Demystifying Bitcoin

Prof R. Guerraoui EPFL
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 10000$ through 20000$

From trading hardware to general trading

2014: Ethereum (CH) - Now 800 $

2020: Libra - Facebook Coin
Perspectives

(1) The journalist

(2) The user

(3) The participant

(4) The engineer

(5) The scientist
(2) The User
(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!

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>2</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>7</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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>0000c5f693ac77a18ae73ace5df932457fc62e8dfe23c2f3c6d8ebb125ba7843</td>
<td></td>
</tr>
</tbody>
</table>

[Button: 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

- Joining (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
Hashing

Input Data → Hashing Algo → Output Hash

Blockgeeks
The Big Picture

Bitcoin block

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

How? By trying different nonces (brute force)
<table>
<thead>
<tr>
<th>Block:</th>
<th>0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonce:</td>
<td>2790</td>
</tr>
<tr>
<td>Data:</td>
<td>NCore</td>
</tr>
<tr>
<td>Hash:</td>
<td>0000c5f693ac77a18ae73ace5df932457fc62e8dfe23c2f3c6d8ebb125ba7843</td>
</tr>
</tbody>
</table>

Mine
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://codebrhma.com
partner_1 = contract.storage[1_PARTNER_1]
partner_2 = contract.storage[1_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]:
            mctx(tx.data[1], tx.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)
        mctx(partner_1, balance / 2, 0, 0)
        mctx(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
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)
Every service can be implemented in a highly available manner using Consensus

**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.
Consensus Impossibility (84)

Consensus is impossible in an asynchronous system.
Payment System

Can we implement a payment system asynchronously?
The infinitely big

The infinitely small
Message Passing

p1 -> Send -> p2

p2 -> Receive -> p3
Shared Memory

Write()

1

Read()

Message Passing
Atomic Shared Memory

write(1) - ok

p1

read() - 1

read() - 1

p2

p3
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

read() - 0

read() - 1
Message Passing $\iff$ Shared Memory

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

Quorums (asynchrony)
Message Passing $\iff$ 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

$7 \times 13 = ?$

$? \times ? = 91$

Asynchronous vs Synchronous

p1  \[
\text{Write}(P1)
\]

Write(P1)

p2  \[
\text{Write}(P2)
\]

Write(P2)
Payment System

Atomicity
Wait-freedom

Can we implement a payment system asynchronously?
A counter has two operations $inc()$ and $read()$; it maintains an integer $x$ \textit{init to 0}.

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

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

\textit{inc():}
\begin{itemize}
  \item Reg[i].write(Reg[i].read() + 1);
  \item return(ok)
\end{itemize}

\textit{read():}
\begin{itemize}
  \item sum := 0;
  \item for j = 1 to N do
        \begin{itemize}
          \item sum := sum + Reg[j].read();
        \end{itemize}
  \item return(sum)
\end{itemize}
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?
A snapshot has operations `update()` and `scan()`; it maintains an array `x` of size `N`.

- `scan()`:
  - `return(x)`

- `update(i,v)`:
  - `x[i] := v;`
  - `return(ok)`
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

p1

scan() - [1,0,2]

p2

update(3,2) - ok

p3
Atomicity?

update(1,1) - ok

update(3,2) - ok

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

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)