Demystifying Bitcoin

Prof R. Guerraoui
EPFL & Collège de France & UM6P
Have you heard about?

- Bitcoin
- Blockchain
- Ethereum
- Signatures
- Proof of work
- Smart contracts
- Turing Completeness
- NP vs P
- Consensus
- Snapshot
Perspectives

(1) The journalist

(2) The user

(3) The participant

(4) The engineer

(5) The scientist
(1) The Journalist

2008: Financial crisis – Nakamoto (1/21m)
- From 1c to 8000$ through 20000$

From trading hardware to general trading

2014: Ethereum (CH) - Now 800 $

2020: Libra - FacebookCoin
Perspectives

(1) The journalist
(2) The user
(3) The participant
(4) The engineer
(5) The scientist
## (2) The User

### BlockChain

**BE YOUR OWN BANK.**

<table>
<thead>
<tr>
<th>Amount</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00000546 BTC</td>
<td></td>
</tr>
<tr>
<td>0.102338636803627092 ETH</td>
<td></td>
</tr>
<tr>
<td>$23.08</td>
<td></td>
</tr>
</tbody>
</table>

**Send**

- **SENT**
  - **July 21 @ 10:10 AM**
  - To: 0x9970b7e233555a037311be1f5261b59393d6961f
  - From: My Ethereum Wallet
  - Add a description
  - 0.0001 ETH

- **SENT**
  - **July 18 @ 02:54 PM**
  - To: 0x16a6920db1f14fc4f325c94a5e2d20c1fba868
  - From: My Ethereum Wallet
  - Add a description
  - 0.0001416 ETH

**Received**

- **RECEIVED**
  - **July 17 @ 11:44 AM**
  - To: My Ethereum Wallet
  - From: 0x3bc51ab9de1e5b7b6e3e5b960285805c41736
  - Add a description
  - 0.08380039 ETH

- **RECEIVED**
  - **July 13 @ 03:03 PM**
  - To: My Ethereum Wallet
  - From: 0xced16856d551569d134530ee3967ec79995e2051
  - Add a description (test, hey Jamie!)
  - 0.01966193 ETH
(2) The User

The wallet: 1 private key + several public keys

Transaction validation
- signing + gossiping + mining + chaining

Transaction commitment
- After time $t$: thousands of users have seen it
(3) The Participant

Honey, I'm home!
I found a block today!

Miner Jack
(3) The Participant

<table>
<thead>
<tr>
<th>Block:</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonce:</td>
<td>2790</td>
<td></td>
</tr>
<tr>
<td>Data:</td>
<td>NCore</td>
<td></td>
</tr>
<tr>
<td>Hash:</td>
<td>0000c5f693ac77a18ae73ace5df932457fc62e8dfa23c2f3c6d8eb125ba7843</td>
<td></td>
</tr>
</tbody>
</table>

Mine
(3) The Participant

To validate a transaction, a miner has to solve a puzzle including it
- Fairness and cooperation

Incentive: 12 bitcoins / puzzle
- 50 bitcoins 3 years ago

Total: 21 millions bitcoins
- Now: 17 millions
(4) The Engineer

- Joinning (a P2P network)
- Signing (a transaction)
- Gossiping (the transaction)
- Gathering (a block)
- Mining (proof of work - nonce)
- Chaining (hash)
- Gossiping (the block)
- Committing/Aborting
TECHNOLOGIES OF A BLOCKCHAIN

Asymmetric Encryption
Transaction signing

Hash Functions
Transaction/block hashing as well as obfuscating public keys

Merkle Trees
Efficient way to package transactions into blocks

Key-Value Database
Lookups of previous transactions (prevent double-spends)

P2P Communication Protocol
Sharing transactions and blocks

Proof of Work
Method to achieve consensus
The Big Picture

Bitcoin block

Mining: find $\texttt{nonce}$ such that $\texttt{This #} < d$

How? By trying different nonces (brute force)
Smart Contracts

Option contract written as code into a blockchain.

