Distributed Consensus

Portland State
Computer Science

Motivation

- A rogue blockchain
  - Consider Peer B wishing to tamper with Block #3
    - https://anders.com/blockchain/distributed
    - Modifies data, then re-calculates hashes
- How can you pick between two valid blockchains?
  - With a consensus protocol!
  - Algorithm for supporting consensus is a fundamental design choice for any blockchain

Consensus

- Distributed agreement on state (including ordering of events) in the absence of a trusted, central authority
  - One of the hardest problems in CS
- Consensus protocol for achieving it needs two properties
  - Safety (consistency and correctness)
    - Each node is guaranteed the same state
    - Algorithm must behave identically to a single node system that executes each transaction atomically one at a time
  - Liveness
    - Each non-faulty node will eventually receive every submitted transaction, assuming that communication does not fail.

Lamport (1978)

- Leslie Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", CACM, July 1978
  - Seminal paper on causal ordering, synchronized clocks
  - Strict consistency (e.g. total ordering of events) requires absolute global time (impossible to implement)
  - Causal consistency
    - Achieve a looser event ordering
    - Writes that are causally related must be seen by all processes in the same order
    - Writes that are not causally related can occur in any order
  - How?
    - Use communication events to causally order events between processes
    - Logical timestamps on events determine partial ordering in a distributed system
    - Easier to achieve
  - Paper describes how to get causally consistent consensus in an asynchronous system (e.g. no synchronized clocks)
    - Assumes an *absence* of faults

Other kinds of consistency

- Sequential consistency
  - Result of any execution is the same as if all operations on data store were executed in some sequential order
- Eventual consistency
  - If no new updates, after some window, all accesses will return last updated value (DNS)

FLP impossibility result (1985)

- Fischer, Lynch, Paterson, "Impossibility of Distributed Consensus with One Faulty Process", JACM 1985
  - Governs all solutions to distributed consensus protocols

  "Consensus impossible in asynchronous network (i.e. unbounded message delays) with a deterministic protocol"

  - Inherent trade-off between liveness and consistency (can not have both) when attempting to achieve consensus in the presence of failures

- Note: From cloud class, directly related to the CAP theorem (consistency, availability, partition-tolerance) for distributed, replicated databases
  - You can only support 2 out of the 3

Non-byzantine consensus

Paxos, Raft

Motivation

- From distributed computing systems
  - "Can you implement a distributed database that can tolerate the failure of any number of its processes (possibly all of them) without losing consistency, and that will resume normal behavior when more than half the processes are again working properly?"

  - Essential problem that must be solved in horizontally scalable databases

Paxos (1989)

- Lamport's "The Part-Time Parliament", ACM Transactions on Computer Systems, May 1998, p. 133-169.
  - Replication protocol for distributed systems
- Fictional Greek island called Paxos with a part-time parliament
  - Lawmakers come and go frequently on vacation
  - A consistent, replicated ledger containing all laws that have been passed must be kept
  - If a majority of the legislators present for a sufficient time period, then any decree proposed by a legislator in the Chamber is passed
  - Every decree that has been passed should eventually appear in the ledger of every legislator in the Chamber.

- Written in 1989, submitted in 1990, published in 1998 (!)

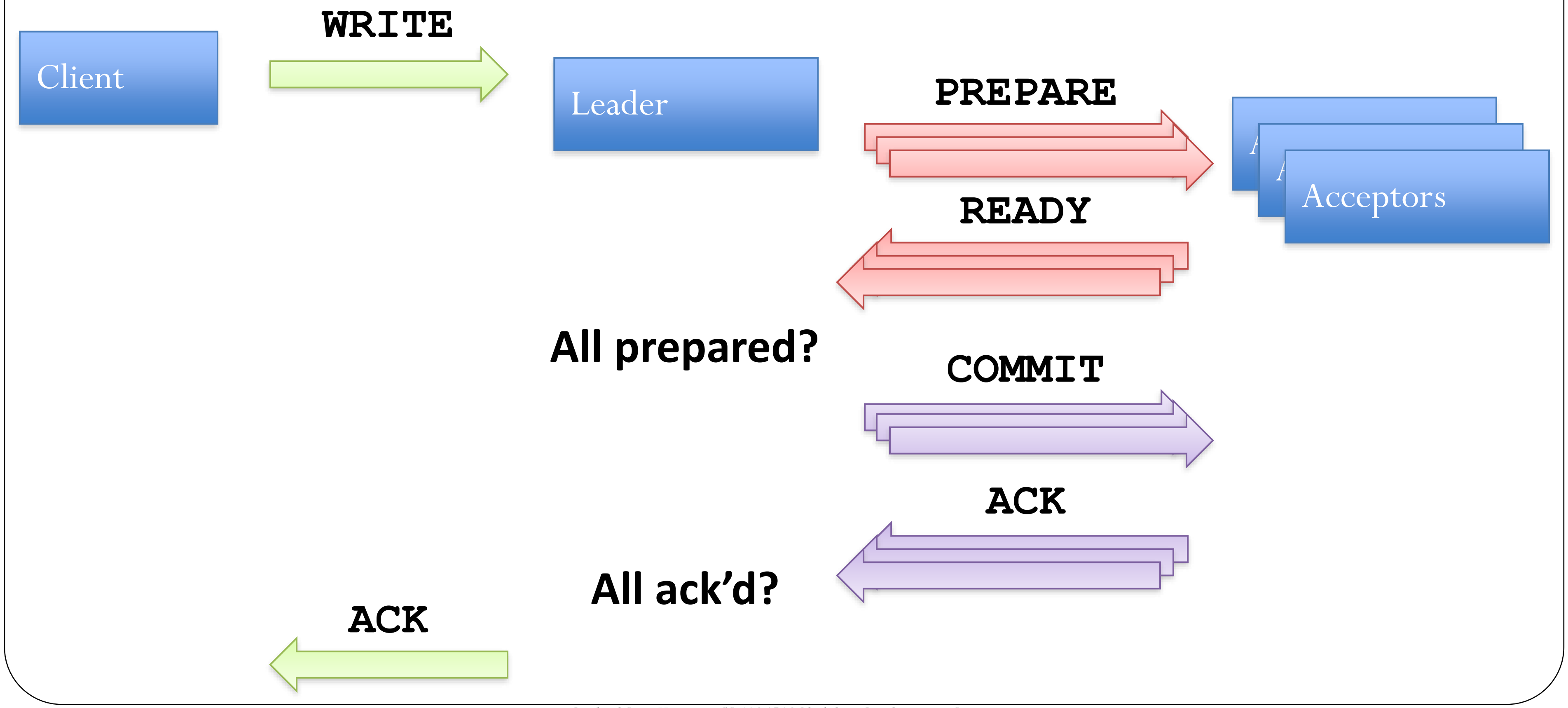
"I submitted the paper to TOCS in 1990. All three referees said that the paper was mildly interesting, though not very important, but that all the Paxos stuff had to be removed. I was quite annoyed at how humorless everyone working in the field seemed to be, so I did nothing with the paper."

Paxos characteristics

- Strong consistency in the presence of crash faults
  - All state changes are seen by all distributed processes in the same order (sequentially)
  - Uses a consensus protocol to avoid blocking in the presence of an arbitrary single failure
  - Provably correct if `N` processes want to agree on a value, when fewer than `F` crash faults occur in a window, if `N > 2F+1` processes (e.g. majority)
- Paxos and FLP
  - Chooses consistency over liveness (availability) in an asynchronous environment (network partition)
    - e.g. progress is blocked until a majority is present
    - "In an asynchronous environment that admits crash failures, no consensus protocol can guarantee termination, and the Synod protocol is no exception."

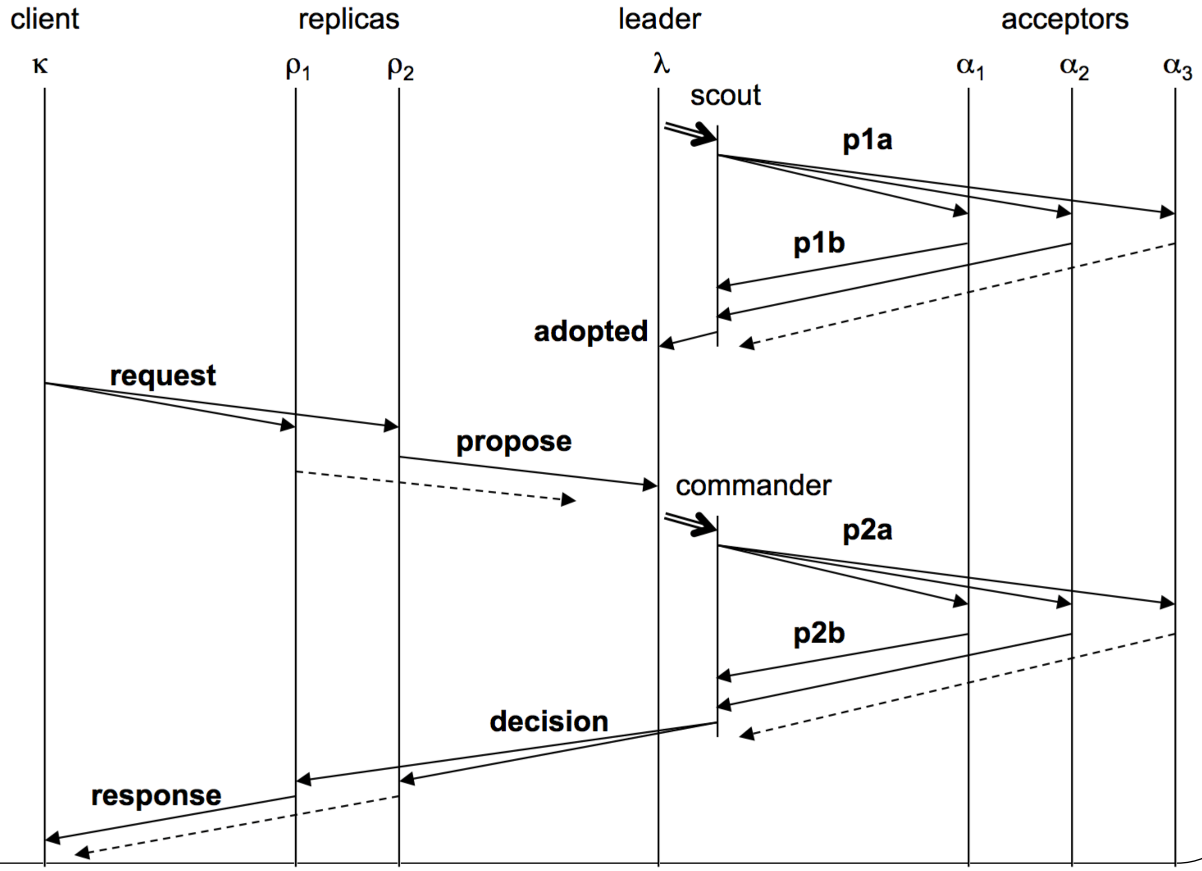
Paxos 2-phase commit protocol

- Similar to marriage protocol
  - Do you?
  - I do
  - I now pronounce you, Kiss the bride



