



# RubyとMac OS Xの未来

The Future of Ruby and Mac OS X

**Laurent Sansonetti**

Ruby Ninja  
Apple Inc.

RubyKaigi 2008 - <http://jp.rubyist.net/RubyKaigi2008/english.html>

# Agenda

- Mac OS XにおけるRubyの未来 The Future of Ruby in Mac OS X
  - 現状 Today's Situation
  - 未来を考える Thoughts for the Future
  - Q&A

# 過去から現在までの歩み

# 過去から現在までの歩み

- 10.2 “Jaguar”
  - Ruby 1.6.7
- 10.3 “Panther”
  - Ruby 1.6.8
- 10.4 “Tiger”
  - Ruby 1.8.2
- 10.5 “Leopard”
  - Ruby 1.8.6

# 過去から現在までの歩み

- 10.2 “Jaguar”
  - Ruby 1.6.7
- 10.3 “Panther”
  - Ruby 1.6.8
- 10.4 “Tiger”
  - Ruby 1.8.2
- 10.5 “Leopard”
  - Ruby 1.8.6 + new stuff!

# Ruby in Leopard: What's New?

- 配布方法が改良された Better distribution
  - Ruby.framework
  - RubyGems + many gems
  - DTrace support
- Ruby による Mac OS X 開発を公式にサポート

Mac OS X development in Ruby now officially supported

# Mac OS X Development Stack

# Mac OS X Development Stack

User Experience

Application Frameworks

Graphics and Media

Darwin



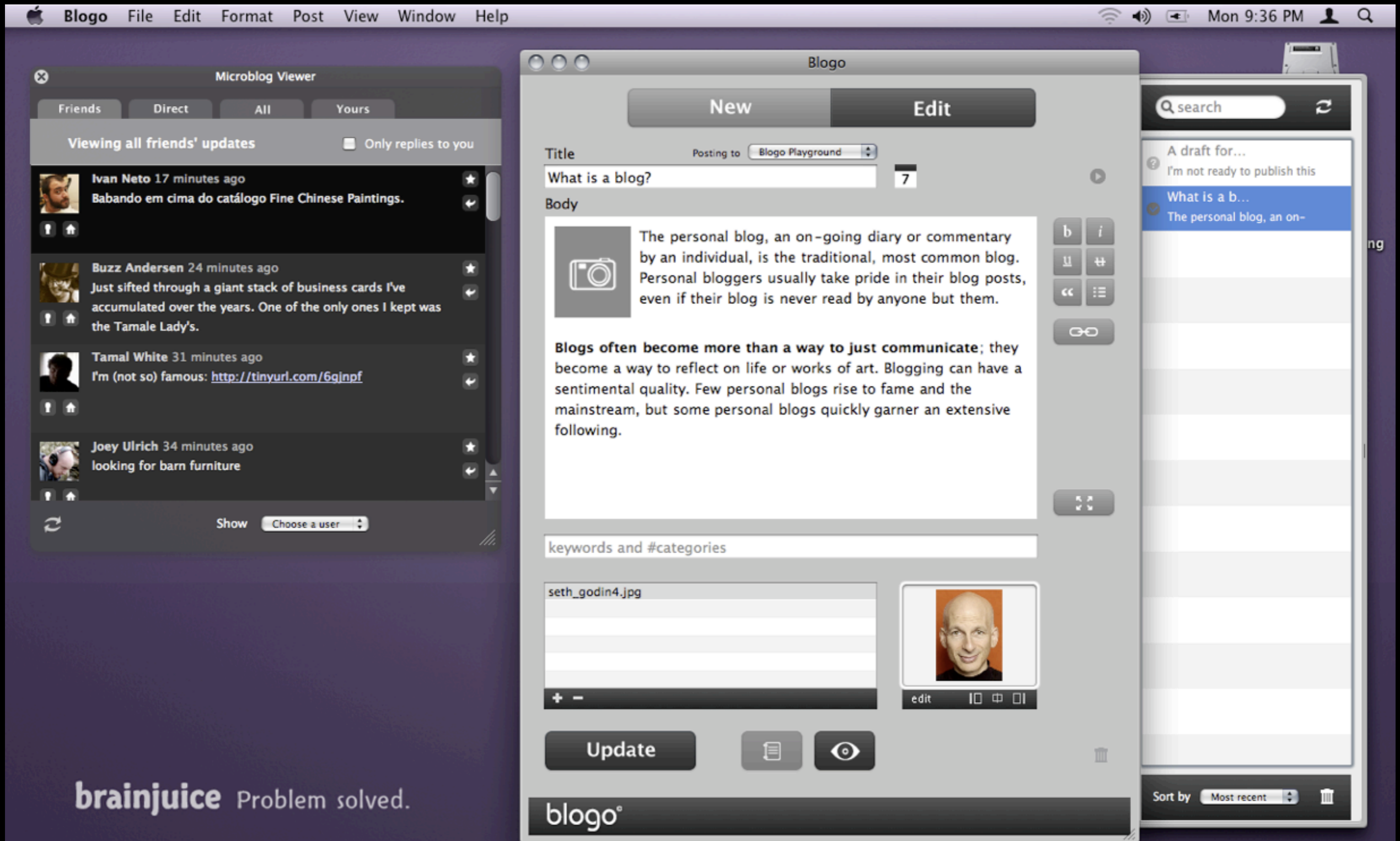
# RubyCocoa

- Ruby/Objective-C bridge
  - Cocoaだけでなく Not Cocoa specific
  - 素のC APIとのブリッジとして機能する Can bridge pure C APIs too!
- Dual LGPL/Ruby licensed
- 2001年に藤本尚邦が開発 Created in 2001 by FUJIMOTO Hisakuni
  - 現在は私(Laurent)がメンテナンス Now maintained by me
- Status: stable
  - 0.13.2が最新バージョン 0.13.2 is the latest version
- **フリーや商用のプロジェクトで使用されている**  
Used by free and commercial projects

