# Build, Scale, Operate

## Building a robust bitcoin ecosystem

*Eric Lombrozo & Meltem Demirors*
*Scaling Bitcoin Phase III*

# Our Goals Today

- Share our perspective on challenges in the bitcoin ecosystem and how they evolved
- Offer constructive ideas to begin growing our ecosystem and solicit input
- Leave with some shared goals for our community

Share anonymous comments, ideas, feedback at bit.ly/ScalingIdeas

# The Bitcoin Ecosystem is Fragmented

# Key Stakeholders

## BUILD

- Core developers and contributors
- CTOs / engineers
- Academics
- Business people
- Product leads

## SCALE

- Visionaries and influencers
- Product UI / UX
- Business Development
- Marketing and Sales
- PR / Communications
- Designers

## OPERATE

- Miners
- Exchanges
- Bitcoin 2.0 companies
- Industry collaboration projects / coalitions
- Foundations and other community groups

*We need coordination among stakeholders to keep bitcoin development moving forward*

# Challenges in the Bitcoin Today

| | |
|---|---|
| **Inherent Complexities** | Native to the bitcoin protocol and the dependencies within it - features can’t be solved for by other implementations |
| **Accidental Complexities** | Limitations arising from the manner in which bitcoin has developed over time and the techniques used - real issue is maintaining compatibility and not breaking consensus |
| **Inadequate Methods & Techniques** | Method of development is challenging - hard to gain experience, meaningful contribution requires extremely skilled developers, and many nuances and complexities in dev process |
| **Continuous Re-invention and Re-discovery** | Re-creating incompatible or sub-par solutions to problems that have already been solved, building different products to implement the same services and features |

# A Quick Outline

- 15 min | Eric - Contributing in Bitcoin - Perspectives and Suggestions for Scalable Development
- 15 min | Meltem - Operating in Bitcoin - Connecting the Dots in the Ecosystem
- 5 min | Review of Suggested Projects to Pursue
- 10 min | Q&A / Discussion

# Contributing in Bitcoin

*Scalable Development*

# First Dive into Bitcoin Development

https://en.bitcoin.it/wiki/Original_Bitcoin_client/API_calls_list

# Original Bitcoin client/API calls list

Bitcoin API call list (as of version 0.8.0)

Note: up-to-date API reference can be found here.

**Contents** [hide]



## Common operations

### Listing my bitcoin addresses

Listing the bitcoin addresses in your wallet is easily done via *listreceivedbyaddress*. It normally lists only addresses which already have received transactions, however you can list all the addresses by setting the first argument to 0, and the second one to true.

Accounts are used to organize addresses.

https://en.bitcoin.it/wiki/Original_Bitcoin_client/API_calls_list

## Full list

Required arguments are denoted inside < and > Optional arguments are inside [ and ].

| Command | Parameters | Description | Requires unlocked wallet? (v0.4.0+) |
|---|---|---|---|
| `addmultisigaddress` | <nrequired> <'["key","key"]'> [account] | Add a nrequired-to-sign multisignature address to the wallet. Each key is a bitcoin address or hex-encoded public key. If [account] is specified, assign address to [account]. Returns a string containing the address. | N |
| `addnode` | <node> <add/remove/onetry> | **version 0.8** Attempts add or remove <node> from the addnode list or try a connection to <node> once. | N |
| `backupwallet` | <destination> | Safely copies wallet.dat to destination, which can be a directory or a path with filename. | N |
| `createmultisig` | <nrequired> <'["key,"key"]'> | Creates a multi-signature address and returns a json object | |
| `createrawtransaction` | [{"txid":txid,"vout":n},...] {address:amount,...} | **version 0.7** Creates a raw transaction spending given inputs. | N |
| `decoderawtransaction` | <hex string> | **version 0.7** Produces a human-readable JSON object for a raw transaction. | N |
| `dumpprivkey` | <bitcoinaddress> | Reveals the private key corresponding to <bitcoinaddress> | Y |
| `encryptwallet` | <passphrase> | Encrypts the wallet with <passphrase>. | N |
| `getaccount` | <bitcoinaddress> | Returns the account associated with the given address. | N |
| `getaccountaddress` | <account> | Returns the current bitcoin address for receiving payments to this account. If <account> does not exist, it will be created along with an associated new address that will be returned. | N |

# Getting More Advanced

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

# Protocol documentation

This page *describes* the behavior of the reference client. The Bitcoin protocol is specified by the behavior of the reference client, not by this page. In particular, while this page is quite complete in describing the network protocol, it does not attempt to list all of the rules for block or transaction validity.

Type names used in this documentation are from the C99 standard.

For protocol used in mining, see getblocktemplate.

**Contents** [hide]

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

## Message types

### version

When a node creates an outgoing connection, it will immediately advertise its version. The remote node will respond with its version. No further communication is possible until both peers have exchanged their version.

Payload:

| Field Size | Description | Data type | Comments |
|---|---|---|---|
| 4 | version | int32_t | Identifies protocol version being used by the node |
| 8 | services | uint64_t | bitfield of features to be enabled for this connection |
| 8 | timestamp | int64_t | standard UNIX timestamp in seconds |
| 26 | addr_recv | net_addr | The network address of the node receiving this message |
| Fields below require version ≥ 106 | | | |
| 26 | addr_from | net_addr | The network address of the node emitting this message |
| 8 | nonce | uint64_t | Node random nonce, randomly generated every time a version packet is sent. This nonce is used to detect connections to self. |
| ? | user_agent | var_str | User Agent (0x00 if string is 0 bytes long) |
| 4 | start_height | int32_t | The last block received by the emitting node |
| Fields below require version ≥ 70001 | | | |
| 1 | relay | bool | Whether the remote peer should announce relayed transactions or not, see BIP 0037 |

A "verack" packet shall be sent if the version packet was accepted.

The following services are currently assigned:

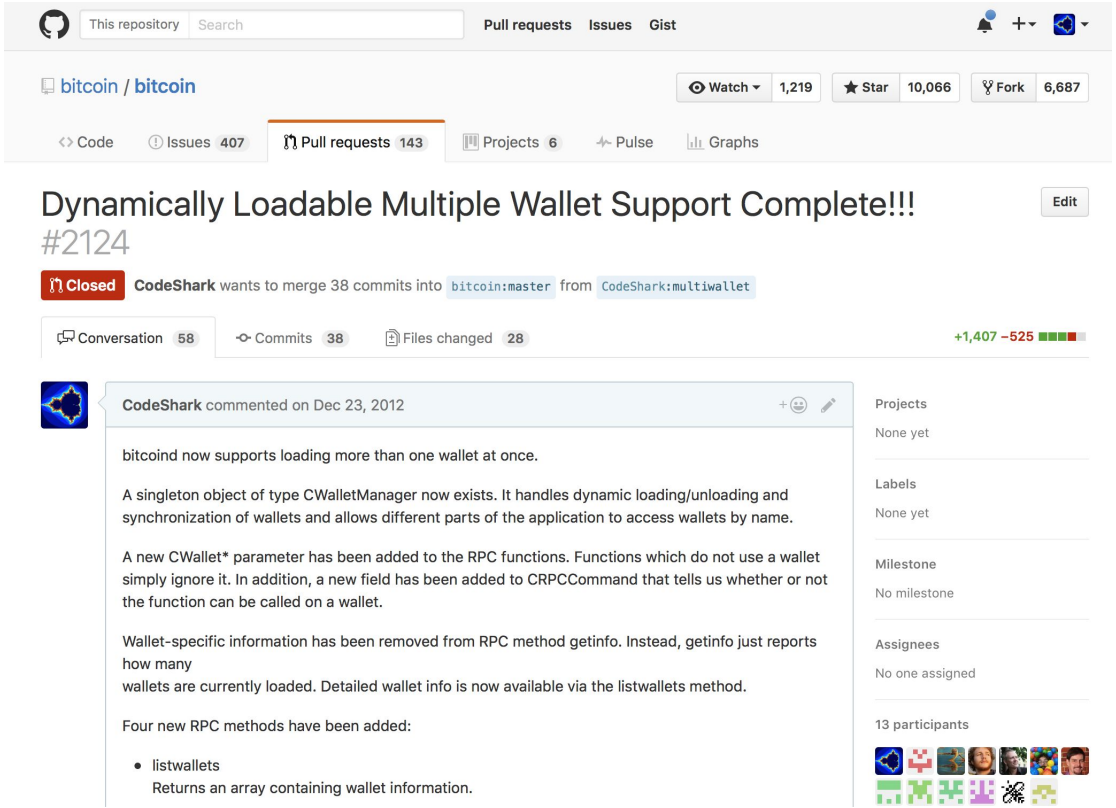
| Value | Name | Description |
|---|---|---|
| 1 | NODE_NETWORK | This node can be asked for full blocks instead of just headers. |

