CS61065: Theory And Applications of Blockchain

# Consensus in Permissionless Settings

Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Sandip Chakraborty
sandipc@cse.iitkgp.ac.in

Shamik Sural
shamik@cse.iitkgp.ac.in

# Permissionless Model

- Open network
  - Anyone can join in the network and initiate transactions
  - Participants are free to leave the network, and can join later again

# Permissionless Model

- Open network
  - Anyone can join in the network and initiate transactions
  - Participants are free to leave the network, and can join later again

- **Assumption: More than 50% of the participants are honest**
  - A society cannot run if majority of its participants are dishonest !!

# Permissionless Model

- Open network
  - Anyone can join in the network and initiate transactions
  - Participants are free to leave the network, and can join later again
- **Assumption: More than 50% of the participants are honest**
  - A society cannot run if majority of its participants are dishonest !!

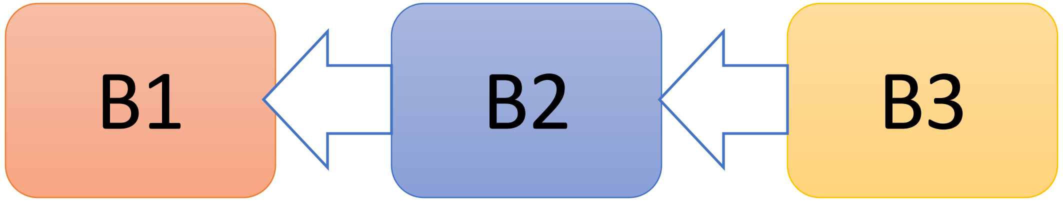
# Consensus in a Permissionless Model - Challenges

- Participants do not know others
    - Cannot use message passing !!

- Anyone can propose a new block
    - Who is going to add the next block in the blockchain?

- The network is asynchronous
    - We do not have any global clock
    - Theoretically, a node may see the blocks in different orders

# Consensus in a Permissionless Model - Challenges



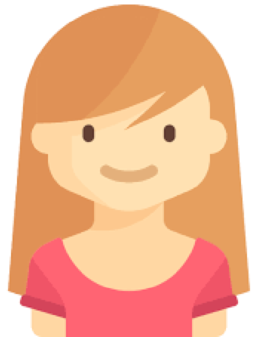
B1

B2

B3

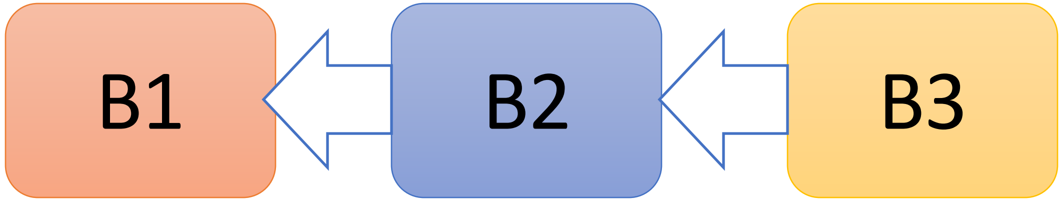
- Participants do not know others
  - Cannot use message passing !!

- Anyone can propose a new block
  - Who is going to add the next block in the blockchain?
- The network is asynchronous
  - We do not have any global clock
  - Theoretically, a node may see the messages in different orders

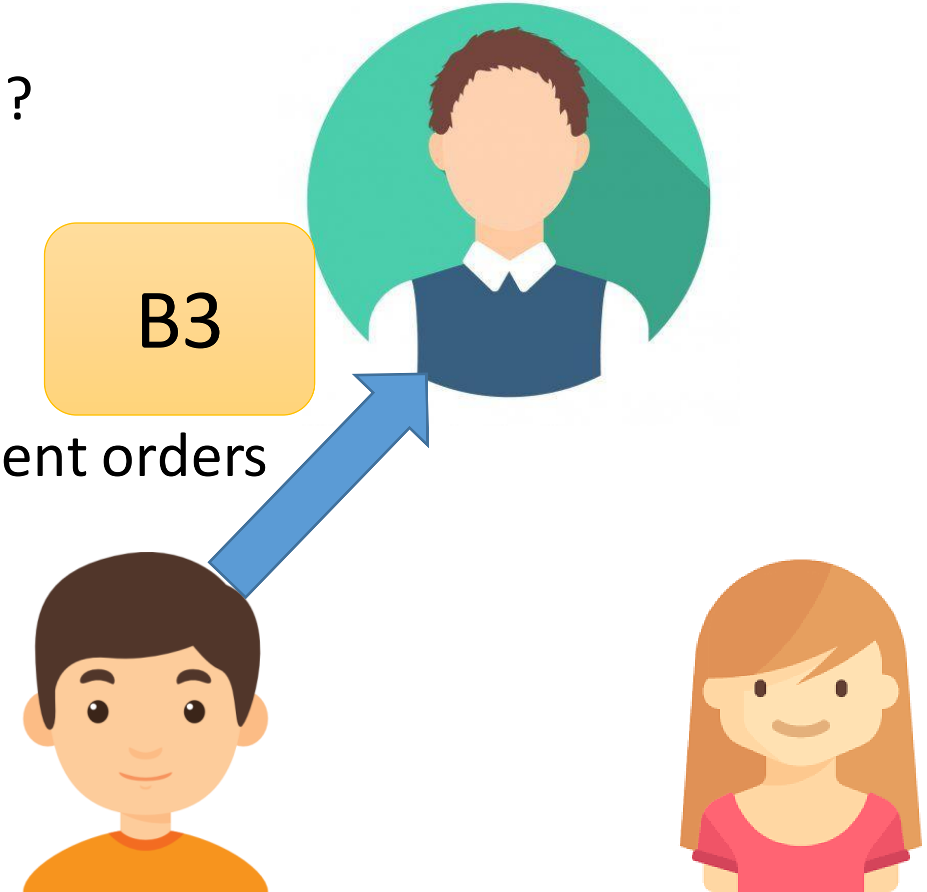


# Consensus in a Permissionless Model - Challenges

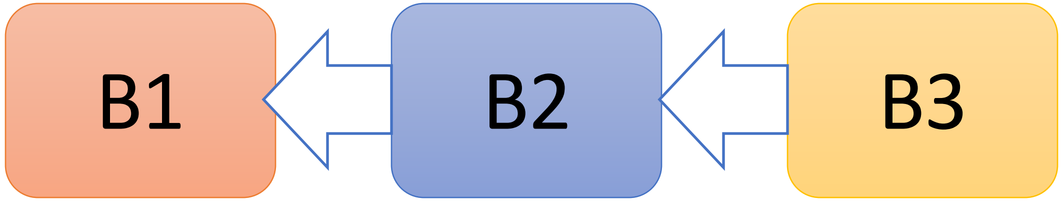
- Participants do not know others
  - Cannot use message passing !!



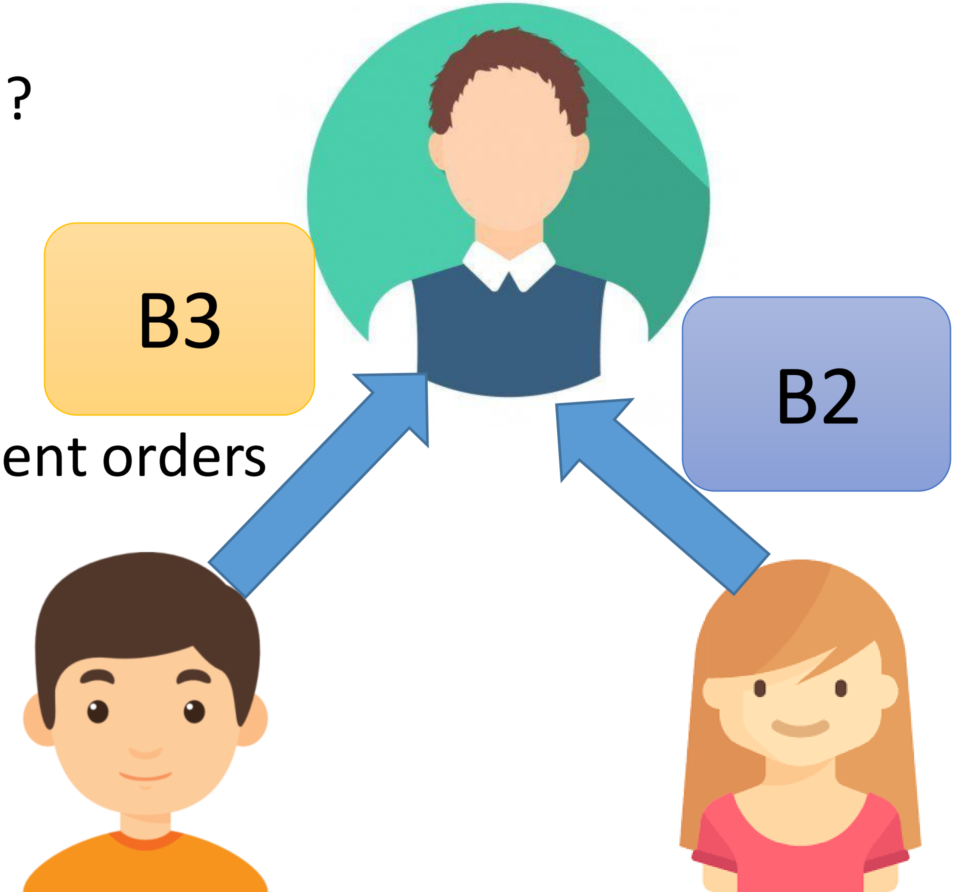
- Anyone can propose a new block
  - Who is going to add the next block in the blockchain?
- The network is asynchronous
  - We do not have any global clock
  - Theoretically, a node may see the messages in different orders



# Consensus in a Permissionless Model - Challenges



- Participants do not know others
    - Cannot use message passing !!
- Anyone can propose a new block
    - Who is going to add the next block in the blockchain?
- The network is asynchronous
    - We do not have any global clock
    - Theoretically, a node may see the messages in different orders



# Consensus in a Permissionless Model - Challenges

- Participants do not know others
  - Cannot use message passing !!
- Anyone can propose a new block
  - Who is going to add the next block in the blockchain?
- The network is asynchronous
  - We do not have any global clock
  - Theoretically, a node may see the messages in different orders
- Any types of monopoly needs to be prevented
  - A single user or a group of users should not gain the control – we don't trust anyone

# Remember the FLP Impossibility?

- Synchronous vs Asynchronous Networks
  - **Synchronous**: I am sure that I'll get the message within a predefined time threshold
  - **Asynchronous**: I am not sure whether and when the message will arrive

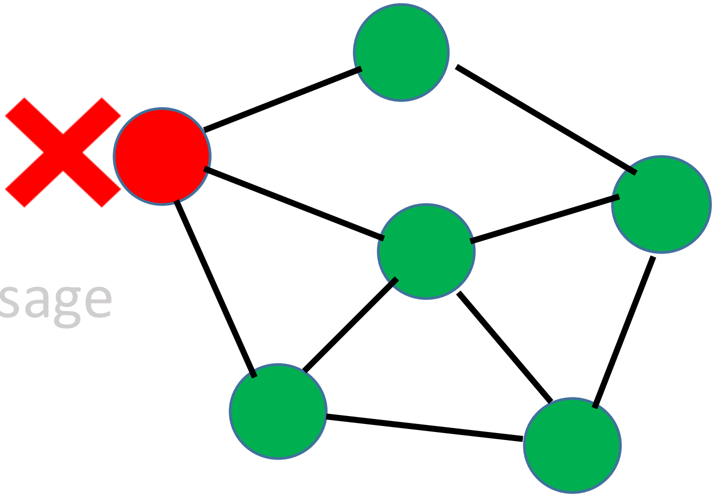
- Failures in a network --
  - **Crash Fault**: A node stops responding
  - **Link Fault** (or Network Fault): A link fails to deliver the message
  - **Byzantine Fault**: A node starts behaving maliciously

- **The Impossibility Theorem**: Consensus is not possible in a perfect asynchronous network even with a single crash failure

# Remember the FLP Impossibility?

- Synchronous vs Asynchronous Networks
  - **Synchronous**: I am sure that I'll get the message within a predefined time threshold
  - **Asynchronous**: I am not sure whether and when the message will arrive

- Failures in a network --
  - **Crash Fault**: A node stops responding
  - **Link Fault** (or Network Fault): A link fails to deliver the message
  - **Byzantine Fault**: A node starts behaving maliciously

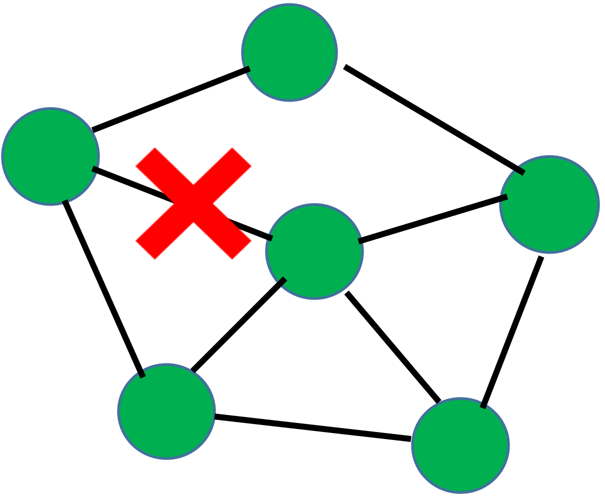


- **The Impossibility Theorem**: Consensus is not possible in a perfect asynchronous network even with a single crash failure

# Remember the FLP Impossibility?

- Synchronous vs Asynchronous Networks
  - **Synchronous**: I am sure that I'll get the message within a predefined time threshold
  - **Asynchronous**: I am not sure whether and when the message will arrive

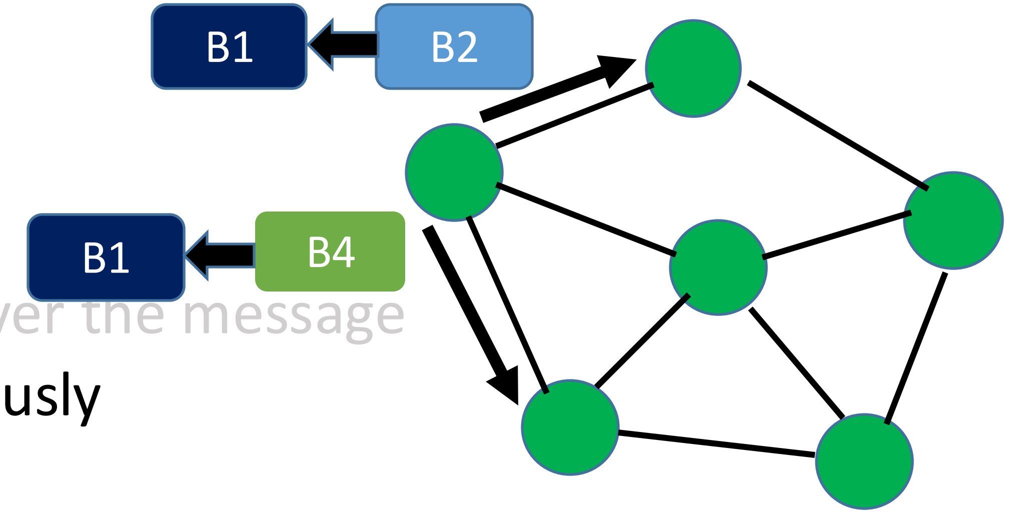
- Failures in a network --
  - **Crash Fault**: A node stops responding
  - **Link Fault** (or Network Fault): A link fails to deliver the message
  - **Byzantine Fault**: A node starts behaving maliciously



- **The Impossibility Theorem**: Consensus is not possible in a perfect asynchronous network even with a single crash failure

# Remember the FLP Impossibility?

- Synchronous vs Asynchronous Networks
  - **Synchronous**: I am sure that I'll get the message within a predefined time threshold
  - **Asynchronous**: I am not sure whether and when the message will arrive
- Failures in a network --
  - **Crash Fault**: A node stops responding
  - **Link Fault** (or Network Fault): A link fails to deliver the message
  - **Byzantine Fault**: A node starts behaving maliciously



- **The Impossibility Theorem**: Consensus is not possible in a perfect asynchronous network even with a single crash failure

# Remember the FLP Impossibility?

- Synchronous vs Asynchronous Networks
  - **Synchronous**: I am sure that I'll get the message within a predefined time threshold
  - **Asynchronous**: I am not sure whether and when the message will arrive

- Failures in a network --
  - **Crash Fault**: A node stops responding
  - **Link Fault** (or Network Fault): A link fails to deliver the message
  - **Byzantine Fault**: A node starts behaving maliciously

- **The Impossibility Theorem**: Consensus is not possible in a perfect asynchronous network even with a single crash failure
  - Cannot ensure safety and liveness simultaneously

# Safety vs Liveness Dilemma

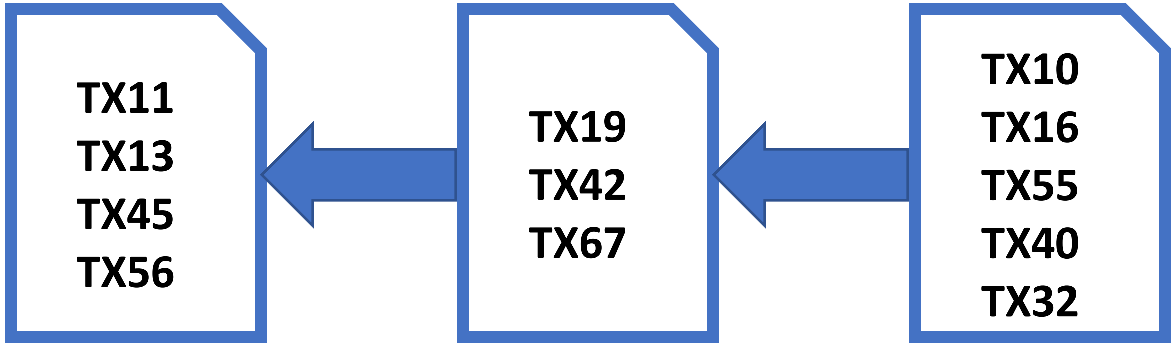
- Synchronous vs Asynchronous Networks
  - **Synchronous**: I am sure that I'll get the message within a predefined time threshold
  - **Asynchrono** arrive
- Failures in a n
  - **Crash Fault**:
  - **Link Fault** (c
  - **Byzantine Fa**
- **The Impossibility Theorem**: Consensus is not possible in a perfect asynchronous network even with a single crash failure
  - Cannot ensure safety and liveness simultaneously