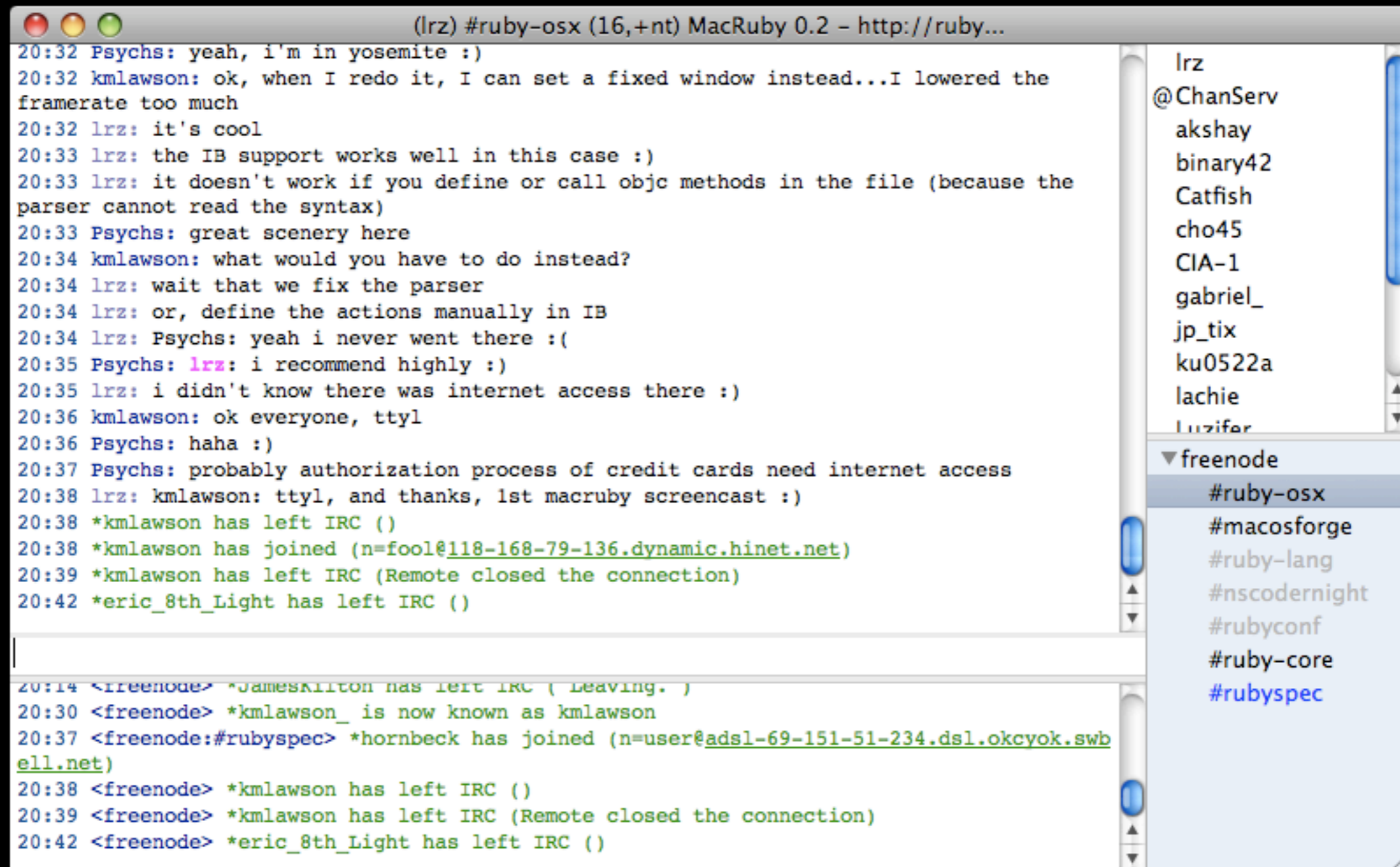
# RubyCocoa Use Cases

- アプリケーションのプロトタイピング Application prototyping
  - 手早く楽しく Ruby でコーディング  
Writing Ruby code is fast (and fun!)
- アプリケーションのデバッグ Application debugging
  - Ruby はインタプリタでダイナミック  
Ruby is interpreted and dynamic!
- アプリケーション開発 Application development
  - 面倒な仕事のほとんどは(CやObjective-Cで実装された)既存のフレームワークがやってくれる  
Most of the hard work is done in frameworks

# Blogo: Weblog Editor



# LimeChat: IRC client



The screenshot shows the LimeChat IRC client interface. The main chat window displays a conversation between users Psychs, kmlawson, and lrz. The chat history includes messages about Yosemite, fixed windows, and parser issues. A user list on the right side shows the current channel, #ruby-osx, and other channels like #macosforge, #ruby-lang, #nscodernight, #rubyconf, #ruby-core, and #rubyspec. The interface also shows system messages about users joining and leaving the channel.

```
(lrz) #ruby-osx (16,+nt) MacRuby 0.2 - http://ruby...
20:32 Psychs: yeah, i'm in yosemite :)
20:32 kmlawson: ok, when I redo it, I can set a fixed window instead...I lowered the
framerate too much
20:32 lrz: it's cool
20:33 lrz: the IB support works well in this case :)
20:33 lrz: it doesn't work if you define or call objc methods in the file (because the
parser cannot read the syntax)
20:33 Psychs: great scenery here
20:34 kmlawson: what would you have to do instead?
20:34 lrz: wait that we fix the parser
20:34 lrz: or, define the actions manually in IB
20:34 lrz: Psychs: yeah i never went there :(
20:35 Psychs: lrz: i recommend highly :)
20:35 lrz: i didn't know there was internet access there :)
20:36 kmlawson: ok everyone, ttyl
20:36 Psychs: haha :)
20:37 Psychs: probably authorization process of credit cards need internet access
20:38 lrz: kmlawson: ttyl, and thanks, 1st macruby screencast :)
20:38 *kmlawson has left IRC ()
20:38 *kmlawson has joined (n=fool@118-168-79-136.dynamic.hinet.net)
20:39 *kmlawson has left IRC (Remote closed the connection)
20:42 *eric_8th_Light has left IRC ()

20:14 <freenode> *JamesKilton has left IRC ( Leaving. )
20:30 <freenode> *kmlawson_ is now known as kmlawson
20:37 <freenode:#rubyspec> *hornbeck has joined (n=user@adsl-69-151-51-234.dsl.okcyok.swb
ell.net)
20:38 <freenode> *kmlawson has left IRC ()
20:39 <freenode> *kmlawson has left IRC (Remote closed the connection)
20:42 <freenode> *eric_8th_Light has left IRC ()
```

