Blockchain → db distribuito (immutabile).
ci sono più parti che possono modificare i dati → aggiungendo informazioni.

→ lo stato corrente del db è ricostruito applicando tutte le transazioni allo stato iniziale.

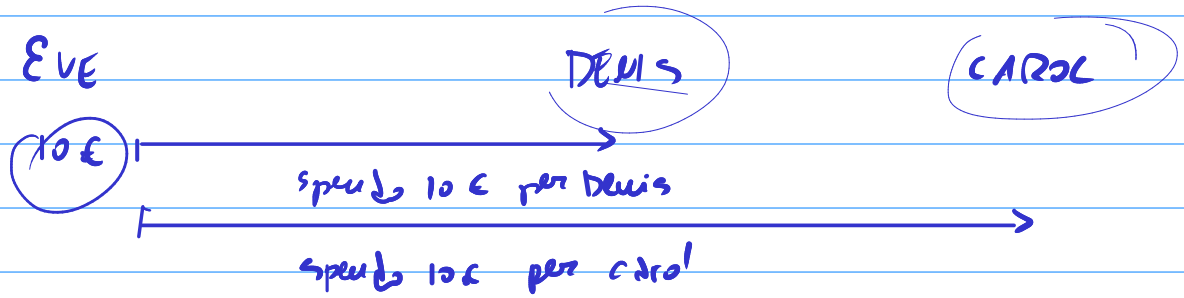
→ distribuito → più attori possono aggiungere blocchi.

È impossibile (senza ulteriori ipotesi) garantire contemporaneamente

- accessibilità dei dati
- consistenza dei dati
- resistenza alle partizioni.

→ teorema CAP →

In particolare in una blockchain ci può essere il problema di attori che agiscono in modo attivo contro gli obiettivi della stessa.




DOUBLE SPEND.

entrambi verificano che EVE abbia 10 € e la verifica è positiva.

SOLUZIONE DI BITCOIN (prob. generali bizantini) NON IMPORTA QUALE SIA LA TRANSAZIONE "VERA" → basta solo che ne sia

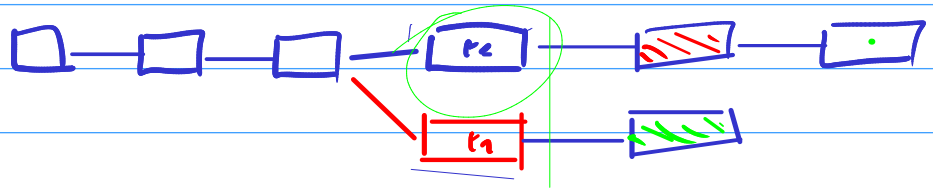
accettata una sola.

Lasciamo decidere alla "rete" quale transazione sia quella valida.

↓

eventuale le transazioni sono raggruppate con altre in blocchi ma nessun blocco le contiene tutte e due.

I blocchi sono inseriti in delle catene




dopo un tempo "abbastanza grande" una delle catene (quella che contiene $t_2$ o quella che contiene $t_1$) sarà più lunga dell'altra. (è risulta "impossibile" per la seconda catena raggiungerla).

A quel punto si prende come valida la sola catena più lunga e si scarta l'altra.

# Blockchains

1. Immutable distributed database
2. Byzantine agreement protocol

**Remark**

A blockchain might offer other facilities:

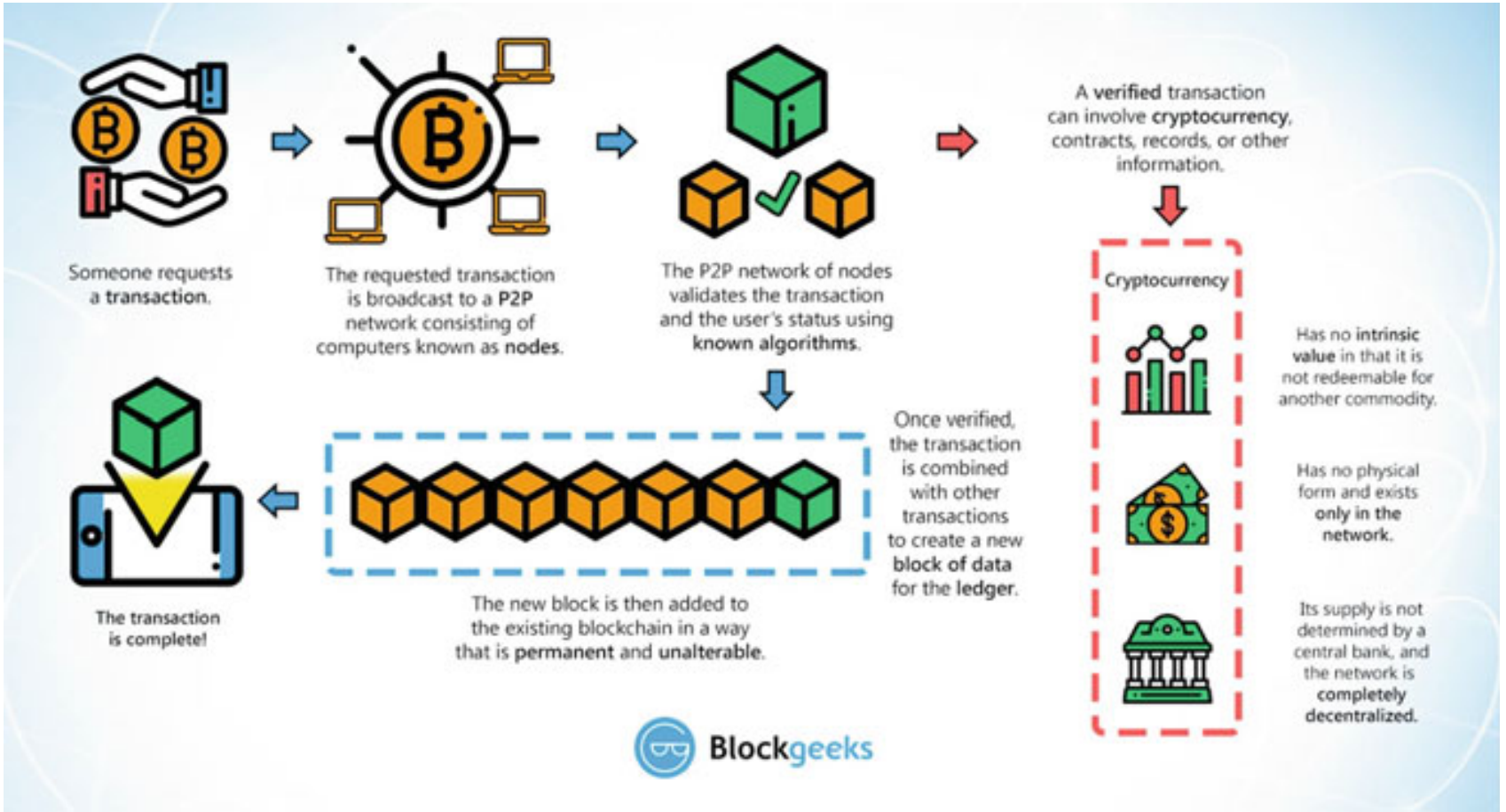
- Virtual Machines
- Smart contracts
- Storage optimizations (Merkle trees)
- etc.

# Immutable distributed database

A blockchain is

- Distributed: data can be read and written by a set of non-coordinated agents;
- Immutable: once accepted the data cannot be altered in any way.

# Blockchain


Someone requests a transaction.
The requested transaction is broadcast to a P2P network consisting of computers known as nodes.
The P2P network of nodes validates the transaction and the user's status using known algorithms.
A verified transaction can involve cryptocurrency, contracts, records, or other information.
Cryptocurrency
Has no intrinsic value in that it is not redeemable for another commodity.
Has no physical form and exists only in the network.
Its supply is not determined by a central bank, and the network is completely decentralized.
Once verified, the transaction is combined with other transactions to create a new block of data for the ledger.
The new block is then added to the existing blockchain in a way that is permanent and unalterable.
The transaction is complete!
Blockgeeks

# Consensus

- Different agents can have a different view of the database (fork).
- We need an algorithm for deciding which is the version of the database to be trusted in the case of conflicts (byzantine agreement).

N.B. non importa quale versione

# Limitations

## Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services

Seth Gilbert* Nancy Lynch*

**Abstract**

When designing distributed web services, there are three properties that are commonly desired: consistency, availability, and partition tolerance. It is impossible to achieve all three. In this note, we prove this conjecture in the asynchronous network model, and then discuss solutions to this dilemma in the partially synchronous model.