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

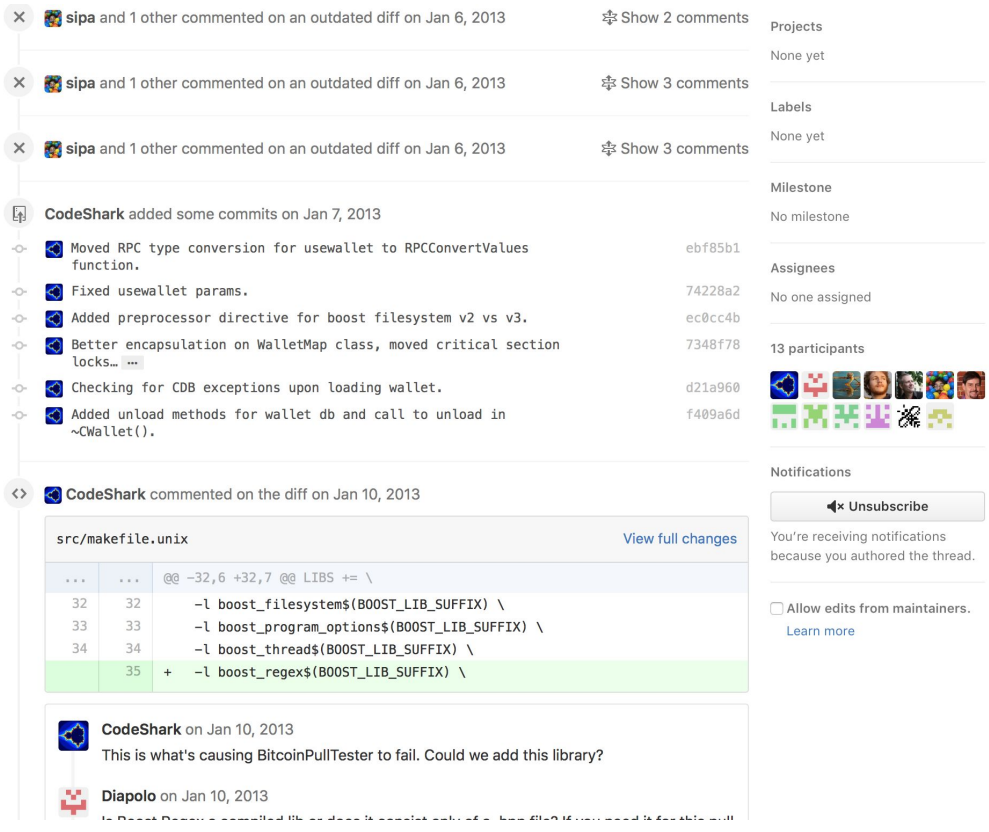
```
0000   f9 be b4 d9 76 65 72 73 69 6f 6e 00 00 00 00 00  ....version.....
0010   64 00 00 00 35 8d 49 32 62 ea 00 00 01 00 00 00  d...5.I2b.......
0020   00 00 00 00 11 b2 d0 50 00 00 00 00 01 00 00 00  .......P........
0030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff  ................
0040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050   00 00 00 00 00 00 00 00 ff ff 00 00 00 00 00 00  ................
0060   3b 2e b3 5d 8c e6 17 65 0f 2f 53 61 74 6f 73 68  ;..]...e./Satosh
0070   69 3a 30 2e 37 2e 32 2f c0 3e 03 00              i:0.7.2/.>..

Message Header:
 F9 BE B4 D9                                                       - Main network magic bytes
 76 65 72 73 69 6F 6E 00 00 00 00 00                               - "version" command
 64 00 00 00                                                       - Payload is 100 bytes long
 3B 64 8D 5A                                                       - payload checksum

Version message:
 62 EA 00 00                                                       - 60002 (protocol version 60002)
 01 00 00 00 00 00 00 00                                           - 1 (NODE_NETWORK services)
 11 B2 D0 50 00 00 00 00                                           - Tue Dec 18 10:12:33 PST 2012
 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 00 00 00 00 00 00 - Recipient address info - see Network Address
 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 00 00 00 00 00 00 - Sender address info - see Network Address
 3B 2E B3 5D 8C E6 17 65                                           - Node ID
 0F 2F 53 61 74 6F 73 68 69 3A 30 2E 37 2E 32 2F                   - "/Satoshi:0.7.2/" sub-version string (string is 15 bytes long)
 C0 3E 03 00                                                       - Last block sending node has is block #212672
```

# Developmental Bottlenecks

# Developmental Bottlenecks



# Developmental Bottlenecks



sipa and 1 other commented on an outdated diff on Jan 6, 2013 — Show 2 comments

sipa and 1 other commented on an outdated diff on Jan 6, 2013 — Show 3 comments

sipa and 1 other commented on an outdated diff on Jan 6, 2013 — Show 3 comments

CodeShark added some commits on Jan 7, 2013

| | |
|---|---|
| Moved RPC type conversion for usewallet to RPCConvertValues function. | ebf85b1 |
| Fixed usewallet params. | 74228a2 |
| Added preprocessor directive for boost filesystem v2 vs v3. | ec0cc4b |
| Better encapsulation on WalletMap class, moved critical section locks… | 7348f78 |
| Checking for CDB exceptions upon loading wallet. | d21a960 |
| Added unload methods for wallet db and call to unload in ~CWallet(). | f409a6d |

CodeShark commented on the diff on Jan 10, 2013

src/makefile.unix — View full changes

```
@@ -32,6 +32,7 @@ LIBS += \
    -l boost_filesystem$(BOOST_LIB_SUFFIX) \
    -l boost_program_options$(BOOST_LIB_SUFFIX) \
    -l boost_thread$(BOOST_LIB_SUFFIX) \
+   -l boost_regex$(BOOST_LIB_SUFFIX) \
```

**CodeShark** on Jan 10, 2013
This is what's causing BitcoinPullTester to fail. Could we add this library?

**Diapolo** on Jan 10, 2013

Projects
None yet

Labels
None yet

Milestone
No milestone

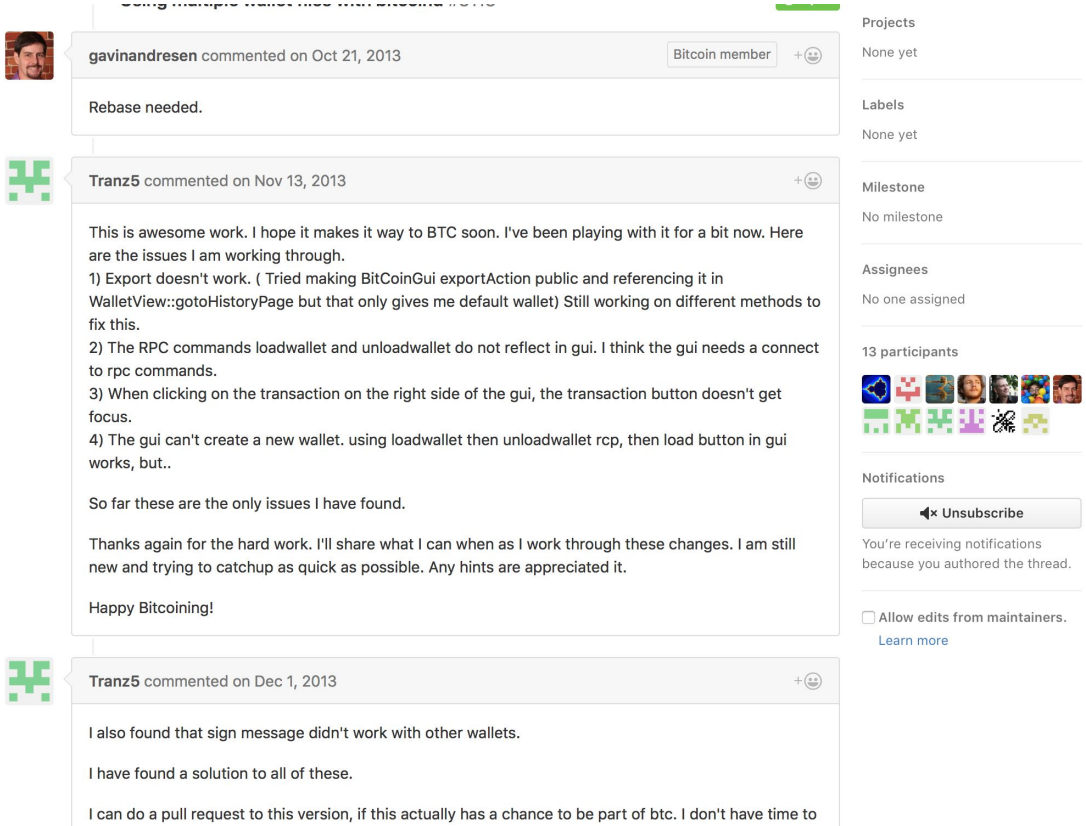
Assignees
No one assigned

13 participants

Notifications
Unsubscribe
You're receiving notifications because you authored the thread.

Allow edits from maintainers.
Learn more

# Developmental Bottlenecks



**gavinandresen** commented on Oct 21, 2013 — Bitcoin member

Rebase needed.

**Tranz5** commented on Nov 13, 2013

This is awesome work. I hope it makes it way to BTC soon. I've been playing with it for a bit now. Here are the issues I am working through.
1) Export doesn't work. ( Tried making BitCoinGui exportAction public and referencing it in WalletView::gotoHistoryPage but that only gives me default wallet) Still working on different methods to fix this.
2) The RPC commands loadwallet and unloadwallet do not reflect in gui. I think the gui needs a connect to rpc commands.
3) When clicking on the transaction on the right side of the gui, the transaction button doesn't get focus.
4) The gui can't create a new wallet. using loadwallet then unloadwallet rcp, then load button in gui works, but..

So far these are the only issues I have found.

Thanks again for the hard work. I'll share what I can when as I work through these changes. I am still new and trying to catchup as quick as possible. Any hints are appreciated it.

Happy Bitcoining!

**Tranz5** commented on Dec 1, 2013

I also found that sign message didn't work with other wallets.

I have found a solution to all of these.

