret sharing
- 1992: Pricing functions to prevent e-mail spam (Monir Naor and Cynthia Dwork)
  - Hard functions that are required to be computed before access to a resource can be granted.
- 1997: Hashcash (Adam Back)
  - Proof of Work system to control e-mail spam: if legitimate users want to send e-mails then they are required to compute a hash as a proof that they have spent a reasonable amount of computing resources before sending the e-mail.
- 2008: Invention of bitcoin (cryptocurrency)

  *Satoshi Nakamoto, "Bitcoin: a Peer-to-Peer Electronic cash system"*
- 2009: Implementation of bitcoind
  - Solves the problem of distributed consensus in a trustless network
  - Public key cryptography with hashcash as PoW
  - chain of blocks => blockchain
- 2013: usage of blockchain in other areas than cryptocurrency

# The history: electronic cash

- 1980: e-cash protocols (David Chaum)
- Accountability: cash is spendable only one (double spending problem)
- Anonymity: protect users' privacy. Make it impossible to trace back spending to the individual who paid the money.
- Solutions:
  - Blind signature: sign a document without seeing it
  - Secret sharing: enables the detection of double spending when using the same e-cash token twice

# Blockchain defined

- Layman's definition: Blockchain is an ever-growing secure, shared record keeping system in which each user of the data holds a copy of the records, which can only be updated of all parties involved in a transaction agree to update.
- Technical definition: Blockchain at its core is a peer-to-peer distributed ledger that is cryptographically secure, append-only, immutable (extremely hard to change), and updateable only via consensus or agreement among peers.

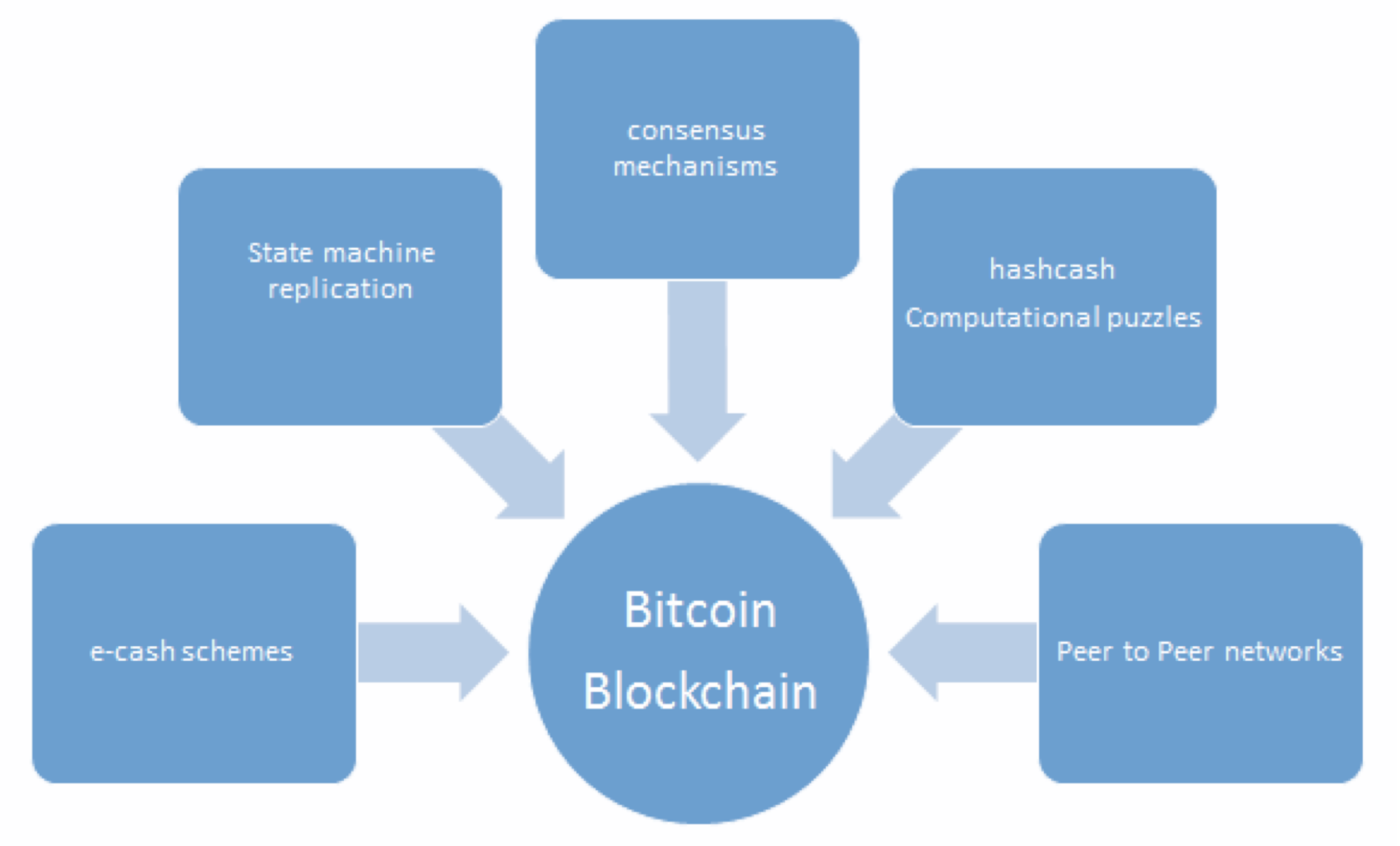
# Distributed consensus problem

- *Distributed system is a group of computers working together to achieve a goal*
  - *Lack of global clock*
  - Concurrency of processes
  - Faulty processes
  - Message passing
- *Consensus is a process of agreement between distrusting nodes on a final state of data.*
- It is easy to reach an agreement between two nodes (for example in client-server systems) but when multiple nodes are participating in a distributed system and they need to agree on a single value it becomes very difficult to achieve consensus

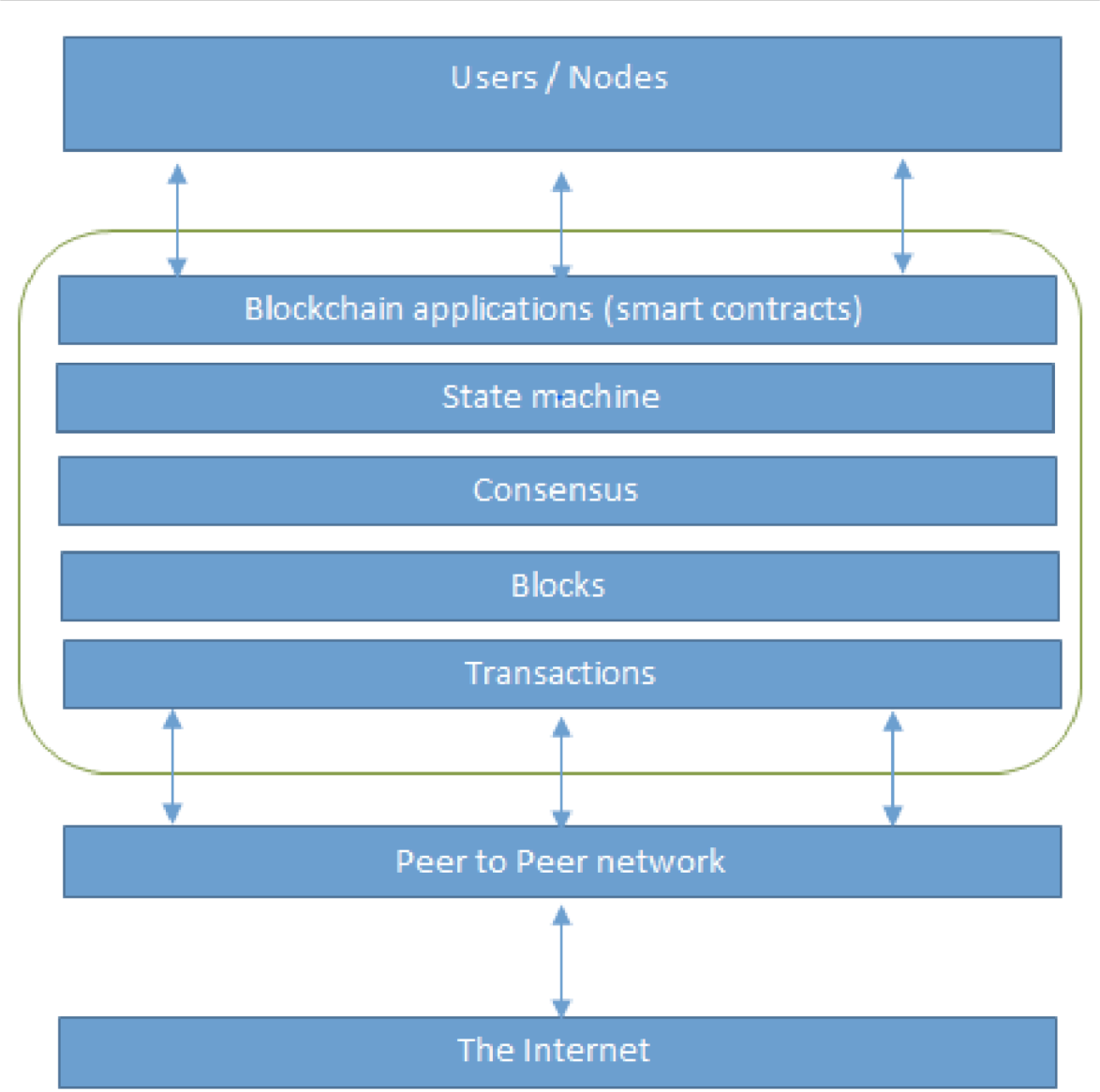
# Distributed and peer-to-peer systems

- Peer-to-peer: no central controller in the network, and all participants talk to each other directly.
  - No involved third parties in the transactions processing such as a bank
- Distributed ledger: a ledger is spread across the network mong all peers in the network, and each peer holds a copy
- Cryptographically-secure: cryptography is used to provide a ledge secure against tampering and misuse (non-repudiation, data integrity, and data origin authentication)
- Append-only: data can only be added to the blockchain in time-ordered sequential order => practically immutable
- Updateable via consensus: no central authority is in control of updating the ledger. Update is made after a consensus has been reached among all participating peers/nodes in the network.

# Blockchain: a mix of concepts

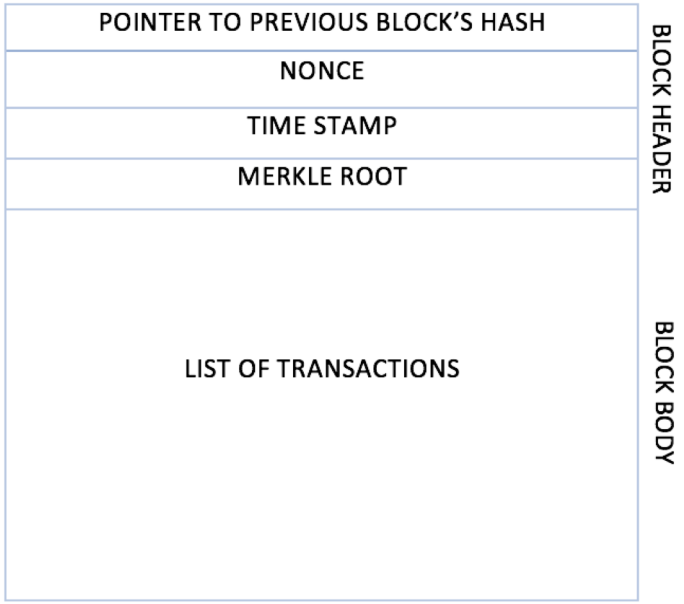


# Components of a Blockchain



# The block

- A block is a container data structure that aggregates transactions for inclusion in the blockchain
- Previous block's hash: A reference to the hash of the previous (parent) block in the chain
- None: A counter used for the proof-of-work algorithm
- Timestamp: The approximate creation time of this block (seconds from Unix Epoch)
- Merle root: A hash of the root of the Merkle-Tree of this block's transactions
- Transaction: represents a transfer of value from one address to another



