東京大学
情報理工学系研究科
創造情報学専攻

Ricsin:
RubyにCを埋め込むシステム

Ricsin: A System for “C Mix-in to Ruby”

東京大学大学院情報理工学系研究科創造情報学専攻
笹田 耕一 / ささだ こういち
sasada@ci.i.u-tokyo.ac.jp

Agenda

- 背景：現状のRubyとC拡張ライブラリ
- 提案：Ricsin: RubyにCを埋め込むシステム
  - **従来に比べ，書きやすく，低オーバヘッド**
  - Ricsin の利用例と処理の流れ
  - Ricsin の記法
  - Ricsin の設計と実装
- 評価と考察
- 関連研究
- まとめと今後の課題

Ricsin記法での記述例

```
def open_fd(path) # Ruby での記述
  fd = _C_(%q{
    /* C での記述 */
    return INT2FIX(open(RSTRING_PTR(path), O_RDONLY));
  })
  raise 'open error' if fd == -1
  yield fd
ensure
  raise 'close error' if -1 == _C_(%q{
    /* C での記述 */
    return INT2FIX(close(FIX2INT(fd)));
  })
end
```

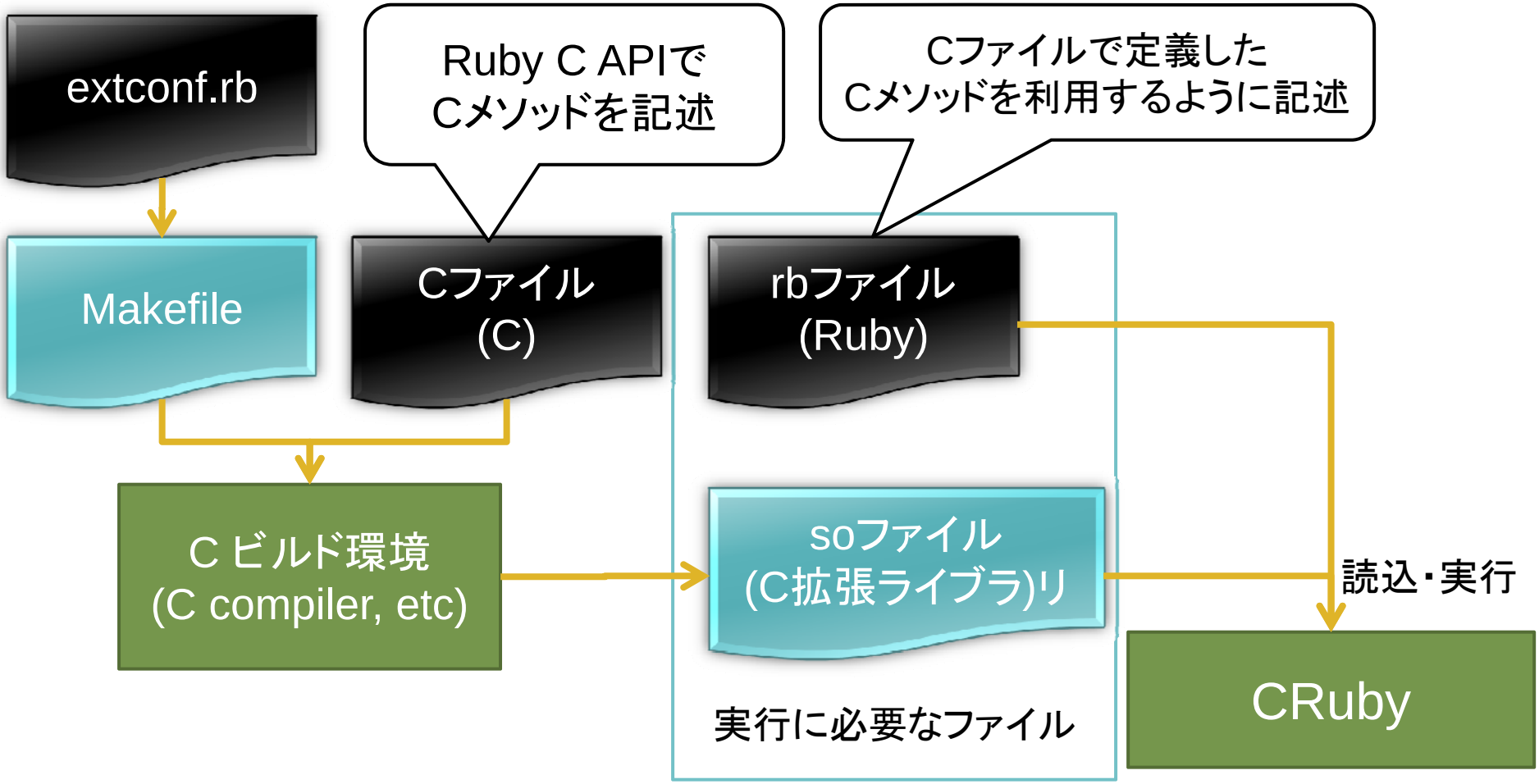
背景：Ruby

- オブジェクト指向スクリプト言語Ruby
- 最新版 Ruby処理系 Ruby 1.9.1
  - もっとも利用されているRuby処理系
  - 200x年y月リリース予定
