Ruby の GC の問題点と改善手法についての一考察

A study on issues and improvements on Ruby's Garbage Collection

Koichi Sasada

Heroku, Inc.

ko1@heroku.com

自己紹介

- 所属
  - Asakusa.rb（初期から）
  - Ruby Association（理事）
  - 日本 Ruby の会（一般会員）
  - Heroku, Inc.（MTS）
  - Ruby（CRuby/MRI）コミッタ
    - 趣味で Ruby を書いています
    - 仮想マシンなどの基盤部分の担当
    - コミット数は少ない

PROGRAMMING
Language

Ruby 2.0.0 Release!!

- Ruby 2.0.0 was released!!
  - 2013/02/24 (20th anniversary of Ruby lang.)
  - Rubyist Magazine Ruby 2.0.0 特集
    - 日本語
    - English!!（昨日）
    - お陰様で充実した特集になったと思います
  - VM 回りを整理しました
    - あまりドラスティックには弄っていない
    - ≒ Ruby 1.9.4 相当
    - 互換性！！
      - mame 曰く：2.0 のような、互換性が気になる節目に互換性を売りにしてよかった。...
  - デバッガ API 回りを整理
    - debugger2 (experimental release)

heroku

How it Works | Pricing | Add-ons | Dev Ce

Blog

Matz on Ruby 2.0 at Heroku's Waza

by Craig - Mar 06

Matz, the creator of Ruby, spoke at Waza for the 20th anniversary of the language and the release of Ruby 2.0. If you weren't in the sold out crowd, not to worry. Information should flow free and experiences should be shared; in line with those concepts you can watch Matz's talk right here, then read about what's new in this version of Ruby and how to run it on Heroku.

*With slides available on **speakerdeck***

Keep reading for more information on Ruby 2.0 or check out our first batch of **videos from Waza 2013**. To stay up to date as we post new videos, follow us **@heroku**.

Running 2.0 on Heroku

If you're interested in taking advantage of these new features give it a try on Heroku today. To run Ruby 2.0 on Heroku you'll need this line in your `Gemfile`:

```
ruby "2.0.0"
```

Then commit to git:

```
$ git add .
$ git commit -m "Using Ruby 2.0 in production"
```

We recommend that you test your app using 2.0 locally and deploy to a staging app before pushing to production. Now when you `$ git push heroku master` our Ruby buildpack will see that you've declared your Ruby version and make sure you get the right one.

もちろん Heroku でも使えます！

20 years of simplicity, elegance, and programmer happiness

Heroku, since its founding, has been aligned with the key values of Ruby – simplicity, elegance, and programmer happiness. Heroku still believes in the power and flexibility of Ruby, and we've invested in the language by hiring **Yukihiro "Matz" Matsumoto**, **Koichi Sasada** and **Nobuyoshi Nakada**. We would like to thank them and the whole Ruby core team for making this release happen. Join us in celebrating Ruby's successes and in looking forward to the next twenty years by trying Ruby 2.0 on Heroku today.

オレオレ

（リンク先間違ってるんだけど…）

Ruby apps are running using 1.8.7, you should upgrade. Ruby 1.8.7 is approaching End of Life (EOL) in three months on June 2013. EOL for Ruby 1.8.7 means no security or bug patches will be provided by the maintainers. Not upgrading means you're potentially opening up your application and your users to vulnerabilities. Don't wait till the final hour, upgrade now to be confident and secure.

Speed

Ruby 2.0 has a faster garbage collector and is **Copy on Write** friendly. Copy on Write or COW is an optimization that can reduce the memory footprint of a Ruby process when it is copied. Instead of allocating duplicate memory when a process is forked, COW allows multiple processes to share the same memory until one of the processes needs to modify a piece of information. Depending on the program, this optimization can dramatically reduce the amount of memory used to run multiple processes. Most Ruby programs are memory bound, so reducing your memory footprint with Ruby 2.0 may allow you to run more processes in fewer dynos.

*If you're not already running a concurrent backend consider trying the **Unicorn web server**.*

Features

In addition to running faster than 1.9.3, and having a smaller footprint, Ruby 2.0 has a number of new features added to the language including:

Mention about “Speed”

Ruby 2.0 has a faster **garbage collector** and is **Copy on Write** frien
optimization th
a Ruby proces
duplicate mem
allows multiple processes to share the same memory until one of the processes needs to modify a piece of information. Depending on the program, this optimization can dramatically reduce the amount of memory used to run multiple processes. Most Ruby programs are memory bound, so reducing your memory footprint with Ruby 2.0 may allow you to run more processes i

If you’re not alrea
consider trying the Unicorn web server.

要約：GC が bitmap marking になって
CoW friendly になってよくなったよ

要約：Unicorn 使ってみてね！

本発表の概要

- 概要
  - 現在の Ruby の GC について問題点を整理し、実装の改善手法について述べ、今後の展望を述べる。
  - 主に CRuby/MRI について示す。
- Agenda
  1. Ruby の GC の概要
  2. Ruby の GC の問題点
  3. Ruby の GC の改善案の検討
  4. Ruby の GC の今後の展望

本発表の趣旨

- コンセプト
  - @kakutani says “「生活発表会」はテーマというか開催コンセプトですねえ。Asakusa.rb(周辺)の人が、Asakusa.rbとそうでない人に向けて、それぞれ普段やってることや興味を持ってること話すので、温かく見守ってくださいね、という。”

    https://twitter.com/kakutani/status/299131003593687041
  - ということで、笹田が普段興味をもって検討していることをお話しします。暖かく見守って下さい。本当に、いつもいつもいつもいつも、こういう事を考えています。電車の中で退屈しません。
- 本発表が対象とする人
  - Ruby プログラミングをしていて、GC に不満がある人
  - Ruby 処理系に興味がある人
  - Ruby の処理系を改善したいと考えている人
  - 笹田は金もらってるのに、本当に仕事してるのか、興味がある人