**The Nakamoto Consensus (Proof of Work)**

**Liveness is more important than Safety**

**Immediate focus is on liveness with a minimum safety guarantee, full safety will be ensured eventually**

# Breaking the "Safety vs Liveness" Dilemma



TX11
TX13
TX45
TX56

TX19
TX42
TX67

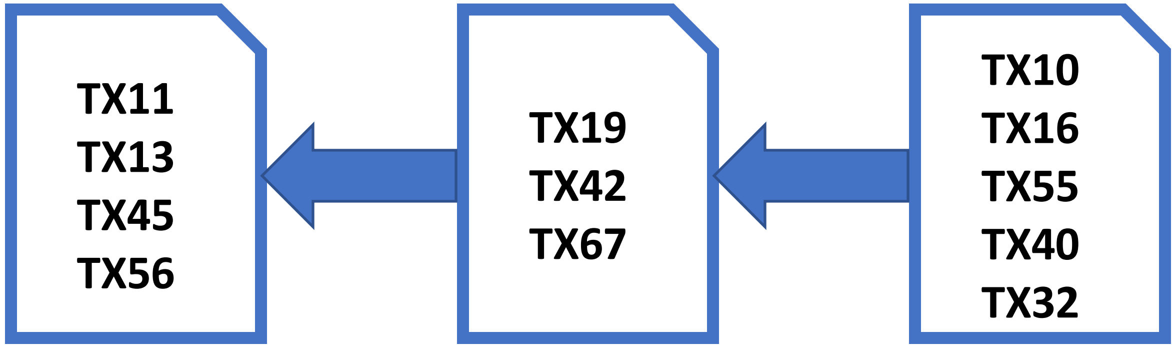
TX10
TX16
TX55
TX40
TX32

Miner 1

Miner 2

Miner 3

# Breaking the "Safety vs Liveness" Dilemma

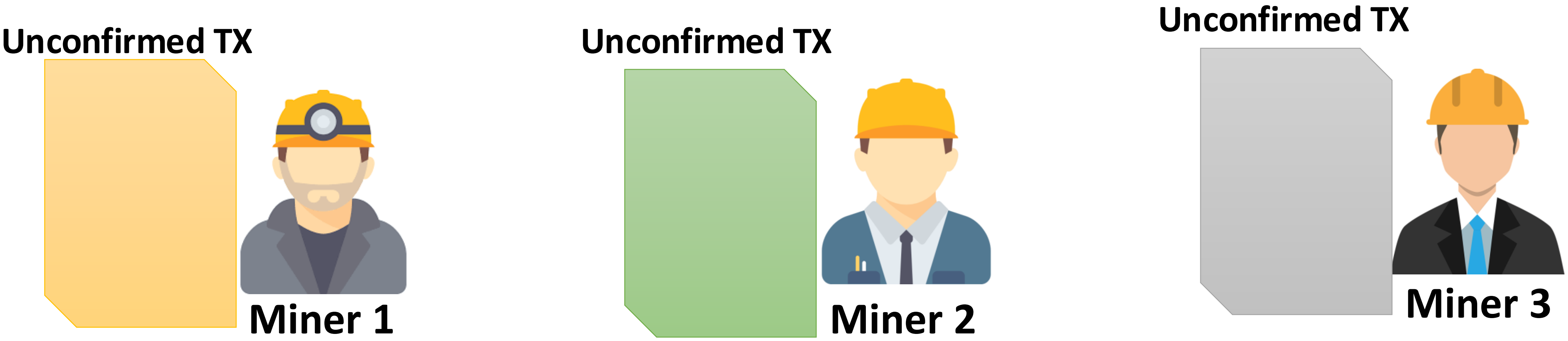


TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

**Bitcoin Unconfirmed TX (mempool) :** https://www.blockchain.com/btc/unconfirmed-transactions



Unconfirmed TX

Miner 1

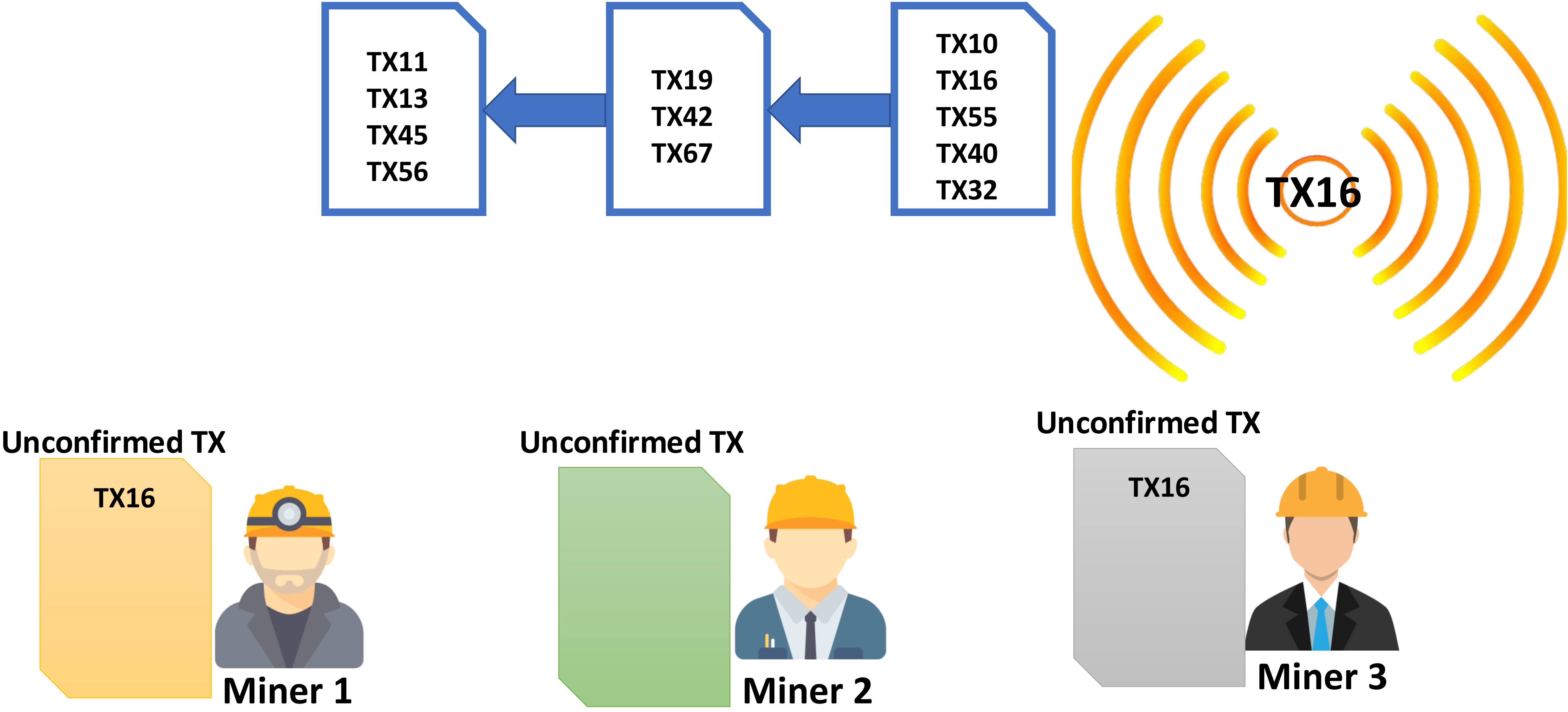
Unconfirmed TX

Miner 2

Unconfirmed TX

Miner 3

# Breaking the "Safety vs Liveness" Dilemma



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

TX16

Unconfirmed TX

TX16

Miner 1

Unconfirmed TX

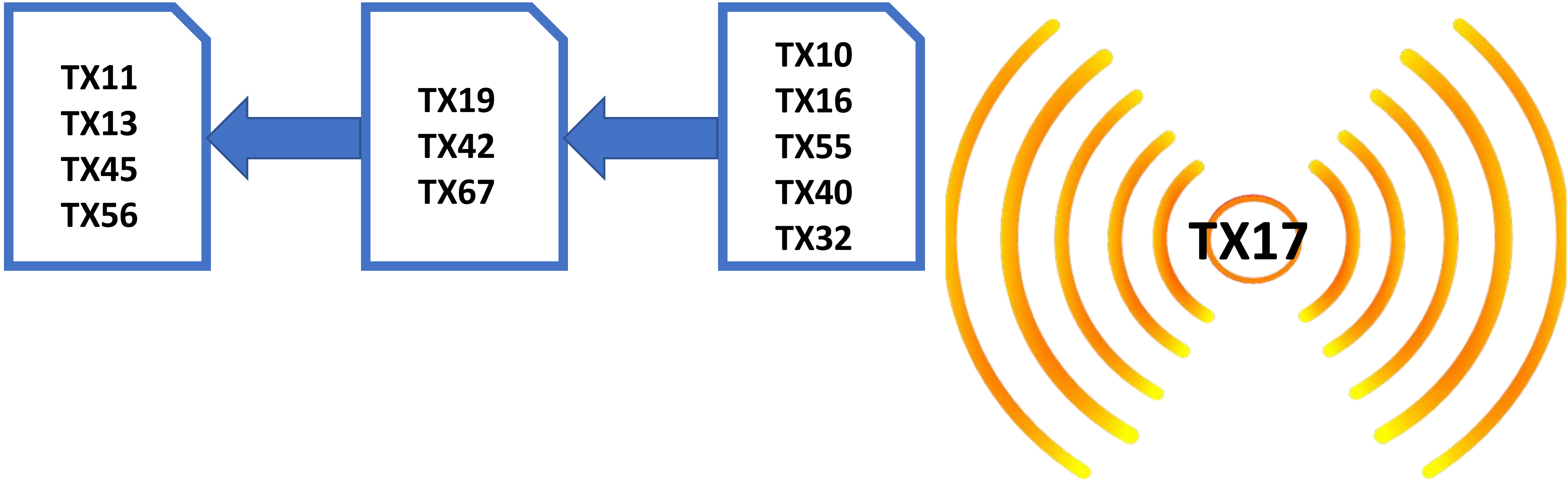
Miner 2

Unconfirmed TX

TX16

Miner 3

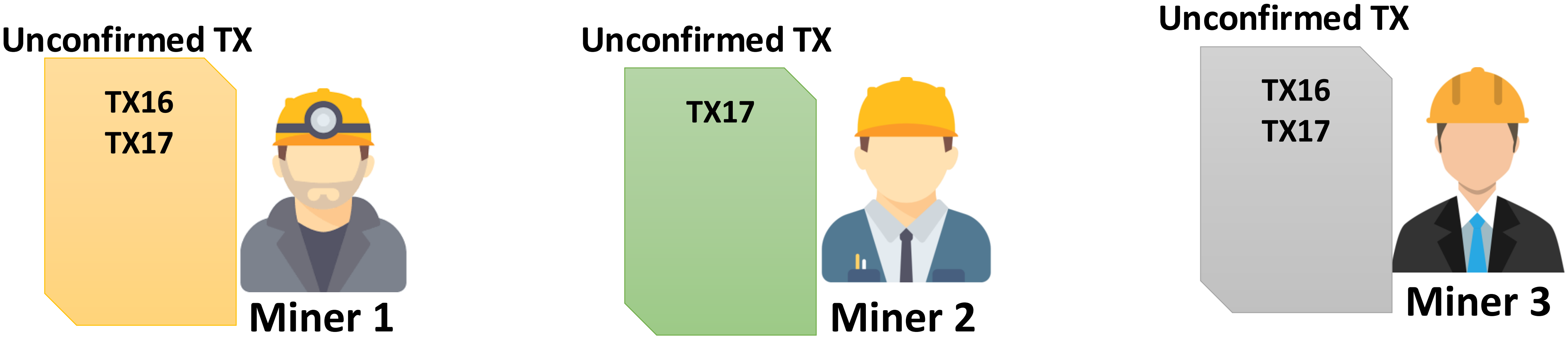
# Breaking the "Safety vs Liveness" Dilemma



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32



Unconfirmed TX

TX16
TX17

Miner 1

Unconfirmed TX

TX17

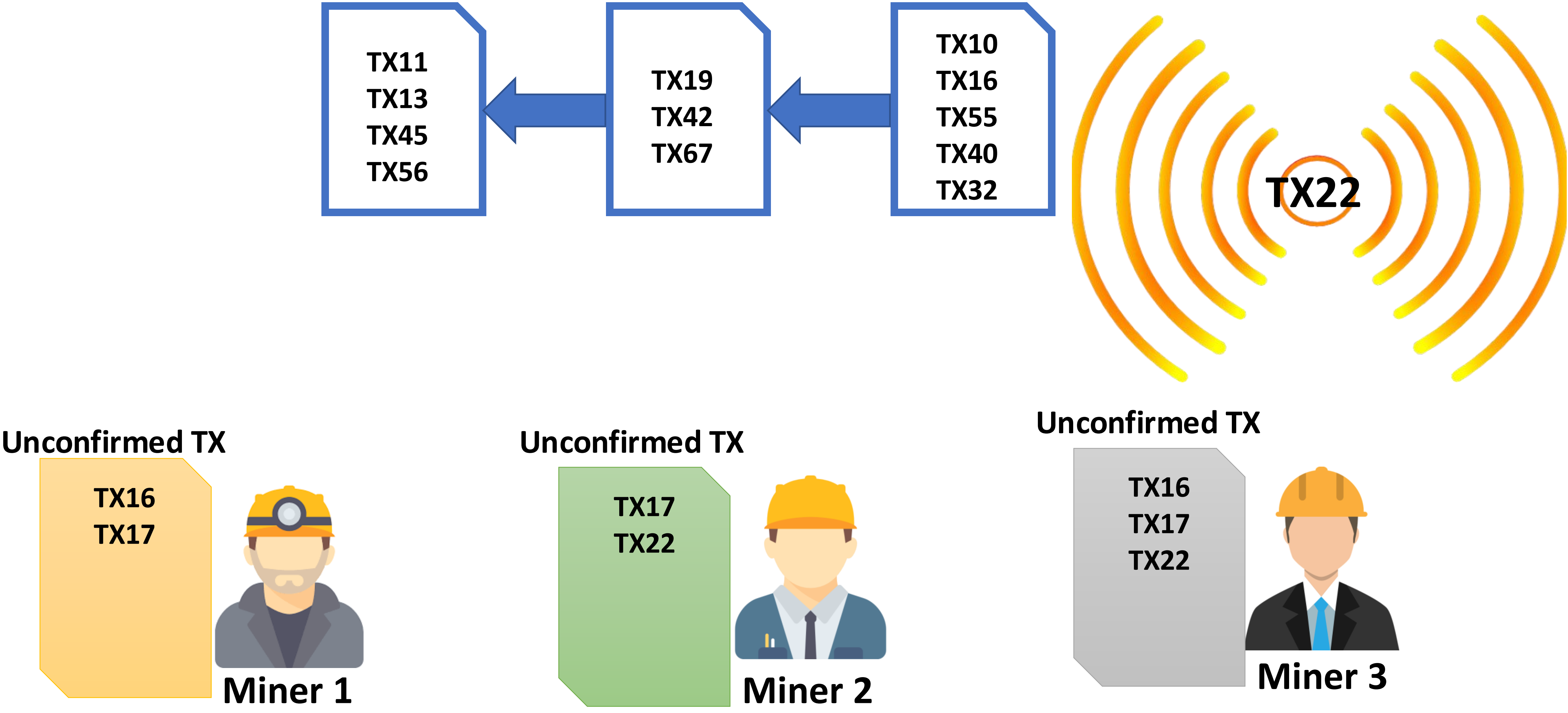
Miner 2

Unconfirmed TX

TX16
TX17

Miner 3

# Breaking the "Safety vs Liveness" Dilemma



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

TX22

Unconfirmed TX

TX16
TX17

Miner 1

Unconfirmed TX

TX17
TX22

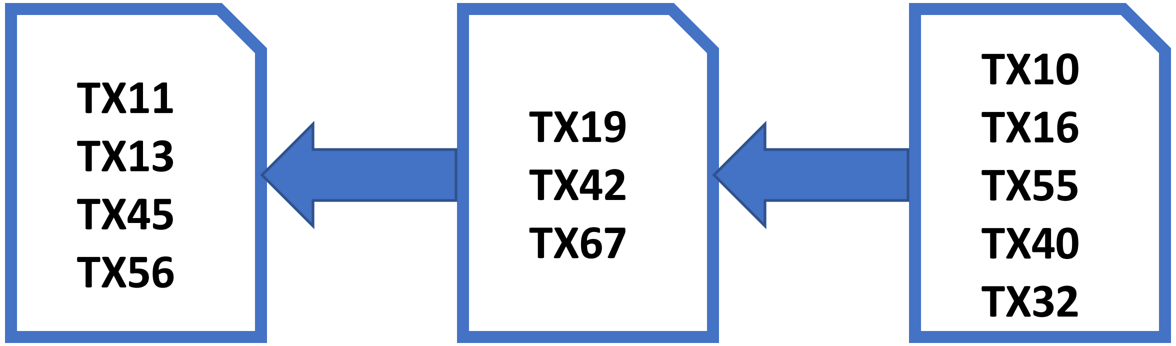
Miner 2

Unconfirmed TX

TX16
TX17
TX22

Miner 3

# Breaking the "Safety vs Liveness" Dilemma



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

Unconfirmed TX

TX16
TX17
TX31
TX87
TX49
TX37

Miner 1

Unconfirmed TX

TX17
TX22
TX49
TX87
TX37
TX38

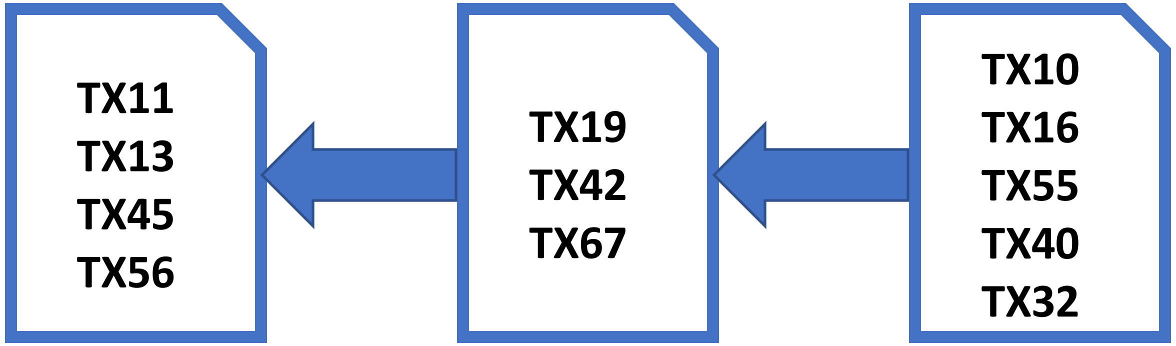
Miner 2

Unconfirmed TX

TX16
TX17
TX22
TX31
TX49
TX87

Miner 3

# Breaking the "Safety vs Liveness" Dilemma



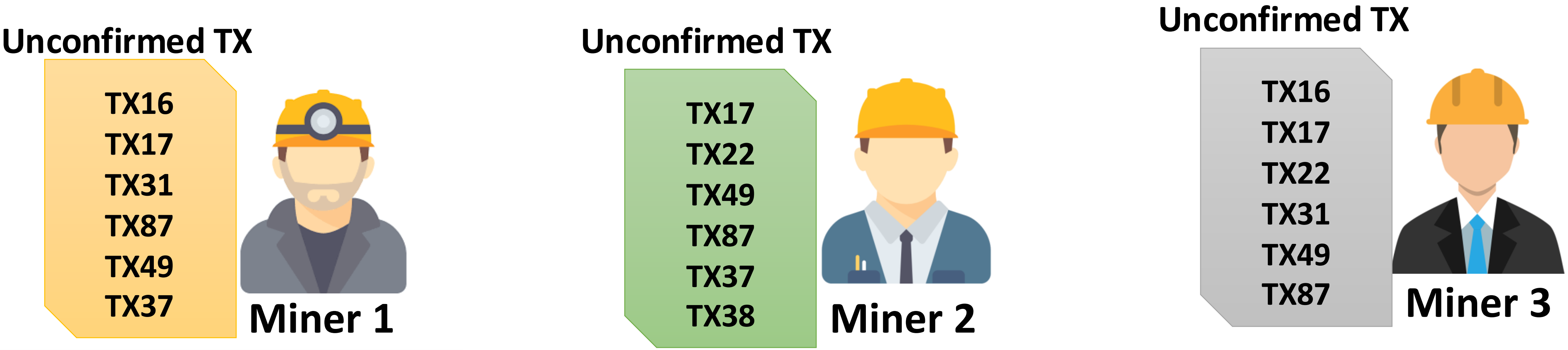
TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

