bitcoin
the Protocol

G-J van Rooyen

20 February 2014

"With e-currency based on cryptographic proof, without the need to trust a third-party middleman, money can be secure and transactions effortless."

– Satoshi Nakamoto

“We can laugh at Bitcoin, but real guys, in real basements, are losing real fake money right now.”

– David Clinch

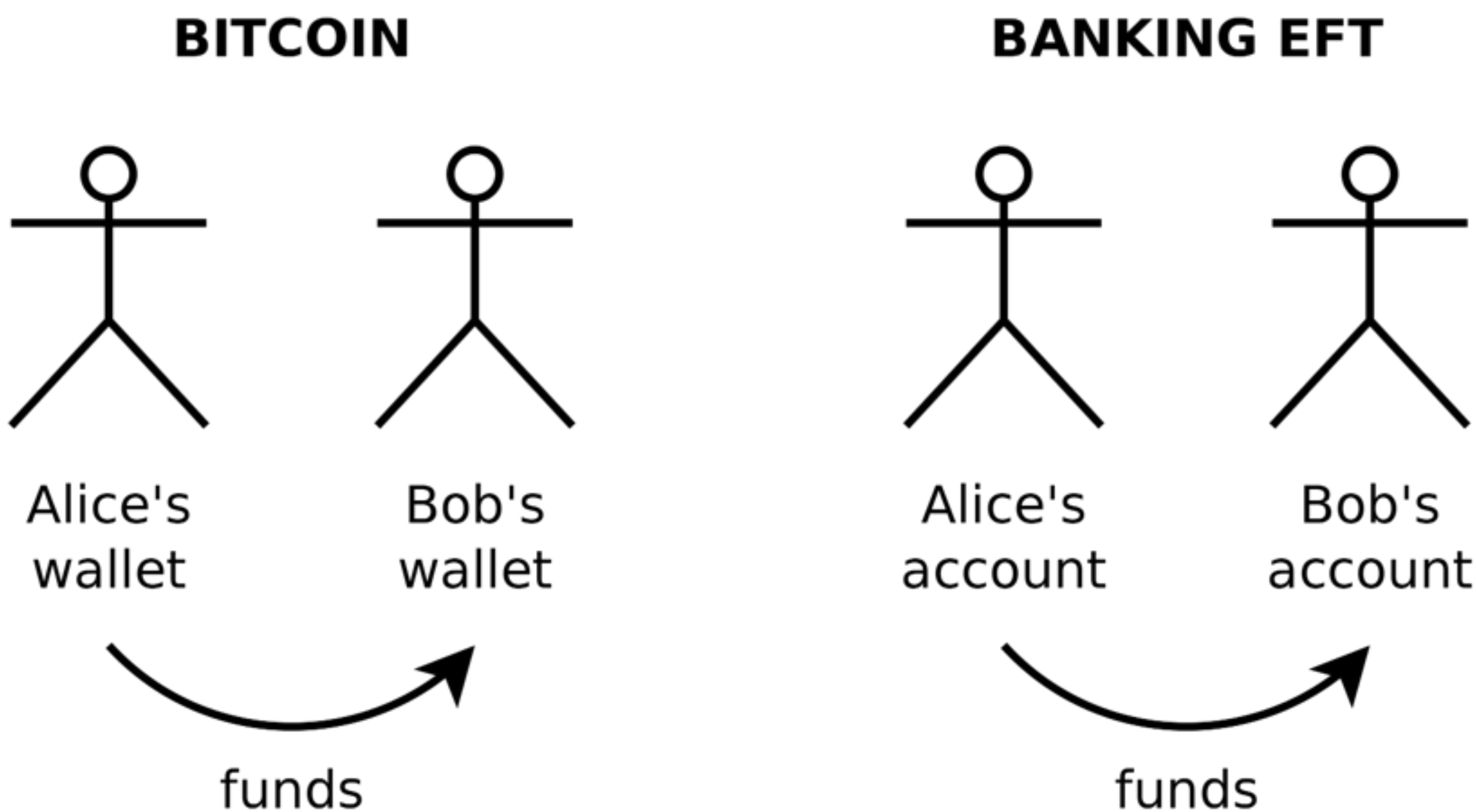
This talk is not about…

- …is Bitcoin “real” money?
- …is Bitcoin a good investment?
- …will Bitcoin replace the dollar/rand/yen?
- …is Dogecoin/Litecoin better than Bitcoin?
- …exchange volatility

