Segwit in Bitcoin: Lessons Learned

Gregory Sanders

Myself

- Elements Project
  - Elements Alpha
  - Liquid
- Did review for Segwit in Core
  - Upstream review important for downstream
- Scaling(?)
  - How do we scale protocol development?

Segwit as a Solution

- First developed as an “element” of Elements Alpha
- Solves the problem of unintentional malleability.
  - TL;DR Allows safe chaining of pre-signed transactions for smart contracting in Bitcoin.
  - Payment Channels, Lightning
- But doesn’t fit into Bitcoin really.
  - We can’t just change txid formulation on a whim
  - No matter the technical benefits, hard to imagine non-security-fix HF

Segwit as Deployed

- Key insight: If you can deploy a type of “extension block”. Inside this extension nearly any rules can be enforced, turning hard forks into soft forks
- Many ideas like Confidential Transactions as Softfork
- Segwit, a highly desired extension, fits the bill
    - Special new transaction types are signaled via empty scriptSig and commitment to the witnessScript in scriptPubkey
    - Think P2SH, but hiding the new data inside the extension block
    - Now signatures are out of txid calculation for both new and old nodes.
    - Backwards compatible
- Old news now

Isn’t Segwit already done?

- Segwit is active in testnet, and close to release in mainnet
- One of the largest changes to Bitcoin ever
  - Touched nearly every part of the codebase: serialization, p2p, wallet, consensus
- Talking about this has two purposes:
  - Informational for those not privy to the sausage-making
  - What I see as takeaways from the exercise
- Any opinions are just mine

Minimum Viable Product (What didn’t make it)

- New address format(BIP142)
  - Hesitancy to perpetuate base58+checksum
  - Nested P2SH for now
- Additional tweaks to commitment structure
  - Arbitrary segwit commitment tree and fast hashing
  - Previous block witness commitment
- Validation Cost Metric
- Fraud Proofs
- New scripting*

* for the most part

Minimum Viable Product

- Each proposal needs a champion
- Each proposal increases demand for review
  - Already strapped for review as-is
- Even “too many BIP numbers” can be a problem
  - Downstream developers can’t figure out why signatures are failing (BIP143)
  - People still expecting BIP142 addresses

Sidenote: Scripting differences / similarities

- CHECKMULTISIG still requires an additional dummy argument in the stack
  - Null dummy softfork: #8636
- Sighash serialization overhaul (BIP143)
  - O(n) hashing
  - Value under hash!
- “Minimal if” as policy: #8526
- No uncompressed pubkeys as policy #8499 (?)
- Nullfail as policy #8634
- Fixes SIGHASH_SINGLE “one” bug
- ~~Low_s softfork~~
- Script versioning

Segwit Developed

- #segwit-dev (still ~52 users there, for some reason)
- Contention about where it should be discussed
  - Mailing list used to announce BIPs, major changes
  - Hesitancy to flood #bitcoin(-core)-dev
  - Further partitioning of IRC development presence
- Bulk of design done pre-PR
- 4 segnet iterations, starting with segnet1 in Dec. 2015
  - Were actively used by downstream developers
  - Spam 4MB blocks

