Developing a Cryptocurrency

NAV

NAVCOIN

About me

- Alex Vazquez
- Core Developer of NAV Coin (C++)
- http://www.github.com/navcoin/navcoin-core
- alex@encrypt-s.com

Resources

- Bitcoin Whitepaper: https://bitcoin.org/bitcoin.pdf
- Jameson Lopp: https://lopp.net/bitcoin.html
- Andreas Antonopoulos books:
  - The Internet of Money (ISBN: 978-1537000459)
  - Mastering Bitcoin (ISBN: 978-1491954386)
    - Free @ https://github.com/bitcoinbook/bitcoinbook

Structure

- Brief description of how a cryptocurrency works:
  - What is a Blockchain. Distributed Model of Database. Consensus algorithm.
  - Architecture of a Blockchain. Bitcoin Scripting.
- What is NAV Coin:
  - Key differences with Bitcoin.
- Implementing new features: The Community Fund.
- Demo.
- Building and version control:
  - Deterministic build system.
  - Git.
- Questions round.

Traditional money vs. cryptocurrencies

Traditional money:

- Money is not supported by gold anymore. Authorities can create money from thin air!
- Banks are open to manipulation.
- Inflationary economy. Extreme devaluation of currency.
- Lack of transparency.
- Expensive fees.

Why do we still use it?

Paper money was a recent invention where there's trust between two parties which accept some outside force (banks, the government) will underwrite the value of the paper.

Cryptocurrency 101

Bank → Centralized database

Cryptocurrency → Internet Protocol for a decentralized database.

- Members of a group which agree on a set of rules create a P2P network.
- Permission-less: you don't need to sign up or receive approval to take part on the network.
  - Nodes can freely join and leave. Users can freely create pair of keys and submit transactions.
- Decentralized: there's no central part which can change the rules. Rules are determined by consensus between the nodes.
  - A consensus algorithm (Proof of Stake for Navcoin, Proof of Work for Bitcoin) ensures the fulfillment of the rules while majority of the network is honest.
- As long as you trust the code and cryptography you can trust the technology behind a cryptocurrency!

Cryptocurrency 101

Asymmetric Cryptography: Private/Public Key

Funds are secured using Asymmetric ECC (Elliptic Curve Cryptography) based on the algebraic structure of elliptic curves over finite fields.

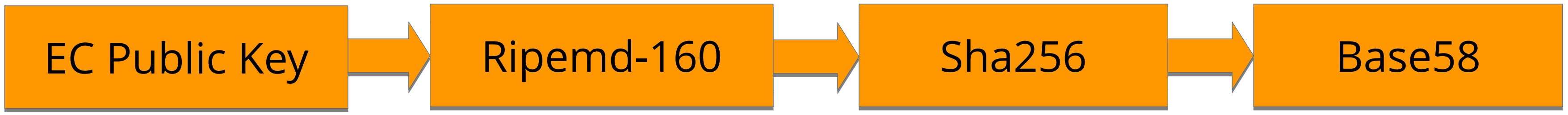
“Public-key cryptography is based on the intractability of certain mathematical problems. Early public-key systems are secure assuming that it is difficult to factor a large integer composed of two or more large prime factors. For elliptic-curve-based protocols, it is assumed that finding the discrete logarithm of a random elliptic curve element with respect to a publicly known base point is infeasible: this is the "elliptic curve discrete logarithm problem" (ECDLP). The security of elliptic curve cryptography depends on the ability to compute a point multiplication and the inability to compute the multiplicand given the original and product points. The size of the elliptic curve determines the difficulty of the problem.”

Source: https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Cryptocurrency 101

Address

Coins can be received in/sent to an address. An address is the base58 representation of the RIPEMD160(SHA256()) Hash of a public key and is publicly shared.



- It is possible to get at any time the Base58 Address of a Public Key. Nonetheless this process is not reversible!

  This is considered an extra layer of security:

  Length of a hash: 256 bits – 2^256 = 1,157920892E77 different possibilities.

  - (Number of atoms in the universe between 4E79 and 4E81).

  It would take a long time to brute force it ;-) (without quantum computing :-P)

  Importance of keeping safe your Private Key!

- the Public Address is derived mathematically from the Private Key
- it is necessary for controlling any funds associated with the Public Address

Cryptocurrency 101

Two primitive classes: Transactions and blocks.

A transaction is a transfer of value between users. Transactions are collected in blocks.

The first transactions of a block (called coinbase in POW protocols, coinstake in POS) are special transactions which create new coins instead of transfering them. The amount of generated coins is deterministic!

Block 1	Block 2	Block 3
Coinstake/coinbase	Coinstake/coinbase	Coinstake/coinbase
Transaction 1	Transaction 1	Transaction 1
Transaction 2	Transaction 2	
	Transaction 3	

Both the spacing between blocks and the inflationary policy (how new coins are created) of a cryptocurrency depends on the consensus rules of the network.

Cryptocurrency 101

Structure of a block.

From src/primitives/block.h:

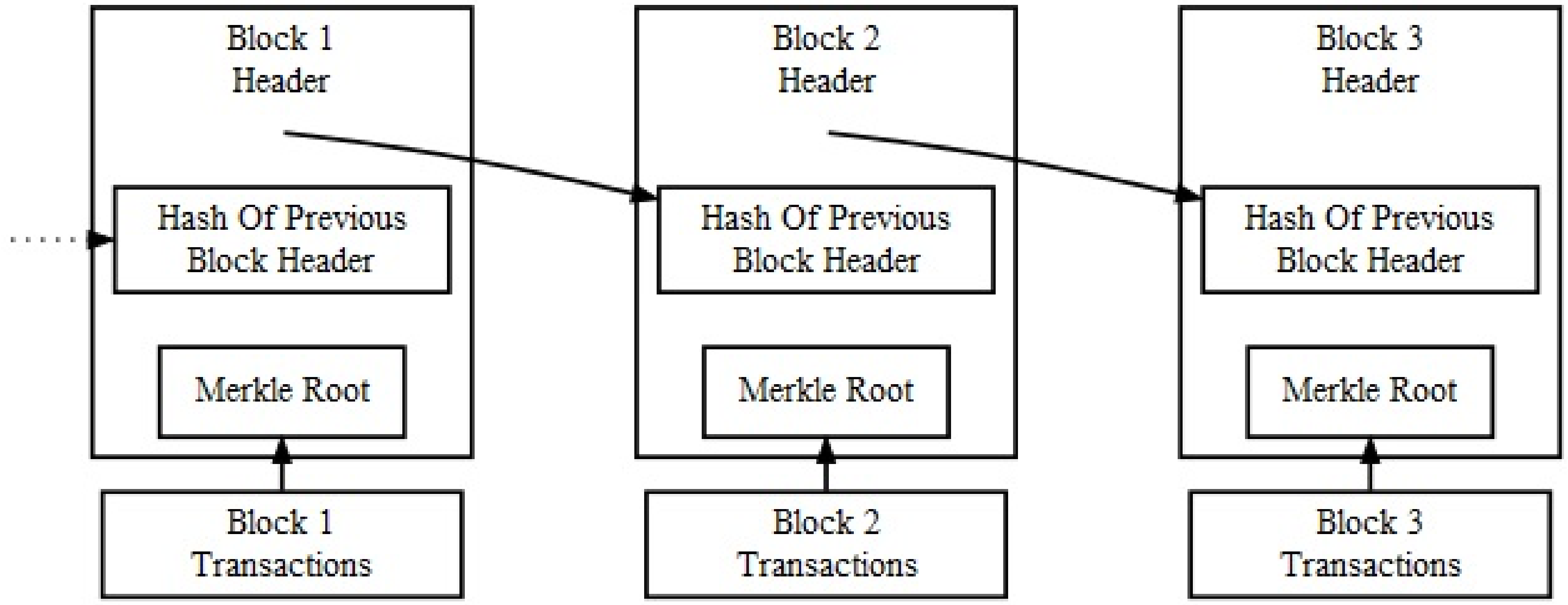
```
class CBlockHeader
{
public:
    // header
    int32_t nVersion;
    uint256 hashPrevBlock;
    uint256 hashMerkleRoot;
    uint32_t nTime;
    uint32_t nBits;
    uint32_t nNonce;
.......
```

```
class CBlock : public CBlockHeader
{
public:
    // network and disk
    std::vector<CTransaction> vtx;
    // navcoin: block signature
    std::vector<unsigned char> vchBlockSig;

...........
```