  - 1.9 からは仮想マシン（VM）を搭載（YARV）
  - C言語で実装されているので CRuby と表記
    - Eg. JRuby, IronRuby, Rubinius
- Cで開発されているので，Cで処理を拡張可能
  - 組み込みクラスやメソッド
  - C拡張ライブラリ（動的リンクライブラリ）

背景：RubyからCの機能を利用

- Cでないと記述できない処理
  - システム依存の処理，レガシーなライブラリの利用
  - Ruby処理系に踏み込まないと記述できない処理
  - Rubyでは性能でない処理
- Cで記述したメソッドを集めたC拡張ライブラリ
  - Cでメソッドを記述 → Cメソッド（vs. Rubyメソッド）
  - つまり，Ruby ⇔ C の遷移は「メソッド」単位
  - CメソッドからRubyメソッドを呼ぶことも可能
  - Ruby C APIを利用して記述
  - C だけで考えるとそれなりに書きやすい

背景：C拡張ライブラリの作成と利用



現状のC拡張ライブラリ作成の問題点

1. **記述性の問題**：RubyとCをメソッド単位で分割
   - CとRubyでファイル分割が必要
   - メソッドより細かい粒度での処理が記述できない
   - 処理が必要な箇所に直接記述できない
     - 例外処理やブロック呼び出しなど，Rubyで書いた方が圧倒的に簡単な処理もCで書かなければならない
2. **性能の問題**：必ずメソッド呼び出しが必要
   - 必ずフレーム生成のオーバヘッドが必要
   - Ruby → C の呼び出し
   - C → Ruby の呼び出し（VM再帰．こっちの方が重い）
     - イテレータを C で実装するとこうなることが多い

