TOSHIBA
Leading Innovation

組込みソフト開発の新潮流
軽量Rubyのご紹介

― 軽量Ruby“mruby”の特徴から 組込みに適応する方法まで ―

東芝情報システム株式会社
新規事業推進室
情報と人、それを未来へ

2012/11/15

東芝グループは、持続可能な地球の未来に貢献します。 eco スタイル

もくじ

情報と人,それを未来へ

- Rubyの概要
- 軽量Rubyの紹介
- 組込みへの適応例（弊社展示デモを例に）
- まとめ
- お知らせ

Rubyの概要

情報と人,それを未来へ

- オブジェクト指向プログラミングに対応したスクリプト言語

- 日本人である"まつもと"氏が開発
  - 日本語の書籍が充実している!
  - “「楽しくプログラミング」できる機能を盛り込んだ”

- 世界で使われる言語
  → 人気ランキング**5位**

- 米Twitter社、楽天、クックパッド等
  大手サイトでの利用実績
  → 戦略技術としてRubyを据える企業も!!

- 来年で誕生20周年を迎える

プログラミング言語の人気度ランキング

1. JavaScript
2. Java
3. PHP
4. Python
5. **Ruby**
6. C#
7. C++
8. C
9. Objective-C
10. Shell
11. Perl
12. Scala
13. Haskell
14. ASP
15. Assembly
16. ActionScript
17. R
18. Visual Basic
19. CoffeeScript
20. Groovy

（2012年9月米産業調査会社 RedMonk調べ）

Rubyの特徴

- **Rubyのコミュニティのサイトによると…**
  - シンプルな文法
  - 普通のオブジェクト指向機能(クラス、メソッドコールなど)
  - 特殊なオブジェクト指向機能(Mixin、特異メソッドなど)
  - 演算子オーバーロード
  - 例外処理機能
  - イテレータとクロージャ
  - ガーベージコレクタ
  - ダイナミックローディング(アーキテクチャによる)
  - 移植性が高い（様々なOS上で動作）

Rubyの評価（1/2）

情報と人,それを未来へ

- **XMLの父であるSun Microsystems社のTim Bray氏**
  - 美しいデザイン
  - **“Rails”がある**
  - ビジネス視点ではTime to Marketの短縮に貢献してくれる
  - 親切で協力的なコミュニティ
  - 実績として記述されたコードの保守性が良い
  - 実績としてセキュアなシステムが多い

Rubyがヒットするきっかけになった
Webアプリケーションのフレームワーク “Ruby on Rails”

- **Ruby on Rails作者David Heinemeier Hansson氏**

「なぜ(Railsを作るのに)Rubyを選んだのですか？」

「極端なことを言うと、Rubyが一番美しく自分のコードが書けるからです。」

Rubyの評価(2/2)

情報と人,それを未来へ

- **元Amazon Engineer/現Google・Steve Yegge氏 ("バベル案内"より)**

「RubyはPerlの文字列処理とUnix統合をそのまま取り入れ、Perlの最良の部分を手にしている。そして最高のリスト処理をLispから取り入れ、最高のオブジェクト指向をSmalltalkとその他の言語から、最高のイテレータをCLUから。あらゆることの最良の部分をあらゆるところから取り入れた」

… ここがRubyが楽しくプログラミングできる所以。

「エンジニアの初心者からエキスパートまで自在にプログラムできる」

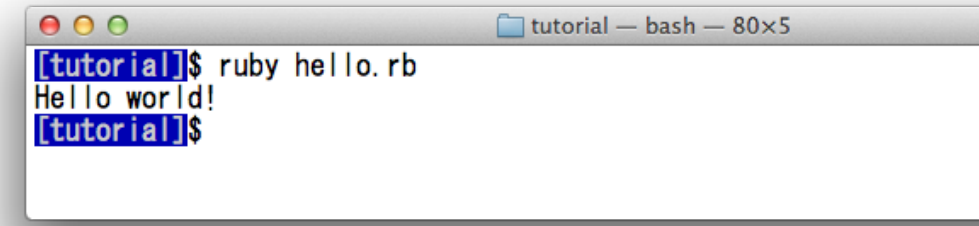
Rubyの簡単な文法と動作を紹介(1/7)

情報と人,それを未来へ

- **まずは “Hello world”…**
  - セミコロンがあってもなくてもよい
  - カッコがあってもなくてもよい
  - コンパイル不要でいきなり実行
    - “#! /.../bin/ruby”
      …と「shebang」を記述すれば
      直接実行可能
- **実行順序は上から下へ**
  - # 以降がコメント
  - 関数定義も先に準備しておく
    - def *func* ～ end
  - 変数もいきなり使える
  - requireで他のファイルも
    読み込める

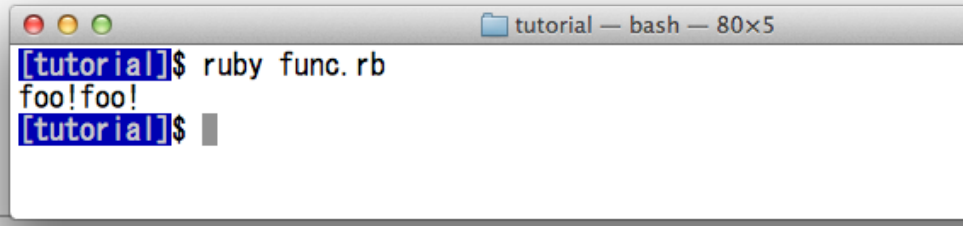
hello.rb

```
puts "Hello World!"
```



func.rb

```
a = "foo!"
b = 2

# foo(a, b)   #<= ここで呼ぶとまだ定義されていないのでエラー

def foo(a, b)
  puts a * b
end

foo(a, b)     #<= 関数呼び出し
```



Rubyの簡単な文法と動作を紹介(2/7)

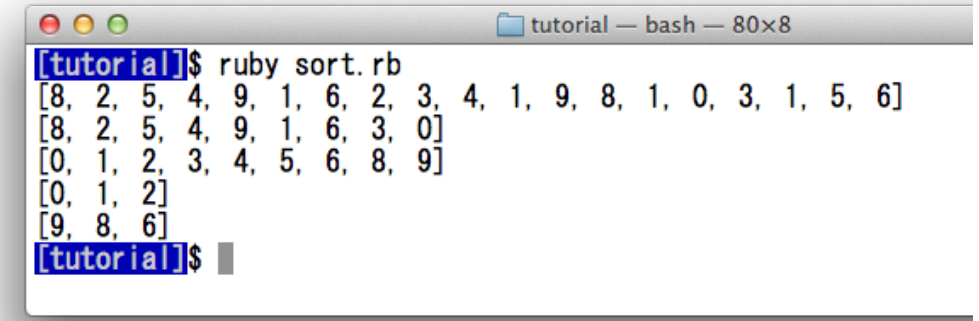
情報と人,それを未来へ

- **あらゆるものがオブジェクト**
  - 整数も変数も文字列も配列も
  - オブジェクトのメソッドを呼び出して操作する
  - メソッドは数珠つなぎで呼び出せる

※ p コマンドはオブジェクトを自動的に分かりやすく表示してくれる

sort.rb

```
a = [8,2,5,4,9,1,6,2,3,4,1,9,8,1,0,3,1,5,6]

p a
p a.uniq
p a.uniq.sort
p a.uniq.sort[0..2]
p a.uniq.sort.reverse[0..2]
```



Rubyの簡単な文法と動作を紹介(3/7)

情報と人,それを未来へ

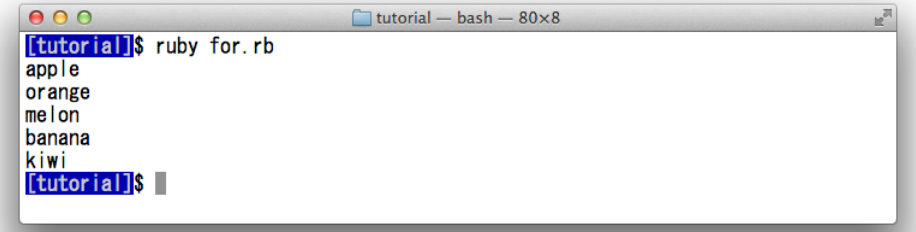
- **繰り返しの表現もいろいろ**
  - シェルスクリプトっぽい記法も可
  - rubyっぽく書くならeach
  - 変数に代入しなくても
    そのまま使える
  - もちろんC言語っぽく
    書くのも可能

for.rb

```
fruits = ["apple", "orange", "melon", "banana", "kiwi"]
for i in fruits do
  puts i
end
```

each.rb

```
["apple", "orange", "melon", "banana", "kiwi“].each do |i|
  puts i
end
```



Rubyの簡単な文法と動作を紹介(4/7)

情報と人,それを未来へ

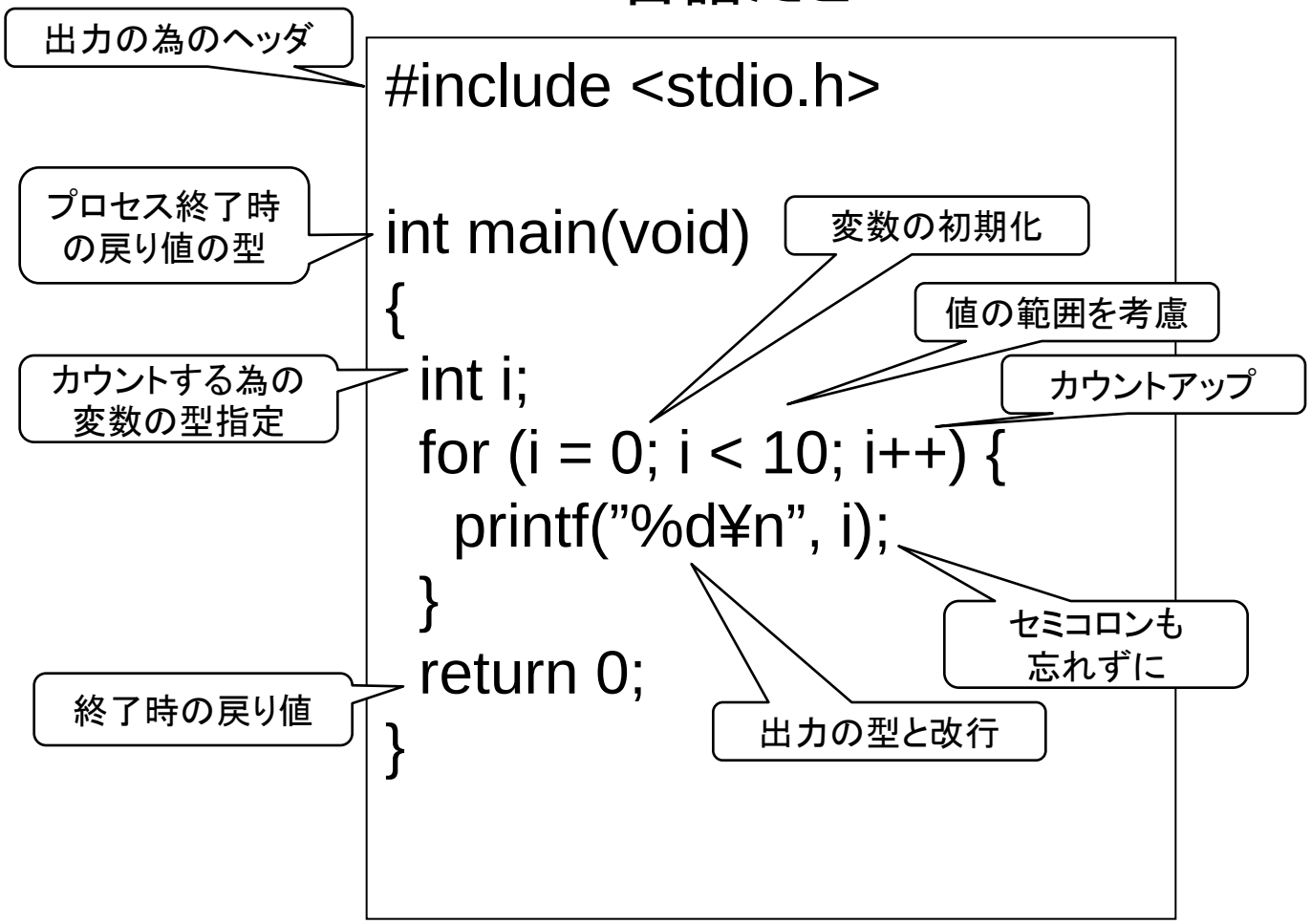
- このようにシンプルで柔軟な文法なので
  - アルゴリズムの表現に
    本来不要な準備がいらない
    → 思考のじゃまをしない

