UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Lecture 13 – Blockchain

Prof. Esposito Christian

::.. Summary

- Introduction
  - Transactions and trusted entities;
  - Blockchain Classification;
  - Distributed Consensus.
- Smart Contracts
  - Introduction;
  - BitCoin, Etherium, Hyperledger Fabric.
- Applications in the IoT context

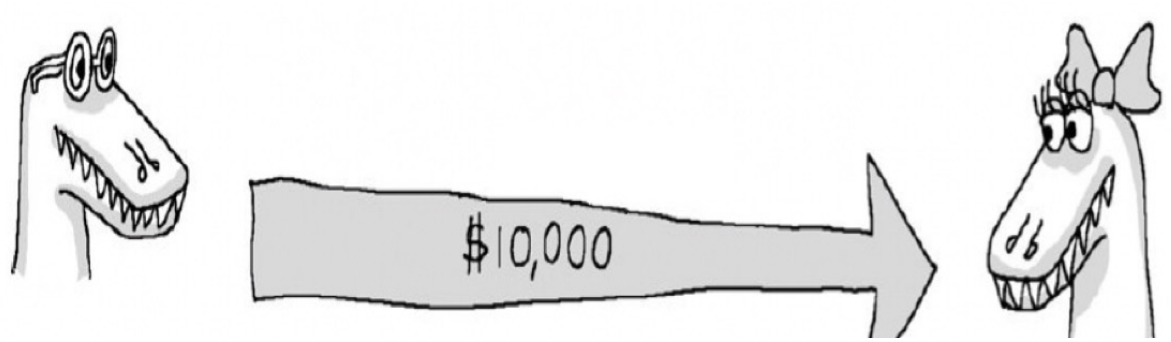
::.. Key Lectures

- W. Wang et al., "A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks ", in IEEE Access, vol. 7, pp. 22328-22370, 2019.
- Y. Xiao, N. Zhang, W. Lou and Y. T. Hou, "A Survey of Distributed Consensus Protocols for Blockchain Networks", in IEEE Communications Surveys & Tutorials, vol. 22, no. 2, pp. 1432-1465, 2020.
- https://ethereum.org/en/developers/
- https://hyperledger-fabric.readthedocs.io/en/release-2.2/
- V. Y. Kemmoe, W. Stone, J. Kim, D. Kim and J. Son, "Recent Advances in Smart Contracts: A Technical Overview and State of the Art", in IEEE Access, vol. 8, pp. 117782-117801, 2020.
- Reyna, Ana, et al. "On blockchain and its integration with IoT. Challenges and opportunities." Future generation computer

Introduction to Blockchain

::.. Asset Transfer (1/2)

- Two nodes in a network intend to exchange an asset.
- The goal is to be able to transfer this asset reliably and securely.



::.. Asset Transfer (2/2)

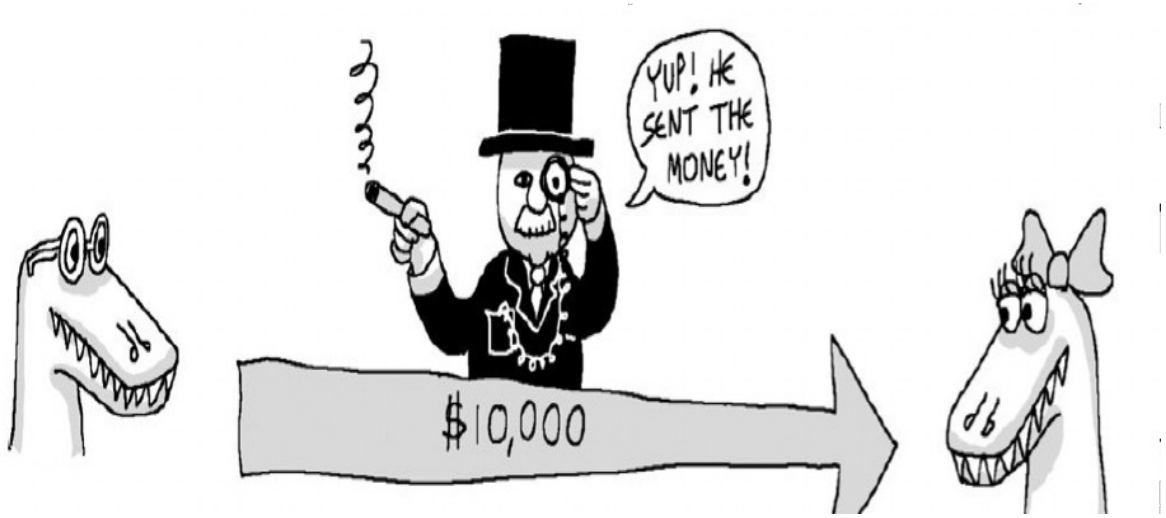
Failure Model:

- **Node Failures** – crash or hang.
- **Link Failures** – persistent or intermittent crash.
- **Network Misbehaviours** – Loss, modification or abnormal delay of packets or network partitioning.

Attack Model:

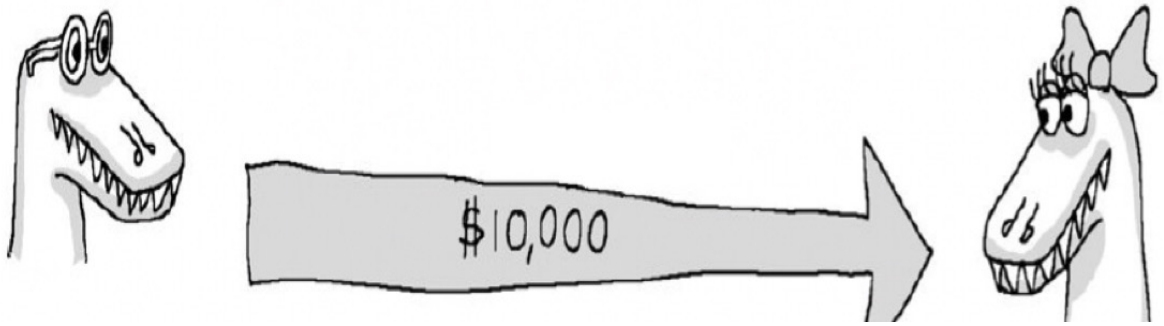
- **Replay Attack**: a valid transaction is maliciously repeated or delayed;
- **Men-in-the-middle Attack**: the communication is observed and altered by an unauthorized third party;
- **Masquerade Attack**: a malicious node uses an identity forged ad hoc for communication purposes:
  - Special case: one node impersonates another(*impersonation*).
- **Byzantine Behaviour**: a node behaves unexpectedly**.**

::.. Transactions with a Mediator



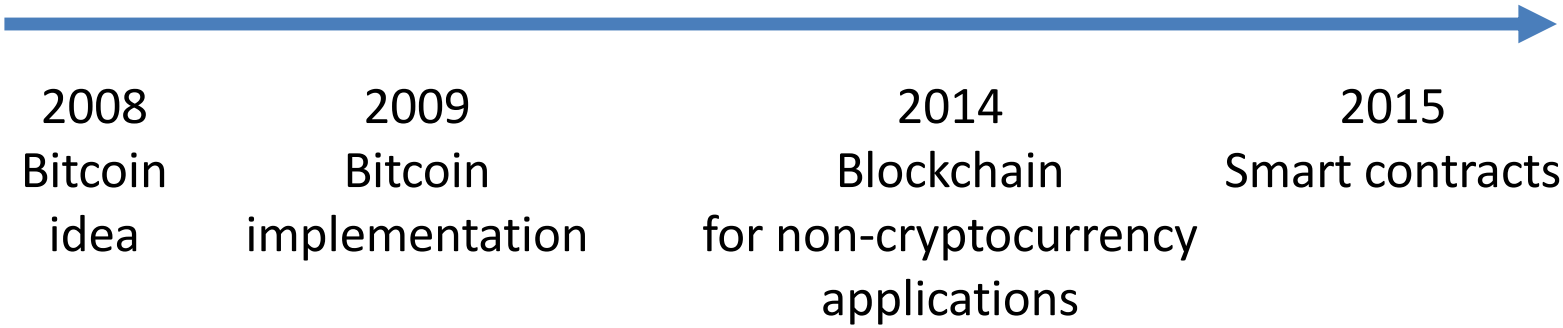
- Transaction management is accomplished by employing a third party entity (e.g. a bank) that validates, supervises and records transactions.
- This solution is implemented for example with the two-phase commit protocol (2PC).
- Typically, the solution with a mediator allows you to handle attacks of the Masquerade or Byzantine type. The third party is required to be trusted and reliable.
- The cost of this operation is high (it represents a performance bottleneck, a single-point-of-failure, a vulnerability).

::.. Transactions without a Mediator



- To send an asset without the cooperation of a third party entity, a protocol can be defined to perform reliable and secure transactions in an unreliable and insecure environment.
- This solution presents several problems related to:
  - Verification of the integrity of the received messages;
  - Verification of the identity of the participating nodes;
  - Verification of the validity of transactions;
  - …

::.. Blockchain & BitCoin

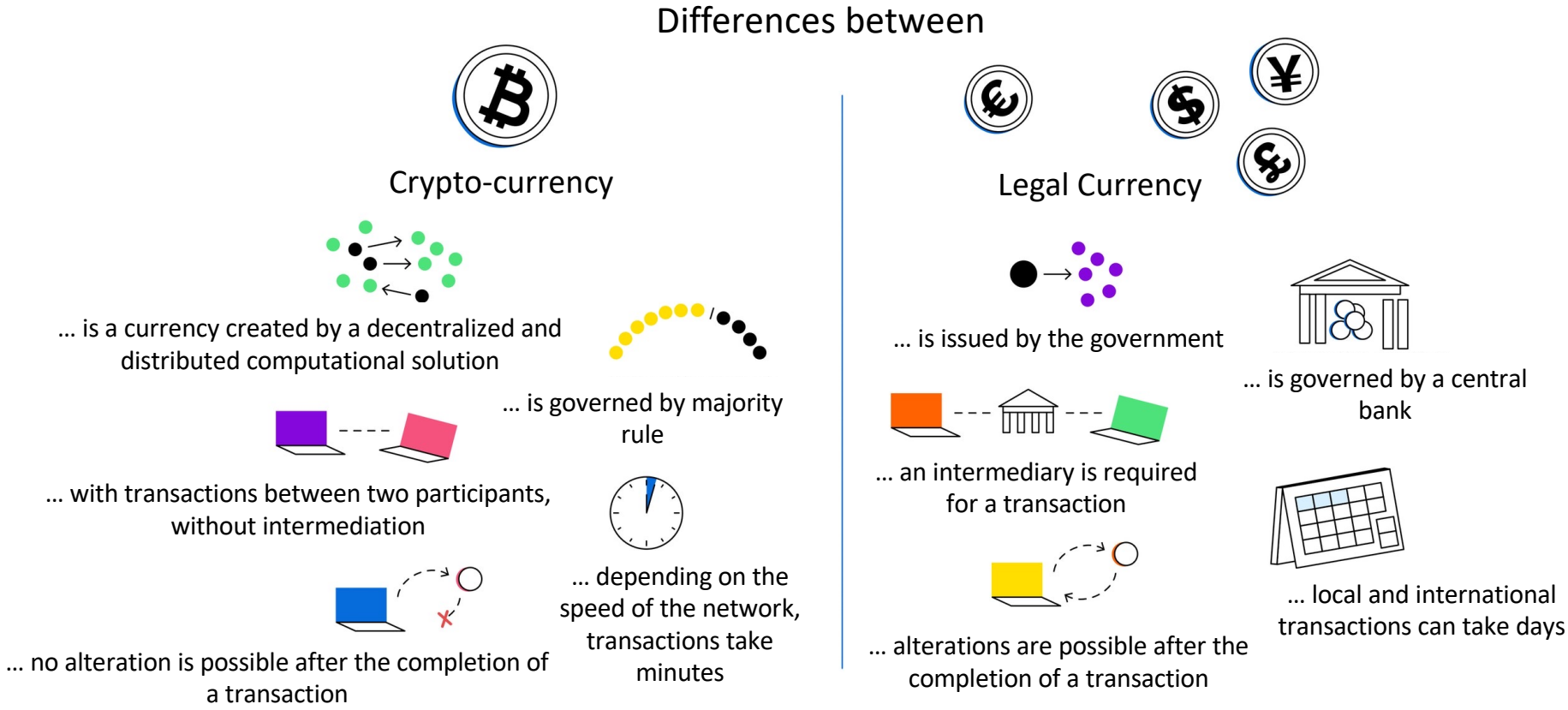


- The first area of application (2008) is the financial one, in particular the management of crypto-currencies. The purpose was to perform secure transactions using insecure infrastructures.
- In general, the goal of Blockchain technology is to create a decentralized environment in which no "third party" has control of the transactions and data.
  - *No trusted entities* (e.g., bank)

::.. Crypto-currencies (1/3)

Blockchains have been theorized to support so-called crypto-currencies:

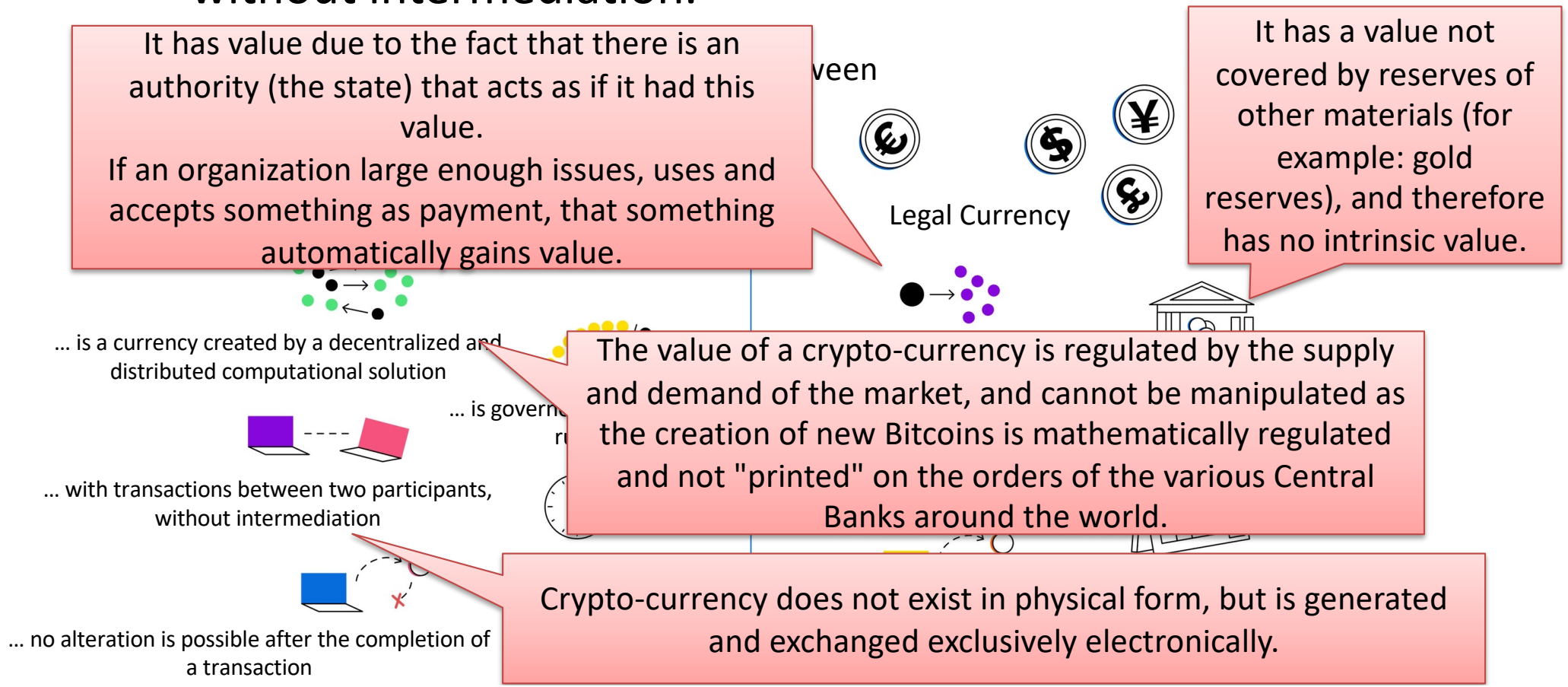
- a digital representation of value based on cryptography and without intermediation.



::.. Crypto-currencies (1/3)

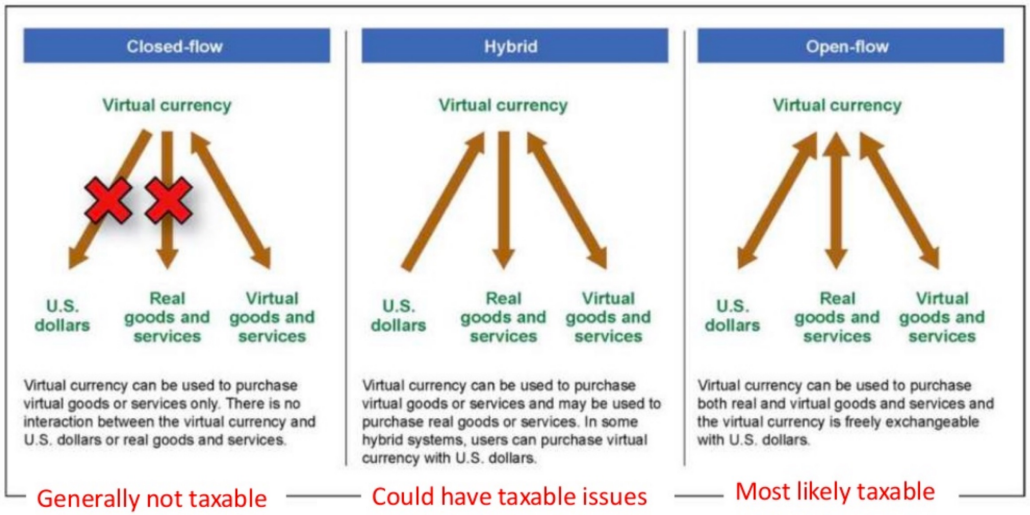
Blockchains have been theorized to support so-called crypto-currencies:

- a digital representation of value based on cryptography and without intermediation.



::.. Crypto-currencies (2/3)

Crypto-currencies fall within the case of virtual currencies but are different from digital currencies:



- A virtual currency is a digital representation of value that is neither issued by a central bank nor a public authority, nor necessarily linked to a legal tender currency, but is accepted as a means of payment and can be transferred, stored or exchanged electronically.
- Crypto-currencies are two-way virtual currencies.
- A digital currency is a virtual representation of a fiat currency.

::.. Crypto-currencies (3/3)

Physical Currency

Commodity Currency

Made with materials that have intrinsic value, or whose value is related to the stock of materials in a state.

Fiat Currency

It has no intrinsic value.
Used as money by national governments and controlled by the central bank.

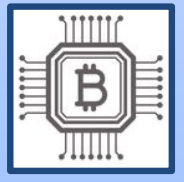
Digital Currency

Digital Fiat Currency

Digital representations of fiat currencies that can be exchanged digitally.
Used as a currency in a community of users.

Virtual Currency

Digital representations of values that can be exchanged digitally.
They do not have legal currency status.



::.. Crypto-currencies (3/3)

Physical Currency

Commodity Currency

Made with materials that have intrinsic value, or whose value is related to the stock of materials in a state.

Fiat Currency

It has no intrinsic value.
Used as money by national governments and controlled by the central bank.

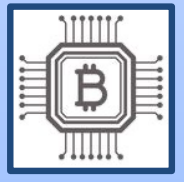
Digital Currency

Digital Fiat Currency

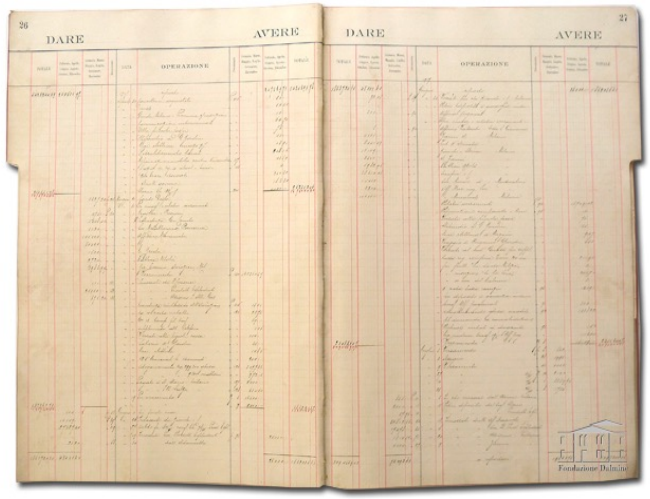
Digital representations of fiat currencies that can be exchanged digitally.
Used as a currency in a community of users.

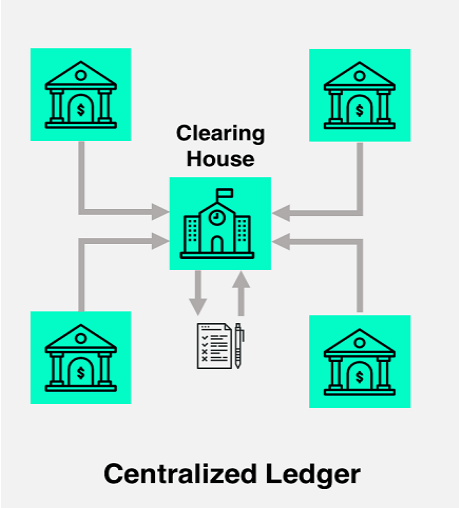
Virtual Currency

Crypto-currencies are a digital currency that is convertible and decentralized.



::.. Ledger



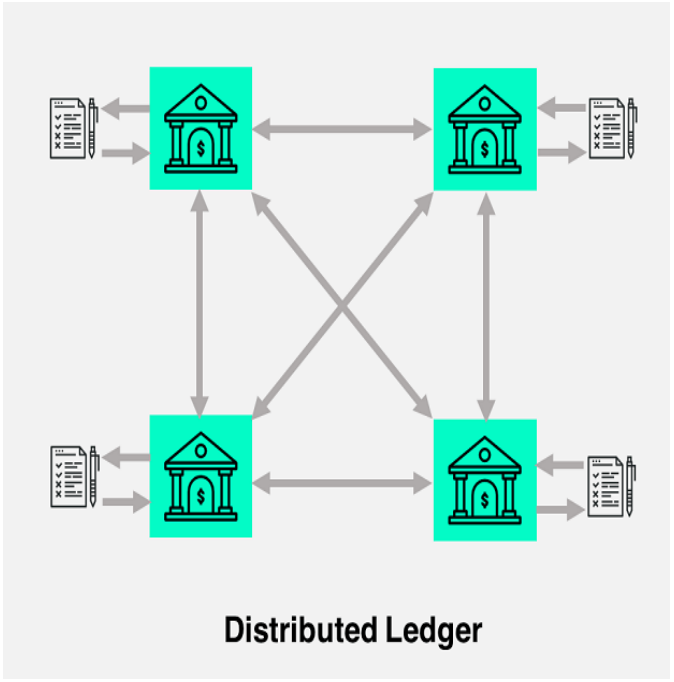


- The ledger is a register that contains all the transactions carried out between the participants of a system.
- The ledger is characterized by systematic entries; writes are recorded based on the subject they refer to.
- With reference to each object, the operations are indicated in chronological order.
- It is an "append-only multi-party system of record", which is a "log" of transactions.

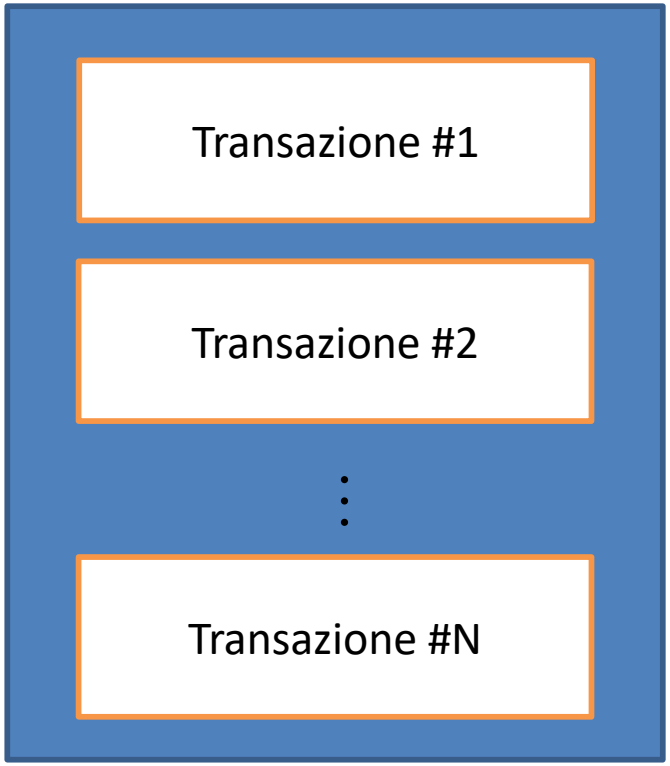
FONDAZIONE BANCO di NAPOLI

::.. Distributed Ledger

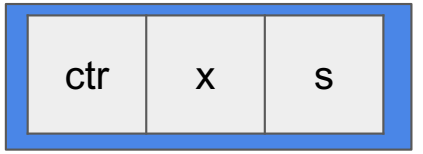
- In a distributed ledger, each node on the network stores records (transactions) in a local copy of the book sequentially, in the form of time-ordered blocks.
- For blocks to be considered valid, participants must reach a consensus quorum.
- The ledger is built like a chain of blocks.
  - Each block (except the genesis block) contains information about the previous block in the chain.



::.. Block



A block is a collection of transactions



A data structure with 3 elements:

- nonce (ctr), data (x), reference (s);

Typically called a block header, data (x) depends on the application:

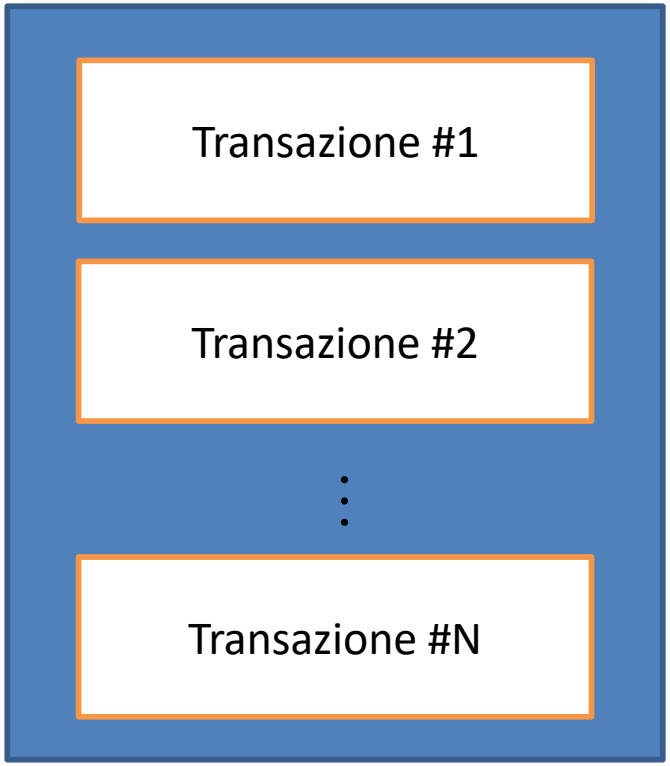
- In Bitcoin it stores financial data (based on "UTXO");
- In Ethereum it stores the contract data (based on the account)
- In Namecoin stores name data

For now we leave it undefined - we will come back to this later.

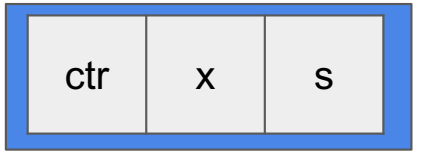
Block Validity:

- Data must be valid.

::.. Block

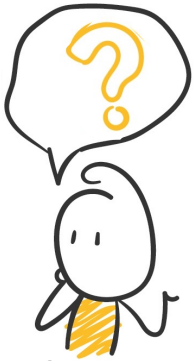


A block is a collection of transactions



A data structure with 3 elements:

- nonce (ctr), data (x), reference (s);



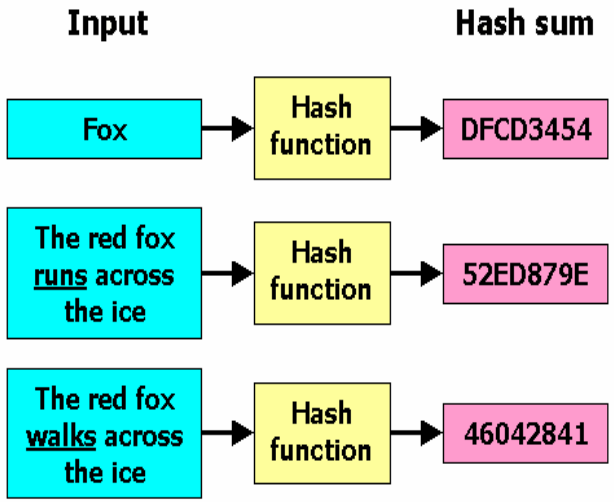
How do I connect the blocks to build the chain?

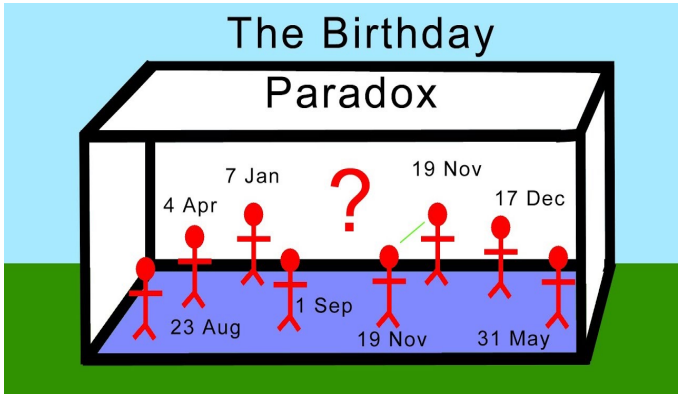
- To send an asset without the collaboration of a third party entity, a protocol can be defined to perform reliable and secure transactions in an unreliable and insecure environment.
- This solution presents several problems related to:
  - Verification of the integrity of the received messages;
  - Verification of the identity of the participating nodes;
  - Verification of the validity of transactions;
  - …

How to resolve these issues?

::.. Hash and Birthday Paradox

- Hash functions return a string of a few hundred bits (called a hash or fingerprint) starting from a sequence of bits of arbitrary length.

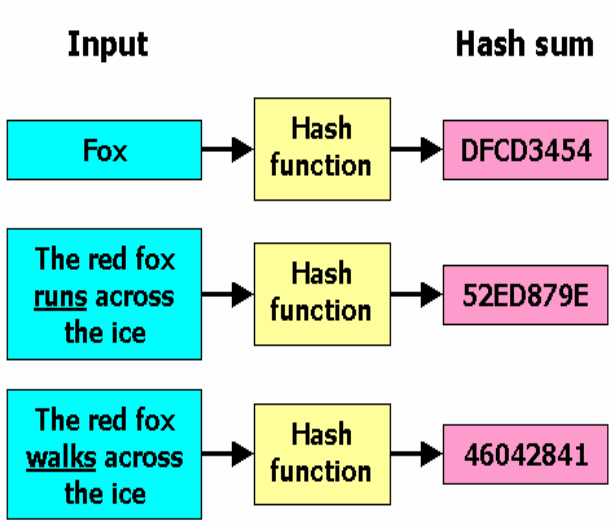


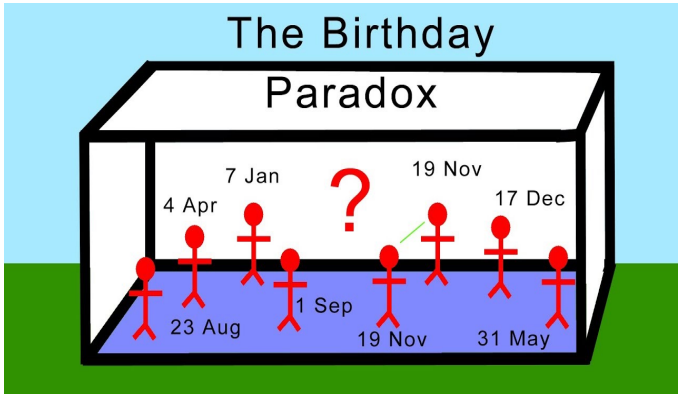


- The possibility that two sequences of bits submitted to the same hash function return the same value (collision) is defined as the "birthday paradox":

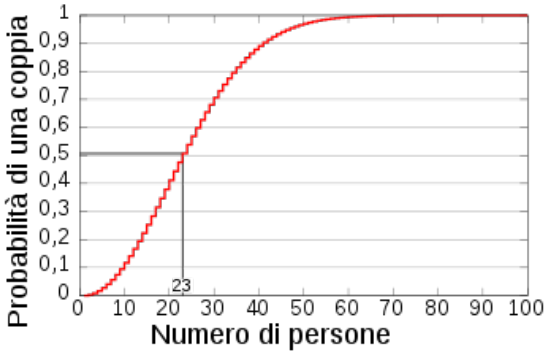
::.. Hash and Birthday Paradox

- Hash functions return a string of a few hundred bits (called a hash or fingerprint) starting from a sequence of bits of arbitrary length.





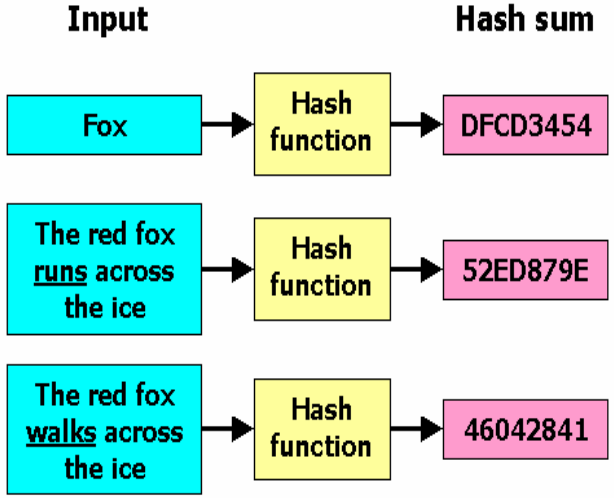
- The possibility that two sequences of bits submitted to the same hash function return the same value (collision) is defined as the "birthday paradox":



  - the likelihood of at least two people in a group having a birthday on the same day is far greater than intuition might say.

::.. Hash and Birthday Paradox

- Hash functions return a string of a few hundred bits (called a hash or fingerprint) starting from a sequence of bits of arbitrary length.

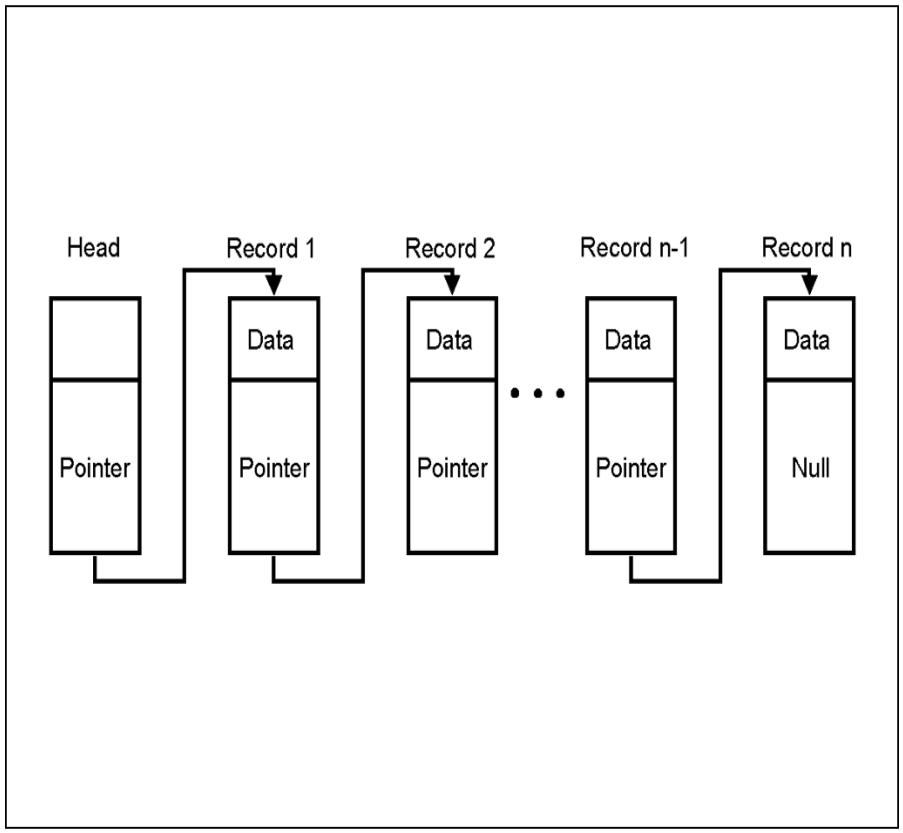


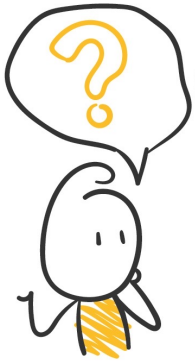
A hash function that produces a result on N bits will be considered insecure when $2^{\frac{N}{2}}$ results are generated as there is a probability of more than 50% of having found a collision, the result is evidently well below of the 2^{N-1} necessary elements suggested by intuition.

- The possibility that two sequences of bits submitted to the same hash function return the same value (collision) is defined as the "birthday paradox".
- Cryptographic hash functions try to make it computationally inadmissible to find two sequences that produce the same fingerprint.

::.. Block Chain

- A block is linked to the previous one by a hash of the previous block.
- Changing a block of the chain by a node would result in the generation of a different hash value:
  - The other nodes can check the integrity of the blocks and reject changes to already validated transactions;
  - The contents of the ledger are thus made immutable.

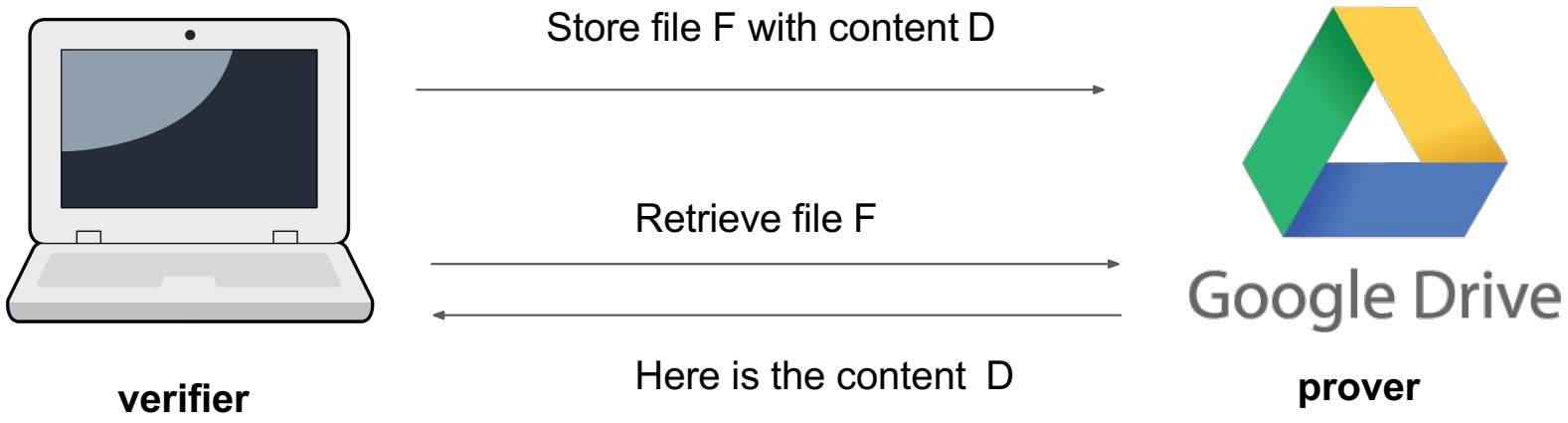




How to hash the overall block with limited overhead?

::.. Hash Tree Authentication (1/5)

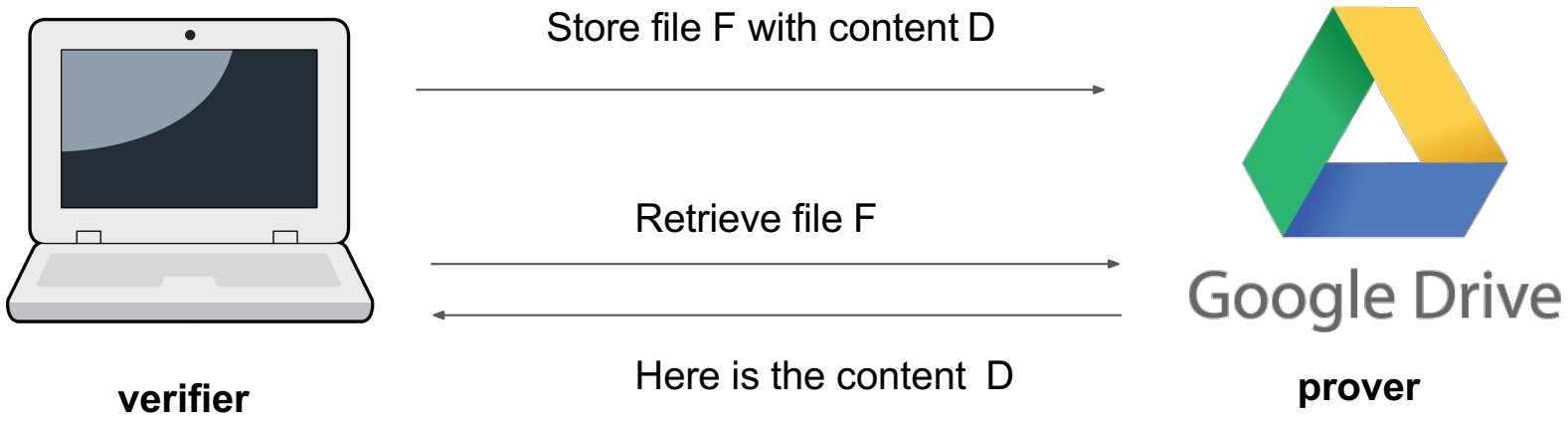
Problem of storing authenticated files



- The user wants to archive a file on a server;
- The file has a name F and content D;
- Users want to recover the F file at a later time.

::.. Hash Tree Authentication (1/5)

Problem of storing authenticated files



- The user sends the file F with the content D to the server;
- The serve files (F, D);
- The user deletes D;
- The user requests F from the server;
- The server returns D;
- The customer recovered D.

What happens if the server is contradictory and returns D' ≠ D?

::.. Hash Tree Authentication (2/5)

Naive Solution:

- The user does not delete D;
- When the server returns D', the user compares D and D'.

... what happens if the user does not have enough memory to store D for a long time?

::.. Hash Tree Authentication (2/5)

Naive Solution:

- The user does not delete D;
- When the server returns D', the user compares D and D'.

... what happens if the user does not have enough memory to store D for a long time?

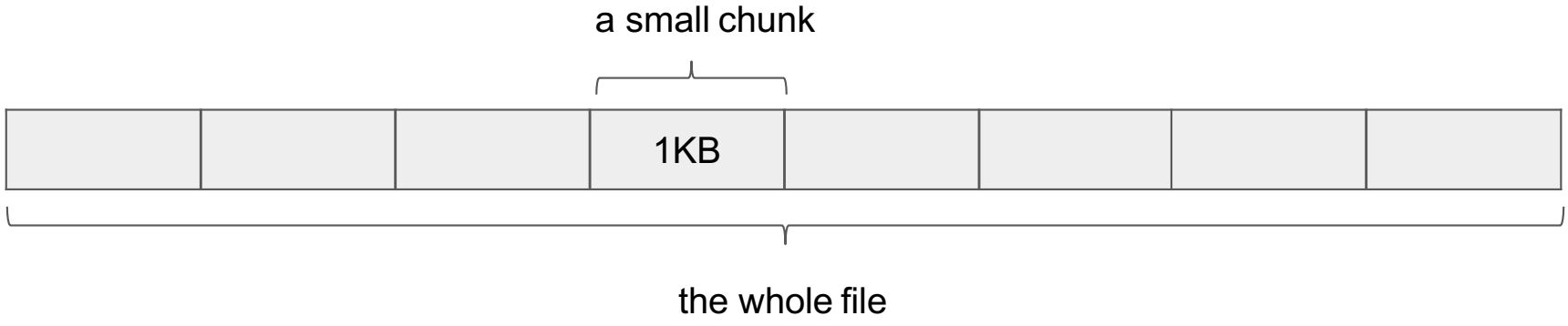
Solution with Hash:

- The user sends the F file with the D data to the server;
- The server files (F, D);
- The user stores H (D), deletes D;
- The user requests F from the server;
- The server returns D';
- The user compares $H(D') \stackrel{?}{=} H(D)$.

::.. Hash Tree Authentication (3/5)

File segmented in chuncks:

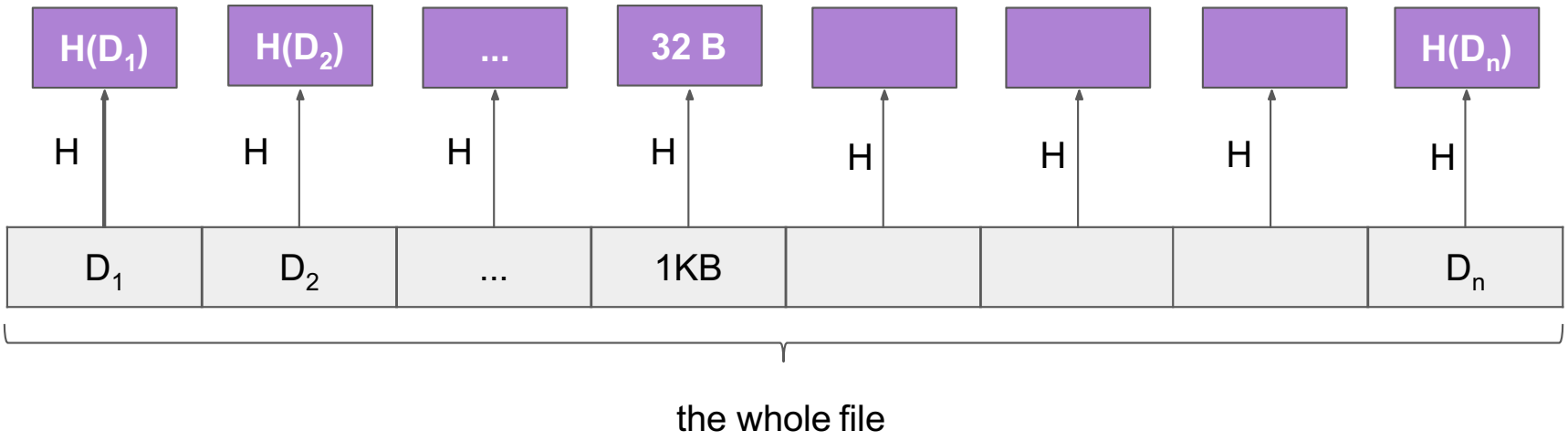
- What if the client wants to retrieve the 200,019 bytes of the file?
- You need to download the whole file ... but to avoid this we can use Merkle Trees!



::.. Hash Tree Authentication (3/5)

File segmented in chuncks:

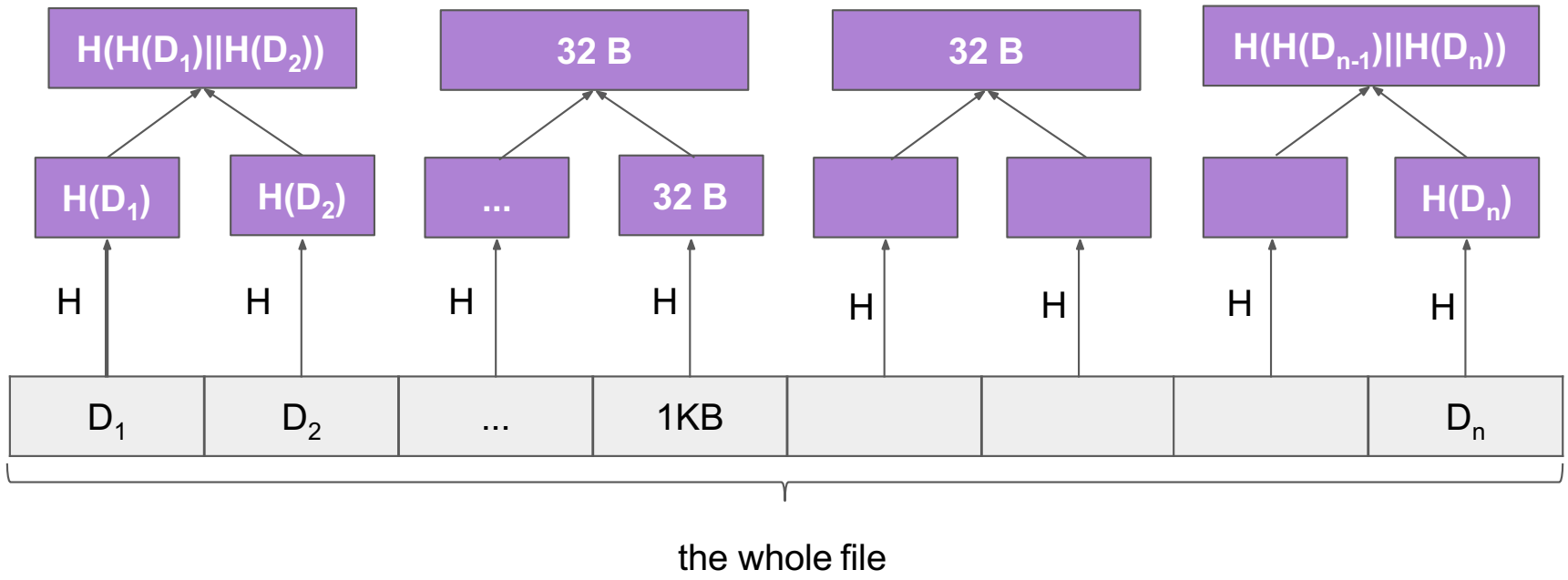
- Calculate the Hash of each block using a cryptographic hash function (SHA256);
- Convention: The arrows show the direction of applying the hash function.



