BLOCKCHAIN UNDER THE HOOD

TECHNICAL MASTERCLASS

PRESENTED BY

neuroware

NEUROWARE - MEET THE FOUNDERS

MSC MALAYSIA Status Company

Mark Smalley - CEO

Living in Malaysia for the past 20 years

Building Fintech Solutions for 15+ years

Spent 10 years building tech communities

Building blockchain apps for 5+ years

Ruben Tan - CTO

More than 10+ years of software engineering exp

Active community evangelist & technology speaker

Early developer in MyTeksi, OnApp, Bookya, etc

Studied distributed consensus as a hobby

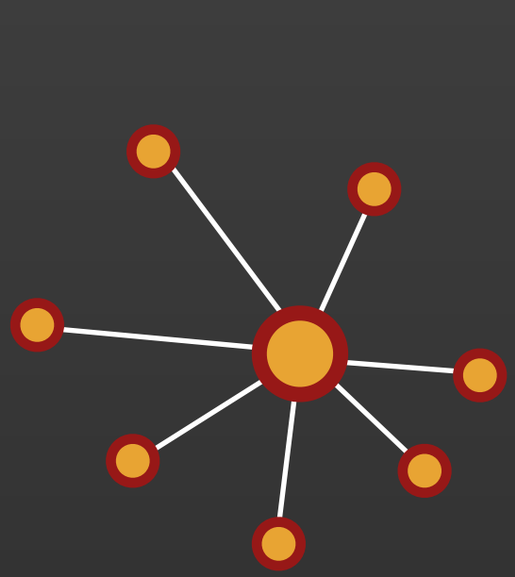
DISTRIBUTED SYSTEMS

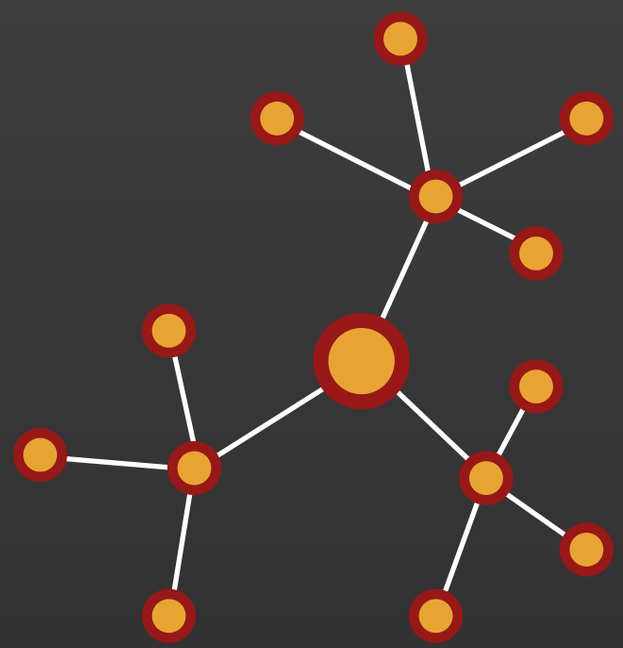
Definition and general traits

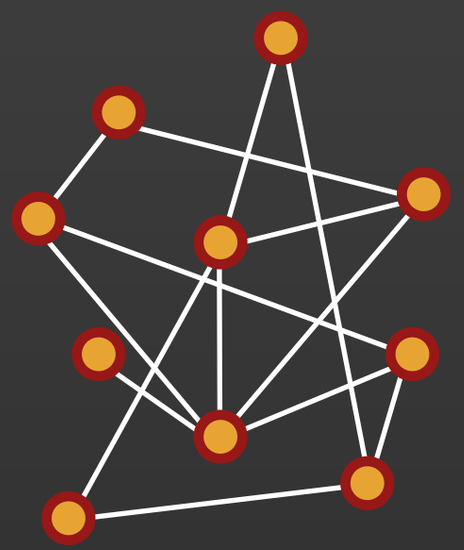
DISTRIBUTED SYSTEMS OVERVIEW

- What we will cover:
  - Centralised vs. Decentralised vs. Distributed
  - Phonebook Exercise
  - CAP Theorem Review
  - Distributed Consensus & Byzantine General Problem
  - Fallacies of Distributed Computing
  - Two Phase Commit
  - Three Phase Commit
  - Paxos (Basic)

EVOLUTION OF TOPOLOGIES/BUSINESS MODELS



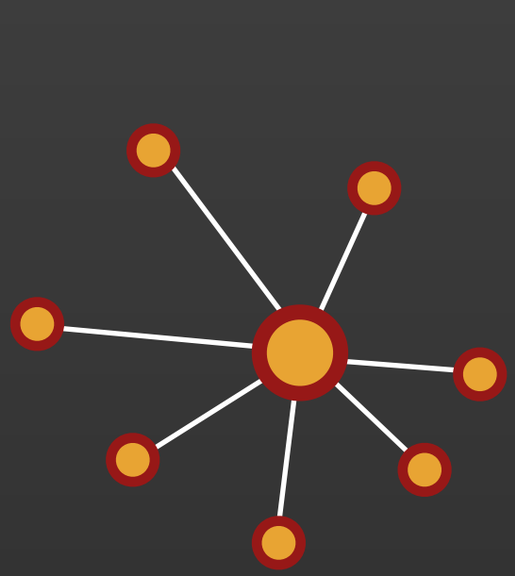


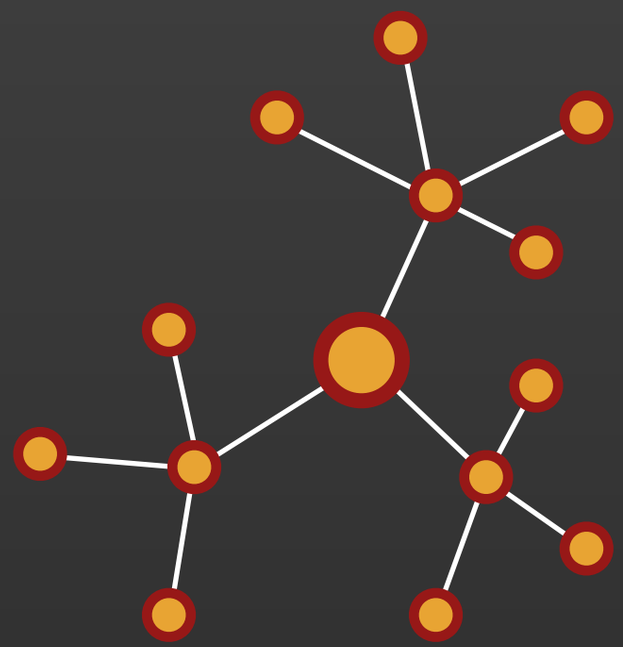


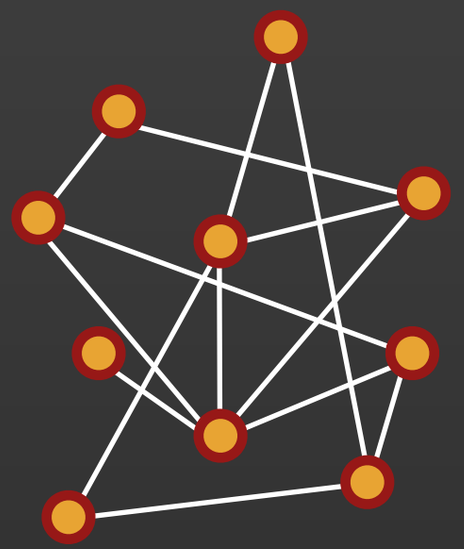
Centralised	Cloud	Distributed
• Single point of failure • Expensive to scale	• Basically outsourced infrastructure • Mitigates failure • Requires trust	• More secure as more nodes exist/join • Highly resilient to data loss and attacks

Evolution →

EVOLUTION OF BUSINESS MODELS



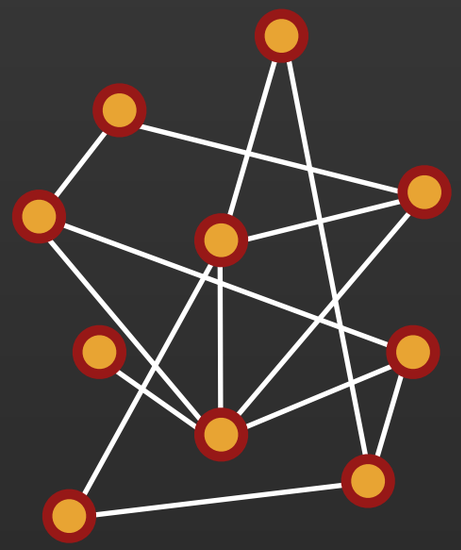




Centralised	Cloud	Distributed
Telephone Service	Whatsapp	Firechat
Walmart	Amazon	Open Bazaar
Taxi Services	Uber	Lazooz

Evolution →

DISTRIBUTED TECHNOLOGIES



- **Pros**
  - The more participating nodes, the more durable the network is
  - Harder to achieve 100% data loss due to massive redundancy
  - Can still function under partial availability or split network
  - Can distribute computing power to save costs
  - Usually less infrastructure to manage and control
  - Every participant has equal power (or less variance in power)
- **Cons**
  - Lack of central authority means harder to make big changes
  - Requires a consensus of some kind to maintain order
  - Does not have a good reputation (bittorrent, bitcoin, etc)

THE PHONE BOOK EXERCISE

Understanding the difficulties of a distributed system

PHONE BOOK EXERCISE

WE HAVE A NEIGHBOURHOOD

How do we connect businesses to customers?

PHONE BOOK EXERCISE

WE HAVE A NEIGHBOURHOOD

How do we connect businesses to customers?

1. Start a personal phonebook containing contact information of everybody

PHONE BOOK EXERCISE

WE HAVE A NEIGHBOURHOOD

How do we connect businesses to customers?

1. Start a personal phonebook containing contact information of everybody
2. Share the phonebook with neighbours

PHONE BOOK EXERCISE

WE HAVE A NEIGHBOURHOOD

How do we connect businesses to customers?

1. Start a personal phonebook containing contact information of everybody
2. Share the phonebook with neighbours

PHONE BOOK EXERCISE

WE HAVE A NEIGHBOURHOOD

How do we connect businesses to customers?

1. Start a personal phonebook containing contact information of everybody
2. Share the phonebook with neighbours
3. What happens when a new entry is added to your book?

PHONE BOOK EXERCISE

WE HAVE A NEIGHBOURHOOD

How do we connect businesses to customers?

1. Start a personal phonebook containing contact information of everybody
2. Share the phonebook with neighbours
3. What happens when a new entry is added to your book?
4. Share the new entry with this neighbour

PHONE BOOK EXERCISE

WE HAVE A NEIGHBOURHOOD

How do we connect businesses to customers?

1. Start a personal phonebook containing contact information of everybody
2. Share the phonebook with neighbours
3. What happens when a new entry is added to your book?
4. Share the new entry with this neighbour
5. Neighbour must then share the same entry with somebody else

PHONE BOOK EXERCISE

WE HAVE A NEIGHBOURHOOD

How do we connect businesses to customers?

1. Start a personal phonebook containing contact information of everybody
2. Share the phonebook with neighbours
3. What happens when a new entry is added to your book?
4. Share the new entry with this neighbour
5. Neighbour must then share the same entry with somebody else
6. If multiple participant try to add new entries, how do you know who has the latest information?

PHONE BOOK EXERCISE

WE HAVE A NEIGHBOURHOOD

How do we connect businesses to customers?

1. Start a personal phonebook containing contact information of everybody
2. Share the phonebook with neighbours
3. What happens when a new entry is added to your book?
4. Share the new entry with this neighbour
5. Neighbour must then share the same entry with somebody else
6. If multiple participant try to add new entries, how do you know who has the latest information?
7. If somebody adds an invalid phone number, how do you know?

PHONE BOOK EXERCISE

WE HAVE A NEIGHBOURHOOD

How do we connect businesses to customers?

1. Start a personal phonebook containing contact information of everybody
2. Share the phonebook with neighbours
3. What happens when a new entry is added to your book?
4. Share the new entry with this neighbour
5. Neighbour must then share the same entry with somebody else
6. If multiple participant try to add new entries, how do you know who has the latest information?
7. If somebody adds an invalid phone number, how do you know?
8. Who has control of the phone book and the contents within?

THE PHONE BOOK EXERCISE

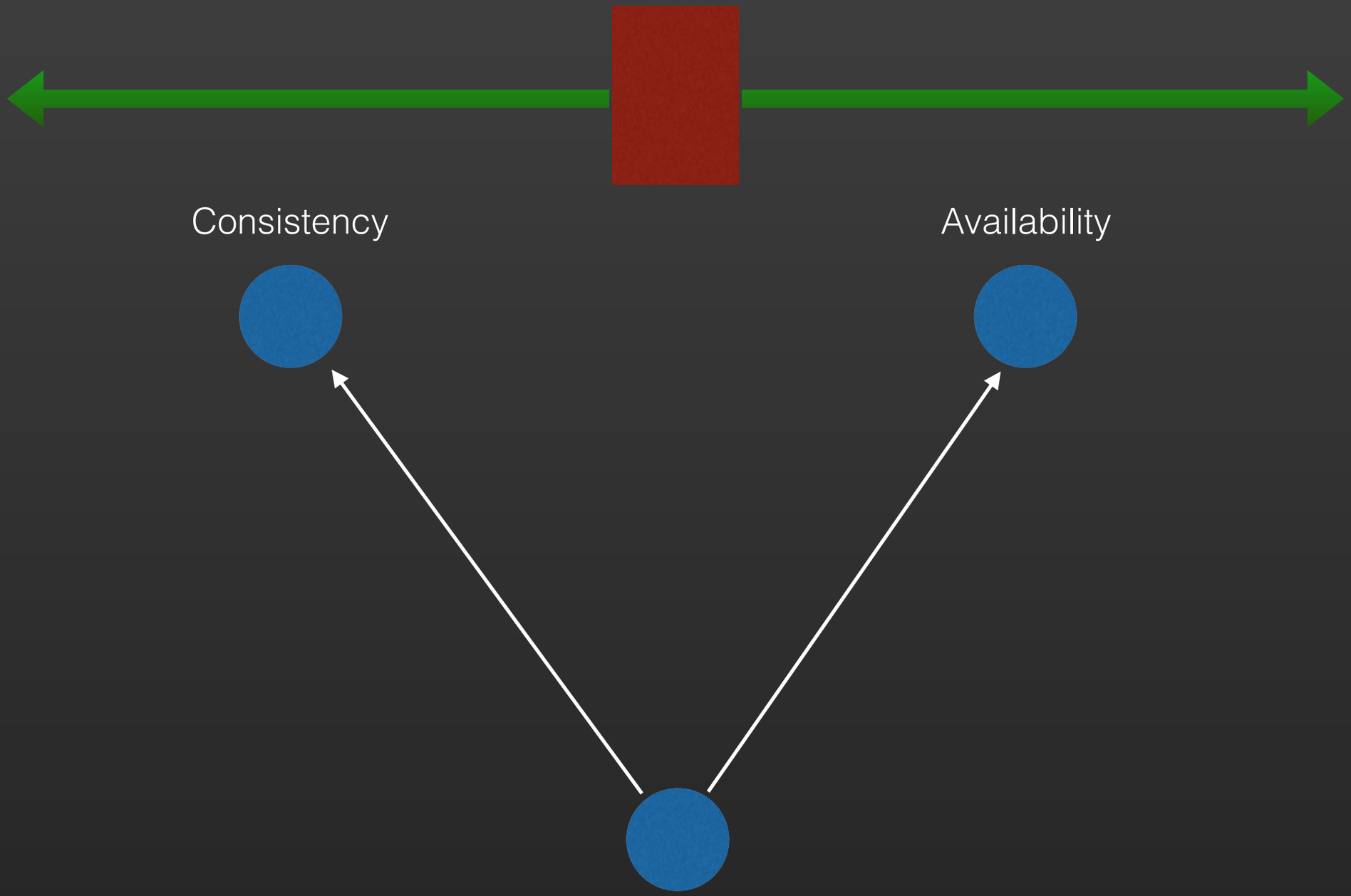
So, why not just use a centralised solution?

POWER OF DISTRIBUTED TECHNOLOGY

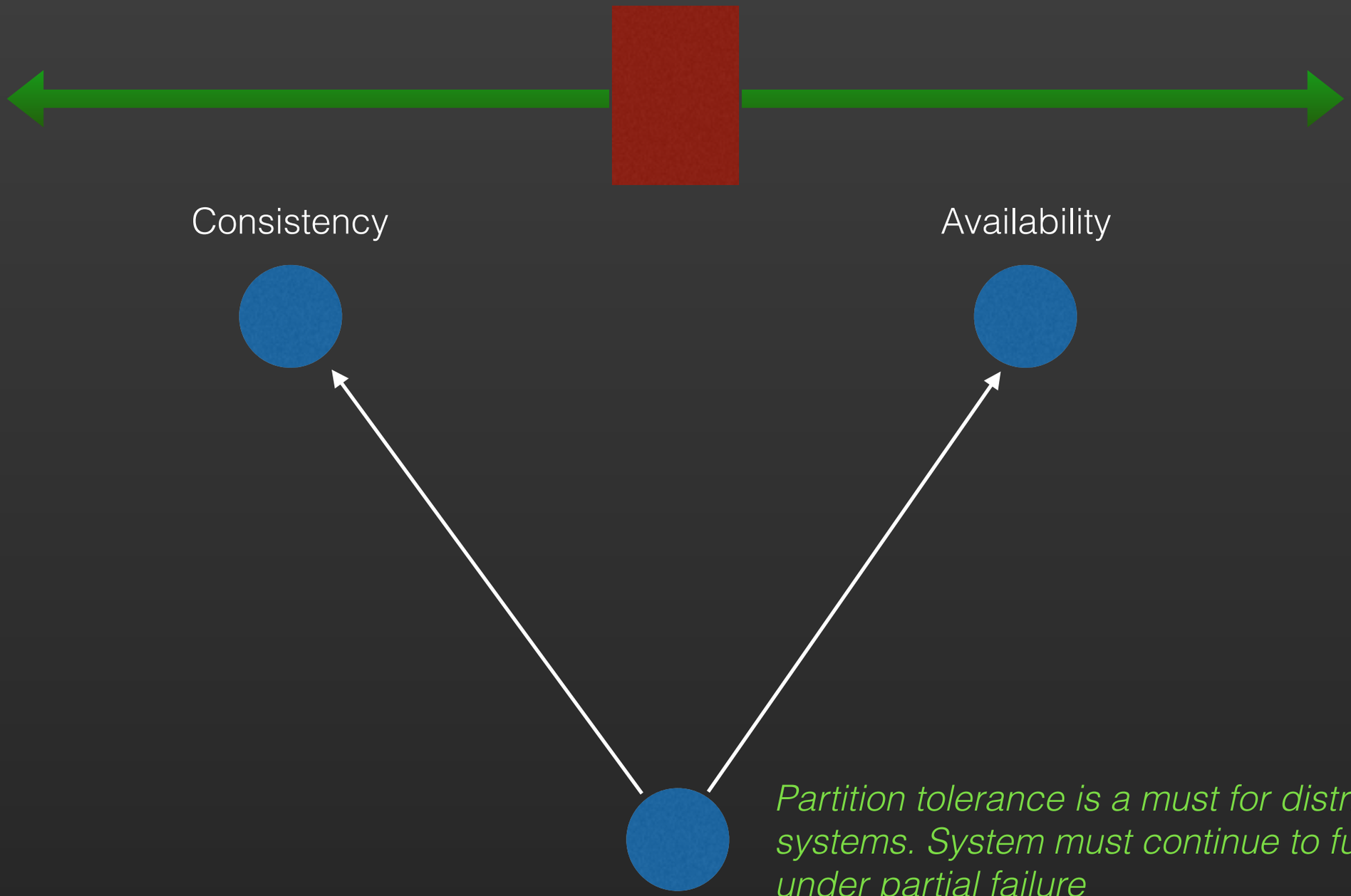
Why not a centralised solution?

- Anybody can copy a phonebook from a peer, and receive immutable updates frequently
- No need to wait for a centralised body to update information
- No need to contact a centralised body to add information
- No need to trust human instrument or deviate from protocol to update the phonebook
- Phonebook becomes a neutral platform controlled by nobody
- Code is law, protocol is gospel

CONSENSUS IS EVERYTHING

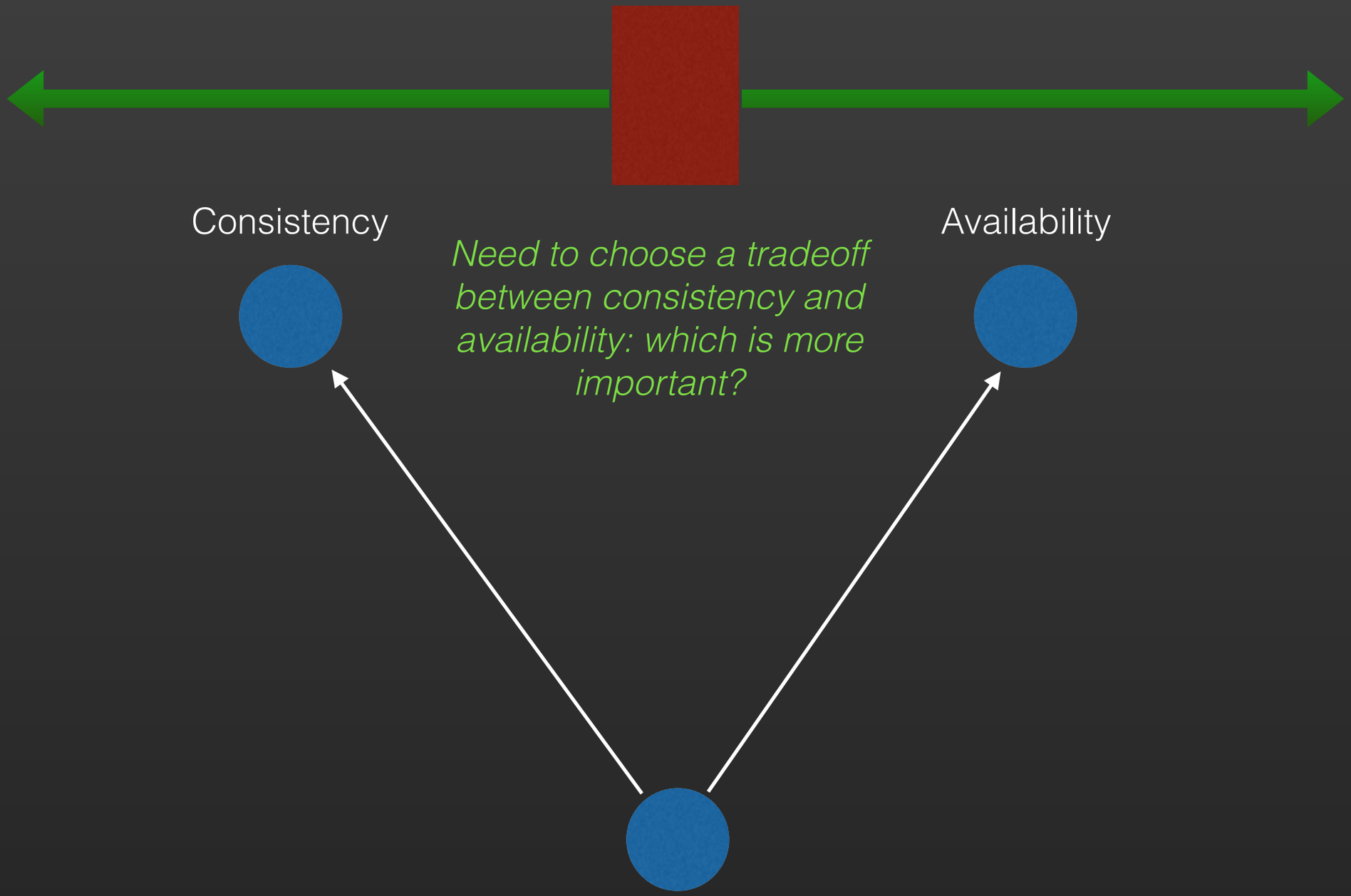


CONSENSUS IS EVERYTHING

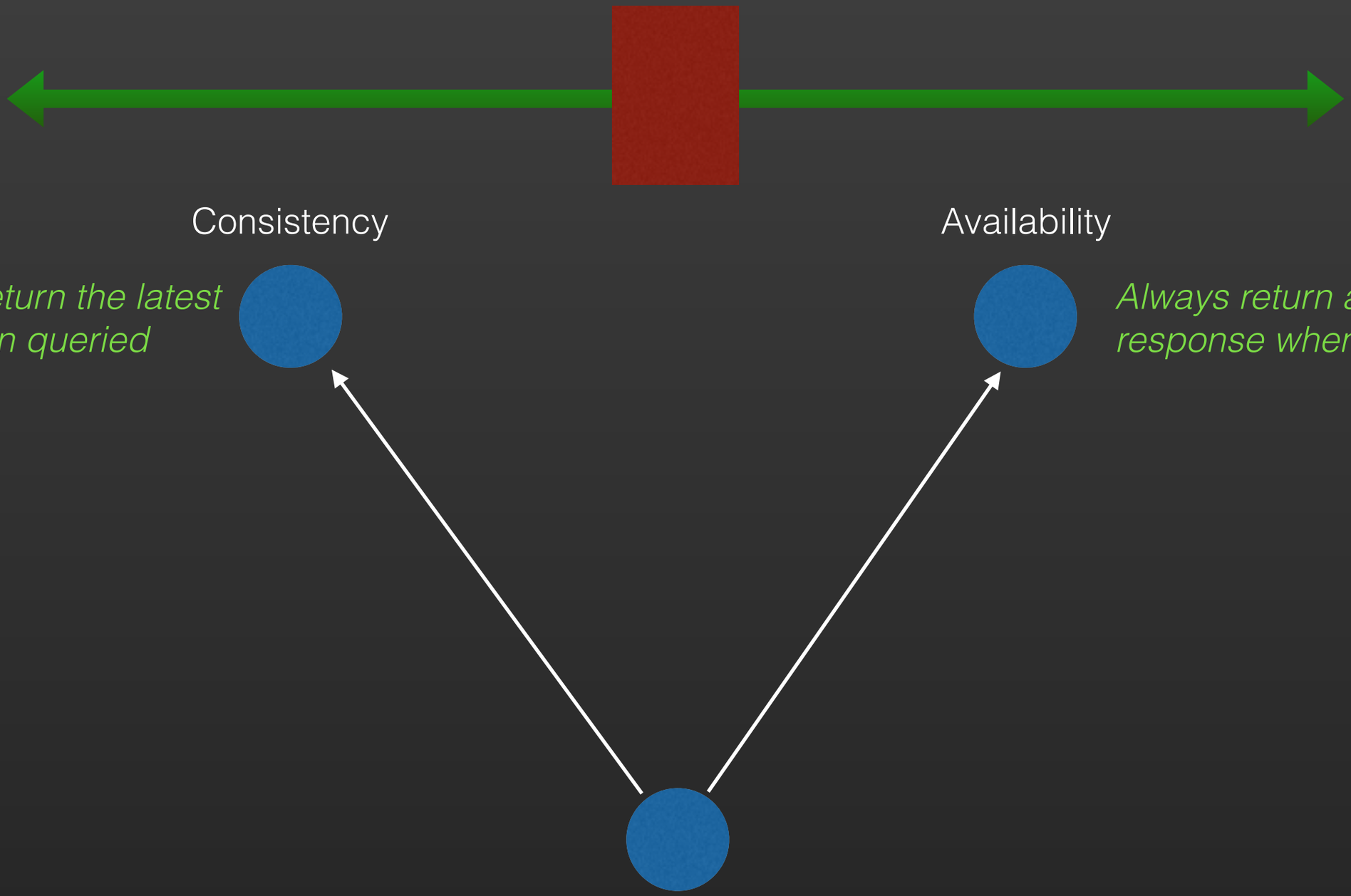


Partition tolerance is a must for distributed systems. System must continue to function under partial failure

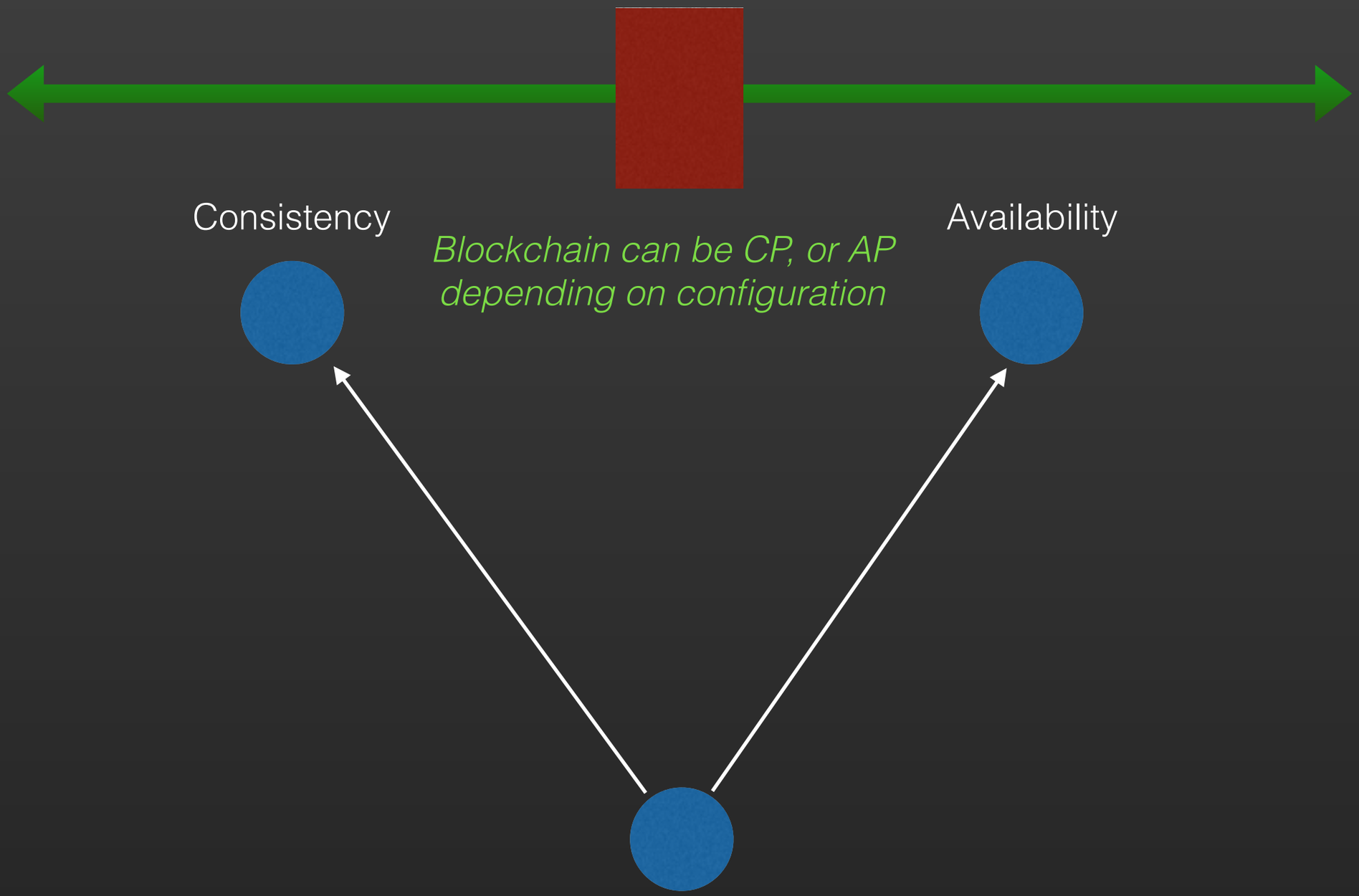
CONSENSUS IS EVERYTHING



CONSENSUS IS EVERYTHING



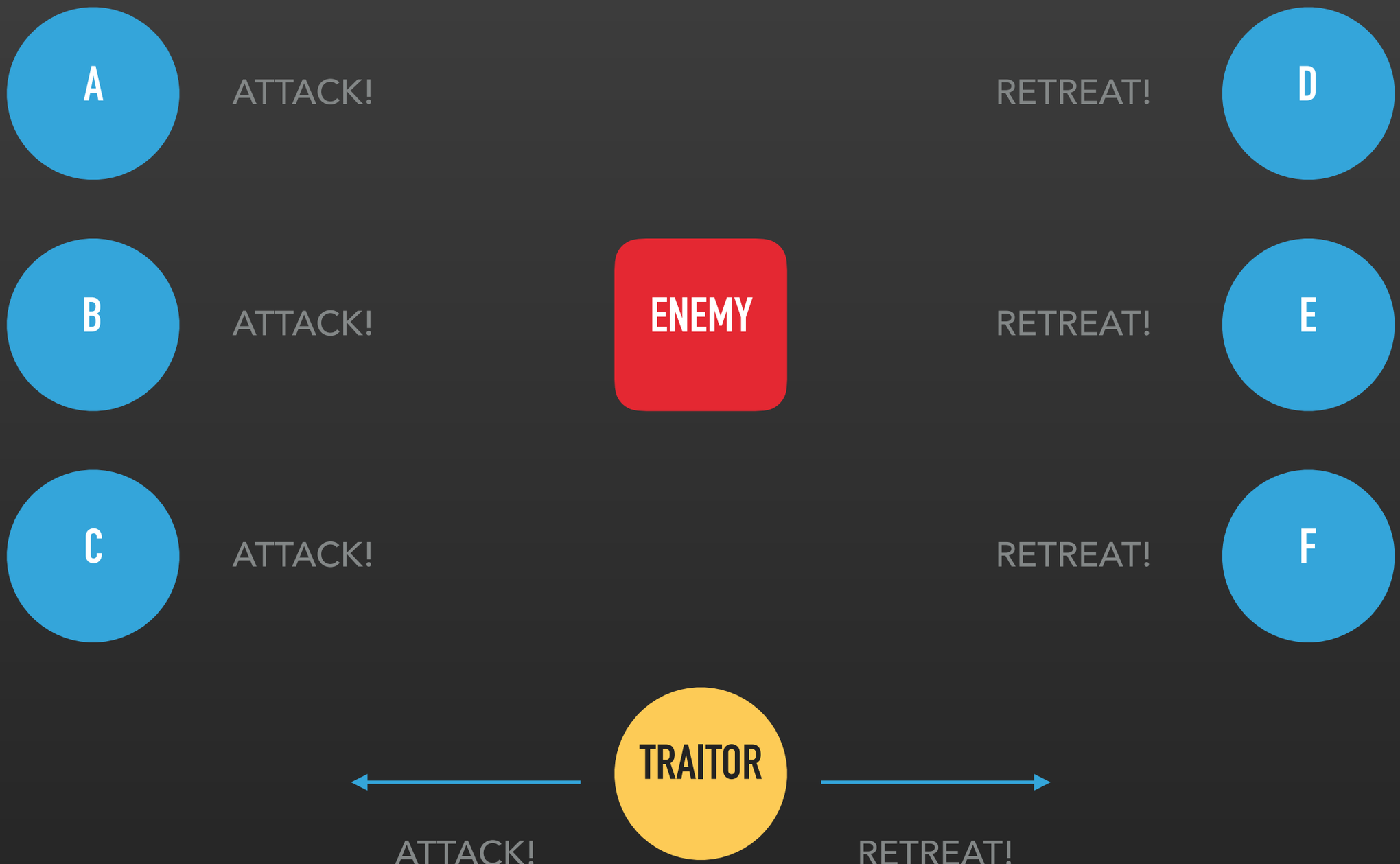
CONSENSUS IS EVERYTHING



BYZANTINE GENERAL'S PROBLEM

- Consensus algorithms ensure the **consistency** of a distributed system
- Failure modes
  - **Fail-stop** - a node dies and does not recover
  - **Fail-recover** - a node dies and recovers some time later
  - **Byzantine** - a node misbehaves or displays unexpected behavior
- Byzantine general's problem
  - One of the first impossibility proof in computer communications
  - Impossibility proof = impossible to solve in a perfect manner
  - Originated from the **Two General's Problem** (1975)
  - Explored in detail in the **Byzantine General Problem** (1982)
  - Remains as the benchmark in evaluating distributed consensus algorithms today
- Common terms
  - **Byzantine fault** - any fault that presents different symptoms to different observers
  - **Byzantine failure** - loss of system service reliant on consensus due to byzantine fault
  - **Byzantine fault tolerance** - a system that is resilient against byzantine faults

BYZANTINE GENERAL'S PROBLEM

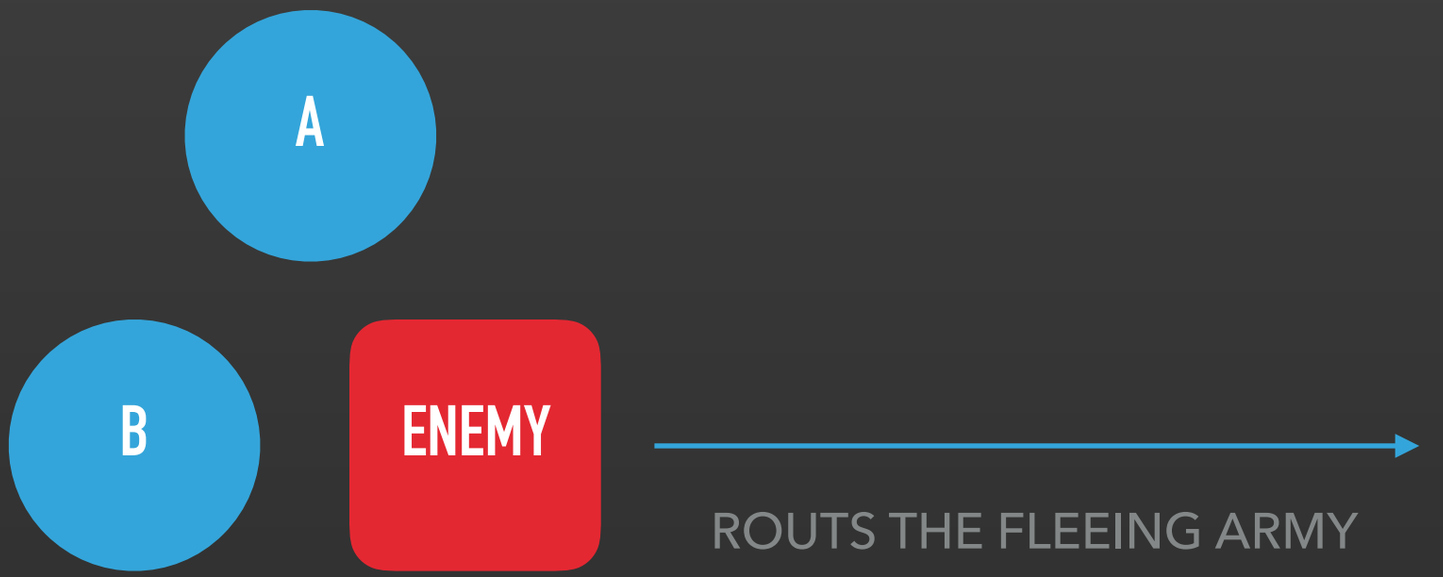


BYZANTINE GENERAL'S PROBLEM

ATTACKERS HAVE

INSUFFICIENT FORCE

AND ARE DESTROYED



MUAHAHA, NO CONSENSUS!

DISTRIBUTED SYSTEMS ARE HARD

- Distributed computing is inherently **unreliable**
  - Peter Deutsch, Bill Joy, Tom Lyon, and James Gosling - **The Eight Fallacies of Distributed Computing** (1994-1997)

- Today many software engineers still believe in some or all of the fallacies:
  - The network is reliable
  - Latency is zero
  - Bandwidth is infinite
  - The network is secure
  - Topology doesn't change
  - There is only one administrator
  - Transport cost is zero
  - The network is homogeneous

DISTRIBUTED SYSTEMS ARE HARD

When you believe in any of the eight fallacies...