**Safety-1:** The next block should be "correct" in practice

- Transactions are verified, the block contains the correct Hash and Nonce



**Unconfirmed TX**

TX16
TX17
TX31
TX87
TX49
TX37

**Miner 1**

**Unconfirmed TX**

TX17
TX22
TX49
TX87
TX37
TX38

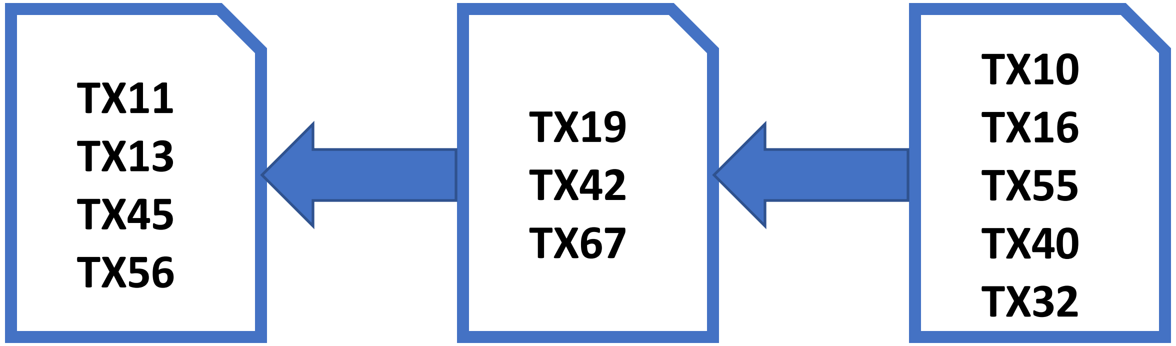
**Miner 2**

**Unconfirmed TX**

TX16
TX17
TX22
TX31
TX49
TX87

**Miner 3**

# Breaking the "Safety vs Liveness" Dilemma



This can be ensured – the block mined by a miner is verified by all

## **Safety-1:** The next block should be "correct" in practice

- Transactions are verified, the block contains the correct Hash and Nonce

**Unconfirmed TX**

TX16
TX17
TX31
TX87
TX49
TX37

**Miner 1**

**Unconfirmed TX**

TX17
TX22
TX49
TX87
TX37
TX38

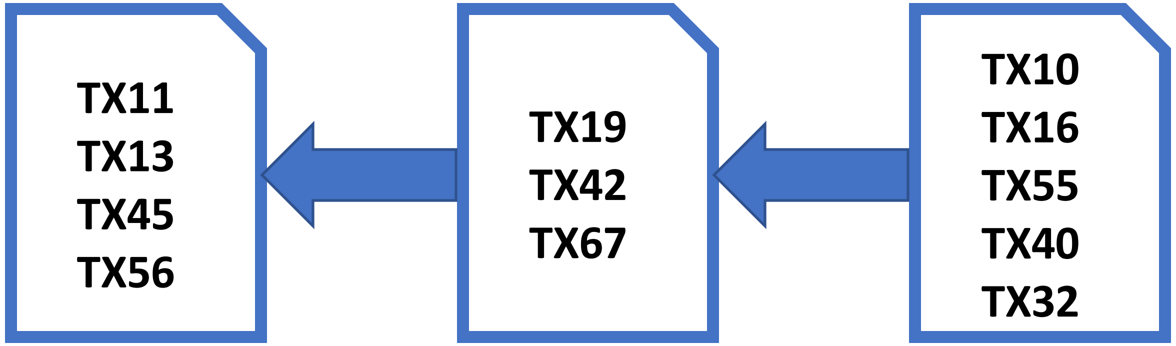
**Miner 2**

**Unconfirmed TX**

TX16
TX17
TX22
TX31
TX49
TX87

**Miner 3**

# Breaking the "Safety vs Liveness" Dilemma



TX11
TX13
TX45
TX56

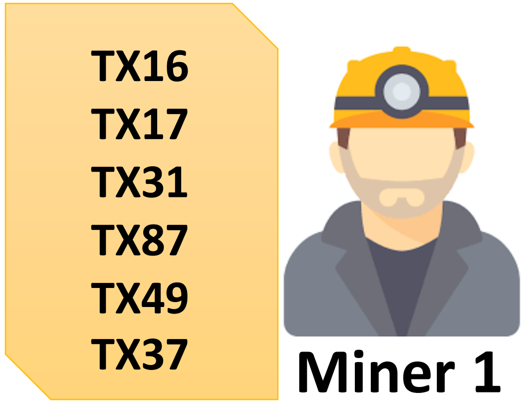
TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

**Safety-2:** All the miners should agree on a single block

- The next block of the blockchain should be selected unanimously

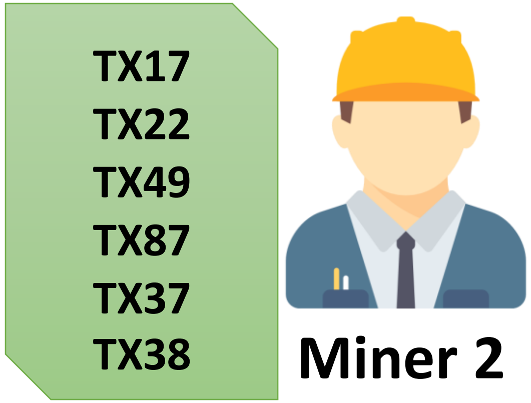
Unconfirmed TX



TX16
TX17
TX31
TX87
TX49
TX37

Miner 1

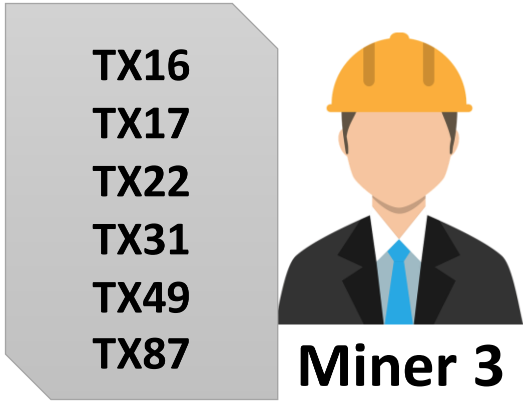
Unconfirmed TX



TX17
TX22
TX49
TX87
TX37
TX38

Miner 2

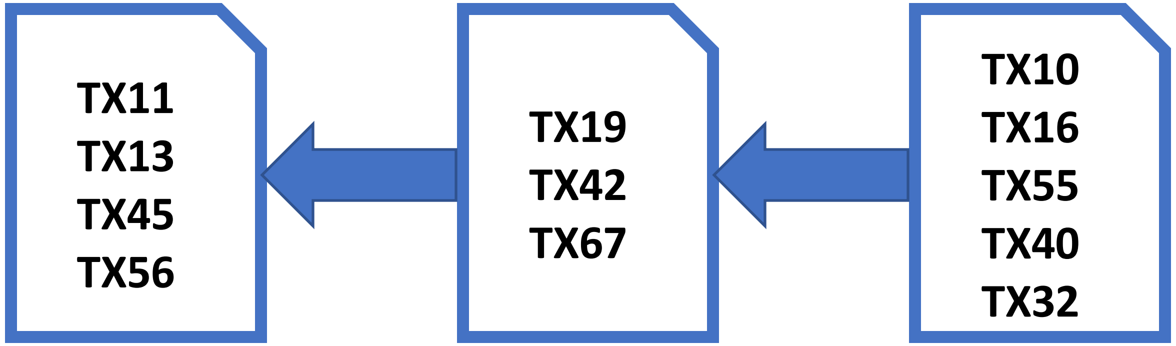
Unconfirmed TX



TX16
TX17
TX22
TX31
TX49
TX87

Miner 3

# Breaking the "Safety vs Liveness" Dilemma



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

Miners do not know each other – how can they agree on the same block?

**Safety-2:** All the miners should agree on a single block

- The next block of the blockchain should be selected unanimously

**Unconfirmed TX**

TX16
TX17
TX31
TX87
TX49
TX37

**Miner 1**

**Unconfirmed TX**

TX17
TX22
TX49
TX87
TX37
TX38

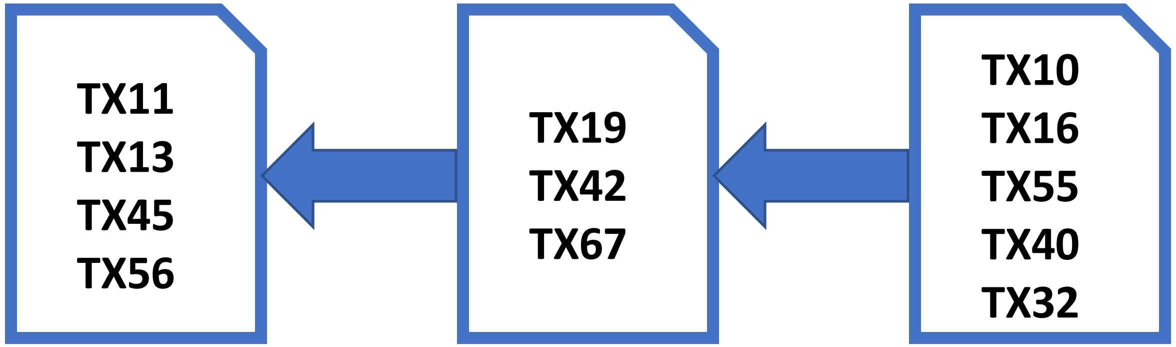
**Miner 2**

**Unconfirmed TX**

TX16
TX17
TX22
TX31
TX49
TX87

**Miner 3**

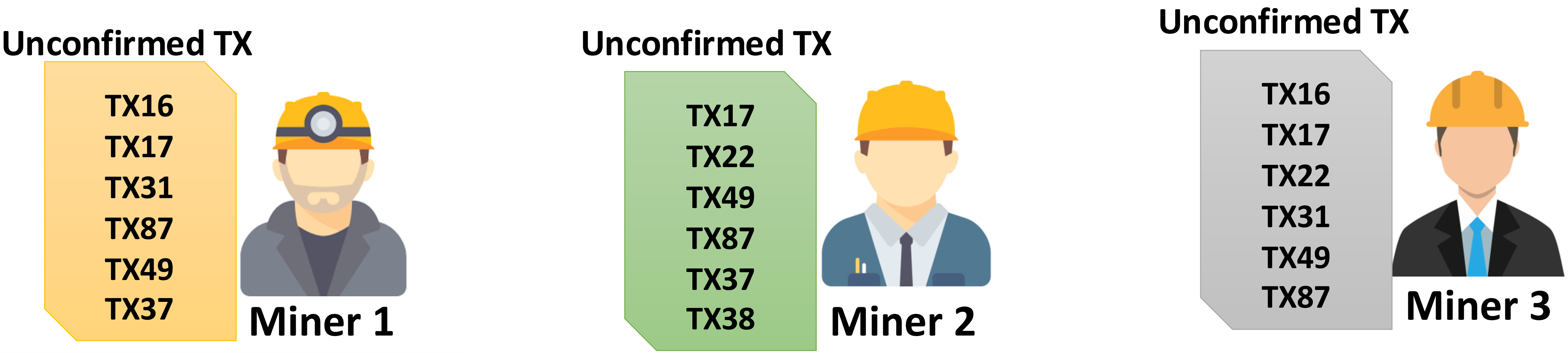
# Breaking the "Safety vs Liveness" Dilemma



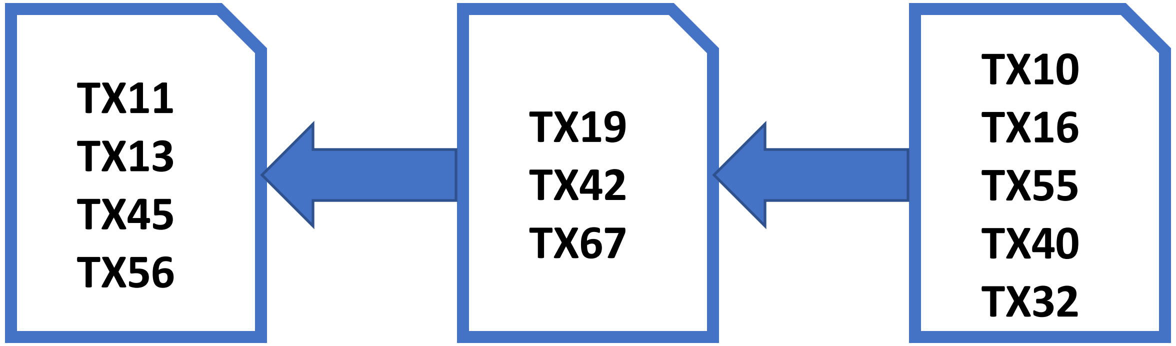
We compromise here!

**Safety-2:** All the miners should agree on a single block

- The next block of the blockchain should be selected unanimously



# Breaking the "Safety vs Liveness" Dilemma



**Liveness:** Add a block as long as it is correct (contains valid transactions from the unconfirmed TX list) and move further

**Unconfirmed TX**

TX16
TX17
TX31
TX87
TX49
TX37

**Miner 1**

**Unconfirmed TX**

TX17
TX22
TX49
TX87
TX37
TX38

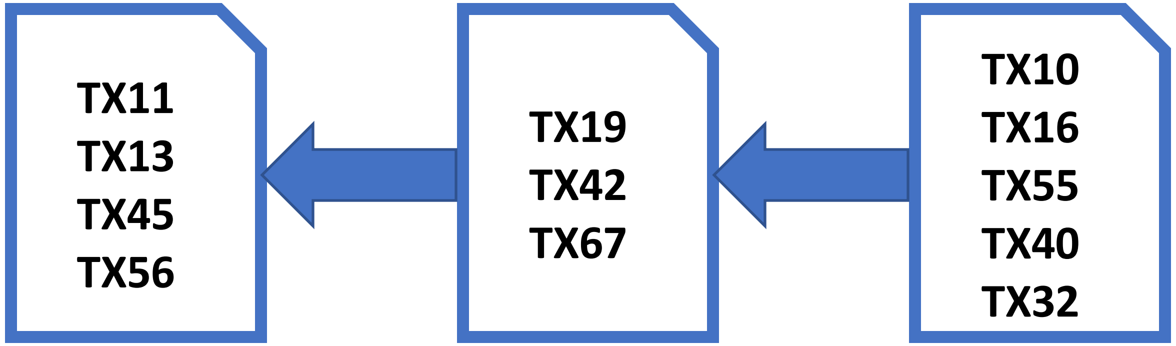
**Miner 2**

**Unconfirmed TX**

TX16
TX17
TX22
TX31
TX49
TX87

**Miner 3**

# Breaking the "Safety vs Liveness" Dilemma



TX11<br>TX13<br>TX45<br>TX56

TX19<br>TX42<br>TX67

TX10<br>TX16<br>TX55<br>TX40<br>TX32

Two (or more) different miners may add two (or more) different blocks

**Liveness:** Add a block as long as it is correct (contains valid transactions from the unconfirmed TX list) and move further

**Unconfirmed TX**

TX16
TX17
TX31
TX87
TX49
TX37

**Miner 1**

**Unconfirmed TX**

TX17
TX22
TX49
TX87
TX37
TX38

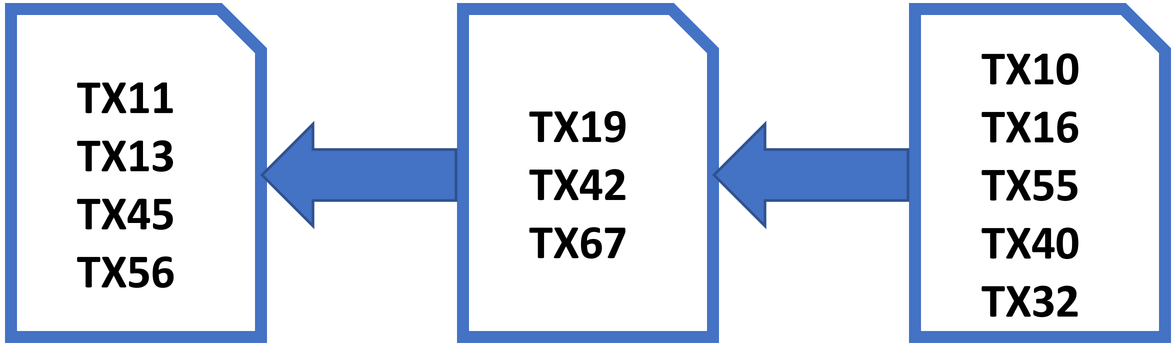
**Miner 2**

**Unconfirmed TX**

TX16
TX17
TX22
TX31
TX49
TX87

**Miner 3**

# Breaking the "Safety vs Liveness" Dilemma



TX11
TX13
TX45
TX56

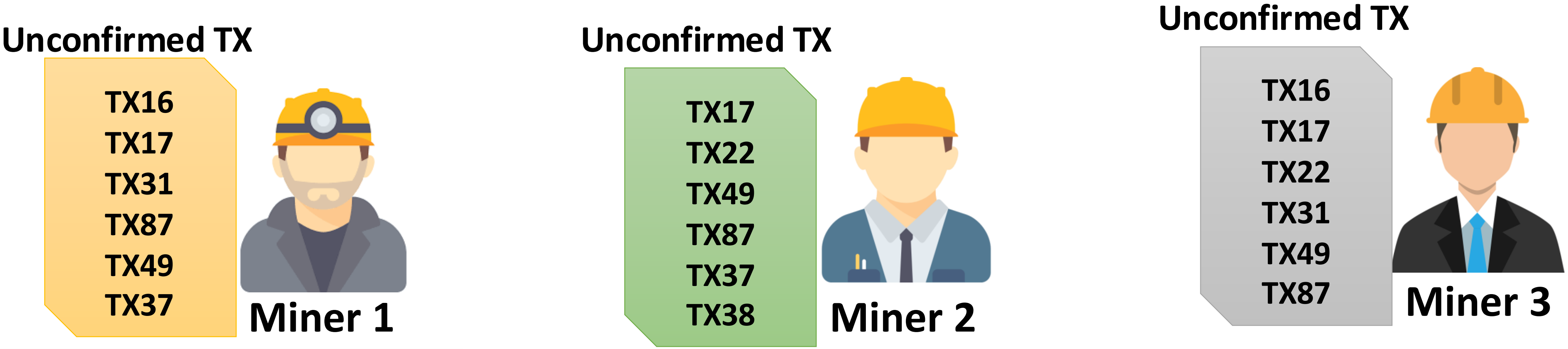
TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

Two (or more) different miners may add two (or more) different blocks

Will resolve this later!

