プログラム言語 **Ruby** における メソッドキャッシング手法の検討

東京農工大学大学院博士後期課程
（日本 Ruby の会）
笹田　耕一

2005 3/3 情報処理学会全国大会

背景:オブジェクト指向プログラム言語Ruby

- インタプリタ
- 使いやすいと評判
  - 言語処理系開発にも！
- 世界中で広く利用
  - Ruby-talk ML では一日に百通以上
  - 最近 Ruby on Rails が人気
- 日本発（主開発者：まつもとゆきひろ（Matz）氏）
  - 日本発で世界に通用する数少ないソフトウェア
- コミュニティ活動も活発
  - 日本Rubyの会，Rubyist Magazine

背景:
Rubyist Magazine 0005 (Feb. 2005)

http://jp.rubyist.net/magazine/

背景: Ruby の動作速度

- 構文木をそのままたどるインタプリタ
  - 遅い

→　命令セットを定義し，それを実行する VM が必要

背景: YARV-Yet Another RubyVM

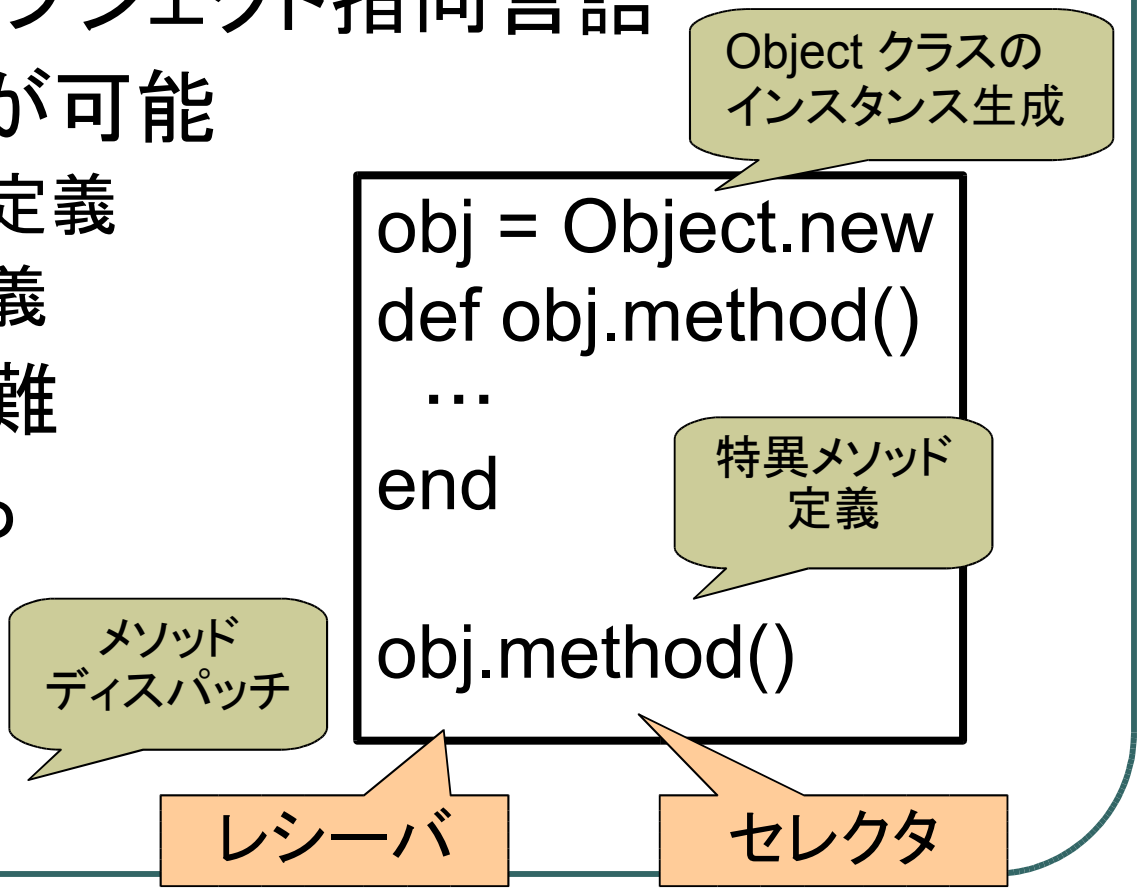
- 高速化のために仮想マシンを再設計
  - VM命令セットの設計
  - コンパイラの設計・開発
  - 仮想機械(RubyVM)の設計・開発
  - JIT (Just In Time), AOT (Ahead of Time) コンパイラ