# Demo: RubyCocoa

RubyCocoaはどのように動くのか？

# RubyCocoaはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```

```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```

# RubyCocoaはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```

```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```

NSButton



# RubyCocoaはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```

```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```

OSX::NSButton

NSButton

# RubyCocoaはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```



OSX::NSButton

```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```

NSButton

# RubyCocoaはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```

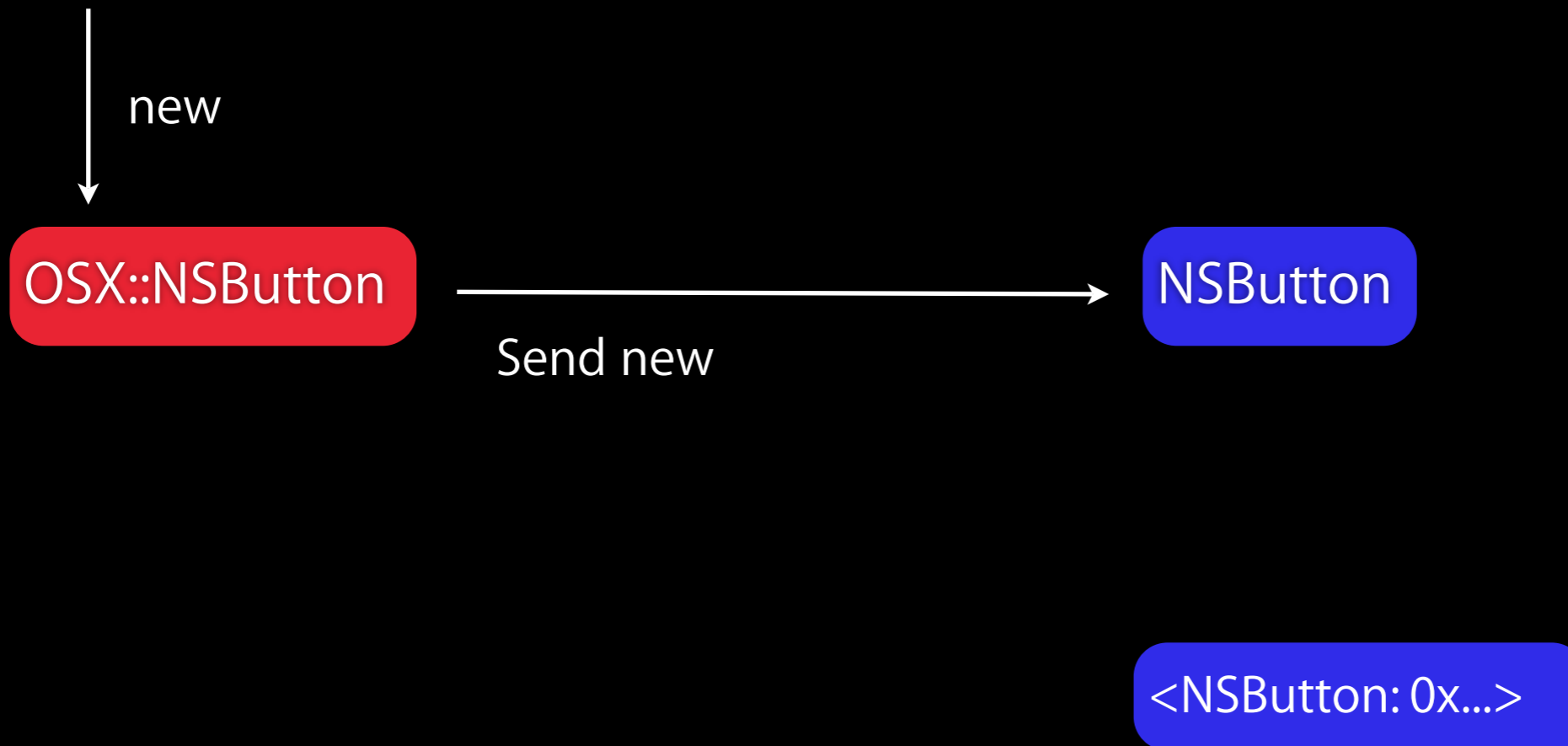
```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```



# RubyCocoaはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```