**Liveness:** Add a block as long as it is correct (contains valid transactions from the unconfirmed TX list) and move further



**Unconfirmed TX**

TX16
TX17
TX31
TX87
TX49
TX37

**Miner 1**

**Unconfirmed TX**

TX17
TX22
TX49
TX87
TX37
TX38

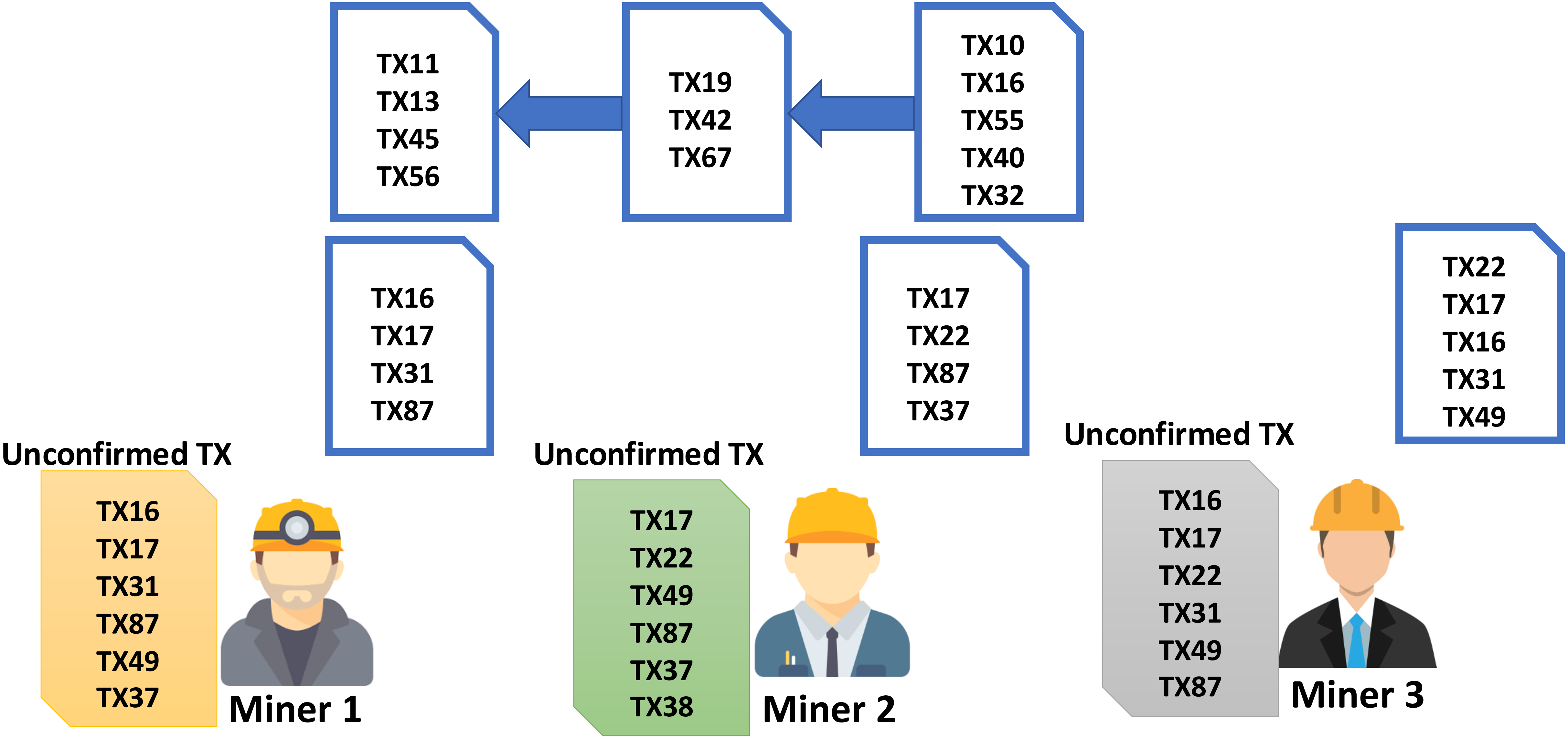
**Miner 2**

**Unconfirmed TX**

TX16
TX17
TX22
TX31
TX49
TX87

**Miner 3**

# Nakamoto Consensus (Proof of Work)



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

TX16
TX17
TX31
TX87

TX17
TX22
TX87
TX37

TX22
TX17
TX16
TX31
TX49

Unconfirmed TX

TX16
TX17
TX31
TX87
TX49
TX37

Miner 1

Unconfirmed TX

TX17
TX22
TX49
TX87
TX37
TX38

Miner 2

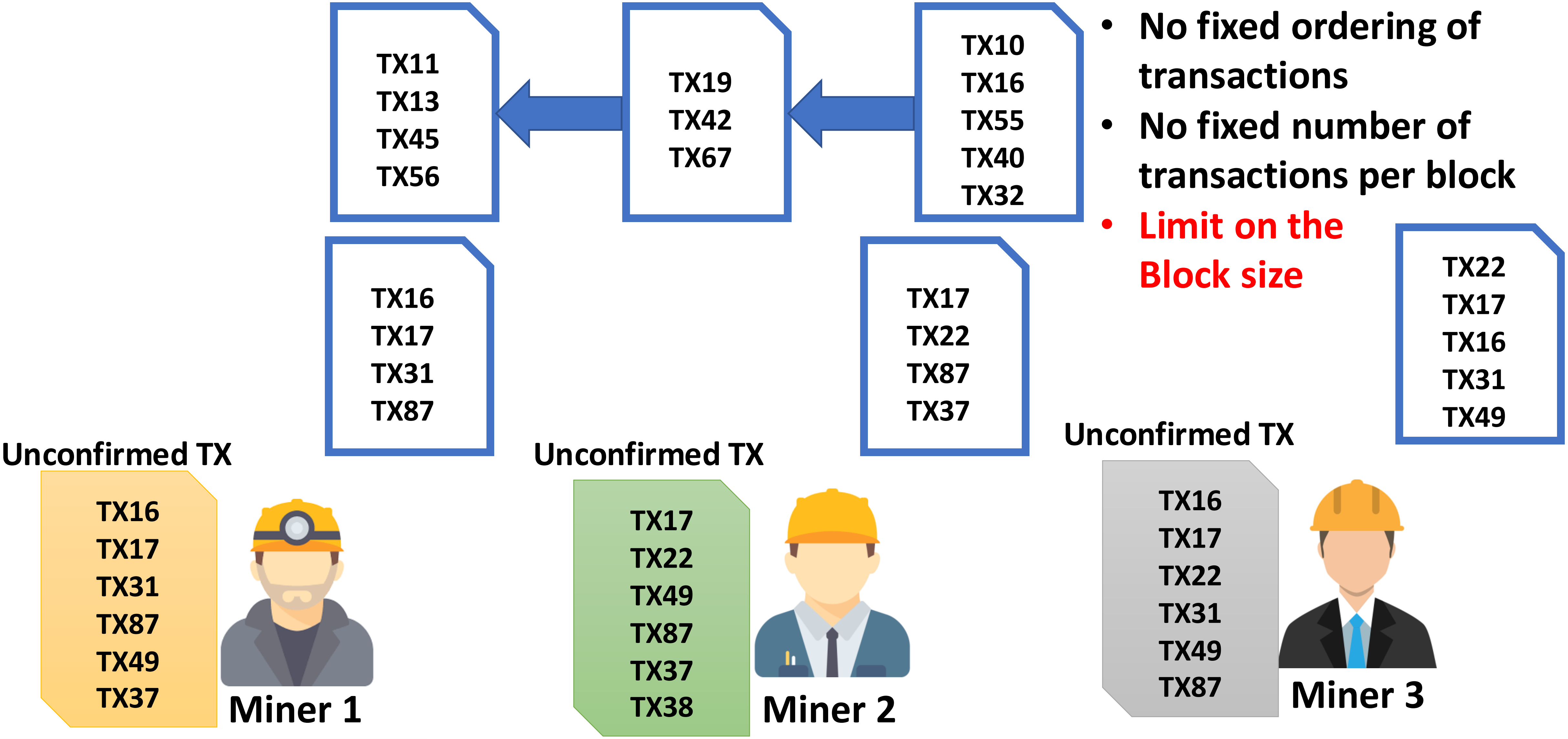
Unconfirmed TX

TX16
TX17
TX22
TX31
TX49
TX87

Miner 3

# Nakamoto Consensus (Proof of Work)

- No fixed ordering of transactions
- No fixed number of transactions per block
- Limit on the Block size



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

TX16
TX17
TX31
TX87

TX17
TX22
TX87
TX37

TX22
TX17
TX16
TX31
TX49

Unconfirmed TX

TX16
TX17
TX31
TX87
TX49
TX37

Miner 1

Unconfirmed TX

TX17
TX22
TX49
TX87
TX37
TX38

Miner 2

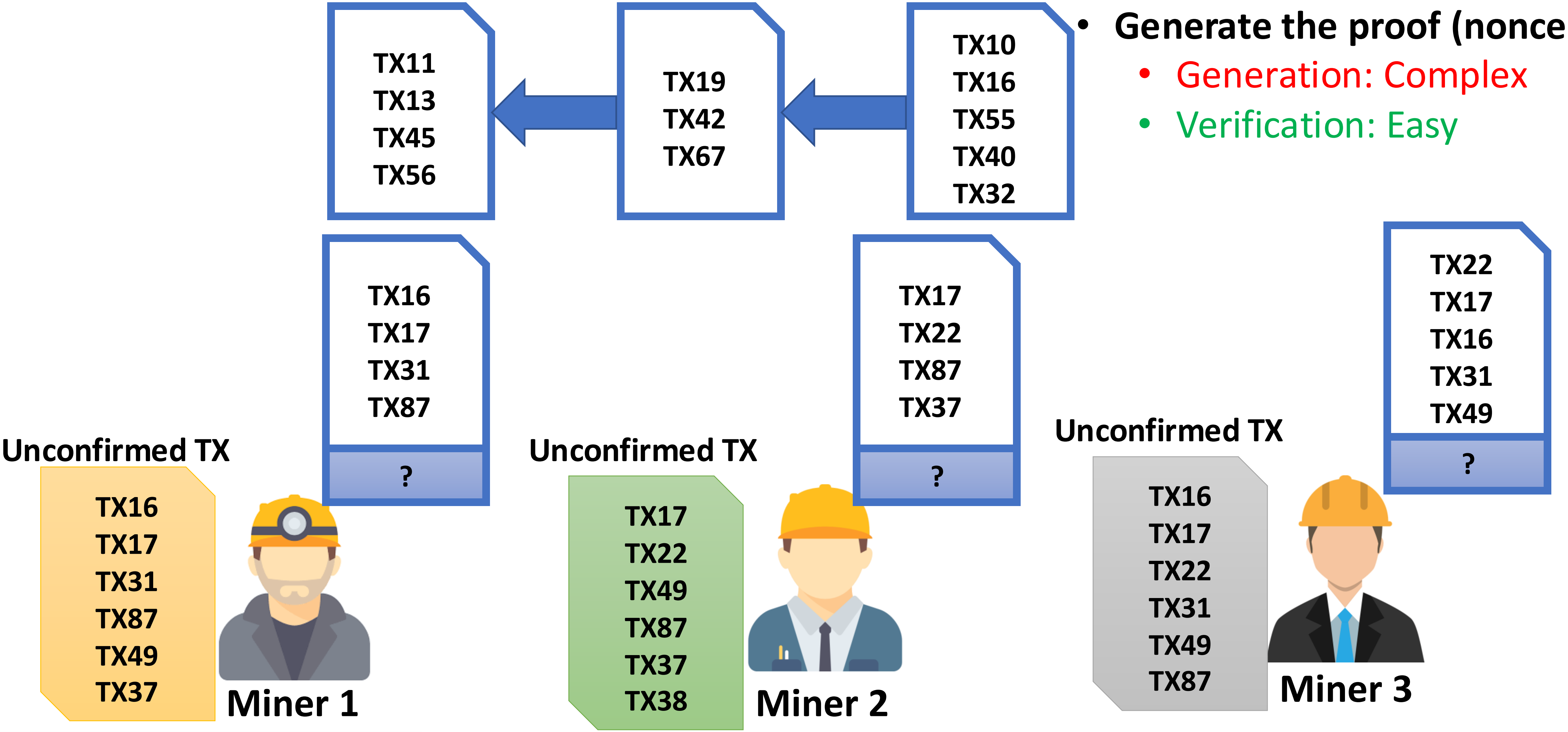
Unconfirmed TX

TX16
TX17
TX22
TX31
TX49
TX87

Miner 3

# Nakamoto Consensus (Proof of Work)

- **Generate the proof (nonce)**
  - Generation: Complex
  - Verification: Easy



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

TX16
TX17
TX31
TX87
?

TX17
TX22
TX87
TX37
?

TX22
TX17
TX16
TX31
TX49
?

**Unconfirmed TX**

TX16
TX17
TX31
TX87
TX49
TX37

**Miner 1**

**Unconfirmed TX**

TX17
TX22
TX49
TX87
TX37
TX38

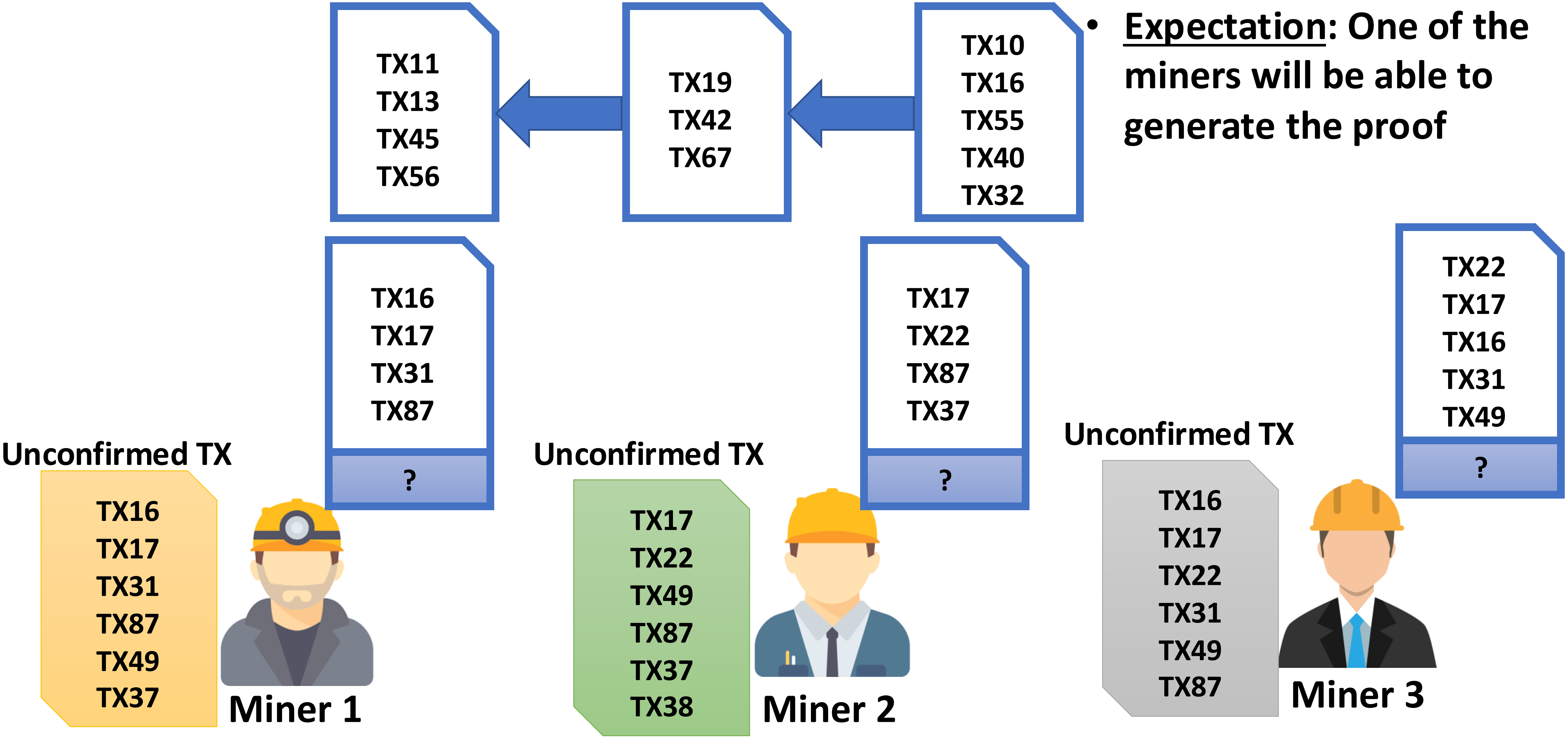
**Miner 2**

**Unconfirmed TX**

TX16
TX17
TX22
TX31
TX49
TX87

**Miner 3**

# Nakamoto Consensus (Proof of Work)



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

TX16
TX17
TX31
TX87

?

Unconfirmed TX

TX16
TX17
TX31
TX87
TX49
TX37

Miner 1

TX17
TX22
TX87
TX37

?

Unconfirmed TX

TX17
TX22
TX49
TX87
TX37
TX38

Miner 2

TX22
TX17
TX16
TX31
TX49

?

