Transactions and Blocks in Bitwise Detail

Ryan X. Charles
Blockchain University
Tokyo, Dec. 19, 2015

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Outline

Blockchain Overview

Number Formats

Public Keys, Signatures, & Addresses

Inputs & Outputs

Transactions

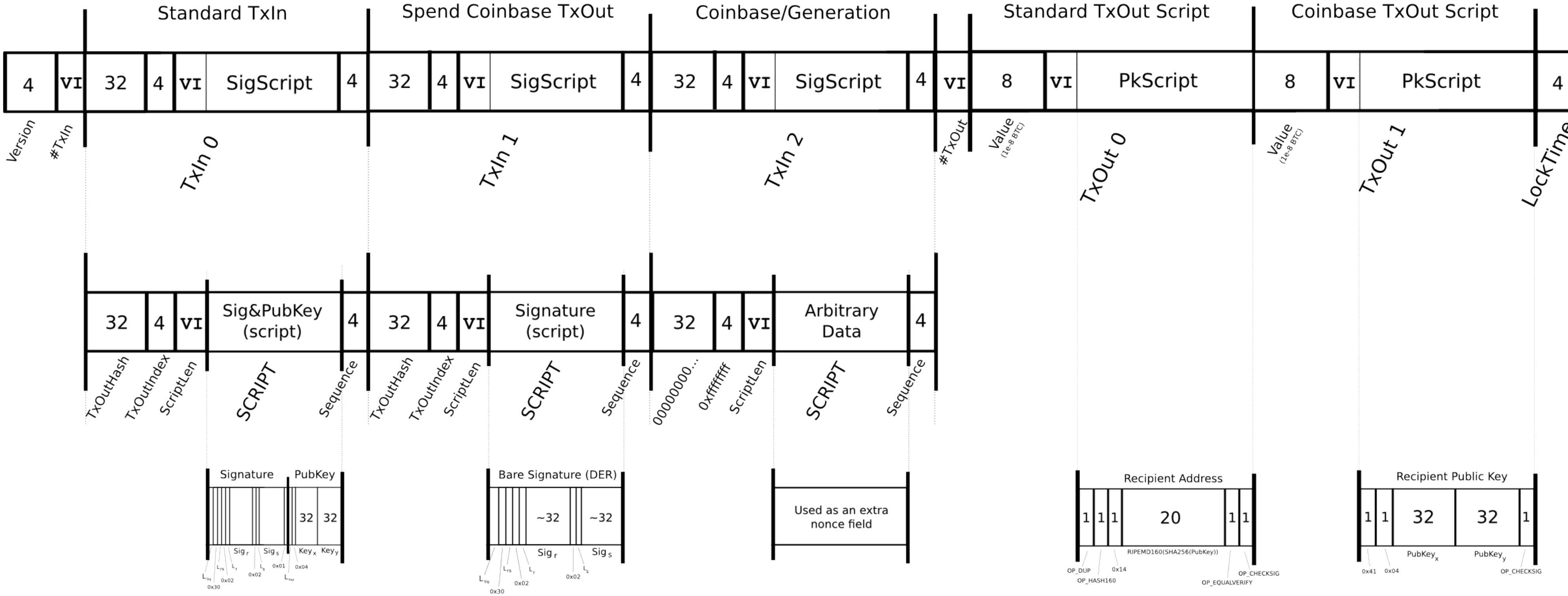
Blocks

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Transaction Detail Overview



Summary: Transactions have inputs and outputs. Plot by Alan Rainer.

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Number Formats

Which number format does bitcoin use?

- Big endian
- Little endian
- Sign+magnitude
- Two's complement
- Fixed point
- Floating point
- String

- 8 bit
- 16 bit
- 32 bit
- 64 bit
- 256 bit
- Custom variable size
- Another variable size

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Number Formats

Which number format does bitcoin use?

All of them.

"I wanted to see how many number formats I could use in a single protocol, so I invented bitcoin."
- Satoshi Nakamoto (apocryphal)

Endianness

Big endian
- One thousand
- 1000 (decimal)
- 3e8 (hex)
- 3 232 (decimal bytes)
- 03 e8 (hex bytes)
- 00000011 11101000 (bin bytes)

Little endian
- One thousand
- 0001 (decimal)
- 8e3 (hex)
- 232 3 (decimal bytes)
- e8 03 (hex bytes)
- 11101000 00000011 (bin bytes)

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Sign (+/-)

Sign magnitude

Most significant bit: 0 pos., 1 neg.

