The Blockchain Backbone Model

Presented by Dionysis Zindros

Invented by **Juan Garay**, **Aggelos Kiayias**, **Nikos Leonardos**
Algoholics - National Technical University of Athens, November 2018

What we’ll talk about

- The **blockchain backbone**
- The **mathematical theory** that governs **bitcoin**,
  ethereum, litecoin, monero, bitcoin cash, ethereum classic, zcash, doge
  and hundreds of other cryptocurrencies
  (Originally called the “*bitcoin* backbone”, but really quite generic)
- The **algorithmic parts** of blockchains
- The **consensus layer** of blockchains
- The nature of a blockchain: What a blockchain is
- What **problem** do blockchains solve?
- **Proof-of-work** blockchains only!
  (Not proof-of-stake such as Cardano, Decred)

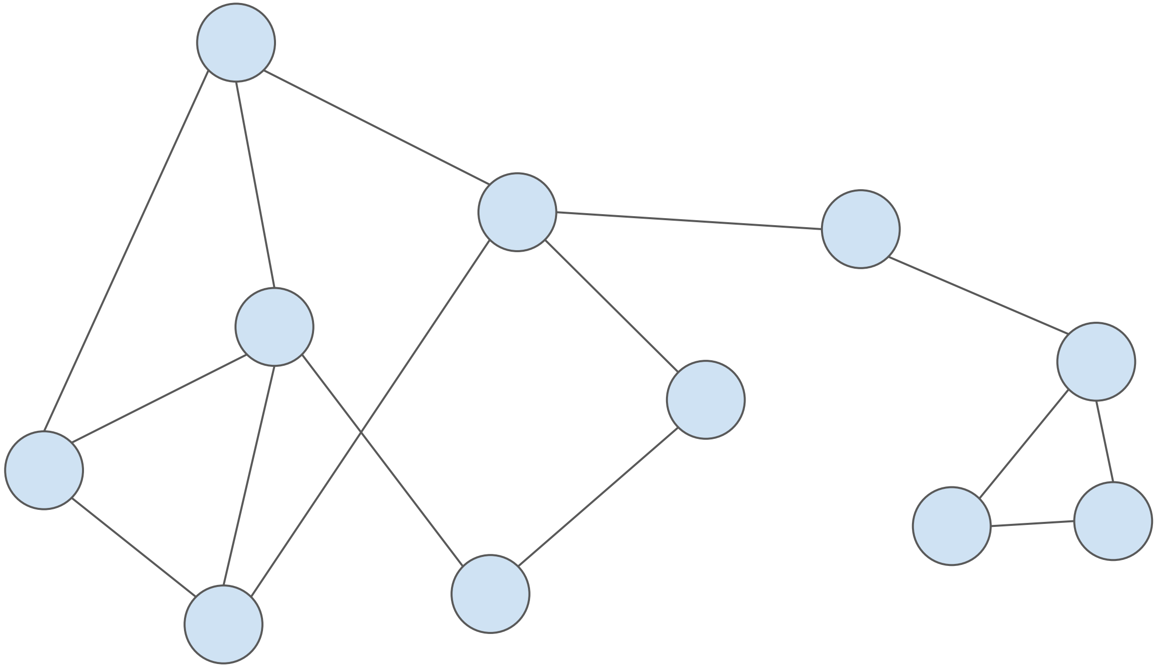
What we **won't** talk about

- Blockchains are a complicated subject
- Many engineering and science problems are open
- We won't talk about **engineering challenges**
- We will only speak about **one model**
- Here we treat the problem in the closed setting of **computer science theory**
- We won't be concerned with **implementation details**
- Blockchains deal with a lot more than the **consensus layer**, including the **application layer**
- We won't speak about **specific cryptocurrencies**
- We will make many **simplifying assumptions**
- We won't **prove** anything today -- I will talk only about definitions

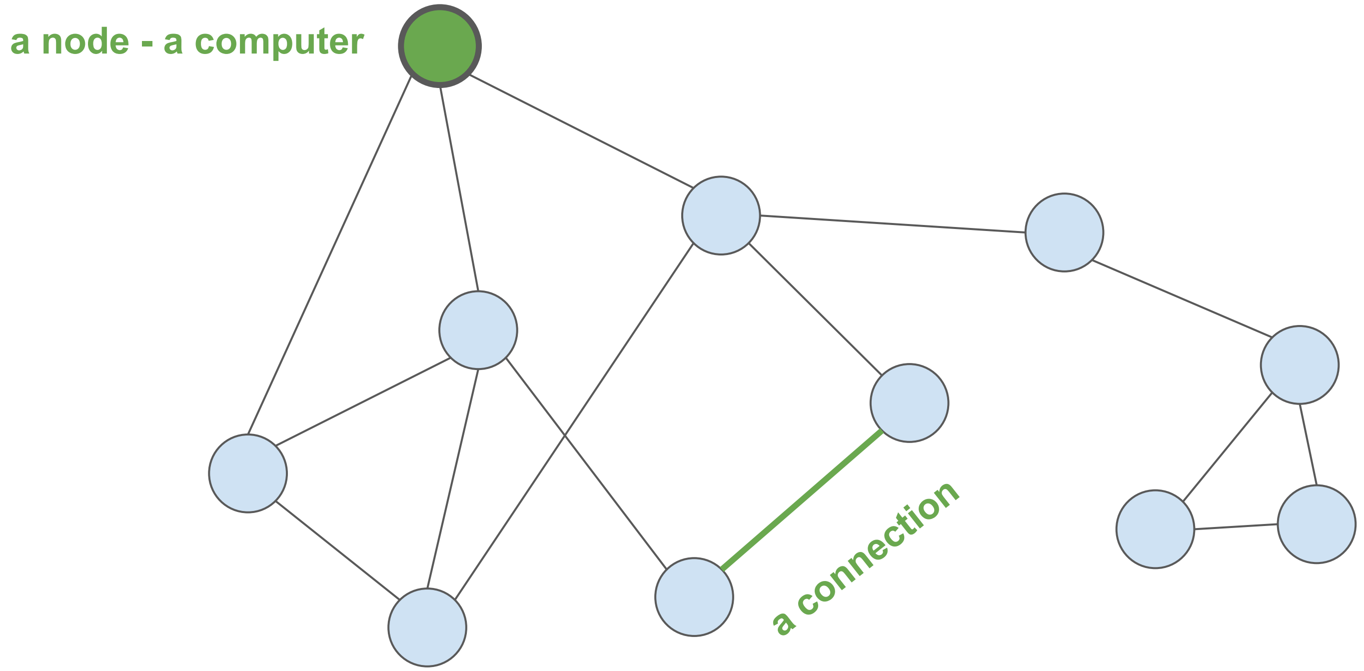
The Blockchain Backbone

- The *first theory* formalizing the security of bitcoin & cryptocurrencies
- Best model we have so far
- It answers the question: **Why is bitcoin secure?**
- This question was not answered in the original Bitcoin paper!
- Invented and published in EUROCRYPT ‘15 by
  Juan Garay, Aggelos Kiayias, and Nikos Leonardos
- Revisited and published in CRYPTO ‘17 by same authors
- One of the most significant works after Nakamoto’s original paper

A peer-to-peer network of money



A peer-to-peer network of money



What’s a peer-to-peer network of money?