See blob_base and uint256
class definitions
at src/uint256h !

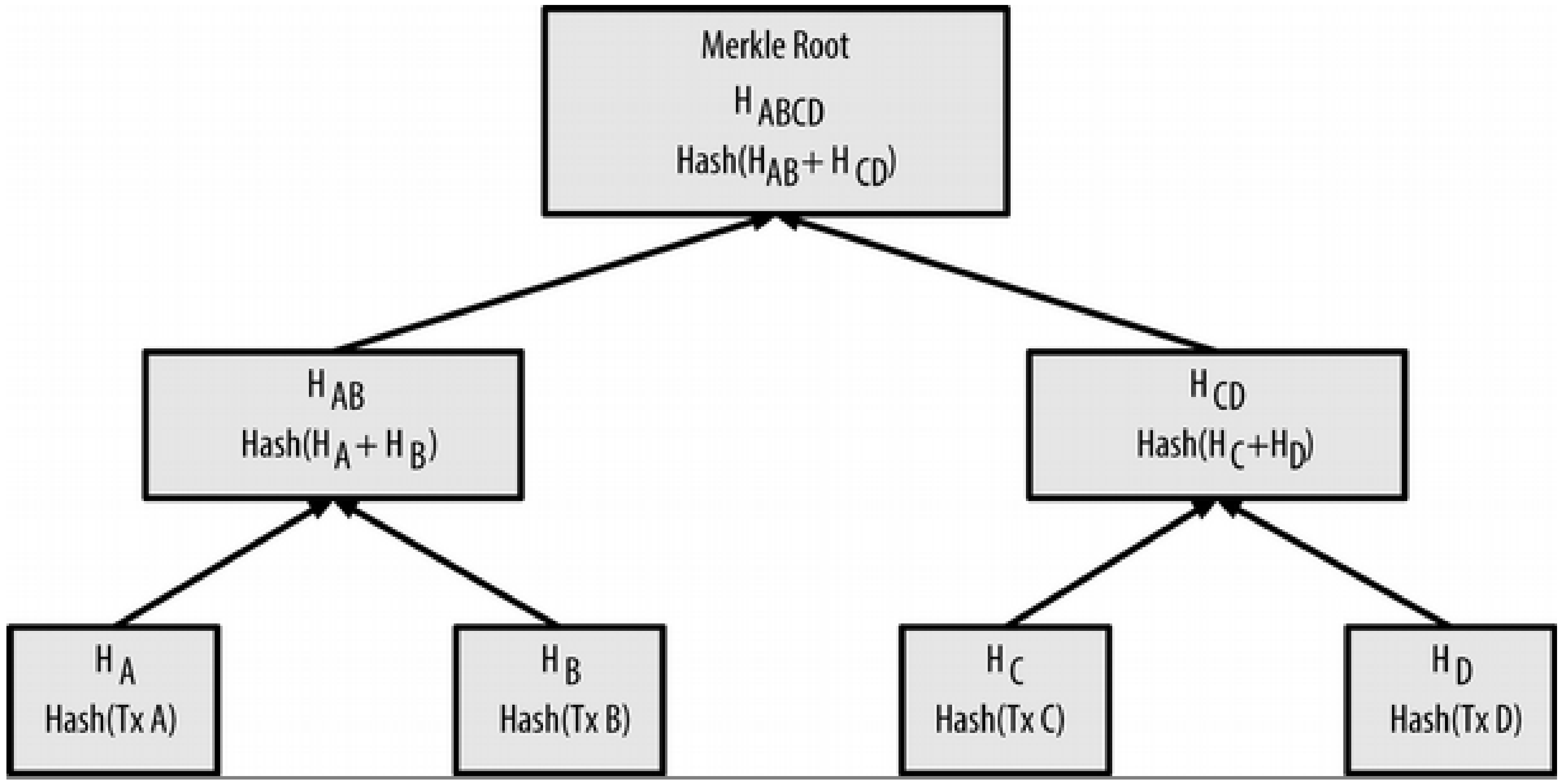
Block chain



Simplified Bitcoin Block Chain

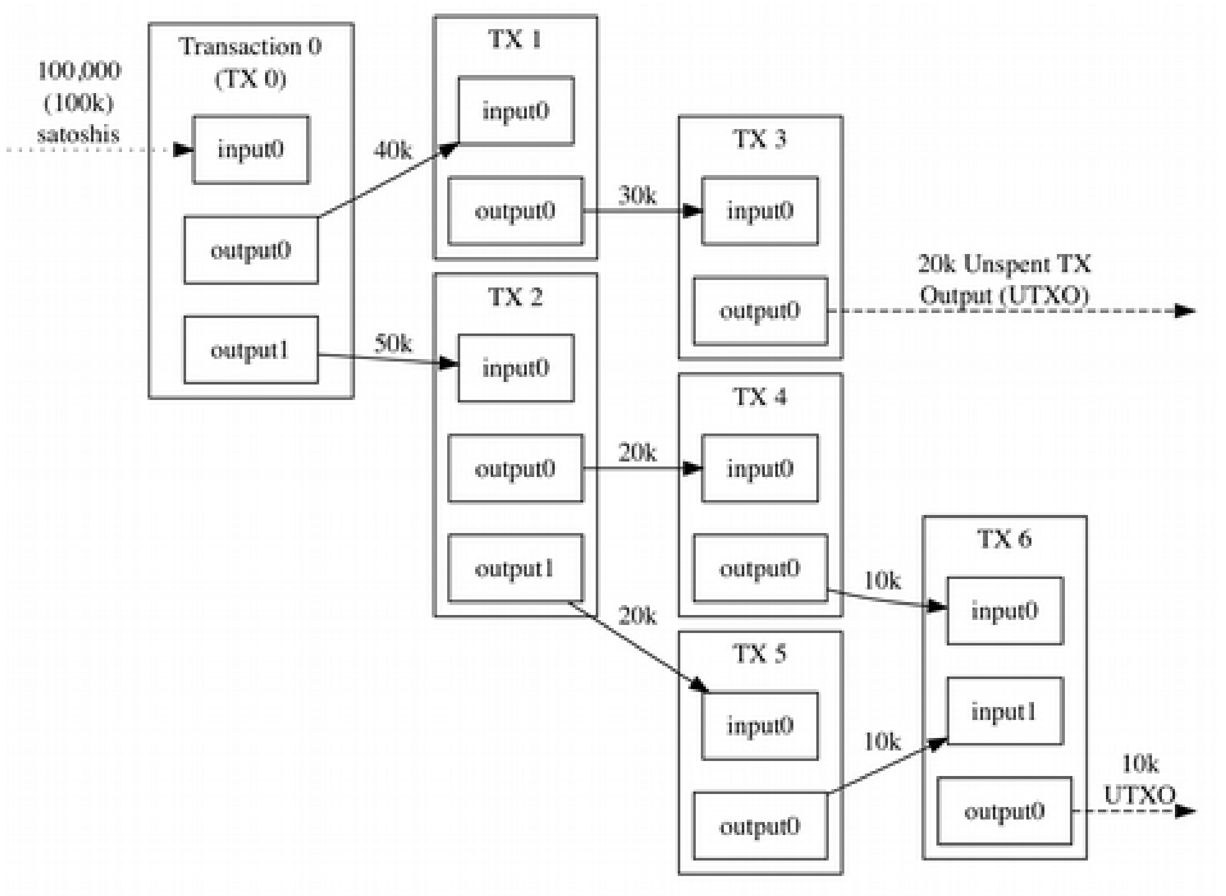
Source: bitcoin.org

Transaction Merkle Root Hash



Source: bitcoin.org

Structure of a transaction



Triple-Entry Bookkeeping (Transaction-To-Transaction Payments) As Used By Bitcoin

Source bitcoin.org

Structure of a transaction.

From src/primitives/transaction.h:

```
/** The basic transaction that is broadcasted on the network and contained in
* blocks.  A transaction can contain multiple inputs and outputs.
*/
class CTransaction
{
public:
    const int32_t nVersion;
    unsigned int nTime;