### 1 Introduction

At PODC 2000, Brewer[1], in an invited talk [2], made the following conjecture: it is impossible for a web service to provide the following three guarantees:

- Consistency
- Availability
- Partition-tolerance

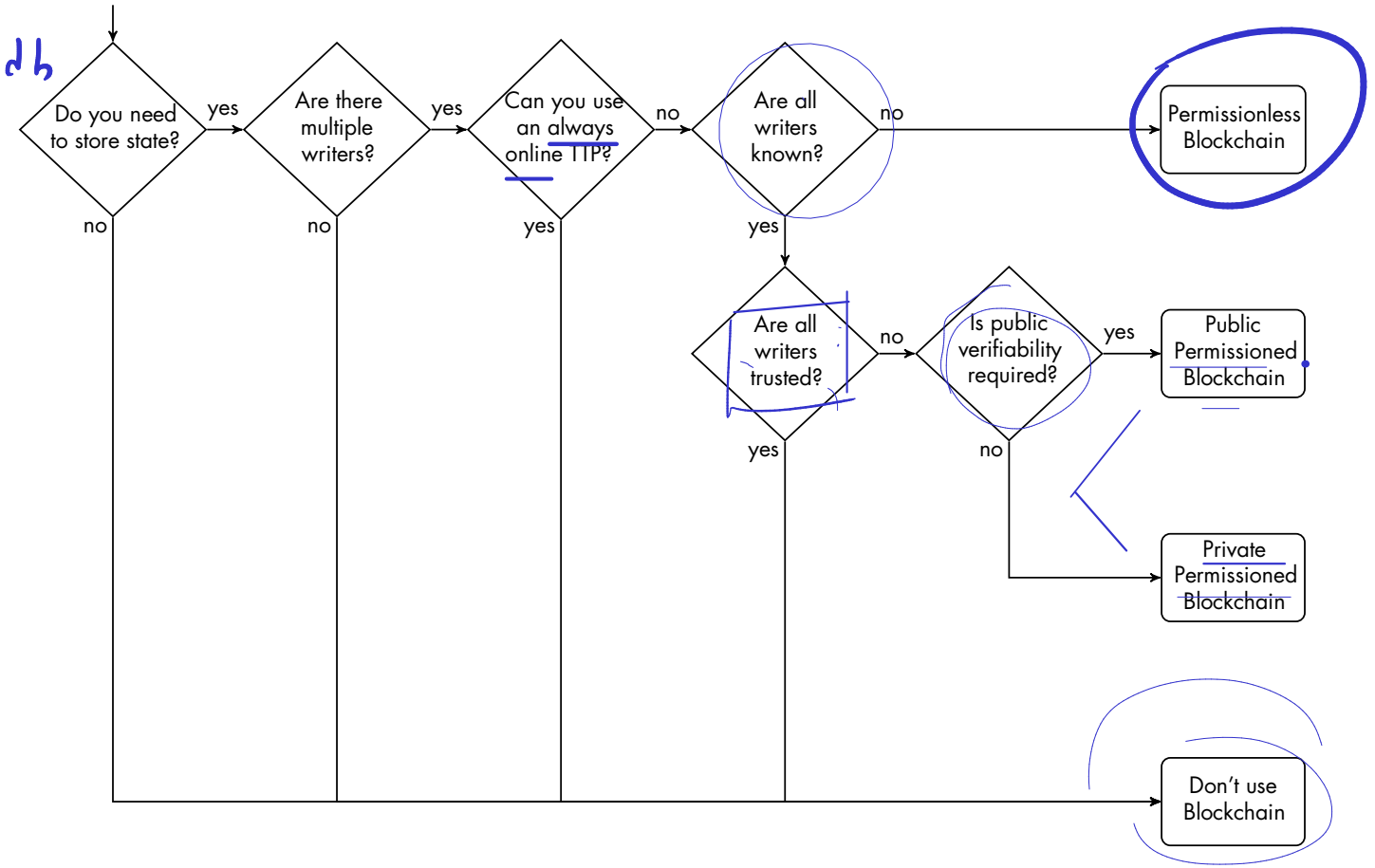
All three of these properties are desirable – and expected – from real-world web services. In this note, we will first discuss what Brewer meant by the conjecture; next we will formalize these concepts and prove the conjecture;

# Taxonomy of blockchains

Various kinds:

- Public/Private
- Permissioned/Permissionless
- Decentralized/Centralized

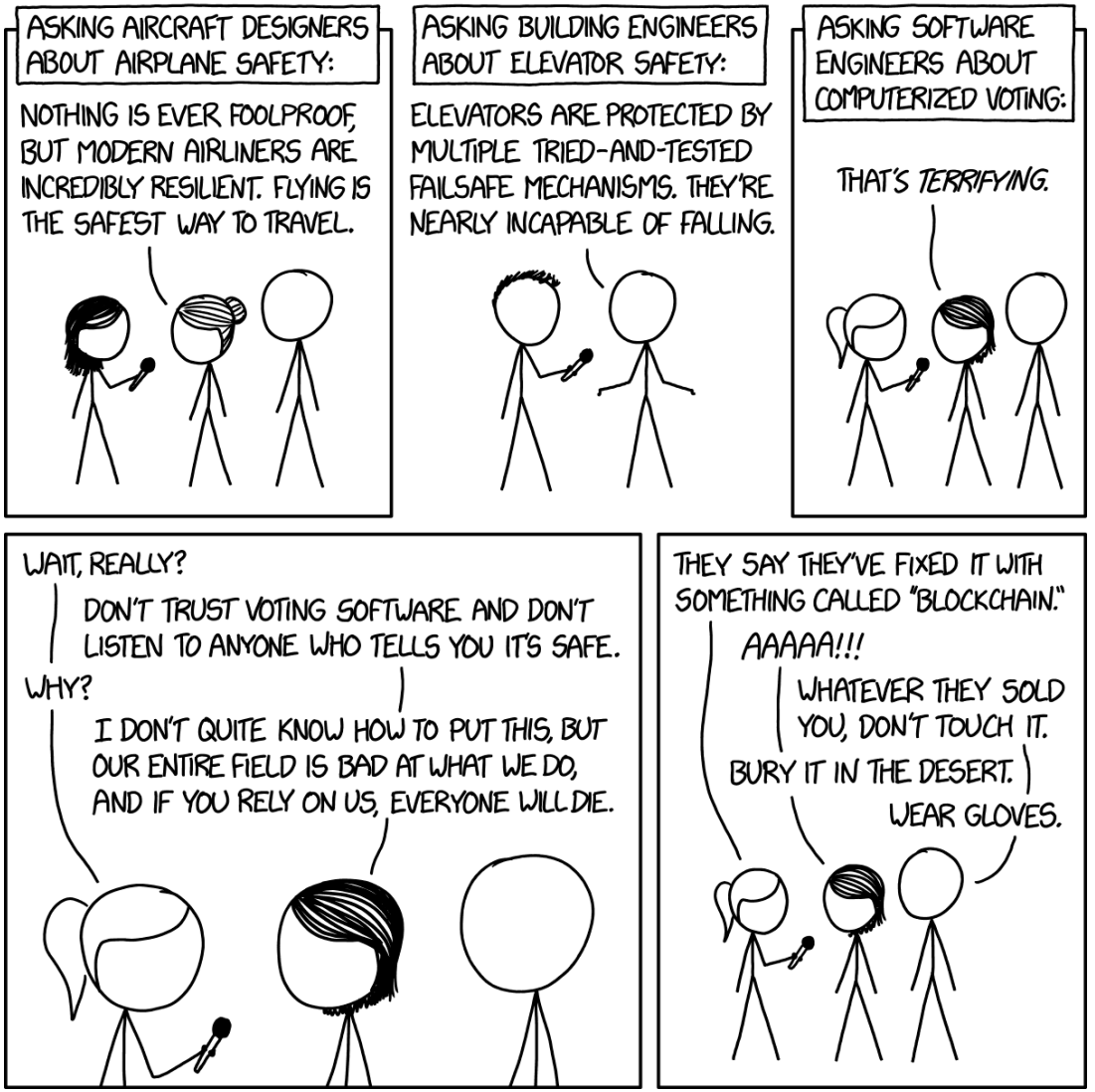
# Taxonomy of blockchains/2



# Applications

- Electronic currencies (bitcoin, ethereum, libra, etc.)
- Smart contracts (ethereum, libra, hyperledger)
- Asset tracking (hyperledger)
- Manufacturing quality control
- E-voting (?)
- etc.

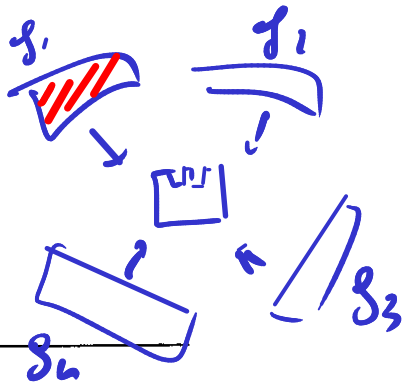
# Uses of blockchains



# Byzantine agreement

## The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International



Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement. It is shown that, using only oral messages, this problem is solvable if and only if more than two-thirds of the generals are loyal; so a single traitor can confound two loyal generals. With unforgeable written messages, the problem is solvable for any number of generals and possible traitors. Applications of the solutions to reliable computer systems are then discussed.

Categories and Subject Descriptors: C.2.4. [**Computer-Communication Networks**]: Distributed Systems—*network operating systems*; D.4.4 [**Operating Systems**]: Communications Management—*network communication*; D.4.5 [**Operating Systems**]: Reliability—*fault tolerance*

General Terms: Algorithms, Reliability

Additional Key Words and Phrases: Interactive consistency

# Byzantine agreement

A byzantine agreement protocol is a distributed protocol among agents $A_1, \ldots, A_k$ such that

1. all non-faulty/honest agents terminate the protocol in a finite number of steps;
2. all non-faulty/honest agents agree upon termination on the same decision;
3. different decisions are possible.

**Note**

The decision taken is *irrelevant* as far as all non-faulty agents agree.

# Byzantine fault

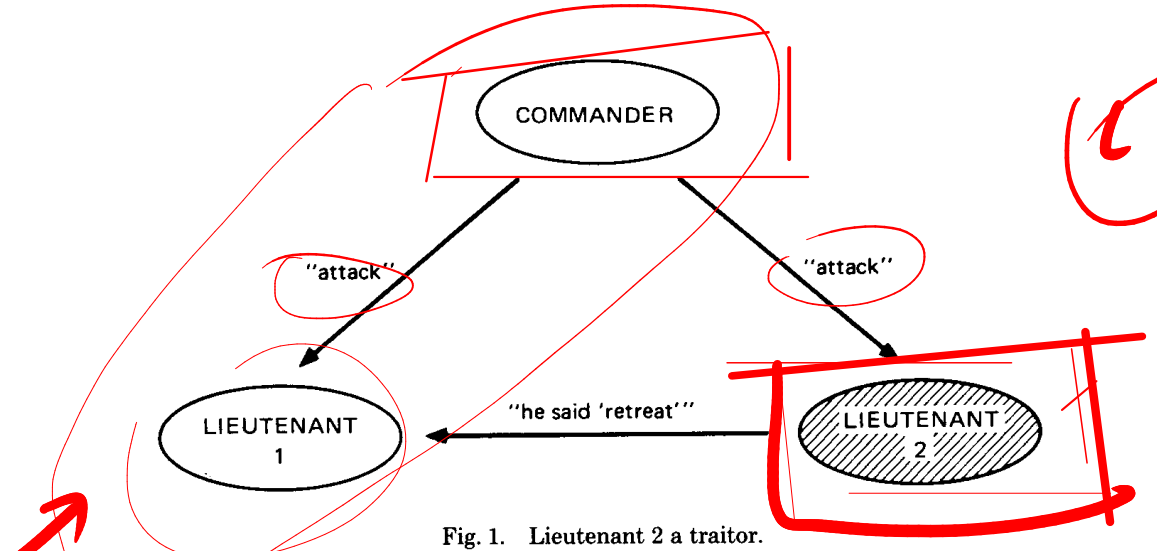



Fig. 1. Lieutenant 2 a traitor.