本発表の趣旨

- 本発表は、いかに中村成洋(@nari3)さんが素晴らしい仕事をされたかをまとめることで、中村さんの貢献に感謝し、中村さんのトリの発表の露払いをすることです。

(浅草のおもてなし)

(怖くないコミッタ)

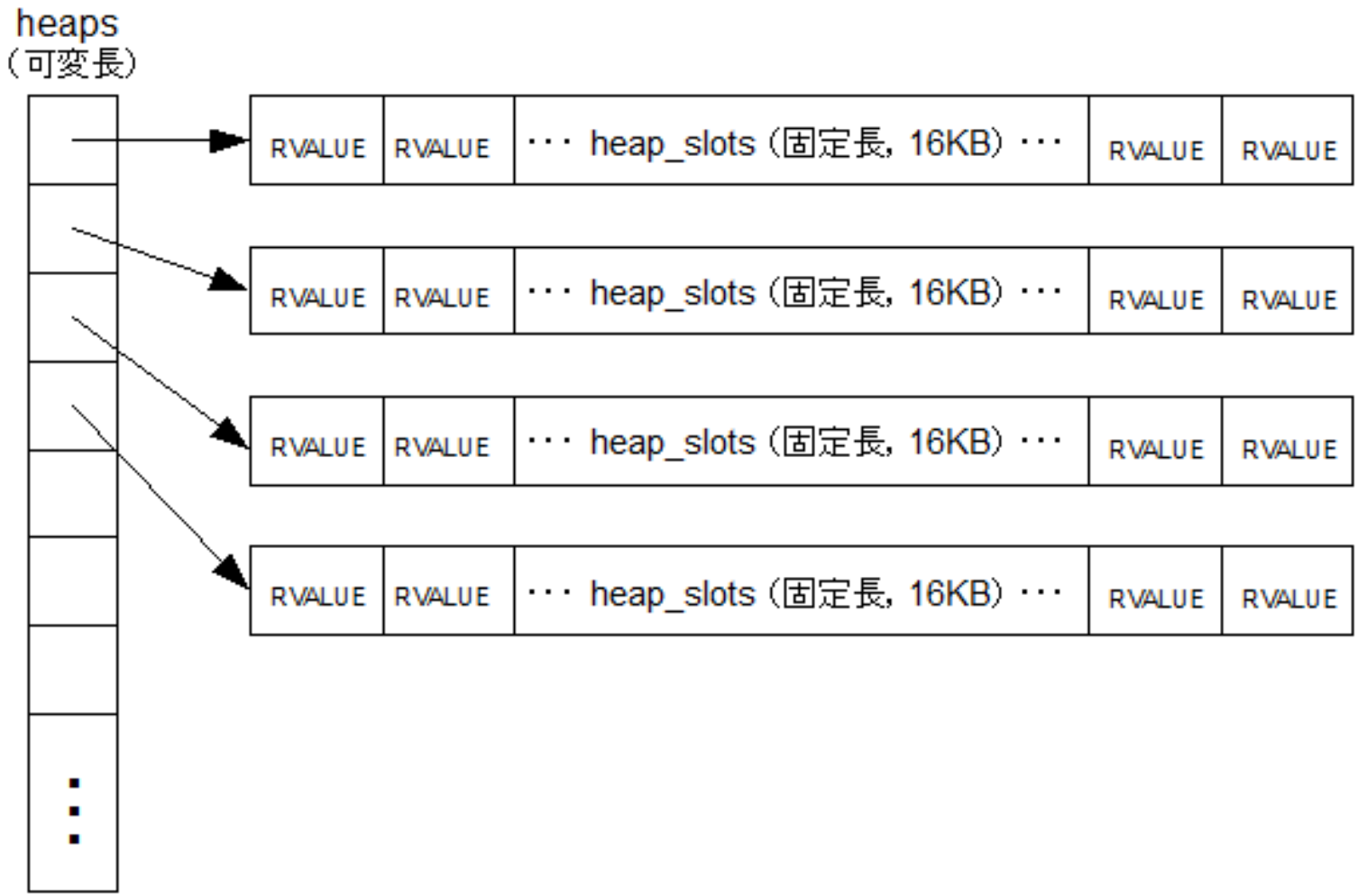
(1) Ruby の GC の概要

- 基本的には Mark & Sweep
- いくつかの特長
  - Conservative (元々)
  - Lazy Sweep (from Ruby 1.9.3)
  - Bitmap marking (from Ruby 2.0)
  - Stack-less marking (from Ruby 2.0)

