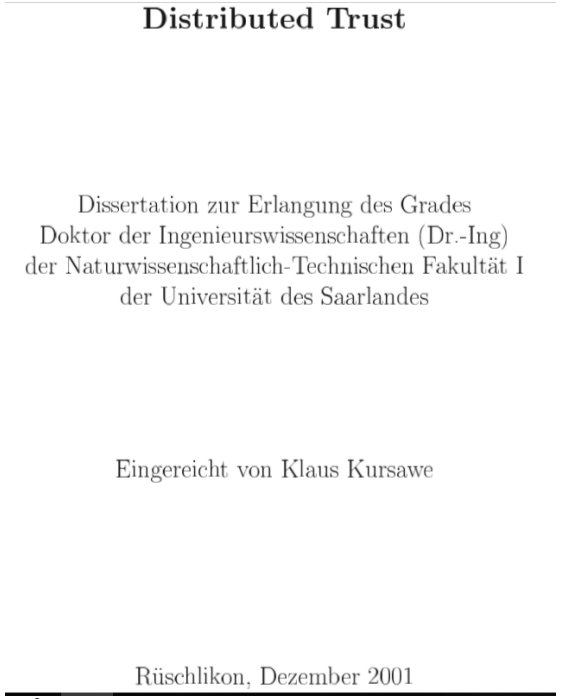
What the Flock: Blockchains for the Derivative Market

Klaus@vega.xyz

About Me



Distributed Trust

Dissertation zur Erlangung des Grades
Doktor der Ingenieurswissenschaften (Dr.-Ing)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Eingereicht von Klaus Kursawe

Rüschlikon, Dezember 2001

By History:

- PhD on Byzantine Fault Tolerant Ordering protocols in 2001
  - First fully asynchronous, leaderless, practical BFT protocol (with implementation & formal verification)

As noone cared about this back then:

- Other stuff in Security and Privacy

Now:

- Talking people out of using blockchains where it doesn't make sense
- Former advisor to ChainSpace.io, Libra
- Now: VEGA

Privacy-Preserving Matching of DNA Profiles
Fons Bruekers and Stefan Katzenbeisser and Klaus Kursawe and Pim Tuyls
2002/134 (PS PS.GZ PDF)

Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems
Christian Cachin and Klaus Kursawe and Anna Lysyanskaya and Reto Strobl
2001/022 (PS PS.GZ PDF)

Optimistic Asynchronous Atomic Broadcast
Klaus Kursawe and Victor Shoup
2001/006 (PS PS.GZ PDF)

Secure and Efficient Asynchronous Broadcast Protocols
Christian Cachin and Klaus Kursawe and Frank Petzold and Victor Shoup
2000/034 (PS PS.GZ PDF)

Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement using Cryptography
Christian Cachin and Klaus Kursawe and Victor Shoup

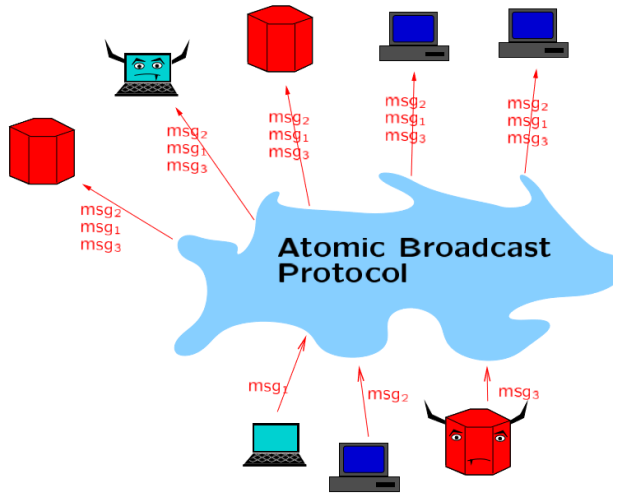
[Cryptology ePrint archive]

Theoretical Background, or why it’s hard what we’re doing

What a blockchain is (scientific view)

Byzantine Fault tolerant total ordering protocol

Also atomic broadcast, virtual synchrony, …

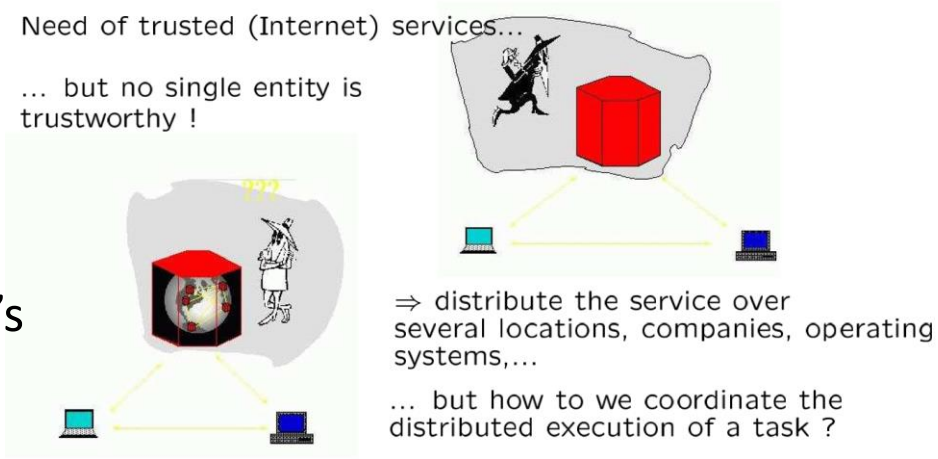


- agree on an ordered set of transactions

  (i.e, what happened and in which order)
- even if some participants are byzantine (corrupted)

Good for

- Digital Currencies
- Distributed State machines
- Can also be used as a distributed database, but that's overkill



The core of it all: Byzantine Agreement

Pronunciation:/ˈbizənˌtēn, bəˈzan-, -ˌtīn/

(also **byzantine**) (of a system or situation) excessively complicated, typically involving a great deal of administrative detail: "*Byzantine insurance regulations*" characterized by deviousness or underhanded procedure:*Byzantine intrigues"he has the most Byzantine mind in politics"*

Term coined by Leslie Lamport (1982), A. C. Koestler (1937), F.M.A. Voltaire (ca. 1750), Greek Mythology

Given n processors $P_1, P_2 \ldots P_n$, which are connected by an asynchronous point-to-point network. All parties have a (binary) input value v_i. A protocol solves Byzantine Agreement if the following conditions are satisfied:

Validity

If all honest processes start with input value x, then the decision value must be x.

Agreement

If one honest party decides 0, then no honest party decides 1.

Termination

All honest parties eventually decide.

Constantinople (Istanbul), circa 1893

Annoying Impossibility results (1)

Life ain’t fair.

For small scale systems, this can be resolved(ish) using threshold encryption. Unless you worry about metadata or scale.

“If all honest parties see event A happen before event B , then A should be scheduled before B.”

This is impossible in the presence of a single a corrupted participant

Annoying Impossibility results (2)

Evil prevails even with inferior numbers.

In an asynchronous system, agreement is only possible if less than 1/3 of all participants are corrupt.

"When I introduced Byzantine failures, it was meant to model arbitrary but independent failures, not coordinated malicious ones. The assumption that a dedicated attacker is bound by attacking only one third of all parties is ridiculous."

Leslie Lamport, 2001

"Do you have any better argument why not more than 1/3 of the servers get corrupted than your inability to do better ?"

My bosses boss, 2000

Annoying Impossibility results (2)

It is an illusion that the scale of major PoW blockchains overcomes the thresholds automatically.

Malicious attacks are not independent.
How many miners have different

- Operating Systems
- Implementations
- Libraries
- Legal authorities
- Influenceability by countries controlling the lead currency
- … ?

This was addressed 1978 with resynchronization and 1997 with General Adversary Structures.

Ethereum:

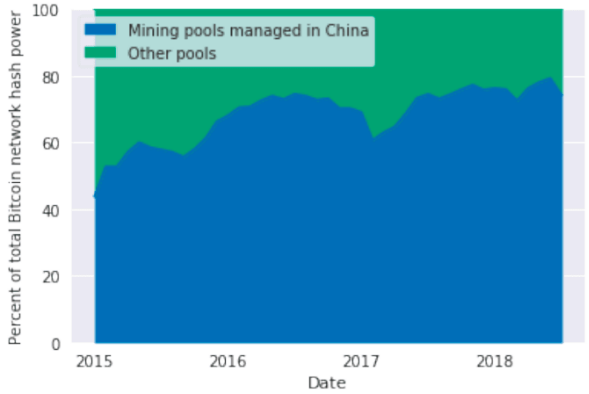
One third of all assets (for PoS) is about 100 wallets if everybody else participates.