Contract is part of the public blockchain.

Parties involved in the contract are anonymous.

Contract executes itself when the conditions are met.

Regulators use blockchain to keep an eye on contracts.

Happy Hustlin’

https://codebrahma.com
partner_1 = contract.storage[I_PARTNER_1]
partner_2 = contract.storage[I_PARTNER_2]

if state == S_PROPOSED and tx.sender == partner_2 and tx.data[0] == partner_1:
    contract.storage[I_STATE] = S_MARRIED

else if state == S_MARRIED and tx.sender == partner_1 or tx.sender == partner_2:
    if tx.data[0] == TX_WITHDRAW:
        creator = contract.storage[I_WITHDRAW_CREATOR]
        if creator != 0 and contract.storage[I_WITHDRAW_TO] == tx.data[1] and contract.storage[I_WITHDRAW_AMOUNT] == tx.data[2]:
            mtxt(x.data[1], x.data[2], 0, 0)
            contract.storage[I_WITHDRAW_TO] = 0
            contract.storage[I_WITHDRAW_AMOUNT] = 0
            contract.storage[I_WITHDRAW_CREATOR] = 0
        else:
            contract.storage[I_WITHDRAW_TO] = tx.data[1]
            contract.storage[I_WITHDRAW_AMOUNT] = tx.data[2]
            contract.storage[I_WITHDRAW_CREATOR] = tx.sender

    else if tx.data[0] == TX_DIVORCE:
        creator = contract.storage[I_DIVORCE_CREATOR]
        if creator != 0 and creator != tx.sender:
            balance = block.account_balance(contract.address)
            mtxt(partner_1, balance / 2, 0, 0)
            mtxt(partner_2, balance / 2, 0, 0)
            contract.storage[I_STATE] = S_DIVORCED
Perspectives

(1) The journalist

(2) The user

(3) The participant

(4) The engineer

(5) The scientist
(5) The Scientist

- Conjecture 1: Turing Universality
- Conjecture 2: P is not NP
- Theorem 1: Lamport (Consensus) Universality
- Theorem 2: Consensus Impossibility
Turing Universality (36)
P vs NP (Nash/GV 50 – Ford 70)

? * ? = 91

7 * 13 = ?
Lamport Universality (78)
Consensus Universality (78)

**Safety:** No two nodes must choose different values.

The chosen value must have been proposed by a node.

**Liveness:** Each node must eventually choose a value.

Every service can be implemented in a highly available manner using Consensus.
Consensus is impossible in an asynchronous system
Perspectives

(1) The journalist
(2) The user
(3) The participant
(4) The engineer
(5) The scientist
Can we implement a payment system asynchronously?
The infinitely big

The infinitely small
Message Passing

p1 Send

p2

p3 Receive
Shared Memory

Write()  

p1  

1  

Read()  

p2  

Registers

Message Passing
Atomic Shared Memory

\[ \text{write}(1) - \text{ok} \]

\[ \text{read}() - 1 \]

\[ \text{read}() - 1 \]
Atomic Shared Memory

write(1) - ok

read() - 1

read() - 0
Non-Atomic Shared Memory

write(1) - ok

read() - 0

read() - 1
Non-Atomic Shared Memory

write(1) - ok

p1

read() - 0

p2

read() - 1

p3
Message Passing ⇔ Shared Memory

Quorums (asynchrony)
Message Passing $\iff$ Shared Memory

Quorums (asynchrony)
Message Passing $\leftrightarrow$ Shared Memory

Quorums (asynchrony)
« To understand a distributed computing problem: bring it to shared memory » T. Lannister

« Optimization is the source of all evil » D. Knuth
P vs NP

7 * 13 = ?