Algorithm

- Run election protocol to define leader and acceptor groups (view change)
- Clients propose values to replicas
- Leader uses two-phase commit protocol with acceptors for writes from replicas proposing values
- Acceptors monitor leader for liveness and execute view change to elect new leader on failure

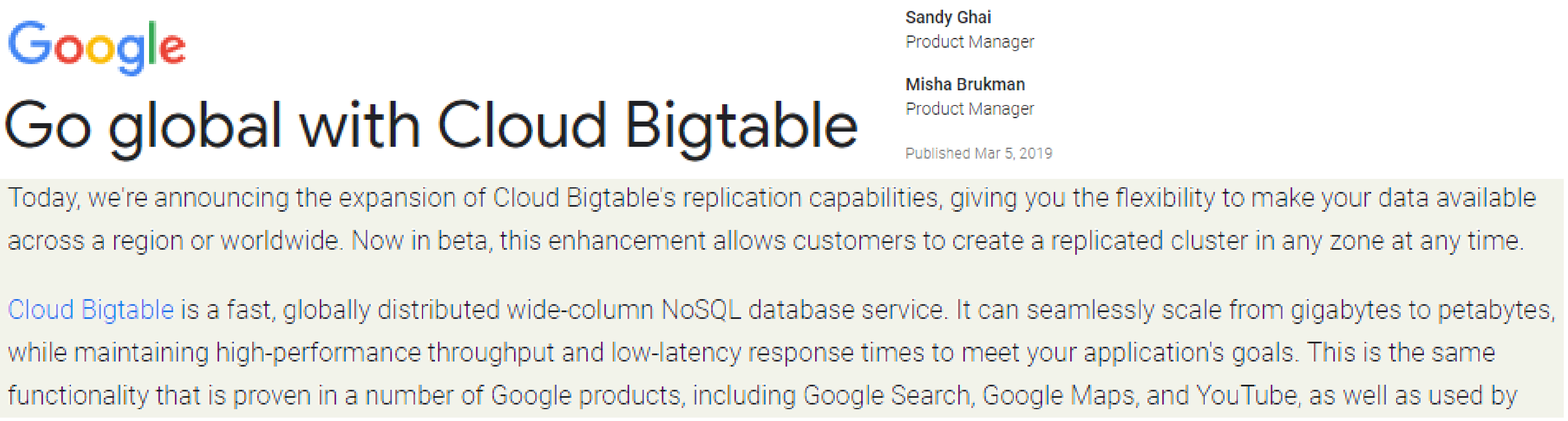


What about failures?

- If an acceptor fails:
- If the leader fails?
- What's the algorithm for picking a new leader?
  - Need to make sure everybody agrees on leader!
  - Need to make sure that "group" is known
- What happens when the leader election fails?
- Extremely complex due to corner cases…
- But…

Impact

- Underlies many distributed storage and database systems driving search, maps, gaming (http://paxos.systems/)
  - Used by Google in Spanner, BigTable
  - Used by Apache Hadoop FS, AWS ECS


Google

Go global with Cloud Bigtable

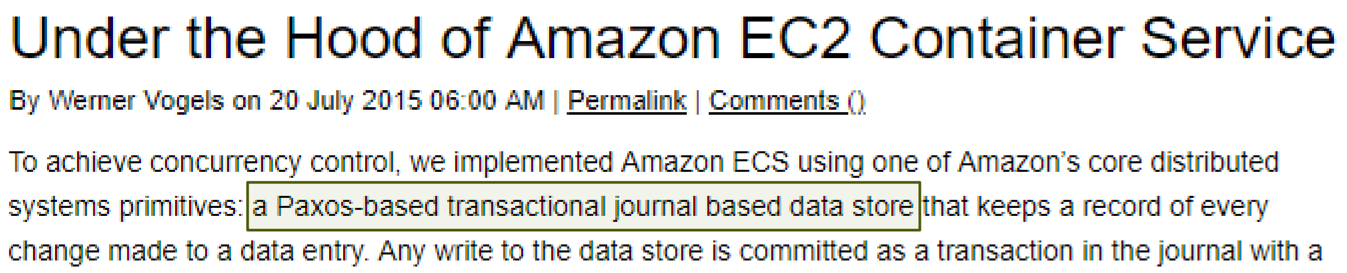
Sandy Ghai
Product Manager

Misha Brukman
Product Manager

Published Mar 5, 2019

Today, we're announcing the expansion of Cloud Bigtable's replication capabilities, giving you the flexibility to make your data available across a region or worldwide. Now in beta, this enhancement allows customers to create a replicated cluster in any zone at any time.

Cloud Bigtable is a fast, globally distributed wide-column NoSQL database service. It can seamlessly scale from gigabytes to petabytes, while maintaining high-performance throughput and low-latency response times to meet your application's goals. This is the same functionality that is proven in a number of Google products, including Google Search, Google Maps, and YouTube, as well as used by

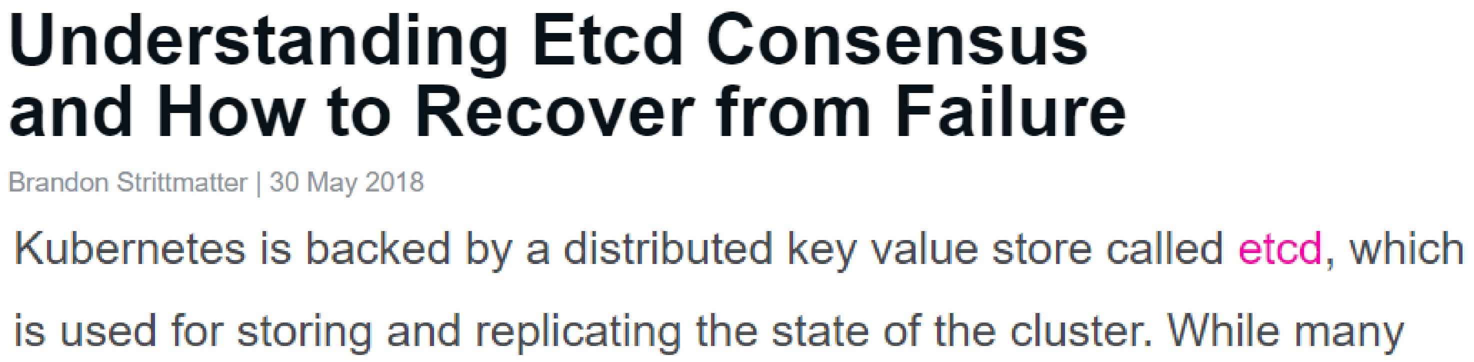

Under the Hood of Amazon EC2 Container Service

By Werner Vogels on 20 July 2015 06:00 AM | Permalink | Comments ()

To achieve concurrency control, we implemented Amazon ECS using one of Amazon's core distributed systems primitives: a Paxos-based transactional journal based data store that keeps a record of every change made to a data entry. Any write to the data store is committed as a transaction in the journal with a

Raft (2013)

- Alternative to Paxos by Engardo, Ousterhout
  - Built for understandability
    - https://raft.github.io/
  - Formally verified for safety
    - https://verdi.uwplse.org/raft-proof.pdf
  - As with Paxos
    - Quorum must exist to make progress
    - Leaders elected to coordinate consensus
    - Election occurs when a leader crashes
  - Underlies `etcd` (Kubernetes)



Understanding Etcd Consensus and How to Recover from Failure

Brandon Strittmatter | 30 May 2018

Kubernetes is backed by a distributed key value store called etcd, which is used for storing and replicating the state of the cluster. While many

Issues with Paxos, Raft

- Only handles crash-failures!
  - Faults consist of nodes that disconnect, can be declared dead, and are replaced
  - Appropriate for cloud environments with trusted participants
- Does not handle malicious nodes intentionally attempting to subvert state
  - What if a malicious node is elected leader?
  - Can a malicious node corrupt the election process indefinitely?
  - Inappropriate for public environments with untrusted participants (especially when $ involved!!)
  - Typically used for private and/or permissioned backends

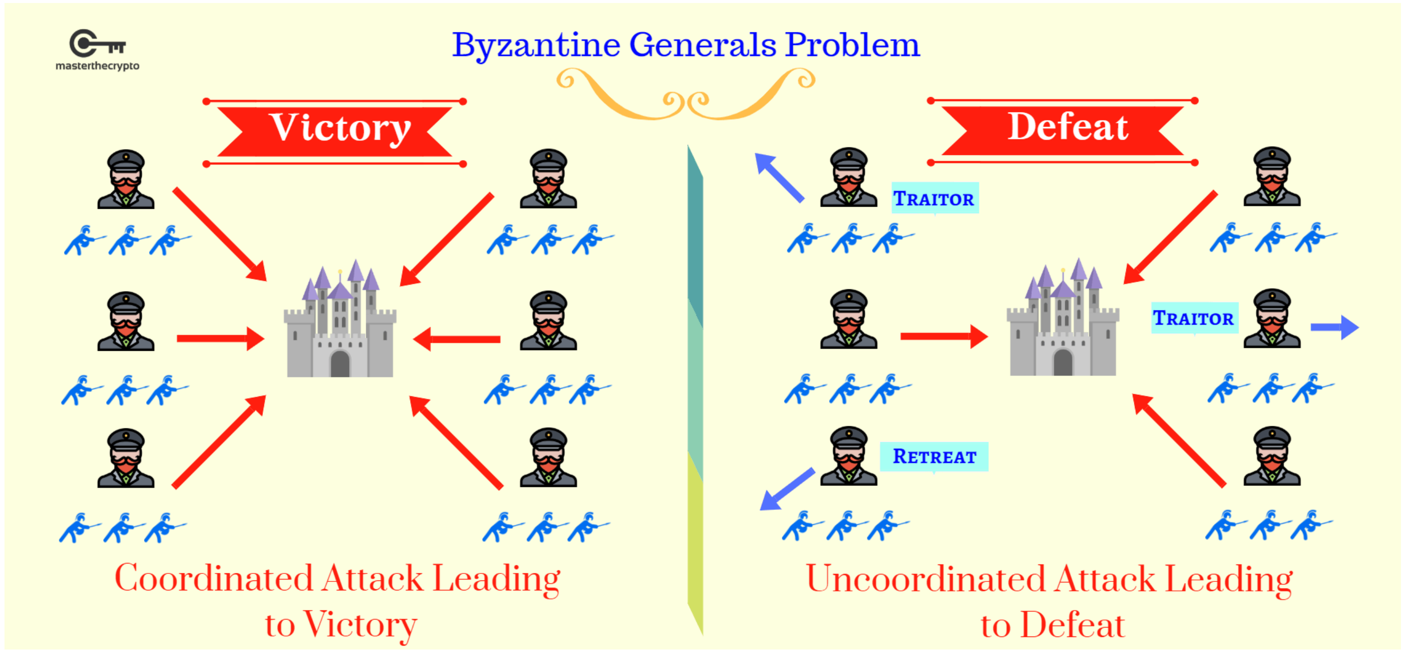
Byzantine consensus with strong consistency

Byzantine Generals Problem (1982)

