Ruby bindings 2016

How to create bindings 2016

Kouhei Sutou

ClearCode Inc.

RubyKaigi 2016
2016-09-09

Acknowledgment
感謝

@SoManyHs and @drbrain

They fixed English
in this slide
@RubyKaigi 2016 Official Party

昨日のパーティーで英語をチェックしてくれた！ありがとう！

Silver Sponsors

ClearCode Inc.

https://www.clear-code.com/

Silver sponsor

Free software is important in ClearCode. We develop/support software with our free software development experiences. We feed back our business experiences to free software.

Goal
目標

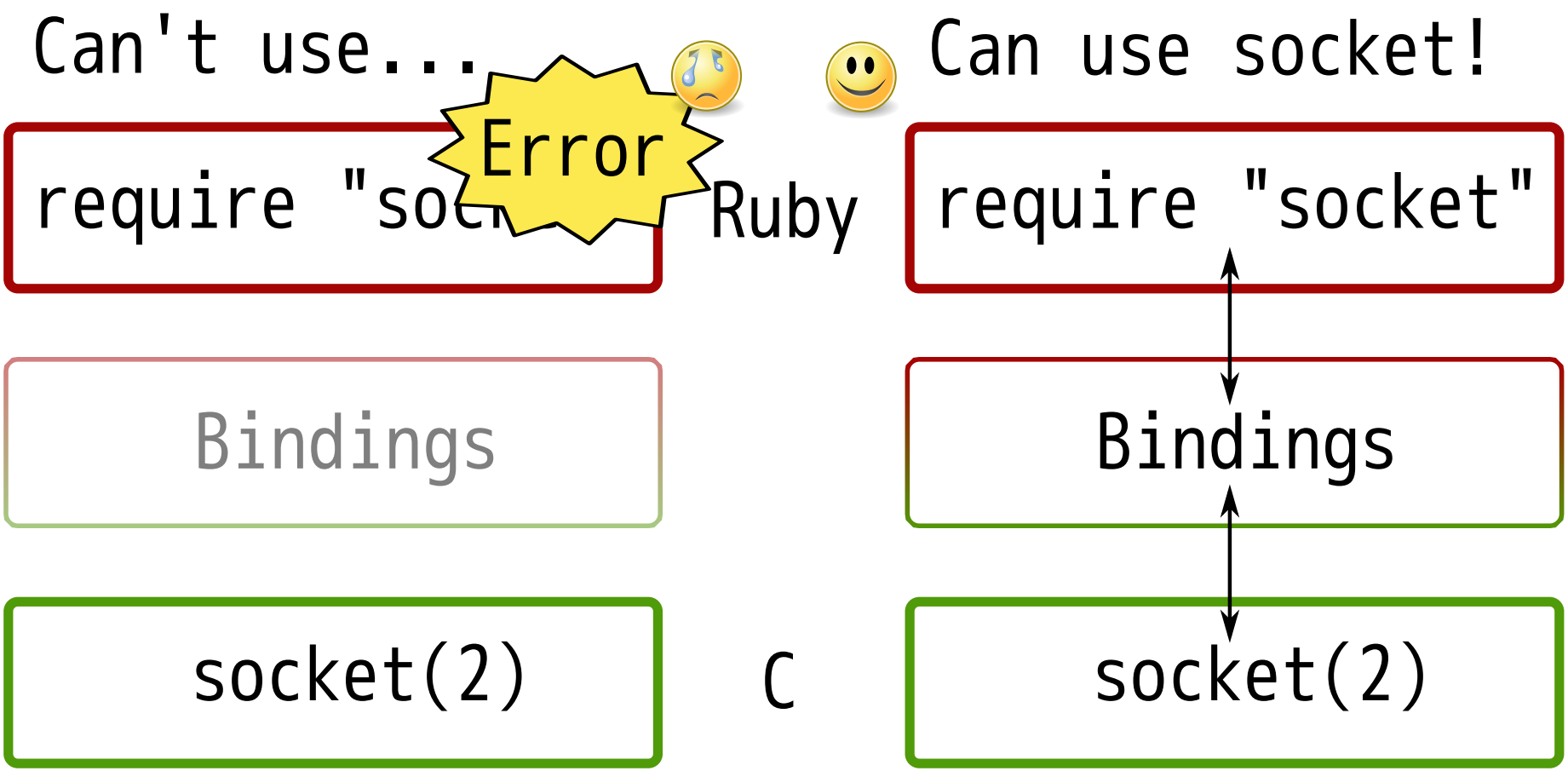
You become
a new bindings developer

あなたが新しくバインディング開発者になること

Bindings?
バインディングとは

- Glue of (mainly) C and Ruby
  主にCとRubyをつなぐもの
  - You can use features implemented in C from Ruby
    Cで実装された機能をRubyから使える
  - e.g.: Socket, OpenSSL, YAML, ...
    例：ソケット、OpenSSL、YAML…

Bindings
バインディング



Why should I learn?
なんでバインディング開発者になるの？

- To use Ruby in more cases
  Rubyをもっといろんな場面で使うため
  - e.g.: Machine leaning, multimedia, full text search, cipher and so on
    例：機械学習、画像・動画・音声処理、全文検索、暗号
- Can use existing features in Ruby through bindings
  バインディングがあると既存のいい機能をRubyで使える

Incr. bindings developer
バインディング開発者が増えるといいな

- Why do I become a bindings developer?
  バインディング開発者になりませんか？
  - To expand use cases of Ruby!
    Rubyを使えるケースを増やすために！
  - Not just a user for provided features
    提供された機能を使う1ユーザーではなく

Summary
概要

	Ext★	SWIG	FFI	GI☆
Base Tech	Ext	Ext	libffi	libffi
Impl. by	Hand	Gene-rate	Hand	Gene-rate

★ Extension library（拡張ライブラリー）
☆ GObject Introspection: Recommended（オススメ）

Demo
デモ

Make powerful auto generated bindings

自動生成されたバインディングがいかに強力か

What is ext?

