BC^2

Jeremy Rubin

概要
Overview

- 3つの課題
  1. Bitcoin 開発プロセスの紹介
  2. Bitcoin Improvement Proposals (BIP) の紹介
  3. Bitcoin 性能の開発

- 3 Major Sections
  1. Intro Bitcoin Development Process
  2. Intro to Bitcoin Improvement Proposals
  3. Bitcoin Performance Engineering

Bitcoin 開発プロセスの紹介

Intro to Bitcoin Development

Bitcoin デベロッパーになるのか

So You Want to Be a Bitcoin Developer

- 基礎
- 開発環境
- はじめての Bitcoin コード
- 貢献
- 基本アドバイス

- Foundations
- Development Environment
- Beginning to Code Bitcoin
- Contributing
- General Advice

基礎
FOUNDATIONS

開発の考え方を理解する
Understand Development Philosophy

- 様々な Bitcoinユーザをリスペクトする
- 自分の問題は自分で解決する
- Bitcoin の利用は言論の自由
- ゆっくりと着実な
- **同じプロジェクトでもそれぞれの目的は違うかも**

- Respect all kinds of Bitcoin users
- Scratch your own itch
- Bitcoin use is free speech
- Slow and steady
- **Not everyone has same goal for project**

コミュニケーション
Communications

- おそらく誰かが同じ質問を聞いた
- 他のデベロッパーは現在何をディスカッションしているのかに気を使う
  - いろんなオンラインフォーラムは Bitcoin 開発 HQと思われる☺

- Someone already probably asked your question online
- Be respectful of what other developers are working on or currently discussing
  - The various forums are like Bitcoin Dev HQ ☺

コミュニケーション

Communications

- bitcoincore.slack.com に参加する
- IRC チャネル
  - #bitcoin, #bitcoin-wizards, #bitcoin-core-dev
  - 週1回の会議は #bitcoin-core-dev にて
- github.com/bitcoin/bitcoin をフォローする
- Linux Foundation のメーリングリストに参加する
  - bitcoin-core-dev, bitcoin-dev, bitcoin-discuss
- Bitcoin StackExchange

- Join bitcoincore.slack.com
- IRC channels
  - #bitcoin, #bitcoin-wizards, #bitcoin-core-dev
  - Weekly meeting in #bitcoin-core-dev
- Follow github.com/bitcoin/bitcoin
- Join Linux Foundation mailing lists
  - bitcoin-core-dev, bitcoin-dev, bitcoin-discuss
- Bitcoin StackExchange

開発環境
DEVELOPMENT ENVIRONMENT

フォーク、クローン、ビルド
Fork, Clone, Build

- `git clone git@github.com:bitcoin/bitcoin.git`
  – それとも自分のフォークでやる!
- `git checkout –b my-devel-branch`
- ビルド手順通りにビルドする（1回目は時間がかかる）

- `git clone git@github.com:bitcoin/bitcoin.git`
  – or your own fork!
- `git checkout –b my-devel-branch`
- Follow build instructions (first build is slow)

Bitcoin Node を実行する

Run Bitcoin Nodes

- 先ほどコンパイルしたバイナリーをコピーする
  - src/bitcoind and src/bitcoin-cli
- `./bitcoind -debug=bench` はノードを実行する、`./bitcoin-cli` を使ってテストする
- Testnet ノード: `./bitcoind -testnet -debug=bench`, `./bitcoin-cli -testnet`
- 複数のサーバが持ってれば便利！

- Copy the binaries you just compiled
  - src/bitcoind and src/bitcoin-cli
- `./bitcoind -debug=bench` will run a node, use `./bitcoin-cli` to test it
- Testnet node: `./bitcoind -testnet -debug=bench`, `./bitcoin-cli -testnet`
- Useful if you have a few servers to develop on!

はじめてのBITCOIN開発
BEGINNING TO CODE BITCOIN

良いの“悪い考え”を選ぶ

Pick a Good “Bad Idea”

- 私のはじめてのプロジェクト
  - DoSを防ぐためにDBのエントリーをランダム化する
  - 概念実証を応用する
  - デベロッパーからフィードバックを聞く
    - 悪いこと
      - いつかDBは反復が出来る順番は欲しいかも
      - 悪いアクセスのパターンを証明できない
    - 良いこと
      - 勉強の経験

- My First Project
  - Randomize order of a databases entries to prevent DoS on bad DB access pattern
  - Implemented a proof of concept
  - Asked a developer for feedback
    - Bad
      - We may eventually want DB iterable in order
      - Couldn’t demonstrate an example of bad access pattern
    - Good
      - Learning experience

C++技術を高める

Build C++ Expertise

- BitcoinはC++11で書かれた
- 始めながらC++11をよく勉強する（最初の25章ぐらい、1日1~2章）
- cppreference.com!
- C++：読むは“簡単”、書くは“難しい”

- Bitcoin is written in C++11
- Learn C++11 well as you start (first ~25 chapters or so, a chapter or two a day)
- cppreference.com!
- C++: “easy” to read, “hard” to write

ctags を使う
Using ctags

- Bitcoin のコードは多い! ctags で早めに見れる
- Vim ですが、Emacs/Atom 同じことがある!
  - src dir 中に `ctags –R .` で generate する
  - `C-]` で定義に飛んで行く、複数のところであれば `:ts` で選択する
  - `C-t` で前のところへ行く

- Bitcoin has a lot of code! ctags helps you browse it quickly
- Vim specific, but Emacs/Atom has equivalent!
  - `ctags –R .` in src dir to generate
  - Hit `C-]` in vim to jump to definition, `:ts` to select if there are multiple possible locations
  - Hit `C-t` to go back to prior location

Bitcoinをテストする

Testing Bitcoin

- `make check` でユニットテストを実行する
- `./qa/pull-tester/rpc-tests.py` でRPCテストを実行する
- Travis CI を設定すると自動的にやっている
- 新しいテストを書くことと古いテストを読むことでコードの理解を深める！
  - Bitcoinのテスト網羅度は良くないけど、良くなっている！
  - 単調な作業じゃないよ！

- `make check` runs unit tests
- `./qa/pull-tester/rpc-tests.py` to run rpc tests
- Enable Travis CI on your fork for testing to happen automatically
- Writing new tests & reading old ones is a great way to ensure you understand how the code works!
  - Bitcoin test coverage is not great, but getting better!
  - It’s not grunt work!

gdb/lldb を使う
Use gdb/lldb

- 短いチュートリアル
- 基本的に
  - 1段ずつコードを見る
  - バグを探す
  - 実行しているプログラムを検討する
- 注意 : コンパイラーがだますことがある！

- Brief Tutorial
- Basically
  - See what your code is doing step-by-step
  - Find bugs
  - Inspect running programs
- Warning: Compiler can play tricks on you!

`g++ -std=c++11 -g main.cpp`をコンパイルする

Compile `g++ -std=c++11 -g main.cpp`

```
1 void fn(int& x) {
2     ++x;
3 }
4 int main() {
5     int x = 0;
6     fn(x);
7 }
```

これを実行する：
Now Run:

1. gdb --args a.out
   - starts gdb running with a.out
2. b fn
   - sets a breakpoint at fn
3. r
   - runs ./a.out

gdb ノート
gdb cheatsheet

- up/down
  - jump up/down function callstack
- continue
  - resume executing code till next breakpoint
- step
  - take one step through the program
- b *function*
  - breakpoint at function
- r
  - start executing
- b *file:lineno*
  - *breakpoint at location*
- p <x>
  - prints out the variable x
- info thread
  - lists running threads
- thread <n>
  - switches debugger to thread n
- C-X C-a
  - Shows source code

他のコードをレビューする

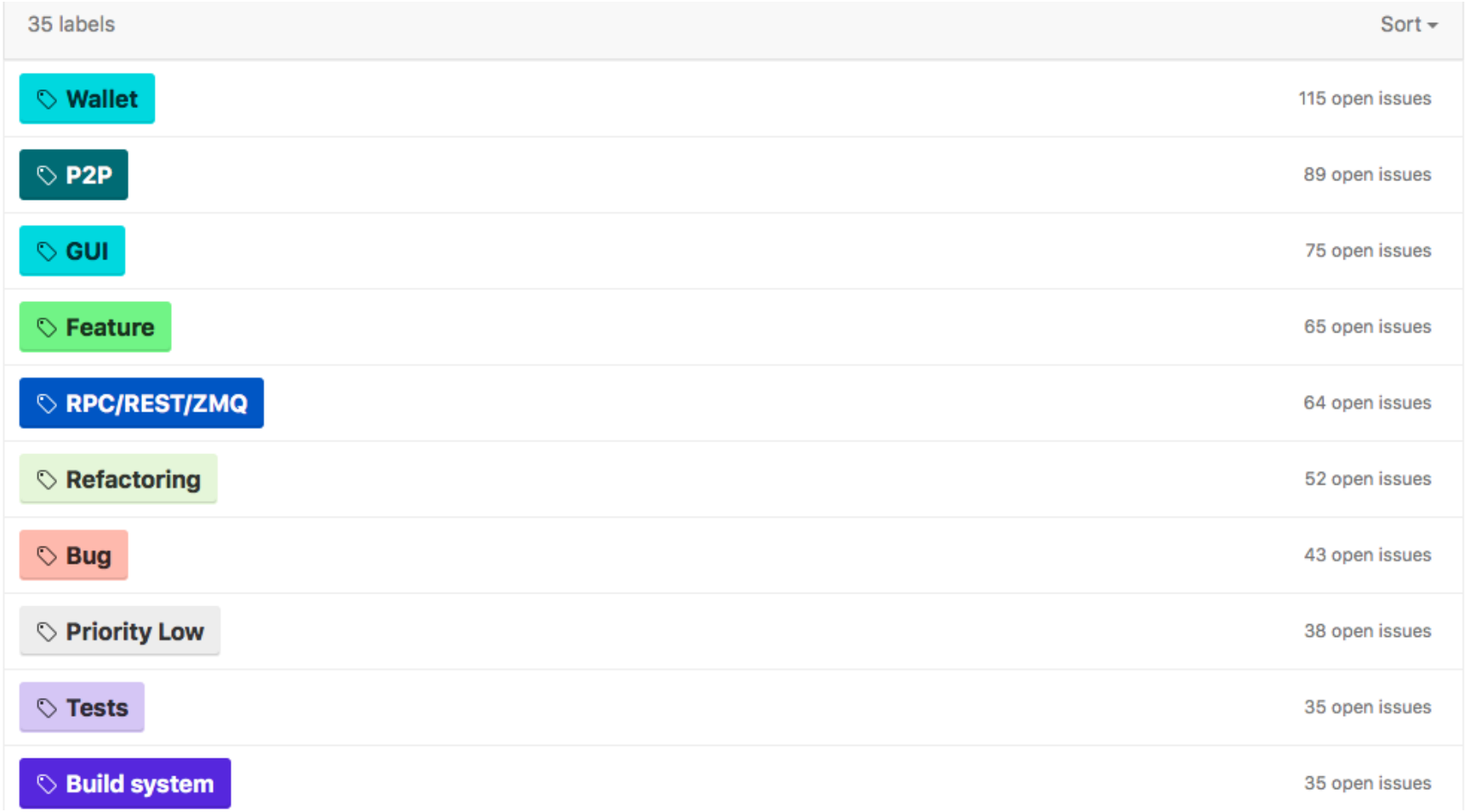
Review Others' Code

- github.com/bitcoin/bitcoin/pulls を見る
- 勉強になること:
  - 他の人は何の課題に作業するか
  - どんな方法でフィードバックをあげる
  - どんなフィードバックをもらえる
- 役に立てるフィードバックをあげれば、もらう人はハッピーになる

- Look through github.com/bitcoin/bitcoin/pulls
- You'll learn
  - What *topics* people are working on
  - *How* people communicate feedback
  - What *kinds* of feedback people get
- If you leave *useful* feedback for someone, **they will be happy**

