

RubyGCを どげんかせんと いかん

発表者

authorNari

注意

これから喋る事は
マニアックすぎて
ついていけない
可能性があります

ご了承ください
m(_ _)m

ここで
緊急アンケート

CRubyの
gc.cを
読んだ事がある方
ノシ

GC

超興味あるよ！

と言う方

ノシ

ご協力

ありがとうございます

m (' - ') m

「サブリミナル」



本題に入る前に

RubyGCは
なぜ改善せねば
ならないのか？

_ [Ruby] GCの改善について

あちこちでRuby(MRI)のGCについてけなされている。

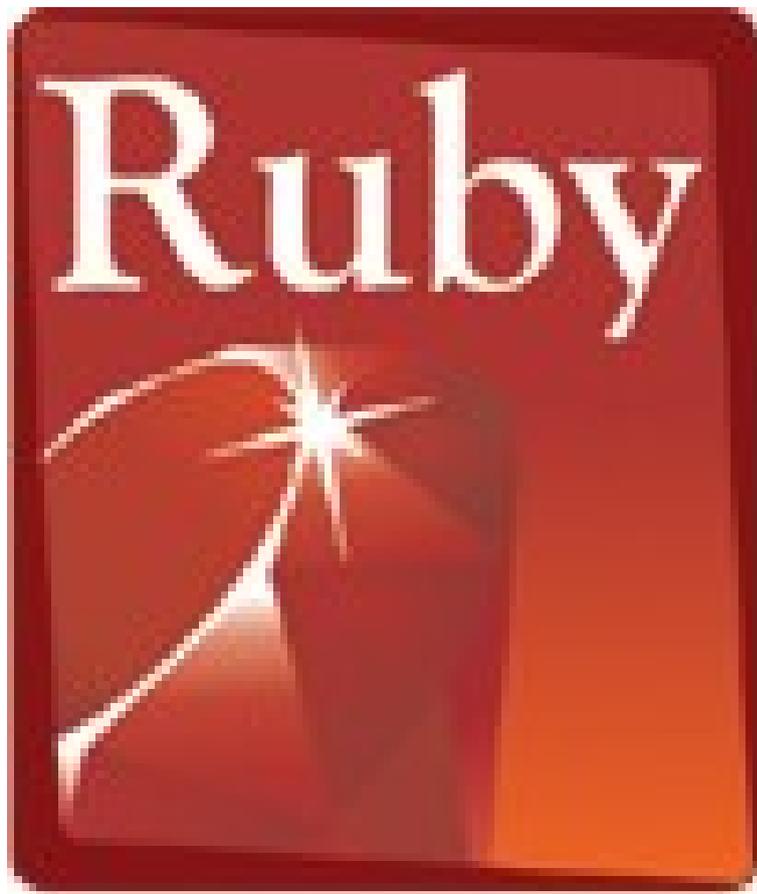
Matz日記より抜粋

why?

その理由は

3つのRubyの実装
を比較すると
わかりやすい

まずは



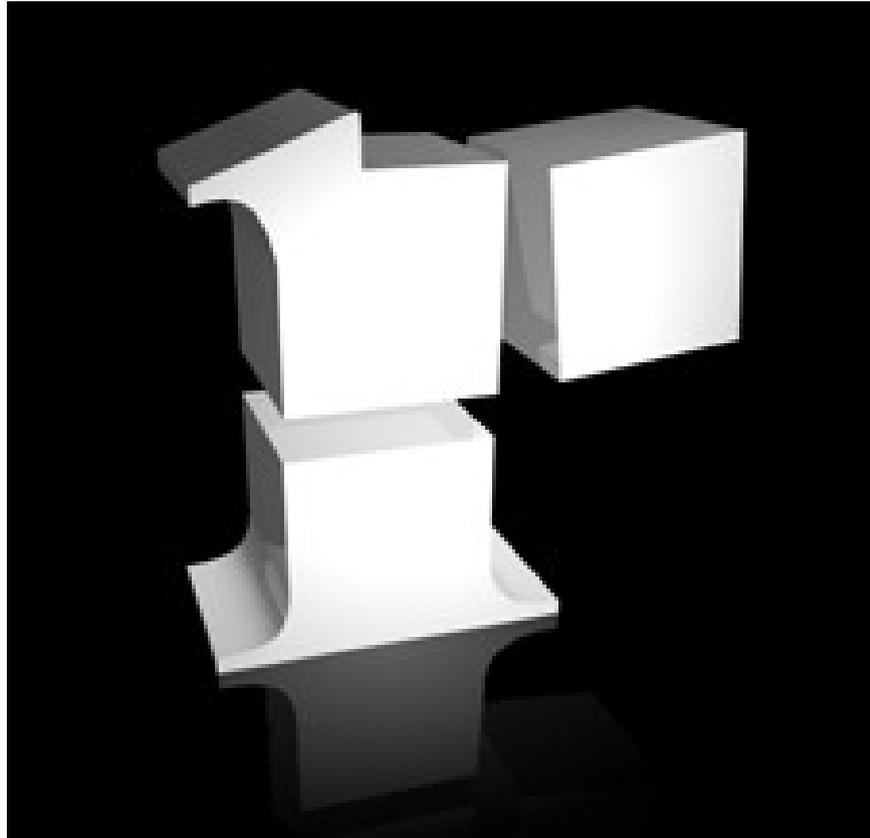
CRuby
(MRI)

まつもとゆきひろ氏
によって開発され
はじめたC言語による
Ruby実装

Ruby = CRuby

という事で
大体いいんじゃない
でしょうか

次に



Rubinius

コアな部分はC言語
その他の大半部は
Rubyで書かれた
Rubyの実装

最後に



JR Ruby

J Ruby

100%Java
によって書かれた
Rubyの実装

有名なので
ご存知の方も
多いはず

これら
3つは

外はRubyでも
中身は全然違います

つまり

GCの実装も
まったく違う

これらを
GCの目線で
比較する

この理由がわかる

_ [Ruby] GCの改善について

あちこちでRuby(MRI)のGCについてけなされている。

Matz日記より抜粋

JRubyのGC

mark-and-compact
copying

incremental gc

concurrent gc

generational gc

parallel gc



RubiniusのGC

mark-and-compact
copying
generational gc



CRubyのGC

mark-and-sweep



こっ、これは。。。。

JRuby

Rubinius



CRuby



Matz



な、なるほど。。。。

_ [Ruby] GCの改善について

あちこちでRuby(MRI)のGCについてけなされている。

ということでは

RubyGCを どげんかせんと いかん

発表者

authorNari

遅れましたが
自己紹介

authorNari(nari, nari3)
というか中村です

- 出身

福岡出身(宮崎とは無関係)

- 所属

ネットワーク応用通信研究所

- Wiki

GCアルゴリズム詳細

http://wiki.livedoor.jp/author_nari/d/GC

・ブログ

I'm Cruby

<http://d.hatena.ne.jp/authorNari/>

[ウェブ](#) [画像](#) [地図](#) [ニュース](#) [グループ](#) [Gmail](#) [more ▼](#)

Google™

Ruby GC バグ

検索

[検索オプション](#)
[表示設定](#)

ウェブ全体から検索 日本語のページを検索

ウェブ

続きはWebで

[lazysweepのバグ取れた - I an](#)

