

Database and Distributed Computing Foundations of Blockchains

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

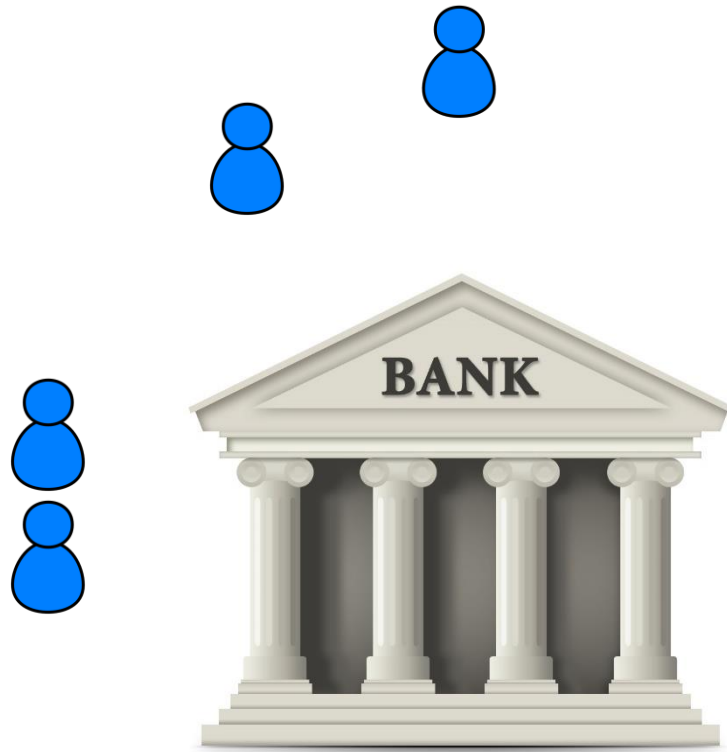
{sujaya-maiyya, victorzakhary, amiri, agrawal, amr}@cs.ucsb.edu

Traditional Banking Systems

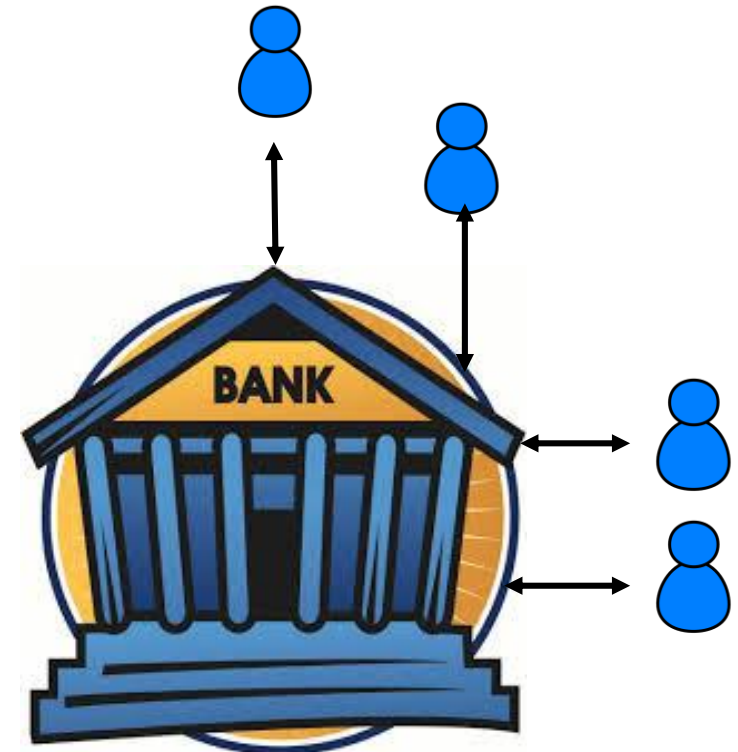
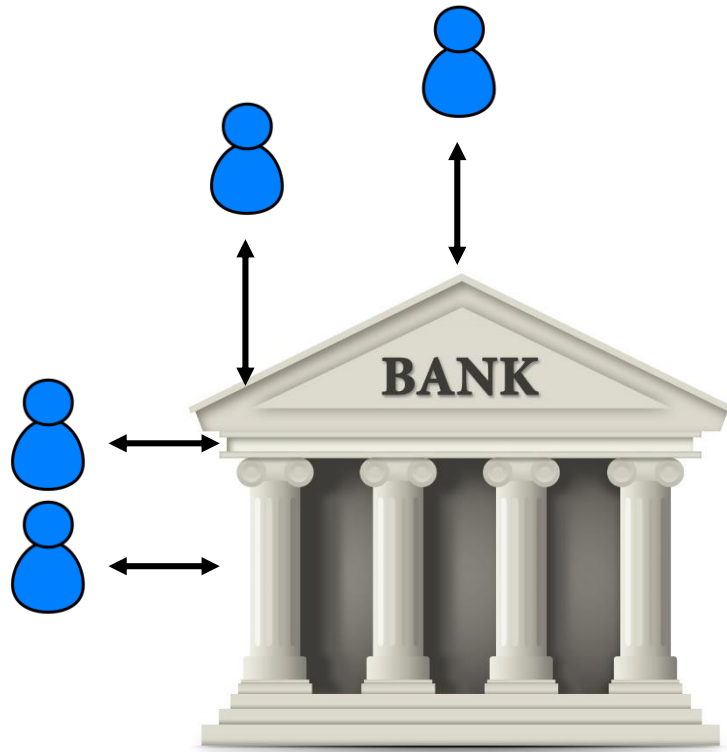
Traditional Banking Systems



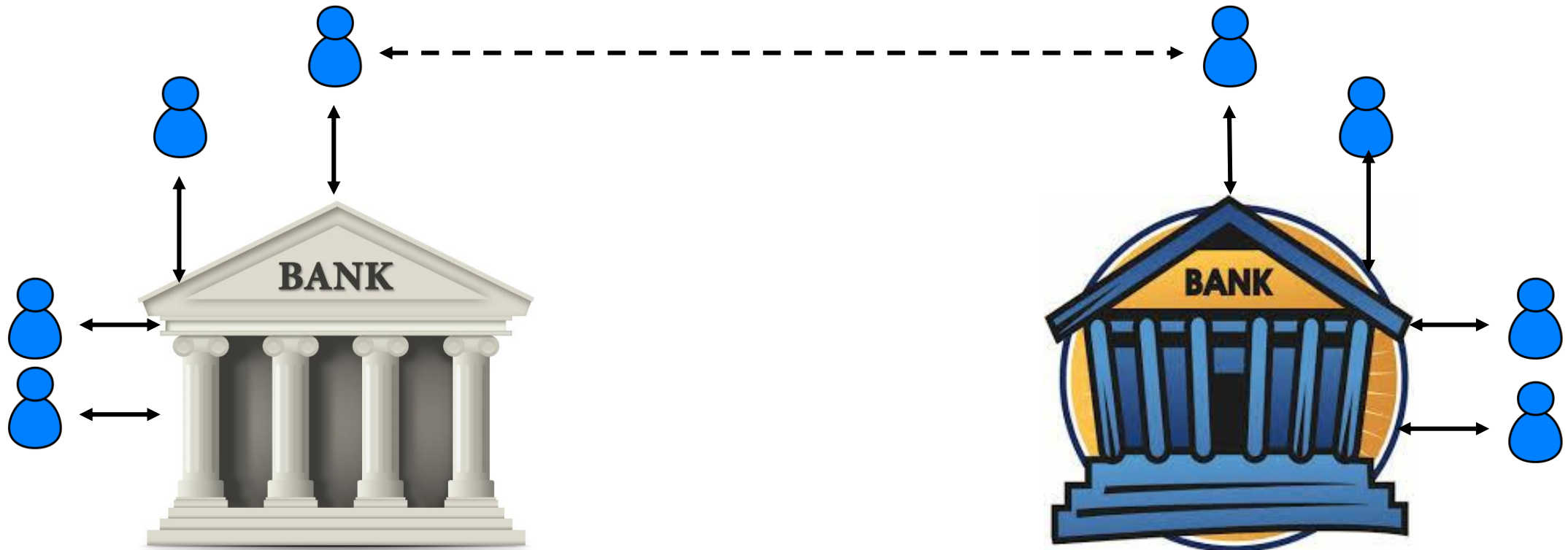
Traditional Banking Systems



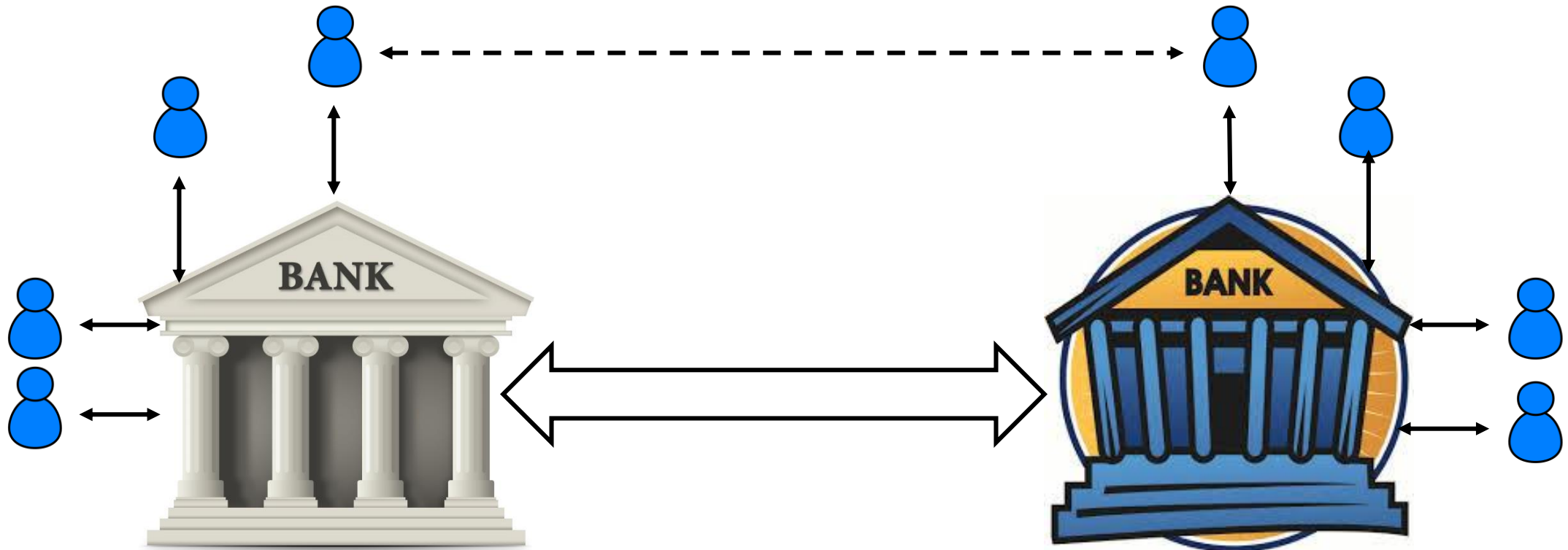
Traditional Banking Systems



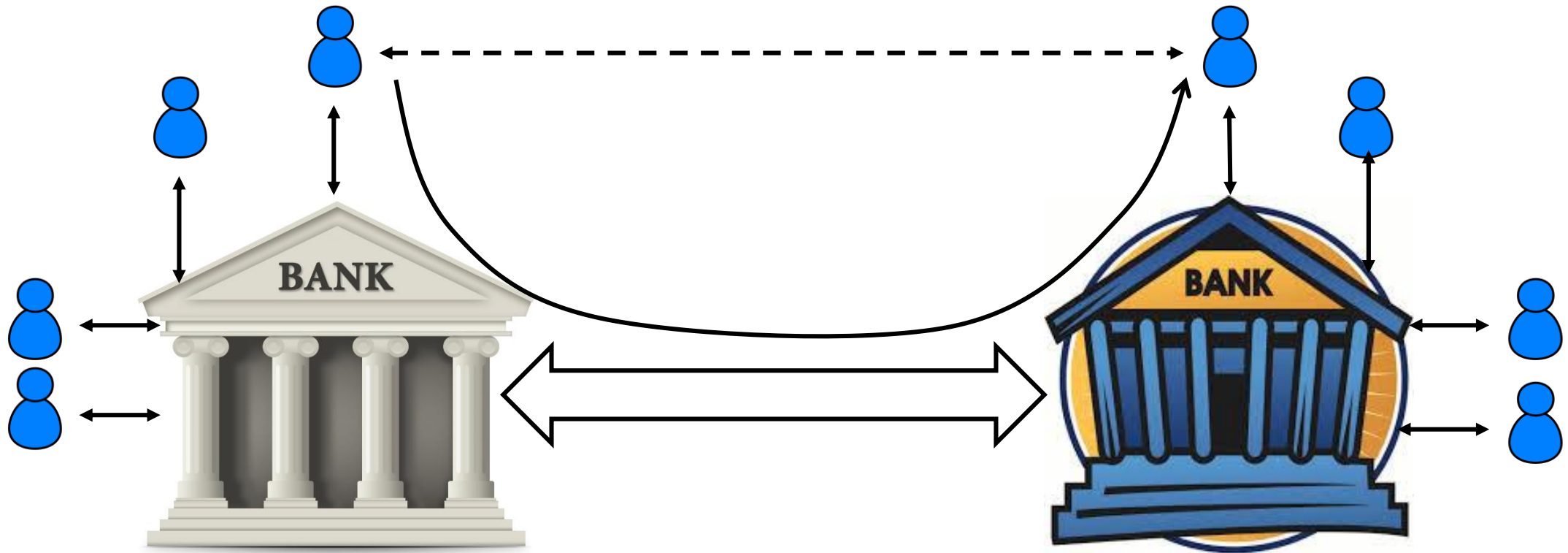
Traditional Banking Systems



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

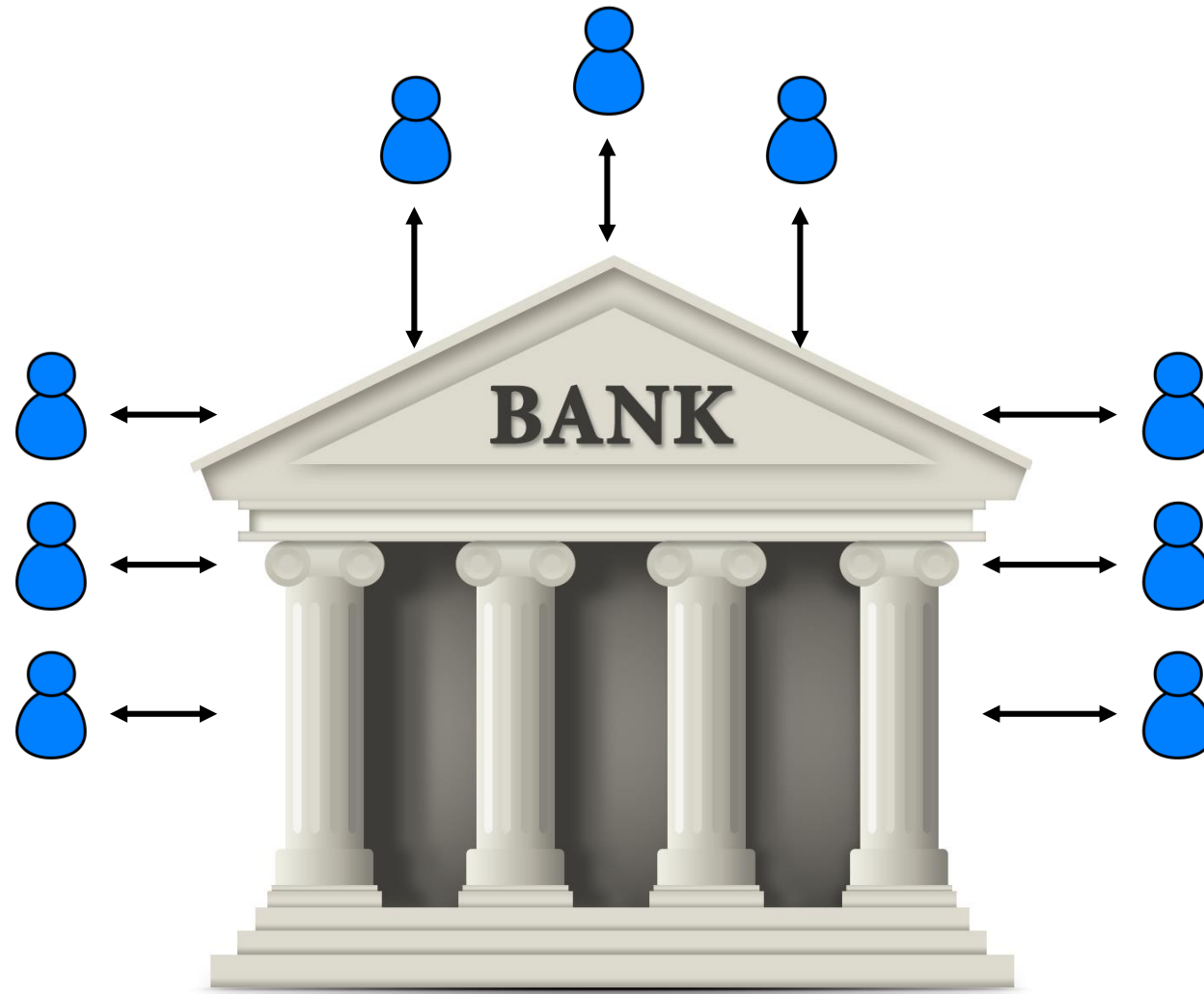


Traditional Banking Systems

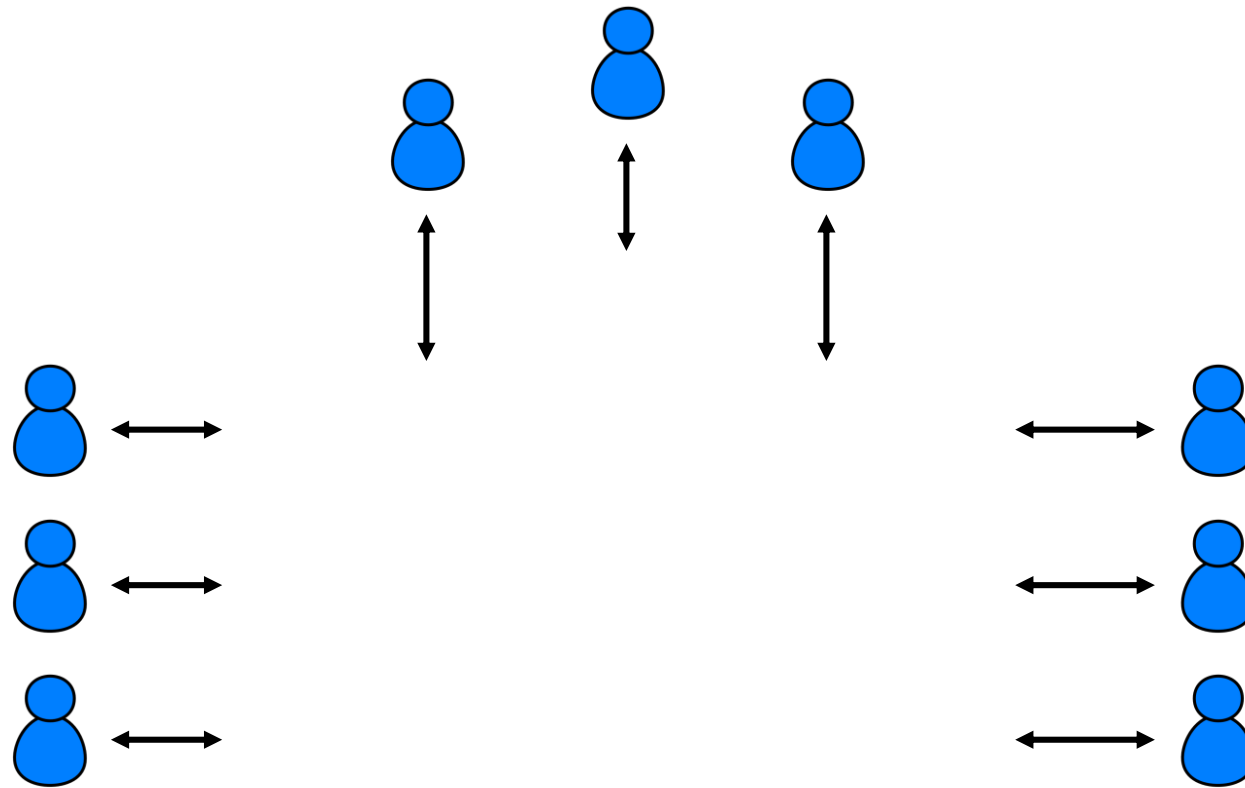
- From Database and Distributed Computing Perspective
- Identities and Signatures
 - You are your signature [ID, username and password]
- Ledger
 - The balance of each identity (saved in a DB)
- Transactions
 - Move money from one identity to another
 - Concurrency control to serialize transactions (prevent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



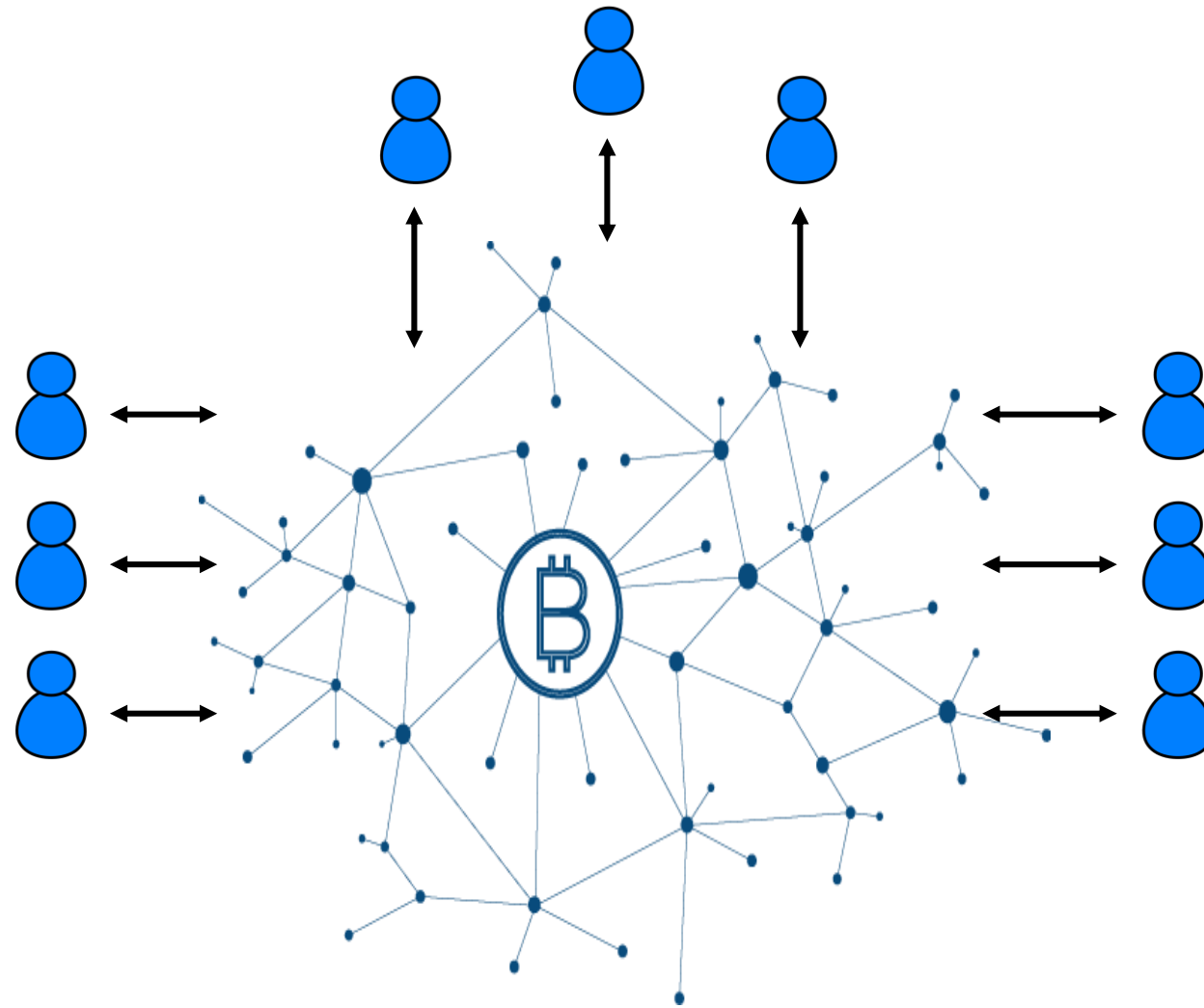
Bitcoin



Bitcoin



Bitcoin



Bitcoin: A Peer-to-Peer Electronic Cash System

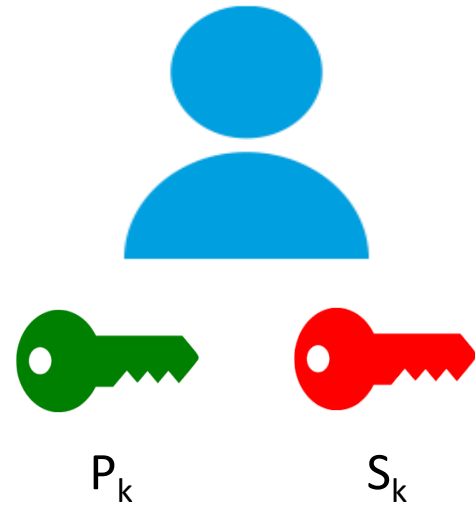
- From Database and Distributed Computing Perspective
- Identities and Signatures
 - Public/Private key pair
- Ledger
 - The balance of each identity (saved in the blockchain)
- Transactions
 - Move bitcoins from one identity to another
 - Concurrency control to serialize transactions (Mining and PoW)
 - 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



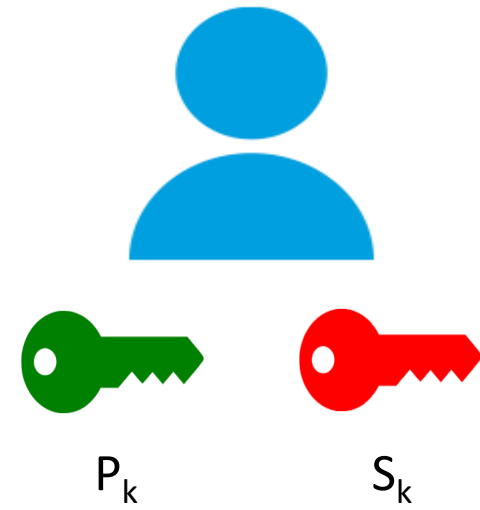
Digital Signatures

- $P_k, S_k \leftarrow \text{Keygen}(\text{keysize})$



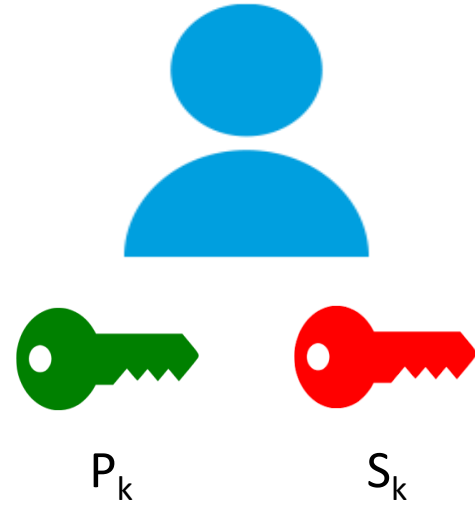
Digital Signatures

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



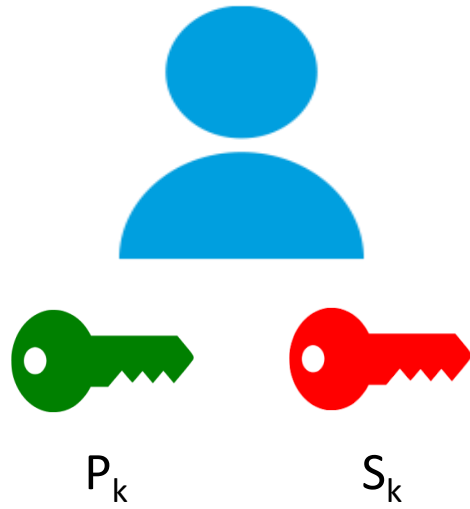
Digital Signatures

- $P_k, S_k \leftarrow \text{Keygen}(\text{keysize})$
- 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



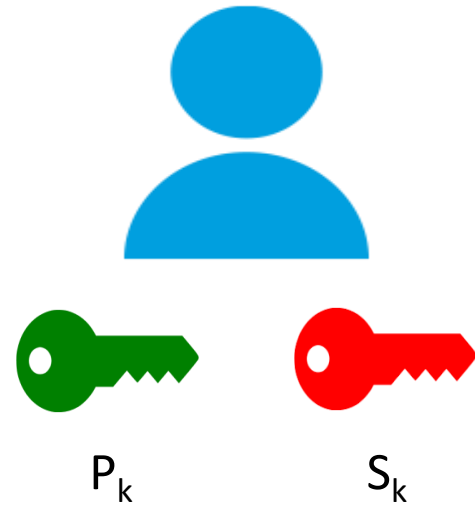
Digital Signatures

- 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

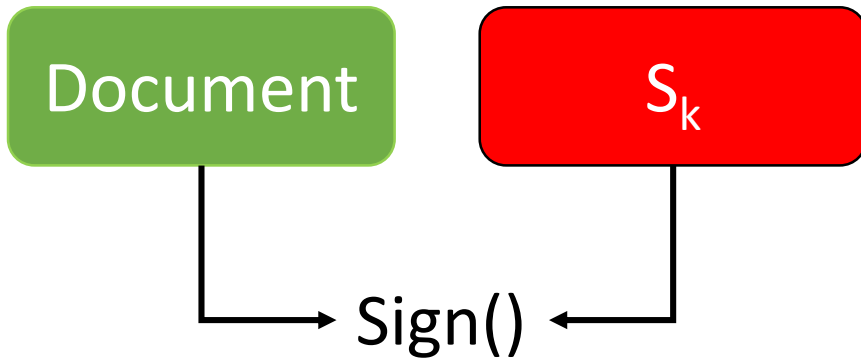
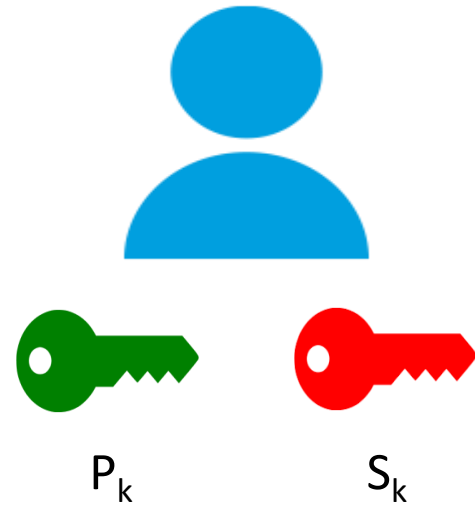


Document

S_k

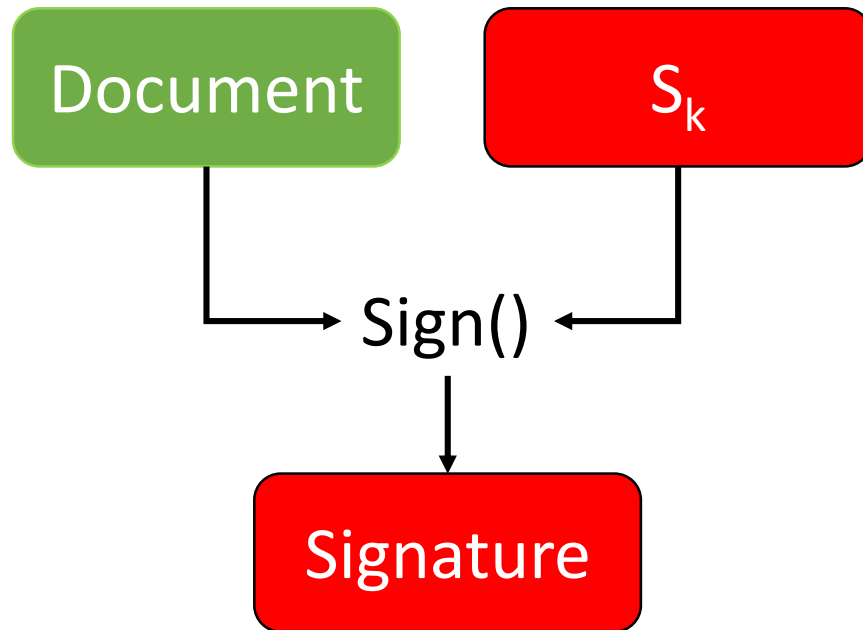
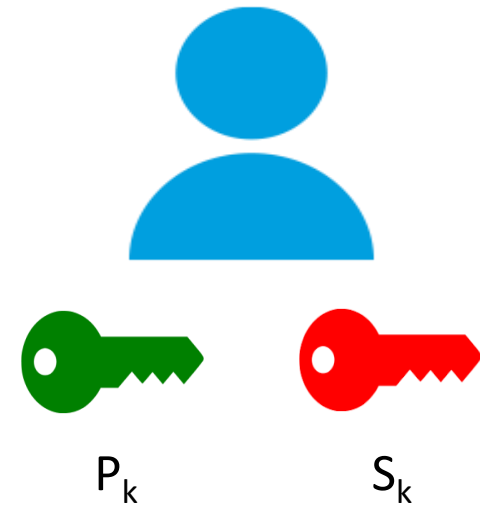
Digital Signatures

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



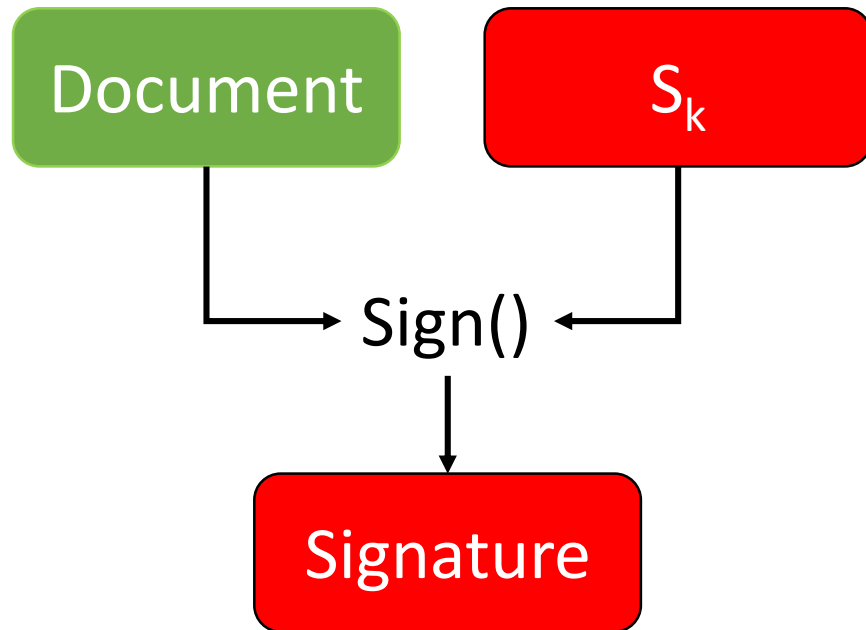
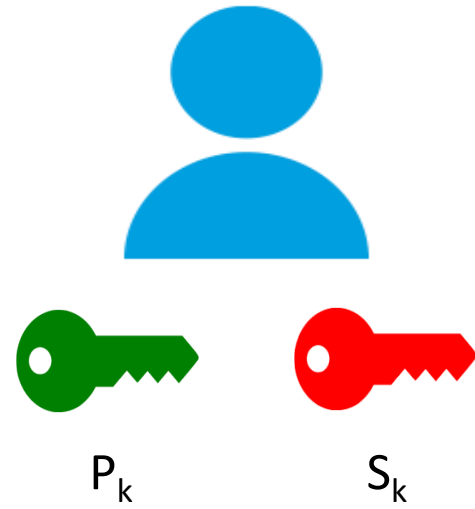
Digital Signatures

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



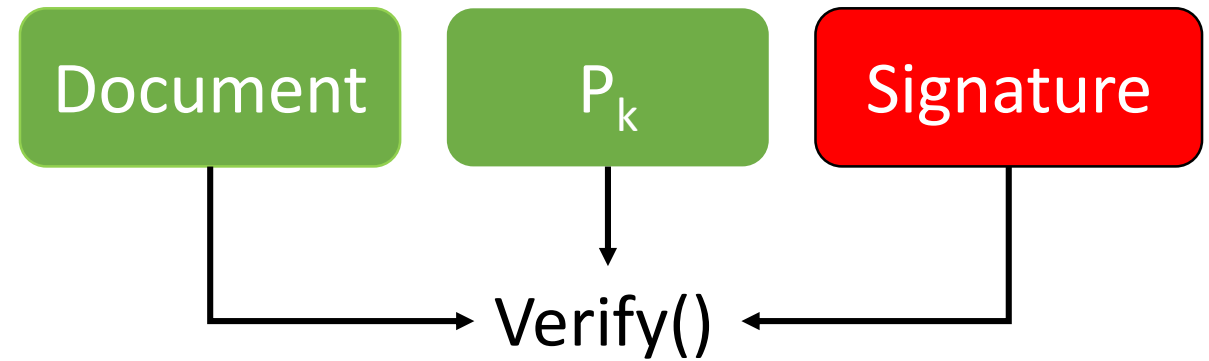
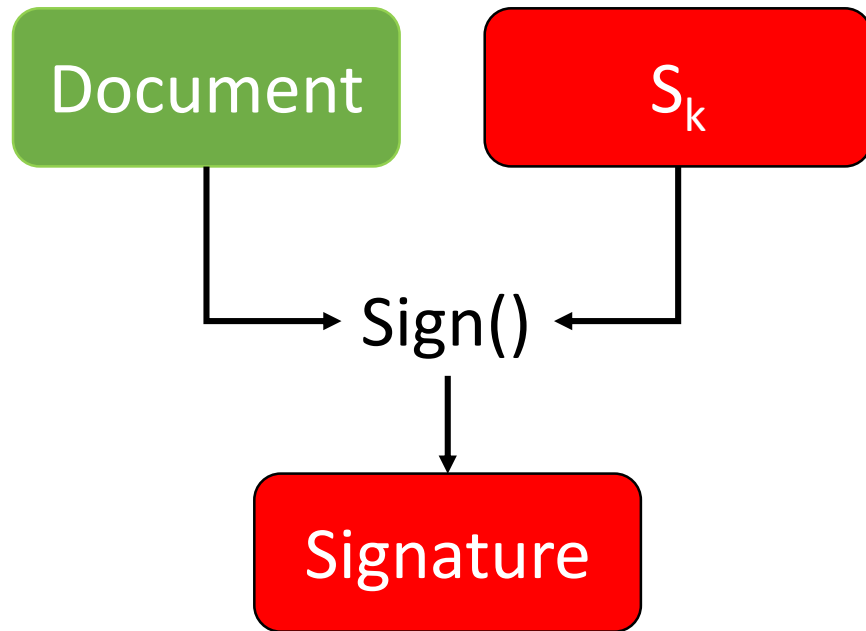
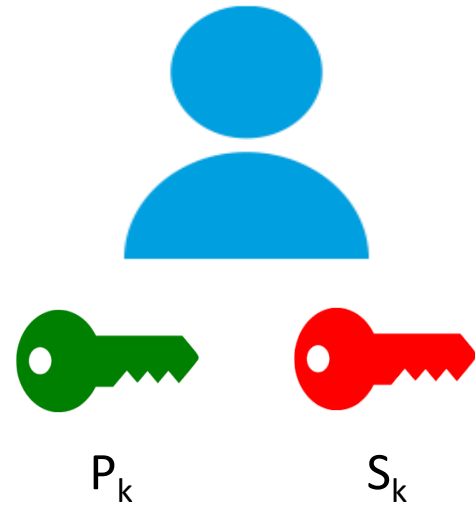
Digital Signatures

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



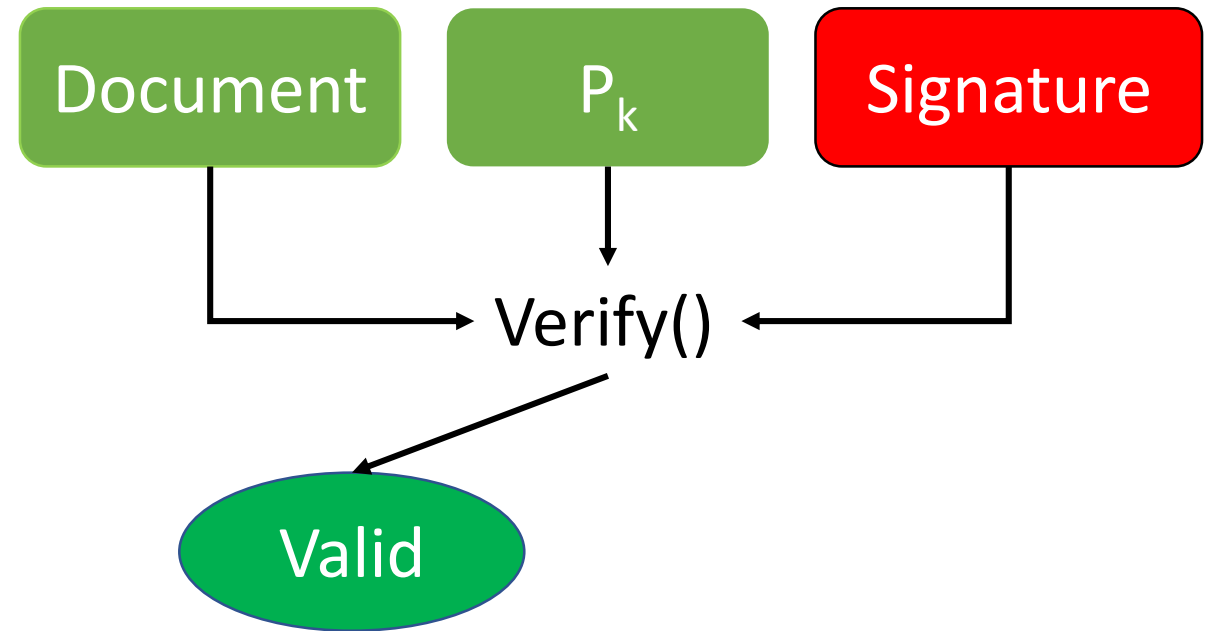
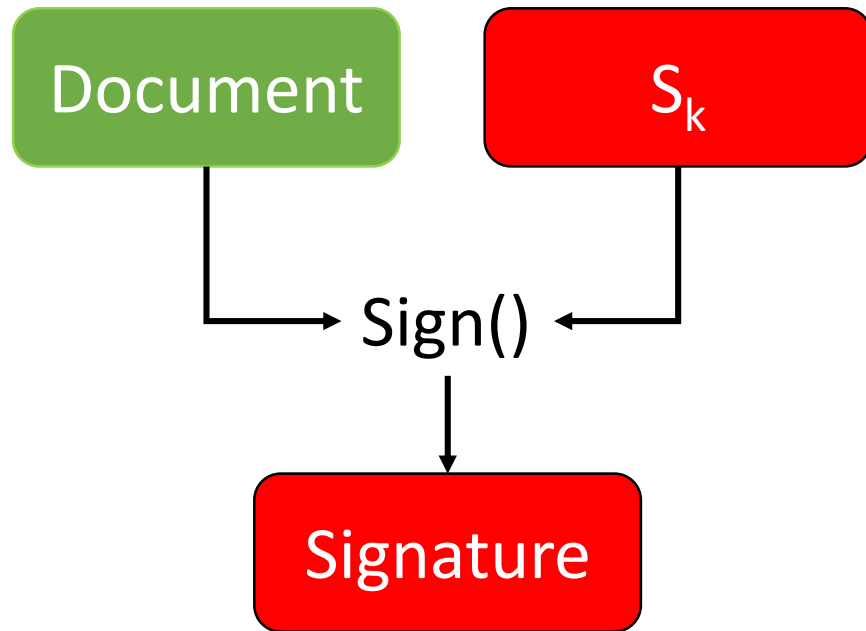
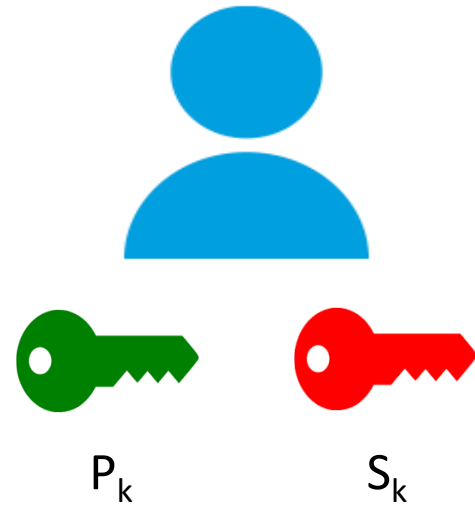
Digital Signatures

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



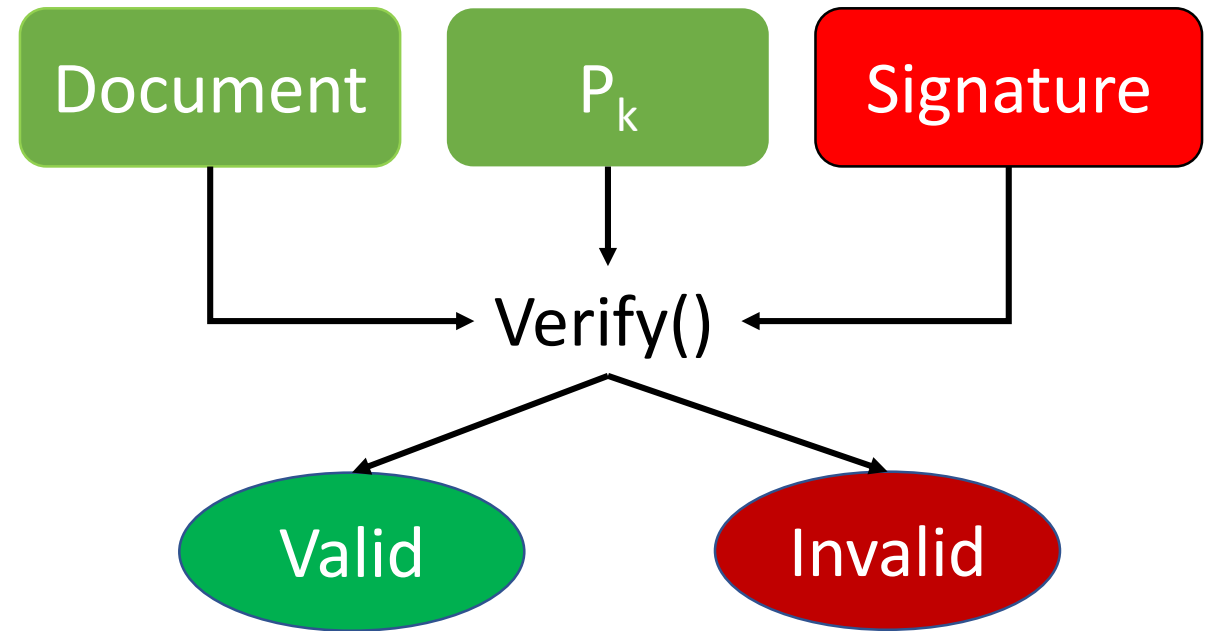
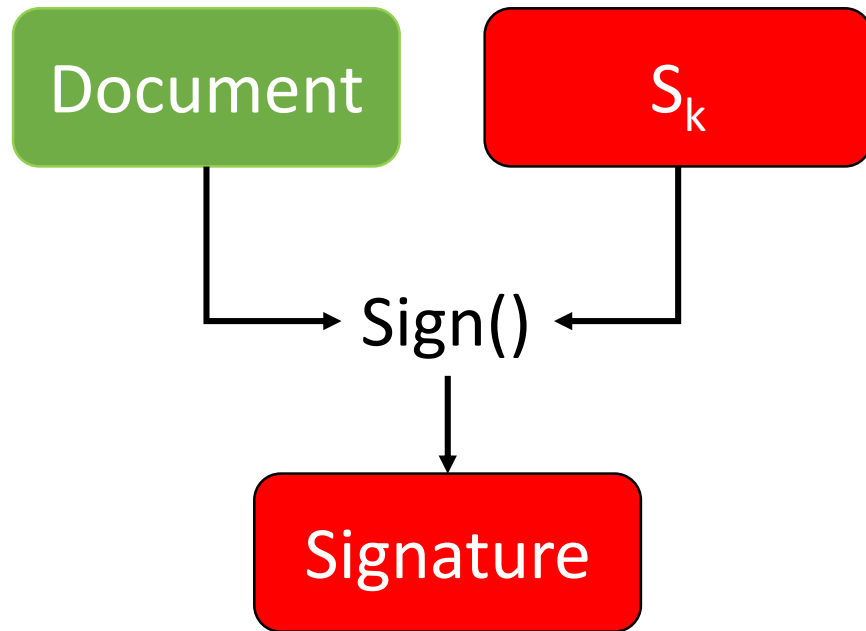
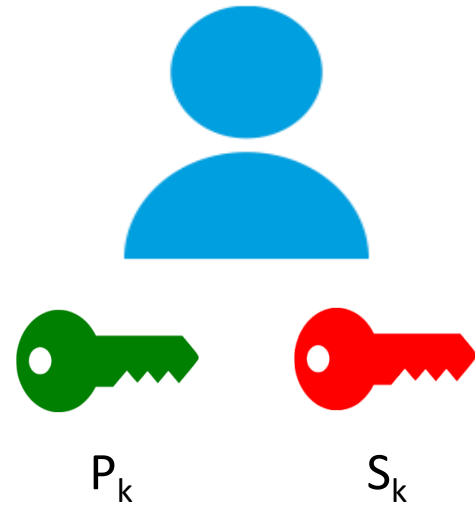
Digital Signatures

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



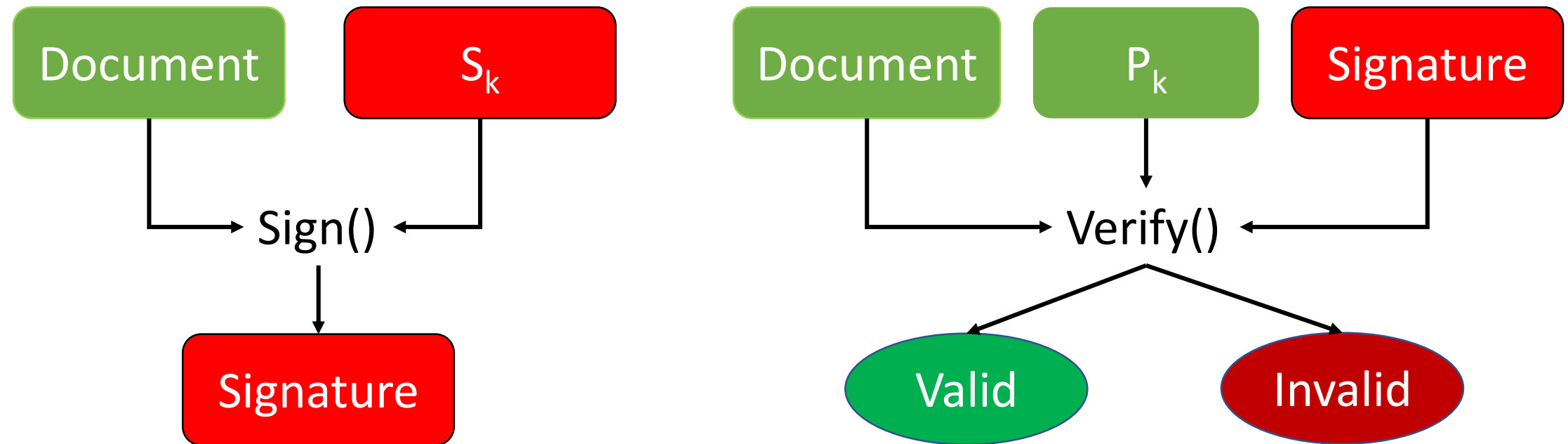
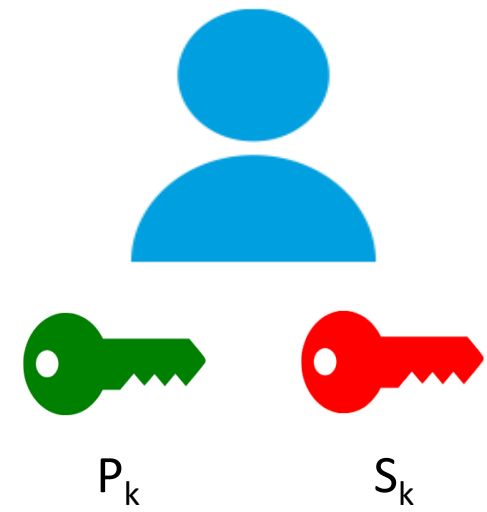
Digital Signatures

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



Digital Signatures

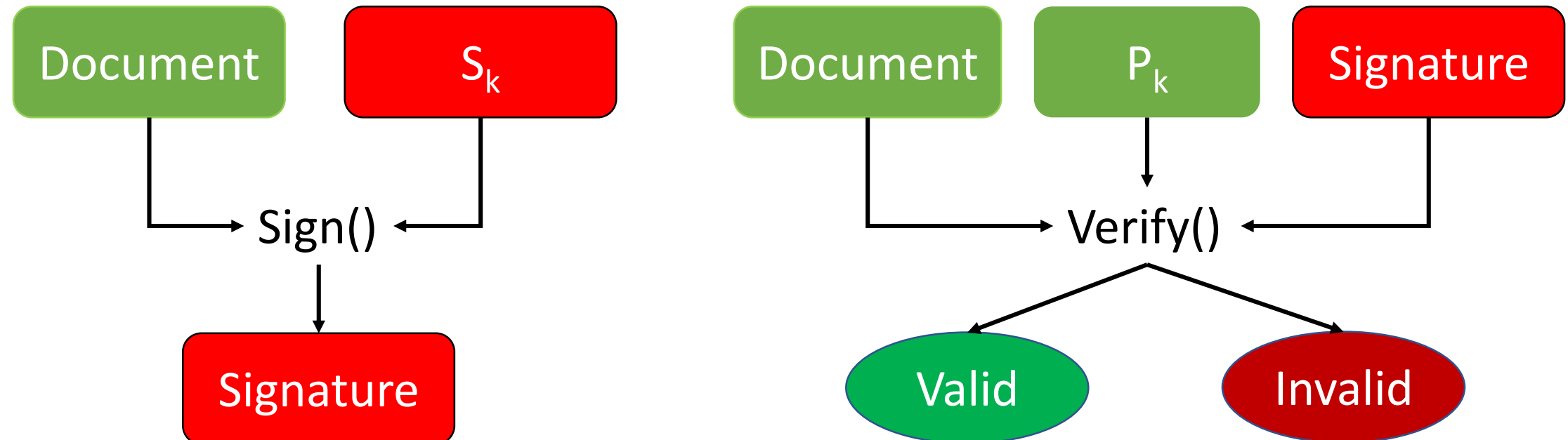
- P_k is made public and used to verify documents signed by S_k
- S_k is private



Used for Authentication not privacy

Digital Signatures

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



Digital Signatures and Bitcoin

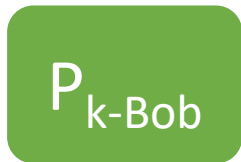
- A bitcoin is a chain of digital signatures
 - Coin owners digitally sign their coins to transfer them to other recipients

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

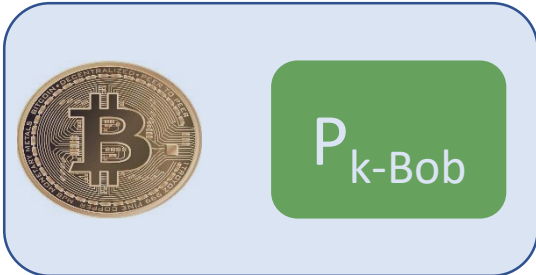
Digital Signatures and Bitcoin

- 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



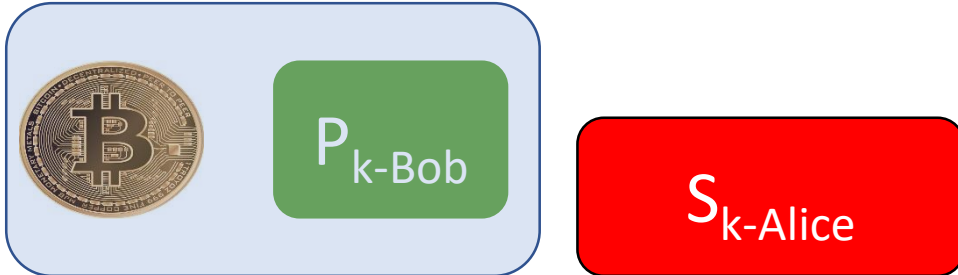
Digital Signatures and Bitcoin

- 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



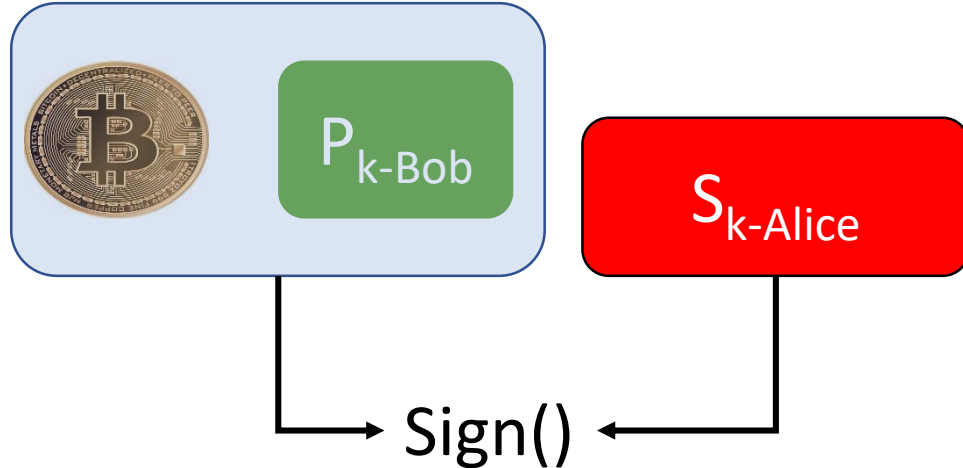
Digital Signatures and Bitcoin

- 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



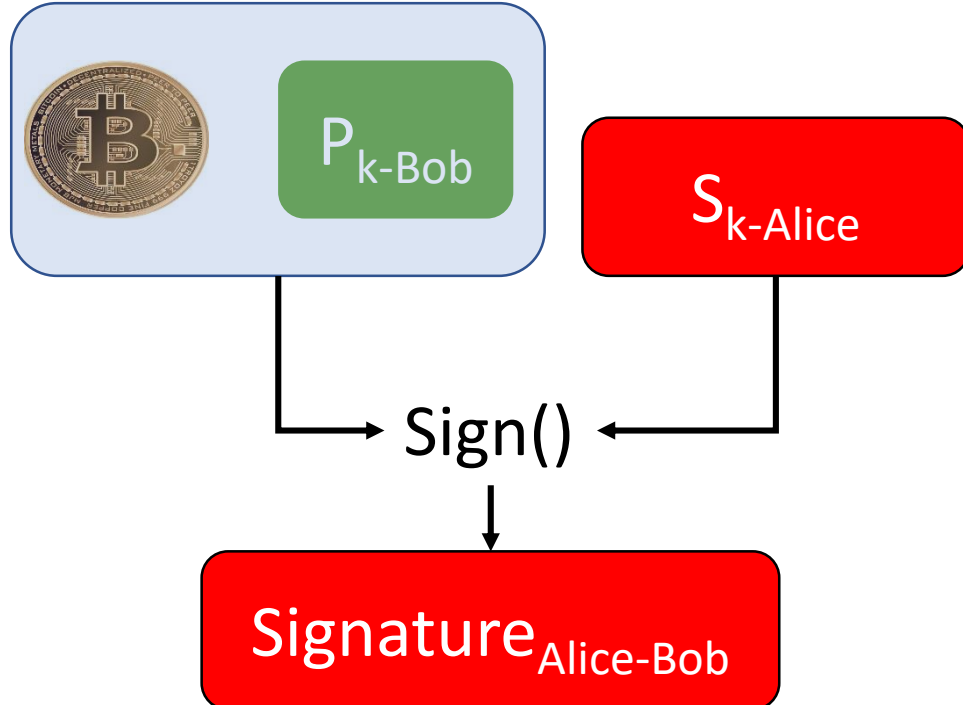
Digital Signatures and Bitcoin

- 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



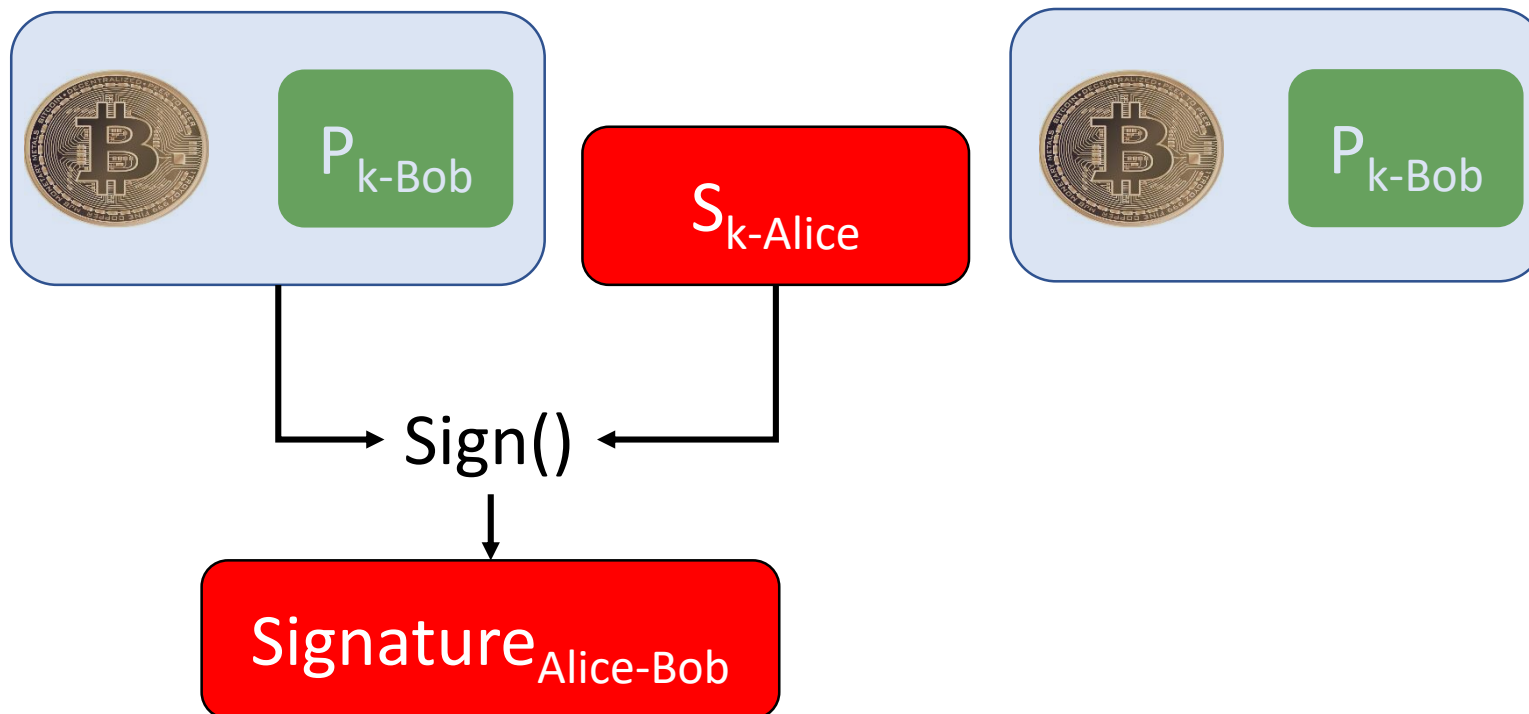
Digital Signatures and Bitcoin

- 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



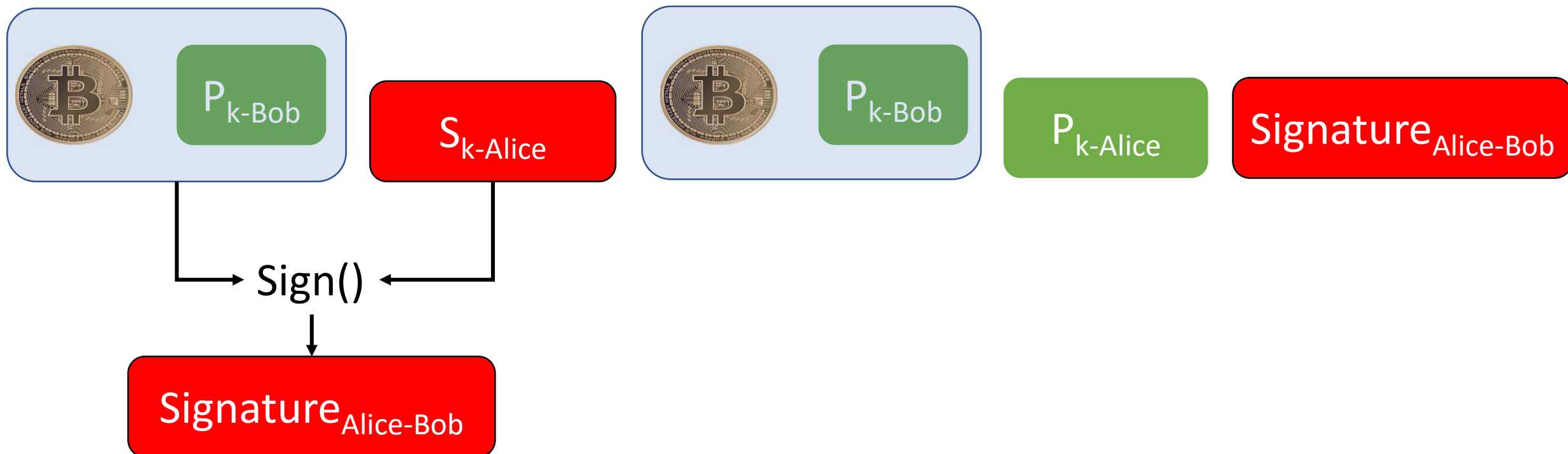
Digital Signatures and Bitcoin

- 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



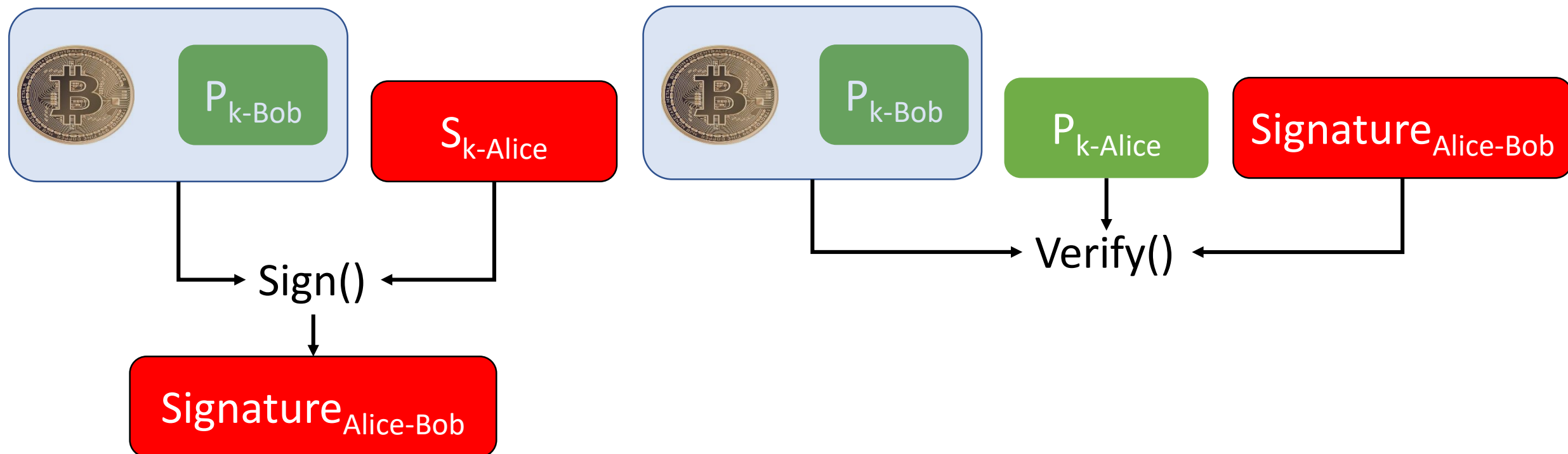
Digital Signatures and Bitcoin

- 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



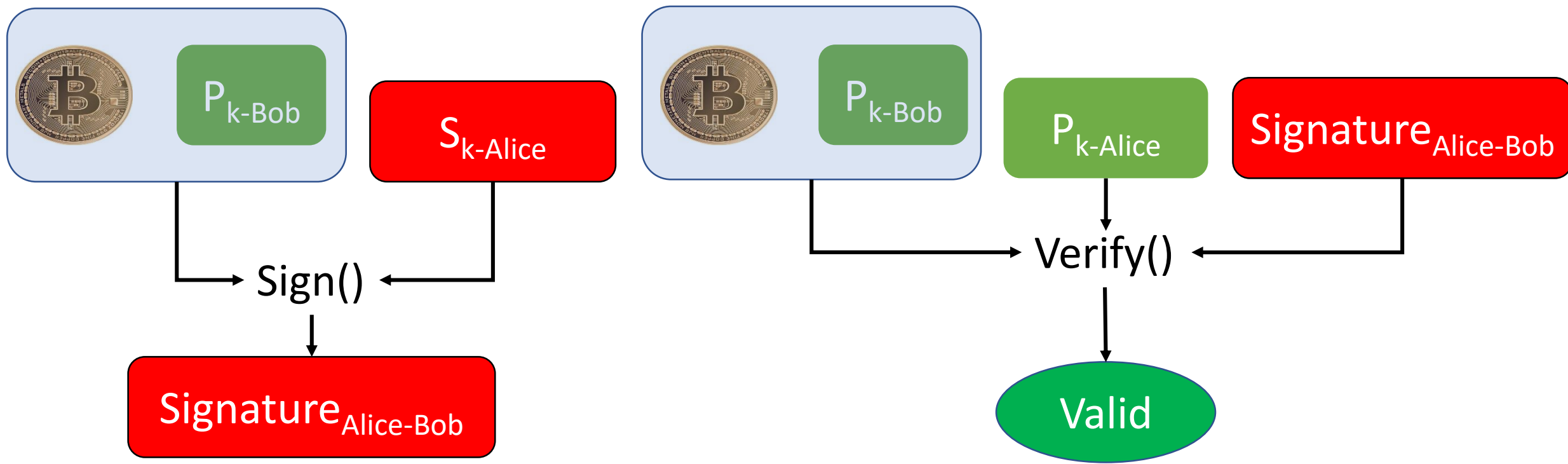
Digital Signatures and Bitcoin

- 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

- 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



Signature_{Alice-Bob}

Digital Signatures and Bitcoin

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



Signature_{Alice-Bob}

Signature_{Alice-Bob}

$P_{k-Diana}$

Digital Signatures and Bitcoin

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



Signature_{Alice-Bob}

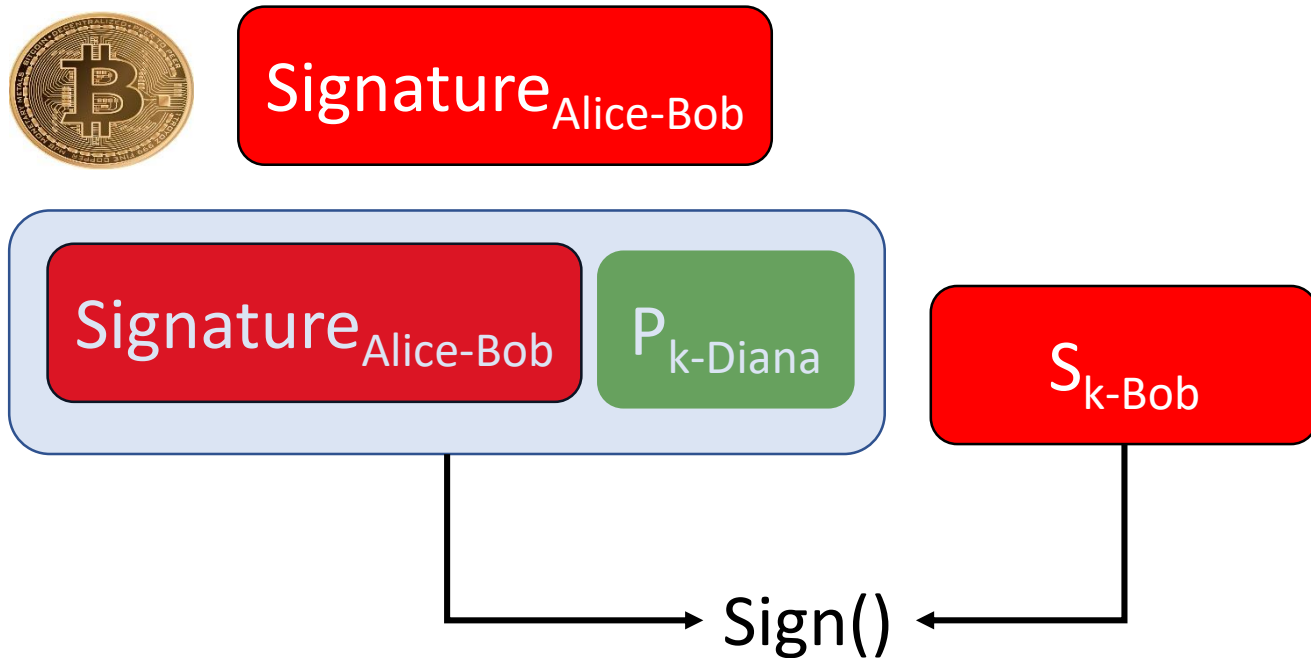
Signature_{Alice-Bob}

$P_{k-Diana}$

S_{k-Bob}

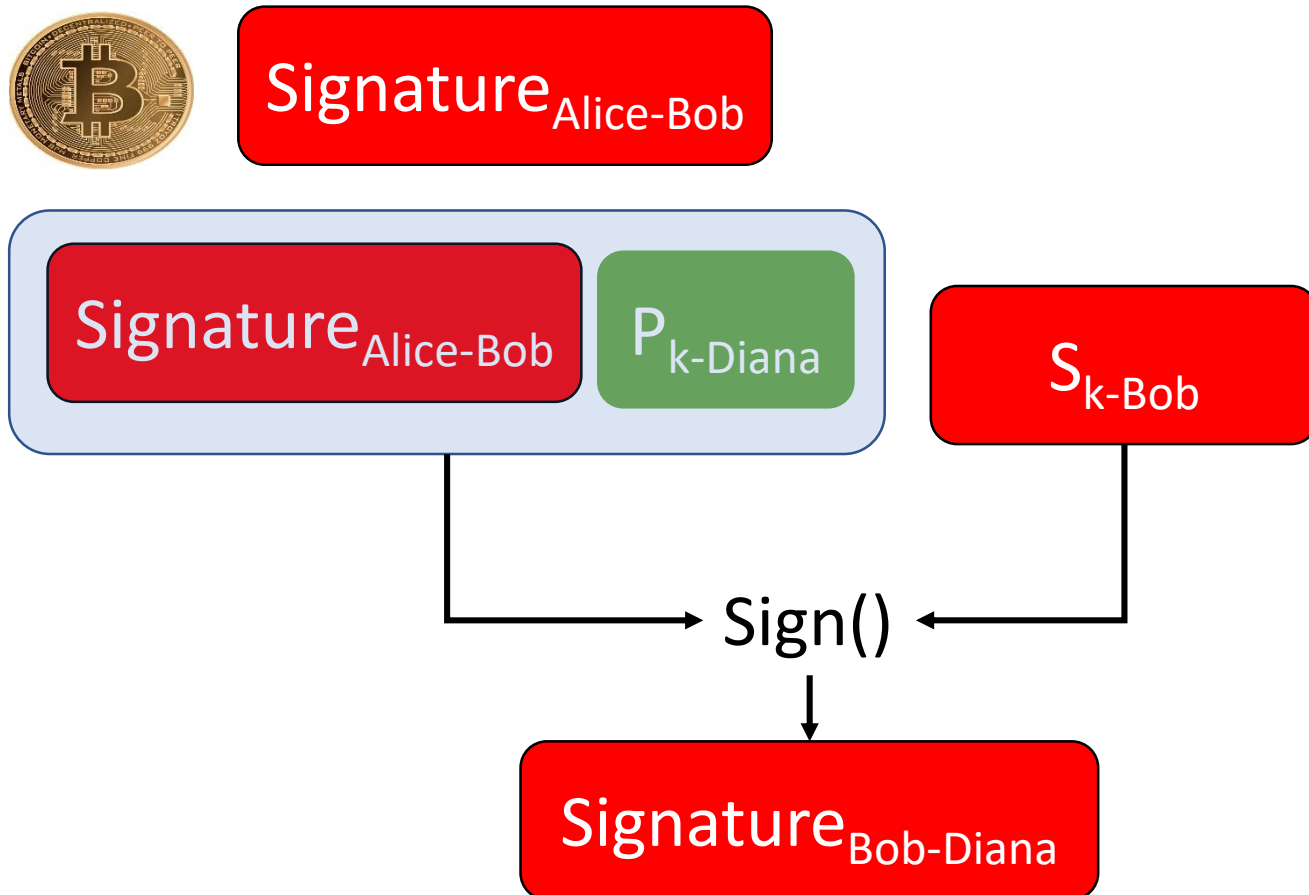
Digital Signatures and Bitcoin

- 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



A Bitcoin Big Picture

A Bitcoin Big Picture

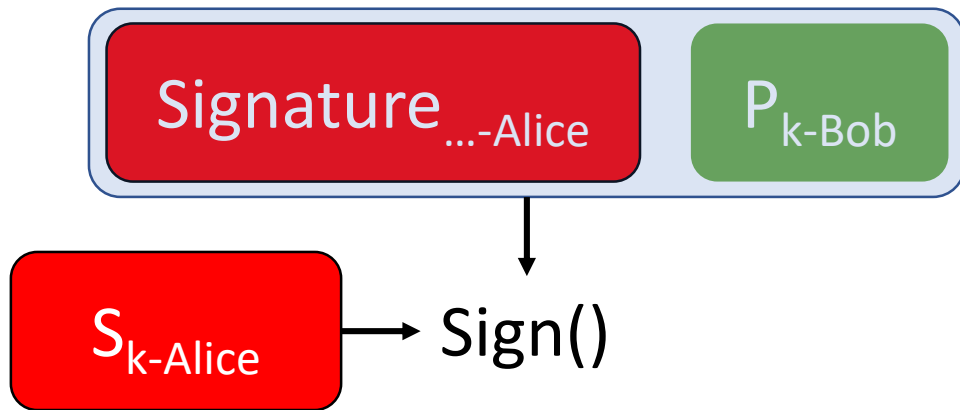
Signature...-Alice

A Bitcoin Big Picture

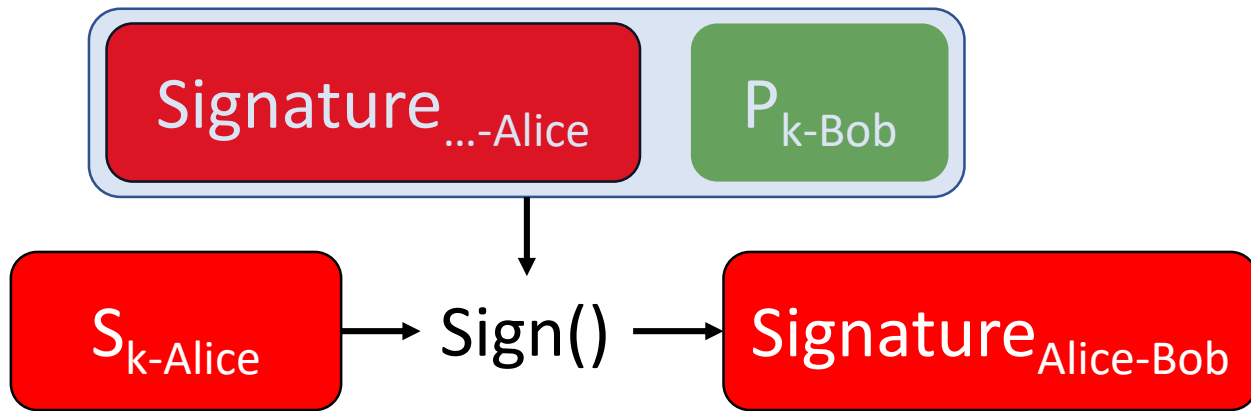
Signature...-Alice

$P_{k\text{-Bob}}$

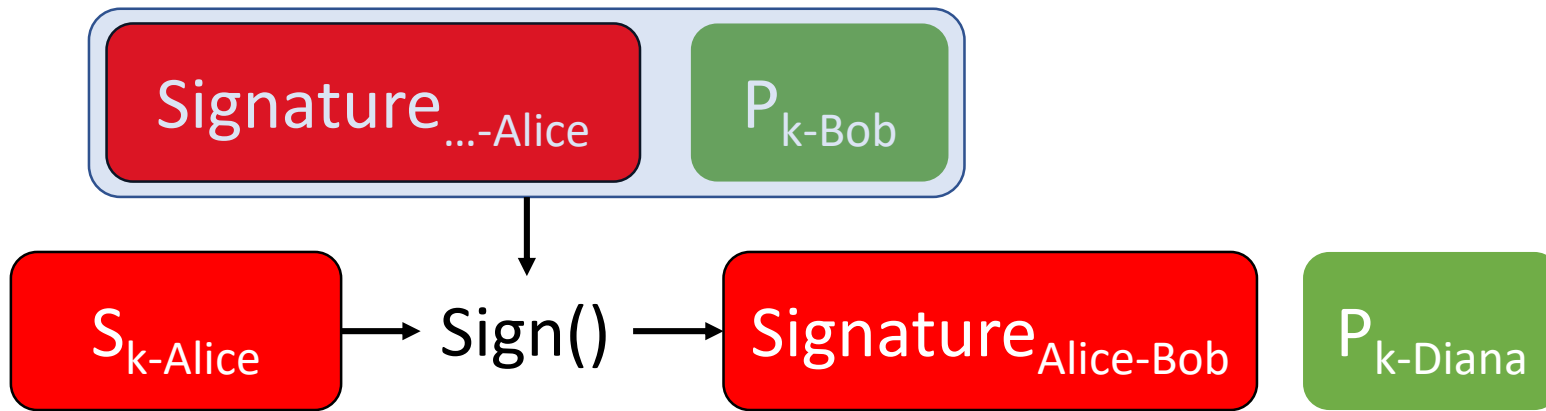
A Bitcoin Big Picture



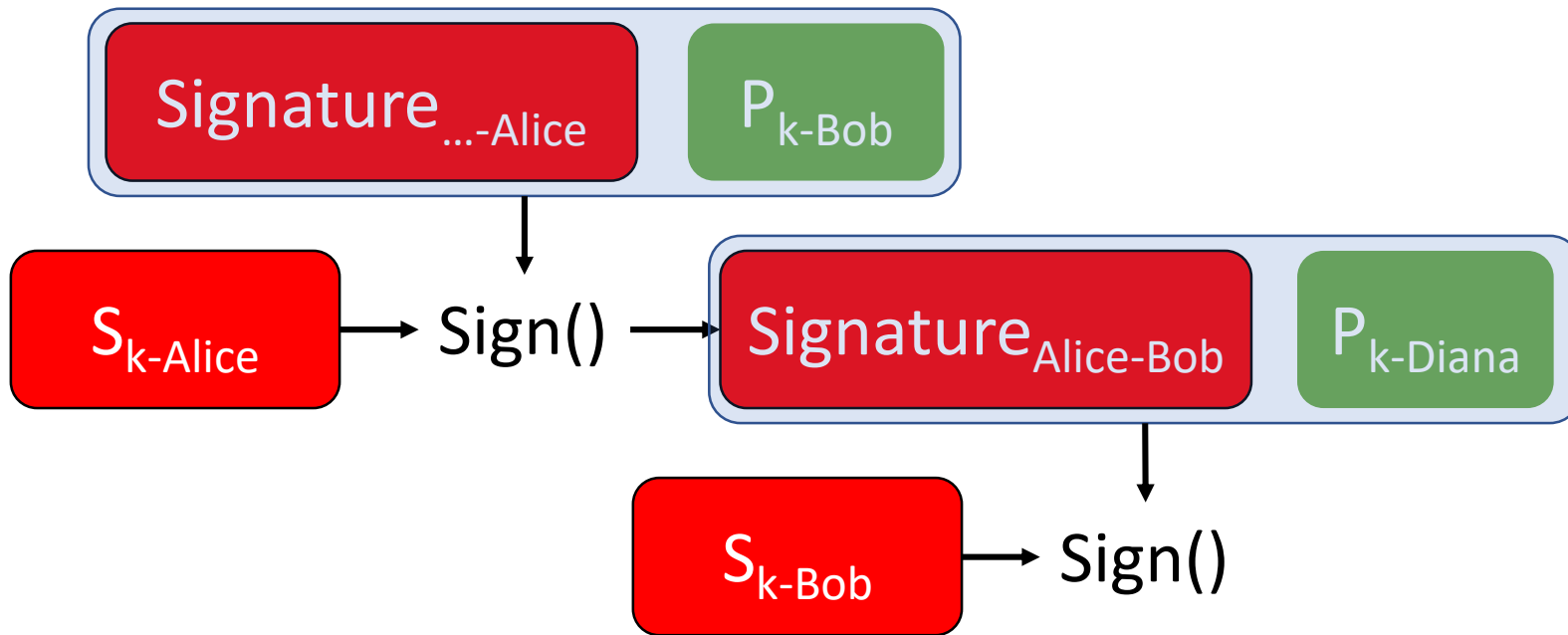
A Bitcoin Big Picture



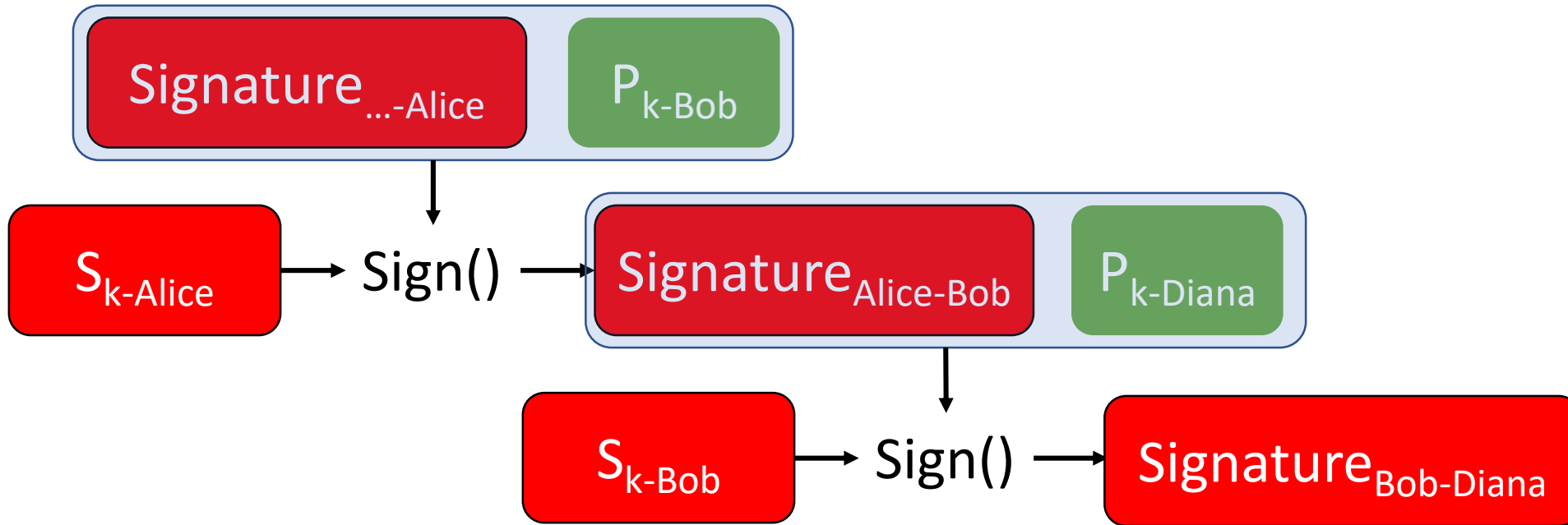
A Bitcoin Big Picture



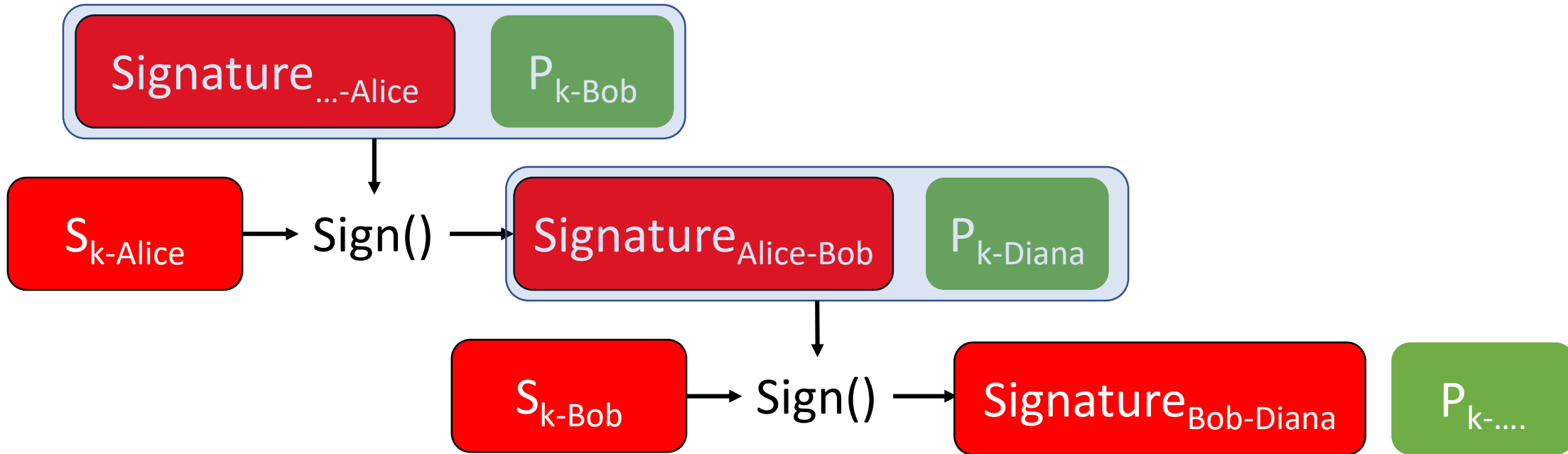
A Bitcoin Big Picture



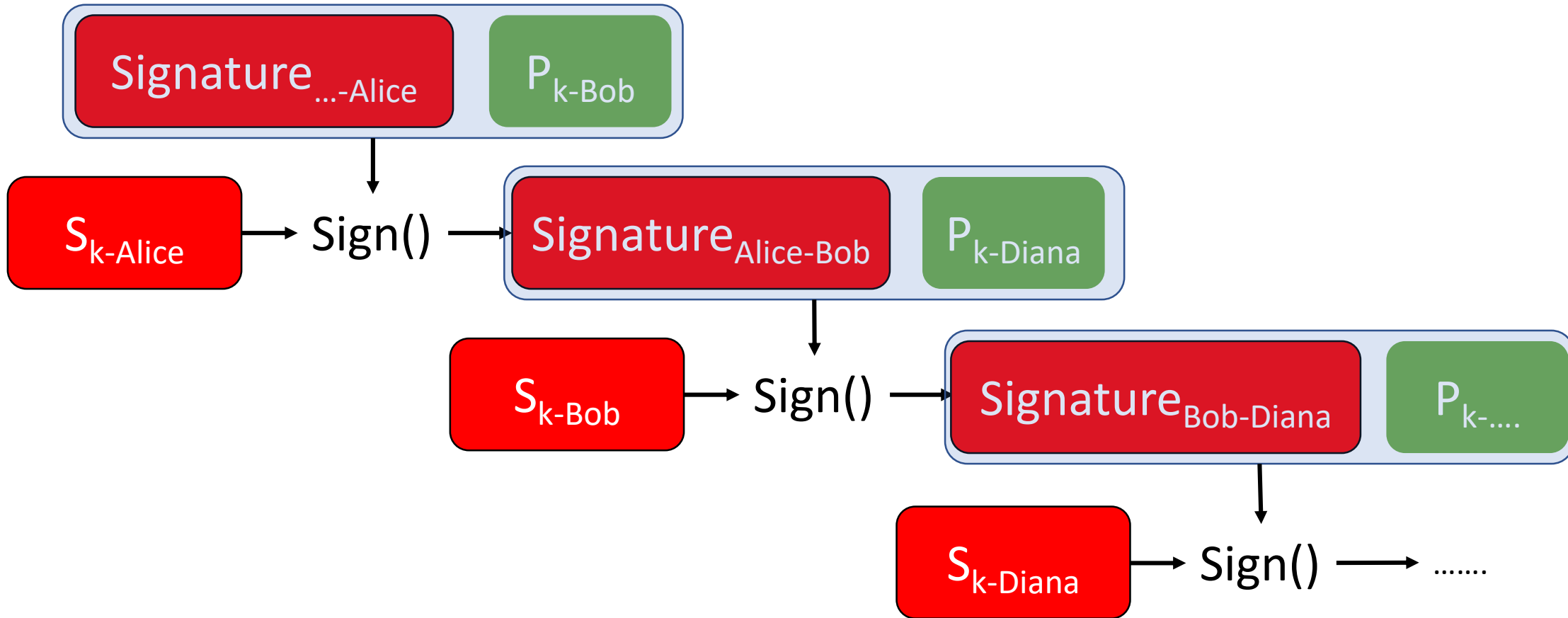
A Bitcoin Big Picture



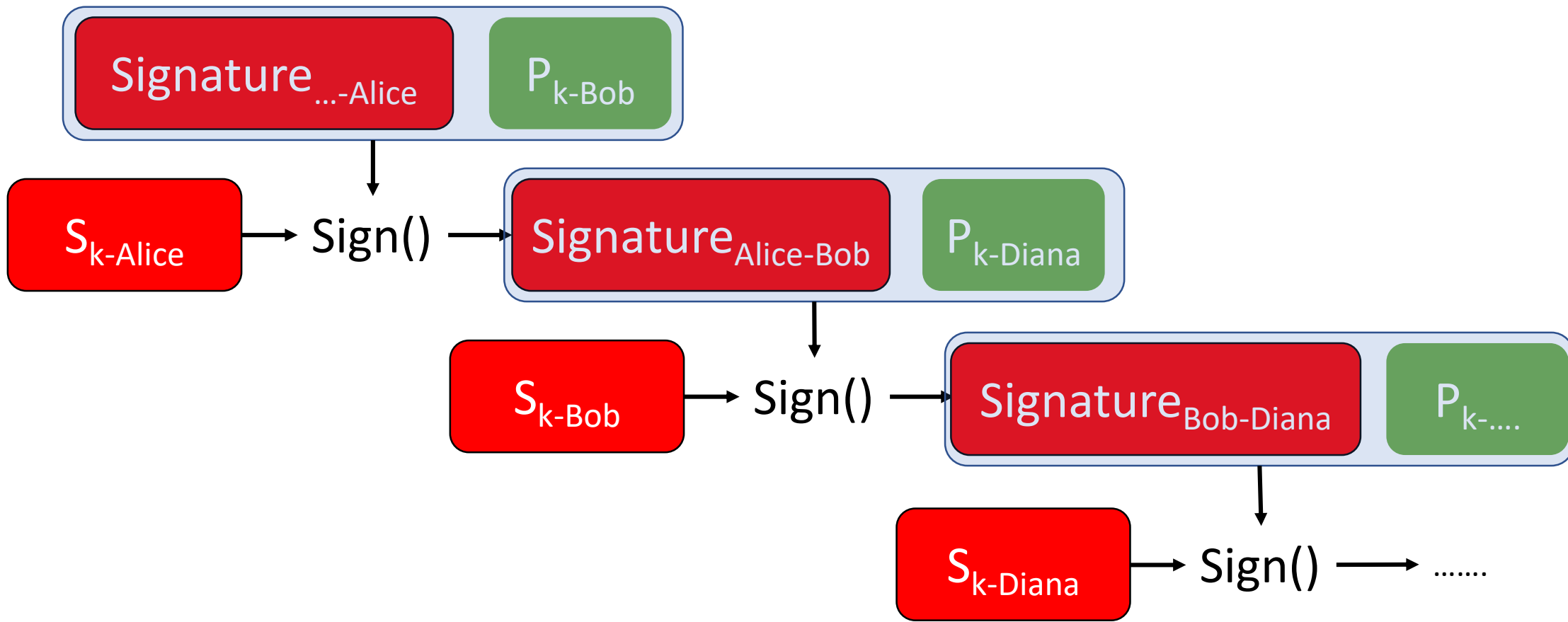
A Bitcoin Big Picture



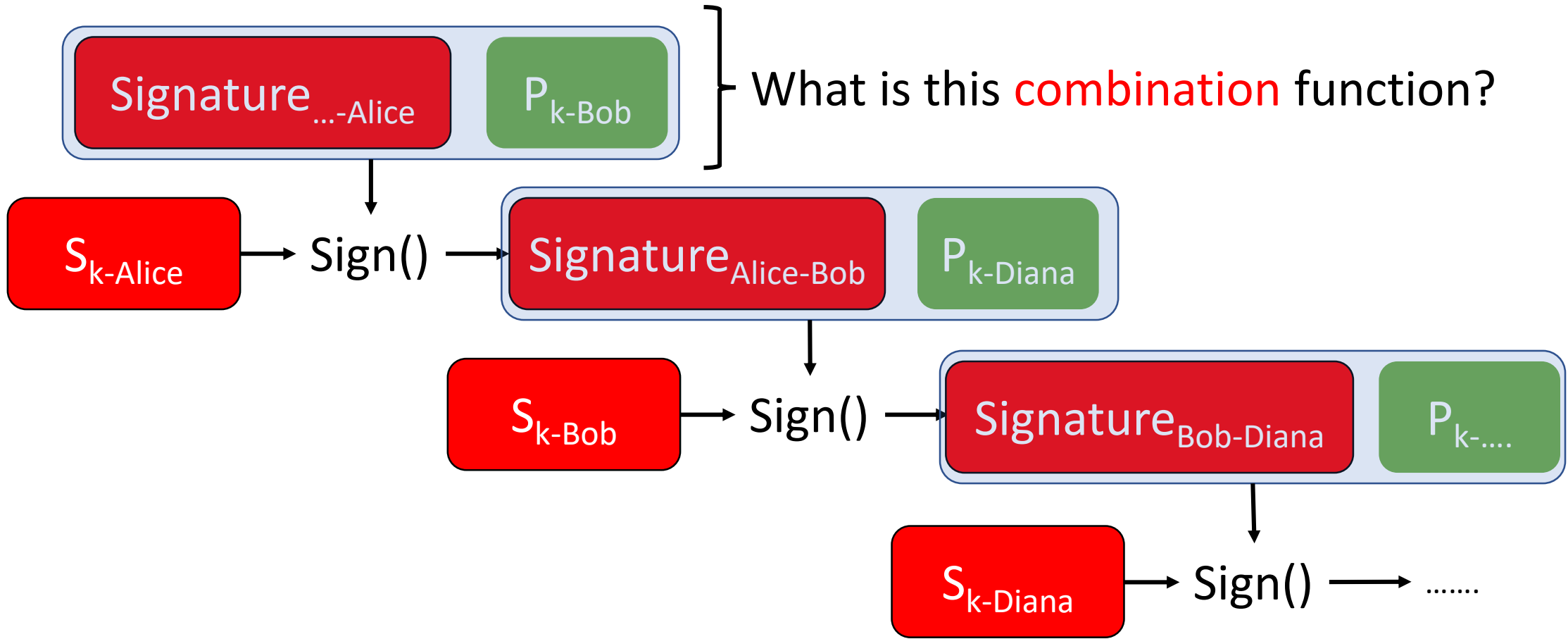
A Bitcoin Big Picture



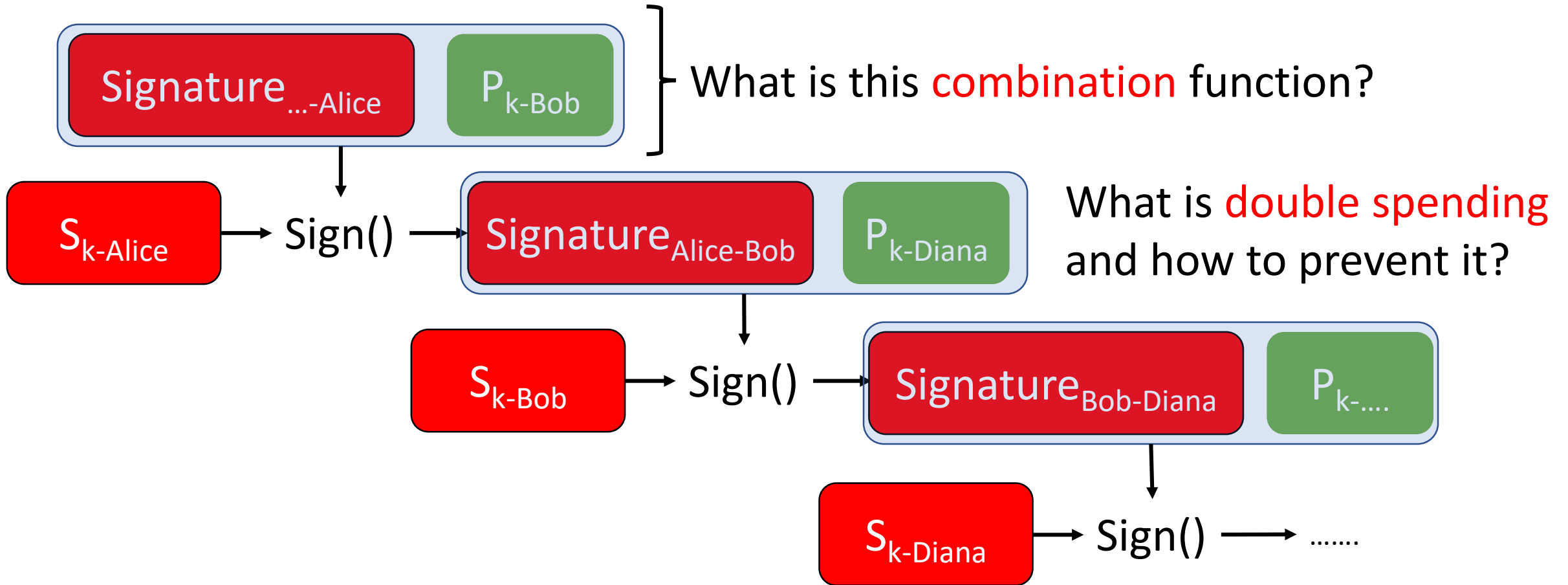
What About's?



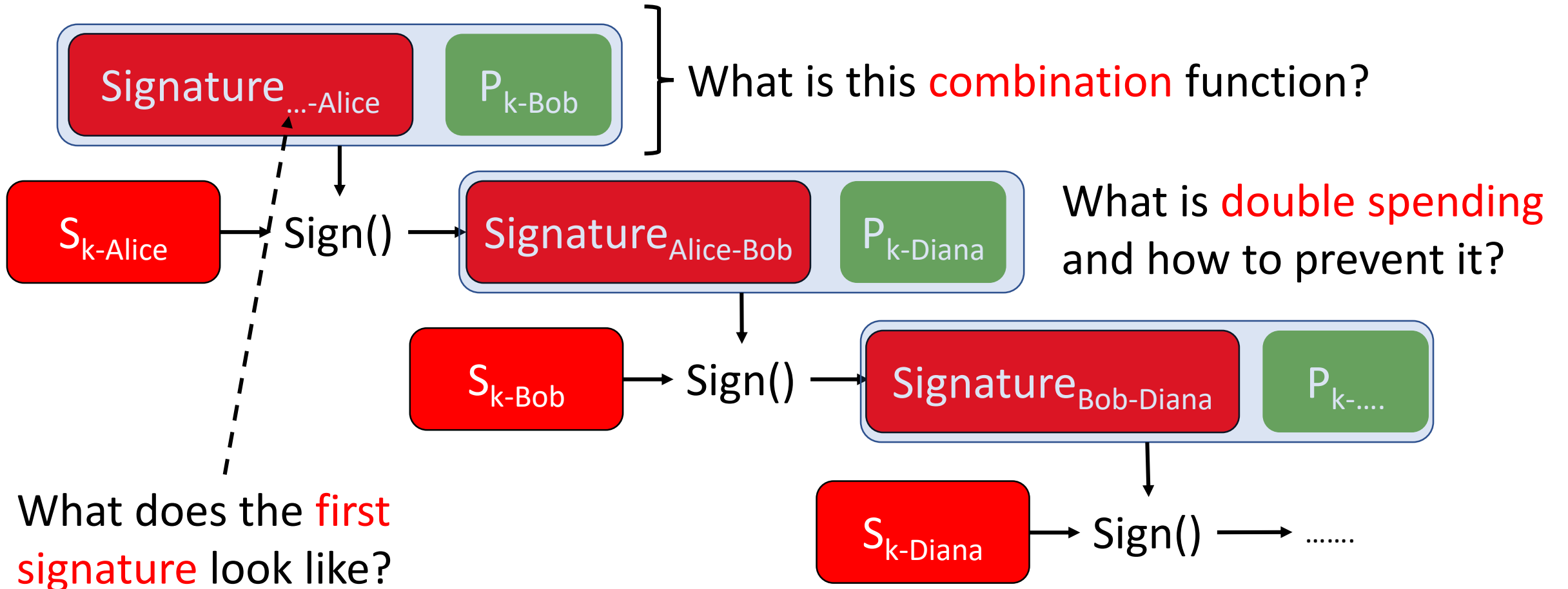
What About's?



What About's?



What About's?



Hashing $H(x)$

Signature_{Alice-Bob}

$P_{k-Diana}$

Hashing $H(x)$



- Signatures and public keys are combined using **Hashing**

Hashing $H(x)$

Signature_{Alice-Bob}

$P_{k-Diana}$

- 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 $H(x)$

Signature_{Alice-Bob}

$P_{k-Diana}$

- 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: $H(ID \parallel x) \in Y$

Bitcoin uses SHA-256

Signature_{Alice-Bob}

$P_{k-Diana}$

Bitcoin uses SHA-256

Signature_{Alice-Bob}

P_{k-Diana}

SHA256(Signature_{Alice-Bob} || P_{k-Diana}) =
256-bit (32-byte) unique string

Bitcoin uses SHA-256

Signature_{Alice-Bob}

P_{k-Diana}

SHA256(Signature_{Alice-Bob} || P_{k-Diana}) =

256-bit (32-byte) unique string

Bitcoin uses SHA-256

Signature_{Alice-Bob}

P_{k-Diana}

SHA256(Signature_{Alice-Bob} || P_{k-Diana}) =

256-bit (32-byte) unique string

SHA256(abc) =

ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad

Bitcoin uses SHA-256

Signature_{Alice-Bob}

P_{k-Diana}

SHA256(Signature_{Alice-Bob} || P_{k-Diana}) =

256-bit (32-byte) unique string

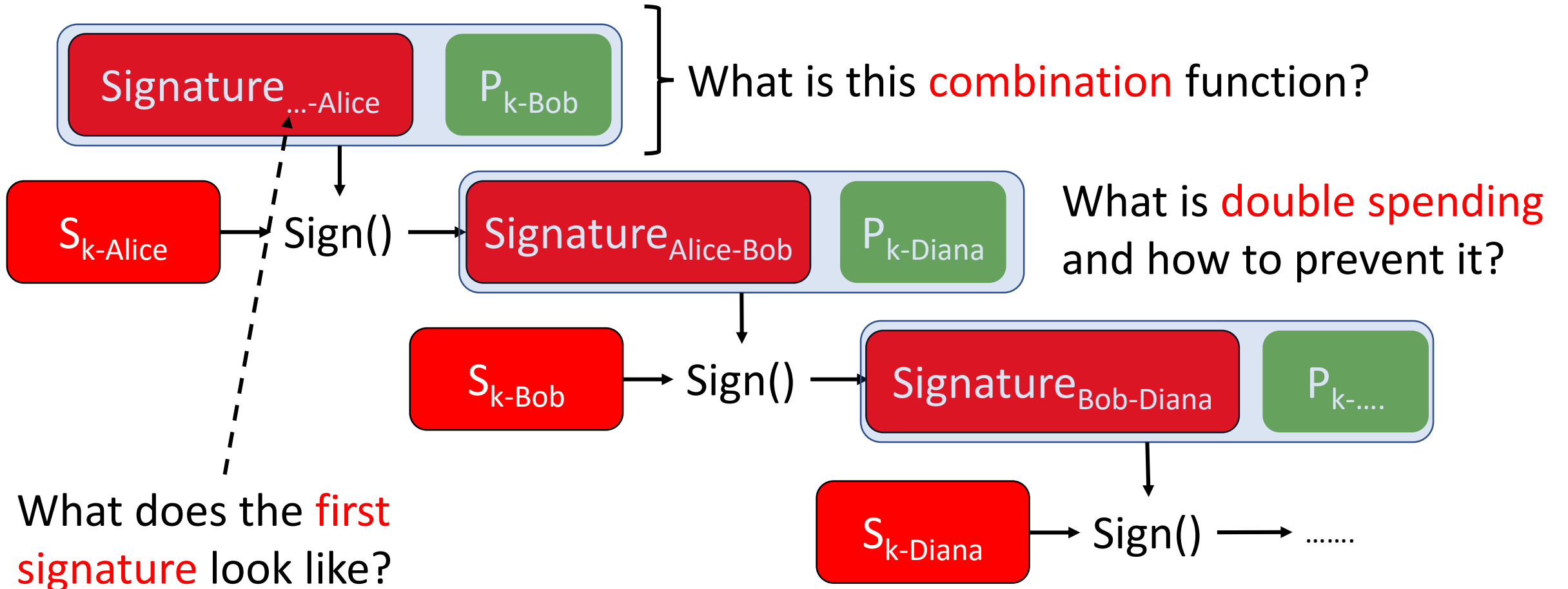
SHA256(abc) =

ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad

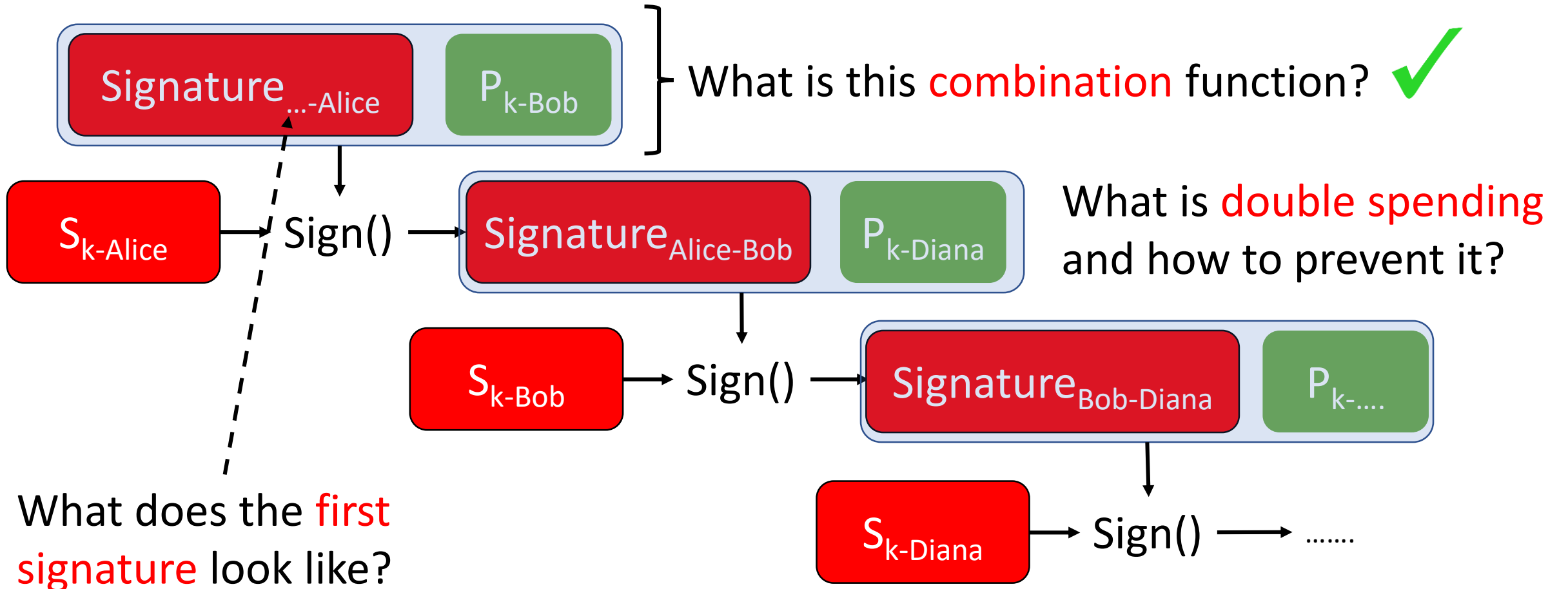
SHA256(abC) =

0a2432a1e349d8fdb9bfca91bba9e9f2836990fe937193d84deef26c6f3b8f76

What About's?



What About's?