I can do a pull request to this version, if this actually has a chance to be part of btc. I don't have time to

Projects
None yet

Labels
None yet

Milestone
No milestone

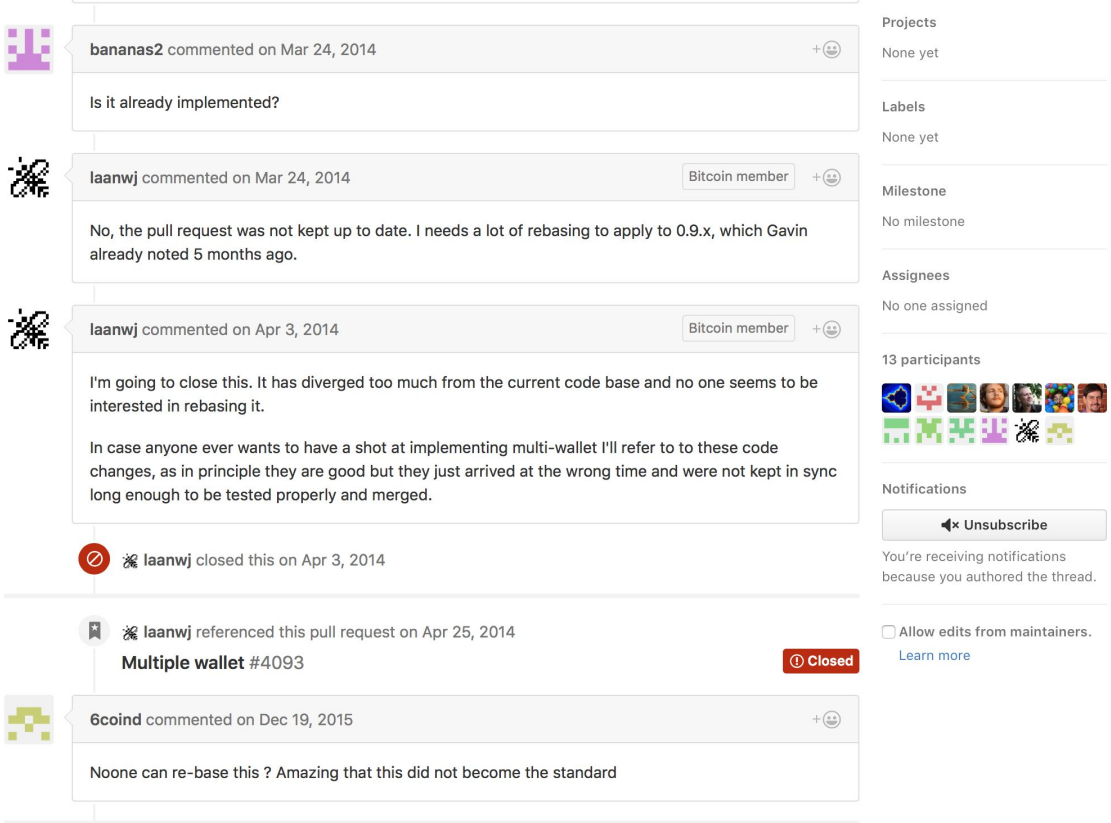
Assignees
No one assigned

13 participants

Notifications
Unsubscribe
You're receiving notifications because you authored the thread.

Allow edits from maintainers.
Learn more

# Developmental Bottlenecks



**bananas2** commented on Mar 24, 2014

Is it already implemented?

**laanwj** commented on Mar 24, 2014 — Bitcoin member

No, the pull request was not kept up to date. I needs a lot of rebasing to apply to 0.9.x, which Gavin already noted 5 months ago.

**laanwj** commented on Apr 3, 2014 — Bitcoin member

I'm going to close this. It has diverged too much from the current code base and no one seems to be interested in rebasing it.

In case anyone ever wants to have a shot at implementing multi-wallet I'll refer to to these code changes, as in principle they are good but they just arrived at the wrong time and were not kept in sync long enough to be tested properly and merged.

**laanwj** closed this on Apr 3, 2014

**laanwj** referenced this pull request on Apr 25, 2014

**Multiple wallet** #4093 — Closed

**6coind** commented on Dec 19, 2015

Noone can re-base this ? Amazing that this did not become the standard

Projects

None yet

Labels

None yet

Milestone

No milestone

Assignees

No one assigned

13 participants

Notifications

Unsubscribe

You're receiving notifications because you authored the thread.

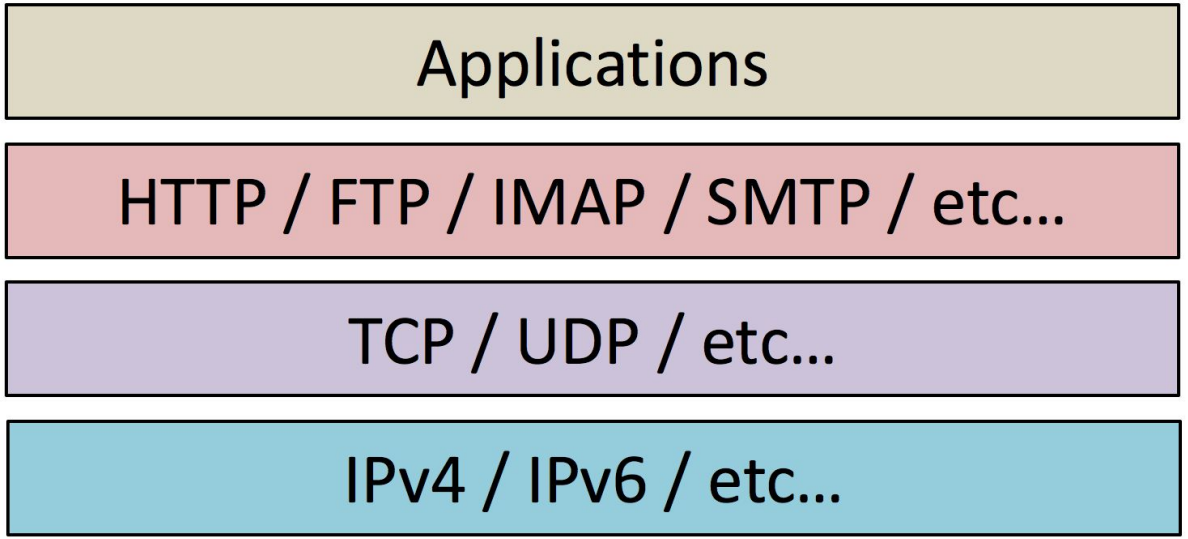
Allow edits from maintainers.

Learn more

# Layered Protocols

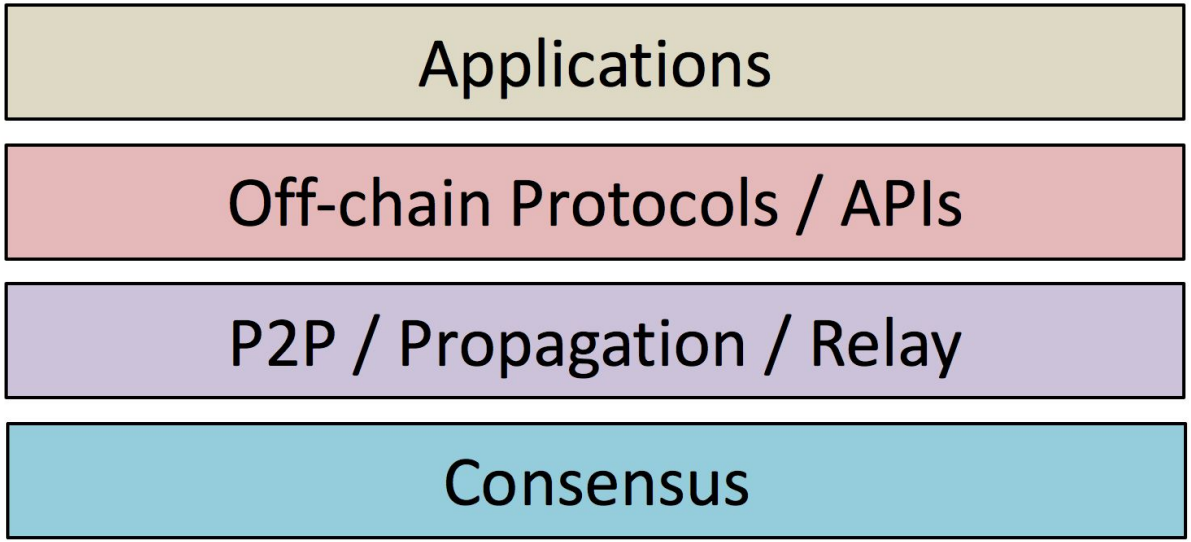
# Layered Protocols

## Internet Protocol Layers



# Layered Protocols

## Bitcoin Protocol Layers



# Layered Protocols

BIP 123

# Layered Protocols

```
BIP: 123
Layer: Process
Title: BIP Classification
Author: Eric Lombrozo <elombrozo@gmail.com>
Status: Draft
Type: Process
Created: 2015-08-26
```

**Table of Contents**



## Abstract


This document describes a classification scheme for BIPs.

BIPs are classified by system layers with lower numbered layers involving more intricate interoperability requirements.

The specification defines the layers and sets forth specific criteria for deciding to which layer a particular standards BIP belongs.

# Layered Protocols

### 4. Applications Layer

The applications layer specifies high level structures, abstractions, and conventions that allow different applications to support similar features and share data.

# Layered Protocols

### 3. API/RPC Layer

The API/RPC layer specifies higher level calls accessible to applications. Support for these BIPs is not required for basic network interoperability but might be expected by some client applications.

There's room at this layer to allow for competing standards without breaking basic network interoperability.

# Layered Protocols

## 2. Peer Services Layer

The peer services layer specifies how nodes find each other and propagate messages.

Only a subset of all specified peer services are required for basic node interoperability. Nodes can support further optional extensions.

It is always possible to add new services without breaking compatibility with existing services, then gradually deprecate older services. In this manner, the entire network can be upgraded without serious risks of service disruption.

