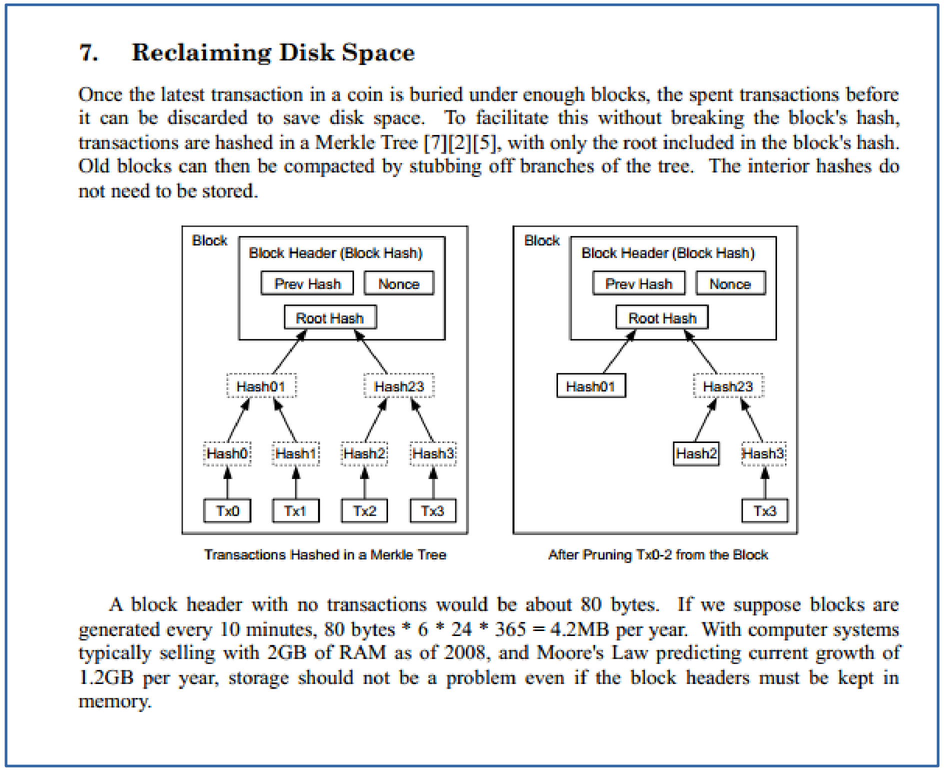
Scaling Bitcoin

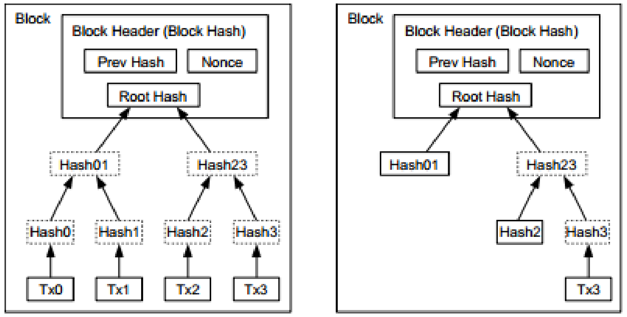
memoized genetic trees and loaded wallets