```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```



# RubyCocoaはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```

```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```

↓  
new

OSX::NSButton

→  
Send new

NSButton

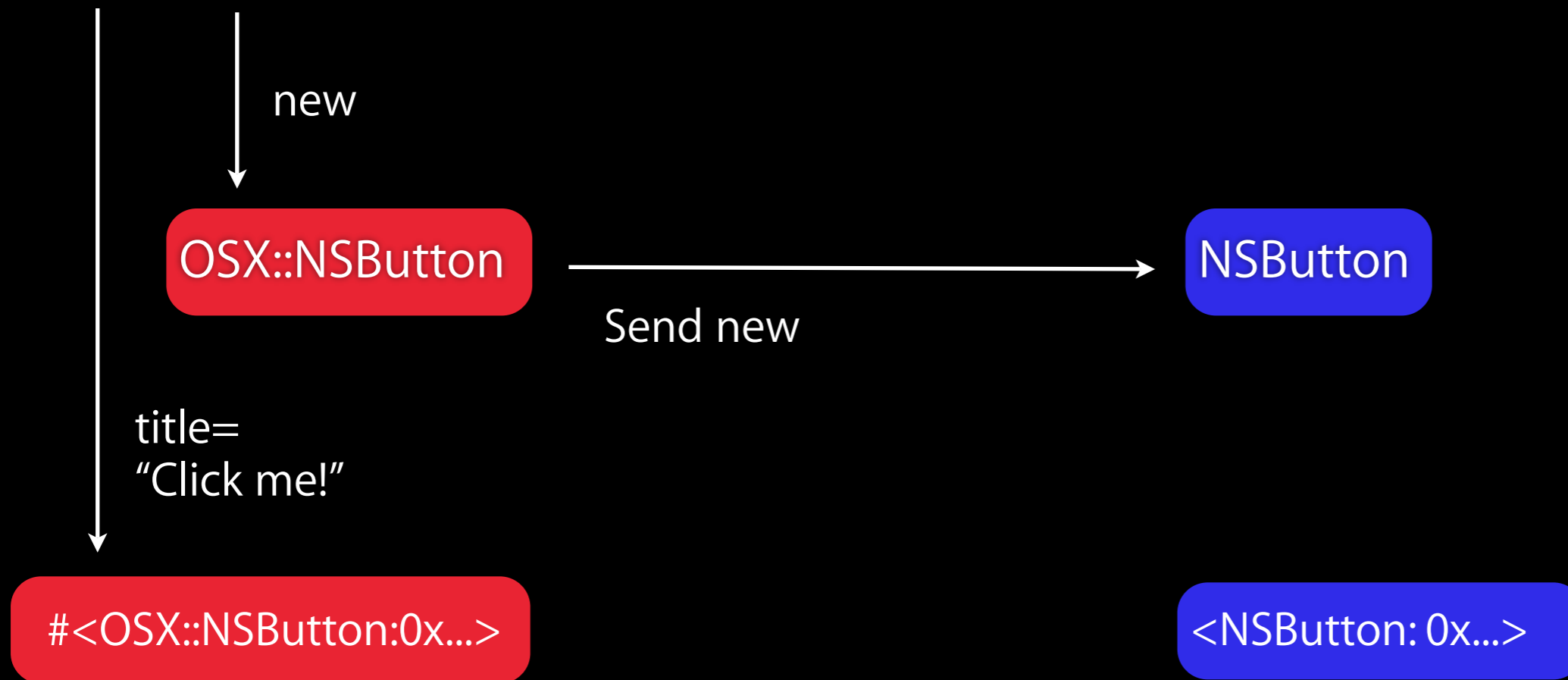
#<OSX::NSButton:0x...>

<NSButton: 0x...>

# RubyCocoaはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```

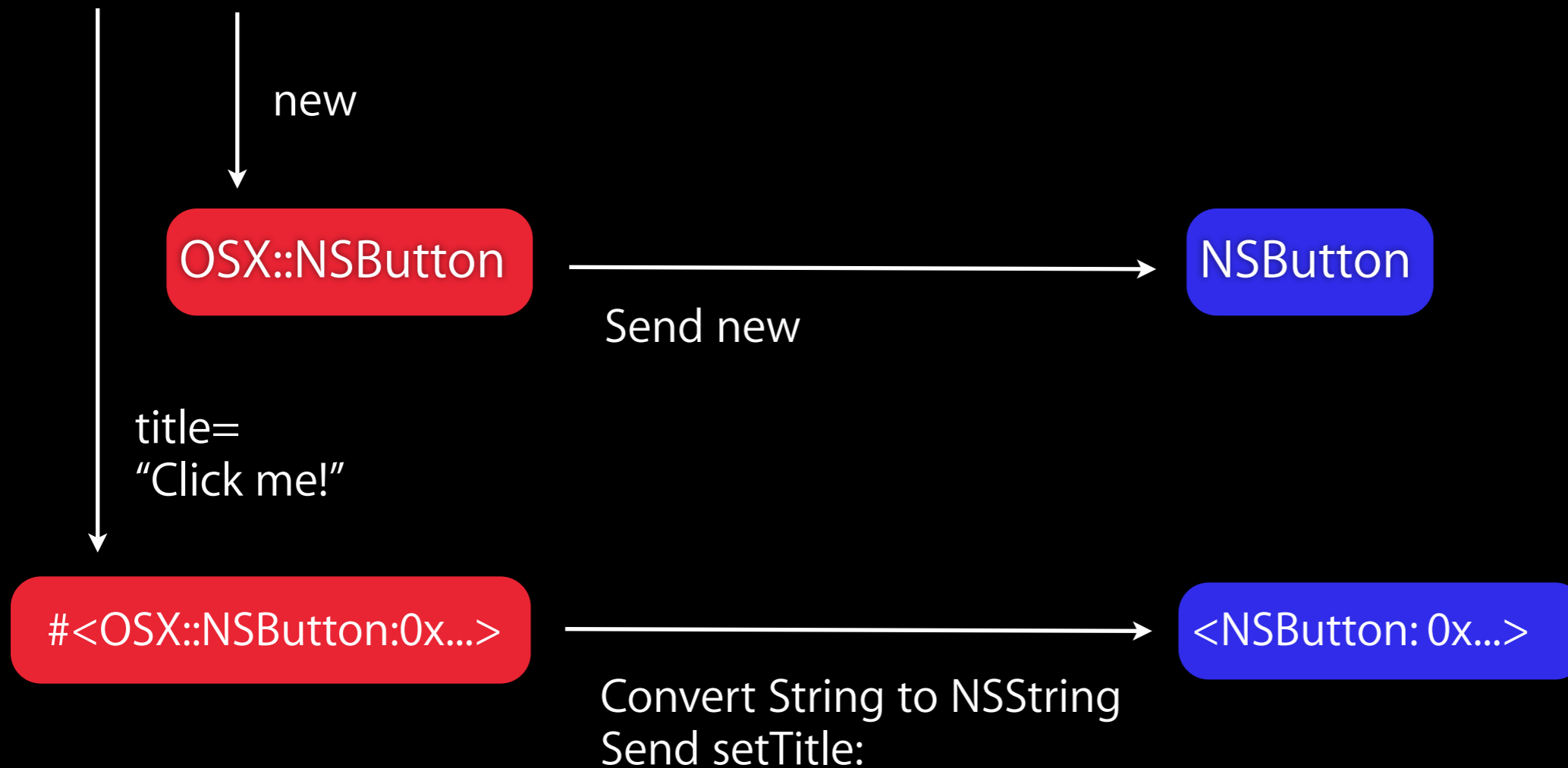
```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```



# RubyCocoaはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```

```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```



# RubyCocoaはどのように動くのか？

- RubyCocoaは基本的に Basically, RubyCocoa
  - Proxyを作る Creates proxies
  - メッセージを送る Forwards messages
  - 型を変換する Converts types
  - 例外を変換する Converts exceptions



では、RubyCocoaは何が問題なのか？

# では、RubyCocoaは何が問題なのか？

- RubyCocoaは基本的に Basically, RubyCocoa
  - Proxyを作る Creates proxies
  - メッセージを送る Forwards messages
  - 型を変換する Converts types
  - 例外を変換する Converts exceptions

# RubyCocoaの問題点

# RubyCocoaの問題点

- ブリッジ Bridging
- 構文 Syntax
- スレッド Threading
- メモリ管理 Memory

# ブリッジの問題

- (RubyとObjective-Cの)両方のランタイムで、クラスとオブジェクトモデルの管理が必要

Class and object model must be maintained in both runtimes.

- オブジェクトや例外がブリッジを通るときに変換が必要

Objects and exceptions must be converted when they cross the bridge.

- 無駄な複製を避けるために、インスタンスのキャッシュが必要

Instances must be cached to avoid unnecessary duplication.

- ▶ メモリの浪費と遅いディスパッチ

Memory overhead and slow dispatch.

# 構文の問題

- Objective-Cでは、引数の名前がメソッド名に含まれる

In Objective-C, the name of arguments are part of the method name.

