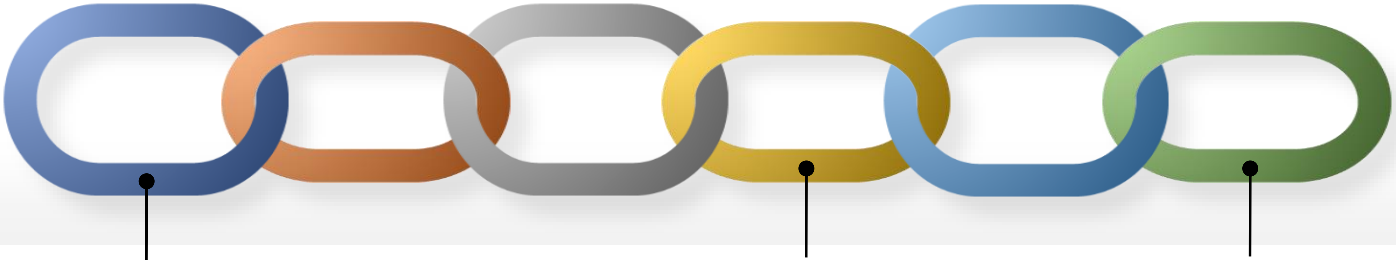
Blockchain consensus mechanisms

Ivan Visconti
ivan.visconti@gmail.com

International School on Algorand Smart Contracts

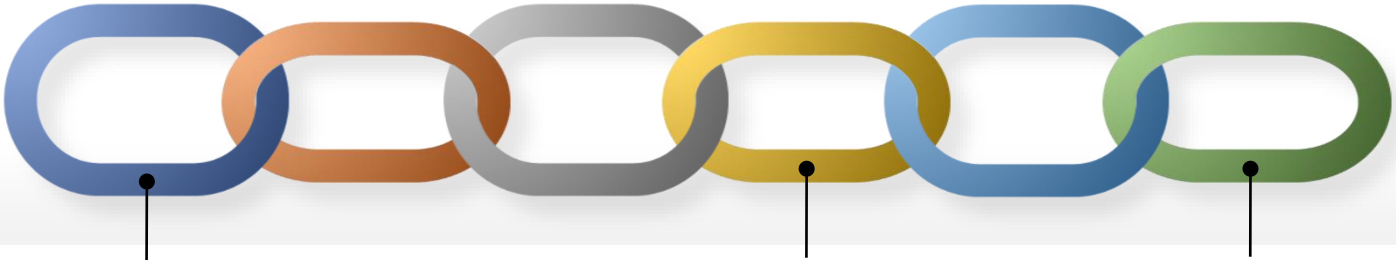
Outline

- Part 1: Old-School Consensus
  (i.e., Permissioned Blockchains)
- Part 2: Nakamoto Consensus
  (i.e., Permissionless Blockchains with slow finality)
- Part 3: Algorand Consensus
  (i.e., Permissionless Blockchains with fast finality)



Informal definition of blockchain

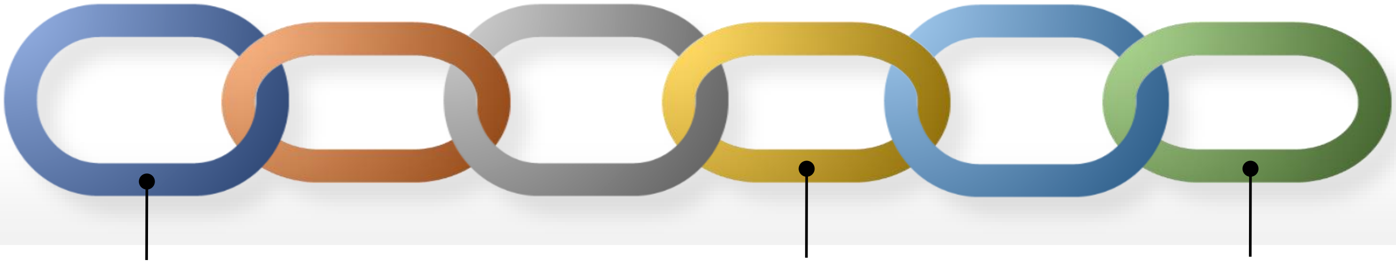
A blockchain is a decentralized computer publicly running programs (smart contracts) on inputs received through transactions. The state of this computer is uniquely defined by the sequence of transactions executed from the genesis.



Where is the blockchain stored?

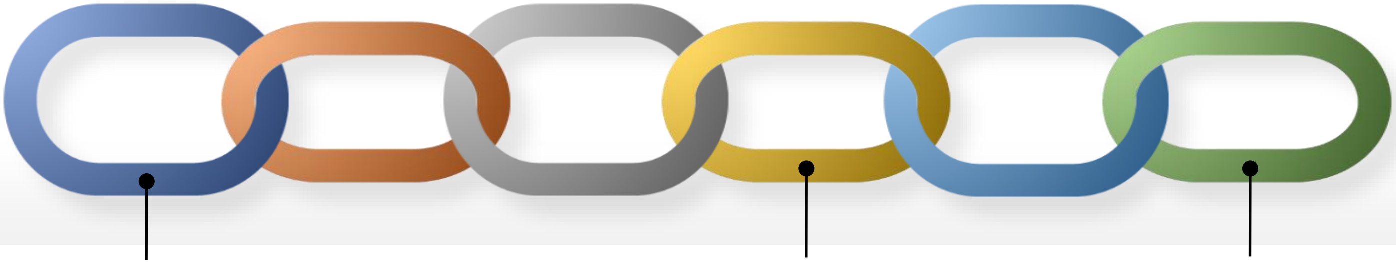
The only requirement is that it is jointly maintained by several (pretty much mutually distrustful) computers.

For simplicity, we will assume that "jointly maintained" means that multiple computers store a fully copy of the blockchain (i.e., the entire sequence of transactions). This is true in several cases (e.g., full nodes in Bitcoin) but it is not always necessarily true.



Major problem towards designing a blockchain:

How do we make sure that all computers maintaining a blockchain have a common view of its state?



Major problem towards designing a blockchain:

How do we make sure that all computers maintaining a blockchain have a common view of its state?

…even in the presence of faults (i.e., crashes, omissions, byzantine behavior)…

Blockchain: a possible view in multiple layers

Application Layer
Scalability Layer
Consensus Layer

Consensus

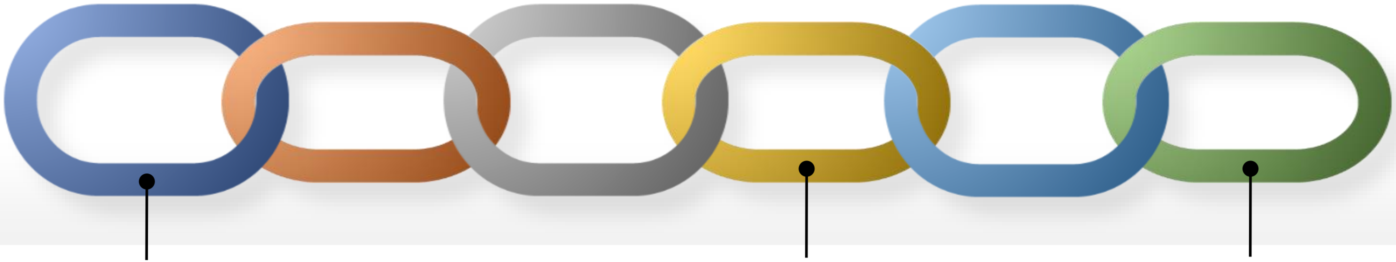
[LSP 82]: Byzantine Agreement /Broadcast (Generals) problem

Some generals are attacking a fortress and must decide as a group only whether to attack or retreat. Some generals may prefer to attack, while others may prefer to retreat. The important thing is that all generals agree on a common decision.

[LSP 82]: Byzantine Agreement /Broadcast (Generals) problem

Some generals are attacking a fortress and must decide as a group only whether to attack or retreat. Some generals may prefer to attack, while others may prefer to retreat. The important thing is that all generals agree on a common decision.

Subtlety: if there is only one source then it's a specific (simpler) case called broadcast and it's the actual Byzantine Generals problem. With multiple generals we have the Byzantine Agreement problem.



Cryptography

Several cryptographic tools have been proposed to design blockchains with improved features. We will recall some of such useful tools just before using them.