Unconfirmed TX

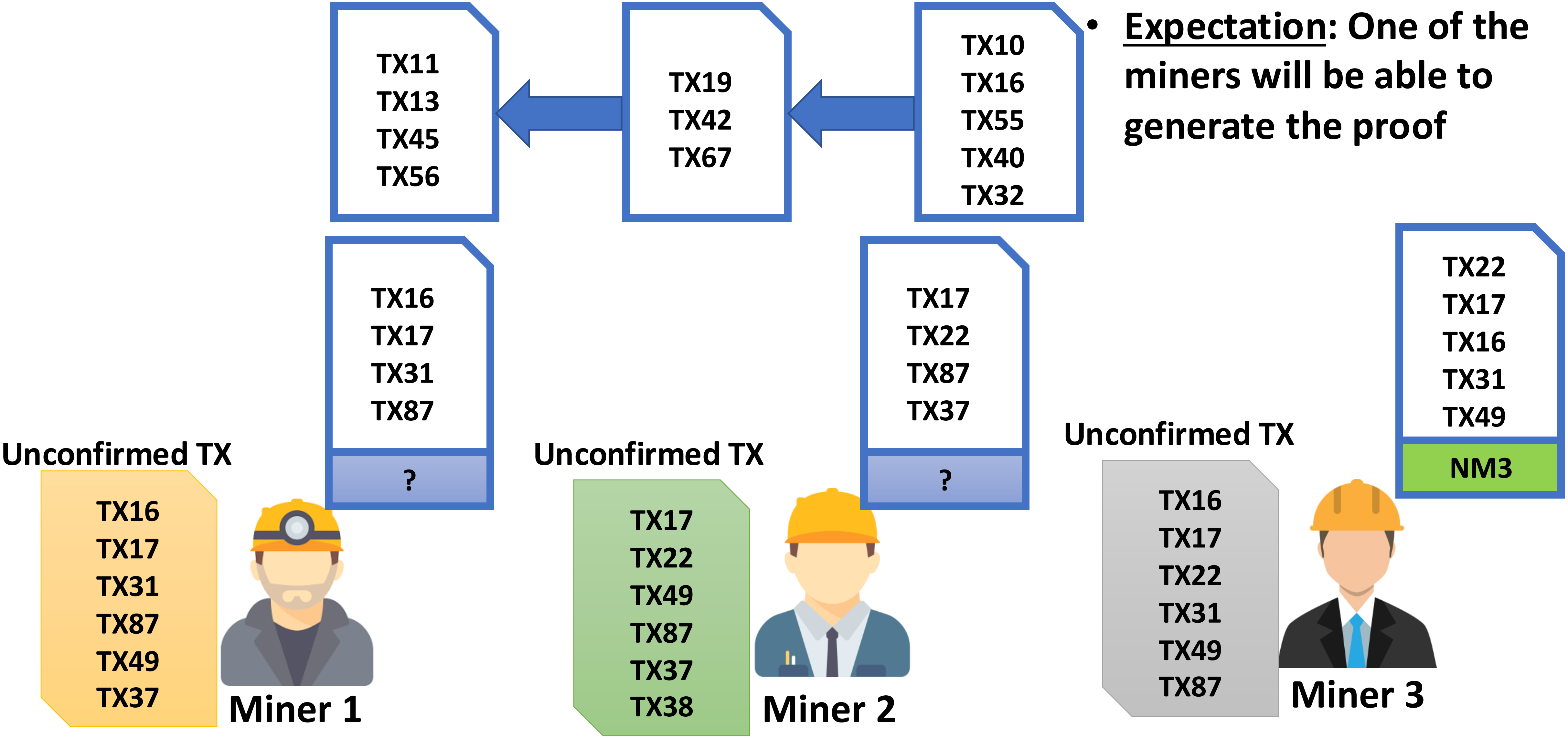
TX16
TX17
TX22
TX31
TX49
TX87

Miner 3

- **Expectation**: One of the miners will be able to generate the proof

# Nakamoto Consensus (Proof of Work)

- **Expectation**: One of the miners will be able to generate the proof



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

Unconfirmed TX

TX16
TX17
TX31
TX87
TX49
TX37

TX16
TX17
TX31
TX87
?

Miner 1

Unconfirmed TX

TX17
TX22
TX49
TX87
TX37
TX38

TX17
TX22
TX87
TX37
?

Miner 2

Unconfirmed TX

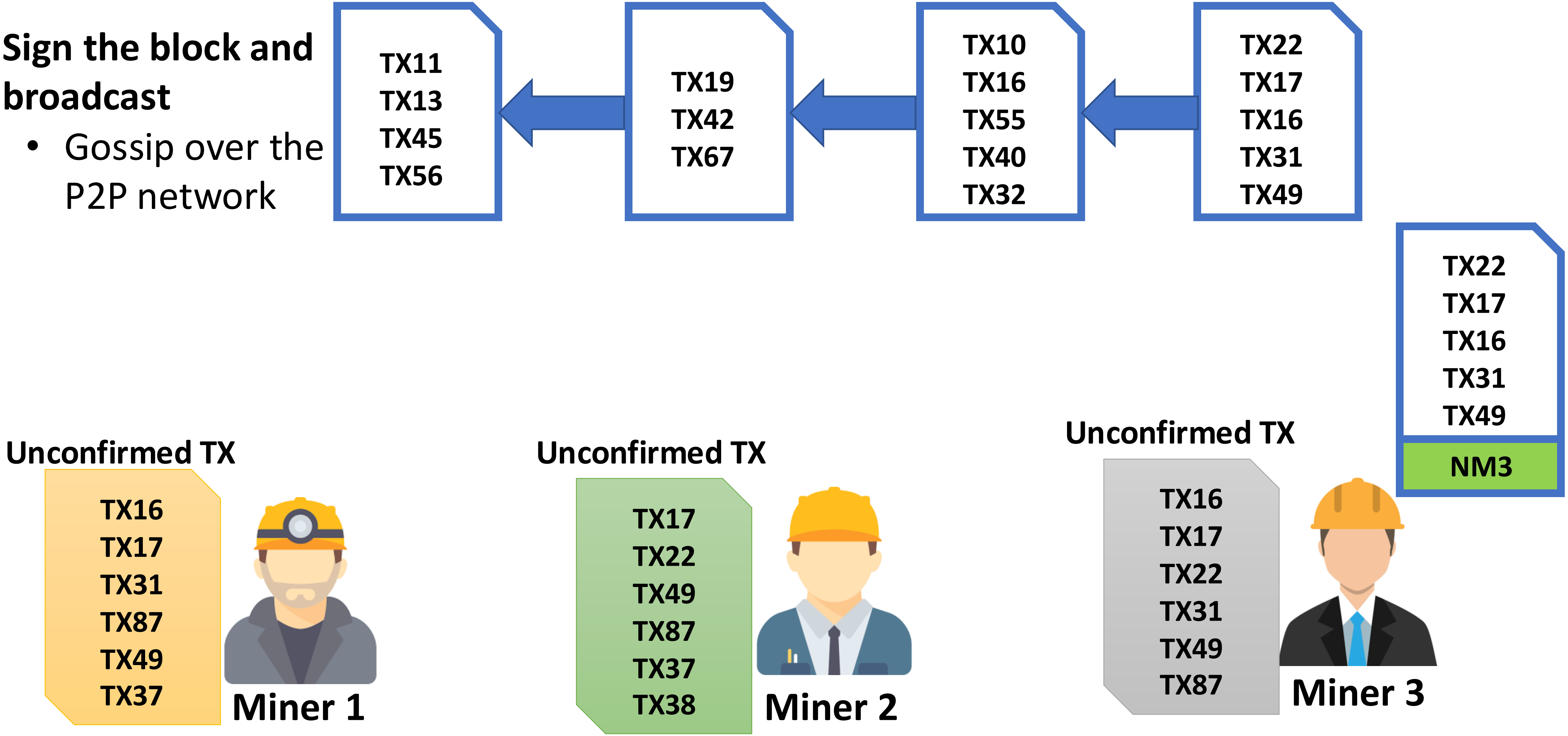
TX16
TX17
TX22
TX31
TX49
TX87

TX22
TX17
TX16
TX31
TX49
NM3

Miner 3

# Nakamoto Consensus (Proof of Work)

- **Sign the block and broadcast**
    - Gossip over the P2P network



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

TX22
TX17
TX16
TX31
TX49

TX22
TX17
TX16
TX31
TX49
NM3

**Unconfirmed TX**

TX16
TX17
TX31
TX87
TX49
TX37

**Miner 1**

**Unconfirmed TX**

TX17
TX22
TX49
TX87
TX37
TX38

**Miner 2**

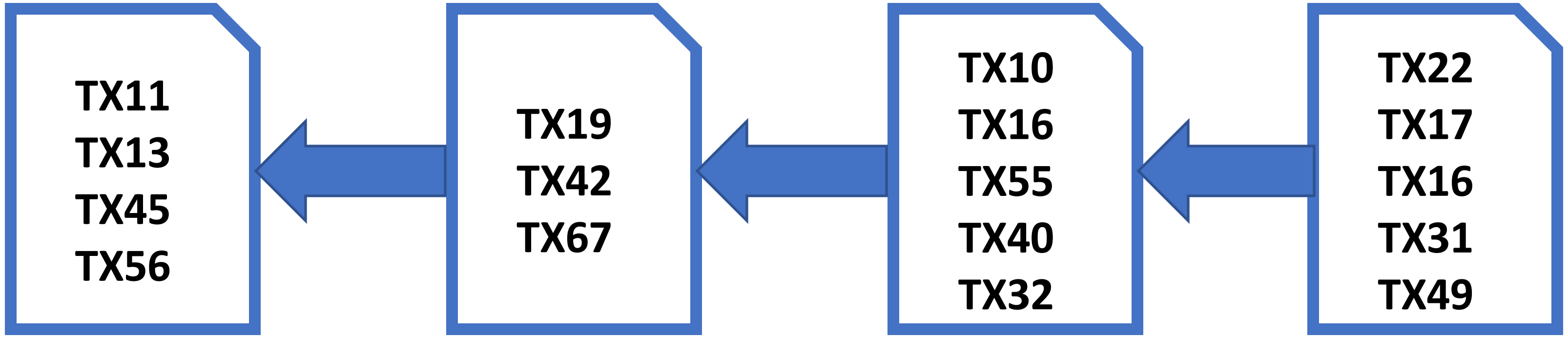
**Unconfirmed TX**

TX16
TX17
TX22
TX31
TX49
TX87

**Miner 3**

# Nakamoto Consensus (Proof of Work)

- Remove the transactions from unconfirmed TX list

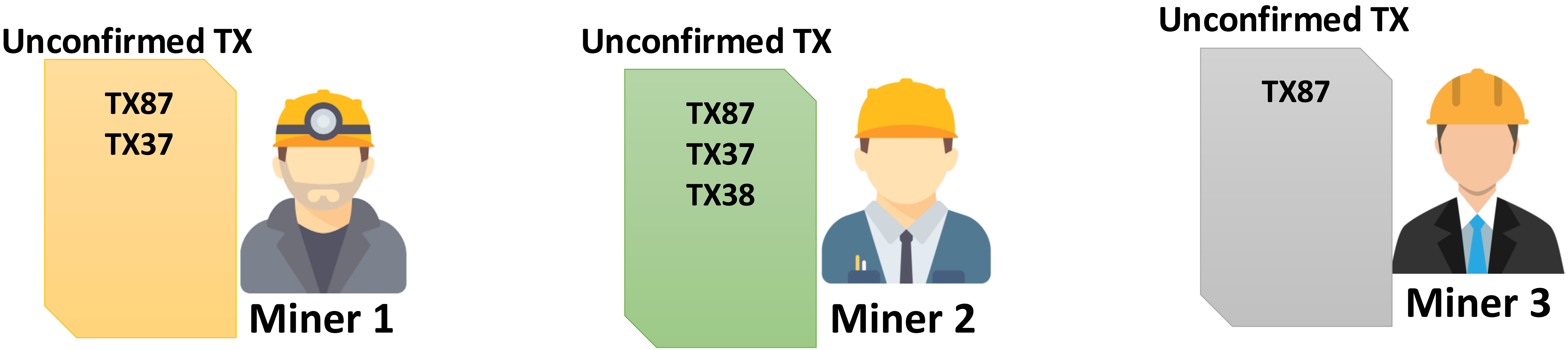


TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

TX22
TX17
TX16
TX31
TX49



Unconfirmed TX

TX87
TX37

Unconfirmed TX

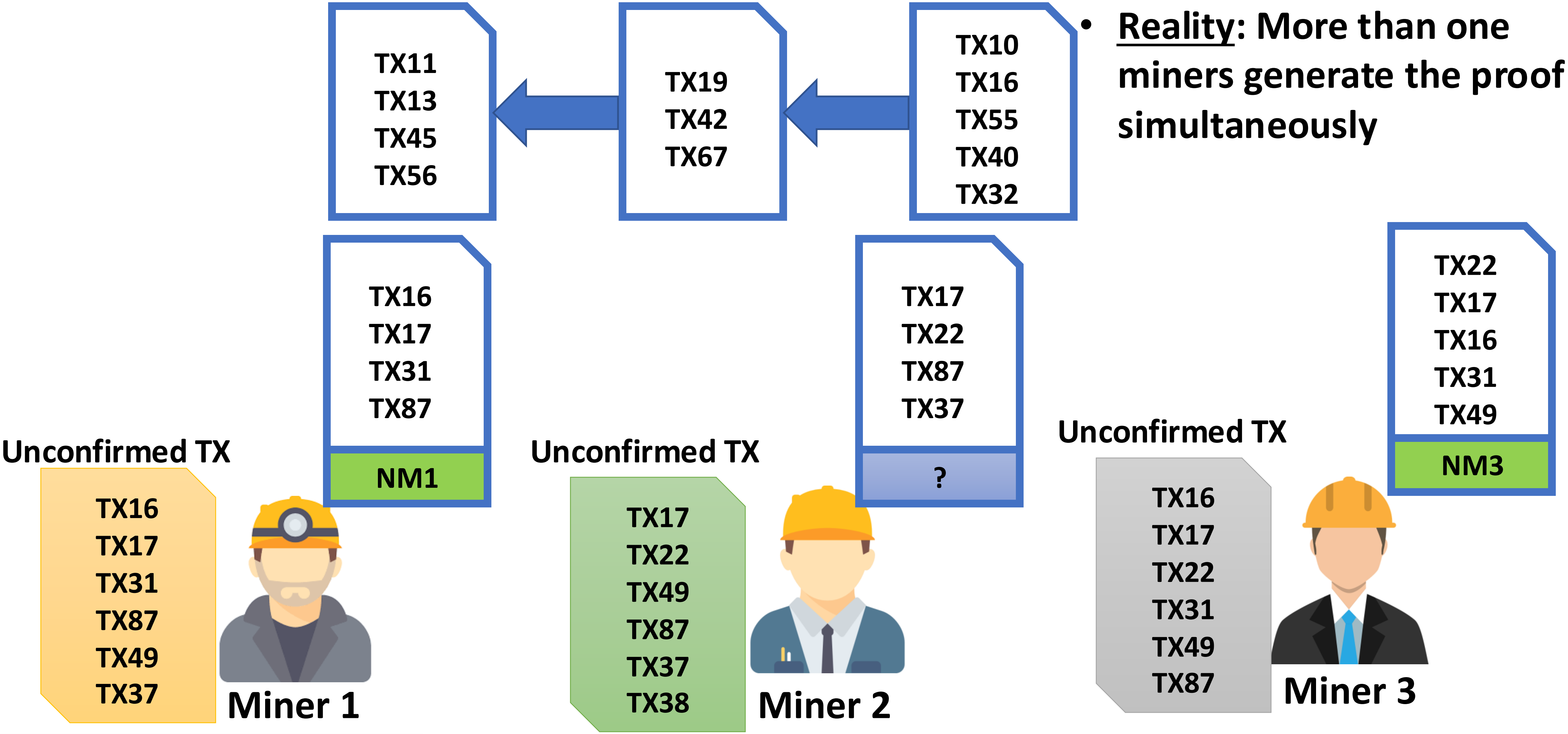
TX87
TX37
TX38

Unconfirmed TX

TX87

# Nakamoto Consensus (Proof of Work)

- **Reality**: More than one miners generate the proof simultaneously



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

TX16
TX17
TX31
TX87
NM1

TX17
TX22
TX87
TX37
?

TX22
TX17
TX16
TX31
TX49
NM3

Unconfirmed TX
TX16
TX17
TX31
TX87
TX49
TX37

Miner 1

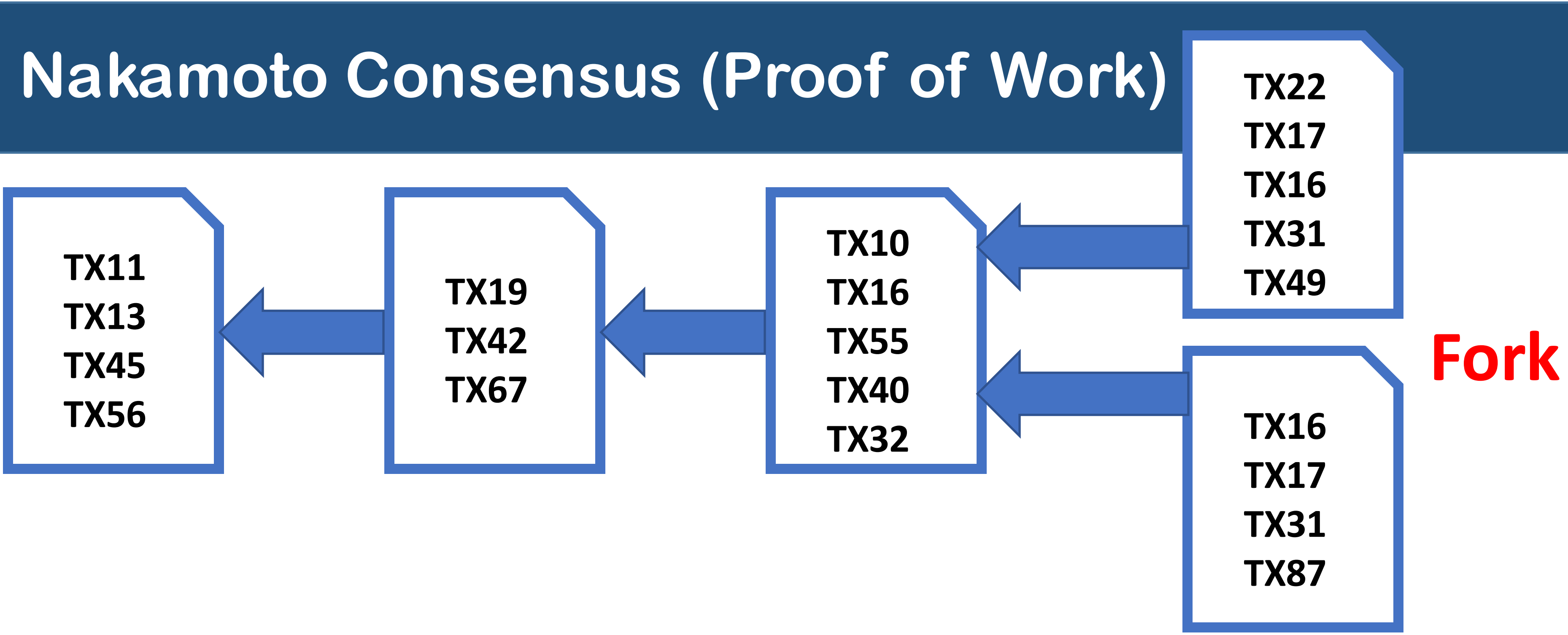
Unconfirmed TX
TX17
TX22
TX49
TX87
TX37
TX38

