Database and Distributed Computing
Foundations of Blockchains

Sujaya Maiyya, Victor Zakhary, Mohammad Javad Amiri, Divyakant Agrawal, Amr El Abbadi

{sujaya-maiyya, victorzakhary, amiri, agrawal, amr}@cs.ucsb.edu
Traditional Banking Systems
Traditional Banking Systems
Traditional Banking Systems
Traditional Banking Systems
Traditional Banking Systems
Traditional Banking Systems
Traditional Banking Systems
Traditional Banking Systems

• From Database and Distributed Computing Perspective
Traditional Banking Systems

• From Database and Distributed Computing Perspective
• Identities and Signatures
Traditional Banking Systems

• From Database and Distributed Computing Perspective
• Identities and Signatures
  • You are your signature [ID, username and password]
Traditional Banking Systems

• From Database and Distributed Computing Perspective
• Identities and Signatures
  • You are your signature [ID, username and password]
• Ledger
Traditional Banking Systems

• From Database and Distributed Computing Perspective
• Identities and Signatures
  • You are your signature [ID, username and password]
• Ledger
  • The balance of each identity (saved in a DB)
Traditional Banking Systems

• From Database and Distributed Computing Perspective
• Identities and Signatures
  • You are your signature [ID, username and password]
• Ledger
  • The balance of each identity (saved in a DB)
• Transactions
Traditional Banking Systems

• From Database and Distributed Computing Perspective
• Identities and Signatures
  • You are your signature [ID, username and password]
• Ledger
  • The balance of each identity (saved in a DB)
• Transactions
  • Move money from one identity to another
Traditional Banking Systems

- From Database and Distributed Computing Perspective
- Identities and Signatures
  - You are your signature [ID, username and password]
- Ledger
  - The balance of each identity (saved in a DB)
- Transactions
  - Move money from one identity to another
  - Concurrency control to serialize transactions (prevent double spending)
Traditional Banking Systems

• From Database and Distributed Computing Perspective

• Identities and Signatures
  • You are your signature [ID, username and password]

• Ledger
  • The balance of each identity (saved in a DB)

• Transactions
  • Move money from one identity to another
  • Concurrency control to serialize transactions (prevent double spending)
  • Typically backed by a transactions log
Traditional Banking Systems

• From Database and Distributed Computing Perspective
• Identities and Signatures
  • You are your signature [ID, username and password]
• Ledger
  • The balance of each identity (saved in a DB)
• Transactions
  • Move money from one identity to another
  • Concurrency control to serialize transactions (prevent double spending)
  • Typically backed by a transactions log
    • Log is persistent
Traditional Banking Systems

- From Database and Distributed Computing Perspective
- Identities and Signatures
  - You are your signature [ID, username and password]
- Ledger
  - The balance of each identity (saved in a DB)
- Transactions
  - Move money from one identity to another
  - Concurrency control to serialize transactions (prevent double spending)
  - Typically backed by a transactions log
    - Log is persistent
    - Log is immutable and tamper-free (end-users trust this)
Bitcoin
Bitcoin
Bitcoin
Bitcoin: A Peer-to-Peer Electronic Cash System

- From Database and Distributed Computing Perspective
- Identities and Signatures
  - Public/Private key pair
- Ledger
  - The balance of each identity (saved in the blockchain)
- Transactions
  - Move bitcoins from one identity to another
  - Concurrency control to serialize transactions (Mining and PoW)
  - Typically backed by a transactions log (blockchain)
    - Log is persistent (replicated across the network nodes)
    - Log is immutable and tamper-free (PoW and Hash pointers)
Digital Signatures
Digital Signatures

• $P_k, S_k \leftarrow \text{Keygen}(\text{keysize})$
Digital Signatures

• $P_k, S_k \xleftarrow{\text{Keygen}} \text{keysize}$
• Your $P_k$ is your identity (username, e-mail address)
Digital Signatures

• $P_k$, $S_k \leftarrow \text{Keygen}(\text{keysize})$
• Your $P_k$ is your identity (username, e-mail address)
• Your $S_k$ is your signature (password)
• $P_k$ is made public and used to verify documents signed by $S_k$
• $S_k$ is private
Digital Signatures

• $P_k$ is made public and used to verify documents signed by $S_k$
• $S_k$ is private
Digital Signatures

- $P_k$ is made public and used to verify documents signed by $S_k$
- $S_k$ is private
Digital Signatures

- $P_k$ is made public and used to verify documents signed by $S_k$
- $S_k$ is private

Diagram:
- Document
- $S_k$
- $P_k$
- $S_k$
Digital Signatures

- $P_k$ is made public and used to verify documents signed by $S_k$
- $S_k$ is private

Diagram:
- **Document** $\rightarrow$ $S_k$ $\rightarrow$ Sign() $\rightarrow$ Signature
- $P_k$ and $S_k$ are keys, with $P_k$ being green (public) and $S_k$ being red (private).
Digital Signatures

• $P_k$ is made public and used to verify documents signed by $S_k$
• $S_k$ is private

Document $\xrightarrow{\text{Sign()}}$ Signature

$S_k$ $P_k$
Digital Signatures

- $P_k$ is made public and used to verify documents signed by $S_k$
- $S_k$ is private
Digital Signatures

- $P_k$ is made public and used to verify documents signed by $S_k$
- $S_k$ is private

Document $\xrightarrow{\text{Sign}()} S_k \xrightarrow{\text{Signature}}$

Document $\xrightarrow{\text{Verify}()} P_k \xrightarrow{\text{Valid}}$ Signature
Digital Signatures

- $P_k$ is made public and used to verify documents signed by $S_k$
- $S_k$ is private

![Diagram of digital signatures process]

- Document $\xrightarrow{Sign()} S_k$ $\xrightarrow{Signature}$
- Document $\xrightarrow{Verify()} P_k$ $\xrightarrow{Signature}$ $\rightarrow$ Valid/Invalid
Digital Signatures

- $P_k$ is made public and used to verify documents signed by $S_k$
- $S_k$ is private

Used for Authentication not privacy
Digital Signatures

- Unique to the signed document
- Mathematically hard to forge
- Mathematically easy to verify

```
Document  \rightarrow Sk \rightarrow Sign() \rightarrow Signature
```

```
Document  \rightarrow PK \rightarrow Verify() \rightarrow Valid/Invalid
```
Digital Signatures and Bitcoin

• A bitcoin is a chain of digital signatures
  • Coin owners digitally sign their coins to transfer them to other recipients
Digital Signatures and Bitcoin

• A bitcoin is a chain of digital signatures
  • Coin owners digitally sign their coins to transfer them to other recipients
  • Alice wants to move a bitcoin to Bob
Digital Signatures and Bitcoin

• A bitcoin is a chain of digital signatures
  • Coin owners digitally sign their coins to transfer them to other recipients
  • Alice wants to move a bitcoin to Bob
Digital Signatures and Bitcoin

- A bitcoin is a chain of digital signatures
  - Coin owners digitally sign their coins to transfer them to other recipients
  - Alice wants to move a bitcoin to Bob
Digital Signatures and Bitcoin

• A bitcoin is a chain of digital signatures
  • Coin owners digitally sign their coins to transfer them to other recipients
  • Alice wants to move a bitcoin to Bob
Digital Signatures and Bitcoin

• A bitcoin is a chain of digital signatures
  • Coin owners digitally sign their coins to transfer them to other recipients
  • Alice wants to move a bitcoin to Bob
Digital Signatures and Bitcoin

• A bitcoin is a chain of digital signatures
  • Coin owners digitally sign their coins to transfer them to other recipients
  • Alice wants to move a bitcoin to Bob

\[
\text{Alice} \quad \text{Sign()} \quad \text{Bob}
\]

\[
\text{Signature}_{\text{Alice-Bob}}
\]
Digital Signatures and Bitcoin

• A bitcoin is a chain of digital signatures
  • Coin owners digitally sign their coins to transfer them to other recipients
  • Alice wants to move a bitcoin to Bob

\[ \text{$P_{k-Bob}$} \quad \text{$S_{k-Alice}$} \quad \text{$P_{k-Bob}$} \]

\[ \text{Sign()} \]

\[ \text{Signature}_{\text{Alice-Bob}} \]
Digital Signatures and Bitcoin

- A bitcoin is a chain of digital signatures
  - Coin owners digitally sign their coins to transfer them to other recipients
  - Alice wants to move a bitcoin to Bob

\[ P_{k-Bob} \xrightarrow{\text{Sign()}} S_{k-Alice} \xrightarrow{P_{k-Bob}} P_{k-Alice} \xrightarrow{\text{Signature}_{Alice-Bob}} \]
Digital Signatures and Bitcoin

• A bitcoin is a chain of digital signatures
  • Coin owners digitally sign their coins to transfer them to other recipients
  • Alice wants to move a bitcoin to Bob
Digital Signatures and Bitcoin

- A bitcoin is a chain of digital signatures
  - Coin owners digitally sign their coins to transfer them to other recipients
  - Alice wants to move a bitcoin to Bob

![Diagram of digital signatures and bitcoin transaction](attachment:diagram.png)
Digital Signatures and Bitcoin

• Now what if Bob wants to move his coins to Diana
Digital Signatures and Bitcoin

• Now what if Bob wants to move his coins to Diana
Digital Signatures and Bitcoin

• Now what if Bob wants to move his coins to Diana

\[ \text{Signature}_{\text{Alice-Bob}} \]

\[ \text{P}_{k-\text{Diana}} \]
Digital Signatures and Bitcoin

• Now what if Bob wants to move his coins to Diana
Digital Signatures and Bitcoin

- Now what if Bob wants to move his coins to Diana
Digital Signatures and Bitcoin

• Now what if Bob wants to move his coins to Diana

- $\text{Signature}_{\text{Alice-Bob}}$
- $\text{Signature}_{\text{Alice-Bob}}$
- $\text{Signature}_{\text{Bob-Diana}}$
- $\text{P}_{k-Diana}$
- $\text{S}_{k-Bob}$

Sign()
A Bitcoin Big Picture
A Bitcoin Big Picture

Signature ...-Alice
A Bitcoin Big Picture

Signature \[\ldots\] -Alice

\[P_k\text{-Bob}\]
A Bitcoin Big Picture

Signature \_\_\_-Alice

\text{Sign}()

S_k-Alice

P_k-Bob
A Bitcoin Big Picture

\[ S_k-\text{Alice} \rightarrow \text{Sign()} \rightarrow \text{Signature}_{\text{Alice-Bob}} \]

\[ \text{Signature}_{\text{...-Alice}} \rightarrow P_k-\text{Bob} \]
A Bitcoin Big Picture

Signature

Alice

- 

Bob

Signature

Alice

- 

Bob

Sign()

S

k-Alice

⇒

Signature

Alice-Bob

P

k-Diana

P

k-Bob
A Bitcoin Big Picture

Signature...

Alice

\[ S_k\text{-Alice} \rightarrow \text{Sign()} \rightarrow \text{Signature}_{\text{Alice-Bob}} \rightarrow \text{Sign()} \rightarrow S_k\text{-Bob} \]

\[ P_k\text{-Bob} \rightarrow \text{Signature}_{\text{Alice-Bob}} \rightarrow P_k\text{-Diana} \]
A Bitcoin Big Picture

$\text{Sign}_{\text{Alice}} \rightarrow \text{Sign}() \rightarrow \text{Signature}_{\text{Alice-Bob}} \rightarrow \text{Sign}() \rightarrow \text{Sign}_{\text{Bob-Diana}}$
A Bitcoin Big Picture

Signature ...-Alice

$S_{k-Alice}$ \rightarrow \text{Sign()} \rightarrow \text{Signature}_{Alice-Bob} \rightarrow \text{Signature}_{Bob-Diana}

$P_{k-Bob}$

$P_{k-Diana}$

$P_{k-...}$
A Bitcoin Big Picture

- Signature
  - Alice

- $P_k$-Bob

- $S_k$-Alice
  - Sign() 

- Signature
  - Alice-Bob

- $P_k$-Diana

- $S_k$-Bob
  - Sign() 

- Signature
  - Bob-Diana

- $P_k$-... 

- $S_k$-Diana
  - Sign() 

...
What About’s?

- **Signature**: ...-Alice
  - **P_k-Bob**: Sign()
    - **S_k-Alice**: Sign()
    - **P_k-Diana**: Sign()
      - **S_k-Bob**: Sign()
        - **Signature**: Alice-Bob
          - **P_k-Diana**: ...
            - **S_k-Diana**: ...
What About’s?

What is this combination function?
What About’s?

- Signature_{Alice-Bob} → Sign()
- $S_k$-Alice → Sign()
- $S_k$-Bob → Sign()
- Signed by Diana

What is this **combination** function?

What is **double spending** and how to prevent it?
What About’s?

What is this combination function?

What is double spending and how to prevent it?

What does the first signature look like?
Hashing $H(x)$
Hashing $H(x)$

- Signatures and public keys are combined using **Hashing**
Hashing $H(x)$

- Signatures and public keys are combined using **Hashing**
- Takes *any* string $x$ of *any length* as input
- **Fixed** output size (e.g., 256 bits)
Hashing $H(x)$

- Signatures and public keys are combined using Hashing
- Takes any string $x$ of any length as input
- Fixed output size (e.g., 256 bits)
- Efficiently computable.
- Satisfies:
  - Collision Free: no two $x$, $y$ s.t. $H(x) = H(y)$
    - Message digest.
  - Hiding: Given $H(x)$ infeasible to find $x$ (one-way hash function)
    - Commitment: commit to a value and reveal later
  - Puzzle Friendly: Given a random puzzle ID and a target set $Y$ it is hard to find $x$ such that: $H(ID | x) \in Y$
Bitcoin uses SHA-256
Bitcoin uses SHA-256

SHA256( $\text{Signature}_{\text{Alice-Bob}} || P_k-\text{Diana}$ ) = 256-bit (32-byte) unique string
Bitcoin uses SHA-256

\[ \text{SHA256}( \text{Signature}_{\text{Alice-Bob}} || \text{P}_{k-Diana} ) = \text{256-bit (32-byte) unique string} \]
Bitcoin uses SHA-256

\[ \text{SHA256( } \text{Signature}_{\text{Alice-Bob}} \ || \ \text{P}_{k-\text{Diana}} \text{) } = \]

256-bit (32-byte) unique string

SHA256(abc) = ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
Bitcoin uses SHA-256

$$\text{SHA256}(\text{Signature}_{\text{Alice-Bob}} || \text{P}_k-\text{Diana}) =$$

256-bit (32-byte) unique string

\[
\text{SHA256}(\text{abc}) = ba7816bf8f01cfea41440de5dae2223b00361a396177a9cb410ff61f20015ad
\]

\[
\text{SHA256}(\text{abC}) = 0a2432a1e349d8fdb9bfca91bba9e9f2836990fe937193d84deef26c6f3b8f76
\]
What About's?

What is this combination function?

What is double spending and how to prevent it?

What does the first signature look like?
What About's?

What is this combination function? ✓

What is double spending and how to prevent it?

What does the first signature look like?
Double Spending

• Spending the same digital cash asset more than once
• Impossible to do in physical cash
• Prevented in traditional banking systems through concurrency control
Double Spending

• Spending the same digital cash asset more than once
• Impossible to do in physical cash
• Prevented in traditional banking systems through concurrency control

Signature $\text{Alice-Bob}$
Double Spending

• Spending the same digital cash asset more than once
• Impossible to do in physical cash
• Prevented in traditional banking systems through concurrency control

Signature_{Alice-Bob}
Double Spending

• Spending the same digital cash asset more than once
• Impossible to do in physical cash
• Prevented in traditional banking systems through concurrency control
Double Spending

• Spending the same digital cash asset more than once
• Impossible to do in physical cash
• Prevented in traditional banking systems through concurrency control
Double Spending

• Spending the same digital cash asset more than once
• Impossible to do in physical cash
• Prevented in traditional banking systems through concurrency control
Double Spending

• Spending the same digital cash asset more than once
• Impossible to do in physical cash
• Prevented in traditional banking systems through concurrency control
Double Spending

• Spending the same digital cash asset more than once
• Impossible to do in *physical cash*
• Prevented in traditional banking systems through *concurrency control*

\[ \text{Signature}_{\text{Alice-Bob}} \rightarrow \text{P}_{k-\text{Diana}} \]

\[ \text{Sign()} \]

\[ \text{Signature}_{\text{Bob-Diana}} \rightarrow \text{S}_{k-\text{Bob}} \]

\[ \text{I took her car} \]

\[ \text{Signature}_{\text{Bob-Marty}} \rightarrow \text{S}_{k-\text{Bob}} \]

\[ \text{I took his ring} \]
Double Spending Prevention

- Centralized
Double Spending Prevention

• Centralized
  • Transactions on coins go through a trusted 3rd party (Trent)
Double Spending Prevention

• Centralized
  • Transactions on coins go through a trusted 3rd party (Trent)

Signature_{Trent-Bob}
Double Spending Prevention

• Centralized
  • Transactions on coins go through a trusted 3rd party (Trent)

I want to transfer 20 coins to Diana
Double Spending Prevention

• Centralized
  • Transactions on coins go through a trusted 3\textsuperscript{rd} party (Trent)

I want to transfer 20 coins to Diana

50 BTC

Signature\textsubscript{Trent-Bob}

Signature\textsubscript{Trent-Bob}
Double Spending Prevention

• Centralized
  • Transactions on coins go through a trusted 3\textsuperscript{rd} party (Trent)

I want to transfer 20 coins to Diana Wasn’t spent before? Good

Signature\textsubscript{Trent-Bob}
Double Spending Prevention

- Centralized
  - Transactions on coins go through a trusted 3rd party (Trent)

I want to transfer 20 coins to Diana

Wasn’t spent before? Good

50 BTC

Signature

Trent-Bob
Double Spending Prevention

• Centralized
  • Transactions on coins go through a trusted 3rd party (Trent)

50 BTC  
Signature_{Trent-Bob}

I want to transfer 20 coins to Diana

Wasn’t spent before? Good

30 BTC  
Signature_{Trent-Bob}

20 BTC  
Signature_{Trent-Diana}
Double Spending Prevention

- Centralized
  - Transactions on coins go through a trusted 3rd party (Trent)

I want to transfer 20 coins to Diana

Wasn’t spent before? Good

Signature\textsubscript{Trent-Bob} 50 BTC

Signature\textsubscript{Trent-Bob} 30 BTC

Signature\textsubscript{Trent-Diana} 20 BTC
Double Spending Prevention

- Centralized
  - Transactions on coins go through a trusted 3rd party (Trent)

I want to transfer 20 coins to Diana

Wasn’t spent before? Good

Same old, same old!
Double Spending Prevention

• Decentralized
Double Spending Prevention

• Decentralized
  • A network of nodes maintains a ledger
Double Spending Prevention

• Decentralized
  • A network of nodes maintains a ledger
  • Network nodes work to agree on transactions order
    • Serializing transactions on every coin prevents double spending
Double Spending Prevention

• Decentralized
  • A network of nodes maintains a ledger
  • Network nodes work to agree on transactions order
    • Serializing transactions on every coin prevents double spending
  • What is the ledger?
Double Spending Prevention

• Decentralized
  • A network of nodes maintains a ledger
  • Network nodes work to agree on transactions order
    • Serializing transactions on every coin prevents double spending
  • What is the ledger?
  • How to agree on transaction order?
Double Spending Prevention

• Decentralized
  • A network of nodes maintains a ledger
  • Network nodes work to agree on transactions order
    • Serializing transactions on every coin prevents double spending
  • What is the ledger?
  • How to agree on transaction order?
  • What incentives network nodes to maintain the ledger?
What is the Ledger?
What is the Ledger?