Assuming there are a lot of passive assets, it is even worse

Bitcoin:

Due to mining pools and rigs, this assumption is long gone.

It doesn't even need malice to cause issues.



Mining pool	Located in China	Estimated share of network hash rate
F2Pool	Yes	22.17
AntPool	Yes	21.54
BTCC	Yes	12.79
BitFury	No	12.39
BW Pool	Yes	7.84
KnCMiner	No	4.89
SlushPool	No	4.72
21 Inc.	No	2.27

https://arxiv.org/pdf/1810.02466.pdf

Annoying impossibility results (3)

Things don’t scale.

The communication complexity of voting based protocols is quadratic(ish) in the number of participants

Reality check: There never where millions of participants, anyhow

- The number of parties that run the show is more 100-1000
- That was worrisome on the previous slide, but here it’s helpful

Hybrid models at least can handle absentee participants. Re-Synchronisation protocols can be a base for sharding.

Annoying impossibility results (4)

We can’t agree.

In an asynchronous system, there is always an action the adversary can perform to prevent termination with an agreement, even if she is only allowed to crash one party (FLP85).

So we gave the authors an award and have been trying to cheat around it ever since

Impossibility of Distributed Consensus with One Faulty Process

MICHAEL J. FISCHER

Yale University, New Haven, Connecticut

NANCY A. LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

MICHAEL S. PATERSON

University of Warwick, Coventry, England

Abstract. The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of nontermination, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the “Byzantine Generals” problem.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols–*protocol architecture*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems–*distributed applications; distributed databases; network operating systems*; C.4 [**Performance of Systems**]: Reliability, Availability, and Serviceability; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation–*parallelism*; H.2.4 [**Database Management**]: Systems–*distributed systems*; *transaction processing*

General Terms: Algorithms, Reliability, Theory

Additional Key Words and Phrases: Agreement problem, asynchronous system, Byzantine Generals problem, commit problem, consensus problem, distributed computing, fault tolerance, impossibility proof, reliability

1. *Introduction*

The problem of reaching agreement among remote processes is one of the most fundamental problems in distributed computing and is at the core of many

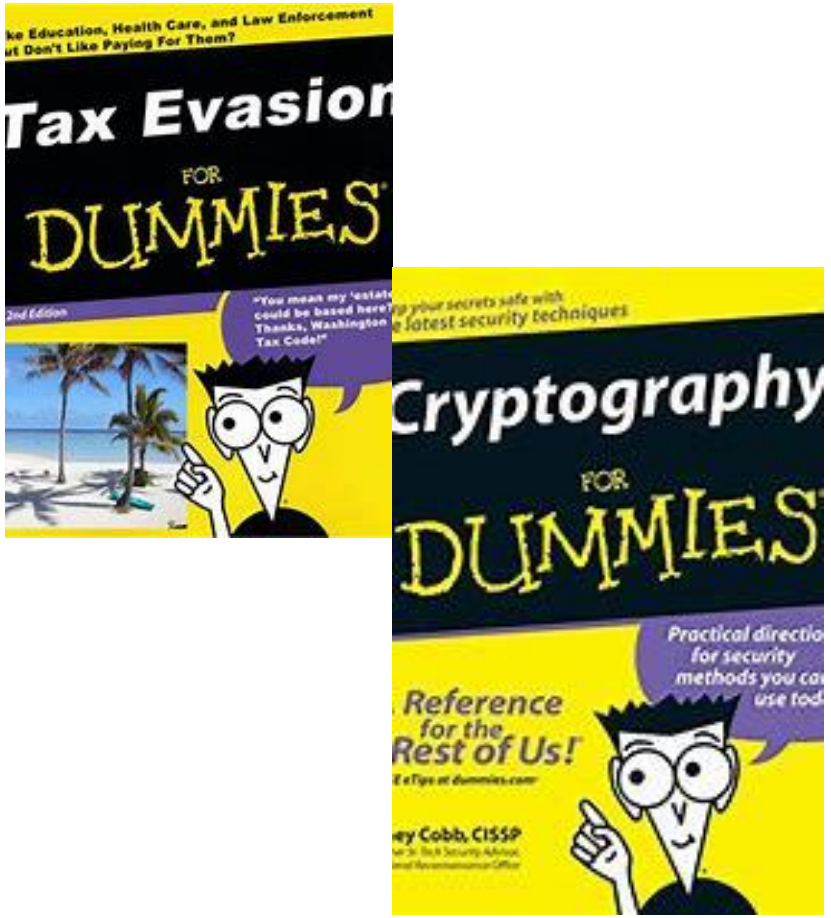
Editing of this paper was performed by guest editor S. L. Graham. The Editor-in-Chief of JACM did not participate in the processing of the paper.

This work was supported in part by the Office of Naval Research under Contract N00014-82-K-0154, by the Office of Army Research under Contract DAAG29-79-C-0155, and by the National Science Foundation under Grants MCS-7924370 and MCS-8116678.

This work was originally presented at the 2nd ACM Symposium on Principles of Database Systems, March 1983.

Authors' present addresses: M. J. Fischer, Department of Computer Science, Yale University, P.O. Box 2158, Yale Station, New Haven, CT 06520; N. A. Lynch, Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139; M. S. Paterson, Department of Computer Science, University of Warwick, Coventry CV4 7AL, England

Cryptograph is like taxes: It is possible to cheat, but you'd better really know what you're doing.* **



* Or at least hire someone who does.

** Also, you'd better have a very good memory.

Cheating the Byzantine Agreement impossibilities

Proof of Work: Terminating is overrated, anyhow

- If I never definitively decide, the proof doesn't apply

Voting based (BFT): We never knew how to model timing assumptions on the Internet, but we do it anyhow

- No adversary should have that much network control. Noone managed to find a good model for what they can do though (and many tried)
  - DoS, router hacking, Nation states rerouting half the internet, …,…
  - The more careful the assumption, the bigger the performance loss

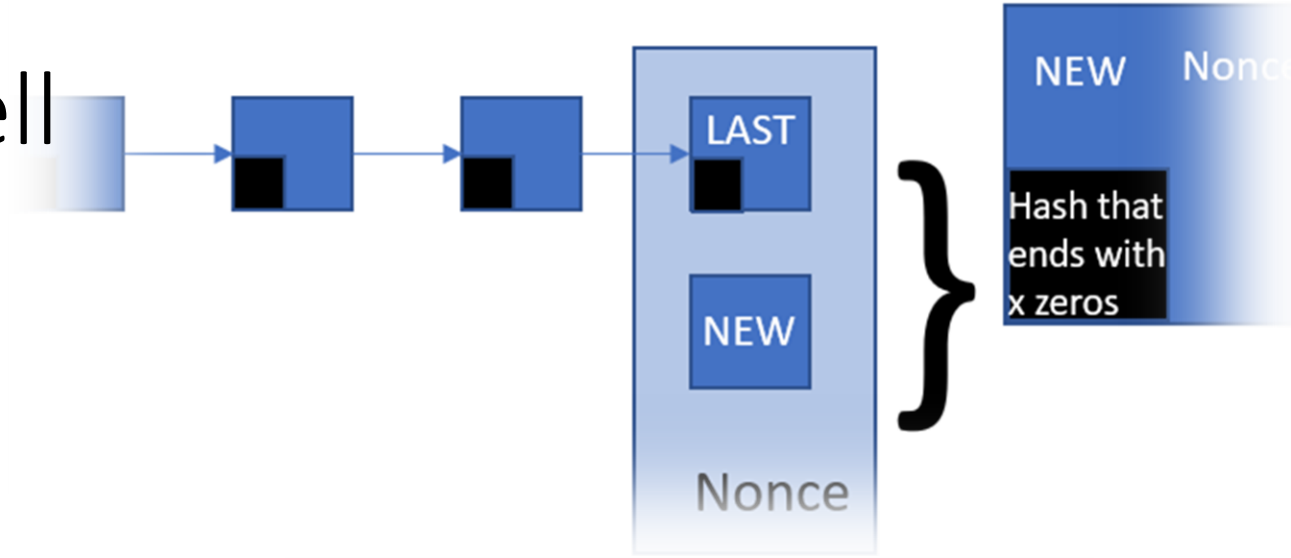
Voting based (Oh come on): We don't know how bad the adversary is, but surely not worse case bad.