Miner 2

Unconfirmed TX
TX16
TX17
TX22
TX31
TX49
TX87

Miner 3

# Nakamoto Consensus (Proof of Work)



**Unconfirmed TX**

TX16
TX17
TX31
TX87
TX49
TX37

**Miner 1**

**Unconfirmed TX**

TX17
TX22
TX49
TX87
TX37
TX38

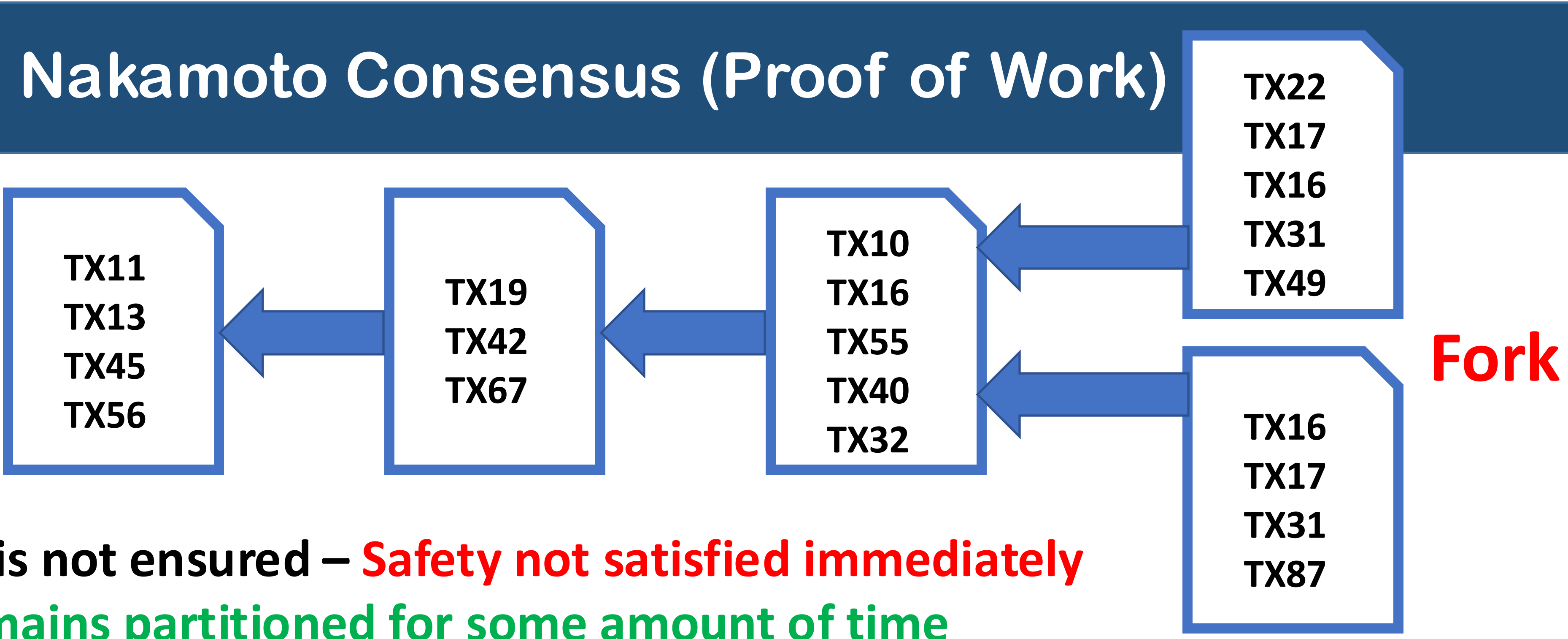
**Miner 2**

**Unconfirmed TX**

TX16
TX17
TX22
TX31
TX49
TX87

**Miner 3**

# Nakamoto Consensus (Proof of Work)



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

TX22
TX17
TX16
TX31
TX49

TX16
TX17
TX31
TX87

Fork

**Consensus finality is not ensured – Safety not satisfied immediately**

- **The network remains partitioned for some amount of time**

**Unconfirmed TX**

TX16
TX17
TX31
TX87
TX49
TX37

**Miner 1**

**Unconfirmed TX**

TX17
TX22
TX49
TX87
TX37
TX38

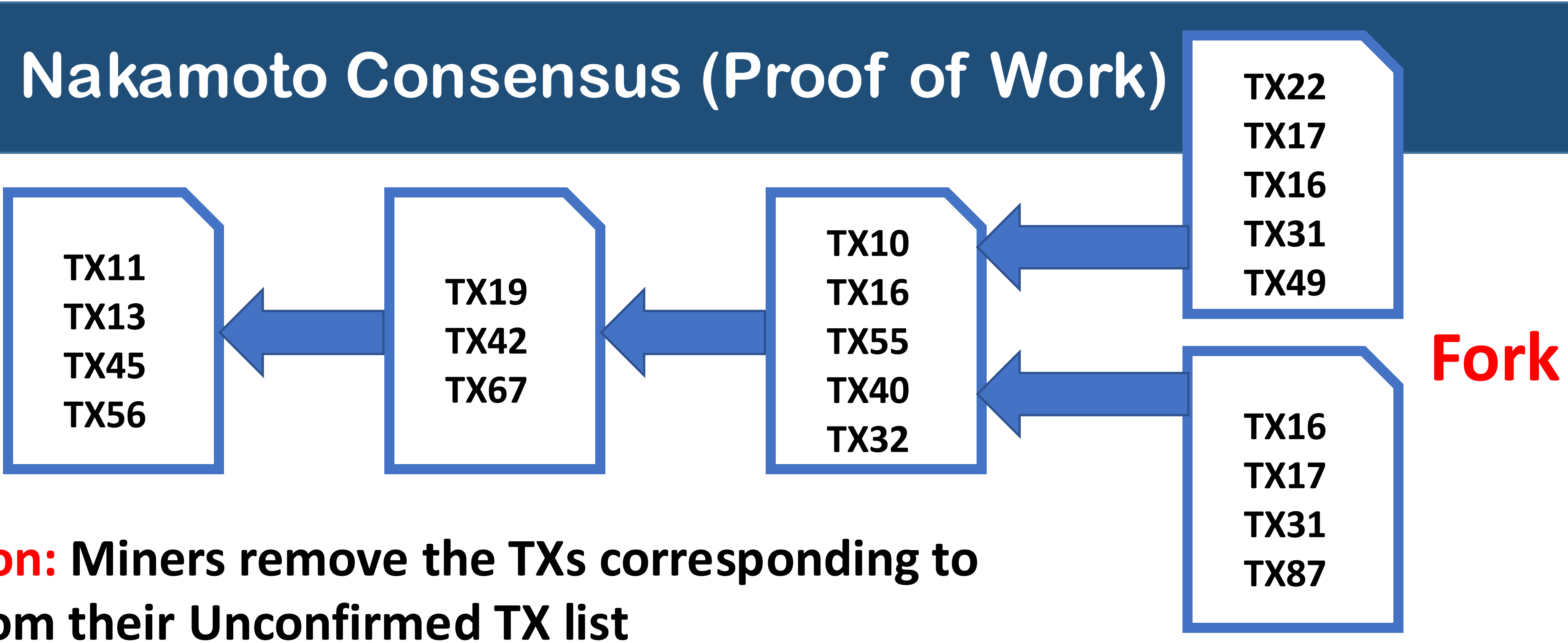
**Miner 2**

**Unconfirmed TX**

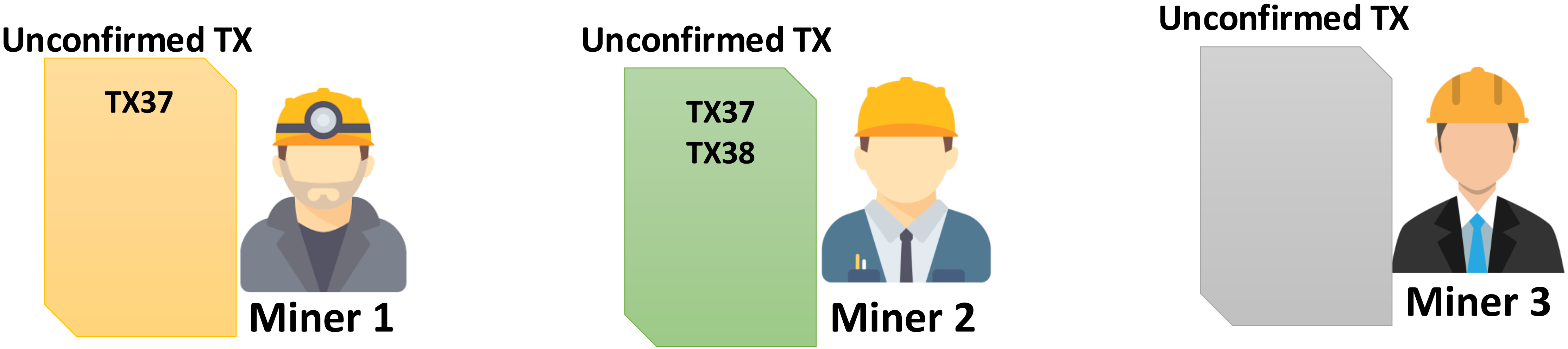
TX16
TX17
TX22
TX31
TX49
TX87

**Miner 3**

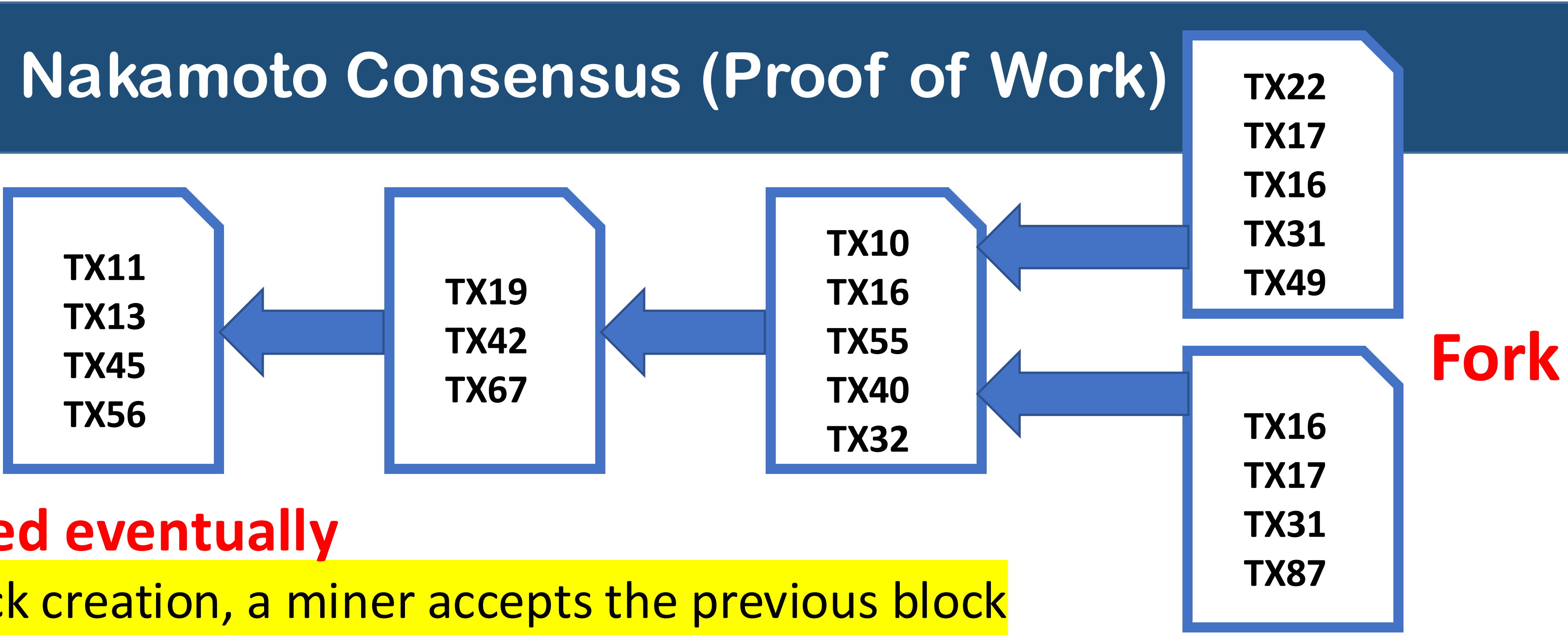
# Nakamoto Consensus (Proof of Work)



**Momentary Decision:** Miners remove the TXs corresponding to both the blocks, from their Unconfirmed TX list

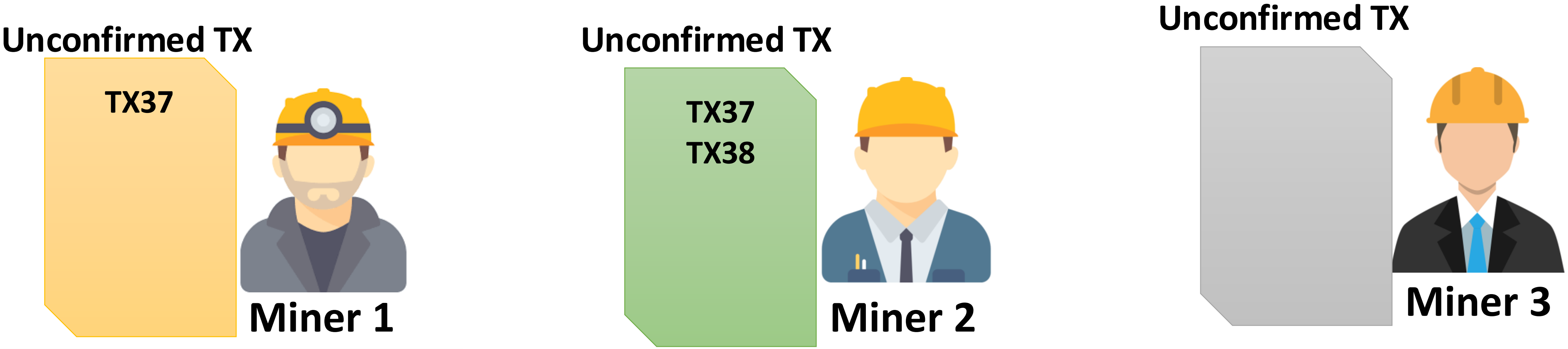


# Nakamoto Consensus (Proof of Work)

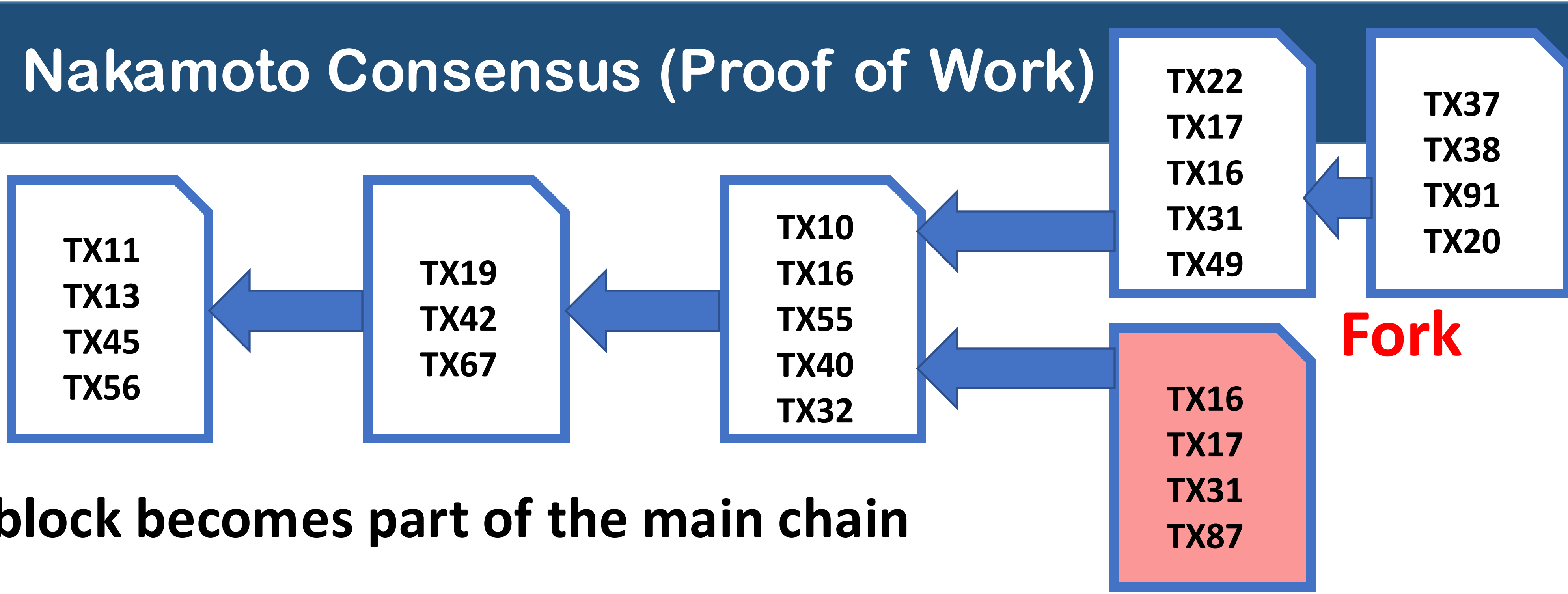


**Forks are resolved eventually**

- For the next block creation, a miner accepts the previous block that it hears from the majority of the neighbor