    std::vector<CTxIn> vin;
    std::vector<CTxOut> vout;
    const uint32_t nLockTime;
    std::string strDZeel;
...
```

Structure of a transaction

A transaction spends ‘inputs’ (vector vin) and send coins to the ‘outputs’ (vector vout).

From src/primitives/transaction.h:

```
/** An input of a transaction.  It contains the location of the previous
 * transaction's output that it claims and a signature that matches the
 * output's public key.
 */
class CTxIn
{
public:
    COutPoint prevout;
    CScript scriptSig;
...
```

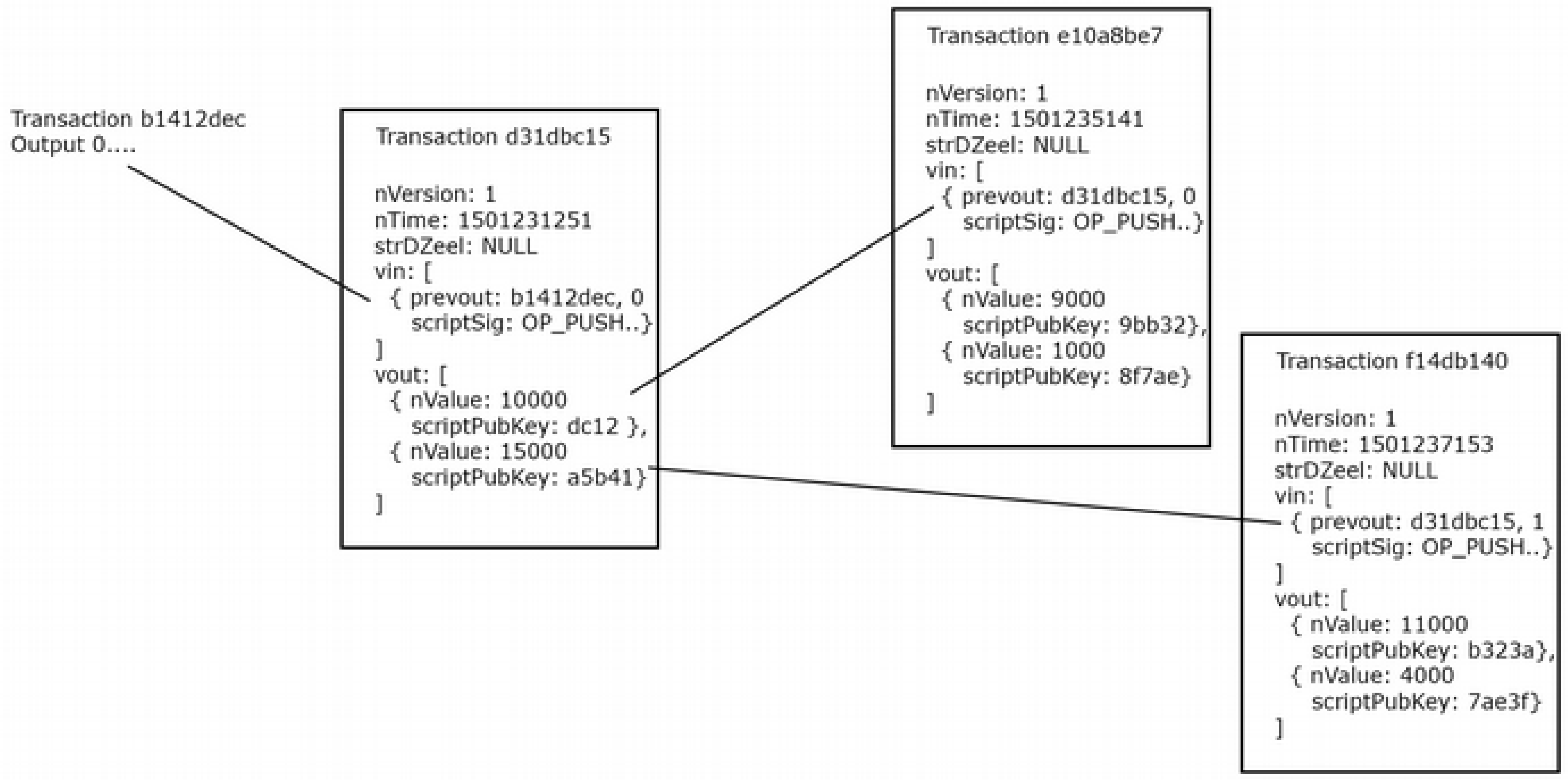
Structure of a transaction

Outputs can contain many different recipients, one of them usually sends the change back to the sender.