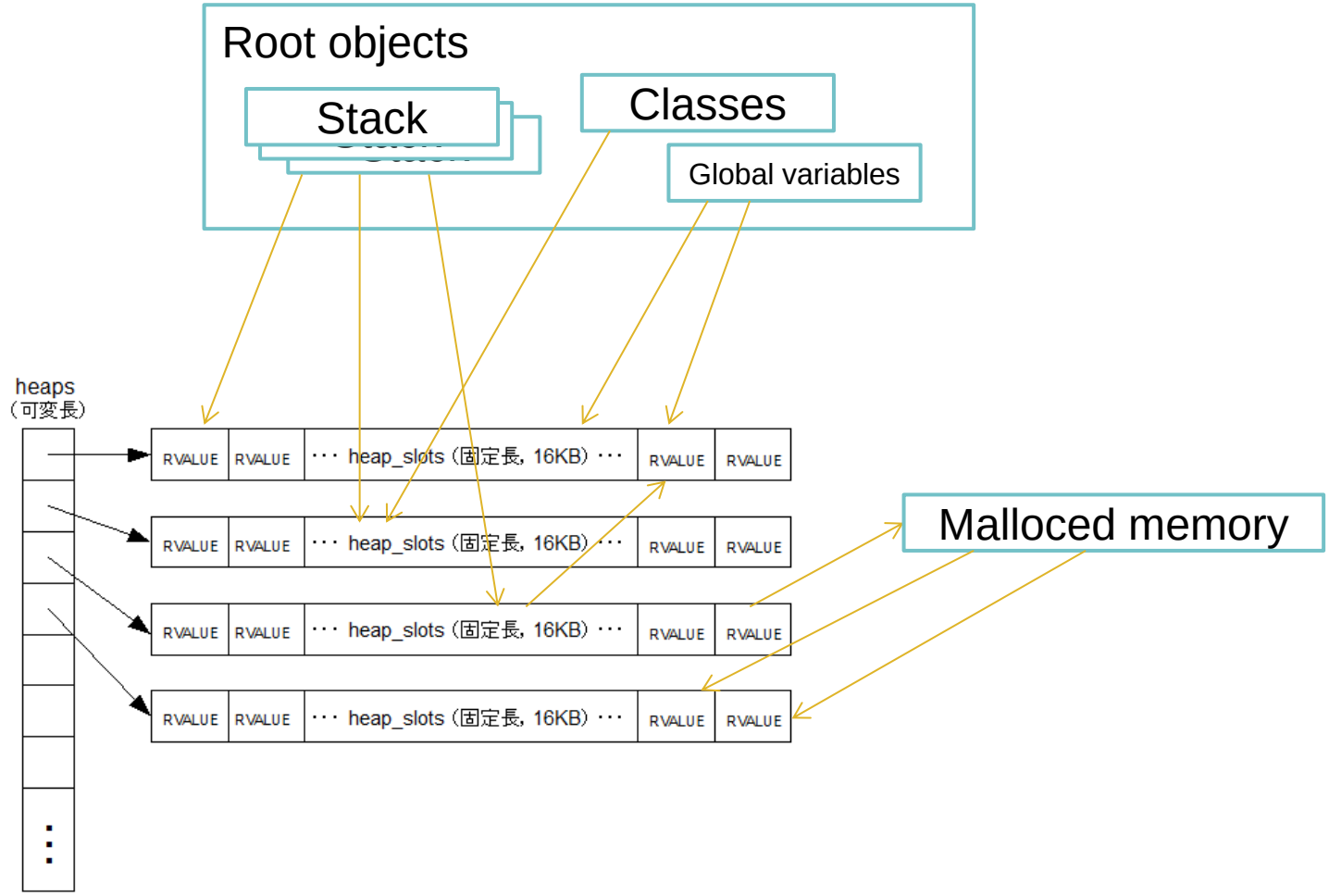
Mark & Sweep

- マークしてスイープ
  - 辿ることのできるオブジェクトを mark
  - 辿られなかったオブジェクト (!marked) を sweep(回収)
  - 単純：簡単（世界で最初に作られたGC）

