



BLOCKCHAIN PROGRAMMING ABSTRACTIONS

E. Anceaume, A. Del Pozzo, R. Ludinard, M. Potop, **S. Tucci-Piergiovanni**

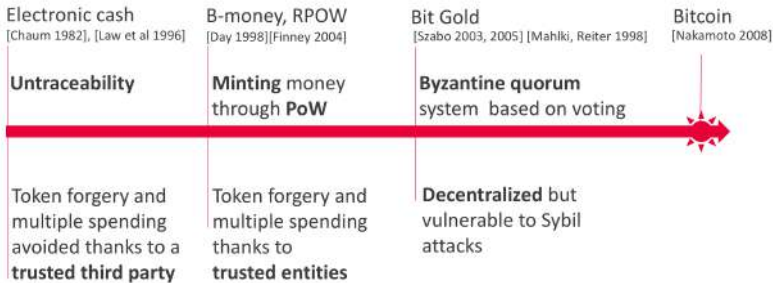
Algorithmique Distribuée, @LaBRI, Bordeaux

March 4th, 2019



Historical perspective

From the early 80s the vision of digital money has been around – but it took more than a quarter of century before a fully decentralized solution became a reality.



Bitcoin [Nakamoto 2008]

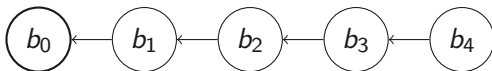
Combination of all the above-mentioned techniques for **full decentralization in an open system of untrusted peers.**

Proof-of-Work used to

- Limit the number of votes per entity – against Sybil Attack
- Limit multiple spending – coupled with longest chain rule
- Minting and Incentives for miners: miners as rational profit seekers, it must be profitable to follow the protocol



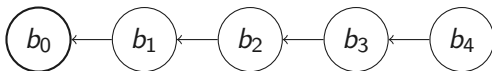
Blockchain in a nutshell



A Data Structure

- A sequence of blocks, each containing transactions and the solution of the PoW, replicated at each process p_i
- A block b_h at level h is linked to the block b_{h-1} at level $h - 1$ by containing the hash of b_{h-1}
- Immutability and Non-Repudiability

Blockchain in a nutshell



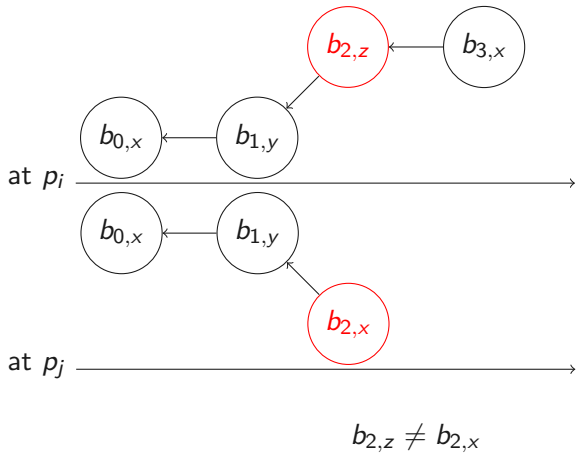
A Data Structure

- A sequence of blocks, each containing transactions and the solution of the PoW, replicated at each process p_i
- A block b_h at level h is linked to the block b_{h-1} at level $h - 1$ by containing the hash of b_{h-1}
- Immutability and Non-Repudiability

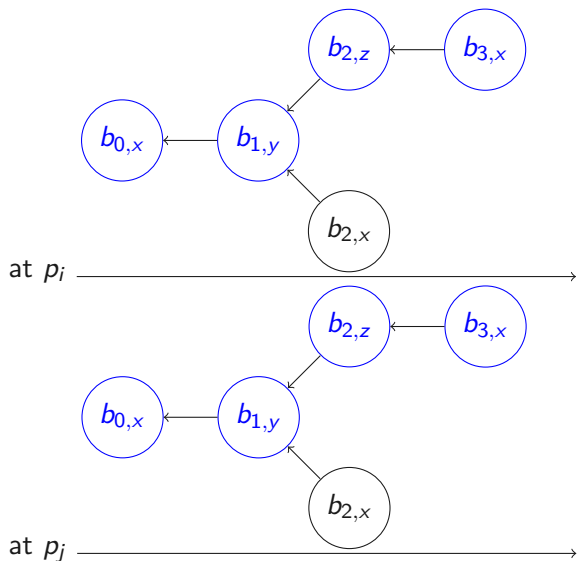
The (Bitcoin) Protocol to update the data structure at p_i

- Make a block b_h solving PoW
- Broadcast b_h
- Upon reception of b_h : verify b_h and locally append b_h if b_h is valid
- b_h contains the reward for the miner that made it

Consistency Issues: forks



Consistency Issues: reconciliation



Which consistency is really needed?