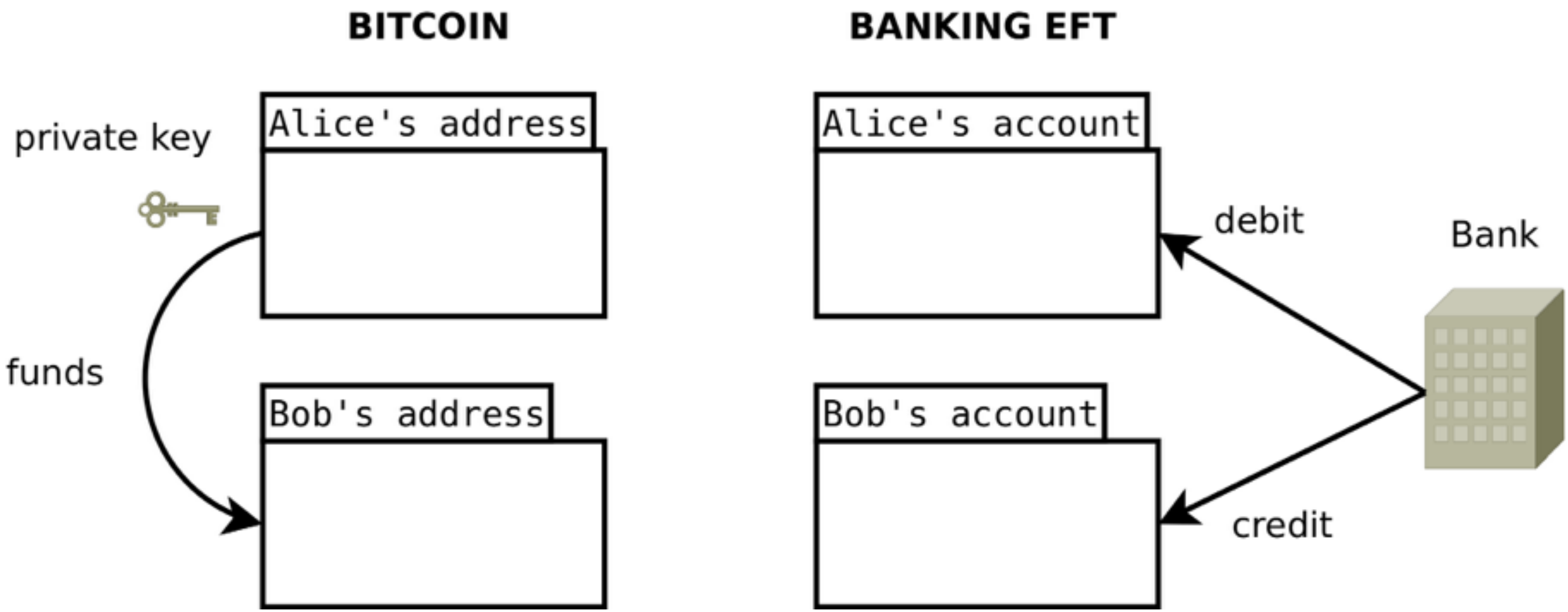
We will talk about…

- …difficulty of **trust-free agreement** in a decentralised P2P network (*Byzantine Generals*)
- …**triple-entry accounting**
- …how Bitcoin **transactions** are built and verified
- …the **scripting language** built into the protocol
- …scripted **contracts** *("Bitcoin 2.0")*