# Layered Protocols

## 1. Consensus Layer

The consensus layer defines cryptographic commitment structures. Its purpose is ensuring that anyone can locally evaluate whether a particular state and history is valid, providing settlement guarantees, and assuring eventual convergence.

The consensus layer is not concerned with how messages are propagated on a network.

Disagreements over the consensus layer can result in network partitioning, or forks, where different nodes might end up accepting different incompatible histories. We further subdivide consensus layer changes into soft forks and hard forks.

### Soft Forks

In a soft fork, some structures that were valid under the old rules are no longer valid under the new rules. Structures that were invalid under the old rules continue to be invalid under the new rules.

### Hard Forks

In a hard fork, structures that were invalid under the old rules become valid under the new rules.

# Consensus Rule Changes

# Satoshi's Vision

**satoshi**
Founder
Sr. Member

Activity: 364

**Re: Transactions and Scripts: DUP HASH160 ... EQUALVERIFY CHECKSIG**
June 17, 2010, 06:46:08 PM

#2

The nature of Bitcoin is such that once version 0.1 was released, the core design was set in stone for the rest of its lifetime. Because of that, I wanted to design it to support every possible transaction type I could think of. The problem was, each thing required special support code and data fields whether it was used or not, and only covered one special case at a time. It would have been an explosion of special cases. The solution was script, which generalizes the problem so transacting parties can describe their transaction as a predicate that the node network evaluates. The nodes only need to understand the transaction to the extent of evaluating whether the sender's conditions are met.

The script is actually a predicate. It's just an equation that evaluates to true or false. Predicate is a long and unfamiliar word so I called it script.

The receiver of a payment does a template match on the script. Currently, receivers only accept two templates: direct payment and bitcoin address. Future versions can add templates for more transaction types and nodes running that version or higher will be able to receive them. All versions of nodes in the network can verify and process any new transactions into blocks, even though they may not know how to read them.

The design supports a tremendous variety of possible transaction types that I designed years ago. Escrow transactions, bonded contracts, third party arbitration, multi-party signature, etc. If Bitcoin catches on in a big way, these are things we'll want to explore in the future, but they all had to be designed at the beginning to make sure they would be possible later.

I don't believe a second, compatible implementation of Bitcoin will ever be a good idea. So much of the design depends on all nodes getting exactly identical results in lockstep that a second implementation would be a menace to the network. The MIT license is compatible with all other licenses and commercial uses, so there is no need to rewrite it from a licensing standpoint.

**Source**: Bitcoin talk circa June 2010. https://bitcointalk.org/index.php?topic=195.msg1611#msg1611

**satoshi**
Founder
Sr. Member

Activity: 364

## Re: Transactions and Scripts: DUP HASH160 ... EQUALVERIFY CHECKSIG

June 18, 2010, 04:17:14 PM #5

A second version would be a massive development and maintenance hassle for me. It's hard enough maintaining backward compatibility while upgrading the network without a second version locking things in. If the second version screwed up, the user experience would reflect badly on both, although it would at least reinforce to users the importance of staying with the official version. If someone was getting ready to fork a second version, I would have to air a lot of disclaimers about the risks of using a minority version. This is a design where the majority version wins if there's any disagreement, and that can be pretty ugly for the minority version and I'd rather not go into it, and I don't have to as long as there's only one version.

I know, most developers don't like their software forked, but I have real technical reasons in this case.

> **Quote from: gavinandresen on June 17, 2010, 07:58:14 PM**
>
> I admire the flexibility of the scripts-in-a-transaction scheme, but my evil little mind immediately starts to think of ways I might abuse it. I could encode all sorts of interesting information in the TxOut script, and if non-hacked clients validated-and-then-ignored those transactions it would be a useful covert broadcast communication channel.
>
> That's a cool feature until it gets popular and somebody decides it would be fun to flood the payment network with millions of transactions to transfer the latest Lady Gaga video to all their friends...

That's one of the reasons for transaction fees. There are other things we can do if necessary.

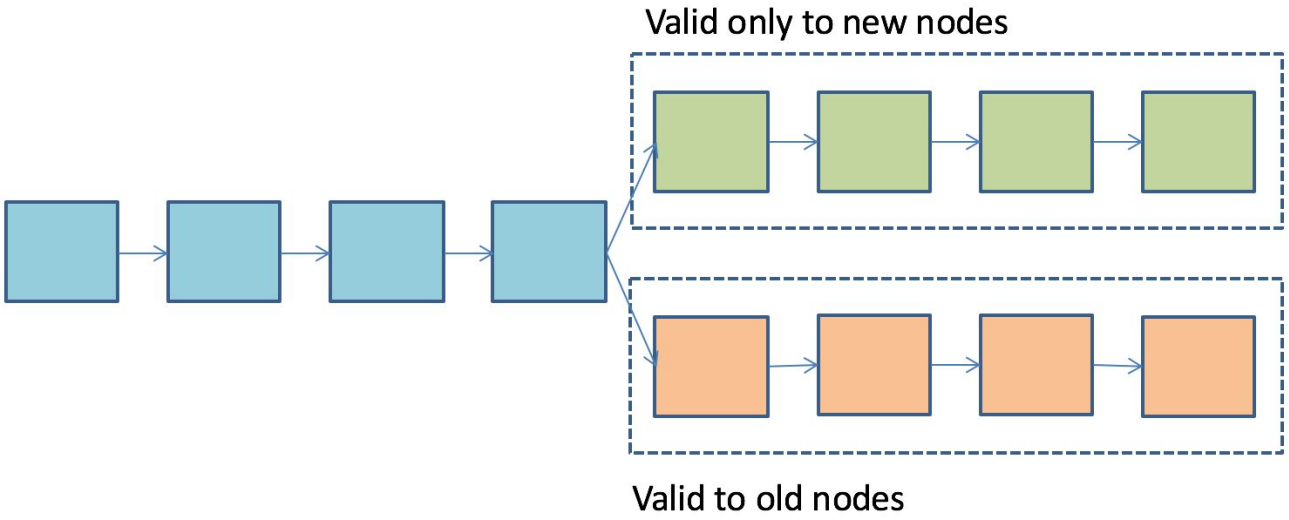
> **Quote from: laszlo on June 17, 2010, 06:50:31 PM**
>
> How long have you been working on this design Satoshi? It seems very well thought out, not the kind of thing you just sit down and code up without doing a lot of brainstorming and discussion on it first. Everyone has the obvious questions looking for holes in it but it is holding up well 😐

Since 2007. At some point I became convinced there was a way to do this without any trust required at all and couldn't resist to keep thinking about it. Much more of the work was designing than coding.

Fortunately, so far all the issues raised have been things I previously considered and planned for.

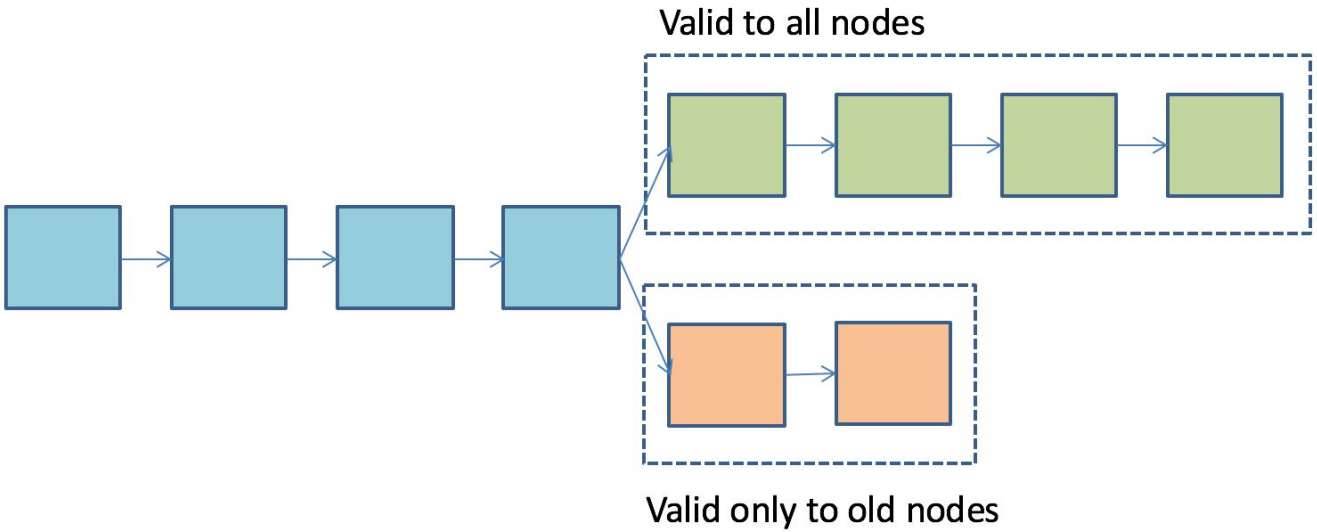
**Source**: Bitcoin talk circa June 2010. https://bitcointalk.org/index.php?topic=195.msg1617#msg1617

# Hard Forks: convergence not guaranteed




Blocks that used to be invalid become valid
(not enforceable by miners)

# Soft Forks: convergence guaranteed




Blocks that used to be valid become invalid
(de facto enforceable by miners)

