

Distributed Systems and Consensus

50.037 Blockchain Technology
Paweł Szalachowski

“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.”

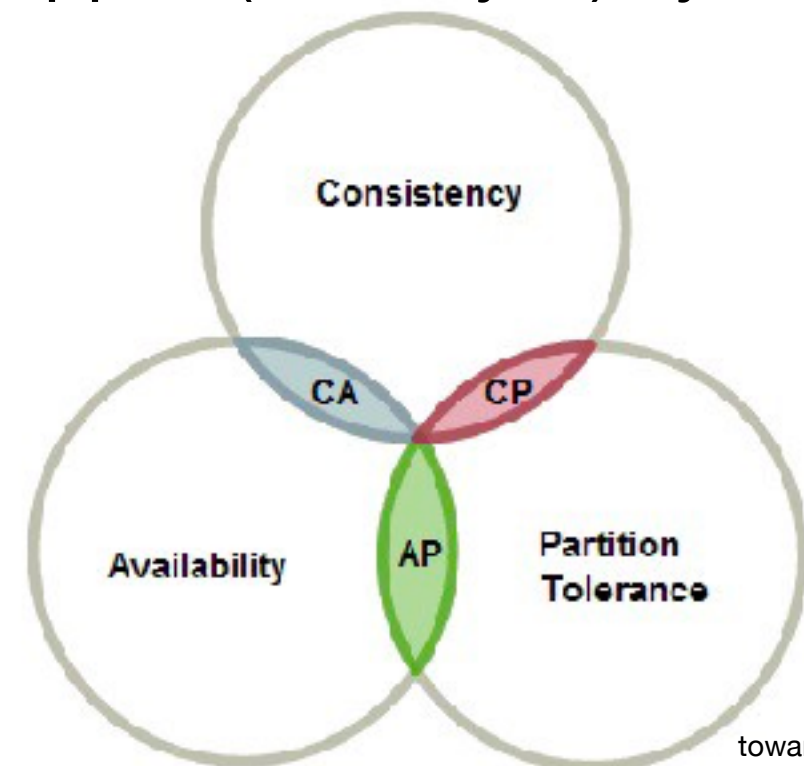
–Leslie Lamport

Intro

- Systems get large and complex
- Distributed system (selected topics)
 - CAP Theorem
 - Naming
 - Failure Tolerance and Consensus
 - PBFT-like Consensus
 - Nakamoto Consensus

CAP theorem

- *“it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees”* - Eric Brewer
- Consistency: Every read receives the most recent write or an error
- Availability: Every request receives a (non-error) response – without guarantee that it contains the most recent write
- Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes
- Examples?



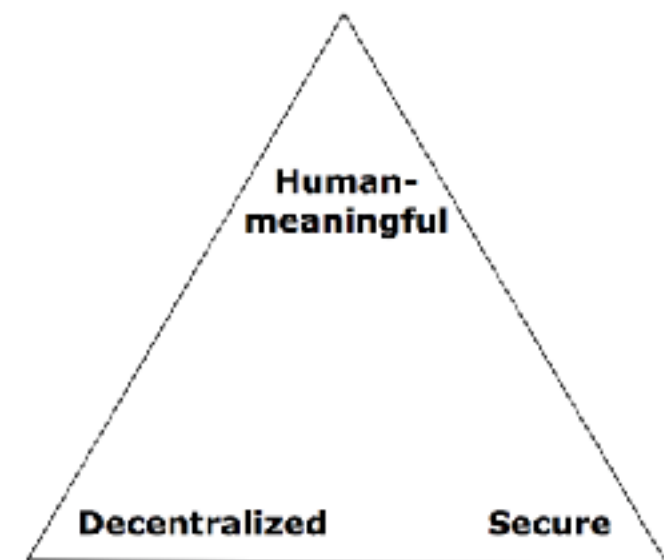
Naming

Distributed Naming

- Naming in distributed systems is hard
 - Especially for security
- How to name machines, organizations, persons, entities, ... ?
 - Name collision may lead to authentication/authorization failures
- Names exist in contexts
- What namespaces do you know?