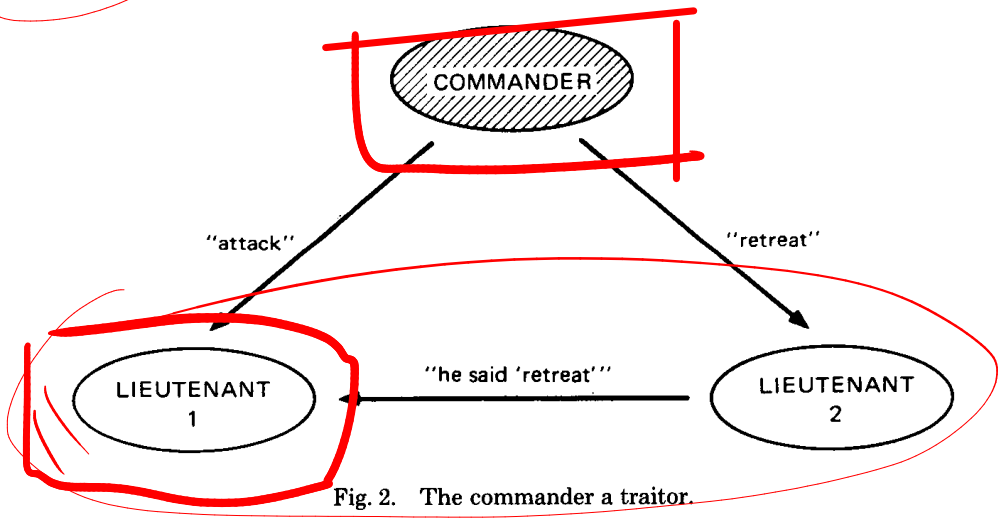



Fig. 2. The commander a traitor.

# Byzantine agreement

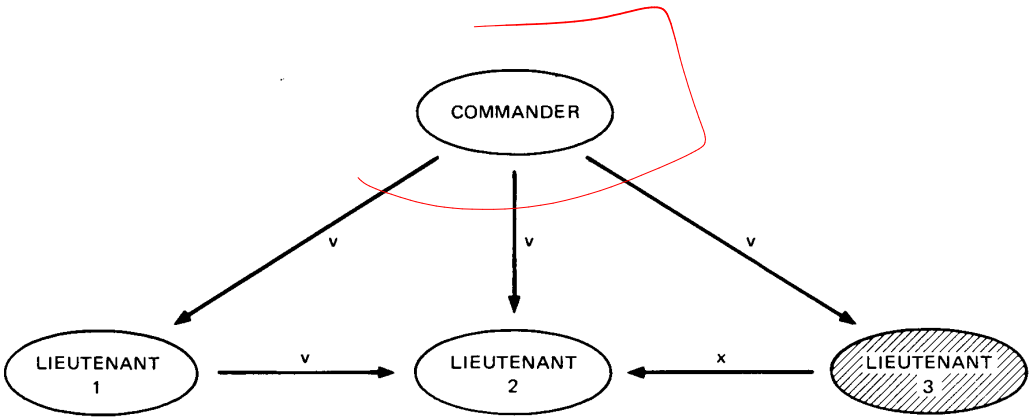



Fig. 3. Algorithm OM(1); Lieutenant 3 a traitor.

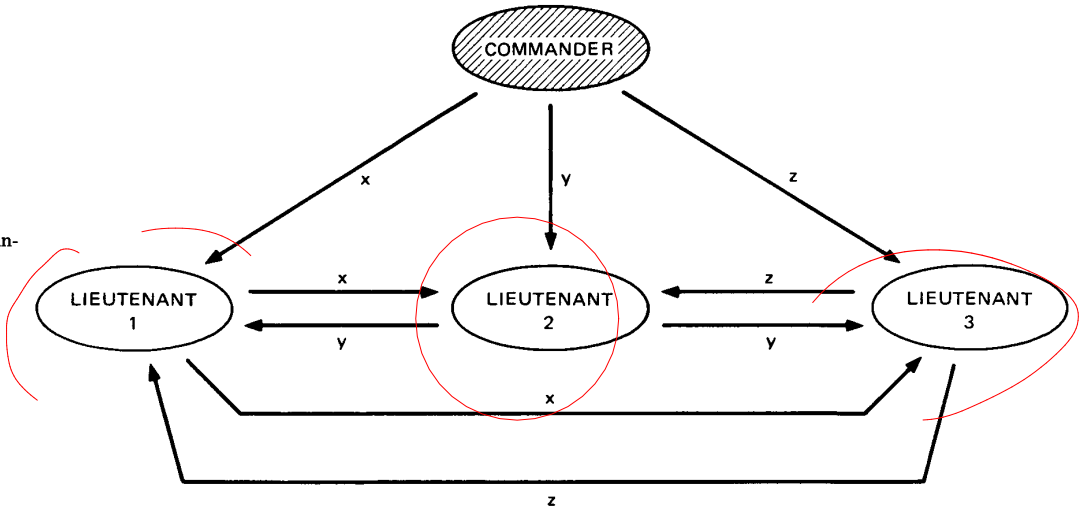



Fig. 4. Algorithm OM(1); the commander a traitor.

# Byzantine agreement/limits

## Consensus in the Presence of Partial Synchrony

CYNTHIA DWORK AND NANCY LYNCH

*Massachusetts Institute of Technology, Cambridge, Massachusetts*

AND

LARRY STOCKMEYER

*IBM Almaden Research Center, San Jose, California*

Abstract. The concept of partial synchrony in a distributed system is introduced. Partial synchrony lies between the cases of a synchronous system and an asynchronous system. In a synchronous system, there is a known fixed upper bound $\Delta$ on the time required for a message to be sent from one processor to another and a known fixed upper bound $\Phi$ on the relative speeds of different processors. In an asynchronous system no fixed upper bounds $\Delta$ and $\Phi$ exist. In one version of partial synchrony, fixed bounds $\Delta$ and $\Phi$ exist, but they are not known a priori. The problem is to design protocols that work correctly in the partially synchronous system regardless of the actual values of the bounds $\Delta$ and $\Phi$. In another version of partial synchrony, the bounds are known, but are only guaranteed to hold starting at some unknown time $T$, and protocols must be designed to work correctly regardless of when time $T$ occurs. Fault-tolerant consensus protocols are given for various cases of partial synchrony and various fault models. Lower bounds that show in most cases that our protocols are optimal with respect to the number of faults tolerated are also given. Our consensus protocols for partially synchronous processors use new protocols for fault-tolerant "distributed clocks" that allow partially synchronous processors to reach some approximately common notion of time.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*; *distributed databases*; *network operating systems*; C.4 [**Computer Systems Organization**]: Performance of Systems—*reliability, availability, and serviceability*; H.2.4 [**Database Management**]: Systems—*distributed systems*

# Byzantine agreement/limits



TABLE I. SMALLEST NUMBER OF PROCESSORS $N_{min}$ FOR WHICH A $t$-RESILIENT CONSENSUS PROTOCOL EXISTS

| Failure type | Synchronous | Asynchronous | Partially synchronous communication and synchronous processors | Partially synchronous communication and processors | Partially synchronous processors and synchronous communication |
|---|---|---|---|---|---|
| Fail-stop | $t$ | $\infty$ | $2t + 1$ | $2t + 1$ | $t$ |
| Omission | $t$ | $\infty$ | $2t + 1$ | $2t + 1$ | $[2t, 2t + 1]$ |
| Authenticated Byzantine | $t$ | $\infty$ | $3t + 1$ | $3t + 1$ | $2t + 1$ |
| Byzantine | $3t + 1$ | $\infty$ | $3t + 1$ | $3t + 1$ | $3t + 1$ |