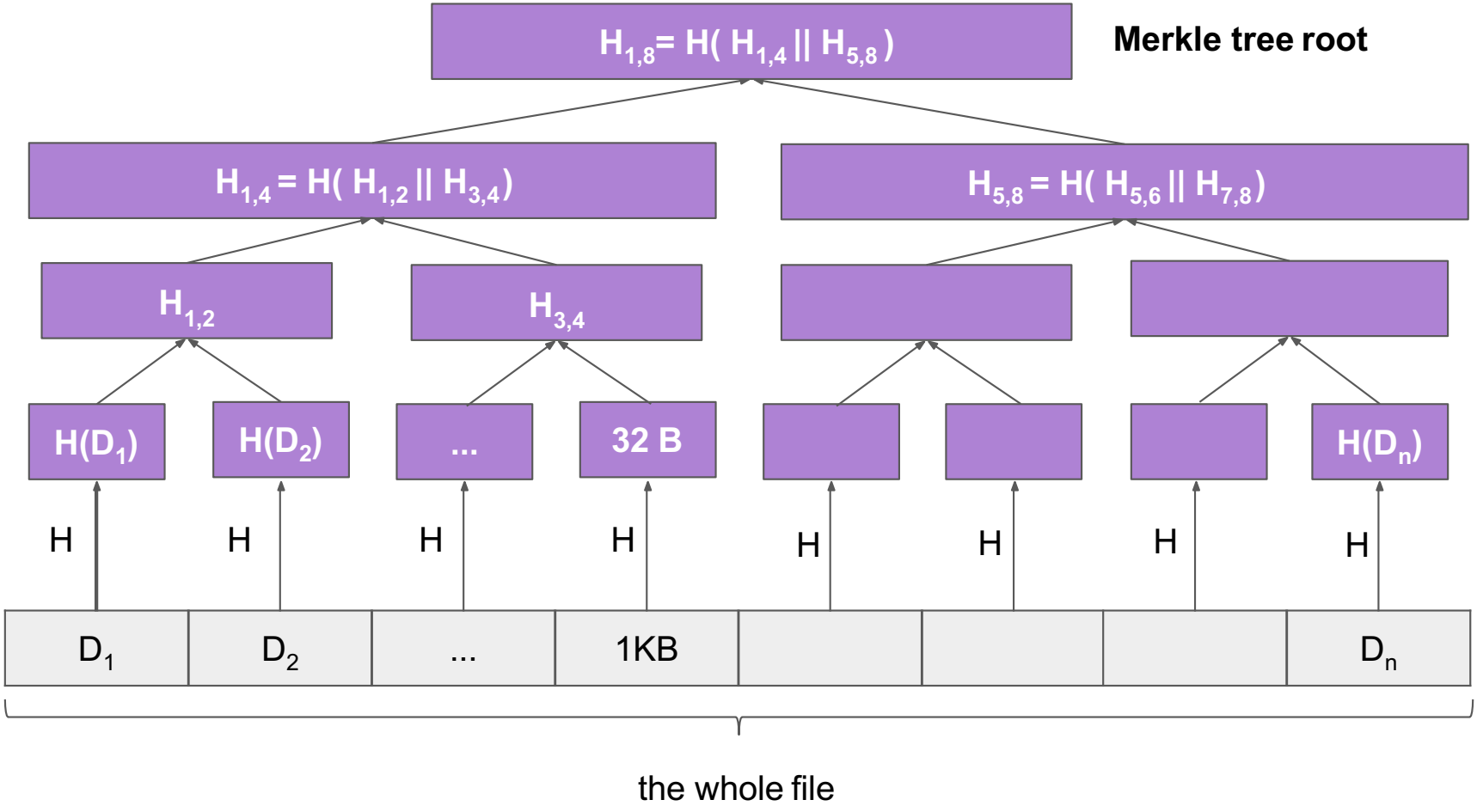
::.. Hash Tree Authentication (3/5)

File segmented in chuncks:

- Combine hashes in pairs to create a binary tree;
- Each node stores the hash of its children's concat.



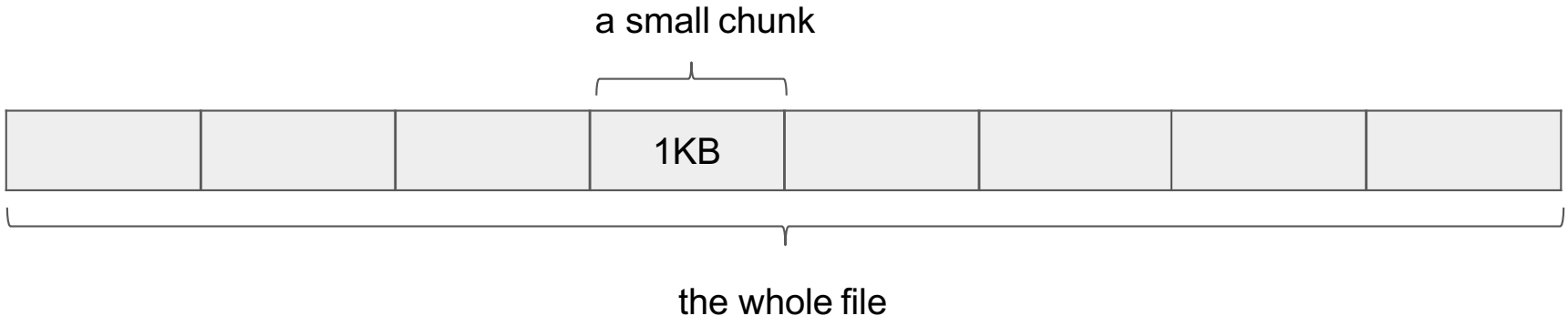
::.. Hash Tree Authentication (3/5)



::.. Hash Tree Authentication (3/5)

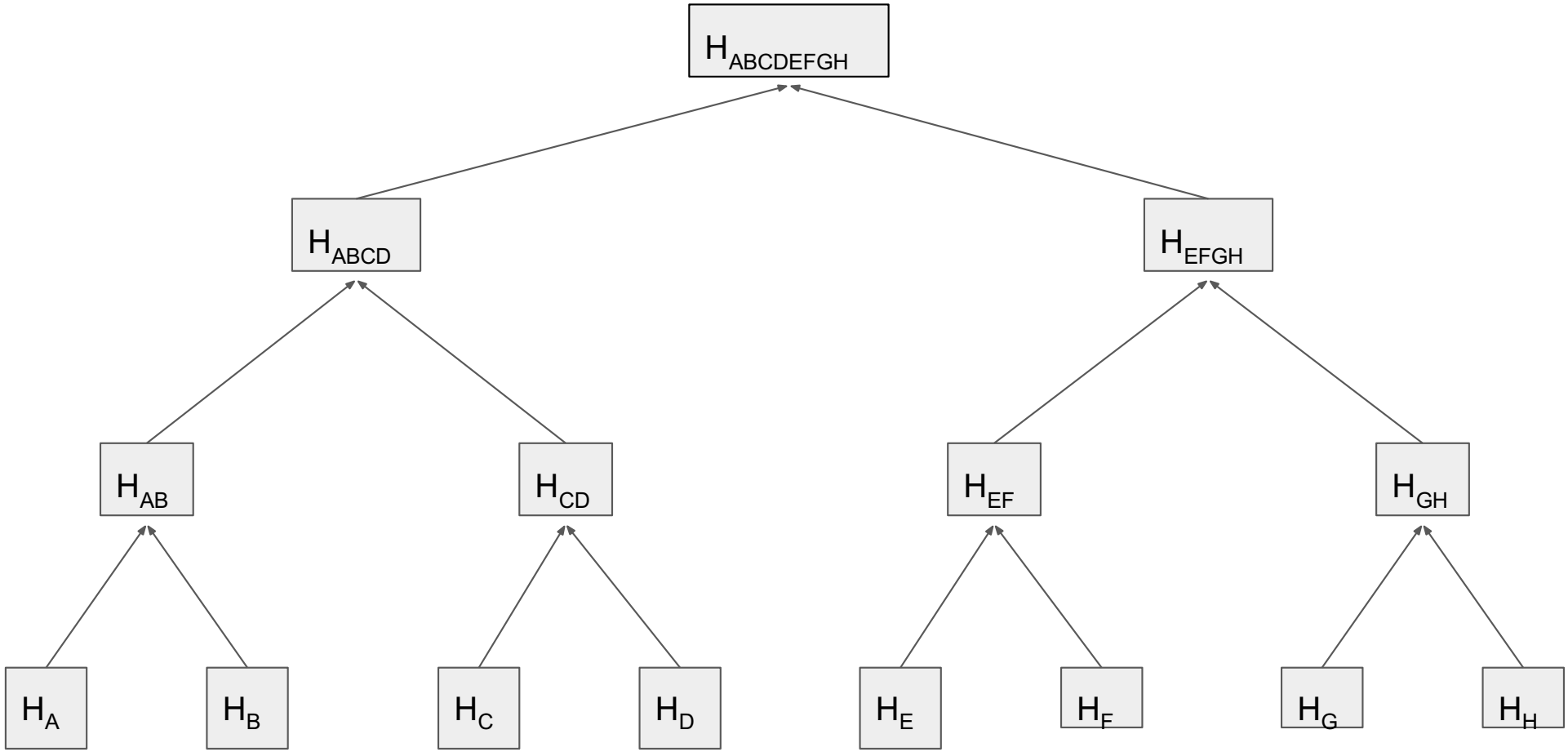
File segmented in chuncks – Solution with Merkle Trees:

- The user creates the root MTR of the Merkle Tree from the data of the initial file D;
- The user sends the data of file D to the server;
- The user deletes the data D, but stores MTR (32 bytes);
- The user requests block x from the server;
- The user returns block x and a short inclusion test π;
- The user checks that block x is included in MTR using the π proof.



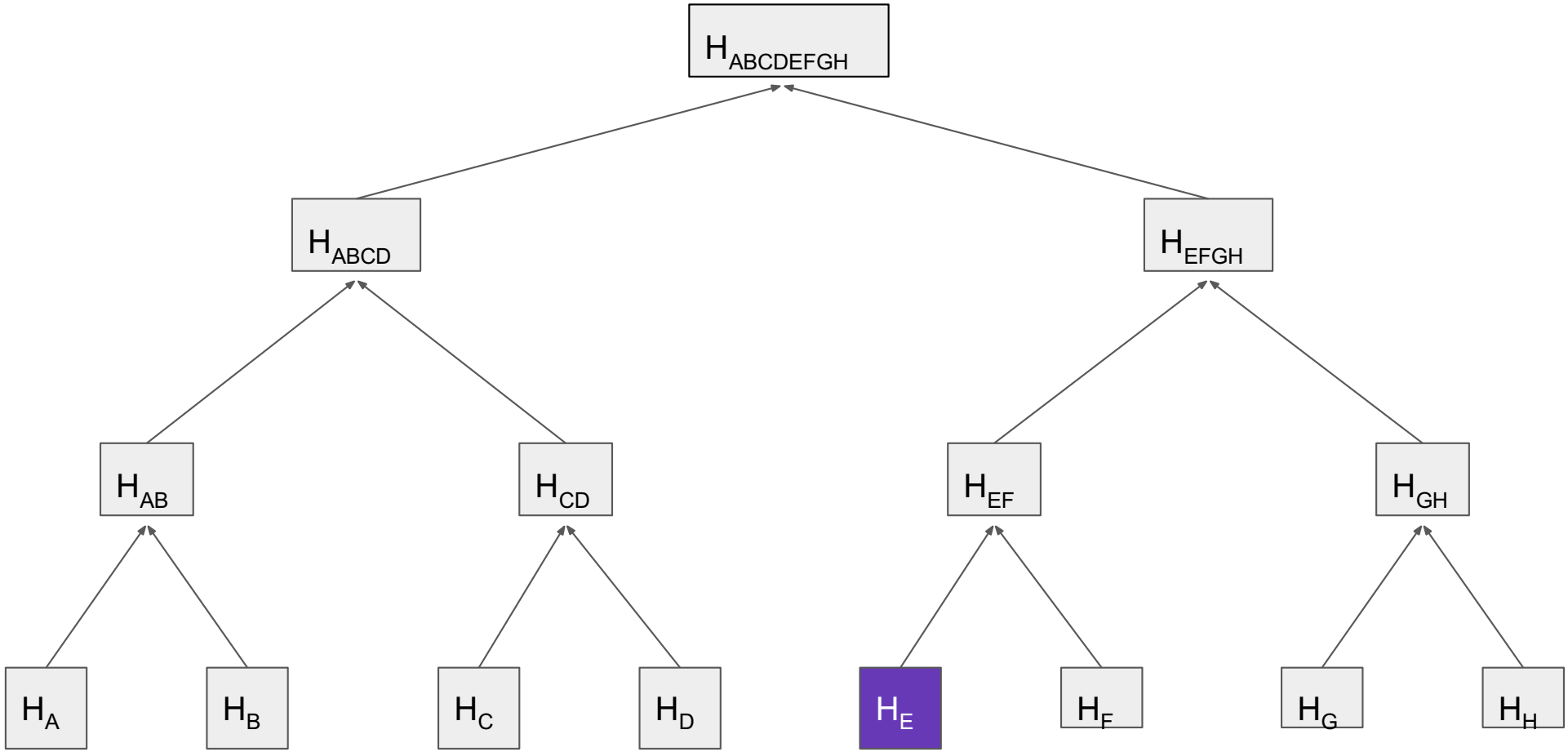
::.. Hash Tree Authentication (4/5)

Merkle tree: proof of inclusion



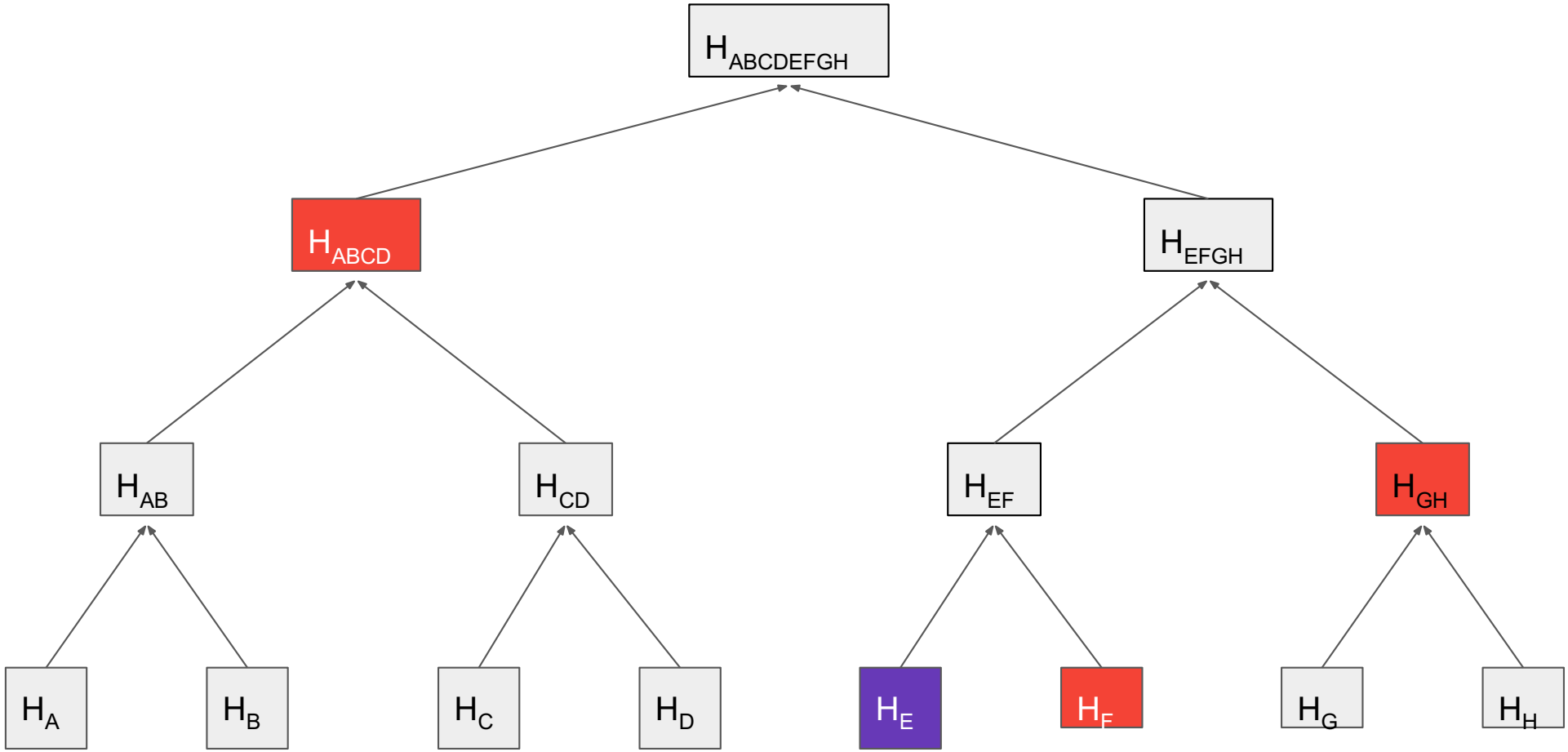
::.. Hash Tree Authentication (4/5)

Merkle tree: proof of inclusion



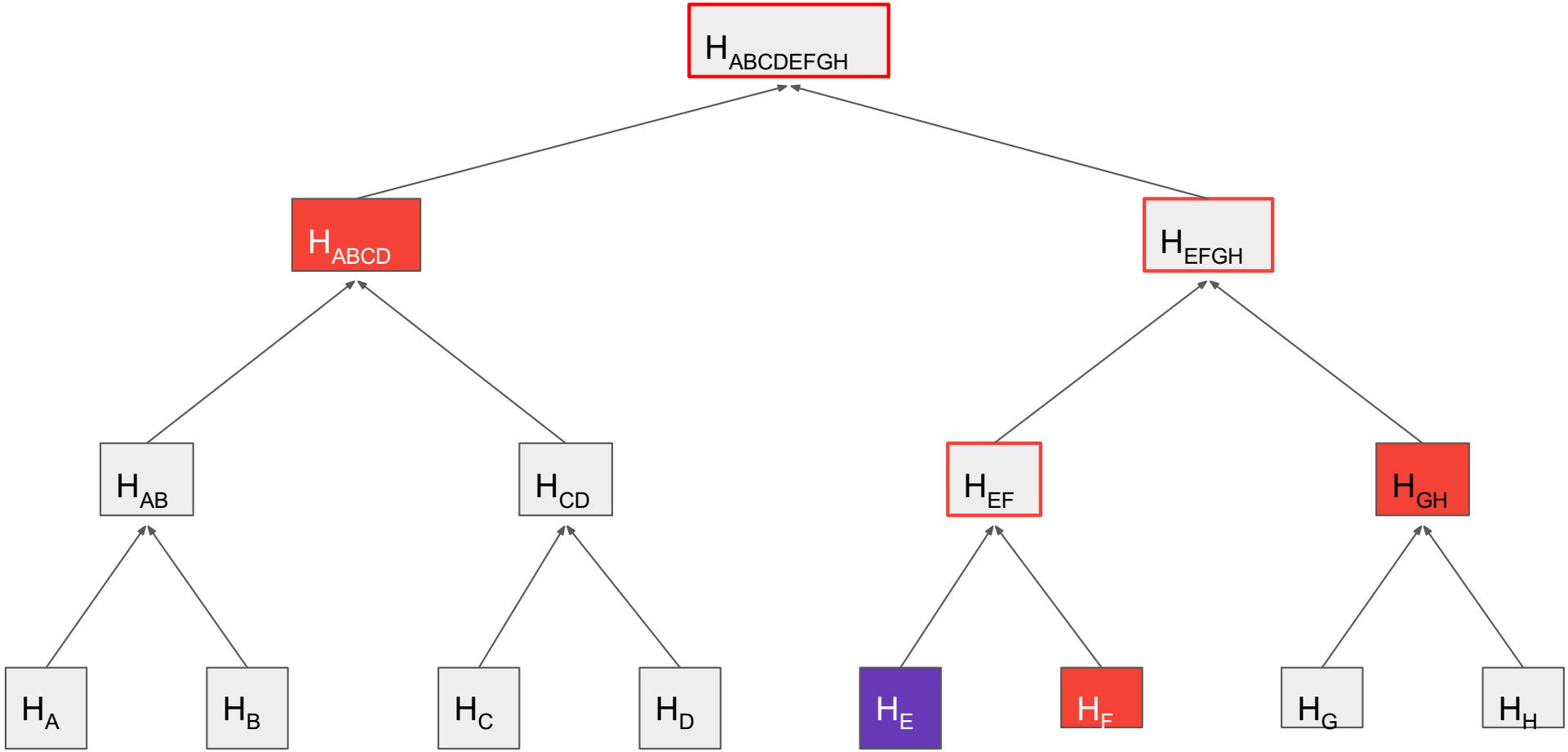
::.. Hash Tree Authentication (4/5)

Merkle tree: proof of inclusion



::.. Hash Tree Authentication (4/5)

Merkle tree: proof of inclusion



::.. Hash Tree Authentication (5/5)

File segmented in chuncks – Solution with Merkle Trees:

- The Prover sends a chunck;
- The Prover sends the neighbors along the path that connects the leaf to the MTR;
- The Verifier calculates the hashes along the path that connects the leaf to the MTR;
- The Verifier checks which computed root corresponds to MTR.
- How big is the inclusion test?

$$|\pi| \in \Theta(\lg|D|)$$

If the opponent can present inclusion proof for a wrong leaf node, then we can compromise the hash function.

::.. Node Identity

- Although applied to problems where you want to do without trusted entities, blockchain technologies also use digital signature (and trusted certification authorities) for security problems typical of distributed systems:
  - Authentication: authentication of the network nodes;
  - Integrity: verification of violations of the integrity of the messages exchanged;
  - Non-repudiation: Non-repudiation of sent messages by nodes.

::.. Transaction Validity

- Consider a node that wants to "sell" an asset through a transaction.
- The node prepares the transaction by itself, which it must submit to the other nodes so that it is approved (validated) by a majority.

- It is possible that the node tries to concurrently "spend" the same good twice (double spending):
  - In a conscious way: byzantine behavior
  - Unconsciously: network malfunctions (packet duplication), replay attack.

::.. Nonce

- A first step in solving the double spending problem is to determine if a message is "fresh" or is a re-transmitted copy.
- It is necessary to be able to uniquely identify a message containing a block of executed transactions.
- A nonce is defined as "a time-varying value that has at most a negligible chance of repeating, e.g., a random value that is generated anew for each use, a timestamp, a sequence number, or some combination of these." [NIST SP-800-90].

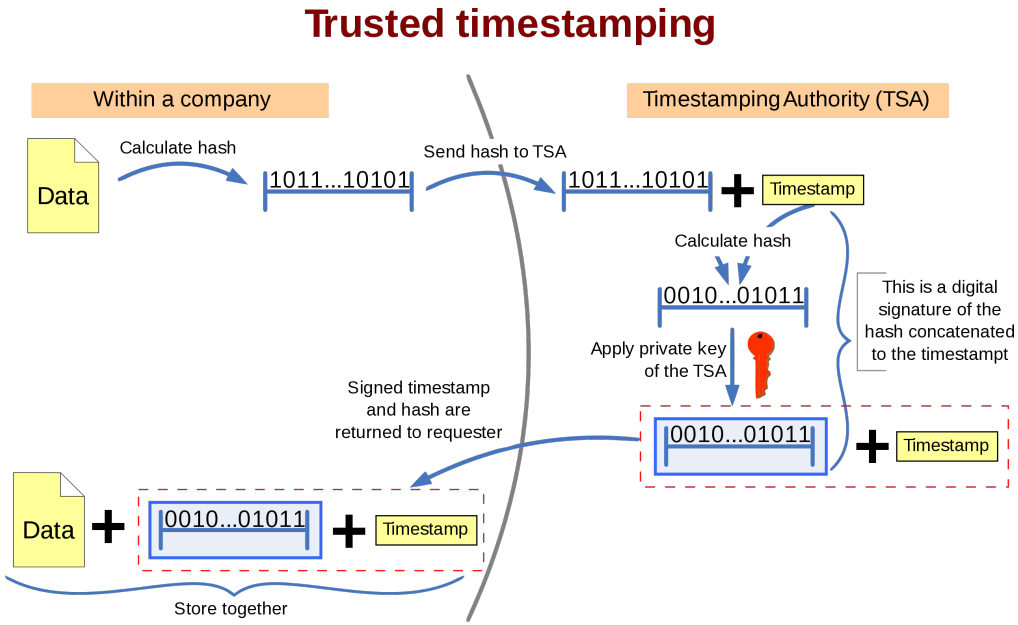
- A nonce can then be obtained through a "timestamp":
  - To ensure that each block has a different value, this value must be generated from a "trusted" source.

::.. Time Certification (1/4)

- Time certification is used to verify when a document was created, or when the last modification was made in a reliable and non-falsifiable way.
- A Trusted Time Server (TTS) is used:
  - First Solution:
    - The user sends a copy of the document to TTS
    - TTS adds date and time, and keeps it in the deposit box.
  - Second solution - A cryptographic solution is used to protect the privacy of the document, and reduce transmission and storage costs:
    - The user sends the hash of the document to the TTS
    - The TTS adds date, time and signature and sends this certificate to the user.

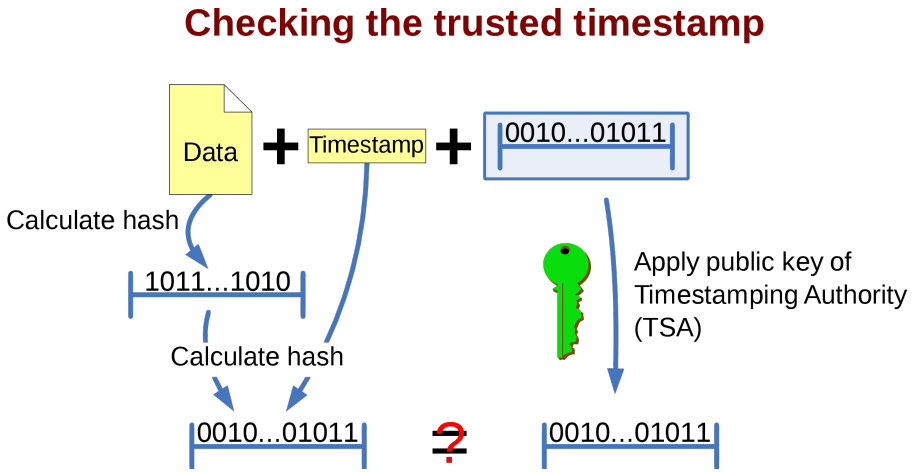
::.. Time Certification (2/4)

- The ANSI ASC X9.95 Standard indicates that a trusted timestamp is issued by a a Time Stamping Authority (TSA).



A hash is calculated and sent to the TSA. The TSA concatenates a timestamp to the hash and calculates a hash, which is digitally signed with the private key of the TSA. This signed hash and timestamp are returned.

Anyone trusting the timestamper can then verify that the data was not created after the date that the timestamper vouches.

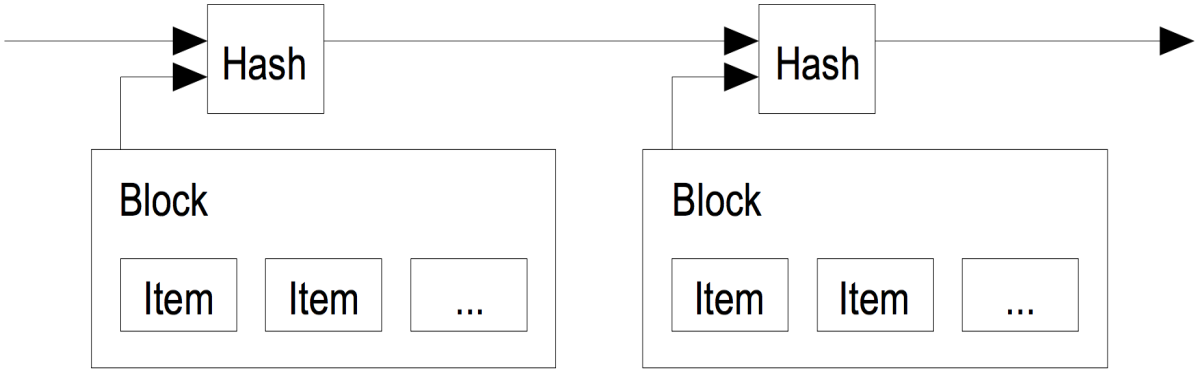


::.. Time Certification (3/4)

- Regardless of the correctness of the certified time, we want two documents sequentially submitted to the TTS to have a nonce that expresses this sequentiality:
  - Each certificate issued contains the hash of the document, date, time, TSS signature and the hash of the previous certificate (or part of it).
  - Collectively, the certificates form a chain of blocks.
  - It is inadmissible to subsequently insert a certificate within the chain, the TTS should detect collisions for the hash function.

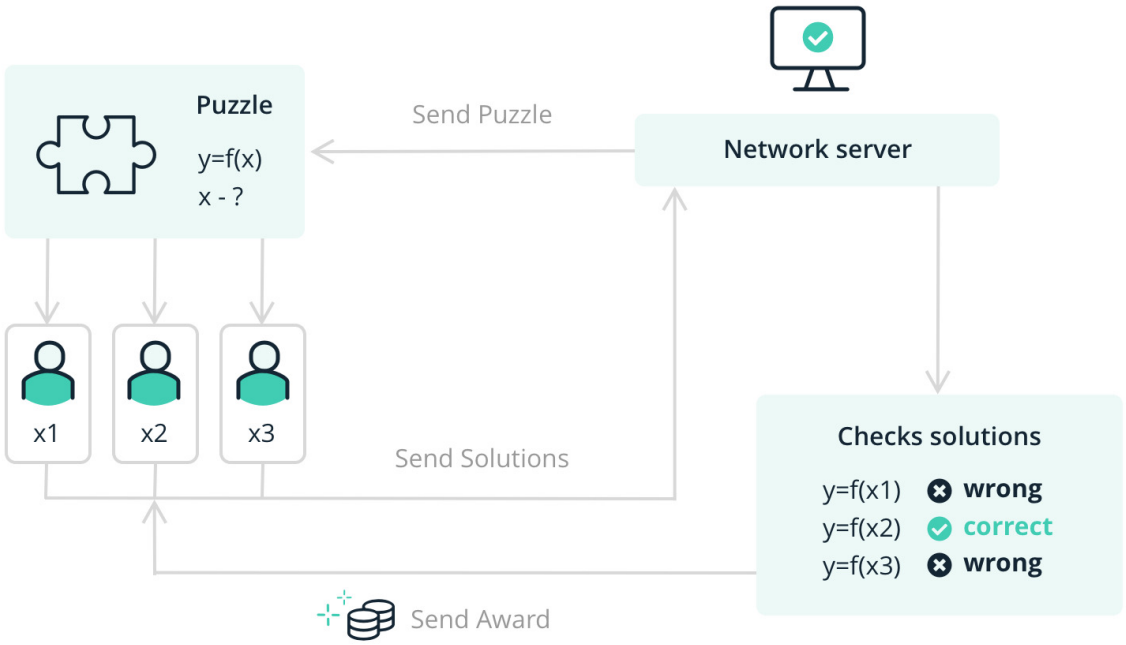
::.. Time Certification (3/4)

- This solution has a number of advantages:
  - Adding the timestamp makes the hash more resistant to collisions;
  - It solves the problem of double spending by assigning a temporal order to transactions and publishing them.



::.. Proof-of-Work (1/3)

- To free themselves from the use of a timestamp server, some blockchain solutions exploit a different mechanism for generating nonce.



- This mechanism is called Proof-of-Work (PoW).

::.. Proof-of-Work (2/3)

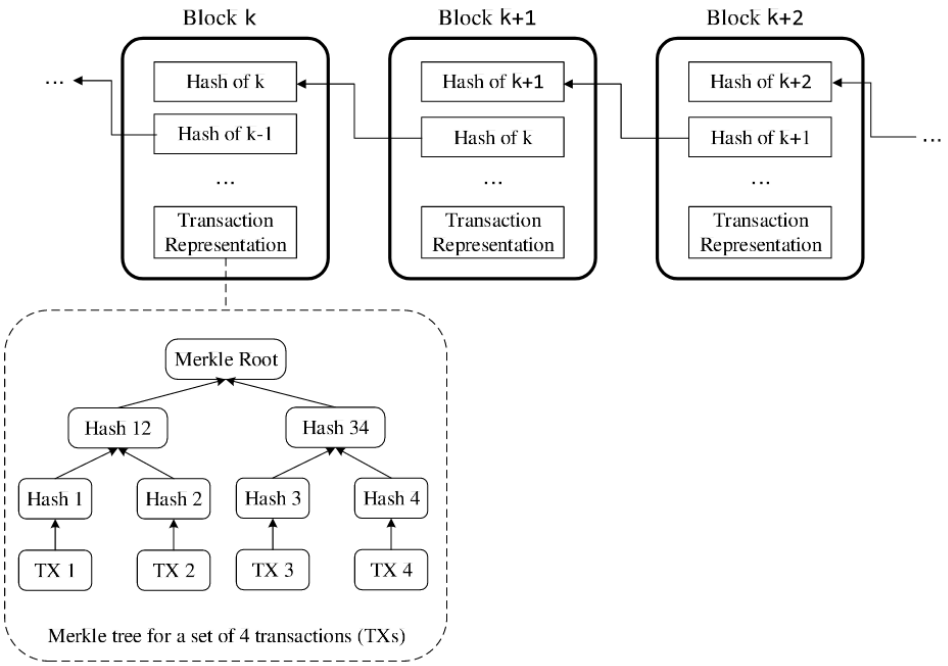
- Proof-of-Work: to propose a new block to be added to the ledger, a node must demonstrate that it has used enough computational resources to solve a mathematical puzzle, the result of which is used inserted in the block itself.
- Proof-of-Work involves computation of a hash value with a given number of initial bits equal to zero.

$$H(ctr \,||\, \mathbf{x} \,||\, s) <= T$$

- Each node (miner) tries to find a nonce that satisfies the constraint on the number of zeros produced by the hash function.

::.. Proof-of-Work (3/3)

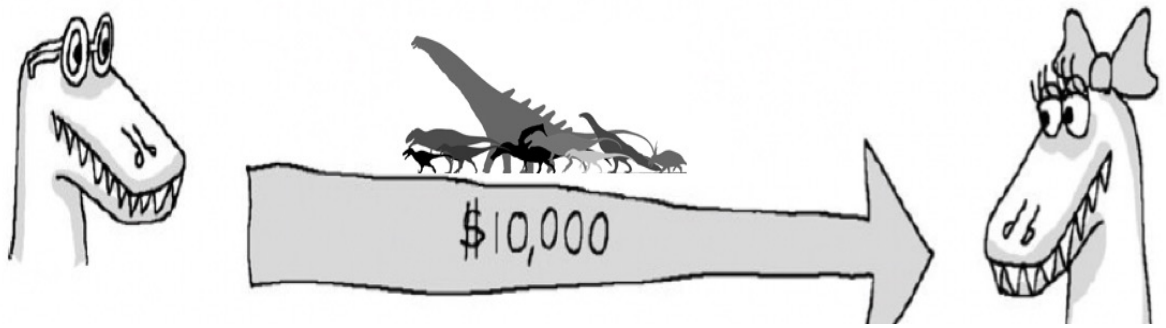
- The average work required is exponential with respect to the number of zero bits required and can be verified by performing a single hash.
- This operation is very expensive from a computational point of view and requires the collaboration of several CPUs to be able to be carried out.



- The 51% attack becomes very onerous

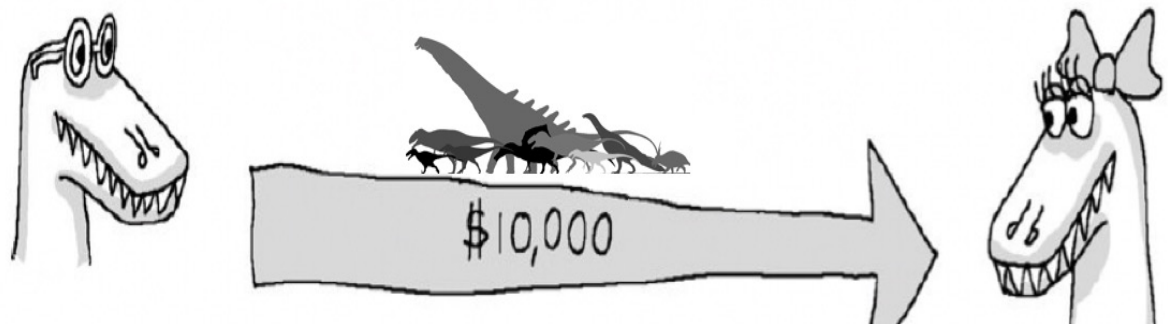
::.. Transactions with Distr. Agree. (1/3)

- All nodes of the network are aware of all transactions and actively participate in the validation of blocks by reaching consensus.
- The security model must also include the so-called "51% Attack": a group of malicious nodes possesses more than 50% of the computational power of the network.



:.. Transactions with Distr. Agree. (2/3)

- "No algorithm can guarantee the achievement of consensus in an asynchronous system in the case of even a single failure due to a crash of a process" (Impossibility Theorem).
- To reach consensus in asynchronous systems in the presence of failures you can:
  - Relax the constraints of agreement;
  - Make the system less "asynchronous"
    - Take advantage of periods of synchrony.



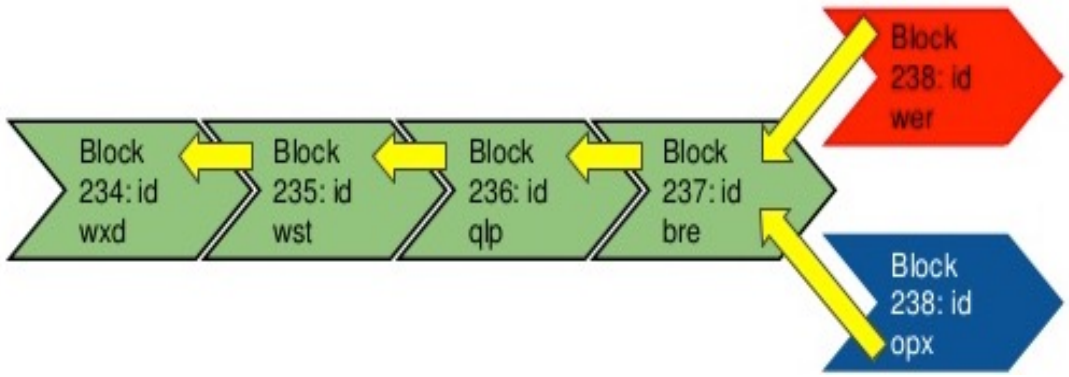
::.. Transactions with Distr. Agree. (3/3)

Relax the Constraints of Consent

- It is permissible in some cases that consensus is not reached among all nodes in the network, i.e. only a subset of nodes reach consensus.
- "51% Attack" is possible.
- There are many different solutions that can help strengthen consensus to avoid this type of attack.
  - Adding a "collective control": Anyone who validates a transaction must spend computational time (How much? Enough to discourage the attack)
  - There is an incentive mechanism: a reward for the work done.

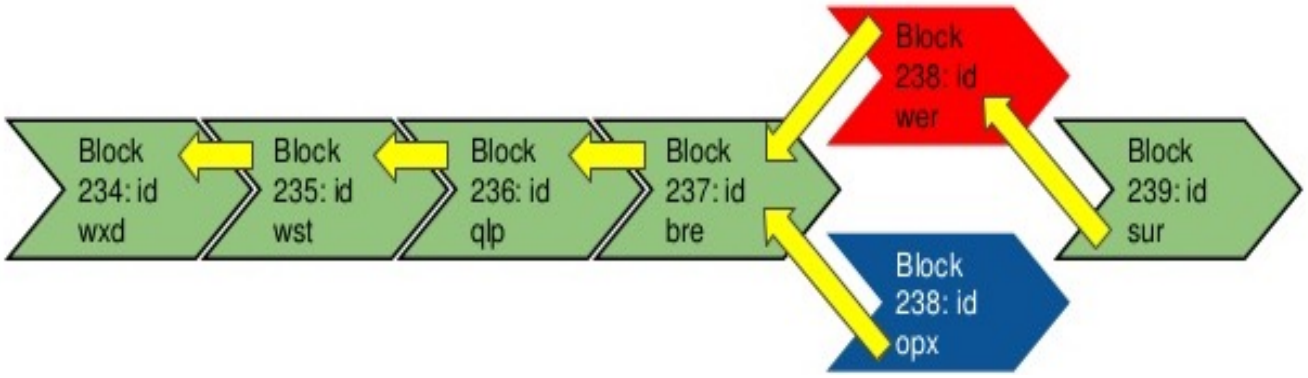
::.. Blockchain Forking (1/3)

- The relaxation of consensus constraints can give rise to bifurcations of the blockchain.
- In the case of "simultaneous" production of two blocks, some blocks will add the red block to their blockchain and others the blue block, until a "forked blockchain" is created.



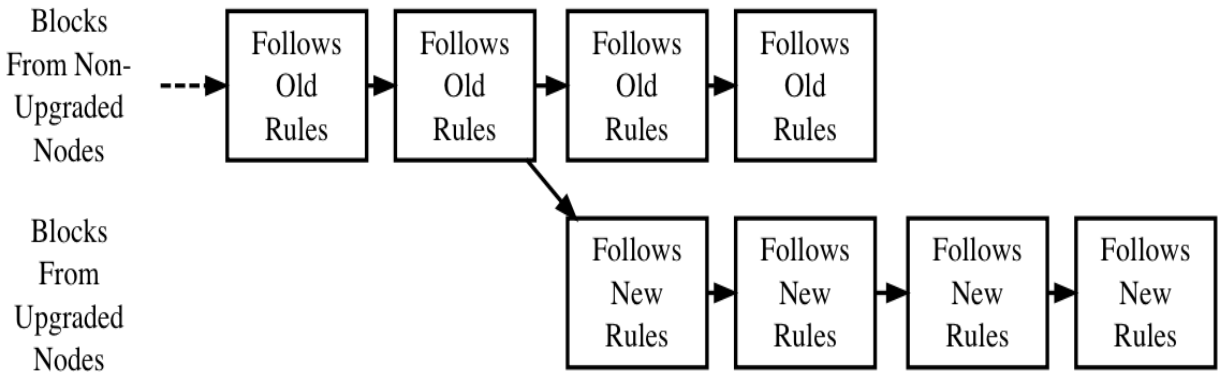
::.. Blockchain Forking (2/3)

- When producing the next block, it will refer to only one of the two previously created blocks.
- This operation involves removing the branch of the blockchain consisting of the blue node.



::.. Blockchain Forking (3/3)

- In a certain instant of time the community may deem it necessary to change the rules of the blockchain so that certain transactions deemed "invalid" up to that instant are considered "valid" starting from that instant and in the future
- This condition generates a hard fork, which involves the "desired" generation of a new branch of the blockchain to distinguish the nodes that follow the new regulation from those faithful to the old one.
- At any time, nodes can decide to leave one branch to follow the other.



::.. Nakamoto Consensus Algo. (1/4)

1. Each new transaction is broadcast to all nodes.
2. Each node collects new transactions in a block.
3. Each node looks for a "difficult" PoW for its block:

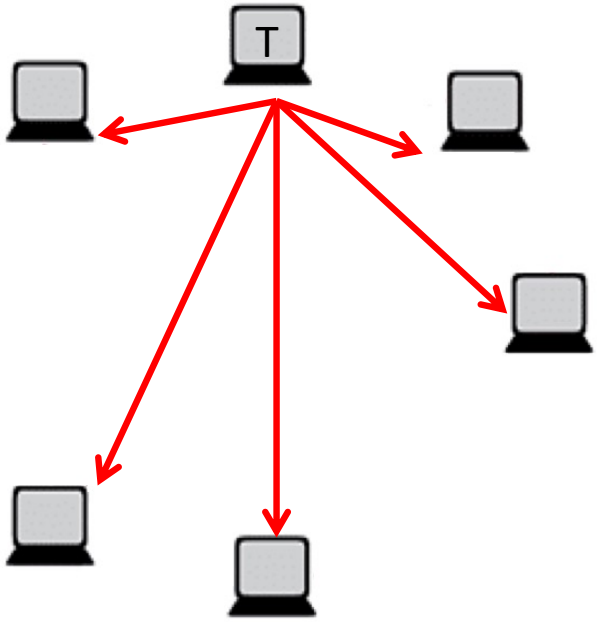
   PoW is the computation of a value (also a nonce) such that a block hash + nonce starts with an established number of zeros.
4. When a node finds a PoW, it broadcasts the block containing the transactions and the PoW to all nodes.
5. A node accepts the block only if:
   - All transactions are valid;
   - Transactions are not related to an asset already spent;
   - PoW is valid. For this purpose, the node calculates the hash of the block in turn, and verifies that it has the same number of leading zeros.
6. The nodes show acceptance of the block by adding it to the blockchain when creating the next block, using the hash of the accepted block to create the next block.

::.. Nakamoto Consensus Algo. (2/4)

- Two nodes can concurrently broadcast different blocks (step 4).
- Different nodes can receive the two broadcasts in different order.
- Each node works on the first block it receives, but keeps the other branch.
- The indecision is resolved when a branch becomes longer:
  - The nodes that work on the shortest branch leave it and continue to work on the longest chain.

::.. Nakamoto Consensus Algo. (3/4)

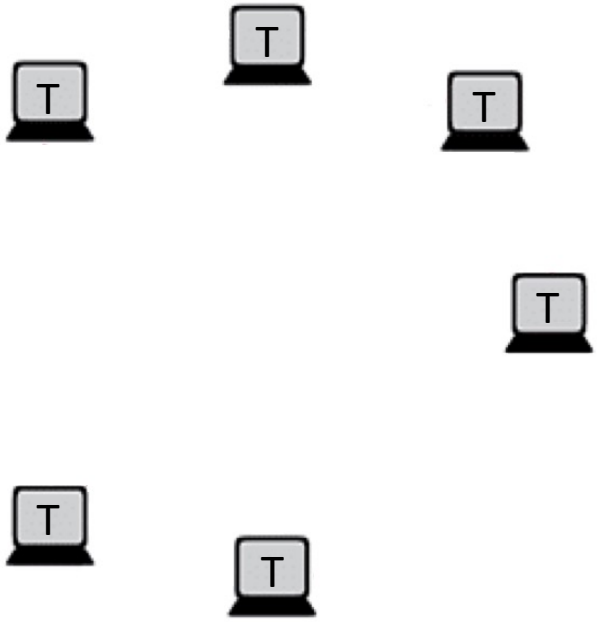
1. A node generates a new transaction T and broadcasts it to all nodes.



Current Blockchain

::.. Nakamoto Consensus Algo. (3/4)

1. A node generates a new transaction T and broadcasts it to all nodes.

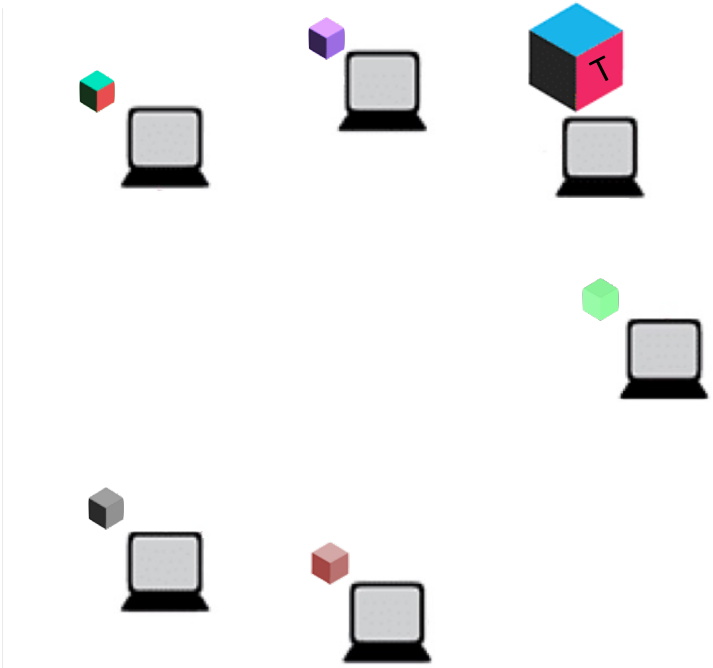


The other nodes receive T.