(4 bytes, big endian)

- -1 (decimal)
- -1 (hex)
- 128 0 0 1 (decimal bytes)
- `80 00 00 01` (hex bytes)
- `10000000 00000000 00000000 00000001` (bin bytes)

Two's complement

$X \rightarrow 2^N - X$, where N is bits

$-1: 1 \rightarrow 2^{32} - 1 = 4294967295$

(4 bytes, big endian)

- -1 (decimal)
- -1 (hex)
- 255 255 255 255 (decimal bytes)
- `ff ff ff ff` (hex bytes)
- `11111111 11111111 11111111 11111111` (bin bytes)

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Sign (+/-)

Sign magnitude

Most significant bit: 0 pos., 1 neg.

(4 bytes, little endian)

- -1 (decimal)
- -1 (hex)
- 1 0 0 128 (decimal bytes)
- `01 00 00 80` (hex bytes)
- `00000001 00000000 00000000 10000000` (bin bytes)

Two's complement

$X \to 2^N - X$, where N is bits

-1: $1 \to 2^{32} - 1 = 4294967295$

(4 bytes, little endian)

- -1 (decimal)
- -1 (hex)
- 255 255 255 255 (decimal bytes)
- `ff ff ff ff` (hex bytes)
- `11111111 11111111 11111111 11111111` (bin bytes)

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Decimal to Hex Examples

Decimal	**4 hex bytes, little endian**
5	`05 00 00 00` (unsigned)
5	`05 00 00 00` (sign+magnitude)
5	`05 00 00 00` (two's complement)
-5	`05 00 00 80` (sign+magnitude)
-5	`fb ff ff ff` (two's complement)
-6	`06 00 00 80` (sign+magnitude)
-6	`fa ff ff ff` (two's complement)

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Varint (Compact Size)

- How to transmit the size of data?
  - Could always use 4 bytes
    - e.g., “600 bytes to follow” stored as 32 bit uint
    - Data size limited to max 4 byte int (4294967295 bytes)
- ...but **everyone** stores the blockchain.
    - `00 00 00 01` is 3 bytes of wasted space
    - Should use minimal # of bytes
- Bitcoin's “Compact Size”
  - Varies in size from 1 byte for small nums, 9 bytes for big

Varint (Compact Size)

- Compact Size, a.k.a. Varint
  - [header byte][0 – 8 extra bytes]
- Header byte:
  - 0-253: followed by 0 bytes; header represents size
  - 254: followed by 2 byte unsigned LE specifying size
  - 255: followed by 4 byte unsigned LE specifying size
  - 256: followed by 8 byte unsigned LE specifying size

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Varint Examples

Varint	**Decimal**
`00`	0
`01`	1
`fc`	253
`fd 00 80`	32768
`fd 03 42`	16899
`fd 00 00`	0
`fe 00 00 00 80`	2147483648
`fe 00 00 00 00`	0
`ff 00 00 00 00 00 00 00 80`	9223372036854775808
`ff 00 00 0b ef 88 54 b1 f2`	13123412341234

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Big Integers

- Largest standard types in C++ are 8 bytes
- secp256k1 uses 256 bit = 32 byte numbers
  - e.g. N (order of curve), G (base point, x, y both big ints)
- Hash function often interpreted as 256 bit int
  - PoW hash is interpreted as little endian 256 bit
- Sometimes big endian, sometimes little endian
- Sometimes unsigned, sometimes sign+magnitude
  - AFAIK big ints never appear as two’s complement
- Example, in big endian, hex:
  - `N = ffffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141`

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

ScriptNum

- Numbers on the stack, used in numerical operations.
- 0 – 4 bytes, little endian, …
- …unless they overflow. Can be more 5 – 8 bytes.
- Script number operations are only valid on 4 byte numbers.
  - e.g., OP_ADD, OP_SUB

