

mrubyの現状と事例紹介

九州工業大学 情報工学部
田中 和明

RubyAssociation
軽量Rubyフォーラム

自己紹介

- 田中 和明 (たなか かずあき)
- 九州工業大学 情報工学部 機械情報工学科
- 出身は情報工学で、力学は不得意
機械らしいことを研究テーマに. . .
- 組込みシステム (ハード+ソフト) の研究

Rubyとmruby

- Ruby
 - オブジェクト志向プログラム言語
ISO30170 / JIS X 3017
 - Webアプリケーション開発では標準的な言語
- mruby
 - Rubyの仕様に従って，軽量化した言語
 - **実行時**に必要なリソースを少なくする

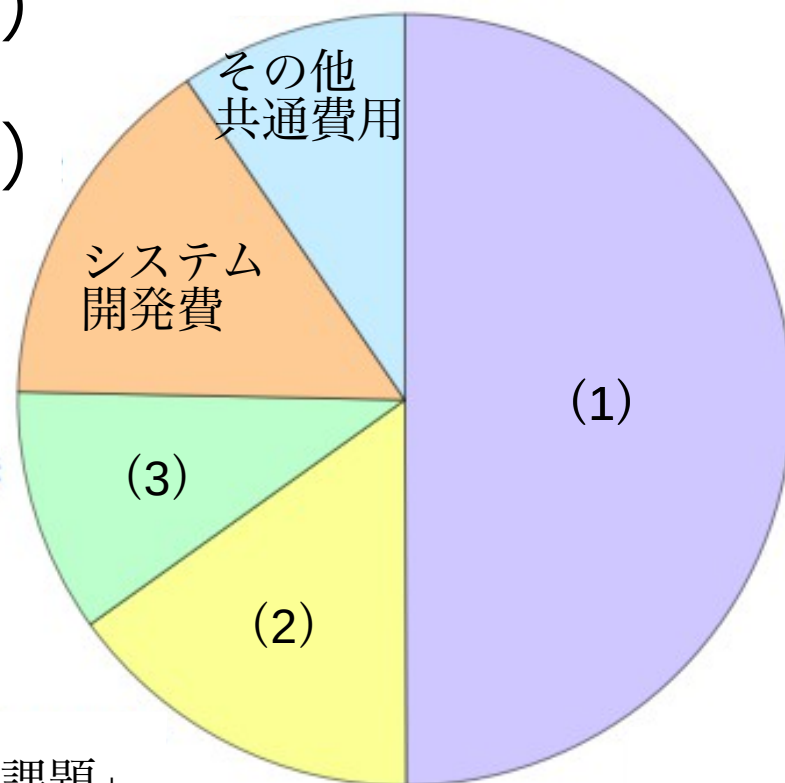
組込みシステム開発の課題

- 組込みシステムの開発費内訳
 - (1) (2) (3) に入るのは？

A : ハードウェア (電子系)

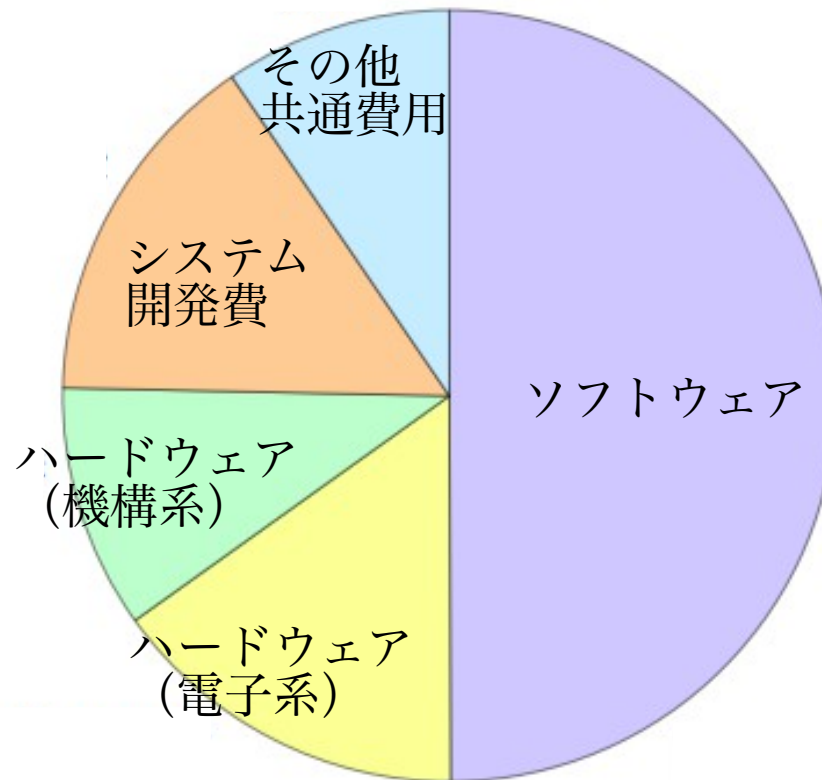
B : ハードウェア (機構系)

C : ソフトウェア



組込みシステム開発の課題

- 組込みシステムの開発費内訳



IPA 「組込みソフトウェア産業の現状と課題」
より抜粋

開発費内訳について

- 開発費は
ソフト > ハード（電子） > ハード（機構）
- ハード（機構）の開発は、
相当のノウハウが蓄積されてきている
- 物理的な制約から、自由度は少ない
- （組込みシステムにおいて）
ソフトウェアの開発は、あまり進歩していない
- 自由度が大きい（→ 簡単に複雑化してしまう）

Rubyが使われる理由

- ソフトウェア開発がしやすい
 - 開発者のためのプログラム言語である
(まつもとゆきひろ氏)
- プログラムの可読性が高い
 - 自分のプログラム, 他人のプログラムを理解しやすい
- 多くのライブラリが用意されている