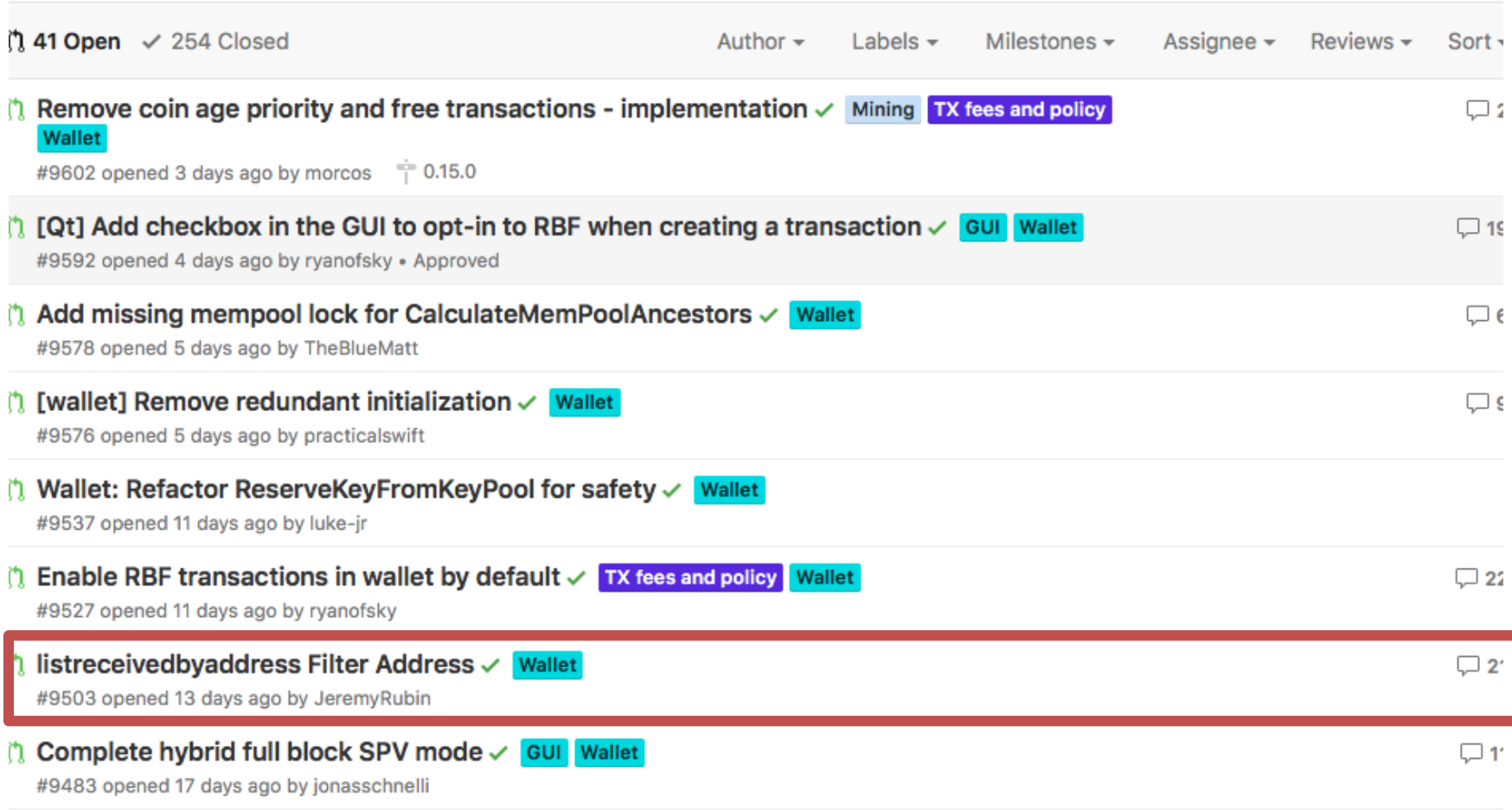
何が盛ん出るのかを見てみましょう

Let’s See What’s Hot



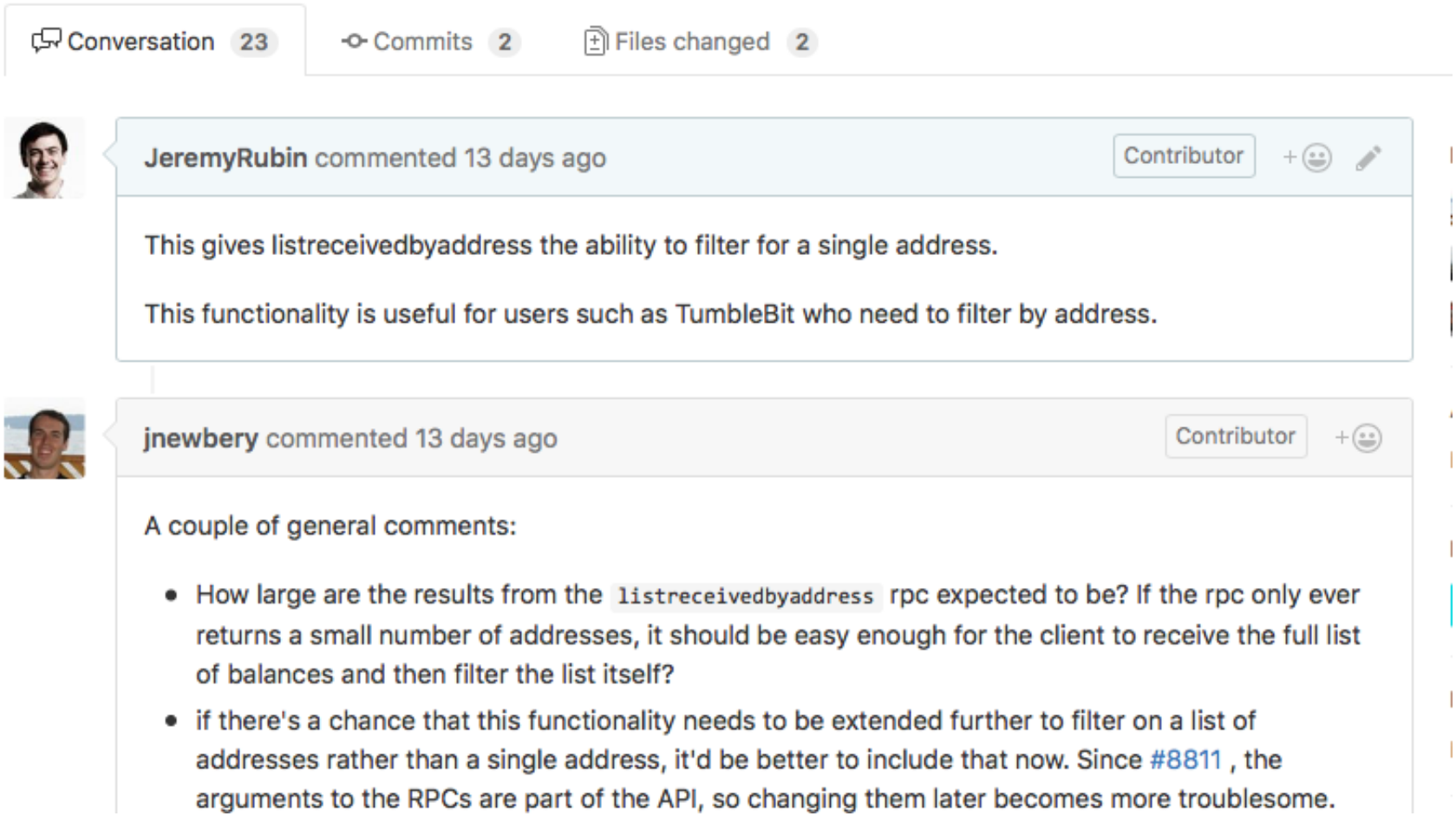
これは私の一つの例

Here's One of Mine



何が起きたかを見ましょう

Let’s See What Happened



何が起きたかを見ましょう

Let’s See What Happened

I think (2) isn't really needed since if you need more than a small number you can just do un-filtered or a few repeated calls. I'm no expert on this use-case though.

JeremyRubin commented 20 days ago Contributor

@EthanHeilman thoughts?

Another optimization would be to allow for caching of this table on construction (maybe keep_cache/clear_cache parameters). This could reduce the O(n*m) complexity for making m repeated calls to O(n + m).

NicolasDorier commented 20 days ago • edited Member

@JeremyRubin The reason why I am interested into that is that here is the code I am using for querying the transactions of a scriptPubKey: Using listtransactions in tumblebit.

貢献
CONTRIBUTING

Bitcoinへの大事な貢献
Important Contributions to Bitcoin

- ドキュメンテーション
- 斬新なアイデアと研究
- 他の人のコードとアイデアのレビュー
- 綿密なテスト
- カンフェレンス / コミュニティの運営
- デベロッパー向けのツール
- **新しいコードを書く!**

- Documentation
- Novel ideas and research
- Review others' code & ideas
- Rigorous Testing
- Conference/Community organizing
- Tools for developers
- **Write new code!**

良いコードを書く

Write Good Code

1. 大事だと思っている課題を探す
2. 書く:
    1. 問題を解決できるパッチ
    2. 明確なドキュメンテーション
    3. コードを含まれるテスト
3. 自分のフォークのブランチにプッシュする

1. Find an issue that you think is important
2. Write
    1. Patches that you think solves it
    2. Clear documentation
    3. Tests that cover the code
3. Push to a branch on your fork

早い段階でフィードバックを探す
Seek Early Feedback

- この人にメッセージを送る:
  - 似たようなことをやっているBitcoinコントリビュータ
  - TheBlueMatt, theuni, jonasschnelli, と私は優しい窓口
  - #bitcoin-core-dev にフィードバックを依頼する
- 心を広げましょう！作業したものの批判はあなた自体の批判じゃない！

- Write a message to:
  - A Bitcoin contributor who works on similar things
  - TheBlueMatt, theuni, jonasschnelli, and myself are friendly default contacts
  - ping #bitcoin-core-dev with a request for feedback
- Be gracious! Negativity on your work is not negativity to you!

新しいノードを実行する
Run New Nodes

- メインとテストネット
  - 違うサーバを使う
  - デフォルトノードのデバッグログを比較して見て。あなたのバージョンの方が良い？正しい？

- Main and test net
  - Recommend using a different server
  - Compare debug logs to compare to your default nodes. Is your version better/correct?

コード変更の再構築
Restructuring Code Changes

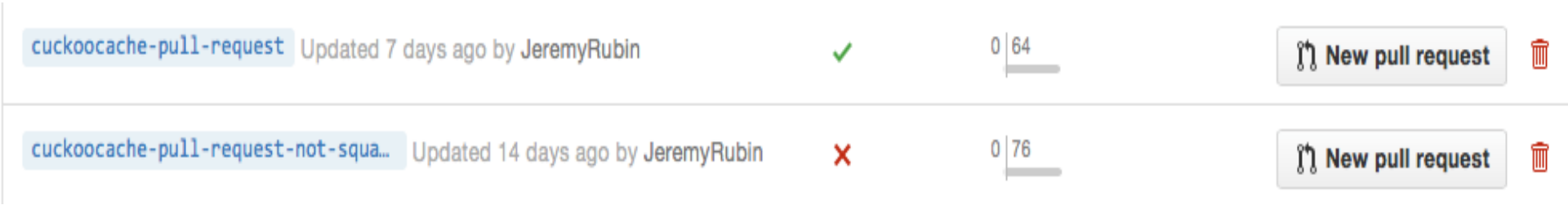
- テストは別のコミットにしましょう
  - ‘Tests after Code’ か ‘Code after Tests’
- 意味をなす小さいコミットにしましょう
  - 時々いらないコードを書きなおすべきになる
- 1つのpull-requestに入れすぎないように
  - 10x 改善のコードは1つの2xプラス後の5xの方が良い
- `git rebase –i` を良く使いましょう

- Tests should be a separate commit
  - Either ‘Tests after Code’ or ‘Code after Tests’
- Small commits that individually make sense
  - Sometimes you rewrite code you don’t actually need to make that work
- Don’t pack too much in one pull-request
  - Better to leave a 10x improvement to an initial 2x and a follow up 5x later
- `git rebase –i` is your friend

Githubでpull-requestを開く

Open a Pull-Request on Github

- 自分のブランチのフォークにTravisが成功したことを確認する



- 変更の理由と説明を明確に書く
- 参考のために他のmergeされたPRを見る

- Make sure Travis is enabled and passing on your fork for your branch
- Write up a few paragraphs motivating and describing the changes
- Look at other merged PRs for examples

待つこと
Waiting Game

- レビューは時間かかる！
- フィードバックをもらう時点に返事をする

- Review takes time!
- Respond to feedback as you get it

初めての貢献
Your First Contribution

- 低いリスクでBitcoin PRプロセスを体験する
- これをやって見てください
  - ドキュメンテーションを追加する
  - テストを追加する
  - タイプミスを直す
- 宿題：今週末小さいPRをやってね☺□
- 貢献のガイドを読む：
  - https://github.com/bitcoin/bitcoin/blob/master/CONTRIBUTING.md
  - https://github.com/bitcoin/bitcoin/blob/master/doc/developer-notes.md

- Experience Bitcoin PR process with low stakes trial
- Try to
  - Add documentation
  - Add tests
  - Fix typos
- Homework: Do a small PR this weekend ☺
- Read Official Contribution Guides:
  - https://github.com/bitcoin/bitcoin/blob/master/CONTRIBUTING.md
  - https://github.com/bitcoin/bitcoin/blob/master/doc/developer-notes.md

基本アドバイス
GENERAL ADVICE

読んだほうが良い
Good to Read

- Bitcoin SoK Paper [Felten et al.]
- Pull Requests/Issues on Repo
- Bitcoin Improvement Proposal notes [BIPs]
- Kanzure's Archive [diyhpl.us/~bryan/papers2/bitcoin]
- Peter Todd's Blog [petertodd.org]
- Computer Systems Security (6.858, 6.857, 6.875) [css.csail.mit.edu]
- Game Theory with Engineering Applications [OCW]
- Follow CCS/Oakland/Crypto/FC/… conferences
- Scaling Bitcoin Archive [scalingbitcoin.org]
- Twitter

読まないほうが良い
Bad to Read

- 大体避けたほうが良い:
  - Redditのコメント
  - NYT/Economist/etc からのBitcoinの開発の記事
- “読んだほうが良い” ものは十分ある！

- Mostly Avoid:
  - Comments on reddit
  - Articles in NYT/Economist/etc about Bitcoin development
- There’s enough “Good to Read” to keep you busy and happy!

交流しましょう
Socialize

- 喋られる良い人はいっぱいいる！
- Scaling Bitcoinのカンファレンスへ行く
- ローカルのBitcoin Meetup (来週にやります！太郎さんに聞いてね)
- Twitter

- Lots of really great people to talk to!
- Go to Scaling Bitcoin conference
- Local Bitcoin Meetups (Ask Taro-San, Next Week!)
- Twitter

我慢して落ち着きましょう
Be Patient

- Bitcoinは**セキュリティ**に中心するソフト
  - 開発は**ゆっくり**で行く…
  - デベロッパーはコードを**厳しくチェック**する…
  - 壊れたコードで**ムカついている**…
  - でも…あなたの作業は**強い影響**を持つ！

- Bitcoin is **security** focused software
  - Development will be **slow**…
  - Developers will **nitpick** your code…
  - Broken code will **upset** you…
  - but… Your work is **high impact!**

質問 / 休憩時間

Questions/Break Time

BIPのプロセス
The BIP Process

Bitcoin Improvement Proposals

- 大きなアーキテクチャーの変更
- https://github.com/bitcoin/bips にある
- 準公式なプロセス
  - この方法で開発進めることが多い、これではない方法で開発進めることも多い

- Process for large, architectural changes to Bitcoin
- Kept at https://github.com/bitcoin/bips
- Semi-Formal Process
  - Lots of development happens this way, lots of development doesn't

詳しくは https://github.com/bitcoin/bips/blob/master/bip-0002.mediawiki にて。

BIPとは…

A BIP Can Be…

1. 新しい機能の提案
2. ある課題に関するコミュニティの意見
3. 既に決まったデザインのドキュメンテーション

1. A New Feature Proposal
2. Community Input on an Issue
3. Documenting Design Decisions that have gone into Bitcoin

BIPはこれじゃない…

A BIP Shouldn’t be…

1. 非標準かインプレメンテーションに頼る小さい機能
2. 前に議論して却下されたアイデア
3. 重複の作業
4. 形が間違えている BIP
5. モチベーションが無いもの
6. 幅が広いすぎるもの
7. リファレンス実装が無いもの

1. Small Features that are non-standard or implementation dependent
2. Ideas previously discussed and rejected
3. Duplicated Efforts
4. Malformatted BIPS
5. Without Motivation
6. Overly Broad
7. Without Reference Implementations

BIPの基本の形
Basic BIP Skeleton

- ヘッダー
- 要約
- スペック
- モチベーション
- 解釈
- 前のバージョンの互換性
- リファレンス実装

・Header
・Abstract
・Specification
・Motivation
・Rationale
・Backwards compatibility
・Reference implementation

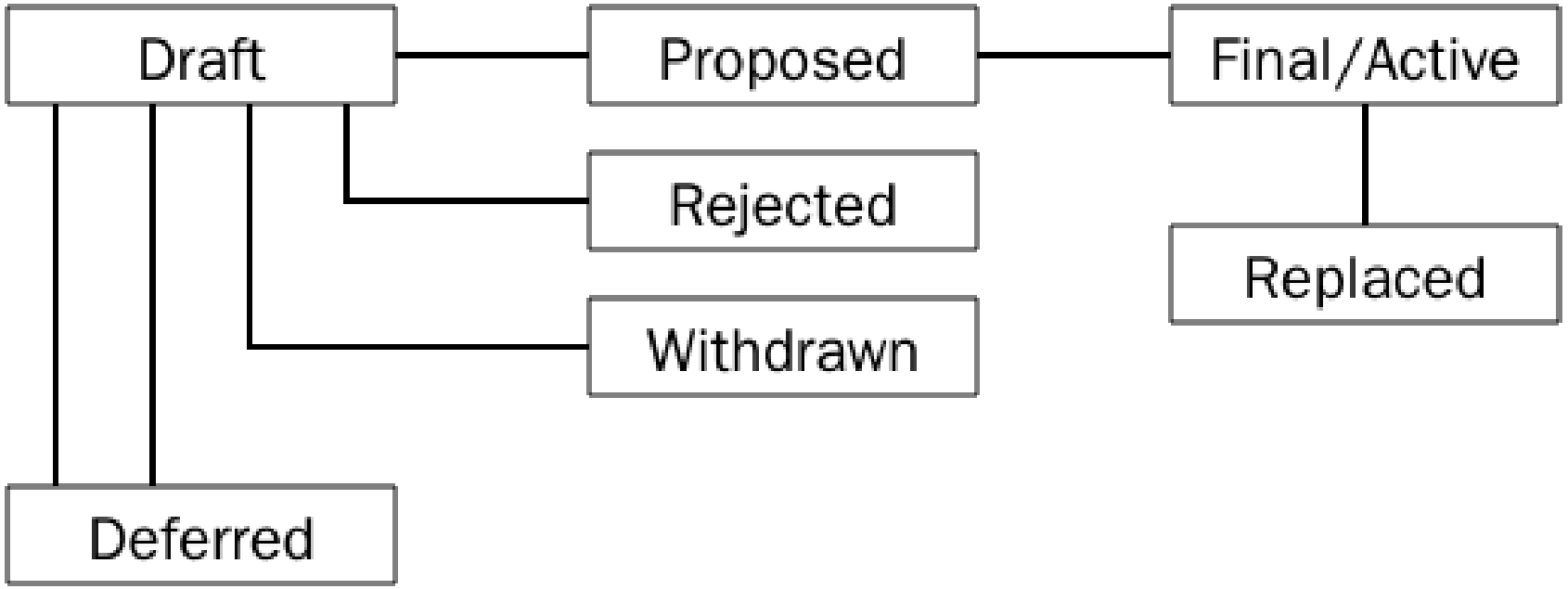
3種類のBIP
Three Kinds of BIP

- スタンダード・トラック
  - ほぼ全部のBitcoinのインプレメンテーションに影響がある
    - ネットワークのプロトコル
    - ブロック / トランザクションの有効性のルール変更
    - Bitcoinを用いるアプリの総合運用性に影響がある変更
  - 2つのパーツ:
    - デザインの資料
    - リファレンス実装
- 情報
  - Bitcoinのデザイン、それとも一般的なやり方の説明
  - 新しい機能の提案じゃない
  - コンセンサスじゃない！
- プロセス
  - Bitcoinのガバナンスの課題

