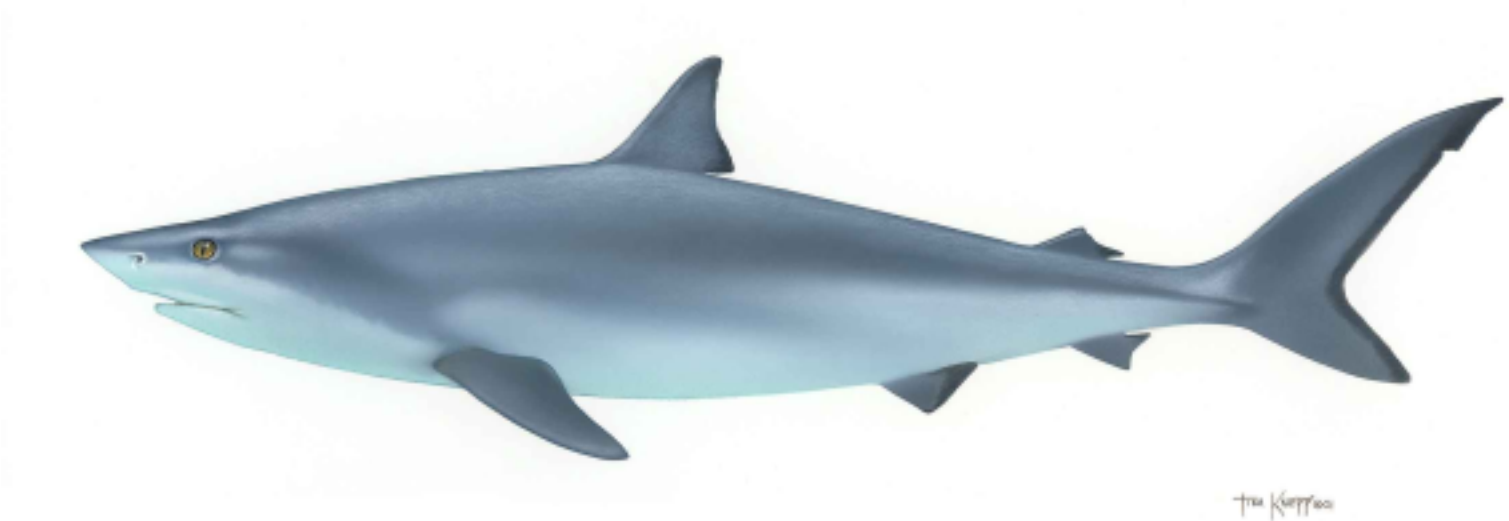
Feeding the Sharks

Ruby Association
Heroku

@yukihiro_matz
Yukihiro "Matz" Matsumoto

or

Clash of Types



OSS Community is like a shark

Especially developers' community

We have to move forward, or die

If we lose interest, we will go away

To somewhere else, more interesting

We have to feed the community

We have to attract the community

Somehow

By showing how to earn money

Rails!

By enlightening people

Philosophy

By showing the future

The possible future

Past Keynotes of RubyConf

I gave presentations about the future

RubyConf 2001

The first RubyConf in Tampa

- Virtual Machine

- Virtual Machine (1.9 2007)

RubyConf 2002

In Seattle

- M17N
- Native thread
- Generational GC

- M17N (1.9 2007)
- Native thread (1.9 2007)
- Generational GC (2.1 2013)

RubyConf 2003

In Austin

- Local variable scope
- Multiple assignment
- Keyword argument
- Method combination
- Selector namespace
- Optional static type

- Local variable scope (--)
- Multiple assignment(1.9 2007)
- Keyword argument (2.0 2013)
- Method combination (2.0 2013)
- Selector namespace (2.0 2013)
- Optional static type (--)

RubyConf 2004

In Washington DC

My yougest daughter was born

Brad Cox gave a keynote

Koichi gave his first talk on YARV

RubyConf 2005

In San Diego

- Stabby lambda (->)
- Real multi-value
- Traits

- Stabby lambda (1.9 2007)
- Real multi-value (--)
- Traits (--)