- The attacker still can always prevent termination. However, this requires a level of network control that's just not realistic.

Voting based (Randomisation): Terminate with probability 1.

- The attacker still can always prevent termination. What she needs to do to achieve that is determined at random after her action.

Proof of Work in a nutshell



- Take a new, valid, block of transactions and the end of the longest chain you know of
- Find a nonce so that the hash of these three elements ends with a lot of zeros
- Publish the new block
- **If, at any point, you see a longer chain, drop everything and use that one**

Cheating like Bitcoin does (PoW)

No transaction is ever final

- In theory, every transaction can be undone is someone finds a longer chain
- Decisions are final by policy, not protocol

As no one actually *decides* in a formal way, the impossibility proofs don't hold.

Price to pay:

- Performance
- Occasionally, one underestimates (Ether Classic)
- Code is not law; policy decides when a transaction is valid
- If the network partitions, re-synchronization uses the 'Sucks-to-be-you' principle
- Censorship/race attacks are possible
- Fun legal question: Where is the ₿₿₿ during validation ?
  - Can I make all my Bitcoin disappear during the tax deadline ?

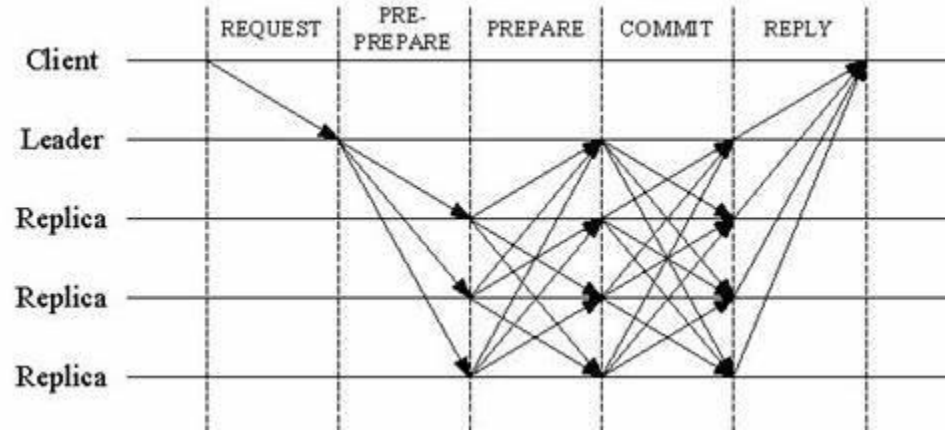
Cryptocurrency	Confirmations Required	Estimated Time*
Augur (REP)	30 Confirmations	6 Minutes
Bitcoin (XBT)	6 Confirmations	60 Minutes
Bitcoin Cash (BCH)	15 Confirmations	2.5 Hours (150 Minutes)
Bitcoin SV (BSV)	30 Confirmations	5 Hours (300 Minutes)
Cardano (ADA)	15 confirmations	10 Minutes
Dash (DASH)	6 Confirmations	15 Minutes
Dogecoin (XDG)	20 Confirmations	20 Minutes
Eos (EOS)	N/A	Near-instant
Ether (ETH)	30 Confirmations	6 Minutes
Ether Classic (ETC)	120 Confirmations	Disabled
Gnosis (GNO)	30 Confirmations	6 Minutes
Litecoin (LTC)	12 Confirmations	30 Minutes
Watermelon (MLN)	30 Confirmations	6 Minutes
Monero (XMR)	15 Confirmations	30 Minutes

Source: www.kraken.com

Cheating like BFT does (Proof of Stake)

“If we keep doubling the timeout, eventually we hit something realistic”

Even if we’re wrong, the worst thing that can happen is lack of progress.



Optimistic phase of PBFT. A leader orders client requests; the rest of the protocol prevents it from creating an inconsistent order. If it tries, or omits requests, the replicas complain and chose a new leader. This is where it gets complicated.

Price to pay:

Easy to derail the protocol with a very limited DoS

Easy to massively violate fairness (Use a DoS to kick out every leader until it’s me)

This even works without participating (Use a DoS to kick out every leader about to make a decision I don’t like)

Timing Models

- There is a zoo of timing models that all more or less model reality
  - This makes it hard to compare protocols
- GST seems most popular, but it divorces proof from reality
  - GST: Eventually, everything behaves nice and easy (forever)
  - Implementations: If we keep adapting our timeouts, eventually we get them right
- We've been there before (Chandra-Toueg 96)
  - Failure detectors separate the timing assumption from the protocol
  - Worked very well in the crash failure setting
  - Byzantine Failure Detectors are difficult (we have some ideas, but they're not easy)

Unreliable Failure Detectors for Reliable Distributed Systems

TUSHAR DEEPAK CHANDRA

I.B.M. Thomas J. Watson Research Center, Hawthorne, New York

AND

SAM TOUEG

Cornell University, Ithaca, New York

We introduce the concept of unreliable failure detectors and study how they can be used to solve Consensus in asynchronous systems with crash failures. We characterise unreliable failure detectors in terms of two properties—completeness and accuracy. We show that Consensus can be solved even with unreliable failure detectors that make an infinite number of mistakes, and determine which ones can be used to solve Consensus despite any number of crashes, and which ones require a majority of correct processes. We prove that Consensus and Atomic Broadcast are reducible to each other in asynchronous systems with crash failures; thus, the above results also apply to Atomic Broadcast. A companion paper shows that one of the failure detectors introduced here is the weakest failure detector for solving Consensus [Chandra et al. 1992].

Categories and Subject Descriptors: C.2.4 **[Computer-Communication Networks]**: Distributed Systems—*distributed applications; distributed databases; network operating systems*; C.4 **[Performance of Systems]**: *reliability, availability, and serviceability*; D.1.3 **[Programming Techniques]**: Concurrent programming—*distributed programming*; D.4.5 **[Operating Systems]**: Reliability—*fault-tolerance*; F.1.1 **[Computation by Abstract Devices]**: Models of Computation—*automata; relations among models*; F.1.2 **[Computation by Abstract Devices]**: Modes of Computation—*parallelism and concurrency*; H.2.4 **[Database Management]**: Systems—*concurrency; distributed systems; transaction processing*

General Terms: Algorithms, Reliability, Theory

Cheating with Randomization (Proof of Stake)

Terminate with probability 1

Adversary can stall the protocol forever in theory, but cannot prevent fast termination with high probability

Completely asynchronous *, adapts to real network speed

Price: We need a good source of common randomness, which is hard to do at scale.

Basic Structure of Probabilistic Agreement

1. $v_i \leftarrow$ Initial value of Party p_i
2. **while** not finished **do**
3. perform subprotocol vote with party p_i voting v_i
4. depending on the result, do one of:
   - decide on some value σ
   - set $v_i \leftarrow \sigma$.
   - set $v_i \leftarrow$ Common Coin
5. **endwhile**

* Apart from implementation requirements, such as TCP/IP

Generating a common coin

Required coin Properties:

Unpredictability: An (n,k) coin is unpredictable (i.e., its value cannot be guessed with a probability significantly higher than 0.5) until k parties initiate the common coin protocol.

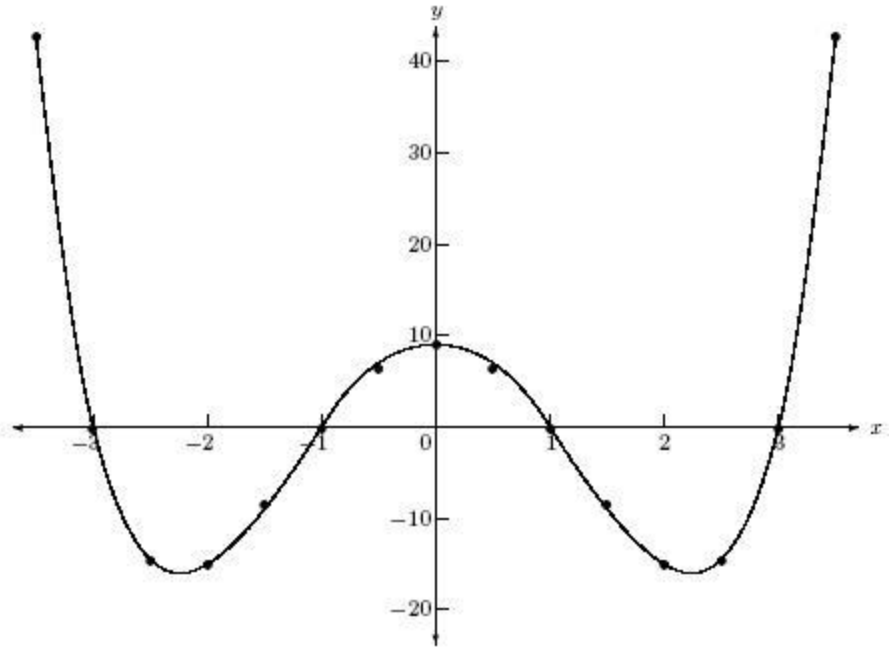
Robustness: It is infeasible for an adversary to create honest parties disagree on the coin.