IMPOSSIBILITY PROOF

- Distributed systems achieve **consensus** when all nodes are acting as **one entity**

- Michael J. Fisher, Nancy A. Lynch, and Michael S. Patterson - **Impossibility of Distributed Consensus with One Faulty Process** (1985, Dijkstra Award winner, 2001)
  - Also known as the **FLP Impossibility Proof**
  - Consensus is **mathematically impossible** - tradeoffs instead of solutions

IMPOSSIBILITY PROOF

- Distributed systems achieve **consensus** when all nodes are acting as **one entity**

- Michael J. Fisher, Nancy A. Lynch, and Michael S. Patterson - **Impossibility of Distributed Consensus with One Faulty Process** (1985, Dijkstra Award winner, 2001)
  - Also known as the **FLP Impossibility Proof**
  - Consensus is **mathematically impossible** - tradeoffs instead of solutions

CONSENSUS ALGORITHMS

- Algorithm is a “formula” to allow a system to operate without human interference to solve problems
- **Consensus algorithm** allows a distributed system to agree on a value as the truth
- A good algorithm must have:
  - **Termination** - all nodes eventually decide on a value
  - **Agreement** - all nodes must agree on the same value
  - **Integrity** - values must be proposed by a node and not a default value
  - **Validity** - if a value is proposed and accepted, all nodes must accept the same value
- Common consensus algorithms:
  - **Two phase commit** - we will do a demo in a moment
  - **Three phase commit**
  - **Paxos**
  - **Proof-of-work** - we will be going into detail in the next segment

TWO PHASE COMMIT

Simplest consensus algorithm

TWO PHASE COMMIT

Coordinator

Responsible for sending out commit requests

Cohort

Responsible for processing requests and replying commit/ abort

Cohort

Cohort

TWO PHASE COMMIT

*Protocol consists of 2 phases: the **voting** phase, and the **commit** phase.*

Coordinator

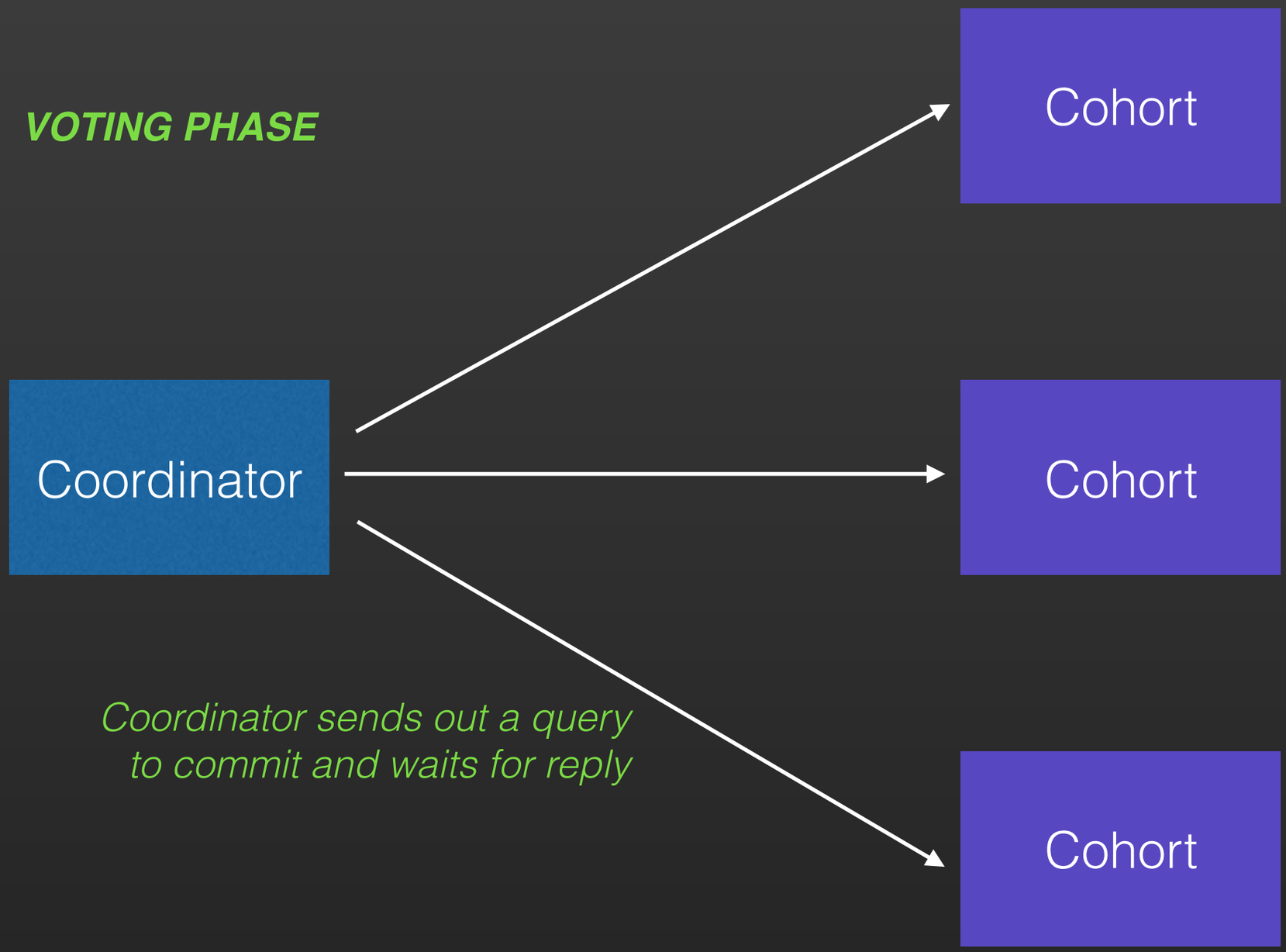
Cohort

Cohort

Cohort

TWO PHASE COMMIT

VOTING PHASE



Coordinator sends out a query to commit and waits for reply

TWO PHASE COMMIT

VOTING PHASE

Coordinator

Cohort

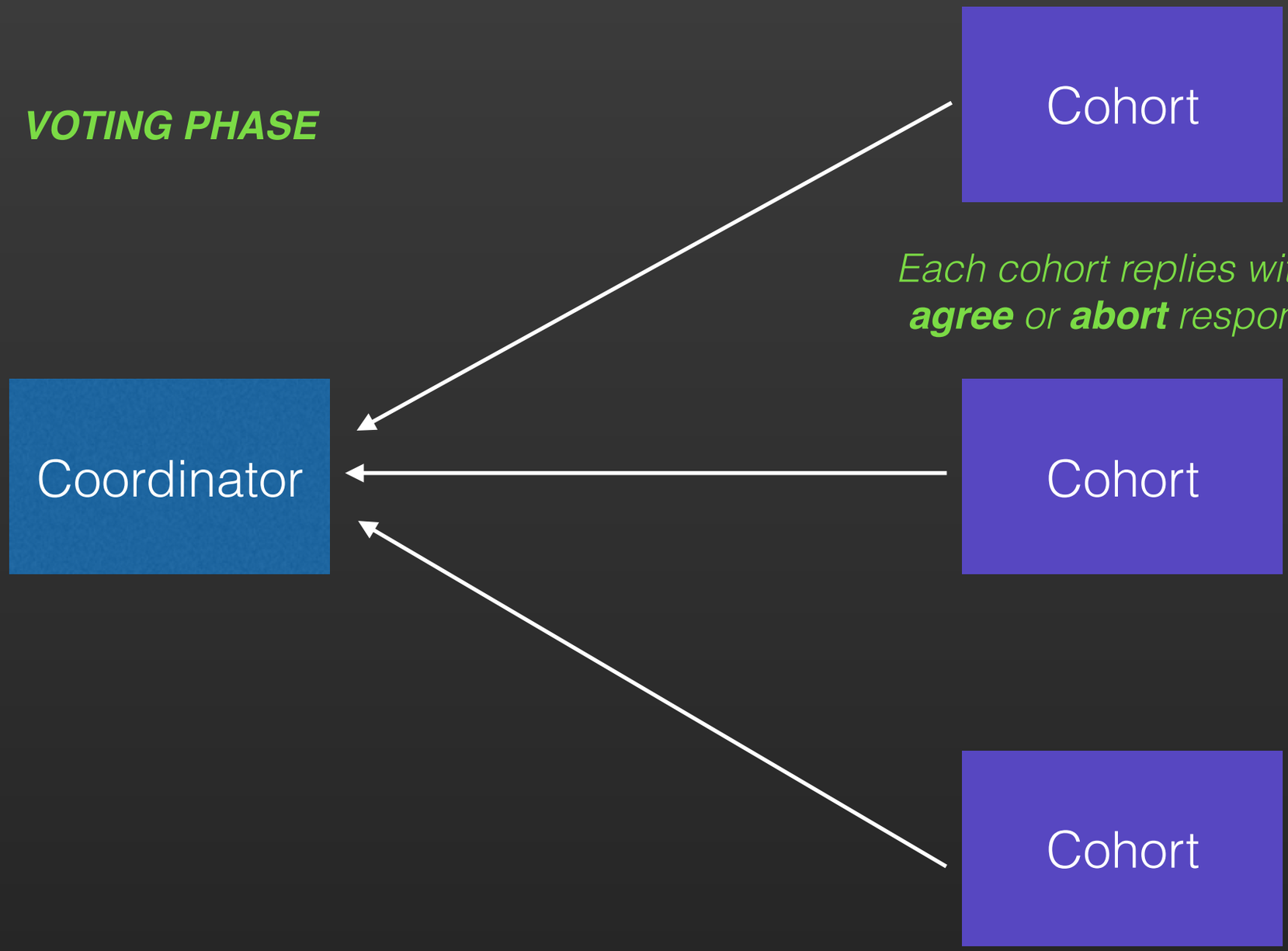
Cohorts processes message and performs work

Cohort

Cohort

TWO PHASE COMMIT

VOTING PHASE



TWO PHASE COMMIT

COMMIT PHASE

Coordinator

Cohort

Cohort

Cohort

TWO PHASE COMMIT

COMMIT PHASE

Coordinator

*If **all** cohort **agrees**, send a **commit** message*

Cohort

Agree

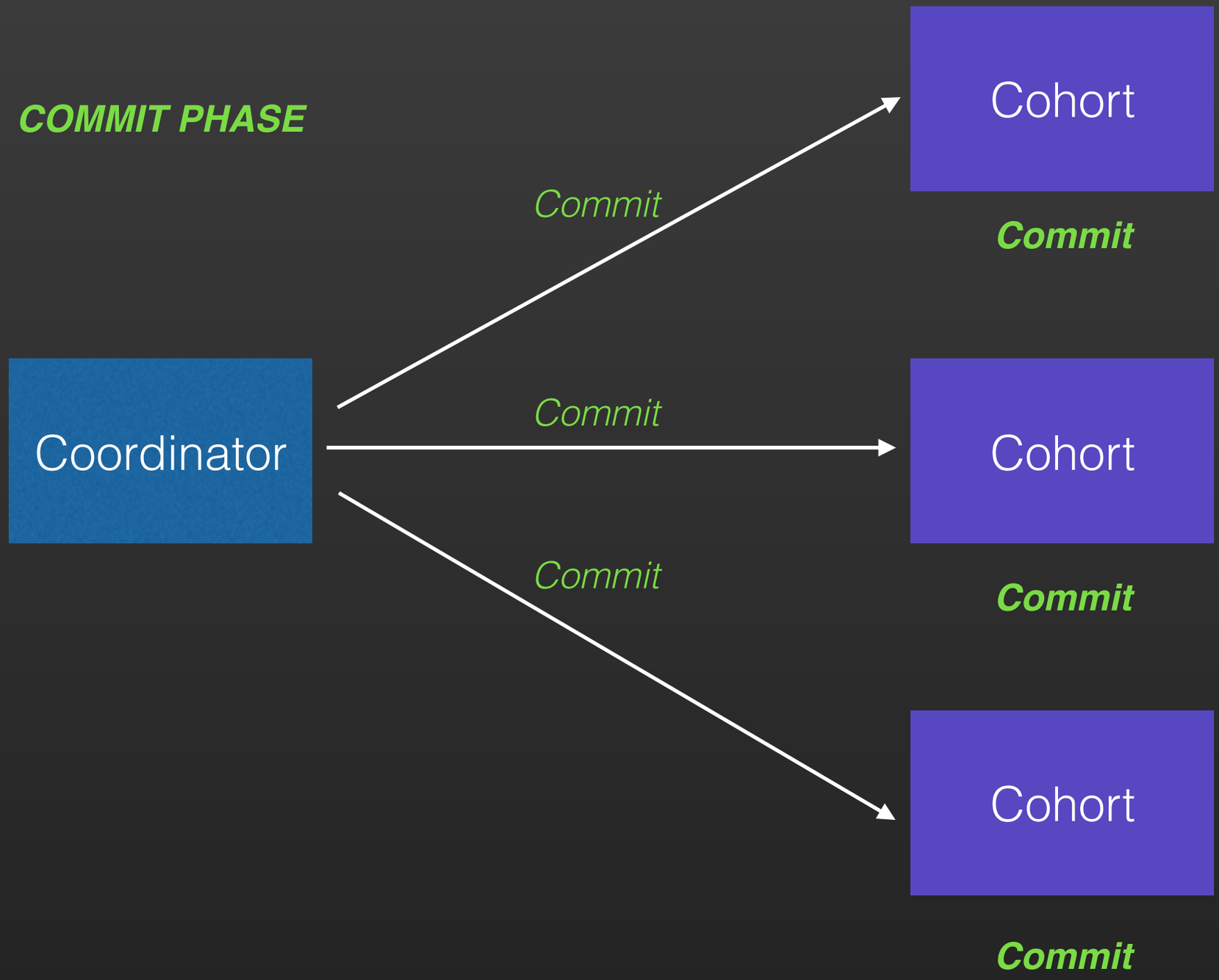
Cohort

Agree

Cohort

Agree

TWO PHASE COMMIT



TWO PHASE COMMIT

COMMIT PHASE

Coordinator

*If **any** cohort **aborts**, send a **abort** message*

Cohort

Agree

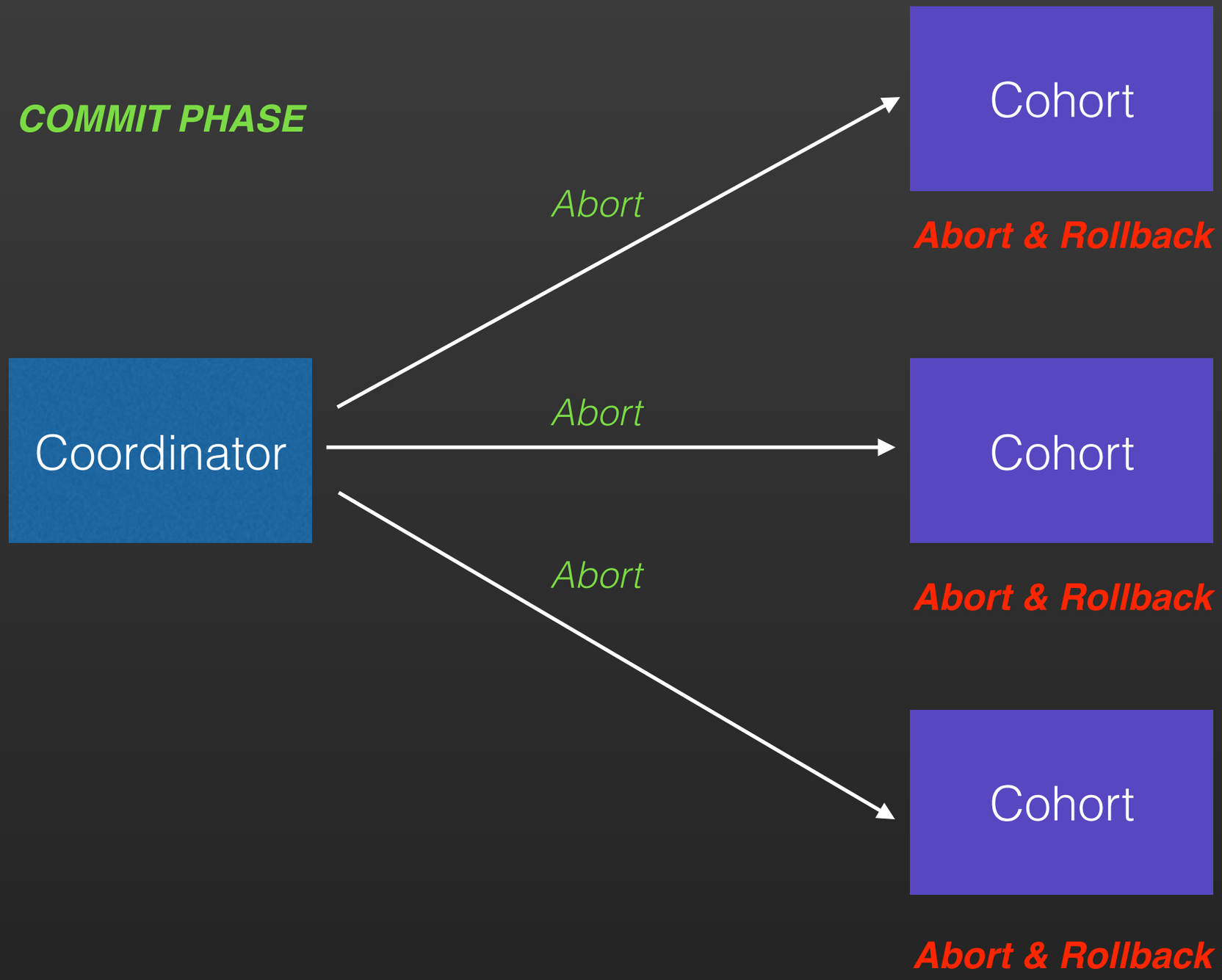
Cohort

Abort

Cohort

Agree

TWO PHASE COMMIT



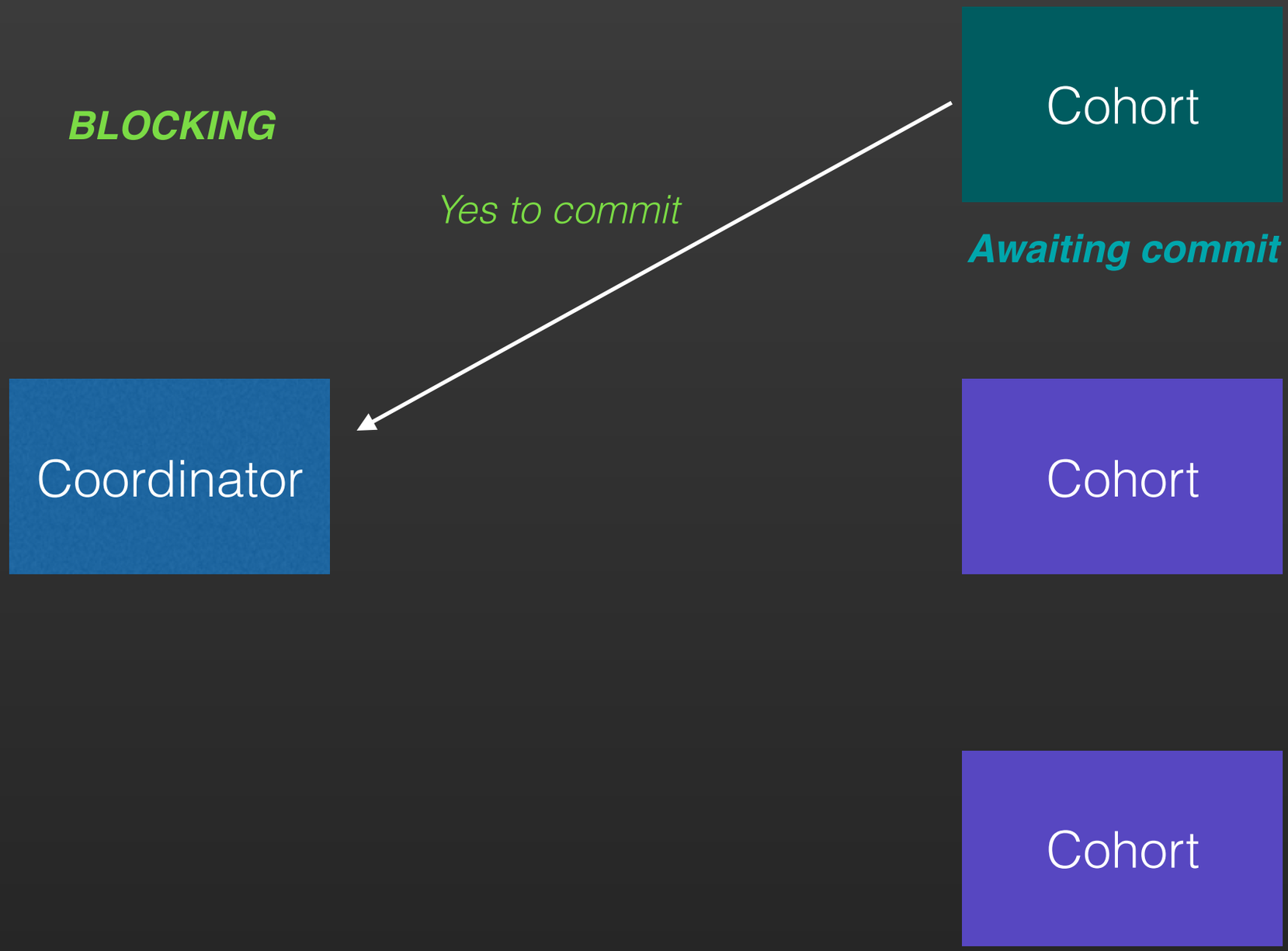
TWO PHASE COMMIT

- Two-phase commits are **simple** and **robust**
  - Major strength - **node failure can never be silent**

- Evaluation criteria:
  - **Termination** - yes, in both phases, and in protocol
  - **Agreement** - yes, either agree or abort
  - **Integrity** - yes, both agree and abort are not default values
  - **Validity** - yes, all nodes accept the same truth eventually

TWO PHASE COMMIT

BLOCKING



TWO PHASE COMMIT

BLOCKING

Coordinator

Dies

Cohort

Awaiting commit

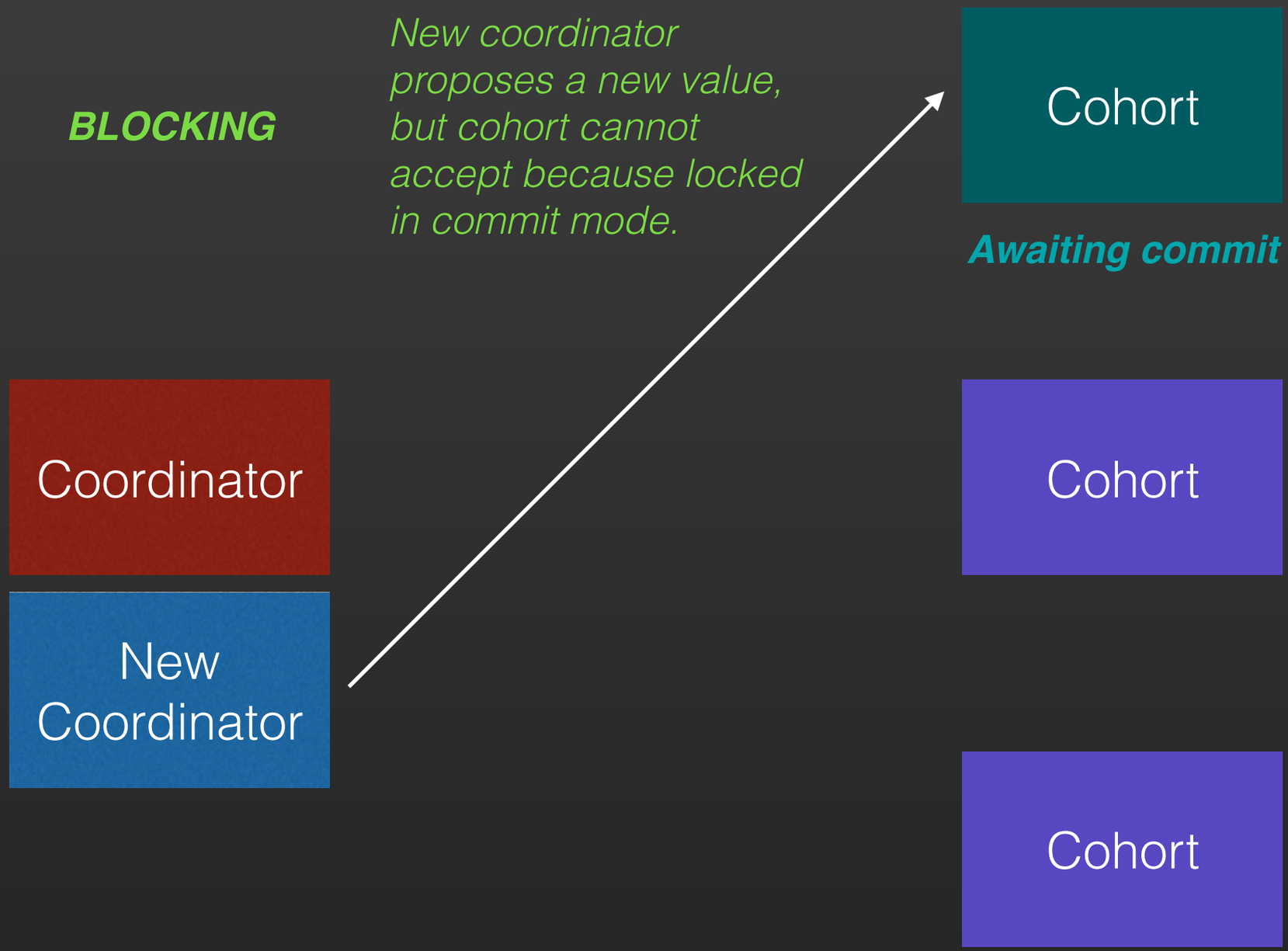
Cohort

Cohort

TWO PHASE COMMIT

BLOCKING

New coordinator proposes a new value, but cohort cannot accept because locked in commit mode.



THREE PHASE COMMIT

Solving two phase commit's locking problem

THREE PHASE COMMIT

Coordinator

Responsible for sending out commit requests

Cohort

Responsible for processing requests and replying commit/ abort

Cohort

Cohort

THREE PHASE COMMIT

*Protocol consists of 3 phases: the **voting** phase, the **pre-commit** phase, and the **commit** phase.*

Coordinator

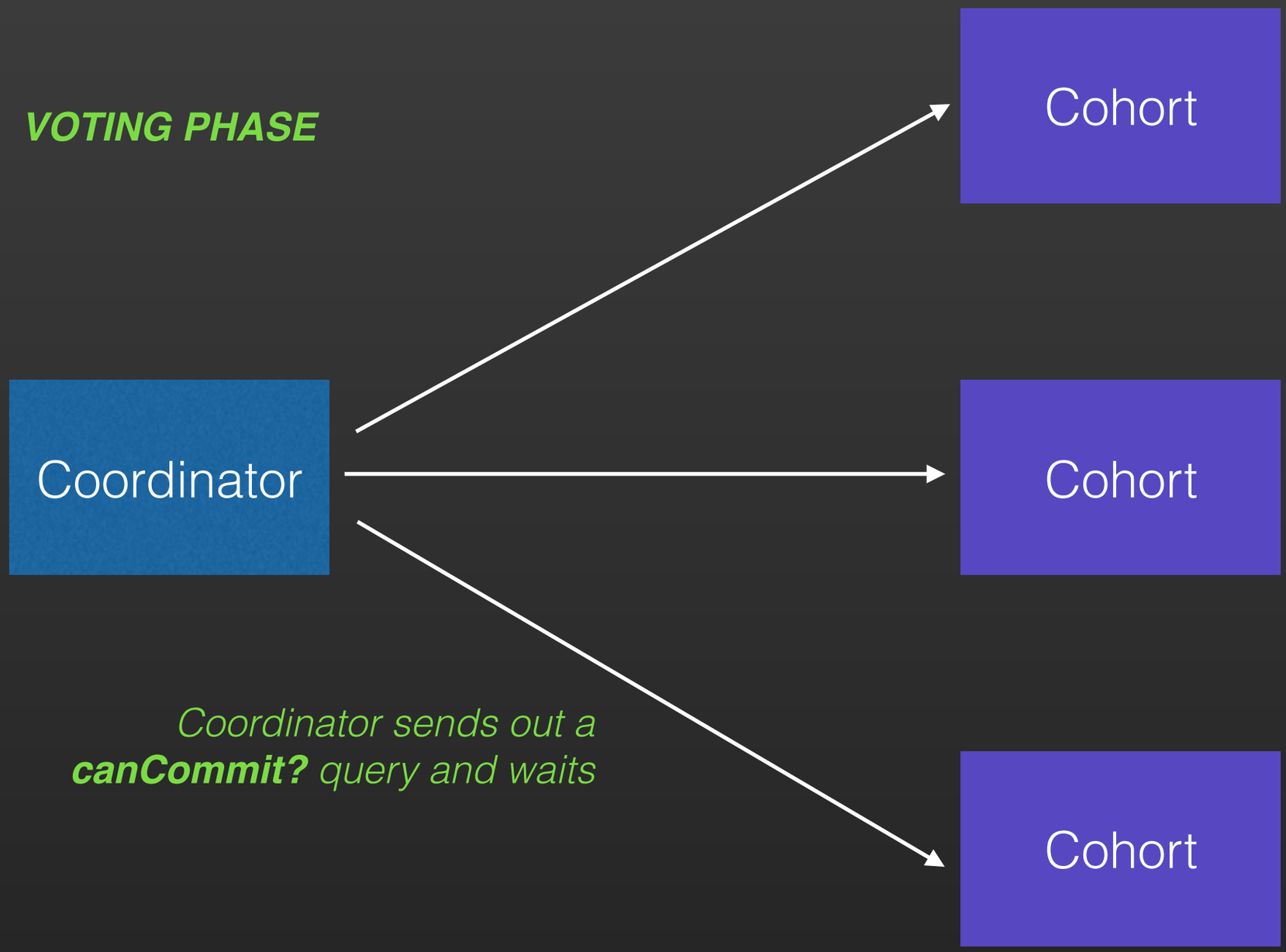
Cohort

Cohort

Cohort

THREE PHASE COMMIT

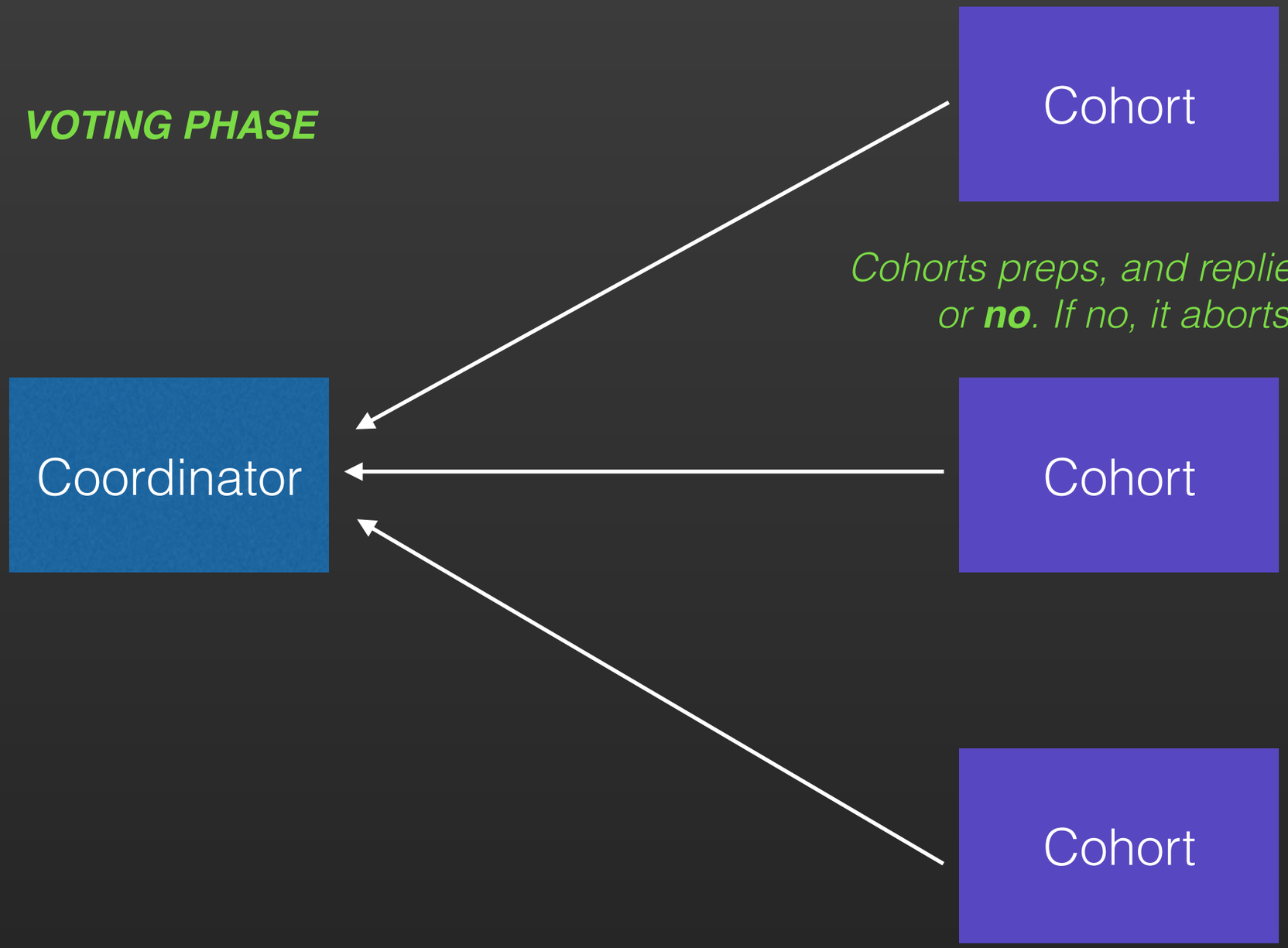
VOTING PHASE



Coordinator sends out a ***canCommit?*** *query and waits*

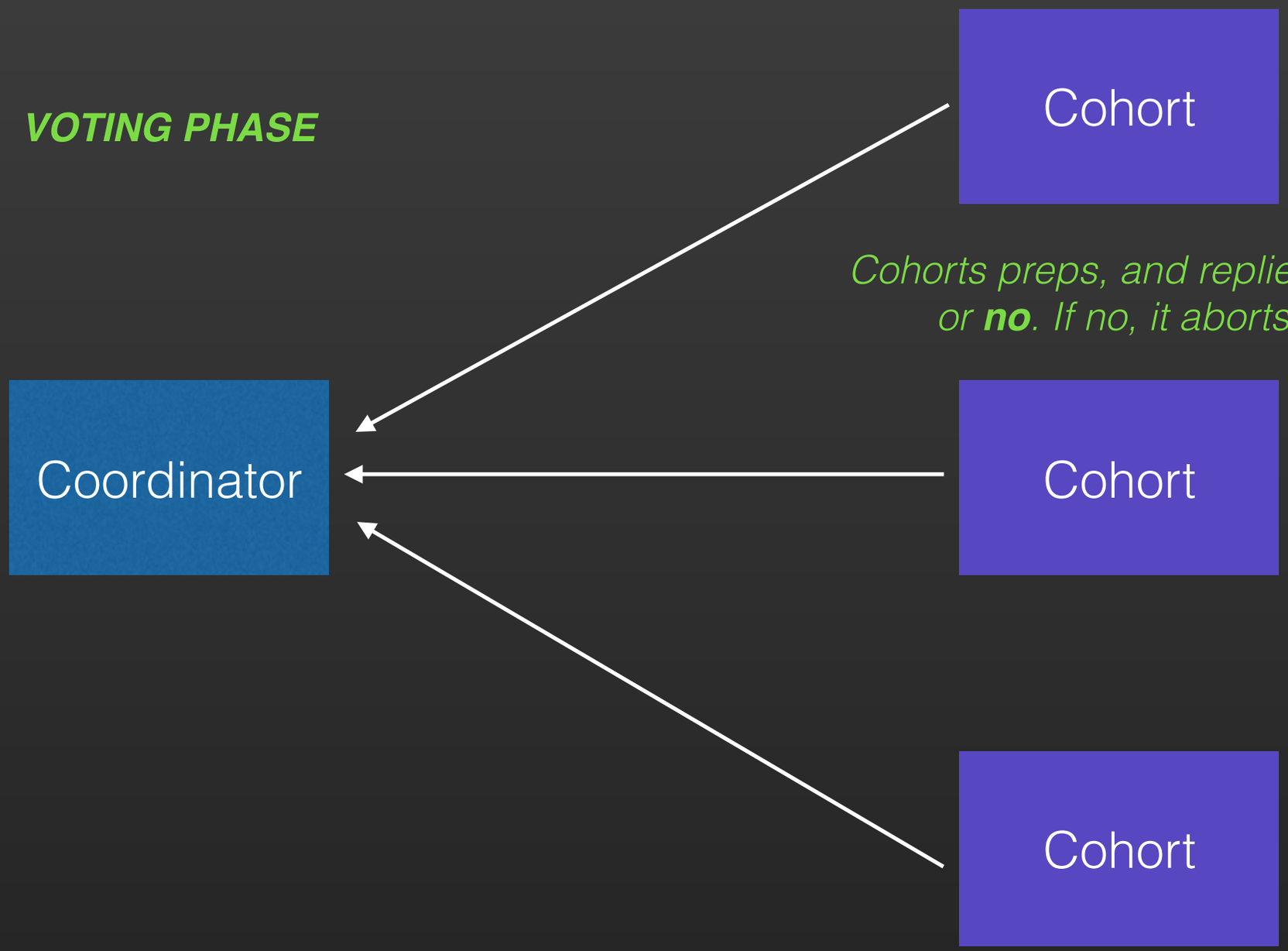
THREE PHASE COMMIT

VOTING PHASE



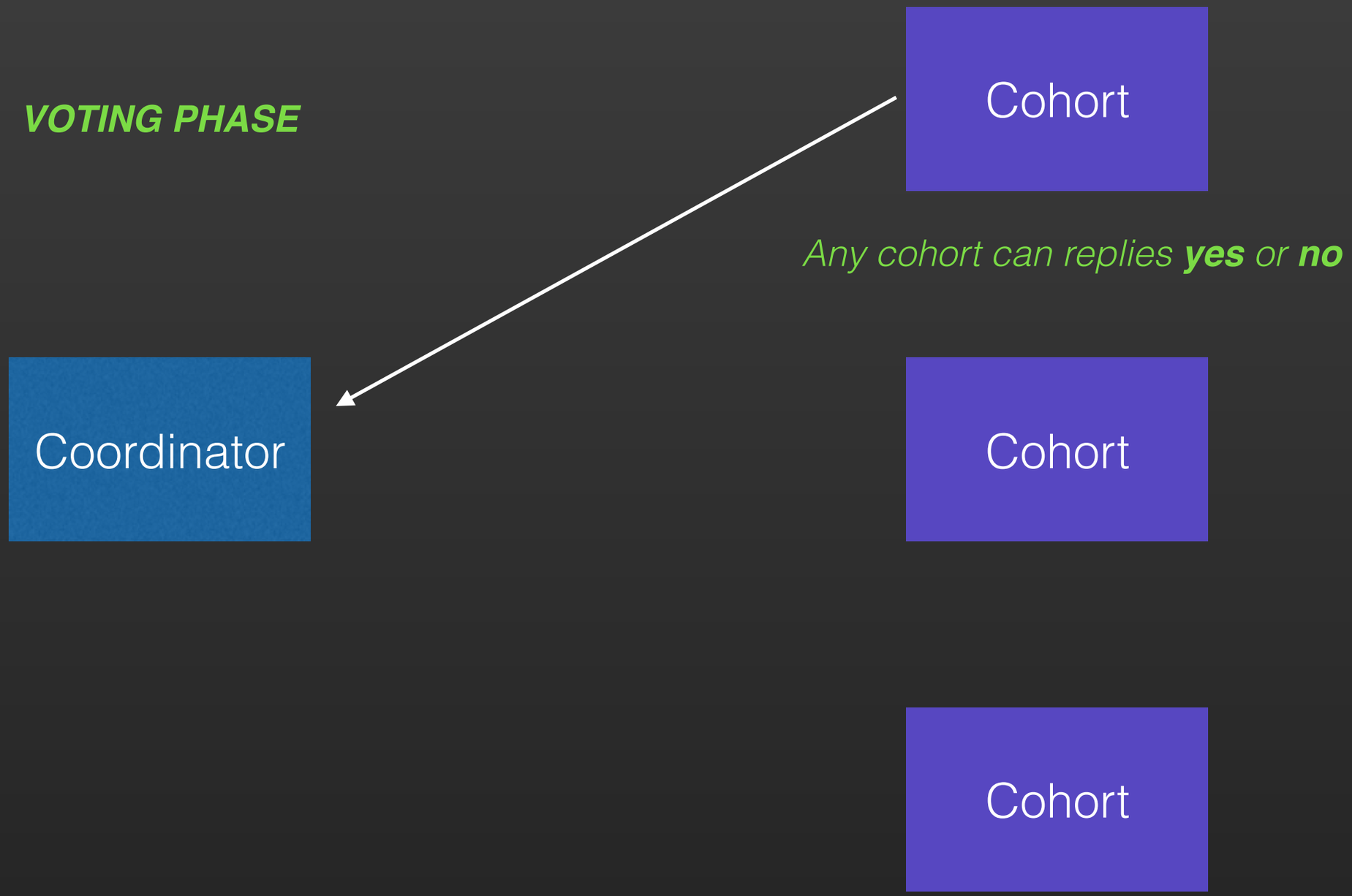
THREE PHASE COMMIT

VOTING PHASE



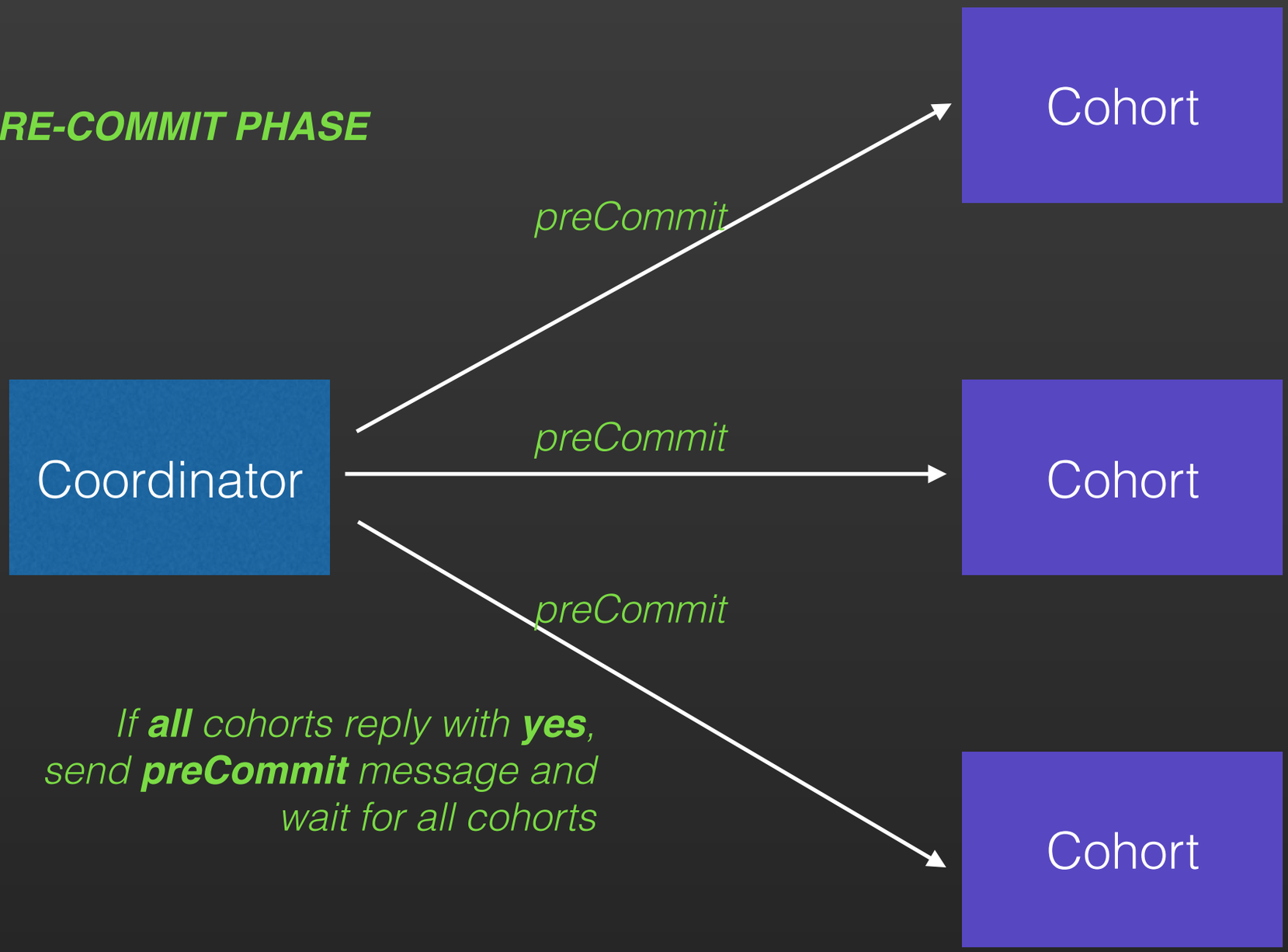
THREE PHASE COMMIT

VOTING PHASE



THREE PHASE COMMIT

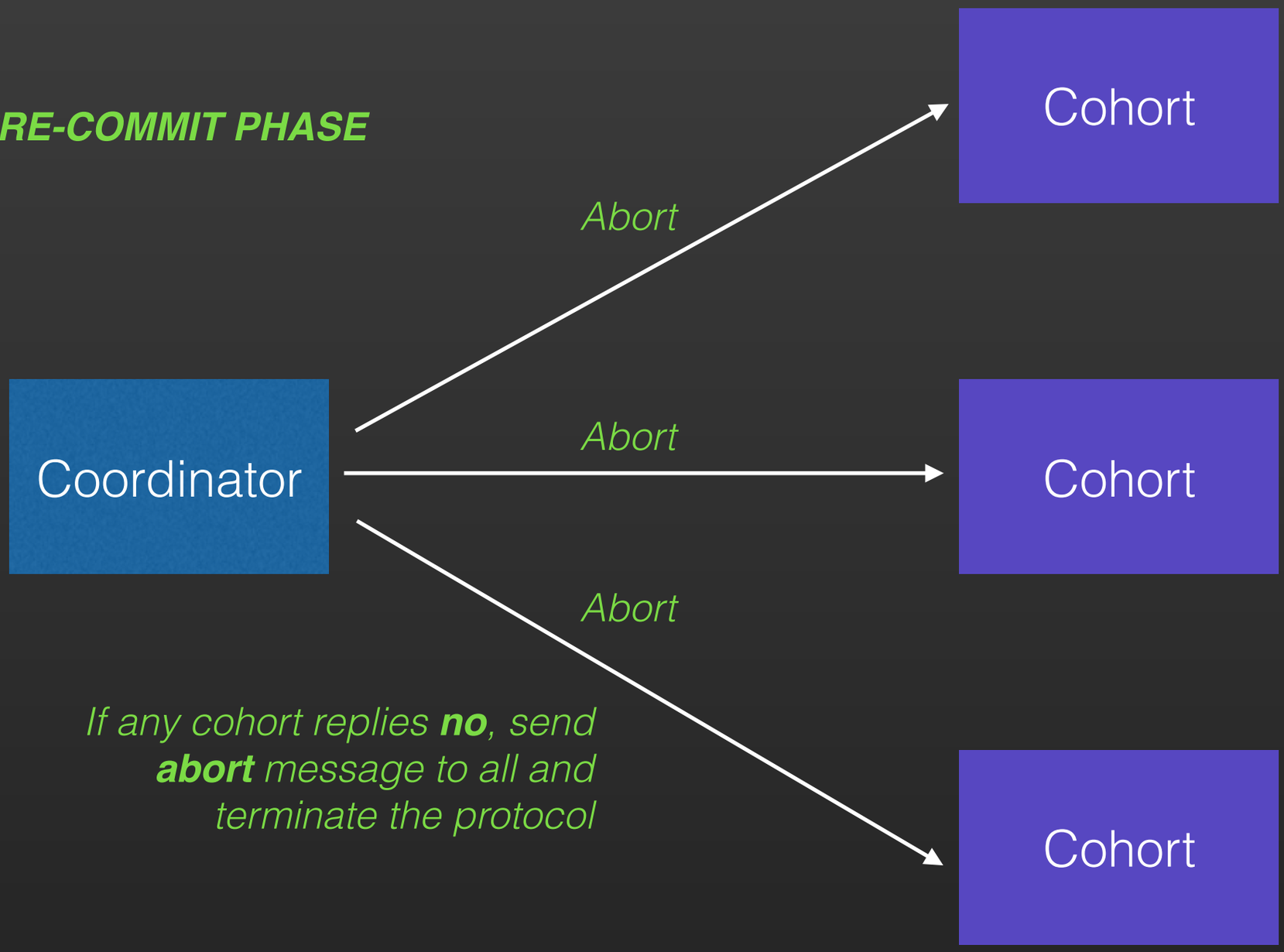
PRE-COMMIT PHASE



*If **all** cohorts reply with **yes**, send **preCommit** message and wait for all cohorts*

