

Rubyの中身の予備知識

— RubyKaigiの前に知り鯛！

2025.03.28 RubyKaigi 2025事前勉強会

近藤 うちお (@udzura)

SmartHR プロダクトエンジニア

こんばんは！



自己紹介

近藤 宇智朗 (@udzura)

株式会社 SmartHR / Fukuoka.rb 

プロダクトエンジニア 労務基本機能担当

好きなRubyメソッド: String#unpack

好きなSmartHRの機能: 申請

好きな魚: ふぐ 



2025は3日目にしゃべりますよ！

RubyKaigi 2025

Schedule Speakers Events Sponsors Goodies Policies About X LinkedIn GitHub

Schedule

< Back



Uchio KONDO
Rubyist, Rustacean wannabe, and system programming novice.
Working as a product engineer at SmartHR, Inc. Writer of "mruby system programming (Japanese)" and one of translators on the Japanese edition of "Learning eBPF." Organizer of Fukuoka.rb "2nd season." RubyKaigi speaker since 2016, RubyKaigi local organizer in 2019 (@Fukuoka).

EN

Running ruby.wasm on Pure Ruby Wasm Runtime

The speaker has developed a WebAssembly(wasm) runtime named Wardite, which is implemented entirely in pure Ruby. Wardite implements core wasm specifications and instructions, enabling the successful execution of ruby.wasm with basic Ruby functionalities. This talk will explore the technical challenges of implementing a wasm runtime in pure Ruby and problems encountered during development. Key topics include the implementation of WASI preview 1 support, performance enhancements using ruby-prof and perf, and core wasm specification compliance testing. The talk will provide a comprehensive overview of the progress made so far and the future directions for Wardite, highlighting its potential impact on the Ruby and WebAssembly ecosystems. Attendees will gain insights into the current status of Wardite, its architecture, and the approaches taken to efficiently implement WebAssembly runtime in Ruby.

Wardite



War鰐te



今日のテーマ ～RubyKaigiの前に知り鯛こと～



SmartHRの エンジニアに聞きました

RubyKaigi どうですか？アンケート

- ・RubyKaigi のテーマについて、
 - ・RubyKaigiで聴いて面白かった/面白そうなもの
 - ・RubyKaigiで聴いて難しかった/難しそうなもの
- ・それぞれ最大3つ選んでください！

アンケートの選択肢一覧について

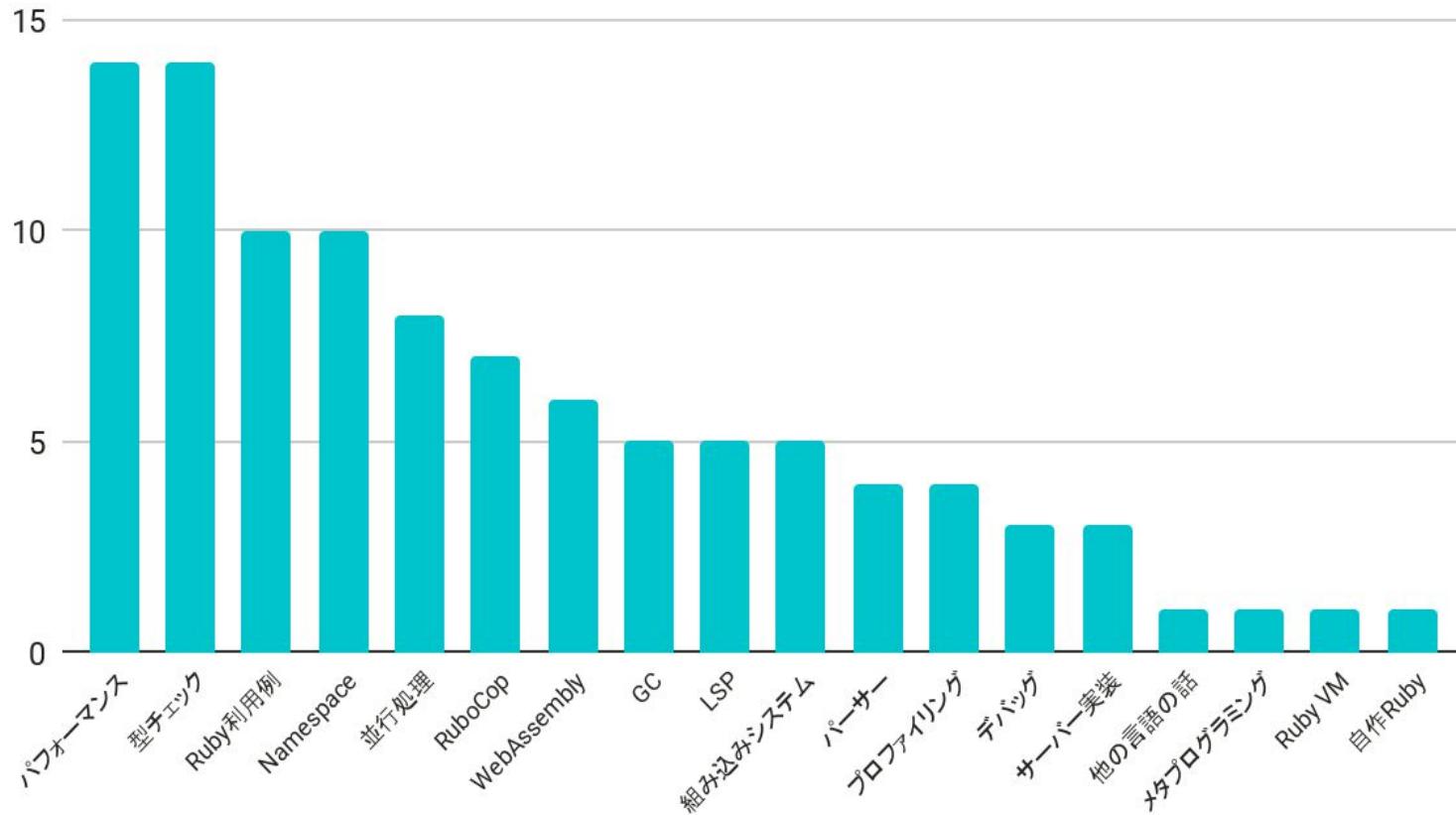
- ・Google NotebookLMに2022~2025のトークテーマを突っ込んで、
「RubyKaigiでよく取り上げられるテーマは？」と聞いてみた
- ・その結果を見て俺LMでちょっと調整して決定した