al più $\frac{1}{3}$ di disonesti

# Byzantine agreement with signatures

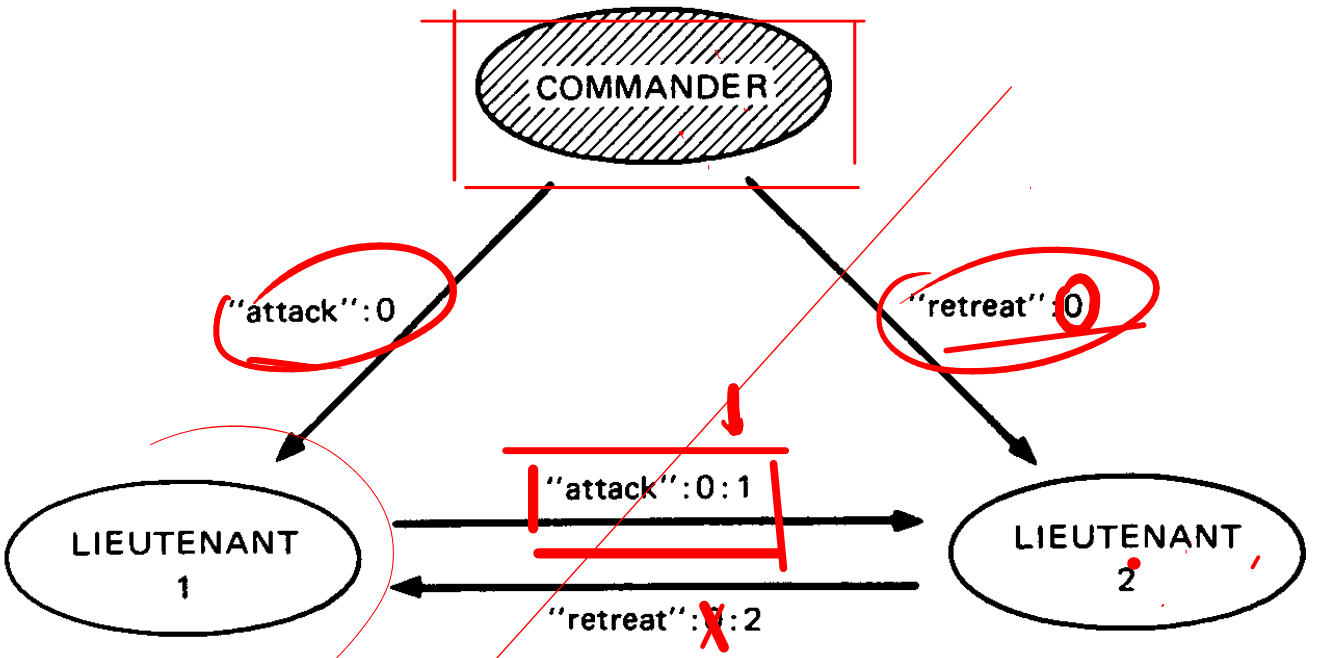
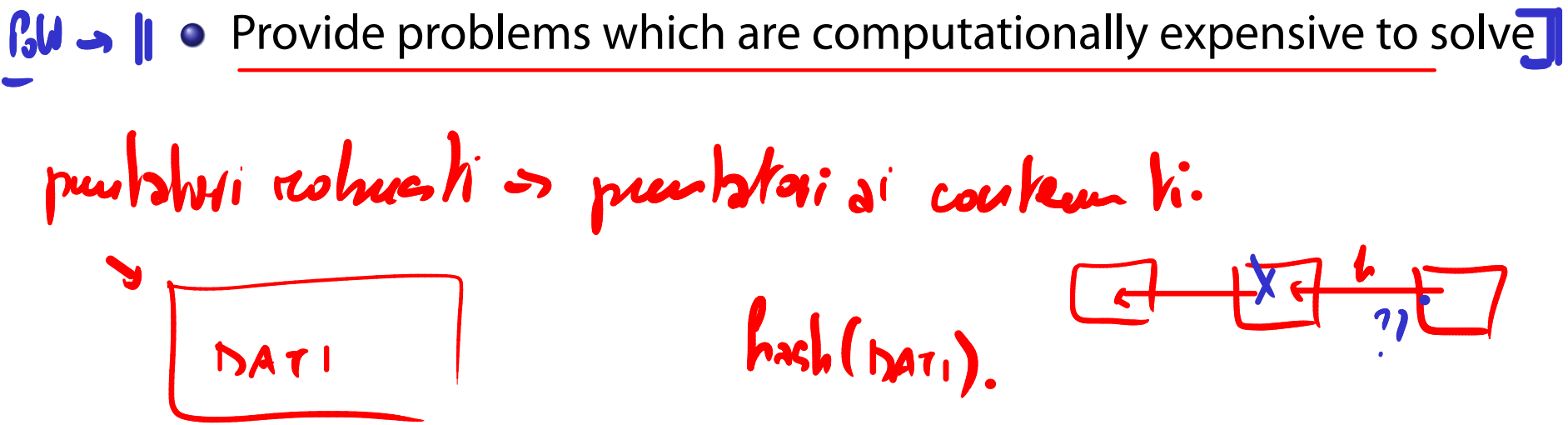



Fig. 5. Algorithm SM(1); the commander a traitor.

# Hash functions (MDC)

- Provide *digests* of messages to simplify signatures
- Provide a way to construct *robust pointers*
- Provide problems which are computationally expensive to solve



# Bitcoin

## Bitcoin: A Peer-to-Peer Electronic Cash System

"valuta" indipendente dagli stati e dalle banche.

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

**Abstract.** A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

### 1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as

# Bitcoin

- Electronic cash not baked by external entities
- Based on a blockchain which is
  1. Public
  2. Permissionless
  3. Distributed

# Bitcoin/2

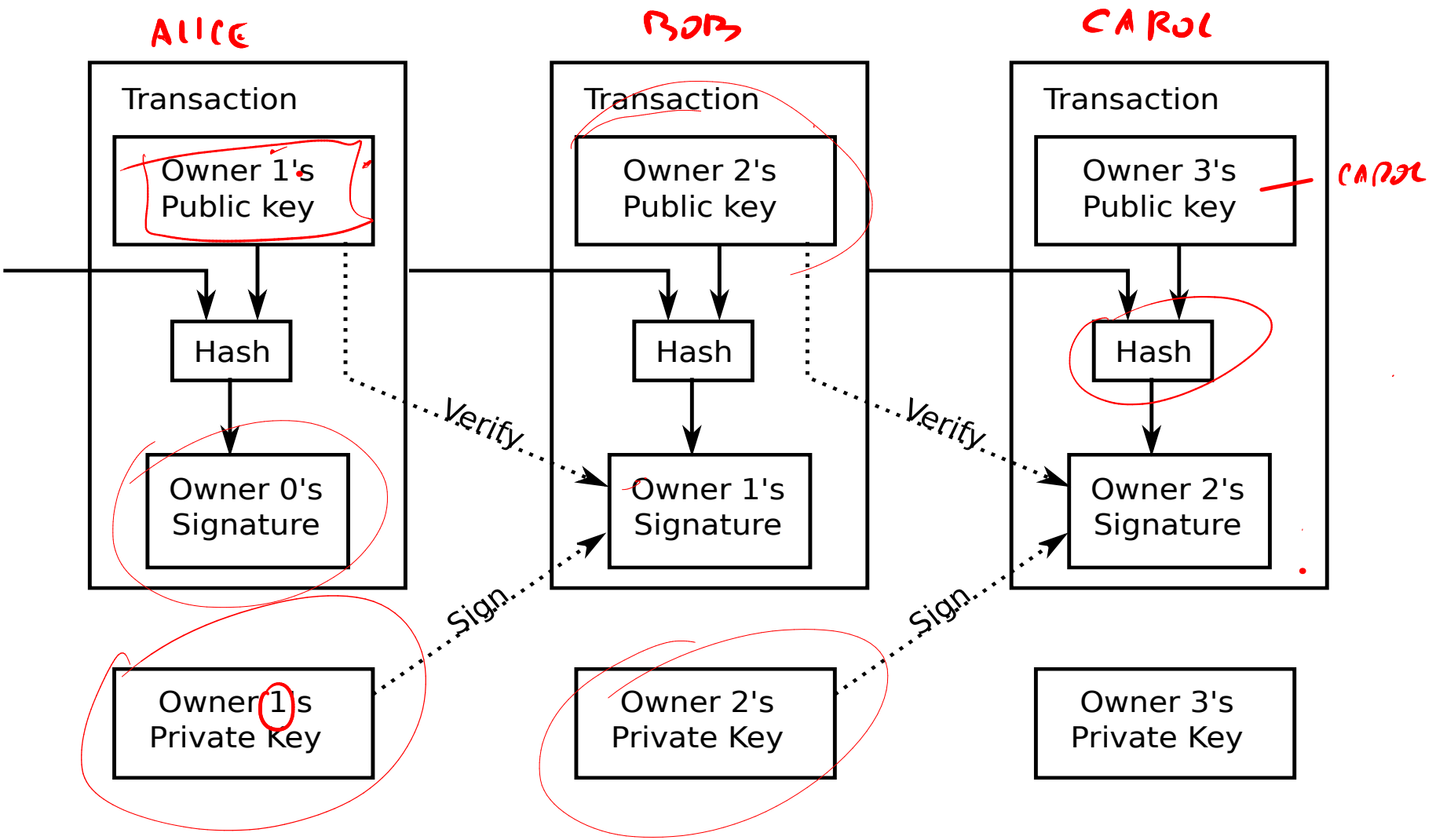
- Fully decentralized and peer-to-peer;
- Database: list of all transactions between pseudonimous accounts;
- Blocks are linked by means of hash functions acting as pointers;
- Transactions are validated by means of digital signatures;
- Consensus is reached by proof-of-work.

garantisce l'integrità

# Bitcoin/transactions

- People in bitcoin are identified by their public keys.
- Suppose Alice to have keys $(pk_A, sk_A)$ and Bob to have keys $(pk_B, sk_B)$.
- A payment form Alice to Bob is a message $m$ containing as payee $pk_B$ signed with $sk_A$ and indicating a certain number of bitcoins to be transferred.
- The database is updated by subtracting the number of bitcoins paid by Alice from the account $pk_A$ and crediting the same amount on the account $pk_B$.
- Digital signatures guarantee authentication.
- Transactions are batched in *blocks*.