- Blockchain
What is the Ledger?

- Blockchain

Yay!
What is the Ledger?

- Blockchain

- Transactions are grouped into blocks
What is the Ledger?

• Blockchain
  
  • Transactions are grouped into blocks
    • Blocks are chained to each other through pointers (Hence blockchain)
What is the Ledger?

• Blockchain

• Transactions are grouped into blocks
  • Blocks are chained to each other through pointers (Hence blockchain)
What is the Ledger?

• Blockchain

• Transactions are grouped into blocks
  • Blocks are chained to each other through pointers (Hence blockchain)
What is the Ledger?

- Blockchain

- Transactions are grouped into blocks
  - Blocks are chained to each other through pointers (Hence blockchain)

```
TX_1
TX_2
...  
TX_n
```
What is the Ledger?

• Blockchain

• Transactions are grouped into blocks
  • Blocks are chained to each other through pointers (Hence blockchain)
What is the Ledger?

- Blockchain

- Transactions are grouped into blocks
  - Blocks are chained to each other through pointers (Hence blockchain)
The Ledger’s What About's?
The Ledger’s What About's?

- Where is the ledger stored?
The Ledger’s What About's?

- Where is the ledger stored?
  - Each network node maintains its copy of the ledger
The Ledger’s What About's?

- Where is the ledger stored?
  - Each network node maintains its copy of the ledger
- How is the ledger tamper-free?
The Ledger’s What About's?

• Where is the ledger stored?
  - Each network node maintains its copy of the ledger

• How is the ledger tamper-free?
  1. Blocks are connected through hash-pointers
The Ledger’s What About's?

- Where is the ledger stored?
  - Each network node maintains its copy of the ledger

- How is the ledger tamper-free?
  1. Blocks are connected through hash-pointers
     - Each block contains the hash of the previous block
     - This hash gives each block its location in the blockchain
     - Tampering with the content of any block can easily be detected (is this enough? NO)
Tampering with the Ledger
Tampering with the Ledger

\[ \text{Hash()} \]

\[ TX_1 \]
\[ TX_2 \]
\[ \ldots \]
\[ TX_n \]

\[ \text{Hash()} \]

\[ TX_1 \]
\[ TX_2 \]
\[ \ldots \]
\[ TX_n \]

\[ \text{Hash()} \]

\[ TX_1 \]
\[ TX_2 \]
\[ \ldots \]
\[ TX_n \]
Tampering with the Ledger
Tampering with the Ledger

\[\text{TX}_1\]
\[\text{TX}_2\]
\[\cdots\]
\[\text{TX}_n\]
Tampering with the Ledger

Inconsistent Blockchain
Tampering with the Ledger

However,
Tampering with the Ledger

However,

Inconsistent Blockchain
Tampering with the Ledger

Consistent Blockchain

Inconsistent Blockchain

However,
The Ledger’s What About’s?

• How is the ledger tamper-free?
  1. Blocks are connected through **hash-pointers**
     • Each block contains the hash of the previous block
     • This hash gives each block its location in the blockchain
     • Tampering the content of any block can easily be detected (**is this enough? NO**)
The Ledger’s What About’s?

• How is the ledger tamper-free?

  1. Blocks are connected through hash-pointers
     • Each block contains the hash of the previous block
     • This hash gives each block its location in the blockchain
     • Tampering the content of any block can easily be detected (is this enough? NO)

  2. Replacing a consistent blockchain with another tampered consistent block chain should be made very hard, How?
Network Nodes Big Picture
Network Nodes Big Picture
Network Nodes Big Picture
Making Progress
Making Progress

• The ledger is fully replicated to all network nodes
Making Progress

• The ledger is fully replicated to all network nodes
• To make progress:
Making Progress

• The ledger is fully replicated to all network nodes
• To make progress:
  • Network nodes group new transactions into a block
Making Progress

• The ledger is fully replicated to all network nodes

• To make progress:
  • Network nodes group new transactions into a block
  • Blocks are fixed in size (1MB)
Making Progress

• The ledger is fully replicated to all network nodes

• To make progress:
  • Network nodes group new transactions into a block
  • Blocks are fixed in size (1MB)
  • Network nodes validate new transactions to make sure that:
Making Progress

• The ledger is fully replicated to all network nodes

• To make progress:
  • Network nodes group new transactions into a block
  • Blocks are fixed in size (1MB)
  • Network nodes validate new transactions to make sure that:
    • Transactions on the new block do not conflict with each other
    • Transactions on the new block do not conflict with previous blocks transactions
Making Progress

• The ledger is fully replicated to all network nodes

• To make progress:
  • Network nodes group new transactions into a block
  • Blocks are fixed in size (1MB)
  • Network nodes **validate** new transactions to make sure that:
    • Transactions on the new block do not conflict with each other
    • Transactions on the new block do not conflict with previous blocks transactions
  • Network nodes need to agree on the next block to be added to the blockchain
Making Progress

• The ledger is fully replicated to all network nodes

• To make progress:
  • Network nodes group new transactions into a block
  • Blocks are fixed in size (1MB)
  • Network nodes **validate** new transactions to make sure that:
    • Transactions on the new block **do not conflict** with each other
    • Transactions on the new block **do not conflict** with previous blocks transactions
  • Network nodes need to agree on the next block to be added to the blockchain

Consensus
Consensus

• Types of systems: synchronous and asynchronous
Consensus

• Types of systems: synchronous and asynchronous

• Problem statement: given n processes and one leader:
  • Agreement: all correct processes agree on the same value
  • Validity: If initiator does not fail, all correct processes agree on its value
Consensus

• Types of systems: synchronous and asynchronous
• Problem statement: given n processes and one leader:
  • Agreement: all correct processes agree on the same value
  • Validity: If initiator does not fail, all correct processes agree on its value
• Types of failure:
  • Crash
  • Malicious (or Byzantine)
Consensus

• Types of systems: synchronous and asynchronous

• Problem statement: given n processes and one leader:
  • Agreement: all correct processes agree on the same value
  • Validity: If initiator does not fail, all correct processes agree on its value

• Types of failure:
  • Crash
  • Malicious (or Byzantine)

• Important Impossibility Results:
Consensus

- Types of systems: synchronous and asynchronous
- Problem statement: given $n$ processes and one leader:
  - Agreement: all correct processes agree on the same value
  - Validity: If initiator does not fail, all correct processes agree on its value
- Types of failure:
  - Crash
  - Malicious (or Byzantine)
- Important Impossibility Results:
  - FLP, in asynchronous systems:
    - With even 1 crash failure, termination isn’t guaranteed (no liveness)
Consensus

• Types of systems: synchronous and asynchronous

• Problem statement: given n processes and one leader:
  • Agreement: all correct processes agree on the same value
  • Validity: If initiator does not fail, all correct processes agree on its value

• Types of failure:
  • Crash
  • Malicious (or Byzantine)

• Important Impossibility Results:
  • FLP, in asynchronous systems:
    • With even 1 crash failure, termination isn’t guaranteed (no liveness)
  • Synchronous systems:
    • Termination is guaranteed if number of failed malicious processes (f) is at most 1/3 n
(Multi-) Paxos
(Multi-) Paxos

- Paxos is a consensus algorithm
  - Processes want to agree on a value (e.g., the next block to be added to the chain)
(Multi-) Paxos

• Paxos is a consensus algorithm
  • Processes want to agree on a value (e.g., the next block to be added to the chain)
• Paxos is currently used to manage local data in global-scale systems
  • Spanner [OSDI’12, SIGMOD’17], Megastore [CIDR’11], etc
(Multi-) Paxos

• Paxos is a consensus algorithm
  • Processes want to agree on a value (e.g., the next block to be added to the chain)
• Paxos is currently used to manage local data in global-scale systems
  • Spanner [OSDI’12, SIGMOD’17], Megastore [CIDR’11], etc
• Multi-Paxos, simplified:
(Multi-) Paxos

• Paxos is a consensus algorithm
  • Processes want to agree on a value (e.g., the next block to be added to the chain)

• Paxos is currently used to manage local data in global-scale systems
  • Spanner [OSDI’12, SIGMOD’17], Megastore [CIDR’11], etc

• Multi-Paxos, simplified:
  • Initially, a leader is elected by a majority quorum
(Multi-) Paxos

- Paxos is a consensus algorithm
  - Processes want to agree on a value (e.g., the next block to be added to the chain)
- Paxos is currently used to manage local data in global-scale systems
  - Spanner [OSDI’12, SIGMOD’17], Megastore [CIDR’11], etc
- Multi-Paxos, simplified:
  - Initially, a leader is elected by a majority quorum
  - **Replication**: Leader replicates new updates to a majority quorum
(Multi-) Paxos

- Paxos is a consensus algorithm
  - Processes want to agree on a value (e.g., the next block to be added to the chain)
- Paxos is currently used to manage local data in global-scale systems
  - Spanner [OSDI’12, SIGMOD’17], Megastore [CIDR’11], etc
- Multi-Paxos, simplified:
  - Initially, a leader is elected by a majority quorum
  - **Replication**: Leader replicates new updates to a majority quorum
(Multi-) Paxos

- Paxos is a consensus algorithm
  - Processes want to agree on a value (e.g., the next block to be added to the chain)
- Paxos is currently used to manage local data in global-scale systems
  - Spanner [OSDI’12, SIGMOD’17], Megastore [CIDR’11], etc
- Multi-Paxos, simplified:
  - Initially, a leader is elected by a majority quorum
  - Replication: Leader replicates new updates to a majority quorum
(Multi-) Paxos

• Paxos is a consensus algorithm
  • Processes want to agree on a value (e.g., the next block to be added to the chain)
• Paxos is currently used to manage local data in global-scale systems
  • Spanner [OSDI’12, SIGMOD’17], Megastore [CIDR’11], etc
• Multi-Paxos, simplified:
  • Initially, a leader is elected by a majority quorum
  • Replication: Leader replicates new updates to a majority quorum
  • Leader Election: If the leader fails, a new leader is elected
Can Network Nodes Use Paxos?
Can Network Nodes Use Paxos?
Can Network Nodes Use Paxos?
Paxos Consensus
Paxos Consensus

• All **participants** should be known *a priori*
Paxos Consensus

• All participants should be known a priori
  • Permissioned vs Permissionless settings
Paxos Consensus

• All participants should be known a priori
  • Permissioned vs Permissionless settings
  • Permissionless setting:
    • Network nodes freely join or leave the network at anytime
Paxos Consensus

• All participants should be known a priori
  • Permissioned vs Permissionless settings
  • Permissionless setting:
    • Network nodes freely join or leave the network at anytime

• Tolerates only Crash failures
Paxos Consensus

• All **participants** should be known **a priori**
  • Permissioned vs Permissionless settings
  • Permissionless setting:
    • Network nodes freely join or leave the network at anytime

• Tolerates only **Crash** failures
  • However, network nodes can be **Malicious**
Paxos Consensus

• All participants should be known *a priori*
  • Permissioned vs Permissionless settings
  • Permissionless setting:
    • Network nodes freely join or leave the network at anytime

• Tolerates only **Crash** failures
  • However, network nodes can be **Malicious**
  • To make progress, at least **1/2** of the participants should be **alive**
  • Progress is not guaranteed (FLP impossibility)
Paxos Consensus

• All participants should be known \textit{a priori}
  • Permissioned vs Permissionless settings
  • Permissionless setting:
    • Network nodes freely join or leave the network at anytime

• Tolerates only \textbf{Crash} failures
  • However, network nodes can be \textbf{Malicious}
  • To make progress, at least $1/2$ of the participants should be \textbf{alive}
  • Progress is not guaranteed (FLP impossibility)

• Also, Paxos has high network overhead
Practical Byzantine Fault Tolerance (PBFT)
Practical Byzantine Fault Tolerance (PBFT)

- Goal: Implement a deterministic replication service with arbitrary malicious faults in an asynchronous environment
Practical Byzantine Fault Tolerance (PBFT)

• Goal: Implement a deterministic replication service with arbitrary malicious faults in an asynchronous environment
• No assumptions about faulty behavior
• No bounds on delays
Practical Byzantine Fault Tolerance (PBFT)

• Goal: Implement a deterministic replication service with arbitrary malicious faults in an asynchronous environment
• No assumptions about faulty behavior
• No bounds on delays
• Provides safety in asynchronous system and assume eventual time bounds for liveness
Practical Byzantine Fault Tolerance (PBFT)

• Goal: Implement a deterministic replication service with arbitrary malicious faults in an asynchronous environment
• No assumptions about faulty behavior
• No bounds on delays
• Provides safety in asynchronous system and assume eventual time bounds for liveness
• Assumptions:
Practical Byzantine Fault Tolerance (PBFT)

• Goal: Implement a deterministic replication service with arbitrary malicious faults in an asynchronous environment
• No assumptions about faulty behavior
• No bounds on delays
• Provides safety in asynchronous system and assume eventual time bounds for liveness
• Assumptions:
  • 3f+1 replicas to tolerate f Byzantine faults (optimal)
Practical Byzantine Fault Tolerance (PBFT)

• Goal: Implement a deterministic replication service with arbitrary malicious faults in an asynchronous environment
• No assumptions about faulty behavior
• No bounds on delays
• Provides safety in asynchronous system and assume eventual time bounds for liveness
• Assumptions:
  • 3f+1 replicas to tolerate f Byzantine faults (optimal)
    • quorums have at least 2f+1 replicas
    • quorums intersect in f+1, hence have at least one correct replica
  • Strong cryptography
  • Only for liveness: eventual time bounds
Algorithm

The algorithm has three main phases: (1) *pre-prepare* picks order of requests  (2) *prepare* ensures order within views, (3) *commit* ensures order across views

- replica 0 (Primary)
- replica 1
- replica 2
- replica 3 *fail*
Algorithm

The algorithm has three main phases: (1) \textit{pre-prepare} picks order of requests (2) \textit{prepare} ensures order within views, (3) \textit{commit} ensures order across views

(1) A client sends a request for a service to the primary
Algorithm

The algorithm has three main phases: (1) _pre-prepare_ picks order of requests, (2) _prepare_ ensures order within views, (3) _commit_ ensures order across views.
Algorithm

The algorithm has three main phases: (1) *pre-prepare* picks order of requests  (2) *prepare* ensures order within views, (3) *commit* ensures order across views

(2) The primary multicasts the request to the backups
Algorithm

The algorithm has three main phases: (1) *pre-prepare* picks order of requests  (2) *prepare* ensures order within views, (3) *commit* ensures order across views
The algorithm has three main phases: (1) *pre-prepare* picks order of requests  (2) *prepare* ensures order within views, (3) *commit* ensures order across views

(3) Backups multicast *PREPARE* message
Algorithm

The algorithm has three main phases: (1) \textit{pre-prepare} picks order of requests  (2) \textit{prepare} ensures order within views, (3) \textit{commit} ensures order across views
Algorithm

The algorithm has three main phases: (1) **pre-prepare** picks order of requests  (2) **prepare** ensures order within views, (3) **commit** ensures order across views

(4) If a replica receives at least $2f$ matching PREPARE message, multicasts a COMMIT message
Algorithm

The algorithm has three main phases: (1) *pre-prepare* picks order of requests  (2) *prepare* ensures order within views, (3) *commit* ensures order across views
Algorithm

The algorithm has three main phases: (1) pre-prepare picks order of requests  (2) prepare ensures order within views, (3) commit ensures order across views

(5) If a replica receives at least 2f COMMIT messages, reply the result to the client
Algorithm

The algorithm has three main phases: (1) **pre-prepare** picks order of requests  (2) **prepare** ensures order within views, (3) **commit** ensures order across views

(6) The client waits for **f+1** replies from different replicas with the **same** result
PBFT Consensus

- Tolerates **Byzantine (Malicious)** failures
  - To make progress, at least $2/3$ of the participants should be **correct**
  - Progress is not guaranteed (FLP impossibility)

- However, PBFT is **Permissioned**
  - All participants should be known **a priori**

- Also, PBFT has high network overhead $O(N^2)$ [number of messages]
  - Every node multi-casts their responses to every other node
Nakamoto’s Consensus

• Intuitively, network nodes race to solve a puzzle
• This puzzle is computationally expensive
• Once a network node finds (mines) a solution:
  • It adds its block of transactions to the blockchain
  • It multi-casts the solution to other network nodes
  • Other network nodes accept and verify the solution
Mining Details
Mining Details
Mining Details

TX_1
TX_2
...
TX_n
Mining Details
Mining Details
Mining Details
Mining Details

TX_1
TX_2
·
TX_n
Mining Details

TX_1 → TX_2 → ... → TX_n

TX_1 → TX_2 → ... → TX_n

TX_1

TX_1 → TX_2 → ... → TX_n
Mining Details
Mining Details
Mining Details

TX_1
TX_2
. .
TX_n

TX_1
TX_2
. .
TX_n

TX_1
TX_2
. .