Ricsin: RubyにCを埋め込むシステム

- **提案：RubyにCを埋め込むシステムRicsin**

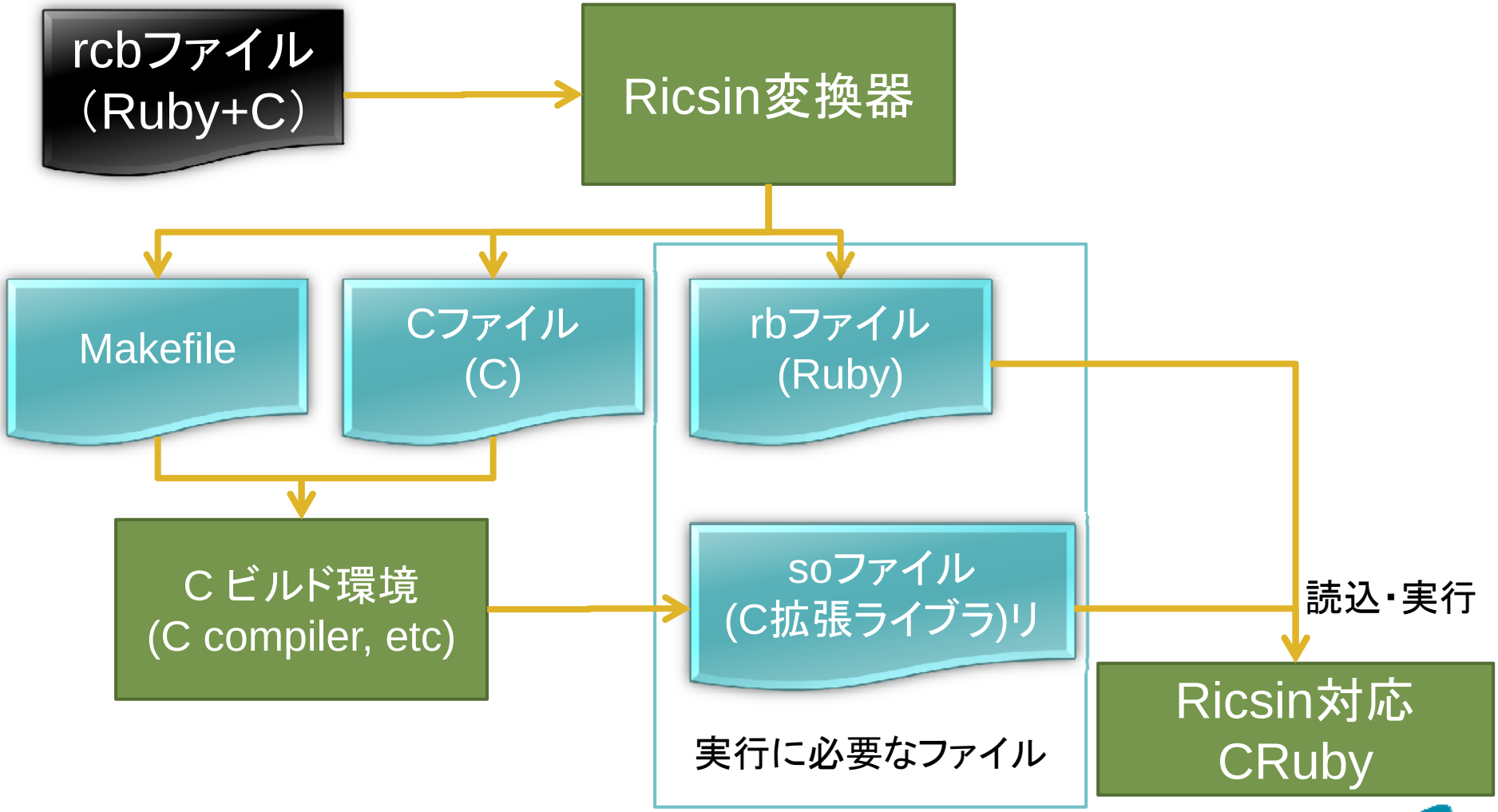
- **記述性を向上**：Rubyプログラム中にCを記述
  - RubyにCプログラム片を直接埋め込む書式
  - Rubyのコンテキスト情報をCプログラム片から直接参照

- **性能を改善**：VM命令を新設し呼び出しコスト削減
  - Ruby⇔C の遷移をVM命令/関数呼び出しコストで
  - メソッドの一般化が不要で特殊な決めうちが可能

Ricsinの利用例

- Cでないと記述できない処理を，Ricsinでは：
  - システム依存の処理，レガシーなライブラリの利用
    → **必要な箇所に必要な処理を直接Cで埋め込み**
  - Ruby処理系に踏み込まないと記述できない処理
    → **RubyにRuby C APIを直接埋め込み**
    例：Ruby処理系，Ruby C APIのテスト
  - Rubyでは性能でない処理
    → **Rubyに直接Cを埋め込み，継続的高速化**
    記述性の高さ，効率の良さ相互に影響