# The genesis

- Generated by Satoshi Nakamoto: January 3th, 2009

https://blockexplorer.com/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

## Block #0

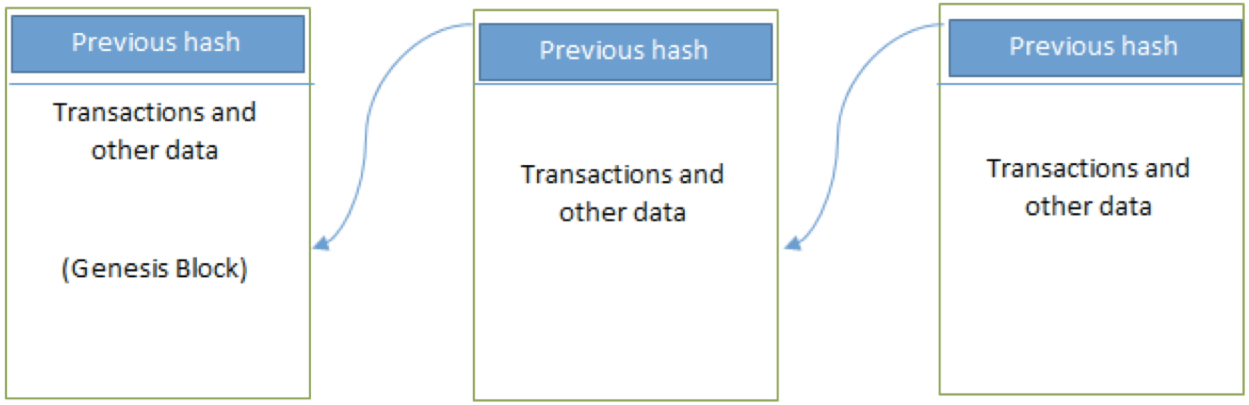
**BlockHash** 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

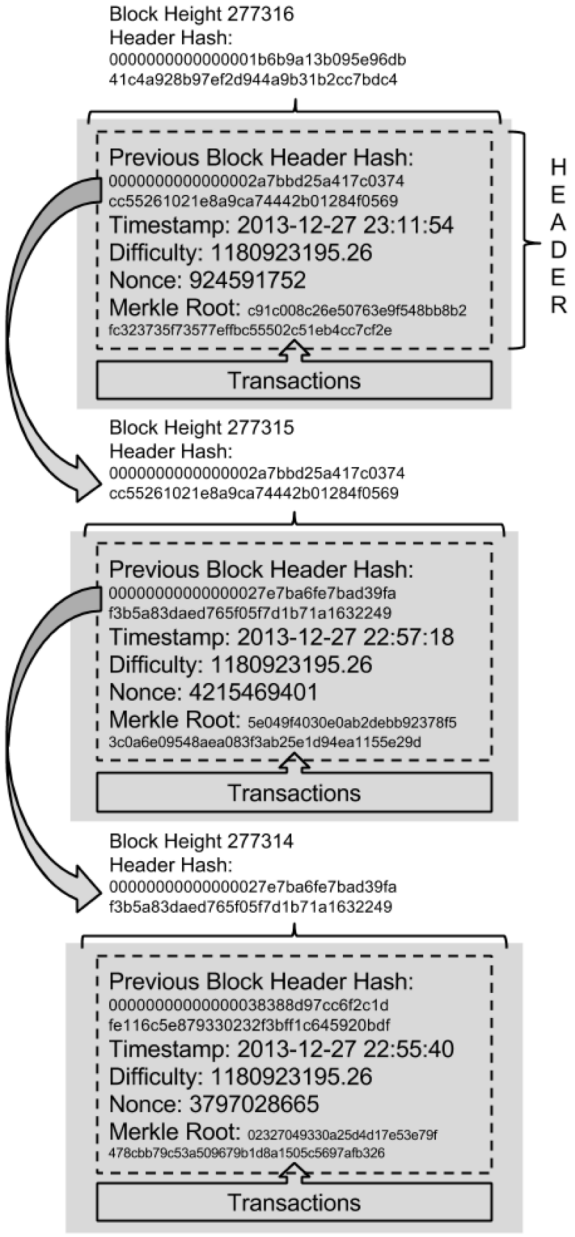
## Summary

| | | | |
|---|---|---|---|
| **Number Of Transactions** | 1 | **Difficulty** | 1 |
| **Height** | 0 (Mainchain) | **Bits** | 1d00ffff |
| **Block Reward** | 50 BTC | **Size (bytes)** | 285 |
| **Timestamp** | Jan 3, 2009 7:15:05 PM | **Version** | 1 |
| **Mined by** | | **Nonce** | 2083236893 |
| **Merkle Root** | 4a5e1e4baab89f3a32518a88c31bc... | **Next Block** | 1 |

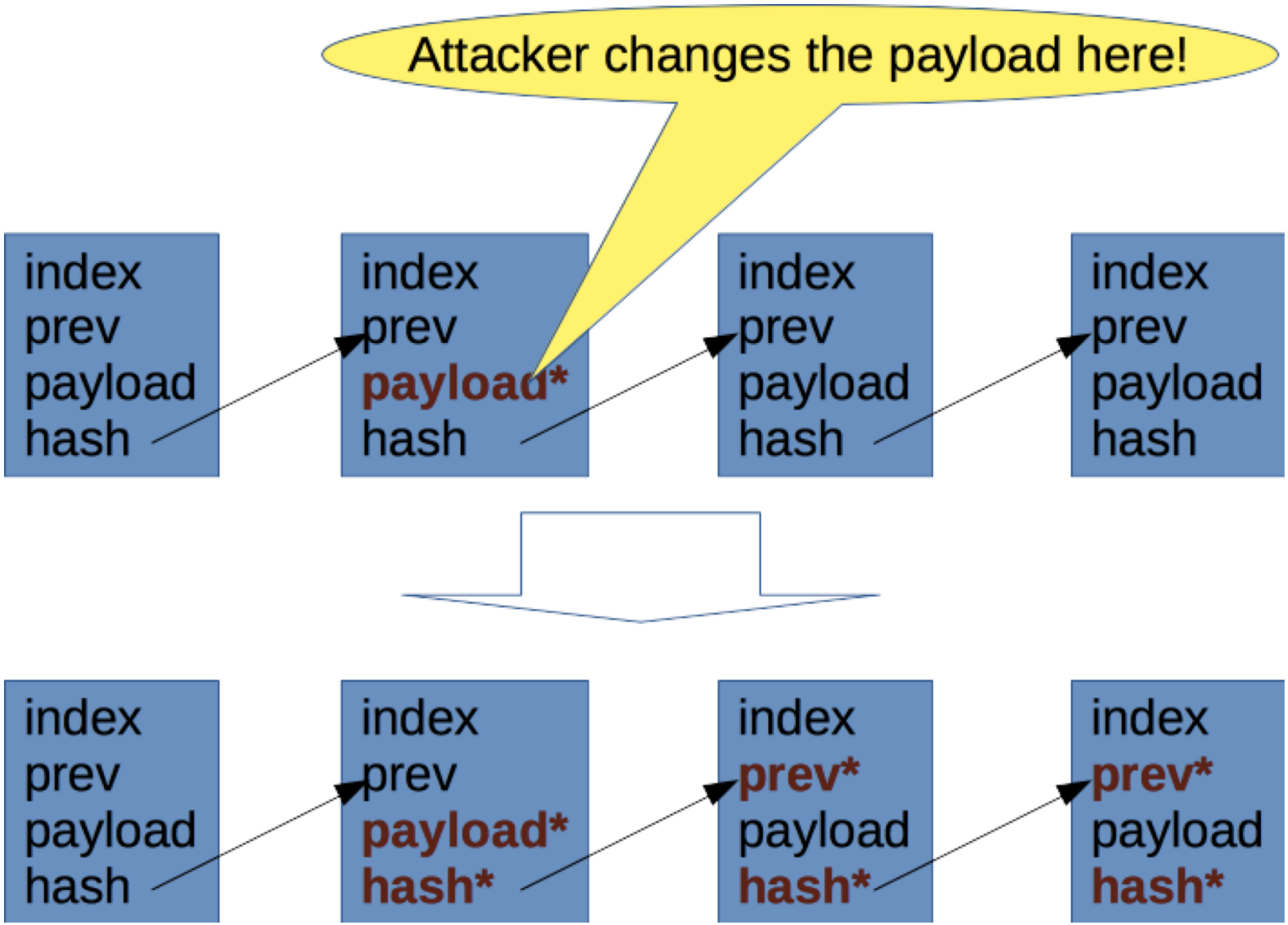
# The generic chain

- A blockchain is a sequence of hash-chained records
- A block chain containing three blocks, each containing the hash of the previous block, and each containing a sequence of transactions and a nonce.
- Each block in the chain *commits* to all previous blocks and transactions



# What happen if we change a block ?

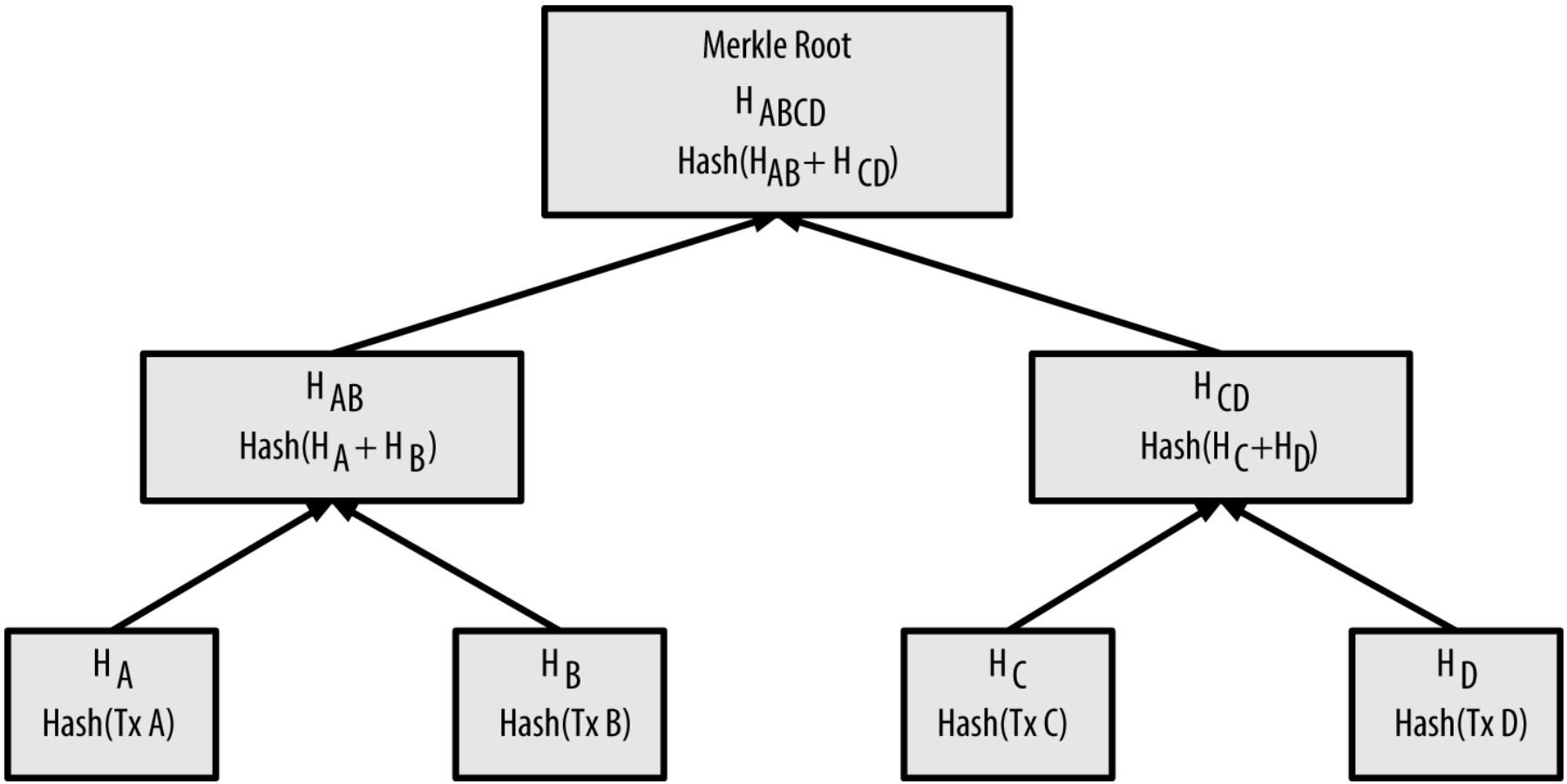


# Merkle trees

- A Merkle Tree, also known as a Binary Hash Tree is a data structure used for efficiently summarizing and verifying the integrity of large sets of data.
- Merkle Trees are binary trees containing cryptographic hashes.
- Verify if a transaction is included in a block: at most 2*log(N) operations
- The merkle tree is constructed bottom-up