Zooko's Triangle

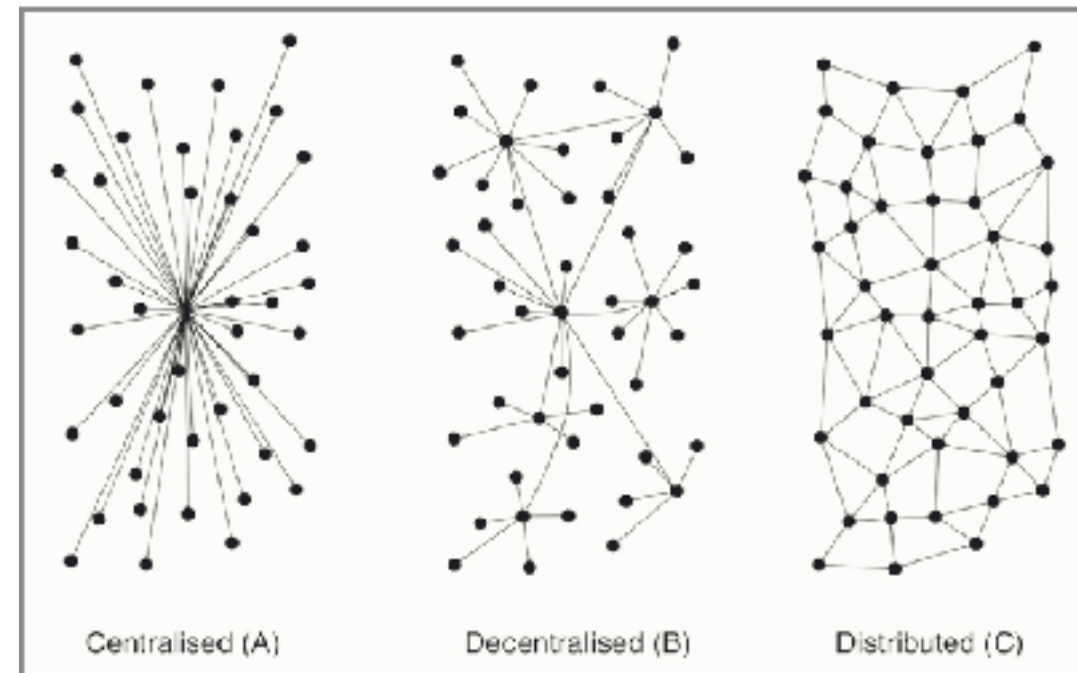
- No single kind of name can achieve more than two:
 - Human-meaningful: Meaningful and memorable (low-entropy) names are provided to the users.
 - Secure: The amount of damage a malicious entity can inflict on the system should be as low as possible.
 - Decentralized: Names correctly resolve to their respective entities without the use of a central authority or service.
- Examples
 - DNSSec
 - .onion and Self-certifying File System (SFS)
- It is believed that open and distributed consensus (blockchains) relaxes it



Ledger

System Types

- Computing, Trust, Network, Decision making, ...
- Centralized/Monarchy/Monopoly
 - One trusted node decides (can be replicated)
- Decentralized/Oligarchy/Oligopoly
 - Multiple trusted nodes can decide (each individually)
- Distributed/Open
 - Nodes collectively decide (no node is individually trusted)



Public Ledger

- Goal: design a public ledger
- Properties
 - Immutability, Transparency, Availability, Censorship-resistance, ...
- Potential
 - General-purpose global database (with arbitrary processing logic on the top)
 - All kinds of transactions, notaries, timestamping, state encoding, ...
- Challenge
 - How to provide these properties with (de)centralized systems?
 - Node have to *trusted* in some scope (e.g., availability)
 - Censorship-resistance may be problematic too
 - But then how to build a distributed ledger? Such that everyone can *write*
 - A large-scale consensus mechanism is inevitable
 - Incentives: who and why would run such an open infrastructure?

Consensus

Goals

- How N nodes can achieve consensus in the presence of a number of faulty nodes?
 - (nodes are also called processes, actors, participants, hosts, ...)
 - Different equivalent problem formulations (all about agreement)
- Properties
 - Safety: something bad will never happen
 - Agreement: all correct nodes select the same value
 - Liveness: something good will happen eventually
 - Termination: all correct nodes eventually decide

System Models

- Network: fully connected with message ordering controlled by adversary
- Timing
 - Synchronous: message sent at T is delivered by $T+d$, where d is known
 - Eventually Synchronous: message sent at T is delivered by $\max(T+d, T_g+d)$, where T_g is unknown
 - Asynchronous: sent messages are eventually delivered
- Faults: f nodes can be faulty (usually, relative to N)
 - Crash: would be honest but is (sometimes) unavailable
 - Byzantine: arbitrary (e.g., adversarial) behaving
- What would you assume for the Internet?