Rubyの簡単な文法と動作を紹介(4/7)

情報と人,それを未来へ

- **例えば、1から10までを出力するのに…**

C言語だと…



Rubyの簡単な文法と動作を紹介(4/7)

情報と人,それを未来へ

Rubyなら…

```
for i in 1..10
  puts i
end
```

Rubyの簡単な文法と動作を紹介(5/7)

情報と人,それを未来へ

- **クラスの記述は class**
  - 継承は “<”
  - メソッドの探索順序は
    オブジェクトが所属するクラス
    →その上位のクラス
    …他の言語と同じ

```
class Robot
  @@vers =  "1.0"
  def say_vers
     print "Version is " + @@vers + "¥n"
  end
end

class Android < Robot
  def initialize(name)
    @name = name
  end

  def say_name
    print "Hi, my name is " + @name + "¥n"
  end
end

obj1 = Android.new("Bob")
obj1.say_name
obj1.say_vers if $DEBUG
```

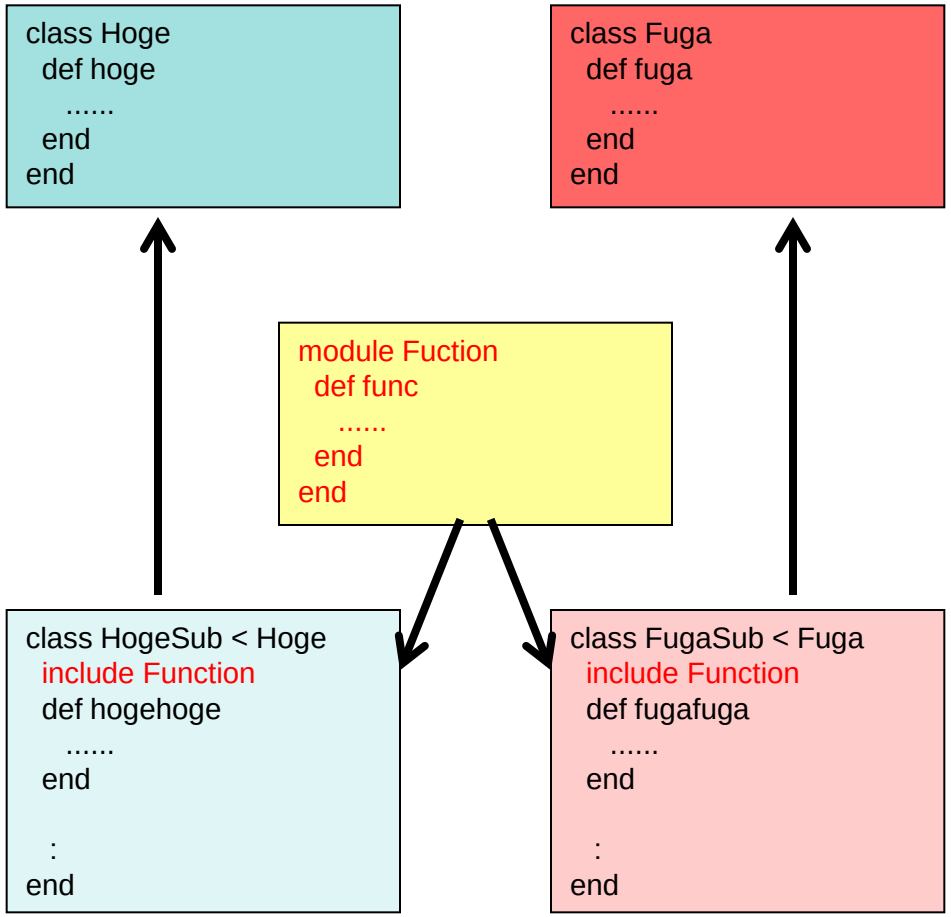
- **変数のスコープと定数**
  - アルファベット大文字始まりは定数
  - 小文字または”_”はローカル変数
  - “@”始まりはインスタンス変数
  - “@@”始まりはクラス変数
  - “$”始まりはグローバル変数
    $_、$1などの(Perlっぽい)組込み変数もある

Rubyの簡単な文法と動作を紹介(6/7)

情報と人,それを未来へ

- **多重継承はない**
  その代わりにMix-in
  - 共通のメソッドをModule化し
    クラスに読み込む(include)
    ことで継承関係のない
    クラス同士で共有化可能

- **もちろんアクセス制御もある**
  - public / protected / private



オブジェクト指向による
構造化プログラミングも
バッチリOK

Rubyの簡単な文法と動作を紹介(7/7)

情報と人,それを未来へ

- クラスもオブジェクトなので実行時に操作できる
  - 実行されるメソッドは
    実行時に判定される
  - Mix-inも実行時に後から可能

    → 実行時に挙動を変えられる

    → 「プログラムをプログラムできる“メタプログラミング”」ができる

このメタプログラミングの機能を利用して
Ruby on Railsのようなデータベースとの連携が容易な
Webアプリを作りやすい環境を作ることができた!!

軽量Rubyとは(1/3)

情報と人,それを未来へ

- **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を用いた組込みプラットフォームの研究・開発」(平成22年度・23年度)の成果によって実現**
- **目的**
  - Rubyを幅広い分野に活かすため、コンパクトな言語仕様の策定と省リソース(メモリ・CPU)に向けた実行環境の改良
- **開発メンバー**
  - 福岡CSK
  - まつもと氏(ネットワーク応用通信研究所)
  - 九州工業大学
  - 福岡県
  - SCSK
  - 東芝情報システム

研究主体

アドバイザ

※弊社は、加えて軽量Ruby評価ボードの開発を担当

軽量Rubyとは（2/3）

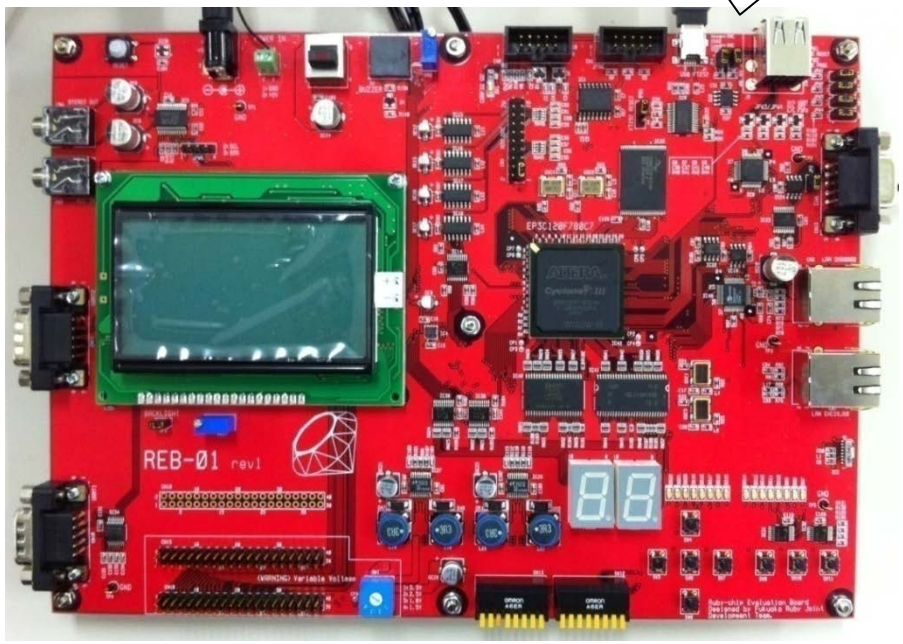
情報と人,それを未来へ

- **軽量Rubyプロジェクトの成果物**
  - 軽量Ruby本体（mruby）

    ※現在GitHub上でオープンソースとして公開中
    今なお改良が続けられている
    - http://github.com/mruby/mruby
  - 軽量Ruby評価ボード
  - ロボット向け評価用実装他

ロボット制御

様々な用途で
評価できるように
周辺I/Oが盛りだくさん



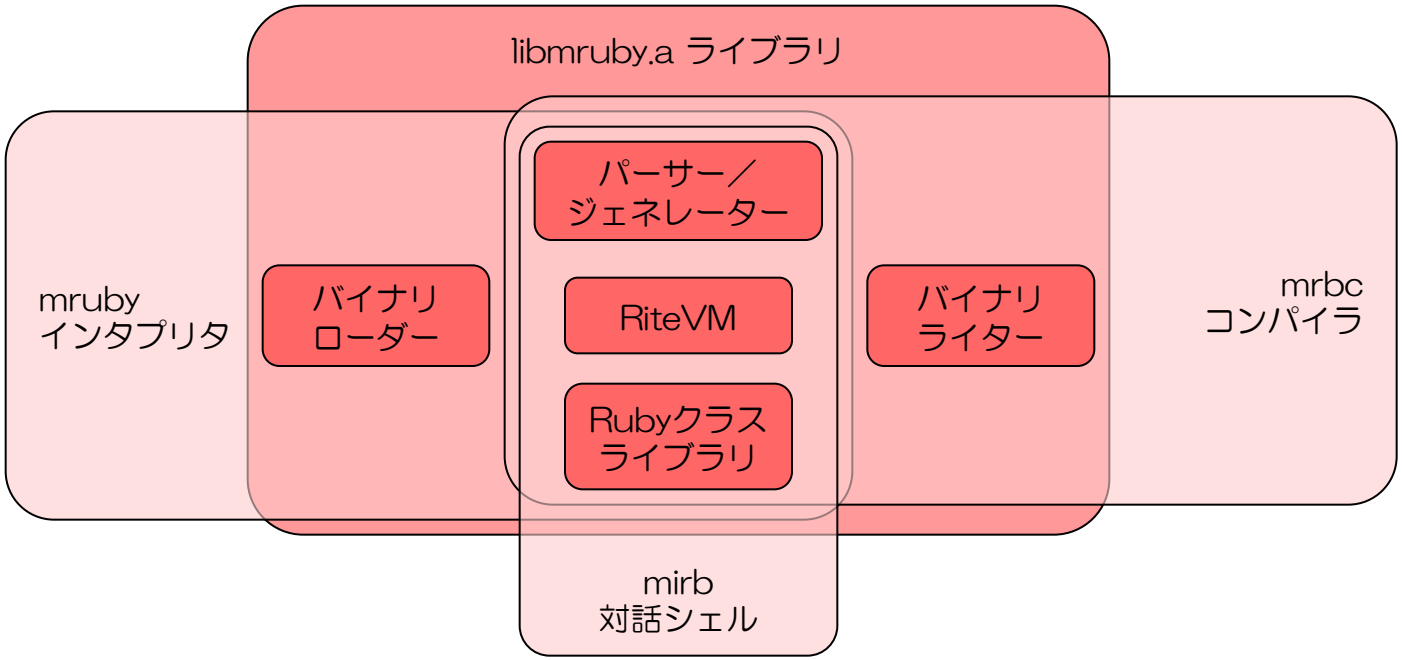
軽量Ruby評価ボード

軽量Rubyとは(3/3)