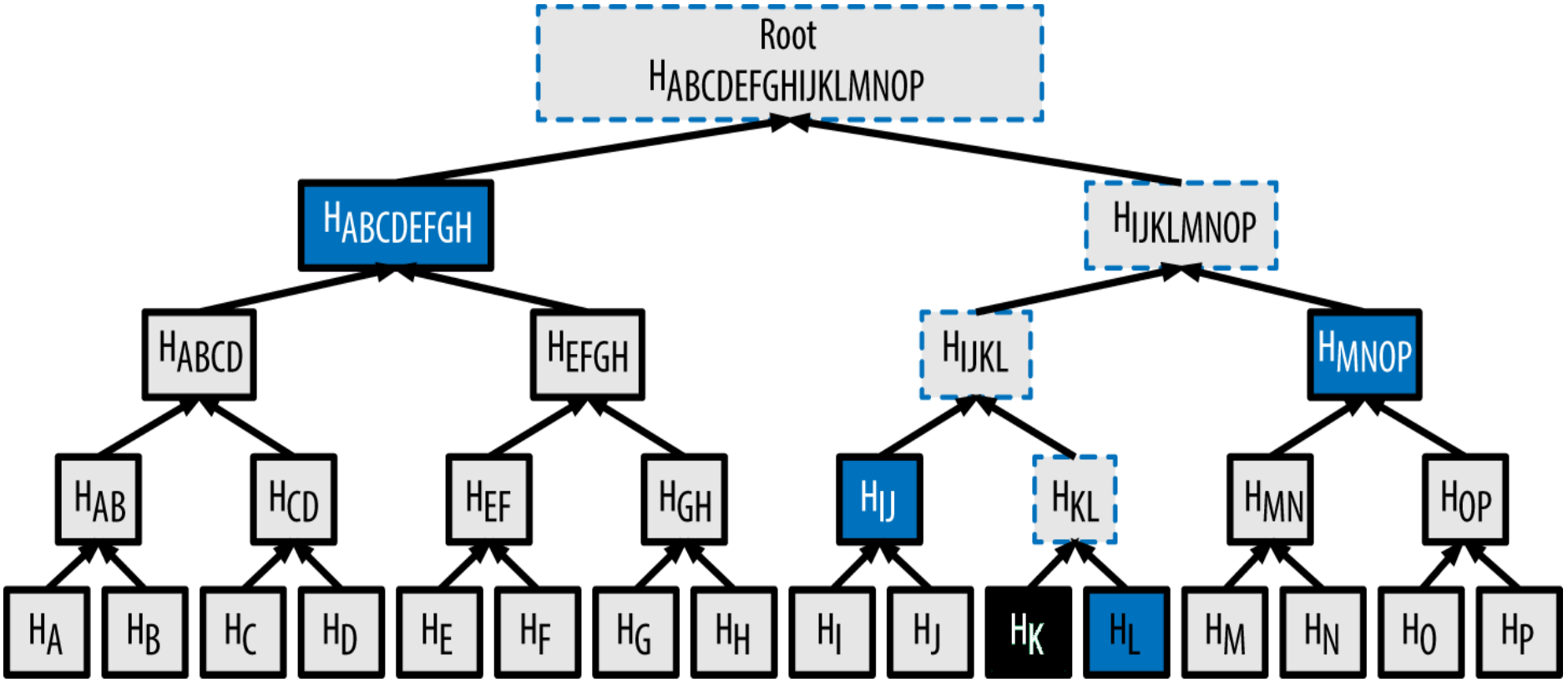
# Merkle trees: building

- $H_A$ = SHA256(SHA256(Transaction A))
- $H_{AB}$ = SHA256(SHA256($H_A$ + $H_B$))



# Merkle trees: inclusion verification

- verify that transaction K is in the block without downloading all transactions
- Only requires the authentication path: $H_L$, $H_{IJ}$, $H_{MNOP}$, and $H_{ABCDEFGH}$



# Merkle tree: efficiency

While the block size increases rapidly, from 4 KB with 16 transactions to a block size of 16 MB to fit 65,535 transactions, the merkle path required to prove the inclusion of a transaction increases much more slowly, from 128 bytes to only 512 bytes.

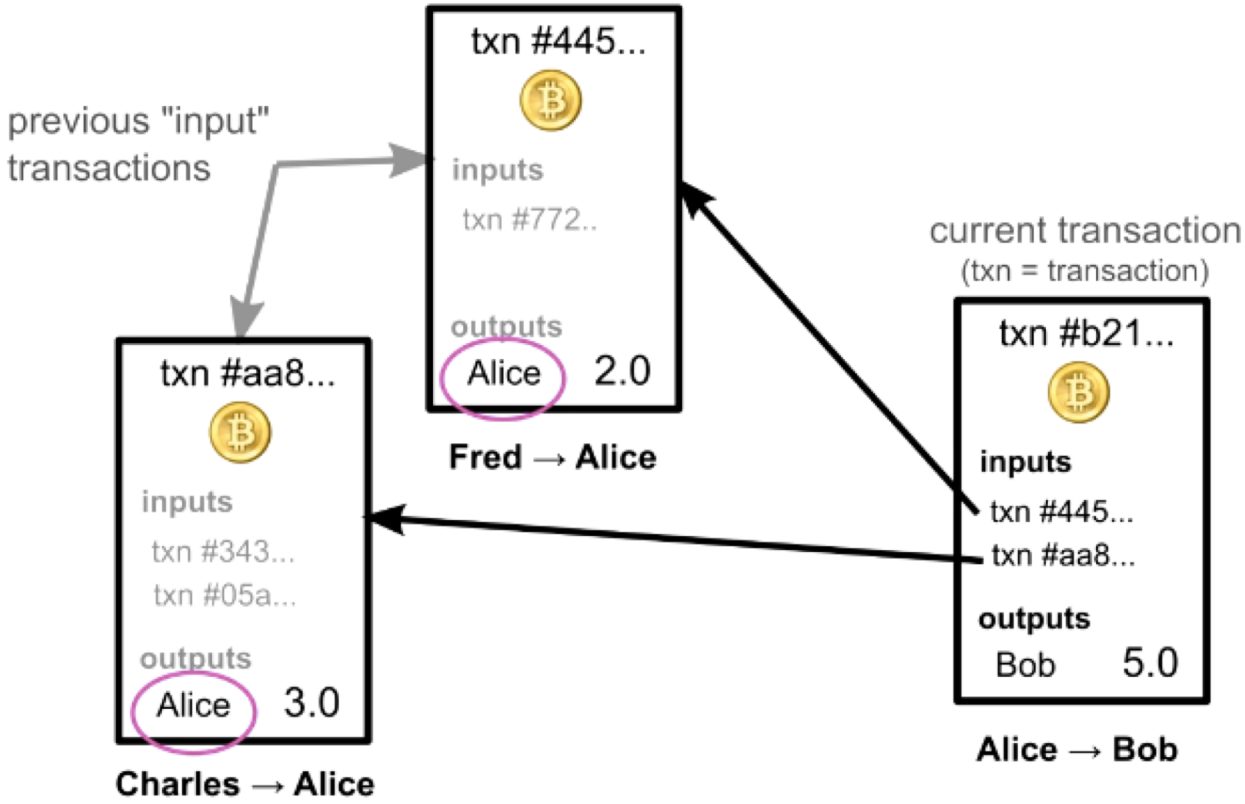
| Number of transactions | Approx. size of block | Path size (hashes) | Path size (bytes) |
|---|---|---|---|
| 16 transactions | 4 kilobytes | 4 hashes | 128 bytes |
| 512 transactions | 128 kilobytes | 9 hashes | 288 bytes |
| 2048 transactions | 512 kilobytes | 11 hashes | 352 bytes |
| 65,535 transactions | 16 megabytes | 16 hashes | 512 bytes |

# How blockchain works ?

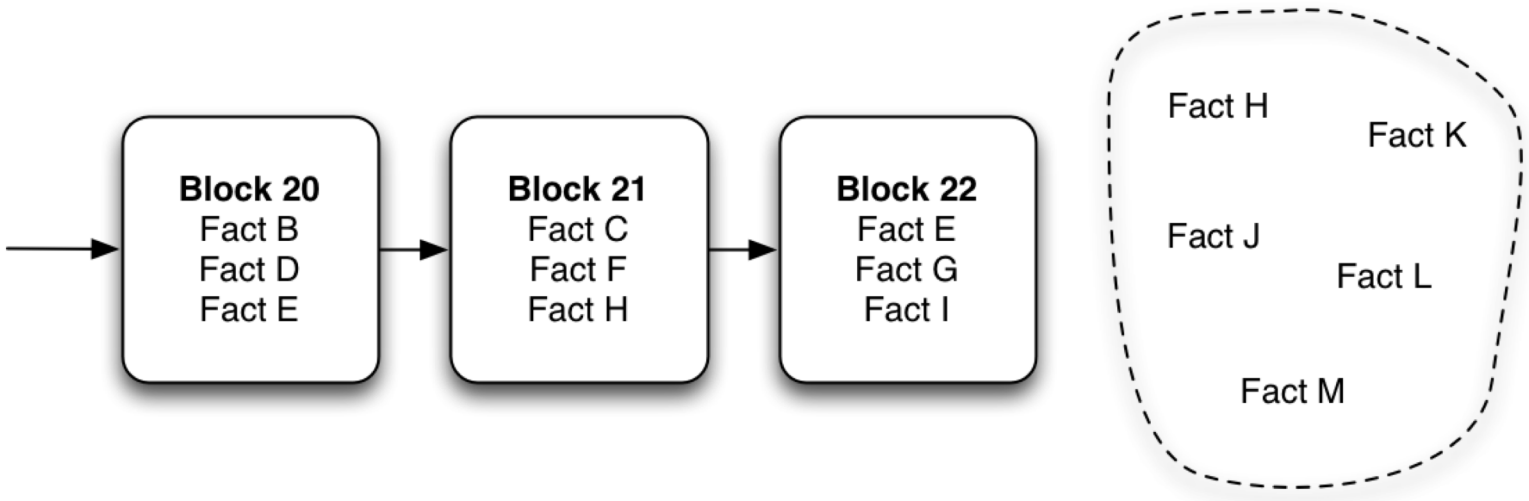
1. A node starts a transaction by first creating and then digitally signing it with its private key. A transaction can represent various actions in a blockchain. Most commonly this is a data structure that represents transfer of value between users on the blockchain network. Transaction data structure usually consists of some logic of transfer of value, relevant rules, source and destination addresses, and other validation information. This will be covered in more detail in specific chapters on Bitcoin and Ethereum later in the book.
2. A transaction is propagated (flooded) by using a flooding protocol, called Gossip protocol, to peers that validate the transaction based on preset criteria. Usually, more than one node are required to verify the transaction.
3. Once the transaction is validated, it is included in a block, which is then propagated onto the network. At this point, the transaction is considered confirmed.
4. The newly-created block now becomes part of the ledger, and the next block links itself cryptographically back to this block. This link is a hash pointer. At this stage, the transaction gets its second confirmation and the block gets its first confirmation.
5. Transactions are then reconfirmed every time a new block is created. Usually, six confirmations in the Bitcoin network are required to consider the transaction final.