Rubyのプログラム

- 次のプログラムの振る舞いを推測してみてください

```
Raspi.digitalWrite 5,  Raspi::HIGH  
Raspi.digitalWrite 13, Raspi::LOW  
Raspi.digitalWrite 26, Raspi::HIGH
```

- (補足説明)
 - digitalWrite : 信号 (HIGH/LOW) を出力する
 - 信号5 : 青色
 - 信号13 : 黄色
 - 信号26 : 赤色

Rubyプログラムの特徴

- プログラムを読みやすい



- プログラムを書きやすい
保守しやすい
不具合を減らせる



- ソフトウェアのコストを減らせる
開発期間を短縮できる
- ソフトウェア開発を試行錯誤できる

Rubyで 組み込みソフトを開発

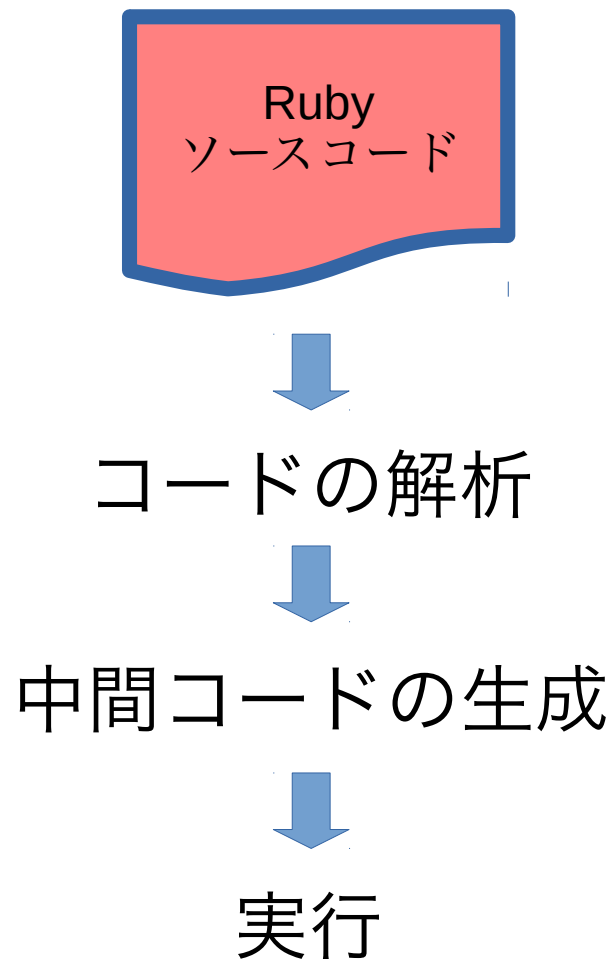
かなり無理

- Rubyインタプリタが多くの資源を必要とする
 - 数十MBのメモリ
 - 数百msの初期化

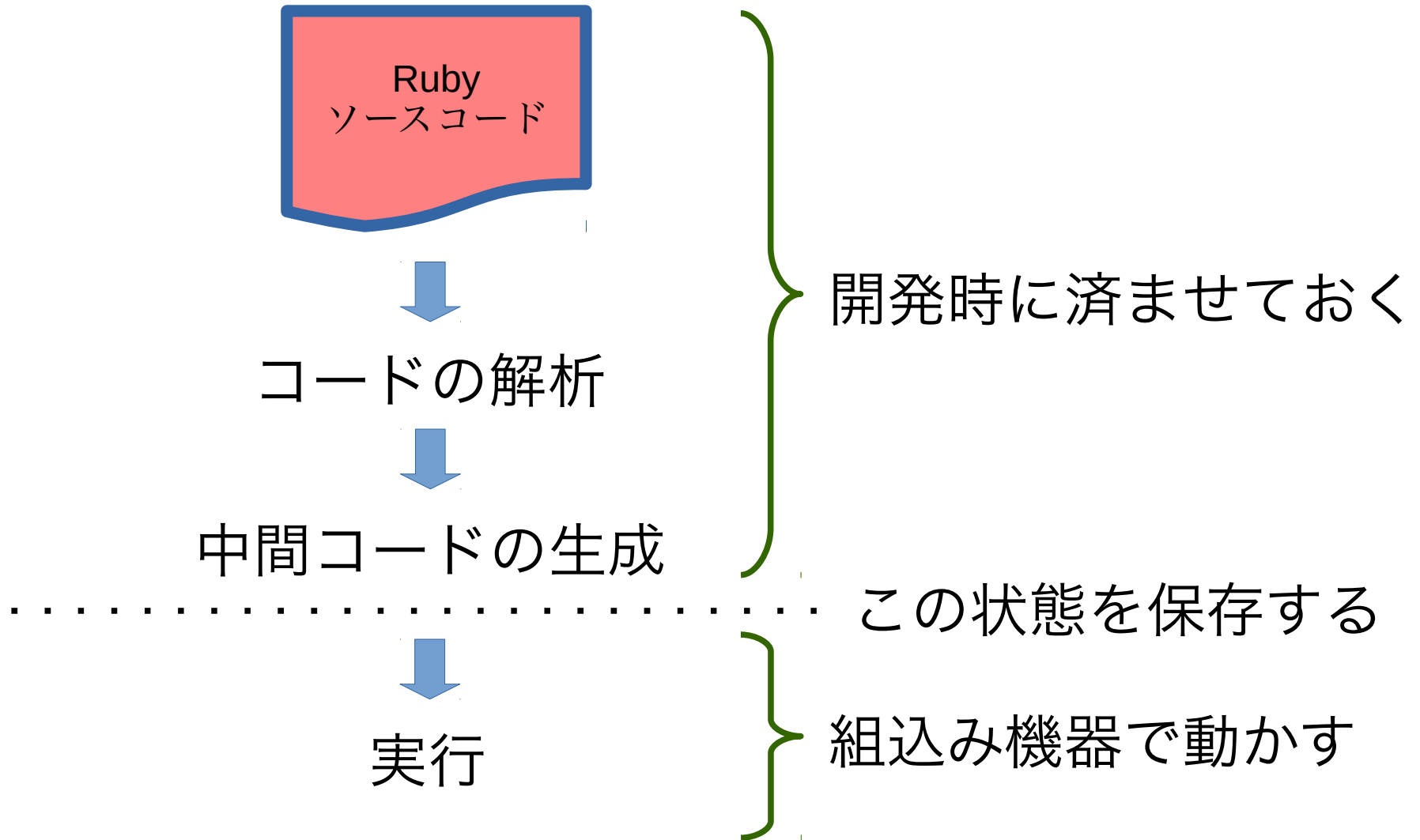
mruby開発へ

- Rubyの特徴（開発者に優しい）を
組み込みソフト開発へ導入する
- 経済産業省
地域イノベーション創出研究開発事業
「軽量Rubyを用いた組み込みプラットフォーム
の研究・開発」
- 2010年～2012年

Rubyプログラムの実行



mrubyの動作



コンパイル

- C言語やJavaのようなコンパイル動作と同じ？
- C言語やJavaでは、
コンパイルにより動作を事前に最適化して
機械語に変換している（コンパイル処理）。
- コンパイル後の機械語を実行する。
- アイディアとしてはCやJavaと同じ
しかし．．．

mruby

- Rubyコンパイラを作れば良い！

Rubyは動的な言語である

- CやJavaのようなコンパイルでは不十分
Rubyの良さが失われてしまう

(参考) 動的とは？

```
a = 15  
p a * 2
```

```
a = 1.2  
p a * 2
```

```
a = "abc"  
p a * 2
```

```
a = [1, 2]  
p a * 2
```

「a」の内容によって
「a * 2」の動作が変わる.

機械語を生成できない

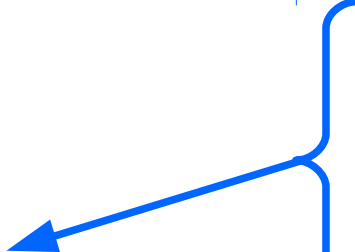
(参考) C言語では

```
#include <stdio.h>

int main(void)
{
    int a1 = 15;
    double a2 = 1.2;

    printf("%d¥n", a1 * 2);
    printf("%f¥n", a2 * 2);

    return 0;
}
```



```
a = 15
p a * 2

a = 1.2
p a * 2

a = "abc"
p a * 2

a = [1, 2]
p a * 2
```

(参考) 動的とは？

- 先の例は, C++の仮想関数でも実現できる
- では以下のようなプログラムは？

```
def func(n)
  return n+1
end

if rand(2)==1 then
  def func(n)
    return n*2
  end
end

p func(5)
```