From src/primitives/transaction.h:

```
/** An output of a transaction.  It contains the public key that the next input
 * must be able to sign with to claim it.
 */
class CTxOut
{
public:
    CAmount nValue;
    CScript scriptPubKey;
...
```

Structure of a transaction



```
Transaction b1412dec
Output 0....
```

```
Transaction d31dbc15

nVersion: 1
nTime: 1501231251
strDZeel: NULL
vin: [
  { prevout: b1412dec, 0
    scriptSig: OP_PUSH..}
]
vout: [
  { nValue: 10000
    scriptPubKey: dc12 },
  { nValue: 15000
    scriptPubKey: a5b41}
]
```

```
Transaction e10a8be7

nVersion: 1
nTime: 1501235141
strDZeel: NULL
vin: [
  { prevout: d31dbc15, 0
    scriptSig: OP_PUSH..}
]
vout: [
  { nValue: 9000
    scriptPubKey: 9bb32},
  { nValue: 1000
    scriptPubKey: 8f7ae}
]
```

```
Transaction f14db140

nVersion: 1
nTime: 1501237153
strDZeel: NULL
vin: [
  { prevout: d31dbc15, 1
    scriptSig: OP_PUSH..}
]
vout: [
  { nValue: 11000
    scriptPubKey: b323a},
  { nValue: 4000
    scriptPubKey: 7ae3f}
]
```

Bitcoin Script language

Bitcoin and cryptocurrencies based on its codebase use an scripting language to lock and unlock coins. Some important facts about it:

- Script does not loop. Not Touring Complete!
- Script always terminates.
- Script memory access is stack-based. Items you push become operands.
- At the end of the script, the top stack item is the return value.

- Examples of a stack-based script:

OP_PUSH 2 OP_PUSH 2 OP_SUM OP_PUSH 4 OP_EQUAL

2	2	4	4	true
	2		4	

OP_PUSH 2 OP_PUSH 5 OP_SUM OP_PUSH 3 OP_EQUAL

2	5	9	3	false
	2		9	

Spending outputs using Script

Transaction outputs include an incomplete Script on the property scriptPubKey.

When another transaction includes an input which tries to spend an output, it should provide the missing part of the script on the property scriptSig.

The combined script must return true for the transaction to be accepted.

Very simple example

Transaction 1

Output 1

  scriptPubKey: OP_PUSH 2 OP_SUM OP_PUSH 4 OP_EQUAL

  nValue: 1,000

Transaction 2

Input 1

  prevout: Transaction 1, Out 1

  scriptSig: OP_PUSH 2

ACCEPTED

Transaction2.scriptSig + Transaction1.scriptPubKey

returns TRUE

Transaction 3

Input 1

  prevout: Transaction 1, Out 1

  scriptSig: OP_PUSH 5

REJECTED

Transaction3.scriptSig + Transaction1.scriptPubKey
returns FALSE

A typical locking script IRL

- P2PKH: Pay to Public Key Hash

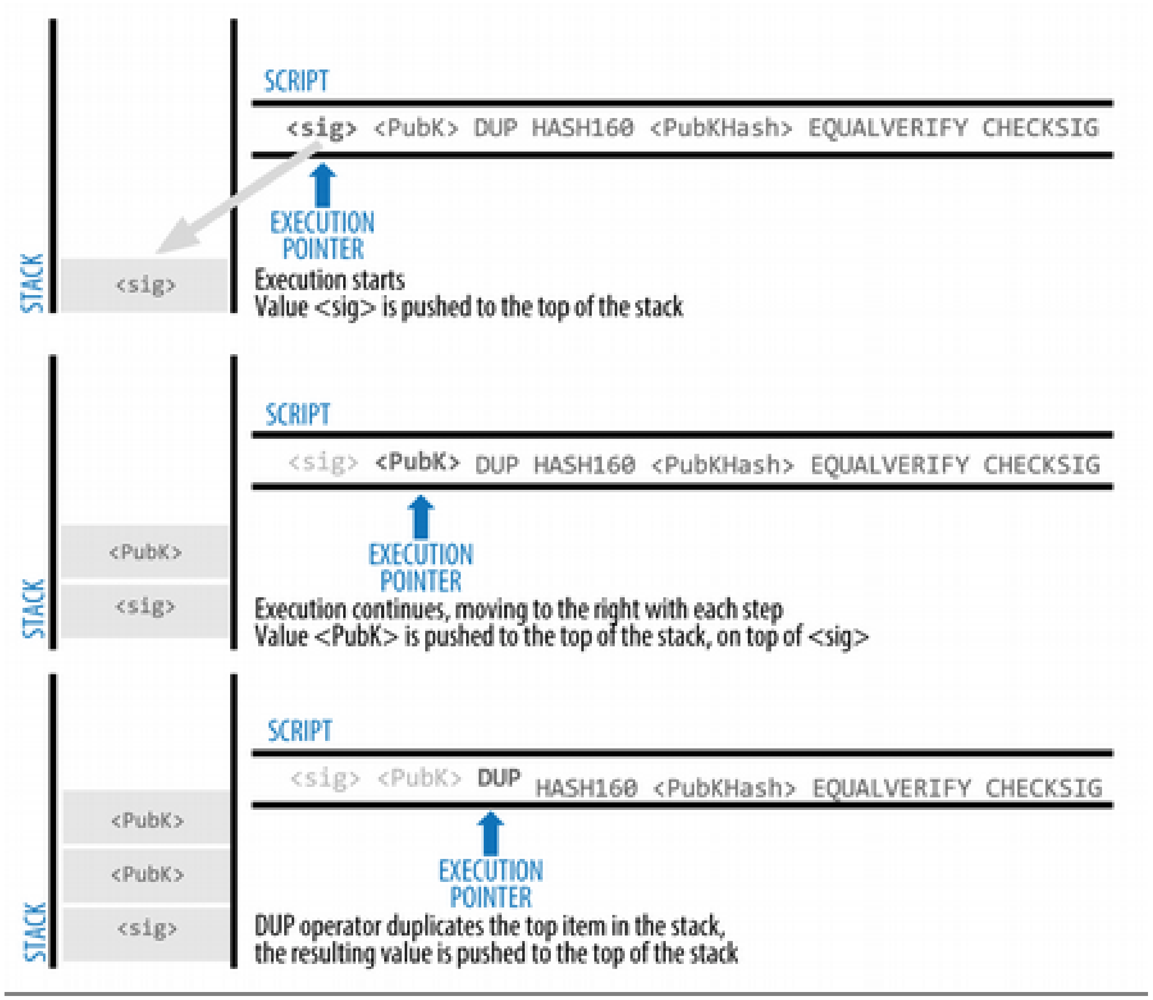
  OP_DUP OP_HASH160 <Receiver's address> OP_EQUAL OP_CHECKSIG

  Note: remember an address is a hash of a public key.
- An output whose locking script matches a P2PKH script will be only spendable by whoever owns the private key of an address so he can construct an input giving:
  - The public key, which will be hashed using Ripemd160(Sha256()) and compared to the receiver’s address in the script.
  - A signature result of signing the transaction hash with the public key.