- Standards Track
- Informational
- Process

BIPのライフサイクル

BIP Life



小さなBIPレビュー
Small BIP Review

- 3つのBIPを見てみましょう
  1. BIP9: VersionBits
  2. BIP141: Segregated Witness
  3. BIP32: Hierarchical Deterministic Wallets

- Let’s take a look at three BIPs
  1. BIP9: VersionBits
  2. BIP141: Segregated Witness
  3. BIP32: Hierarchical Deterministic Wallets

BIP 9: VERSION BITS

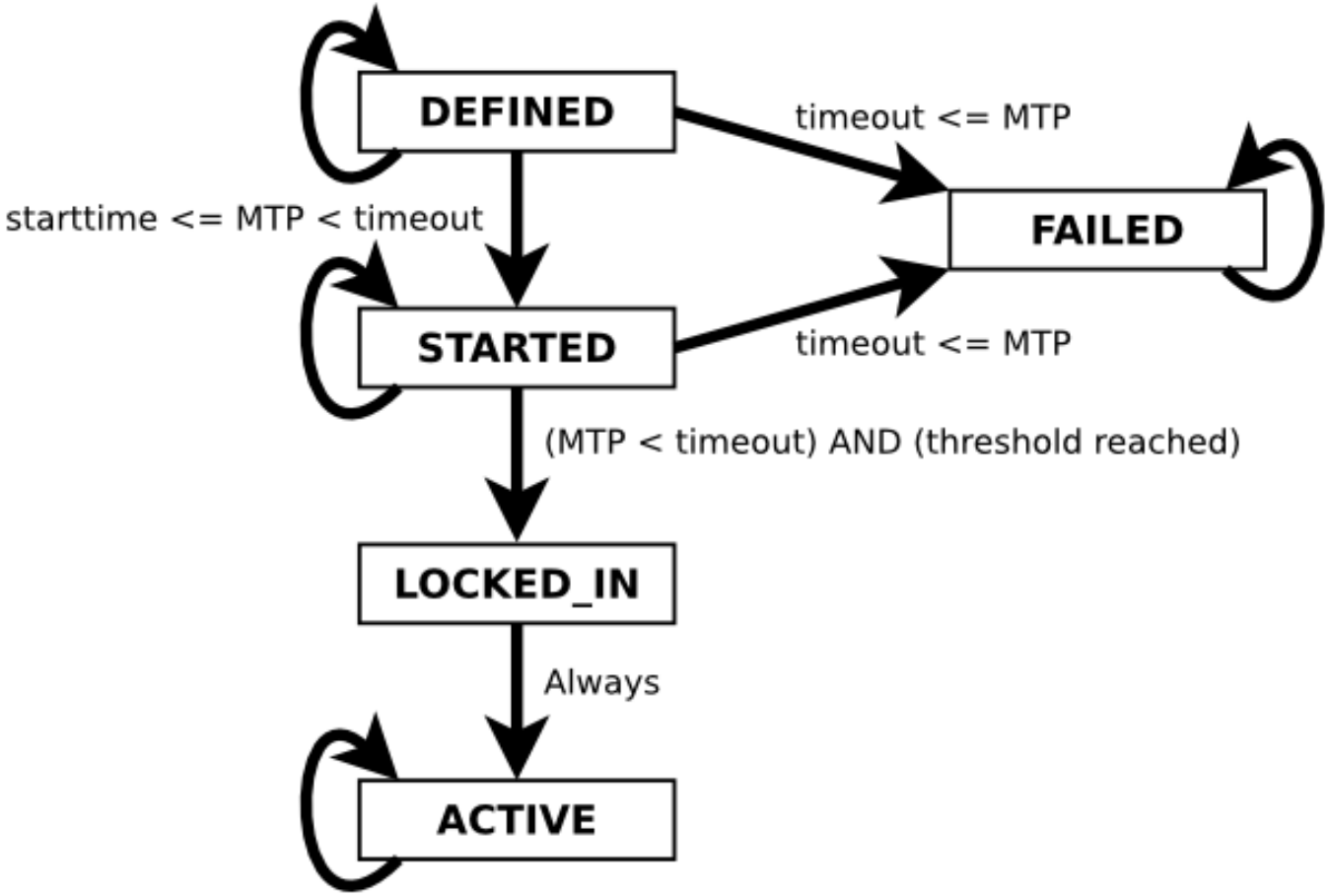
Version Bits

- 情報のBIP
  - meta, Standards Trackのフォークの作成方法
- Bitcoinの32-Bitバージョンフラグを利用して同時に複数soft-forkを実行する方法
- Soft forkは名前、ビット（0~28）、開始時間、タイムアウトを持っている
- 前の2016ブロックはそのビットを設定したのかをチェックする

- Informational BIP
  - meta, how to make Standards Track Forks
- Describes how to use Bitcoin's 32-Bit version flag to deploy simultaneous soft-forks
- Soft fork has a name, a bit (0-28), a start time, and a timeout
- Check that 95% of the preceding 2016 blocks set the bit

バージョンビットのデプロイ

Version Bit Deployment



BIP 141: SEGWIT

Segregated Witness

- Standards Track
- Bitcoinの2つの問題を解決
  - ブロックサイズ（ブロックごとにもっとTx入れる）
  - Transaction Malleability（トランザクション展性）
    - 署名が無効にならなくてもTxの変更はかなり出来る
- Segregated Witness
  - 全部の署名データは別のところになる！

- Standards Track
- Fix two problems in Bitcoin
  - Block Size (slightly more transactions/block)
  - Transaction Malleability
    - Transactions that can be changed significantly without invalidating signatures
- Segregated Witness
  - All Signature data is separately committed to!

SegWitの実装
SegWit Implementation

Before Segwit

- txid = H([nVersion][txins][txouts][nLockTime])
  - txin = previous txid || **witness**
- Txの連鎖は署名に依存している ☹

- Chain of transactions is dependent on the signatures ☹

After Segwit

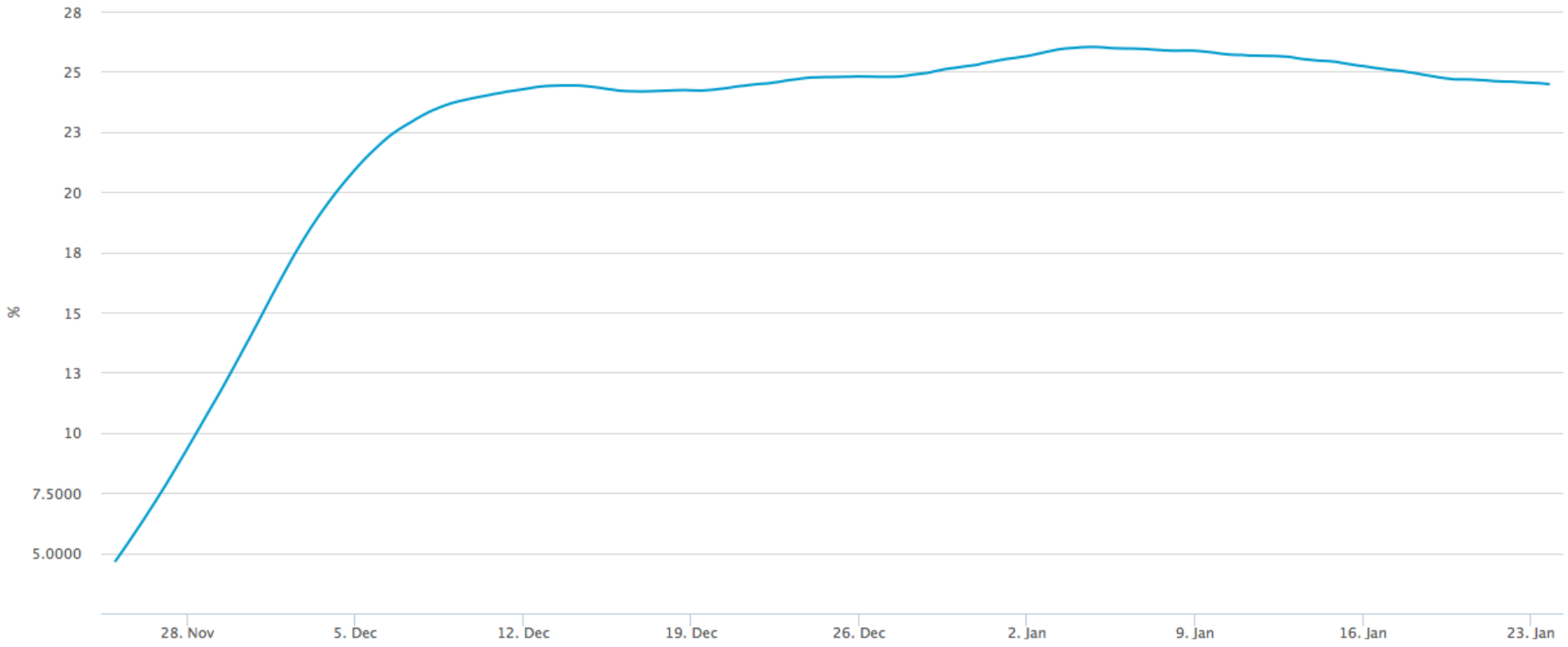
- txid = H([nVersion][txins][txouts][nLockTime])
  - txin = previous txid ~~|| **witness**~~
- wtxid = h([nVersion][marker][flag][txins][txouts][**witness**][nLockTime])
- ブロックの中に別のwtxidsのcommitment
- Txの連鎖は署名と依存していない ☺

- Separate commitment to wtxids in block
- Chain of transactions is independent of the signatures ☺

SegWit実装BIP9 VersionBits

SegWit Deployment BIP9 VersionBits

- name: “segwit”
- bit: 1
- start time: midnight 15 November 2016 UTC
- timeout: midnight 15 November 2017 UTC



これはまずいですか？

Is this Bad?

Matt Corallo <lf-lists@mattcorallo.com> Jan 15 (8 days ago) ☆ Reply to all

to lightning-dev, Rusty, Andrés

On January 15, 2017 9:30:48 PM EST, Rusty Russell <rusty@rustcorp.com.au> wrote:
>If segwit doesn't activate, something is badly broken in Bitcoin. This
>is not really a lightning issue; there's been no significant technical
>objection to segwit, and it really does make Bitcoin work better.

Depends on your point of view.... If you want a digital gold and don't care too much about payments, segwit not activating may be a good thing. It is a strong indication of how hard it is to change Bitcoin, and may even be an indication Bitcoin will never change in a major way again - a strong confirmation of exactly what you'd want from Bitcoin.

...

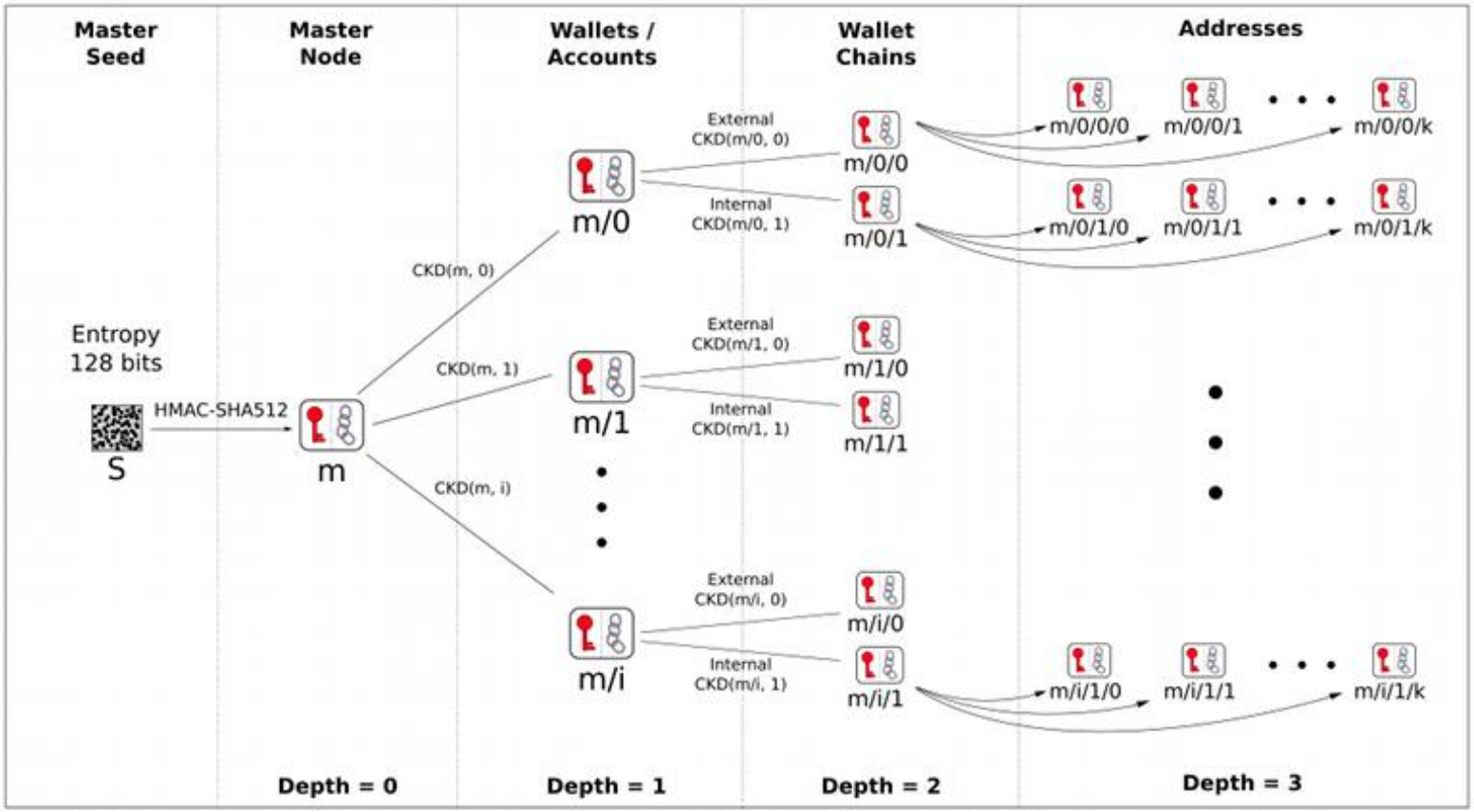
BIP 32:
HIERARCHICAL DETERMINISTIC WALLETS

Hierarchical Deterministic Wallets

- 情報のBIP
- 親の“master key”から子供の鍵が作れる
- “ユーザ” 側のコード、コンセンサスとネットワークじゃない
- でも、ここの標準でセキュリティを高める

- Informational BIP
- Allow deriving a tree of keys from a root “master key”
- “User” code, not consensus or network
- Still, standards here improve user security

BIP 32 - Hierarchical Deterministic Wallets



Child Key Derivation Function ~ $CKD(x,n) = HMAC\text{-}SHA512(x_{Chain}, x_{PubKey} || n)$

質問・休憩時間

Questions/Break Time

Bitcoin性能の開発
Performance Engineering Bitcoin

このセクションの概要
This Section

- Bitcoinのパフォーマンスエンジニアリングについて
- CuckooCache

- Discuss Performance Engineering in Bitcoin
- CuckooCache

パフォーマンスエンジニアリングの必要性
Why Performance Engineering

- スケーラビリティのために速くなるべき
  - スピードが上がるとUXがよくなる
  - TX/sが向上する

- Bitcoin needs to be faster to scale
  - Faster is Better User Experience
  - Handle more TX/s

パフォーマンスエンジニアリングの必要性
Why Performance Engineering

- “簡単”に始められる
  - プロファイリングで測ることができる
  - Bitcoinに関する知識のハードルが低い
    - 調査しながら勉強する
  - 小さくて大きな問題

- “Easy” to get started
  - Can measure objective quantity by profiling
  - Don’t need to know much about Bitcoin to start
    - Learn through investigation
  - Small scale & large scale problems

パフォーマンスエンジニアリングの必要性
Why Performance Engineering

- 多様なエンジニアが関われる問題
  - 署名の検証
  - ネットワークリレー
  - データベースキャッシング
  - ウォレットのスキャニング

- Many “Open” Problems for engineers of different backgrounds
  - Signature Validation
  - Network Relay
  - Database Caching
  - Wallet Scanning

パフォーマンスエンジニアリングの必要性
Why Performance Engineering

- 楽しいから*！
- It’s fun*!