Note: This is easier than Byzantine Agreement

We have a coin since that does this (CKPS00)

- Fast (only one message, 2 exponentiations per coin)
- Robust (impossible to insert bad shares)
- Fully asynchronous setup possible (but not advised)
- Standard cryptographic assumptions (DDH and Random Oracle, no bilinear groups)

Generating a common coin



$x^4 - 10x^2 + 9$

Common coin based on Shamir secret sharing:

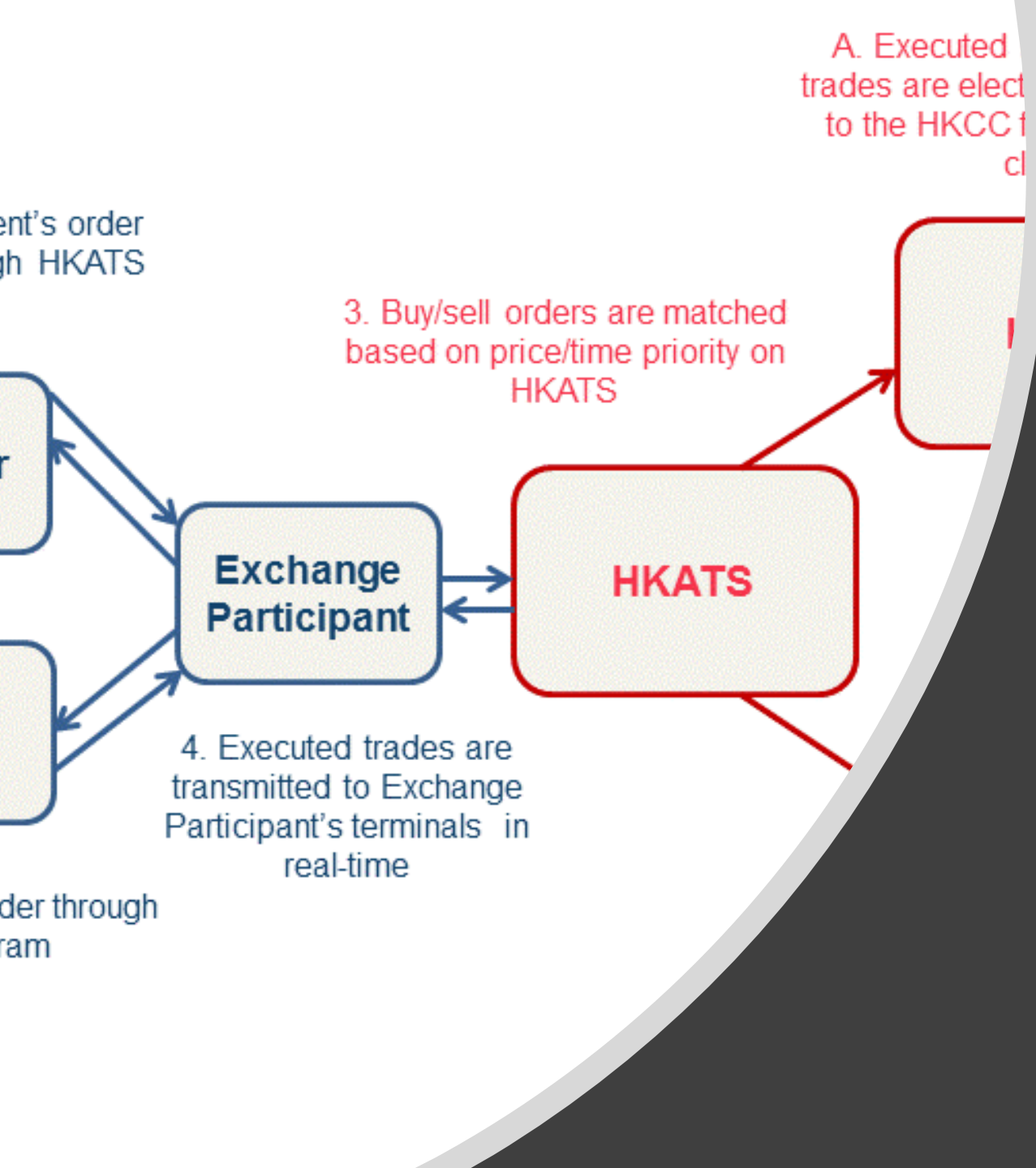
$f(0) = \sum x_i \lambda_i$

$g^{f(0)} = g^{\sum x_i \lambda_i} = \prod g^{x_i \lambda_i}$

Opening a coin means sending g^{x_i} to all parties
Once sufficient shares (e.g., t+1) have arrived, a party can combine the coin.
By that time, at least one honest party opened the coin.

Use Schnorr to make shares verifiable

Note: This is essentially a distributed hash function



Blockchains for the Derivative Market

Assumptions and Requirements

- **Latency is vital**
  - Latency is measured by the time the content of the message is known (and committed), not when it is committed. This is relevant for gossiping based protocols like Tendermint and commit-and reveal schemes
  - We'd rather have bandwidth usage than latency, so no HotStuff Protocol
  - What counts is the final schedule (i.e., Castro-Liskov or Ouroboros won't do)
  - Some traders would prefer never to late; thus, we need a mechanism to block late transactions
- **We'd rather sacrifice on scalability than latency (within reason)**
  - If the ability to scale to 10000 nodes costs latency when we have 100 nodes, this is a no go
- **We need some flexibility**
  - Some sidechains/markets have different requirements
  - Ideally, we can run differently optimised protocols for those (yes we can)

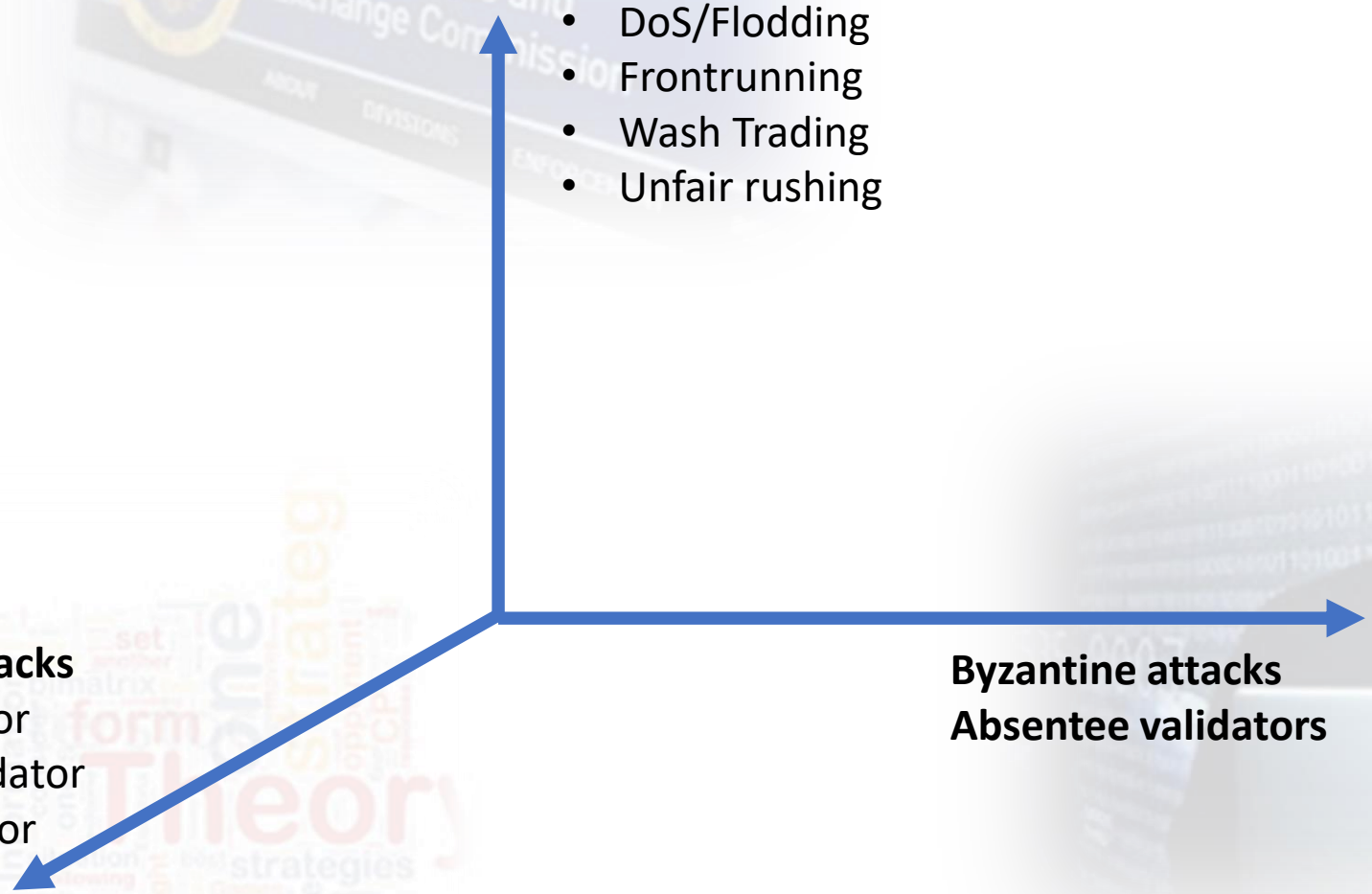
Assumptions and Requirements (2)