Bitcoin guys motto: “*just wait enough*” :

– make a transaction spendable only when it belongs to a block old enough –

Which consistency is really needed?

Bitcoin guys motto: “*just wait enough*” :

– make a transaction spendable only when it belongs to a block old enough –

Intuitively, this means to assume a known bound on the duration Δ of the time interval between any pair of blockchain heights.

Which consistency is really needed?

Bitcoin guys motto: "*just wait enough*" :

– make a transaction spendable only when it belongs to a block old enough –

Intuitively, this means to assume a known bound on the duration Δ of the time interval between any pair of blockchain heights.

...But in reality Δ is unknown so that two different processes can **read an inconsistent state**

Which consistency is really needed?

Bitcoin guys motto: “*just wait enough*” :

– make a transaction spendable only when it belongs to a block old enough –

Intuitively, this means to assume a known bound on the duration Δ of the time interval between any pair of blockchain heights.

...But in reality Δ is unknown so that two different processes can **read an inconsistent state**

...This is particularly true from the point of view of **smart contracts** who manipulate **any type of (replicated) variable**

Which consistency is really needed?

Bitcoin guys motto: “*just wait enough*” :

– make a transaction spendable only when it belongs to a block old enough –

Intuitively, this means to assume a known bound on the duration Δ of the time interval between any pair of blockchain heights.

...But in reality Δ is unknown so that two different processes can **read an inconsistent state**

...This is particularly true from the point of view of **smart contracts** who manipulate **any type of (replicated) variable**

(In the reminder of the presentation keep always in mind that the number of replicas is unknown)

(Ethereum) smart contracts

Smart contracts are programs who “live” in blockchain. The program is compiled and its bytecode is wrapped in a transaction, added to a block.

Client applications – wallets – can call the smart contract through function invocations.

Function invocations are treated as transactions by the Ethereum Virtual Machine (EVM)

A smart contract example

```
1 contract Puzzle{
2   address public owner;
3   bool public locked;
4   uint public reward;
5   bytes32 public diff;
6   bytes public solution;
7
8   function Puzzle() //constructor{
9     owner = msg.sender;
10    reward = msg.value;
11    locked = false;
12    diff = bytes32(11111); //pre-defined difficulty
13  }
14
15  function(){ //main code, runs at every invocation
16    if (msg.sender == owner){ //update reward
17      if (locked)
18        throw;
19      owner.send(reward);
20      reward = msg.value;
21    }
22    else
23      if (msg.data.length > 0){ //submit a solution
24        if (locked) throw;
25        if (sha256(msg.data) < diff){
26          msg.sender.send(reward); //send reward
27          solution = msg.data;
28          locked = true;
29        }}}
```

The first client that sends a valid solution gets a reward.

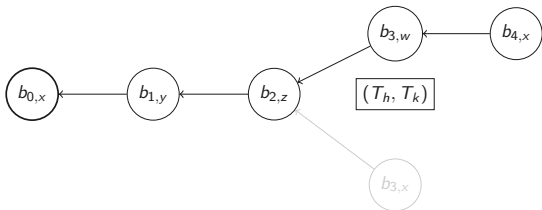
Invariant (Safety): no two winners.

EVM semantics [Luu et al 2016]

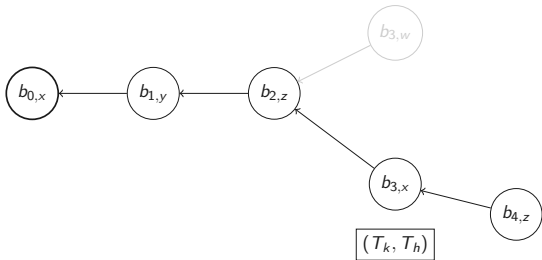
$$\begin{array}{c}
 \text{TXs} \leftarrow \text{Some transaction sequence } (T_1 \dots T_n) \text{ from } \Gamma \\
 B \leftarrow \langle \text{blockid ; timestamp ; TXs ; ...} \rangle \\
 \text{proof-of-work}(B, BC) \\
 \forall i, 1 \leq i \leq n : \sigma_{i-1} \xrightarrow{T_i} \sigma_i \\
 \text{PROPOSE} \frac{}{\langle BC, \sigma_0 \rangle \Downarrow \langle B \cdot BC, \sigma_n \rangle} \\
 \text{Remove } T_1 \dots T_n \text{ from } \Gamma \text{ and broadcast } B \\
 \\
 \text{Receive } B \equiv \langle \text{blockid ; timestamp ; TXs ; ...} \rangle \\
 \text{TXs} \equiv (T_1 \dots T_n) \\
 \forall i, 1 \leq i \leq n : \sigma_{i-1} \xrightarrow{T_i} \sigma_i \\
 \text{ACCEPT} \frac{}{\langle BC, \sigma_0 \rangle \Downarrow \langle B \cdot BC, \sigma_n \rangle} \\
 \text{Remove } T_1 \dots T_n \text{ from } \Gamma \text{ and broadcast } B
 \end{array}$$

Only one “elected leader” executes successfully the Propose rule at a given height of the blockchain BC . Other processes use the Accept rule to “repeat” the transitions $\sigma_{i-1} \rightarrow^{T_i} \sigma_i$ after the leader broadcasts block B .