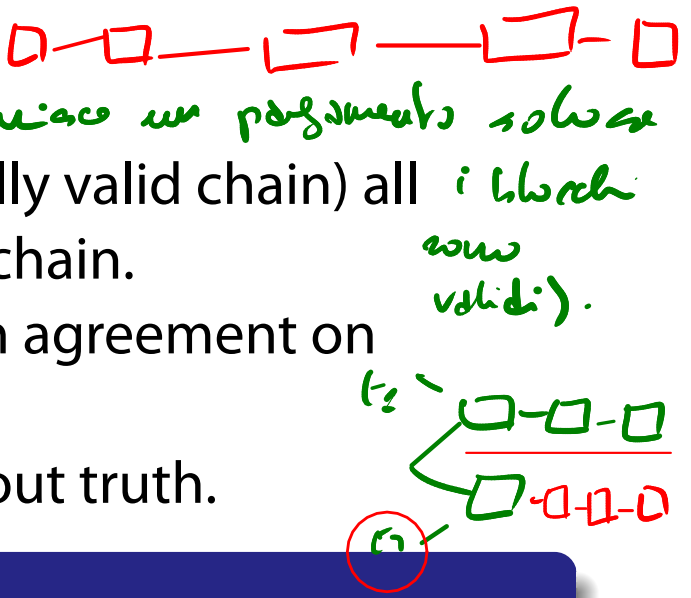
# Transactions



# Bitcoin/consensus: proof of work

How to guarantee consensus? (against malicious agents or errors)

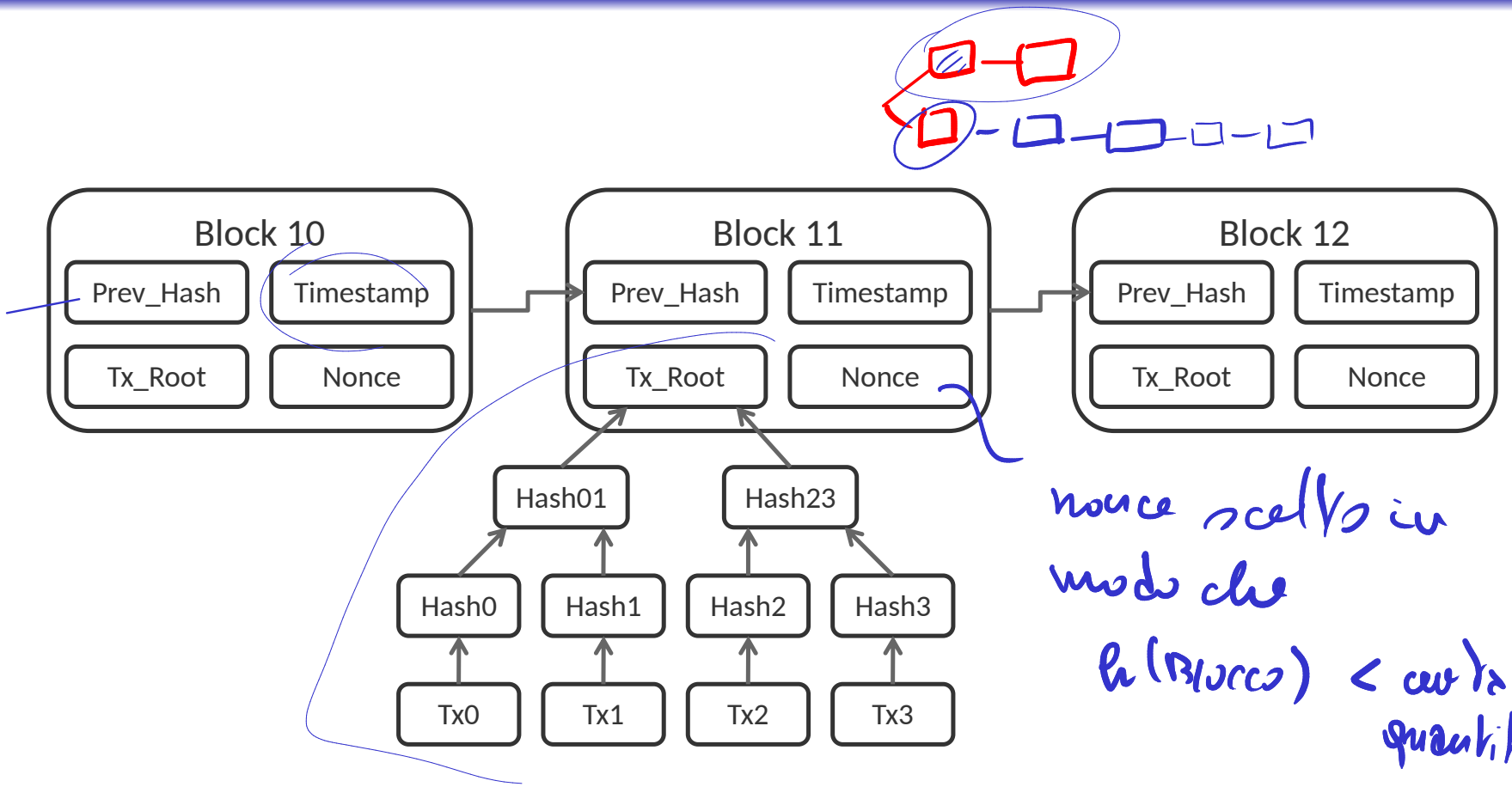
- All transaction blocks must be appended to a linear chain (blockchain).
- Appending blocks is expensive.
- If there is a fork (more than one potentially valid chain) all honest agents must choose the longest chain.
- Ultimately all honest agents will reach an agreement on transactions *deep enough*.
- We want *consistency* and do not care about truth.



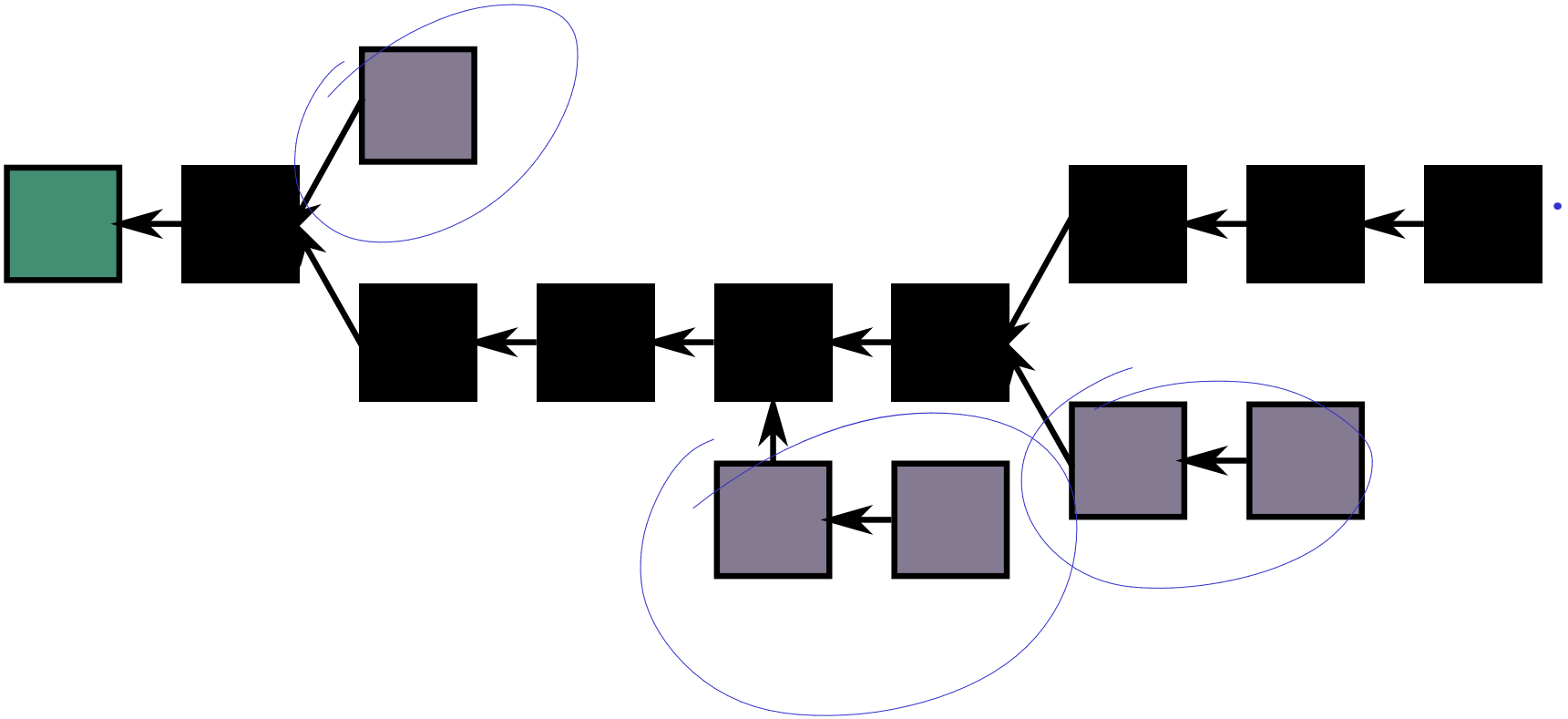
## Note

We are not interested in whether Alice has *really* paid Bob or not but we want that for all agents it is true that either Alice has been debited and Bob credited or Alice has not been debited and Bob has not been credited.

# Mining


Block 10
Prev_Hash
Timestamp
Tx_Root
Nonce
Block 11
Prev_Hash
Timestamp
Tx_Root
Nonce
Block 12
Prev_Hash
Timestamp
Tx_Root
Nonce
Hash01
Hash23
Hash0
Hash1
Hash2
Hash3
Tx0
Tx1
Tx2
Tx3
nonce scelto in modo che h(Blocco) < certa quantità.
MINING = collisioni parziali su di un hash.

# Forks



# Proof of Work

## Pricing via Processing or Combatting Junk Mail

Cynthia Dwork and Moni Naor

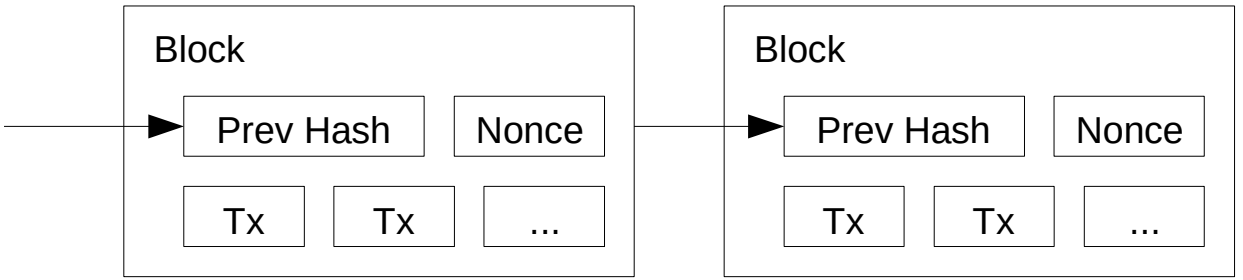
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120

**Abstract.** We present a computational technique for combatting junk mail in particular and controlling access to a shared resource in general. The main idea is to require a user to compute a moderately hard, but not intractable, function in order to gain access to the resource, thus preventing frivolous use. To this end we suggest several *pricing functions*, based on, respectively, extracting square roots modulo a prime, the Fiat-Shamir signature scheme, and the Ong-Schnorr-Shamir (cracked) signature scheme.