Many Bounds

- Two generals
 - Two-party agreement over unreliable medium is impossible
 - msg, ack, ack of ack, ack of ack of ack, ...
- Fischer, Lynch, and Patterson (FLP) Impossibility
 - *No (deterministic) consensus can be guaranteed (liveness and safety) in an asynchronous communication system in the presence of any failures*
 - Intuition: cannot distinguish between failing and slow nodes
 - What to do then?
 - Tweak the model a bit (timing, randomness, failure detectors,...)
- ... and much more: “A Hundred Impossibility Proofs for Distributed Computing”

Byzantine Consensus

Byzantine Nodes Bound

- Byzantine Failure Tolerance (Lamport, Shostak, and Pease)
 - N generals defending Byzantium, f of whom are malicious
 - Synchronous network with authentic and confidential (peer2peer) messages
 - What is max f that can be tolerated? $N \geq 3f + 1$
 - Node with inconsistent info cannot determine who is faulty (note, however, that with digital signatures it can be solved easily.)

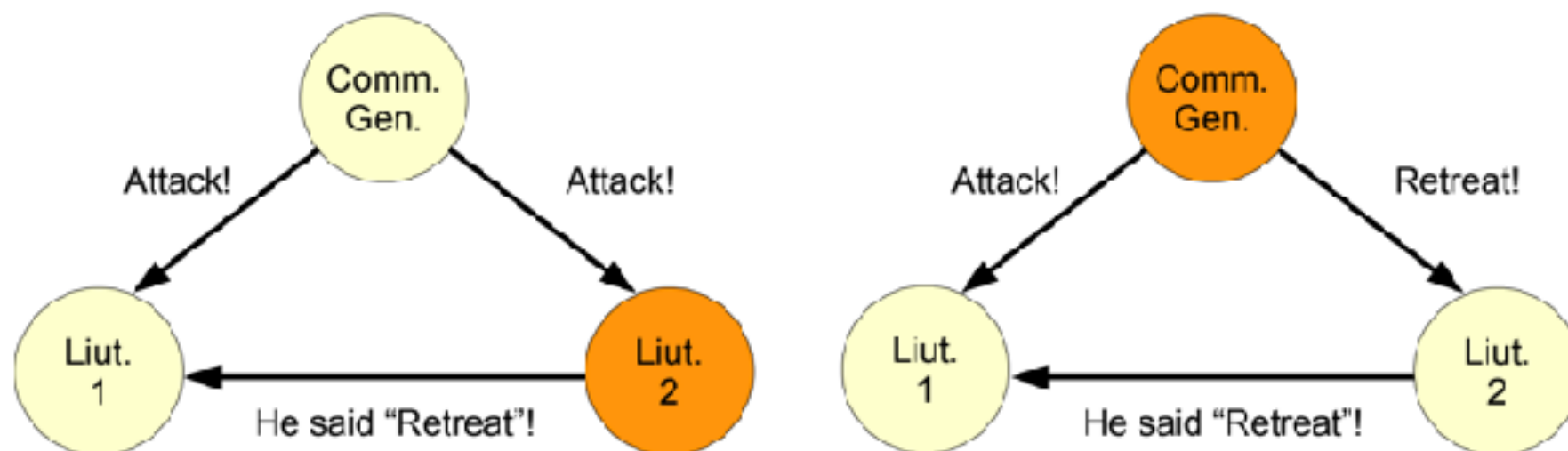


Fig: <http://disi.unitn.it/~montreso/>

Best-Effort Broadcast

Sender:

on input(x)
multicast (send, x)

Issues?

- unreliable communication?
- how does sender know which nodes received x ?