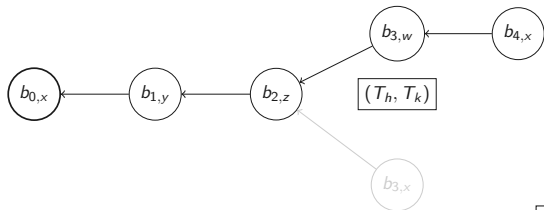
but when a fork occurs...



$T_h : \text{msg.sender} = p_h$
 $T_k : \text{msg.sender} = p_k$

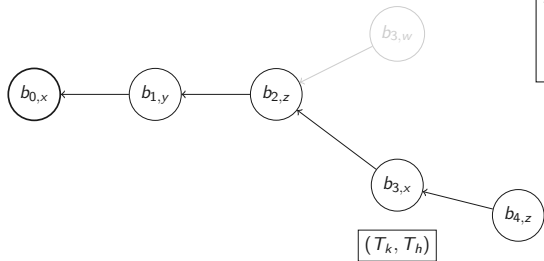


but when a fork occurs...

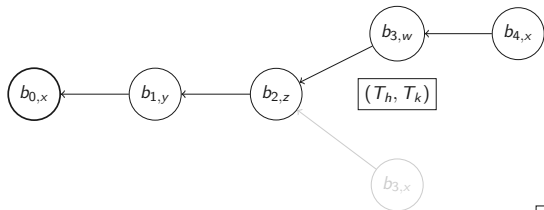


$T_h : msg.sender = p_h$
 $T_k : msg.sender = p_k$

the EVM could manage
to wait long enough
before letting the application
read at the two processes ?

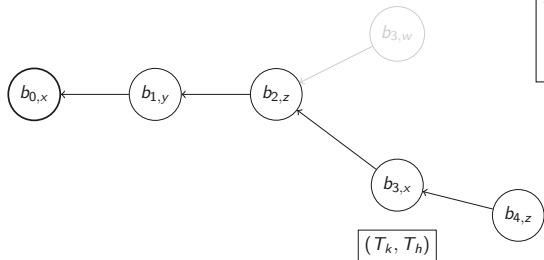


but when a fork occurs...



$T_h : msg.sender = p_h$
 $T_k : msg.sender = p_k$

the EVM could manage
to wait long enough
before letting the application
read at the two processes ?



If Δ is unknown
either we break liveness
– we wait forever –
or we break safety
– two winners –

Blockchains, how many?

2008,  S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System




Blockchains, how many?

2008,  S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System

2015, Ethereum  , Hyperledger 

2016, PeerCensus, ByzCoin, Tendermint

2017, RedBelly, Algorand 


... and many others

Blockchains, how many?

2008,  S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System

2015, Ethereum , Hyperledger 

2016, PeerCensus, ByzCoin, Tendermint

2017, RedBelly, Algorand 

... and many others


Which consistency?

Blockchains, how many?

2008,  S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System

2015, Ethereum , Hyperledger 

2016, PeerCensus, ByzCoin, Tendermint

2017, RedBelly, Algorand 

... and many others

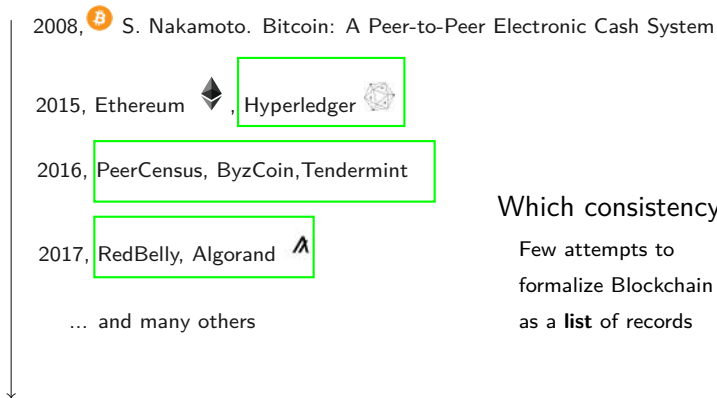
Which consistency?

Few attempts to
formalize Blockchain
as a **list** of records

2017. A. Girault et al., Why You Can't Beat Blockchains: Consistency and High Availability in Distributed Systems.

2018. A. Fernández Anta et al., Formalizing and implementing distributed ledger objects.

Blockchains, how many?