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Fixed Point

- Constant # of decimal places beyond the decimal point.
- 1 satoshi = 0.00000001 bitcoins
- Bitcoins, or BTC, always represented as fixed point decimal with 8 decimal places
- Satoshis are always unsigned integers
  - 2.1 * 10^{14} satoshis; fits comfortably within the precision of javascript's Number type
- Satoshis and bitcoins are NOT floating point
  - 10^{-9} is not a valid bitcoin amount

Floating Point

- Variable # of decimal places beyond the decimal point.
- $(-1)^{s} * c * b^{q}$
- as string, e.g., “6e-45” or “-3.345e12”
- Standard binary format is IEEE 754
- Used to compute mining difficulty
- -6.123e90 in 8 byte BE double: `d2c80bf4c18a3da4`
- -6.123e90 in 8 byte LE double: `a43d8ac1f40bc8d2`

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Number Formats

“If a number format exists, bitcoin uses it somewhere.”
Satoshi Nakamoto (apocryphal)

- Big endian (keys, sigs)
- Little endian (hashes)
- Sign+magnitude (bigints)
- Two's complement (ints)
- Fixed point (BTC/satoshis)
- Floating point (mining)
- String (display, RPC, APIs)
- 8 bit (opcodes, varint)
- 16 bit (opcodes, varint)
- 32 bit (opcodes, varint)
- 64 bit (ScriptNum)
- 256 bit (privkeys, hashes)
- Custom variable size (data sizes)
- Another variable size (script ops)

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Public Keys

- P = pG
  - a public key (P) is a point equal to the private key (p; a big number) multiplied by G (the base point of the curve)
- P = (x, y)
- simplest representation is **[x][y]**
- However, y can be derived from x if you know whether it is odd or even
- ...therefore a more compact representation: **[y is odd][x]**

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Public Keys

- DER format - the format public keys take in a transaction
- “Compressed” - pubkeys are almost always stored this way.
  - **[isOdd][x]**
  - x is 256 bit big endian unsigned integer
  - isOdd is either 2 (y is odd), or 3 (y is even)
- “Uncompressed” - obsolete, unnecessarily inefficient.
  - **[4][x][y]**
  - x, y are 256 bit big endian unsigned integers
- “Other” - header byte can also be 6 or 7, interpret same as uncompressed.
  - http://sourceforge.net/p/bitcoin/mailman/message/29416133/

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Signatures

- An secp256k1 ECDSA signature consists of (r, s), two 256 bit integers
- A “compact” signature, not present in the blockchain, is simply:
  - **[r][s]**, where r and s are both unsigned 256 bit integers
  - Usually encoded as base64, particularly with bitcoin message signing (again, not in the blockchain)

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Signatures

- In the blockchain, a signature is encoded in this form:
  - **[DER signature][hashtype]**
  - hashtype is 1 byte: (SIGHASH_ALL, SIGHASH_NONE, SIGHASH_SINGLE) | SIGHASH_ANYONECANPAY
- DER signature:
  - **[h][l][rh][rl][r][sh][sl][s]**
  - h (0x30), l (length of what follows), rh (2), rl (length of r), sh (2), sl (length of s) are all 1 byte
  - r, s are *sign-magnitude* variable-size big integers, that are never negative
- Summary, in the blockchain: **[0x30][l][2][rl][r][2][sl][s][hashtype]**

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Opcodes

- One byte specifying an operation.

- OP_FALSE: 0x00,
- OP_0: 0x00,
- OP_PUSHDATA1: 0x4c,
- OP_PUSHDATA2: 0x4d,
- OP_PUSHDATA4: 0x4e,
- OP_1NEGATE: 0x4f,
- OP_RESERVED: 0x50,
- OP_TRUE: 0x51,
- OP_1: 0x51,
- OP_2: 0x52,
- OP_3: 0x53,
- OP_4: 0x54,
- OP_5: 0x55,
- OP_6: 0x56,
- OP_7: 0x57,
- OP_8: 0x58,
- OP_9: 0x59,
- OP_10: 0x5a,
- OP_11: 0x5b,
- OP_12: 0x5c,
- OP_13: 0x5d,
- OP_14: 0x5e,
- OP_15: 0x5f,
- OP_16: 0x60,
- OP_NOP: 0x61,
- OP_VER: 0x62,
- OP_IF: 0x63,
- OP_NOTIF: 0x64,
- OP_VERIF: 0x65,
- OP_VERNOTIF: 0x66,
- OP_ELSE: 0x67,
- OP_ENDIF: 0x68,
- OP_VERIFY: 0x69,
- OP_RETURN: 0x6a,
- ... ~175 total, including push data