# Bootstrapping a Global Network

# Key Issues with Building Bitcoin

# Key Issues with Building Bitcoin

- Developmental Bottlenecks

# Key Issues with Building Bitcoin

- Developmental Bottlenecks
- Modularization & Layers

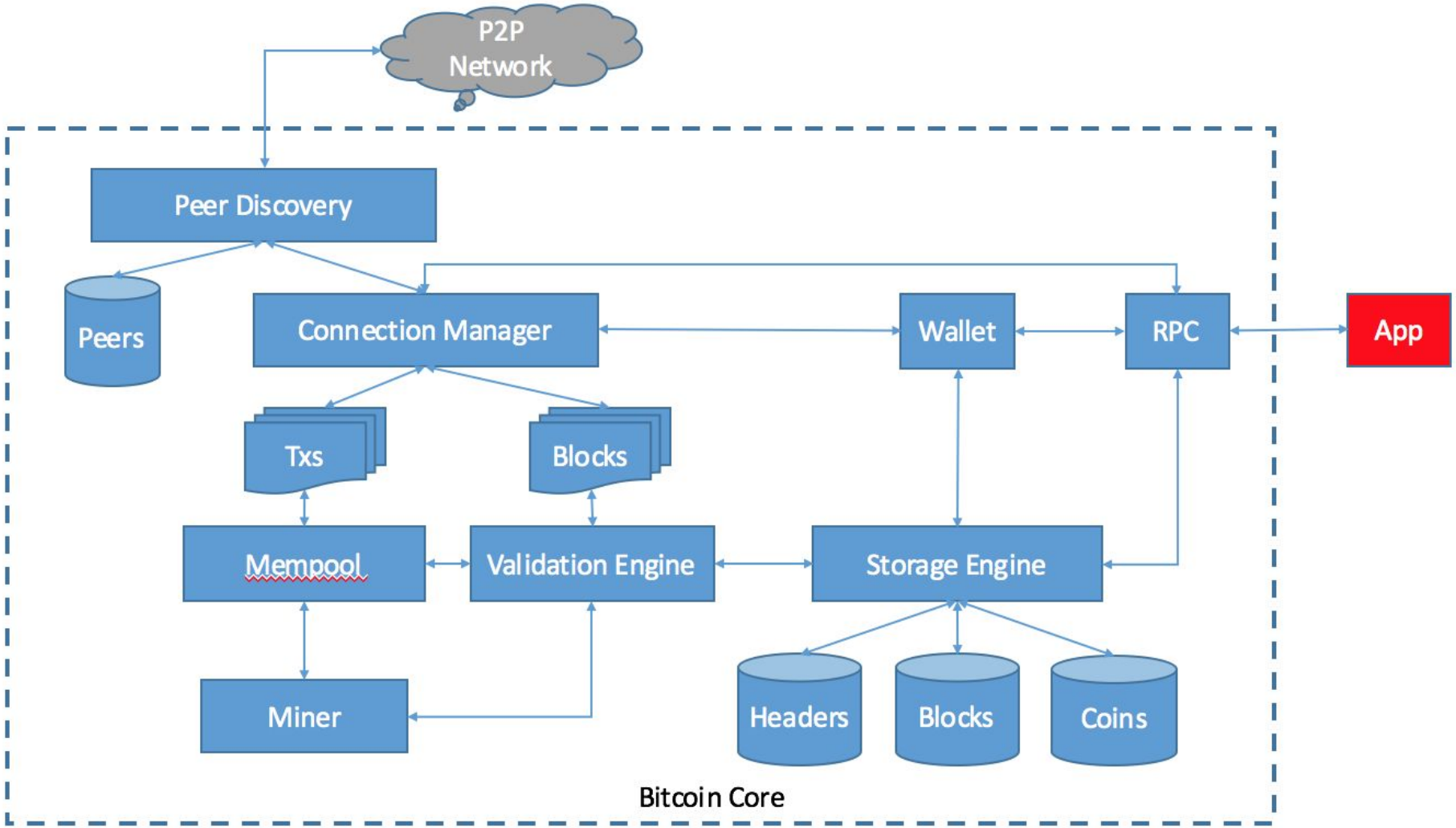
# Key Issues with Building Bitcoin

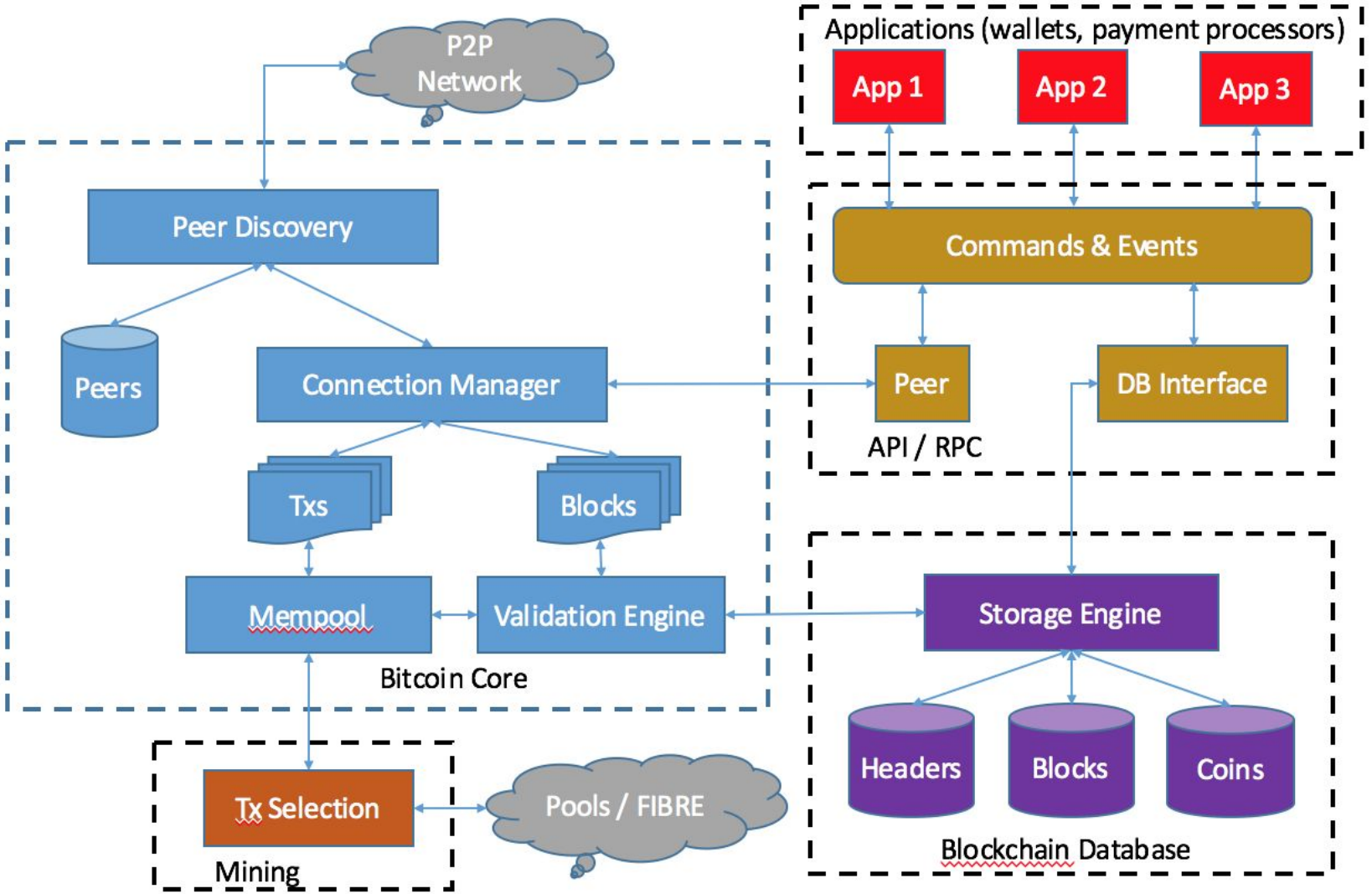
- Developmental Bottlenecks
- Modularization & Layers
- Consensus Rule Changes

