Better Ruby

NaCl
OSS Vision
Ruby Association

Yukihiro "Matz" Matsumoto
@yukihiro_matz

The Closing Keynote

WARNING

This is not a technical talk

Ruby is Good

自画自賛

Tooting one's own horn

Fun to Code

A Programmers’ Best Friend

Rich Set of Standard Features

Features Organized in Classes

Less Restriction

- Integer Size
- Built-in / User Defined

Gems / Tools

Community

Productive

Happiness Leads to Productivity

Ruby on Rails

Rails first Released in 2004

Happy 20th Anniversary, Rails!

Still Being State-of-the-art Framework

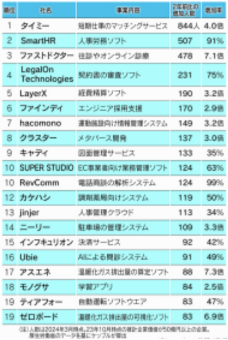
So Many People/Companies use Rails

Ruby/Rails Drives the Society

TOP RUBY COMPANIES

https://toprubycompanies.info/

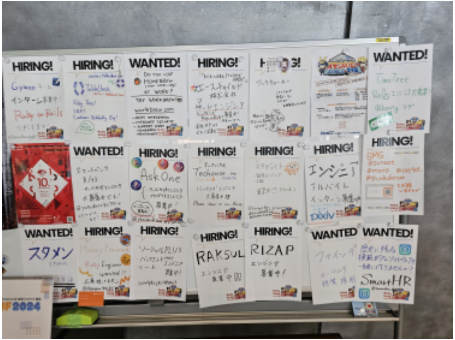
Fast Growing Startups in Japan



順位	社名	事業内容	2年前比の 増加人数	増加率
1	タイミー	短期仕事のマッチングサービス	844人	4.0倍
2	SmartHR	人事労務ソフト	507	91%
3	ファストドクター	往診やオンライン診療	478	7.1倍
4	LegalOn Technologies	契約書の審査ソフト	231	75%
5	LayerX	経費精算ソフト	190	3.2倍
6	ファインディ	エンジニア採用支援	170	2.9倍
7	hacomono	運動施設向け情報管理システム	149	3.2倍
8	クラスター	メタバース開発	137	3.0倍
9	キャディ	図面管理サービス	133	35%
10	SUPER STUDIO	EC事業者向け業務管理ソフト	124	63%
10	RevComm	電話商談の解析システム	124	99%
12	カケハシ	調剤薬局向けシステム	119	50%
13	jinjer	人事管理クラウド	113	34%
14	ニーリー	駐車場の管理システム	109	3.3倍
15	インフキュリオン	決済サービス	92	42%
16	Ubie	AIによる問診システム	91	49%
17	アスエネ	温暖化ガス排出量の算定ソフト	88	7.3倍
18	モノグサ	学習アプリ	84	2.5倍
19	ティアフォー	自動運転ソフトウエア	83	47%
19	ゼロボード	温暖化ガス排出量の可視化ソフト	83	6.9倍

（注）人数は2024年3月時点。23年10月時点の推計企業価値が50億円以上の企業。
厚生労働省のデータを基にケップルが算出

Ruby Sponsors
Timee
aisaac
KOMOJU
WED
IVRy
MEDLEY
STORES
mov
shopify
Platinum Sponsors
esm
Helpfeel
ANDPAD
Leaner
pixiv
Findy
ZOZO
fastly
DeNA
TWOGATE
giftee
VONAGE
食べログ
mybest
Flatt
hacomono
MedPeer
MIXI
techouse
SMS
freee
Wantedly
TokyoDev
CROWD WORKS
KIBIDRAFT
BEING
HireRoo
note
stmn, inc.
SimpleForm
Gold Sponsors
NaCl
timee
CHECKWARE
studist
TimeTree
code Takt
Fusic
Coincheck
RECRUIT
RAKSUL
X Mile
MNTSQ
OPTiM
Silver Sponsors
esa
GLOBIS
RubyKaigi 2024 Sponsors