Scripts

- Series of operations and data to be pushed to the stack.
- Bitcoin has a stack and an “alt stack” - no heap.
- Example: OP_5 OP_5 OP_EQUALVERIFY
- Hex bytes: `555588`
- Example: 3 0x606060 3 0x606060 OP_EQUALVERIFY
- Hex bytes: `036060600360606088`

Transaction Inputs

- **[txhashbuf][txoutnum][scriptvi][script][seqnum]**
- txhashbuf: hash of input tx, 256 bits
- txoutnum: UInt32LE, which prev. output is being input
- scriptvi: varint for length of script
- script: scriptSig - executed after scriptPubKey
- seqnum: mostly unused Uint32LE for updating a transaction
- before it gets in a block, usually (uint)(-1) = `0xffffffff` (note two’s complement)

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Transaction Outputs

- **[value][scriptvi][script]**
- value: UInt64LE - number of satoshis to send
- scriptvi: varint for length of script
- script: the scriptPubKey - executed before scriptSig

Transactions

- **[version][txinsvi][txins][txoutsvi][txouts][nlocktime]**
- version: UInt32LE, presently 1
- txinsvi: Varint, the number of inputs
- txins: the inputs concatenated together
- txoutsvi: Varint, the number of outputs
- txouts: the outputs concatenated together
- nlocktime: UInt32LE, block or time at which tx is valid, usually 0

Transactions

- Random, simple example, in the blockchain:
- txid (reverse of hash):

  f45fa87b358285858073 4cb9c584264c8cfeb4e5b49ed5ec58d31850f2296da9
- tx hash:

  a96d29f25018d358ecd59eb4e5b4fe8c4c2684c5b94c7380858582357ba85ff4
- tx:

  0100000001795b88d47a74e3be0948fc9d1b4737f96097474d57151afa6f77c7879
  61e47cc120000006a47304402202289f9e1ae2ed981cd0bf62f822f6ae4aea40c65
  c7339d90643cea90de93ad1502205c8a08b3265f9ba7e99057d030d5b91c889a1b9
  9f94a3a5b79d7daaada2409b6012103798b51f980e7a3690af6b43ce3467db75bed
  e190385702c4d9d48c0a735ff4a9ffffffff01c0a8320000000000001976a91447b8e
  62e008f82d95d1f565055a8243cc243d32388ac00000000

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Transactions

- Parsing the transaction with fullnode:

```
fullnode> var tx =
Tx().fromHex('0100000001795b88d47a74e3be0948fc9d1b4737f96097474d57151afa6f77c787961e47cc120000006a47304402202289f9e1ae2ed981cd0bf62f822f6
ae4aea40c65c7339d90643cea90de93ad1502205c8a08b3265f9ba7e99057d030d5b91c889a1b99f94a3a5b79d7daaada2409b6012103798b51f980e7a3690af6b43c
e3467db75bede190385702c4d9d48c0a735ff4a9ffffffff01c0a83200000000001976a91447b8e62e008f82d95d1f565055a8243cc243d32388ac00000000')
fullnode> tx.toJSON()
{ version: 1,
  txinsvi: '01',
  txins:
   [ { txhashbuf: '795b88d47a74e3be0948fc9d1b4737f96097474d57151afa6f77c787961e47cc',
       txoutnum: 18,
       scriptvi: '6a',
       script: '71
0x304402202289f9e1ae2ed981cd0bf62f822f6ae4aea40c65c7339d90643cea90de93ad1502205c8a08b3265f9ba7e99057d030d5b91c889a1b99f94a3a5b79d7daaa
da2409b601 33 0x03798b51f980e7a3690af6b43ce3467db75bede190385702c4d9d48c0a735ff4a9',
       seqnum: 4294967295 } ],
  txoutsvi: '01',
  txouts:
   [ { valuebn: '3320000',
       scriptvi: '19',
       script: 'OP_DUP OP_HASH160 20 0x47b8e62e008f82d95d1f565055a8243cc243d323 OP_EQUALVERIFY OP_CHECKSIG' } ],
  nlocktime: 0 }
```

