

A Journey Through Ruby Concurrency

RubyConf Brasil 2016
@jerrydantonio



Thank
You



LOCAWEB

Jerry D'Antonio

Akron, Ohio, USA

Software Developer

Test Double



@jerrydantonio
github.com/jdantonio
concurrent-ruby.com





AKRON
CORPORATION LIMIT

Home of....
LeBron James
2012 OLYMPIC GOLD MEDAL
2012 NBA CHAMPIONSHIP
2012 NBA MVP
2003 **NBA** ROOKIE OF
2004 THE YEAR

AKRON
WELCOMES

Let's get started.

A few weeks ago Koichi Sasada, aka ko1,
proposed a new concurrency model for Ruby 3.

And now everyone is talking about concurrency.

Even more than usual.

Including me.

Especially me.



Which is good. It's an important topic, and critical to Ruby's future.

But we need to have the right conversations.

When discussing Ruby concurrency I frequently hear people say wrong things.





Although well meaning, comments like this can sometimes be unhelpful.

Ruby can't necessarily do what Erlang does, or what Node does, or what language X does.

Nor should it.

Because Ruby isn't Erlang or JavaScript or any other language.

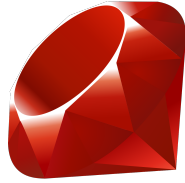
Ruby is Ruby



Ruby is designed to solve different problems.

Ruby has a different runtime,
different philosophies,
and different characteristics.





makes me



Although Ruby can—and should—take
inspiration from other languages.

Ruby must embrace Ruby.

So, with respect to concurrency, what are Ruby's most important characteristics?

Ruby is...

Shared memory

Reference-based

Object oriented

Everything in Ruby is an object.

Ruby has no value types, no true primitives.

Even booleans, symbols, and integers are objects.

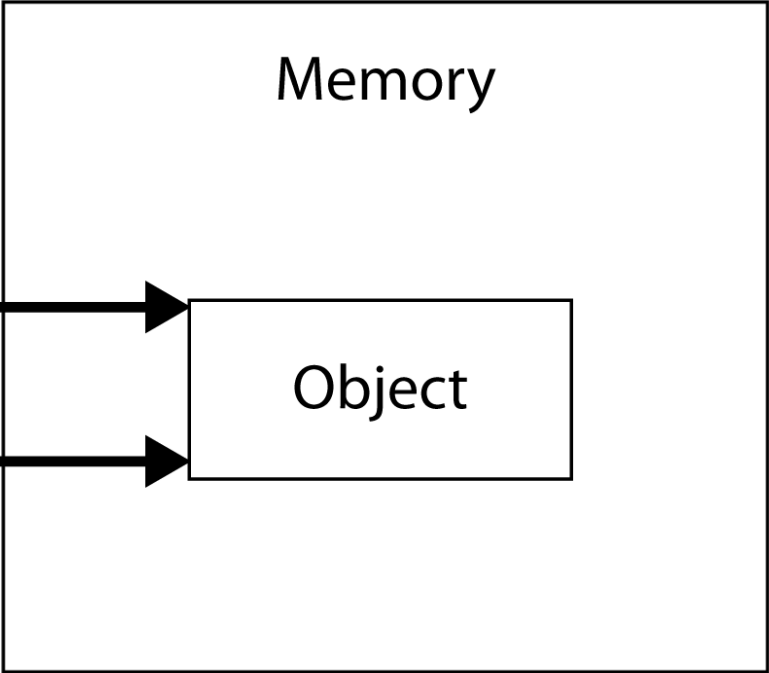
As most of you know...

Variables are not values.

Variables are references to values.

References point to memory locations
on the heap.

Variable Object	
object1	●
object2	●



Objects can be modified from within a method
by following the reference.

And this behavior exists when we pass variables
across thread boundaries.

So two or more threads can each have references to the same object, located in the same memory on the heap.

Which means that, in theory, two threads on two processor cores can attempt to simultaneously modify the same memory.



Not on MRI, of course, because it only uses one core and it has a global interpreter lock (GIL).

But that's sort of a problem, too.

Shared memory, reference-based, object oriented languages offer many advantages.

But they aren't great for concurrency.

Full parallelism with shared-memory, reference-based languages can be very fast and efficient, but is also very difficult to get right.

Locking is hard.

Shared memory, reference-based languages:

Ruby

Python

C, C++

Java

C#

Go

Functional languages like Erlang, Clojure, and Haskell are generally considered better for concurrency.

But they are very different languages.

They favor values over references,
enforce immutability,
and in some cases offer memory isolation.

Erlang, for example...

Has only values, not references.

Has only immutable variables.

Has a very small set of value types.

And enforces strict memory isolation.

This is all very good for concurrency.
But isn't necessarily all good.
Or necessarily better.