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_{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}

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**

Signature_{Alice-Bob}

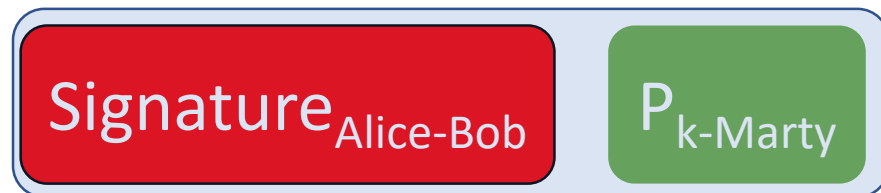
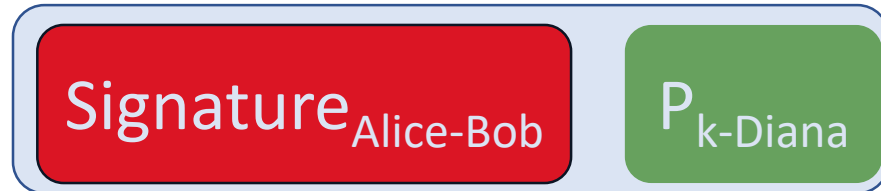
$P_{k\text{-Diana}}$

Signature_{Alice-Bob}

$P_{k\text{-Marty}}$

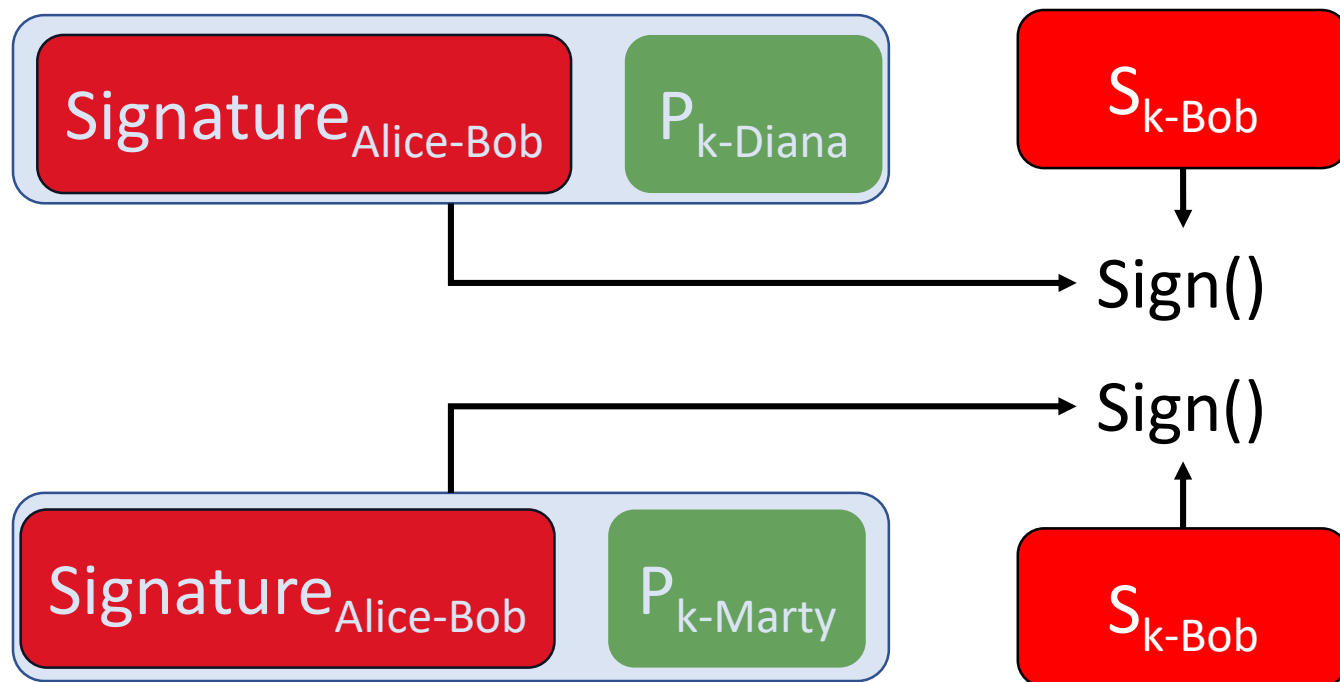
Double Spending

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



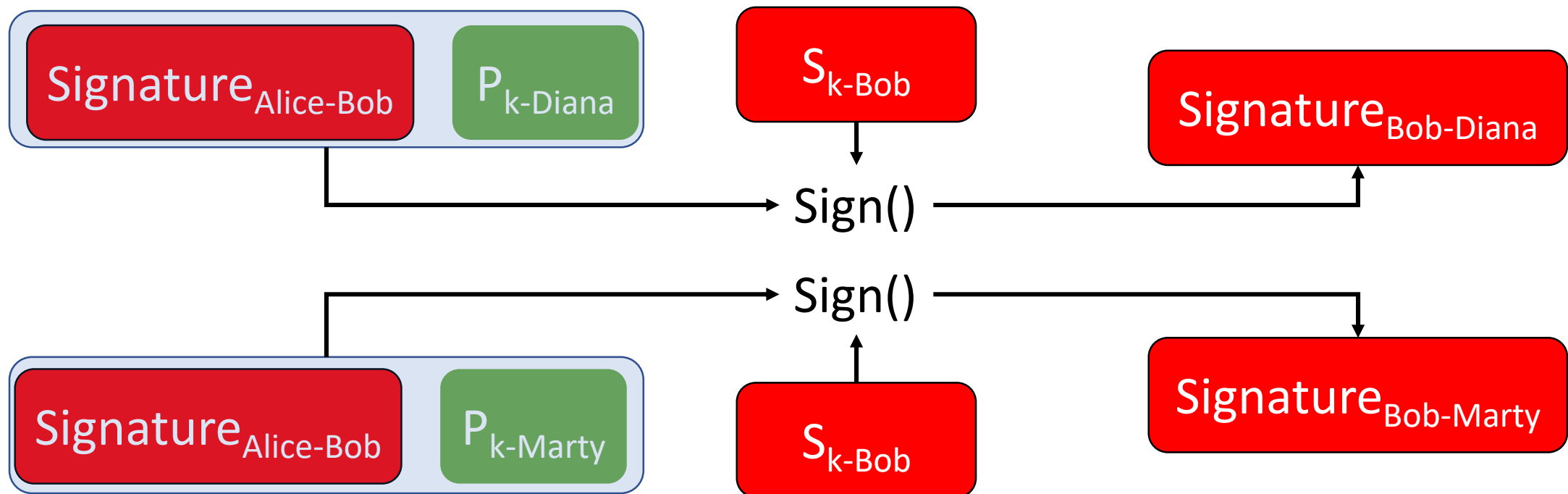
Double Spending

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



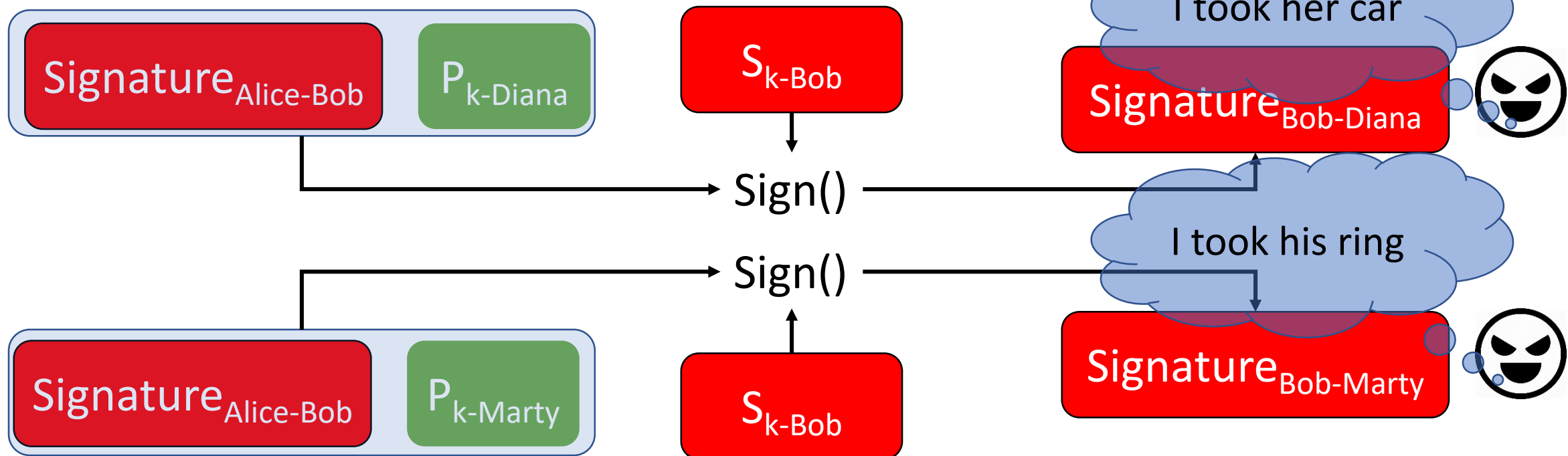
Double Spending

- 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 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)



50 BTC

Signature_{Trent-Bob}

Double Spending Prevention

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



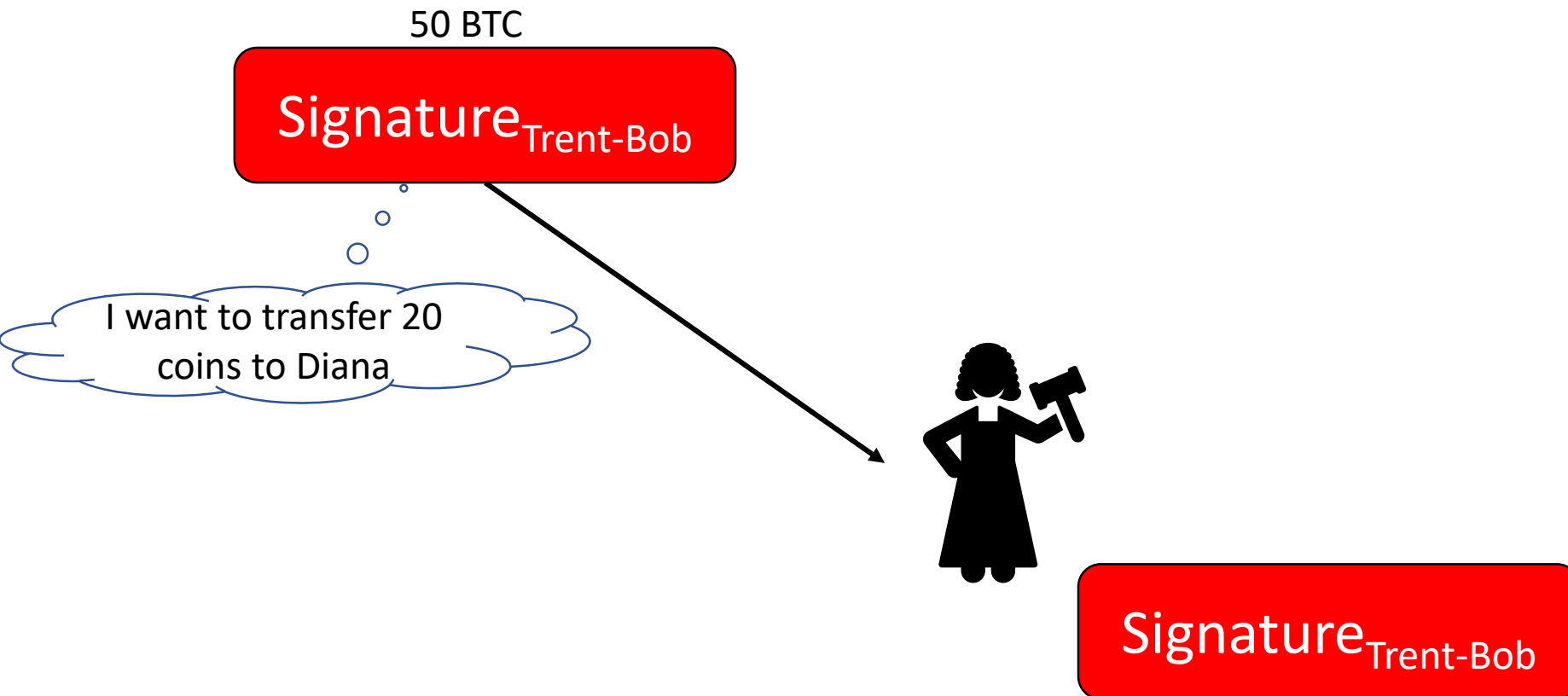
50 BTC

Signature_{Trent-Bob}

I want to transfer 20
coins to Diana

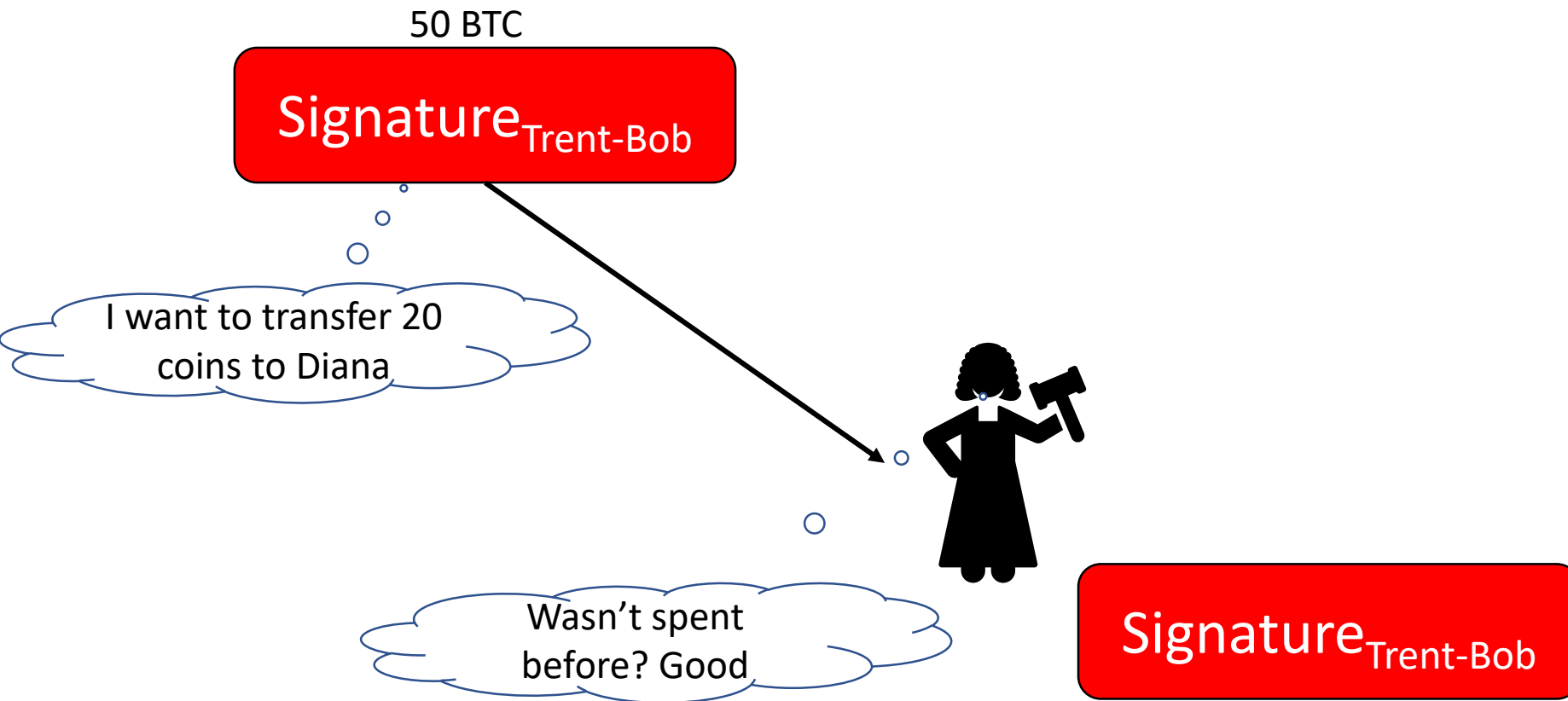
Double Spending Prevention

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



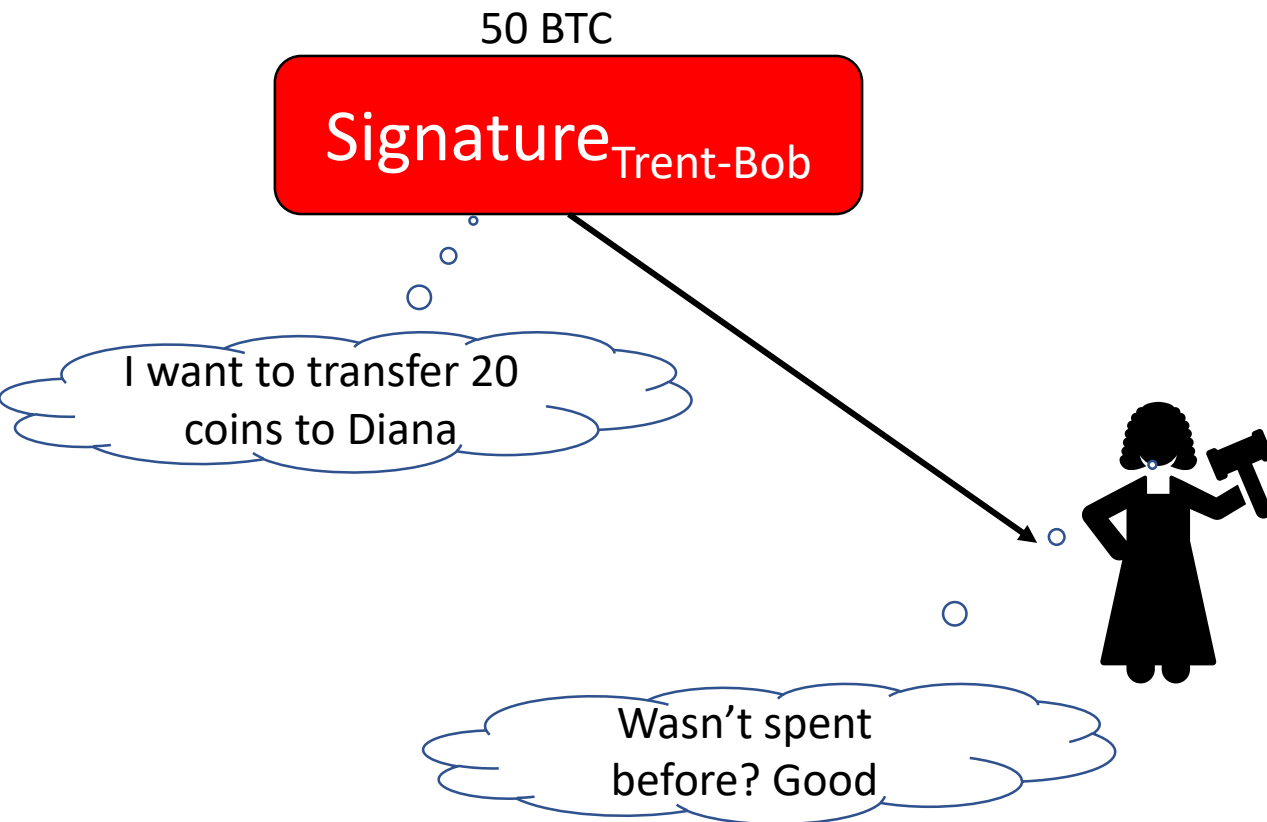
Double Spending Prevention

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



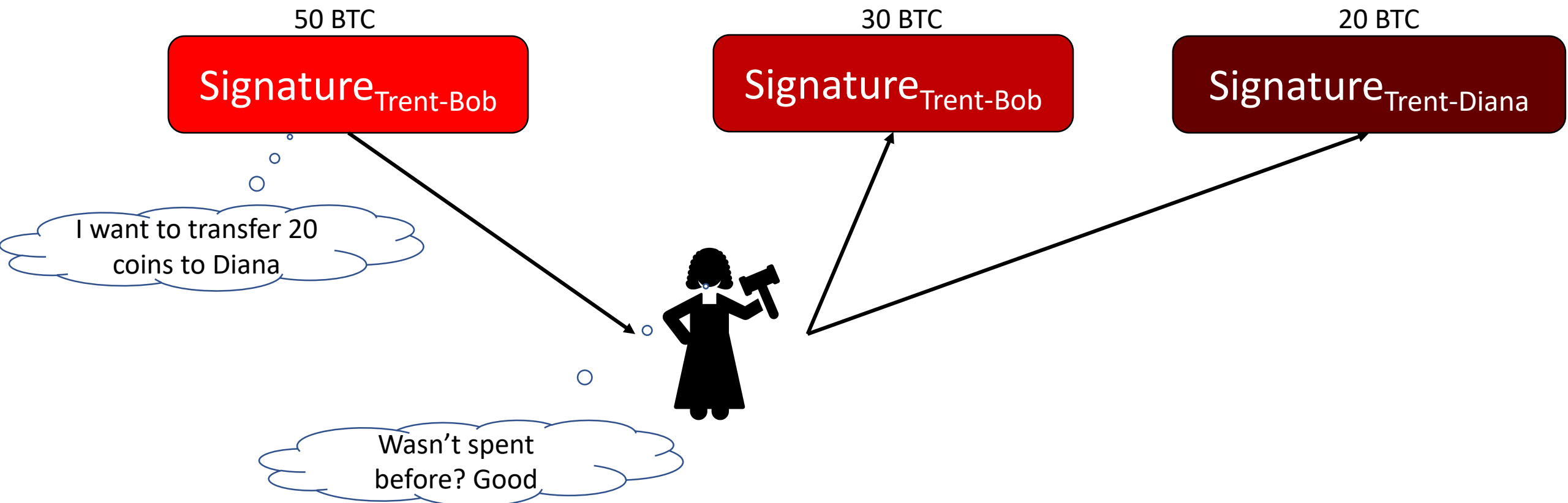
Double Spending Prevention

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



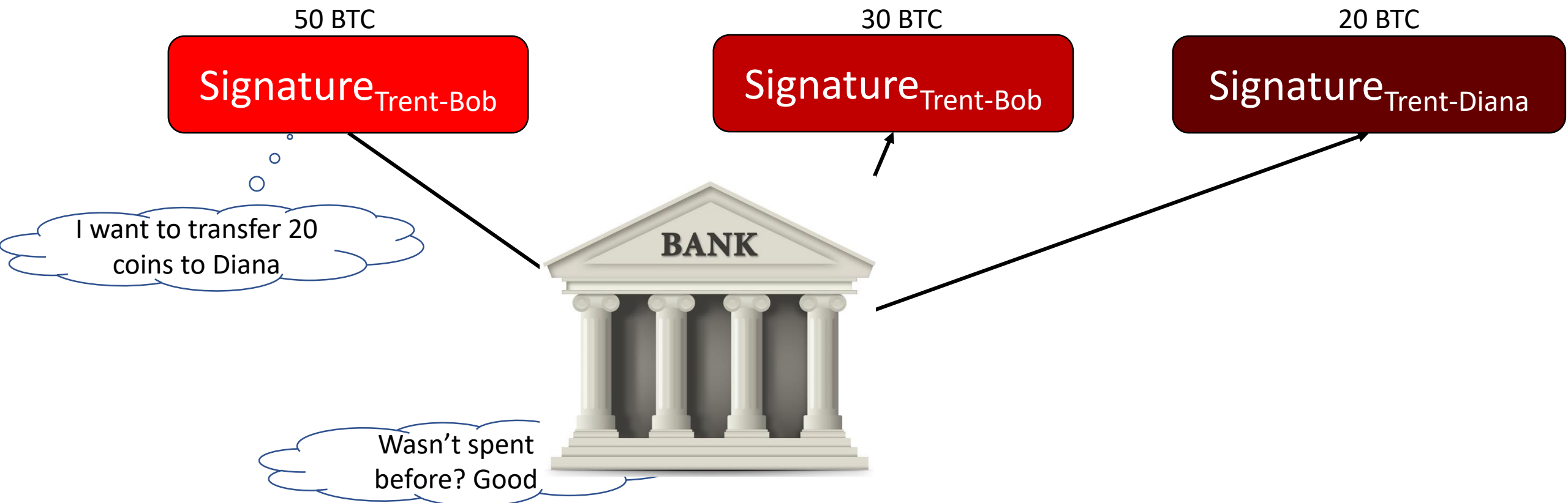
Double Spending Prevention

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



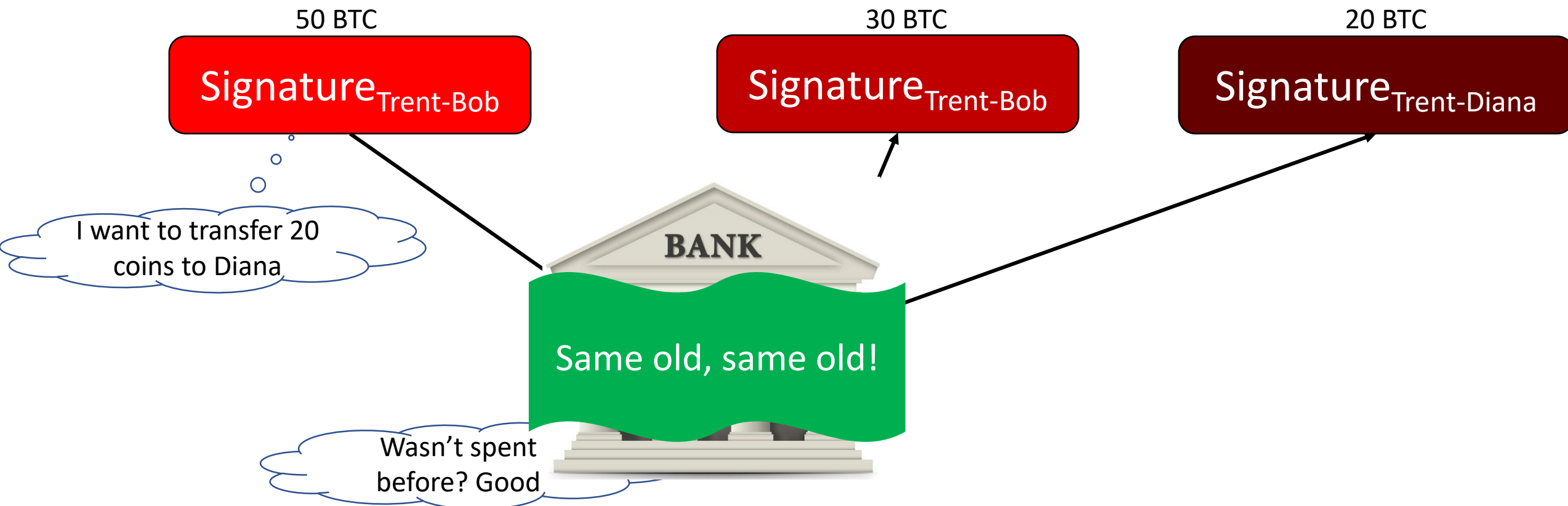
Double Spending Prevention

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



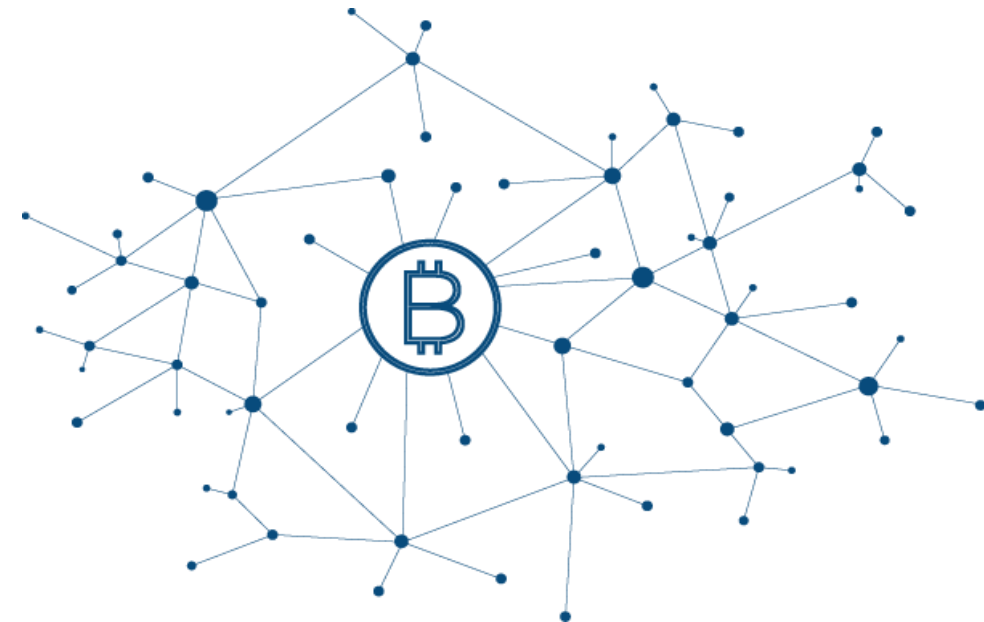
Double Spending Prevention

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



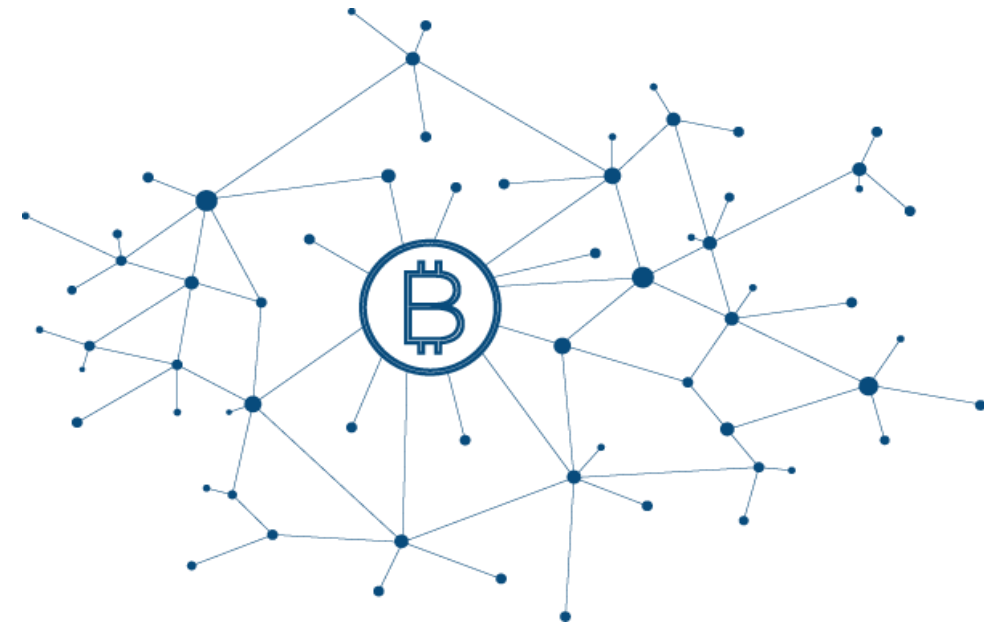
Double Spending Prevention

- Decentralized



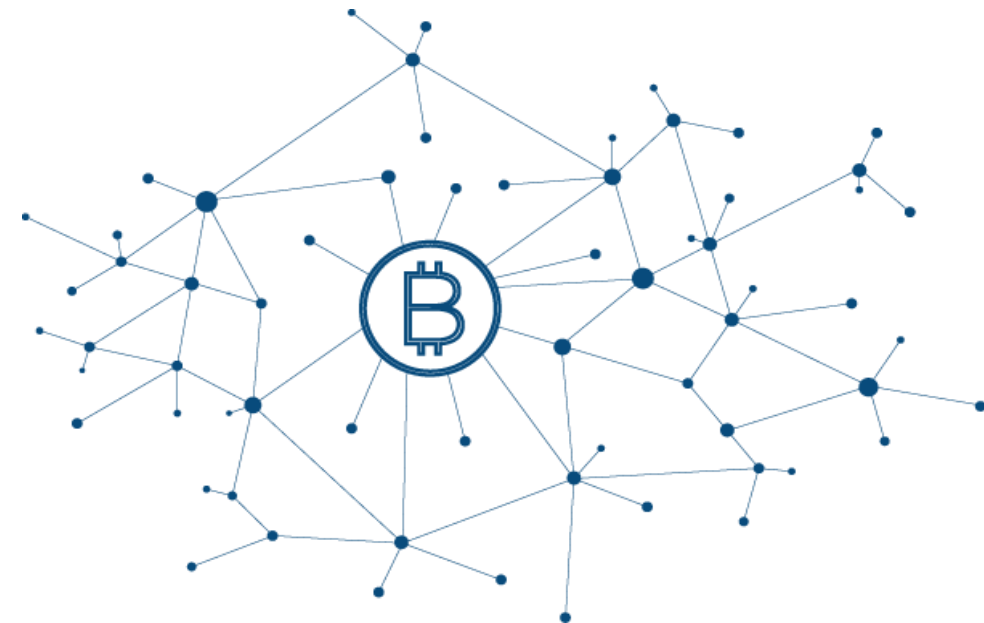
Double Spending Prevention

- Decentralized
 - A network of nodes maintains a ledger



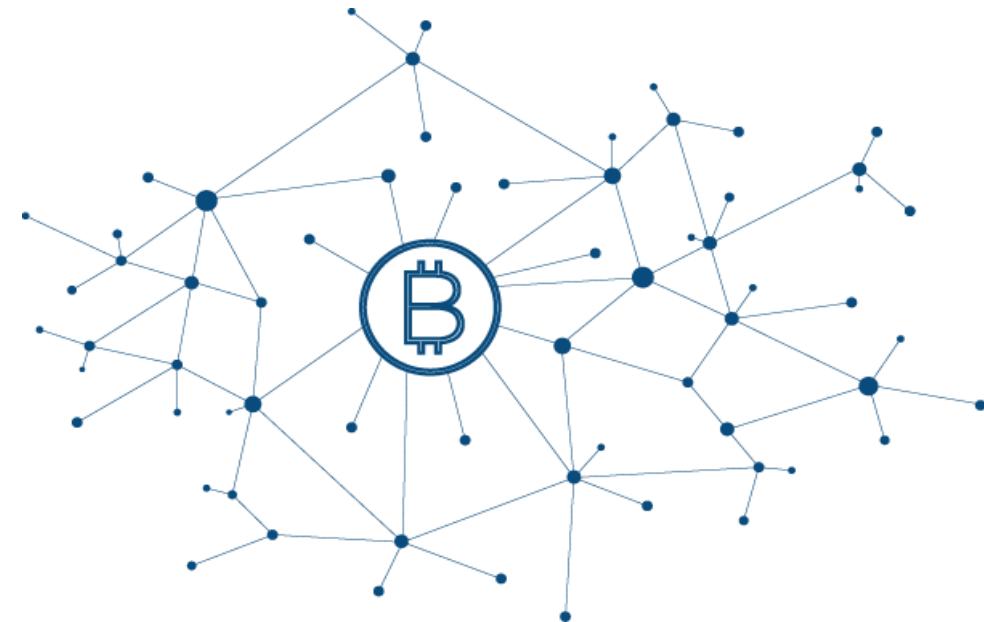
Double Spending Prevention

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



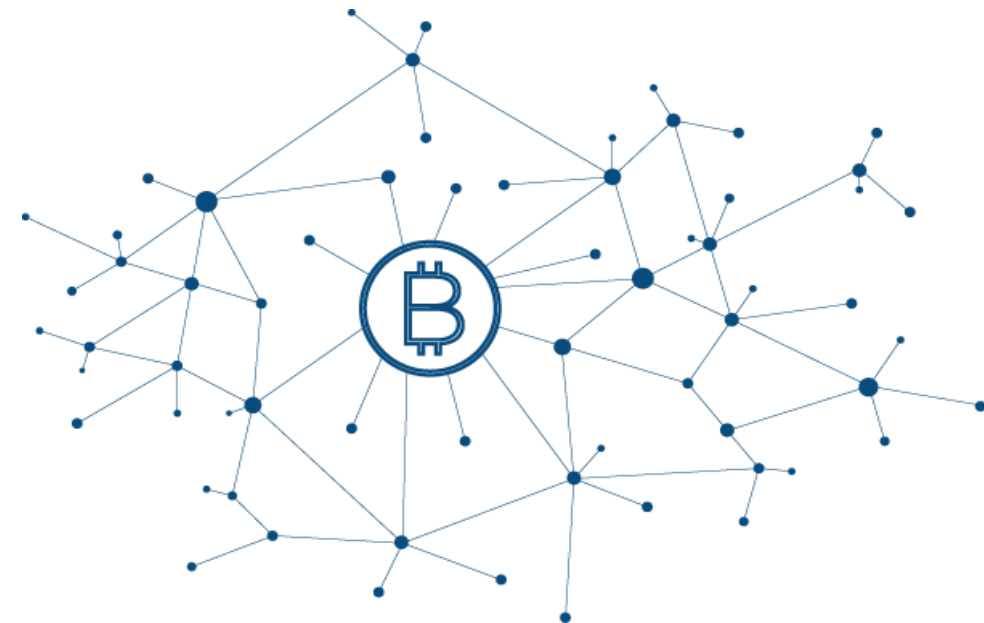
Double Spending Prevention

- 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?**



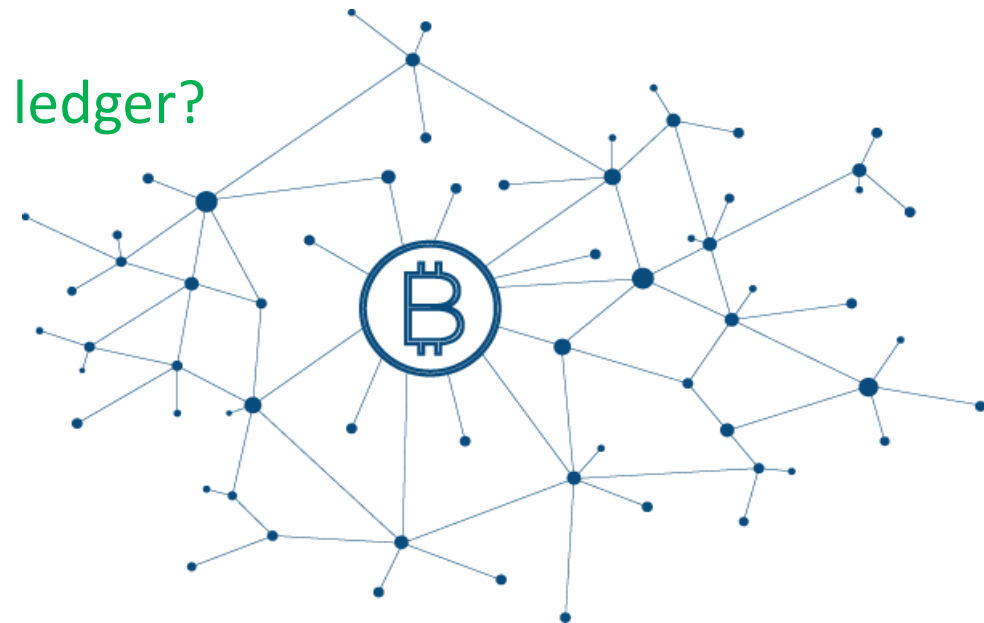
Double Spending Prevention

- 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



What is the Ledger?

- Blockchain



- Transactions are grouped into blocks

What is the Ledger?

- Blockchain



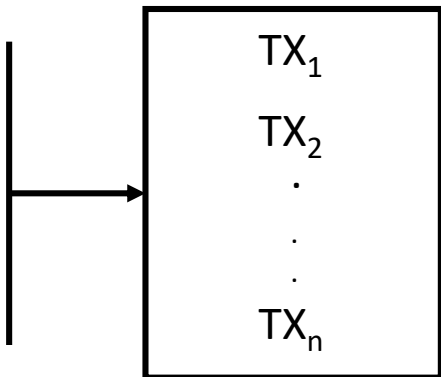
- Transactions are grouped into blocks
 - Blocks are chained to each other through pointers (Hence blockchain)

What is the Ledger?

- Blockchain



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

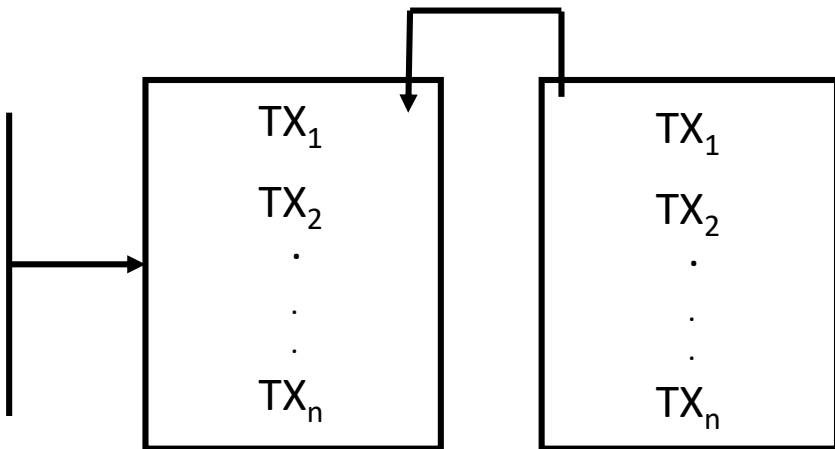


What is the Ledger?

- Blockchain



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

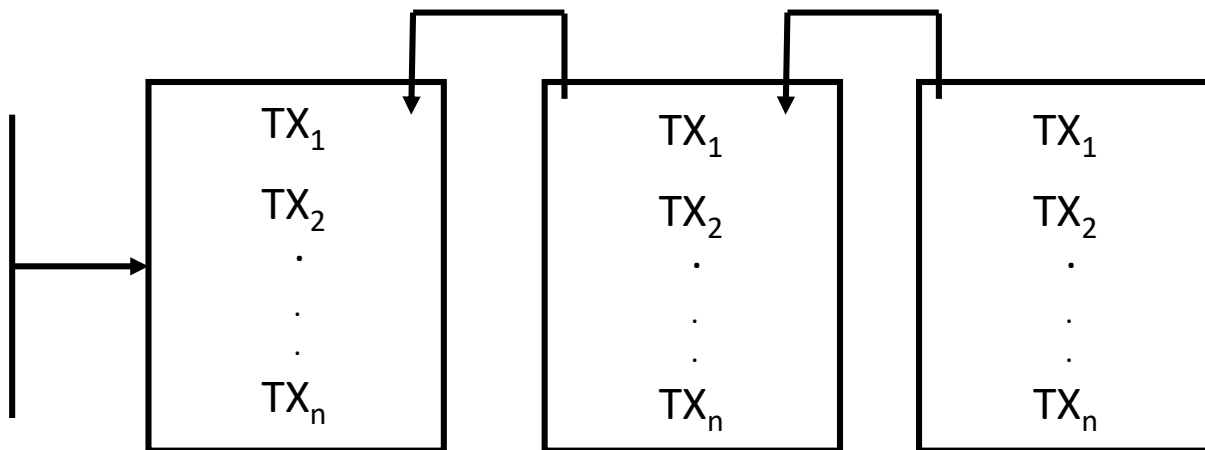


What is the Ledger?

- Blockchain



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

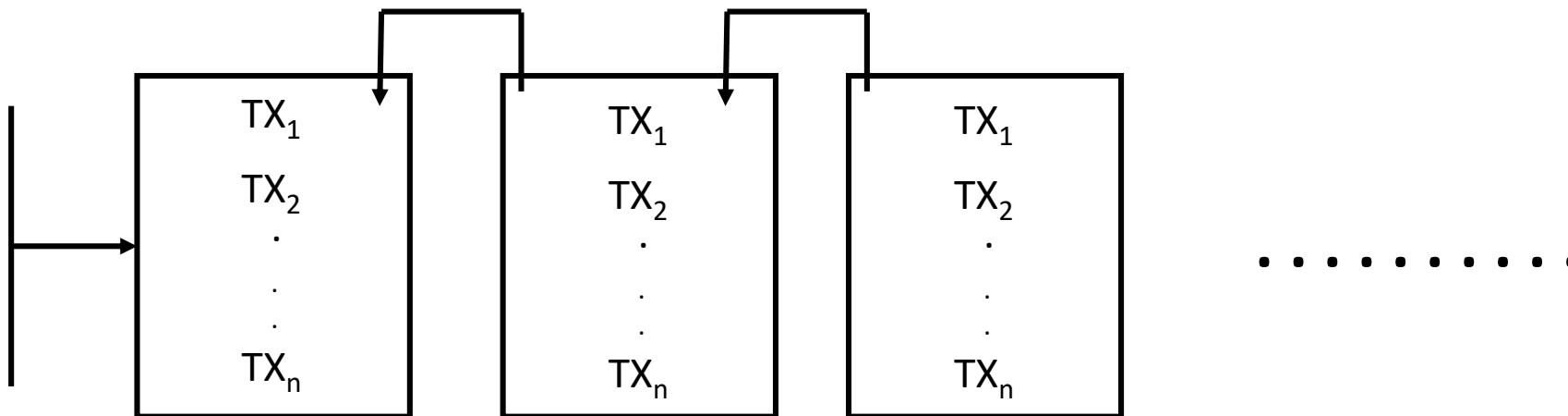


What is the Ledger?

- Blockchain



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

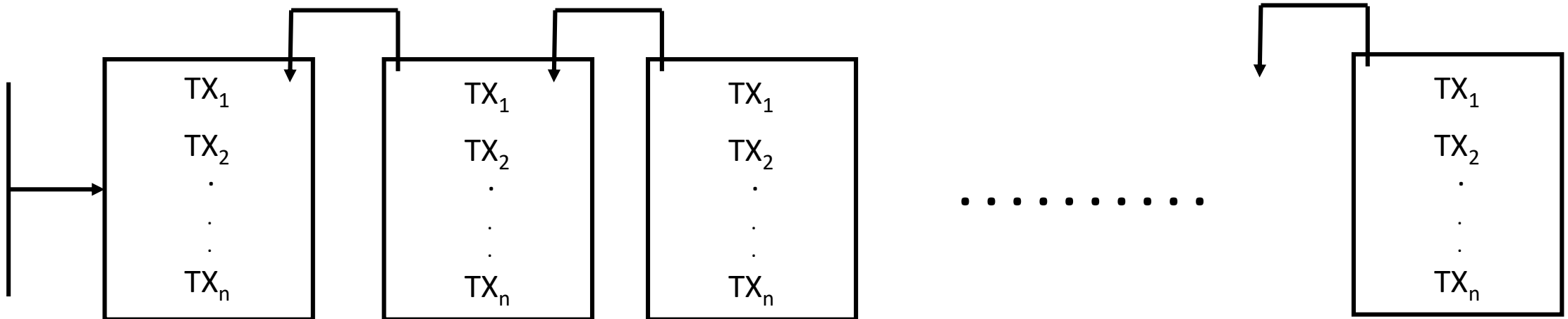


What is the Ledger?

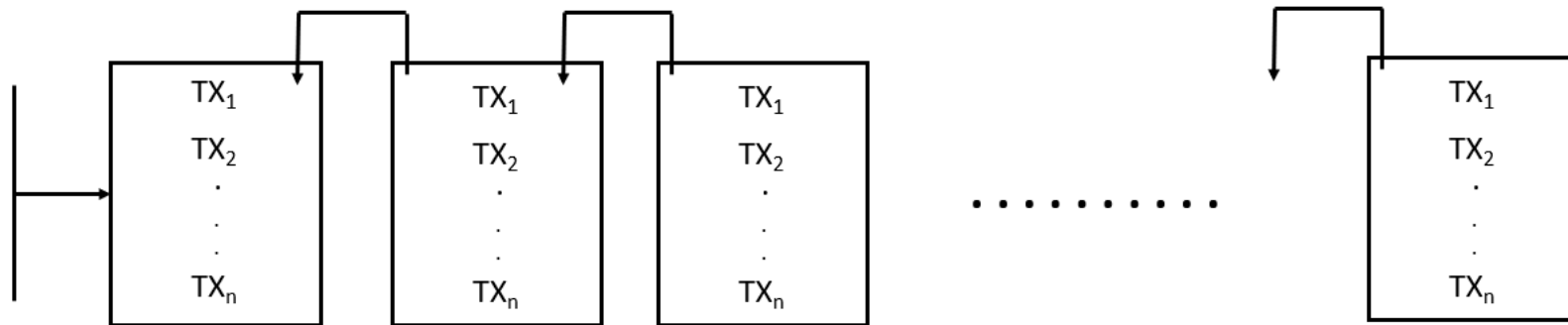
- Blockchain



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

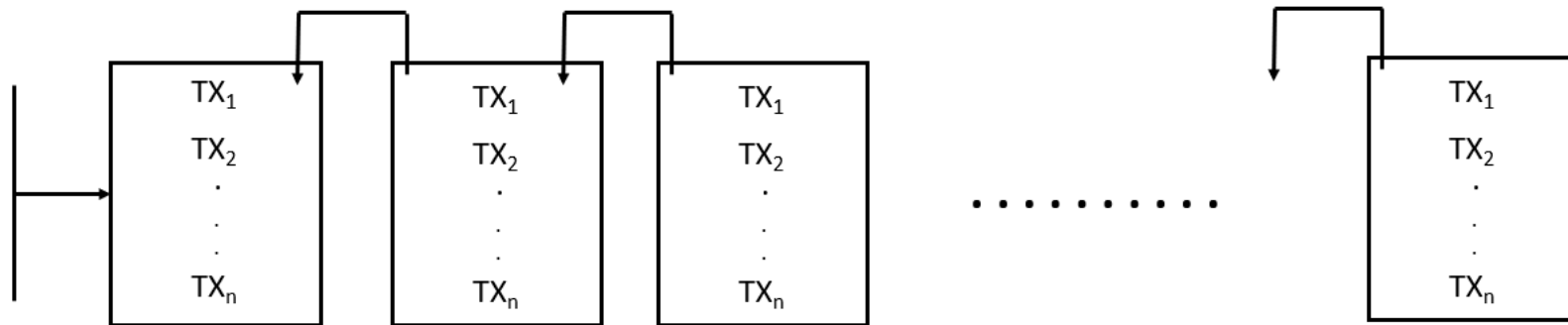


The Ledger's What About's?



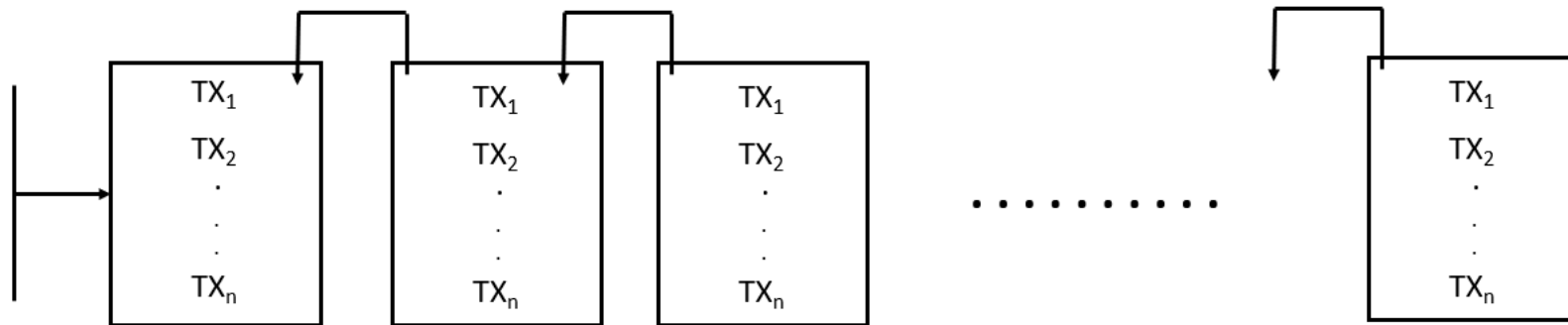
The Ledger's What About's?

- Where is the ledger stored?



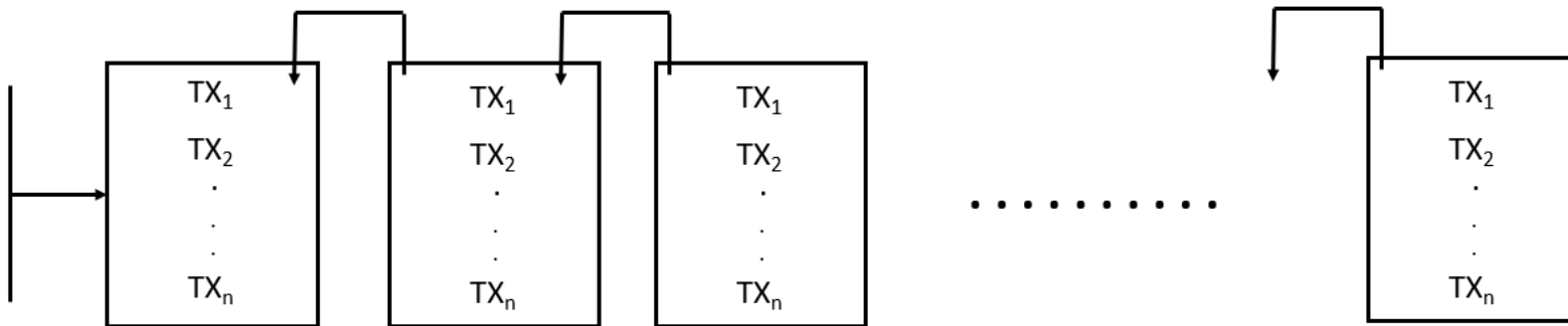
The Ledger's What About's?

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



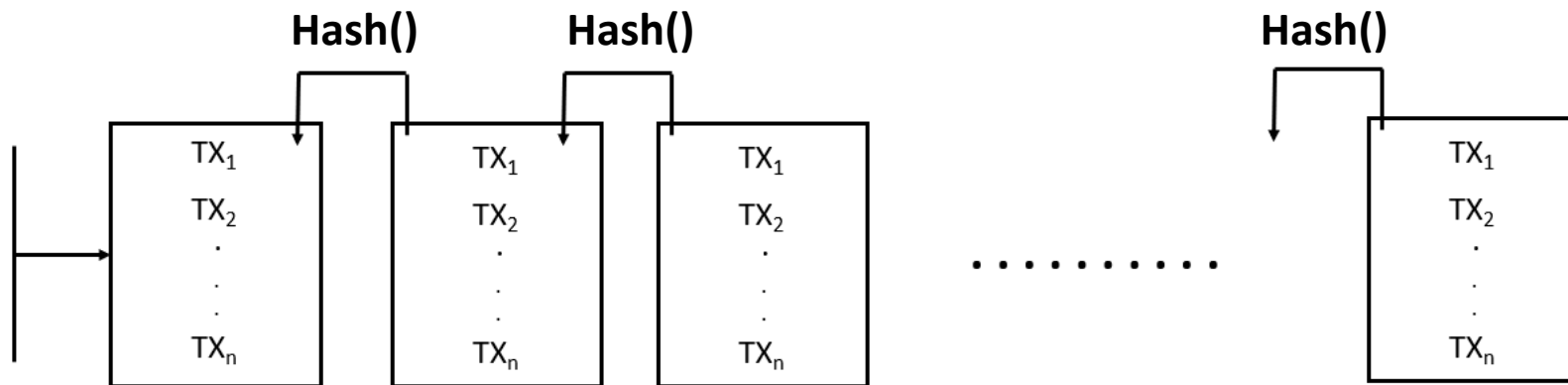
The Ledger's What About's?

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



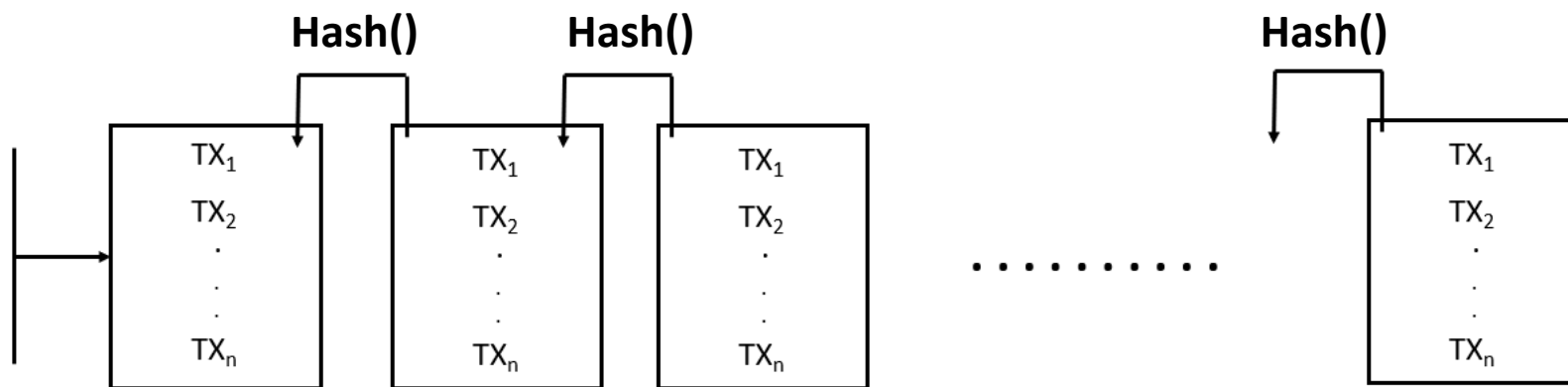
The Ledger's What About's?

- 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**

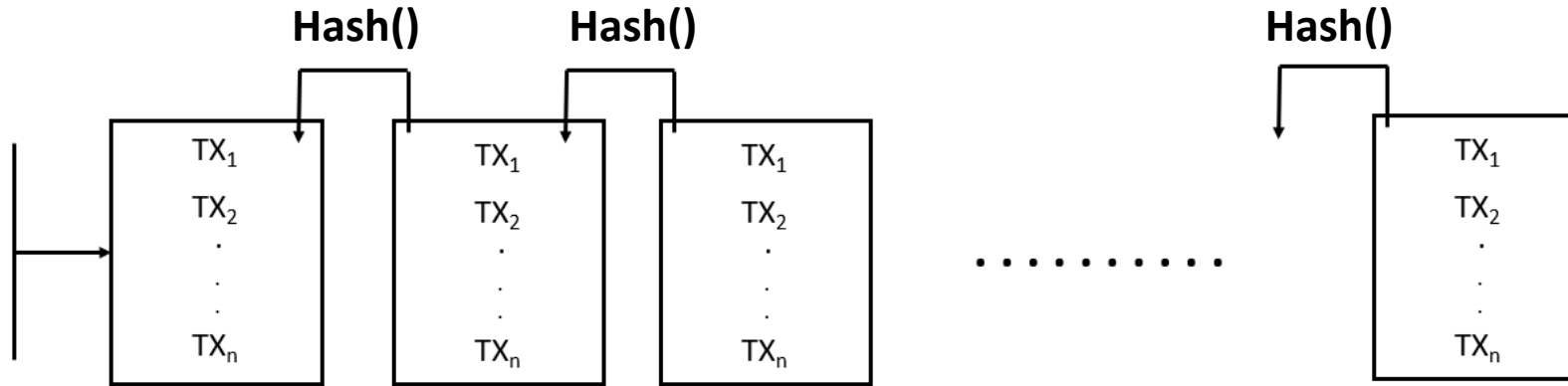


The Ledger's What About's?

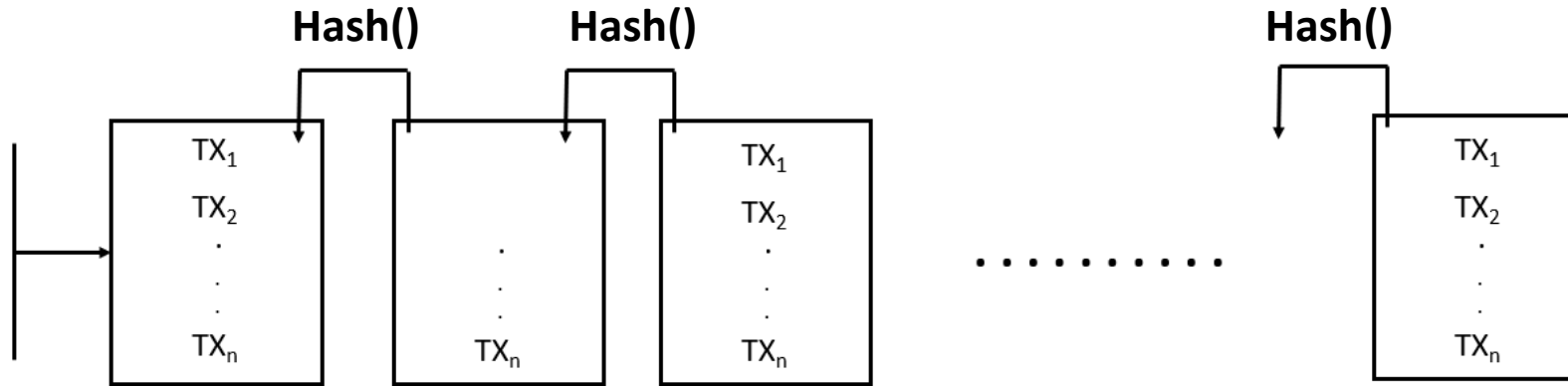
- Where is the ledger stored?
 - Each network node maintains its copy of the ledger
- How is the ledger tamper-free?
 1. Blocks are connected through **hash-pointers**
 - Each block contains the hash of the previous block
 - This hash gives each block its location in the blockchain
 - Tampering with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



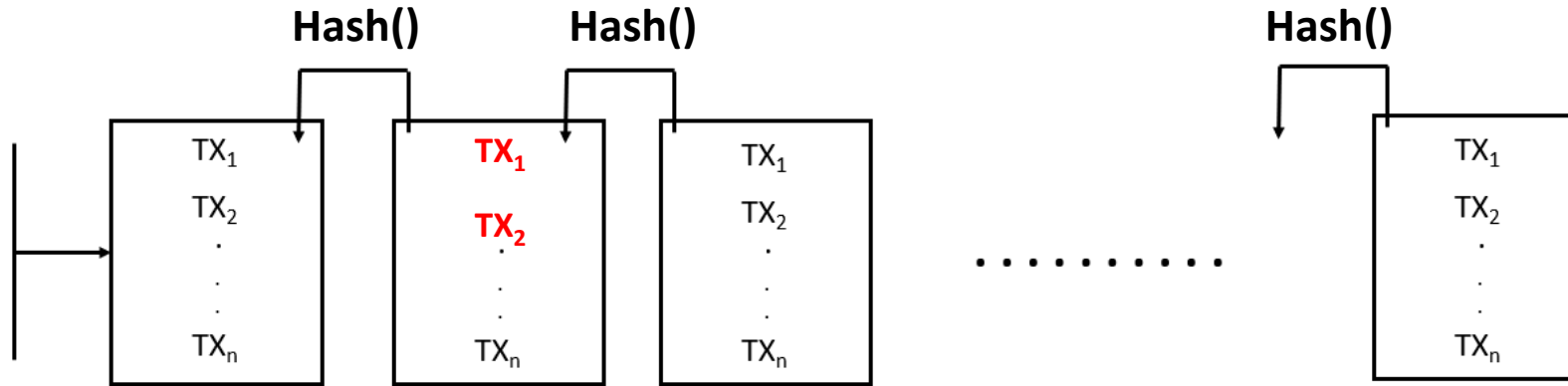
Tampering with the Ledger



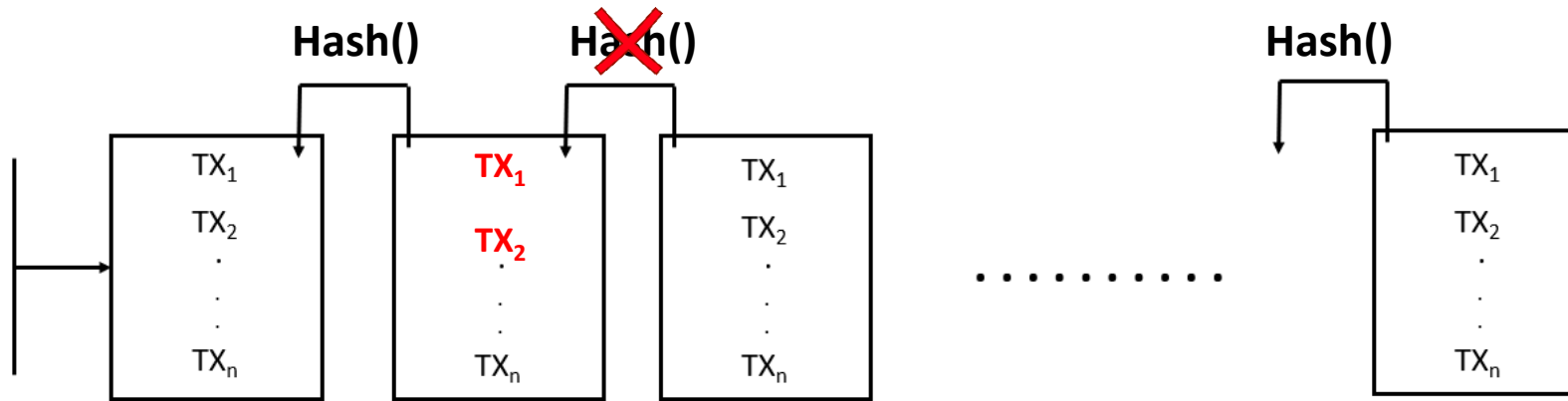
Tampering with the Ledger



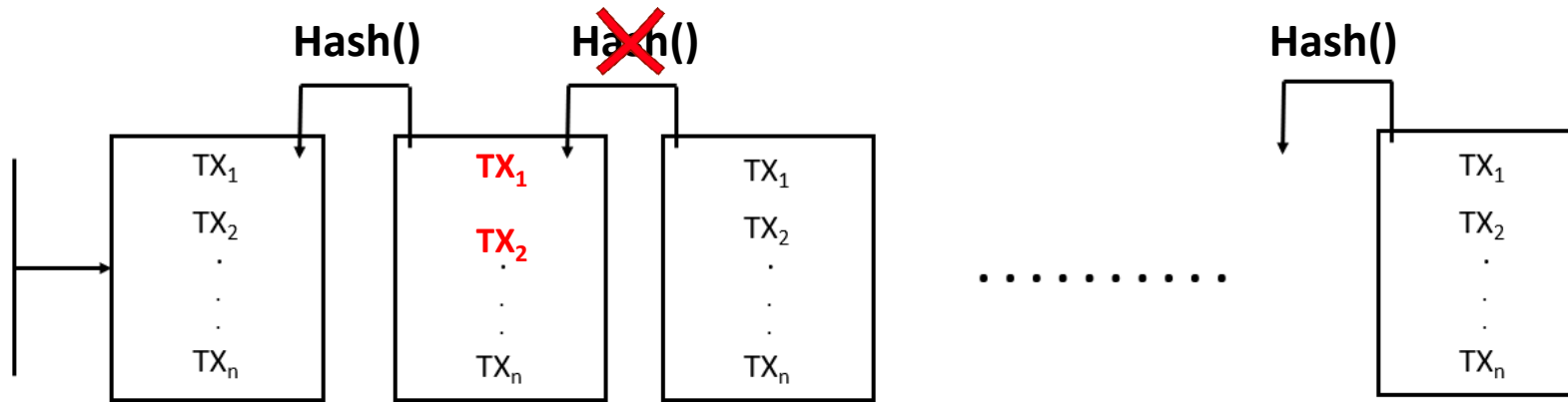
Tampering with the Ledger



Tampering with the Ledger

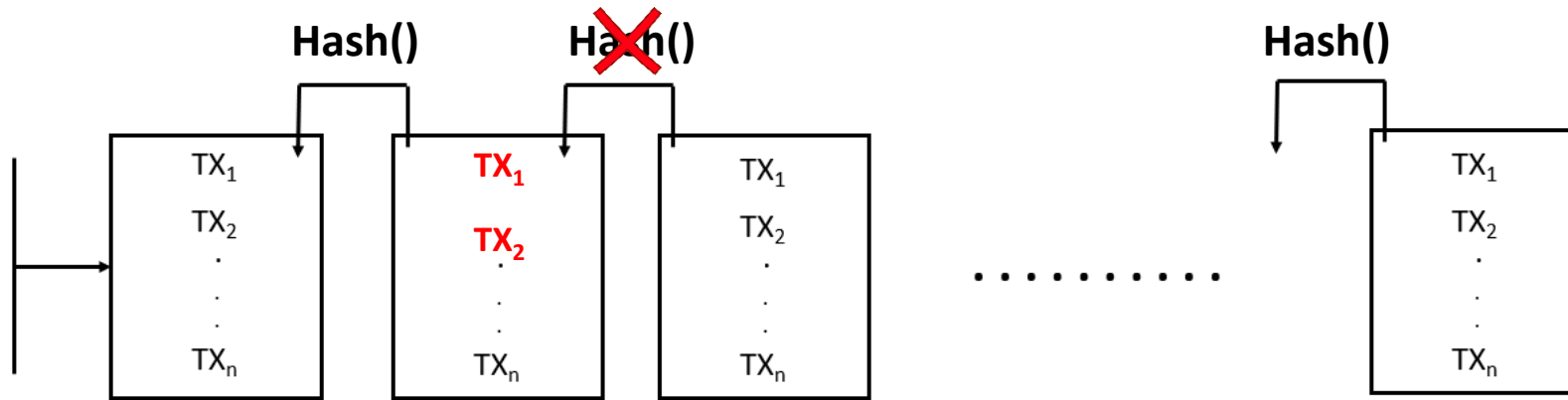


Tampering with the Ledger



Inconsistent Blockchain

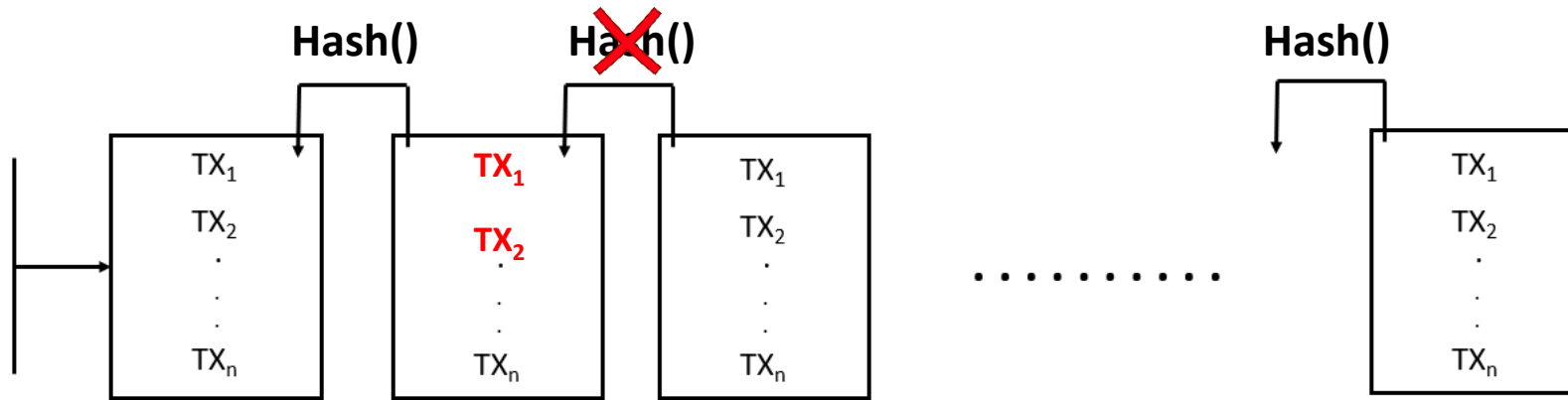
Tampering with the Ledger



Inconsistent Blockchain

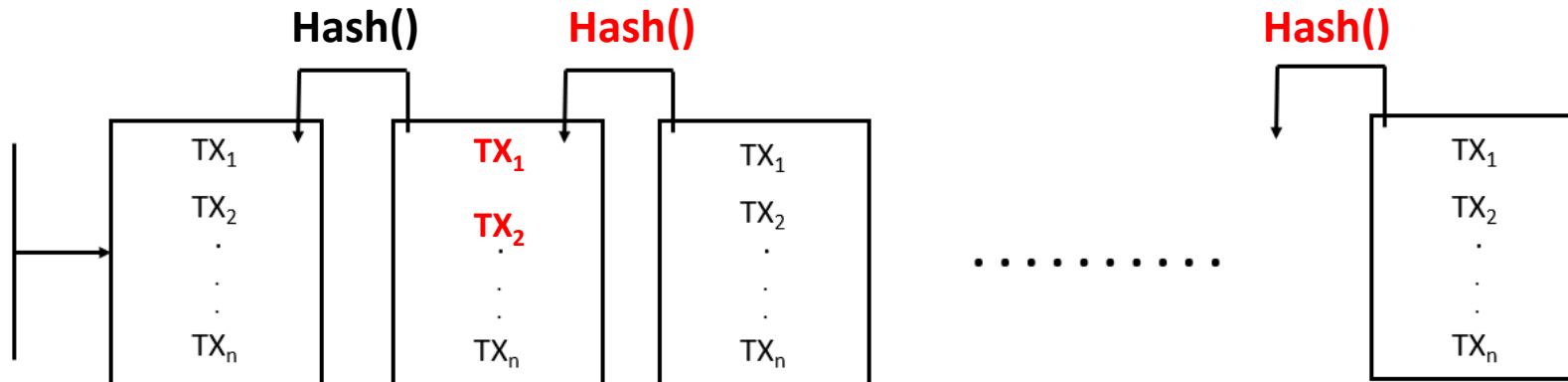
However,

Tampering with the Ledger

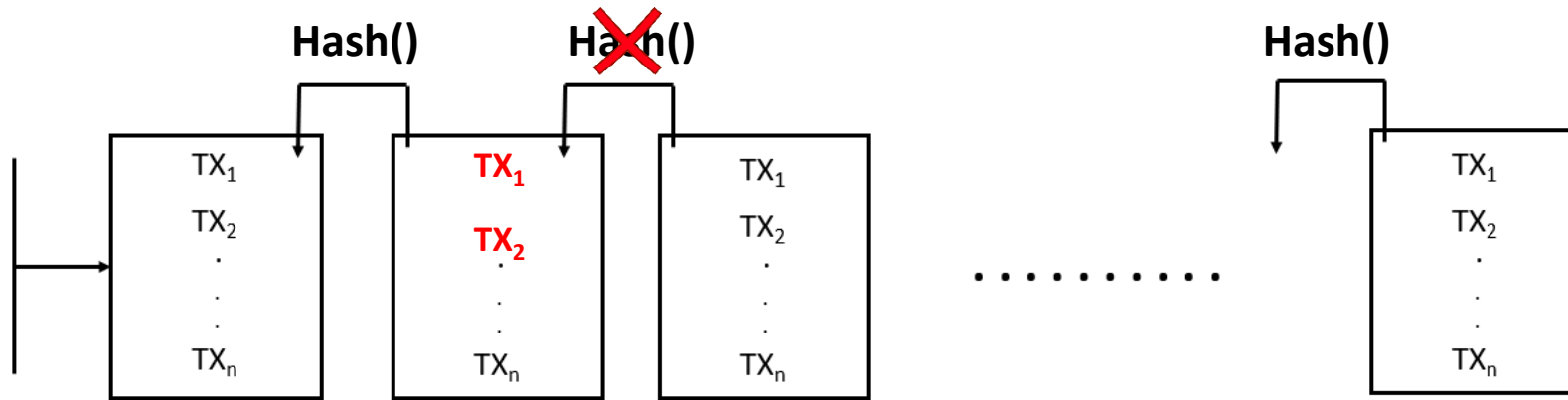


Inconsistent Blockchain

However,

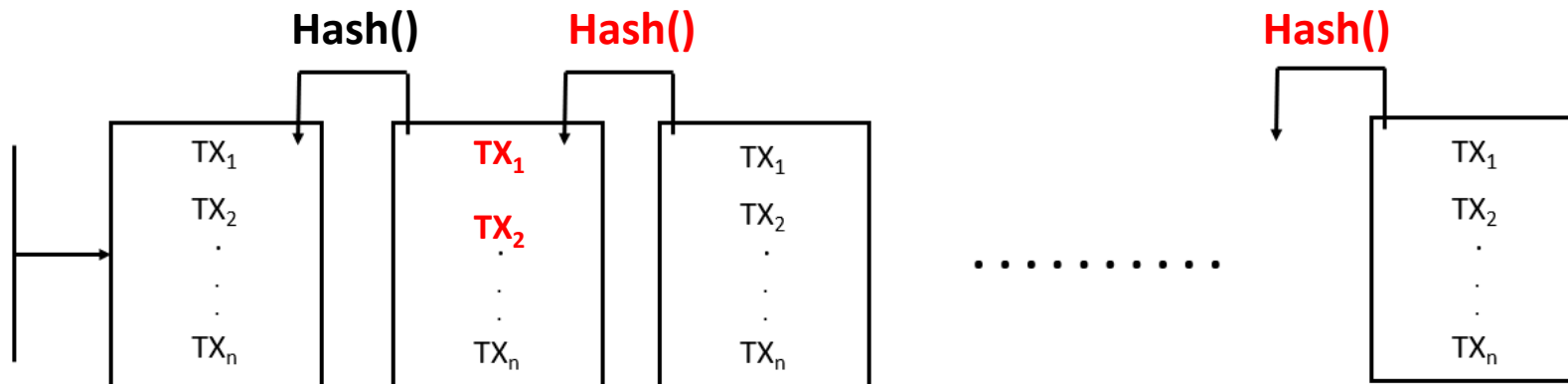


Tampering with the Ledger



Inconsistent Blockchain

However,



Consistent Blockchain

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**)

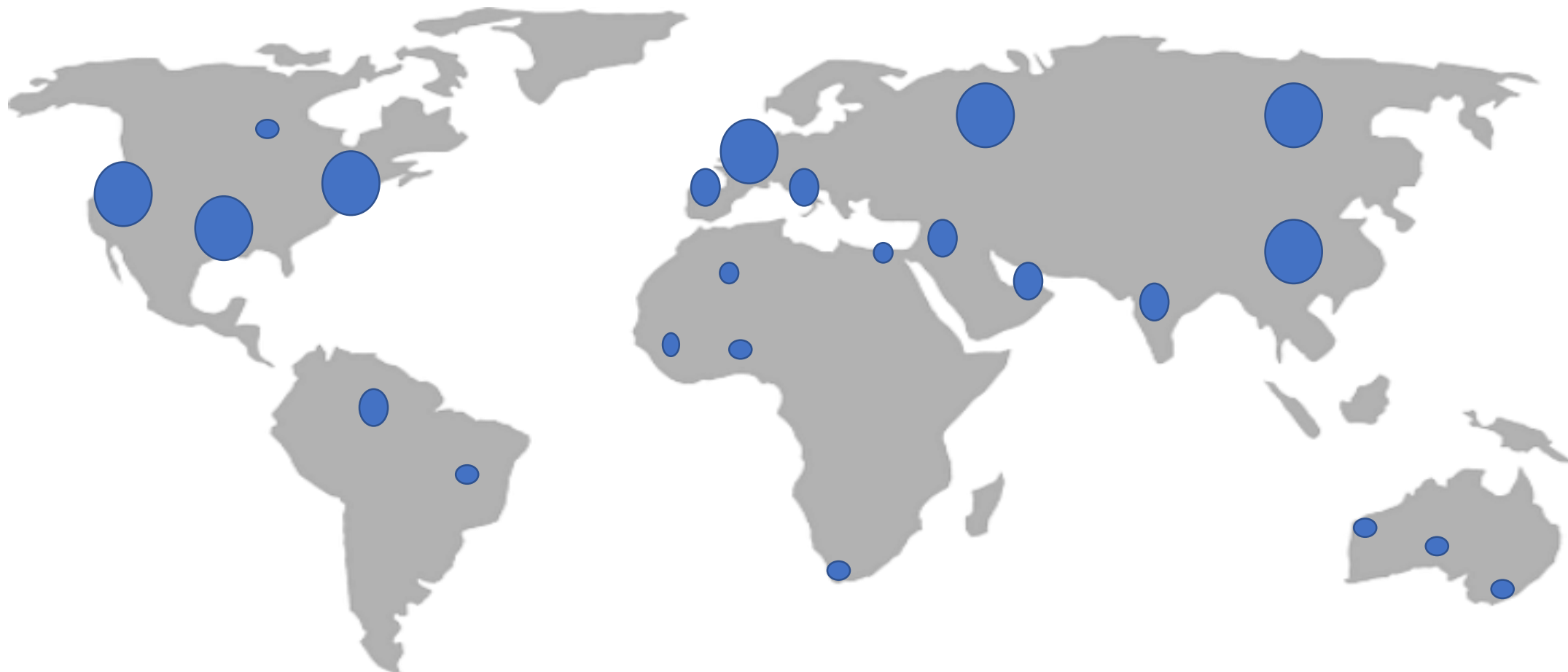
The Ledger's What About's?

- 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?

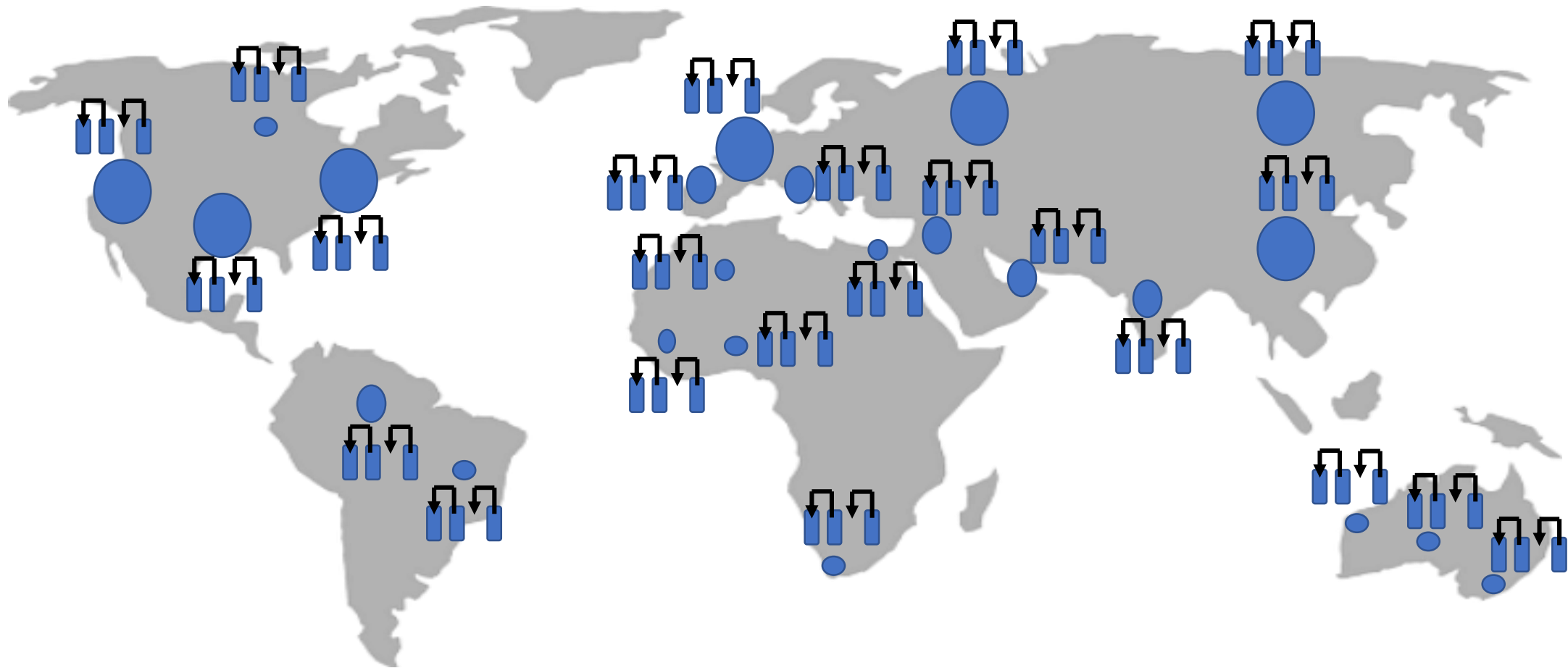
Network Nodes Big Picture



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:

A 

Majority 

(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**



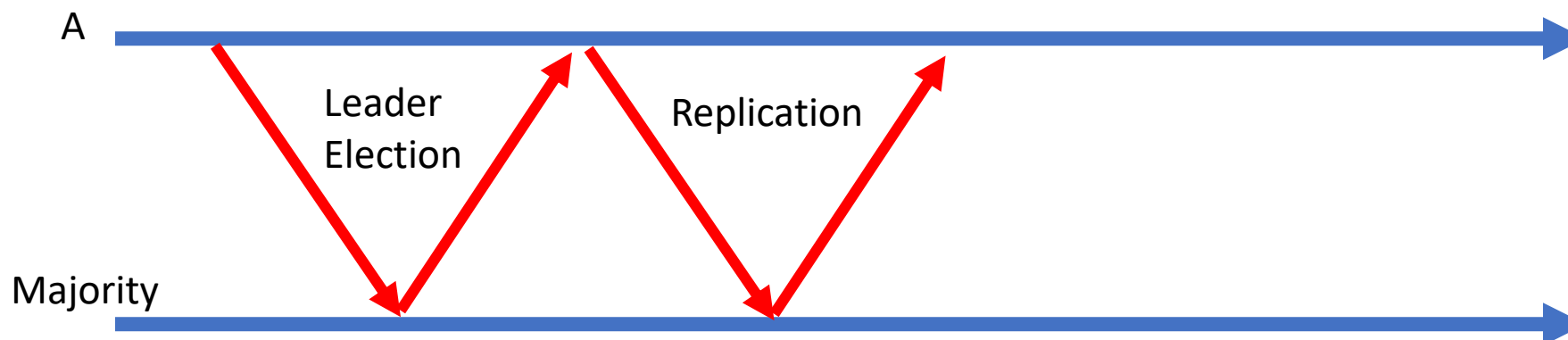
(Multi-) Paxos

- 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**



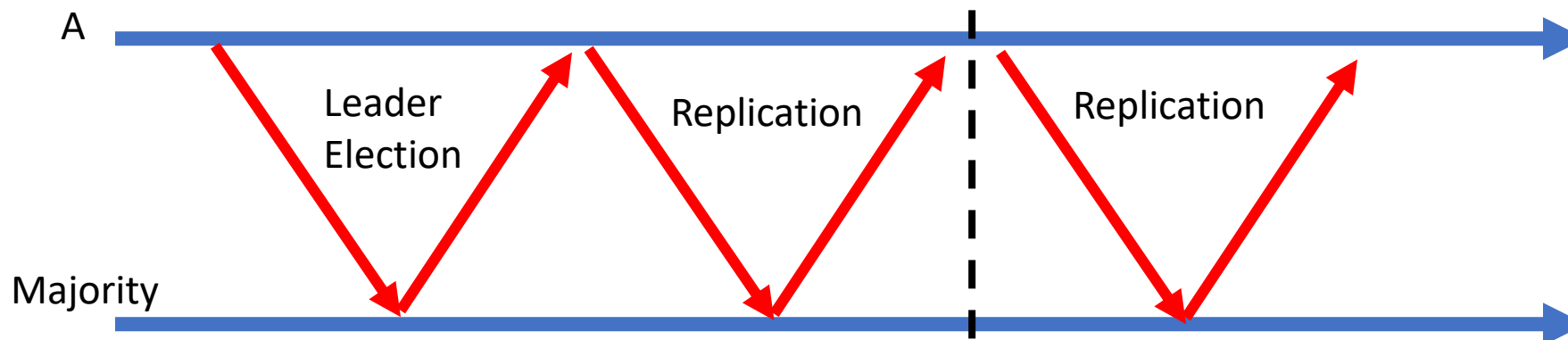
(Multi-) Paxos

- 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**



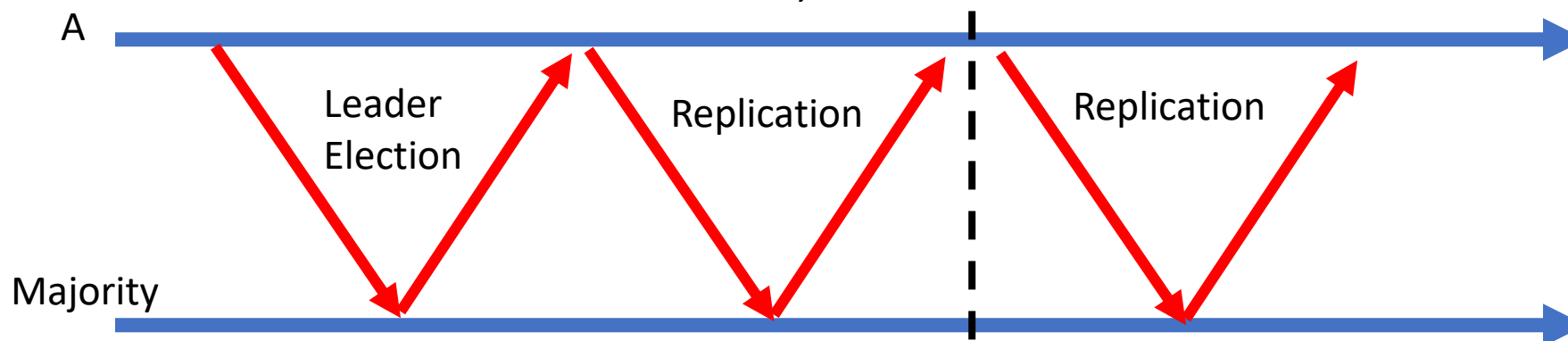
(Multi-) Paxos

- 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**

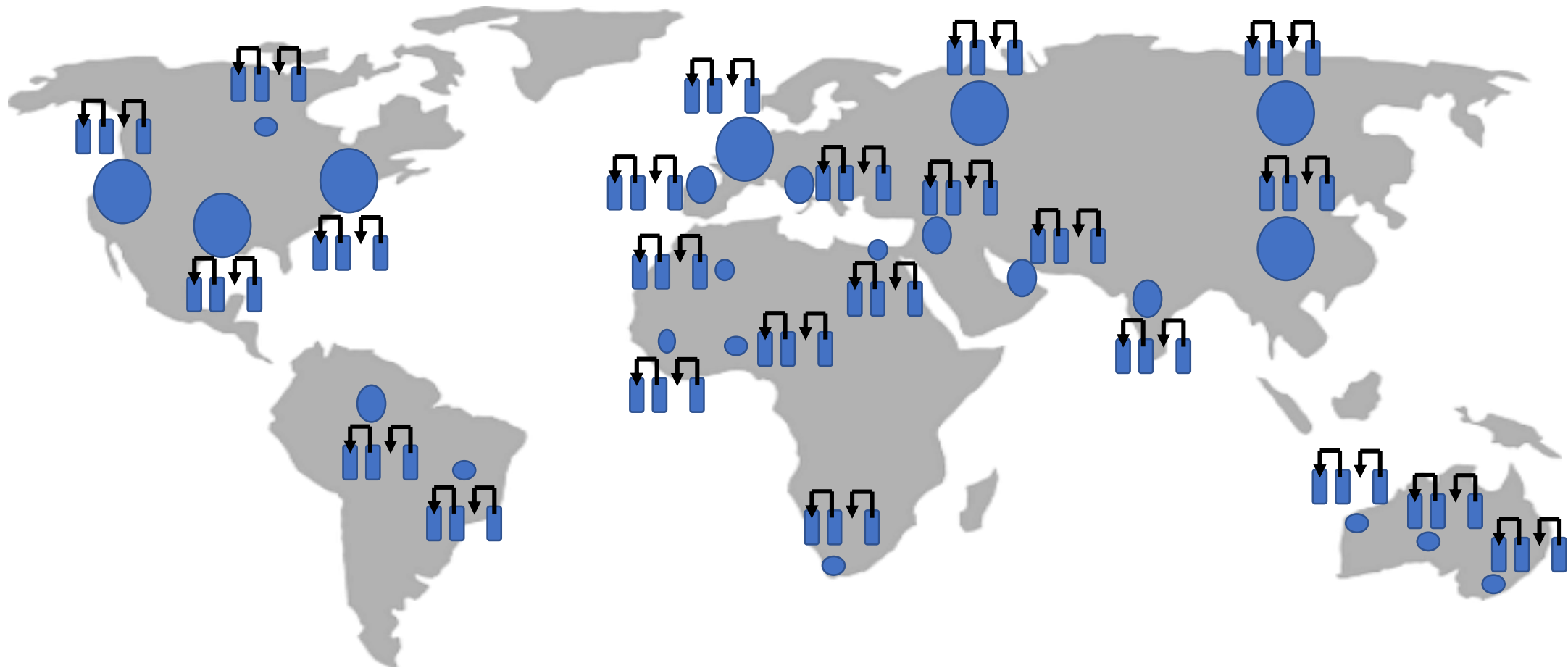


(Multi-) Paxos

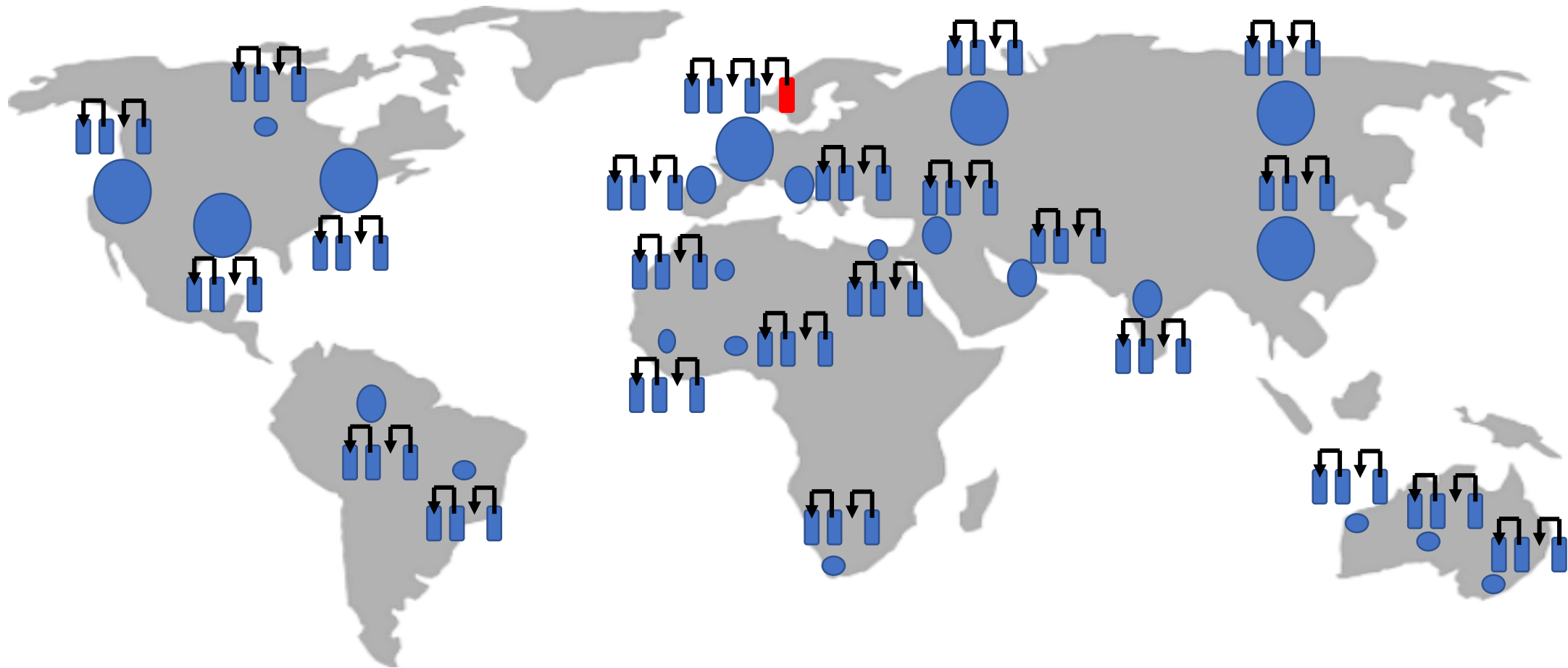
- Paxos is a consensus algorithm
 - Processes want to agree on a value (e.g., the next block to be added to the chain)
- Paxos is currently used to manage local data in global-scale systems
 - Spanner [OSDI'12, SIGMOD'17], Megastore [CIDR'11], etc
- Multi-Paxos, simplified:
 - Initially, a leader is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



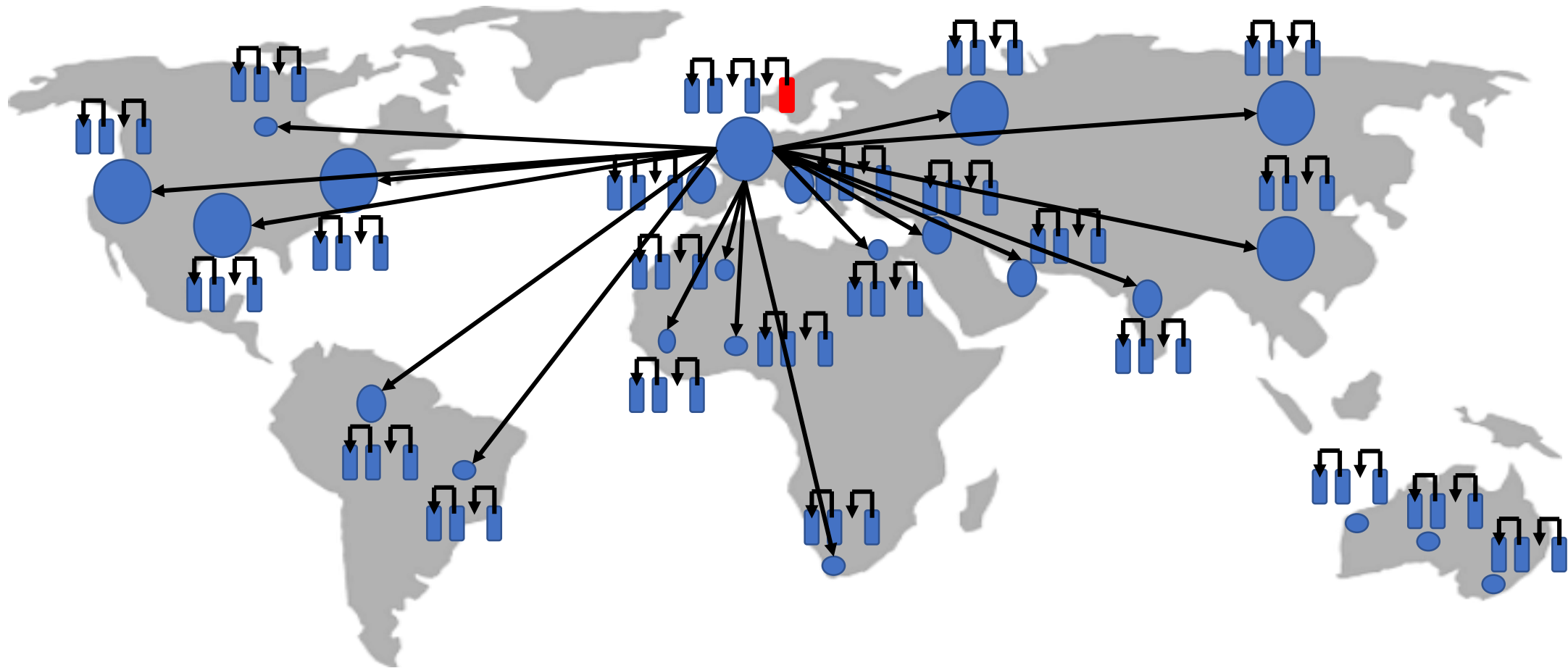
Can Network Nodes Use Paxos?



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 **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)
- 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)

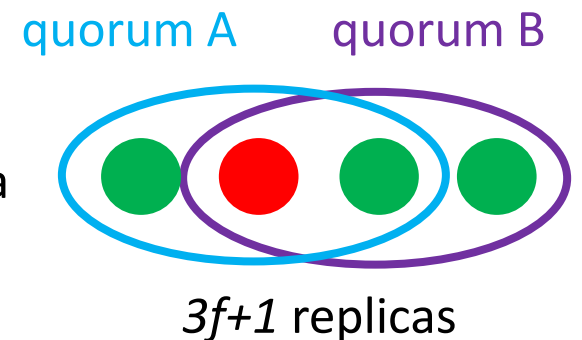
- Goal: Implement a deterministic replication service with **arbitrary malicious faults** in an asynchronous environment
- No assumptions about faulty behavior
- No bounds on delays
- Provides **safety** in asynchronous system and assume eventual time bounds for **liveness**
- Assumptions:

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



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

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



Algorithm

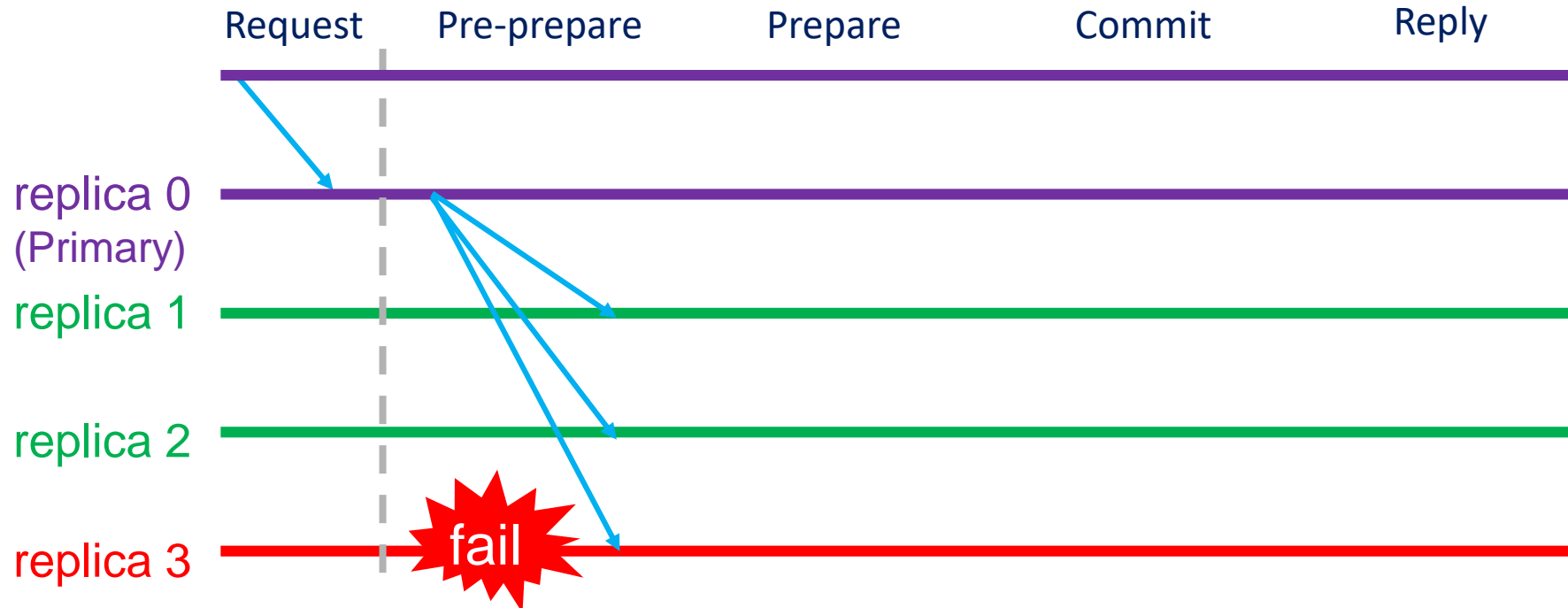
The algorithm has three main phases: (1) *pre-prepare* picks order of requests (2) *prepare* ensures order within views, (3) *commit* ensures order across views



Algorithm

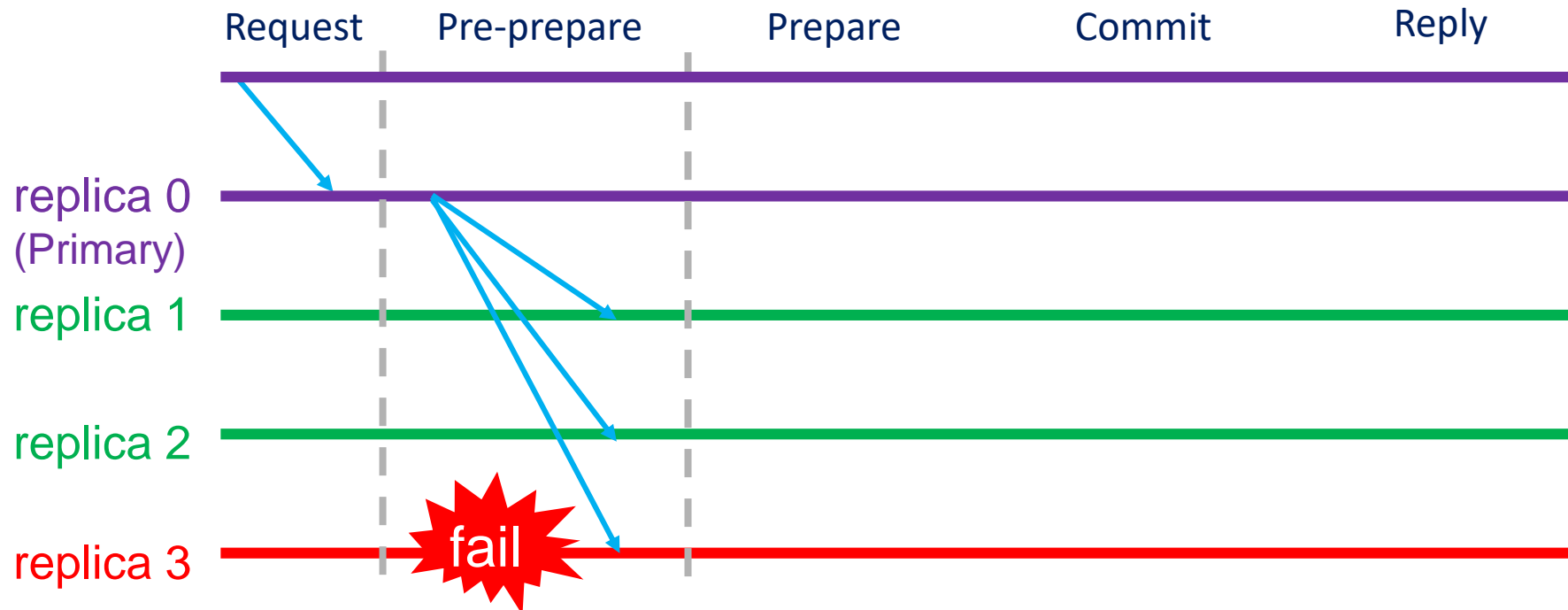
The algorithm has three main phases: (1) *pre-prepare* picks order of requests (2) *prepare* ensures order within views, (3) *commit* ensures order across views

(2) The primary multicasts the request to the backups



Algorithm

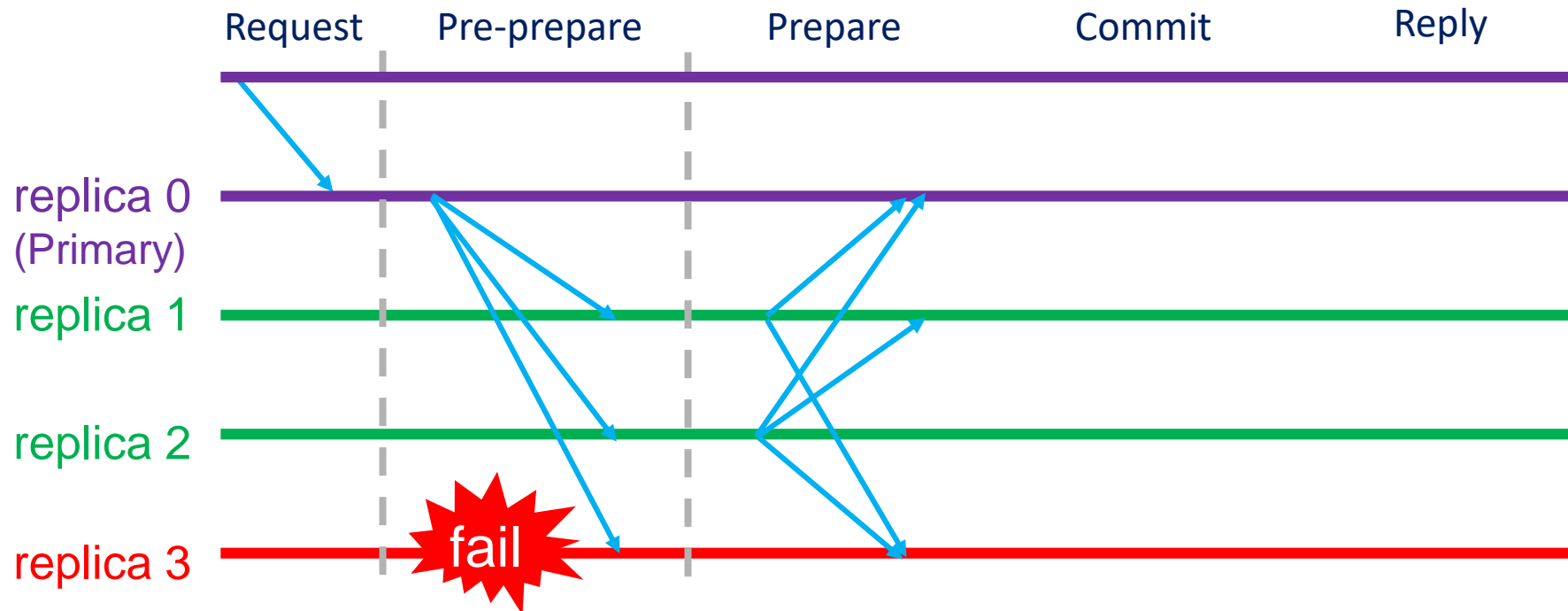
The algorithm has three main phases: (1) *pre-prepare* picks order of requests (2) *prepare* ensures order within views, (3) *commit* ensures order across views



Algorithm

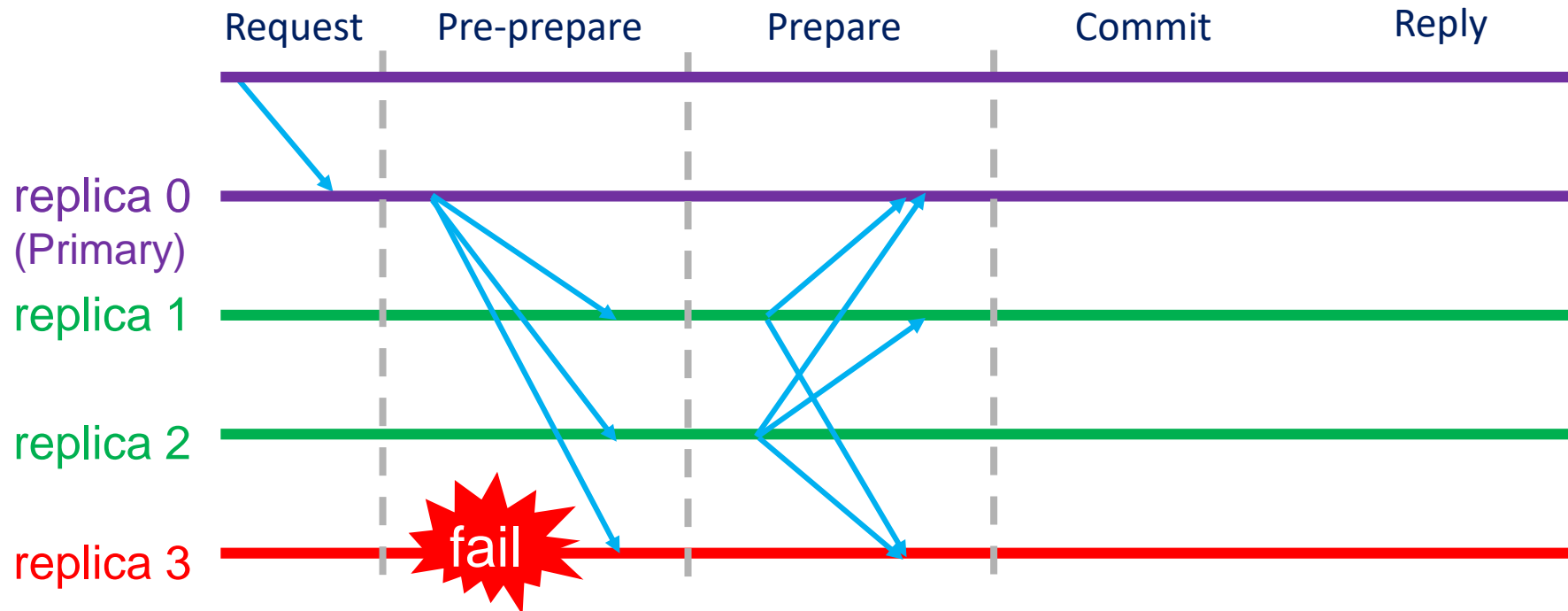
The algorithm has three main phases: (1) *pre-prepare* picks order of requests (2) *prepare* ensures order within views, (3) *commit* ensures order across views

(3) Backups multicast **PREPARE** message



Algorithm

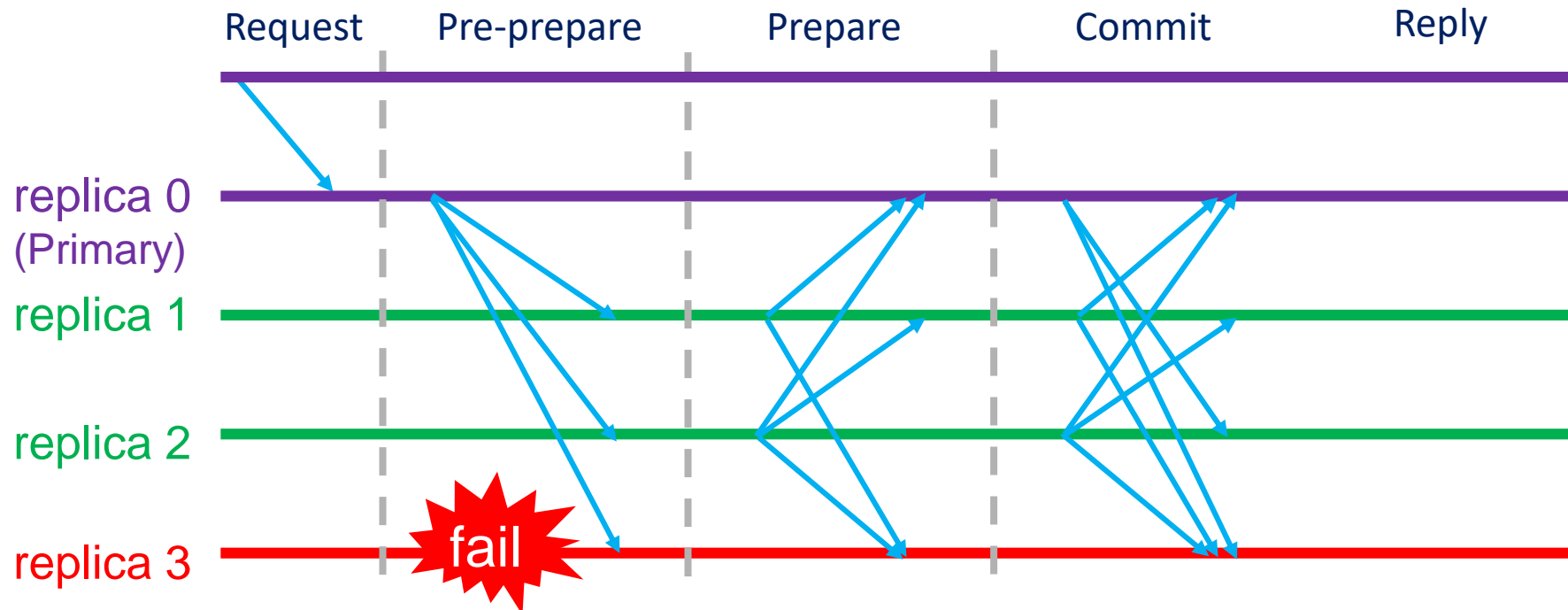
The algorithm has three main phases: (1) *pre-prepare* picks order of requests (2) *prepare* ensures order within views, (3) *commit* ensures order across views



Algorithm

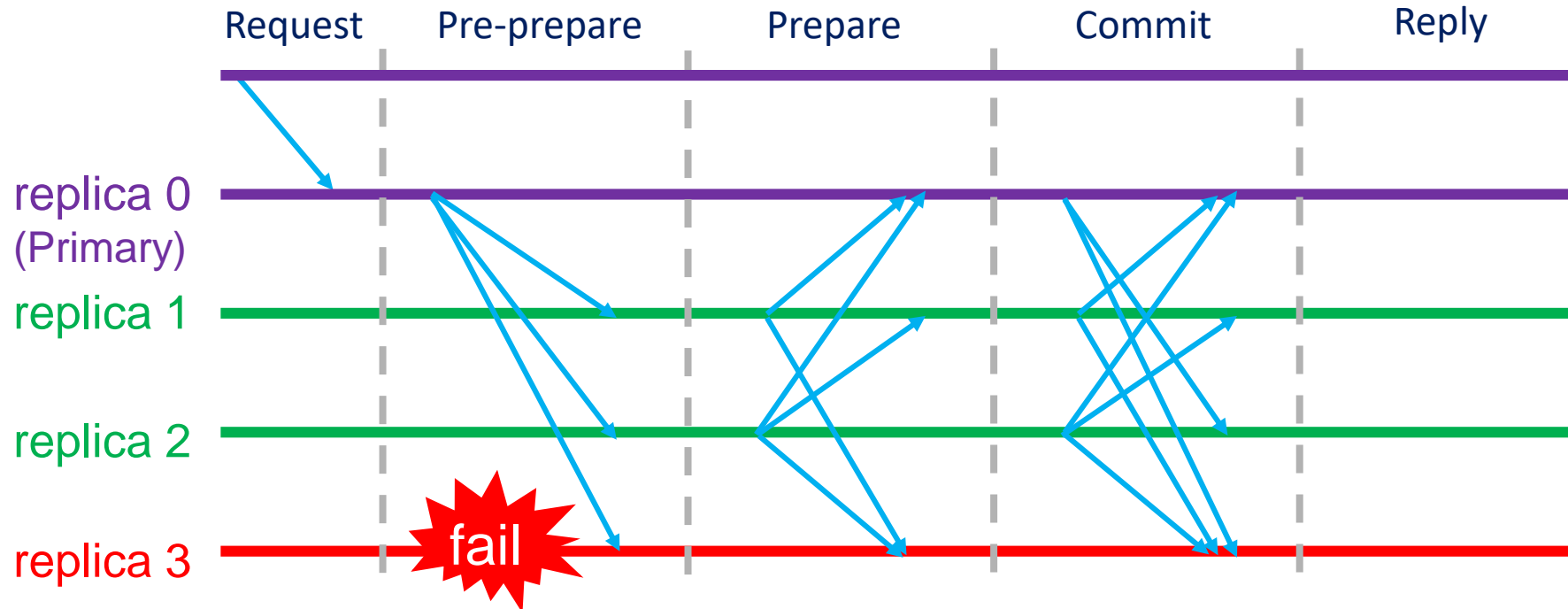
The algorithm has three main phases: (1) *pre-prepare* picks order of requests (2) *prepare* ensures order within views, (3) *commit* ensures order across views

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



Algorithm

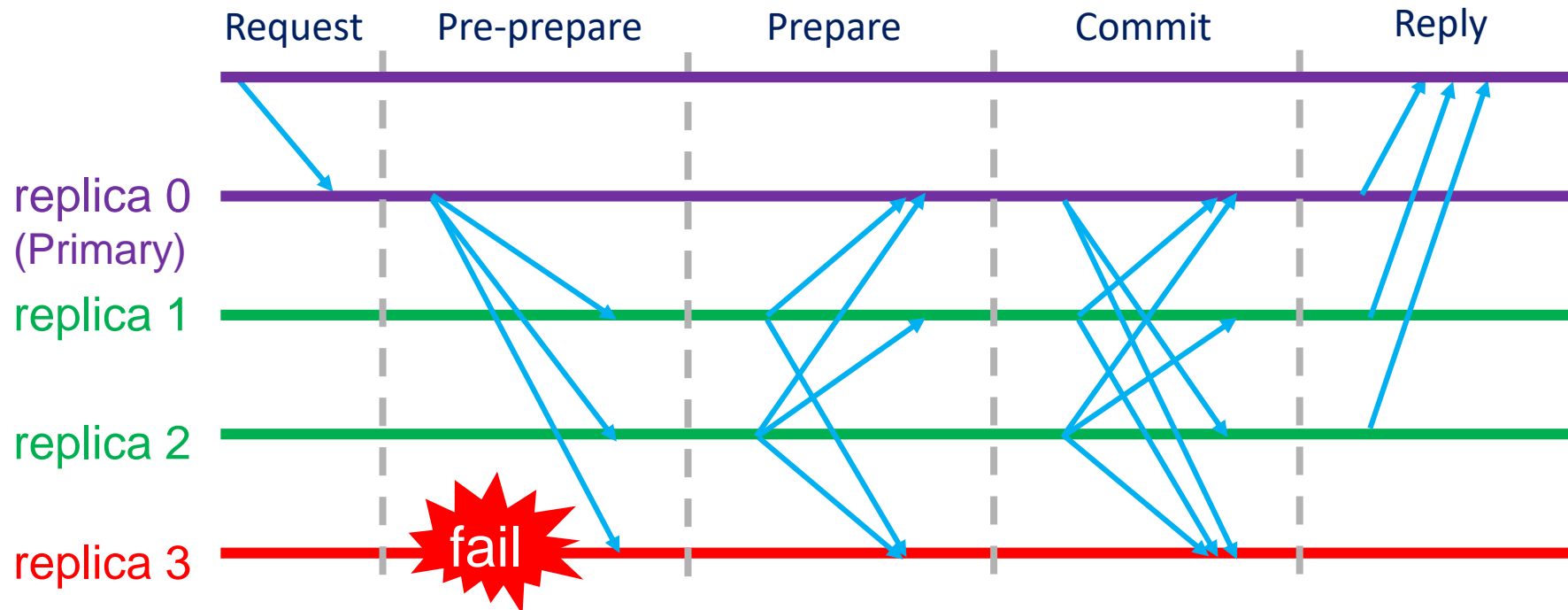
The algorithm has three main phases: (1) *pre-prepare* picks order of requests (2) *prepare* ensures order within views, (3) *commit* ensures order across views



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

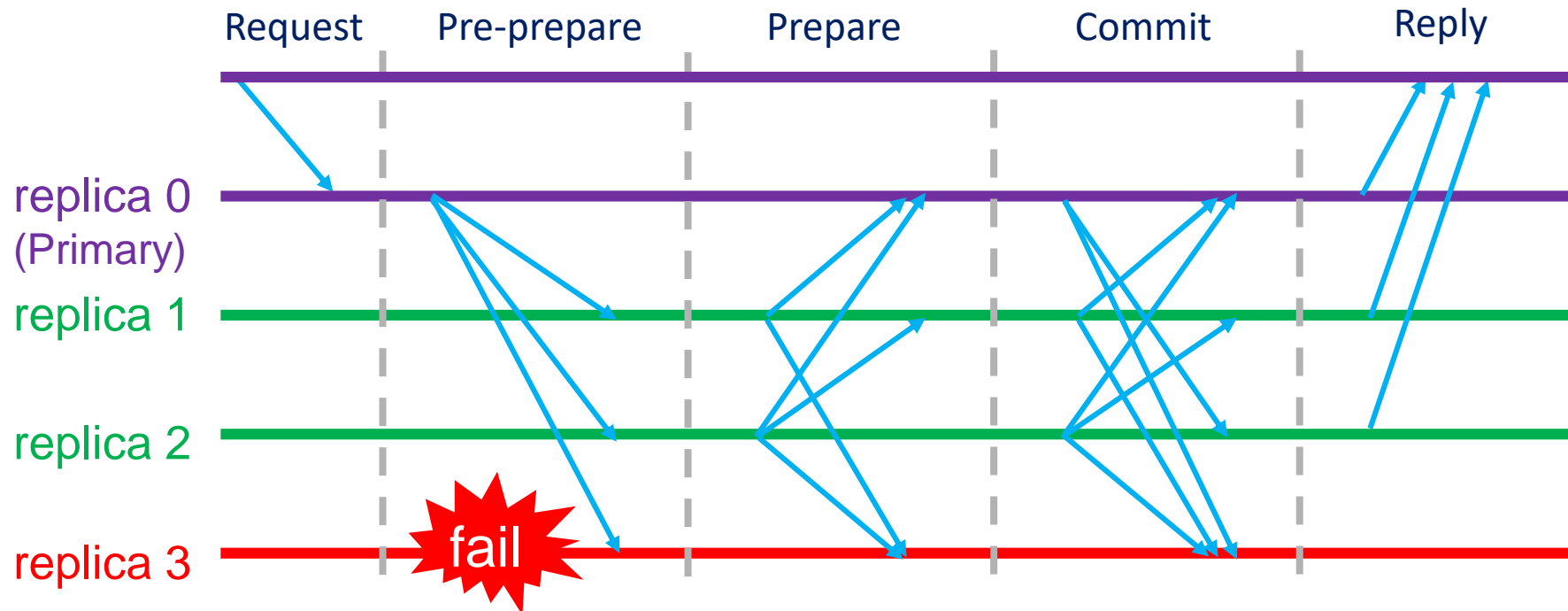
(5) If a replica receives at least $2f$ COMMIT messages, reply the result to the client



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

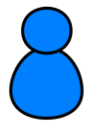
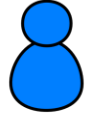
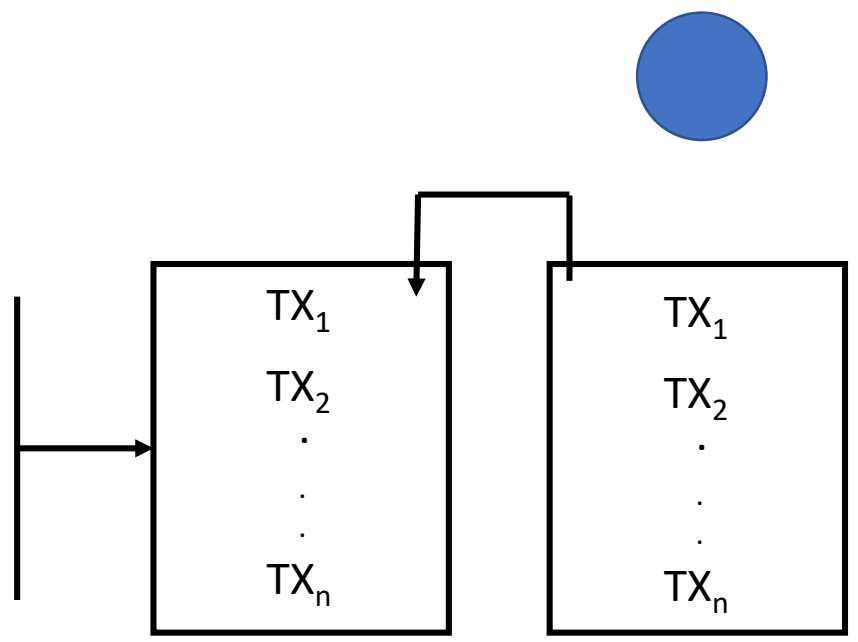
- Intuitively, network nodes race to solve a puzzle
- This puzzle is computationally expensive
- Once a network node finds (mines) a solution:
 - It adds its block of transactions to the blockchain
 - It multi-casts the solution to other network nodes
 - Other network nodes accept and verify the solution

Mining Details

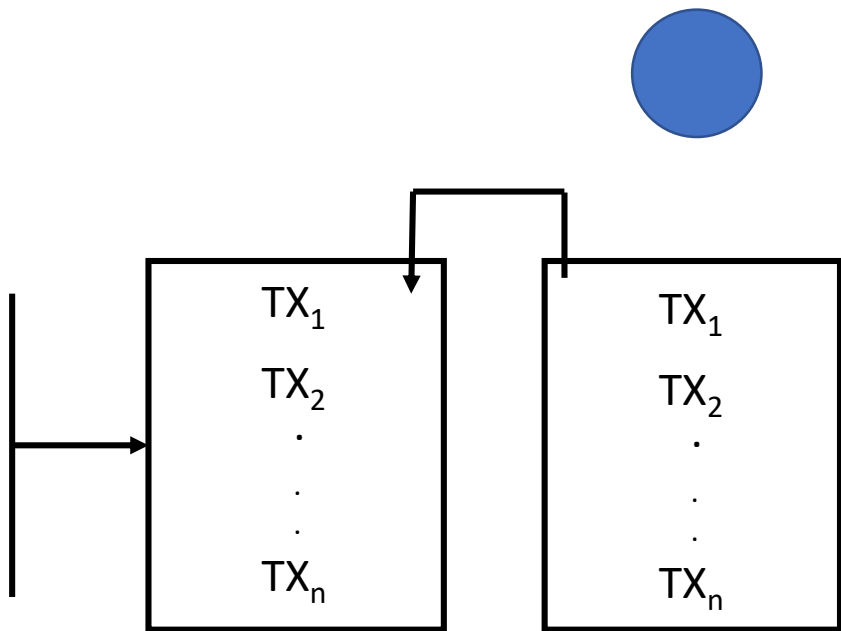
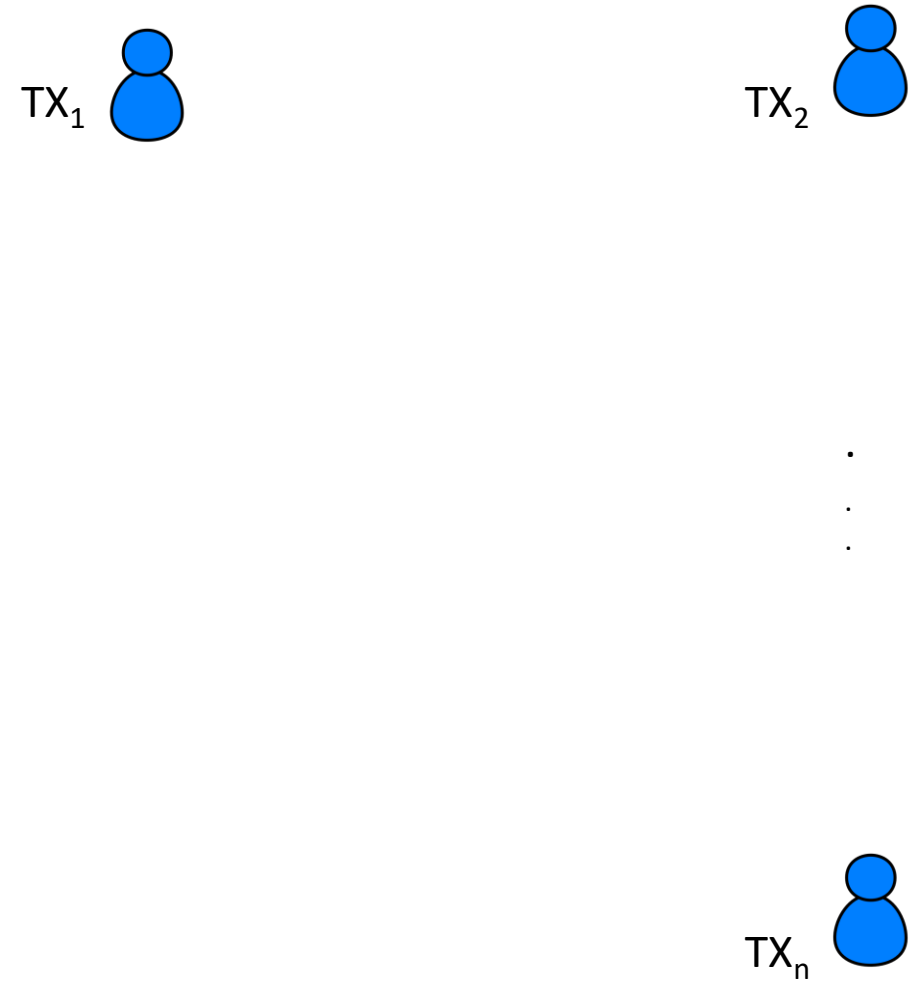
Mining Details



Mining Details



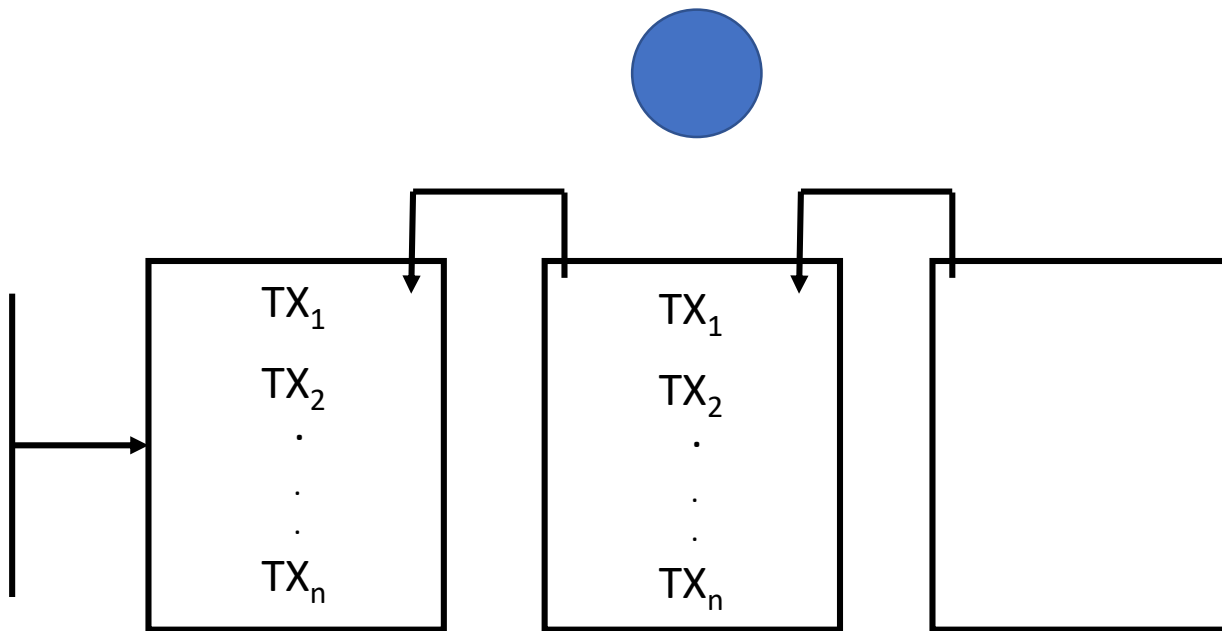
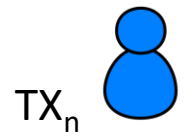
Mining Details