TX_n
Mining Details

TX_1
TX_2
...  
TX_n

TX_1
TX_2
...  
TX_n

TX_1
TX_2
...  
TX_n
Mining Details

TX1
TX2
...
TXn

TX1
TX2
...
TXn

TX1
TX2
...
TXn
Mining Details
Mining Details
Mining Details

TX_1
TX_2
...
TX_n

TX_1
TX_2
...
TX_n

TX_1
TX_2
TX_reward
...
TX_n
Mining Details

Transactions
Mining Details

Transactions

Header

- Version
- Previous Block Header Hash
- Merkle Tree Root Hash
- Time Stamp
- Current Target Bits
- Nonce

TX1
TX2
.. 
TXn

TX1
TX2
.. 
TXn

TX1
TX2
.. 
TXn

TX\text{reward}
Mining Details

Transactions

TX_1
TX_2
...
TX_n

TX_1
TX_2
...
TX_n

TX_1
TX_2
...
TX_n

TX reward
TX_1
TX_2
...
TX_n

SHA256(Header) < D

Version
Previous Block Header Hash
Merkle Tree Root Hash
Time Stamp
Current Target Bits
Nonce
Mining Details

SHA256(
TX_{\text{reward}}

Transactions

<table>
<thead>
<tr>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Block Header Hash</td>
</tr>
<tr>
<td>Merkle Tree Root Hash</td>
</tr>
<tr>
<td>Time Stamp</td>
</tr>
<tr>
<td>Current Target Bits</td>
</tr>
<tr>
<td>Nonce</td>
</tr>
</tbody>
</table>

Header < D
Mining Details

SHA256(

Transactions

Version
Previous Block Header Hash
Merkle Tree Root Hash
Time Stamp
Current Target Bits
Nonce

D < D
Mining Details

- $TX_{\text{reward}}$ is self signed (also called coinbase transaction)
- First signature? Self signed 😊

SHA256($TX_{\text{reward}}$)

Transactions

Header

- Version
- Previous Block Header Hash
- Merkle Tree Root Hash
- Time Stamp
- Current Target Bits
- Nonce

$TX_{\text{reward}}$, $TX_1$, $TX_2$, ..., $TX_n$
Mining Details

- \( \text{TX}_{\text{reward}} \) is self signed (also called coinbase transaction)
- First signature? Self signed 😊
- \( \text{TX}_{\text{reward}} \) is bitcoin’s way to create new coins

\[
\text{SHA256}(\text{TX}_{\text{reward}}) < D
\]

Transactions

- Version
- Previous Block Header Hash
- Merkle Tree Root Hash
- Time Stamp
- Current Target Bits
- Nonce

\[
\begin{array}{c}
\text{TX}_1 \\
\text{TX}_2 \\
\vdots \\
\text{TX}_n \\
\text{TX}_{\text{reward}} \\
\text{TX}_1 \\
\text{TX}_2 \\
\vdots \\
\text{TX}_n \\
\end{array}
\]

Header
Mining Details

- $\text{TX}_{\text{reward}}$ is self signed (also called coinbase transaction)
- First signature? Self signed 😊
- $\text{TX}_{\text{reward}}$ is bitcoin’s way to create new coins
- The reward value is halved every 4 years (210,000 blocks)
Mining Details

- $\text{TX}_{\text{reward}}$ is self signed (also called coinbase transaction)
- First signature? Self signed 😊
- $\text{TX}_{\text{reward}}$ is bitcoin’s way to create new coins
- The reward value is halved every 4 years (210,000 blocks)
- Currently, it’s 12.5 Bitcoins per block
Mining Details

- $\text{TX}_{\text{reward}}$ is self signed (also called coinbase transaction)
- First signature? Self signed 😊
- $\text{TX}_{\text{reward}}$ is bitcoin’s way to create new coins
- The reward value is halved every 4 years (210,000 blocks)
- Currently, it’s 12.5 Bitcoins per block
- Incentives network nodes to mine

Transactions

<table>
<thead>
<tr>
<th>Version</th>
<th>Previous Block Header Hash</th>
<th>Merkle Tree Root Hash</th>
<th>Time Stamp</th>
<th>Current Target Bits</th>
<th>Nonce</th>
</tr>
</thead>
<tbody>
<tr>
<td>[SHA256( $\text{TX}_{\text{reward}}$ ) $\text{TX}_1$ $\text{TX}_2$ $\text{TX}_n$] &lt; D</td>
<td>[SHA256( $\text{TX}_1$ $\text{TX}_2$ $\text{TX}_n$)]</td>
<td>[SHA256( $\text{TX}_1$ $\text{TX}_2$ $\text{TX}_n$)]</td>
<td>[SHA256( $\text{TX}_1$ $\text{TX}_2$ $\text{TX}_n$)]</td>
<td>[SHA256( $\text{TX}_1$ $\text{TX}_2$ $\text{TX}_n$)]</td>
<td>[SHA256( $\text{TX}_1$ $\text{TX}_2$ $\text{TX}_n$)]</td>
</tr>
</tbody>
</table>

• $\text{TX}_{\text{reward}}$ is bitcoin’s way to create new coins
• The reward value is halved every 4 years (210,000 blocks)
• Currently, it’s 12.5 Bitcoins per block
• Incentives network nodes to mine
Mining Details

SHA256(TX\_reward) < D

Transactions

Version
Previous Block Header Hash
Merkle Tree Root Hash
Time Stamp
Current Target Bits
Nonce

TX\_1
TX\_2
\ldots
TX\_n

TX\_1
TX\_2
\ldots
TX\_n

TX\_1
TX\_2
\ldots
TX\_n
Mining Details

Transactions

Header

<table>
<thead>
<tr>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Block Header Hash</td>
</tr>
<tr>
<td>Merkle Tree Root Hash</td>
</tr>
<tr>
<td>Time Stamp</td>
</tr>
<tr>
<td>Current Target Bits</td>
</tr>
<tr>
<td>Nonce</td>
</tr>
</tbody>
</table>

SHA256( TX\_reward )
Mining Details

- D: dynamically adjusted difficulty
Mining Details

- D: dynamically adjusted difficulty
  256 bits

SHA256(TX\textsubscript{reward})

Transactions

<table>
<thead>
<tr>
<th>Header</th>
<th>Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>TX\textsubscript{reward}</td>
</tr>
<tr>
<td>Previous Block Header Hash</td>
<td>TX\textsubscript{1}</td>
</tr>
<tr>
<td>Merkle Tree Root Hash</td>
<td>TX\textsubscript{2}</td>
</tr>
<tr>
<td>Time Stamp</td>
<td>... TX\textsubscript{n}</td>
</tr>
<tr>
<td>Current Target Bits</td>
<td>... TX\textsubscript{n}</td>
</tr>
<tr>
<td>Nonce</td>
<td>... TX\textsubscript{n}</td>
</tr>
</tbody>
</table>
Mining Details

- D: dynamically adjusted difficulty
  - 256 bits

SHA256(

Transactions

- Version
- Previous Block Header Hash
- Merkle Tree Root Hash
- Time Stamp
- Current Target Bits
- Nonce

Transactions

TX\_1
TX\_2
\ldots
TX\_n
Mining Details

- D: dynamically adjusted difficulty
  256 bits
- Difficulty bits
- Difficulty is adjusted every 2016 blocks (almost 2 weeks)

SHA256(

Transactions

Header

<table>
<thead>
<tr>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Block Header Hash</td>
</tr>
<tr>
<td>Merkle Tree Root Hash</td>
</tr>
<tr>
<td>Time Stamp</td>
</tr>
<tr>
<td>Current Target Bits</td>
</tr>
<tr>
<td>Nonce</td>
</tr>
</tbody>
</table>

TX\_1
TX\_2
... 
TX\_n

TX\_reward
TX\_1
TX\_2
... 
TX\_n
Difficulty
Difficulty

- Adjust difficulty every 2016 blocks
Difficulty

• Adjust difficulty every 2016 blocks
• Expected 20160 mins to mine (10 mins per block)
Difficulty

• Adjust difficulty every 2016 blocks
• Expected 20160 mins to mine (10 mins per block)
• Actual time = timestamp of block 2016 – time stamp of block 1
Difficulty

• Adjust difficulty every 2016 blocks
• **Expected** 20160 mins to mine (10 mins per block)
• **Actual** time = timestamp of block 2016 – time stamp of block 1
• New_difficulty = old_difficulty * **expected/actual**
Difficulty

• Adjust difficulty every 2016 blocks
• **Expected** 20160 mins to mine (10 mins per block)
• **Actual** time = timestamp of block 2016 – time stamp of block 1
• New_difficulty = old_difficulty * expected/actual
• Difficulty decreases if actual > expected, otherwise, increases
Mining Big Picture
Mining Big Picture
Mining Big Picture
Mining Big Picture
Mining Details

• Find a **nonce** that results in SHA256(block) < Difficulty
Mining Details

• Find a **nonce** that results in \( \text{SHA256}(\text{block}) < \text{Difficulty} \)
• The solution space is a **set**. Once a solution is found, a block is mined
Mining Details

• Find a nonce that results in SHA256(block) < Difficulty
• The solution space is a set. Once a solution is found, a block is mined
• Easily verified by network nodes
Mining Details

- Find a **nonce** that results in $\text{SHA256(block)} < \text{Difficulty}$
- The solution space is a **set**. Once a solution is found, a block is mined
- Easily verified by network nodes
- Cannot be precomputed
  - Depends on current block transactions and previous blocks
Mining Details

• Find a **nonce** that results in $\text{SHA256(block)} < \text{Difficulty}$
• The solution space is a **set**. Once a solution is found, a block is mined
• Easily verified by network nodes
• Cannot be precomputed
  • Depends on current block transactions and previous blocks
• Cannot be stolen
  • Reward Transaction is signed to the public key of the miner
Mining Details

• Find a **nonce** that results in SHA256(block) < Difficulty
• The solution space is a **set**. Once a solution is found, a block is mined
• Easily verified by network nodes
• Cannot be precomputed
  • Depends on current block transactions and previous blocks
• Cannot be stolen
  • Reward Transaction is signed to the public key of the miner
• Network nodes accept the first found block:
  • The problem is difficult, there is no guaranteed bound to find another block
Mining Details

• Find a **nonce** that results in SHA256(block) < Difficulty
• The solution space is a **set**. Once a solution is found, a block is mined
• Easily verified by network nodes
• Cannot be precomputed
  • Depends on current block transactions and previous blocks
• Cannot be stolen
  • Reward Transaction is signed to the public key of the miner
• Network nodes accept the first found block:
  • The problem is difficult, there is no guaranteed bound to find another block
• What happens when 2 nodes concurrently mine a block? **Fork**
Mining Details

• Find a nonce that results in SHA256(block) < Difficulty
Mining Details

- Find a **nonce** that results in $\text{SHA256(block)} < \text{Difficulty}$

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (4B)</td>
<td>02000000</td>
</tr>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D1D09A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td></td>
</tr>
<tr>
<td>TX&lt;sub&gt;reward&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>TX&lt;sub&gt;1&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
</tr>
<tr>
<td>TX&lt;sub&gt;n&lt;/sub&gt;</td>
<td></td>
</tr>
</tbody>
</table>
Mining Details

• Find a **nonce** that results in SHA256(block) < Difficulty

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (4B)</td>
<td>020000000</td>
</tr>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC4</td>
</tr>
<tr>
<td></td>
<td>AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA</td>
</tr>
<tr>
<td></td>
<td>145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td></td>
</tr>
</tbody>
</table>

The **difficulty** is a function of Current Target Bits: (Largest possible Target/Current Target)

000000000000000000cf3620d570d08d1799a1cafbbf18 zeros 80eca0

Between transactions:

- TX\(_{\text{reward}}\)
- TX\(_1\)
- ...
- TX\(_n\)
Mining Details

• Find a nonce that results in SHA256(block) < Difficulty

<table>
<thead>
<tr>
<th>Version (4B)</th>
<th>02000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{Difficulty is a function of Current Target Bits (Largest possible Target/Current Target)}
\]

\[
\text{SHA256(V,P,M,T,C,0) = BD72804EE251889F9013C100767999B57E92EC5B6ADBDBF64F2DF1B032429C72}
\]

TX\_\text{reward}
TX\_1
.
.
TX\_n
Mining Details

• Find a **nonce** that results in SHA256(block) < Difficulty

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (4B)</td>
<td>02000000</td>
</tr>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td>TX_{reward} TX_1 . . . TX_n</td>
</tr>
</tbody>
</table>

Difficulty is a function of Current Target Bits (Largest possible Target/Current Target)

000000000000000000cf3620d570d08d1799a1cafbbfae512fdb82124665eca0

18 zeros

SHA256(V,P,M,T,C,0) = BD72804EE251889F9013C100767999B57E92EC5B6ADBDBF64F2DF1B0324
Mining Details

• Find a **nonce** that results in SHA256(block) < Difficulty

<table>
<thead>
<tr>
<th>Version (4B)</th>
<th>02000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td>TX\textsubscript{reward} TX\textsubscript{1} . . . TX\textsubscript{n}</td>
</tr>
</tbody>
</table>

Difficulty is a function of Current Target Bits (Largest possible Target/Current Target)

000000000000000000cf3620d570d08d1799a1cafbbfae512fdba2124665eca0

18 zeros

SHA256\((V,P,M,T,C,0)\) = BD72804EE251889F9013C100767999B57E92EC5B6ADDB8F64F2DF1B0324
SHA256\((V,P,M,T,C,1)\) = DF64342507E785FDC04C776D7142BB2BC6467F09E004A3E9F65E38872A45D8
# Mining Details

- **Find a nonce** that results in \( \text{SHA256(block)} < \text{Difficulty} \)

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (4B)</td>
<td>02000000</td>
</tr>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{TX}_{\text{reward}} \]
\[ \text{TX}_1 \]
\[ \vdots \]
\[ \text{TX}_n \]

\[ \text{Difficulty is a function of Current Target Bits (Largest possible Target/Current Target)} \]

\[ 000000000000000000cf362d570d08d1799a1cafbbfae512fdba2124665eca0 \]

18 zeros

\[ \text{SHA256}(V,P,M,T,C,0) = \]
\[ \text{BD72804EE251889F9013C100767999B57E92EC5B6ADBDBF64F2DF1B0324} \]

\[ \text{SHA256}(V,P,M,T,C,1) = \]
\[ \text{DF64342507E785FDC0D4C776D7142BB2BC6467F09E0040A3E9F65E38872} \]
## Mining Details

- **Find a nonce** that results in $\text{SHA256(block)} < \text{Difficulty}$

### Table: Blockchain Data

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (4B)</td>
<td>02000000</td>
</tr>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935B909D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td>$\text{TX}_{\text{reward}}$ $\text{TX}_1$ $\text{TX}_n$</td>
</tr>
</tbody>
</table>

**Difficulty** is a function of Current Target Bits (Largest possible Target/Current Target)

000000000000000000cf3620d570d08d1799a1ca6bbf2ba512f6f2124665eca0

### SHA256 Examples

- $\text{SHA256}(V, P, M, T, C, 0) = \text{BD72804EE251889F9013C100767999B57E92EC5B6ADBDBF64F2DF1B0324}$
- $\text{SHA256}(V, P, M, T, C, 1) = \text{DF64342507E785FDC0D4C776D7142BB2BC6467F09E0040A3E9F65E38872}$
- $\text{SHA256}(V, P, M, T, C, 2) = \text{0000000CC7F94221B95F4E606E037D31C10417435DEE60A61C627B64324590FE}$
## Mining Details

- **Find a **nonce **that results in** SHA256(block) < Difficulty

<table>
<thead>
<tr>
<th>Version (4B)</th>
<th>02000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td>TX&lt;sub&gt;reward&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td>TX&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td></td>
<td>TX&lt;sub&gt;n&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

**Difficulty** is a function of Current Target Bits (Largest possible Target/Current Target)

- SHA256(V,P,M,T,C,0) = BD72804EE251889F9013C100767999B57E92EC5B6ADBDBF64F2DF1B0324
- SHA256(V,P,M,T,C,2) = 00000000CC7F94221B95F4E606E037D31C10417435DEE60A61C627B64324

---

Legend:
- **18 zeros**
- **7 zeros**
Mining Details

- Find a **nonce** that results in $\text{SHA256}(\text{block}) < \text{Difficulty}$

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (4B)</td>
<td>02000000</td>
</tr>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td>TX\textsubscript{reward} TX\textsubscript{1} \cdots TX\textsubscript{n}</td>
</tr>
</tbody>
</table>

**Difficulty** is a function of Current Target Bits \( \text{(Largest possible Target/Current Target)} \)

- SHA256\((V,P,M,T,C,0)\) = BD72804EE251889F9013C100767999B57E92EC5B6ADBDF64F2DF1B0324
- SHA256\((V,P,M,T,C,1)\) = DFE4342507E785FDC0D4C776D7142BB2BC6467F09E0040A3E9F65E38872
- SHA256\((V,P,M,T,C,2)\) = 00000000CC7F94221B95F4E606E037D31C10417435DEE60A61C627B64324

- SHA256\((V,P,M,T,C,01F04A1C)\) = 000000000000000000000001E3BFE56AD29732B81128B79356442C8B87F6CED8B6610
# Mining Details

- **Find a **nonce** that results in SHA256(block) < Difficulty**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version</strong></td>
<td>0200000000</td>
</tr>
<tr>
<td><strong>Previous Block Hash</strong></td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td><strong>Merkle Tree Root Hash</strong></td>
<td>4E04D109A3A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td><strong>Time Stamp</strong></td>
<td>5C9F3E20</td>
</tr>
<tr>
<td><strong>Current Target Bits</strong></td>
<td>172E6117</td>
</tr>
<tr>
<td><strong>Nonce</strong></td>
<td>TX_reward TX_1 . TX_n</td>
</tr>
<tr>
<td><strong>Difficulty</strong></td>
<td>18 zeros</td>
</tr>
</tbody>
</table>

Difficulty is a function of Current Target Bits (Largest possible Target/Current Target)

- SHA256(V,P,M,T,C,0) = BD72804EE251889F9013C100767999B57E92EC5B6ADBDBF64F2DF1B0324
- SHA256(V,P,M,T,C,1) = DF64342507E785FDC0D4C776D7142BB2BC6467F09E0040A3E9F65E38872
- SHA256(V,P,M,T,C,2) = 0000000000000001E3BF56AD29732B81128B79356442C8B87F6CED8