```
[obj doSomething];           # method is doSomething
[obj doSomethingWith:42];    # method is doSomethingWith:
[obj doSomethingWith:42 and:42]; # method is doSomethingWith:and:
```

- Ruby(1.8+RubyCocoa)では、ブリッジがメッセージセレクトの ":" を "\_" に置き換える

Ruby (1.8 + RubyCocoa) doesn't, so the bridging convention is to replace ':' with '\_'

```
obj.doSomething
obj.doSomethingWith(42)
obj.doSomethingWith_and(42, 42)
```

- ▶ メッセージ送信の構文がRubyとObjective-Cではぜんぜん違う Syntax is not very Ruby friendly.

# スレッドの問題

- Rubyはスレッドセーフではないので、Objective-Cのスレッドコードを使うことができない。

Because Ruby is not thread-safe, it is impossible to use Objective-C threaded code.

- Rubyスレッドはネイティブスレッドではないので、スレッドごとにObjective-Cデータは保存/復元される。

Because Ruby threads are not native, per-thread Objective-C data has to be saved/restored.

- ▶ RubyCocoaでスレッドを使うと、とても遅く不安定になる。

Use of threading in RubyCocoa is very slow and unstable.

# メモリの問題

- RubyのGCは、ゴミ集めの間、メインスレッドを止めてしまう

Ruby's GC stops the main thread while collecting memory.

- Objective-C 2.0 の GC を使って書かれてアプリケーションで Ruby を使うことができない

Cannot integrate Ruby in an application written with Objective-C garbage collection.



# MacRuby

- Objective-Cランタイム上のRuby

Ruby on top of the Objective-C common runtime

- Keyed/named メソッド引数構文

Keyed/named method arguments syntax

- Ruby 1.9をベースに実装

Implements the 1.9 language

- Rubyライセンス

Covered by the Ruby license

- 2008年初期、Appleによって開発

Created in early 2008 by Apple

# MacRuby が目指すこと

# MacRuby が目指すこと

“Rubyを使う楽しみのためにパフォーマンスを犠牲にすることなく、成熟したMac OS Xアプリケーションの作成を可能にする”

*“enable the creation of full-fledged Mac OS X applications which do not sacrifice performance in order to enjoy the benefits of using Ruby.”*

# Ruby on top of Objective-C

- All Ruby classes depend on NSObject.
- All Ruby objects are Objective-C objects.
- All Ruby methods are Objective-C methods.
- Ruby builtin types (String, Array and Hash) are Objective-C types (NSString, NSArray and NSDictionary).
- Using the Objective-C garbage collector.
- 高速なObjective-Cディスパッチ Very fast Objective-C dispatch.
- ▶• Objective-Cの世代別GCは、スレッドごとに高速に実行される Objective-C GC runs in separate thread and performs fast generational collections.

# Keyed/Named Arguments

- MacRubyでは、Objective-Cメソッドを呼ぶための新しい構文が導入された

MacRuby introduces a new syntax to call Objective-C methods.

```
NSWindow *window = [[NSWindow alloc] initWithContentRect:frame  
                    styleMask:NSBorderlessWindowMask  
                    backing:NSBackingStoreBuffered  
                    defer:false];
```

```
window = NSWindow.alloc.initWithContentRect frame,  
         styleMask:NSBorderlessWindowMask,  
         backing:NSBackingStoreBuffered,  
         defer:false
```

# Keyed/Named Arguments

- そして、Objective-Cメソッドの定義の構文

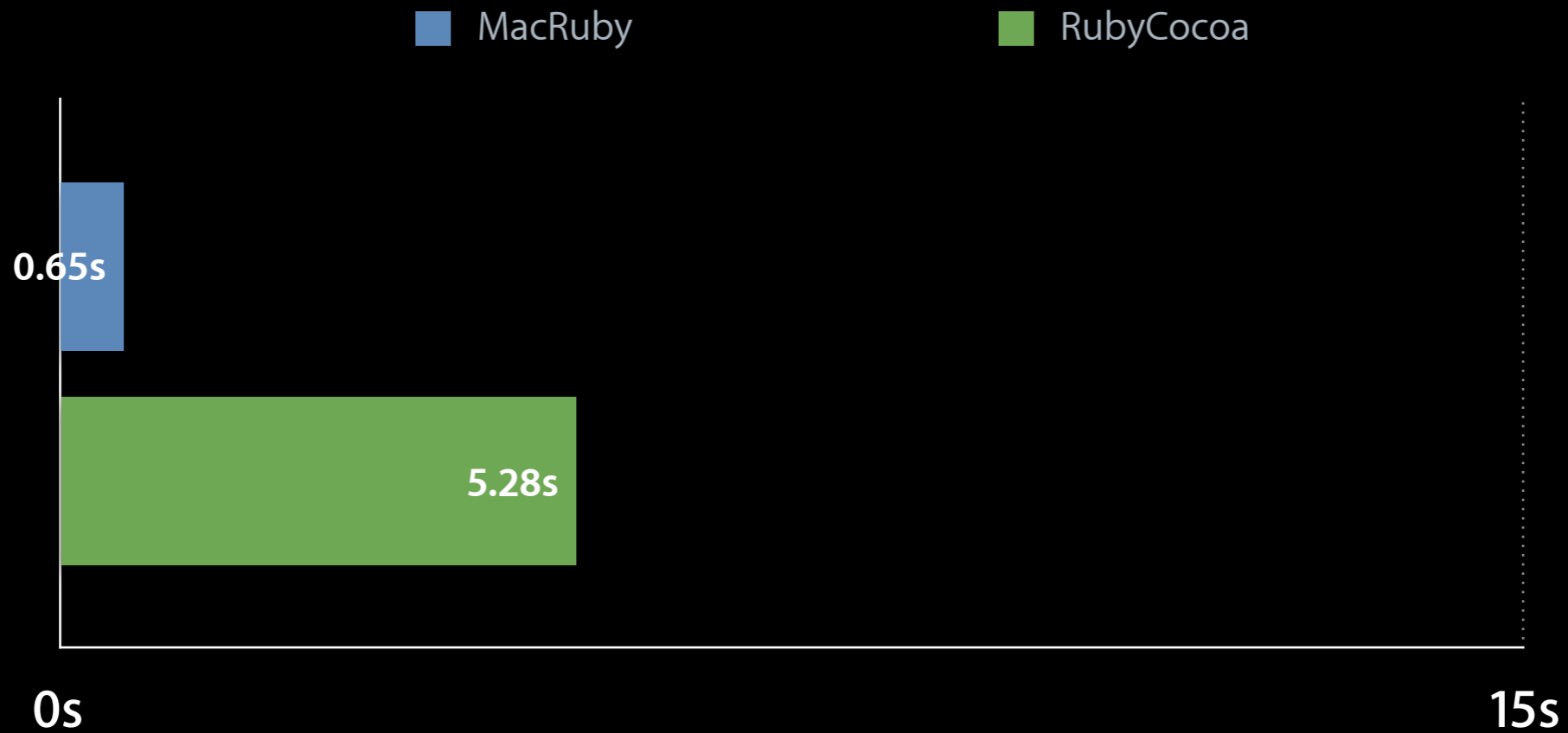
And also to define Objective-C methods.