Ricsinの全体像


rcbファイル
（Ruby+C）
Ricsin変換器
Makefile
Cファイル
(C)
rbファイル
(Ruby)
C ビルド環境
(C compiler, etc)
soファイル
(C拡張ライブラ)リ
読込・実行
Ricsin対応
CRuby
実行に必要なファイル

Ricsin記法

- Ricsin記法：Ruby + C混在プログラム記述記法
  - ファイルの拡張子はrcb（rb に c を埋め込んだ）
- **完全に有効なRubyプログラム**として設計
  - Rubyパーサ，検証器をそのまま利用可能
  - __C__ などの**特定のメソッド呼び出し**を**Cプログラム片埋め込み指示子**として利用
  - **文字列リテラル**でCプログラムを指定
- 埋め込んだCプログラム片は若干変換
  - 埋め込んだCプログラム中からは**Rubyの環境**がそのまま参照可能
  - Ruby C APIを利用したCプログラム

Ricsin記法での記述例

```
def open_fd(path) # Ruby での記述
  fd = _C_(%q{
    /* C での記述 */
    return INT2FIX(open(RSTRING_PTR(path), O_RDONLY));
  })
  raise 'open error' if fd == -1
  yield fd
ensure
  raise 'close error' if -1 == _C_(%q{
    /* C での記述 */
    return INT2FIX(close(FIX2INT(fd)));
  })
end
```

Ricsinで利用するレシーバ

- __C__
  - Cプログラム片をその場に埋め込み
- __Cdecl__
  - 関数，マクロ，グローバル変数定義を記述
- __Cinit__
  - ロード時に一度だけ実行する処理を記述
- __Cb__
  - RubyのブロックをCで記述
- __Ccont__
  - Cの中にRubyを記述

__Ccont__

- Cの中にRubyを埋め込み

→ CとRubyを並べるための指示子 __Ccont__

- 便利な省略記法を用意
- ただし，Cの変数
  スコープは切れる

→ Rubyのローカル変数
  で値を共有

```
v = true
__Ccont__('while (v != Qnil) {') # (a)
  v = nil                        # (b)
__Ccont__('  rb_p(v);')          # (c)
__Ccont__('}')                   # (d)
```

```
v = true
#C while (v == Qnil) { /* (a) */
      v = nil              # (b)
#C    rb_p(v);             /* (c) */
#C  }                      /* (d) */
```

コンテキスト情報へのアクセス

- CからRubyのコンテキスト（変数）情報へ直接アクセス可能
  - ローカル変数 (lv)
  - インスタンス変数 (@iv)，グローバル変数 ($gv) 等
  - 定数
  - self 擬似変数

コンテキスト情報へのアクセス例

```
def open_fd(path) # Ruby での記述
  fd = _C_(%q{
    /* C での記述 */
    return INT2FIX(open(RSTRING_PTR(path), O_RDONLY));
  })
  raise 'open error' if fd == -1
  yield fd
ensure
  raise 'close error' if -1 == _C_(%q{
    /* C での記述 */
    return INT2FIX(close(FIX2INT(fd)));
  })
end
```

変換と実行

- rcb -> C+Rubyに変換
  - rcbファイルをパースして，埋め込みCを抽出
  - 埋め込み式をCの関数（*1）に変換
  - 変換プロセスの詳細は省略
- 実行時，埋め込み式の実行
  - 専用VM命令 **opt_ricsin_callを新設**（VM唯一の修正点）
  - (*1) の関数ポインタを渡すようにバイトコードコンパイル
  - Cメソッド呼び出しから**VM命令実行＋関数呼び出し**に
  - この命令は，VM命令を拡張する命令ということも可能

変換と実行 (cont.)

```
# rcb ファイル
v = 42
r = __C__(%q{
  /* 埋め込み C ボディ部 */
  rb_p(self); /* main と表示 */
  return INT2FIX(
    FIX2INT(v) + 1);
})
p r #=> 43 と表示
```