拡張ライブラリーってなに？

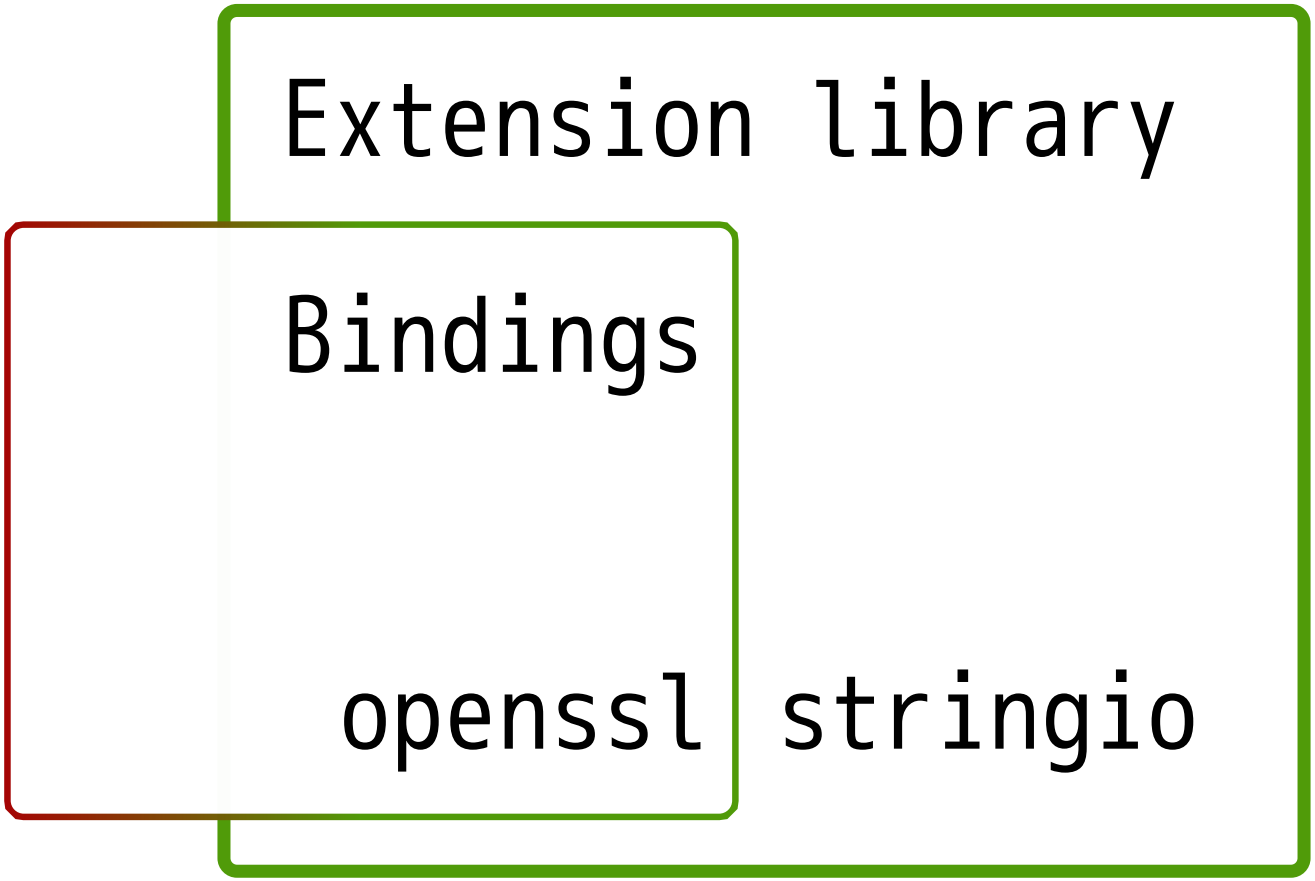
	Ext★	SWIG	FFI	GI☆
Base Tech	**Ext**	**Ext**	lib ffi	lib ffi
Impl. by	Hand	Gene-rate	Hand	Gene-rate

★ Extension library（拡張ライブラリー）
☆ GObject Introspection: Recommended（オススメ）

Extension library
拡張ライブラリーとは

- ✓ Ruby library written in C
  Cで書かれたRuby用ライブラリー
- ✓ Most bindings are ext lib
  多くのバインディングは拡張ライブラリーとして実装
  - ✓ openssl is ext lib and bindings
    oepnsslは拡張ライブラリーでバインディング
  - ✓ Exc.: stringio is ext lib but...
    例外：stringioは拡張ライブラリーだが…

Ext lib and bindings
拡張ライブラリーとバインディング



What is libffi?

libffiってなに？

	Ext★	SWIG	FFI	GI☆
Base Tech	Ext	Ext	**libffi**	**libffi**
Impl. by	Hand	Gene-rate	Hand	Gene-rate

★ Extension library（拡張ライブラリー）
☆ GObject Introspection: Recommended（オススメ）

libffi and FFI: 1

libffiとFFIとは：1

- libffi: Library to impl. FFI
  FFIを実現するためのライブラリー
- Foreign Function Interface
  - Generally:
    All APIs to impl. bindings
    一般的にはバインディングを実装するためのすべてのAPI
  - Ruby doesn't use "FFI" term
    Rubyでは「FFI」という単語を使わない