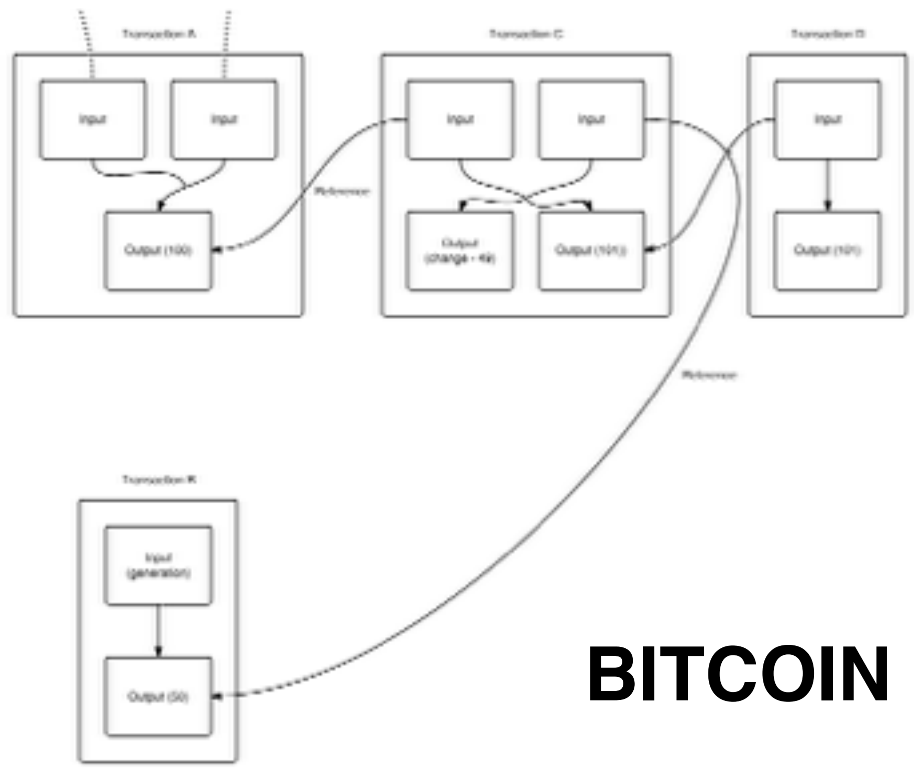
Abstraction, Level 1



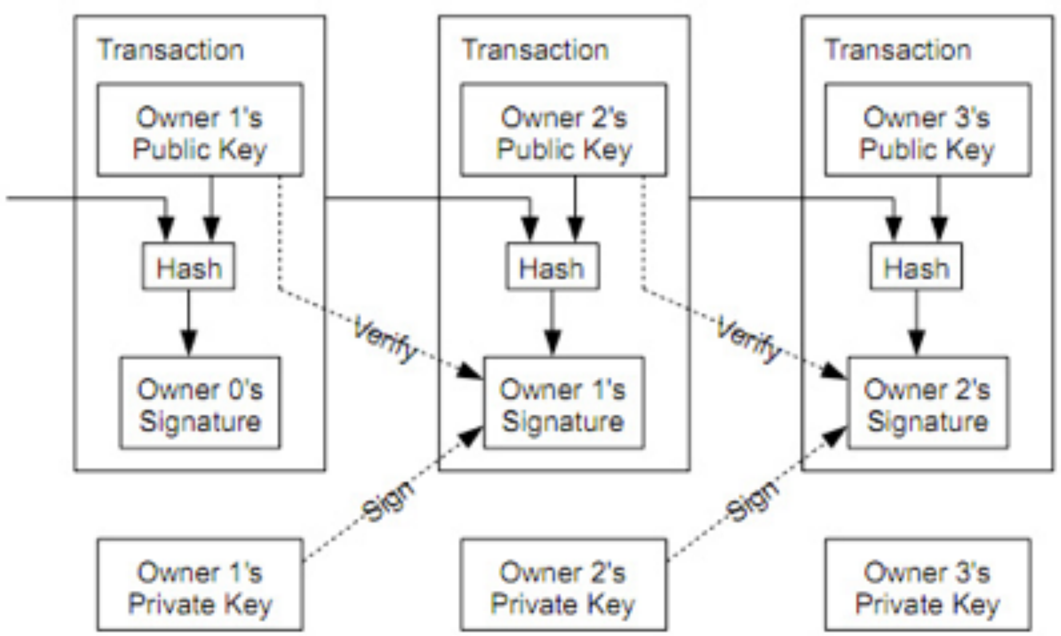
Abstraction, Level 2



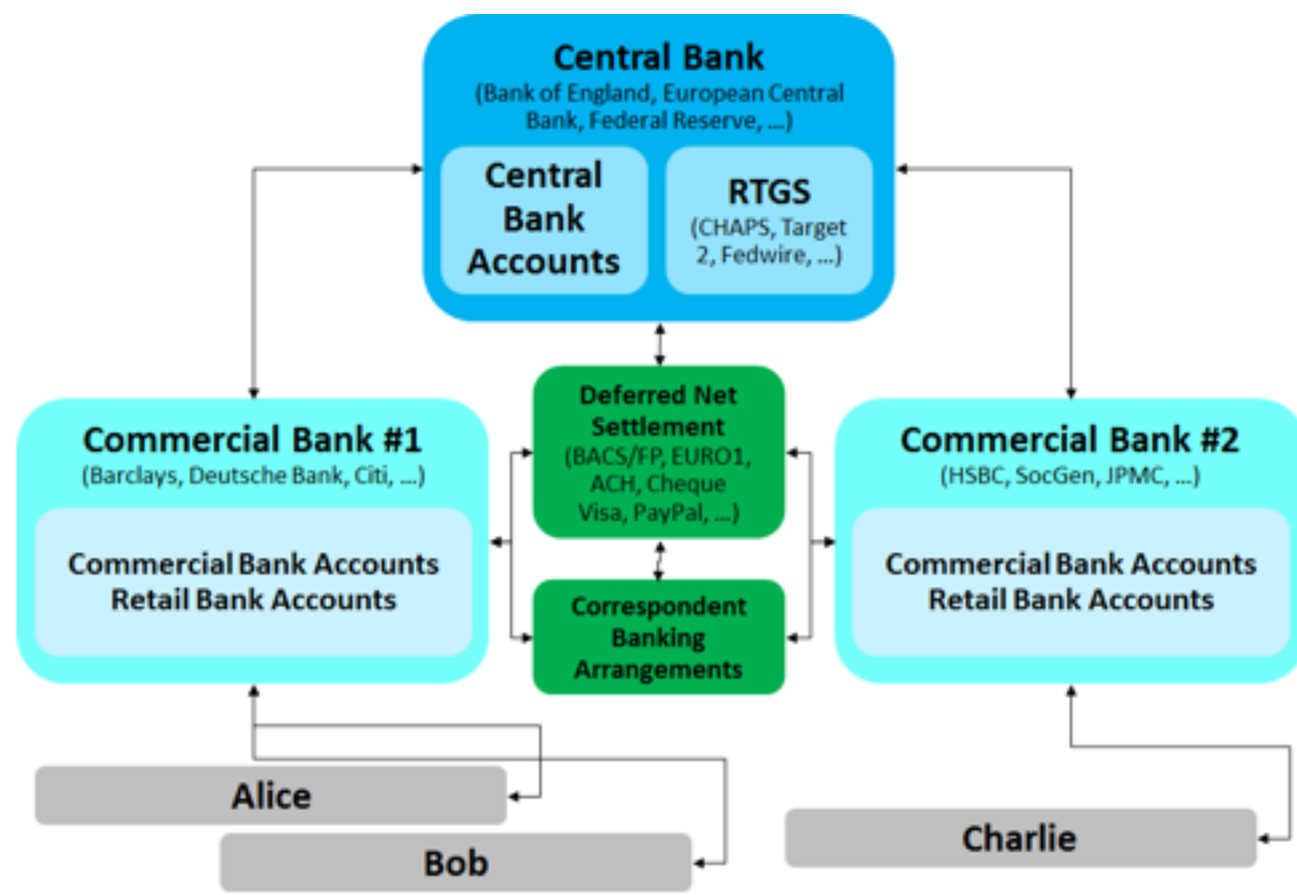
Abstraction, Level 3



BITCOIN



BANKING EFT



Byzantine Generals

- *N* generals need to coordinate an attack
- Messages are passed amongst each other
- Traitorous generals may pass on false messages
- Consensus **very difficult**
- Lamport: solution for 2/3 trust (later > 50%)

Nakamoto’s Solution

- Scenario: generals have to **agree on time** to attack
- A random general proposes a time and distributes the message
- Other generals “sign off” (agree) on time adding a hash that’s computationally **difficult to compute** (but trivial to verify)
- A **chain of time-plus-hashes** builds up and is distributed
- Over time, the generals become convinced that the **majority of the computational power** of the network has reached consensus.
- If an attacker injects a fake time to spread confusion, the network selects the chain with the **longest sequence** of valid hashes

Proof-of-work

attack at 10:00!	nonce #1	hash #1	nonce #2	hash #2	...