? * ? = 91

Asynchronous vs Synchronous
Can we implement a payment system asynchronously?
Counter: Specification

A counter has two operations inc() and read(); it maintains an integer x init to 0

read():
return(x)

inc():
x := x + 1;
return(ok)
Counter: Algorithm

- The processes share an array of registers \( \text{Reg}[1, \ldots, N] \)

- \( \text{inc}() \):
  - \( \text{Reg}[i].\text{write}(\text{Reg}[i].\text{read()} + 1); \)
  - return(ok)

- \( \text{read}() \):
  - \( \text{sum} := 0; \)
  - for \( j = 1 \) to \( N \) do
    - \( \text{sum} := \text{sum} + \text{Reg}[j].\text{read}(); \)
  - return(sum)
Counter*: Specification

Counter* has, in addition, operation dec()

dec():
if x > 0 then x := x - 1; return(ok)
else return(no)

Can we implement Counter* asynchronously?
2-Consensus with Counter*

- Registers R0 and R1 and Counter* C - initialized to 1

- Process pI:
  - propose(vI)
  - RI.write(vI)
  - res := C.dec()
  - if(res = ok) then
    - return(vI)
  - else return(R{1-I}.read())
Impossibility [FLP85,LA87]

- **Theorem**: no asynchronous algorithm implements *consensus* among two processes using *registers*

- **Corollary**: no asynchronous algorithm implements Counter* among two processes using *registers*
- **Theorem**: no *asynchronous* algorithm implements *set-agreement* using *registers*
The **consensus number** of an object is the maximum number of processes than can solve consensus with it.
Payment Object (PO): Specification

Pay(a, b, x): transfer amount x from a to b if a > x (return ok; else return no)

NB. Only the owner of a invokes Pay(a, *, *)

- **Questions**: can PO be implemented asynchronously? what is the consensus number of PO?
Payment Object (PO): Specification

Pay(a,b,x): transfer amount x from a to b if a > x (return ok; else return no)

NB. Only the owner of a invokes Pay(a,*,*)

Questions: can PO be implemented asynchronously? what is the consensus number of PO?
A snapshot has operations \textit{update()} and \textit{scan()}; it maintains an array $x$ of size $N$.

\begin{itemize}
  \item \textit{scan()}:
    \begin{itemize}
      \item return($x$)
    \end{itemize}

  \item \textit{update}(i, v):
    \begin{itemize}
      \item $x[i] := v$
      \item return(ok)
    \end{itemize}
\end{itemize}
Algorithm?

The processes share one array of N registers Reg[1,..,N]

scan():
  for j = 1 to N do
    x[j] := Reg[j].read();
  return(x)

update(i,v):
  Reg[i].write(v); return(ok)
Atomicity?

update(1,1) - ok

scan() - [1,0,2]

update(3,2) - ok
Atomicity?

update(1,1) - ok
scan() - [1,0,2]
update(3,2) - ok
Atomicity?

```plaintext
scan() - [0,0,10]

update(2,1) - ok

update(3,10) - ok
```
Key idea for atomicity

To *scan*, a process keeps reading the entire snapshot (i.e., *collecting*), until two arrays are the same.

Key idea for wait-freedom

To update, scan then write the value and the scan.

To *scan*, a process keeps collecting and returns a collect if it did not change, or some collect returned by a concurrent *scan*. 
The Payment Object: Algorithm

Every process stores the sequence of its outgoing payments in its snapshot location.

To **pay**, the process scans, computes its current balance: if bigger than the transfer, updates and returns ok, otherwise returns no.

To **read**, scan and return the current balance.
PO can be implemented Asynchronously

Consensus number of PO is 1

Consensus number of PO(k) is k
(5) The Scientist

Conjecture 1: Turing Universality

Conjecture 2: P is not NP

Theorem 1: Lamport (Consensus) Universality

Theorem 2: Consensus Impossibility

Theorem 3: PO < Consensus
Payment System (AT2)

- AT2_S
- AT2_D
- AT2_R

Number of lines of code: one order of magnitude less

Latency: seconds (at most)