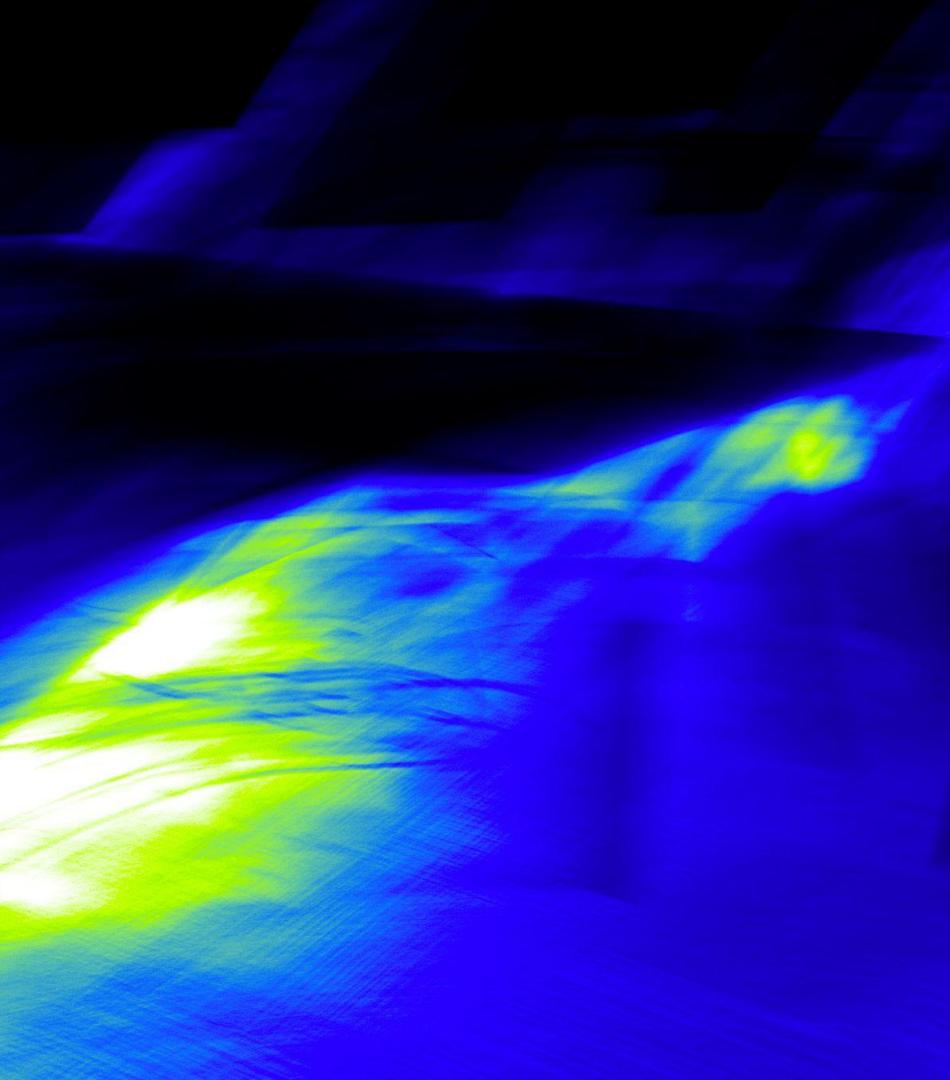


# なぜRubyを わざわざ作り直すのか

笹田耕一  
テクノロジー部門 / 技術推進本部 / Ruby Developmentチーム