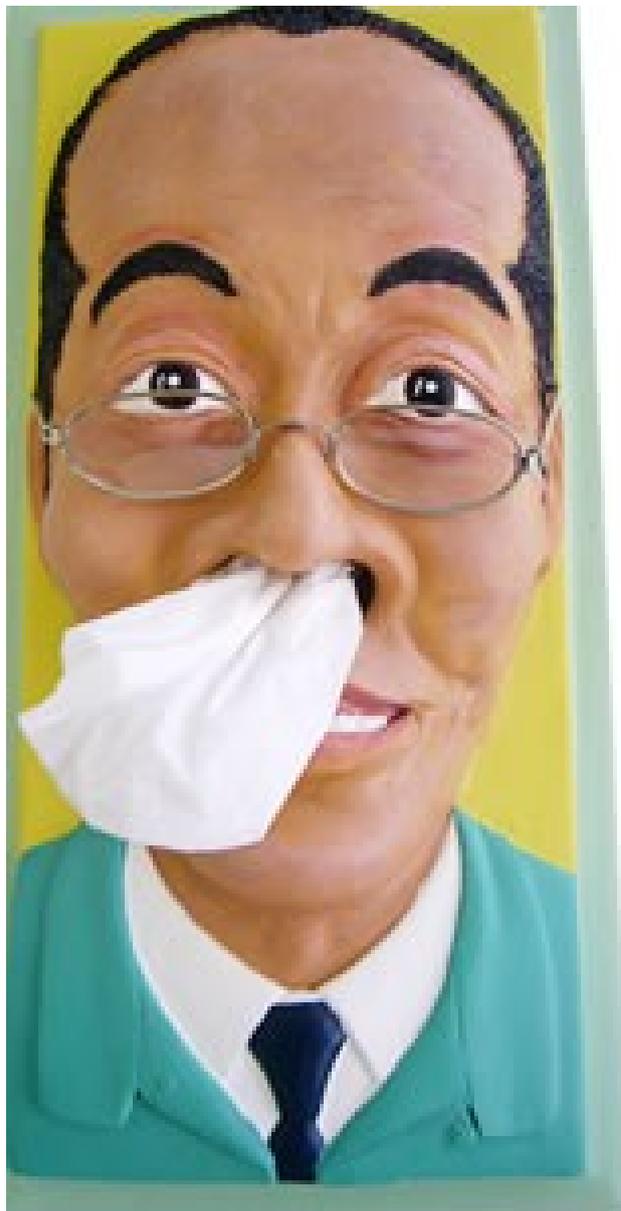
[GC][Ruby]lazysweepのバグ取れた · [Ruby][GC]LazySweepを更新というか全書き換え · [抜粋] 幻惑の死と使途 - 森博嗣 · [Ruby]Rubyist九州会議 2008 · [GC][Ruby]lazysweep作りなおし · [GC]MoshとかGaucheとか · [etc]無職の一日 · [GC]ircnet#gc ...

d.hatena.ne.jp/authorNari/20080521/1211335348 - 35k - [キャッシュ](#) - [関連ページ](#) - [メモをとる](#)

- GC研究歴
半年くらい

間違ったことを言うかもしれません
その時は教えてくださいm(__)m

サブリミナル



News!!

東国原知事が、
ティッシュケースに
なりました。

アジェンダ

アジェンダ

- ・ GCとは
- ・ GC愛
- ・ CRubyのGC詳細
- ・ 欠点と改善
- ・ これからやりたいこと

アジェンダ

- ・ GCとは
- ・ GC愛
- ・ CRubyのGC詳細
- ・ 欠点と改善
- ・ これからやりたいこと

GCとは

GC

(Garbage Collection)

ごみ

集め

GC

(Garbage Collection)

ごみ集め

何をやるものか？

プログラムが動的に
確保したメモリ領域のうち
不要になった領域を
自動的に解放する機能
by Wikipedia

わかりにくいので
簡単に言うと

ごみになったメモリ領域を
ヨロシクやってくれる
いい奴

最近の言語はほぼ標準搭載

- Java
- Ruby
- perl
- Haskell
- etc...

GCのおかげで
メモリ管理は
凄く楽になった

GC素晴らしい

アジェンダ

- ・ GCとは
- ・ GC愛
- ・ CRubyのGC詳細
- ・ 欠点と改善
- ・ これからやりたいこと

GC愛

「GC作りませんか？」
と聞いてみた

主な3つ回答

● ゴミ集めだもんね



● 地味だよね



● やりがいなさそう



そんな事ない！

GCに対する僕のイメージ

- ゴミ片付ける優しさ 
- 目立つ事のない奥ゆかしさ 
- 強烈なバグを出すお茶目さ
ある教授いわく
「GCにはかかわらない方がいい」 

すごく
やりがいがありそうじゃ
ないか！

そこで
お手頃なGICを
いじりましょう

アジェンダ

- ・ GC愛
- ・ CRubyのGC詳細
- ・ 欠点と改善
- ・ これからやりたいこと

CRubyのGC詳細

CRubyGCの
アルゴリズムは
Mark - and - Sweep

Rubyの サンプルプログラム を元に説明

サンプルプログラム

```
class Child; end
```

```
arry1 = []
```

```
arry1 << Child.new
```

```
arry2 = []
```

```
arry2 << Child.new
```

```
arry2 << Child.new
```

```
arry2 = nil
```

```
GC.start
```

実際にどう動くのか

ルート
オブジェクトへの
参照が格納され
ている
「可能性」のある
すべての領域

ヒープ領域
GCが管理する
データが確保さ
れる領域

arry1の
ポインタ

RArray



```
class Child; end
```

```
arry1 = []
```

```
arry1 << Child.new
```

```
arry2 = []
```

```
arry2 << Child.new
```

```
arry2 << Child.new
```