Ittai Abraham, Oct 23, 2021 (twitter, [Abr21]):
"Cryptography is eating the world"

Cryptography

Several cryptographic tools have been proposed to design blockchains with improved features. We will recall some of such useful tools just before using them.

We start with digital signatures.

Ittai Abraham, Oct 23, 2021 (twitter, [Abr21]):
"Cryptography is eating the world"

Digital signatures schemes

(sk, pk) = KeyGen(keylength)

s = Sign(sk, m)
(can be deterministic)

0/1 = Ver(pk, m, s)

Properties of a signature scheme

Ver(pk, m, Sign(sk, m)) == 1

No adversary can produce an accepting signature of a new (i.e., not already signed by the owner of sk) message

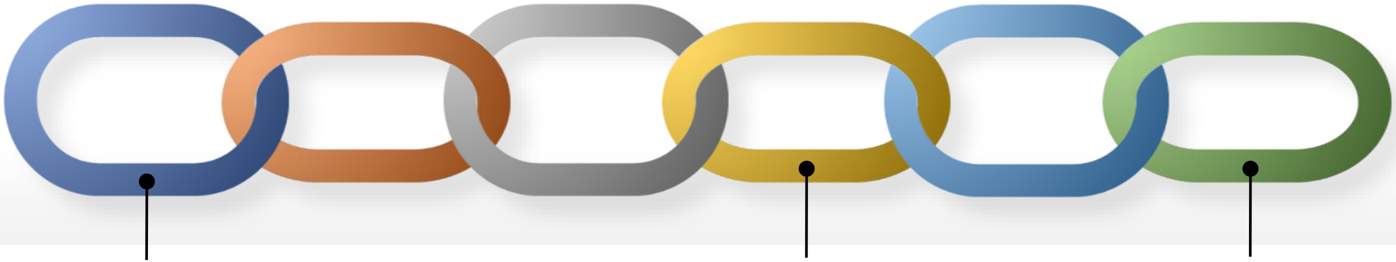
Signature schemes achieve non-repudiability

PKI and "Authenticated" functionalities

A public-key infrastructure guarantees in the eyes of all honest players, that a given public key belongs to some specific entity.

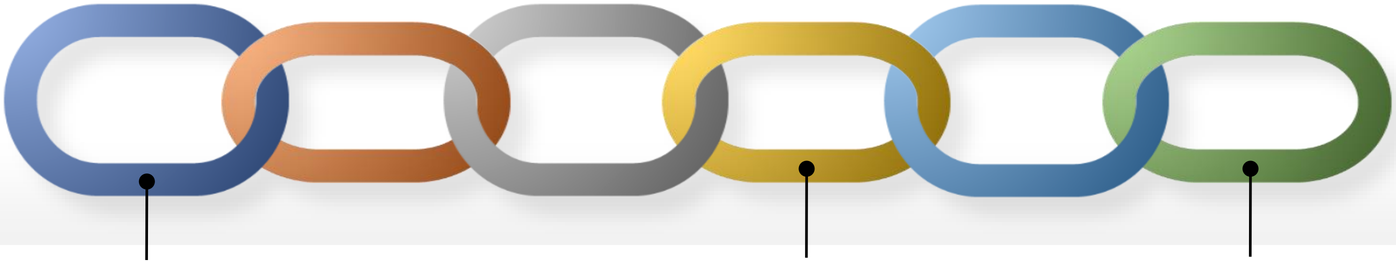
This is beneficial in protocol design since an adversary that generates two inconsistent messages can be detected and slashed.

Functionalities achieved through protocols that explore this feature are sometimes called "authenticated".



State Machine Replication [Lam78,Sch90] (Ledger Consensus)

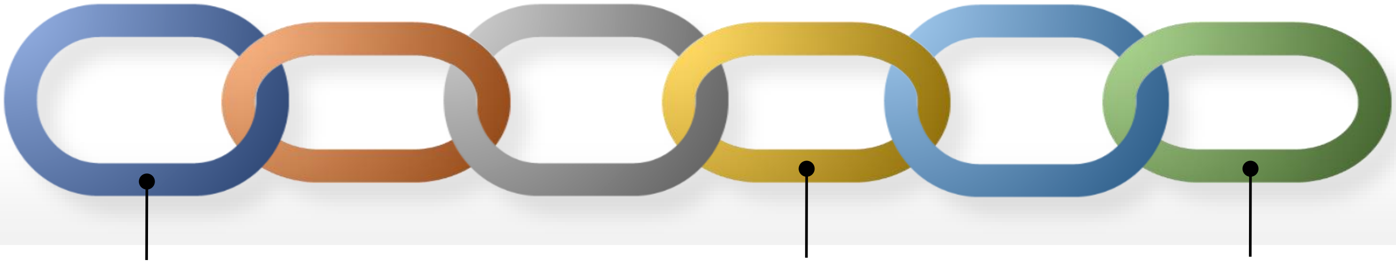
One server (a machine) is split into multiple servers all sharing the same state (i.e., realizing the same machine) to prevent faults, even byzantine faults (i.e., corruption), therefore leaning to distributed (and even decentralized) systems.
It corresponds to what is nowadays known as *permissioned blockchain* where the number and the identities of nodes maintaining the blockchain is known up front.



State Machine Replication [Lam78,Sch90] (Ledger Consensus)

One server (a machine) is split into multiple servers all sharing the same state (i.e., realizing the same machine) to prevent faults, even byzantine faults (i.e., corruption), therefore leaning to distributed (and even decentralized) systems.
It corresponds to what is nowadays known as permissioned blockchain where the number and the identities of nodes maintaining the blockchain is known up front.
Realizing SMR (i.e., resolving the ledger consensus problem) requires to achieve two main goals, namely:

Consistency: honest nodes must have the same view of the state of the machine

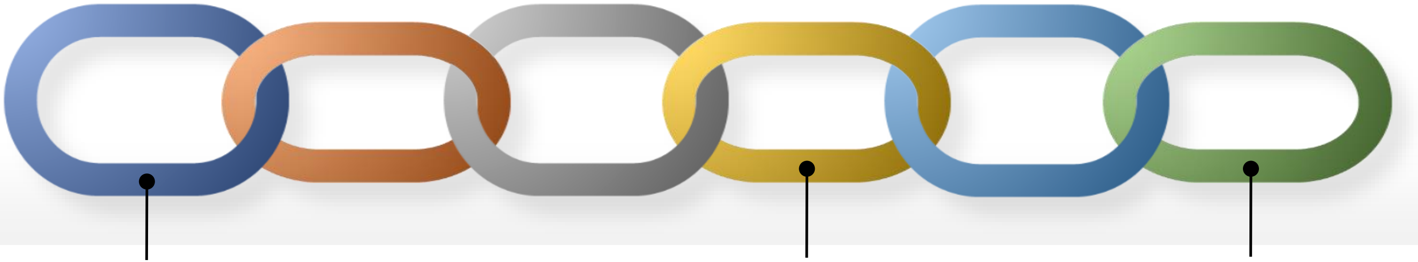


State Machine Replication [Lam78,Sch90] (Ledger Consensus)

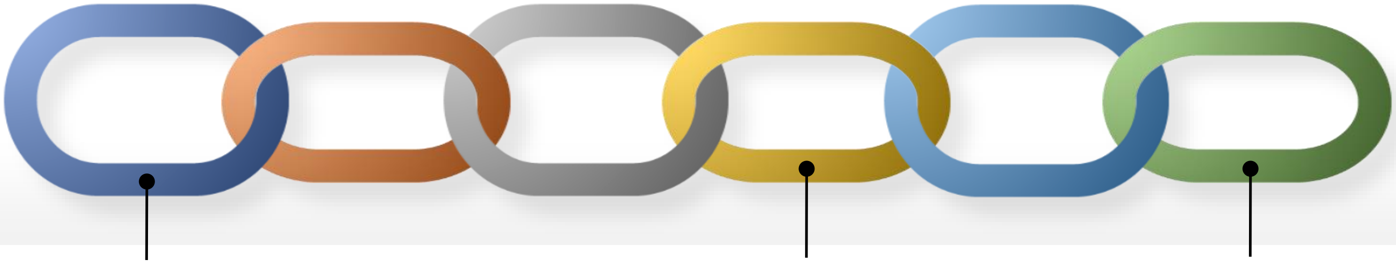
One server (a machine) is split into multiple servers all sharing the same state (i.e., realizing the same machine) to prevent faults, even byzantine faults (i.e., corruption), therefore leaning to distributed (and even decentralized) systems.
It corresponds to what is nowadays known as permissioned blockchain where the number and the identities of nodes maintaining the blockchain is known up front.
Realizing SMR (i.e., resolving the ledger consensus problem) requires to achieve two main goals, namely:

Consistency: honest nodes must have the same view of the state of the machine

Liveness: every valid transaction will be executed updating the state of the machine



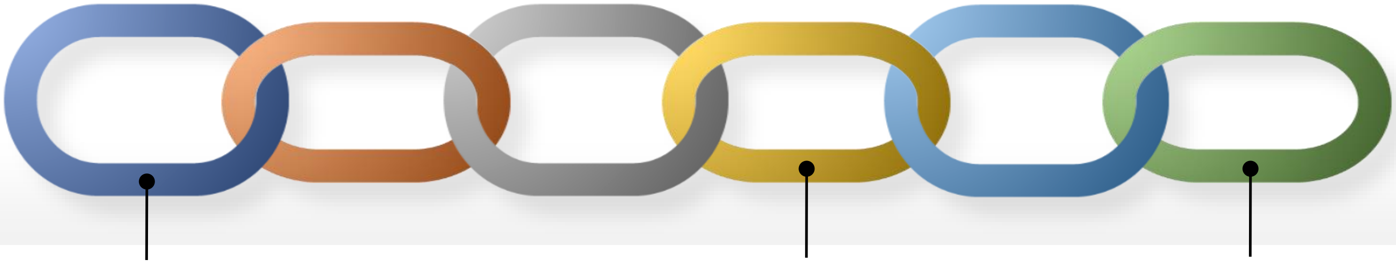
Permissioned blockchain assumes
known governance (i.e., nodes that maintain the full list of transactions)



Permissioned blockchain assumes

known governance (i.e., nodes that maintain the full list of transactions)

In the synchronous model, without adversaries, solving the consensus problem is trivial: just with rotating leaders each announcing a block of transactions when leading, you get a permissioned blockchain (consistency and liveness are trivially satisfied).



Permissioned blockchain assumes

known governance (i.e., nodes that maintain the full list of transactions)

In the synchronous model, without adversaries, solving the consensus problem is trivial: just with rotating leaders each announcing a block of transactions when leading, you get a permissioned blockchain (consistency and liveness are trivially satisfied).

In the synchronous model there is a global clock and all nodes are aligned to it. Everyone can expect what others are doing at a given timestep, and a message sent during timestamp t is delivered by timestamp t+1.

This might be excessive in some real-world scenarios, particularly when breaking synchrony (e.g., through a DDOS attack) can have a devastating impact.

From Byzantine Broadcast to SMR / Ledger Consensus

In the synchronous model, without adversaries, solving the consensus problem is trivial: just with rotating leaders each announcing a block of transactions when leading, you get a permissioned block chain (consistency and liveness are trivially satisfied).

From Byzantine Broadcast to SMR / Ledger Consensus

In the synchronous model, without adversaries, solving the consensus problem is trivial: just with rotating leaders each announcing a block of transactions when leading, you get a permissioned block chain (consistency and liveness are trivially satisfied).

If a byzantine broadcast (BB) protocol is used by the leader, then we get a permissioned blockchain in the presence of byzantine adversaries:

BB: agreement (honest nodes get same msg) => consistency

BB: validity (sender is honest with input m, honest nodes get msg=m) => liveness

From Byzantine Broadcast to SMR / Ledger Consensus

In the synchronous model, without adversaries, solving the consensus problem is trivial: just with rotating leaders each announcing a block of transactions when leading, you get a permissioned block chain (consistency and liveness are trivially satisfied).

If a byzantine broadcast (BB) protocol is used by the leader, then we get a permissioned blockchain in the presence of byzantine adversaries:

BB: agreement (honest nodes get same msg) => consistency

BB: validity (sender is honest with input m, honest nodes get msg=m) => liveness
(we omit the (obvious) termination condition, for simplicity)

[DS83] gives a simple BB protocol (with sync+permissioned+PKI)
for any number of faulty nodes (but for useful SMR honesty should be majority).

From Byzantine Broadcast to SMR / Ledger Consensus

In the synchronous model, without adversaries, solving the consensus problem is trivial: just with rotating leaders each announcing a block of transactions when leading, you get a permissioned block chain (consistency and liveness are trivially satisfied).

If a byzantine broadcast (BB) protocol is used by the leader, then we get a permissioned blockchain in the presence of byzantine adversaries:

BB: agreement (honest nodes get same msg) => consistency

BB: validity (sender is honest with input m, honest nodes get msg=m) => liveness

(we omit the (obvious) termination condition, for simplicity)

[DS83] gives a simple BB protocol (with sync+permissioned+PKI)
for any number of faulty nodes (but for useful SMR honesty should be majority).
PKI is crucial to tolerate that faulty nodes are more than 1/3 of the total [PSL80]

Recap

SMR / Ledger Consensus: (n rotating nodes proposing state updates)
Goal: Consistency + Liveness

Byzantine Broadcast: (one node sending a message to all others)
Goal: Agreement + Validity

Rotating Leaders + BB [DS83] + sync model + permissioned + PKI
==> SMR with honest majority (without PKI then super majority is required)

Relaxing the Synchronous Model

We have seen that in the synchronous model we can construct efficient protocols, but the model might not correspond sufficiently well to reality.

Relaxing the Synchronous Model

We have seen that in the synchronous model we can construct efficient protocols, but the model might not correspond sufficiently well to reality.

(Fully) asynchronous model: there is no upperbound on the time needed for a sent message to reach the receiver (but messages eventually are received).
PROs: great applicability to concrete scenarios;
CONs: very hard setting affected by negative results (e.g., [FLP85]: impossibility of deterministic byzantine agreement even with just one crashing node, due to the difficulty of distinguishing a crash from a delayed message).

Relaxing the Synchronous Model

We have seen that in the synchronous model we can construct efficient protocols, but the model might not correspond sufficiently well to reality.

(Fully) asynchronous model: there is no upperbound on the time needed for a sent message to reach the receiver (but messages eventually are received).
PROs: great applicability to concrete scenarios;
CONs: very hard setting affected by negative results (e.g., [FLP85]: impossibility of deterministic byzantine agreement even with just one crashing node, due to the difficulty of distinguishing a crash from a delayed message).

Fortunately, we can get the best of both worlds: the partially synchronous model [DLS88].

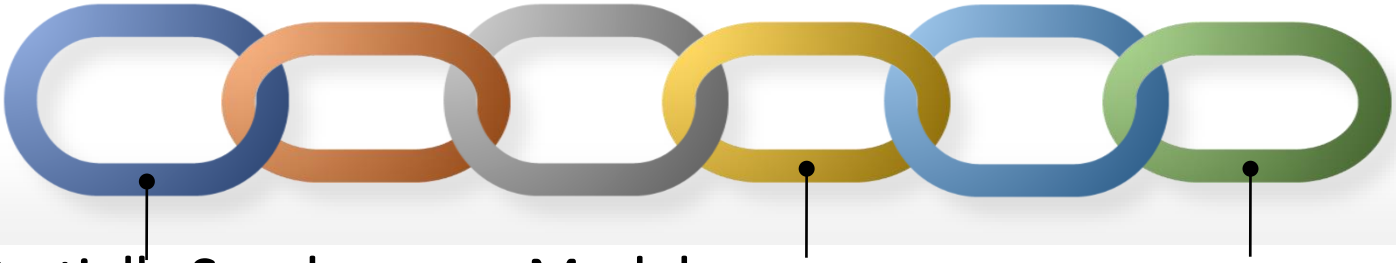
The Partially Synchronous Model

Main idea: we assume to be in general the synchronous model but knowing that sometimes for a time window of unknown length, the synchronous model can fail.

Summing up, we want to guarantee:

- while good sync, consistency + liveness
- while poor sync, consistency

Note: Nakamoto showed the power of targeting liveness instead of consistency when things go wrong (i.e., in the presence of forks).



The Partially Synchronous Model

Summing up, we want:

- while good sync, consistency + liveness
- while poor sync, consistency

The above goal can be achieved if and only if corruption is under the 1/3 threshold [DLS88].