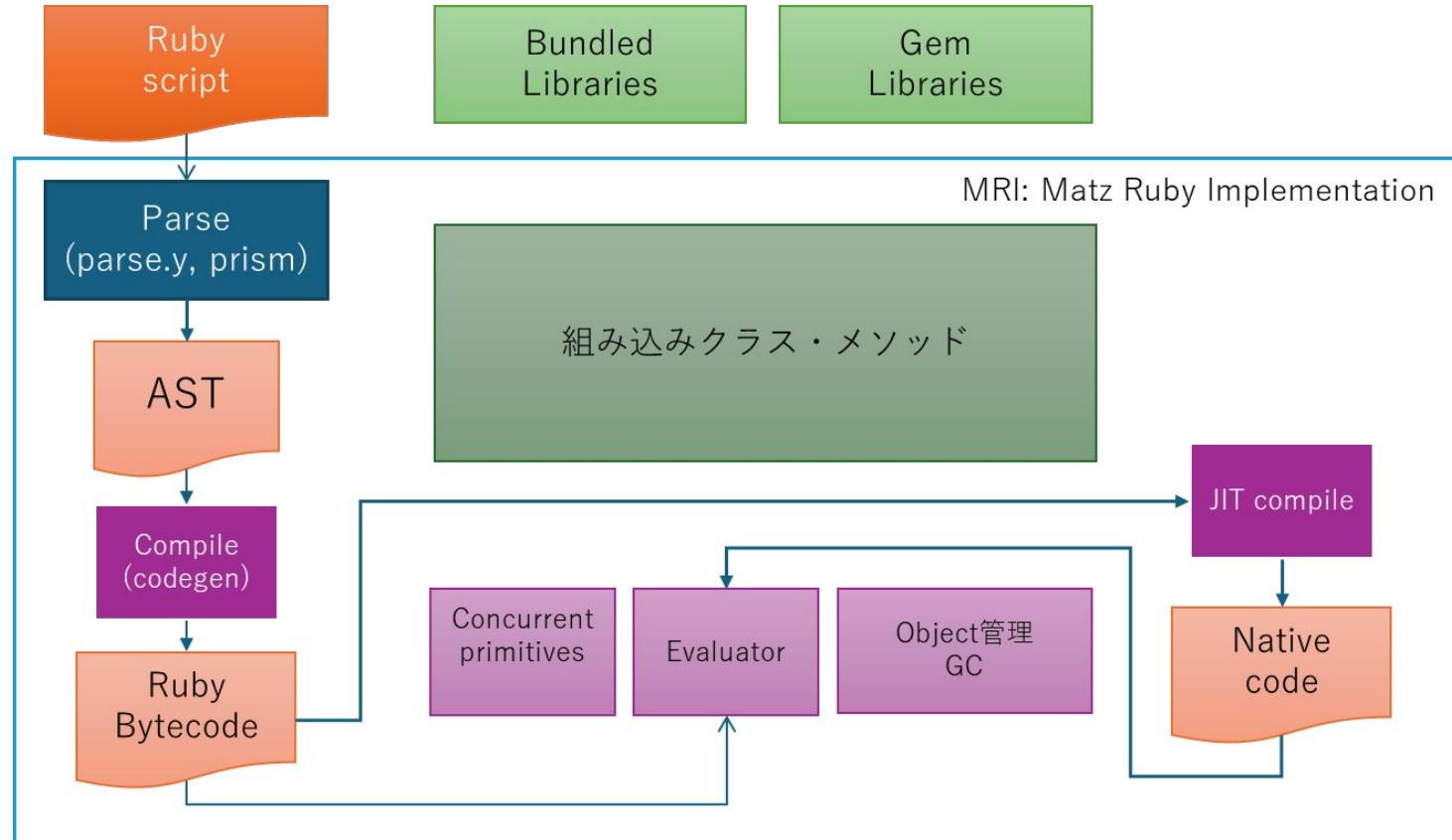


# Just for Fun!

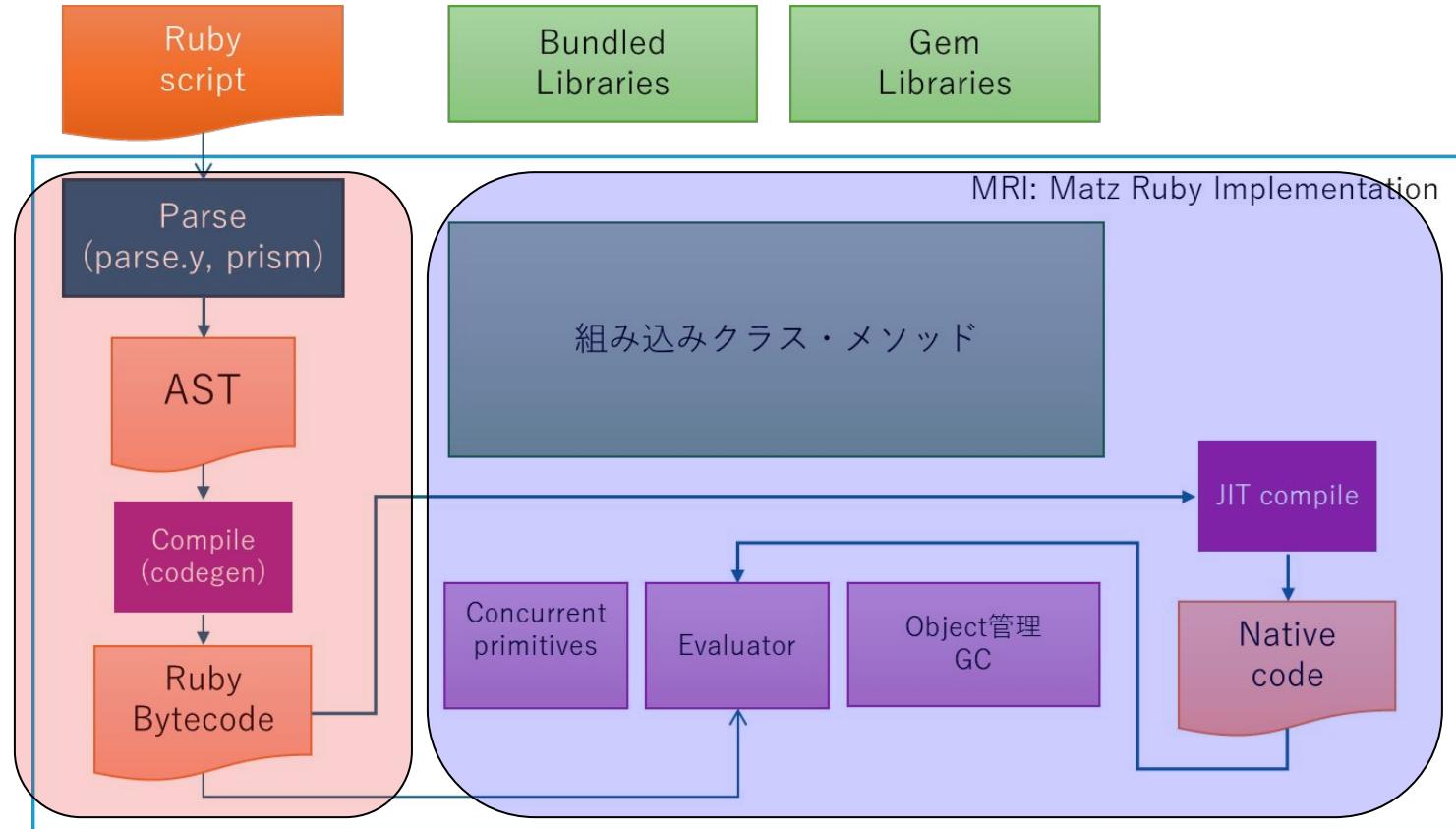
こだわりや情熱、たのしみによって駆  
動される経済をつくる  
(STORES ミッション)