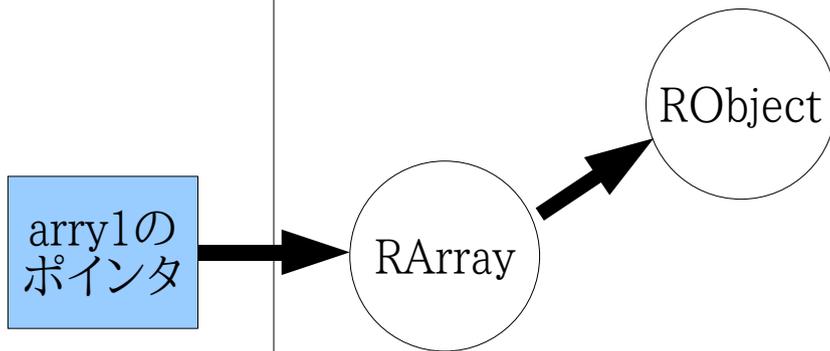
```
arry2 = nil
```

```
GC.start
```

RObject作成、RArray参照

ルート

ヒープ領域

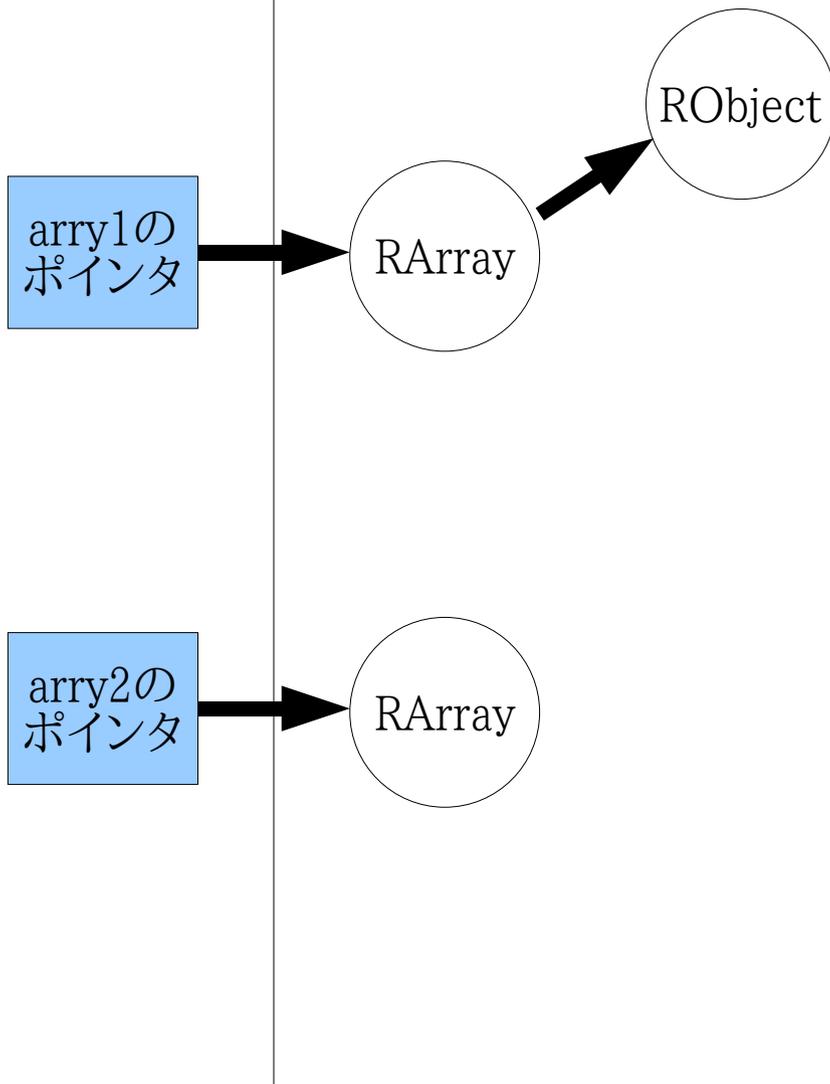


```
class Child; end  
  
arry1 = []  
arry1 << Child.new  
  
arry2 = []  
arry2 << Child.new  
arry2 << Child.new  
arry2 = nil  
  
GC.start
```

RArray作成

ルート

ヒープ領域



```
class Child; end
```

```
arry1 = []
```

```
arry1 << Child.new
```

```
arry2 = []
```

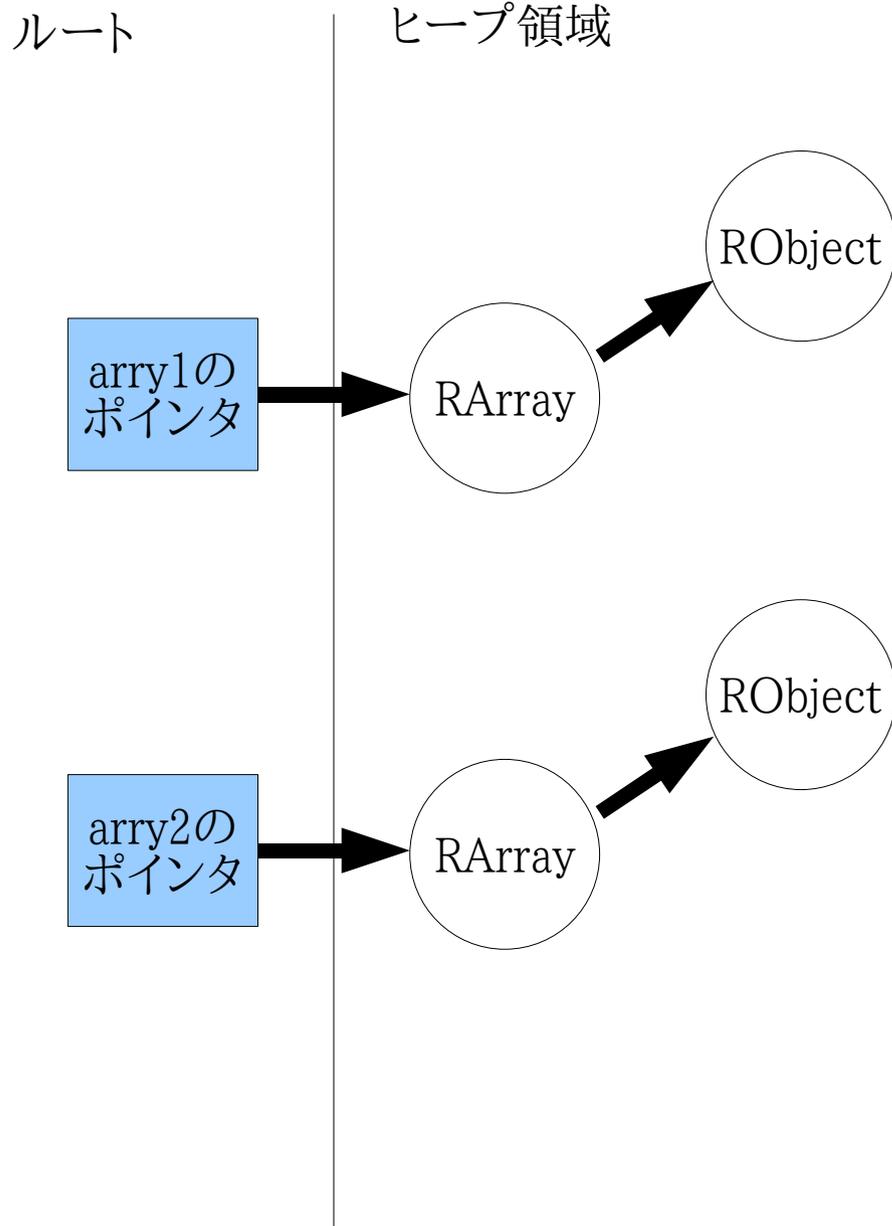
```
arry2 << Child.new
```

```
arry2 << Child.new
```

```
arry2 = nil
```

```
GC.start
```

RObject作成、RArray参照



```
class Child; end
```

```
arry1 = []
```

```
arry1 << Child.new
```

```
arry2 = []
```

```
arry2 << Child.new
```