生成

```
/* 生成されるCソースファイルの一部 */
#define v (cfp->lfp[3])
#define r (cfp->lfp[2])
VALUE ricsin_func_1(
  rb_control_frame_t *cfp)
{
  const VALUE self = cfp->self;
  {
    /* 埋め込み C ボディ部 */
    rb_p(self);
    return INT2FIX(FIX2INT(v) + 1);
  }
  return Qnil;
}
#undef v
#undef r
```

拡張ライブラリに

バイトコードコンパイル

関数呼び出し

```
[ADDR] [INSN]          [OPERAND]
0000 putobject         42
0002 setlocal          v
0004 opt_call_ricsin   <funcptr>
0006 setlocal          r
0008 putnil
0009 getlocal          r
0011 send              :p, 1
0017 leave
```

__Ccont__ の変換

```
v = true
#C while (v == Qnil) { /* (a) */
     v = nil            # (b)
#    rb_p(v);           /* (c) */
#C }                    /* (d) */
```

生成

```
#define v (cfp->lfp[2])
VALUE ricsin_func_1(
   rb_control_frame_t *cfp)
{
  switch (GET_PC()) {
    case 10: goto label_10;
    case 20: goto label_20;
  }
  {
  label_10:;
    while (v != Qnil) {;  /* (a) */
      SET_PC(10); return Qnil;
    label_20:;
      rb_p(v);            /* (c) */
    };                    /* (d) */
    SET_PC(25); return Qnil;
  }
}
  return Qnil;
}
```

バイトコードコンパイル

```
[ADDR] [INSN]            [OPERAND]
 0002  putobject         true
 0004  setlocal          v
 0008  opt_call_ricsin   <funcptr>
 0010  pop
 0013  putnil
 0014  setlocal          v
 0018  opt_call_ricsin   <funcptr>
 0020  leave
```

生成されたC言語ファイルのレイアウト

- rcb 1 ファイルにつき
  1つのCファイルを生成
  (詳細は割愛)

評価

- 主に性能に関する評価
- 評価環境
  - 評価環境1：Intel Xeon E5335, Linux
  - 評価環境2：SPARC T2, SunOS 5.10
    - 傾向は変わらず．どちらでも実行できることを示すため.
- 評価項目
  1. Cの機能呼び出しの実行時間比較
  2. イテレータの高速化の例
  3. 行列計算の高速化