```
- (id)tableView:(NSTableView *)aTableView
    objectValueForTableColumn:(NSTableColumn *)aTableColumn
    row:(NSInteger)rowIndex
{
    // ...
}
```

```
def tableView view, objectValueForTableColumn:column, row:index
    # ...
end
# New method name: tableView:objectValueForTableColumn:row:
```

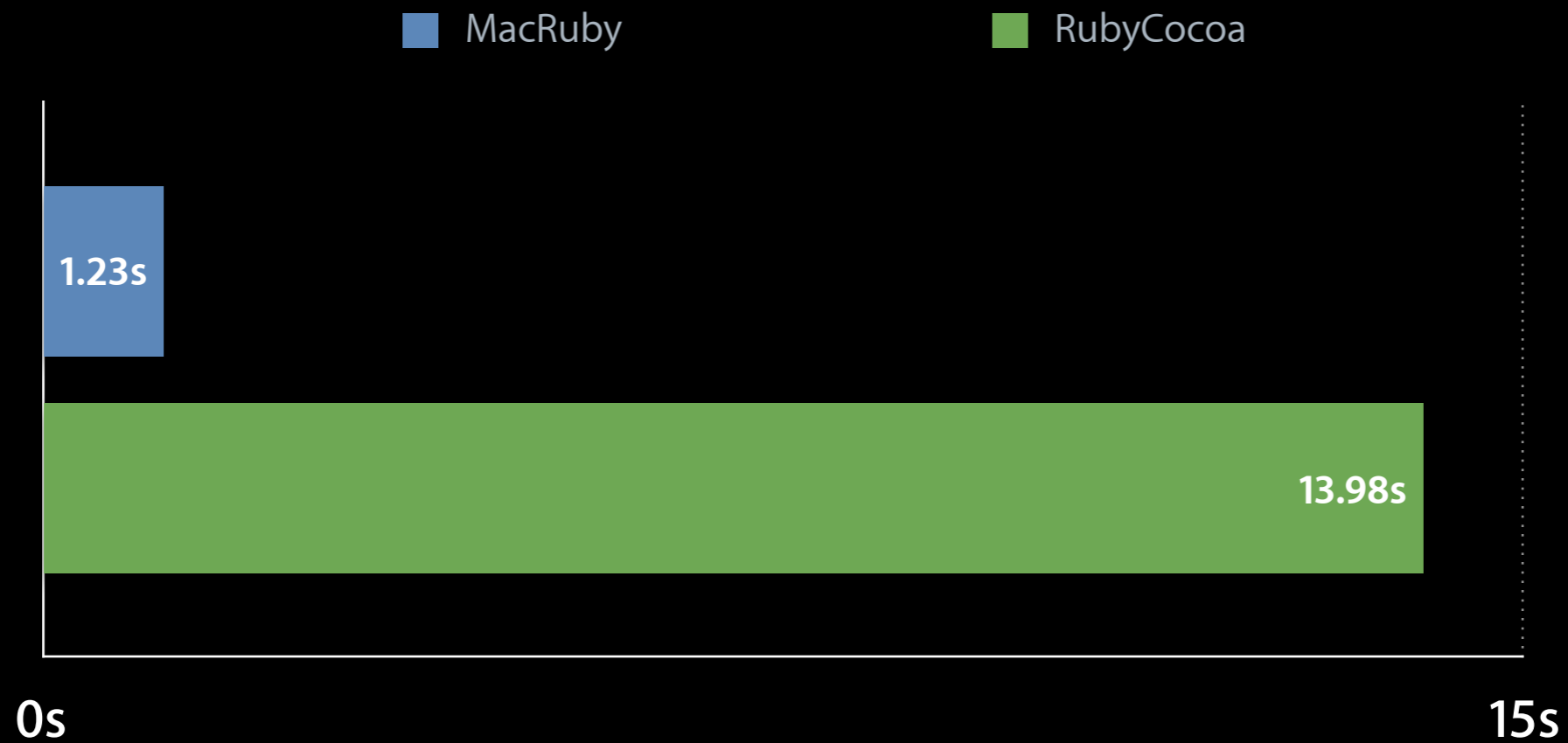
# MacRuby vs RubyCocoa

```
o = Dummy.new  
1_000_000.times { o.doSomething }
```



# MacRuby vs RubyCocoa