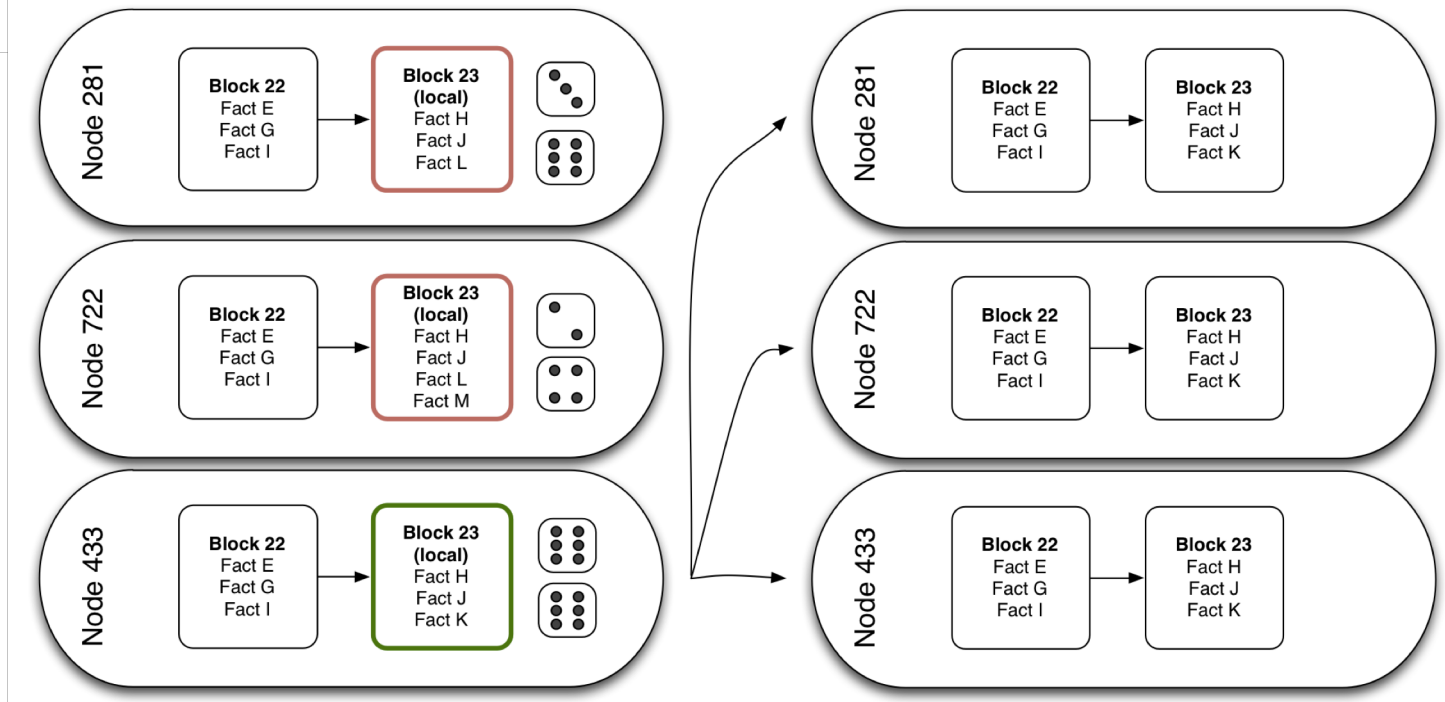
# A transaction

- A transaction in Blockchain certainly can include, but is not limited to valuta.
- A big difference with current banking : blockchain will not keep track of the balance.
- Keeps track of all the transactions and whether or not they have already been 'spent'.
- The balance can be recalculated, but it is in itself not stored.



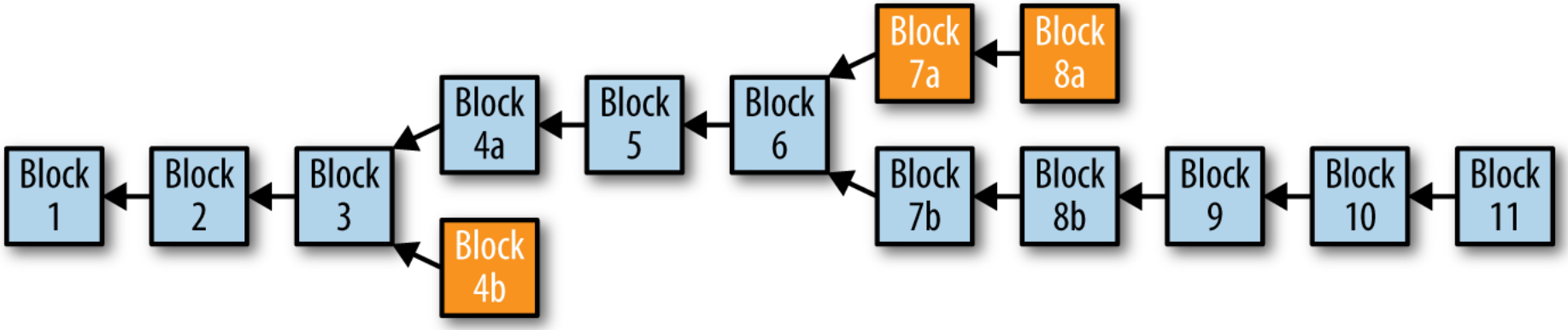
# Pending transactions



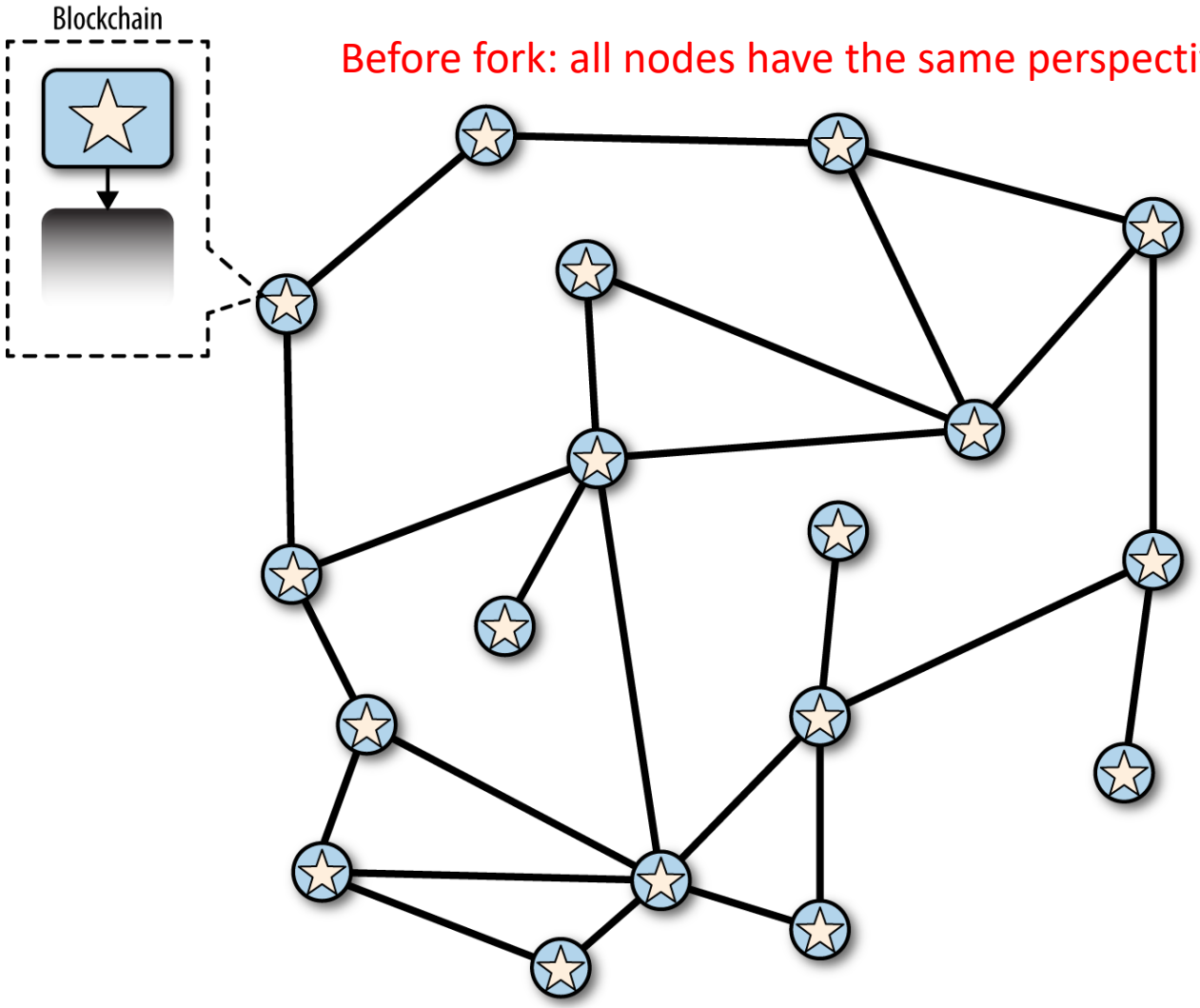
# Blockchain forks

- Blockchain is a decentralized data structure, so different copies of it are not always consistent
- Blocks might arrive at different nodes at different times, causing the nodes to have different perspectives of the blockchain
- Each node always selects and attempts to extend the chain of blocks that represents the most Proof-of-Work, also known as the longest chain or greatest cumulative work chain

