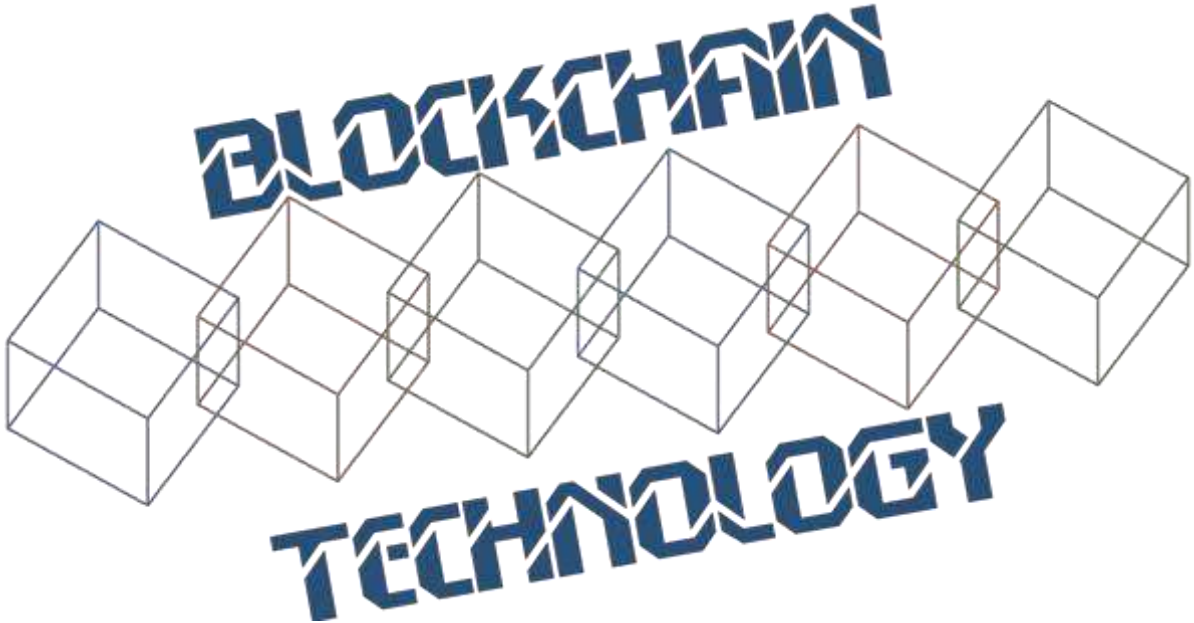
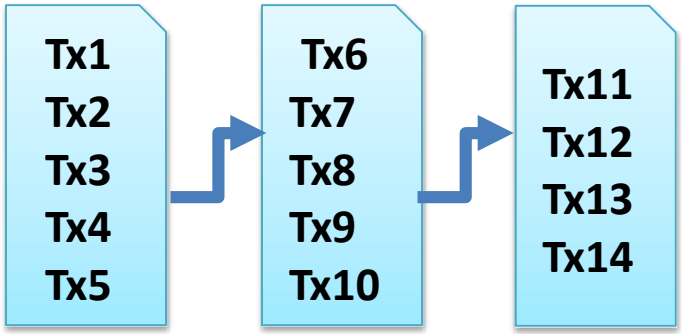
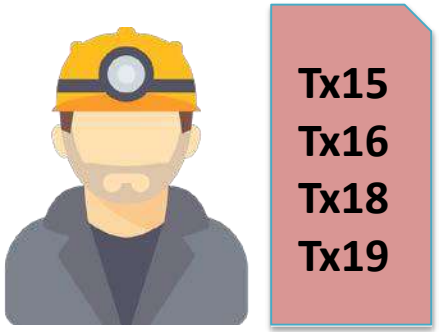