```
o = Dummy.new  
s = 'foo'  
1_000_000.times { o.doSomethingWith(s) }
```

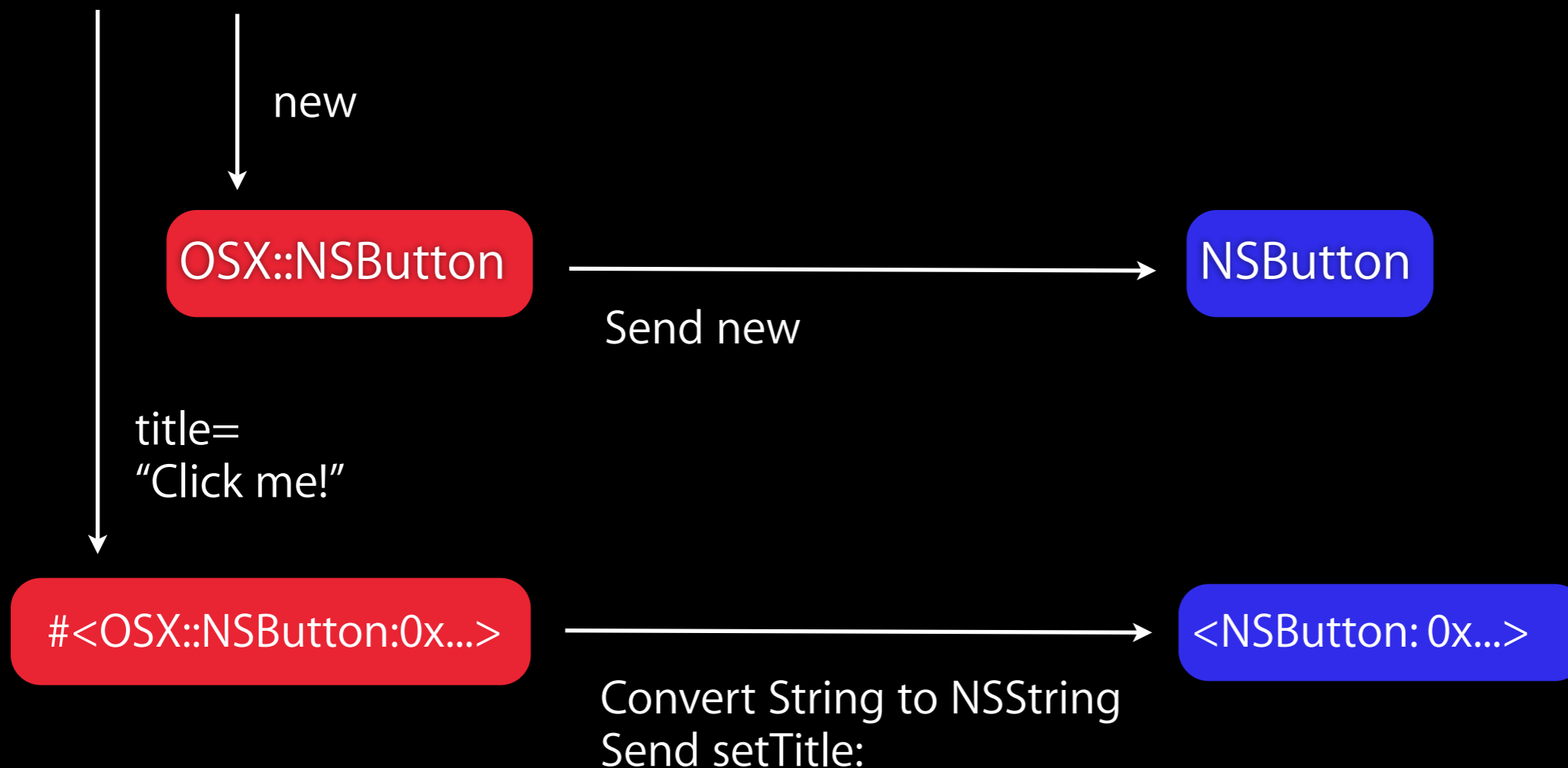




# RubyCocoaはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```

```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```



# MacRubyはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```

```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```

# MacRubyはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```

```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```

NSButton

# MacRubyはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```

```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```

new



NSButton

# MacRubyはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```