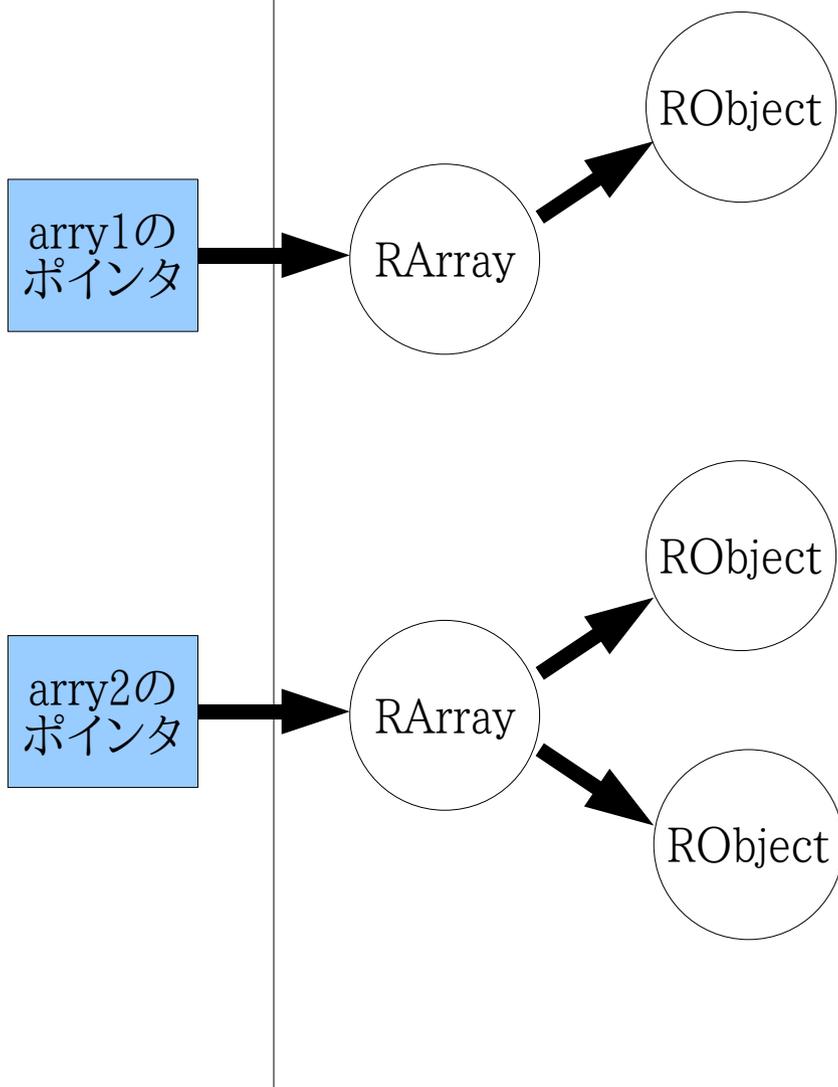
```
arry2 << Child.new
```

```
arry2 = nil
```

```
GC.start
```

ルート

ヒープ領域



```
class Child; end
```

```
arry1 = []
```

```
arry1 << Child.new
```

```
arry2 = []
```

```
arry2 << Child.new
```

```
arry2 << Child.new
```

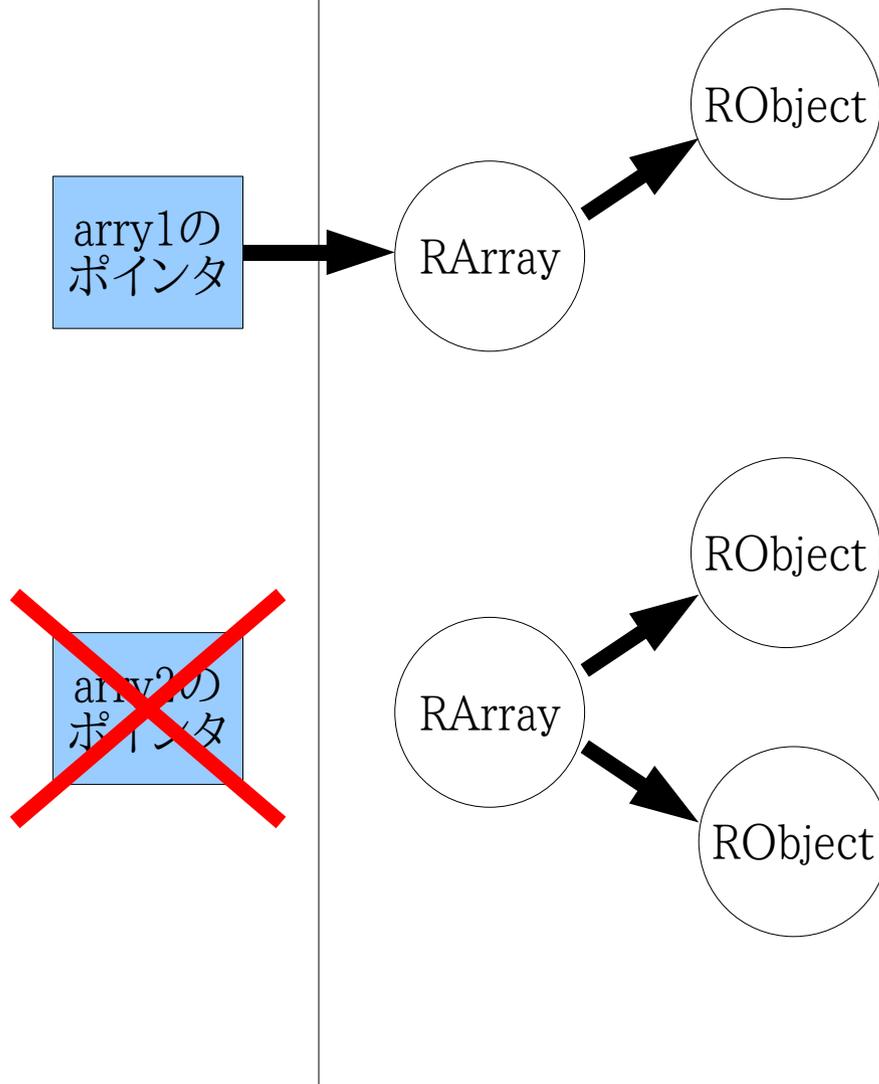
```
arry2 = nil
```

```
GC.start
```

RArrayの唯一の参照元削除

ルート

ヒープ領域



```
class Child; end
```

```
arry1 = []
```

```
arry1 << Child.new
```

```
arry2 = []
```

```
arry2 << Child.new
```

```
arry2 << Child.new
```

```
arry2 = nil
```

```
GC.start
```

ここからGCのはじまり

ルート

ヒープ領域

arry1の
ポインタ

RArray

RObject

~~arry2の
ポインタ~~

RArray

RObject

RObject

```
class Child; end
```

```
arry1 = []
```

```
arry1 << Child.new
```

```
arry2 = []
```

```
arry2 << Child.new
```

```
arry2 << Child.new
```

```
arry2 = nil
```

```
GC.start
```



Mark処理

ルート

ヒープ領域

ルートから必要な
オブジェクトをた
どる

arry1の
ポインタ

RArray

RObject

再帰的に
Markビットを
1にする

不要な
オブジェクトには
マークされない

RArray

RObject

RObject

```
class Child; end
```

```
arry1 = []
```

```
arry1 << Child.new
```

```
arry2 = []
```

```
arry2 << Child.new
```

```
arry2 << Child.new
```

```
arry2 = nil
```

```
GC.start
```