Application to ownership transfer

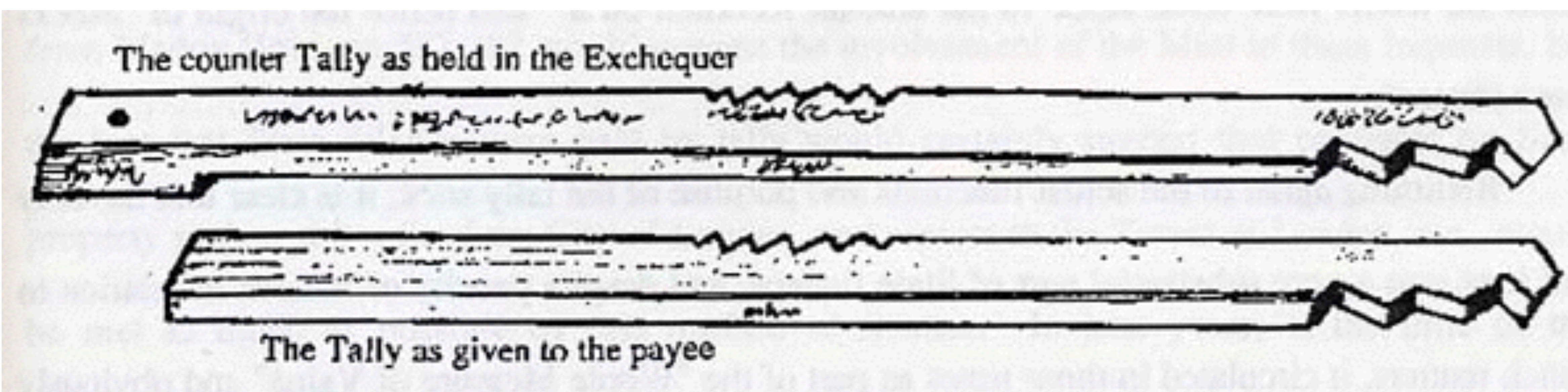
- I can sign a “cheque” giving away money I own
- Everyone can verify the transaction is valid
- A double-spend of money is always invalid
- People who “audited” the transaction sign it off by proof-of-work

Single-entry accounting

- “Write down income and expenses”
- If you leave out a number, no-one will know
- Bookkeeper always has plausible deniability *(it was an honest mistake!)*
- Limited businesses to family and crown

Double-entry accounting

- Florence, late 13th century
- Much more difficult to "cook books"
- Gave rise to the modern enterprise



Bitcoin as triple-entry

- Alice debits her wallet, and credits Bob's (double-entry)

- Ivan audits transaction

- Ivan commits it to the public ledger (third entry)

- No central authority

- Non-repudiable transaction

The basics of Bitcoin

Back to Abstraction, Level 2

The basics of Bitcoin: Private keys

- Each “account” = **random 256-bit number**
- Private key, must be kept **secret**
- Need not be stored digitally – can be on paper or memorised
- Want to open an account? Guess a number!