mruby

- Rubyに特化したコンパイラと実行環境

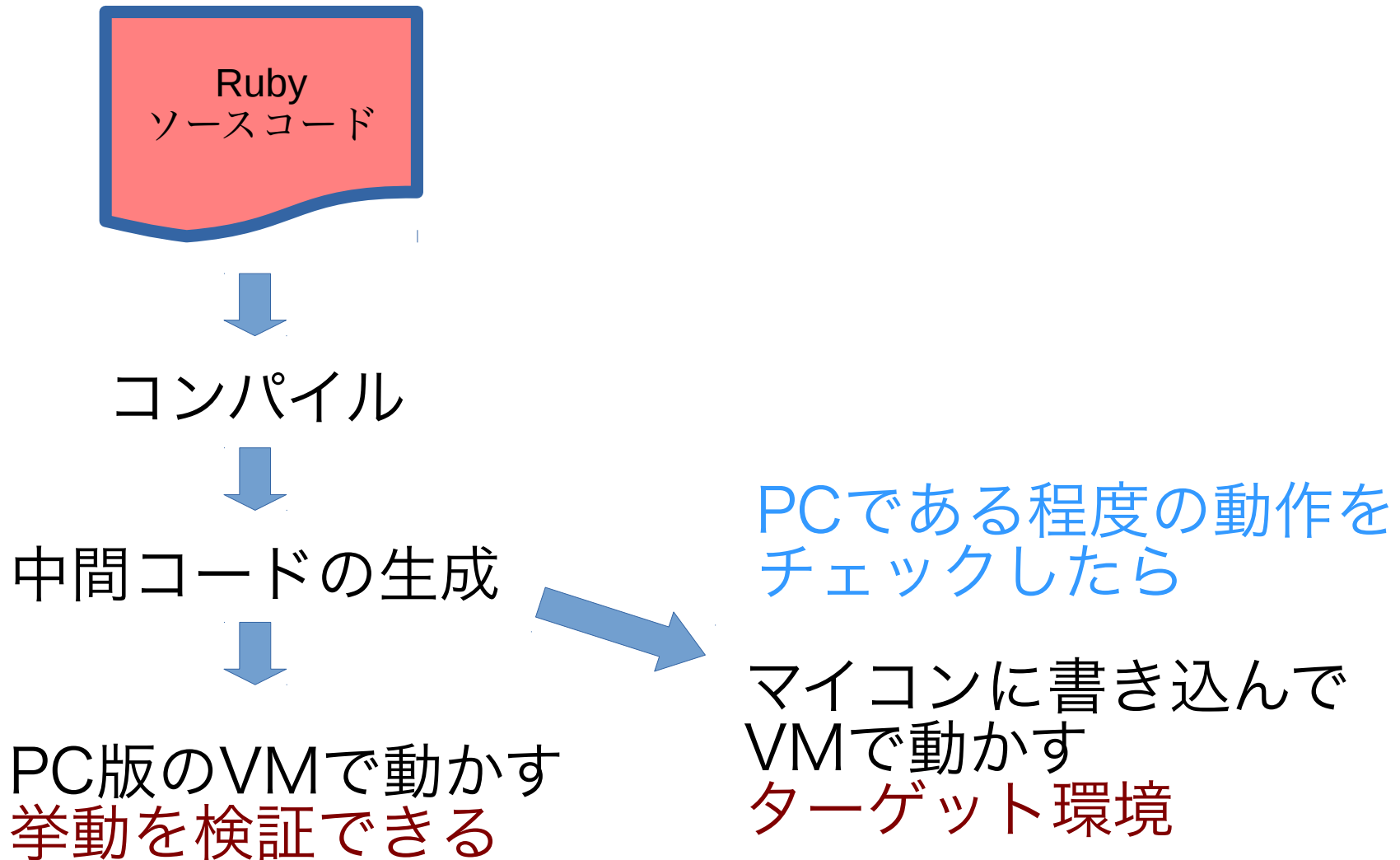
さらに・・・

- さまざまな実行環境に対応する
 - 他機種への対応：
 - 実行環境に依存する部分だけを
実装するだけで良い

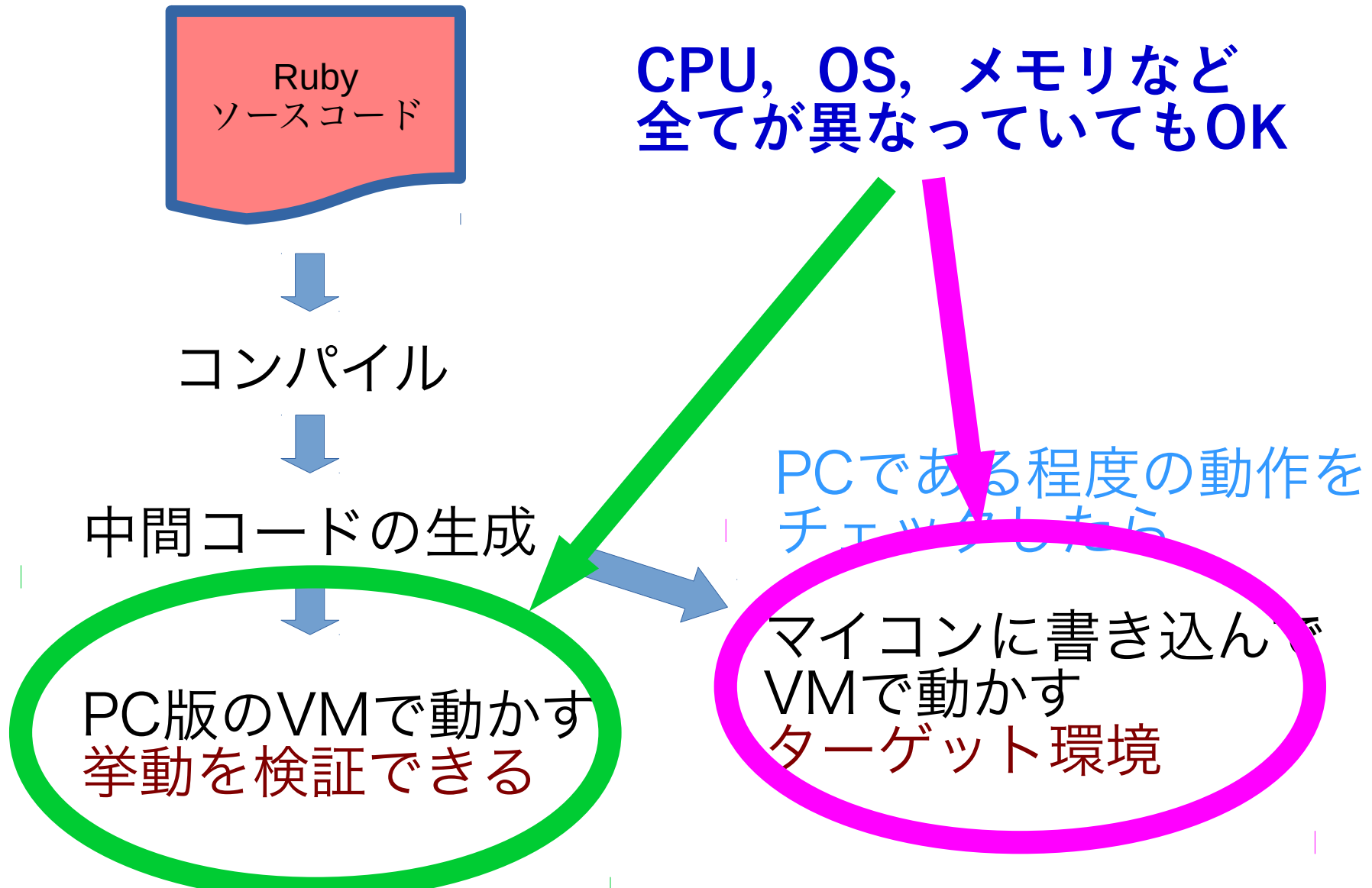
VMで実行することで 可能となること

- VMさえ移植すれば，どのような環境でも動作する！
- VMのビルドに必要なのは，Cコンパイラだけ
- マイコン用のVM
- PC用のVM

「mrubyで実現する」 組み込みソフトの開発



「mrubyで実現する」 組込みソフトの開発



DEMO

- エミュレーションと実行

```
loop do
  Raspi.digitalWrite 26, Raspi::LOW
  Raspi.digitalWrite 13, Raspi::HIGH
  Raspi.sleep 1
  Raspi.digitalWrite 13, Raspi::LOW
  Raspi.digitalWrite 5, Raspi::HIGH
  Raspi.sleep 1
  Raspi.digitalWrite 5, Raspi::LOW
  Raspi.digitalWrite 26, Raspi::HIGH
  Raspi.sleep 1
end
```