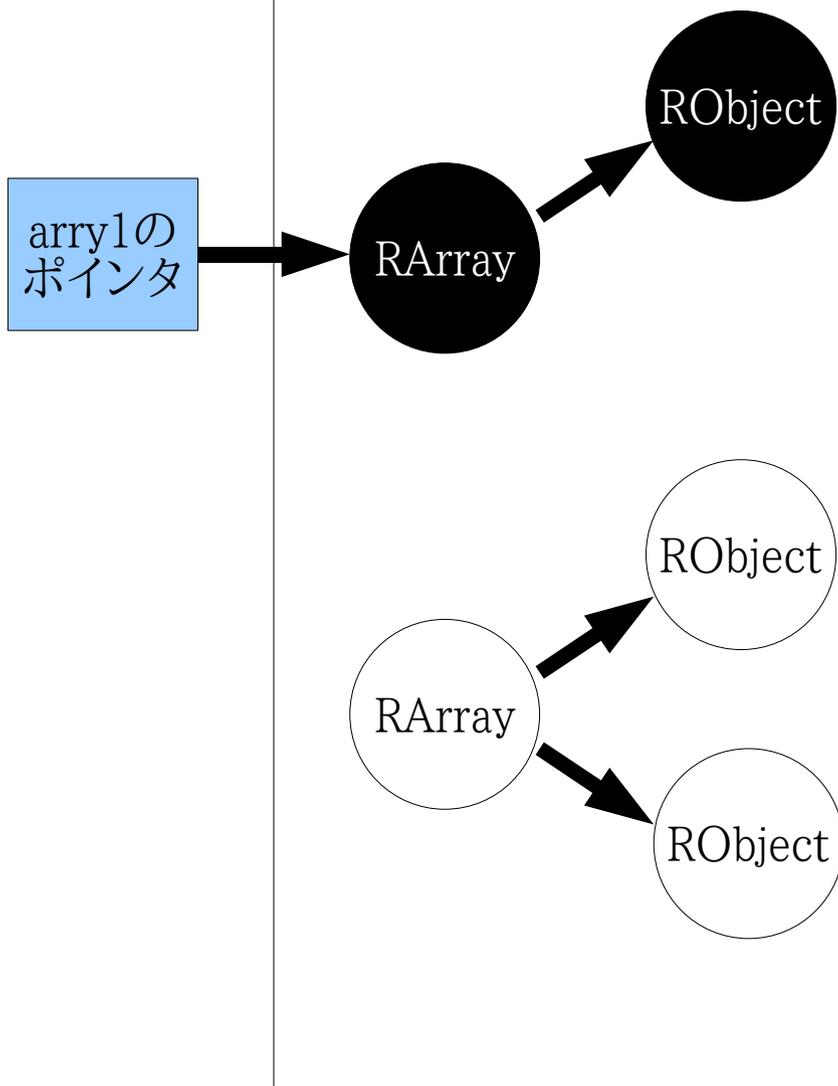


ヒープ領域を全
て走査する

Sweep処理

ルート

ヒープ領域



```
class Child; end
```

```
arry1 = []
```

```
arry1 << Child.new
```

```
arry2 = []
```

```
arry2 << Child.new
```

```
arry2 << Child.new
```

```
arry2 = nil
```

```
GC.start
```



ヒープ領域を全
て走査する

Sweep処理

ルート

ヒープ領域

arry1の
ポインタ

RArray

RObject

Markbitが立っている場合
Markbitを0にする

RObject

RArray

RObject

```
class Child; end
```

```
arry1 = []
```

```
arry1 << Child.new
```

```
arry2 = []
```

```
arry2 << Child.new
```

```
arry2 << Child.new
```

```
arry2 = nil
```

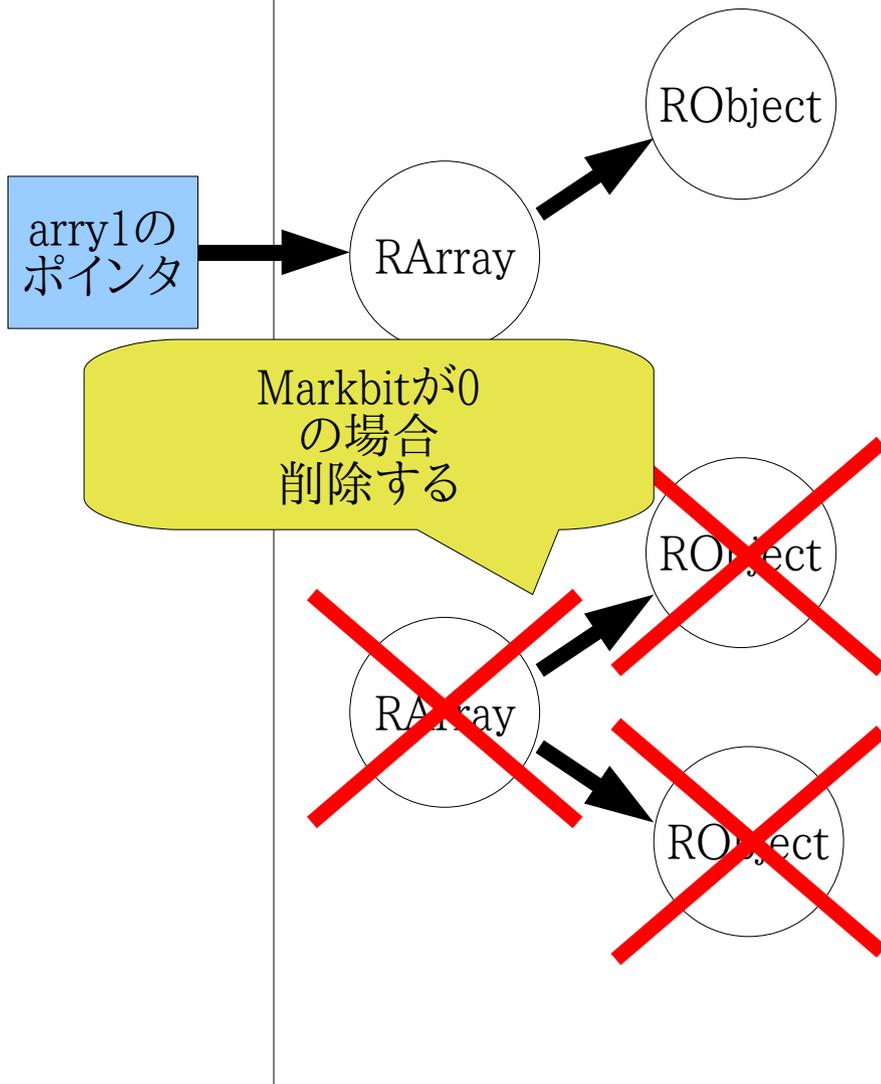
```
GC.start
```



Sweep処理

ルート

ヒープ領域



```
class Child; end
```

```
arry1 = []
```

```
arry1 << Child.new
```

```
arry2 = []
```

```
arry2 << Child.new
```

```
arry2 << Child.new
```

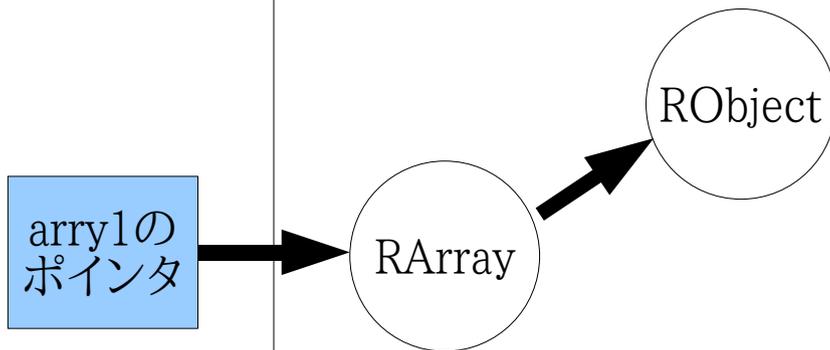
```
arry2 = nil
```

```
GC.start
```

GC終了

ルート

ヒープ領域



```
class Child; end
```

```
array1 = []
```

```
array1 << Child.new
```

```
array2 = []
```

```
array2 << Child.new
```

```
array2 << Child.new
```

```
array2 = nil
```

```
GC.start
```