THREE PHASE COMMIT

PRE-COMMIT PHASE



*If any cohort replies **no**, send **abort** message to all and terminate the protocol*

THREE PHASE COMMIT

PRE-COMMIT PHASE

Coordinator

Cohort

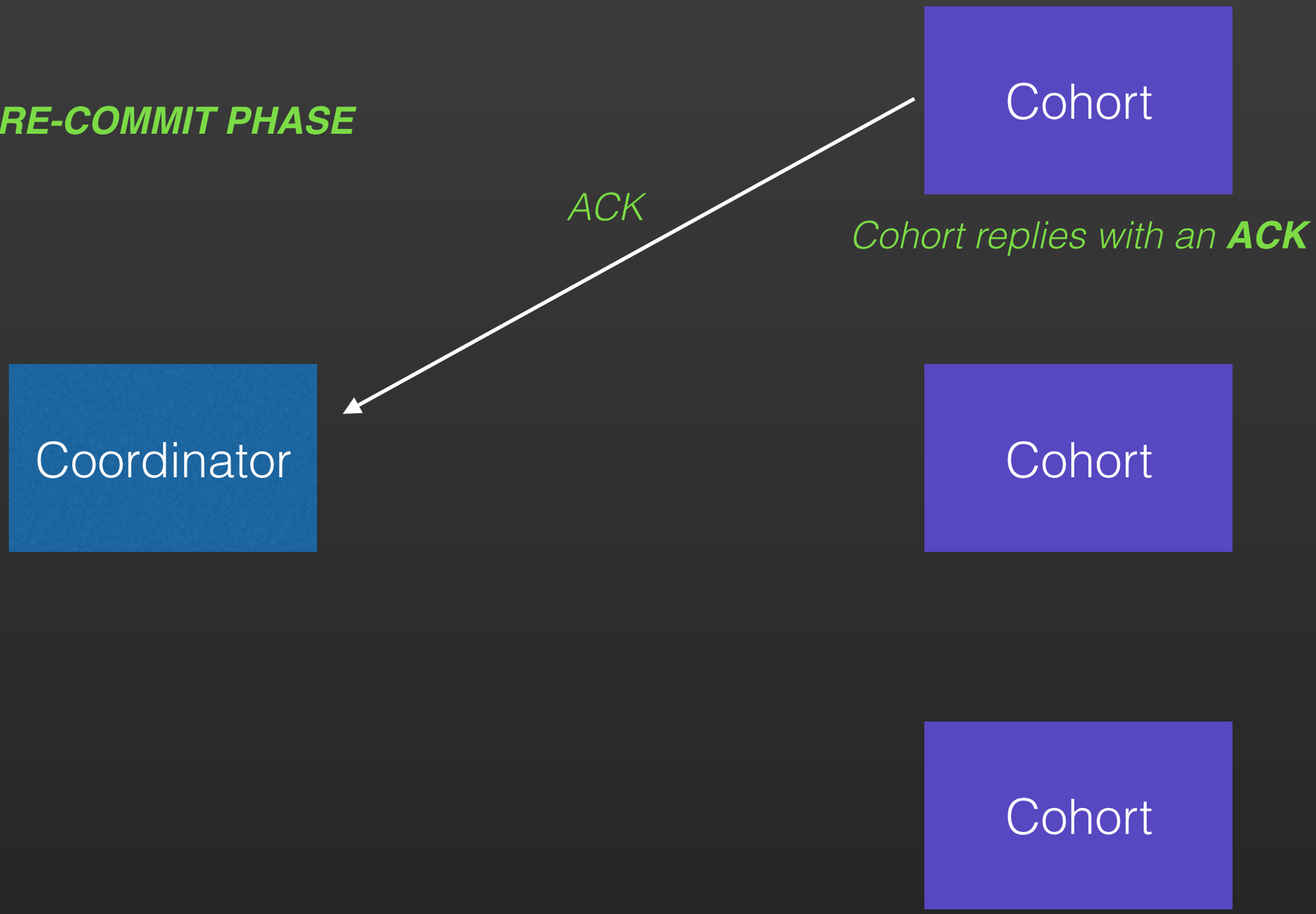
*Cohort receives **preCommit**, and prepares to commit*

Cohort

Cohort

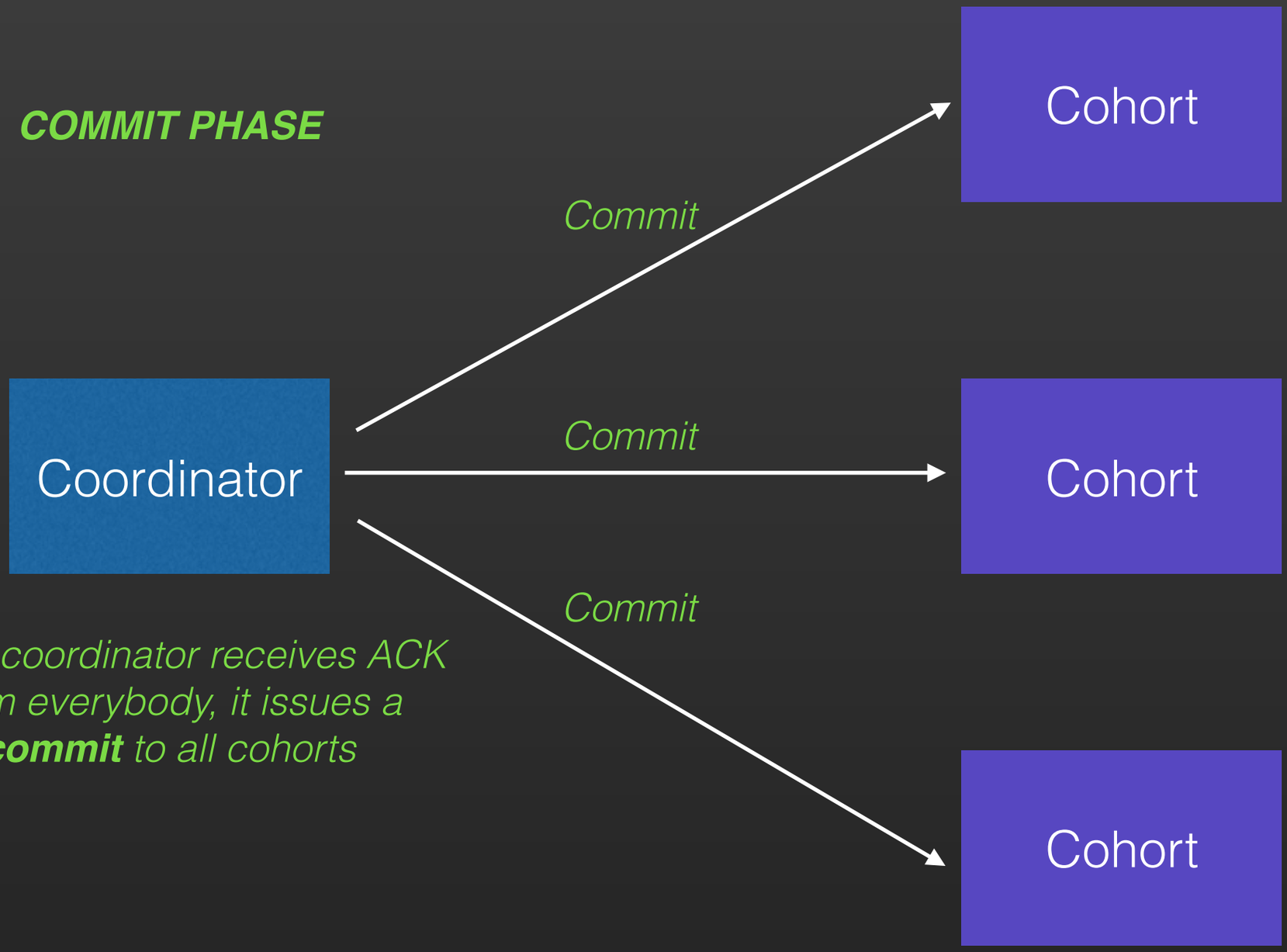
THREE PHASE COMMIT

PRE-COMMIT PHASE



THREE PHASE COMMIT

COMMIT PHASE



Once coordinator receives ACK from everybody, it issues a ***commit*** *to all cohorts*

THREE PHASE COMMIT

FAILURE IN VOTING PHASE

Cohort

If cohort is uncertain, timeout will cause coordinator to abort

Coordinator

Cohort

Cohort

THREE PHASE COMMIT

IMPROVING 2PC

Coordinator

Dies after receiving first cohort's ACK.

Cohort

Prepared to commit

Cohort

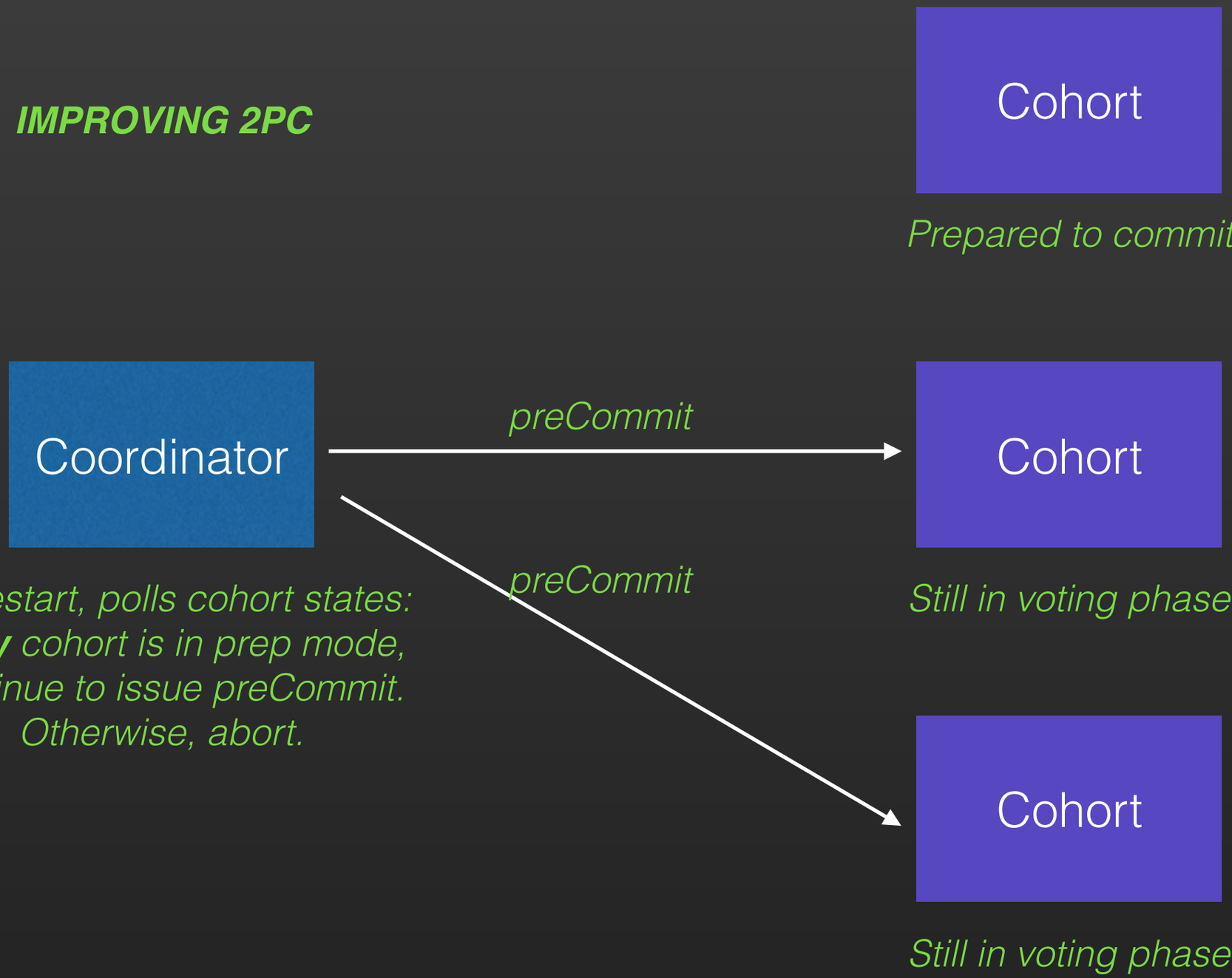
Still in voting phase

Cohort

Still in voting phase

THREE PHASE COMMIT

IMPROVING 2PC



*On restart, polls cohort states: if **any** cohort is in prep mode, continue to issue preCommit. Otherwise, abort.*

THREE PHASE COMMIT

IMPROVING 2PC

Coordinator

Successfully resumed

Cohort

Prepared to commit

Cohort

Prepared to commit

Cohort

Prepared to commit

THREE PHASE COMMIT

- Three-phase commits are more resilient to single-point failures
  - Major strength - **fixes 2PC's blocking issue**

- Important observations:
  - Protocol complexity increased
  - More roundtrips to ensure consistency in the system
  - Introduces new avenues of failure due to increased complexity
  - Is not byzantine fault tolerant

PAXOS (BASIC)

Looking at a functional real world solution

PAXOS

- Presented by **Leslie Lamport** in **The Part-time Parliament (1988)** whitepaper
- Named after Paxos civilisation's legislation
- Important observations
  - The hardest to understand in theory
  - The hardest to implement
  - The closest we can get to **ideal consensus**
- Used in
  - Apache Zookeeper
  - Google Bigtable
  - Google Spannar
  - Apache Mesos
  - etc

PAXOS

Proposer

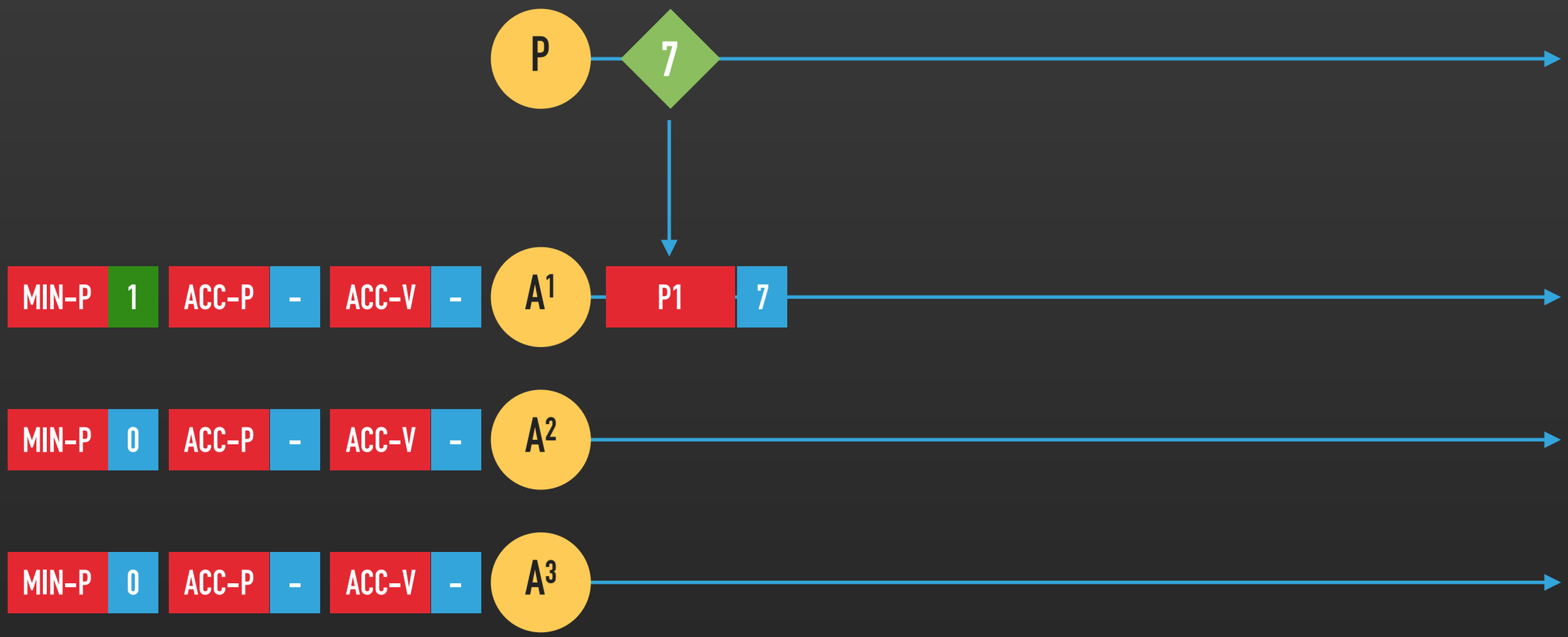
Proposes a value to acceptors

Acceptor

Responds with votes

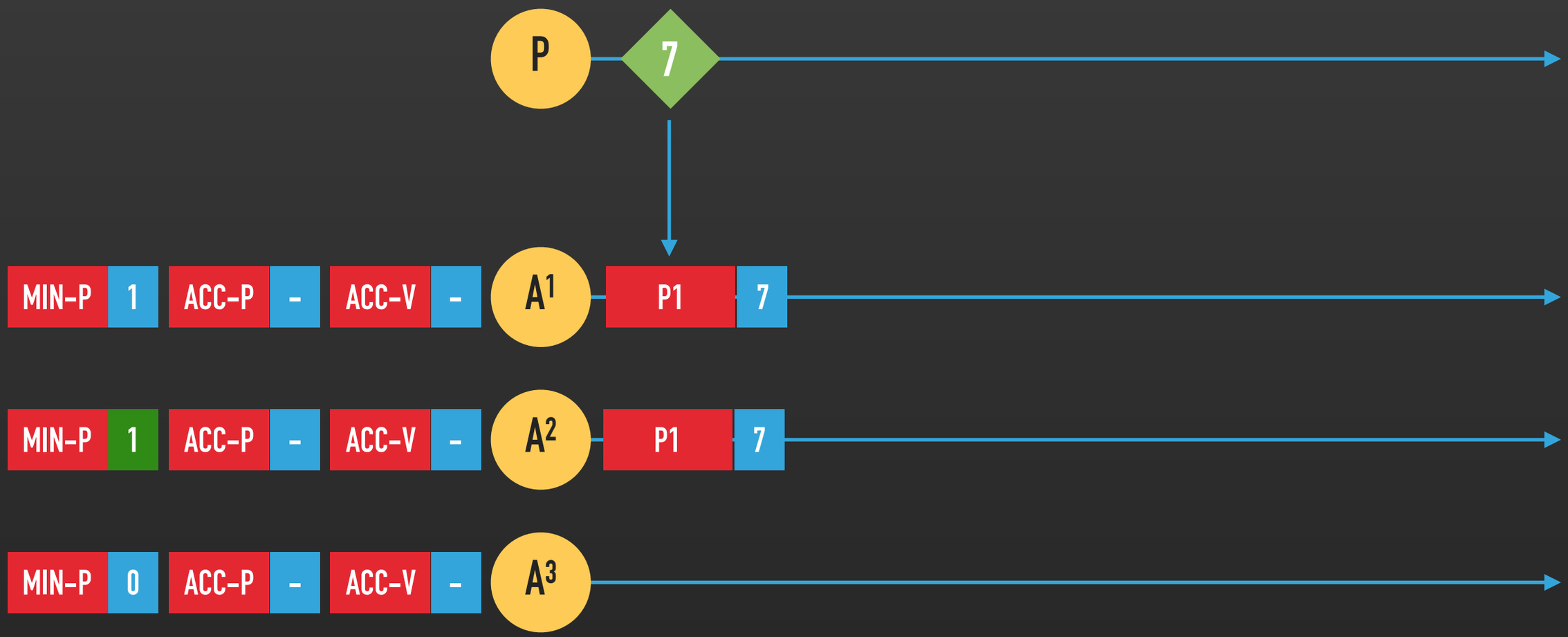
PAXOS

v7 is proposed with p1



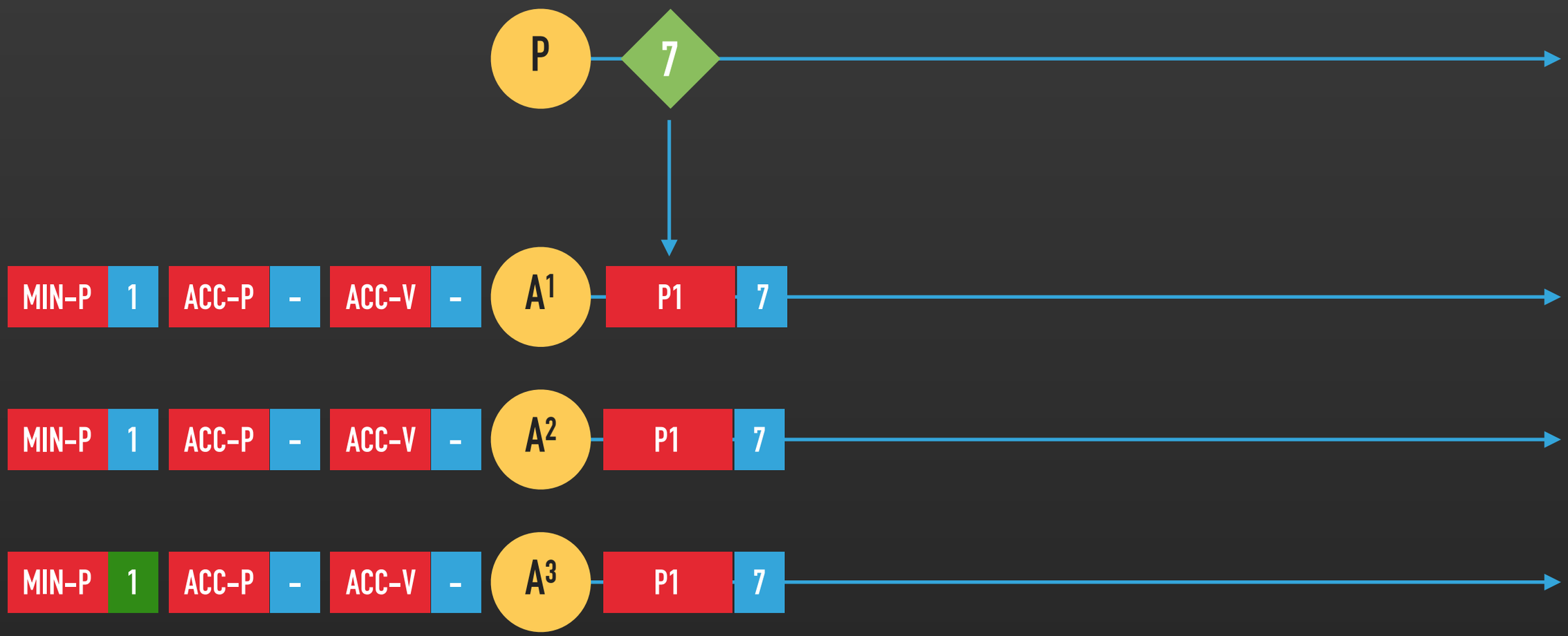
PAXOS

v7 is proposed with p1



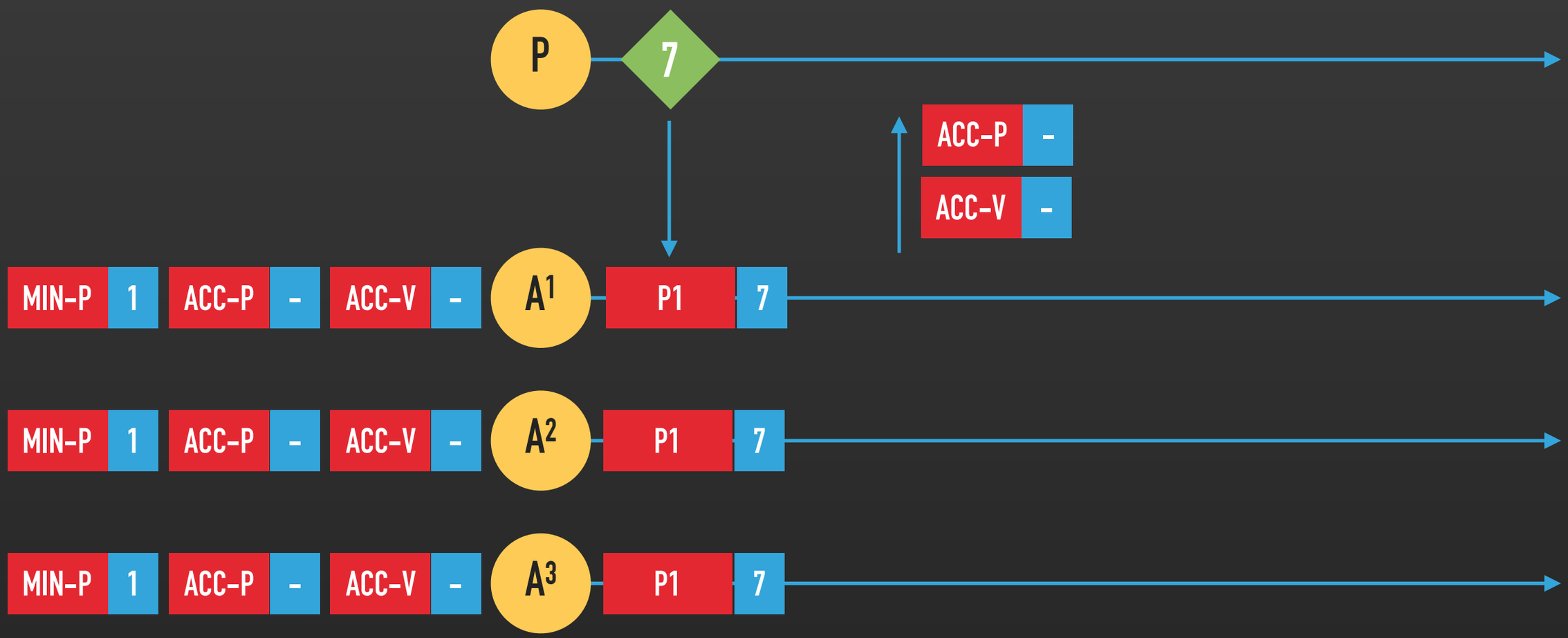
PAXOS

v7 is proposed with p1



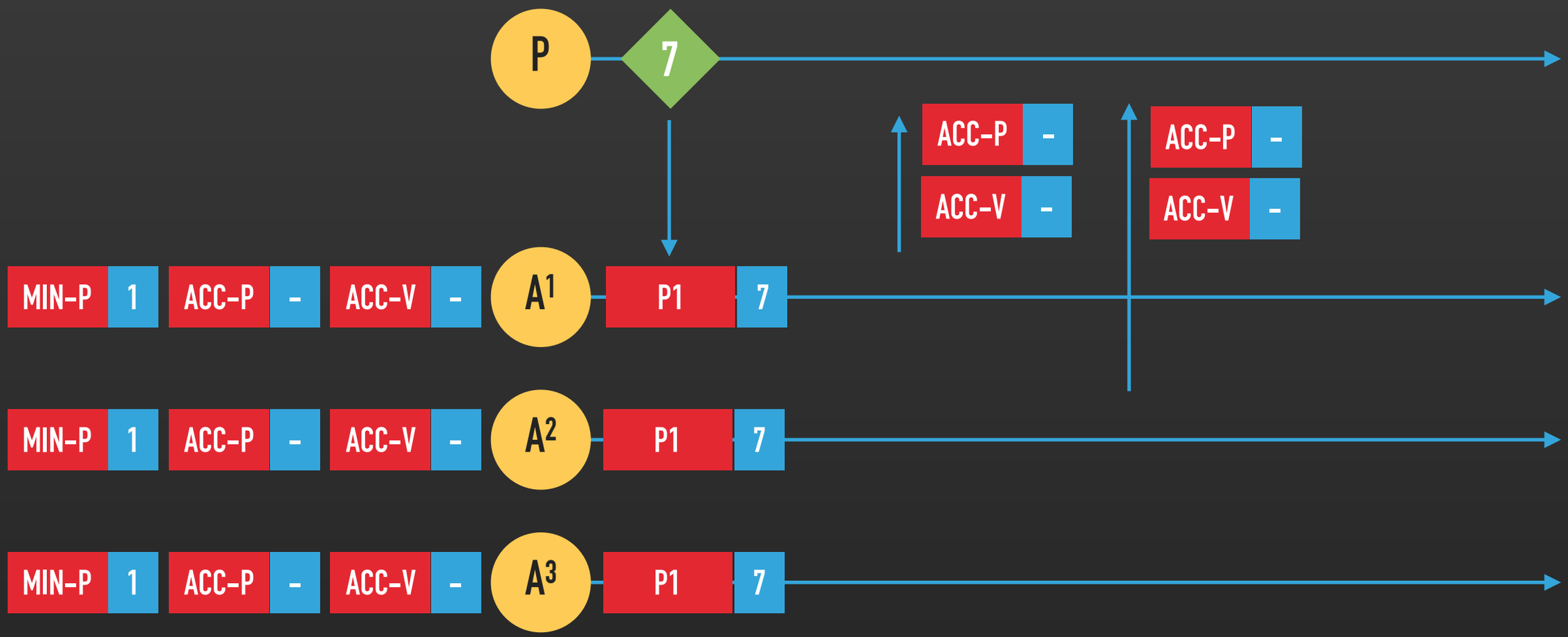
PAXOS

v7 is proposed with p1



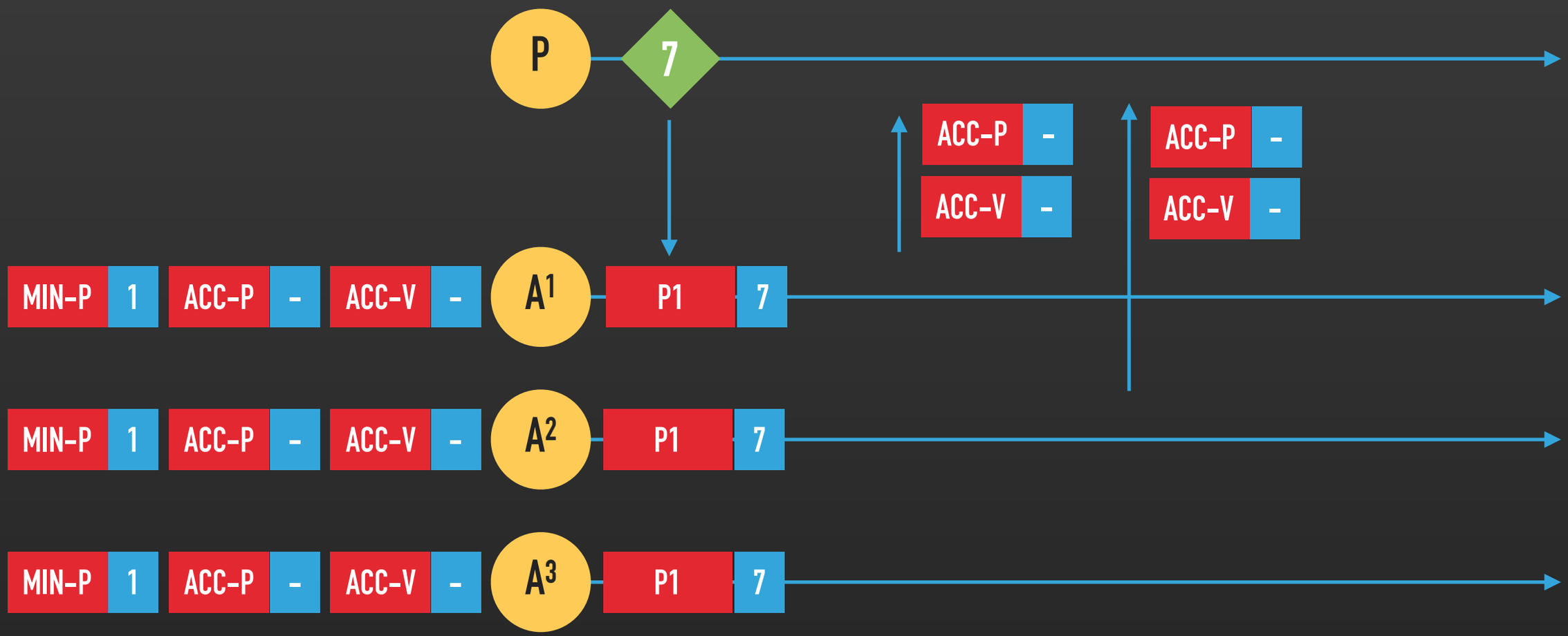
PAXOS

v7 is proposed with p1



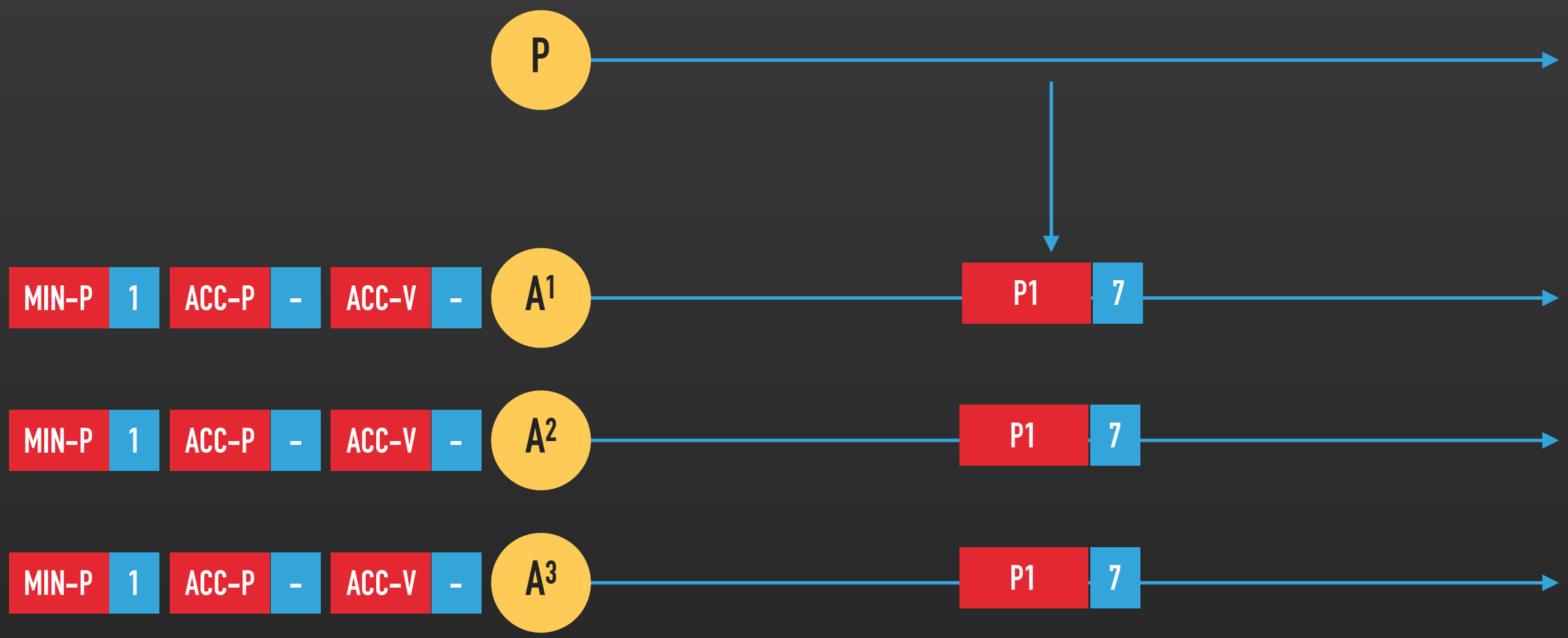
PAXOS

Has majority! Since acc-p and acc-v are both null, we know that we are the only proposers in the network so far

