



# THE PEER-TO-PEER NETWORK

---

JOHN NEWBERY

[@jfnewbery](#)

[github.com/jnewbery](https://github.com/jnewbery)

---

# THE PEER-TO-PEER NETWORK

- ▶ Introduction
- ▶ Types of nodes
- ▶ Message format
- ▶ Control messages
- ▶ Transaction propagation
- ▶ Block propagation



---

# THE PEER TO PEER NETWORK

---

## WHAT IS THE PEER-TO-PEER NETWORK?

- ▶ How transactions and blocks are propagated to Bitcoin nodes
- ▶ Open, flat peer-to-peer network - no authentication, no special nodes
- ▶ Must be resistant to attacks:
  - ▶ Denial-Of-Service attacks
  - ▶ Sybil attacks

---

# NETWORK COMMANDS

- ▶ VERSION
- ▶ VERACK
- ▶ ADDR
- ▶ GETADDR
- ▶ INV
- ▶ GETDATA
- ▶ GETBLOCKS
- ▶ GETHEADERS
- ▶ TX
- ▶ BLOCK
- ▶ HEADERS
- ▶ PING
- ▶ PONG

---

## CONNECTING TO THE PEER-TO-PEER NETWORK

- ▶ Nodes initially connect to one or more seed nodes
- ▶ Addresses of other nodes on the network are gossiped using *ADDR* messages
- ▶ A Bitcoin Core node will connect to up to eight outbound peers
- ▶ Nodes may or may not accept inbound peers

---

## DISCONNECTING AND BANNING (1)

- ▶ Nodes which misbehave need to be removed:
  - ▶ They waste system resources
  - ▶ They take up slots that could be used for honest peers

---

## DISCONNECTING AND BANNING (2)

- ▶ 'bad behavior' may include:
  - ▶ Invalid transactions or blocks
  - ▶ Unconnected blocks
  - ▶ Stalling
  - ▶ *Non-standard* transactions
  - ▶ Malformed messages



---

## DISCONNECTING AND BANNING (3)

- ▶ Depending on the misbehavior, Bitcoin Core may:
  - ▶ Ignore the problem and continue
  - ▶ Disconnect the peer immediately
  - ▶ *Ban* the peer (disconnect and don't allow connections from the same IP address for 24 hours)
  - ▶ Apply DoS points. When the DoS score reaches 100, ban the peer



---

# TYPES OF NODES

---

## FULL NODE

- ▶ Also called a *fully validating* node
- ▶ Receives blocks as they are mined and propagated around the network
- ▶ Verifies the validity of all blocks and all transactions included in those blocks
- ▶ Enforces the *consensus rules* of the Bitcoin network
- ▶ Maintains a collection of all the unspent outputs
- ▶ The most secure and private way to use Bitcoin

---

## PRUNED NODE

- ▶ A type of full node
- ▶ Discards old block data to save disk space
- ▶ Retains at least 2 days of blocks and undo data to allow for re-orgs
- ▶ Propagates new blocks but cannot serve old blocks
- ▶ As secure as a non-pruned full node

---

## 'ARCHIVAL' NODE

- ▶ Unlike a pruned node, retains all old block and undo data
- ▶ Can serve old blocks to peers on the network
- ▶ Signaled using **NODE\_NETWORK** in the version handshake

---

## SIMPLE PAYMENT VERIFICATION (SPV) NODE (1)

- ▶ Only downloads:
  - ▶ the block headers
  - ▶ information about specific transactions
- ▶ Can validate proof-of-work
- ▶ Can't validate other network rules:
  - ▶ Can't detect invalid or double-spend transactions
  - ▶ Can't verify money supply

---

## SIMPLE PAYMENT VERIFICATION (SPV) NODE (2)

- ▶ Can verify that a transaction is included in a block (by asking for Merkle proofs)
- ▶ Can't verify that a transaction hasn't appeared in the blockchain
- ▶ Can use Bloom filters to preserve (some) privacy

---

## OTHER NODE OPTIONS

- ▶ **-blocksonly** - full node which doesn't propagate transactions
- ▶ **-nolisten** - node which makes outbound connections but doesn't accept inbound connections
- ▶ **-onion** - connect to peers using Tor
- ▶ **-proxy** - connect to peers via a proxy
- ▶ **-whitelist=<IP address or subnet>** -