これが

Mark - and - Sweep

です

そして
CRubyGCの大まか
な動作です

アジェンダ

- ・ CRubyのGC詳細
- ・ 欠点と改善
- ・ これからやりたいこと

欠点と改善

CrubyGCには
いろいろな欠点が
あります

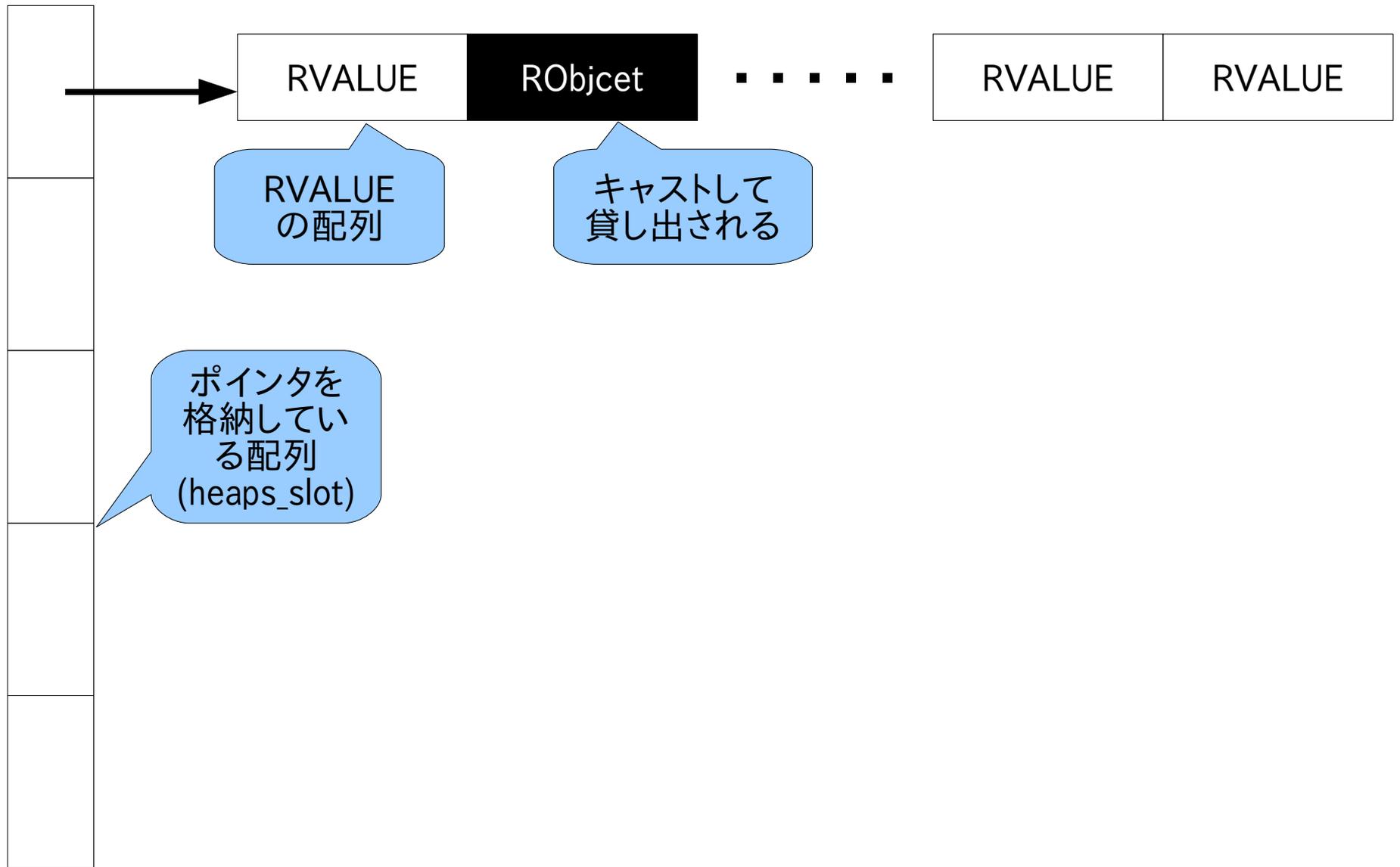
その中で2つだけ
ご紹介

— つめの
欠点

プロセス
メモリサイズの
肥大化

CrubyのHeap構造
を見ると分かる

これがCRubyのHeap構造だ！



Heapの増え方

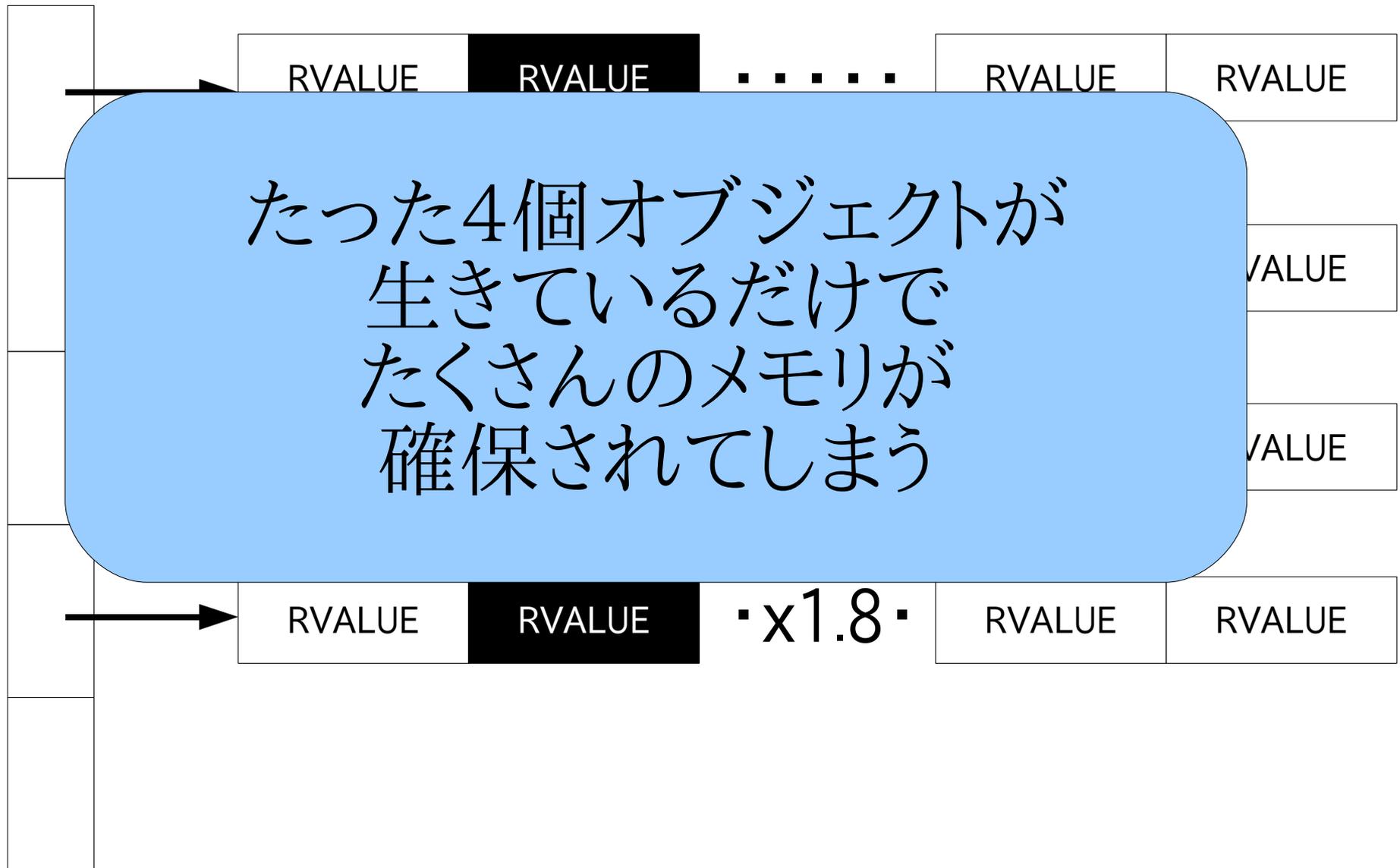
メモリ領域の解放

メモリ領域の解放



問題点

フラグメント化

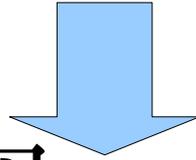


これは
どげんかせんと

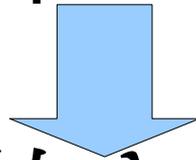
改善

改善方法

RVALUEの配列を小さくする
(1.8倍しない)