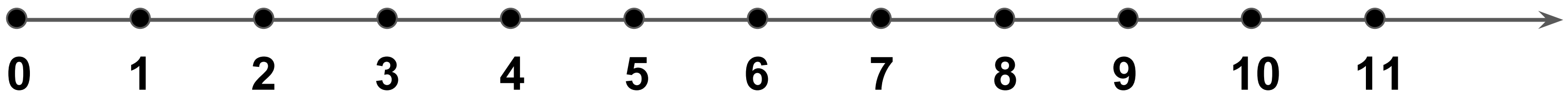
- Every node on the network knows how much money is in **every** *account*
- When someone pays someone else, they generate a *transaction* -- ***tx***
- The tx subtracts money from one account and adds it to another
- tx are *broadcast to everyone*
- By receiving tx, nodes can update their view of others’ accounts
- A tx can only be created by the owner sending the money:
  It needs a digital signature

Most importantly: The network is **decentralized**. There is **no trusted third party**!
(Everything is very easy if we have a trusted bank or PayPal)

Assumption: Time is discrete

We think of time as happening in **discrete rounds**: 1, 2, 3, …, *r*

Information (transactions) appears in rounds

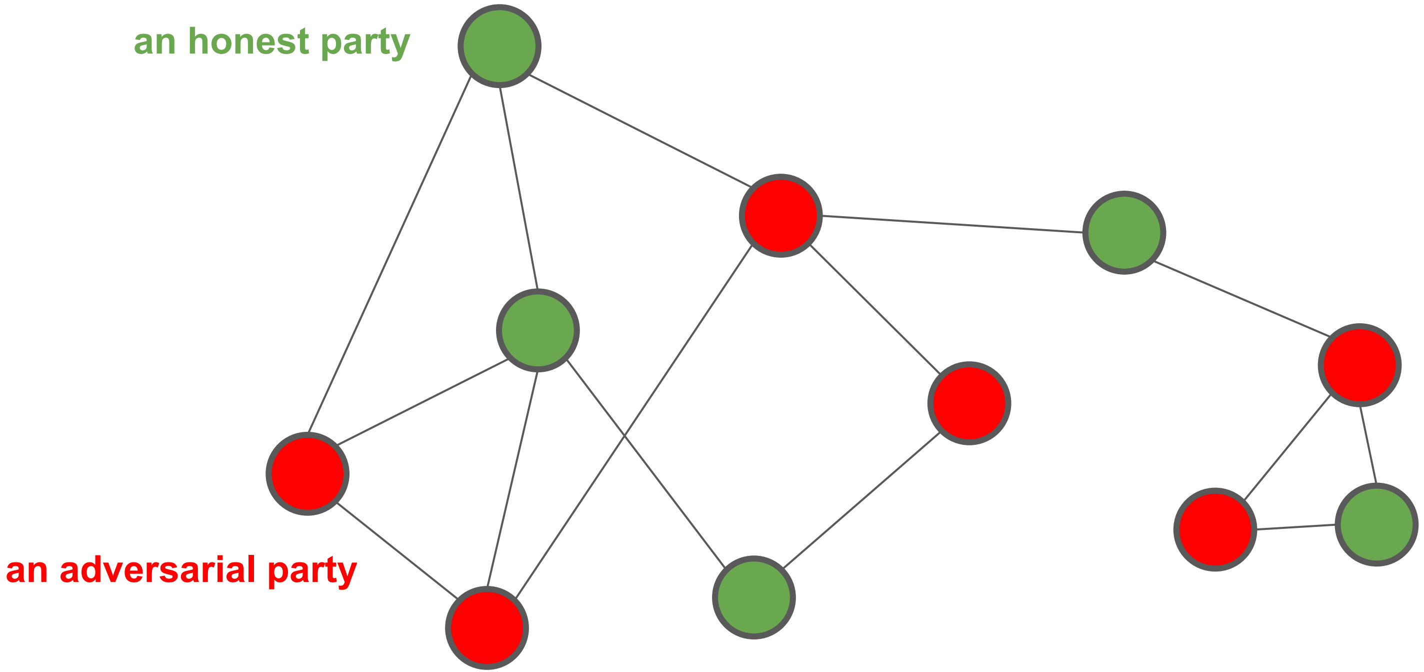


The network is **synchronous**:
A message sent by an honest party during round *r* is received at round *r* + 1 by all other honest parties. One round in practice corresponds to the time needed to traverse the whole network.