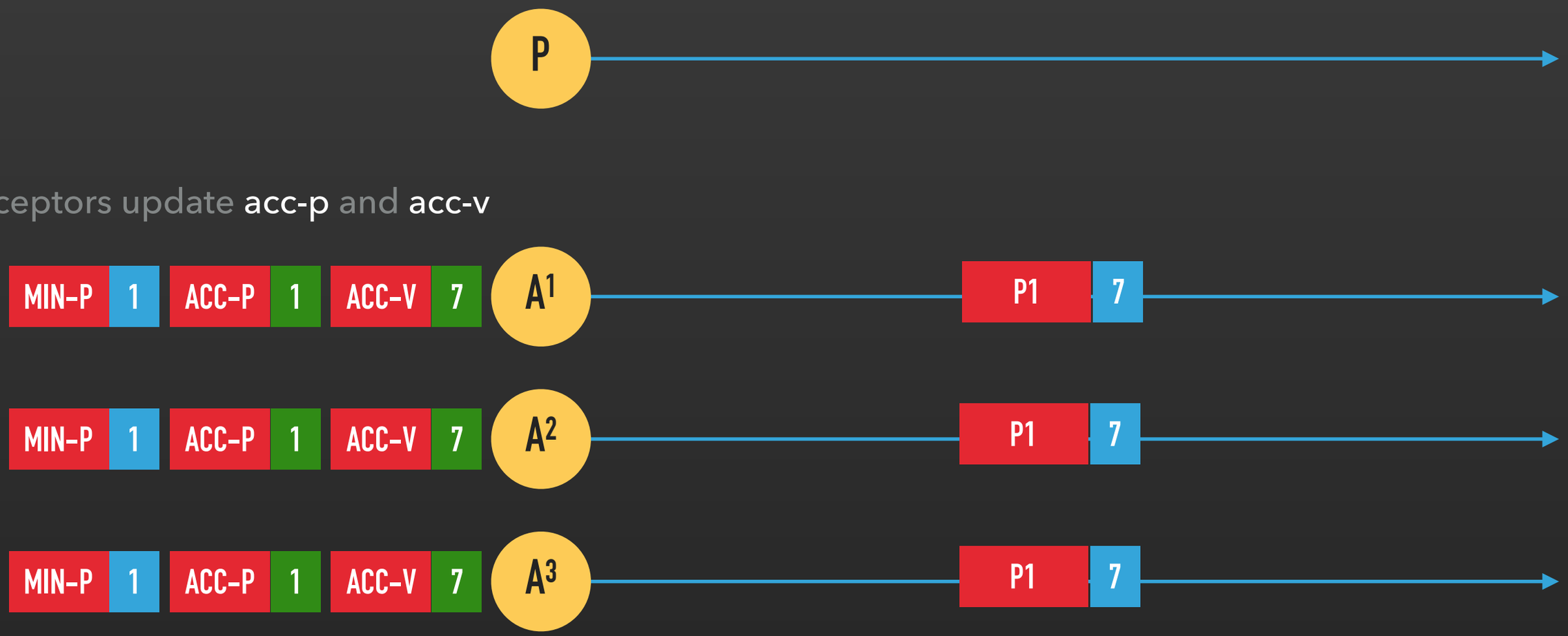


PAXOS

Now, we send out p and v in the accept phase

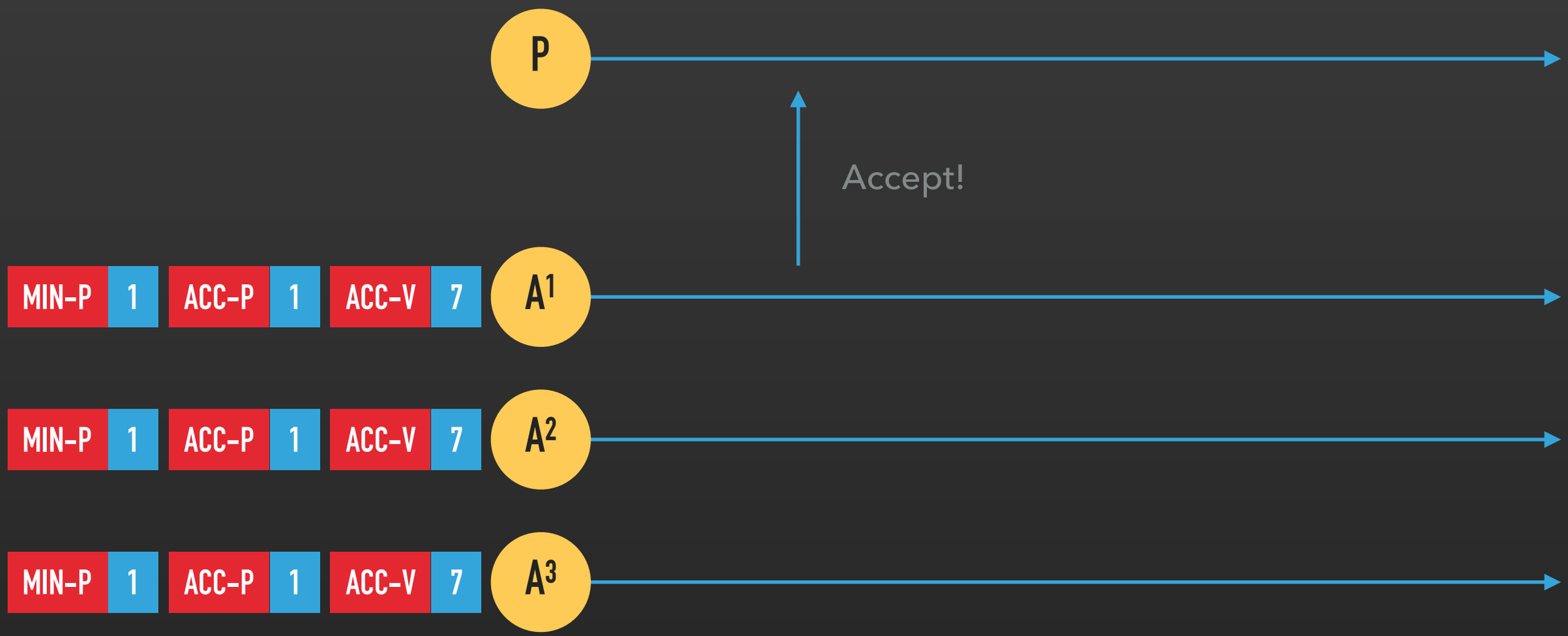


PAXOS

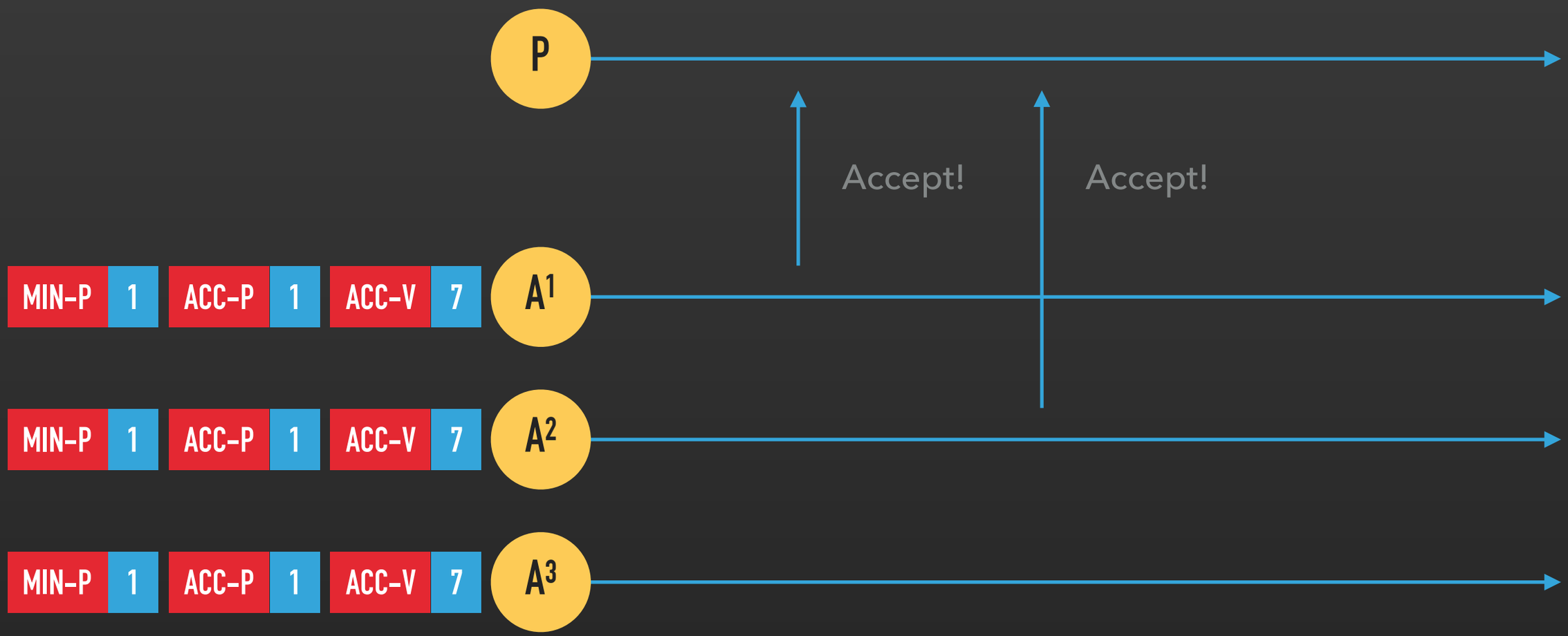


Acceptors update acc-p and acc-v

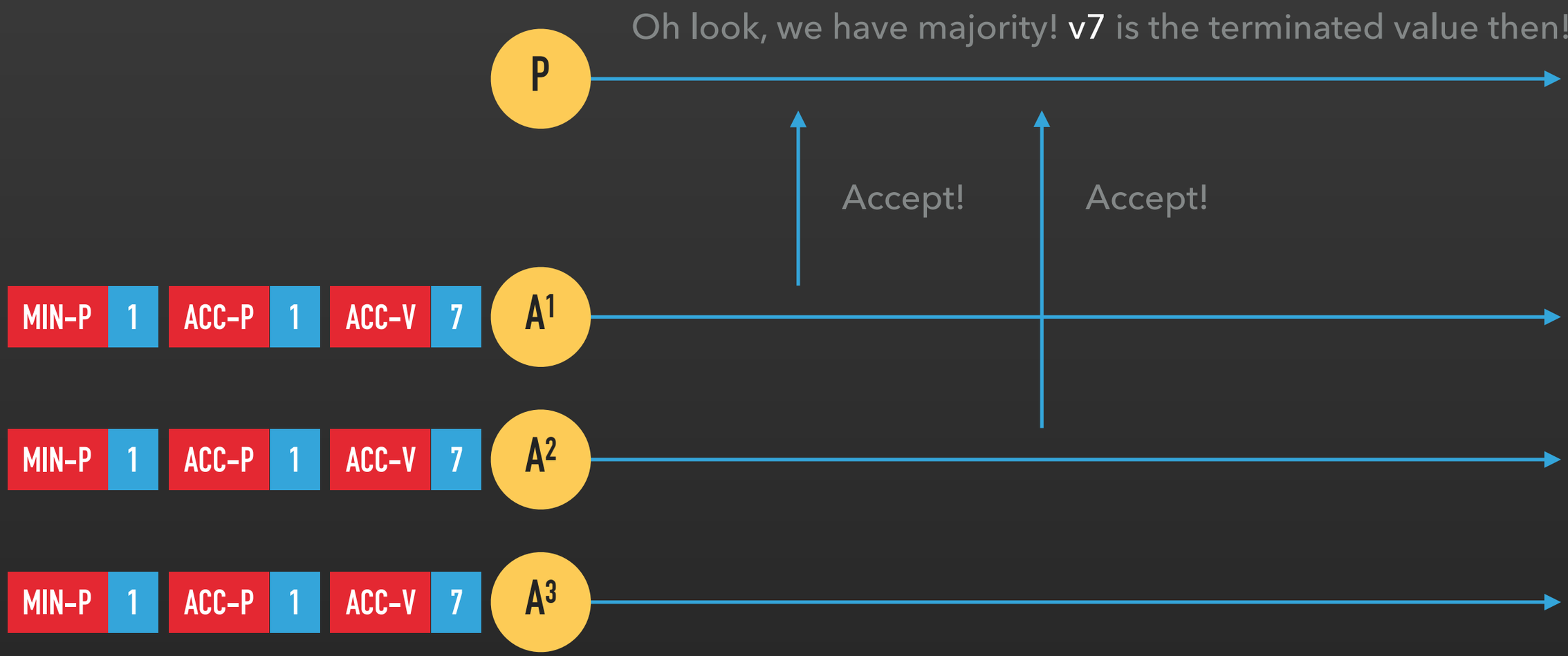
PAXOS


P
Accept!
MIN-P 1 ACC-P 1 ACC-V 7
A1
MIN-P 1 ACC-P 1 ACC-V 7
A2
MIN-P 1 ACC-P 1 ACC-V 7
A3

PAXOS


P
Accept!
Accept!
MIN-P 1 ACC-P 1 ACC-V 7
A1
MIN-P 1 ACC-P 1 ACC-V 7
A2
MIN-P 1 ACC-P 1 ACC-V 7
A3

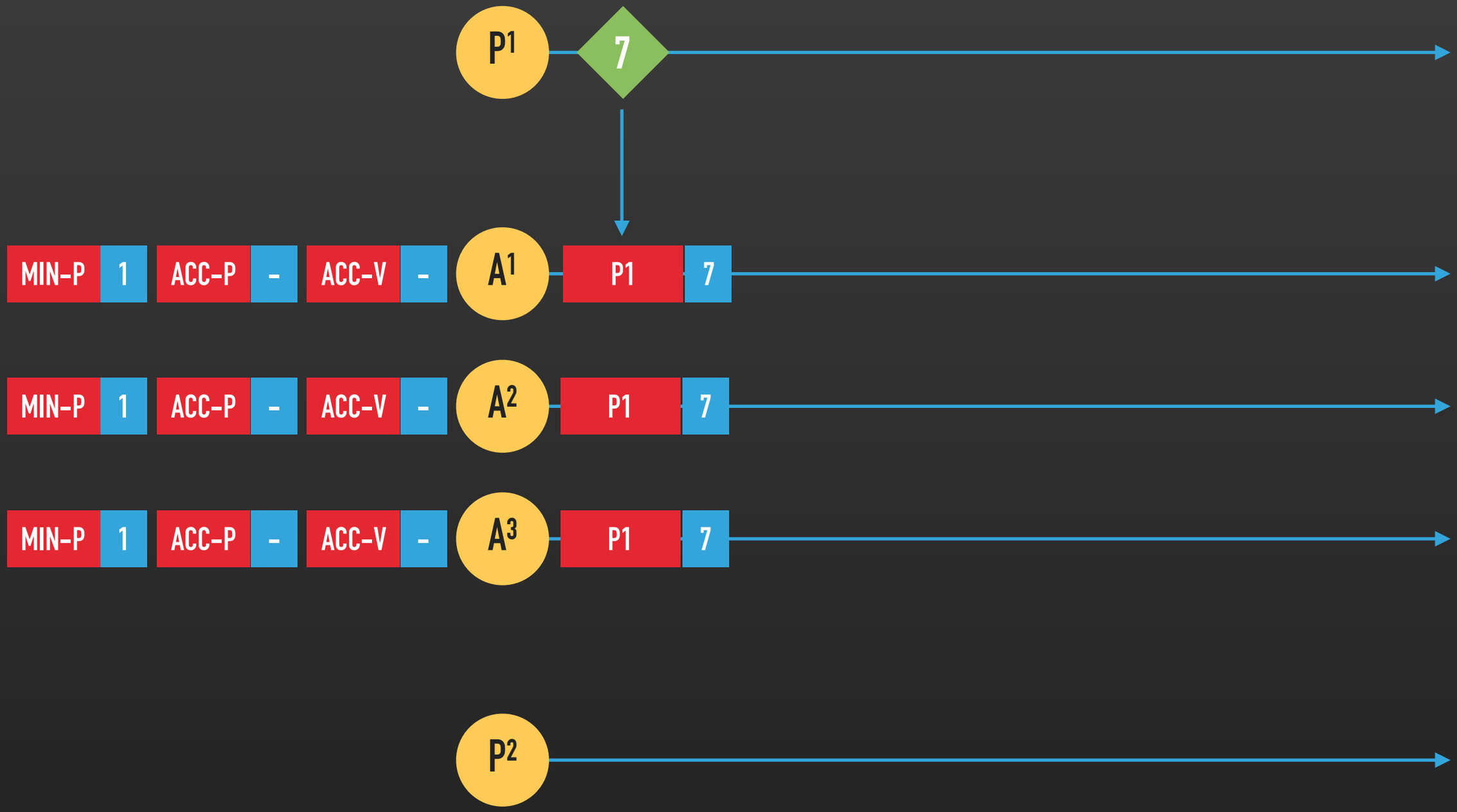
PAXOS


Oh look, we have majority! v7 is the terminated value then!
P
Accept!
Accept!
MIN-P 1
ACC-P 1
ACC-V 7
A1
MIN-P 1
ACC-P 1
ACC-V 7
A2
MIN-P 1
ACC-P 1
ACC-V 7
A3

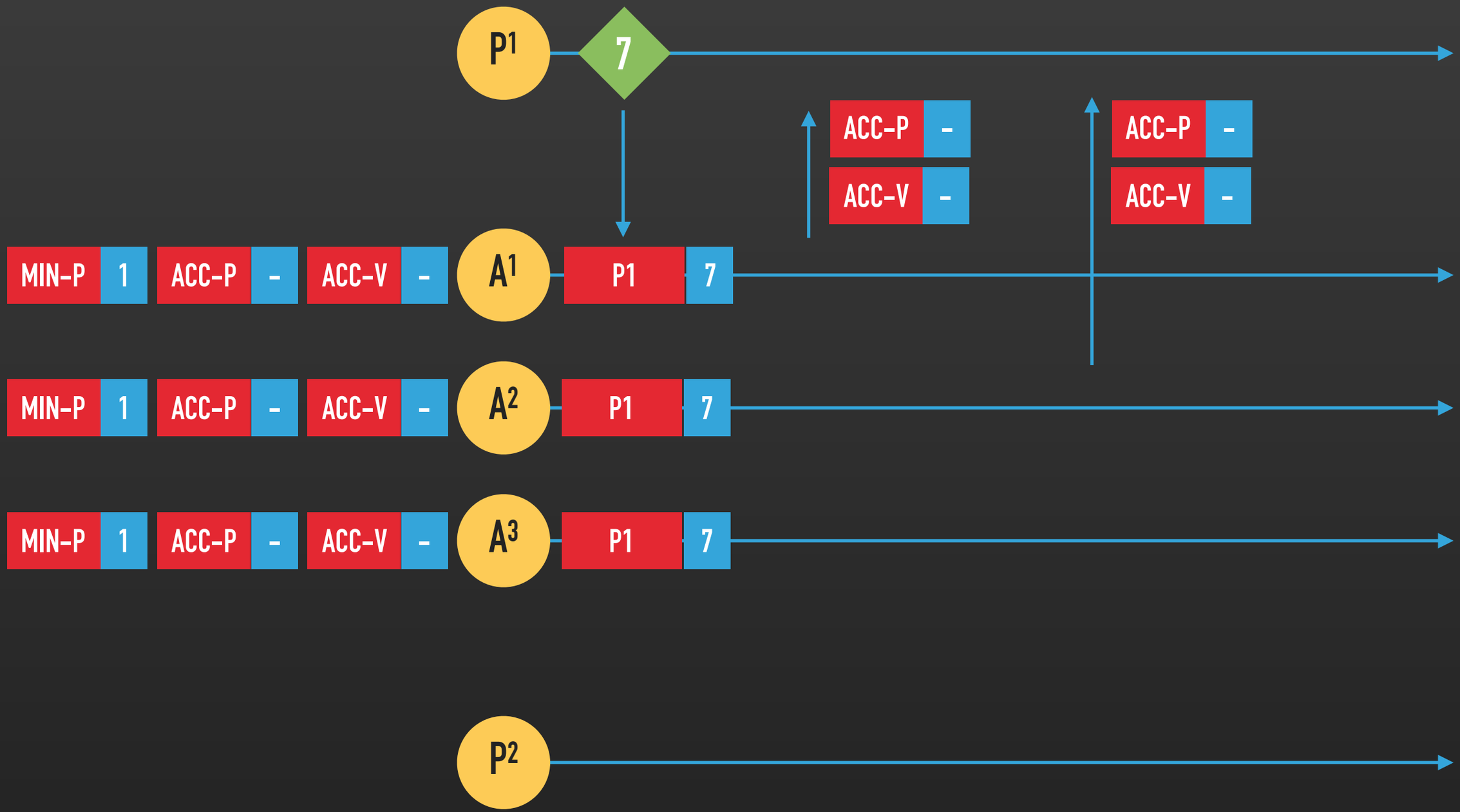
PAXOS

What if we had multiple proposers?

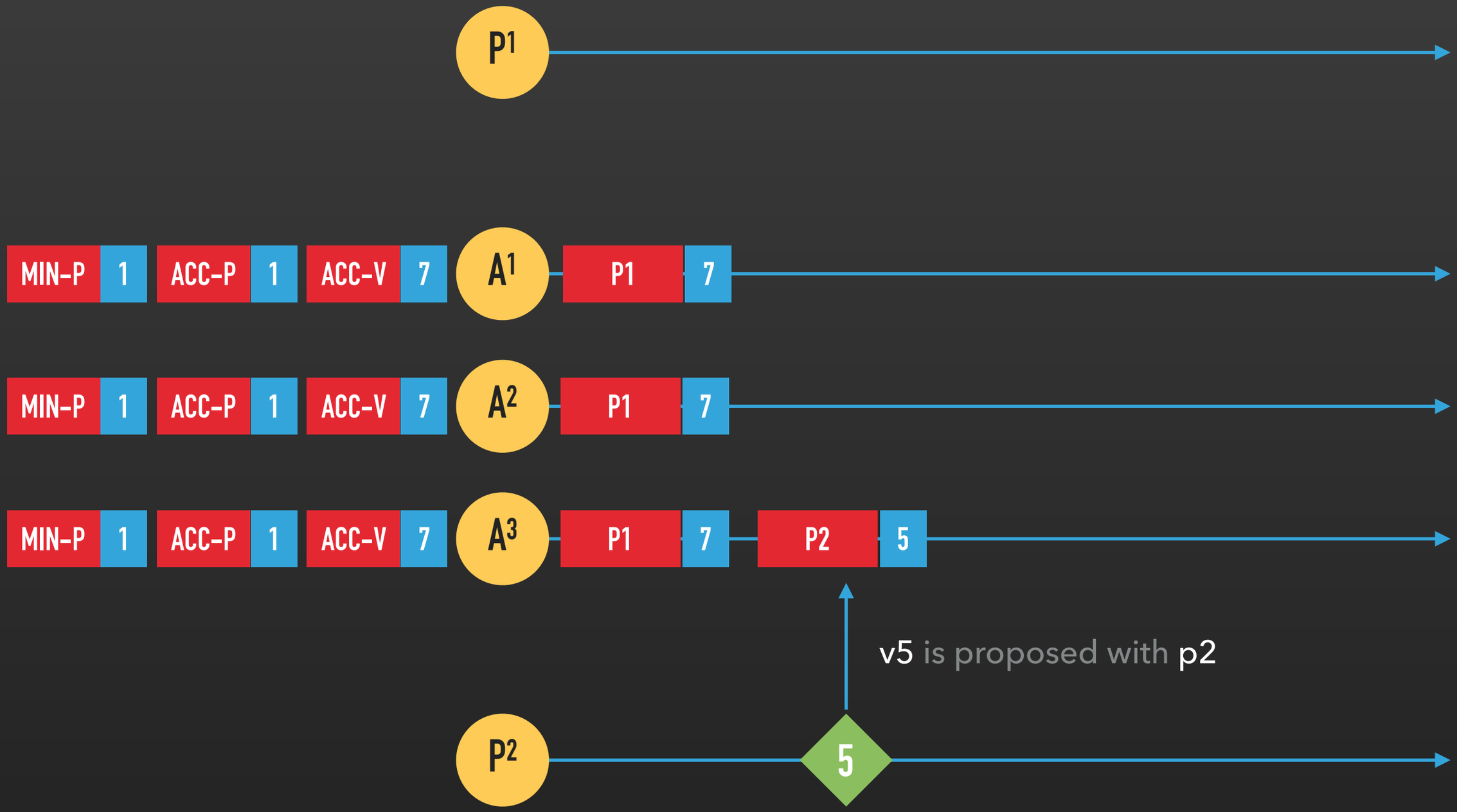
PAXOS


P1
7
MIN-P
1
ACC-P
-
ACC-V
-
A1
P1
7
MIN-P
1
ACC-P
-
ACC-V
-
A2
P1
7
MIN-P
1
ACC-P
-
ACC-V
-
A3
P1
7
P2

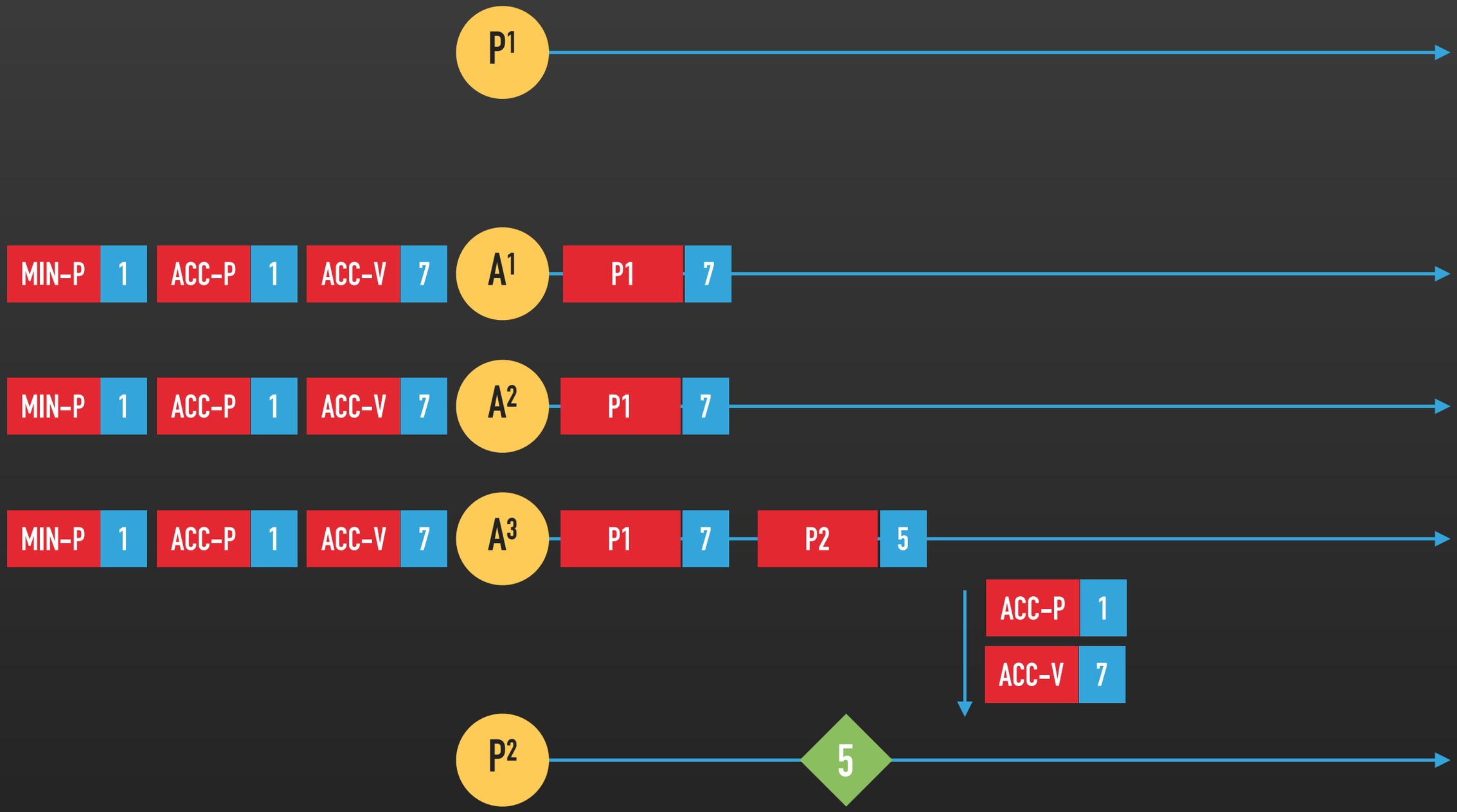
PAXOS


P1
7
ACC-P -
ACC-V -
ACC-P -
ACC-V -
MIN-P 1
ACC-P -
ACC-V -
A1
P1
7
MIN-P 1
ACC-P -
ACC-V -
A2
P1
7
MIN-P 1
ACC-P -
ACC-V -
A3
P1
7
P2

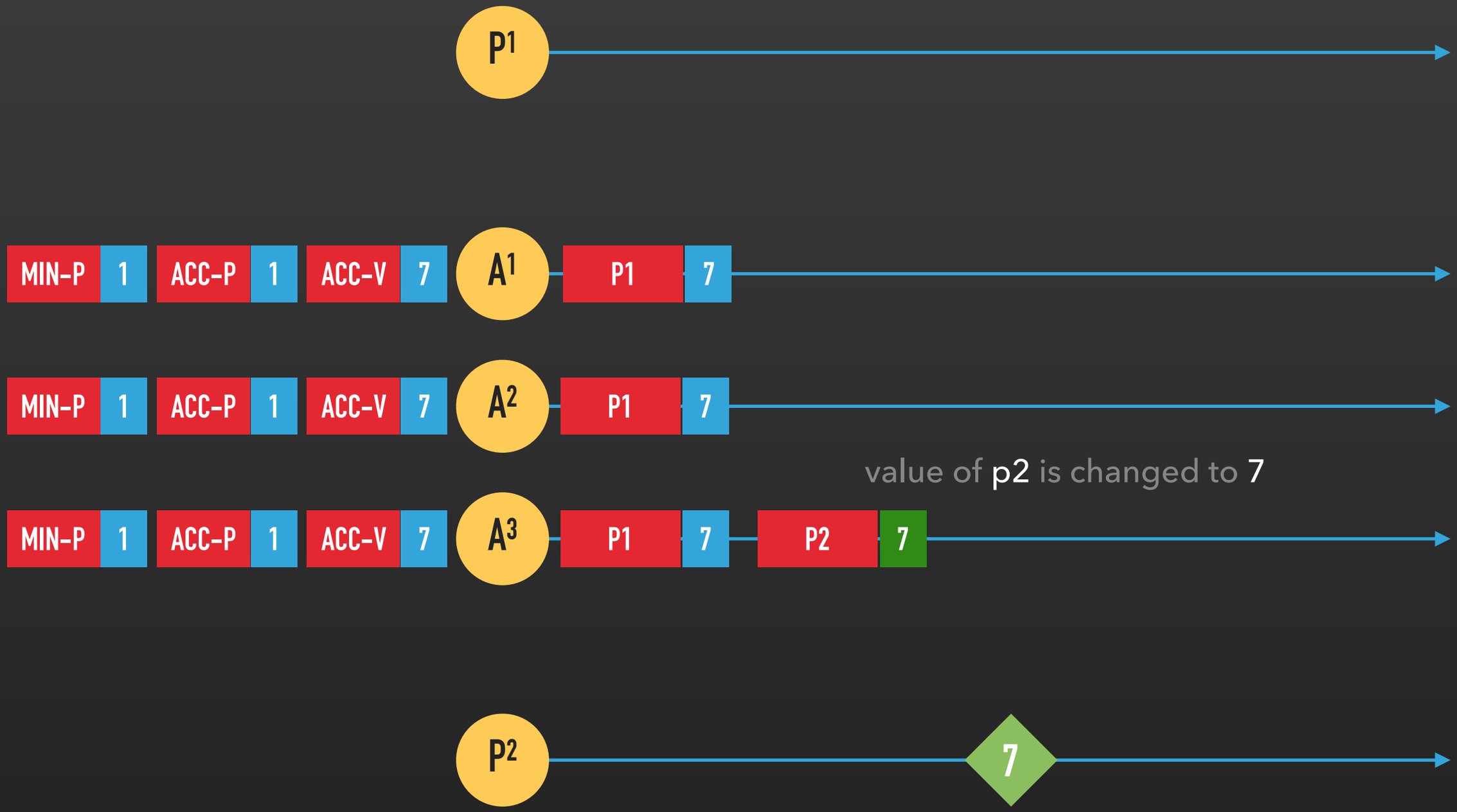
PAXOS


P1
MIN-P 1
ACC-P 1
ACC-V 7
A1
P1 7
MIN-P 1
ACC-P 1
ACC-V 7
A2
P1 7
MIN-P 1
ACC-P 1
ACC-V 7
A3
P1 7
P2 5
v5 is proposed with p2
P2
5

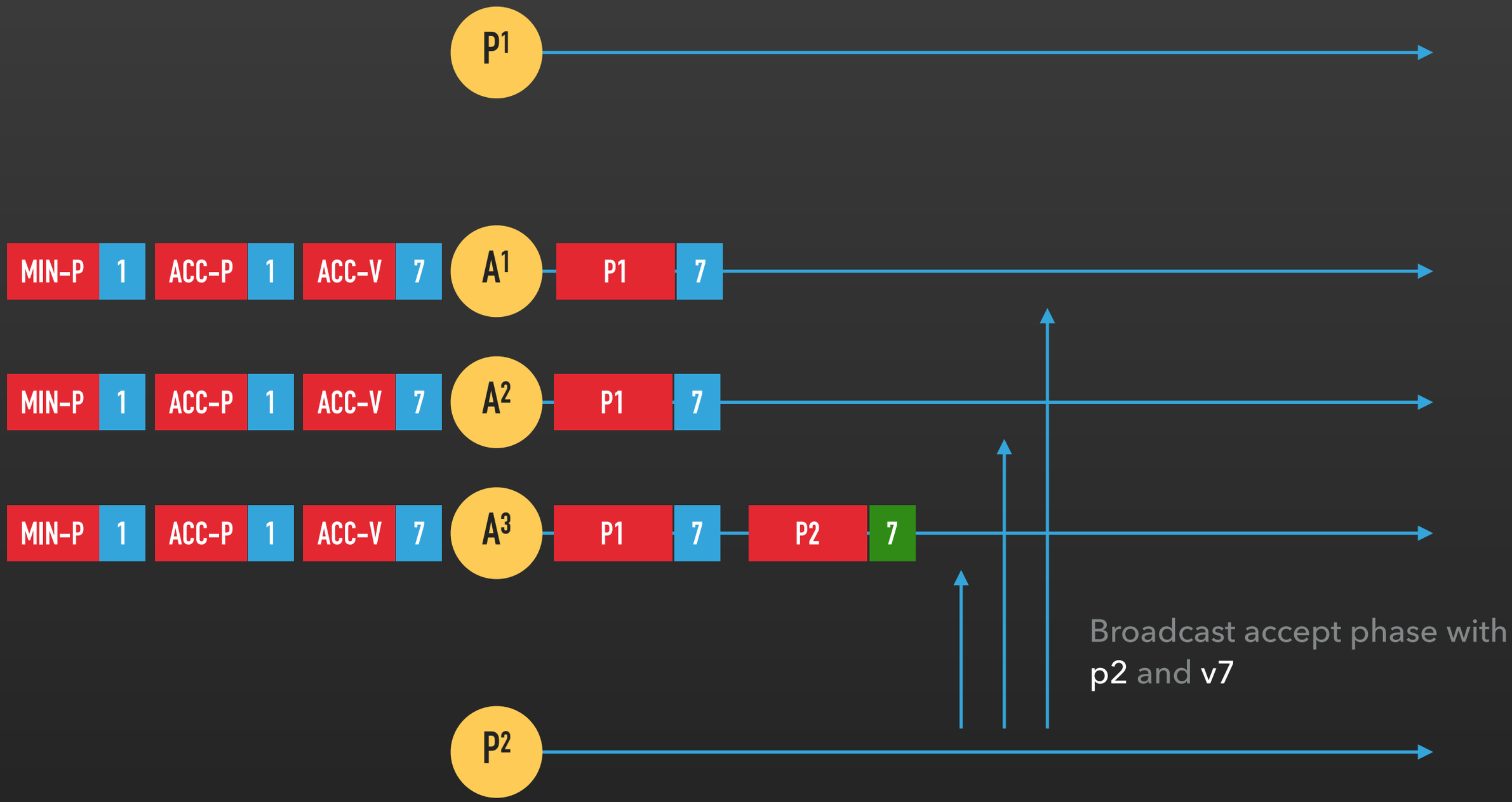
PAXOS


P1
MIN-P 1
ACC-P 1
ACC-V 7
A1
P1 7
MIN-P 1
ACC-P 1
ACC-V 7
A2
P1 7
MIN-P 1
ACC-P 1
ACC-V 7
A3
P1 7
P2 5
ACC-P 1
ACC-V 7
P2
5

PAXOS

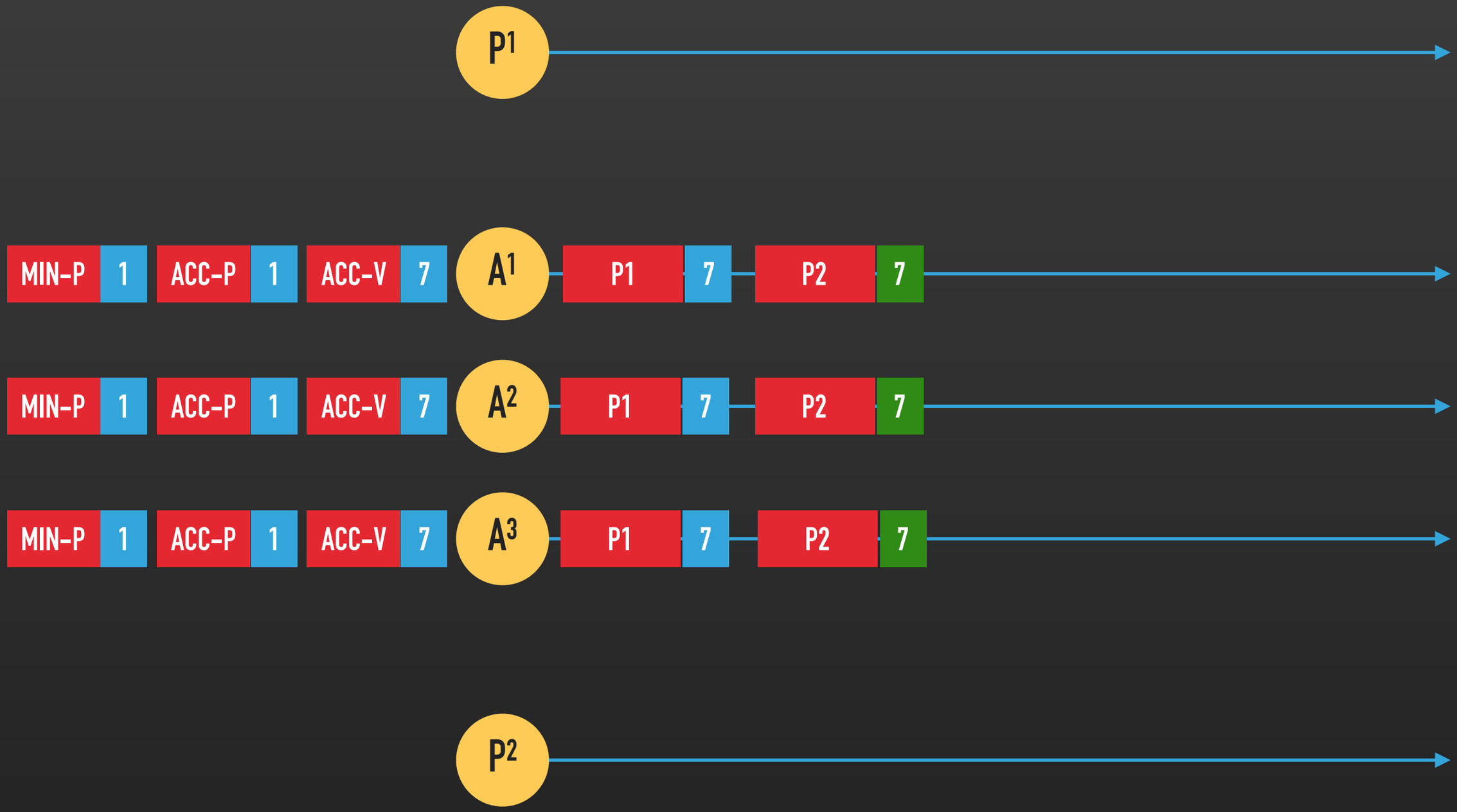

P1
MIN-P 1
ACC-P 1
ACC-V 7
A1
P1 7
MIN-P 1
ACC-P 1
ACC-V 7
A2
P1 7
value of p2 is changed to 7
MIN-P 1
ACC-P 1
ACC-V 7
A3
P1 7
P2 7
P2
7

PAXOS


P1
MIN-P 1 ACC-P 1 ACC-V 7
A1
P1 7
MIN-P 1 ACC-P 1 ACC-V 7
A2
P1 7
MIN-P 1 ACC-P 1 ACC-V 7
A3
P1 7 P2 7
P2
Broadcast accept phase with p2 and v7

PAXOS

Both proposer succeed! No blocking here.



PAXOS

- This is only Basic Paxos

- It has vulnerabilities
  - Is not Byzantine Fault Tolerant
  - Not easy to implement correctly
  - Not easy to achieve BFT while having termination, agreement, validity, integrity, etc

- Most consensus algorithms are variants of Paxos (e.g. Raft)

- Forms the basis of Distributed Computing research

- **Nakamoto's consensus (bitcoin proof-of-work) skirts the byzantine issue using economics**

SEGMENT I - BREAK TIME

QUESTION & ANSWER SESSION

15 mins

SESSION BREAK

Next: Bitcoin Protocol Part 1

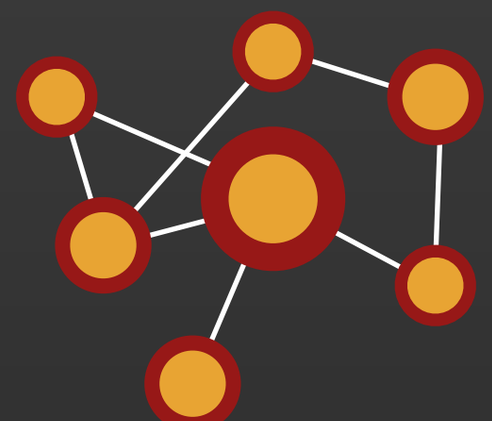
BITCOIN PROTOCOL

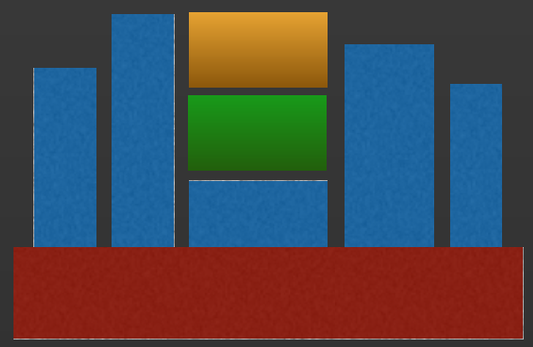
Understanding how it works

BITCOIN PROTOCOL OVERVIEW

- What we will cover:
  - Bitcoin Traits
  - Data Structures
    - Block format
    - Transaction format
    - Inputs & Outputs
  - Consensus Protocol
    - Proof-of-work
    - Longest Chain

BITCOIN TRAITS





Bitcoin stores a ledger

- Each node stores a complete copy of all transactions
- Miner nodes work to secure the ledger from attacks and also mint new coins

Bitcoin is a network

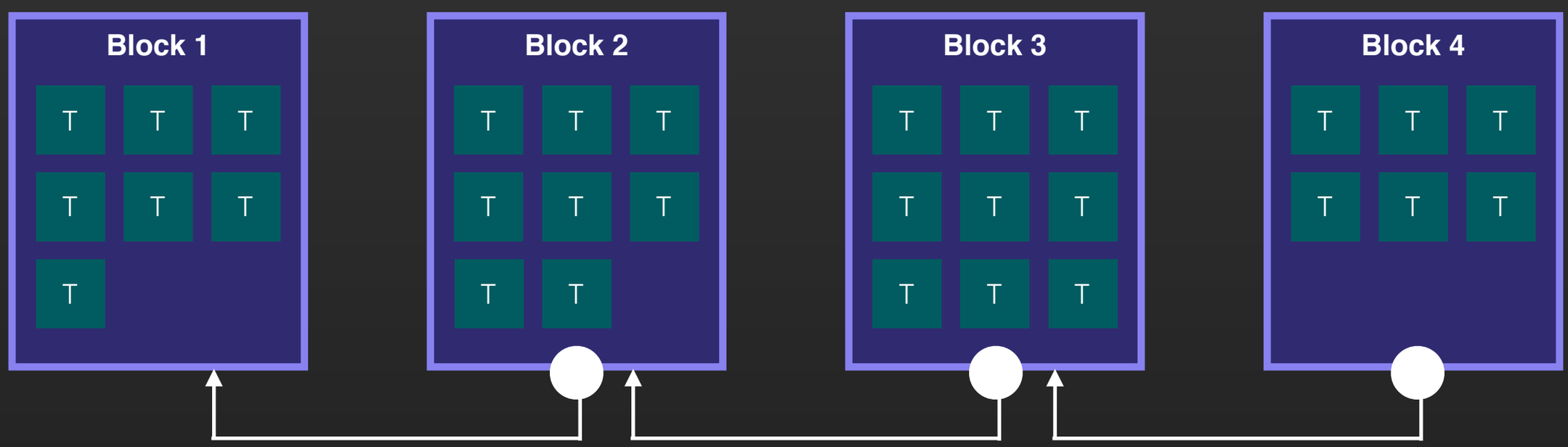
- Fully distributed, no central authority controls it
- Open participation where anybody can join, or even start mining coins

Bitcoin is p2p digital cash

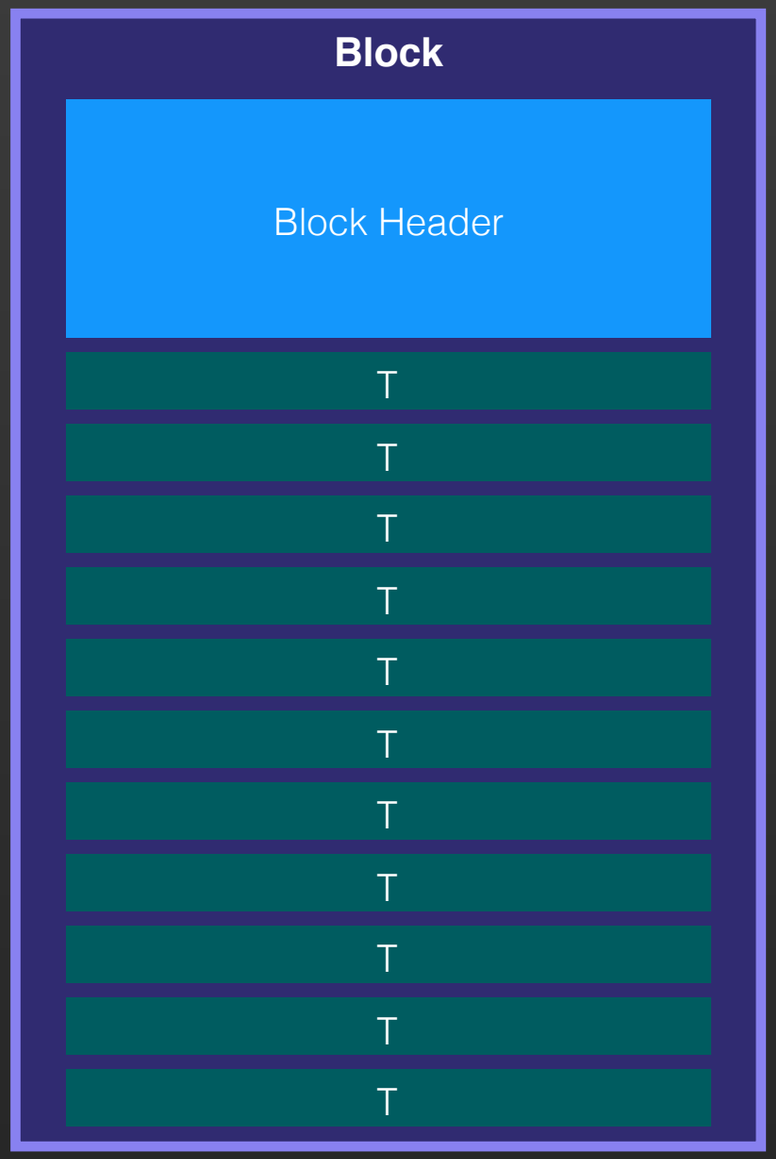
- Enables almost instant payments to anybody anywhere in the world
- Has escrow and multi-signature features, thus making it possible to automate settlements