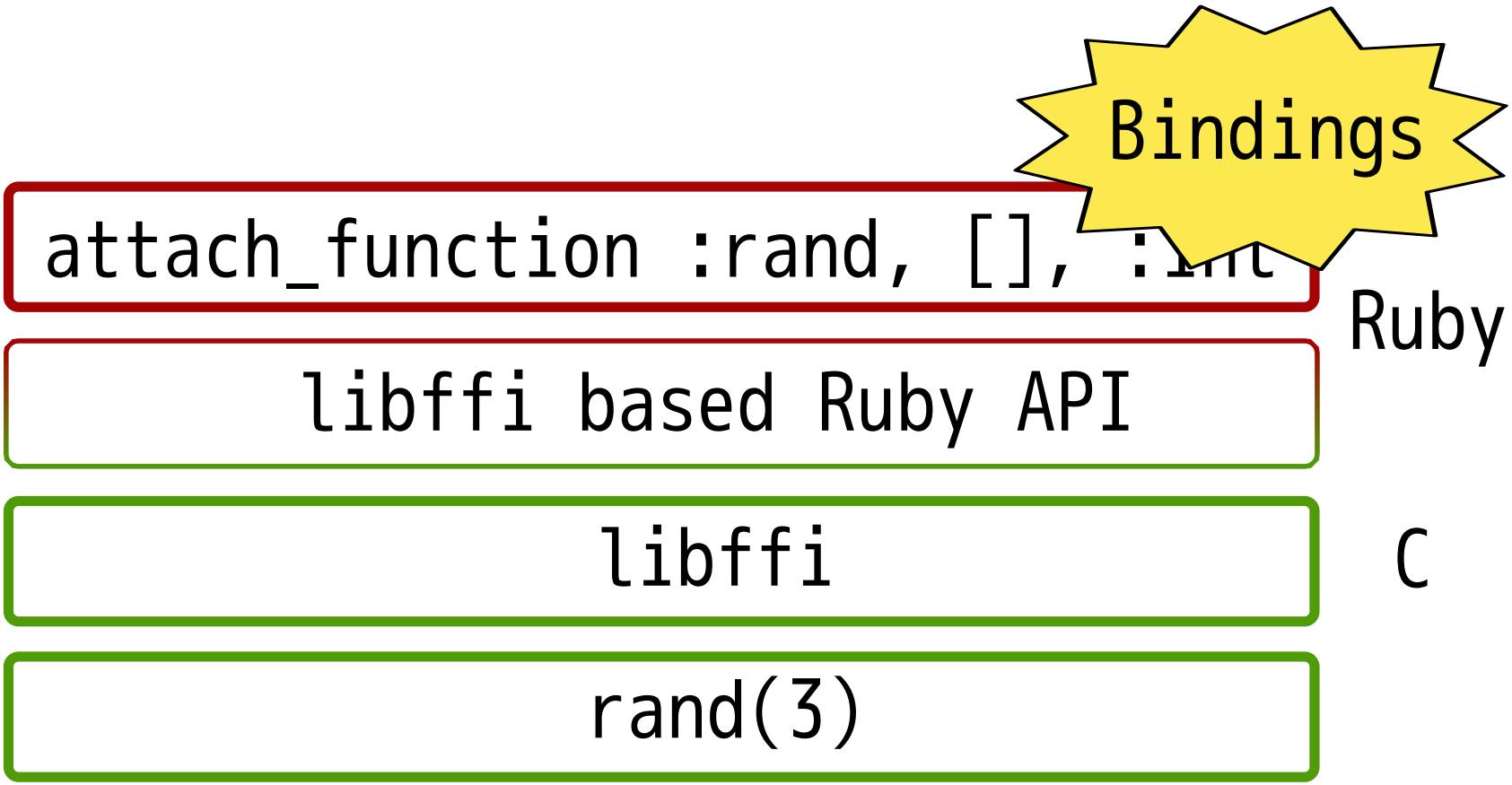
libffi and FFI: 2

libffiとFFIとは：2

- ✓ Ruby FFI
  - ✓ Library based on libffi
    libffiを使ったライブラリー
  - ✓ Provides **Ruby** API
    to implement bindings
    バインディングを実装するためのRubyのAPIを提供

libffi and bindings

libffiとバインディング



Impl. language
実装言語

	Ext★	SWIG	FFI	GI☆
Base Tech	Ext	Ext	lib ffi	lib ffi
Impl. lang	C	C+α	Ruby	Ruby

★ Extension library（拡張ライブラリー）
☆ GObject Introspection: Recommended（オススメ）

Ext impl. by
拡張ライブラリーの実装方法

	Ext★	SWIG	FFI	GI☆
Base Tech	Ext	Ext	libffi	libffi
Impl. by	**Hand**	**Generate**	Hand	Gene-rate

★ Extension library（拡張ライブラリー）
☆ GObject Introspection: Recommended（オススメ）

Ext impl. by: Target
拡張ライブラリーの実装方法：対象

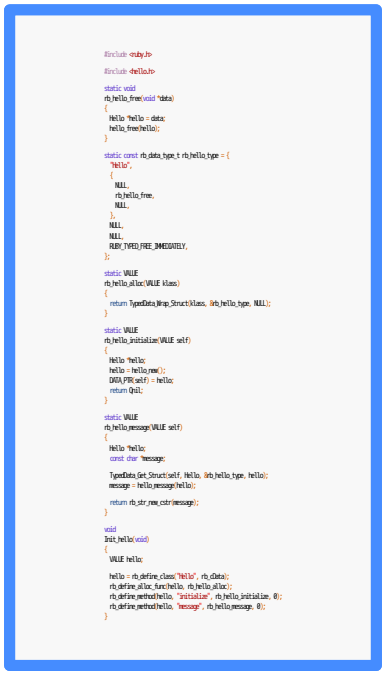
```
/* hello.h */
#pragma once
typedef struct hello_t Hello;
Hello       *hello_new    (void);
void         hello_free   (Hello *hello);
const char *hello_message(Hello *hello);
```

Ext impl. by hand
手動での拡張ライブラリーの実装

Implemented in C by hand
Use C API provided by Ruby

Ext impl. by hand
手動での拡張ライブラリーの実装

https://github.com/kou/rabbit-slide-kou-rubykaigi-2016/blob/master/c-api/bindings/hello.c