# Proof of Work (PoW)

- In order to append a block to the chain it is necessary to solve a computationally hard problem. → costoso!
- The first to solve the problem has the right to append the block.
- Each participant chooses as valid chain the longest available.
- To alter the contents of a block it would be necessary to solve several PoW problems faster than the growth of the chain.

# Bitcoin/PoW




## Work to be done

- Determine a `Nonce` such that the hash of the *combined block* is less than $N = 2^h$ (for suitable $h$).

nature sustainability

ANALYSIS

https://doi.org/10.1038/s41893-018-0152-7

# Quantification of energy and carbon costs for mining cryptocurrencies

Max J. Krause[1*] and Thabet Tolaymat[2]

**There are now hundreds of cryptocurrencies in existence and the technological backbone of many of these currencies is blockchain—a digital ledger of transactions. The competitive process of adding blocks to the chain is computation-intensive and requires large energy input. Here we demonstrate a methodology for calculating the minimum power requirements of several cryptocurrency networks and the energy consumed to produce one US dollar's (US$) worth of digital assets. From 1 January 2016 to 30 June 2018, we estimate that mining Bitcoin, Ethereum, Litecoin and Monero consumed an average of 17, 7, 7 and 14 MJ to generate one US$, respectively. Comparatively, conventional mining of aluminium, copper, gold, platinum and rare earth oxides consumed 122, 4, 5, 7 and 9 MJ to generate one US$, respectively, indicating that (with the exception of aluminium) cryptomining consumed more energy than mineral mining to produce an equivalent market value. While the market prices of the coins are quite volatile, the network hashrates for three of the four cryptocurrencies have trended consistently upward, suggesting that energy requirements will continue to increase. During this period, we estimate mining for all 4 cryptocurrencies was responsible for 3–15 million tonnes of $CO_2$ emissions.**

Decentralized cryptocurrencies represent a potentially revolutionary new technology for securely transferring money or information from one entity to another[1–4]. Many cryptocurrencies utilize blockchain, a public ledger, to accurately and continuously record transactions among many decentralized nodes[5]. A process of consensus, or agreement, is performed by 'miners' (44 TWh yr$^{-1}$ in 2017)[13], but significantly lower estimates also exist (4–5 TWh yr$^{-1}$ in 2017)[14]. All of these estimates indicate that cryptocurrencies already consume a non-negligible fraction of the world's energy production.

With Bitcoin energy demand now estimated to be equivalent to some countries, new questions arise. Do all cryptocurren-

# Proof of Work/costs

Joule

CellPress

Article

# The Carbon Footprint of Bitcoin

Christian Stoll,[1,2,5,*] Lena Klaaßen,[3] and Ulrich Gallersdörfer[4]

## SUMMARY

**Participation in the Bitcoin blockchain validation process requires specialized hardware and vast amounts of electricity, which translates into a significant carbon footprint. Here, we demonstrate a methodology for estimating the power consumption associated with Bitcoin's blockchain based on IPO filings of major hardware manufacturers, insights on mining facility operations, and mining pool compositions. We then translate our power consumption estimate into carbon emissions, using the localization of IP addresses. We determine the annual electricity consumption of Bitcoin, as of November 2018, to be 45.8 TWh and estimate that annual carbon emissions range from 22.0 to 22.9 $MtCO_2$. This means that the emissions produced by Bitcoin sit between the levels produced by the nations of Jordan and Sri Lanka, which is comparable to the level of Kansas City. With this article, we aim to gauge the external costs of Bitcoin and inform the broader debate on the costs and benefits of cryptocurrencies.**

## INTRODUCTION

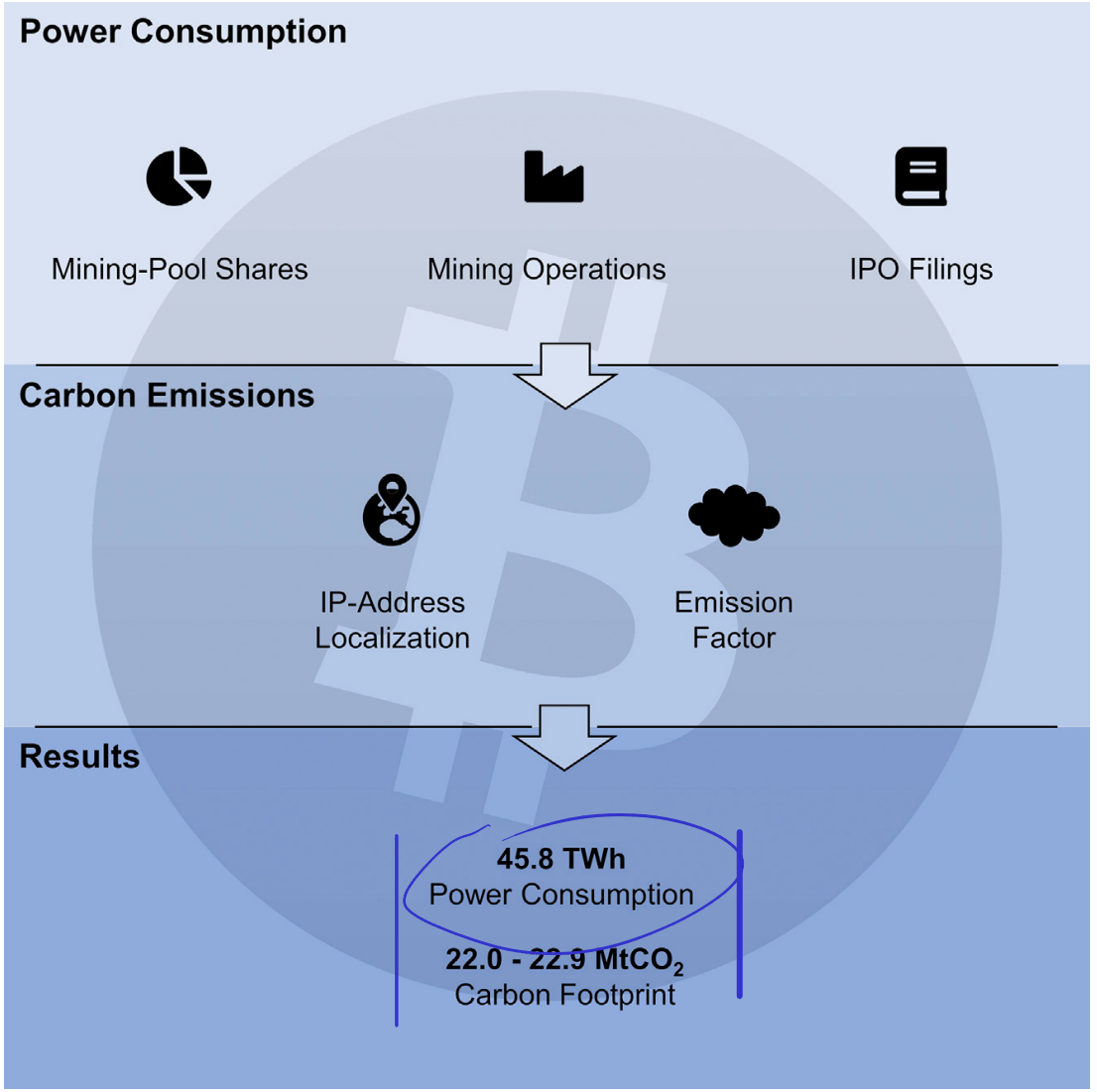
In 2008, Satoshi, the pseudonymous founder of Bitcoin, published a vision of a dig

## Context & Scale

Blockchain technology has its roots in the cryptocurrency Bitcoin, which was the first successful attempt to validate transactions via a decentralized data protocol. This validation process requires vast amounts of electricity, which translates into a significant level of carbon emissions. Our approximation of Bitcoin's carbon footprint underlines the need to tackle the environmental externalities that result from cryptocurrencies.

Blockchain solutions are

# Proof of Work/costs



# Proof of Work/considerations

**Remark**

A Byzantine attacker in bitcoin aims to keep a fork alive; ultimately this costs more than the expected gain.

- Mining works well for e-currencies
- It does not work so well for:
  1. Tracking external items
  2. Validating code to be executed
  3. Enforcing fairness

# P2P consensus algorithms

- Proof of Work
- Proof of Burn
- Proof of Stake
- Proof of Authority (endorsement)

# Practical Byzantine Fault tolerance

Appears in the *Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, USA, February 1999*

## Practical Byzantine Fault Tolerance

Miguel Castro and Barbara Liskov
*Laboratory for Computer Science,*
*Massachusetts Institute of Technology,*
*545 Technology Square, Cambridge, MA 02139*
{castro,liskov}@lcs.mit.edu

### Abstract

This paper describes a new replication algorithm that is able to tolerate Byzantine faults. We believe that Byzantine-fault-tolerant algorithms will be increasingly important in the future because malicious attacks and software errors are increasingly common and can cause faulty nodes to exhibit arbitrary behavior. Whereas previous algorithms assumed a synchronous system or were too slow to be used in practice, the algorithm described in this paper is practical: it works in asynchronous environments like the Internet and incorporates several important optimizations that improve the response time of previous algorithms by more than an order of magnitude. We implemented a Byzantine-fault-tolerant NFS service using our algorithm and measured its performance. The results show that our service is only 3% slower than a standard unreplicated NFS.

### 1 Introduction

and replication techniques that tolerate Byzantine faults (starting with [19]). However, most earlier work (e.g., [3, 24, 10]) either concerns techniques designed to demonstrate theoretical feasibility that are too inefficient to be used in practice, or assumes synchrony, i.e., relies on known bounds on message delays and process speeds. The systems closest to ours, Rampart [30] and SecureRing [16], were designed to be practical, but they rely on the synchrony assumption for correctness, which is dangerous in the presence of malicious attacks. An attacker may compromise the safety of a service by delaying non-faulty nodes or the communication between them until they are tagged as faulty and excluded from the replica group. Such a denial-of-service attack is generally easier than gaining control over a non-faulty node.