// おわり

# Ruby internal



# Ruby internal



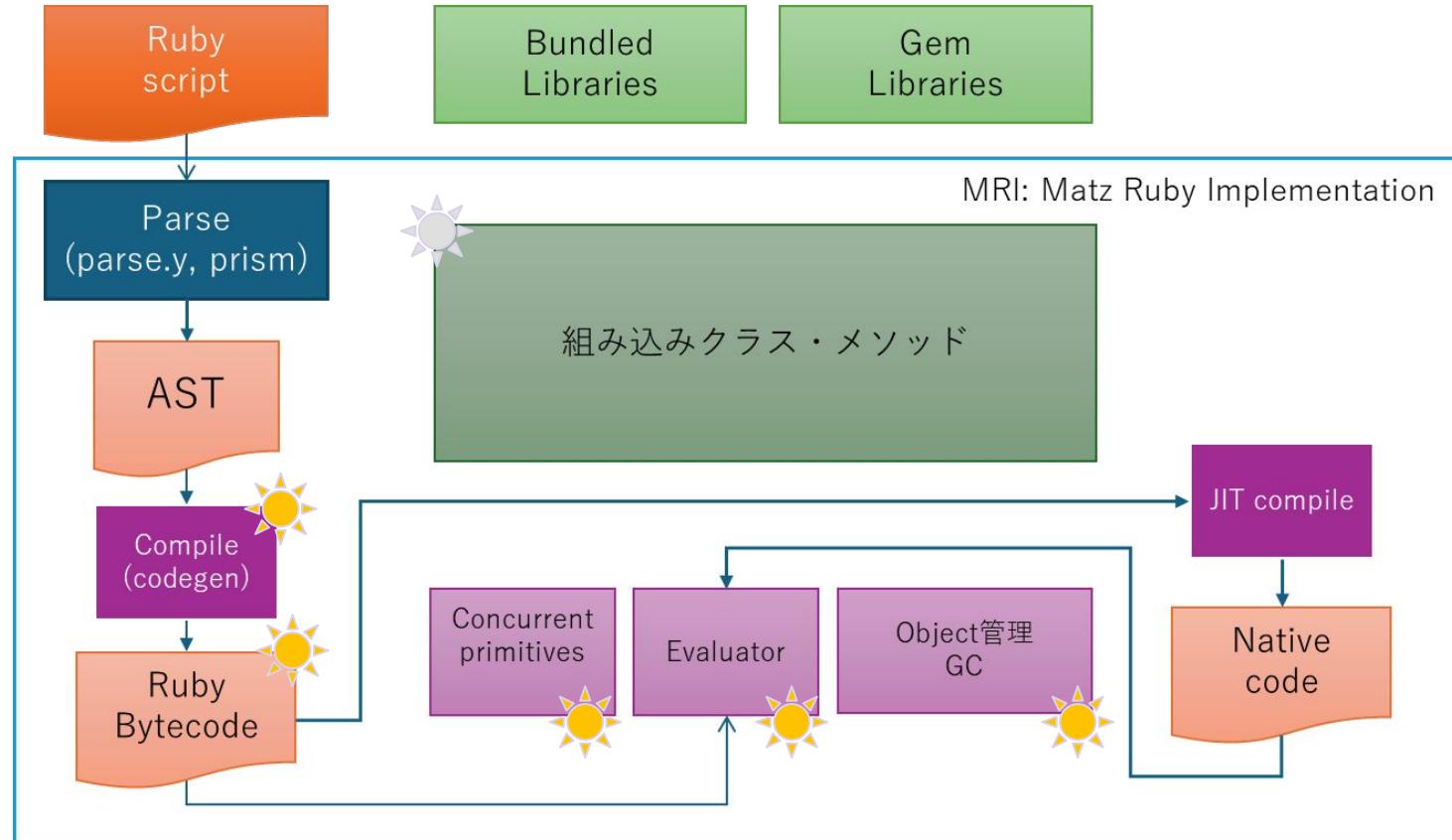
よみこむ

うごかす

What would you do?  
STORES Tech Conf 2025

- Ruby コミッター(2007～), STORES(2023～)
- 主に低レイヤの仕組み
  - 仮想マシン(入れた)
  - ガーベージコレクタ(変えた)
  - 並行・並列制御プリミティブ
    - スレッド(変えた)
    - Fiber(入れた)
    - Ractor(入れた)