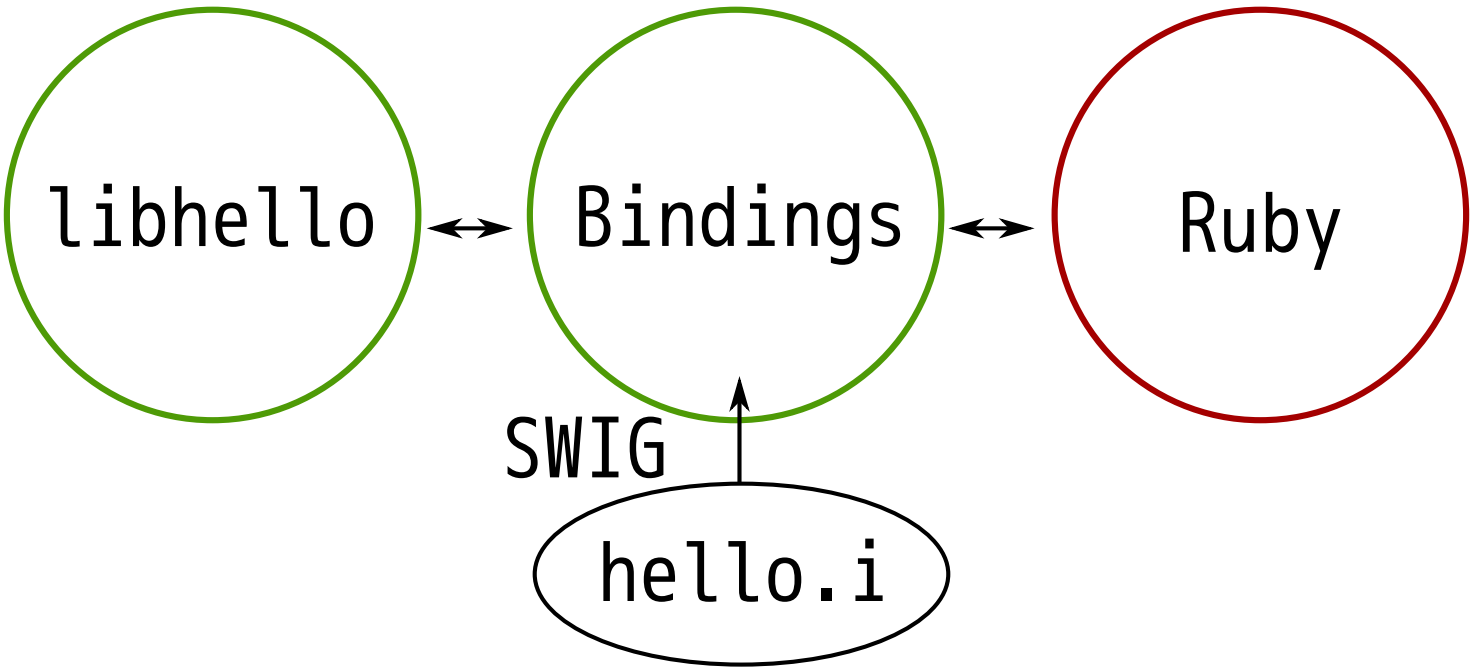
```c
#include <ruby.h>

#include <hello.h>

static void
rb_hello_free(void *data)
{
  Hello hello = data;
  hello_free(hello);
}

static const rb_data_type_t rb_hello_type = {
  "Hello",
  {
    NULL,
    rb_hello_free,
    NULL,
  },
  NULL,
  NULL,
  RUBY_TYPED_FREE_IMMEDIATELY,
};

static VALUE
rb_hello_alloc(VALUE klass)
{
  return TypedData_Wrap_Struct(klass, &rb_hello_type, NULL);
}

static VALUE
rb_hello_initialize(VALUE self)
{
  Hello hello;
  hello = hello_new();
  DATA_PTR(self) = hello;
  return Qnil;
}

static VALUE
rb_hello_message(VALUE self)
{
  Hello hello;
  const char *message;

  TypedData_Get_Struct(self, Hello, &rb_hello_type, hello);
  message = hello_message(hello);

  return rb_str_new_cstr(message);
}

void
Init_hello(void)
{
  VALUE hello;

  hello = rb_define_class("Hello", rb_cData);
  rb_define_alloc_func(hello, rb_hello_alloc);
  rb_define_method(hello, "initialize", rb_hello_initialize, 0);
  rb_define_method(hello, "message", rb_hello_message, 0);
}
```

Ext impl. by generation

自動生成での拡張ライブラリーの実装

Generated by SWIG

Use C API provided by Ruby



Ext impl. by generation

自動生成での拡張ライブラリーの実装

- SWIG can generate impl.
  - Simplified Wrapper and Interface Generator
  - SWIGで実装を生成できる
- A user
  - TensorFlow: A library for machine intelligence
    - TensorFlow：機械知能のライブラリー
    - Used to generate Python bindings
    - Pythonバインディングを生成するために使用

Ext impl. by SWIG

SWIGでの拡張ライブラリーの実装

```
// swig -ruby hello.i -> hello_wrap.c
%module hello
%{
#include <hello.h>
%}
%include <hello.h>
```

Use ext by SWIG

SWIGで作った拡張ライブラリーを使う

```
irb(main):001:0> require "hello"
=> true
irb(main):002:0> hello = Hello.hello_new
=> #<SWIG::TYPE_p_hello_t:0x000...>
irb(main):003:0> Hello.hello_message(hello)
=> "Hello"
irb(main):004:0> Hello.hello_free(hello)
=> nil
# Raw C API... :<
```

By SWIG again

SWIGでもう一度

```
%module hello
%{#include <hello.h>%}
typedef struct hello_t {
  %extend {
    hello_t() {return hello_new();}
    ~hello_t() {hello_free($self);}
    const char *message() {return hello_message($self);}
  }
} Hello;
```

Use again
もう一度使う

```
irb(main):001:0> require "hello"
=> true
irb(main):002:0> hello = Hello::Hello.new
=> #<Hello::Hello:0x000...>
irb(main):003:0> hello.message
=> "Hello"
# Object oriented API!
```

Wrap up1: Ext
まとめ1：拡張ライブラリー

- By hand: Needs more work
  手動：たくさん書かないといけない
- By generation: Less work
  生成：書くことが少ない
  - Needs more work for easy use
    使いやすいAPIにするにはたくさん書かないといけない

Wrap up2: Ext
まとめ2：拡張ライブラリー

- On maintenance

  e.g.: New functions, enums

  メンテナンス時：（例：新しい関数・列挙値が追加された）

  - By hand: Needs more work

    手動：追加作業あり

  - By generation: No more work

    (But the bindings may not be easy to use)

    生成：（使いやすくないけど）追加作業なし

    This is a large benefit!