どうしてこのような実行が可能なのか？

- VMがコードを実行する仕組み
- 中間コードは環境に依存しない形式
- ハードウェアに依存する機能（関数など）は、名前呼び出す

Raspi.digitalWrite 26, Raspi::LOW

↑
VMのライブラリから
digitalWriteの機能を探して
その機能を実行する

mruby関連情報

- mruby本体

<https://github.com/mruby/mruby>

<http://forum.mruby.org/>

- 「Ruby・mruby活用ガイドブック」

中国経済産業局のホームページからダウンロードできます。

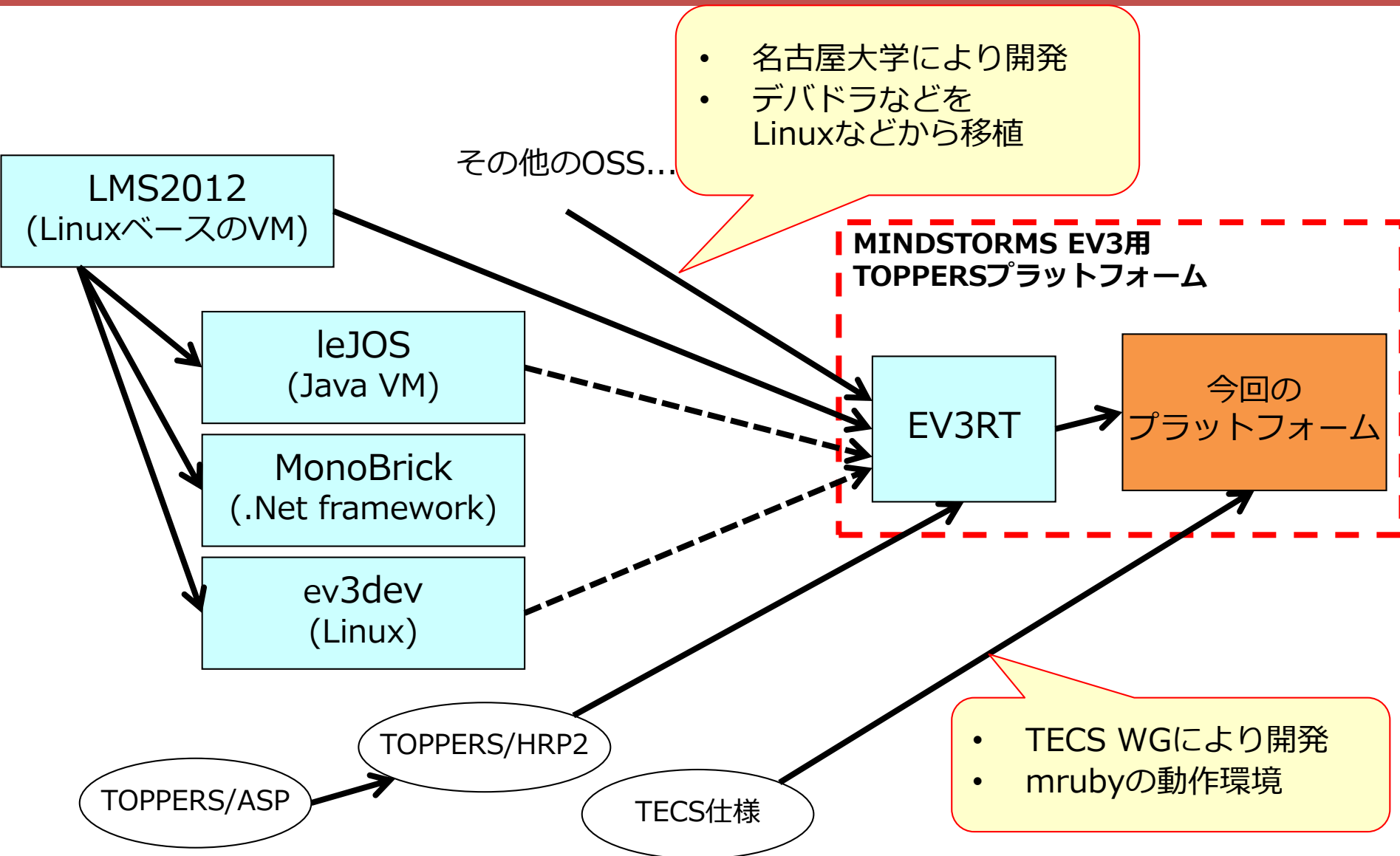
付録：

EV3でmrubyを動かしてみる

資料作成協力者

安積 卓也	(大阪大学)
石川 拓也	(名古屋大学)
長原 裕希	(立命館大学)
長谷川 涼	(大阪大学)

EV3用プラットフォーム



mruby, TECSを利用可能なPF

- mruby on ev3rt+tecs
 - http://www.toppers.jp/tecs.html#mruby_ev3rt
 - TECSとはTOPPERSプロジェクトで開発しているコンポーネントシステム(アプリケーションをコンポーネントベース開発するためのシステム)
 - ev3rt+tecsはEV3RT上でのアプリケーションのコンポーネントベース開発が可能な環境
 - C++APIのようにEV3のモジュールをTECSコンポーネントとして扱うことができる
 - TECSの提供するmrubyブリッジ機能により, mrubyからev3rt+tecsが提供するモジュール機能を利用できる
 - mrubyからCコードを呼ぶためのI/Fを自動生成

最新版を入手

- パッケージをダウンロードして解凍

- http://www.toppers.jp/tecs.html#mruby_ev3rt

mruby on ev3rt+tecs

EV3RTをベースとしたmruby-on-ev3rt+tecsプラットフォームです。mrubyでEV3を操作できます。スタンドアローン版で、一通りのセンサ、モータをmrubyで利用できます。今後機能を拡張していきます。動作環境、使用方法はreadme.txtを参照してください。詳しくは、下記のドキュメントを参照してください。