Rubyのメモリ管理
heaps と heap_slots



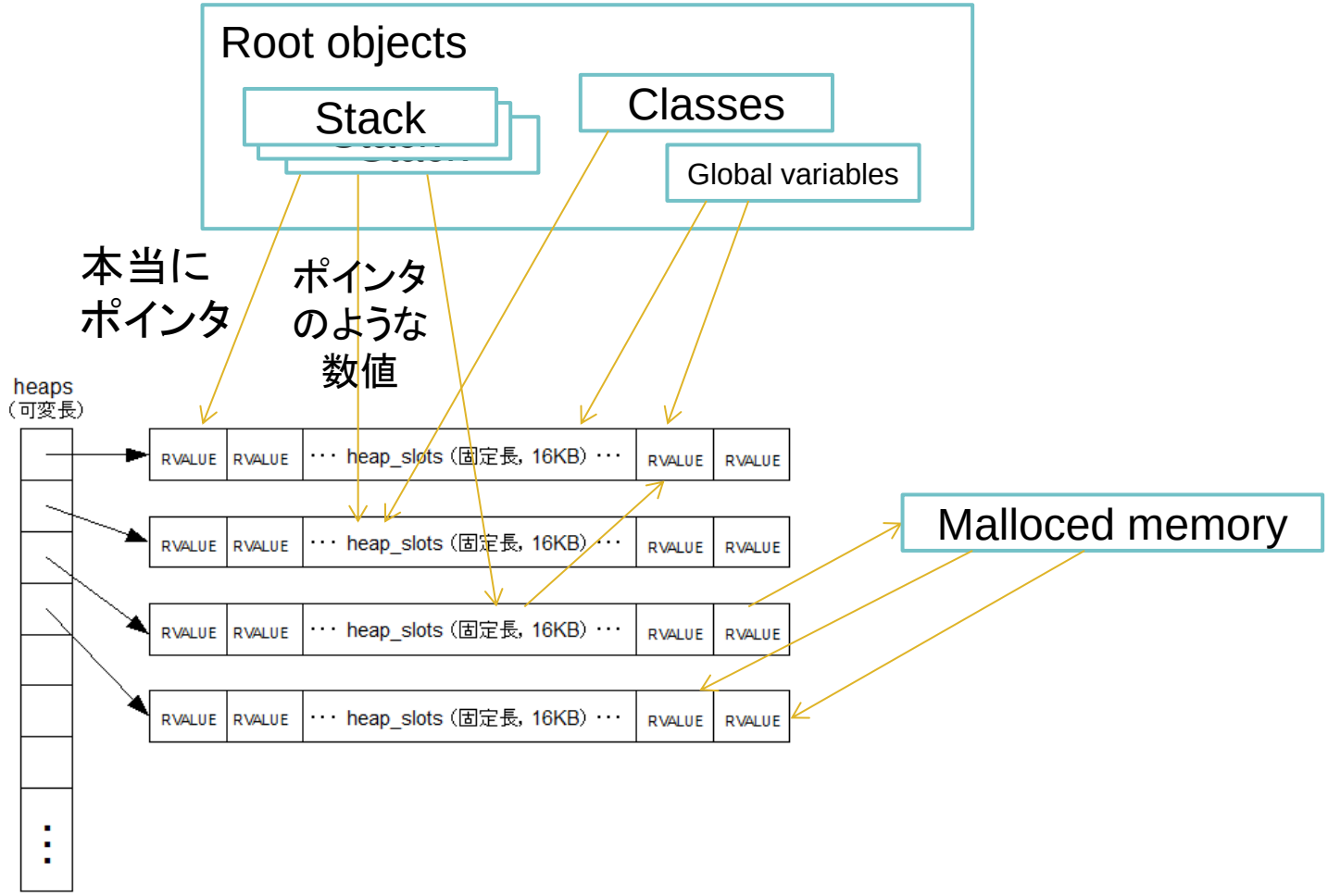
Ruby の Mark&Sweep



Conservative Mark&Sweep

- Ruby の M&S は保守的
  - ベタメモリの中に何があるかわからない
  - オブジェクトへのポインタっぽいものがあれば、それをオブジェクトへのポインタとして取り扱ってしまおう
- 利点
  - C プログラム中にテキトーに Ruby のオブジェクトを作成して作っておいても、テキトーにマークしてくれる。
  - とくに、C のマシンスタックをマークするのは便利
  - 他の処理系では、マクロとか沢山使わないといけない
  - C で処理が書きやすい！→ C Friendly!!
- 欠点
  - 死んでいるオブジェクトを「生きている」と判断してしまう可能性がある（でも、滅多にないらしい）
  - 難しいバグを生む（後述）

Ruby の Conservative Mark&Sweep

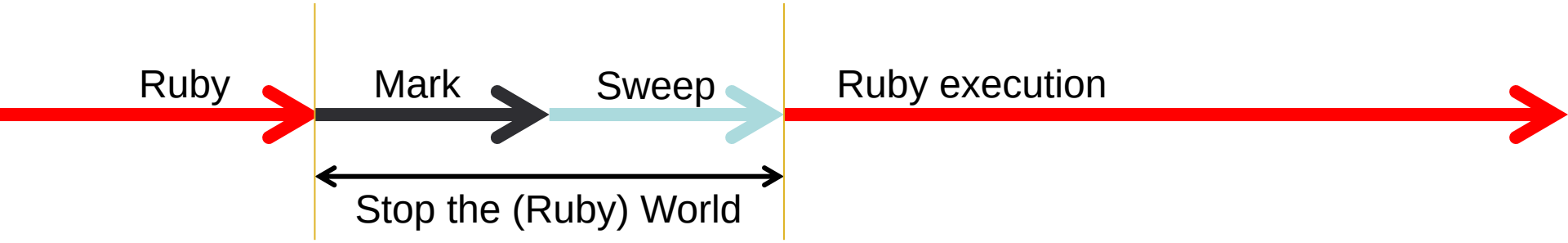


Lazy sweep

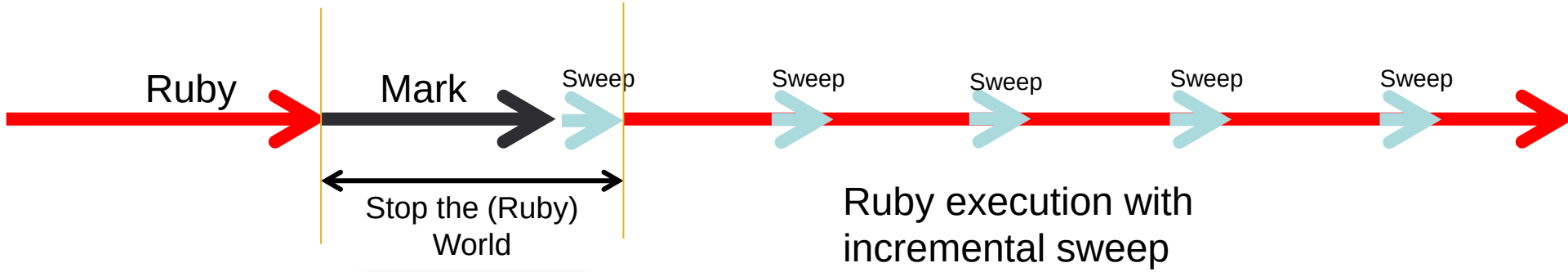
- mark と sweep をまとめて行うと、GC で止まる時間が長くなる → アプリが止まってしまうように見える
- sweep は後でやっても問題無いから、sweep を細切れに実行しよう

Lazy sweep

- **Before 1.9.3**: Stop the world mark and sweep



- **After 1.9.3**: Stop the world mark, and incremental sweep



Shorter stop time

Lazy sweep

- 利点
  - 1回の GC の停止時間が短くなる
  - キャッシュのローカリティが高くなる（可能性がある）
- 欠点
  - メモリ解放が遅れるため、プロセスが消費するメモリ量が大きくなる可能性がある
  - スループット（全実行時間）は変わらない（可能性がある）