1000個 → 100個



配列ごと開放される可能性
が10倍高くなる

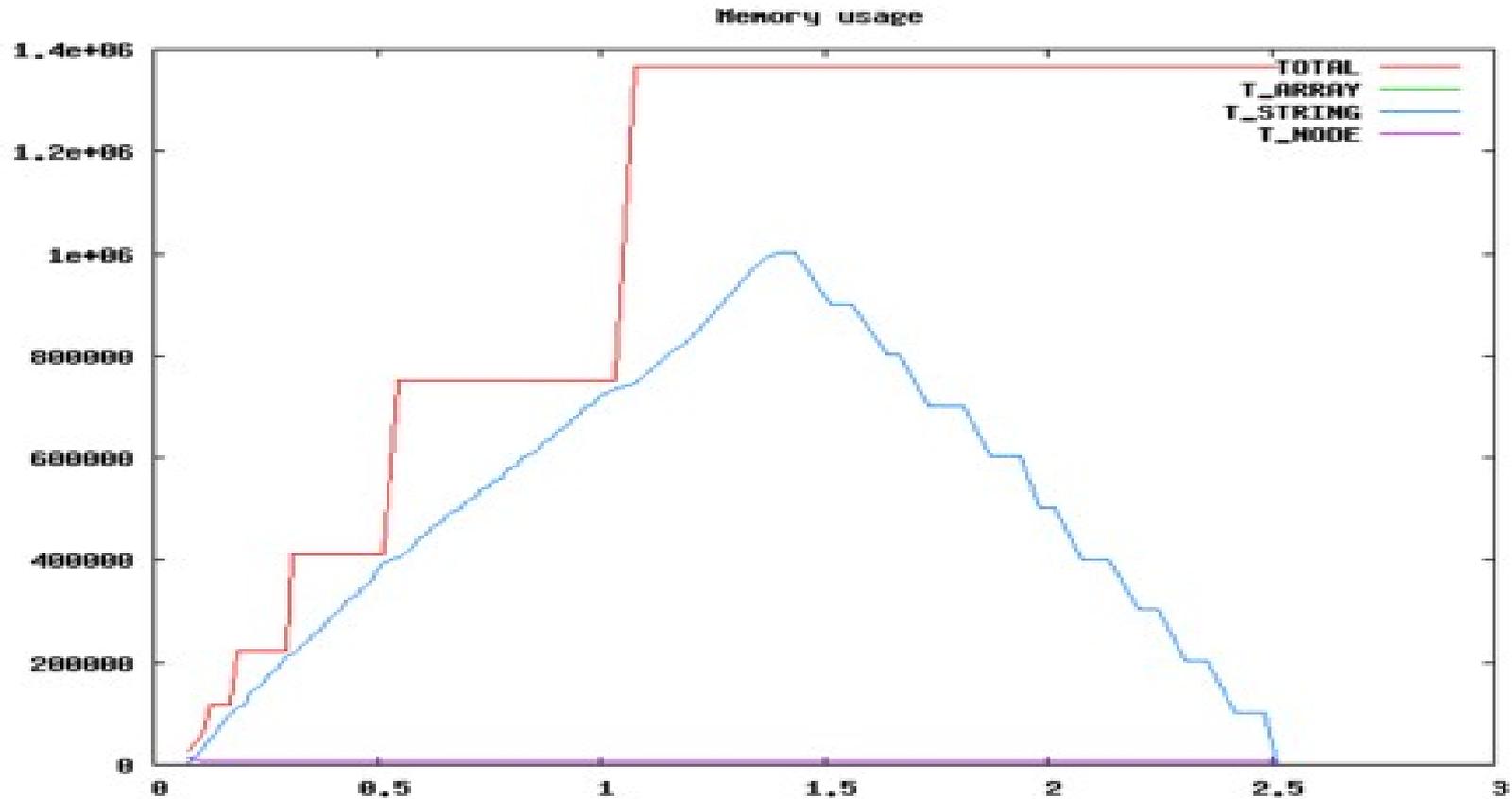
パッチ作りしました

[ruby-dev:34162] GC heap size less patch

thanks:作るときに非常に参考にしました
<http://lloydforge.org/projects/ruby/>

評価

StringをたくさんArrayに詰め込むコード
before

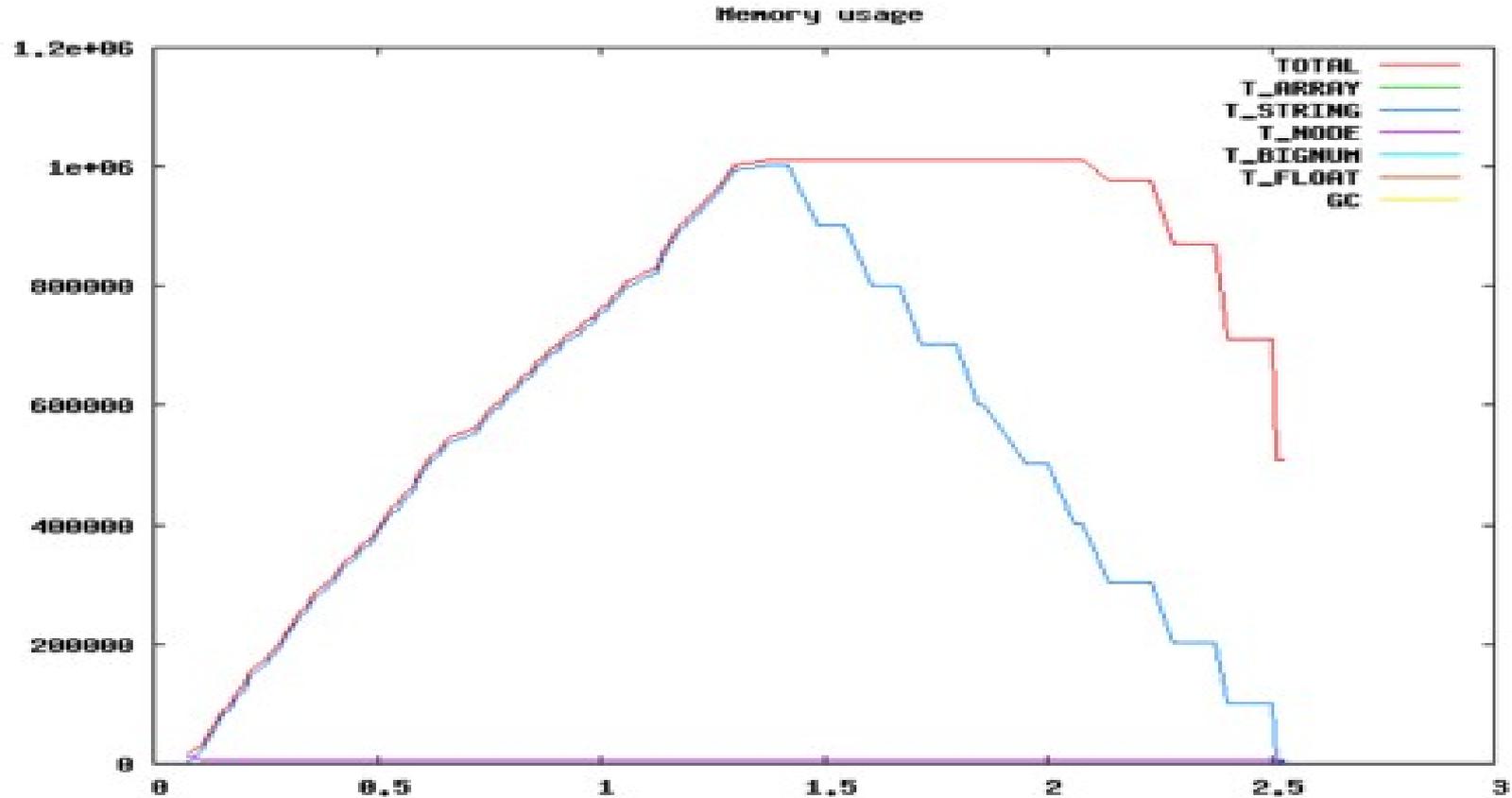


赤い線:トータルメモリサイズ
青い線:実際に使用しているオブジェクト

miura1729さんにてテストしていただきました
<http://d.hatena.ne.jp/miura1729/20080510/1210420290>

評価

StringをたくさんArrayに詰め込むコード
after



赤い線:トータルメモリサイズ
青い線:実際に使用しているオブジェクト

miura1729さんにてテストしていただきました
<http://d.hatena.ne.jp/miura1729/20080510/1210420290>

このパッチは まつもとさんの改良の後 1.9に無事取り込まれ ました

Fri Apr 25 17:54:10 2008 Yukihiro Matsumoto <matz@ruby-lang.org>

- * gc.c (HEAP_SIZE): use smaller heap segment (2K) for more chance to be freed. based on patch from authorNari <authornari at gmail.com>.
- * gc.c (rb_newobj_from_heap): eventually allocate heap segments.

どもです ><

そして
二つめの欠点

Stop - the - world

Rubyの
Mark - and - sweep
は全ての処理を
止めて行われる

問題点

Mark

生きているオブジェクトを再帰的にMarkしまくる

Sweep

Heap全体を舐め回す

中々、大変そうだ

つまり

mark、sweep
を行っている間は
グッと固まります

たとえば、
GUIとか触っていて固
まるとイラつきますよ
ね。

これが

Stop - the - world

たった0.2秒くらいでも
リアルタイム性が
重要なプログラムだと
致命的

例えば
ロボットとか

ロボットが歩いているときにGCが 走った良い例



ロボットが歩いているときにGCが 走った良い例



ど〜ん



その他では

例えば
ゲームとか？

Rubyで作った
ゲームとか？

実際に見てみましょう

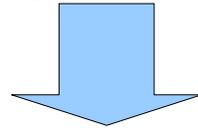
デモ

Stop - the - world

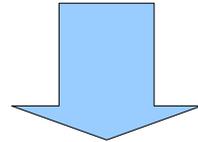
改善

改善方法

Markし終わった後、Sweepを
ちびちび行う



Sweep時は少ししか止まらない