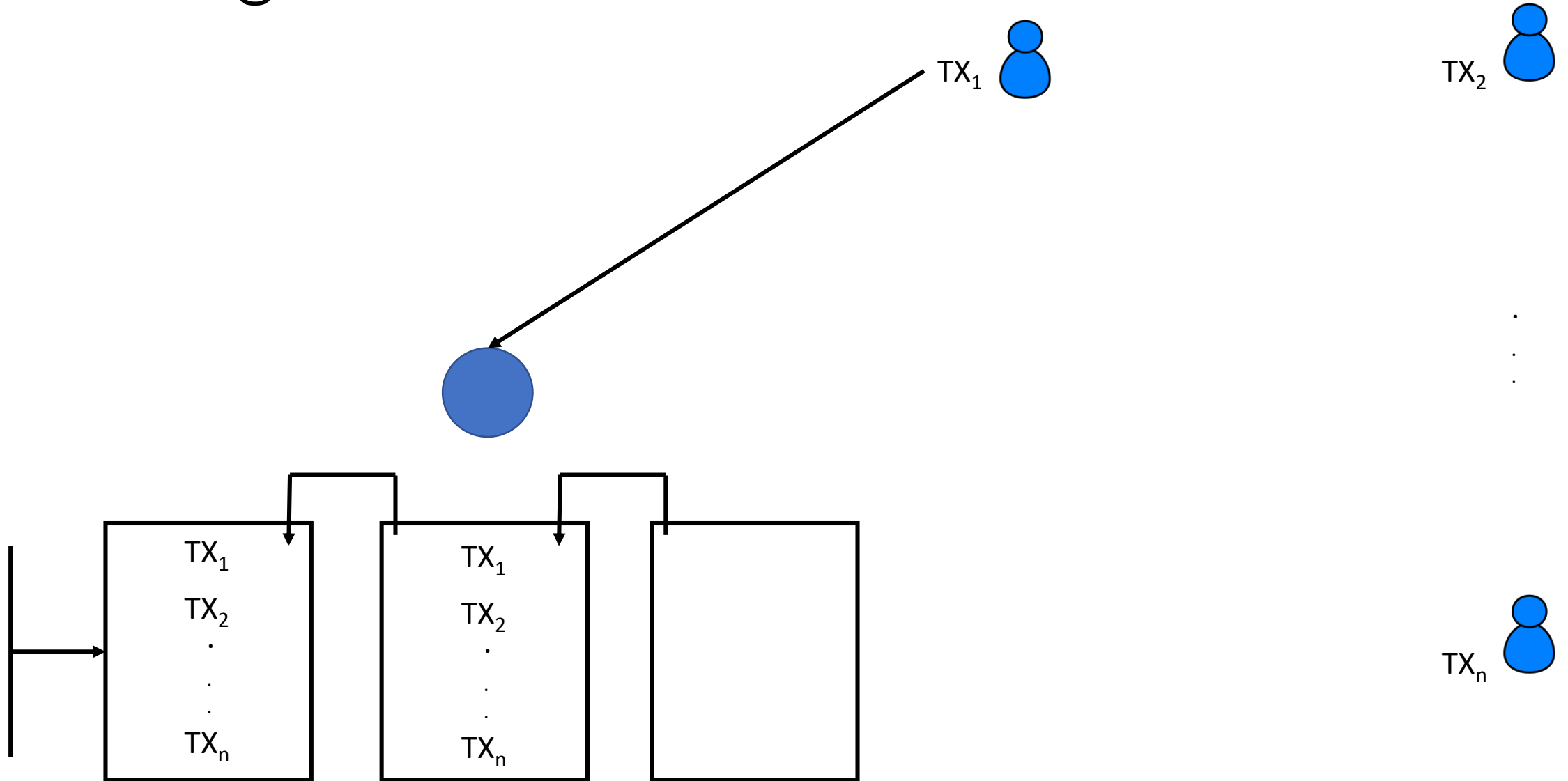
Mining Details



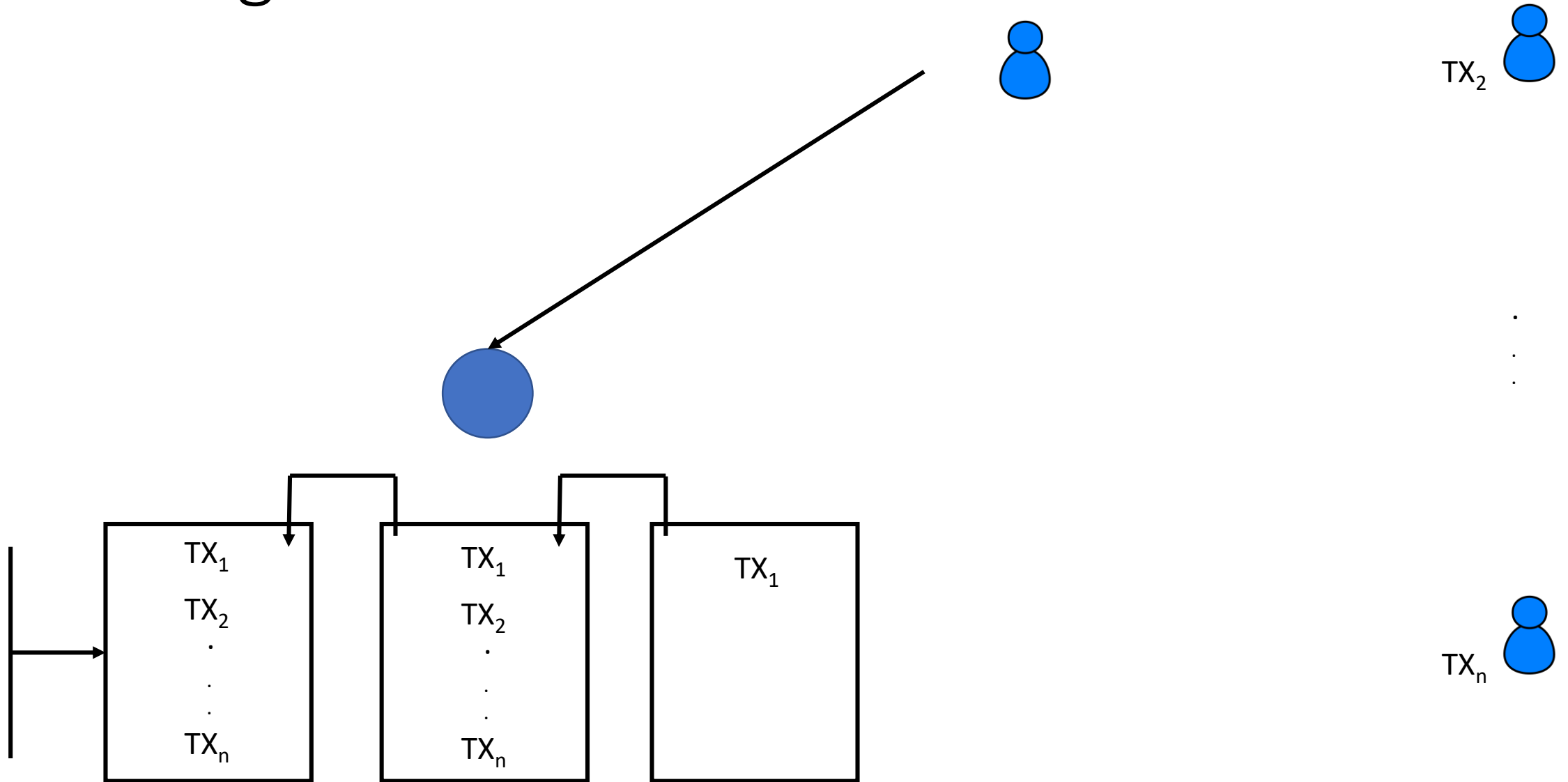
⋮



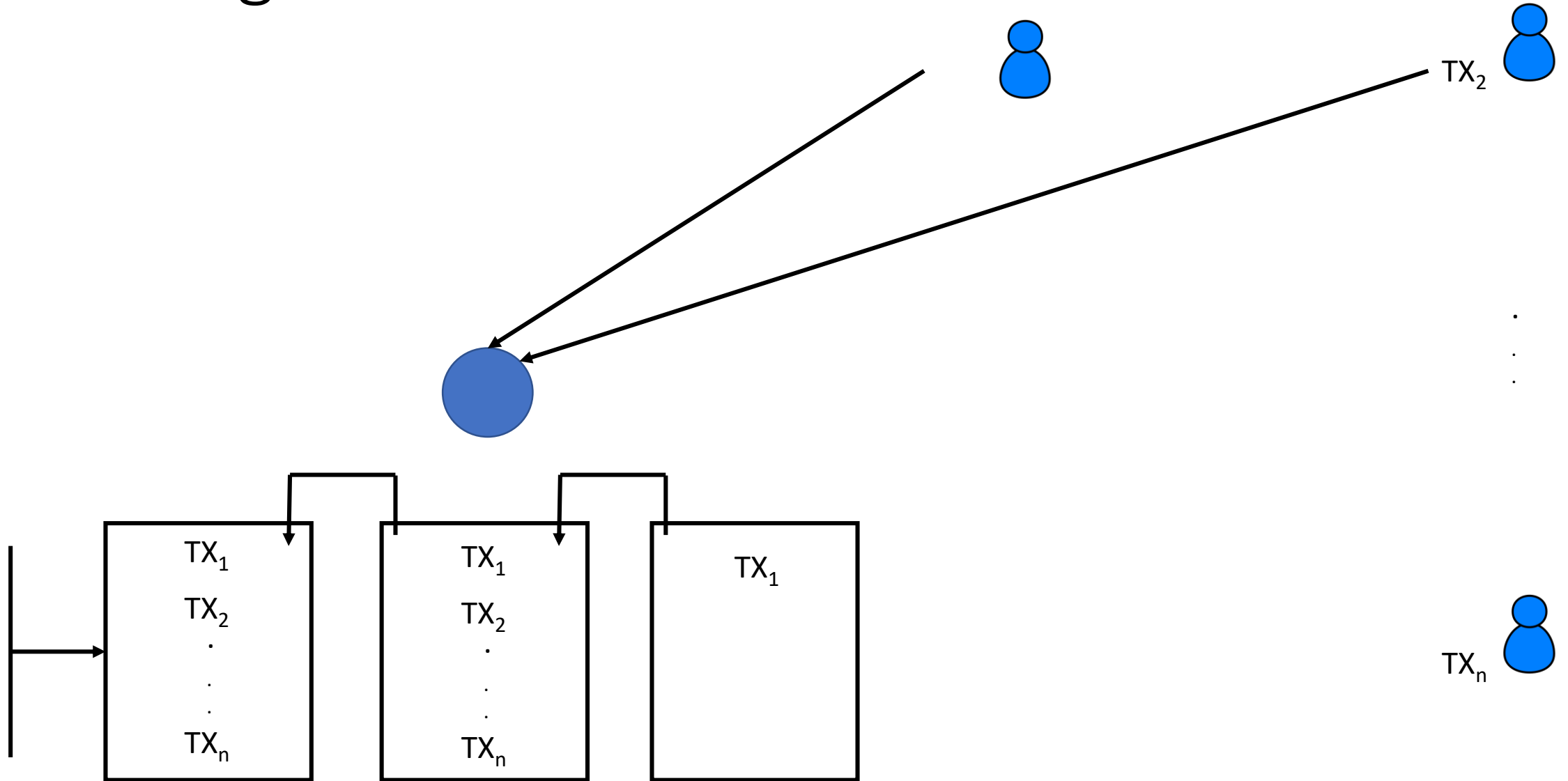
Mining Details



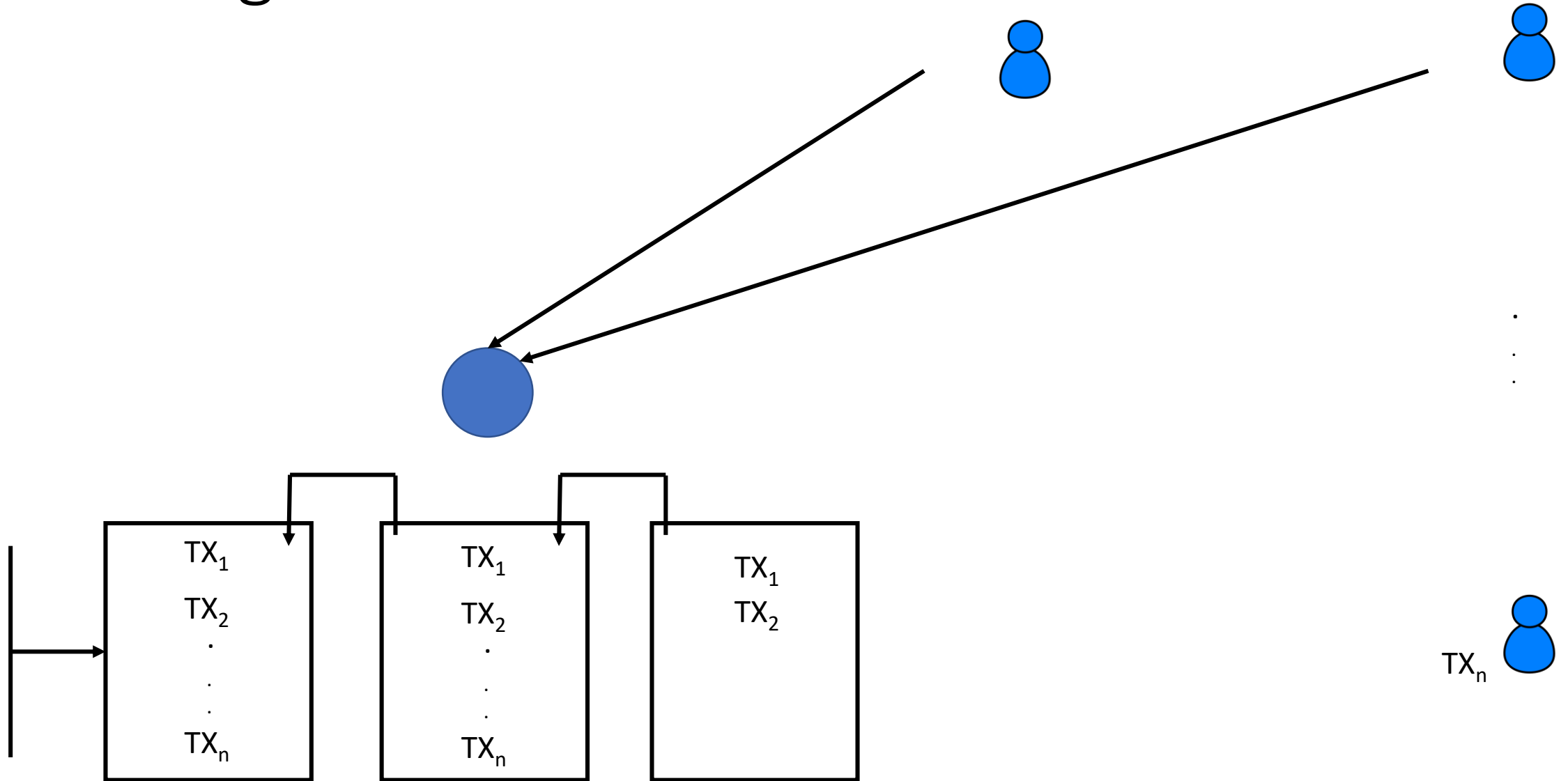
Mining Details



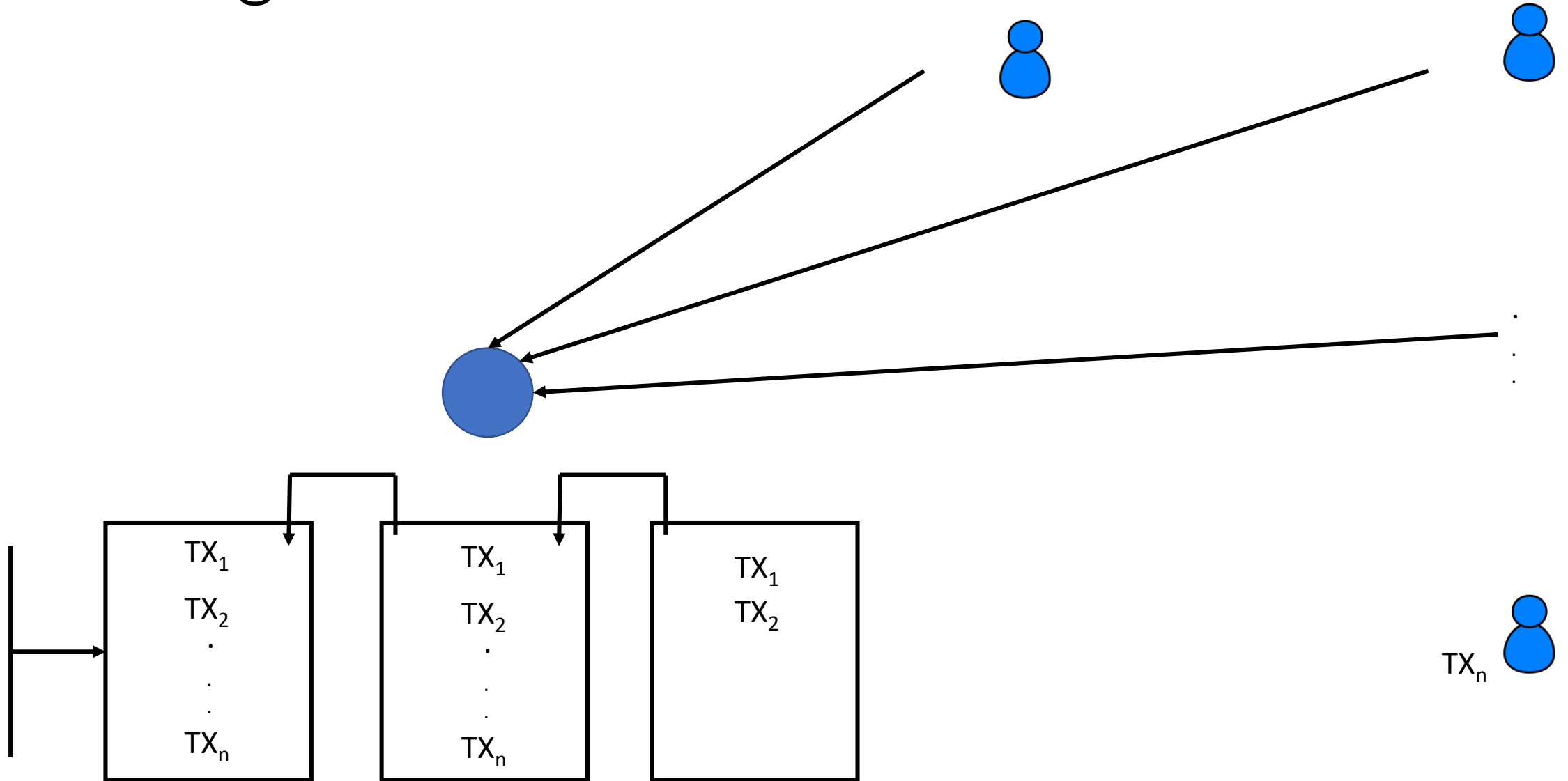
Mining Details



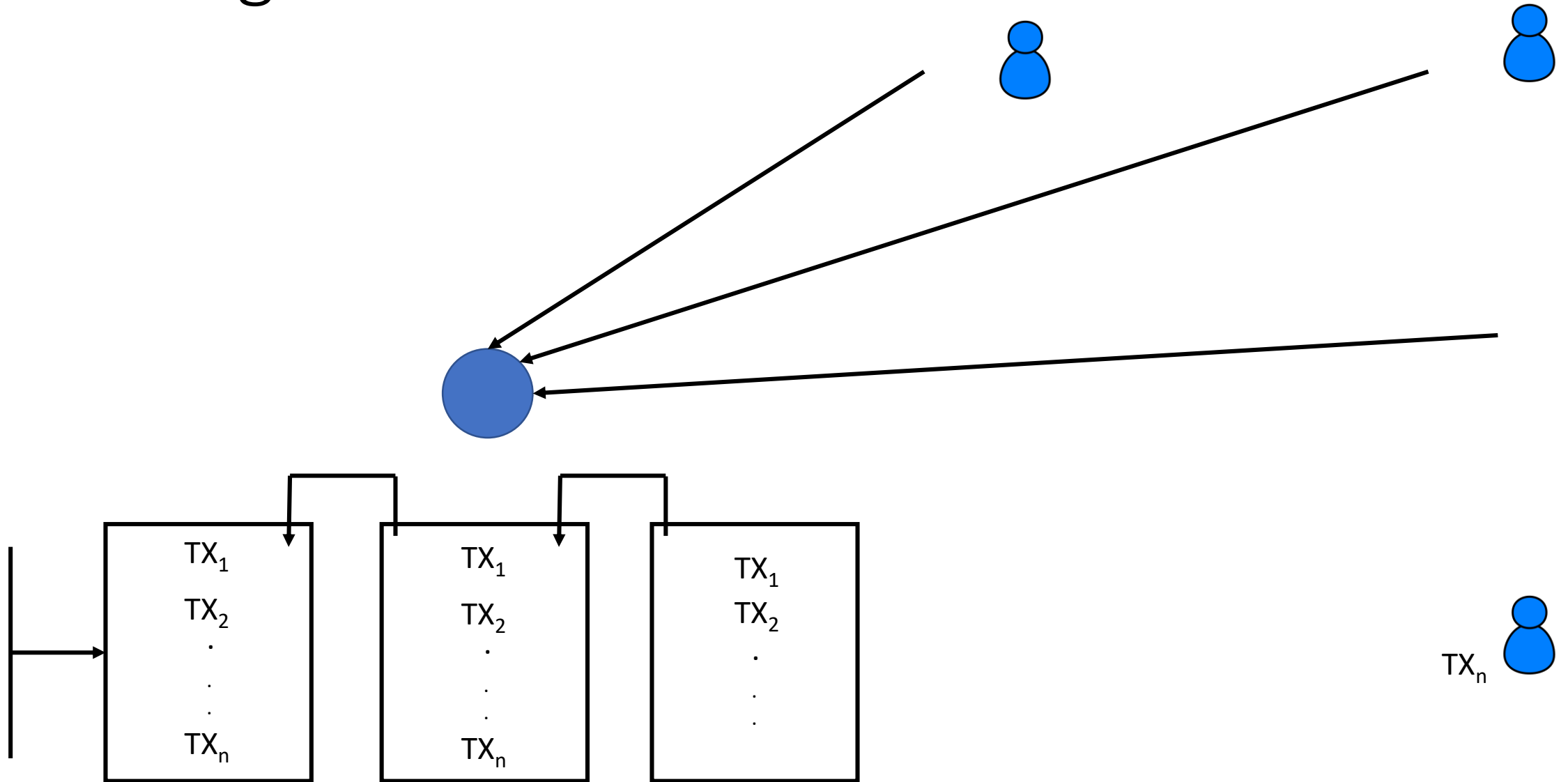
Mining Details



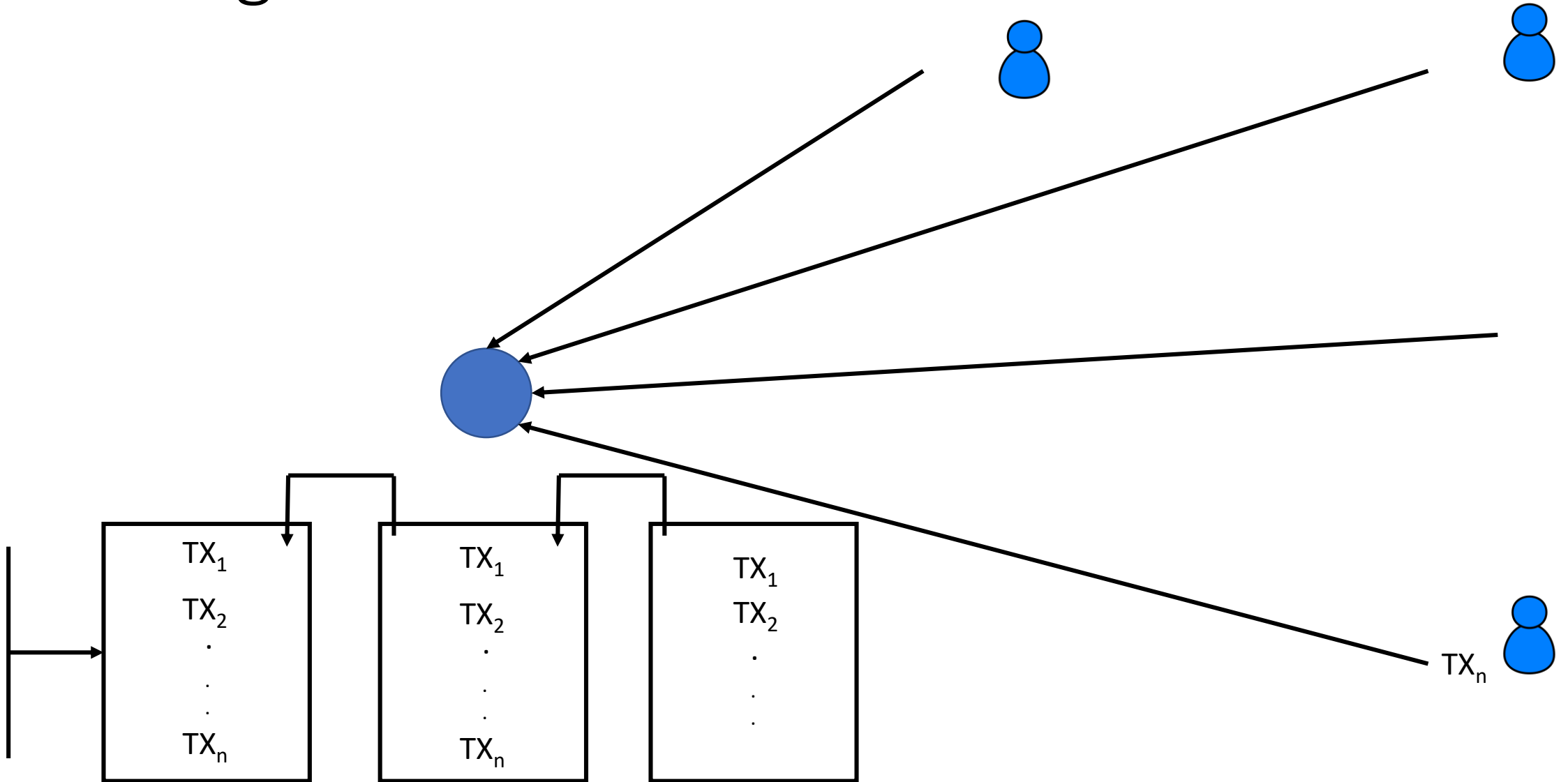
Mining Details



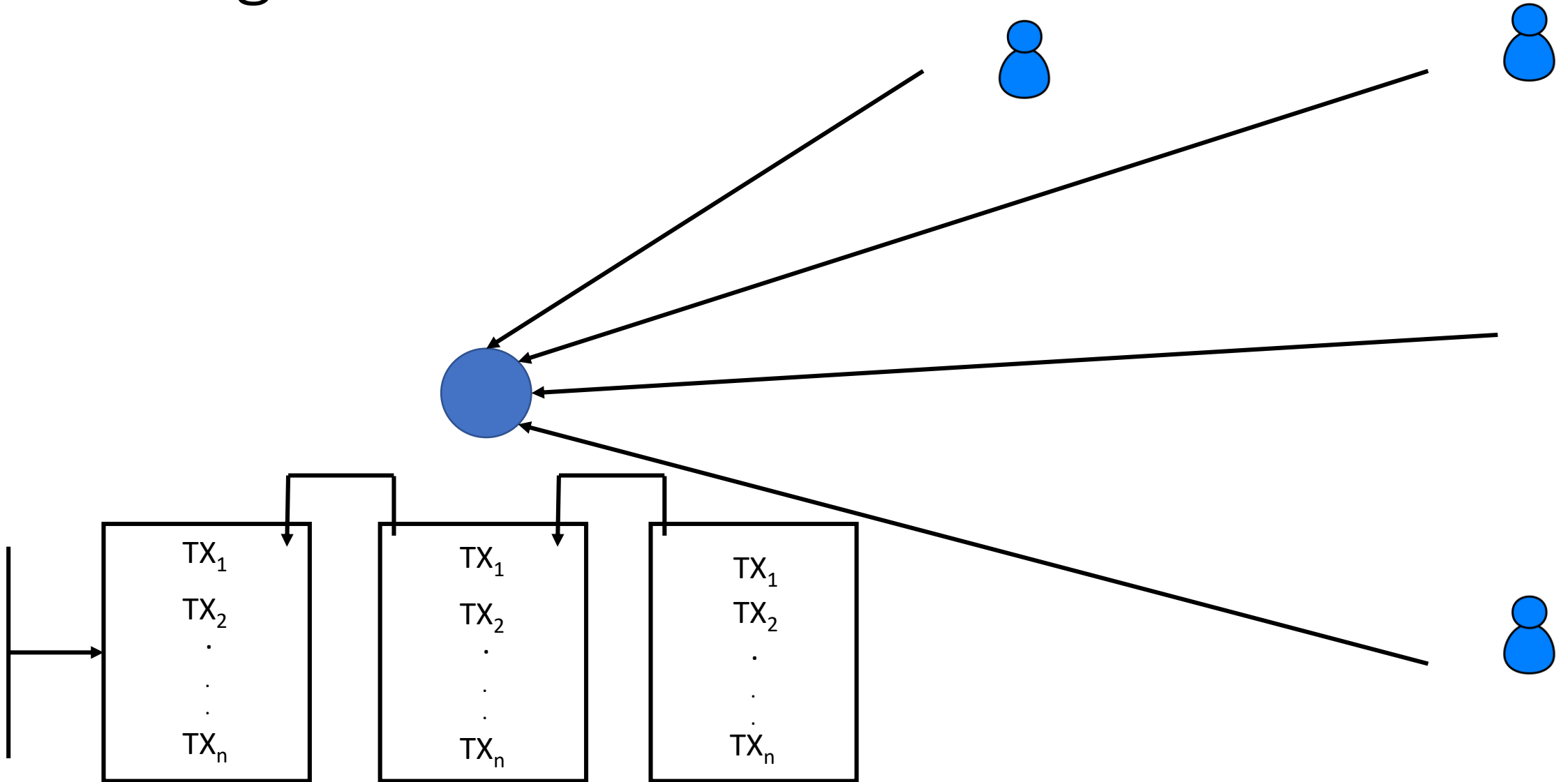
Mining Details



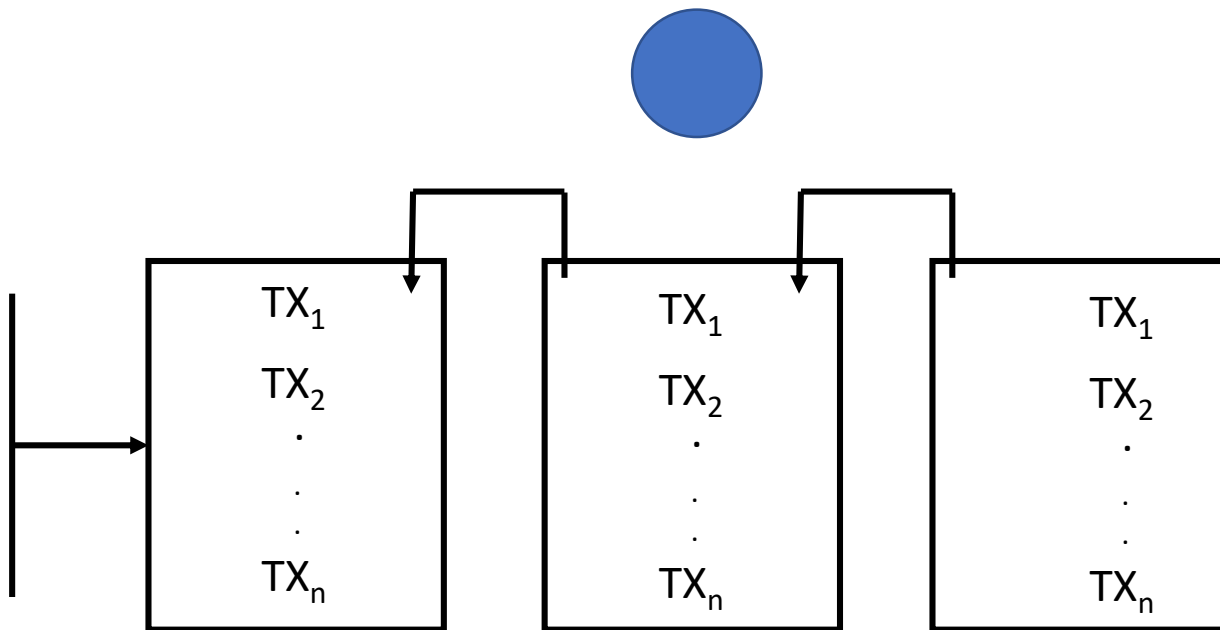
Mining Details



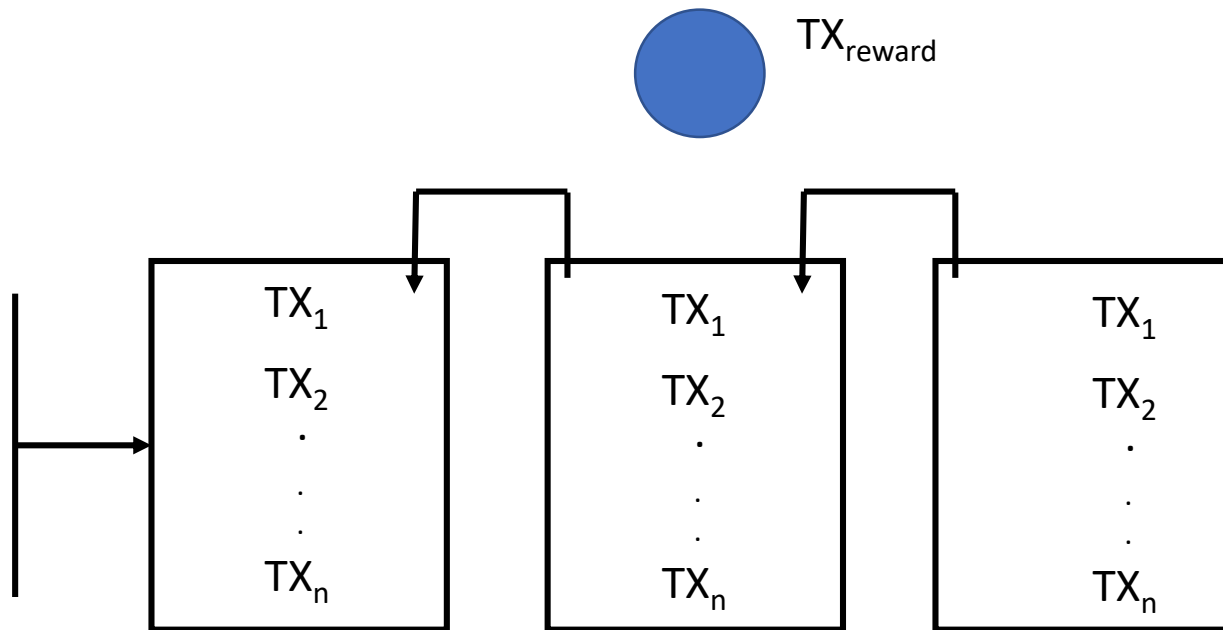
Mining Details



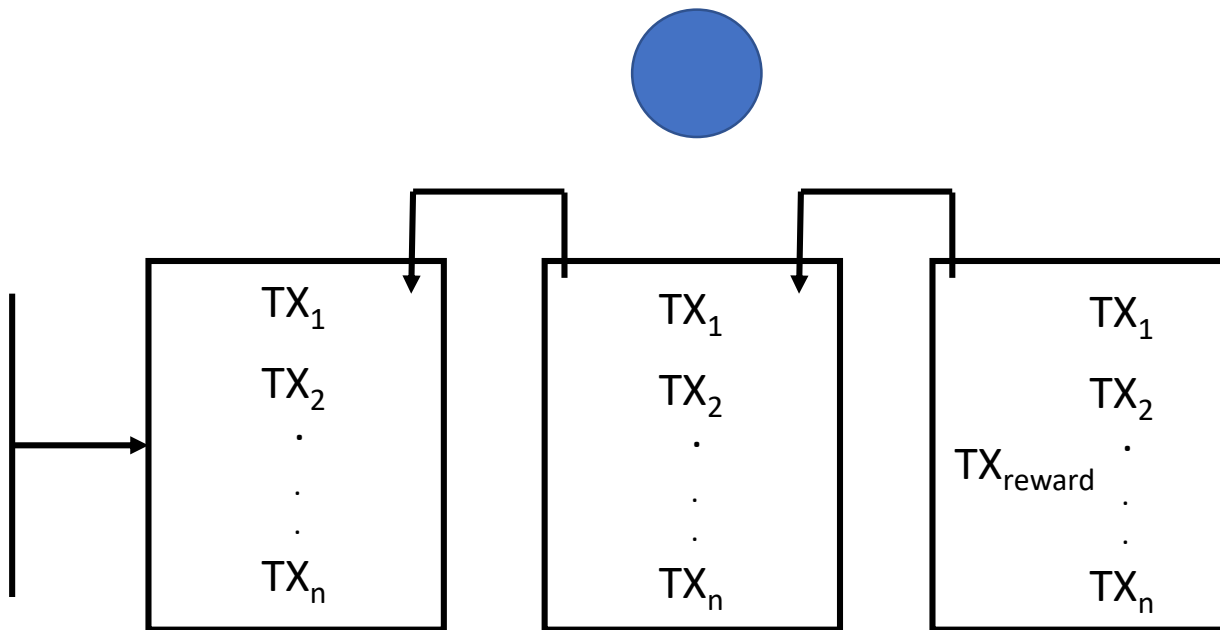
Mining Details



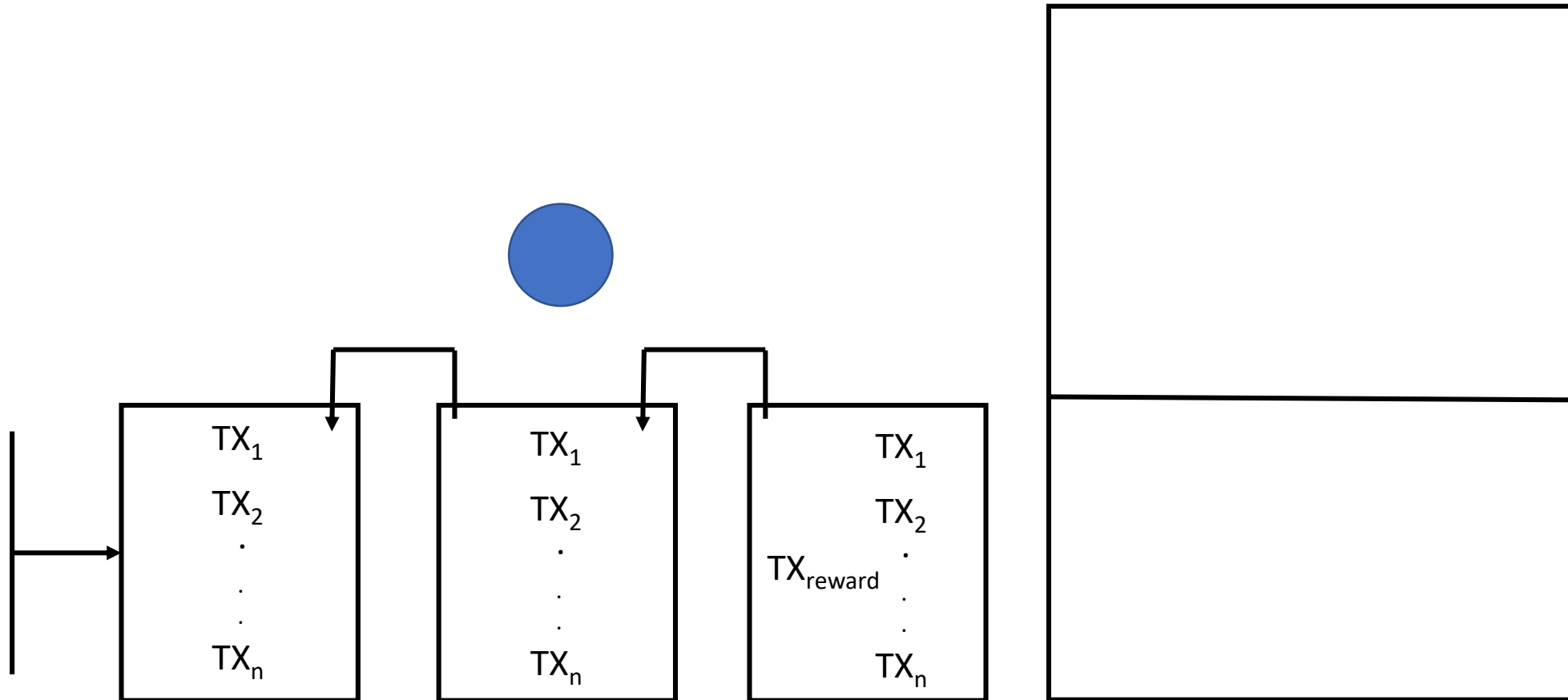
Mining Details



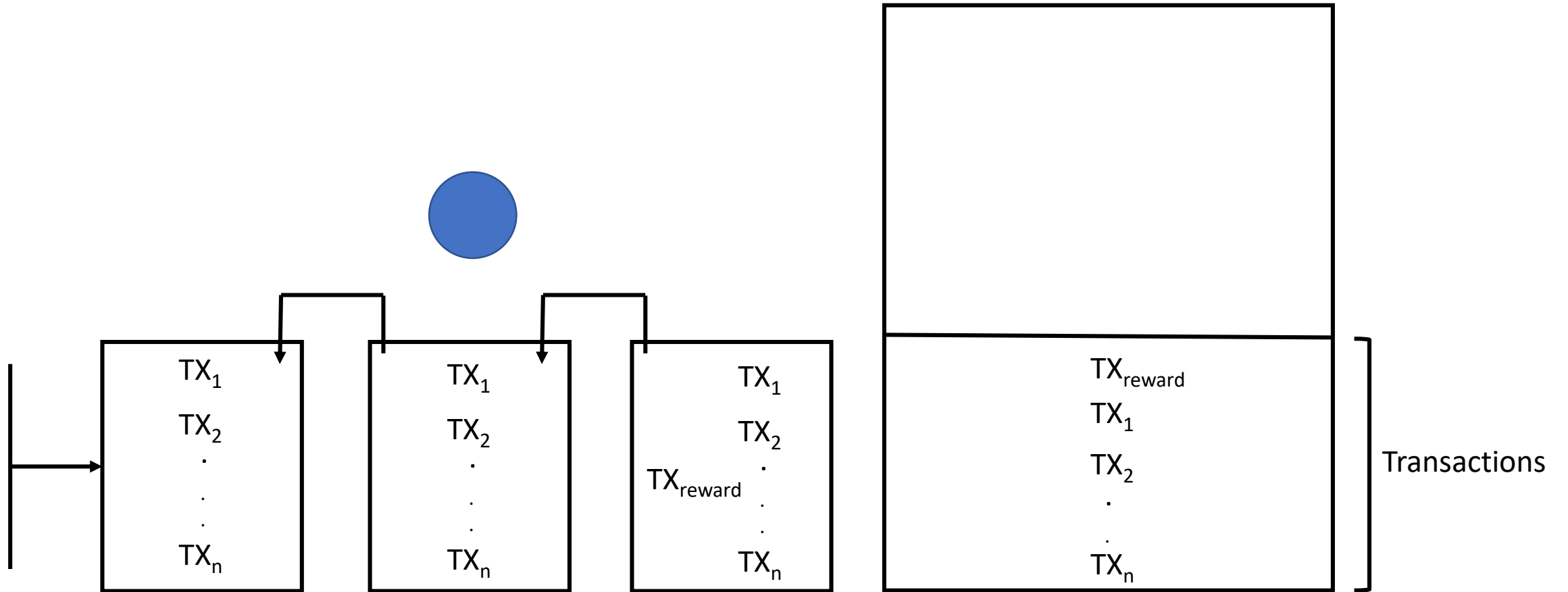
Mining Details



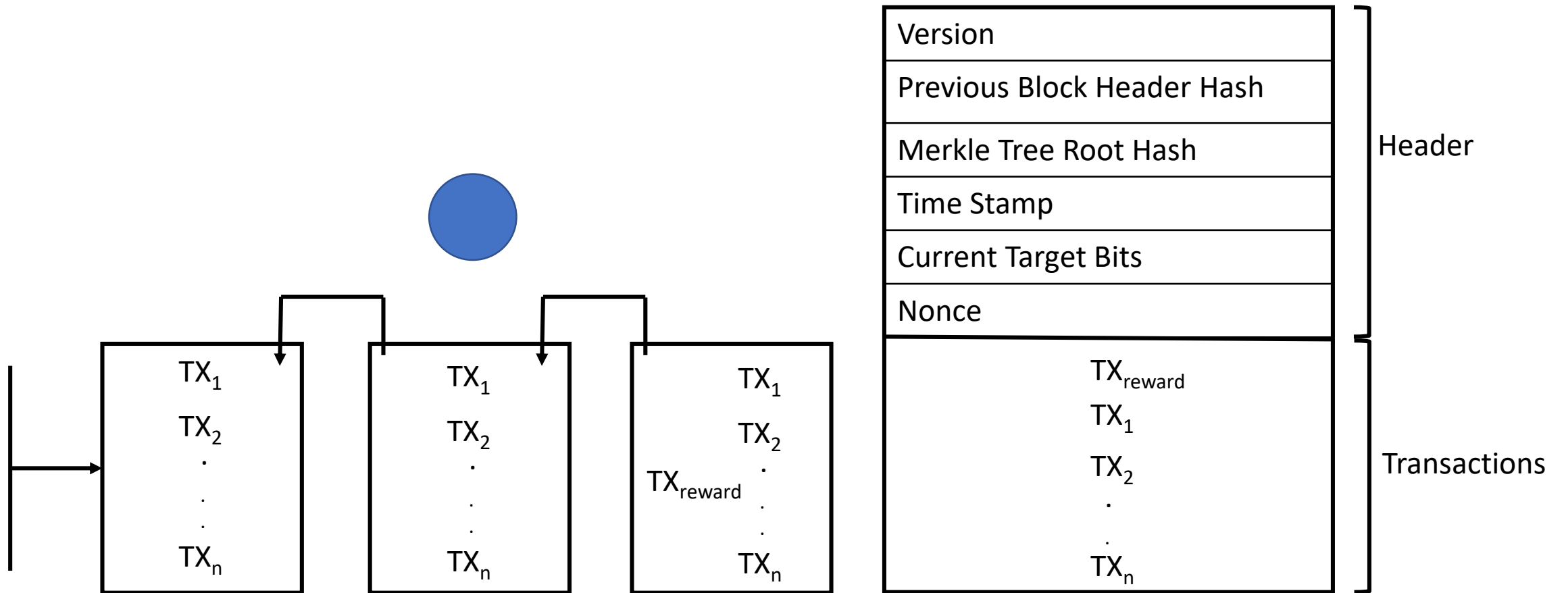
Mining Details



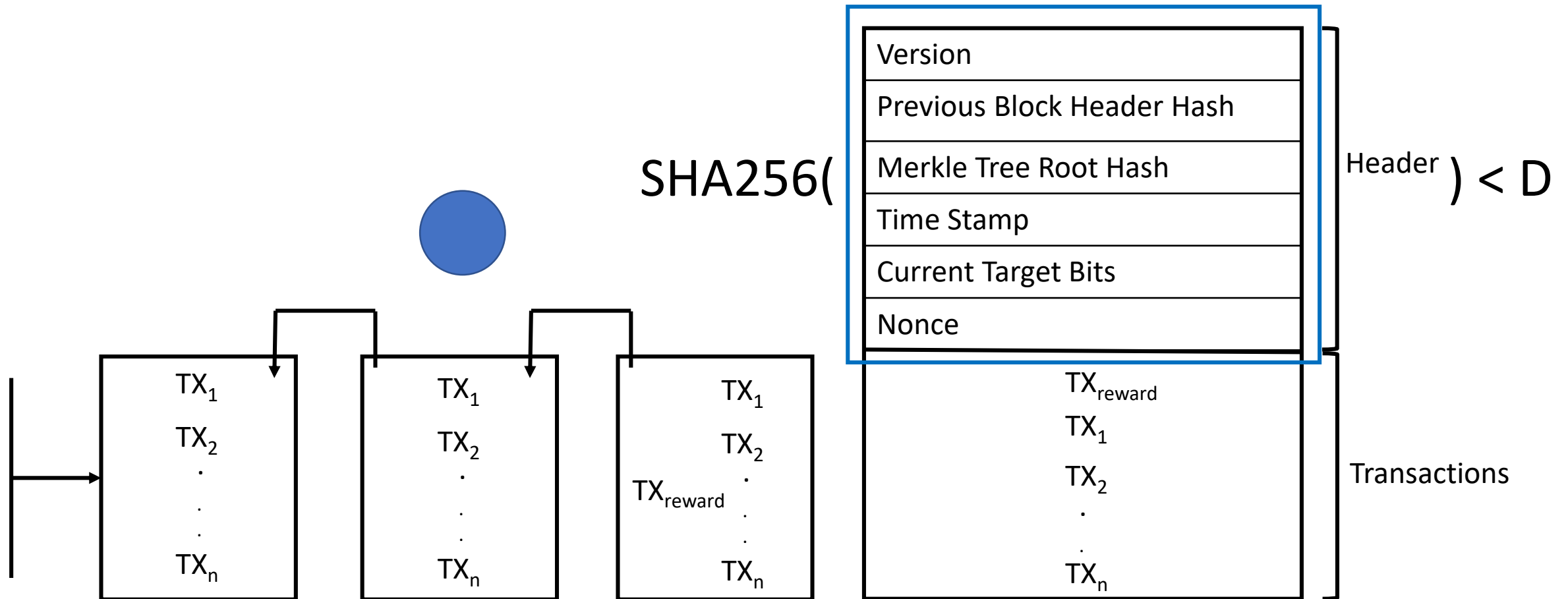
Mining Details



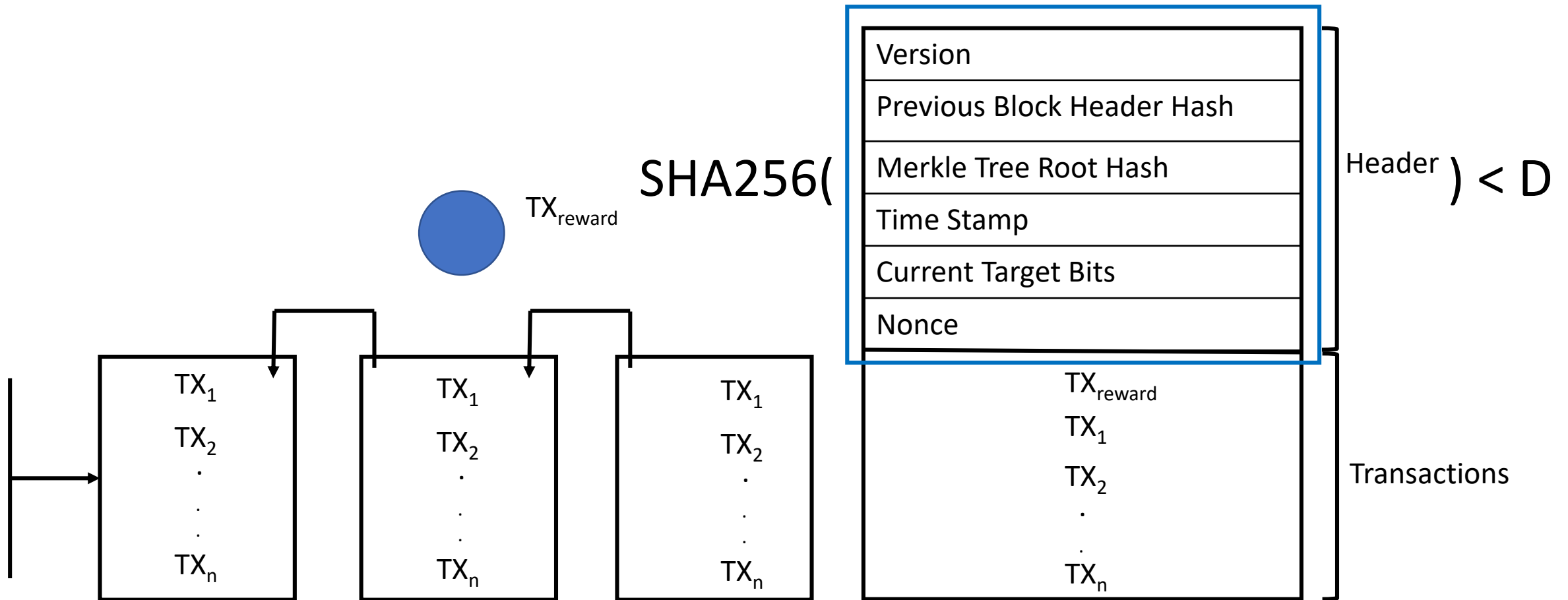
Mining Details



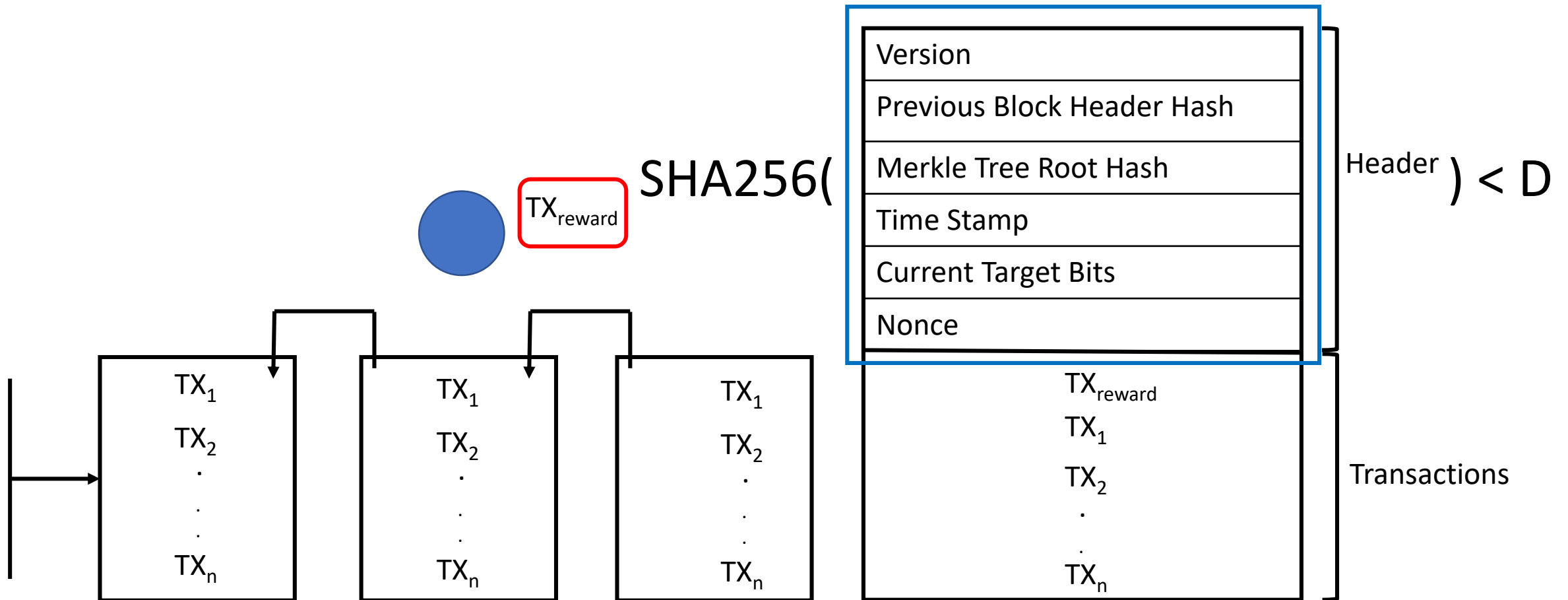
Mining Details



Mining Details



Mining Details



Mining Details

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

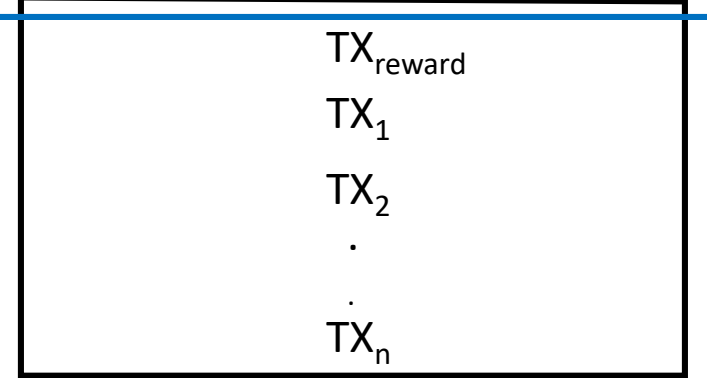
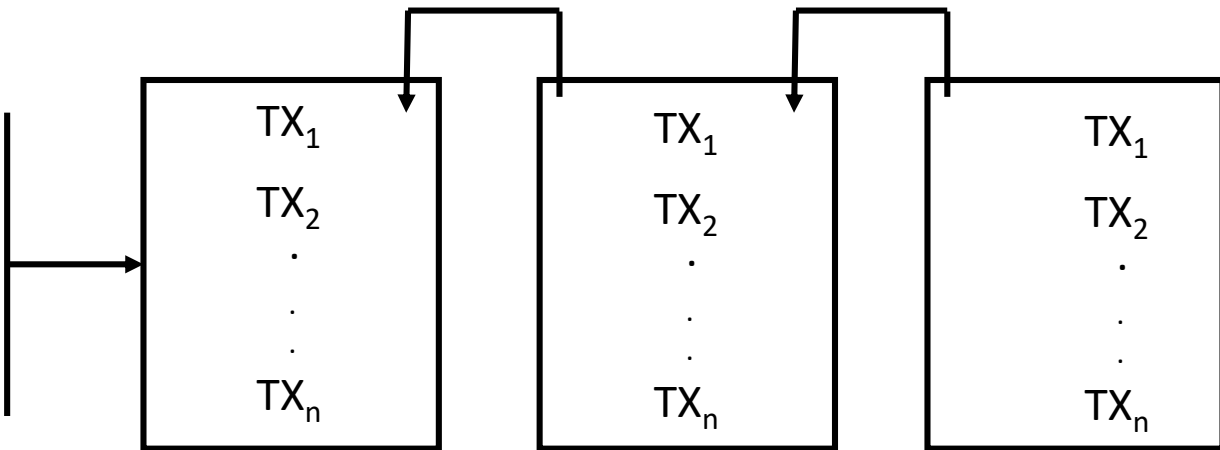
SHA256(



TX_{reward}

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

Header) < D



Transactions

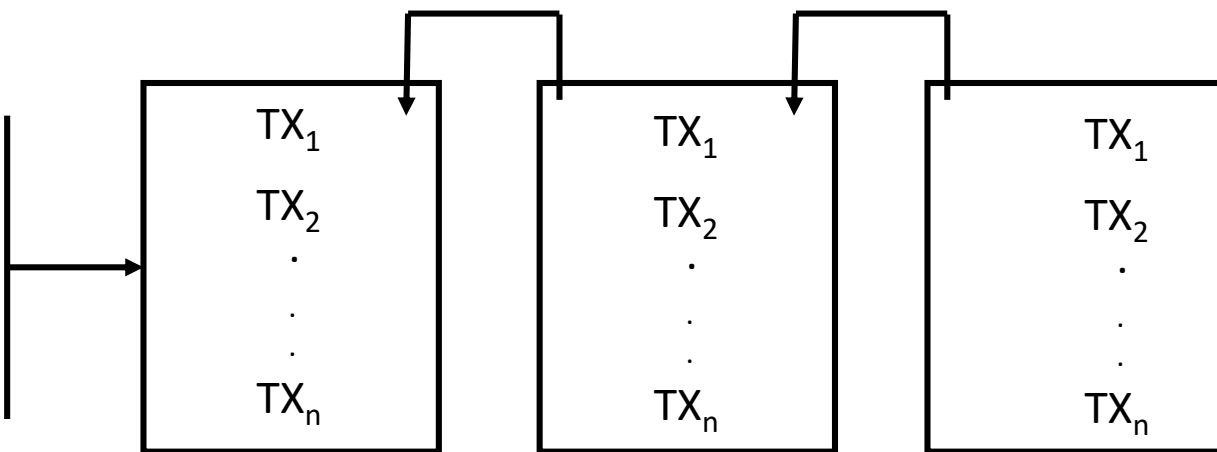
Mining Details

- TX_{reward} is self signed (also called coinbase transaction)
- First signature? Self signed ☺
- TX_{reward} is bitcoin's way to create new coins

SHA256(



TX_{reward}



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

Header) < D

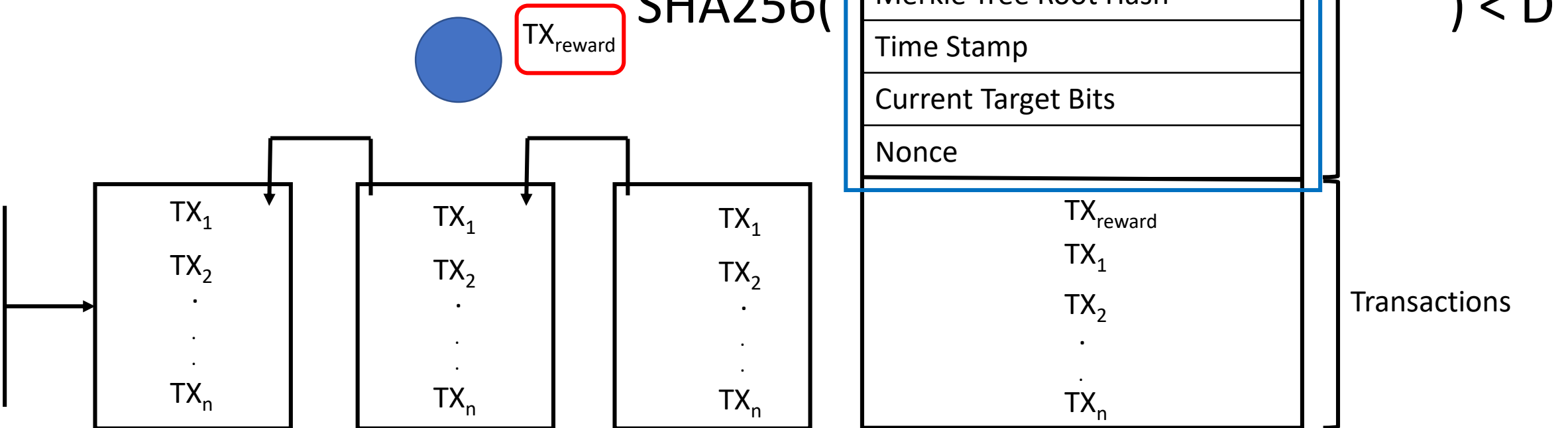
TX_{reward}
TX_1
TX_2
⋮
TX_n

Transactions

Mining Details

- TX_{reward} is self signed (also called coinbase transaction)
- First signature? Self signed ☺
- TX_{reward} is bitcoin's way to create new coins
- The reward value is halved every 4 years (210,000 blocks)

SHA256(



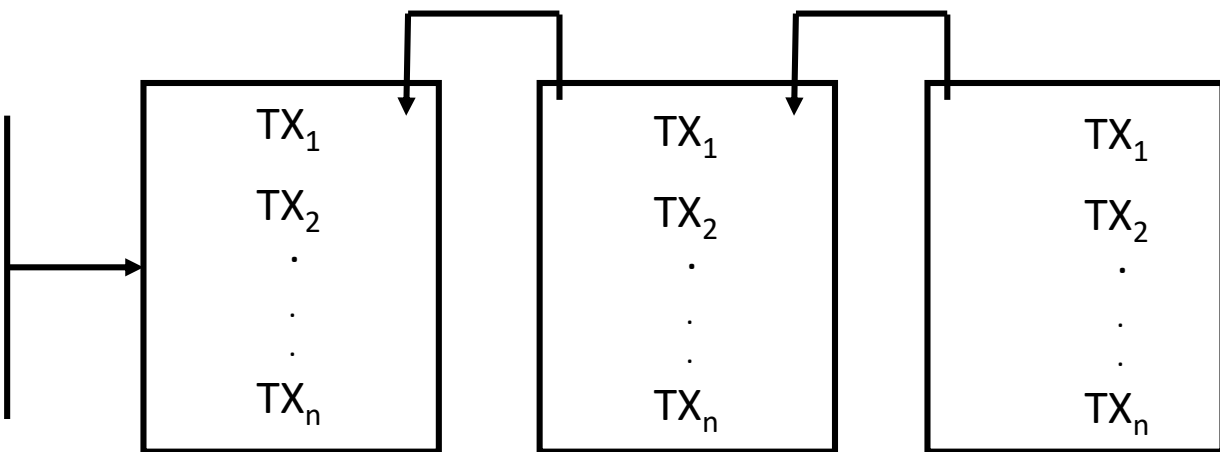
Mining Details

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

SHA256(

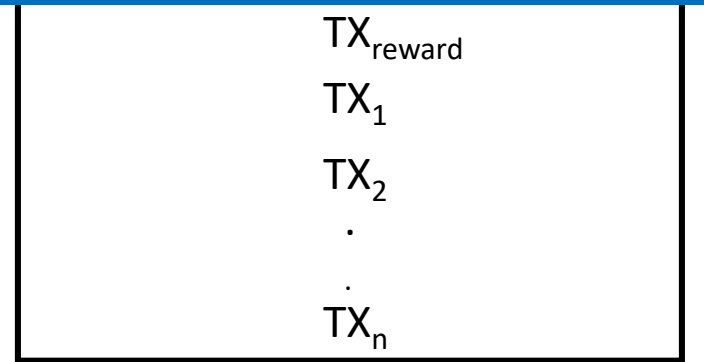


TX_{reward}



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

Header) < D

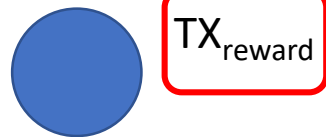


Transactions

Mining Details

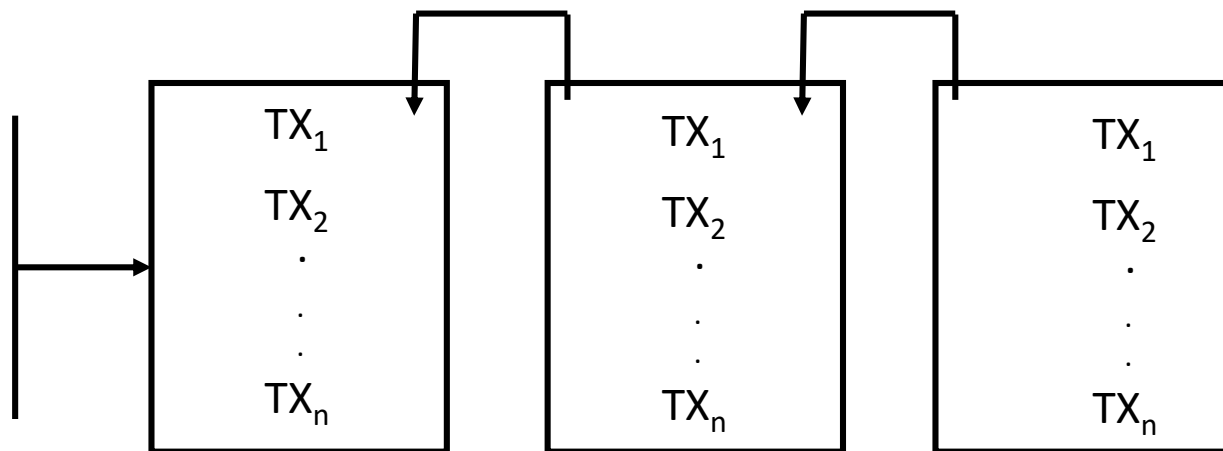
- TX_{reward} is self signed (also called coinbase transaction)
- First signature? Self signed ☺
- TX_{reward} is bitcoin's way to create new coins
- The reward value is halved every 4 years (210,000 blocks)
- Currently, it's 12.5 Bitcoins per block
- Incentives network nodes to mine

SHA256(



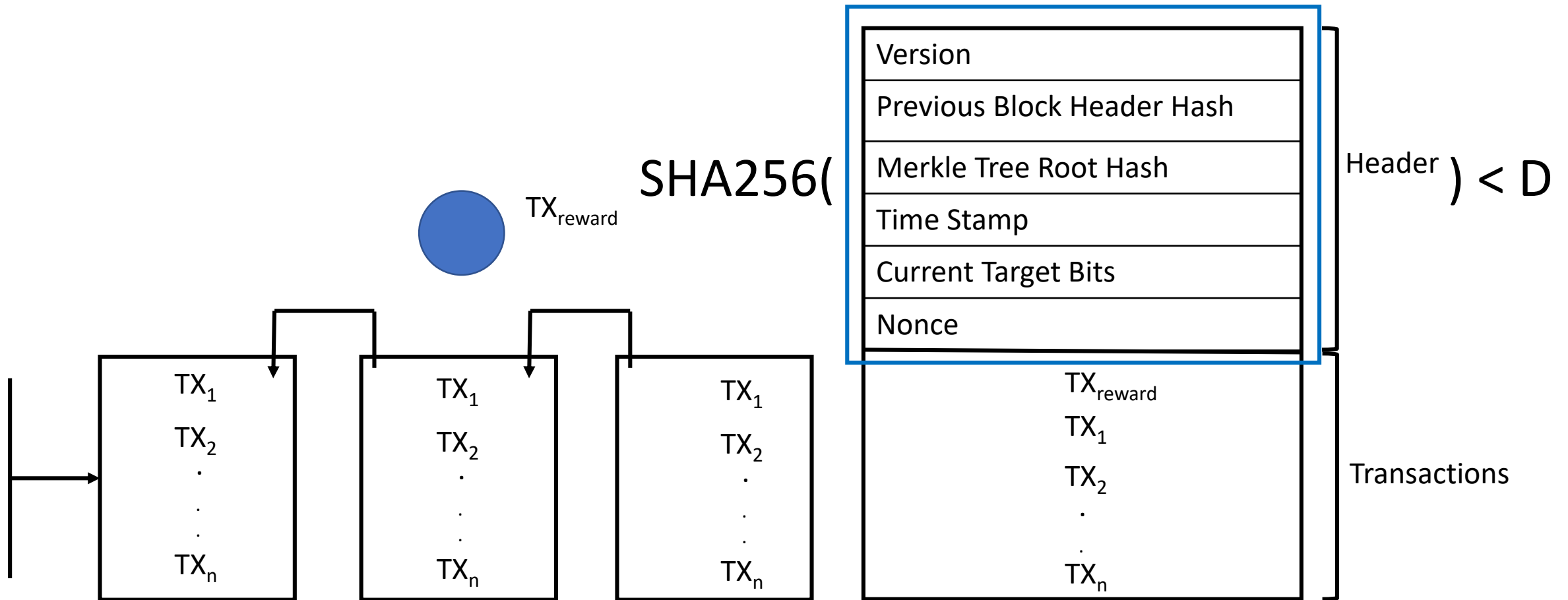
Version
Previous Block Header Hash
Merkle Tree Root Hash
Time Stamp
Current Target Bits
Nonce

Header) < D

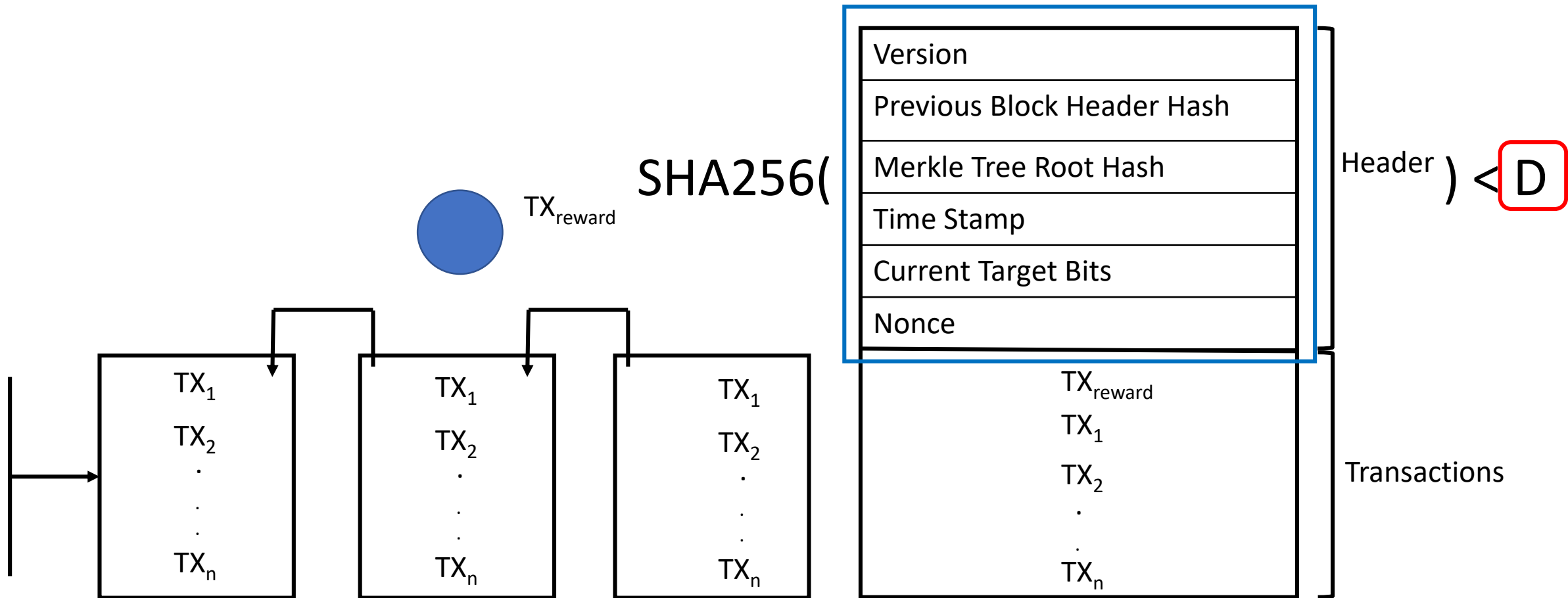


Transactions

Mining Details

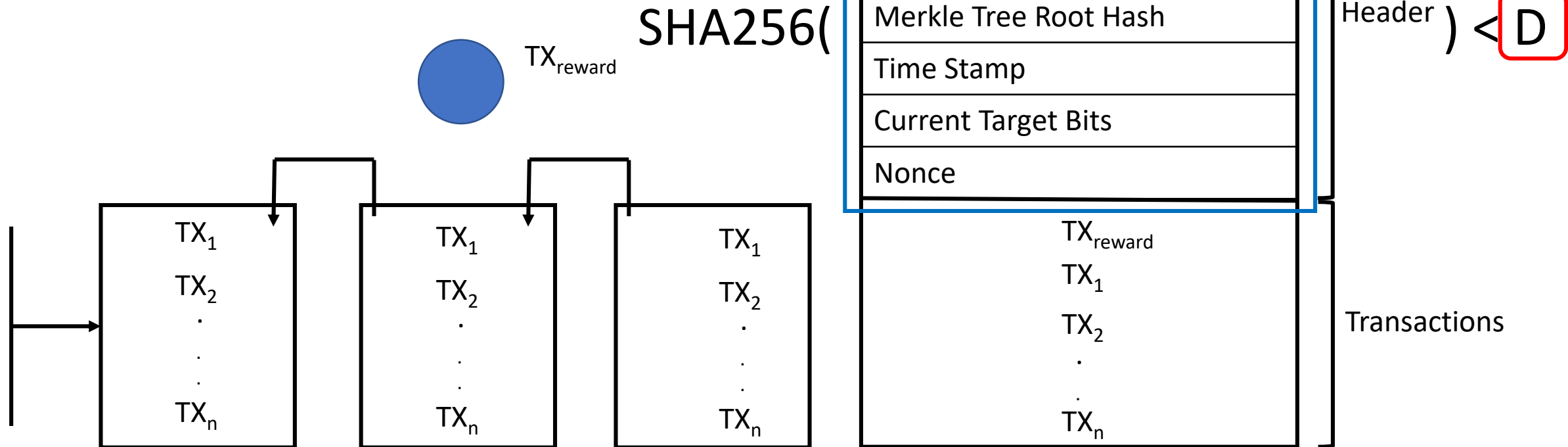


Mining Details



Mining Details

- D: dynamically adjusted difficulty



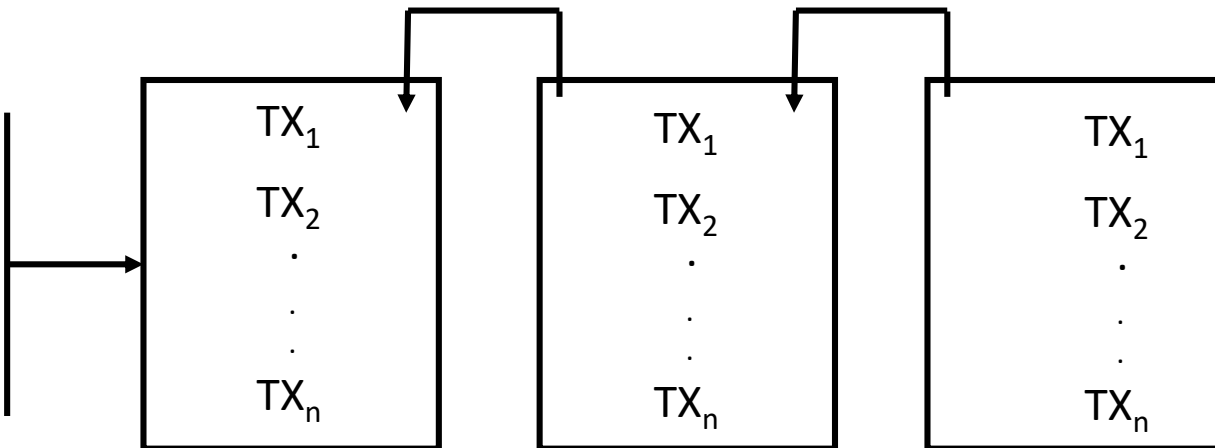
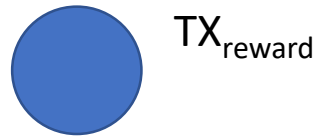
Mining Details

- D: dynamically adjusted difficulty

256 bits



SHA256(



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

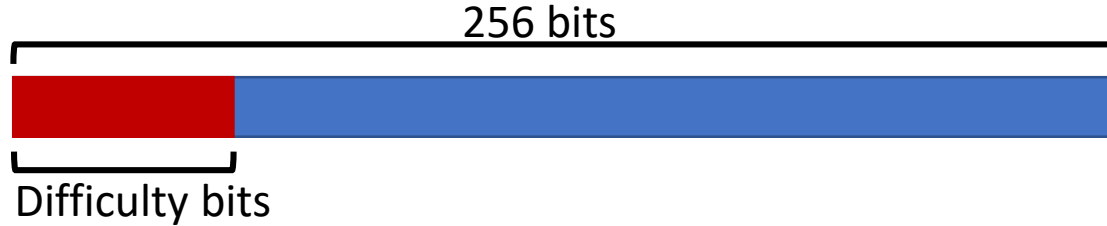
Header) < D

TX_reward
TX_1
TX_2
...
TX_n

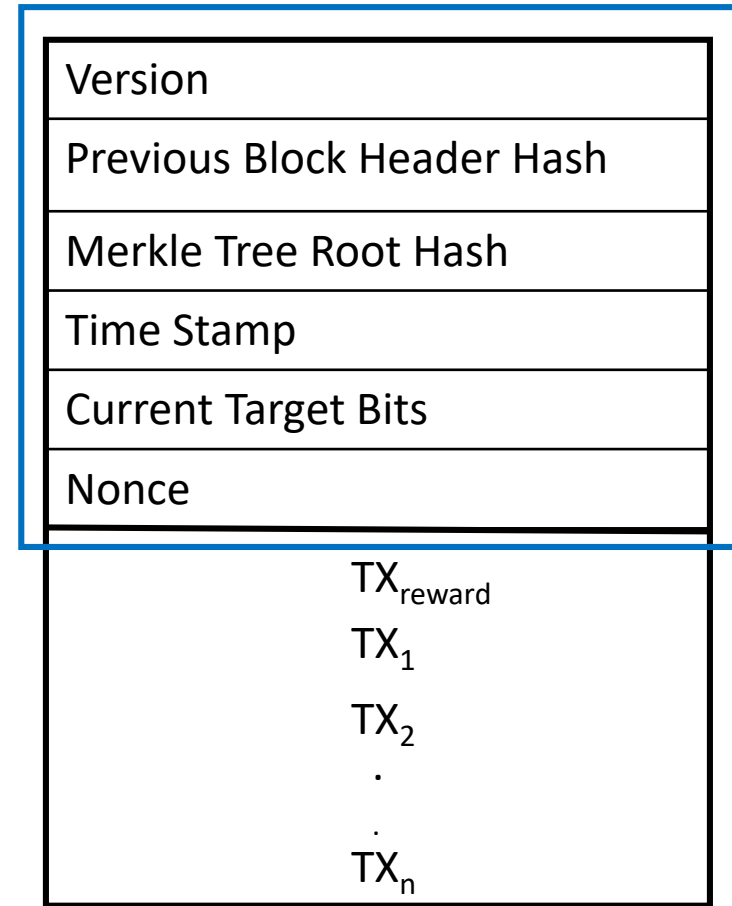
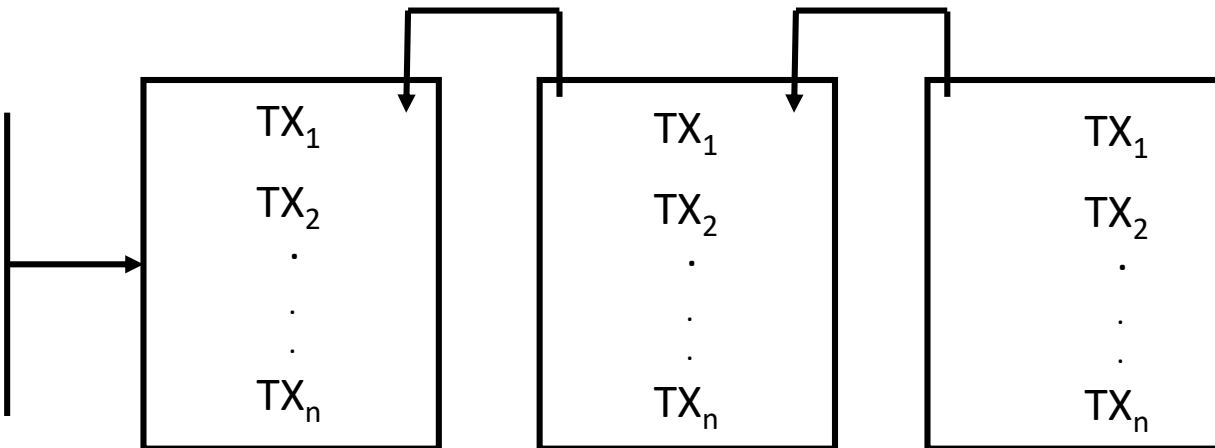
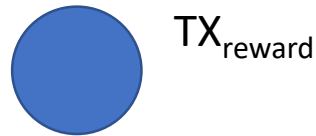
Transactions

Mining Details

- D: dynamically adjusted difficulty



SHA256(



Header) < D

Transactions

Mining Details

- D: dynamically adjusted difficulty

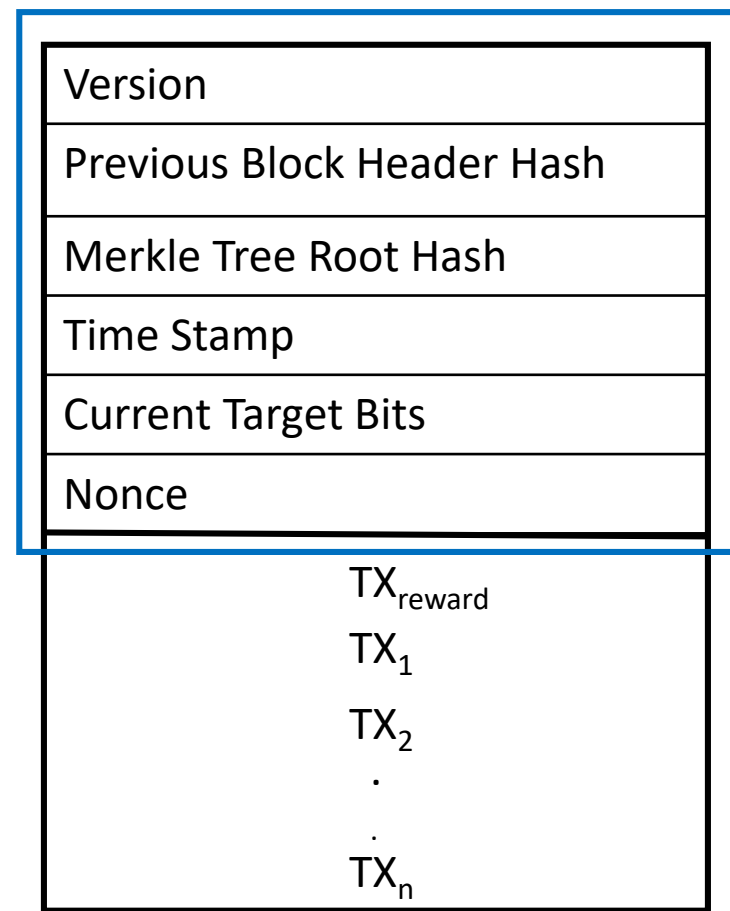
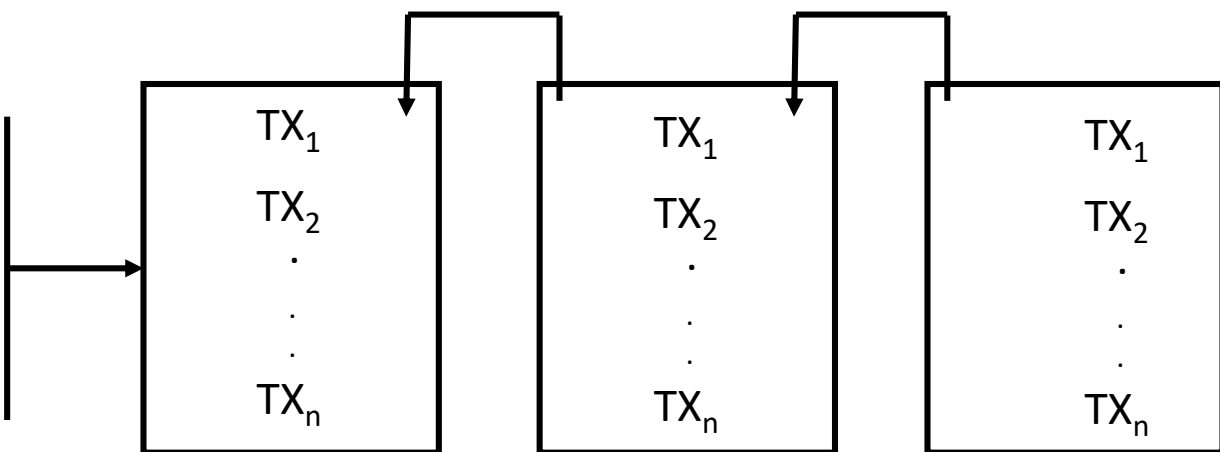
256 bits



Difficulty bits

- Difficulty is adjusted every 2016 blocks (almost 2 weeks)

SHA256(



Header) < D

Transactions

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

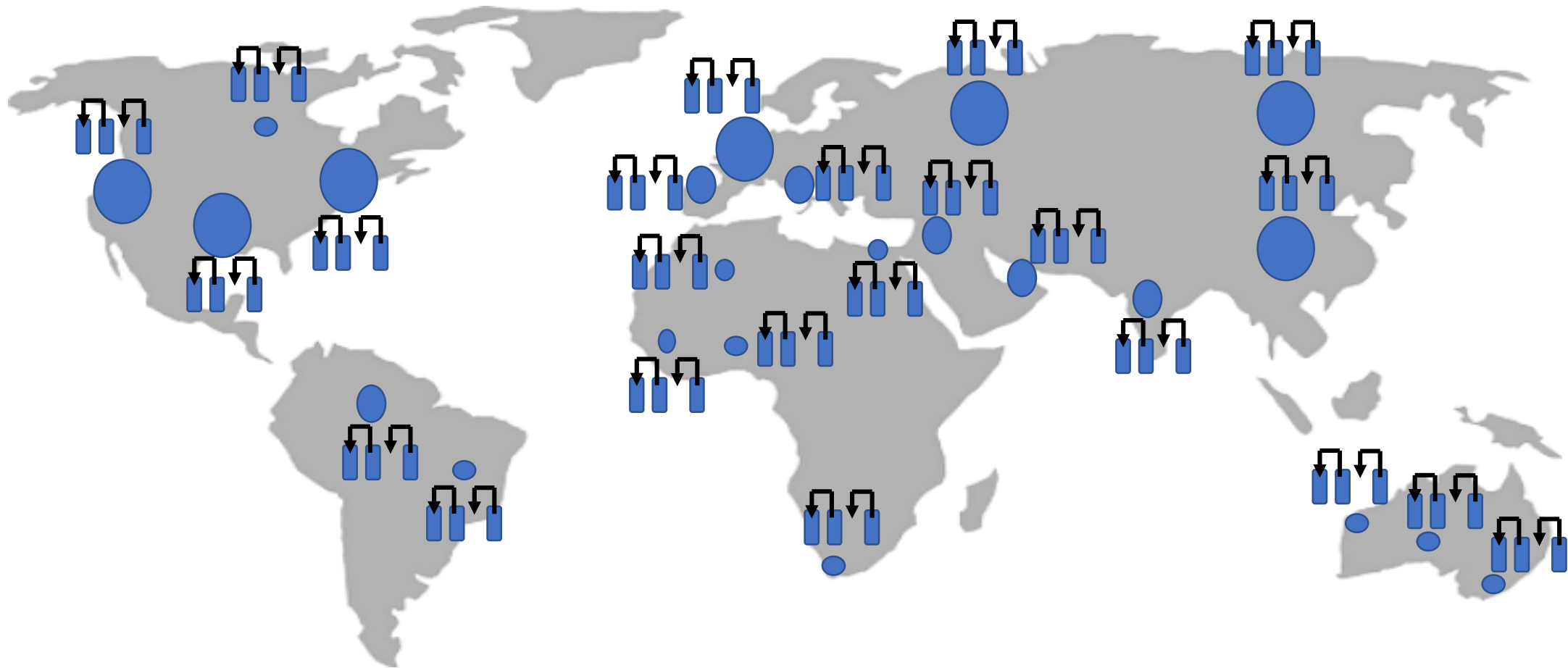
Difficulty

- 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
- $\text{New_difficulty} = \text{old_difficulty} * \frac{\text{expected}}{\text{actual}}$

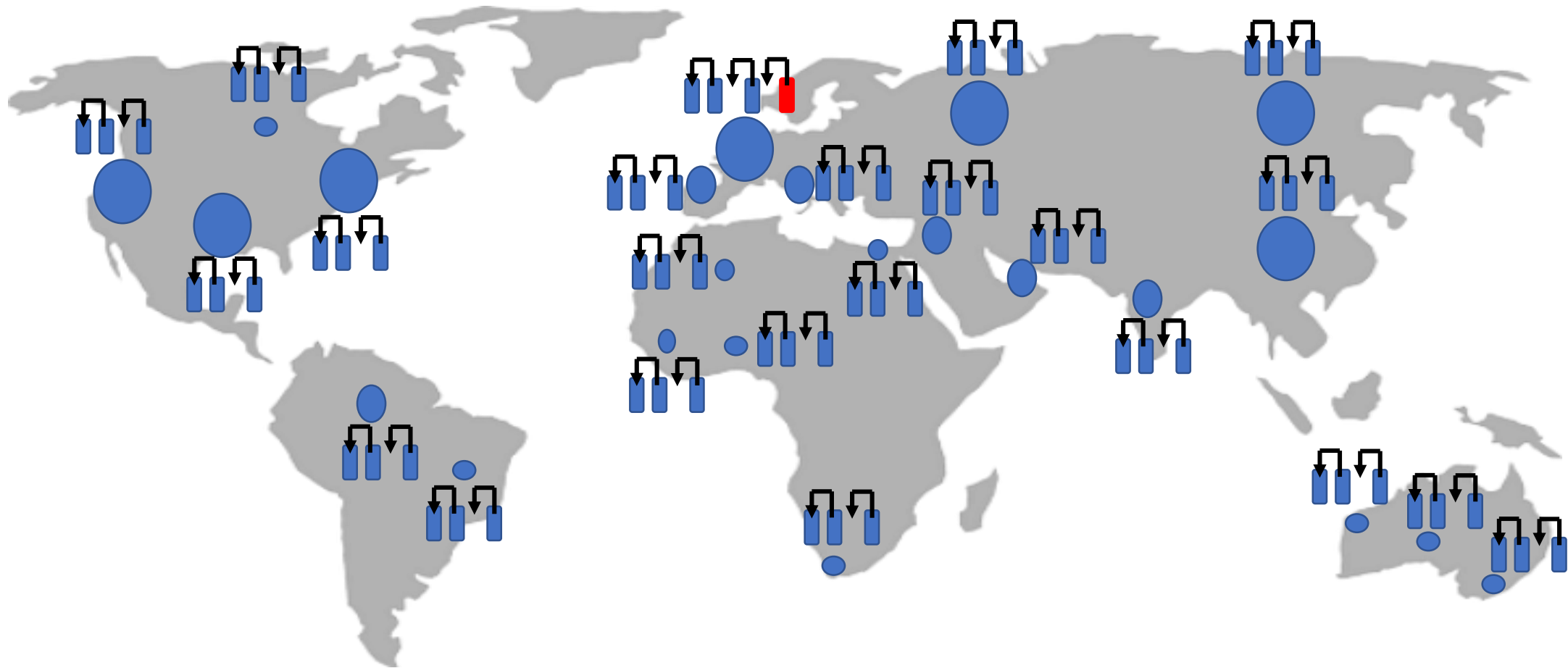
Difficulty

- 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
- $\text{New_difficulty} = \text{old_difficulty} * \frac{\text{expected}}{\text{actual}}$
- Difficulty decreases if $\text{actual} > \text{expected}$, otherwise, increases

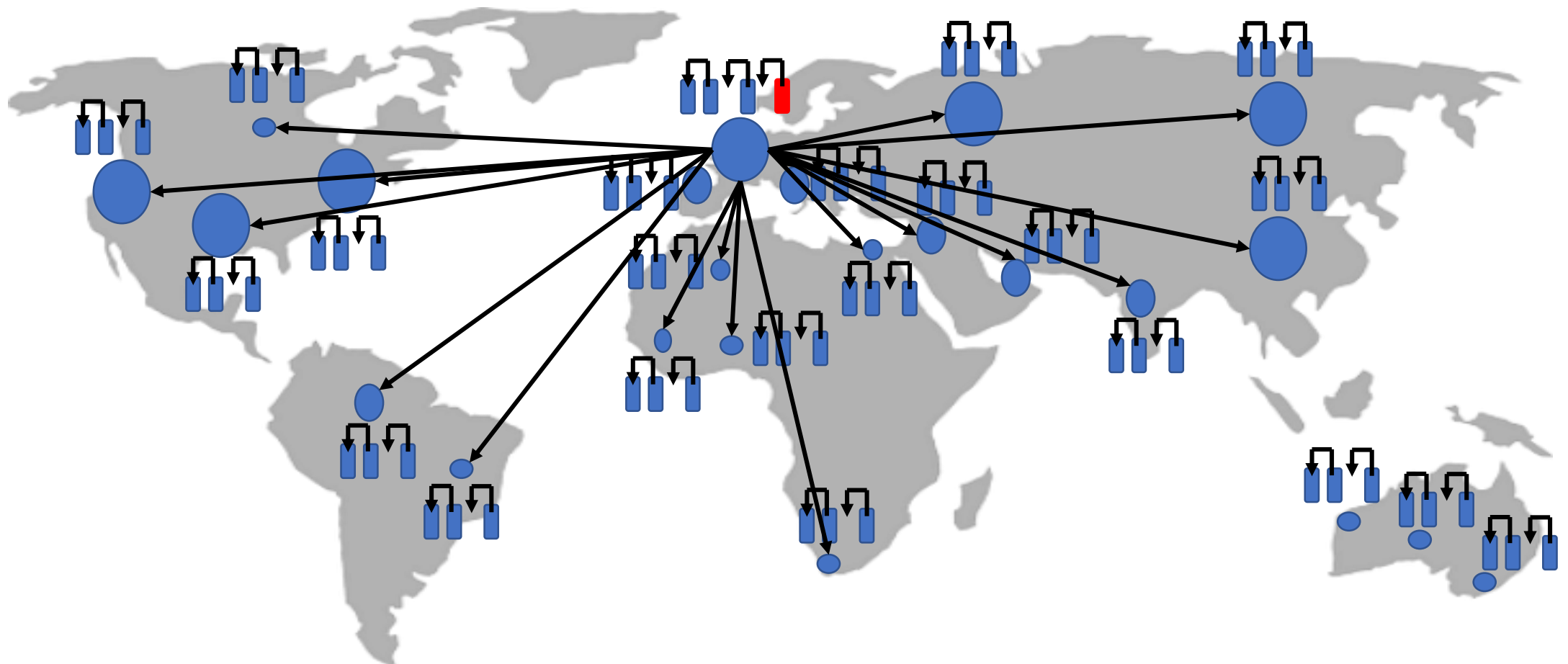
Mining Big Picture



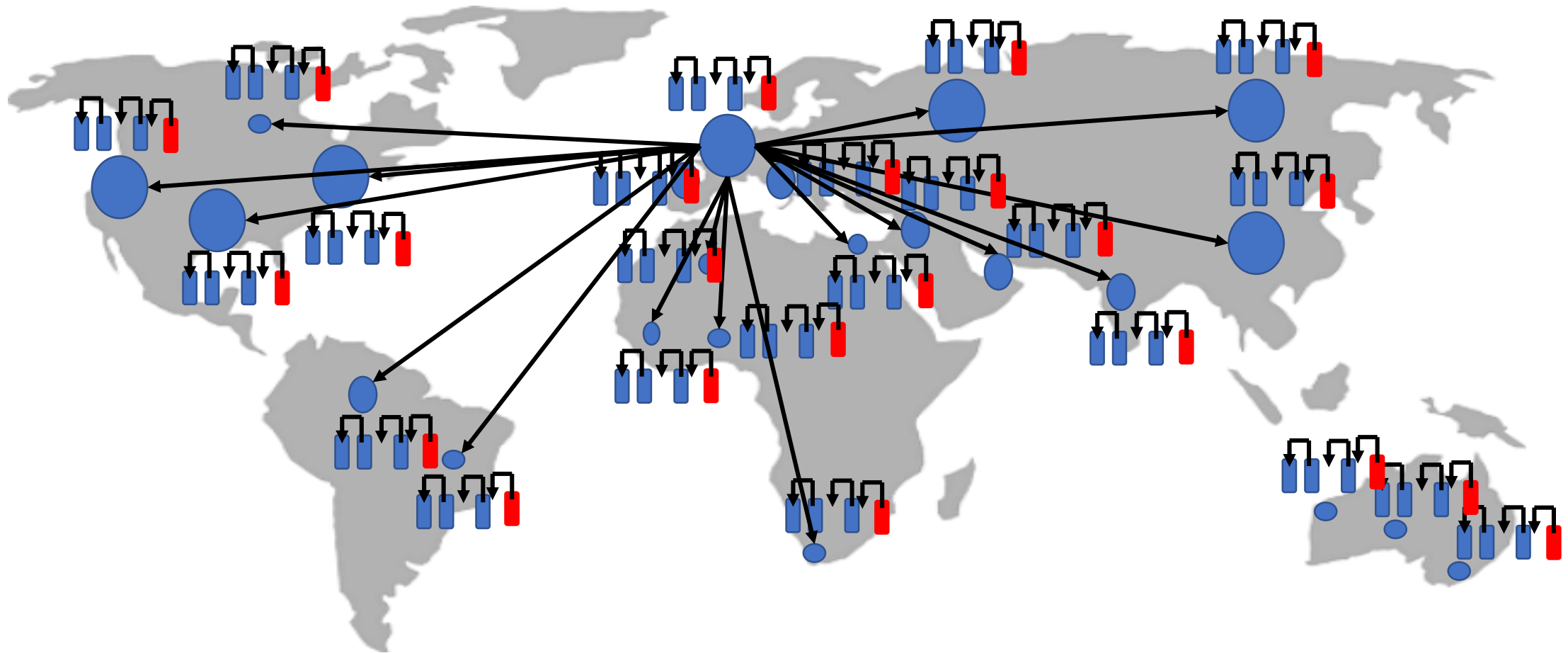
Mining Big Picture



Mining Big Picture



Mining Big Picture



Mining Details

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

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 $\text{SHA256}(\text{block}) < \text{Difficulty}$
- 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 $\text{SHA256}(\text{block}) < \text{Difficulty}$
- 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 $\text{SHA256}(\text{block}) < \text{Difficulty}$
- 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 $\text{SHA256}(\text{block}) < \text{Difficulty}$
- 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
- 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 $\text{SHA256}(\text{block}) < \text{Difficulty}$
- 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
- 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 $\text{SHA256}(\text{block}) < \text{Difficulty}$

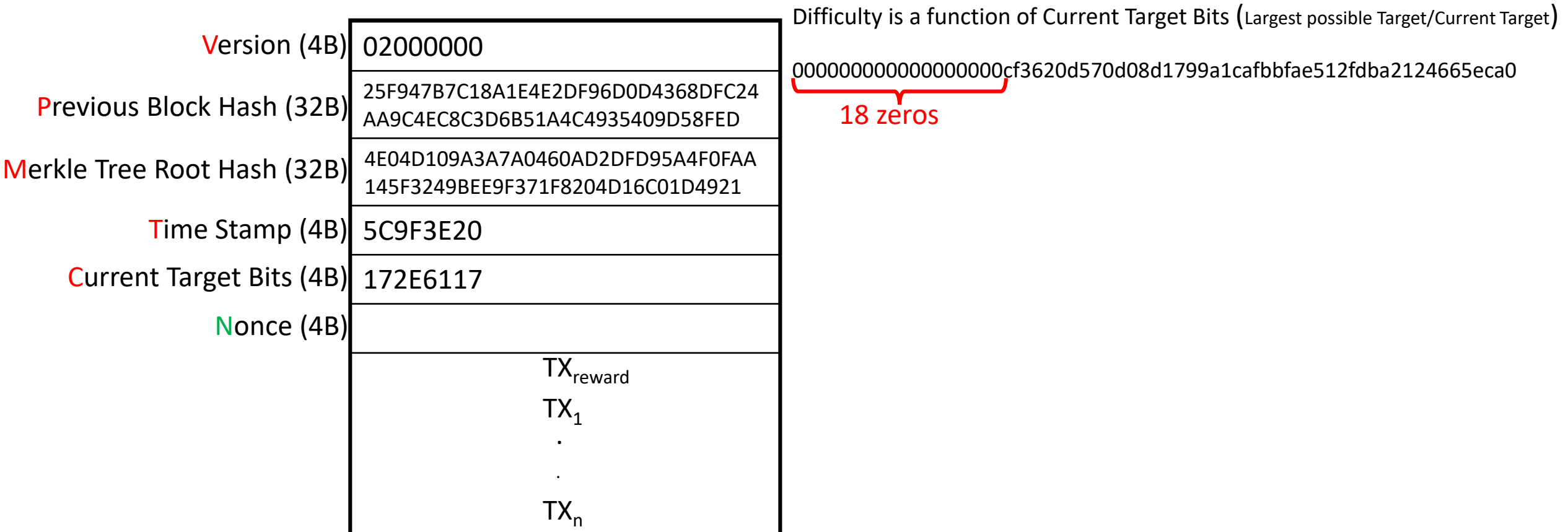
Mining Details

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

Version (4B)	02000000
Previous Block Hash (32B)	25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED
Merkle Tree Root Hash (32B)	4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921
Time Stamp (4B)	5C9F3E20
Current Target Bits (4B)	172E6117
Nonce (4B)	
	TX _{reward} TX ₁ · · TX _n

Mining Details

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



Mining Details

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

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

Version (4B)	02000000
Previous Block Hash (32B)	25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED
Merkle Tree Root Hash (32B)	4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921
Time Stamp (4B)	5C9F3E20
Current Target Bits (4B)	172E6117
Nonce (4B)	
	TX _{reward} TX ₁ · · TX _n

00000000000000000000cf3620d570d08d1799a1cafbbfae512fdb2124665eca0

18 zeros

$\text{SHA256}(\text{V,P,M,T,C,O}) =$
BD72804EE251889F9013C100767999B57E92EC5B6ADBDBF64F2DF1B032429C72

Mining Details

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

Version (4B)	02000000
Previous Block Hash (32B)	25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED
Merkle Tree Root Hash (32B)	4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921
Time Stamp (4B)	5C9F3E20
Current Target Bits (4B)	172E6117
Nonce (4B)	
	TX _{reward} TX ₁ · · TX _n

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

00000000000000000000cf3620d570d08d1799a1cafbbfae512fdb2124665eca0
18 zeros

$\text{SHA256}(V, P, M, T, C, O) =$
 BD72804EE251889F9013C100767999B57E92EC5B6ADBDBF64F2DF1B0324



Mining Details

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

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


Version (4B)	02000000
Previous Block Hash (32B)	25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED
Merkle Tree Root Hash (32B)	4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921
Time Stamp (4B)	5C9F3E20
Current Target Bits (4B)	172E6117
Nonce (4B)	
	TX _{reward} TX ₁ · · TX _n

00000000000000000000cf3620d570d08d1799a1cafbbfae512fdb2124665eca0

18 zeros

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

SHA256(V,P,M,T,C,1) =
DF64342507E785FDC0D4C776D7142BB2BC6467F09E0040A3E9F65E38872A45D8



Mining Details

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

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


Version (4B)	02000000
Previous Block Hash (32B)	25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED
Merkle Tree Root Hash (32B)	4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921
Time Stamp (4B)	5C9F3E20
Current Target Bits (4B)	172E6117
Nonce (4B)	
	TX _{reward} TX ₁ . . TX _n

00000000000000000000cf3620d570d08d1799a1cafbbfae512fdb2124665eca0

18 zeros

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

SHA256(V,P,M,T,C,1) =
DF64342507E785FDC0D4C776D7142BB2BC6467F09E0040A3E9F65E38872



Mining Details

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

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

Version (4B)	02000000
Previous Block Hash (32B)	25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED
Merkle Tree Root Hash (32B)	4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921
Time Stamp (4B)	5C9F3E20
Current Target Bits (4B)	172E6117
Nonce (4B)	
	TX _{reward} TX ₁ . . TX _n

00000000000000000000cf3620d570d08d1799a1cafbbfae512fdb2124665eca0

18 zeros

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

SHA256(V,P,M,T,C,1) =
DF64342507E785FDC0D4C776D7142BB2BC6467F09E0040A3E9F65E38872 🙄

SHA256(V,P,M,T,C,2) =
0000000CC7F94221B95F4E606E037D31C10417435DEE60A61C627B64324590FE

Mining Details

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

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

Version (4B)	02000000
Previous Block Hash (32B)	25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED
Merkle Tree Root Hash (32B)	4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921
Time Stamp (4B)	5C9F3E20
Current Target Bits (4B)	172E6117
Nonce (4B)	
	TX _{reward} TX ₁ . . TX _n

00000000000000000000cf3620d570d08d1799a1cafbbfae512fdb2124665eca0


18 zeros

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

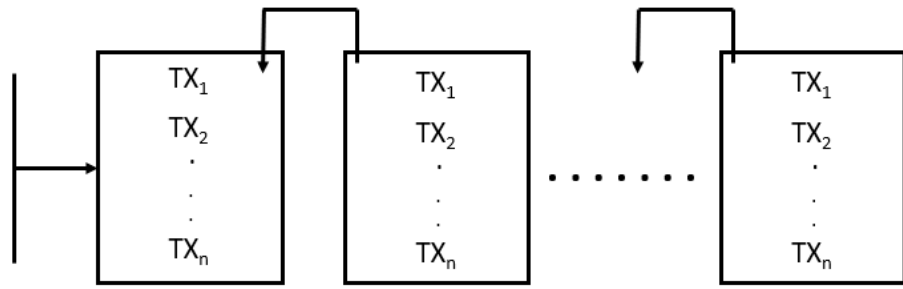
SHA256(V,P,M,T,C,1) =
DF64342507E785FDC0D4C776D7142BB2BC6467F09E0040A3E9F65E38872

SHA256(V,P,M,T,C,2) =
0000000CC7F94221B95F4E606E037D31C10417435DEE60A61C627B64324!

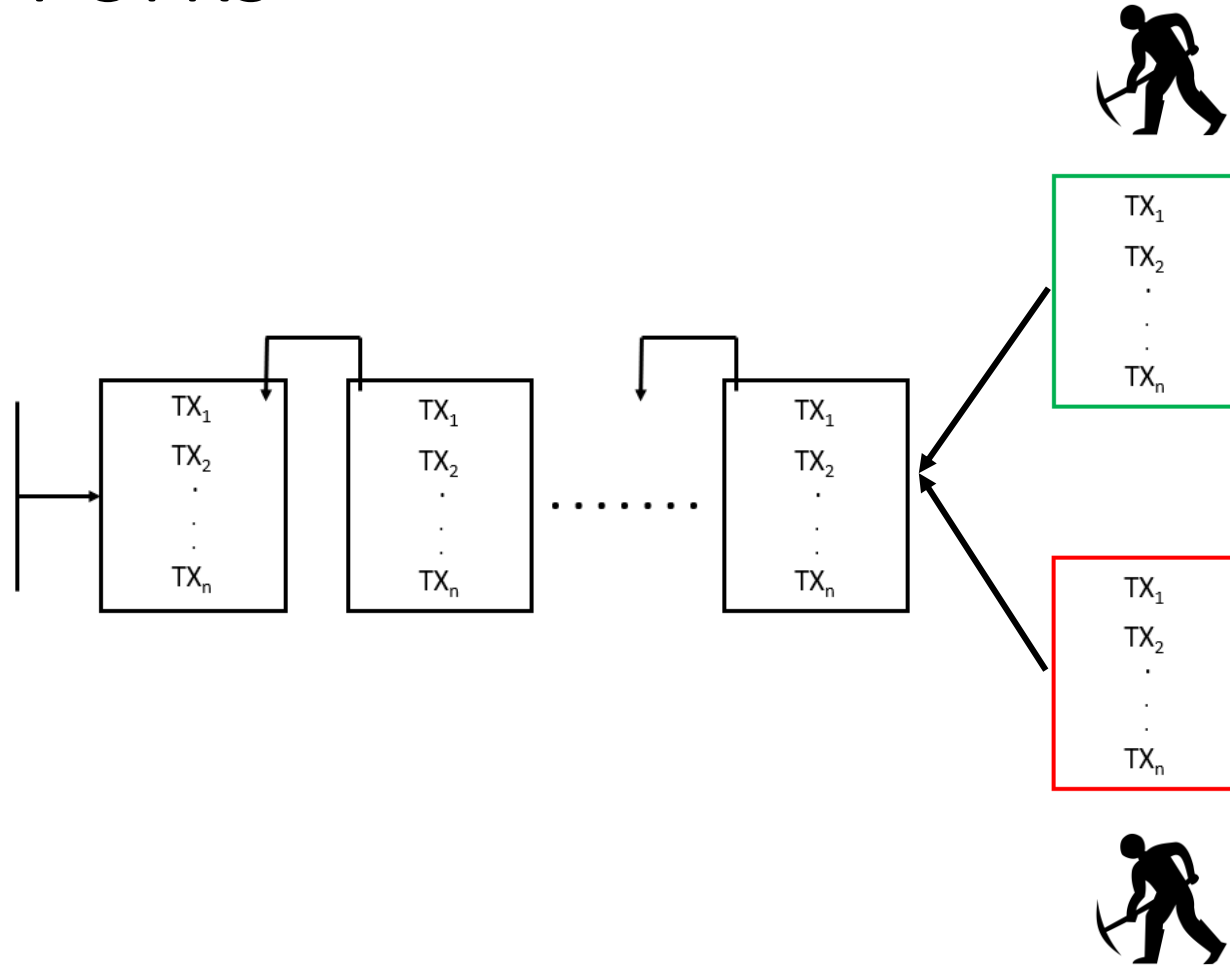
7 zeros



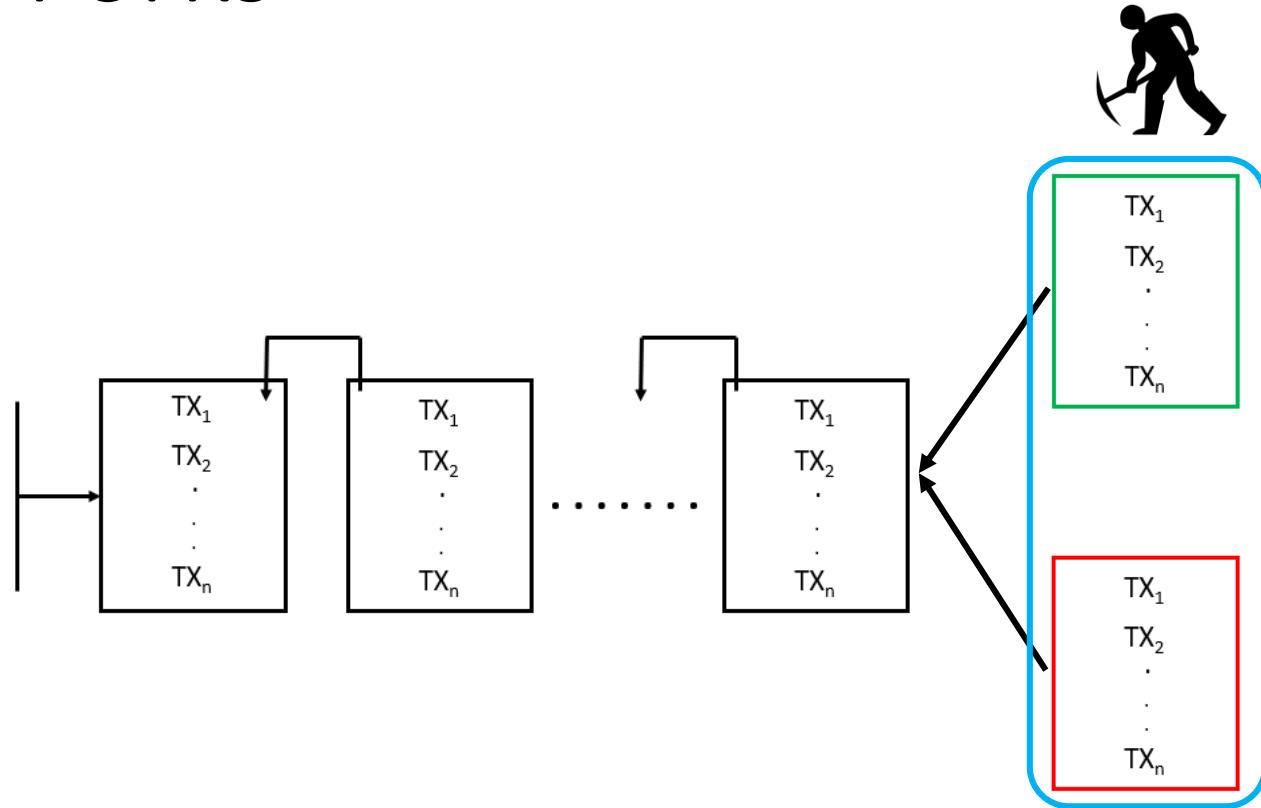
Forks



Forks

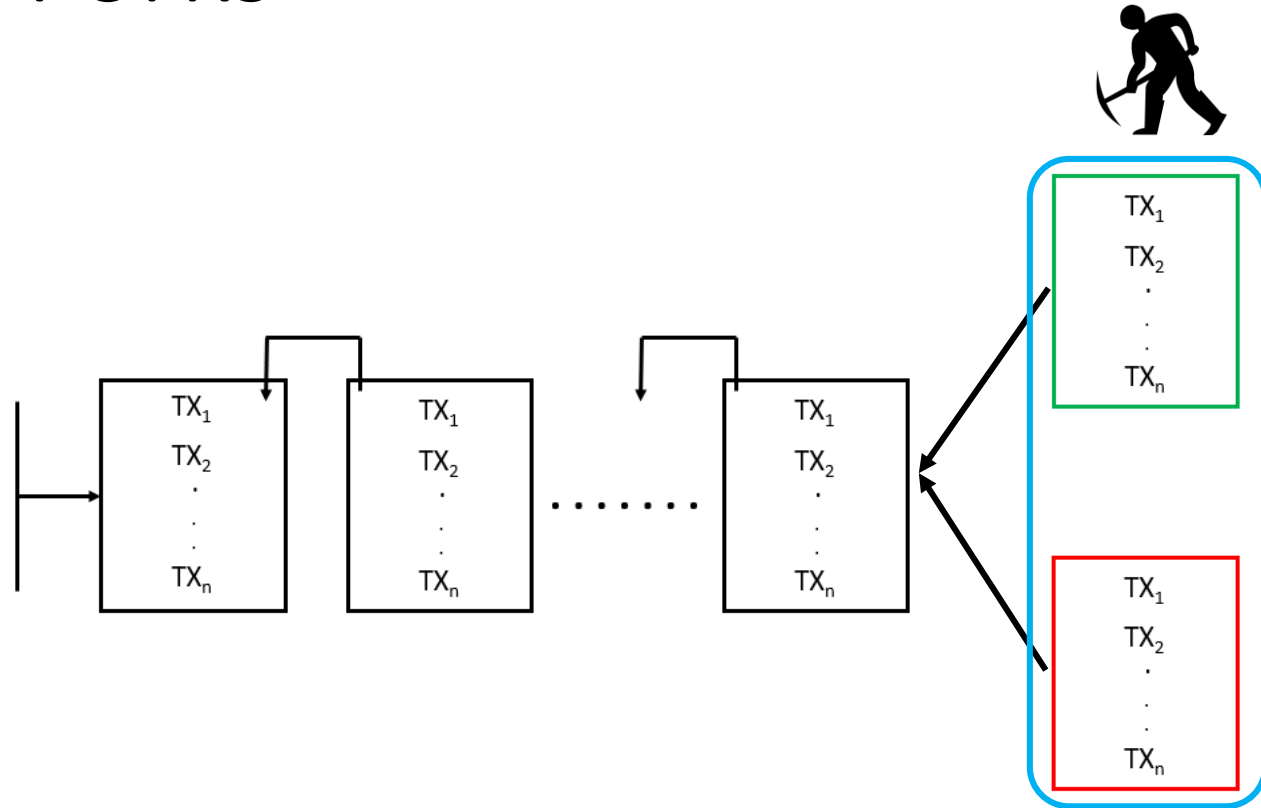


Forks



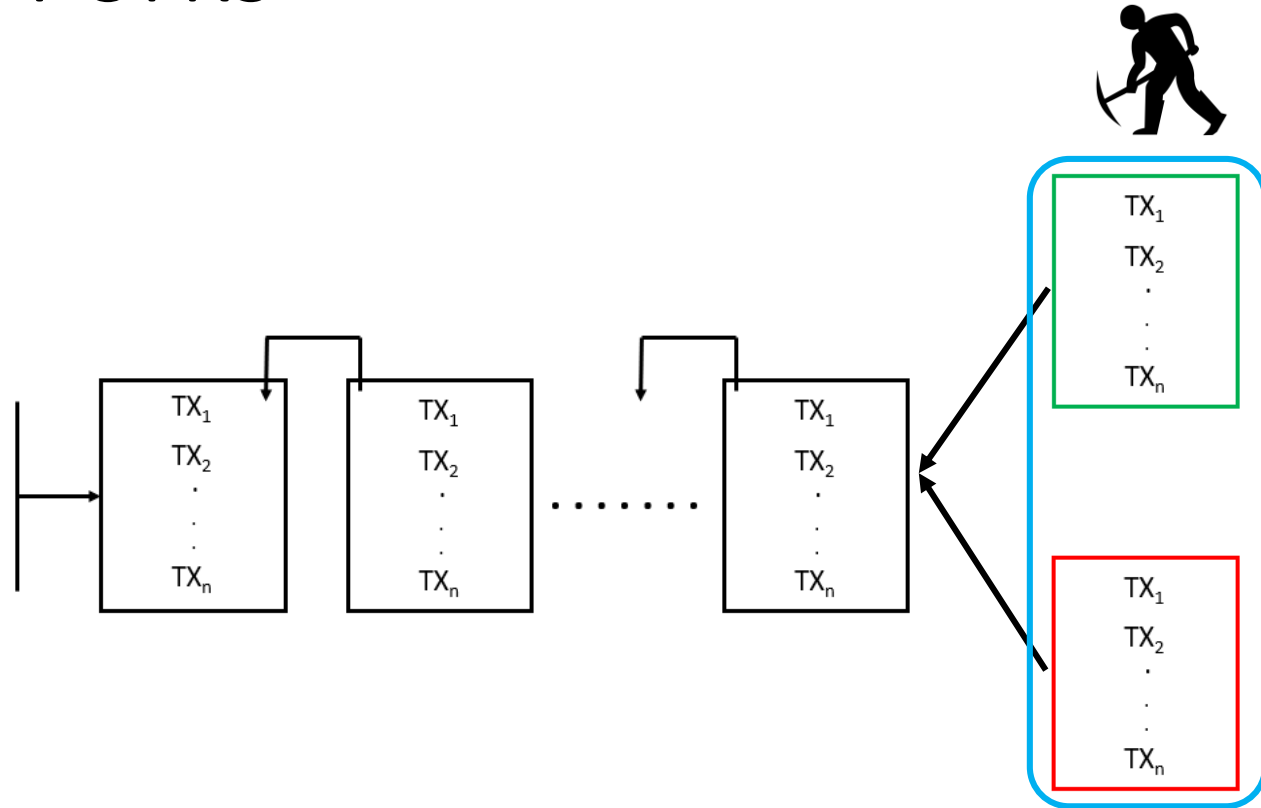
- Transactions in the forked blocks might have conflicts

Forks



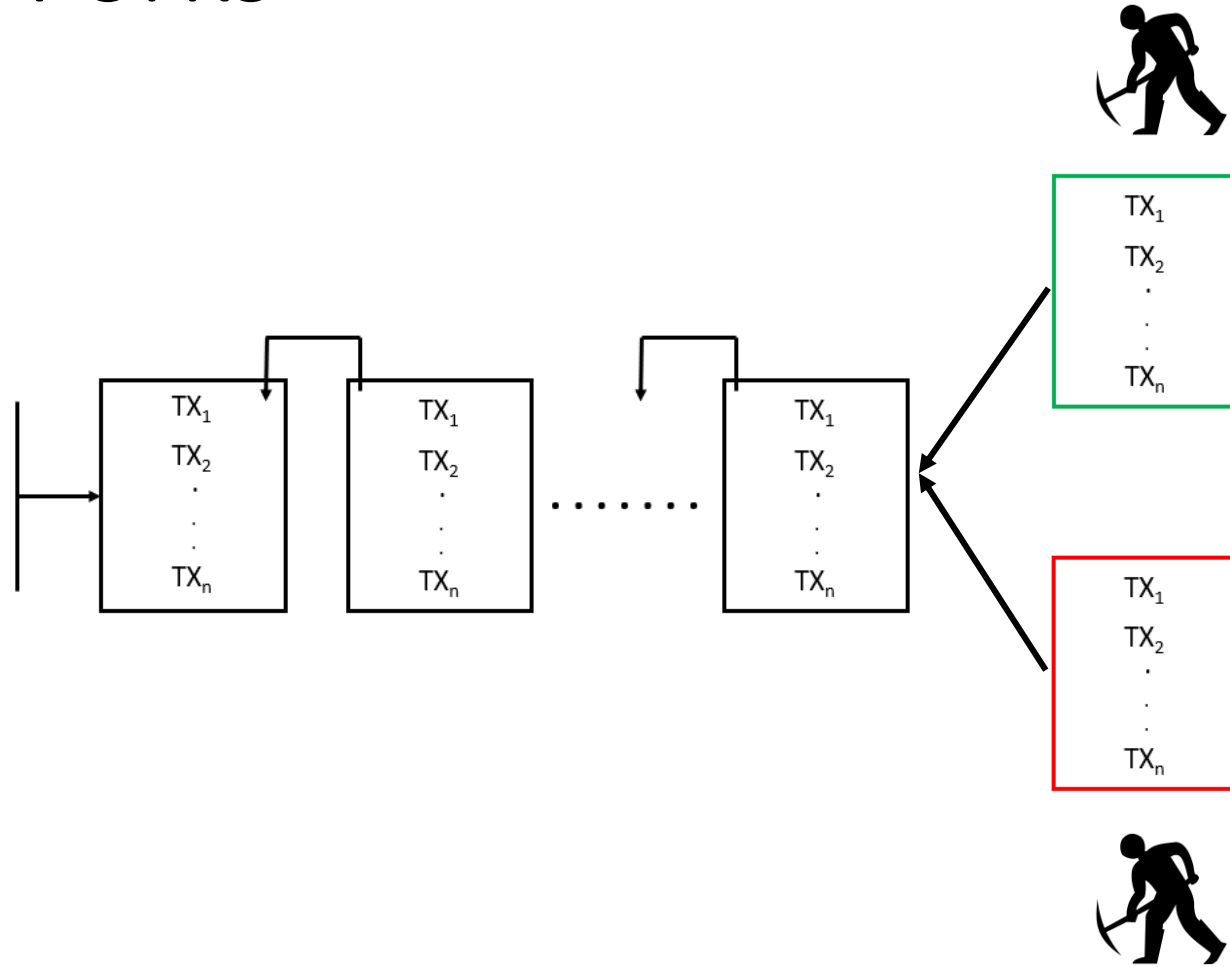
- Transactions in the forked blocks might have conflicts
- Could lead to double spending

Forks

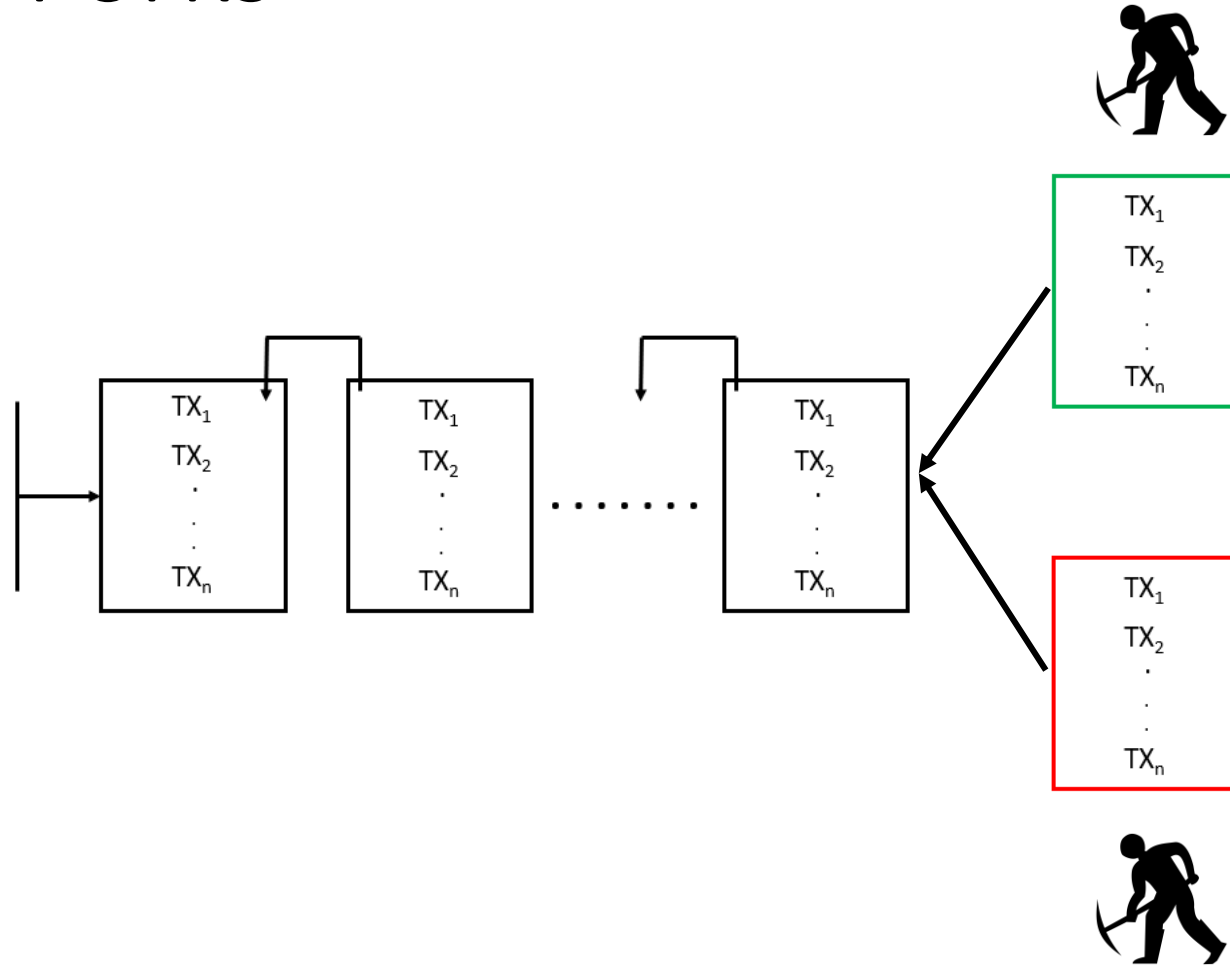


- Transactions in the forked blocks might have conflicts
- Could lead to double spending
- Forks have to be eliminated