*多分だけど

*maybe

CUCKOOCACHE

署名
Signatures

- 署名というのは verify(sig, pubkey) がtrueになる文字列 sig
- 署名の検証は決定論的
  - verify(sig, pubkey) がtrueなら、何度実行しても同じ結果（ true ）になる
- 署名を計算するのはコストが高い

- A signature is a string sig such that verify(sig, pubkey) is true
- A signature verification is deterministic
  - if verify(sig, pubkey) is true, then running it again will also be true
- A signature is expensive to compute

Mempoolのバリデーション

Mempool Validation

- Mempoolのバリデーションのプロセスではトランザクションの中の全ての署名をチェックする

- Mempool validation process checks all signatures in a transaction

ブロックのバリデーション

Block Validation

- ブロックのバリデーションでは、ブロックの中の全てのトランザクションをチェックする。
- トランザクションのチェックでは、トランザクションの中の全ての署名をチェックする。

- Block validation process checks all transactions in the block
- Checking a transaction checks all signatures

Can we avoid checking signatures 2x?

署名を2回チェックするのは避けられるか？

署名のキャッシュ
Signature Cache

- verify(sig, pubkey) がtrueの時にSHA256(sig || pubkey) をHash Setにを保存する
- ハッシュを検索するのは署名のチェックよりコストが低い

- if verify(sig, pubkey), then save SHA256(sig || pubkey) in a hash table
- Hash table is cheaper than a signature check

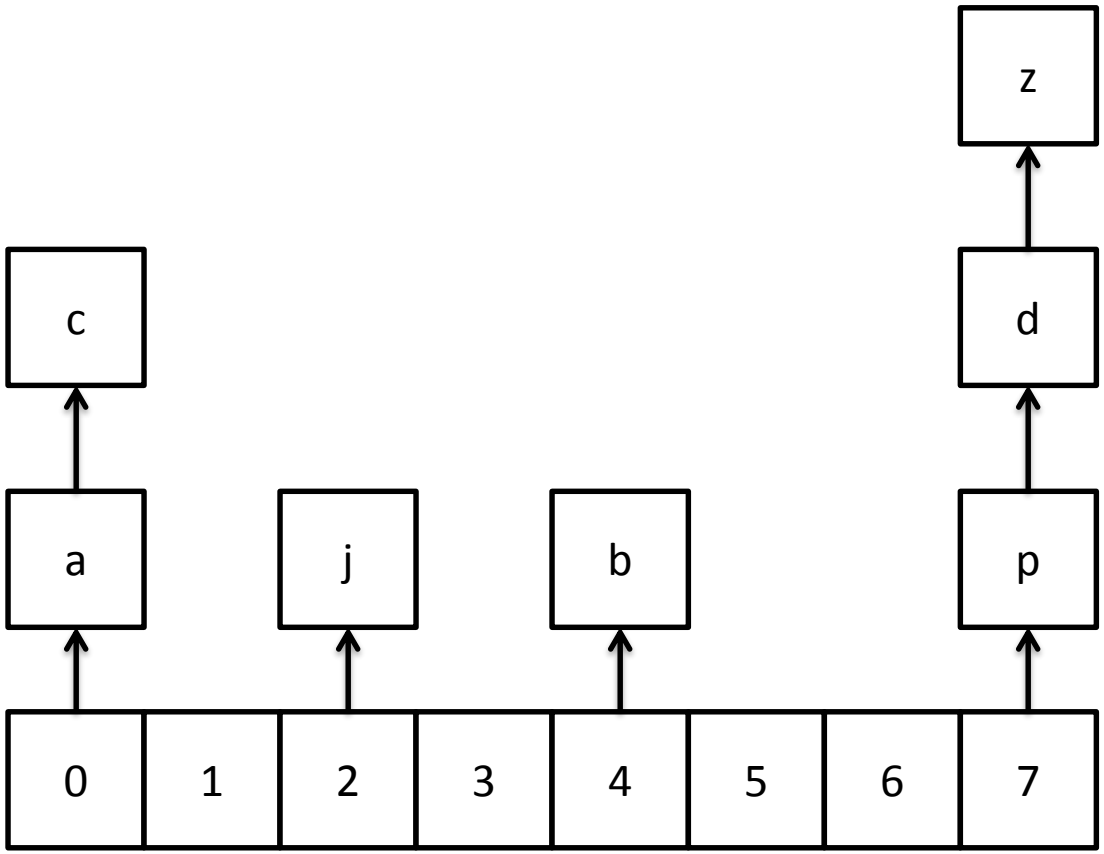
Bitcoinのナイーブ(イマイチ)な実装

Bitcoin's Naïve Implementation

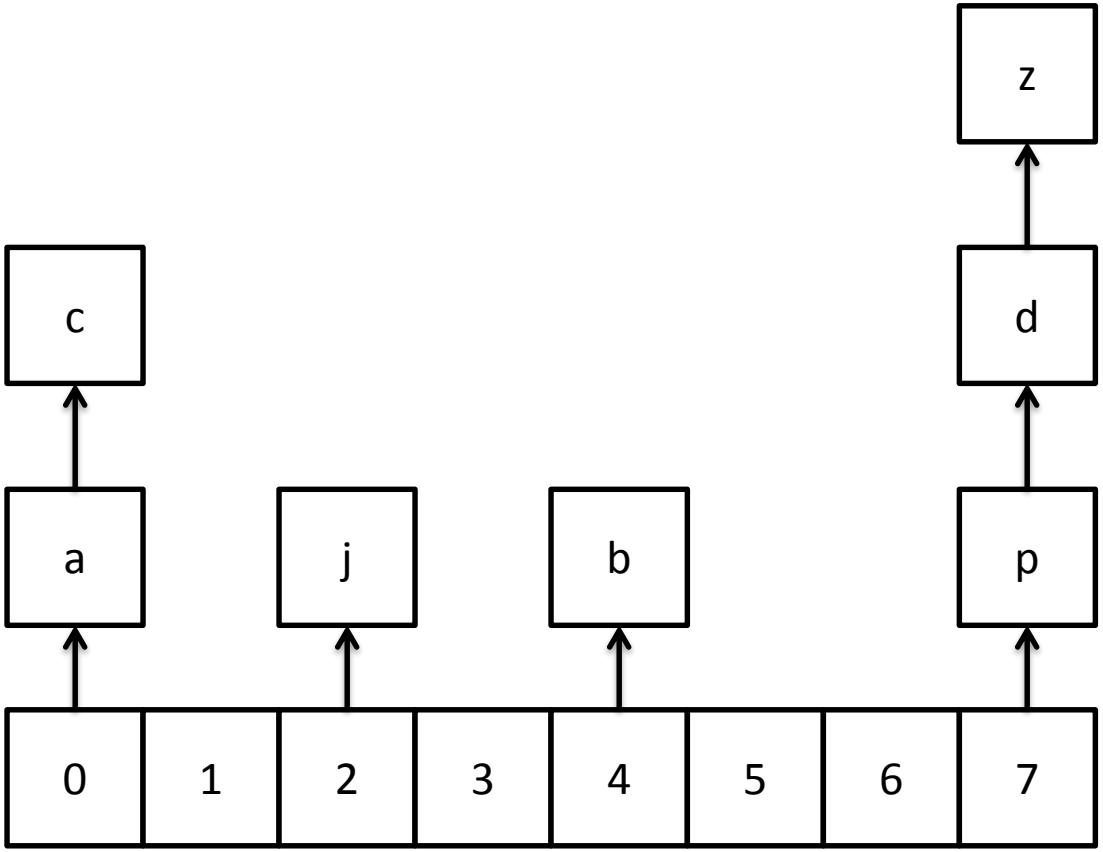
- unordered_set<Hash> set を利用
- unordered_set にはオーバーヘッド、メモリ確保、間接参照がたくさんある
- もし DynamicMemoryUsage(set) が制限値を超えると
  - 制限の範囲に収まるまで、ランダムにsetの要素(バケット)を捨てる
  - 無限ループは可能！
- 同時並行性(16コアで実行する方が8コアよりひどい ☹)
  - 追加時の書き込みロック
  - 読込時の読み込みロック
  - 削除時の書き込みロック☹
  - 読込成功する都度setからその要素を削除 ☹

- use unordered_set<Hash> set
- unordered_set: bad overhead, allocation, and indirection
- if DynamicMemoryUsage(set) > limit
  - Discard random buckets of elements until <= limit
  - Could loop forever!
- Concurrency (Worse to run on 16 Cores than 8 ☹)
  - Read Lock to read
  - Write Lock to insert
  - Write Lock to erase ☹
  - erase on every successful read ☹

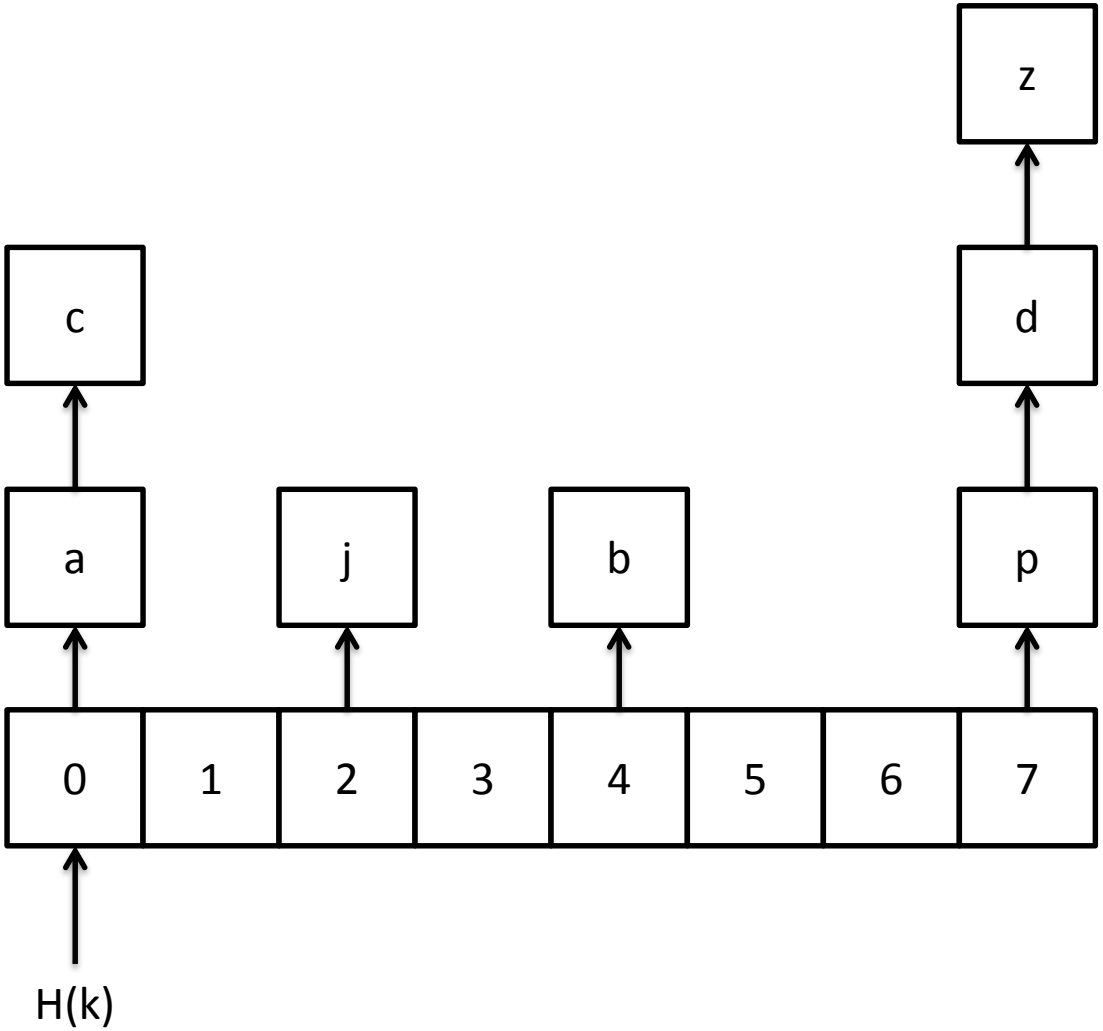
非整列セット・unordered_set



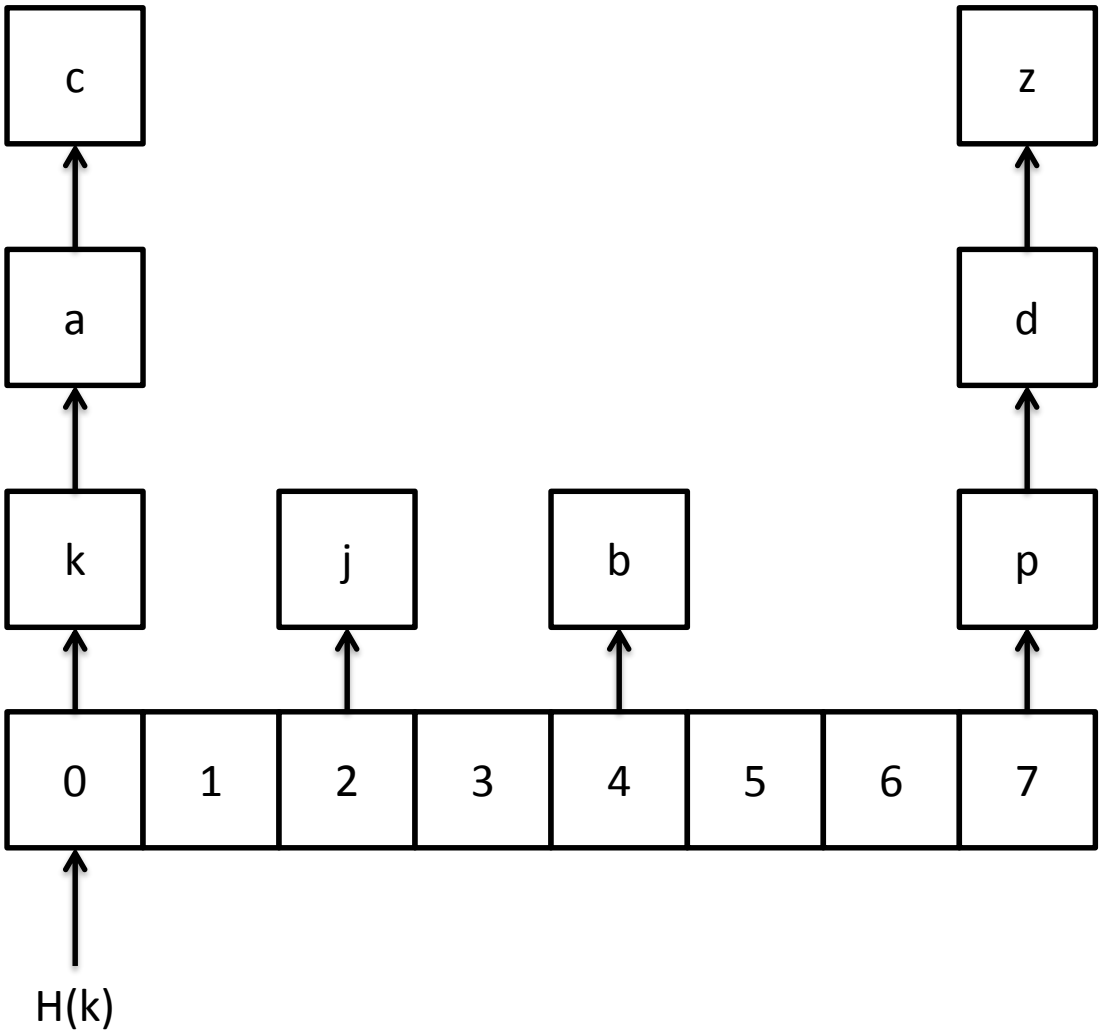
非整列セット・unordered_set
insert(k)