Current Blockchain

::.. Nakamoto Consensus Algo. (3/4)

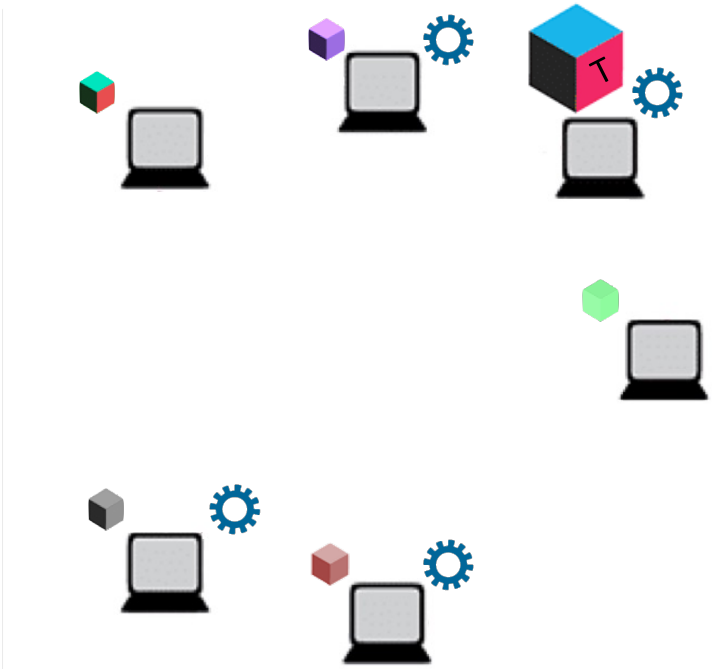
2. Each node collects new transactions in its own block.



Current Blockchain

::.. Nakamoto Consensus Algo. (3/4)

3. Nodes look for a Proof-of-Work for their block.

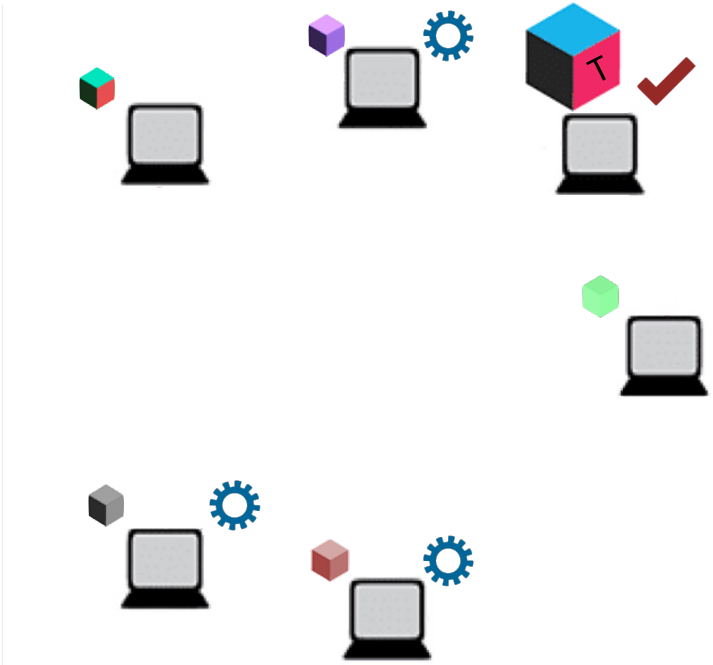


Calculation of the PoW

N.B .: The blocks on which different nodes calculate the PoW are generally different.

::.. Nakamoto Consensus Algo. (3/4)

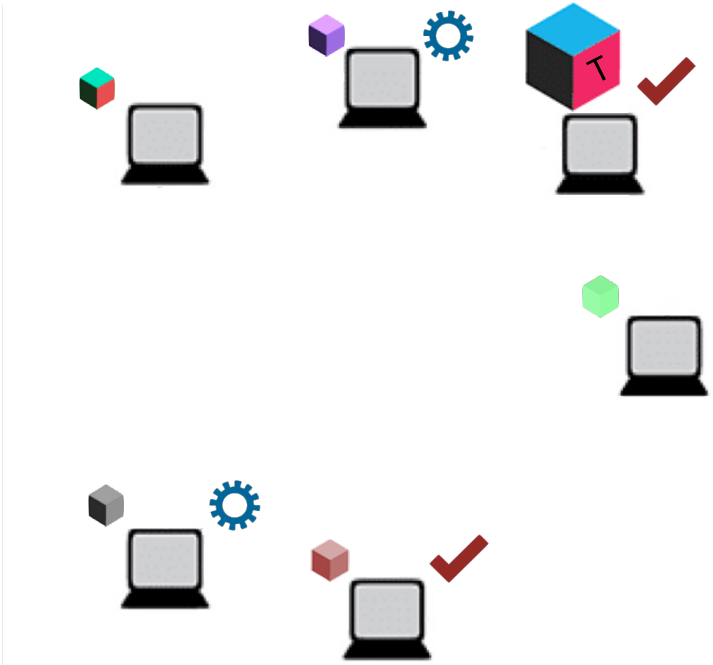
3. Nodes look for a Proof-of-Work for their block.



One node completes the PoW.

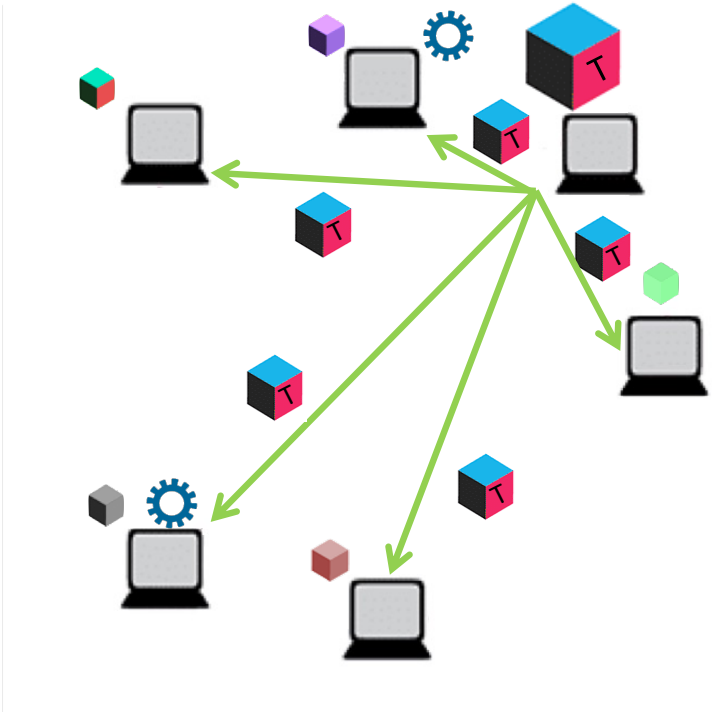
::.. Nakamoto Consensus Algo. (3/4)

3. Nodes look for a Proof-of-Work for their block.



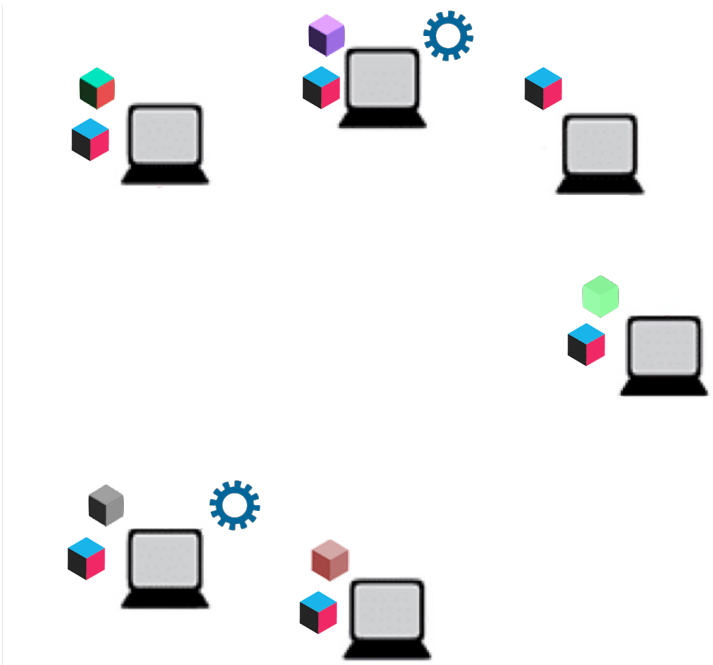
Another node complete the PoW

::.. Nakamoto Consensus Algo. (3/4)



4. When a node finds a Proof-of-Work, it broadcasts the block to all nodes.

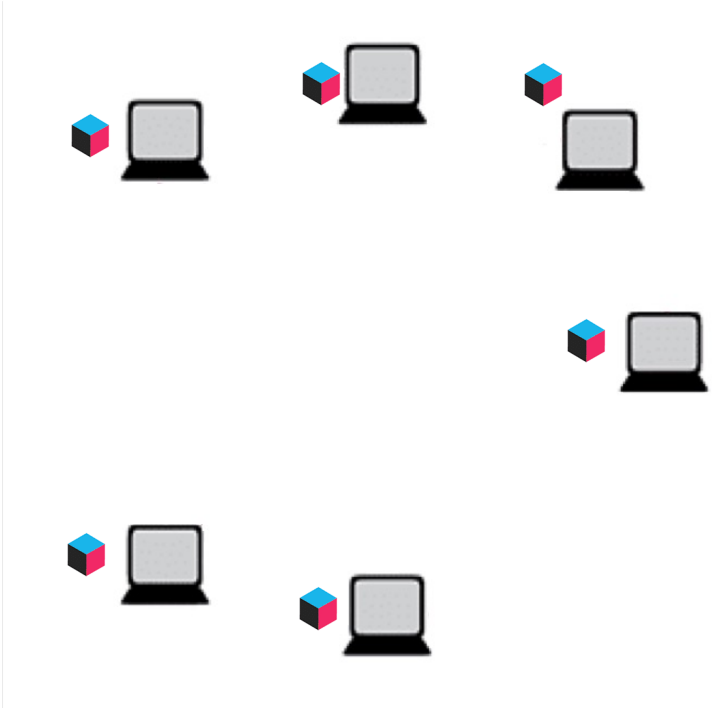
::.. Nakamoto Consensus Algo. (3/4)



4. When a node finds a Proof-of-Work, it broadcasts the block to all nodes.

The nodes receive the lock.

::.. Nakamoto Consensus Algo. (3/4)

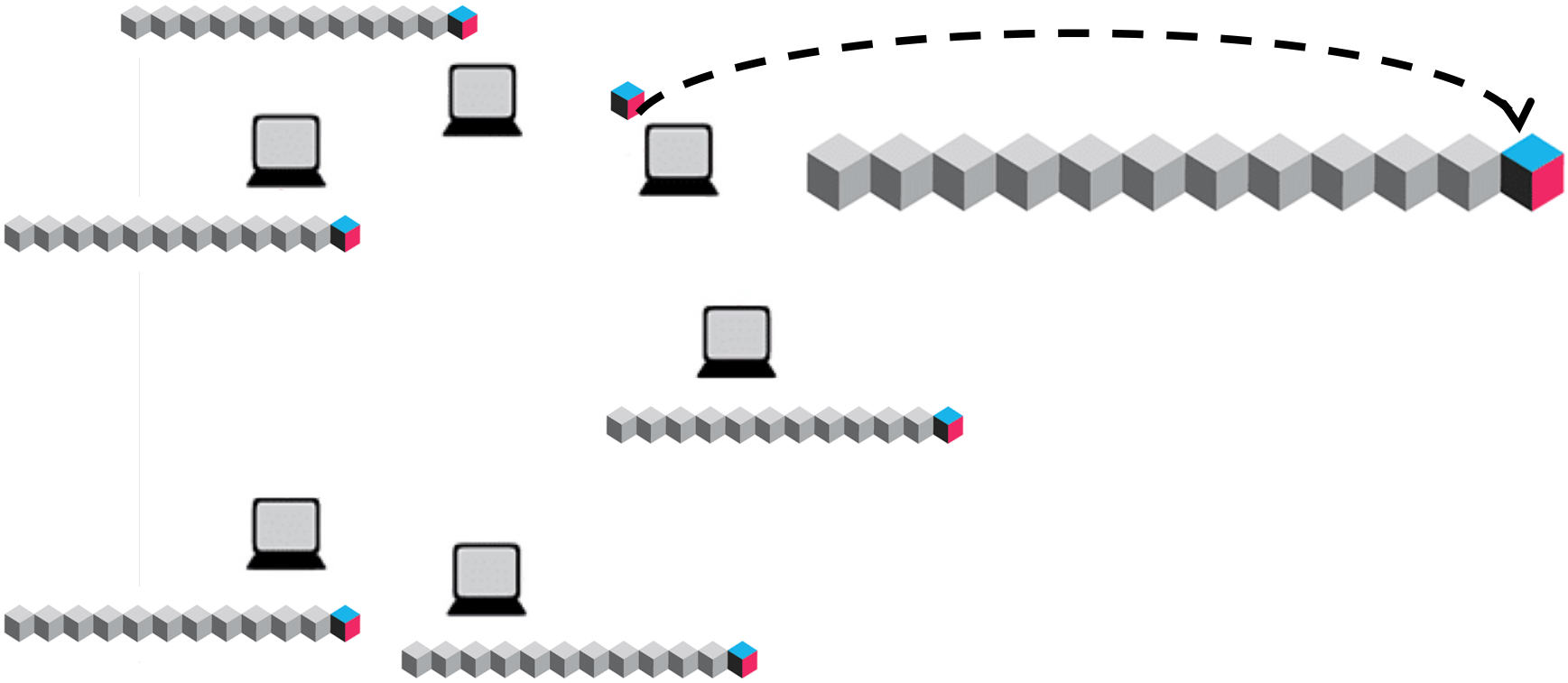


All nodes accept the block
(the blockchain is still unaltered).

5. A node accepts the lock only if:
   - All transactions are valid;
   - The transactions are not related to an asset already spent;
   - The PoW is valid. For this purpose, the node calculates the hash of the block in turn, and verifies that it has the same number of leading zeros.

::.. Nakamoto Consensus Algo. (3/4)

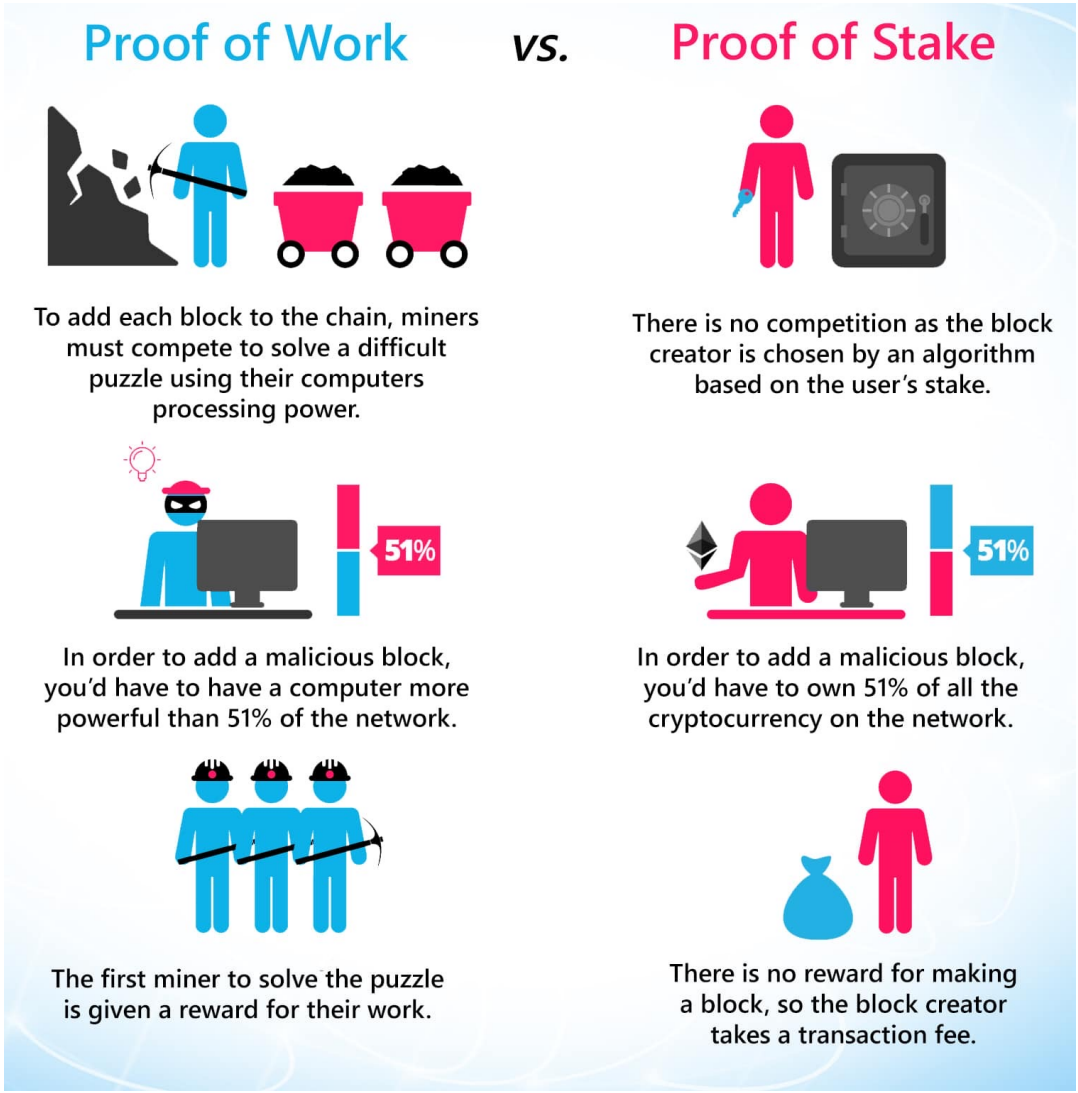
6. The nodes show acceptance of the block by adding it to the (local copy of the) blockchain when creating the next block, using the accepted block hash to create the next block.



::.. Nakamoto Consensus Algo. (4/4)

- Cooperation between nodes in the creation of consensus should be encouraged (validation of "correct" transactions):
  - for a block to be validated there must be a majority of correct nodes that spend resources (building the PoW);
  - since PoW is expensive, even the right nodes may not be willing to cooperate.
- There is an incentive policy:
  - encourage correct processes to collaborate despite the computational burden, putting malicious processes in the minority;
  - the malicious nodes will have to have even more computing power to “counter” many correct nodes that validate the blocks.
- The first transaction of each block assigns compensation to the block creator node.

::.. Proof-of-Stake (1/4)



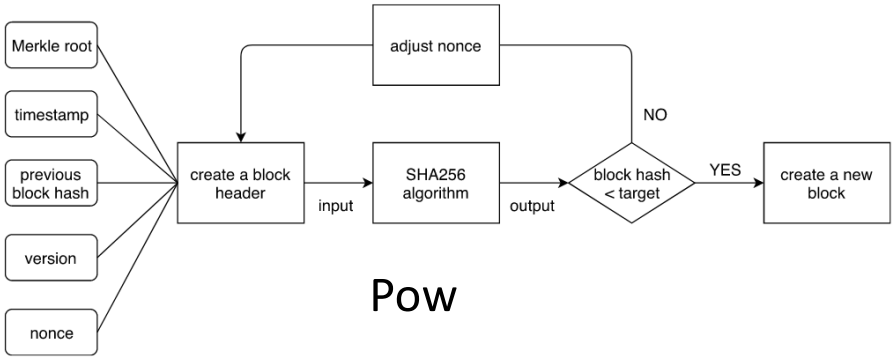
Proof-of-Stake (PoS) is an energy-efficient alternative to PoW.

A stake refers to the coins or tokens owned by a participant that can be invested in the consensus process.

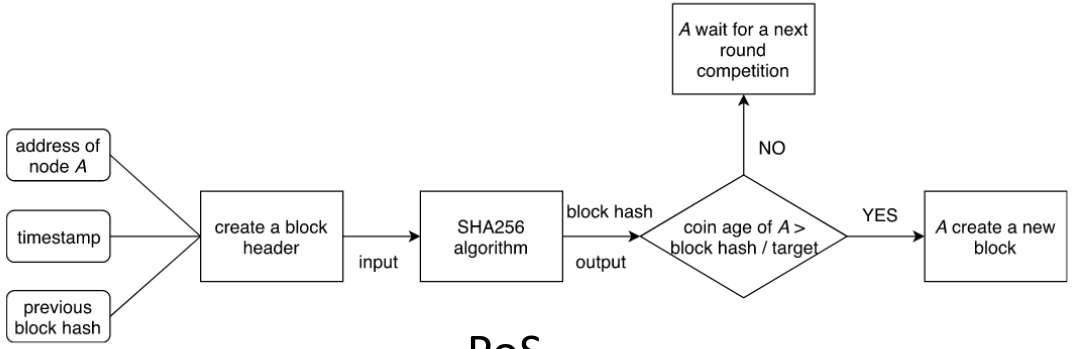
Compared to PoW whose possibility of proposing a block is proportional to its computing power, the possibility of proposing a block for PoS is proportional to the value of its stake.

::.. Proof-of-Stake (2/4)

PoS does not rely on expensive hashing to generate blocks. Since the difficulty of the hashing puzzle decreases with the value of the minter stake, the expected number of hashing attempts for a minter to solve the puzzle can be significantly reduced if its stake value is high. Therefore, PoS avoids the brute-force hashing competition that would occur if PoW were used thus achieving a significant reduction in power consumption.



Pow

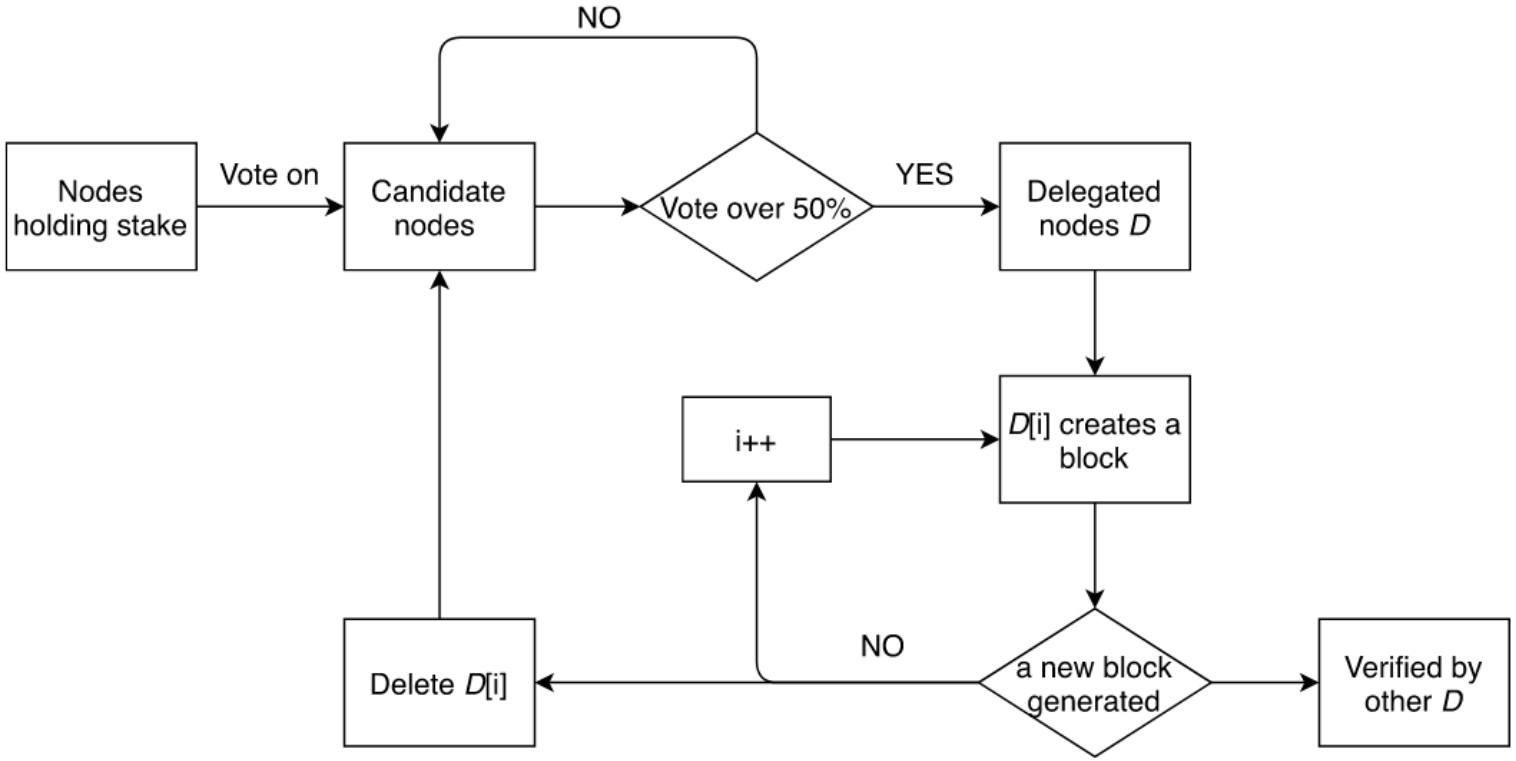


PoS

This implies the absence of the reward for those who insert the right block in the blockchain, and of mining, as no new cryptocurrency units are created with the creation of each block. Validators are rewarded with a commission for validated transactions.

::.. Proof-of-Stake (3/4)

Proof of Stake Delegated (DPoS): Allows nodes holding the largest stake to vote to elect block verifiers. This causes stake holders to grant the right to create blocks to the delegates they claim instead of creating blocks themselves, thereby reducing their computational power consumption to 0.



::.. Proof-of-Stake (4/4)

PoW benefits miners who have invested the most in hardware, PoS gives disproportionate influence to those who own a large number of cryptocurrencies, with the risk of centralization of wealth in the hands of a few, following the dogma according to which "Rich people get richer" .

Another problem is “nothing at stake” for which in the case of a network fork the validators will be incentivized to operate on both chains, possibly resulting in double-spending problems - this problem is less evident in a DPoS system.

PoS and DPoS have been applied in the context of classic BFT algorithms, such as PBFT, and in order to allow their application in open and permissionless contexts.

::.. Blockchain (1/5)

- A blockchain is a distributed public ledger of digital transactions or events executed and shared among participants.
- Each transaction reported in the blockchain is validated by reaching consensus between the system nodes.
- Once stored, the information cannot be deleted or modified.
- Each block contains one or more records (one record per transaction).
- As in a ledger, the history of the records is immutable and can be verified starting from the first block (genesis block).

::.. Blockchain (2/5)

- The properties that a Blockchain provides are:
  - Public Verifiability: Each transaction can be verified by each participant;
  - Transparency: each participant has access to a subset of information;
  - Privacy: the identity of the person carrying out a specific transaction must be protected;
  - Integrity: Information is not modified by unauthorized sources;
  - Redundancy: repeated data for each participant in the system;
  - Absence of a "Trust Anchor".

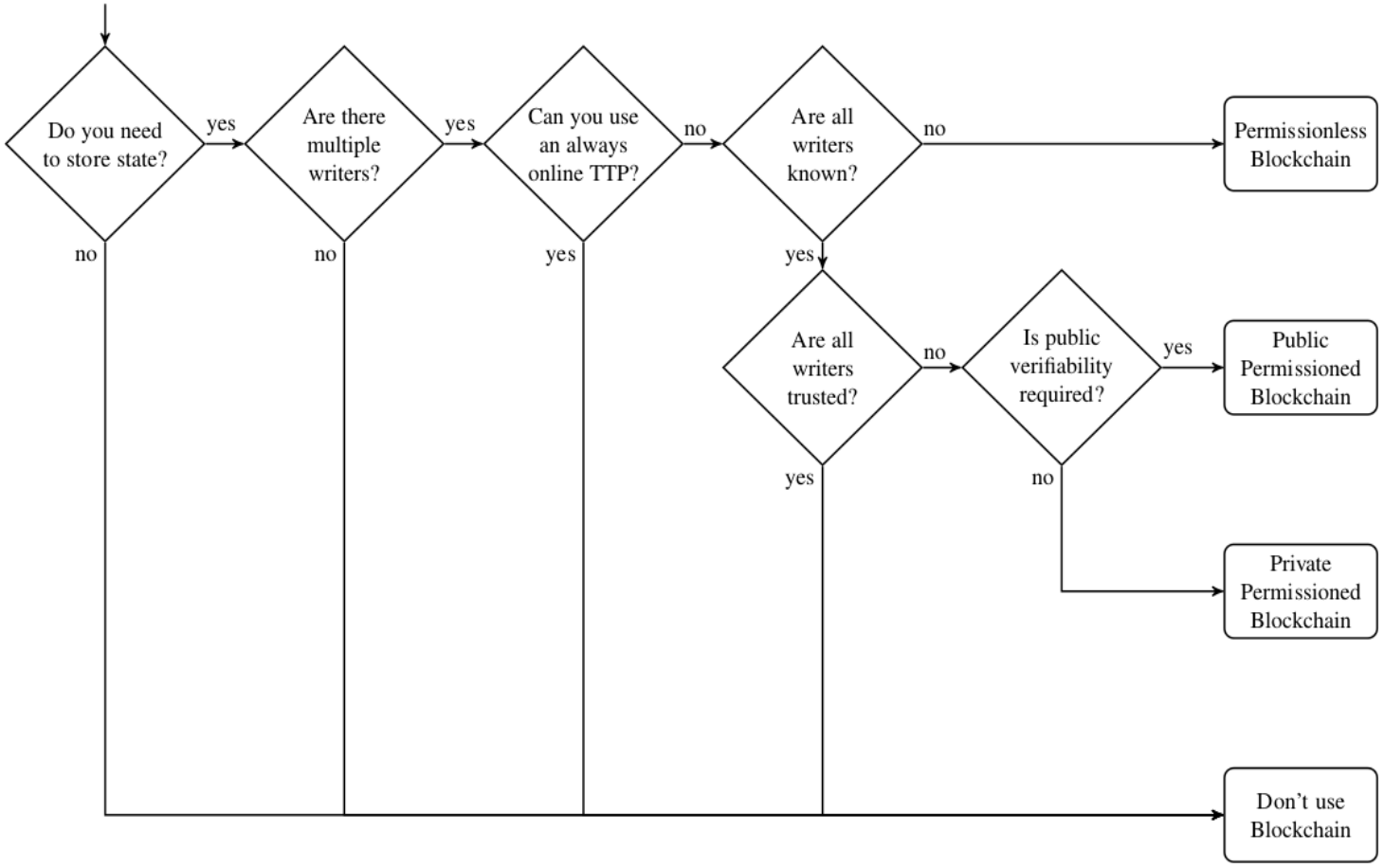
::.. Blockchain (3/5)

- The actual implementation of this technology is highly dependent on the type of Blockchain to be created:
  - Permissionless: participants do not have to be previously "authorized" for the role they intend to play;
  - Permissioned: operations (all or even only some) can only be carried out by previously authorized nodes.

::.. Blockchain (4/5)

- They can be classified according to the area:
  - Public: All blocks are visible to all nodes and each node can participate in consensus;
  - Private: a specific organization decides which nodes can read blocks and participate in consensus (very high transaction throughput);
  - Consortium: a few predetermined nodes can read the blocks and participate in the consensus.

::.. Blockchain (5/5)



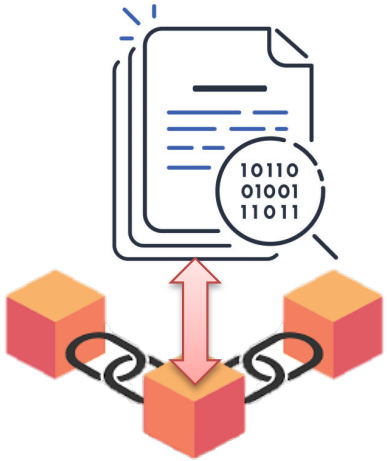
*TTP: Trusted Third Party

Smart Contracts

::.. Smart Contract (1/5)

A contract is a legally binding agreement that recognizes and governs the rights and obligations of the parties to the contract.

A smart contract is the "transposition" of a contract into code, using "if / then" functions incorporated in software or IT protocols, to automatically verify the fulfillment of certain conditions and to self-execute actions when the conditions are met and verified.



It is a deterministic program that processes information in a blockchain: with the same input, the results returned will be identical. This guarantees the parties an absolute "certainty of objective judgment" excluding any form of interpretation.

::.. Smart Contract (2/5)

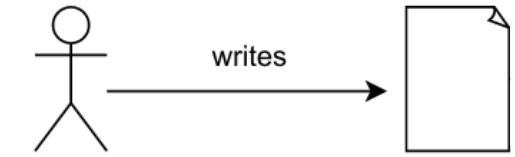
A blockchain smart contract must meet the following objectives: observability, verifiability, confidentiality and applicability. On a blockchain, a smart contract cannot be changed, but it can be easily observed, verified, self-enforced, and depending on how the blockchain is accessed, privacy can be achieved.

Here is a progressive lifecycle of a smart contract:

::.. Smart Contract (2/5)

A blockchain smart contract must meet the following objectives: observability, verifiability, confidentiality and applicability. On a blockchain, a smart contract cannot be changed, but it can be easily

Step 1: Developers write the logic for the smart contract in a programming language supported by the blockchain platform they want to use. Using a specific compiler (usually provided by the blockchain platform), they compile the source code of their smart contract and get its representation in bytecode.

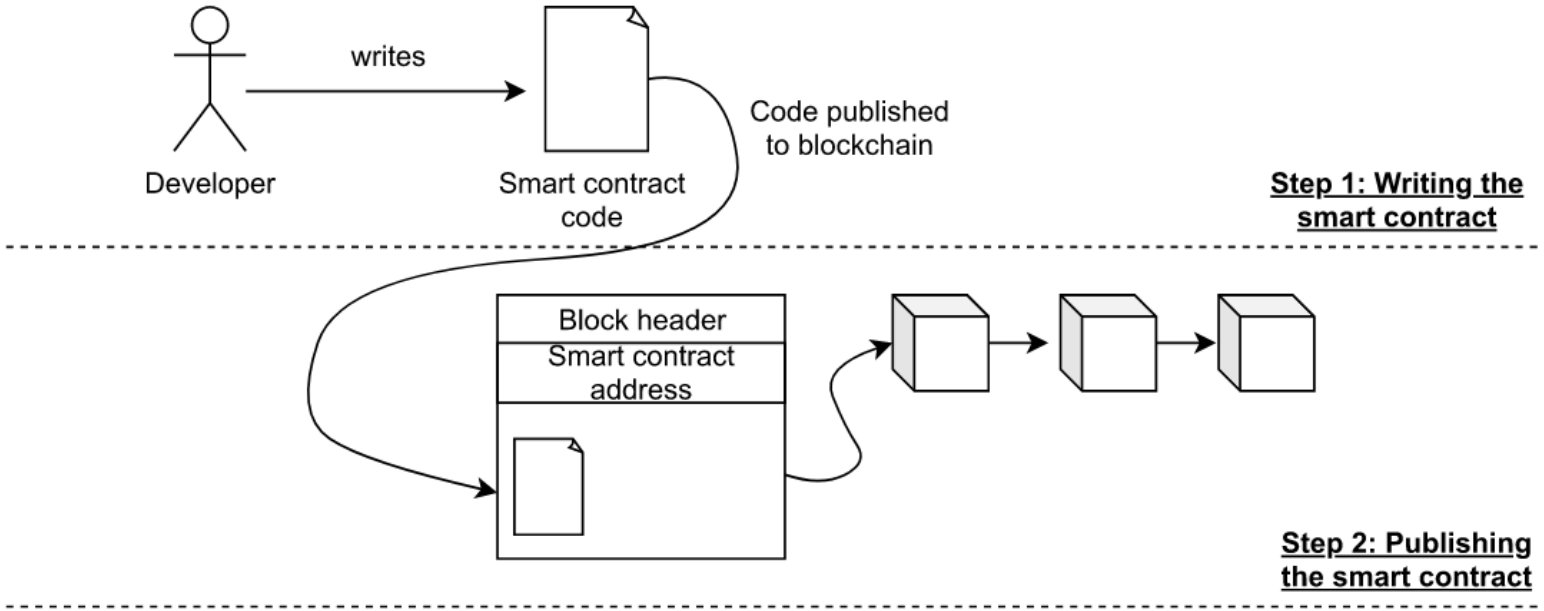


Step 1: Writing the smart contract

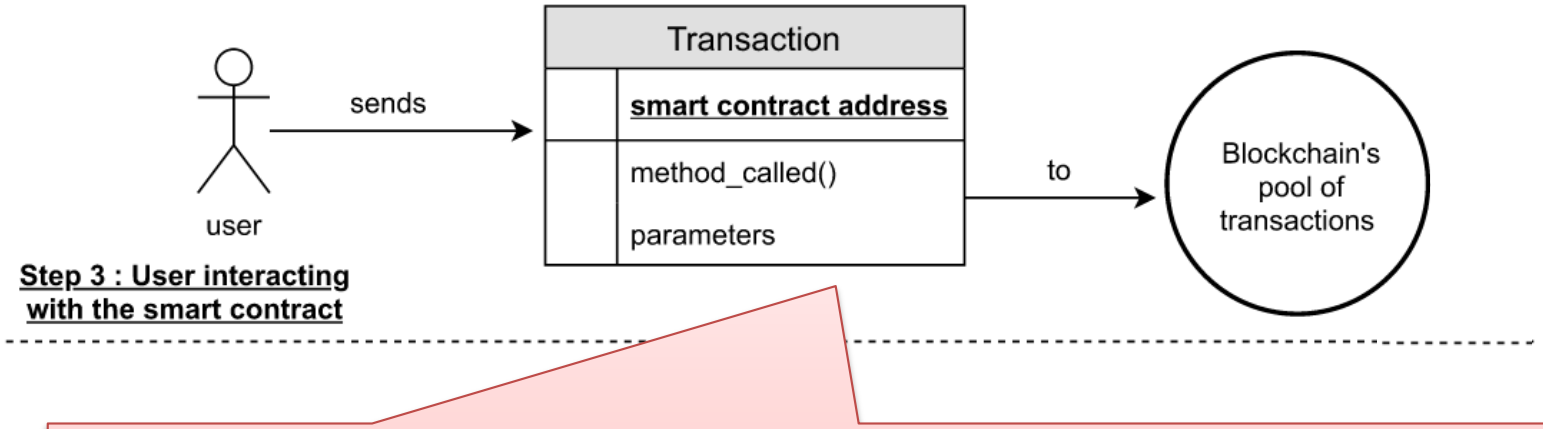
::.. Smart Contract (2/5)

A blockchain smart contract must meet the following objectives: observability, verifiability, confidentiality and applicability. On a

Step 2: after obtaining the byte code, the smart contract is published on the blockchain platform and archived. Once the smart contract is published, it will be read-only or editable. In case it is read-only, to provide an update, developers will need to publish a new version of the smart contract and redirect users to it. Once loaded, the smart contract is in its initial state, equal to the initial values of the internal variables.



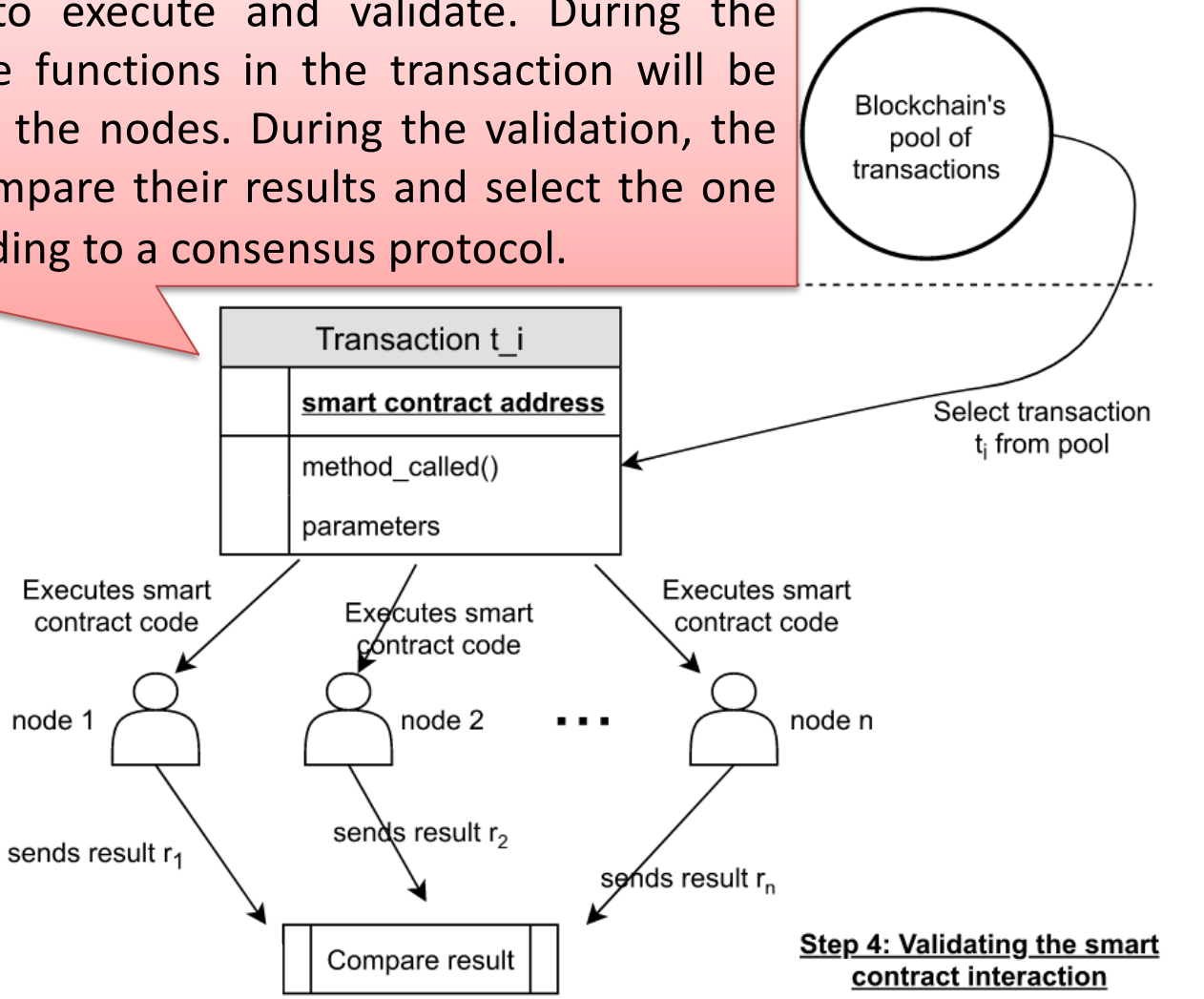
::.. Smart Contract (3/5)



Step 3: Access to a published smart contract program depends on the blockchain platform, such as an address returned when the smart contract is loaded into the platform. This address can be used to interact with the smart contract, sending transactions containing the function they want to use and the arguments of the function. If an amount of platform currency is required to initiate function execution, that amount will be transferred along with the transaction. The transaction will be stored in the blockchain platform's transaction pool waiting to be executed and validated.

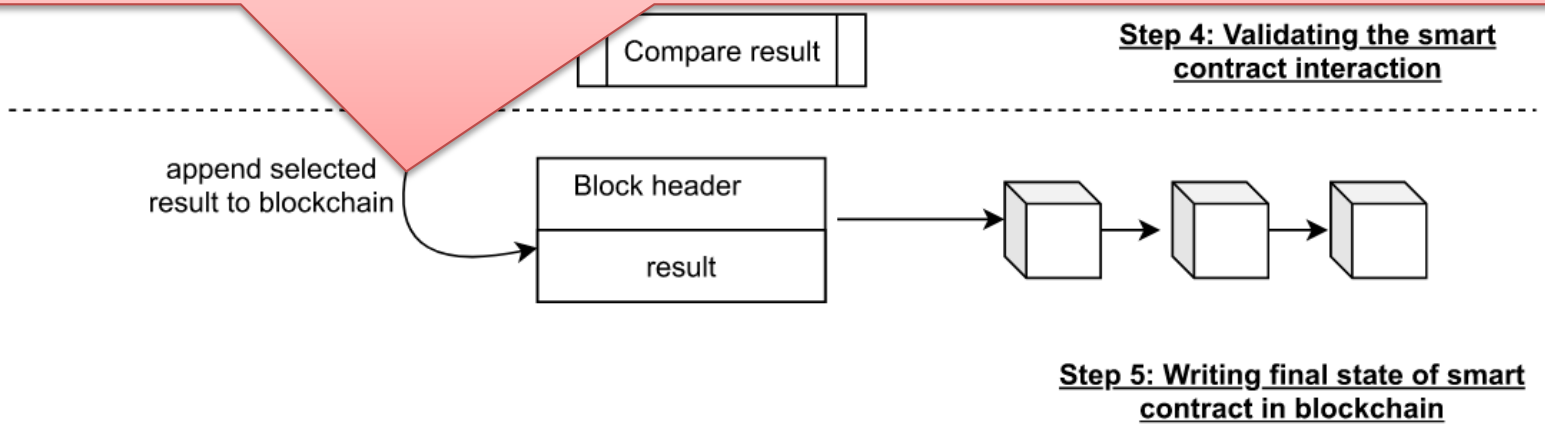
::.. Smart Contract (3/5)

Step 4: The blockchain platform will select the transactions to execute and validate. During the execution, the functions in the transaction will be performed by the nodes. During the validation, the nodes will compare their results and select the one to keep according to a consensus protocol.



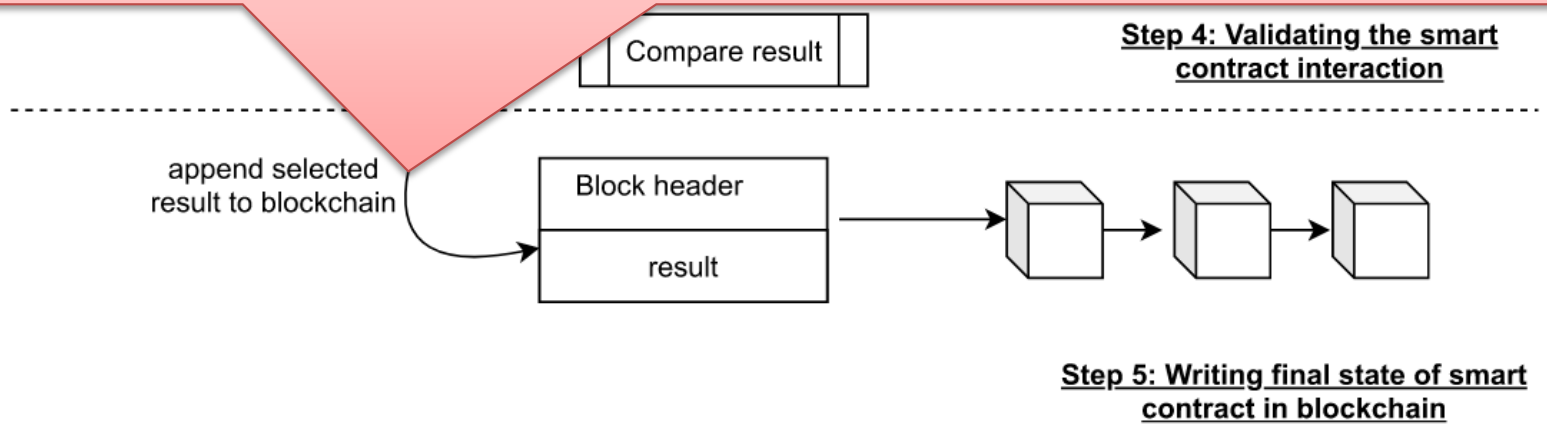
::.. Smart Contract (4/5)

Step 5: Once the valid result is selected, it will be placed in a block to add to the blockchain. If a validated transaction has altered the internal variables of a smart contract, the new values will be considered as initial values from future transactions on the smart contract.



::.. Smart Contract (4/5)

Step 5: Once the valid result is selected, it will be placed in a block to add to the blockchain. If a validated transaction has altered the internal variables of a smart contract, the new values will be considered as initial values from future transactions on the smart contract.



The implementation of smart contracts is not standardized and each blockchain platform offers its own solution.