Receiver:

on receiving (send, x)
output x

Consistent Broadcast

- Sender has an input x to be broadcast
- Termination: if sender is honest, then every honest node outputs x
- Consistency: if any two nodes output x and y , then $x=y$
- Model
 - Asynchronous network
 - Byzantine faults with $f < N/3$

Consistent Broadcast

Sender:

on input(x)
multicast (send, x)

Receiver:

on receiving (send, x)
multicast (echo, x)

on receiving (echo, x) from $\geq (N+f+1)/2$ nodes
output x

Liveness:

With honest sender, $N-f$ honest nodes receive (send, x)
thus $N-f$ correct nodes multicast (echo, x)
thus each honest node receives $N-f$ echo msgs

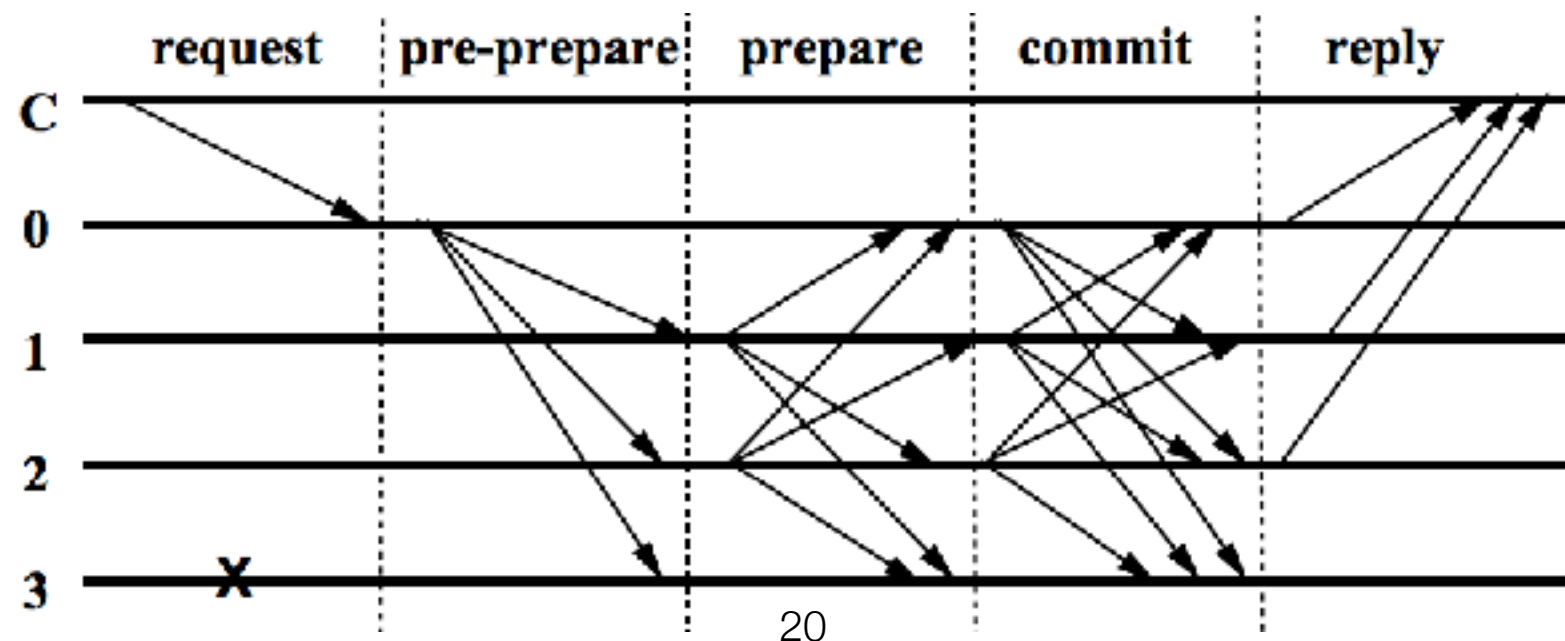
Safety:

Assume two honest nodes outputting $a \neq b$
Must have received $(N+f+1)$ echo msgs in total
At least $f+1$ nodes sent two conflicting echo msgs
That cannot happen as only f nodes can be faulty

Do you see any problem(s) of that protocol?
Faulty sender?

PBFT

- Castro and Liskov “Practical Byzantine Fault Tolerance”, 1999
 1. A client sends a request to invoke a service operation to the primary
 2. The primary multicasts the request to the backups
 3. Replicas execute the request and send a reply to the client
 4. The client waits for $f+1$ replies from different replicas with the same result; this is the result of the operation.