```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```

new



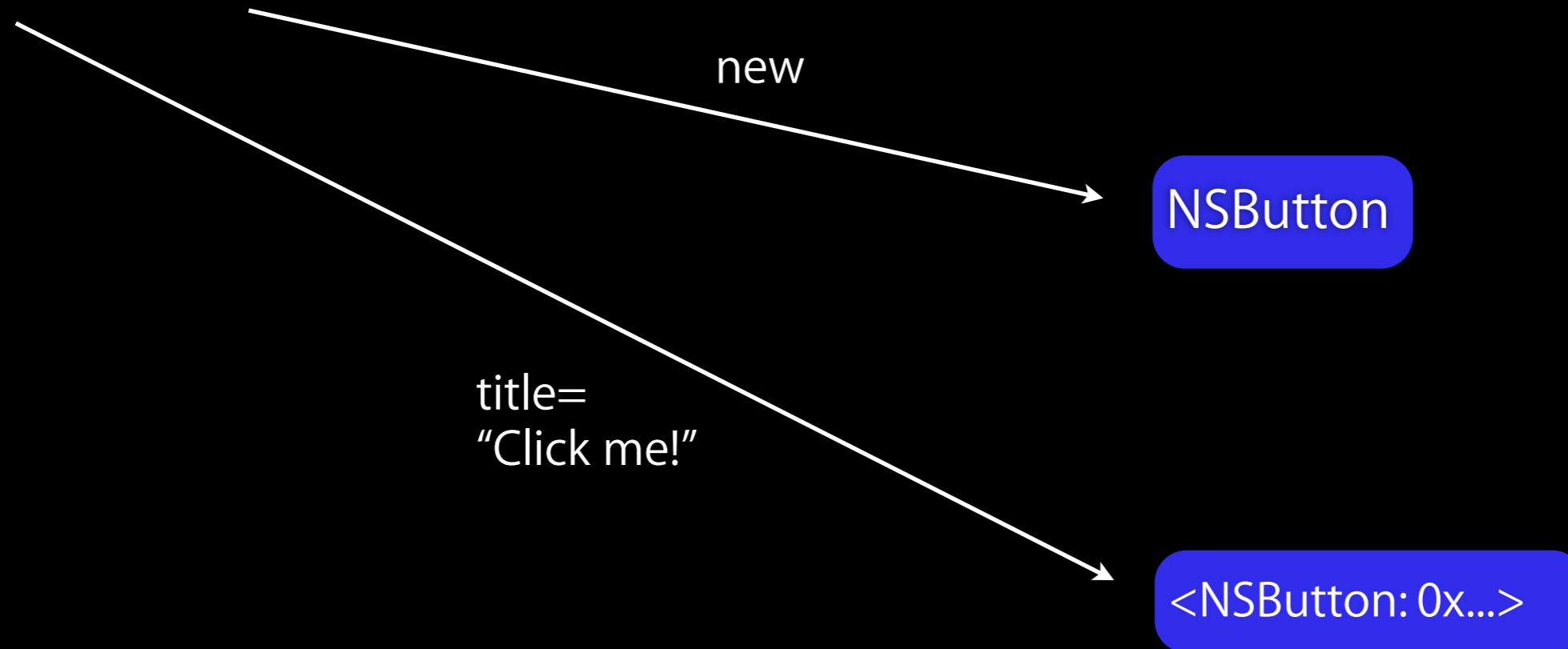
NSButton

<NSButton: 0x...>

# MacRubyはどのように動くのか？

```
button = NSButton.new  
button.title = "Click me!"
```

```
button = [NSButton new]  
[button setTitle:@"Click me!"]
```



# Demo: MacRuby

# State of MacRuby

- Current release is 0.2
- Objective-CとGCの統合を完了 Objective-C and GC integration done
- CoreFoundationベースの実装を完了 CoreFoundation-based implementation done
- Xcode support, samples
- Alpha/Experimental
  - もはやSEGVはほとんど発生しない doesn't SEGV a lot anymore :-)
  - `sample/test.rb` passes
  - most of bootstrap test and `test/ruby/test_*.rb` pass
  - `erb`, `irb`, `ri`, `rdoc` work



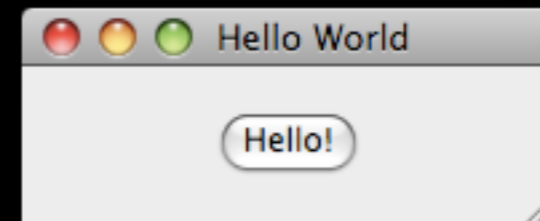
# MacRuby 0.3 Roadmap

- RubyCocoa compatibility (optional)
  - Help people port RubyCocoa applications to MacRuby
- HotCocoa.rb: Rubyish access to Cocoa!
- All `test/**/test_*.rb` will pass
- Make RubyGems and Rails work
- Xcode, IB, Instruments.app (DTrace) support
- Performance work!
- **First production release**
- **Planned for the end of the year (RubyConf?)**

# HotCocoa.rb

- Problems with Cocoa APIs
  - Too verbose
  - Do not use Ruby semantics (blocks, etc...)
- **HotCocoa.rb**: Ruby layer on top of Cocoa
  - Pure Ruby API
  - Based on Cocoa classes
  - Can be used to write from simple to complex GUI
- Project started last night (06/21) with help from Rich Kilmer!

# Hello World with Cocoa



# Hello World with Cocoa

```
framework 'Cocoa'

app = NSApplication.sharedApplication

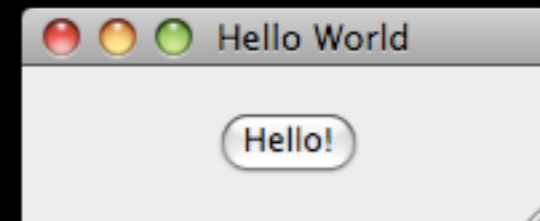
win = NSWindow.alloc.initWithContentRect([0, 0, 200, 60],
    styleMask:NSTitledWindowMask|NSClosableWindowMask|NSMiniaturizableWindowMask|NSResizableWindowMask,
    backing:NSBackingStoreBuffered,
    defer:false)
win.title = 'Hello World'

button = NSButton.alloc.initWithFrame(NSZeroRect)
win.contentView.addSubview(button)
button.bezelStyle = NSRoundedBezelStyle
button.title = 'Hello!'
button.sizeToFit
button.frameOrigin = NSMakePoint((win.contentView.frameSize.width / 2.0) - (button.frameSize.width / 2.0),
    (win.contentView.frameSize.height / 2.0) - (button.frameSize.height / 2.0))

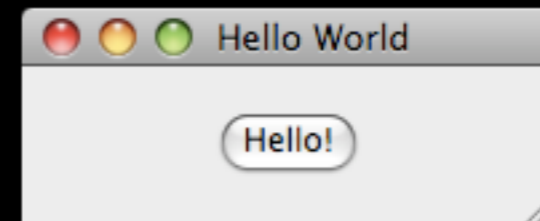
button_controller = Object.new
def button_controller.sayHello(sender)
    puts "Hello World!"
end
button.target = button_controller
button.action = 'sayHello:'

win.display
win.orderFrontRegardless

app.run
```

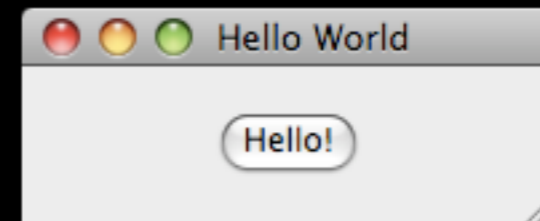


# Hello World with HotCocoa.rb



# Hello World with HotCocoa.rb

```
require 'hotcocoa'  
include HotCocoa  
  
app do  
  window :title => 'Hello World', :frame => [0, 0, 120, 120] do |w|  
    button :title => 'Click me' do |b|  
      b.on_action { puts 'Hello World!' }  
      w << b  
    end  
  end  
end
```



# After MacRuby 0.3

- Open Scripting Architecture conformance
- AppleEvent API
  - Make MacRuby a *true* replacement for AppleScript
- More performance work
  - Using LLVM to improve YARV and generate closures

# 最後に

- AppleはRubyによるMac OS X開発をサポート推奨する  
Apple supports and encourages Mac OS X development using Ruby
- For production, use RubyCocoa
- Please try MacRuby too, and report feedback!



# More Information

**RubyCocoa**

<http://rubycocoa.sourceforge.net>

**MacRuby**

<http://macruby.org>

# Q&A

**Laurent Sansonetti**

[lsansonetti@apple.com](mailto:lsansonetti@apple.com)

RubyKaigi 2008 - <http://jp.rubyist.net/RubyKaigi2008/english.html>