Our algorithm is not vulnerable to this type of attack because it does not rely on synchrony for

# Practical Byzantine Fault tolerance

Several phases:

1. Request: a client sends a request for an update to a primary;
2. Pre-Prepare: the primary notifies the backups that has received a request for a given view *v* and assigns a request number;
3. Prepare: all backups which accepted pre-prepare enter the prepare phase by multicasting a message with the sequence number and the view number;
4. Commit: once a replica has received a sufficient number of prepare messages sends a commit message to the others;
5. Reply: the primary and the backup reply to the request.

# Practical Byzantine Fault tolerance

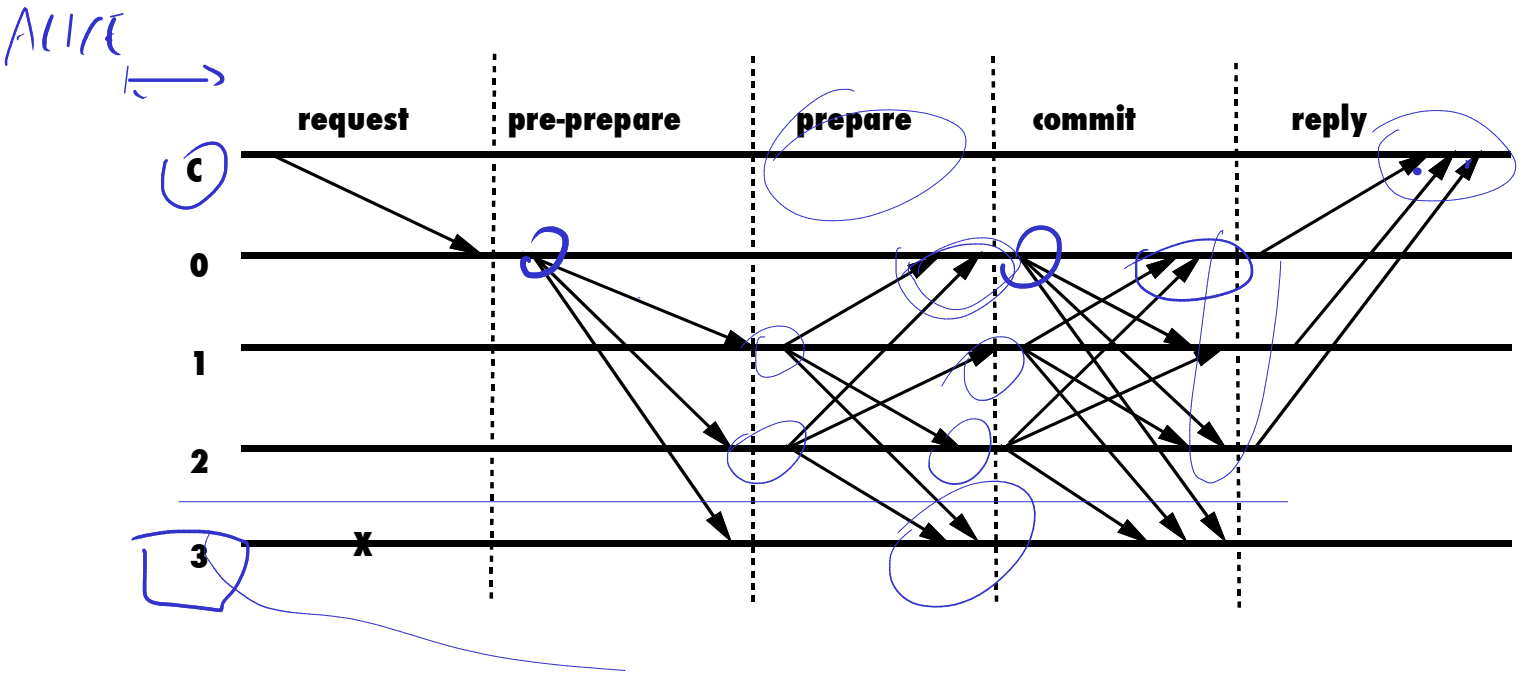



Figure 1: Normal Case Operation

# Redundant Byzantine Fault tolerance

2013 IEEE 33rd International Conference on Distributed Computing Systems

## RBFT: Redundant Byzantine Fault Tolerance

Pierre-Louis Aublin
Grenoble University

Sonia Ben Mokhtar
CNRS - LIRIS

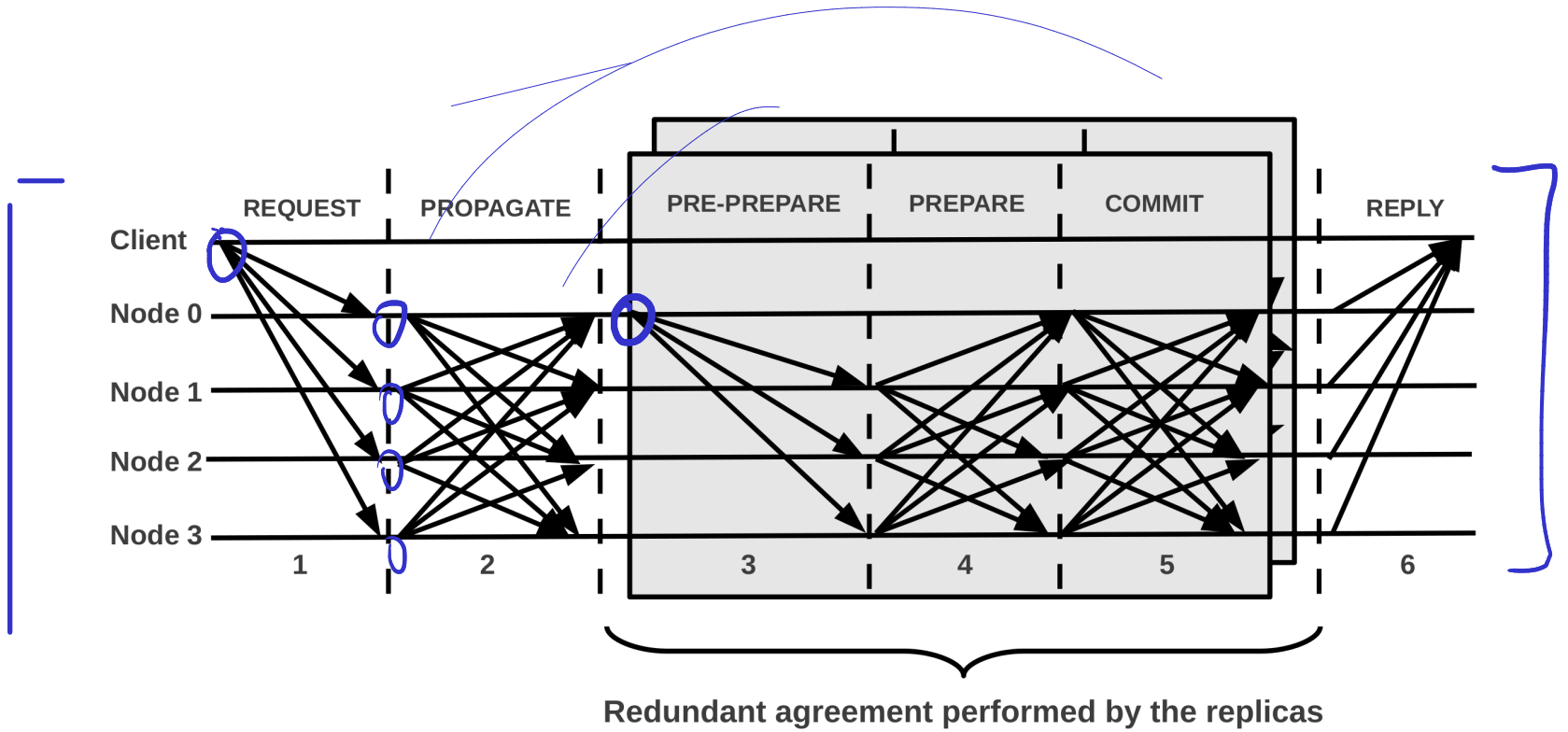
Vivien Quéma
Grenoble INP

***Abstract*—Byzantine Fault Tolerant state machine replication (BFT) protocols are replication protocols that tolerate arbitrary faults of a fraction of the replicas. Although significant efforts have been recently made, existing BFT protocols do not provide acceptable performance when faults occur. As we show in this paper, this comes from the fact that all existing BFT protocols targeting high throughput use a special replica, called the primary, which indicates to other replicas the order in which requests should be processed. This primary can be *smartly* malicious and degrade the performance of the system without being detected by correct replicas. In this paper, we propose a new approach, called RBFT for Redundant-BFT: we execute multiple instances of the same BFT protocol, each with a primary replica executing on a different machine. All the instances order the requests, but only the requests ordered by one of the instances, called the master instance, are actually executed. The performance of the different instances is closely monitored, in order to check that the master instance provides adequate performance. If that is not the case, the primary replica of the master instance is considered malicious and replaced. We implemented RBFT and compared its performance to that of other existing robust protocols. Our evaluation shows that RBFT achieves similar performance as the most robust protocols when there is no failure and that, under faults, its maximum performance degradation is about 3%, whereas it is at least equal to 78% for existing protocols.**

