Sorbet:
A Typechecker for Ruby

Dmitry @darkdimius Petrashko
Nelson @nelhage Elhage
Paul @ptarjan Tarjan

stripe

Introductions

Speaker notes

- dmitry: PhD Compiler architecture & a bit of type theory @ next major version of Scala Compiler(3.0)
- nelhage: MIT grad, One of the longest tenured engineers at Stripe
- pt: Stanford grad, Previously at Facebook on HHVM and Hack

pt starts talking here

Background

Speaker notes

Don't say anything

Stripe

- Platform that developers use to accept payments
- 25 countries, 100,000 business worldwide
- 60% of people in US have used it in the last year
- If you're running an internet business, check us out
- Have a Tokyo office
  - Many in the audience
  - Come chat!
  - stripe.com/jobs

Developer productivity

Large dedicated team

- Testing
- Code
- Dev Env
- Abstractions
- Language Tools
- Etc.

Speaker notes

- Context
- Not overhead

Ruby at Stripe

- Ruby is the primary programming language
  - No Rails
  - Enforced subset of Ruby (thanks @bbatsov for Rubocop)
- Most product code is in a monorepo (intentionally!)
- ~10 macroservices with a few microservices
- New code mostly goes into an existing service

Speaker notes

- Millions of lines of Ruby
- Bozhidar Batsov

Stats

Language	lines	Language	lines
Ruby	34%	Scala	7%
Javascript	16%	HTML	6%
YAML	10%	Go	6%

Millions of lines of code

Hundreds of engineers

Thousands of commits per day

Speaker notes

- Top languages
- Most product engineers use Ruby.
- Next language is 2% - bash

Other ruby typing

- DRuby / PRuby / RubyDust / RTC / RDL by Jeff Foster
- Unreleased GitHub experiment by @charliesom
- Presentations tomorrow by @soutaro and @mametter
- Contracts from JetBrains by @valich

Speaker notes

- Diamondback Ruby
- RDL Jeff Foster Maryland
- Charlie Somerville at Github
- Soutaro Matsumoto
- Yusuke Endoh
- Valentin Fondaratov

Open source?

- Yes! Eventually
- Prove it out internally first
- Have questions? Reach us at sorbet@stripe.com

Speaker notes

- Yes, we very much would like to open source this.
- today gather feedback and find folks who are interested in collaborating.
- I hate throwing code over the wall. Instead we want to release when we can dedicate the time to helping the community use it.
- But wait, this isn't just a vaporware announcement :) We have a browser demo.

Now open: https://sorbet.run/#%221%22%20%2B%202

Try it ⇩

https://sorbet.run

Speaker notes

And with that brief teaser, let me hand off to Dmitry to dive into our type system.

- Explicit

  We're willing to write annotations, and in fact see them as beneficial; They make code more readable and predictable. We're here to help readers as much as writers.

- Feel useful, not burdensome

  While it is explicit, we are putting effort into making it concise. This shows in multiple ways:

  - error messages should be clear
  - verbosity should be compensated with more safety

- As simple as possible, but powerful enough

  Overall, we are not strong believers in super-complex type systems. They have their place, and we need a fair amount of expressive power to model (enough) real Ruby code, but all else being equal we want to be simpler. We believe that such a system scales better, and -- most importantly -- is easiest for users to learn+understand.

- Compatible with Ruby

  In particular, we don't want new syntax. Existing Ruby syntax means we can leverage most of our existing tooling (editors, etc). Also, the whole point here is to improve an existing Ruby codebase, so we should be able to adopt it incrementally.

- Scales

  On all axes: in speed, team size, codebase size and time (not postponing hard decisions). We're already a large codebase, and will only get larger.

- Can be adopted gradually

  In order to make adoption possible at scale, we cannot require all the teams to adopt it at once, thus we need to support teams adopting it at different pace.

Demo (usage)

Speaker notes

One of the most areas where those principles can be seen is error messages. The following slides show several examples of them.

Usage: Calls into stdlib

```ruby
# Integer
([1, 2].count + 3).next
```

```ruby
"str" + :sym
```

```
Expression passed as an argument `arg0` to method `+`
      does not match expected type
      github.com/stripe/sorbet/wiki/7002
     1 |"str" + :sym
          ^^^^^^^^^^^^

    github.com/stripe/sorbet/tree/master/rbi/core/string.rbi#L18:
    Method `+` has specified type of argument `arg0` as `String`
    18 |      arg0: String,
              ^^^^^

  Got Symbol(:"sym") originating from:
    -e:1:
     1 |"str" + :sym
                ^^^^
```

Usage: truthiness

```ruby
foo = array_of_strings[0]
# foo is a T.nilable(String) now
return true if foo.nil?
# foo is a String now
return foo.empty?
```

```ruby
foo = array_of_strings[0]
return foo.empty?
```

```
Method `empty?` does not exist on `NilClass`
component of `T.nilable(String)`
     5 |return foo.empty?
                 ^^^^^^^^^^
```

Usage: dead code

```
if Random.rand > 0.5
    foo = 1
else
    foo = 2
end
# foo is an Integer
```

```
if Random.rand
    foo = 1
else
    foo = 2
end
```

```
This code is unreachable
     6 |     foo = 2
```

Speaker notes

- We see that foo is assinged on both cases in first example
  - second example has a bug that could have been made by a C person.