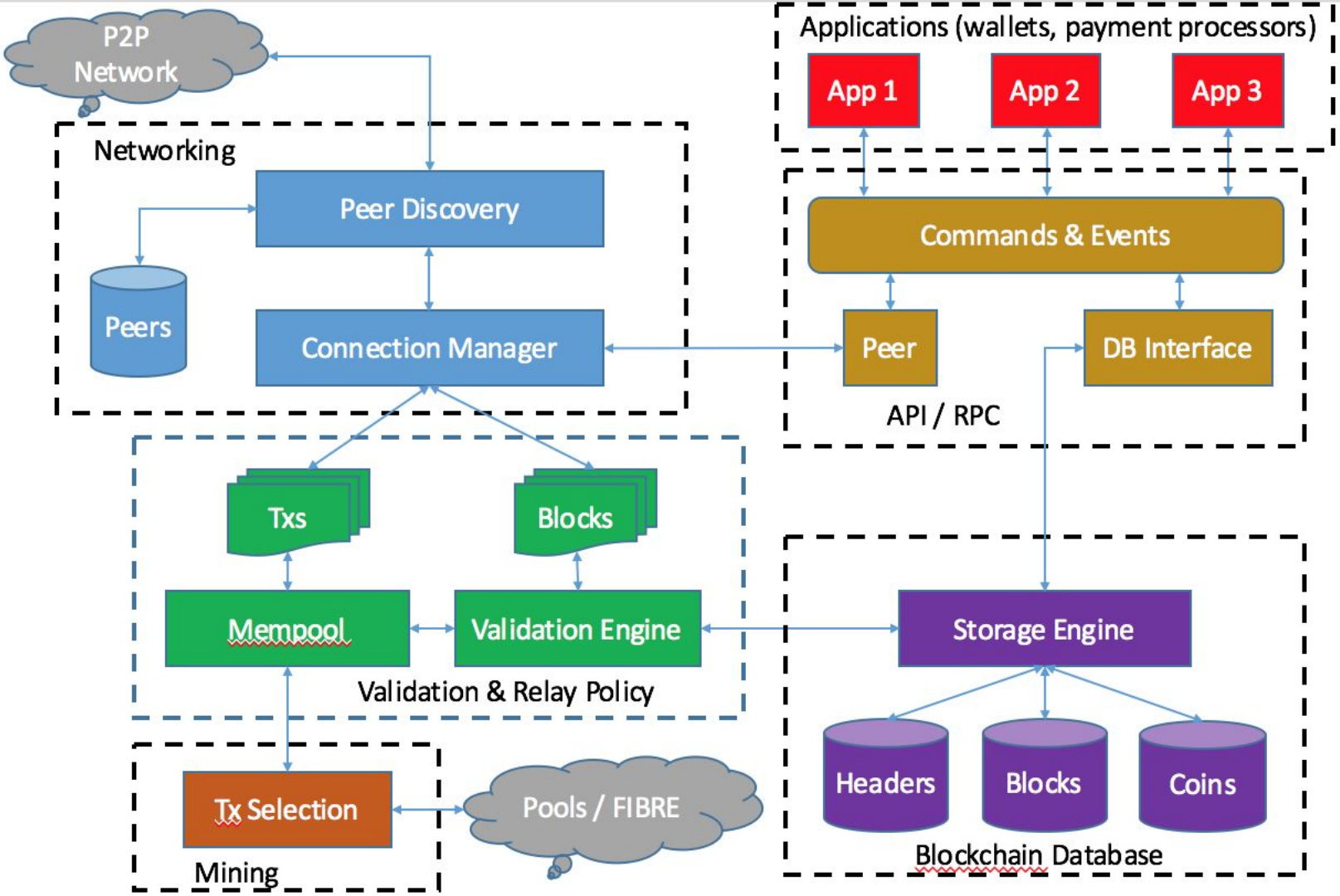
# Key Issues with Building Bitcoin

- Developmental Bottlenecks
- Modularization & Layers
- Consensus Rule Changes
- Bootstrapping a Global Network

# Node Architecture


P2P
Network
Peer Discovery
Peers
Connection Manager
Wallet
RPC
App
Txs
Blocks
Mempool
Validation Engine
Storage Engine
Miner
Headers
Blocks
Coins
Bitcoin Core


P2P Network
Applications (wallets, payment processors)
App 1
App 2
App 3
Peer Discovery
Commands & Events
Peers
Connection Manager
Peer
DB Interface
API / RPC
Txs
Blocks
Mempool
Validation Engine
Storage Engine
Bitcoin Core
Tx Selection
Pools / FIBRE
Headers
Blocks
Coins
Mining
Blockchain Database

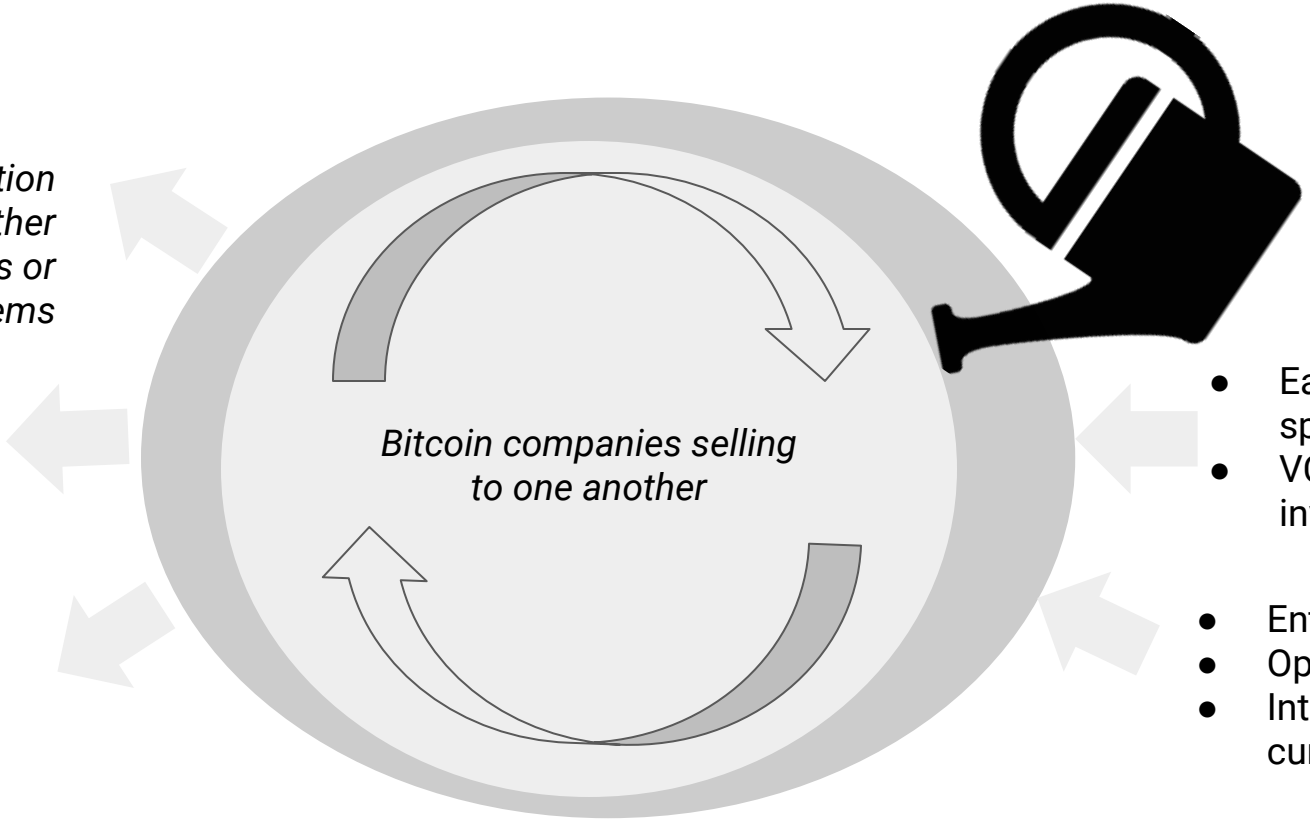

P2P Network
Networking
Peer Discovery
Peers
Connection Manager
Applications (wallets, payment processors)
App 1
App 2
App 3
Commands & Events
Peer
DB Interface
API / RPC
Txs
Blocks
Mempool
Validation Engine
Validation & Relay Policy
Storage Engine
Headers
Blocks
Coins
Blockchain Database
Tx Selection
Pools / FIBRE
Mining

# Operating in Bitcoin

*Reducing entropy*

# Bitcoin Today

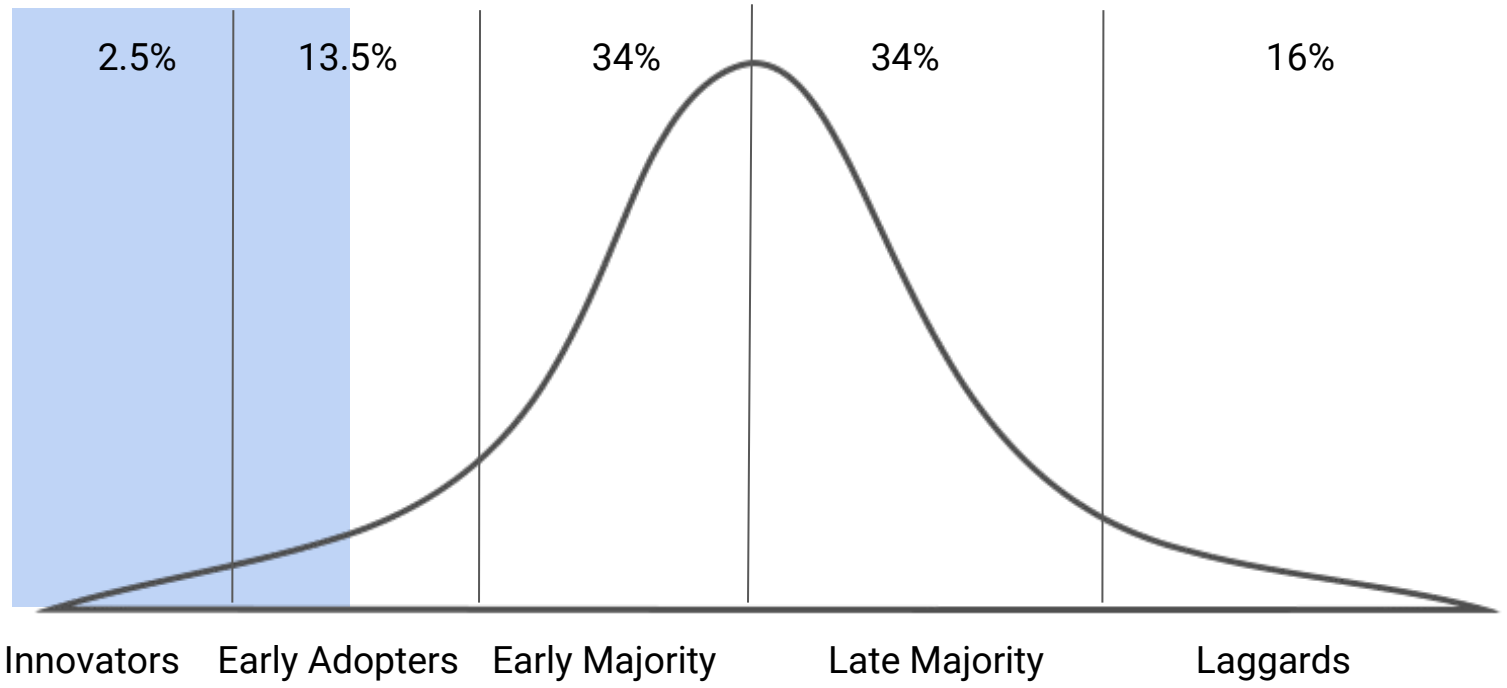
*Slow adoption in other communities or ecosystems*