RubyConf 2006

In Denver

Bikeshed argument encouraged

No new ideas

RubyConf 2007

In Charlotte

1.9 introduced

No new ideas

RubyConf 2008

In Portland

Philosophy explained

No new ideas

RubyConf 2009

In San Francisco

Power of DSL explained

RubyKaigi 2009

- Complex literal
- Rational literal
- True division (1/2 => 1/2)
- Bitmap marking
- Symbol GC

- Complex literal (2.1 2013)
- Rational (2.1 2013)
- True division (--)
- Bitmap marking (2.0 2013)
- Symbol GC (2.2 2014)

RubyConf 2010

In New Orleans

- Mix (traits)
- Module#prepend
- Refinement
- Rite (mruby)

- Mix (--)
- Module#prepend (2.0 2013)
- Refinement (2.0 2013)
- mruby (2012)

RubyConf 2011-2013

New Orleans, Denver and Miami

No new ideas

After all,

Some may become true, some may not

False rate
7/22 ≒ 32%

2001-2005

Exciting (but uncertain) ideas

2006-2008

Nothing new, but philosophy

2009-2013

Improving implementation

We need fuel to move on

It's about time start talking about:

Ruby 3.0

Ruby 2.2

May happen in next 10 years

- Concurrency
- JIT
- Static typing

Concurrency

JIT (LLVM?)

Static typing

Static typing?

All new kids in the street

Scala

TypeScript

Dart

Go

Why not Ruby?

Clash of Types

Feature #9999

by Davide D'Agostino

Type Annotations

```
def connect(r -> Stream, c -> Client) -> Fiber
  ...
end
```

Python

PEP: 3107

Function Annotations

```
def connect(r: Stream, c: Client) ->  Fiber:
  ...
```

mypy

Optional static type checker

Benefits of static typing?

- Performance
- Compile-time check
- Documentation

Performance

No one complains for faster Ruby

But do we really need static typing for speed?

JavaScript V8

LuaJIT

JIT

Specialization

Performace with dynamic typing

We don't need static typing for speed

Compile-time check

Static analysis

Refactoring

Test coverage

But less flexible

Against Duck typing

Documentation

Much better than comments

No contradiction

No investigation into details

That is PEP-3107's intention

Why not static typing?

- Duck typing
- Optional
- DRY

Duck typing

Static typing is against duck typing

Guy Decoux

Optional

Optional typing is only useful with 99% coverage

TypeScript

dynamic

Ruby without duck typing, really Ruby?

Ruby should keep being Ruby, forever

DRY

Don't Repeat Yourself

Avoid duplication

Static typing is against DRY principle

Code & Declaration

Soft-typing[1]

[1] Soft Typing, Robert Cartwright and Mike Fagan, 1991

[2] Soft typing: An approach to type checking for dynamically typed languages, Mike Fagan, 1991

No declaration needed

Best-effort type checker

Based on duck typing

Type inference

```
a=1   # type of a is Integer
```

```ruby
# x requires to have to_int
def foo(x)
  print x.to_int
end

foo(1)    # OK: 1 has to_int
foo("a") # NG: "a" does not have to_int
```

Type is represented by:

- Set of methods
  - name
  - number and type of arguments
- Class (as set of methods)

Compile-time check

Best-effort type checker

Targets subset of the language

Restricted dynamic nature

For example,

- require
- define_method
- method_missing

Documentation

Unlike other languages

You don't tell compiler types

Compiler will guess your intention

And report back to you

And generates doc / IDE info

Closer communication between compiler and you

Soft typing means 2 languages in one

- Statically soft typed language
- Dynamic typed language

When soft typing is not applicable

It fallbacks to dynamic typing

Strongly encouraging the former

First, it should be done by a static analyzer

For quicker error detection

Or for better IDE integration

This is just an idea

May or may not happen

But it's about time to start new things

That leads us Ruby 3.0

Prepare for the future

Happy hacking

Thank you