非整列セット・unordered_set
insert(k)

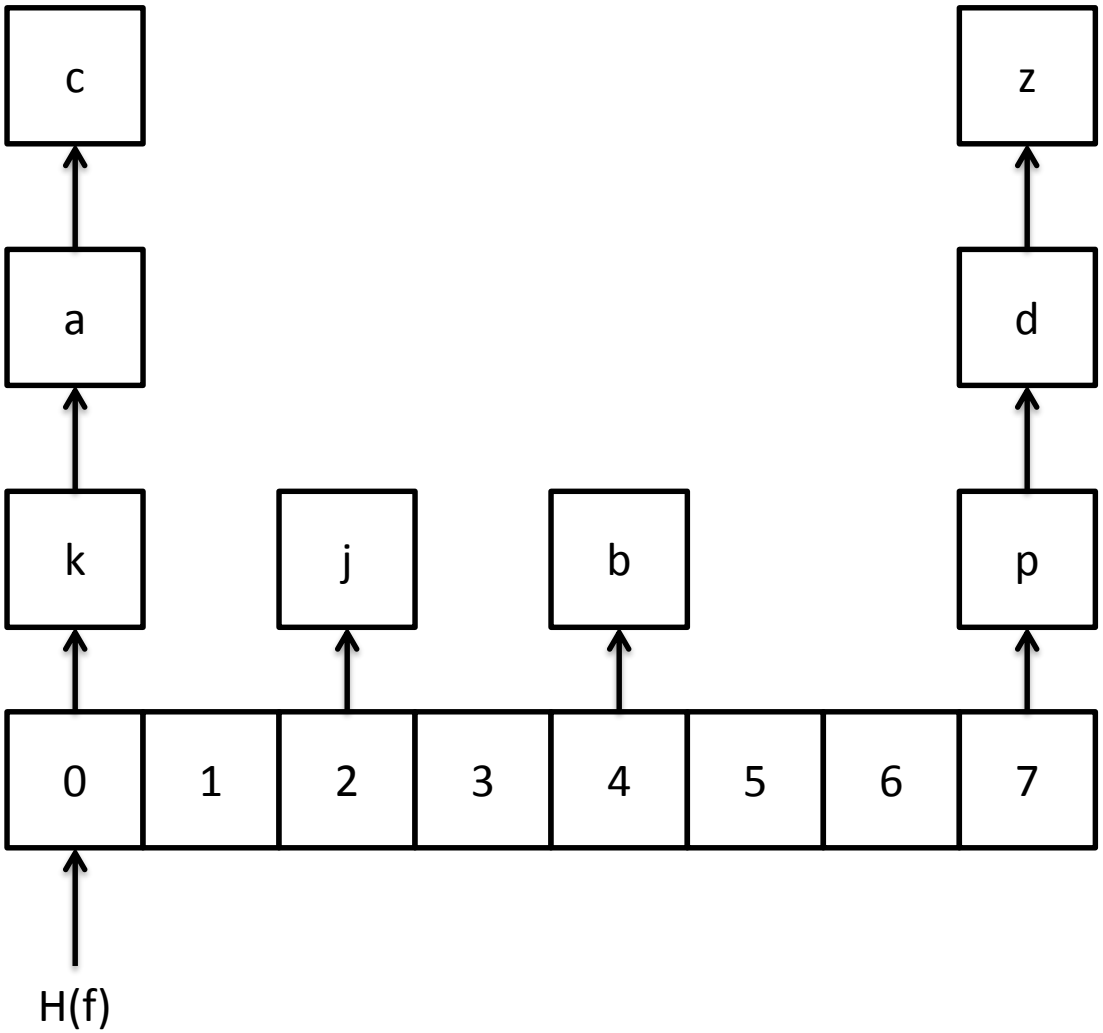


非整列セット・unordered_set
insert(k)