- SHA256(V,P,M,T,C,01F04A1C) = 000000000000000001E3BF56AD29732B81128B79356442C8B87F6CED8
Forks
Forks
Transactions in the forked blocks might have conflicts.
Transactions in the forked blocks might have conflicts
Could lead to double spending
Transactions in the forked blocks might have conflicts
Could lead to double spending
Forks have to be eliminated
Forks
Forks
Forks
Forks
Forks
Forks
• Miners join the longest chain to resolve forks
Forks
Forks

- Transactions in this block have to be resubmitted
Forks

- Transactions in this block have to be resubmitted
Forks: The Big Picture
Forks: The Big Picture
Forks: The Big Picture
Forks: The Big Picture
Forks: The Big Picture
Forks: The Big Picture

Abandoned
Longest Chain
Forks: The Big Picture

- Abandoned
- Longest Chain
Forks: The Big Picture

- Abandoned
- Longest Chain
Forks: The Big Picture

- Abandoned
- Longest Chain
51% Attack

- If 51% of the computation (hash) power are malicious:
  - They can cooperate to fork the chain at any block
- Can lead to double spending
51% Attack

• If 51% of the computation (hash) power are malicious:
  • They can cooperate to fork the chain at any block
• Can lead to double spending
51% Attack

- If 51% of the computation (hash) power are malicious:
  - They can cooperate to fork the chain at any block
- Can lead to double spending
51% Attack

• If 51% of the computation (hash) power are malicious:
  • They can cooperate to fork the chain at any block
• Can lead to double spending
51% Attack

- If 51% of the computation (hash) power are malicious:
  - They can cooperate to fork the chain at any block
- Can lead to double spending
51% Attack

• If 51% of the computation (hash) power are malicious:
  • They can cooperate to fork the chain at any block

• Can lead to double spending
51% Attack

• If 51% of the computation (hash) power are malicious:
  • They can cooperate to fork the chain at any block
• Can lead to double spending
51% Attack

• If 51% of the computation (hash) power are malicious:
  • They can cooperate to fork the chain at any block
• Can lead to double spending
51% Attack

• If 51% of the computation (hash) power are malicious:
  • They can cooperate to fork the chain at any block
• Can lead to double spending
51% Attack

- If 51% of the computation (hash) power are malicious:
  - They can cooperate to fork the chain at any block
- Can lead to double spending
Selfish Mining

Selfish Mining

Selfish Mining

Selfish Mining

Selfish Mining


- Block found, yay!
Selfish Mining

- Block found, yay!
- Don’t immediately announce it

Selfish Mining

- Block found, yay!
- Don’t immediately announce it
- Let honest miners waste their mining power on an obsolete block

Selfish Mining

• Block found, yay!
• Don’t immediately announce it
• Let honest miners waste their mining power on an obsolete block
• Start mining the next block (Advantage)

Selfish Mining

- Block found, yay!
- Don’t immediately announce it
- Let honest miners waste their mining power on an obsolete block
- Start mining the next block (Advantage)

Selfish Mining

- Block found, yay!
- Don’t immediately announce it
- Let honest miners waste their mining power on an obsolete block
- Start mining the next block (Advantage)

Selfish Mining

- Block found, yay!
- Don’t immediately announce it
- Let honest miners waste their mining power on an obsolete block
- Start mining the next block (Advantage)

Selfish Mining

- Block found, yay!
- Don’t immediately announce it
- Let honest miners waste their mining power on an obsolete block
- Start mining the next block (Advantage)
- Two possible outcomes

Selfish Mining

- First Outcome

Selfish Mining

• First Outcome
  • Selfish miner finds the following block first

Selfish Mining

• First Outcome
  • Selfish miner finds the following block first
  • Once an honest miner finds a block

Selfish Mining

- First Outcome
  - Selfish miner finds the following block first
  - Once an honest miner finds a block
    - Selfish miner announces 2 blocks

Selfish Mining

- First Outcome
  - Selfish miner finds the following block first
  - Once an honest miner finds a block
    - Selfish miner announces 2 blocks
    - Honest miner loses the reward

Selfish Mining

• First Outcome
  • Selfish miner finds the following block first
  • Once an honest miner finds a block
    • Selfish miner announces 2 blocks
  • Honest miner loses the reward

Selfish Mining

Selfish Mining

- Second Outcome

Selfish Mining

- Second Outcome
  - An honest miner finds a block first

Selfish Mining

• Second Outcome
  • An honest miner finds a block first
  • Selfish miner immediately announces the previously found block

Selfish Mining

- Second Outcome
  - An honest miner finds a block first
  - Selfish miner immediately announces the previously found block
  - This splits the power of honest miners

Selfish Mining

- Second Outcome
  - An honest miner finds a block first
  - Selfish miner immediately announces the previously found block
  - This splits the power of honest miners

Selfish Mining

- If selfish miner successfully splits honest miners:
  - The probability of finding the next red block is 2/3 (secures the reward of the previously found block)

Selfish Mining

- Also,
  - The probability of selfish miner to find the next red block is 1/2 even if selfish miner has 1/3 of the mining resources (Advantage)

Limitations of Bitcoin
Limitations of Bitcoin

• High transaction-confirmation latency
Limitations of Bitcoin

• High transaction-confirmation latency

• Probabilistic consistency guarantees
Limitations of Bitcoin

• High transaction-confirmation latency

• Probabilistic consistency guarantees

• Very low TPS (Transactions per second) - average of 3 to 7 TPS
Limitations of Bitcoin

• High transaction-confirmation **latency**

• **Probabilistic** consistency guarantees

• **Very low TPS** (Transactions per second) - average of 3 to 7 TPS

• New block added every **10 minutes**.
How to scale Bitcoin?
How to scale Bitcoin?

• Two obvious options for increasing Bitcoin’s transaction throughput:
How to scale Bitcoin?

• Two obvious options for increasing Bitcoin’s transaction throughput: 
  increase the size of blocks, or decrease the block interval
Increasing Block Size
Increasing Block Size
Increasing Block Size

1MB/10 mins
1MB = 4200 Txns
7 Txns/ second
Increasing Block Size

- 1MB/10 mins
  - 1MB = 4200 Txns
  - 7 Txns/ second

- 10MB/10 mins
  - 10MB = 42000 Txns
  - 70 Txns/ second
Increasing Block Size

- **1MB/10 mins**
  - 1MB = 4200 Txns
  - 7 Txns/ second

- **10MB/10 mins**
  - 10MB = 42000 Txns
  - 70 Txns/ second

- **100MB/10 mins**
  - 100MB = 420000 Txns
  - 700 Txns/ second
Increasing Block Size

1MB/10 mins
1MB = 4200 Txns
7 Txns/ second

10MB/10 mins
10MB = 42000 Txns
70 Txns/ second

100MB/10 mins
100MB = 420000 Txns
700 Txns/ second

........
Increasing Block Size

• Why they don’t work?

  • Decreases fairness - giving large miners an advantage
  • Requires more storage space (1 → 10 → 100 MB/ 10 mins)
  • Requires more Network bandwidth
  • Requires more verification time
Decrease Block Interval
Decrease Block Interval
Decrease Block Interval

1MB/10 mins
1MB = 4200 Txns
7 Txns/ second
Decrease Block Interval

1MB/10 mins
1MB = 4200Txns
7Txns/second

1MB/5 mins
1MB = 4200 Txns
14 Txns/second
Decrease Block Interval

1MB/10 mins
1MB = 4200 Txns
7 Txns/ second

1MB/5 mins
1MB = 4200 Txns
14 Txns/ second

1MB/1 min
1MB = 4200 Txns
70 Txns/ second
Decrease Block Interval

- 1MB/10 mins
- 1MB = 4200 Txns
- 7 Txns/ second

- 1MB/5 mins
- 1MB = 4200 Txns
- 14 Txns/ second

- 1MB/1 min
- 1MB = 4200 Txns
- 70 Txns/ second
Decrease Block Interval

• Requires to mining decrease difficulty
• Leads to more forks
• Results on network instability (many branches)
Overview

- Increase throughput by reducing consensus from all nodes to smaller set
Overview

- Increase throughput by reducing consensus from all nodes to smaller set

Mine once, publish txns many times
Overview

• Increase throughput by reducing consensus from all nodes to smaller set

- Mine once, publish txns many times
  - BitcoinNG

- Form a committee to vouch for new block
  - ByzCoin
Overview

- Increase throughput by reducing consensus from all nodes to smaller set

- Mine once, publish txns many times
- Form a committee to vouch for new block
- Shard txns across different committees
BitcoinNG (Next Generation)
BitcoinNG (Next Generation)

Observation: In Bitcoin, blocks provide two purpose: consensus and txn verification.
Observation: In Bitcoin, blocks provide two purpose: consensus and txn verification.
Observation: In Bitcoin, blocks provide two purpose: consensus and txn verification.

Keyblocks:
Used for Leader Election and created using Proof-of-work.
Observation: In Bitcoin, blocks provide two purpose: **consensus** and **txn verification**

**Keyblocks**: Used for Leader Election and created using Proof-of-work

**Microblocks**: Contains txns and is generated by the epoch leader, signed by leader's private key

- Key-block miner → **leader** till next key-block is mined
- Leader publishes micro-blocks while in tenure

Allowing one miner to be a leader, even for a brief interval, presents many concerns!!

ByzCoin
ByzCoin

- Uses key-blocks and micro-blocks
- Key-block miner (PoW) in window becomes a trustee
- Micro-block decided by trustees
- Trustees use PBFT to reach consensus on next micro-block
- Each block is signed using Collective Signing approach

Elastico
Elastico

• Key idea: split all servers into smaller sized groups, committees
Elastico

- Key idea: split all servers into smaller sized groups, **committees**
Elastico

- Key idea: split all servers into smaller sized groups, committees
- Each committee processes a disjoint shard of txns
Elastico

• Key idea: split all servers into smaller sized groups, committees
• Each committee processes a disjoint shard of txns
• Each committee runs any BFT to reach consensus on a block
Elastico

- Key idea: split all servers into smaller sized groups, **committees**
- Each committee processes a **disjoint shard of txns**
- Each committee runs any **BFT** to reach consensus on a block
Elastico

- Key idea: split all servers into smaller sized groups, **committees**
- Each committee processes a **disjoint shard of txns**
- Each committee runs any **BFT** to reach consensus on a block
- A special **Final committee** aggregates all chosen shards and publishes next block in the chain
Elastico

• Key idea: split all servers into smaller sized groups, committees

• Each committee processes a disjoint shard of txns

• Each committee runs any BFT to reach consensus on a block

• A special Final committee aggregates all chosen shards and publishes next block in the chain

Sharding as a Scalability Solution
Sharding as a Scalability Solution
Sharding as a Scalability Solution
Sharding as a Scalability Solution
Sharding as a Scalability Solution
Sharding as a Scalability Solution
Classes of Transactions
Classes of Transactions
Classes of Transactions

Single Shard Transactions
Classes of Transactions
Classes of Transactions
Classes of Transactions

Cross-Shard Transactions
Classes of Transactions

Requires Atomic Cross-Shard Commitment Protocol

Cross-Shard Transactions
The Landscape
The Landscape

Source: coinmarketcap.com on June 7th at 5:00pm PST

Cryptocurrencies: 2225 • Markets: 18851 • Market Cap: $257,486,187,881 • 24h Vol: $66,548,083,112 • BTC Dominance: 55.4%

### Top 100 Cryptocurrencies by Market Capitalization

<table>
<thead>
<tr>
<th>#</th>
<th>Cryptocurrency</th>
<th>Market Cap</th>
<th>Price</th>
<th>Volume (24h)</th>
<th>Circulating Supply</th>
<th>Change (24h)</th>
<th>Price Graph (7d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bitcoin</td>
<td>$142,627,334,795</td>
<td>$8,036.77</td>
<td>$19,138,268,181</td>
<td>17,746,837 BTC</td>
<td>3.15%</td>
<td><img src="url" alt="Graph" /></td>
</tr>
<tr>
<td>2</td>
<td>Ethereum</td>
<td>$26,732,290,299</td>
<td>$251.25</td>
<td>$8,364,736,132</td>
<td>106,397,483 ETH</td>
<td>1.70%</td>
<td><img src="url" alt="Graph" /></td>
</tr>
<tr>
<td>3</td>
<td>XRP</td>
<td>$17,876,222,703</td>
<td>$0.423217</td>
<td>$1,658,461,942</td>
<td>42,238,947,941 XRP*</td>
<td>1.25%</td>
<td><img src="url" alt="Graph" /></td>
</tr>
<tr>
<td>4</td>
<td>Litecoin</td>
<td>$7,281,728,951</td>
<td>$117.21</td>
<td>$5,141,139,892</td>
<td>62,124,551 LTC</td>
<td>6.28%</td>
<td><img src="url" alt="Graph" /></td>
</tr>
<tr>
<td>5</td>
<td>Bitcoin Cash</td>
<td>$7,157,820,741</td>
<td>$401.55</td>
<td>$1,572,103,916</td>
<td>17,826,688 BCH</td>
<td>2.02%</td>
<td><img src="url" alt="Graph" /></td>
</tr>
</tbody>
</table>

*Source: coinmarketcap.com on June 7th at 5:00pm PST*
# The Landscape

Source: coinmarketcap.com on June 7th at 5:00pm PST

## Cryptocurrencies: 2225 • Markets: 18851 • Market Cap: $257,486,187,881 • 24h Vol: $68,548,083,112 • BTC Dominance: 55.4%

## Top 100 Cryptocurrencies by Market Capitalization

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Market Cap</th>
<th>Price</th>
<th>Volume (24h)</th>
<th>Circulating Supply</th>
<th>Change (24h)</th>
<th>Price Graph (7d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bitcoin</td>
<td>$142,627,334,795</td>
<td>$8,036.77</td>
<td>$19,138,268,181</td>
<td>17,746,837 BTC</td>
<td>3.15%</td>
<td><img src="https://example.com" alt="Price Graph" /></td>
</tr>
<tr>
<td>2</td>
<td>Ethereum</td>
<td>$26,732,290,299</td>
<td>$251.25</td>
<td>$8,364,736,132</td>
<td>106,397,483 ETH</td>
<td>1.70%</td>
<td><img src="https://example.com" alt="Price Graph" /></td>
</tr>
<tr>
<td>3</td>
<td>XRP</td>
<td>$17,876,222,703</td>
<td>$0.423217</td>
<td>$1,658,461,942</td>
<td>42,238,947,941 XRP</td>
<td>1.25%</td>
<td><img src="https://example.com" alt="Price Graph" /></td>
</tr>
<tr>
<td>4</td>
<td>Litecoin</td>
<td>$7,281,728,951</td>
<td>$117.21</td>
<td>$5,141,138,982</td>
<td>62,124,551 LTC</td>
<td>6.28%</td>
<td><img src="https://example.com" alt="Price Graph" /></td>
</tr>
<tr>
<td>5</td>
<td>Bitcoin Cash</td>
<td>$7,157,820,741</td>
<td>$401.55</td>
<td>$1,572,103,916</td>
<td>17,825,688 BCH</td>
<td>2.02%</td>
<td><img src="https://example.com" alt="Price Graph" /></td>
</tr>
</tbody>
</table>
The Landscape

**Cryptocurrencies: 2225 • Markets: 18851**

Source: coinmarketcap.com on June 7th at 5:00pm PST

### Top 100 Cryptocurrencies by Market Capitalization

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Market Cap</th>
<th>Price</th>
<th>Volume (24h)</th>
<th>Circulating Supply</th>
<th>Change (24h)</th>
<th>Price Graph (7d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bitcoin</td>
<td>$142,627,334,795</td>
<td>$8,036.77</td>
<td>$19,138,268,181</td>
<td>17,746,837 BTC</td>
<td>3.15%</td>
<td><img src="https://via.placeholder.com/50x50" alt="Graph" /></td>
</tr>
<tr>
<td>2</td>
<td>Ethereum</td>
<td>$26,732,290,299</td>
<td>$251.25</td>
<td>$8,364,736,132</td>
<td>106,397,483 ETH</td>
<td>1.70%</td>
<td><img src="https://via.placeholder.com/50x50" alt="Graph" /></td>
</tr>
<tr>
<td>3</td>
<td>XRP</td>
<td>$17,876,222,703</td>
<td>$0.423217</td>
<td>$1,658,461,942</td>
<td>42,238,947,941 XRP</td>
<td>1.25%</td>
<td><img src="https://via.placeholder.com/50x50" alt="Graph" /></td>
</tr>
<tr>
<td>4</td>
<td>Litecoin</td>
<td>$7,281,728,951</td>
<td>$117.21</td>
<td>$5,141,138,982</td>
<td>62,124,551 LTC</td>
<td>6.28%</td>
<td><img src="https://via.placeholder.com/50x50" alt="Graph" /></td>
</tr>
<tr>
<td>5</td>
<td>Bitcoin Cash</td>
<td>$7,157,820,741</td>
<td>$401.55</td>
<td>$1,572,103,916</td>
<td>17,825,688 BCH</td>
<td>2.02%</td>
<td><img src="https://via.placeholder.com/50x50" alt="Graph" /></td>
</tr>
</tbody>
</table>
The Landscape

Cryptocurrencies: 2225 • Markets: 18851

Market Cap: $257,486,187,861 • 24h Vol: $66,548,083,112

Source: coinmarketcap.com on June 7th at 5:00pm PST
The Landscape
The Landscape

• Thousands of Blockchains
The Landscape

• Thousands of Blockchains
• Tens of thousands of markets
The Landscape

• Thousands of Blockchains
• Tens of thousands of markets
• Exchanges to trade tokens for USD
The Landscape

- Thousands of Blockchains
- Tens of thousands of markets
- Exchanges to trade tokens for USD
- Direct token transactions in one blockchain
The Landscape

• Thousands of Blockchains
• Tens of thousands of markets
• Exchanges to trade tokens for USD
• Direct token transactions in one blockchain
• Direct token transactions across blockchains, how?
The Landscape