DIVING DEEPER INTO BITCOIN

- A bitcoin node stores a ledger of all transactions that has ever happened
- The ledger is made out of blocks
- Each block contains a series of transactions at a specific point of time
- Each block contains condensed information about the previous block
  - This creates a chain of blocks
  - Chain cannot be broken easily
  - Unbreakable chain leads to immutability of data



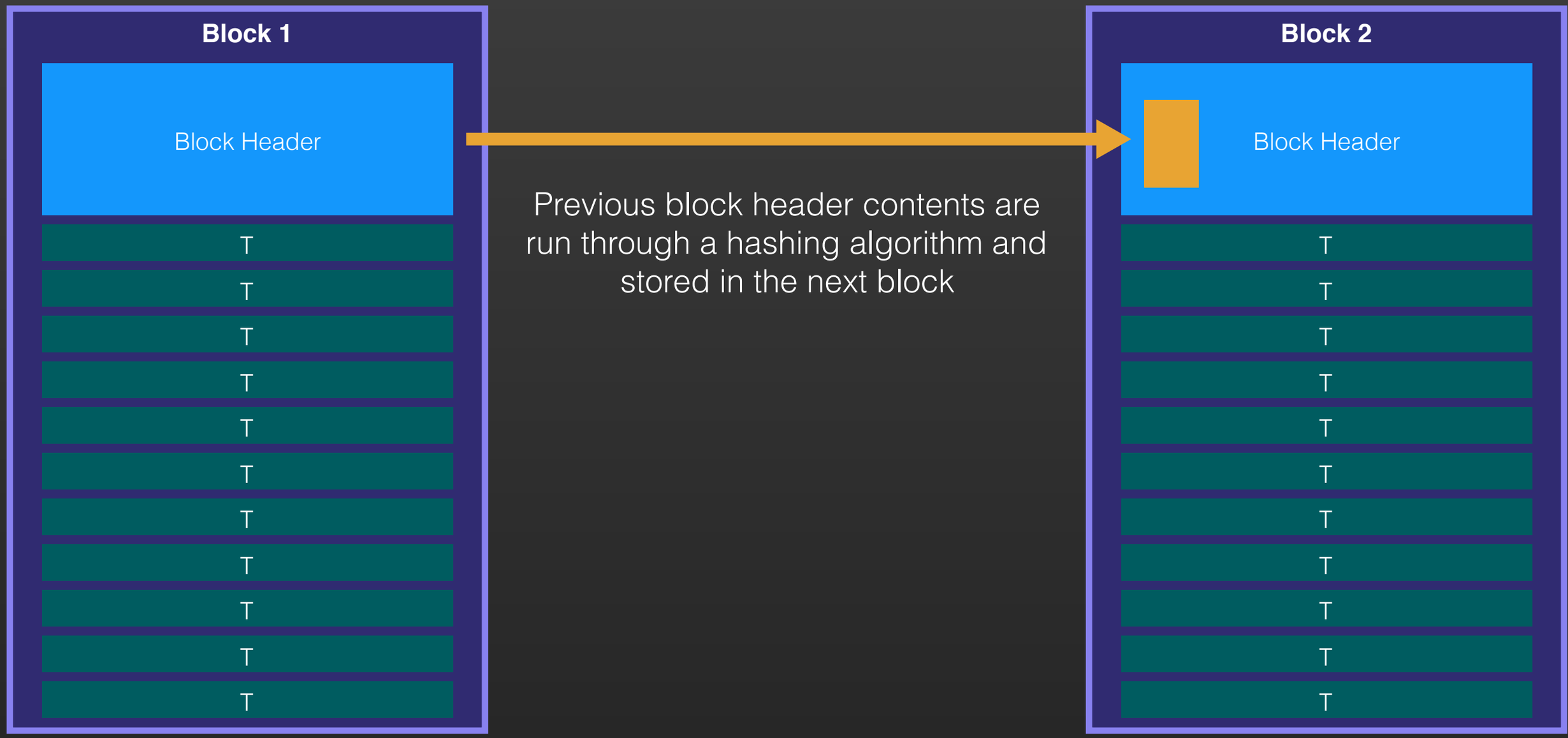
BLOCK FORMAT



Block Format

- Blocks are the most basic primitive in bitcoin
- Each block is composed of a block header + a list of transactions
- Block hash = *sha256(sha256(block contents))*
- Block header contains:
  - Version
  - **Hash of previous block header**
  - **Merkle root hash**
  - Time
  - Block Difficulty
  - **nonce**

BLOCK FORMAT

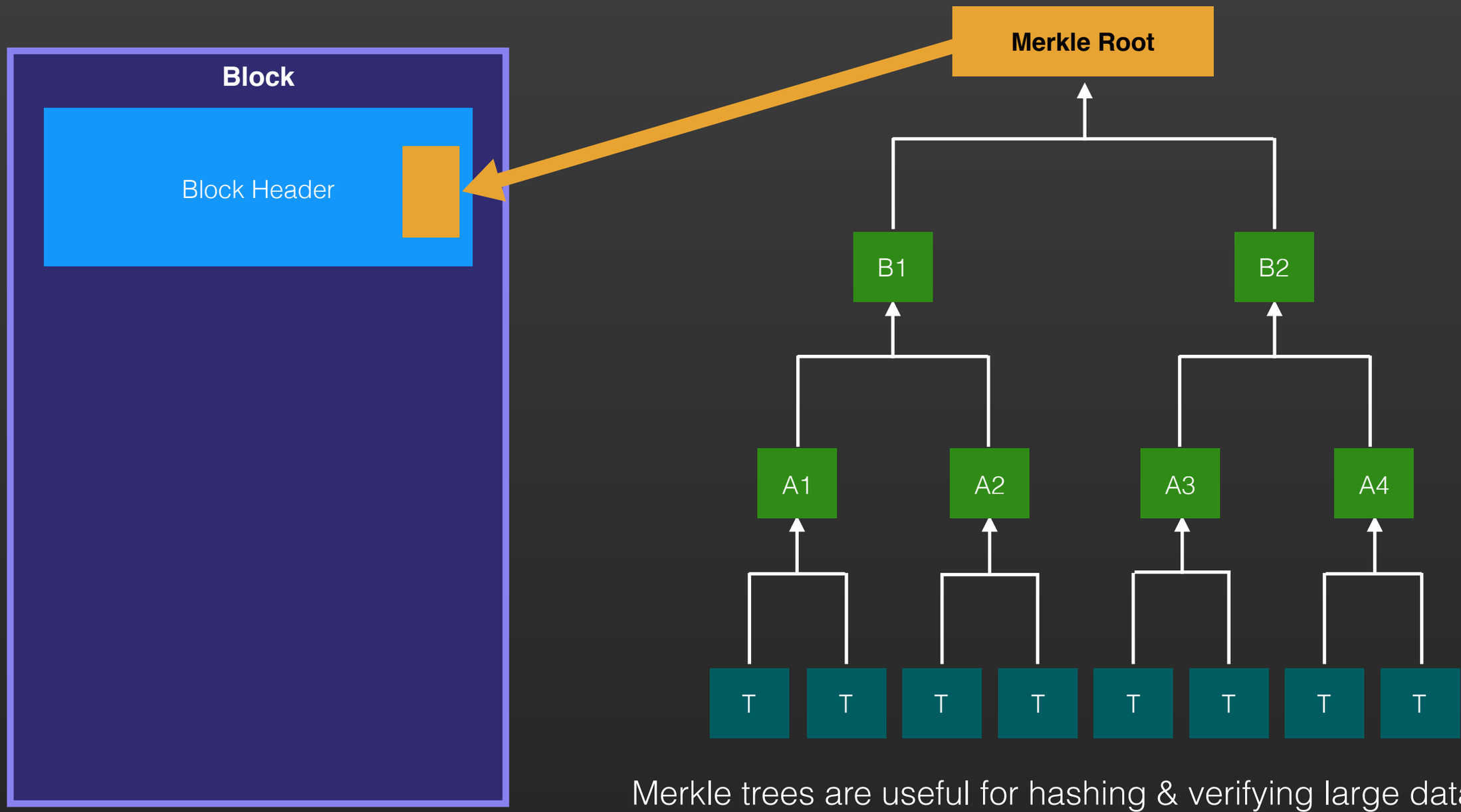


BLOCK FORMAT

curl https://api.blockcypher.com/v1/btc/main/blocks/120000

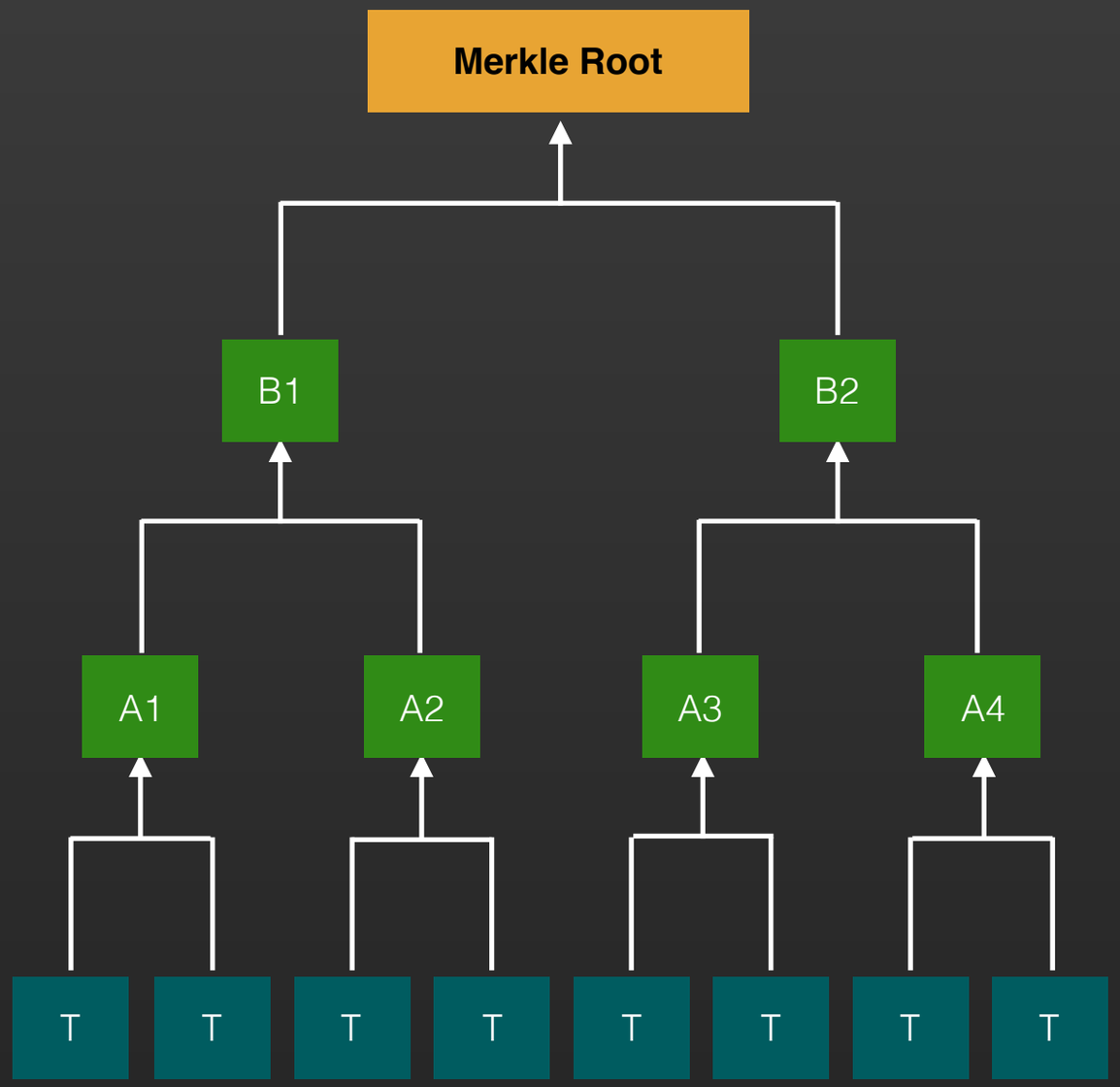

```
{
  "hash": "0000000000000e07595fca57b37fea8522e95e0f6891779cfd34d7e537524471",
  "confirmations": 346454,
  "strippedsize": 18483,
  "size": 18483,
  "weight": 73932,
  "height": 120000,
  "version": 1,
  "versionHex": "00000001",
  "merkleroot": "6dbba50b72ad0569c2449090a371516e3865840e905483cac0f54d96944eee28",
  "tx": [
    "1829755bdbe84f5a1ca579c1ed2f78a051a817b66ed4dfff35704cd6d4d644f9",
    ...
  ],
  "time": 1303687201,
  "mediantime": 1303684718,
  "nonce": 4273989260,
  "bits": "1b00b5ac",
  "difficulty": 92347.59095209427,
  "chainwork": "0000000000000000000000000000000000000000000003d1f1b76f3db5e74",
  "previousblockhash": "000000000000337828025b947973252acf8d668b3bb459c1c6e70b2e5827bca4",
  "nextblockhash": "00000000000218e1979853b278e527b6c447603942b461014964c4368c27482"
}
```

BLOCK FORMAT



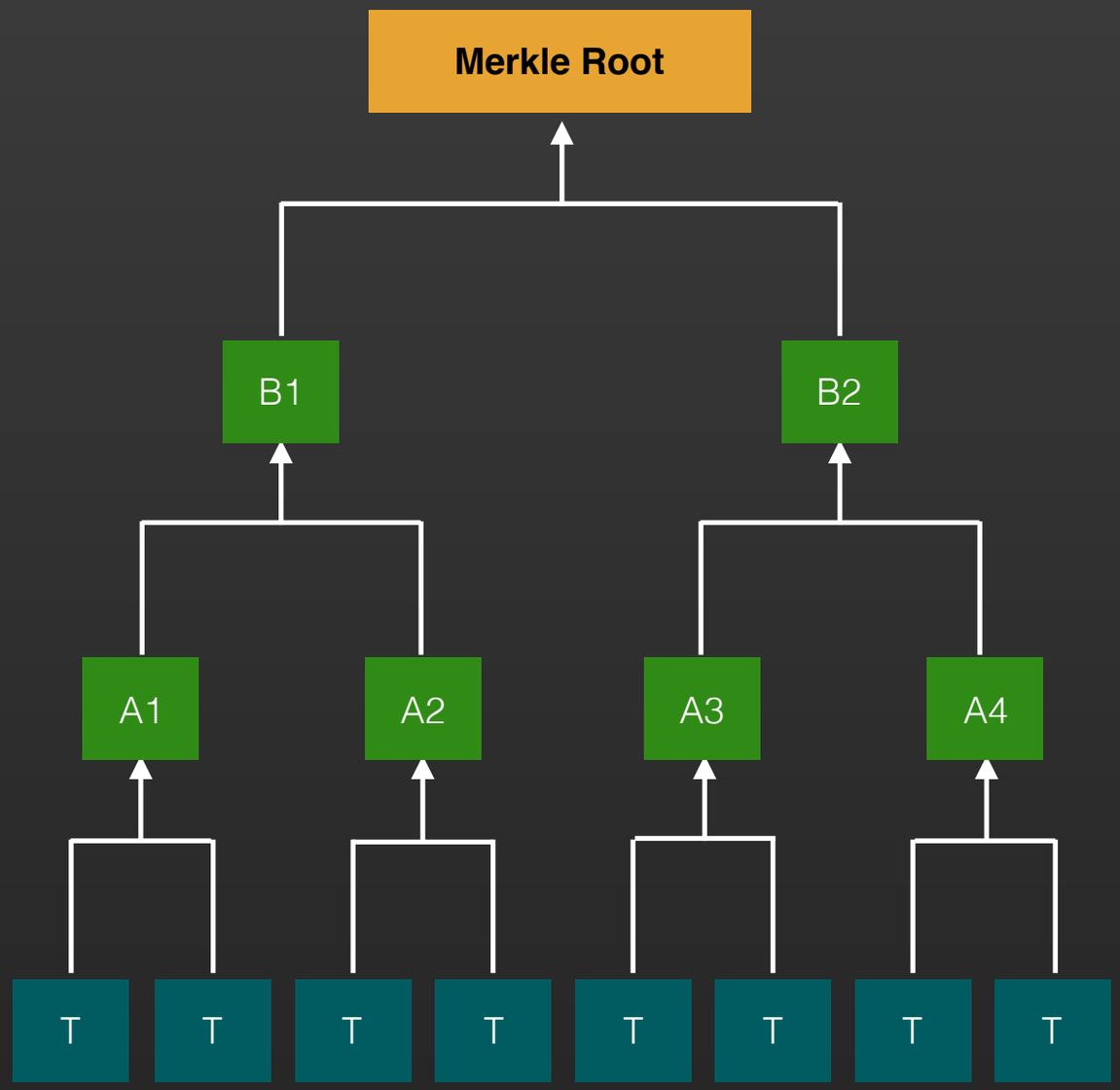
Merkle trees are useful for hashing & verifying large datasets

MERKLE TREE



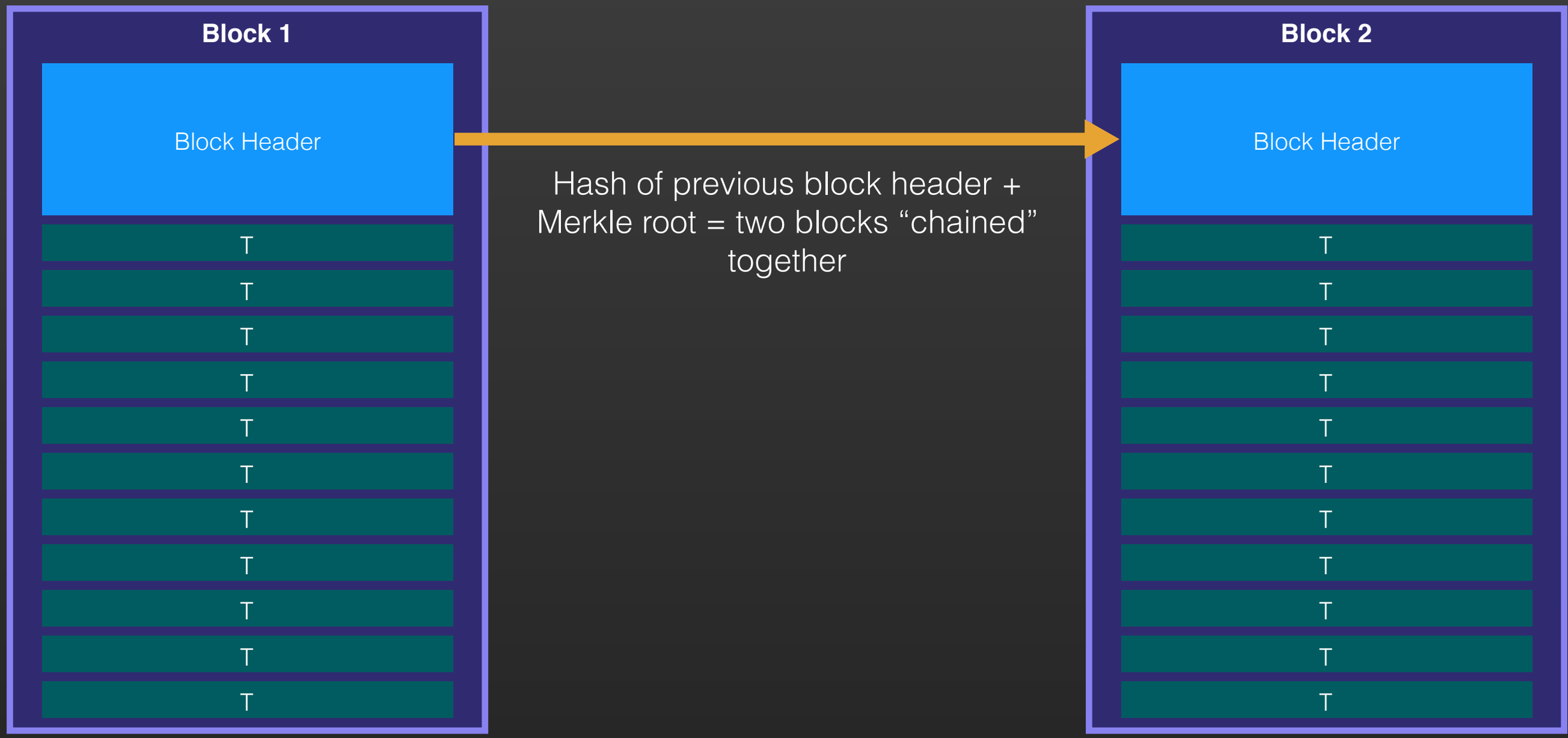
- Merkle trees allows efficient verification of large datasets/data structures
- Used in bitcoin to hash all transactions in a block
- Also used in:
  - IPFS, Btrfs, ZFS file systems
  - BitTorrent protocol
  - Apache Wave protocol
  - Git
  - Apache Cassandra, Riak, Dynamo
  - Certificate Transparency framework

MERKLE TREE

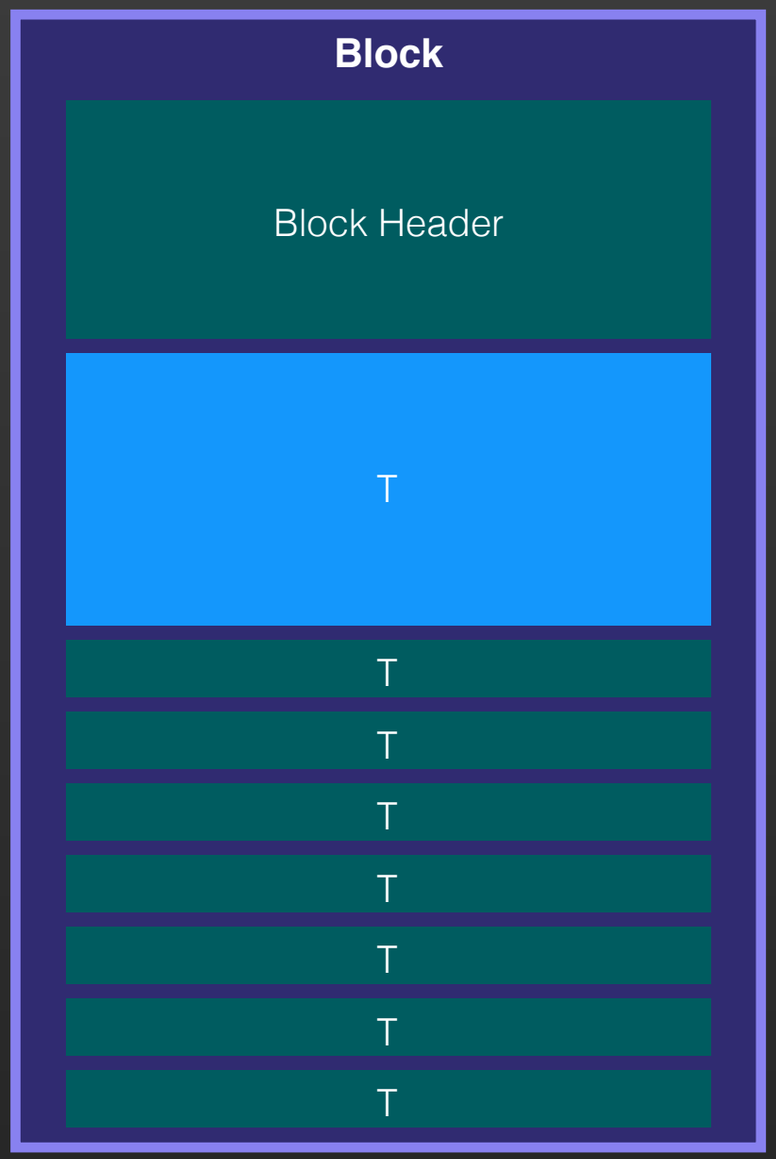


- Each raw transaction is hashed to create a txid - **sha256(sha256(x))**
- Each binary leaf is hashed using the same algorithm to create the parent node
- This continues up the merkle tree. Non-binary leafs are duplicated and then hashed in the same way
- If anything within a transaction, or the order of transactions are changed, the merkle root will also change

BLOCK FORMAT

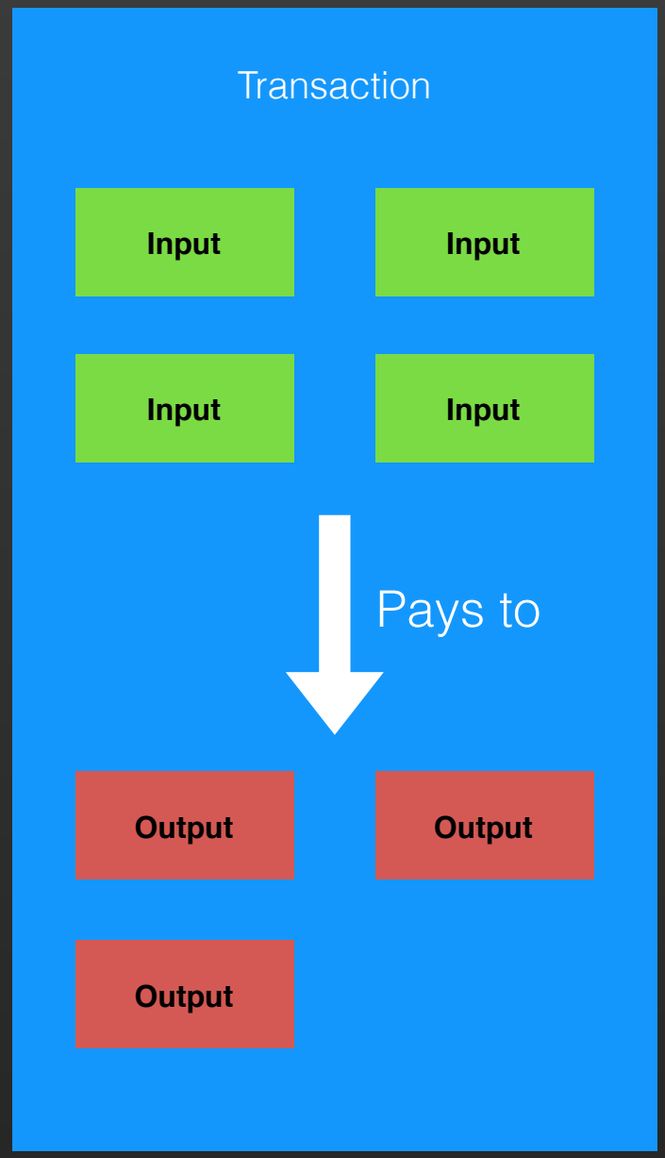


TRANSACTION FORMAT

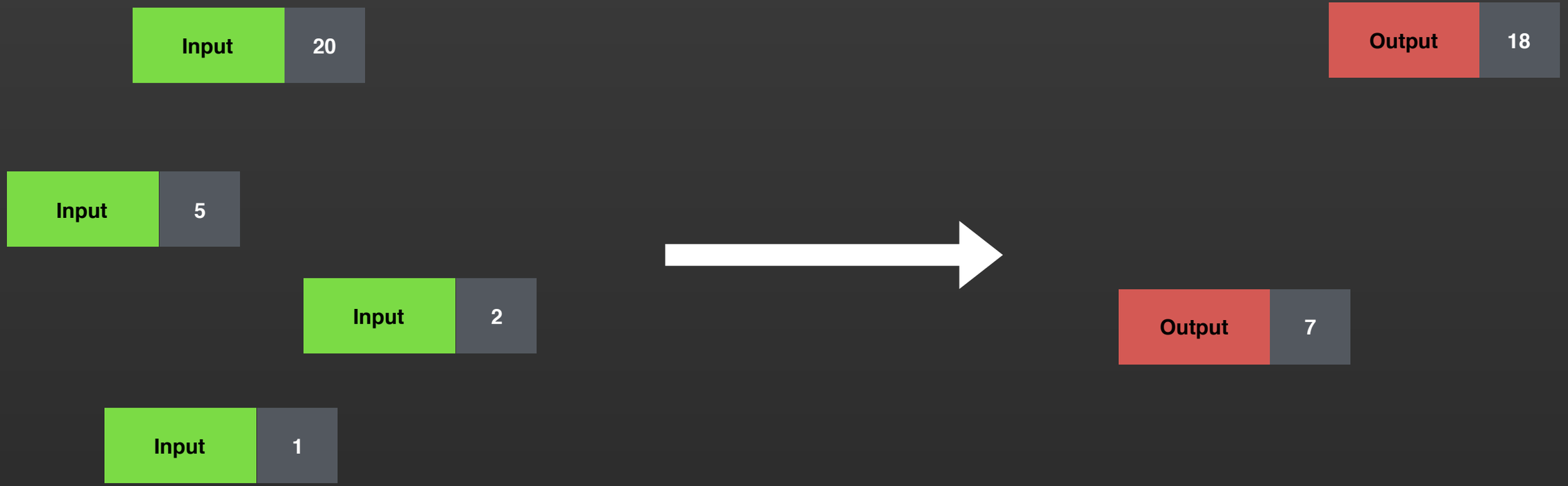


Transaction Format

- Version
- Total number of inputs
- List of inputs
- Total number of outputs
- List of outputs
- Lock time

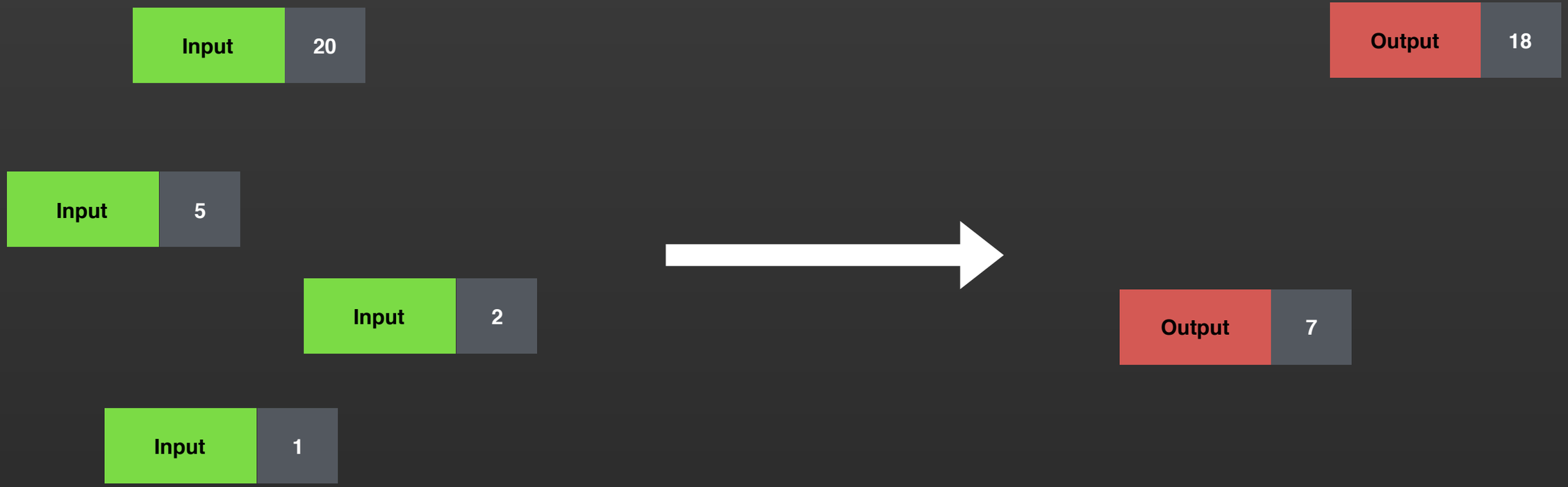


TRANSACTION - INPUTS AND OUTPUTS



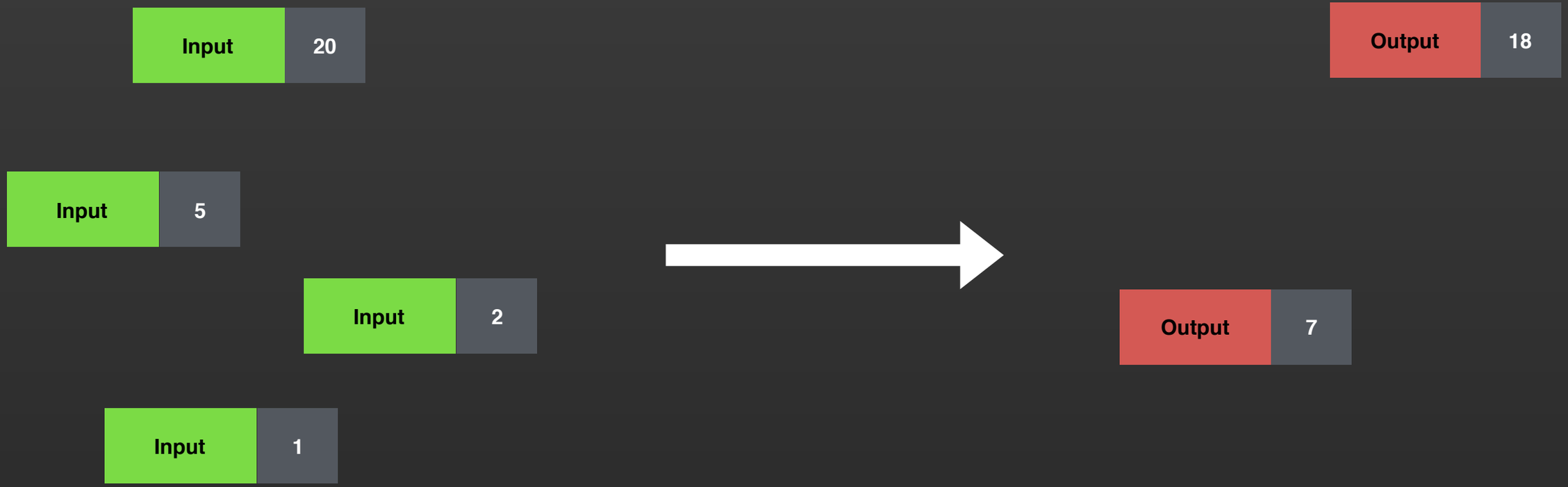
Total input ?

TRANSACTION - INPUTS AND OUTPUTS



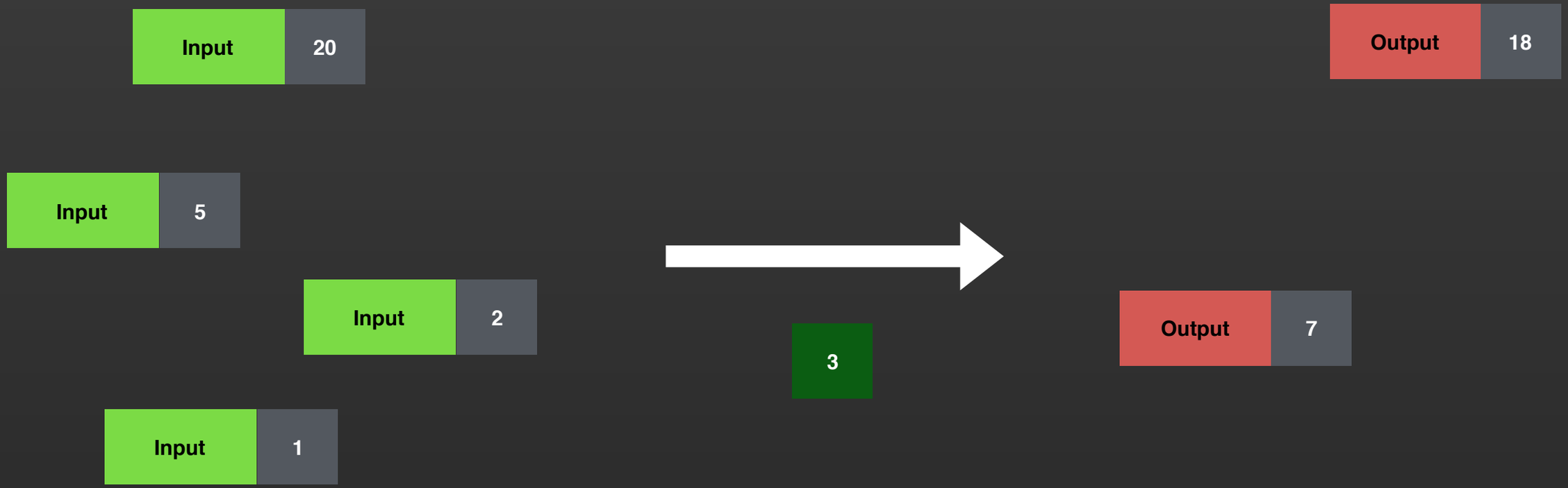
Total output ?

TRANSACTION - INPUTS AND OUTPUTS



Total fees ?

TRANSACTION - INPUTS AND OUTPUTS



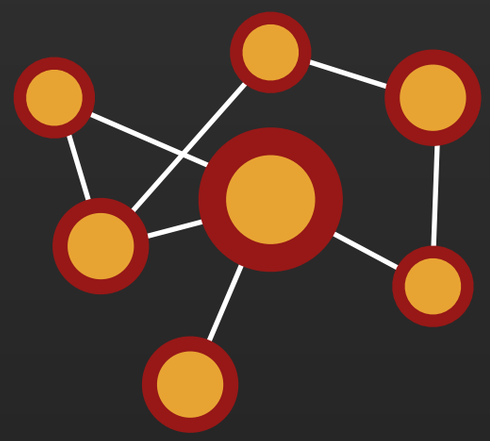
Fees = Total input - Total output
= 28 - 25
= 3

TRANSACTION FORMAT

```
curl https://api.blockcypher.com/v1/btc/main/txs/
1829755bdbe84f5a1ca579c1ed2f78a051a817b66ed4dfff35704cd6d4d644f9
```

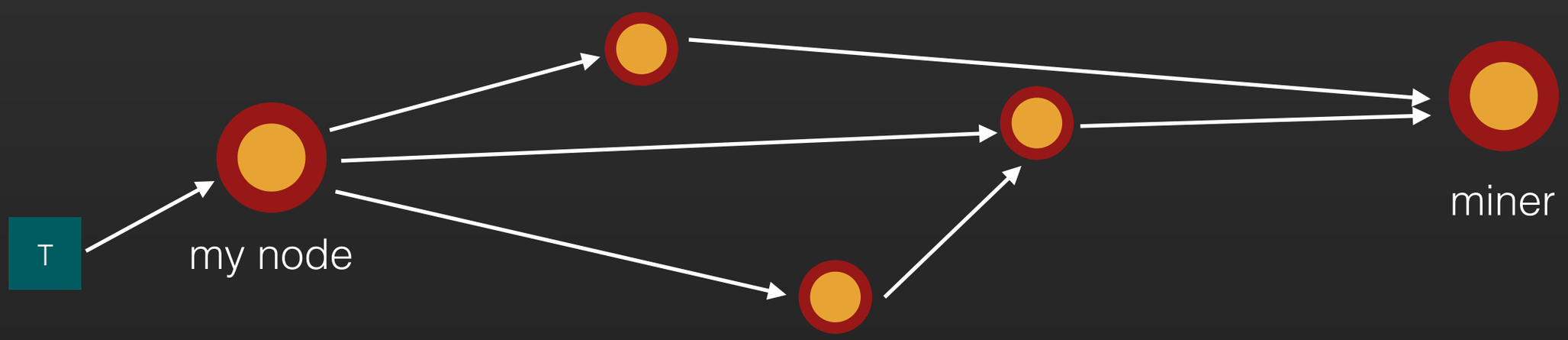
BITCOIN NETWORK

- Each node is connected to one another, maintaining a list of peers
- Nodes stay consistent with each other through a consensus algorithm called proof of work
- New transactions can be created on any node, and once created is relayed to other nodes
- Transactions are packed into new blocks on the blockchain through a process called mining



PEER TO PEER GOSSIP

- New transactions can be created on any node, and once created is relayed to other nodes
  - Any node can create a transaction
  - New transactions are relayed to other nodes using a gossip protocol
  - New transactions reside in a **pool**
  - Miner nodes eventually will receive a copy of the transaction

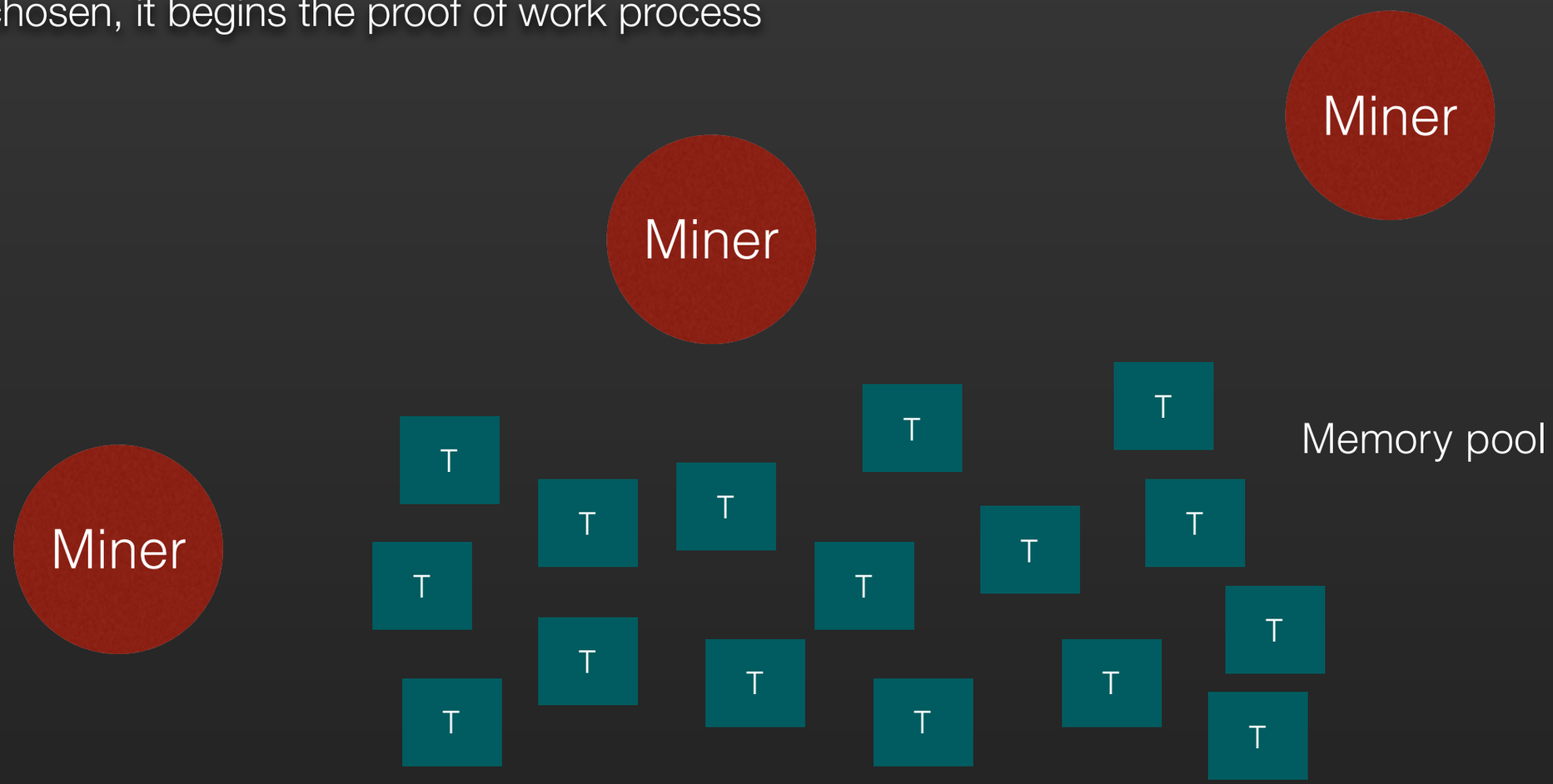


MINING

- Miners are special nodes with the capability to **add blocks to the chain**
- Miners ensure that everybody's data is always consistent
- Miners are also a source of new coins
    - Every time a new block is added to the chain, the miner that added it is rewarded with newly created bitcoins
- Miners must perform a **proof-of-work** to gain the right to add a block to the chain