- doc/EV3RT_mruby_API_Reference.pdf
- doc/mruby_on_ev3rt+tecs_build.pdf
- doc/mruby_sample.pdf

mruby on ev3rt+tecs		
パッケージ	サイズ	リリース日
mruby-on-ev3rt+tecs_package-alpha1.0.2.tar.gz	17.9MB	2015-07-12
mruby-on-ev3rt+tecs_package-alpha1.0.1.tar.gz	23.8MB	2015-06-17
mruby-on-ev3rt+tecs_package-alpha1.0.0.tar.gz	23.9MB	2015-06-09

- `$ tar xvzf mruby-on-ev3rt+tecs_package-alpha1.0.2.tar.gz`

- ダウンロードしたパッケージ名

環境の構築

- Windows7、 Windows 8、 Windows 8.1
- Cygwinインストール
 - ruby
 - GNU Make
 - bison
- クロスコンパイラ
 - arm-none-eabi-gcc.exe (GNU Tools for ARM Embedded Processors)
https://launchpad.net/gcc-arm-embedded/4.8/4.8-2014-q3-update/+download/gcc-arm-none-eabi-4_8-2014q3-20140805-win32.exe
- mkimage
 - Windows用バイナリはパッケージに同梱
- **Cygwin及びクロスコンパイラのインストールはEV3RTの開発環境構築を参照してください。**
 - http://dev.toppers.jp/trac_user/ev3pf/wiki/DevEnvWin

パスの通し方

- クロスコンパイラをインストールしたディレクトリにPATHを通す
 - C:\Program Files (x86)\GNU Tools ARM Embedded\4.8 2014q1\bin
 - ※フォルダ名は、クロスコンパイラのバージョンごとに変わります。
- arm-none-eabi-gccへパスが通っているかを確認

```
Nagahara@T440-226 ~/mruby_ev3
$ arm-none-eabi-gcc --version
arm-none-eabi-gcc.exe (GNU Tools for ARM Embedded Processors) 4.8.3 20140228 (release) [ARM/embedded-4_8-branch revision 208322]
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

- コンパイラバージョンが表示されない場合：Pathを確認

ディレクトリ構造

- bin
 - Windows向け開発環境のバイナリ
 - mkimageを含む
- doc
 - mrubyリファレンス
 - ビルド手順
 - サンプルプログラムの説明
- hr-tecs
 - TOPPERS/HRP2及びEV3プラットフォーム
 - サンプルプログラム (hr-tecs/workspace/mruby_samples)
- mruby
 - mruby ver1.1.0
 - build_config.rbでEV3用のクロスコンパイルを指定

コンパイル手順

- mrubyのビルド（host用コンパイル→ARM用クロスコンパイル）
- パッケージを展開したディレクトリで
 - \$ cd mruby
 - \$ make
 - mrubyディレクトリでmakeを実行すると、mrubyがビルドされる。

※ビルドには、ruby及びbisonのインストールが必要
下記の出力がされればビルド成功

```
=====
Config Name: ARM
Output Directory: build/ARM
Included Gems:
  mruby-print - standard print/puts/p
  mruby-toplevel-ext - extended toplevel object (main) methods
=====
```

コンパイル手順

• EV3RT+TECS・EV3ドライバ・アプリケーションのビルド準備

– SDカードへコピー

- デフォルトでは、cygwinが使用されていることを想定の上で、Eドライブにコピーされる。
- ドライブを変更するには、サンプルコードのMakefileを編集。

hr-tecs/workspace/mruby_samples/Makefile

- Makefile内を "SD_DIR" で検索して、変数にSDカードドライブを指定
- SDカードドライブのドライブレターに合わせてください
- ここで指定したディレクトリにEV3用イメージファイルがコピーされます

```
# SDのドライブ文字を指定  
SD_DIR = /cygdrive/e/
```

この名前をSDカードのドライブ名に変更

実行したいプログラムを選択

hr-tecs/workspace/mruby_samples/Makefile

```
↓  
# mrubyのアプリケーションファイル名↓  
#APP_RB = battery_sample.rb↓  
APP_RB = button_sample.rb↓  
#APP_RB = color_sample.rb↓  
#APP_RB = color_sample2.rb↓  
#APP_RB = gyro_sample.rb↓  
#APP_RB = lcd_sample.rb↓  
#APP_RB = lcd_sample2.rb↓  
#APP_RB = lcd_sample3.rb↓  
#APP_RB = led_sample.rb↓  
#APP_RB = motor_sample.rb↓  
#APP_RB = motor_sample2.rb↓  
#APP_RB = speaker_sample.rb↓  
#APP_RB = speaker_sample2.rb↓  
#APP_RB = touch_sample.rb↓  
#APP_RB = ultrasonic_sample.rb↓  
↓
```

実行したいアプリアプリケーション
を一つ選ぶ

コンパイル手順

- EV3RT+TECS ・ EV3ドライバ ・ アプリケーションのビルド
 - パッケージを展開してディレクトリで
 - `$ cd hr-tecs/workspace/mruby_samples/`
 - `$ make tecs`
 - GNU Makeがtecsgenを実行してくれます
 - `$ make depend`
 - ファイルの依存関係を抽出します
 - ヘッダファイルなど、読み込まれるファイルを更新していなければ、実行する必要ありません。
 - `$ make`
 - カレントディレクトリをサンプルアプリケーションのディレクトリに移動し、makeを行うことでコンパイルが行える。

コンパイル手順 : ① make tecs

```
$ make tecs
  TECSGEN tEV3Sample.cdl
../../tecs/tecs/tecs/tecs.exe -k euc -R -D TECS -D TECS_CPP ¥
```