QR code representation of the private key
5KQx3qRcMD5FyogomtVnABuToGCoVVDC9HvPMwDgARWBqzzzNte

The basics of Bitcoin: Public keys and addresses

- **ECDSA** is used to generate a public key from the PrivKey
- The PubKey can be used to **verify transactions** signed using the PrivKey
- 64-byte PubKeys are unwieldy, and are hashed down to **20-byte addresses**

QR code representation of the address
`1MZhiFUaJSLpUyrCj8de7d5UMvZLtyuu1z`

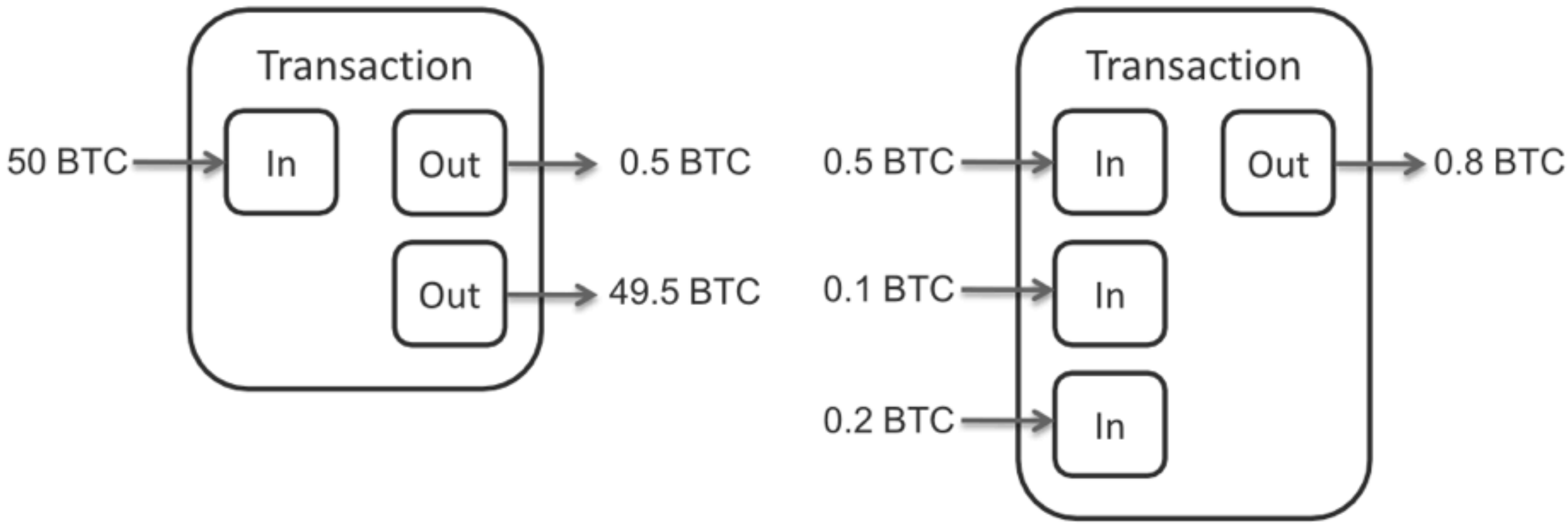
Crypto-primer: Hashes

- **Hashing**: $D = H(M)$
  - D is usually much shorter than M
  - It is impossible to get back to M just from D
  - SHA256 and RIPEMD-160 used in Bitcoin