The functions of a smart contract can be called directly from clients or indirectly from other smart contracts. To ensure that calls end, the client is charged a fee for each call and its duration. If the charge exceeds what the customer is willing to pay, the calculation is stopped and the transactions canceled.

::.. Smart Contract (5/5)

The current approach to developing smart contracts limits throughput because they do not allow competition.

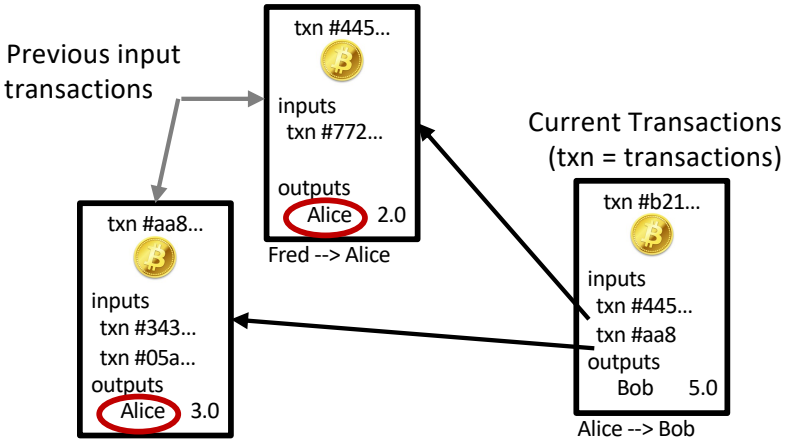
- When a miner creates a block, it assembles a sequence of transactions and calculates a new interim state by running the smart contracts of those transactions in series, in the order they occur in the block.
- A miner cannot execute smart contracts in parallel, because there may be conflicting accesses to shared data and arbitrary interleaving may result in an inconsistent final state. Often for smart contracts, it is not possible to say in advance whether contract executions are in conflict.

Main Platforms

::.. BitCoin (1/6)

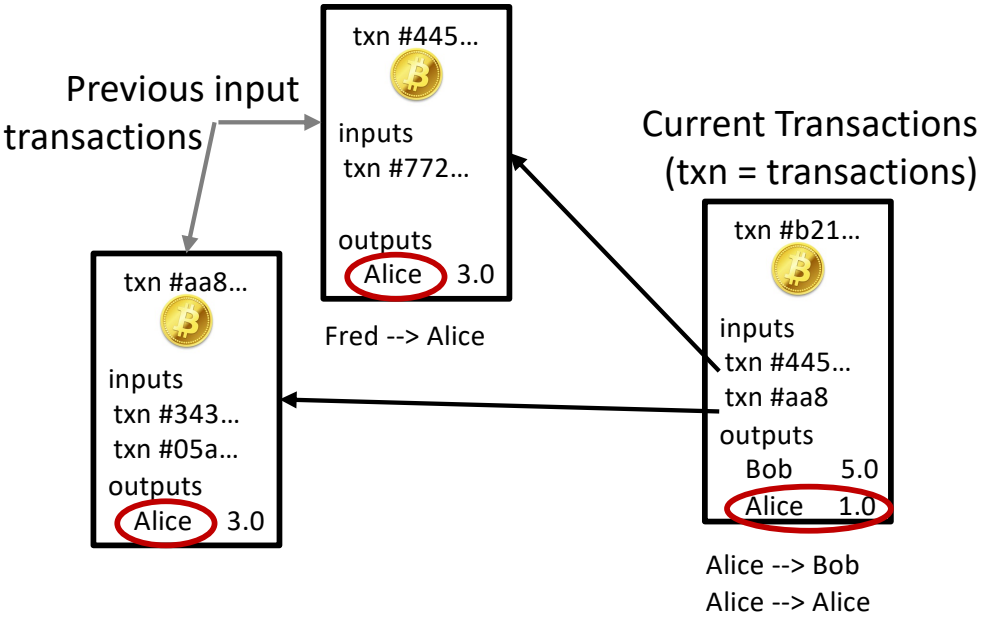
The BitCoin blockchain contains blocks of transactions encoded according to the UTXO (Unspent Transaction Output) model.

Each transaction has in- and out-coming Bitcoins. In input, there is the address of never used transactions and the sum of the outputs is equivalent to the output of the transaction that contains them.



The recipients of a transaction are identified by a Bitcoin address, a 26-35 character identifier. Addresses can be generated for free by any user.

::.. BitCoin (2/6)

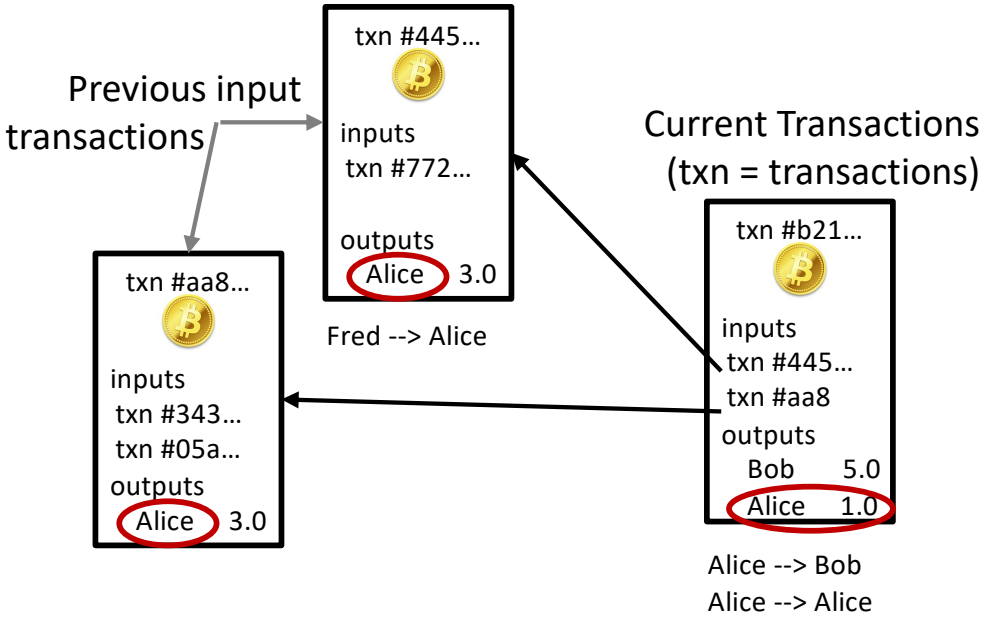


It is often difficult to have transactions that, when added together, are exactly equal to the quantity to be transferred. The excess must be transferred to the author of the transaction.

The UTXO model does not allow to find on the blockchain the current balance of a particular user, which must be reconstructed by collecting all the transactions that have a particular user in output and those with such transactions in input.

UTXO allows parallel transactions because there is no account, it also allows anonymity since a user can have multiple BitCoin addresses. Being stateless, it makes the implementation of applications such as smart contracts complex.

::.. BitCoin (2/6)



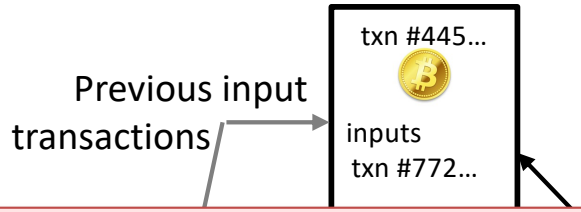
It is often difficult to have transactions that, when added together, are exactly equal to the quantity to be transferred. The excess must be transferred to the author of the transaction.

The UTXO model does not allow to find on the blockchain the current balance of a particular user, which must be reconstructed by collecting

The stateless nature of the network allows for resilience and elasticity, as well as the ability for any instance to perform any activity.

UTXO allows ansactions because there is no account, it also allows anony since a user can have multiple BitCoin addresses. Being stateless, it makes the implementation of applications such as smart contracts complex.

::.. BitCoin (2/6)



Current Transactions (txn = transactions)

It is often difficult to have transactions that, when added

Users have many Bitcoin addresses, which act as pseudonyms to ensure privacy, and it is customary to use one address for each transaction:

- A naive way to accept Bitcoin payments is to tell your customers to send money to a certain address.
- Since Bitcoin transactions are public, if a customer Alice sends Bitcoin, a malicious agent Bob could see the transaction and send an email stating that he was the originator of the payment.
- The recipient of the payment cannot distinguish whether the transfer was actually done by Bob or Alice.
- For this reason it is customary to indicate an address for each customer who has to make a payment.

UTXO allows parallel transactions because [illegible] no account, it also allows anonymity since a user can have multiple BitCoin addresses. Being stateless, it makes the implementation of applications such as smart contracts complex.

::.. BitCoin (3/6)

It is possible to employ a transaction by knowing its identifier, which is public. The problem arises of preventing malicious users from employing transactions that are not their own, and to formulate authentication control mechanisms for the legitimacy of use, a scripting system has been created on BitCoin, the precursor of smart contracts.

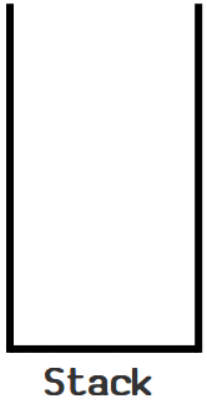
Script is a simple, imperative, stack-based program processed from left to right:

- A script is essentially a list of instructions recorded with each transaction describing how the next person who wants to spend the transferred Bitcoins can access it.
- A transaction is valid if nothing results in an error and the top element of the stack is True (non-zero) at the end of the script.
- Stacks contain byte vectors.

::.. BitCoin (4/6)

The scripting language has two types of statements:

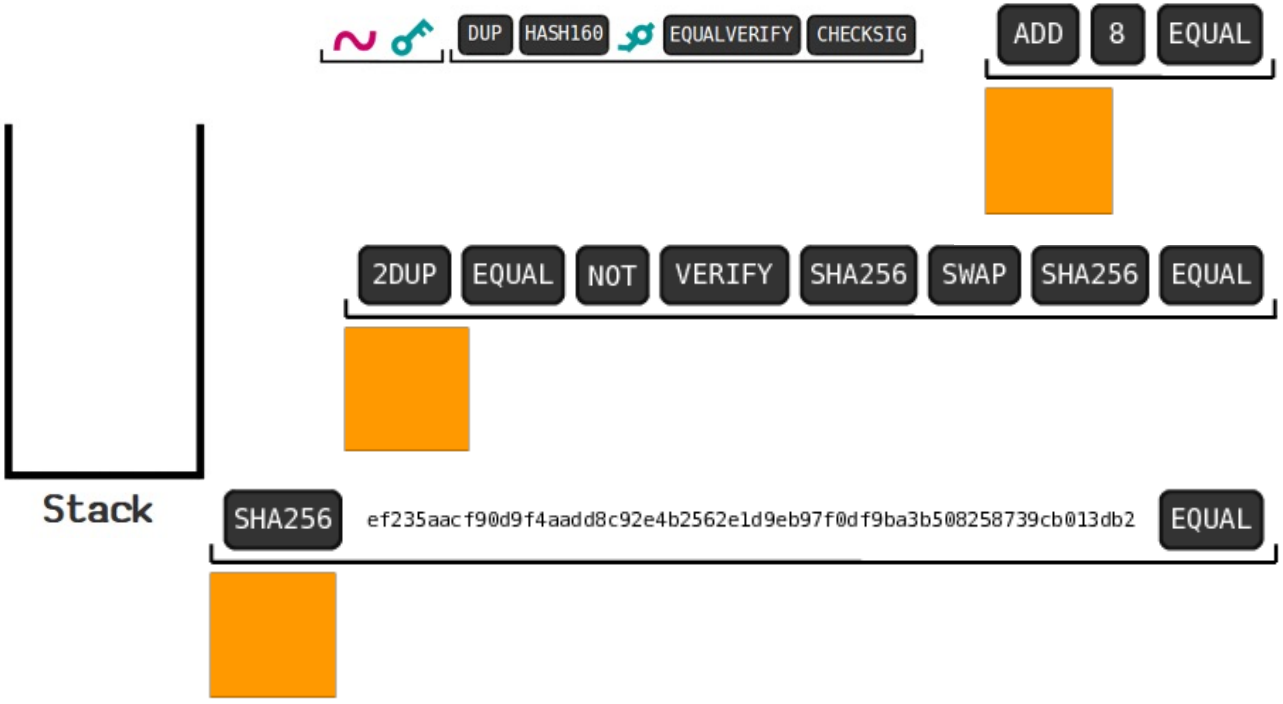
- Data statements simply contain a value and are enclosed in angle brackets (i.e. <data>).
- OP_CODEs are specific operations belonging to the Bitcoin Scripting language that acts on the value at the top of the stack and places their result also at the top of the stack.



::.. BitCoin (4/6)

The scripting language has two types of statements:

- Data statements simply contain a value and are enclosed in angle brackets (i.e. <data>).
- OP_CODEs are specific operations belonging to the Bitcoin Scripting language that acts on the value at the top of the stack and places their result also at the top of the stack.



To spend this output, you need to supply two numbers whose sum is 8.

To spend this output, two different data strings must be supplied that produce the same hash result.

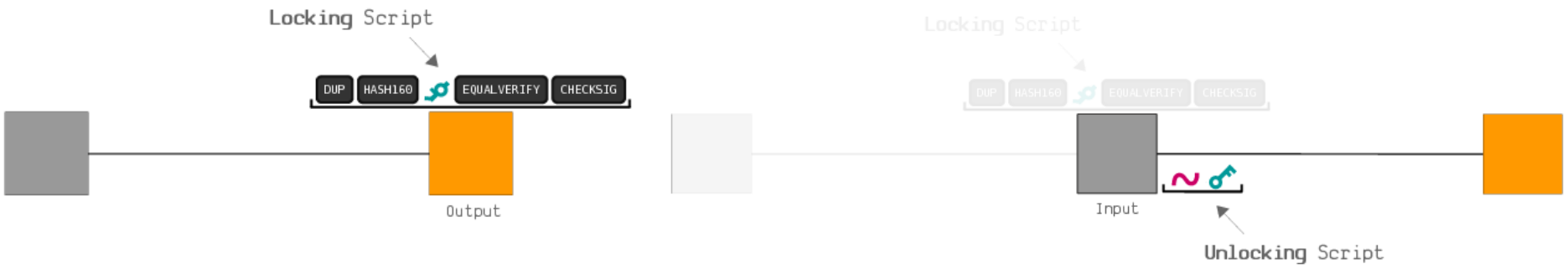
To spend this output, something has to be given that has the same hash as what's inside the block script.

::.. BitCoin (5/6)


```
"vout": [
    {
      "value": 0.10000000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160
7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK"
        ]
      }
    }
  ]
```


Each opcode has a corresponding hexa-decimal representation, which is used to provide its encoding.

A locking script is placed on each output created in a transaction, while an unlocking script must be provided for each input you wish to spend in a transaction.



::.. BitCoin (6/6)

Transazione precedente

txn #3be…

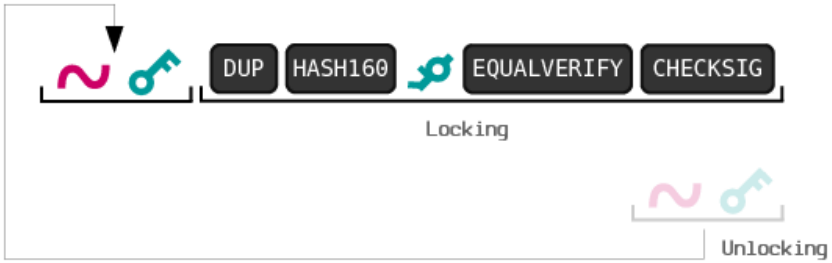
```
"in": [
 {
   "prev_out": [
     "hash": "…"
     "n": 0
   ],
     "scriptSig": "…"
 },
   …
],
"out": [
  {
   "prev_out": [
     "value": "10.12287097"
      "scriptPubKey": "OP_DUP…"
  },
   …
]
```

Transazione corrente

txn #b21…

```
"in": [
 {
   "prev_out": [
     "hash": "3be…"
     "n": 0
   ],
     "scriptSig": "3044…"
 },
   …
],
"out": [
  {
   …
  },
   …
]
```

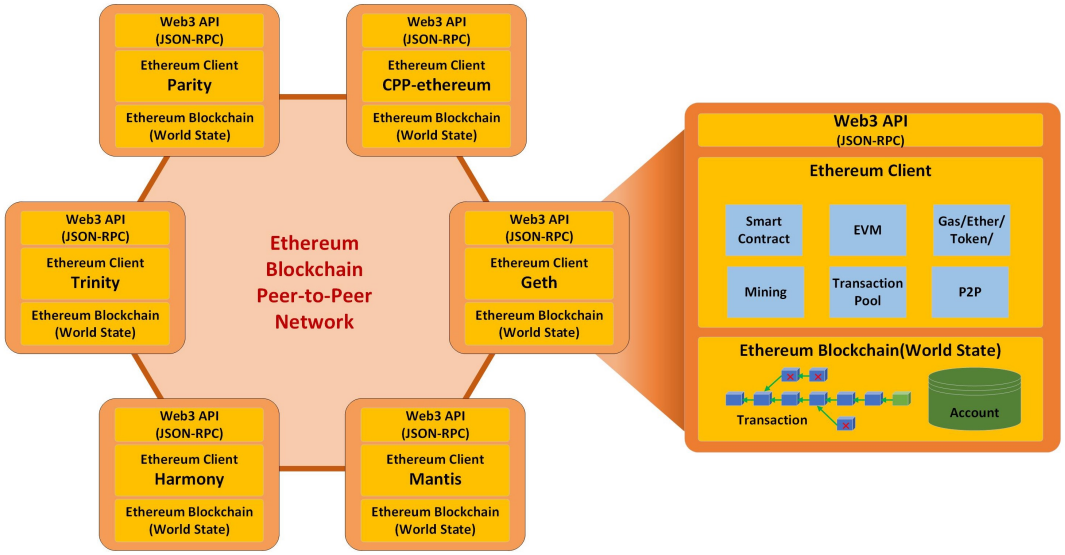
Although the unlocking script is provided after the initial locking script, it is actually placed first when running both scripts.



The language is not Turing-complete: there are no conditional instructions and cycles, because we want to predict the execution time, which depends deterministically on the number of instructions contained in them. Furthermore, it is stateless, unaware of the unlocked currency, or not able to access to the block that contains it.

::.. Ethereum (1/11)

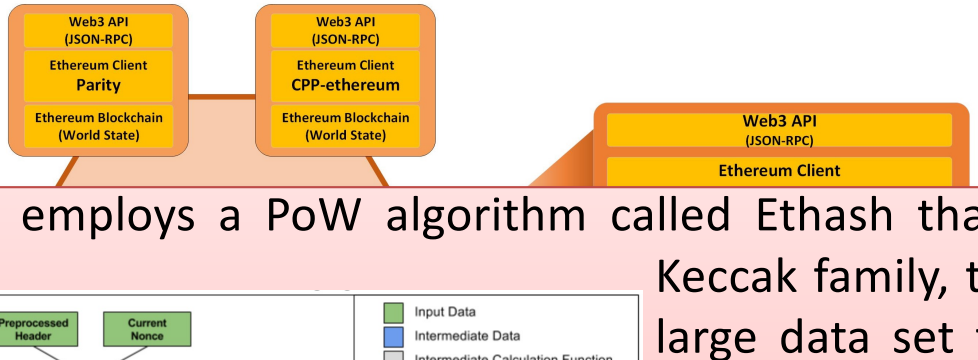
Ethereum is the largest decentralized and open-source software platform that allows the development of smart contracts and applications implemented on top of a permissionaless Blockchain.



- A blockchain network (with PoW and in transition to PoS) and a cryptocurrency like BitCoin, called Ether. Used to pay for gas, a unit of calculation used in transactions and other state transitions.
- The blockchain is characterized by accounts, which interact with each other through transactions (messages) and are characterized by a status and a 20-byte address that identifies them.

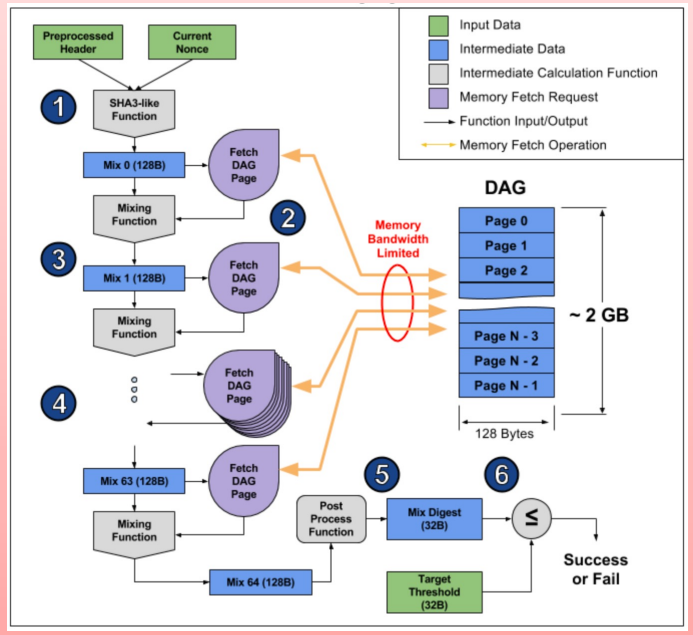
::.. Ethereum (1/11)

Ethereum is the largest decentralized and open-source software platform that allows the development of smart contracts and applications implemented on top of a permissionaless Blockchain.



- A blockchain network (with PoW and in transition to PoS)

It employs a PoW algorithm called Ethash that uses a hash algorithm belonging to the Keccak family, the same as SHA-3 hash functions. It uses a large data set that is periodically regenerated and grows slowly over time.



It performs a series of intensive in-memory random access reads to the 2 GB dataset structured as a direct acyclic graph (DAG). The resolution is achieved with 64 cycles starting from the initial information hash with pages extracted from the DAG. The digest is compared to a target threshold. If less than or equal, the calculation is considered successful. Otherwise, the algorithm is rerun with a different nonce (incrementing the current one or choosing a new one at random).

::.. Ethereum (1/11)

Ethereum is the largest decentralized and open-source software

ts and

n.

(with

to PoS)

cy like

Used to

calcu-

ons and

ate transitions.

Ethereum manages two different types of accounts:

- Externally owned accounts (EOA), they are able to send and receive Ether, and send transactions to Smart Contracts;
- Contract accounts, which in addition to the functionality of the EOAs have associated code that implement actions "triggered" by EOA or other Contract accounts.

Executing the code modifies the information contained in your storage space (allowing you to use the Blockchain for purposes other than cryptocurrencies).

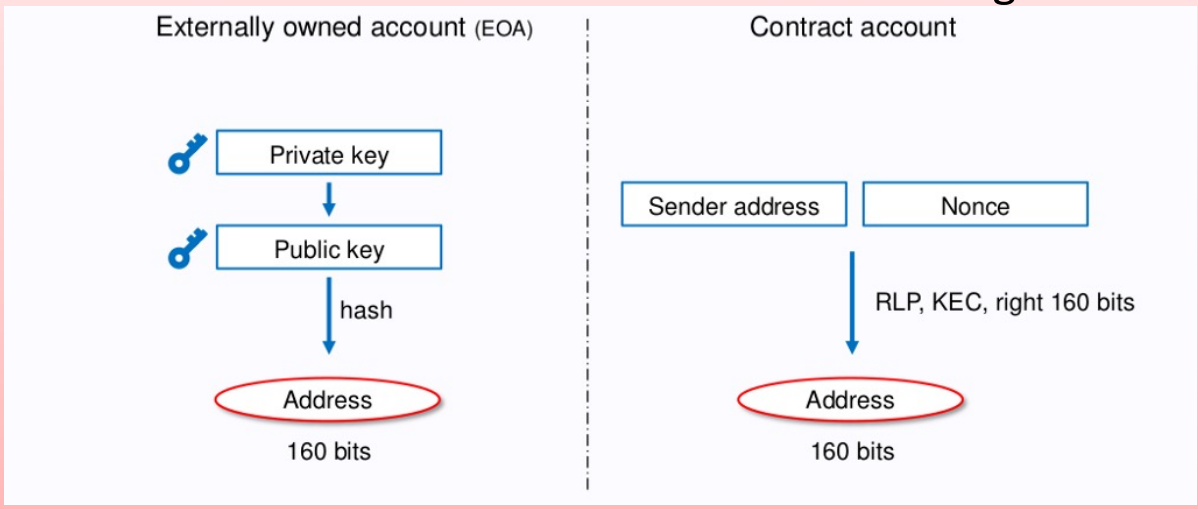
- The blockchain is characterized by accounts, which interact with each other through transactions (messages) and are characterized by a status and a 20-byte address that identifies them.

::.. Ethereum (1/11)

Ethereum is the largest decentralized and open-source software platform that allows the development of smart contracts and applications implemented on top of a permissionaless Blockchain.

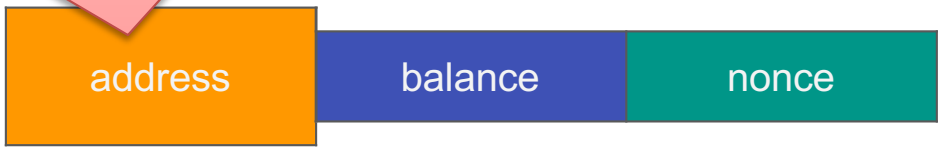
- A blockchain network (with

Ethereum addresses contain 40 hexadecimal digits.



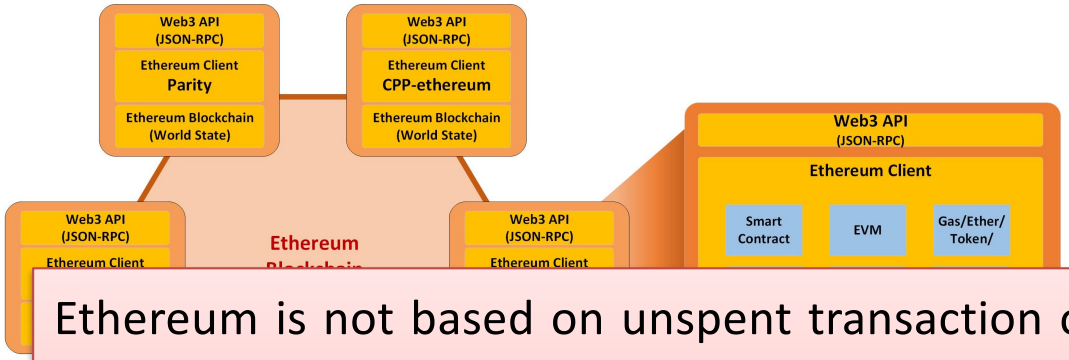
EOA addresses begin with the prefix "0x" followed by the rightmost 20 bytes of the public key hash. Contract addresses are in the same format, but are determined by the sender and the nonce equal to the transaction number sent by the sender.

each ot sactions (messages) and are characterized by a status a yte address that identifies them.



::.. Ethereum (1/11)

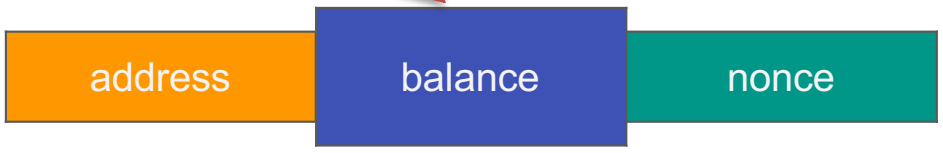
Ethereum is the largest decentralized and open-source software platform that allows the development of smart contracts and applications implemented on top of a permissionaless Blockchain.



- A blockchain network (with PoW and in transition to PoS) and a cryptocurrency like

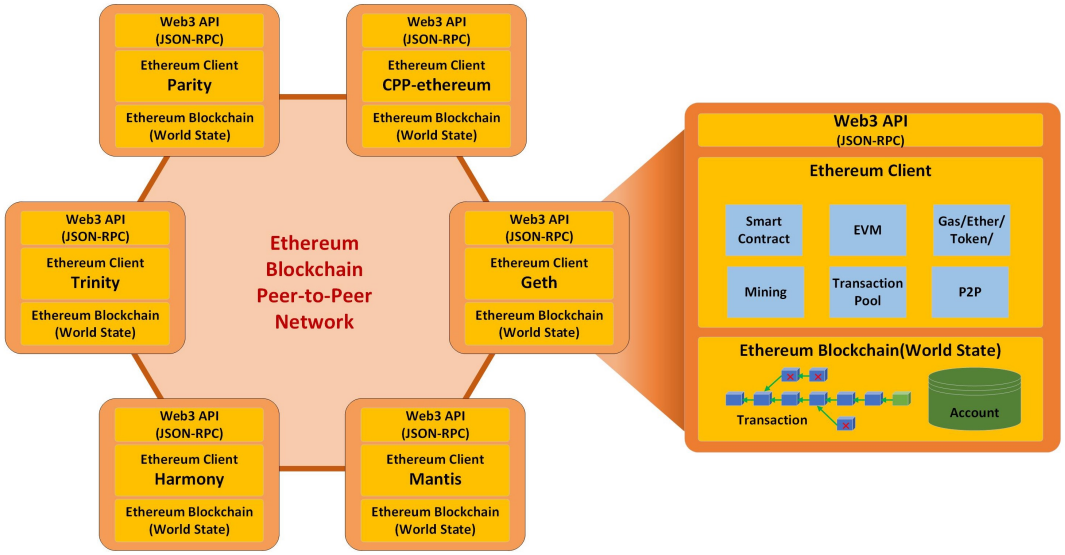
Ethereum is not based on unspent transaction outputs (UTXO), but accounts have a status that indicates the current balance. Status is not stored on the blockchain, but in a separate Merkle Patricia tree. A cryptocurrency wallet stores public and private keys (a user can have multiple addresses), which can be used to receive or spend ether. Wallets are like applications that allow you to interact with an Ethereum account, similar to e-banking apps and a bank account.

each oth actions (messages) and are characterized by a status and a 20-b dress that identifies them.



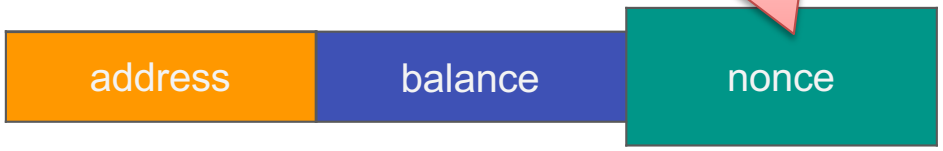
::.. Ethereum (1/11)

Ethereum is the largest decentralized and open-source software platform that allows the development of smart contracts and applications implemented on top of a permissionaless Blockchain.



- A blockchain network (with PoW and in transition to PoS) and a cryptocurrency like BitCoin, called Ether. Used to pay for gas, a unit of calculation used in transactions and other state transitions.
- The blockchain is characterized by accounts, which interact with each other through transactions cterized by a status and a 20-byte address that ifies them.

Total number of transactions.



::.. Ethereum (2/11)

Contracts generally have 4 purposes:

- Manage a "data store" that contains useful information for other contracts or for the outside world;
- Behave as an EOA with more advanced access policies (a sort of filter that allows the forwarding of "messages" to certain EOAs only if certain conditions are met);
- Manage an ongoing contract or relationship between multiple users (an example is a contract that automatically pays whoever submits a valid solution to some mathematical problem, or demonstrates that it is providing a computational resource);
- Provide functionality to other contracts (a kind of library).

The code is interpreted by the Ethereum Virtual Machine (EVM), and represented in an EVM bytecode.

::.. Ethereum (3/11)

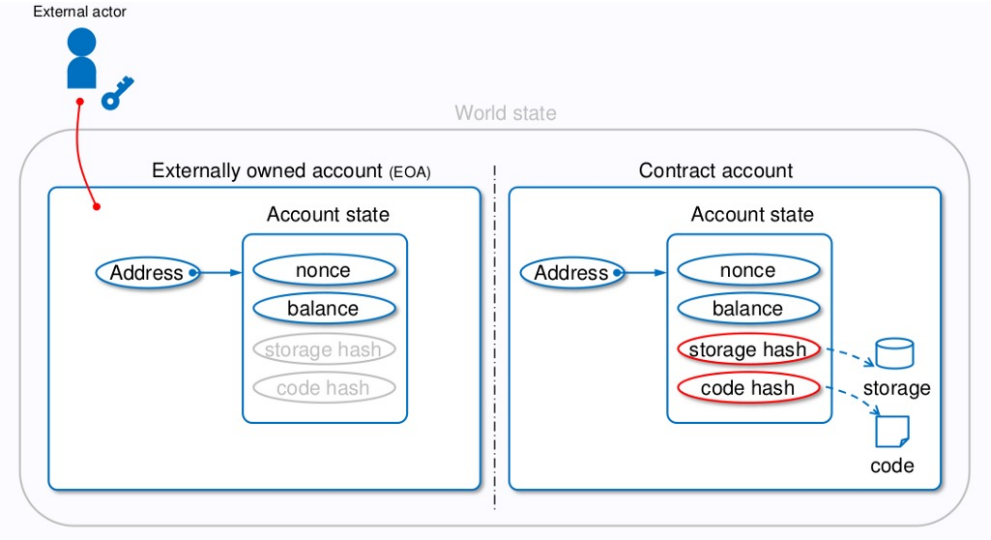
Contract accounts are different:

	Personal Account	Contract Account
address	H(pub_key)	H(addr + creator nonce)
code	∅	Code
storage	∅	Data
balance	ETH Balance (in Wei)	ETH Balance (in Wei)
nonce	# sent transactions	# sent transactions

::.. Ethereum (3/11)

Contract accounts are different:

	Personal Account	Contract Account
address	H(pub_key)	H(addr + creator nonce)
code	∅	Code
storage	∅	Data
balance	ETH Balance (in Wei)	ETH Balance (in Wei)
nonce	# sent transactions	# sent transactions

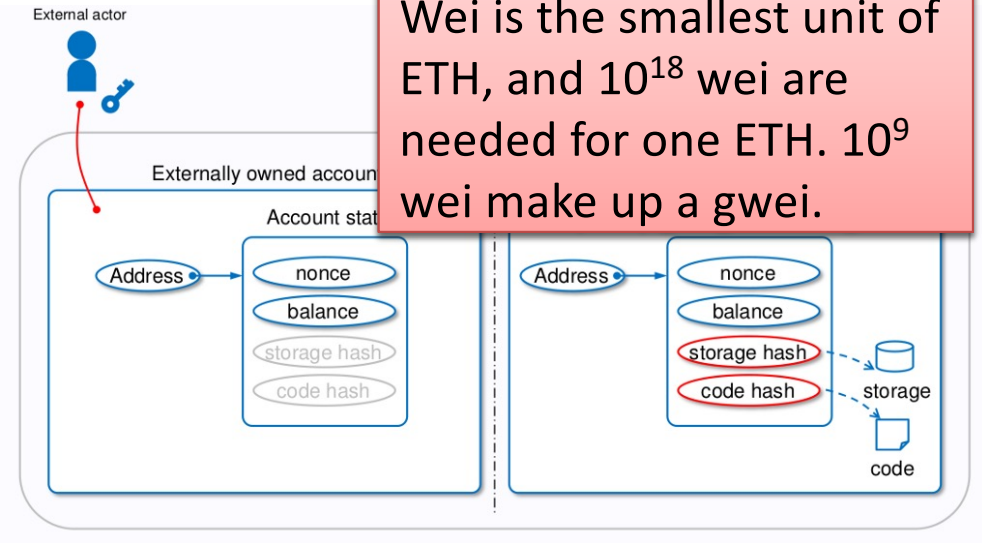


::.. Ethereum (3/11)

Contract accounts are different:

	Personal Account	Contract Account
address	H(pub_key)	H(addr + creator nonce)
code	∅	Code
storage	∅	Data
balance	ETH Balance (in Wei)	ETH Balance (in Wei)
nonce	# sent transactions	# sent transactions

Wei is the smallest unit of ETH, and 10^{18} wei are needed for one ETH. 10^{9} wei make up a gwei.

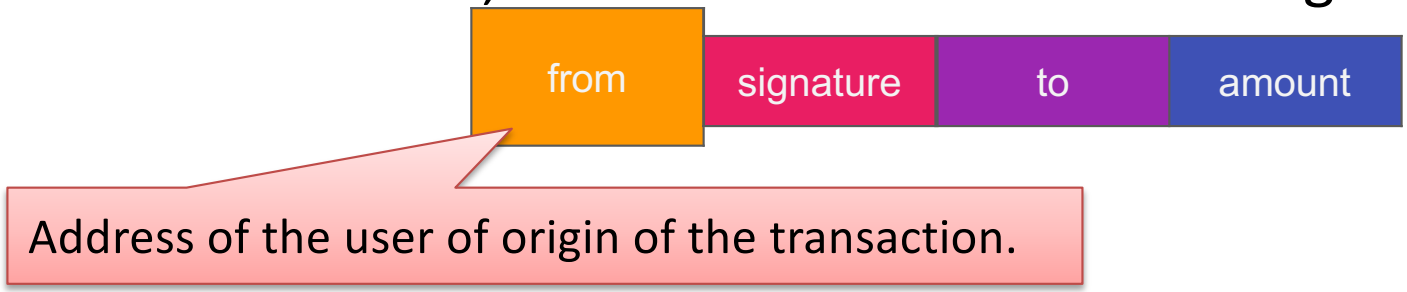


::.. Ethereum (3/11)

Contract accounts are different :

	Personal Account	Contract Account
address	H(pub_key)	H(addr + creator nonce)
code	∅	Code
storage	∅	Data
balance	ETH Balance (in Wei)	ETH Balance (in Wei)
nonce	# sent transactions	# sent transactions

On the other hand, a transaction has the following data structure:

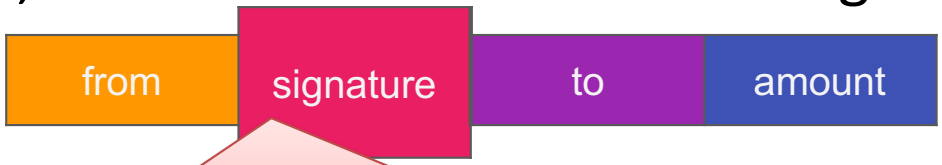


::.. Ethereum (3/11)

Contract accounts are different :

	Personal Account	Contract Account
address	H(pub_key)	H(addr + creator nonce)
code	∅	Code
storage	∅	Data
balance	ETH Balance (in Wei)	ETH Balance (in Wei)
nonce	# sent transactions	# sent transactions

On the other hand, a transaction has the following data structure:



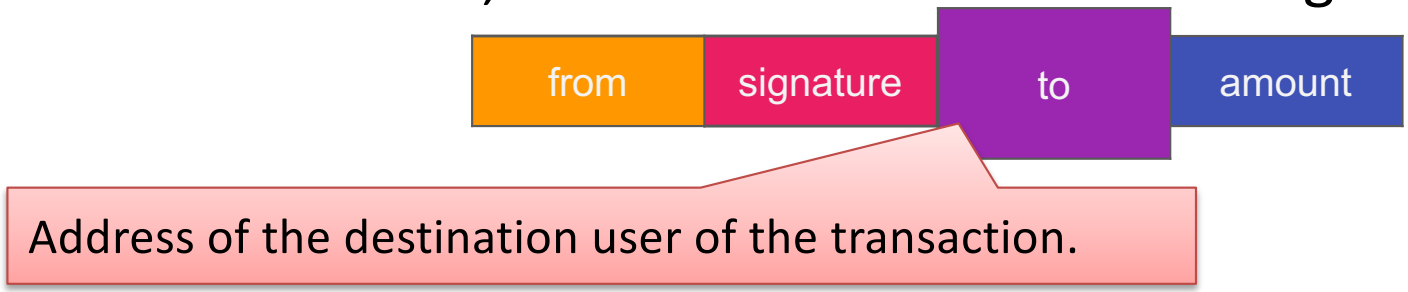
Signing of the new transaction using the private key of the creator user.

::.. Ethereum (3/11)

Contract accounts are different :

	Personal Account	Contract Account
address	H(pub_key)	H(addr + creator nonce)
code	∅	Code
storage	∅	Data
balance	ETH Balance (in Wei)	ETH Balance (in Wei)
nonce	# sent transactions	# sent transactions

On the other hand, a transaction has the following data structure:

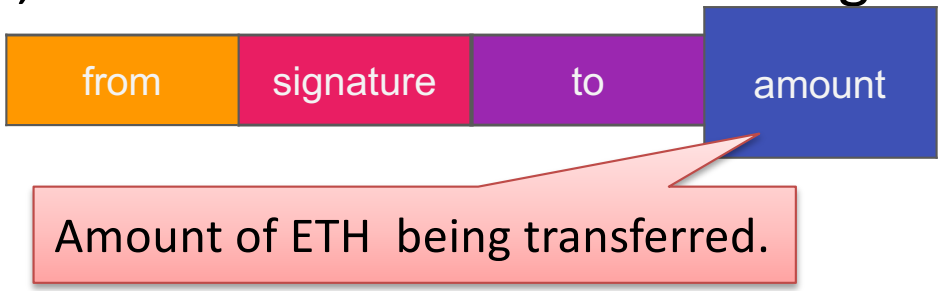


::.. Ethereum (3/11)

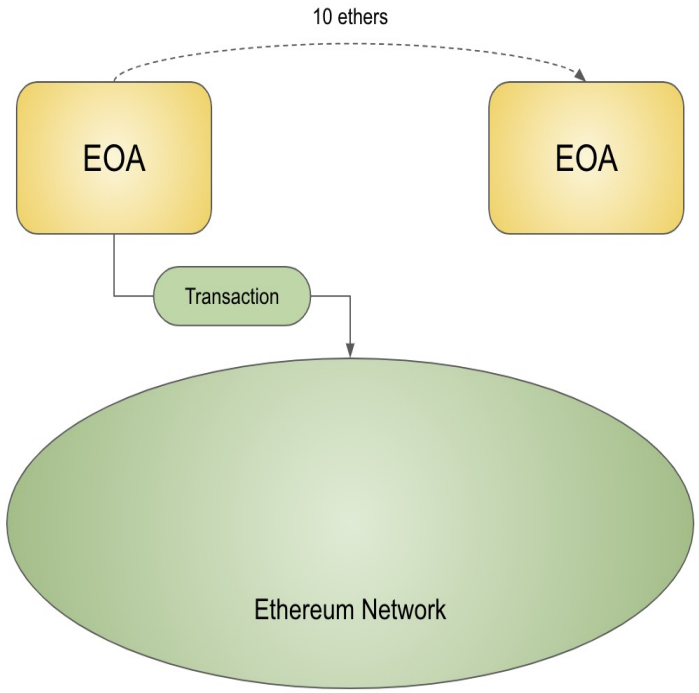
Contract accounts are different :

	Personal Account	Contract Account
address	H(pub_key)	H(addr + creator nonce)
code	∅	Code
storage	∅	Data
balance	ETH Balance (in Wei)	ETH Balance (in Wei)
nonce	# sent transactions	# sent transactions

On the other hand, a transaction has the following data structure:



::.. Ethereum (4/11)



- Web3.fromWei converts a value expressed in Wei to Ether;
- Web3.toWei operates the reverse;
- The transaction required a commission (in gas), so the value associated with the account[0] is less than 90 Ether.

Ethereum terminal through which to call functions of the main class of the web3.js library.

```
> web3.fromWei(eth.getBalance(eth.accounts[0]))
100
> web3.fromWei(eth.getBalance(eth.accounts[1]))
100
> eth.sendTransaction({
...... from: eth.accounts[0],
...... to: eth.accounts[1],
...... value: web3.toWei(10)
...... })
"0x497913c178f65613035b22340fcf5bc59c7ed474bfa3c1e798c6dffbeda9da5b"
>
> web3.fromWei(eth.getBalance(eth.accounts[0]))
89.99958
> web3.fromWei(eth.getBalance(eth.accounts[1]))
110
```

- The successful transaction generates a hash code as a receipt.

::.. Ethereum (5/11)

Contract accounts are different :

	Personal Account	Contract Account
address	H(pub_key)	H(addr + creator nonce)
code	∅	Code
storage	∅	Data
balance	ETH Balance (in Wei)	ETH Balance (in Wei)
nonce	# sent transactions	# sent transactions

On the other hand, a transaction has the following data structure:

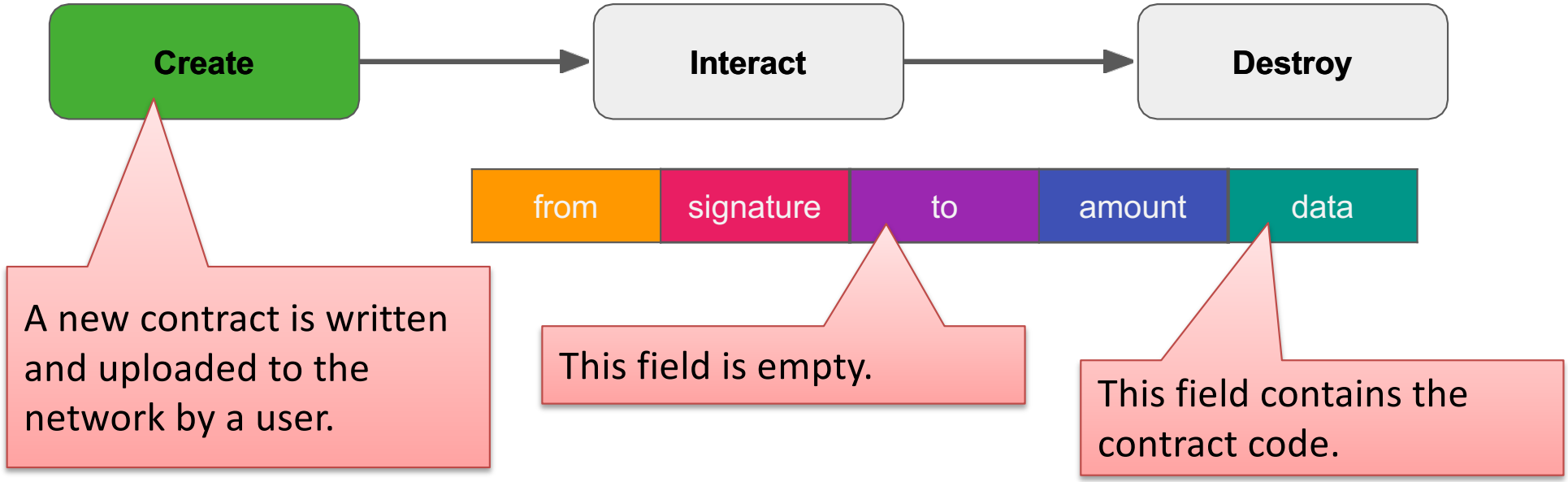
In the case of transactions for smart contracts, the structure changes to also contain the data for the contract:

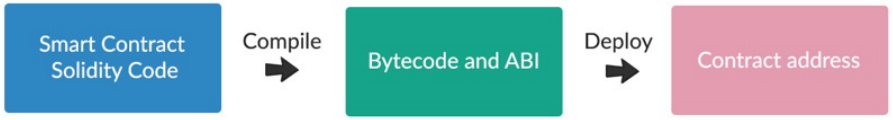
::.. Ethereum (6/11)

The life cycle of smart contracts consists of 3 states:

::.. Ethereum (6/11)

The life cycle of smart contracts consists of 3 states:

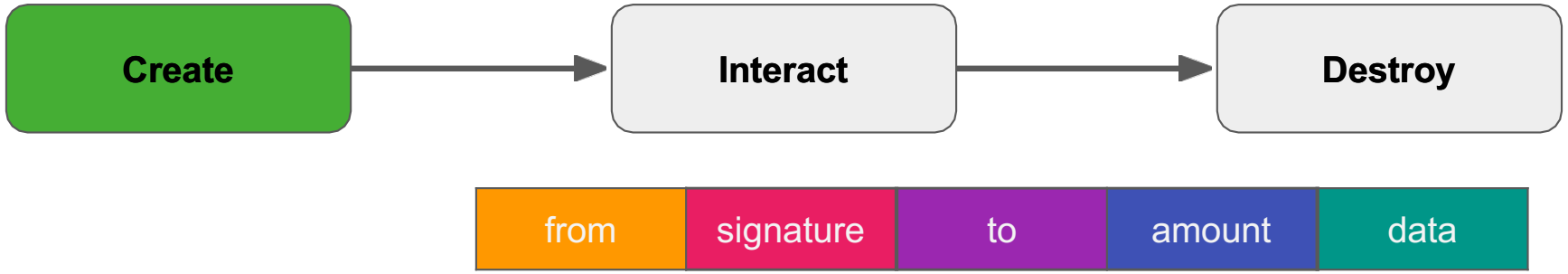


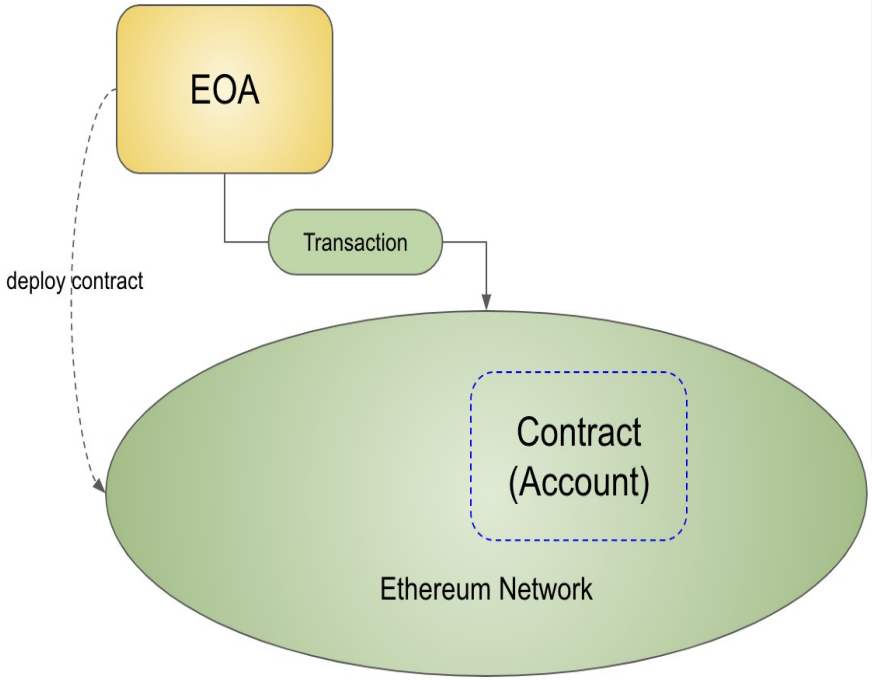


Smart contracts are written in high-level programming languages (e.g. Solidity), this code is compiled by the EVM and deployed in the Blockchain in the form of EVM Bytecode.