• Thousands of Blockchains
• Tens of thousands of markets
• Exchanges to trade tokens for USD
• Direct token transactions in one blockchain
• Direct token transactions across blockchains, how?
• Cross-chain transactions
Cross-ChainTransaction Example
Cross-ChainTransaction Example
Cross-ChainTransaction Example
Cross-ChainTransaction Example
Cross-ChainTransaction Example
Cross-ChainTransaction Example
Cross-ChainTransaction Example
Cross-ChainTransaction Example
Cross-ChainTransaction Example

X bitcoins
Cross-ChainTransaction Example

X bitcoins

Y ethers
Cross-Chain Transaction Example

Atomic Cross-Chain Commitment Protocol

Y ethers

X bitcoins
Cross-Chain Transaction Example

Atomic Cross-Chain Commitment Protocol

Y ethers
X bitcoins
Cross-ChainTransaction Example

Atomic Cross-Chain Commitment Protocol

X bitcoins

Y ethers
Cross-Chain Transaction Example

Atomic Cross-Chain Commitment Protocol

Y ethers

X bitcoins
Smart Contracts
Smart Contracts

• Like classes in Object Oriented Programming Languages
Smart Contracts

• Like classes in Object Oriented Programming Languages
• Allow end-users to:
Smart Contracts

• Like classes in Object Oriented Programming Languages
• Allow end-users to:
  • Store generic data objects in the blockchain
Smart Contracts

• Like classes in Object Oriented Programming Languages
• Allow end-users to:
  • Store generic data objects in the blockchain
  • Define the functions that manipulate these data objects
Smart Contracts

• Like classes in Object Oriented Programming Languages
• Allow end-users to:
  • Store generic data objects in the blockchain
  • Define the functions that manipulate these data objects
• Have attributes (e.g., represents a car)
Smart Contracts

- Like classes in Object Oriented Programming Languages
- Allow end-users to:
  - Store generic data objects in the blockchain
  - Define the functions that manipulate these data objects
- Have attributes (e.g., represents a car)
- Have functions (e.g., rent, buy, etc)
Smart Contracts

• Like classes in Object Oriented Programming Languages
• Allow end-users to:
  • Store generic data objects in the blockchain
  • Define the functions that manipulate these data objects
• Have attributes (e.g., represents a car)
• Have functions (e.g., rent, buy, etc)
• Can be used to implement generic transaction logic:
Smart Contracts

• Like classes in Object Oriented Programming Languages
• Allow end-users to:
  • Store generic data objects in the blockchain
  • Define the functions that manipulate these data objects
• Have attributes (e.g., represents a car)
• Have functions (e.g., rent, buy, etc)
• Can be used to implement generic transaction logic:
  • Conditionally lock assets in the blockchain
Smart Contracts

• Like classes in Object Oriented Programming Languages
• Allow end-users to:
  • Store generic data objects in the blockchain
  • Define the functions that manipulate these data objects
• Have attributes (e.g., represents a car)
• Have functions (e.g., rent, buy, etc)
• Can be used to implement generic transaction logic:
  • Conditionally lock assets in the blockchain
  • Transfer asset ownership on some condition
Smart Contracts
class AtomicSwap {
    sender: s // Alice
    recipient: r // Bob
    asset: a // X bitcoins
    secretHash: h
    constructor() {
    }
    redeem (secret srt) {
        if(hash(srt) == h)
            transfer a to r
    }
    .....
}
Atomic Swap[Nolan’13, Herlihy’18]

• Alice wants to trade Bitcoin for Ethereum with Bob
Atomic Swap[Nolan’13, Herlihy’18]

• Alice wants to trade Bitcoin for Ethereum with Bob
Atomic Swap[Nolan’13, Herlihy’18]

- Alice wants to trade Bitcoin for Ethereum with Bob

- Create a secret $s$
- Calculate its hash $h = H(s)$
Atomic Swap [Nolan’13, Herlihy’18]

- Alice wants to trade Bitcoin for Ethereum with Bob

- Create a secret $s$
- Calculate its hash $h = H(s)$
Atomic Swap[Nolan’13, Herlihy’18]

• Alice wants to trade X Bitcoin for Y Ethereum with Bob

\[ SC_1 \text{ Move } X \text{ bitcoins to Bob if } \text{ Bob provides secret } s \mid h = H(s) \]
Atomic Swap[Nolan’13, Herlihy’18]

• Alice wants to trade X Bitcoin for Y Ethereum with Bob

Bob

Alice

Bitcoin blockchain

SC_1 Move X bitcoins to Bob if Bob provides secret s | h = H(s)

s and h
Atomic Swap [Nolan’13, Herlihy’18]

- Alice wants to trade X Bitcoin for Y Ethereum with Bob

Bob

Alice

SC₁

Move X bitcoins to Bob if Bob provides secret $s \mid h = H(s)$

Bitcoin blockchain

$s$ and $h$
Atomic Swap[Nolan’13, Herlihy’18]

• Now, h is announced in Bitcoin blockchain and made public

**Diagram:***

- Alice's X bitcoins are locked in the smart contract SC₁.
Atomic Swap [Nolan’13, Herlihy’18]

- Now, $h$ is announced in Bitcoin blockchain and made public

Bob

$SC_2$ Move $Y$ Ethereum to Alice if Alice provides secret $s \mid h = H(s)$

Alice

$SC_1$ Alice’s $X$ bitcoins are locked in the smart contract $SC_1$
Atomic Swap [Nolan’13, Herlihy’18]

• Now, \( h \) is announced in Bitcoin blockchain and made public

Ethereum blockchain

\[ SC_2 \text{ Move Y Ethereum to Alice if Alice provides secret } s \mid h = H(s) \]

Bob

| SC_2 |

| Ethereum blockchain |

---

Bitcoin blockchain

| Alice’s X bitcoins are locked in the smart contract \( SC_1 \) |

Alice

| SC_1 |

---

| Bitcoin blockchain |

| \( s \) |

| Bob |

| Alice |
Atomic Swap [Nolan’13, Herlihy’18]

- Now, h is announced in Bitcoin blockchain and made public

![Diagram of atomic swap](image)

- SC₂ Move Y Ethereum to Alice if Alice provides secret $s \mid h = H(s)$
- Alice’s X bitcoins are locked in the smart contract SC₁
Atomic Swap[Nolan’13, Herlihy’18]

• Now, for Alice to execute SC₂ and redeem Y Ethereum, she reveals s
Atomic Swap[Nolan’13, Herlihy’18]

• Now, for Alice to execute $SC_2$ and redeem $Y$ Ethereum, she reveals $s$. 

Bob’s $Y$ Ethereum are locked in smart contract $SC_2$

Alice’s $X$ bitcoins are locked in smart contract $SC_1$
Atomic Swap [Nolan’13, Herlihy’18]

Now, for Alice to execute SC\textsubscript{2} and redeem Y Ethereum, she reveals s
Atomic Swap[Nolan’13, Herlihy’18]

- Revealing $s$, executes $SC_2$. Now $s$ is public in Ethereum’s blockchain.

Bob’s Y Ethereum are locked in smart contract $SC_2$

Alice’s X bitcoins are locked in smart contract $SC_1$
Atomic Swap [Nolan’13, Herlihy’18]

- Now, Bob uses $s$ to execute $SC_1$ and redeem his Bitcoins

Diagram:

- Ethereum blockchain
  - Bob’s Y Ethereum are locked in smart contract $SC_2$

- Bitcoin blockchain
  - Alice’s X bitcoins are locked in smart contract $SC_1$
Atomic Swap [Nolan’13, Herlihy’18]

- Now, Bob uses $s$ to execute $SC_1$ and redeem his Bitcoins.
Atomic Swap Example: What can go wrong?

• Alice locks her X Bitcoins in Bitcoin’s blockchain through SC₁
Atomic Swap Example: What can go wrong?

• Alice locks her X Bitcoins in Bitcoin’s blockchain through SC₁
• Bob sees SC₁ but refuses to publish SC₂
Atomic Swap Example: What can go wrong?

• Alice locks her X Bitcoins in Bitcoin’s blockchain through SC₁
• Bob sees SC₁ but refuses to publish SC₂
• Now, Alice’s Bitcoins are locked for good
  • A conforming party (Alice) ends up worse off because Bob doesn’t follow the protocol
Atomic Swap Example: What can go wrong?

- Alice locks her $X$ Bitcoins in Bitcoin’s blockchain through $SC_1$
- Bob sees $SC_1$ but refuses to publish $SC_2$
- Now, Alice’s Bitcoins are locked for good
  - A conforming party (Alice) ends up worse off because Bob doesn’t follow the protocol
- Prevention
  - Use timelocks to expire a contract
  - Specify that an expired contract is refunded to the creator of this contract
Atomic Swap [Nolan’13, Herlihy’18]: Timelocks
Atomic Swap[Nolan’13, Herlihy’18]: Timelocks

SC₁: Move X bitcoins to Bob if Bob provides secret s | h = H(s)

Refund SC₁ to Alice if Bob does not execute SC₁ before 48 hours
Atomic Swap [Nolan’13, Herlihy’18]: Timelocks

Refund SC₂ to Bob if Alice does not execute SC₂ before 24 hours

SC₂: Move Y Ethereum to Alice if Alice provides secret $s \mid h = H(s)$

Refund SC₁ to Alice if Bob does not execute SC₁ before 48 hours

SC₁: Move X bitcoins to Bob if Bob provides secret $s \mid h = H(s)$
Atomic Swap [Nolan’13, Herlihy’18]: Timelocks

Refund $SC_2$ to Bob if Alice does not execute $SC_2$ before 24 hours.

$SC_2$: Move Y Ethereum to Alice if Alice provides secret $s \mid h = H(s)$.

Refund $SC_1$ to Alice if Bob does not execute $SC_1$ before 48 hours.

$SC_1$: Move X bitcoins to Bob if Bob provides secret $s \mid h = H(s)$.
Atomic Swap Example [Nolan’13, Herlihy’18]

Alice-Bob in Bitcoin

Bob-Alice in Ethereum

Y ethers

X bitcoins

e.g., $\Delta = 12$hr
Atomic Swap Example [Nolan’13, Herlihy’18]

Alice - Bob in Bitcoin

Bob - Alice in Ethereum

\( \Delta \) for both transactions with a time delay of 12 hours.

\( X \) bitcoins

\( Y \) ethers

\( \Delta = 12 \text{hr} \)
Atomic Swap Example [Nolan’13, Herlihy’18]

Alice reveals the secret to Bob’s contract and claims the $Y$ ether

\[ e.g., \Delta = 12\text{hr} \]
Atomic Swap Example [Nolan’13, Herlihy’18]

Alice reveals the secret to Bob’s contract and claims the Y ether.

Supposedly, Bob takes the secret, reveals it to Alice’s contract and claims the X bitcoins.

\[ e.g., \Delta = 12\text{hr} \]
Atomic Swap Example [Nolan’13, Herlihy’18]

- Alice-Bob in Bitcoin
- Bob-Alice in Ethereum

Alice reveals the secret to Bob’s contract and claims the Y ether.

Supposedly, Bob takes the secret, reveals it to Alice’s contract and claims the X bitcoins.

e.g., $\Delta = 12$ hr
What can go wrong?

Alice - Bob in Bitcoin

Bob - Alice in Ethereum

\( \Delta = \text{e.g., 12hr} \)

Y ethers

X bitcoins

e.g., \( \Delta = 12 \text{hr} \)
What can go wrong?

Alice - Bob in Bitcoin

If Bob fails or suffers a network denial of service attack for a \( \Delta \), Alice’s contract will expire and Bob will lose his \( X \) bitcoins.

Bob - Alice in Ethereum

\[ \text{e.g., } \Delta = 12 \text{hr} \]
What can go wrong?

If Bob fails or suffers a network denial of service attack for a Δ, Alice’s contract will expire and Bob will lose his X bitcoins.

X bitcoins are refunded to Alice any time after the contract expires.

e.g., \( \Delta = 12\text{hr} \)
What can go wrong?

Alice - Bob in Bitcoin

Bob - Alice in Ethereum

If Bob fails or suffers a network denial of service attack for a $\Delta$, Alice’s contract will expire and Bob will lose his X bitcoins.

Atomicity Violation

X bitcoins are refunded to Alice any time after the contract expires.

Y ethers

X bitcoins

e.g., $\Delta = 12$ hr
Atomicity Violation

• Using timelocks leads to Atomicity violation
Atomicity Violation

• Using timelocks leads to Atomicity violation

• Our Atomicity-based Approach:
  • The decision of both transactions should be made atomic
    • Once the decision is taken, both transactions either commit or abort
Atomicity Violation

- Using timelocks leads to Atomicity violation
- Our Atomicity-based Approach:
  - The decision of both transactions should be made atomic
    - Once the decision is taken, both transactions either commit or abort
  - A transaction cannot commit unless a commit decision is reached
  - A transaction cannot abort unless an abort decision is reached
Atomic Commitment Across Blockchains

Victor Zakhary, Divyakant Agrawal, Amr El Abbadi
Building block: Cross-Chain Verification

• How can miners of one blockchain:
  • Verify a transaction in another blockchain?
Building block: Cross-Chain Verification

• How can miners of one blockchain:
  • Verify a transaction in another blockchain?
  • Without maintaining a copy of this other blockchain.
Building block: Cross-Chain Verification
Building block: Cross-Chain Verification
Building block: Cross-Chain Verification

Need to verify that $TX_1$ is actually in verified blockchain
Building block: Cross-Chain Verification

Need to verify that $TX_1$ is actually in verified blockchain
Building block: Cross-Chain Verification

Need to verify that TX\(_1\) is actually in verified blockchain
Building block: Cross-Chain Verification

Need to verify that TX$_1$ is actually in verified blockchain
Building block: Cross-Chain Verification

Need to verify that $\text{TX}_1$ is actually in verified blockchain.
Need to verify that $TX_1$ is actually in verified blockchain
Building block: Cross-Chain Verification

Need to verify that $\text{TX}_1$ is actually in verified blockchain
Building block: Cross-Chain Verification

Need to verify that $TX_1$ is actually in verified blockchain.
Building block: Cross-Chain Verification

Need to verify that TX₁ is actually in verified blockchain.
Building block: Cross-Chain Verification

Need to verify that TX₁ is actually in verified blockchain
Building block: Cross-Chain Verification

• Verification process:
Building block: Cross-Chain Verification

• Verification process:
  • Each header includes the hash of the previous header
Building block: Cross-Chain Verification

• Verification process:
  • Each header includes the hash of the previous header
Building block: Cross-Chain Verification

• Verification process:
  • Each header includes the hash of the previous header
Building block: Cross-Chain Verification

• Verification process:
  • Each header includes the hash of the previous header
  • The proof of work of each header is correct
Building block: Cross-Chain Verification

• Verification process:
  • Each header includes the hash of the previous header
  • The proof of work of each header is correct
Building block: Cross-Chain Verification

• Verification process:
  • Each header includes the hash of the previous header
  • The proof of work of each header is correct
Building block: Cross-Chain Verification

• Verification process:
  • Each header includes the hash of the previous header
  • The proof of work of each header is correct
  • TX₁ is correct
Building block: Cross-Chain Verification

• Verification process:
  • Each header includes the hash of the previous header
  • The proof of work of each header is correct
  • $TX_1$ is correct
Building block: Cross-Chain Verification

• Verification process:
  • Each header includes the hash of the previous header
  • The proof of work of each header is correct
  • $TX_1$ is correct
  • $TX_1$ is buried under $d$ blocks
Building block: Cross-Chain Verification

• Verification process:
  • Each header includes the hash of the previous header
  • The proof of work of each header is correct
  • $TX_1$ is correct
  • $TX_1$ is buried under $d$ blocks

• The cost of generating evidence:
  • Choose $d$ to make this cost > the value transacted in $TX_1$
  • If true, a malicious user has no incentive to create a fake evidence
Atomic Commitment Across Blockchains

• Use another blockchain to witness the Atomic Swap
Atomic Commitment Across Blockchains

- Use another blockchain to witness the Atomic Swap
- The witness blockchain decides the commit or the abort of a swap
Atomic Commitment Across Blockchains

- Use another blockchain **to witness** the Atomic Swap
- The *witness blockchain* decides the *commit or the abort* of a swap
- Once a decision is made:
  - All sub-transactions in the swap must follow the decision
  - Achieves atomicity, **either all committed or all aborted**
Atomic Commitment Across Blockchains

• Use another blockchain to witness the Atomic Swap
• The witness blockchain decides the commit or the abort of a swap
• Once a decision is made:
  • All sub-transactions in the swap must follow the decision
  • Achieves atomicity, either all committed or all aborted
• Cross chain verification is leveraged twice
Atomic Commitment Across Blockchains

• Use another blockchain to witness the Atomic Swap
• The witness blockchain decides the commit or the abort of a swap
• Once a decision is made:
  • All sub-transactions in the swap must follow the decision
  • Achieves atomicity, either all committed or all aborted
• Cross chain verification is leveraged twice
  • Miners of the witness network verify the publishing of contracts in asset blockchains
Atomic Commitment Across Blockchains

• Use another blockchain to witness the Atomic Swap
• The witness blockchain decides the commit or the abort of a swap
• Once a decision is made:
  • All sub-transactions in the swap must follow the decision
  • Achieves atomicity, either all committed or all aborted
• Cross chain verification is leveraged twice
  • Miners of the witness network verify the publishing of contracts in asset blockchains
  • Miners of assets’ blockchains verify the decision made in the witness network
Protocol Sketch
Protocol Sketch

• Deploy a contract $SC_w$ in the witness network with state $Published\ (P)$
Protocol Sketch

• Deploy a contract $SC_w$ in the witness network with state *Published* ($P$)
Protocol Sketch

• Deploy a contract $SC_w$ in the witness network with state *Published* ($P$)
Protocol Sketch

• Deploy a contract $SC_w$ in the witness network with state *Published* ($P$)
Protocol Sketch

- Deploy a contract $SC_w$ in the witness network with state *Published* ($P$).
- $SC_w$ has a header of a block at depth $d$ of all blockchains in the swap.
Protocol Sketch Cont’d

Witness Blockchain

Verifier

Bitcoin Blockchain

Verified

Ethereum Blockchain

Verified

$\text{SC}_w \{S=P\}$
Protocol Sketch Cont’d

Witness Blockchain
Verifier

Bitcoin Blockchain
Verified

Ethereum Blockchain
Verified

\[ SC_w \{ S=P \} \]
Protocol Sketch Cont’d

- Participants deploy their contracts in the corresponding blockchains
Protocol Sketch Cont’d

• Participants deploy their contracts in the corresponding blockchains
Protocol Sketch Cont’d

• Participants deploy their contracts in the corresponding blockchains
• Participants add the header of $SC_w$ to their contracts

Witness Blockchain

Verifier

Bitcoin Blockchain

Verified

Ethereum Blockchain

Verified
Protocol Sketch Cont’d

Witness Blockchain
Verifier

Bitcoin Blockchain
Verified

Ethereum Blockchain
Verified

SC \{ S=P \}

SC \{ S=P \}

SC \{ S=P \}
Protocol Sketch Cont’d

Witness Blockchain

Verifier

Bitcoin Blockchain

Verified

Ethereum Blockchain

Verified

SCw {S=P}

SC1 {S=P}

SC2 {S=P}

d blocks
Protocol Sketch Cont’d

Witness Blockchain

Verifier

Bitcoin Blockchain

Verified

Ethereum Blockchain

Verified

d blocks
Protocol Sketch Cont’d

• Participants submit evidence of publishing the smart contracts in Assets Blockchains
• Participants submit evidence of publishing the smart contracts in Assets Blockchains

• If all contracts are published and correct, $SC_w$’s state is altered to redeem (RD)

Witness Blockchain

Verifier

Bitcoin Blockchain

Verified

Ethereum Blockchain

Verified

The Evidence

d blocks
Protocol Sketch Cont’d

Witness Blockchain
- Verified

Bitcoin Blockchain
- Verifier

Ethereum Blockchain
- Verifier
Witness Blockchain

Bitcoin Blockchain

Ethereum Blockchain

Verifier

Verified

\[ SC_w \{ S=P \} \]

\[ SC_1 \{ S=P \} \]

\[ SC_2 \{ S=P \} \]

\[ SC_w \{ S=RD \} \]
Protocol Sketch Cont’d

Witness Blockchain
Verified

Bitcoin Blockchain
Verifier

Ethereum Blockchain
Verifier

SC\textsubscript{w} \{ S=P \}

SC\textsubscript{w} \{ S=RD \}

d blocks

SC\textsubscript{1} \{ S=P \}

SC\textsubscript{2} \{ S=P \}
• Participants submit evidence of Redeem State (RD) from the Witness Blockchain to the Assets Blockchains.
Protocol Sketch Cont’d

• Participants submit evidence of Redeem State (RD) from the Witness Blockchain to the Assets Blockchains.
• Participants submit evidence of Redeem State (RD) from the Witness Blockchain to the Assets Blockchains.