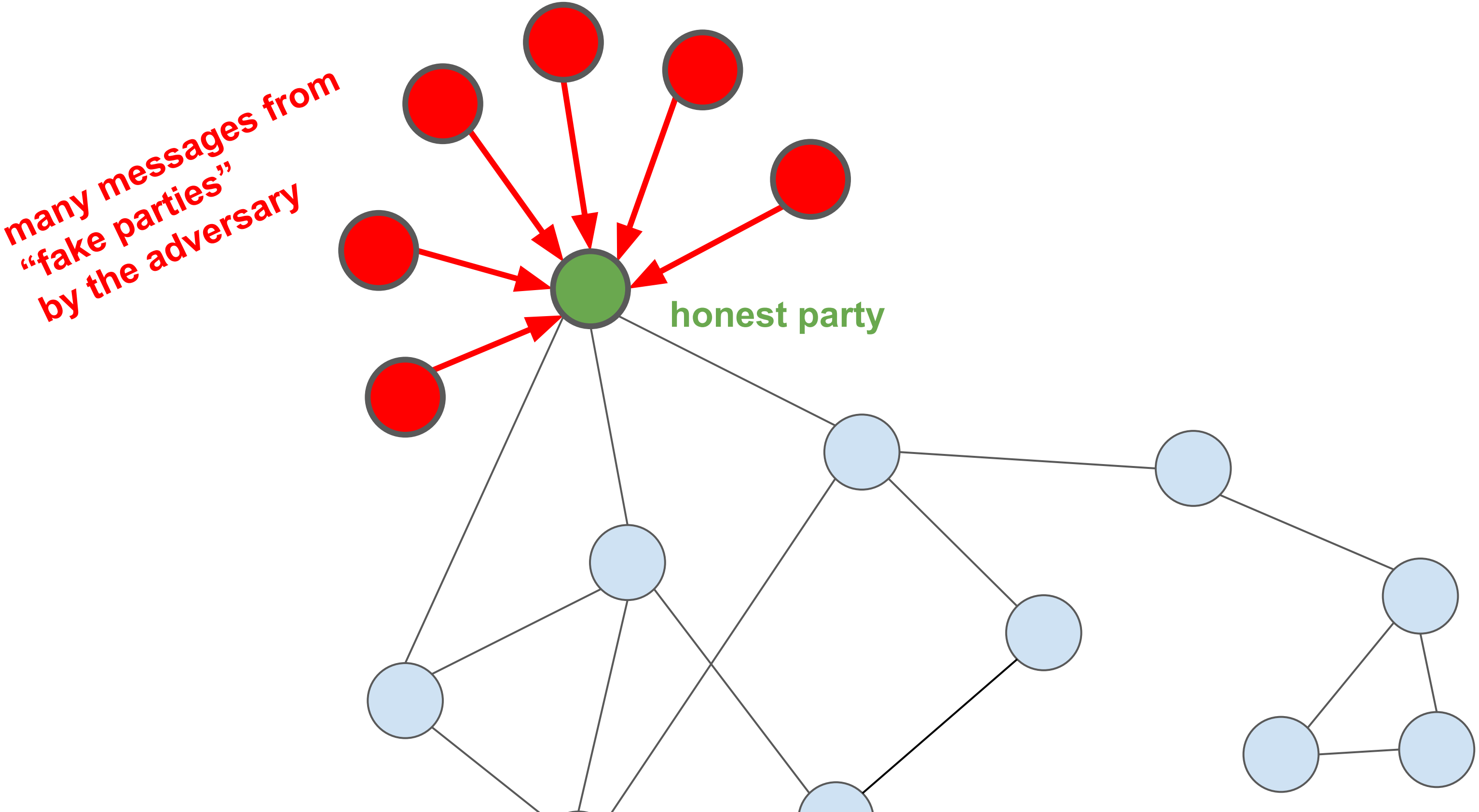
The adversary

- The adversary can be **anything**
- Modelled as an arbitrary probabilistic polynomial-time Turing Machine
- This analysis is powerful:
  We do not say what attacks we have in mind!
- The protocol is secure against **any attack** we may not even think of!
- There is only **one adversary**
- The adversary **controls** multiple parties
- All of them coordinate together to attack the protocol
- Parties controlled by the adversary can communicate arbitrarily *during* a round

Assumption: The adversary controls arbitrary nodes



Assumption: The network is Sybil attackable



The Sybil attack

The Sybil attack captures the fact that the adversary can:

- Fake IP addresses
- Fake e-mail addresses, Facebook accounts, Gmail accounts
- Fake phone numbers

Generally, we cannot rely on any centralized service or network heuristic!
We cannot simply take a vote by counting.

The parties

- There are ***n*** parties in total
- ***t*** < n are controlled by **one** adversary (corrupted)
- ***n - t*** < n are honest
- Each party wishes to publish some *tx* to the rest
- There's new tx coming in by honest parties during various rounds

The consensus problem

- It may take some time for tx to travel on the network
- How can the parties agree **which tx happened first**?
- This question is crucial for security:
  If a tx did/didn't happen, a party may/may not have sufficient money!
- **tx validity depends on past tx**

time $t = 1$

Eve really has 1 BTC

Bob & **Alice** think Eve has 1 BTC

time $t = 2$

Eve spends her 1 BTC to buy coffee from Alice

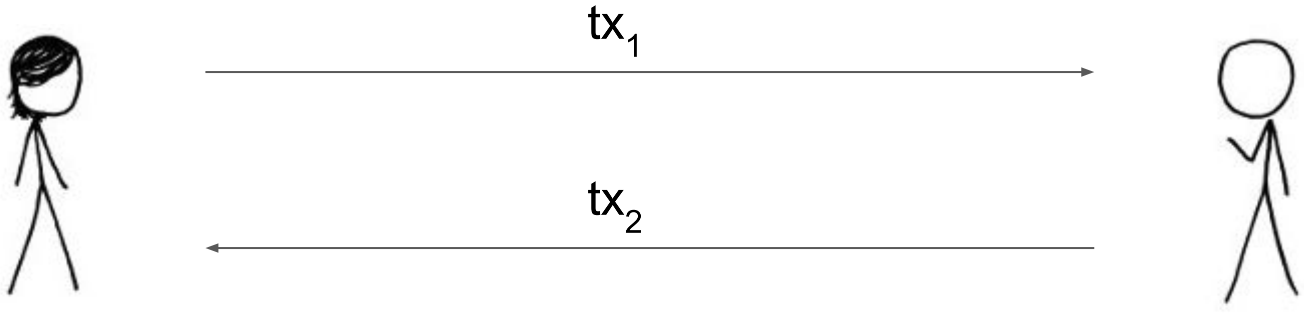
Sends transaction tx_1 to **Alice**

...simultaneously...

time $t = 2$

Eve spends the *same* 1 BTC to pay back ***herself***

Sends transaction tx_2 to **Bob**



Alice and **Bob** exchange the transactions on the network and see a **double spend**!

The two transactions are irreconcilable.
Which one is the right one?

Simple ideas don’t work

**“Oldest transaction is the right one.
Reject the newer double spend!”**

Bob and **Alice** don’t agree on which transaction happened first!

Alice thinks tx_1 happened first. **Bob** thinks tx_2 happened first.

Simple ideas don’t work

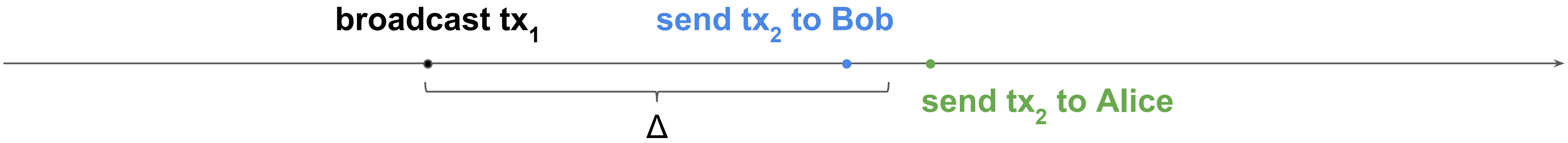
“If you see a double spend, you know it’s a bad actor!
Reject the transaction!”

How long do you wait until you know there has been *no double spend*?

What if **Eve** bought a cup of coffee from **Alice**?
When does **Alice** deliver the coffee?

Simple ideas don’t work

“Wait for some constant time Δ.
If no double spend has occurred within time Δ, accept the transaction.
If a double spend occurs within time Δ, reject both transactions.
If a double spend occurs after time Δ, reject the double spend.”



Eve creates tx_1 and sends it to both **Alice** and **Bob** at time $t = 0$.
Then **Eve** sends tx_2 to Bob at time $t = \Delta - 1$, rendering tx_1 invalid according to **Bob**.
Eve sends tx_2 to **Alice** at time $t = \Delta + 1$, rendering tx_1 valid according to **Alice**.