::.. Ethereum (6/11)

The life cycle of smart contracts consists of 3 states:





Contract

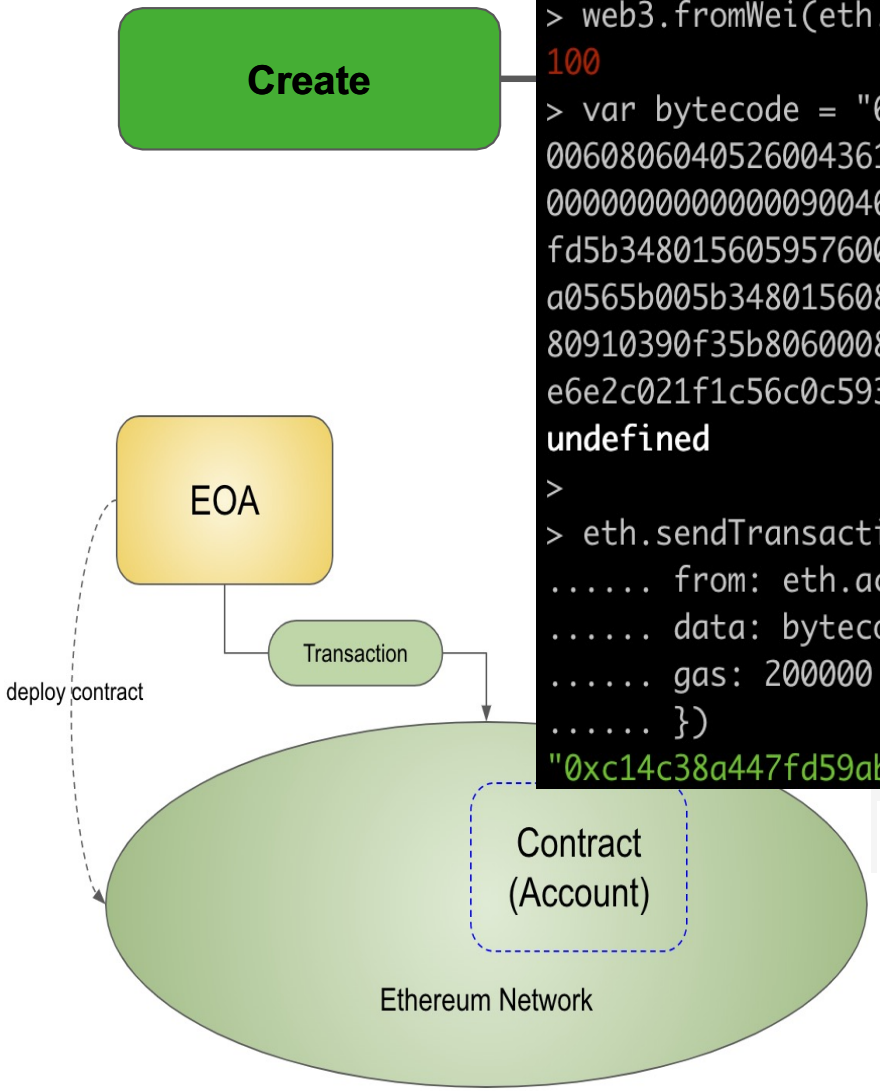
```
pragma solidity ^0.4.0;

contract SimpleStorage {

uint storedData;

function set(uint x) public {
  storedData = x;
}

function get() constant public returns (uint retVal) {
  return storedData;
}

}
```

Contract's Bytecode

```
"6080604052348015610010576000080fd5b5060df8061001f6000396000f30060806
040526004361060495760003357c0100000000000000000000000000000000000000
0000000000000000900463ffffffff16806360fe47b114604e5780636d4ce63c146
078575b600080fd5b348015605957600080fd5b506076600480360381019080803359
060200190929190505050060a0565b005b34801560835760080fd5b50608a60aa565
b60405180828152602001915050604051809103905f35b806000819055505056 5b600
080549050905600a165627a7a7230582080122bb351e6e2c021f1c56c0c5933087e7
62ea6e7a3360b902b39cbed5a38f10029"
```

::.. Ethereum (6/11)

The life cycle of smart contracts consists of 3 states:



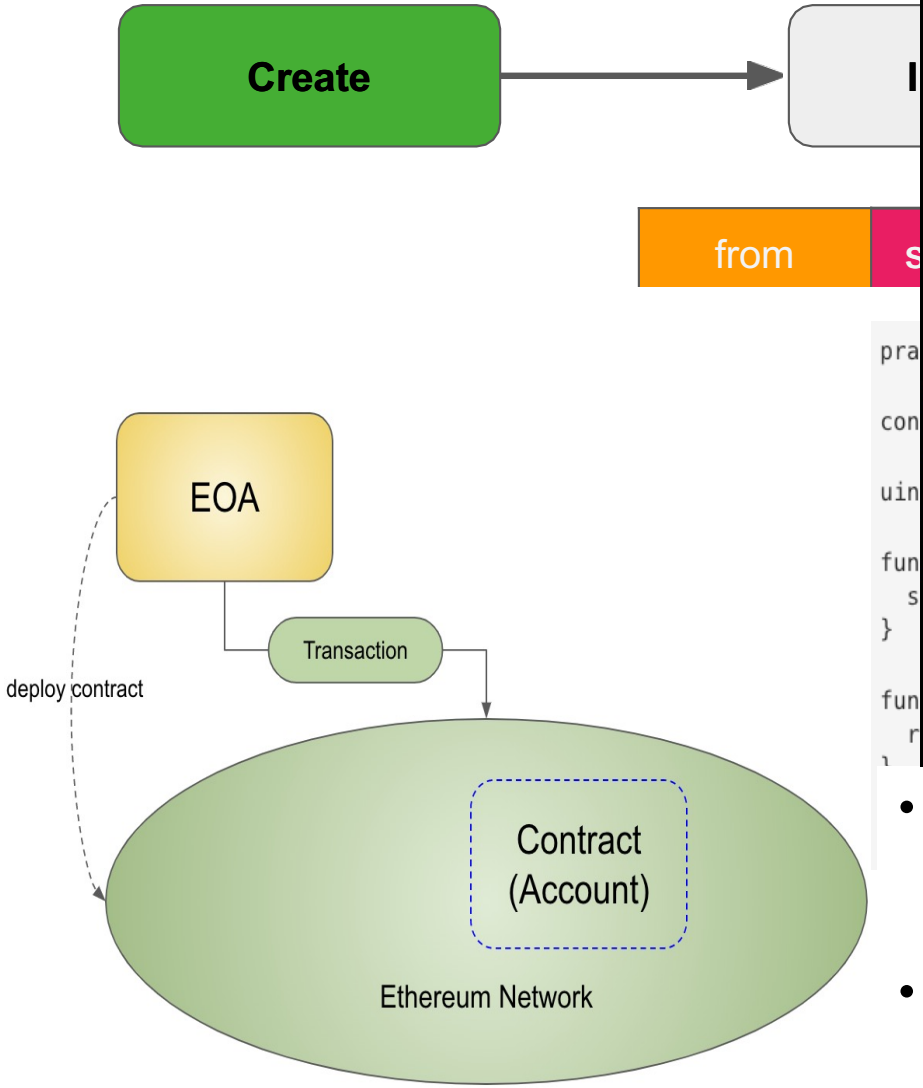

```
> web3.fromWei(eth.getBalance(eth.accounts[0]))
100
> var bytecode = "6080604052348015610010576000080fd5b5060df8061001f6000396000f3
006080604052600436106049576000357c0100000000000000000000000000000000000000000000
0000000000000900463ffffffff16806360fe47b114604e5780636d4ce63c146078575b600080
fd5b348015605957600080fd5b506076600480360381019080803590602001909291905050506060
a0565b005b348015608357600080fd5b50608a60aa565b604051808281526020019150506040518
80910390f35b8060008190555050565b600080549050905600a165627a7a7230582080122bb351
e6e2c021f1c56c0c5933087e762ea6e7a3360b902b39cbed5a38f10029"
undefined
>
> eth.sendTransaction({
...... from: eth.accounts[0],
...... data: bytecode,
...... gas: 200000
...... })
"0xc14c38a447fd59ab6eae4df47bd7c15f3125446596675f9ea8741e81f79890d9"
```

- The EOA creates a contract with a transaction containing the related bytecode and administering a certain quantity of gas.
- Executing the transaction returns a hash code.

62ea6e7a3360b902b39cbed5a38f10029"

::.. Ethereum (6/11)

The life cycle of smart contracts consists of 3 states:

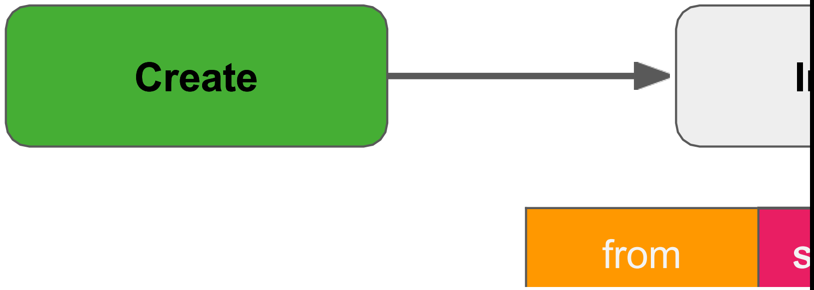



```
> eth.getTransaction("0xc14c38a447fd59ab6eae4df47bd7c15f3125446596675f9ea8741e
81f79890d9")
{
  blockHash: "0xe8a1ed7403baa039f966a22b442cedbf5adbd28c9802fea6806e57d75c8ce4
cf",
  blockNumber: 1,
  from: "0x747e967c24abec02b7243e3287cc5ec0f4534a89",
  gas: 200000,
  gasPrice: 20000000000,
  hash: "0xc14c38a447fd59ab6eae4df47bd7c15f3125446596675f9ea8741e81f79890d9",
  input: "0x6080604052348015610010576000080fd5b5060df8061001f6000396000f3006080
60405260043610604957600035 7c0100000000000000000000000000000000000000000000000000
00000000900463ffffffff16806360fe47b114604e5780636d4ce63c146078575b600080fd5b34
8015605957600080fd5b506076600480360381019080803590602001909291905050506 0a0565b
005b348015608357600080fd5b50608a60aa565b604051808281526020019150506040518091 03
90f35b8060008190555050565b600080549050905600a165627a7a7230582080122bb351e6e2c0
21f1c56c0c5933087e762ea6e7a3360b902b39cbed5a38f10029",
  nonce: 0,
  to: "0x0",
  transactionIndex: 0,
  value: 0
}
```

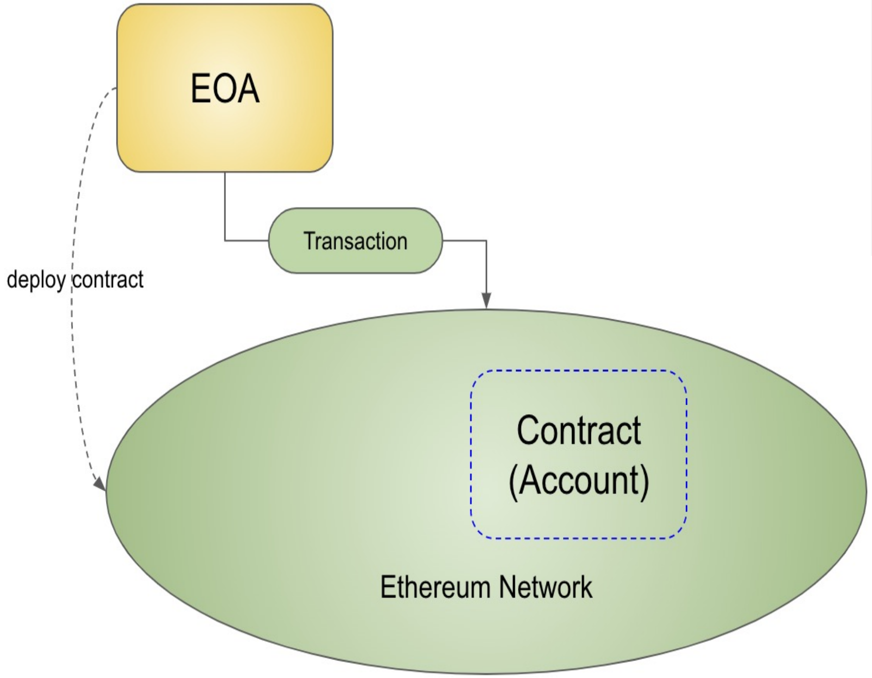
- By executing getTransaction (hash_value) information about the executed transaction is obtained.
- The report obtained also informs us about the block created within the Blockchain.

::.. Ethereum (6/11)

The life cycle of smart contracts c



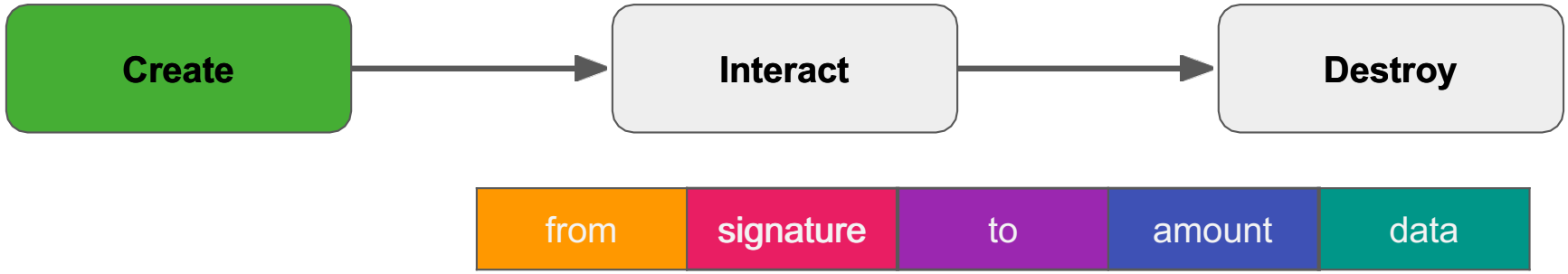

```
> eth.getTransactionReceipt("0xc14c38a447fd59ab6eae4df47bd7c15f3125446596675f9
ea8741e81f79890d9")
{
  blockHash: "0xe8a1ed7403baa039f966a22b442cedbf5adbd28c9802fea6806e57d75c8ce4
cf",
  blockNumber: 1,
  contractAddress: "0xa8e28f1a7031968fb830e5a70c4b246b07f64d2a",
  cumulativeGasUsed: 112213,
  gasUsed: 112213,
  logs: [],
  logsBloom: "0x00000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000",
  status: "0x1",
  transactionHash: "0xc14c38a447fd59ab6eae4df47bd7c15f3125446596675f9ea8741e81
f79890d9",
  transactionIndex: 0
}
```

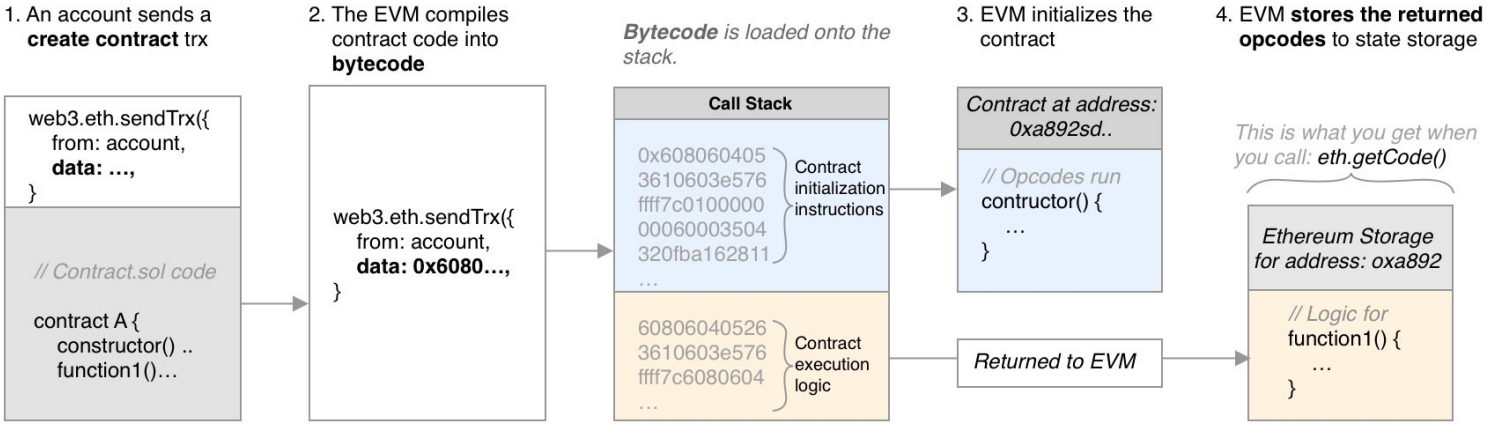


- By executing getTransactionRecipt (hash_value) you get a "receipt", with the address of the contract and the information on the gas spent.
- Upon entry to the transaction, a gas value equal to 200,000 was entered, but since the "gaslimit" is equal to 90000, the unspent portion is returned to the EOA.

::.. Ethereum (6/11)

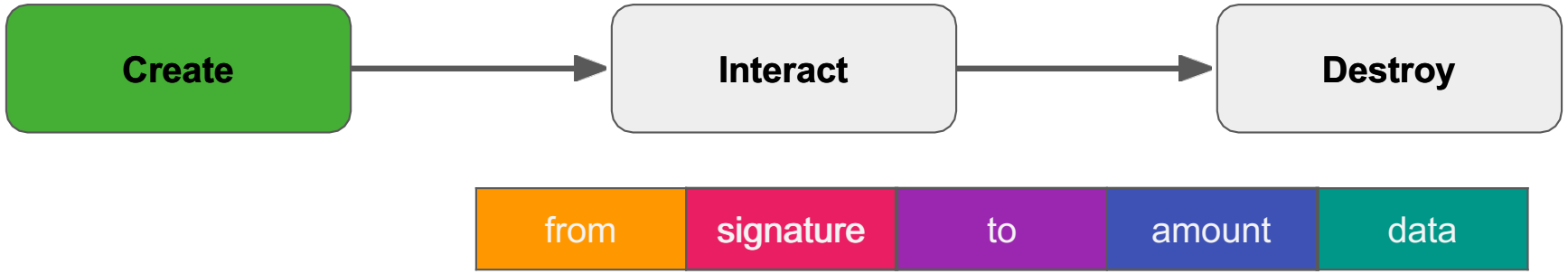
The life cycle of smart contracts consists of 3 states:



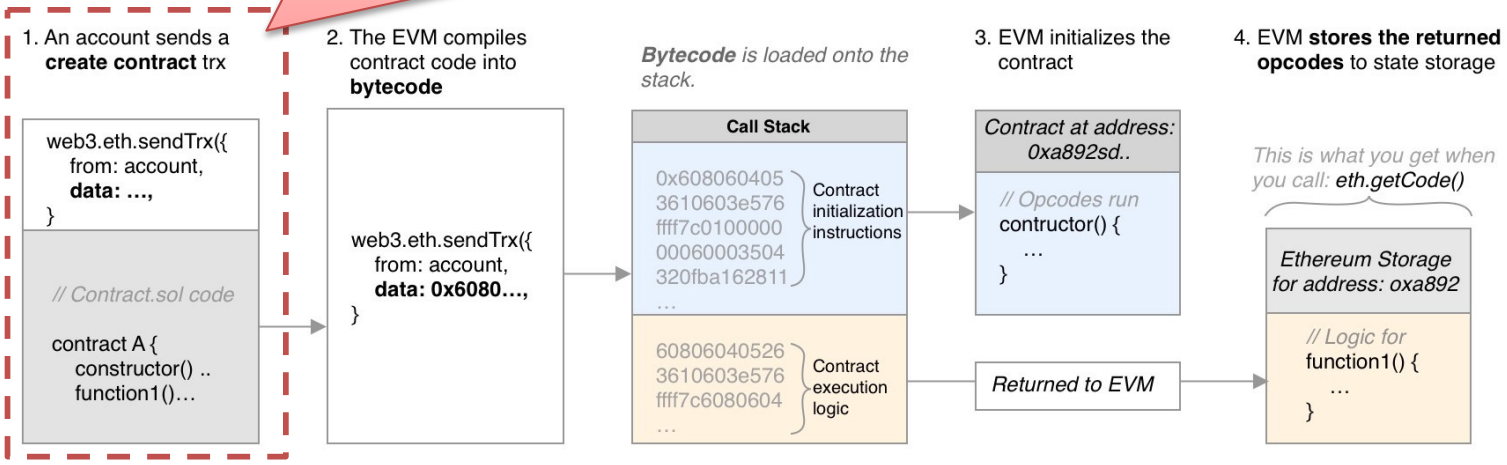


::.. Ethereum (6/11)

The life cycle of smart contracts consists of 3 states:

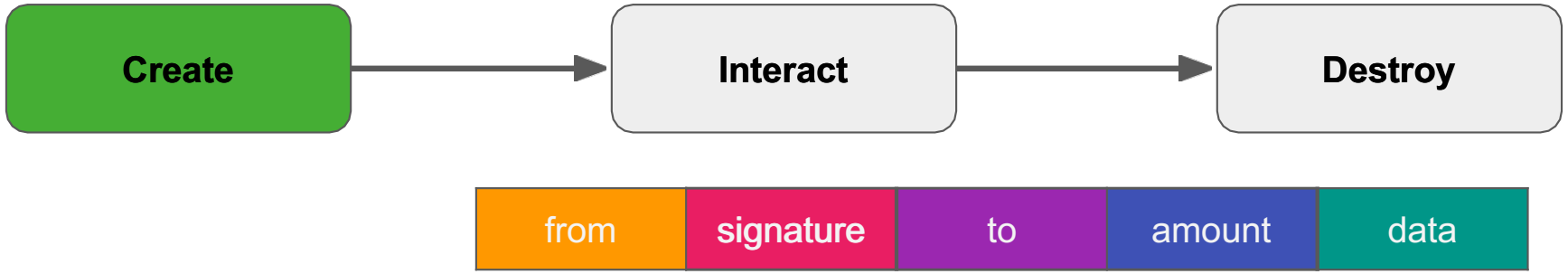


An EOA or contract sends a transaction containing data, but no recipient address. This format tells the EVM that it is a contract creation, not a normal send / call transaction.

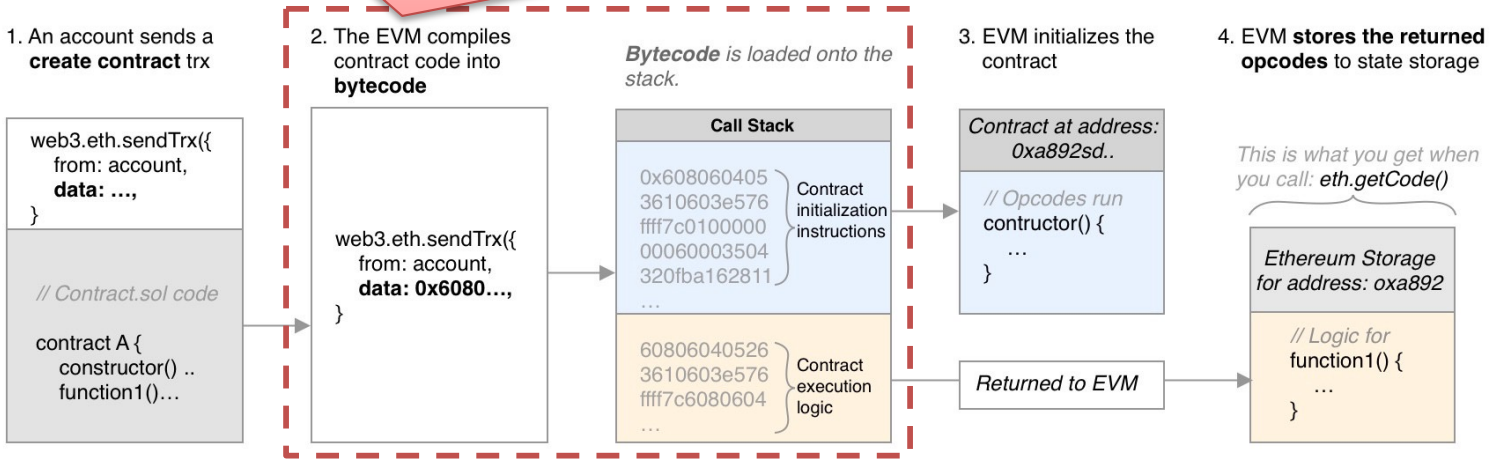


::.. Ethereum (6/11)

The life cycle of smart contracts consists of 3 states:

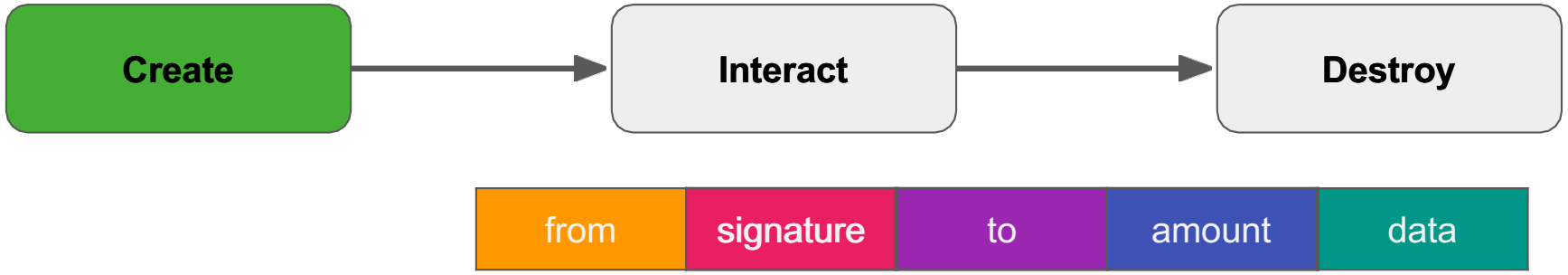


EVM compiles the contract code in Solidity obtaining the bytecode, which translates the contract code directly into opcode, which are executed in a single call stack.

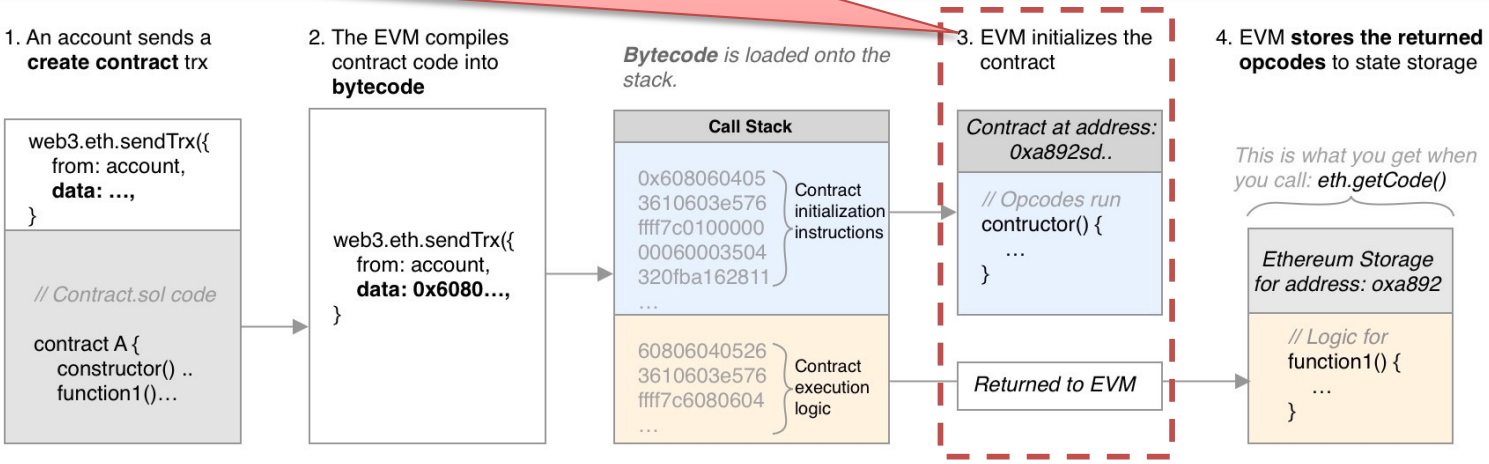


::.. Ethereum (6/11)

The life cycle of smart contracts consists of 3 states:

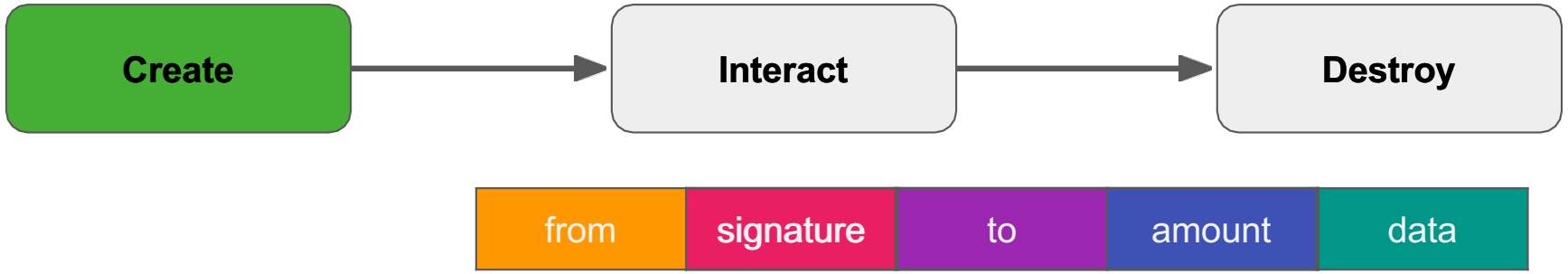


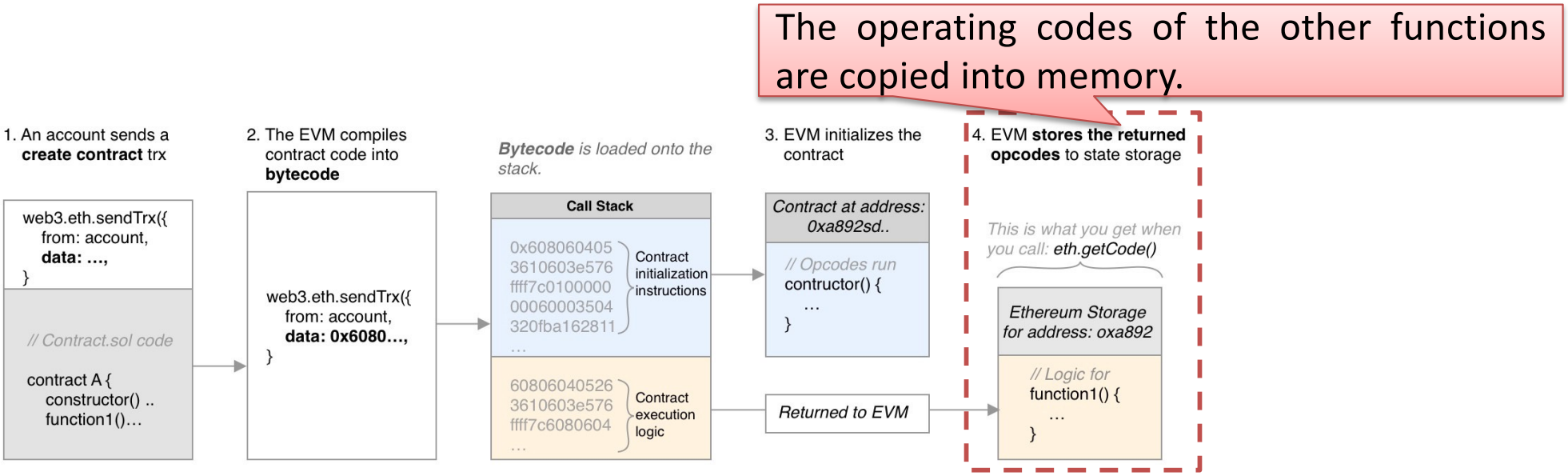
While creating the contract, EVM only executes the initialization code until it reaches the first STOP or RETURN statement in the stack, the content beyond that point is returned to the EVM and represents the functions that can be invoked. During this phase, an address is assigned to the contract.



::.. Ethereum (6/11)

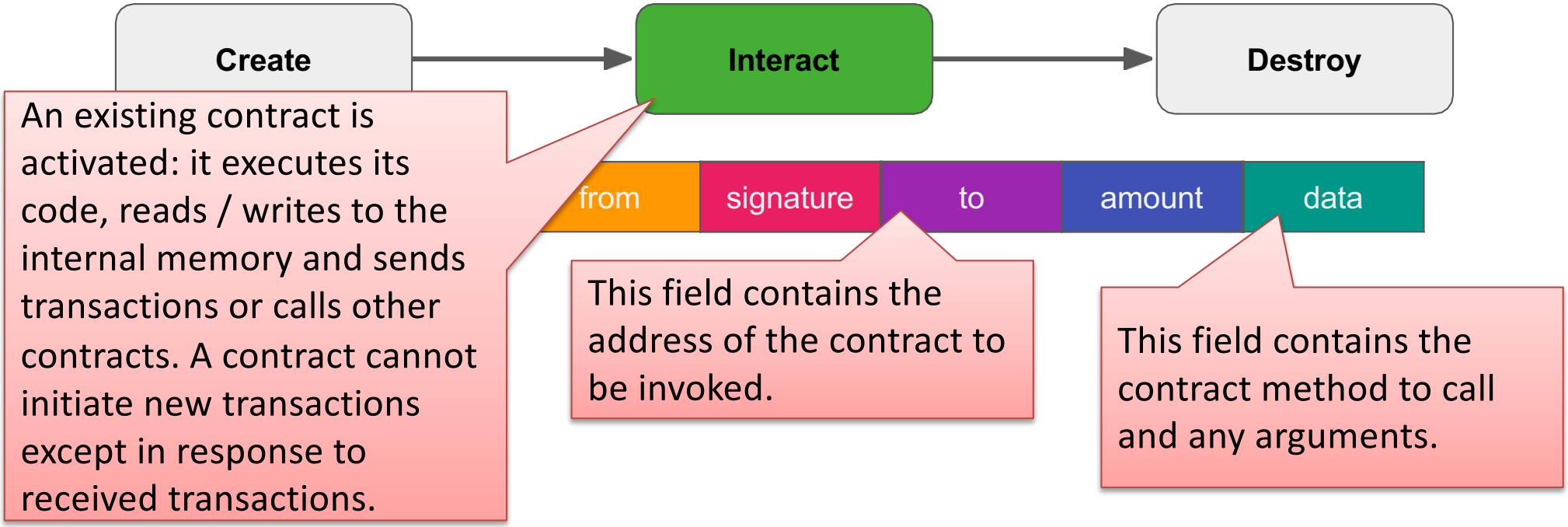
The life cycle of smart contracts consists of 3 states:

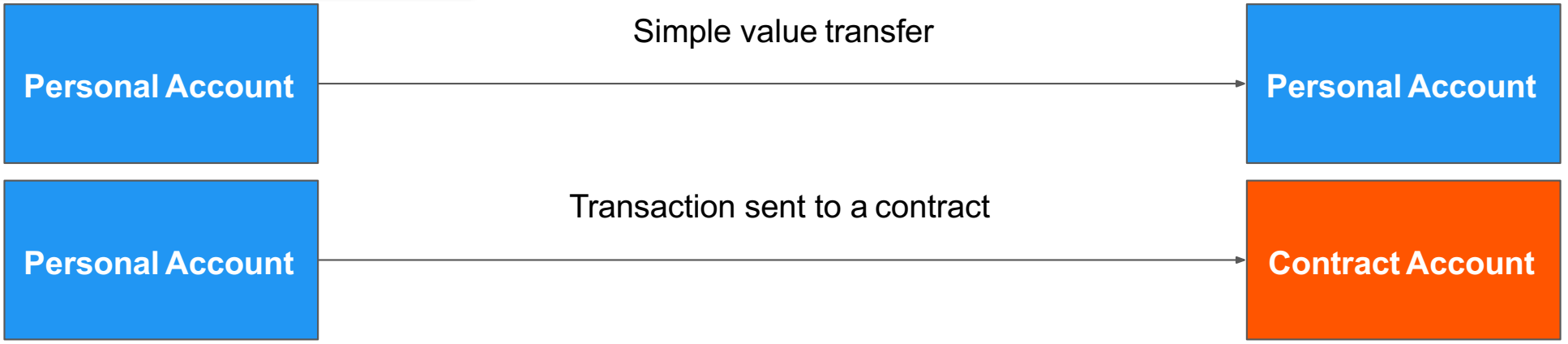




::.. Ethereum (6/11)

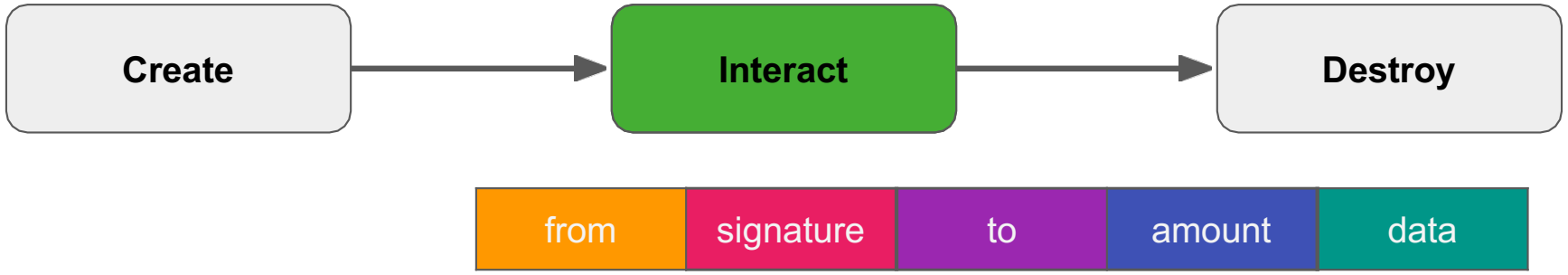
The life cycle of smart contracts consists of 3 states:



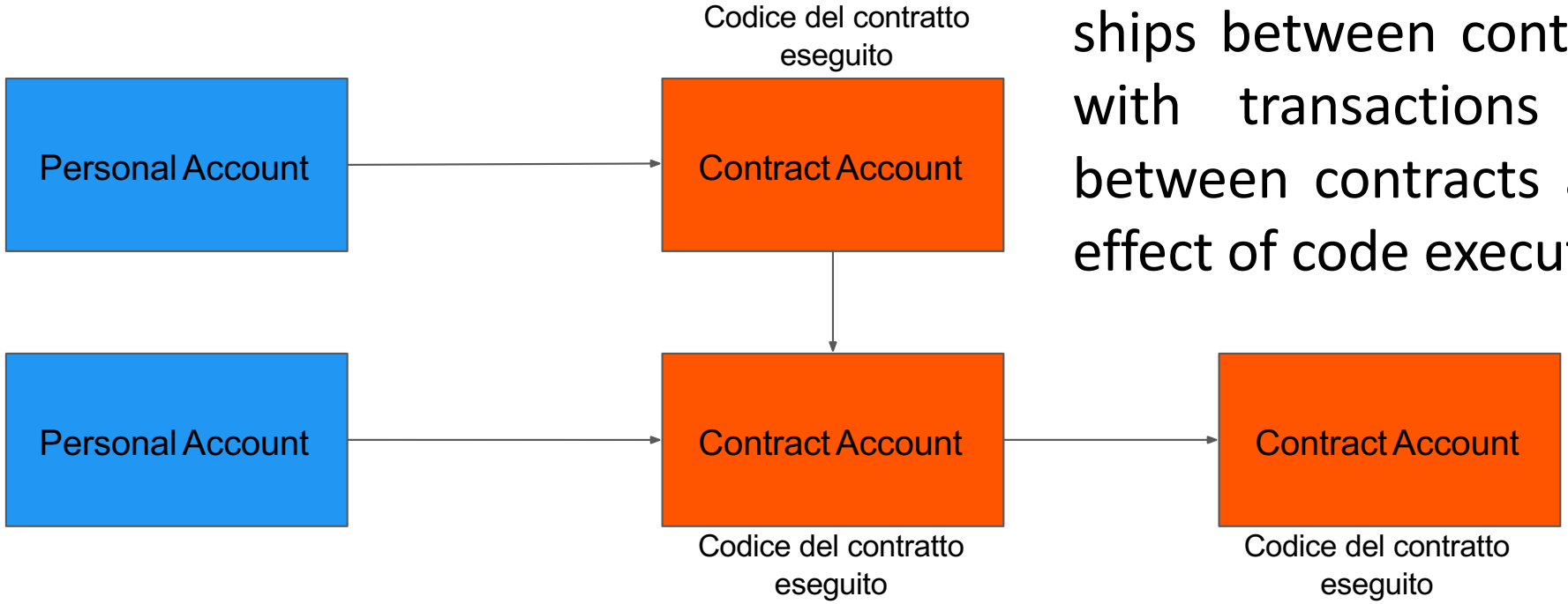


::.. Ethereum (6/11)

The life cycle of smart contracts consists of 3 states:

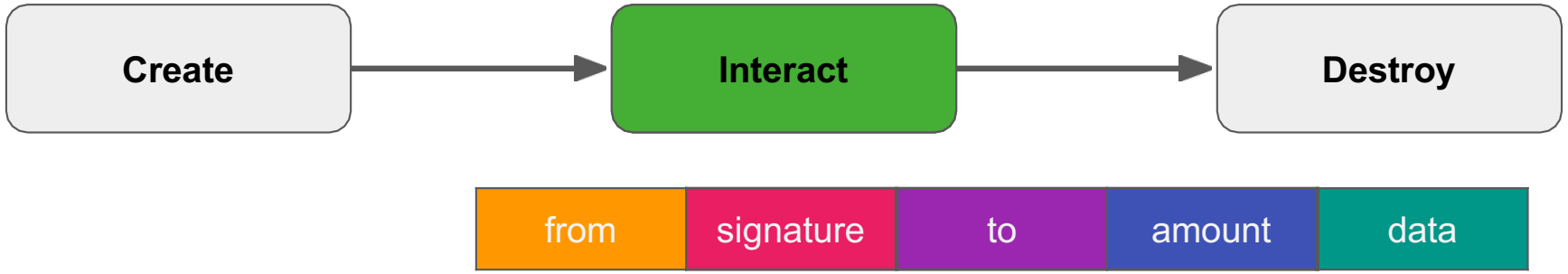


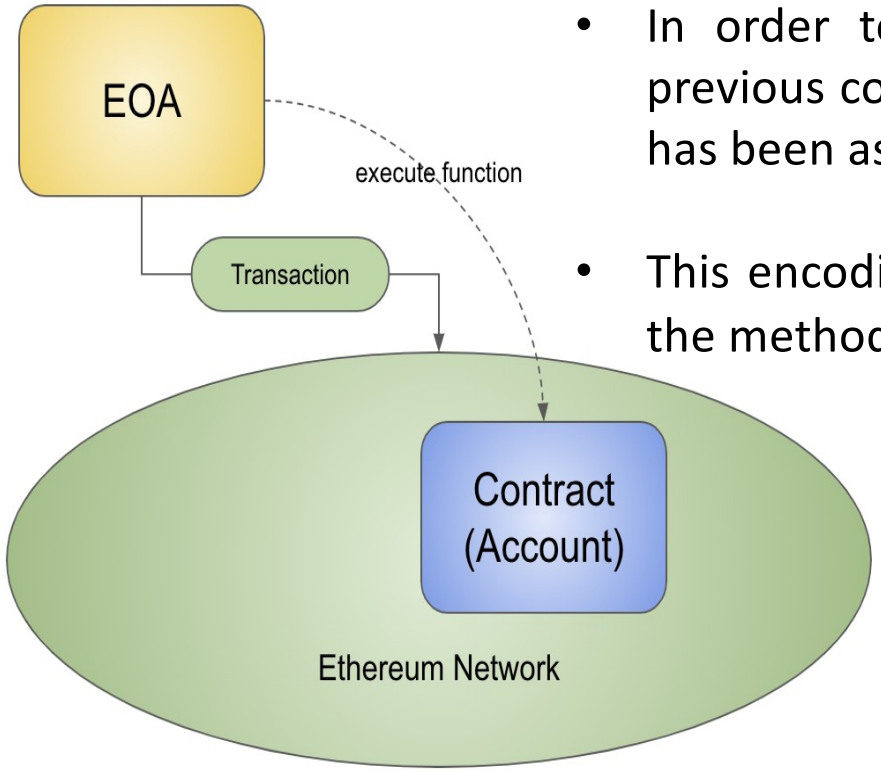
There may be relationships between contracts, with transactions sent between contracts as an effect of code execution.



::.. Ethereum (6/11)

The life cycle of smart contracts consists of 3 states:

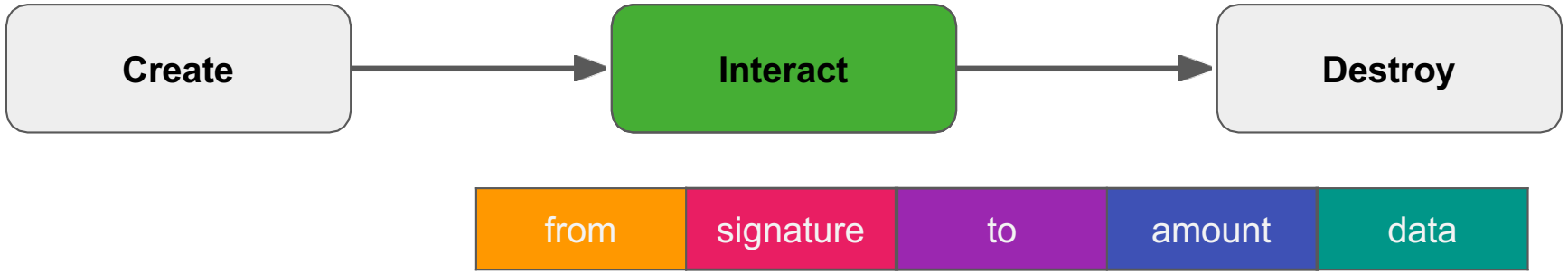


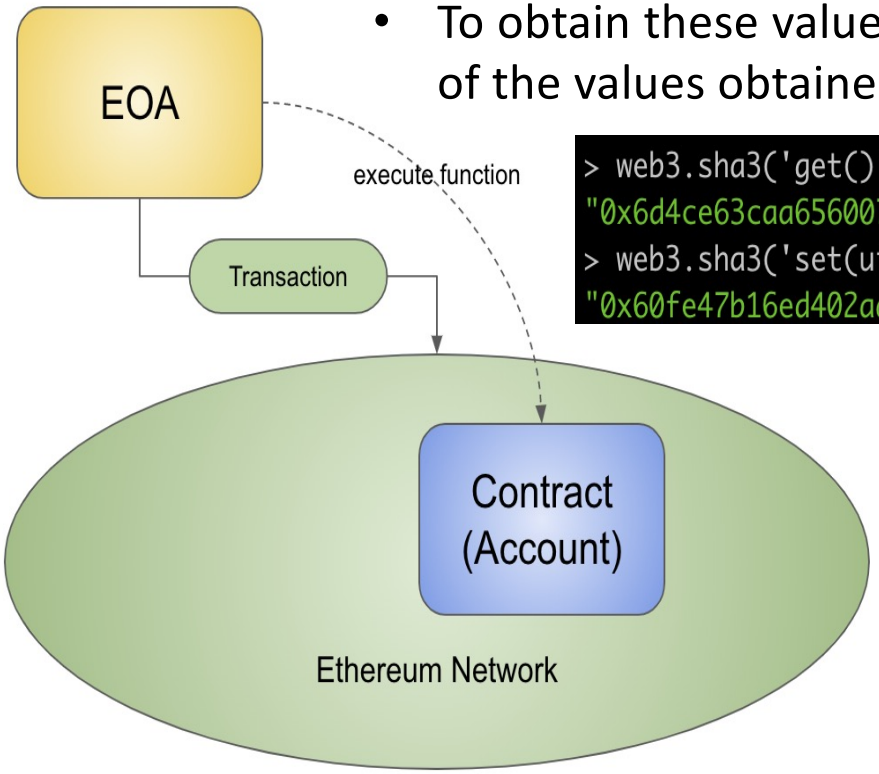


- In order to access the functions implemented in the previous contract it is necessary to obtain the coding that has been assigned to it in the bytecode;
- This encoding is achieved by applying a hash function to the method signature.