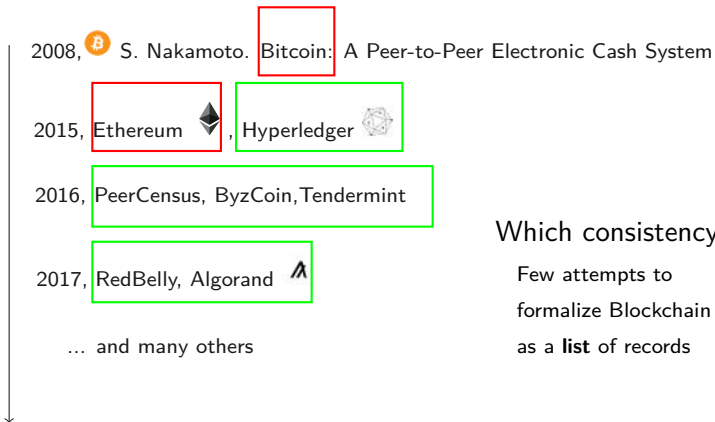
Which consistency?

Few attempts to
formalize Blockchain
as a **list** of records

2017. A. Girault et al., Why You Can't Beat Blockchains: Consistency and High Availability in Distributed Systems.

2018. A. Fernández Anta et al., Formalizing and implementing distributed ledger objects.

Blockchains, how many?



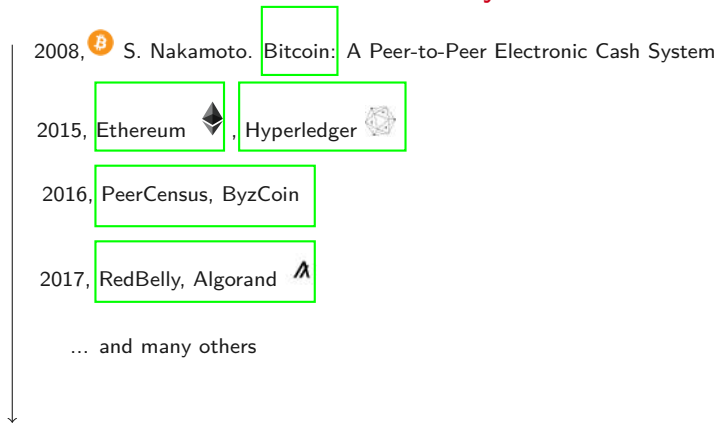
Which consistency?

Few attempts to formalize Blockchain as a **list** of records

2017. A. Girault et al., Why You Can't Beat Blockchains: Consistency and High Availability in Distributed Systems.

2018. A. Fernández Anta et al., Formalizing and implementing distributed ledger objects.

Blockchains, how many?



Our contribution

a formal unified framework providing blockchain consistency criteria to map current blockchains and state implementability results.

Formalization key points

1. Blockchain formalized as an Abstract Data Type to formally define the semantics of sequential and concurrent specifications;
2. The data type is a tree of blocks: the BlockTree Abstract Data Type;
3. The block generation process is encapsulated as separate data type: the Θ Token Oracle Abstract Data Type.

The Abstract Data Type



Abstract Data Types

- An abstract data type refers to a 6-tuple $T = \langle A, B, Z, \xi_0, \tau, \delta \rangle$ where:
 - A and B are countable sets called input alphabet and output alphabet;
 - Z is a countable set of abstract states and ξ_0 is the initial abstract state;
 - $\tau : Z \times A \rightarrow Z$ is the transition function;
 - $\delta : Z \times A \rightarrow B$ is the output function.

Abstract Data Types

- An abstract data type refers to a 6-tuple $T = \langle A, B, Z, \xi_0, \tau, \delta \rangle$ where:
 - A and B are countable sets called input alphabet and output alphabet;
 - Z is a countable set of abstract states and ξ_0 is the initial abstract state;
 - $\tau : Z \times A \rightarrow Z$ is the transition function;
 - $\delta : Z \times A \rightarrow B$ is the output function.
- An abstract data type, by its transition system, defines the sequential specification of an object. If we consider a path that traverses its system of transitions, then the word formed by the subsequent labels on the path is a sequential history.

Abstract Data Types

- An abstract data type refers to a 6-tuple $T = \langle A, B, Z, \xi_0, \tau, \delta \rangle$ where:
 - A and B are countable sets called input alphabet and output alphabet;
 - Z is a countable set of abstract states and ξ_0 is the initial abstract state;
 - $\tau : Z \times A \rightarrow Z$ is the transition function;
 - $\delta : Z \times A \rightarrow B$ is the output function.
- An abstract data type, by its transition system, defines the sequential specification of an object. If we consider a path that traverses its system of transitions, then the word formed by the subsequent labels on the path is a sequential history.
- Concurrent histories are defined considering a partial order relations among events executed by different processes $H = \langle \Sigma, E, \wedge, \mapsto, \prec, \nearrow \rangle$, where $o \in \Sigma = A \cup (A \times B)$ are operations.