Alice thinks tx_1 is valid, but **Bob** thinks tx_1 is invalid.

We need a better protocol!

What is the form of such protocol?
We need to define *what code each node will run*

The code must:

- **Injection**: Allow “publishing” a local *tx*
- **Reading**: When locally asked “what tx have happened?”, returns a **ledger**

A *ledger* is an *ordered list of tx*
This is the **local view** of the honest party

Desired properties of a ledger

1. **Liveness**

   If I "inject" a transaction, it will *eventually* appear in the *ledger* of *all honest parties*

2. **Persistence**

   If a transaction appears in a certain position *i* within an honest party's ledger, it will appear *in the same position* for *all* honest parties

Liveness & Persistence

- The solution is easy if we give up one of the properties. **How?**
- Without **liveness**:

  Always return an empty ledger.
  tx injected will *never* appear in any ledger.
  Persistence is trivially satisfied.
- Without **persistence**:

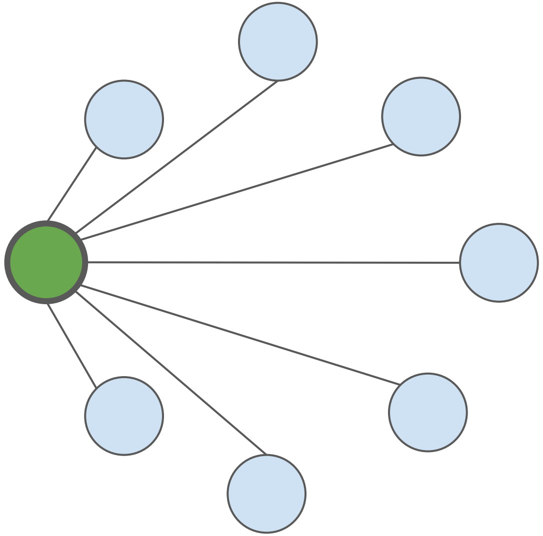
  Append all tx you see on network to ledger immediately.
  Parties won’t agree about order of tx.
  Liveness is trivially satisfied.

Assumption: Total connectivity

Each node is **connected to everyone** else

In particular: **all honest parties are directly connected** to each other

Not true in practice, but realistic model:
All nodes are connected *indirectly*; a message is broadcast, then relayed. Allows us to abstract out the notion of relaying.

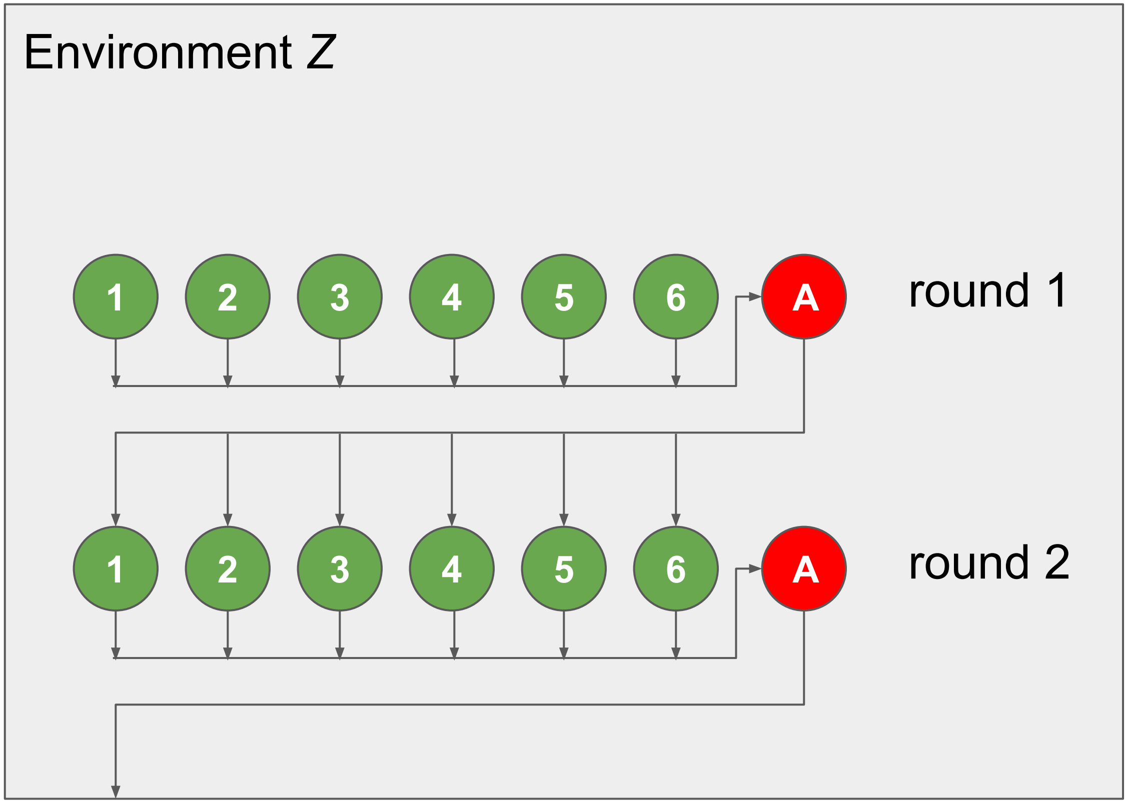


Summary: The network model

1. The network is **synchronous**:
   A message by an honest party at round *r* is delivered to all parties at *r* + 1
2. The network is **reliable**:
   The adversary cannot drop honest messages
3. The network is **Sybil attackable**:
   The adversary can inject arbitrarily many messages
   The adversary can send different messages to different parties
4. Parties **do not have pre-known identities**:
   The adversary can reorder and shuffle network messages

The execution

- Every party is modelled as an Interactive Turing Machine
- Execution is coordinated by a special PPT TM, the *environment Z*
- The environment captures the notion of “rounds” (an iteration of Z)
- For every round, it invokes *each honest party*
  - It gives it network messages from the previous round
  - It receives from it a network message to be delivered in the next round
- At the end of each round, it invokes the adversary (a “*rushing adversary*”)
- The adversary can see all honest party messages, reorder them, or inject fake ones (Sybil attack), but not delete them. The changes may be different for each honest party.
- The new modified messages are delivered to honest parties at the next round
- Parties can keep local state



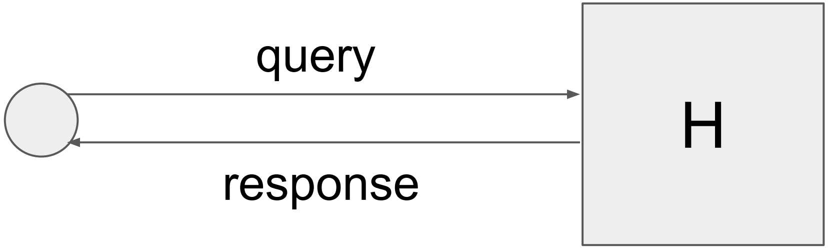
Environment Z
1
2
3
4
5
6
A
round 1
1
2
3
4
5
6
A
round 2