How to use?

- Implement a distributed key:value storage (filesystem)
 - Universal (easy to implement other primitives on top)
 - Distributed locks
 - Leader election (often protocols provide it by default)
 - Membership enumeration
 - ...
- Easy to combine with other services, load balancers, etc...
- Easy to implement a distributed ledger

Distributed Ledger

- Collect transactions to commit
- Elect leader and let the leader to append her view of transactions
 - Leader can be changed every round (why helpful?)
 - If leader dies new leader is elected
- Everyone synchronizes transactions and keeps replicated state of the system

Other protocols

- Paxos (prior PBFT)
 - Synchronization problem
 - Used by Google (Chubby, Spanner, Megastore, ...)
 - OpenReplica, IBM SAN Volume Controller, ...
- Raft
 - Designed as an alternative to Paxos
 - More understandable



Properties

- Scale to large # of transactions
 - Up to (several) thousands
- Scale only to small # of nodes
 - Only several to tens, due to $O(N^2)$ message complexity
- Resilient to 1/3 malicious nodes
- Needs known and static set of participants (identities)
 - Authority has to allow nodes to participate

Open Consensus

- So far consensus is closed/permissioned
 - Someone has to allow us to participate
- Can we just do it open/permissionless by allowing anyone to participate ?
 - First of all, scalability (limited # of nodes)
 - but even if (somehow) we can solve scalability, can we really open the protocol to anyone?
 - Voting, reputation systems,

Sybil Attack

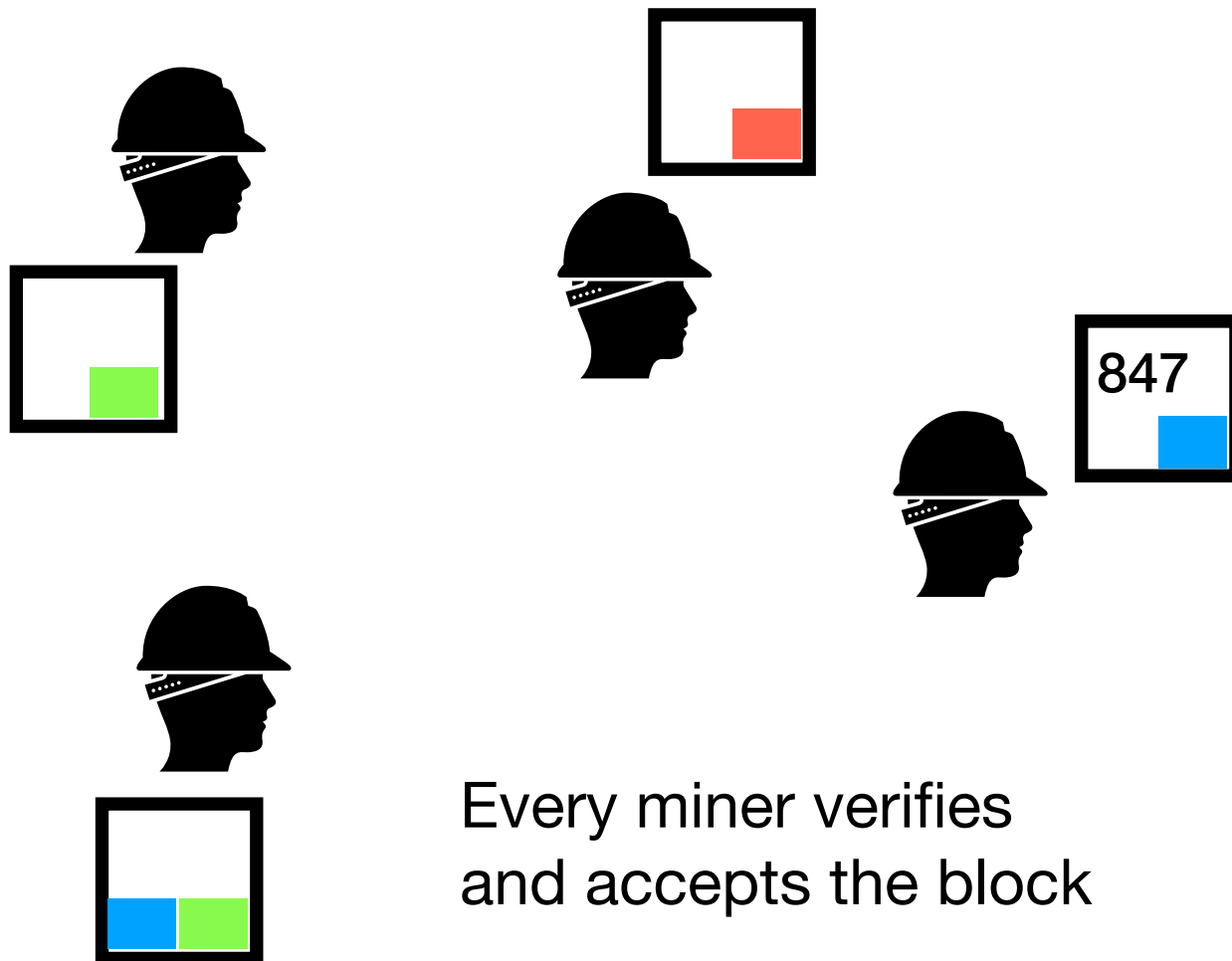
- Generic attack against open voting/reputation/p2p systems
 - No trusted authority that vouches identities
- Adversary creates many identities to attack system
 - It is cheap to create multiple identities in open system
 - e.g., generate keypairs, create email accounts, create multiple connections, imitate browser instances, ...
- How to validate identities in such a system?
 - It has to be based on resources (assuming adversary with limited resources)
 - Time (CPU), Space (Memory), and Bandwidth (Network Connection)