# Ruby internal とわたし



## なぜソフトウェアを作り直すのか？「ぼくのかんがえたさいきょうの XXX」

- バグの修正
- 新機能が欲しい
- 高性能なソフトウェアが欲しい
  - 速くしてほしい
  - 省リソースにしてほしい
    - 省メモリ
    - 省電力

## なぜ Ruby を作り直すのか？「ぼくのかんがえたさいきょうの Ruby」

- バグの修正 ← だいたい小修整
- 新機能が欲しい ← 小修整、時々大修整
- 高性能なソフトウェアが欲しい ← 結構大修整
  - 速くしてほしい
  - 省リソースにしてほしい
    - 省メモリ
    - 省電力

成果を計りやすいので好き

## Ruby インタプリタの歴史1(~2007)

「今では当たり前」の機能が当たり前に動いていてスゴイ

- 1993年開発開始、1995年リリース(30周年！)

- ノードを辿る評価器

- 例外処理

- スレッド

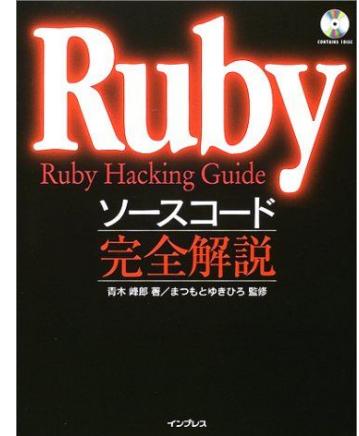
- 外部ライブラリロード

- オブジェクト指向機能

- メモリ管理、ガーベージコレクション(M&S)

- 豊富な組み込みクラス・メソッド

- 組み込む方法の設計



## Ruby インタプリタの歴史2(2007~2020)

- 2007年 エンコーディングの導入(追加)
- 2007年 仮想マシンYARVの導入(わたし)
  - バイトコードインタプリタで評価器を書き直し
    - 例外処理(置き換え)
    - スレッド(置き換え)
    - Fiber(追加)
- 2012年 GC の改善(わたし)
  - 世代別GC(追加)
  - 段階的GC(追加)

下回りをやっていると、  
全体を俯瞰できるので、  
挑戦できことが多い

## Ruby インタプリタの歴史3(2020~)

- 2020年 GC の改善
  - Variable Width Allocation (VWA)
- 2020年 Ractor の導入(わたし)
  - 2023年 M:Nスレッドの導入
  - 2025年 Ractor local GC の実装(中)
- 2021年 Ruby 3.1 YJIT の導入

## 意外と変わっていない Ruby インタプリタ

基礎がしっかりしていたのが発展のキモ？  
まつもとさんのかだわりと情熱？

### 評価器

ノードを辿るインタプリタ → 仮想マシン

+ MJIT → YJIT → ZJIT

### メモリ管理・GC

Mark & Sweep

+ 世代別、段階的GC

+ VWA + ModularGC

+ Ractor local GC(今開発中)

### 並行制御

+ 1:N スレッド → 1:1 スレッド → M:N スレッド

+ Fiber, Fiber scheduler

+ Ractor

### オブジェクト指向機能(あんまり変わらず)

### 豊富な組み込みクラス・メソッド(あんまり変わらず)

## 自慢 仮想マシンの導入(Ruby 1.9~)

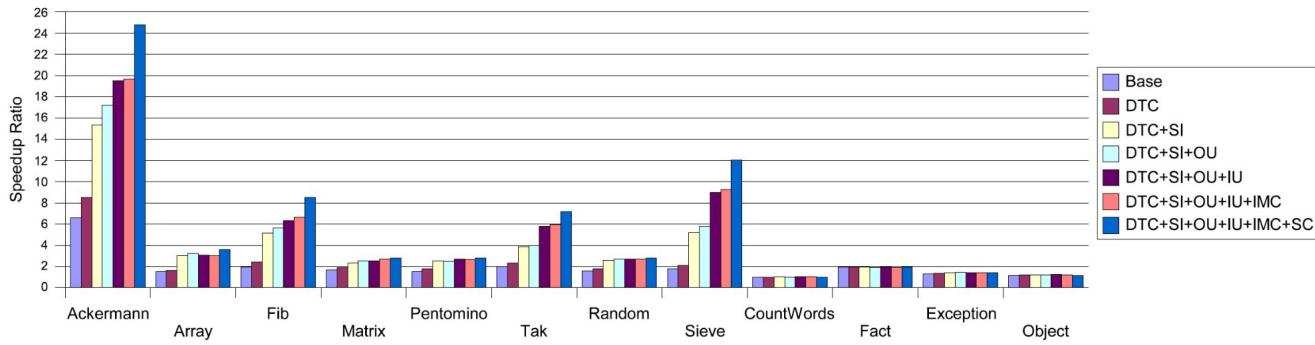


図 13 x86\_64 プロセッサ上でのベンチマークプログラムの評価

“Ruby 用仮想マシンYARVの実装と評価”, 2025

「Rubyって遅い」という評価に対する挑戦  
やれば速くなることはわかっていた。  
でも、意外とわからないことが多かった。  
考えるのが楽しい。  
メモリアクセスを一個でも減らそうと悪戦苦闘。