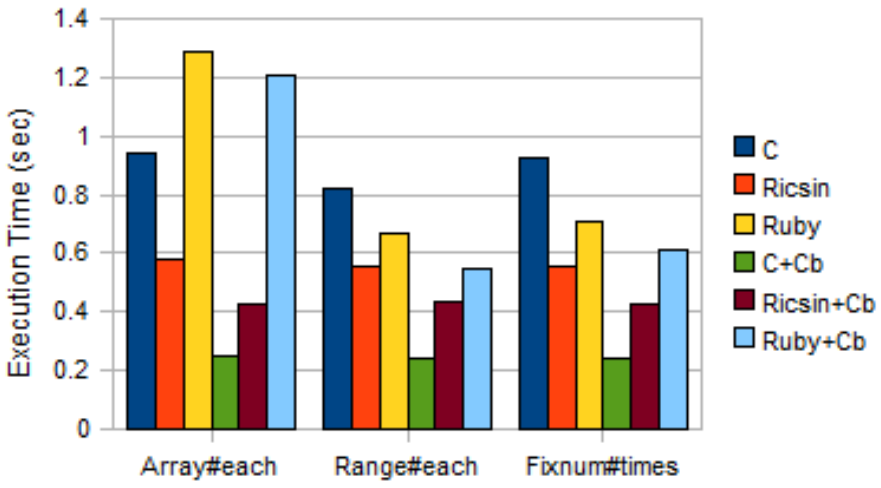
評価：Cの機能呼び出しの実行時間比較

- 空のC機能の呼び出し時間の比較
  - 空のCメソッド
  - 空の__C__

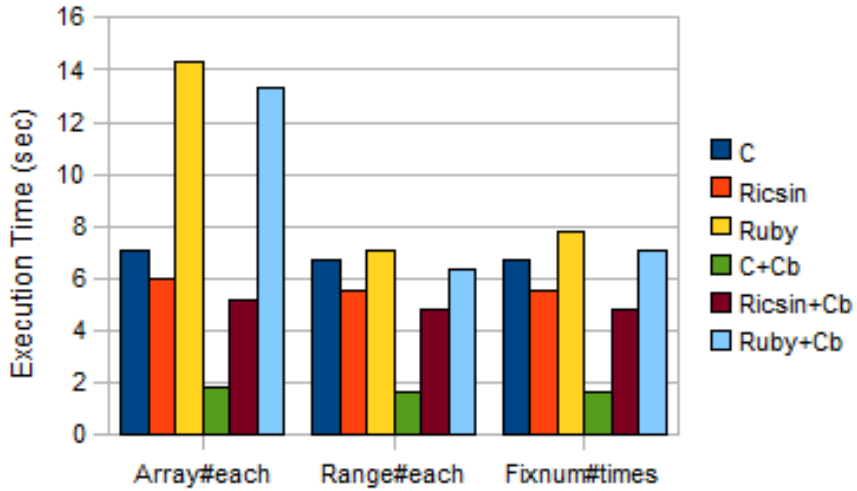
	C (sec)	Ricsin (sec)	C/Ricsin
評価環境1（Intel）	0.44	0.05	8.8
評価環境2（SPARC）	4.56	0.44	10.4

評価：イテレータの高速化の例

- イテレータをRicsinで書き直し
  - C: 従来
  - Ricsin: __Ccont__ を利用して Ruby/C をミックス
  - Ruby: Rubyで書き直し



評価環境1（Intel）



評価環境2（SPARC）

評価：行列計算の高速化

- 行列の乗算（要素は整数）
- 12行のRuby Scriptを36行のCプログラム片で**直接置き換え**

	Ruby (sec)	Ricsin (sec)	Ruby/Ricsin
評価環境1（Intel）	10.57	0.57	20.33
評価環境2（SPARC）	85.31	6.73	12.68