YARV: Yet Another Ruby VM
として開発中(オープンソースソフトウェア)
(**IPA 2004**年度未踏ユース採択)

→ 本発表:**YARV** のメソッドディスパッチの高速化

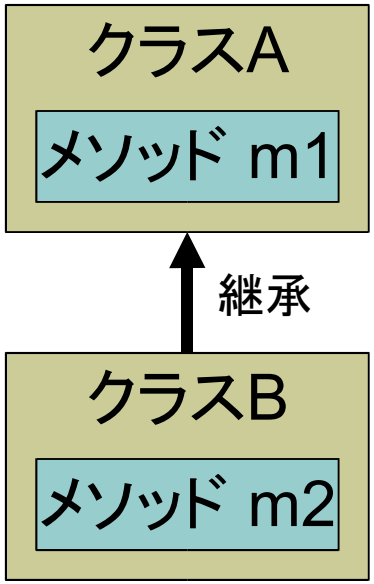
Ruby の特徴

- クラスベースオブジェクト指向言語
- 動的な再定義が可能
  - 特異メソッドの定義
  - メソッドの再定義
- 静的解析が困難
- その他いろいろ



メソッドディスパッチ

- ナイーブなメソッドディスパッチ
  - 基底クラスへメソッドを検索
    → 遅い
- さまざまなメソッドディスパッチ手法
  - キャッシュ法（簡単）
  - ディスパッチ表法（再定義に弱い）
  - 直接束縛法（大変）



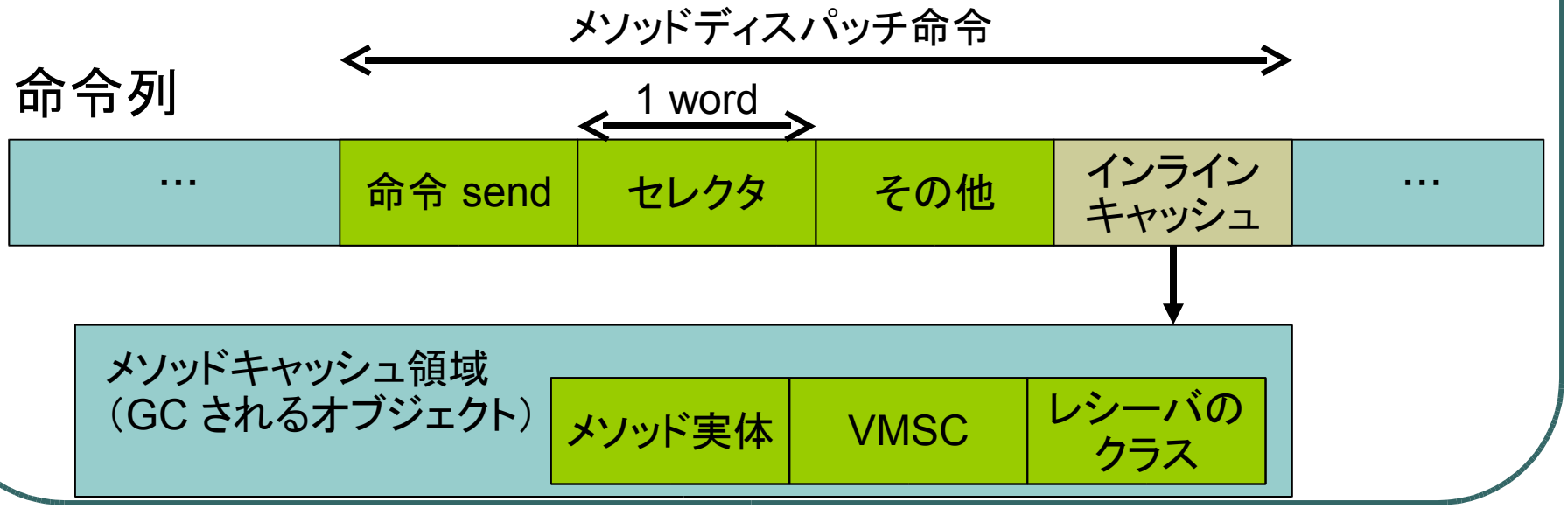
m1 の検索には
クラス A まで
検索が必要

現在のRubyにおけるメソッドディスパッチ

- グローバルメソッドキャッシュ
  - VM 全体でひとつのハッシュを用意
  - 鍵は [レシーバクラス, セレクタ]
  - 値はメソッドの実体（と、クラス：検証用）
- 再定義時、そのセレクタのエントリをクリア
- 95% のヒット率
- いくつかの手順が必要（オーバヘッド）