- パフォーマンス (YJIT、RJIT)
- 型チェック (RBS、Sorbet、Steep)
- Ruby VMの実装
- 並行処理 (Ractor、Fiber、dRuby)
- パーサー (Prism、Lrama)
- RuboCop
- Language Server Protocol
- ガベージコレクション (GC)
- WebAssembly (ruby.wasm)
- 組み込みシステム (mruby、mruby/c、PicoRuby)
- メタプログラミング
- Namespace
- デバッグ (debug.rb)
- JRuby
- 自作Ruby (monoruby)
- サーバー実装 (Falcon、Puma)
- セキュリティ
- プロファイリング/プロファイル (ruby-prof、Pf2、Vernier)
- 他の言語の話 (Elixir、Crystal、Rust...)
- Rubyの面白い利用例

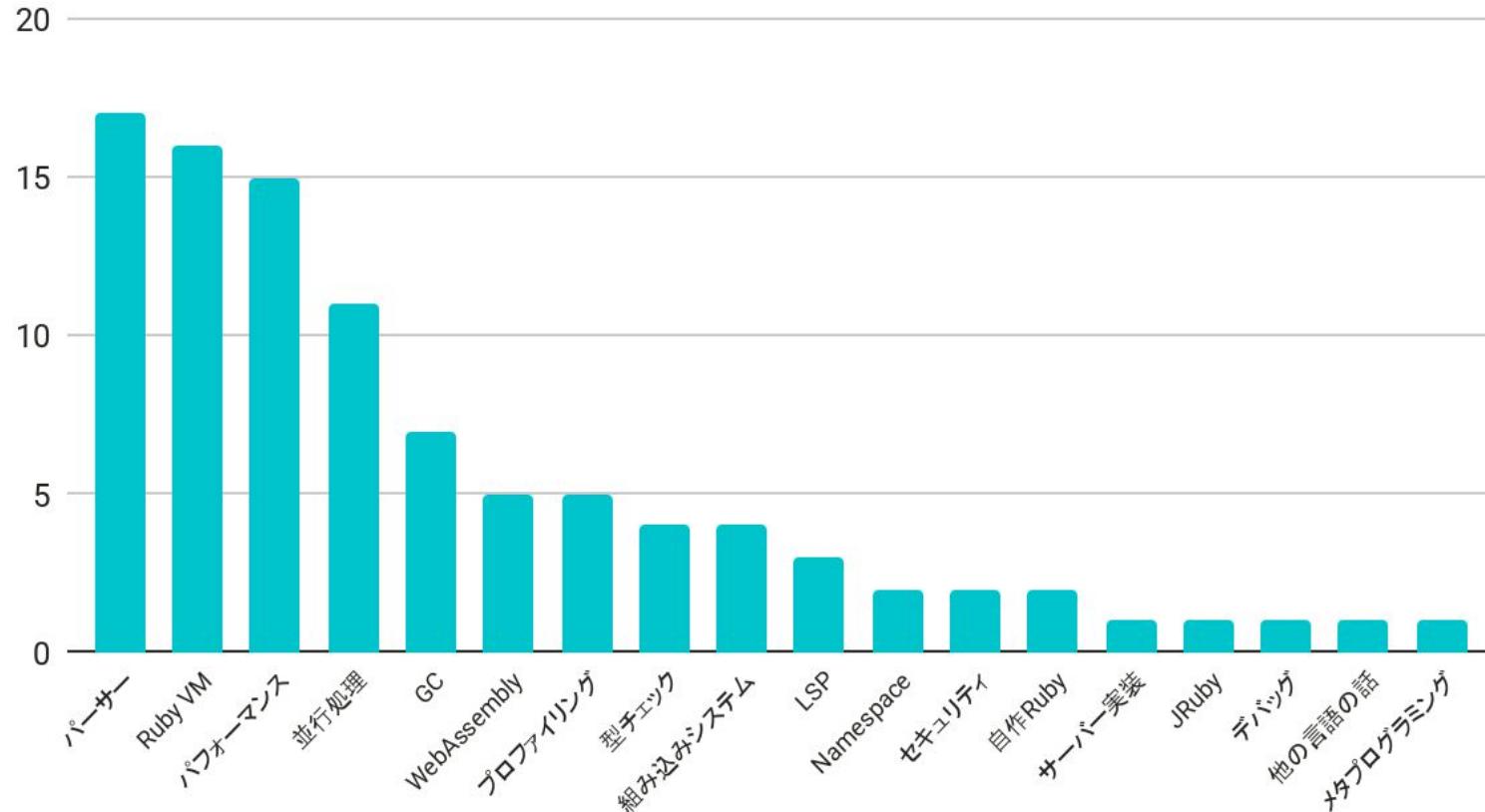
結果



面白そう/面白かった



難しそう/難しかった



考察



こういう傾向があるかも？

- ・身近に考えやすい／ユースケースが想像しやすいものは
「面白い」の上位に入っていそう
- ・例：型チェックは業務でどう役に立つか想像しやすそう
- ・難しい印象になるのは、パーサ、VMなど**言語実装的**な部分
 - ・YJITもある意味でそうかも