- "The Byzantine Generals Problem," L. Lamport, R. Shostak, and M. Pease. ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3 (July 1982)
- Classic problem in distributed systems
- Collection of lieutenant generals surrounding Rome
    - Some generals loyal
    - Some generals traitorous
    - Need to coordinate attack or retreat through a series of messages delivered to each other by messengers
    - All loyal generals should perform the desired action
- Problem is easy with the use of public-key cryptography and an authority (commanding general)

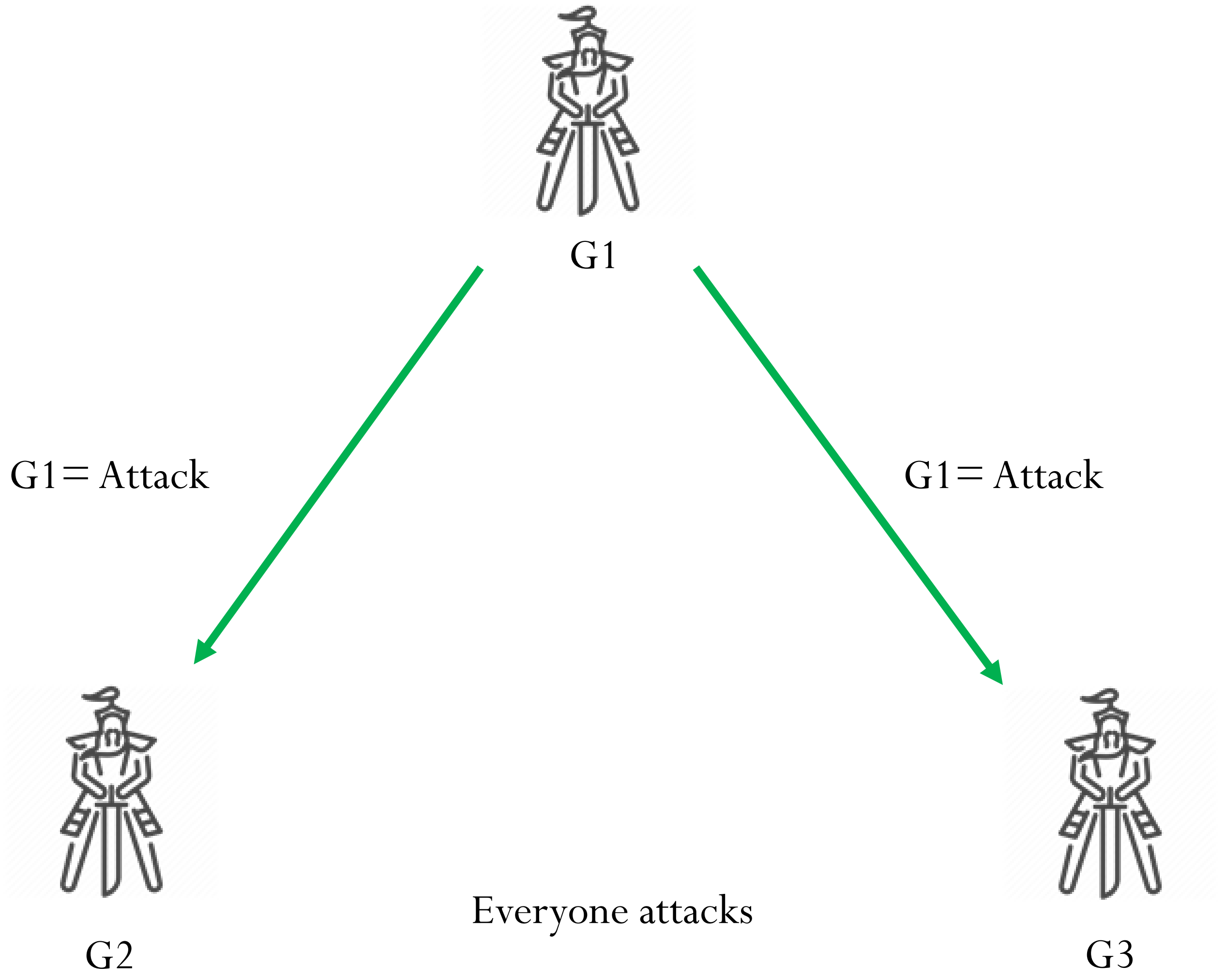
Problem

- More difficult with co-equal generals (e.g. distributed consensus in a public blockchain)
  - Each general has a vote to determine whether to attack or not
  - Majority vote determines action
  - Conditions to meet
    - Every loyal general must obtain the same value for all other loyal generals
    - If a general is loyal, the value she sends must be used by every loyal general


masterthecrypto
Byzantine Generals Problem
Victory
Defeat
Traitor
Traitor
Retreat
Coordinated Attack Leading to Victory
Uncoordinated Attack Leading to Defeat