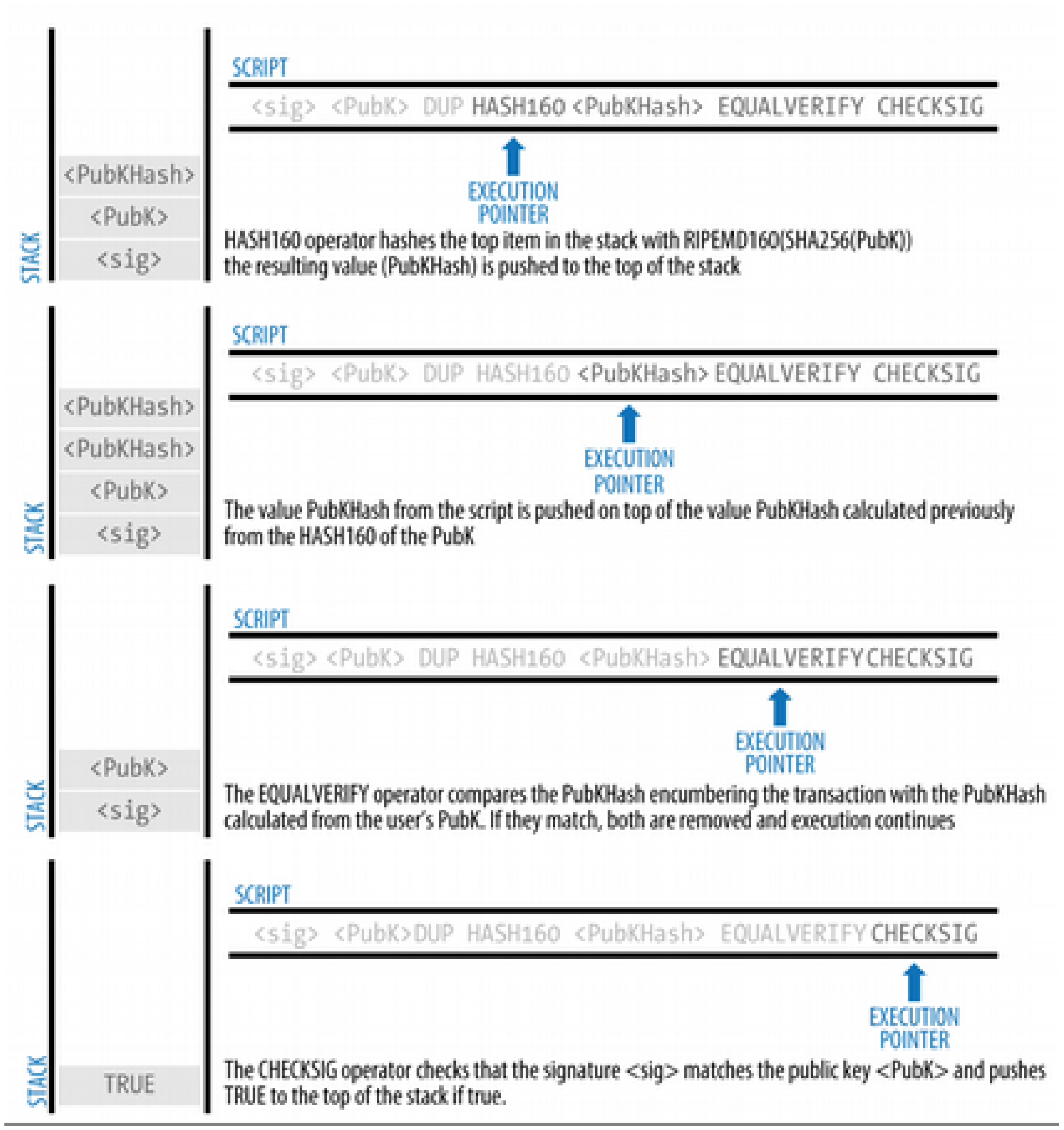
  Next page, example of a P2PKH Script execution stack-trace.

P2PKH Script Stack

Input provides <sig> and <pubK>


SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<sig>
Execution starts
Value <sig> is pushed to the top of the stack
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<PubK>
<sig>
Execution continues, moving to the right with each step
Value <PubK> is pushed to the top of the stack, on top of <sig>
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<PubK>
<PubK>
<sig>
DUP operator duplicates the top item in the stack,
the resulting value is pushed to the top of the stack

P2PKH Script Stack



SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<PubKHash>
<PubK>
<sig>
HASH160 operator hashes the top item in the stack with RIPEMD160(SHA256(PubK)) the resulting value (PubKHash) is pushed to the top of the stack
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<PubKHash>
<PubKHash>
<PubK>
<sig>
The value PubKHash from the script is pushed on top of the value PubKHash calculated previously from the HASH160 of the PubK
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<PubK>
<sig>
The EQUALVERIFY operator compares the PubKHash encumbering the transaction with the PubKHash calculated from the user's PubK. If they match, both are removed and execution continues
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
TRUE
The CHECKSIG operator checks that the signature <sig> matches the public key <PubK> and pushes TRUE to the top of the stack if true.

Source: https://www.linkedin.com/pulse/bitcoin-blockchain-part-iii-transactions-junbang-huang/

What is NAV Coin

- Nav Coin is a decentralized cryptocurrency built on top of Bitcoin Core v0.13.1, with added features and functionality.
- It was created to make online payments easy to do, at low cost, and all the while making sure to protect your privacy.
- Open Source project.
- Publicly traded since 2014. Valued at 245M$ (as of 7.01.2017).
- http://www.navcoin.org
- Uses Proof of Stake instead of Proof of Work as consensus algorithm.
- Shorter block time of 30 secs (vs. 10 min Bitcoin)
- Bigger max. block weight (2+6MB with Segregated Witness vs. 1+3MB of Bitcoin)
- Uses a second layer of nodes and a different blockchain to mix coins and obfuscate transactions.