それを踏まえて
今日のテーマ

Rubyを通して 言語実装の知識をつけよう

本日のスタンス



今日の資料のスタンスについて

- ・Rubyにおける実装を正しく解説するというより、わかりやすさのため細かい挙動を省略・変更しています。ご了承ください
- ・今回の発表は、いち言語実装好きとして、好きに語っている資料という感じで...
- ・自分の理解で話していますが、こういう観点もある、等はぜひ！

余談について

- ・頑張って内容を絞ったつもりですが、「余談」と称して少し高度な話題に言及している時があります。
- ・書籍のコラムみたいな感じで、よくわからなければスキップしつつお聞きいただければと思います。

心の声：言語実装、深すぎ！

そもそも





「言語」が動くって？

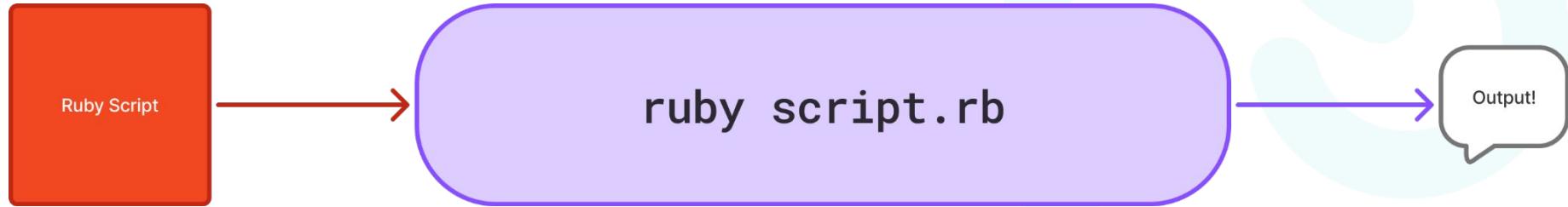


uchio.kondo — -zsh — 58x15

uchio.kondo@uchio 4:14PM ~\$