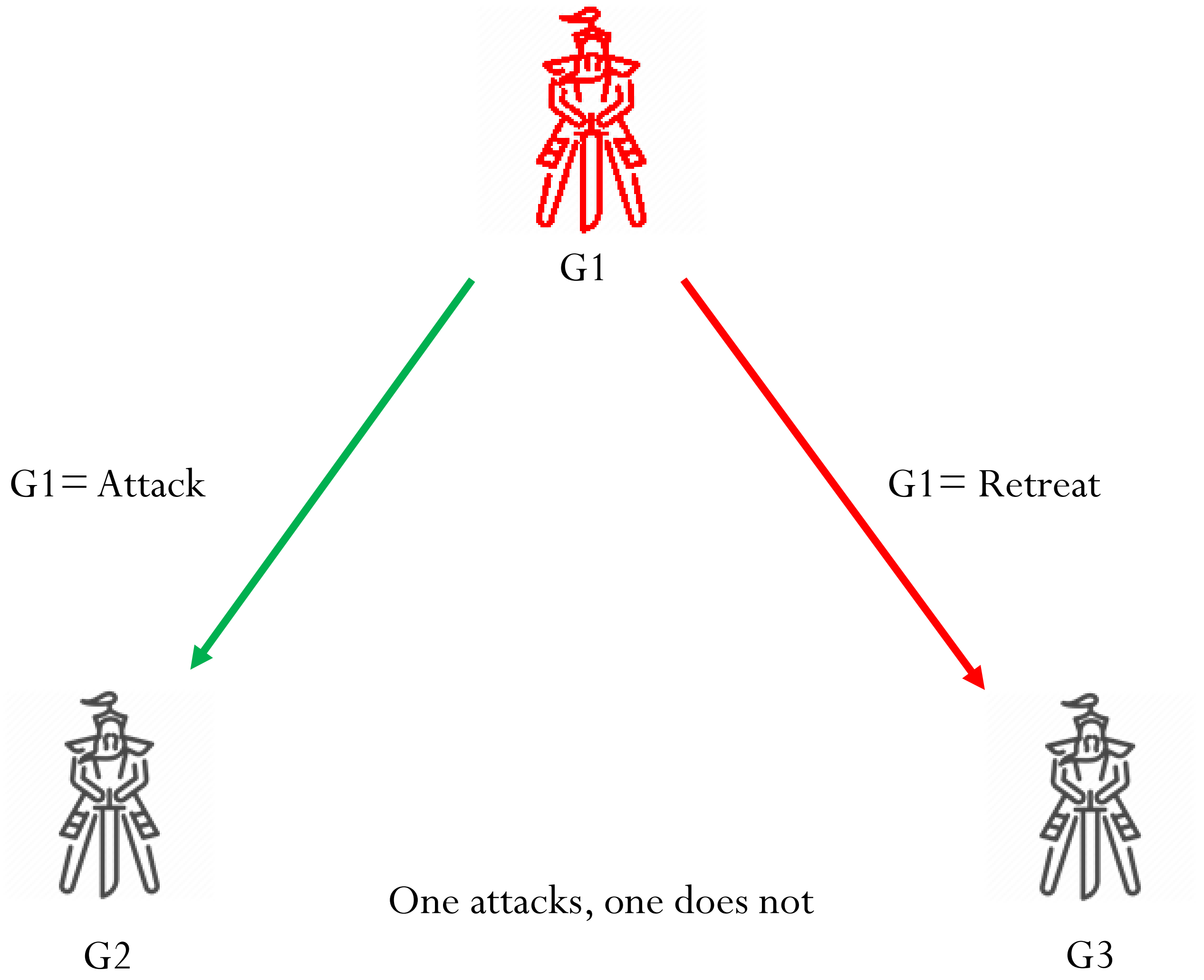
Non-malicious generals

- G1 attack or retreat?
- Trivial agreement when all are loyal



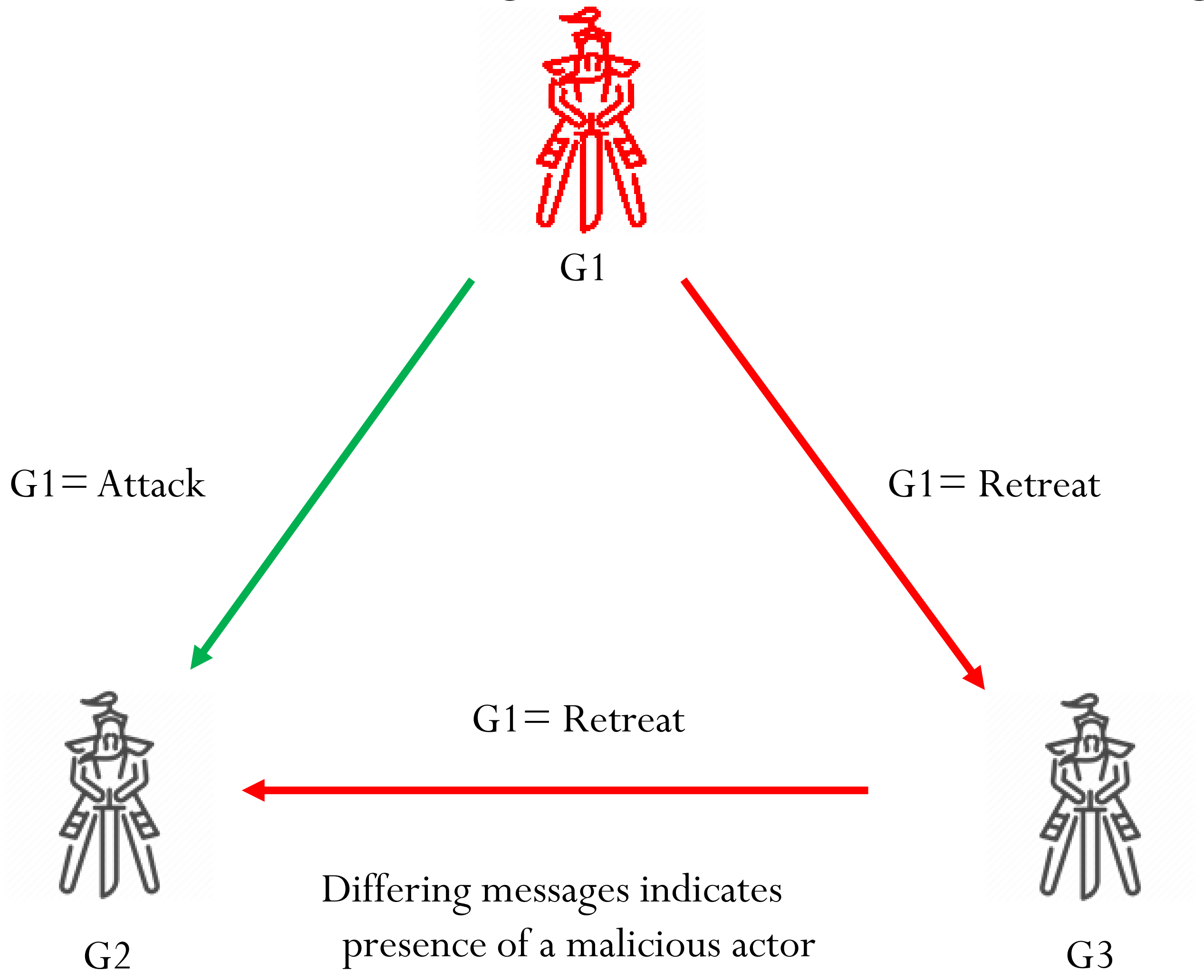
Malicious general

- Can convince disagreement between non-malicious generals



Malicious general

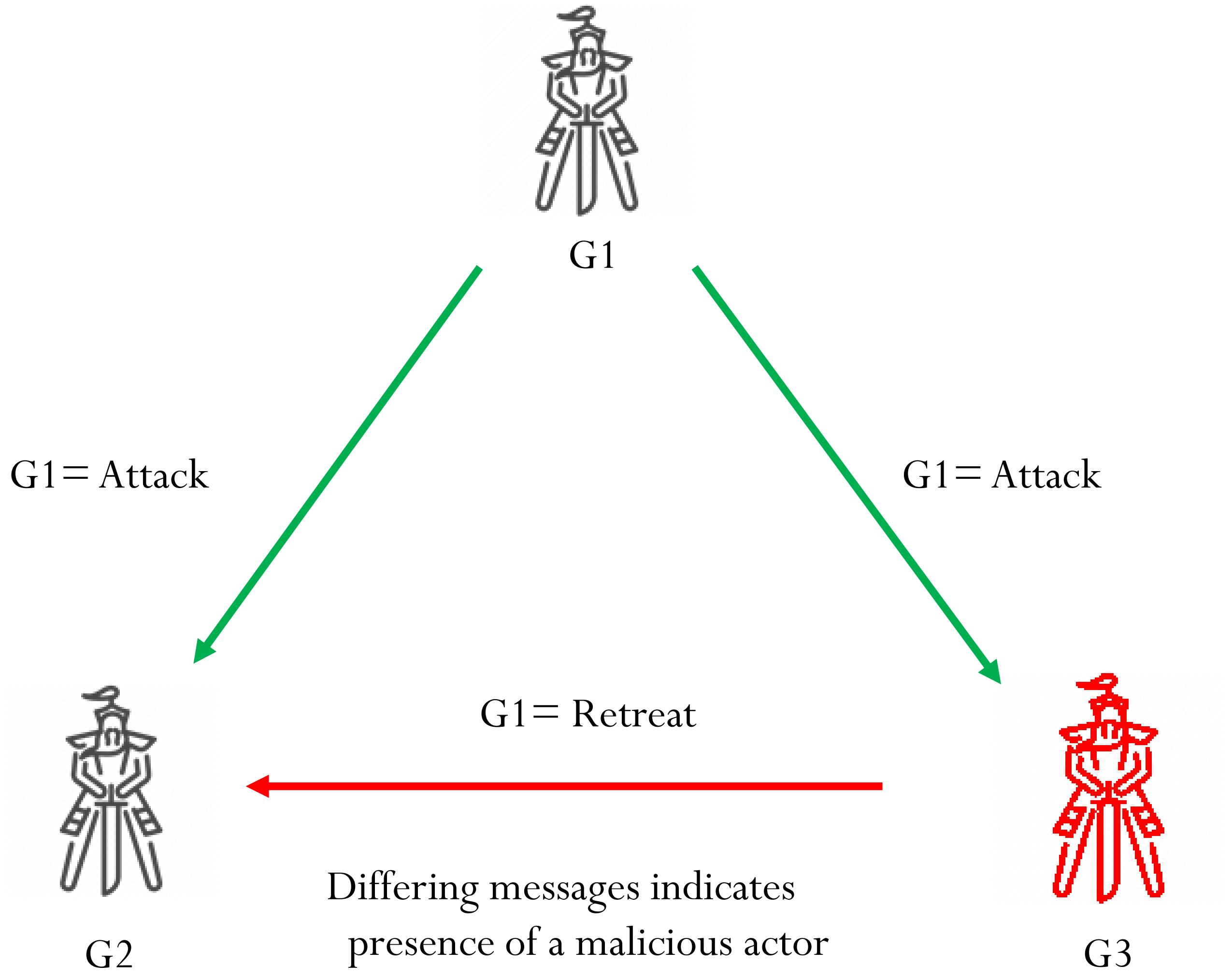
- Extra message exchange can detect presence of malicious actor
- Can G2 determine who is being malicious based on two messages it gets?



Differing messages indicates presence of a malicious actor

Malicious general

- No.
  - Looks the same to G2 as before!

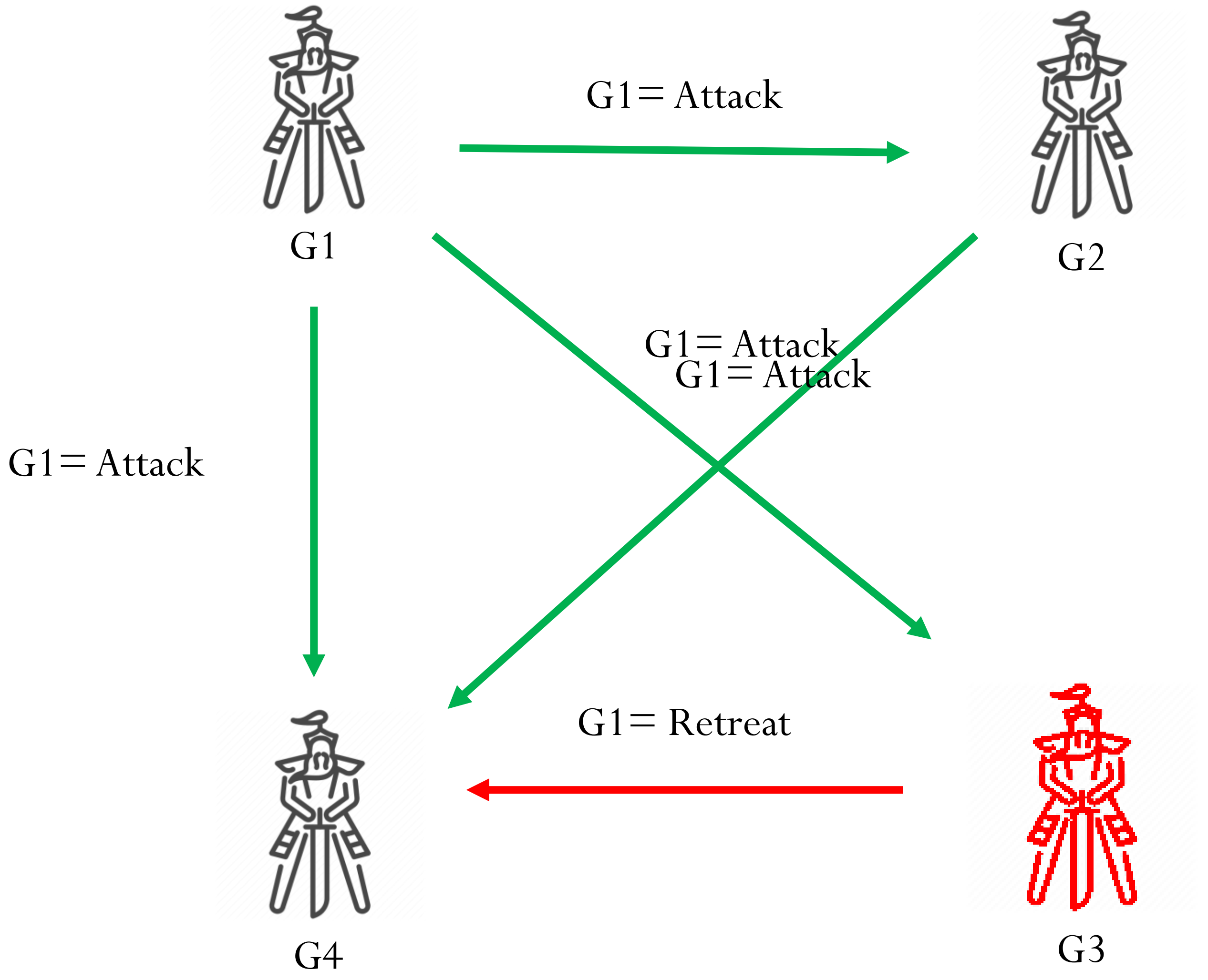


Lamport's result and algorithm

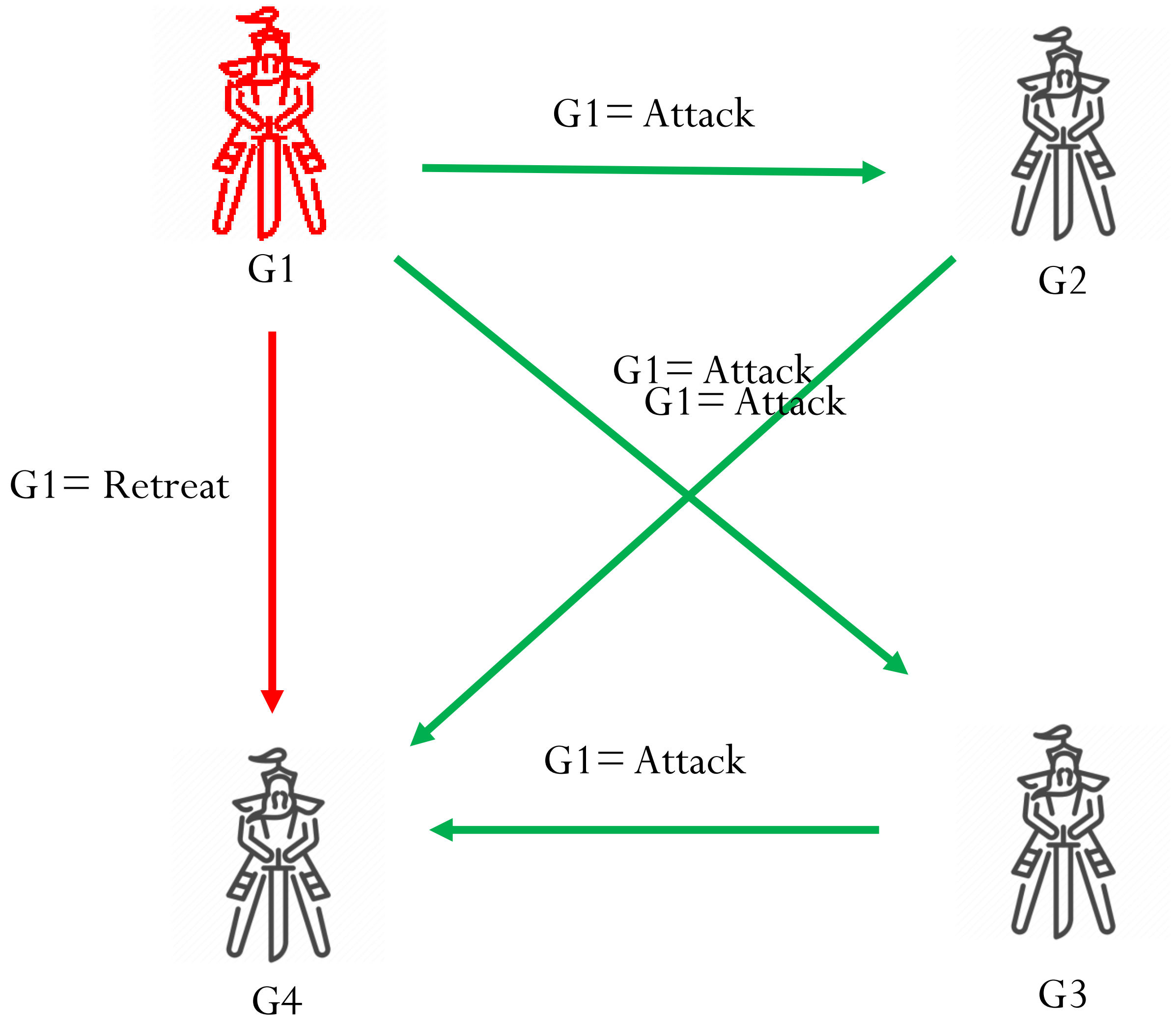
- Assumptions
  - Messages are not signed
  - Communication is synchronous (i.e. with bounded delays)
    - No solution if message delivery times unbounded
- Consensus achieved using a deterministic algorithm in a synchronous network (i.e. bounded delays on messages) iff
  - `n = 3t + 1` participants (`n` = # of generals, `t` = # of traitors)
    - i.e. 2/3 of generals must be non-malicious to solve the problem
  - Consensus in `poly(n)` steps