- Early adopters and speculators
- VC and institutional investment

- Entrepreneurs
- Opportunists
- Intellectually curious people

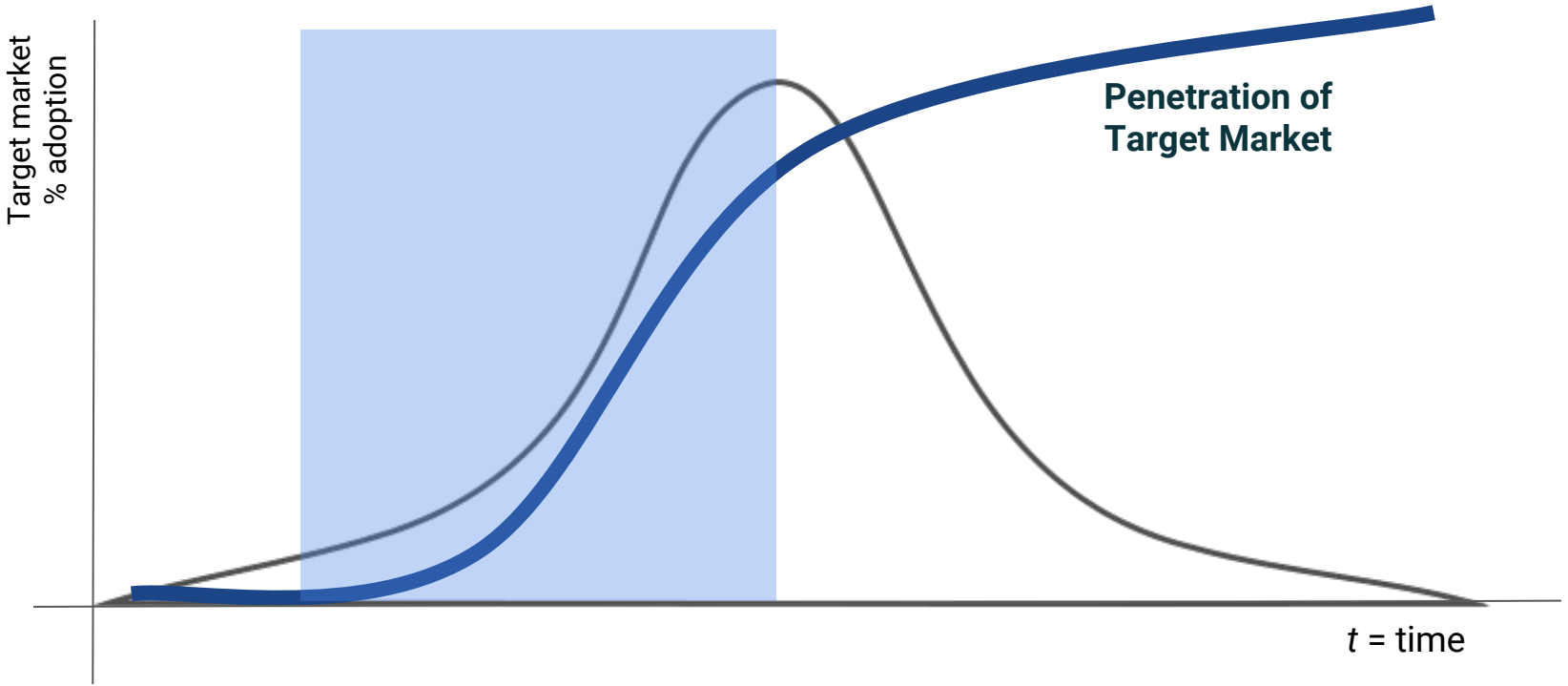
# Who is Bitcoin for?



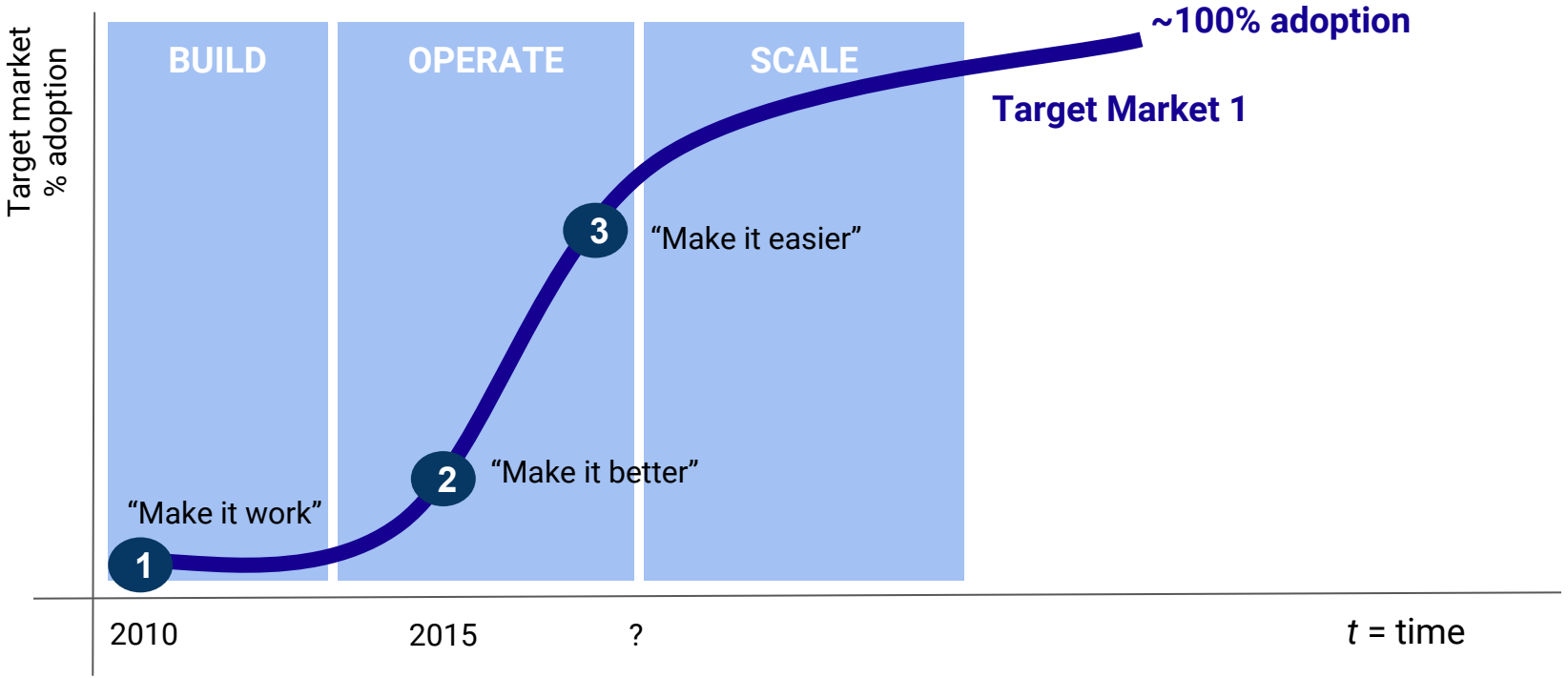
# Stakeholders in Bitcoin Today

| | Ability to Influence | Level of Interest |
|---|---|---|
| Bitcoin Core / Dev | | |
| Miners | | |
| Industry / Exchanges | | |
| Industry / Wallets & Secuity | | |
| Corporates | | |
| Academics | | |
| Policymakers | | |
| HOLDERS / observers | | |
| General Public | | |