uchio.kondo@uchio 4:14PM ~\$ ruby script.rb

Rubyスクリプトを実行すると何が起こる？

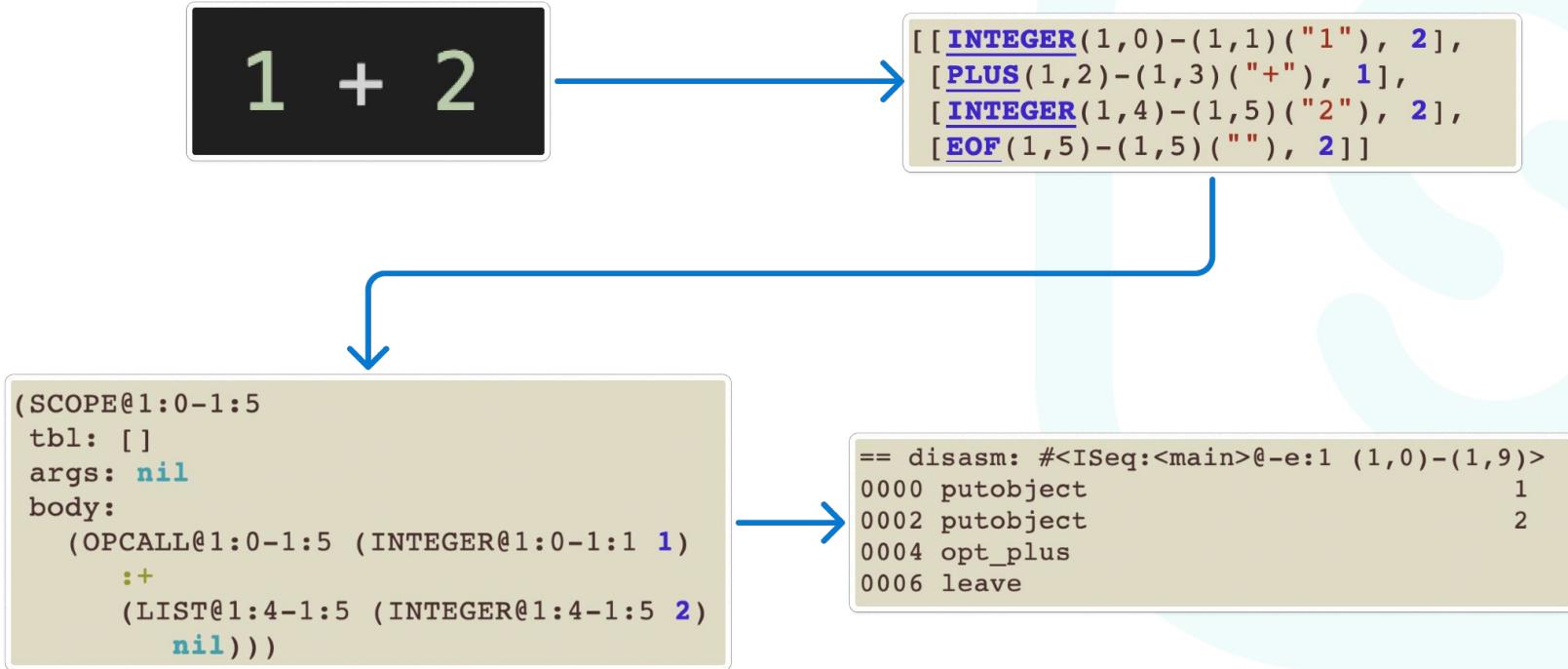


Rubyスクリプトを実行すると何が起こる？

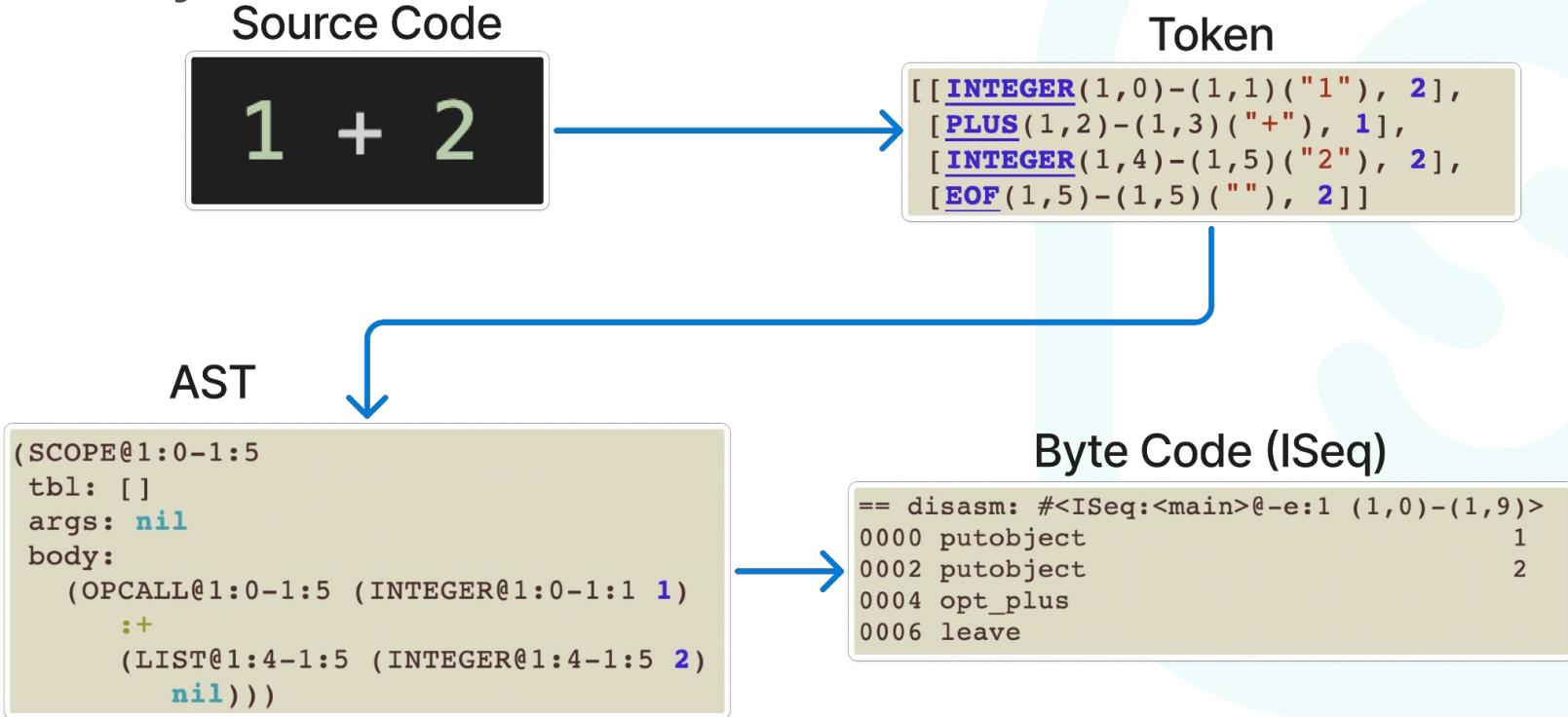
1 + 2



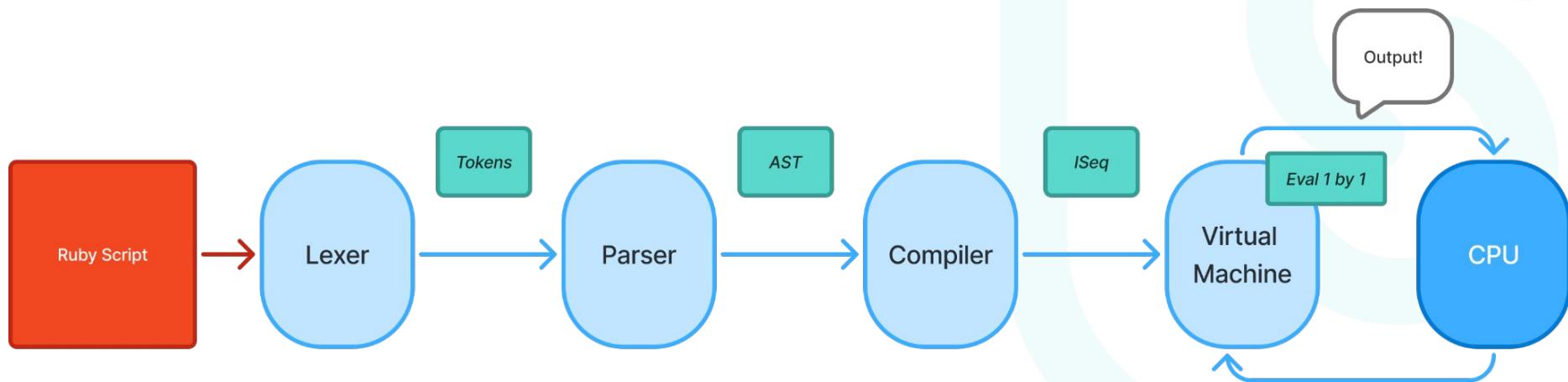
Rubyスクリプトを実行すると何が起こる？



Rubyスクリプトを実行すると何が起こる？



Rubyスクリプトが実行されるプロセス模式図



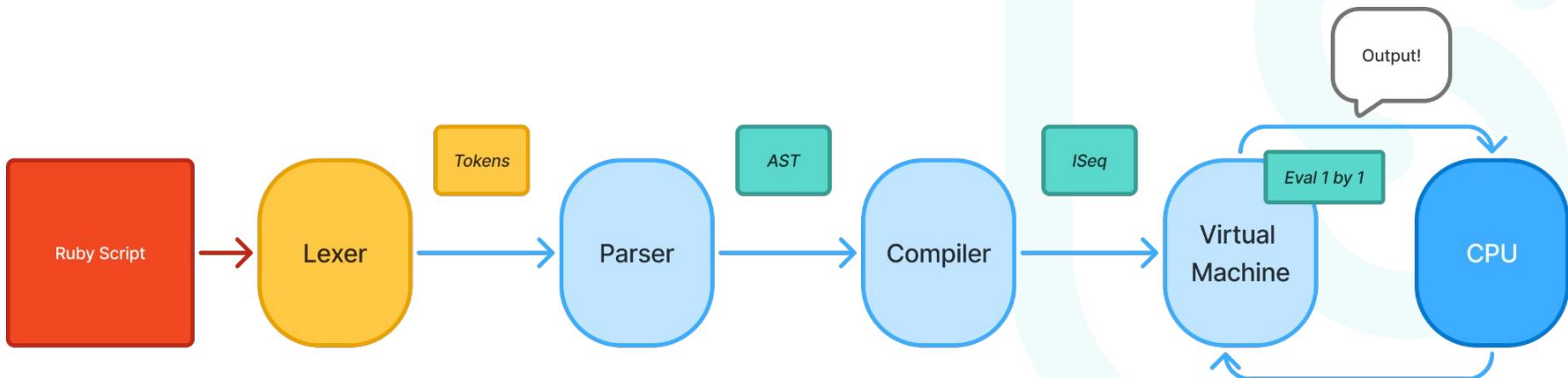
Lexer



Lexer とは？

- ・プログラミング言語のソースコードの文字列をトークンに分割する
- ・トークン: プログラミング言語でそれ以上分けられない最小単位
- ・Lexical Analyzer の略。字句解析器とも
- ・Lexical = 単語の、語彙の