    これは大きな利点！

libffi impl. by

libffiベースでの実装方法

	Ext★	SWIG	FFI	GI☆
Base Tech	Ext	Ext	libffi	libffi
Impl. by	Hand	Gene-rate	**Hand**	**Generate**

★ Extension library（拡張ライブラリー）
☆ GObject Introspection: Recommended（オススメ）

libffi impl. by hand

手動でのlibffiベースの実装

Implemented in Ruby by hand

Use Ruby API provided by Ruby FFI

libffi impl. by hand

手動でのlibffiベースの実装

```
require "ffi"

module LibHello
  extend FFI::Library
  ffi_lib "../../libhello/lib/libhello.so"
  attach_function :hello_new, [], :pointer
  attach_function :hello_message, [:pointer], :string
  attach_function :hello_free, [:pointer], :void
end
```

Use libffi impl. by hand

手動でのlibffiベースの実装を使う

```
irb(main):001:0> require "hello"
=> true
irb(main):002:0> hello = LibHello.hello_new
=> #<FFI::Pointer address=0x00000002b07ef0>
irb(main):003:0> LibHello.hello_message(hello)
=> "Hello"
irb(main):004:0> LibHello.hello_free(hello)
=> nil
# Raw C API... :<
```

Wrap FFI impl. 1

FFIでの実装をラップ1

```
class Hello
  def initialize
    hello = LibHello.hello_new
    hello_free = LibHello.method(:hello_free)
    @hello =
      FFI::AutoPointer.new(hello, hello_free)
  end
end
```

Wrap FFI impl. 2

FFIでの実装をラップ2

```
class Hello
  def message
    LibHello.hello_message(@hello)
  end
end
```

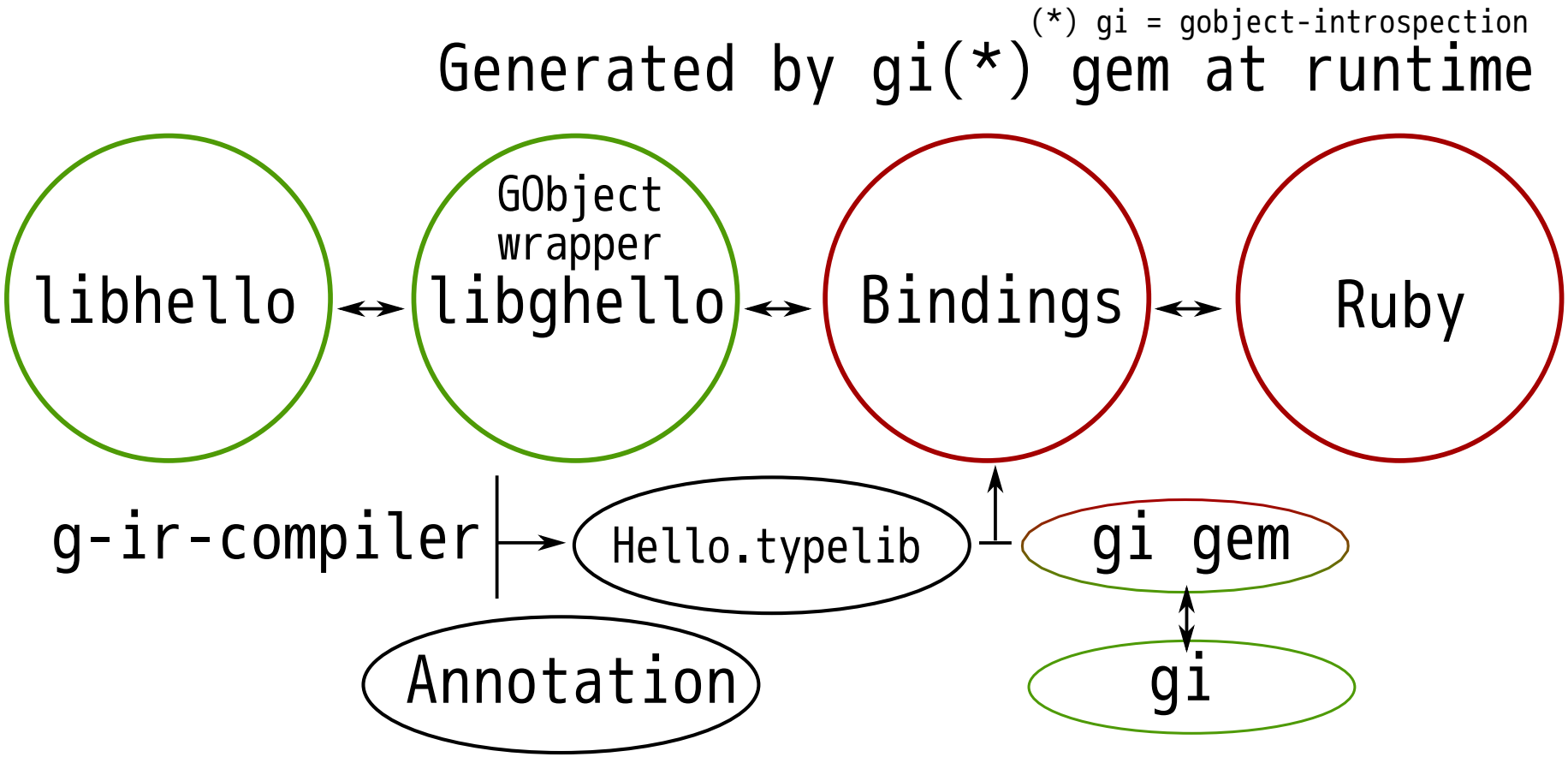
Use wrapped impl.

ラップした実装を使う

```
irb(main):001:0> require "hello"
=> true
irb(main):002:0> hello = Hello.new
=> #<Hello:0x000...>
irb(main):003:0> hello.message
=> "Hello"
# Object oriented API!
```

libffi impl. by gen.