HIRING!
Gyazo
インターン募集中!
Ruby on Rails
HIRING!
TableCheck
WANTED!
workbrew.com
HIRING!
エースチャイルド
HIRING!
WANTED!
TimeTree
Rails エンジニア大募集!
WANTED!
HIRING!
AskOne
募集中!
HIRING!
Techouse
HIRING!
HIRING!
エンジニア
アルバイト
インターン募集中
pixiv
HIRING!
WANTED!
スタメン
HIRING!
Money Forward
Ruby Engineer
wanted!!
HIRING!
募集中!
HIRING!
RAKSUL
エンジニア
募集中
HIRING!
RIZAP
エンジニア
募集中!
WANTED
WANTED!
SmartHR
2024

Efficient

Ruby **was** Slow

Ruby **is** Fast (Enough)

- GitHub
- Shopify
- Square (Block)
- ...

Ruby is Good

Ruby is Great

We are Greedy

Can we Make Ruby Better?

Is it Possible?

Yes

But How?

4 Aspects

1. Performance

Performance is Important

Everyone Loves Faster Languages

Everyone Loves Benchmarks

- YARV
- MJIT
- YJIT

YARV (2007)

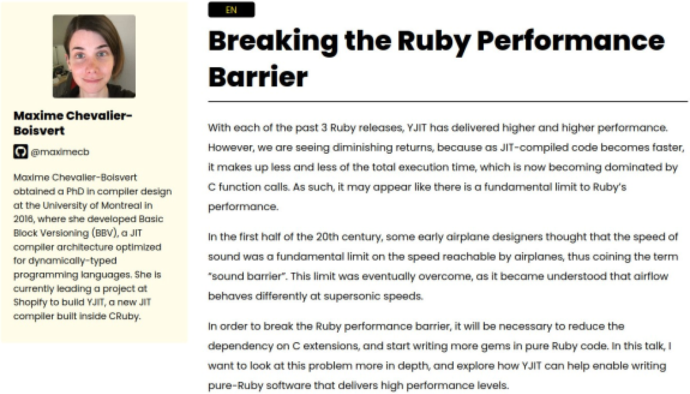
- Bytecode VM
- Faster than Tree-Walking Interpreter
- 5-50 times Faster

MJIT (2018)

- Ruby3x3 (2014)
- 3 times Faster than Ruby2.0
- With some benchmarks (OptCarrot)
- Not with Rails apps

YJIT (2022)

- Faster JIT
- Basic Block Versioning
- Written in Rust
- Rails apps run 1.8x Faster
- Thanks to Shopify



Maxime Chevalier-Boisvert

@maximecb

Maxime Chevalier-Boisvert obtained a PhD in compiler design at the University of Montreal in 2016, where she developed Basic Block Versioning (BBV), a JIT compiler architecture optimized for dynamically-typed programming languages. She is currently leading a project at Shopify to build YJIT, a new JIT compiler built inside CRuby.

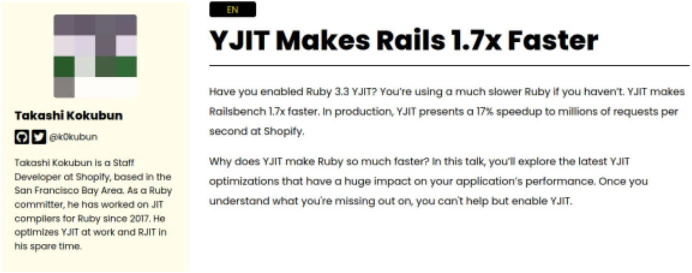
EN

Breaking the Ruby Performance Barrier

With each of the past 3 Ruby releases, YJIT has delivered higher and higher performance. However, we are seeing diminishing returns, because as JIT-compiled code becomes faster, it makes up less and less of the total execution time, which is now becoming dominated by C function calls. As such, it may appear like there is a fundamental limit to Ruby's performance.

In the first half of the 20th century, some early airplane designers thought that the speed of sound was a fundamental limit on the speed reachable by airplanes, thus coining the term "sound barrier". This limit was eventually overcome, as it became understood that airflow behaves differently at supersonic speeds.

