Bitcoin Rewards

2009- 2012: 50 BTC
2012- 2016: 25 BTC
2016-2020: 12.50 BTC
2020-2024:6.25 BTC
2024-2028: 3.125
A new block is created for every 10 Minutes (Avg)

6 Blocks(in 1 hour)*24(hours/day)=144 blocks/day

144*14(days/2 wees)= 2016 blocks for every two weeks

Bitcoin Basics – Creation of Coins

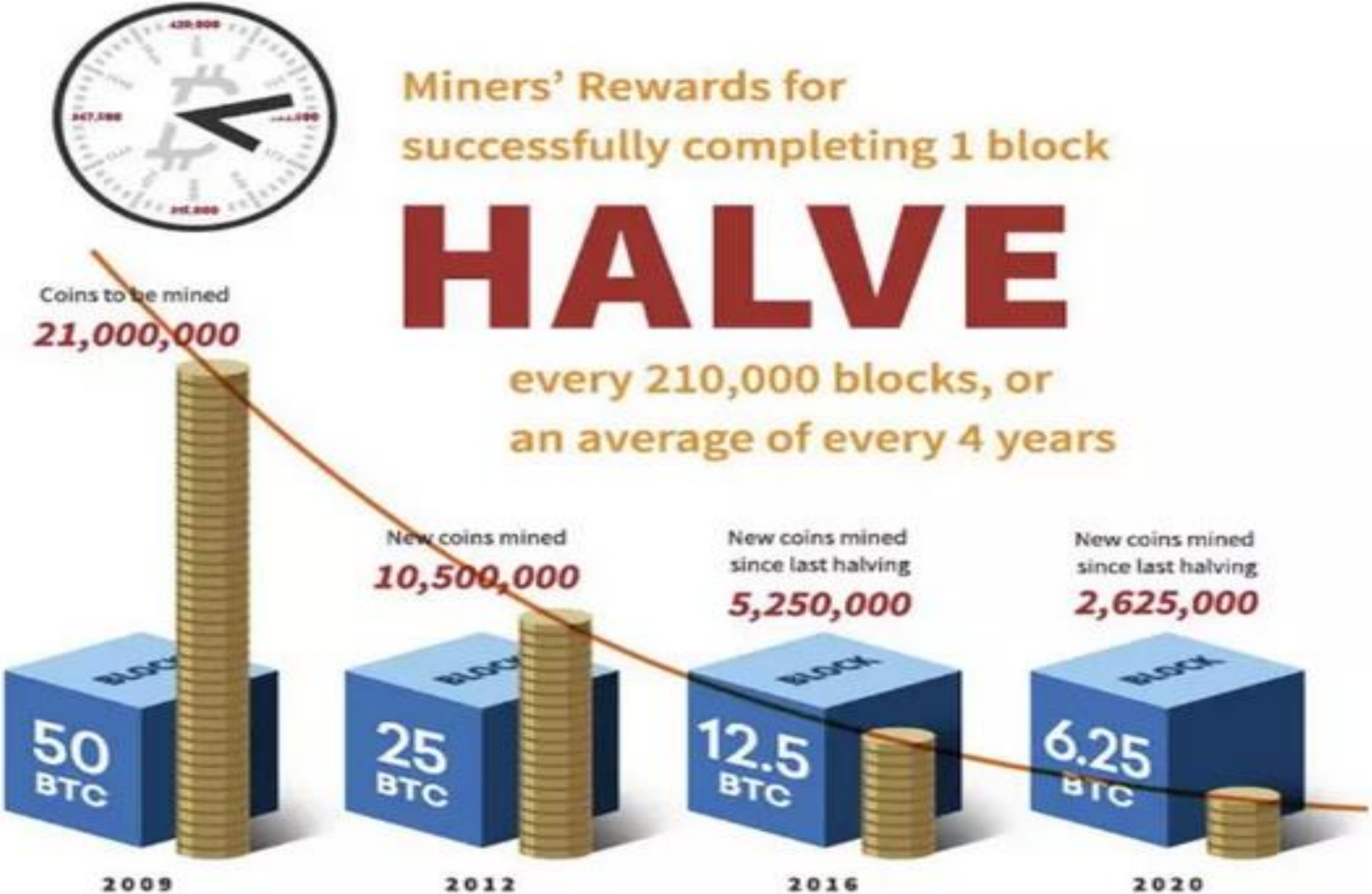
- The number of bitcoins generated per block is set to decrease **geometrically**, with a 50% reduction for every 210,000 blocks, or approximately 4 years

- This reduces, with time, the amount of bitcoins generated per block
  - Theoretical limit for total bitcoins: Slightly less than *21 million*
  - Miners will get less reward as time progresses
  - How to pay the mining fee – increase the transaction fee

Miners' Rewards for successfully completing 1 block

HALVE

every 210,000 blocks, or an average of every 4 years



Why Use Bitcoins?