Intuition

- 4 general solution (G4 determining G1's vote with malicious G3)
  - Majority vote to determine consensus action



- 4 general solution (G4 determining G1's vote with malicious G1)
  - Majority vote to determine consensus action



Achieves strong consistency

- BFT-based protocols always maintain a single, consistent, global state
  - Always correct, no inconsistencies (even temporarily)
- Done as rounds of consensus on state followed by state commitment
  - Often made hierarchical (in order to scale)
- BFT-based approaches being used by many fintech blockchains (IBM, JPMorgan, Tendermint) as a result
  - Consistency paramount!
  - Protocols choose consistency over liveness in presence of partition

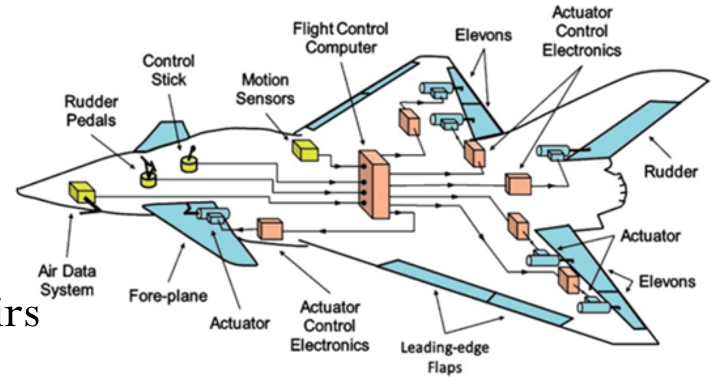
BFT approaches

- Practical Byzantine Fault-Tolerance (PBFT) (1999)
  - OSDI 1999 (Miguel Castro and Barbara Liskov)
  - Works in loosely asynchronous environments (i.e. Internet) via replicated message passing and voting
  - Widely used now (but not until after Bitcoin)
    - ByzCoin (public), Hyperledger Fabric, Hyperledger Iroha (Sumeragi) (private, permissioned)
    - Docker Compose and parts of Kubernetes

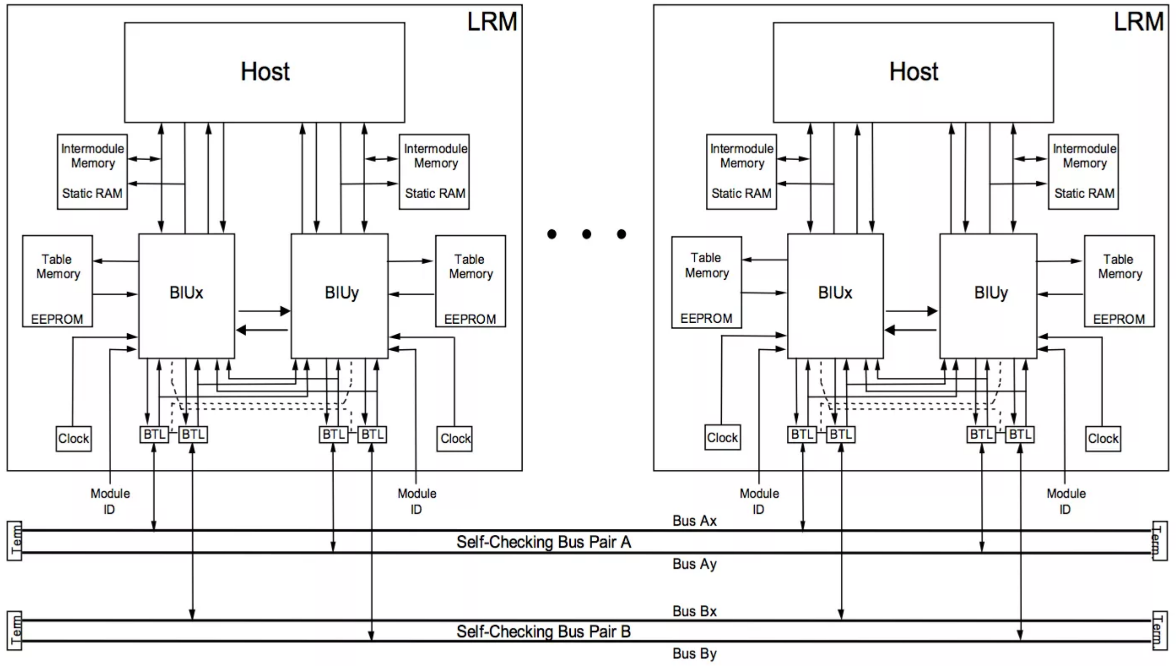
- "Blockchain Consensus Protocols in the Wild", https://arxiv.org/abs/1707.01873v2
  - Tangaroa (Byzantine Fault Tolerant Raft)
    - http://www.scs.stanford.edu/17au-cs244b/labs/projects/wang_tai_an.pdf
    - Adds randomization in election of leadership to address Byzantine actors
  - Redundant Byzantine Fault-Tolerance (RBFT)
  - Tendermint https://tendermint.com/intro/consensus-overview
  - Ripple Protocol consensus algorithm https://ripple.com/consensus-whitepaper/

Byzantine faults in the real-world

- Byzantine Fault Tolerant airplanes
  - Airbus, Boeing implement ARINC 659 SAFEbus network
  - Duplicate transmitters sending through two bus pairs
  - Recipient nodes each receive four copies and only record a message if all 4 are the same
    - Airbus A380 with 13k pounds of wiring for 611,000 pound plane



Basic elements of the FWB control system.



BFT issues

- Sybils
  - Add malicious participants until you subvert system
    - https://cryptoinsider.21mil.com/byzantine-fault-tolerance-blockchain-systems/
    - "Generally, membership in Byzantine agreement systems is set by a central authority or closed negotiation."
  - Makes BFT protocols difficult to use for public blockchains
- Complexity of protocol
  - Implementation
  - Message exchanges
- Deterministic protocol that assumes *synchrony* to achieve consensus

Randomized distributed consensus

- Recall FLP impossibility
  - "Consensus impossible in asynchronous network (i.e. unbounded message delays) with a *deterministic* protocol"
- Solve consensus in an asynchronous network using a non-deterministic protocol with probability close to 1
  - Coin flipping or common coin approach gives consensus with high probability in the presence of failures
  - Observation: Tiny fraction of bad executions in FLP result in loss of liveness
    - Use random coin flips to make them less probable and make eventual progress on consensus
    - Converges to agreement within a fixed, small expected number of rounds
  - Keep in mind for Bitcoin…

Intuition behind randomized algorithms

- 2 pedestrians passing each other in hallway
  - Deterministic algorithm
    - If both on left, go right.
    - If both on right, go left
    - Leads to starvation!
  - Randomized algorithm avoids starvation probabilistically
- Papers from Ben-Or, Rabin, Aspnes, Attiya for theoretical underpinnings applied to Byzantine agreement protocols
  - Linked from course site

DLS consensus (1988)

- Tweak FLP impossibility
  - "Consensus impossible in *asynchronous* network (i.e. unbounded message delays) with a deterministic protocol"
- Cynthia Dwork, Nancy Lynch, Larry Stockmeyer "Consensus in the Presence of Partial Synchrony", JACM v. 35, no. 2, p. 288-323, April 1988.
  - Partial synchrony allows one to make some guarantees on consistency and liveness via a deterministic protocol

- Two cases
  - Upper-bound on delay exists, but not known a priori (protocol takes it into account automatically)
  - Known upper-bound on delay will eventually be achieved at some point in the future to make progress

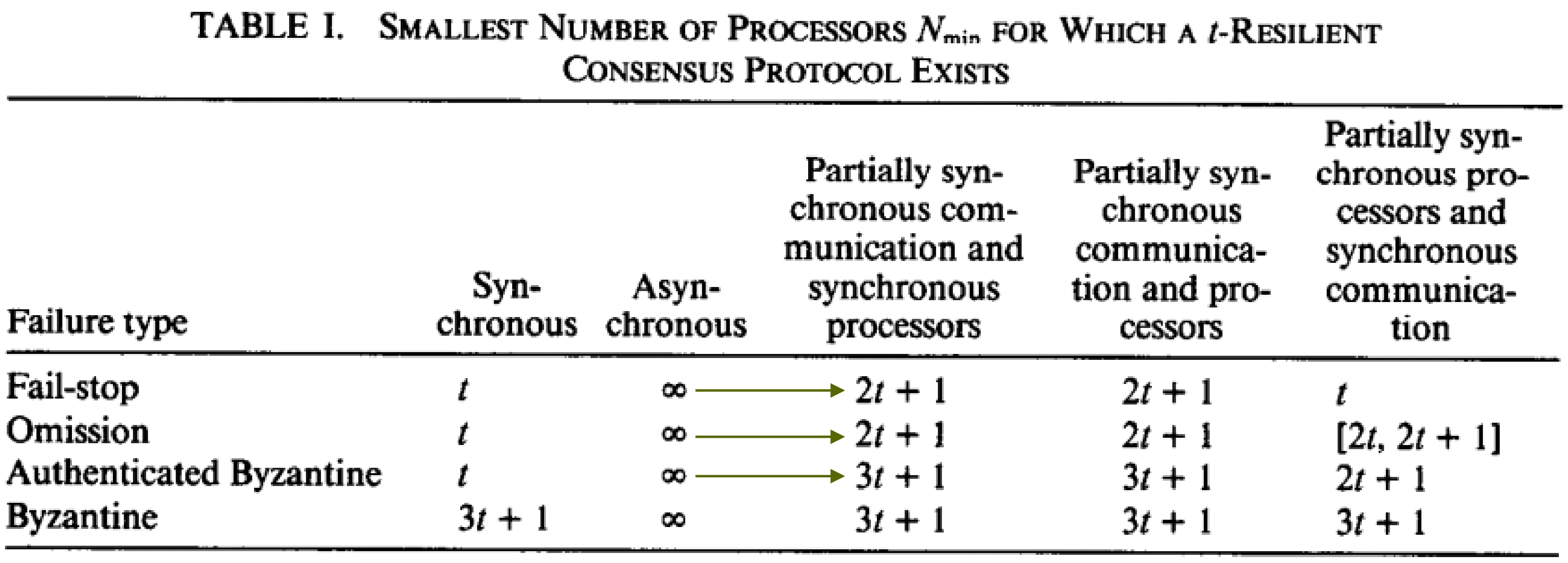


TABLE I. SMALLEST NUMBER OF PROCESSORS N_{min} FOR WHICH A t-RESILIENT CONSENSUS PROTOCOL EXISTS

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

XFT

- Take DLS approach and tweak asynchrony assumption of FLP
- Byzantine fault-tolerant Paxos
  - Provides safety and liveness as long as *a majority of replicas are correct and can communicate with each other synchronously* (a minority of the replicas are Byzantine-faulty, or partitioned due to a network fault)
  - In return it uses only the same number of resources (replicas) as asynchronous crash fault-tolerant systems