情報と人,それを未来へ

- **軽量Rubyの構成**
  - mrubyコマンド(インタプリタ)
  - mrbcコマンド(コンパイラ)
  - mirbコマンド(インタラクティブ操作用コマンド(mrubyシェル?))
  - libmruby.aライブラリ(コアコンポーネント兼アプリ組込み用)
    (コンパイラ/VM("RiteVM")機能、Rubyクラスライブラリを含む)

※本家Ruby(CRuby/MRI)には
コンパイラはない



軽量Rubyの特徴(1/2)

情報と人,それを未来へ

- **コンパイル言語として“も”動作**
  - 内部構成がコンポーネント化されているため、従来のインタプリタ型の実行モデルの他、コンパイラと仮想マシン(RiteVM)による実行モデルで動作させることが可能　※後程解説
    → コンパイル済みバイナリはどのRiteVMでも動作させることが可能で、(うまく環境を用意すれば)クロス開発が不要となる
- **高い移植性**
  - C99で規定されたC言語で実装され、様々な環境で動作が可能
    (gccがあればほぼ一発?!)
- **高い互換性**
  - (ISO/IEC 30170 仕様準拠の)Rubyの資産を活用することも可能
  - 従来のRubyと同様、C言語により実装作成されたライブラリも利用することが可能(API名称の変更が必要)

軽量Rubyの特徴(2/2)

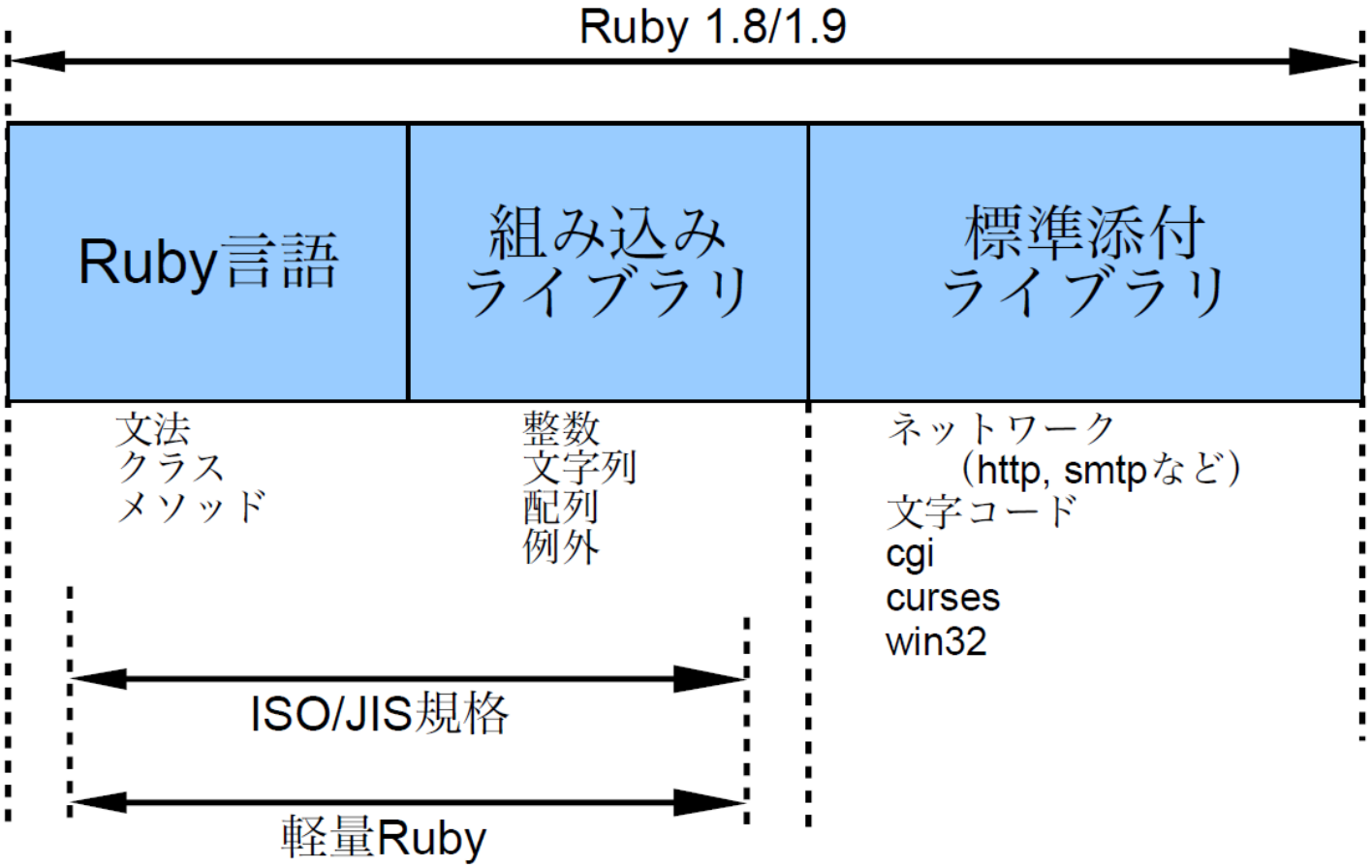
情報と人,それを未来へ

- **細粒度化(≒省資源化)**
  - コンポーネント化やコンフィギュアラブル化することで、動作要件が小さく、組み込みシステムでも動作可能(ファイルシステム不要、OSなしでも動作可能)
    - 使用メモリは、利用するライブラリに依存し、数100Kバイト～数Mバイト程度
- **ソースコードを秘匿可能**
  - コンパイル済みバイナリを配布対象とすることで、ソースコードを秘匿することが可能
- **ハードウェア支援が可能**
  - RiteVMは、ハードウェア支援機能により高速化することが可能
    → “軽量Ruby評価ボード”のFPGAで可能
- **組込みに適したライセンス**
  - MITオープンソースライセンスであるため、組込みでも安心

軽量Ruby言語仕様

情報と人,それを未来へ

- **標準Rubyのサブセットとして ISO/JIS規格に準拠予定**
  - 1.9系の言語仕様に近い



軽量Rubyの注意点

情報と人,それを未来へ

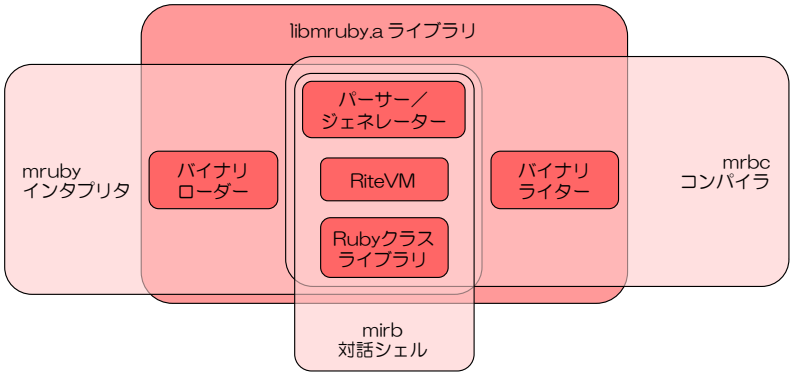
- **現在公開されているmrubyは最小構成であり ISO/JIS規格の全てが実装されているわけではない （今後拡充予定）**
  - 現在はミニマル版
  - スタンダード版でISO範囲を目指し、
  - フル版でCRuby/MRI相当は実現予定

- **動作環境によって（ファイルシステムの有無等）で制約される機能がある**
  - 例）require
    現状、RubyやC言語で実装された外部のクラスライブラリを利用するには、予めlibmruby.aに組み込んでおくか、C言語拡張用APIを利用し、実行時に組み込む必要がある。

様々な実行形態(1/4)

情報と人,それを未来へ

- 動作パターンは全部で5種類
  - インタプリタ実行(①)
  - コンパイル&VM実行
    - バイトコード実行(②)
    - プログラム組み込み
      - バイトコードの読み込み実行(③)
      - C言語変換("-B"オプション)(④)
      - C言語変換("-C"オプション)(⑤)



→ 用途に応じて使い分ける

様々な実行形態(2/4)

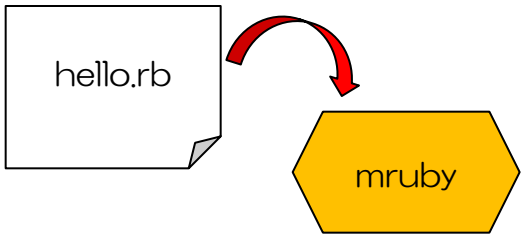
情報と人,それを未来へ

①インタプリタ実行

- Rubyスクリプト(.rb)をmrubyコマンドで直接実行する
- CRubyと同じ感覚で利用可能
- Linuxなど高機能な環境向け

コマンド実行例:

```
$ mruby hello.rb
```

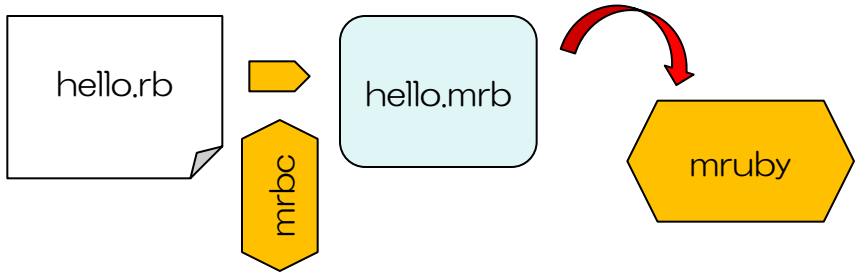


②バイトコード実行

- コンパイラ(mrbc)でバイトコードに変換しmrubyコマンド(-bオプション付き)で実行する
- ソースを秘匿したい場合

コマンド実行例:

```
$ mrbc hello.rb
$ mruby –b hello.mrb
```



様々な実行形態(3/4)

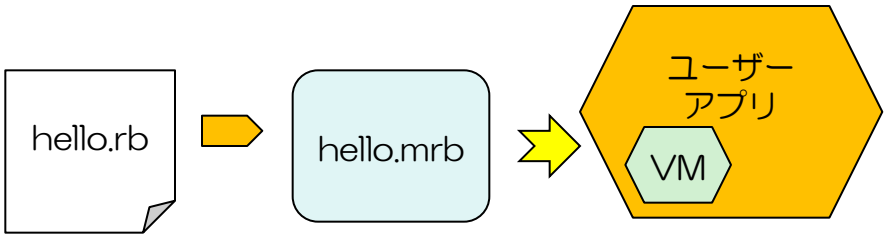
情報と人,それを未来へ

③バイトコードの読み込み実行

- ②と同様にコンパイルし、ユーザーアプリで
  バイトコードを読み込んで実行する
- “mruby”コマンドの–bオプションと
  ほぼ同じ動作をアプリで実施する
- Rubyのプログラムを動的に
  利用したい(アプレット的な利用?)