Rubyソースコード → トークン の列にする



具体例を考える

1 + 2

- ・これはRubyとして正しいコードですね



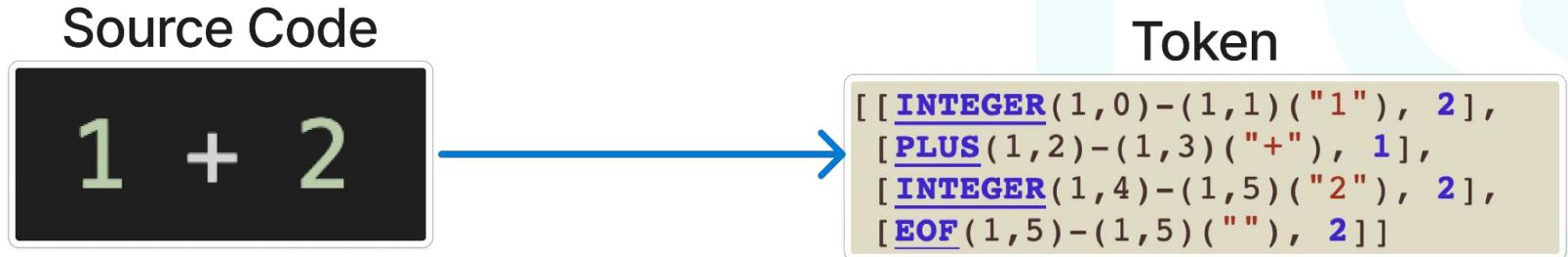
トークンに分割するには？

Prism の機能を使ってソースコードを分割してみましょう

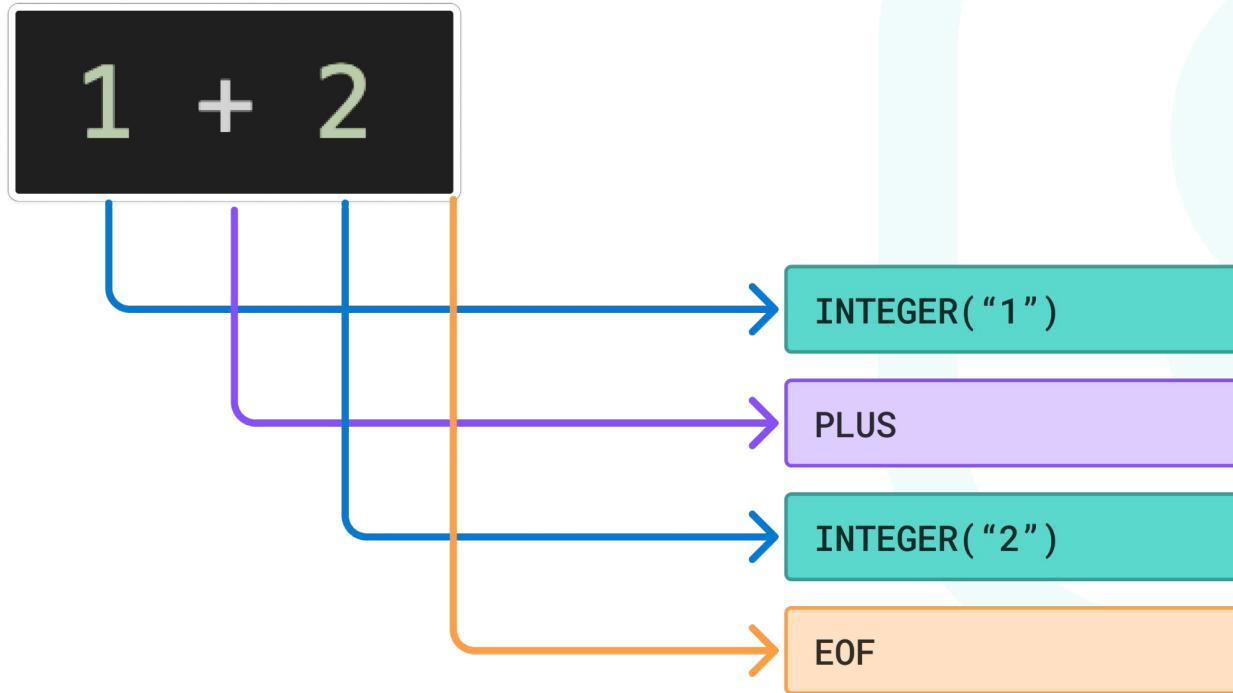
```
Prism.lex("1 + 2")
```

こういうRubyコードを実行します(IRBでOK)

トークンに分割するには？



対応づけ



Parser

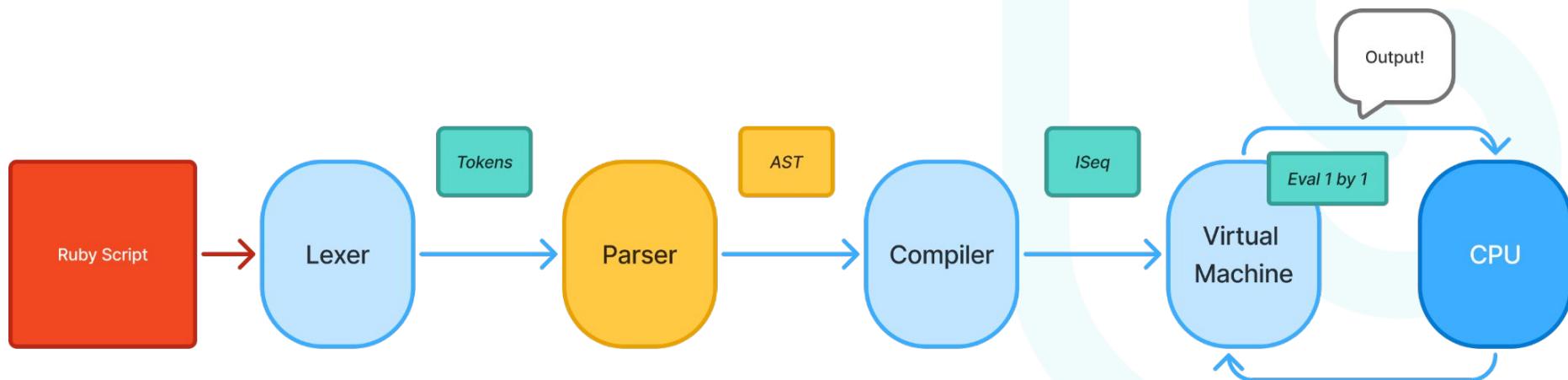