The environment

```
tampered_messages = [ε, ε, …, ε]
for round = 1 to poly(κ)
  for i = 1 to n - t
    next_messages[i] = P[i](tampered_messages[i])
  end for
  tampered_messages = A(next_messages)
  for m in next_messages
    for i = 1 to n - t
      assert(m in tampered_messages[i])
    end for
  end for
end for
```

The random oracle (RO) model

- A truly random function H with output $\{0, 1\}^\kappa$ is chosen before the beginning of the execution
- Any party can query the function
- The function is shared among all honest parties and the adversary
- It gives the *same* output for the same input even if asked by different parties
- We think of it as a *cryptographically secure hash function*



The honest majority assumption

- We try to capture that the adversary has *less computational power* than the honest parties combined
- This is how we will distinguish the honest opinion from the adversarial opinion
- Computational power cannot be Sybil attacked
- We allow ***q*** RO queries per round for each honest party
- We allow the adversary ***tq*** RO queries per round
- We require:

$$\underbrace{t}_{\text{adversarial parties}} \leq \underbrace{(1 - \delta)}_{\text{a small gap}} \underbrace{(n - t)}_{\text{honest parties}}$$

(the honest majority equation)

Summary: Our assumptions

- The network is **synchronous, connected, Sybil-attackable**
- The analysis is in the **random oracle model**
- The **majority** of computational power is **honest at all times**
- **We don't analyze incentives**
- **The number of parties is constant**

Many of these assumptions can be lifted, but the analysis becomes more complex
(e.g. the "constant number of parties" assumption is relaxed in follow-up work)

Proof-of-Work (PoW)

How can we prove that we hold certain computational power?

- Give a *fresh input* to RO
- If the output is sufficiently small, we have (probabilistically) proven that we have expended some of our *q* RO queries
- We can use this mechanism for *voting*
- Let *T* be the bound required, then we want:

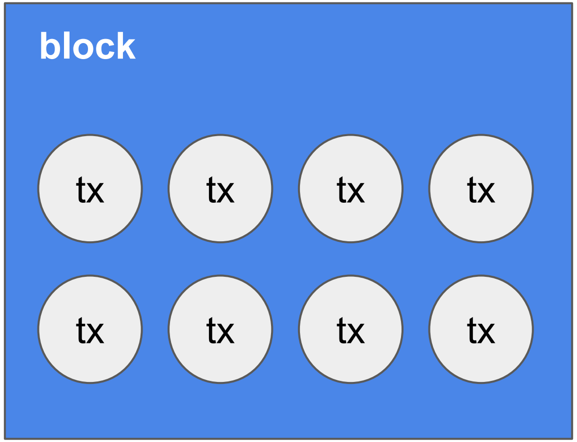
$$\mathbf{H(x) < T}$$

(the proof-of-work equation)

- H(x) and T are both κ-bit strings compared as binary numbers

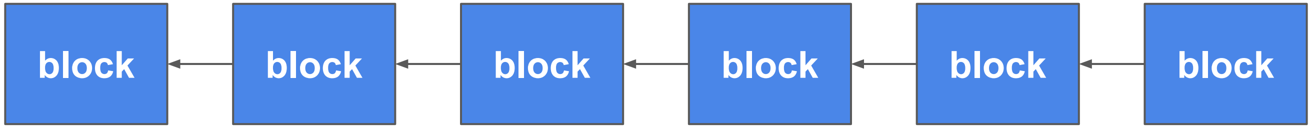
A block

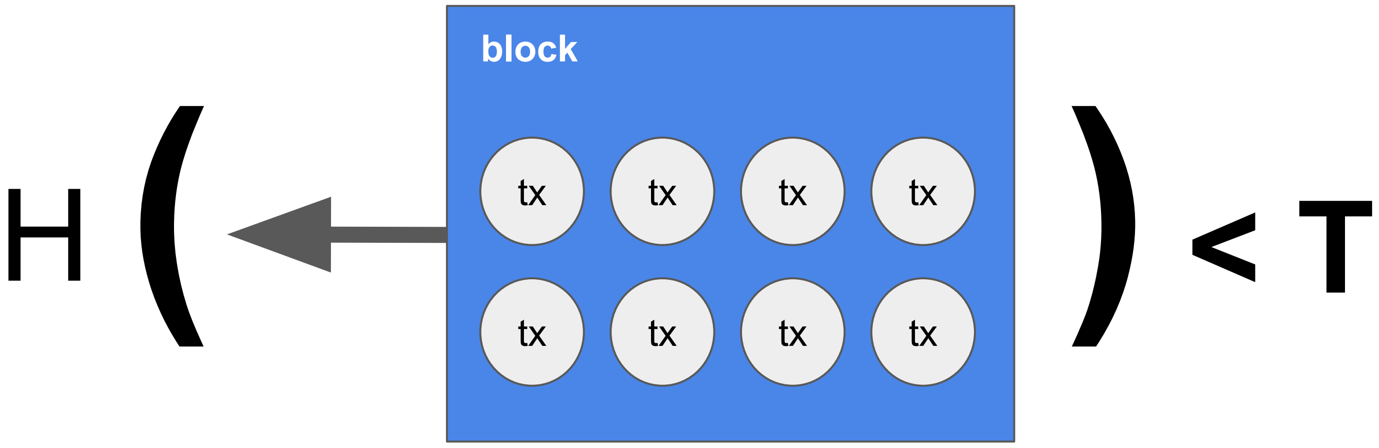
- Contains an ordered sequence of transactions
- The order must be valid (no double spends)
- Contains some proof-of-work, proving computational power has been expedited in creating it: **$H(B) < T$**



A blockchain

- A sequence of blocks
- Each block contains the hash of its parent
- **A block could not have been generated before its parent** (Why?)
- Therefore, blocks are generated in order
- A block buried under sufficient other blocks is **voted on** by computational power




H (
block
tx
tx
tx
tx
tx
tx
tx
tx
) < T

Mining

- Each honest party *adopts* a blockchain *C*
- Not all honest parties necessarily adopt the same chain
  (we would like that, but we cannot ensure it due to the consensus problem)
- During each round, each honest party attempts to extend their adopted chain
- ...by creating a new block *B* on top of *C* such that *CB* is a valid chain
- *B* points to *C*[-1]
- *B* must contain valid PoW. It contains some nonce *ctr* to achieve the PoW.
- *B* contains a vector of tx x that the honest party wishes to inject
  and have not yet been included
- If honest party is successful in finding *B*, it broadcasts *CB* to the network