Bitmap marking from Ruby 2.0.0

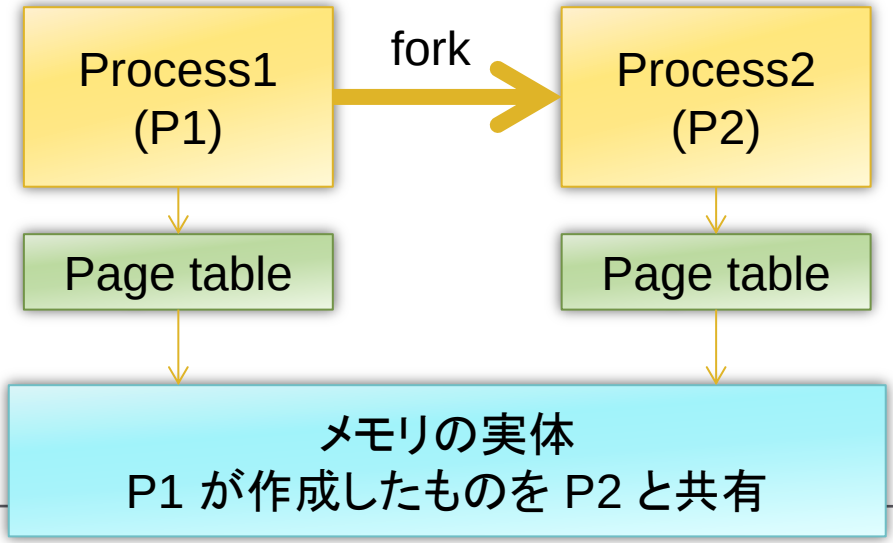
- mark bit を object と分離することで、fork 時の CoW friendly にする

  CoW friendly...?
  そもそも fork なんてする人いるの？
  僕 Windows ユーザなんですが...

- REE (Ruby Enterprise Edition) が採用

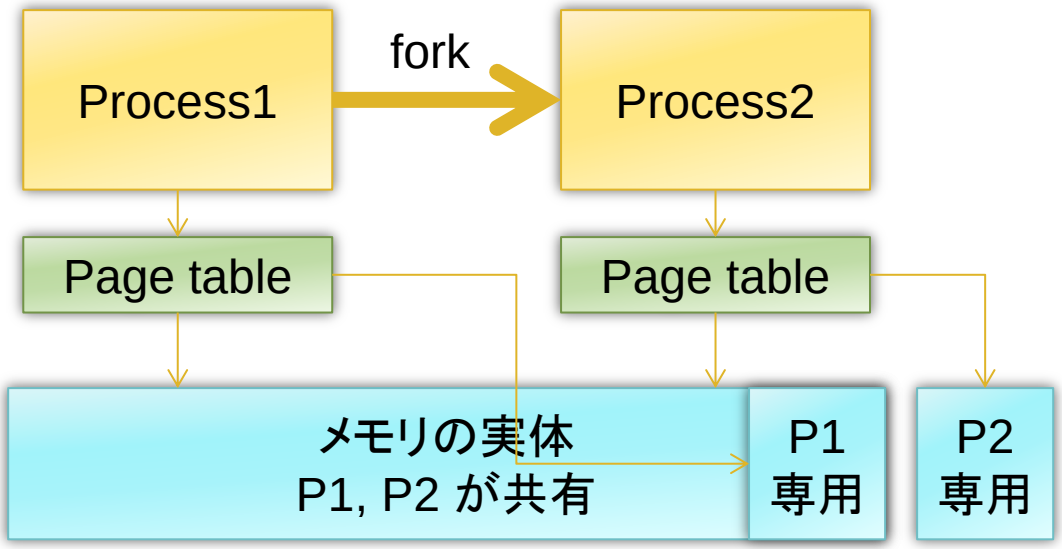
Bitmap marking CoW friendly?

- forkシステムコール（Unix）を行うと、全く同じメモリ空間のコピーを持つプロセスが生成される
- メモリコピーするのは無駄なので、メモリは共有させておく
  - 具体的には、ページテーブルが同じメモリの実体をさす