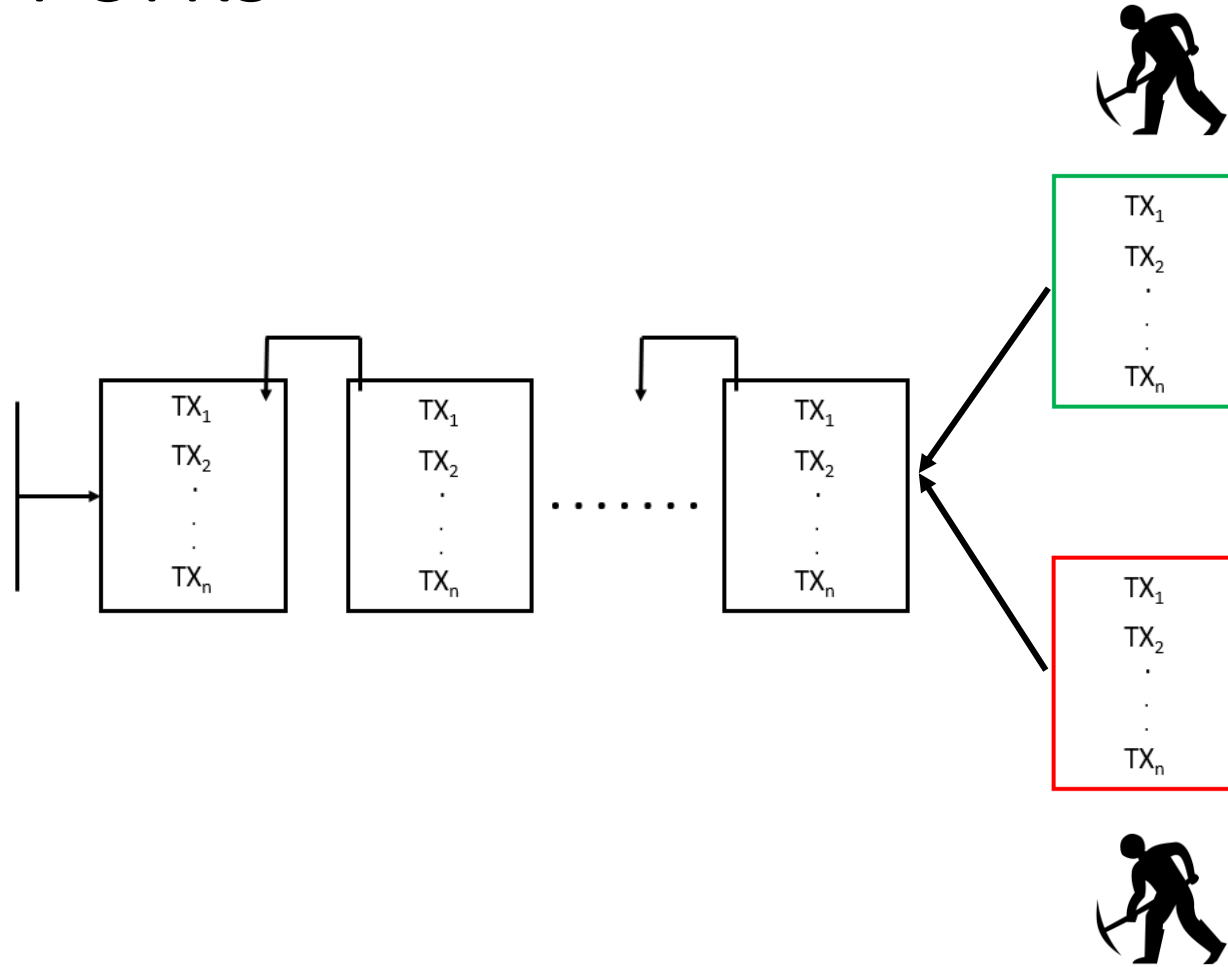
Forks



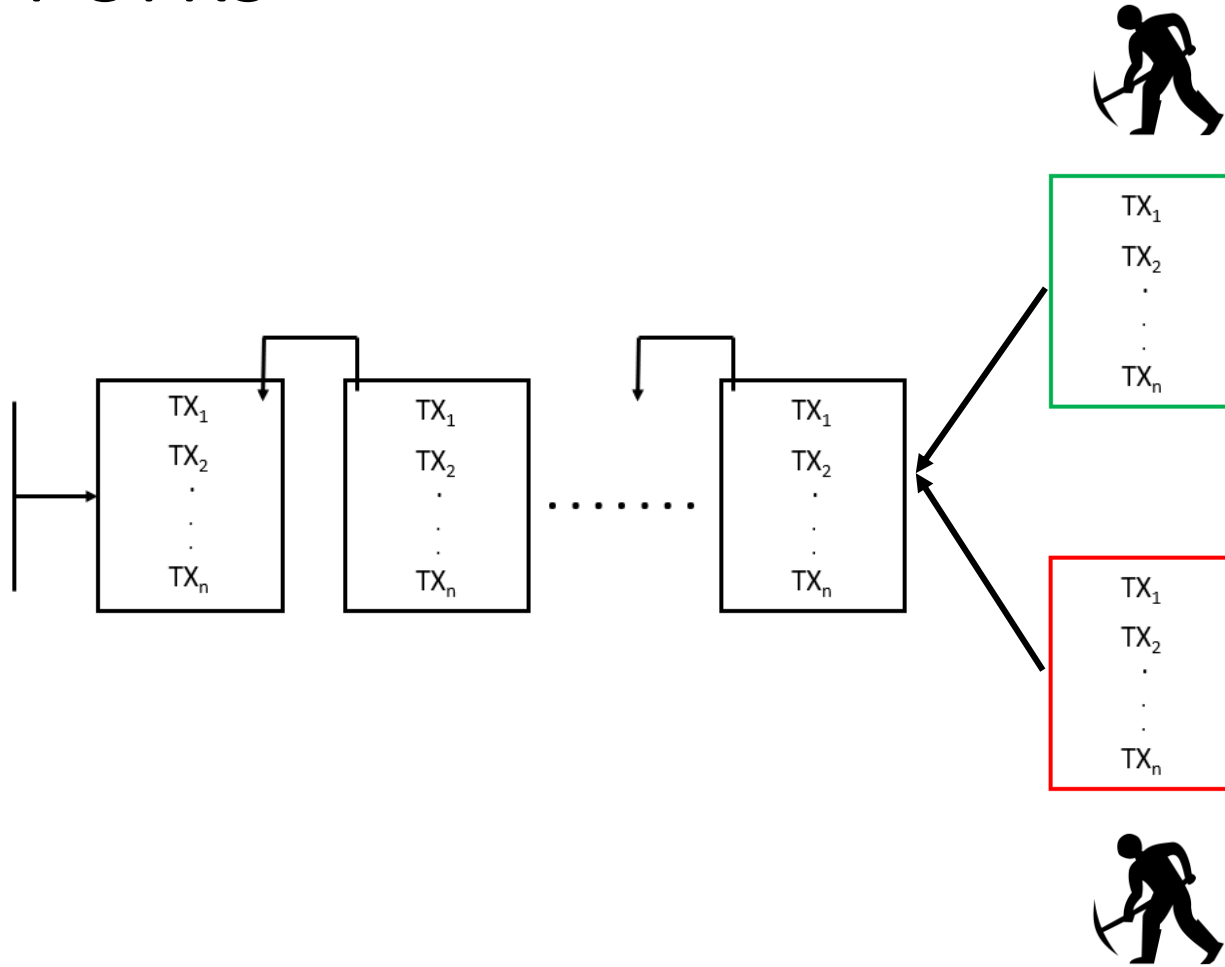
Forks



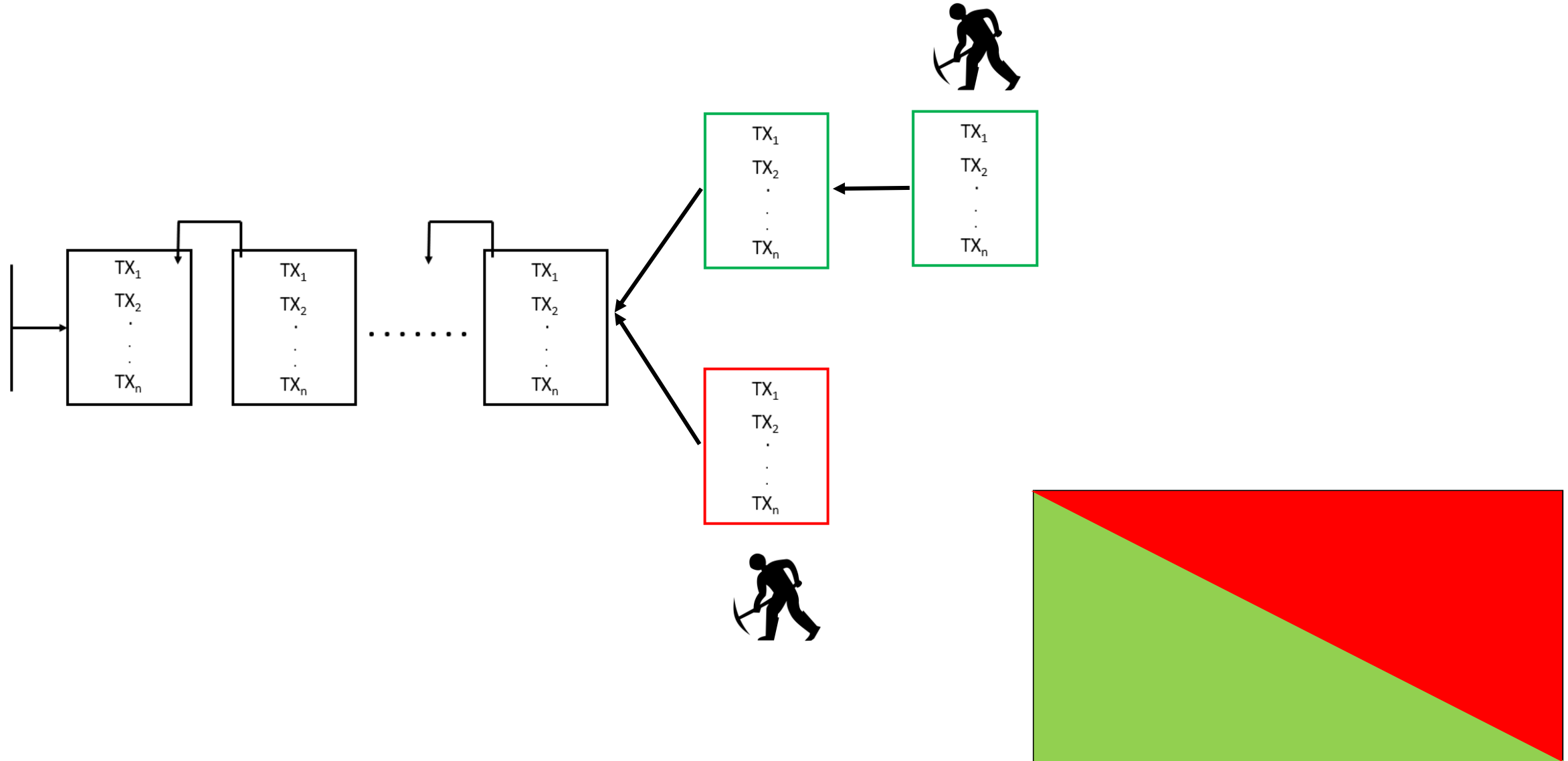
Forks



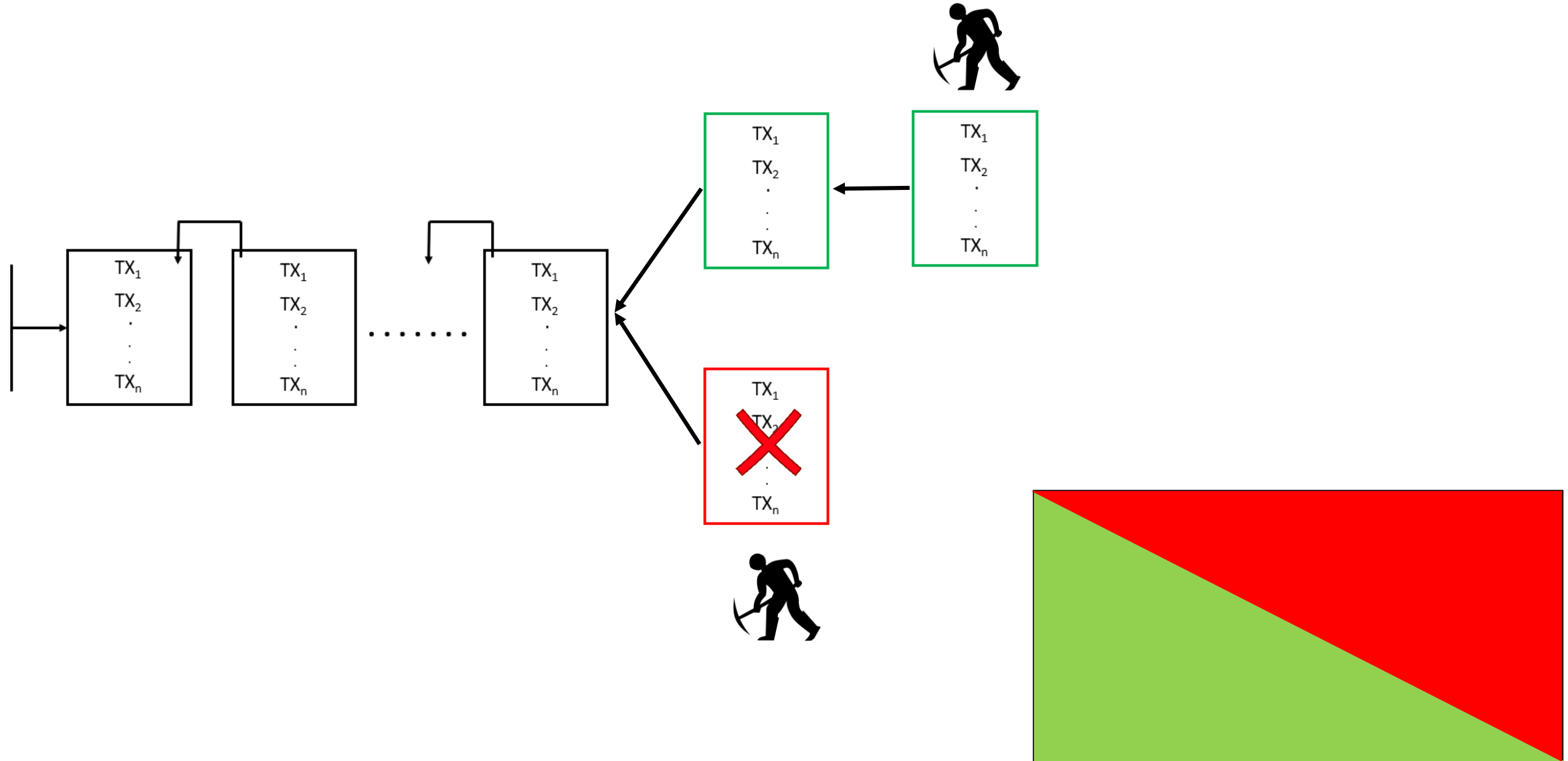
Forks



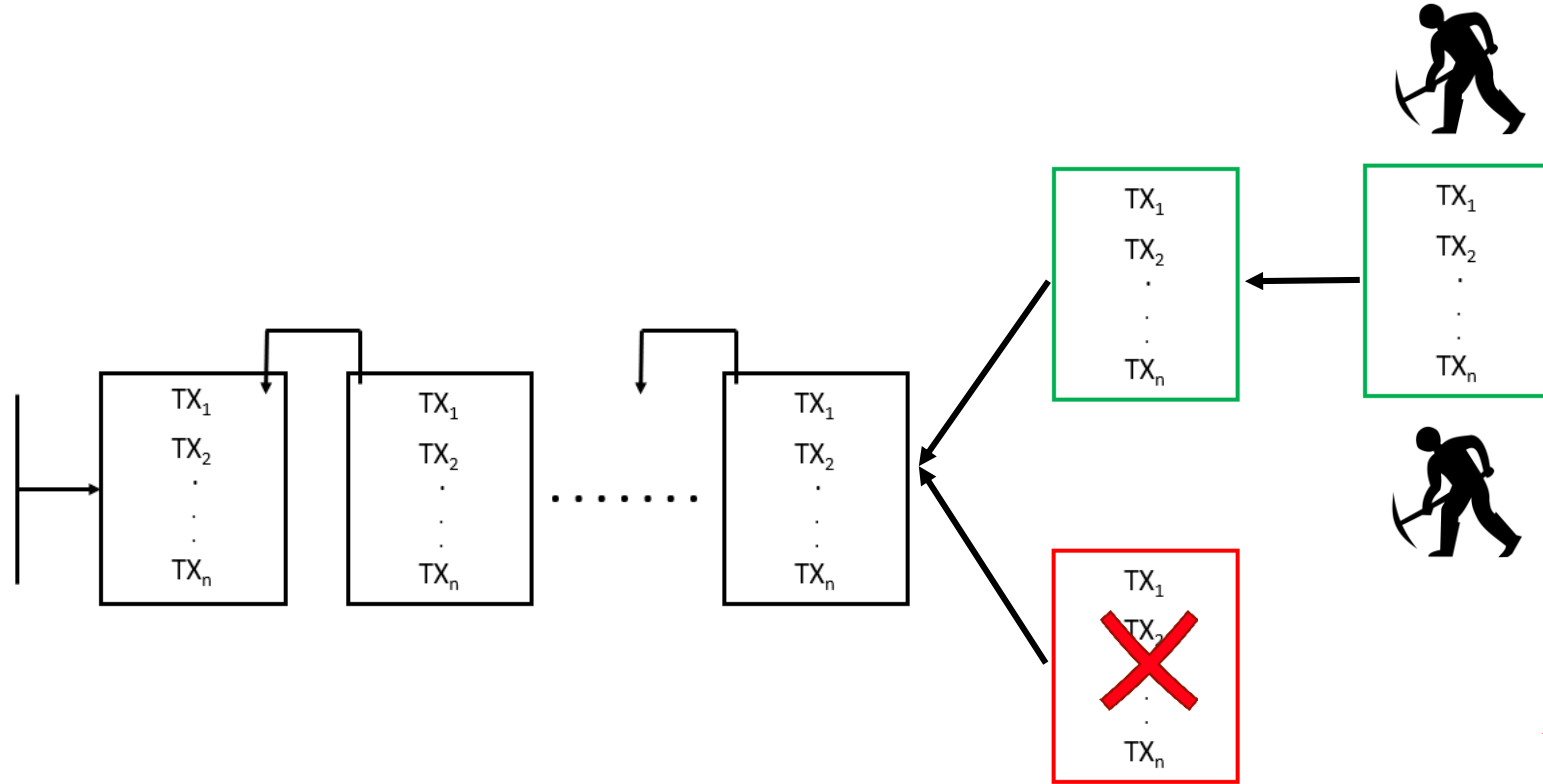
Forks



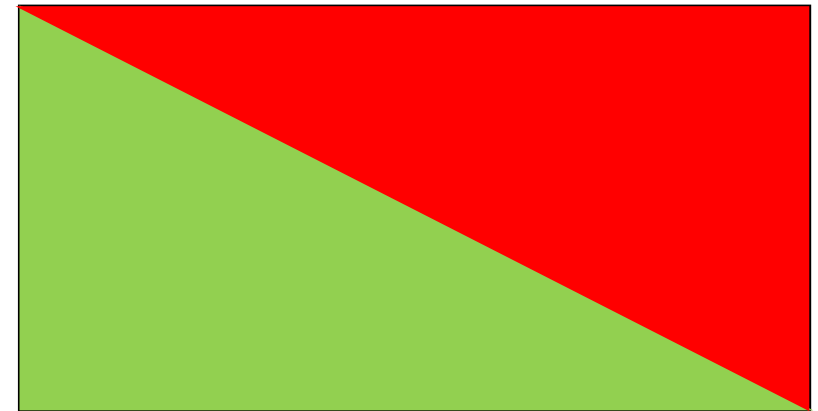
Forks



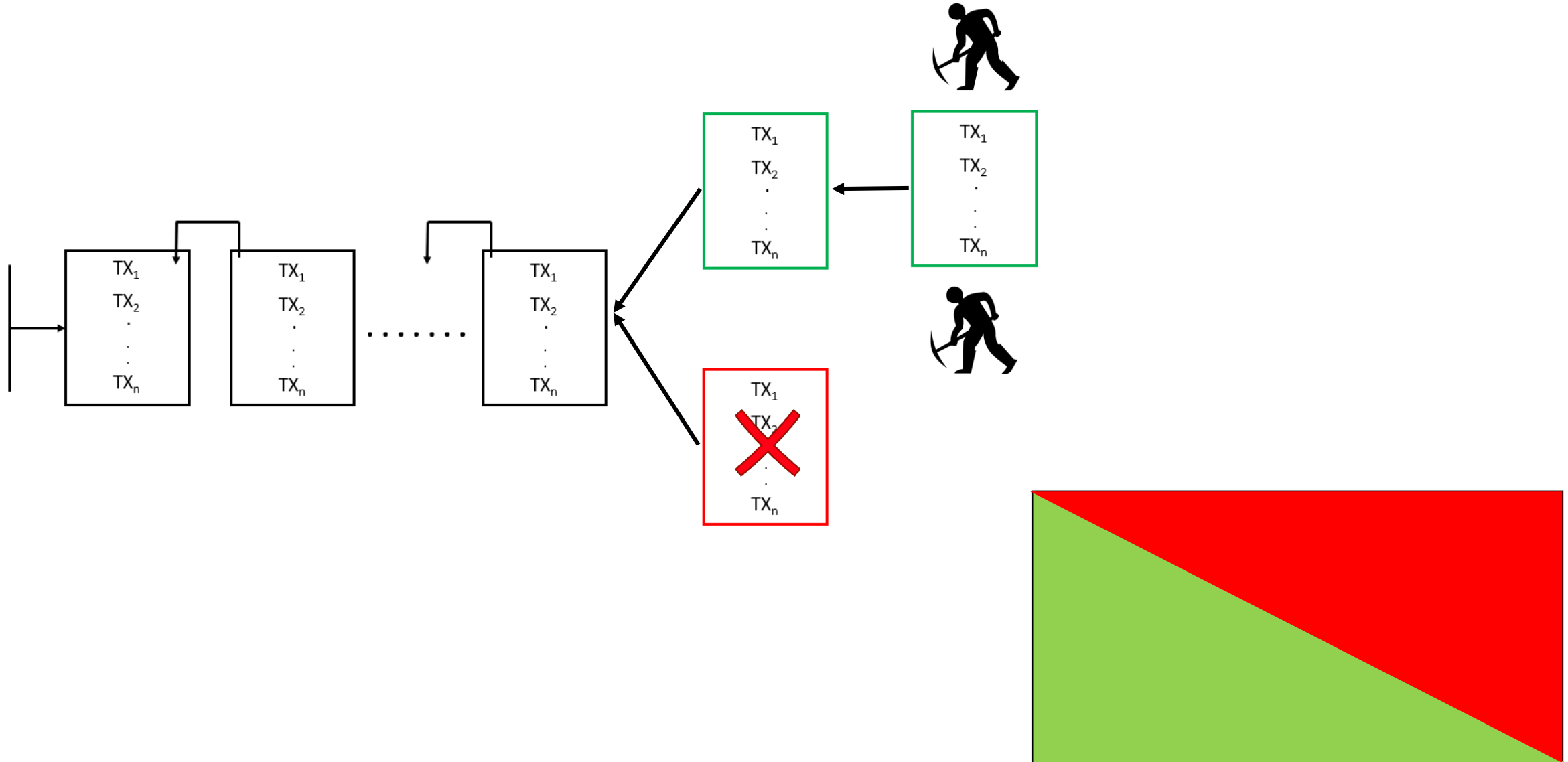
Forks



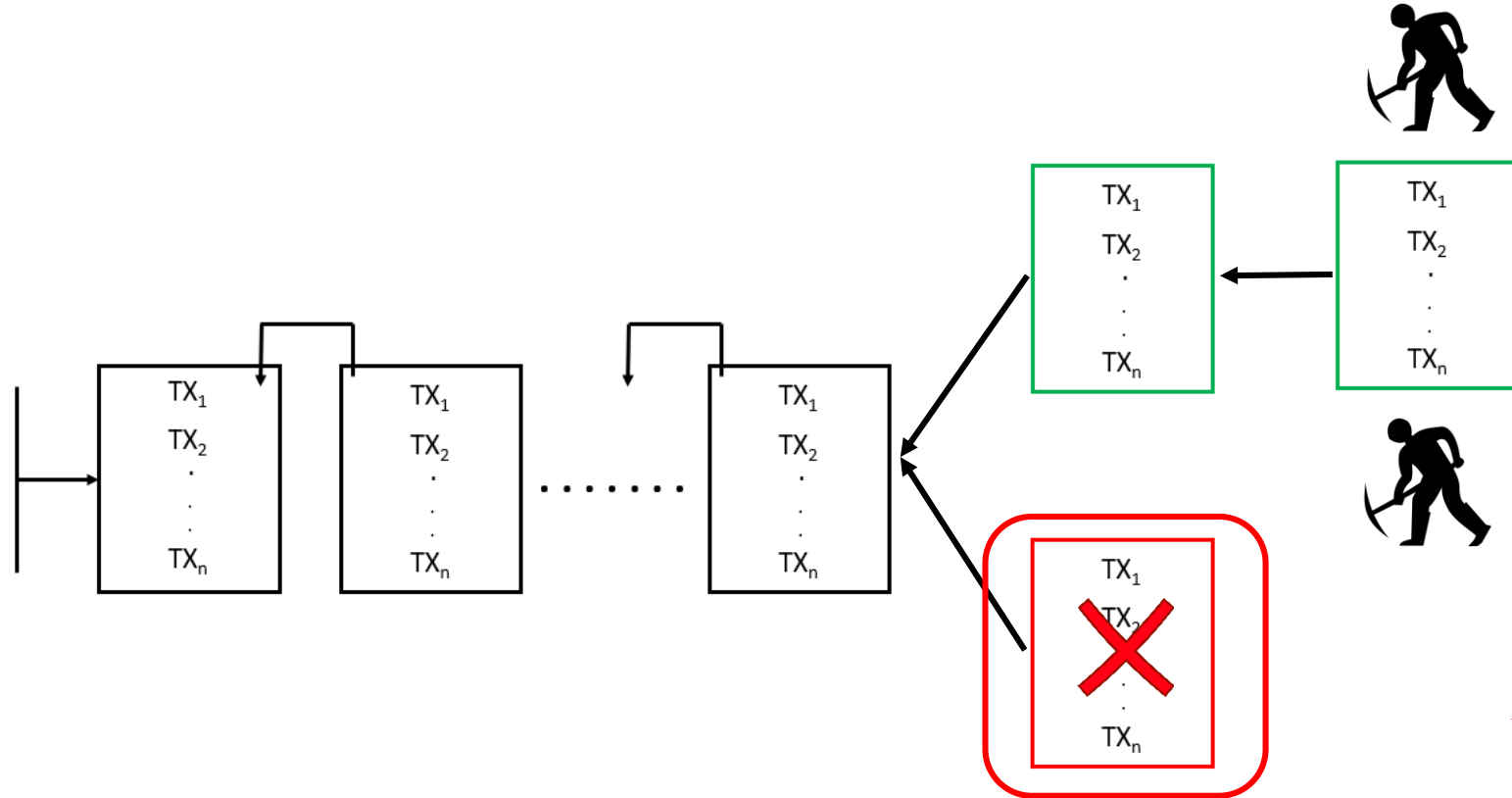
- Miners join the longest chain to resolve forks



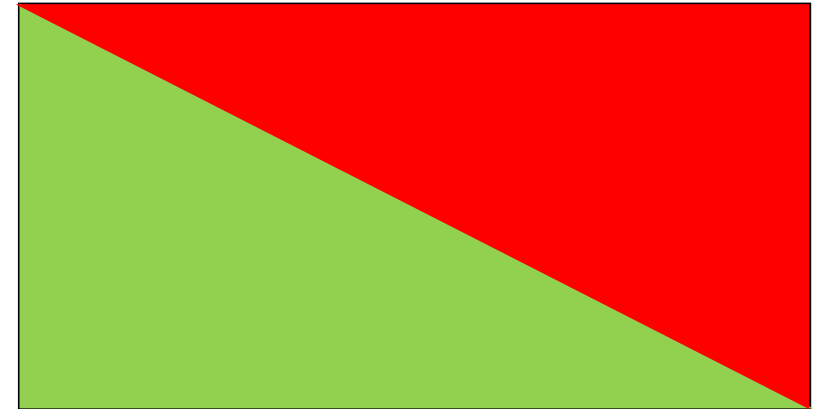
Forks



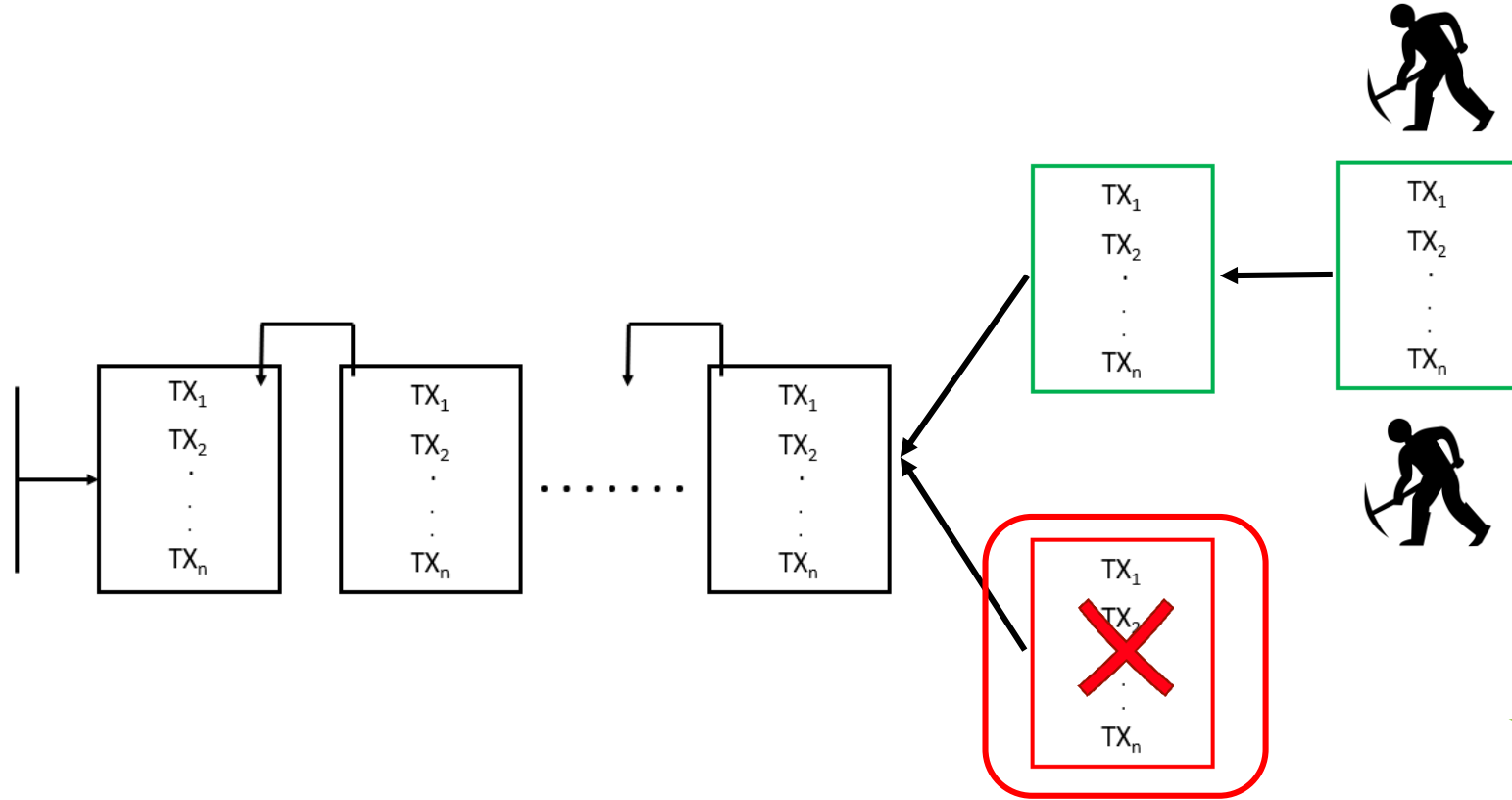
Forks



- Transactions in this block have to be resubmitted



Forks



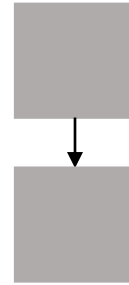
- Transactions in this block have to be resubmitted

Forks: The Big Picture

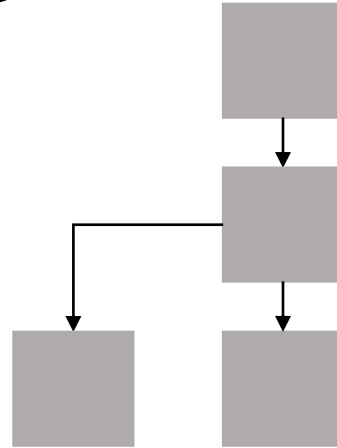
Forks: The Big Picture



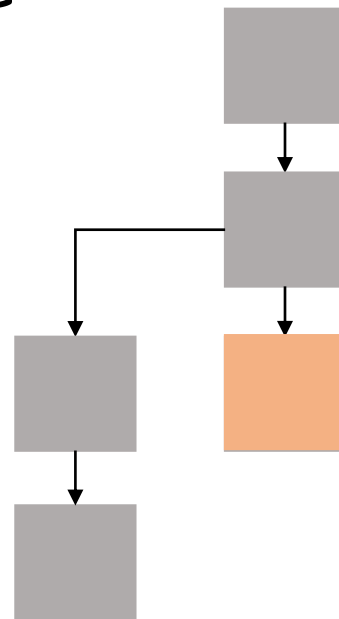
Forks: The Big Picture



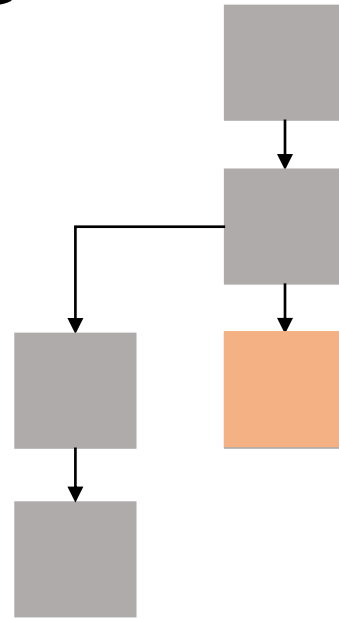
Forks: The Big Picture





Forks: The Big Picture

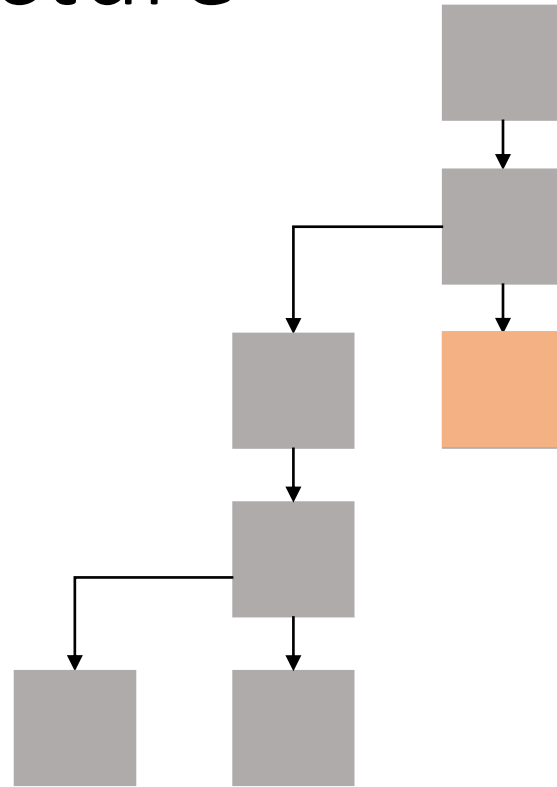




Forks: The Big Picture



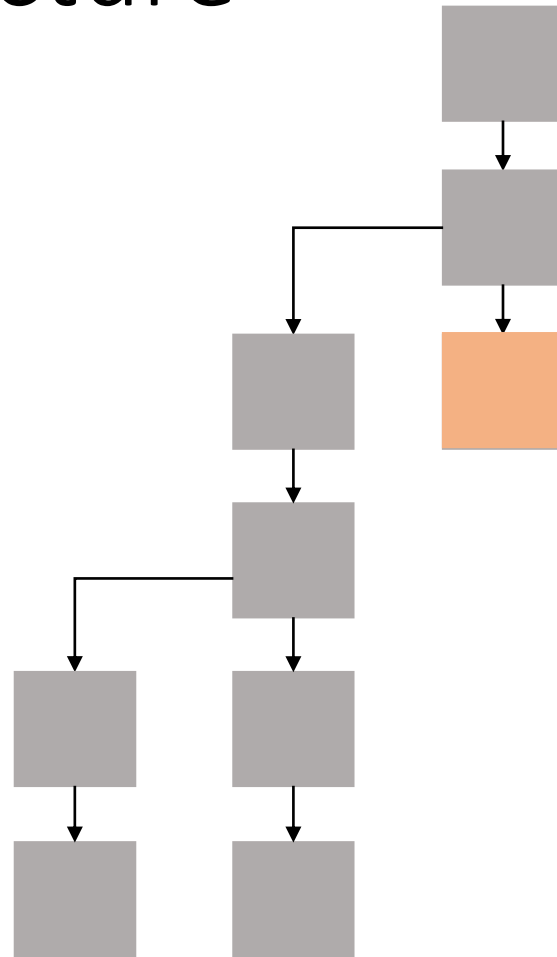
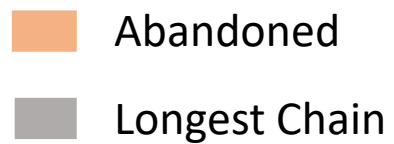
-  Abandoned
-  Longest Chain

Forks: The Big Picture

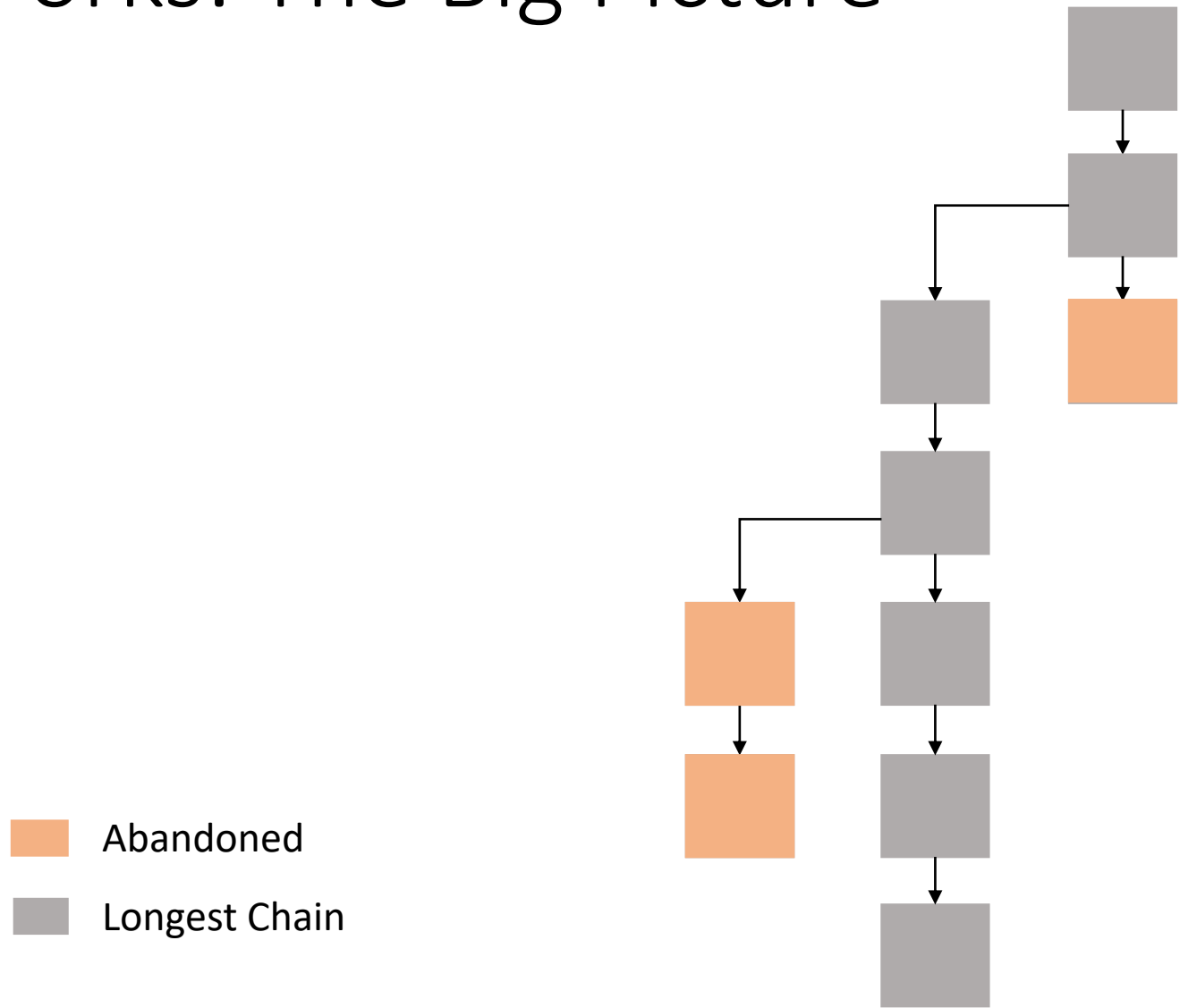


-  Abandoned
-  Longest Chain

Forks: The Big Picture



Forks: The Big Picture

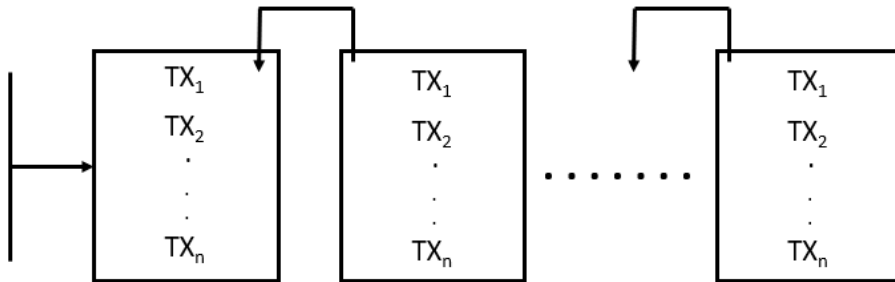


51% Attack

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

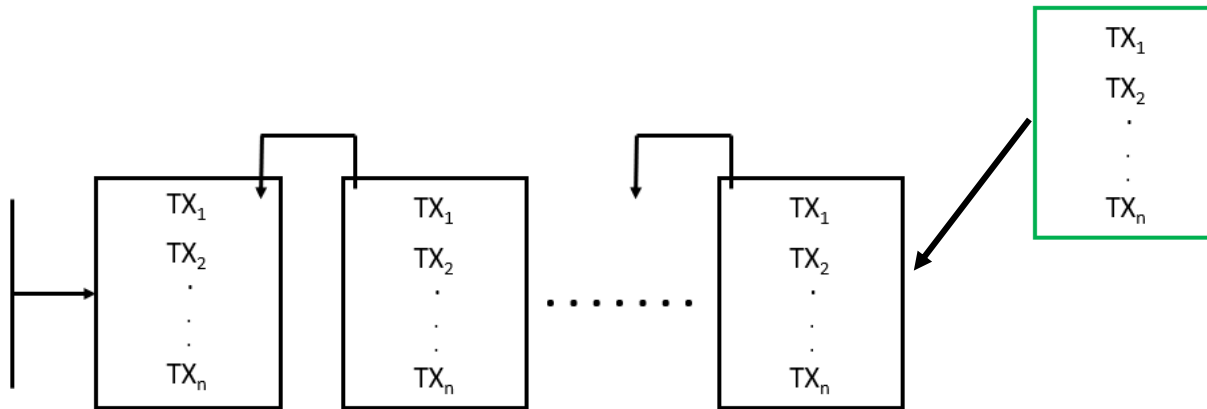
51% Attack

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



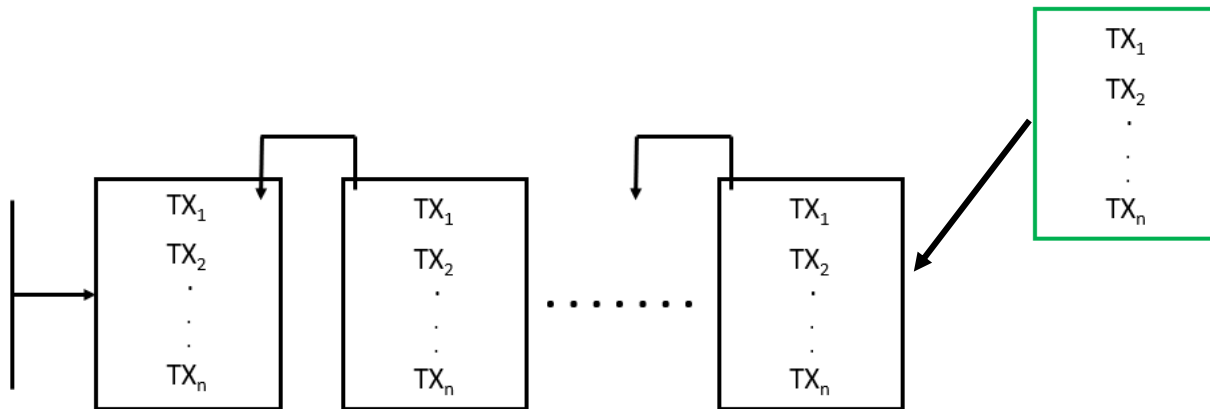
51% Attack

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



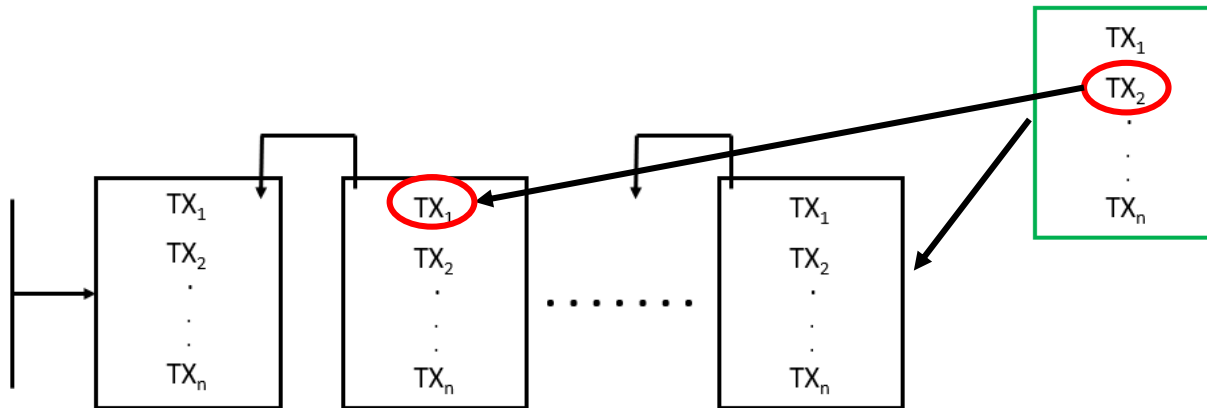
51% Attack

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



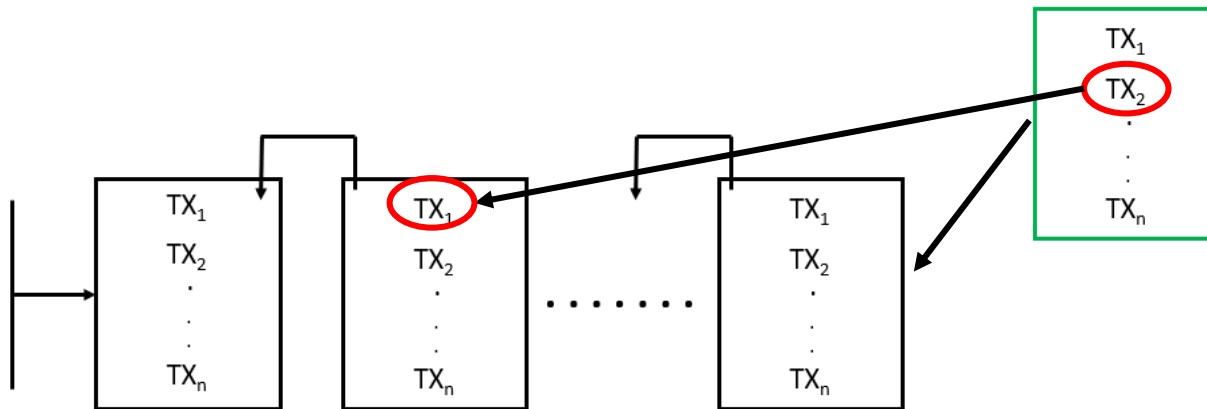
51% Attack

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



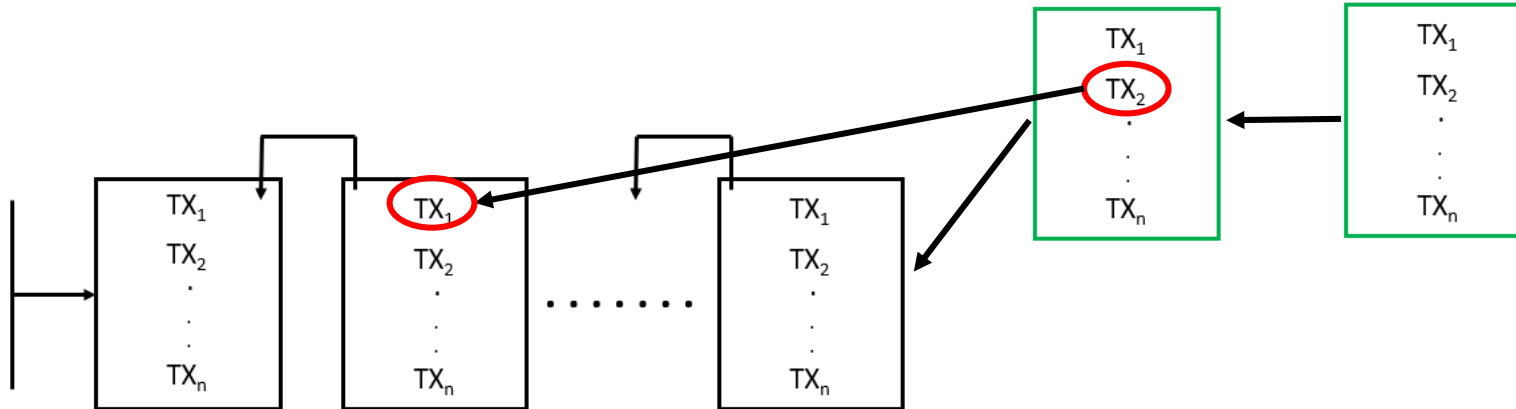
51% Attack

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



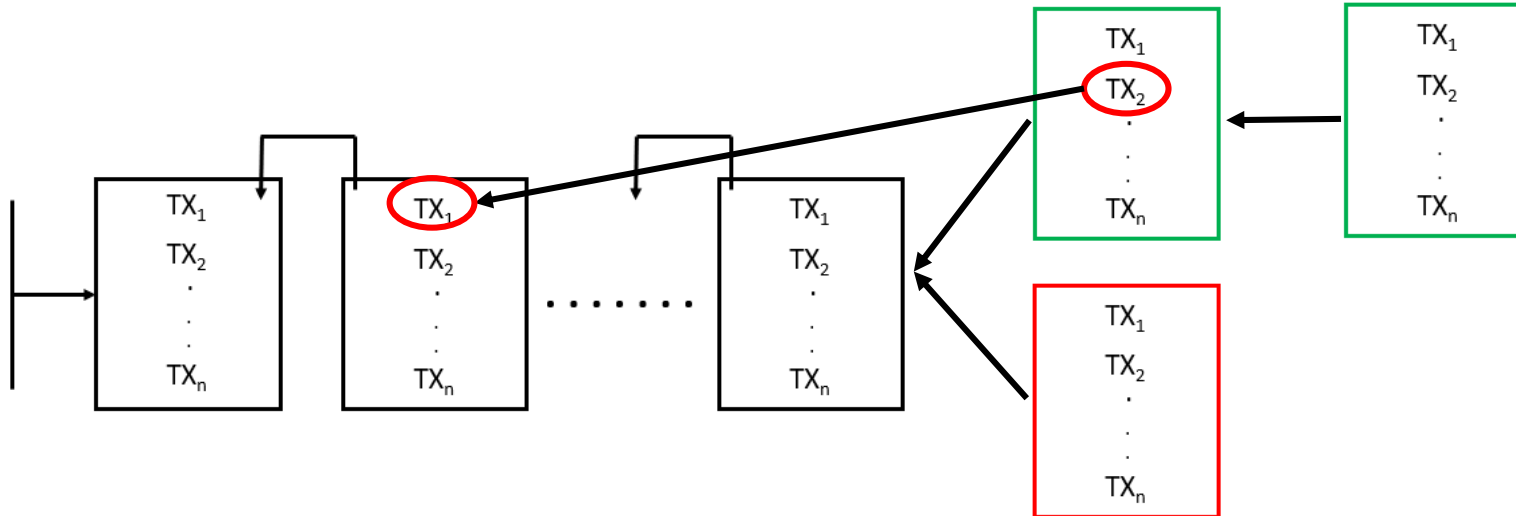
51% Attack

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



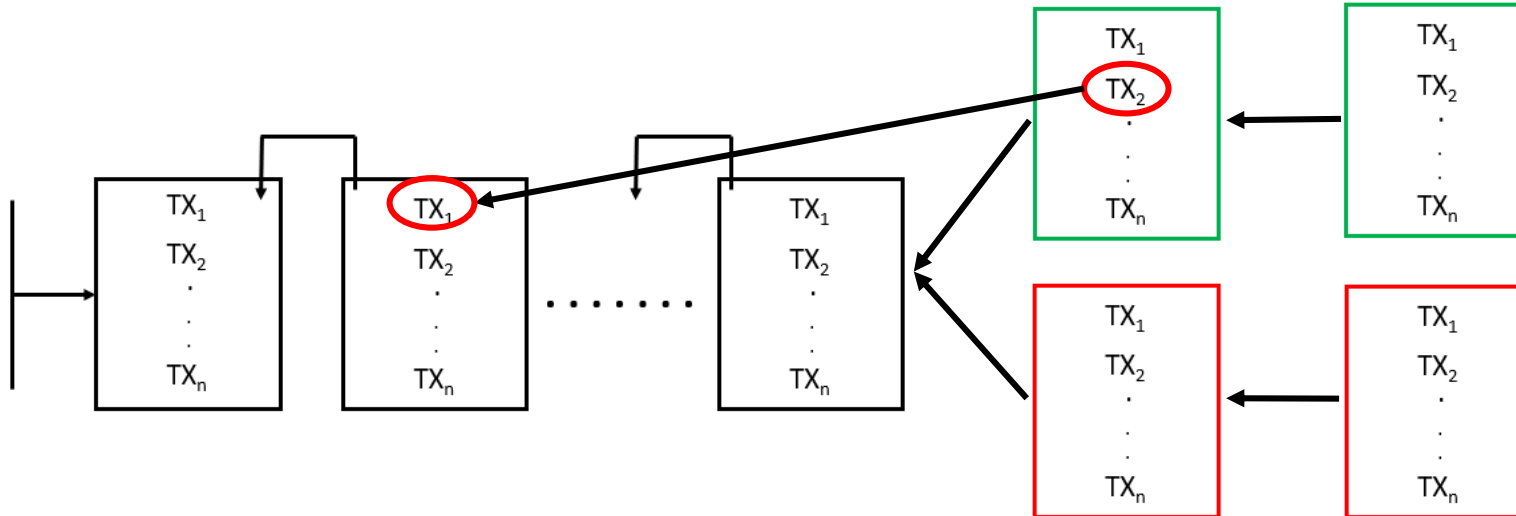
51% Attack

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



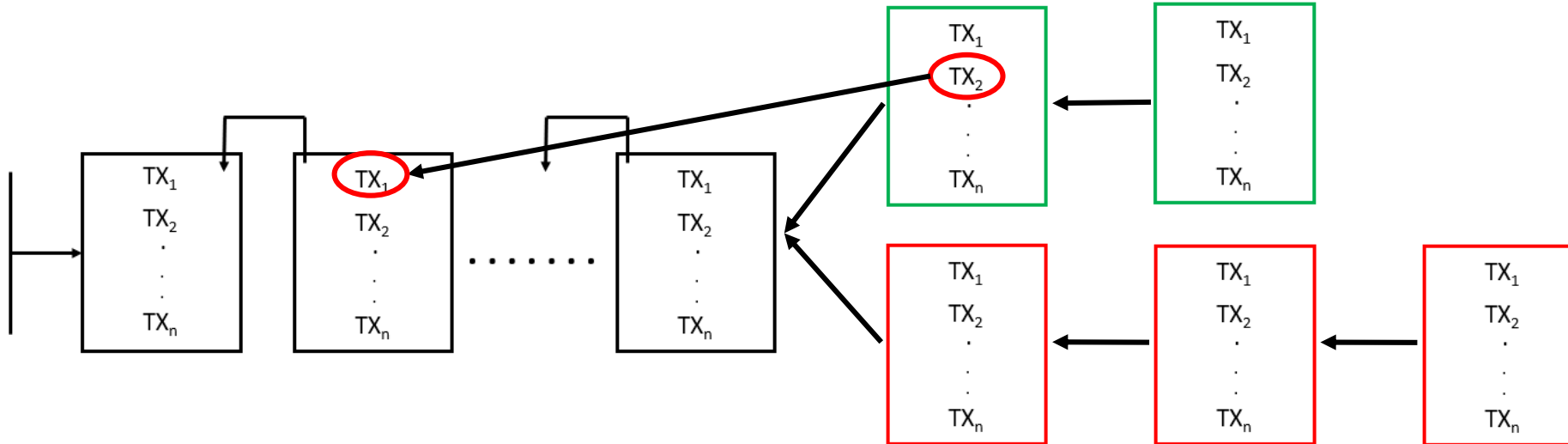
51% Attack

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



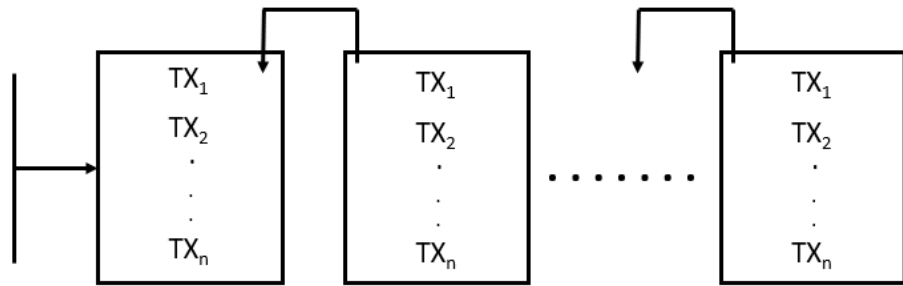
51% Attack

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



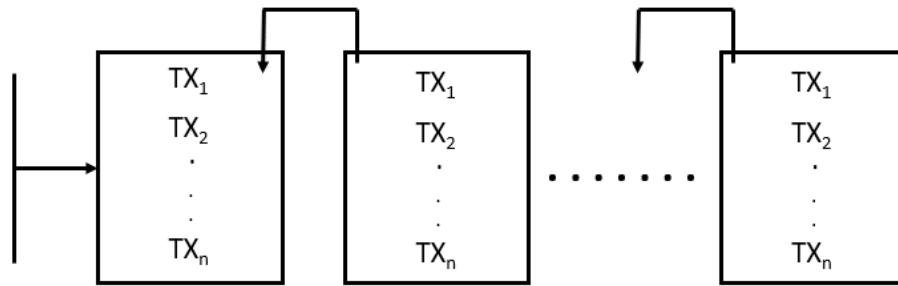
Selfish Mining

Selfish Mining



Selfish Mining

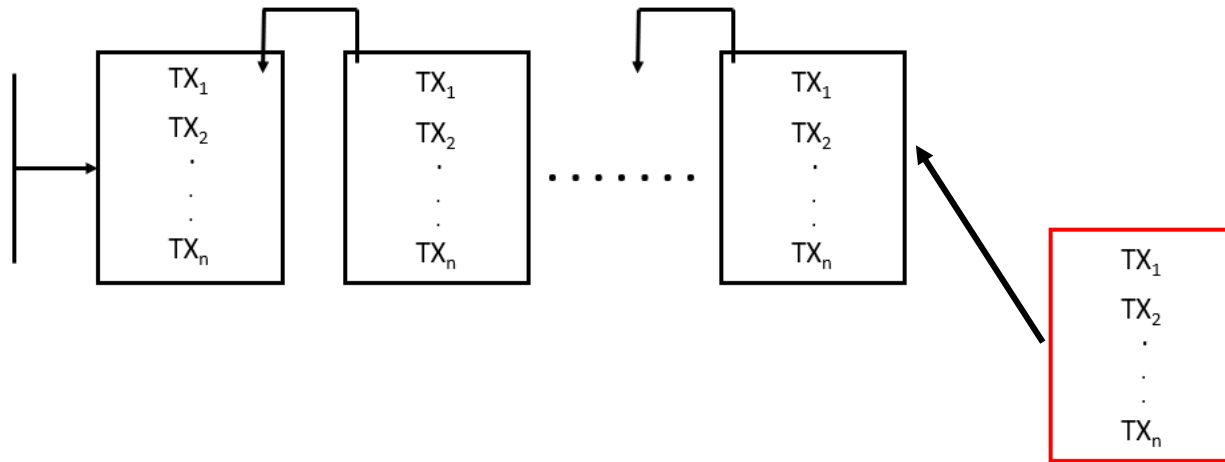
Honest Miner



Selfish Miner

Selfish Mining

Honest Miner

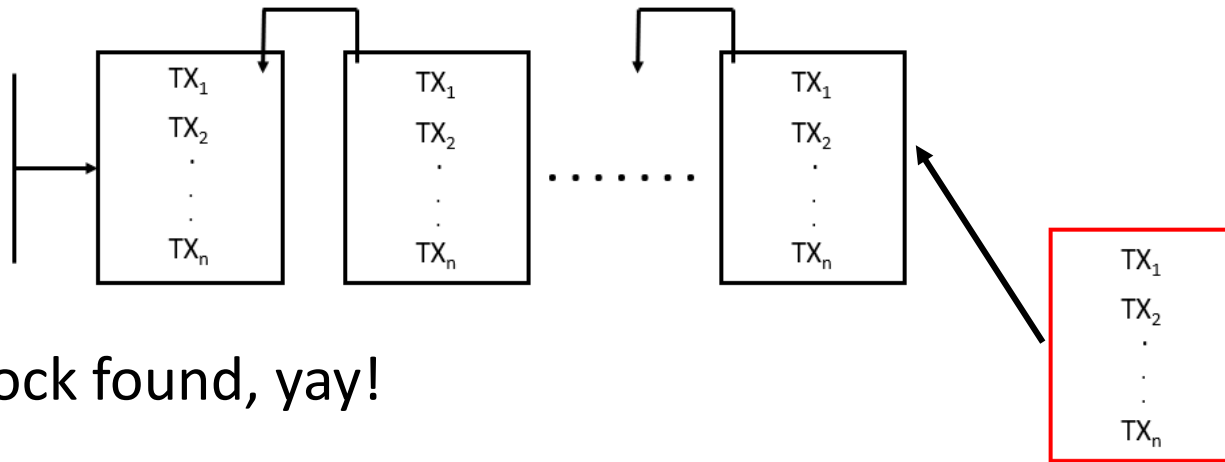


Selfish Miner



Selfish Mining

Honest Miner



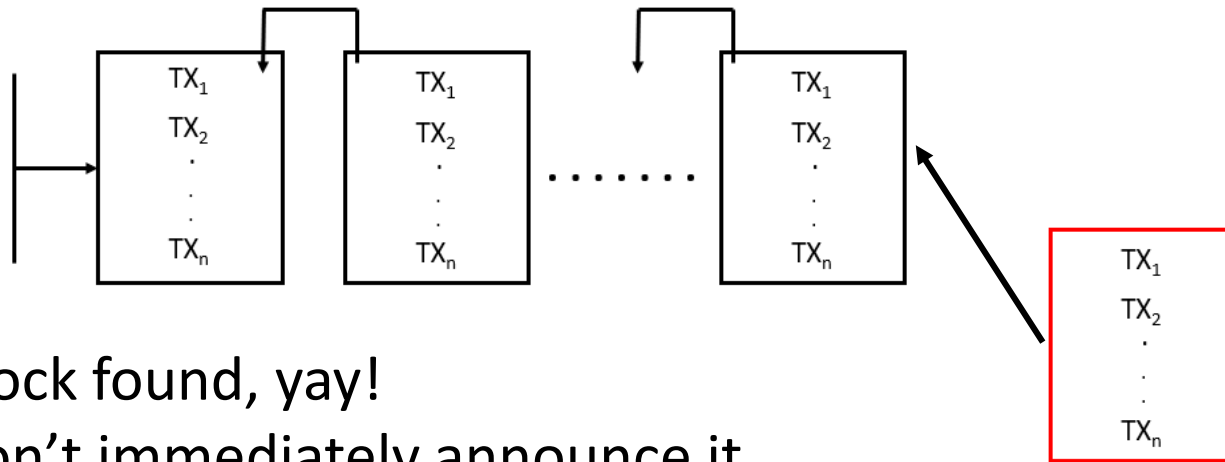
- Block found, yay!



Selfish Miner

Selfish Mining

Honest Miner



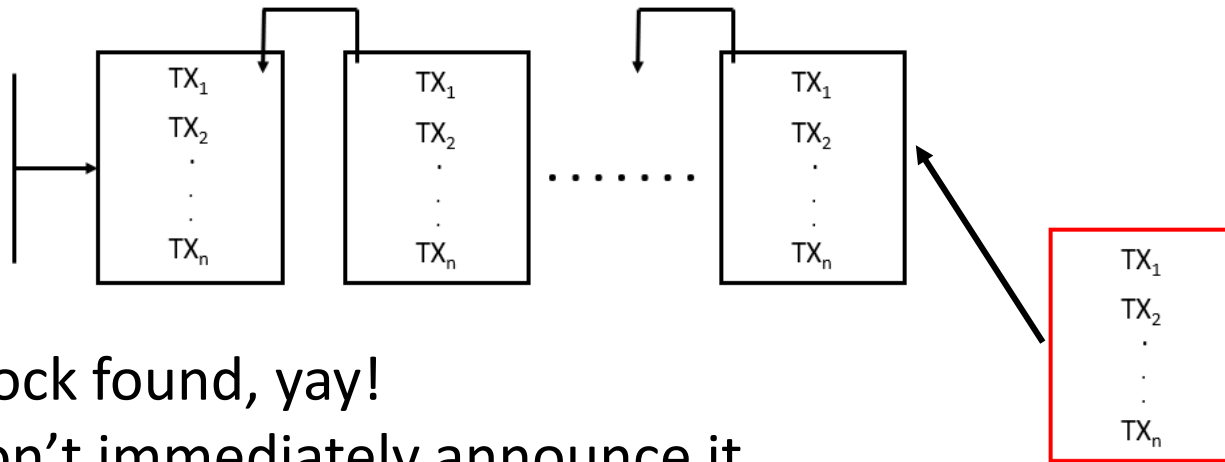
- Block found, yay!
- Don't immediately announce it



Selfish Miner

Selfish Mining

Honest Miner



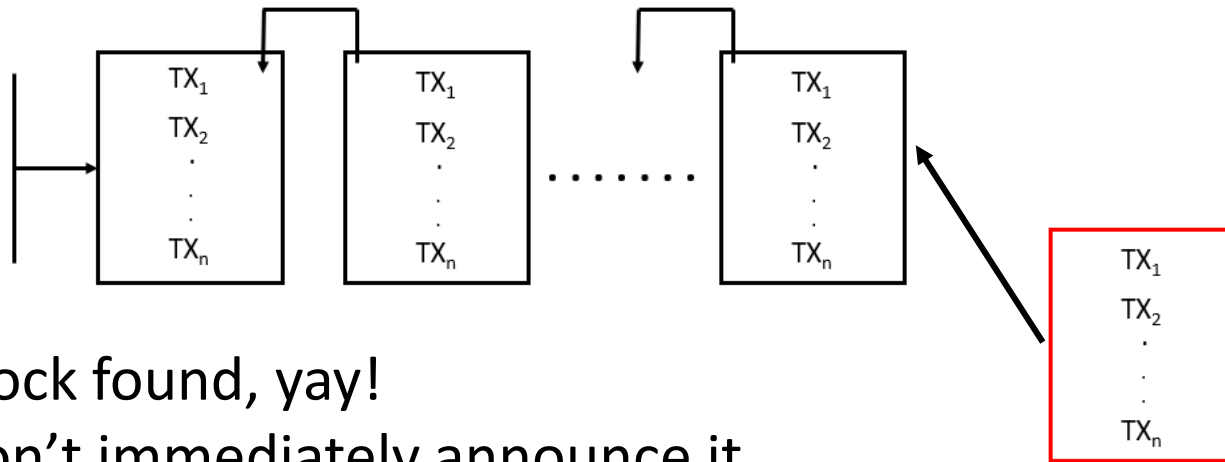
- Block found, yay!
- Don't immediately announce it
- Let honest miners waste their mining power on an obsolete block



Selfish Miner

Selfish Mining

Honest Miner



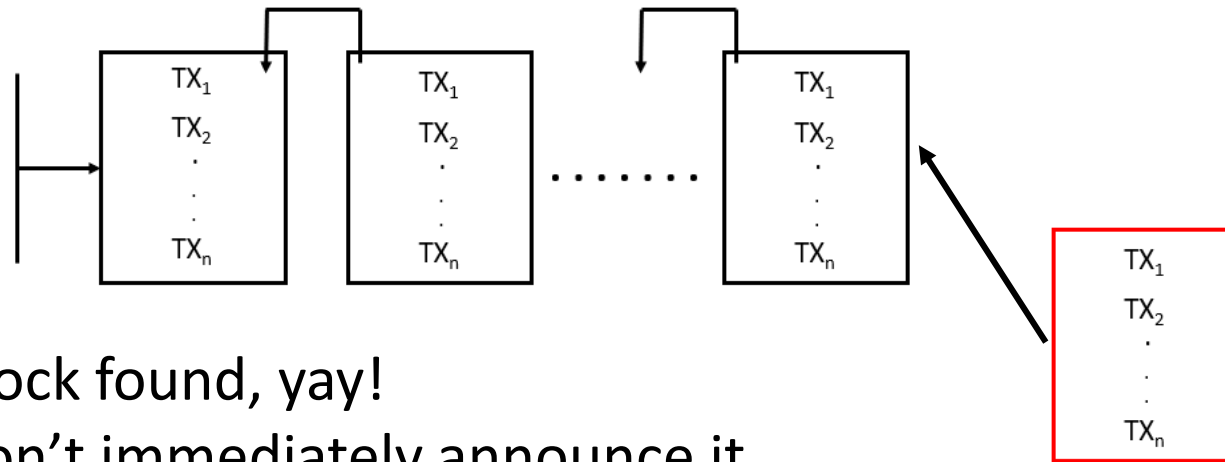
- Block found, yay!
- Don't immediately announce it
- Let honest miners waste their mining power on an obsolete block
- Start mining the next block (Advantage)



Selfish Miner

Selfish Mining

Honest Miner



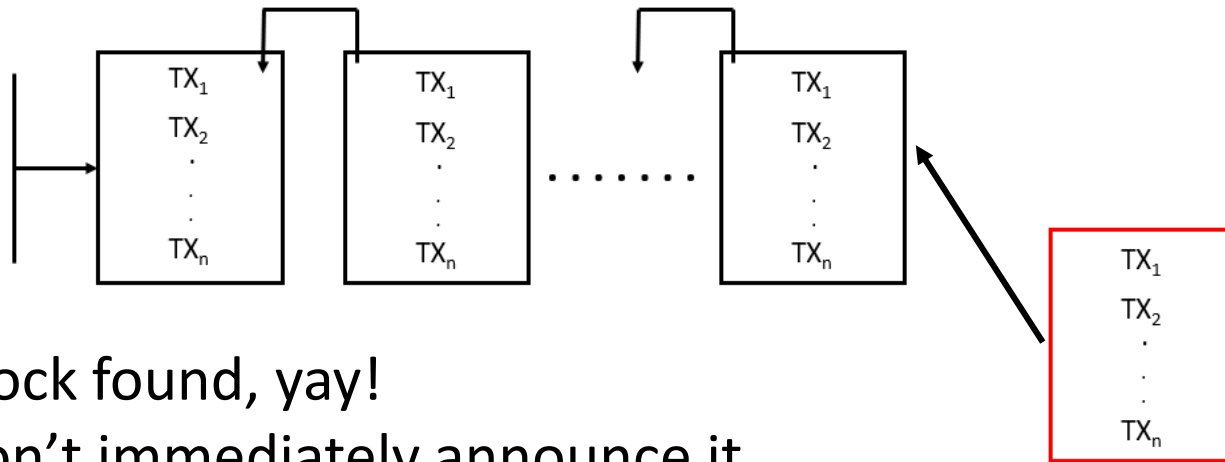
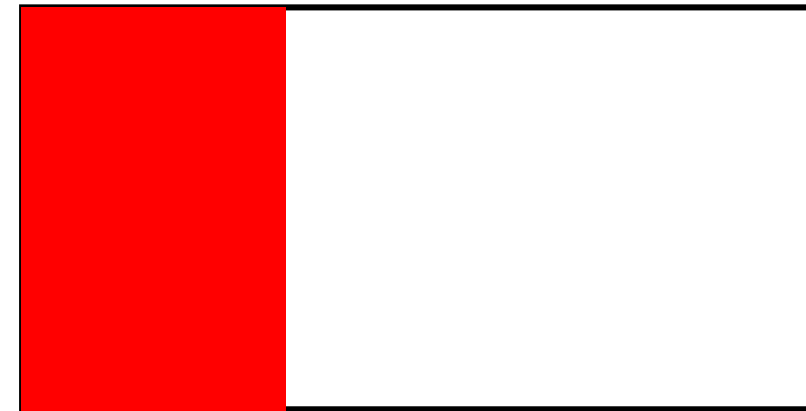
- Block found, yay!
- Don't immediately announce it
- Let honest miners waste their mining power on an obsolete block
- Start mining the next block (Advantage)



Selfish Miner

Selfish Mining

Honest Miner



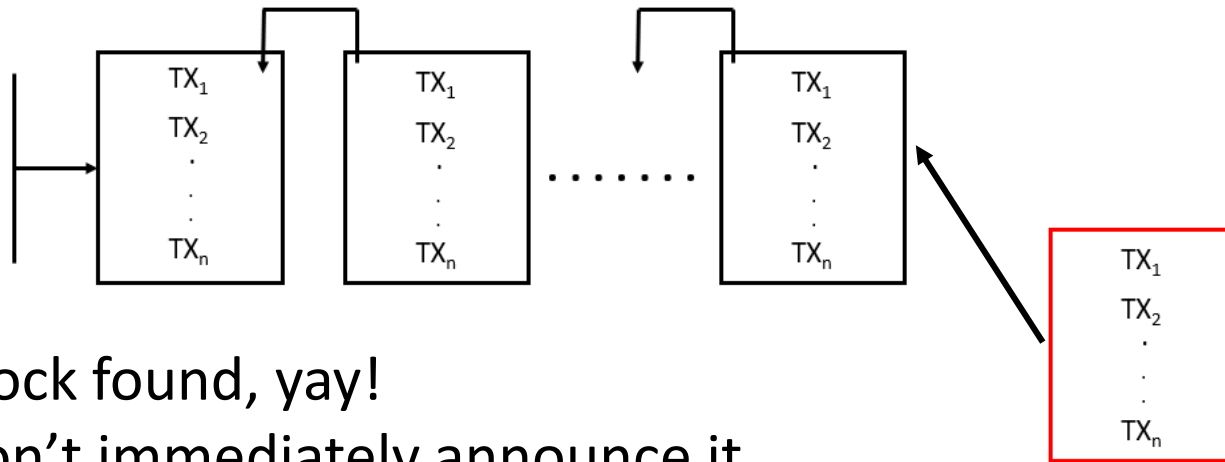
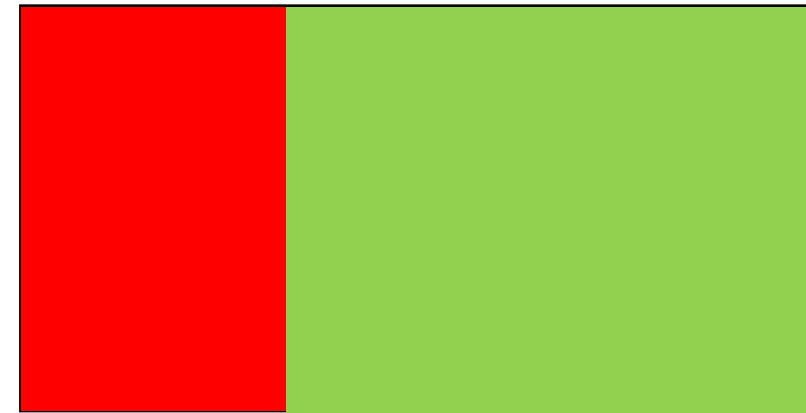
- Block found, yay!
- Don't immediately announce it
- Let honest miners waste their mining power on an obsolete block
- Start mining the next block (Advantage)



Selfish Miner

Selfish Mining

Honest Miner



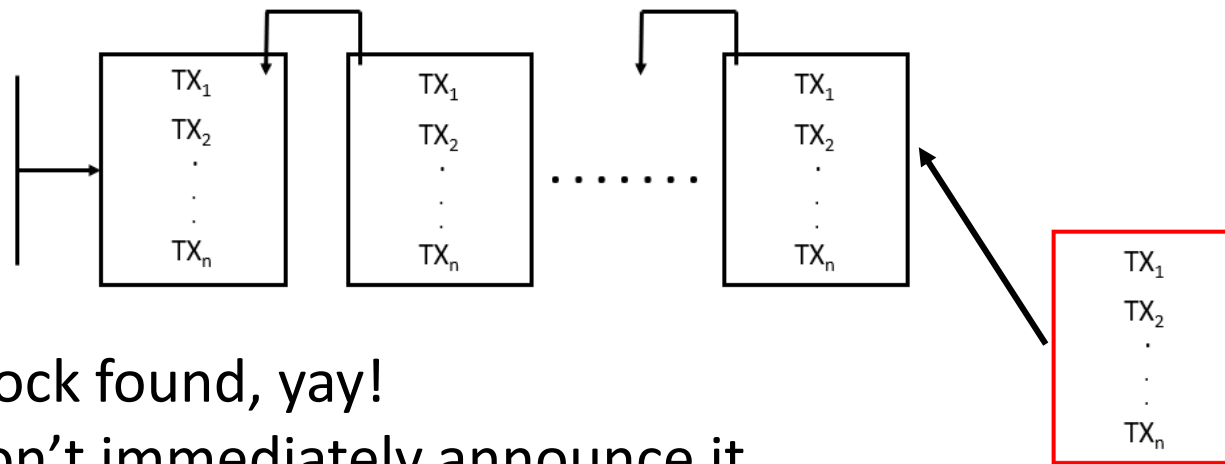
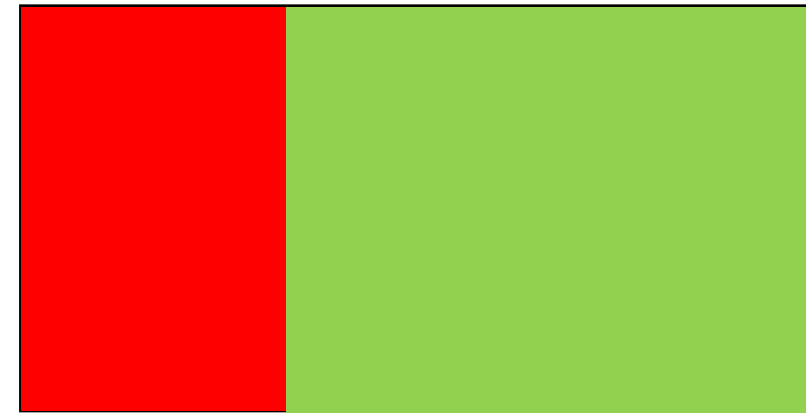
- Block found, yay!
- Don't immediately announce it
- Let honest miners waste their mining power on an obsolete block
- Start mining the next block (Advantage)



Selfish Miner

Selfish Mining

Honest Miner



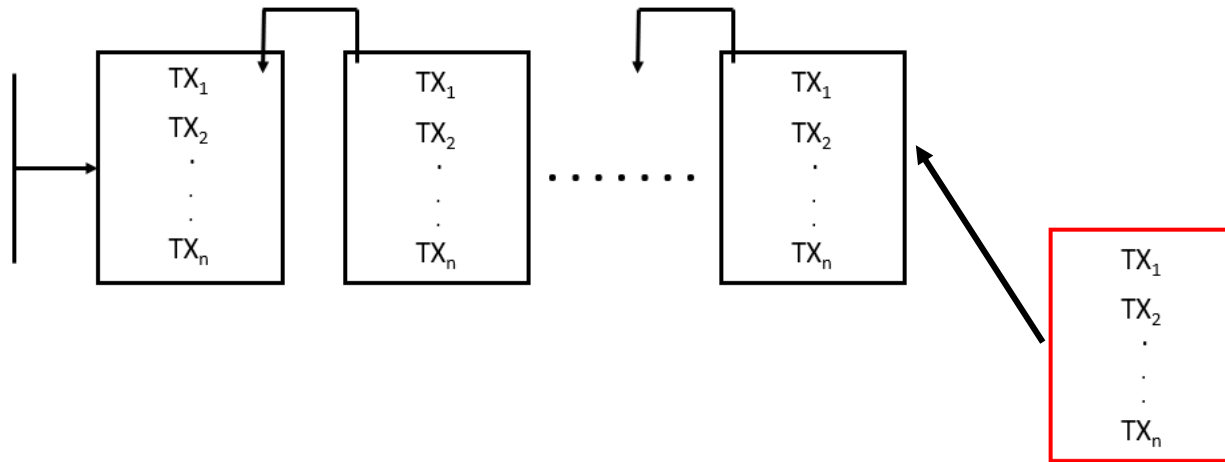
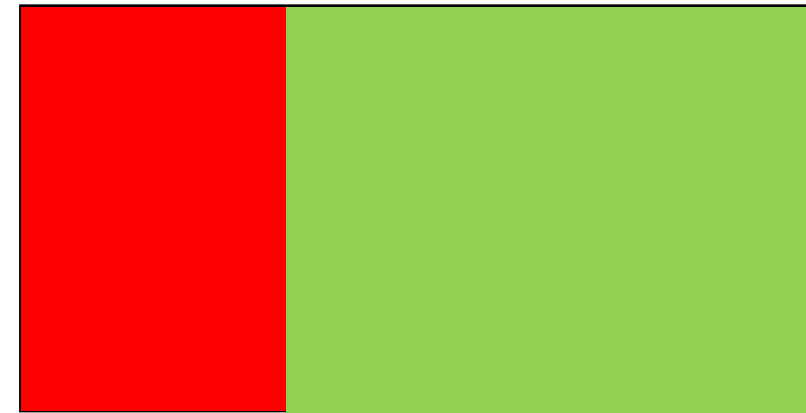
- Block found, yay!
- Don't immediately announce it
- Let honest miners waste their mining power on an obsolete block
- Start mining the next block (Advantage)
- Two possible outcomes



Selfish Miner

Selfish Mining

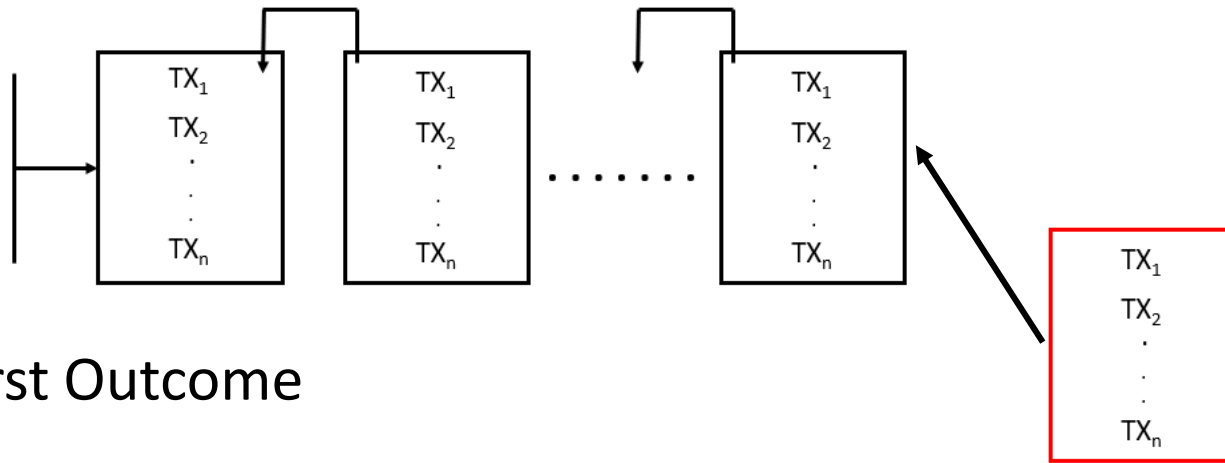
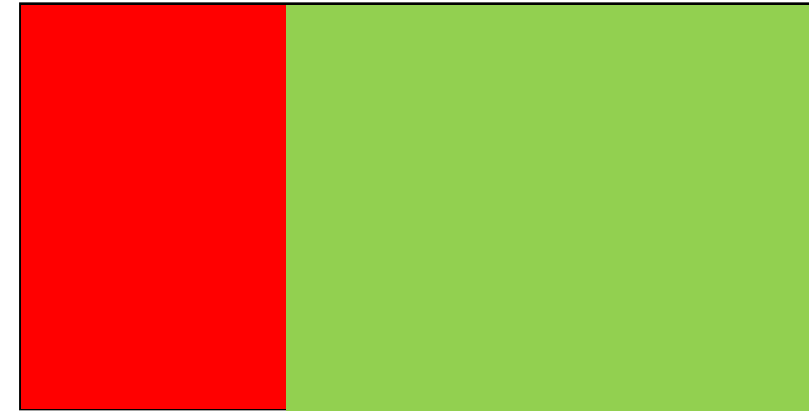
Honest Miner



Selfish Miner

Selfish Mining

Honest Miner

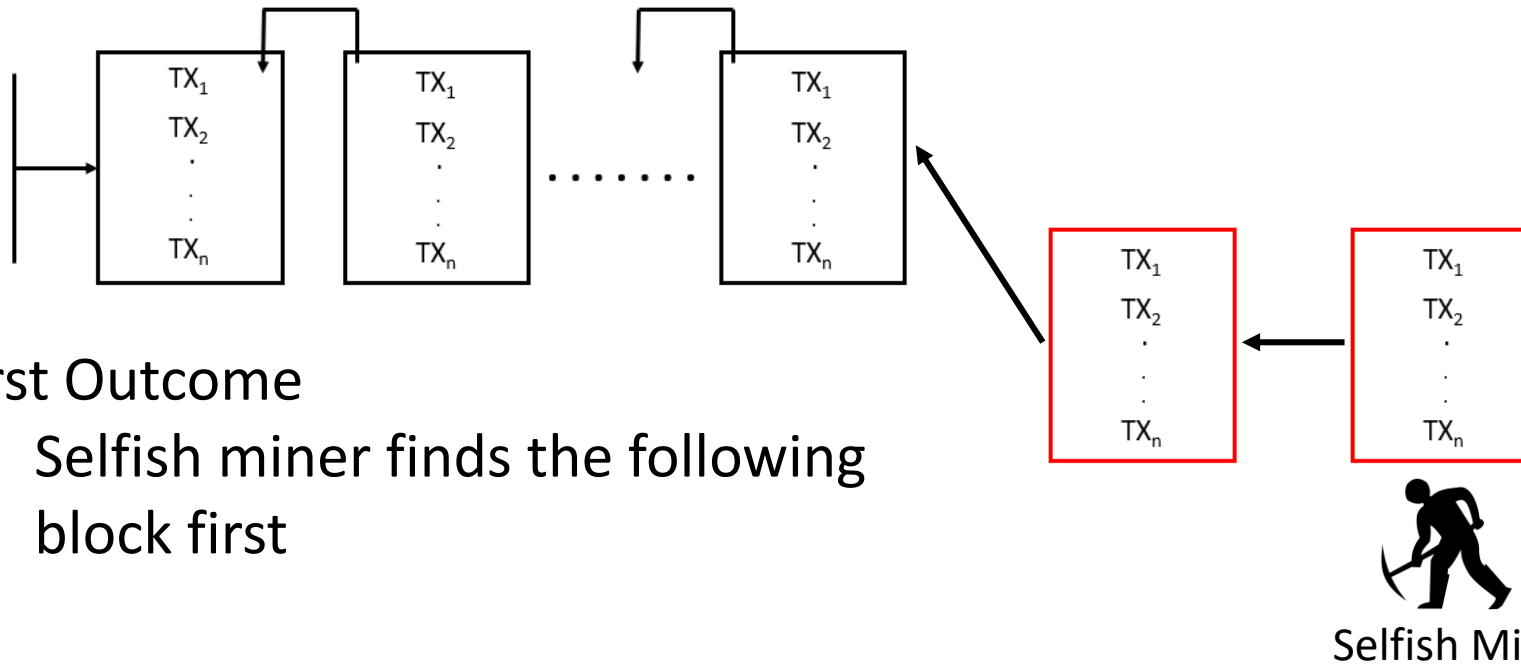
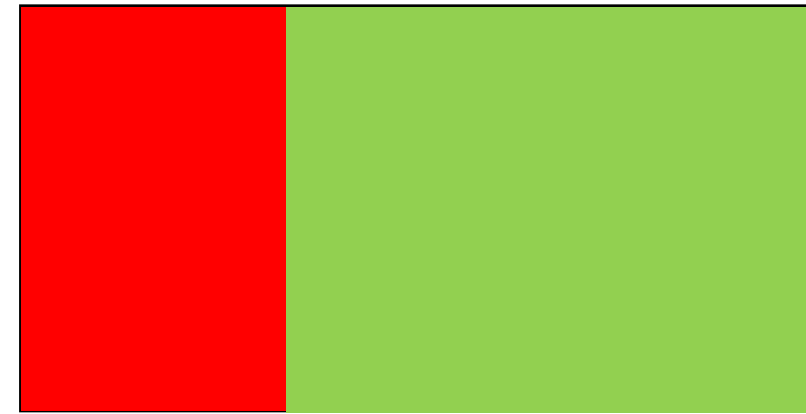


- First Outcome


Selfish Miner

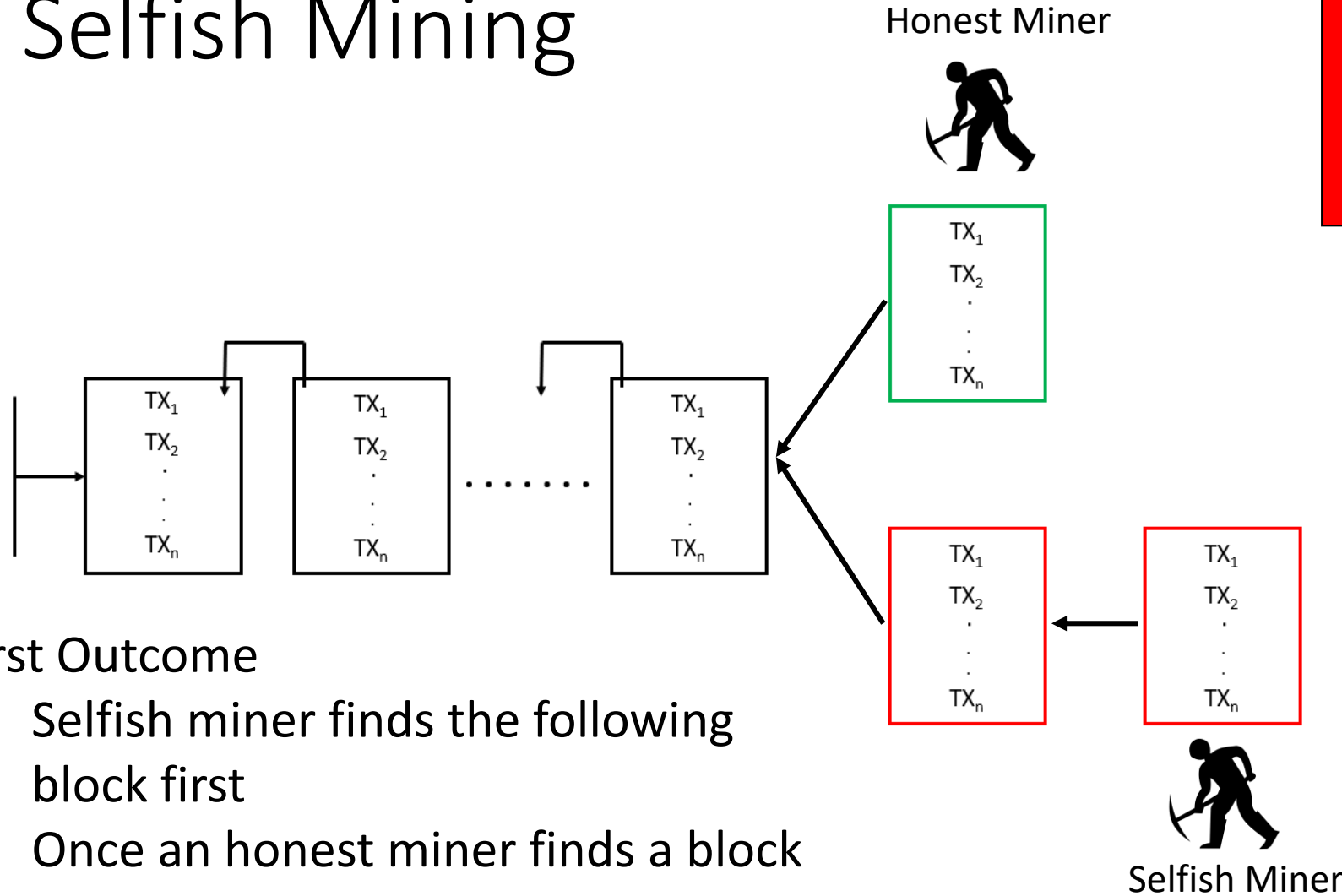
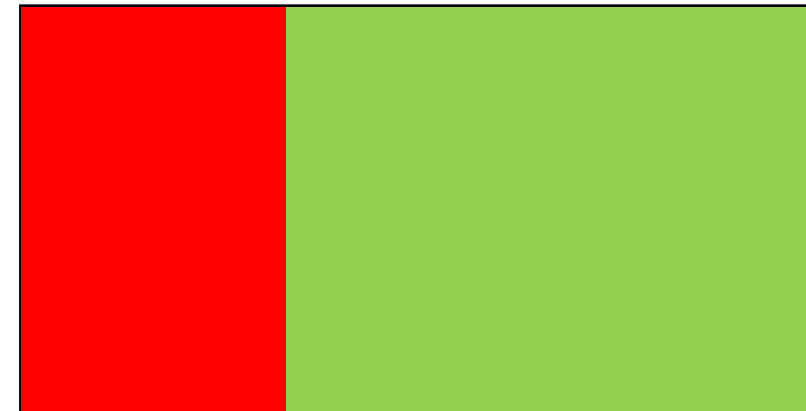
Selfish Mining

Honest Miner



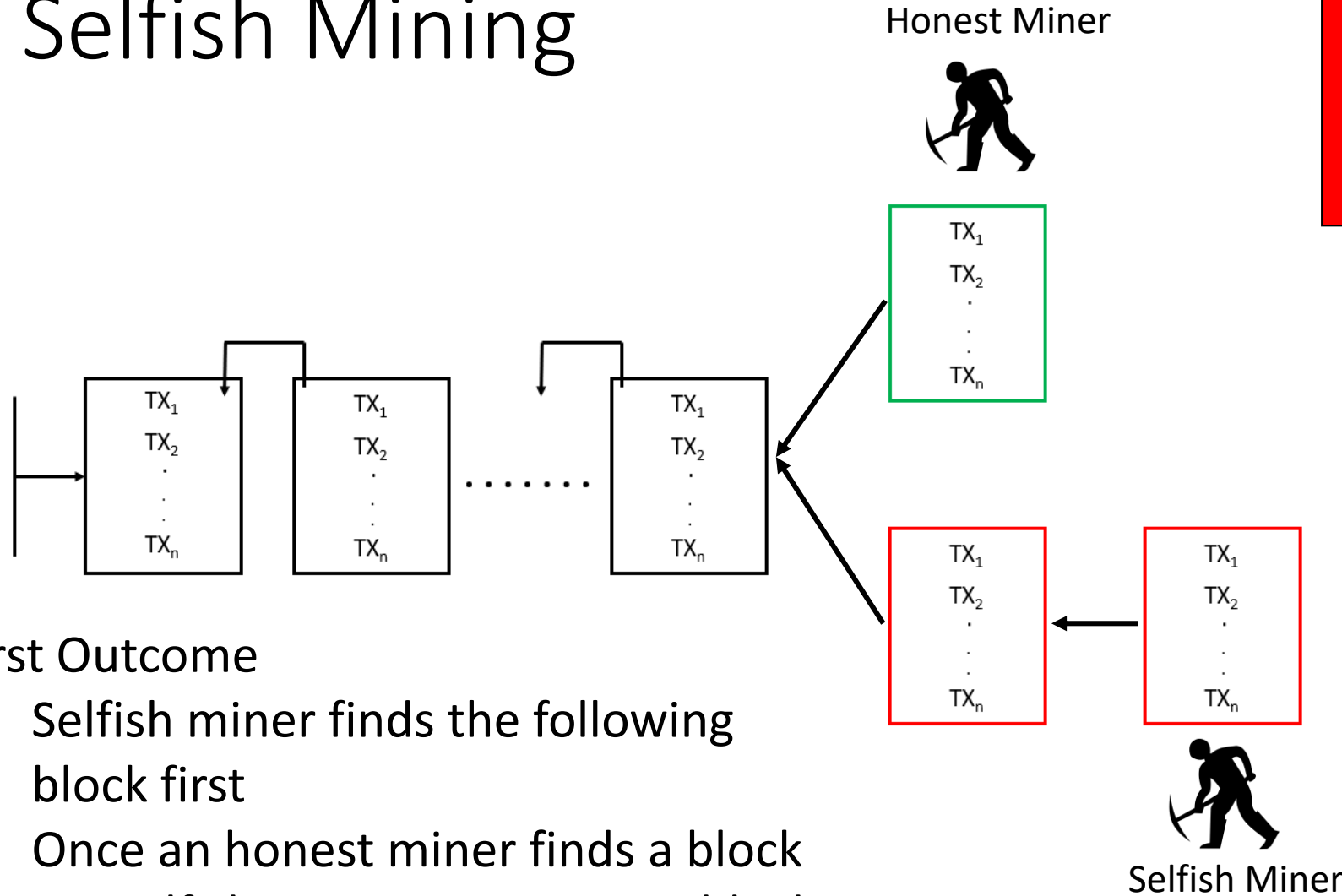
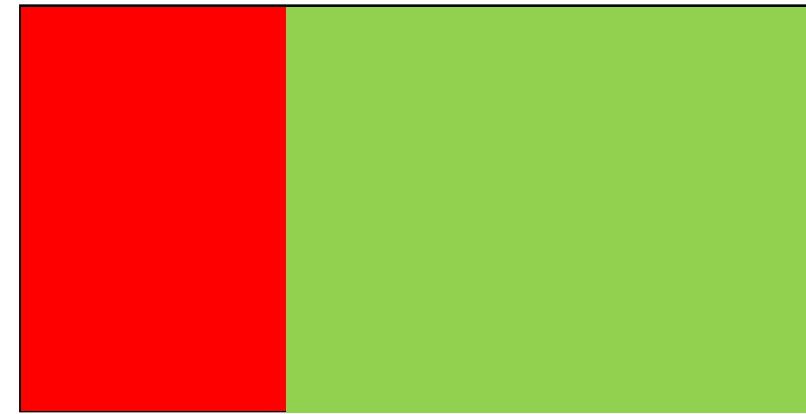
- First Outcome
 - Selfish miner finds the following block first

Selfish Mining



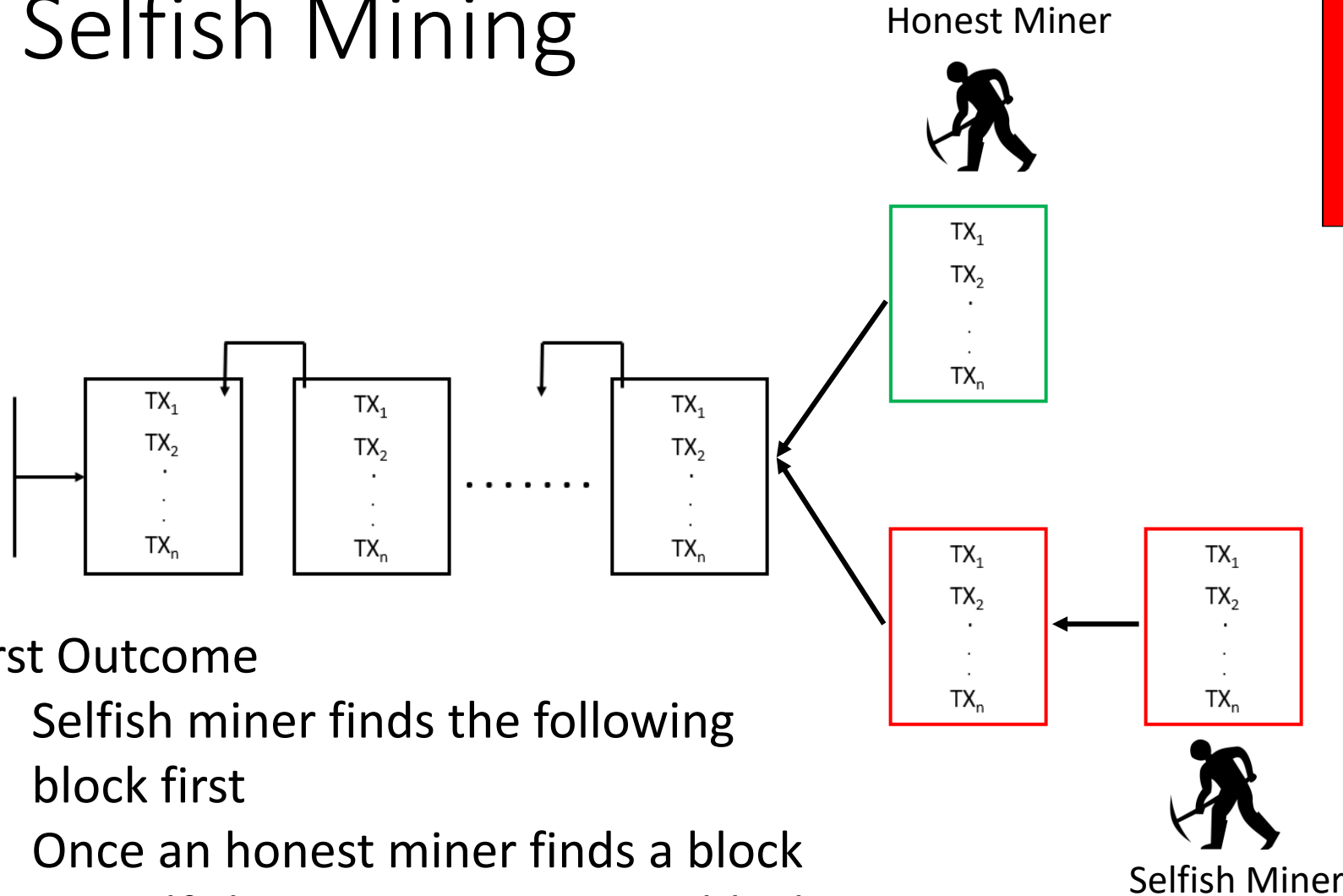
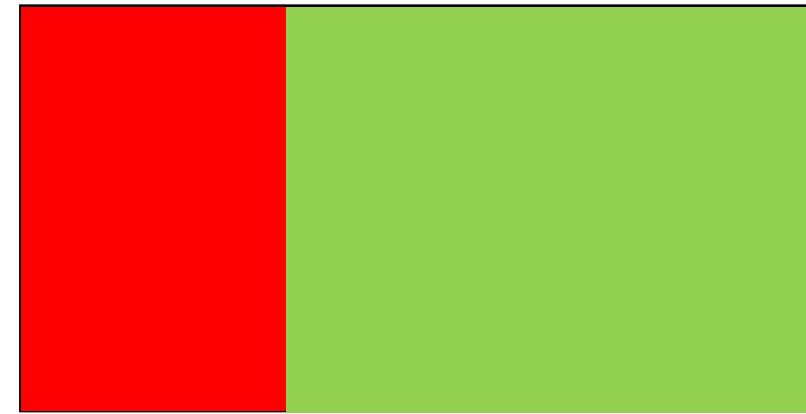
- First Outcome
 - Selfish miner finds the following block first
 - Once an honest miner finds a block

Selfish Mining



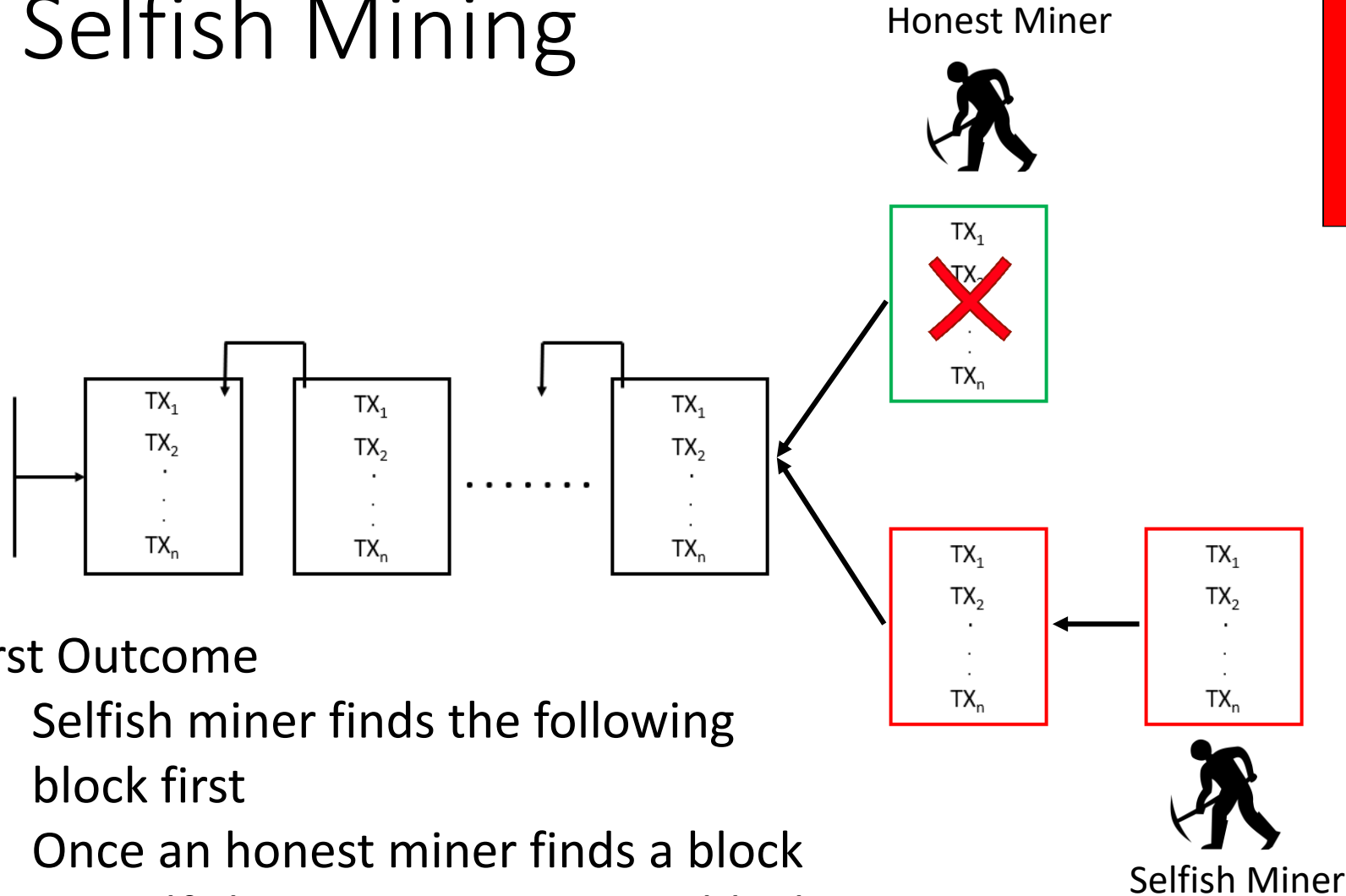
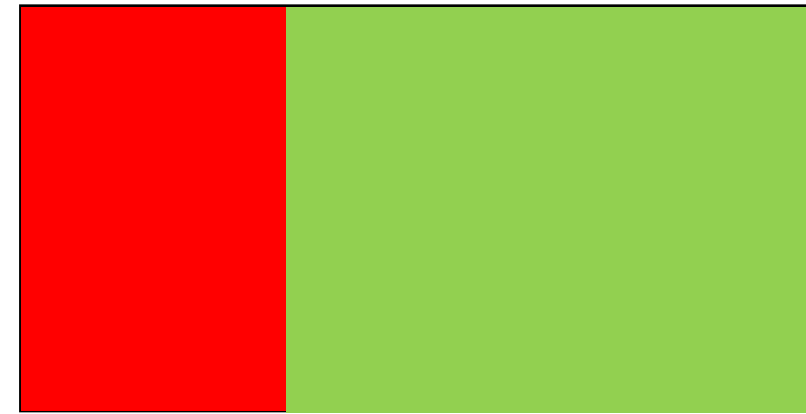
- First Outcome
 - Selfish miner finds the following block first
 - Once an honest miner finds a block
 - Selfish miner announces 2 blocks

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

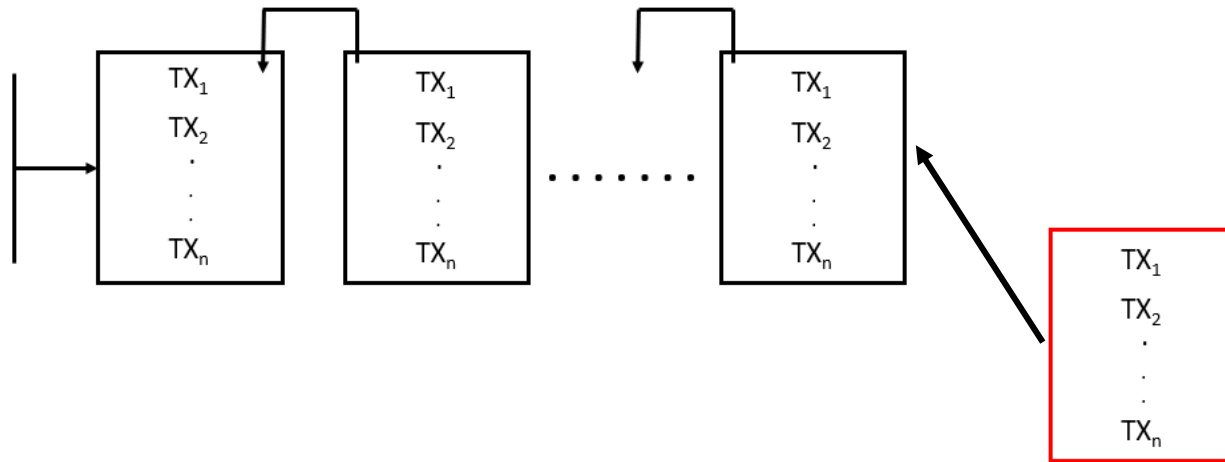
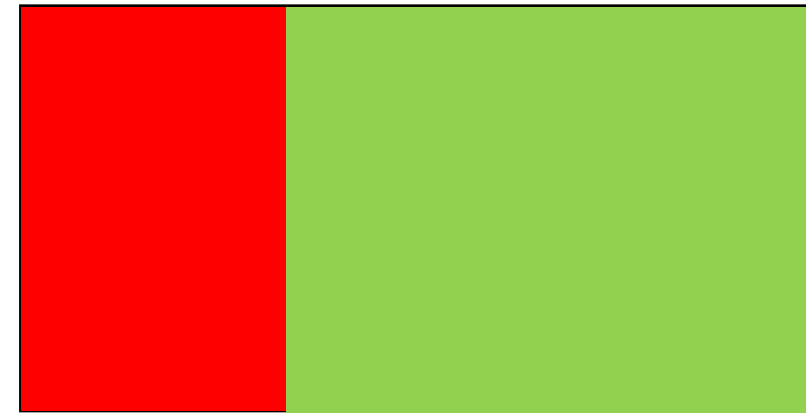
Selfish Mining



- 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

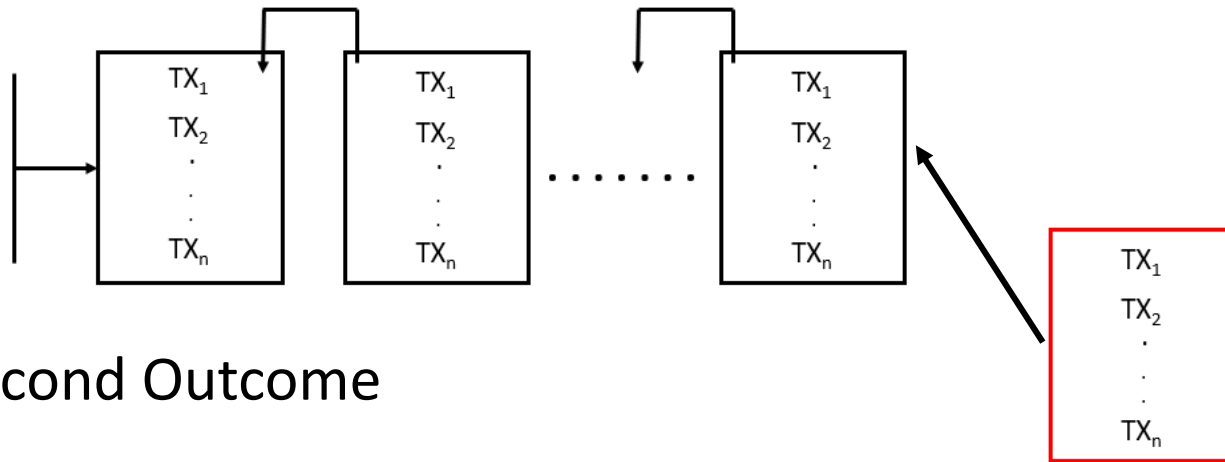
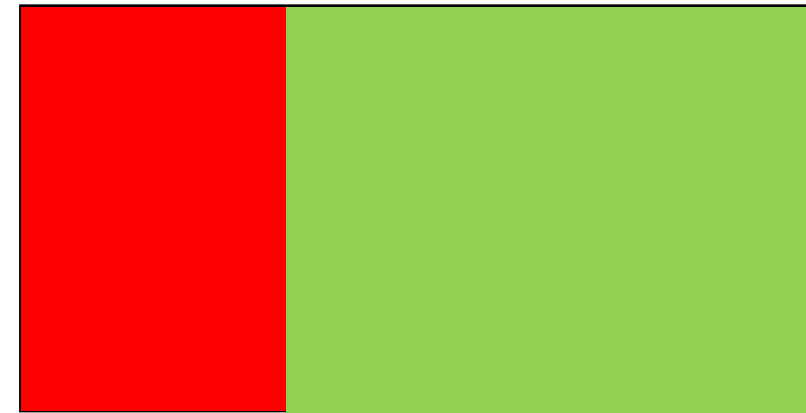
Honest Miner




Selfish Miner

Selfish Mining

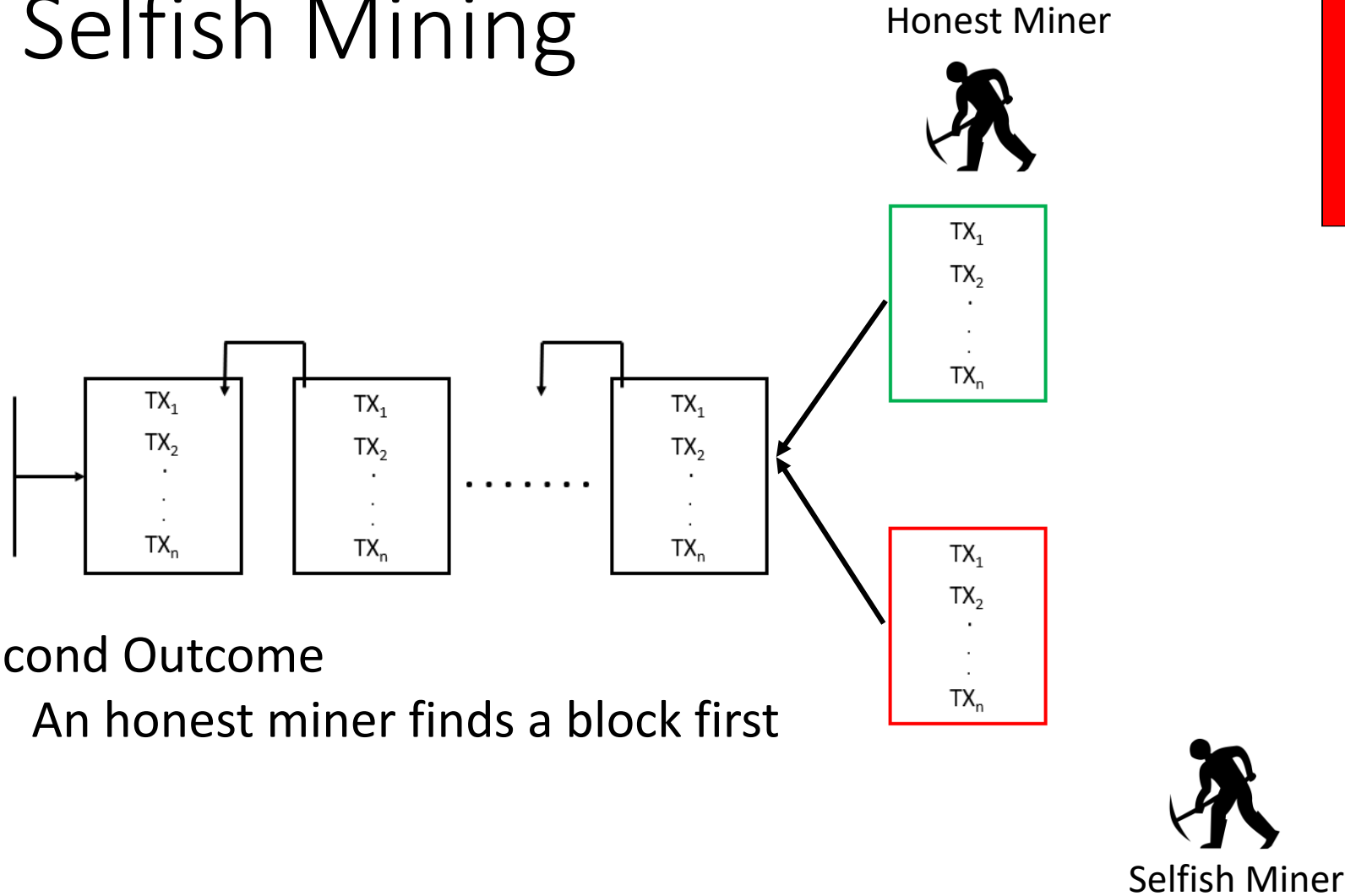
Honest Miner



- Second Outcome

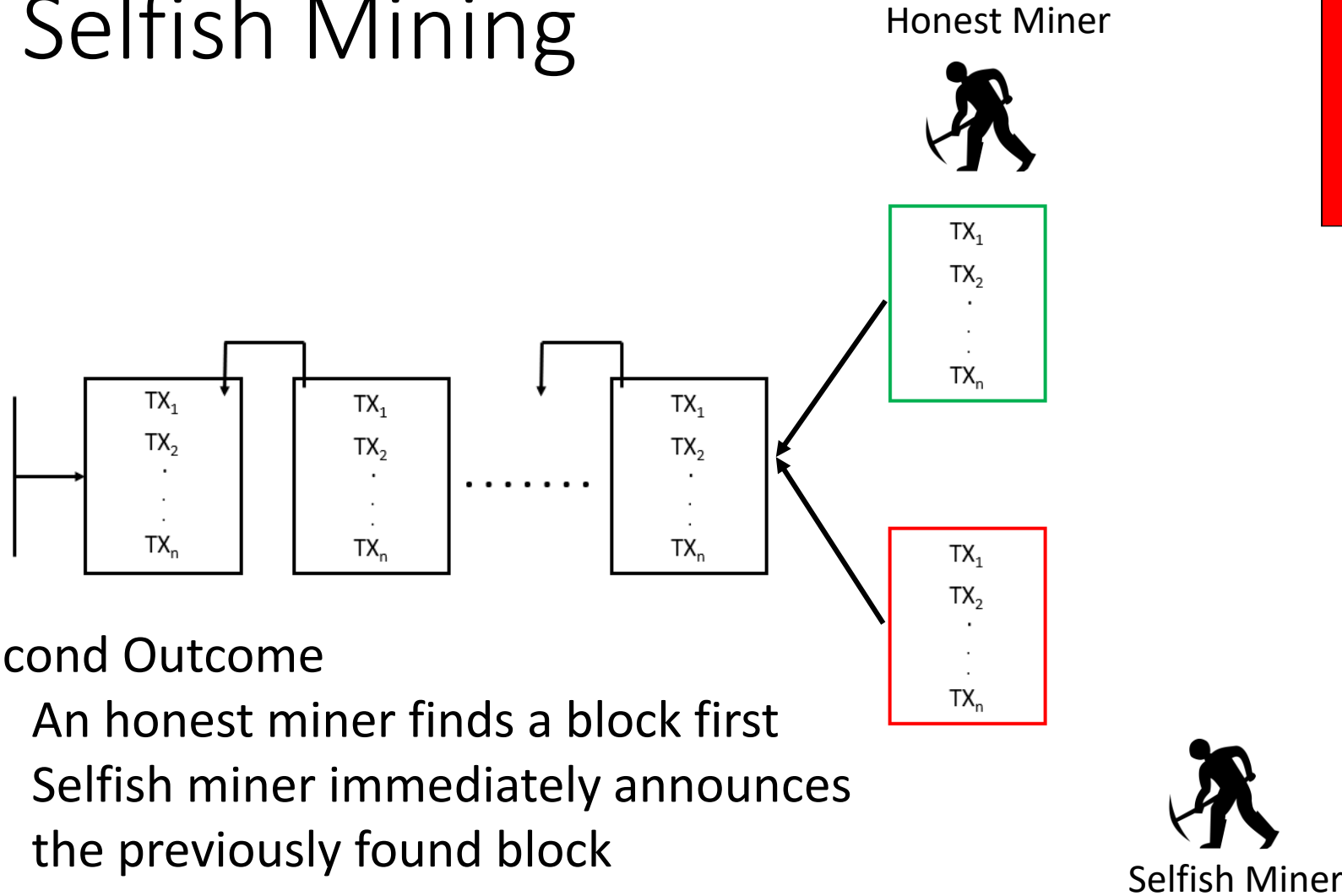
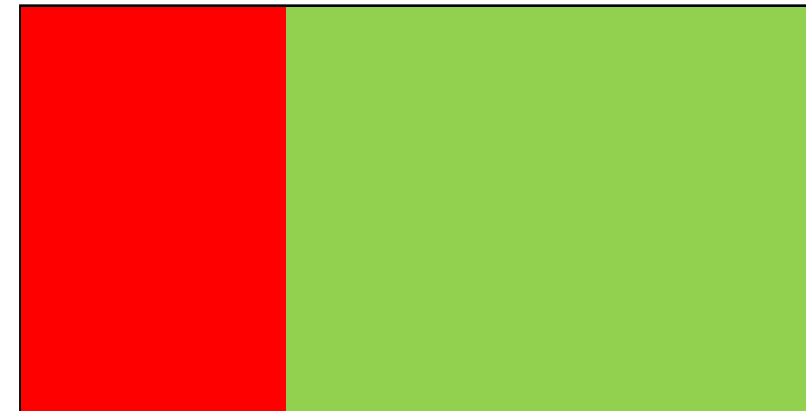

Selfish Miner

Selfish Mining



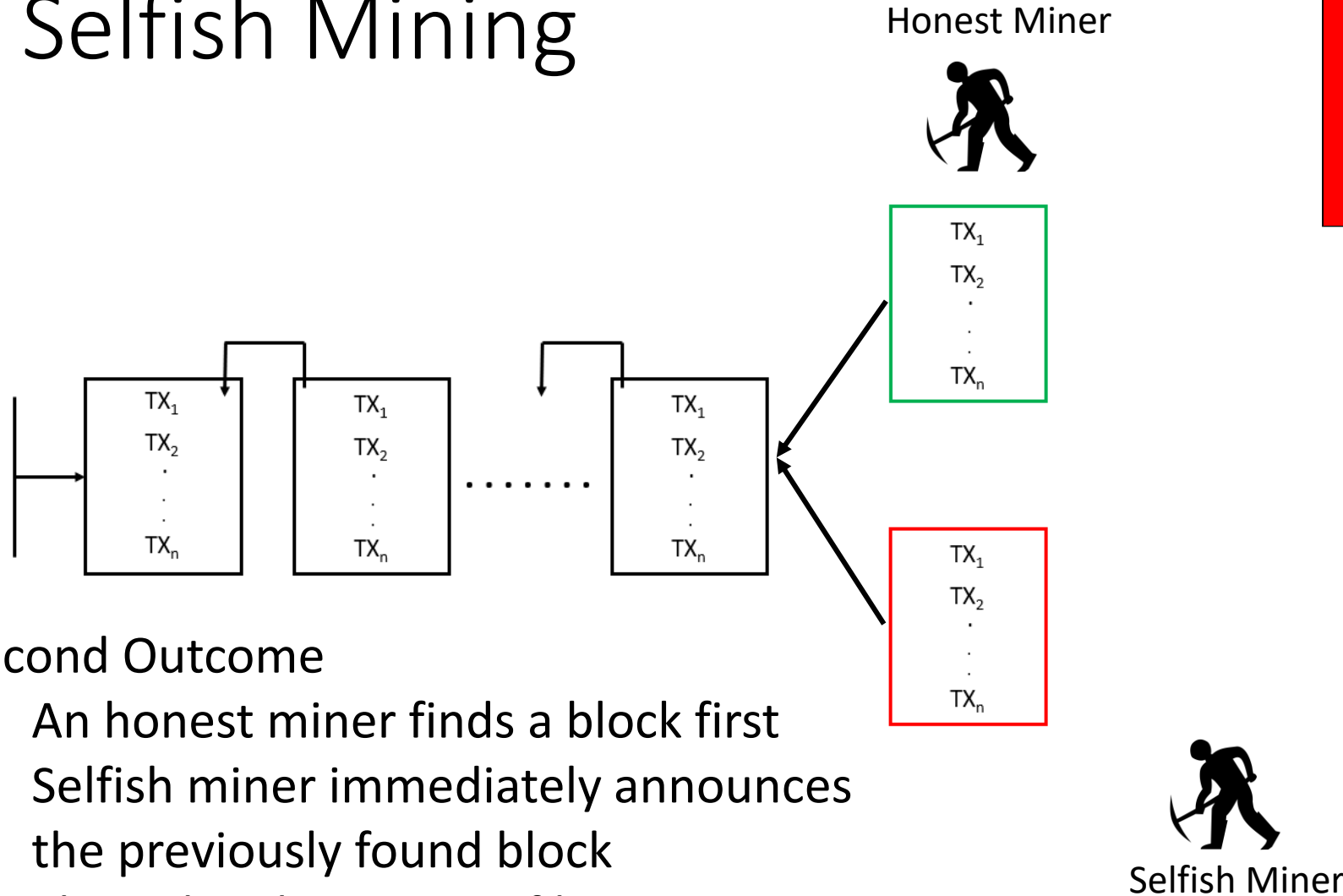
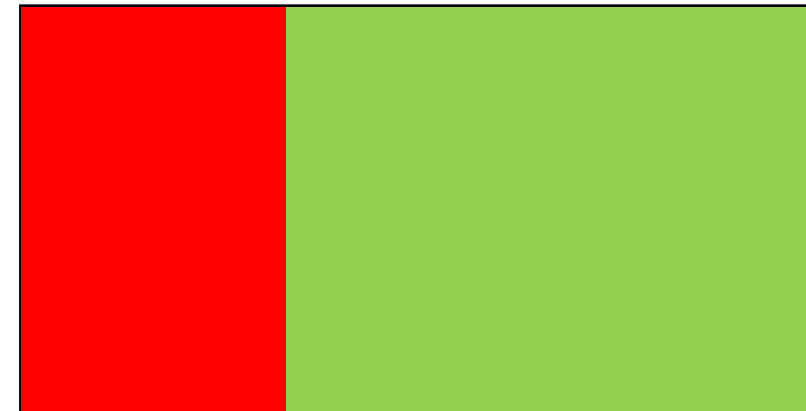
- Second Outcome
 - An honest miner finds a block first