• After evidence verification, participants redeem their assets from the Assets Blockchains.
Atomic Commitment Across Blockchains
Atomic Commitment Across Blockchains

• $SC_w$’s state determines the commit (RD) or the abort (RF) decision
Atomic Commitment Across Blockchains

• $SC_w$’s state determines the commit (RD) or the abort (RF) decision
• Once $SC_w$’s state is altered and the block is buried under $d$ blocks:
Atomic Commitment Across Blockchains

• $SC_w$’s state determines the commit (RD) or the abort (RF) decision
• Once $SC_w$’s state is altered and the block is buried under $d$ blocks:
  • All sub-transactions must follow this decision
Atomic Commitment Across Blockchains

- $S_{C_w}$’s state determines the commit (RD) or the abort (RF) decision.
- Once $S_{C_w}$’s state is altered and the block is buried under d blocks:
  - All sub-transactions must follow this decision.
  - None of the sub-transactions can decide on a different decision.
Atomic Commitment Across Blockchains

• $SC_w$’s state determines the commit (RD) or the abort (RF) decision

• Once $SC_w$’s state is altered and the block is buried under $d$ blocks:
  • All sub-transactions must follow this decision
  • None of the sub-transactions can decide on a different decision

• Even if a participant fails or faces a network denial of service:
Atomic Commitment Across Blockchains

• $SC_w$’s state determines the commit (RD) or the abort (RF) decision

• Once $SC_w$’s state is altered and the block is buried under $d$ blocks:
  • All sub-transactions must follow this decision
  • None of the sub-transactions can decide on a different decision

• Even if a participant fails or faces a network denial of service:
  • When the participant recovers, the evidence of the decision still exists
Atomic Commitment Across Blockchains

• $SC_w$’s state determines the commit (RD) or the abort (RF) decision
• Once $SC_w$’s state is altered and the block is buried under $d$ blocks:
  • All sub-transactions must follow this decision
  • None of the sub-transactions can decide on a different decision
• Even if a participant fails or faces a network denial of service:
  • When the participant recovers, the evidence of the decision still exists
  • This evidence can be used to redeem or refund the contracts
Atomic Commitment Across Blockchains

• SC_w’s state determines the commit (RD) or the abort (RF) decision

• Once SC_w’s state is altered and the block is buried under d blocks:
  • All sub-transactions must follow this decision
  • None of the sub-transactions can decide on a different decision

• Even if a participant fails or faces a network denial of service:
  • When the participant recovers, the evidence of the decision still exists
  • This evidence can be used to redeem or refund the contracts

• The only way to violate atomicity is to fork the witness blockchain
Atomic Commitment Across Blockchains

• $SC_w$’s state determines the commit (RD) or the abort (RF) decision
• Once $SC_w$’s state is altered and the block is buried under $d$ blocks:
  • All sub-transactions must follow this decision
  • None of the sub-transactions can decide on a different decision
• Even if a participant fails or faces a network denial of service:
  • When the participant recovers, the evidence of the decision still exists
  • This evidence can be used to redeem or refund the contracts
• The only way to violate atomicity is to fork the witness blockchain
• Economic incentives prevent this attack
Atomic Commitment Across Blockchains

• $\text{SC}_w$’s state determines the commit (RD) or the abort (RF) decision

• Once $\text{SC}_w$’s state is altered and the block is buried under $d$ blocks:
  • All sub-transactions must follow this decision
  • None of the sub-transactions can decide on a different decision

• Even if a participant fails or faces a network denial of service:
  • When the participant recovers, the evidence of the decision still exists
  • This evidence can be used to redeem or refund the contracts

• The only way to violate atomicity is to fork the witness blockchain

• Economic incentives prevent this attack

• Any protocol is prone to fork attacks
Permissioned Blockchain
Any applications other than Cryptocurrency?
Supply Chain Management: Tracking Fish from Ocean to Table

• Ocean fishing represents more than $70B in worldwide trade\(^1\)
  • Estimates suggest at least 20% of all fish are caught illegally—yet only a tiny fraction are ever inspected\(^2\).
  • Nearly one in three fish were mislabeled by sellers\(^3\)
  • 87% of snapper and 59% of tuna were mislabelled\(^4\)
  • 95% of all sushi restaurants were serving mislabeled fish\(^4\)

\(^1\) Food and Agriculture Organization, United Nations. 2016. The State of World Fisheries and Aquaculture 2016.
\(^4\) Oceana Study Reveals Seafood Fraud Nationwide. 2013.
Supply Chain Management: Tracking Fish from Ocean to Table

• Ocean fishing represents more than $70B in worldwide trade\(^1\)
  • Estimates suggest at least 20% of all fish are caught illegally—yet only a tiny fraction are ever inspected\(^2\).
  • Nearly one in three fish were mislabeled by sellers\(^3\)
    • 87% of snapper and 59% of tuna were mislabelled\(^4\)
    • 95% of all sushi restaurants were serving mislabeled fish\(^4\)

• Challenges:
  • Many different paths from ocean to table
  • Lack of global authority for tracing
  • Proprietary tracing systems do not scale
  • Most existing processes are paper-based
  • The supply chain is extremely complex and includes many participants from different industries

---

\(^1\) Food and Agriculture Organization, United Nations. 2016. The State of World Fisheries and Aquaculture 2016.
\(^4\) Oceana Study Reveals Seafood Fraud Nationwide. 2013.
Seafood Supply Chain in the real world

Seafood Supply Chain in Blockchain

Source: Advancing Traceability in the Seafood Industry, FishWise

Subsistence Fishing -> Recreational Fishing -> Aquaculture -> Wild Capture Fisheries -> Processing and Distribution
Seafood is caught by fishermen and physically tagged with IOT enabled sensors.
Seafood is caught by fishermen and physically tagged with IOT enabled sensors. Sensors continuously transmit data about time and location to Blockchain.

Source: Advancing Traceability in the Seafood Industry, FishWise
Seafood is caught by fishermen and physically tagged with IOT enabled sensors.

Sensors continuously transmit data about time and location to Blockchain.

Source: Advancing Traceability in the Seafood Industry, FishWise
Seafood is caught by fishermen and physically tagged with IOT enabled sensors. Sensors continuously transmit data about time and location to Blockchain. Blockchain facilitates and tracks possession changes through the distribution.
Seafood is caught by fishermen and physically tagged with IOT enabled sensors.

Sensors continuously transmit data about time and location to Blockchain.

Blockchain facilitates and tracks possession changes through the distribution.

The buyer can access a comprehensive record of the fish’s provenance.

Source: Advancing Traceability in the Seafood Industry, FishWise
Seafood is caught by fishermen and physically tagged with IOT enabled sensors. Sensors continuously transmit data about time and location to Blockchain. Blockchain facilitates and tracks possession changes through the distribution. The buyer can access a comprehensive record of the fish’s provenance.

Source: Advancing Traceability in the Seafood Industry, FishWise

Subsistence Fishing
Recreational Fishing
Aquaculture
Wild Capture Fisheries
Processing and Distribution
Blockchain for Supply Chains

- Eliminate information silos and ensure *provenance* with immutable records
- Access end-to-end supply chain data instantly and easily with full *transparency*
- Minimize waste and allocate inventory using insights from real-time demand forecasts
Blockchain for Supply Chains

• Eliminate information silos and ensure *provenance* with immutable records
• Access end-to-end supply chain data instantly and easily with full *transparency*
• Minimize waste and allocate inventory using insights from real-time demand forecasts

Physical Flow

Digital Flow

Blockchain Network
Blockchain for Supply Chains

- Eliminate information silos and ensure *provenance* with immutable records
- Access end-to-end supply chain data instantly and easily with full *transparency*
- Minimize waste and allocate inventory using insights from real-time demand forecasts
Blockchain for Supply Chains

- Eliminate information silos and ensure *provenance* with immutable records
- Access end-to-end supply chain data instantly and easily with full *transparency*
- Minimize waste and allocate inventory using insights from real-time demand forecasts
Blockchain for Supply Chains

- Eliminate information silos and ensure *provenance* with immutable records
- Access end-to-end supply chain data instantly and easily with full *transparency*
- Minimize waste and allocate inventory using insights from real-time demand forecasts
Blockchain for Supply Chains

- Eliminate information silos and ensure *provenance* with immutable records
- Access end-to-end supply chain data instantly and easily with full *transparency*
- Minimize waste and allocate inventory using insights from real-time demand forecasts
Blockchain for Supply Chains

- Eliminate information silos and ensure *provenance* with immutable records
- Access end-to-end supply chain data instantly and easily with full *transparency*
- Minimize waste and allocate inventory using insights from real-time demand forecasts
The difference between Bitcoin and Supply Chain?!

In Supply Chain Participants are known and Identified.
The difference between Bitcoin and Supply Chain?!

In Supply Chain Participants are known and Identified

Traditional **Consensus Protocols** can be used
A Permissioned Blockchain system consists of a set of known, identified nodes that might not fully trust each other.
Permissioned Blockchain

• Run a blockchain among a set of known, identified participants

• Provides a way to secure the interactions among a group of entities that have a common goal but which do not fully trust each other

• The ledger is distributed among all the nodes
Permissioned Blockchain

• Run a blockchain among a set of known, identified participants
• Provides a way to secure the interactions among a group of entities that have a common goal but which do not fully trust each other
• The ledger is distributed among all the nodes

<table>
<thead>
<tr>
<th></th>
<th>Permissionless</th>
<th>Permissioned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participants</td>
<td>Anonymous, Could be malicious</td>
<td>Known, Identified</td>
</tr>
<tr>
<td>Consensus Mechanisms</td>
<td>Proof of Work, Proof of Stake, ...</td>
<td>Byzantine fault tolerance</td>
</tr>
<tr>
<td></td>
<td>• Large energy consumption</td>
<td>Consensus, e.g., PBFT</td>
</tr>
<tr>
<td></td>
<td>• No finality</td>
<td>• Lighter</td>
</tr>
<tr>
<td></td>
<td>• 51% attack</td>
<td>• Faster</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Low energy consumption</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Enable finality</td>
</tr>
<tr>
<td>Transaction Approval time</td>
<td>Long (Bitcoin: 10 min or more)</td>
<td>Short (100x msec)</td>
</tr>
</tbody>
</table>
Consensus Protocols in Permissioned Networks

• Types of systems: synchronous and asynchronous

• Problem statement: given $N$ processes (one of them is the leader):
  • Agreement: all correct processes agree on the same value
  • Validity: If initiator does not fail, all correct processes agree on its value

• Types of failure:
  • Crash
  • Malicious (or Byzantine)

• Important impossibility result:
  • FLP, in asynchronous systems:
    • With even one crash failure, termination is not guaranteed (no liveness)
  • Synchronous systems:
    • Termination is guaranteed if number of failed malicious processes ($f$) is at most $1/3 n$
Bitcoin review
Bitcoin review
Bitcoin review

- Clients **multicasts** their requests
Bitcoin review

• Clients **multicasts** their requests
• Nodes **validate** the transactions, put them into the blocks, and try to **solve the puzzle**
Bitcoin review

- Clients **multicasts** their requests
- Nodes **validate** the transactions, put them into the blocks, and try to **solve the puzzle**
- The lucky node who solves the puzzle first **multicasts** the block
Bitcoin review

- Clients **multicasts** their requests
- Nodes **validate** the transactions, put them into the blocks, and try to **solve the puzzle**
- The lucky node who solves the puzzle first **multicasts** the block
- Each node **validates** the transactions within the block
Bitcoin review

- Clients **multicasts** their requests
- Nodes **validate** the transactions, put them into the blocks, and try to **solve the puzzle**
- The lucky node who solves the puzzle first **multicasts** the block
- Each node **validates** the transactions within the block
- Transactions are **deterministically executed** by every node and **appended** to the ledger
Order-execute Architecture
Order-execute Architecture
Order-execute Architecture

• A set of nodes (might be all of them) *orders* transactions, puts them into blocks, multicasts them to all the nodes.
Order-execute Architecture

• A set of nodes (might be all of them) *orders* transactions, puts them into blocks, multicasts them to all the nodes.

• Each node then *executes* the transactions and *updates* the ledger.
Order-execute Architecture

• A set of nodes (might be all of them) orders transactions, puts them into blocks, multicasts them to all the nodes.

• Each node then executes the transactions and updates the ledger.

• Limitations of Order-Execute
Order-execute Architecture

• A set of nodes (might be all of them) orders transactions, puts them into blocks, multicasts them to all the nodes.

• Each node then executes the transactions and updates the ledger.

• Limitations of Order-Execute
  • Sequential execution: Transactions are sequentially executed on all peers (performance bottleneck)
Order-execute Architecture

• A set of nodes (might be all of them) orders transactions, puts them into blocks, multicasts them to all the nodes.

• Each node then executes the transactions and updates the ledger.

• Limitations of Order-Execute
  • **Sequential execution**: Transactions are sequentially executed on all peers (*performance bottleneck*)
  • **Non-deterministic code**: any non-deterministic execution results in “fork” in the distributed ledger
Order-execute Architecture

- A set of nodes (might be all of them) orders transactions, puts them into blocks, multicasts them to all the nodes.
- Each node then executes the transactions and updates the ledger.

Limitations of Order-Execute
- **Sequential execution:** Transactions are sequentially executed on all peers (*performance bottleneck*)
- **Non-deterministic code:** any non-deterministic execution results in “fork” in the distributed ledger
- **Confidentiality of execution:** all smart contracts run on all peers!
Execute-Order-Validate Architecture
Execute-Order-Validate Architecture

• Each transaction (of an application) is first executed by a subset of nodes (endorsers of the application)
Execute-Order-Validate Architecture

- Each transaction (of an application) is first **executed** by a subset of nodes (endorsers of the application)
- A separate set of nodes (orderers) **orders** the transactions, puts them into blocks, and multicasts them to all the nodes.
Execute-Order-Validate Architecture

• Each transaction (of an application) is first executed by a subset of nodes (endorsers of the application)

• A separate set of nodes (orderers) orders the transactions, puts them into blocks, and multicasts them to all the nodes.

• Each node validates the transactions within a block and updates the ledger
Hyperledger Fabric

**Execute-Order-Validate Architecture:** Transactions are first *executed*, then *ordered*, and finally, *validated*

Hyperledger Fabric

**Execute-Order-Validate Architecture:** Transactions are first *executed*, then *ordered*, and finally, *validated*.

**Non-deterministic Execution:** smart contracts can be written in general-purpose languages instead of domain specific languages.

Hyperledger Fabric

**Execute-Order-Validate Architecture:** Transactions are first *executed*, then *ordered*, and finally, *validated*.

**Non-deterministic Execution:** Smart contracts can be written in general-purpose languages instead of domain specific languages.

**Confidential transactions:** Exposes only the data you want to share to the parties you want to share it with.

Hyperledger Fabric

**Execute-Order-Validate Architecture:** Transactions are first *executed*, then *ordered*, and finally, *validated*.

**Non-deterministic Execution:** smart contracts can be written in general-purpose languages instead of domain specific languages.

**Confidential transactions:** Exposes only the data you want to share to the parties you want to share it with.

**Pluggable architecture:** Tailors the blockchain to industry needs with a pluggable architecture rather than a one size fits all approach.

Hyperledger Fabric

**Execute-Order-Validate Architecture:** Transactions are first *executed*, then *ordered*, and finally, *validated*.

**Non-deterministic Execution:** Smart contracts can be written in general-purpose languages instead of domain specific languages.

**Confidential transactions:** Exposes only the data you want to share to the parties you want to share it with.

**Pluggable architecture:** Tailors the blockchain to industry needs with a pluggable architecture rather than a one size fits all approach.