!=

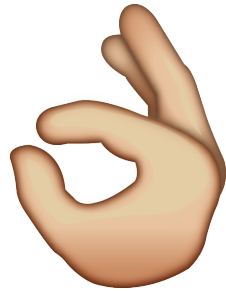




\neq

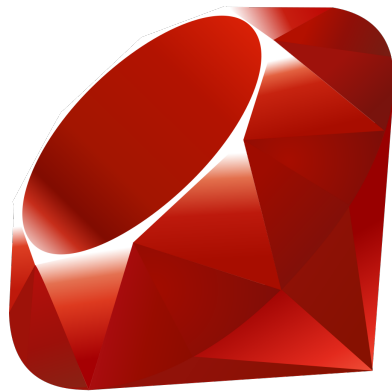


Functional languages like Erlang require different paradigms and patterns, but also pose different challenges.



Some people like programming in functional languages.

I've worked professionally with functional languages and I'm quite fond of them.



Ruby cannot have the same concurrency model as most functional languages without becoming a very different language.


This is a very critical point.

Ruby simply cannot copy what Erlang, Clojure,
or Haskell does without ceasing to be Ruby.

So what do we do?

And by “we” I mean the Concurrent Ruby team.

Because we’ve been working on this problem
for several years.





We create concurrency abstractions that...

Encourage proven designs.

Provide the strongest possible guarantees.

And embrace the Ruby language.



We do three things...



Concurrent Ruby takes its *inspiration* from other languages.

We leverage *ideas* from Erlang, Clojure, Java,
JavaScript, and other languages.
As well as from academic research.

We don't reinvent the wheel, or try to outsmart
the experts.

We create abstractions which enable Ruby programmers to easily implement those proven patterns and practices.

And when programmers follow the guidelines
for what we provide, they create safe
concurrent systems.



Concurrent Ruby provides the strongest possible thread safety guarantees.

Emphasis on the word “possible.”

We can't provide stronger guarantees than what
the runtime provides.

No concurrency library can.

We use the runtime, we don't replace it.

Unfortunately, Ruby does not have a formal memory model.

So we do the best we can with what we have.

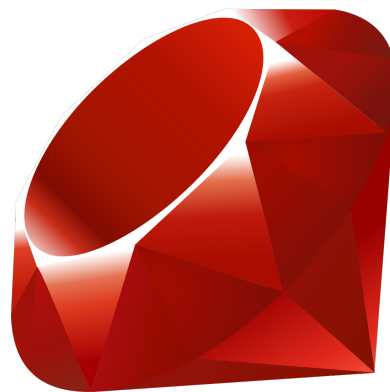
What we cannot guarantee is that your code will be thread safe and free from errors.

- \ (ツ) / -

We can only guarantee that if you follow the guidelines and use our tools the way they were intended that you'll probably be OK.

Which is the only guarantee any concurrency library can give you.





Even though we take inspiration from other languages, we don't directly copy them.

Instead, we interpret them in a uniquely Ruby way.



optimized for programmer





optimized for Rubyist



And I think we do a pretty good job.





A concurrency model called “guilds” was recently proposed.

The core ideas are well-formed and will likely be the basis for concurrency in the next major version of Ruby.

Not surprisingly, it does pretty much everything
we've talked about so far.