Selfish Mining



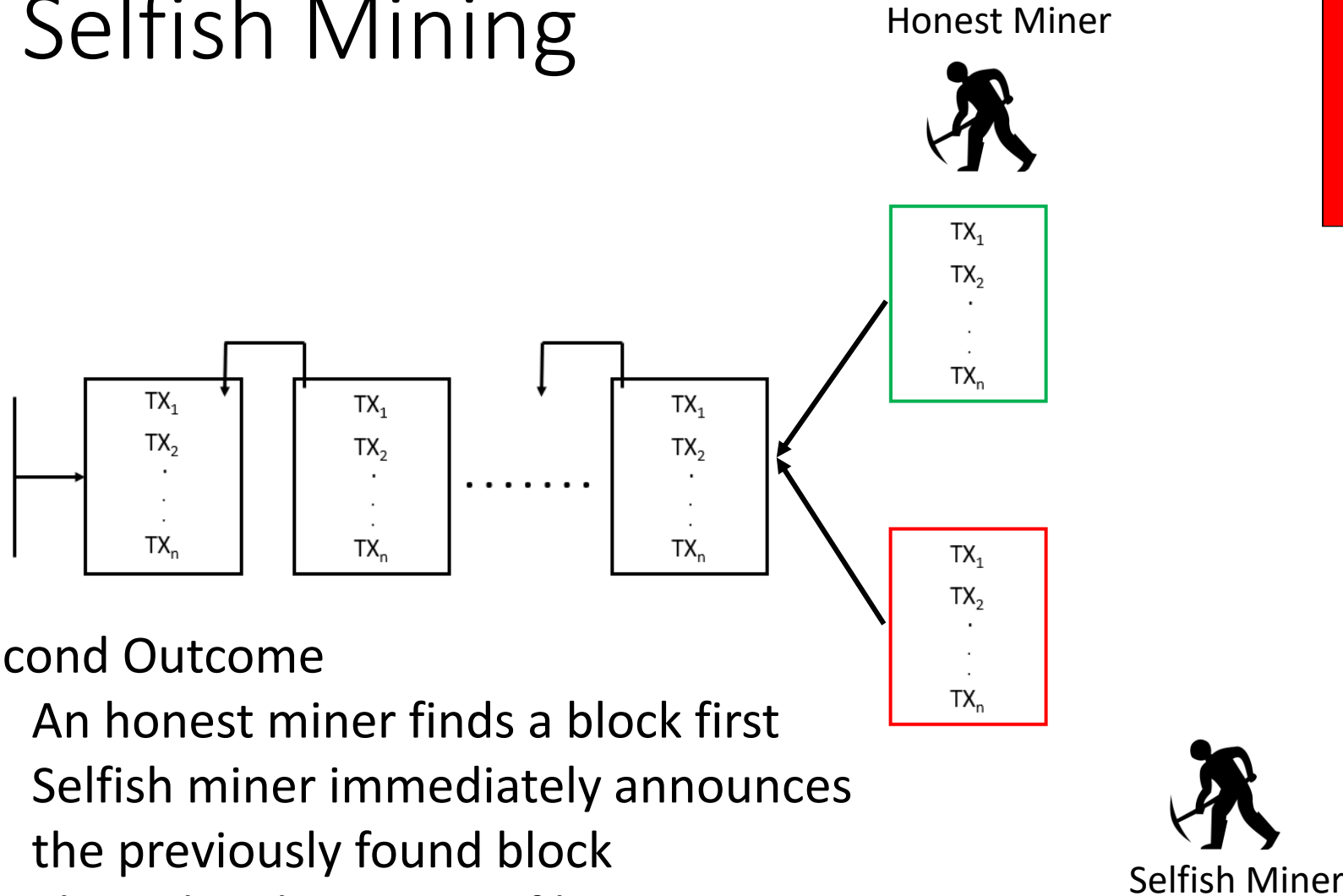
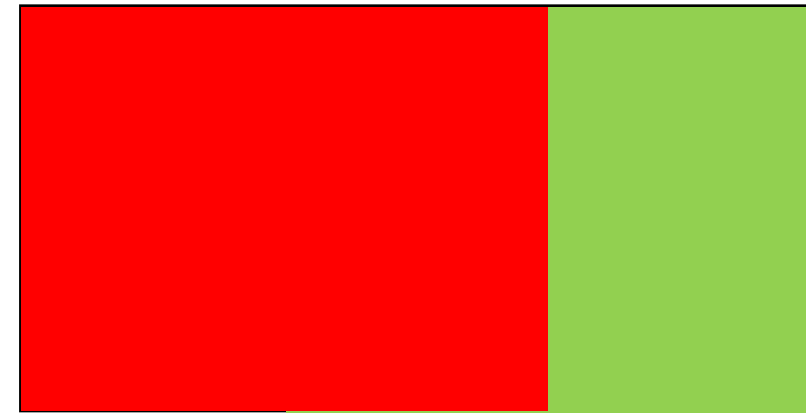
- Second Outcome
 - An honest miner finds a block first
 - Selfish miner immediately announces the previously found block

Selfish Mining



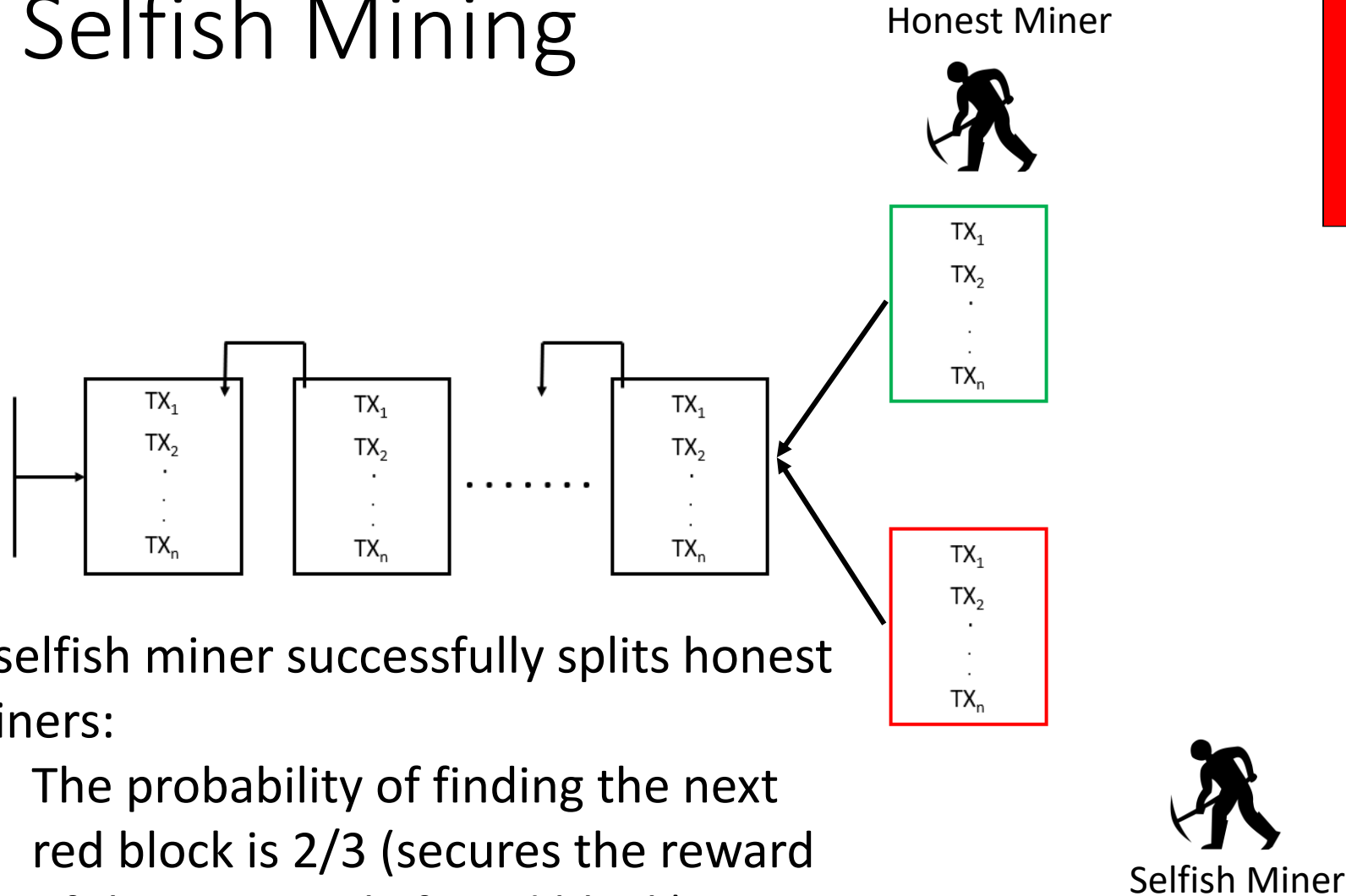
- Second Outcome
 - An honest miner finds a block first
 - Selfish miner immediately announces the previously found block
 - This splits the power of honest miners

Selfish Mining



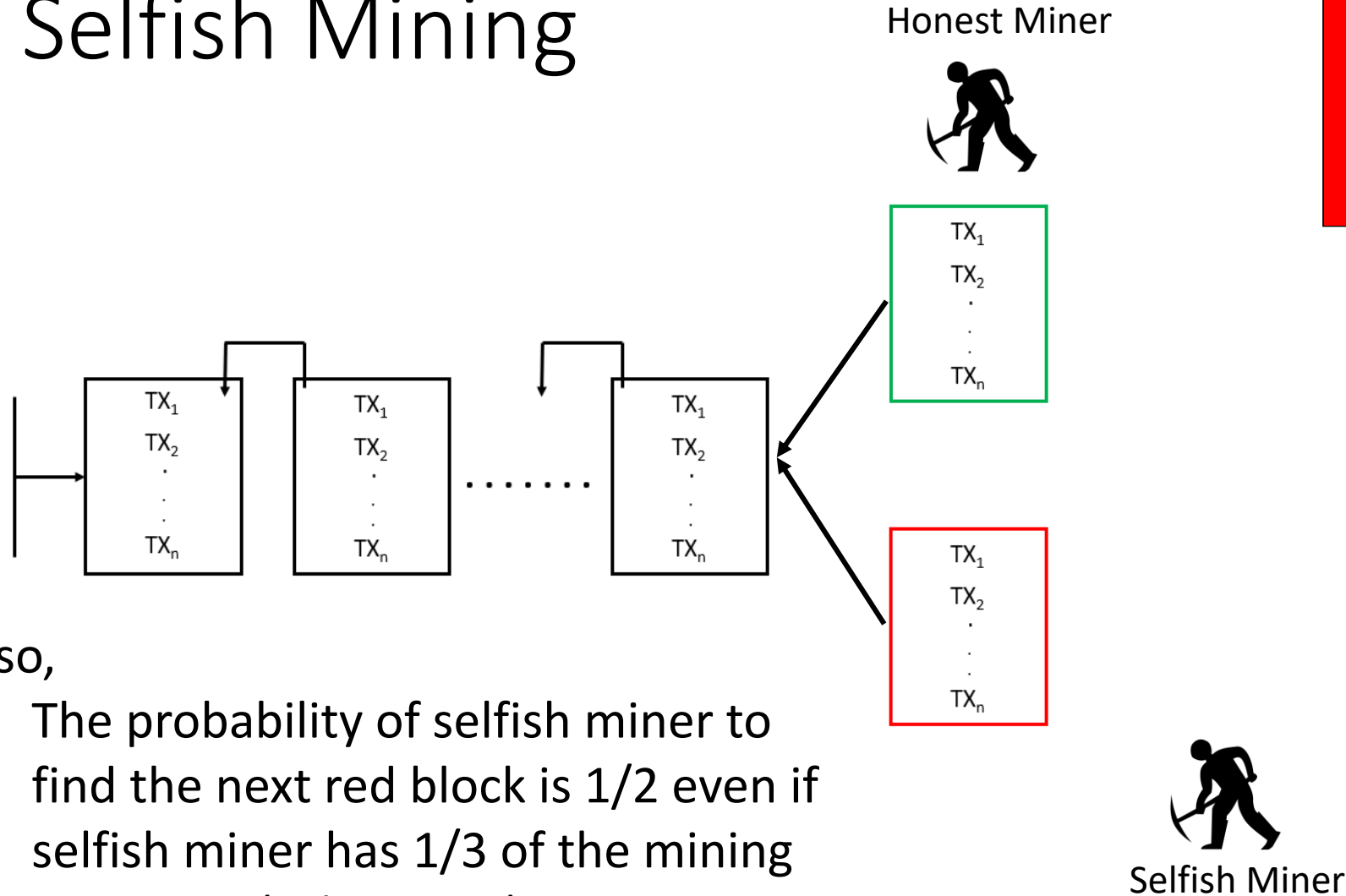
- Second Outcome
 - An honest miner finds a block first
 - Selfish miner immediately announces the previously found block
 - This splits the power of honest miners

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 = 4200 Txns
7 Txns/ 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)

DSL at UCSB

Overview

Overview

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

DSL at UCSB

Overview

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

Mine once, publish txns many times

BitcoinNG

DSL at UCSB

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

DSL at UCSB

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

Shard txns across different committees

Elastico

BitcoinNG (Next Generation)

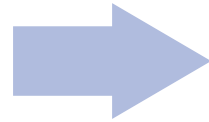
BitcoinNG (Next Generation)

Observation: In Bitcoin,
blocks provide two
purpose:

consensus and
txn verification

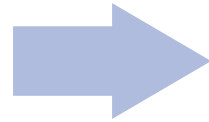
BitcoinNG (Next Generation)

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



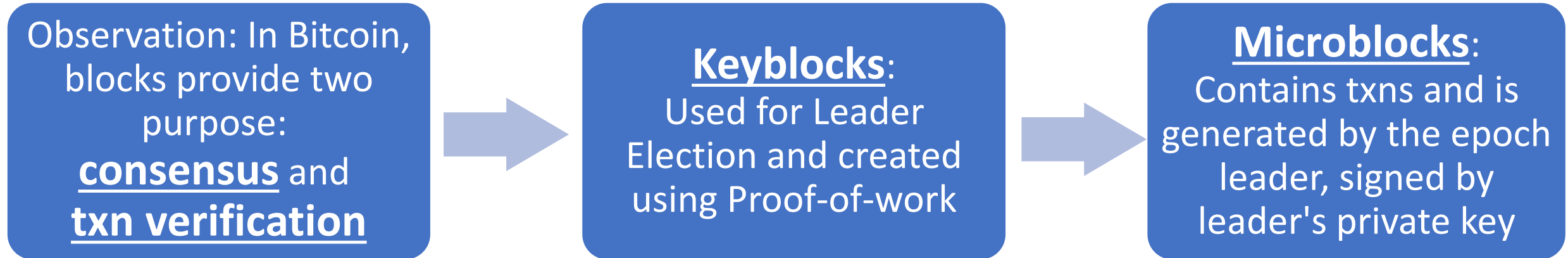
BitcoinNG (Next Generation)

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



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

BitcoinNG (Next Generation)



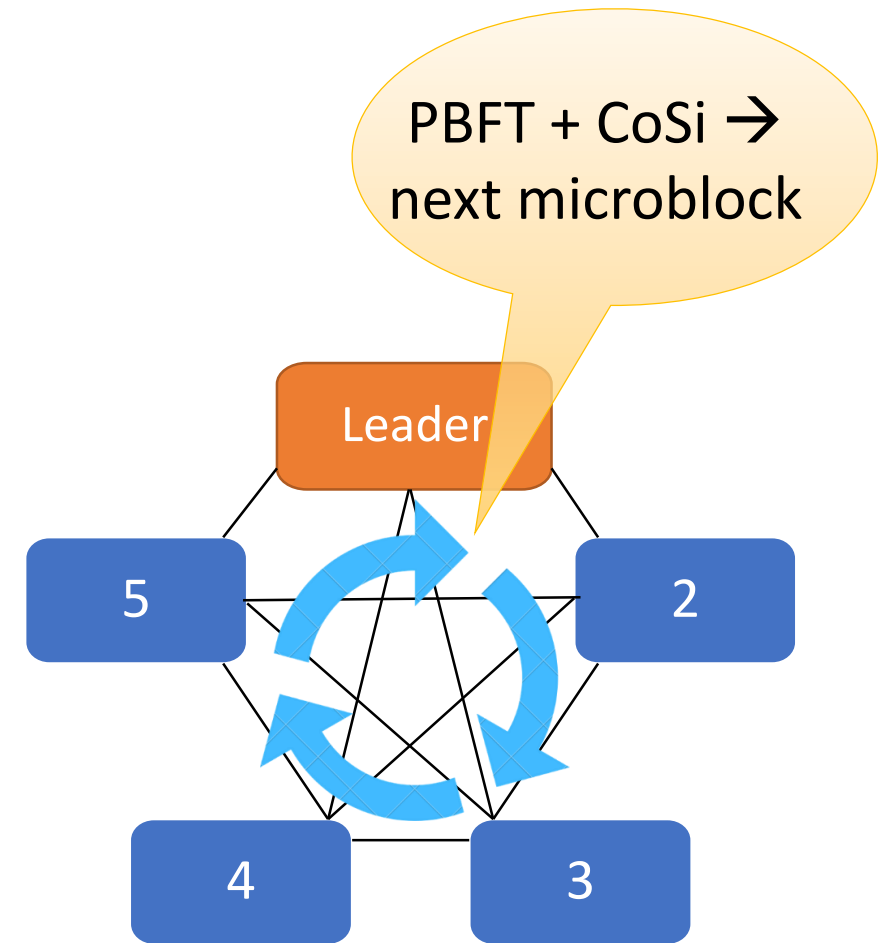
- Key-block miner → **leader** till next key-block is mined
- Leader publishes micro-blocks while in tenure

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



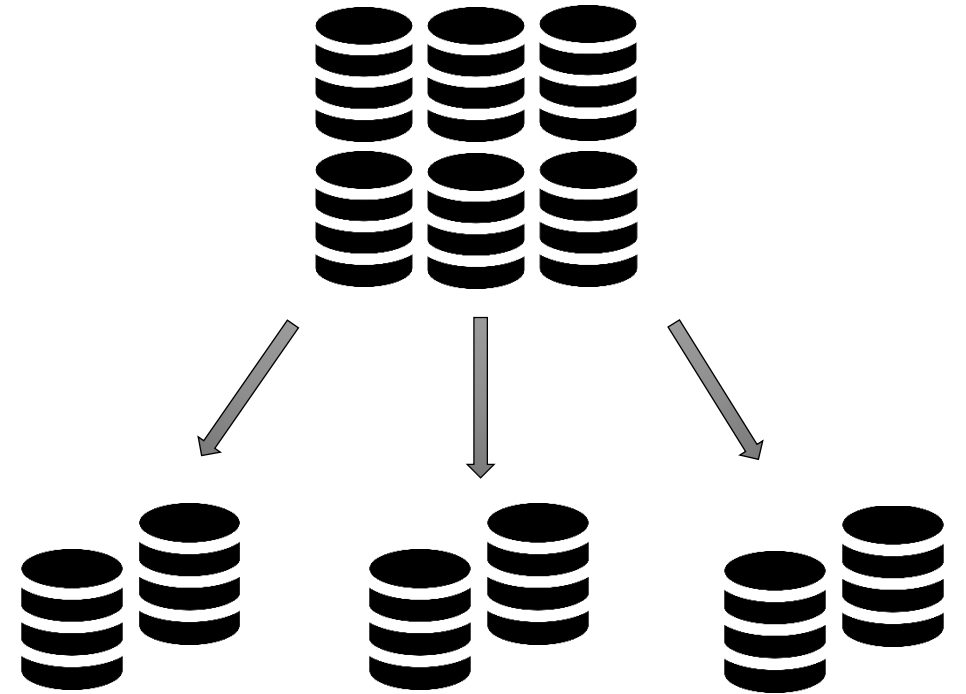
Elastico

Elastico

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

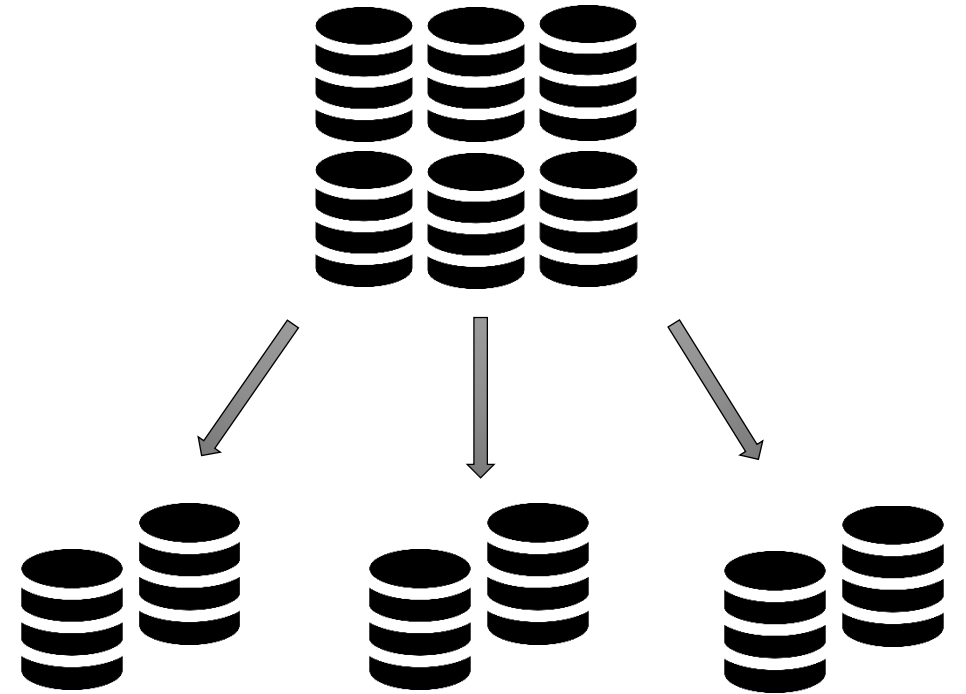
Elastico

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



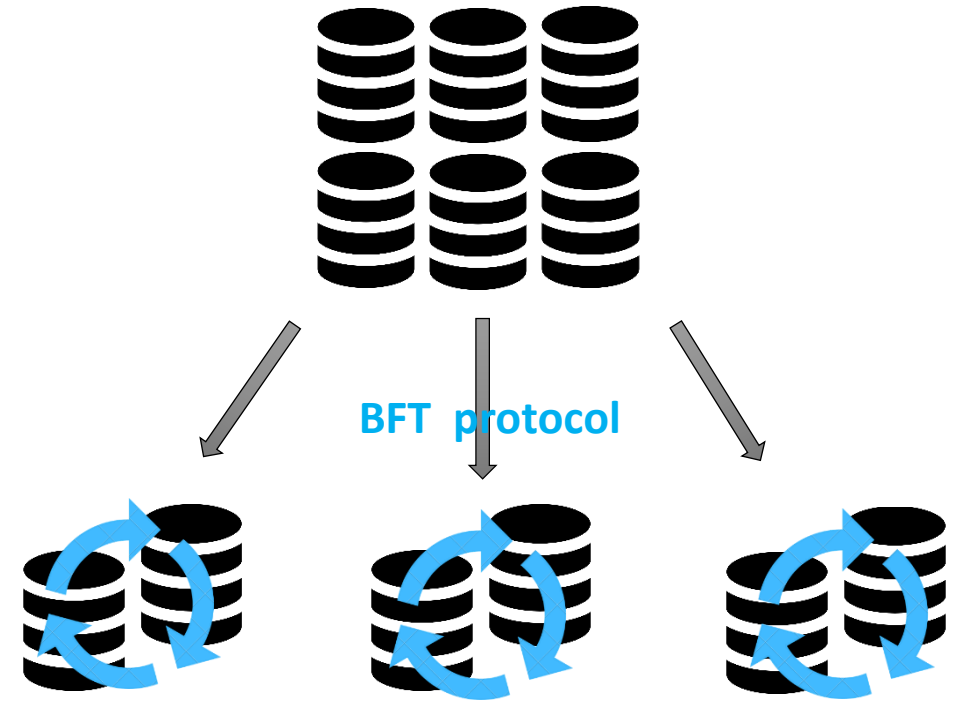
Elastico

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



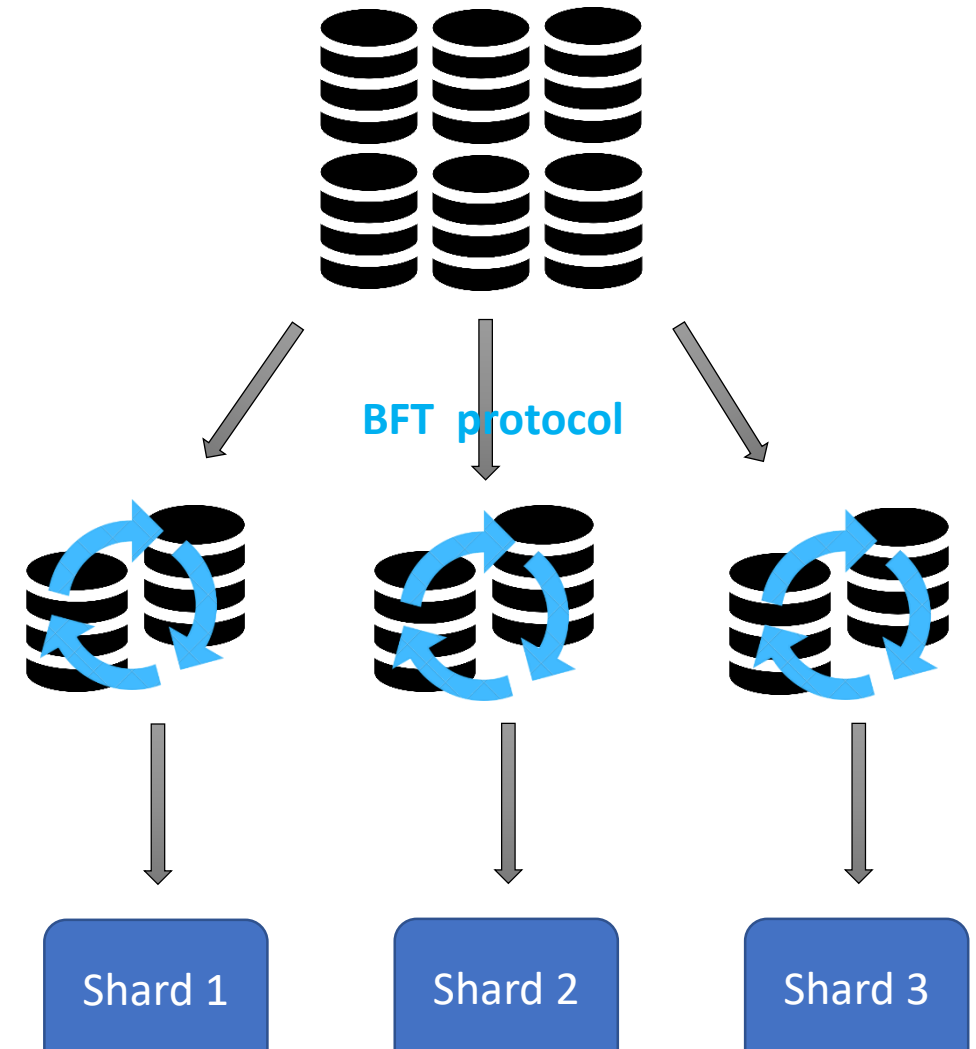
Elastico

- 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



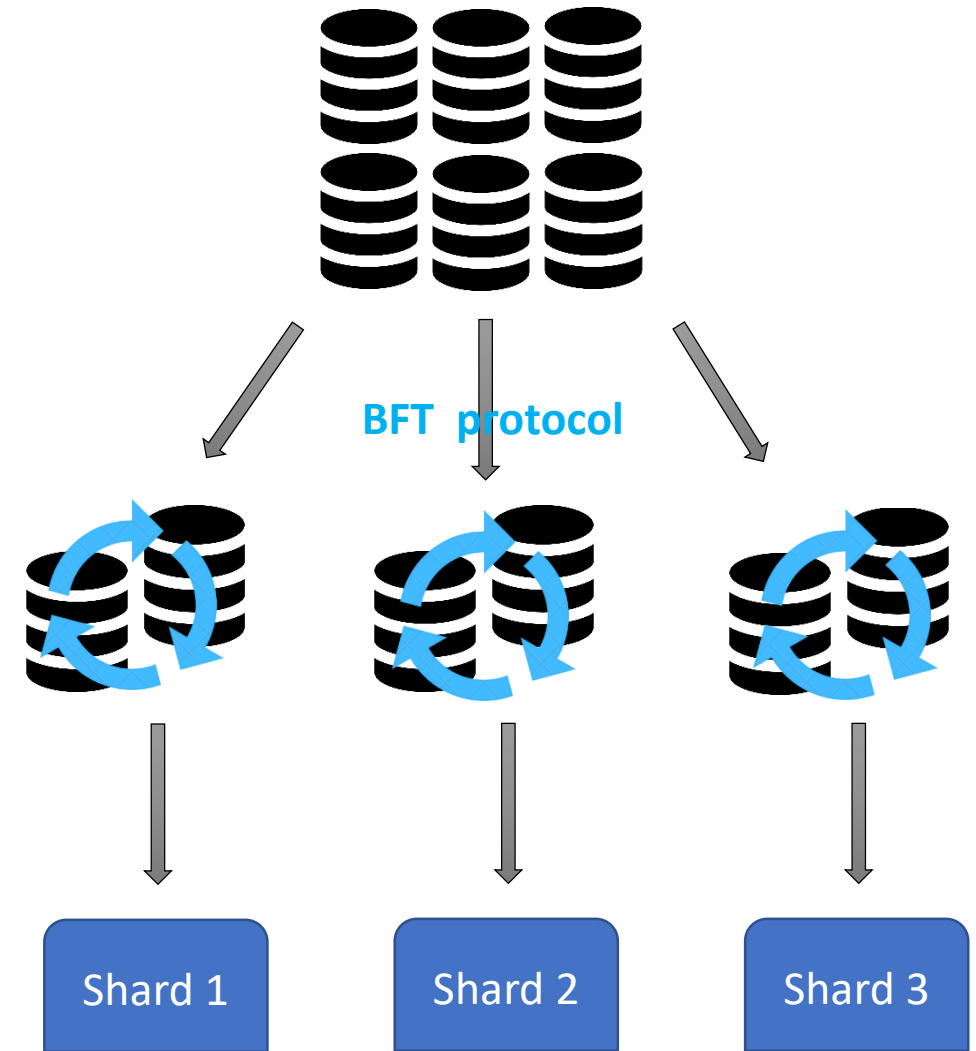
Elastico

- 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



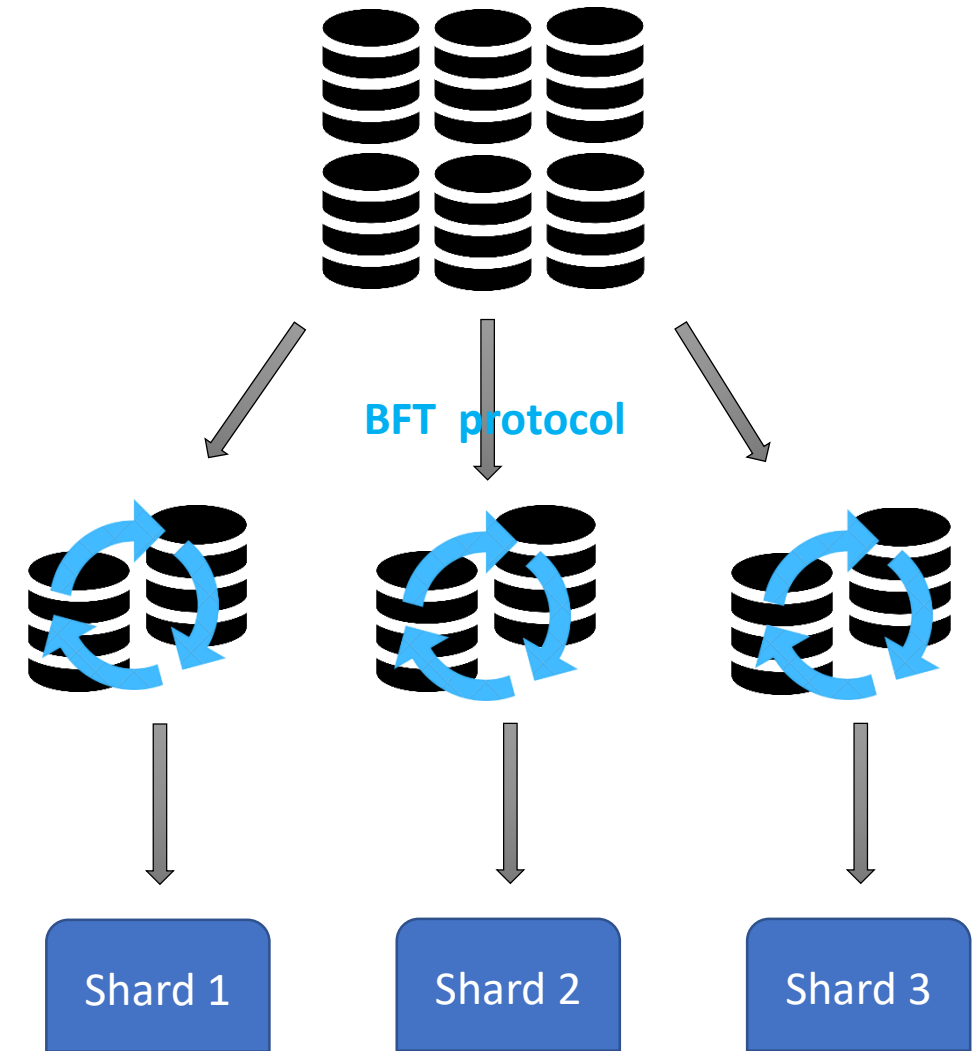
Elastico

- 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



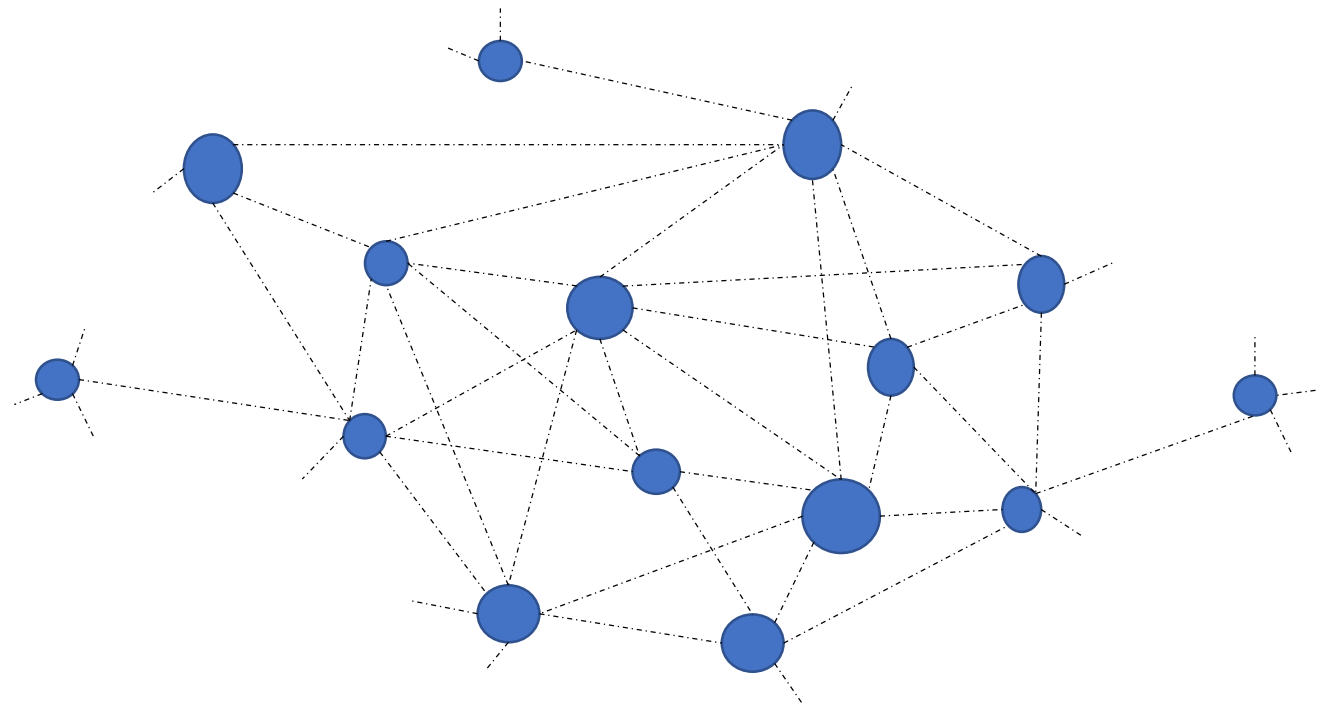
Elastico

- 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

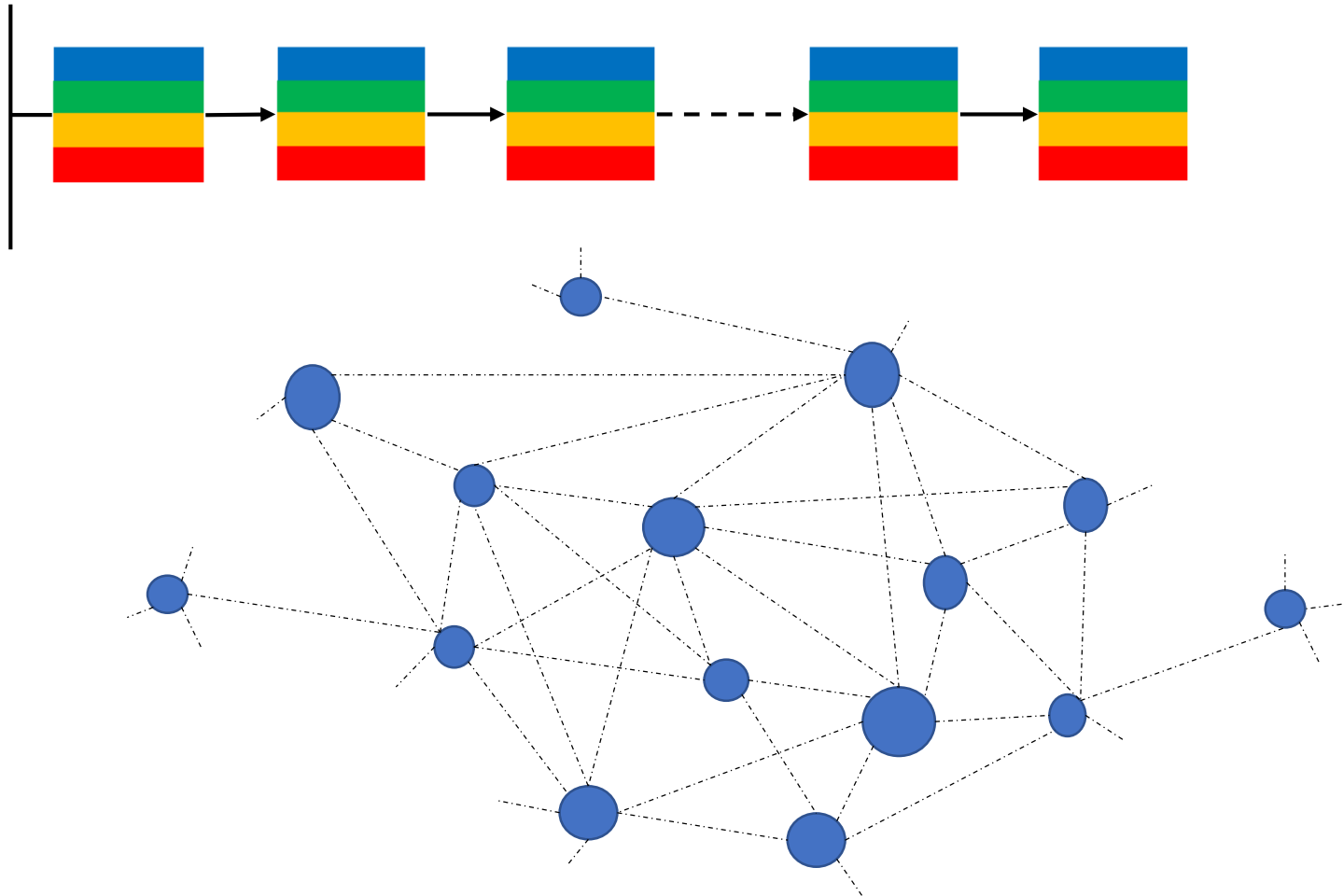


Sharding as a Scalability Solution

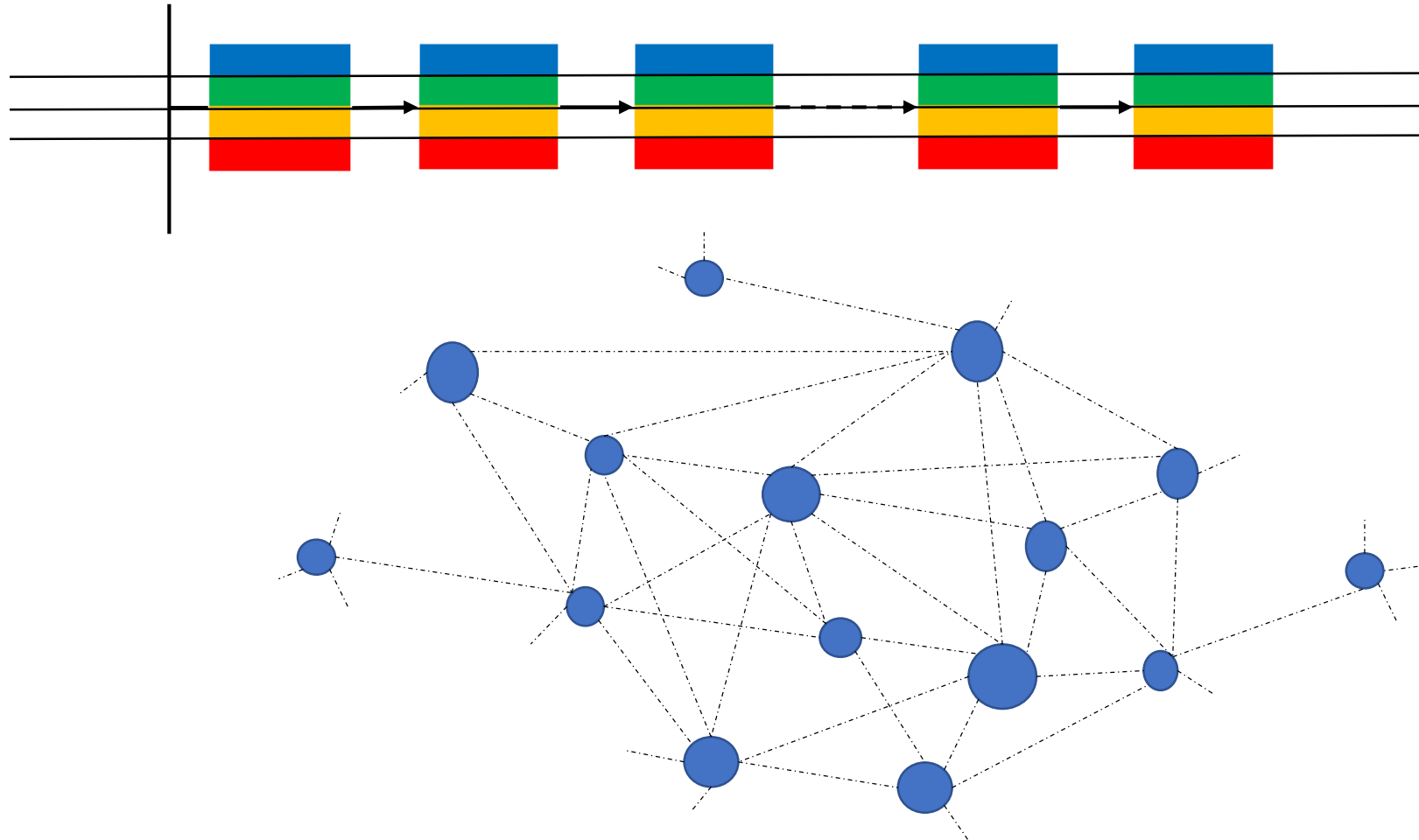
Sharding as a Scalability Solution



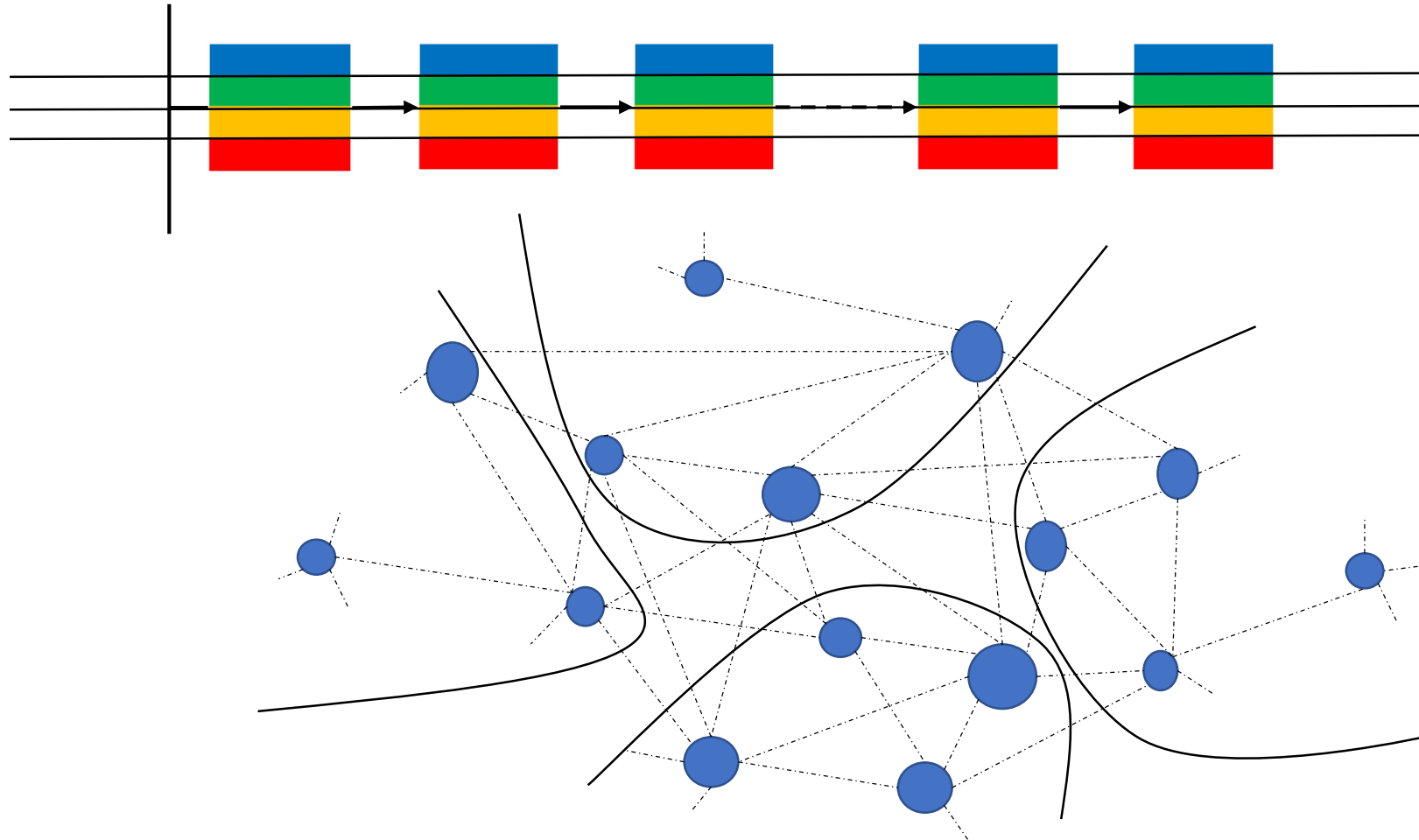
Sharding as a Scalability Solution



Sharding as a Scalability Solution

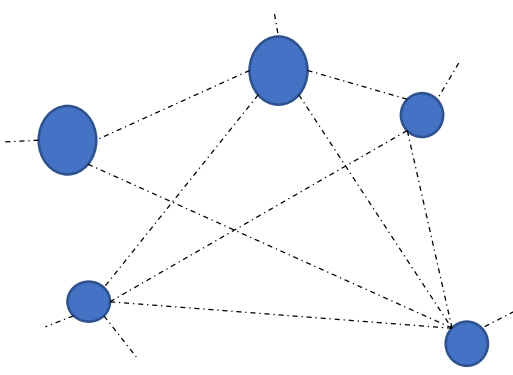
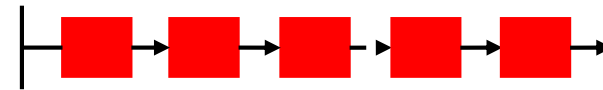
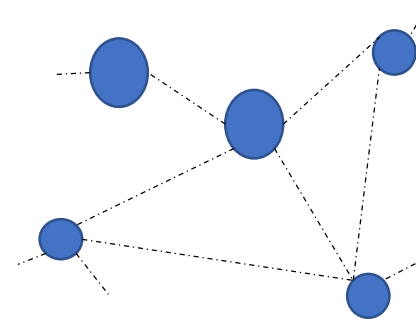
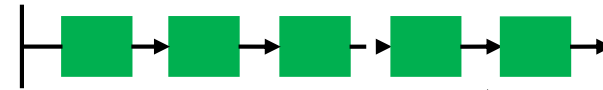
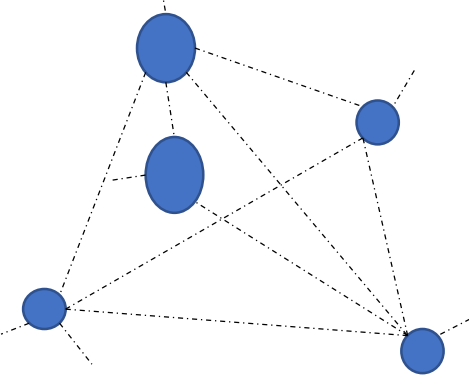
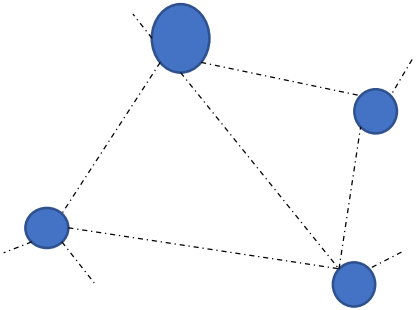
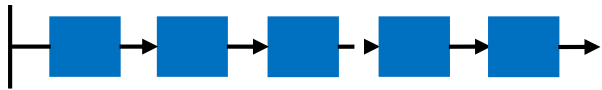


Sharding as a Scalability Solution

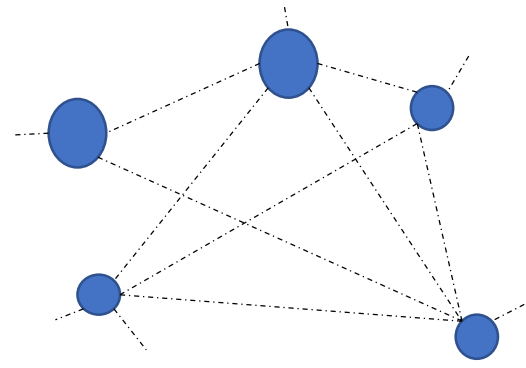
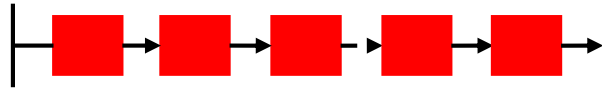
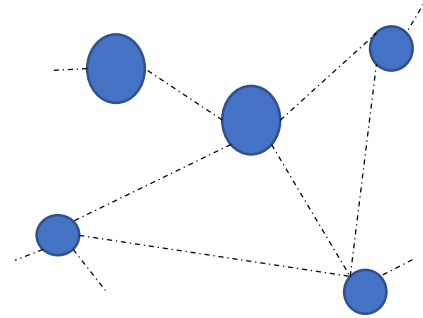
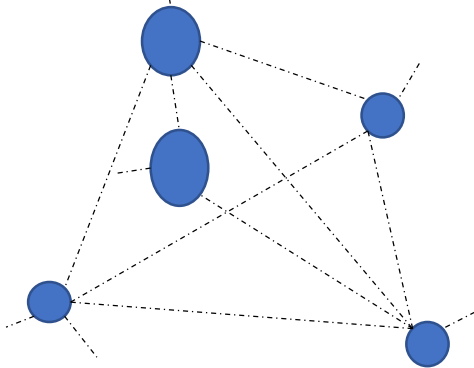
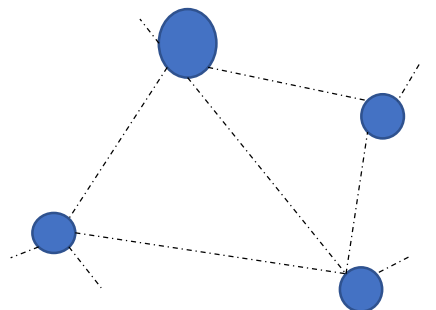
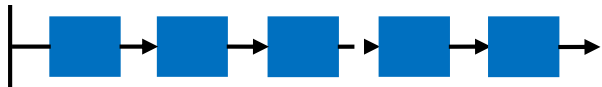


Sharding as a Scalability Solution

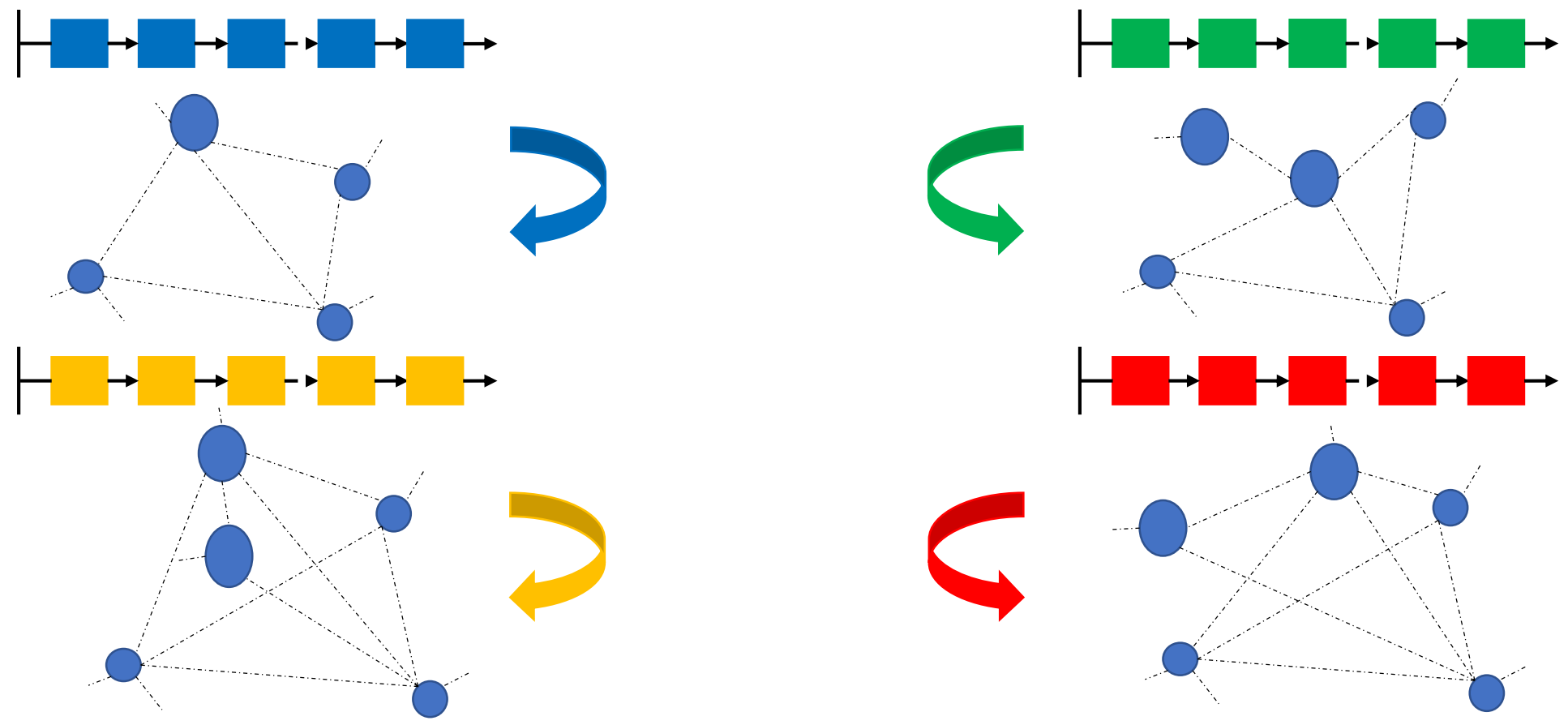
Sharding as a Scalability Solution



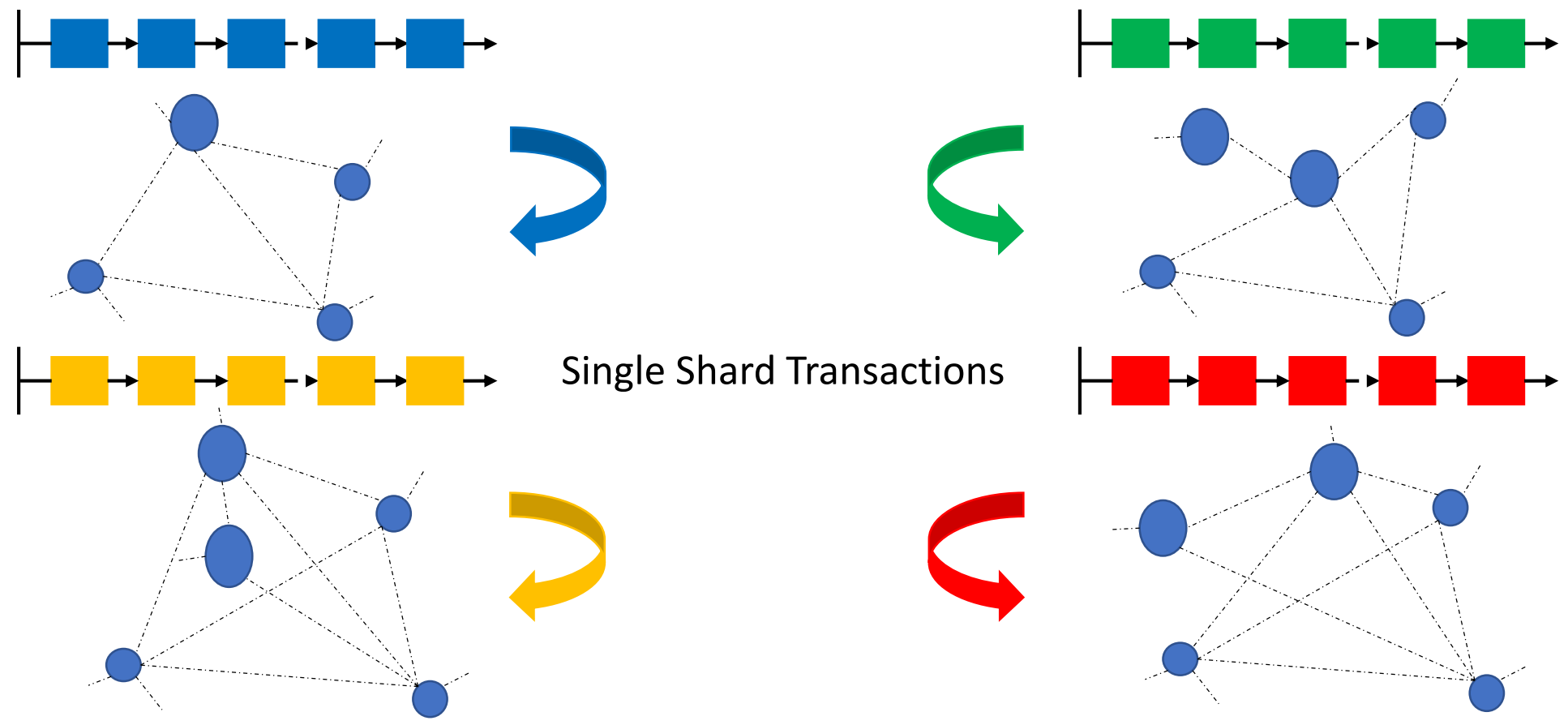
Classes of Transactions



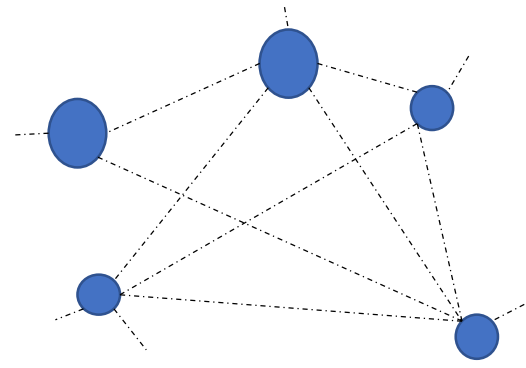
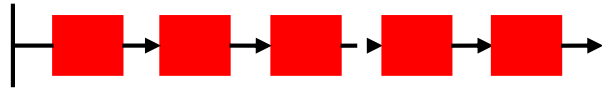
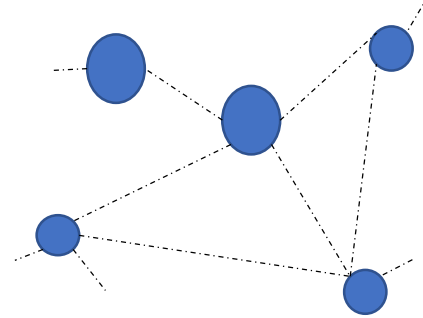
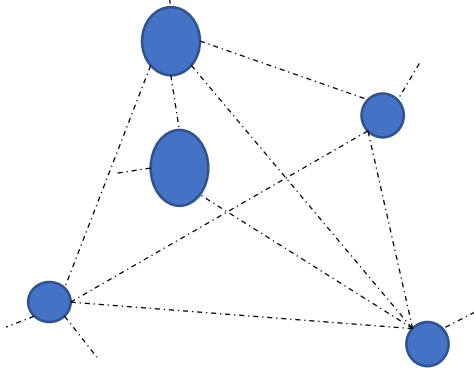
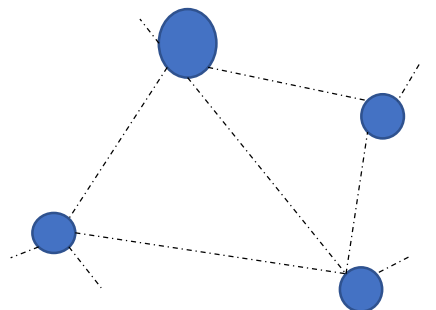
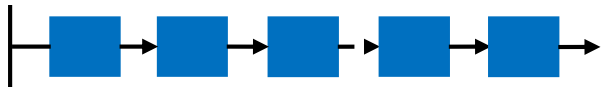
Classes of Transactions



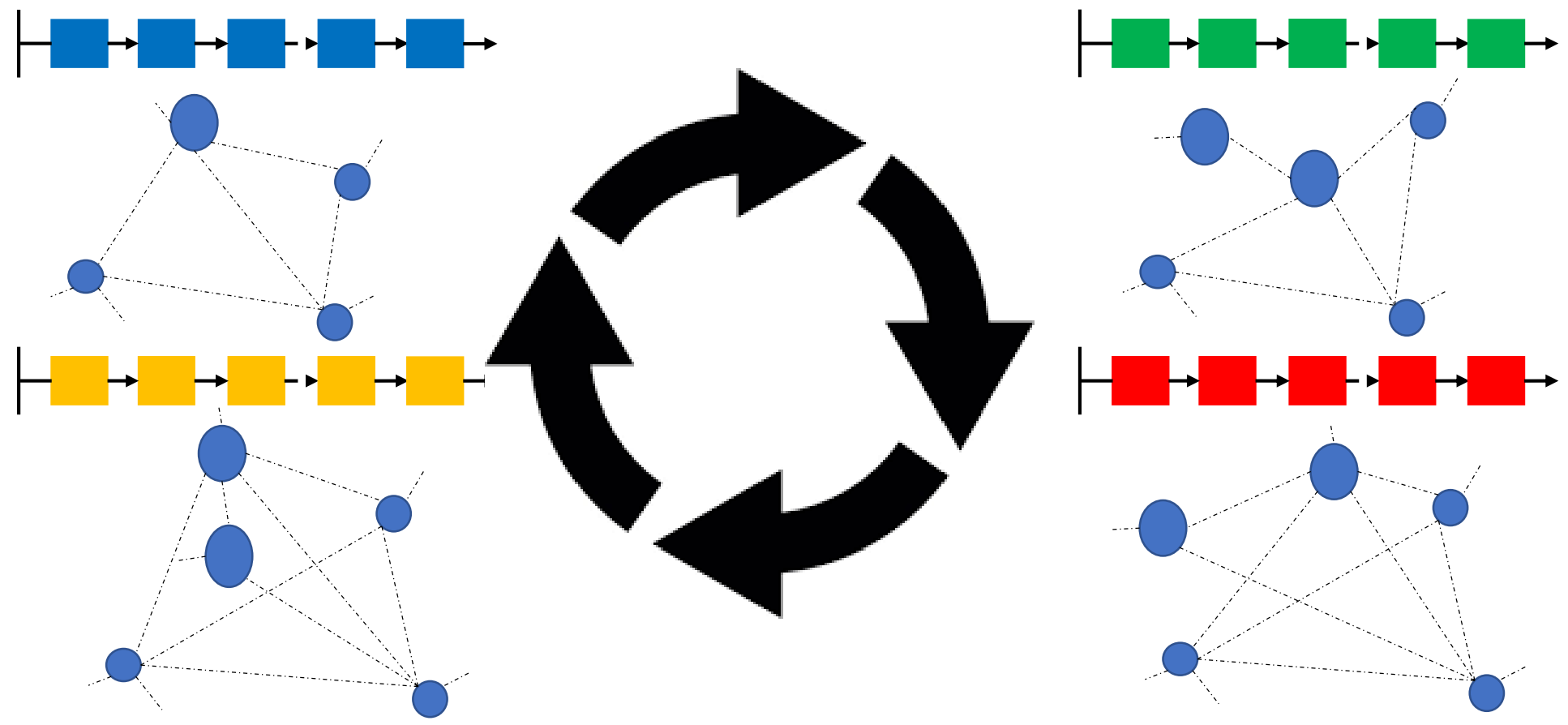
Classes of Transactions



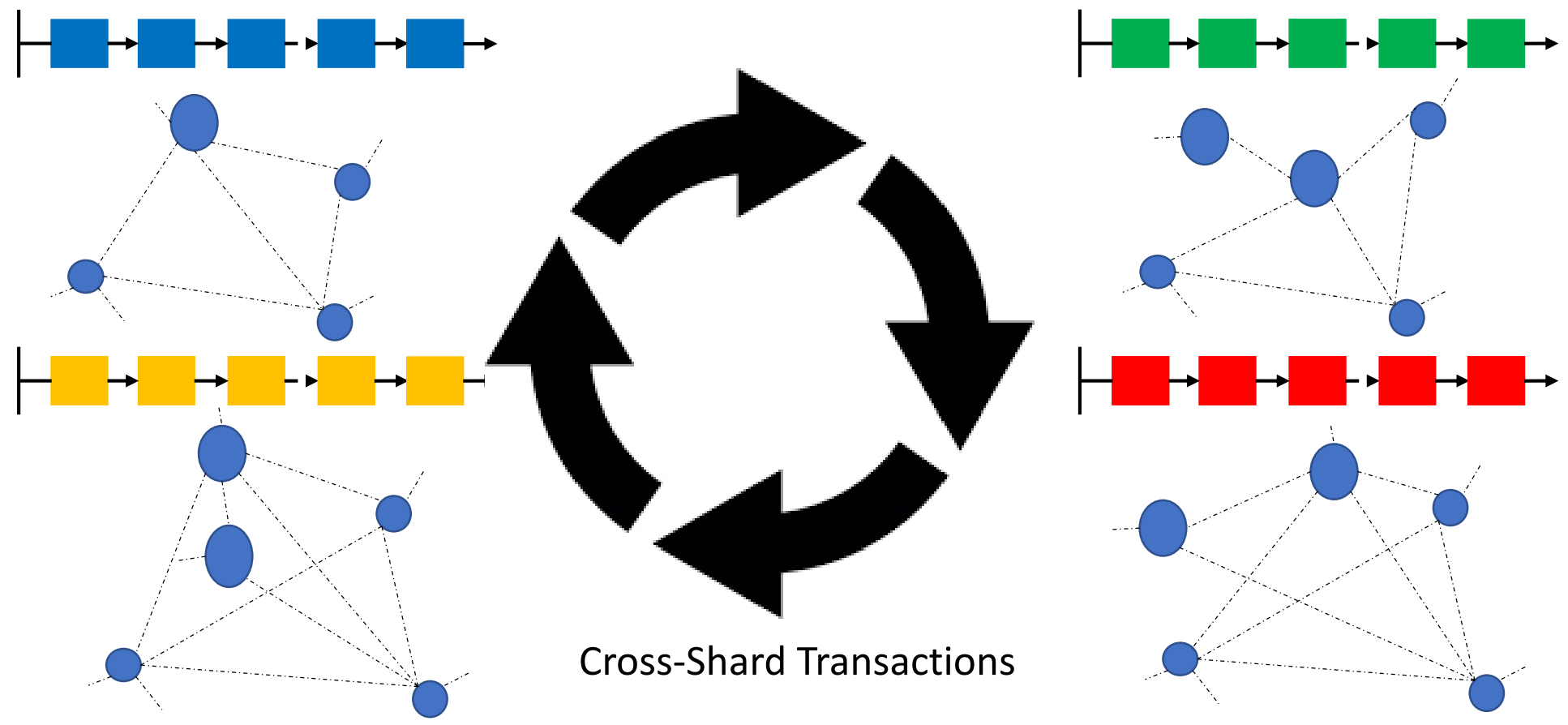
Classes of Transactions



Classes of Transactions

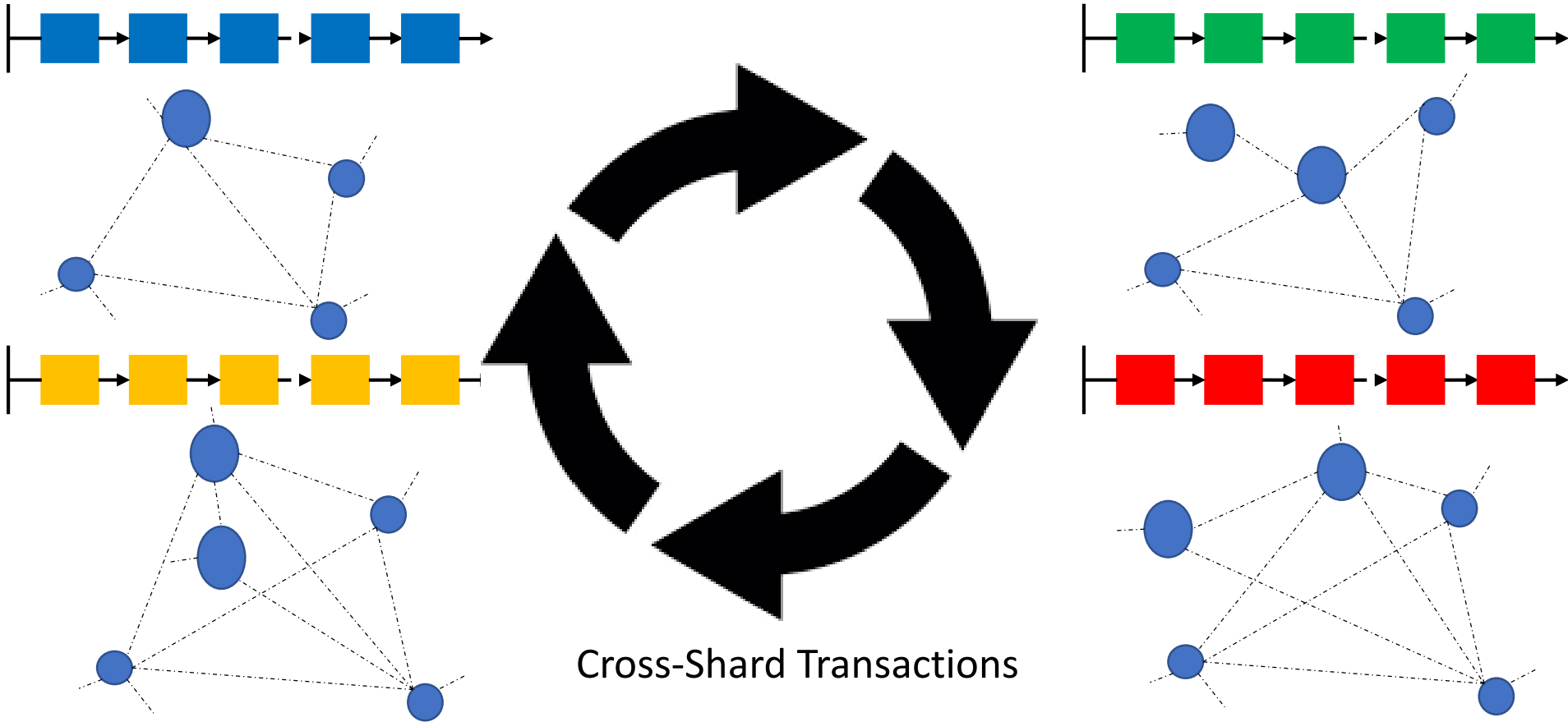


Classes of Transactions



Classes of Transactions

Requires Atomic Cross-Shard Commitment Protocol



The Landscape

The Landscape

Cryptocurrencies: [2225](#) • Markets: [18851](#) • Market Cap: [\\$257,486,187,861](#) • 24h Vol: [\\$66,548,083,112](#) • BTC Dominance: [55.4%](#)



[Rankings](#) [Tools](#) [Resources](#) [Blog](#) [...](#)

Top 100 Cryptocurrencies by Market Capitalization

Cryptocurrencies ▾		Exchanges ▾		Watchlist		USD ▾	Next 100 →	View All
#	Name	Market Cap	Price	Volume (24h)	Circulating Supply	Change (24h)	Price Graph (7d)	
1	Bitcoin	\$142,627,334,795	\$8,036.77	\$19,138,268,181	17,746,837 BTC	3.15%		...
2	Ethereum	\$26,732,290,299	\$251.25	\$8,364,736,132	106,397,463 ETH	1.70%		...
3	XRP	\$17,876,222,703	\$0.423217	\$1,658,461,942	42,238,947,941 XRP *	1.25%		...
4	Litecoin	\$7,281,728,951	\$117.21	\$5,141,138,982	62,124,551 LTC	6.28%		...
5	Bitcoin Cash	\$7,157,820,741	\$401.55	\$1,572,103,916	17,825,688 BCH	2.02%		...

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

The Landscape

Cryptocurrencies: [2225](#) • Markets: [18851](#) • Market Cap: [\\$257,486,187,861](#) • 24h Vol: [\\$66,548,083,112](#) • BTC Dominance: [55.4%](#)



[Rankings](#) [Tools](#) [Resources](#) [Blog](#) [...](#)

Top 100 Cryptocurrencies by Market Capitalization

Cryptocurrencies ▾ [Exchanges ▾](#) [Watchlist](#) USD ▾ [Next 100 →](#) [View All](#)

#	Name	Market Cap	Price	Volume (24h)	Circulating Supply	Change (24h)	Price Graph (7d)
1	Bitcoin	\$142,627,334,795	\$8,036.77	\$19,138,268,181	17,746,837 BTC	3.15%	
2	Ethereum	\$26,732,290,299	\$251.25	\$8,364,736,132	106,397,463 ETH	1.70%	
3	XRP	\$17,876,222,703	\$0.423217	\$1,658,461,942	42,238,947,941 XRP *	1.25%	
4	Litecoin	\$7,281,728,951	\$117.21	\$5,141,138,982	62,124,551 LTC	6.28%	
5	Bitcoin Cash	\$7,157,820,741	\$401.55	\$1,572,103,916	17,825,688 BCH	2.02%	

The Landscape

Cryptocurrencies: 2225 • Markets: 18851



[Rankings](#)
[Tools](#)
[Resources](#)
[Blog](#)
⋮

Top 100 Cryptocurrencies by Market Capitalization

[Cryptocurrencies](#) ▾
 [Exchanges](#) ▾
 [Watchlist](#)

[USD](#) ▾
 [Next 100](#) →
 [View All](#)

#	Name	Market Cap	Price	Volume (24h)	Circulating Supply	Change (24h)	Price Graph (7d)
1	Bitcoin	\$142,627,334,795	\$8,036.77	\$19,138,268,181	17,746,837 BTC	3.15%	
2	Ethereum	\$26,732,290,299	\$251.25	\$8,364,736,132	106,397,463 ETH	1.70%	
3	XRP	\$17,876,222,703	\$0.423217	\$1,658,461,942	42,238,947,941 XRP *	1.25%	
4	Litecoin	\$7,281,728,951	\$117.21	\$5,141,138,982	62,124,551 LTC	6.28%	
5	Bitcoin Cash	\$7,157,820,741	\$401.55	\$1,572,103,916	17,825,688 BCH	2.02%	

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

The Landscape

Cryptocurrencies: **2225** • Markets: **18851**

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

Cryptocurrencies ▾ Exchanges ▾ Watchlist USD ▾ Next 100 → View All

#	Name	Market Cap	Price	Volume (24h)	Circulating Supply	Change (24h)	Price Graph (7d)
1	Bitcoin	\$142,627,334,795	\$8,036.77	\$19,138,268,181	17,746,837 BTC	3.15%	
2	Ethereum	\$26,732,290,299	\$251.25	\$8,364,736,132	106,397,463 ETH	1.70%	
3	XRP	\$17,876,222,703	\$0.423217	\$1,658,461,942	42,238,947,941 XRP *	1.25%	
4	Litecoin	\$7,281,728,951	\$117.21	\$5,141,138,982	62,124,551 LTC	6.28%	
5	Bitcoin Cash	\$7,157,820,741	\$401.55	\$1,572,103,916	17,825,688 BCH	2.02%	

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

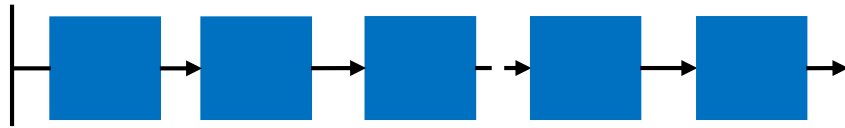
- Thousands of Blockchains
- Tens of thousands of markets
- Exchanges to trade tokens for USD
- Direct token transactions in one blockchain
- Direct token transactions across blockchains, **how?**

The Landscape

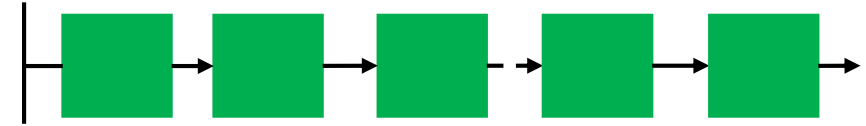
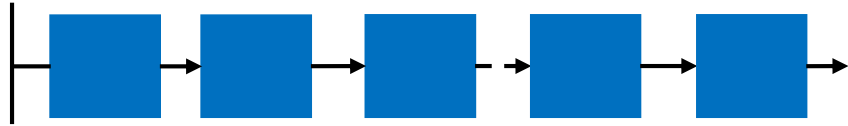
- Thousands of Blockchains
- Tens of thousands of markets
- Exchanges to trade tokens for USD
- Direct token transactions in one blockchain
- Direct token transactions across blockchains, **how?**
- **Cross-chain transactions**

Cross-ChainTransaction Example

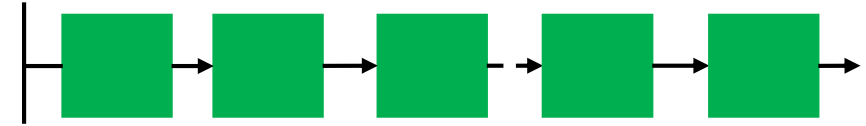
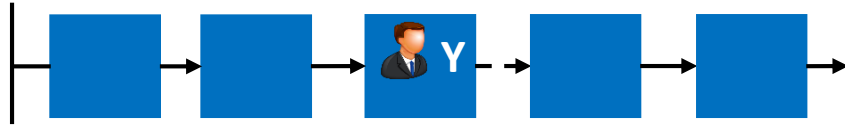
Cross-ChainTransaction Example



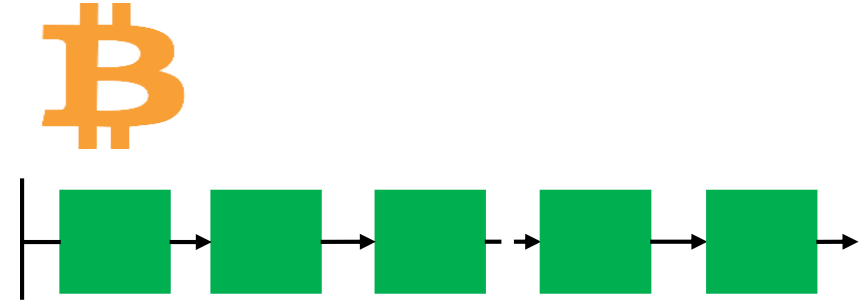
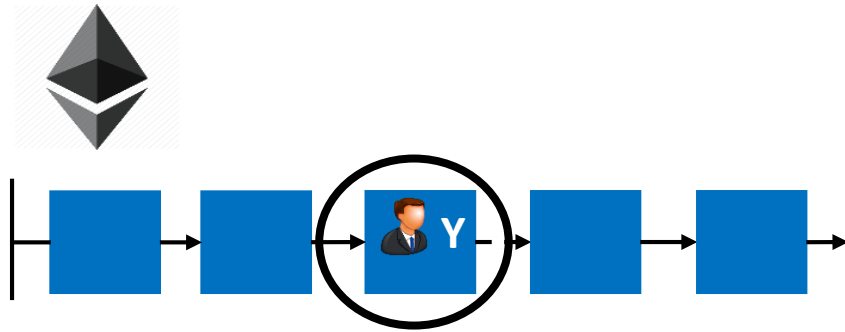
Cross-Chain Transaction Example



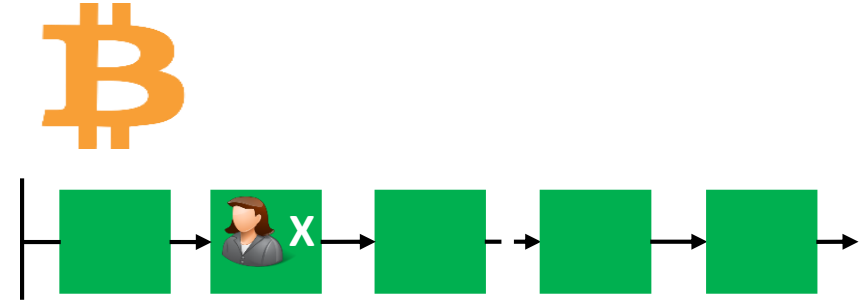
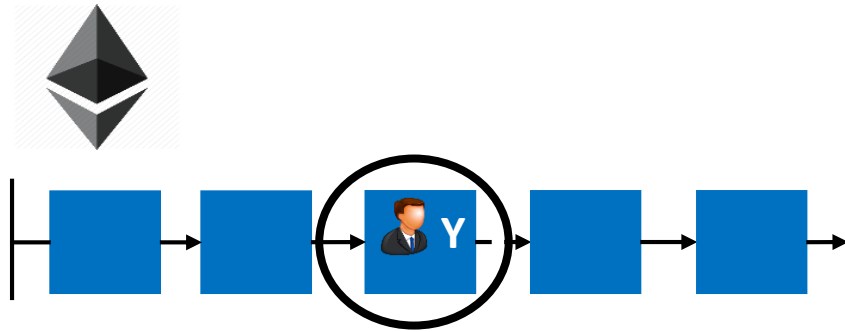
Cross-Chain Transaction Example



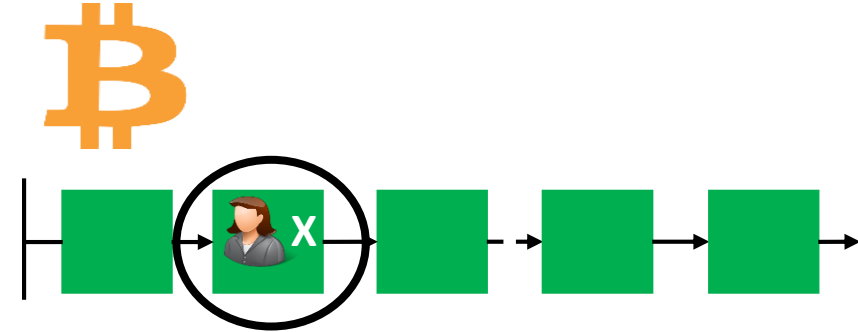
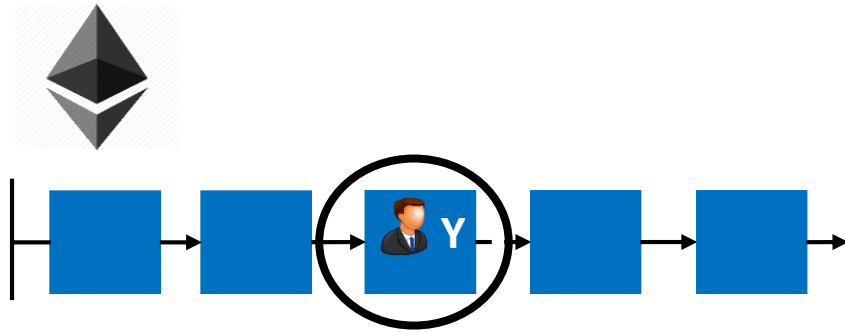
Cross-Chain Transaction Example



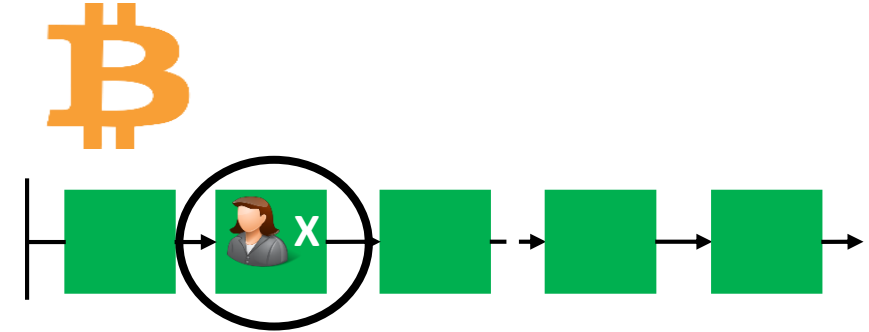
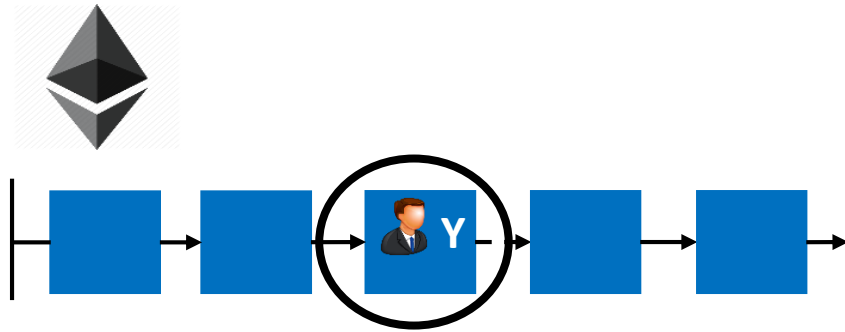
Cross-Chain Transaction Example



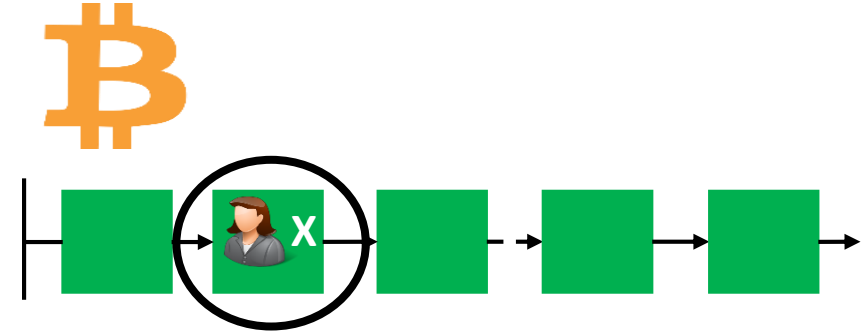
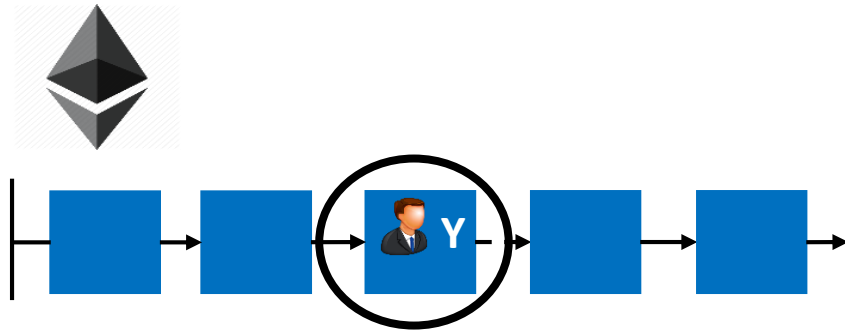
Cross-Chain Transaction Example



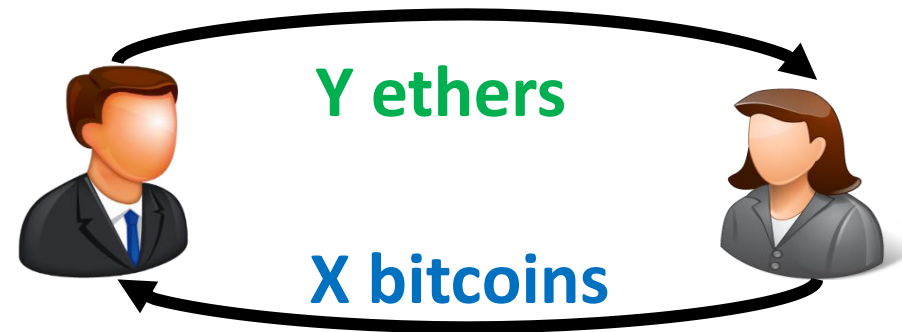
Cross-Chain Transaction Example



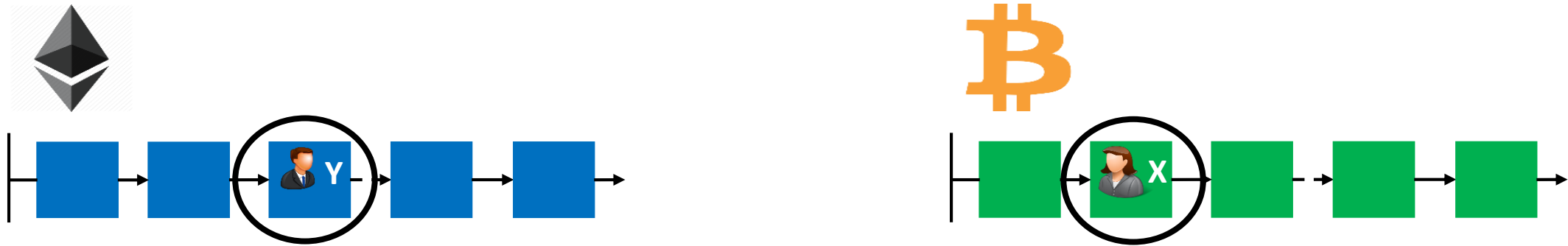
Cross-Chain Transaction Example



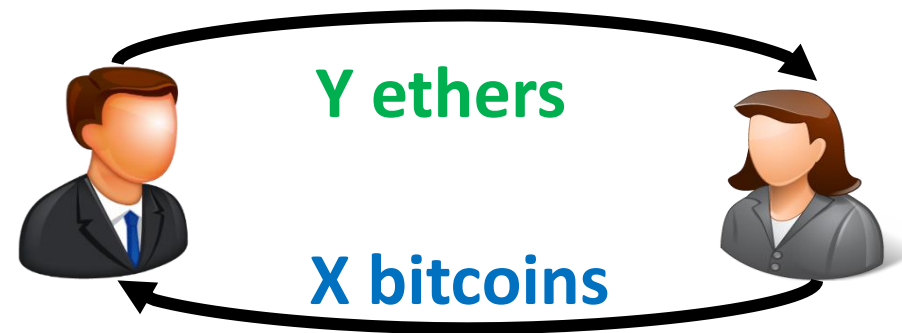
Cross-Chain Transaction Example



Cross-Chain Transaction Example



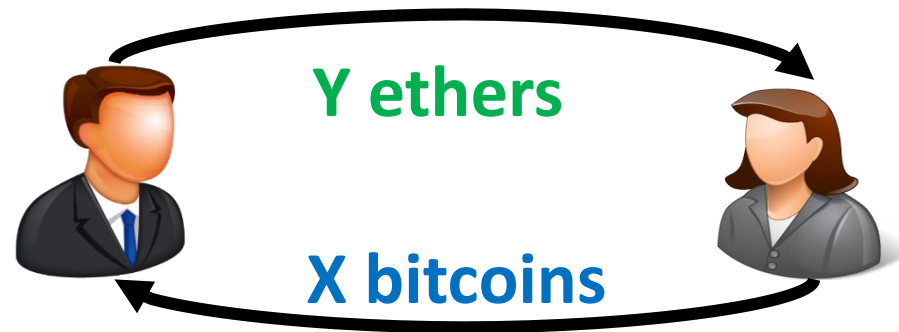
Atomic Cross-Chain Commitment Protocol



Cross-Chain Transaction Example



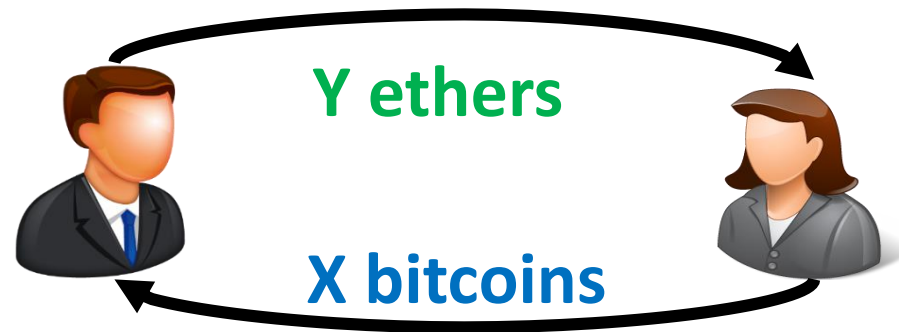
Atomic Cross-Chain Commitment Protocol



Cross-Chain Transaction Example



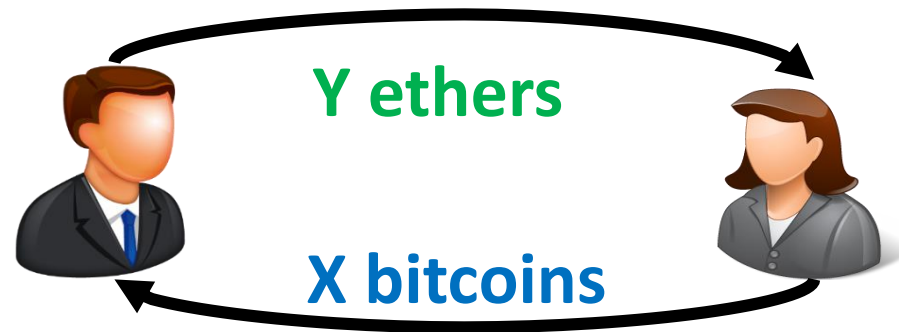
Atomic Cross-Chain Commitment Protocol



Cross-Chain Transaction Example



Atomic Cross-Chain Commitment Protocol



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

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



Bob



Alice


Atomic Swap[Nolan'13, Herlihy'18]

- Alice wants to trade Bitcoin for Ethereum with Bob



Bob



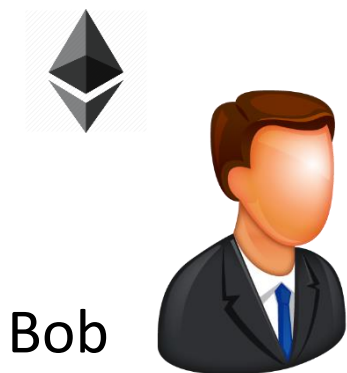
- Create a secret s 
- Calculate its hash $h = H(s)$




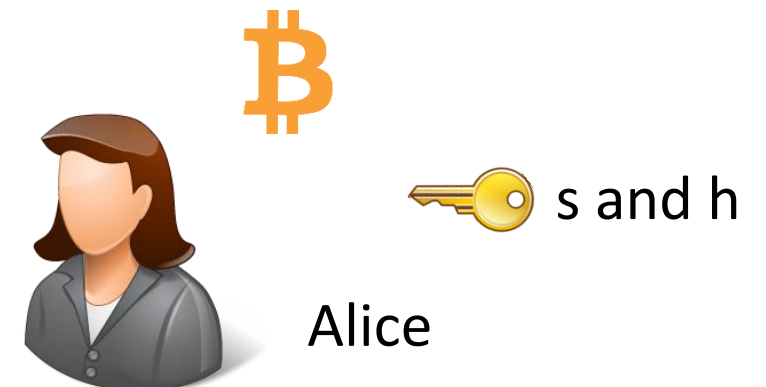
Alice

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



Bob



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



s and h



Alice

Atomic Swap[Nolan'13, Herlihy'18]

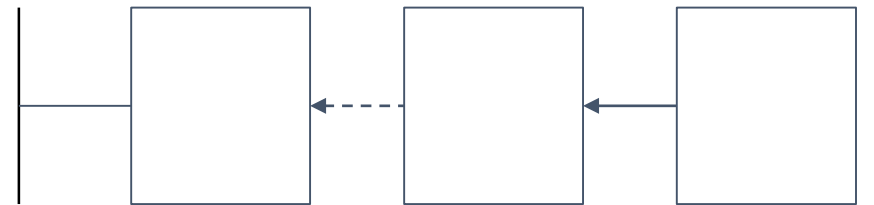
- Alice wants to trade X Bitcoin for Y Ethereum with Bob



Bob



Bitcoin blockchain



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



s and h



Alice

Atomic Swap[Nolan'13, Herlihy'18]

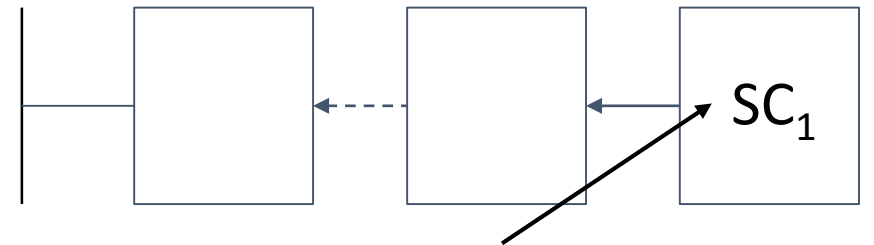
- Alice wants to trade X Bitcoin for Y Ethereum with Bob



Bob



Bitcoin blockchain



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



s and h



Alice

Atomic Swap[Nolan'13, Herlihy'18]

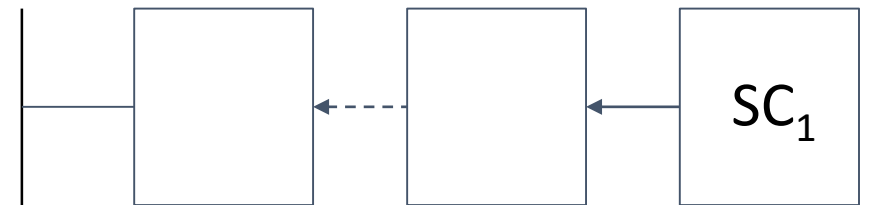
- Now, h is announced in Bitcoin blockchain and made public



Bob



Bitcoin blockchain



Alice's X bitcoins are locked in the smart contract SC_1



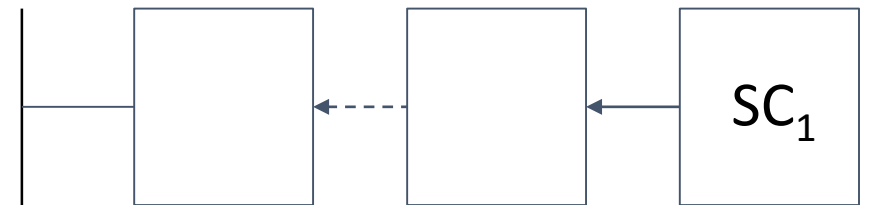
Alice



Atomic Swap[Nolan'13, Herlihy'18]

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

Bitcoin blockchain



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



Bob



Alice's X bitcoins are locked in the smart contract SC_1

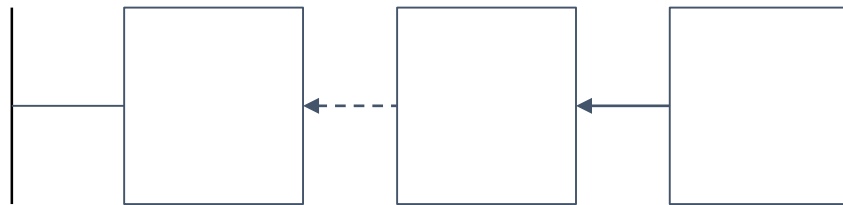


Alice

Atomic Swap[Nolan'13, Herlihy'18]

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

Ethereum blockchain



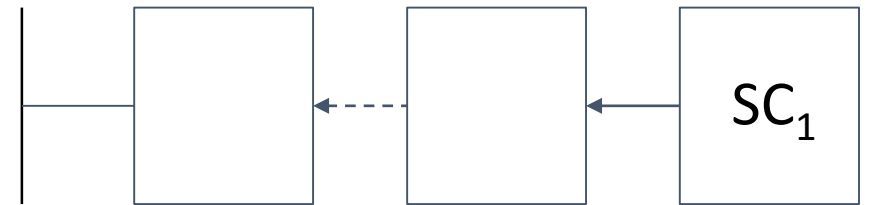
SC_2 Move Y Ethereum to Alice if Alice provides secret s | $h = H(s)$



Bob



Bitcoin blockchain



Alice's X bitcoins are locked in the smart contract SC_1

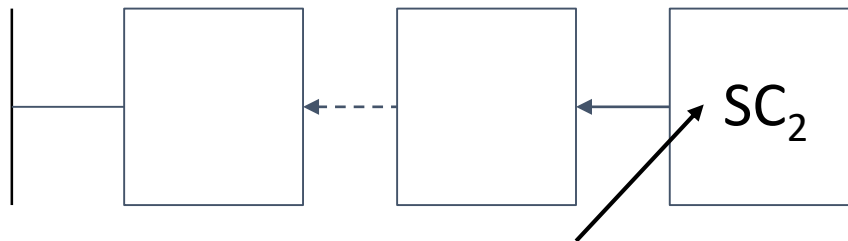


Alice

Atomic Swap[Nolan'13, Herlihy'18]

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

Ethereum blockchain



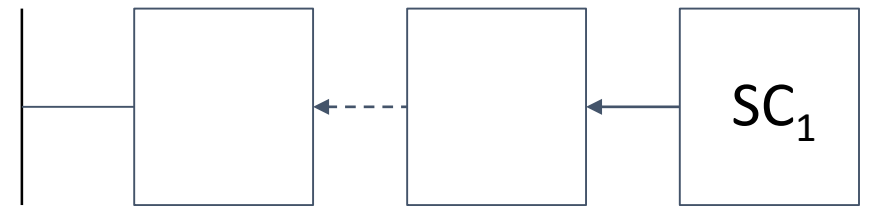
SC_2 Move Y Ethereum to Alice if Alice provides secret $s \mid h = H(s)$



Bob



Bitcoin blockchain



Alice's X bitcoins are locked in the smart contract SC_1



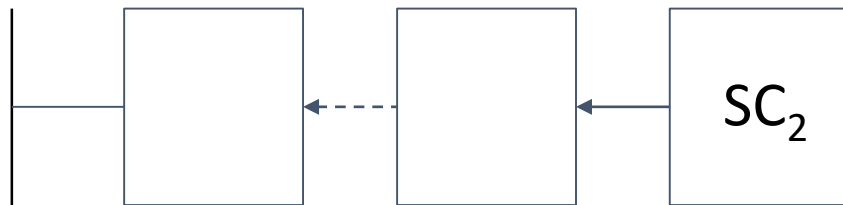
Alice



Atomic Swap[Nolan'13, Herlihy'18]

- Now, for Alice to execute SC_2 and redeem Y Ethereum, she reveals s

Ethereum blockchain



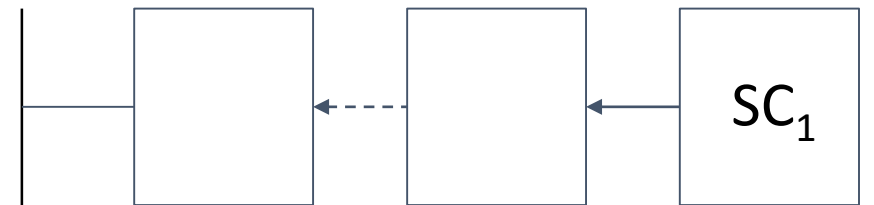
Bob's Y Ethereum are locked in smart contract SC_2



Bob



Bitcoin blockchain



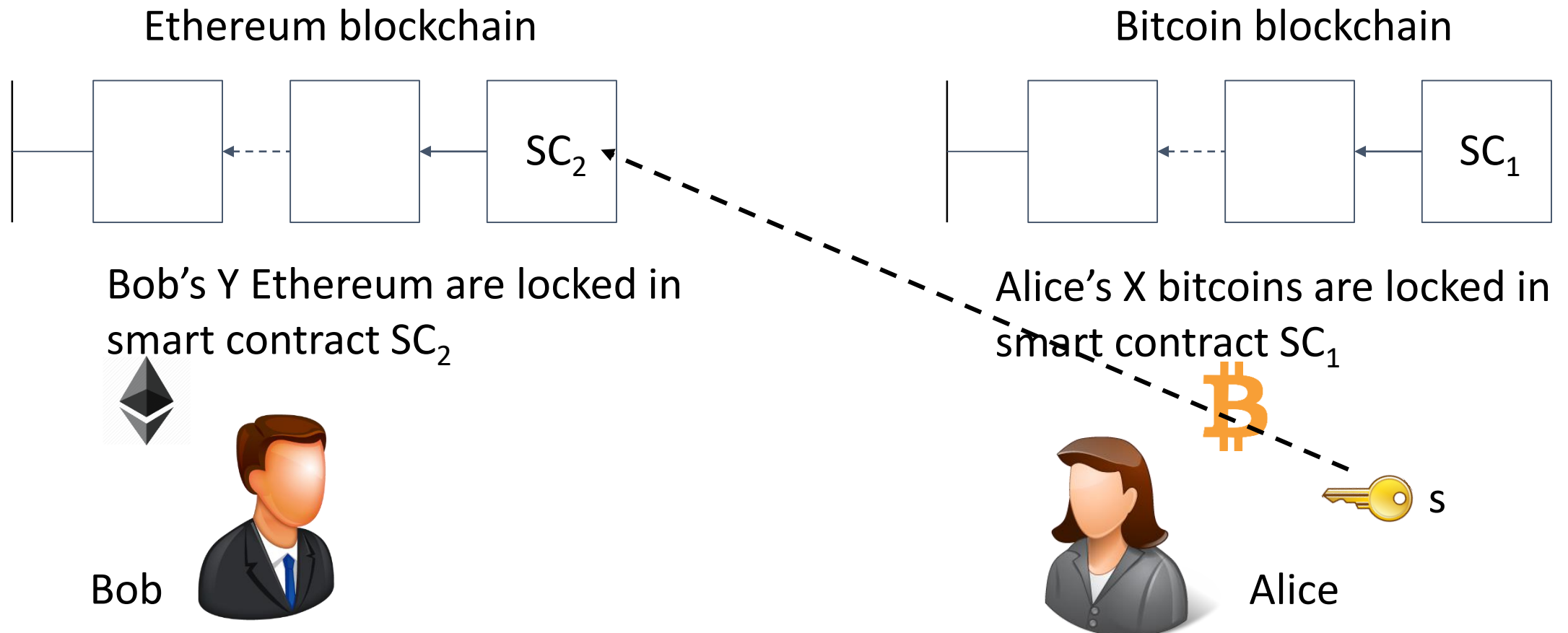
Alice's X bitcoins are locked in smart contract SC_1



Alice

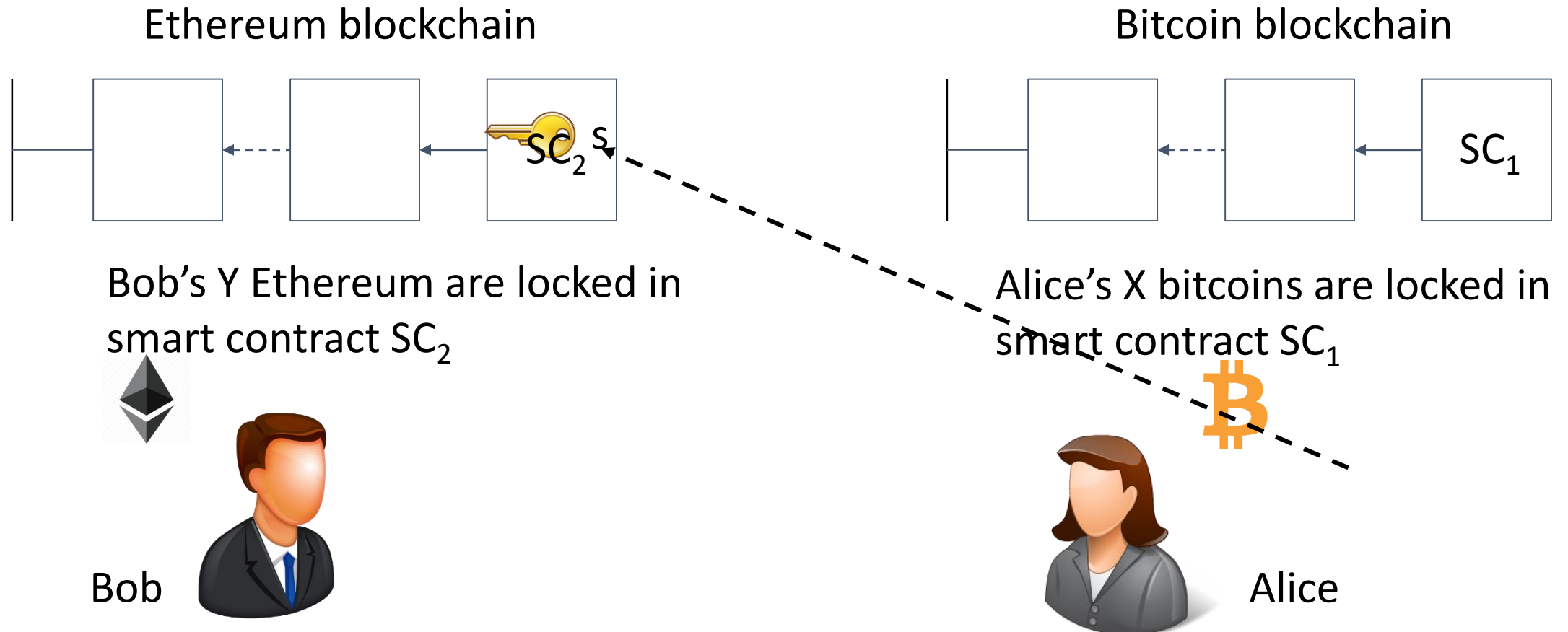
Atomic Swap[Nolan'13, Herlihy'18]

- Now, for Alice to execute SC_2 and redeem Y Ethereum, she reveals s



Atomic Swap[Nolan'13, Herlihy'18]

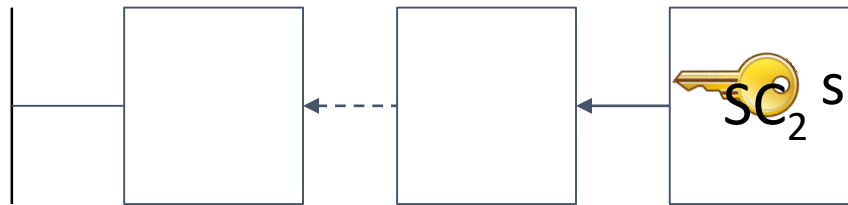
- Now, for Alice to execute SC_2 and redeem Y Ethereum, she reveals s



Atomic Swap[Nolan'13, Herlihy'18]

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

Ethereum blockchain



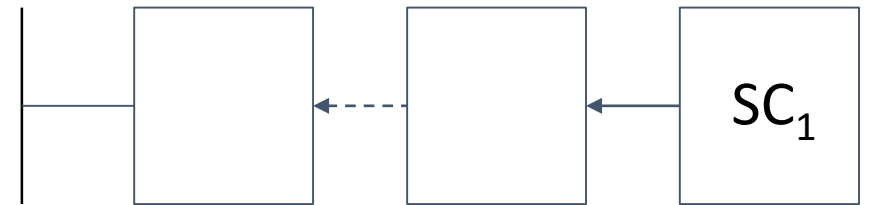
Bob's Y Ethereum are locked in smart contract SC_2



Bob



Bitcoin blockchain



Alice's X bitcoins are locked in smart contract SC_1



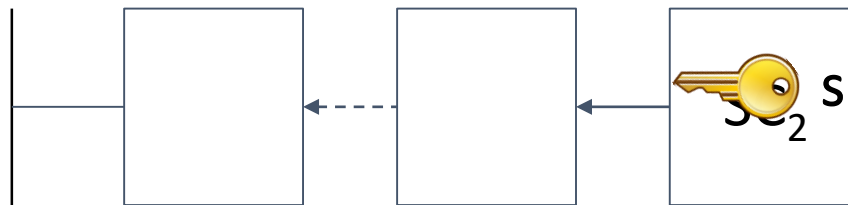
Alice



Atomic Swap[Nolan'13, Herlihy'18]

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

Ethereum blockchain



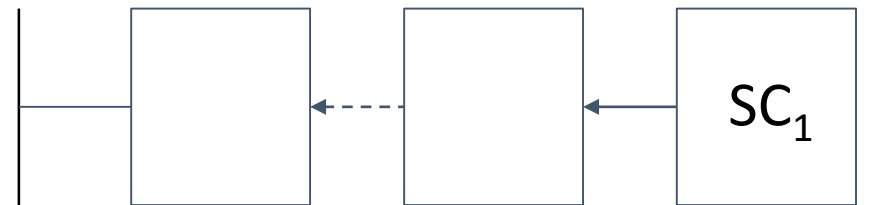
Bob's Y Ethereum are locked in smart contract SC_2



Bob



Bitcoin blockchain



Alice's X bitcoins are locked in smart contract SC_1



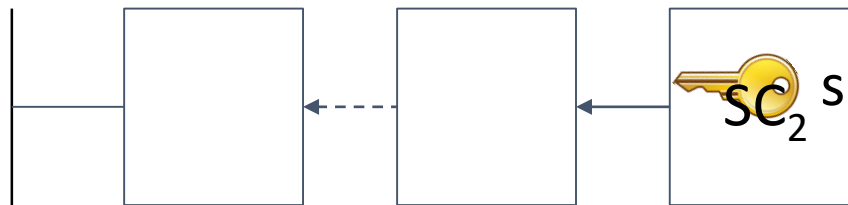
Alice



Atomic Swap[Nolan'13, Herlihy'18]

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

Ethereum blockchain



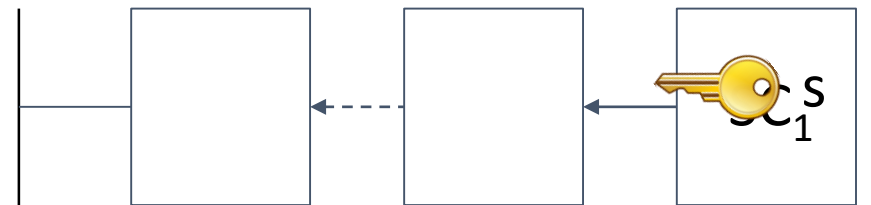
Bob's Y Ethereum are locked in smart contract SC_2



Bob



Bitcoin blockchain



Alice's X bitcoins are locked in smart contract SC_1



Alice



Atomic Swap Example: What can go wrong?

- Alice locks her X Bitcoins in Bitcoin's blockchain through SC_1

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

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

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



Bob



Alice

Atomic Swap[Nolan'13, Herlihy'18]: Timelocks



Bob



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 | $h = H(s)$



Alice

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 $s \mid h = H(s)$



Bob



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 $s \mid h = H(s)$



Alice

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 $s \mid h = H(s)$



Bob



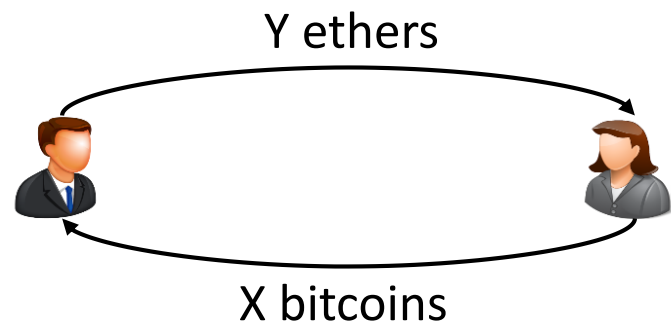
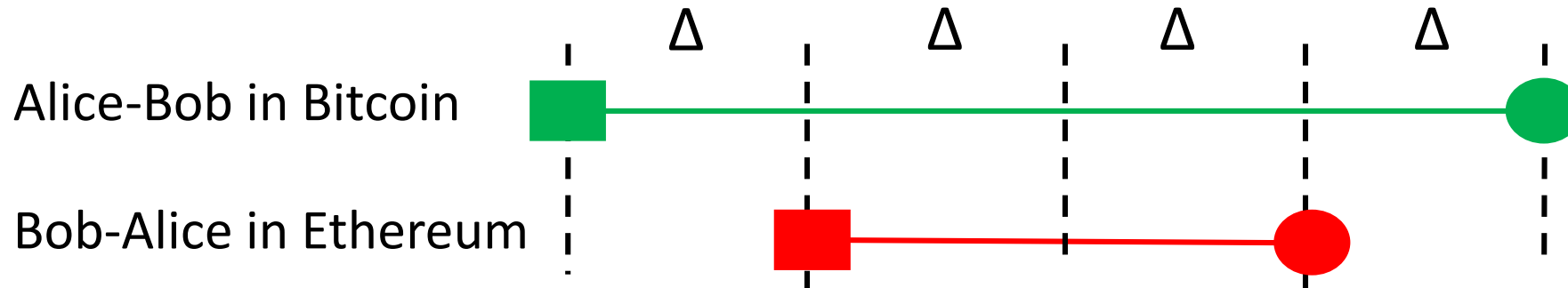
Refund SC_1 to Alice if Bob does not execute SC_1 before **48** hours

SC_1 : Move X bitcoins to Bob if Bob provides secret $s \mid h = H(s)$



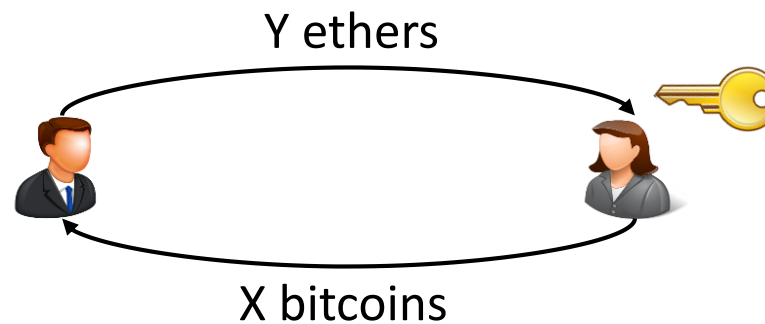
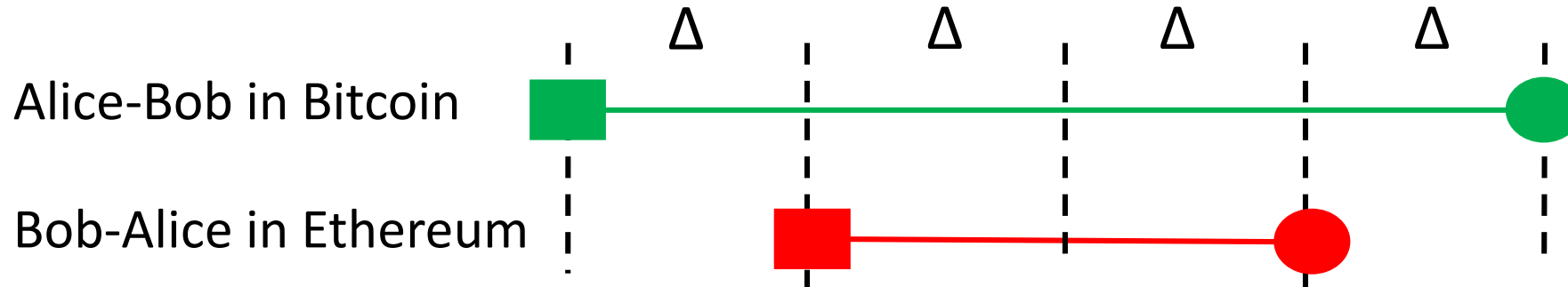
Alice

Atomic Swap Example [Nolan'13, Herlihy'18]



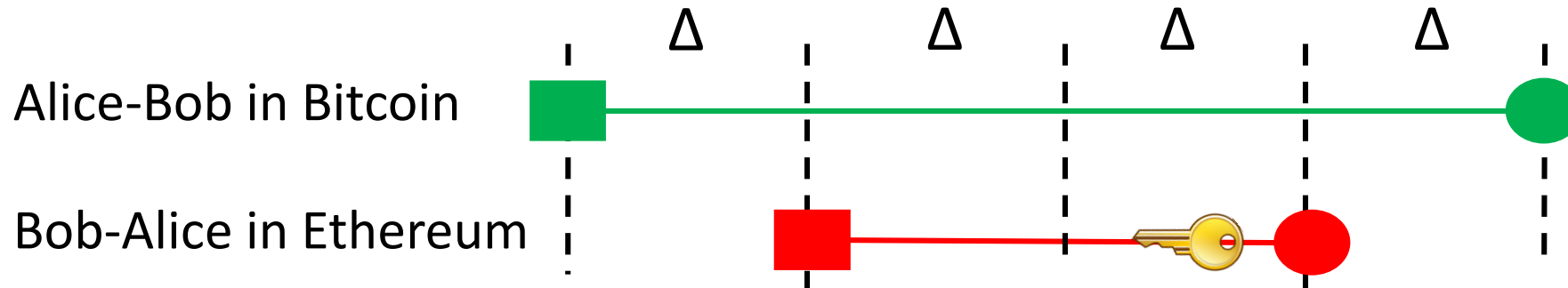
e.g., $\Delta = 12\text{hr}$

Atomic Swap Example [Nolan'13, Herlihy'18]

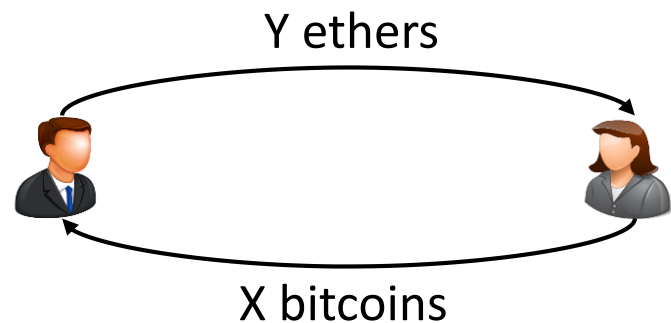


e.g., $\Delta = 12\text{hr}$

Atomic Swap Example [Nolan'13, Herlihy'18]

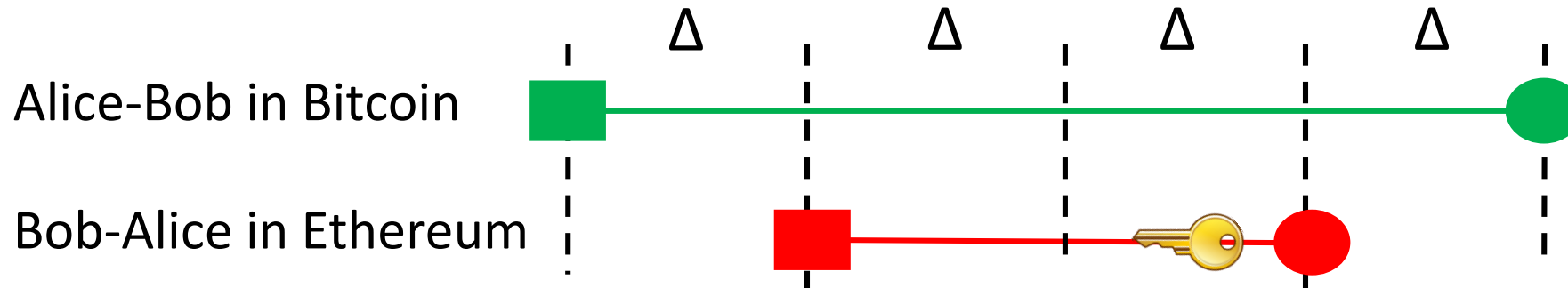


Alice reveals the secret to Bob's contract and claims the Y ether



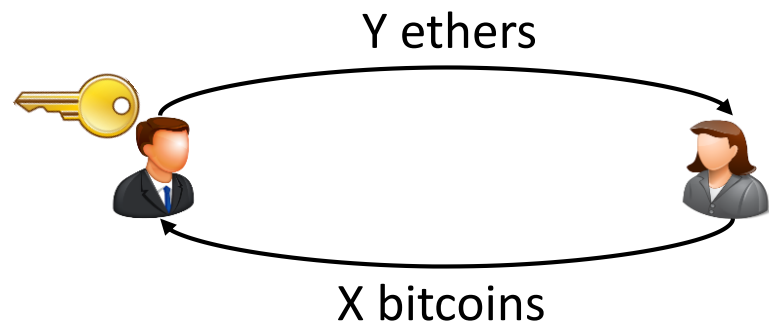
e.g., $\Delta = 12\text{hr}$

Atomic Swap Example [Nolan'13, Herlihy'18]



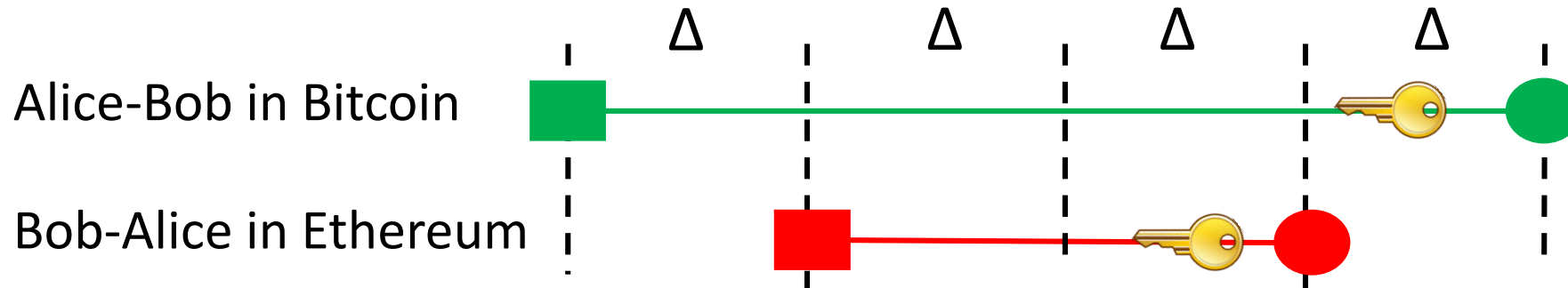
Alice reveals the secret to Bob's contract and claims the Y ether

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



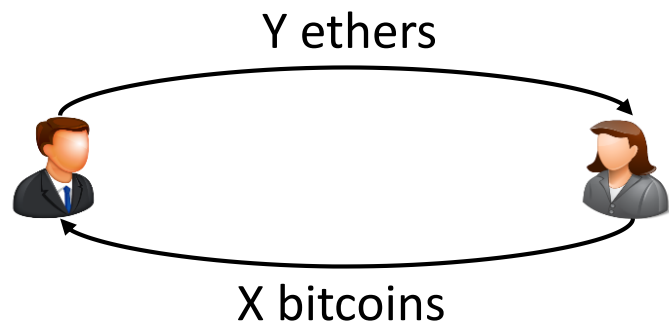
e.g., $\Delta = 12\text{hr}$

Atomic Swap Example [Nolan'13, Herlihy'18]



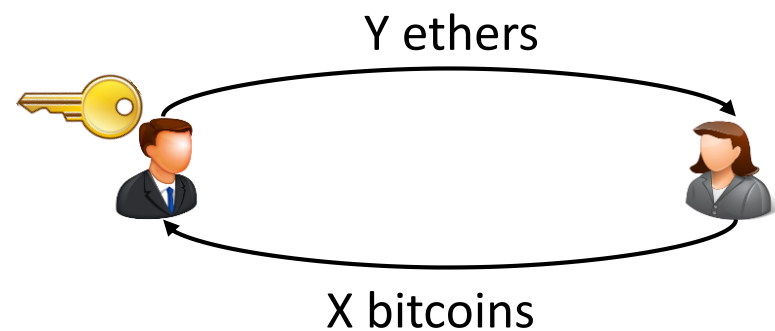
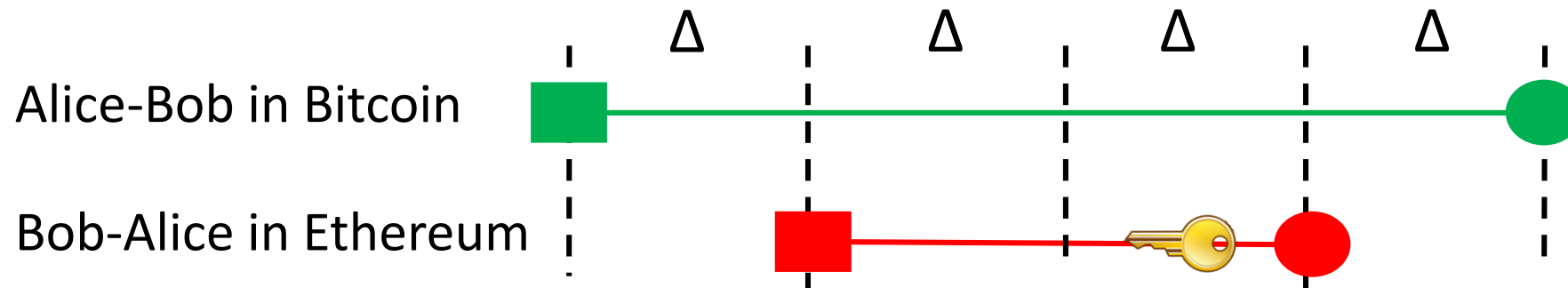
Alice reveals the secret to Bob's contract and claims the Y ether

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



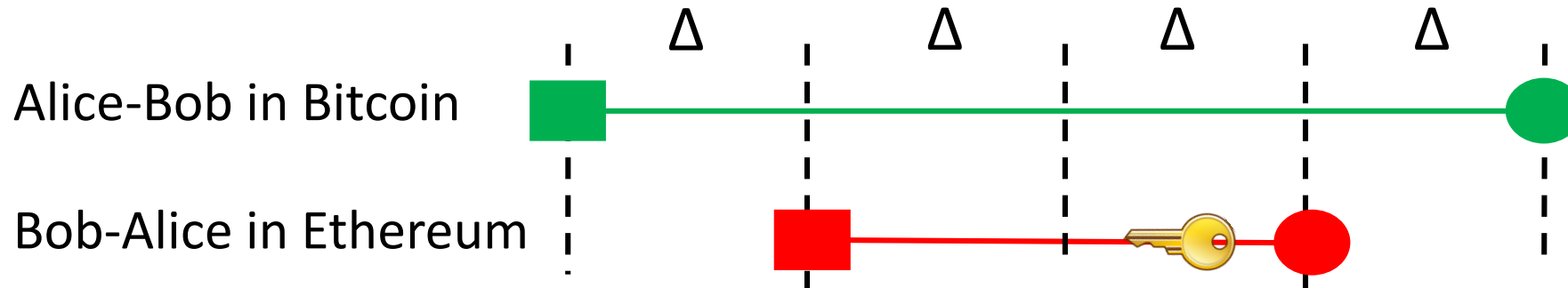
e.g., $\Delta = 12\text{hr}$

What can go wrong?