Byzantine consensus with weak consistency

Nakamoto consensus

- Tweak FLP impossibility
  - "Consensus impossible in *asynchronous* network with *deterministic* protocol"
- Nakamoto consensus cleverly tweaks both parts to get consensus
  - https://bitcoin.org/bitcoin.pdf
- Make a strong synchrony assumption
  - Replicas with tight relay links to broadcast new blocks in order to maintain consistency
  - Delay in propagating new blocks (10 sec) must be an order of magnitude less than the time between creation of new blocks (10 min)
- Hedge the failure of that assumption using randomized protocol
  - Govern the time for creation of new blocks by randomized proof-of-work hash puzzle (partial hash collision)
  - Difficulty adapts as technology improves

Non-determinism via Proof of work

- Bounded asynchrony
  - Make the time to create next block much larger than the propagation time for broadcasting a newly accepted block
  - Find a solution to a public puzzle that is hard to solve, but easy to verify
  - Get a majority of nodes to accept it as valid
- Typically used for public "trustless" block-chains

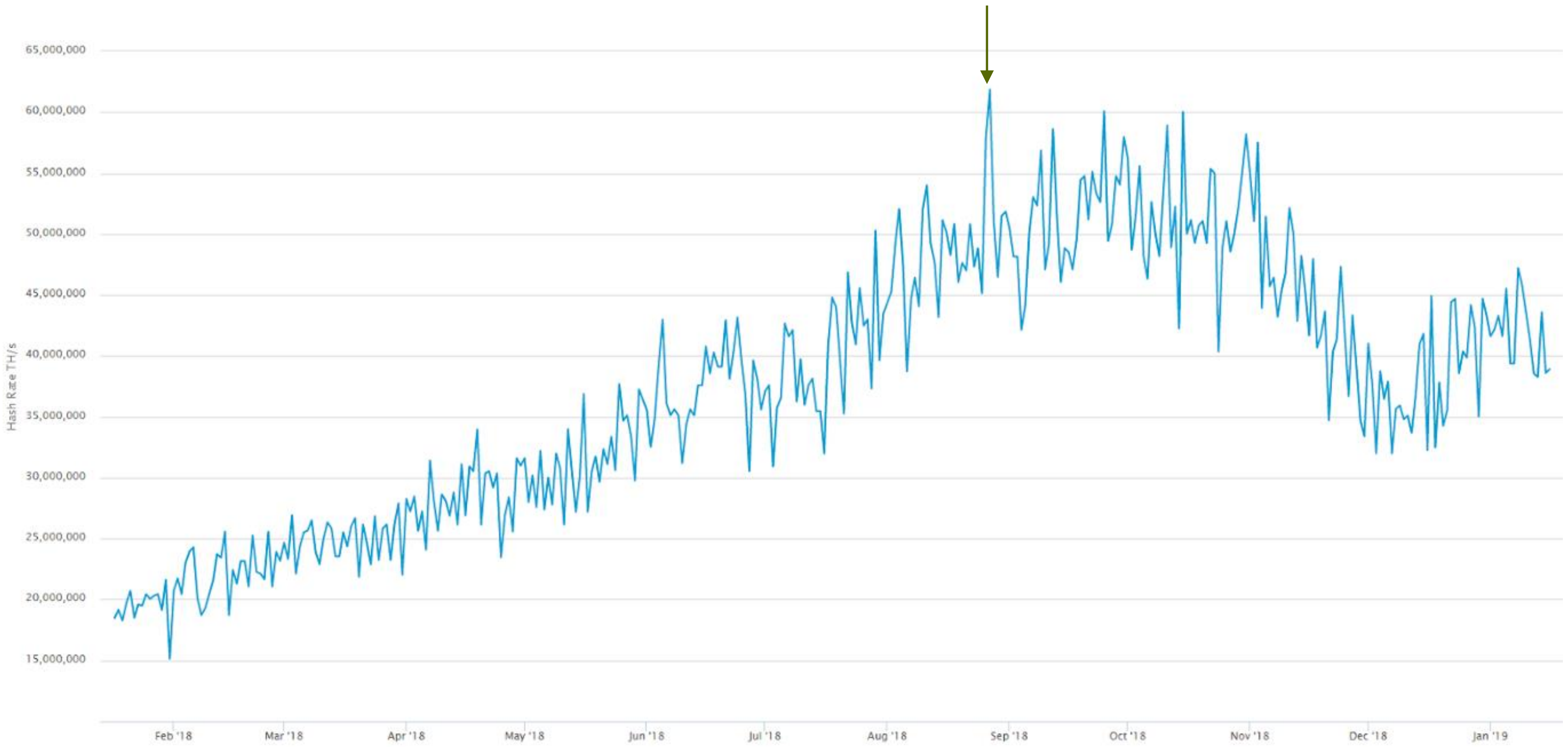
What if?

- Synchrony assumption fails?
  - Individual partitions continue to run!
  - Transactions validated and added to both chains
- How does one reconcile the chain upon reconnection?
  - Longest-chain "wins" and is always the one that miners will choose
  - Invalidates the transactions on shorter chain that were added after partition
  - Randomization governs progress made and which state is accepted

Issue #1: Proof-of-work power consumption

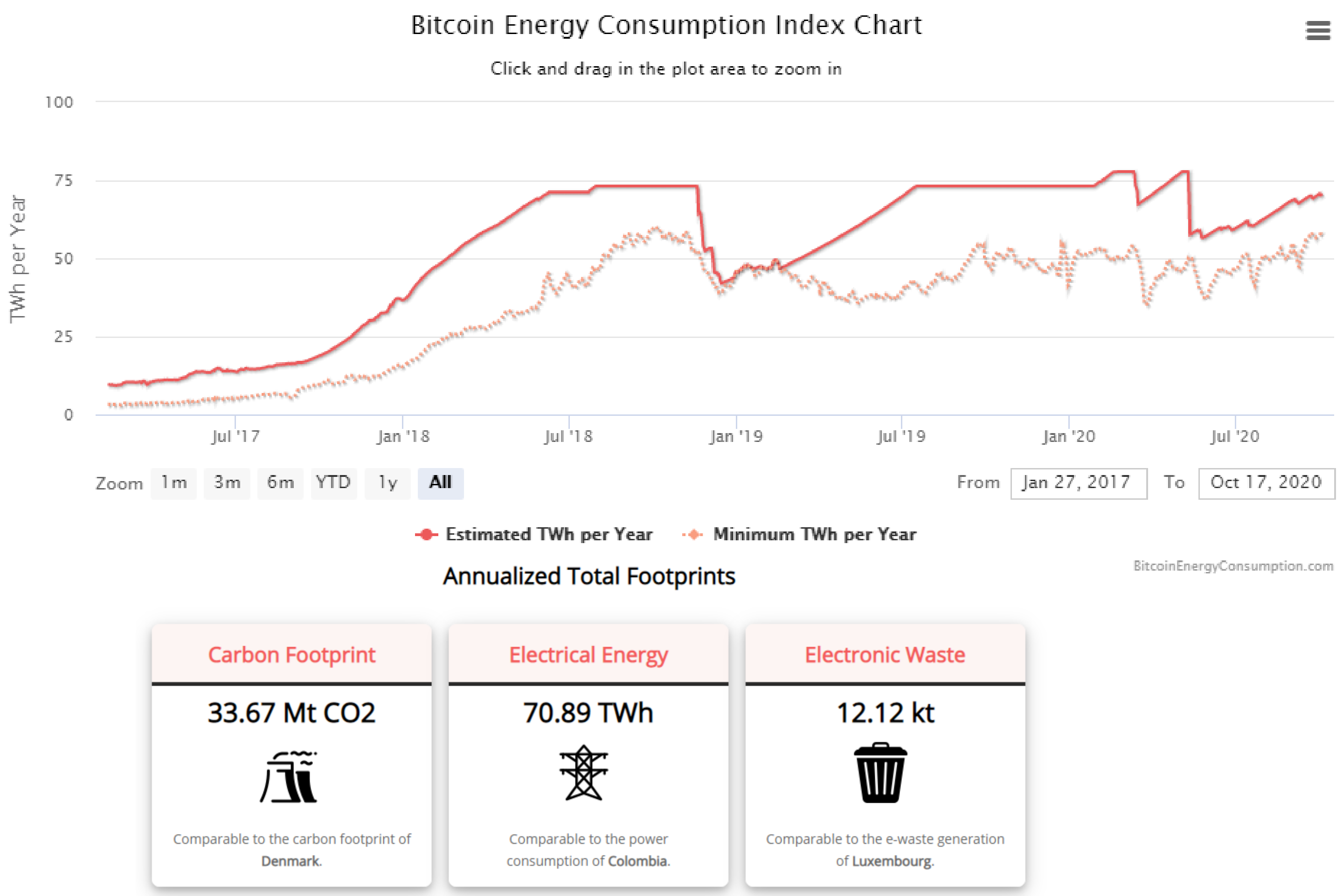
- Number of tera-hashes (trillions) per second being performed on Bitcoin network

$6*10^{19}$ hashes per second!