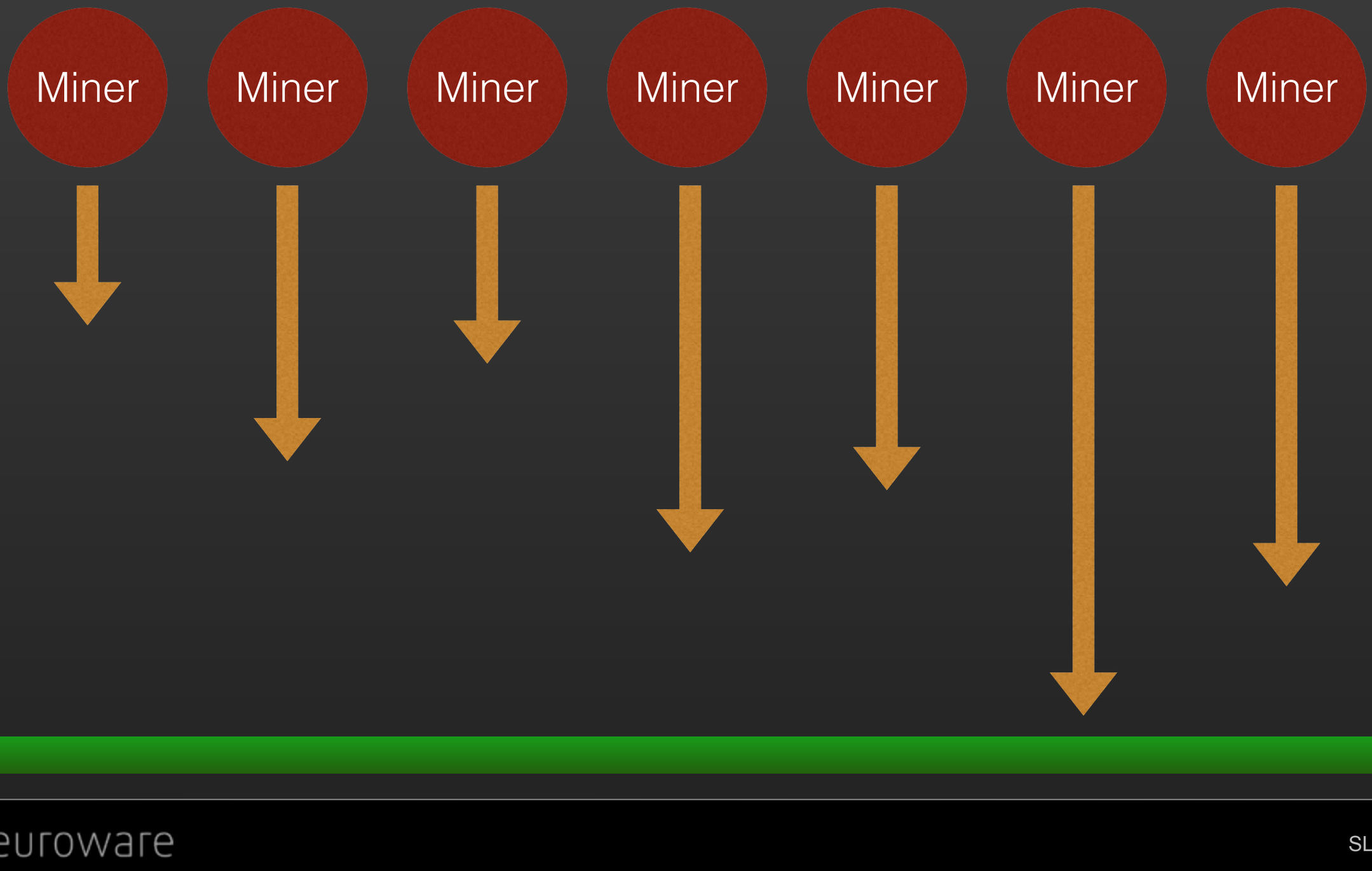
MINING

- All unconfirmed transactions goes into a memory pool
- Each miner must pick a list of transactions to create a block
- Different miner have different criteria of which transactions to pick
- Once a transaction is chosen, it begins the proof of work process



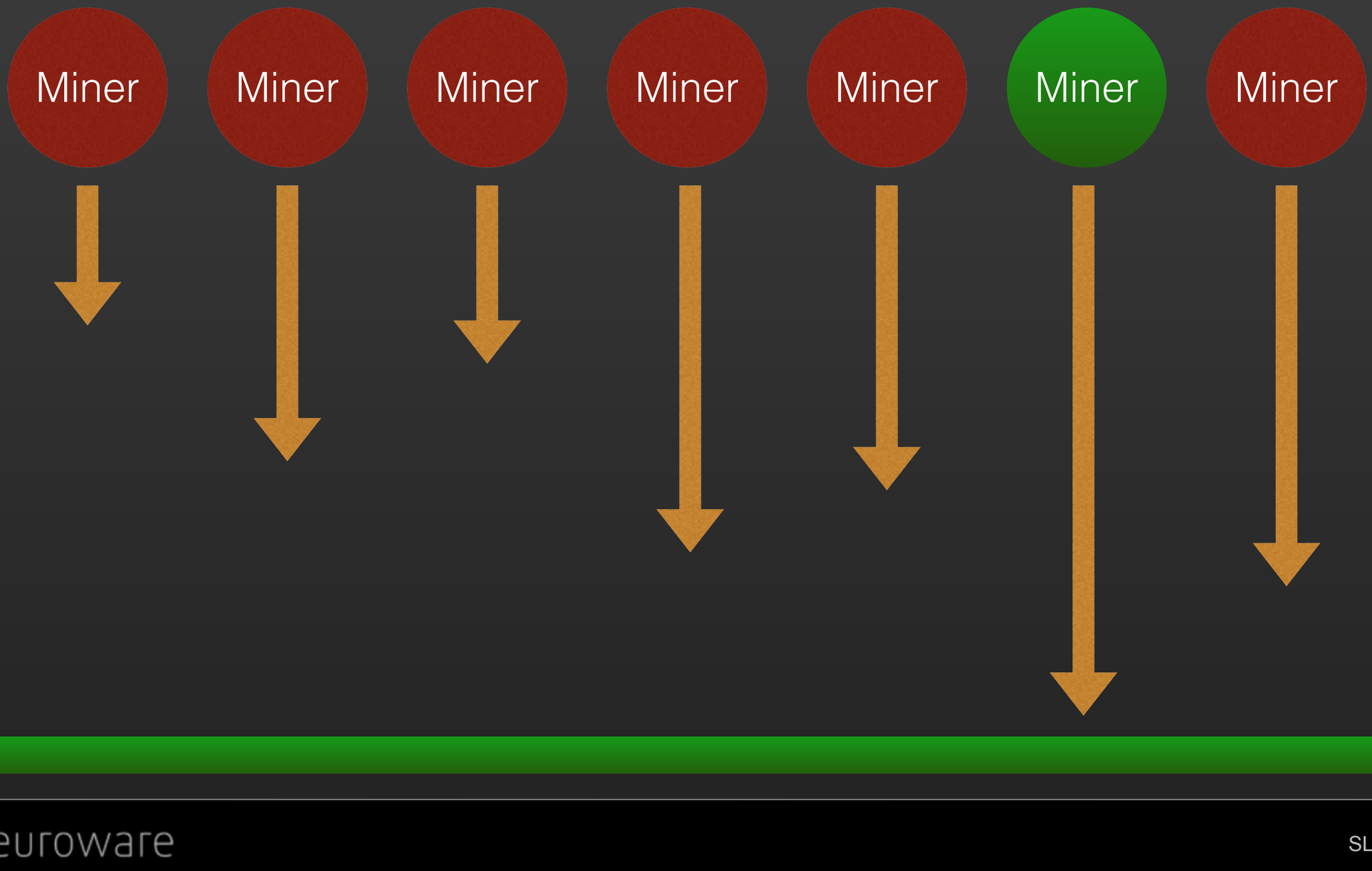
MINING PROCESS

Thousands of miners race to solve a math puzzle



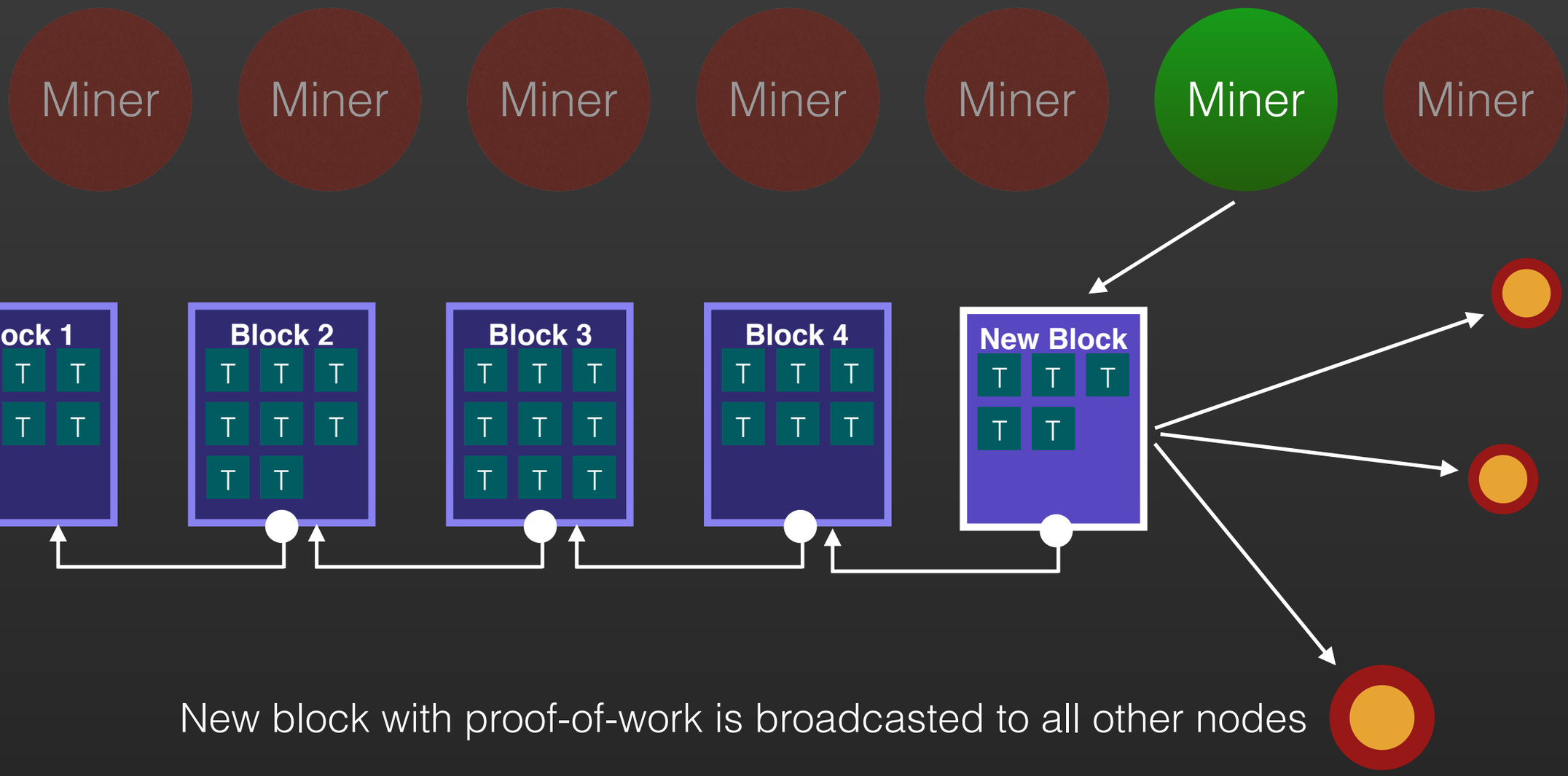
MINING PROCESS

First to solve gets the right to **add a new block**



MINING PROCESS

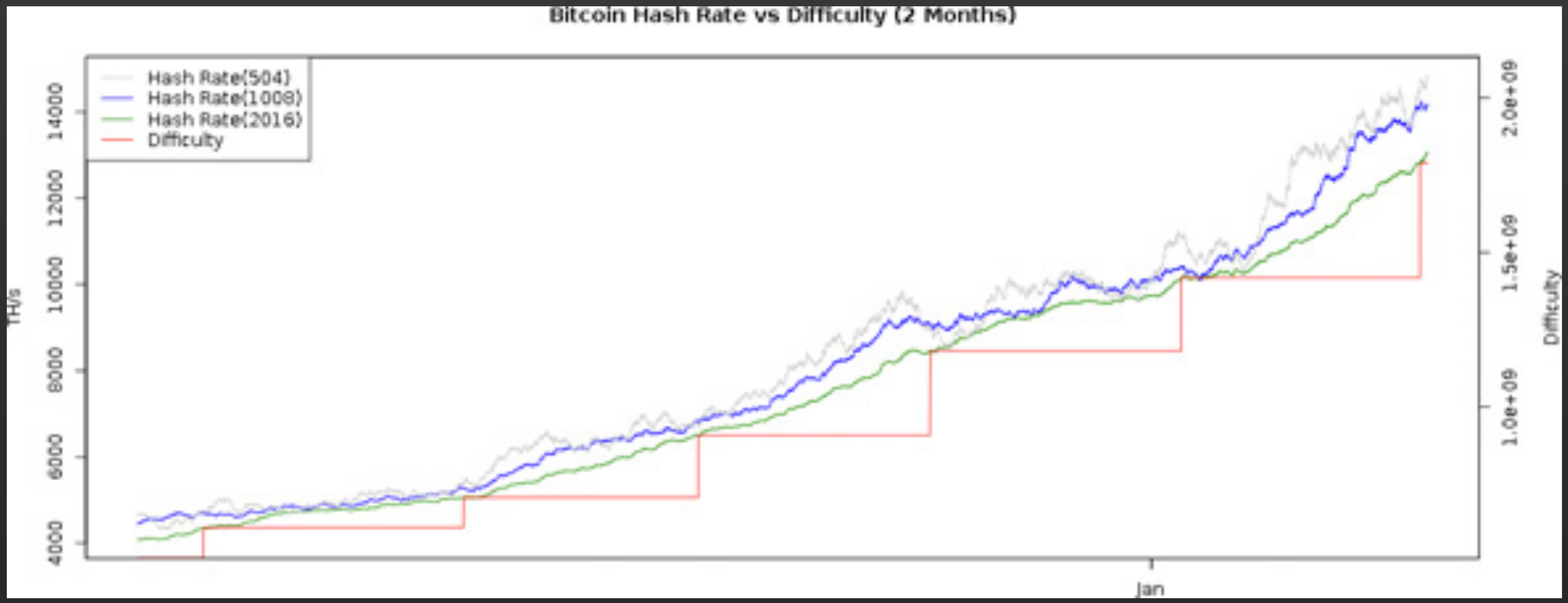
Miner is rewarded with **new coins** and **transaction fees**



New block with proof-of-work is broadcasted to all other nodes

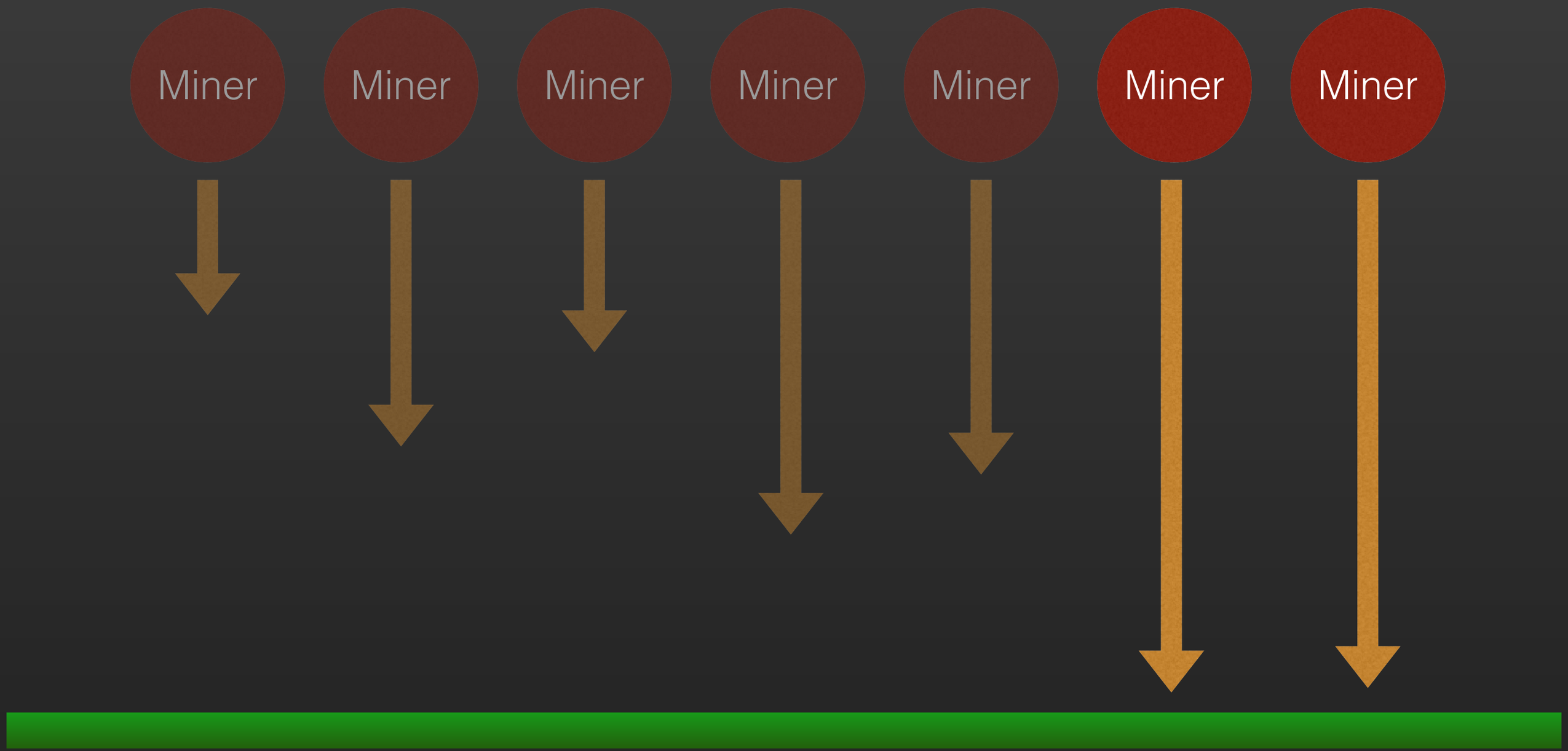
AUTO-ADJUSTING DIFFICULTY

Auto-adjusting difficulty every 2016 blocks acts as **traffic control**



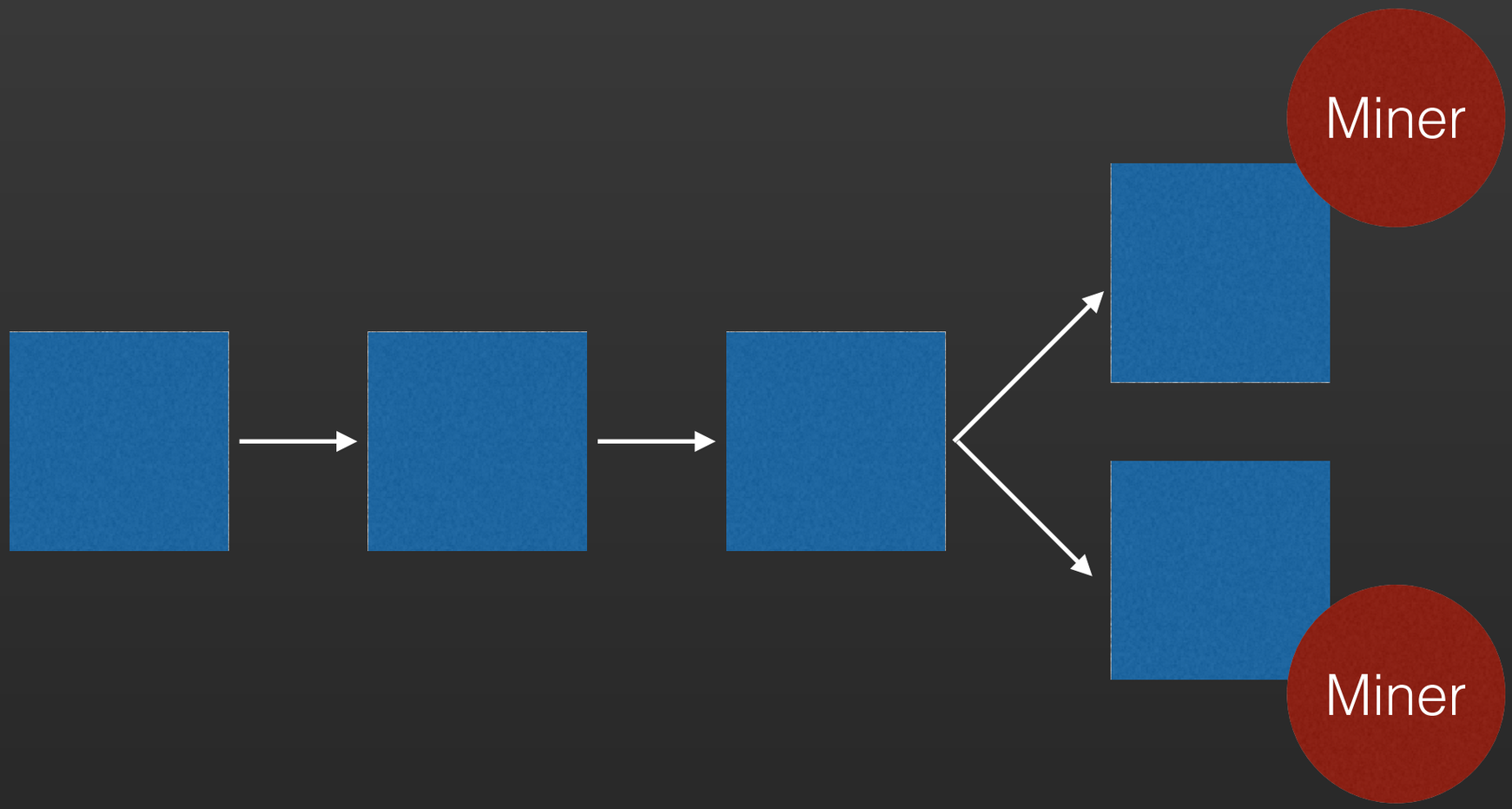
FORKING

What happens when 2 miners find the solution simultaneously?



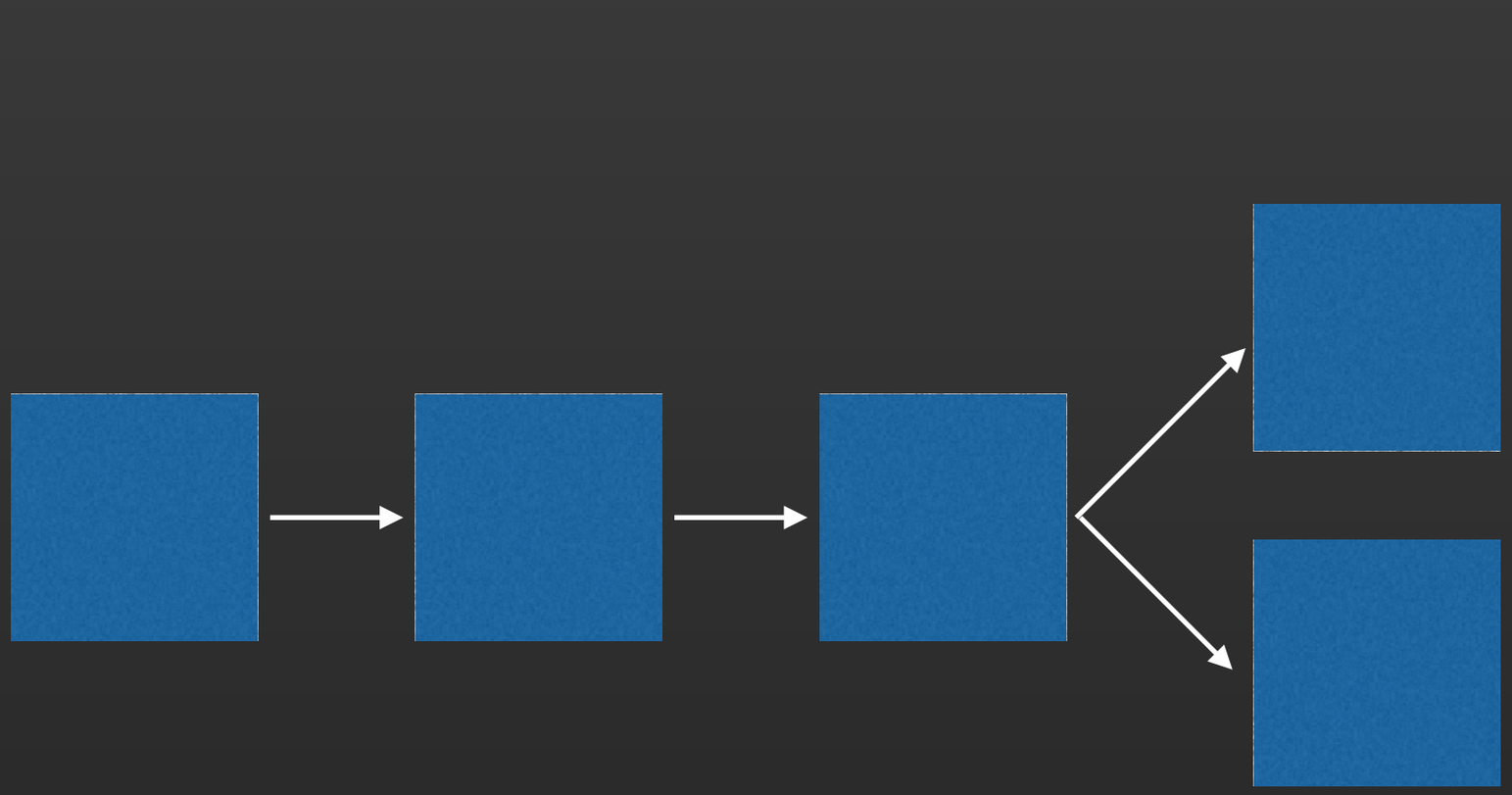
FORKING

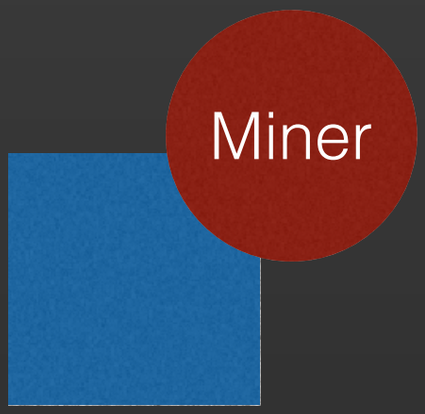
A fork happens!



FORKING

Longest-chain protocol comes into effect

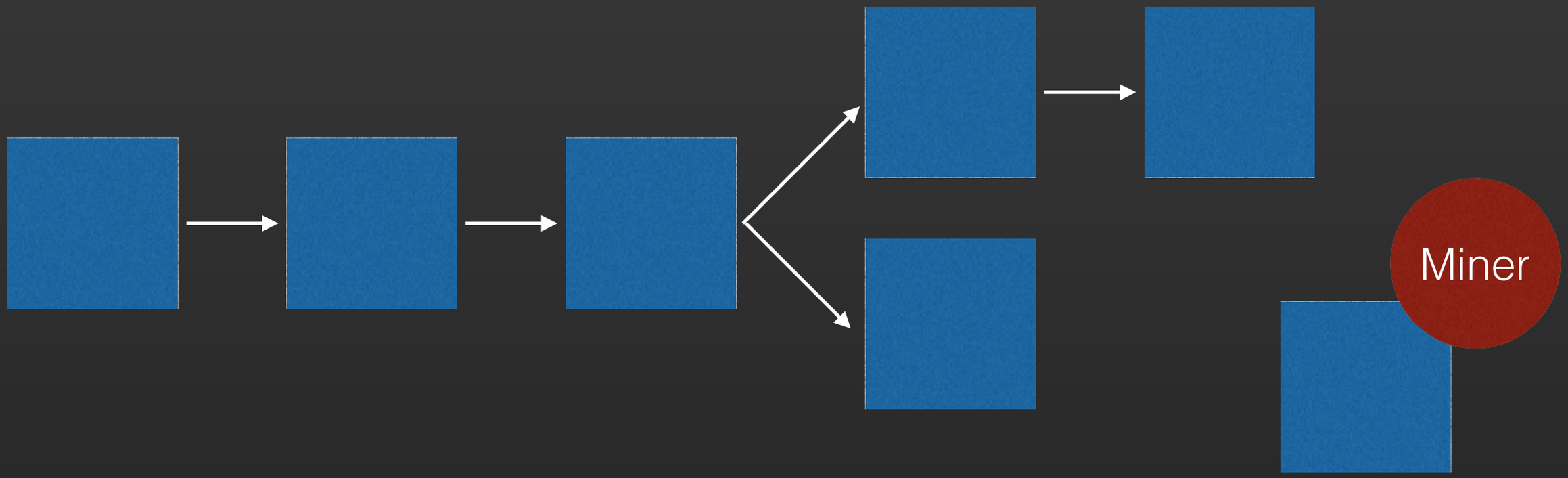




Randomly picks one chain if both forks are the same length

FORKING

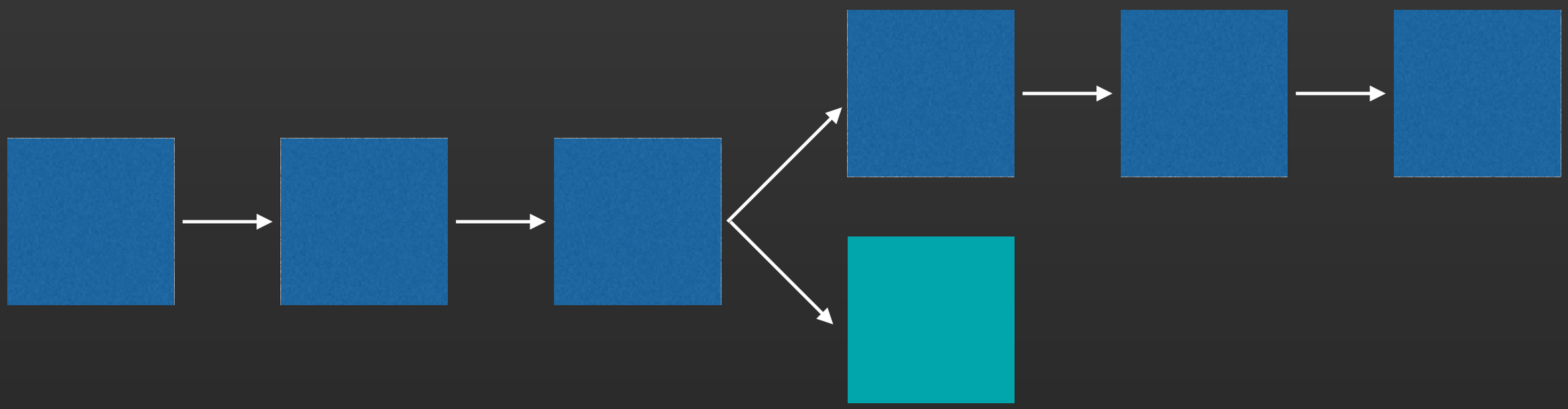
Longest-chain protocol comes into effect



If one fork is longer than the other, pick the longest fork

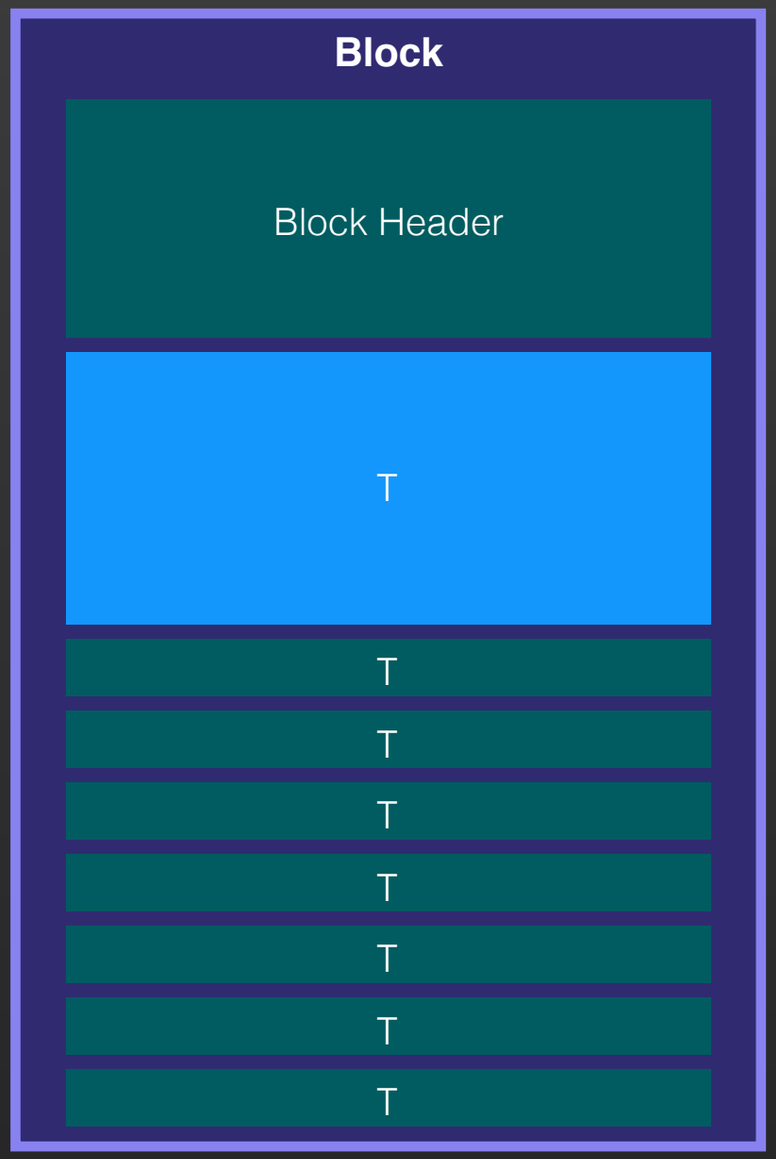
FORKING

Longest-chain protocol comes into effect



This block is now orphaned! After a while, all transactions within are returned to the memory pool

TRANSACTION FORMAT (REVISITED)

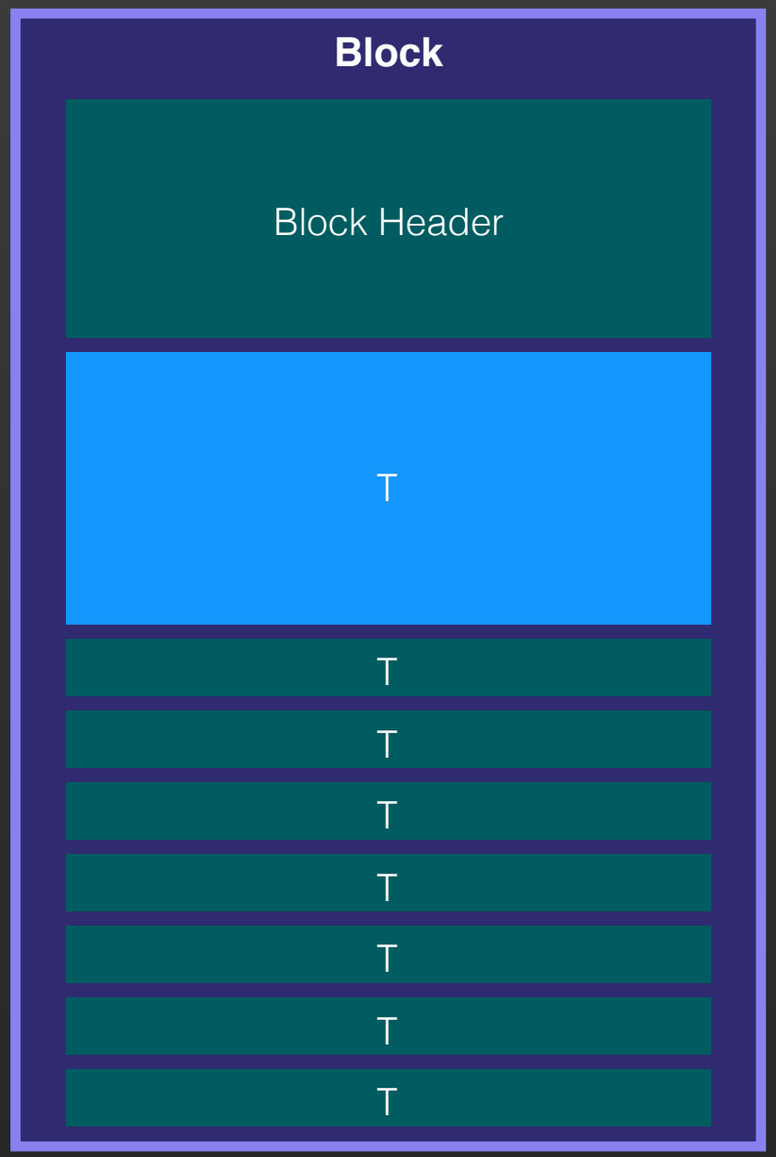


First transaction is always a **coinbase**

x = Current block reward

y = Miner's address

TRANSACTION FORMAT (REVISITED)



Coinbase

- Coinbase starts at 50 btc
- Halves every 210,000 blocks
- Current block height - 466,517 (as of 16 May 2017)
  - Halved twice (210,000, 420,000)
- Current block reward - 12.5 btc
- Coinbase + transaction fee = block reward
  - Cannot be spent until at least 100 confirmations
- Possible to have blocks without any transactions (only coinbase)

DATA IMMUTABILITY

- 3 factors gives bitcoin **byzantine fault tolerance:**
  - Chain of blocks ensures that once data is stored, it cannot be changed, or the chain will break
  - Network of miners ensure that new blocks are valid in order to reap the rewards in bitcoin
  - Auto-adjusting difficulty ensures that block intervals grow in parallel to hashing power

- Attack vectors:
  - **Sybil attack** - many malicious nodes in order to perform localised malicious operations
  - **Denial-of-service** - transaction spam, protocol attacks, resource starvation
  - **Quantum computing** - probably not a problem in the short-term
  - **51% attack** - achieve quorum power, needs a lot of money ($461,684,264.00 hardware)
  - **Double spending** - consensus protocol ensures that double spends do not happen

SEGMENT 2 - BREAK TIME

QUESTION & ANSWER SESSION

15 mins

SESSION BREAK

Next: Bitcoin protocol part 2

BITCOIN PROTOCOL II

More advanced concepts

BITCOIN PROTOCOL OVERVIEW

- What we will cover:
  - Key Management
  - Bitcoin Scripting
  - Basic Payment Contract
  - Multi-signature Technology
  - Refundable Deposits
  - Storing Data

KEY MANAGEMENT

Private Key

- Generated by your own means, or by the wallet client you use
- Must be a valid ECDSA private key format

Public Key

- Generated as part of the ECDSA private-public keypair

Bitcoin **Address**

1. SHA256 the public key
2. RIPEMD160 the result from #1
3. Add version byte to the front of result from #2
4. Double SHA256 the result from #3
5. Take the checksum (first 4 bytes) from #4, add to the end of result from #3
6. Convert into base58 string using base58check encoding

KEY MANAGEMENT

Private Key

Public Key

Bitcoin **Address**

One way hash

KEY MANAGEMENT

Private Key

- Kept secret by yourself, must not be shared with anybody else
- Lose your private key, and your wallet is gone

Public Key

- Attached to transactions in certain kind of payment contracts
- Can be given out without fear of revealing your private key

Bitcoin **Address**

- Compressed form easy to pass around
- Can be given to others when requesting payment
- Can be safely embedded in websites to solicit payments

DETERMINISTIC WALLETS

Seed

String 1
First name

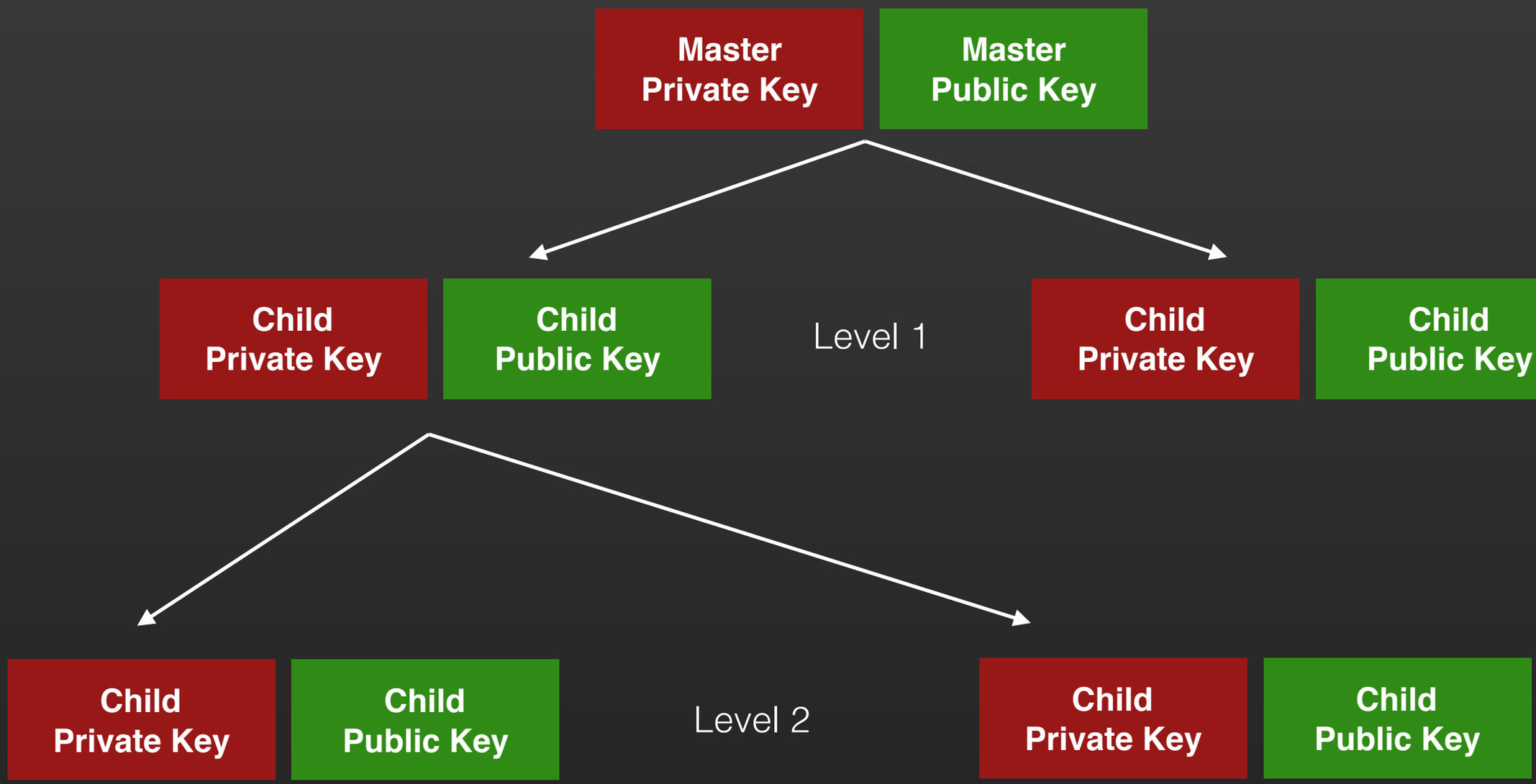
String 2
Last name

String 3
Email

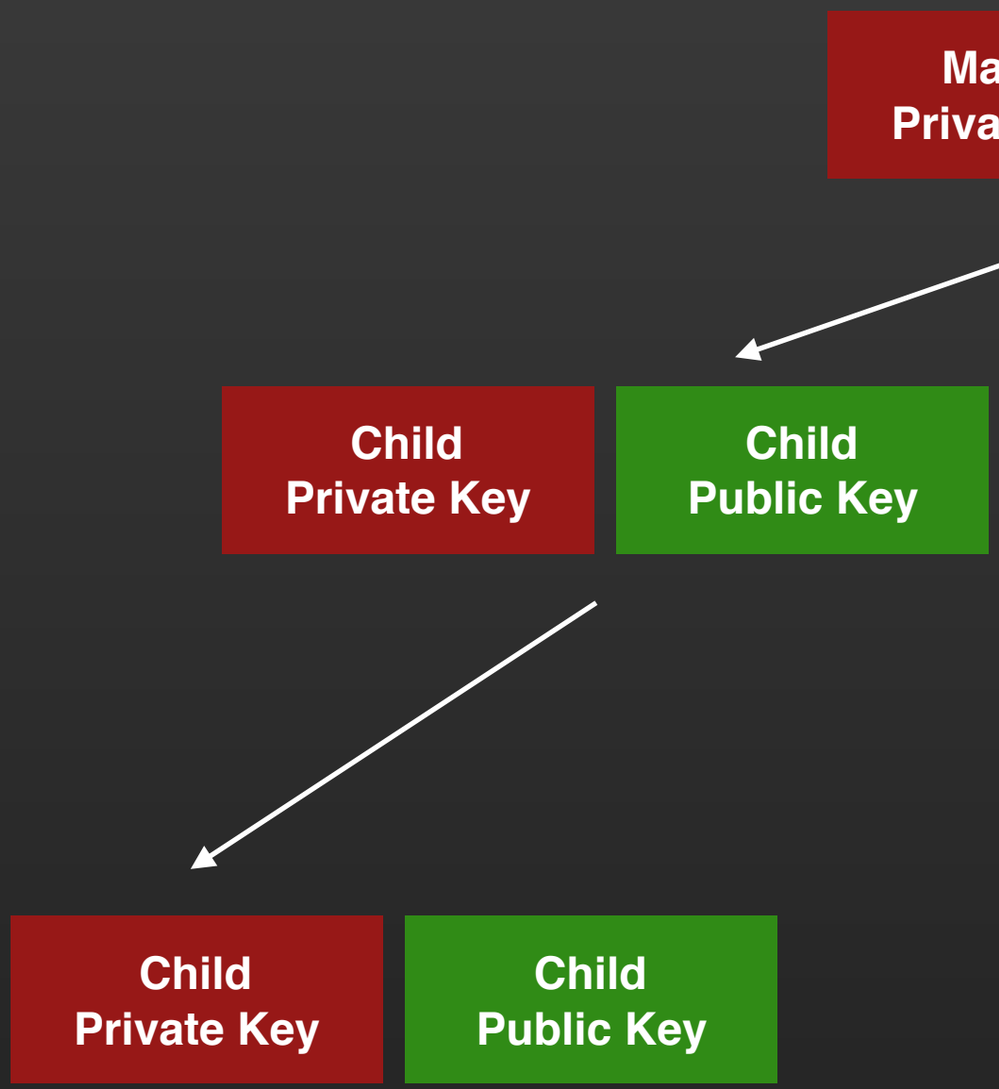
String 4
Password

String 5
Biometrics

HIERARCHICAL DETERMINISTIC WALLETS



HIERARCHICAL DETERMINISTIC WALLETS



- Master private keys can be used to derive **any** child private keys
- Child public keys can be given out to auditors to track transactions of all derived public keys, but not parent keys
- Can be used in a corporate structure where departments get their own derived keys

SETUP YOUR FIRST WALLET

Let’s get paid!