☑ It's Fast

Transactions are instantaneous if they are "zero confirmation" transactions, Or, they can take around 10 minutes is a merchant requires confirmation

☑ It's Cheap

Bitcoin transaction fees are minimal, or in some cases free

☑ It's Decentralized

Because the currency is decentralized, you own it. No central authority has control

☑ No Chargebacks

Once Bitcoins have been sent, they're gone. A person who has sent Bitcoins cannot try to retrieve them without the recipient's consent

Payment Security ☑

Transactions don't require you to give up any secret information. They use two key: Public key, and a private one

It isn't inflationary ☑

Only 21 million will ever be created under the original specification. So inflation won't be a problem

It's Private ☑

It's like having a clear plastic wallet with no visible owner. Everyone can look inside it, but no one knows whose it is.

Create your own Money ☑

You can certainly buy Bitcoins on the open market, but you can also mine your own if you have enough computing power

Bitcoin Wallets:

- coinbase.com,
- wazirx,
- binance,
- bitcoinwallet.com,
- mycelium (mobile client)
- Blockchain.com
- Zebpay

Bitcoin Basics – Sending Payments

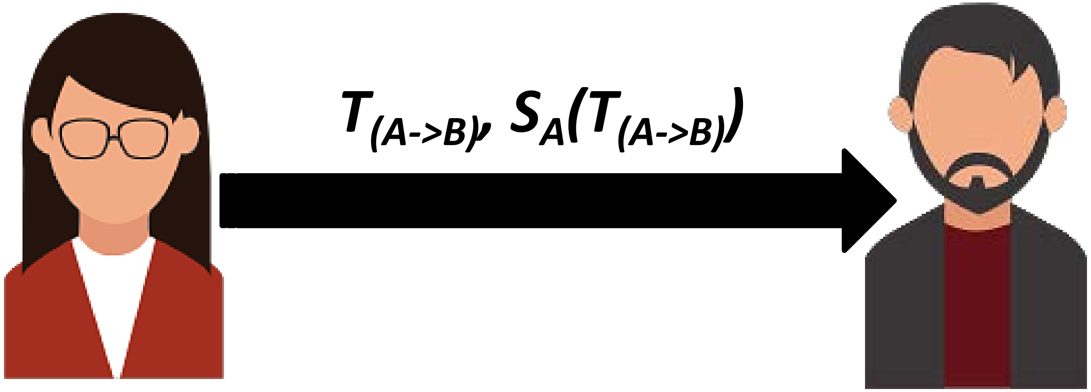
- Need to ensure that Eve cannot spend Alice’s bitcoins by creating transactions in her name.

- Bitcoin uses **public key cryptography** to make and verify digital signatures.

- Each person has one or more addresses each with an associated pair of public and private keys (may hold in the bitcoin wallet)

Bitcoin Basics – Sending Payments

- Alice wish to transfer some bitcoin to Bob.
  - Alice can sign a transaction with her private key
  - Anyone can validate the transaction with Alice’s public key



$T_{(A\text{->}B)}, S_A(T_{(A\text{->}B)})$

Bitcoin Basics – Sending Payments

- Alice wants to send bitcoin to Bob
  - Bob sends his address to Alice
  - Alice adds Bob's address and the amount of bitcoins to transfer in a "transaction" message
  - Alice signs the transaction with her private key, and announces her public key for signature verification
  - Alice broadcasts the transaction on the Bitcoin network for all to see

Bitcoin Anonymity

- Bitcoin is permission-less.

- The public and the private keys do not need to be registered, the wallet can generate them for the users

- The **bitcoin address** is used for transaction, not the user name or identity

Bitcoin Anonymity

- A **bitcoin address** mathematically corresponds to a public key based on ECDSA – the digital signature algorithm used in bitcoin

- A sample bitcoin address: 1PHYrmdJ22MKbJevpb3MBNpVckjZHt89hz

- Each person can have many such addresses, each with its own balance
  - Difficult to know which person owns what amount