Abstract Data Types

- An abstract data type refers to a 6-tuple $T = \langle A, B, Z, \xi_0, \tau, \delta \rangle$ where:
 - A and B are countable sets called input alphabet and output alphabet;
 - Z is a countable set of abstract states and ξ_0 is the initial abstract state;
 - $\tau : Z \times A \rightarrow Z$ is the transition function;
 - $\delta : Z \times A \rightarrow B$ is the output function.
- An abstract data type, by its transition system, defines the sequential specification of an object. If we consider a path that traverses its system of transitions, then the word formed by the subsequent labels on the path is a sequential history.
- Concurrent histories are defined considering a partial order relations among events executed by different processes $H = \langle \Sigma, E, \Lambda, \mapsto, \prec, \nearrow \rangle$, where $o \in \Sigma = A \cup (A \times B)$ are operations.
- A consistency criterion is a function $C : \mathcal{T} \rightarrow \mathcal{P}(\mathcal{H})$ where \mathcal{T} is the set of abstract data types, \mathcal{H} is a set of histories and $\mathcal{P}(\mathcal{H})$ is the sets of parts of \mathcal{H} . An algorithm A_T implementing the ADT $T \in \mathcal{T}$ is C -consistent with respect to criterion C if all the operations terminate and all the admissible executions are C -consistent, i.e. they belong to the set of histories $C(T)$.

The Block Tree



BlockTree Abstract Data Type

The BlockTree Abstract Data Type exposes two operations:

- `read()`: selects a blockchain in the blocktree;
- `append(b)`: appends the block *b* to the blocktree if such block is valid, i.e., it satisfies a predicate *P*.

Sequential specification

- \mathcal{B} : countable and non empty set of blocks;
- $\mathcal{B}' \subseteq \mathcal{B}$ a countable and non empty set of valid blocks, i.e., $\forall b \in \mathcal{B}', P(b) = \top$.
By assumption $b_0 \in \mathcal{B}'$;
- \mathcal{BT} a a countable non empty set of blocktrees. A *directed rooted tree* $bt = (V_{bt}, E_{bt})$ where each vertex of the BlockTree is a *block* and any edge points backward to the root, called *genesis block*;
- \mathcal{BC} a countable non empty set of blockchains, where a blockchain is a path from a leaf of bt to b_0 .
- \mathcal{F} is a countable non empty set of selection functions, $f \in \mathcal{F} : \mathcal{BT} \rightarrow \mathcal{BC}$.

Def. BT-ADT = $\langle A = \{\text{append}(b), \text{read}(): b \in \mathcal{B}\}, B = \mathcal{BC} \cup \{\text{true}, \text{false}\},$
 $Z = \mathcal{BT} \times \mathcal{F} \times (\mathcal{B} \rightarrow \{\text{true}, \text{false}\}), \xi_0 = (b_0, f), \tau, \delta \rangle$

Sequential specification (cont.)

the transition function $\tau : Z \times A \rightarrow Z$ is defined by

$$\tau((bt, f, P), \text{read}()) = (bt, f, P);$$

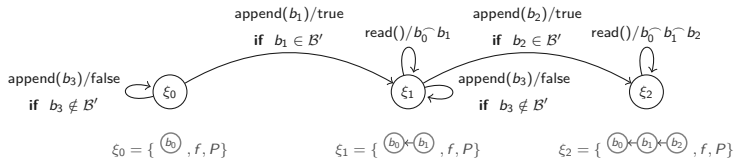
$$\tau((bt, f, P), \text{append}(b)) = \begin{cases} (\{b_0\} \frown f(bt) \frown \{b\}, f, P) & \text{if } b \in \mathcal{B}' \\ (bt, f, P) & \text{otherwise} \end{cases}$$

and the output function $\delta : Z \times A \rightarrow B$ is defined by

$$\delta((bt, f, P), \text{append}(b)) = \begin{cases} \text{true} & \text{if } b \in \mathcal{B}' \\ \text{false} & \text{otherwise} \end{cases}$$

$$\delta((bt, f, P), \text{read}()) = \begin{cases} \{b_0\} & \text{if } bt = b_0 \\ \{b_0\} \frown f(bt) & \text{otherwise} \end{cases}$$

Sequential histories



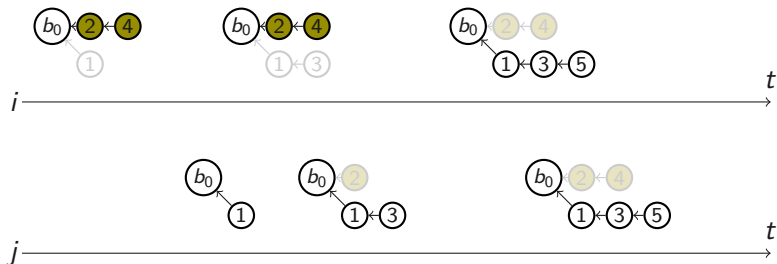
Consistency criteria

Two consistency criteria:

- eventual consistency;
- strong consistency.