Our additional requirements

- Self Managing Network
  - VEGA essentially wants to reduce itself to standards organization and provider/coordinator of one implementation
- Open Membership, but preferable delegation to parties with adequate resources
  - Communication, Computation, Operational Security, ..

- Clean abstraction to (consensus) layer
  - Each layer should be able to be oblivious of how other layers implement their services
    - Allows for various protocols
    - Allows for third party implementations even of single layers
    - Separate implementation of additional properties and services from the consensus layer

The Three Attack Dimensions



Our Assets

- **Validators have resources**
  - Bandwidth, computing speed are available; rather use an extra signature than an extra roundtrip
  - Pipelining and local parallelisation are options
  - Message size is secondary vs number of roundtrips
- **We can punish traitors and thus attack their business case**
  - Traitor Tracing is an option
  - Unfortunately, some adversaries may be poor economists, so on the consensus layer we restrict ourselves to
    - Attacks that slow things down
    - Attacks that prefer some transactions over others
    - Attacks of adversaries that are to stupid to cheat (e.g., signing contradictory messages)
- **Generic analysis layer can provide data for higher level market protocols**

Consensus Layering

Layer Function	**Main Interface**	**Layer Name**
Market mechanisms (ask someone else from VEGA)		App layer
Special Client configuration (e.g, don't execute after time x)	Add to individual request	Client
Configure what happens on the market layer	Add votes as requests	Market Mgnt
Pre/Post Protocols, Pre/Post Transaction Validator	Function call from consensus*	Market Makers
Membership Vote, Consensus Configuration (Timeout, Protocol Parameters, Protocol Version, …)	Add votes as requests	Consensus Mgnt
Timestamps, Randomness, Commit/Reveal	Piggyback payload into votes	Consensus Services
Define what t+1, 2t+1, n-t means	Function call from consensus	Stake Definition
Define Threshold/Multisig/alternative schemes	Function call from consensus	Message Validity
Run the actual consensus protocol		Consensus
Measure if there's oddities/inefficiencies in the network	Monitor messages & times	Monitoring
Reliable/Secure/Consistent Broadcast, either low latency of gossipy	Message pipe & Maintenance	Gossip
Reliable message delivery	Message pipe & Maintenance	P2P

Protocol Agility

- Many protocols can be the best under some circumstances
  - Network timing, best case (no attacker present), average case, worst case, network size, compute/network bound, latency vs throughput, message vs bit complexity, flexibility for staking, desired properties such as finality, front running protection, pipelining, …
- We simply don’t know yet what to expect
- Self Governance
  - We can provide options, but the stakeholders choose which ones are taken
  - No hard forks, no single point of failure through the coders

➔Can we design protocol agility into the system so that the validators can choose from a suite of protocols on the fly ?

➔Verified with CL99, KS00, CKPS01, Tendermint, HotStuff/Libra, Ouroboros*

➔Similarly, we can replace other layers (Gossip, P2P, threshold signatures, …)

* Needs a transition protocol

Fairness Protection

Annoying Impossibility results (1)

Life ain’t fair.

For small scale systems, this can be resolved(ish) using threshold encryption. Unless you worry about metadata or scale.

“If all honest parties see event A happen before event B , then A should be scheduled before B.”

This is impossible in the presence of a single a corrupted participant

Frontrunning & Rushing

Frontrunning: A validator can see a transaction before it is scheduled and attempt to get their own transactions scheduled before

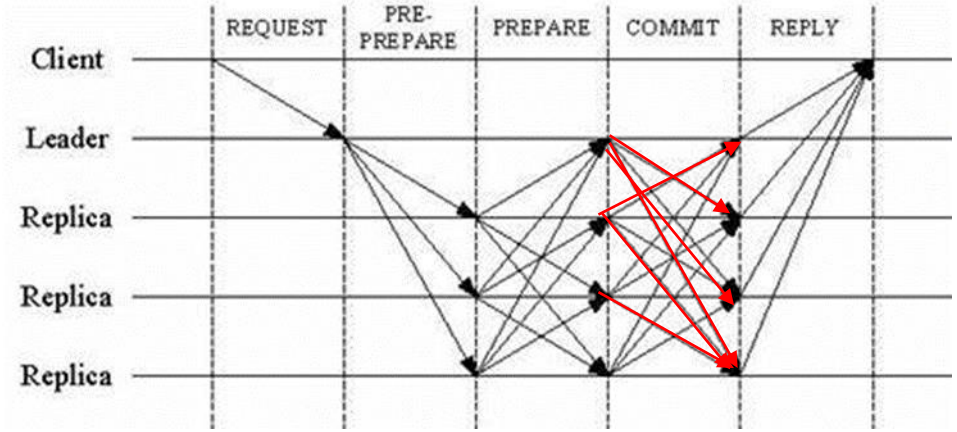
Rushing: A validator helps a trader being scheduled with priority as a reaction of a market event

Steps to make this harder are

- Leaderless protocols (CKPS00, Honeybadger, Leader-Removal-Preprotocol)
- Hard to predict, quickly changing leaders (HotStuff)
- DoS protection for leaders (Tendermint)

Causality (Commit and Reveal)

- We can (verifiably) encrypt a transaction so that a quorum of validators is needed to decrypt it
  - Either the transaction content or also the originator
  - The latter needs extra spam protection
- The decryption can be piggybagged into existing protocol messages (with care)



Introducing (some) fairness

“Suppose there is a time source that provides each party with a time*. If there is a time x so that all honest validators see request r_1 before their timer reaches x and r_2 after their timer reaches x, then r_1 has to be scheduled before r_2”

*While not required, it would be enormously helpful in a practical setting if that time was somewhat synchronized, e.g., by using GPS time.

“If t+1 honest parties see request r_1 before r_2, then r_1 is scheduled in the same or an earlier block than r_2.”*

We can then, for example

- Give all of them the same conditions
- Randomize the execution order
- Go by timestamps

*Some constraints on these definition will be required to avoid, for example, arbitrary block sizes

Spam Protection

Issue: Rouge traders can spam the network with thousands of empty transactions (or play Cryptokitties, use our chain for file sharing, …)

Approach:

Every valid request needs to be hash-bound to an already scheduled block no older than x blocks (or y seconds). Binding more requests to the same block (by the same party) requires solving increasingly hard puzzles (Bitcoin style).

Disadvantage: Traders better be fast enough to keep up. Me might want a mitigation mechanism for very slow traders (e.g., we allow 1 unbound transaction per trader in the pool for every 10 blocks, but will not accept any additional ones to even enter the pool.