::.. Ethereum (6/11)

The life cycle of smart contracts consists of 3 states:





- To obtain these values it is necessary to consider the first 4 bytes of the values obtained from the hash of the functions:

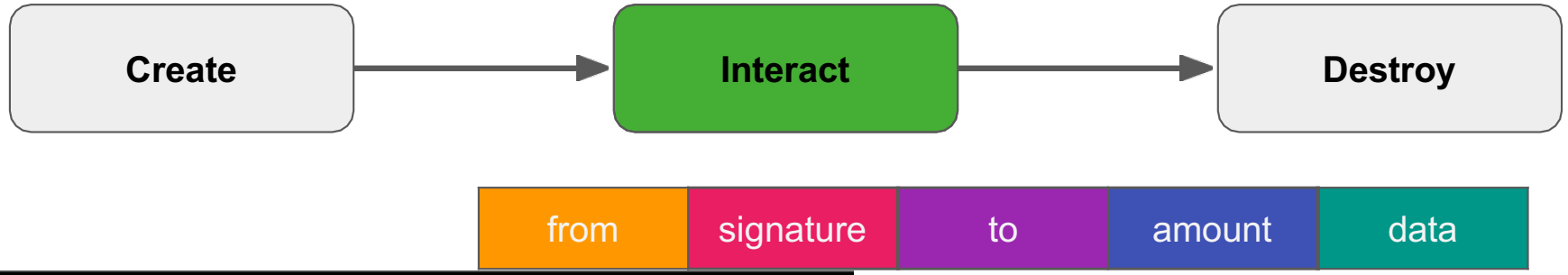
```
> web3.sha3('get()')
"0x6d4ce63caa65600744ac797760560da39ebd16e8240936b51f53368ef9e0e01f"
> web3.sha3('set(uint256)')
"0x60fe47b16ed402aae66ca03d2bfc51478ee897c26a1158669c7058d5f24898f4"
```

Contract's Bytecode

```
"6080604052348015610010576000080fd5b5060df8061001f6000396000f30060806
040526004361060495760003357c0100000000000000000000000000000000000000
0000000000000000900463ffffffff16806360fe47b114604e5780636d4ce63c146
078575b600080fd5b348015605957600080fd5b506076600480360381019080803590
6020019092919050505060a0565b005b348015608357600080fd5b50608a60aa565
b60405180828152602001915050604051809103905b8060008190555050565b600
080549050905600a165627a7a7230582080122bb351e6e2c021f1c56c0c5933087e7
62ea6e7a3360b902b39cbed5a38f10029"
```

::.. Ethereum (6/11)

The life cycle of smart contracts consists of 3 states:




```
> var contractAddress = "0xa8e28f1a7031968fb830e5a70c4b246b07f64d2a"
undefined
> eth.call({
...... to: contractAddress,
...... data: "0x6d4ce63c"
...... })
"0x0000000000000000000000000000000000000000000000000000000000000000"
```

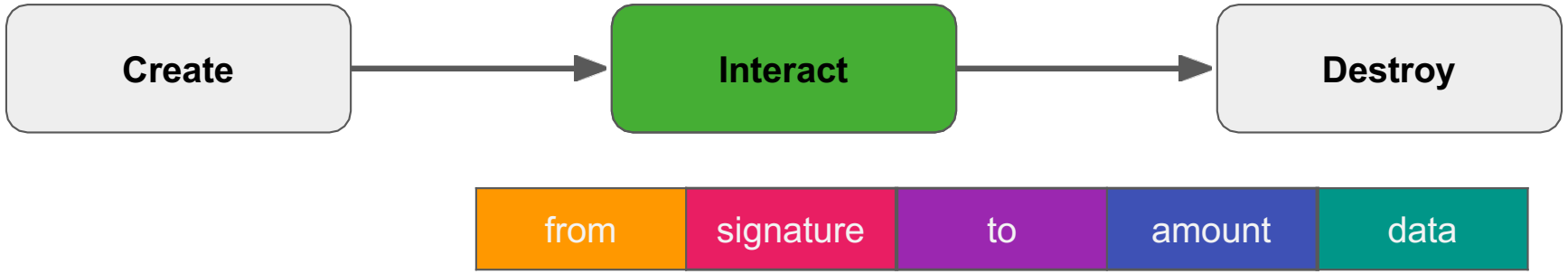
```
> var newValue = "0000000000000000000000000000000000000000000000000000000000000
00FF"
undefined
> eth.sendTransaction({
...... from: eth.accounts[0],
...... to: contractAddress,
...... data: "0x60fe47b1" + newValue
...... })
"0x221bad932cd4c6135b46c926eda9f1d234a6fd8def5ad89fd5c7b549a7be8830"
>
> eth.call({
...... to: contractAddress,
...... data: "0x6d4ce63c"
...... })
"0x00000000000000000000000000000000000000000000000000000000000000ff"
```

- Making a call verifies that the value contained in the storedData variable is 0;
- A new value is loaded via the sendTransaction;
- By repeating the call, the updated value is obtained.

- Calls are transactions executed directly by the VM and not reported in the Blockchain.

::.. Ethereum (6/11)

The life cycle of smart contracts consists of 3 states:



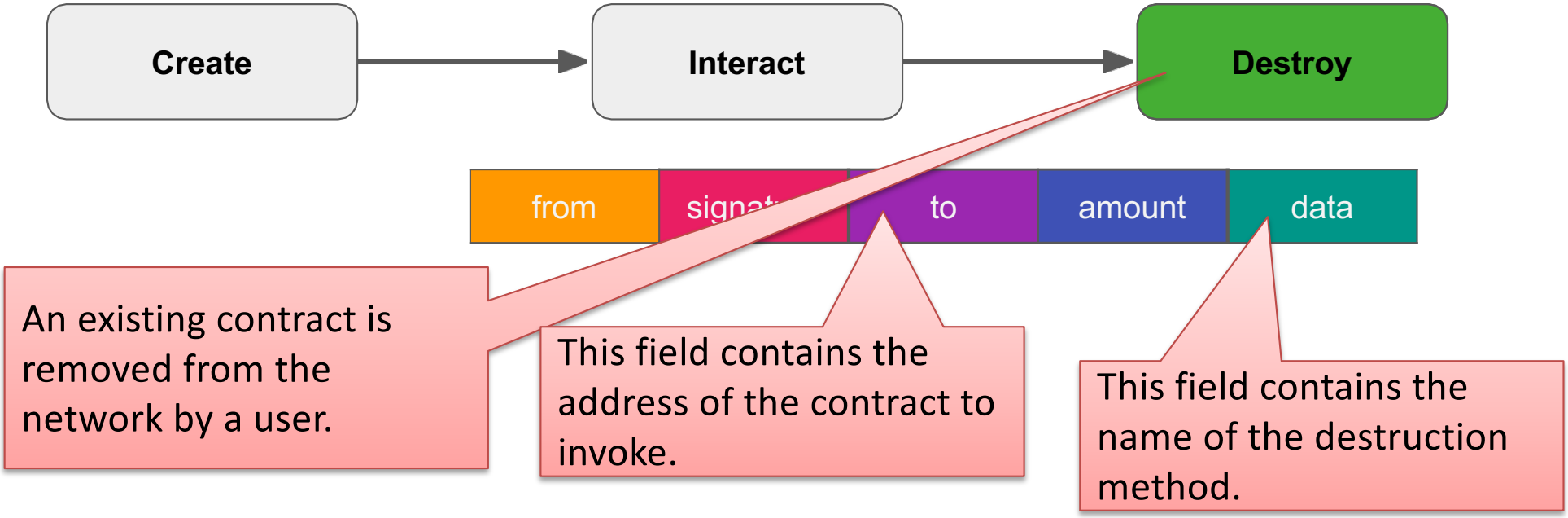

```
> eth.getTransaction("0x221bad932cd4c6135b46c926eda9f1d234a6fd8def5ad89fd5c7b5
49a7be8830")
{
  blockHash: "0x57cd471a42d2c23860dcda5cb32867cb19d1d3f3a044b7dd0312a45859
21",
  blockNumber: 2,
  from: "0x747e967c24abec02b7243e3287cc5ec0f4534a89",
  gas: 90000,
  gasPrice: 20000000000,
  hash: "0x221bad932cd4c6135b46c926eda9f1d234a6fd8def5ad89fd5c7b549a7be883
  input: "0x60fe47b1000000000000000000000000000000000000000000000000000000
0000ff",
  nonce: 1,
  to: "0xa8e28f1a7031968fb830e5a70c4b246b07f64d2a",
  transactionIndex: 0,
  value: 0
}
```

```
> eth.getTransactionReceipt("0x221bad932cd4c6135b46c926eda9f1d234a6fd8def5ad89
fd5c7b549a7be8830")
{
  blockHash: "0x57cd471a42d2c23860dcda5cb32867cb19d1d3f3a044b7dd0312a45859ecb6
21",
  blockNumber: 2,
  contractAddress: null,
  cumulativeGasUsed: 41669,
  gasUsed: 41669,
  logs: [],
  logsBloom: "0x0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000",
  status: "0x1",
  transactionHash: "0x221bad932cd4c6135b46c926eda9f1d234a6fd8def5ad89fd5c7b549
a7be8830",
  transactionIndex: 0
}
```

Gas change following the execution of call transactions.

::.. Ethereum (6/11)

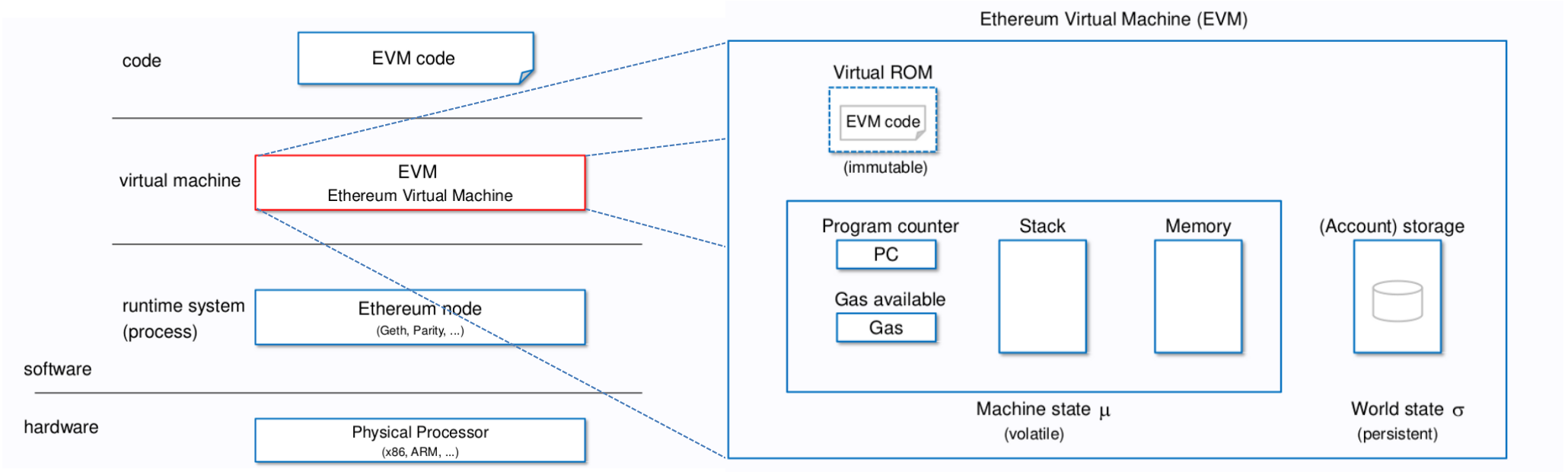
The life cycle of smart contracts consists of 3 states:



::.. Ethereum (6/11)

The life cycle of smart contracts consists of 3 states:

EVM is an isolated (sandboxed) contract execution environment with a stack, internal 32-byte parallelism, and opcodes that form a near-Turing-complete machine.



::.. Ethereum (7/11)

Solidity is an object-oriented, static-typed programming language for writing applications that implement business logic embedded in smart contracts on Ethereum and other blockchain platforms.

- It is designed around the ECMAScript syntax but differs in static typing and variable return types.
- It also supports user-defined complex types, such as structs and enumerations, which allow you to group related data types.
- Contracts support inheritance, including multiple inheritance with C3 linearization.
- A binary application interface (ABI) was also introduced that facilitates multiple type-safe functions within a single contract.

::.. Ethereum (7/11)

Solidity is an object-oriented, static-typed programming language for writing applications that implement busin logic embedded in smart contracts on Ethereum and other blockch

Type checking of contract member data structures occurs at compile time, not at run time as with dynamically typed languages.

typing and variable return types.

- It also supports user-defined complex types, such as structs and enumerations, which allow you to group related data types.
- Contracts support inheritance, including multiple inheritance with C3 linearization.
- A binary application interface (ABI) was also introduced that facilitates multiple type-safe functions within a single contract.

::.. Ethereum (7/11)

Solidity is an object-oriented, static-typed programming language for writing applications that implement business logic embedded in smart contracts on Ethereum and other blockchain platforms.

- It is designed around the ECMAScript syntax but differs in static typing and variable return types.
- It also supports user-defined complex types, structs and C3 linearization.
- A binary application interface (ABI) was also introduced that facilitates multiple type-safe functions within a single contract.

ECMAScript is the technical specification of a scripting language, intended to ensure the interoperability of Web pages between different Web browsers. The best known implementation of this language is JavaScript.

::.. Ethereum (7/11)

Solidity is an object-oriented, static-typed programming language for writing applications that implement business logic embedded in smart contracts on Ethereum and other blockchain platforms.

- It is designed around the ECMAScript syntax but differs in static typing and variable return types.
- It also supports user-defined comp… …s, such as structs and …
- …
- A binary application interface (ABI) was also introduced that facilitates multiple type-safe functions within a single contract.

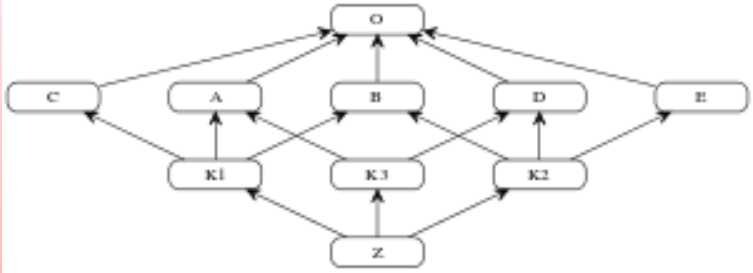
A variable return parameter accepts zero or more values of a specified type, and the function returns a variable amount of values.

::.. Ethereum (7/11)

Solidity is an object-oriented, static-typed programming language for

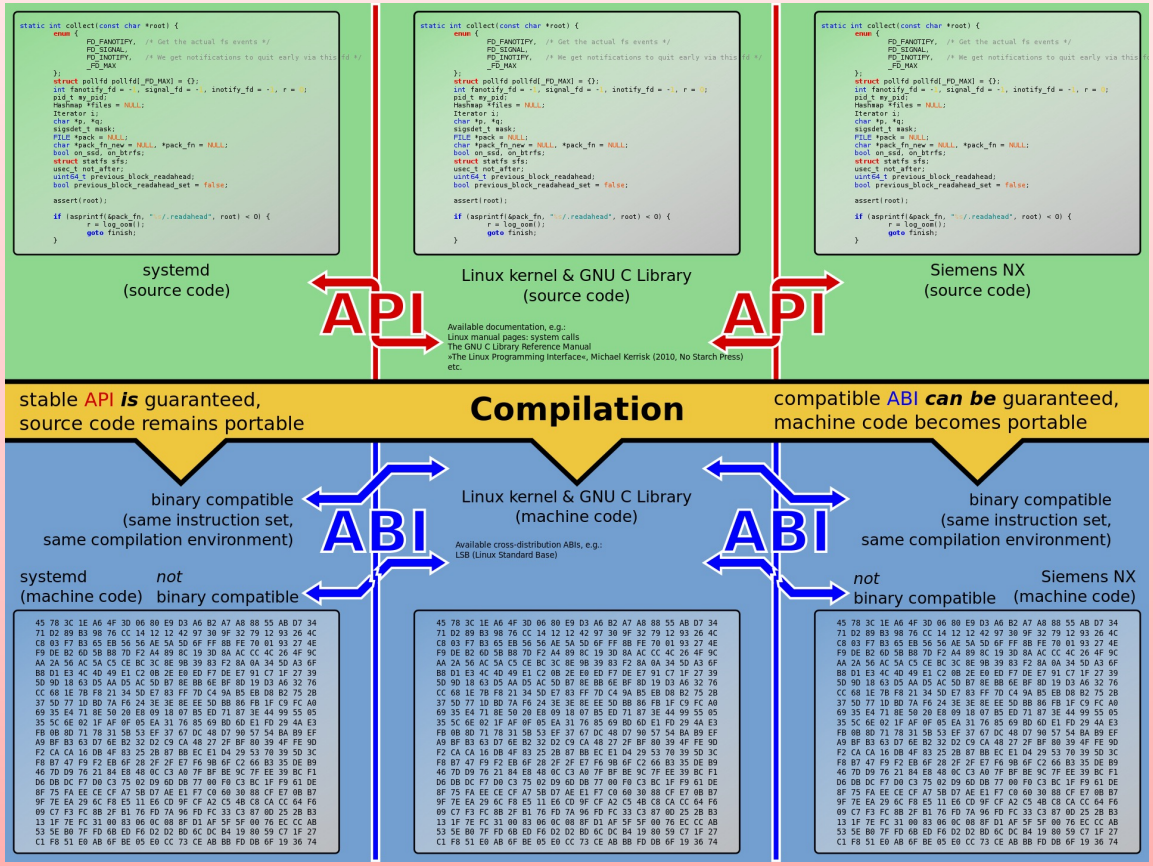
The C3 linearization is an algorithm used to obtain the order in which methods must be inherited in the presence of multiple inheritance.
The linearization is the sum of the class plus a merge of the linearizations of its parents and a list of the parents themselves. Merge is performed by selecting the first list header that does not appear in any other queue. The selected item is removed and added to the output list. If a valid header cannot be selected, then the merge is impossible to calculate due to inconsistent dependency ordering and no linearization is possible.

- Contracts support inheritance, including multiple inheritance with C3 linearization.



L(O) := [O]
L(A) := [A] + merge(L(O), [O]) = [A] + merge([O], [O]) = [A, O]
L(K1) := [K1] + merge(L(A), L(B), L(C), [A, B, C]) = [K1] + merge([A, O], [B, O], [C, O], [A, B, C]) = [K1, A] + merge([O], [B, O], [C, O], [B, C]) = [K1, A, B] + merge([O], [O], [C, O], [C]) = [K1, A, B, C] + merge([O], [O], [O]) = [K1, A, B, C, O]

::.. Ethereum (7/11)



An ABI is an interface between two binary program modules, and defines how data structures or routines in machine code are accessed. It is expressed in a low-level, hardware-dependent format; in contrast, an API defines this access in source code, in a relatively high-level, hardware-independent format.

- A binary application interface (ABI) was also introduced that facilitates multiple type-safe functions within a single contract.

::.. Ethereum (8/11)

It is possible to write applications in Solidity using the Remix IDE.

https://remix.ethereum.org

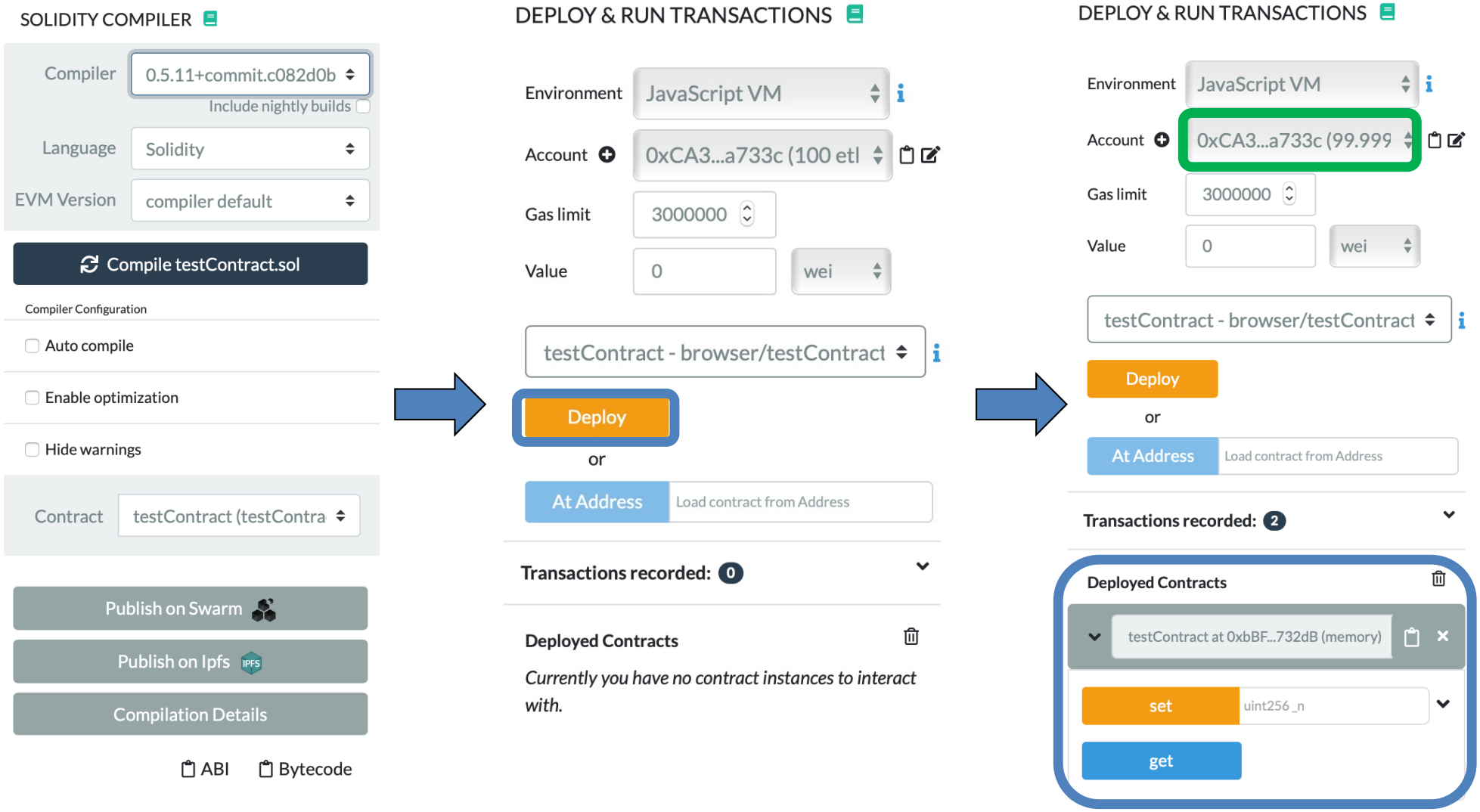
First example of a smart contract:

testContract.sol

```
1  pragma solidity ^0.5.1;
2
3  contract testContract {
4
5      uint amount=13;
6
7      function set(uint _n) public {
8          amount = _n;
9      }
10
11     function get() view public returns (uint) {
12         return amount;
13     }
14 }
```

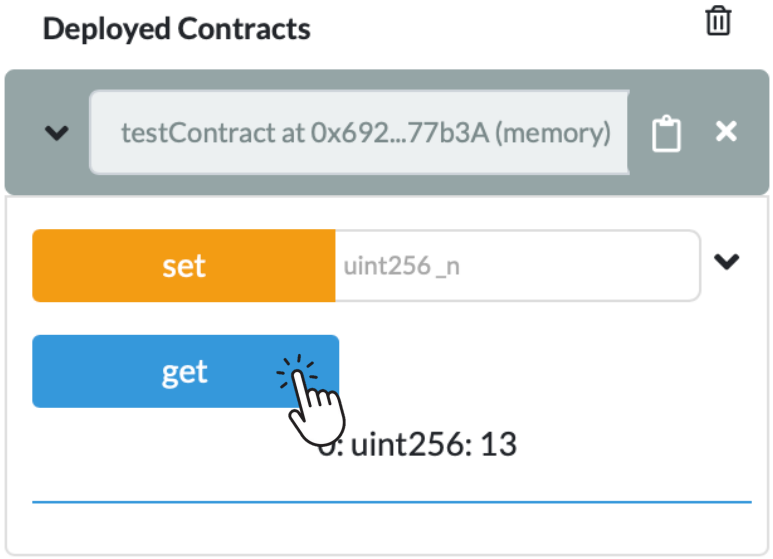
::.. Ethereum (8/11)

Deployment of the smart contract:

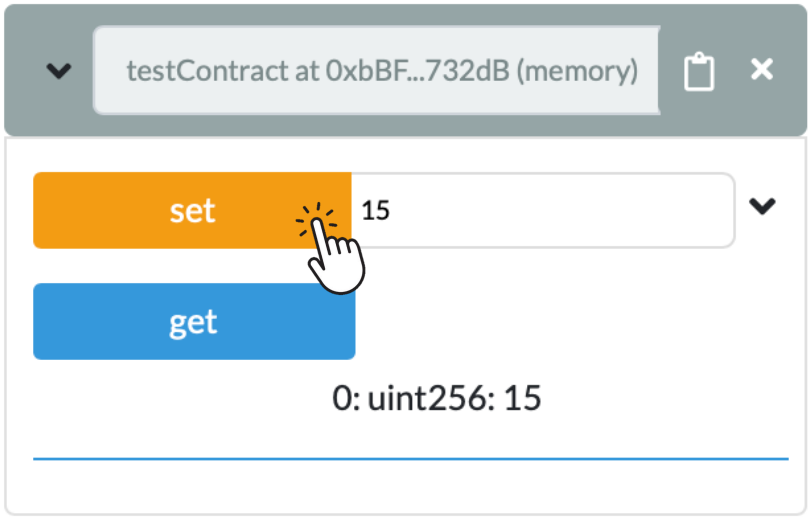


::.. Ethereum (9/11)

Example of get



Example of set and a consequent get



::.. Ethereum (10/11)

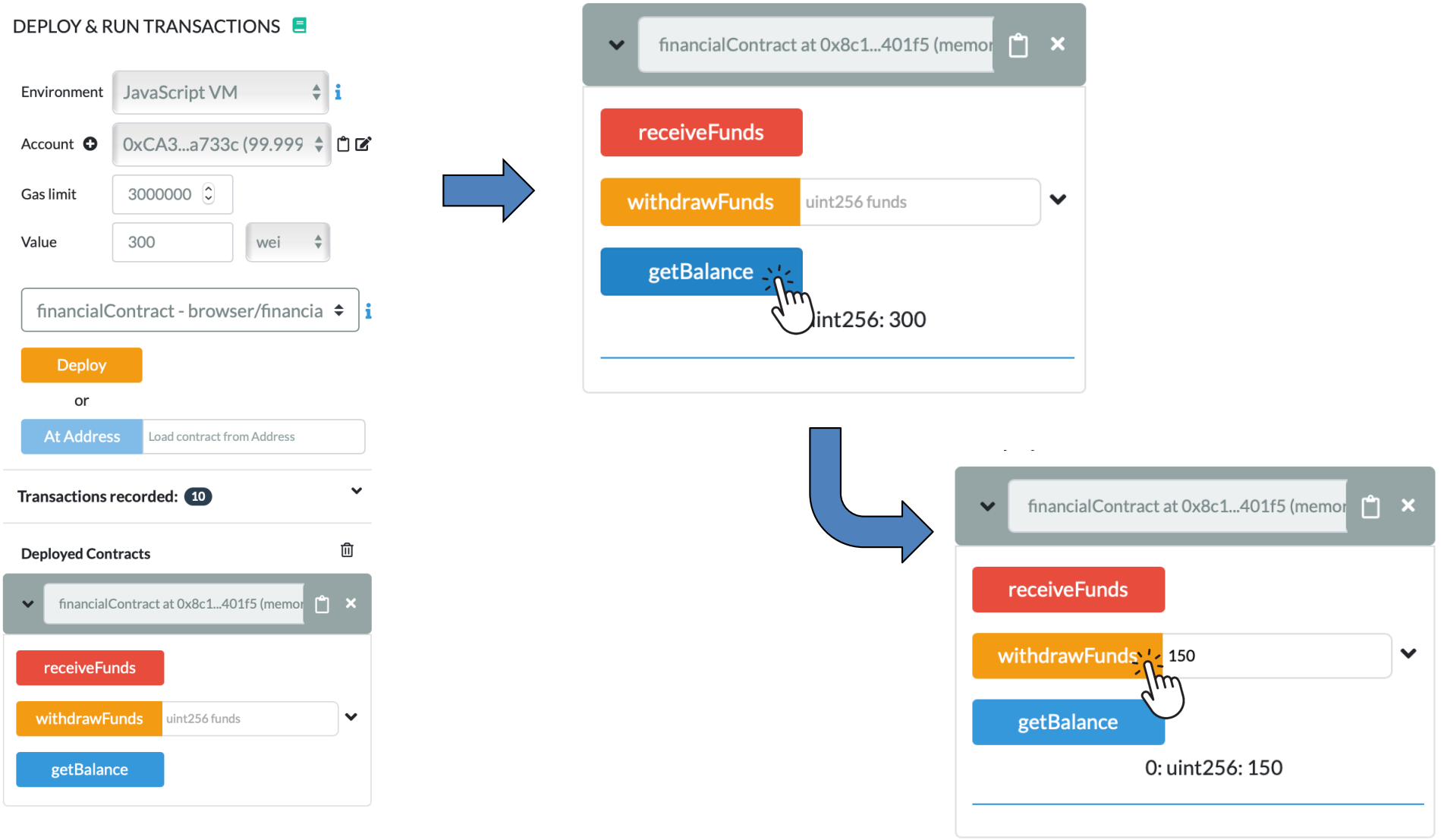
Second example of a smart contract:

testContract.sol | financialContract.sol

```
1  pragma solidity ^0.5.1;
2
3  contract financialContract {
4
5      address payable issuer;
6
7      constructor() public{
8          issuer = msg.sender;
9      }
10
11     function receiveFunds() payable public{
12
13     }
14
15     function getBalance() view public returns (uint){
16         return address(this).balance;
17
18     }
19
20     function withdrawFunds(uint funds) public{
21         assert(issuer == msg.sender);
22         issuer.transfer(funds);
23     }
24 }
```

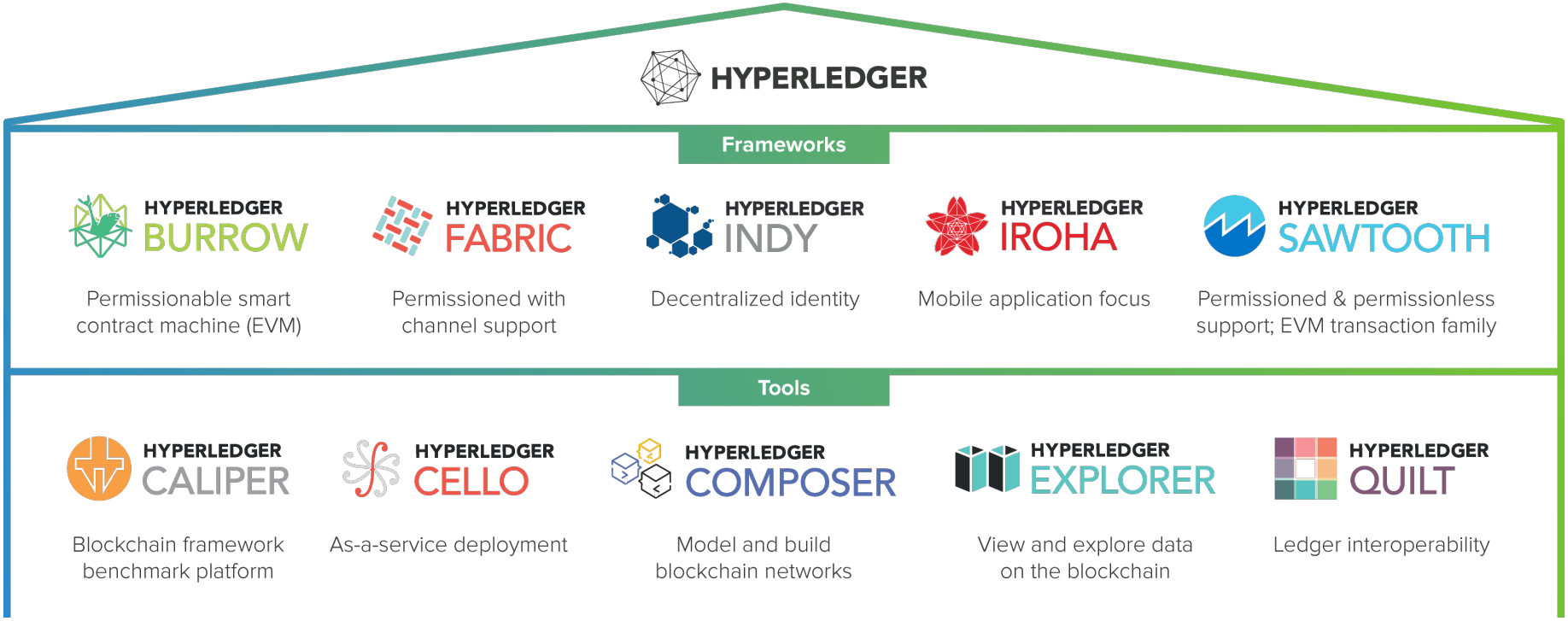
::.. Ethereum (11/11)

Deployment of the smart contract:



::.. Hyperledger Fabric (1/16)

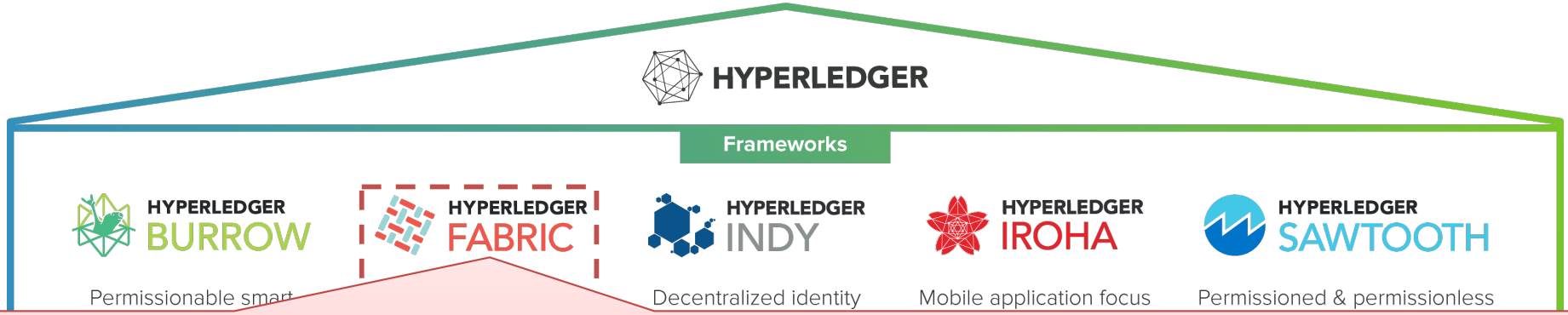
Hyperledger is a family of open source blockchain projects launched in December 2015 by the Linux Foundation, to support the collaborative development of DLT solutions.



It is divided into frameworks for the construction of blockchain platforms, and tools to support applications and smart contracts.

::.. Hyperledger Fabric (1/16)

Hyperledger is a family of open source blockchain projects launched in December 2015 by the Linux Foundation, to support the collaborative development of DLT solutions.



Hyperledger Fabric is an IBM enterprise project, and consists of a permissioned DLT platform, with some key differences from other popular solutions:

- It has a highly modular and configurable architecture;
- It supports smart contracts (called Chaincode) written in common programming languages, rather than using domain-specific languages;
- The participants are known, rather than being anonymous, by adopting a governance model built on the trust that exists among the participants;
- It supports pluggable consent protocols, to be chosen based on the needs, and without a native cryptocurrency to incentivize expensive mining or to fuel the execution of smart contracts.

::.. Hyperledger Fabric (2/16)

Organizations that take part in building the Hyperledger Fabric network are called "members," each responsible for setting up their peers to participate in the network. These peers must be configured with appropriate cryptographic materials to authenticate their identity or secure communication channels.

- Peers receive transaction requests from clients using an SDK or REST web services to interact with the Hyperledger Fabric network, and are connected to each other by channels that allow for data isolation and confidentiality.
- All peers keep their own unique ledger per channel they are subscribed to, but unlike Ethereum, they have different roles:
    1. Peer endorser: After a request from a client, it validates the transaction and executes the Chaincode by simulating the outcome of the transaction without updating the blockchain. At the end, the Endorser can approve or not the transaction.

::.. Hyperledger Fabric (2/16)

Organizations that take part in building the Hyperledger Fabric network are called "members," each responsible for setting up their peers to participate in the network. These peers must be configured with appropriate cryptographic materials to authenticate their identity or secure communication channels.

- Peers receive transaction requests from clients using an SDK or REST web services to interact with the Hyperledger Fabric network, and are connected to each other by channels that allow for data isolation and confiden
- All peers keep their subscribed to, but unlike Ether y have different roles:
  1. Peer endorser: After a reque from a client, it validates the transaction and executes the Chaincode by simulating the outcome of the transaction without updating the blockchain. At the end, the Endorser can approve or not the transaction.

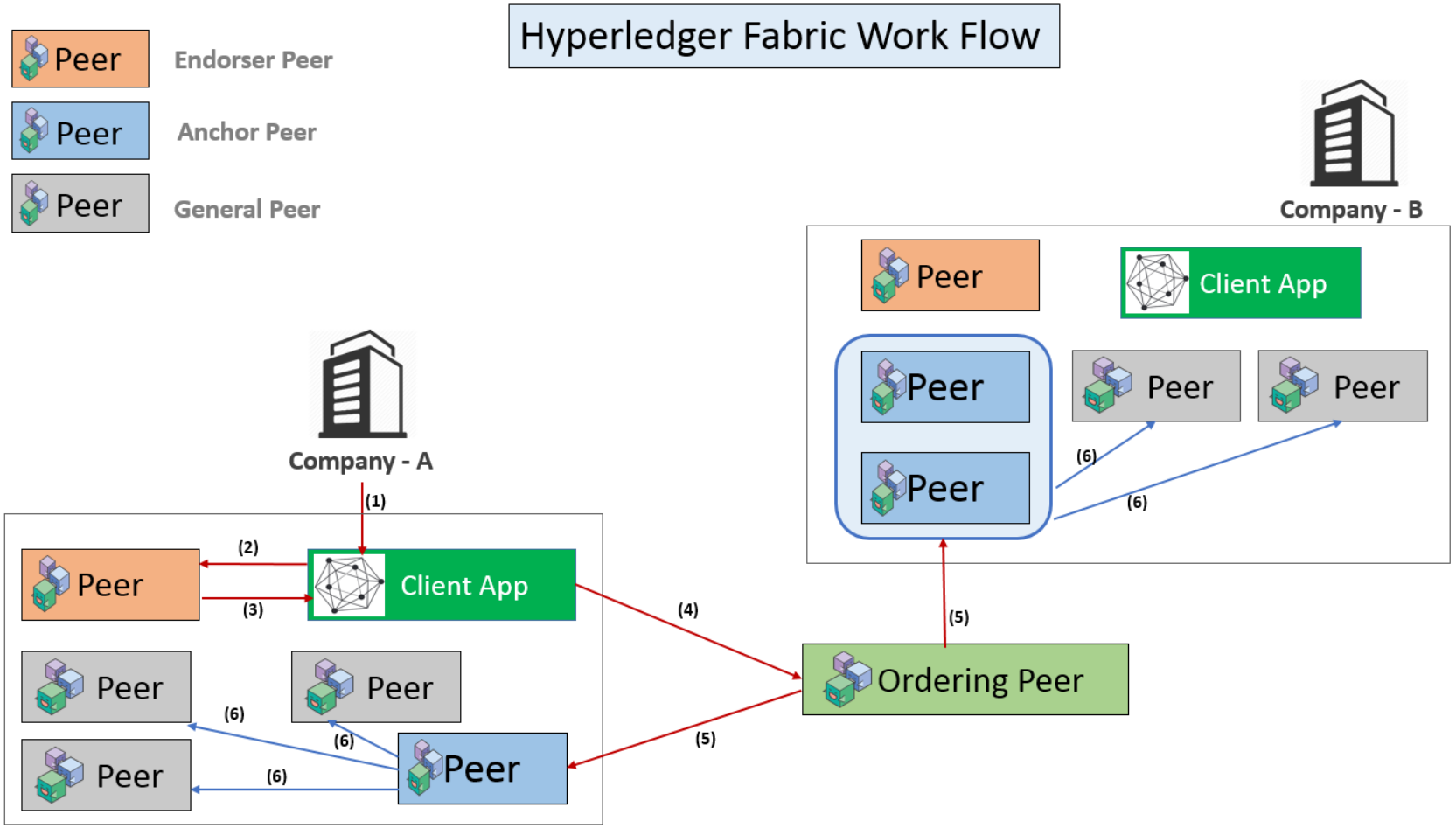
Only the Endorser node performs the Chaincode, so it is not necessary to install it on every node in the network.

::.. Hyperledger Fabric (3/16)

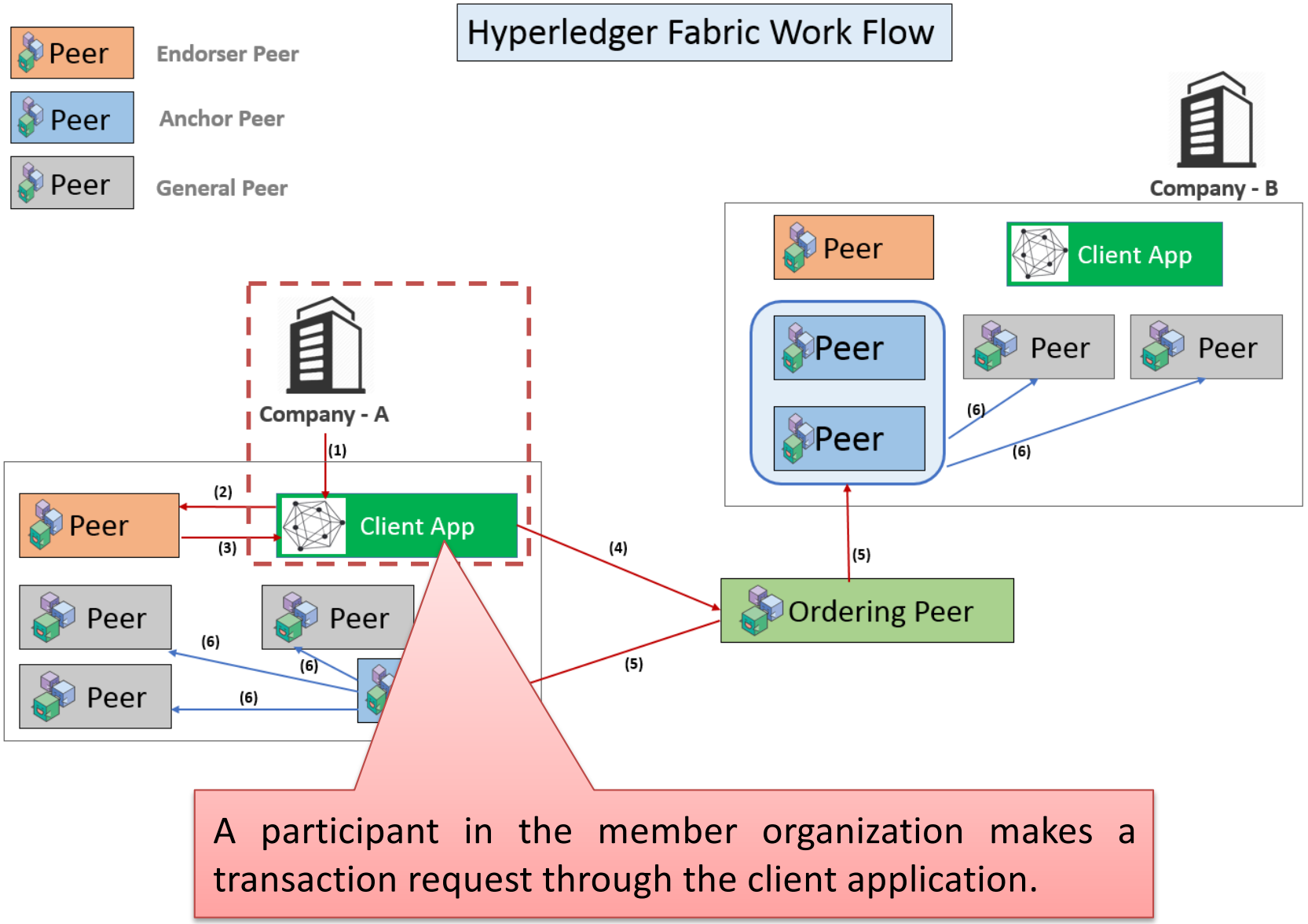
2. Anchor peer – It exchanges updates to other peers in the organization. Channels can be configured among peers, and transactions in a channel are visible only to its participants.
3. Peer orderer: it represents the central element in a channel among peers and is responsible for the state of the consistent state of the ledger by ordering transactions and placing them in new blocks. There are currently two options:
    - Only: a single orderer to use for development, not in production, because it is not very scalable and resilient.
    - Kafka: Apache streaming processing solution that guarantees the reception of messages in the order in which they are sent.

There are two types of transactions: Deploy transactions to create a new chaincode and install it on the Blockchain, and Invoke transactions to call the function of a chaincode.