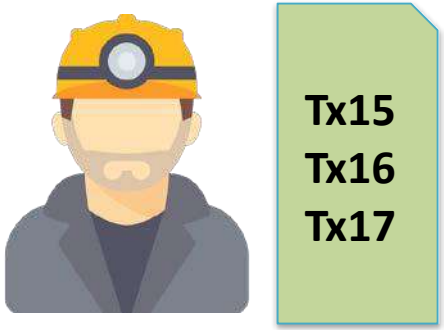
Image courtesy: http://beetfusion.com/

CONSENSUS IN BITCOIN

Consensus in Bitcoin

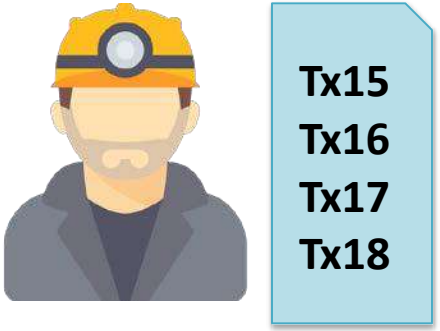




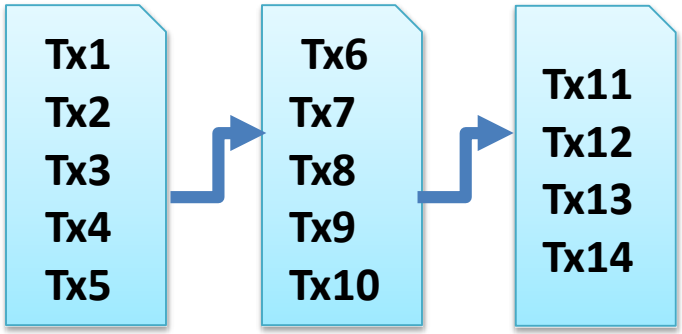


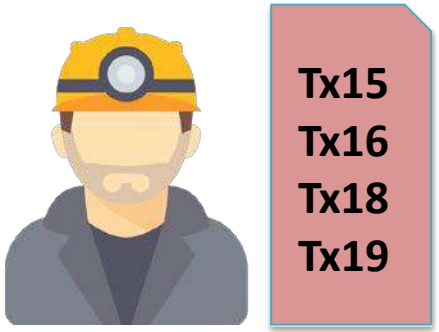
Bitcoin Consensus Objective:

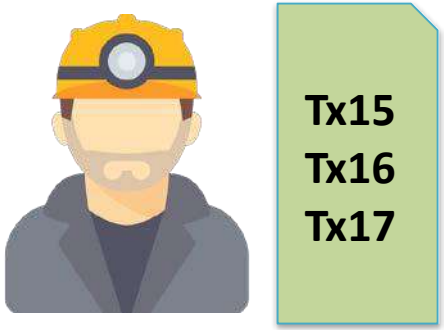
Which block do we add next?



Consensus in Bitcoin



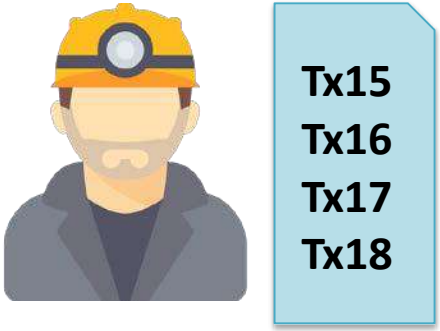




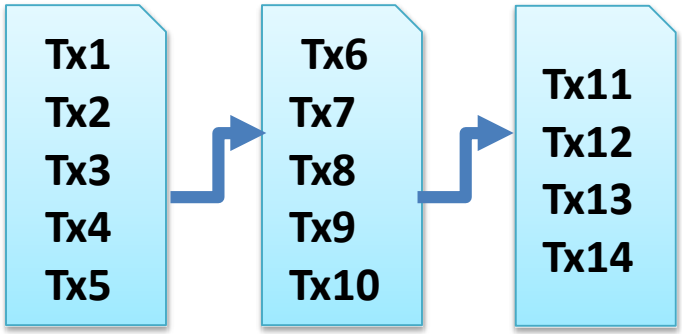
Bitcoin Consensus Objective: **Which block do we add next?**

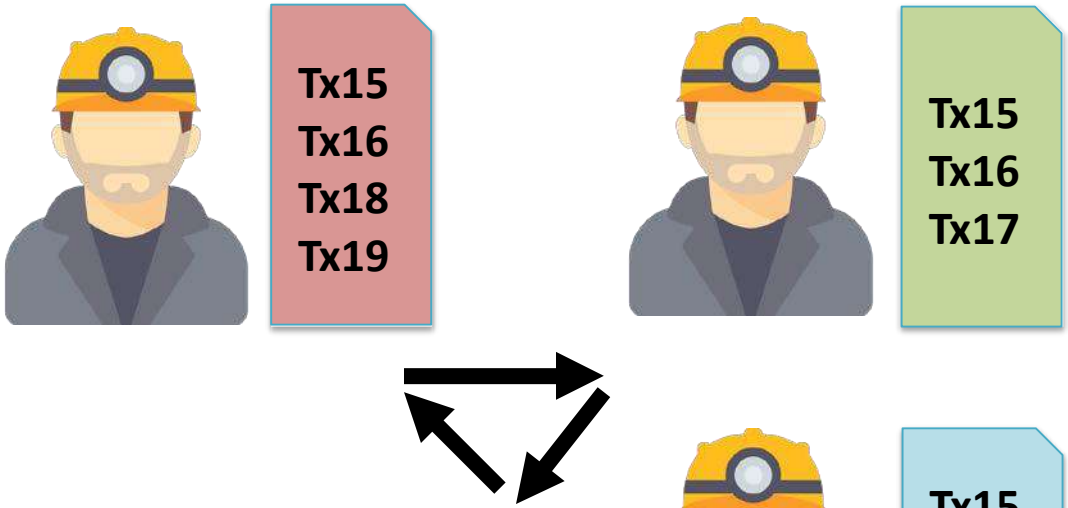
Challenge:

The miners do not know each other (Because its an Open environment)



Consensus in Bitcoin

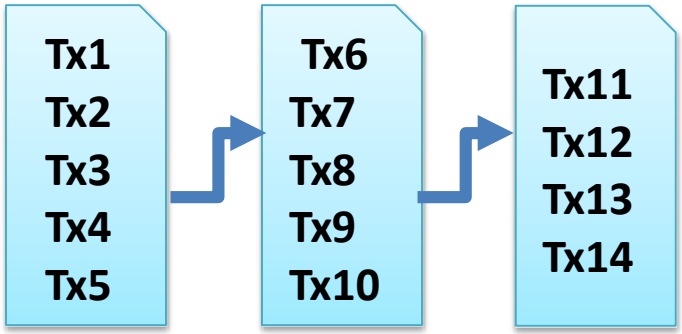


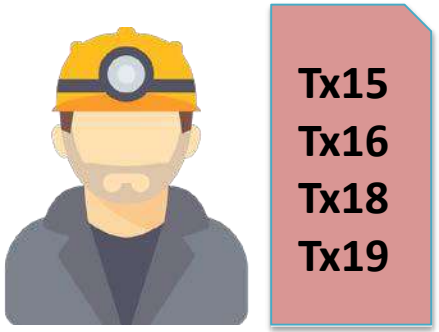


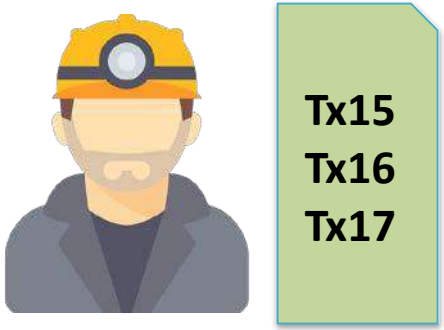
Possible Solution:

Broadcast the information in the network then wait for certain amount of time (Technically infeasible, because miners don't know about their neighbors, about other mined blocks) and then apply a choice function to select one of the block – traditional distributed consensus algorithms

Consensus in Bitcoin

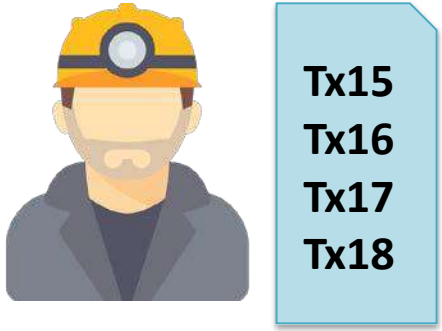




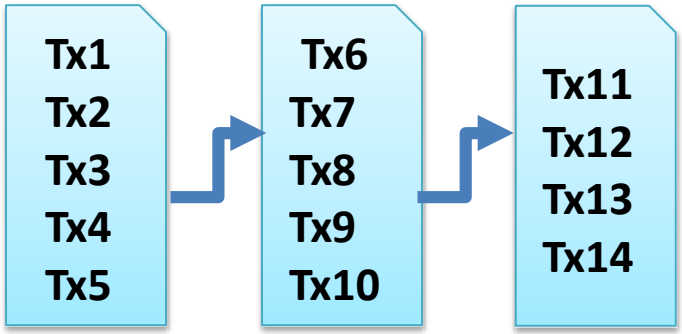


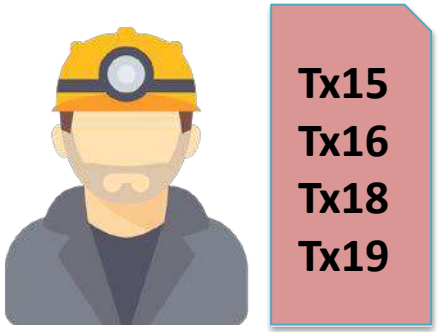
May not be Feasible:

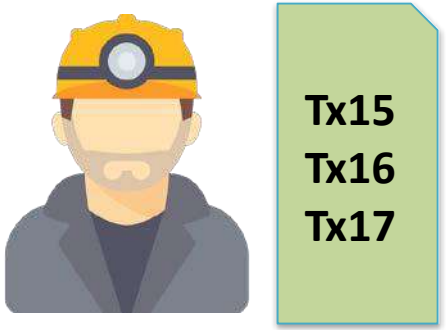
You do not have a global clock! How much time will you wait to hear the transactions Remember the impossibility result (In distributed asynchronous environment, even with a single failure)



Consensus in Bitcoin

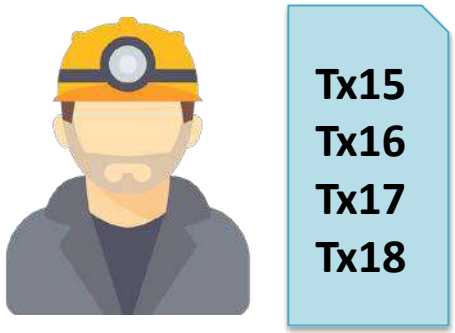




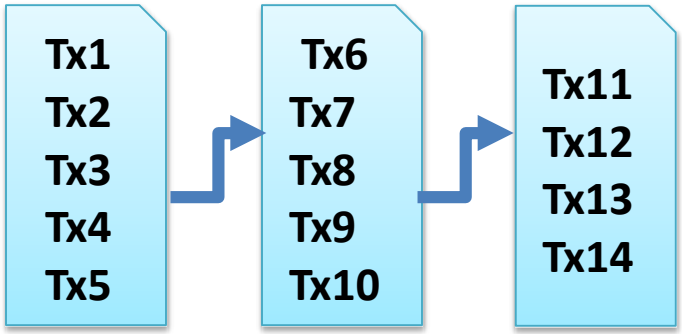


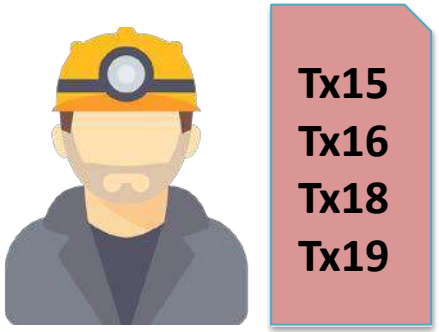
Observation - 1:

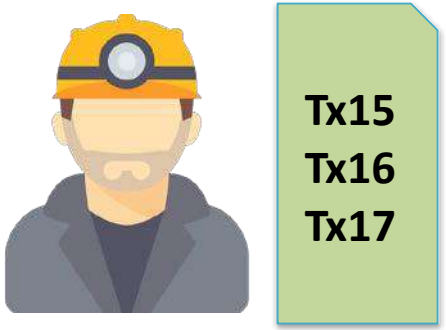
- In BC environment, any valid block (a block with all valid transactions) can be accepted, even if it is proposed by only one miner
- It is not necessary that the block will be proposed by multiple miners so if there is only one valid block then add it to the existing BC



Consensus in Bitcoin

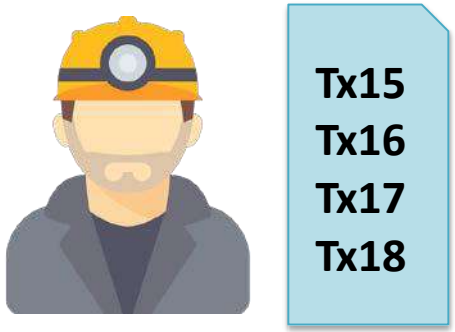




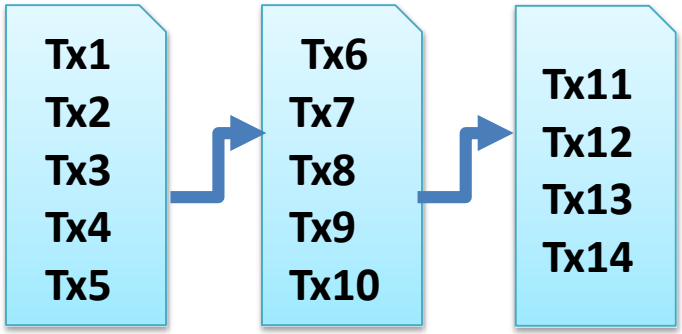


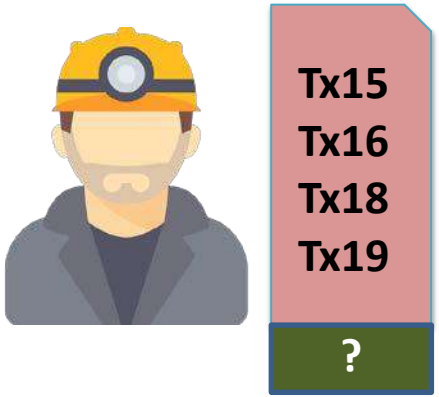
Observation - 2:

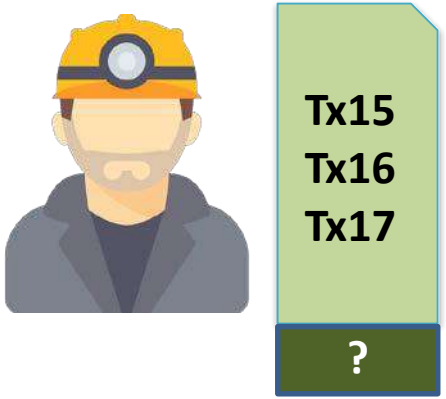
- The protocol can work in rounds
  - Broadcast the accepted block to the peers
  - Collect the next set of transactions



Consensus in Bitcoin

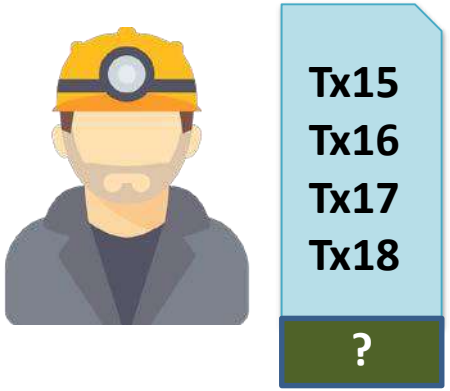




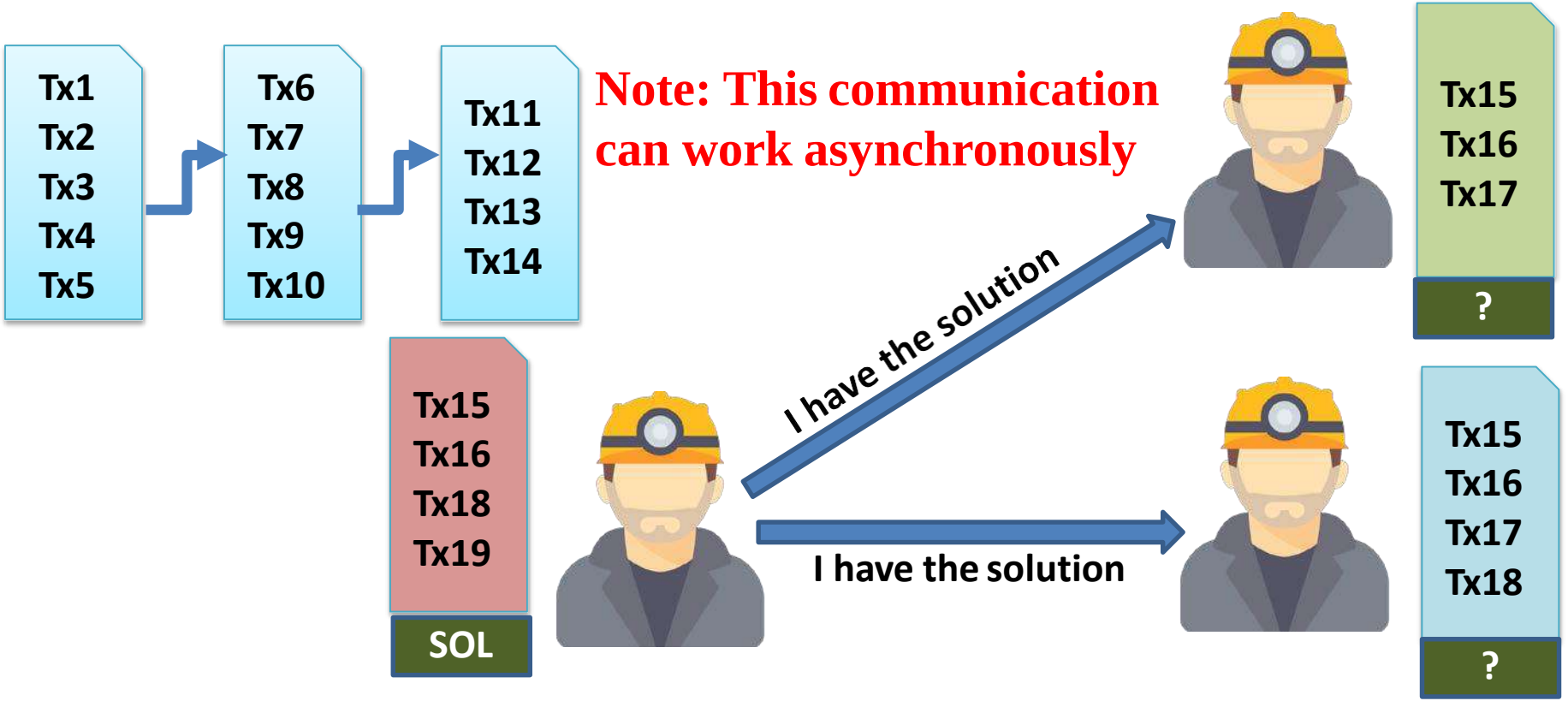