Late Delivery Protection

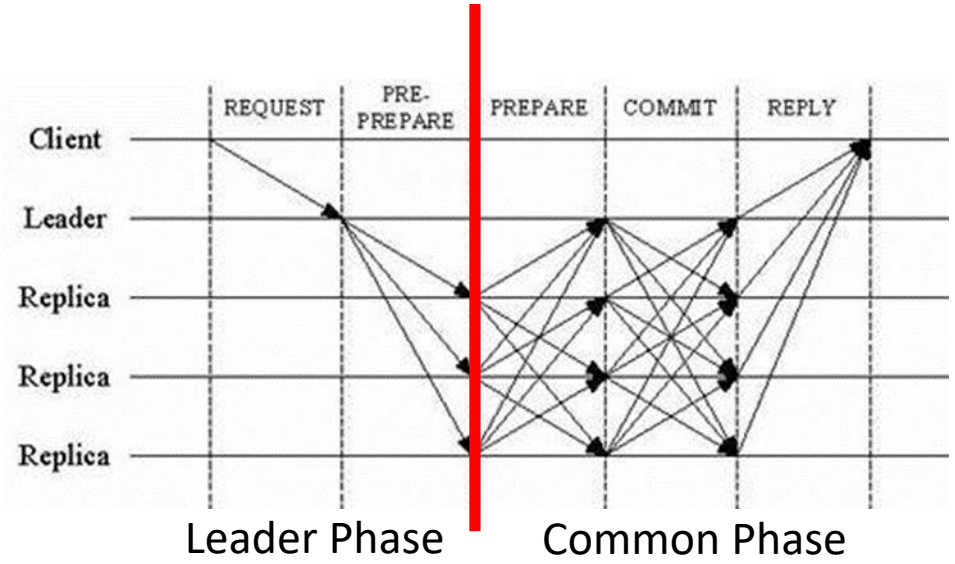
Issue: In some markets, traders prefer requests to not be scheduled if they suffer a too big delay

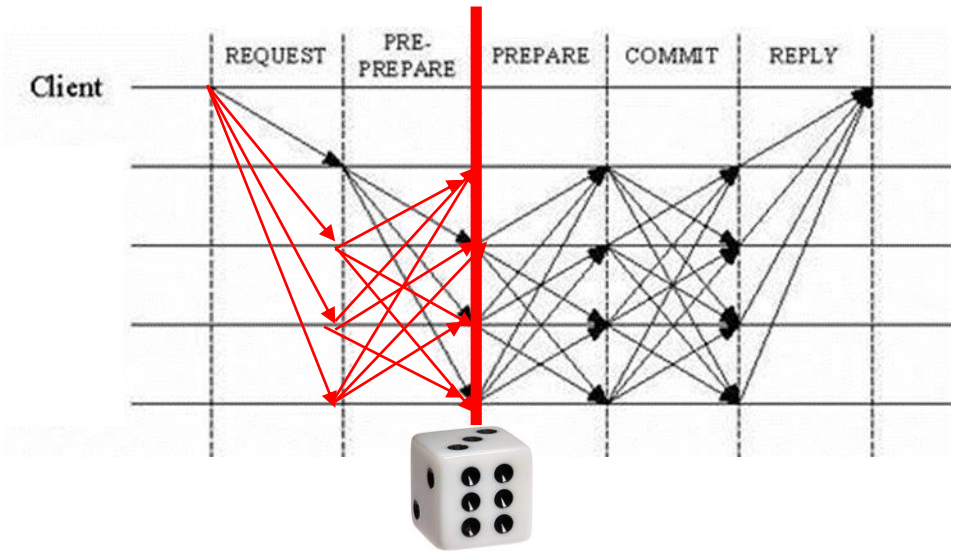
Approach: On the market-makers discretion, traders can add a maximum time or a maximum block number to a request. After a block is scheduled, the requests validity is verified against the block timestamp and number, and it is not executed if either test fails.

DoS and Leader Protection pre-protocol

- Prevent an attacker from targeting the next leader, e.g., by DoS

- Everybody acts as if they where the next leader and initiates a vote
- Piggybacked on that vote is a reveal for a (n-t) common dice between 1 and n
- The leader is the one the dice fell on. By this time the leaders job is likely already done
- Obviously, this only makes sense if the leader changes fast





Latency Minimization

Parallel Protocol Runs

- Pre- and post protocols (e.g., for fairness) can be run in parallel to previous/following rounds
  - When sending the final vote, add the transactions for the next round
  - Send decryption shares while the leader performs the next broadcast

- As in Hotstuff, the next leader can start when the previous one sent the first message
  - Needs to be done with case not to cause bad interference