Algorithm 3 The *proof of work* function, parameterized by q, T and hash functions $H(\cdot), G(\cdot)$. The input is $(x, \mathcal{C})$.

function pow$(x, \mathcal{C})$
    if $\mathcal{C} = \varepsilon$ **then** injection ▷ Determine proof of work instance
        $s \leftarrow 0$
    else
        $\langle s', x', ctr' \rangle \leftarrow \mathrm{head}(\mathcal{C})$
        $s \leftarrow H(ctr', G(s', x'))$
    end if
    $ctr \leftarrow 1$
    $B \leftarrow \varepsilon$
    $h \leftarrow G(s, x)$
    while $(ctr \leq q)$ **do**
        if $(H(ctr, h) < T)$ **then** ▷ This $H(\cdot)$ invocation subject to the q-bound
            $B \leftarrow \langle s, x, ctr \rangle$ PoW equation
            break
        end if
        $ctr \leftarrow ctr + 1$
    end while
    $\mathcal{C} \leftarrow \mathcal{C}B$ ▷ Extend chain
    return $\mathcal{C}$
end function

Validating a chain

- Check that the transactions it contains form a valid sequence
- Check the proof-of-work of each block
- Check that each block points to its parent

Algorithm 1 The *chain validation predicate*, parameterized by q, T, the hash functions $G(\cdot), H(\cdot)$, and the *content validation predicate* $V(\cdot)$. The input is $\mathcal{C}$.

function validate($\mathcal{C}$)
  $b \leftarrow V(\mathbf{x}_{\mathcal{C}})$ validate transaction sequence
  if $b \wedge (\mathcal{C} \neq \varepsilon)$ **then** ▷ The chain is non-empty and meaningful w.r.t. $V(\cdot)$
    $\langle s, x, ctr \rangle \leftarrow \mathrm{head}(\mathcal{C})$
    $s' \leftarrow H(ctr, G(s, x))$
    repeat validate chain ancestry
      $\langle s, x, ctr \rangle \leftarrow \mathrm{head}(\mathcal{C})$
      if $\mathrm{validblock}_q^T(\langle s, x, ctr \rangle) \wedge (H(ctr, G(s, x)) = s')$ **then**
        $s' \leftarrow s$ ▷ Retain hash value
        $\mathcal{C} \leftarrow \mathcal{C}^{\lceil 1}$ validate block PoW ▷ Remove the head from $\mathcal{C}$
      else
        $b \leftarrow$ False
      end if
    until $(\mathcal{C} = \varepsilon) \vee (b = \text{False})$
  end if
  return (b)
end function

The Longest Chain Rule

- Which chain should I adopt?
- The longest chain! (In terms of **number of blocks**, not number of tx!)
- Why? It contains the most accumulated proof-of-work
- This signifies that a lot of computational power is vouching for it
- Hence it will be what the honest parties believe!
  (intuitively -- exact property formulated below)

Algorithm 2 The function that finds the "best" chain, parameterized by function $\max(\cdot)$. The input is $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$.

```
function maxvalid(C_1, ..., C_k)
    temp ← ε
    for i = 1 to k do
        if validate(C_i) then
            temp ← max(C_i, temp)
        end if
    end for
    return temp
end function
```

The blockchain backbone protocol

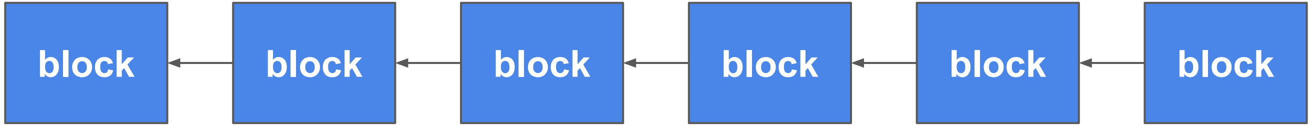
- Maintain a blockchain
- When seeing other blockchains on the network, adopt the longest one
- Mine on top of longest chain
- If mining successful, broadcast mined chain

Algorithm 4 The Bitcoin backbone protocol, parameterized by the *input contribution function* $I(\cdot)$ and the *chain reading function* $R(\cdot)$. At the onset it is assumed "init= True".

if (init) **then**
  $\mathcal{C} \leftarrow \varepsilon$
  $st \leftarrow \varepsilon$
  $round \leftarrow 1$
  $init \leftarrow$ False
else
  $\tilde{\mathcal{C}} \leftarrow \mathsf{maxvalid}(\mathcal{C}$, any chain $\mathcal{C}'$ found in $\textsc{Receive}())$
  if $\textsc{Input}()$ contains $\textsc{Read}$ **then**
    write $R(\tilde{\mathcal{C}})$ to $\textsc{Output}()$ ▷ Produce necessary output before the POW stage.
ledger reading
  end if
  $\langle st, x\rangle \leftarrow I(st, \tilde{\mathcal{C}}, round, \textsc{Input}(), \textsc{Receive}())$ ▷ Determine the x-value.
  $\mathcal{C}_{\mathsf{new}} \leftarrow \mathsf{pow}(x, \tilde{\mathcal{C}})$
injection
  if $\mathcal{C} \neq \mathcal{C}_{\mathsf{new}}$ **then**
    $\mathcal{C} \leftarrow \mathcal{C}_{\mathsf{new}}$
    $\textsc{Diffuse}(\mathcal{C})$ ▷ Broadcast the chain in case of adoption/extension.
  else
    $\textsc{Diffuse}(\perp)$ ▷ Signals the end of the round to the diffuse functionality.
  end if
  $round \leftarrow round + 1$
end if

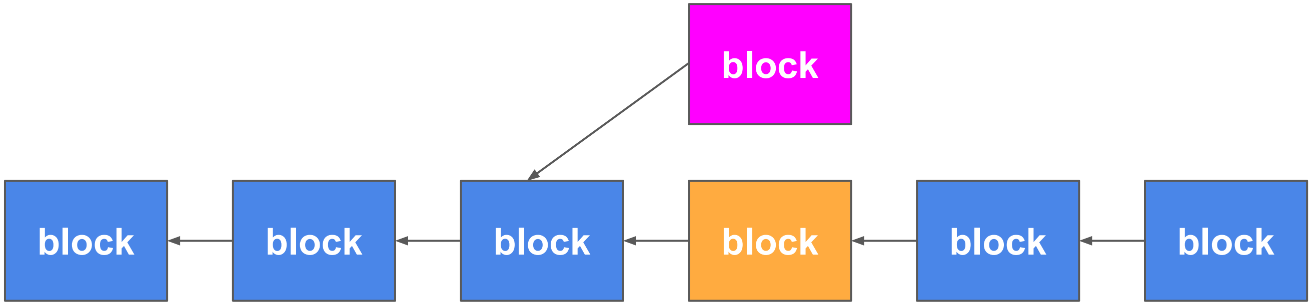
An honest execution

- Consider: What happens if all parties are honest?
- An honest party finds a block, broadcasts it
- The rest of the network adopts the new chain, as it is longer
- Another honest party finds a new block, broadcasts it…
- The chain keeps growing and is the same among all honest parties
- The rate of growth corresponds to the parameters T, q, n