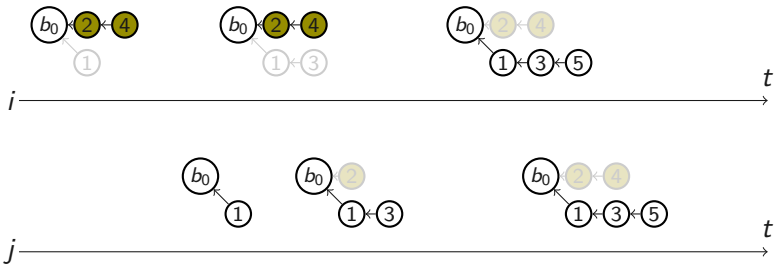
Each criteria is a conjunction of properties.

Validity Property



Validity property: all the blocks read are valid and have been appended by some process.

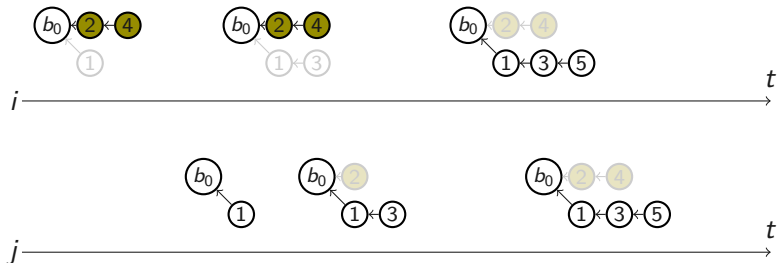
Local Monotonic Read Property



Local monotonic read property: the **score** of the sequence of blockchains read at the same process never decreases.

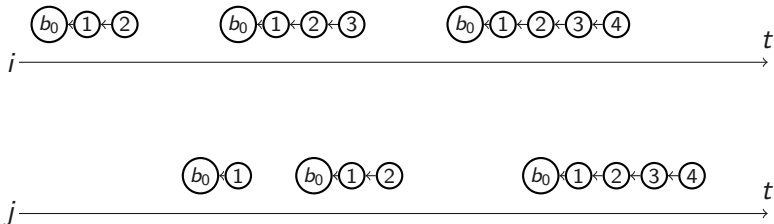
score: it can be the length, the weight, etc...

Ever Growing Tree Property



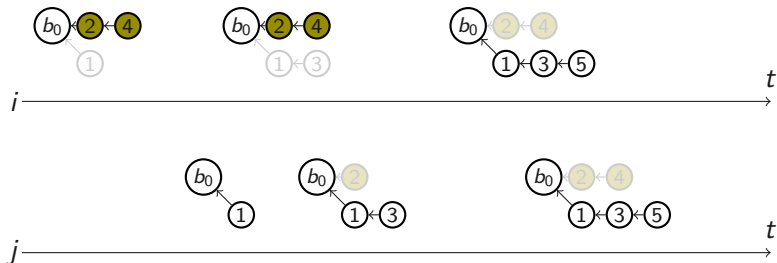
Ever growing tree property: the score of returned blockchains eventually grows.

Strong Prefix Property



Strong prefix property: for each pair of `read()` operations, one returns a blockchain that is the prefix of the other or vice versa.

Eventual Prefix Property



Eventual prefix property: For each read blockchain with a score s , eventually all the subsequent read blockchains share a maximum common prefix with a score of at least s .

Consistency Criteria

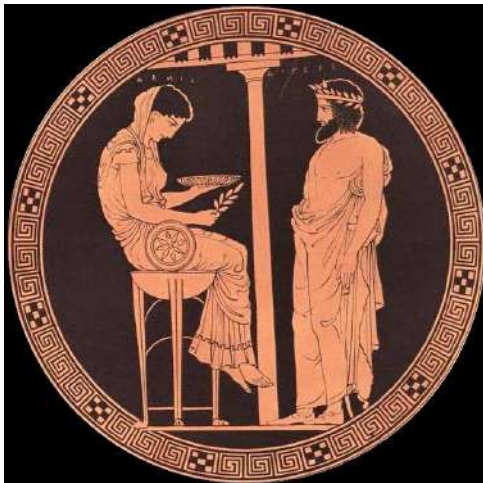
Eventual Consistency Criterion (EC):

- Local Monotonic Read;
- Validity;
- Ever Growing Tree;
- **Eventual Prefix.**

Strong Consistency Criterion (SC) :