コマンド実行例:

```
$ mrbc hello.rb
$ gcc app.c –lmruby –o app
(appからhello.mrbをロード)
```

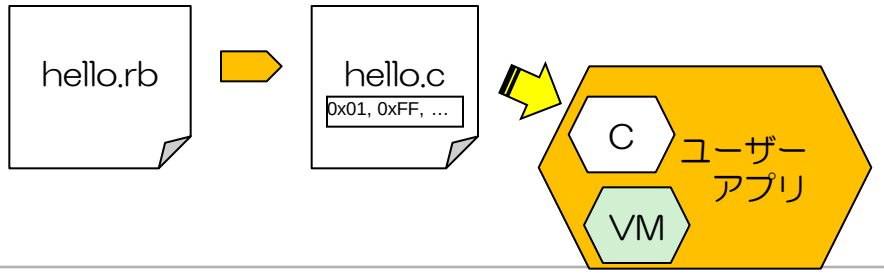


④C言語変換(”-B”オプション)

- RubyスクリプトをC言語の
  バイトコード列(16進)配列に変換し、
  ユーザーアプリと一緒に
  コンパイル&リンクして実行する
- ファイルシステムがない
  組込み向け

コマンド実行例:

```
$ mrbc -Bhello hello.rb
$ gcc app.c hello.c -lmruby -o app
```



様々な実行形態(4/4)

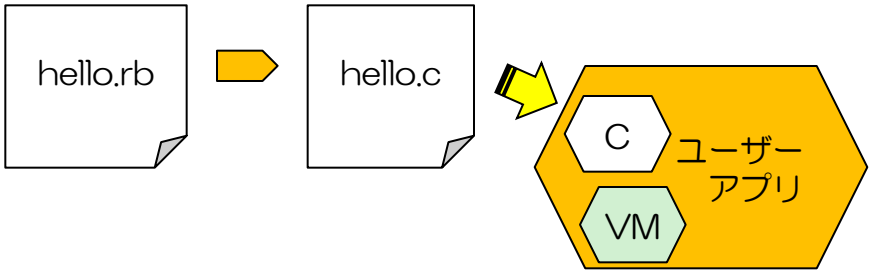
情報と人,それを未来へ

⑤C言語変換(”-C”オプション)

- ④と似ているが、
  Rubyスクリプトを
  内部処理に相当するC言語に変換し、
  ユーザーアプリと一緒に
  コンパイル&リンクして実行する
- mruby内部開発者もしくは
  細かくカスタマイズしたい向け
  (実行速度はやや遅い)

コマンド実行例:

```
$ mrbc -Chello hello.rb
$ gcc app.c hello.c -lmruby -o app
```



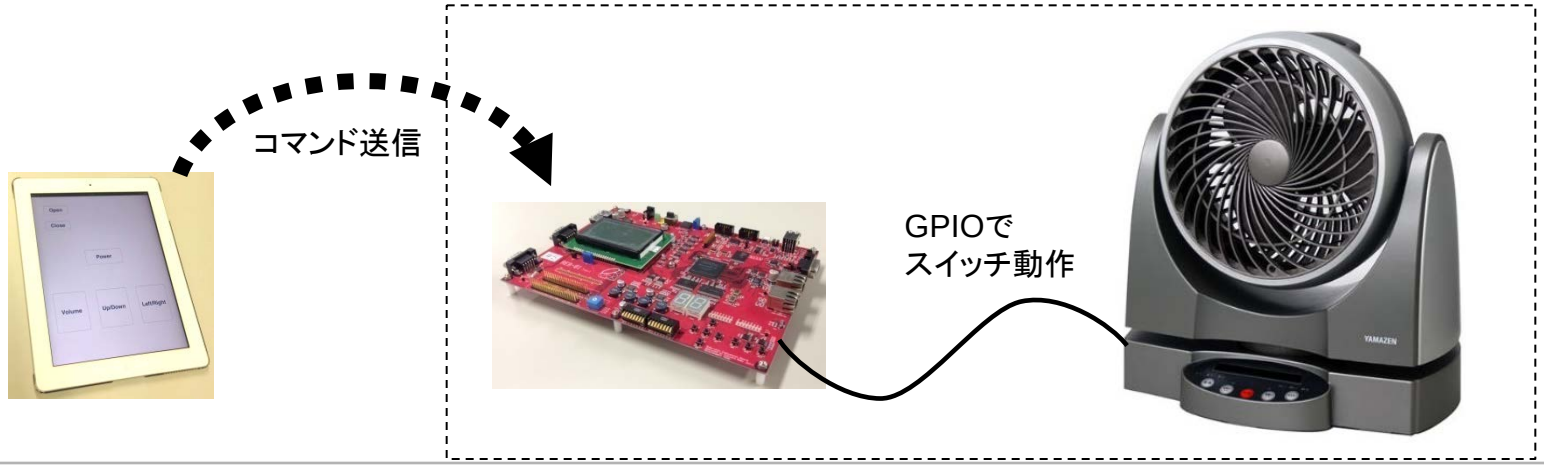
組込みへの適応例

組込みへの適応例(弊社ブースデモ)

情報と人,それを未来へ

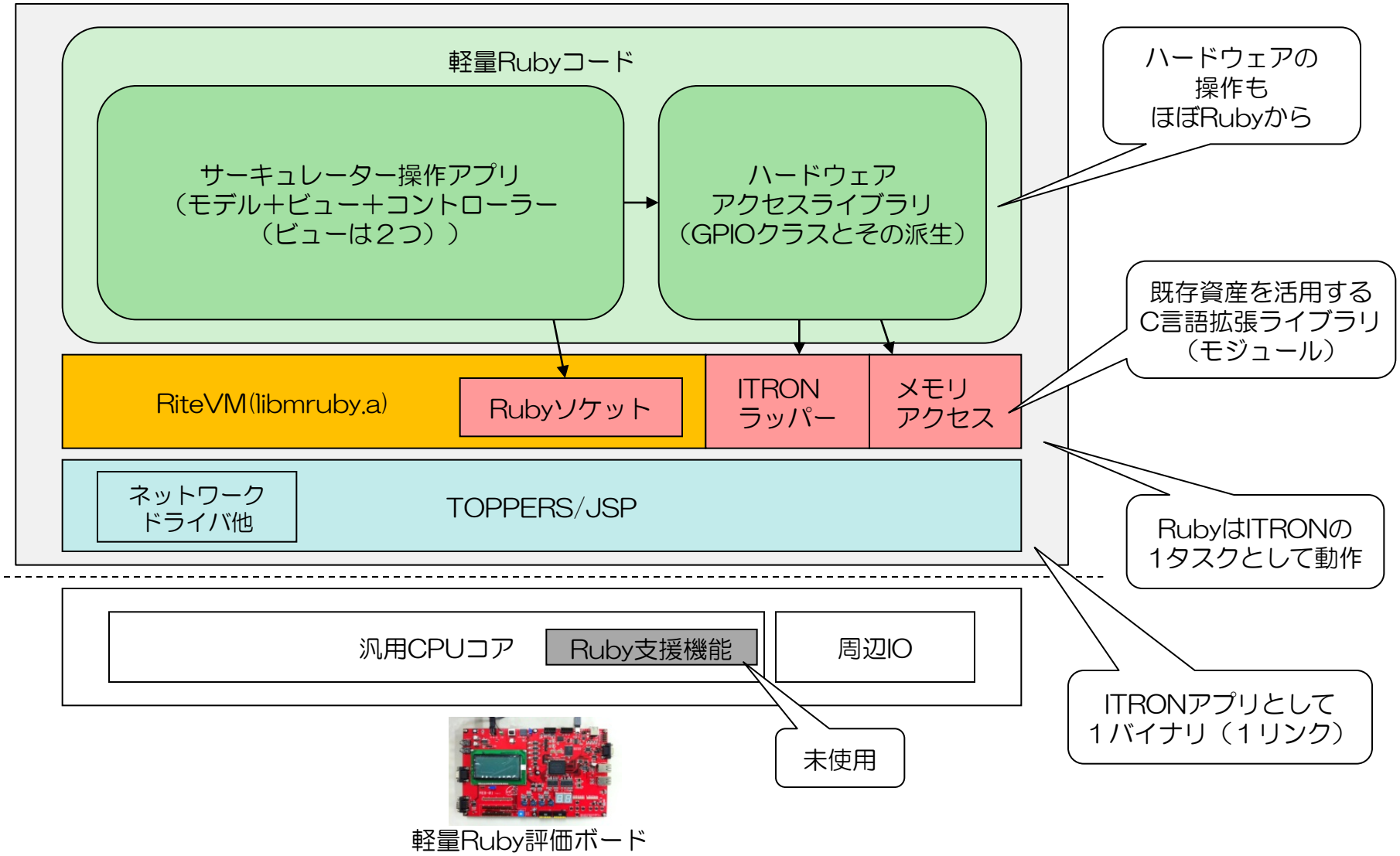
- **サーキュレーターをネットワーク経由で遠隔操作**
  - 軽量Rubyが家電に組み込まれていることを模擬し、
  - タブレットからWiFi経由で
    軽量Rubyで構築されたサーバーにコマンドを送信
  - コマンドに応じてサーキュレーターを駆動
    - On/Off動作などスイッチ動作もほぼRubyで実装
    - Rubyの例外機能を利用し、ネットワークが不正な時に
      ボタンモードへ遷移

    → いわゆる“スマート家電”の簡易版?