```
==== rDomainBridge is included in["rDomainBridge", "rDomainApplication"]
==== end check my domain HRP2Kernel ====
==== begin check regions HRP2Kernel ====
["OMIT", "OMIT", "OMIT", "OMIT"]
Array
"TACP_KERNEL"
"TACP_KERNEL"
"TACP_KERNEL"
"TACP_KERNEL"
acv = ["TACP_KERNEL", "TACP_KERNEL", "TACP_KERNEL", "TACP_KERNEL"]
==== end check regions HRP2Kernel ====
==== end tKernel plugin ====
touch tecs.timestamp
```



tecs.timestampが出力されれば成功

コンパイル手順 : ②make depend

```
$ make depend
if ! [ -f Makefile.depend ]; then ¥
    rm -f kernel_cfg.timestamp kernel_cfg.h kernel_cfg.c kernel_mem2.c ; ¥
    rm -f cfg1_out.c cfg1_out.o cfg1_out cfg1_out.syms cfg1_out.srec; ¥
    rm -f makeoffset.s offset.h; ¥
fi
rm -f Makefile.depend
CC      ../../arch/arm_gcc/common/start.S
CFG[1]  cfg1_out.c
CC      cfg1_out.c
LINK    cfg1_out
NM      cfg1_out.syms
OBJCOPY cfg1_out.srec
CFG[2]  kernel_cfg.timestamp
touch -r kernel_cfg.c kernel_cfg.timestamp
CFG[3]  offset.h
Generating Makefile.depend.
touch omit_svc.h
```

 touch omit_svc.hが生成されてからしばらく待つ
(処理中) : 長い場合数分

コンパイル手順 : ③make

```
$ make
```

```
../../../../bin/mkimage.exe -A arm -O linux -T kernel -C none -a 0xc0008000 -e 0xc0008000 -n  
"hrp2 kernel" -d hrp2.bin uImage  
Image Name:   hrp2 kernel  
Created:      Mon Jun 08 21:38:55 2015  
Image Type:   ARM Linux Kernel Image (uncompressed)  
Data Size:    1037732 Bytes = 1013.41 kB = 0.99 MB  
Load Address: c0008000  
Entry Point:  c0008000  
chmod +x uImage  
cp uImage /cygdrive/h/
```



このような出力が出れば成功

下記のようなエラーになる場合は、SD_DIRの指定が間違っている
もしくは、SDカードを認識していない（ささっていない）



```
chmod +x uImage  
cp uImage /cygdrive/e/  
cp: 通常ファイル `/cygdrive/e/' を作成できません: No medium found  
Makefile:543: ターゲット 'uImage' のレシピで失敗しました  
make: *** [uImage] エラー 1
```

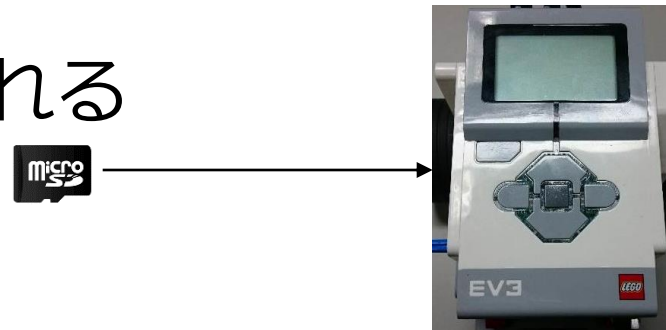
コンパイル手順 : ulmage

- ビルドの確認

- mkimageが生成され、ulmageがコピーされていれば、ビルド成功です

```
../../../../bin/mkimage.exe -A arm -O linux -T kernel -C none -a 0xc0008000 -e 0xc0008000 -n  
"hrp2 kernel" -d hrp2.bin ulmage  
Image Name:      hrp2 kernel  
Created:         Mon Jun 08 21:38:55 2015  
Image Type:      ARM Linux Kernel Image (uncompressed)  
Data Size:       1037732 Bytes = 1013.41 kB = 0.99 MB  
Load Address:    c0008000  
Entry Point:     c0008000  
chmod +x ulmage  
cp ulmage /cygdrive/h/
```

- SDカードのルートディレクトリに、ulmageというファイルができます
- SDカードをEV3本体に入れる



本体操作基本操作

中央 (Enter) ボタンで
電源オン



OS起動中



LEDが赤
の間はOS
の起動中

中央 (Enter) ボタンで
mrubyプログラム開始



LEDが緑になればOS起動完了

戻る (Back) ボタン
長押しで電源オフ

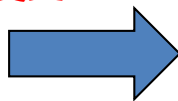


コンパイル手順：サンプルプログラムの変更の場合

サンプルプログラムの変更後はmakeだけで良い
(make tecs, make dependは一度だけ)

```
# mrubyのアプリケーションファイル名
#APP_RB = battery_sample.rb
APP_RB = button_sample.rb
#APP_RB = color_sample.rb
#APP_RB = color_sample2.rb
```

サンプル
プログラムの
変更



```
# mrubyのアプリケーションファイル名
#APP_RB = battery_sample.rb
#APP_RB = button_sample.rb
APP_RB = color_sample.rb
#APP_RB = color_sample2.rb
```



make

ev3way_sample

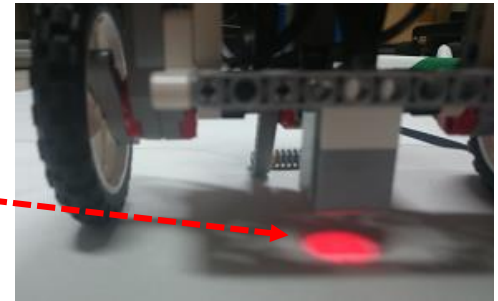
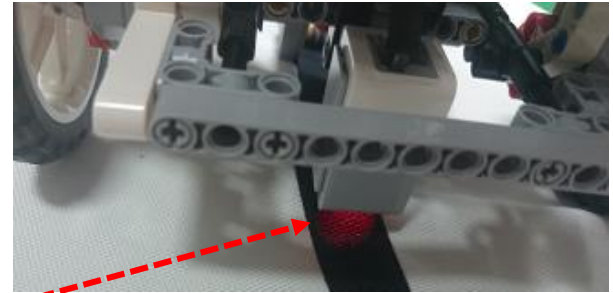
- ETロボコン用のサンプル
 - 倒立制御しながら、ライントレースを行う
- タッチセンサ (:port_1)
- カラーセンサ (:port_2)
- ジャイロセンサ (:port_3)
- 超音波センサ (:port_4)
- しっぽモータ (:port_a)
- 右モータ (:port_b)
- 左モータ (:port_c)



ev3way_sample

• 操作手順

- 電源を入れる
- 黒色のライン上にカラーセンサを移動
- タッチセンサを押す：黒色の値を取得
- 白色の上にカラーセンサを移動
- タッチセンサを押す：白色の値を取得
：しっぽを下ろす
- ライン上移動
- タッチセンサを押す：ラインレーススタート



ev3way_sample.rb : 初期化

begin