## 自慢 世代別GCとかの改善

- **Unprotected object** を用いた世代別GCの導入

**Table 1.** RDoc benchmark results

	Total time (sec)	GC time (sec)
Disabled	30.46	10.20
Enabled	22.57	1.63

“GradualWrite-Barrier Insertion into a Ruby Interpreter” (2019)

「RubyのGCって遅い」という評価に対する挑戦  
既存処理系を諦める以外、誰も解決策を知らなかった。  
誰も知らないことを明らかにするのが楽しい。  
本当に動くのか最後までよくわからなかつた。

## 自慢 Ractor の導入

```
def tarai(x, y, z) =
  x <= y ? y : tarai(tarai(x-1, y, z),
                      tarai(y-1, z, x),
                      tarai(z-1, x, y))

require 'benchmark'
Benchmark.bm do |x|
  # sequential version
  x.report('seq'){ 4.times{ tarai(14, 7, 0) } }

  # parallel version
  x.report('par'){
    4.times.map do
      Ractor.new { tarai(14, 7, 0) }
    end.each(&:take)
  }
end
```

```
Benchmark result:
      user      system      total      real
seq  64.560736  0.001101  64.561837 ( 64.562194)
par  66.422010  0.015999  66.438009 ( 16.685797)
```

結果は Ubuntu 20.04, Intel(R) Core(TM) i7-6700 (4 cores, 8 hardware threads)  
で実行したものになります。逐次実行したときよりも、並列化によって3.87倍の高  
速化していることがわかります。

<https://www.ruby-lang.org/ja/news/2020/12/25/ruby-3-0-0-released/>

## 新しい言語デザインの挑戦

MutableとImmutableの共存に挑戦  
言語として、あんまりない

世界観をデザインするのが楽しい  
実装がむっちゃ大変なのが(楽し辛)い

## 今、Ractor local GC (が、どうかな...?)

こだわりを実現できて、楽しい

こだわって、高性能省資源を追及して、楽しい

こだわって、新技術の発明を追及して、楽しい

こだわって、新しい世界観を追及して、楽しい

人ができなかつたこと、思いもつかなかつたことを発見できると面白い。  
難しいコーディングが完成できたら鼻が高い。バグが取れると清々しい。  
たくさんのみんなが使ってくれると誇らしい。褒めてもらえると嬉しい。