In order to break the Ruby performance barrier, it will be necessary to reduce the dependency on C extensions, and start writing more gems in pure Ruby code. In this talk, I want to look at this problem more in depth, and explore how YJIT can help enable writing pure-Ruby software that delivers high performance levels.



YJIT Makes Rails 1.7x Faster

Have you enabled Ruby 3.3 YJIT? You're using a much slower Ruby if you haven't. YJIT makes Railsbench 1.7x faster. In production, YJIT presents a 17% speedup to millions of requests per second at Shopify.

Why does YJIT make Ruby so much faster? In this talk, you'll explore the latest YJIT optimizations that have a huge impact on your application's performance. Once you understand what you're missing out on, you can't help but enable YJIT.

Takashi Kokubun

@k0kubun

Takashi Kokubun is a Staff Developer at Shopify, based in the San Francisco Bay Area. As a Ruby committer, he has worked on JIT compilers for Ruby since 2017. He optimizes YJIT at work and RJIT in his spare time.

Further Performance Improvement Planned

Performance Heals Every Issue

Make Ruby VM Faster

Make Ruby Greater

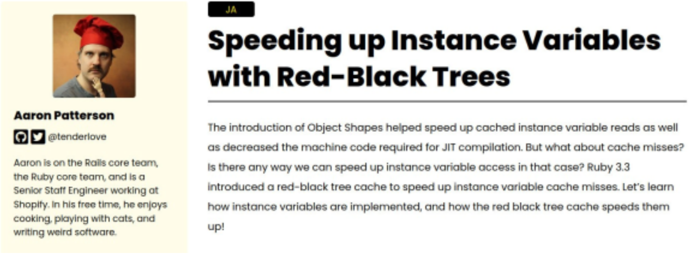
2. Performance

Performance Heals Every Issue

VM is not the only bottleneck

Memory Management

- Object Heap Compaction
- Variable Width Allocation
- Object Shapes
- GC Improvements



JA

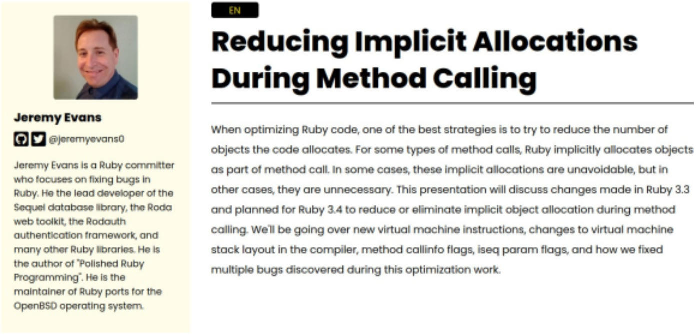
Speeding up Instance Variables with Red-Black Trees

The introduction of Object Shapes helped speed up cached instance variable reads as well as decreased the machine code required for JIT compilation. But what about cache misses? Is there any way we can speed up instance variable access in that case? Ruby 3.3 introduced a red-black tree cache to speed up instance variable cache misses. Let's learn how instance variables are implemented, and how the red black tree cache speeds them up!

Aaron Patterson

@tenderlove

Aaron is on the Rails core team, the Ruby core team, and is a Senior Staff Engineer working at Shopify. In his free time, he enjoys cooking, playing with cats, and writing weird software.



EN

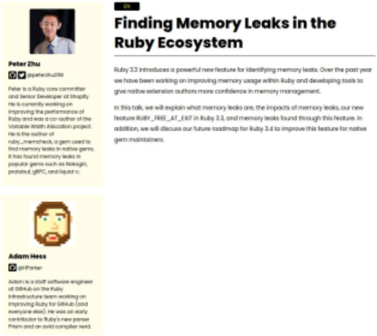
Reducing Implicit Allocations During Method Calling

Jeremy Evans

@jeremyevans0

Jeremy Evans is a Ruby committer who focuses on fixing bugs in Ruby. He the lead developer of the Sequel database library, the Roda web toolkit, the Rodauth authentication framework, and many other Ruby libraries. He is the author of "Polished Ruby Programming". He is the maintainer of Ruby ports for the OpenBSD operating system.