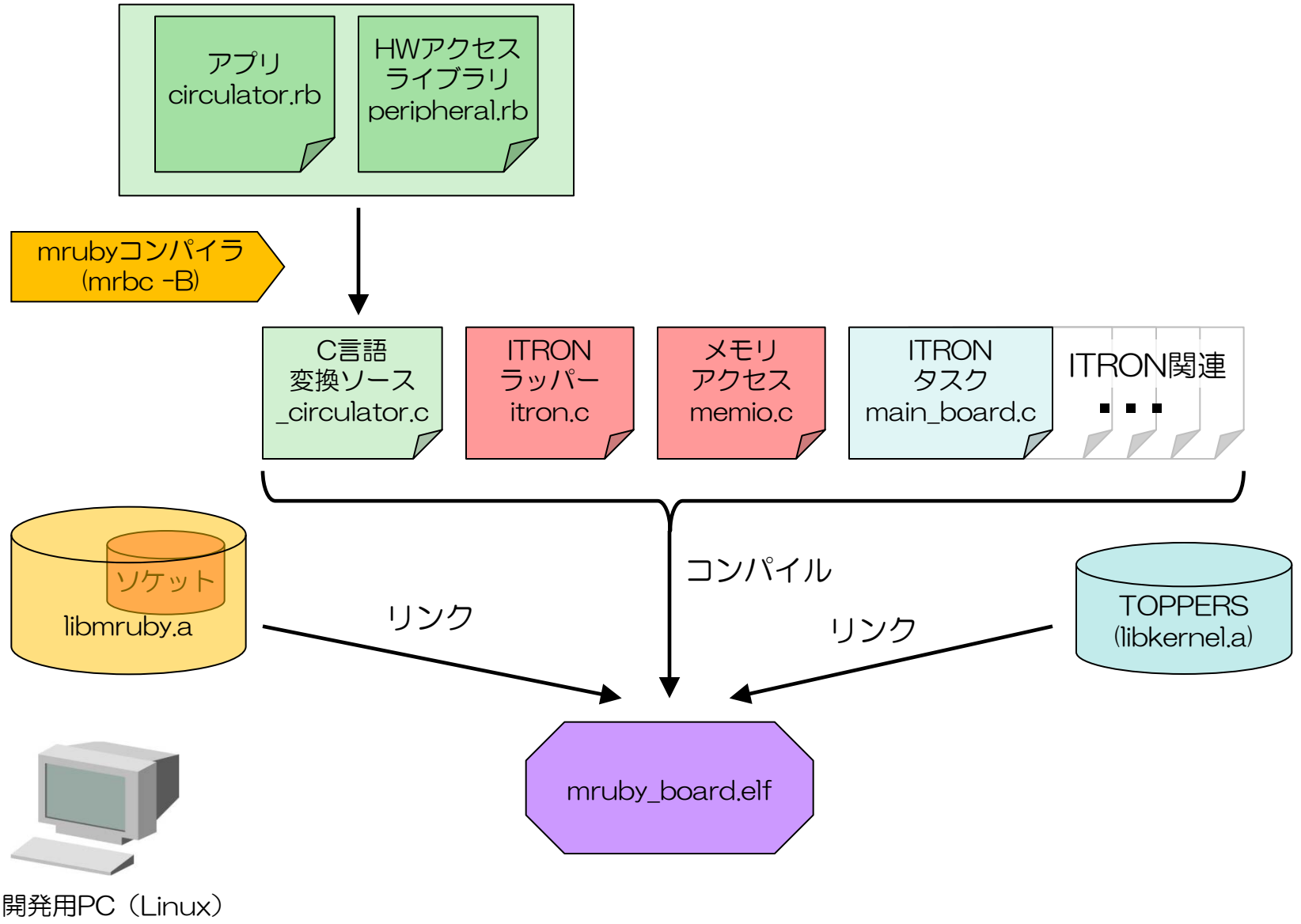
ソフトウェア構成イメージ

情報と人,それを未来へ



軽量Ruby評価ボード

コンパイル実行イメージ



動作環境

情報と人,それを未来へ

- **タブレット側**
  - iPadを使用
  - iOS SDKを利用（CFNetworkによるソケット通信）

    ※iOSアプリ開発にも軽量Rubyが利用できる
    “MobiRuby”というプロジェクトもあります。
- **ボード側**
  - 軽量Ruby評価ボードを使用
    - Altera社製FPGA向けCPU IPコア“Nios2”を搭載
  - TOPPERS/JSPを利用（標準でNios2へ対応済み）
  - 弊社製ITRON向けTCP/IPプロトコルスタック（BSDソケット）
    評価ボードに搭載さているDAVICOM社Etherチップ“DM9000”用
    ドライバを移植

この環境に軽量Rubyを組み込み、アプリを実装

開発環境

情報と人,それを未来へ

- **開発PC**
  - Mac（OS X 10.8）
    → タブレットアプリの開発とシミュレーション環境での動作確認
  - Linux（CentOS 6.3 64bit版、Mac上の仮想環境として）
    → AlteraのツールによるTOPPERS環境のビルドと、
    シミュレーション環境での動作確認
- **開発ツール**
  - Altera社Nios2 Embedded Designe Suite開発環境
    - Nios2向けGNUツールチェーン（nios2-elf-gcc）
    - J-TAGデバッガ（ボードに付属）

開発の手順

情報と人,それを未来へ

⓪ タブレット側アプリの作成

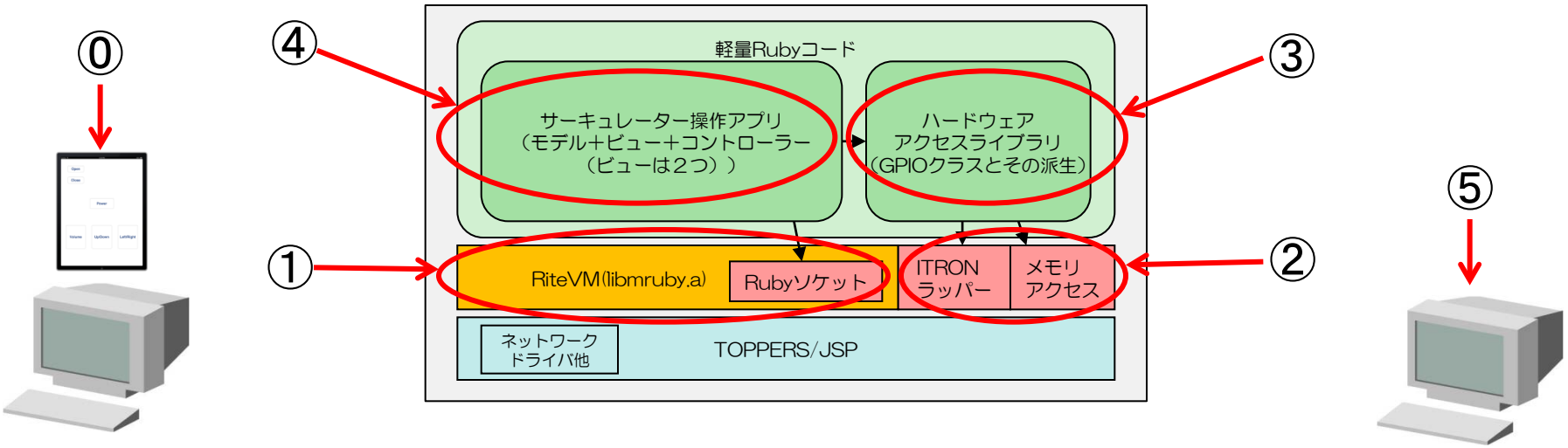
① 軽量Rubyの準備＋ネットワーク動作確認

② デバイスとRTOSアクセス用のC言語拡張ライブラリの作成

③ デバイス操作用モジュールの作成

④ アプリ本体の作成

⑤ シミュレーション用ホスト環境の作成（おまけ）



開発のポイント

情報と人,それを未来へ

⓪ “本家Rubyを利用した並行開発”

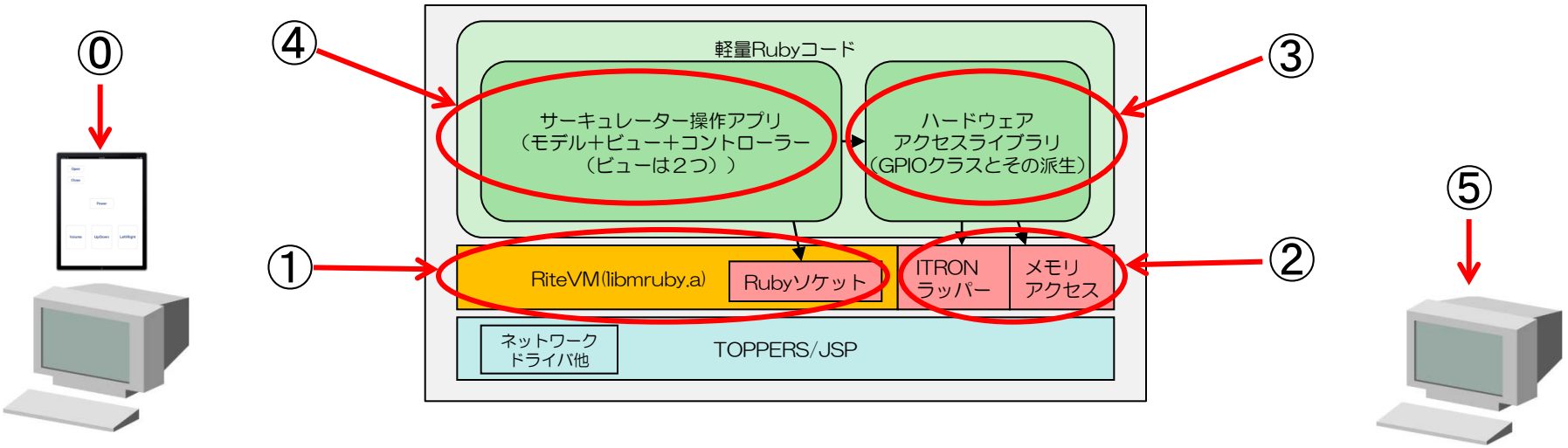
① “クロス環境の構築方法、本家拡張ライブラリの流用方法”

② “既存資産であるC言語ライブラリ等の活用方法”

③ “デバイスドライバとRubyコードの分担方法”

④ “Rubyでのオブジェクト指向プログラミング方法”

⑤ “ホスト環境でのシミュレーションによる開発支援方法”