こだわりを妥協せずに

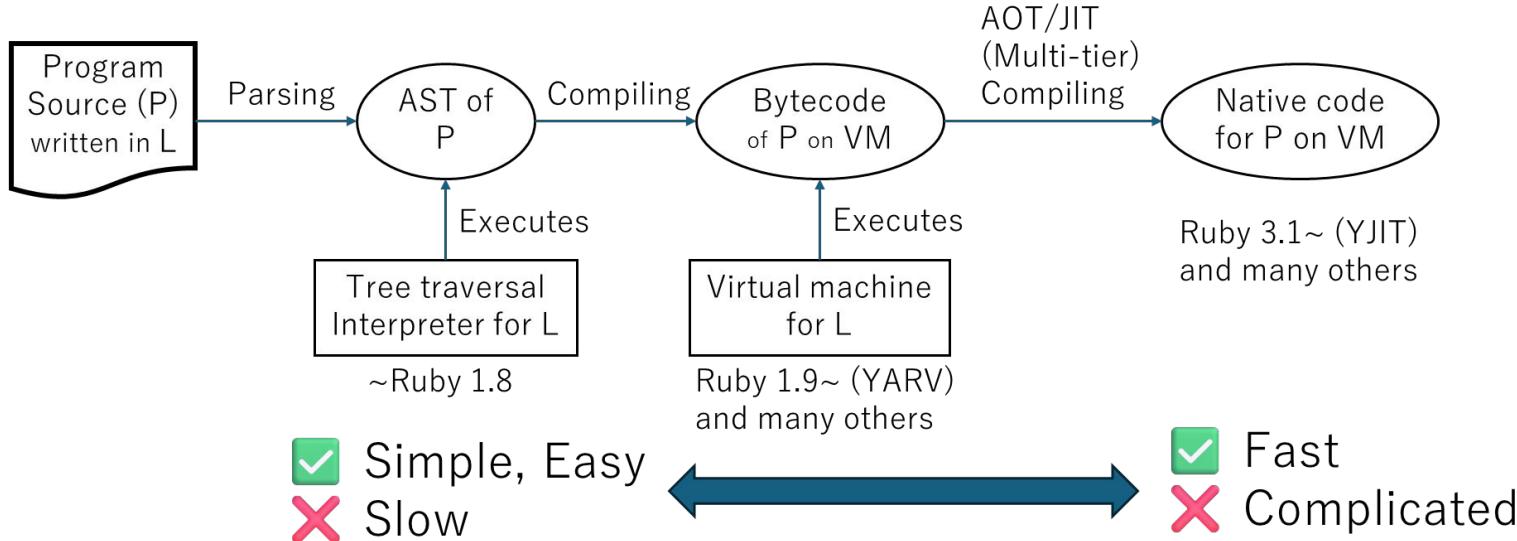
突き詰めると、楽しい

# 次の挑戦

ASTro: AST-based  
reusable optimization framework  
新しいインタプリタ開発手法の提案

こだわり: 絶対に楽をしたい

## Motivation (1) Implementation complexity



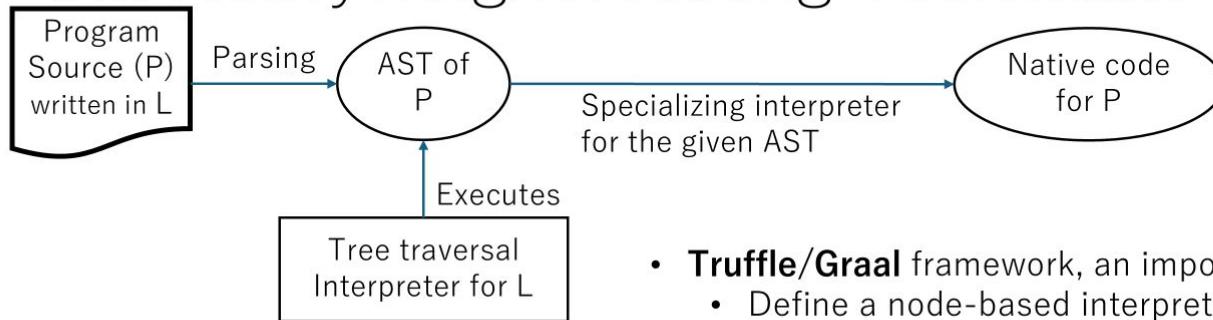
? Is it possible to achieve both speed and simplicity?  
Disclosure: I dislike assembly programming.

“ASTro: An AST-Based Reusable Optimization Framework” (2025)

## 新しいインタープリタ開発手法(既知の技術への不満)

部分評価ってのが  
楽だけど実装が大変...

## Motivation (2) Partial Evaluation (PE), but Heavyweight Existing Toolchains



- Simple, Easy and Fast!!
- Heavyweight Framework  
Hard to make and modify

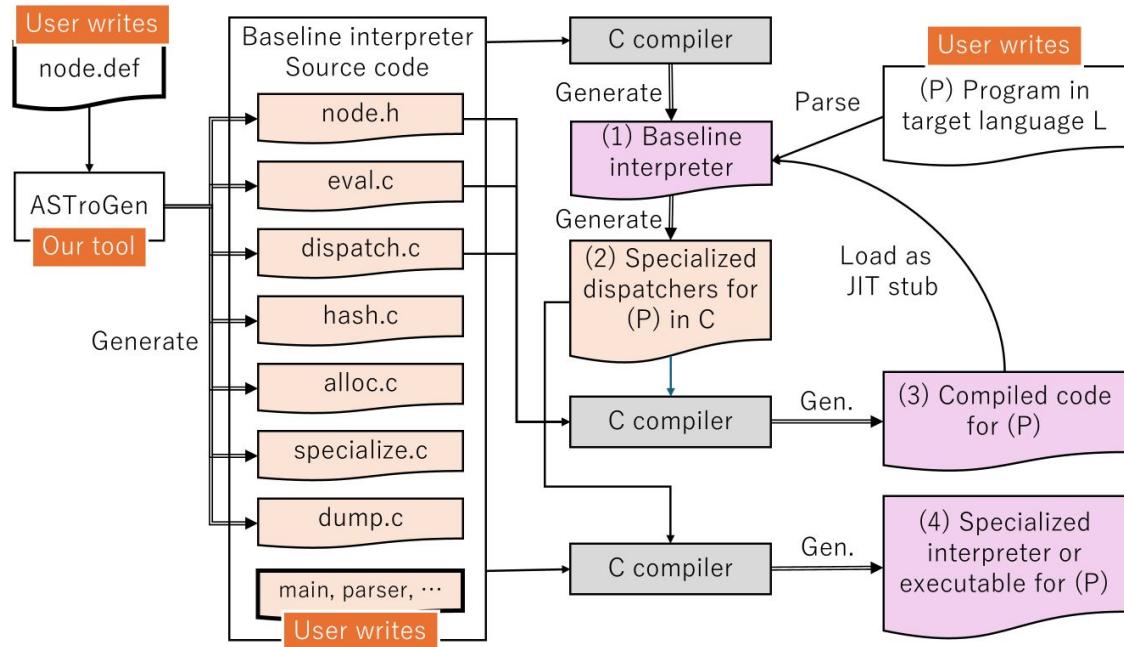
- **Truffle/Graal** framework, an important system:
  - Define a node-based interpreter in **Truffle SDL**
  - Profile given node execution
  - Specialize (PE) the given AST and interpreter using **Truffle framework** and generate native code with **Graal compiler**

## ? Can we make a lightweight PE framework?

Do you know of a tool that can generate highly optimized native code for most of target architectures and is widely available and easy to use?