Parserとは何だろう？

- ・ここまでで、トークンの列を生成した
- ・トークンから文法的に意味のある構造を作る必要がある
- ・その実行が Parse (= 構文解析)



トークン → ASTという形式にする



Parserとは何だろう？

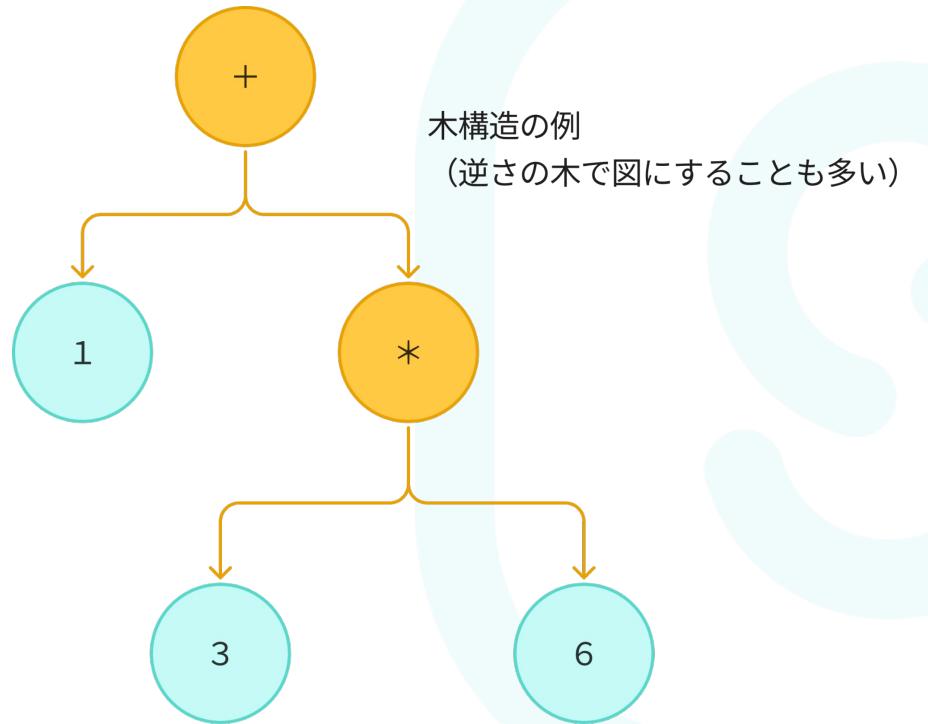
- ・ここまで、トークンの列を生成した
- ・トークンから文法的に意味のある構造を作る必要がある
- ・その実行が Parse (= 構文解析)
- ・意味のある構造 = **AST (Abstract Syntax Tree)**