---

# MESSAGE FORMAT

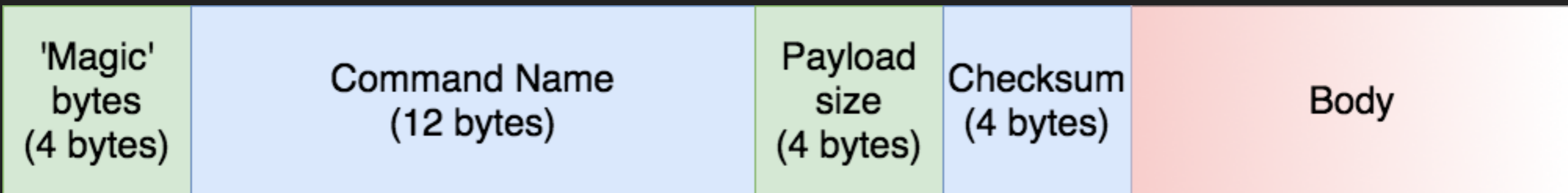
---

# MESSAGE FORMAT

- ▶ Bitcoin P2P messages contain a header and a payload
- ▶ Header is 24 bytes:
  - ▶ Magic (4 bytes): indicates the network (0xf9beb4d9 for Bitcoin)
  - ▶ Command name (12 bytes): eg ADDR, INV, BLOCK, etc
  - ▶ Payload size (4 bytes): how large the payload is in bytes
  - ▶ Checksum (4 bytes): Double SHA256 of payload
- ▶ Payload: up to 32MB. Each command has its own defined format

---

# MESSAGE FORMAT



---

## EXAMPLE HEADER

- ▶ **f9beb4d976657261636b0000000000000000000000005df6e0e2**
- ▶ **f9beb4d9** : network magic for Bitcoin main net
- ▶ **76657261636b0000000000000000** : VERACK with zero padding
- ▶ **00000000** : payload size is zero
- ▶ **5df6e0e2** : checksum  $\text{SHA256}(\text{SHA256}(""))$



---

# CONTROL MESSAGES

---

## VERSION HANDSHAKE

- ▶ P2P connection starts with a version handshake
- ▶ Used by nodes to exchange information about themselves
- ▶ A node responds to a VERSION message with VERACK

---

## VERSION MESSAGE (1)

- ▶ Version (4 bytes)
  - ▶ *highest version the transmitting node can connect to*
- ▶ Services (8 bytes)
  - ▶ *bitfield of services supported by the transmitting node*
- ▶ Timestamp (8 bytes)
  - ▶ *Unix timestamp of transmitting node*

---

## VERSION MESSAGE (2)

- ▶ `addr_rcv` services (8 bytes)
  - ▶ *services supported by the receiving node*
- ▶ `addr_rcv` IP address and port (16 + 2 bytes)
  - ▶ *IPv6 address and port of receiving node*
- ▶ `addr_trans` services (8 bytes)
  - ▶ *services supported by the transmitting node (should be same as Services field)*
- ▶ `addr_trans` IP address and port (16 + 2 bytes)
  - ▶ *IPv6 address and port of transmitting node*