# Nakamoto Consensus (Proof of Work)



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

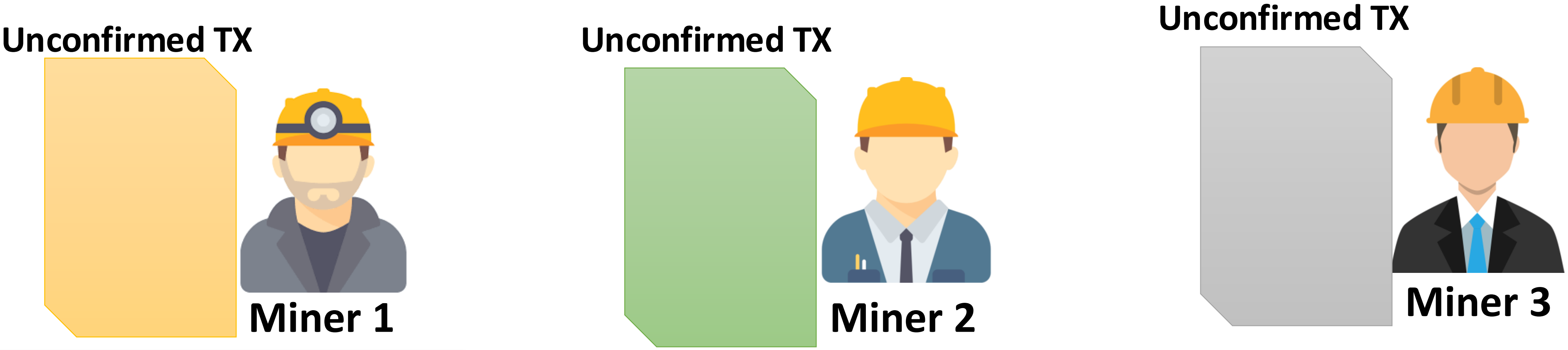
TX22
TX17
TX16
TX31
TX49

TX37
TX38
TX91
TX20

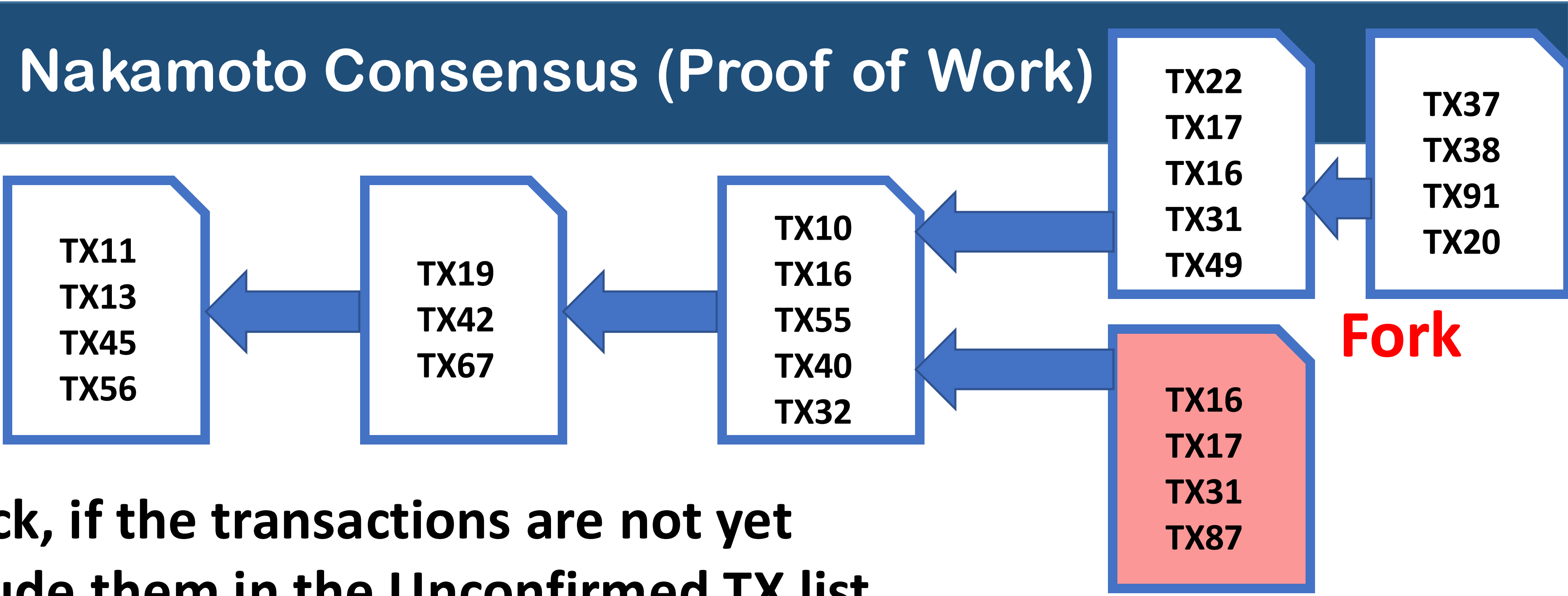
TX16
TX17
TX31
TX87

Fork

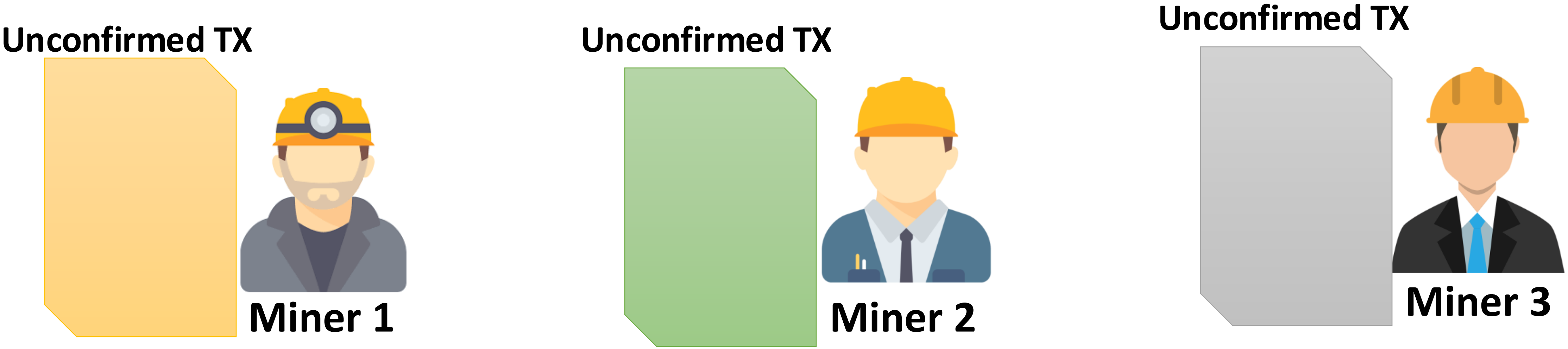
Eventually, one block becomes part of the main chain



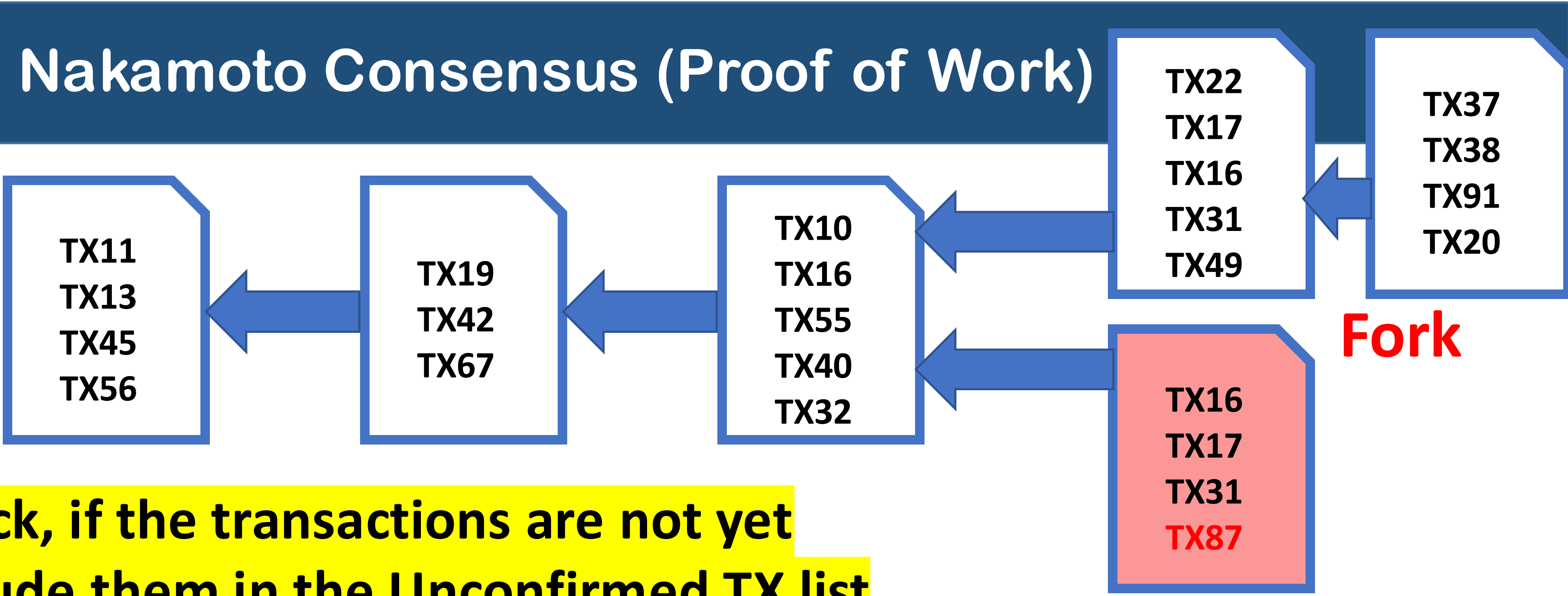
# Nakamoto Consensus (Proof of Work)



For a forked block, if the transactions are not yet committed, include them in the Unconfirmed TX list



# Nakamoto Consensus (Proof of Work)



TX11
TX13
TX45
TX56

TX19
TX42
TX67

TX10
TX16
TX55
TX40
TX32

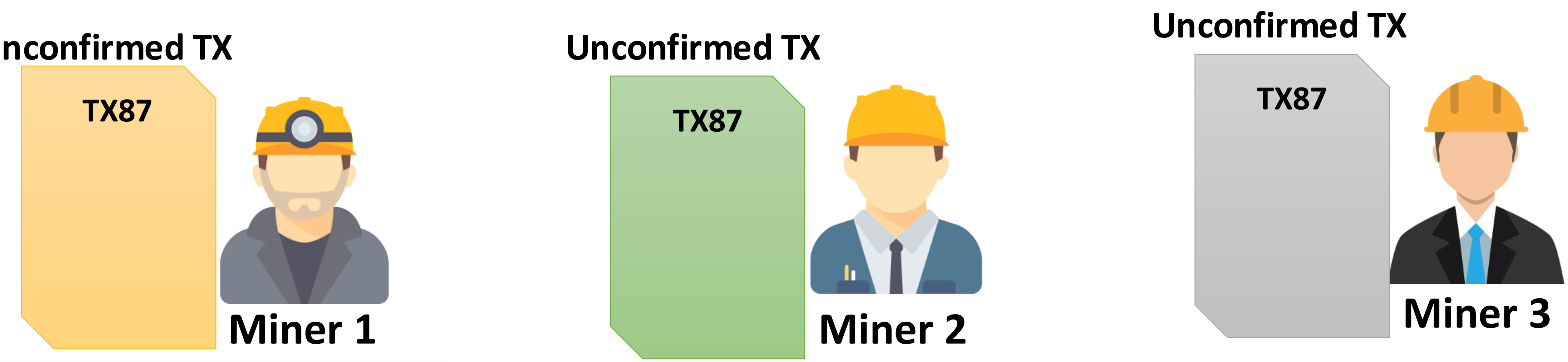
TX22
TX17
TX16
TX31
TX49

TX37
TX38
TX91
TX20

Fork

TX16
TX17
TX31
TX87

**For a forked block, if the transactions are not yet committed, include them in the Unconfirmed TX list**



Unconfirmed TX

TX87

Miner 1

Unconfirmed TX

TX87

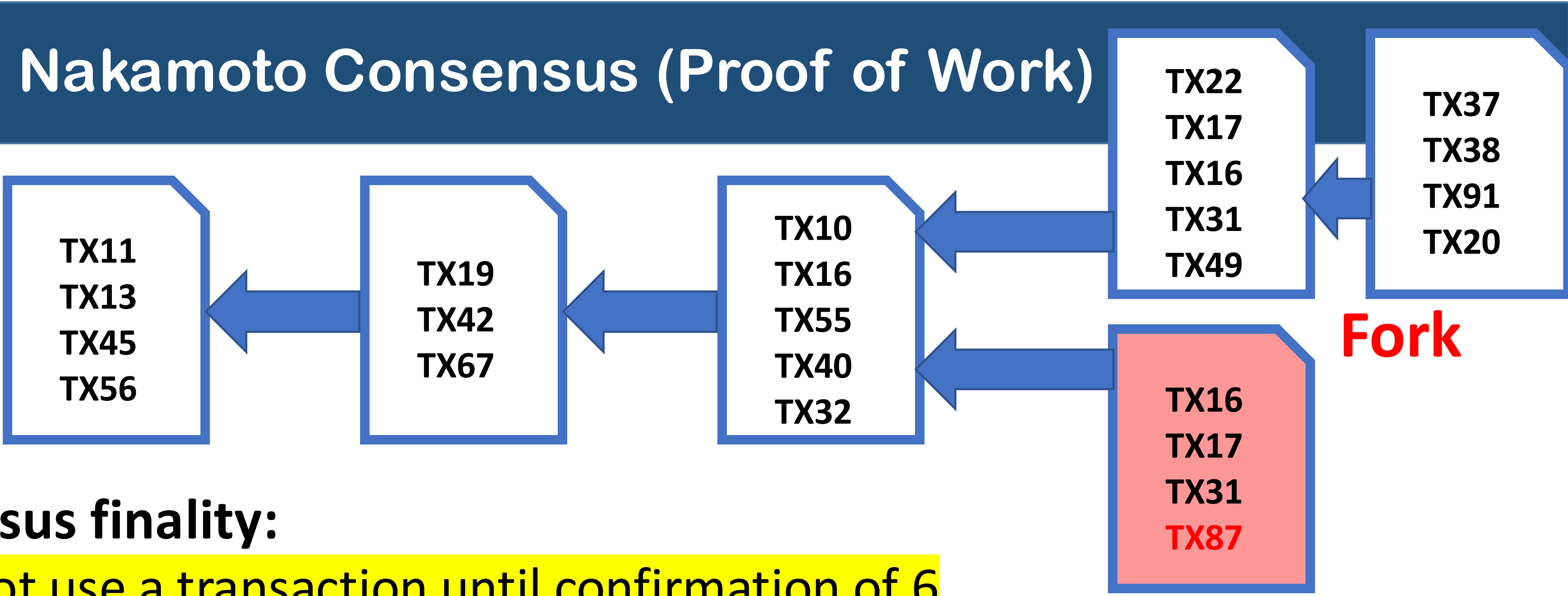
Miner 2

Unconfirmed TX

TX87

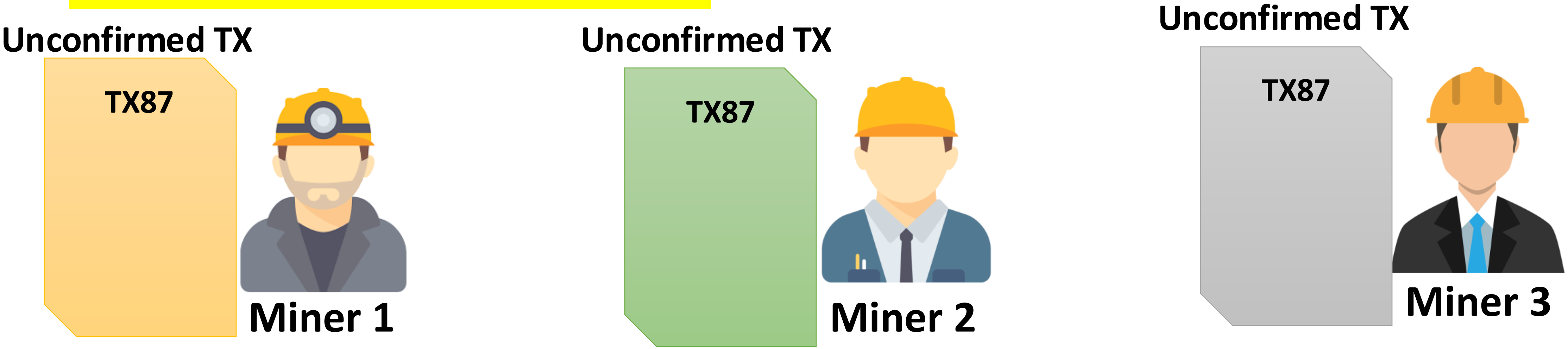
Miner 3

# Nakamoto Consensus (Proof of Work)



**Eventual consensus finality:**

- (Bitcoin) Cannot use a transaction until confirmation of 6 blocks – ensured through scripts



# Security Measures for PoW

- **Sybil Attacks**
  - Attacker attempts to fill the network with the clients under its control
  - Create multiple identities (multiple public key addresses) to control the network – refuse to relay valid blocks or relay attacked blocks
  - **Solution:** Diversify the connections – Bitcoin allows one outbound connection to per /16 block of IP addresses – cannot make both 202.141.81.2/16 and 202.141.80.18/16 as the peers

# Security Measures for PoW

- **Sybil Attacks**
  - Attacker attempts to fill the network with the clients under its control
  - Create multiple identities (multiple public key addresses) to control the network – refuse to relay valid blocks or relay attacked blocks
  - **Solution:** Diversify the connections – Bitcoin allows one outbound connection to per /16 block of IP addresses – cannot make both 202.141.81.2/16 and 202.141.80.18/16 as the peers

- **Denial of Service (DoS)**
  - Send a lot of data to a node – block the processing power
  - **Solution:** Limit forwarding of blocks, disconnect a peer that sends too many transactions

# Breaking PoW

- Bitcoin PoW is **computationally difficult** to break, but not **impossible**

- Attackers can deploy high power servers to do more work than the total work of the blockchain

- A known case of successful double-spending
  - (November 2013) “it was discovered that the GHash.io mining pool appeared to be engaging in repeated payment fraud against *BetCoin Dice*, a gambling site” [Source: https://en.bitcoin.it/]

# The Monopoly Problem

- PoW depends on the computing resources available to a miner
  - Miners having more resources have more probability to complete the work

- Monopoly can increase over time (*Tragedy of the Commons*)
  - Miners will get less reward over time
  - Users will get discouraged to join as the miner
  - Few miners with large computing resources may get control over the network

- **51% Attack:** A group of miners control more than 50% of the hash rate of the network
  - Hypothetical as of now for Bitcoin (as the network is large), but not impossible (happened for Kryptom – Ethereum based blockchain, in August, 2016)

# The Limit of PoW

- **The Good:** A fully decentralized consensus for permissionless models
  - works good for cryptocurrencies – serves its purposes

# The Limit of PoW

- **The Good:** A fully decentralized consensus for permissionless models
  - works good for cryptocurrencies – serves its purposes

- **The Bad:** Do not trust the individuals, but trust the society as a whole
  - You need a real large network to prevent the 51% attack – **not at all suitable for enterprise applications**

# The Limit of PoW

- **The Good:** A fully decentralized consensus for permissionless models
  - works good for cryptocurrencies – serves its purposes
- **The Bad:** Do not trust the individuals, but trust the society as a whole
  - You need a real large network to prevent the 51% attack – **not at all suitable for enterprise applications**
- **The Ugly**: Low transaction throughput, Overuse of computing power !!
  - (Bitcoin) 3.3 to 7 transactions per second, (Ethereum) ~15 transactions per second
  - Millions of miners – thousands tries, but only one gets the success