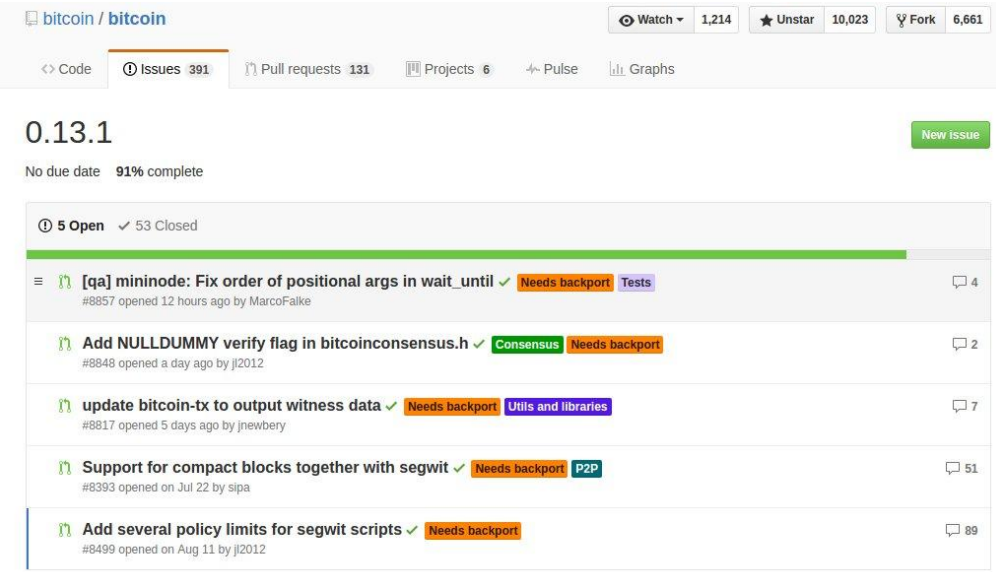
Segwit PR’d

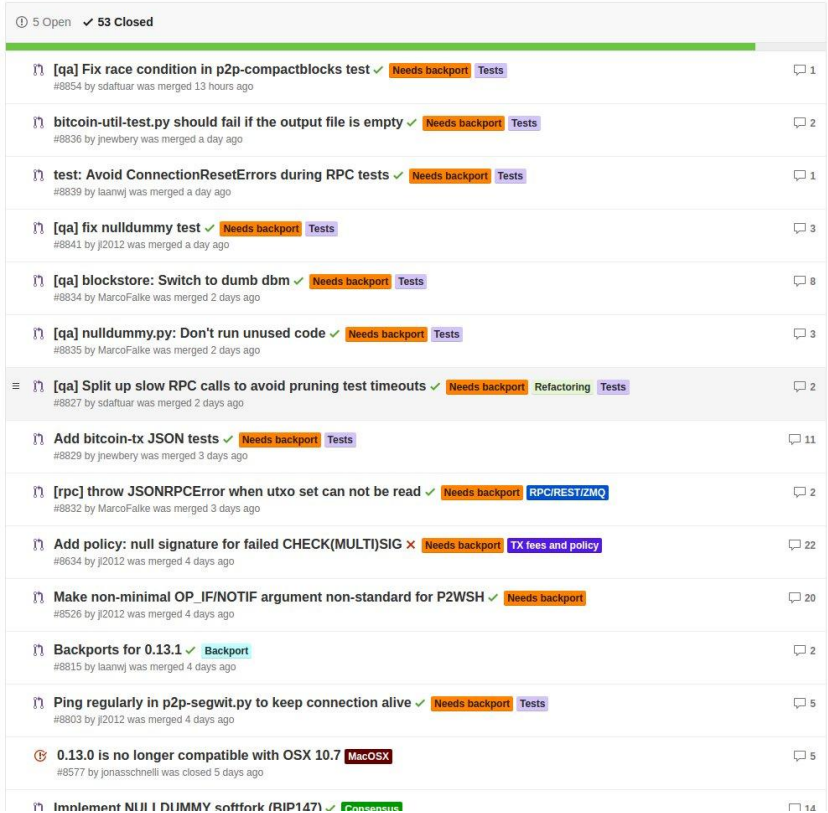
- 32 participants
- April 9th to June 24th
- The branch where comments were targeted
- ~1,486 lines of code for implementation
- ~3,338 lines of code for tests

Segwit Rebased

- Identical diff
- June 6 to June 26
- 0.13, Compact Blocks, and Segwit
  - Contention on when each should be merged
  - 0.12 backport promises?
- Merged, activated on testnet

Segwit merged to master. Done!