Solution:

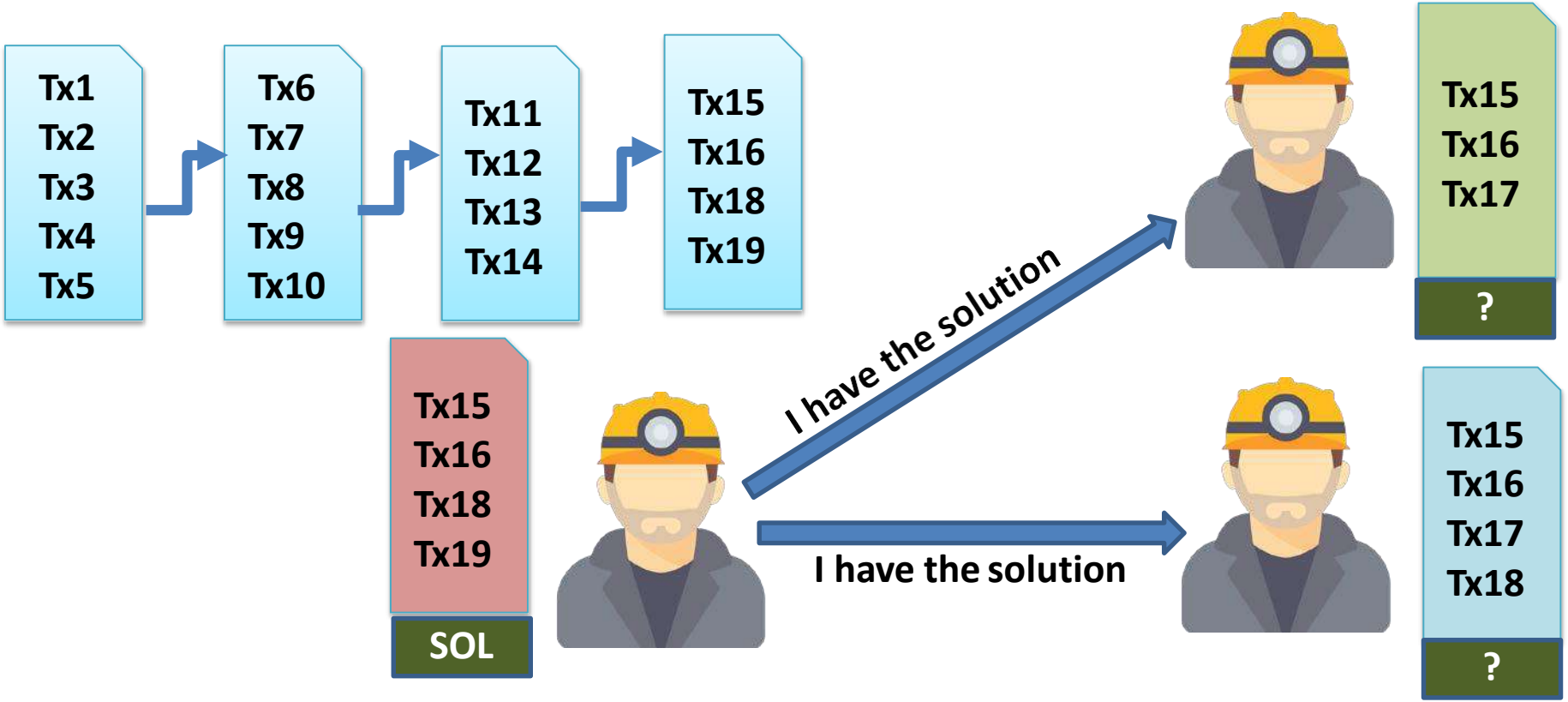
- Every miner independently tries to solve a challenge (provided by the network)
- The block is accepted for the miner who can **prove first** that the challenge has been solved (This term known as proof of work (PoW))