4sec (Mark: 2sec、Sweep: 2sec)
止まる所が
2秒 (Mark: 2) ですむ

この手法を
LazySweepという

パッチ作りしました

[ruby-dev:34768] Improvement of lazy sweep

デモ

LazySweep摘要版

評価

たくさんオブジェクトを作るコード

	before	after	
GC最大停止時間	231.9ms	195.77ms	17%減
プログラム時間	55.39s	59.8s	8%増

これはまだ
取り込まれていません

もっと
改良しないと。。。。

アジェンダ

- ・ 欠点と改善
- ・ これからやりたいこと

これから
やりたいこと

これからやりたいこと

- ・ BoehmGCを組み込んでみる
もしかしたら早くなるかも？
- ・ thread localなfreelist
キャッシュが効いて早くなるかも？
- ・ incremental gc
writebarrierが難しいけど><

詳しい説明は
割愛します

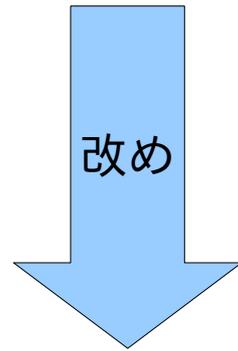
m(_ _)m

この通り
まだCRubyGCは
色々遊べます

CRubyで
もっとも未開拓な
部分

みなさんの
アイデアをGCにぶつ
けてみませんか？

どげんかせんといかん



一緒にどげんか
しましょう！

興味のある方は
この後にも

最後に

簡単まとめ

まとめ

- ・GCって素晴らしい！
- ・Ruby1.9で少しでも改善されそう(GC)
- ・でも、まだまだ改善の余地がある
- ・誰かやりませんか？

謝辭

この場を

提供していただいた

スタッフの皆様へ感謝

そして

なによりも

こんなにマニアックな

セッションを最後まで

聞いていただいた

皆様に感謝

ご清聴
ありがとうございます
ございました

質疑応答

の前に

想定される質問を
回答しました

Q1

ReferenceCount
(参照カウント)

使えばいいじゃないっ
すか？

A1

循環参照の問題

Cの拡張ライブラリで
意識しなくてはならなく
なるので
嫌です。

A1

それに
カウント処理とか
参照時に
毎回動いて結構遅い
ので

Q2

世代別GC使えば
いいじゃないですか？

A2

古い世代からの
参照が問題
ゴミでないものが
開放される恐れがある

A2

その為には
WriteBarrierという
機構をいれるが
C拡張には入れたくな
いよね

Q3

NaClって
どんな感じなの？

A3

非常にみなさん
仲がいい会社です。
交通費も出して
頂きました。

m(_ _)m

A3

昼ご飯を
みんなわきあいあいと
一緒に
食べます。

A3

ちなみに
僕の前の席は
前田さん

A3

前の前の席が
まつもとさん

Q4

Ruby認定試験
受かりました？

A4

Ruby技術者認定試験 Association Certified Ruby Program 試験結果レポート	
HIRO NAKAMURA	
100	受験者ID
7	受験日
Association Certified Ruby Programmer Silver	
点	合格点
結果： 不合格	

A4

僕にはこれだけが残った



と言う事で

質疑応答