- Local Monotonic Read;
- Validity;
- Ever Growing Tree;
- **Strong Prefix.**

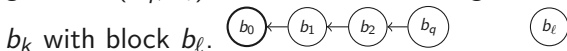
Token Oracle



Token Oracle

The Token Oracle Θ_k Abstract Data Type exposes two operations:

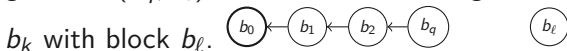
- $\text{getToken}(b_q, b_\ell)$: returns or not the right to extend the block



Token Oracle

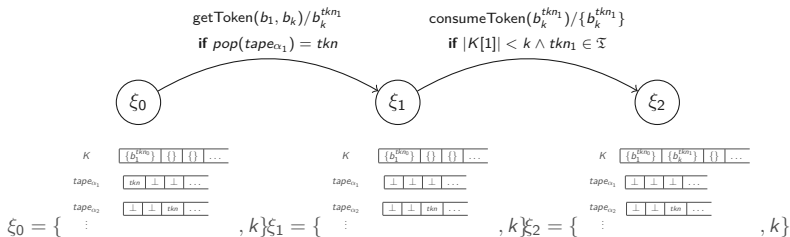
The Token Oracle Θ_k Abstract Data Type exposes two operations:

- $\text{getToken}(b_q, b_\ell)$: returns or not the right to extend the block



- $\text{consumeToken}(b_\ell^{b_q})$: allows a valid block to be appended or not, depending on how many blocks already extend b_q .

Sequential histories



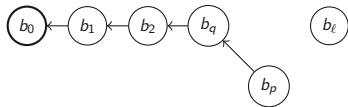
getToken() if called a finite (but unknown) number of times returns a valid token.

consumeToken() returns the token that has been appended.

Frugal and Prodigious Token Oracles

A Frugal Oracle $\Theta_{F,k}$ allows to append at most k blocks to the same block.

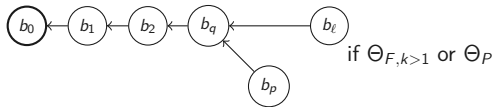
A Prodigious Oracle Θ_P allows to append an unlimited number of blocks to any block.



Frugal and Prodigious Token Oracles

A Frugal Oracle $\Theta_{F,k}$ allows to append at most k blocks to the same block.

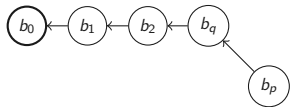
A Prodigious Oracle Θ_P allows to append an unlimited number of blocks to any block.



Frugal and Prodigious Token Oracles

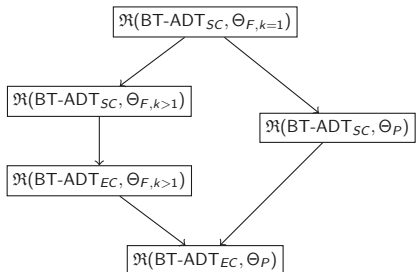
A Frugal Oracle $\Theta_{F,k}$ allows to append at most k blocks to the same block.

A Prodigious Oracle Θ_P allows to append an unlimited number of blocks to any block.



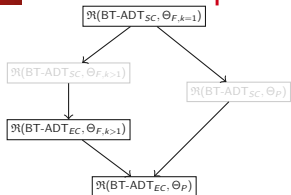
if $\Theta_{F,k=1}$

Blocktree and Oracle ADT hierarchy



- We refine the BlockTree ADT `append()` with the Oracle ADT, the refinement is denoted as $\mathfrak{R}(\text{BT-ADT}, \Theta)$
- We organize the refinements in a hierarchy. In this way, we can state impossibility results on the weakest combination and propagate them above.

Implementability results



- (1) $\Theta_{F,k=1}$ has Consensus number ∞
- (2) Θ_P has Consensus number 1
- (3) Reliable Communication is necessary in eventual consistent blockchains
- (4) Impossible to implement Strong Consistency if forks can occur. Direct implication: non-implementability of refinements $\mathfrak{R}(\text{BT-ADT}_{SC}, \Theta_P)$ and $\mathfrak{R}(\text{BT-ADT}_{SC}, \Theta_{F,k>1})$.
- (5) **From (1),(3) and (4): Consensus and Reliable Communication are necessary in non-forkable blockchains.**
- (6) **From (1),(3) and (4): Forkable blockchains with a possible unbounded number of forks are implementations of atomic storage.**

Mapping with existing solutions

References	Refinement
Bitcoin	$\mathfrak{R}(BT-ADT_{EC}, \Theta_P)$
Ethereum	$\mathfrak{R}(BT-ADT_{EC}, \Theta_P)$
Algorand	$\mathfrak{R}(BT-ADT_{SC}, \Theta_{F,k=1})$
ByzCoin	$\mathfrak{R}(BT-ADT_{SC}, \Theta_{F,k=1})$
PeerCensus	$\mathfrak{R}(BT-ADT_{SC}, \Theta_{F,k=1})$
Redbelly	$\mathfrak{R}(BT-ADT_{SC}, \Theta_{F,k=1})$
Hyperledger	$\mathfrak{R}(BT-ADT_{SC}, \Theta_{F,k=1})$
Tendermint	$\mathfrak{R}(BT-ADT_{SC}, \Theta_{F,k=1})$

Conclusions and Future Work

We presented a formal specification of blockchains using a modular approach.