When optimizing Ruby code, one of the best strategies is to try to reduce the number of objects the code allocates. For some types of method calls, Ruby implicitly allocates objects as part of method call. In some cases, these implicit allocations are unavoidable, but in other cases, they are unnecessary. This presentation will discuss changes made in Ruby 3.3 and planned for Ruby 3.4 to reduce or eliminate implicit object allocation during method calling. We'll be going over new virtual machine instructions, changes to virtual machine stack layout in the compiler, method callinfo flags, iseq param flags, and how we fixed multiple bugs discovered during this optimization work.



EN

Finding Memory Leaks in the Ruby Ecosystem

Ruby 3.3 introduces a powerful new feature for identifying memory leaks. Over the past year we have been working on improving memory usage within Ruby and developing tools to give native extension authors more confidence in memory management.

In this talk, we will explain what memory leaks are, the impacts of memory leaks, our new feature RUBY_FREE_AT_EXIT in Ruby 3.3, and memory leaks found through this feature. In addition, we will discuss our future roadmap for Ruby 3.4 to improve this feature for native gem maintainers.

Peter Zhu

@peterzhu2118

Peter is a Ruby core committer and Senior Developer at Shopify. He is currently working on improving the performance of Ruby and was a co-author of the Variable Width Allocation project. He is the author of ruby_memcheck, a gem used to find memory leaks in native gems. It has found memory leaks in popular gems such as Nokogiri, protobuf, gRPC, and liquid-c.

Adam Hess

@HParker

Adam is a staff software engineer at GitHub on the Ruby Infrastructure team working on improving Ruby for GitHub (and everyone else). He was an early contributor to Ruby's new parser Prism and an avid compiler nerd.

Memory is Expensive

We Need More Memory

We Should Reduce Memory (for VM)

If Ruby use Less Memory

We can Save tons of Money

Make Ruby use Less Memory

Make Ruby Greater

3. Performance

I could not predict Multi-Core Age

Concurrency for System Architecture

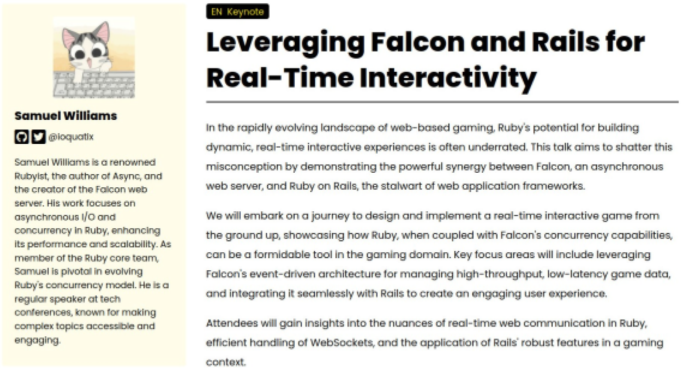
Concurrency for Performance

- Threads
- GVL
- Processes
- Fiber (for I/O)
- Ractors (for CPU)

- NxM Threads
- Lightweight Ractors
- Ractor local GC
- Async Fibers

Make Ruby use More Concurrency

Make Ruby Greater



EN Keynote

Leveraging Falcon and Rails for Real-Time Interactivity

In the rapidly evolving landscape of web-based gaming, Ruby's potential for building dynamic, real-time interactive experiences is often underrated. This talk aims to shatter this misconception by demonstrating the powerful synergy between Falcon, an asynchronous web server, and Ruby on Rails, the stalwart of web application frameworks.

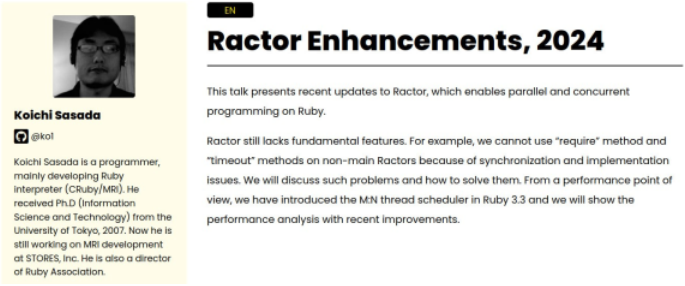
We will embark on a journey to design and implement a real-time interactive game from the ground up, showcasing how Ruby, when coupled with Falcon's concurrency capabilities, can be a formidable tool in the gaming domain. Key focus areas will include leveraging Falcon's event-driven architecture for managing high-throughput, low-latency game data, and integrating it seamlessly with Rails to create an engaging user experience.