Abstract Syntax Tree(抽象構文木)

- ・プログラミング言語から意味のある情報を取り出し、
木のような構造にまとめ上げたもの
- ・言語の意味を表現するために木構造が扱いやすい

木構造の例：



※ Rubyと直接関係のない仮の例です

ASTを作つてみよう

IRBなどでこういうコードを実行する

```
RubyVM::AbstractSyntaxTree.parse("1 + 2")
```

ASTを表すインスタンスを作れる

ASTを作つてみよう

1 + 2



ASTを作つてみよう

1 + 2

```
(SCOPE@1:0-1:5
tbl: []
args: nil
body:
(OPCALL@1:0-1:5 (INTEGER@1:0-1:1 1)
:+
(LIST@1:4-1:5 (INTEGER@1:4-1:5 2)
nil)))
```

じゃあ、これは？



じゃあ、これは？(エラー)

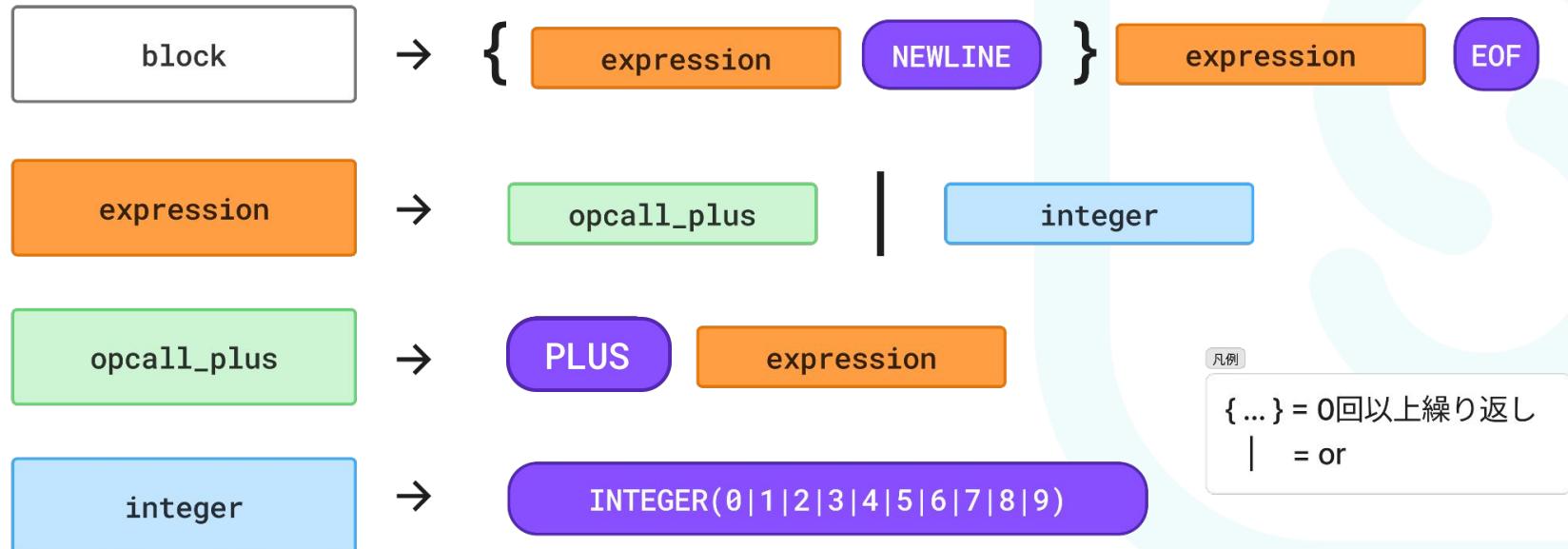
```
+ 1 2
```

```
-e: -e:1: syntax errors found (SyntaxError)
> 1 +  
   | +  
   | ^ unexpected end-of-input, expected a receiver for unary `+`  
   | ^ unexpected end-of-input, assuming it is closing the parent top level context
```

トークン列は
一定のルールに従って解釈され、ASTになる



例えば、以下のようなルールがあるとする



トークン列を受け入れていく

PLUS

INTEGER("1")

INTEGER("2")