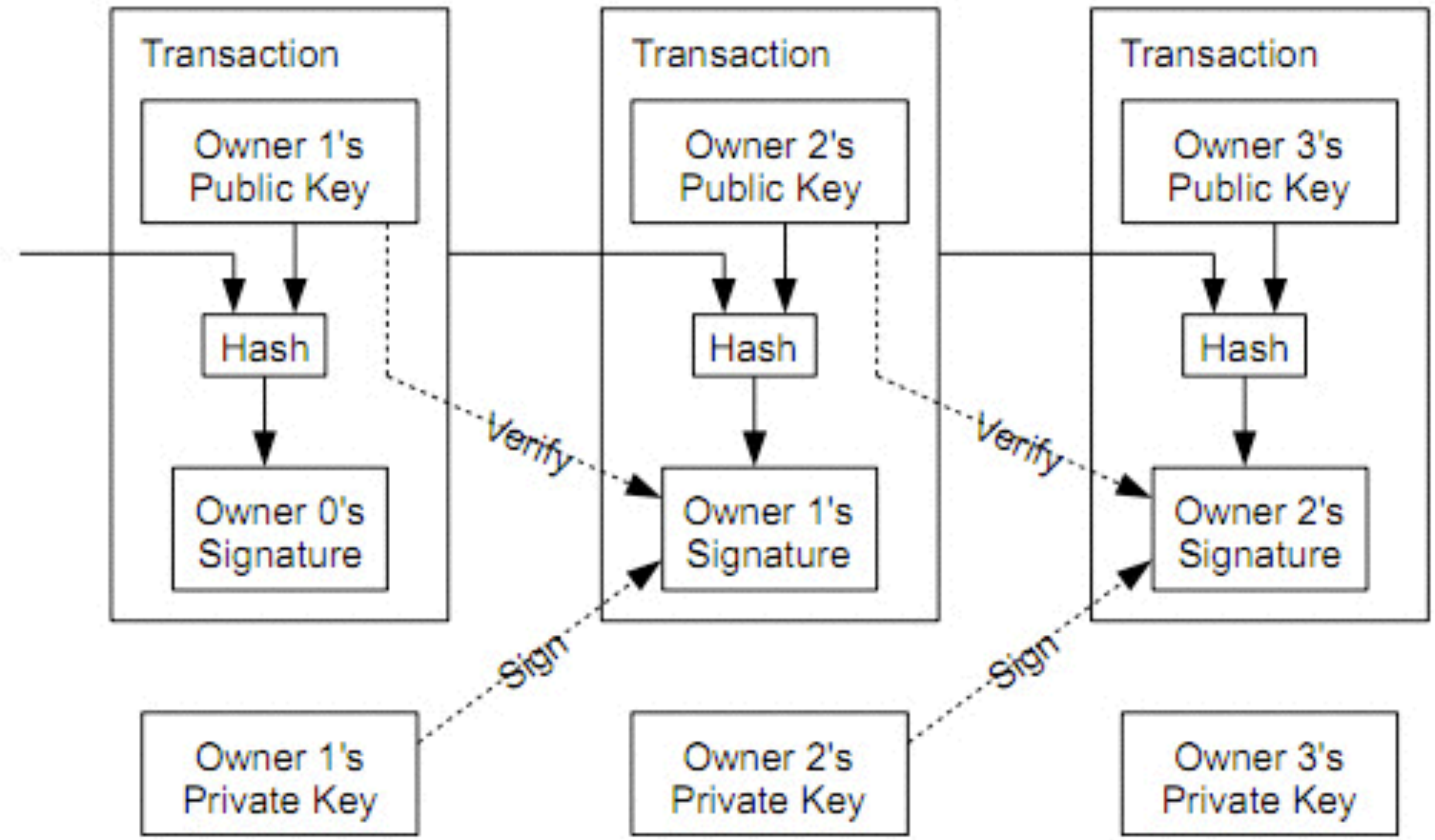
Crypto-primer: Signatures

- **Hashing**: $D = H(M)$
- **Signing**: $\sigma = S(D, P_r)$
- **Verification**: $\beta = V(D, \sigma, P_u)$
    - Only the owner of the private key can sign a message (transaction)
    - Anyone who knows a user's public key can verify that she signed it
    - ECDSA used in Bitcoin

Bitcoin transactions



Signing inputs



A full transaction

Field	Description	Size [b]
Version #	Currently 1	4
In-counter	Positive integer	1-9
List of inputs	References to outputs of previous transactions	
Out-counter	Positive integer	1-9
List of outputs	Values of outputs, and scripts dictating how they may be claimed	
Lock time	Time stamp when transaction becomes final (default 0 = immediately)	4

A sample transaction

```
Input:
Previous tx:
f5d8ee39a430901c91a5917b9f2dc19d6d1a0e9cea205b009ca73dd04
470b9a6
Index: 0
scriptSig:
304502206e21798a42fae0e854281abd38bacd1aeed3ee3738d9e1446
618c4571d1090db022100e2ac980643b0b82c0e88ffdfec6b64e3e6ba
35e7ba5fdd7d5d6cc8d25c6b241501

Output:
Value: 5000000000
scriptPubKey: OP_DUP OP_HASH160
404371705fa9bd789a2fcd52d2c580b65d35549d
OP_EQUALVERIFY OP_CHECKSIG
```

The output script

- Each output has a **script** specifying how it may be claimed
- **FORTH**-like scripting language
- Deliberately **Turing-incomplete**
- **Can specify anything**:
    - “anyone can have this”
    - pay to specific address
    - highly complex contracts (e.g. “pay out when I die”)

The simplest script

- **Pay-to-PubkeyHash** (give money to an address)
- **scriptPubKey**: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
- **scriptSig**: <sig> <pubKey>
- scriptSig and scriptPubKey are combined, and then **stack processing** is done operation-by-operation

The simplest script: Step 1

- **scriptSig** and **scriptPubKey** are combined
- **Unprocessed script**:
  <sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

STACK

(empty)