Attendees will gain insights into the nuances of real-time web communication in Ruby, efficient handling of WebSockets, and the application of Rails' robust features in a gaming context.

Samuel Williams

@ioquatix

Samuel Williams is a renowned Rubyist, the author of Async, and the creator of the Falcon web server. His work focuses on asynchronous I/O and concurrency in Ruby, enhancing its performance and scalability. As member of the Ruby core team, Samuel is pivotal in evolving Ruby's concurrency model. He is a regular speaker at tech conferences, known for making complex topics accessible and engaging.



EN

Ractor Enhancements, 2024

This talk presents recent updates to Ractor, which enables parallel and concurrent programming on Ruby.

Ractor still lacks fundamental features. For example, we cannot use “require” method and “timeout” methods on non-main Ractors because of synchronization and implementation issues. We will discuss such problems and how to solve them. From a performance point of view, we have introduced the M:N thread scheduler in Ruby 3.3 and we will show the performance analysis with recent improvements.

Koichi Sasada

@ko1

Koichi Sasada is a programmer, mainly developing Ruby interpreter (CRuby/MRI). He received Ph.D (Information Science and Technology) from the University of Tokyo, 2007. Now he is still working on MRI development at STORES, Inc. He is also a director of Ruby Association.

4. Performance

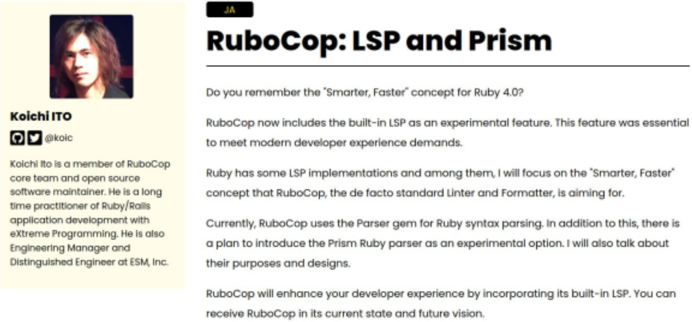
Software Performance is Important

Developer Performance is More Important

Ruby Programming is Fun

Better Experience by Tools Support

- Ruby-LSP
- Rubocop
- Steep
- Copilot
- ...



JA

RuboCop: LSP and Prism

Koichi ITO

@koic

Koichi Ito is a member of RuboCop core team and open source software maintainer. He is a long time practitioner of Ruby/Rails application development with eXtreme Programming. He is also Engineering Manager and Distinguished Engineer at ESM, Inc.

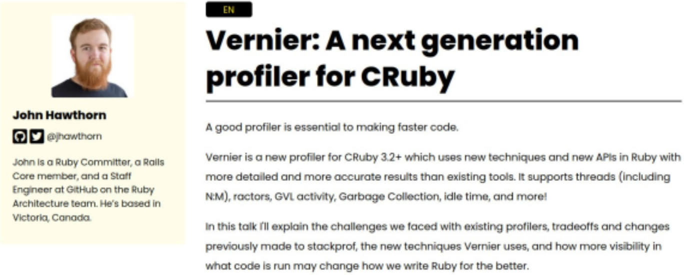
Do you remember the "Smarter, Faster" concept for Ruby 4.0?

RuboCop now includes the built-in LSP as an experimental feature. This feature was essential to meet modern developer experience demands.

Ruby has some LSP implementations and among them, I will focus on the "Smarter, Faster" concept that RuboCop, the de facto standard Linter and Formatter, is aiming for.

Currently, RuboCop uses the Parser gem for Ruby syntax parsing. In addition to this, there is a plan to introduce the Prism Ruby parser as an experimental option. I will also talk about their purposes and designs.

RuboCop will enhance your developer experience by incorporating its built-in LSP. You can receive RuboCop in its current state and future vision.