NAV Coin Community Fund

- Finding a better way to fund Nav Coin and accelerate progress on our major side projects. Empowering the community.
- For this purpose, we're creating a decentralized NAV Coin Community fund
- Feature currently in beta phase. Users need to use version 4.1.x of the wllet and connect to the testnet to participate on it.
- Activated through a voting process along the members of the network.
- Details: https://navcoin.org/community-fund/
- TL;DR:

  Fixed amount of 0.25 NAV Coin/block to NAV Coin Community fund

  Anyone can make a funding proposal

  Payout's of NAV happen when everyone votes that the work is completed

  What do we need at a protocol level to implement something like this?

Adding new scripting opcodes

- From src/script.h:

```
/** Script opcodes */
enum opcodetype
{
    // push value
    OP_0 = 0x00,
    OP_FALSE = OP_0,
    OP_PUSHDATA1 = 0x4c,
    OP_PUSHDATA2 = 0x4d,
     ....
    OP_CFUND = 0xc1,
    OP_PROP = 0xc2,
    OP_PREQ = 0xc3,
    OP_YES = 0xc4,
    OP_NO = 0xc5,
     .....
```

NEW OPCODES

Changes in the code for the new Fund Contribution Script

- Namespace CFund: Declared in src/consensus/cfund.h

```
void CFund::SetScriptForCommunityFundContribution(CScript &script)
{
    script.resize(2);
    script[0] = OP_RETURN;
    script[1] = OP_CFUND;
}
```

- Extension of the CScript class:

```
bool CScript::IsCommunityFundContribution() const
{
    return (this->size() == 2 &&
      (*this)[0] == OP_RETURN &&
      (*this)[1] == OP_CFUND);
}
```

Adding a 0.25 NAV contribution per block

New consensus rule!!

- When creating new blocks, the coinstake transaction need to include a new output.

From src/wallet/wallet.cpp:

□

```
bool CWallet::CreateCoinStake(const CKeyStore& keystore, unsigned int nBits, int64_t nSearchInterval, int64_t
nFees, CMutableTransaction& txNew, CKey& key) {

.....

    // Adds Community Fund output if enabled

    if(IsCommunityFundEnabled(pindexPrev, Params().GetConsensus())) {

        int nFundIndex = txNew.vout.size();

        txNew.vout.resize(nFundIndex + 1);

        CFund::SetScriptForCommunityFundContribution(txNew.vout[nFundIndex].scriptPubKey);

        txNew.vout[fundIndex-1].nValue = COMMUNITY_FUND_AMOUNT;

    }

....
```

Checking every block contains the contribution

Checking the new consensus rule…

- When validating new blocks, the coinstake transaction needs to include an output with a contribution script.

From src/main.cpp:

```
bool ConnectBlock(const CBlock& block, CValidationState& state, CBlockIndex* pindex,
                  CCoinsViewCache& view, const CChainParams& chainparams, bool fJustCheck, bool fProofOfStake)
{
……
              if(IsCommunityFundEnabled(pindex->pprev, Params().GetConsensus()))
              {
                if(!tx.vout[tx.vout.size() - 1].IsCommunityFundContribution())
                  return state.DoS(100, error("ConnectBlock(): block does not contribute to the community fund"),
REJECT_INVALID, "no-cf-amount");
                if(tx.vout[tx.vout.size() - 1].nValue != COMMUNITY_FUND_AMOUNT)
                  return state.DoS(100, error("ConnectBlock(): block pays incorrect amount to community fund (actual=%d vs
consensus=%d)", tx.vout[tx.vout.size() - 1].nValue, COMMUNITY_FUND_AMOUNT), REJECT_INVALID, "bad-cf-amount");
….
```

Other needed changes…

- Creating CProposal and CPaymentRequest classes.
- Adding methods to create proposals and payment requests.
- Storing proposals and p.req. on disk. Class CCFundDB. Using levelDB by Google.
- Tracking new proposals and payment requests.
- Signaling for votes. Casting votes.
- Unlocking coins based on votes.

I'll show and explain you some code…

https://github.com/NAVCoin/navcoin-core/blob/v4.1.2/src/consensus/cfund.cpp

https://github.com/NAVCoin/navcoin-core/blob/v4.1.2/src/txdb.cpp

https://github.com/NAVCoin/navcoin-core/blob/v4.1.2/src/miner.cpp#L200

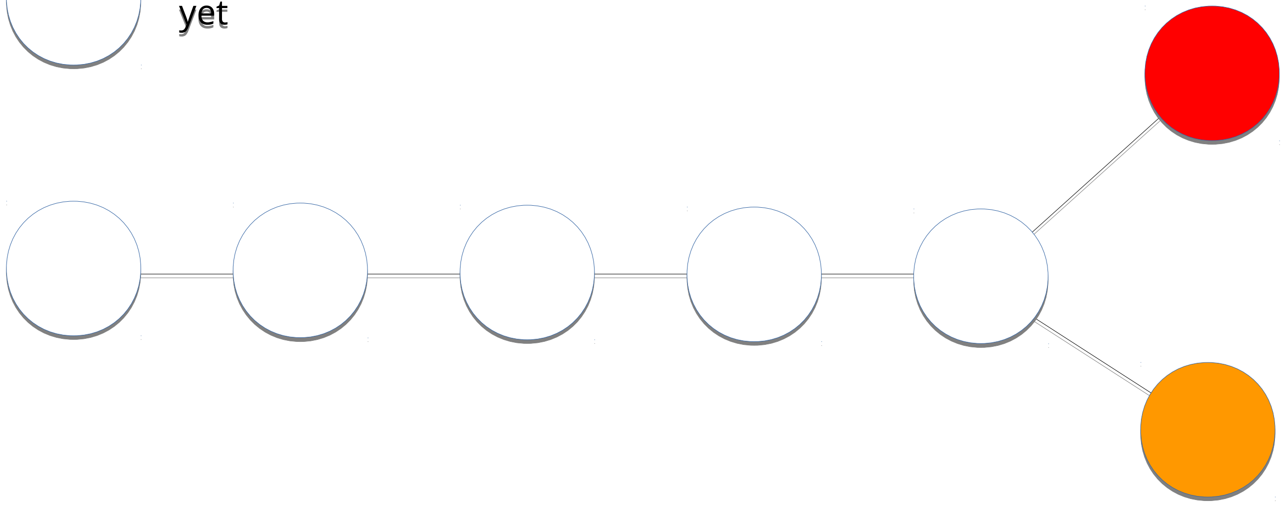
https://github.com/NAVCoin/navcoin-core/blob/v4.1.2/src/main.cpp#L3472