"only if": the point is that with t corrupted nodes, the honest node can not wait for more than n-t messages, but t of those n-t messages could come from the adversary. So, when t>=n/3 we have that the messages from honest parties out of those n-t messages are not majority. PKI does not avoid this bound.

The Partially Synchronous Model

Summing up, we want:

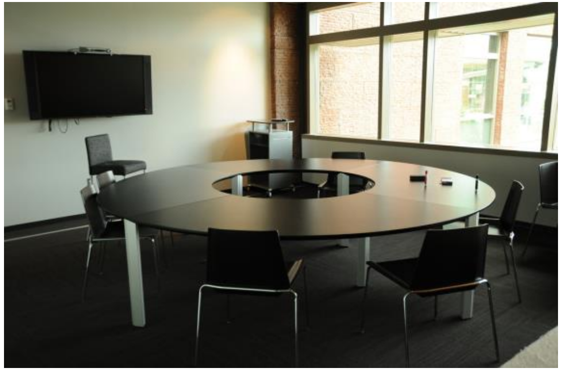
- while good sync, consistency + liveness
- while poor sync, consistency

The above goal can be achieved if and only if corruption is under the 1/3 threshold [DLS88].

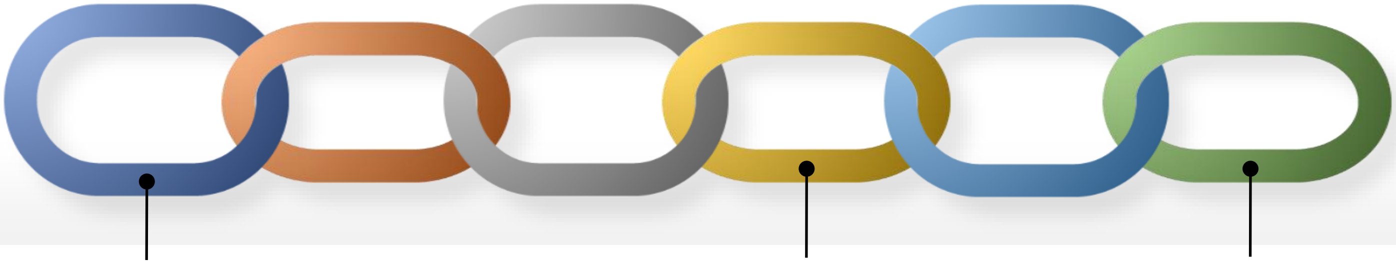
There exists many protocols achieving this bound and often they are grouped in the alias "BFT protocol". Recall: so far, we have considered the permissioned setting.

Permissioned Blockchain

some known organizations decide the state of the computer



- no waste of energy
- strong consistency
- fast transactions if network not under attack
- honest majority (usually super majority) is required



Outline

- Part 1: Old-School Consensus
  (i.e., Permissioned Blockchains)
- Part 2: Nakamoto Consensus
  (i.e., Permissionless Blockchains with slow finality)
- Part 3: Algorand Consensus
  (i.e., Permissionless Blockchains with fast finality)

From Permissioned to Permissionless: is it possible at all?

(recall) Permissioned blockchain assumes

known governance (i.e., nodes that maintain the full list of transactions)

From Permissioned to Permissionless: is it possible at all?

(recall) Permissioned blockchain assumes
known governance (i.e., nodes that maintain the full list of transactions)

In the synchronous model, without adversaries, solving the consensus problem is trivial: just with rotating leaders each announcing a block of transactions when leading, you get a permissioned blockchain (consistency and liveness are trivially satisfied).

From Permissioned to Permissionless: is it possible at all?

(recall) Permissioned blockchain assumes
known governance (i.e., nodes that maintain the full list of transactions)

In the synchronous model, without adversaries, solving the consensus problem is trivial: just with rotating leaders each announcing a block of transactions when leading, you get a permissioned blockchain (consistency and liveness are trivially satisfied).

In the permissionless setting the above approach fails spectacularly: nothing is certified, governance is open, the set of possible leaders is unknown and dynamic.

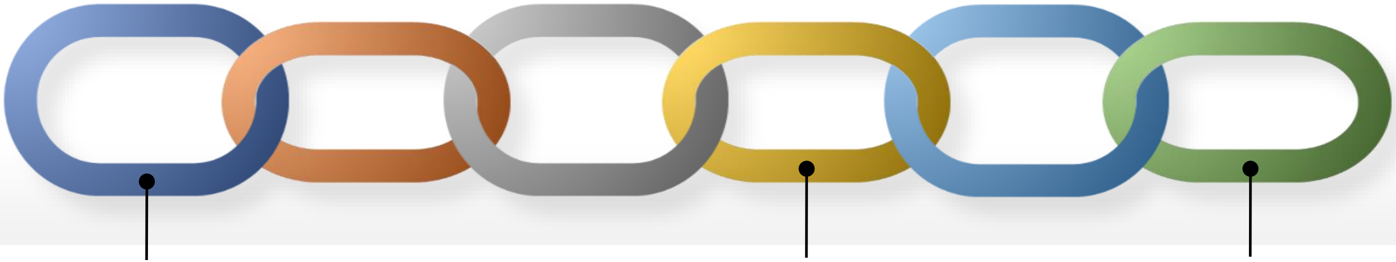
From Permissioned to Permissionless: is it possible at all?

In the permissionless setting the above approach fails spectacularly: nothing is certified, governance is open, the set of possible leaders is unknown and dynamic.

- Who is the next leader?
- If multiple possible leaders propose conflicting blocks, which one we pick?
- Honest majority does not make sense without a PKI, through sybil attacks the adversary can be always in power.

Natural intuition: if players are anonymous then Consensus is impossible

- A basic cloning technique (known as Sybil Attack) allows the adversary to easily reach a dishonest majority
- This is a limitation of the «one person ➔ one vote» approach in the permissioned setting



From Permissioned to Permissionless: is it possible at all?

In the permissionless setting the above approach fails spectacularly: nothing is certified, governance is open, the set of possible leaders is unknown and dynamic.

- Who is the next leader?
- If multiple possible leaders propose conflicting blocks, which one we pick?
- Honest majority does not make sense without a PKI, through sybil attacks the adversary can be always in power.

No much progress to solve this (seemingly unfeasible) problem until the breakthrough of Nakamoto [Nak08], with cryptographic tools and spectacular ideas.

Collision-Resistant Hash Functions

A function $H:\{0,1\}^m \dashrightarrow \{0,1\}^n$ is a CRHF if:

1) $m>n$

2) it is hard to find two different inputs x,y such that $H(x)=H(y)$

despite the obvious existence of many of such "collisions"

Collision-Resistant Hash Functions

A standard CRHFs currently used in the real world is SHA256

SHA256: $\{0,1\}^* \rightarrow \{0,1\}^{256}$

Parameterized Puzzle

Starting with a random rand

Parameterized Puzzle

Starting with a random rand

we can define a small set *Z, and the size of Z is a difficulty parameter*

Parameterized Puzzle

Starting with a random rand

we can define a small set *Z*, *and the size of Z is a difficulty parameter*

The puzzle consists in finding x such that H(rand || x) ∈ Z.
Verification is fast!