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Transactions

- The one and only input:

```
fullnode> tx.txins[0].toJSON()
{ txhashbuf:
'795b88d47a74e3be0948fc9d1b4737f96097474d57151afa6f77c787961e47cc',
  txoutnum: 18,
  scriptvi: '6a',
  script: '71
0x304402202289f9e1ae2ed981cd0bf62f822f6ae4aea40c65c7339d90643cea90de93ad15
02205c8a08b3265f9ba7e99057d030d5b91c889a1b99f94a3a5b79d7daaada2409b601 33
0x03798b51f980e7a3690af6b43ce3467db75bede190385702c4d9d48c0a735ff4a9',
  seqnum: 4294967295 }
```

Transactions

- The one and only output:

```
fullnode> tx.txouts[0].toJSON()
{ valuebn: '3320000',
  scriptvi: '19',
  script: 'OP_DUP OP_HASH160 20 0x47b8e62e008f82d95d1f565055a8243cc243d323
OP_EQUALVERIFY OP_CHECKSIG' }
```

Transactions

- Input script:

71
0x304402202289f9e1ae2ed981cd0bf62f822f6ae4aea40c65c7339d90643cea90de93ad15
02205c8a08b3265f9ba7e99057d030d5b91c889a1b99f94a3a5b79d7daaada2409b601 33
0x03798b51f980e7a3690af6b43ce3467db75bede190385702c4d9d48c0a735ff4a9

- Output script:

OP_DUP OP_HASH160 20 0x47b8e62e008f82d95d1f565055a8243cc243d323
OP_EQUALVERIFY OP_CHECKSIG

- ...will explain scripts in detail in another talk.

Blocks

- Meta data
- List of transactions
- Block header
  - Contains hash of previous block (hence “block chain”, chain of blocks back to genesis block)
  - Contains Merkle root of transactions list

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Merkle Trees

- How to download a piece of a file, and know that the piece is correct?
- Hash data one time:
- [data1][data2][data3][data4]
- [hash]
- ...must download all data pieces to confirm hash

Merkle Trees

- Hash data in pieces:
- [data1][data2][data3][data4]
- [hash1][hash2][hash3][hash4]
- ...better, but must download all hashes, which still scales the same as the size of the data, O(N)

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Merkle Trees

- Hash data in a tree:
- [data1][data2][data3][data4]
- [hash1][hash2][hash3][hash4]
- [hash1'][hash2']
- [hash1'']
- ...best, only have to download branch of the tree that is relevant to the data piece you are downloading

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Merkle Trees

- Hash data in a tree:
- **[data1]**[data2][data3][data4]
- **[hash1][hash2]**[hash3][hash4]
- **[hash1'][hash2']**
- **[hash1'']**
- ...**verified** in O(log(N))

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Merkle Trees

- Hash transactions in a tree:
- [tx1][tx2]**[tx3]**[tx4][tx5][tx6][tx7]
- [h1][h2]**[h3][h4]**[h5][h6][h7][h7] ← last one dup.
- **[h1'][h2']**[h3'][h4']
- **[h1''][h2'']**
- **[h1''']** ← merkle root
- ...**bolded** hashes must be downloaded to confirm tx3 is in the block

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Merkle Trees

- Merkle trees are most useful for SPV nodes, so you can validate that your transaction (e.g., a payment to you) has in fact been placed in a block, without needing the full blockchain.

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Note on Hashes

- Transaction and block hashes are reversed when displayed.
- i.e., they are internally little endian, but displayed big endian.
- e.g., displayed block hash is big endian:
  - `00000000000000001176c3c6922c5ade5c6c10ab69d0545a4abe07705dfd8e31`
- But computed hash looks like this:
  - `318efd5d7007be4a5a54d069ab106c5cde5a2c92c6c376110000000000000000`

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Target

- Hash that next block must be lower than
- Example block hash:
  - 00000000000000001176c3c6922c5ade5c6c10ab69d0545a4abe07705dfd8e31
- Lower than hypothetical target:
  - 0000000000000000120000000000000000000000000000000000000000000000
- “Difficulty” is inverse of target
- Compact 4 byte floating point representation of difficulty stored in blocks, as “bits”