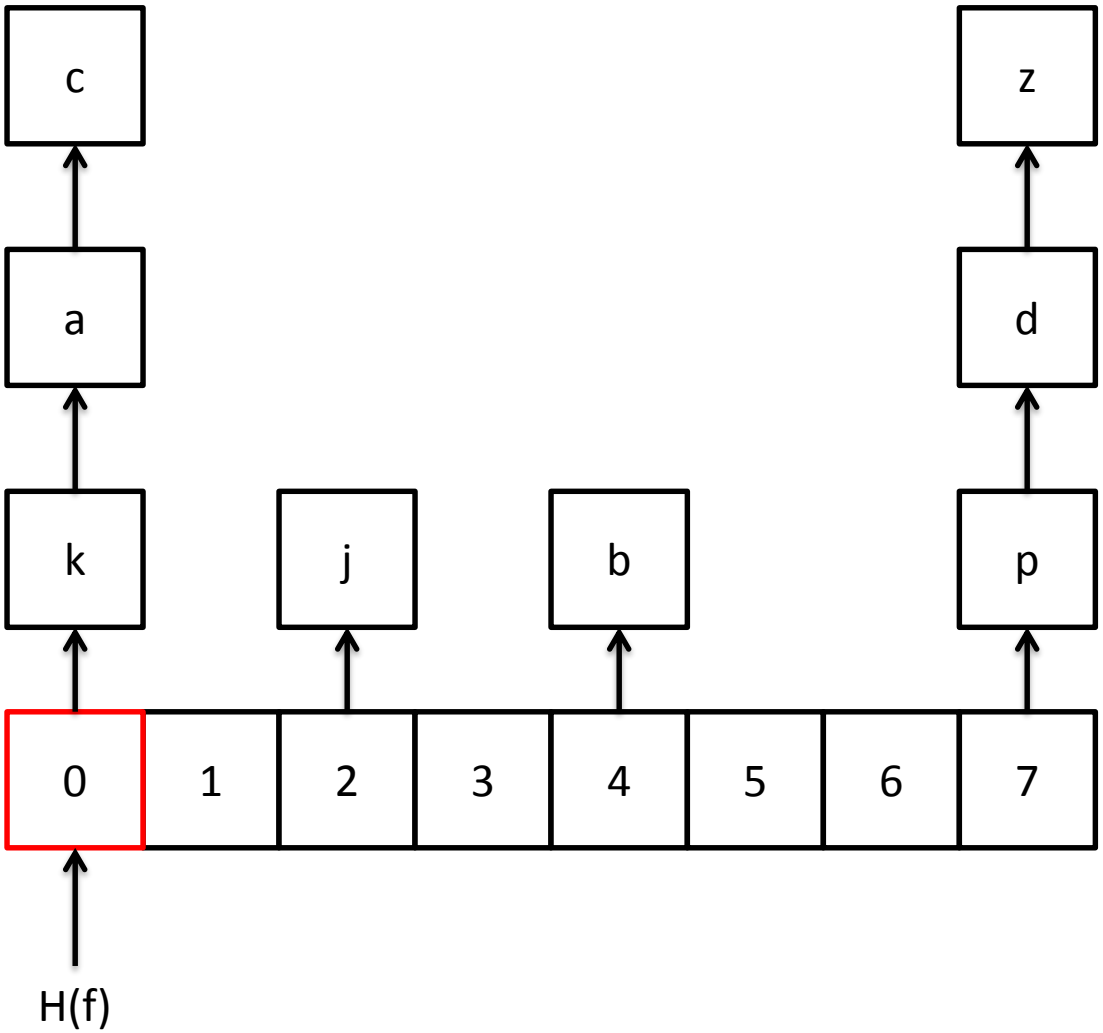
非整列セット・unordered_set

find(f) == false



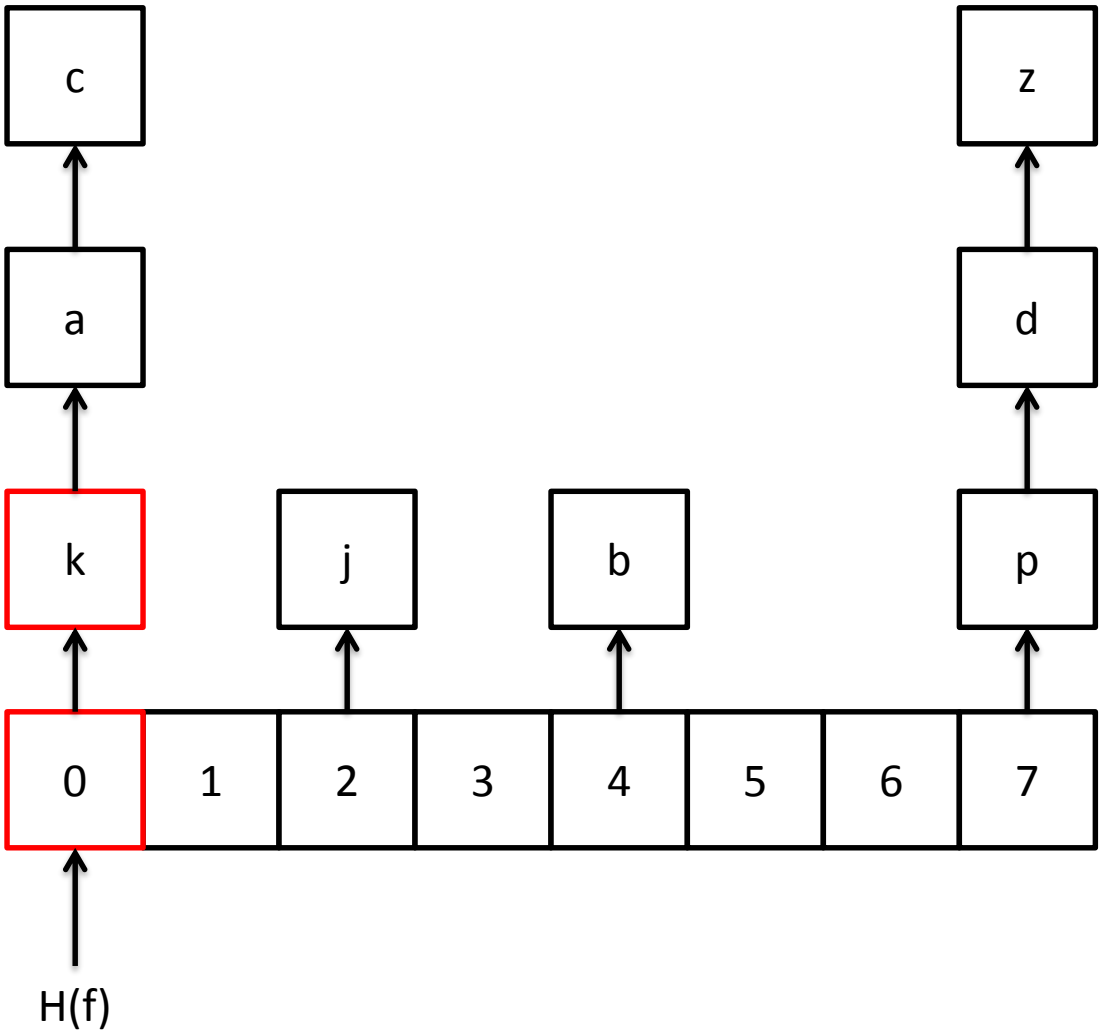
非整列セット・unordered_set

find(f) == false



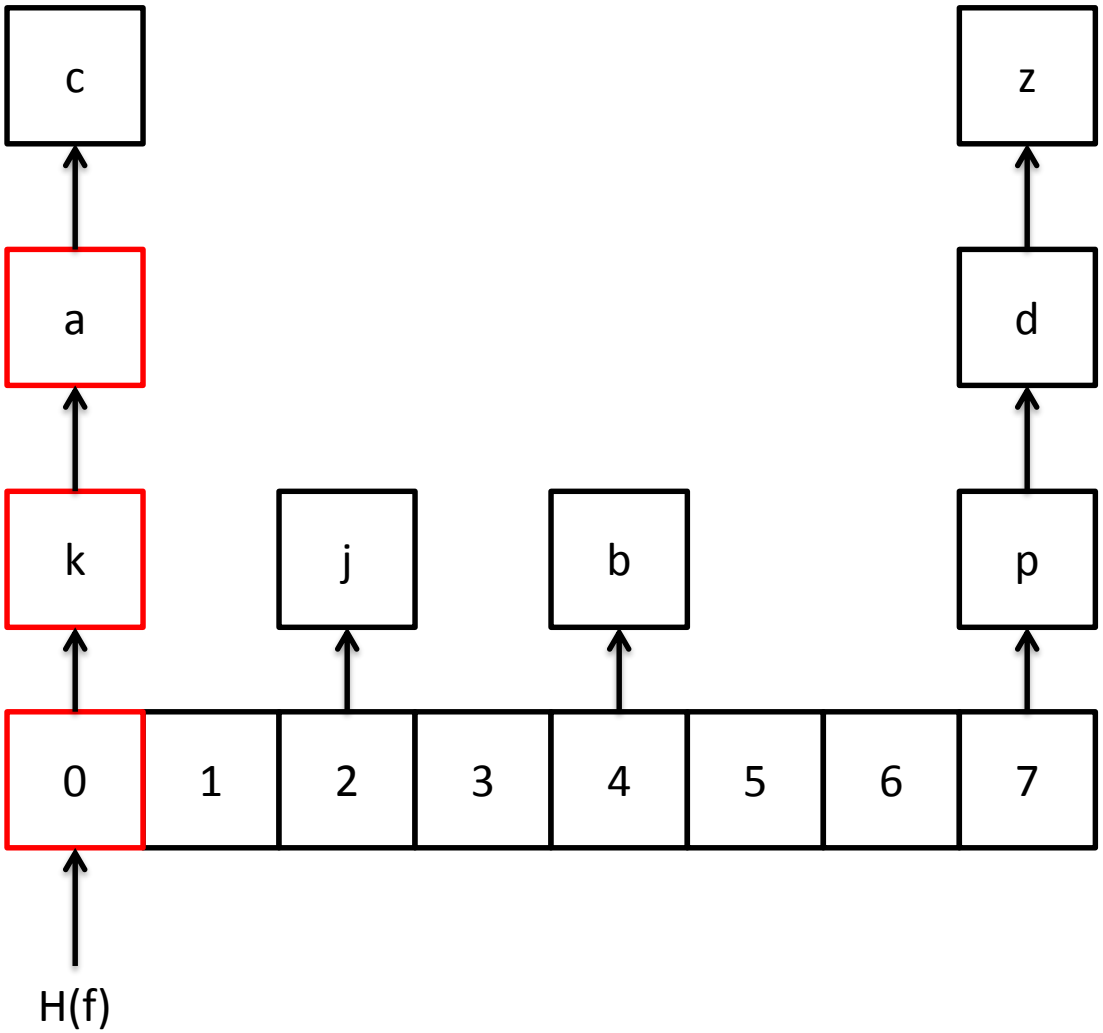
非整列セット・unordered_set

find(f) == false



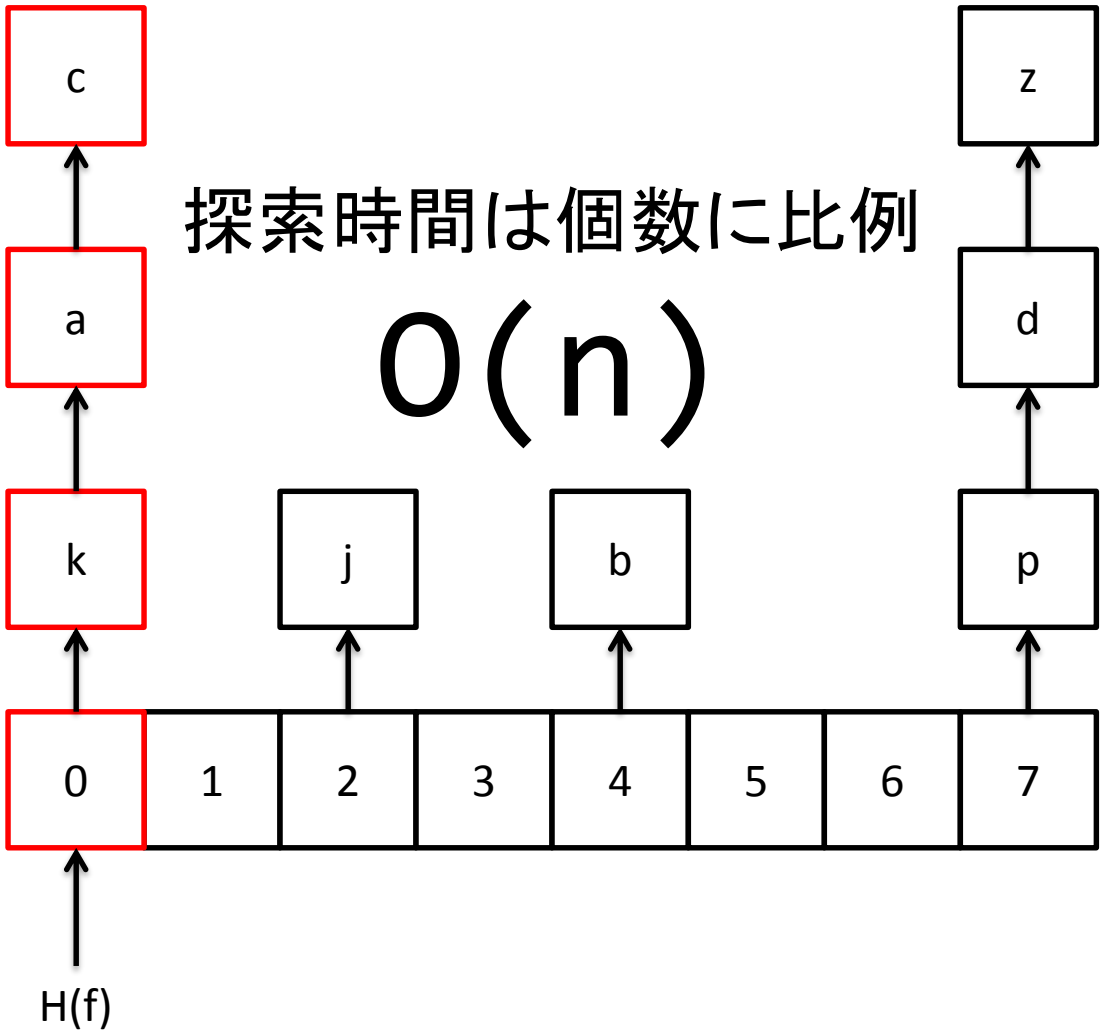
非整列セット・unordered_set

find(f) == false



非整列セット・unordered_set

find(f) == false



Bitcoinのナイーブ(イマイチ)な実装

Bitcoin's Naïve Implementation

- unordered_set<Hash> set を利用
- unordered_set にはオーバーヘッド、メモリ確保、間接参照がたくさんある
- もし DynamicMemoryUsage(set) が制限値を超えると
  - 制限の範囲に収まるまで、ランダムにsetの要素(バケット)を捨てる
  - 無限ループは可能！
- 同時並行性(16コアで実行する方が8コアよりひどい ☹)
  - 追加時の書き込みロック
  - 読込時の読み込みロック
  - 削除時の書き込みロック☹
  - 読込成功する都度setからその要素を削除 ☹

- use unordered_set<Hash> set
- unordered_set: bad overhead, allocation, and indirection
- if DynamicMemoryUsage(set) > limit
  - Discard random buckets of elements until <= limit
  - Could loop forever!
- Concurrency (Worse to run on 16 Cores than 8 ☹)
  - Read Lock to read
  - Write Lock to insert
  - Write Lock to erase ☹
  - erase on every successful read ☹

Cache Wish List

- 必要な時にだけ削除する
- いっぱいなったら、一番古いエントリーを先に削除する
- 同時並行性が良い
- 関節参照が少ない
- メモリー負担が少ない
- 無限ループ(の可能性)がない
- メモリーの割り当てが少ない
- RAMキャッシュの競合が少ない
- 検索時間の改善

- Only erase when new space needed
- When full, delete less needed entries first
- Improve Concurrency
- Decrease Indirection
- Decrease Memory Overhead
- No (potentially) Infinite Loops
- Decrease Memory Allocation
- Decrease RAM Cache Contention
- Improve Lookup Runtime

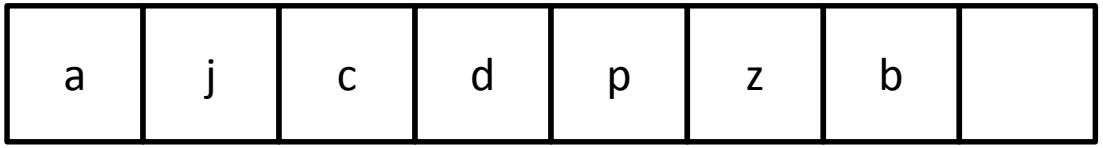
全部直すことは出来る？
CAN WE FIX IT ALL?

基本のアイデア : カッコーハッシュ
Base Idea: Cuckoo Hashing

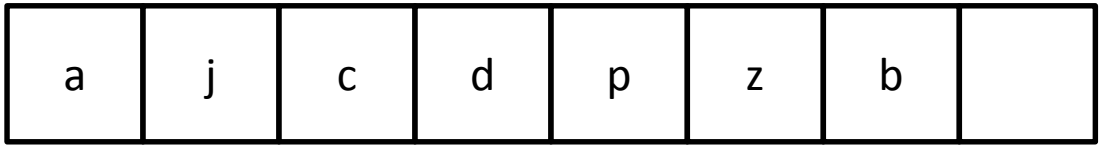
- アルゴリズムはシンプル :
  - **配列:** hash a[NELEMS]
  - **ハッシュ関数:** hash H1(elem e), hash H2(elem e)
  - insert(k, h = H1, iterations = 0)
    - a[h(k)] に k2 があれば
      - a[h(k)] ← k
      - insert(k2, H1(k2) == h(k) ? H2 : H1, iterations+1)
    - 繰り返しがO(log(NELEMS)) になったら,テーブルサイズを倍にする
  - lookup(k) return a[H1(k)] == k || a[H2(k)] == k
- 性能が良い。ハッシュ関数をH3,H4 …と増やせば特に。(証明は難しい…)

- Very simple algorithm:
  - **array:** hash a[NELEMS]
  - **hash functions:** hash H1(elem e), hash H2(elem e)
  - insert(k, h = H1, iterations = 0)
    - if a[h(k)] is occupied by k2
      - a[h(k)] ← k
      - insert(k2, H1(k2) == h(k) ? H2 : H1, iterations+1)
    - if hit O(log(NELEMS)) iterations, double table size
- Performs very well, especially with more hashes H3, H4… (complicated proof…)

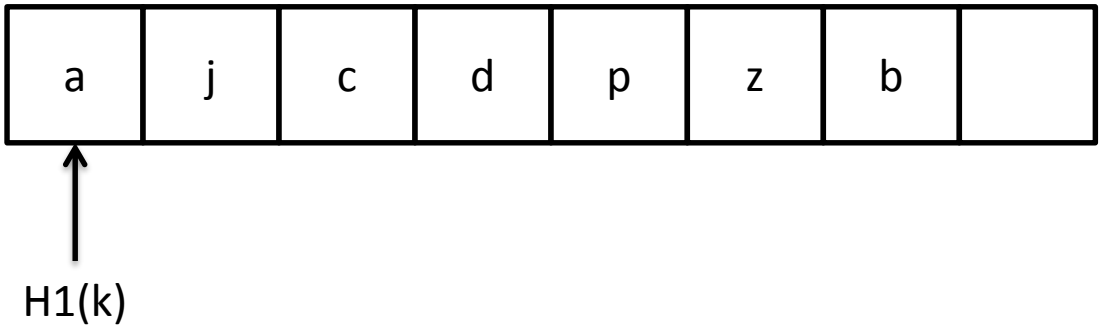
カッコーテーブル・cuckoo table



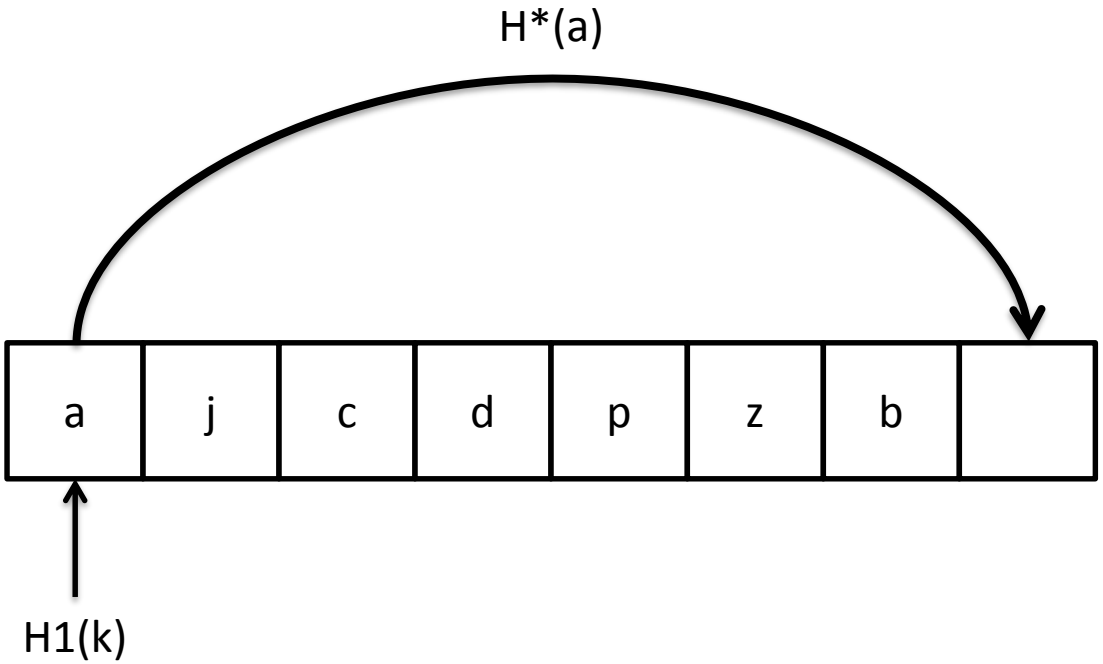
カッコーテーブル・cuckoo table
insert(k)



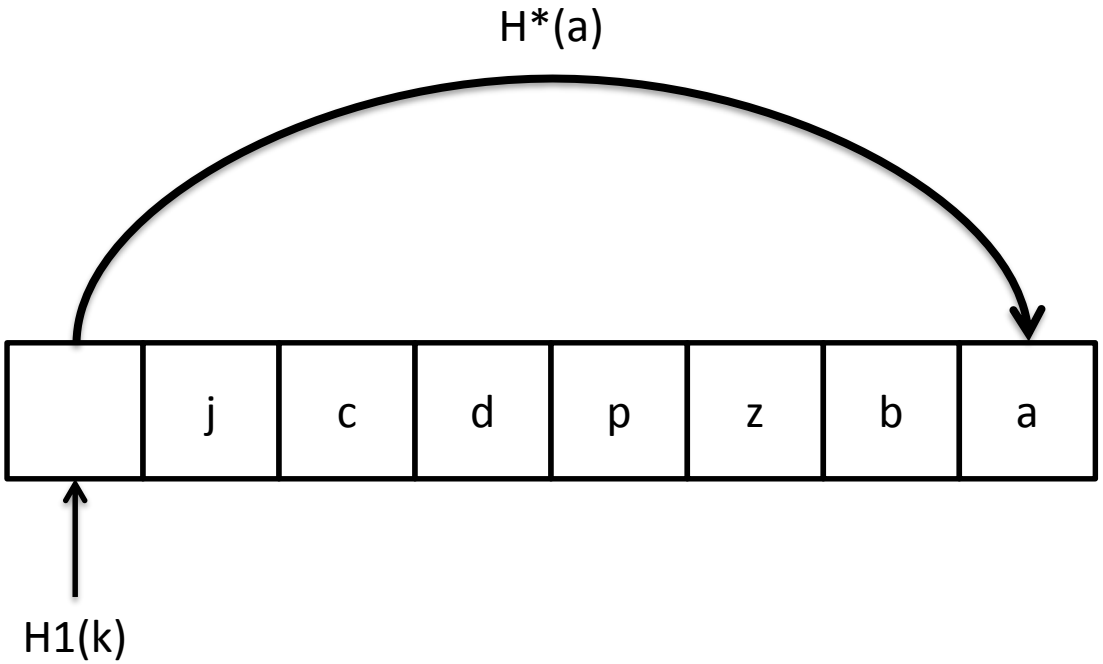
カッコーテーブル・cuckoo table insert(k)



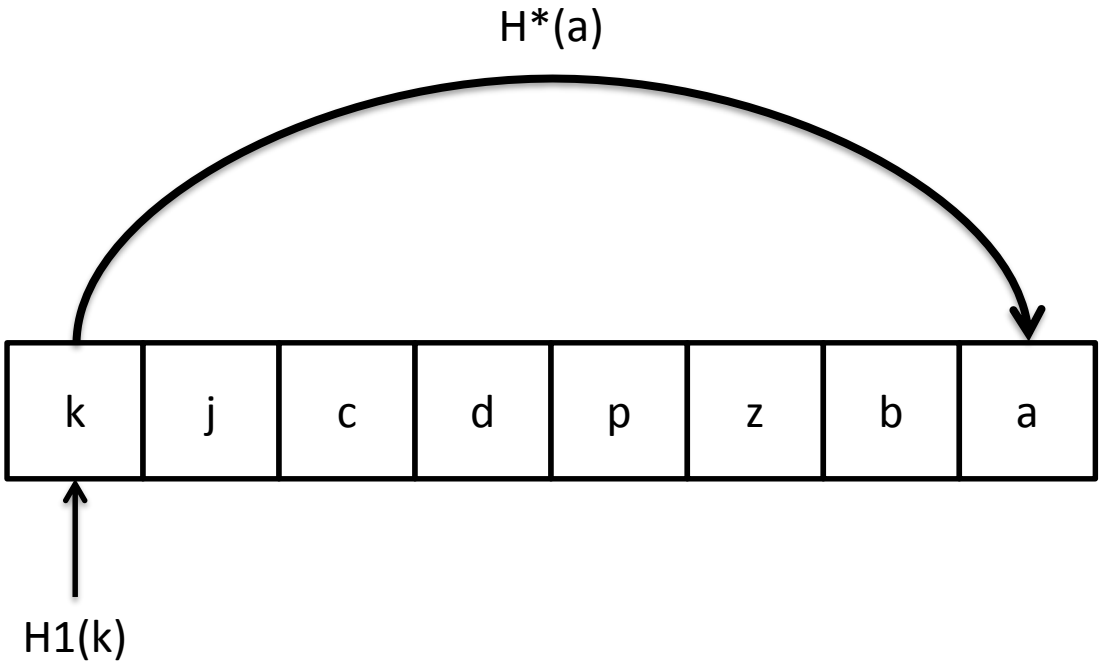
カッコーテーブル・cuckoo table insert(k)



カッコーテーブル・cuckoo table insert(k)