インラインメソッドキャッシュ

- 命令オペランドにメソッドの実体をキャッシュ
- 「だいたい前回と同じレシーバクラスだろう」
- YARV ではグローバルメソッドキャッシュと併用



VM状態カウンタ (VMSC)

- 再定義処理のため**VM**状態カウンタを導入
  - VM にひとつのカウンタ
  - 定義操作（メソッド定義など）でカウンタ増加
- キャッシュ時、内容と一緒にVMSC も格納
  - キャッシュヒットの判定をレシーバクラス，**VMSC**で
- VMSC を変化 → 全キャッシュをクリア
  - 「再定義操作なんて滅多に起こらないだろう」

本手法の利点と欠点

- ○キャッシュのクリアが容易
- ○実装が容易
- ×すべてのキャッシュをクリア
  - 特定のセレクタのみをクリアするべき？
  - クリアのコスト（実装，処理時間）とのトレードオフ

評価: マイクロベンチマーク

```
# 常にキャッシュヒット
i=0
obj = Object.new
def obj.m(); end
while i<MAX
  i+=1
  obj.m(); obj.m(); …
end
```

```
# 常にキャッシュミス
i=0
obj1 = Object.new
obj2 = Object.new
def obj1.m(); end
def obj2.m(); end
while i<MAX
  i+=1
  obj = (i%2) == 0 ? obj1 : obj2
  obj.m(); obj.m(); …
end
```

評価: マイクロベンチマークの結果

- インラインメソッドキャッシュの効果あり
- VMSC チェックのコスト(3%)を気にする？

→その他のクリア手法

- キャッシュミスのコスト(3%)を気にする？

→ キャッシュミスが閾値以上起こった場合、メソッドキャッシュしない

	レシーバのみチェック（不完全）	VMSCもチェック	インラインキャッシュなし
多態なし (必ずヒット)	2.79 (右と比べ3%高速)	2.88 (右と比べ8%高速)	3.11
多態あり	3.45 (?)	3.40 (右と比べ3%低速)	3.28

単位: 秒

評価: ヒット率の評価

- いくつかのプログラムでヒット率を計算
  - Cal : カレンダー
  - RSS: RSS リーダ
  - ReXMLbib: XML で記述された文献DB を HTML化
  - SOAP: SOAP を利用した通信プログラム
  - Webrick: Ruby による HTTP サーバ
- YARV では上記は完全には動作しないため、現在のインタプリタでヒット率を計算
  - メソッドディスパッチの履歴を保存して計算

評価: ヒット率の評価

- （初期ミスは除外）
- 大抵の場合ヒット
- ミスする場所は集中

→ キャッシュミスが閾値以上起こった場合、メソッドキャッシュしない

- ミスするセレクタはある程度予測可能
  - naming convention

	キャッシュヒット率(%)	ミスが発生したディスパッチ率(%)
cal	98.7	0.9
RSS	97.2	9.8
ReXMLbib	76.6	8.3
SOAP	95.5	26.4
Webrick	97.8	6.2

まとめ

- Ruby にインラインメソッドキャッシュを搭載
  - 最大 8% の高速化
- VM状態カウンタを利用して容易に再定義に対応
- Rubyプログラムのメソッドキャッシュヒット率を調査
  - 多くの場合 95% 以上のヒット率

今後の課題

- 一般的なプログラムの再定義操作回数の調査
- JIT / AOT コンパイラ向けメソッドキャッシュ手法の検討
- スレッドセーフに
- その他のクリア手法の検討
  （本方式で本当にいいのか検討）

終わり

- ご清聴ありがとうございました。

YARV: Yet Another RubyVM

http://www.atdot.net/yarv/

ko1@atdot.net

インラインメソッドキャッシュコード

```
if(klass == ic->ic_klass &&
 GET_VM_STATE_VERSION() == ic->ic_vmstat){
   mn = ic->ic_method;
 }
 else{
   mn = rb_method_node(klass, id);
   ic->ic_klass  = klass;
   ic->ic_method = mn;
   ic->ic_vmstat = GET_VM_STATE_VERSION();
 }
```

GCC の分岐予測を考慮したコーディング

```
#define LIKELY(x)   (__builtin_expect(x, 1))
#define UNLIKELY(x) (__builtin_expect(x, 0))
if(LIKELY(レシーバクラス一緒？) &&
   LIKELY(VMSC 一緒？)){
  // ヒット時の処理
} else {
  // ミス時の処理
}
```