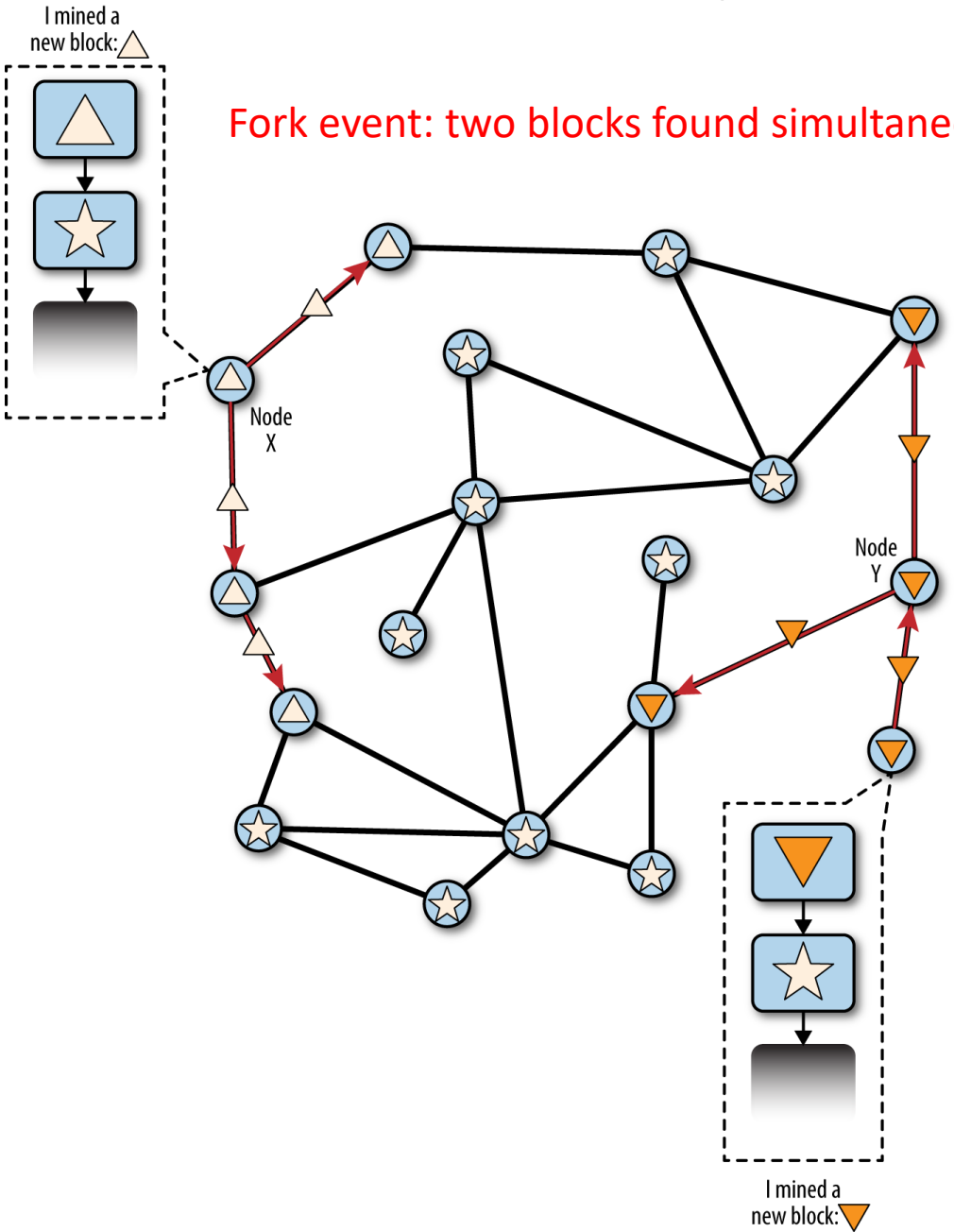


# Blockchain forks: example

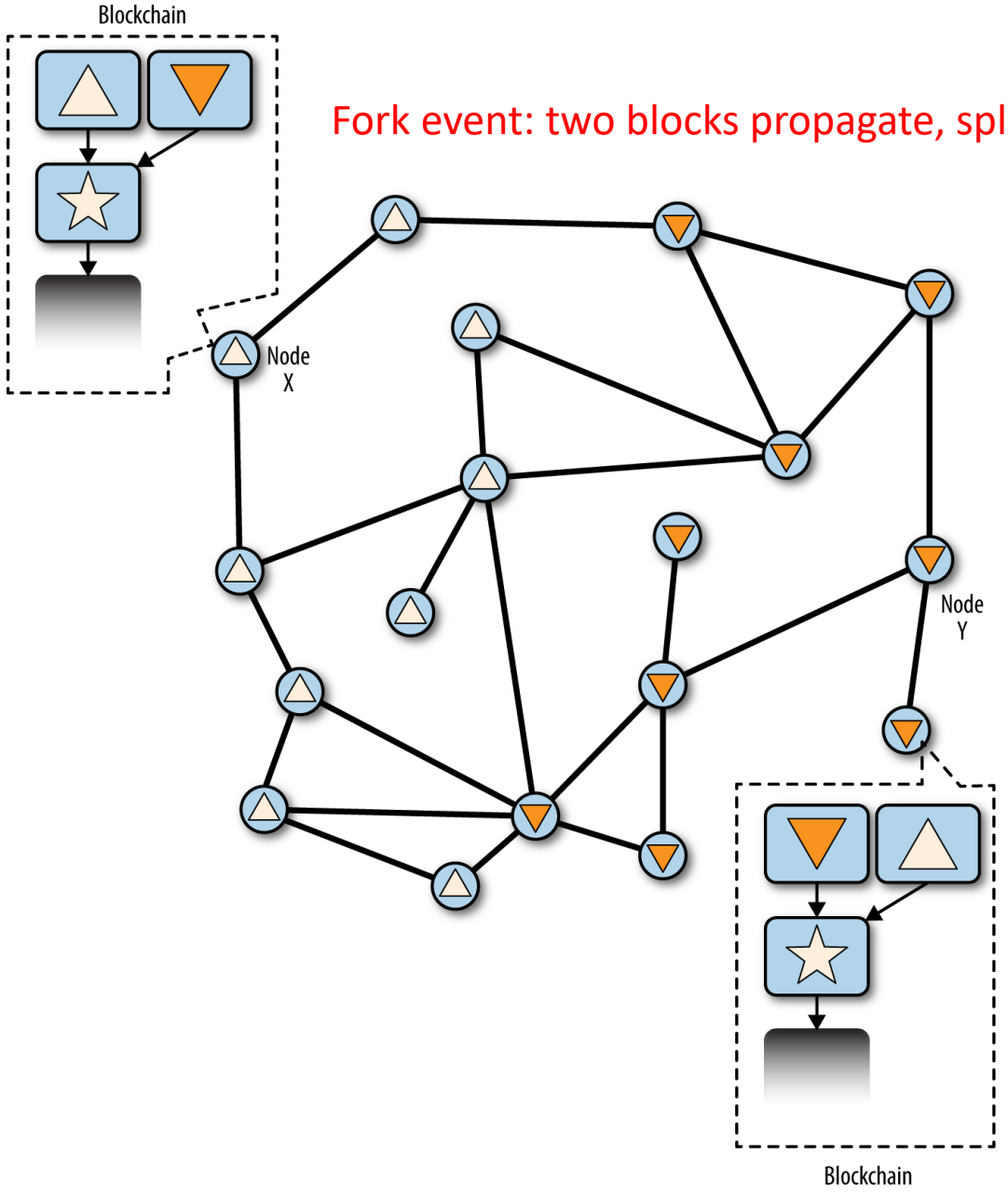
Each node has its own perspective of the global blockchain. As each node receives blocks from its neighbors, it updates its own copy of the blockchain, selecting the greatest-cumulative-work chain



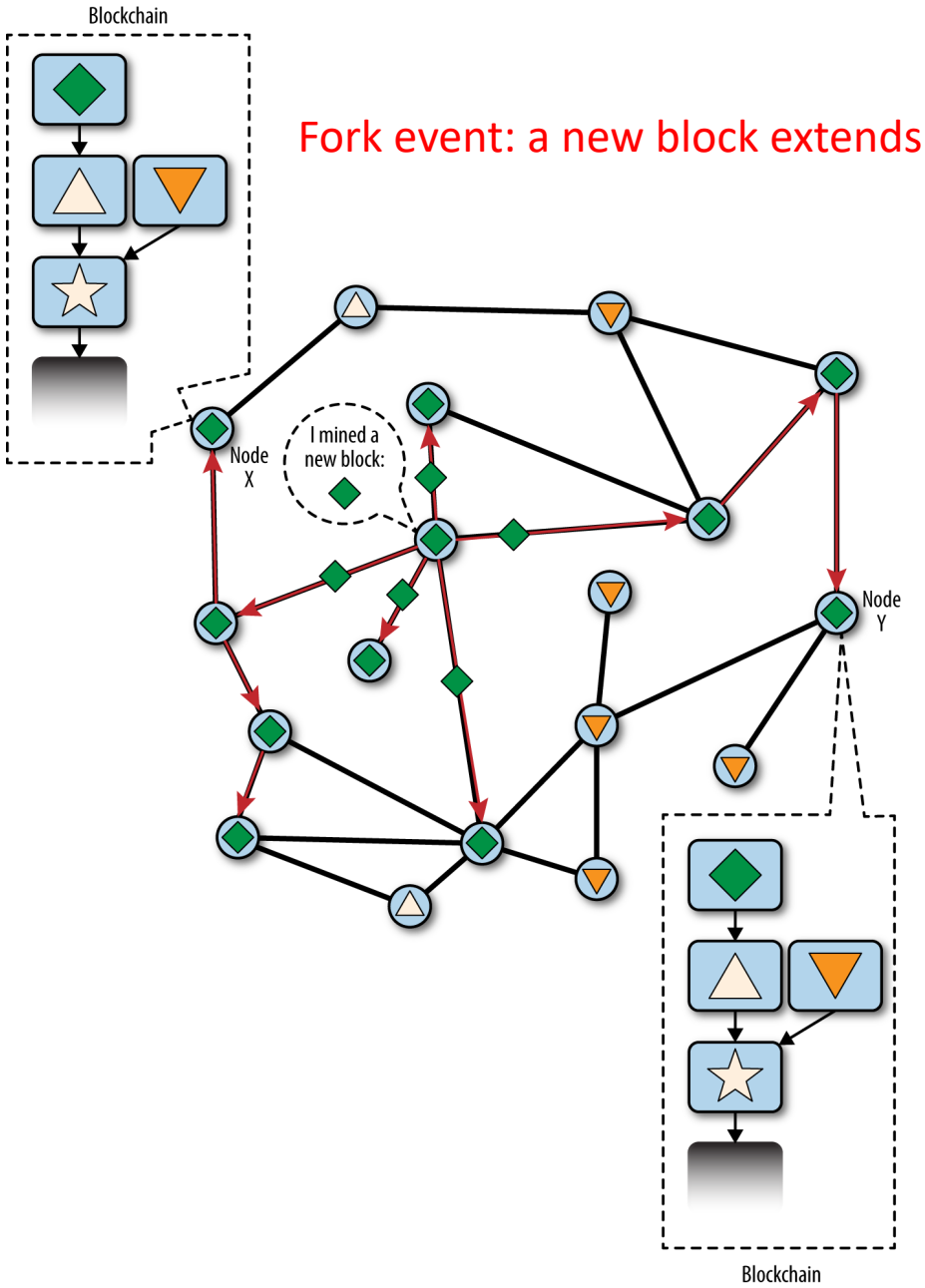
# Blockchain forks: example



# Blockchain forks: example

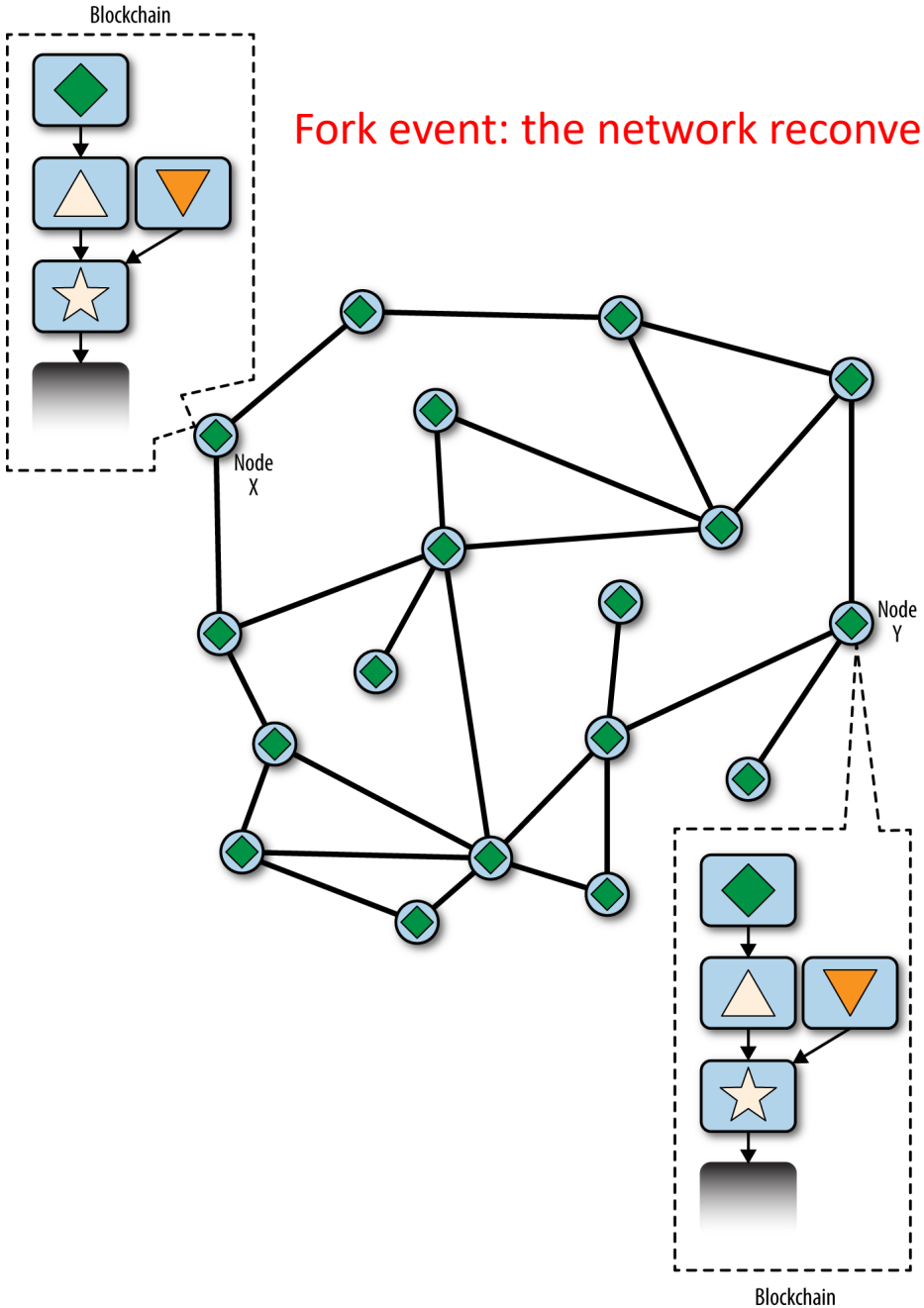


# Blockchain forks: example




Fork event: a new block extends one fork, reconverging the network

# Blockchain forks: example




Fork event: the network reconverges on a new longest chain

# Blockchain: tradeoff

- Bitcoin's block interval of 10 minutes is a design compromise between fast confirmation times (settlement of transactions) and the probability of a fork
- A faster block time would make transactions clear faster but lead to more frequent blockchain forks
- A slower block time would decrease the number of forks but make settlement slower.