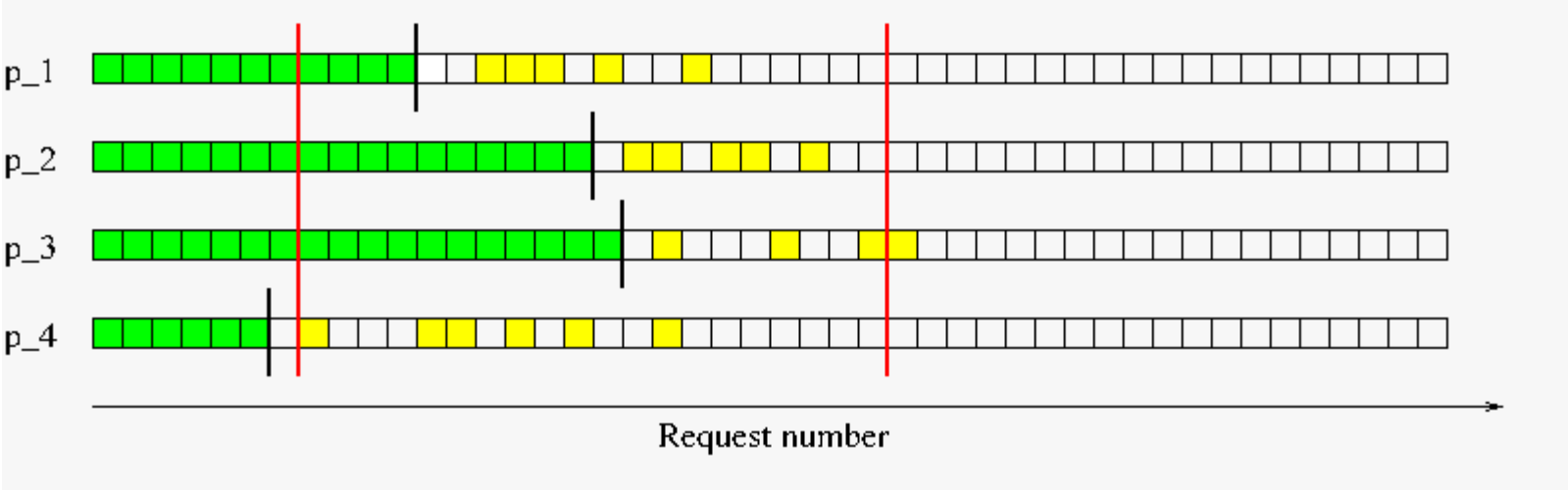
Pipelining

The original BFT Protocols created an enumeration, not a hash-chain

Functionally equivalent, but different way of thinking

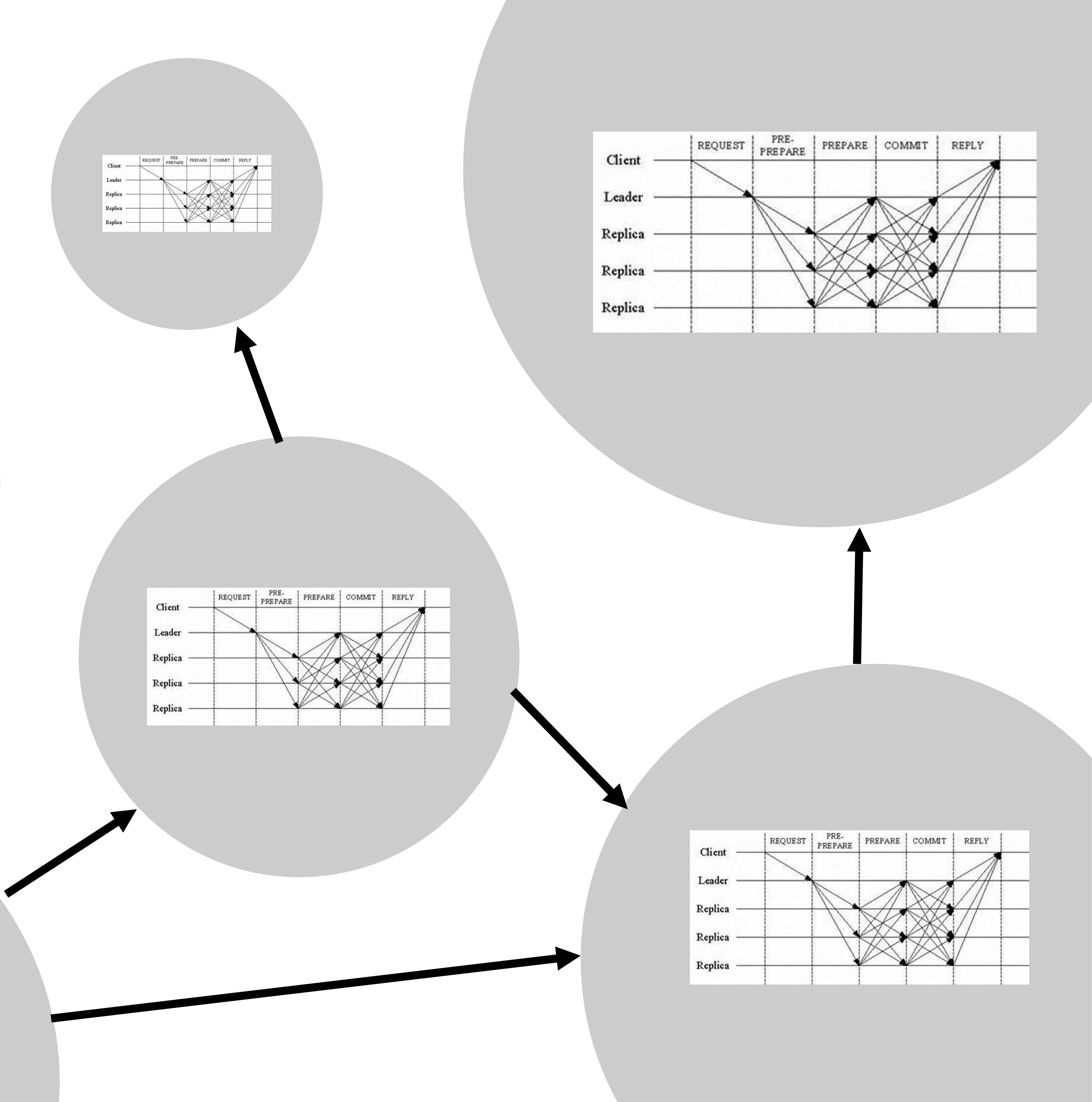
This made it easier to process several blocks in parallel

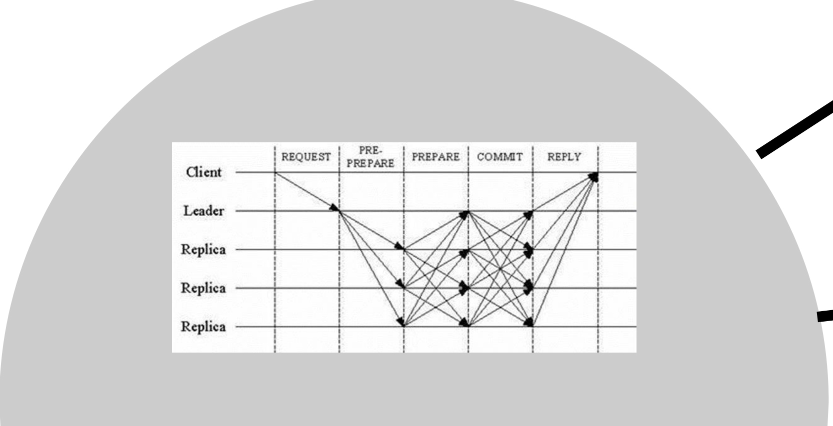
- Lower latency (every block gets processed immediately)
- More mess if things go wrong
- Better resource utilization
- Different fairness issues (more unconfirmed blocks floating around)
- Requires lazy verification, as a transaction can be valid when started and invalid when done
- There is no fundamental reason to not integrate this is todays protocols



Pipelining and Parallel Processing

- Suppose a validator consists of several processors.
- If there is a deterministic matching between processor and serial number, each request can be executed by its own processor
- Inter-processor communication is only needed for the final validation
- There is no reason processors need to be geographically close; rather, they can be close to other replicas processors





Flocking

One issue with latency is that there is a physical limit for messages to travel the world; e.g., New York-Frankfurt =88ms (roundtrip around the works: >600 ms). Thus, even the fastest protocols need at least 1 second latency if the validators are well distributed.

With flocking, we define an active set of validator nodes that are close to each other. This is a little bit like Algorand, but with geography instead of randomness

Flocking: Setup & Management

- We assume that we have validator organizations, which are able to run multiple validator nodes distributed over the world. At any given time, only one of these nodes is active.
- Active nodes of the different validator organizations are geographically close together, thus minimizing latency
- A validator organization can switch to a different node, e.g., through
  - Time of day – keep the flock in the part of the world that has working hours
  - Latency measurement – if another node is closer to the flock, switch to that one
  - Timeout: You've just been active for too long
- All flock coordination can be done based on timing measurements and on chain management commands

Flocking: Dangers

Flocknapping: Geographic proximity increases the danger that a large part of the active validators end up in the same country, especially for big countries (China, US, Russia).

Higher Attack Surface: Attacker can do some damage with controlling only one processor of a replica (esp. for threshold cryptography)

Uncooperative nodes: We need to be Able to force nodes to step back

Managing Stakes and Votes

Annoying Impossibility results (1)

It is an illusion that the scale of major blockchains overcomes the 1/3 threshold implicitly.

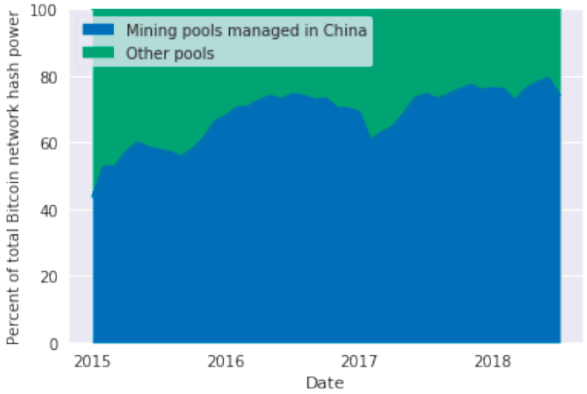
Malicious attacks are not independent.
How many miners have different

- Operating Systems
- Implementations
- Libraries
- Legal authorities
- Influenceability by countries controlling the lead currency
- … ?

This was addressed 1978 with resynchronization and 1997 with General Adversary Structures.

Ethereum:
One third of all assets (for PoS) is about 100 wallets if everybody else participates.
Assuming there are a lot of passive assets, it is even worse

Bitcoin:
Due to mining pools and rigs, this assumption is long gone.
It doesn't even need malice to cause issues.



Mining pool	Located in China	Estimated share of network hash rate
F2Pool	Yes	22.17
AntPool	Yes	21.54
BTCC	Yes	12.79
BitFury	No	12.39
BW Pool	Yes	7.84
KnCMiner	No	4.89
SlushPool	No	4.72
21 Inc.	No	2.27

https://arxiv.org/pdf/1810.02466.pdf

Beyond thresholds: Property based protocols

If we use thresholds in our protocols, what do we actually mean ?

wait for t+1 messages

Property: you can expect to have input from at least one honest guy

wait until you heard from people from at least 5 countries

wait until you heard from at least 3 different implementations

wait for 2t+1 messages

Property: you expect to have a honest majority

wait until you heard from people of at least 9 countries

wait until you heard from at least 5 different implementations

wait for n-t messages

Property: it doesn’t make sense to wait any longer

wait until you heard of all countries active in the last 24 hours minus 4

Wait until you got 2/3 of all people active in the last 24 hours

Wait until you heard from people that sum up to 2/3 of the combined votes of the last 3 months

Or even more general…

Let $\mathcal{P}$ be the set of all participants. An *adversary structure* $\mathcal{Z}$ is a monotone set of classes (C, B) of subsets of $\mathcal{P}$ (i.e., $C, B \subset \mathcal{P}$) [FHM99]. The adversary structure $\mathcal{Z}$ satisfies the predicate $\mathcal{Q}^{(3,2)}(\mathcal{P}, \mathcal{Z})$, if $\forall (B_1, C_1), (B_2, C_2), (B_3, C_3) \in \mathcal{Z} : \{B_1 \cup B_2 \cup B_3 \cup C_1 \cup C_2\} \neq \mathcal{P}$.