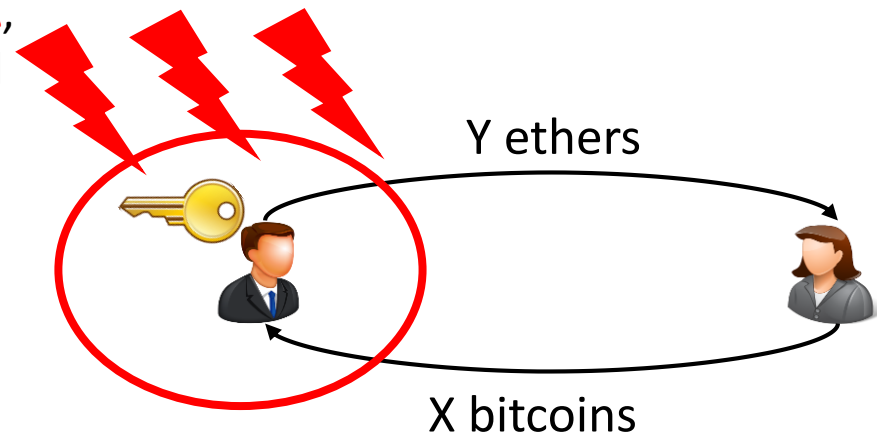


e.g., $\Delta = 12\text{hr}$

What can go wrong?

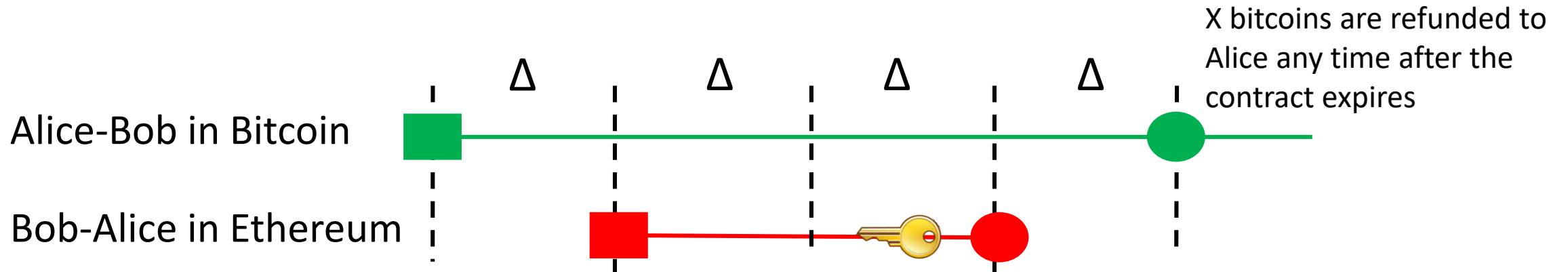


If Bob fails or suffers a network denial of service attack for a Δ , Alice's contract will expire and Bob will lose his X bitcoins

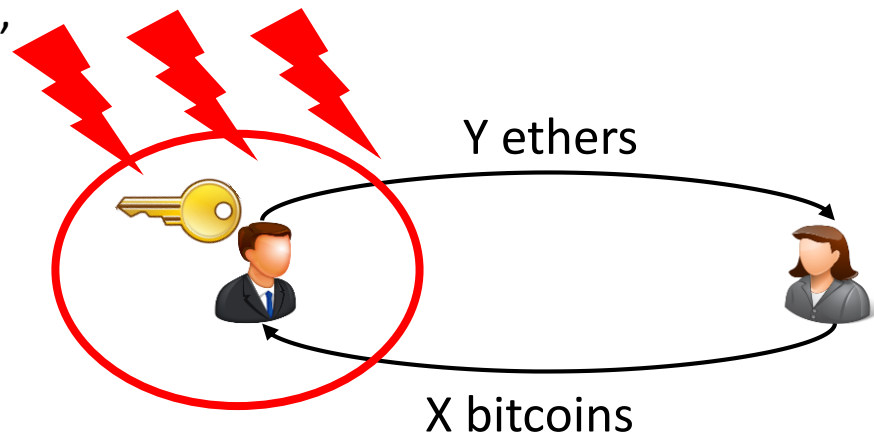


e.g., $\Delta = 12\text{hr}$

What can go wrong?

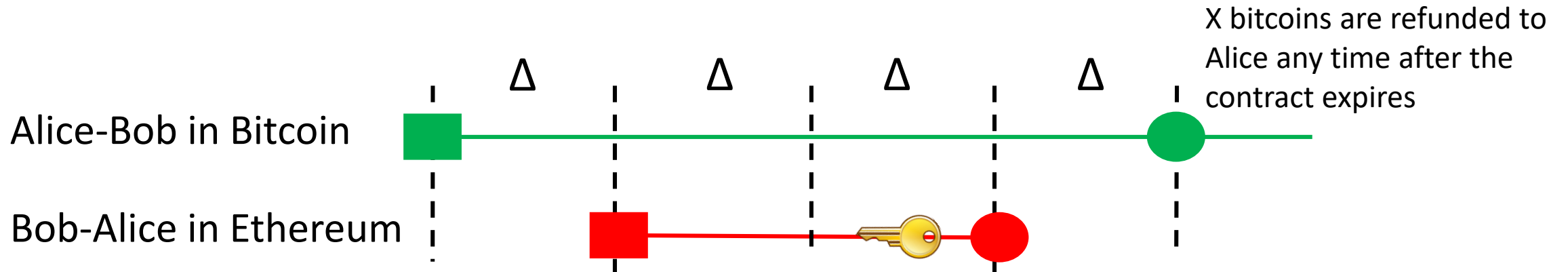


If Bob fails or suffers a network denial of service attack for a Δ , Alice's contract will expire and Bob will lose his X bitcoins



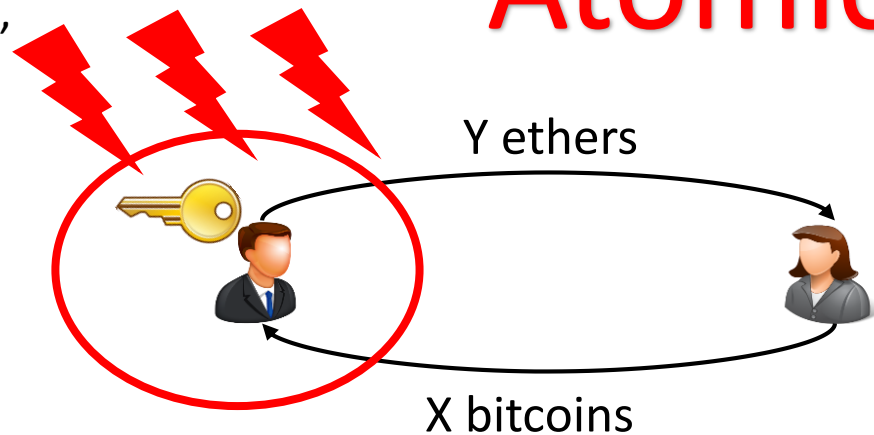
e.g., $\Delta = 12\text{hr}$

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

Atomicity Violation



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

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

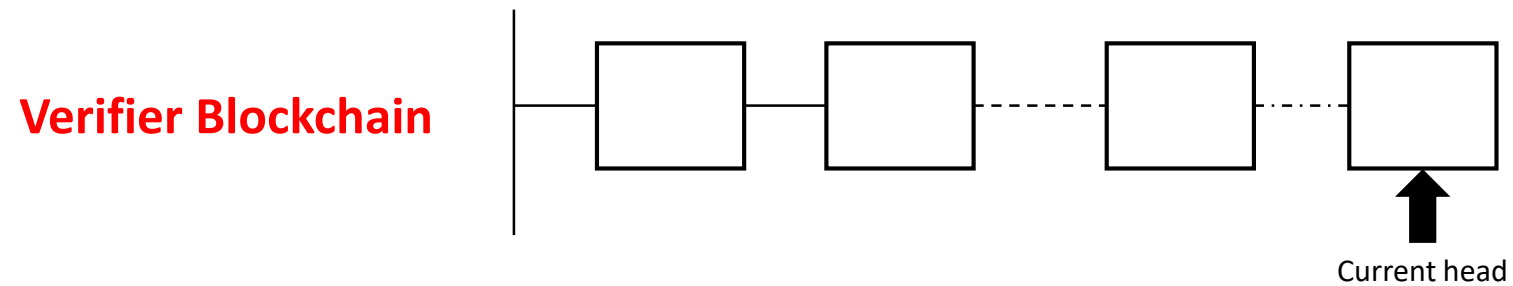
- How can miners of one blockchain:
 - Verify a transaction in another blockchain?

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.

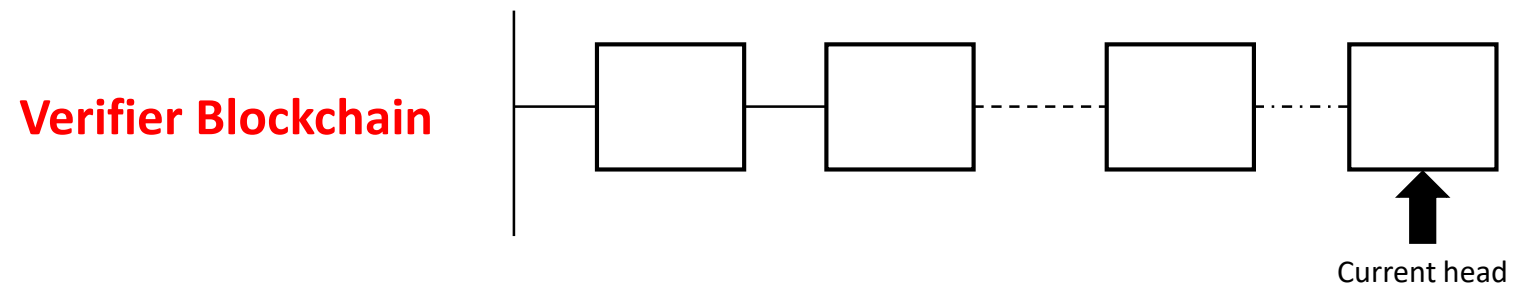
Building block: Cross-Chain Verification

Building block: Cross-Chain Verification



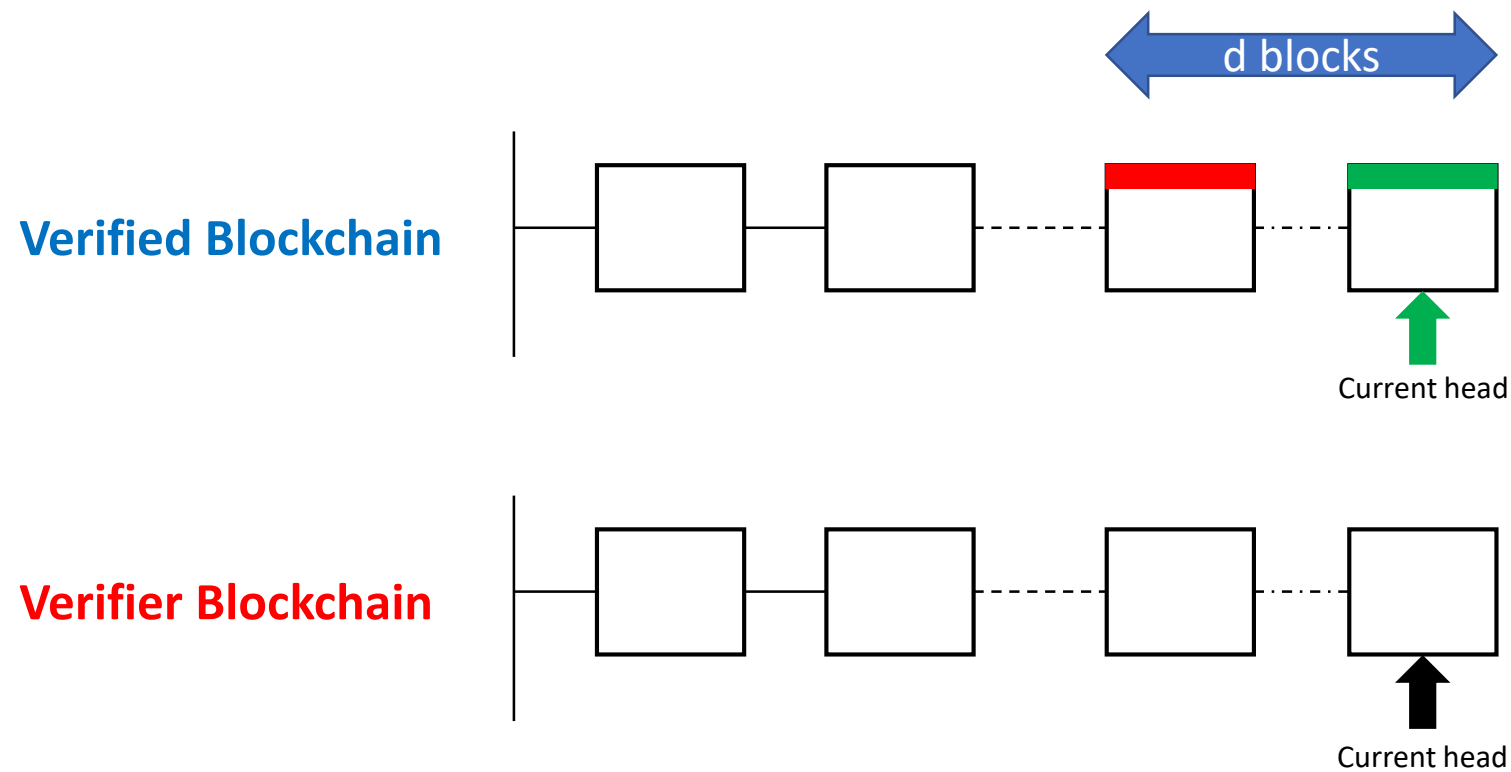
Building block: Cross-Chain Verification

Need to **verify** that TX_1 is actually
in **verified** blockchain



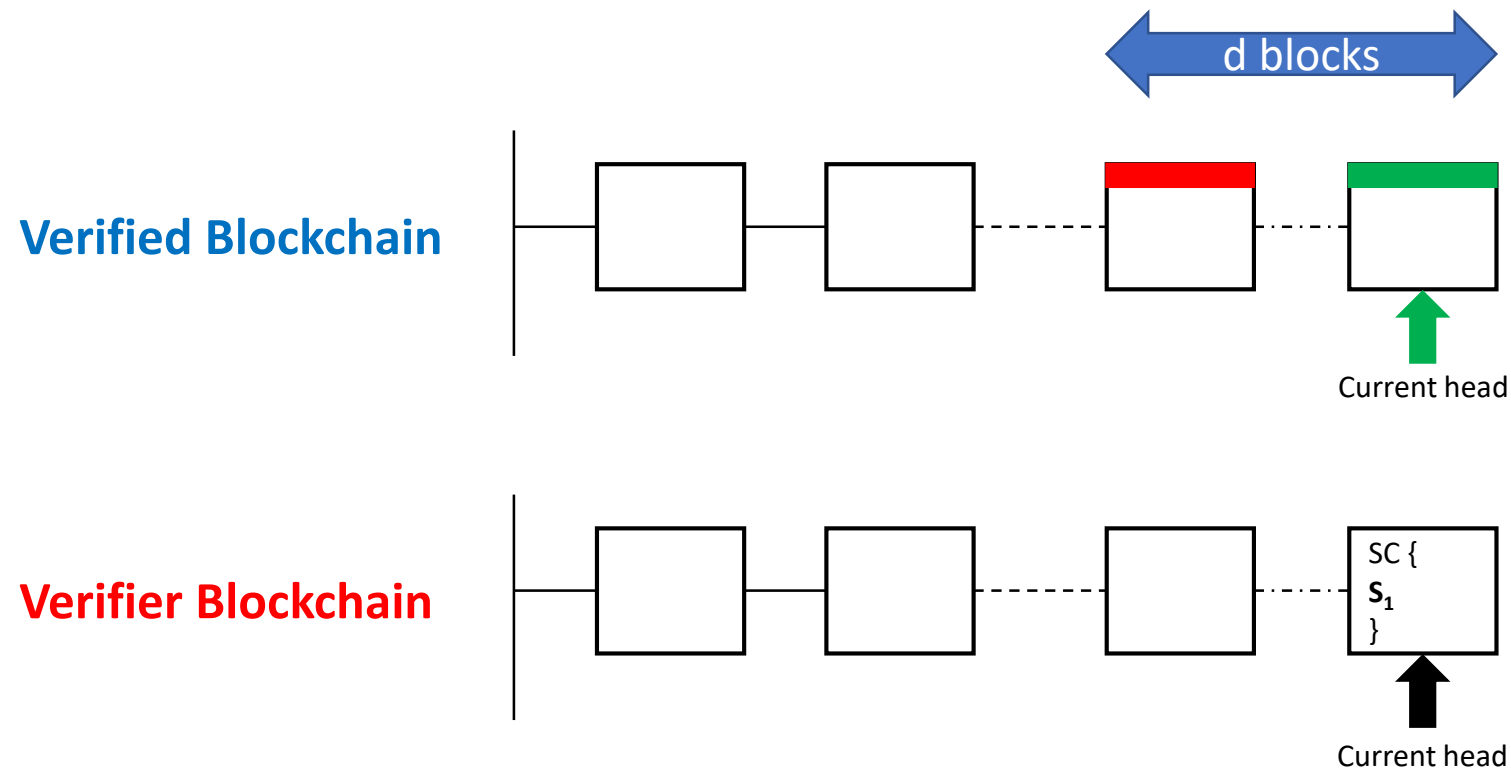
Building block: Cross-Chain Verification

Need to **verify** that TX_1 is actually in **verified** blockchain



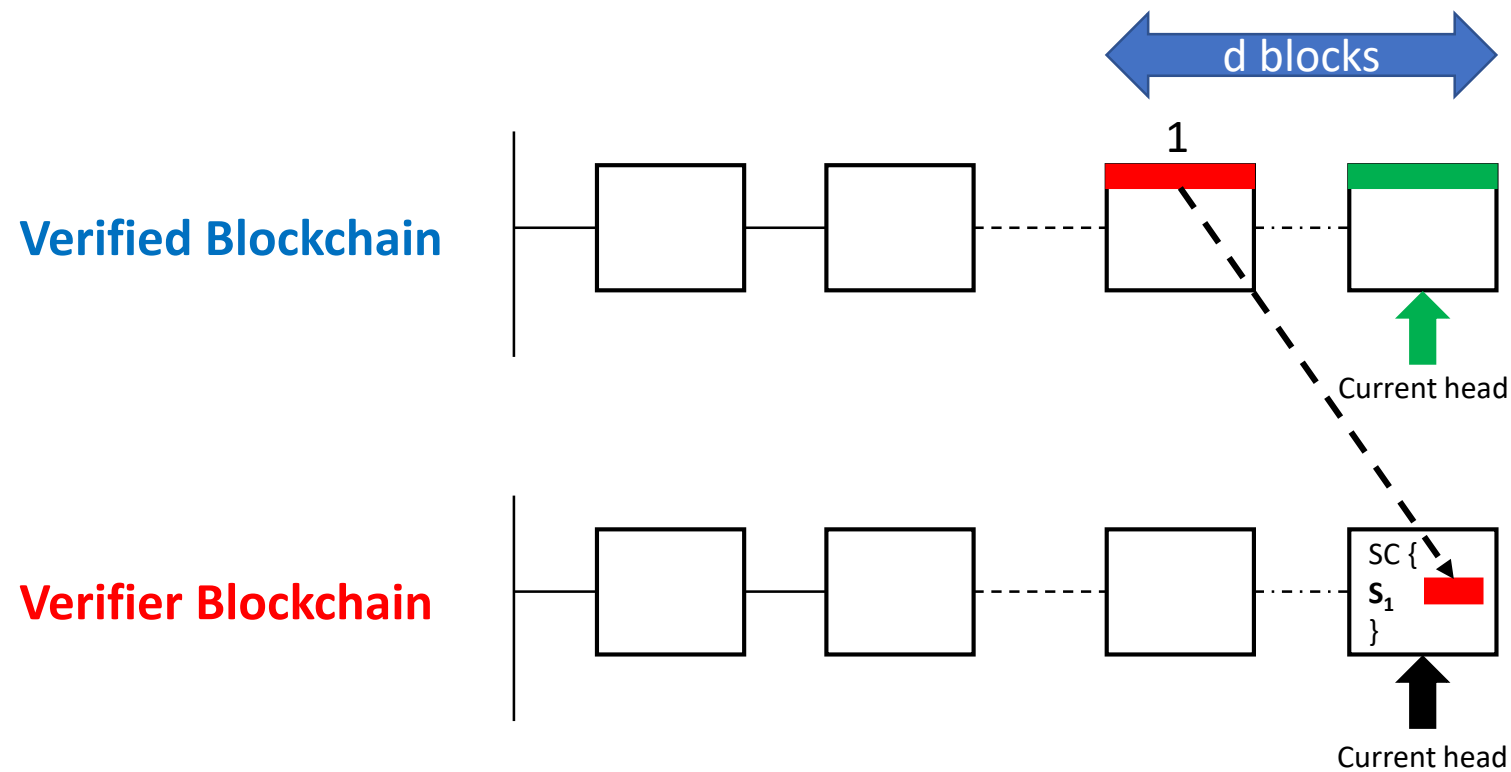
Building block: Cross-Chain Verification

Need to **verify** that TX_1 is actually
in **verified** blockchain



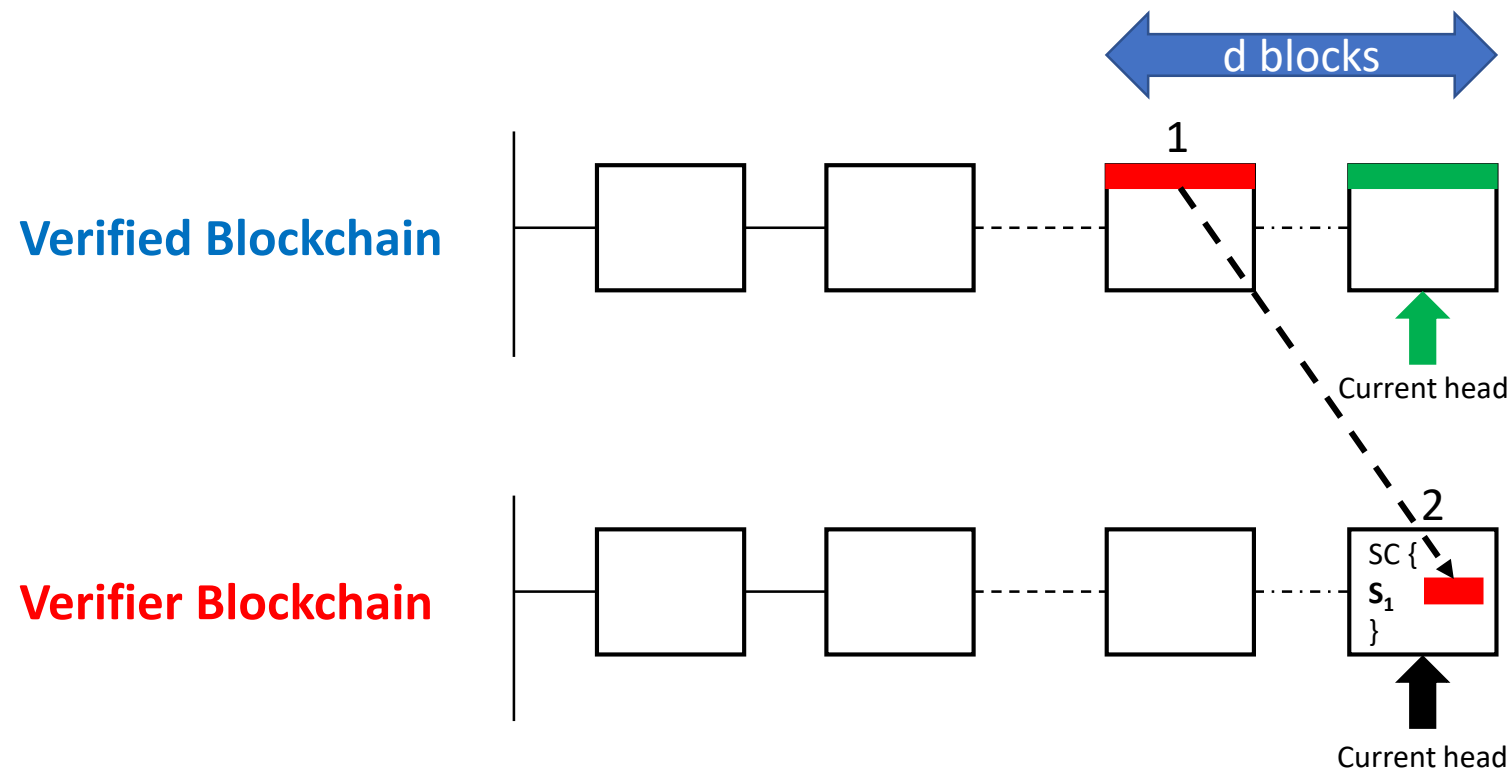
Building block: Cross-Chain Verification

Need to **verify** that TX_1 is actually in **verified** blockchain



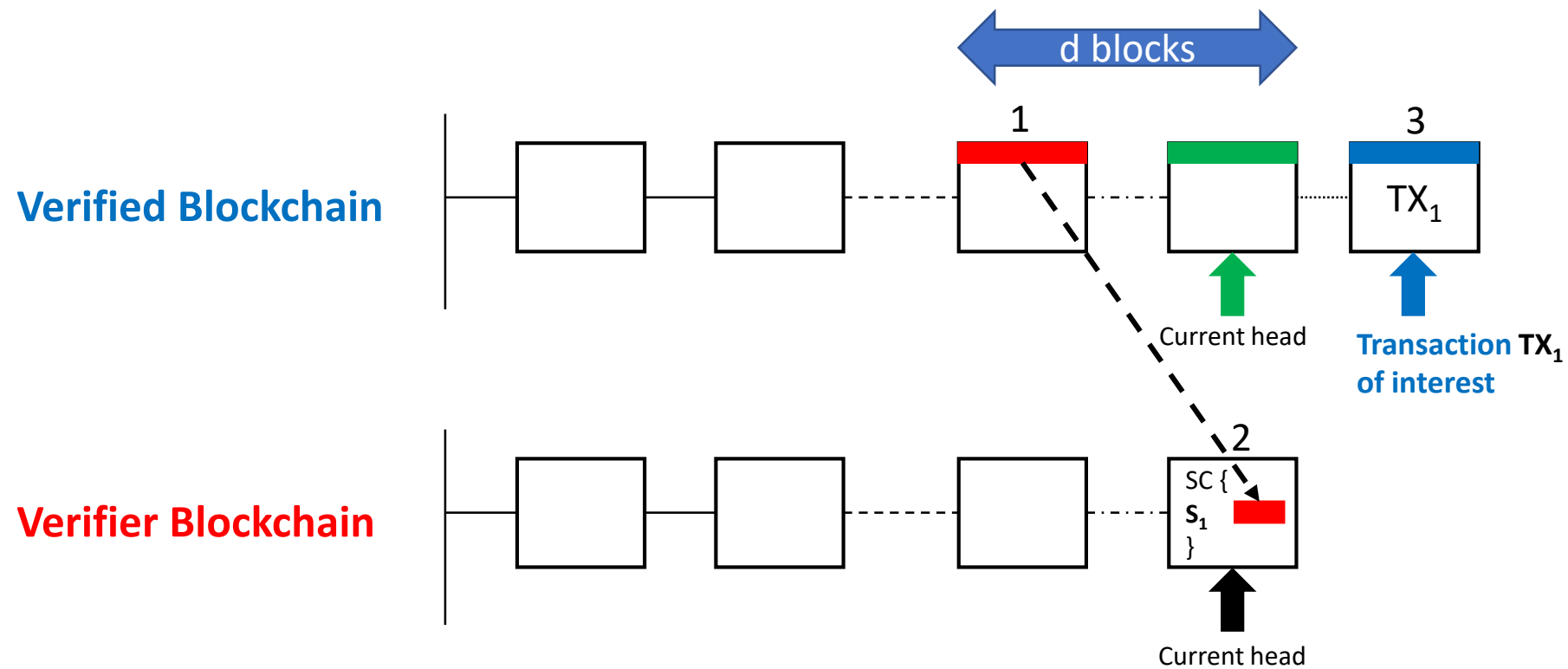
Building block: Cross-Chain Verification

Need to **verify** that TX_1 is actually in **verified** blockchain



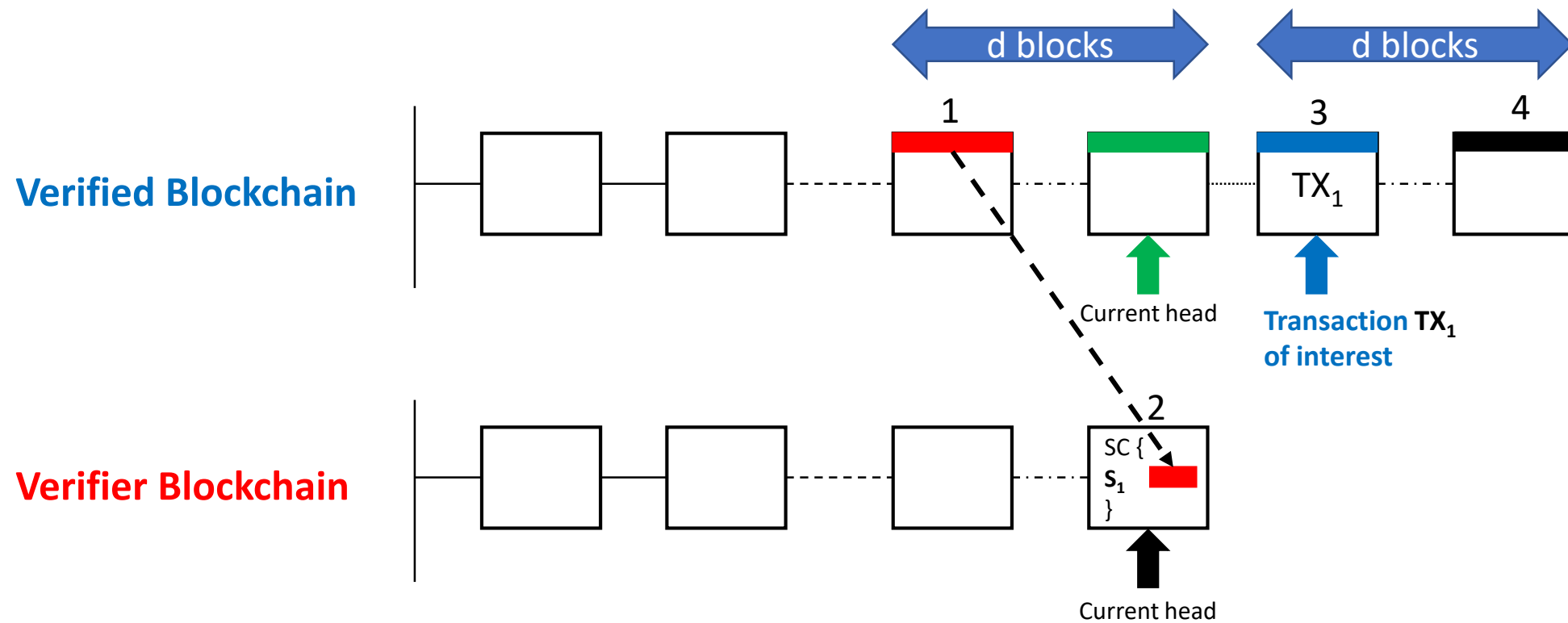
Building block: Cross-Chain Verification

Need to **verify** that TX_1 is actually in **verified** blockchain



Building block: Cross-Chain Verification

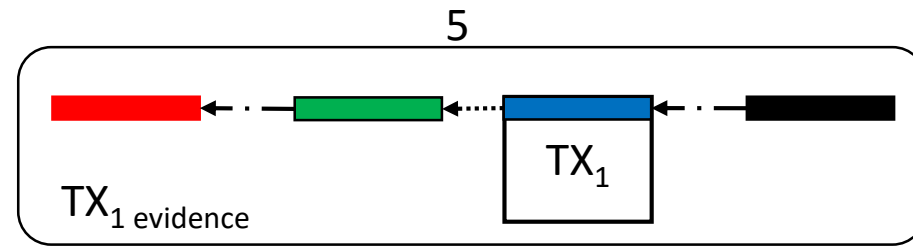
Need to **verify** that TX_1 is actually in **verified** blockchain



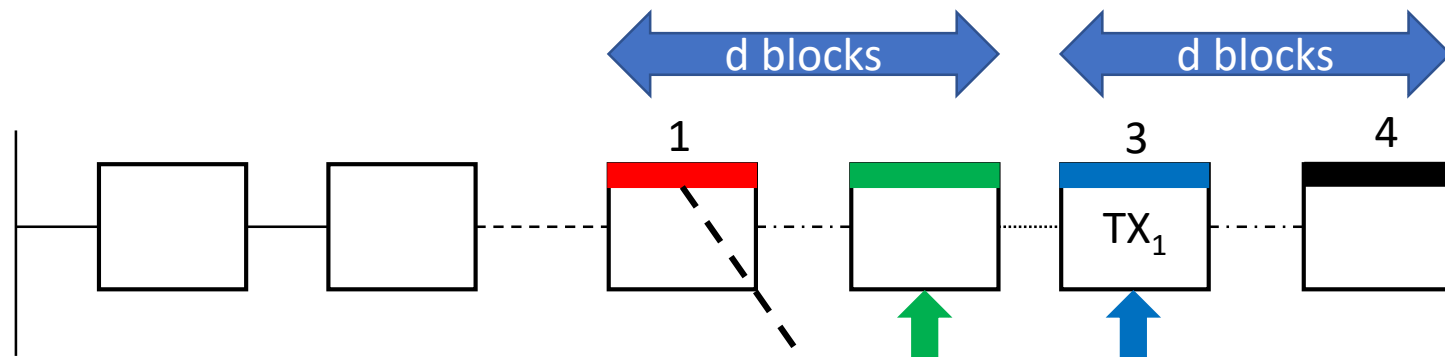
Building block: Cross-Chain Verification

Need to **verify** that TX_1 is actually in **verified** blockchain

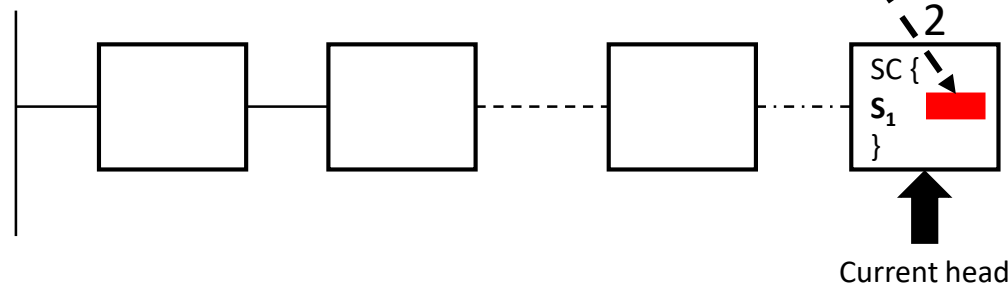
TX_1 Evidence



Verified Blockchain

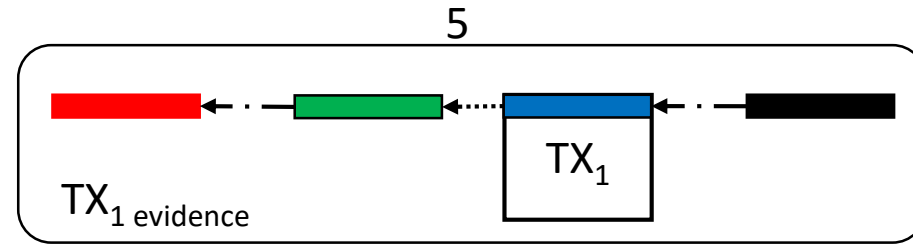


Verifier Blockchain



Building block: Cross-Chain Verification

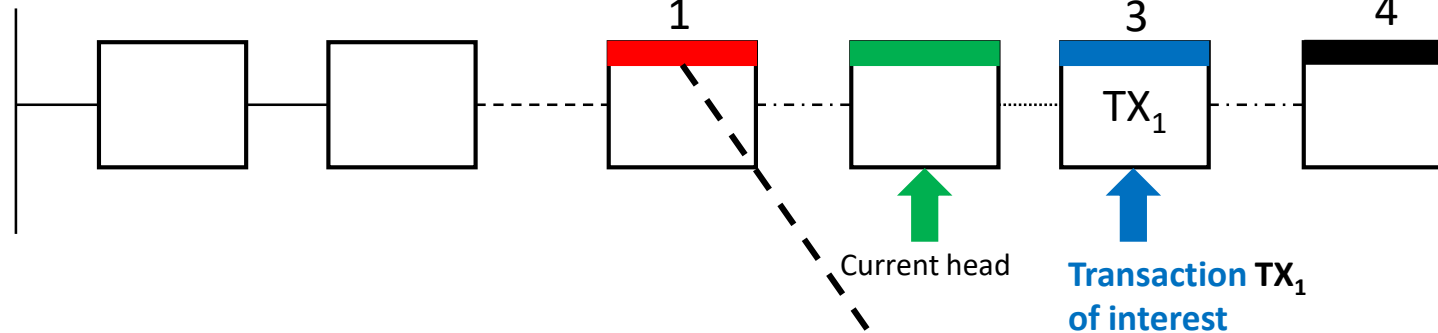
Need to **verify** that TX_1 is actually in **verified** blockchain



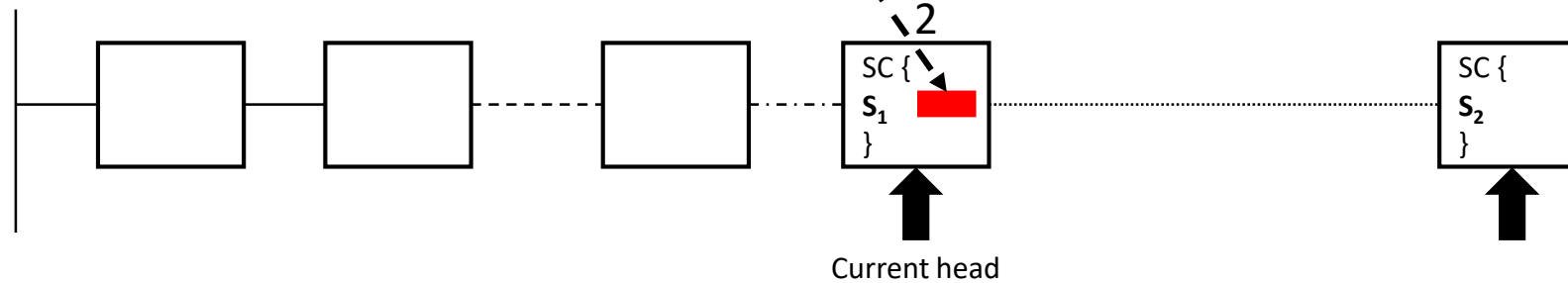
TX₁ Evidence



Verified Blockchain



Verifier Blockchain



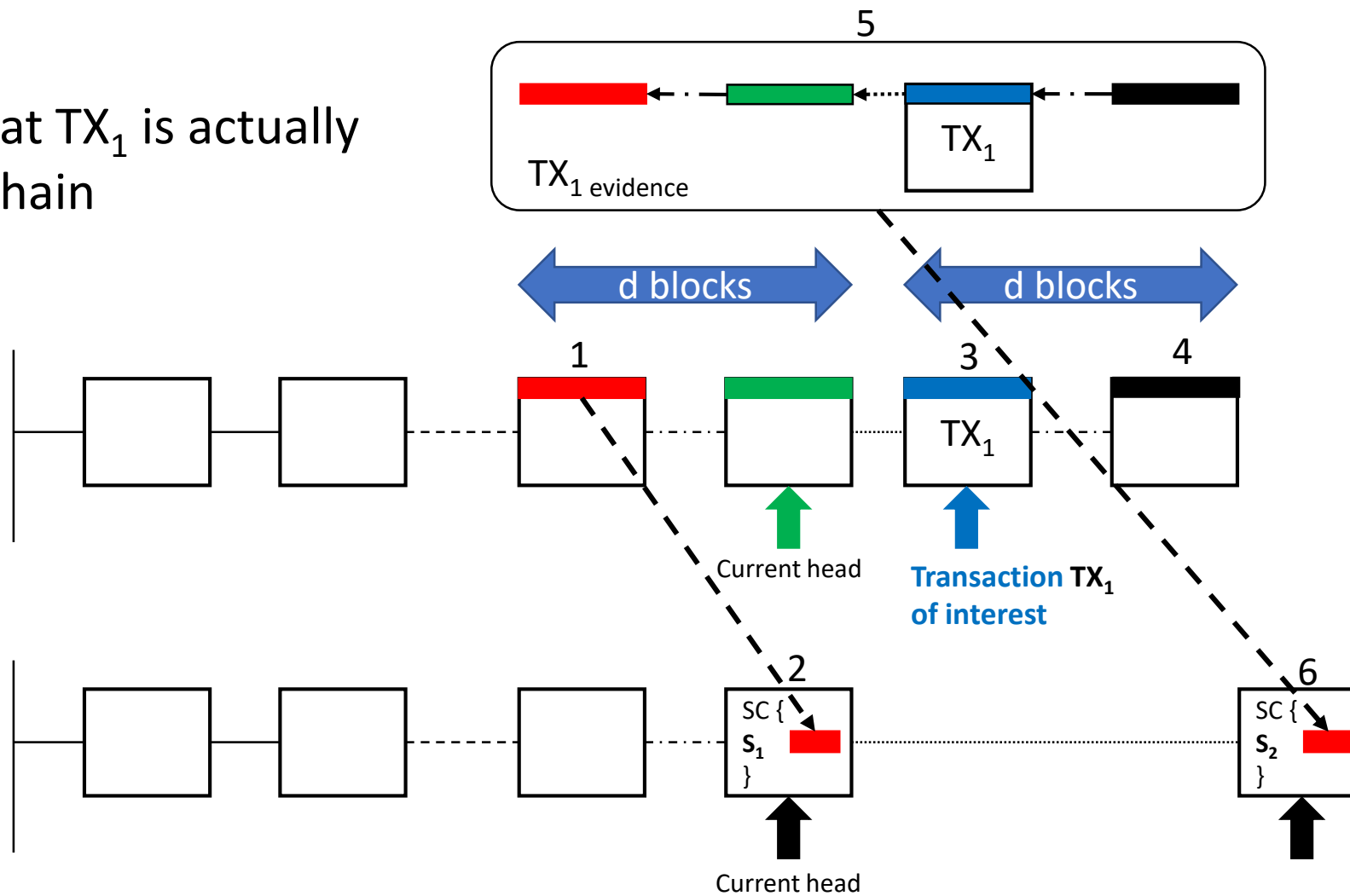
Building block: Cross-Chain Verification

Need to **verify** that TX_1 is actually in **verified** blockchain

TX_1 Evidence

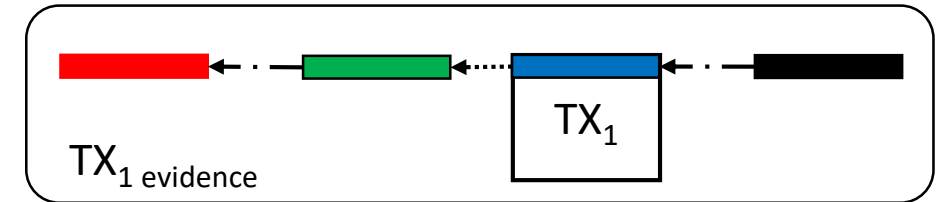
Verified Blockchain

Verifier Blockchain



Building block: Cross-Chain Verification

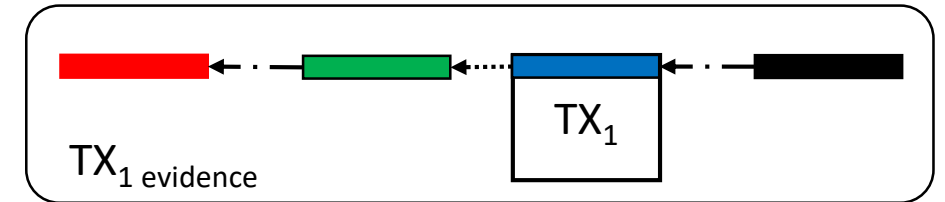
- Verification process:



Building block: Cross-Chain Verification

- Verification process:

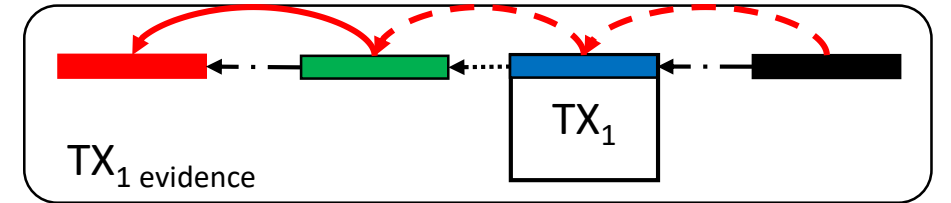
- Each header includes the hash of the previous header



Building block: Cross-Chain Verification

- Verification process:

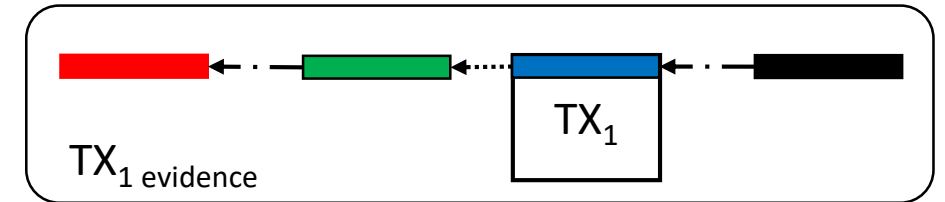
- Each header includes the hash of the previous header



Building block: Cross-Chain Verification

- Verification process:

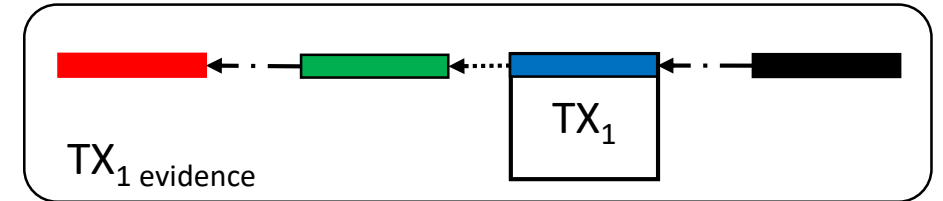
- Each header includes the hash of the previous header



Building block: Cross-Chain Verification

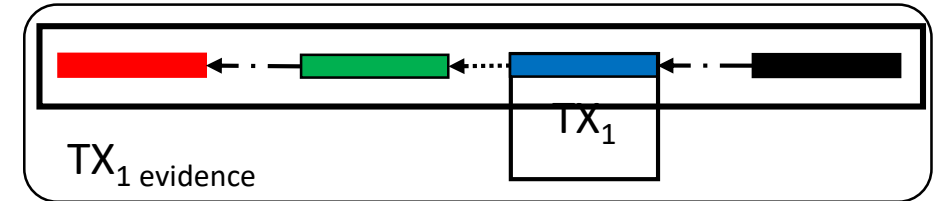
- Verification process:

- Each header includes the hash of the previous header
- The proof of work of each header is correct



Building block: Cross-Chain Verification

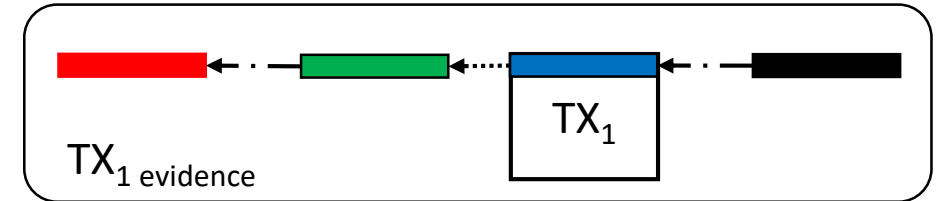
- Verification process:
 - Each header includes the hash of the previous header
 - The proof of work of each header is correct



Building block: Cross-Chain Verification

- Verification process:

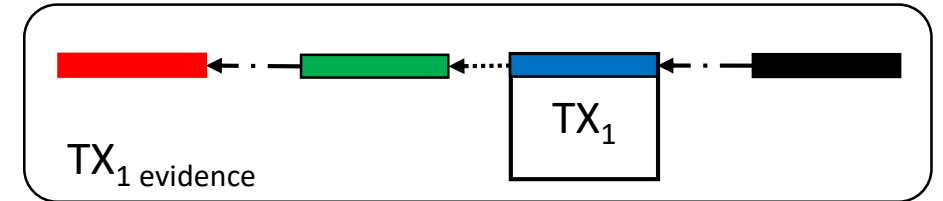
- Each header includes the hash of the previous header
- The proof of work of each header is correct



Building block: Cross-Chain Verification

- Verification process:

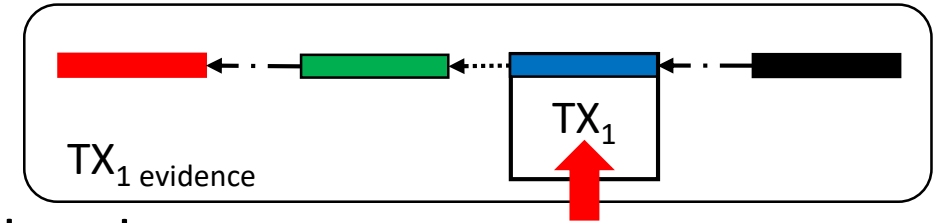
- Each header includes the hash of the previous header
- The proof of work of each header is correct
- TX_1 is correct



Building block: Cross-Chain Verification

- Verification process:

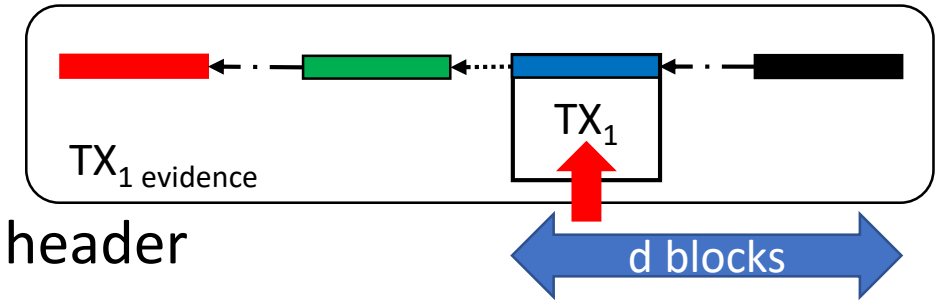
- Each header includes the hash of the previous header
- The proof of work of each header is correct
- TX_1 is correct



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



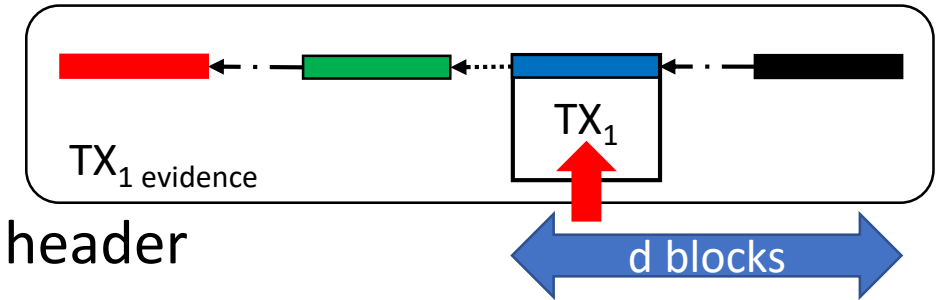
Building block: Cross-Chain Verification

- 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**

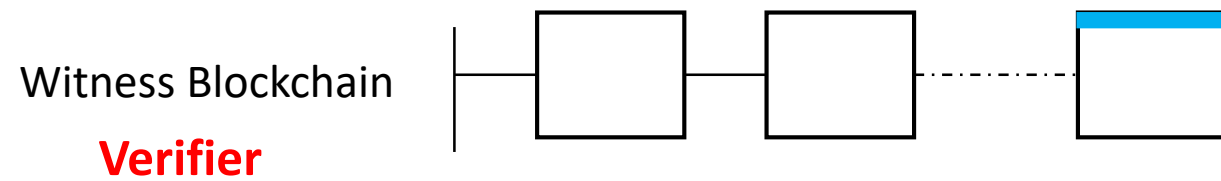
Protocol Sketch

Protocol Sketch

- Deploy a contract SC_w in the witness network with state *Published (P)*

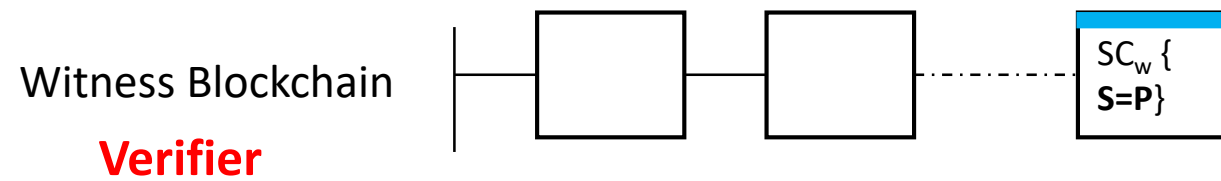
Protocol Sketch

- Deploy a contract SC_w in the witness network with state *Published (P)*



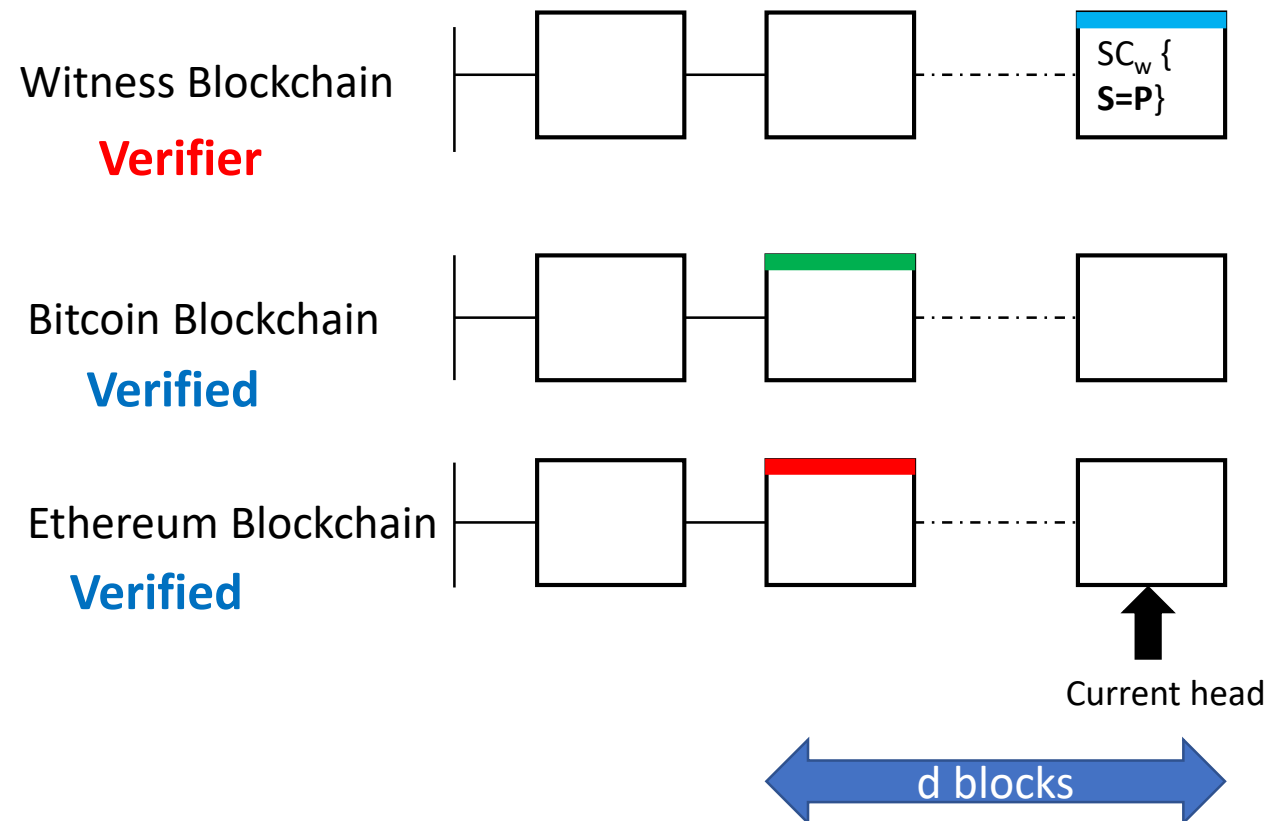
Protocol Sketch

- Deploy a contract SC_w in the witness network with state *Published* (P)



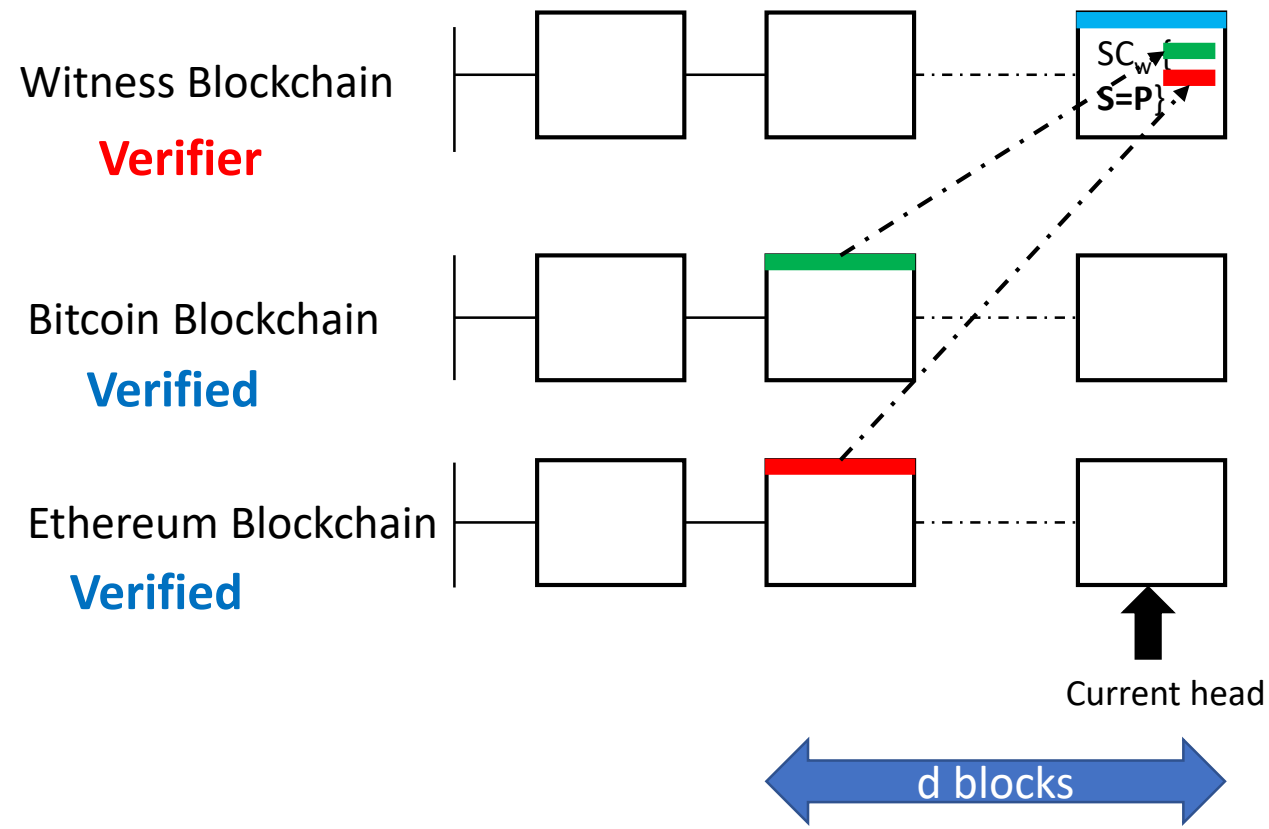
Protocol Sketch

- Deploy a contract SC_w in the witness network with state *Published* (P)

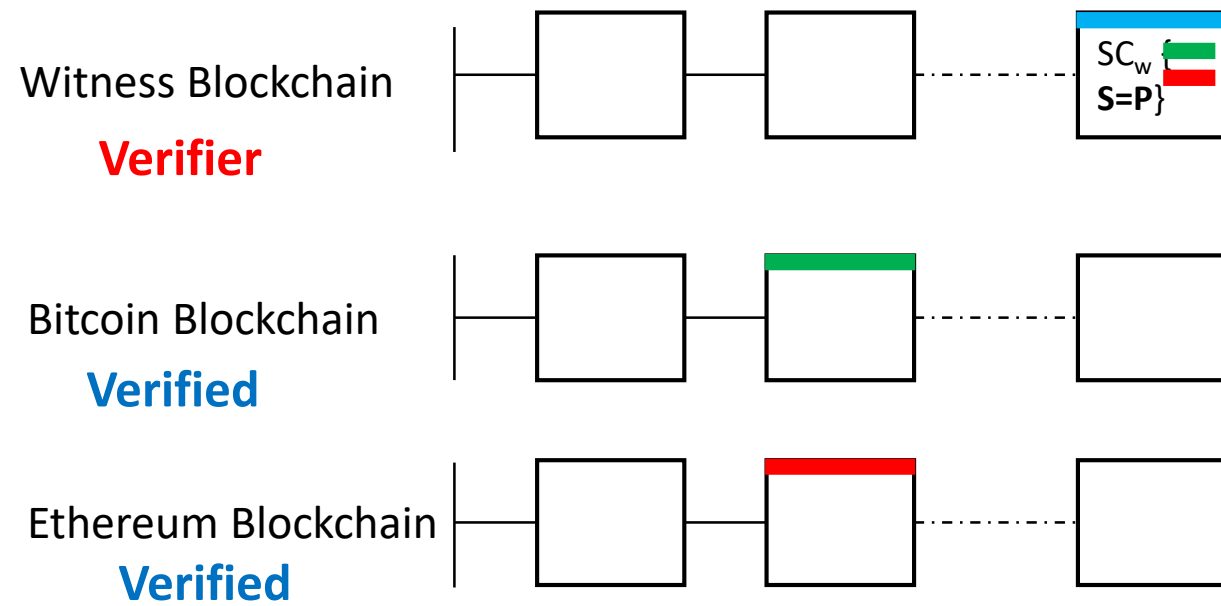


Protocol Sketch

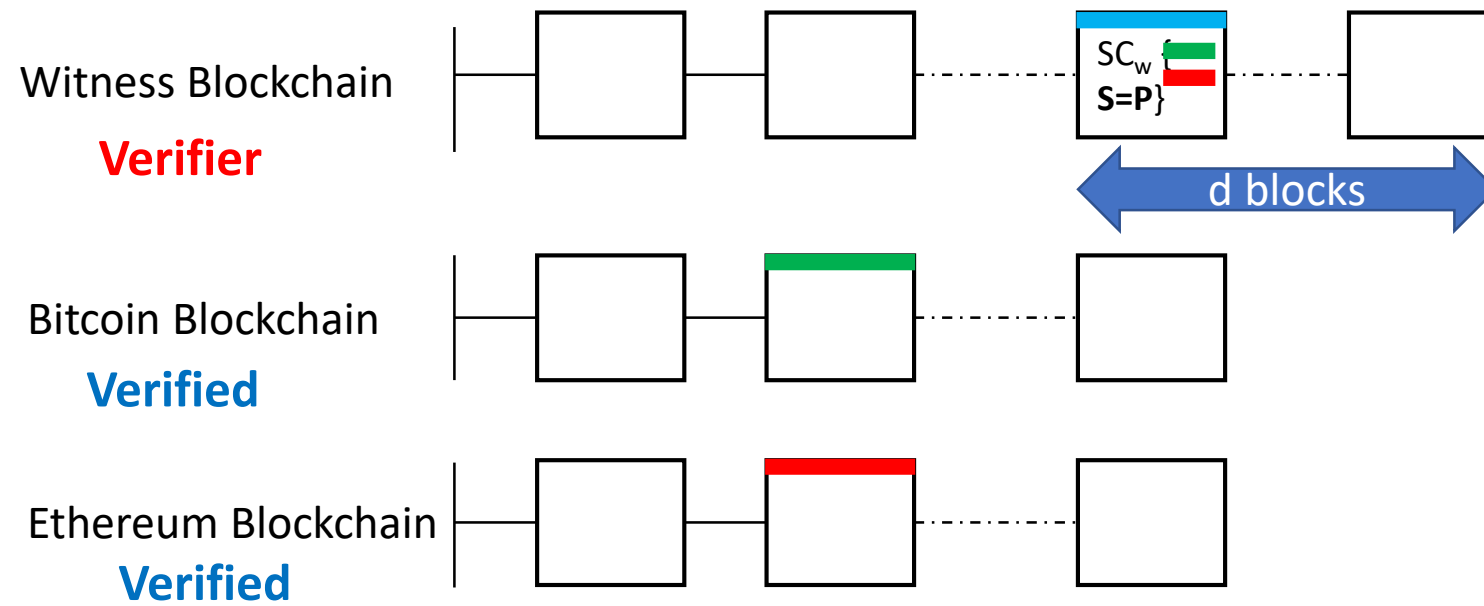
- Deploy a contract SC_w in the witness network with state *Published* (P)
- SC_w has a header of a block at depth d of all blockchains in the swap



Protocol Sketch Cont'd

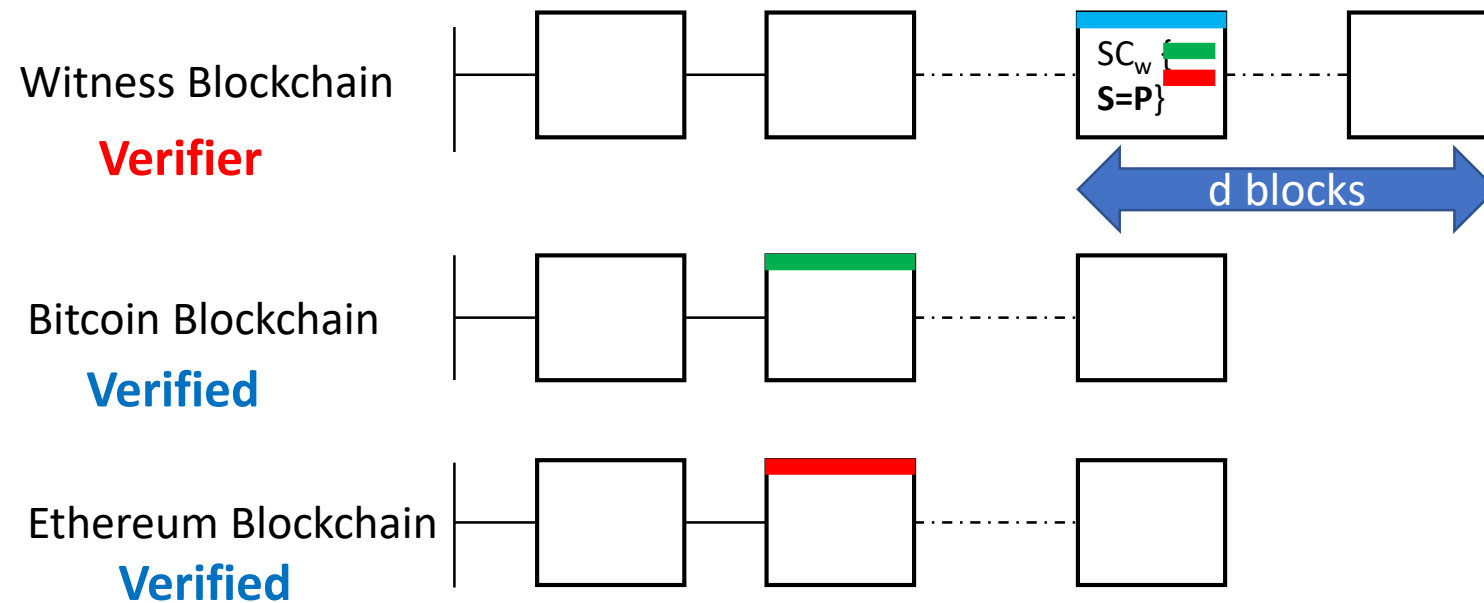


Protocol Sketch Cont'd



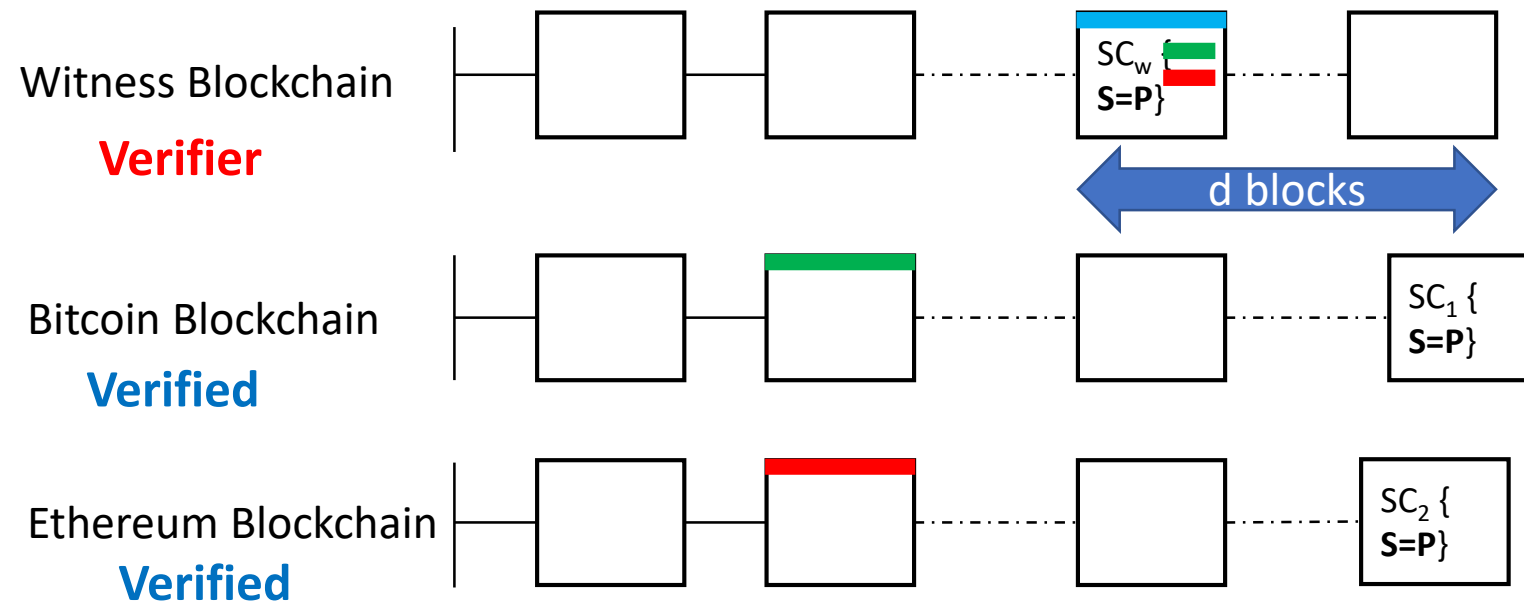
Protocol Sketch Cont'd

- Participants deploy their contracts in the corresponding blockchains



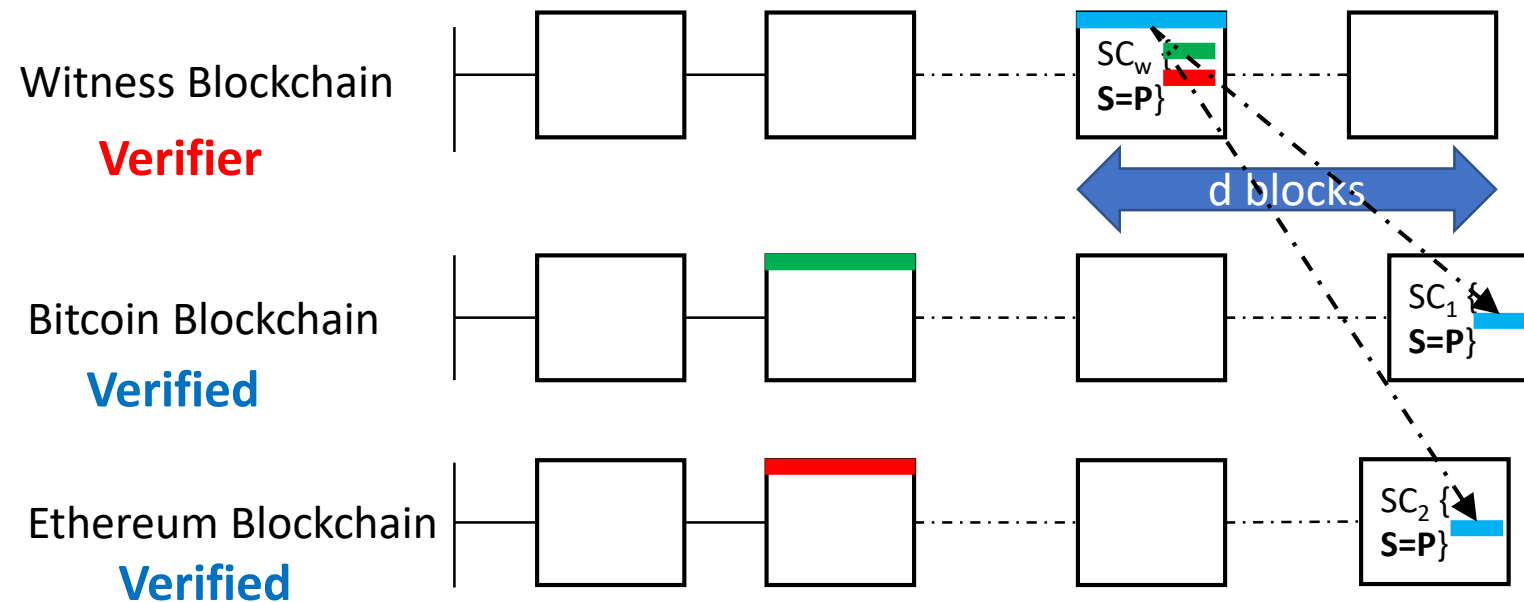
Protocol Sketch Cont'd

- Participants deploy their contracts in the corresponding blockchains

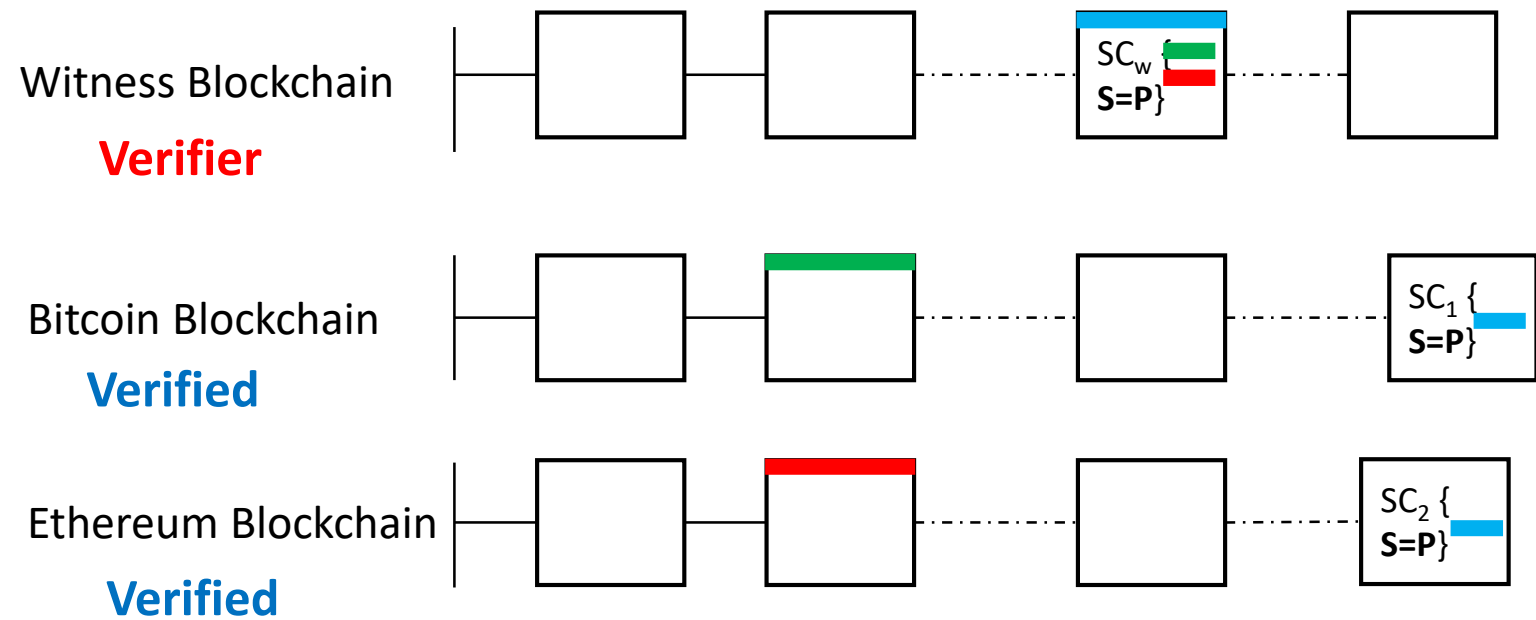


Protocol Sketch Cont'd

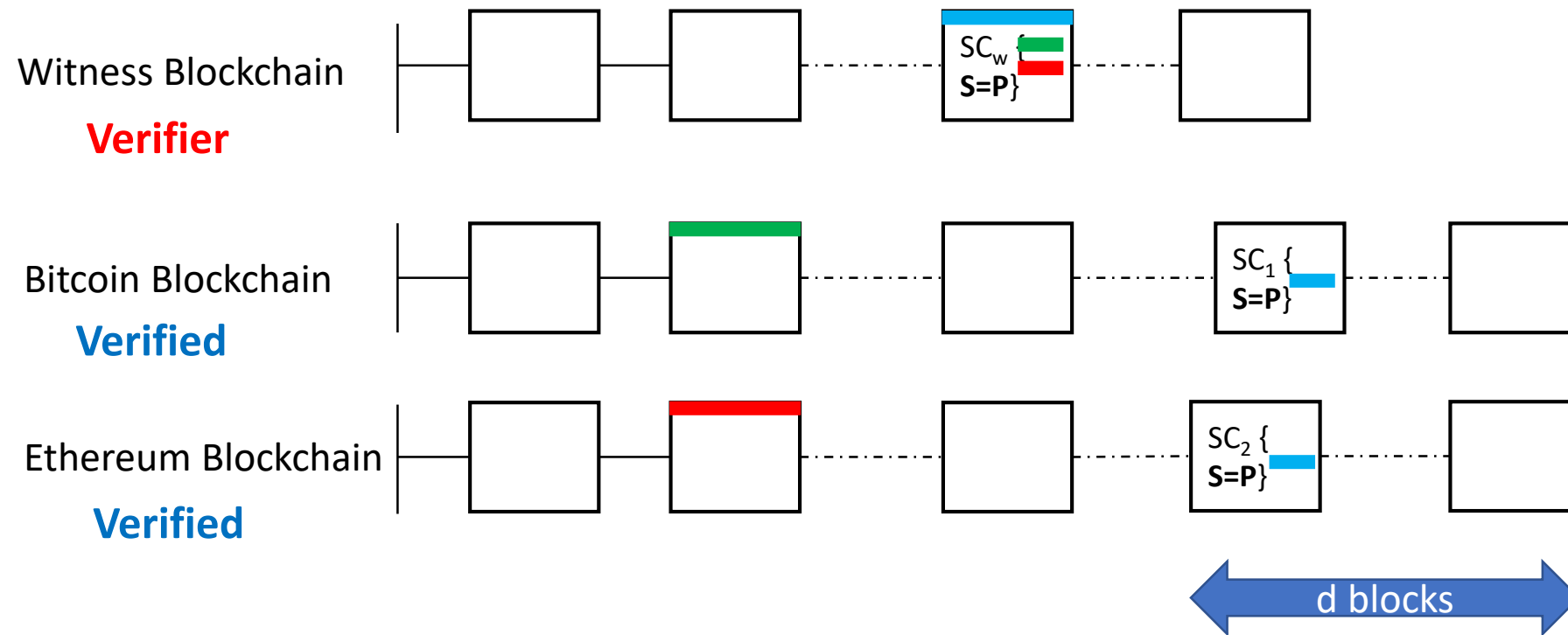
- Participants deploy their contracts in the corresponding blockchains
- Participants add the header of SC_w to their contracts



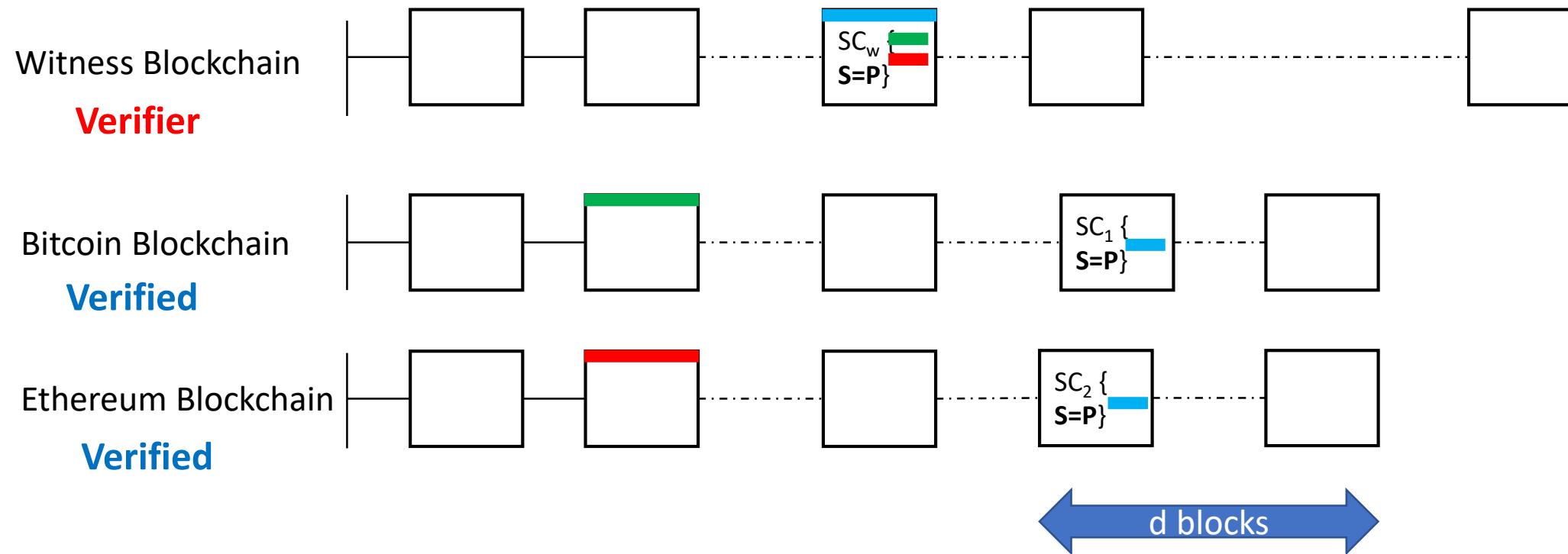
Protocol Sketch Cont'd



Protocol Sketch Cont'd

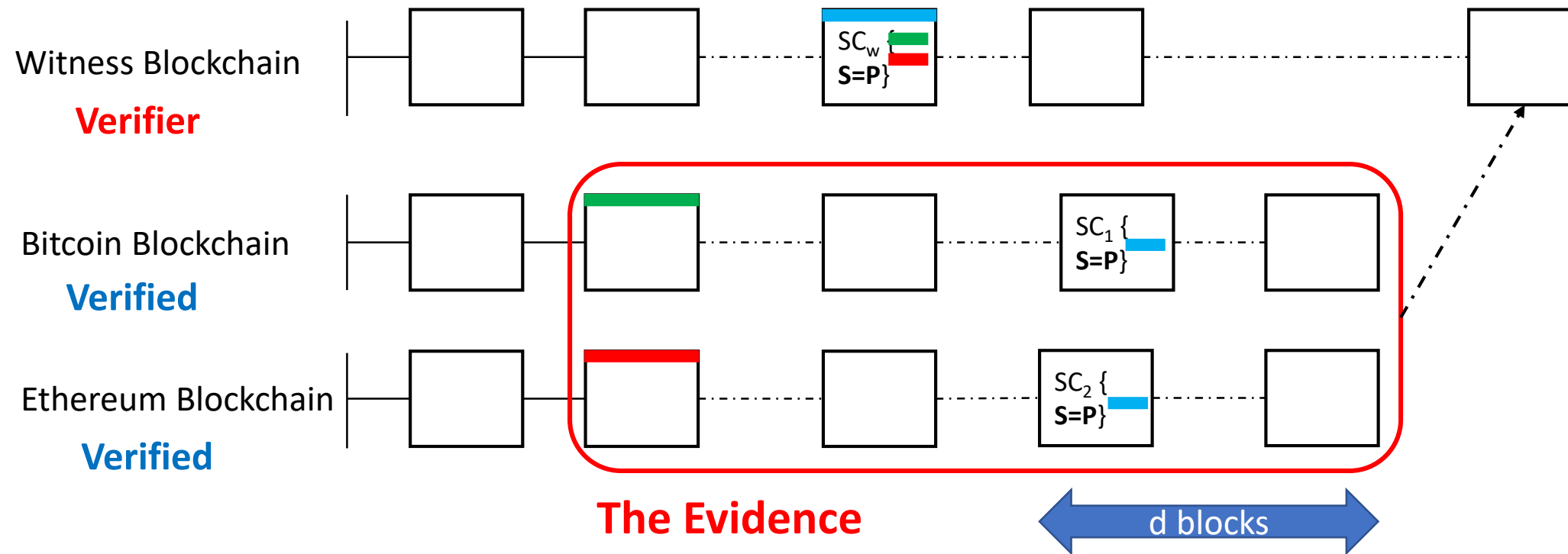


Protocol Sketch Cont'd



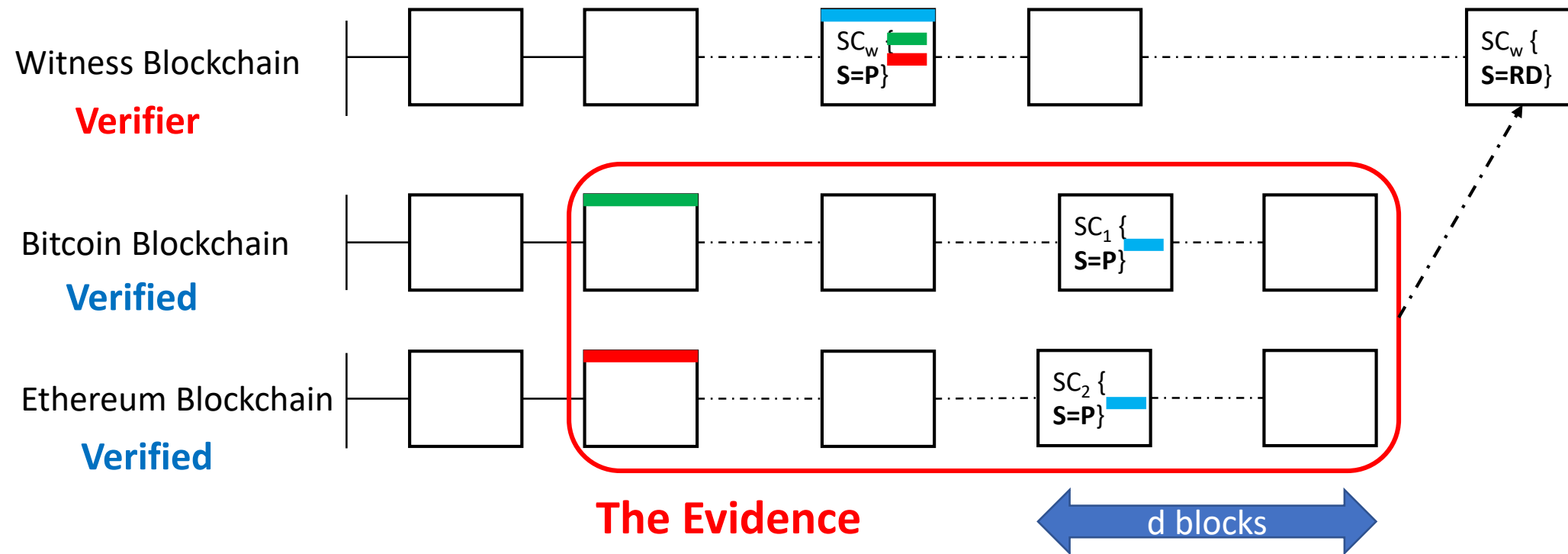
Protocol Sketch Cont'd

- Participants submit **evidence** of publishing the smart contracts in **Assets Blockchains**



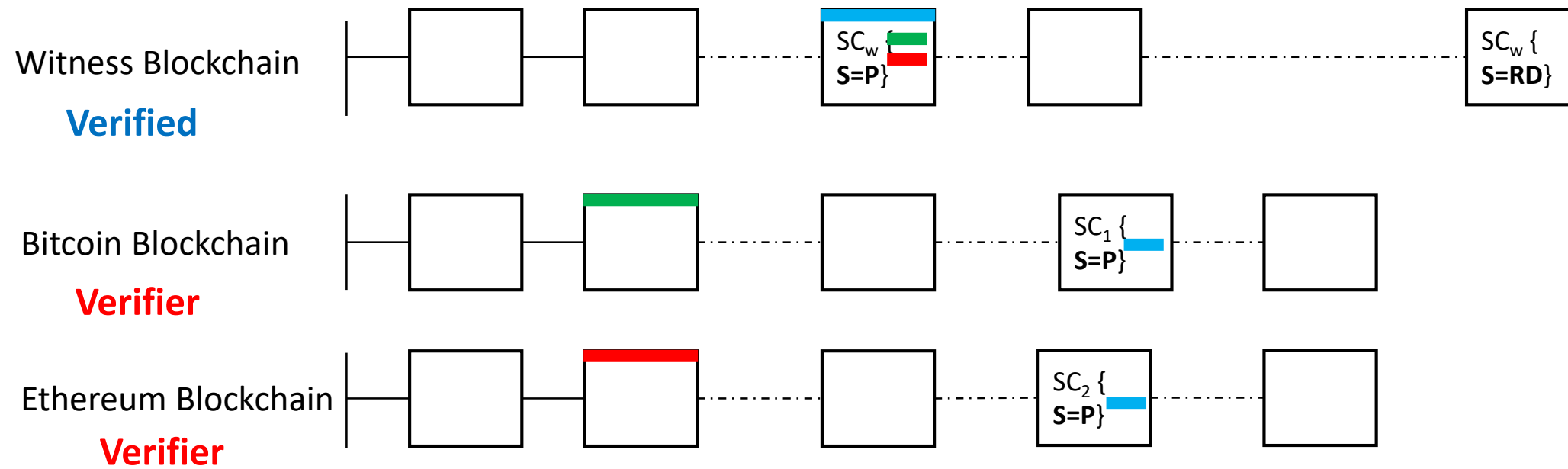
Protocol Sketch Cont'd

- 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)



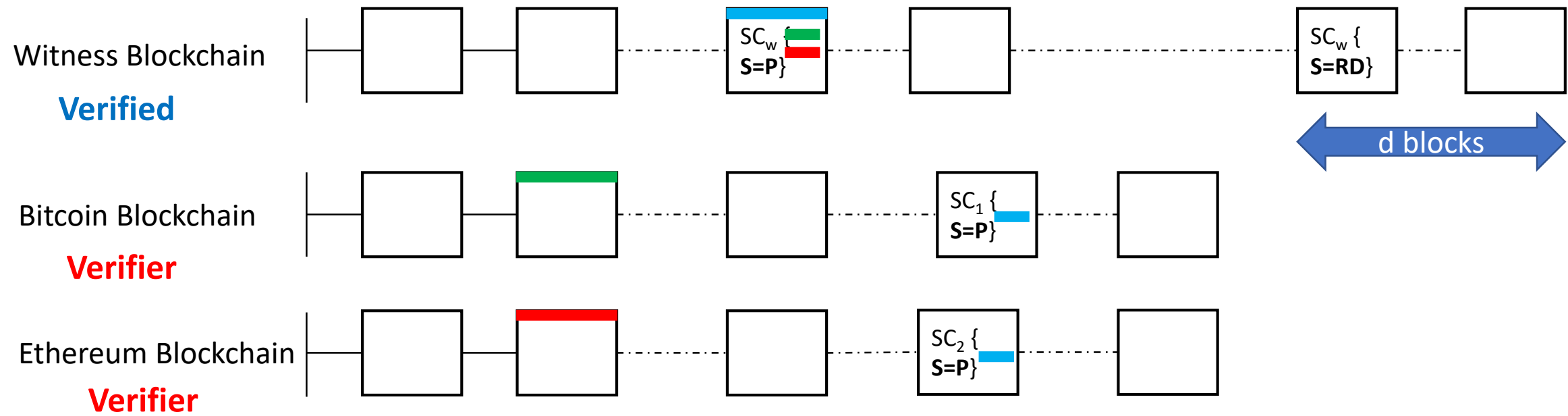
DSL at UCSB

Protocol Sketch Cont'd



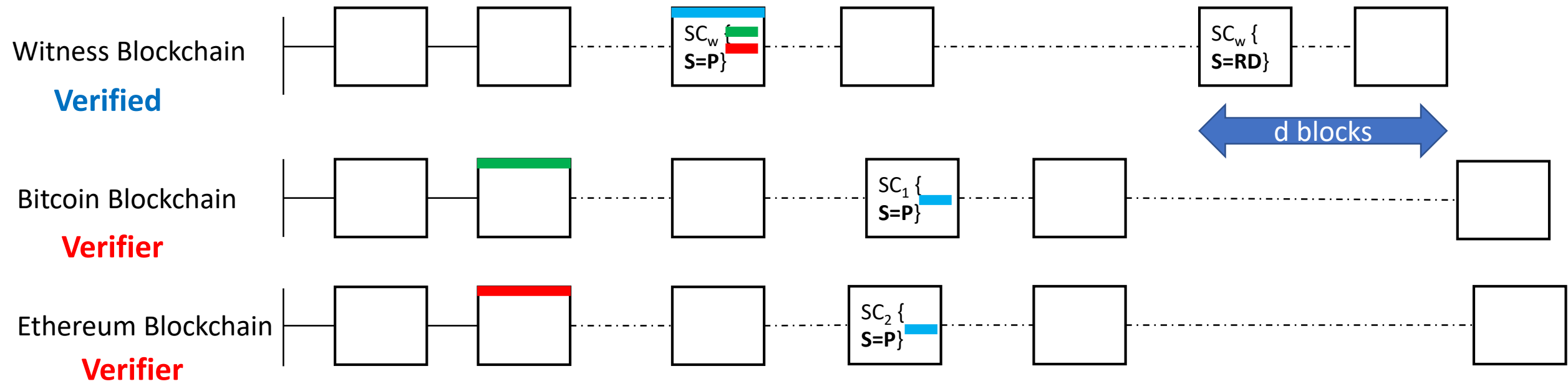
DSL at UCSB

Protocol Sketch Cont'd



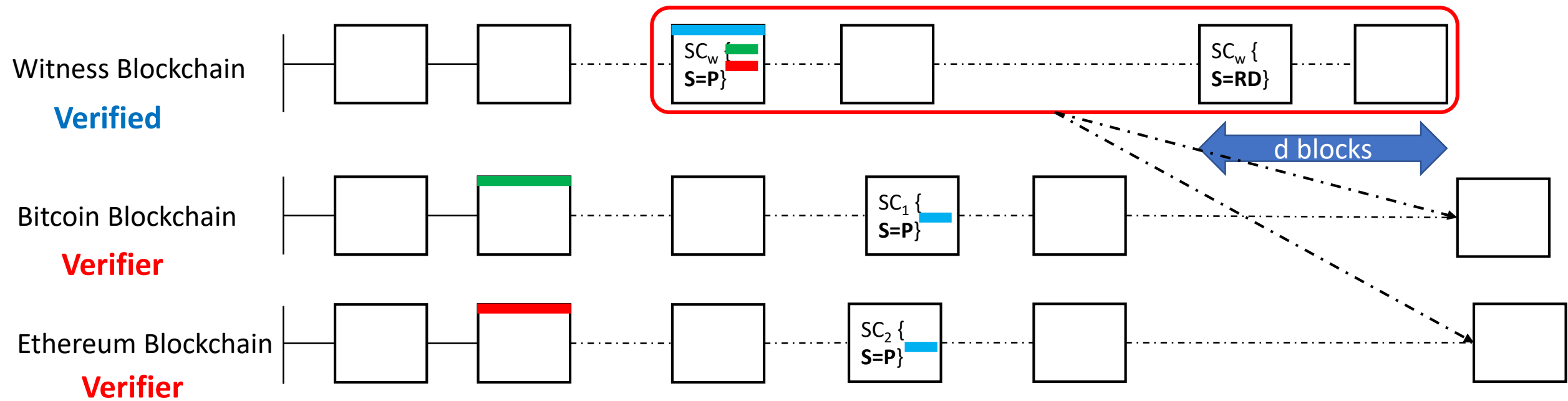
DSL at UCSB

Protocol Sketch Cont'd



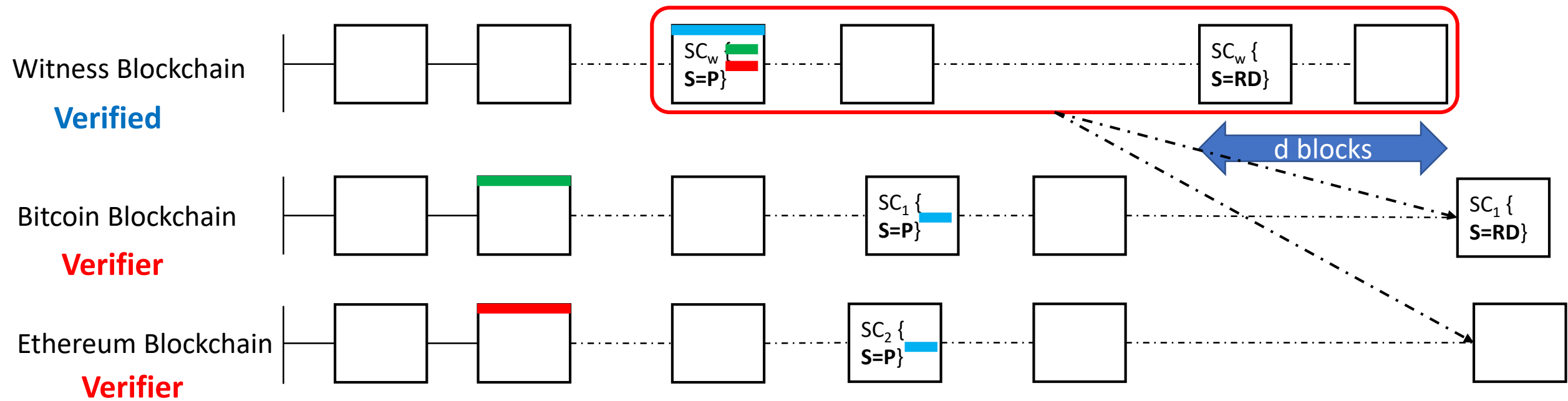
Protocol Sketch Cont'd

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



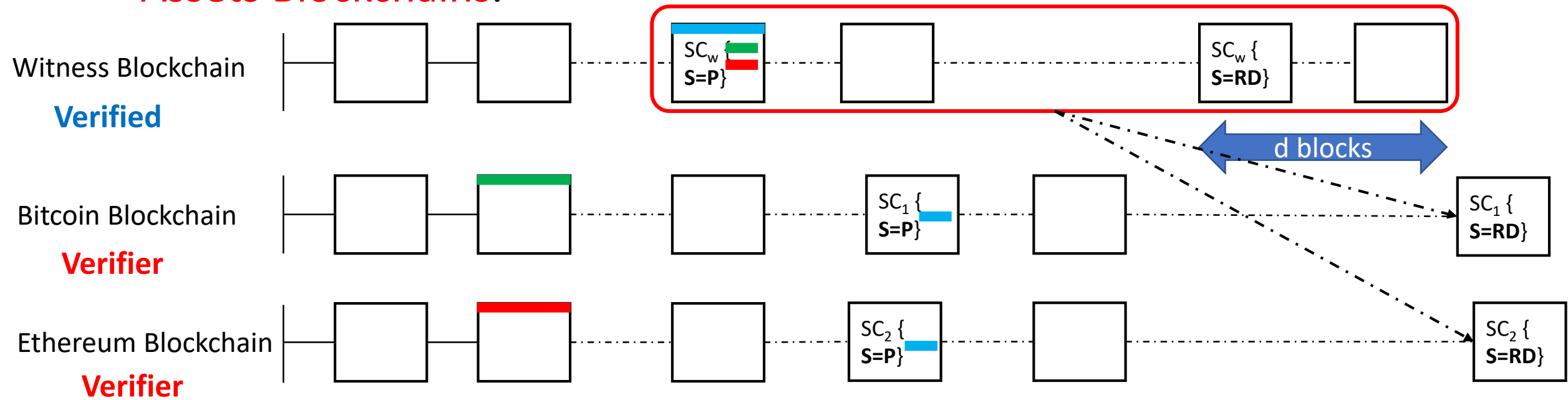
Protocol Sketch Cont'd

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



Protocol Sketch Cont'd

- 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

- 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

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

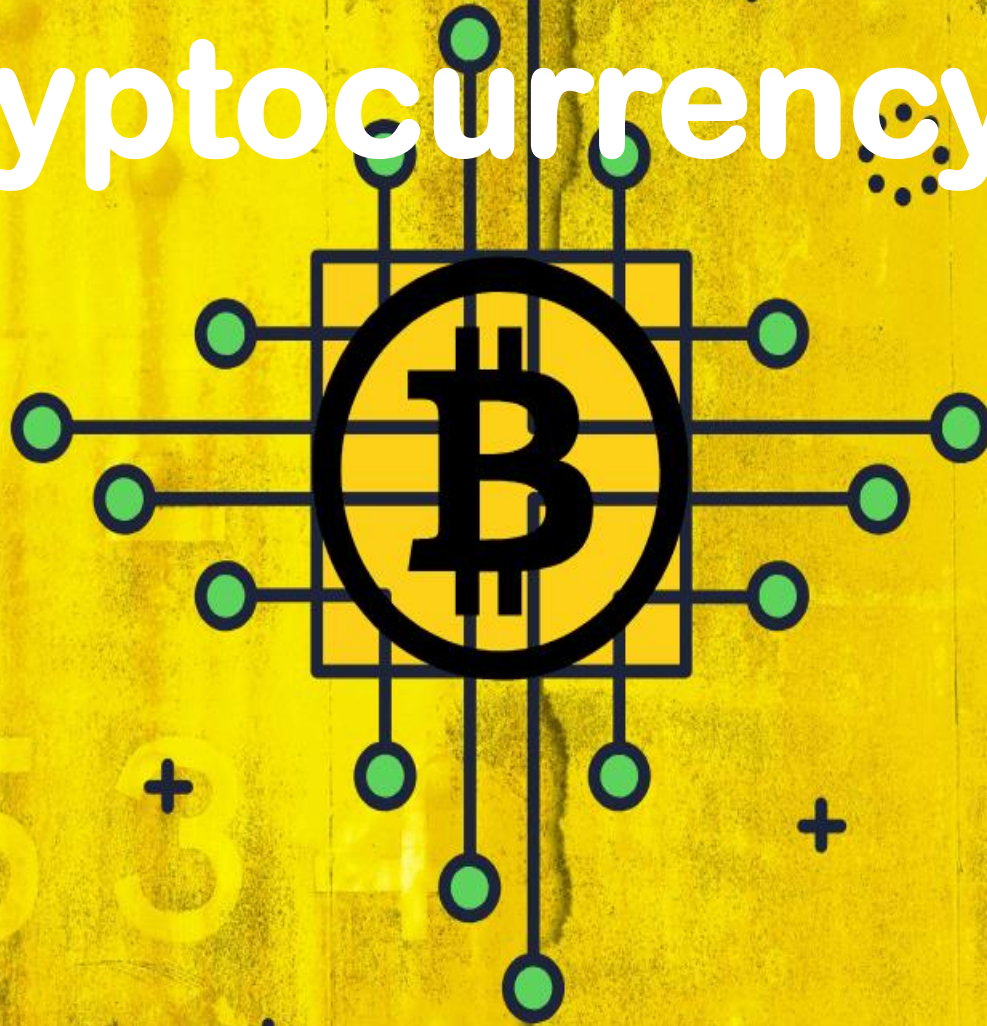
Atomic Commitment Across Blockchains

- 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
- 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 trade¹
 - Estimates suggest at least **20%** of all fish are **caught illegally**—yet only a tiny fraction are ever inspected².
 - Nearly **one in three** fish were **mislabeled** by sellers³
 - **87%** of **snapper** and **59%** of **tuna** were **mislabeled**⁴
 - **95%** of all **sushi** restaurants were serving **mislabeled** fish⁴

¹ Food and Agriculture Organization, United Nations. 2016. The State of World Fisheries and Aquaculture 2016.

² Stolen Seafood: The Impact of Pirate Fishing on Our Oceans. Oceana. 2013.

³ Miguel çngel Pardo, Elisa Jimžnez, Bego–a Pžrez-Villarreal. Misdescription incidents in seafood sector. 2016. Food Control 62 pages 277–283.

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 trade¹
 - Estimates suggest at least **20%** of all fish are **caught illegally**—yet only a tiny fraction are ever inspected².
 - Nearly **one in three** fish were **mislabeled** by sellers³
 - **87%** of **snapper** and **59%** of **tuna** were **mislabeled**⁴
 - **95%** of all **sushi** restaurants were serving **mislabeled** fish⁴
- **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.

² Stolen Seafood: The Impact of Pirate Fishing on Our Oceans. Oceana. 2013.

³ Miguel çngel Pardo, Elisa Jimžnez, Bego–a Pžrez-Villarreal. Misdescription incidents in seafood sector. 2016. Food Control 62 pages 277–283.

⁴ Oceana Study Reveals Seafood Fraud Nationwide. 2013.