```
LCD.puts "ev3way_sample.rb"  
LCD.puts "--- mruby version ---"  
Speaker.volume = 1  
forward = turn = 0
```

ひとつしかないもの（ポート番号指定不要）は、
クラスメソッドとして直接呼び出す

initialize sensors

```
$sonar = UltrasonicSensor.new(SONAR_SENSOR)  
$color = ColorSensor.new(COLOR_SENSOR)  
$color.reflect  
$touch = TouchSensor.new(TOUCH_SENSOR)  
$gyro = GyroSensor.new(GYRO_SENSOR)
```

ポート番号を指定して初期化
（インスタンス化）

initialize motors

```
$motor_l = Motor.new(LEFT_MOTOR)  
$motor_r = Motor.new(RIGHT_MOTOR)  
$motor_t = Motor.new(TAIL_MOTOR)  
$motor_t.reset_count
```

Signal calibration

```
LED.color = :orange
```

...

ev3way_sample.rb : 黒色、白色の取得

```
# Calibration
```

```
$black_value = color_calibration
```

```
LCD.puts "black::#{ $black_value}"
```

```
$white_value = color_calibration
```

```
LCD.puts "white::#{ $white_value}"
```

```
threshold = (( $black_value + $white_value ) / 2).round
```

```
# wait start
```

```
LCD.puts "Ready to start"
```

タッチセンサが押されるまで待つ

カラーセンサn回取得し、
平均値を取得

ライントレースの
基準値を計算

```
def color_calibration(n=10)
  loop {
    break if $touch.pressed?
    RTOS.delay(10)
  }
  col = 0
  n.times { col += $color.reflect}
  col = (col / n).round
  Speaker.tone(:a4, 200)
  RTOS.delay(500)
  col
end
```

ev3way_sample.rb : スタート準備

```
# wait start
LCD.puts "Ready to start"
loop {
  #initialize tail
  tail_control(TAIL_ANGLE_STAND_UP)
  RTOS.delay(10)
  # Touch sensor start
  break if $touch.pressed?
}
# reset motor encoder
$motor_l.reset_count
$motor_r.reset_count
# reset Gyro sensor
$gyro.reset

# Signal start status
LED.color = :green
```

しっぽの位置を指定された角度に保つ
(フィードバック制御)

```
def tail_control(angle)
```

目標値

現在の値

```
pwm = ((angle - $motor_t.count) * P_GAIN).to_i
pwm = (pwm > PWM_ABS_MAX) ? PWM_ABS_MAX :
(pwm < -PWM_ABS_MAX) ? -PWM_ABS_MAX : pwm
$motor_t.power = pwm
$motor_t.stop(true) if pwm == 0
```

```
end
```

ev3way_sample.rb : ライトトレース

障害物まで一定の距離以下になると止まる

```
# main loop
forward = turn = 0
loop {
  start = RTOS.msec
  # up tail
  tail_control(TAIL_ANGLE_DRIVE)
  if sonar_alert
    forward = turn = 0
  else
    # Line trace
    turn = $color.reflect >= threshold ? 20 : -20
    forward = 30
  end
  ...
}
```

サンプルでは、30に固定

```
def sonar_alert
  $sonar_counter += 1
  if $sonar_counter == 10
    distance = $sonar.distance
    $sonar_alert = distance <=
      SONAR_ALERT_DISTANCE
      && distance >= 0
    $sonar_counter = 0
  end
  $sonar_alert
end
```

カラーセンサと閾値と比較し
どちらかに曲がる
ここを変更すると、
自前のライトトレースが可能

ev3way_sample.rb : 倒立制御

```
# main loop
loop {
  start = RTOS.msec
  ...
  # call balance control API
  pwm_l, pwm_r = Balancer.control(
    forward.to_f,
    turn.to_f,
    $gyro.rate.to_f,
    GYRO_OFFSET,
    $motor_l.count.to_f,
    $motor_r.count.to_f,
    Battery.mV.to_f)
  $motor_l.stop(true) if pwm_l == 0
  $motor_l.power = pwm_l
  $motor_r.stop(true) if pwm_r == 0
  $motor_r.power = pwm_r
  wait = 4 - (RTOS.msec - start)
  RTOS.delay(wait) if wait > 0
}
```

バランスの返り値が2つ

C言語で実装されたバランスを呼び出す

4ミリ秒周期で実行
現状1ミリ秒程度で処理完了
mubyでも十分制御可能

パッケージ内のドキュメント

- doc/EV3RT_mruby_API_Reference.pdf
 - 提供しているサンプル
- doc/mruby_on_ev3rt+tecs_build.pdf
 - 環境設定
- doc/mruby_sample.pdf
 - サンプルプログラムの説明

問い合わせ先・リンク

- 質問等は下記のMLに
 - users@toppers.jp
- モニタ募集中
- EV3RT
 - http://dev.toppers.jp/trac_user/ev3pf/wiki/WhatsEV3RT