What happens if the consensus is broken

🟠 Block created by node with COMMUNITY_FUND_AMOUNT = 0.25

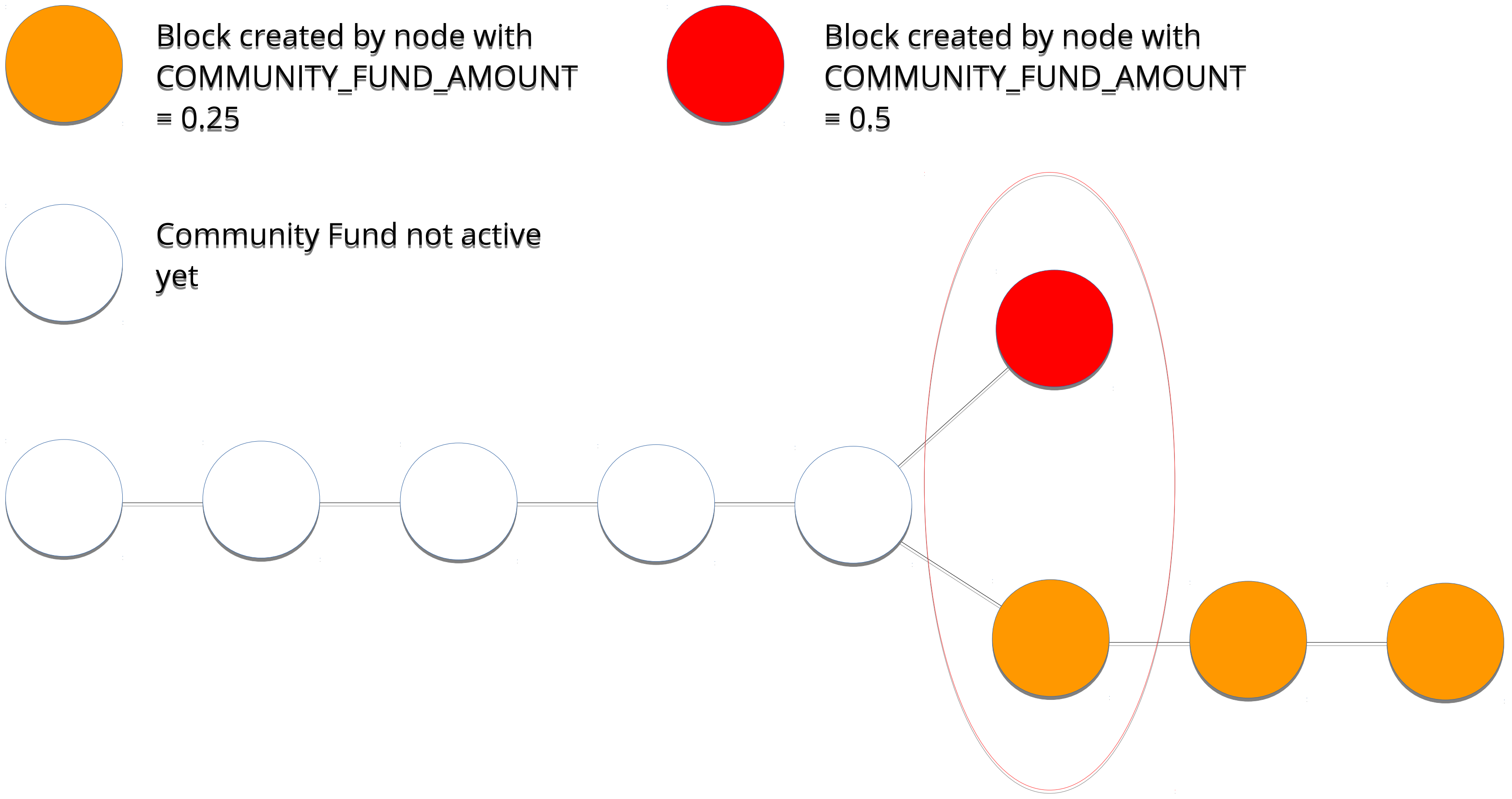
🔴 Block created by node with COMMUNITY_FUND_AMOUNT = 0.5

⚪ Community Fund not active yet



2 different chains are created!
State of the transaction history is conserved up to the fork divergence point.
But nodes can’t add new blocks on the other chain anymore!

What happens if the consensus is broken


Block created by node with COMMUNITY_FUND_AMOUNT = 0.25
Block created by node with COMMUNITY_FUND_AMOUNT = 0.5
Community Fund not active yet

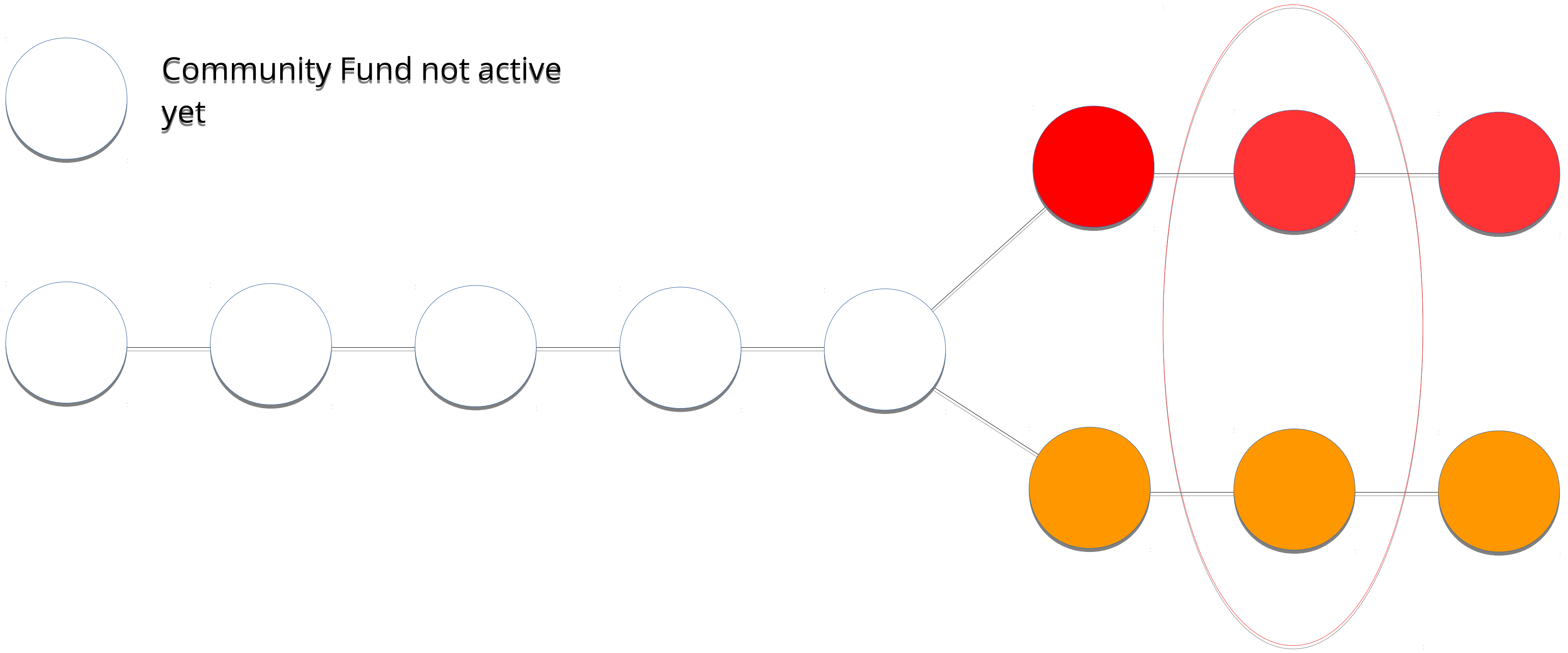
COMMUNITY_FUND_AMOUNT is different between nodes