Parameterized Puzzle

Starting with a random rand

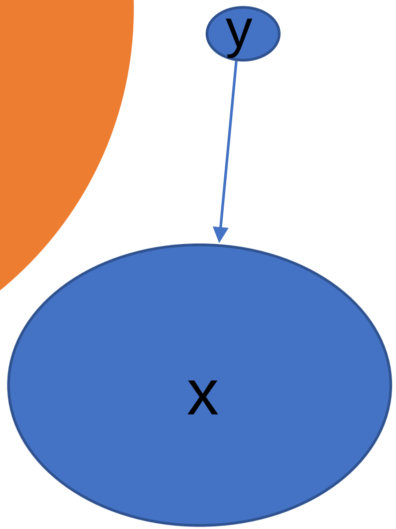
we can define a small set *Z, and the size of Z is a difficulty parameter*

The puzzle consists in finding x such that

H(rand || x) ∈ Z. **Verification is fast!**

The heuristic assumption is that H can only be queried to learn an output (i.e., it is a random oracle), and thus the only solving **strategy** consists of **trying random values for *x*.**

Hash Pointer



H(x) = y can be interpreted as

y is a pointer to x

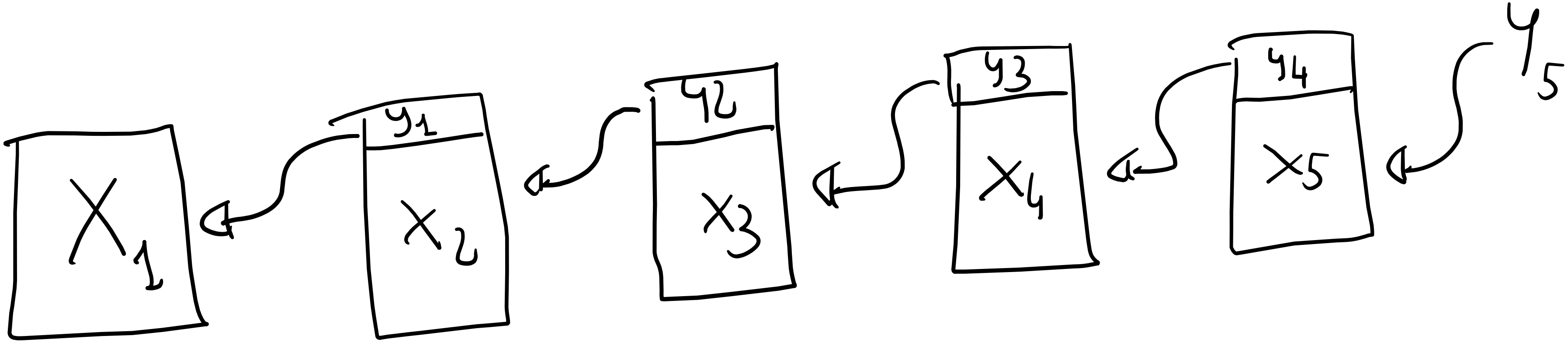
The intuition is that

x is somewhere, and it's big, therefore

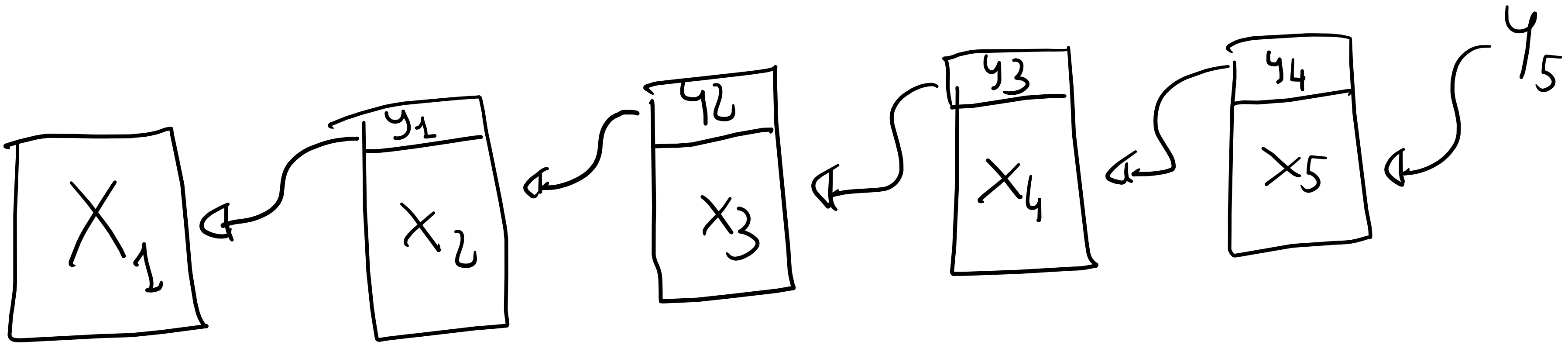
y can be somewhere else

and to understand what y represents you should follow the pointer

Having pointers we can then build a list



Having pointers we can then build a list



Changing a bit of x_j (with any $1<=j<=5$) invalidates all next pointers

y_5 represents $x_1,\ldots,x_5$ when $y_i=H(y_{i-1} \mid\mid x_i)$

Nakamoto's challenging permissionless setting

Peer-to-peer network:

- governance and use transactions are open to anyone (anonymously)

Issues in P2P networks:

- nodes can be offline
- nodes can misbehave
- channel is unreliable
- no common clock

The main problem: Consensus

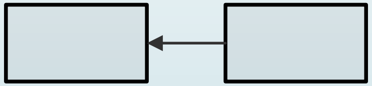
- strong coordination is required to have a common view of what happened in the past
- strong coordination is required to decide what to do next
- this strong coordination must be done in the above fragile P2P anonymous setting

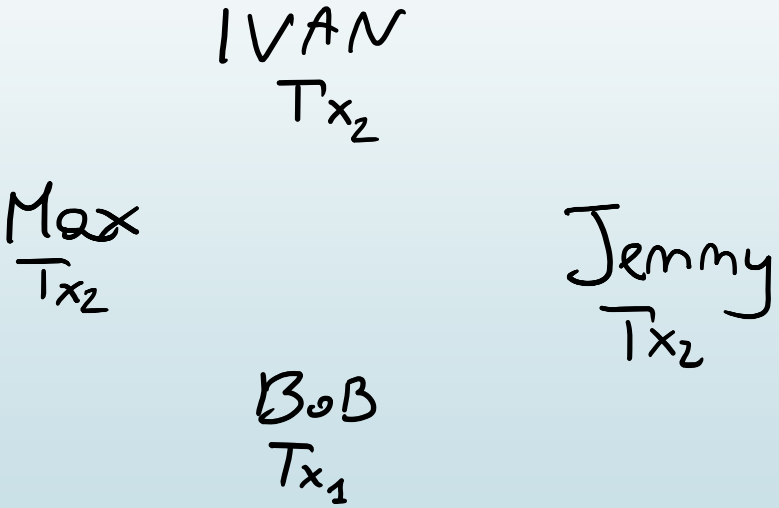
Requirements for Consensus

All correct nodes obtain in output the same valid value

Every valid transaction should eventually be accepted

From known blocks to the next block





Note: we don’t need to opt for the most common proposal

The main point is to agree

Jenny
IVAN
Max
BoB

If honest nodes will sometimes manage to add blocks then liveness will be satisfied

The breakthrough of Nakamoto

from one person → one vote
to one computation → one ticket (a scratch card)

The breakthrough of Nakamoto

from one person ➔ one vote
to one computation ➔ one ticket (a scratch card)

the computation is an attempt to solve a puzzle

find x such that H(x | A | B') has first 70 bits = 0

(x is the solution to the puzzle, A is the previous Block, B' is the new block with just the solution of the puzzle missing and **can't be changed** after the fact)

The breakthrough of Nakamoto

from one person ➔ one vote
to one computation ➔ one ticket (a scratch card)

the computation is an attempt to solve a puzzle

find x such that H(x | A | B') has first 70 bits = 0

(x is the solution to the puzzle, A is the previous Block, B' is the new block with just the solution of the puzzle missing and **can't be changed** after the fact)

rewards attract honest computational power defeating the sybil attack

The breakthrough of Nakamoto

From one person ➔ one vote
to one computation ➔ one ticket (a scratch card).

No reason to talk to others while scratching cards, just make some computations (i.e., proofs of work) and if you win the lottery just announce it.

By giving incentives you get also a huge honest computational power that justifies the main trust assumption: adversary has less than half of the global computational power (i.e., the hash rate of the adversary must be lower than the hash rate of the “correct” nodes)

Key Steps in Nakamoto's Consensus

Transactions are broadcasted

Each node prepares a block and repeatedly scratches cards

The winner announces the new block

Everyone agrees if the block is well formed

The hash of the added block will be part of the next block

If there are more winners the rule is to stick with the longest chain and in case of tie, with the discovered first

It is crucial to start with a public setup, a genesis block that has not been decided (or even seen in advance) by an adversary

We can check…

https://www.blockchain.com/btc/block/0

We can check...

Block 0

USD | BTC

This block was mined on January 03, 2009 at 7:15 PM GMT+1 by Unknown. It currently has 702,122 confirmations on the Bitcoin blockchain.

The miner(s) of this block earned a total reward of 50.00000000 BTC ($2,125,988.50). The reward consisted of a base reward of 50.00000000 BTC ($2,125,988.50) with an additional 0.00000000 BTC ($0.00) reward paid as fees of the 1 transactions which were included in the block. The Block rewards, also known as the Coinbase reward, were sent to this address.