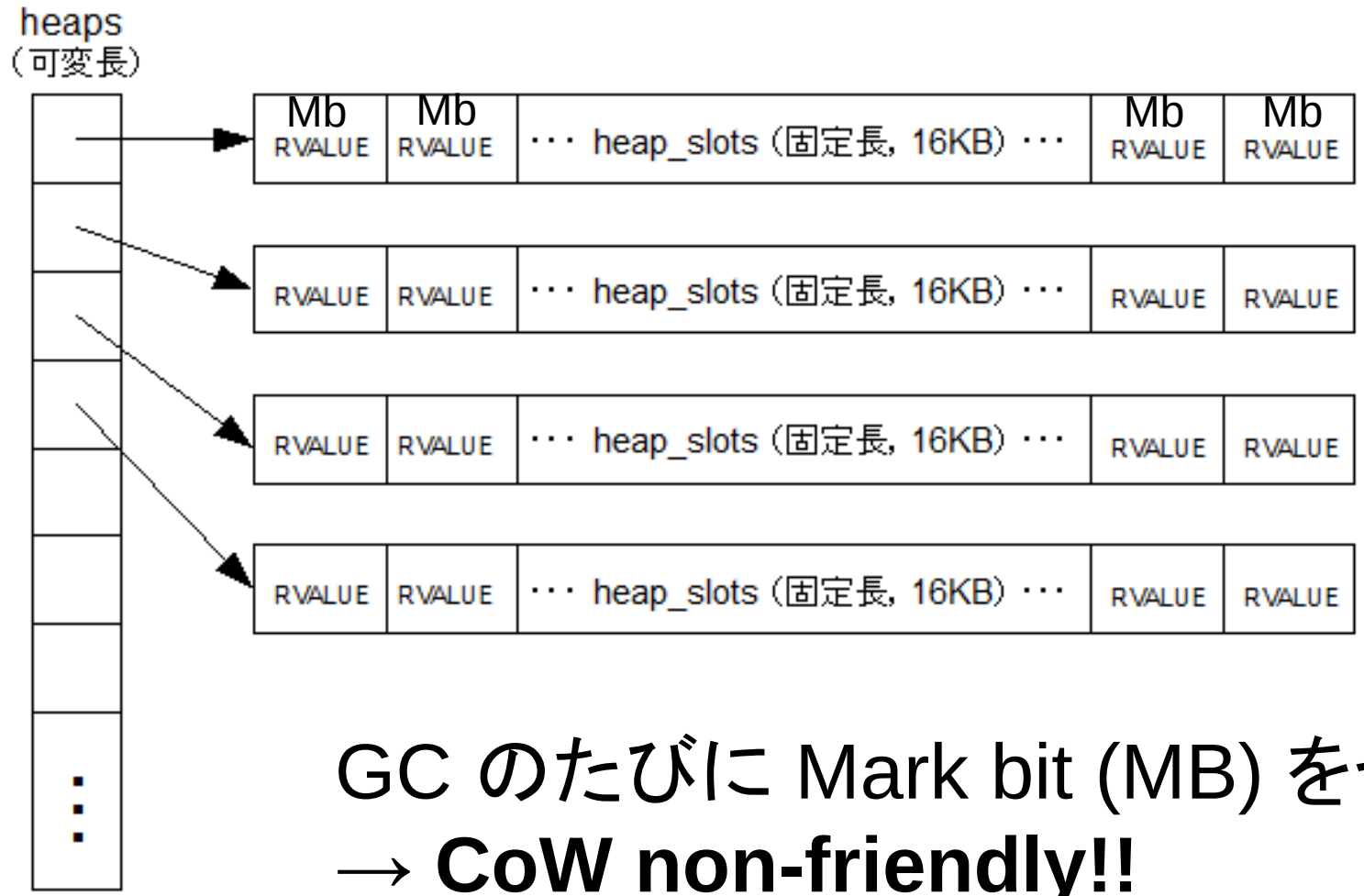
Bitmap marking
CoW friendly?