POPULAR KEY MANAGERS/WALLETS

ELECTRUM

POPULAR KEY MANAGERS/WALLETS

POPULAR KEY MANAGERS/WALLETS

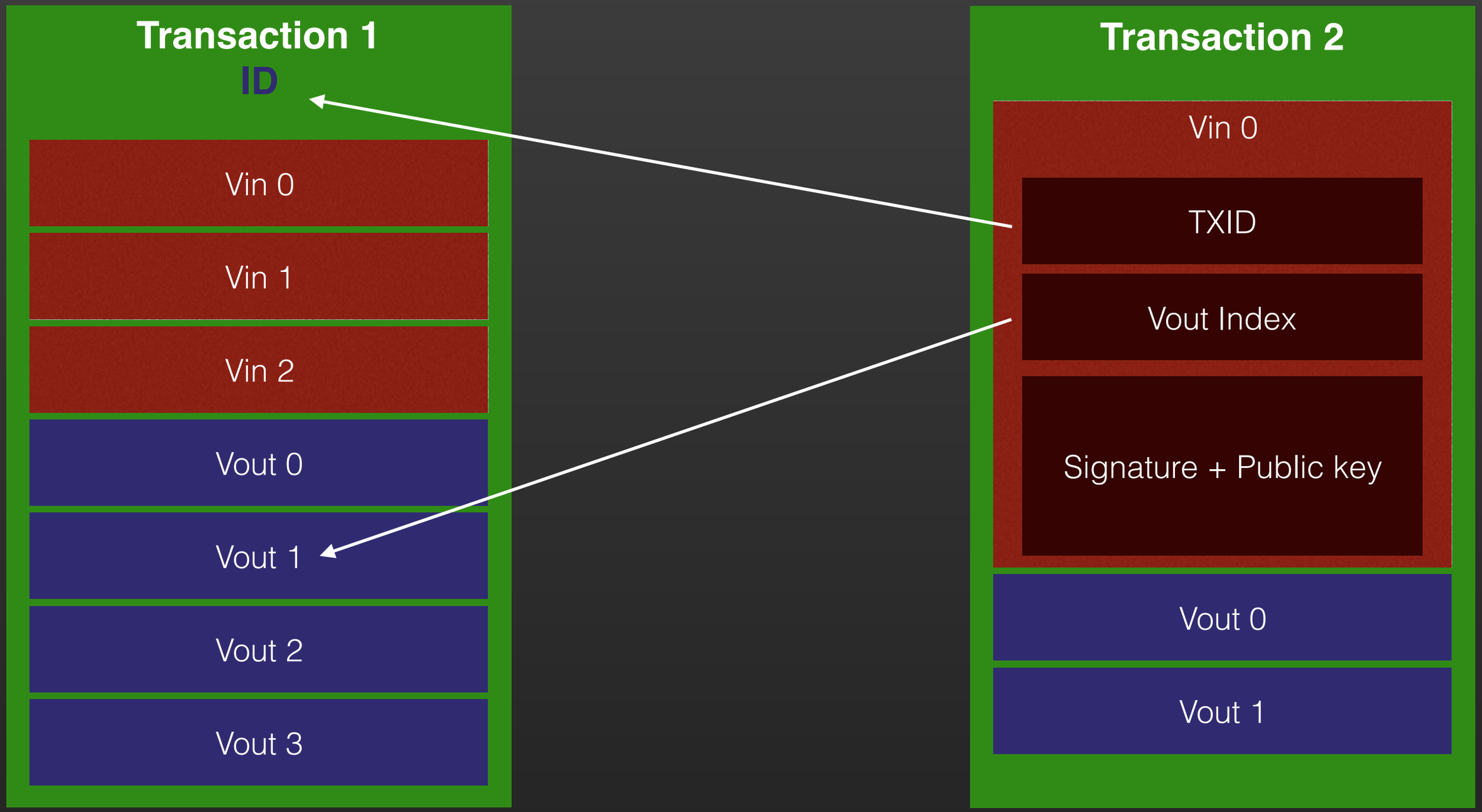
BITCOIN SCRIPTING

https://en.bitcoin.it/wiki/Script

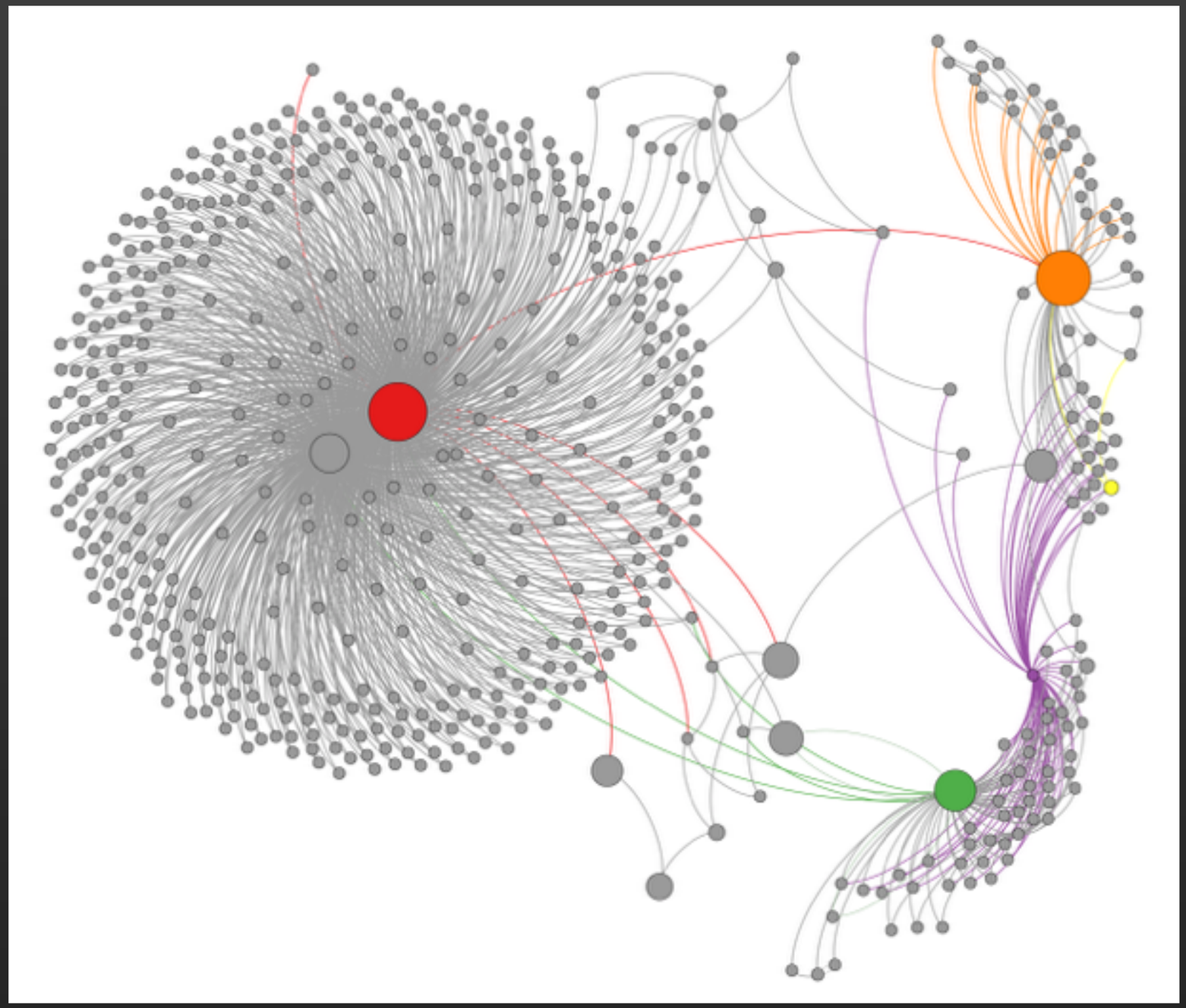
REVISITING TRANSACTIONS

- Each transaction contains a list of inputs and outputs
- Input format:
  - Previous ID of the transaction
  - Index of the output
  - scriptSig
    - Signature
    - Public Key
- Output format:
  - Value (in satoshis)
  - scriptPubKey
    - Script to redeem the value

REVISITING TRANSACTIONS



REVISITING TRANSACTIONS



BITCOIN SCRIPTS

- Bitcoin has a simple scripting language that resembles Forth
- The scripting language it NOT turing complete, and does not contain loops
- A script basically determines how the person that wants to spend bitcoin can gain access to the bitcoins
- Powerful enough to allow various contracts:
  - Refundable deposits
  - Escrow
  - Storing immutable public data

BITCOIN SCRIPTS

Input

Vin 0

Signature

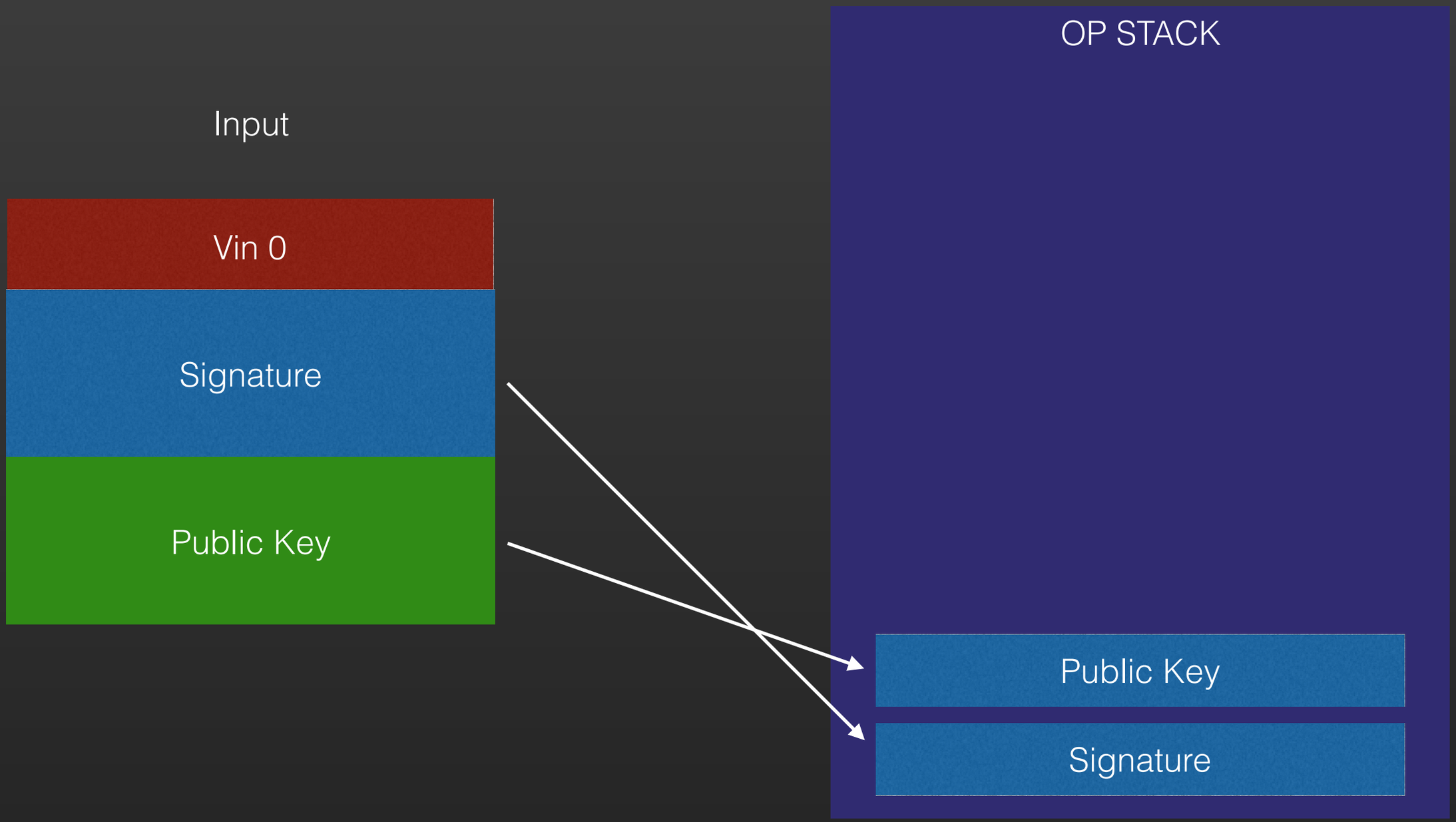
Public Key

Output

Vout 0

```
OP_DUP OP_HASH160
c398efa9c392ba6013
OP_EQUALVERIFY
OP_CHECKSIG
```

BITCOIN SCRIPTS



BITCOIN SCRIPTS

Output

Vout 0

```
OP_DUP OP_HASH160
c398efa9c392ba6013
OP_EQUALVERIFY
OP_CHECKSIG
```

OP STACK

OP_DUP

Public Key

Signature

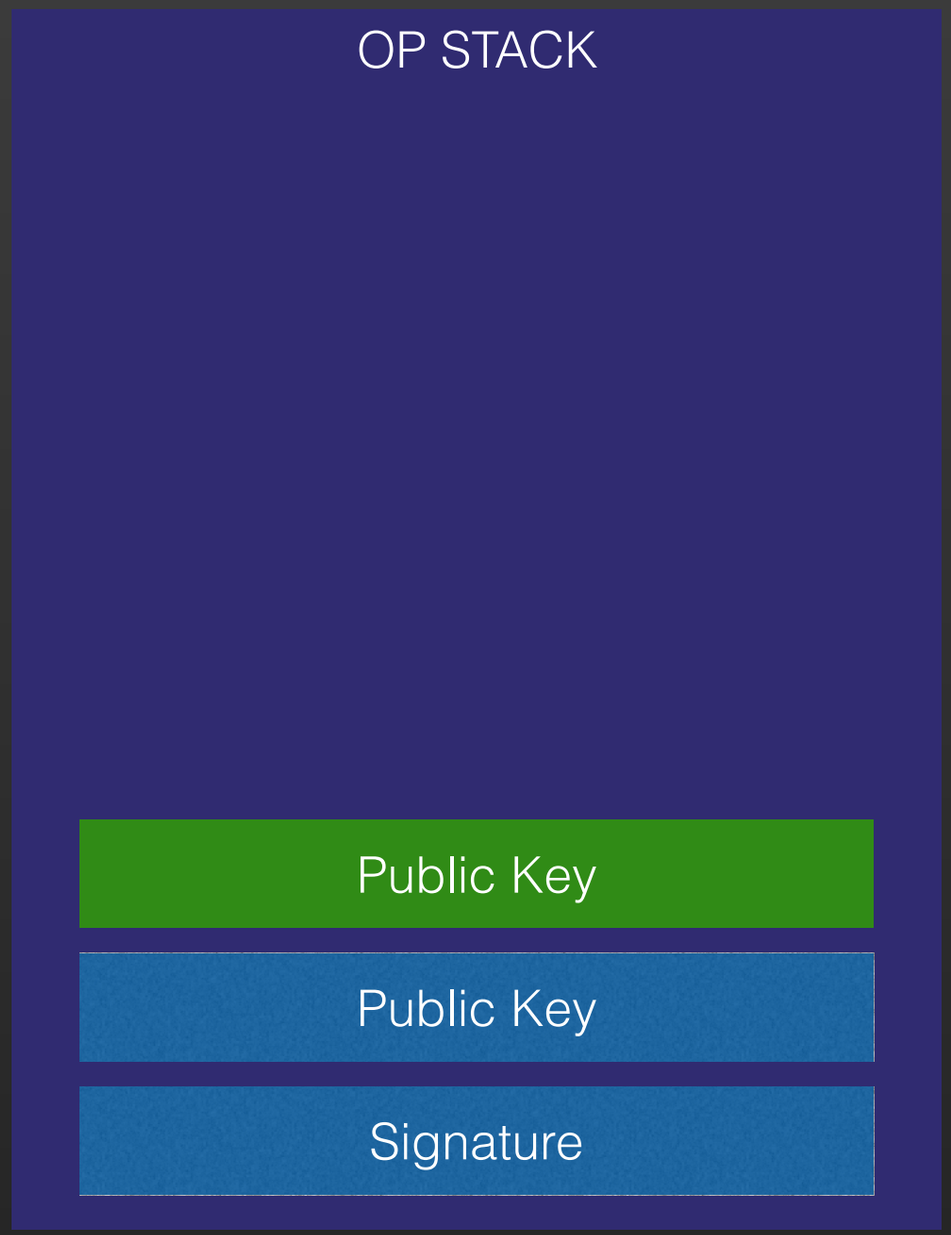
BITCOIN SCRIPTS

Output

Vout 0

```
OP_DUP OP_HASH160
c398efa9c392ba6013
OP_EQUALVERIFY
OP_CHECKSIG
```

OP_DUP - duplicate the top list item

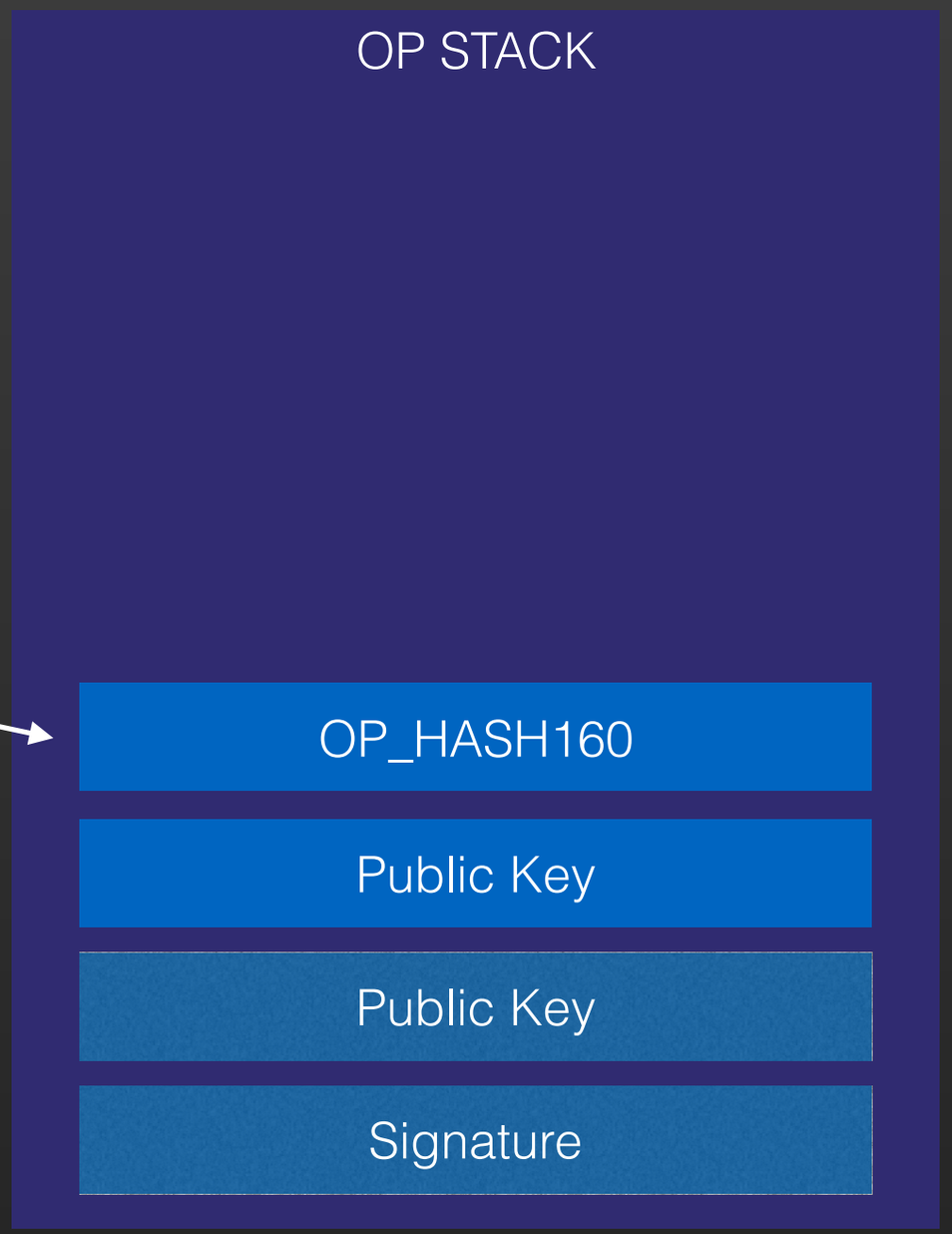


BITCOIN SCRIPTS

Output

Vout 0

```
OP_DUP OP_HASH160
c398efa9c392ba6013
OP_EQUALVERIFY
OP_CHECKSIG
```



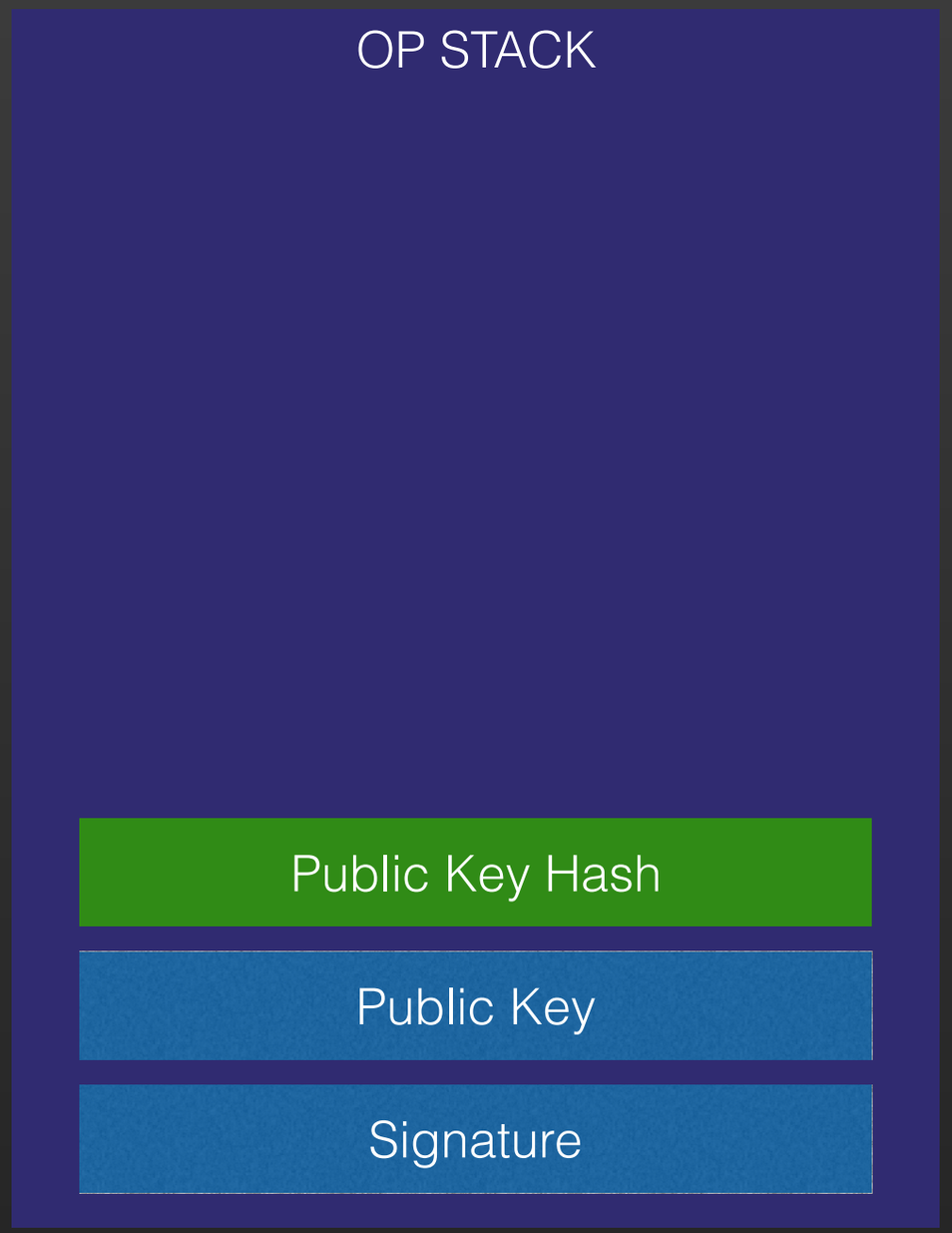
BITCOIN SCRIPTS

Output

Vout 0

```
OP_DUP OP_HASH160
c398efa9c392ba6013
OP_EQUALVERIFY
OP_CHECKSIG
```

OP_HASH160 - Perform hash160 on the top stack item



BITCOIN SCRIPTS

Output

Vout 0

```
OP_DUP OP_HASH160
c398efa9c392ba6013
OP_EQUALVERIFY
OP_CHECKSIG
```

OP STACK

c398efa9c392ba6013

Public Key Hash

Public Key

Signature

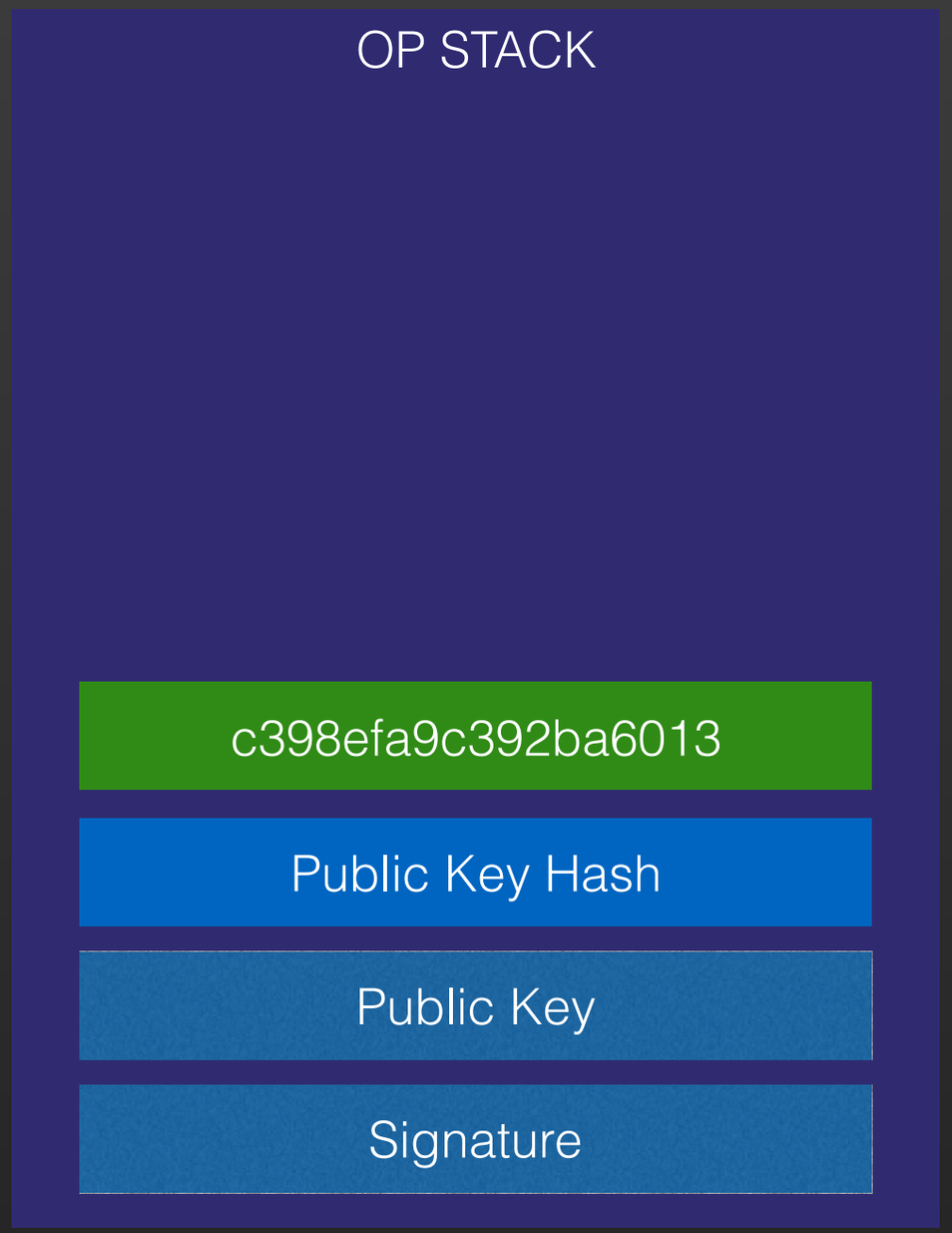
BITCOIN SCRIPTS

Output

Vout 0

```
OP_DUP OP_HASH160
c398efa9c392ba6013
OP_EQUALVERIFY
OP_CHECKSIG
```

Push a constant to the top of the stack



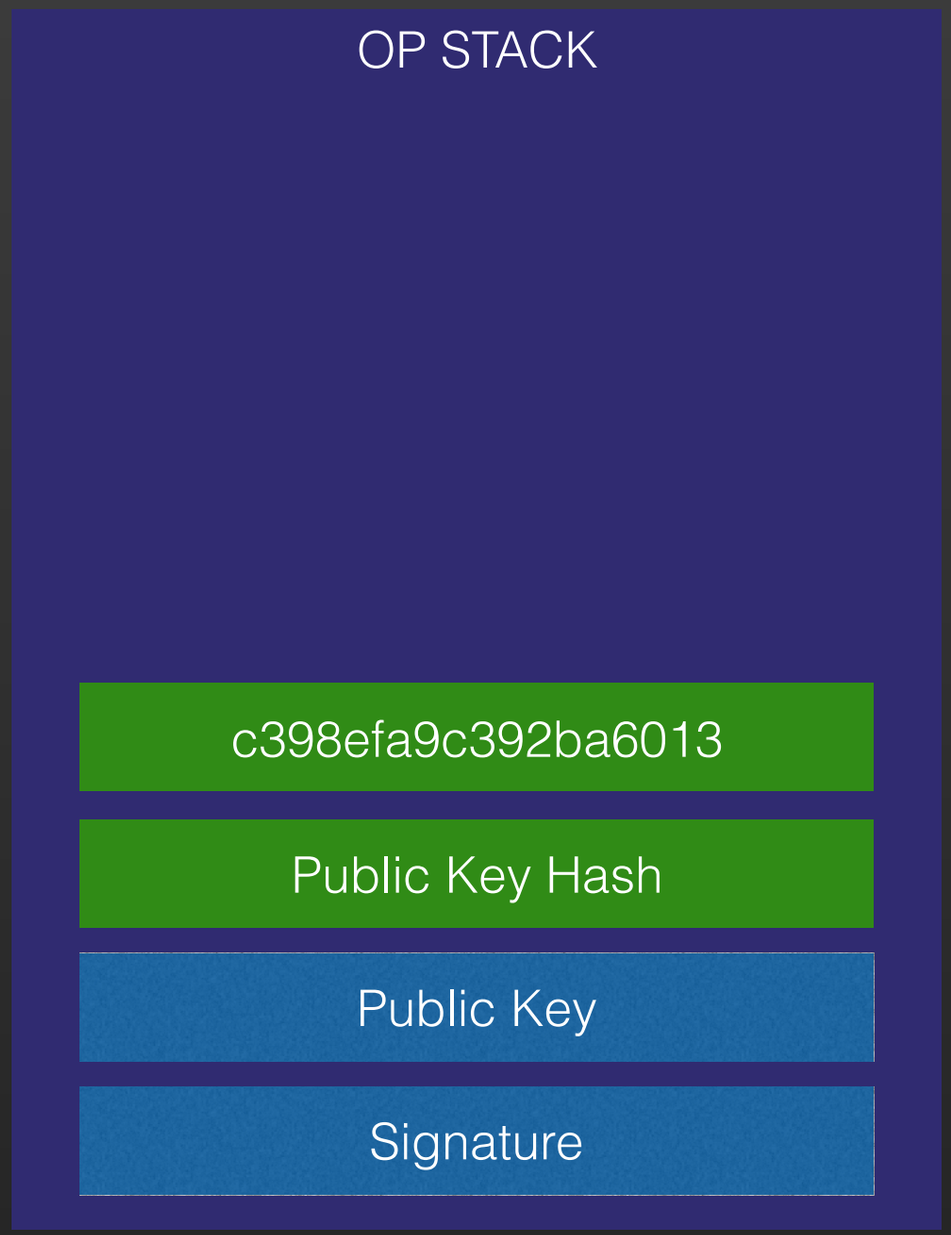
BITCOIN SCRIPTS

Output

Vout 0

```
OP_DUP OP_HASH160
c398efa9c392ba6013
OP_EQUALVERIFY
OP_CHECKSIG
```

Check if the top two stack items are equal.
If yes, pop it from the stack.



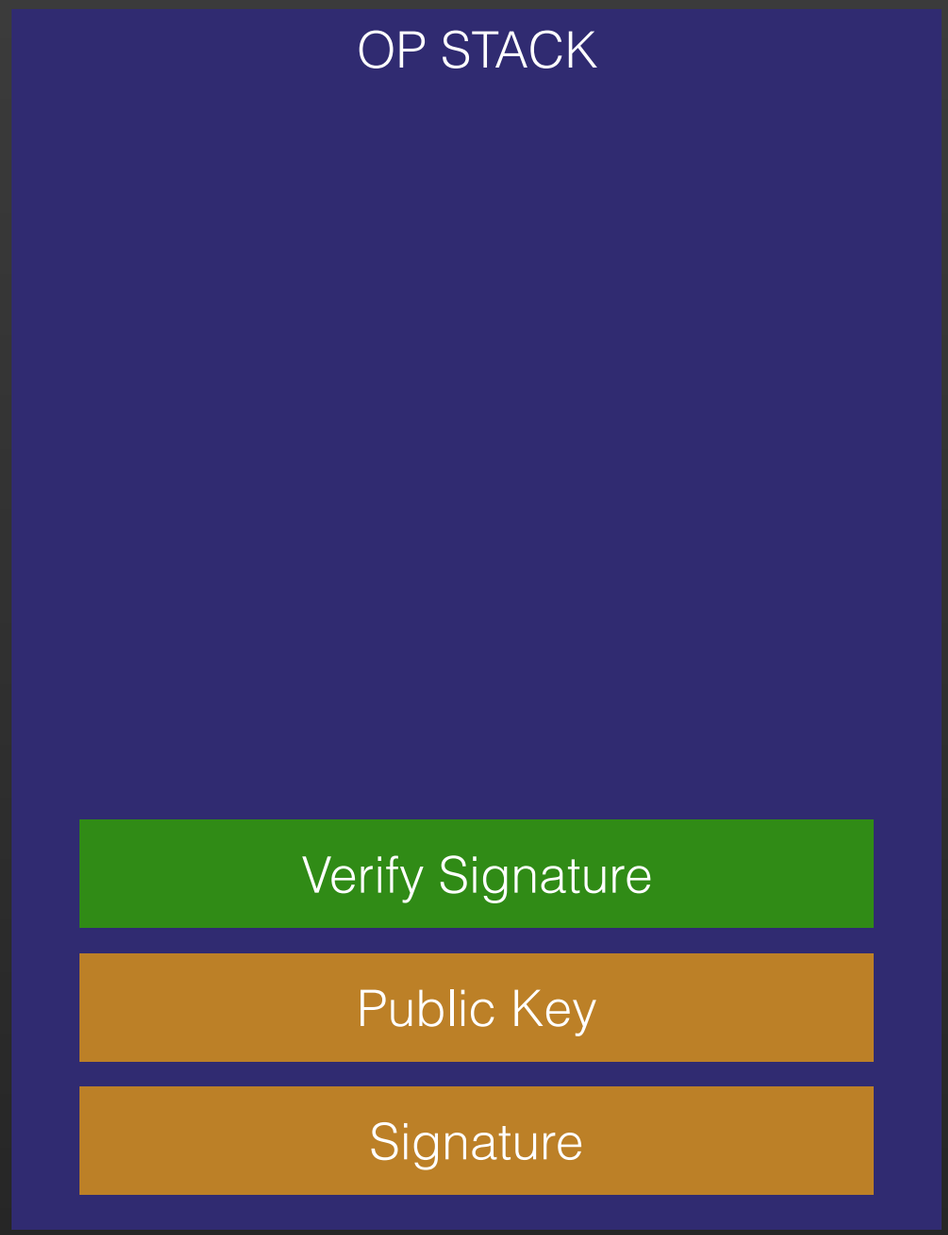
BITCOIN SCRIPTS

Output

Vout 0

```
OP_DUP OP_HASH160
c398efa9c392ba6013
OP_EQUALVERIFY
OP_CHECKSIG
```

Check if the signature is valid



BITCOIN SCRIPTS

- Many different kinds of payment contracts are possible using the scripting language
- Not meant to do much beyond specifying how payments can be redeemed and spent
- Creative use of coding language can lead to interesting contracts:
  - Refundable deposits
  - Multi-signature Escrow
  - Storing data

MULTI-SIGNATURE TECHNOLOGY IN ACTION

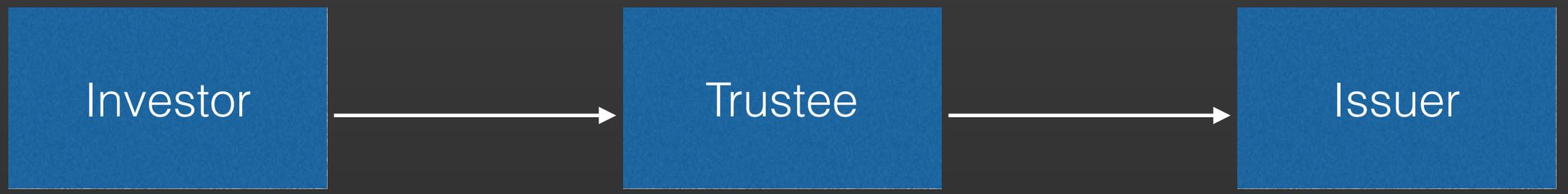
Multi-signature technology can be used to automate business processes.
Here we show a 2-of-3 multi-signature process in a crowd funding model.