Blockchain forks

- Sometimes two blocks will be mined at the same round by two honest parties
- Any of the two alternatives can be adopted by honest parties
  - Which one is chosen by the adversary, as they control the network!
- The tie will be broken when a next block is found
- Then the longest chain will be unique again
- This is why we want the probability of block finding to be small



Successful rounds

We call a round…

- *successful* if (at least) an **honest** party finds a block
- *uniquely successful* if **exactly one** honest party finds a block

It’s possible that the adversary will also find blocks during successful or uniquely successful rounds.

$$f \stackrel{\text{def}}{=} Pr[\text{ round r is successful }]$$

$$= 1 - (1 - T / 2^{\kappa})^{q(n - t)}$$

$$\cong q(n - t)\, T / 2^{\kappa}$$

Blockchains have three security properties

1. **Chain growth**
   The chain adopted by an honest party grows
2. **Chain quality**
   Large chunks of an honestly adopted chain contain honest blocks
3. **Common prefix**
   All honestly adopted chains share a large common prefix

Theorem: **For all PPT adversaries** *A*, the properties hold with *overwhelming probability* in the security parameter.

Chain growth

Consider an honest party P.
If P has adopted chain C_1 at round r, and C_2 at round r + s, then

$$|C_2| \geq |C_1| + \tau\, s$$

(the chain growth equation)

τ is the **chain velocity**. This property holds for sufficiently large *s*.

Intuition for Chain Growth

Consider a successful round with a block by party P_1.
During a successful round, the chain of all honest parties grows.

- The chain of P_1 grows by the new block
- Consider P_2
  - If P_2 is successful, their chain also grows by *their* new block
  - If P_2 is unsuccessful, their chain grows because they receive P_1's extension
- The adversary cannot harm this due to Longest Chain Rule

Intuition for Chain Growth

- Consider a sequence of s rounds
- Each round has a probability f of being successful
- In expectation, $f\,s$ rounds will be successful
- Hence the chain will grow by approximately $f\,s$ blocks
- Therefore $\tau \cong f$

The full proof is by induction on the number of rounds
and a Chernoff bound with overwhelming probability in s

Chain quality

Consider an honestly adopted chain C.
For any *i*,

C[i: i + ℓ] contains at least μ ℓ honest blocks

μ is the proportion of honest blocks. This property holds for sufficiently large ℓ.

Common prefix

Let P_1, P_2 be two honest parties and consider a round *r* at which their adopted chains are C_1 and C_2. Then $C_1[:-k]$ is a prefix of C_2 and vice versa:

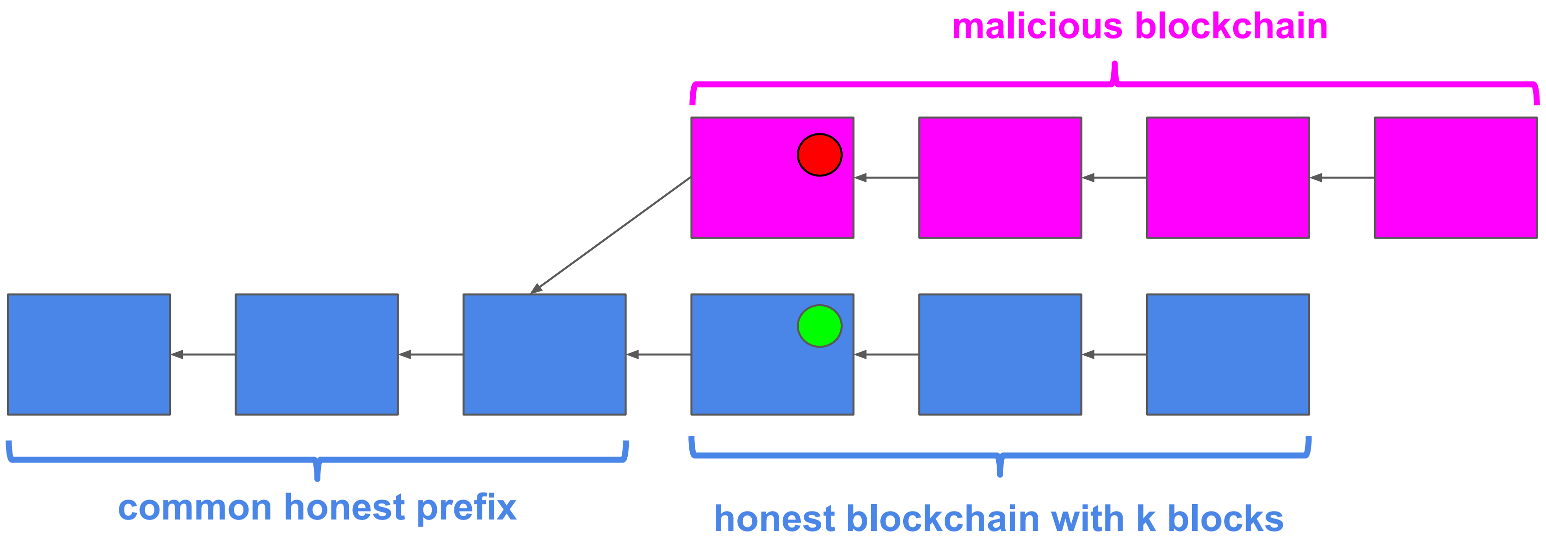
$$C_2[\ :\ |C_1[:-k]|\ ] = C_1[:-k]$$

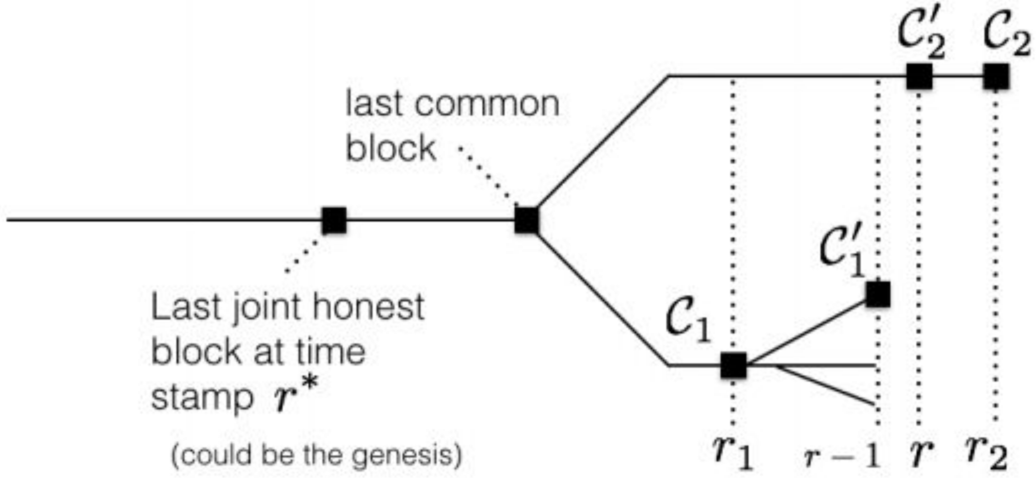
(the common prefix equation)

This property holds for sufficiently large *k*.