What happens if the consensus is broken

🟠 Block created by node with COMMUNITY_FUND_AMOUNT = 0.25

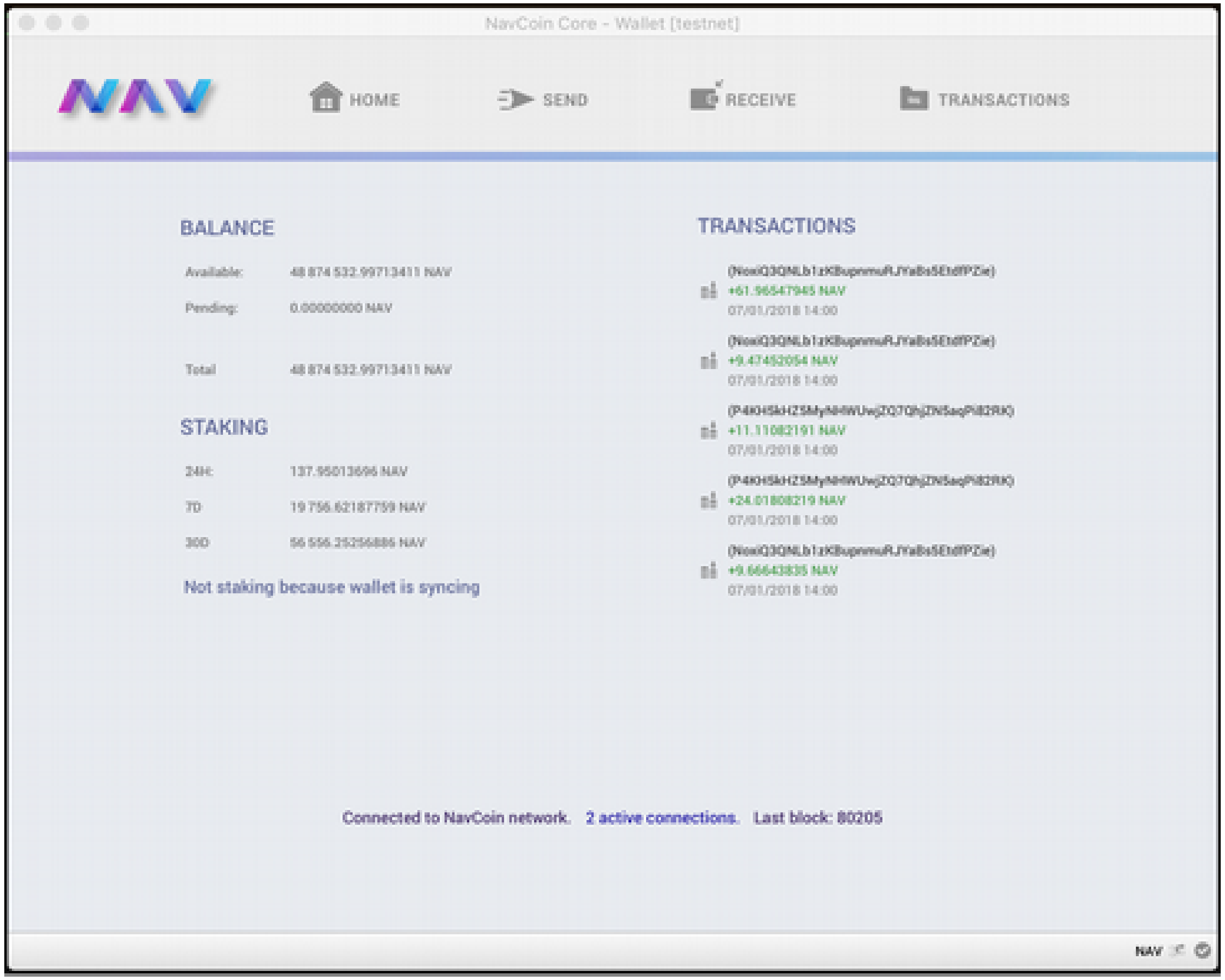
🔴 Block created by node with COMMUNITY_FUND_AMOUNT = 0.5

⚪ Community Fund not active yet



COMMUNITY_FUND_AMOUNT and hashPrevBlock are different between nodes

DEMO



NavCoin Core - Wallet [testnet]

BALANCE

Available:	48 874 532.99713411 NAV
Pending:	0.00000000 NAV
Total	48 874 532.99713411 NAV

STAKING

24H:	137.95013696 NAV
7D	19 756.62187759 NAV
30D	56 556.25256886 NAV

Not staking because wallet is syncing

TRANSACTIONS

(NoxiQ3QNLb1zKBupnmuRJYaBs5EtdfPZie)
+61.96547945 NAV
07/01/2018 14:00

(NoxiQ3QNLb1zKBupnmuRJYaBs5EtdfPZie)
+9.47452054 NAV
07/01/2018 14:00

(P4KHSkHZ5MyNHWUwjZQ7QhjZN5aqPi82RK)
+11.11082191 NAV
07/01/2018 14:00

(P4KHSkHZ5MyNHWUwjZQ7QhjZN5aqPi82RK)
+24.01808219 NAV
07/01/2018 14:00

(NoxiQ3QNLb1zKBupnmuRJYaBs5EtdfPZie)
+9.66643835 NAV
07/01/2018 14:00

Connected to NavCoin network. 2 active connections. Last block: 80205

Deterministic Build System

- A cryptocurrency is a critical piece of software as it’s responsible for securing user’s assets (financial value).
- A user should be able to verify how genuine a binary is.
- A software cannot easily be built reproducibly if the source varies depending on factors that are hard or impossible to control like the ordering of files on a filesystem or the current time.

- Gitian is used to create multi architecture binaries in a virtual machine.

- For each build a new VM is created using an optimized deterministic process everyone can replicate. Binaries can later be signed by builders.

  Every 24 hours last state of the repository is built so we can detect possible issues on the code.

Git

- Master branch always points to latest stable release.
- Tags and branches for different versions.
- GitHub issues used as a bug tracker.
- Contributors can create pull requests, which will be later reviewed and merged if approved.

Questions Round.

?

THANKS FOR COMING.

:)