DBL at UCSB Seafood Supply Chain in the real world



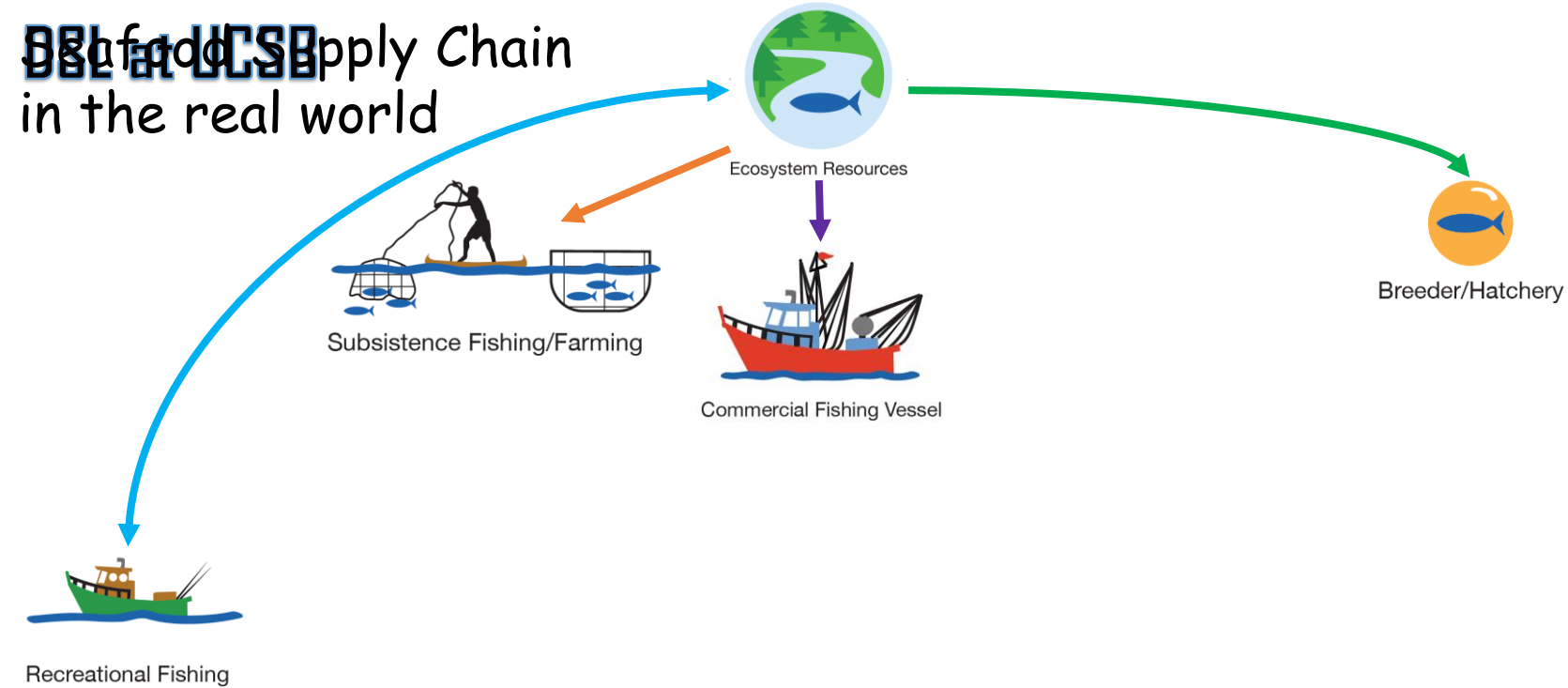
Seafood Supply Chain in Blockchain



Source: Advancing Traceability in
the Seafood Industry, FishWise



Seafood Supply Chain in the real world



Seafood Supply Chain in Blockchain

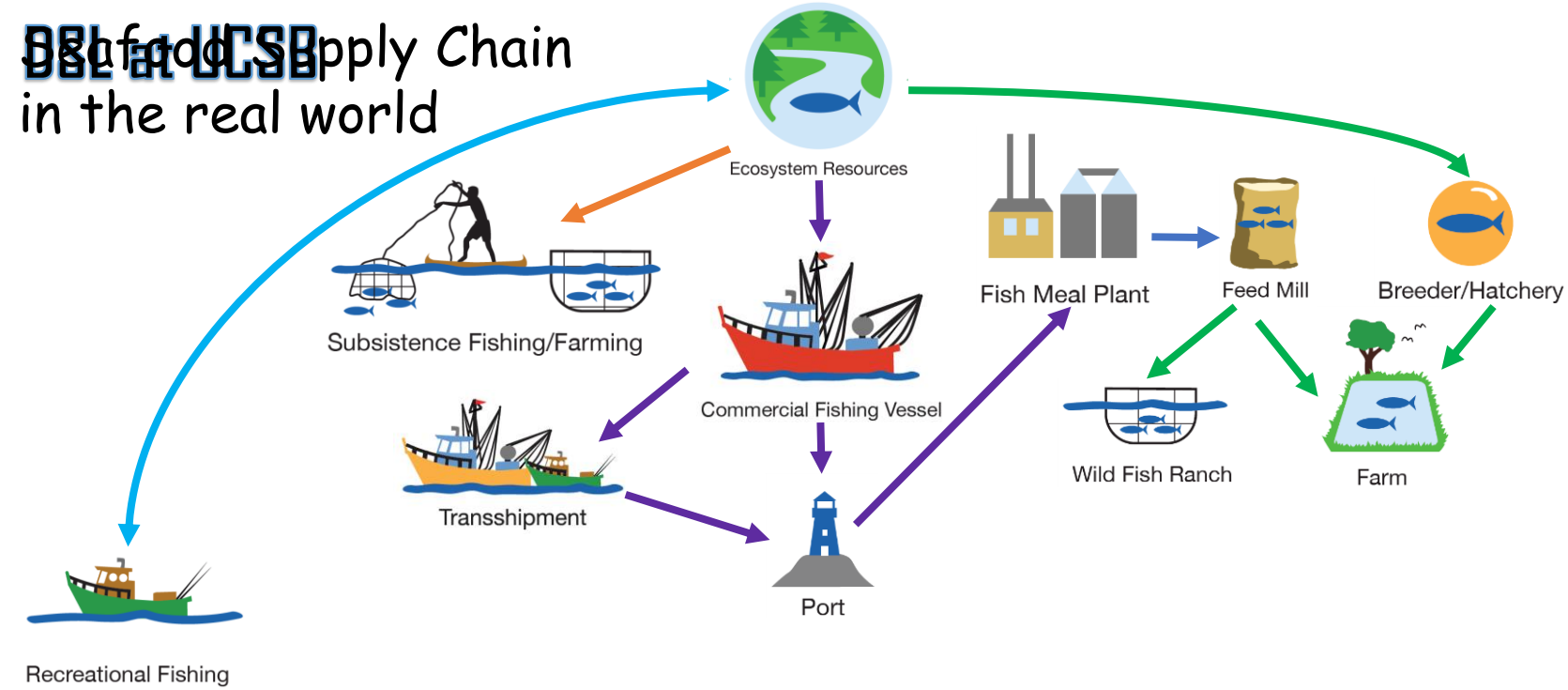
Seafood is caught by fishermen and physically tagged with IOT enabled sensors



Source: Advancing Traceability in the Seafood Industry, FishWise



Seafood Supply Chain in the real world



Seafood Supply Chain in Blockchain

Seafood is caught by fishermen and physically tagged with IOT enabled sensors

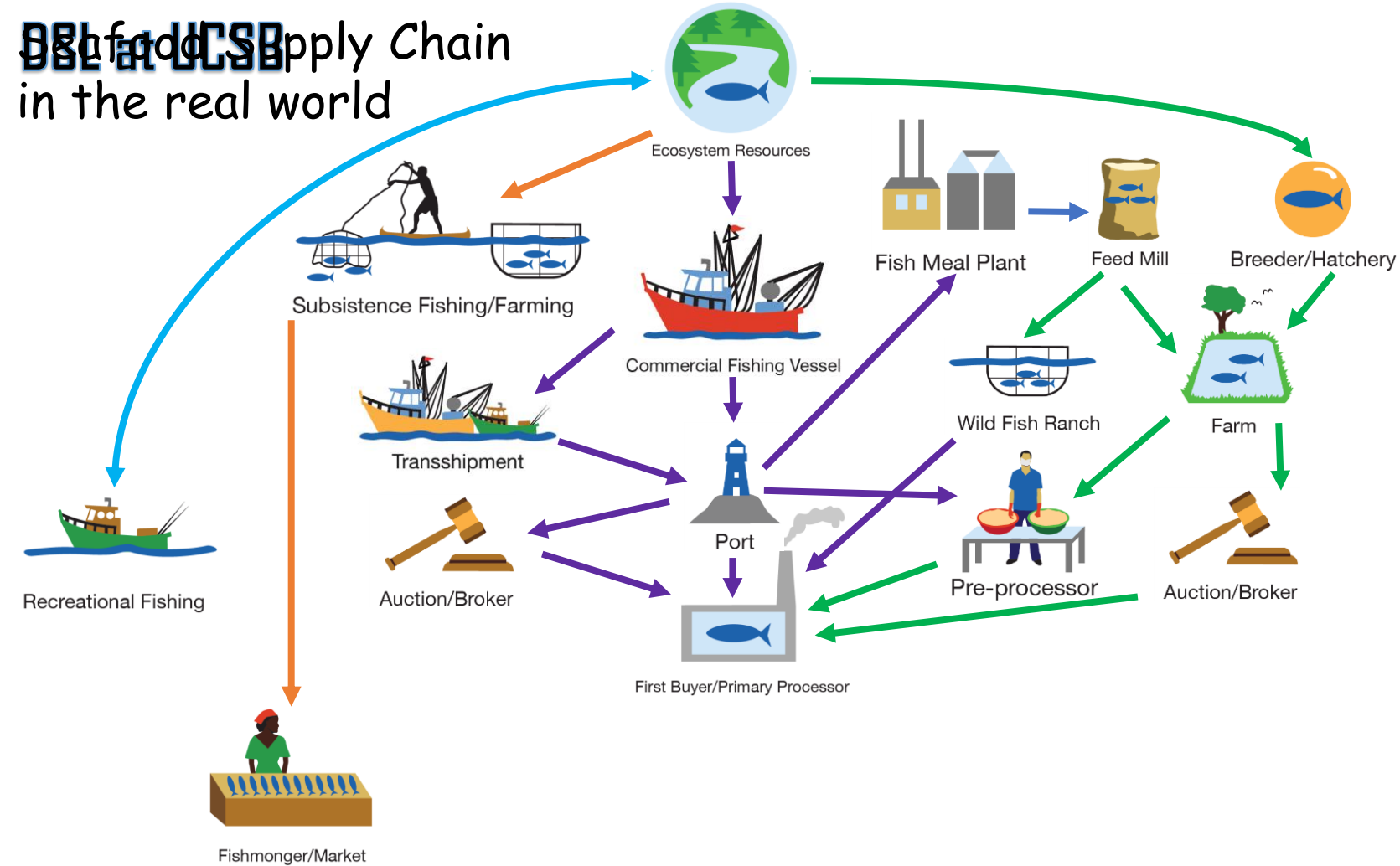
Sensors continuously transmit data about time and location to Blockchain



Source: Advancing Traceability in the Seafood Industry, FishWise



Seafood Supply Chain in the real world



Seafood Supply Chain in Blockchain

Seafood is caught by fishermen and physically tagged with IOT enabled sensors

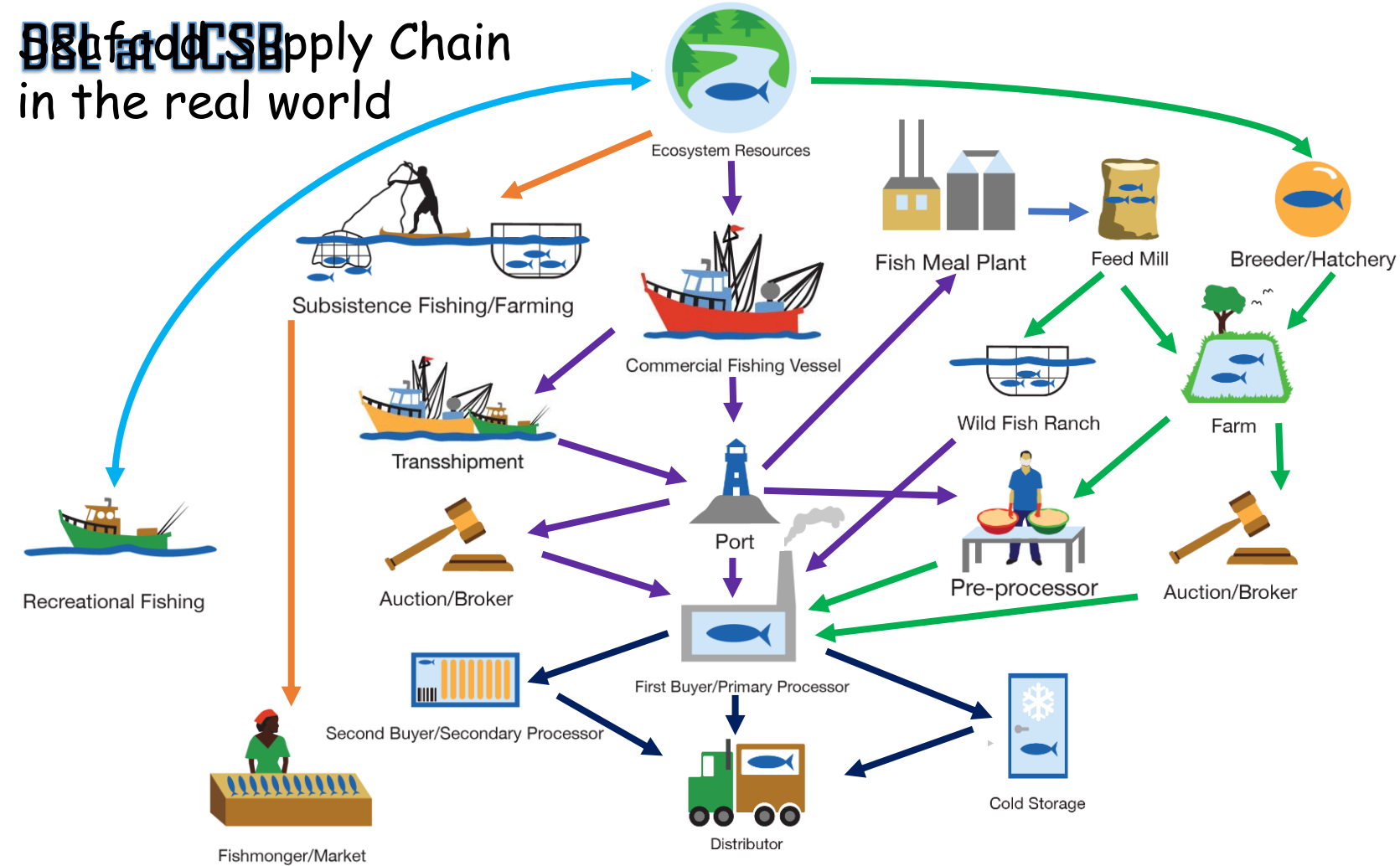
Sensors continuously transmit data about time and location to Blockchain



Source: Advancing Traceability in the Seafood Industry, FishWise



Seafood Supply Chain in the real world

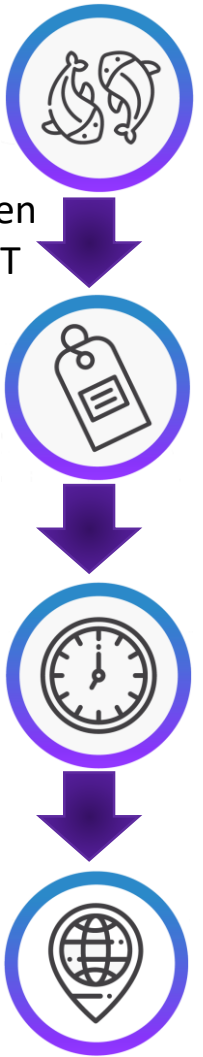


Seafood Supply Chain in Blockchain

Seafood is caught by fishermen and physically tagged with IOT enabled sensors

Sensors continuously transmit data about time and location to Blockchain

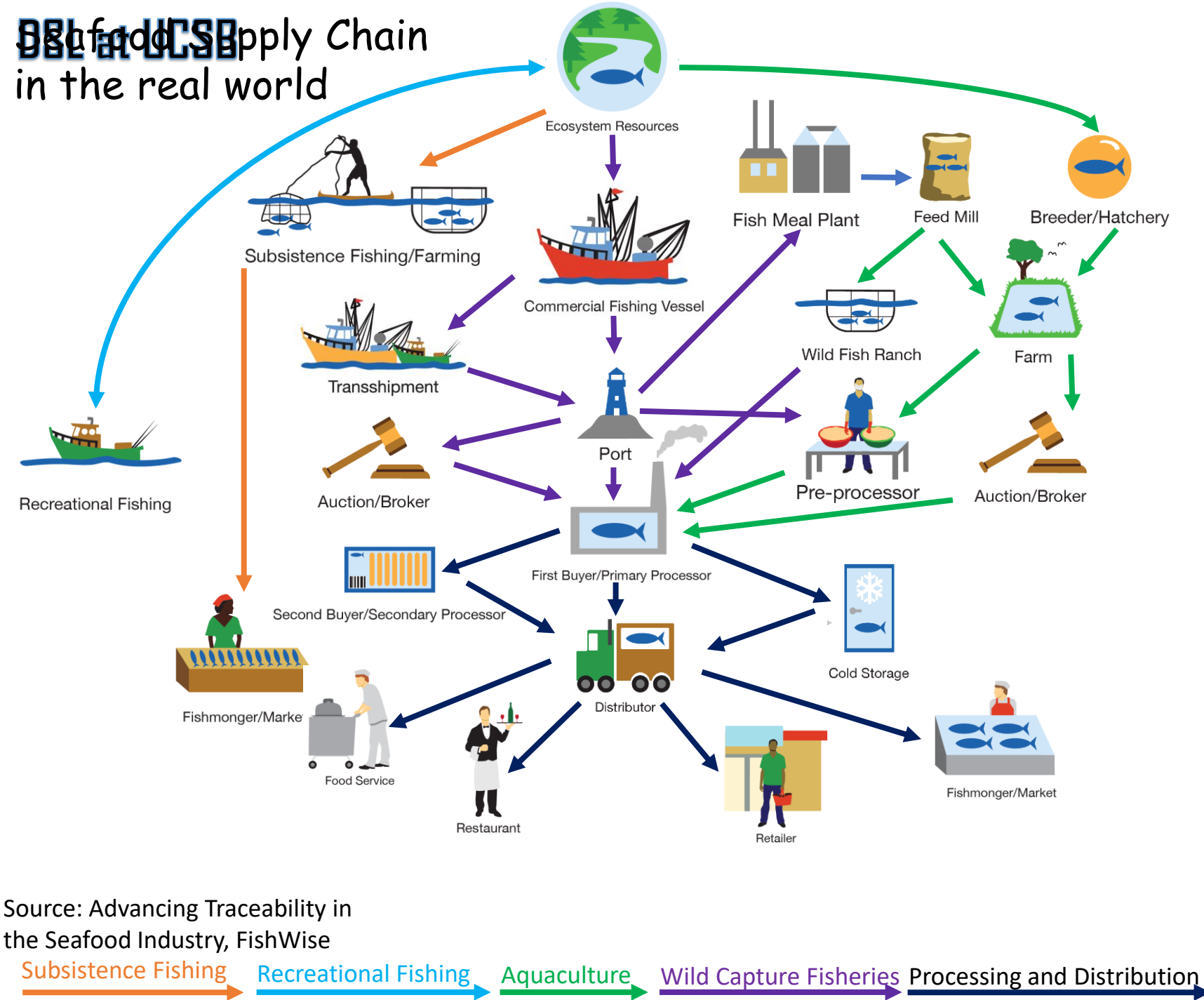
Blockchain facilitates and tracks possession changes through the distribution



Source: Advancing Traceability in the Seafood Industry, FishWise



Seafood Supply Chain in the real world



Seafood Supply Chain in Blockchain

Seafood is caught by fishermen and physically tagged with IOT enabled sensors

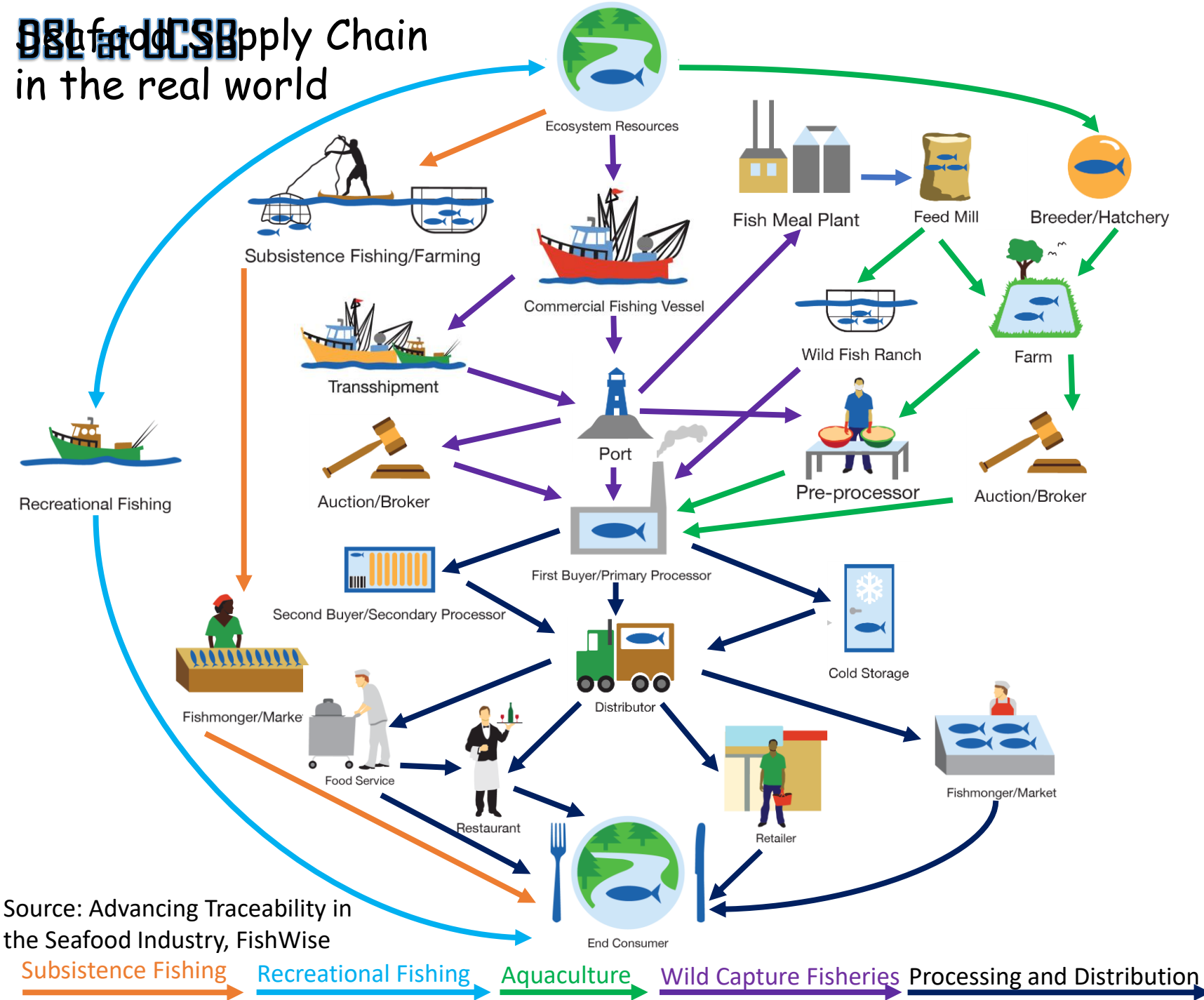
Sensors continuously transmit data about time and location to Blockchain

Blockchain facilitates and tracks possession changes through the distribution

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



Seafood Supply Chain in the real world



Seafood Supply Chain in Blockchain

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



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

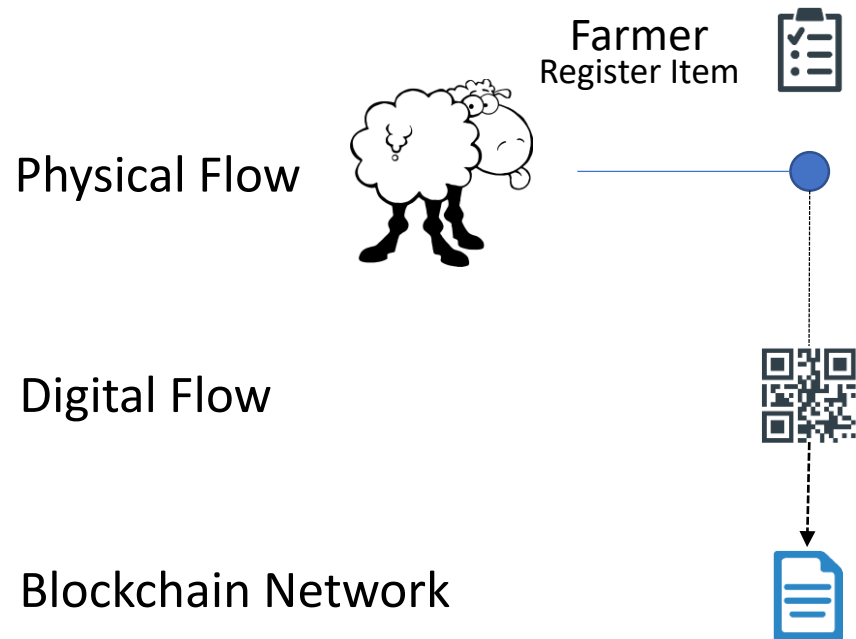


Digital Flow

Blockchain Network

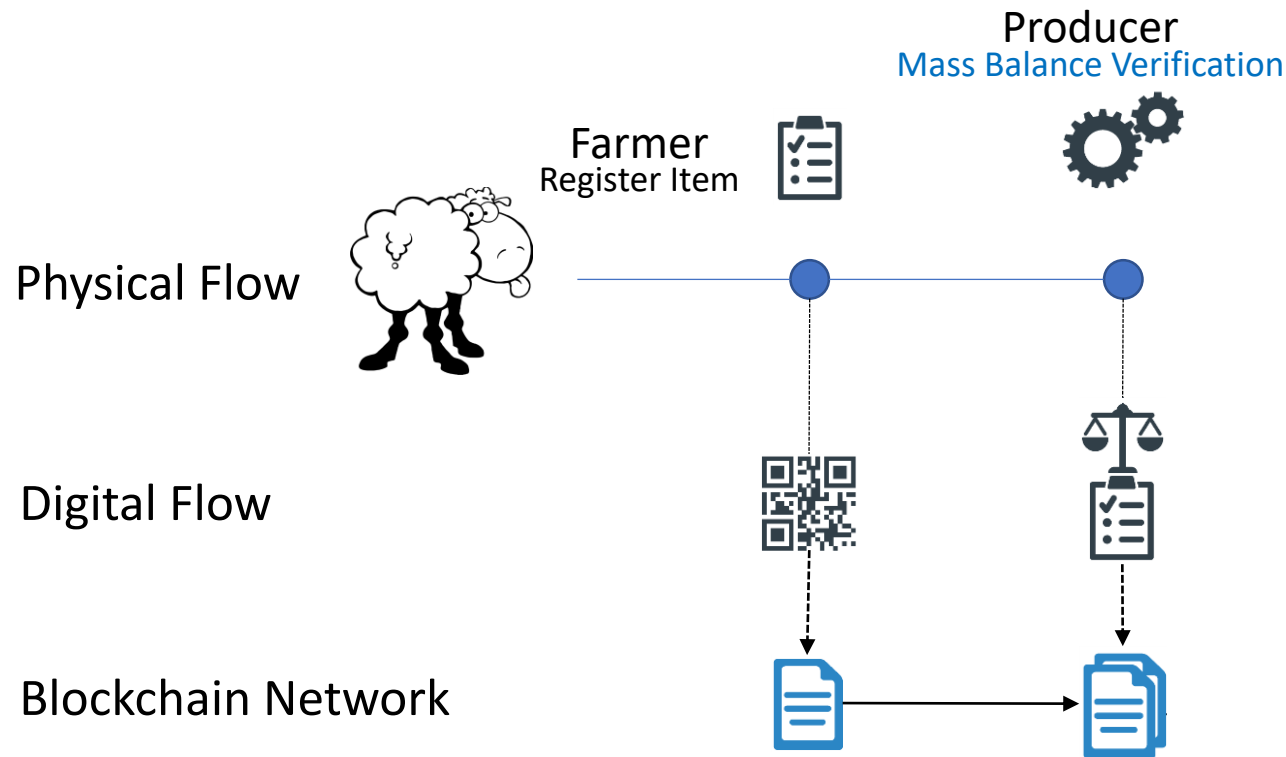
Blockchain for Supply Chains

- 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



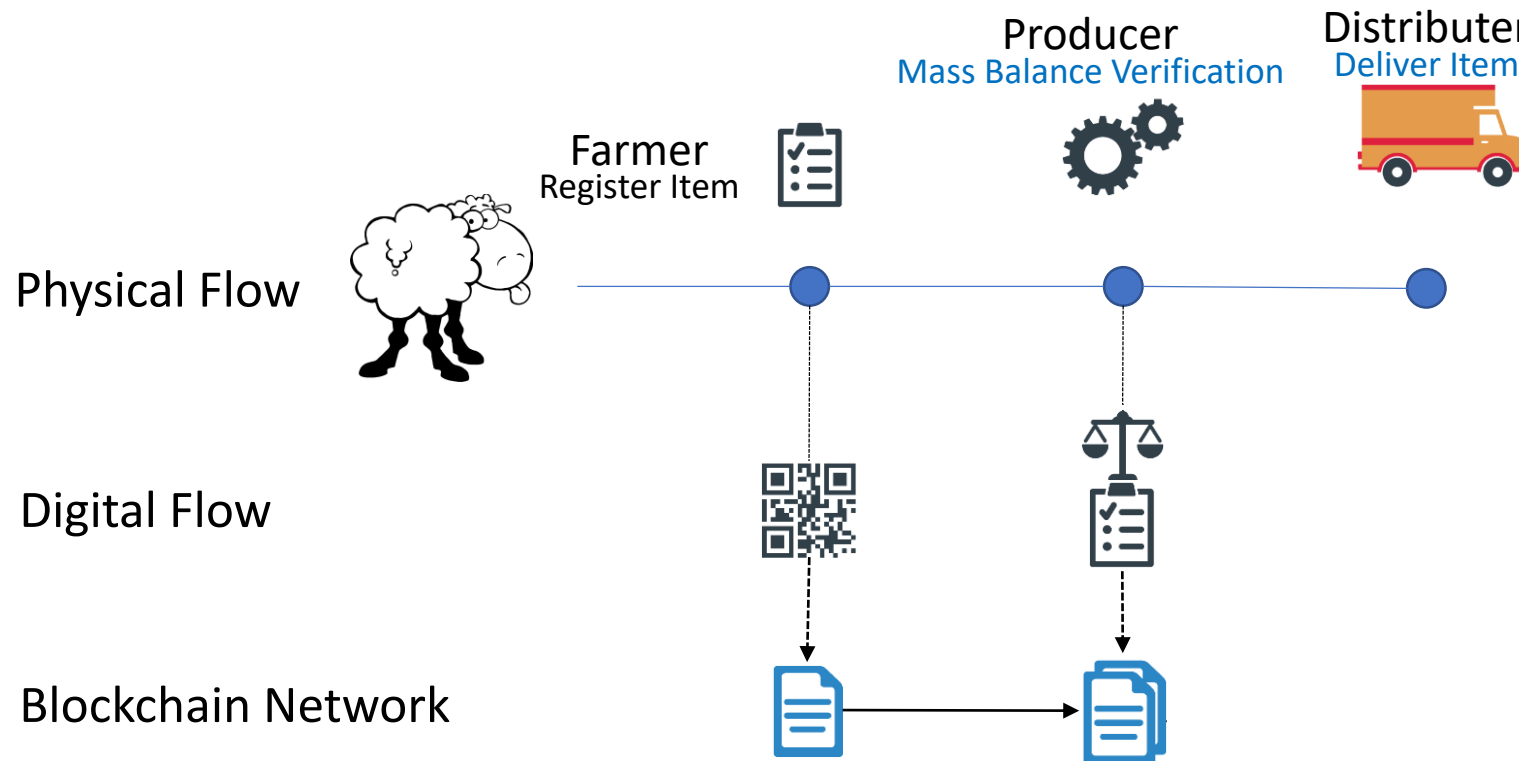
Blockchain for Supply Chains

- 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



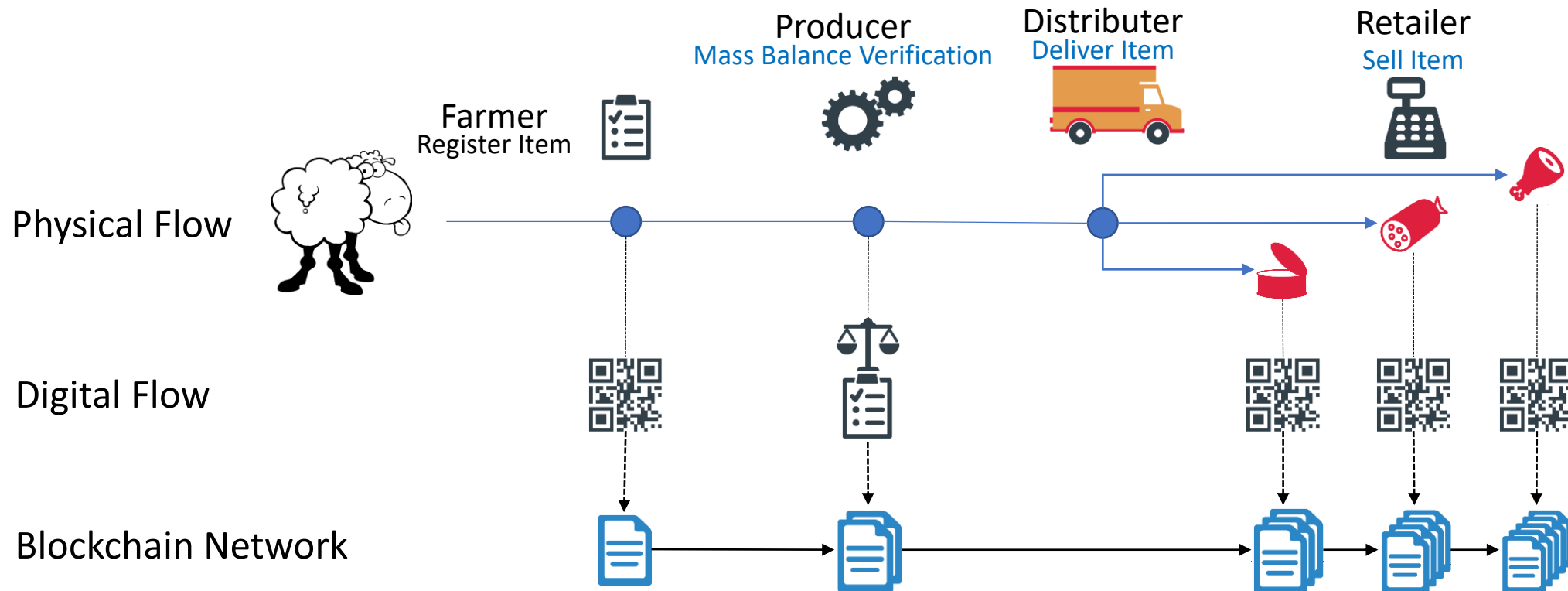
Blockchain for Supply Chains

- 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



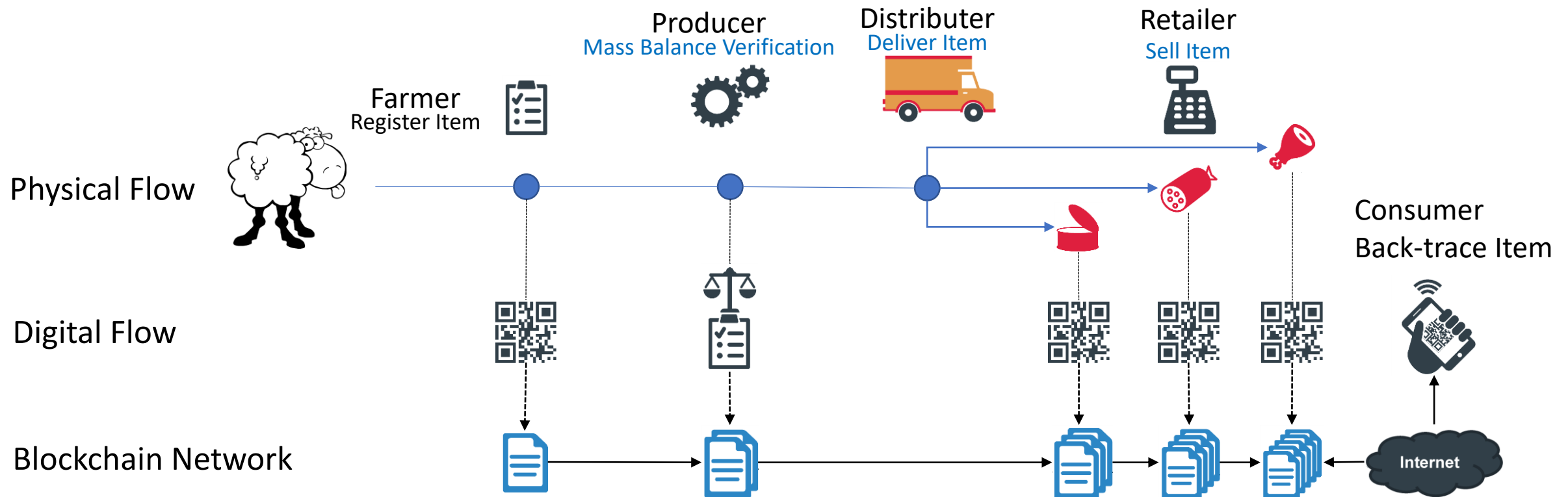
Blockchain for Supply Chains

- 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?!

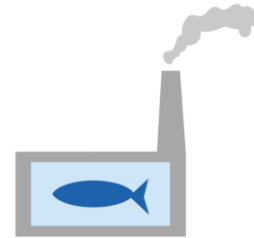
In Supply Chain Participants are **known** and **Identified**



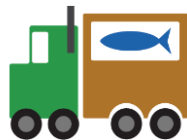
Commercial Fishing Vessel



Transshipment



First Buyer/Primary Processor



Distributor



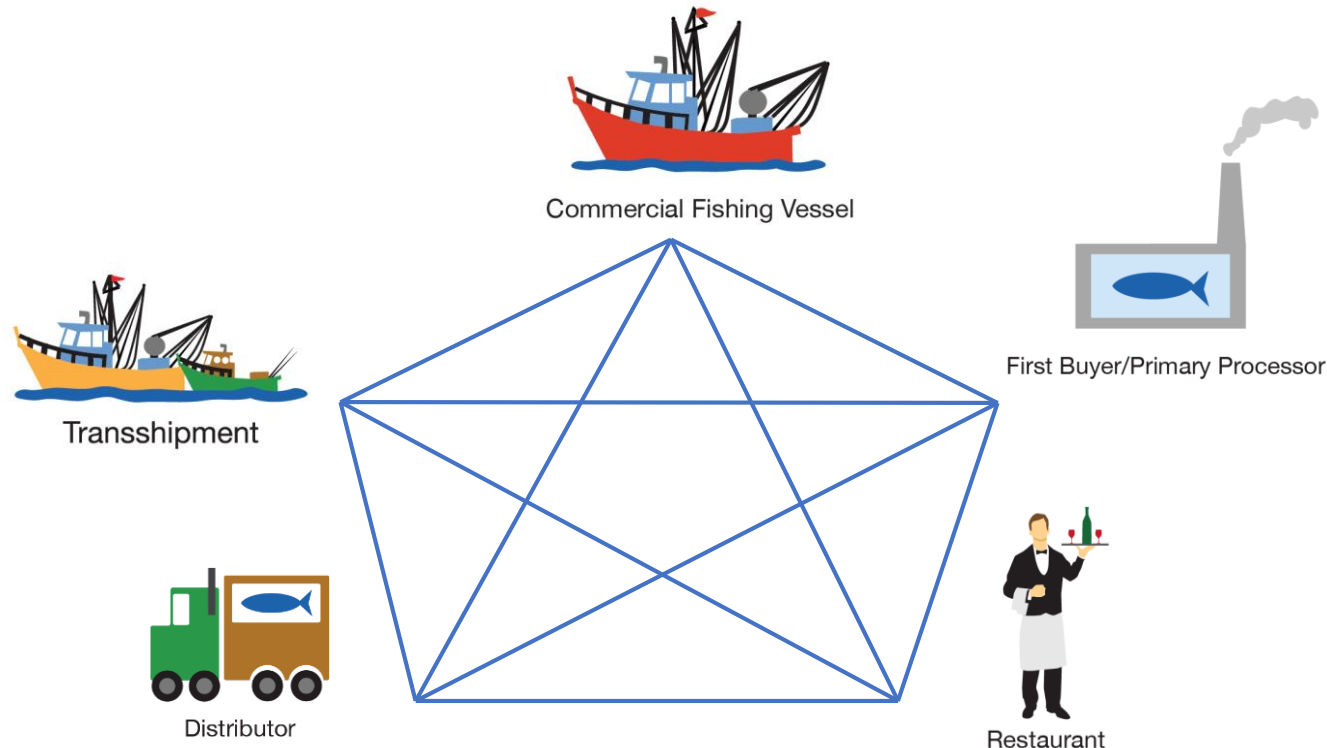
Restaurant

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

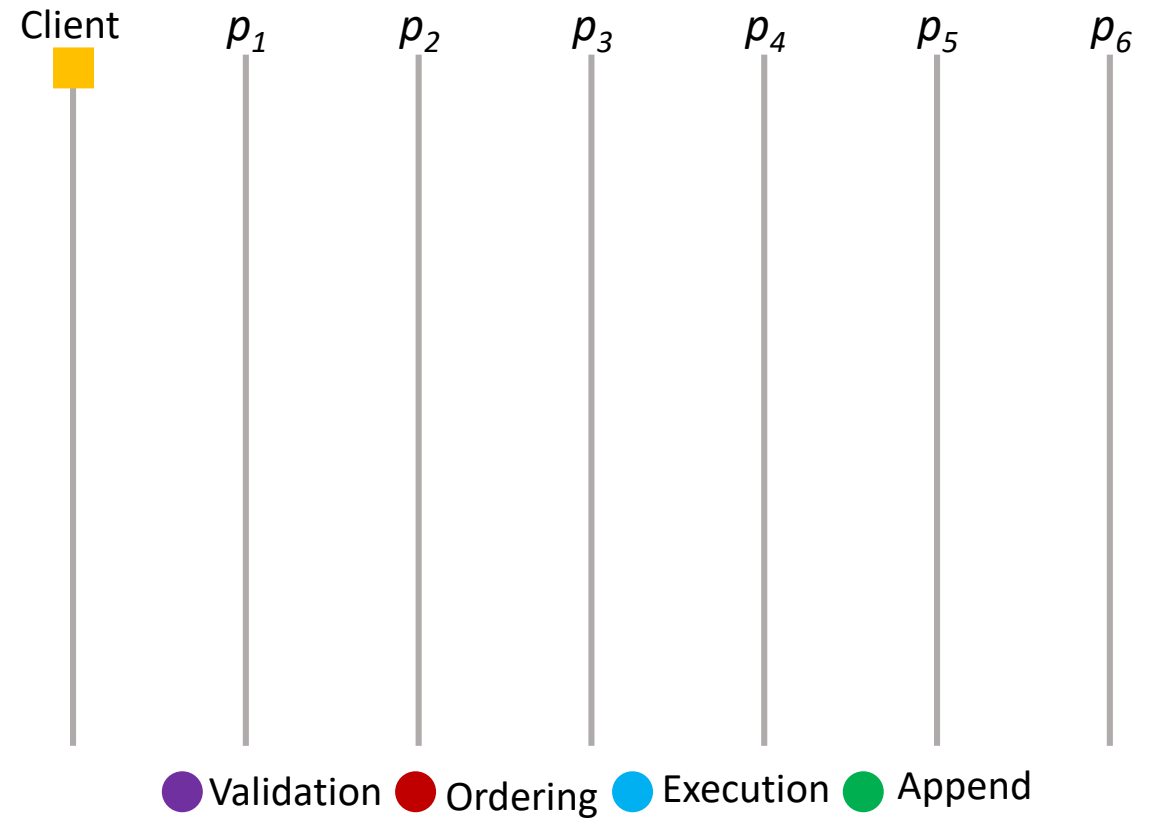
- Run a blockchain among a set of **known, identified** participants
- Provides a way to secure the interactions among a group of entities that have a common goal but which **do not fully trust each other**
- The ledger is distributed among all the nodes

	Permissionless	Permissioned
Participants	Anonymous, Could be malicious	Known, Identified
Consensus Mechanisms	Proof of Work, Proof of Stake, ... <ul style="list-style-type: none"> • Large energy consumption • No finality • 51% attack 	Byzantine fault tolerance Consensus, e.g., PBFT <ul style="list-style-type: none"> • Lighter • Faster • Low energy consumption • Enable finality
Transaction Approval time	Long (Bitcoin: 10 min or more)	Short (100x msec)

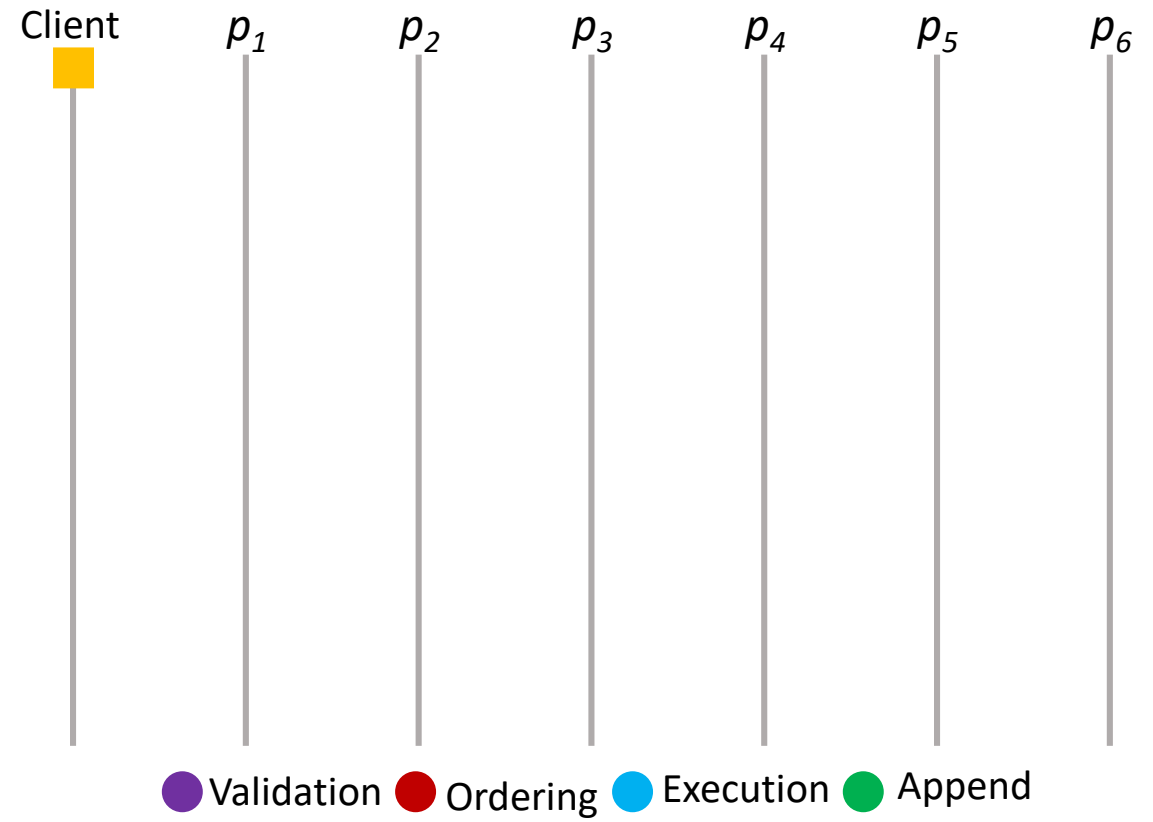
Consensus Protocols in Permissioned Networks

- 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

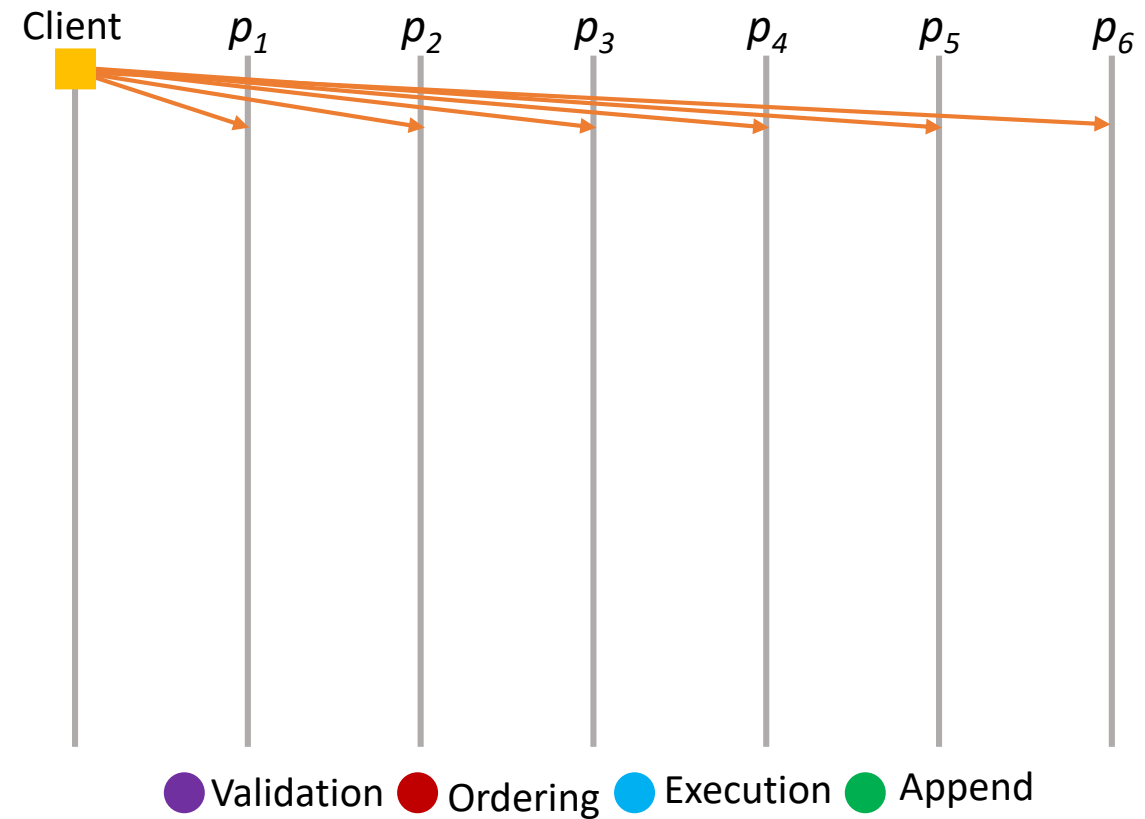


Bitcoin review



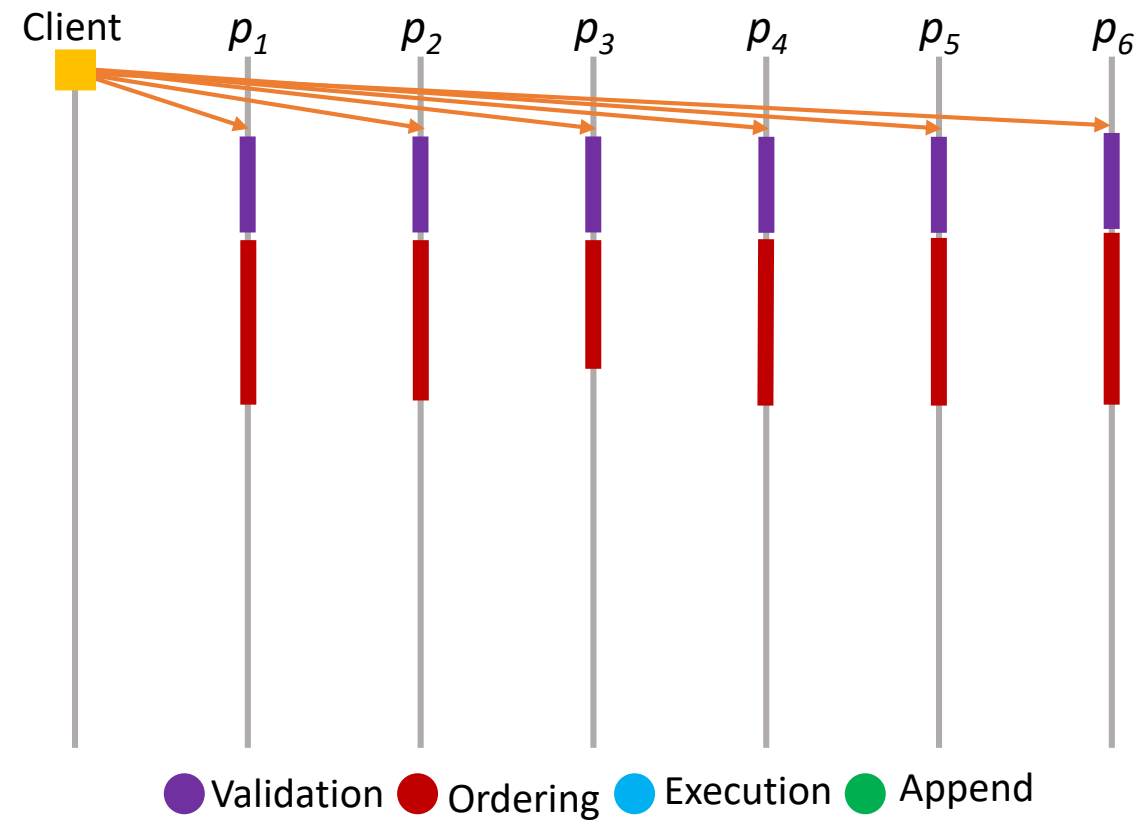
Bitcoin review

- Clients **multicasts** their requests



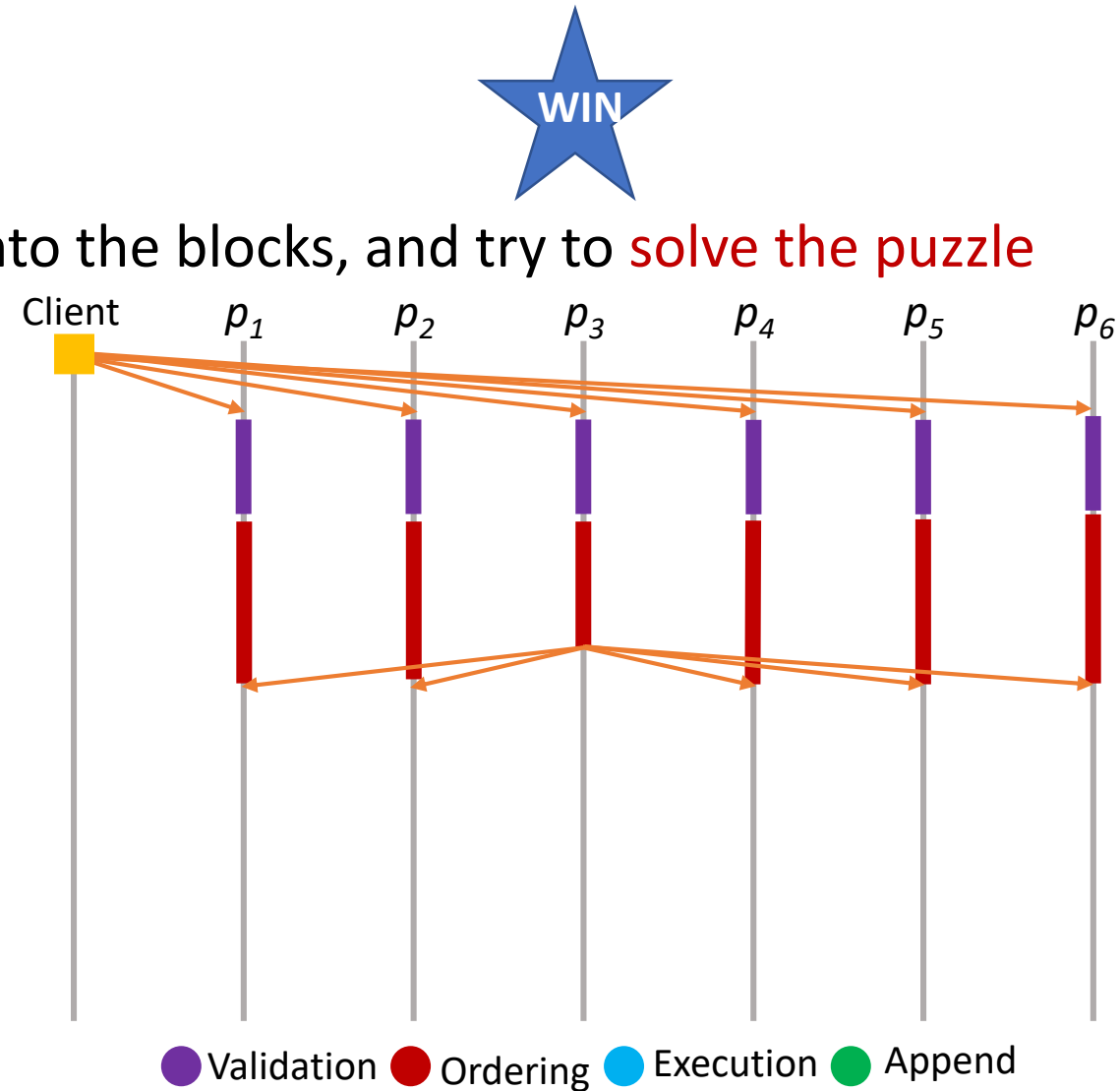
Bitcoin review

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



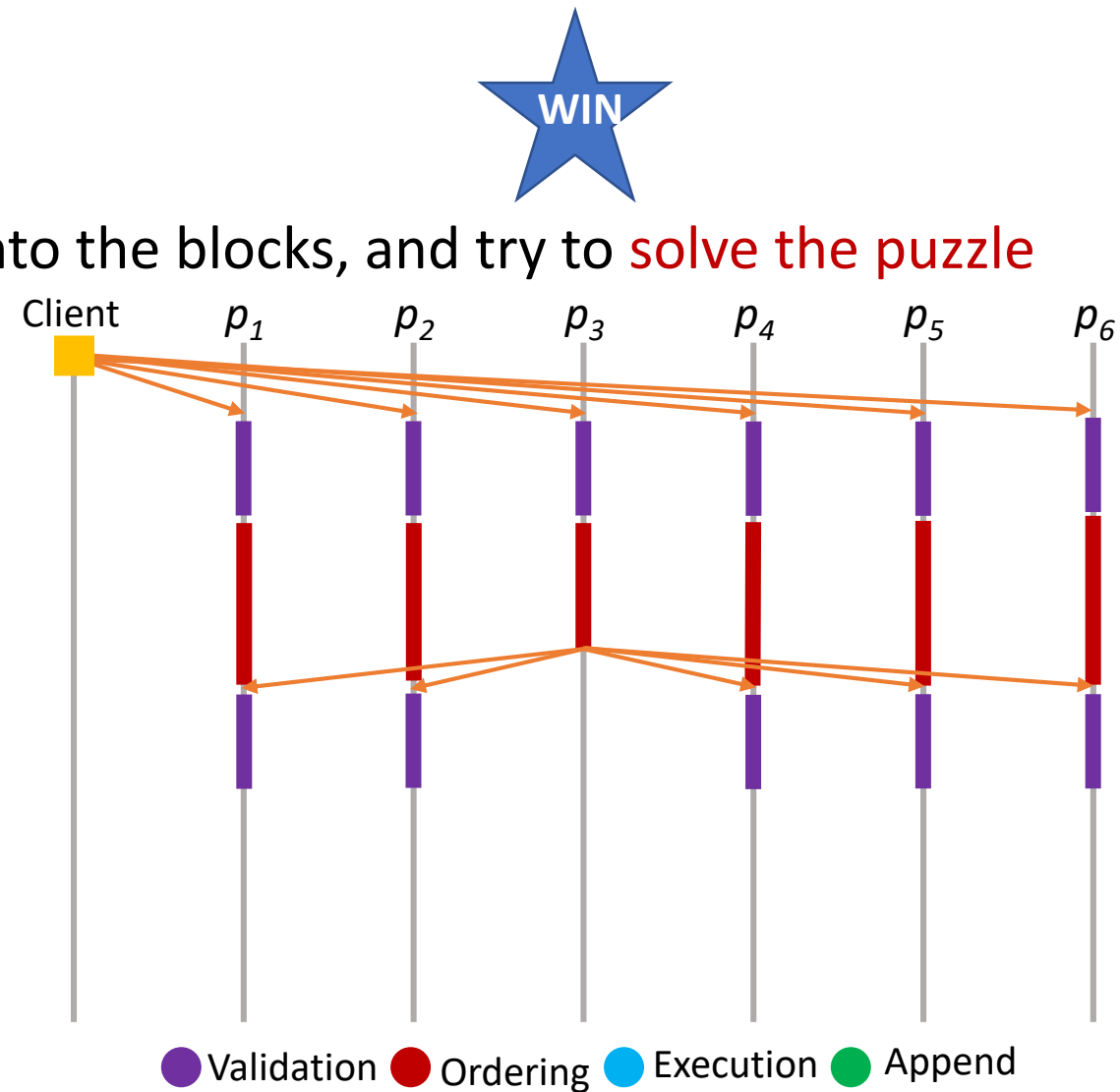
Bitcoin review

- 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



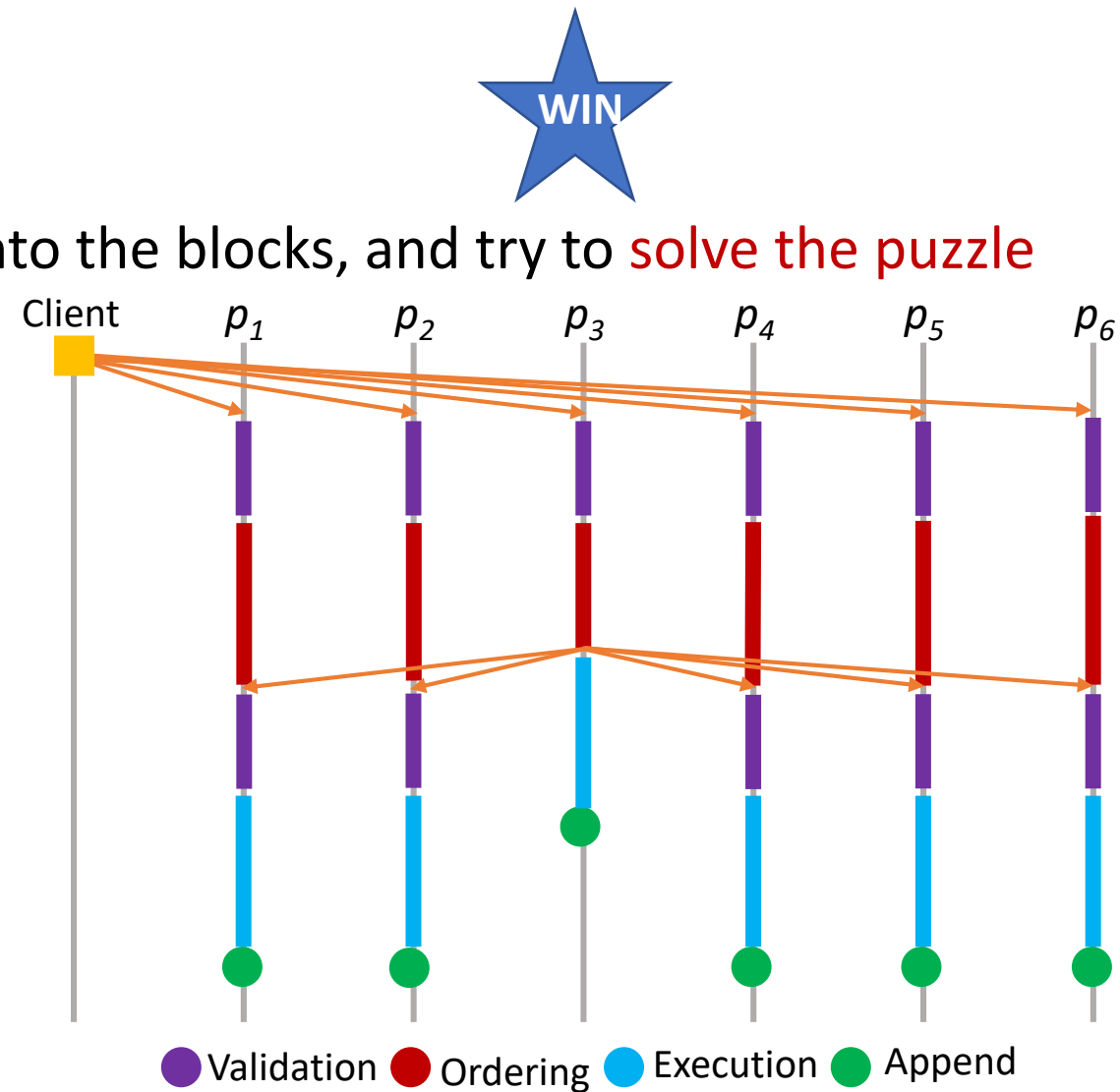
Bitcoin review

- 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

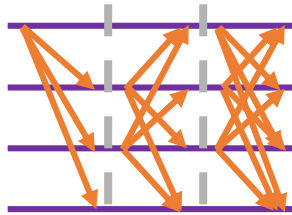


Order-execute Architecture

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)



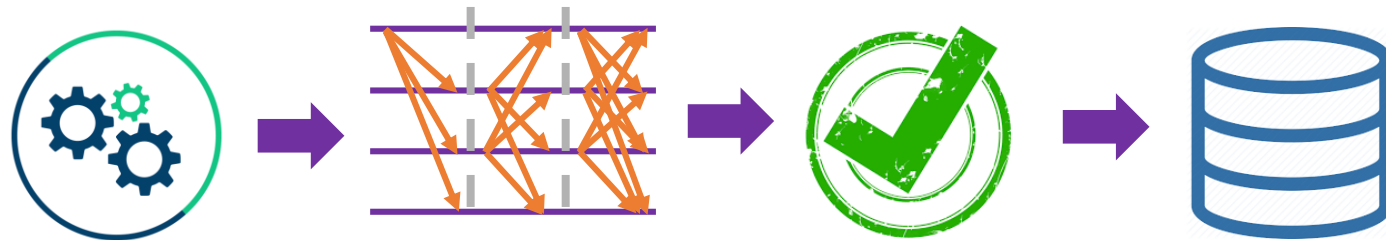
Execute-Order-Validate Architecture

- 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



Androulaki, E., et al., Hyperledger fabric (2018) a distributed operating system for permissioned blockchains. EuroSys, ACM.



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

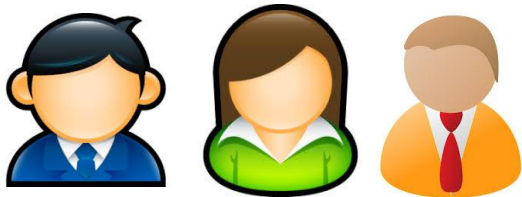
- Three types of Nodes: Clients, Endorsers, and Orderers

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.



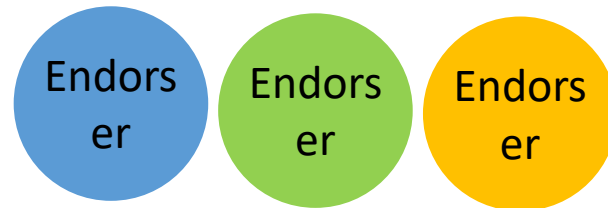
Clients (of different applications)

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



Clients (of different applications)



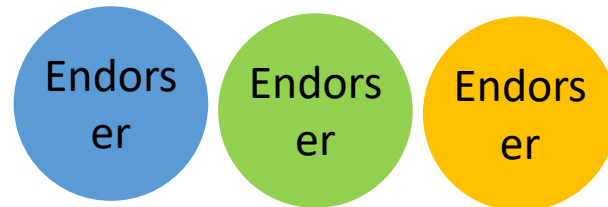
Endorsers (of different applications)

Hyperledger Fabric

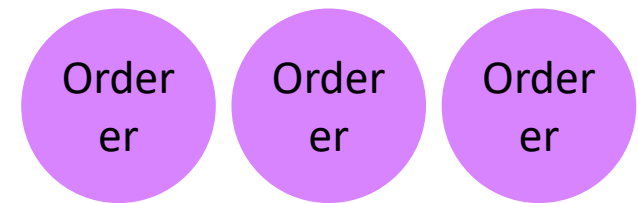
- Three types of Nodes: Clients, Endorsers, and Orderers
 - **Clients** send transactions to be executed.
 - **Endorsers** execute transaction proposals and validate transactions.
 - All endorsers maintain the blockchain ledger
 - Each application has its own set of endorsers
 - **Orderers** establish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



Clients (of different applications)



Endorsers (of different applications)

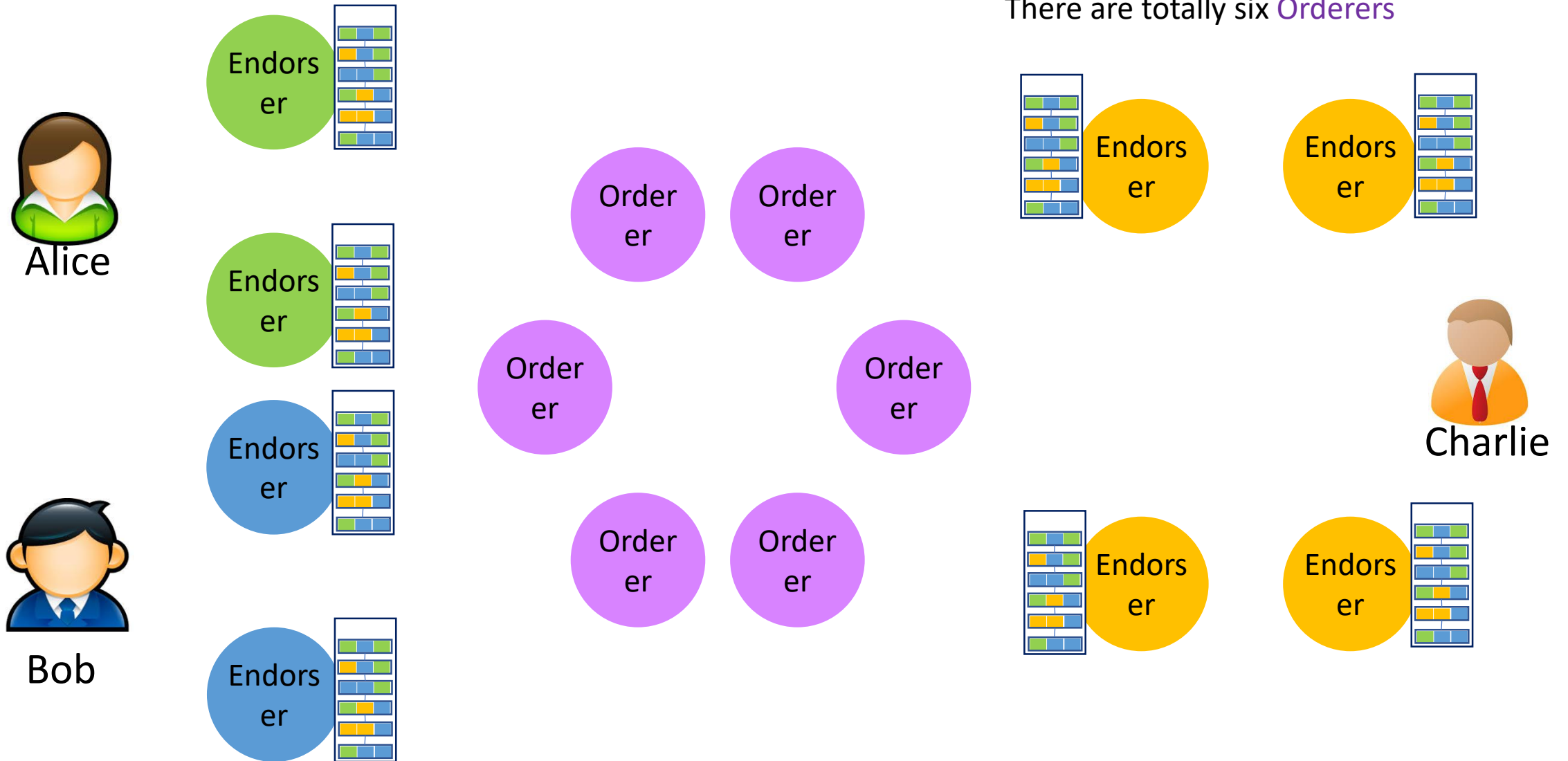


Orderers

Hyperledger Fabric

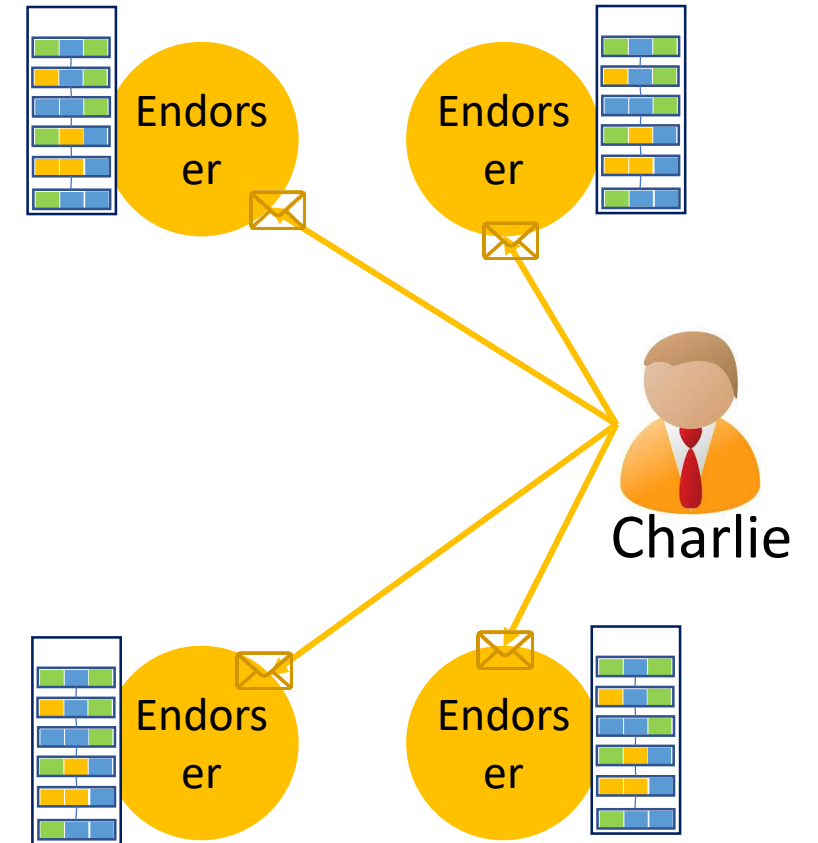
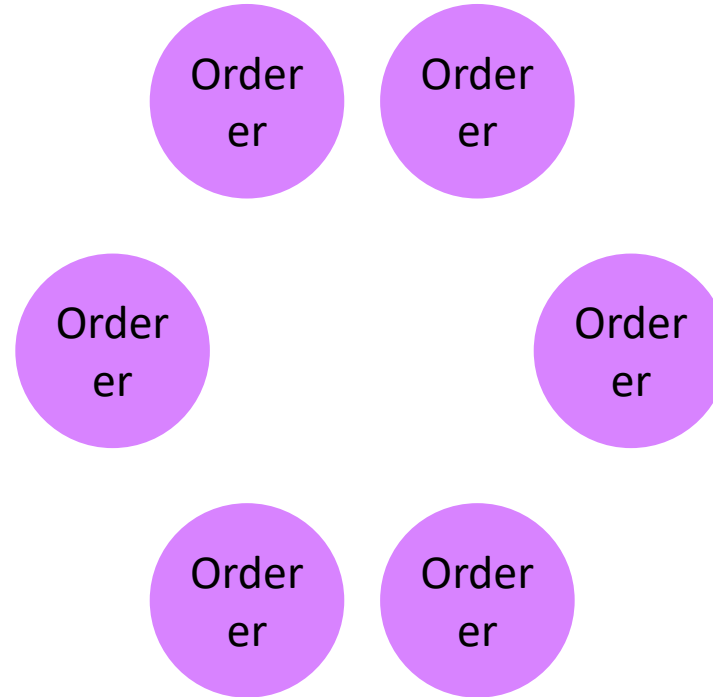
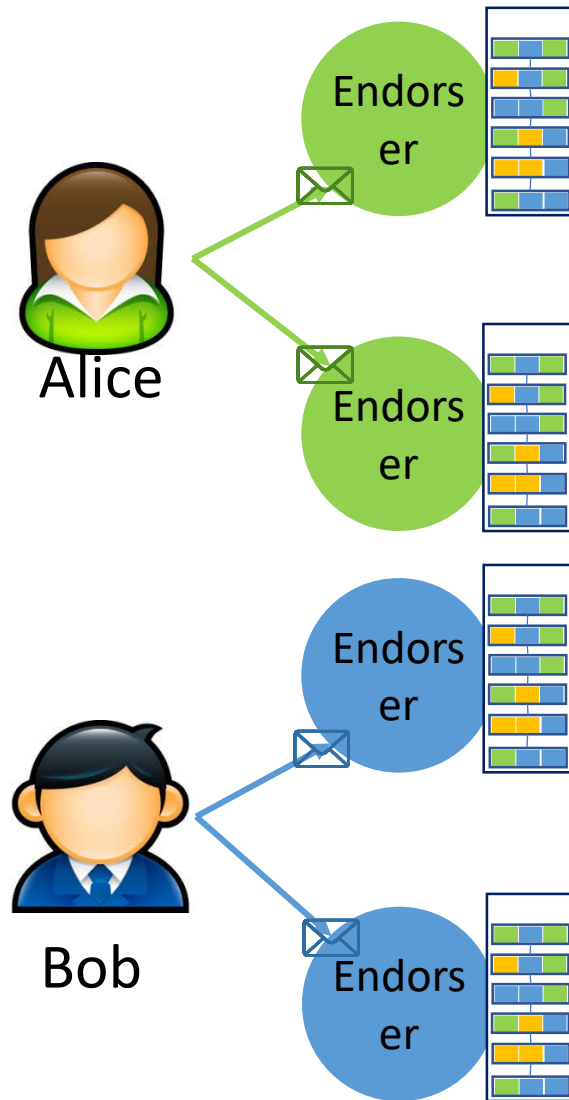
Hyperledger Fabric

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



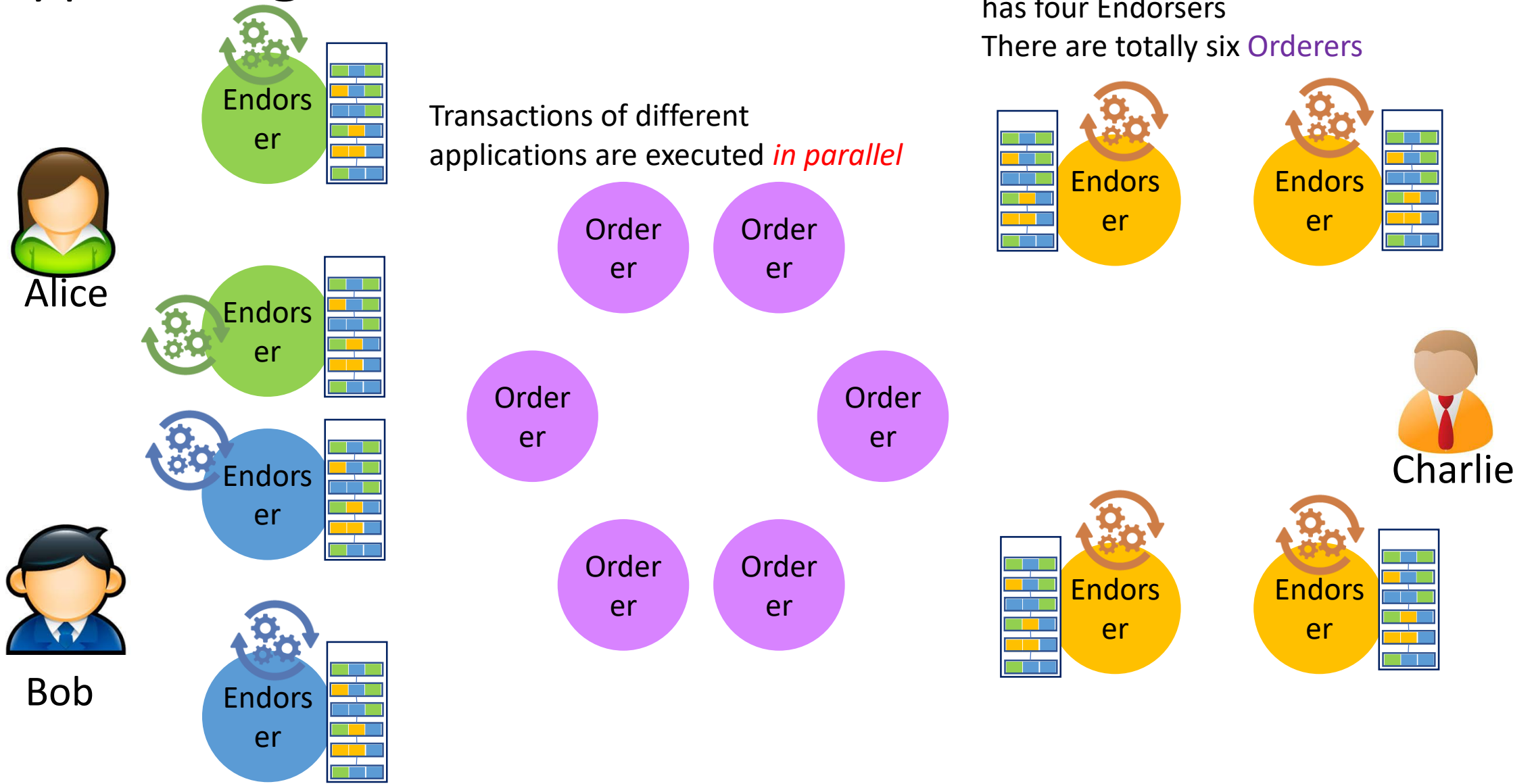
Hyperledger Fabric

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



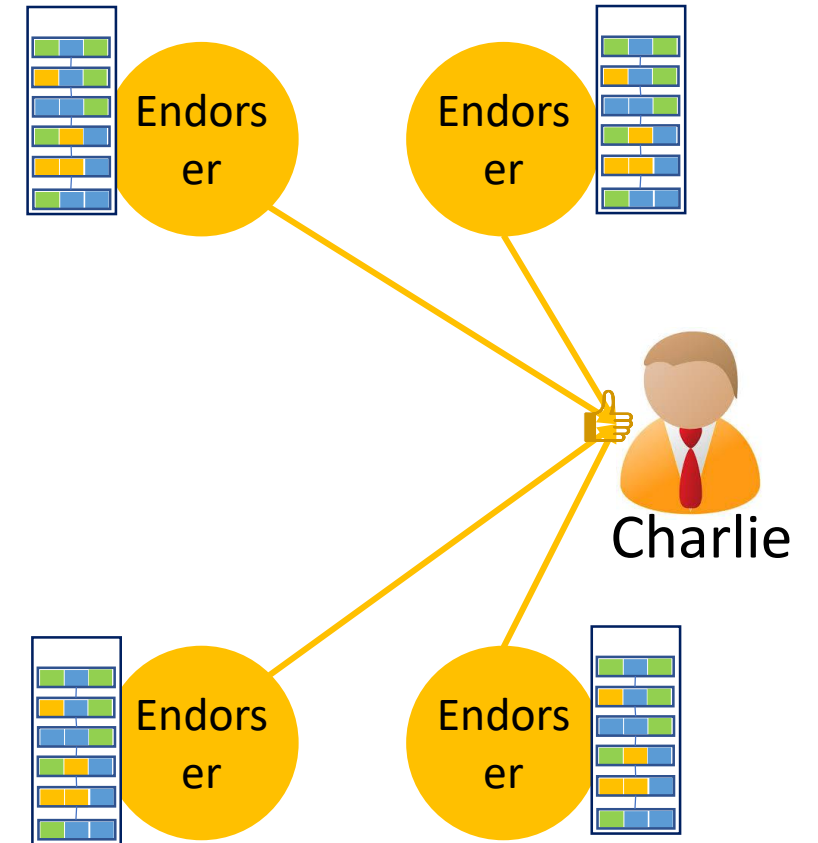
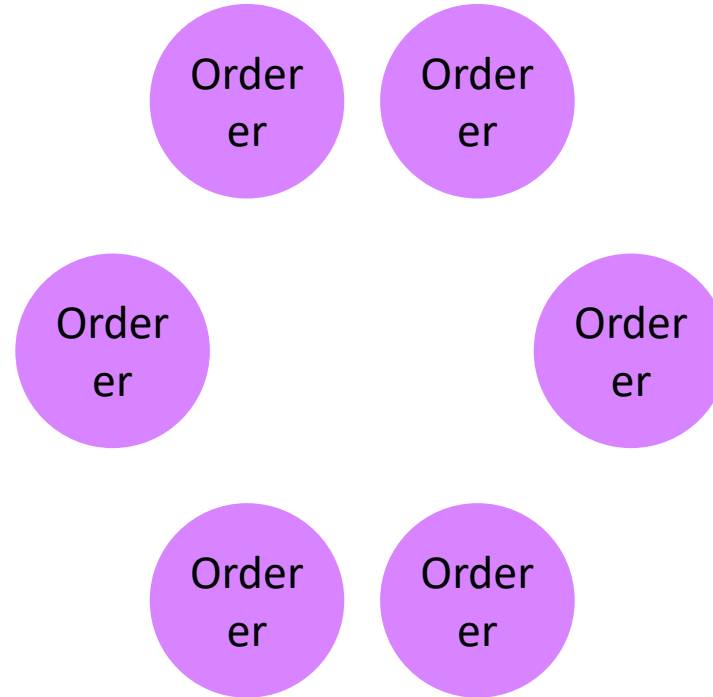
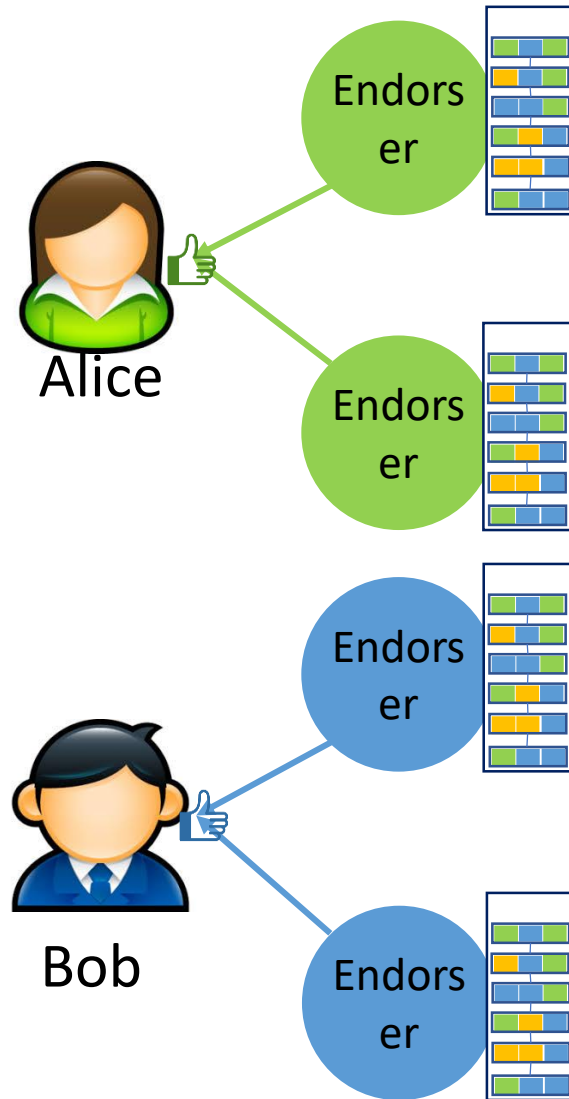
Hyperledger Fabric

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



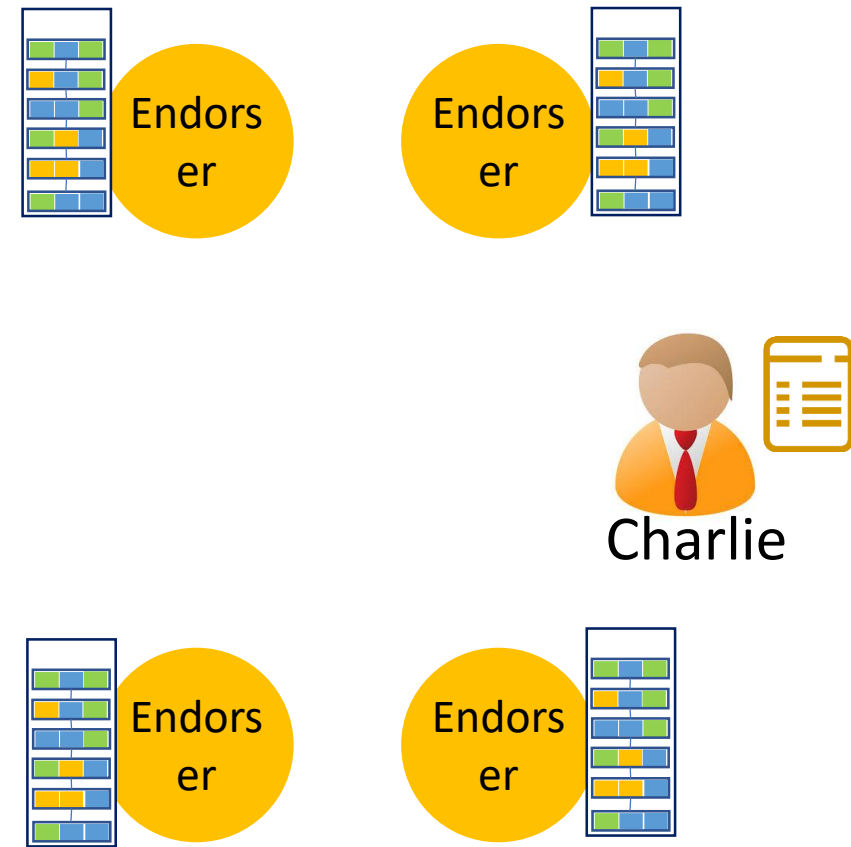
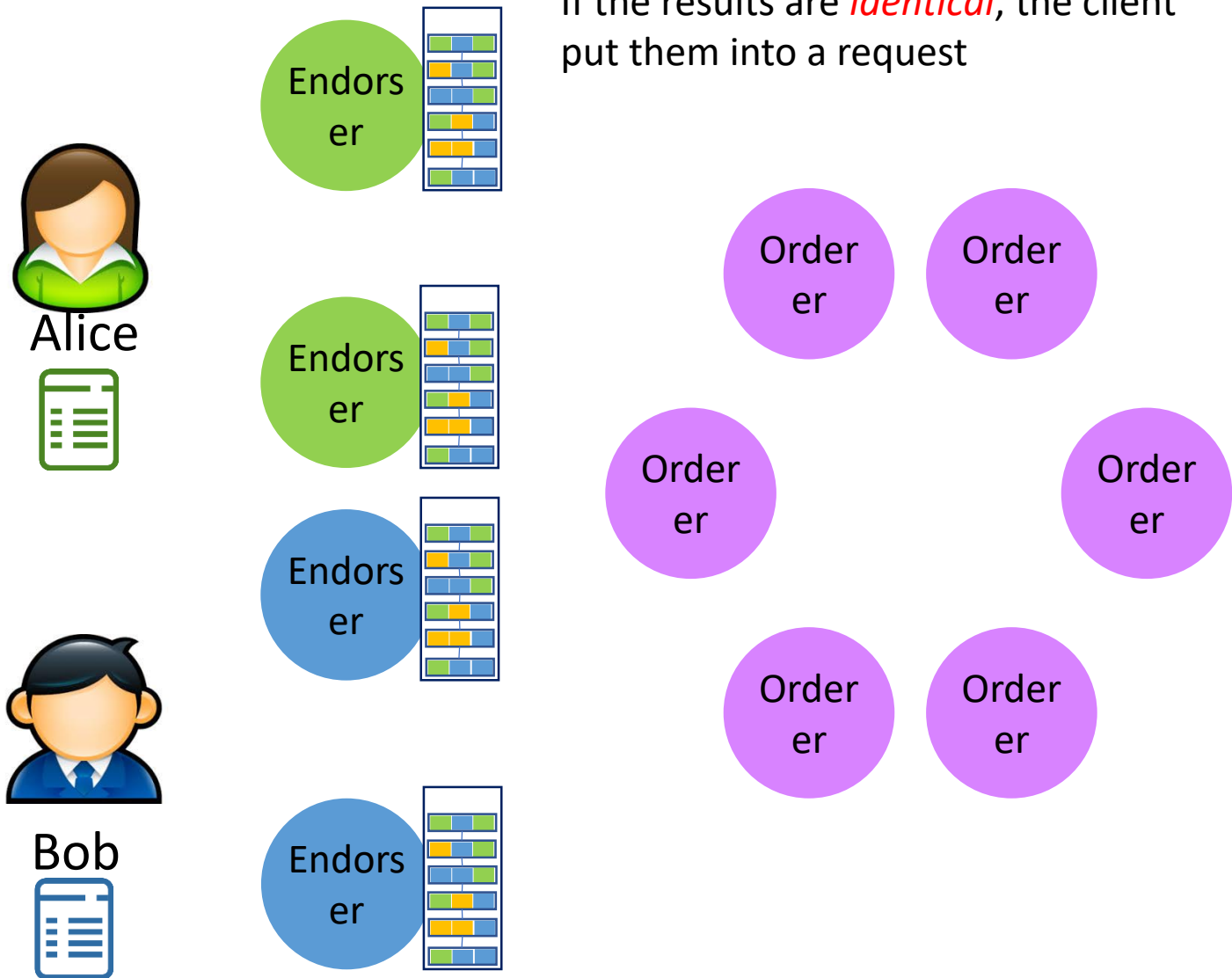
Hyperledger Fabric

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



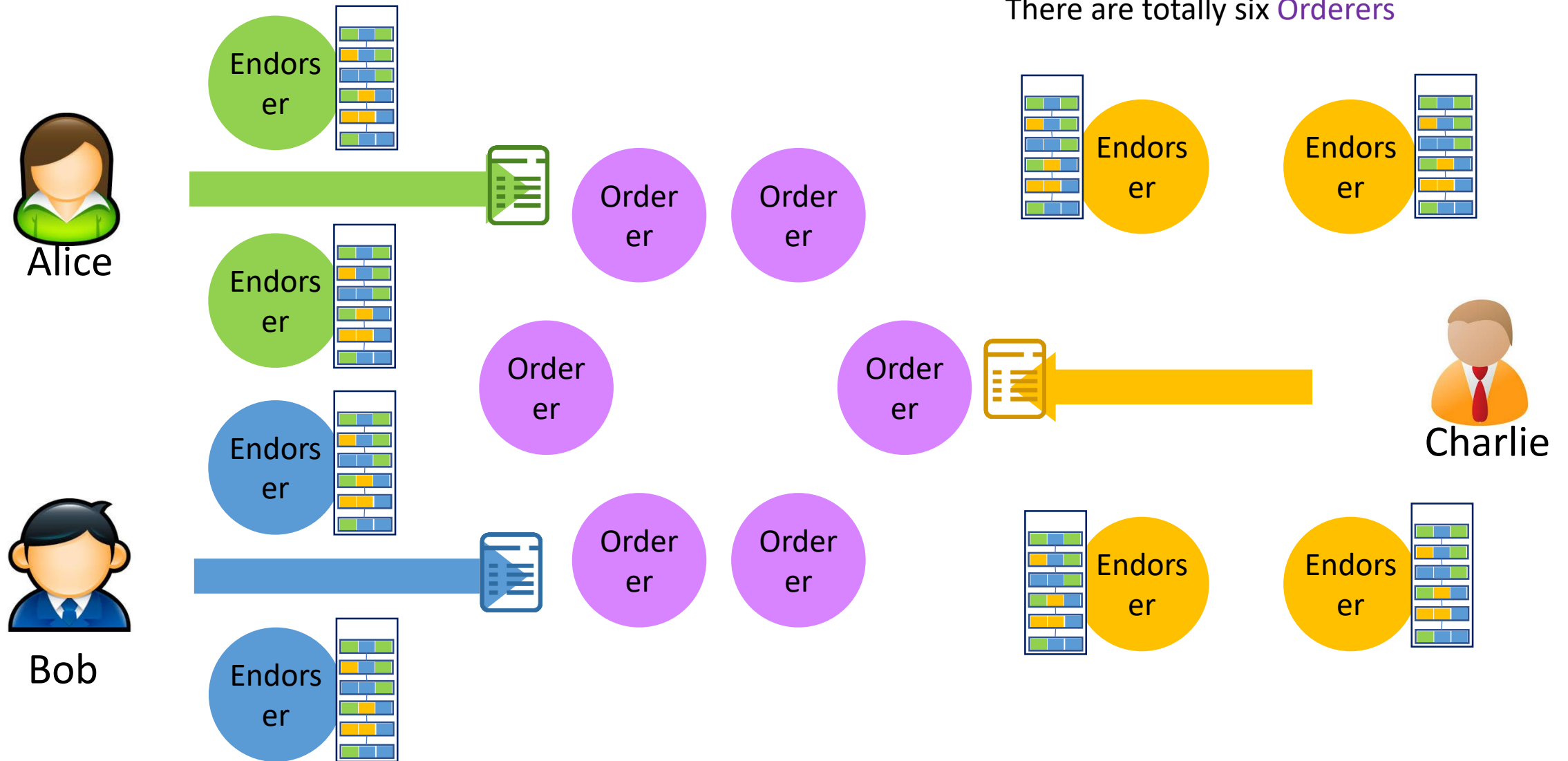
Hyperledger Fabric

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



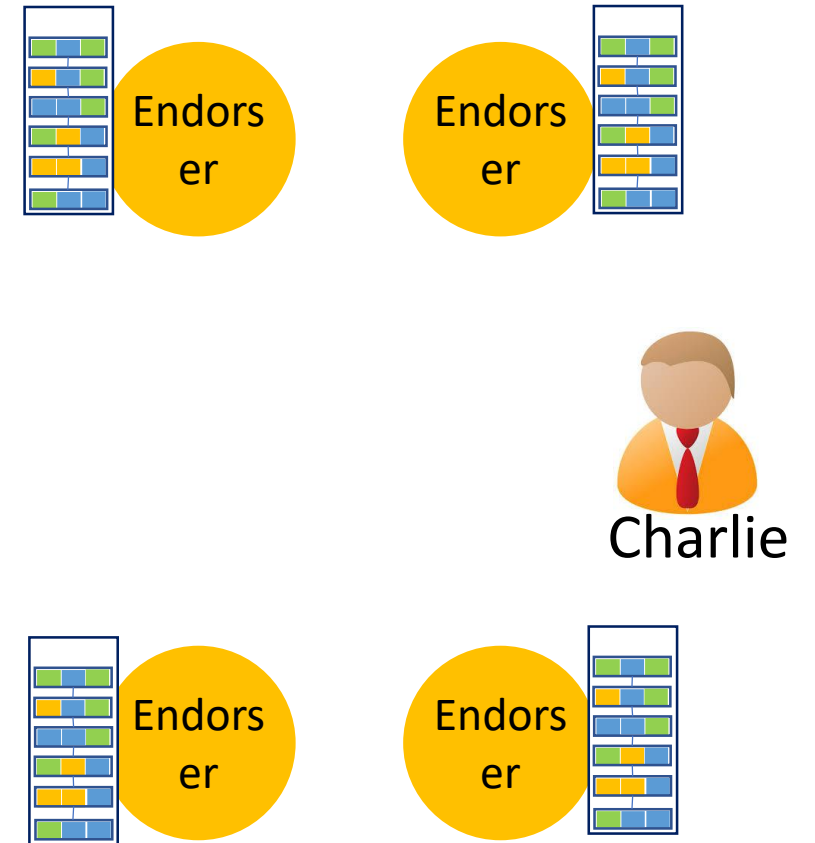
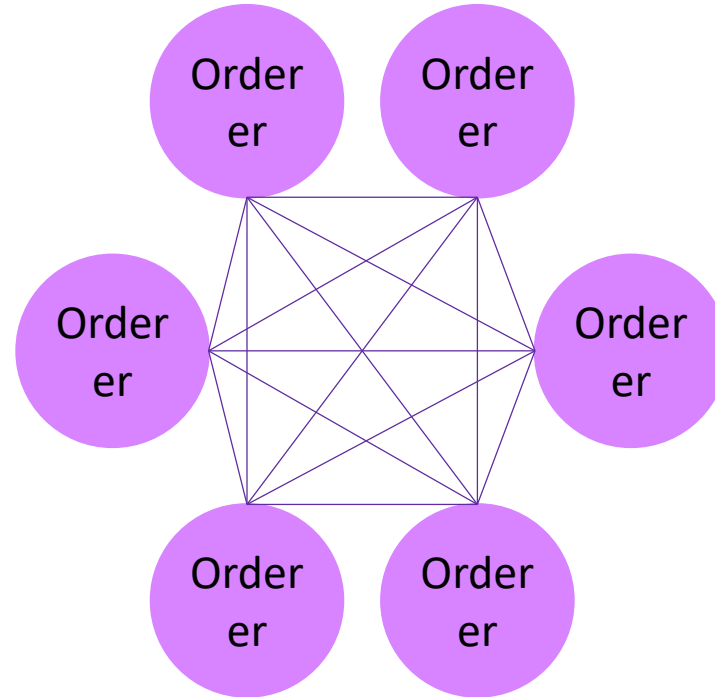
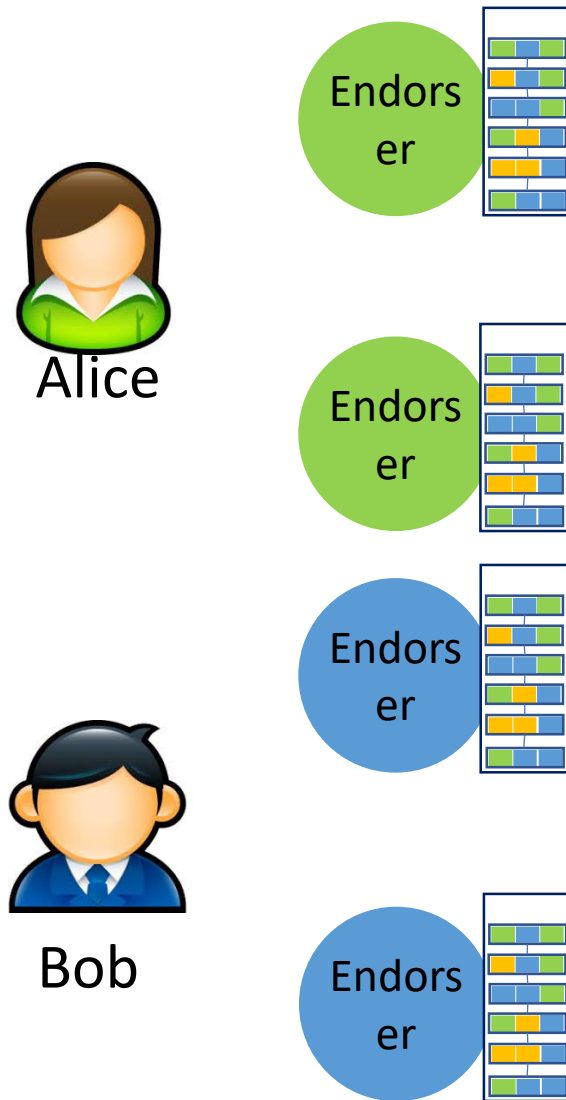
Hyperledger Fabric

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



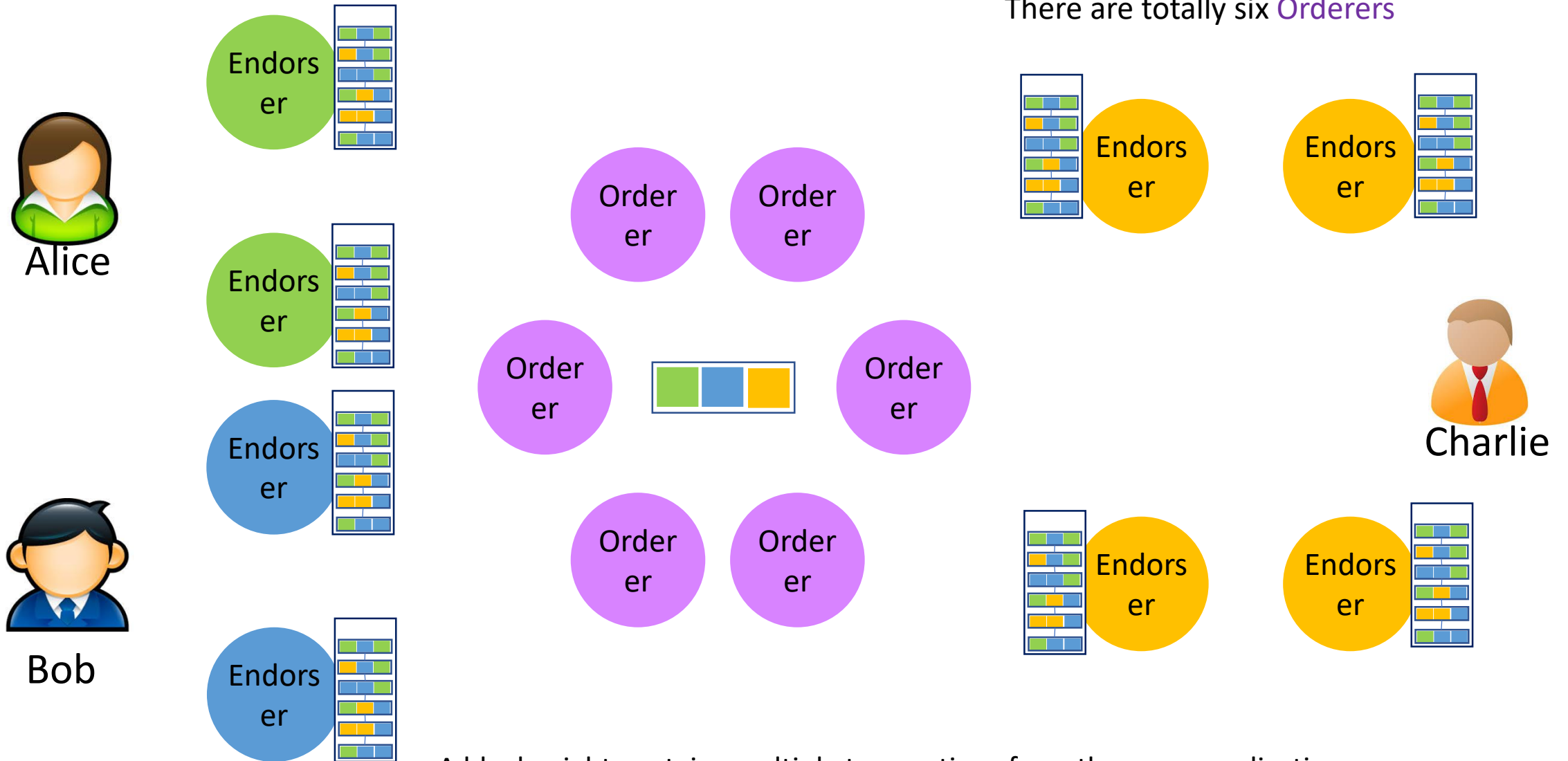
Hyperledger Fabric

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

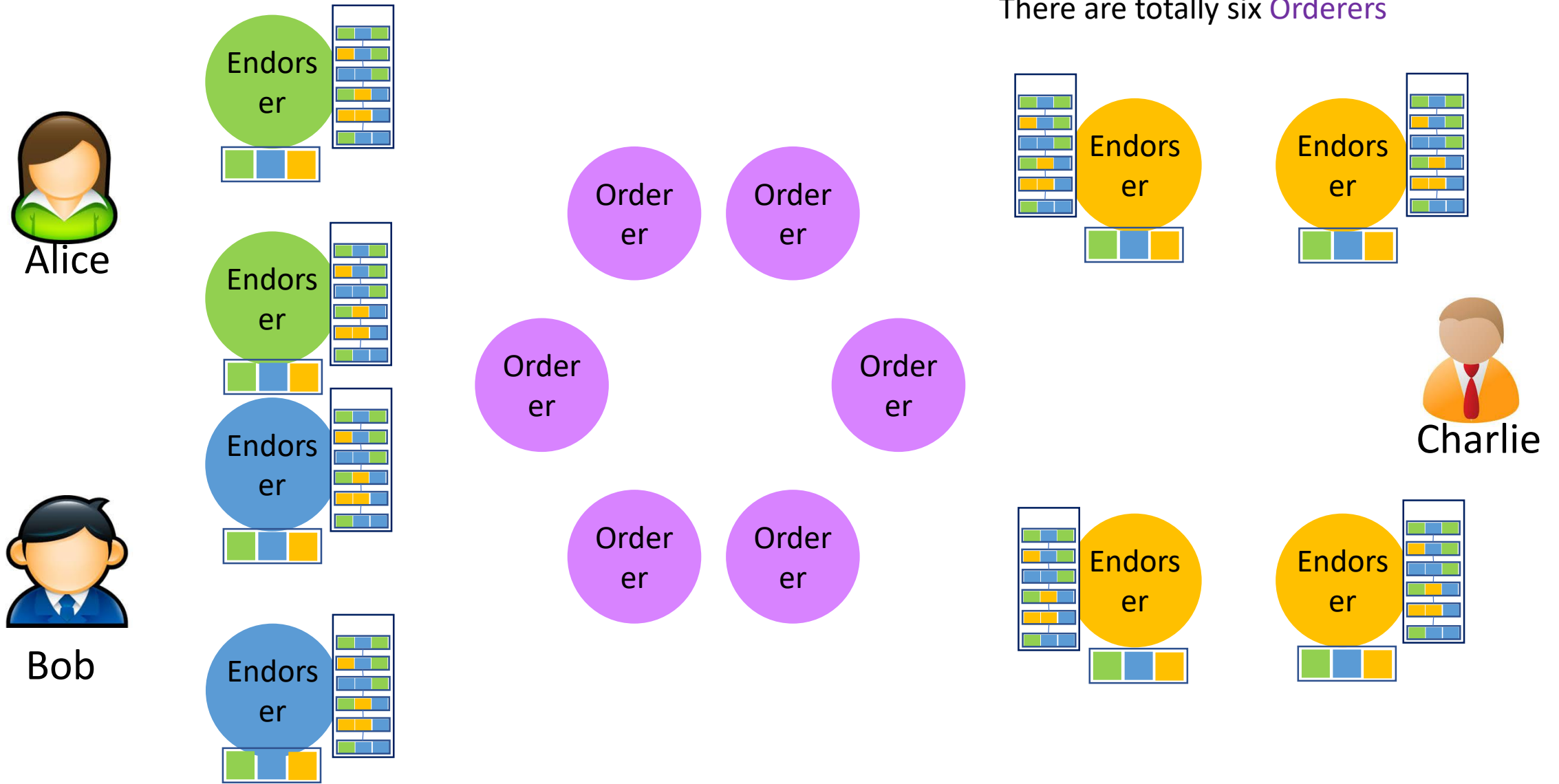
Three Applications (Green, Blue, Yellow)
Three Clients (Alice, Bob, Charlie)
Green and Blue have two Endorsers, Yellow has four Endorsers
There are totally six Orderers



A block might contains multiple transactions from the same application

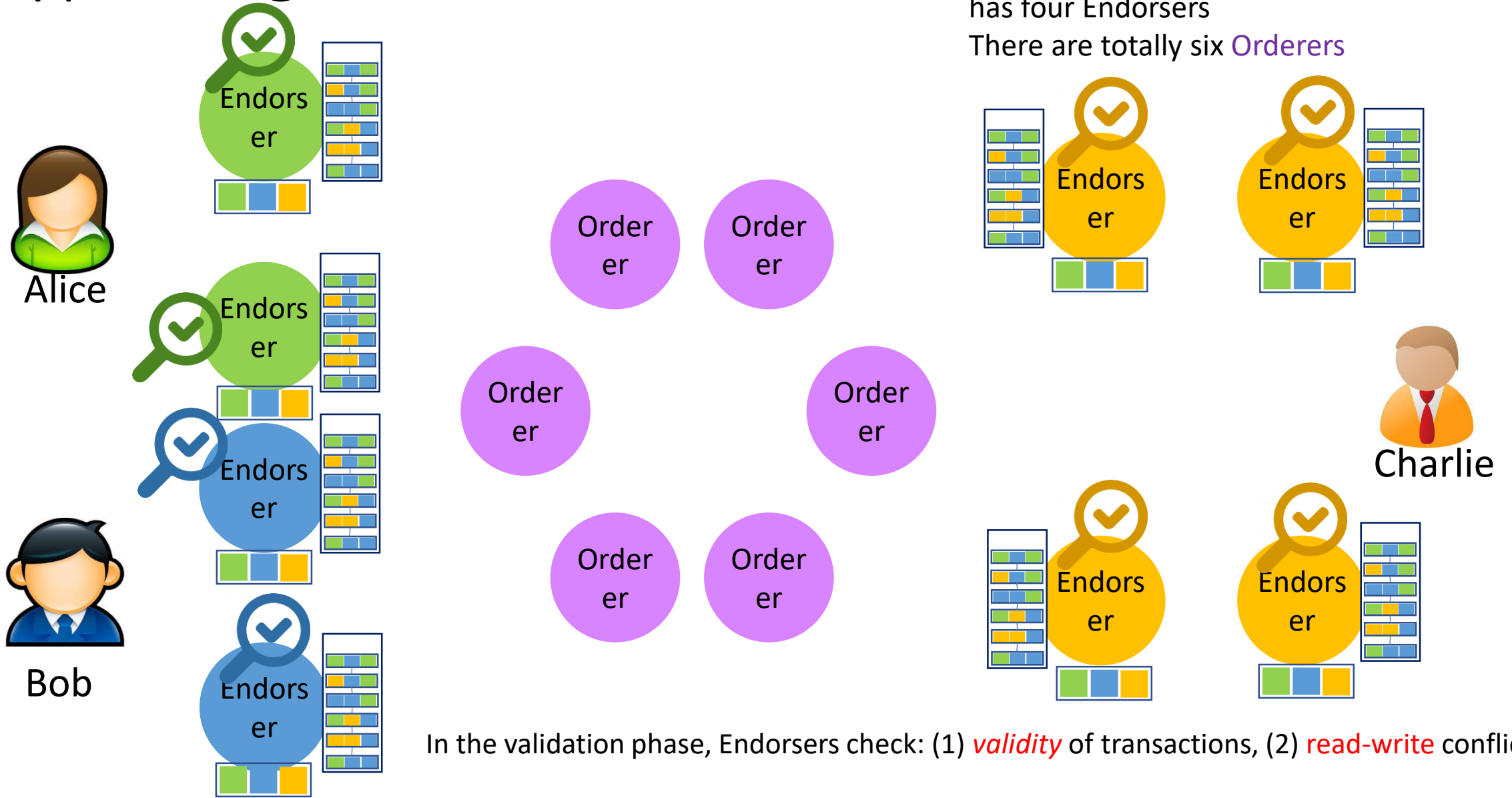
Hyperledger Fabric

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

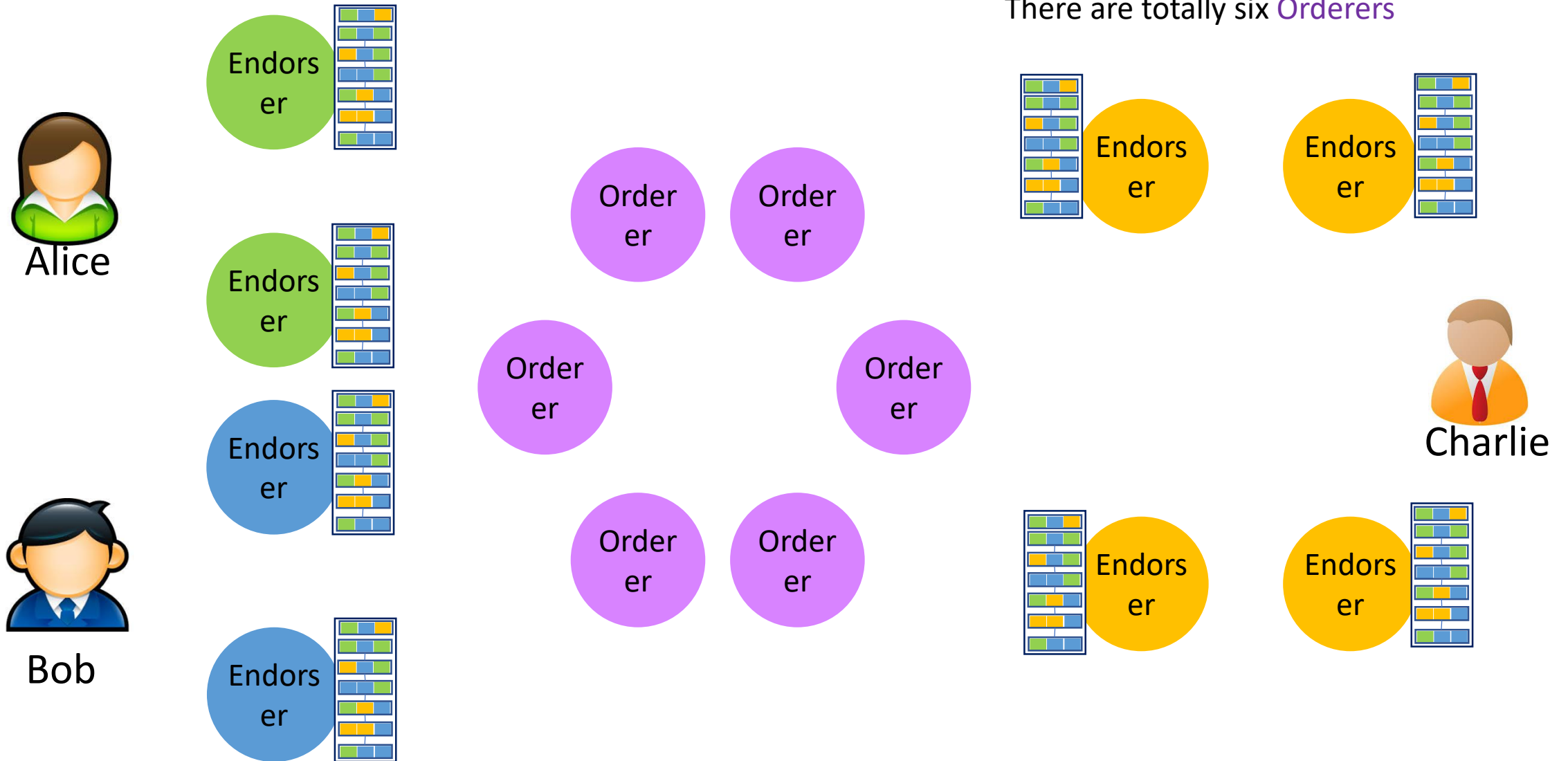
Three Applications (Green, Blue, Yellow)
Three Clients (Alice, Bob, Charlie)
Green and Blue have two Endorsers, Yellow has four Endorsers
There are totally six Orderers



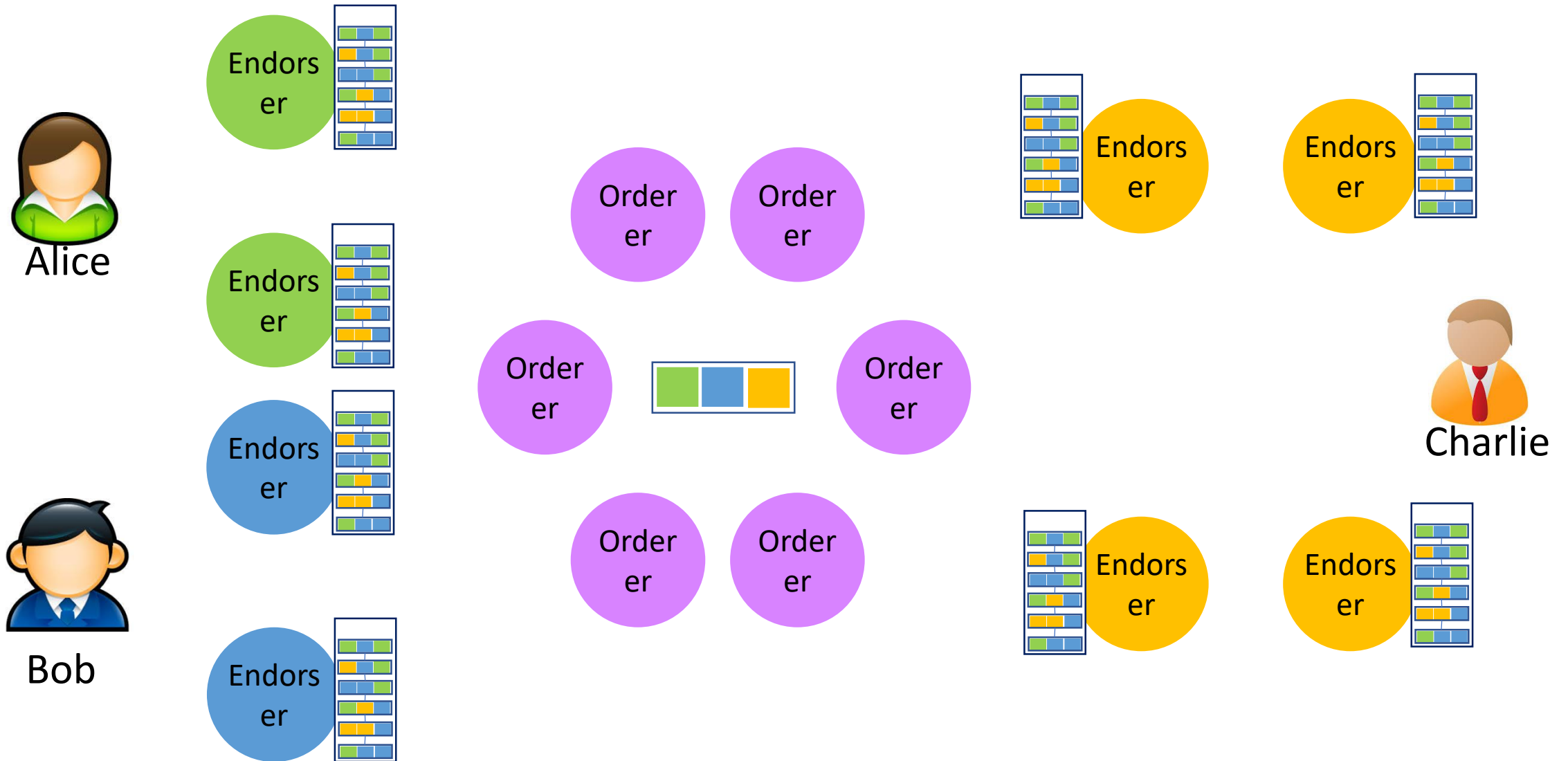
In the validation phase, Endorsers check: (1) *validity* of transactions, (2) *read-write* conflicts

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

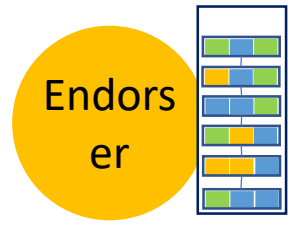
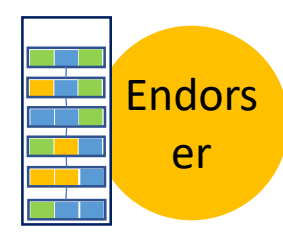
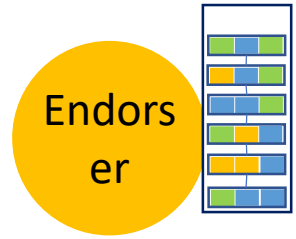
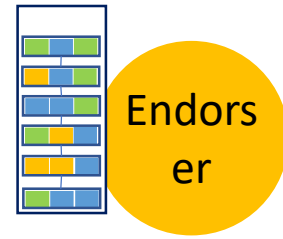
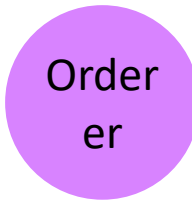
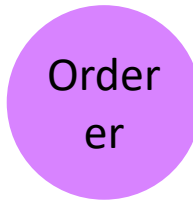
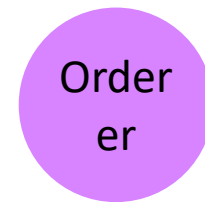
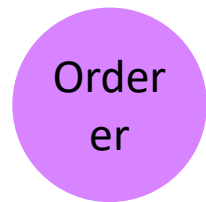
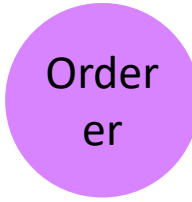
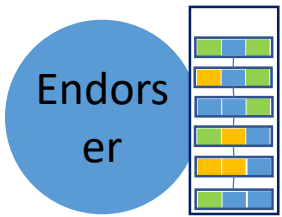
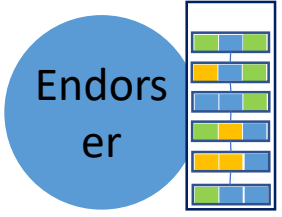
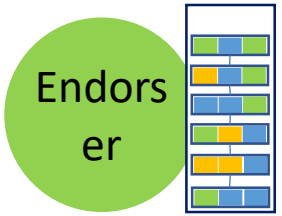
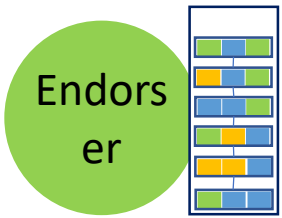


Hyperledger Fabric

-  Writes record A
-  Reads record A
-  Reads record A

What if transactions are **conflicting**?

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

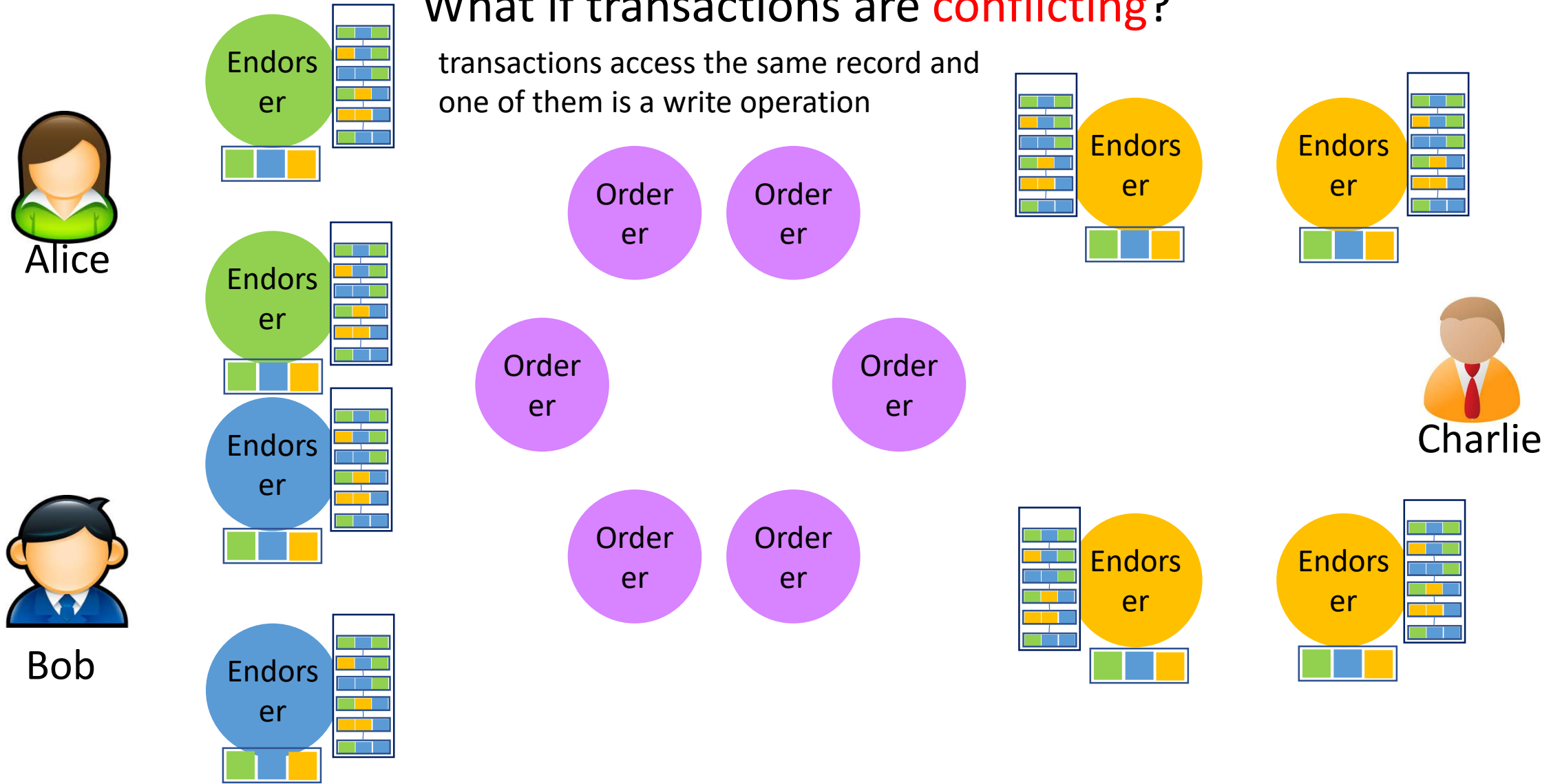


Hyperledger Fabric

- Writes record A
- Reads record A
- Reads record A

What if transactions are **conflicting**?

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

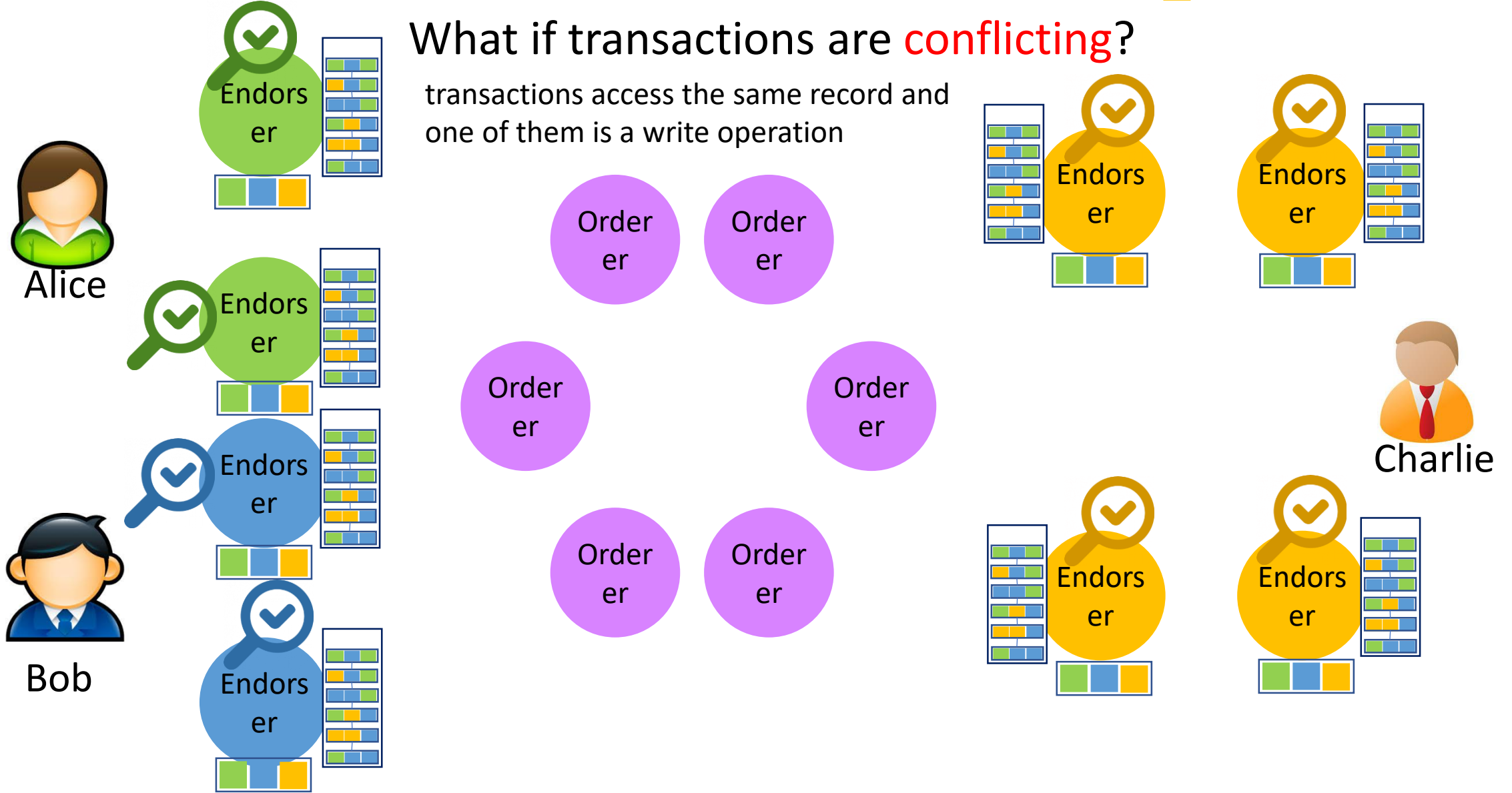


Hyperledger Fabric

- Writes record A
- Reads record A
- Reads record A

What if transactions are **conflicting**?

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

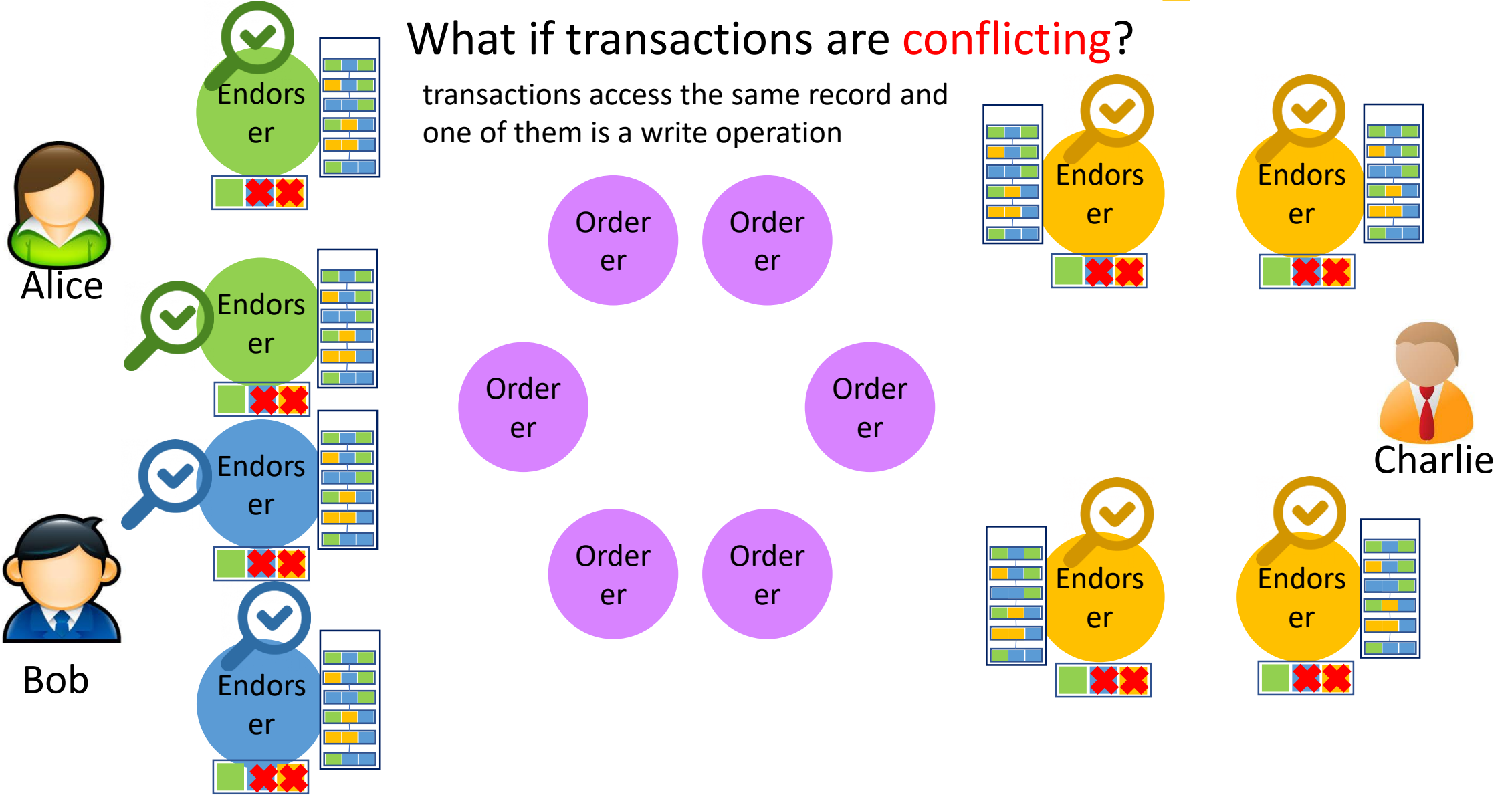


Hyperledger Fabric

- Writes record A
- Reads record A
- Reads record A

What if transactions are **conflicting**?

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



Hyperledger Fabric

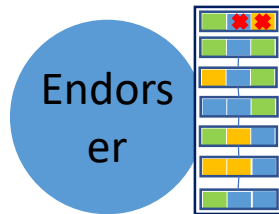
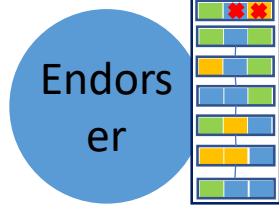
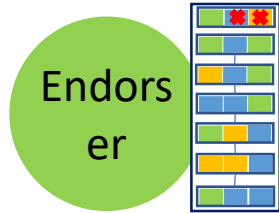
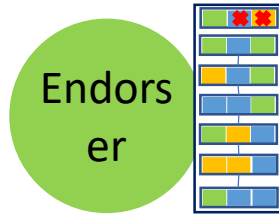
- Writes record A
- Reads record A
- Reads record A

What if transactions are **conflicting**?

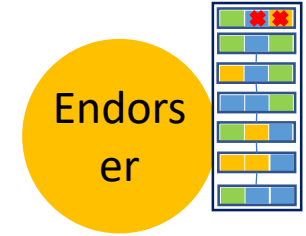
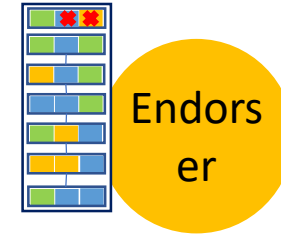
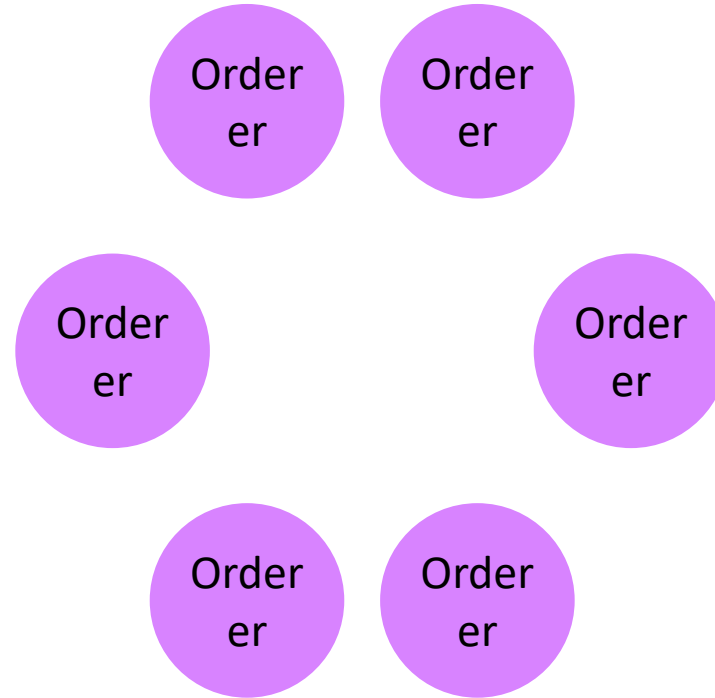
transactions access the same record and one of them is a write operation



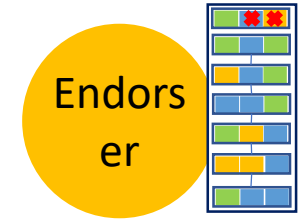
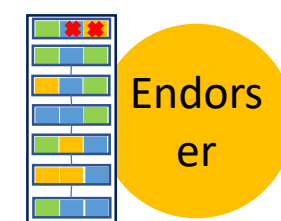
Alice



Bob



Charlie



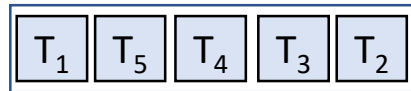
Dependency Graph

- 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}

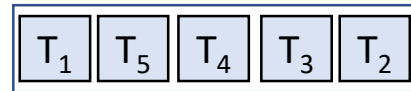
T_4 Read = {b}
Write = {c}

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}

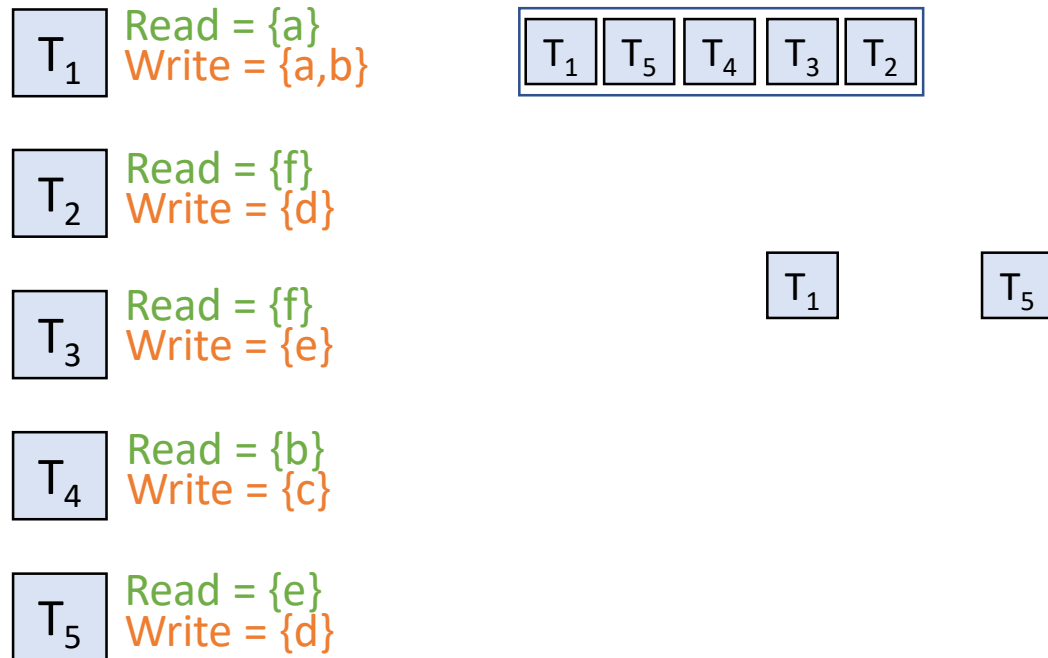
T_1

T_4 Read = {b}
Write = {c}

T_5 Read = {e}
Write = {d}

Dependency Graph

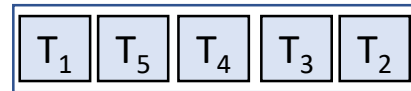
- A dependency graph exposes conflicts between transactions to give a partial order of transactions.