The simplest script: Step 2

- The **constants** <sig> and <pubKey> are added to the stack

- **Unprocessed script**: ~~<sig> <pubKey>~~ OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

STACK

<pubKey>
<sig>

The simplest script: Step 3

- The top stack item is **duplicated**

- **Unprocessed script**:
  ~~<sig> <pubKey>~~ ~~OP_DUP~~
  OP_HASH160 <pubKeyHash>
  OP_EQUALVERIFY
  OP_CHECKSIG

STACK

<pubKey>
<pubKey>
<sig>

The simplest script: Step 4

- The top stack item is **hashed**
- This calculates and **address** from the claimant's **public key**
- (we must ensure this is the same as the TXout's address)
- **Unprocessed script**: ~~<sig> <pubKey> OP_DUP OP_HASH160~~ <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

STACK

<pubHashA>
<pubKey>
<sig>

The simplest script: Step 5

- Another constant (the previous output's **destination address**) is **added to the stack**

- **Unprocessed script**:
  ~~<sig> <pubKey> OP_DUP~~
  ~~OP_HASH160~~ ~~<pubKeyHash>~~
  OP_EQUALVERIFY
  OP_CHECKSIG

STACK

<pubKeyHash>
<pubHashA>
<pubKey>
<sig>

The simplest script: Step 6

- Verify that the claimant's **public key** actually **matches** the previous transaction's output **address**
- **If false**, the transaction is rejected an not distributed further
- **Unprocessed script**:
  ~~<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY~~
  OP_CHECKSIG

STACK

<pubKey>
<sig>

The simplest script: Step 7

- Verify that the claimant's **public key** confirms the transaction's **signature**
- **If false**, the transaction is rejected an not distributed further
- **Unprocessed script**:
  ~~<sig> <pubKey> OP_DUP~~
  ~~OP_HASH160 <pubKeyHash>~~
  ~~OP_EQUALVERIFY~~
  ~~OP_CHECKSIG~~

STACK

(empty)

We just used 4 opcodes...

```
enum opcodetype
{
    // push value
    OP_0 = 0x00,
    OP_FALSE = OP_0,
    OP_PUSHDATA1 = 0x4c,
    OP_PUSHDATA2 = 0x4d,
    OP_PUSHDATA4 = 0x4e,
    OP_1NEGATE = 0x4f,
    OP_RESERVED = 0x50,
    OP_1 = 0x51,
    OP_TRUE=OP_1,
    OP_2 = 0x52,
    OP_3 = 0x53,
    OP_4 = 0x54,
    OP_5 = 0x55,
    OP_6 = 0x56,
    OP_7 = 0x57,
    OP_8 = 0x58,
    OP_9 = 0x59,
    OP_10 = 0x5a,
    OP_11 = 0x5b,
    OP_12 = 0x5c,
    OP_13 = 0x5d,
    OP_14 = 0x5e,
    OP_15 = 0x5f,
    OP_16 = 0x60,

    // control
    OP_NOP = 0x61,
    OP_VER = 0x62,
    OP_IF = 0x63,
    OP_NOTIF = 0x64,
    OP_VERIF = 0x65,
    OP_VERNOTIF = 0x66,
    OP_ELSE = 0x67,
    OP_ENDIF = 0x68,
    OP_VERIFY = 0x69,
    OP_RETURN = 0x6a,

    // stack ops
    OP_TOALTSTACK = 0x6b,
    OP_FROMALTSTACK = 0x6c,
    OP_2DROP = 0x6d,
    OP_2DUP = 0x6e,
    OP_3DUP = 0x6f,
    OP_2OVER = 0x70,
    OP_2ROT = 0x71,
    OP_2SWAP = 0x72,
    OP_IFDUP = 0x73,
    OP_DEPTH = 0x74,
    OP_DROP = 0x75,
    OP_DUP = 0x76,
    OP_NIP = 0x77,
    OP_OVER = 0x78,
    OP_PICK = 0x79,
    OP_ROLL = 0x7a,
    OP_ROT = 0x7b,
    OP_SWAP = 0x7c,
    OP_TUCK = 0x7d,

    // splice ops
    OP_CAT = 0x7e,
    OP_SUBSTR = 0x7f,
    OP_LEFT = 0x80,
    OP_RIGHT = 0x81,
    OP_SIZE = 0x82,

    // bit logic
    OP_INVERT = 0x83,
    OP_AND = 0x84,
    OP_OR = 0x85,
    OP_XOR = 0x86,
    OP_EQUAL = 0x87,
    OP_EQUALVERIFY = 0x88,
    OP_RESERVED1 = 0x89,
    OP_RESERVED2 = 0x8a,

    // numeric
    OP_1ADD = 0x8b,
    OP_1SUB = 0x8c,
    OP_2MUL = 0x8d,
    OP_2DIV = 0x8e,
    OP_NEGATE = 0x8f,
    OP_ABS = 0x90,
    OP_NOT = 0x91,
    OP_0NOTEQUAL = 0x92,

    OP_ADD = 0x93,
    OP_SUB = 0x94,
    OP_MUL = 0x95,
    OP_DIV = 0x96,
    OP_MOD = 0x97,
    OP_LSHIFT = 0x98,
    OP_RSHIFT = 0x99,

    OP_BOOLAND = 0x9a,
    OP_BOOLOR = 0x9b,
    OP_NUMEQUAL = 0x9c,
    OP_NUMEQUALVERIFY = 0x9d,
    OP_NUMNOTEQUAL = 0x9e,
    OP_LESSTHAN = 0x9f,
    OP_GREATERTHAN = 0xa0,
    OP_LESSTHANOREQUAL = 0xa1,
    OP_GREATERTHANOREQUAL =
    OP_MIN = 0xa3,
    OP_MAX = 0xa4,

    OP_WITHIN = 0xa5,

    // crypto
    OP_RIPEMD160 = 0xa6,
    OP_SHA1 = 0xa7,
    OP_SHA256 = 0xa8,
    OP_HASH160 = 0xa9,
    OP_HASH256 = 0xaa,
    OP_CODESEPARATOR = 0xab,
    OP_CHECKSIG = 0xac,
    OP_CHECKSIGVERIFY = 0xad,
    OP_CHECKMULTISIG = 0xae,
    OP_CHECKMULTISIGVERIFY =

    // expansion
    OP_NOP1 = 0xb0,
    OP_NOP2 = 0xb1,
    OP_NOP3 = 0xb2,
    OP_NOP4 = 0xb3,
    OP_NOP5 = 0xb4,
    OP_NOP6 = 0xb5,
    OP_NOP7 = 0xb6,
    OP_NOP8 = 0xb7,
    OP_NOP9 = 0xb8,
    OP_NOP10 = 0xb9,


    // template matching params
    OP_SMALLDATA = 0xf9,
    OP_SMALLINTEGER = 0xfa,
    OP_PUBKEYS = 0xfb,
    OP_PUBKEYHASH = 0xfd,
    OP_PUBKEY = 0xfe,

    OP_INVALIDOPCODE = 0xff,
};
```