BITCOIN SCRIPT

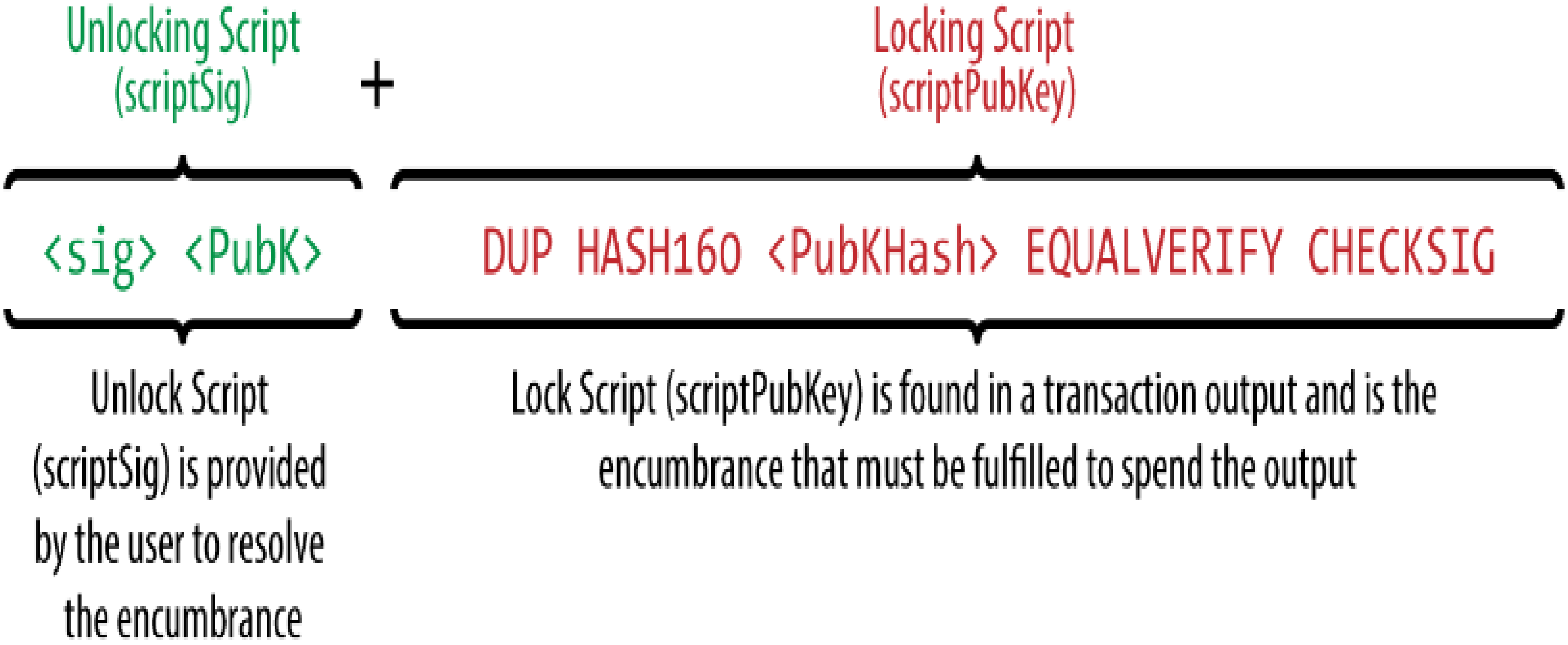
Bitcoin – Script Processing

Bitcoin developer has selected Forth programming language to write code. The code consists of the sender’s public key, signature, operations. Every transaction has a minimum of two codes, the first code is called ScriptSig, and the second one is called ScriptPubKey.

ScriptSig: It contains the public key and signature of the sender.

ScriptPubKey: It contains operations code, sender bitcoin address, and other data.

A locking script and an unlocking script



Unlocking Script (scriptSig)

+

Locking Script (scriptPubKey)

```
<sig> <PubK>
```

```
DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
```

Unlock Script (scriptSig) is provided by the user to resolve the encumbrance

Lock Script (scriptPubKey) is found in a transaction output and is the encumbrance that must be fulfilled to spend the output

Bitcoin Script

- Alice makes a transaction of BTC 20 to Bob. How Bob will claim those transactions?

- A transaction is characterized by two parameters
  - Alice sends some bitcoins: **the output (*out*) of the transaction**
  - Bob receives some bitcoins: **the input (*in*) of the transaction**

- We need to determine that **a transaction input correctly claims a transaction output**

Bitcoin Script

- A programming language to validate bitcoin transactions
  - A list of instructions recorded with each transaction
  - Describes how the next person can gain access to the bitcoins, if that person wants to spend them

- FORTH-like language, stack based and processed left to write

How FORTH Works

- A stacked based computer programming language originally designed by Charles Moore
  - A procedural programming language without type checking
  - Use a **stack** for recursive subroutine execution
  - Uses **reverse Polish notation (RPN)** or **postfix notation**

Example of Postfix Notation:

Suppose there is a mathematical expression:

(25 * 10 + 50).

In Postfix notation, this will be written as 25 10 * 50 + **CR.**

Now, we will solve this using stack. Remember following rules:

Insert operands on the top of stack.

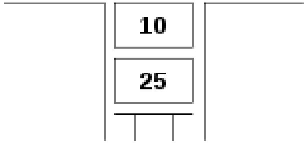
Take two operands out of the stack when you encounter a operator and put the result in the stack.

Repeat above until CR is left.

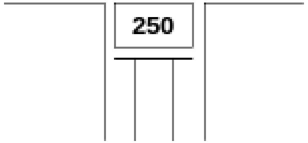
Example of Postfix Notation:

Let's solve expression 25 10 * 50 + **CR. :**

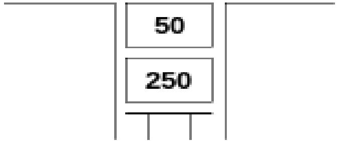
- Push 25, 10 in the stack.



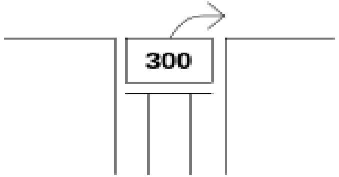
- Multiply(*) is encountered. Pop 10 and 25. Push the result back in the stack.



- Push 50 in the stack.



- Addition operator(+) is encountered. Pop 50 and 250. Perform 50+250 and push the result back in the stack.



- **CR** moves the output to the new line and . prints the output to user.