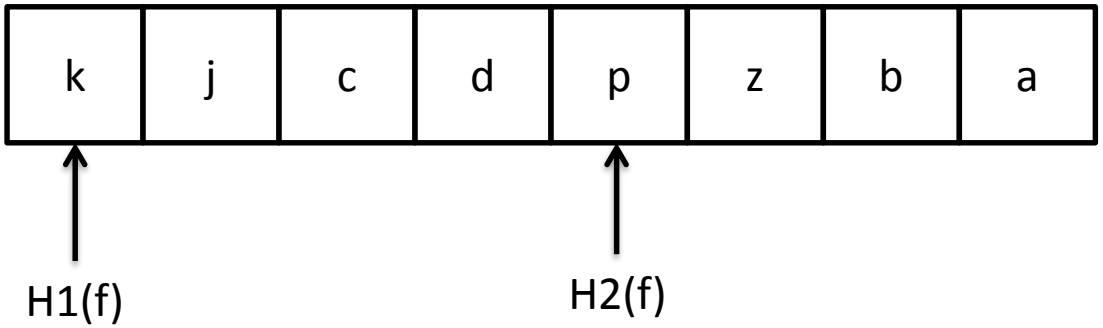
カッコーテーブル・cuckoo table insert(k)



カッコーテーブル・cuckoo table
find(f) == false

個数が増えても
探索時間が一定

O(1)



カッコーキャッシュ

CuckooCache

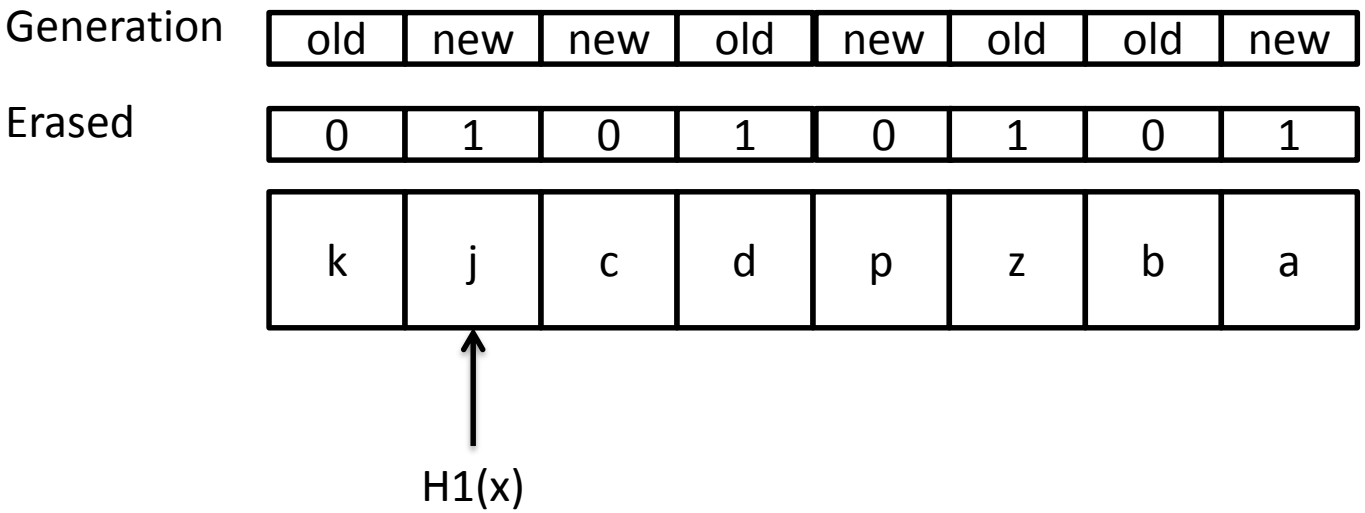
- 固定サイズのハッシュテーブルを構築し（2倍にしない）
  - O(log(NELEMS)) の反復制限で最後の様子を削除する
- ハッシュ関数 H1…H8 を利用する
  - SHA256は32bytesだから、32 bytes/4 bytes = 8
- アトミックなパックしたビット配列を保持する
  - 「削除可能」をマーキングするため
  - 新しいデータまでに様子を削除しない
- 「世代」のマーキングのためのパックしたビットを保持する
  - テーブルはいっぱいになったら世代交代する
  - 制限を超えたら世代を上げる

- Fixed size table (no doubling)
  - drop last element at O(log(NELEMS)) recursion limit
- Use H1…H8 hash functions
  - because SHA256 is 32 bytes/4 bytes = 8
- Keep a array of atomic bit-packed bytes
  - for marking erasable
  - don’t erase element until new data
- Keep a array of bit-packed bytes for generation
  - retire generations when table gets cramped
  - Age when older than THRESHOLD

カッコーキャッシュ・CuckooCache

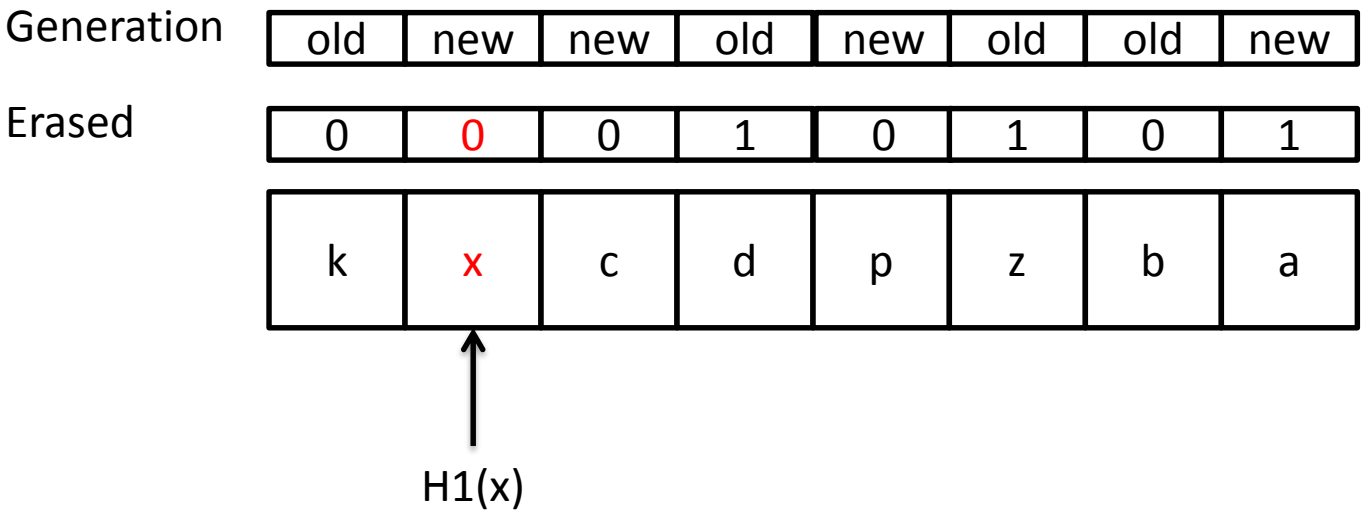
カッコーキャッシュ・CuckooCache insert(x)

ハッシュ結果が空き(削除済)の枠

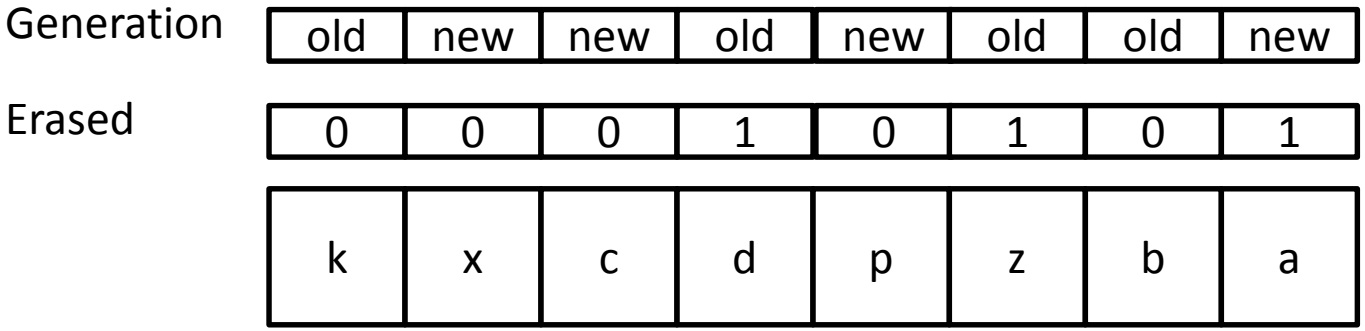


カッコーキャッシュ・CuckooCache
insert(x)

そこに格納して使用中に



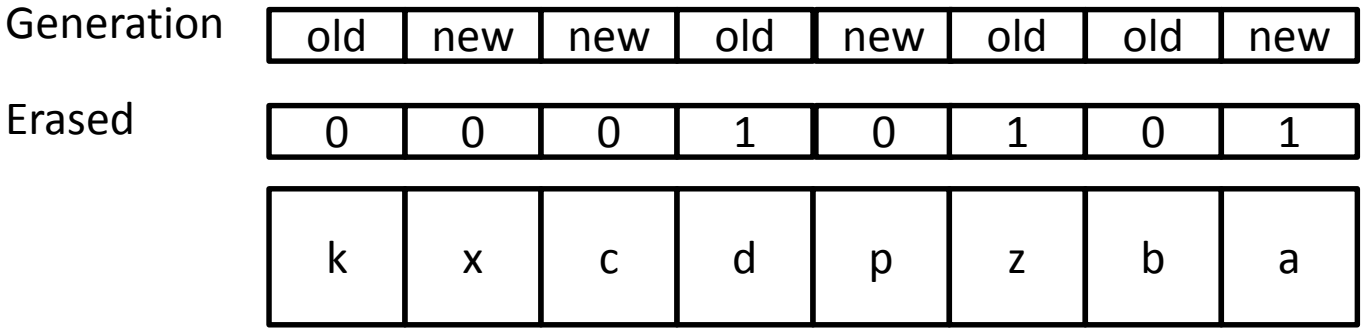
カッコーキャッシュ・CuckooCache
insert(y)



カッコーキャッシュ・CuckooCache
insert(y)

「新」世代の使用中の数が閾値を超過！

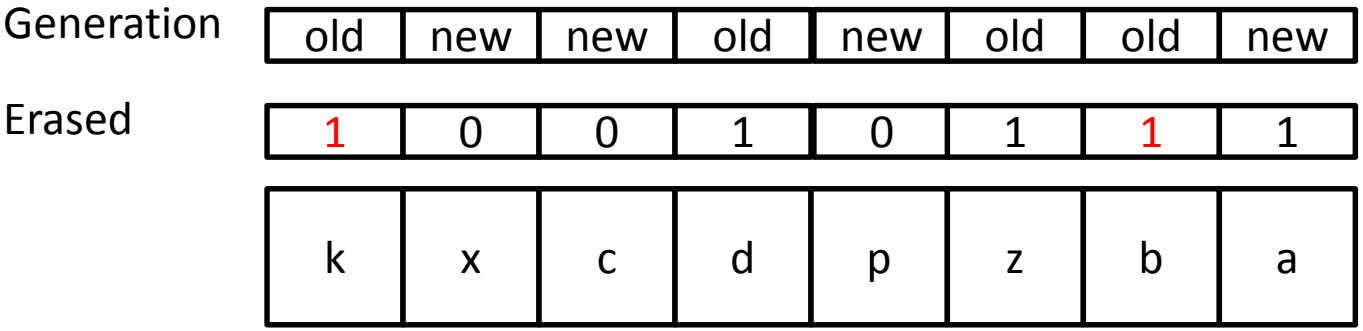
(# new && ! erased) > THRESHOLD



カッコーキャッシュ・CuckooCache
insert(y)

「旧」世代の使用中を削除済=空きに

Erase Old

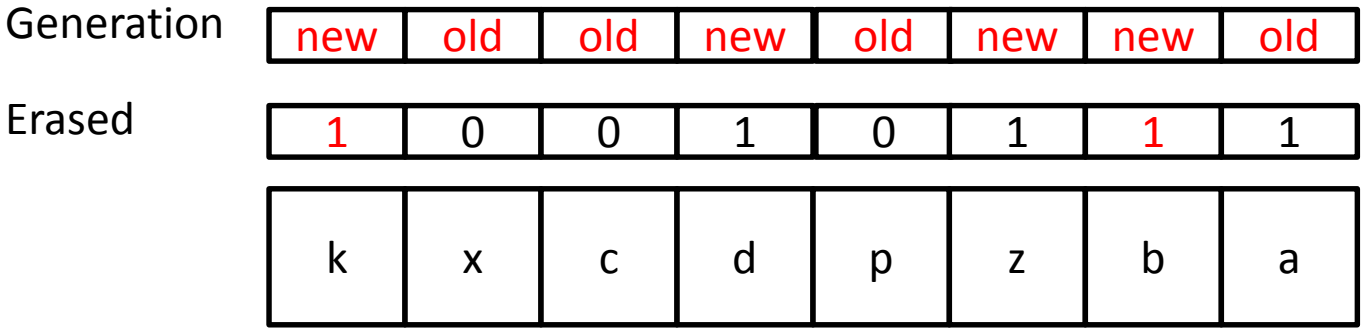


カッコーキャッシュ・CuckooCache
insert(y)

世代交代:
「新」を「旧」に、「旧」を「新」に

Retire Generation

※「新」世代は全て空き



どんな結果が出ましたか？

How’d We Do?

- ✓ 別スレッドでのエントリー追加時に遅延削除する
  - ✓ ボーナス：Reorgの時にパーフォマンスが上がる
- ✓ 古いエントリーから先に削除することを世代で保証できる
  - ✓ 古い要素→必要性が低い
- ✓ 読込時と削除時にロックが不要となる
- ✓ 関節参照はほぼ**ゼロとなる**
- ✓ メモリーの負担はほぼ**ゼロとなる** (< 1%)
- ✓ 全ての処理が必ず終了する
- ✓ メモリーの追加の割り当ては**ゼロとなる**
- ✓ 16コアで40%早くなる
- ✓ O(1)の検索

- ✓ Lazy erasure on future inserts
  - ✓ Bonus: Better performance during Reorgs
- ✓ Generations ensure older entries deleted first
  - ✓ Older → Less needed
- ✓ **Zero** locks needed for Reads and Erases
- ✓ Almost **Zero** indirection
- ✓ Almost **Zero** memory overhead (< 1%)
- ✓ All operations terminate
- ✓ **Zero** memory allocation
- ✓ **40%** faster on 16 cores
- ✓ **O(1) lookup**

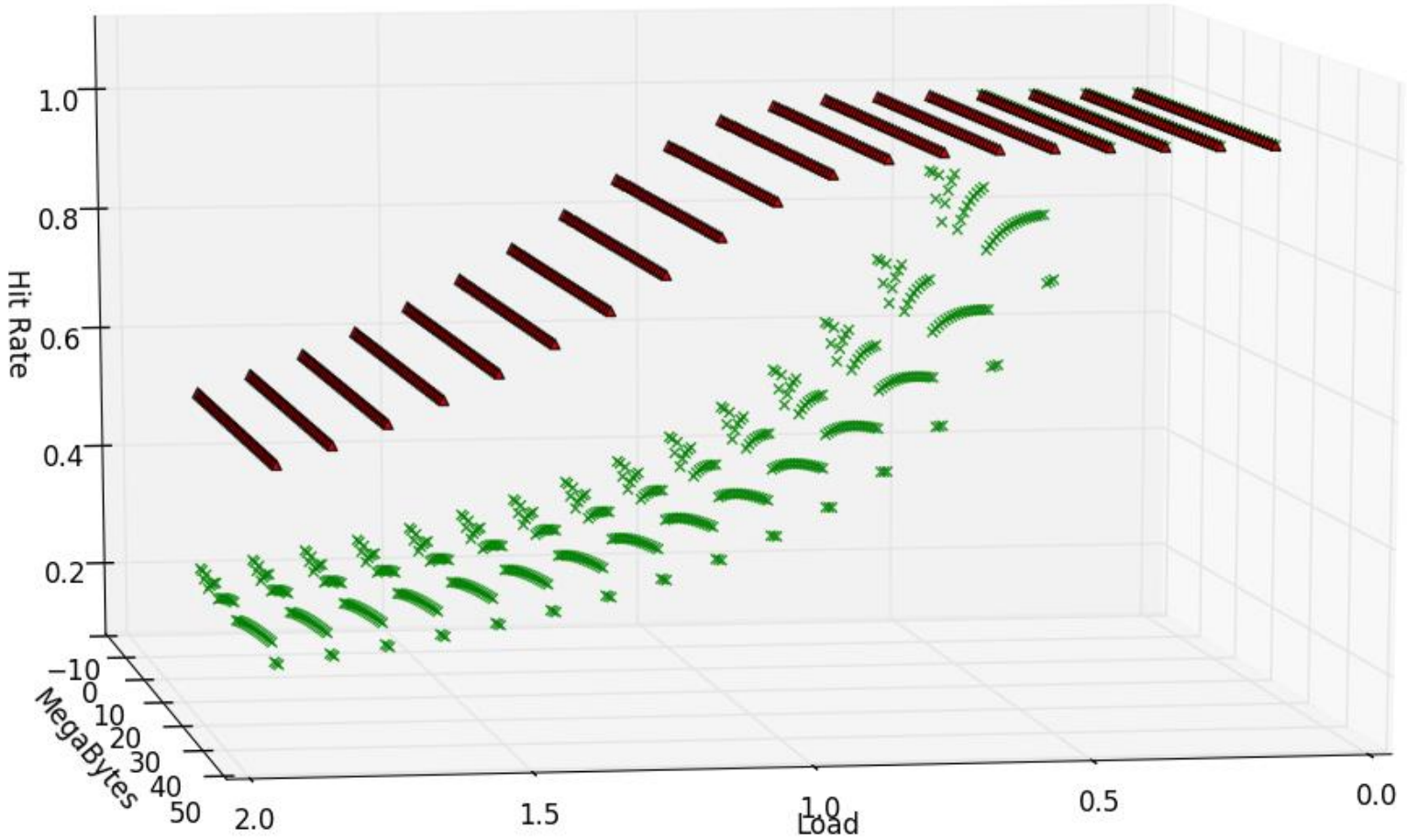
How’d I know I did better?