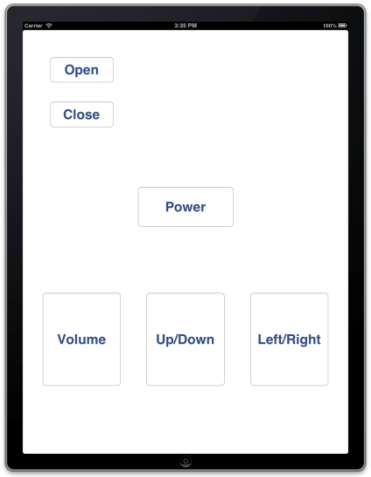
⓪タブレット側アプリの作成

情報と人,それを未来へ

- **本家Rubyを利用した並行開発**
  - 右のようなCRubyによる簡易サーバーを用意（Rubyリファレンスサイトより拝借）

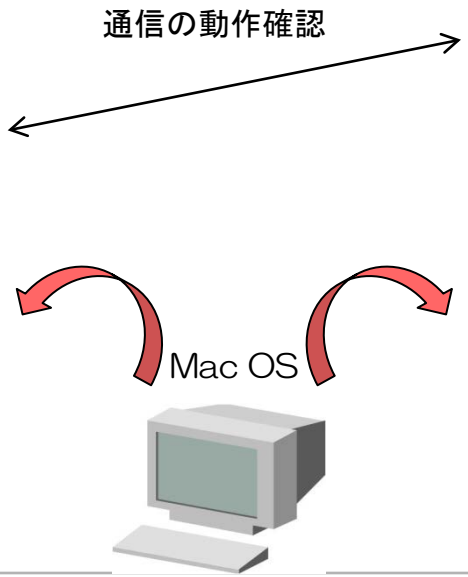
    → これがこのままmrubyでのサーバー処理のベースとして動作可能

    → iOSシミュレーターで動作確認



シミュレータ上で

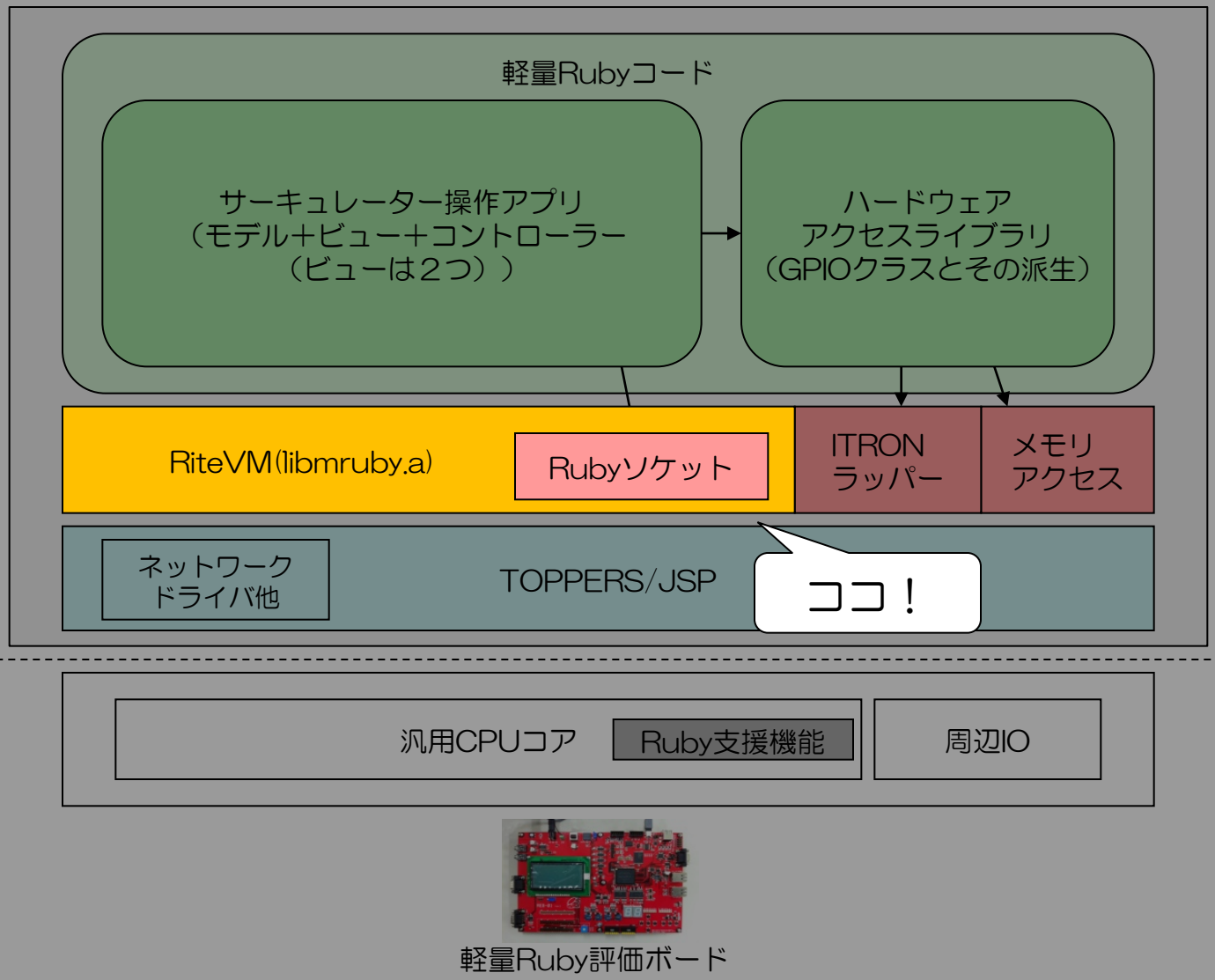
通信の動作確認




```ruby
#!/usr/bin/env ruby

require 'socket'

gs = TCPServer.open(12345)
socks = [gs]
addr = gs.addr
addr.shift
printf("server is on %s\n", addr.join(":"))

while true
  nsock = select(socks)
  next if nsock == nil
  for s in nsock[0]
    if s == gs
      socks.push(s.accept)
      print(s, " is accepted\n")
      else
      if s.eof?
        print(s, " is gone\n")
        s.close
        socks.delete(s)
        else
        str = s.gets
        puts str
        s.write(str)
      end
    end
  end
end
```

Macのターミナル上で

①軽量Rubyのビルド＋動作確認



軽量Ruby評価ボード

①軽量Rubyのビルド＋動作確認

情報と人,それを未来へ

- **クロス環境の構築**
  - Makefileを修正しホスト用とターゲット用の2種類のライブラリ(libmruby.aとlibmruby-nios2.a)とホスト用のmrubyツール群を用意

- **本家Rubyからの拡張ライブラリの流用**
  - ライブラリは2種類ある(RubyかC言語)
  - Rubyベースのライブラリなら ソースツリーの mruby/mrblib の下に 放り込めばOK
  - C言語ベースのライブラリなら
    - mrubyのC言語拡張API用に書き直す
    - 起動時に初期化されるよう初期化関数の追記
  - 最後にlibmruby.aの再ビルド

C言語拡張ライブラリと同じ作法(後述)

ディレクトリ構成;

```
mruby/
  + bin/
  |  + mruby
  |  + mrbc
  |  + mirb
  + include/
  |  + mruby.h
  |  + mrbconf.h
  |  + mruby/
  |     + compile.h
  |     +  :
  |     + ext/
  |        + rubysocket.h
  |        +  :
  + lib/
     + libmruby.a
     + libmruby-nios2.a
```

①軽量Rubyのビルド＋動作確認

情報と人,それを未来へ

- **Rubyソースをコンパイル**

  $ mrbc -Bhello hello.rb

  → hello.c

```
puts "hello world!"
```

⇨

```
const char hello[] = {
0x52,0x49,0x54,0x45,0x30,.
.....
};
```

- **呼び出し部を記述（main.c）**
  1. mrb構造体を確保
  2. バイトコード命令列の読込み
  3. RiteVMを実行
  4. 例外種別の判別
  5. mruby構造体の解放
- **コンパイル&リンク**

  $ nios2-elf-gcc main.c hello.c
  -lmruby -o main.elf

main.c：

```
void
main_task(VP_INT exinf)
{
  /* new interpreter instance */
  mrb_state *mrb;
  int n = -1;
  extern const char hello[];

  mrb = mrb_open();

  n = mrb_read_irep(mrb, hello);

  if (n >= 0) {
   mrb_run(mrb, mrb_proc_new(mrb, mrb->irep[n]),
mrb_top_self(mrb));
    if (mrb->exc) {
     mrb_p(mrb, mrb_obj_value(mrb->exc));
    }
  }
  printf("exit mruby.¥n");

  mrb_close(mrb);

  kernel_exit();
}
```

①軽量Rubyのビルド＋動作確認

情報と人,それを未来へ

- **Rubyソケットの動作確認**
  - hello.rb から右の socket.rb へ差し替えて本家Rubyから流用してきたRubyソケットクラスライブラリの動作確認をします。

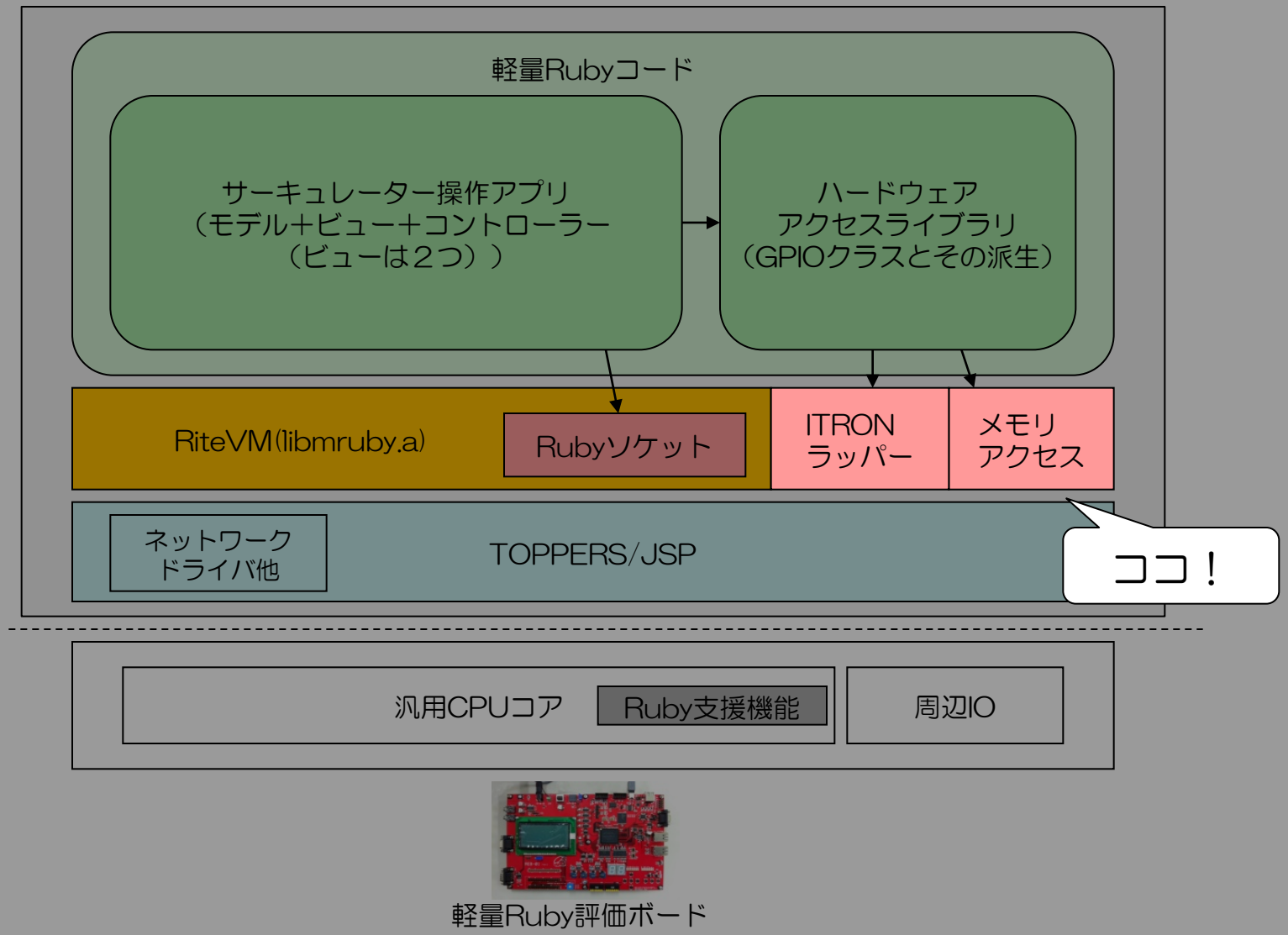
※実は今回作成したRubyソケットクラスライブラリが中途半端なため、実は少々修正した。

socket.rb：

```
#!/usr/bin/env ruby

require 'socket'

gs = TCPServer.open(12345)
socks = [gs]
addr = gs.addr
addr.shift
printf("server is on %s¥n", addr.join(":"))

while true
  nsock = select(socks)
  next if nsock == nil
  for s in nsock[0]
    if s == gs
      socks.push(s.accept)
      print(s, " is accepted¥n")
    else
      if s.eof?
        print(s, " is gone¥n")
        s.close
        socks.delete(s)
      else
        str = s.gets
        puts str
        s.write(str)
      end
    end
  end
end
```

②C言語拡張ライブラリの作成

情報と人,それを未来へ



軽量Ruby評価ボード

②C言語拡張ライブラリの作成

情報と人,それを未来へ

- **既存資産であるC言語ライブラリ等の活用方法**
  - メモリのアクセス用のモジュール “MemIO”
    - （Rubyで低レベルのハードウェアアクセスの実験も兼ねて）
    - TOPPERSのメモリアクセス用API（sil_reb_mem等）を利用
    - バイト幅を指定して、read 、writeができる機能を用意

    →「Ruby の世界にポインタアクセスの穴を開けた」

  - ITRONラッパーのモジュール“ITRON”
    - 単純にRubyからITRONシステムコール（API）呼ぶために用意

②C言語拡張ライブラリの作成

情報と人,それを未来へ

- C言語拡張ライブラリの作成
  - クラスもしくはモジュールの設計
    Rubyからの利用される機能を決める
  - RubyからC言語拡張APIを利用し、
    - “クラス・メソッドの定義をする”部分と
    - “Ruby⇔C言語間の型変換をしつつ処理を実現する”部分の

    2つのパートを実装する

    （JavaでいうところのJNIに相当）

②C言語拡張ライブラリの作成

情報と人,それを未来へ

- **C言語拡張実装についてのポイント**
  - Ruby C言語拡張APIの例
    - mrb_get_args() … Rubyからの引数を変換
    - mrb_fixnum_value( ) … int型等からRubyで使用できるFixnumへ

    - mrb_define_module() … モジュールの定義
    - mrb_define_module_function() … モジュール関数として登録

②C言語拡張ライブラリの作成

情報と人,それを未来へ

- **MemIO.readの例（関数本体）**

```
/*
 * MemIO.read(addr, width = 4)  => 値
 *
 * 引数 addr で指定されたメモリをアクセス幅 width バイトで読み込む
 */
mrb_value
mrb_memio_read(mrb_state *mrb, mrb_value obj)
{
  int argc;
  mrb_int addr, width, value = 0;

  /* 引数のパーズ */
  argc = mrb_get_args(mrb, "i|i", &addr, &width);

  /* 引数 width のチェック */
  if (argc == 1) {
    width = 4;
  }

  /* I/O */
  switch (width) {
  case 1:
    value = sil_reb_mem((VP)addr);
    break;
  case 2:
```