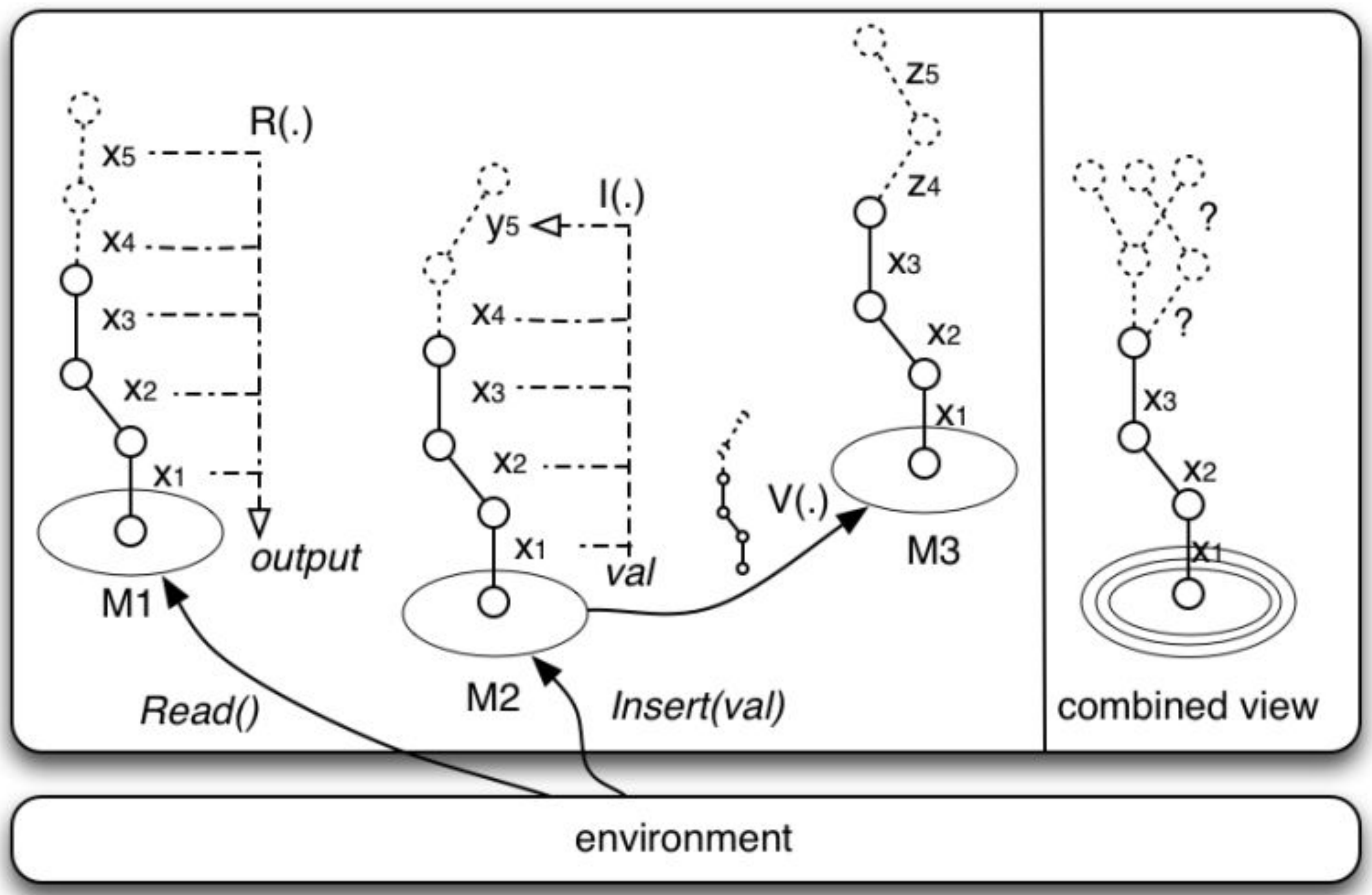
Intuition for Common Prefix

- Consider an adversary who double spends
- They have to extend a chain faster than the honest parties
- This is impossible if the honest chain is large enough





$\mathcal{C}_2'$
$\mathcal{C}_2$
last common block
$\mathcal{C}_1'$
$\mathcal{C}_1$
Last joint honest block at time stamp r^*
(could be the genesis)
r_1
$r-1$
r
r_2



R(.)
x5
x4
x3
x2
x1
output
M1
I(.)
y5
x4
x3
x2
x1
val
M2
V(.)
z5
z4
x3
x2
x1
M3
?
?
x3
x2
x1
combined view
Read()
Insert(val)
environment

Implementing the Ledger functionality

Now that we have a blockchain, it's easy...

- Take C[:-k] and obtain its transactions
- Return that ledger!

Proof of Liveness

Suppose we wish to inject a *tx* starting at round *r*

- By the *Chain Growth* property, the chain will keep growing
- After *s* rounds, there will be a growth by τs new blocks
- For sufficiently large s (“eventually”), we have that $\tau s \mu \geq 1$
- Therefore, after s rounds, an honestly generated block B will be adopted by all honest parties
- Due to *Chain Growth*, B will eventually be buried under *k* blocks
- It will then be reported in the ledger

Proof of Persistence

Suppose a transaction is reported in the ledger of an honest party P_1

- Then it is in C_1[:-k] of P_1
- Suppose it is reported at a different position by some other party P_2
- Then it is in C_2[:-k] of P_2
- But this means C_1[:-k] does not share a prefix with C_2[:-k]
- This violates the Common Prefix property!

References

1. Juan Garay, Aggelos Kiayias, and Nikos Leonardos. *The bitcoin backbone protocol: Analysis and applications*. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 281–310. Springer, 2015.
2. Juan Garay, Aggelos Kiayias, and Nikos Leonardos. *The bitcoin backbone protocol with chains of variable difficulty*. In Annual International Cryptology Conference, pages 291–323. Springer, 2017.

45DC 00AE FDDF 5D5C B988 EC86 2DA4 50F3 AFB0 46C7

Thanks! Questions?

INPUT | OUTPUT

CARDANO

National and Kapodistrian
UNIVERSITY OF ATHENS