Nakamoto Consensus

Consensus via PoW

- Leader election per transactions set (block)
- Leaders are self-elected (vote for themselves)
- Voting is via PoW
 - Finding a block whose hash < target

Miners
(protocol participants)



f54ee8e79bad... H(1)

37f082290f4f7... H(2)

bb9c721da24... H(3)

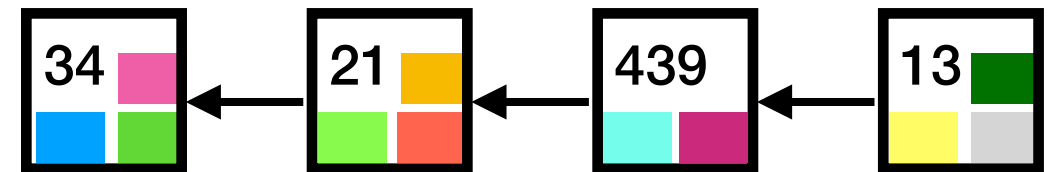
...

000000000c5... H(847)



Blockchain/Ledger

- Hash chain of blocks
- Block
 - Set of transactions
 - Forming hash tree
 - Special number (Nonce)
 - Hash pointer/link to the previous block
- Proof of work
 - $H(\text{block}) < \text{target}$ (target defines difficulty)
 - You can see leading zeros in hashes



$$H\left(\begin{array}{|c|c|} \hline 34 & \text{pink} \\ \hline \text{blue} & \text{green} \\ \hline \end{array}\right)$$

00000000892fcb17...

$$H\left(\begin{array}{|c|c|} \hline 21 & \text{yellow} \\ \hline \text{green} & \text{red} \\ \hline \end{array}\right)$$

00000000490747db...