Folding Transaction Graphs



7. Reclaiming Disk Space

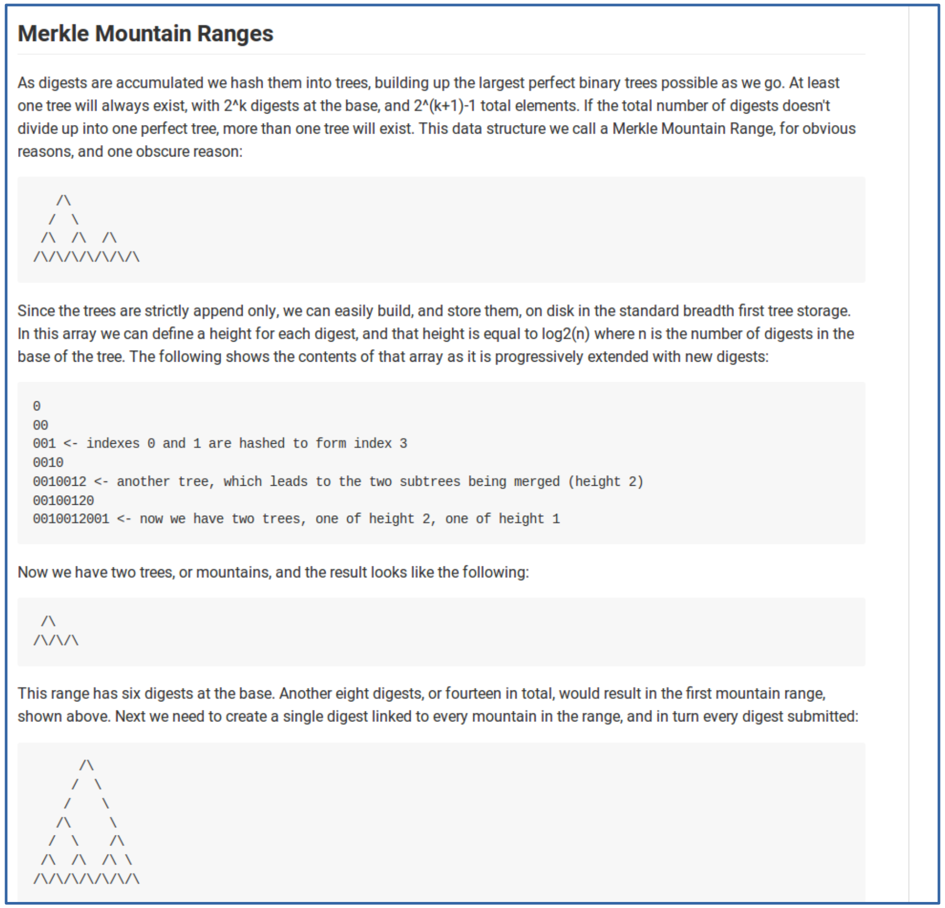
Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



Transactions Hashed in a Merkle Tree

After Pruning Tx0-2 from the Block

A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.



Merkle Mountain Ranges

As digests are accumulated we hash them into trees, building up the largest perfect binary trees possible as we go. At least one tree will always exist, with 2^k digests at the base, and 2^(k+1)-1 total elements. If the total number of digests doesn't divide up into one perfect tree, more than one tree will exist. This data structure we call a Merkle Mountain Range, for obvious reasons, and one obscure reason:

```
   /\
  /  \
 /\  /\  /\
/\/\/\/\/\/\/\
```

Since the trees are strictly append only, we can easily build, and store them, on disk in the standard breadth first tree storage. In this array we can define a height for each digest, and that height is equal to log2(n) where n is the number of digests in the base of the tree. The following shows the contents of that array as it is progressively extended with new digests:

```
0
00
001 <- indexes 0 and 1 are hashed to form index 3
0010
0010012 <- another tree, which leads to the two subtrees being merged (height 2)
00100120
0010012001 <- now we have two trees, one of height 2, one of height 1
```

Now we have two trees, or mountains, and the result looks like the following:

```
 /\
/\/\/\
```

This range has six digests at the base. Another eight digests, or fourteen in total, would result in the first mountain range, shown above. Next we need to create a single digest linked to every mountain in the range, and in turn every digest submitted:

```
      /\
     /  \
    /    \
   /\     \
  /  \    /\
 /\  /\  /\  \
/\/\/\/\/\/\/\
```

Folding TxOs

- < gmaxwell> “imagine we have just a regular blockchain, and we append new txouts to it as they are created. We compute a binary tree over this whole thing... using a tree update scheme that has ~O(1) append.