Takeaways:

- BitCoin and Ethereum are implementations of registers in a dynamic system subject to malicious attacks, included the Sybil attack
- To guarantee no-forks a Consensus oracle is needed: the generation of the block as output of a Consensus instance solved inside a selected committee [Amoussou et al OPODIS 2018]
- Reliable communication in a dynamic system subject to Byzantine behavior is required;

Future works

- Continue to work on implementability of defined ADTs in a message-passing dynamic system;
- Fairness properties for oracles;
- Formal definition of a blockchain execution model for smart contract virtual machines, necessary to prove invariants w.r.t. the blockchain consistency level provided.

The material of this presentation has been taken mainly from:

Anceaume et al 2018. E. Anceaume, A. Del Pozzo, R. Ludinard, M. Potop-Butucaru, and S. Tucci-Piergiovanni. Blockchain Abstract Data Type. In CoRR abs/1802.09877 and Poster at PPOPP 2019

Other references:

Chaum 1982. David Chaum. Blind Signatures for Untraceable Payments. In CRYPTO '82: Proceedings of the 2nd Conference on Advances in Cryptology. 199–203.

Law et al. 1996. Law, Sabett and Solinas. How to Make a Mint: The Cryptography of Anonymous Electronic Cash. American University Law Review 46, 4, 1996, 1131–1162

Dai 1998. Wei Dai. 1998. B-Money. <http://www.weidai.com/bmoney>

Finney 2004. Hal Finney. 2004. RPOW. (2004). <http://cryptome.org/rpow.htm>

Szabo 2003. Nick Szabo. 2003. Advances in Distributed Security.

Szabo 2005. Nick Szabo. 2005. Bit Gold. <http://unenumerated.blogspot.de/2005/12/bit-gold.html>

Malkhi and Reiter 1998. Dahlia Malkhi and Michael Reiter. 1998. Byzantine quorum systems. Distributed Computing 11, 4 (1998), 203–213.

Nakamoto 2008a. Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system.

Garay 2014. J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In EUROCRYPT 2015.

Amoussou et al OPODIS 2018. Amoussou-Guenou, Del Pozzo, Potop-Butucaru, Tucci-Piergiovanni. Correctness of Tendermint-core Blockchains. OPODIS 2018

Gilad et al 2017. Gilad, Hemo, Micali, Vlachos and Zeldovich: Algorand: Scaling Byzantine Agreements for Cryptocurrencies. SOSP 2017

Luu et al 2016. L. Luu, D. Chu, H. Olickel, P. Saxena, and A. Hobor. 2016. Making Smart Contracts Smarter. CCS 2016.

Herlihy 1991. M. Herlihy. Wait-free synchronization. ACM Trans. Program. Lang. Syst., 13(1):124–149, 1991.

Aguilera 2011. M. K. Aguilera, I. Keidar, D. Malkhi, and Al. Shraer. Dynamic atomic storage without consensus. J. ACM 58, 2, Article 7 (April 2011)

Tokenomics

- <http://www.tokenomics2019.org/tokenomics/>



The screenshot shows the homepage of the Tokenomics 2019 conference website. At the top, there is a banner image of a modern building at night with the text: "Tokenomics International Conference on Blockchain Economics, Security and Protocols May 6 and 7, 2019 Paris". Below the banner is a navigation menu with links for "COMMITTEES", "INFO FOR AUTHORS", "INFO FOR ATTENDEES", and "CONTACTS". The main content area contains a paragraph describing the conference as an international forum for theory, design, analysis, implementation, and applications of blockchain and smart contracts. It also states the goal of the conference is to bring economists together with computer science researchers and practitioners working on blockchain in various projects. At the bottom, there is a section titled "Sponsored Cities" listing January 15, 2019 (Sorbonne Institute for selected papers), March 1, 2019 (Nipponponne conference), and May 6-7, 2019 (Conference). The footer features logos for sponsors: list, CREST, ENS, PSL, and SORBONNE UNIVERSITE.

Thank you
Questions ?

Commissariat à l'énergie atomique et aux énergies alternatives
CEA Tech List
Centre de Saclay — 91191 Gif-sur-Yvette Cedex
www-list.cea.fr