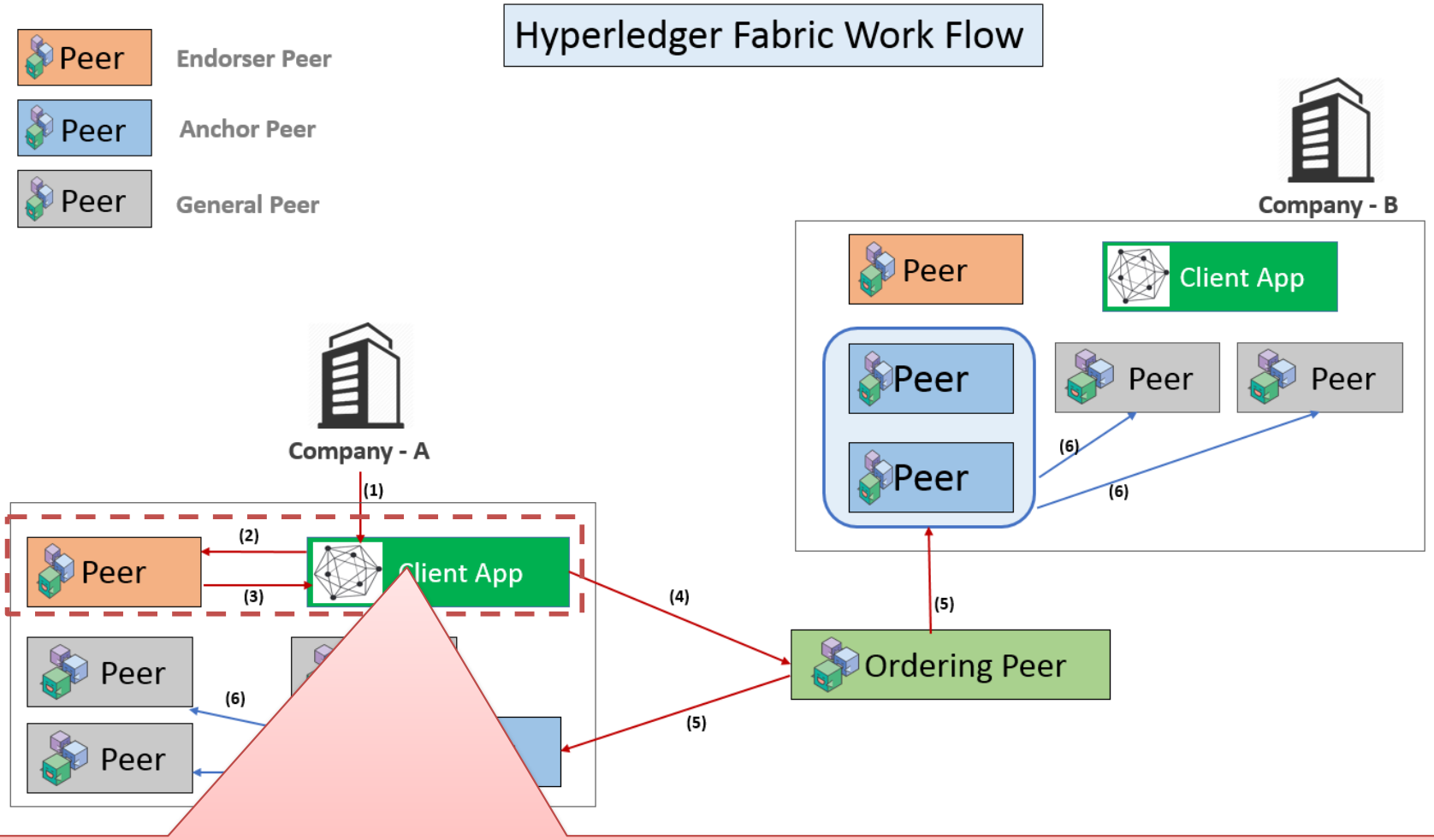
::.. Hyperledger Fabric (4/16)



::.. Hyperledger Fabric (4/16)

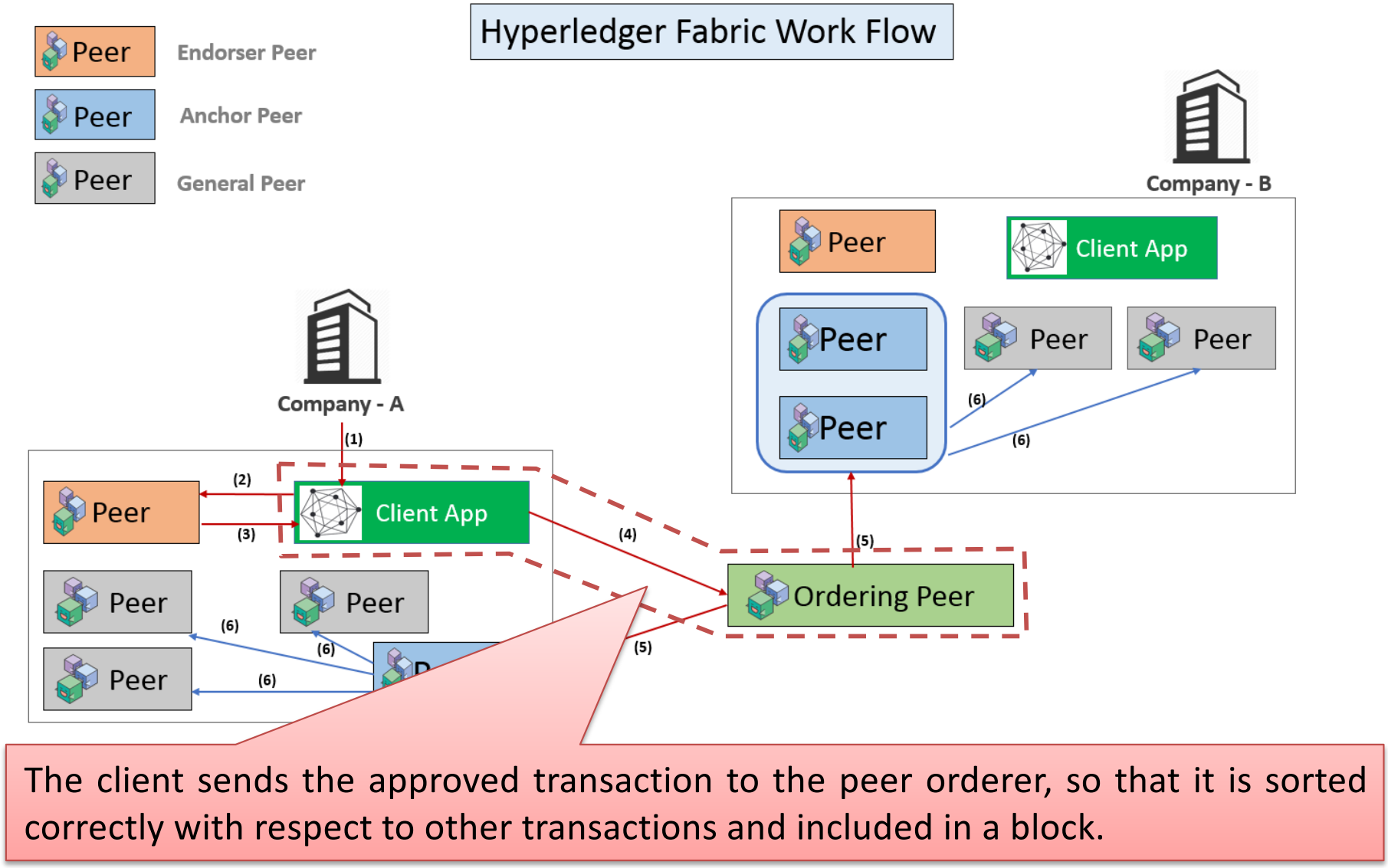


::.. Hyperledger Fabric (4/16)



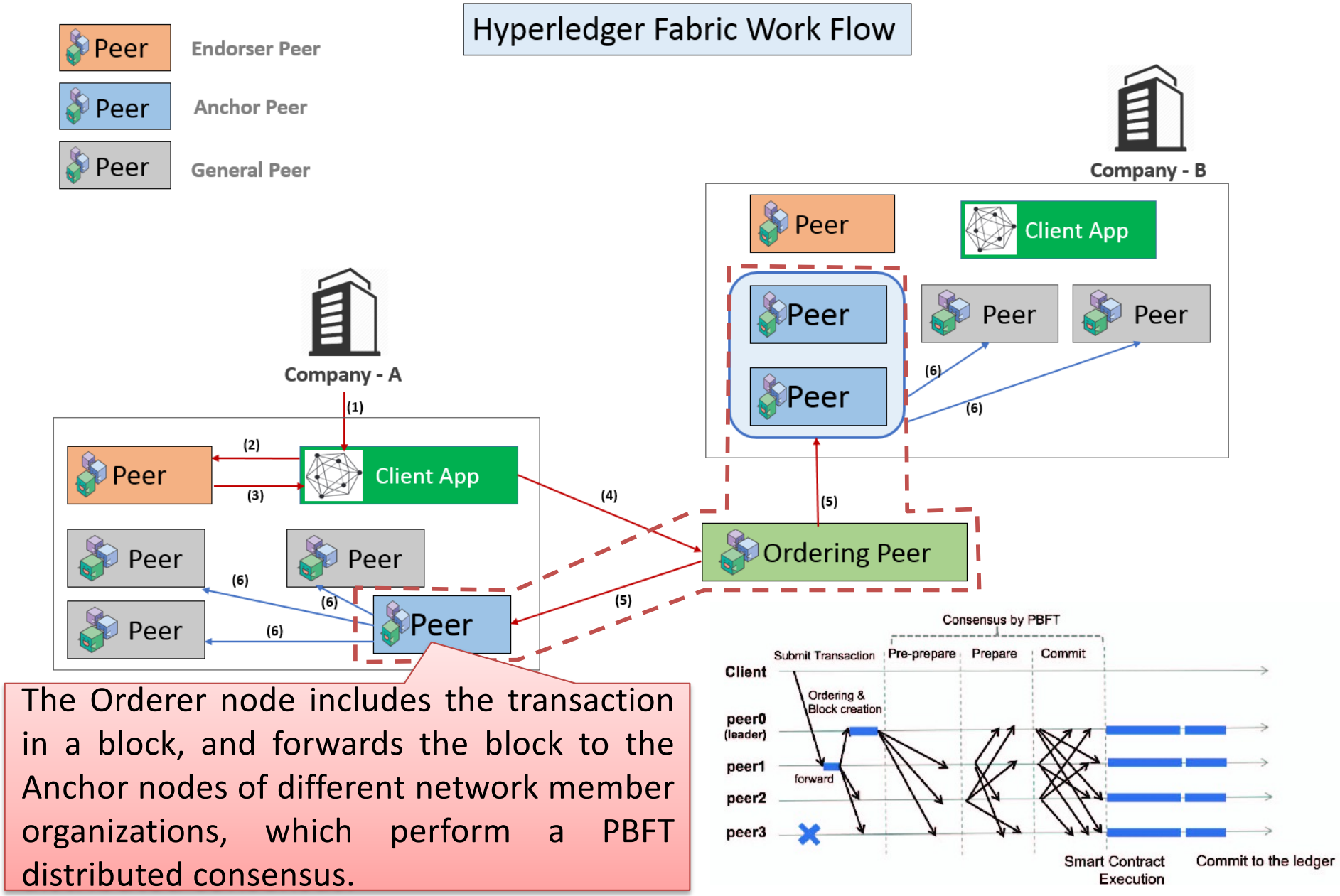
The client application broadcasts the request to the peer Endorser, which checks the certificate details and other details to validate the transaction and optionally executes the Chaincode and returns the Endorsement responses, accepting or rejecting the transaction.

::.. Hyperledger Fabric (4/16)



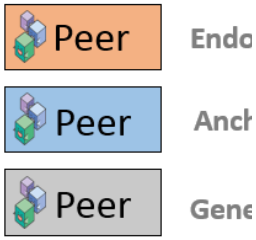
The client sends the approved transaction to the peer orderer, so that it is sorted correctly with respect to other transactions and included in a block.

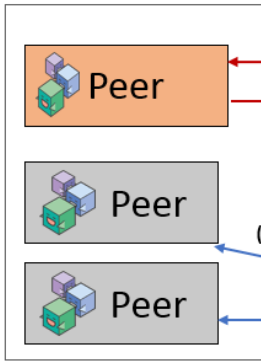
::.. Hyperledger Fabric (4/16)



The Orderer node includes the transaction in a block, and forwards the block to the Anchor nodes of different network member organizations, which perform a PBFT distributed consensus.

::.. Hyperledger Fabric (4/16)

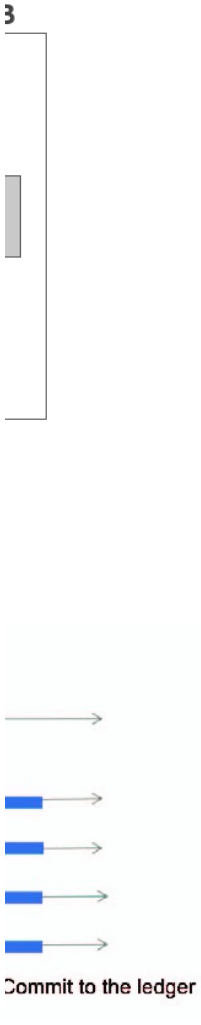




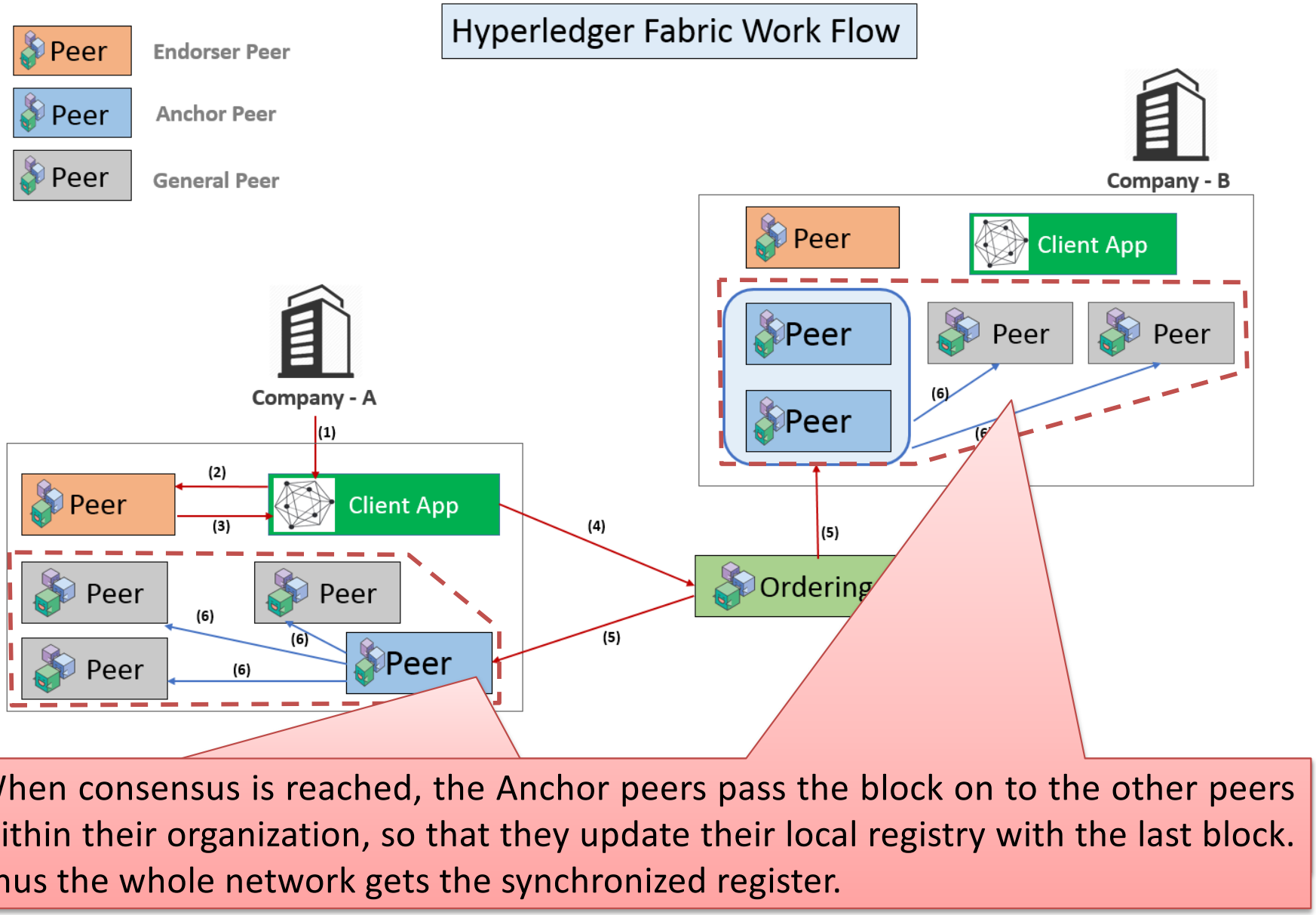
The Orderer n
in a block, an
Anchor nodes
organizations,
distributed consensus.

TABLE 2. COMPARISON OF CONSENSUS ALGORITHMS USED IN HYPERLEDGER FRAMEWORKS

Consensus Algorithm	Consensus Approach	Pros	Cons
Kafka in Hyperledger Fabric Ordering Service	Permissioned voting-based. Leader does ordering. Only in-sync replicas can be voted as leader. ("Kafka," 2017).	Provides crash fault tolerance. Finality happens in a matter of seconds.	While Kafka is crash fault tolerant, it is not Byzantine fault tolerant, which prevents the system from reaching agreement in the case of malicious or faulty nodes.
RBFT in Hyperledger Indy	Pluggable election strategy set to a permissioned, voting-based strategy by default ("Plenum," 2016). All instances do ordering, but only the requests ordered by the master instance are actually executed. (Aublin, Mokhtar & Quéma, 2013)	Provides Byzantine fault tolerance. Finality happens in a matter of seconds.	The more nodes that exist on the network, the more time it takes to reach consensus. The nodes in the network are known and must be totally connected.
Sumeragi in Hyperledger Iroha	Permissioned server reputation system.	Provides Byzantine fault tolerance. Finality happens in a matter of seconds. Scale to petabytes of data, distributed across many clusters (Struckhoff, 2016).	The more nodes that exist on the network, the more time it takes to reach consensus. The nodes in the network are known and must be totally connected.
PoET in Hyperledger Sawtooth	Pluggable election strategy set to a permissioned, lottery-based strategy by default.	Provides scalability and Byzantine fault tolerance.	Finality can be delayed due to forks that must be resolved.



::.. Hyperledger Fabric (4/16)



When consensus is reached, the Anchor peers pass the block on to the other peers within their organization, so that they update their local registry with the last block. Thus the whole network gets the synchronized register.

::.. Hyperledger Fabric (5/16)

A Certificate Authority (CA) issues certificates to allow

- organizations to authenticate themselves on the network;
- client applications to authenticate transaction proposals;
- peers to approve proposals and upload transactions to the ledger if valid.

There can be one or more CAs on the network, and organizations can choose to use their own CA.

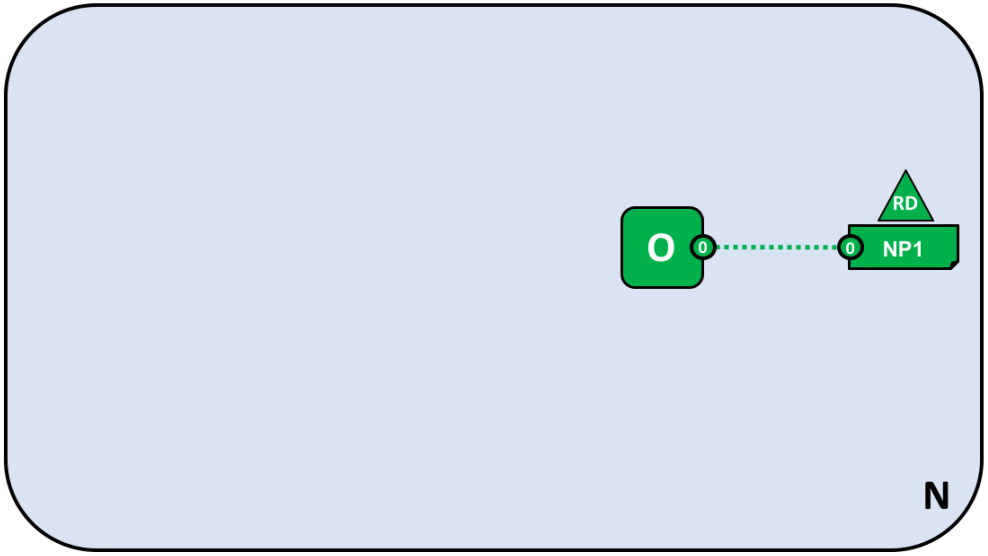
::.. Hyperledger Fabric (5/16)

A Certificate Authority (CA) issues certificates to allow

- organizations to authenticate themselves on the network;
- client applications to authenticate transaction proposals;
- peers to approve proposals and upload transactions to the ledger if valid.

There can be one or more CAs on the network, and organizations can choose to use their own CA.

The network is created by defining the ordering service with the configuration for channels within the network, including access policies and membership information (such as X509 root certificates) for each channel member.



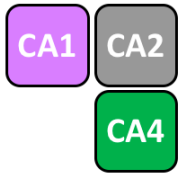
::.. Hyperledger Fabric (5/16)

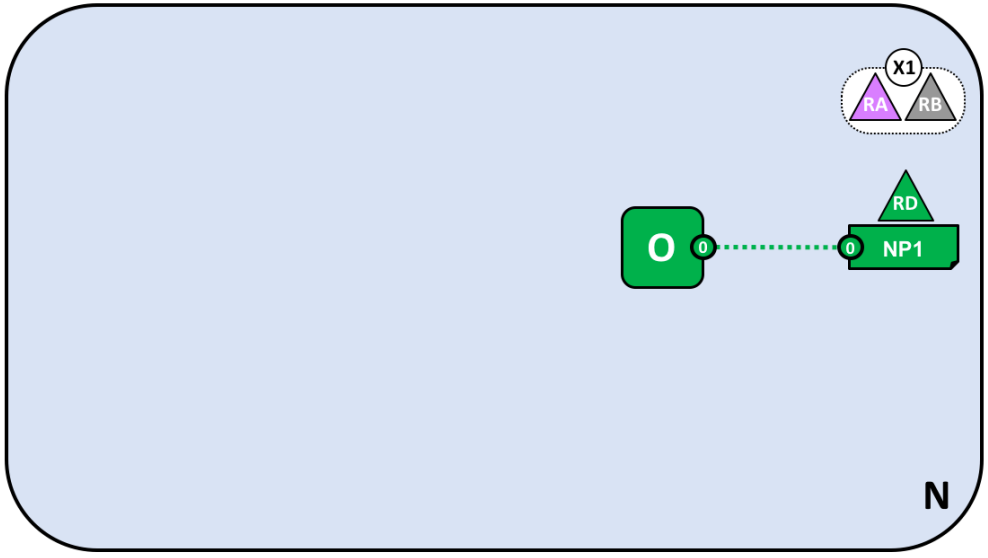
A Certificate Authority (CA) issues certificates to allow

- organizations to authenticate themselves on the network;
- client applications to authenticate transaction proposals;
- peers to approve proposals and upload transactions to the ledger if valid.

There can be one or more CAs on the network, and organizations can choose to use their own CA.

The consortium for the network is established by defining the certificates of the member organizations and their CAs.





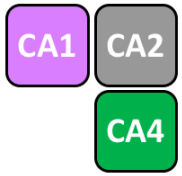
::.. Hyperledger Fabric (5/16)

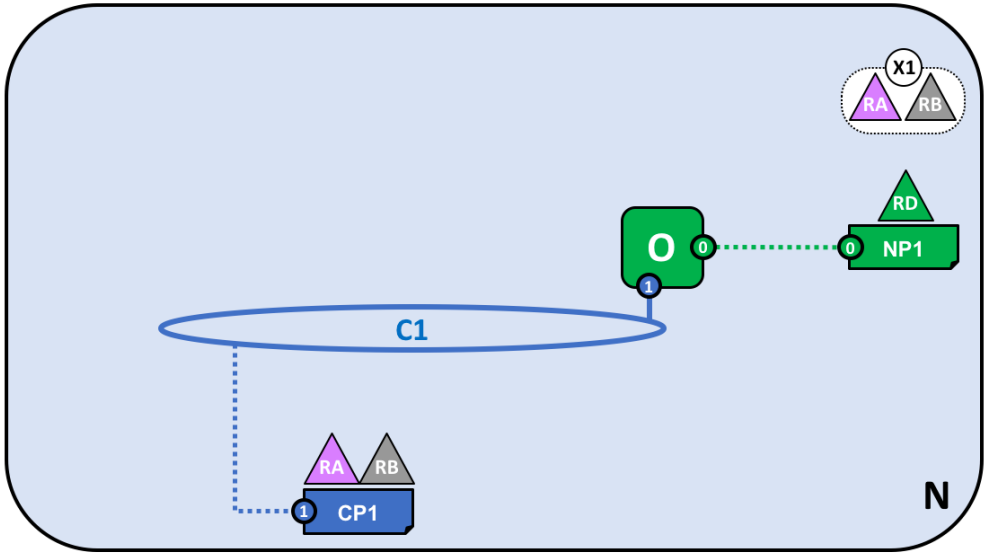
A Certificate Authority (CA) issues certificates to allow

- organizations to authenticate themselves on the network;
- client applications to authenticate transaction proposals;
- peers to approve proposals and upload transactions to the ledger if valid.

There can be one or more CAs on the network, and organizations can choose to use their own CA.

A C1 channel is created by generating the configuration block on the ordering service, which evaluates the validity of the channel configuration. The use of the channels are governed by the criteria with which they are configured.





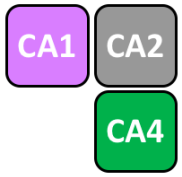
::.. Hyperledger Fabric (5/16)

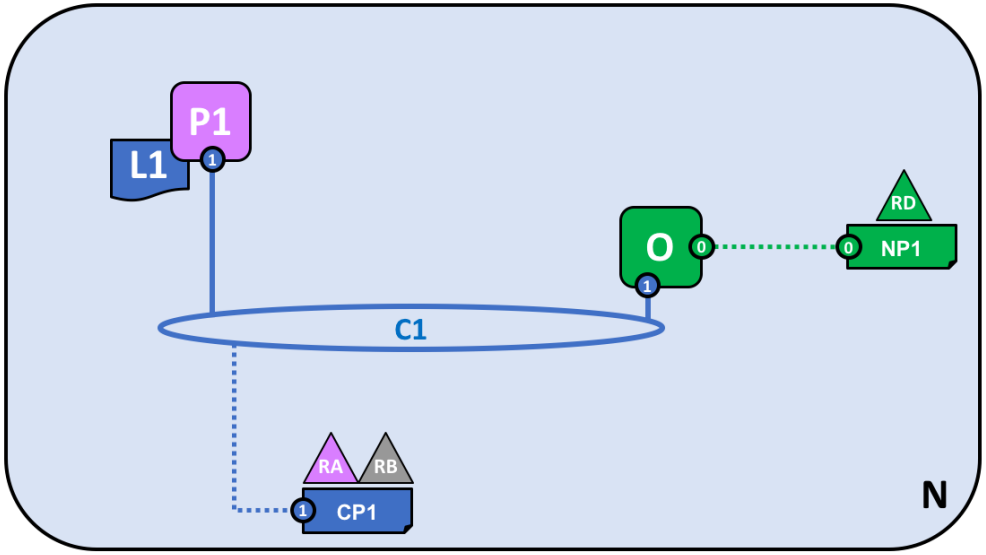
A Certificate Authority (CA) issues certificates to allow

- organizations to authenticate themselves on the network;
- client applications to authenticate transaction proposals;
- peers to approve proposals and upload transactions to the ledger if valid.

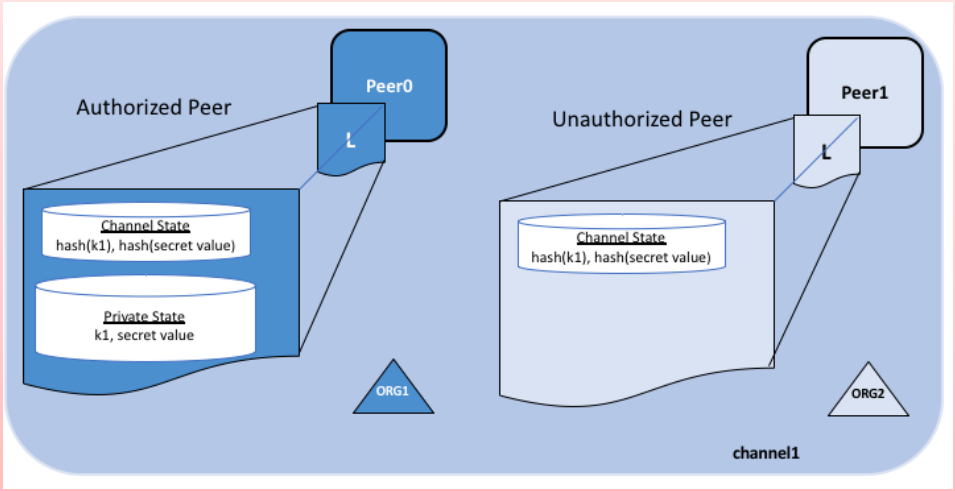
There can be one or more CAs on the network, and organizations can choose to use their own CA.

Peers are joined to channels by the organizations that own them, and there can be multiple peer nodes on channels within the network. P1 is the peer that keeps a copy of the L1 blockchain state for transactions on C1.





::.. Hyperledger Fabric (5/16)



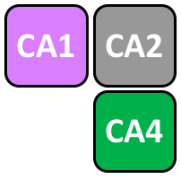
The ledger also contains private information, only accessible to some of the organizations on the channel. They are contained in a SideDB and follow the same endorsement and commit process as public data, but the blockchain only contains its hash.

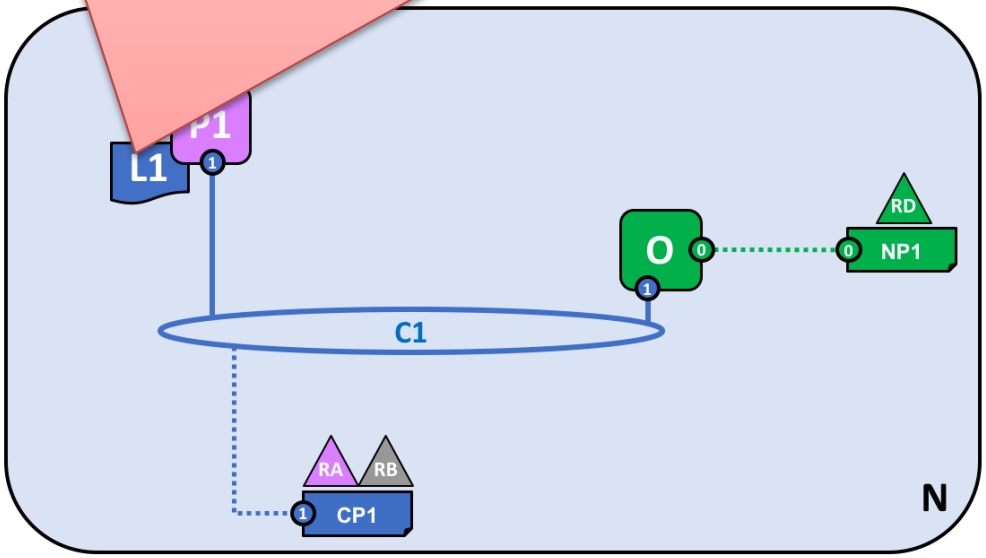
the

tions

can choose to use their own CA.

Peers are joined to channels by the organizations that own them, and there can be multiple peer nodes on channels within the network. P1 is the peer that keeps a copy of the L1 blockchain state for transactions on C1.





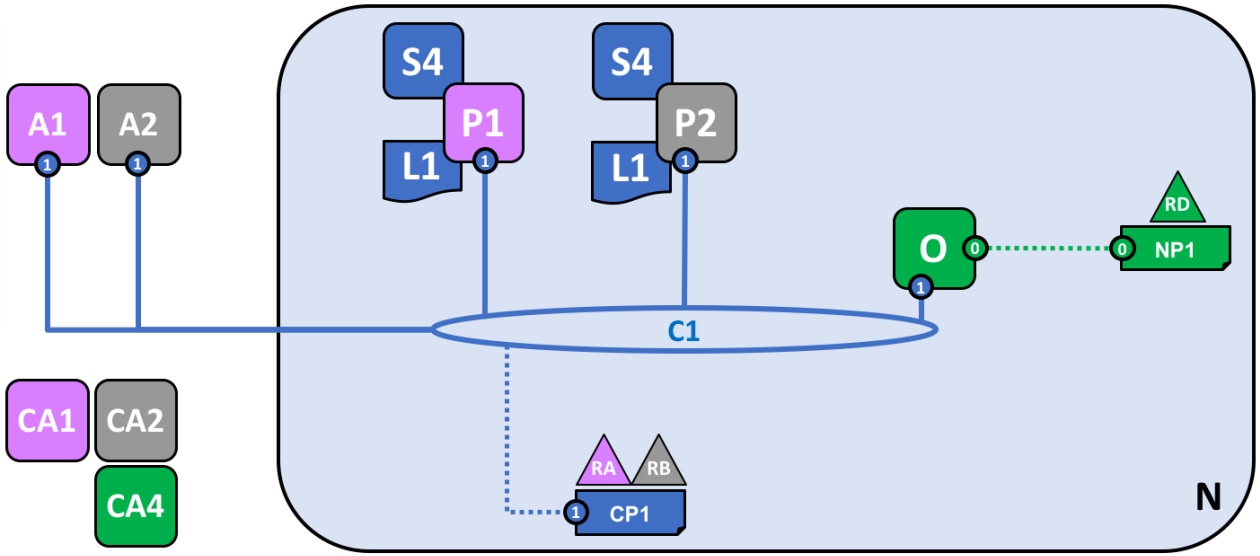
::.. Hyperledger Fabric (5/16)

A Certificate Authority (CA) issues certificates to allow

- organizations to authenticate themselves on the network;
- client applications to authenticate transaction proposals;
- peers to approve proposals and upload transactions to the ledger if valid.

There can be one or more CAs on the network, and organizations can choose to use their own CA.

There are no theoretical limits to the size of a network, and the gossip protocol is used to accommodate a large number of peer nodes on the network.



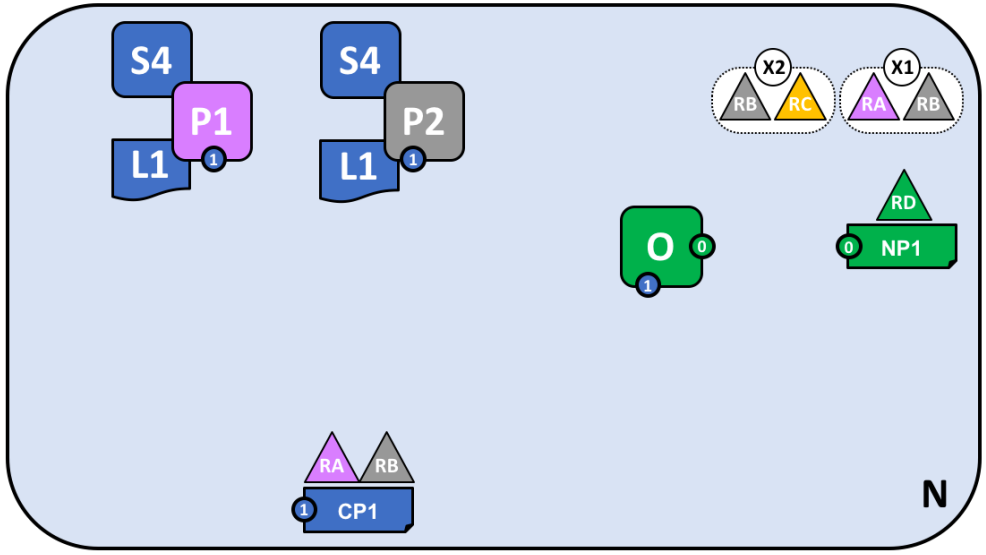
::.. Hyperledger Fabric (5/16)

A Certificate Authority (CA) issues certificates to allow

- organizations to authenticate themselves on the network;
- client applications to authenticate transaction proposals;
- peers to approve proposals and upload transactions to the ledger if valid.

There can be one or more CAs on the network, and organizations can choose to use their own CA.

It is possible to define another consortium, where a third organization interacts with the second.



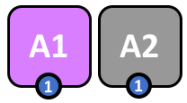
::.. Hyperledger Fabric (5/16)

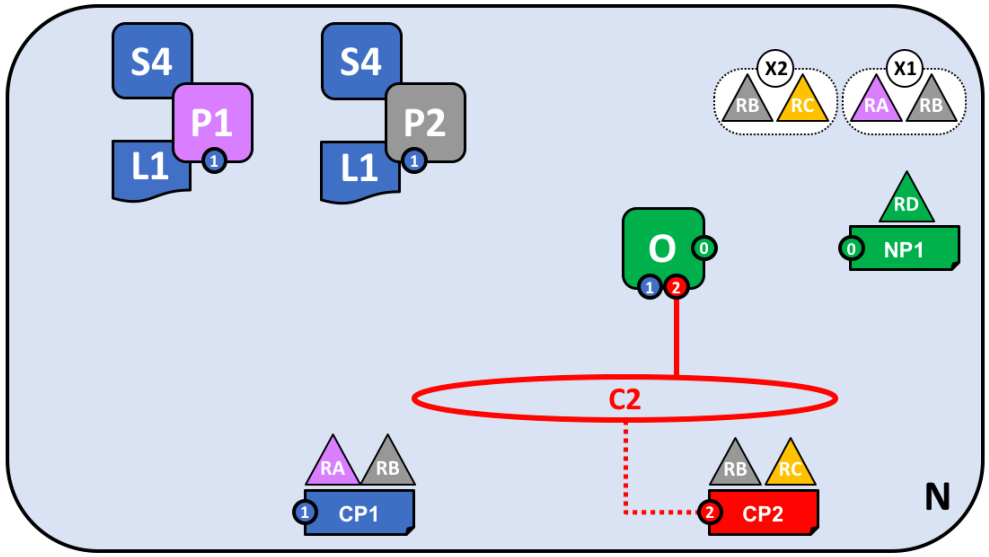
A Certificate Authority (CA) issues certificates to allow

- organizations to authenticate themselves on the network;
- client applications to authenticate transaction proposals;
- peers to approve proposals and upload transactions to the ledger if valid.

There can be one or more CAs on the network, and organizations can choose to use their own CA.

A second channel within the second consortium can be defined, with its own access policies.





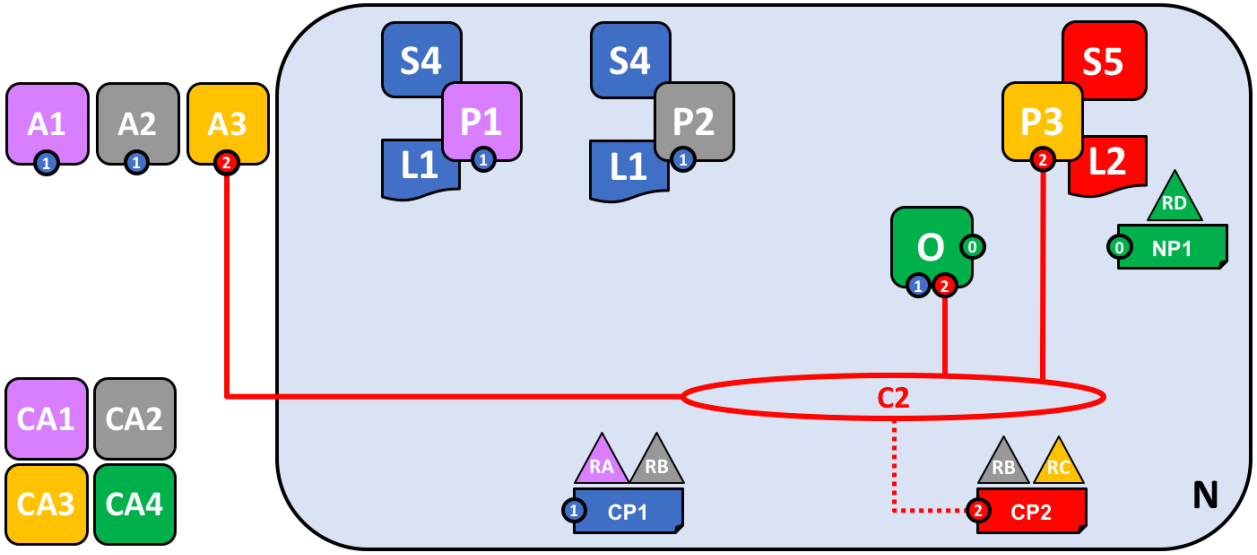
::.. Hyperledger Fabric (5/16)

A Certificate Authority (CA) issues certificates to allow

- organizations to authenticate themselves on the network;
- client applications to authenticate transaction proposals;
- peers to approve proposals and upload transactions to the ledger if valid.

There can be one or more CAs on the network, and organizations can choose to use their own CA.

A peer for the third organization can be instantiated and connected to the second channel, dispoegando the chaincode of the smart contract S5.

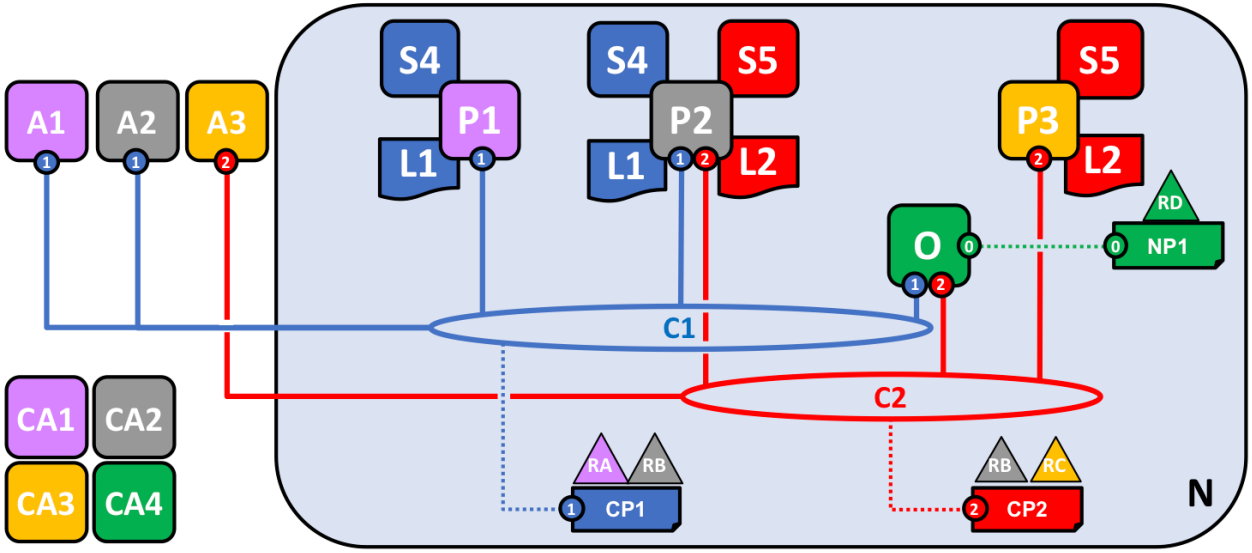


::.. Hyperledger Fabric (5/16)

A Certificate Authority (CA) issues certificates to allow

- organizations to authenticate themselves on the network;
- client applications to authenticate transaction proposals;
- peers to approve proposals and upload transactions to the ledger if valid.

There can be one or more CAs on the network, and organizations can choose to use their own CA.

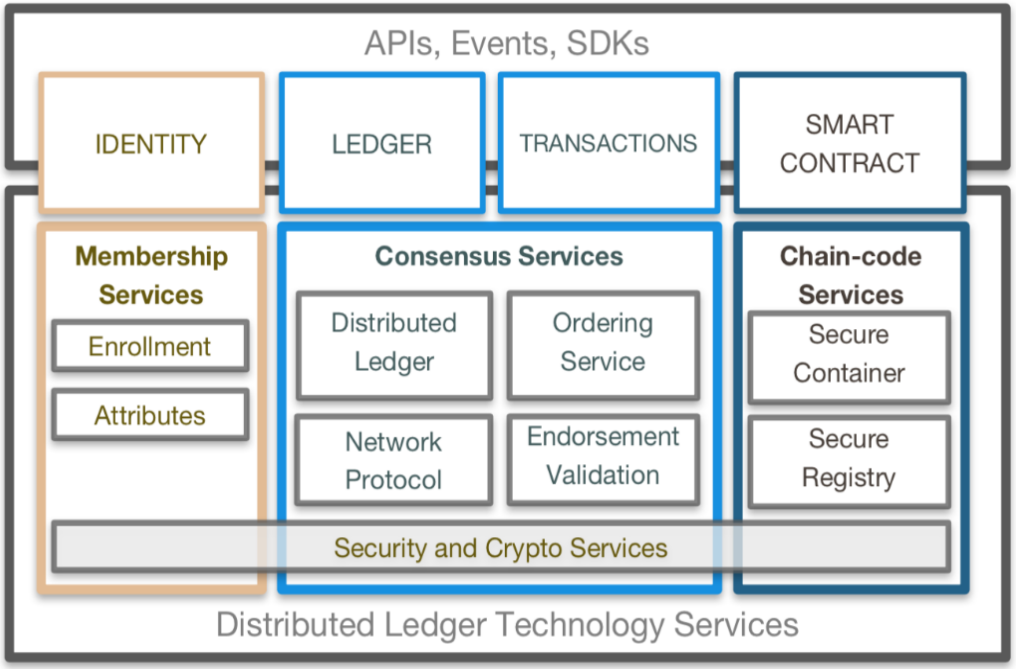


::.. Hyperledger Fabric (6/16)

Overall, the Hyperledger architecture is made up of three macro-areas:

1. One for identity management, peer membership of associations and peer registration in channels;

2. A ledger with the current data status and a transaction history;

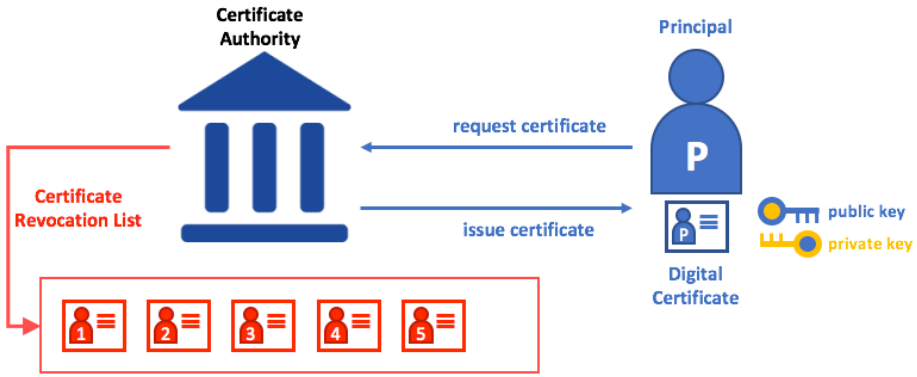


3. The chaincode execution environment for smart contracts.

Transversely, cryptographic primitives are available to support these three layers.

::.. Hyperledger Fabric (7/16)

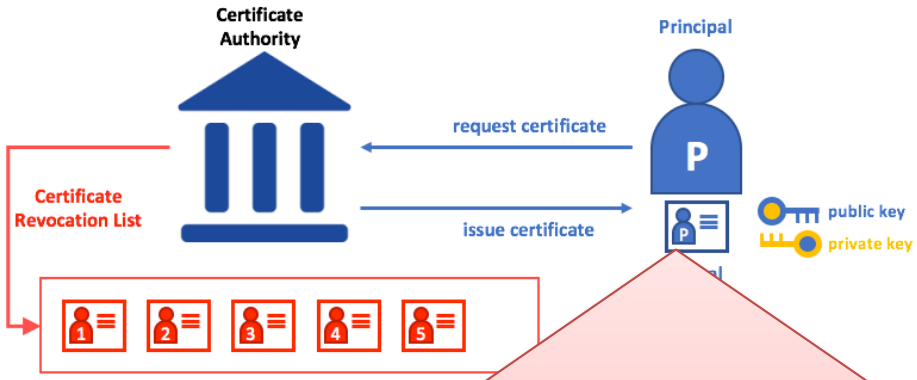
Each element has a digital identity encapsulated in an X.509 digital certificate, also expressing the necessary permissions for resources and access to functionality in the blockchain network.



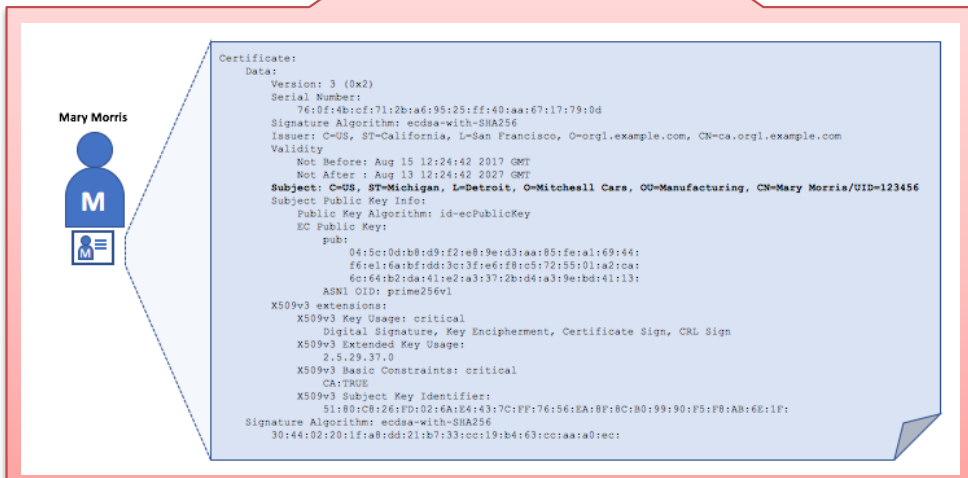
For an identity to be verifiable, it must come from a trusted authority, such as the CA with a hierarchical PKI model.

::.. Hyperledger Fabric (7/16)

Each element has a digital identity encapsulated in an X.509 digital certificate, also expressing the necessary permissions for resources and access to functionality in the blockchain network.