- でも、実際は別々なので、どちらかのプロセスが何かしら書き込みがあった時に書き込んだ場所だけ実際のコピーを行う（Copy on Write）



- CoW 起こりづらい
= なるべく共有する
= CoW friendly
- fork したプロセス群の総メモリ利用量に影響
（だから Unicorn）

Bitmap marking
従来の Ruby の GC は？



GC のたびに Mark bit (MB) をセット

→ **CoW non-friendly!!**

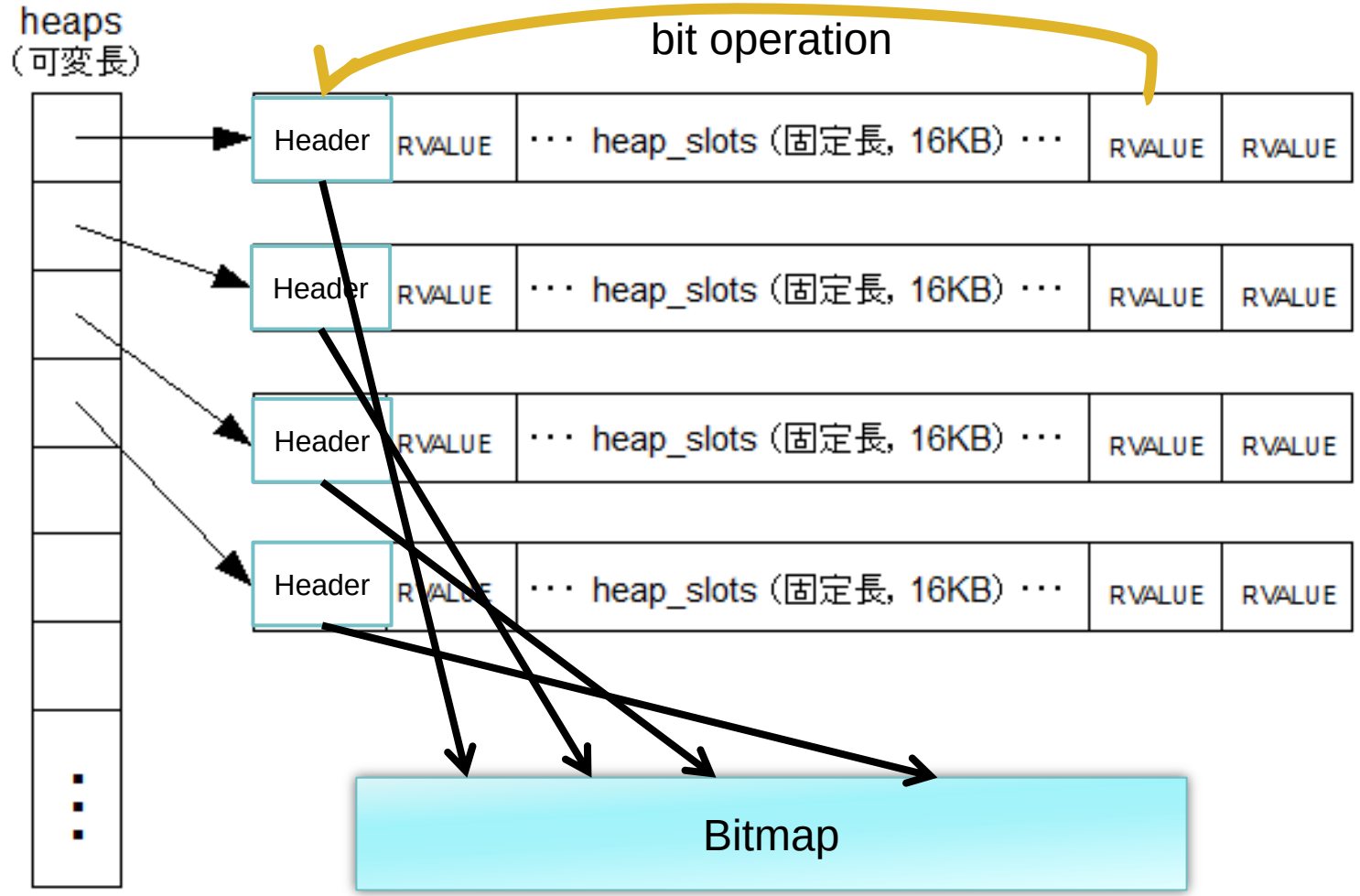
Bitmap marking
解決法

- Mark bit を各オブジェクトごとに持たせるのではなく、Mark bit をどこかに持たせればよい

  → まとめました！

  → Bitmap!!

Bitmap marking
Bitmap の探索手法



Bitmap marking

- 利点
  - CoW friendly になり、fork する人が幸せに
  - キャッシュ効率 up!
    - mark 時、write する箇所が一箇所に固まる
    - sweep 時、read する箇所が一箇所に固まる
    - ただし、キャッシュ効率について詳しい評価結果を聞いていない
- 欠点
  - bit の場所の計算に時間がかかるかも（ただし、キャッシュ効率に比べたら問題無い、はず）

Bitmap marking DEMO

- Bitmap っていったら画像？

Non-recursive GC from Ruby 2.0.0

- 従来、mark 処理は再帰関数で書いていた

```
mark(obj) {
  mark(obj->ref1);
  mark(obj->ref2);
}
```

- マシンスタックオーバーフローが起こったりした
  - とくに、Fiber 利用時に起こっていた
    - ただし、Ruby 2.0.0 から、Fiber のマシンスタックのデフォルト値は大きくなったので、あまり問題ないかも（設定可能）
- 再帰じゃなくした！（そのまんま）

Non-recursive GC algorithm

```
mark(obj){
  stack.push(obj->ref1);
  stack.push(obj->ref2);
  ...
}
mark_all(){
  root_objects.each{|obj| mark(obj);}
  while(!stack.empty?){
    mark(stack.pop);
  }
}
```

Non-recursive GC

- 利点
  - スタックオーバーフローが起こらなくなった
  - 並列 mark にそのまま拡張できる
- 欠点
  - 再帰のほうが速かったりしない？
  - GC 中にスタックのアロケーション起きたらどうなる？

(2) Ruby の GC の問題点

- 効率が悪い
  - 遅い
    - ポーズタイムが長い
    - スループットが悪い
  - メモリ沢山食う

(2) Ruby の GC の問題点

- 世代別じゃない
  - 世代別 GC
  - 多くのオブジェクトは短寿命→短寿命だけGC
- compaction しないためフラグメンテーション発生
  - 細切れになった領域はメモリの再利用がしづらい
  - コピー GC などを利用すれば ok
- 保守的 GC に起因するバグ
  - コンパイラの最適化によって mark されたりされなかったり…
- その他…