*Let's say there are two blockchains: chain A and chain B, that find a new block every 10 minutes, 15 seconds resp.*
*A fork occurs when two blocks are found within lets say a second. Furthermore the assumption is made that the distribution of the time between two blocks found is exponential. Some simple math will show that chain B has many more forks:*

*Chain A*

$X$ = *the number of blocks found in one second*

$X \sim Poisson\left(\frac{1}{10 \times 60}\right)$

$$\begin{aligned}\mathbb{P}(X > 1) &= 1 - \mathbb{P}(X = 0 \text{ or } X = 1) \\ &= 1 - \left(\mathbb{P}(X = 0) + \mathbb{P}(X = 1)\right) \\ &\approx 1 - \left(0,999998613\right) \\ &\approx 0,00000138735\end{aligned}$$

*Chain B*

$Y$ = *the number of blocks found in one second*

$Y \sim Poisson\left(\frac{1}{15}\right)$

$$\begin{aligned}\mathbb{P}(Y > 1) &= 1 - \mathbb{P}(Y = 0 \text{ or } X = 1) \\ &= 1 - \left(\mathbb{P}(Y = 0) + \mathbb{P}(Y = 1)\right) \\ &\approx 1 - \left(0,997874117\right) \\ &\approx 0,002125883\end{aligned}$$

*For both chains the probability of a fork is very small. However chain B has rougly 1532 times more probability of a fork than chain A.*

# Types of blockchain

- Public blockchains: As the name suggests, these blockchains are open to the public and anyone can participate as a node in the decision-making process. Users may or may not be rewarded for their participation. These ledgers are not owned by anyone and are publicly open for anyone to participate in. All users of the permission-less ledger maintain a copy of the ledger on their local nodes and use a distributed consensus mechanism in order to reach a decision about the eventual state of the ledger. These blockchains are also known as permission-less ledgers

# Types of blockchain

- Private blockchains: Private blockchains as the name implies are private and are open only to a consortium or group of individuals or organizations that has decided to share the ledger among themselves.
- Semi-private blockchains: Here part of the blockchain is private and part of it is public. The private part is controlled by a group of individuals whereas the public part is open for participation by anyone. It is a concept, and no real world POC have yet been developed.
- Sidechains: More precisely known as pegged sidechains, this is a concept whereby coins can be moved from one blockchain to another and moved back. Common uses include the creation of new altcoins (alternative cryptocurrencies) whereby coins are burnt as a proof of adequate stake. There are two types of sidechain. The example provided above for burning coins is applicable to a one-way pegged sidechain. The second type is called a two-way pegged sidechain, which allows the movement of coins from the main chain to the sidechain and back to the main chain when required.

# Types of blockchain

- Permissioned ledger: A permissioned ledger is a blockchain whereby the participants of the network are known and already trusted. Permissioned ledgers do not need to use a distributed consensus mechanism, instead an agreement protocol can be used to maintain a shared version of truth about the state of the records on the blockchain. There is also no requirement for a permissioned blockchain to be private as it can be a public blockchain but with regulated access control.
- Tokenized blockchains: These blockchains are standard blockchains that generate cryptocurrency as a result of a consensus process via mining or via initial distribution.

# Blockchain features

- Distributed consensus: present a single version of the truth, which is agreed upon by all parties without the requirement of a central authority
- Transaction verification: any transactions posted from the nodes on the blockchain are verified based on a predetermined set of rules. Only valid transactions are selected for inclusion in a block.
- Platform for smart contracts: a platform on which programs can run to execute a business logic on behalf of users. Not available on all blockchains

# Blockchain features

- Transferring value between peers: blockchain enables the transfer of value between its users via tokens.
- Generation of cryptocurrency: optional feature depending on the type of blockchain in use. A blockchain can create cryptocurrency as an incentive to its miners who validate the transaction and spend resources to secure the blockchain.
- Immutability: once records are added to the blockchain, they are immutable.
- Uniqueness: every transaction is unique and has not already been spent (double spend problem).

# Blockchain applications

- Loans: Quite similar to bitmortage but slightly broader. According to Microsoft blockchain can be used for all sorts of loans.
- Obligations: As an obligation can be viewed as a special type of loan.
- Voting system: One of the things smart contracts on blockchain can be used for. Ethereum already built-in smart contracts which can be easily used for this purpose.
- Supply chains: If all parties involved in supply chains can join in one blockchain, this can ease the communication. Everything will be visible for all parties at all times, making the whole process run smoother.
- Identity management and verification: 'Ownership of data will shift back from the central parties to the individual’
- Energy reserve supply market: Creating a supply and demand market for energy where smaller parties can join just as easily as big ones.

# Demo: How blockchain works ?

- http://blockchain.mit.edu/how-blockchain-works
- https://anders.com/blockchain/hash.html
- http://learnmeabitcoin.com/browser/node/
- https://www.youtube.com/channel/UCj9MFr-7a02d_qe4xVnZ1sA/videos

# Achieving consensus

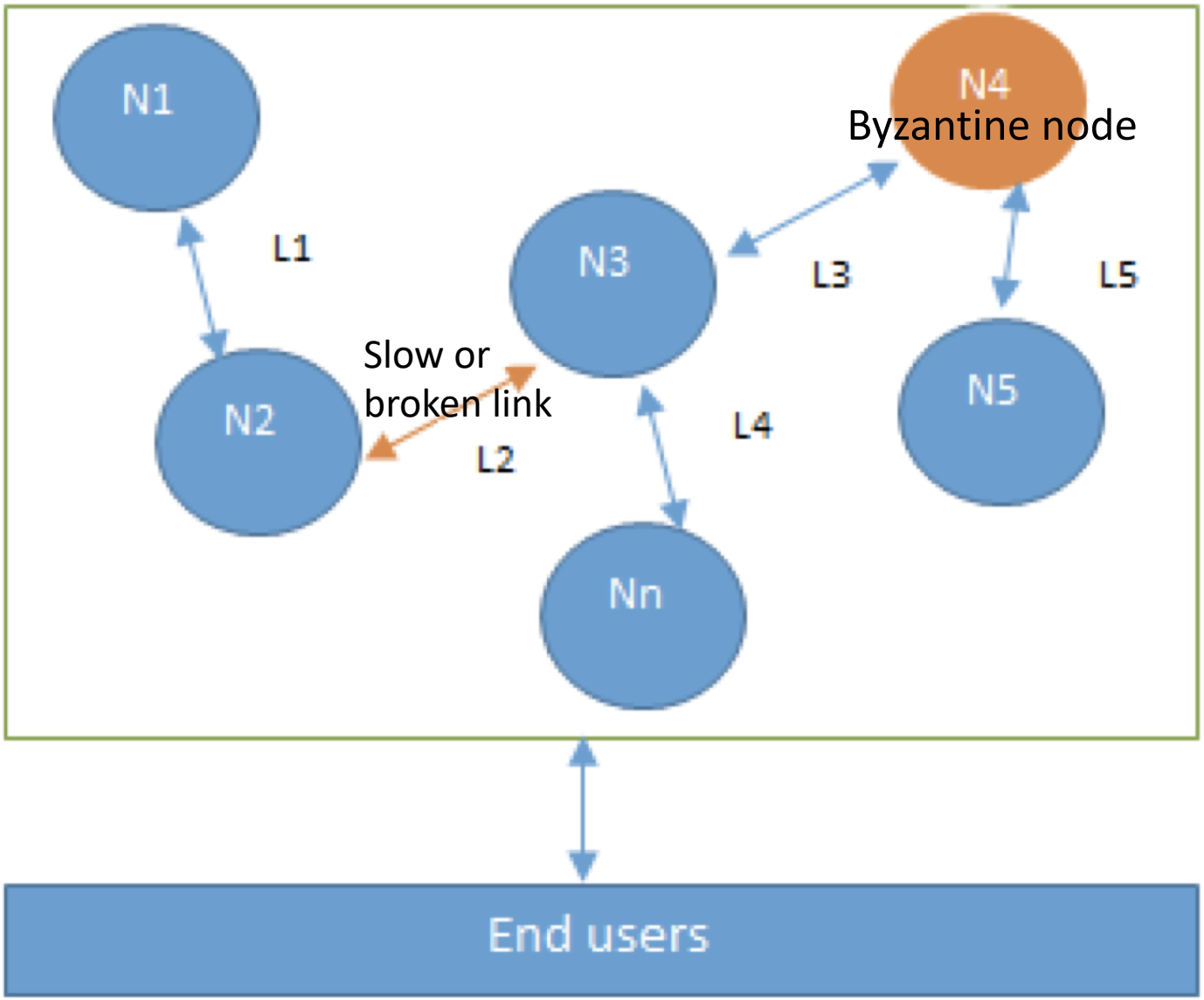
- Blockchain is a distributed ledger which can be centralized or decentralized.
- Blockchain is intended to be decentralized distributed system
- All nodes are capable of sending and receiving messages to and from each other.
- Nodes can be honest, faulty, or malicious
- A node with an irrational behaviour is a Byzantine node: Byzantine Generals problem
- How can everyone in the network agree on a single universal “truth” about who owns what, without having to trust anyone?

# The Byzantine Generals problem

*In 1982 a thought experiment was proposed by Lamport et al. whereby a group of army generals who are leading different parts of the Byzantine army are planning to attack or retreat from a city. The only way of communication between them is a messenger and they need to agree to attack at the same time in order to win. The issue is that one or more generals can be traitors and can communicate a misleading message. Therefore there is a need to find a viable mechanism that allows agreement between generals even in the presence of treacherous generals so that the attack can still take place at the same time. As an analogy with distributed systems, generals can be considered as nodes, traitors can be considered Byzantine (malicious) nodes, and the messenger can be thought of as a channel of communication between the generals.*

- The problem was solved in 1999 by Castro and Liskov who presented the Practical Byzantine Fault Tolerance (PBFT) algorithm, where consensus is reached after a certain number of messages are received containing the same signed content.
- Later on in 2009, the first practical implementation was made with the invention of bitcoin where the Proof of Work (PoW) algorithm was developed as a mechanism to achieve consensus.

# Example of a distributed system



# Distributed Consensus

Consensus is a process of agreement between distrusting nodes on a final state of data.

- Agreement: All honest nodes decide on the same value.
- Termination: All honest nodes terminate execution of the consensus process and eventually reach a decision.
- Validity: The value agreed upon by all honest nodes must be the same as the initial value proposed by at least one honest node.
- Fault tolerant: The consensus algorithm should be able to run in the presence of faulty or malicious nodes (Byzantine nodes).
- Integrity: This is a requirement where by no node makes the decision more than once. The nodes make decisions only once in a single consensus cycle.

# Consensus mechanisms

- Byzantine fault tolerance (BFT)-based: With no compute intensive operations such as partial hash inversion, this method relies on a simple scheme of nodes that are publishing signed messages. Eventually, when a certain number of messages are received, then an agreement is reached.
- Leader-based consensus mechanisms: This type of mechanism requires nodes to compete for the leader-election lottery and the node that wins it proposes a final value. PoW used in bitcoin falls into this category

# Consensus in blockchain

- Proof-based, leader-based, or the Nakamoto consensus whereby a leader is elected and proposes a final value: fully decentralized or permissionless type
- Byzantine fault tolerance-based, which is a more traditional approach based on rounds of votes: consortium or permissioned type

***Distributed consensus protocol.*** There are *n* nodes that each have an input value. Some of these nodes are faulty or malicious. A distributed consensus protocol has the following two properties:

- It must terminate with all honest nodes in agreement on the value
- The value must have been generated by an honest node

signed by Alice

Pay to $pk_{Bob}$ : H( )

# Distributed Consensus: a simplified version

**Bitcoin consensus algorithm (simplified)**

*This algorithm is simplified in that it assumes the ability to select a random node in a manner that is not vulnerable to Sybil attacks.*

1. New transactions are broadcast to all nodes
2. Each node collects new transactions into a block
3. In each round a random node gets to broadcast its block
4. Other nodes accept the block only if all transactions in it are valid (unspent, valid signatures)
5. Nodes express their acceptance of the block by including its hash in the next block they create

Implicit consensus: in each round, a random node is somehow selected, and this node Gets to propose the next block in the chain. There is no consensus algorithm for selecting the block, and no voting of any kind.
The chosen node unilaterally proposes what the next block in the block chain will be.

# Distributed consensus: available algorithms

- Proof of Work (PoW): This type of consensus mechanism relies on proof that adequate computational resources have been spent before proposing a value for acceptance by the network. This scheme is used in Bitcoin, Litecoin, and other cryptocurrency blockchains.
- Proof of Stake (PoS): This algorithm works on the idea that a node or user has an adequate stake in the system that is, the user has invested enough in the system so tat any malicious attempt by that user would outweight the benefits of performing such an attack on the network. This idea was first introduced by Peercoin, and it is going to be used in Ethereum blockchain version called Serenity.