Backport Backlog

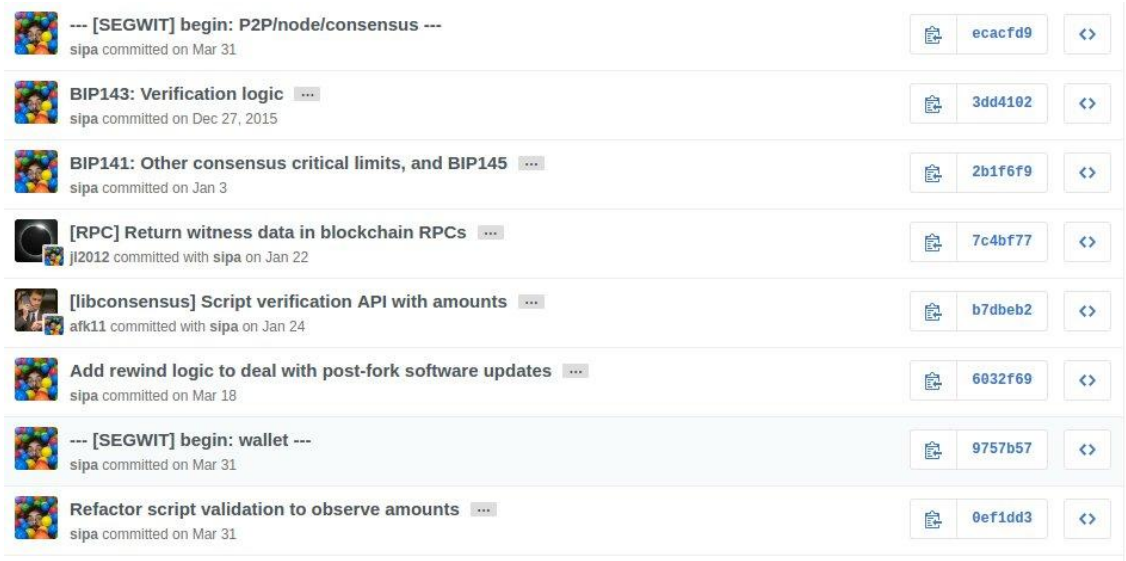
- Fix Segwit transaction blinding via reject filter
  - Spammers could temporarily stop a transaction from propagating
- Compact blocks for Segwit
  - Plus versioning negotiation
- Segwit wallet cleanups
- Softfork/policy follow-ons
- Slew of bugfixing backports
  - https://github.com/bitcoin/bitcoin/milestone/22
- Weeks of somewhat tedious irc dev meetings
- Getting close!

Proposal(s)

- Any non-trivial consensus change to Bitcoin in the future should have an actively-used testnet spun up.
  - If supposed downstream users aren't actively testing, is the change even desired?
  - Regular (ab)use helps to surface issues early
- Ride-along changes should be discussed, implemented, and tested as early as possible into the development cycle.
  - Other issues will surely pop up
- Tests should take up a large fraction of the loc changes
- Any additional technical channels should be carefully spun up, logged, and spun down at appropriate times
  - Avoid loss of design history, communal knowledge

Proposal(s)

- If a PR spans a number of layers:
  - Keep commits in logical partitions
  - Split sections with empty commits marking start/end



Proposal(s)

- Stop talking about block size
  - Let’s talk about “weight” and “throughput”
- Only spend time backporting major consensus changes when there is demand
- Don’t do large changes like this often
  - Higher amount of risk compared to usual
  - Slows other technical debt cleanup (libconsensus, network refactor, etc)
  - Is more uninviting for review
  - Collides into regular release schedule, causes confusion/tension

Thanks!

Softforks with increased risks

- Segwit nodes must find sufficient number of upgraded nodes
  - You may be partitioned off the network if you can’t find peers to serve new data
  - Same with any “extension block” style softfork
  - P2SH et al. suffer no such risks
- Mitigations:
  - Preferential connections used to mitigate
  - Find more compatible peers faster
    - “feeler” connections
- Need a long lead-time to ensure safety
- Punish bad peer behavior without redoing all networking code