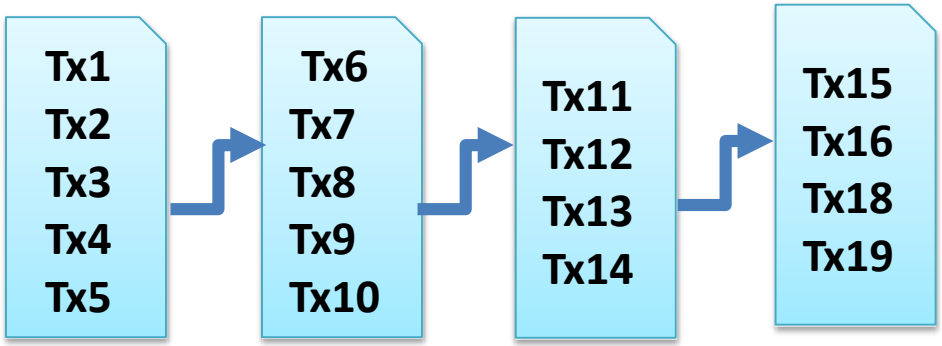
Consensus in Bitcoin

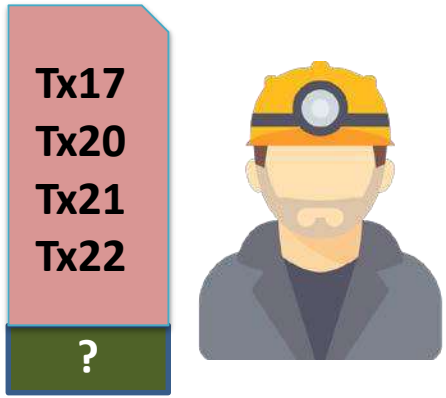


Consensus in Bitcoin

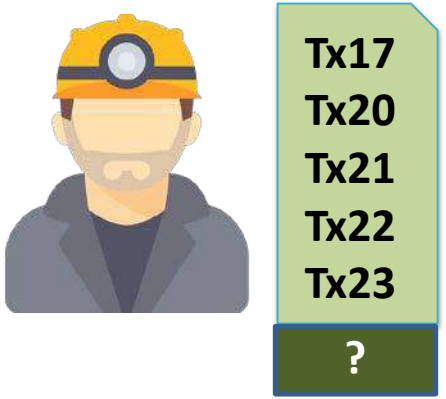


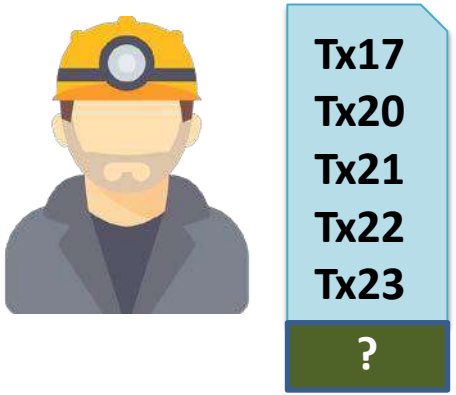
Consensus in Bitcoin





Note: Everyone can see that Tx18 and Tx19 have been committed, but Tx17 has not been committed. Include that in the next round





Proof of Work (Pow)

- An economic measure to deter service abuses by requiring some work from the service requester (usually processing time by a computer)

- The idea came from Dwork and Naor (1992), to combat junk emails
  - You have to do some work to send a valid email
  - The attacker would be discouraged to send junk emails

Dwork, Cynthia; Naor, Moni (1993). "Pricing via Processing, Or, Combatting Junk Mail, Advances in Cryptology". *CRYPTO'92: Lecture Notes in Computer Science No. 740*. Springer: 139–147.

Proof of Work (PoW) Features

- **Asymmetry**
  - The work must be moderately hard, but feasible for the service requester
  - The work must be easy check for the service provider
- Service requesters will get discouraged to forge the work, but service providers can easily check the validity of the work
- **Hashcash** is a proof-of-work system used to limit email spam and DoS attacks, and more recently used in bitcoin (and other cryptocurrencies) as part of the mining algorithm.
- **Hashcash** was proposed in 1997 by Adam Back and described more formally in Back's 2002 paper "Hashcash - A Denial of Service Counter-Measure

How the Cryptographic Hash can work as a good indicator for the PoW

- Use the **puzzle friendliness** property of cryptographic hash function as the work. It says that
  - Given X and Y, find out k, such that $Y = \text{Hash}(X||k)$ (Appended to each other)
  - If X and Y are known then it is difficult (but not infeasible) to find such k (To find it, apply random forest, brute force, etc)
  - However, once you have a k, you can easily verify the challenge