全部ひっくるめて、CRuby の C-Friendly さが足かせ！（保守的GCの戦略）

なぜ C-friendly が足かせに？

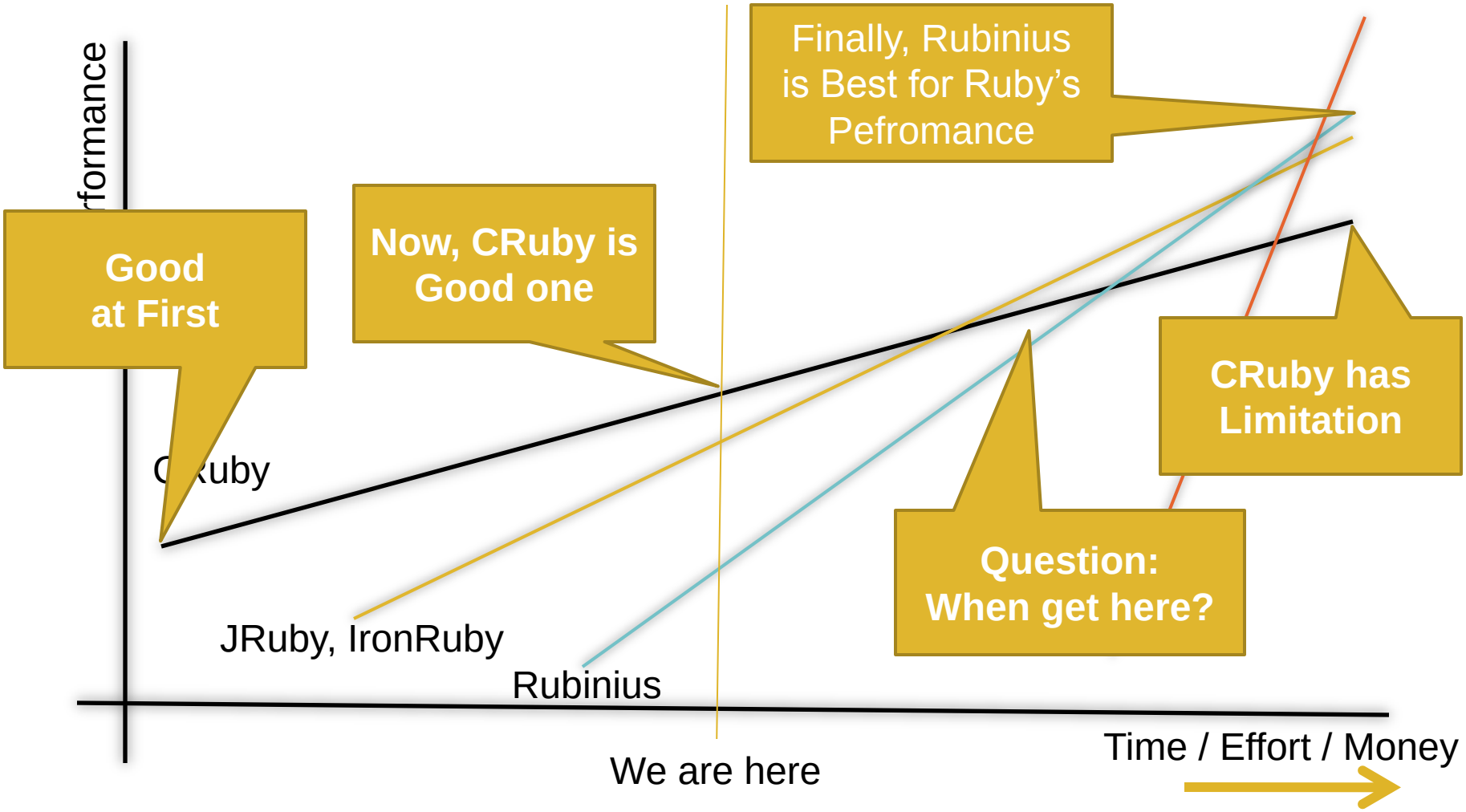
- 世代別GC
  - **ライトバリアが必要**
- コピーGC（compaction）
  - **ライトバリア等が必要**
- インクリメンタルGC
  - **ライトバリアが必要**
- バグの無い mark 処理
  - **メモリ上のアドレスのアノテーションが必要**

これらは、C プログラムに
何か手を加えないと無理！！

C-friendly と対立する Interpreter-friendly

- C で拡張が書きやすい → 嬉しい
  - 初期は C で性能が稼ぎやすい
  - C のライブラリに乗っかりやすい
- C で書きやすい → C 拡張が増える（現状）
- C で書いた拡張があると GC と相性が悪い
- 最終的に性能が出づらい

Evolution of VM Performance
My Prediction



(3) Ruby の GC の改善案の検討

いろいろな改善案

- …
- …

（事情により削除）

世代別GC の導入と valgrind を用いた世代別GC導入支援

- 世代別 GC を導入するには write-barrier が必須
  - write-barrier が何か、は今回は省略しています
- C のプログラムに write-barrier を適切に挿入する（過不足無く挿入する）のは一般的に困難
  - 間違って挿入し忘れると実行に問題
  - 間違って挿入すると性能と実行に問題
  - しかも、問題はなかなか気づけない

困難な例

Wed May 13 22:34:31 2009 Narihiro Nakamura <authorNari@gmail.com>

* gc.c: **add longlife garbage** collection. [ruby-dev:38423]

≒制限つき世代別GC

...

Sat May 16 17:26:04 2009 Narihiro Nakamura <authorNari@gmail.com>

* iseq.c (rb_iseq_clone): use longlife object and insert write barrier.

...

Mon Aug 10 10:57:59 2009 Narihiro Nakamura <authorNari@gmail.com>

* gc.c: reject unused longlife gc. longlife gc target is longlife NODE by method table and vm inline cache. but, fixed it at r24085, r24128. so I **rejected** longlife gc.

世代別GC の導入と valgrind を用いた世代別GC導入支援

- 制限つき世代別GCは、いくつかの面で有効
  - いくつかのクラス、場面に割り切って導入
- ただし、導入が困難
  - 前述したとおり、適切に導入するのは困難

→ この導入を支援するツールの開発が必要

→ valgrind を用いたツールの開発

事情によりスキップ

(4) Ruby の GC の今後の展望

- いろいろやることがあって
- わたしはお金をもらって Ruby 開発ができる

- これから頑張ります

まとめ

- Ruby の GC だいぶは良くなった
  - Thanks, Nari-san
- でも、Ruby の GC はひどい
  - Ruby 2.0.0 も、まだまだひどい
  - 良いと思っていた “C-friendly” が足かせに
- なんとかする方法を検討中
  - いつもこういうことを考えています
  - 仕事してるよ！
- Ruby 2.1 (or later) にご期待下さい

Thank you!

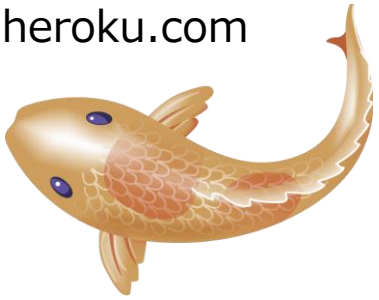
Ruby の GC の問題点と
改善手法についての一考察

A study on issues and improvements
on Ruby's Garbage Collection

Koichi Sasada

Heroku, Inc.

ko1@heroku.com



←To be continued...