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Proof-of-Work

- Finding a low hash requires iterating the nonce many, many times
- Hypothetical target:
  - 0000000000000000120000000000000000000000000000000000000000000000
- Lower hypothetical target:
  - 0000000000000000001200000000000000000000000000000000000000000000
- “Difficulty” is inverse of target
- …requires exponentially more iterations (2^8 = 256) to find a hash lower than a target with one byte more zeroes

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Block Header

- **[version][prevblockhash][merkleroot][time][bits][nonce]**
- version: UInt32LE, presently 2
- prevblockhash: 256 bits, hash of previous block
- merkleroot: 256 bits, root of transaction merkle tree
- time: UInt32LE, somewhat inaccurate time in seconds
- bits: 4 bytes, compact form of mining target
- nonce: 4 bytes, iterated for mining

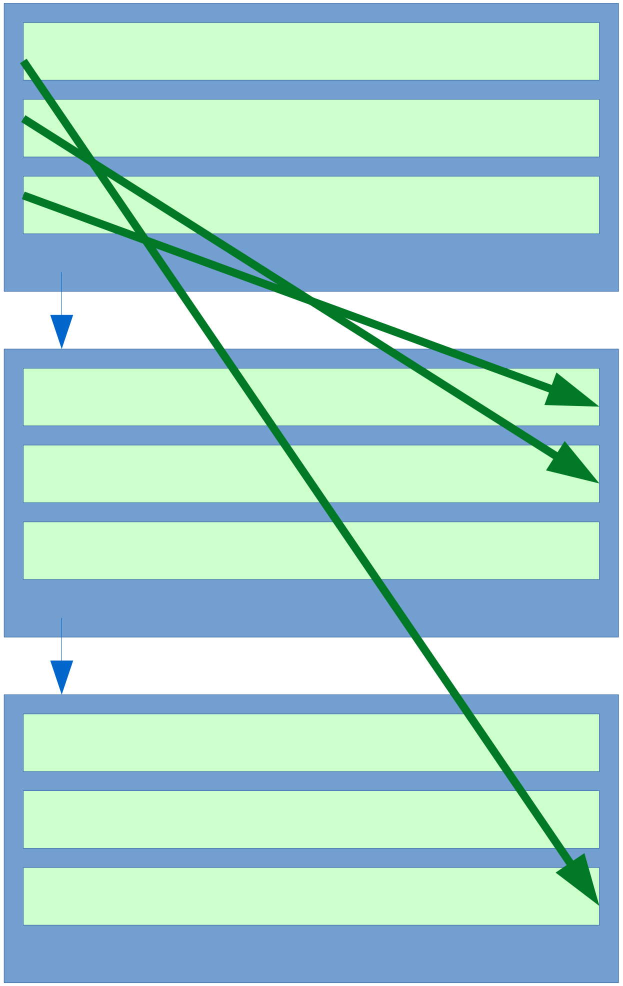
Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Block

- **[magicnum][blocksize][blockheader][txsvi][txs]**
- magicnum: UInt32LE, always 0xd9b4bef9
- blocksize: UInt32LE
- blockheader
- txsvi: Varint, number of transations
- txs: list of transactions

Summary

- Transactions have inputs and outputs – inputs link to earlier outputs
  - Coinbase transactions have null inputs, i.e. do not link to earlier blocks
- Blocks link to previous block
  - Genesis block has null previous block



Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles

Code Samples and Slides at:
github.com/ryanxcharles/blockchain-university

Datt

Homework

- Try converting some numbers (0, 1, -1, 5, -5, 5000, -5000) into 2-byte big endian and little endian sign+magnitude and two's complement numbers. Show the results in hex bytes and binary.
- Using standard libraries, find a recent transaction and a recent block, and hash them, and confirm that you get the correct hashes (and recall that the display hash is the reverse of the actual hash)
- Using standard libraries, parse a recent transaction and a recent block and spit out the structure of the transaction and block in JSON.
- Using standard libraries, find a recent block, and build a merkle tree, and confirm you get the correct merkle root found in the block.
- Try to "mine" the statement "Blockchain University" by appending a nonce and finding a hash that starts with at least one 0-byte.
- Implement a bitcoin full node using only cryptography libraries and no standard bitcoin libraries.

Ryan X. Charles
Founder of Datt (datt.co)
twitter.com/ryanxcharles
github.com/ryanxcharles