- Mining operations

- Leads to an environmental disaster…
  https://digiconomist.net/bitcoin-energy-consumption/



From Jan 27, 2017 To Oct 17, 2020

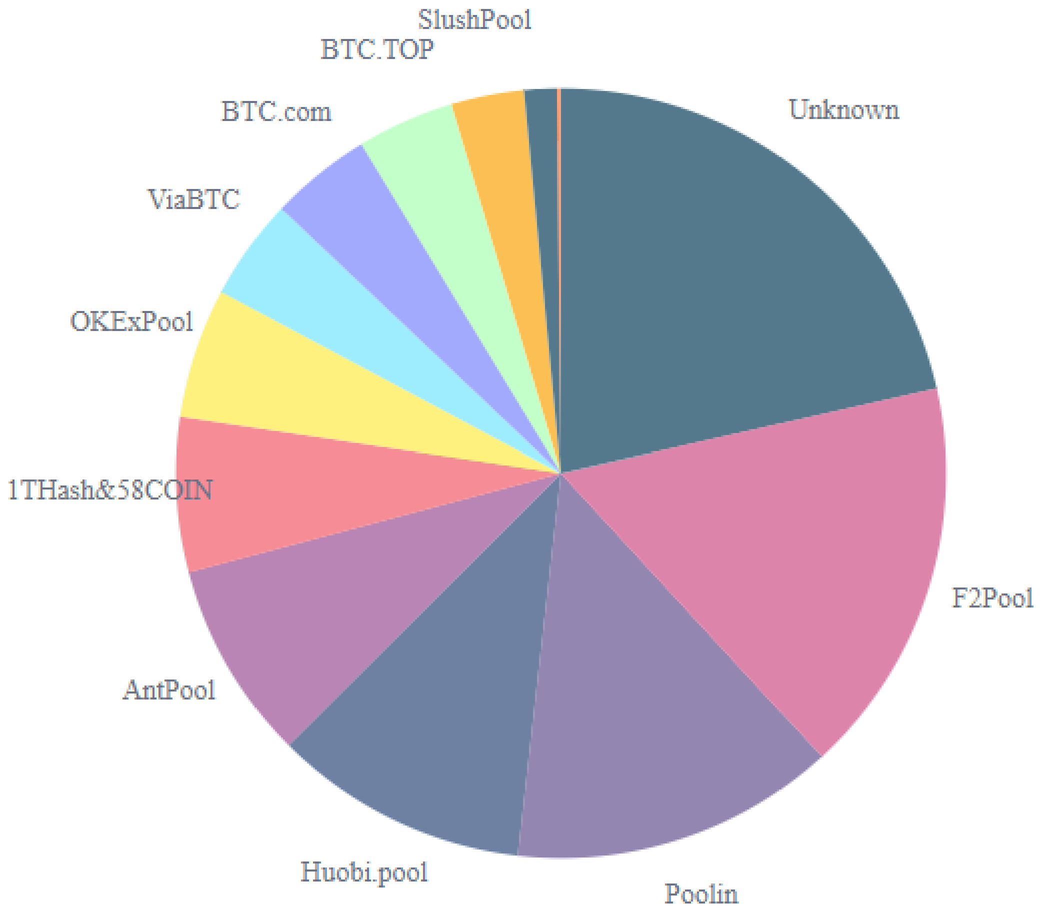
Annualized Total Footprints

Carbon Footprint	Electrical Energy	Electronic Waste
33.67 Mt CO2	70.89 TWh	12.12 kt
Comparable to the carbon footprint of **Denmark**.	Comparable to the power consumption of **Colombia**.	Comparable to the e-waste generation of **Luxembourg**.

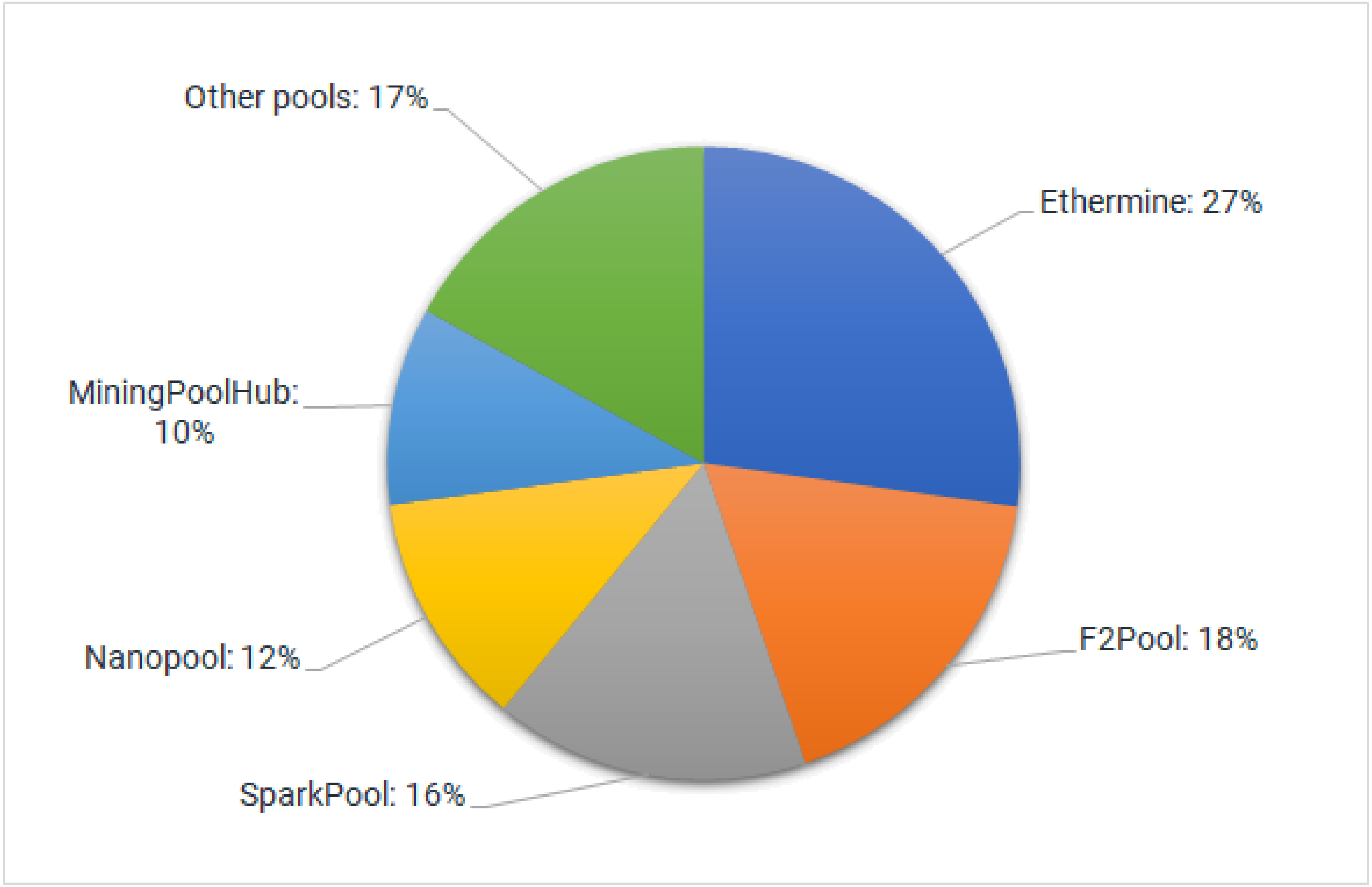
- An environmental disaster…
- Alternatives?

Issue #2: Hashrate distribution

- Centralization based on hardware resources for solving puzzle
  - Bitcoin mining pools https://www.blockchain.com/pools



- Ethereum mining pools
  - https://miningpoolstats.stream/ethereum

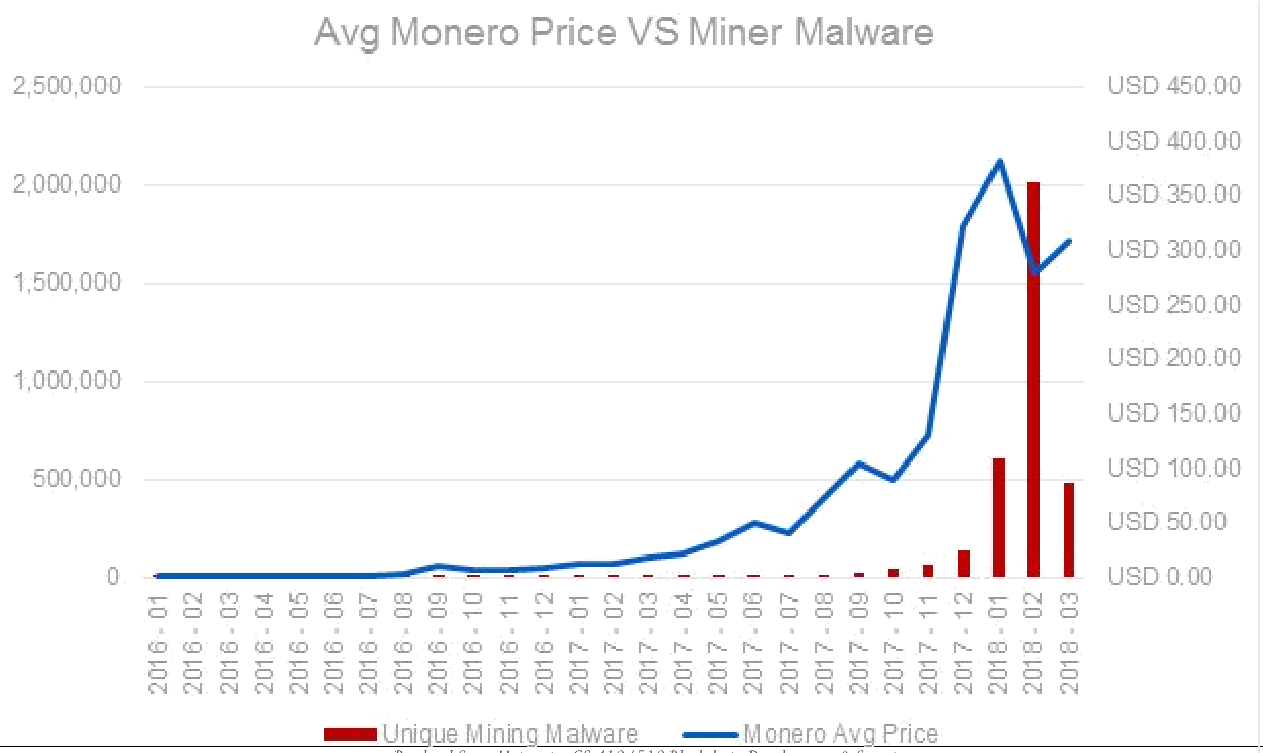


- Monero
  - Proof-of-work algorithm that resists parallelization
  - Allows any computer to potentially find a solution
  - Cryptojacking on the rise…

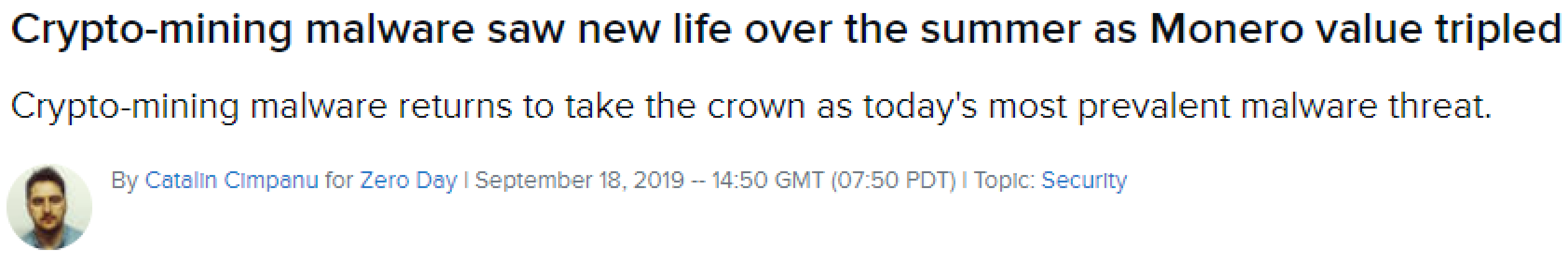
Over 4 percent of all Monero was mined by malware botnets

Academics say malware authors might have cashed out at least $57 million worth of Monero over the course of the last four years.