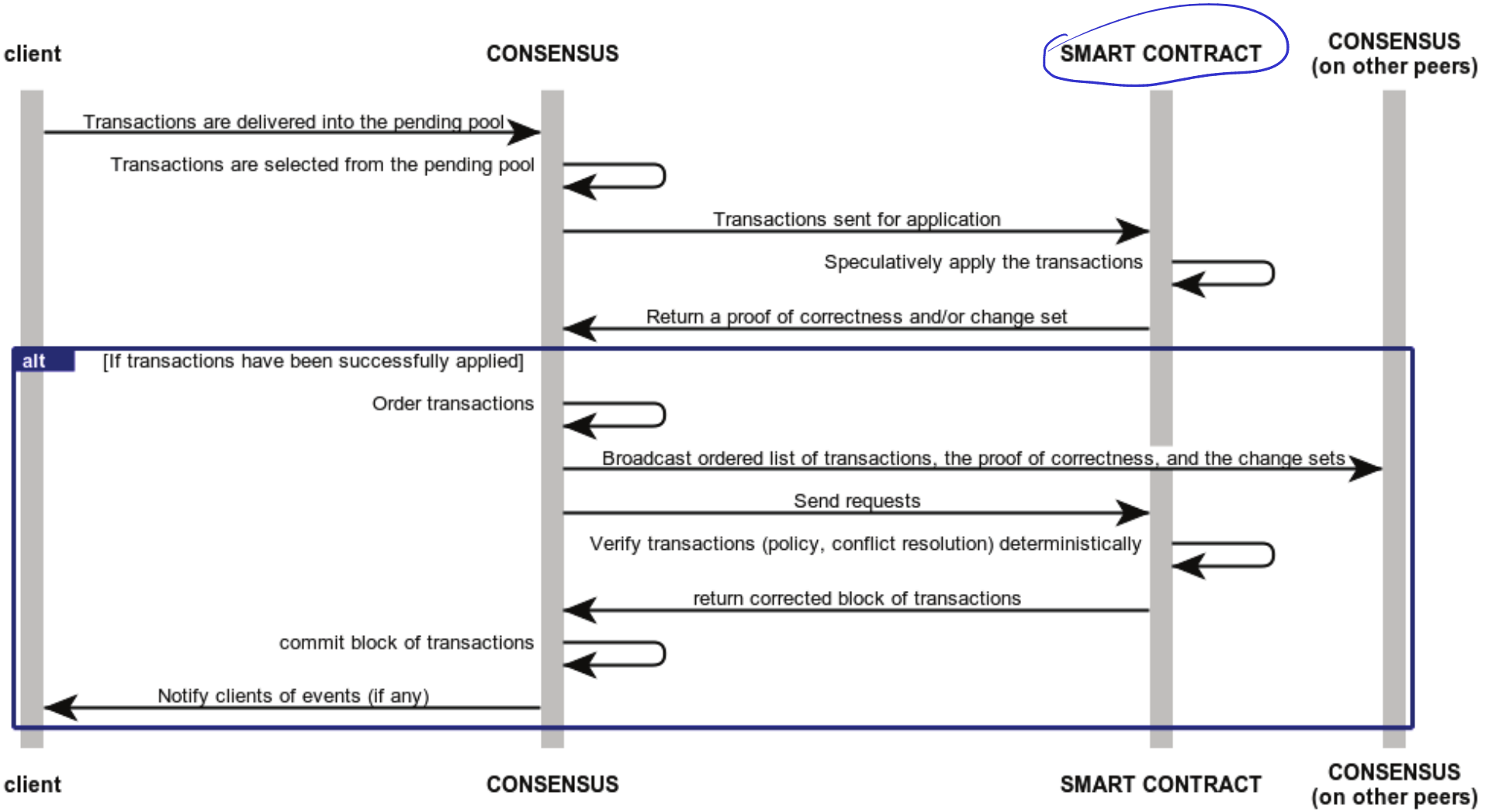
### I. Introduction

Byzantine Fault Tolerant (BFT) state machine replication to order requests. Even if there exists several mechanisms to detect and recover from a malicious primary, the primary can be *smartly* malicious. Despite efforts from other replicas to control that it behaves correctly, it can slow the performance down to the detection threshold, without being caught. To design a really robust BFT protocol, a legitimate idea that comes to mind is to avoid using a primary. One such protocol has been proposed by Boran and Schiper [4]. This protocol has a theoretical interest, but it has no practical interest. Indeed, the price to pay to avoid using a primary is that, before ordering every request, replicas need to be sure that they received a message from all other correct replicas. As replicas do not know which replicas are correct, they need to wait for a timeout (that is increased if it is not long enough). This yields very poor performance and this explains why this protocol has never been implemented. A number of other protocols have been devised to enforce intrusion tolerance (e.g., [18]). These protocols rely on what is called *proactive recovery*, in which nodes are periodically rejuvenated (e.g., their cryptographic keys are changed and/or a clean version of their operating system is loaded). If performed sufficiently often, node rejuvenation makes it difficult for an attacker to corrupt enough nodes to harm the system. These solutions are complementary to the robustness mechanisms studied in this

# Redundant Byzantine Fault tolerance



# Hyperledger

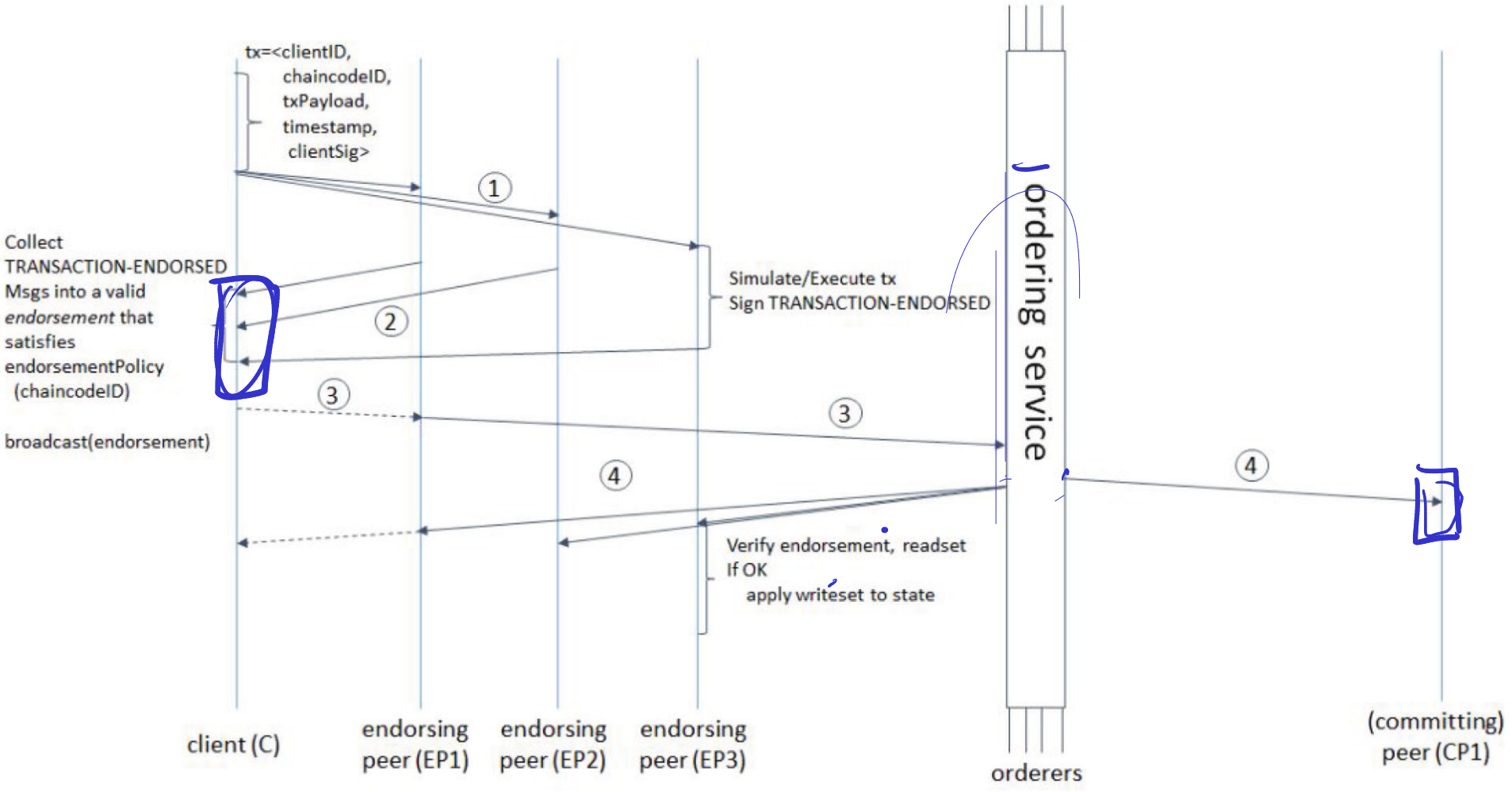


# Hyperledger

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



# Virtual Machines

- Blockchains implement VMs for increased flexibility
- These VMs can be either
  1. non-Turing complete (bitcoin)
  2. Turing complete with bounds on resource consumption (ethereum, libra, etc.)
- Blockchains as distributed computing frameworks;
- Blockchains as hosts for *smart contracts*.

# Smart contracts

Procedures

- safely stored on a platform (blockchain)
- automatically triggered by events
- audited and controlled only by the platform itself (not server-side)
- able to trigger new events.

**Remark**

In general a smart contract acts *only* on the state of the blockchain.

**Warning**

Smart contracts can be dangerous and hard to debug.

# Smart contracts

Preprint self-archived version. Published version forthcoming in *Bitcoin and Beyond* (Routledge).

## Chapter 8

## Experiments in Algorithmic Governance: A history and ethnography of "The DAO," a failed Decentralized Autonomous Organization

Quinn DuPont
University of Toronto

This chapter describes an emerging form of algorithmic governance, using the case study of "The DAO," a short-lived attempt to create a decentralized autonomous organization on the Ethereum blockchain platform. In June, 2016, The DAO launched and raised an unprecedented $250m USD in investment. Within days of its launch, however, The DAO was exploited and drained of nearly 3.7m Ethereum tokens.

This study traces the rise and fall of this emerging technology, and details the governance structures that were promised and hoped for, and those that were actually observed in its discourses. Through 2016-2017, these

# Applications to industry and robotics

- Auditability and tracking of manufacturing steps
- Robustness and replication of commands
- Distributed computation and collaborative logic
- Transparency and accountability
- Bidding and decentralized business models
- Economy of things (Machine to machine interaction)

# M2M interaction in Industry 4.0

- On demand manufacturing
- Auditing and diagnostics
- Traceability
- Authentication
- Subscription production
- Quality and stock control