“ASTro: An AST-Based Reusable Optimization Framework” (2025)

# ASTro: AST-based reusable optimization Framework



部分評価ベースの  
高速インタプリタ構成法  
with C コンパイラ

- ## Key ideas
- うまい木の表現
  - ハッシュを用いたID

## 簡単な評価

x86\_64

	loop	fib	call	prime_count
naruby	naruby/interpret	<b>0.786</b>	<b>4.870</b>	6.760
	naruby/compiled	<b>0.001</b>	<b>1.093</b>	<b>3.435</b>
	naruby/pg	0.001	<b>1.143</b>	<b>2.061</b>
C	gcc/-O0	0.042	0.480	<b>0.490</b>
	gcc/-O1	0.023	0.400	<b>0.005</b>
	gcc/-O2	0.001	0.115	0.318

RISC-V

	loop	fib	call	prime_count
naruby	naruby/interpret	<b>8.096</b>	<b>45.580</b>	62.268
	naruby/compiled	<b>0.003</b>	<b>6.834</b>	<b>17.657</b>
	naruby/pg	0.003	<b>7.053</b>	<b>8.566</b>
C	gcc/-O0	0.544	5.481	8.385
	gcc/-O1	0.061	1.132	3.151
	gcc/-O2	0.002	0.604	1.575

naruby は Not a Ruby  
(むっちゃ単純化した Ruby)  
小さい言語には効果ありそう  
大きな言語(Ruby)には？？

- In seconds, lower is better
- Measuring while process execution
  - Including AST building, compiled code fetching and execution
  - Excluding native code compiling time

なんでわざわざ新しいものを作るの？

- 既存の常識を覆すのが楽しい

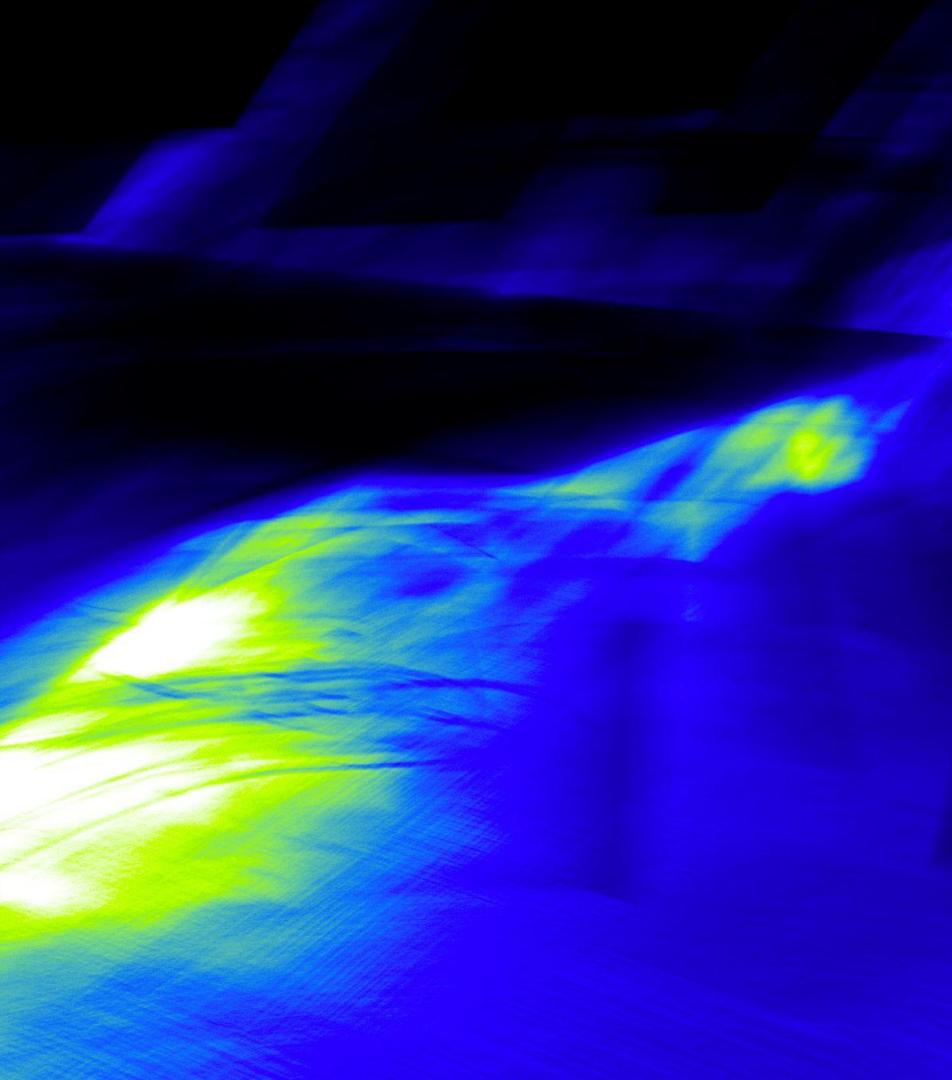
(覆すことができる保証はないんだけど)