---

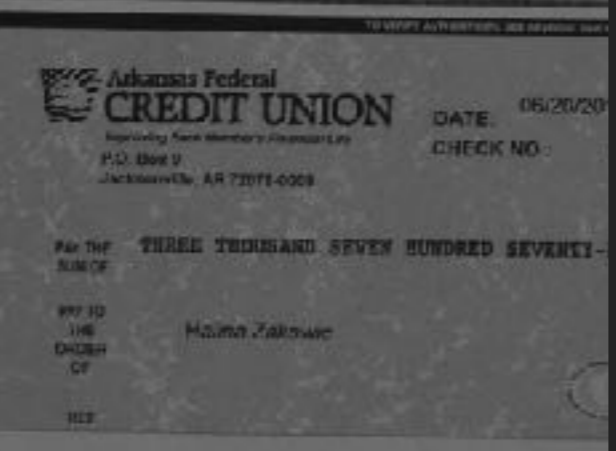
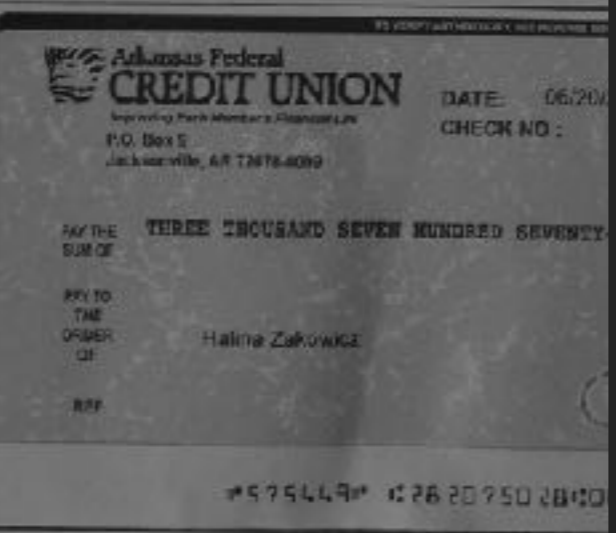
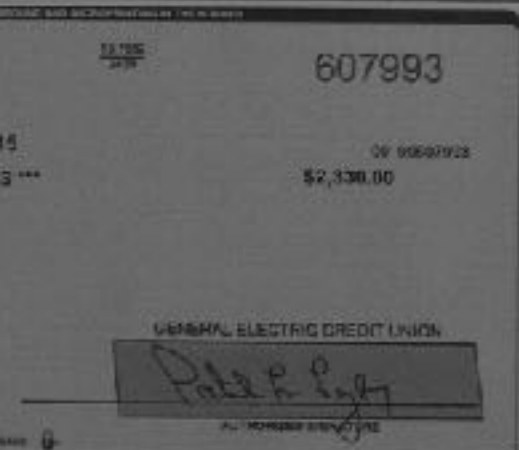
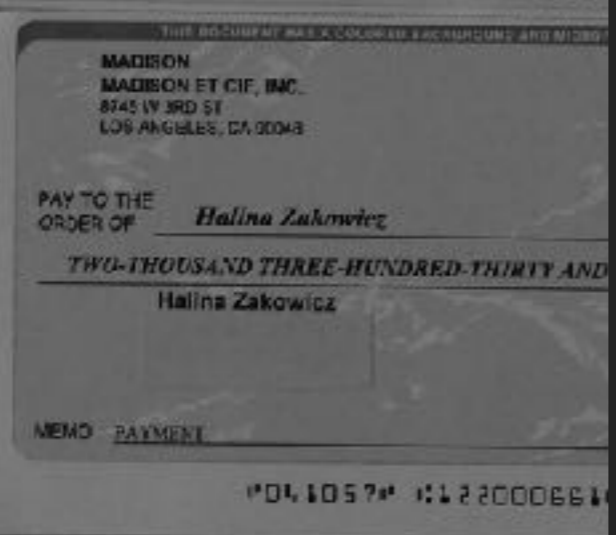
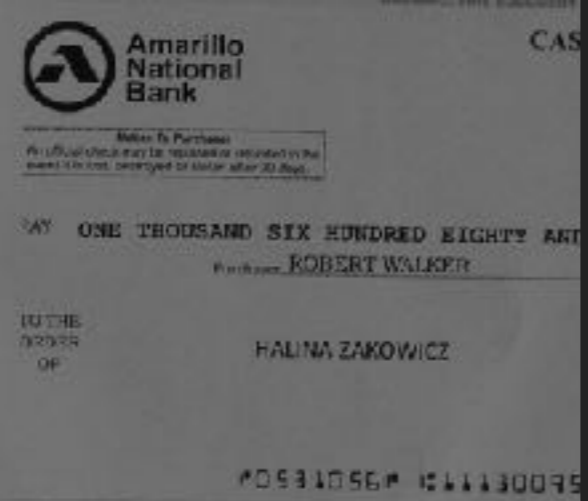
## VERSION MESSAGE (3)

- ▶ nonce (8 bytes)
  - ▶ *random number used to detect if a node is connecting to itself*
- ▶ user\_agent (compactSize + len)
  - ▶ *string indicating the software the node is using*
- ▶ start\_height (4 bytes)
  - ▶ *height of the transmitting node's best blockchain*
- ▶ relay (bool - optional)
  - ▶ *indicates whether **INV** or **TX** messages should be sent to the transmitting node*