**Parallel Execution:** Transactions of different applications can be executed in parallel.

Hyperledger Fabric

• Three types of Nodes: Clients, Endorsers, and Orderers
Hyperledger Fabric

• Three types of Nodes: Clients, Endorsers, and Orderers
Hyperledger Fabric

• Three types of Nodes: Clients, Endorsers, and Orderers
  • Clients send transactions to be executed.
Hyperledger Fabric

• Three types of Nodes: Clients, Endorsers, and Orderers
  • Clients send transactions to be executed.
  • Endorsers execute transaction proposals and validate transactions.
    • All endorsers maintain the blockchain ledger
    • Each application has its own set of endorsers
Hyperledger Fabric

- Three types of Nodes: Clients, Endorsers, and Orderers
  - **Clients** send transactions to be executed.
  - **Endorsers** execute transaction proposals and validate transactions.
    - All endorsers maintain the blockchain ledger
    - Each application has its own set of endorsers
  - **Orderers** establish the total order of all transactions using a consensus protocol
    - Do not maintain the blockchain ledger or smart contracts
    - The consensus protocol is pluggable
Hyperledger Fabric
Hyperledger Fabric

Three Applications (Green, Blue, Yellow)
Three Clients (Alice, Bob, Charlie)
Green and Blue have two Endorsers, Yellow has four Endorsers
There are totally six Orderers
Hyperledger Fabric

Three Applications (Green, Blue, Yellow)
Three Clients (Alice, Bob, Charlie)
Green and Blue have two Endorsers, Yellow has four Endorsers
There are totally six Orderers
Three Applications (Green, Blue, Yellow)
Three Clients (Alice, Bob, Charlie)
Green and Blue have two Endorsers, Yellow has four Endorsers
There are totally six Orderers

Transactions of different applications are executed in parallel

Transactions of different applications are executed in parallel
Hyperledger Fabric

Three Applications (Green, Blue, Yellow)
Three Clients (Alice, Bob, Charlie)
Green and Blue have two Endorsers, Yellow has four Endorsers
There are totally six Orderers
Hyperledger Fabric

Three Applications (Green, Blue, Yellow)
Three Clients (Alice, Bob, Charlie)
Green and Blue have two Endorsers, Yellow has four Endorsers
There are totally six Orderers

If the results are identical, the client put them into a request
Three Applications (Green, Blue, Yellow)
Three Clients (Alice, Bob, Charlie)
Green and Blue have two Endorsers, Yellow has four Endorsers
There are totally six Orderers
Three Applications (Green, Blue, Yellow)
Three Clients (Alice, Bob, Charlie)
Green and Blue have two Endorsers, Yellow has four Endorsers
There are totally six Orderers
Hyperledger Fabric

Three Applications (Green, Blue, Yellow)
Three Clients (Alice, Bob, Charlie)
Green and Blue have two Endorsers, Yellow has four Endorsers
There are totally six Orderers

A block might contain multiple transactions from the same application
Three Applications (Green, Blue, Yellow)
Three Clients (Alice, Bob, Charlie)
Green and Blue have two Endorsers, Yellow has four Endorsers
There are totally six Orderers
Hyperledger Fabric

Three Applications (Green, Blue, Yellow)
Three Clients (Alice, Bob, Charlie)
Green and Blue have two Endorsers, Yellow has four Endorsers
There are totally six Orderers

In the validation phase, Endorsers check: (1) validity of transactions, (2) read-write conflicts
Three Applications (Green, Blue, Yellow)
Three Clients (Alice, Bob, Charlie)
Green and Blue have two Endorsers, Yellow has four Endorsers
There are totally six Orderers
Hyperledger Fabric
What if transactions are **conflicting**?

transactions access the same record and one of them is a write operation

What if transactions are conflicting?
What if transactions are conflicting?

transactions access the same record and one of them is a write operation

What if transactions are conflicting?

transactions access the same record and one of them is a write operation
Hyperledger Fabric

What if transactions are conflicting?
transactions access the same record and one of them is a write operation

What if transactions are conflicting?

transactions access the same record and one of them is a write operation
What if transactions are conflicting?

transactions access the same record and one of them is a write operation.
Hyperledger Fabric

What if transactions are conflicting?
transactions access the same record and one of them is a write operation

What if transactions are conflicting?
transactions access the same record and one of them is a write operation

What if transactions are conflicting?
transactions access the same record and one of them is a write operation

What if transactions are conflicting?
transactions access the same record and one of them is a write operation

What if transactions are conflicting?
transactions access the same record and one of them is a write operation

What if transactions are conflicting?
transactions access the same record and one of them is a write operation

What if transactions are conflicting?
transactions access the same record and one of them is a write operation

What if transactions are conflicting?
transactions access the same record and one of them is a write operation
Dependency Graph

• A dependency graph exposes conflicts between transactions to give a partial order of transactions.
Dependency Graph

- A dependency graph exposes conflicts between transactions to give a partial order of transactions.

```
T_1  Read = \{a\}  Write = \{a,b\}
T_2  Read = \{f\}  Write = \{d\}
T_3  Read = \{f\}  Write = \{e\}
T_4  Read = \{b\}  Write = \{c\}
T_5  Read = \{e\}  Write = \{d\}
```
Dependency Graph

- A dependency graph exposes conflicts between transactions to give a partial order of transactions.

```
T_1  Read = \{a\}
     Write = \{a, b\}

T_2  Read = \{f\}
     Write = \{d\}

T_3  Read = \{f\}
     Write = \{e\}

T_4  Read = \{b\}
     Write = \{c\}

T_5  Read = \{e\}
     Write = \{d\}
```
A dependency graph exposes conflicts between transactions to give a partial order of transactions.

- $T_1$: Read = \{a\}, Write = \{a,b\}
- $T_2$: Read = \{f\}, Write = \{d\}
- $T_3$: Read = \{f\}, Write = \{e\}
- $T_4$: Read = \{b\}, Write = \{c\}
- $T_5$: Read = \{e\}, Write = \{d\}
A dependency graph exposes conflicts between transactions to give a partial order of transactions.

- \( T_1 \): Read = \{a\}, Write = \{a, b\}
- \( T_2 \): Read = \{f\}, Write = \{d\}
- \( T_3 \): Read = \{f\}, Write = \{e\}
- \( T_4 \): Read = \{b\}, Write = \{c\}
- \( T_5 \): Read = \{e\}, Write = \{d\}

\( T_4 \) reads b that is written by \( T_1 \).
A dependency graph exposes conflicts between transactions to give a partial order of transactions.

\[
\begin{align*}
T_1 & \quad \text{Read} = \{a\}, \quad \text{Write} = \{a, b\} \\
T_2 & \quad \text{Read} = \{f\}, \quad \text{Write} = \{d\} \\
T_3 & \quad \text{Read} = \{f\}, \quad \text{Write} = \{e\} \\
T_4 & \quad \text{Read} = \{b\}, \quad \text{Write} = \{c\} \\
T_5 & \quad \text{Read} = \{e\}, \quad \text{Write} = \{d\}
\end{align*}
\]

$T_4$ reads $b$ that is written by $T_1$

$T_3$ writes $e$ that is read by $T_5$
Dependency Graph

• A dependency graph exposes conflicts between transactions to give a partial order of transactions.

T_1 \text{ Read } = \{a\} \text{ Write } = \{a, b\}

T_2 \text{ Read } = \{f\} \text{ Write } = \{d\}

T_3 \text{ Read } = \{f\} \text{ Write } = \{e\}

T_4 \text{ Read } = \{b\} \text{ Write } = \{c\}

T_5 \text{ Read } = \{e\} \text{ Write } = \{d\}

T_4 \text{ reads } b \text{ that is written by } T_1

T_3 \text{ writes } e \text{ that is read by } T_5

T_2 \text{ writes } d \text{ that is written by } T_5
Order-Parallel Execute (OXII) Architecture
Order-Parallel Execute (OXII) Architecture

• A separate set of nodes (orderers) orders the transactions, puts them into blocks, generates a dependency graph for the block, and multicasts it to all the nodes.
Order-Parallel Execute (OXII) Architecture

• A separate set of nodes (orderers) orders the transactions, puts them into blocks, generates a dependency graph for the block, and multicasts it to all the nodes.

• Each transaction (of an application) is then validated and executed by a subset of nodes (executors of the application) following the dependency graph.
Order-Parallel Execute (OXII) Architecture

• A separate set of nodes (orderers) **orders** the transactions, puts them into blocks, generates a **dependency graph** for the block, and multicasts it to all the nodes.

• Each transaction (of an application) is then **validated** and **executed** by a subset of nodes (executors of the application) following the dependency graph.

• The nodes multicast the results of execution and append the block.
ParBlockchain

ParBlockchain

**Order-Execute Architecture**: Transactions are first ordered, and then executed

ParBlockchain

Order-Execute Architecture: Transactions are first ordered, and then executed

Parallel Execution: non-conflicting transactions of the same or different applications are executed in parallel

ParBlockchain

**Order-Execute Architecture:** Transactions are first ordered, and then executed

**Parallel Execution:** non-conflicting transactions of the same or different applications are executed in parallel

**Conflict detection:** any conflict (contention) between transaction is detected in the ordering phase and considered in the execution phase

ParBlockchain

**Order-Execute Architecture:** Transactions are first ordered, and then executed.

**Parallel Execution:** non-conflicting transactions of the same or different applications are executed in parallel.

**Conflict detection:** any conflict (contention) between transaction is detected in the ordering phase and considered in the execution phase.

**Pluggable architecture, Confidential transaction, non-deterministic execution** similar to Hyperledger Fabric, Parblockchain has these three properties.

ParBlockchain

**Order-Execute Architecture:** Transactions are first ordered, and then executed

**Parallel Execution:** non-conflicting transactions of the same or different applications are executed in parallel

**Conflict detection:** any conflict (contention) between transaction is detected in the ordering phase and considered in the execution phase

**Pluggable architecture, Confidential transaction, non-deterministic execution**

similar to Hyperledger Fabric, Parblockchain has these three properties

**Non-deterministic Execution:** inconsistent execution results can be detected in the last phase (results in decreasing the performance)

ParBlockchain

Clients

Orderers

Executors

Each application has a set of Executors
Each Executor stores a copy of ledger and Data
Each transaction of an application include records to be read and written.
The orderers order transactions using a consensus protocol (e.g. PBFT)
ParBlockchain

Clients

- T1: Read = {a}, Write = {a, b}
- T2: Read = {f}, Write = {d}
- T3: Read = {f}, Write = {e}
- T4: Read = {b}, Write = {c}
- T5: Read = {e}, Write = {d}

Each orderer generates a dependency graph for the block and multicasts it to all Executors.

Orderers

- O1
- O2
- O3
- O4

Executors

- Application A1
- Application A2
- Application A3

KVS

Ledger

Pre-prepare | Prepare | Commit

- T5
- T4
- T2
- T1
- T3
- T4
Clients

- $T_1$: Read = \{a\}, Write = \{a, b\}
- $T_2$: Read = \{f\}, Write = \{d\}
- $T_3$: Read = \{f\}, Write = \{e\}
- $T_4$: Read = \{b\}, Write = \{c\}
- $T_5$: Read = \{e\}, Write = \{d\}

Orderes

- $O_1$: Read = \{\}, Write = \{\}
- $O_2$: Read = \{\}, Write = \{\}
- $O_3$: Read = \{\}, Write = \{\}
- $O_4$: Read = \{\}, Write = \{\}

Executors

- $A_1$: Ledger
- $A_2$: Ledger
- $A_3$: Ledger

Application $A_1$, $A_2$, $A_3$

Executors of each application execute the corresponding transactions following the dependency graph and multicast the results.
Optimistic vs. Pessimistic Execution

Two ways to look at the problem!

- Supporting non-deterministic execution
- Supporting High Contention Workloads
Optimistic vs. Pessimistic Execution

Two ways to look at the problem!

Supporting non-deterministic execution
- Executes first (does not submit transactions with inconsistent results)

Supporting High Contention Workloads
- Validates read-write conflicts last (aborts conflicting transactions)

Hyperledger
**Optimistic vs. Pessimistic Execution**

Two ways to look at the problem!

**Supporting non-deterministic execution**

- **Hyperledger**
  - Executes first (does not submit transactions with inconsistent results)

- **ParBlcockchain**
  - Validates non-determinist execution last (aborts transactions with inconsistent results)

**Supporting High Contention Workloads**

- Validates read-write conflicts last (aborts conflicting transactions)

- Checks conflicts first (generates a dependency graph)
Blockchain Scalability

• *Scalability* is one of the main roadblocks to business adoption of blockchains
Blockchain Scalability

• *Scalability* is one of the main roadblocks to business adoption of blockchains
• Two classes of solutions for Scalability:
  1) **Off-chain (layer two)**: built on top of the main chain, move a portion of the transactions off the chain, e.g. lightning networks
Blockchain Scalability

• *Scalability* is one of the main roadblocks to business adoption of blockchains
• Two classes of solutions for Scalability:
  1) **Off-chain (layer two):** built on top of the main chain, move a portion of the transactions off the chain, e.g. lightning networks
  2) **On-chain (layer one):** increase the throughput of the main chain
     • **Vertical techniques:** more power is added to each node to perform more tasks
     • **Horizontal techniques:** increase the number of nodes in the network
Blockchain Scalability

- *Scalability* is one of the main roadblocks to business adoption of blockchains
- Two classes of solutions for Scalability:
  1. **Off-chain (layer two):** built on top of the main chain, move a portion of the transactions off the chain, e.g. lightning networks
  2. **On-chain (layer one):** increase the throughput of the main chain
     - **Vertical techniques:** more power is added to each node to perform more tasks
     - **Horizontal techniques:** increase the number of nodes in the network

**Sharding** (as a horizontal technique): Partitioning the data into multiple shards that are maintained by different subsets of nodes
Sharding Blockchains
Sharding Blockchains

- Partition the nodes into clusters of $3f+1$ nodes (to guarantee safety in each cluster in the presence of $f$ malicious nodes)
Sharding Blockchains

• Partition the nodes into clusters of $3f+1$ nodes (to guarantee safety in each cluster in the presence of $f$ malicious nodes)

• How to form clusters such that each cluster includes at most $f$ faulty nodes?
  • Assign nodes to clusters in a random manner (uniform distribution): works if $f$ is very large
  • Assume that $N$ is much larger than $3f+1$ (reasonable assumption in blockchain environment)
Sharding Blockchains

• Partition the nodes into clusters of $3f+1$ nodes (to guarantee safety in each cluster in the presence of $f$ malicious nodes)

• How to form clusters such that each cluster includes at most $f$ faulty nodes?
  • Assign nodes to clusters in a random manner (uniform distribution): works if $f$ is very large
  • Assume that $N$ is much larger than $3f+1$ (reasonable assumption in blockchain environment)

• Shard the data
  • Shard the application data and assign shards to clusters
  • Each data shard is replicated across nodes of a cluster
  • Different clusters process the transactions of their corresponding shard in parallel
Sharding Blockchains

• Partition the nodes into clusters of $3f+1$ nodes (to guarantee safety in each cluster in the presence of $f$ malicious nodes)

• How to form clusters such that each cluster includes at most $f$ faulty nodes?
  • Assign nodes to clusters in a random manner (uniform distribution): works if $f$ is very large
  • Assume that $N$ is much larger than $3f+1$ (reasonable assumption in blockchain environment)

• Shard the data
  • Shard the application data and assign shards to clusters
  • Each data shard is replicated across nodes of a cluster
  • Different clusters process the transactions of their corresponding shard in parallel
  • The Blockchain ledger is also sharded
Sharding Blockchains

- Partition the nodes into clusters of $3f+1$ nodes (to guarantee safety in each cluster in the presence of $f$ malicious nodes)

- How to form clusters such that each cluster includes at most $f$ faulty nodes?
  - Assign nodes to clusters in a random manner (uniform distribution): works if $f$ is very large
  - Assume that $N$ is much larger than $3f+1$ (reasonable assumption in blockchain environment)

- Shard the data
  - Shard the application data and assign shards to clusters
  - Each data shard is replicated across nodes of a cluster
  - Different clusters process the transactions of their corresponding shard in parallel
  - The Blockchain ledger is also sharded

- Cross-Shard transactions
  - Need the participant of all (and only) involved clusters
SharPer: Sharding Permissioned Blockchains

Amiri, Mohammad Javad, Divyakant Agrawal, and Amr El Abbadi. Sharding Permissioned Blockchains, IEEE International Conference on Blockchain, 2019

Amiri, Mohammad Javad, Divyakant Agrawal, and Amr El Abbadi. SharPer: Sharding Permissioned Blockchains Over Network Clusters. (In submission)
SharPer: Sharding Permissioned Blockchains

• The blockchain ledger is generalized from a linear chain to a directed acyclic graph (DAG)

Amiri, Mohammad Javad, Divyakant Agrawal, and Amr El Abbadi. Sharding Permissioned Blockchains, IEEE International Conference on Blockchain, 2019

Amiri, Mohammad Javad, Divyakant Agrawal, and Amr El Abbadi. SharPer: Sharding Permissioned Blockchains Over Network Clusters. (In submission)
SharPer: Sharding Permissioned Blockchains

- The blockchain ledger is generalized from a linear chain to a directed acyclic graph (DAG)
- Each block includes a single transaction
SharPer: Sharding Permissioned Blockchains

- The blockchain ledger is generalized from a linear chain to a directed acyclic graph (DAG)
- Each block includes a single transaction
- The total order is captured by chaining the transactions (blocks) together
  - Each transaction includes the cryptographic hash of the previous transaction
SharPer: Sharding Permissioned Blockchains

- The blockchain ledger is generalized from a linear chain to a directed acyclic graph (DAG)
- Each block includes a single transaction
- The total order is captured by chaining the transactions (blocks) together
  - Each transaction includes the cryptographic hash of the previous transaction
- Cross-chain transactions include the hash of the previous transactions of all involved shards.
SharPer: Sharding Permissioned Blockchains

- The blockchain ledger is generalized from a linear chain to a directed acyclic graph (DAG)
- Each block includes a single transaction
- The total order is captured by chaining the transactions (blocks) together
  - Each transaction includes the cryptographic hash of the previous transaction
- Cross-chain transactions include the hash of the previous transactions of all involved shards.
- The entire blockchain ledger is not maintained by any node
- Each node only maintains its own view of the blockchain ledger
  - including the transactions that access the data shard of the cluster