  [github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md]

  When someone wants to spend a coin, they give you a proof that shows you the coin is in the tree.

  Which is also the same data you need to replace that coin with a "deleted coin" entry and update the root.

  (by the same reason we can compose non-compressed proofs) so miners and full nodes just need to store the leading edge of this tree (log2(history)) hashes.

  and any transactions they recieve will have enough data to let them mark the inputs elsewhere in the tree as spent.”

- Key Idea: Fold all TxOs to **one hash**, that is enough to prove any of their existence.

- Issues:

  - Light/Full nodes are required

  - Live Wallets

    <gmaxwell> “But there is a trade-off: wallets have to actively monitor the network to process updates to their own utxo proofs or they will lose the ability to spend their coins.”

Folding UTxOs

```
ver
vin [
  Ref to prev tx
  utxo’s index
  scriptsig length
  scriptsig
  seq
  ] ...
vout [
  scriptpubkey length
  scriptpubkey
  ] ...
```

Transactions in bitcoin, are of the form:

[v +

V_{in} +

(U +

i +

l_{ss} +

S_s +

s) ... +

V_o +

(L_{spk} +

S_{pk}) + ...]

Hashed genetic family trees

- We store them as

  $\mathbf{U_t}$ = Hash [v + V_{in} + [($\mathbf{U_{t-1}}$ +i + l_{ss} + S_s + s) + ...] + V_o + [(l_{spk} + S_{pk}) + ...]] (t refers to state)

- Key Idea: we notice recursion

- Difference between linkedlists and blockchains

	Reference to previous	Independence (in its own)
linked list	address in memory	address in memory
blockchain	authorised content at that time(hash)	authorised content at that time(hash)
tx (??)	hash	hash

- Instead of seeing TxOs as mountains, we see each UTxOs history as a genetic family tree of that coin.

- Using the above analogy: Similar to how a miner is allowed to create a block, if they have the entire blockchain & refer to its last block's hash, we allow a user to create a UTxO if they have that coin's entire history and refer to its last UTxOs. This is similar to a miner giving you entire blockchain instead of just the latest block.

  The expanded form of above equation is the proof that wallets carry. Thus wallets can go offline.

- Issues:
  - Light/Full nodes are required
  - Proof verification takes time

UTxO Memoization

- UTxO calculation is expensive.

  U_t = Hash [v + V_{in} + [(U_{t-1} +i + l_{ssi} + S_{si} + s_i) + (U_{t-1} +j + l_{ssj} + S_{sj} + s_j) + ...] + V_o + [(l_{spk} + S_{pk}) + ...]]

- Key Idea: memoization of UtxO calculation

- Whenever we change a UTxO value we keep the previous values of the UTxOs it is changing.

- Issues:
  - Light/Full nodes

Coinbase UTxO

- Using the linked-list analogy again, if each UtxO redeem is giving us entire proof of its history, right upto its genesis in the coinbase of some block,

  we can do better **if**, like blocks referring all the way back to **a data that we trust** (the genesis block), we had some algorithm for UtxO tree to go back to some data we trust.

- Since we do not have any previous values stored for coinbase blocks we need a definition anyway.

- Key Idea: since hashing represents difficulty we can use it.

- For coinbase:

  U_{t-1} = Hash(blockHeight + blockHash)

- The way to enter the bitcoin today is to get the genesis block from a friend and know the protocol. Then ask your peers for data and you can verify and do everything based on those two things alone, even if there is one honest peer.

  In our scheme, you ask your friend for last block's hash, which is now a merkelized root of all the UtxOs & their memoizations. Then you ask your peers for data & you can verify everything.

  - First, this hash helps you download the last entire utxos & cached hashs tree & thus sync with current nodes without trust. Now like everyone we are waiting for next block from the miners.

  - Secondly, when a transaction is providing all of the expanded proof, it also supplies the blockhash for a blockheight, thus in case of conflict we trust the higher value.