---

## OTHER CONTROL MESSAGES

- ▶ VERACK - sent in response to a VERSION message
- ▶ ADDR - gossips connection information about other nodes
- ▶ GETADDR - requests information about other nodes
- ▶ PING/PONG - confirms connectivity
- ▶ FILTERLOAD / FILTERADD / FILTERCLEAR - sets and unsets bloom filters for SPV transaction propagation



TRANSACTION PROPAGATION

---

## INVENTORY ANNOUNCEMENT

- ▶ New transactions are announced in an INV message
- ▶ INV messages contain the txid
  - ▶ (Can also contain block hashes)
- ▶ If the receiving node wants the announced inventory, it responds with a GETDATA message
- ▶ The announcing node then sends a TX messages

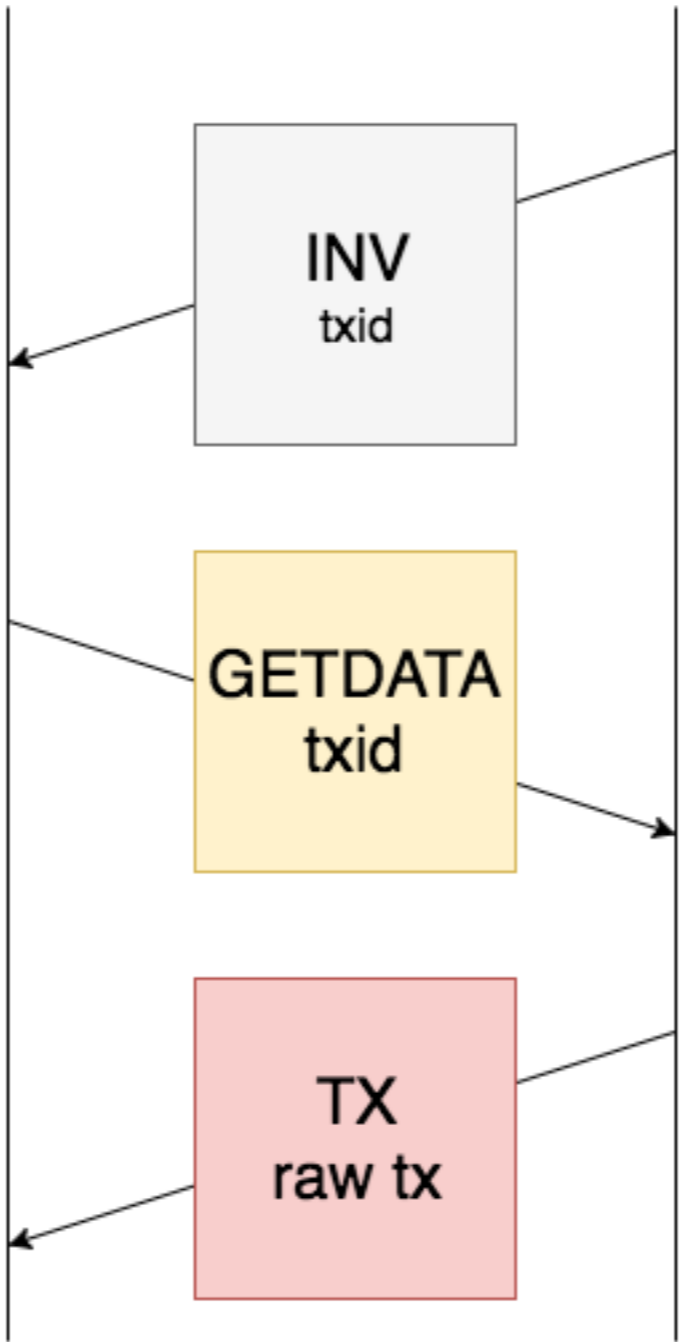
Receiving  
Node

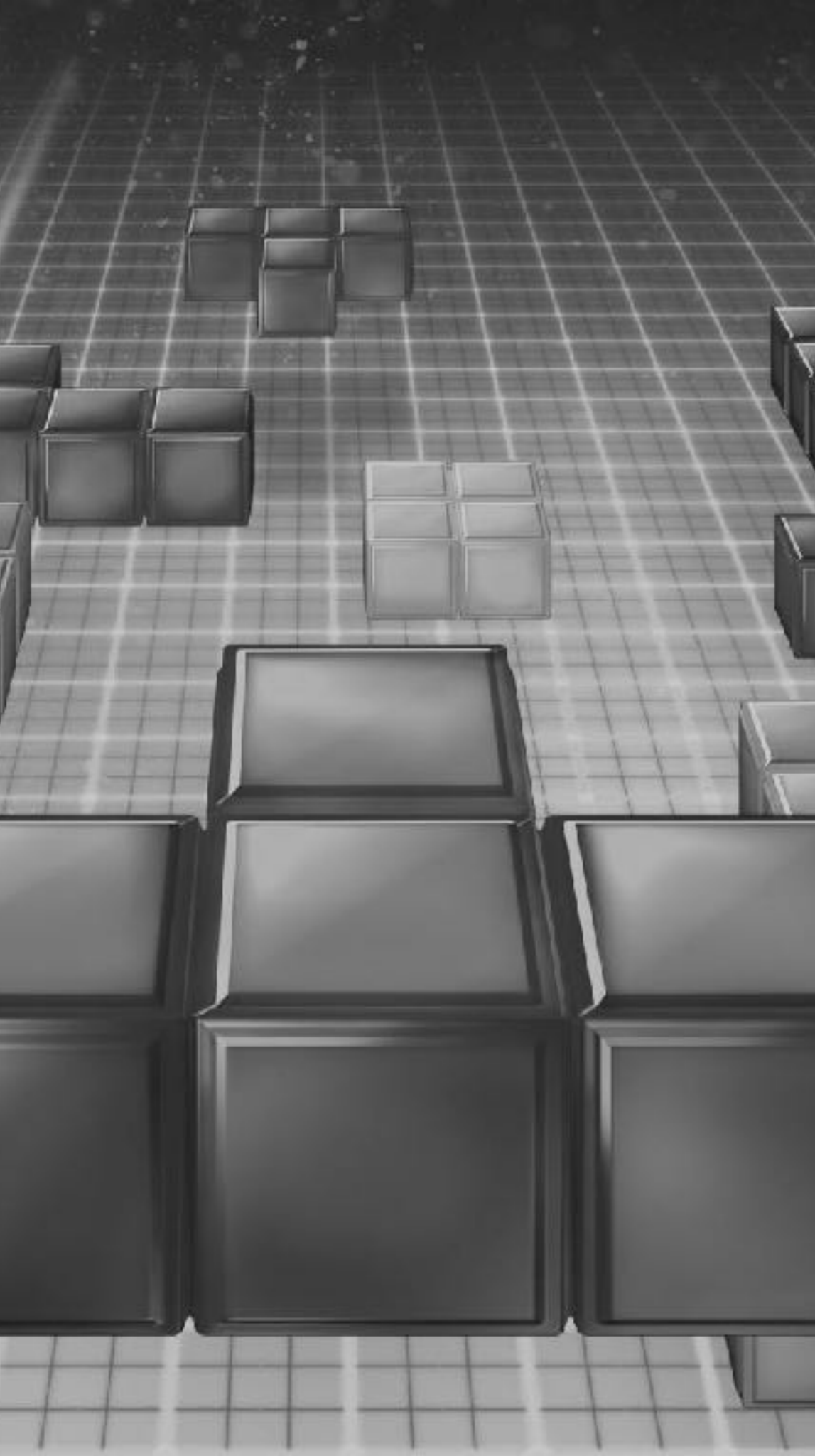
Transmitting  
Node

INV  
txid

GETDATA  
txid

TX  
raw tx





---

# BLOCK PROPAGATION

---

## BLOCK PROPAGATION

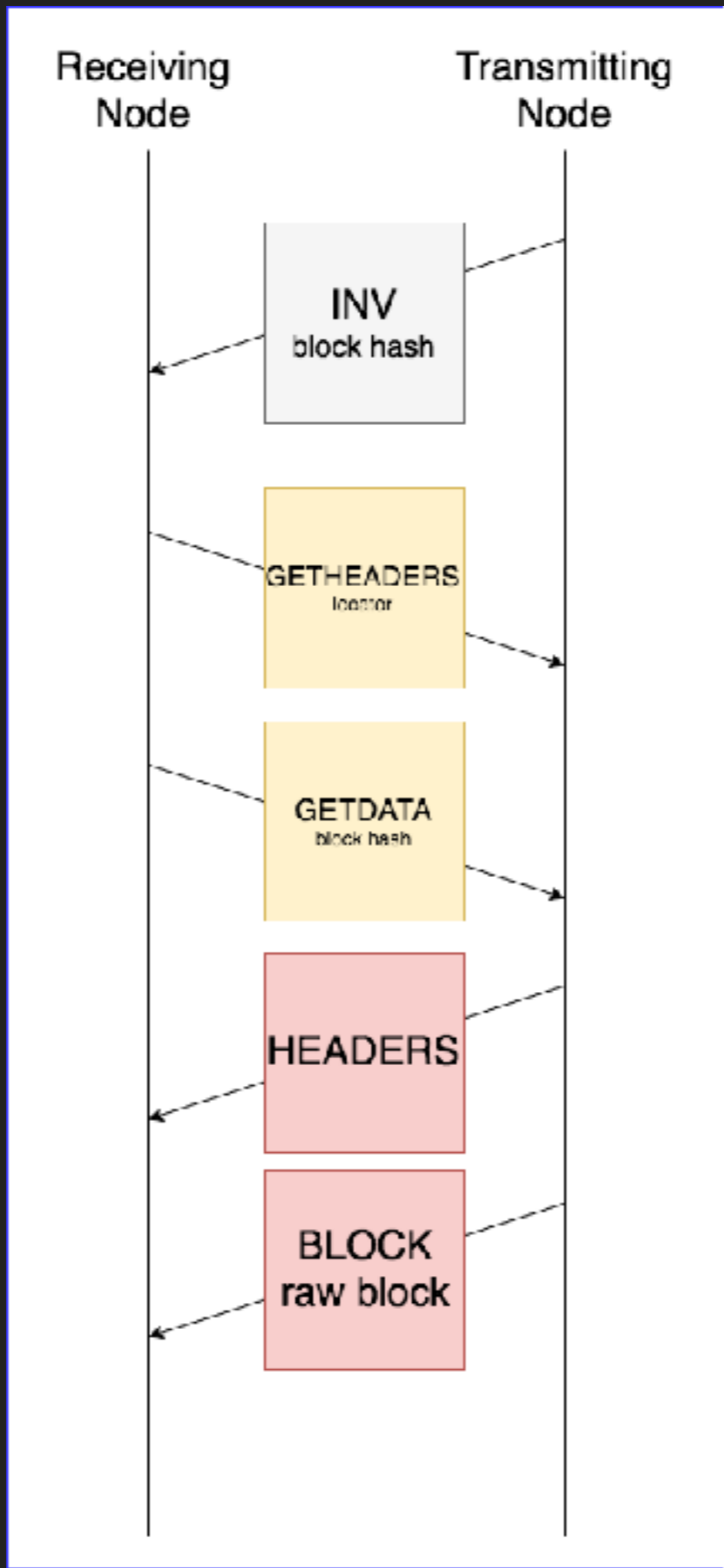
- ▶ Originally, blocks were propagated using INV-GETDATA-BLOCK
- ▶ v0.10.0 introduced 'headers first' syncing
- ▶ v0.12.0 introduced the SENDHEADERS message
- ▶ v0.13.0 introduced compact blocks
- ▶ v0.14.0 introduced High Bandwidth compact blocks

---

## HEADERS-FIRST SYNCING

- ▶ Transmitting node sends INV with block hash as normal
- ▶ Receiving node responds with:
  - ▶ GETHEADERS (for block headers up to the tip)
  - ▶ GETDATA (for the tip block)
- ▶ Transmitting node sends:
  - ▶ HEADERS (connecting tip to the receiving node's best block)
  - ▶ BLOCK (containing the tip block)





---

# SENDHEADERS

- ▶ SENDHEADERS is a new control message in protocol version 70012
- ▶ Sent immediately after VERSION handshake
- ▶ Indicates that the transmitting node would prefer to receive HEADERS messages instead of INVs
- ▶ Saves a INV-GETHEADERS round trip
- ▶ Defined in BIP 130