EN

Vernier: A next generation profiler for CRuby

A good profiler is essential to making faster code.

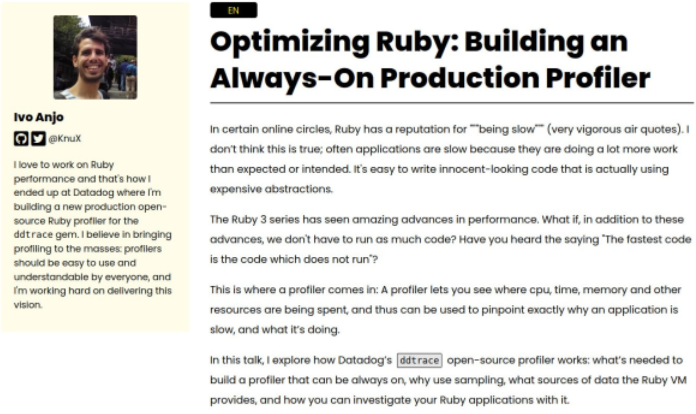
Vernier is a new profiler for CRuby 3.2+ which uses new techniques and new APIs in Ruby with more detailed and more accurate results than existing tools. It supports threads (including N:M), ractors, GVL activity, Garbage Collection, idle time, and more!

In this talk I'll explain the challenges we faced with existing profilers, tradeoffs and changes previously made to stackprof, the new techniques Vernier uses, and how more visibility in what code is run may change how we write Ruby for the better.

John Hawthorn

@jhawthorn

John is a Ruby Committer, a Rails Core member, and a Staff Engineer at GitHub on the Ruby Architecture team. He's based in Victoria, Canada.



EN

Optimizing Ruby: Building an Always-On Production Profiler

Ivo Anjo

@KnuX

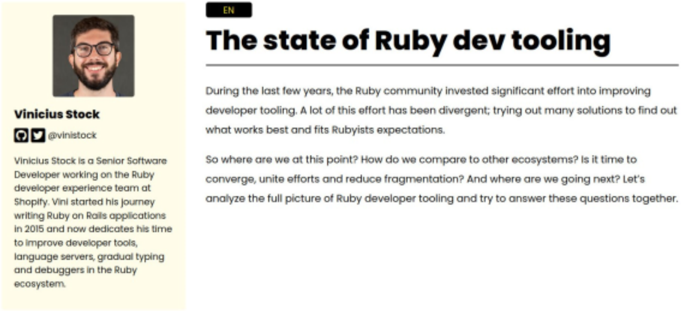
I love to work on Ruby performance and that's how I ended up at Datadog where I'm building a new production open-source Ruby profiler for the `ddtrace` gem. I believe in bringing profiling to the masses: profilers should be easy to use and understandable by everyone, and I'm working hard on delivering this vision.

In certain online circles, Ruby has a reputation for ""being slow"" (very vigorous air quotes). I don't think this is true; often applications are slow because they are doing a lot more work than expected or intended. It's easy to write innocent-looking code that is actually using expensive abstractions.

The Ruby 3 series has seen amazing advances in performance. What if, in addition to these advances, we don't have to run as much code? Have you heard the saying "The fastest code is the code which does not run"?

This is where a profiler comes in: A profiler lets you see where cpu, time, memory and other resources are being spent, and thus can be used to pinpoint exactly why an application is slow, and what it's doing.

In this talk, I explore how Datadog's `ddtrace` open-source profiler works: what's needed to build a profiler that can be always on, why use sampling, what sources of data the Ruby VM provides, and how you can investigate your Ruby applications with it.



EN

The state of Ruby dev tooling

Vinicius Stock

@vinistock

Vinicius Stock is a Senior Software Developer working on the Ruby developer experience team at Shopify. Vini started his journey writing Ruby on Rails applications in 2015 and now dedicates his time to improve developer tools, language servers, gradual typing and debuggers in the Ruby ecosystem.

During the last few years, the Ruby community invested significant effort into improving developer tooling. A lot of this effort has been divergent; trying out many solutions to find out what works best and fits Rubyists expectations.

So where are we at this point? How do we compare to other ecosystems? Is it time to converge, unite efforts and reduce fragmentation? And where are we going next? Let's analyze the full picture of Ruby developer tooling and try to answer these questions together.