Usage: union types

```
str_or_int = ["1", 2].sample
hash = str_or_int.succ
```

```
str_or_int_or_array = ["1", 2, [3]].sample
hash = str_or_int_or_array.succ
```

```
Method `succ` does not exist on `Array` component of
`T.any(String, Integer, T::Array[Integer])`
     4 |hash = str_or_int_or_array.succ
                                 ^^^^^
```

Declaration Syntax

Declaration: Compatible syntax

```
extend T::Helpers
...
sig(
    # Both positional and named parameters are referred to by name.
    # You must declare all parameters (and the return value below).
    amount: Integer,
    currency: String,
)
.returns(Stripe::Charge)
def create_charge(amount, currency)
    ...
end
```

Speaker notes

here you can see the syntax used to declare types of arguments of the method. This syntax is a ruby dsl. As you can see (~read the slide)

Declaration: Runtime Typesystem

```
sig(amount: Integer, currency: String)
.returns(Stripe::Charge)
def create_charge(amount, currency)
    ...
end

create_charge(10_000, :jpy)
```

```
#<TypeError: Parameter currency:
  Expected type String, got type Symbol>
```

Speaker notes

Even before we've started building a static typechecker, we have built a dynamic one. This dynamic one uses data provided via same dsl to perform runtime type checks in production. If they fail, it raises an error that will include a complete stacktrace as well as data that violated the type constraint.

Declaration: Local Inference

```
sig.returns(String) # Optional but not inferred
def foo
    a = 5 # Integer
    a = T.let("str", String) # String
end
```

Speaker notes

some people think that typesystems can be too verbose. We agree. This is why we want our to be concise. In particular here you don't need to specify type of variable a, we can infer that it is integer.

If you do want to declare a type of variable, we you can do so with T.let. In this example, you re-assign a to a string and you explicitly daclare that you wanted a to become a string.

Declaration: strictness level

```
#
# Basic checks such as syntax errors are enabled by default
```

```
# typed: true
# Enables type checking
```

```
# typed: strict
# Additionally requires ivars to be declared
```

```
# typed: strong
# The safest level: disallows calling untyped code
```

Declaration: Generic classes

```
class Box
  extend T::Generic

  Elem = type_member

  sig.returns(Elem)
  attr_reader :x

  sig(x: Elem).returns(Elem)
  attr_writer :x
end
```

Speaker notes

Our typesystem also has minimal number of features to model Ruby code. One of such features is generics. In this example, we model a box that can store an element of a specific type. Box declaration does not know what it will store. I will be specified by use site.

The most common use of this is to model various containers: arrays, sets, hashes

Declaration: Generic methods

```
class Array
  Elem = type_member

  type_parameters(:U).sig(
      blk: T.proc(arg0: Elem).returns(T.type_parameter(:U)),
  )
  .returns(T::Array[T.type_parameter(:U)])
  def map(&blk); end
end
```

Speaker notes

We can also model methods with complex signatures such as Array map that require generic methods. They are somewhat verbose, but they are very descriptive and express the usecase well.

Hand of to Nelson

Practical experience

Speaker notes

Throwback to the title of our talk -- "a practical typechecker for Ruby" -- and I want to link this to our experience at Stripe

Internal rollout

- Working on this since last year
- Runtime types have been deployed for 6 months
- Static checker in internal beta
  - engineers can opt in
- Command-line tool

Early adoption

- Human-authored signatures: 3k
- Files annotated by users: 150+
- Generated signatures: 240k

Some bugs we found

Speaker notes

These are some bugs we found in the process of rolling out the typechecker, that slipped through CI and code review. Fortunately our test coverage and processes are pretty good so none of these were critical, but they are pretty informative of the experience of using the tool and the kinds of issues we can catch.

- Examples are simplified but based on real code.