A total of 0.00000000 BTC ($0.00) were sent in the block with the average transaction being 0.00000000 BTC ($0.00). Learn more about how blocks work.

Hash	000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
Confirmations	702,122
Timestamp	2009-01-03 19:15
Height	0
Miner	Unknown
Number of Transactions	1
Difficulty	1.00
Merkle root	4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b
Version	0x1
Bits	486,604,799
Weight	1,140 WU
Size	285 bytes
Nonce	2,083,236,893
Transaction Volume	0.00000000 BTC
Block Reward	50.00000000 BTC
Fee Reward	0.00000000 BTC

Block Transactions

Fee 0.00000000 BTC
(0.000 sat/B - 0.000 sat/WU - 204 bytes) 50.00000000 BTC

Hash 4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afded... 2009-01-03 19:15

COINBASE (Newly Generated Coins) → 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa 50.00000000 BTC

60

We can check...

Fee 0.00000000 BTC
(0.000 sat/B - 0.000 sat/WU - 204 bytes)

50.00000000 BTC

Hash 4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7af...

2009-01-03 19:15

COINBASE (Newly Generated Coins) → 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa 50.00000000 BTC

This transaction was first broadcast to the Bitcoin network on January 03, 2009 at 7:15 PM GMT+1. The transaction currently has 702,122 confirmations on the network. At the time of this transaction, 50.00000000 BTC was sent with a value of $0.00. The current value of this transaction is now $2,125,988.50. Learn more about how transactions work.

Details

Hash	4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b
Status	Confirmed
Received Time	2009-01-03 19:15
Size	204 bytes
Weight	816
Included in Block	0
Confirmations	702,122
Total Input	0.00000000 BTC
Total Output	50.00000000 BTC
Fees	0.00000000 BTC
Fee per byte	0.000 sat/B
Fee per vbyte	N/A
Fee per weight unit	0.000 sat/WU
Value when transacted	$0.00

Inputs

HEX ASM

Index 0

Details Output

Address

Value N/A

Pkscript N/A

Sigscript ffff001d
OP_4
5468652054696d65732030332f4a616e2f32303039204368616e63656c6c6f72206f6e206272696e6b206f66207365636f6e64206261696c6f757420666f72206
2616e6b73

We can check…

https://www.blockchain.com/btc/block/0

echo
5468652054696d65732030332f4a616e2f32303039204368616e63656c6c6f72206f6e
206272696e6b206f66207365636f6e64206261696c6f757420666f722062616e6b73 |
xxd -r –p

Or use

https://string-functions.com/hex-string.aspx

THE TIMES

£1.50

Eat Out from £5

More than 900 great restaurants, including four **Gordon Ramsay** favourites from £15

Israel prepares to send tanks and troops into Gaza

Chancellor on brink of second bailout for banks

Billions may be needed as lending squeeze tightens

99p

Michael Sheen Frost, Nixon and me

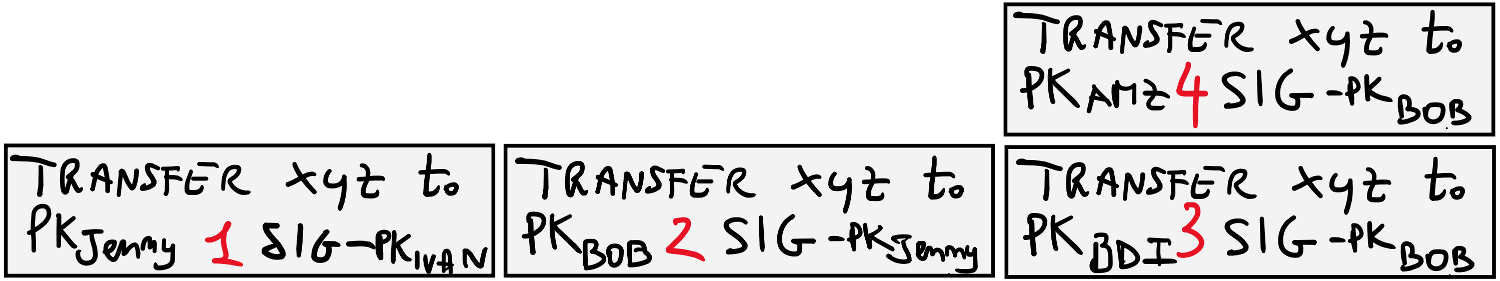
Working mums So that's how she does it

Detox in style The best spas on the planet

Salmon Rushdie I Won't Marry Again

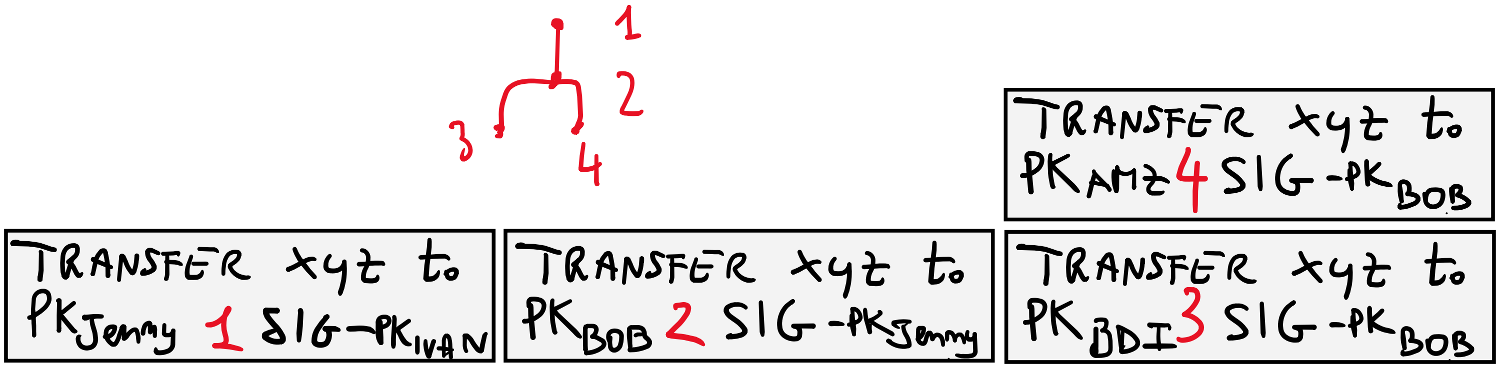
Giant Killing? Guide to the FA Cup Third Round

Forks (and double spending)



If (1,2,3) is announced much before (1,2,4) then clearly everyone will stay with (1,2,3)

Forks (and double spending)



If (1,2,3) and (1,2,4) are announced almost at the same time, then there is a "fork", but most likely one of the two will grow more quickly .

Takeaway: the last blocks are unreliable, commonly in Bitcoin 6 confirmations (i.e., a chain extended with 5 more blocks) are required before considering a transaction finalized in the blockchain.

Note: in general forks can happen and are bad also when everyone is honest (e.g., bids in auctions). These issues must be known to whoever builds applications.

Recall: The Partially Synchronous Model

Main idea: we assume in general the synchronous model but knowing that sometimes for a time windows of unknown length, the synchronous model can fail, still we want to make sure that some properties (i.e., consistency or liveness) are preserved.

...

Note: Nakamoto showed the power of targeting liveness instead of consistency when things go wrong (i.e., in the presence of forks).

In Nakamoto's consensus, during a fork there is no consistency, but valid transactions are (temporary) added (potentially in all branches).

66

PoW in Bitcoin: impressive waste of resources

$>2^{70}$ hashes are generated within 10 minutes to add a new block

The difficulty is adjusted automatically every two weeks (the goal is to have 10 minutes on average)

Nakamoto Consensus from Proofs of Work

it achieves consistency and liveness with the caveat that a transaction can be considered confirmed with high probability only when becoming deeper in the chain

the finality parameter is unspecified and thus up to the user

blocks should not be added too frequently compared to network delays to limit the negative impact of (even honest) forks

Nakamoto Consensus from Proofs of Work

[PSS17] proved that the honest majority of computational power suffices in the synchronous model with a bounded delay (see also [GKL15] for the synchronous model)

Notice that unlike in the permissioned setting, results in the synchronous model do not necessarily hold with bounded delay (e.g., the adversary can exploit delays to gain some advantages with proofs of work)

In the partially synchronous model consistency fails [PSS17]. Result of [LPR20,LPR21] show that this is essentially inherent for PoW-based consensus.