# Distributed consensus: available algorithms

- Proof of Deposit (PoD): In this case, nodes that wish to participate in the network have to make a security deposit before they can mine and propose blocks. This mechanism is used in the Tendeermint blockhain.
- Proof of Elaspsed Time (PoET): Introduced by Intel in 2016, PoET uses a Trusted Execution Environment (TEE) to provide randomness and safety in the leader election process via a guaranteed wait time.
- Proof of Activity (PoA): This scheme is a combination of PoS and PoW, which ensures that a stakeholder is selected in a pseudorandom but uniform fashion.

# Distributed consensus: available algorithms

- Reputation based mechanisms: a leader is elected by the reputation it has built over time on the network. It is based on the votes of other members.
- Federated consensus or federated Byzantine consensus: this mechanism is used in the stellar consensus protocol. Nodes in this protocol retain a group of publicly-trusted peers and propagate only those transactions that have been validated by the majority of trusted nodes.

# Mining and consensus

- Consensus challenge: can we give nodes an incentive for behaving honestly?
- Proof-of-work is that we approximate the selection of a random node by instead selecting nodes in proportion to a resource that we hope that nobody can monopolize. If, for example, that resource is computing power, then it's a proof-of-work system
- Proportion to ownership of the currency, and that's called proof-of-stake
- Mining is the mechanism that underpins the decentralized clearinghouse, by which transactions are validated and cleared.
- The primary purpose of mining is not the reward or the generation of new coins.

# Mining and consensus

- Mining *secures the bitcoin system* and enables the emergence of network-wide *consensus without a central authority*
- The reward of newly minted coins and transaction fees is an incentive scheme that aligns the actions of miners with the security of the network, while simultaneously implementing the monetary supply.
- Miners validate new transactions and record them on the global ledger. A new block, containing transactions that occurred since the last block, is "mined" every 10 minutes on average, thereby adding those transactions to the blockchain.
- Transactions that become part of a block and added to the blockchain are considered "confirmed"

# Bitcoin's decentralized consensus

- Independent verification of each transaction, by every full node, based on a comprehensive list of criteria
- Independent aggregation of those transactions into new blocks by mining nodes, coupled with demonstrated computation through a Proof-of-Work algorithm
- Independent verification of the new blocks by every node and assembly into a chain
- Independent selection, by every node, of the chain with the most cumulative computation demonstrated through Proof-of-Work

# Proof of Work

- Miners receive two types of rewards in return for the security provided by mining: new coins created with each new block, and transaction fees from all the transactions included in the block
- To earn this reward, miners compete to solve a difficult mathematical problem based on a cryptographic hash algorithm.
- The solution to the problem, called the Proof-of-Work, is included in the new block and acts as proof that the miner expended significant computing effort

# Proof of work algorithm

- Hash puzzles: In order to create a block, the node that proposes that block is required to find a number, or nonce , such that when you concatenate the nonce, the previous hash, and the list of transactions that comprise that block and take the hash of this whole string, then that hash output should be a number that falls into a target space that is quite small in relation to the much larger output space of that hash function.

$$H(nonce \,||\, prev\_hash \,||\, tx \,||\, tx \,||\, \ldots \,||\, tx) < target$$

# Hash puzzles

- A hash algorithm takes an arbitrary-length data input and produces a fixed-length deterministic result, a digital fingerprint of the input.
- For any specific input, the resulting hash will always be the same and can be easily calculated and verified by anyone implementing the same hash algorithm.
- The key characteristic of a cryptographic hash algorithm is that it is computationally infeasible to find two different inputs that produce the same fingerprint (known as a *collision*)
- It is also virtually impossible to select an input in such a way as to produce a desired fingerprint, other than trying random inputs.

# SHA256 example

```
MacBook-Air-de-lahmadi:scripts AbdelkaderLahmadi$ python
Python 2.7.14 (default, Sep 22 2017, 00:06:49)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import hashlib
>>> print hashlib.sha256("I am a blockchain").hexdigest()
a23c427ba7b5908e0ccfc1e2e681d30d4a82345387ef5a5a6dac7f2339c8da24
>>>
```

a23c427ba7b5908e0ccfc1e2e681d30d4a82345387ef5a5a6dac7f2339c8da24
This 256-bit number is the *hash* or *digest* of the phrase and depends on every part of the phrase. Adding a single letter, punctuation mark, or any other character will produce a different hash.

# SHA256 puzzle

- If we change the phrase, we obtain different hashes

```
import hashlib
text = "I am a blockchain"
# iterate nonce from 0 to 19
for nonce in xrange(20):
    # add the nonce to the end of the text
    input = text + str(nonce)
    # calculate the SHA-256 hash of the input (text+nonce)
    hash = hashlib.sha256(input).hexdigest()
    # show the input and hash result
    print input, '=>', hash
```

```
I am a blockchain0 => 9427fa52a1661a32ffafbe3a58e6d841cf07565b803a0b0fec58bbeee772f3bd
I am a blockchain1 => 4ffdcc1360522d6339507a7c8a935e266b41e5561e75362088d4b2e9835a0a61
I am a blockchain2 => 9e2eb46d9451e4bf996ee8c9a34919e30314bac681e2f7bd321e01c4b8242fbe
I am a blockchain3 => 37fd75cc76aeb6a7c5fb876c35efc15753084e706906be0ba279ce0cb8c7f2f4
I am a blockchain4 => f68f02e41bd9c31f03cb9ae271792a997b3a1a2e401ef708bc88f5d01c7a7a14
I am a blockchain5 => b3828d5f57ef7ee600cd06850c51e2cd8f79cf934192a50408d287d769c07b91
I am a blockchain6 => 6ed73286965c2cd1eaf3da499c1d230b491063af578a5c927a9de88c84889705
I am a blockchain7 => f454d15844c6dce75b976a0bf908c4a006e674843dd514e1b8a402501031cc01
I am a blockchain8 => 3613fe0839106ee61ee51795555cd645fae5498e288438ef1c84902b9de65650
I am a blockchain9 => 9ecd717e2989c7608d483d8be4f9560a729947d9404c2144e3ba0c56e7bcec12
I am a blockchain10 => e5cf317b690bb82ae3f6616cad4e989d68de7984aa8a8a8a98905e1b03780c9e
I am a blockchain11 => 70d6621be5ce9c07da8611e67342cbf691485784cf37fd0f3aa518186d00f8fb
I am a blockchain12 => 809181900e95a6ce54a6ac5f04aba42e76ac0d534765c59194f315cc255d2e59
I am a blockchain13 => 5be1928b1e74813d4e09633d267df3c4519b8cae545b5158035e0ff68c20ffe2
I am a blockchain14 => c692dc55dcd2bd3d279009fd3a24f13064a280e7f5d21d9d863a9565e383db3e
I am a blockchain15 => 61c849f408119f7ec43ee1f2b8aa07a21771d87ae2dc10a8bb4233ebfd45ac17
I am a blockchain16 => b4d21c719b142160ebe7c4f8f96b751b15101a30ff7e363ab66254f15e360400
I am a blockchain17 => 4b1f8dfbbc1afe5d3e11e9e5160d0b2b4b122dad2145f6d67f3c8281500208ae
I am a blockchain18 => d96c92f770c39e38626ed1da448c2064faabf1f394d748d6652e28c6db58d7b5
I am a blockchain19 => 04c3b7d645094cf0a184b82694a4ad080c262de7cb8fa8da34b6a324b21c3c8f
```

# SHA256 puzzle

```
import hashlib
text = "I am a blockchain"
# iterate nonce from 0 to 19
for nonce in xrange(20):
    # add the nonce to the end of the text
    input = text + str(nonce)
    # calculate the SHA-256 hash of the input (text+nonce)
    hash = hashlib.sha256(input).hexdigest()
    # show the input and hash result
    print input, '=>', hash
```

- The number used as a variable in such a scenario is called a nonce
- The nonce is used to vary the output of a cryptographic function, in this case to vary the SHA-256 fingerprint of the phrase.
- To make a challenge out of this algorithm, let's set an arbitrary target: find a phrase that produces a hexadecimal hash that starts with a zero.
- I am a blockchain19 => 04c3b7d645094cf0a184b82694a4ad080c262de7cb8fa8da34b6a324b21c3c8f
- 19 attempts

# SHA256 puzzle

```
import hashlib
text = "I am a blockchain"
# iterate nonce from 0 to 19
for nonce in xrange(20):
    # add the nonce to the end of the text
    input = text + str(nonce)
    # calculate the SHA-256 hash of the input (text+nonce)
    hash = hashlib.sha256(input).hexdigest()
    # show the input and hash result
    print input, '=>', hash
```

- I am a blockchain19 => 04c3b7d645094cf0a184b82694a4ad080c262de7cb8fa8da34b6a324b21c3c8f
- Wining nonce is 19
- We did the work to find such nonce
- In numerical terms, that means finding a hash value that is less than:
  0x1000000000000000000000000000000000000000000000000000000000000000
- We call this threshold the target and the goal is to find a hash that is numerically less than the target
- If we decrease the target, the task of finding a hash that is less than the target becomes more and more difficult.

# Simplified Proof-of-Work implementation

- The Proof-of-Work must produce a hash that is *less than* the target. A higher target means it is less difficult to find a hash that is below the target. A lower target means it is more difficult to find a hash below the target. The target and difficulty are inversely related.
- The miner constructs a candidate block filled with transactions.
- Next, the miner calculates the hash of this block's header and sees if it is smaller than the current *target*.
- If the hash is not less than the target, the miner will modify the nonce (usually just incrementing it by one) and try again

# Simplified Proof-of-Work implementation

```python
import hashlib
import time

max_nonce = 2 ** 32  # 4 billion

def proof_of_work(header, difficulty_bits):
    # calculate the difficulty target
    target = 2 ** (256 - difficulty_bits)

    for nonce in xrange(max_nonce):
        hash_result = hashlib.sha256(str(header) + str(nonce)).hexdigest()

        # check if this is a valid result, below the target
        if long(hash_result, 16) < target:
            print("Success with nonce %d" % nonce)
            print("Hash is %s" % hash_result)
            return (hash_result, nonce)

    print("Failed after %d (max_nonce) tries" % nonce)
    return nonce

if __name__ == '__main__':
    nonce = 0
    hash_result = ''

    # difficulty from 0 to 31 bits
    for difficulty_bits in xrange(32):
        difficulty = 2 ** difficulty_bits
        print("Difficulty: %ld (%d bits)" % (difficulty, difficulty_bits))
        print("Starting search...")

        # checkpoint the current time
        start_time = time.time()

        # make a new block which includes the hash from the previous block
        # we fake a block of transactions - just a string
        new_block = 'test block with transactions' + hash_result

        # find a valid nonce for the new block
        (hash_result, nonce) = proof_of_work(new_block, difficulty_bits)

        # checkpoint how long it took to find a result
        end_time = time.time()

        elapsed_time = end_time - start_time
        print("Elapsed Time: %.4f seconds" % elapsed_time)

        if elapsed_time > 0:

            # estimate the hashes per second
            hash_power = float(long(nonce) / elapsed_time)
            print("Hashing Power: %ld hashes per second" % hash_power)
```

```
Difficulty: 8 (3 bits)
Starting search...
Success with nonce 9
Hash is 1c1c105e65b47142f028a8f93ddf3dabb9260491bc64474738133ce5256cb3c1
Elapsed Time: 0.0001 seconds
Hashing Power: 91846 hashes per second
Difficulty: 16 (4 bits)
Starting search...
Success with nonce 25
Hash is 0f7becfd3bcd1a82e06663c97176add89e7cae0268de46f94e7e11bc3863e148
Elapsed Time: 0.0002 seconds
Hashing Power: 129453 hashes per second
Difficulty: 32 (5 bits)
Starting search...
Success with nonce 36
Hash is 029ae6e5004302a120630adcbb808452346ab1cf0b94c5189ba8bac1d47e7903
Elapsed Time: 0.0003 seconds
Hashing Power: 141779 hashes per second

Difficulty: 4194304 (22 bits)
Starting search...
Success with nonce 1759164
Hash is 0000008bb8f0e731f0496b8e530da984e85fb3cd2bd81882fe8ba3610b6cefc3
Elapsed Time: 5.5412 seconds
Hashing Power: 317472 hashes per second

Difficulty: 16777216 (24 bits)
Starting search...
Success with nonce 24586379
Hash is 0000002c3d6b370fccd699708d1b7cb4a94388595171366b944d68b2acce8b95
Elapsed Time: 76.6582 seconds
Hashing Power: 320727 hashes per second

Difficulty: 67108864 (26 bits)
Starting search...
Success with nonce 84561291
Hash is 0000001f0ea21e676b6dde5ad429b9d131a9f2b000802ab2f169cbca22b1e21a
Elapsed Time: 240.9574 seconds
Hashing Power: 350938 hashes per second
```