Mining

- “Auditors” collect transactions into a “block” (up to 1 Mb)
- Each transaction in the block is verified for validity
- The miner then does a proof-of-work calculation to “sign off” the block and add it to the blockchain
- Difficult hash calculation takes +/- 10 min regardless of number of miners in the network

Advanced mining

- A miner who successfully finds a suitable hash for a block, gets **reward** (currently 25 XBT = 625 USD)
- Each transaction has optional **transaction fees** (difference between sum of inputs and outputs) that also go to the miner
- **Hash difficulty**: number of “leading zeros” in hash
- **Adjusted dynamically**, aims for 1 block in 10 mins

More interesting contracts

- **Scripting language** can be used to enforce arbitrary constraints on how outputs are spent

- Entire **financial applications** involving transfer of **ownership** can be built using the Bitcoin protocol

Dispute mediation

- Third party (**escrow / arbiter**) may optionally be called in to sign off on a transaction if something goes wrong.

- Script:
  2 <K1> <K2> <K3> 3 OP_CHECKMULTISIGVERIFY

- **2 out of 3** parties must agree on the outcome of the transaction in order to spend the output

- The output may be spent as a **payment** or a **refund**

Micropayment channels

- Each Bitcoin transaction carries a **transaction cost** (or processing delay), so normal transactions aren't ideal for micropayments
- Client send **rapid adjustments** in what it is willing to transfer to the server, *directly to the server*
- These transactions **aren't broadcast until the session ends**, when the final payment is made.

Oracle conditions

- E.g. script to pass on an **inheritance**:
  - <hash> OP_DROP2 <son’s pubkey> <oracle’s pubkey> CHECKMULTISIG
- Uses an external, trusted **oracle** who will only sign off when predetermined condition is met, e.g.
  - if (has_died(‘G-J van Rooyen’, id=‘7609257364083’)) return (10.0, 1MZhiFUaJSLpUyrCj8de7d5UMvZLtyuu1z)

Colored coins

- From “wallet point-of-view”, Bitcoins are **fungible**
- However, transaction outputs are **traceable**
- 0.00000001 XBT outputs can be used to trace **ownership** of associated digital or physical goods **in the real world**
- Software, movies, stocks, cars, houses can be traded without intermediaries

In Conclusion

- The Bitcoin protocol is **brilliant**, **subtle**, **intricate** and (in some places) **horribly complex**
- Proof-of-ownership **protocol** with built-in **scripting** language
- **Currency** (“pay-to-address”) is the “Hallo, world!” of Bitcoin applications
- Understand the protocol. Then go understand **traditional financial systems**

Questions are welcome

Also,
5KQx3qRcMD5FyogomtVnABuToGCoVVDC9HvPMwDgARWBqzzzNte