自動生成でのlibffiベースの実装



libffi impl. by gen.

自動生成でのlibffiベースの実装

```
require "gi"
Hello = GI.load("Hello")
```

Use libffi impl. by gen.

自動生成でのlibffiベースの実装を使う

```
irb(main):001:0> require "hello"
=> true
irb(main):002:0> hello = Hello::Hello.new
=> #<Hello::Hello:0x2a9de98 ptr=0x2ecd540>
irb(main):003:0> hello.message
=> "Hello"
# Object oriented API!
```

Wrap up1: libffi

まとめ1：libffi

- By hand: Needs more work
  手動：たくさん書かないといけない
  - Needs more work for easy use
    使いやすいAPIにするにはさらに書かないといけない
- By generation: Less work
  生成：書くことが少ない
  - No more work for easy use
    しかも使いやすいAPIになる

Wrap up2: libffi

まとめ2：libffi

- On maintenance

  e.g.: New functions, enums

  メンテナンス時：（例：新しい関数・列挙値が追加された）

  - By hand: Needs more work

    手動：追加作業あり

  - By generation: No more work

    生成：追加作業なし

    This is a large benefit!

    これは大きな利点！

Impl. by generation

生成ベースでの実装方法

	Ext★	SWIG	FFI	GI☆
Base Tech	Ext	Ext	lib ffi	lib ffi
Impl. by	Hand	**Generate**	Hand	**Generate**

★ Extension library（拡張ライブラリー）
☆ GObject Introspection: Recommended（オススメ）

SWIG⇔GI: When

SWIG⇔GI：生成タイミング

- When are bindings generated?
  バインディングの生成タイミング
  - On build⇔Runtime
    ビルド時⇔実行時
  - SWIG: Need to build for new ver.
    新しいバージョンがでたらリビルドが必要
  - GI: No more work for new ver.
    新しいバージョンがでても追加作業なし

SWIG⇔GI: Maintenance

SWIG⇔GI：メンテナンス

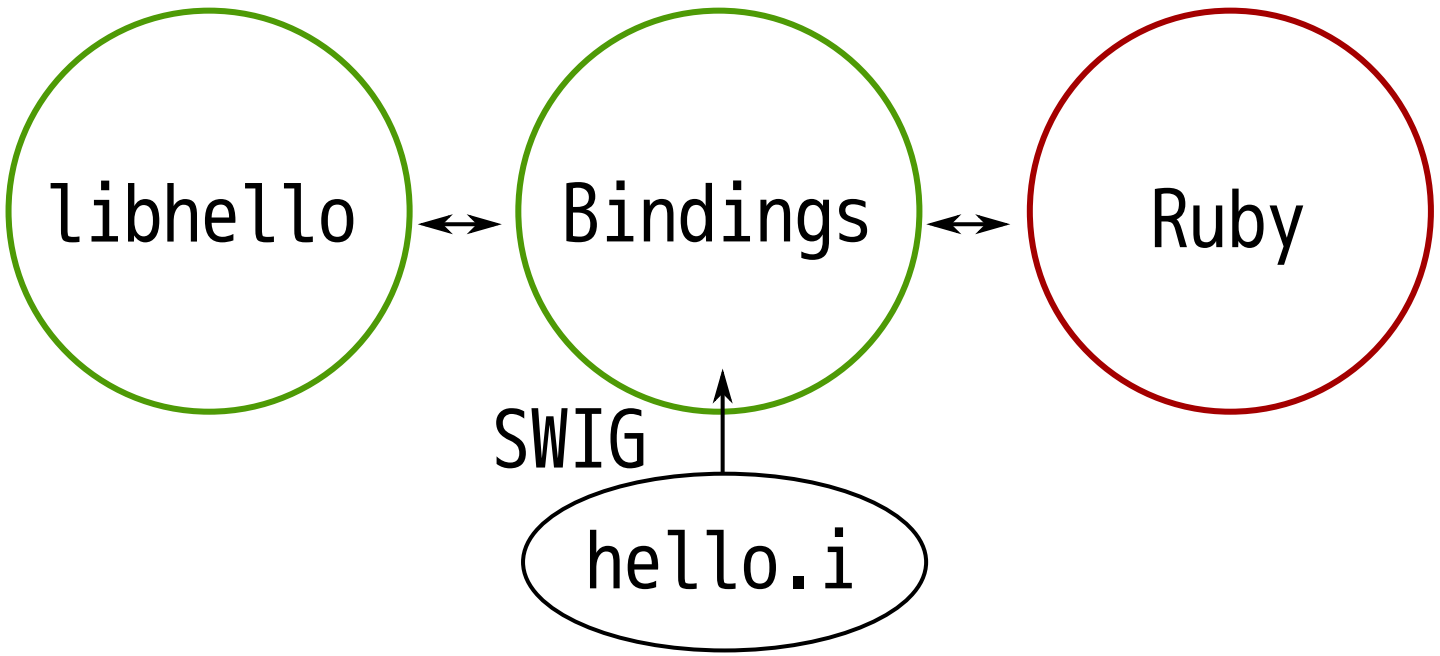
- ✓ SWIG: Maintain .i file for each language binding
  各言語バインディング用に.iファイルをメンテナンス
- ✓ GI: Maintain annotations for all language bindings
  全言語バインディング用にアノテーションをメンテナンス
  - ✓ We can work together with other language binding maintainers

SWIG: Overview (reprise)