Amiri, Mohammad Javad, Divyakant Agrawal, and Amr El Abbadi. Sharding Permissioned Blockchains, IEEE International Conference on Blockchain, 2019

Amiri, Mohammad Javad, Divyakant Agrawal, and Amr El Abbadi. SharPer: Sharding Permissioned Blockchains Over Network Clusters. (In submission)
SharPer Ledger

The Blockchain Ledger and the view of clusters $P_1$, $P_2$, $P_3$, and $P_4$
SharPer Ledger

The Blockchain Ledger and the view of clusters $P_1$, $P_2$, $P_3$, and $P_4$

- Intra-shard transactions of different clusters are processed in parallel
SharPer Ledger

The Blockchain Ledger and the view of clusters $P_1$, $P_2$, $P_3$, and $P_4$

- Intra-shard transactions of different clusters are processed in parallel
SharPer Ledger

The Blockchain Ledger and the view of clusters $P_1$, $P_2$, $P_3$, and $P_4$

- Intra-shard transactions of different clusters are processed in parallel
- Cross-shard transactions with non-overlapping clusters are processed in parallel
- A cross-shard transaction includes multiple hash pointers
SharPer Ledger

The Blockchain Ledger and the view of clusters $P_1$, $P_2$, $P_3$, and $P_4$

- Intra-shard transactions of different clusters are processed in parallel
- Cross-shard transactions with non-overlapping clusters are processed in parallel
- A cross-shard transaction includes multiple hash pointers

![Diagram showing transactions and clusters]
SharPer Ledger

The Blockchain Ledger and the view of clusters $P_1$, $P_2$, $P_3$, and $P_4$

- Intra-shard transactions of different clusters are processed in parallel
- Cross-shard transactions with non-overlapping clusters are processed in parallel
- A cross-shard transaction includes multiple hash pointers
SharPer Ledger

The Blockchain Ledger and the view of clusters $P_1$, $P_2$, $P_3$, and $P_4$

- Intra-shard transactions of different clusters are processed in parallel
- Cross-shard transactions with non-overlapping clusters are processed in parallel
- A cross-shard transaction includes multiple hash pointers
SharPer Ledger

The Blockchain Ledger and the view of clusters $P_1$, $P_2$, $P_3$, and $P_4$

- Intra-shard transactions of different clusters are processed in parallel
- Cross-shard transactions with non-overlapping clusters are processed in parallel
- A cross-shard transaction includes multiple hash pointers
- All clusters might be involved in a cross-shard transaction
SharPer Ledger

The Blockchain Ledger and the view of clusters $P_1$, $P_2$, $P_3$, and $P_4$

- Intra-shard transactions of different clusters are processed in parallel
- Cross-shard transactions with non-overlapping clusters are processed in parallel
- A cross-shard transaction includes multiple hash pointers
- All clusters might be involved in a cross-shard transaction
Consensus in SharPer
Consensus in SharPer

• Intra-Shard Consensus: using any Byzantine fault-tolerant protocols, e.g. PBFT
Consensus in SharPer

- **Intra-Shard Consensus**: using any Byzantine fault-tolerant protocols, e.g. PBFT
- If nodes follow crash failure model, use crash fault-tolerant protocol, e.g., Paxos
Consensus in SharPer

- **Intra-Shard Consensus**: using any Byzantine fault-tolerant protocols, e.g. PBFT
- If nodes follow crash failure model, use crash fault-tolerant protocol, e.g., Paxos
- **Cross-Shard Consensus**: needs the participation of *all the involved clusters*
  - In each step $2f+1$ nodes of *every* involved cluster must participate
Cross-Shard Consensus in SharPer

Non-overlapping cross-shard transactions can be processed in parallel.

\[ c_1 c_2 \quad \quad \quad \quad p_1 \quad p_2 \quad p_3 \quad p_4 \]
Cross-Shard Consensus in SharPer

Non-overlapping cross-shard transactions can be processed in parallel. Clients ($c_1$ and $c_2$) send requests to the (pre-elected) primary nodes.
Cross-Shard Consensus in SharPer

Non-overlapping cross-shard transactions can be processed in parallel. Clients ($c_1$ and $c_2$) send requests to the (pre-elected) primary nodes. Primary nodes multicast propose messages including the hash of their previous transactions to every node of all involved partitions.
Cross-Shard Consensus in SharPer

Non-overlapping cross-shard transactions can be processed in parallel. Clients ($c_1$ and $c_2$) send requests to the (pre-elected) primary nodes. Primary nodes multicast propose messages including the hash of their previous transactions to every node of all involved partitions. Each node multicasts accept message including the hash of its previous transaction to every node of all involved partitions.
Cross-Shard Consensus in SharPer

Non-overlapping cross-shard transactions can be processed in parallel. Clients ($c_1$ and $c_2$) send requests to the (pre-elected) primary nodes. Primary nodes multicast propose messages including the hash of their previous transactions to every node of all involved partitions. Each node multicasts accept message including the hash of its previous transaction to every node of all involved partitions. Upon receiving $2f+1$ matching accept message from each cluster, each node collects hashes of all clusters and multicasts Commit message to every node of all involved partitions.
Non-overlapping cross-shard transactions can be processed in parallel. Clients (c_1 and c_2) send requests to the (pre-elected) primary nodes. Primary nodes multicast propose messages including the hash of their previous transactions to every node of all involved partitions. Each node multicasts accept message including the hash of its previous transaction to every node of all involved partitions. Upon receiving 2f+1 matching accept message from each cluster, each node collects hashes of all clusters and multicasts commit message to every node of all involved partitions. Upon receiving 2f+1 matching commit message from each cluster, each node executes the transaction and appends it to the ledger.
Collaborative Workflow: Supply Chain Management

- Different parties (applications) need to communicate across organizations to provide services.
- The communication follows *Service Level Agreements* (agreed upon by all participants).
- They *do not trust* each other.
- The blockchain system should support *both* cross-application and internal transactions.
- Internal data of each party is *confidential*.
Collaborative Workflow: Supply Chain Management

- Different parties (applications) need to communicate across organizations to provide services.
- The communication follows *Service Level Agreements* (agreed upon by all participants).
- They *do not trust* each other.
- The blockchain system should support *both* cross-application and internal transactions.
- Internal data of each party is *confidential.*
Collaborative Workflows using Blockchain

First Solution: Deploy all applications on the same blockchain system
- Similar to Hyperledger Fabric
- Smart contracts are confidential
- Transactions data and blockchain ledger are replicated on every application
Collaborative Workflows using Blockchain

First Solution: Deploy all applications on the same blockchain system
- Similar to Hyperledger Fabric
- Smart contracts are confidential
- Transactions data and blockchain ledger are replicated on every application

Confidentiality issue
Collaborative Workflows using Blockchain

First Solution: Deploy all applications on the same blockchain system
• Similar to Hyperledger Fabric
• Smart contracts are confidential
• Transactions data and blockchain ledger are replicated on every application

Confidentiality issue

Second Solution: Deploy each application on a separate blockchain system
• Use another blockchain system for the cross-application transactions
Collaborative Workflows using Blockchain

**First Solution:** Deploy all applications on the same blockchain system
- Similar to Hyperledger Fabric
- Smart contracts are confidential
- Transactions data and blockchain ledger are replicated on every application

**Second Solution:** Deploy each application on a separate blockchain system
- Use another blockchain system for the cross-application transactions

Confidentiality issue

Data Integrity issue
Collaborative Workflows using Blockchain

First Solution: Deploy all applications on the same blockchain system
• Similar to Hyperledger Fabric
• Smart contracts are confidential
• Transactions data and blockchain ledger are replicated on every application

Second Solution: Deploy each application on a separate blockchain system
• Use another blockchain system for the cross-application transactions

Third Solution: Deploy each application on a separate blockchain system
• Use cross-chain operation
Collaborative Workflows using Blockchain

First Solution: Deploy all applications on the same blockchain system
• Similar to Hyperledger Fabric
• Smart contracts are confidential
• Transactions data and blockchain ledger are replicated on every application

Confidentiality issue

Second Solution: Deploy each application on a separate blockchain system
• Use another blockchain system for the cross-application transactions

Data Integrity issue

Third Solution: Deploy each application on a separate blockchain system
• Use cross-chain operation

Performance issue
CAPER: A Cross-Application Permissioned Blockchain

• Distributed applications collaborate with each other following SLAs

CAPER: A Cross-Application Permissioned Blockchain

• Distributed applications collaborate with each other following SLAs

CAPER: A Cross-Application Permissioned Blockchain

- Distributed applications collaborate with each other following SLAs

CAPER: A Cross-Application Permissioned Blockchain

• Distributed applications collaborate with each other following SLAs
• Two types of transactions: internal and cross-application
• Cross-application transactions are visible to all applications
• Internal transactions of each application are confidential
CAPER: A Cross-Application Permissioned Blockchain

- Distributed applications collaborate with each other following SLAs.
- Two types of transactions: *internal* and *cross-application*.
- Cross-application transactions are *visible to all* applications.
- Internal transactions of each application are *confidential*.
- The blockchain ledger is formed as a *directed acyclic graph*.

CAPER: A Cross-Application Permissioned Blockchain

• Distributed applications collaborate with each other following SLAs
• Two types of transactions: *internal* and *cross-application*
• Cross-application transactions are *visible to all* applications
• Internal transactions of each application are *confidential*
• The blockchain ledger is formed as a *directed acyclic graph*
• Each application maintains *only* its own view of the ledger
  • including its internal and all cross-application transactions.

The Blockchain Ledger of CAPER
The Blockchain Ledger of CAPER

Each application has its own internal transactions
The Blockchain Ledger of CAPER

Cross-application transactions are maintained by every application
The Blockchain Ledger of CAPER
The Blockchain Ledger of CAPER
The Blockchain Ledger of CAPER
The Blockchain Ledger of CAPER
The Blockchain Ledger of CAPER
Confidentiality of Cross-Application Transactions
Confidentiality of Cross-Application Transactions

• In CAPER:
  • Internal transactions read both private and public data and write on private data
  • Cross-application transactions read/write only public data
Confidentiality of Cross-Application Transactions

• In CAPER:
  • Internal transactions read both private and public data and write on private data
  • Cross-application transactions read/write only public data
• What if a cross-application transaction read/write private data?
• How to validate private transactions without revealing any information?
Confidentiality of Cross-Application Transactions

• In CAPER:
  • Internal transactions read both *private* and *public* data and write on *private* data
  • Cross-application transactions read/write *only public* data

• What if a cross-application transaction read/write *private* data?

• How to validate *private* transactions without revealing any information?
  • Cryptography techniques are needed!
Confidentiality of Cross-Application Transactions

- In CAPER:
  - Internal transactions read both *private* and *public* data and write on *private* data
  - Cross-application transactions read/write only *public* data
- What if a cross-application transaction read/write *private* data?
- How to validate *private* transactions without revealing any information?
- Cryptography techniques are needed!
  - Quorum uses *zero knowledge proof*
  - Fabric defines *Private data collections*
Case Study on Change Healthcare’s use of Hyperledger Fabric

Change Healthcare turned to Hyperledger Fabric to begin blockchain-enabling its Intelligent Healthcare Network, which now processes 50 million transactions a day.

LEARN MORE IN THE BLOG
READ THE CASE STUDY

Join Hyperledger as a Member

Hyperledger Member Summit is coming up July 30-31 in Tokyo, Japan. Now is a great time to consider joining Hyperledger as a member so you can attend this annual event to discuss the current and future state of Hyperledger technologies.

LEARN MORE

Hyperledger Transact Now Available

Announcing our latest project to join the Hyperledger Greenhouse. Hyperledger Transact provides a platform-agnostic library that handles the execution of smart contracts, including all aspects of scheduling, transaction dispatch, and state management.

LEARN MORE IN THE BLOG
START CONTRIBUTING

https://www.hyperledger.org/
The Hyperledger Greenhouse
Business Blockchain Frameworks & Tools Hosted by Hyperledger

Community Stewardship and Technical, Legal, Marketing, Organizational Infrastructure

Frameworks
- Hyperledger Burrow: Permissionable smart contract machine (EVM)
- Hyperledger Fabric: Permissioned with channel support
- Hyperledger Grid: WebAssembly-based project for building supply chain solutions
- Hyperledger Indy: Decentralized identity
- Hyperledger Iroha: Mobile application focus
- Hyperledger Sawtooth: Permissioned & permissionless support, EVM transaction family

Tools
- Hyperledger Aries: Infrastructure for peer-to-peer interactions
- Hyperledger Caliper: Blockchain framework benchmark platform
- Hyperledger Cello: As-a-service deployment
- Hyperledger Composer: Model and build blockchain networks
- Hyperledger Explorer: View and explore data on the blockchain
- Hyperledger Quilt: Ledger interoperability
- Hyperledger Transact: Advanced transaction execution and state management
- Hyperledger Ursa: Shared Cryptographic Library
From Cryptocurrencies to Global Asset Management

Victor Zakhary, Mohammad Amiri, Sujaya Maiyya, Divyakant Agrawal, Amr El Abbadi
From Cryptocurrencies to Global Assets
From Cryptocurrencies to Global Assets

• So far, Mining Node:
From Cryptocurrencies to Global Assets

• So far, Mining Node:
  • Store cryptocurrency units
  • Store ownership
  • Execute Transactions (transfer ownership of currency units)
From Cryptocurrencies to Global Assets

• So far, Mining Node:
  • Store cryptocurrency units
  • Store ownership
  • Execute Transactions (transfer ownership of currency units)

• Mining Nodes → The new public cloud
From Cryptocurrencies to Global Assets

• So far, Mining Node:
  - Store cryptocurrency units
  - Store ownership
  - Execute Transactions (transfer ownership of currency units)

• Mining Nodes → The new public cloud

• Store:
From Cryptocurrencies to Global Assets

• So far, Mining Node:
  • Store cryptocurrency units
  • Store ownership
  • Execute Transactions (transfer ownership of currency units)

• Mining Nodes → The new public cloud

• Store:
  • General Assets (e.g., cars, houses, etc)
From Cryptocurrencies to Global Assets

• So far, Mining Node:
  • Store cryptocurrency units
  • Store ownership
  • Execute Transactions (transfer ownership of currency units)

• Mining Nodes → The new public cloud

• Store:
  • General Assets (e.g., cars, houses, etc)

• Transact on:
From Cryptocurrencies to Global Assets

• So far, Mining Node:
  • Store cryptocurrency units
  • Store ownership
  • Execute Transactions (transfer ownership of currency units)

• Mining Nodes → The new public cloud

• Store:
  • General Assets (e.g., cars, houses, etc)

• Transact on:
  • General Assets (e.g., buy a house, rent a car etc)
Smart Contracts
Smart Contracts
Smart Contracts

• Alice registers her car
Smart Contracts

- Alice registers her car
  - Make: Honda
  - Model: Civic
  - Year: ..
  - VIN: ...
Smart Contracts

- Alice registers her car
  - Make: Honda
  - Model: Civic
  - Year: ..
  - VIN: ...
  - Owner: Alice
  - Price: x ethers
Smart Contracts

- Alice registers her car
  - Make: Honda
  - Model: Civic
  - Year: ..
  - VIN: ...
  - Owner: Alice
  - Price: x ethers

Buy () {
    // transfer ownership code
}

Smart Contracts
Smart Contracts
Smart Contracts

1.8 BTC
Smart Contracts
Smart Contracts
Smart Contracts
Challenges

• Asset Authenticity
• Double Spending
  • Deploy two smart contracts for the same car
  • On the same blockchain or different blockchains
• Legality
  • Implementing taxation laws
Permissioned and Permissionless Unite!

• Permissioned Blockchains
  • Requires trust
  • Trust can be distributed among several organizations
    • Banks
    • Governments
    • NGOs
Global Asset Management
Global Asset Management
Global Asset Management
Global Asset Management

Permissioned Blockchain

DMV SB

Permissioned Blockchain

DMV SD

Permissionless Blockchain
Global Asset Management

Asset Registration

Permissioned Blockchain

DMV SB

Permissionless Blockchain

DMV SD

Asset Registration
Global Asset Management

Asset Registration

Smart Contract Deployment

Permissioned Blockchain

DMV SB

Permissionless Blockchain

DMV SD

Asset Registration

Smart Contract Deployment
Global Asset Management

Asset Registration

DMV SB

Permissioned Blockchain

DMV SD

Permissioned Blockchain

Smart Contract Deployment

Permissionless Blockchain

Smart Contract Deployment

Asset Trading
Challenges Revisited
Challenges Revisited

• Asset Authenticity
Challenges Revisited

• Asset Authenticity
  • Authenticated by the permissioned blockchain
Challenges Revisited

• Asset Authenticity
  • Authenticated by the permissioned blockchain
• Double Spending
Challenges Revisited

• Asset Authenticity
  • Authenticated by the permissioned blockchain

• Double Spending
  • Permissioned blockchain:
    • Allows the deployment of one contract per asset at a time
    • Enables moving the asset from one Permissionless blockchain to another
Challenges Revisited

• Asset Authenticity
  • Authenticated by the permissioned blockchain

• Double Spending
  • Permissioned blockchain:
    • Allows the deployment of one contract per asset at a time
    • Enables moving the asset from one Permissionless blockchain to another

• Legality
Challenges Revisited

• Asset Authenticity
  • Authenticated by the permissioned blockchain

• Double Spending
  • Permissioned blockchain:
    • Allows the deployment of one contract per asset at a time
    • Enables moving the asset from one Permissionless blockchain to another

• Legality
  • Encode the Taxation law in the smart contract code
Open research questions

• Scalability
• Identity theft
• Flexibility of asset marketing
Blockchain: Panacea for all our data problems?

- Resource cost:
  - Proof-of-work consumes resources at the planetary scale

- Mythical notion of democratization:
  - Handful of miners control the progress of Bitcoin blockchain

- False notion of security:
  - An Individual vulnerable to the security of his/her key

- Extreme distribution:
  - is it really worth it?

- Extreme redundancy:
  - is it really necessary?

- Social consequences:
  - Are we comfortable if this technology is used for dark causes?