これを使いたかった

```
  case 4:
    value = sil_rew_mem((VP)addr);
    break;
  default:
    mrb_raisef(mrb, E_ARGUMENT_ERROR, "invalid
width specified %d", width);
    break;
  }
  return mrb_fixnum_value(value);
}
```

②C言語拡張ライブラリの作成

情報と人,それを未来へ

- **MemIOの例（定義・初期化部ッソッド定義）**

```
void
mrb_init_memio(mrb_state *mrb)
{
  struct RClass *mrb_memio;
  mrb_memio = mrb_define_module(mrb, "MemIO");

  mrb_define_module_function(mrb, mrb_memio, "read", mrb_memio_read, ARGS_REQ(1)|ARGS_OPT(1));
     :
}
```

MemIOモジュールメソッドとして…

②C言語拡張ライブラリの作成

情報と人,それを未来へ

main.c：

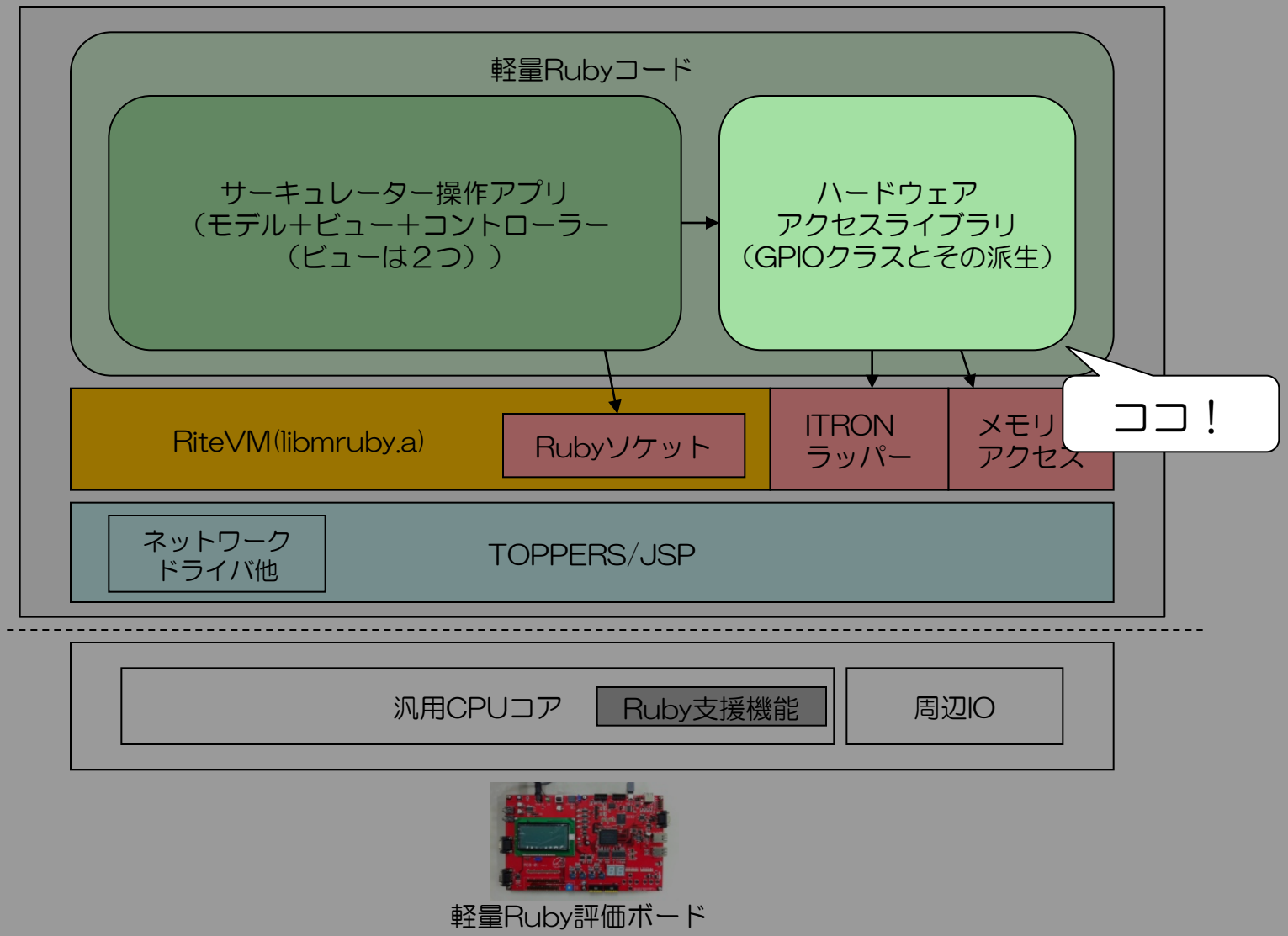
```
void
main_task(VP_INT exinf)
{
  /* new interpreter instance */
  mrb_state *mrb;
  int n = -1;
  extern const char hello[];

  mrb = mrb_open();    // mrb構造体の確保

  mrb_init_uext(mrb);

  n = mrb_read_irep(mrb, hello);

  if (n >= 0) {
    mrb_run(mrb, mrb_proc_new(mrb, mrb->irep[n]),
mrb_top_self(mrb));     // mrubyの実行
    if (mrb->exc) {
      mrb_p(mrb, mrb_obj_value(mrb->exc));
    }
  }
  printf("exit mruby.¥n");

  mrb_close(mrb);

  kernel_exit();
}
```

VMの実行（mrb_run）の前に
初期化関数を呼ぶ
この辺で、
こんな感じに

init_uext.c：

```
#include "mruby.h"

void
mrb_init_uext(mrb_state *mrb)
{
  extern void mrb_init_memio(mrb_state *mrb);
  mrb_init_memio(mrb);
  extern void mrb_init_itron(mrb_state *mrb);
  mrb_init_itron(mrb);
}
```

③デバイス操作用モジュールの作成

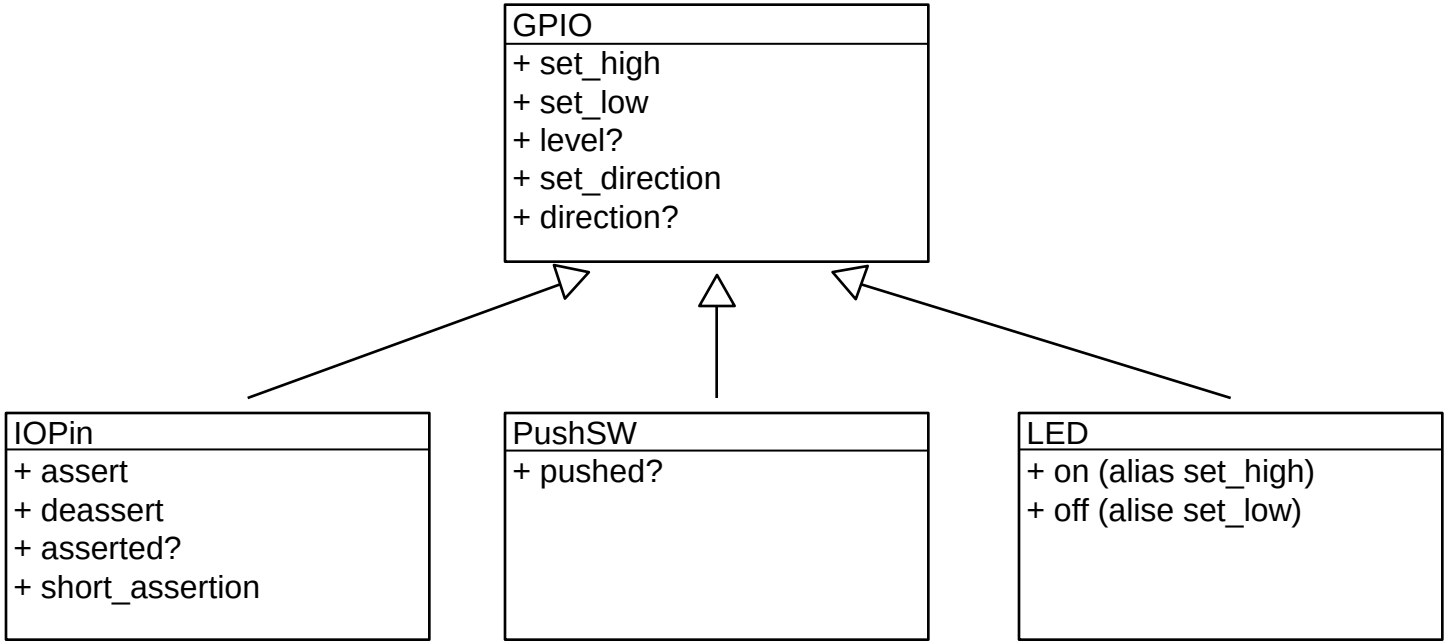


軽量Ruby評価ボード

③デバイス操作用モジュールの作成

情報と人,それを未来へ

- **MemIO（C言語拡張）モジュールを利用した周辺デバイス操作用モジュール**
  - GPIO … コントロールレジスタの抽象化（ポート）、I/O共通の初期化
    - IOPin … I/Oピン番号のGPIOポートのマッピングとアクセスメソッドの提供
    - PushSW … スイッチ番号とGPIOポート等のマッピングとメソッド提供
    - LED … 表示箇所とGPIOポートメソッドの提供



③デバイス操作用モジュールの作成

情報と人,それを未来へ

- GPIOクラス（一部）

```
class GPIO
  # レジスタ・アドレス・マップ
  PORT_REGADDR = {
    :ledr   => 0x0c1813f0,
    :pushsw => 0x0c181460,
    :pio1   => 0x0c1813d0,
       :
  }

  def initialize(port, bitpos, dir)
    @addr = PORT_REGADDR[port]
          :
  end

  def set_high
    puts "set high" if $VERBOSE
    val = MemIO.read(@addr, 1)
    val |= (0b1 << @bit)
    MemIO.write(@addr, val, 1)
  end

end
```

- IOPinクラス（一部）

```
class IOPin < GPIO
  # CN15 GPIO ピンのアドレス/ビッド位置対応表 #No.1～
  IOPIN_PORTBIT = [
      :
    {:port => :pio1,  :bit => 7},  # 3
    {:port => :dpio1, :bit => 7},  # 4
       :
  ]

  def initialize(pin, amode=XX, dir=XX)
    if pin <= 0 || pin > 40
      raise ArgumentError, "invalid argument"
    end
    super(IOPIN_PORTBIT[pin][:port],
IOPIN_PORTBIT[pin][:bit], dir)
    deassert if @amode == GPIO::ACTIVE_LOW
  end

  def assert
     if @amode == GPIO::ACTIVE_HIGH
      set_high
    else
      set_low
    end
  end

end
```