- 色々なシチュエーションで新旧の実装を比較できるキャッシュシミュレータ
- 証明ためのグラフィック
- 最初に作ったものは、シンプルなシミュレータだった！
- ネットワークに接続しているノードのシミュレーションは大変大事ですが、より難しい！もっと良い開発ツールが必要…
  - 歴史を持っているビットコインネットワーク上でシミュレーションするために前は一般公開では無いツールを使った

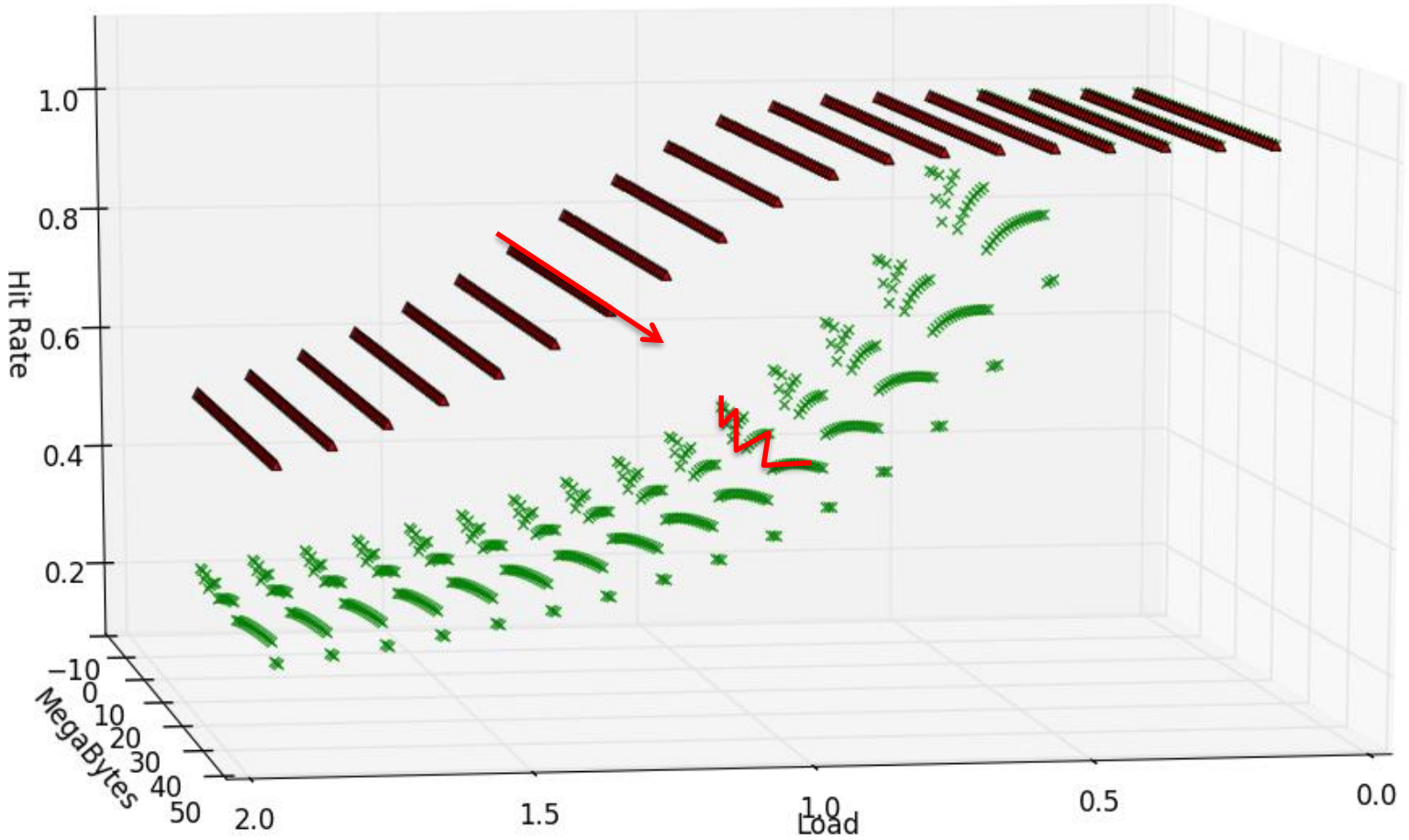
- Cache simulator to compare old & new implementations across situations
- Graphical demonstration
- A simulator was one of the **first things** I did!
- Networked node simulation is very important, but harder to do! Better dev tools needed…
  - non-public tools to simulate historical bitcoin network

ヒットレートは最適に近い！

Near Optimal Hit Rate!



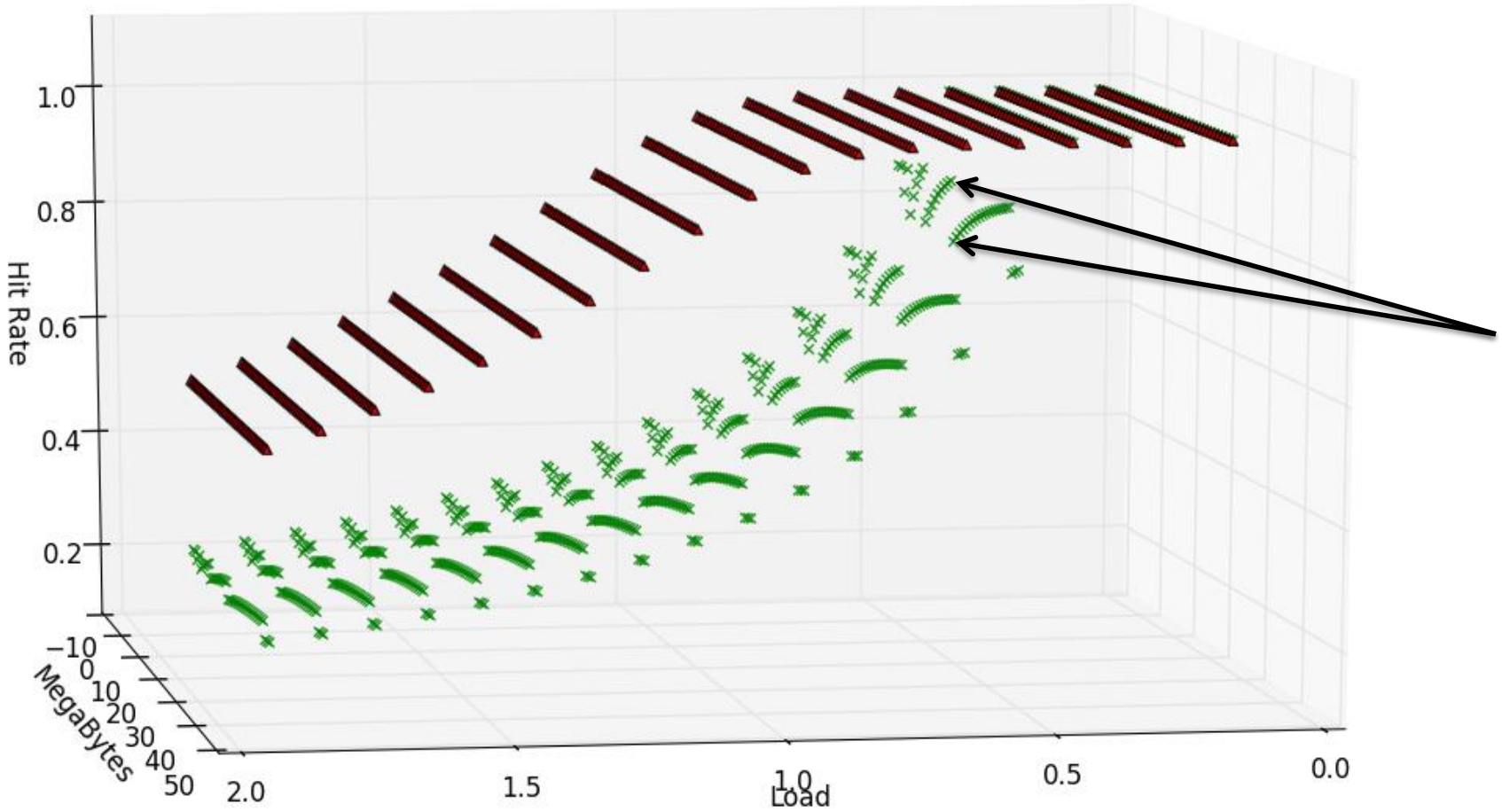
低い分散
Low Variance



(ノ°□°)ノ ⌒ ┻━┻

古い：より小さいサイズ、より良いパフォーマンス

Old: Less Size, Better Performance



How’d I know I did better?

- 色々なシチュエーションで新旧の実装を比較できるキャッシュシミュレータ
- 証明ためのグラフィック
- 最初に作ったものは、シンプルなシミュレータだった！
- ネットワークに接続しているノードのシミュレーションは大変大事ですが、より難しい！もっと良い開発ツールが必要…
  - 歴史を持っているビットコインネットワーク上でシミュレーションするために前は一般公開では無いツールを使った

- Cache simulator to compare old & new implementations across situations
- Graphical demonstration
- A simulator was one of the **first things** I did!
- Networked node simulation is very important, but harder to do! Better dev tools needed…
  - non-public tools to simulate historical bitcoin network

テスト
Testing

- シミュレータを使って:
  - キャッシュのパフォーマンスの悪化を防ぐ
  - プロパティの正しさを確保する

- Used the simulator to
  - prevent cache performance regression
  - ensure correctness properties

レビューのプロセスのタイムライン

Review Process Timeline

- コードの始まり：2016年夏
- PRを開いた：2016年10月5日
  - コメント：ランダムなプラットフォームのエラー…
- 「世代」の追加：2016年10月20日
- 10月21日のコメント：
  - sipa:2^nのサイズに変更し、bitmaskを使ってね、(a%n)は遅いから
- 11月9日のコメント：
  - Deprecated フラグの珍しいビルドのエラー
- PRはmergeされた：2016年12月14日
- sipaは2^nの制限を外した：2017年1月12日

- Code Started: Summer 2016
- PR Opened: Oct 5th, 2016
  - Comments: Random Platform Errors…
- Added Generations: Oct 20th, 2016
- Oct 21st Comments:
  - sipa: please fix size to 2^n and use bitmask, (a%n) is slow
- Nov 9th Comments:
  - More esoteric build errors with deprecated flags
- PR Merged: Dec 14th, 2016
- Power of 2 restriction lifted by sipa: Jan 12th, 2017 -> (a%b) ~~ (((uint64_t) a)*n)>>32

覚えておくべき点：
Takeaways:

- Bitcoinではアルゴリズムの改善できるところは多い
- 全部はトレードオフでは無い！
- 測定基準を**先に**決める
  - 最初はこのコードがあまり改善にならないと思って、捨てようと思った ☺
- レビューされるの時に諦めないで
  - 長い時間かかるから！

- Bitcoin is ripe with opportunity for purely algorithmic improvements
- Not everything needs to be a tradeoff!
- Set up metrics **first**
  - I almost threw away this code because I wasn't sure if it would help much ☺
- Don't Get Demotivated in Review
  - It takes a long time!

コード

Code

- https://github.com/bitcoin/bitcoin/pull/8895

ボーナス

Bonus Section

BITCOINの紳士録
BITCOIN WHO'S WHO

laanwj

- **Wladimir J. van der Laan**
- Bitcoin Core の Lead Maintainer
- MIT DCIの研究員

- **Wladimir J. van der Laan**
- Bitcoin Core Lead Maintainer
- Employed by MIT DCI

sipa

- **Pieter Wuille**
- 大量な貢献のBitcoinデベロッパー
  - SegWitの発明者
- BlockStream Co-Founder
- http://pieterwuillefacts.com
  - Pieter Wuille が1回SHA-2を反転した。でもそれは無理との証明だ。

- **Pieter Wuille**
- Prolific Bitcoin Developer
  - Made SegWit
- BlockStream co-founder
- http://pieterwuillefacts.com
  - Pieter Wuille once inverted SHA-2. But that just proves that it's impossible.

gavinandresen

- **Gavin Andresen**
- 元 Lead Maintainer
- 元 MIT DCI
- 今の活動: ???

- **Gavin Andresen**
- Ex-Lead Maintainer
- Ex-MIT DCI
- Current Activities: ???

theuni

- **Cory Fields**
- Bitcoin Core デベロッパー, ビルドシステムのエクスパート
- MIT DCIの研究員
- 良く言われる:
  - “Coryが居なくなったら誰もバイナリービルドできない”

- **Cory Fields**
- Bitcoin Core Developer, build system guru
- Employed by MIT DCI
- Oft Repeated:
  - “No one would be able to build the bitcoin binary if Cory went away”

TheBlueMatt

- **Matt Corallo**
- Bitcoin Core デベロッパー
- BlockStream Co-Founder
- Relay Network のエクスパート
  – Compact Blocksの発明者
- 髪がとてもブルー

- **Matt Corallo**
- Bitcoin Core Developer
- BlockStream co-founder
- Relay Network Guru
  – invented Compact Blocks
- Very Blue Hair

jonasschnelli

- **Jonas Schnelli**
- ウォレットとUXのコードに注目する
- Bitcoinのハードウェアウォレットを作っている

- **Jonas Schnelli**
- Largely works on wallet & UX code
- Building a bitcoin hardware wallet

morcos & sdaftuar

- **Alex Morcos & Suhas Daftuar**
- Chaincode Labs の Founder

- **Alex Morcos & Suhas Daftuar**
- Founders of Chaincode Labs

gmaxwell

- **Greg Maxwell**
- あなたが考えたことは全部 Greg が既に発明し、bitcointalk に投稿した ☺
- BlockStream の Co-Founder

- **Greg Maxwell**
- Almost anything you can think of, Greg invented it in a post on bitcointalk ☺
- BlockStream co-founder

petertodd

- **Peter Todd**
- Bitcoinの天邪鬼

- **Peter Todd**
- Bitcoin’s Top Contrarian

Adam Back

- BlockStream CEO
- デベロッパーじゃないけど、一流科学者である
- HashCashの発明者!

- BlockStream CEO
- Not a developer, but a leading scientist
- Invented HashCash!

Satoshi Nakamoto

???

☺

ここで名前書いてない人々
Omitted People

- もっといるですが、上記の名前を見たら遠慮なく声をかけてね！

- There are lots more, but the above are names you'll see around & can ask for help!