Nakamoto Consensus from Proofs of Work

Practically validated (e.g., Bitcoin, Ethereum*,…)

Problem: a PoW naturally wastes a lot of energy/resources

Recap on Innovations in Nakamoto Consensus

- Preferring liveness instead of consistency when there is a choice (longest chain rule)

- Limiting the attack surface of the Byzantine leader by making difficult the generation of conflicting blocks (proofs of work rather than signatures)

- Revisiting the generic definition of efficient adversary (i.e., probabilistic polynomial-time machine) proposing instead the honest majority of computational power

- Introducing incentives to make somewhat irrational any deviation from honest behavior

Limitations in Nakamoto Consensus

- Transactions are considered confirmed after long time and only probabilistically (and if the puzzle is too easy to solve then there are too many forks and security decreases)
- Proofs of work waste energy (electricity and dedicated HW) and this is bad for the environment, there is an additional risk of becoming illegal in some countries (e.g., recent issues with the mixer Tornado Cash), affecting decentralization
- Proofs of work are expensive and thus they require proper incentives, and this can be problematic (not clear how to establish a stable incentive mechanism, where to pick resources, what players could adversarially try to do (e.g., selfish mining [ES14]))
- The cost for energy is not the same everywhere in the world, therefore mining could be convenient in some specific locations only, against decentralization
- Concentration of resources in mining pools might damage decentralization

Outline

- Part 1: Old-School Consensus
  (i.e., Permissioned Blockchains)
- Part 2: Nakamoto Consensus
  (i.e., Permissionless Blockchains with slow finality)
- <u>Part 3: Algorand Consensus</u>
  (i.e., Permissionless Blockchains with fast finality)

How to Get Updates After Being Off-Line?

A typical problem in permissionless blockchains is that nodes are not permanently online and there is no trusted server to contact in order to get updates after an offline period.

How to Get Updates After Being Off-Line?

A typical problem in permissionless blockchains is that nodes are not permanently online and there is no trusted server to contact in order to get updates after an offline period.

How do we solve this problem in the real life? If we are part of a community and for a while we do not participate in its activities, how do we get updates when returning active?

How to Get Updates After Being Off-Line?

A typical problem in permissionless blockchains is that nodes are not permanently online and there is no trusted server to contact in order to get updates after an offline period.

How do we solve this problem in the real life? If we are part of a community and for a while we do not participate in its activities, how do we get updates when returning active?

There is a simple answer. We ask a few members of the community, in particular the ones that have more visibility and better reputation; we make sure that all answers are consistent before believing in them.

How to Get Updates After Being Off-Line?

How do we translate the above natural solution to permissionless blockchain?

How to Get Updates After Being Off-Line?

How do we translate the above natural solution to permissionless blockchain? The visible actors are the ones owning a lot of cryptocurrency, so it is natural to refer to them.

How to Get Updates After Being Off-Line?

How do we translate the above natural solution to permissionless blockchain? The visible actors are the ones owning a lot of cryptocurrency, so it is natural to refer to them.

The above reasoning motivates Proof-of-Stake (PoS) Consensus: the leader that will propose the next block should be selected among prior participants, and stake possession is an objective measure of participation.

How to Get Updates After Being Off-Line?

How do we translate the above natural solution to permissionless blockchain? The visible actors are the ones owning a lot of cryptocurrency, so it is natural to refer to them.

The above reasoning motivates Proof-of-Stake (PoS) Consensus: the leader that will propose the next block should be selected among prior participants, and stake possession is an objective measure of participation.

Interestingly, PoS has been considered only after PoW in permissionless blockchains.

Anything Special in PoS Compared to PoW?

In PoS consensus it is easier to leverage BFT protocols since one can talk about a specific number of parties (stakeholders) and their public identities (i.e., public keys corresponding to their stake).

Anything Special in PoS Compared to PoW?

In PoS consensus it is easier to leverage BFT protocols since one can talk about a specific number of parties (stakeholders) and their public identities (i.e., public keys corresponding to their stake).

Assumption of honest majority in terms of stake possession could be considered more realistic than dedicated hardware possession.

Anything Special in PoS Compared to PoW?

In PoS consensus it is easier to leverage BFT protocols since one can talk about a specific number of parties (stakeholders) and their public identities (i.e., public keys corresponding to their stake).

Assumption of honest majority in terms of stake possession could be considered more realistic than dedicated hardware possession.

In PoS the leader proposing a block can not be anonymous unlike in PoW. This might introduce risks of coercion (e.g., a criminal announces that whoever adds transactions from a given public key will be attacked). There are some proposals for "anonymous" PoS but they are far from being practical.

PoS: Consensus

The stake is naturally represented by a public key PK and an amount of cryptocurrency n

PoS: Consensus

The stake is naturally represented by a public key PK and an amount of cryptocurrency n

The paradigm of running a lottery requires to have one ticket for every unit of cryptocurrency

- If the adversary is not very rich, the honest majority will win the lottery more often than the adversary

PoS: Consensus

The stake is naturally represented by a public key PK and an amount of cryptocurrency n

The paradigm of running a lottery requires to have one ticket for every unit of cryptocurrency

- If the adversary is not very rich, the honest majority will win the lottery more often than the adversary

How do we run a lottery?

- Ideally a scratch card lottery

Scratch cards using stake, how?

- Ideally, if I have n units of cryptocurrency I should be able to run
  - Eval(…,1)=…
  - Eval(…,2)=…
  - …
  - Eval(…,n)=…

Scratch cards using stake, how?

- Ideally, if I have n units of cryptocurrency I should be able to run
  - Eval(…,1)=…
  - Eval(…,2)=…
  - …
  - Eval(…,n)=…
- It is useful that Eval be deterministic (otherwise everyone will have infinite attempts)

Scratch cards using stake, how?

- Ideally, if I have n units of cryptocurrency I should be able to run
  - Eval(…,1)=…
  - Eval(…,2)=…
  - …
  - Eval(…,n)=…
- It is useful that Eval be deterministic (otherwise everyone will have infinite attempts)
- It is useful that the outputs of Eval look random (this makes easier to design a fair lottery)

Scratch cards using stake, how?

- For simplicity, if I have n units of cryptocurrency I should be able to run
  - Eval(…,1)=…
  - Eval(…,2)=…
  - …
  - Eval(…,n)=…
- It is useful that Eval be deterministic (otherwise everyone will have infinite attempts)
- It is useful that the outputs of Eval look random (otherwise the lottery could be unfair)
- It is useful that Eval can be run only by stakeholders and that the result be verifiable by everyone

Verifiable Random Functions [MRV99]

- GenKey(keylength)➔ (PK,SK)
- Eval(SK, i || prev_block) ➔ 010101101010101101011110=R
- GenProof(SK, i || prev_block) ➔ 1111010101101010101010101=PROOF
- Verify(PK, i || prev_block, R,PROOF)=1

- All algorithms should be fast (it is ok if GenKey is a bit slow)
- R should look random (i.e., on new inputs every single bit could equally be 0 or 1)
- It is hard to produce a fake key PK so that Verify can be equal to 1 with both (R,PROOF) and (R',PROOF') with R'<>R

Verifiable Random Functions Trivial construction with ROs and Unique Signatures

- GenKey(keylength)➔ (PK,SK)
- Eval(SK, i || prev_block) ➔ 0101011010101011010111110=R
- GenProof(SK, i || prev_block) ➔ 11110101011010101010101=PROOF
- Verify(PK, i || prev_block, R,PROOF)=1

- Let (GenKeyU,SigU,VerU) be a Unique* Signature Scheme (i.e., signatures are deterministic)
- Set GenKey=GenKeyU
- Eval(…) will simply be H(SigU(…)) where H is a random oracle
- GenProof(…) will simply be SigU(…)
- Verify(…) will run VerU on PROOF, and will check that R=H(PROOF)

*additional properties are required but for simplicity we omit them

PoS: state of affairs

- Much greener than PoW
- Above leader selection + longest chain rule is used in Cardano still with slow finality
- Above leader selection + committee selection + BFT is used in Algorand with fast finality
- Some issues not applicable to PoW: Nothing at stake attack/Long range attack
- Liveness issue: I'm a stake owner, small amounts as many others, I like to play with some smart contracts, should I always be online???
- Adaptive corruption: there might be room for a winner to "sell" the content of the block that will be added

Algorand: a (Pure)PoS Blockchain [CM19, CGMV18, GHMVZ17]

- VRF is used to select a block proposer (there can be more than one of course, and there is an associated priority) and to select a committee (hundreds of members); this selection is referred as cryptographic sortition
- A BFT protocol is executed by the committee to approve the proposed block, the initial input is the block with highest priority
- Liveness requires attention since without large participation to the consensus there will be no block created (similarly in case of weak synchronousity), this is because the BA will have too few participants and the required threshold of votes for a block will not be reached
- Adversary must be below 1/3 of the stake
- Famous cryptographers (and more) are part of the team started by the Turing award Silvio Micali

Interesting Features of Algorand [CM19, CGMV18, GHMVZ17]

- Consistency is maintained even in case of long asynchronous periods, as long as they are followed by (even shorter) synchronous periods (partially synchronous model). Probability of a fork < 10^{-18}. In the current implementation a block is added every few seconds. Liveness is not guaranteed during asynchronous periods.