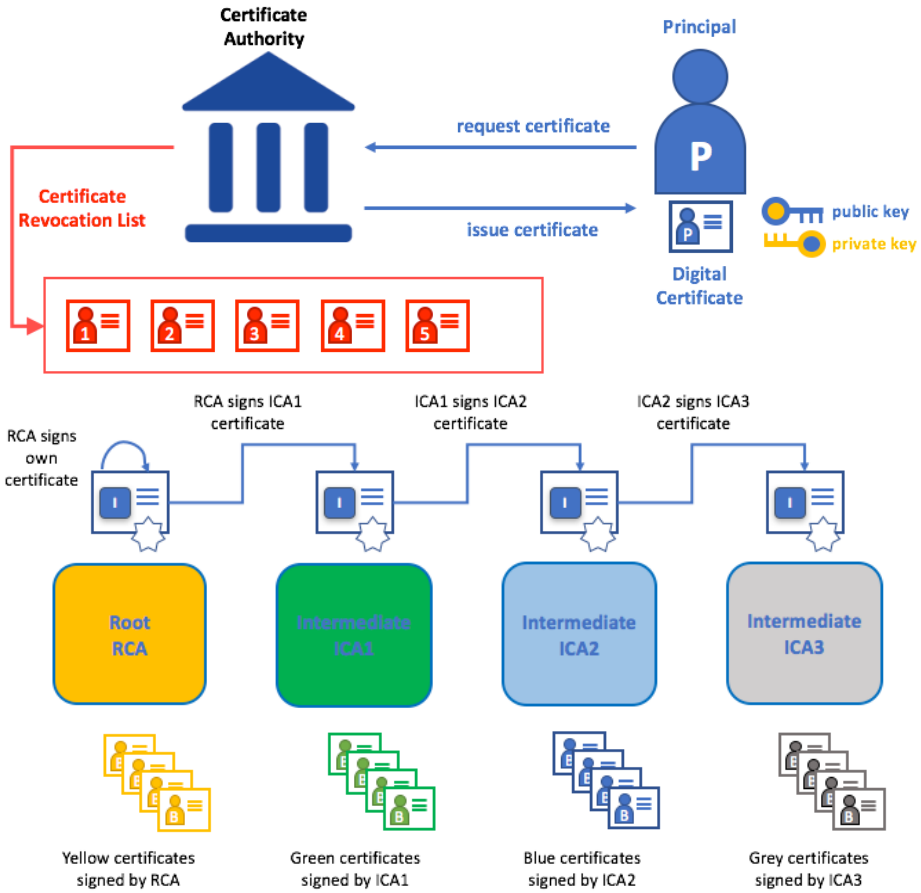
For an identity to be verifiable, it must come from a trusted authority, such as the CA with a hierarchical PKI model.



The digital certificate contains a lot of information about an identity in SUBJECT, but also the public key linked to the identity, while its signature key is not and is kept private. The certificate is countersigned by the CA that issues it and makes it impossible to modify.

::.. Hyperledger Fabric (7/16)

Each element has a digital identity encapsulated in an X.509 digital certificate, also expressing the necessary permissions for resources and access to functionality in the blockchain network.



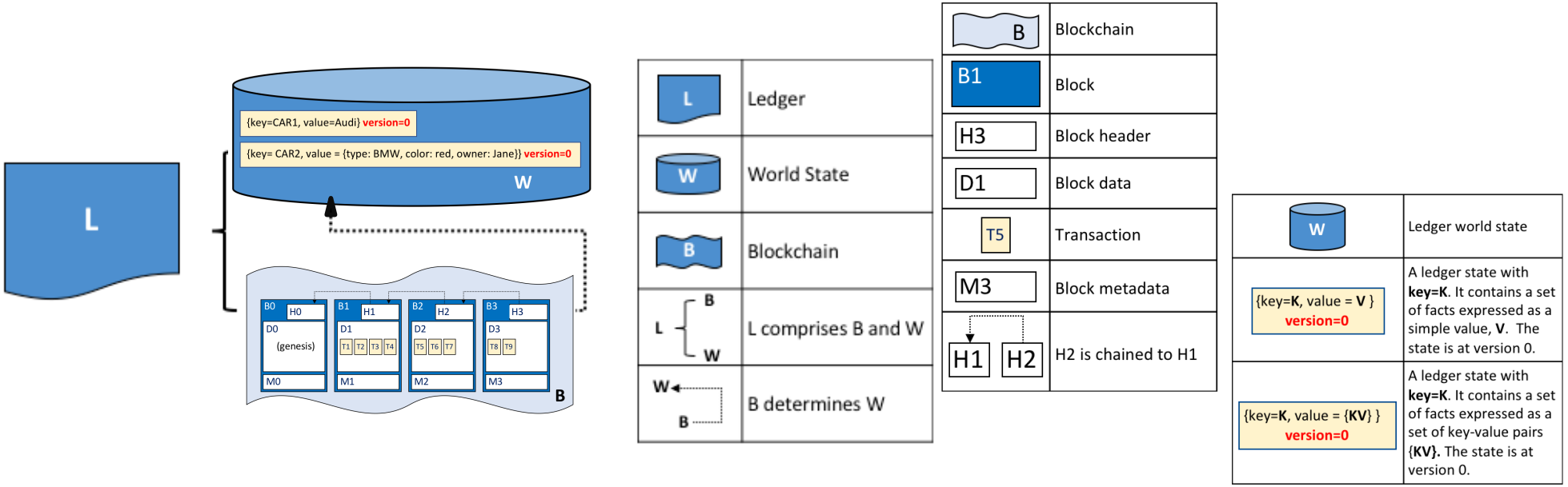
For an identity to be verifiable, it must come from a trusted authority, such as the CA with a hierarchical PKI model.

There is a hierarchy of CAs, with a Root CA as the root and a set of intermediary CAs whose certificates are signed by the top-level CAs. Fabric offers its own root CA.

::.. Hyperledger Fabric (8/16)

The blockchain ledger consists of two distinct parts:

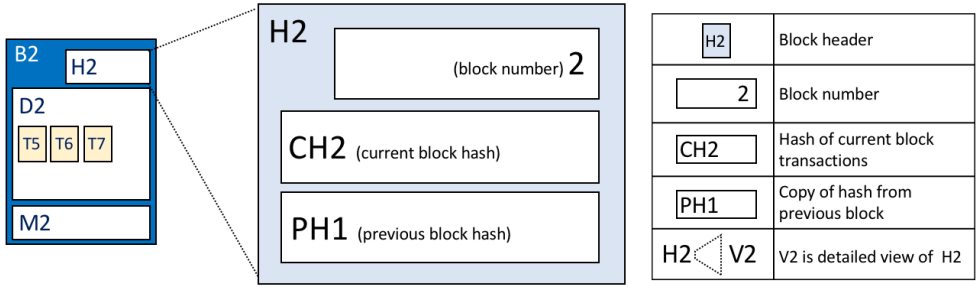
- a global state in a database that contains the current values of a data set that can be addressed by means of a key.
- A history of transactions as a blockchain, with all the changes in the global state.

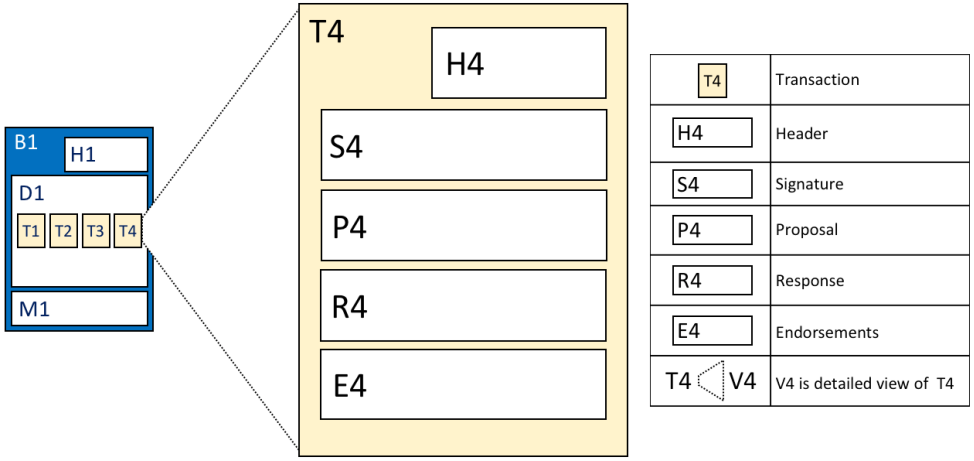


The version number is incremented each time the status changes, and also checked each time the status is updated.

::.. Hyperledger Fabric (9/16)

A block has a header with a sequence number, the hash of the previous block and block, the body with the ordered set of transactions and meta-data with the block creation time, the certificate, the public key and the signature of the block creator.



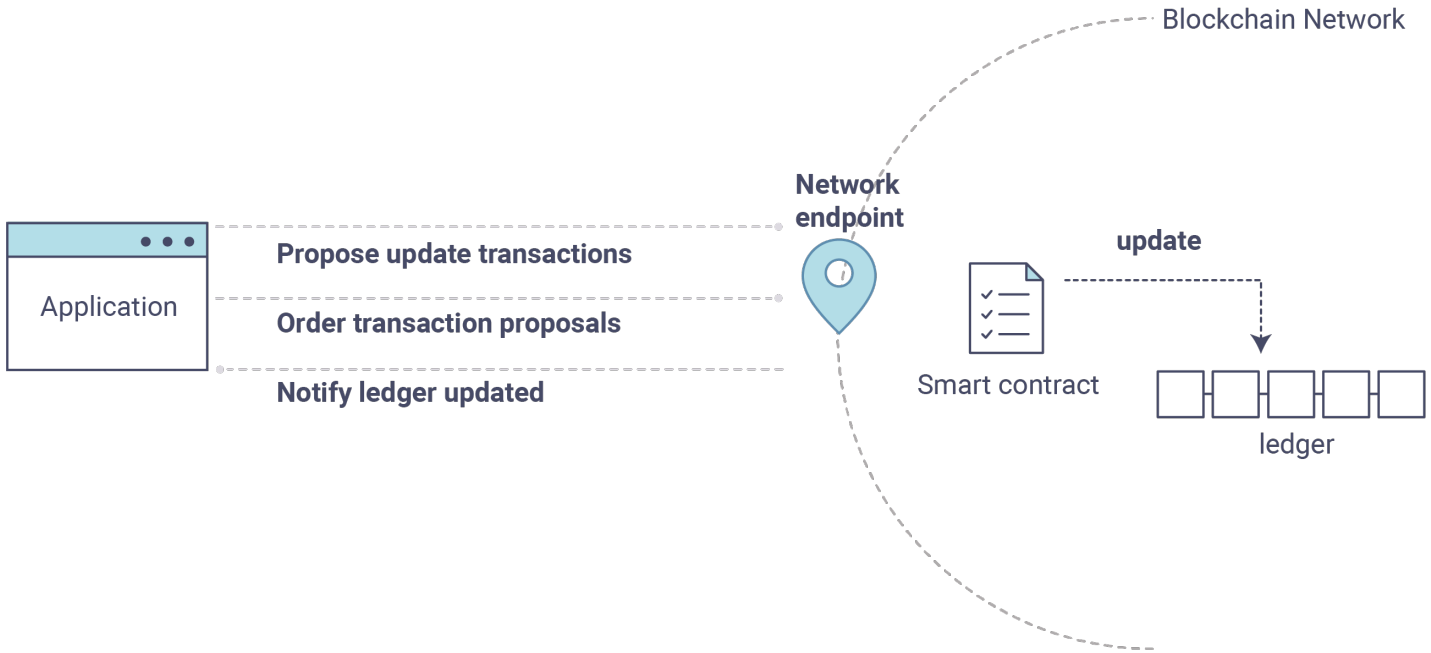


Transactions have a header such as the name of the relevant chaincode and its version, the creator's signature, the proposal and the response with the endorsement.

::.. Hyperledger Fabric (10/16)

Chaincode is a program written in Go, node.js or Java that implements a special interface and runs in a Docker container isolated from the peer Endorser process.

- The state created by a chaincode is limited to that chaincode only and cannot be accessed directly from another chaincode. However, a chaincode can invoke another chaincode to access its state.



::.. Hyperledger Fabric (11/16)

The Chaincode interface specifies the functions to be implemented:

- Init () is called when a chaincode receives an update instance or transaction to perform any necessary initialization, including initialization of the application state;
- Invoke () is called in response to the receipt of an invoke transaction to process the transaction proposals.

The other interface is ChaincodeStubInterface, which is used to access and modify the blockchain and to make invocations between chaincodes.

Since Java support was recently introduced, the Node.js and Go lang APIs are at a more mature level. JavaScript does not lend itself well to numerical calculations.

::.. Hyperledger Fabric (12/16)

The most used language for chaincode implementation is Go. This language does not have the concept of a class, but Go offers structs, a lightweight version of classes, to which methods can be added.

We create a smart contract in Go for the management of cars and their owners. We implement structs for this data.

```
type Car struct{
 modelName string
 color string
 serialNo string
 manufacturer string
 owner Owner  //composition
}


type Owner struct{
 name string
 nationalIdentity string
 gender string
 address string
}
```

The method is defined externally, indicating in the former case whether changes in the method body are reflected on the caller.

```
//attached by reference ==> called as pointer receiver
func (c *Car) changeOwner(newOwner Owner) {
 c.owner = newOwner
}


/** attached by value ==> called as value receiver
func (c Car) changeOwner(newOwner Owner) {
 c.owner = newOwner
}
*/
```

::.. Hyperledger Fabric (13/16)

The interface to be implemented is the following with the two main functions of the 36 currently specified:

```
type Chaincode interface {
// Init method accepts stub of type ChaincodeStubInterface as
// argument and returns peer.Response type object
 Init(stub ChaincodeStubInterface) peer.Response


 Invoke(stub ChaincodeStubInterface) peer.Response
}
```

There is no derivation syntax, but you just need to implement the methods of interest.

```
type CarChaincode struct{
}

//Init implemented by CarChaincode
func (t *CarChaincode) Init(stub shim.ChaincodeStubInterface)
pb.Response {


}


//Invoke implemented by CarChaincode
func (t *CarChaincode) Invoke(stub shim.ChaincodeStubInterface)
pb.Response {


}
```

::.. Hyperledger Fabric (14/16)

We insert the initialization logic:

```
func (t *CarChaincode) Init(stub shim.ChaincodeStubInterface)
pb.Response {


//Declare owners from Owner struct
 tom := Owner{name: "Tom H", nationaIdentity: "ABCD33457", gender:
"M", address: "1, Tumbbad"}
 bob := Owner{name: "Bob M", nationaIdentity: "QWER33457", gender:
"M", address: "2, Tumbbad"}


//Declaree carfrom Car struct
 car := Car{modelName: "Land Rover", color: "white", serialNo:
"334712531234", manufacturer: "Tata Motors", owner: tom}


// convert tom Owner to []byte
 tomAsJSONBytes, _ := json.Marshal(tom)
 //Add Tom to ledger
 err := stub.PutState(tom.nationaIdentity, tomAsJSONBytes)
 if err != nil {
  return shim.Error("Failed to create asset " + tom.name)
 }


//Add Bob to ledger
 bobAsJSONBytes, _ := json.Marshal(bob)
 err = stub.PutState(bob.nationaIdentity, bobAsJSONBytes)
 if err != nil {
  return shim.Error("Failed to create asset " + bob.name)
 }
```

```
//Add car to ledger
 carAsJSONBytes, _ := json.Marshal(car)
 err = stub.PutState(car.serialNo, carAsJSONBytes)
 if err != nil {
  return shim.Error("Failed to create asset " + car.serialNo)
 }


 return shim.Success([]byte("Assets created successfully."))
}
```

::.. Hyperledger Fabric (15/16)

Let's insert the invoke logic:

```
func (c *CarChaincode) Invoke(stub shim.ChaincodeStubInterface)
pb.Response {


 // Read args from the transaction proposal.
 // fc=> method to invoke
 fc, args := stub.GetFunctionAndParameters()
 if fc == "TransferOwnership" {
  return c.TransferOwnership(stub, args)
 }
 return shim.Error("Called function is'nt defined in the chaincode
")
}


func (c *CarChaincode) TransferOwnership(stub
shim.ChaincodeStubInterface, args []string) pb.Response {
 // args[0]=> car serial no
 // args[1]==> new owner national identity
 // Read existing car asset
 carAsBytes, _ := stub.GetState(args[0])
 if carAsBytes == nil {
  return shim.Error("car asset not found")
 }


// Construct the struct Car
 car := Car{}
 _ = json.Unmarshal(carAsBytes, &car)
```

```
//Read newOwnerDetails
 ownerAsBytes, _ := stub.GetState(args[1])
 if ownerAsBytes == nil {
  return shim.Error("owner asset not found")
 }


// Construct the struct Owner
 newOwner := Owner{}
 _ = json.Unmarshal(ownerAsBytes, &newOwner)


// Update owner
 car.changeOwner(newOwner)


 carAsJSONBytes, _ := json.Marshal(car)


// Update car ownership in the
err := stub.PutState(car.serialNo, carAsJSONBytes)
 if err != nil {
  return shim.Error("Failed to create asset " + car.serialNo)
 }
return shim.Success([]byte("Asset modified."))
}
```

::.. Hyperledger Fabric (16/16)

To complete the smart contract it is necessary to specify the preamble:

```
package main


import (
 "encoding/json"


"github.com/hyperledger/fabric/core/chaincode/shim"
 pb "github.com/hyperledger/fabric/protos/peer"
)
```

And the main function:

```
func main() {


logger.SetLevel(shim.LogInfo)


// Start chaincode process
err := shim.Start(new(CarChaincode))
if err != nil {
logger.Error("Error starting PhantomChaincode - ", err.Error()
}
}
```

Blockchain
Applications in IoT

::.. Blockchain & IoT (1/2)

Blockchain has been foreseen by industry and research community as a disruptive technology that is poised to play a major role in managing, controlling, and most importantly securing IoT devices.

::.. Blockchain & IoT (1/2)

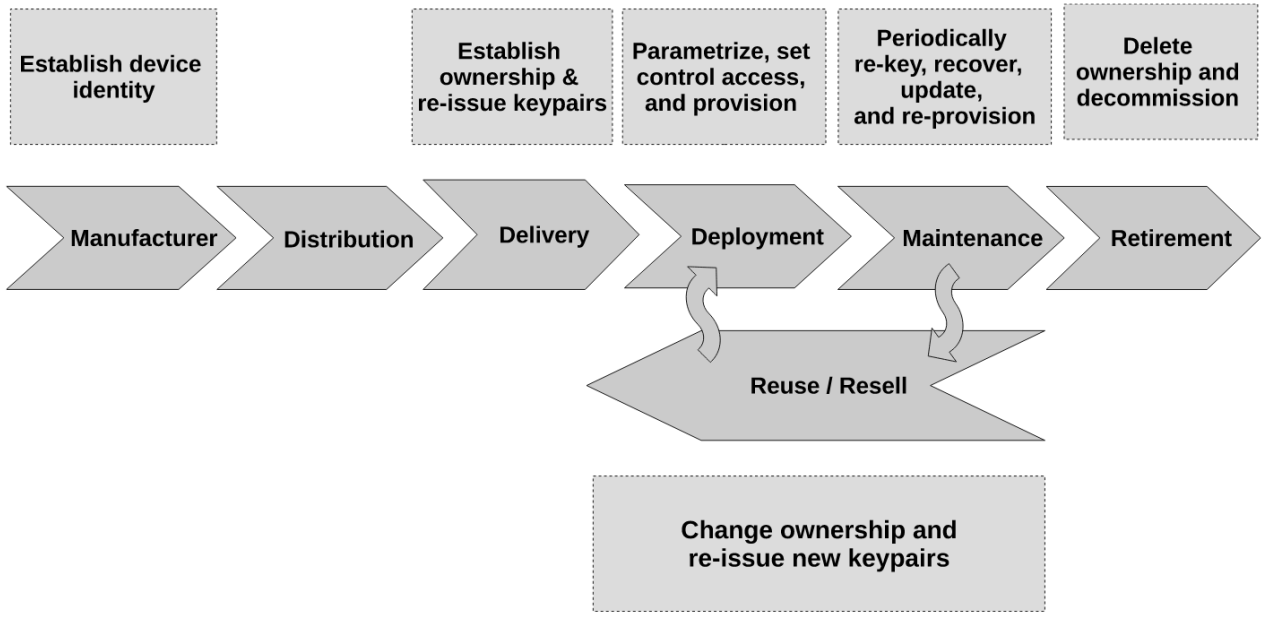
Blockchain has been foreseen by industry and research community as a disruptive technology that is poised to play a major role in managing, controlling, and most importantly securing IoT devices.

- **Data Authentication and Integrity** - In many areas where an exhaustive traceability of assets during their life cycle is required by regulations, data immutability becomes a key challenge and has motivated the application of Blockchain within the context of IoT.

::.. Blockchain & IoT (1/2)

Blockchain has been foreseen by industry and research community as a disruptive technology that is poised to play a major role in managing, controlling, and most importantly securing IoT devices.

- **Data Authentication and Integrity**.
- **Identity of Things (IDoT) and Governance** - Blockchain has been used widely for providing trustworthy and authorized identity registration, ownership tracking and monitoring of assets.



::.. Blockchain & IoT (1/2)

Blockchain has been foreseen by industry and research community as a disruptive technology that is poised to play a major role in managing, controlling, and most importantly securing IoT devices.

- **Data Authentication and Integrity**.
- **Identity of Things (IDoT) and Governance**.
- **Authentication, Authorization, and Privacy** – Blockchain smart contracts have the ability to provide a decentralized authentication rules and logic to provide single and multi-party authentication to an IoT Device. Also, smart contracts can provide a more effective authorization access rules to connected IoT devices with way less complexity when compared with traditional authorization protocols.

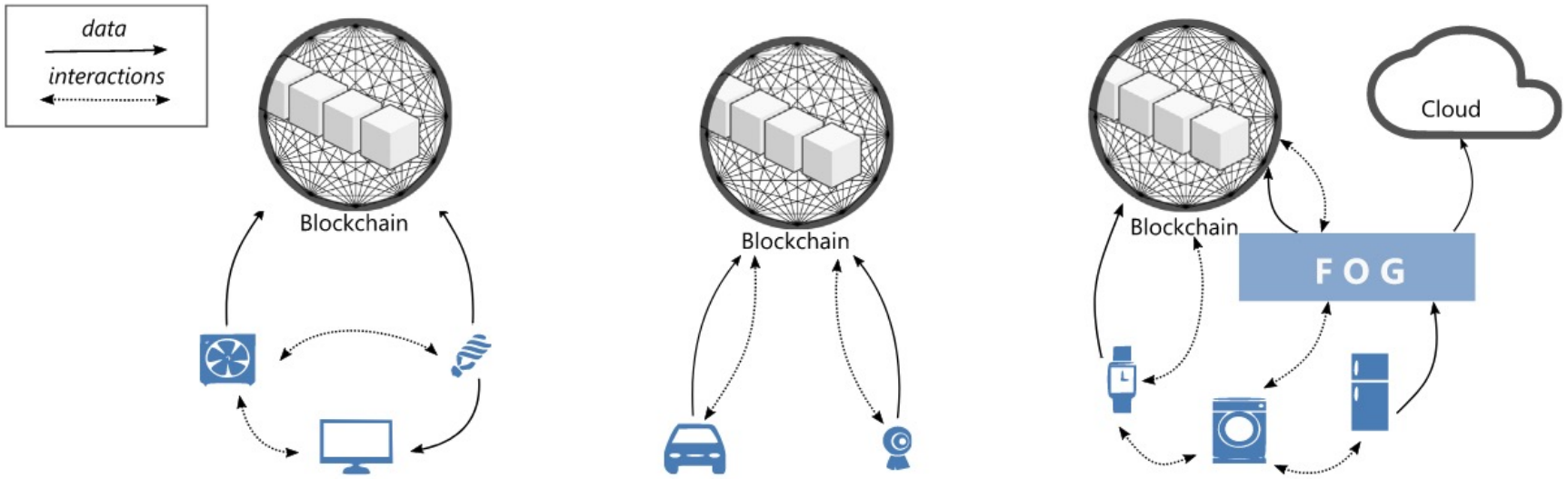
::.. Blockchain & IoT (1/2)

Blockchain has been foreseen by industry and research community as a disruptive technology that is poised to play a major role in managing, controlling, and most importantly securing IoT devices.

- **Data Authentication and Integrity**.
- **Identity of Things (IDoT) and Governance**.
- **Authentication, Authorization, and Privacy**.
- **Secure Communication** - With blockchain, key management, and distribution are totally eliminated, each IoT device would have its own unique GUID and asymmetric key pair once installed and connected to the blockchain network. This will lead also to a significant simplification of security protocols.

::.. Blockchain & IoT (2/2)

Another aspect to take into account is related to the IoT interactions when integrating blockchain, as it is not possible to have the blockchain directly running at the IoT node due to the excessive costs in terms of memory and energy consumption.

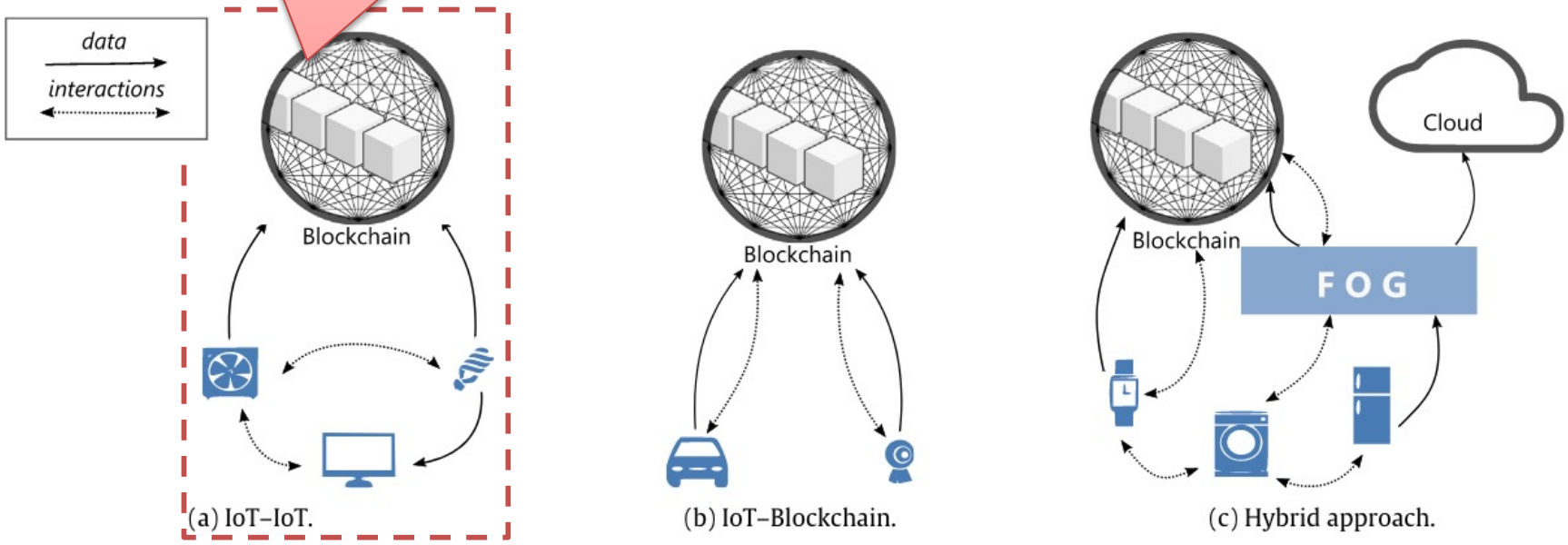


(a) IoT–IoT. (b) IoT–Blockchain. (c) Hybrid approach.

::.. Blockchain & IoT (2/2)

Another aspect to take into account is related to the IoT interactions when integrating blockchain, as it is not possible to have

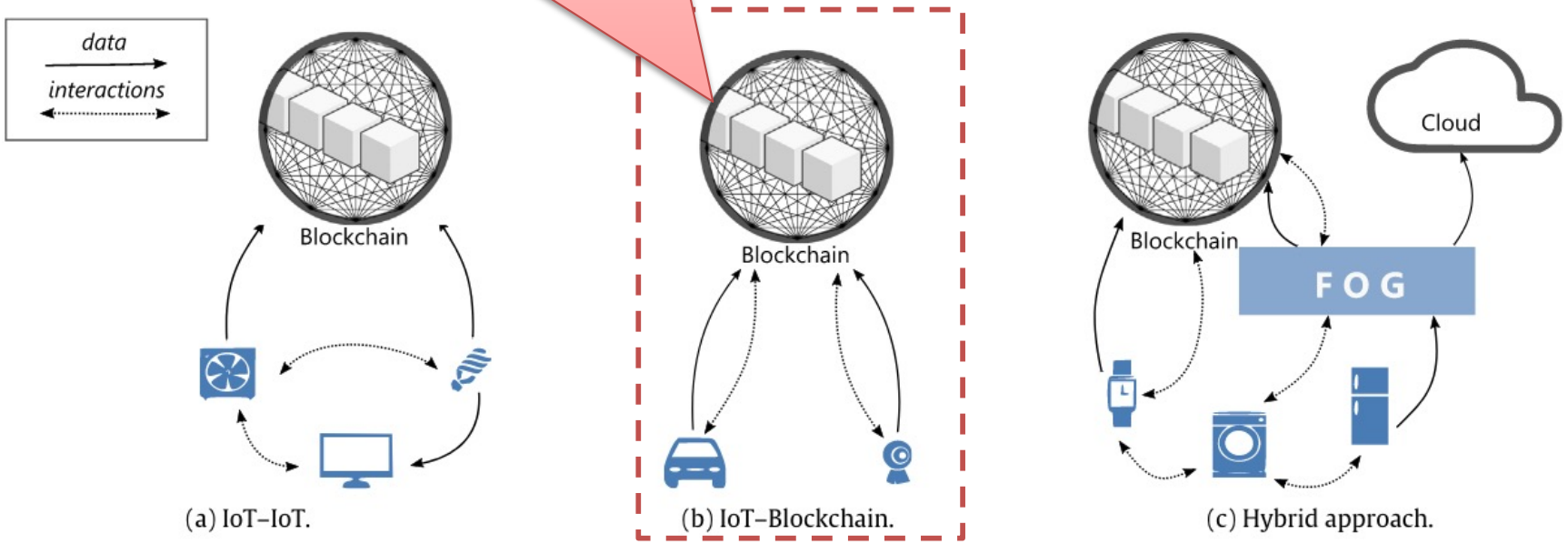
IoT devices are able to communicate with each other, which usually involves discovery and routing mechanisms. Only a part of IoT data is stored in blockchain whereas the IoT interactions take place without using the blockchain.



::.. Blockchain & IoT (2/2)

Another aspect to take into account is related to the IoT interactions when integrating blockchain, as it is not possible to have

All the interactions go through the blockchain, enabling an immutable record of interactions and the possibility of having all the chosen interactions traceable. Recording all the interactions in blockchain would involve an increase in bandwidth and data retrieval time, which is one of the well-known challenges in blockchain.

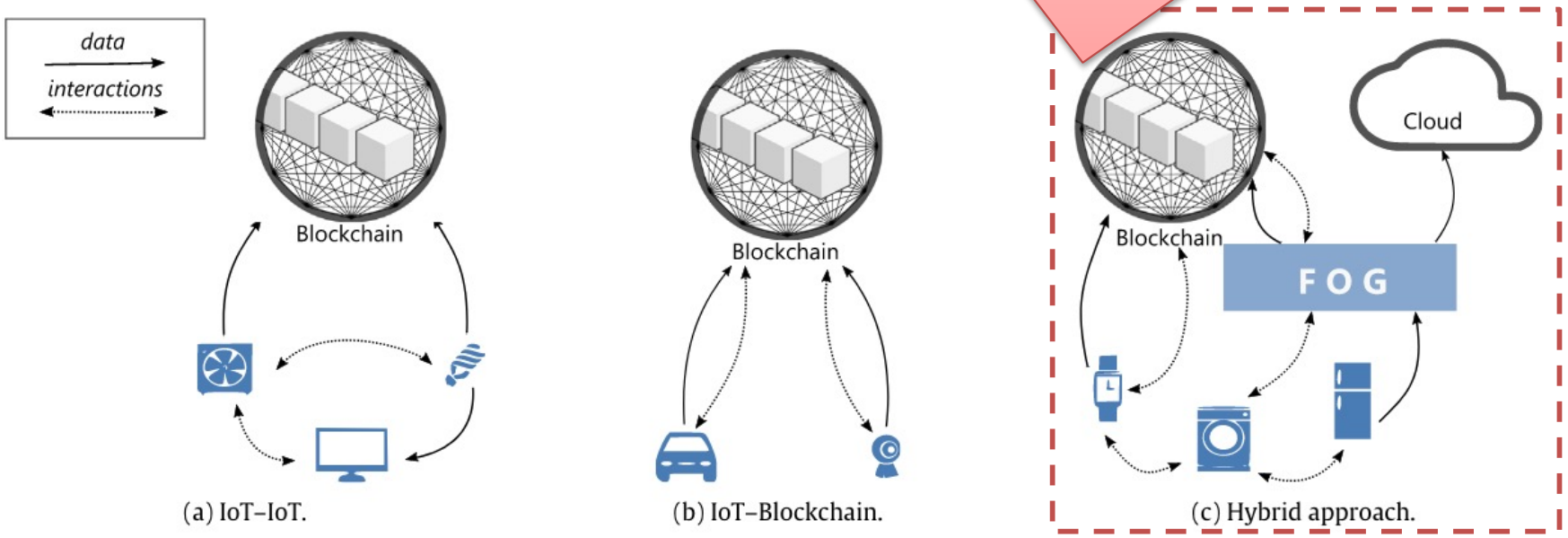


(a) IoT–IoT. (b) IoT–Blockchain. (c) Hybrid approach.

::.. Blockchain & IoT (2/2)

Another aspect to take into account is related to the IoT in-

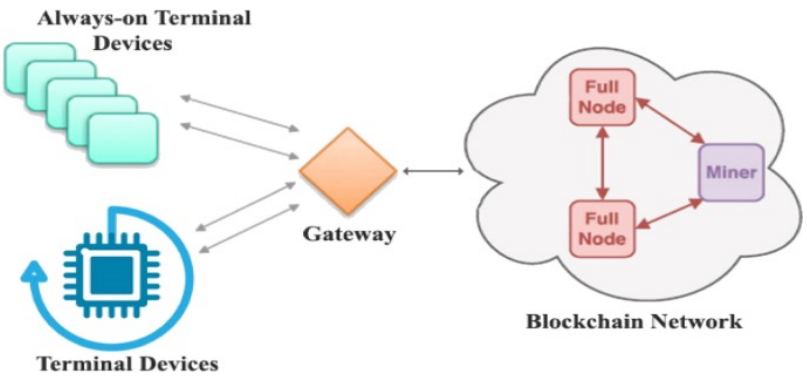
A hybrid design where only part of the interactions and data take place in the blockchain and the rest are directly shared between the IoT devices. One of the challenges in this approach is choosing which interactions should go through the blockchain and providing the way to decide this in runtime. In this approach fog computing could come into play and even cloud computing, to complement the limitations of blockchain and the IoT.



(a) IoT–IoT. (b) IoT–Blockchain. (c) Hybrid approach.

::.. Blockchain & IoT (2/2)

Another aspect to take into account is related to the IoT interactions when integrating blockchain, as it is not possible to have the blockchain directly running at the IoT node due to the excessive costs in terms of memory and energy consumption.

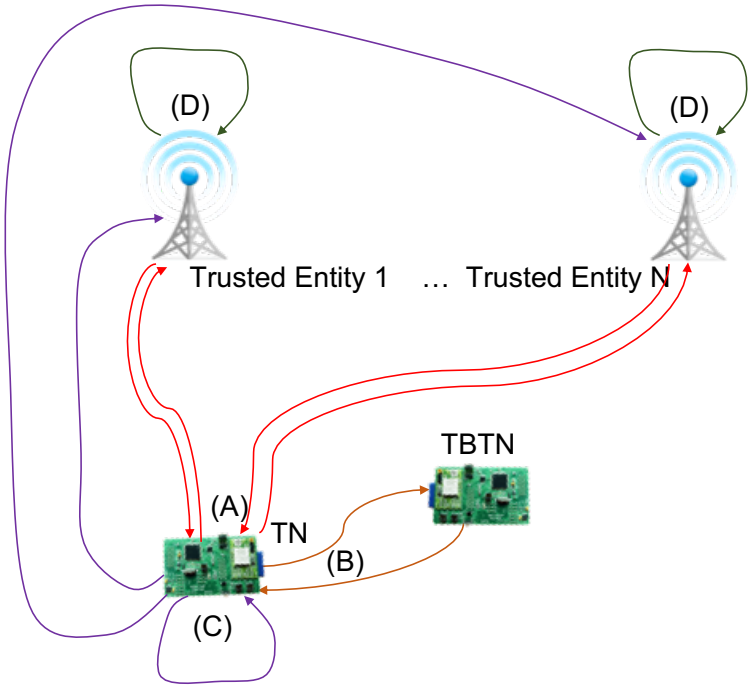


In a typical IoT deployment, limited-resource devices are used as end-nodes that communicate with a gateway that is responsible for forwarding sensor data to upper layers (cloud or server).

Cryptographic functionality could be provided in IoT devices so to realize IoT autonomy, which comes at a cost of a more sophisticated hardware and higher computational expense.

::.. Example - Blockchain & IoT (1/4)

A suitable solution for efficient trust management in the IoT would be to have a federation of replicated Trust Providers, so as to make it suitable for multi-tenant IoT and removing the performance bottleneck and singe point-of-failure represented by a centralised provider.

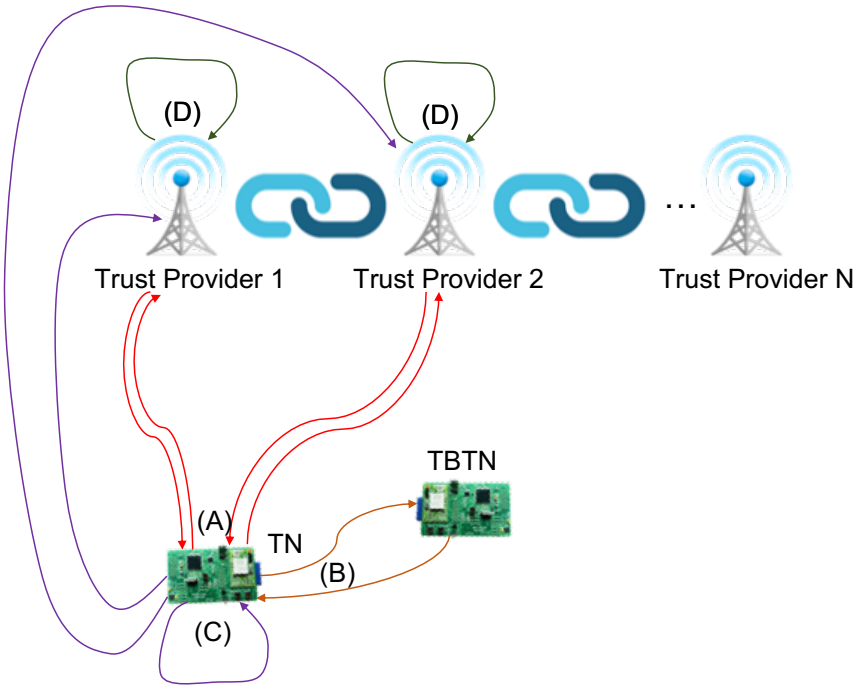


Issues:

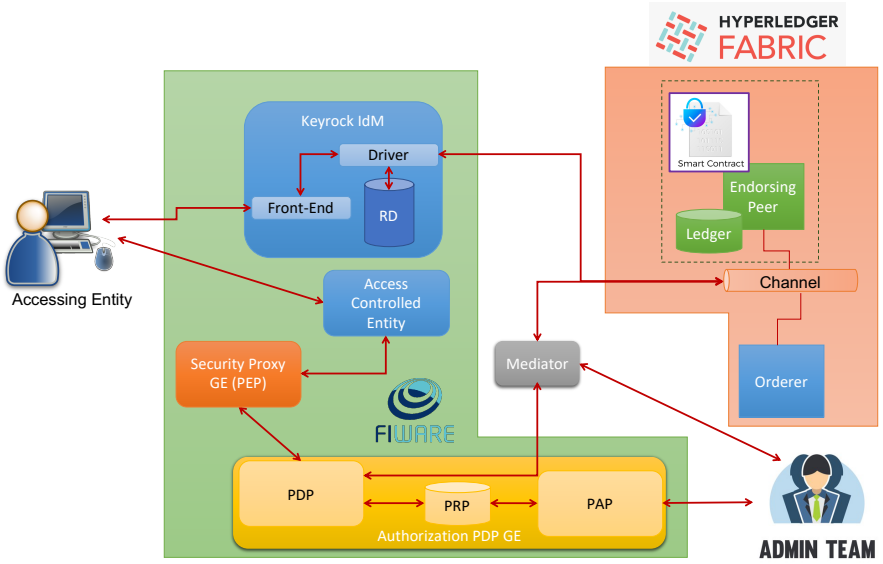
- Protect the communications between the trust providers and the requesting node;
- Avoid revealing information hosted by the trust providers;
- Support consistency among the multiple trust providers.

::.. Example - Blockchain & IoT (2/4)

A proposed solution is to leverage on the blockchain as a shared repository of the trust values. The trust providers are running at the edge/fog level of the IoT infrastructure, but not at the IoT nodes, due to their constraints for the available resources.



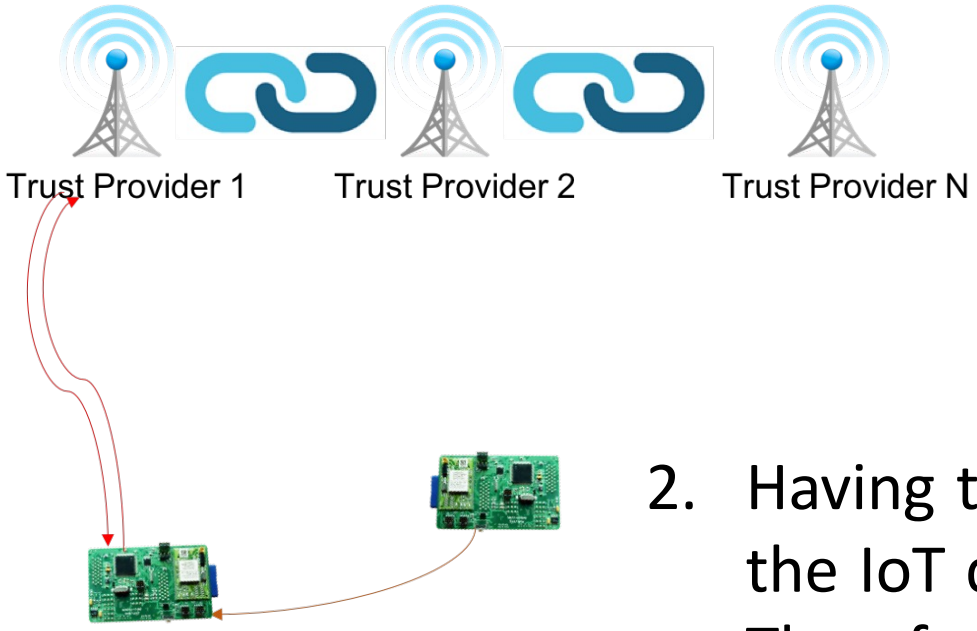
Such a solution can be easily applied also at the context of authentication/authorization, where the provider implements XACML:



::.. Example - Blockchain & IoT (3/4)

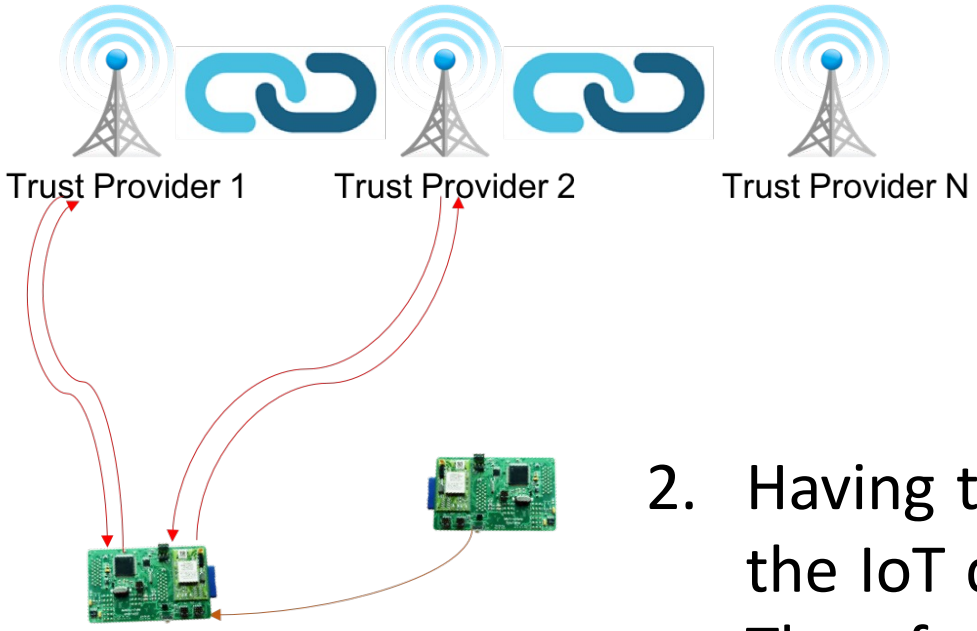
1. The fog/edge nodes and/or the ones within the cloud can write new blocks and add them to the chains. Each chain contains the trust values associated to each entity within the system.

::.. Example - Blockchain & IoT (3/4)



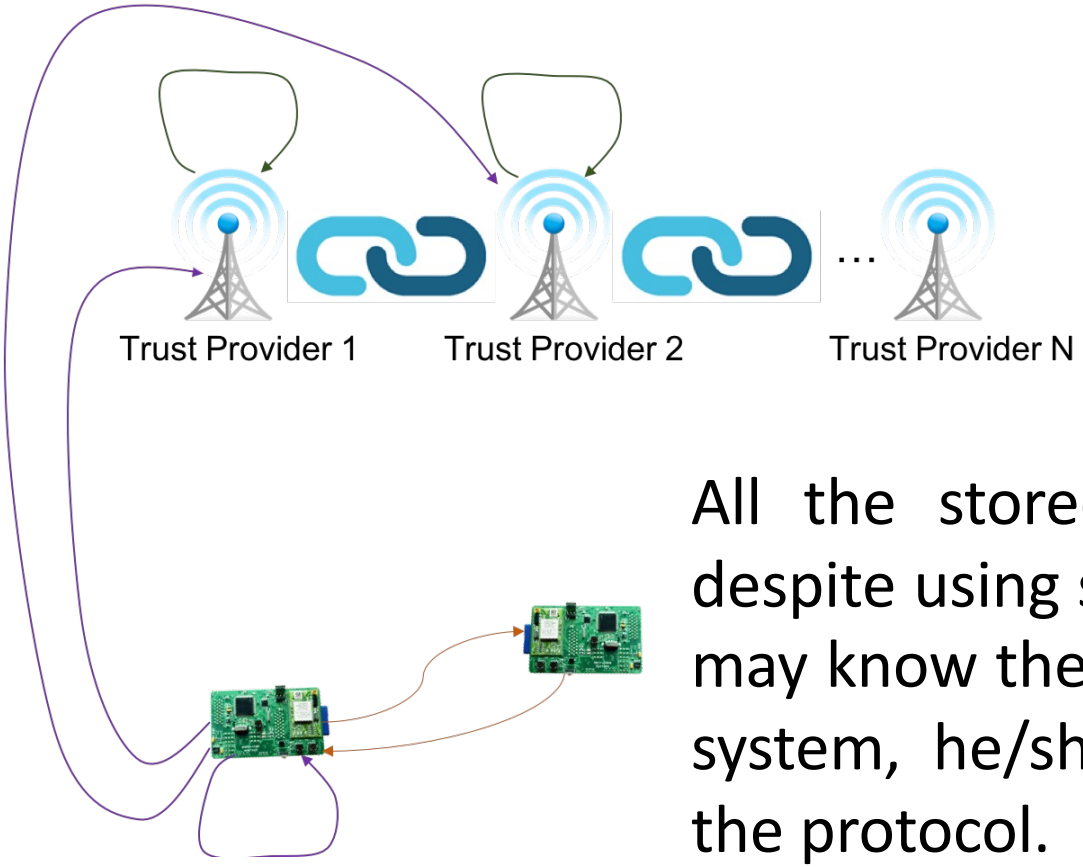
1. The fog/edge nodes and/or the ones within the cloud can write new blocks and add them to the chains. Each chain contains the trust values associated to each entity within the system.
2. Having these nodes to periodically inform the IoT devices of any changes is a waste. Therefore, when a trust value is needed, IoT devices contact any reachable nodes.

::.. Example - Blockchain & IoT (3/4)



1. The fog/edge nodes and/or the ones within the cloud can write new blocks and add them to the chains. Each chain contains the trust values associated to each entity within the system.
2. Having these nodes to periodically inform the IoT devices of any changes is a waste. Therefore, when a trust value is needed, IoT devices contact any reachable nodes.
3. More than one node are contacted so as to avoid incurring in a malicious node impersonating a legitimate one or receiving diverging information.

::.. Example - Blockchain & IoT (4/4)



4. If the direct interaction with an entity bring the sensor to have different trust value than the one collected from the contacted nodes, an update is requested, by creating a novel block.

All the stored data are not encrypted, so despite using secure channels, if the adversary may know the identifier of a valid entity in the system, he/she may be able to compromise the protocol.

To this aim, a pseudonym mechanism is needed to anonymize the trust information and make more complex to trace them back.