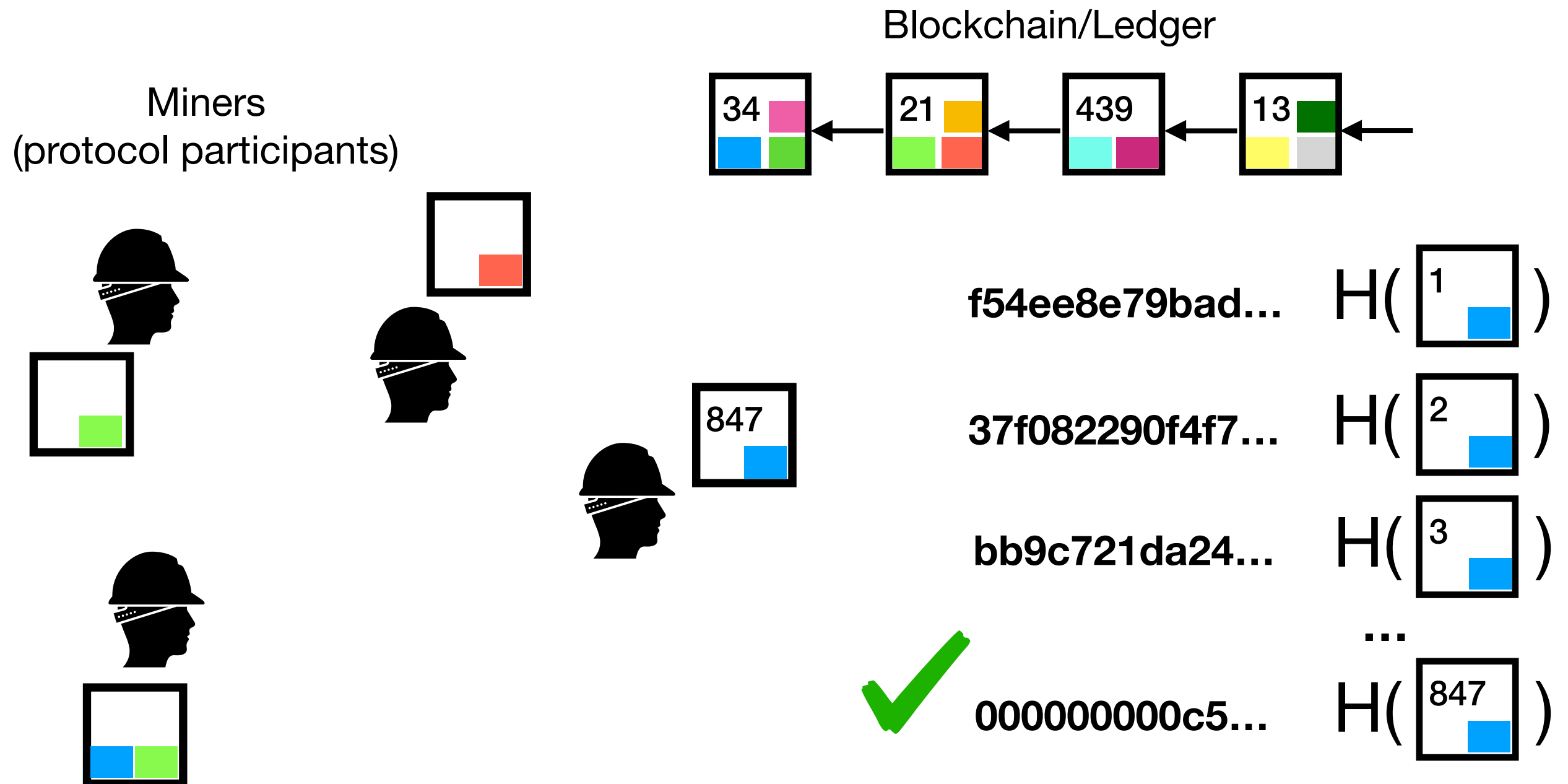
$$H\left(\begin{array}{|c|c|} \hline 439 & \text{cyan} \\ \hline \text{cyan} & \text{magenta} \\ \hline \end{array}\right)$$

00000000a4d587e0...