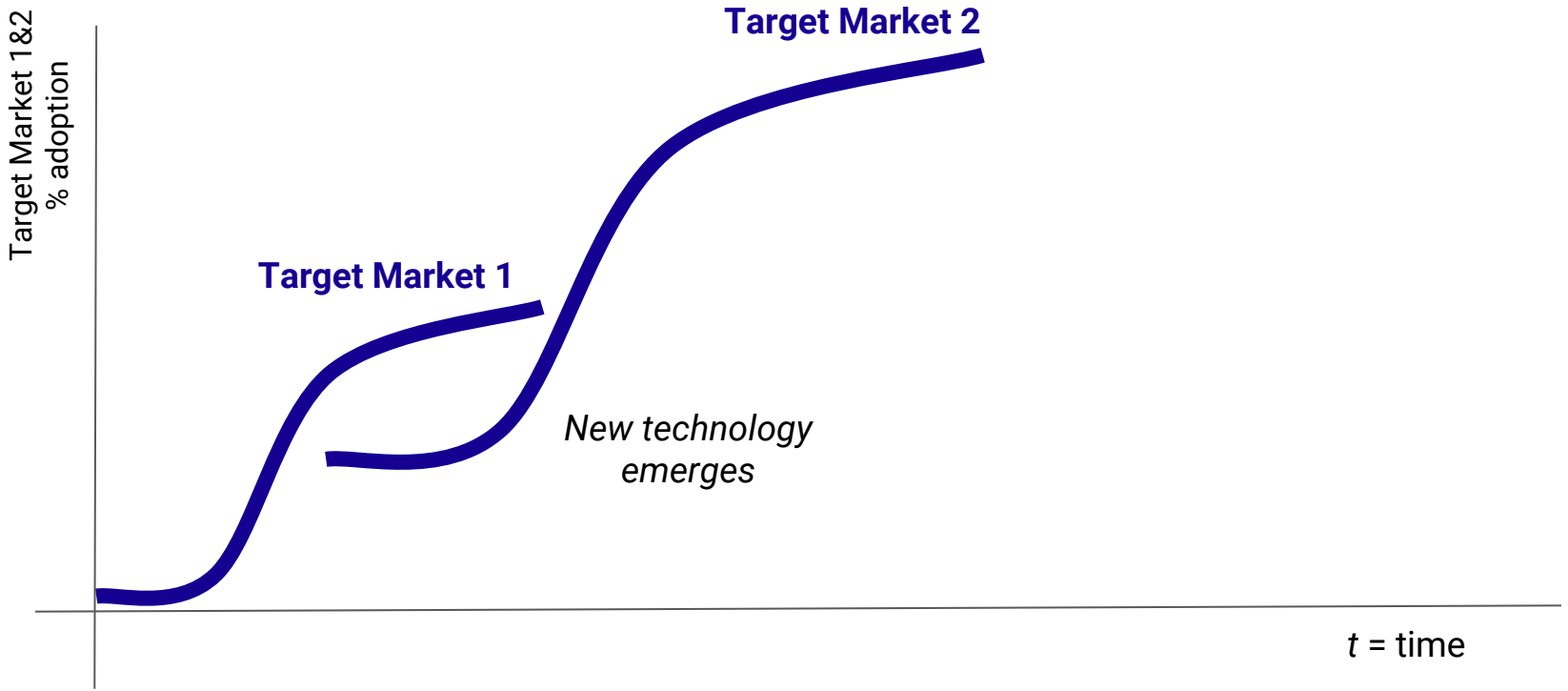
# Why it Matters



# Why it Matters



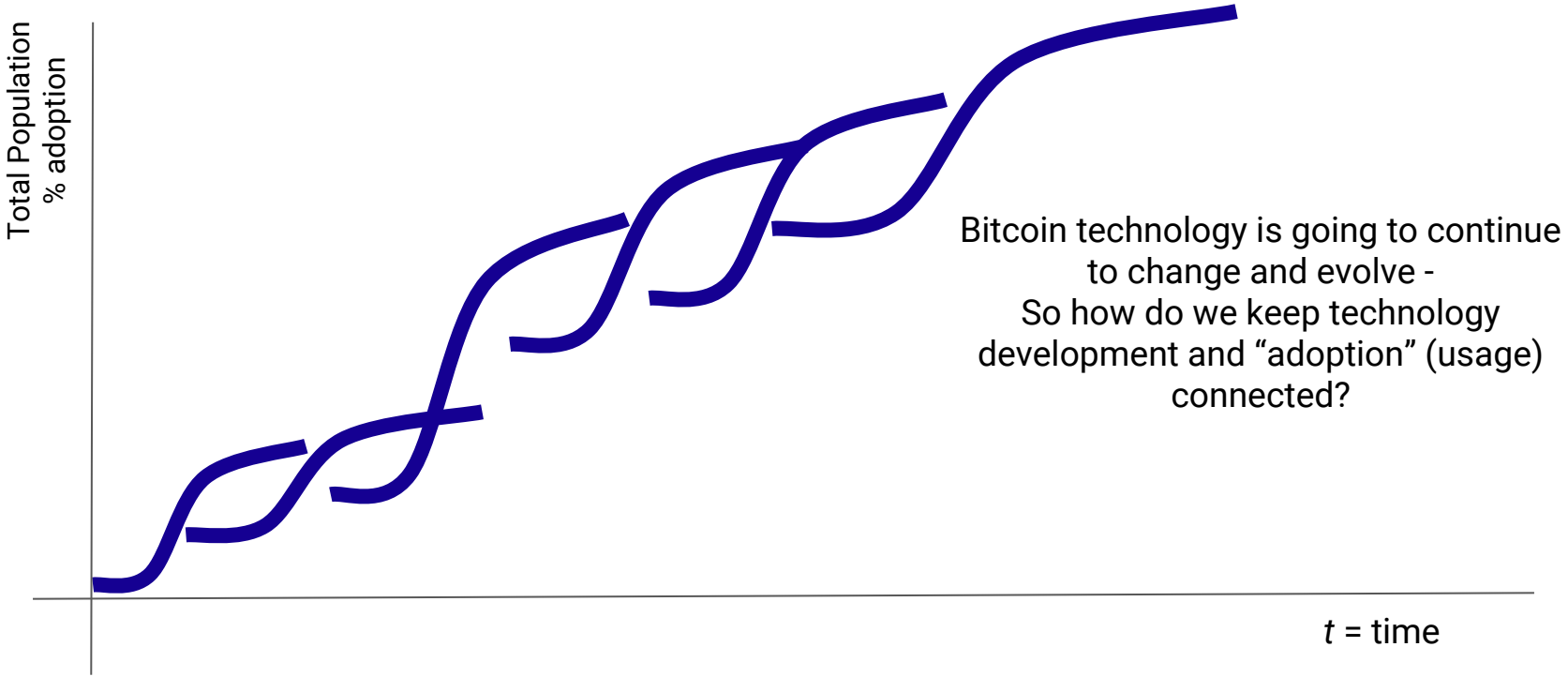
# Why it Matters



# Stakeholders in Bitcoin Tomorrow

| | Ability to Influence | Level of Interest |
|---|---|---|
| Bitcoin Core / Dev | | |
| Miners | | |
| Industry / Exchanges | | |
| Industry / Wallets & Secuity | | |
| Corporates | | |
| Academics | | |
| Policymakers | | |
| HOLDERS / observers | | |
| General Public | | |

# Why it Matters




Bitcoin technology is going to continue to change and evolve -
So how do we keep technology development and “adoption” (usage) connected?

# The Bitcoin Ecosystem is Fragmented

# Are we Our Own Worst Enemy?

Bitcoin is inherently social software - how do groups work?

1. Paradox of Groups - “This is good and must be protected”
2. External enemies as a unifying cause - group cohesion in common cause
3. Religious veneration - nomination and worship of an icon

- We built the system, assumed certain user behaviors
- Users came on and exhibit different behaviors
- People running the system realize technological and social issues can’t *in fact* be uncoupled
- But the conversational context of bitcoin development doesn’t scale

So how could bitcoin scale beyond the “group within the group”?

# Beyond “the group within the group”

| | |
|---|---|
| **Knowledge** | Exposure to or awareness of - knowledge is typically distributed through both formal and informal channels |
| **Attitudes** | The way people view bitcoin has been fairly negative, primarily due to the lack of knowledge and adoption |
| **Adoption** | The decision about whether or not to adopt bitcoin - here features, design, and user experience are key factors |
| **Implementation** | The ability to actually “implement” bitcoin and build bitcoin into other systems or products |
| **Confirmation** | Comparing and evaluating bitcoin against other technologies and determining if it is actually an elegant or efficient solution |

# Attitudes

- Misunderstanding of actual features of bitcoin - misinformation
- Limited body of knowledge around non-technical areas of bitcoin
- Datasets to enable economic modeling / impact assessment for bitcoin adoption difficult to find, limited data available
- Contributing to bitcoin knowledge isn’t required to participate

# Adoption

- Feature set of bitcoin as implemented today limits use cases
- Bitcoin core  development efforts working to resolve these challenges but limited resources and many dependencies
- Limited feedback loops on feature set

# Implementation

- Implementing bitcoin or building with bitcoin is really difficult
- Hard to find talent who can actually build because implementing requires you to “get under the hood”
- Difficulty understanding how components of bitcoin infrastructure interact with one another and what it means for building bitcoin applications

# What Next?

*Community Project Ideas*

# Project Ideas

*Bitcoin Core*

1. Separate dependencies like consensus, database, networking, etc and de-couple into separate units
2. Design interfaces to decouple them into separate units
3. Refactor / reimplement units

*Bitcoin Applications*

4. Expanding research and body of knowledge around non-technical bitcoin issues and opportunities
5. Semi-annual infrastructure workshops for exchanges, wallets, etc.  to manage technical projects more broadly across community
6. Build reference  architecture for how bitcoin fits into enterprise / app infrastructure to ease “adoption” pains

Share anonymous comments, ideas, feedback at bit.ly/ScalingIdeas

# Technical Challenges

- *Points*

# Process and Workflow

- *Points*

# Suggestions

- *Points*

# Validation Engine (libconsensus)

Bitcoin Core Validation Logic

Commit: 2a0836f6d5e7c1d7e97bedb0e0ea33dcaf981f77

```
main.cpp:3732 ProcessNewBlock
        L main.cpp:3665 AcceptBlock
        |       L main.cpp:3617 AcceptBlockHeader
        |       |       L main.cpp:3359 CheckBlockHeader
        |       |       |       L pow.cpp:77 CheckProofOfWork
        |       |       |
        |       |       L Check that we have previous block and it is valid
        |       |       L main.cpp:3431 CheckIndexAgainstCheckpoint
        |       |       L main.cpp:3510 ContextualCheckBlockHeader
        |       |       |       L Check nBits, timestamp, and block version
        |       |       |
        |       |       L main.cpp:3188 AddToBlockIndex
        |       |
```

# Validation Engine

```
|       L main.cpp:3368 CheckBlock
|       |       L main.cpp:3359 CheckBlockHeader (again)
|       |       |       L pow.cpp:77 CheckProofOfWork
|       |       |
|       |       L Check merkle root and that merkle tree is not mutated
|       |       L Check size limits
|       |       L Check first and only first transaction is coinbase
|       |       L FOREACH transaction
|       |       |       L main.cpp:1046 CheckTransaction
|       |       |               L Check vin is not empty
|       |       |               L Check vout is not empty
|       |       |               L Check tx size limit without witness
|       |       |               L Check for negative or overflow output values
|       |       |               L Check for duplicate inputs
|       |       |               L IF is coinbase?
|       |       |                       L THEN: Check coinbase script length
|       |       |                       L ELSE: Check that input prevouts are not null
```