By Catalin Cimpanu for Zero Day | January 21, 2019 -- 21:47 GMT (13:47 PST) | Topic: Security



Cryptojacking replacing ransomware for prevalence



Crypto-mining malware saw new life over the summer as Monero value tripled

Crypto-mining malware returns to take the crown as today's most prevalent malware threat.

By Catalin Cimpanu for Zero Day | September 18, 2019 -- 14:50 GMT (07:50 PDT) | Topic: Security

- *(When cryptocurrency prices skyrocketed to record levels), what was once an outlier in the malware scene had suddenly become the most common form of malware.*

Proof of stake

- Address energy consumption to replace miners replaced by validators in consensus protocol
  - Moves to a BFT-style protocol
  - Uses "stake" for admission into the validation protocol
- Mechanism
  - Validators put up collateral (reputation or currency)
  - Validators randomly elected via probabilistic election based on amount of stake held to insert next block
  - Validators validate and insert block to obtain a reward
  - Collateral taken by network if anything incorrectly validated
- Must hold large percent of ETH to attack consensus
- First used in Peercoin (King 2012), b-money (Dai 1998)

Example: Ethereum 2.0 (Beacon chain)

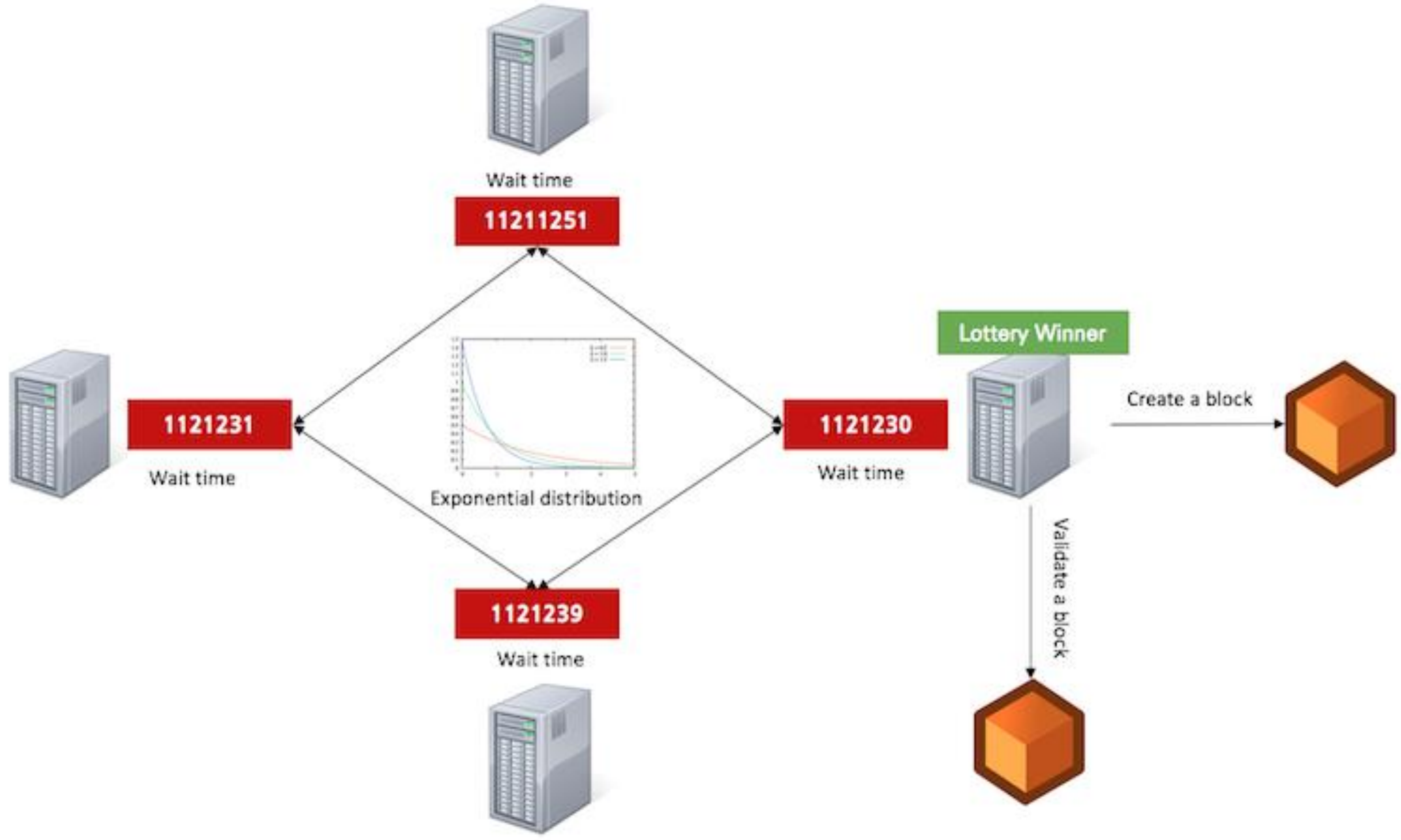
- Validators place 32 ETH into a ETH Foundation deposit contract
  - Validation begins when 524,288 ETH across 16,384 validators staked
  - Each proposed block has random committees of 128 validators attest to it
  - Require 2/3 agreement of validators to attest
  - Probability that an attacker can own enough nodes to get elected at the 2/3 level extremely low!
  - https://consensys.net/blog/blockchain-explained/what-is-proof-of-stake/
- Issue
  - Eventual plutocratic governance by insiders?

Variants

- Delegated Proof-of-Stake (e.g. Congress)
  - Real-time voting with a social system for reputation built-in
  - Users vote for delegates (witnesses)
  - Voting power increases proportionally to amount of tokens held
  - Simply vote out a dishonest delegate
- Leased Proof-of-Stake (e.g. Mutual Funds)
  - Lease coins to a node you trust without giving up ownership
  - Financial reward from node validating blocks split with users whom the node leased from

Proof of elapsed time

- aka PoET
- For permissioned block-chains where nodes pre-selected by enterprise
  - Trusted hardware elects machine that will validate next block to insert
- Algorithm
  - Each node assumed to contain a trusted execution environment (e.g. Intel SGX)
  - Trusted hardware generates a random elapsed time
  - Node with the lowest time waits that amount, then adds next block
  - Delay allows previous block to propagate
  - Puts trust in Intel SGX
    - Perhaps fine since control already centralized in permissioned model
  - Examples: Hyperledger Sawtooth
    - https://intelledger.github.io/introduction.html#proof-of-elapsed-time-poet


Wait time
11211251
1121231
Wait time
Exponential distribution
Lottery Winner
1121230
Wait time
Create a block
Validate a block
1121239
Wait time

Proof of capacity

- Probabilistically choose node to add next block based on hard drive space
  - Nodes store large data sets
  - Those with more given higher probability
  - e.g. proof of storage, proof of space
- Examples
  - Burstcoin https://www.burst-coin.org/proof-of-capacity

Proof of Burn

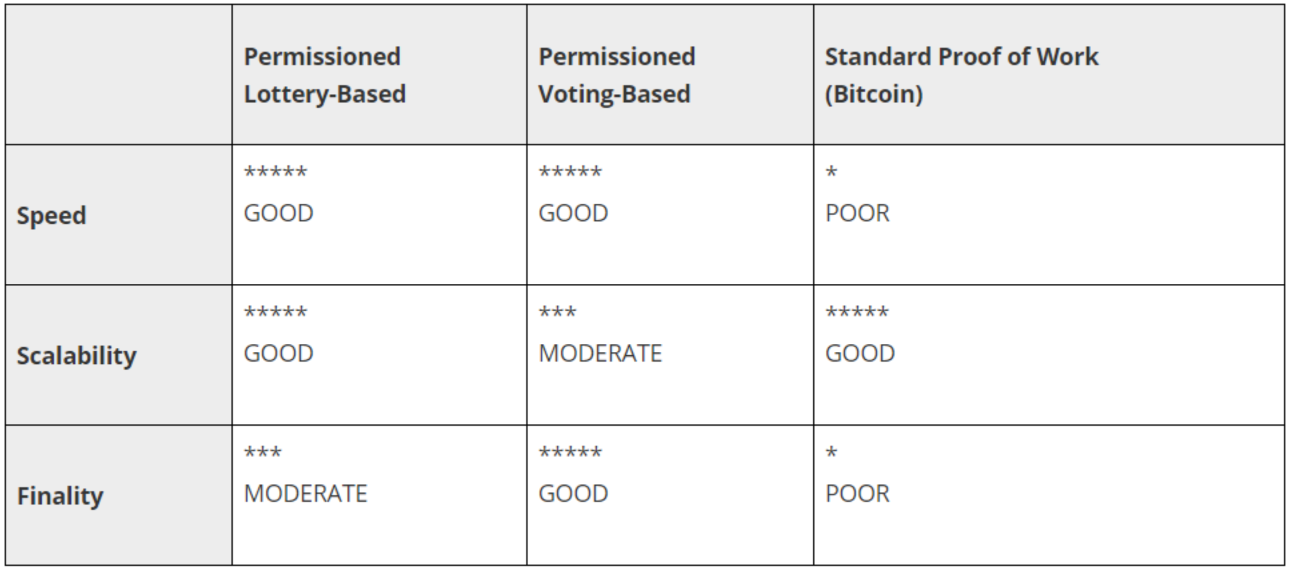
- Miners with prior expenditures to unspendable addresses are randomly selected to insert next block
  - Amount burned < Block validation commission
- Used in Slim Coin http://slimco.in/proof-of-burn-guide/
- https://blockonomi.com/proof-of-burn/

Consensus protocols summary

- Consensus protocol you use typically driven by application
  - Can run a custom one on each side-chain that then syncs up to main network
- Synchronous vs. asynchronous
  - Rounds between participants before progressing vs. no coordination
- Deterministic vs. probabilistic
- Crash-fault tolerance vs. Byzantine fault-tolerance
  - Crash (fail-stop): handle node failures via majority voting
  - Byzantine: handle actively malicous 'live' nodes tampering with voting

Which to use for your blockchain?

- Voting-based algorithms (vulnerable to sybil attacks, but no forking issue)
  - Good for permissioned blockchains requiring strict consistency (fintech)
- Lottery-based algorithms (resistant to sybil attacks, but have forking issue)
  - Good for public blockchains



	Permissioned Lottery-Based	Permissioned Voting-Based	Standard Proof of Work (Bitcoin)
Speed	***** GOOD	***** GOOD	* POOR
Scalability	***** GOOD	*** MODERATE	***** GOOD
Finality	*** MODERATE	***** GOOD	* POOR