- Used in **Hashcash mechanism**, a proof of work that can be added with an email as a "*good-will"* token

Adam Back, "Hashcash - A Denial of Service Counter-Measure", technical report, August 2002

How this Hashcash PoW Mechanism works

- A textual encoding of a hashcash stamp is included in an email header
  - Proof that the sender has expended a modest amount of CPU time calculating the stamp before sending the email
  - If the sender has spent a modest time to generate this Hascash before sending the email then
  - It is unlikely that the sender is a spammer because spammer may be interested to send auto generated email through some scripts
- The Email receiver can verify the hashcash stamp very easily by applying certain hash functions
- Any change in the header requires a change in the hashcash because you are generating the hashcash from the header.
  - Brute force is the only way to find a hashcash
  - That way the attacker is discouraged to temper the email header

Example of a Hashcash PoW

- The hashcash is included in the email header, looks like this

X-Hashcash: (hash cash field)

1:20:200205:Sudeep.tanwar@nirmauni.ac.in::0000000267674 b591257b87:6078

First filed 1: it talks about the version number

Second filed 20: numbers of zero bits available in the hashcode, means you need to generate the hash function, where first 20 bits of the hashfuction would be Zero. It is the challenge posed to the email sender so he need to find out hash function with 20 zeros.

Then, date time stamp value, it was generated on 05 February 2020

Then, certain resource field here I had put my email ID

Then, string of random characters

Then, a counter value, which works as nonce

All fields are fixed except the counter filed, which the sender can change.

Version: number of zero bits in the hashed code: date: resource: optional extension: string of random characters: counter

Constriction of Hashcash PoW at Sender Side

•Construct the header:
1:20:180401:sudeep.tanwar@nirmauni.ac.in::<hash>:<counter>

- The sender initializes the counter value to a random number

- Then, the sender computes 160 bit SHA-1 hash of the header.
  - If the first 20 bit of the hash are all zeros, then it is accepted
  - Else try with a different counter and check again whether the new one has the first 20 bits of hash with all zeros

Hashcash PoW – Recipient Side

- Recipient checks
  - **The date:** – should be within two days (as per hashcash version-1)
  - **Email address:** it checks whether the email address in the hashcash exactly matches with the sender email address.
  - **The random string:** should not be used repeatedly within a certain duration (to prevent replay attack, means if attacker has get the previous hashcash and with every junk email he/she will send the previous hashcash). Means for sending every email, you need to generate new hashvalue, which prevents the replay attack.
- Validation: Compute the 160 bit SHA-1 hash of the entire received string

1:20:180401:sudeep.tanwar@nirmauni.ac.in::0000000267674b591257b87:6078

  - If the first 20 bits are not zero then it is invalid and reject the email

Analysis of Hashcash PoW

- Because of usage of 160 bit SHA, then on average, the sender will have to try 2^{20} hash values to find a valid header (takes about a few seconds in a general purpose computer)
  - There are 2^{160} possible hash values
  - 20 zero bits at the beginning – 2^{140} possible hash values that satisfy this criteria
  - Chance of randomly selecting a header with 20 leading zero bits at the prefix is 1 out of 2^{20}

- The recipient requires around 2 microsecond to validate

Bitcoin Proof of Work (PoW) System

- Most implementations of Bitcoin PoW use double SHA256 hash function
- The miners collect the transactions for 10 minutes (default setup) and starts mining the PoW
- The probability of getting a PoW is low – it is difficult to say which miner will be able to generate the block
  - No miner will be able to control the bitcoin network single handedly

Take home/Lab work …

- http://www.hashcash.org/
  - Download the source and try with different numbers of zero bit targets
  - **At sender side:** Increase the number of targeted zero bits at the hash prefix, say from 20 to 2020, at a step of 100, and observe the time to compute the hashcash. You will see that when you increase the number of leading zeros at prefix then time to compute hash will also increasers exponentially. So at what difficulty you can put the challenge?
  - **At receiver side:** Use **sha1sum tool** (in Linux) to compute the SHA-1 checksum of the obtained hashcash values from the above experiment. **How much time do you require to validate a hashcash?**

hashcash.org
home
faq
documentation
mailing-list
news
media articles
bitcoin
mail plugins
blog plugins
binaries
source
generic c
windows DLL
java lib
python
jscript+java applet
jscript+php
java applet
perl module
C# / .net
sh script
benchmarks
biggest stamp
developers
java applet
papers

hashcash libs

Navigate the menu on the left to select the language you need hashcash source code in. See also the developer notes it is fairly simple to implement hashcash even in shell scripting and other scripting langauges. See for example the sh script which implements hashcash checking only.