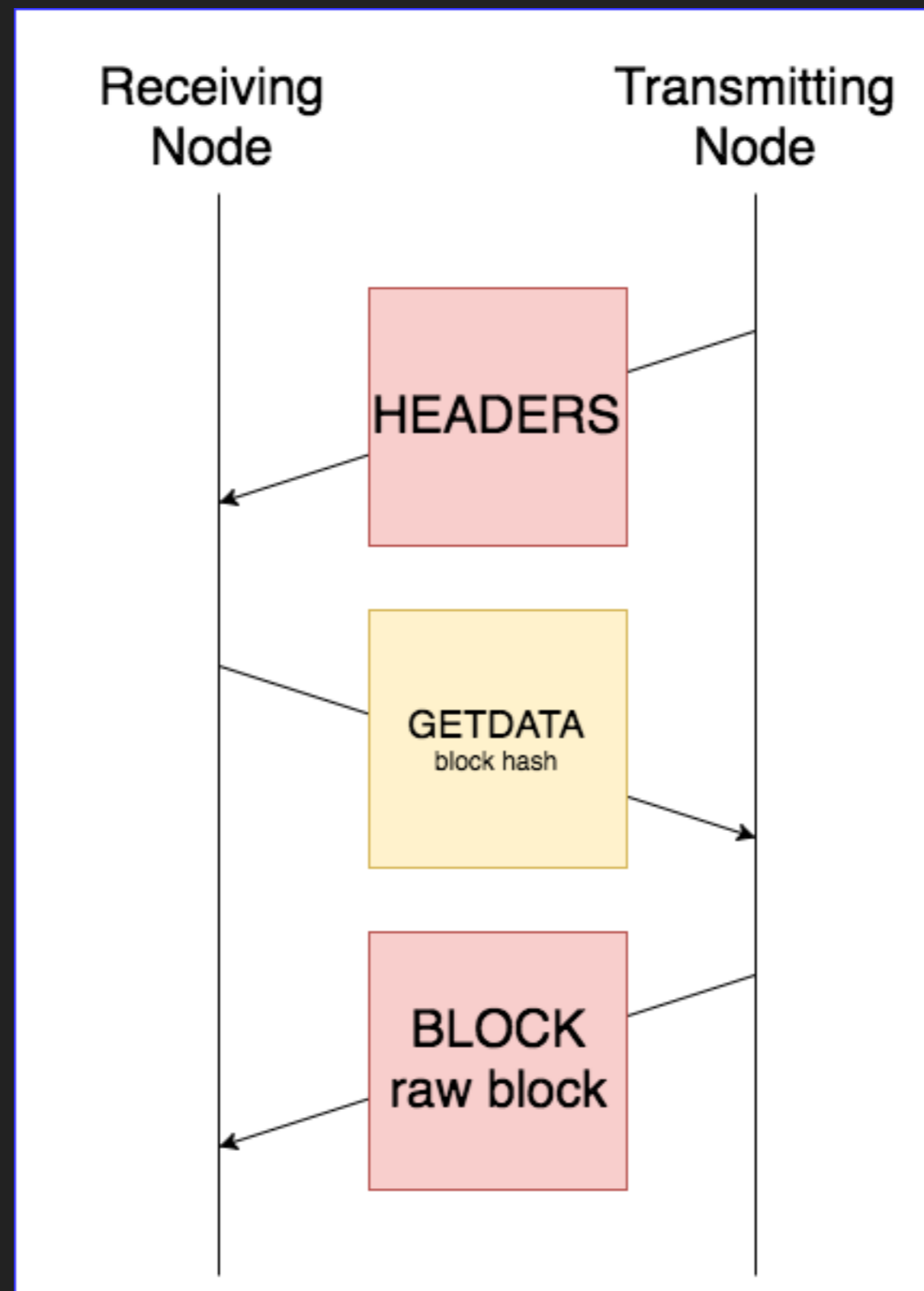
Receiving  
Node

Transmitting  
Node

HEADERS

GETDATA  
block hash

BLOCK  
raw block



---

## COMPACT BLOCKS (1)

- ▶ Reduces time and bandwidth for propagating blocks
- ▶ Relies on fact that peer has already seen most transactions in a new block
- ▶ Enabled by node sending a SENDCMPCT message (similar to SENDHEADERS)
- ▶ Defined in BIP 152

---

## COMPACT BLOCKS (2)

- ▶ Two modes:
  - ▶ low bandwidth - same number of messages as headers first block syncing, but saves on number of transactions sent
  - ▶ high bandwidth - sends cmpctblock message before the block has even been validated

Transmitting Node

Receiving Node