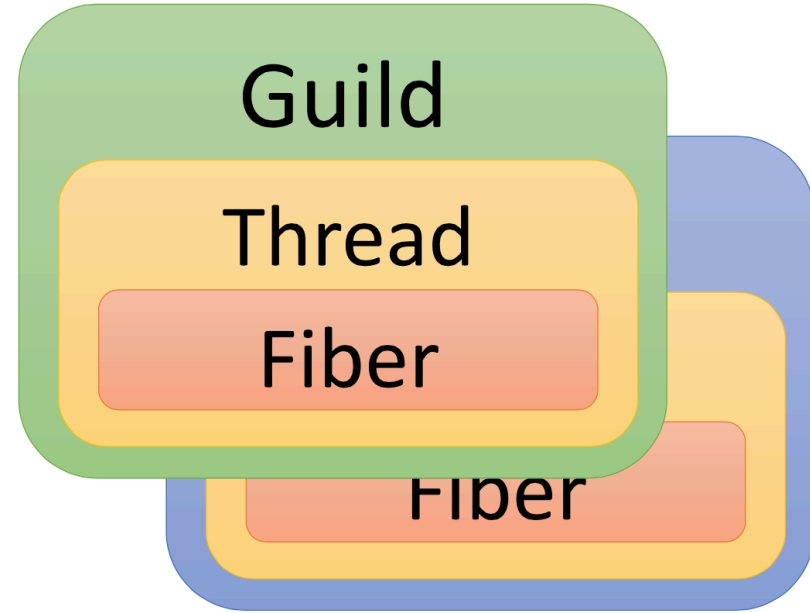
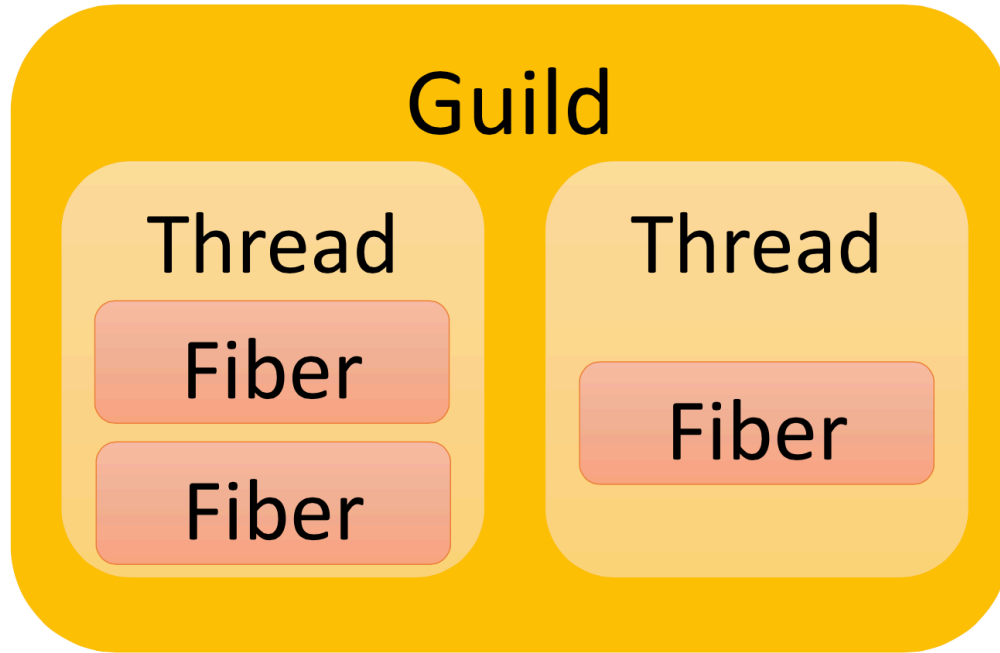
But it does it in the language itself.

Quite simply, guilds...

Encourage proven designs.

Provide the strongest possible guarantees.

Embrace the Ruby language.



** Image totally stolen from ko1's presentation.
Thank you!*

Like Erlang, guilds are isolated from one another.

Guilds cannot access references which are members of other guilds.

Guilds communicate exclusively through channels.

Channels are one-way, and each guild has one implicit channel.

References are passed between guilds through a channel.

References can be passed two ways: deep copy
or membership transfer.

Deep copy is similar to what functional languages do.

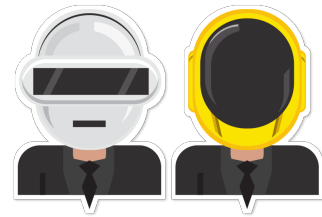
Deep copy can be expensive, and with complex objects can lead to odd situations.

When we transfer membership the object reference is invalidated in the sending guild.

The object now belongs solely and fully to the receiving guild.



Nothing changes. We'll just get harder, better,
faster, stronger.



Guilds are a language-level concurrency model,
not a high level abstraction.

Guilds provide a thread-safe, one-way communication mechanism between isolated threads of execution.

That's it.

This is very important. Critically important. But
it's not enough for most programmers.

Guilds still require much work to be done by the application programmer.

Programmers must:

Design the interactions between guilds

Handle message receipt, probably in a loop.

Setup two-way communication for transferring
results.

Handle exceptions

Communicate errors.

This is why most “concurrent” programming languages provide high-level abstractions in the standard library.

Erlang has `gen_server`, `gen_event`, `gen_fsm`, and
supervisor.

Elixir has Agent and Task.

Scala has Akka.

And there are many other examples.



Guilds do not replace concurrency libraries like
Concurrent Ruby.

Guilds give us better tools to build these
libraries with.

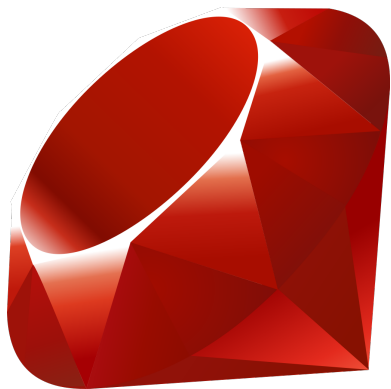


You want to build concurrent applications now.

But you also want to be prepared for Ruby 3.



“So, Jerry, what do *you* think about guilds in Ruby 3?”



Guilds encourage proven concurrency practices.

Guilds provide strong guarantees and a formal memory model.

Guilds allow today's concurrency libraries, like
Concurrent Ruby, to still exist.

But they give us better tools for simplifying our
internals and provide stronger guarantees.

Guilds are heavily influenced by languages that are considered good at concurrency.

While embracing everything I love about Ruby.





Jerry D'Antonio

Akron, OH, USA

@jerrydantonio