自分のこだわりをもって、  
新しいことに挑戦できるのが楽しい

開発生産性の向上品質の向上(バグ低減)

Cコンパイラによる高い性能

コンパイル結果共有による高速起動



なぜRubyを  
わざわざ作り直すのか

# Just for Fun!

こだわりや情熱、たのしみによって  
駆動される経済をつくる  
(STORES ミッション)

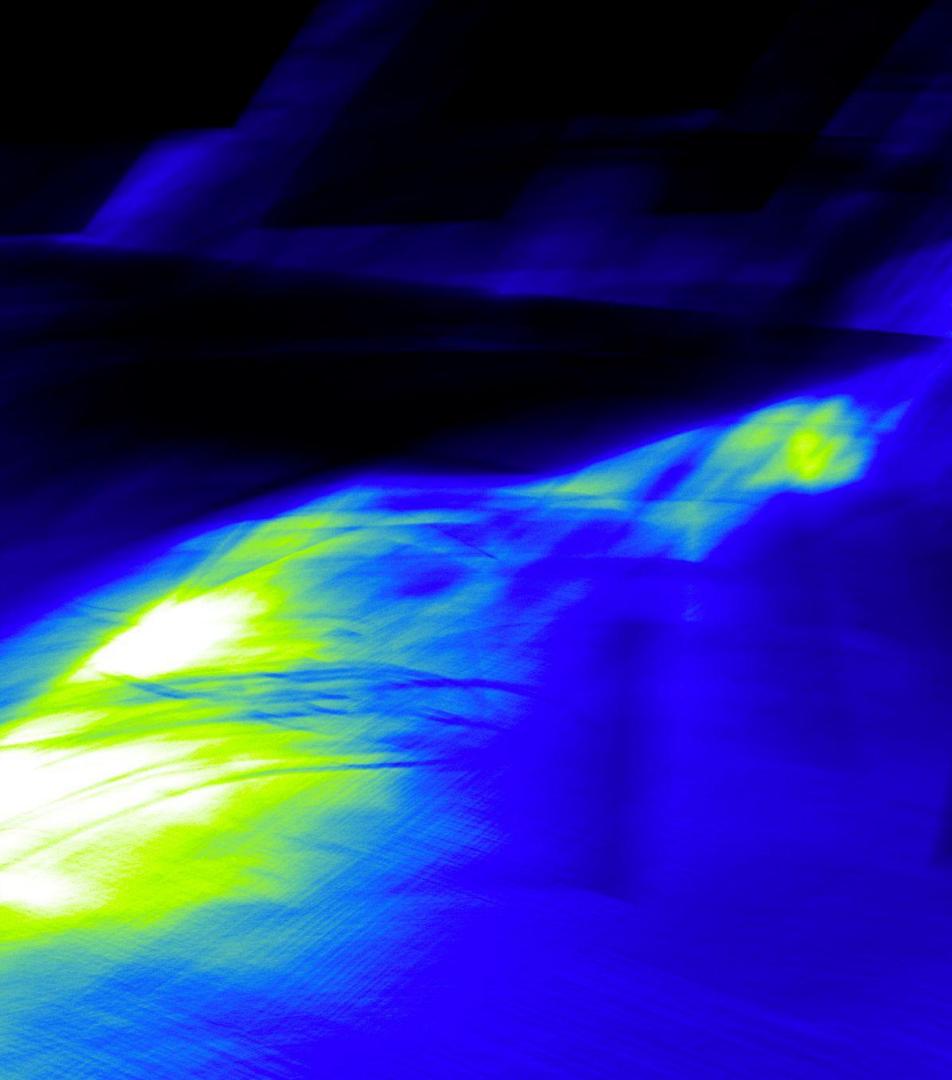
こだわりや情熱、たのしみによって  
駆動される Ruby をつくる  
(われわれのミッション)

// Thank you for listening

Koichi Sasada  
<ko1@st.inc>

What would you do?  
STORES Tech Conf 2025

What would you **do?**  
STORES Tech Conf 2025



削除スライド

## 作り直したいソフトウェア

- **(should do)** 直す価値のあるソフトウェア
  - 利用者が(いっぱい)いる
  - 利用者が作り直すと喜ぶ(≒ 経済的利得がある)
- **(can do)** 直せるソフトウェア
  - 向上の余地がある
  - 技術力をもっている
- **(will do)** 自分が直したいソフトウェア
  - 自分が使いたい、人に使ってもらいたい
  - チャレンジング and/or できそう

## 低レイヤーと高レイヤー

- 低レイヤーから全体像をつかめると、  
やれること  
やりたいこと  
が膨らんでくる
- 高レイヤー(Ruby ユーザー視点)がないと、  
やるべきこと  
がわからない