Need Better Parser

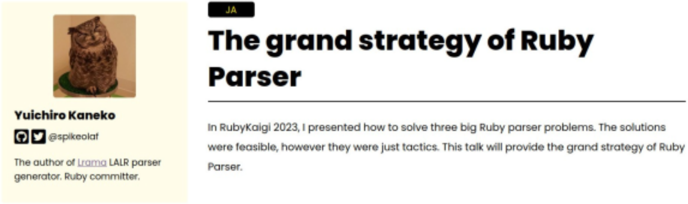
- parser gem
- ripper
- ...

We need the Universal Parser

Prism

- Prism (kddnewton)
- Parser by Lrama (yui-knk)

Sound Competition



JA

The grand strategy of Ruby Parser

In RubyKaigi 2023, I presented how to solve three big Ruby parser problems. The solutions were feasible, however they were just tactics. This talk will provide the grand strategy of Ruby Parser.

Yuichiro Kaneko

@spikeolaf

The author of Lrama LALR parser generator. Ruby committer.

API will be based on Prism

Including AST (fundamental)

Prism will be Prism forever

The Core might be based on Lrama

- Hand-written Parser
- Parser from Parser Generator

Syntax Moratorium

We will Keep the current syntax

for at least a year

or probably 2,3 years

Except for Bug fixes & Clarification

To Give Both Parsers Equal Chance

Better Tooling Improves Productivity

Tools are out of Core Team's Scope

We need Community

We need to Lengthen our Stride

- Ruby Association Grant
- Google Summer of Code
- Independent Community Effort
- Conferences

Ruby Community

Together, We can be Stronger

Together, We can make Ruby Greater

The Power of Ruby Community

In Addition:

The Future of Ruby

Ruby4.0

“Namespace, What and Why”

The Missing Piece

Ruby2 (2004)

not Ruby2.0 (2013)

I once tried to restart Ruby

Just like Perl6 or Python3000

It turned out to be a bad idea

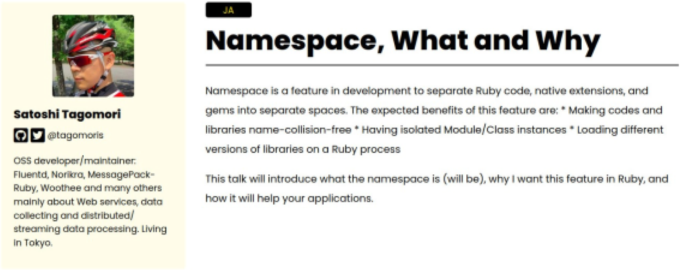
Ruby2 Ideas

- Selector Namespace
- Keyword Arguments
- Method Combination
- Unicode Support
- Pattern Match
- Packages
- JIT Compiler

- Refinement (2.0)
- Real Keyword Arguments (3.0)
- Method Combination (2.0)
- Unicode Support (1.9)
- Pattern Match (2.7)
- Packages
- JIT (2.6)

- Selector Namespace
- Packages

Namespace Separation



Satoshi Tagomori

@tagomoris

OSS developer/maintainer: Fluentd, Norikra, MessagePack-Ruby, Woothee and many others mainly about Web services, data collecting and distributed/ streaming data processing. Living in Tokyo.

JA

Namespace, What and Why

Namespace is a feature in development to separate Ruby code, native extensions, and gems into separate spaces. The expected benefits of this feature are: * Making codes and libraries name-collision-free * Having isolated Module/Class instances * Loading different versions of libraries on a Ruby process

This talk will introduce what the namespace is (will be), why I want this feature in Ruby, and how it will help your applications.

One More Thing

Dream Story

SDGs

Sustainable Development Goals

GX

Green Transformation

Ruby with less Comsuming

Memory & Performance

Single Binary

AOT Compiler

- Type Profiling
- Type Signatures
- Profile Guided Compilation

Sponsored by
NaCl

Sponsored by
OSS Vision

Sponsored by
GitHub Sponsors

Shopify
JetBrain [NEW]

Sponsored by

Ruby Community

Thank you