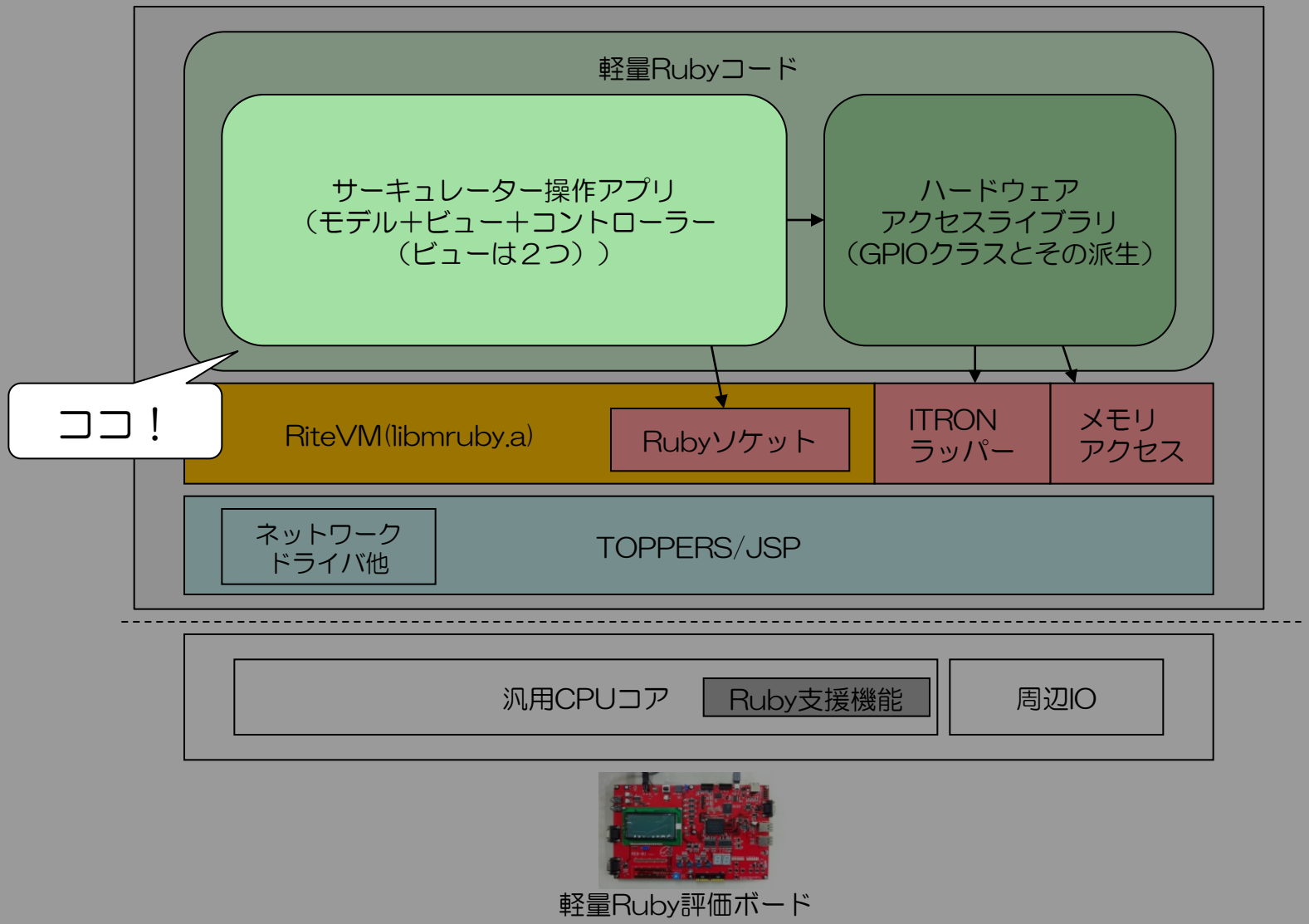
③デバイス操作用モジュールの作成

情報と人,それを未来へ

- **C言語とRubyコードの分担**
  - ②と③はボードベンダーなどから提供されることを想定
  - C言語とRubyの実装の分担は
    - クリティカルな速度を要求される場合 → C言語
    - UI、ネットワーク等クリティカルでないもの → Ruby

    ※今回作成したハードウェアアクセスライブラリでの
    GPIOクラス以下はUI相当であったため、
    Rubyでの実装とした。

④アプリ本体の作成



軽量Ruby評価ボード

④アプリ本体の作成

情報と人,それを未来へ

- **オブジェクト指向プログラミング**
  - もはや組込みでも上位のアプリケーションなどではJavaやC++による構造化プログラミング、オブジェクト指向プログラミングが一般化してきている
  - しかし今後ますます複雑化、高機能化するのにC++やJavaでは言語仕様が複雑

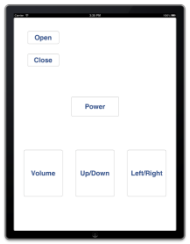
シンプルでかつメタプログラミングができる
Rubyの方が今後組込み開発の生産性を
高められると考えられる

④アプリ本体の作成

情報と人,それを未来へ

- **サーキュレーターをモデル化しMVCっぽく作成（circulator.rb）**
  - モデル：“CirculatorModel”
    電源、風量、首振り動作（上下、左右）、タイマー操作と
    各動作の状態保持を担当
    - 初期化：IOPinオブジェクトの生成
  - ビュー：“CircIOSView”、“CircButtonView”
    モデルから取得した動作状態のLEDへの表示と
    iOS（ソケット通信）またはボタンからの入力を担当
  - コントローラー：“CirculatoerController”
    ビューからの入力に対してモデルへの動作指示と
    全体の動作（ビューの切り替え等）の統括を担当

④アプリ本体の作成



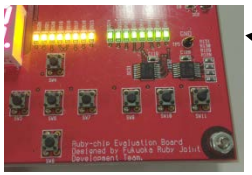
入力イベント・
障害通知

CirculatorController

表示指示

動作指示

ボタン



LED

CirclOSView

状態参照

CirculatorModel

ボタン

CircButtonView

障害発生時に
コントローラから
切り替えられる

④アプリ本体の作成

情報と人,それを未来へ

- CirculatorModel（一部）

```
class CirculatorModel
  attr_reader :power_status, :fan_vol_status

  def initialize
    @power_status = false
          :
    @power_pin = IOPin.new(19)
  end

  def power_toggle
    if @power_status == true
      @power_status = false
    else
      @power_status = true
    end
    @power_pin.short_assertion
  end

end
```

- CirculatorContloller（一部）

```
class CirculatorController
  def initialize
    @model = CirculatorModel.new
    @view = CircIOSView.new(self, :event, @model)
  end

  def start
    begin
      @view.start
    rescue => exc
      @view.stop
      if @view.instance_of?(CircIOSView)
        @view = CircButtonView.new(self, :event, @model)
      else
        @view = CircIOSView.new(self, :event, @model)
      end
      retry
    end
  end

end
```

⑤シミュレーション用ホスト環境の作成(おまけ)

情報と人,それを未来へ

- **現時点での課題(問題点、苦労…)**
  - 現状、デバッガがなく動作を追うのにprintデバッグしかない
  - ターゲット用にビルド＆ロードを繰り返すのも時間がかかる

  - デバッガの開発ももちろんのこと、
    ホスト環境下でのシミュレーションでの動作確認も期待される
  - できればスクリプトのままでも動作させたい

⑤シミュレーション用ホスト環境の作成(おまけ)

情報と人,それを未来へ

- **ホスト環境でのシミュレーションによる開発支援**
  - Rubyコードそのものはポータブル
  - 拡張クラスライブラリをどのように準備するか?
    → ホスト環境でのハードウェアを模擬する環境を用意
  - スクリプトのままで動作させるには?
    同ライブラリを組み込んだインタプリタ
    - mrubyコマンドに直接組み込む
    - 簡易インタプリタを用意する ← 今回はこちらで作成

    詳しくは弊社デモコーナーにて

まとめ

- **軽量RubyがGitHub上にてリリースされた**
  - 誰でも利用可能
- **組込み利用に関して**
  - デバッグ環境等、開発環境としてまだ不便なところが多い
  - しかし、プログラミング環境は非常に魅力的
    - 書いていて楽しい
    - 記述のし易さ、見通しの良さ、メタプログラミングが可能とする環境で今までにない方法で効率向上に寄与する可能性が高い
  - 適応分野は今後複雑性が増すところや製品出荷後等の動作時に機能が決定するようなところか?
- **今後の発展に期待**
  - まだコミュニティでの検討・修正が続いている
  - 未整備な部分もあるので、仕様に関与できる今がチャンス!!

お知らせ

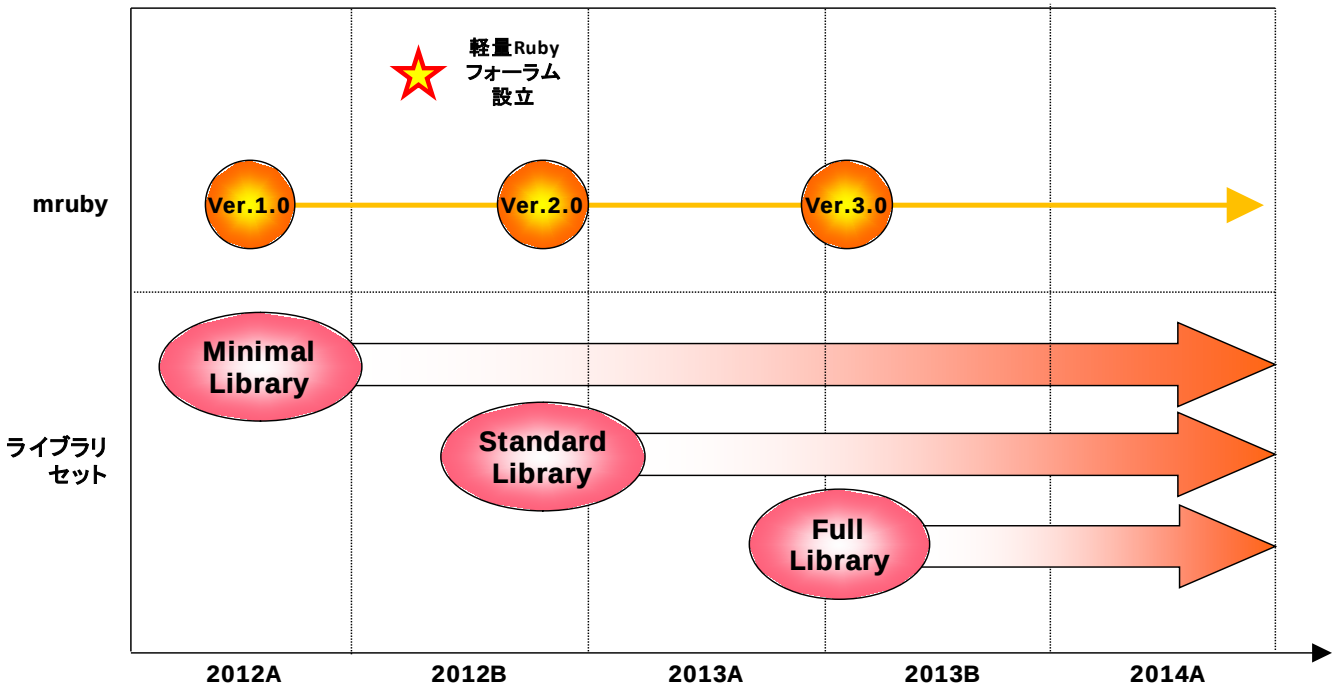
情報と人,それを未来へ

- **NPO法人「軽量Rubyフォーラム」設立**（※認定申請中）
  - 軽量Rubyに携わる企業や団体、個人が活用できるよう、改善を行っていくと同時に、その周辺で必要となる「ツール類の充実」、「利用者への教育」、「導入の指南」、「啓発や周知活動」を行う組織（弊社も発起人として参画している）

詳しくはWeb↓で!!

http://forum.mruby.org

軽量Rubyロードマップ



情報と人、それを未来へ

『人』は『情報』を上手に活用することで、

豊かな社会や生活を築くことができます。

当社はシステムでその実現のお手伝いをしています。

『人』と『情報』が新しい価値を生み出すよう、

それらを大切に育み『未来』に繋げてゆくこと、

これが当社の使命です。

東芝情報システム株式会社

情報と人、それを未来へ

TOSHIBA

Leading Innovation