I.e.: I make a list of all sets of parties that are allowed to be corrupted together. In no three such sets amount to the total set of parties, I can solve agreement.

Using properties prevents me from defining millions of sets.

This is also vital to stay poly-bound in the number of parties.

Integration into existing protocols

- For almost all protocols, this can be integrated easily
  - Replace *(more than t+1)* messages with *(set with at least one honest party)* etc.
  - For an implementation, it would be nice to use an external function call for this
  - *Verify your proofs*
    - This has been done with CL99, CKPS99, KS01
    - Working on Tendermint, Libra, Honeybadger
    - Protocols that count differently are future work (e.g., Ripple)
- Threshold Crypto is doable, but can get ugly in the most general case
  - Also, cryptographic proofs are substantially more involved than protocol proofs

Other option

- 2/3 of the stake in 2/3 of the world regions
- This guarantees a good local distribution
  - Any individual big player can only dominate one region
    - If you get another validator inside a popular region, it just counts (and pays) less
  - Regions also can be weighted
- We can go arbitrary deep
  - 2/3 of the stake of 2/3 of the implementations of 2/3 of the regions…….
- This can also be used to avoid the amazon cloud effect (i.e., 90 out of 100 validators run in the same cloud)

Integrating Threshold Cryptography

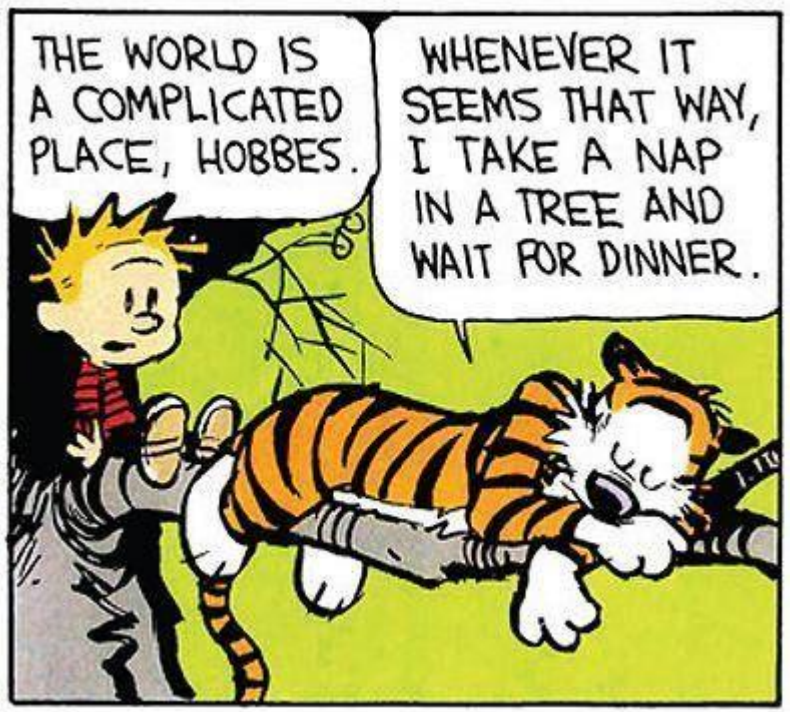
- Threshold cryptography doesn't work well with weights; it's usually one share, one vote
  - We can make 100000 shares and give more to large stakeholders, but then we pay with communication bandwidth
  - If we round down small stakeholders, this could be manageable
- Threshold encryption is only used for commit/reveal; the thresholds here don't need to be the same as for consensus, its only important to delay the reveal until the order is semi-committed
- Threshold signatures are subtle; we don't want to be in a situation where we committed and can't sign, or didn't commit and can.
  - The 2/3 of 2/3 is probably easier to handle, but needs to be looked at
- Threshold Hash function (common randomness) can be managed(ish)
  - The 2/3 of 2/3 works very nicely for that

Multiple Thresholds

- Consensus Layer Management
  - Set parameters, define algorithms, decide on new members
- Voting power
  - Stakes in the token
  - Proven (un) trustworthiness
  - Geographic/Legislative distribution
- Leader (for leader based protocols)
  - Performance/Security is more vital (leader is bottleneck)
  - Trust levels are higher (possibility to cheat by frontrunning)
  - Leaders with secure hardware can speed up the protocol
- Threshold Cryptography
  - Causal Broadcast: Encrypt and reveal on schedule (can use lower threshold)
  - Common Coin/Randomness for randomized protocols (can use lower threshold)
  - Threshold signatures (this is where things get complicated)

Threshold Signatures: It’s complicated

- We have less wiggling room here: A situation where a transaction is agreed and can’t be signed or where it can be signed but hasn’t been agreed is not acceptable.
- Most protocols are interactive, which adds a lot of headache
- Non-Interactive schemes (e.g., Shoup99) work internally, but don’t generate a standard-compliant signature
- We also need to watch stake/membership changes here

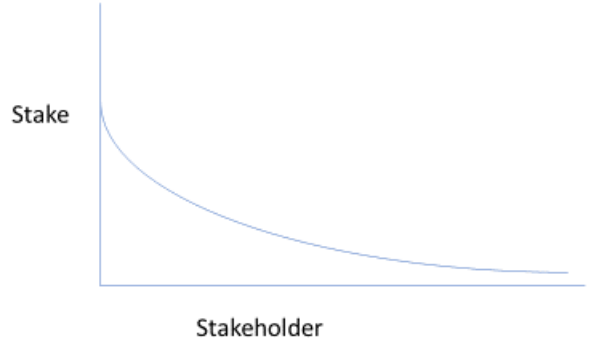


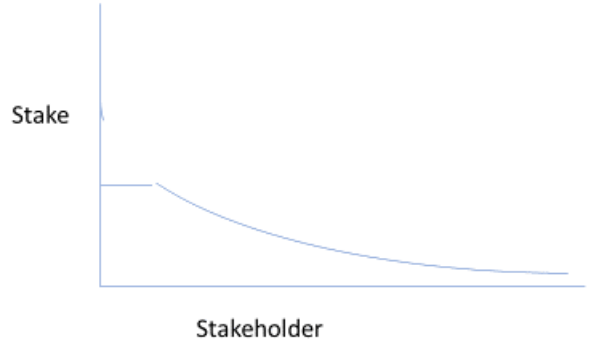
Game Theoretic Issues

Managing validator numbers (options)

Issue: We want to avoid having thousands of validators, and small stakeholders to become missing/ underressourced validatoers

- Stakes are rounded (down)
  - Small changes in ownership don't change voting composition
  - Easier for threshold cryptography
- Stake below a certain threshold (say, .5%) are rounded to zero
  - Motivate people to delegate
  - Push towards validators with sufficient resources
- Stakes of one validator can't count more than 5% (prevent monopolies)
  - As validators need to pay parties that delegate to them, having too big stakes looses someone money
  - This might make limited sense if validators are anonymous





Greedy Validators

Issue: Validators interests can be counterproductive

- Change the anti-monopoly threshold

Trivial, but easy to miss observation:

Once we have a running blockchain, we can organise votes of parties that are not involved in the consensus protocol at all

Thus, we can have an equivalent to separation of power:

- The executive is the validators running the blockchain
- The legislative is a congress of stakeholders (traders, token owners, …) that votes on parameters the executive shouldn't vote on, such as new validators, thresholds for payment, impeaching validators, …
- Just for completeness, the judicative is VEGA (as code is law)

This handles most economic attacks by greedy validators trying to game membership.

Lazy Validators

Issue: A lazy validator can put almost no work in and still get paid.

Approach: In every block, we add a valuation by all validators on how their peers performed (similar to the timestamping protocol). This valuation is used by the market-layer to determine payment (e.g., linked to the performance on the last 100 blocks).

This is not bulletproof (an alliance of validators can decide to ignore one specific validator and then claim it didn’t perform), but that’s still better than freeloading.

Blockchains for the Derivative Market

Klaus@vega.xyz