Investor

Trustee

Issuer

MULTI-SIGNATURE TECHNOLOGY IN ACTION



Investor contacts trustee
to commit funds to an issuer

Operator sets up multi-signature
process with both parties

Funds are “locked” in a special payment contract that can be “unlocked” with 2-of-3 signatures

MULTI-SIGNATURE TECHNOLOGY IN ACTION

Funding success! Time to release funds from investor to issuer

Investor

Trustee

Issuer

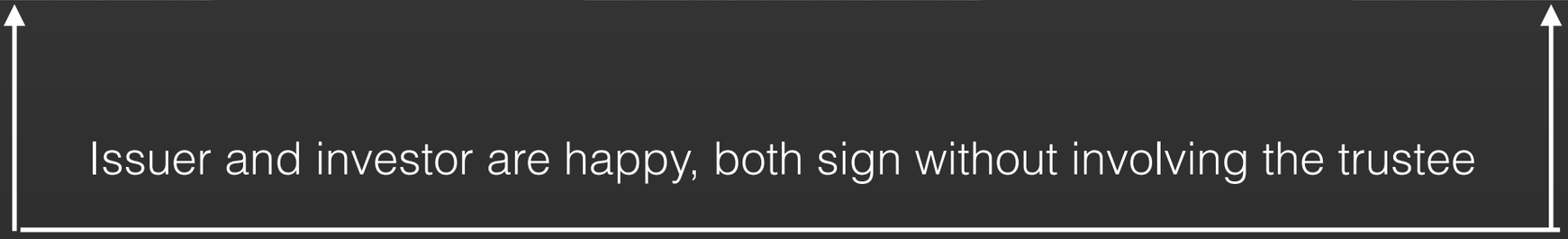
MULTI-SIGNATURE TECHNOLOGY IN ACTION

Funding success! Time to release funds from investor to issuer

Investor

Trustee

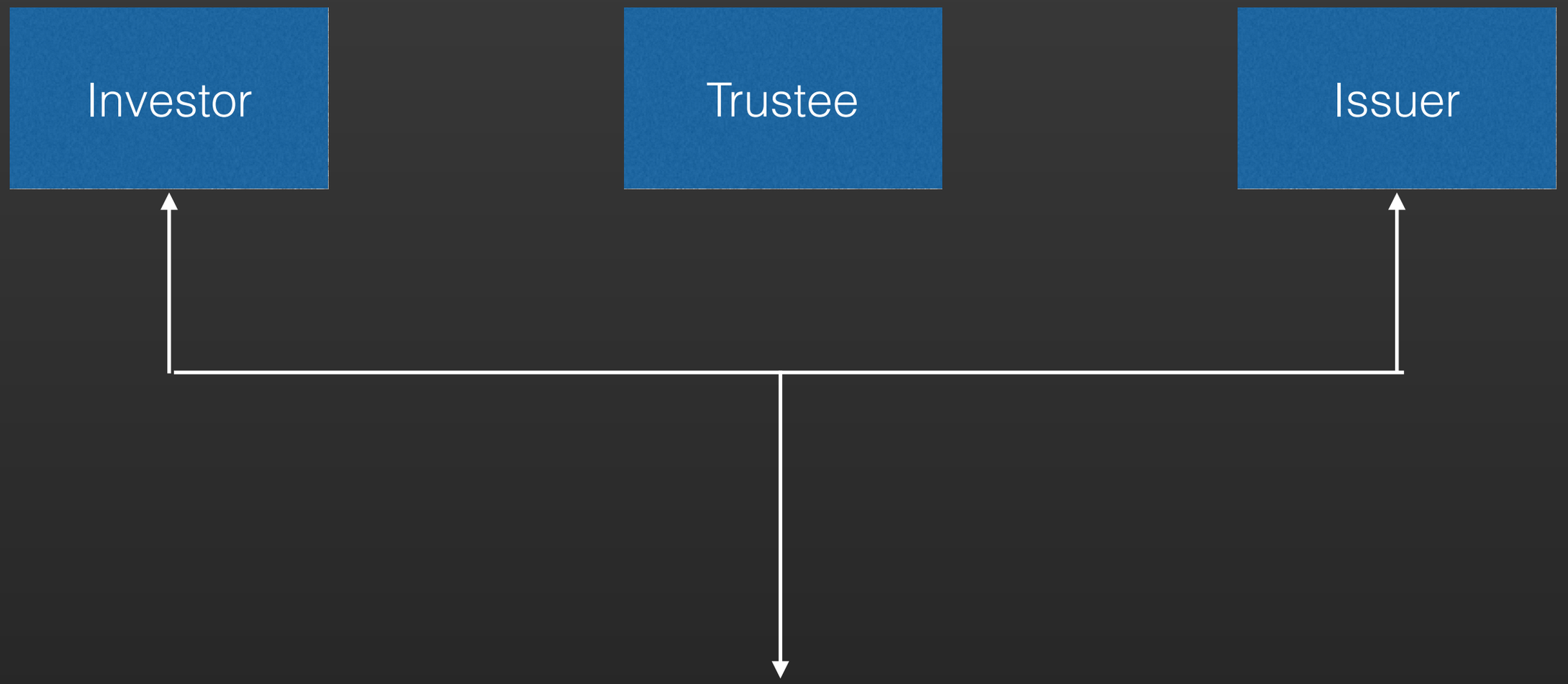
Issuer



Issuer and investor are happy, both sign without involving the trustee

MULTI-SIGNATURE TECHNOLOGY IN ACTION

Funding success! Time to release funds from investor to issuer



Coins are “unlocked” and paid to issuer.

MULTI-SIGNATURE TECHNOLOGY IN ACTION

Funding failed! Time to return the funds to the investor

Investor

Trustee

Issuer

MULTI-SIGNATURE TECHNOLOGY IN ACTION

Funding failed! Time to return the funds to the investor

Investor

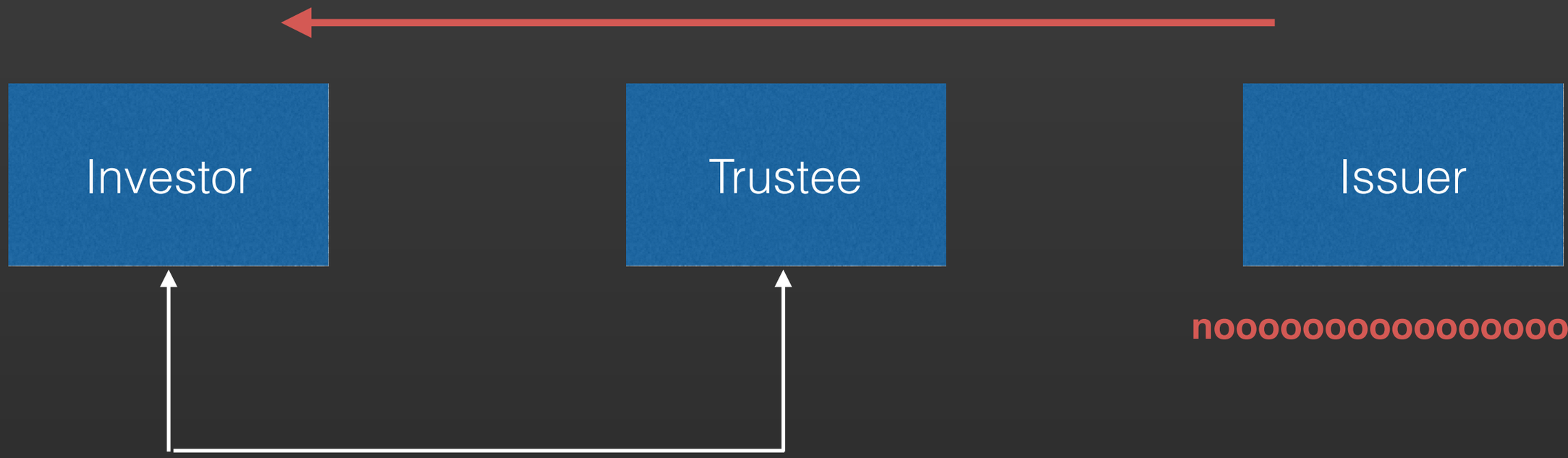
Trustee

Issuer

OH NO, Issuer refuse to sign because they suspect foul play!

MULTI-SIGNATURE TECHNOLOGY IN ACTION

Funding failed! Time to return the funds to the investor



nooooooooooooooooo

Trustee and investor signs anyways, following protocol and enforcing the contract.

USES OF MULTI-SIGNATURE SIGNING

Use cases

- Corporate procurement protocols (2-of-3 signature)
  - Purchase has a key
  - CEO has a key
  - Finance has a key
  - Purchaser + Finance sign for most procurements
  - Purchaser + CEO sign for big procurements

- Voting systems for AGM (50% signature)
  - Special smart contract
  - 6-of-11 signatures required
  - Once at least 6 signatures are signed, transaction is committed

REFUNDABLE DEPOSITS

Tenant

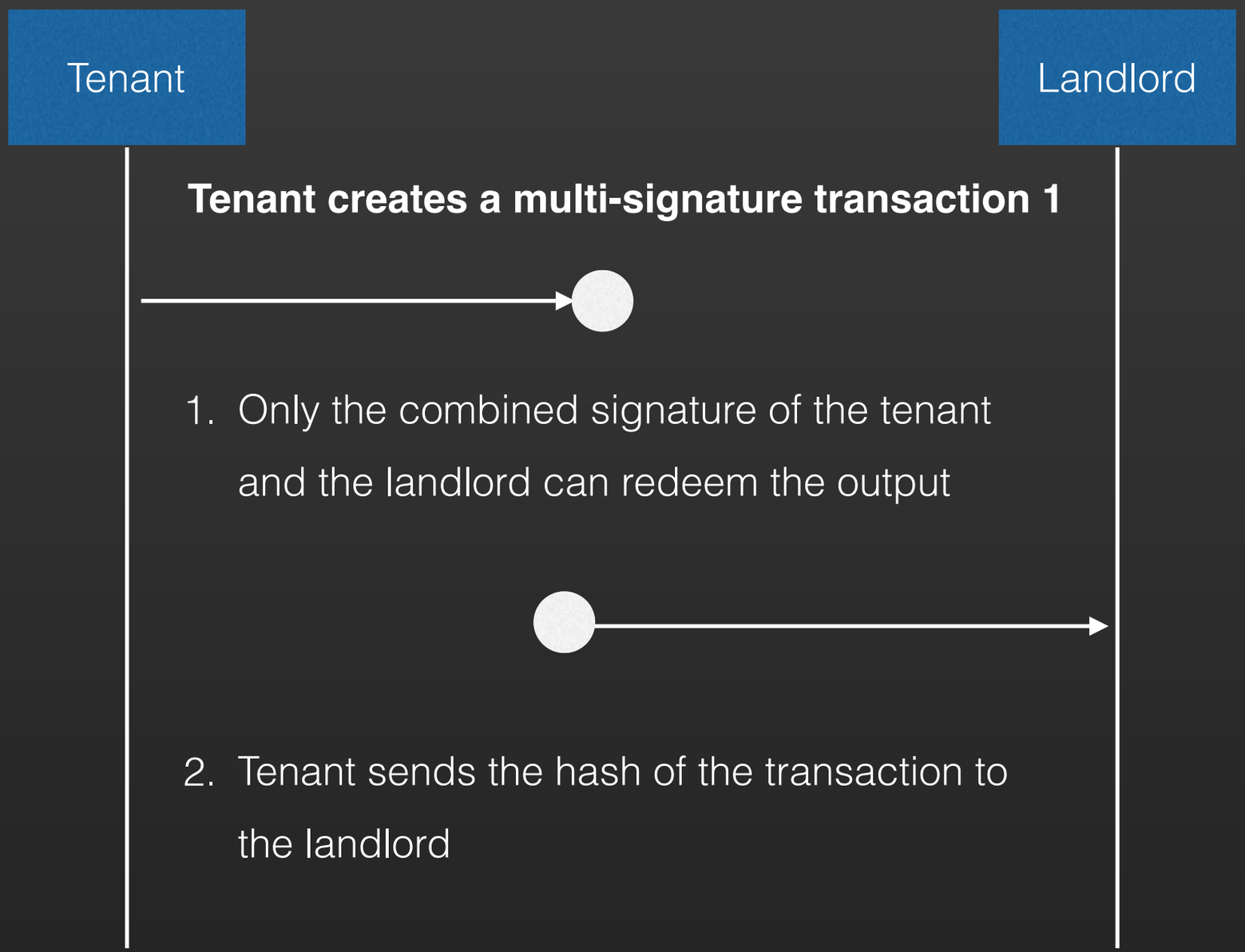
Landlord

REQUIREMENTS

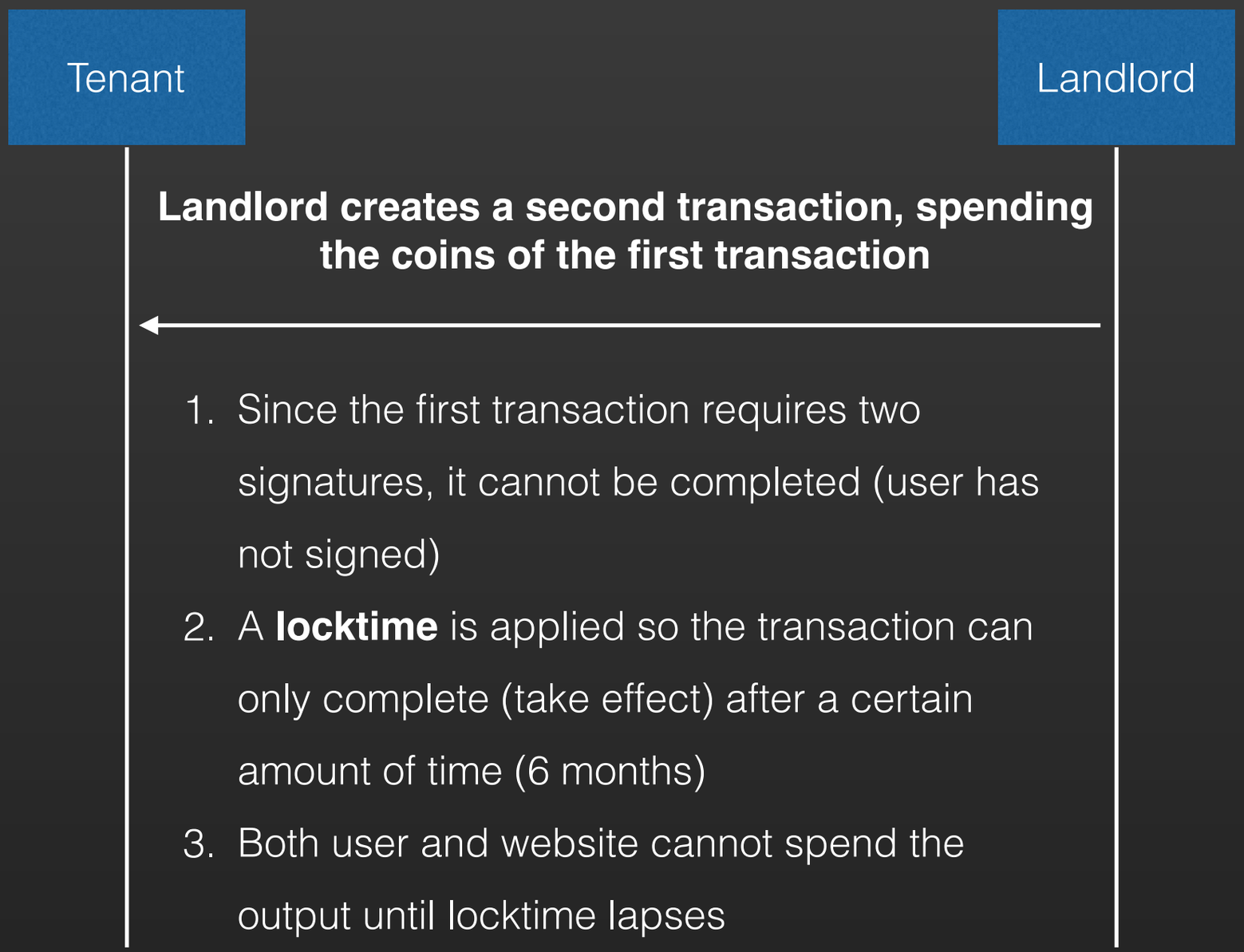
1. If you close the account, the landlord, regardless of dead or alive, must refund you the money

   The money is a proof of your reputation, and commitment
2. If you are abusive, the landlord must be able to enforce the maximum length of contract before you get your refund eventually
3. The landlord must never spend your money

REFUNDABLE DEPOSITS



REFUNDABLE DEPOSITS



REFUNDABLE DEPOSITS

Tenant

Landlord

Early shutdown of account

1. If the user wishes to be refunded earlier than the locktime, the user can request the landlord to re-sign the transaction with a shorter locktime

REFUNDABLE DEPOSITS

Tenant

Landlord

Extension of account

1. If the user wishes to extend the time of the account, the user can request the landlord to re-sign the second transaction with a longer locktime

REFUNDABLE DEPOSITS

Tenant

Landlord

Punishment of abuse

1. If the user is abusive, the landlord may not allow the user to close the account early
2. Alternatively, the landlord may sign a new transaction requesting for more bitcoins, or have longer locktime

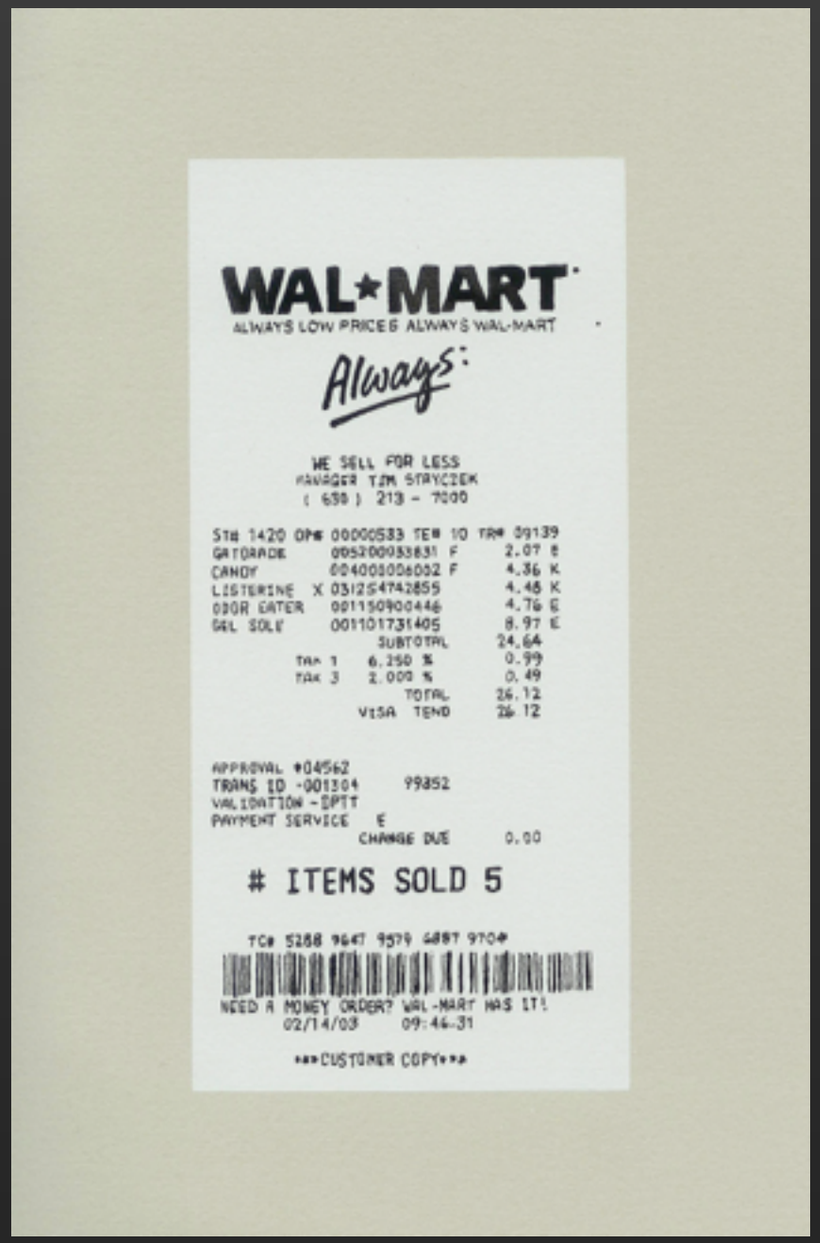
REFUNDABLE DEPOSITS

Tenant

Landlord

All these can be done via bitcoin's script

STORING DATA INTO BLOCKCHAINS



OP_RETURN

- Generic data is not meant to be stored into blockchains
- Data storage can be “forced” using a special command: OP_RETURN
- Anything after OP_RETURN can be pruned from the UTXO set
- People can embed receipts or short messages in OP_RETURN

- **What is OP_RETURN’s original purpose???**

STORING DATA INTO BLOCKCHAINS

STORING DATA INTO BLOCKCHAINS

EVERSTORE

THE DISTRIBUTED DATABASE PROTOCOL

Neuroware's propriety algorithm to store general data into blockchains

USING CRYPTOCURRENCIES

Blockchains from the user's eyes

OVERVIEW

- Wallets
- Sending/Receiving Tokens
- Crypto-exchanges
- API/JSON-RPC

WALLETS

- All blockchains can be interacted in two ways:
  - Programmatically (API calls)
  - Transaction (Wallet)

- Wallet is a generic term used to refer to the pool of private keys that only you own
  - You are 100% responsible for your private keys
  - Lose your private keys = lose your “money”

- Special terms
  - Accounts
  - Private Keys
  - Public Keys
  - Addresses

SENDING/RECEIVING TOKENS

- Participants within the same blockchain using the same cryptocurrency can send each other tokens
  - No central authority needed
  - All wallet clients must conform to protocol

- Transaction and Settlement have very little interval
  - Transaction is made from the user's own node
  - Transaction is settled when enough confirmations are achieved

- Transfers can be programmed and automated
  - Multi-signature/payment contracts/smart contracts can be used
  - Some wallets allow programming of smart contracts

SEND TOKENS TO ONE ANOTHER

Get the economy working!

TYPES OF WALLETS

Light Wallet

Full Wallet

Hosted Wallet

TYPES OF WALLETS

Light Wallet

- Only stores a limited snapshot of data relevant to the user

- Can easily be embedded on limited devices

- Does not contribute to the ecosystem's integrity

Full Wallet

Hosted Wallet

TYPES OF WALLETS

Light Wallet

Full Wallet

- Contains the full node of the blockchain, all transactions are stored
- Helps provide redundancy to the ecosystem
- Requires a lot of space and hardware resources down the line

Hosted Wallet

TYPES OF WALLETS

Light Wallet

Full Wallet

Hosted Wallet

- Wallet with the node hosted by a third-party
- Requires user to trust the host of the node to play nice
- Insecure, and risky at times, but can often be the only feasible solution

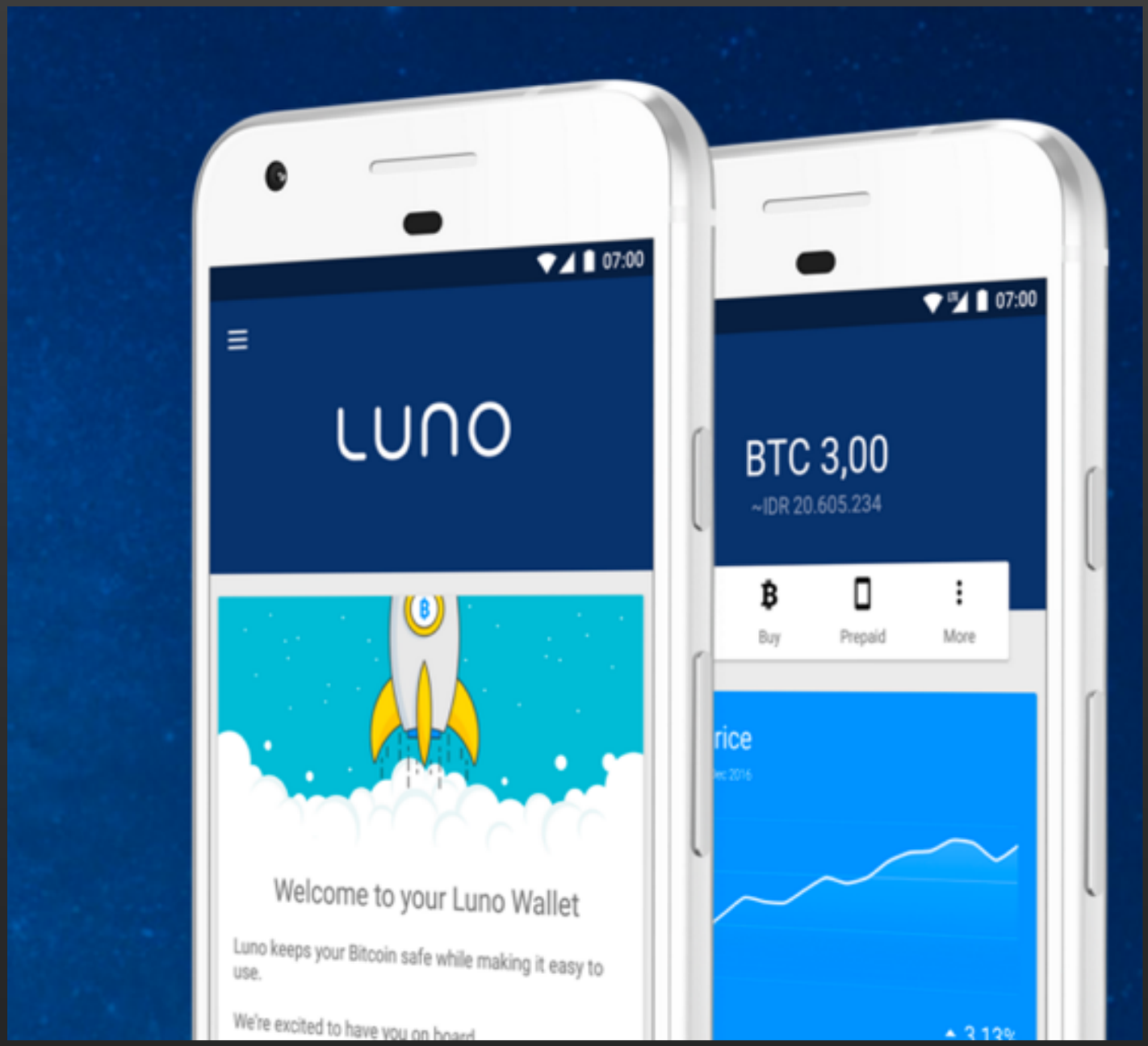
CRYPTO-EXCHANGES

- Only way to gain participation into a blockchain is through crypto-exchanges

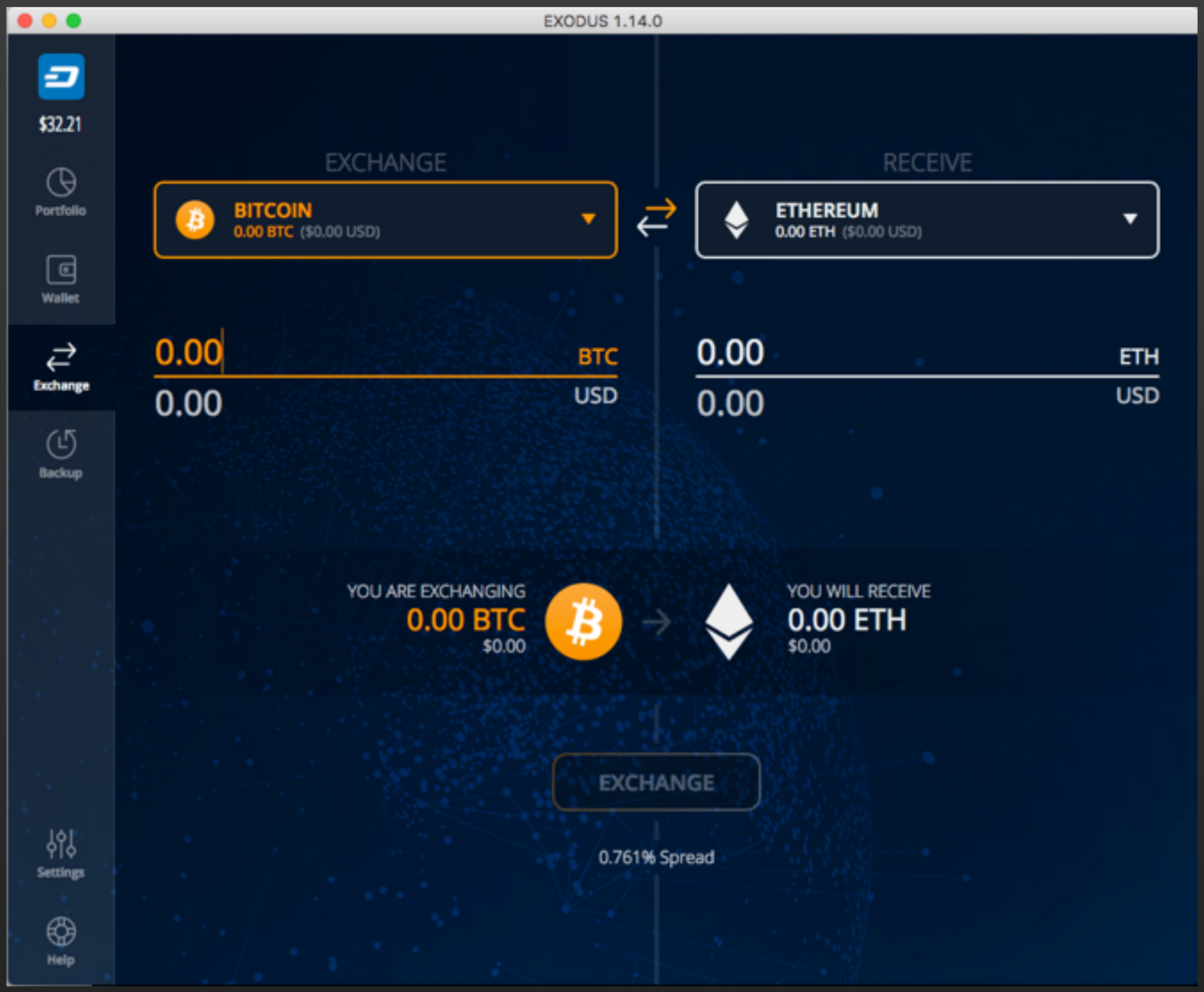
- Steps:
  - Login to an exchange
  - Select amount of cryptocurrency to purchase
  - Pay exchange via credit card/debit card/real money
  - Trust exchange to send you cryptocurrency
  - …
  - Profit?

- Crypto-exchanges are notorious for being vulnerable to hacks
  - Functions the same as banks/stock exchanges
  - Run without regulations

CRYPTO-EXCHANGES



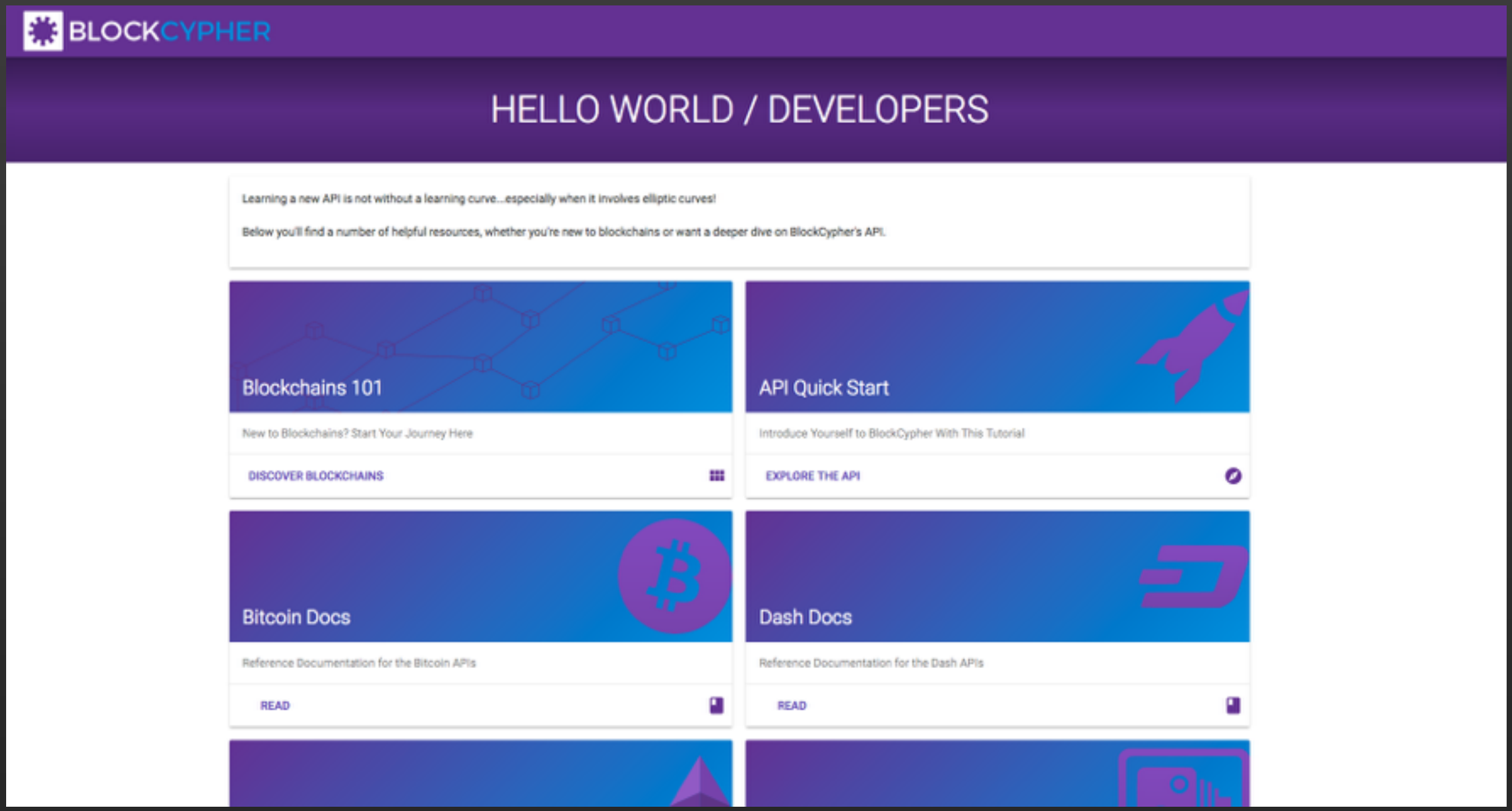
CRYPTO-EXCHANGES



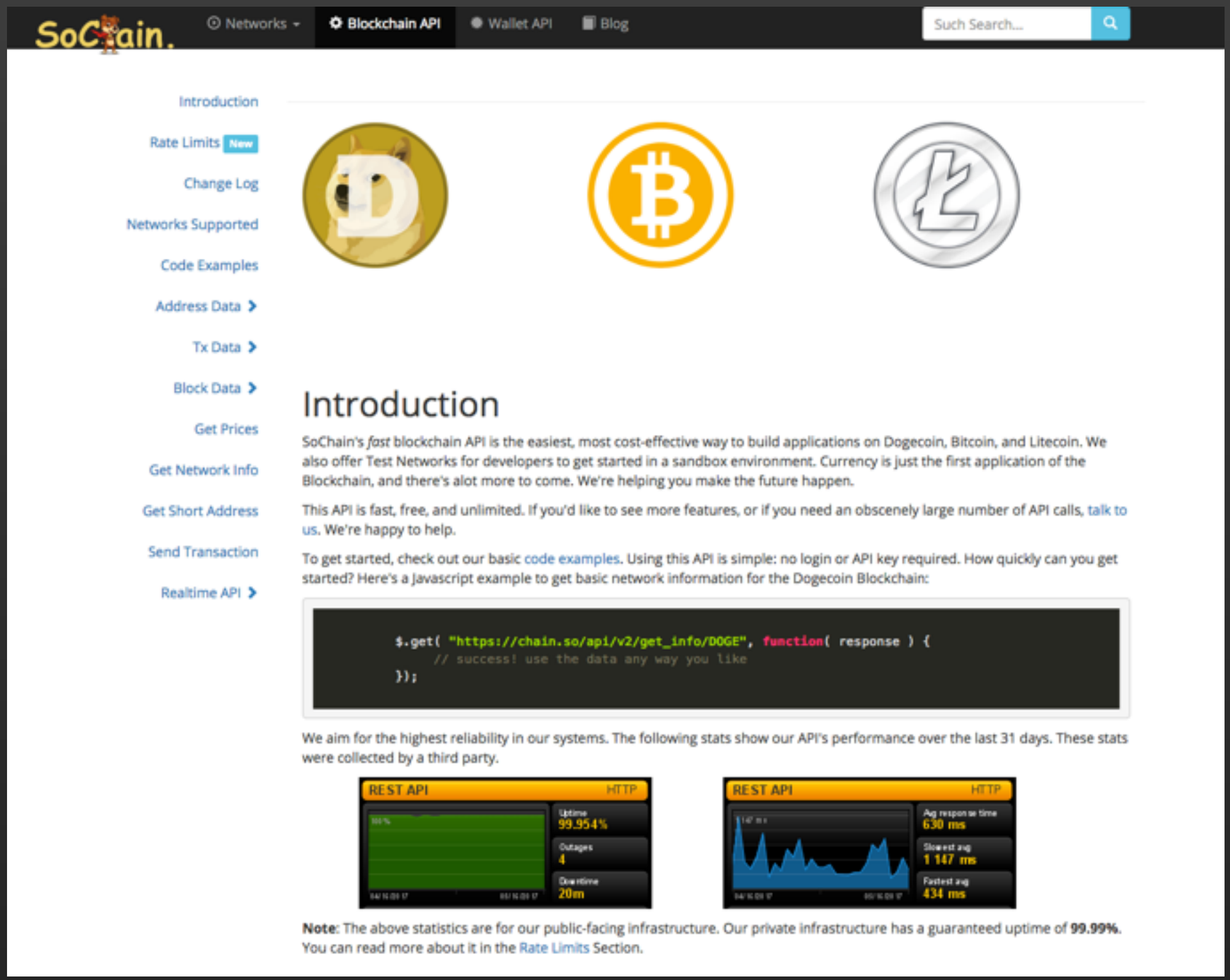
JSON-RPC

- Bitcoin (and most altcoins) have a JSON-RPC endpoint
- Can be accessed through the command line
- Can be used to query the blockchain

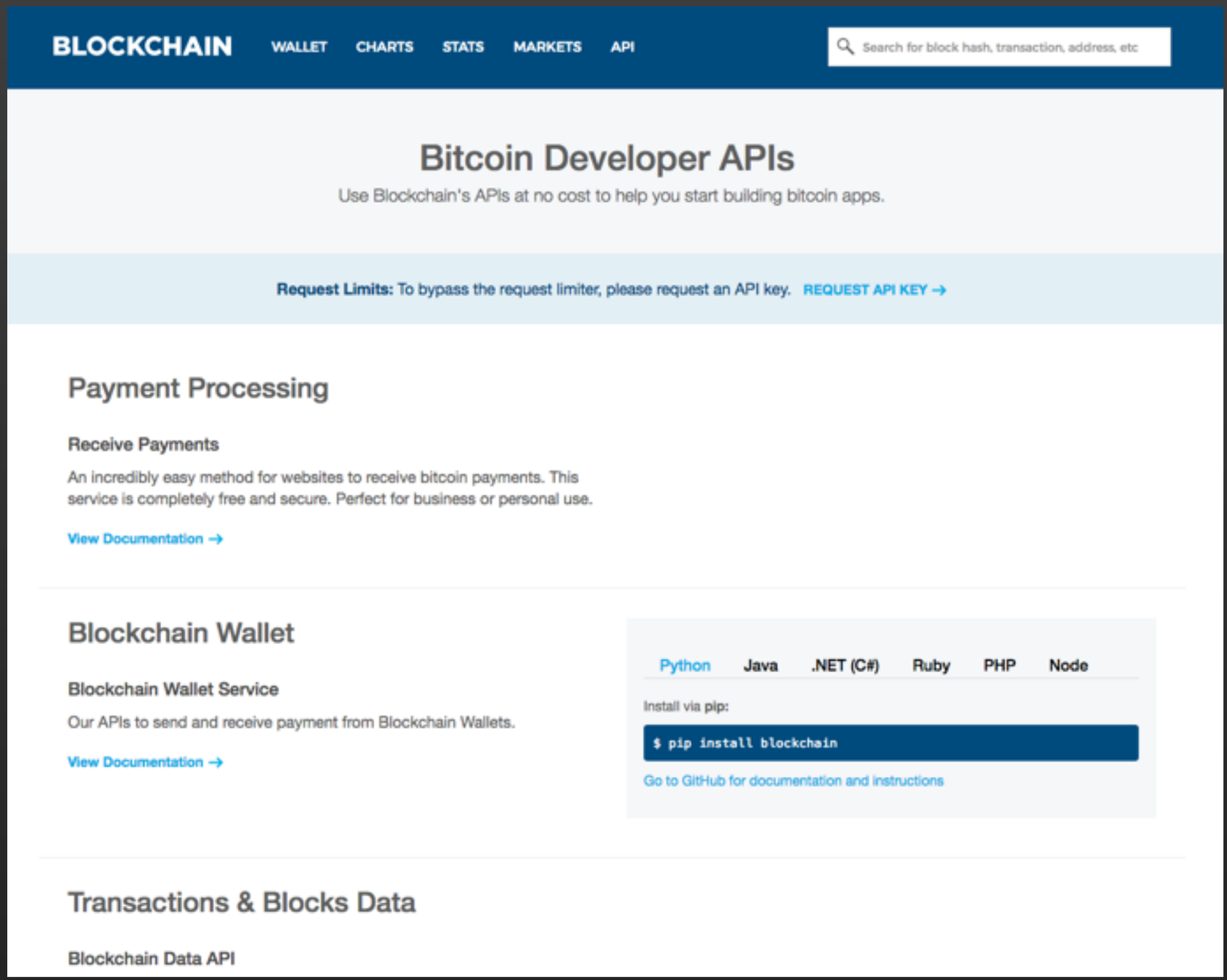
BLOCKCYPHER.COM



CHAIN.SO


SoChain.
Networks
Blockchain API
Wallet API
Blog
Such Search...
Introduction
Rate Limits New
Change Log
Networks Supported
Code Examples
Address Data
Tx Data
Block Data
Get Prices
Get Network Info
Get Short Address
Send Transaction
Realtime API
Introduction
SoChain's *fast* blockchain API is the easiest, most cost-effective way to build applications on Dogecoin, Bitcoin, and Litecoin. We also offer Test Networks for developers to get started in a sandbox environment. Currency is just the first application of the Blockchain, and there's alot more to come. We're helping you make the future happen.
This API is fast, free, and unlimited. If you'd like to see more features, or if you need an obscenely large number of API calls, talk to us. We're happy to help.
To get started, check out our basic code examples. Using this API is simple: no login or API key required. How quickly can you get started? Here's a Javascript example to get basic network information for the Dogecoin Blockchain:
$.get("https://chain.so/api/v2/get_info/DOGE", function(response) {
// success! use the data any way you like
});
We aim for the highest reliability in our systems. The following stats show our API's performance over the last 31 days. These stats were collected by a third party.
REST API
HTTP
Uptime
99.954%
Outages
4
Downtime
20m
REST API
HTTP
Avg response time
630 ms
Slowest avg
1 147 ms
Fastest avg
434 ms
Note: The above statistics are for our public-facing infrastructure. Our private infrastructure has a guaranteed uptime of 99.99%. You can read more about it in the Rate Limits Section.

BLOCKCHAIN.INFO



SEGMENT 4 - BREAK TIME

QUESTION & ANSWER SESSION

15 mins

THANK YOU

LEARN MORE ABOUT

neuroware

http://neuroware.io