Ricsinに関する考察

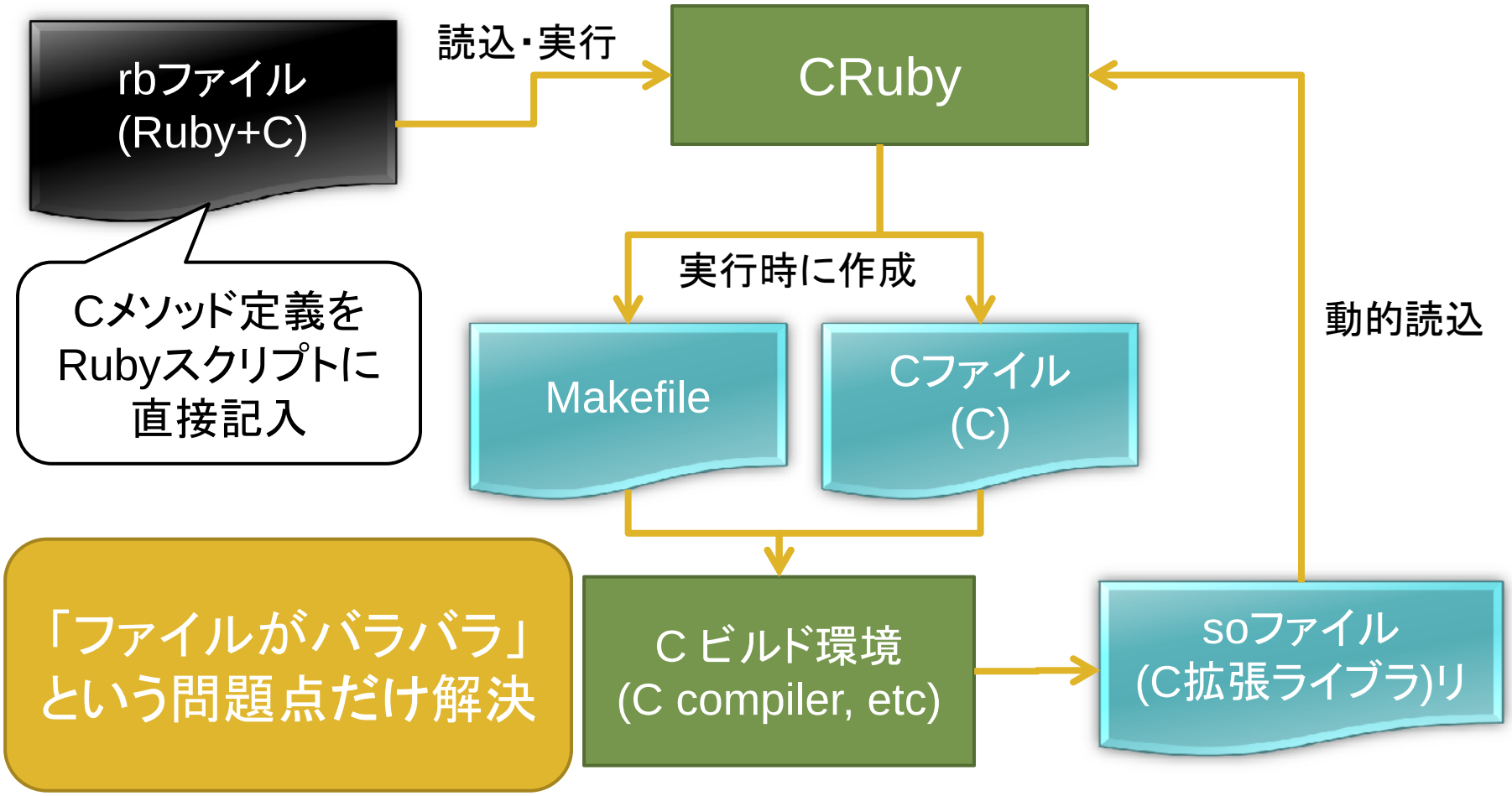
- Cを埋め込むと高度な最適化が出来ないのでは？
  → 最適化が進めばそうなるが，その段階は遠い
  Cを対象にしているためポータビリティも高い
  処理系メンテナンスコストも低い
- 本当に書きやすいの？
  → 数行であれば従来のCメソッドより書きやすい
  エディタ等の対応が必要になるかも

関連研究

- C（とくに gcc）の asm 文（C + assembler）
  - asm 中に C は記述不可能
- HTML埋め込み言語（PHP，JSP，JavaScript）
- Jeannie (OOPSLA 2007)
  - Java+Cコードを混在
  - JavaとJNIコードを生成
  - Java→C，Java→Cは
    メソッド呼び出しに変換

```
public static native void f(int x)
`{                                                    1
  jint y = 0;
  `{                                                  2
    int z;
    z = 1 + `(y = 1 + `(x = 1)4)3 ;
    System.out.println(x);
    System.out.println(z);
  }
  printf("%d\n", y);
}
```

関連研究：RubyInline（CメソッドをRubyスクリプトに記述）



まとめ

- RubyにCを埋め込むRicsinシステムを提案
  - メソッド単位ではなく，Rubyに直接Cを埋め込む記法
  - コンテキスト情報に直接アクセス可能で記述性向上
  - VMに命令を追加することで10倍高速なC機能呼び出し
- 今後の課題
  - C部分を真面目にパースして連携（の検討）
  - AOTコンパイラとの連携

おわり

ご清聴ありがとうございました

Ricsin: RubyにCを埋め込むシステム
http://svn.ruby-lang.org/repos/ruby/branches/ricsin/

ささだこういち
sasada@ci.i.u-tokyo.ac.jp
ko1@atdot.net, ko1@rvm.jp

謝辞
執筆に協力して下さった方々
科研費若手（スタートアップ）19800007