# Bitcoin Energy Consumption

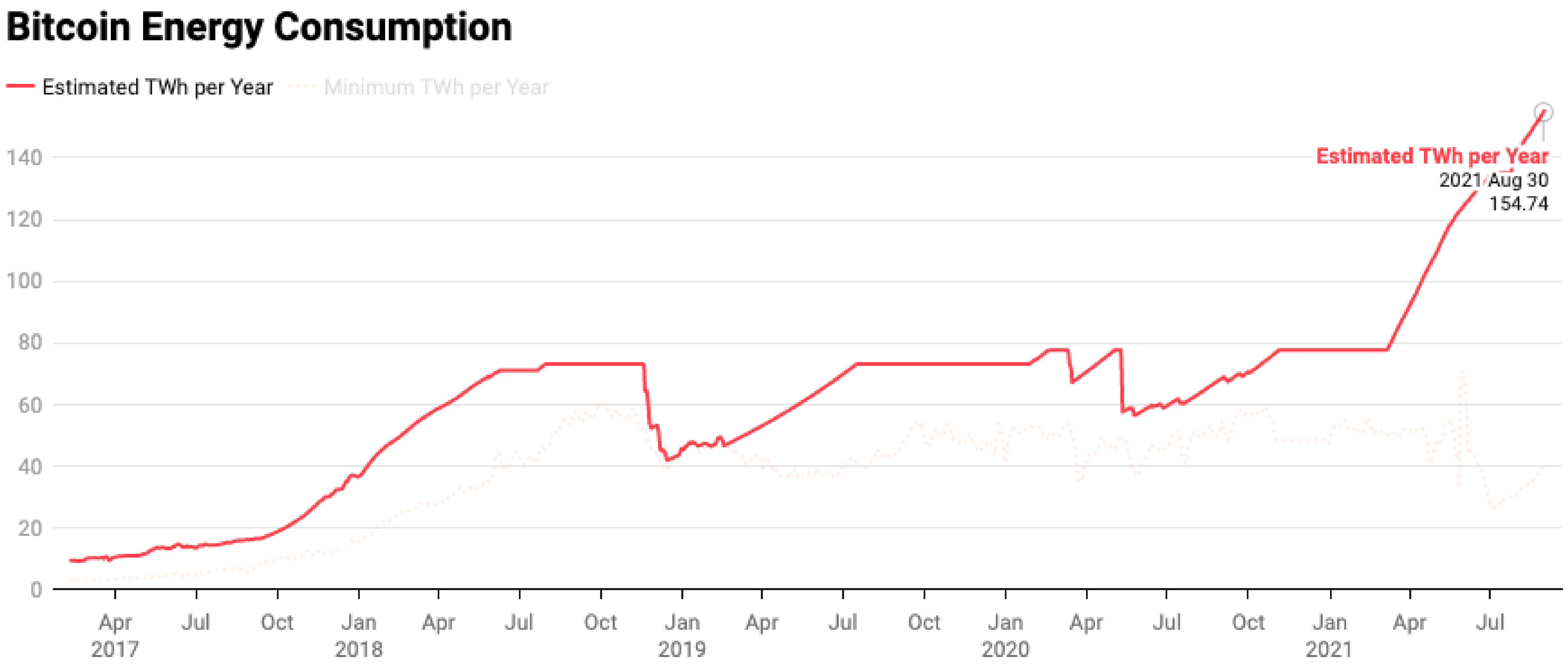
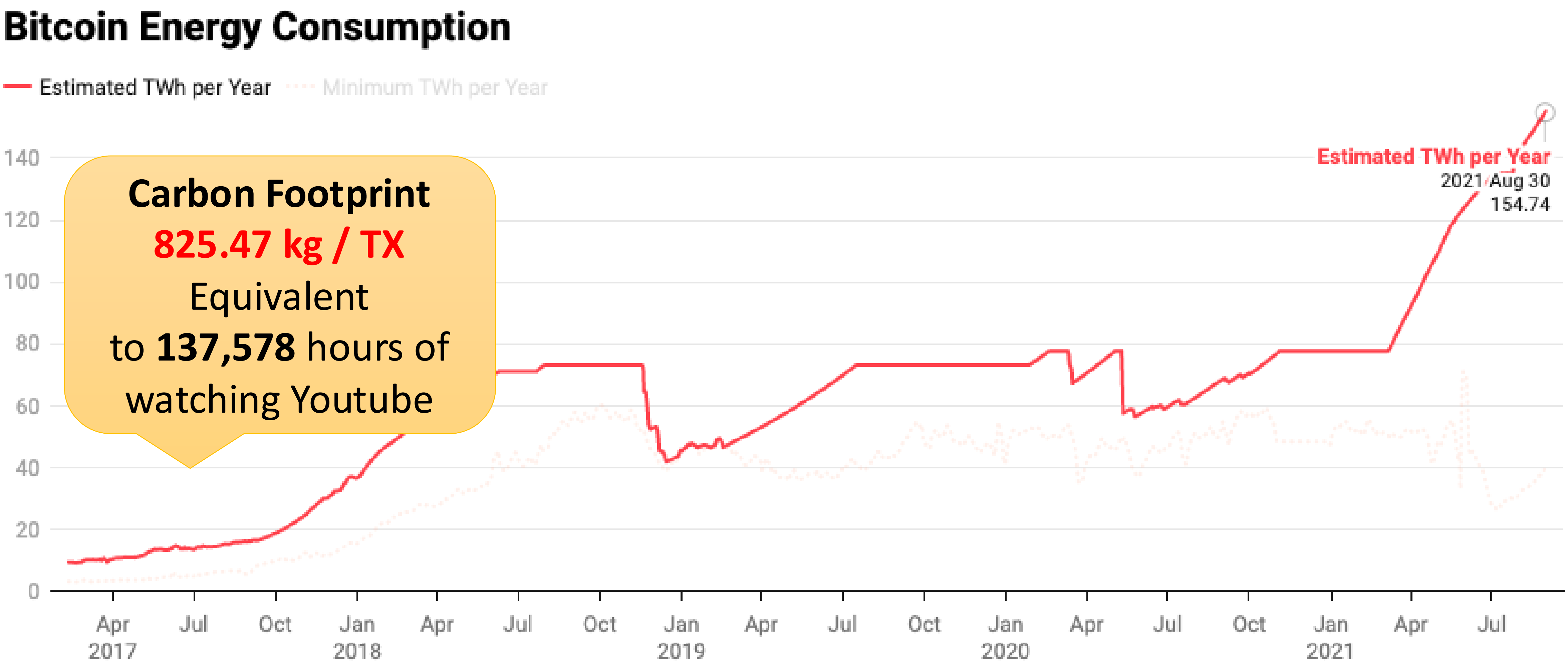


**Image Source: Digiconomist Bitcoin Energy Consumption Index**

# Bitcoin Energy Consumption



**Carbon Footprint**

**825.47 kg / TX**

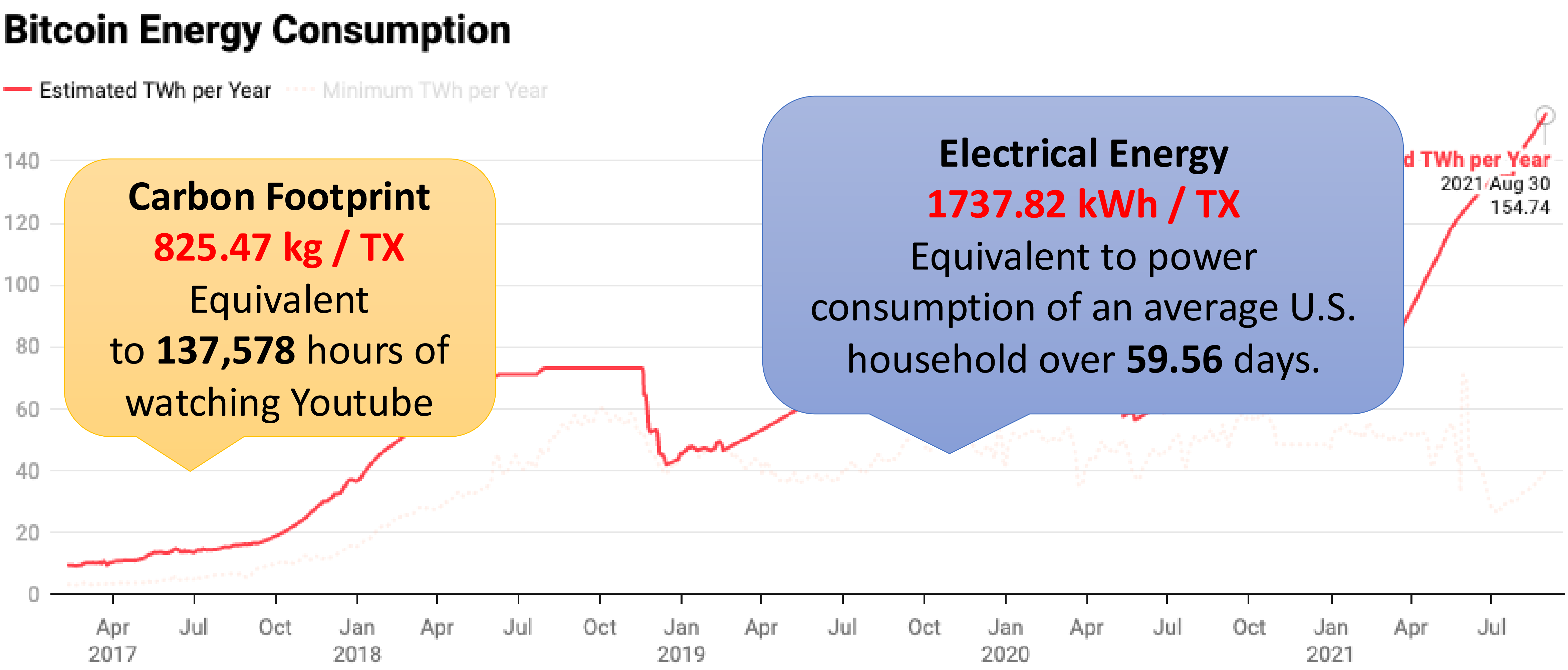
Equivalent

to **137,578** hours of

watching Youtube

**Image Source: Digiconomist Bitcoin Energy Consumption Index**

# Bitcoin Energy Consumption



**Carbon Footprint**

**825.47 kg / TX**

Equivalent to **137,578** hours of watching Youtube

**Electrical Energy**

**1737.82 kWh / TX**

Equivalent to power consumption of an average U.S. household over **59.56** days.

**Image Source: Digiconomist Bitcoin Energy Consumption Index**

# Proof of Stake (PoS)

- Possibly proposed in 2011 by a Member in Bitcoin Forum - https://bitcointalk.org/index.php?topic=27787.0
  - Make a transition from PoW to PoS when bitcoins are widely distributed

- PoW vs PoS
  - **PoW**: Probability of mining a block depends on the work done by the miner
  - **PoS**: Amount of bitcoin that the miner holds – Miner holding 1% of the Bitcoin can mine 1% of the PoS blocks.

# Proof of Stake (PoS)

- Provides increased protection
  - Executing an attack is expensive, you need more Bitcoins
  - **Reduced incentive for attack** – the attacker needs to own a majority of bitcoins – an attack will have more affect on the attacker

- Variants of “stake”
  - Randomization in combination of the stake (*used in Nxt and BlackCoin*)
  - **Coin-age**: Number of coins multiplied by the number of days the coins have been held (*used in Peercoin*)

# Ethereum PoS

- The default consensus mechanism in Ethereum is PoS
  - Switched to PoS from PoW in 2022
- **Validators:** The users who responsible for checking that new blocks propagated over the network are valid
  - Occasionally creates and propagates new blocks themselves.
- **Ethereum PoS: P**rove that validators have put something of value into the network that can be destroyed if they act dishonestly.
  - Validators explicitly stake capital in the form of ETH into a smart contract on Ethereum
  - If they try to defraud the network (*for example by proposing multiple blocks when they ought to send one or sending conflicting attestations*), some or all of their staked ETH can be destroyed.

# Ethereum PoS: Validators

- To participate as a validator, a user must
  - Deposit 32 ETH into the deposit contract
  - Run three separate pieces of software: an execution client, a consensus client, and a validator.

- On depositing their ETH, the user joins an activation queue that limits the rate of new validators joining the network.
  - Once activated, validators receive new blocks from peers on the Ethereum network.

- The transactions delivered in the block are re-executed to check that the proposed changes to Ethereum's state are valid, and the block signature is checked.
  - The validator then sends a vote (called an attestation) in favor of that block across the network.

# Ethereum PoS: Validators

- Time is divided into slots (12 seconds) and epochs (32 slots).
    - One validator is randomly selected (selected pseudo-randomly using RANDAO -- https://www.randao.org/) to be a block proposer in every slot.
    - This validator is responsible for creating a new block and sending it out to other nodes on the network.
- In every slot, a committee of validators is randomly chosen, whose votes are used to determine the validity of the block being proposed.
    - Committees divide up the validator set so that every active validator attests in every epoch, but not in every slot.

# The Need for Randomness

- Two fundamental questions --
  - How do you decide who is going to be the proposer for a new block?
  - How do you decide the committee members?

- The selection process must be **fair** to all validators

# The Need for Randomness

- Two fundamental questions --
  - How do you decide who is going to be the proposer for a new block?
  - How do you decide the committee members?

- The selection process must be fair to all validators

- **Naïve solution:** Make a sorted list of validators (may be, based on the stake), and then allow each validator to produce a block based on "round-robin" on the list
  - **What can be a possible issue with this solution?**

# The Need for Randomness

- Two fundamental questions --
  - How do you decide who is going to be the proposer for a new block?
  - How do you decide the committee members?

- The selection process must be fair to all validators

- **Naïve solution:** Make a sorted list of validators (may be, based on the stake), and then allow each validator to produce a block based on "round-robin" on the list
  - **What can be a possible issue with this solution?** Attackers know whose turn is the next – launch a DoS on that validator, or validators can conspire among their Neighbours

# The Need for Randomness

- **Solution:** Use random numbers to produce block producers in each epoch
  - **But, how do you get on-chain randomness?**

# The Need for Randomness

- **Solution:** Use random numbers to produce block producers in each epoch
  - **But, how do you get on-chain randomness?**

- One of the validators propose a random number – **Does this solution work?**

# The Need for Randomness

- **Solution:** Use random numbers to produce block producers in each epoch
  - **But, how do you get on-chain randomness?**

- One of the validators propose a random number – **Does this solution work?**
  - **What if the proposed value is not a random value generated by the computer, but something that the proposer decide? How do you validate this?**

# The Need for Randomness

- **Solution:** Use random numbers to produce block producers in each epoch
  - **But, how do you get on-chain randomness?**

- One of the validators propose a random number – **Does this solution work?**
  - **What if the proposed value is not a random value generated by the computer, but something that the proposer decide? How do you validate this?**

- Use a **group of validators** to propose their own random number – use a mixing function to generate the final output number
  - `Output → mix (input-1, input-2, input-3, …, input-n)`
  - **Condition:** Use a mixing function that gives equal weight to all the inputs

# The Need for Randomness

- Example of a mixing function: The XOR function
  - $(1111 \oplus 1001) \oplus 0101 \rightarrow 0011$

# The Need for Randomness

- Example of a mixing function: The XOR function
  - $(1111 \oplus 1001) \oplus 0101 \rightarrow 0011$
  - **Do you foresee any issue with this mixing function?**

# The Need for Randomness

- Example of a mixing function: The XOR function
  - $(1111 \oplus 1001) \oplus 0101 \rightarrow 0011$
  - **Do you foresee any issue with this mixing function?** The last validator has an advantage to propose a value that makes the output of their choice
  - The attack is possible with other mixing functions as well, although might be a little hard; for example, using cryptographic hash as the mixing function

# RANDAO

- Example of a mixing function: The XOR function
  - $(1111 \oplus 1001) \oplus 0101 \rightarrow 0011$
  - **Do you foresee any issue with this mixing function?** The last validator has an advantage to propose a value that makes the output of their choice
  - The attack is possible with other mixing functions as well, although might be a little hard; for example, using cryptographic hash as the mixing function

- **Solution:** Let other participants do not see the inputs until all inputs are revealed → Hide the proposed input through cryptographic hash until all the inputs are revealed (**Cryptographic commitments)**
  - Propose the cryptographic hash of the secret (commit phase)
  - Once everyone makes the hash values public, then reveal the original secret value (Reveal phase)

# RANDAO

- RANDAO uses cryptographic commitments along with a mixing function to generate the random number
  - Ethereum uses BLS signatures (earlier used hash onions)

# RANDAO

- RANDAO uses cryptographic commitments along with a mixing function to generate the random number
  - Ethereum uses BLS signatures (earlier used hash onions)

- **One possible issue with RANDAO** -- while they prevent a participant from changing their number, they don't actually force the participant to reveal their number.
  - Solution: Verifiable Delay Function (VDF) -- Feed the RANDAO in a function that delay the output – and use the delayed output (the output is guaranteed to be produced after a delay – you cannot arbitrarily delay to reveal the output)
  - Ethereum does not incorporate such solutions yet, is an interesting research topic

# RANDAO in Ethereum

- Every block contains a field `randao_reveal` which is the RANDAO contribution for the block proposer

- The RANDAO contributions for each block is used to compute the chain's RANDAO till the last block
  - The chain's RANDAO (along with the stake contributions from the validators) is used to select the next block proposer and the committee members to validate that block

- Check this for a detailed description of Ethereum PoS: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/

# Proof of Burn (PoB)

- Miners should show proof that they have *burned* some coins
  - Sent them to a verifiably un-spendable address
  - Expensive just like PoW, but no external resources are used other than the burned coins

- PoW vs PoB – Real resource vs virtual/digital resource

- PoB works by burning PoW mined cryptocurrencies

# Proof of Burn (PoB)

- Miners should show proof that they have *burned* some coins
  - Sent them to a verifiably un-spendable address
  - Expensive just like PoW, but no external resources are used other than the burned coins

- PoW vs PoB – Real resource vs virtual/digita

- PoB works by burning PoW mined cryptocu

**PoS and PoB**

**Ultimately depends on PoW mined cryptocurrencies**

**You cannot use them to bootstrap a new blockchain**

# Proof of Elapsed Time (PoET)

- Proposed by Intel, as a part of Hyperledger Sawtooth – a blockchain platform for building distributed ledger applications

- **Basic idea:**
  - Each participant in the blockchain network waits a random amount of time
  - The first participant to finish becomes the leader for the new block

- How will one verify that the proposer has **really waited** ?
  - Utilize special CPU instruction set – *Intel Software Guard Extension* (SGX) – a trusted execution platform
  - The trusted code is private to the rest of the application
  - The specialized hardware provides an attestation that the trusted code has been set up correctly

# Proof of Elapsed Time (PoET)

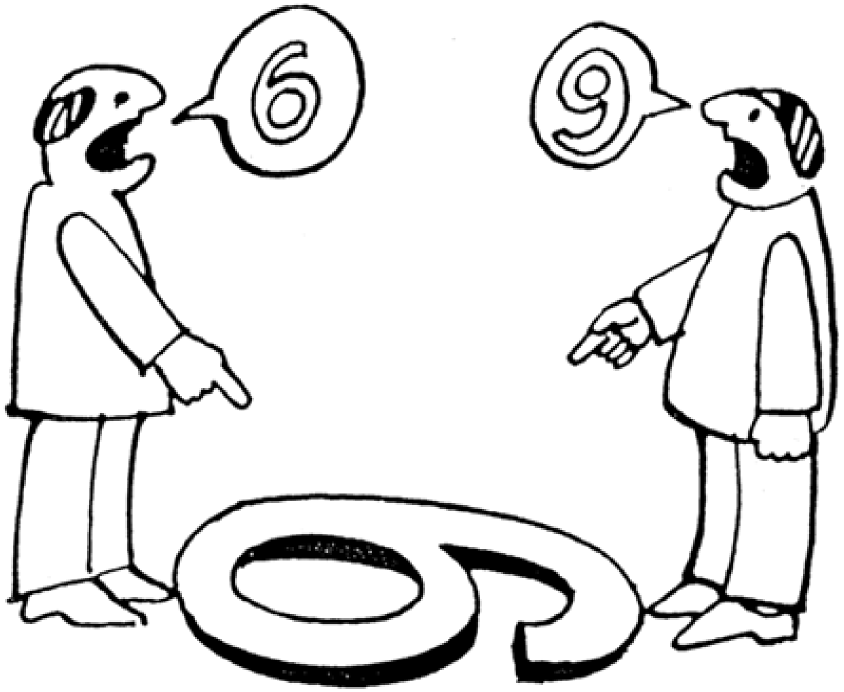
- Proposed by Intel, as a part of Hyperledger Sawtooth – a blockchain platform for building distributed ledger applications

- **Basic idea:**
  - Each participant in the blockchain network waits a random amount of time
  - The first participant to finish becomes the leader for the new block

- How will one verify that the proposer has **really waited** ?
  - Utilize special CPU instruction set – *Intel Software Guard Extension* (SGX) – a trusted execution platform
  - The trusted code is private to the rest of the application
  - The specialized hardware provides an attestation that the trusted code has been set up correctly

# What Next?

- Enterprise blockchains and Consensus thereafter
- Consensus scalability
- A decade of research on Blockchain (Distributed System?) consensus …



thank you!