Dependency Graph

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

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

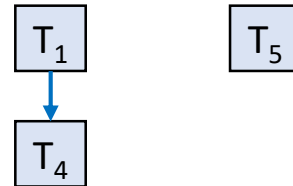


T_2 Read = {f}
Write = {d}

T_3 Read = {f}
Write = {e}

T_4 Read = {b}
Write = {c}

T_5 Read = {e}
Write = {d}

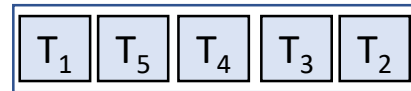


T_4 reads b that is written by T_1

Dependency Graph

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

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

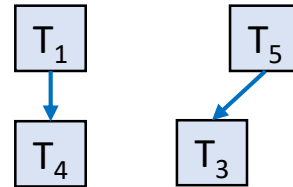


T_2 Read = {f}
Write = {d}

T_3 Read = {f}
Write = {e}

T_4 Read = {b}
Write = {c}

T_5 Read = {e}
Write = {d}



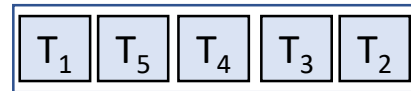
T_4 reads b that is written by T_1

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 Read = {a}
Write = {a,b}

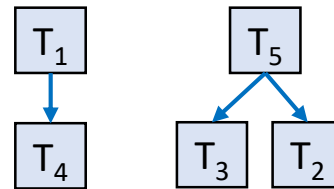


T_2 Read = {f}
Write = {d}

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

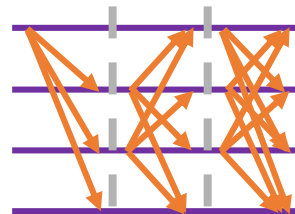
T_3 writes e that is read by T_5

T_2 writes d that is written by T_5

Order-Parallel Execute (OXII) Architecture

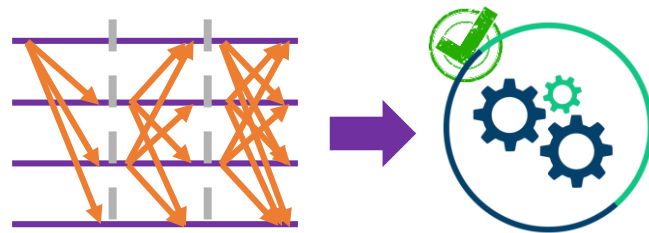
Order-Parallel Execute (OXII) Architecture

- 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

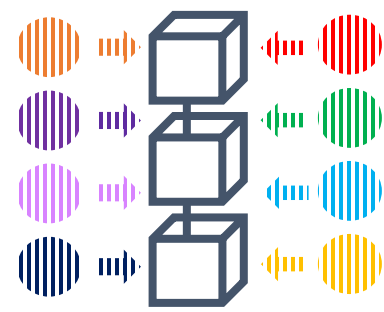


Order-Parallel Execute (OXII) Architecture

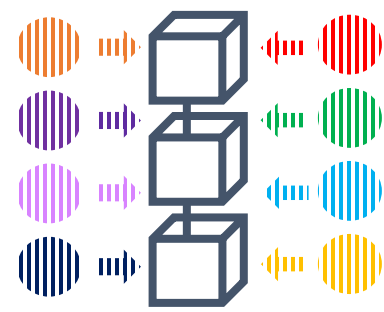
- A separate set of nodes (orderers) **orders** the transactions, puts them into blocks, generates a **dependency graph** for the block, and multicasts it to all the nodes.
- Each transaction (of an application) is then **validated** and **executed** by a subset of nodes (executors of the application) following the dependency graph
- The nodes multicast the results of execution and append the block



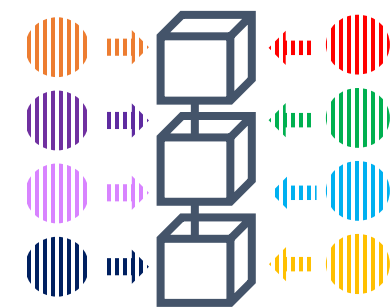
ParBlockchain



ParBlockchain



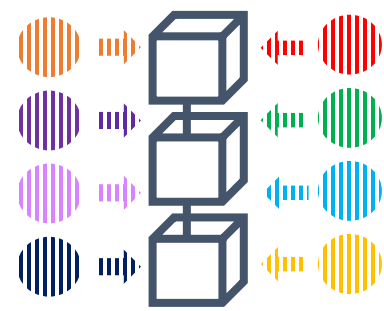
Order-Execute Architecture: Transactions are first ordered, and then executed



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

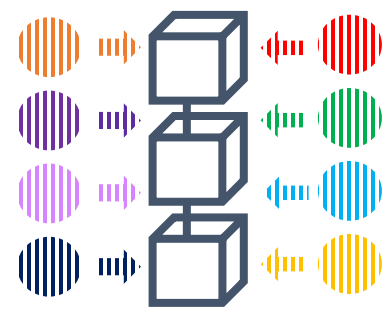


ParBlockchain

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



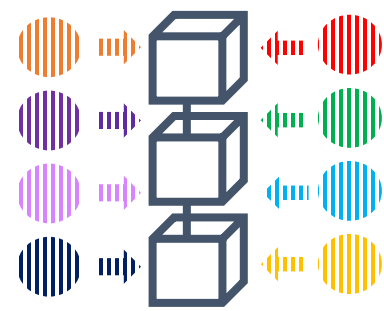
ParBlockchain

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

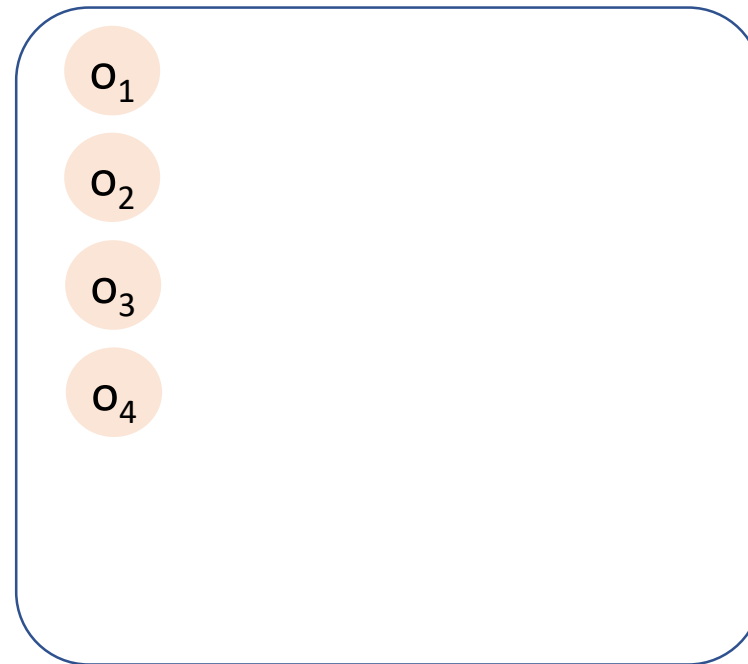
Non-deterministic Execution: inconsistent execution results can be detected in the last phase (results in decreasing the performance)

ParBlockchain

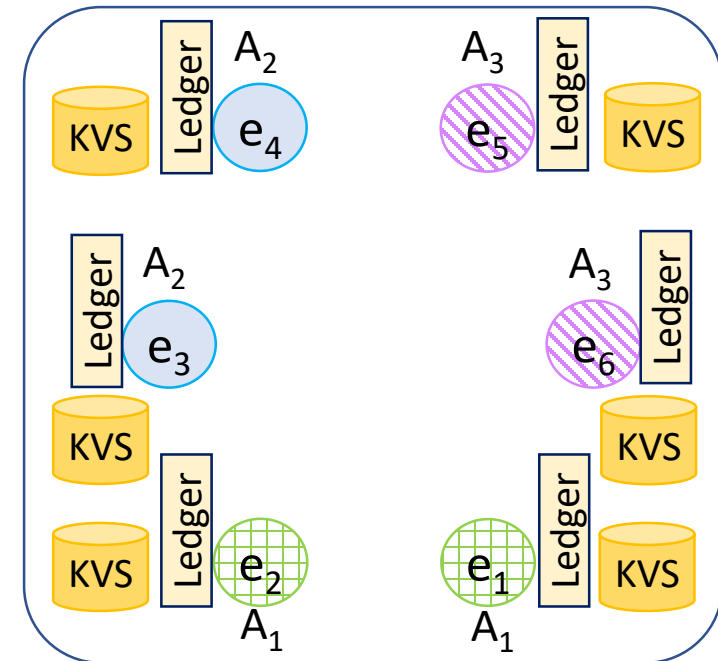
Clients



Orderes



Executors

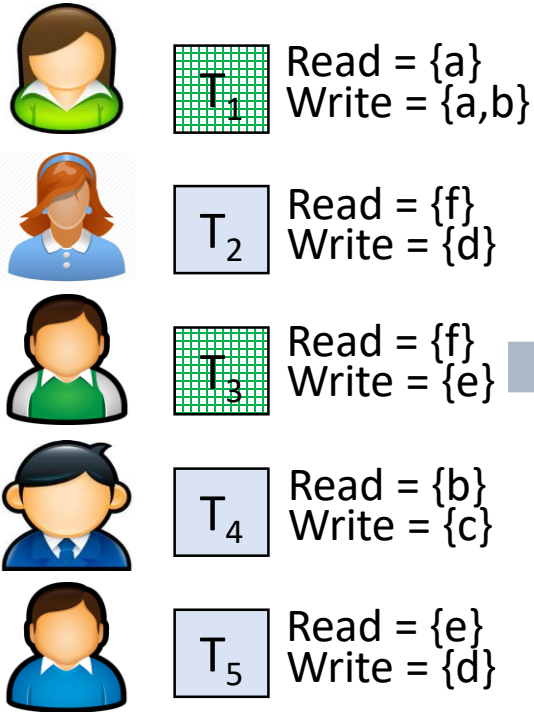


Application A₁ Application A₂ Application A₃

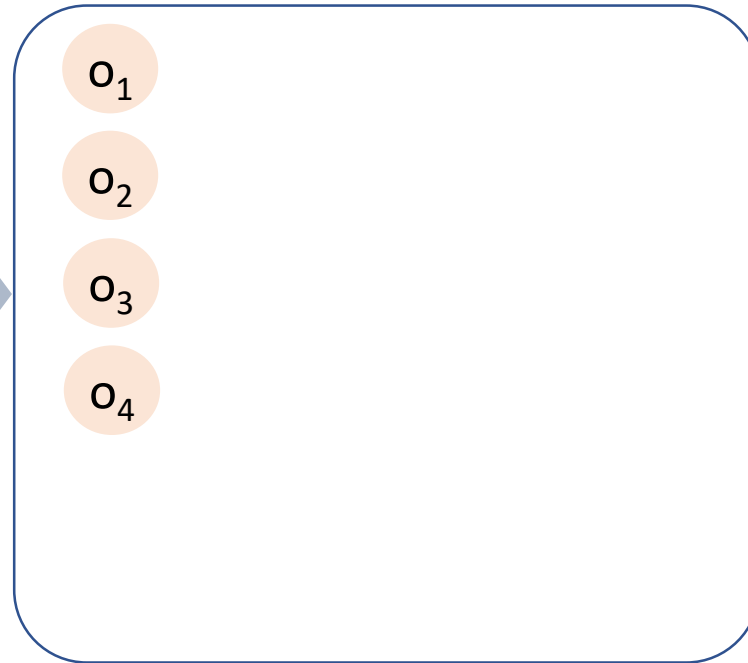
Each application has a set of Executors
Each Executor stores a copy of ledger and Data

ParBlockchain

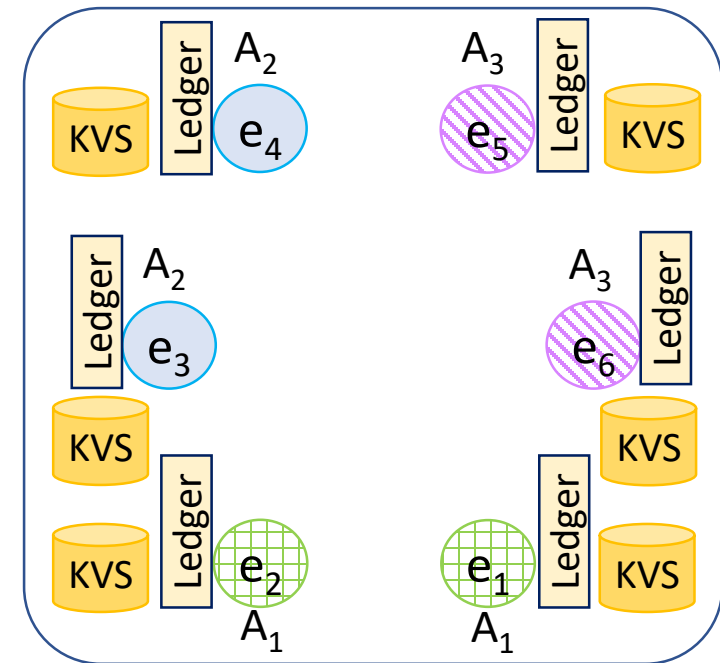
Clients



Orders



Executors

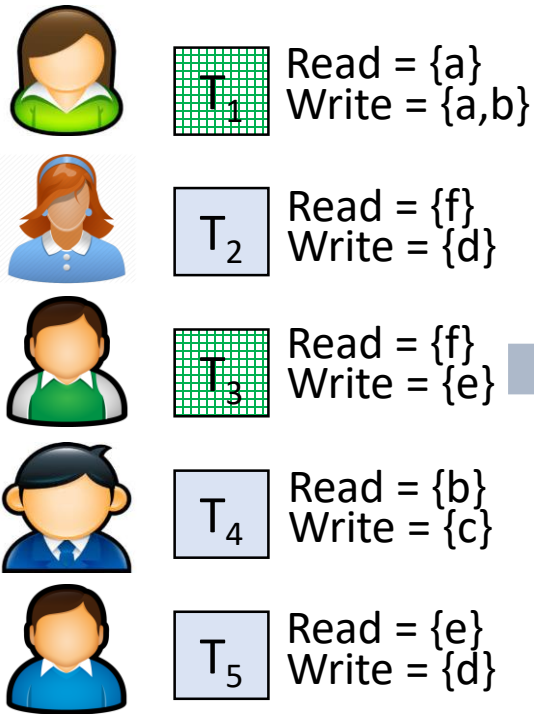


Application A_1
 Application A_2
 Application A_3

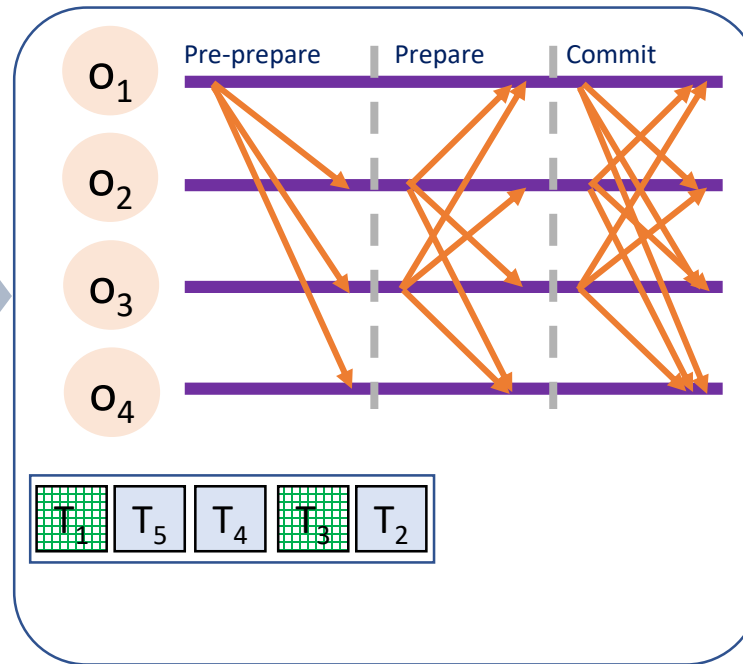
Each transaction of an application include records to be read and written

ParBlockchain

Clients

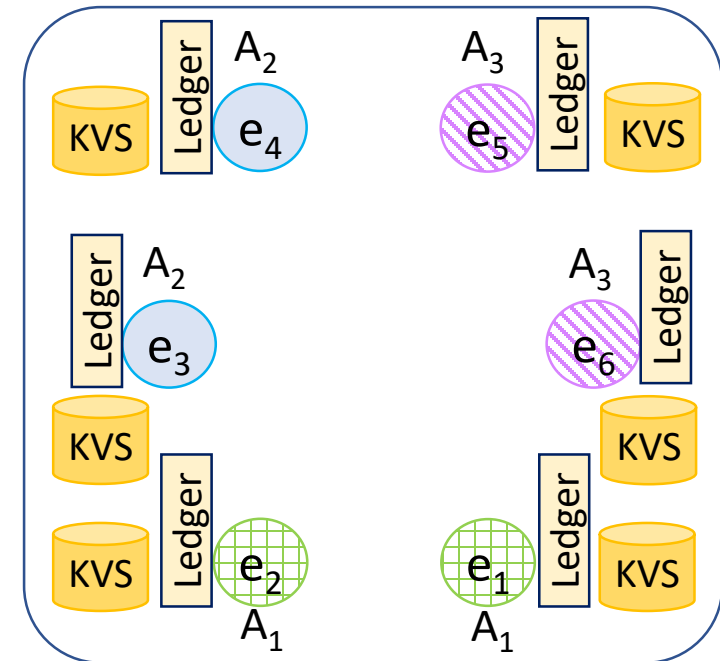


Orderers



The orderers order transactions using a consensus protocol (e.g. PBFT)

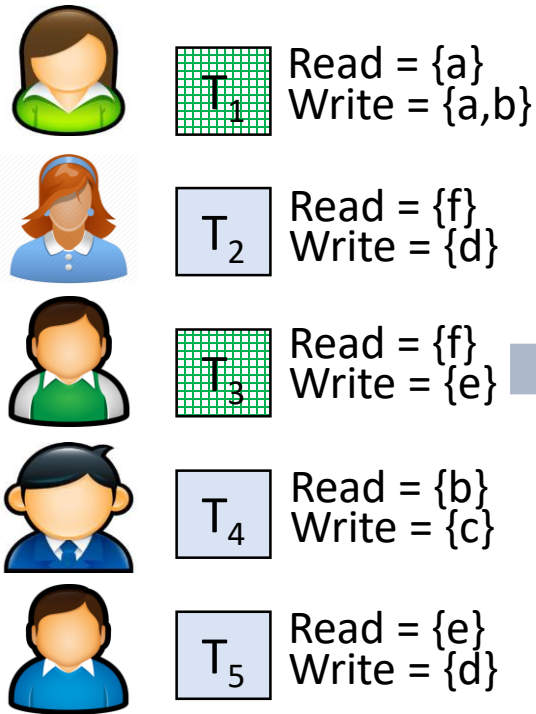
Executors



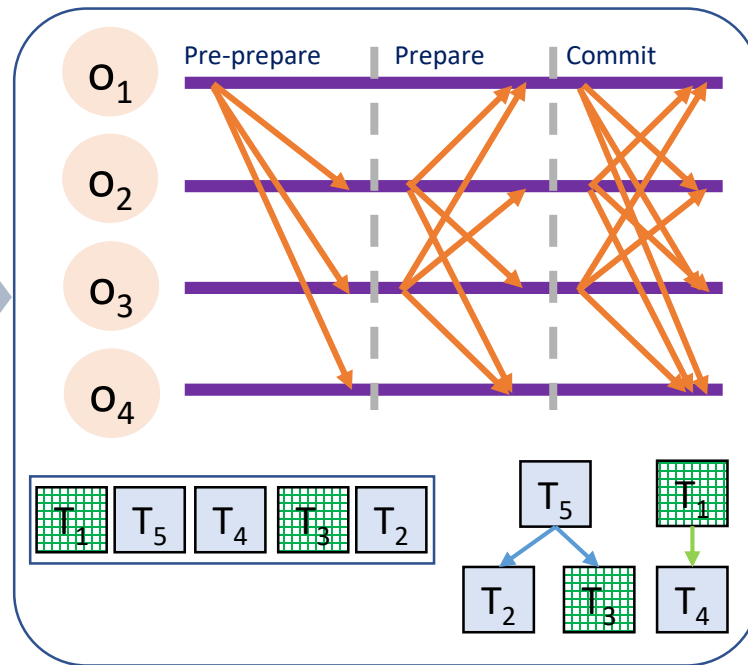
Application A₁ Application A₂ Application A₃

ParBlockchain

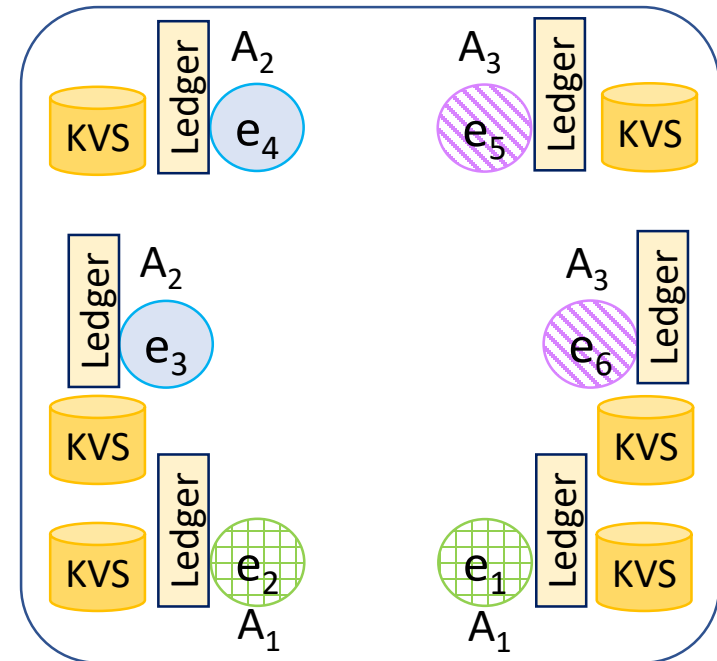
Clients



Orderers



Executors

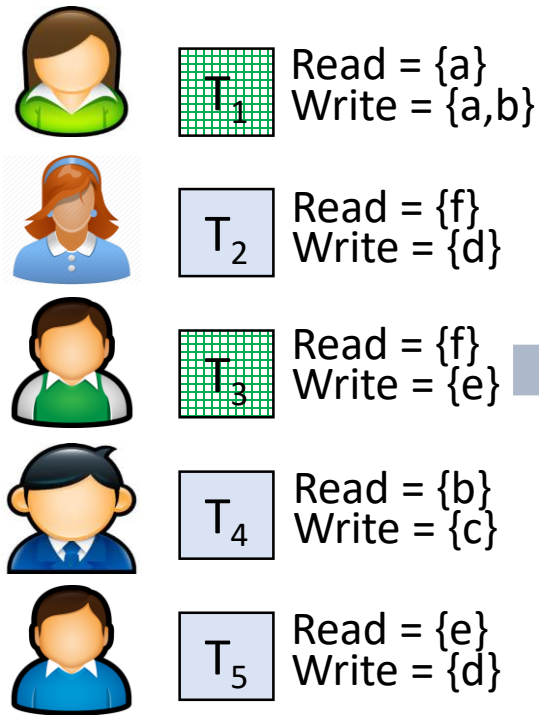


Application A_1
 Application A_2
 Application A_3

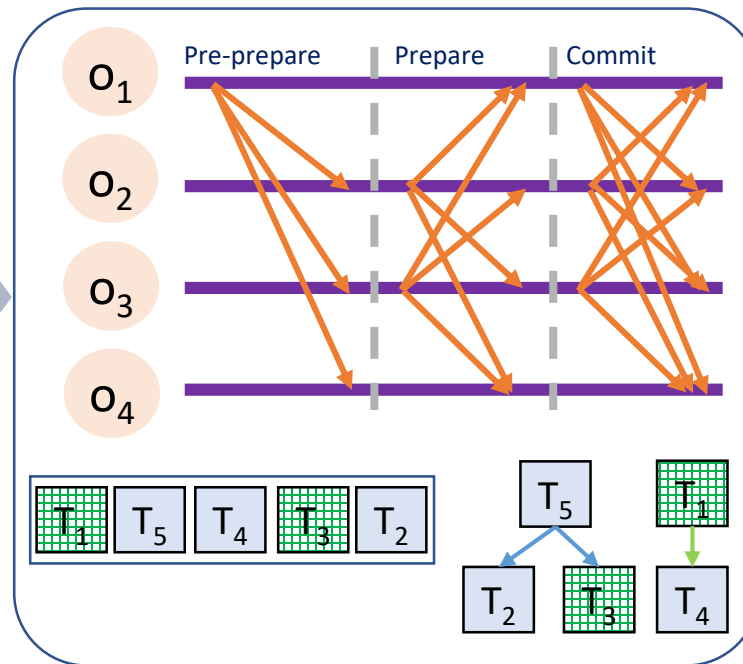
Each orderer generates a dependency graph for the block and multicasts it to all Executors

ParBlockchain

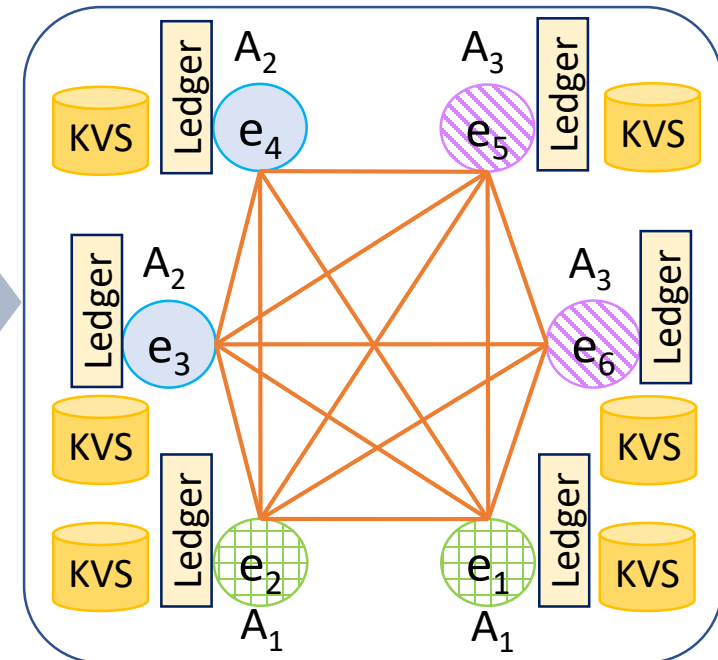
Clients



Orderes



Executors



Application A_1
 Application A_2
 Application 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

Supporting High Contention Workloads

Hyperledger

Executes first (does not submit transactions with inconsistent results)

Validates read-write conflicts last (aborts conflicting transactions)

Optimistic vs. Pessimistic Execution

Two ways to look at the problem!

Supporting non-deterministic execution

Supporting High Contention Workloads

Hyperledger

Executes first (does not submit transactions with inconsistent results)

Validates read-write conflicts last (aborts conflicting transactions)

ParBlockchain

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

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

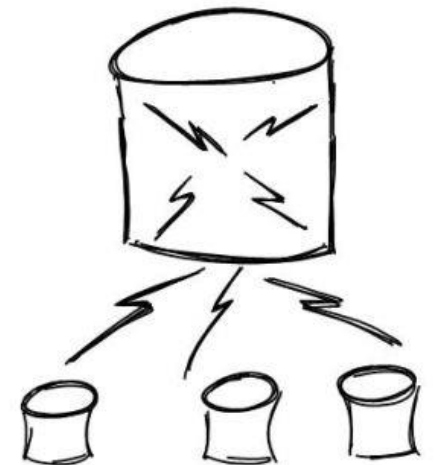
Blockchain Scalability

- *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

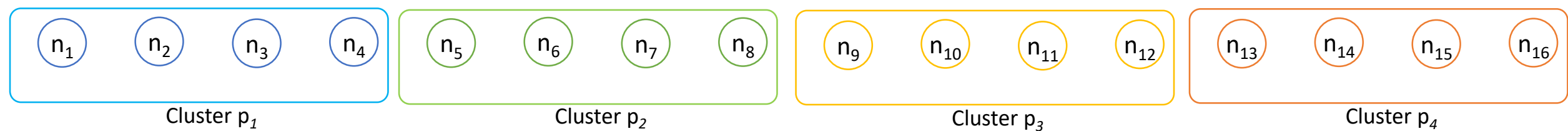
Sharding (as a horizontal technique): Partitioning the data into multiple shards that are maintained by different subsets of nodes



Sharding Blockchains

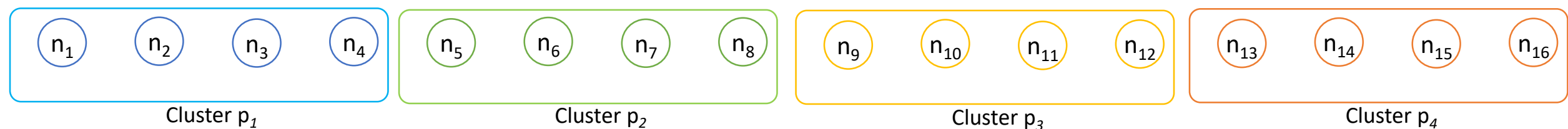
Sharding Blockchains

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



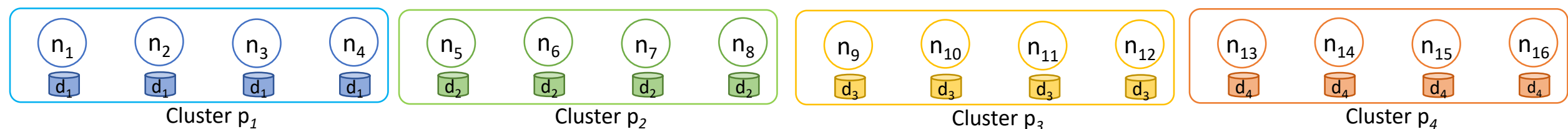
Sharding Blockchains

- 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)



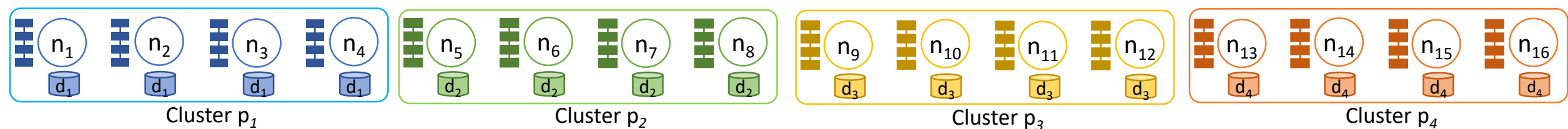
Sharding Blockchains

- 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**



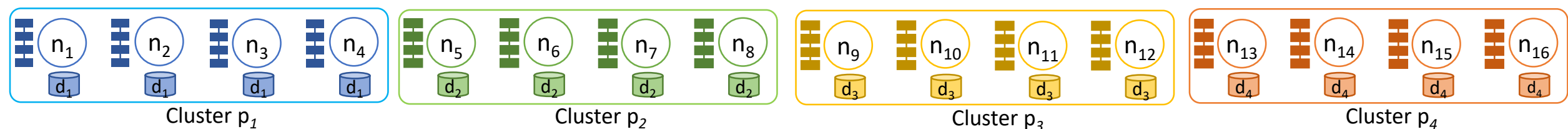
Sharding Blockchains

- 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)**

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

SharPer Ledger

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

λ

λ

λ

λ

λ

P_1

P_2

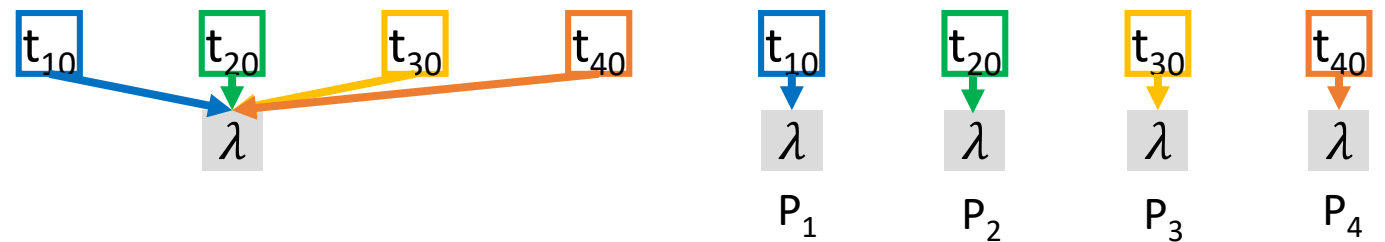
P_3

P_4

SharPer Ledger

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

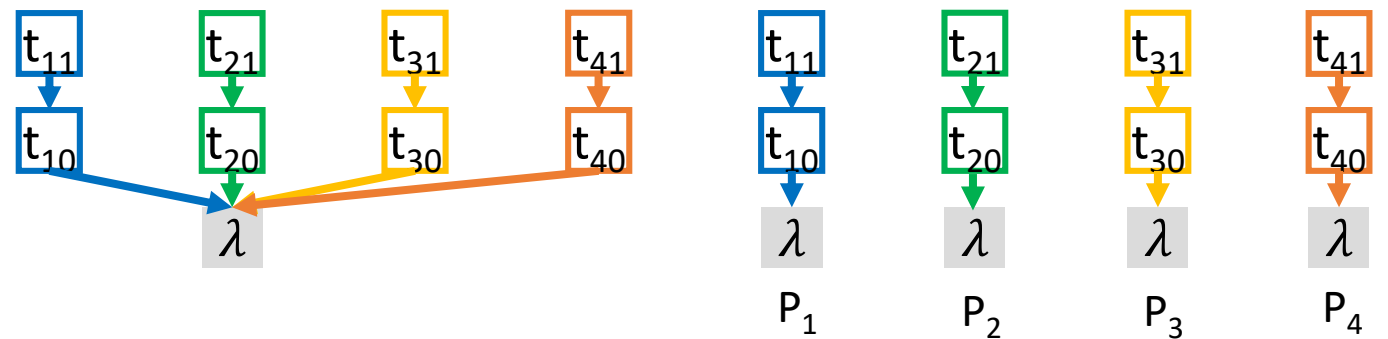
- Intra-shard transactions of different clusters are processed in parallel



SharPer Ledger

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

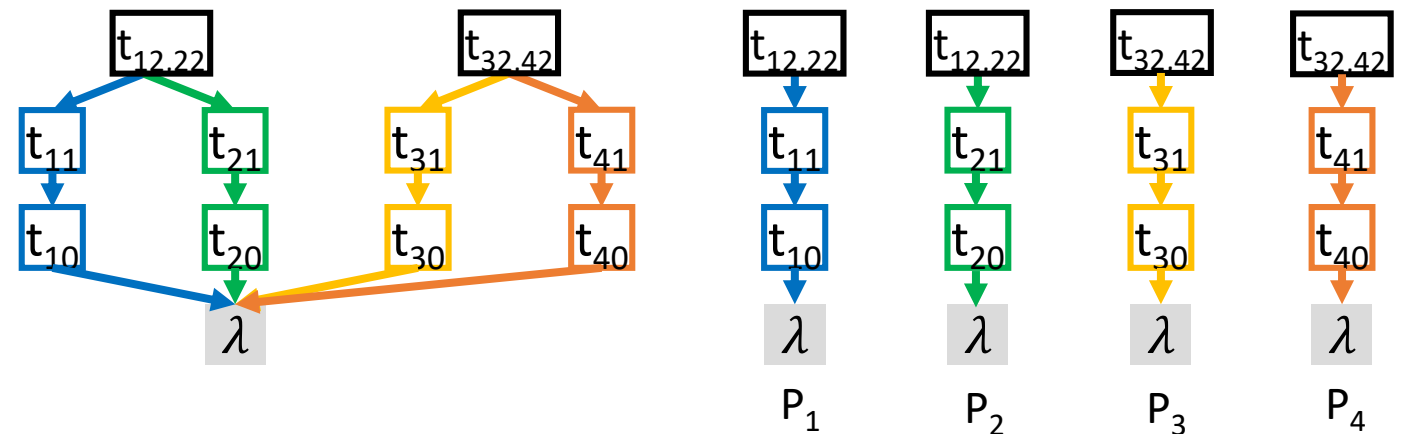
- Intra-shard transactions of different clusters are processed in parallel



SharPer Ledger

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

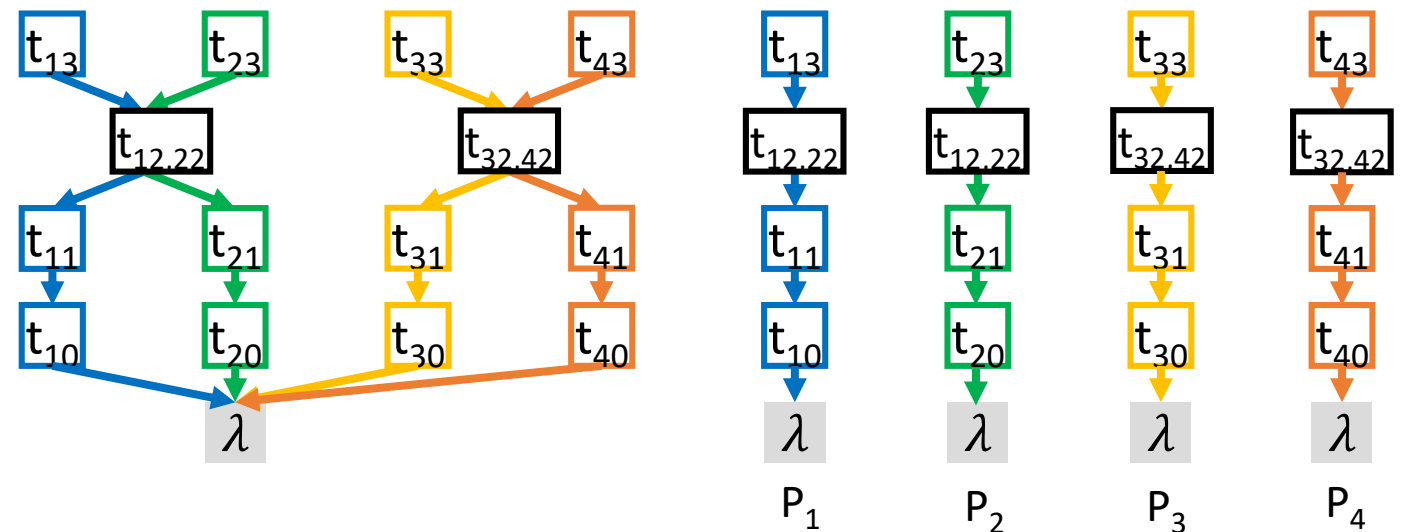
- Intra-shard transactions of different clusters are processed in parallel
- Cross-shard transactions with **non-overlapping clusters** are processed in parallel
- A cross-shard transaction includes multiple hash pointers



SharPer Ledger

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

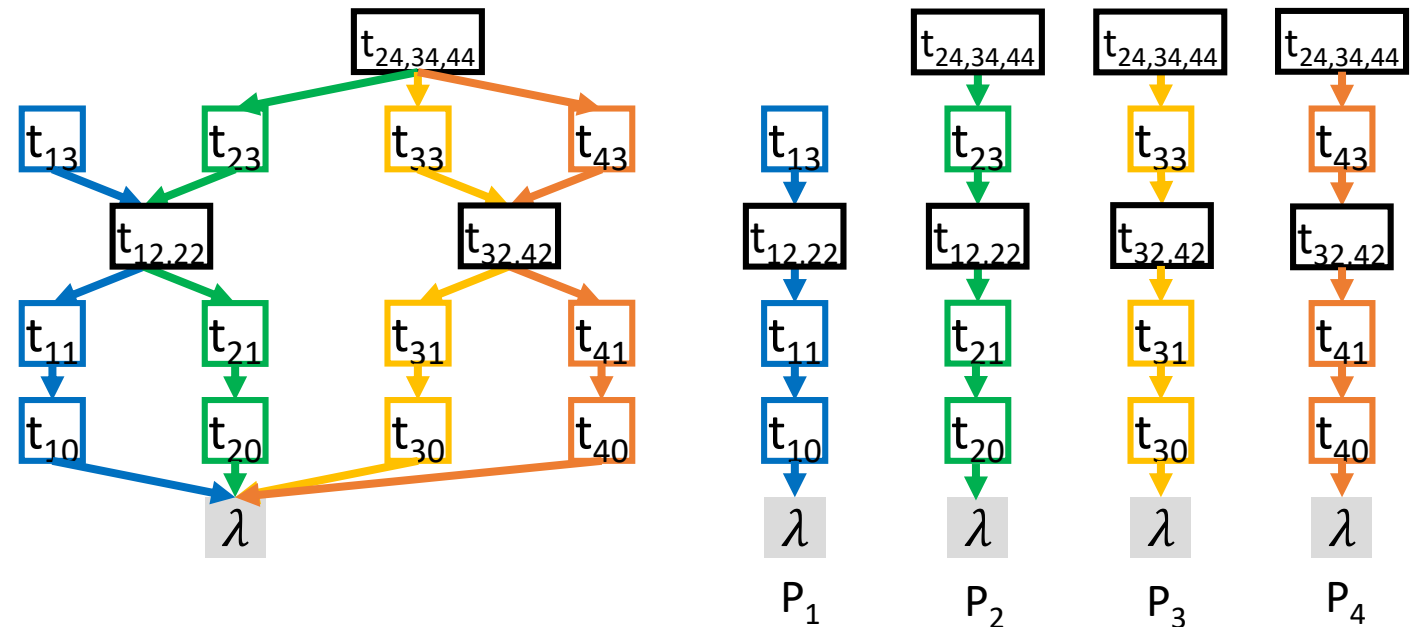
- Intra-shard transactions of different clusters are processed in parallel
- Cross-shard transactions with **non-overlapping clusters** are processed in parallel
- A cross-shard transaction includes multiple hash pointers



SharPer Ledger

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

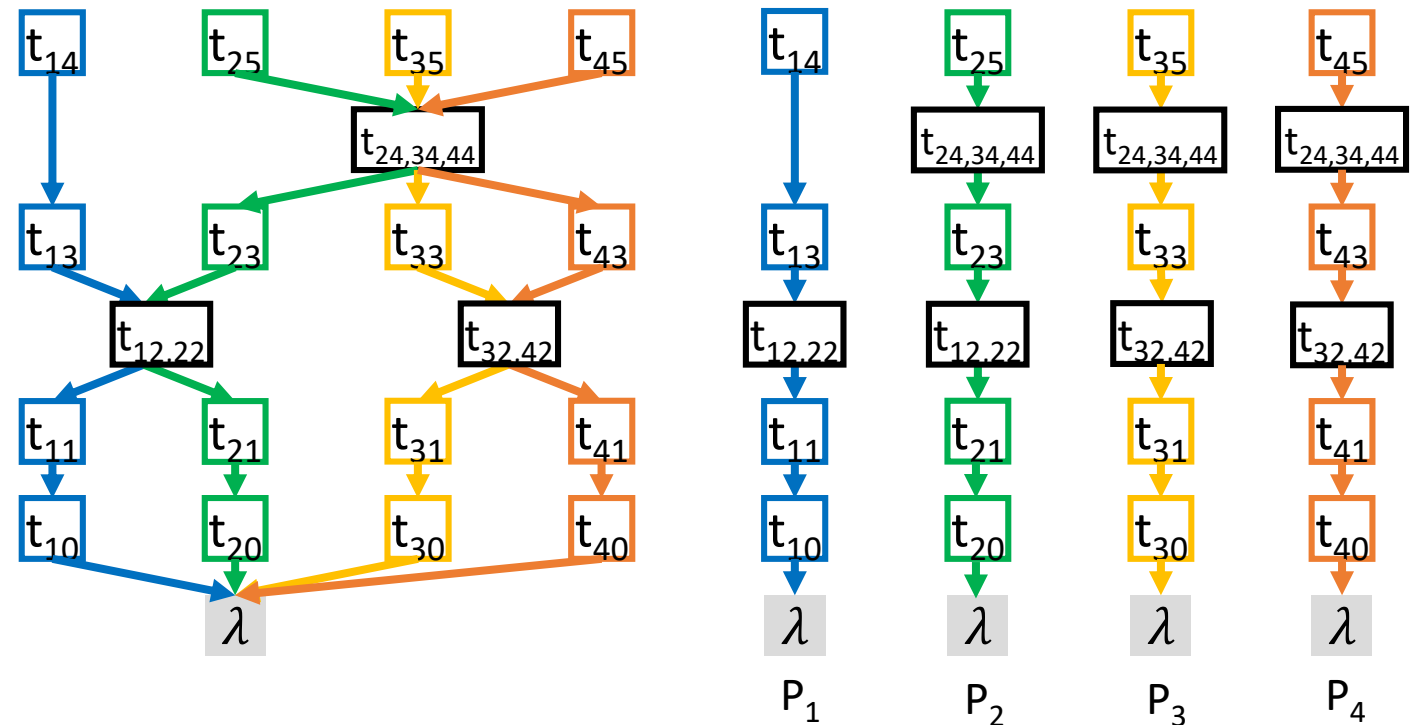
- Intra-shard transactions of different clusters are processed in parallel
- Cross-shard transactions with **non-overlapping clusters** are processed in parallel
- A cross-shard transaction includes multiple hash pointers



SharPer Ledger

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

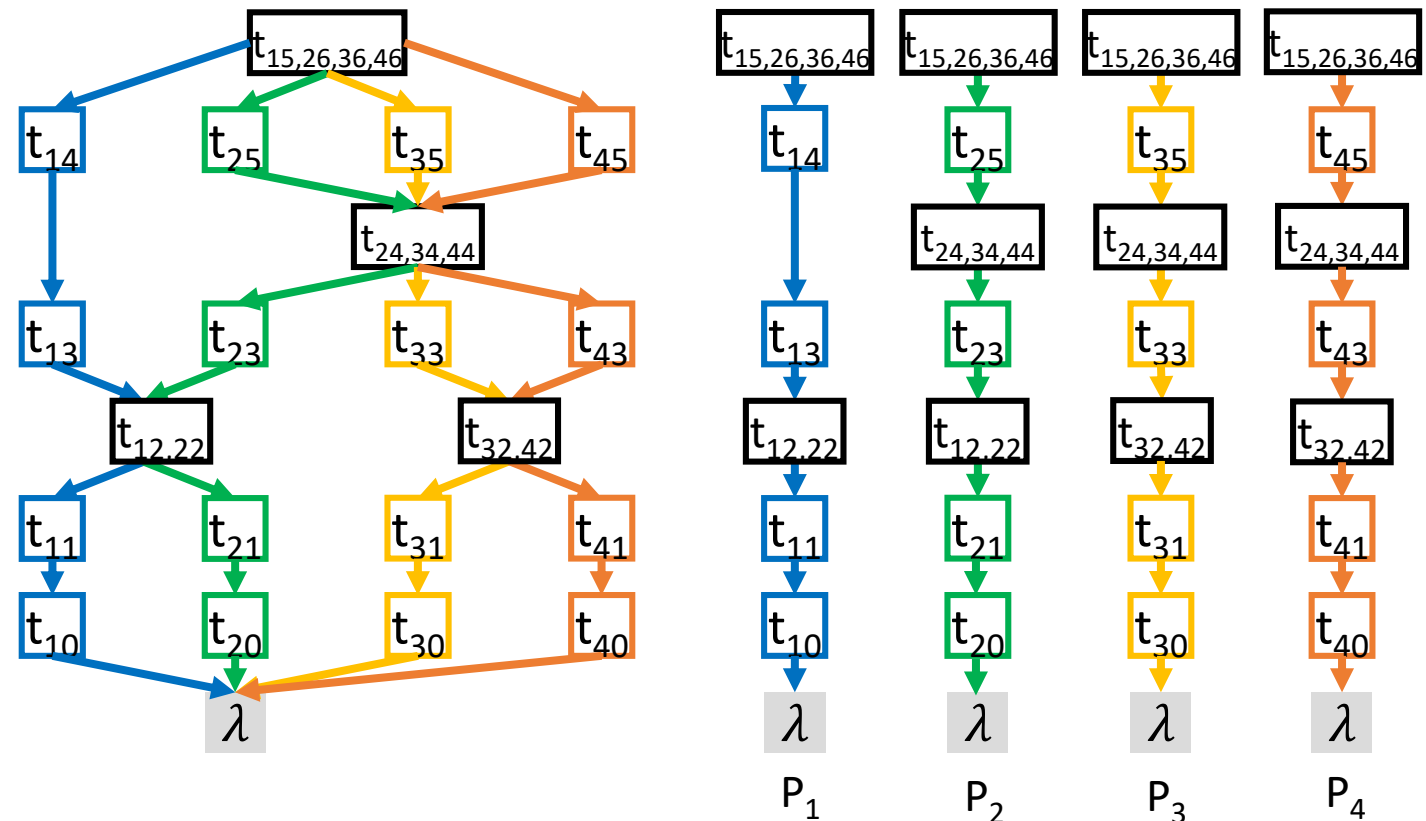
- Intra-shard transactions of different clusters are processed in parallel
- Cross-shard transactions with **non-overlapping clusters** are processed in parallel
- A cross-shard transaction includes multiple hash pointers



SharPer Ledger

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

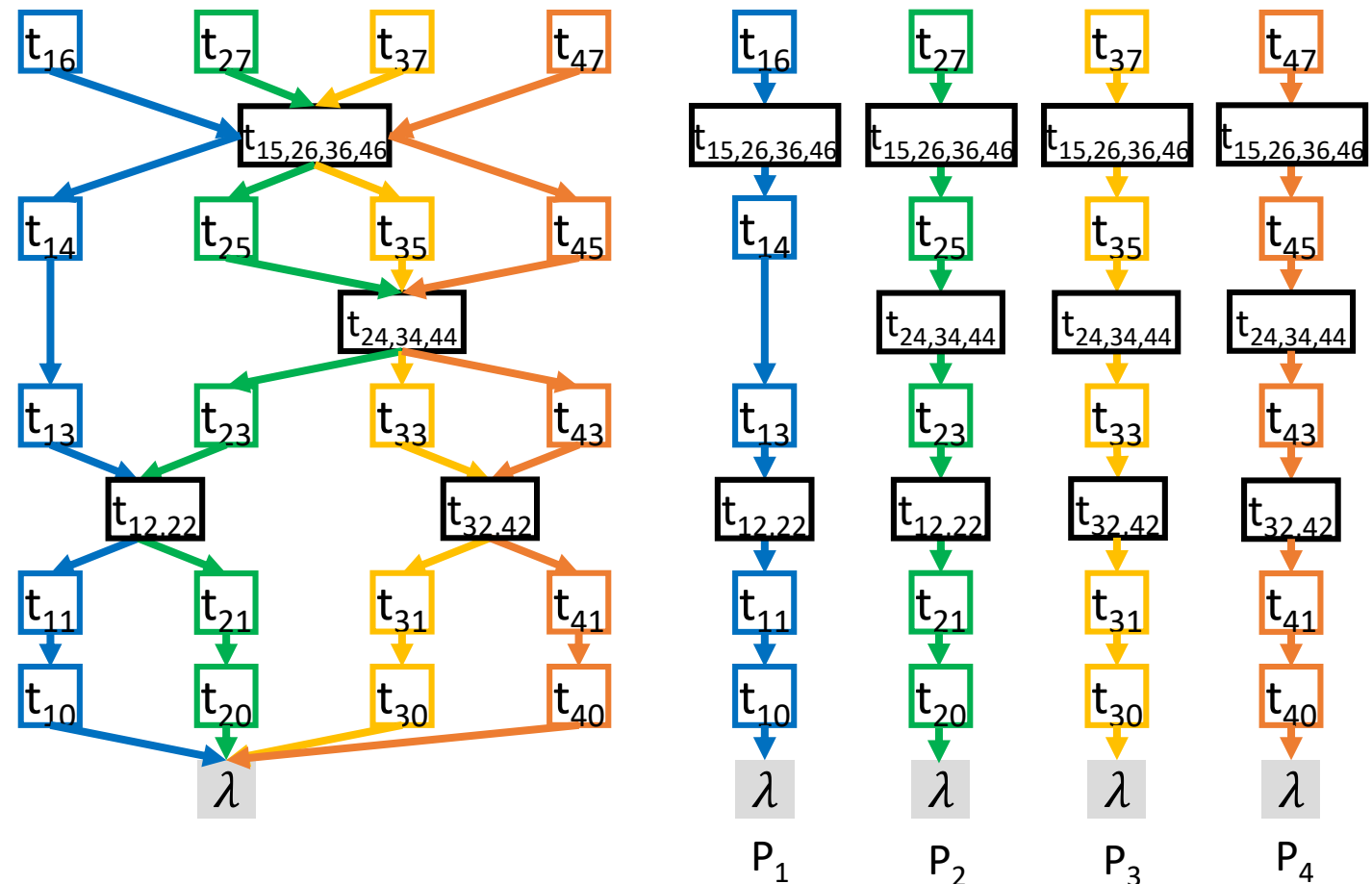
- Intra-shard transactions of different clusters are processed in parallel
- Cross-shard transactions with **non-overlapping clusters** are processed in parallel
- A cross-shard transaction includes multiple hash pointers
- All clusters might be involved in a cross-shard transaction



SharPer Ledger

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

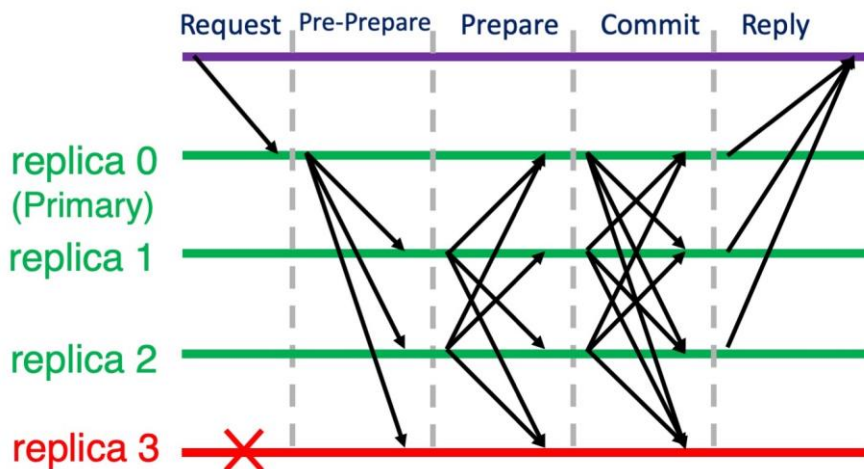
- Intra-shard transactions of different clusters are processed in parallel
- Cross-shard transactions with **non-overlapping clusters** are processed in parallel
- A cross-shard transaction includes multiple hash pointers
- All clusters might be involved in a cross-shard transaction



Consensus in SharPer

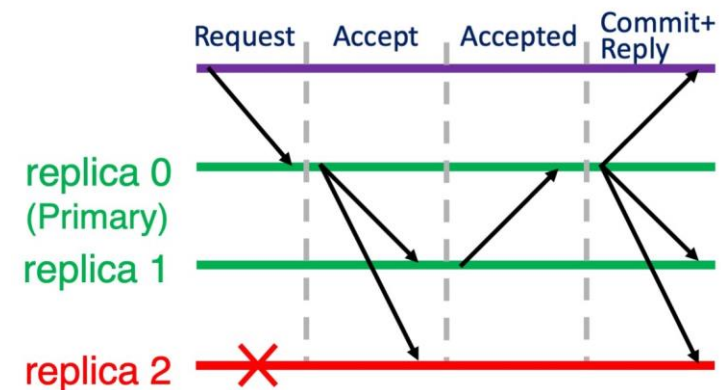
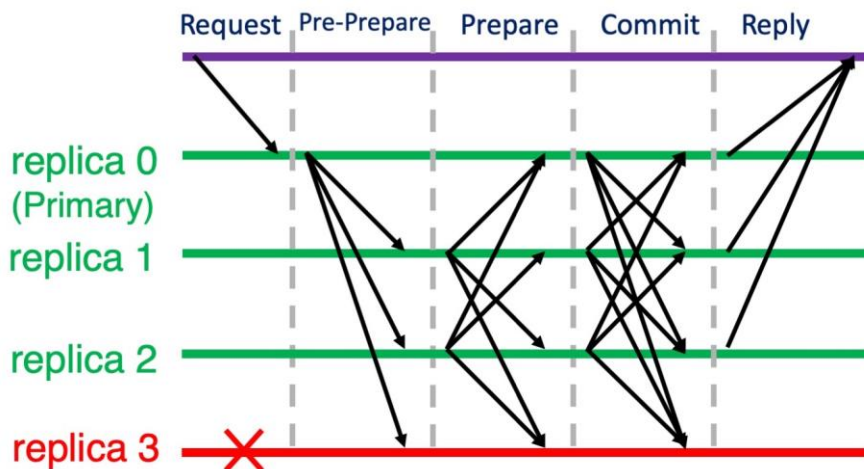
Consensus in SharPer

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



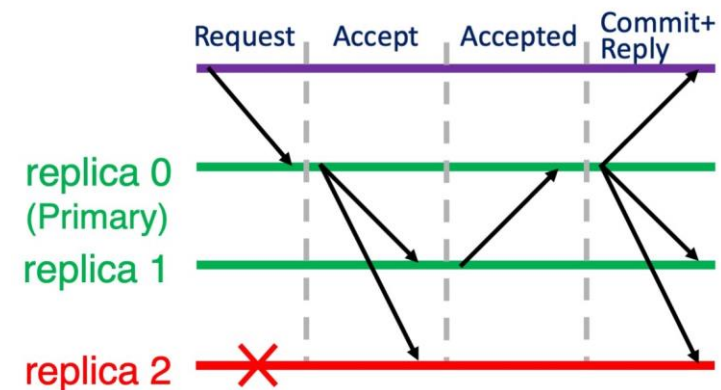
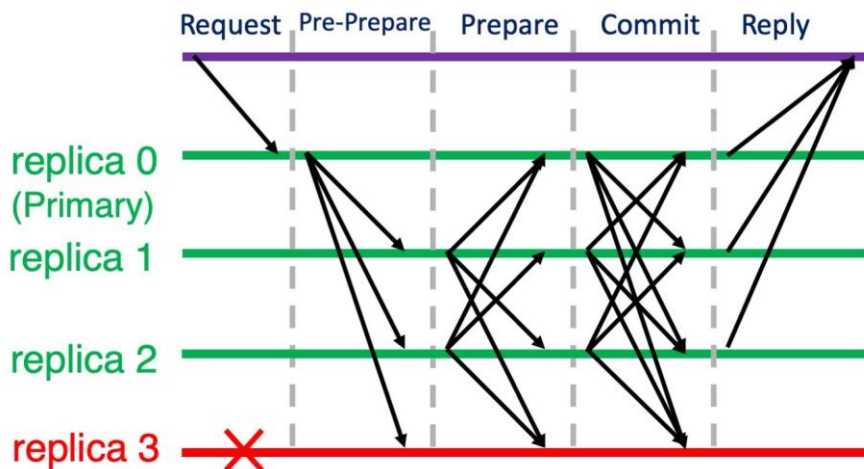
Consensus in SharPer

- **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



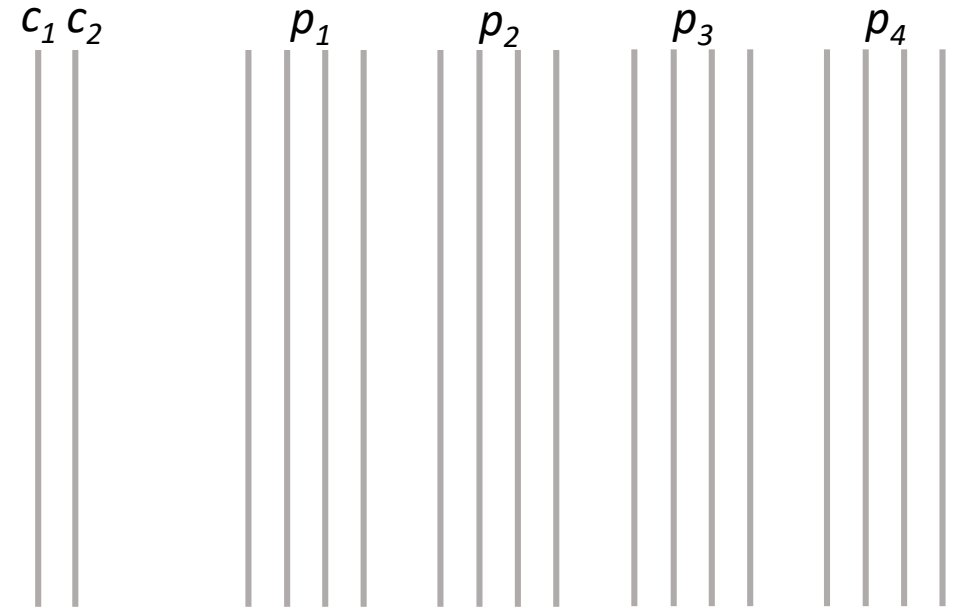
Consensus in SharPer

- **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



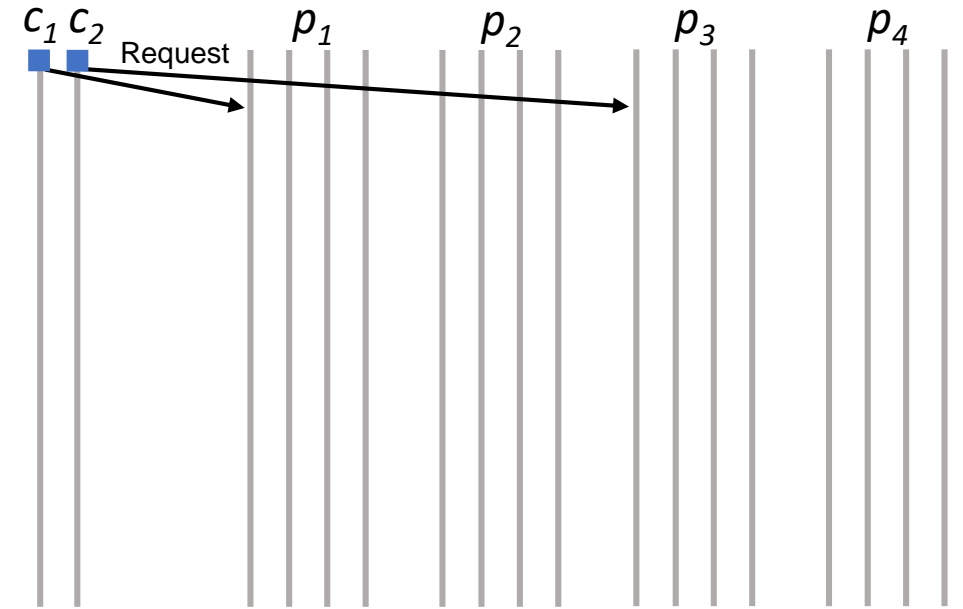
Cross-Shard Consensus in SharPer

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



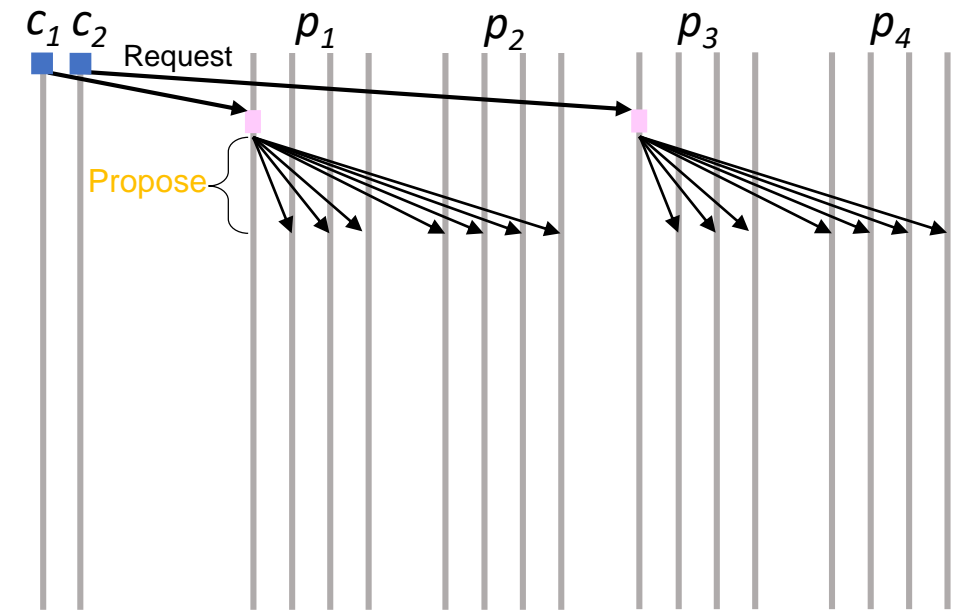
Cross-Shard Consensus in SharPer

Non-overlapping cross-shard transactions can be processed **in parallel**
Clients (c_1 and c_2) send requests to the (pre-elected) primary nodes



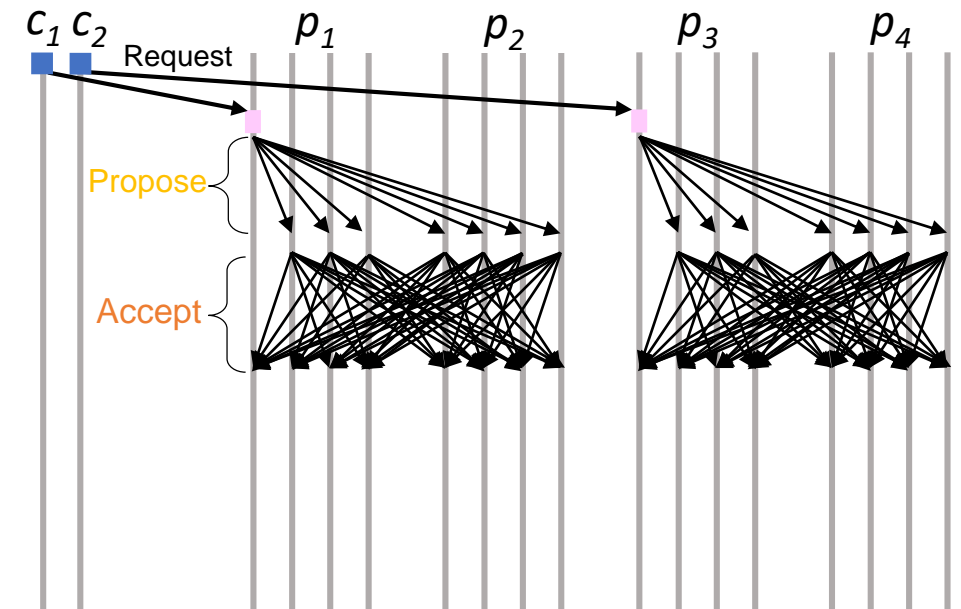
Cross-Shard Consensus in SharPer

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

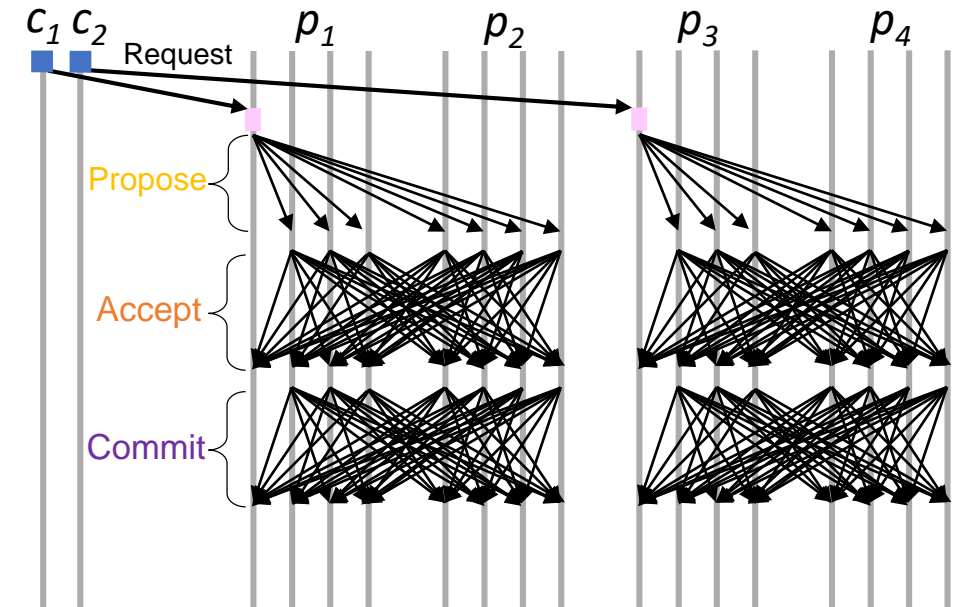
Non-overlapping cross-shard transactions can be processed **in parallel**

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



Cross-Shard Consensus in SharPer

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

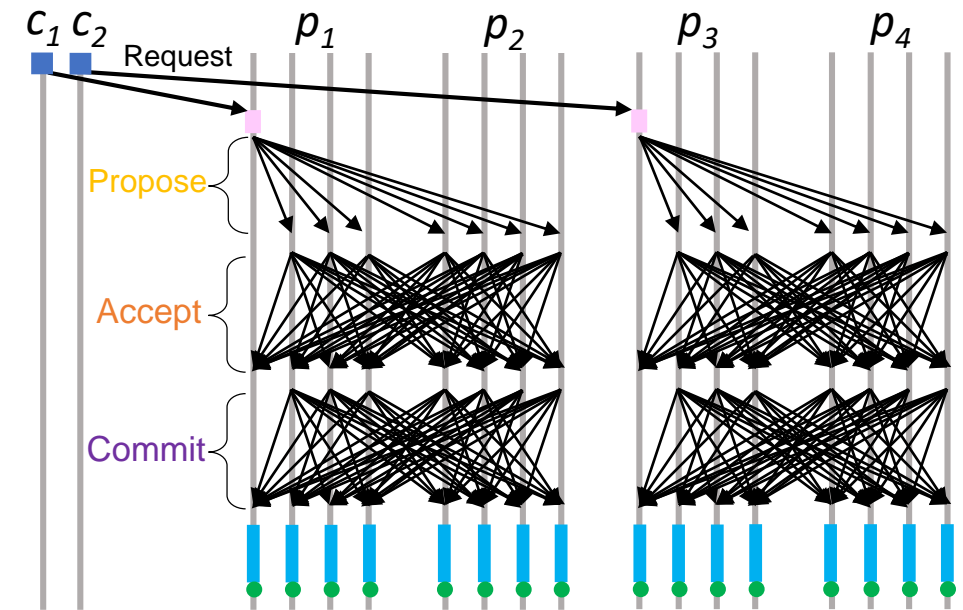
Clients (c_1 and c_2) send requests to the (pre-elected) primary nodes

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

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*



Manufacturer



Middleman



Bulk Buyer



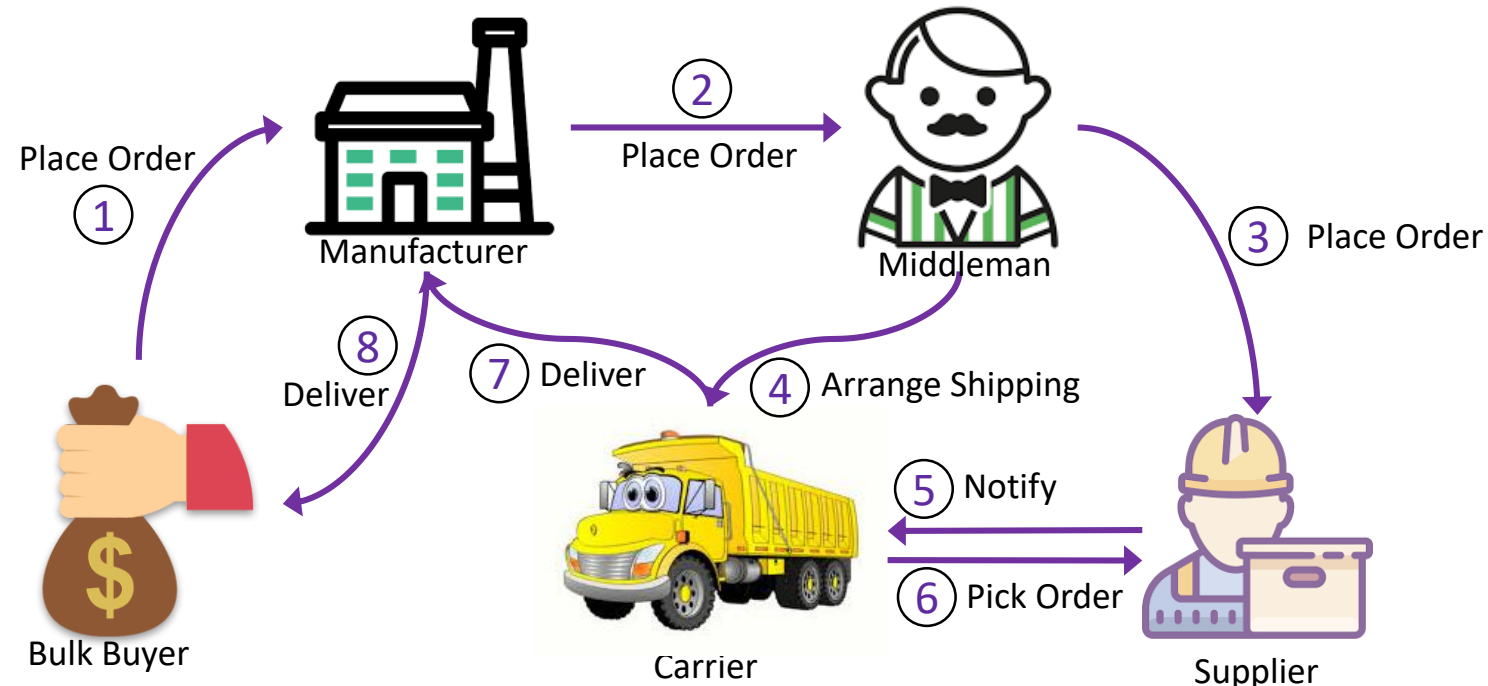
Carrier



Supplier

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

Confidentiality issue

Second Solution: Deploy each application on a separate blockchain system

- Use **another** blockchain system for the cross-application transactions

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

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

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*

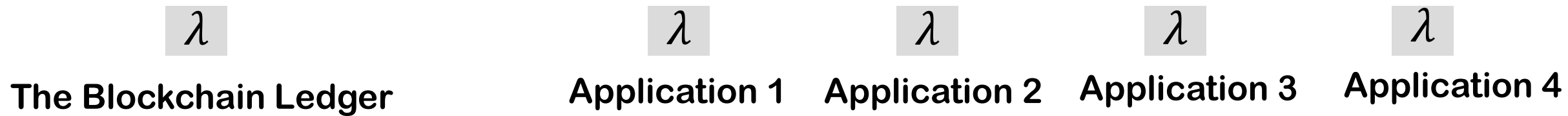
CAPER: A Cross-Application Permissioned Blockchain

- 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

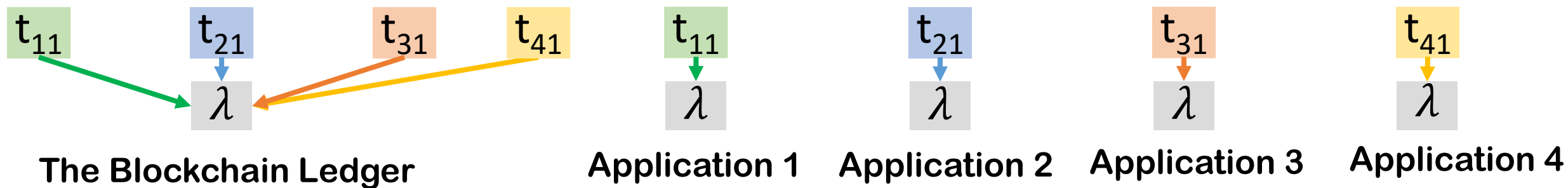
- Distributed applications collaborate with each other following *SLAs*
- Two types of transactions: *internal* and *cross-application*
- Cross-application transactions are *visible to all* applications
- Internal transactions of each application are *confidential*
- The blockchain ledger is formed as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



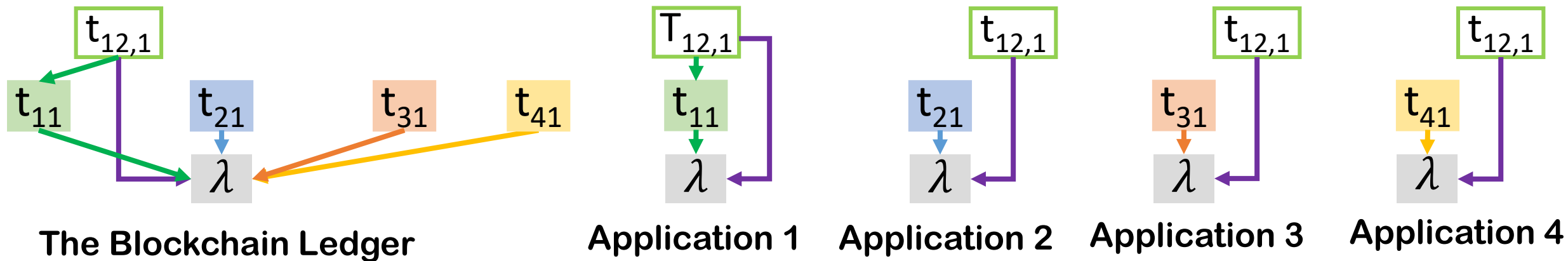
The Blockchain Ledger of CAPER

Each application has its own internal transactions

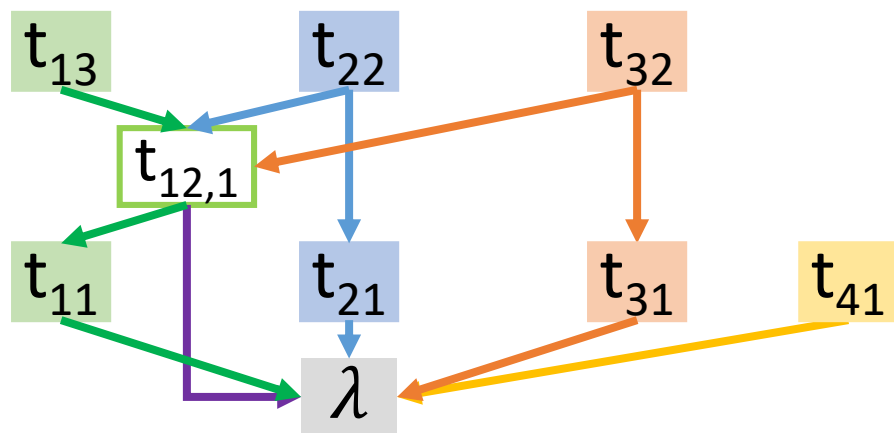


The Blockchain Ledger of CAPER

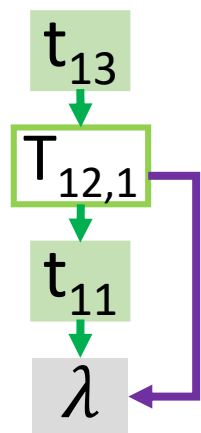
Cross-application transactions are maintained by every application



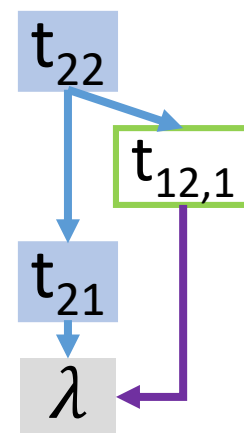
The Blockchain Ledger of CAPER



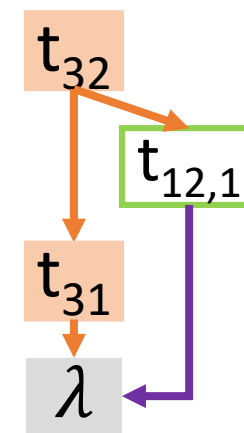
The Blockchain Ledger



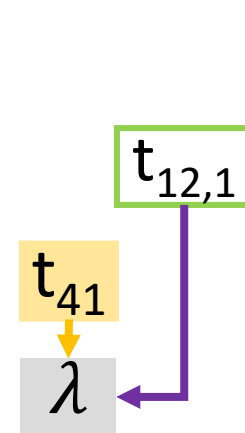
Application 1



Application 2

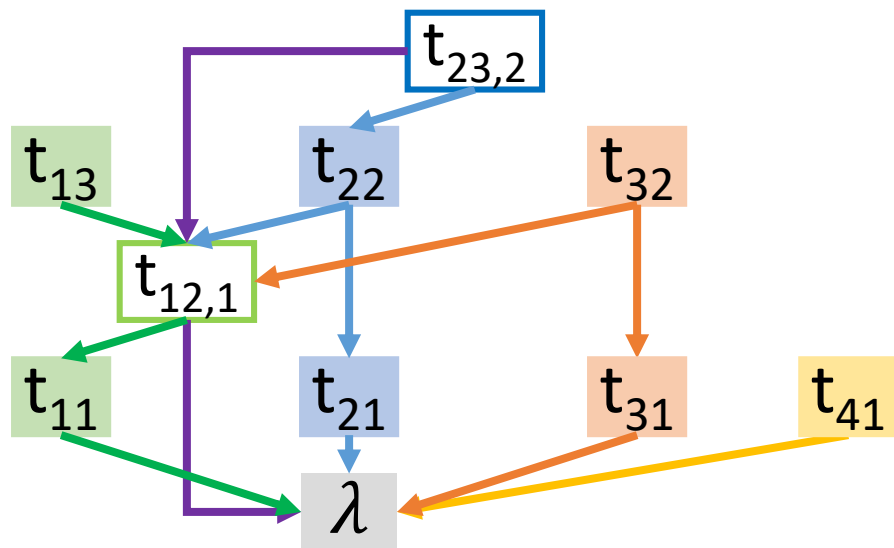


Application 3

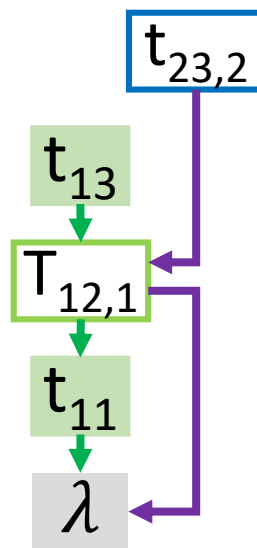


Application 4

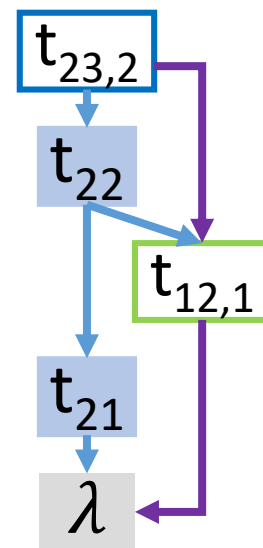
The Blockchain Ledger of CAPER



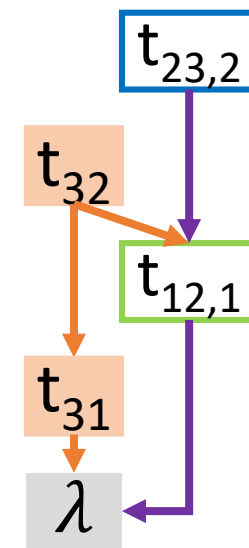
The Blockchain Ledger



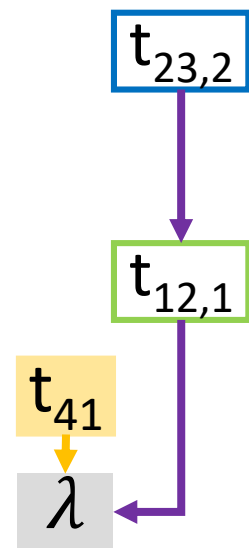
Application 1



Application 2

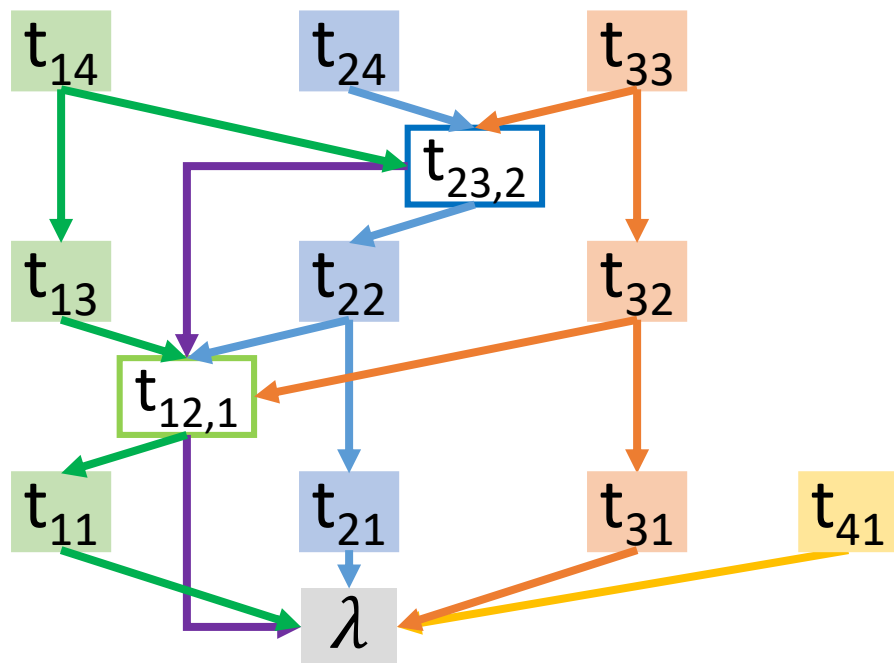


Application 3

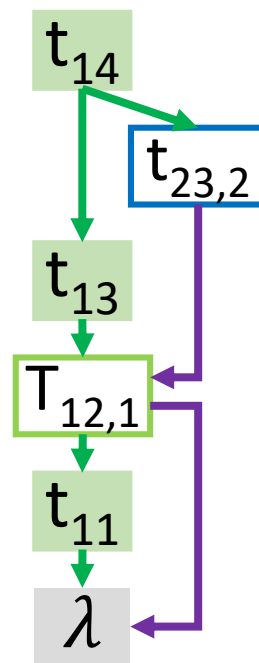


Application 4

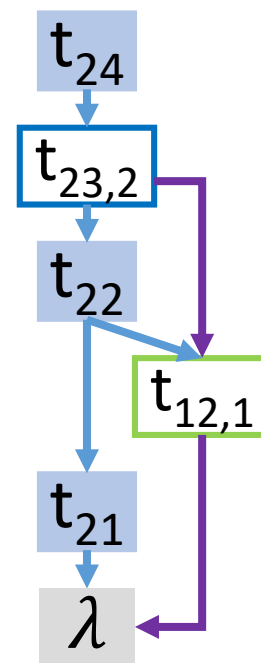
The Blockchain Ledger of CAPER



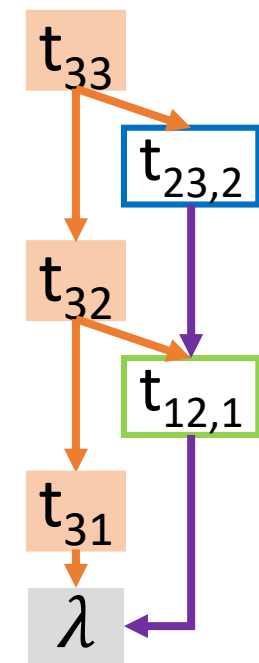
The Blockchain Ledger



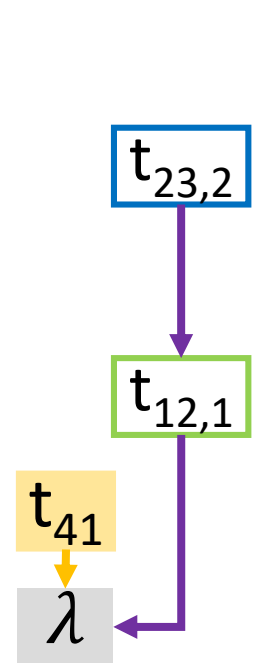
Application 1



Application 2

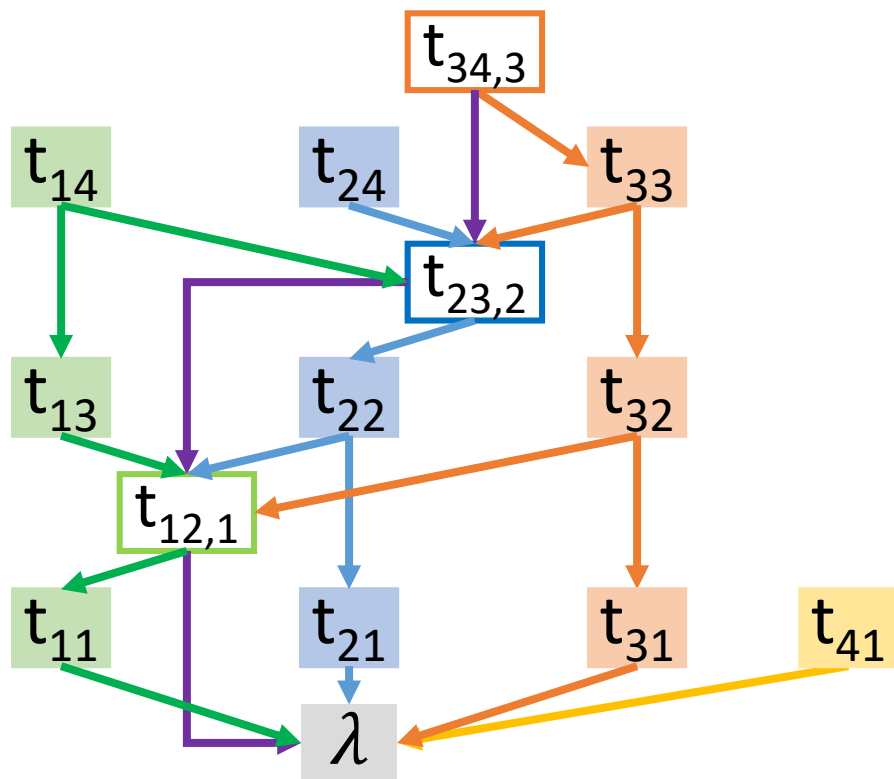


Application 3

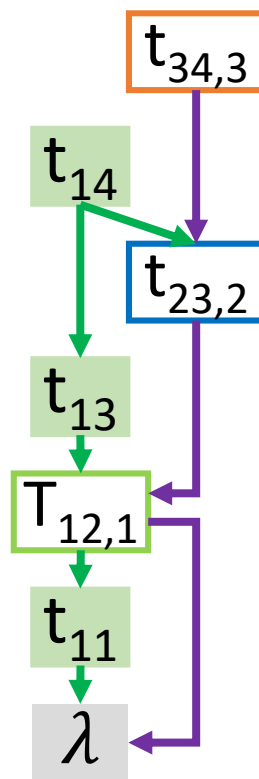


Application 4

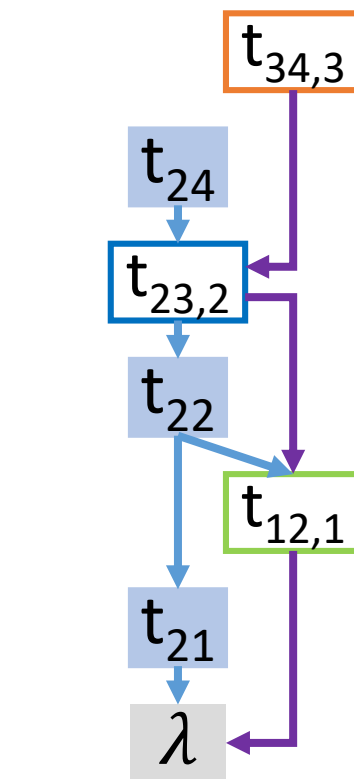
The Blockchain Ledger of CAPER



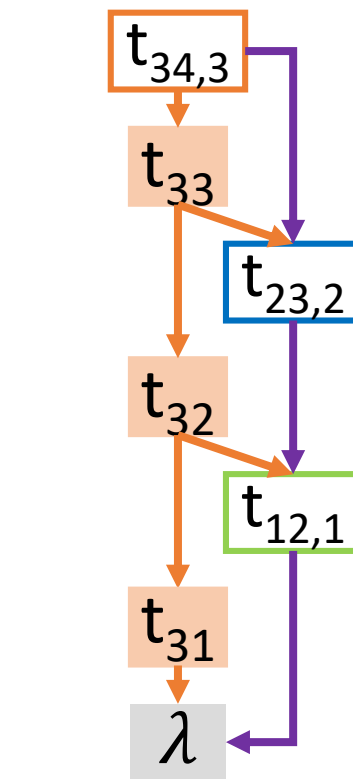
The Blockchain Ledger



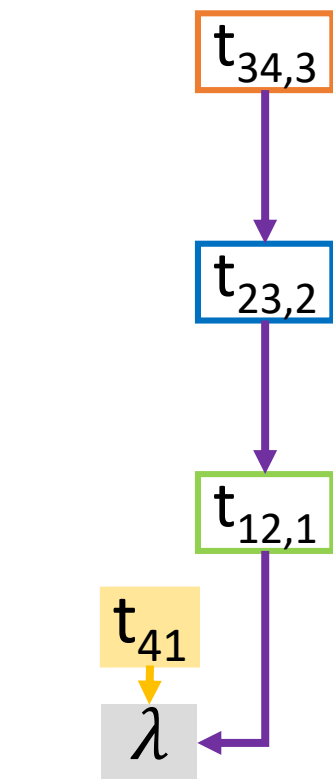
Application 1



Application 2

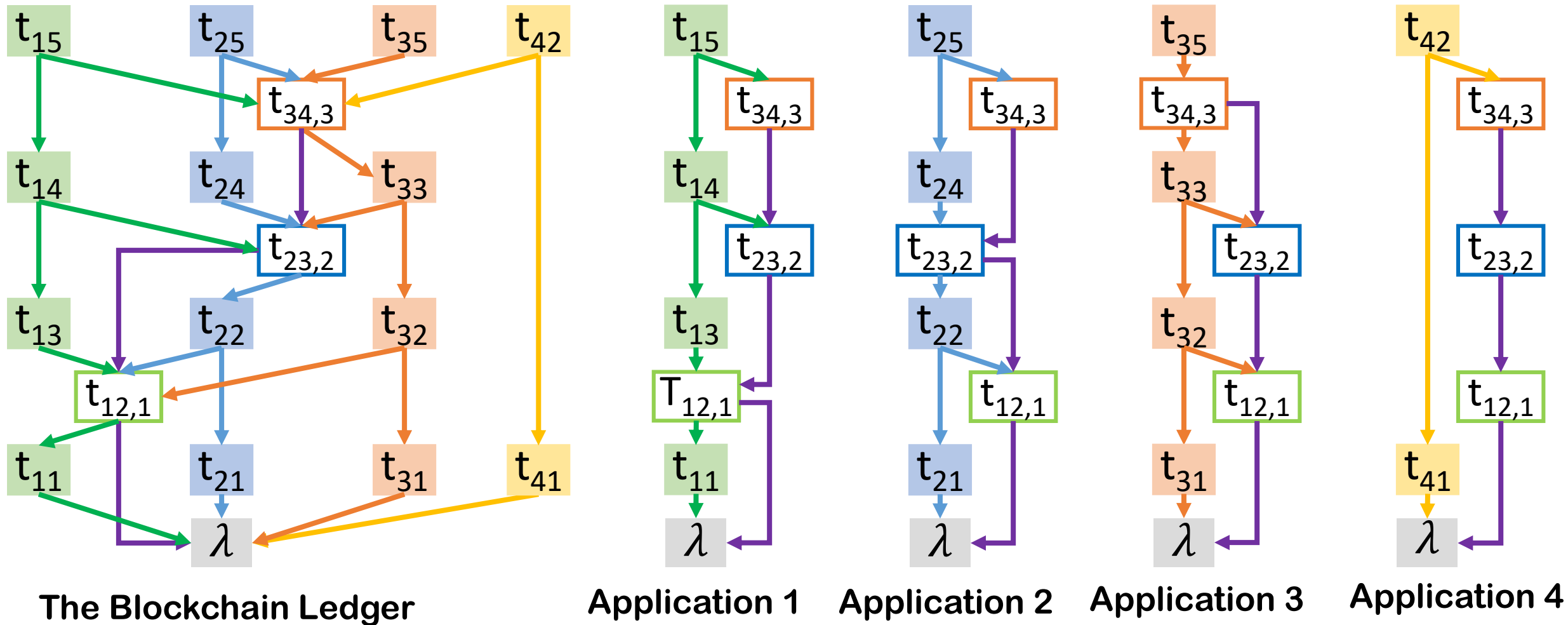


Application 3



Application 4

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!

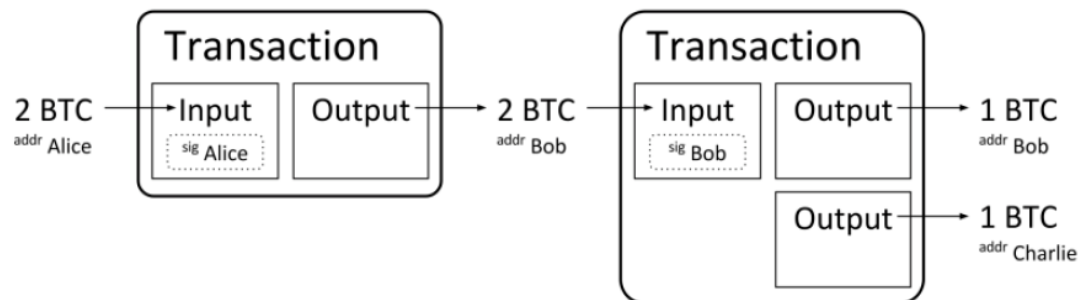


• Quorum uses **zero knowledge proof**

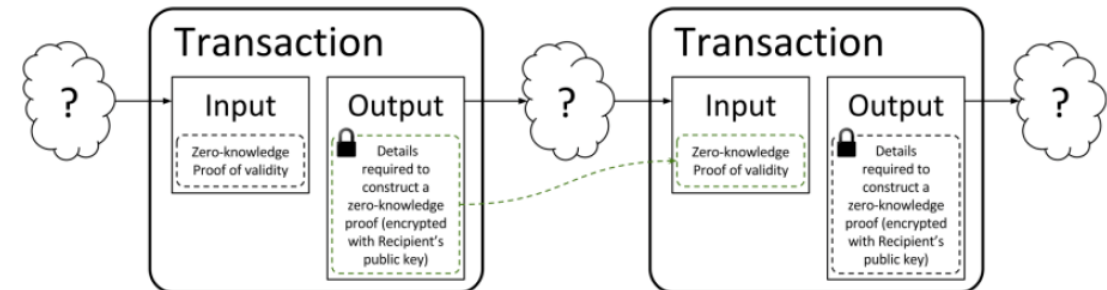


• Fabric defines **Private data collections**

Transaction verification in Bitcoin



Transaction verification in Zcash





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](#)

The Hyperledger Greenhouse

Business Blockchain Frameworks & Tools Hosted by Hyperledger



Community Stewardship and Technical, Legal, Marketing, Organizational Infrastructure

Frameworks



Permissionable smart contract machine (EVM)



Permissioned with channel support



WebAssembly-based project for building supply chain solutions



Decentralized identity



Mobile application focus



Permissioned & permissionless support; EVM transaction family

Tools



Infrastructure for peer-to-peer interactions



Blockchain framework benchmark platform



As-a-service deployment



Model and build blockchain networks



View and explore data on the blockchain



Ledger interoperability



Advanced transaction execution and state management



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

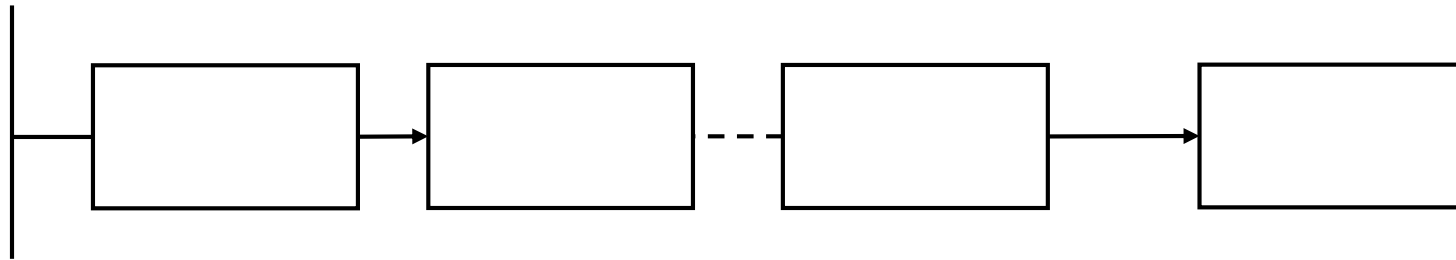
- Make: Honda
- Model: Civic
- Year: ..
- VIN: ...
- Owner: Alice
- Price: x ethers

```
Buy () {  
    // transfer ownership code  
}
```

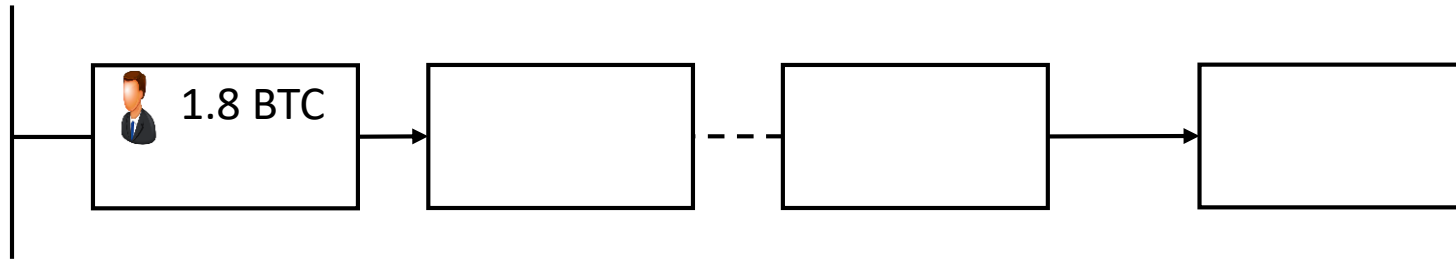


Smart Contracts

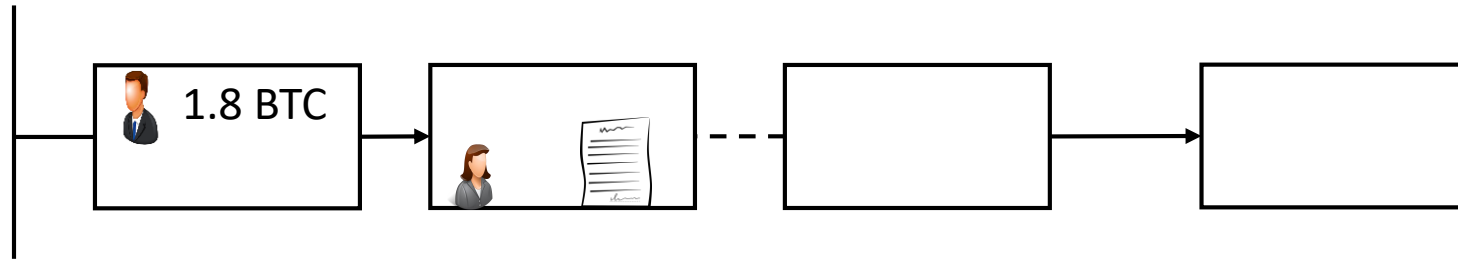
Smart Contracts



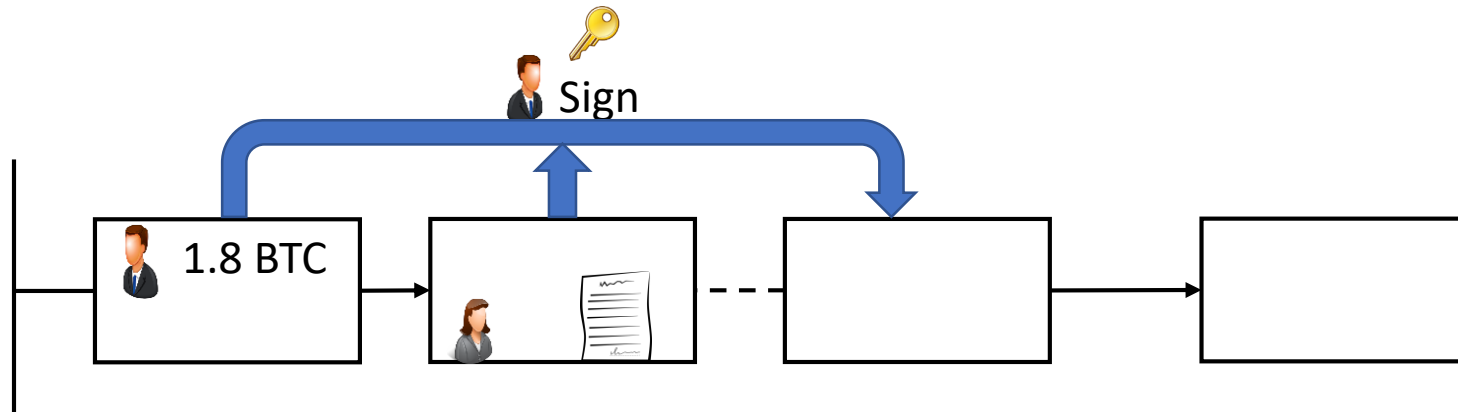
Smart Contracts



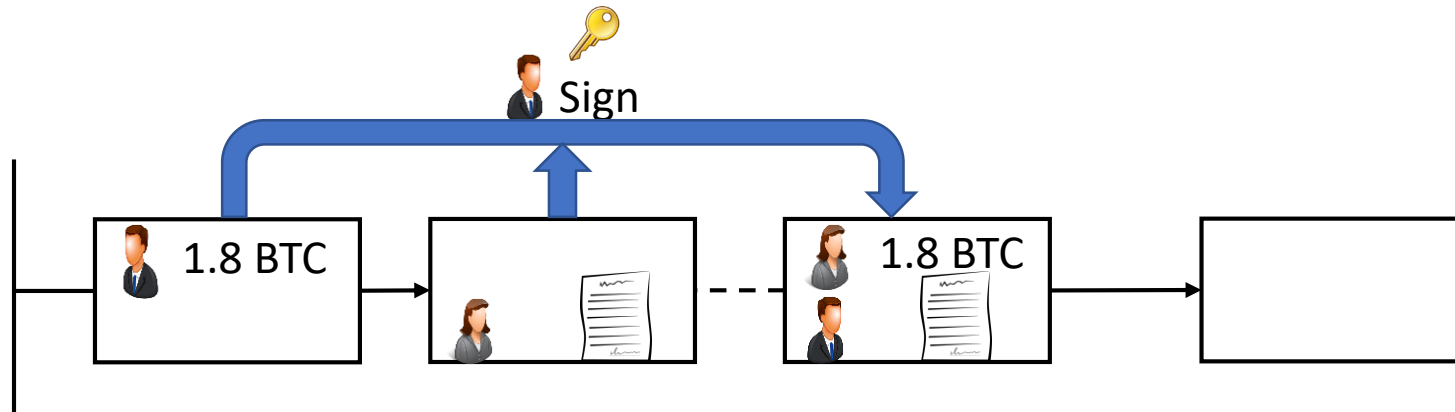
Smart Contracts



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!



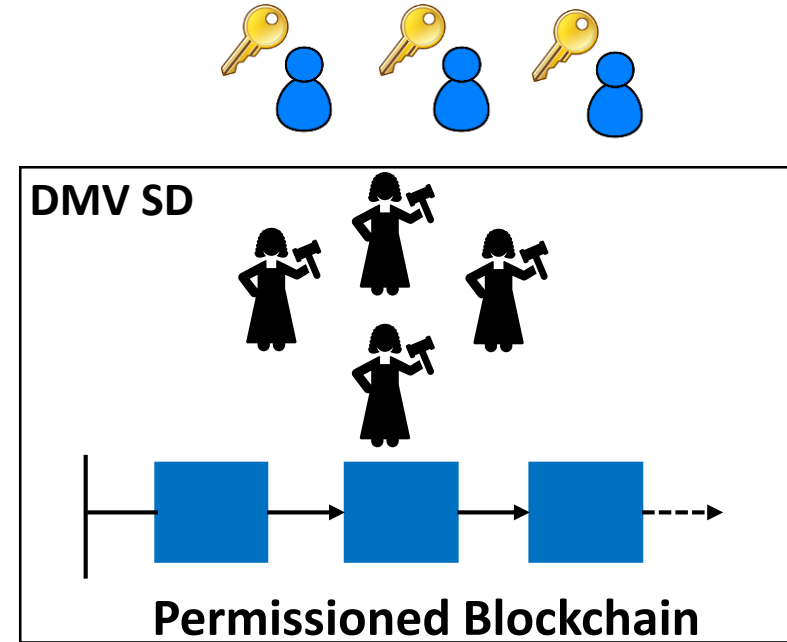
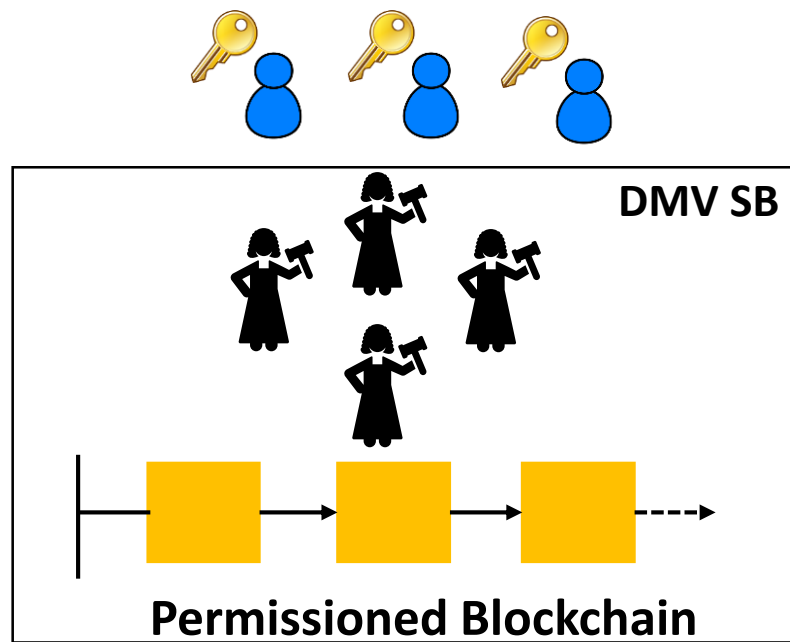
- Permissioned Blockchains
 - Requires trust
 - Trust can be distributed among several organizations
 - Banks
 - Governments
 - NGOs

Global Asset Management

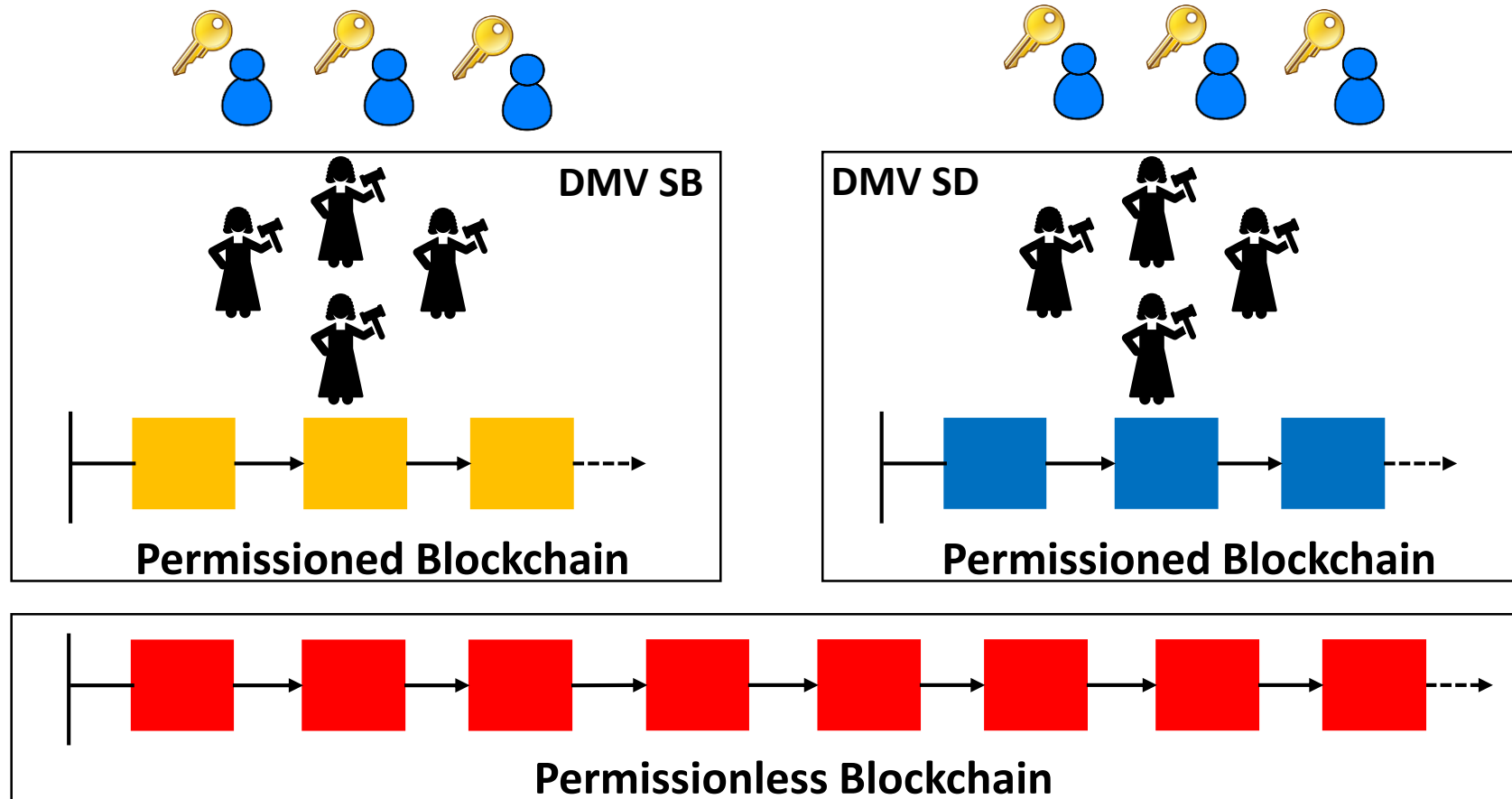
Global Asset Management



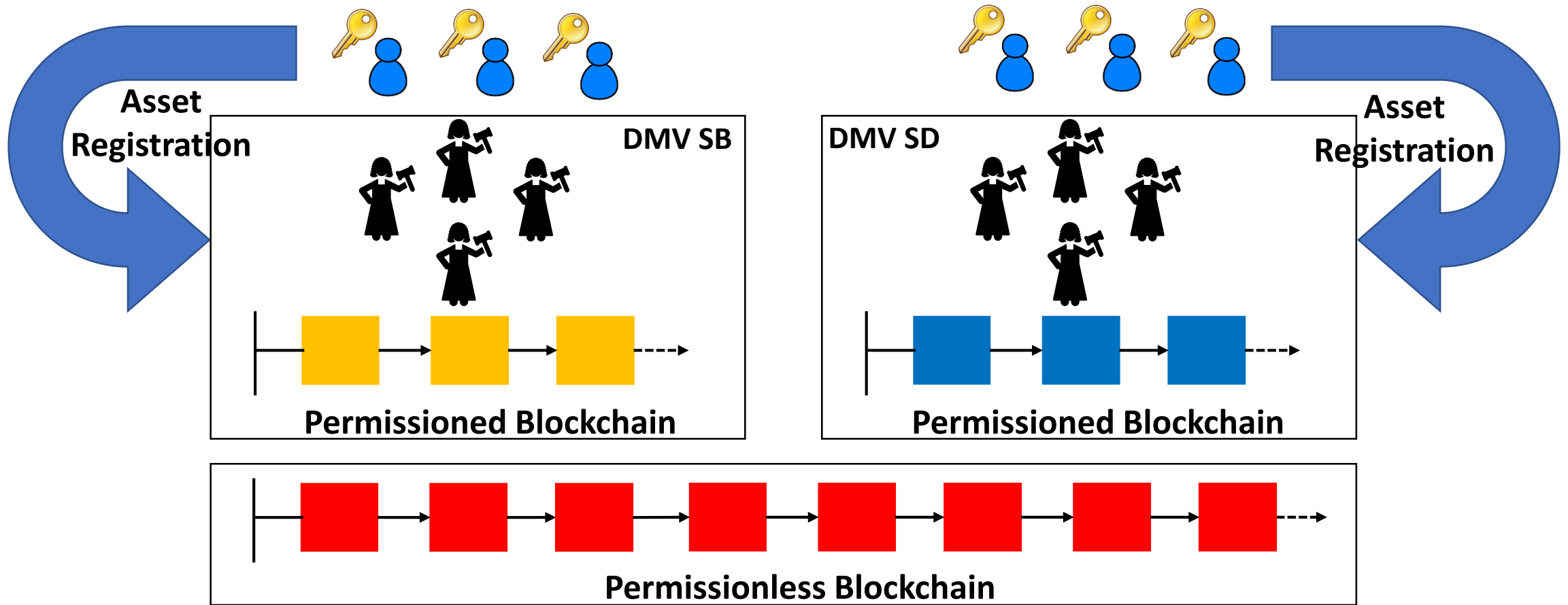
Global Asset Management



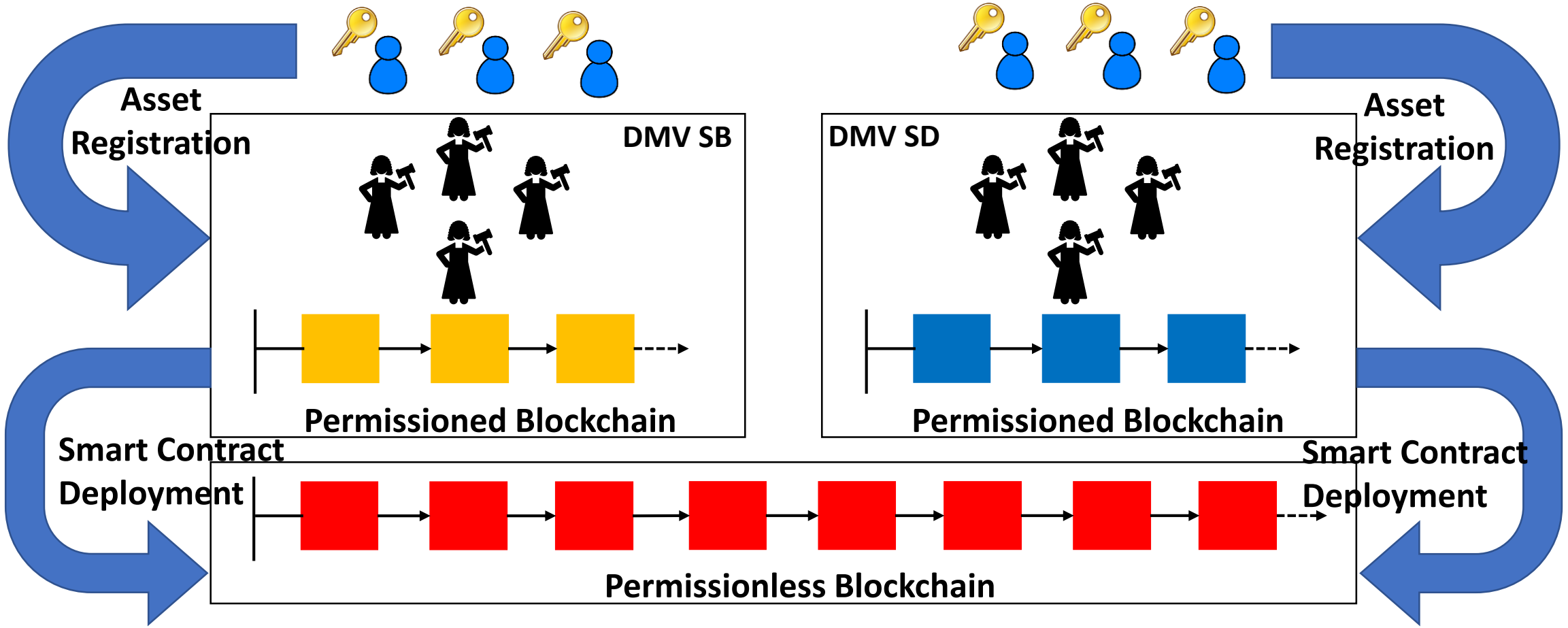
Global Asset Management



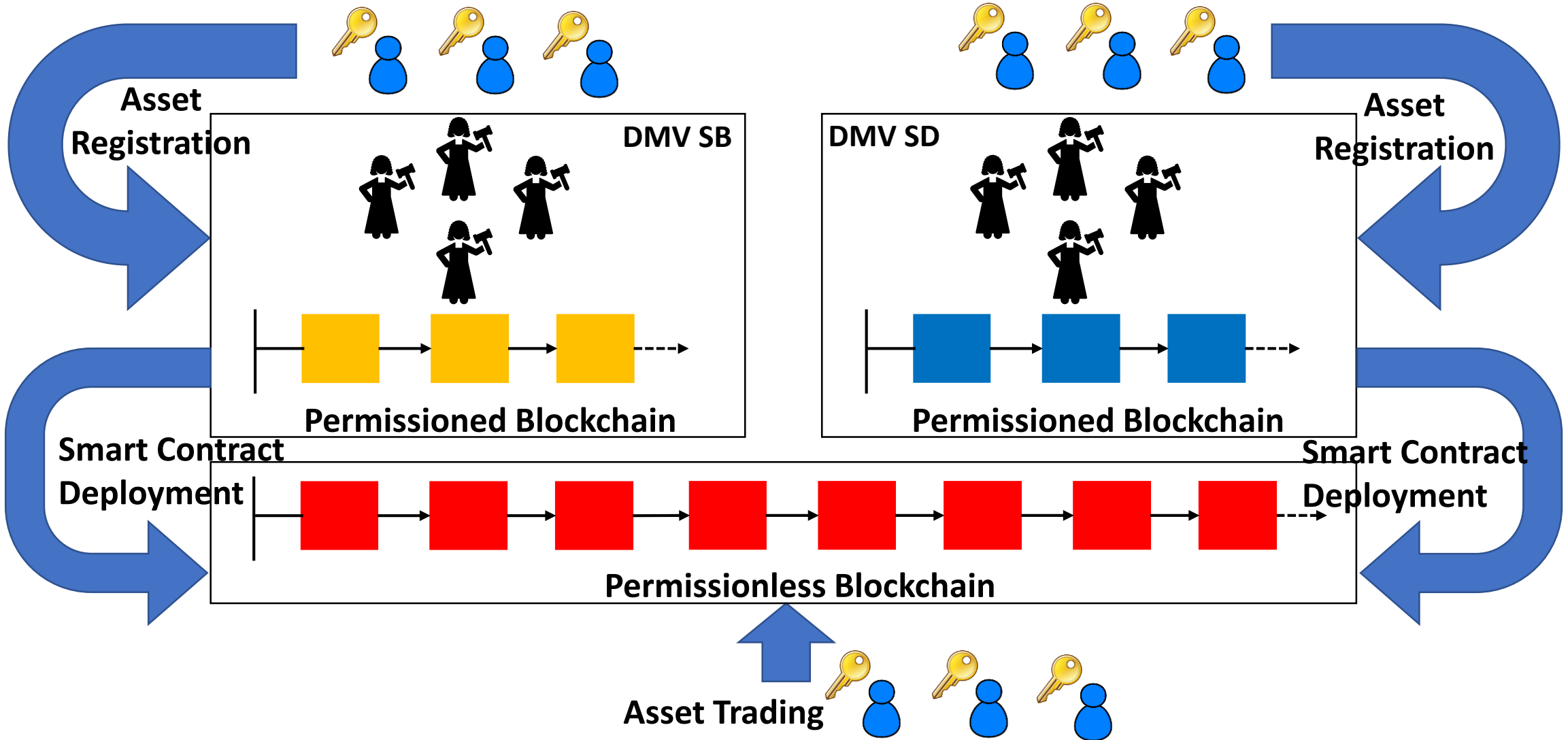
Global Asset Management



Global Asset Management



Global Asset Management



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?