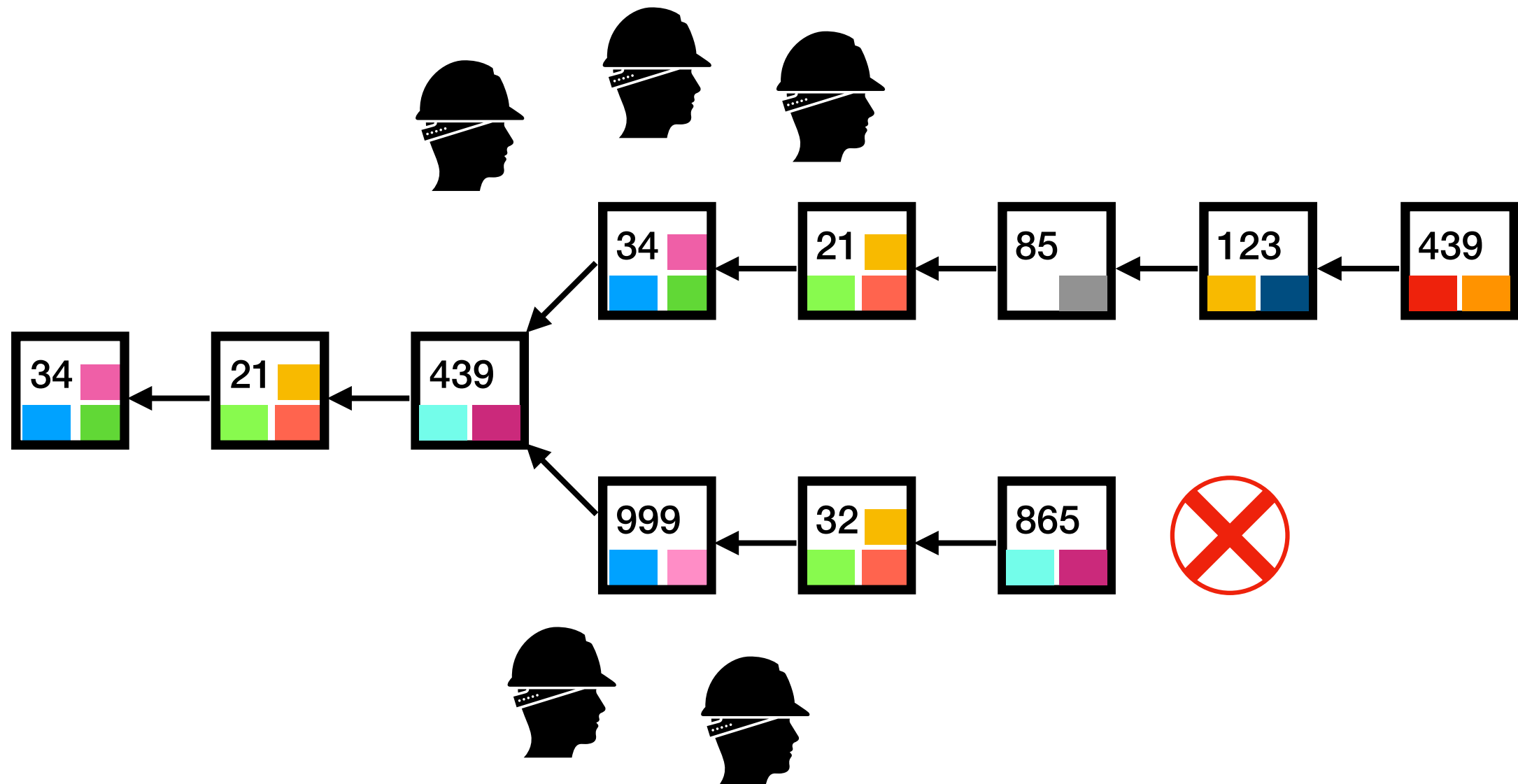
$$H\left(\begin{array}{|c|c|} \hline 13 & \text{green} \\ \hline \text{yellow} & \text{grey} \\ \hline \end{array}\right)$$

00000000ff410ef45...

Consensus & Blockchain



Forks



The longest* chain wins.

* in reality it is the chain with the most PoW accumulated.

Properties

- Open/Permissionless
 - No identities required (but usually public-keys are used as identifiers)
 - Leader are self-elected and set of nodes can be large, dynamic, and unknown (anyone can join freely)
- Extremely Robust
 - Very simple, small communication overhead, dynamic membership, ...
- Probabilistic: you never have 100% guarantee what the current chain is
 - To get a high evidence multiple confirmation blocks are required (e.g., 6 blocks ~ 1h)
- Security Assumption
 - Honest majority: 50% of mining power is honest (spoiler: actually, it is lower)
- Issues
 - Energy consumption: it should be non-trivial to find block (o/w many forks)
 - Target is adjusted to 10min/block, but that increases latency
 - Latency decreases scalability (# of transactions)

Reading

- Textbook: 1.4, 1.5, 2
- https://lpd.epfl.ch/site/_media/education/sdc_byzconsensus.pdf
- <http://pmg.csail.mit.edu/papers/osdi99.pdf>
- <https://bitcoin.org/bitcoin.pdf>
- <https://arxiv.org/pdf/1707.01873.pdf>
- <https://www.freehaven.net/anonbib/cache/sybil.pdf>