Typos in error handling

```
    begin
      data = JSON.parse(File.read(path))
    rescue JSON::ParseError => e
      raise "#{PACKAGE_REL_PATH} contains invalid JSON: #{e}"
    end
```

```
json.rb:6: Unable to resolve constant ParseError
    6 |  rescue JSON::ParseError => e
                ^^^^^^^^^^^^^^^^
   shims/gems/json.rbi:319: Did you mean: ::JSON::ParserError?
    319 |class JSON::ParserError < JSON::JSONError
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Errors in error handling

```
  if look_ahead_days < 1 || look_ahead_days > 30
    raise ArgumentError('look_ahead_days must be between […]')
  end
```

```
argumenterror.rb:9: Method ArgumentError does not exist on […]
```

Speaker notes

Here we see another error inside an error-handling block. Someone tried to call the function `ArgumentError`, instead of calling its constructor -- perhaps they have been writing too much Python.

Here again we see that sorbert has identified the error and reported that that method does not exist.

`nil` checks

```
app.post '/v1/webhook/:id/update' do
  endpoint = WebhookEndpoint.load(params[:id])
  update_webhook(endpoint, params)
end
```

```
webhook.rb:16: Expression passed as an argument `endpoint`
    to method `update_webhook` does not match expected type
  16 |    update_webhook(webhook, params)
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

webhook.rb:10: Method update_webhook has specified type
    of argument endpoint as WebhookEndpoint
```

Speaker notes

- strict nilability found a bug in an endpoint that handled an API request.
- This endpoint takes a Webhook ID from a user and updates it in our system
- the typechecker knows that `load` returns a webhook object or `nil` if that ID doesn't exist, and that `update_webhook` needs a non-`nil` webhook objecrt
- We detect the missing check

`nil` checks (fixed)

```
app.post '/v1/webhook/:id/update' do
  endpoint = WebhookEndpoint.load(params[:id])
  if endpoint.nil?
    raise UserError.new(:webhook_endpoint_not_found)
  end
  update_webhook(endpoint, params)
end
```

Speaker notes

- Easy to fix; If we add the natural check you should have written anyways, we recognize the pattern and quiet the error

Instance variables from `self.`

```ruby
# typed: strict
class ChargeCreator
  def initialize
    @request = …
  end

  def self.log_results
    log.info("charge created request_id=#{@request&.trace_id}")
  end
end
```

```
charge.rb:9: Use of undeclared variable @request
     9 |    log.info("charge created request_id=#{@request&.trace_id}"
                                                  ^^^^^^^^
```

Speaker notes

Files can opt-in to require declaring instance and class variables. Here we found a bug in existing code where an instance variable was set and then attempted to be accessed from the wrong scope.

Incorrect pattern matching

```
case transaction
    ...
    when Stripe::Charge || Stripe::Refund
    ...
end
```

```
case.rb:12: This code is unreachable
    12 |when Stripe::Refund || Stripe::Charge
                             ^^^^^^^^^^^^^^
```

Speaker notes

Do a careful walk through

User response

Nice!! Beautiful errors and damn that was fast!!

“DeveloperHappiness” would be a good name for ruby-typer

I'm trying to use it locally, it's been super helpful to catch bugs in my own code.

I introduced a typo with sed and ruby-typer caught it in CI within seconds of me pushing the branch. It’s really nice to get this kind of notification quickly rather than having to wait for potentially several minutes before the test job fails.

Speed of our typer

100k lines/second/cpu core

Tool	Speed (lines/s/core)
`sorbet`	100,000
`javac`	10,000
`rubocop`	1,000

Versus CI

Speed at Stripe

Tool	Speed	Parallel Machines
Sorbet	seconds	1
CI	10 minutes	tens

Implementation

- C++
- Don't depend on a Ruby VM
- C++ port of `whitequark/parser` by GitHub
- Extensive test suite
- CI runs against Stripe codebase

Metaprogramming support

- Minimal native support
- Reflection-based signature generation

Speaker notes

Hand off to pt after this slide

Can I use it?

Open source

- Sneak peak: https://sorbet.run
- Will open source, timeline TBD
- When released we will support it
- Will post on stripe.com/blog
- sorbet@stripe.com

Speaker notes

- Try out the browser demo.
- core typechecker is done, most of the work for us is around how to roll it out to a big codebase.
- How do you deal with metaprogramming, or unannotated gems, or editor integrations.
- Please play with it and give us feedback.

Using it

- Interested in your use cases
- Large orgs scaling with Ruby
- Will let you know when it's ready for beta
- sorbet@stripe.com

Speaker notes

- Interested in your use cases
- Two classes particularly interested in
- Our dev productivity team existed for 2.5 years now and we've focussed largely on how to scale our Ruby for the needs of Stripe, so we have some tools to share.
- We've put a lot of effort into this space.
- Having said that, we also really want to use your tools and systems too!

Building it

- There are multiple parties working to add types to Ruby
- We'd love to chat and share
- sorbet@stripe.com

Speaker notes

- if you are working on typechecking Ruby or scaling Ruby we would love to chat.
- Please email us or find us somehow.

Take away

- We have a typechecker
- Fast, built thoughtfully
- Useful, not burdensome
- Will open source
- Reach out to us
- sorbet@stripe.com

Thank you!

sorbet@stripe.com

Speaker notes

- Thank you to the conference organizers for letting us speak here today.
- Thanks Ruby for the great language that we all build on.
- With that, thank you so much for listening to us and taking that.
- Arigatou gozaimasu