Interesting Features of Algorand [CM19, CGMV18, GHMVZ17]

- Consistency is maintained even in case of long asynchronous periods, as long as they are followed by (even shorter) synchronous periods (partially synchronous model). Probability of a fork $< 10^{-18}$. In the current implementation a block is added every few seconds. Liveness is not guaranteed during asynchronous periods.
- The Adversary can corrupt dynamically (remaining under the 1/3 threshold), but why can't she just corrupt the committee?

Interesting Features of Algorand [CM19, CGMV18, GHMVZ17]

- Consistency is maintained even in case of long asynchronous periods, as long as they are followed by (even shorter) synchronous periods (partially synchronous model). Probability of a fork < 10^{-18}. In the current implementation a block is added every few seconds. Liveness is not guaranteed during asynchronous periods.

- The Adversary can corrupt dynamically (remaining under the 1/3 threshold), but why can't she just corrupt the committee?

- Each step of the BFT protocol is played by a different group of committee members, and this property is called player replaceability. It makes adaptive attacks less effective. Indeed, the identity of a committee member is known only when she sends her only message, therefore, too late to be corrupted. These steps are repeated until agreement is reached.

Interesting Features of Algorand [CM19, CGMV18, GHMVZ17]

- Consistency is maintained even in case of long asynchronous periods, as long as they are followed by (even shorter) synchronous periods (partially synchronous model). Probability of a fork < 10^{-18}. In the current implementation a block is added every few seconds. Liveness is not guaranteed during asynchronous periods.
- The Adversary can corrupt dynamically (remaining under the 1/3 threshold), but why can't she just corrupt the committee?
- Each step of the BFT protocol is played by a different group of committee members, and this property is called player replaceability. It makes adaptive attacks less effective. Indeed, the identity of a committee member is known only when she sends her only message, therefore, too late to be corrupted. These steps are repeated until agreement is reached.
- It's a pure proof of stake: all the stake is considered part of the cryptographic sortition, not just the one of some delegated people, and not just the one that somebody is willing to freeze; so, participation is easy (no dedicated hardware, no risks, no discrimination).

Interesting Features of Algorand [CM19, CGMV18, GHMVZ17]

- Consistency is maintained even in case of long asynchronous periods, as long as they are followed by (even shorter) synchronous periods (partially synchronous model). Probability of a fork < 10^{-18}. In the current implementation a block is added every few seconds. Liveness is not guaranteed during asynchronous periods.

- The Adversary can corrupt dynamically (remaining under the 1/3 threshold), but why can't she just corrupt the committee?

- Each step of the BFT protocol is played by a different group of committee members, and this property is called player replaceability. It makes adaptive attacks less effective. Indeed, the identity of a committee member is known only when she sends her only message, therefore, too late to be corrupted. These steps are repeated until agreement is reached.

- It's a pure proof of stake: all the stake is considered part of the cryptographic sortition, not just the one of some delegated people, and not just the one that somebody is willing to freeze; so, participation is easy (no dedicated hardware, no risks, no discrimination).

- Algorand consensus achieves decentralization, security, scalability (i.e., it works efficiently in the presence of a huge number of players): it nicely solved the blockchain Trilemma when looking at the consensus layer.

Interesting Features of Algorand [CM19, CGMV18, GHMVZ17]

- Consistency is maintained even in case of long asynchronous periods, as long as they are followed by (even shorter) synchronous periods (partially synchronous model). Probability of a fork < 10^{-18}. In the current implementation a block is added every few seconds. Liveness is not guaranteed during asynchronous periods.
- The Adversary can corrupt dynamically (remaining under the 1/3 threshold), but why can't she just corrupt the committee?
- Each step of the BFT protocol is played by a different group of committee members, and this property is called player replaceability. It makes adaptive attacks less effective. Indeed, the identity of a committee member is known only when she sends her only message, therefore, too late to be corrupted. These steps are repeated until agreement is reached.
- It's a pure proof of stake: all the stake is considered part of the cryptographic sortition, not just the one of some delegated people, and not just the one that somebody is willing to freeze; so, participation is easy (no dedicated hardware, no risks, no discrimination).
- Algorand consensus achieves decentralization, security, scalability (i.e., it works efficiently in the presence of a huge number of players): it nicely solved the blockchain Trilemma when looking at the consensus layer.
- The fact that in general transactions could be expensive to run and verify remains a scalability issue that should be addressed by the scalability layer.

Are Stakeholders Really Required to Continuously Use Their Precious Secret Keys?

- A stakeholder can generate a temporary participation key that can be used to sign a batch (millions) of ephemeral keys, to then delete the participation key. The participation key must be associated to the account (i.e., to a given stakeholder) with a transaction and from that moment the stake participate during consensus.
- In this way the spending secret key can be in cold storage
- Each ephemeral key is valid for a single round of the consensus protocol and can be deleted after it.
- The fact that participation and ephemeral keys are deleted mitigates attacks where the adversary looks at past blocks.
- These activities are performed by participation nodes, which means by everyone. There exist also relay nodes that are hubs useful to preserve efficiency in the communication among nodes, however they are centralized and there is a goal to move towards decentralized/permissionless relay nodes.

References

- [Lam78] - Lamport: Time, Clocks and the Ordering of Events in a Distributed System
- [PSL80] - Pease, Shostak, Lamport: Reaching Agreement in the Presence of Faults
- [LSP82] - Lamport, Shostak, Pease: The Byzantine Generals Problem
- [DS83] - Dolev, Strong: Authenticated Algorithms for Byzantine Agreement
- [FLP85] - Fischer, Lynch, Paterson: Impossibility of Distributed Consensus with One Faulty Process
- [DLS88] - Dwork, Lynch, Stockmeyer: Consensus in the Presence of Partial Synchrony
- [Sch90] - Schneider: The state machine approach: A tutorial
- [MRV99] - Micali, Rabin, Vadhan: Verifiable Random Functions
- [Nak08] - Nakamoto: Bitcoin: A Peer-to-Peer Electronic Cash System
- [ES14] - Eyal, Sirer: Majority is not enough: Bitcoin mining is vulnerable

References

- [GKL15] - Garay, Kiayias, Leonardos: The bitcoin backbone protocol: Analysis and applications
- [PSS17] - Pass, Seeman, Shelat: Analysis of the Blockchain Protocol in Asynchronous Networks
- [GHMVZ17] - Gilad, Hemo, Micali, Vlachos, Zeldovich: Algorand: Scaling Byzantine Agreements for Cryptocurrencies
- [CGMV18] - Chen, Gorbunov, Micali, Georgios Vlachos: ALGORAND AGREEMENT Super Fast and Partition Resilient Byzantine Agreement
- [CM19] - Chen, Micali: Algorand: A secure and efficient distributed ledger
- [LPR20] - Lewis-Pye, Roughgarden: Resource Pools and the CAP Theorem
- [Abr21] - Abraham: https://twitter.com/ittaia/status/1452027925084229637
- [LPR21a] - Lewis-Pye, Roughgarden: How Does Blockchain Security Dictate Blockchain Implementation?
- [LPR21b] - Lewis-Pye, Roughgarden: Byzantine Generals in the Permissionless Setting

Thanks!

ivan.visconti@gmail.com