SWIG：概要（再掲）

Generated by SWIG

Use C API provided by Ruby

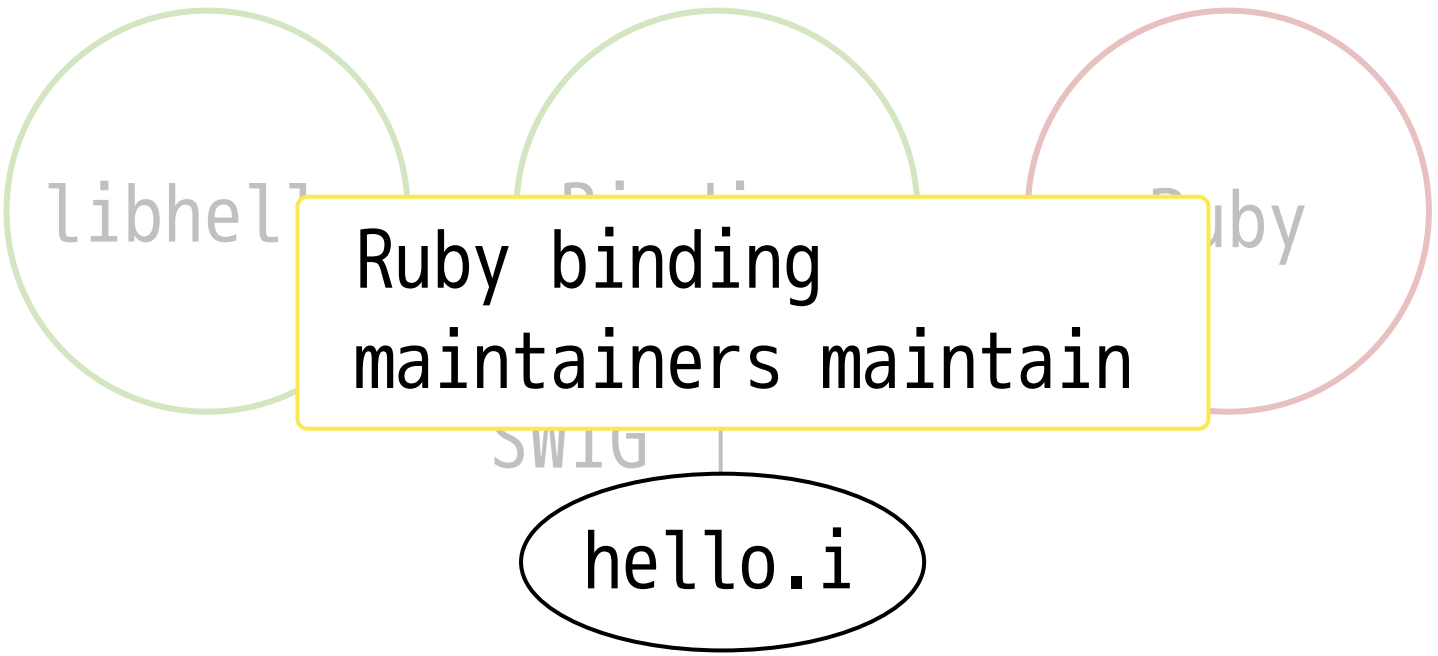


SWIG: Maintenance

SWIG：メンテナンス

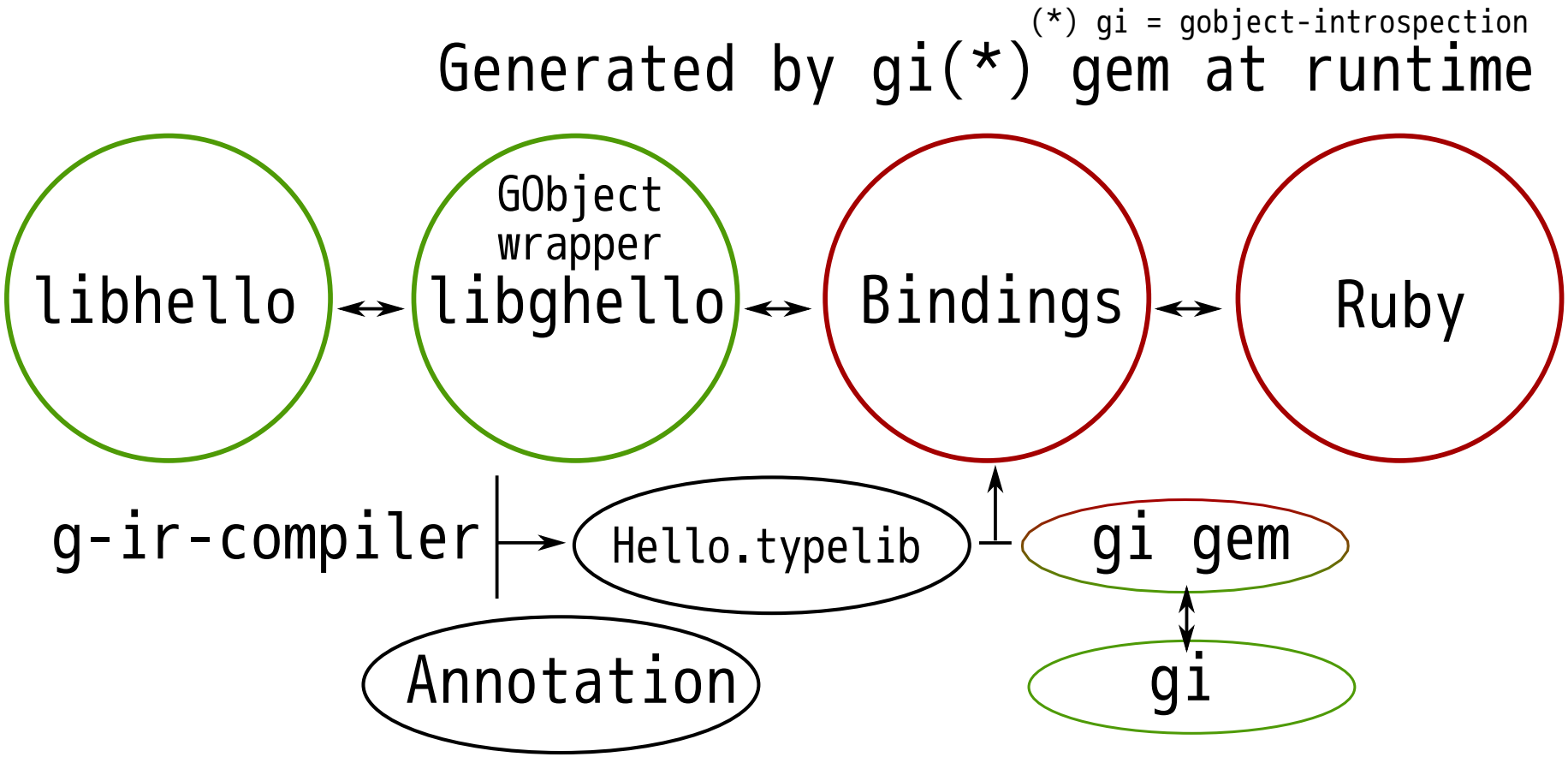
Generated by SWIG

Use C API provided by Ruby



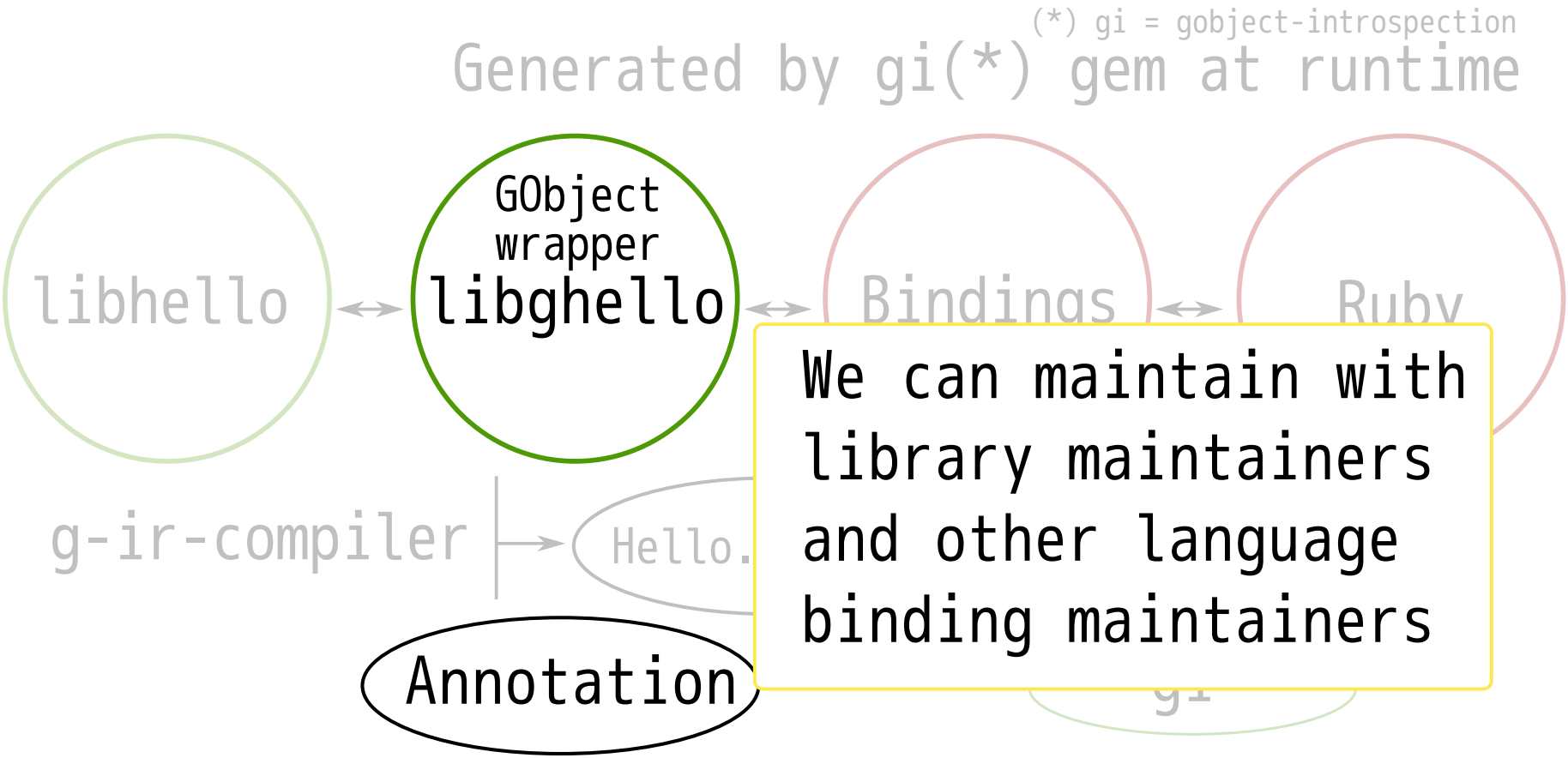
GI: Overview (reprise)

GI：概要（再掲）



GI: Maintenance

GI：メンテナンス



Wrap up1
まとめ1

- Bindings: Glue of C and Ruby
  バインディング：主にCとRubyをつなぐもの
  - You can use features implemented in C from Ruby
    Cで実装された機能をRubyから使える

Wrap up2
まとめ2

- ✓ Bindings increase cases that Ruby can be used
  バインディングはRubyを使えるケースを増やす
  - ✓ Because they provide existing good features to Ruby
    バインディングは既存のよい機能をRubyで使えるようにするから

Wrap up3
まとめ3

- ✓ Recommend GI based bindings
  GIベースのバインディングがオススメ
  - ✓ For easy maintenance
    メンテンナンスしやすいから
  - ✓ For easy to use API
    使いやすいAPIになるから

GI: GObject Introspection

Wrap up4
まとめ4

Let's become a bindings developer!

バインディング開発者になろう！

Advertisement
宣伝

- OSS Gate
  - Helps people who want to be an OSS developer (but not yet)
    OSS開発者になりたいけど一歩踏み出せていない人を支援
- ClearCode booth（クリアコードブース）
  - You can discus about OSS Gate and more technical details
    OSS Gateやもっと技術的な話や雑談をできる

Wrap up (reprise)
まとめ（再掲）

Let's become a bindings developer!

バインディング開発者になろう！

Things not covered
話さなかったこと

- ✓ Bindings are difficult to install on Win. Any idea?
  Windowsでインストールが大変。どうすれば？
- ✓ Performance・Annotation
  性能・アノテーション
- ✓ Details of each binding's create method
  それぞれのバインディング作成方法の詳細