# Simplified Proof-of-Work implementation

- As you can see, increasing the difficulty by 1 bit causes a doubling in the time it takes to find a solution.
- If you think of the entire 256-bit number space, each time you constrain one more bit to zero, you decrease the search space by half
- The bitcoin network is attempting to find a block whose header hash is less than:

0000000000000000029AB900000000000000000000000000000000000000

- There are a lot of zeros at the beginning of that target, meaning that the acceptable range of hashes is much smaller, hence it's more difficult to find a valid hash
- It will take on average more than 1.8 zeta-hashes (thousand billion billion hashes) per second for the network to discover the next block

# Hash target representation

Target = 0x1903a30c


Difficulty target = coefficient * $2^{(8*(\text{exponent}-3))}$

Using that formula, and the difficulty bits value 0x1903a30c, we get:

target = 0x03a30c * $2^{0x08*(0x19-0x03)}$

=> target = 0x03a30c * $2^{(0x08*0x16)}$

=> target = 0x03a30c * $2^{0xB0}$

which in decimal is:

$\Rightarrow$ target = 238,348 * $2^{176}$

Target = 22,829,202,948,393,929,850,749,706,076,701,368,331,072,452,018,388,575,715,328

Target = 0x0000000000000003A30C0000000000000000000000000000000000000000000

# Retargeting: adjust difficulty

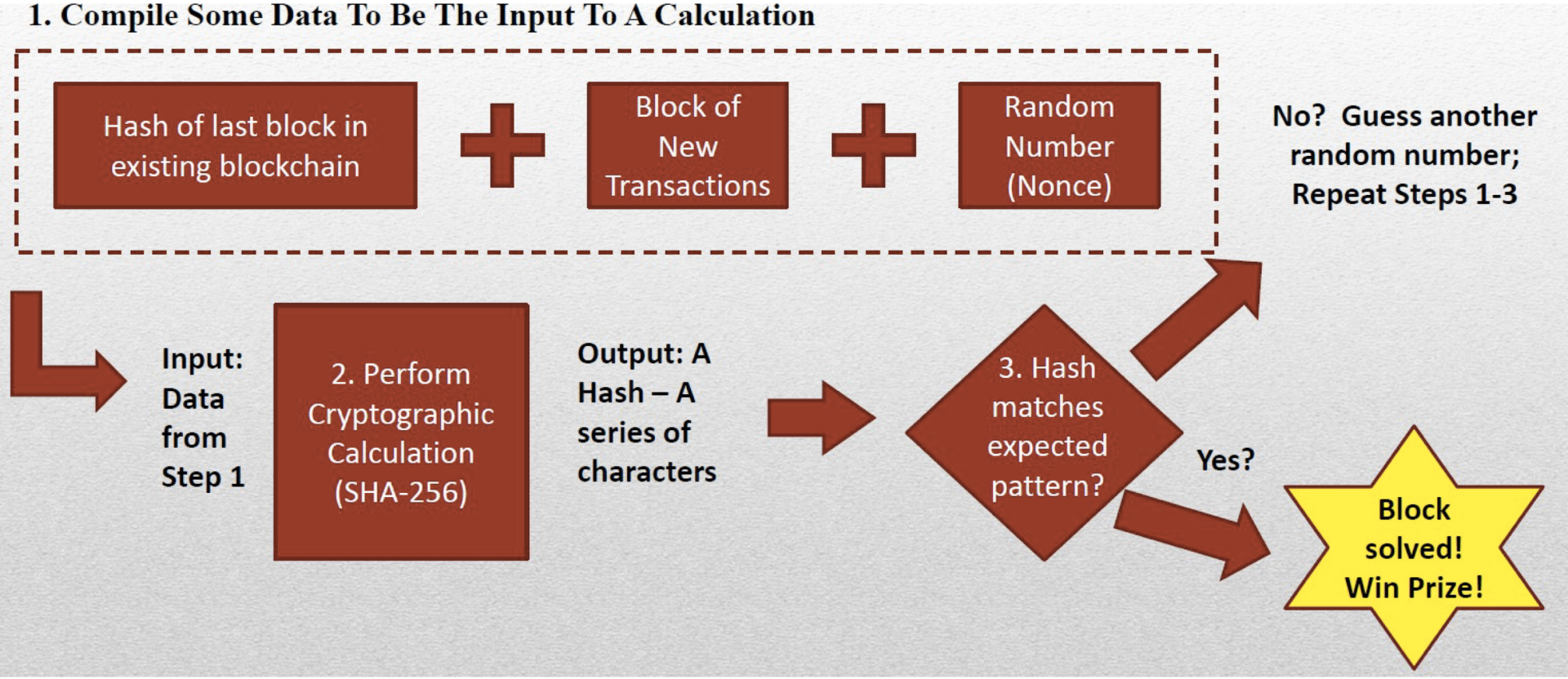
- The target determines the difficulty and therefore affects how long it takes to find a solution to the Proof-of-Work algorithm.
- Why is the difficulty adjustable, who adjusts it, and how?
- Bitcoin's blocks are generated every 10 minutes, on average it has to remain constant not just over the short term, but over a period of many decades
- Retargeting occurs automatically and on every node independently.
- Every 2,016 blocks, all nodes retarget the Proof-of-Work
- If the network is finding blocks faster than every 10 minutes, the difficulty increases (target decreases). If block discovery is slower than expected, the difficulty decreases (target increases).

```
New Target = Old Target * (Actual Time of Last 2016 Blocks / 20160 minutes)
```

# Successfully mining the block

- Produce a block hash less than the target
- Transmits the block to all the peers
- They receive, validate, and then propagate the new block
- As the block ripples out across the network, each node adds it to its own copy of the blockchain, extending it to a new height
- As mining nodes receive and validate the block, they abandon their efforts to find a block at the same height and immediately start computing the next block in the chain

# PoW steps


1. Compile Some Data To Be The Input To A Calculation
Hash of last block in existing blockchain
Block of New Transactions
Random Number (Nonce)
Input: Data from Step 1
2. Perform Cryptographic Calculation (SHA-256)
Output: A Hash – A series of characters
3. Hash matches expected pattern?
No? Guess another random number; Repeat Steps 1-3
Yes?
Block solved! Win Prize!

# Validating a new block

- The third step in the consensus mechanism is independent validation of each new block by every node on the network.
- The newly solved block moves across the network, each node performs a series of tests to validate it before propagating it to its peers:
  - The block data structure is syntactically valid
  - The block header hash is less than the target (enforces the Proof-of-Work)
  - The block timestamp is less than two hours in the future (allowing for time errors)
  - The block size is within acceptable limits
  - All transactions within the block are valid using the transaction

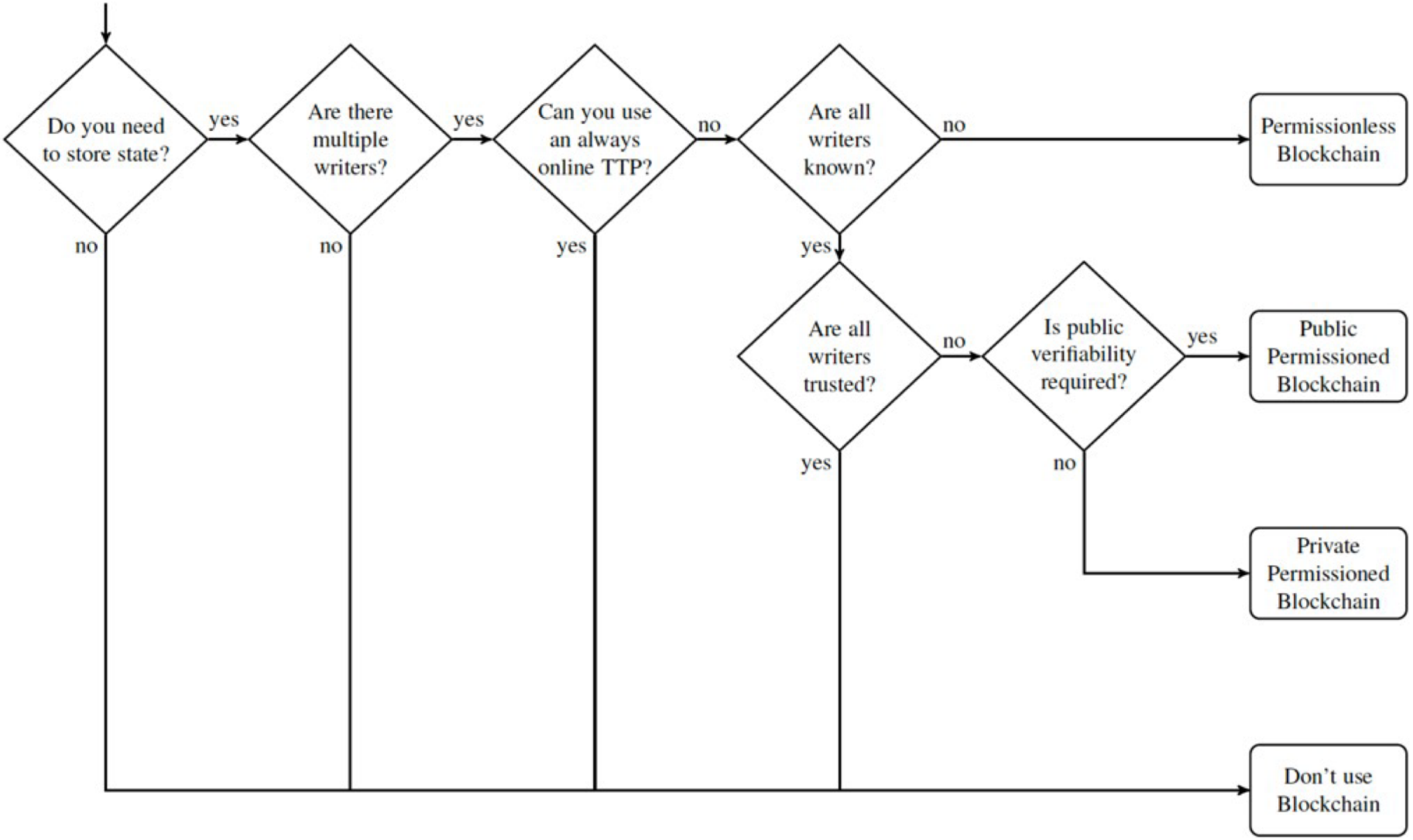
# Assembling chains of blocks

- Assembly of blocks into chains and the selection of the chain with the most Proof-of-Work
- Nodes maintain three sets of blocks:
  - those connected to the main blockchain,
  - those that form branches off the main blockchain (secondary chains),
  - blocks that do not have a known parent in the known chains (orphans)
- The "main chain" at any time is whichever *valid* chain of blocks has the most cumulative Proof-of-Work associated with it
- By selecting the greatest-cumulative-work valid chain, all nodes eventually achieve network-wide consensus

# Consensus attacks

- Consensus attacks can only affect future consensus, or at best, the most recent past (tens of blocks).
- The ledger becomes more and more immutable as time passes
- 51% attack:  group of miners, controlling a majority (51%) of the total network's hashing power, collude to attack bitcoin
  - With the ability to mine the majority of the blocks, the attacking miners can cause deliberate "forks" in the blockchain and double-spend transactions or execute denial-of-service attacks against specific transactions or addresses
  - A fork/double-spend attack is where the attacker causes previously confirmed blocks to be invalidated by forking below them and re-converging on an alternate chain
  - Note that a double-spend can only be done on the attacker's own transactions, for which the attacker can produce a valid signature
- To protect against this kind of attack, a merchant selling large-value items must wait at least six confirmations before giving the product to the buyer

# Conclusions: do you need a blockchain?




K. Wiust and A. Gervais., "Do You Need a Blockchain?", https://eprint.iacr.org/2017/375.pdf