EOF

※ + 1 2 が作られたトークン

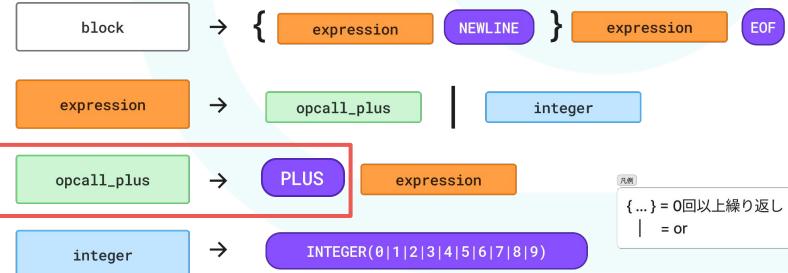
PLUS

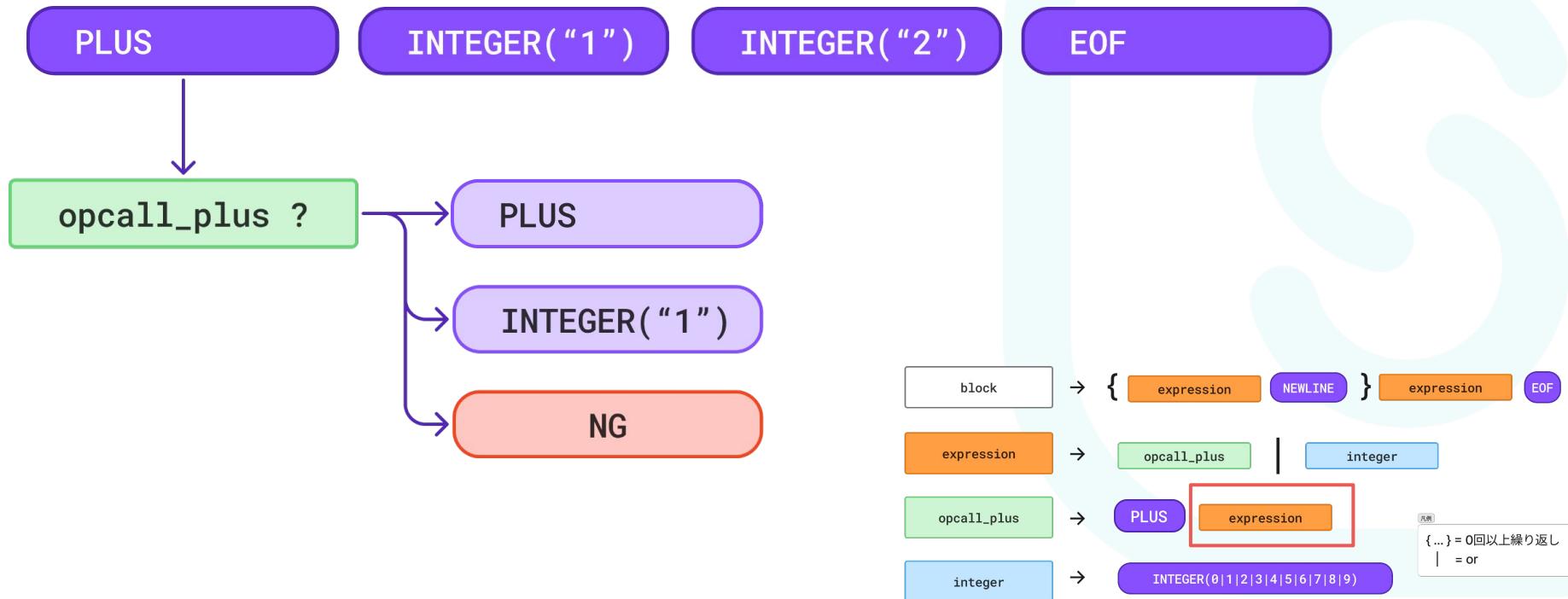
INTEGER("1")

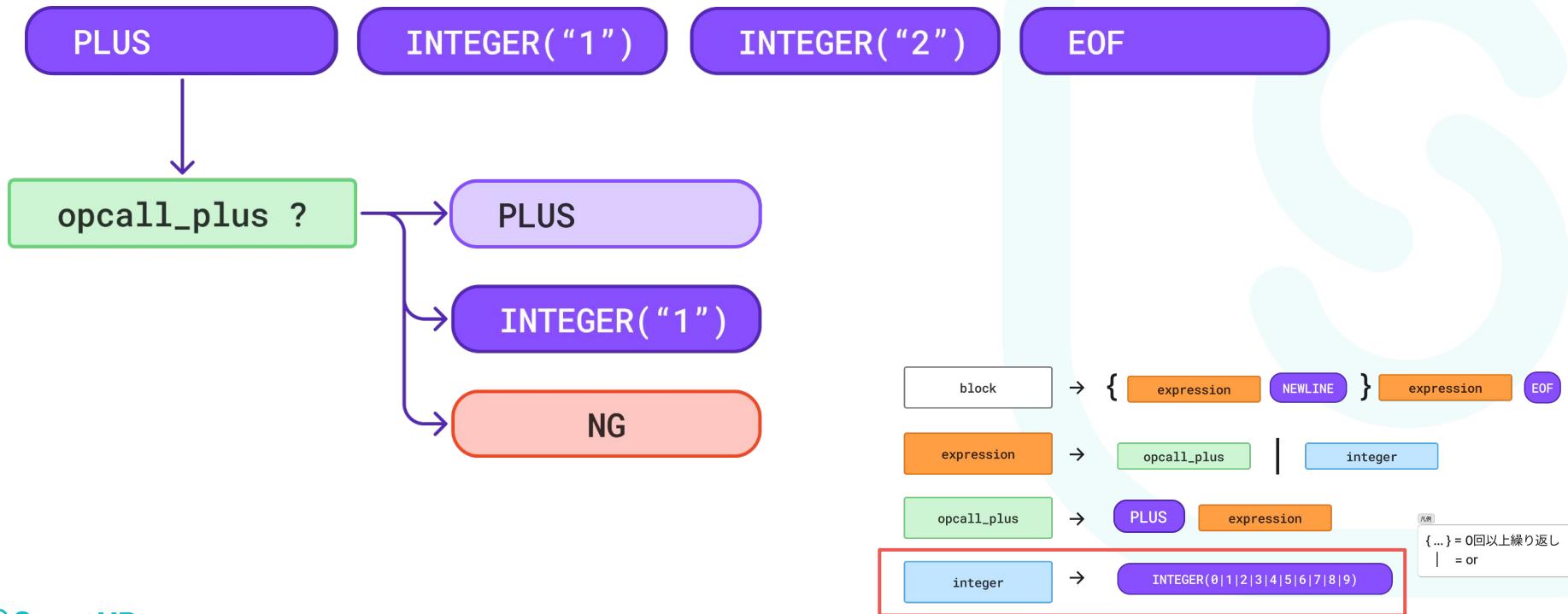
INTEGER("2")

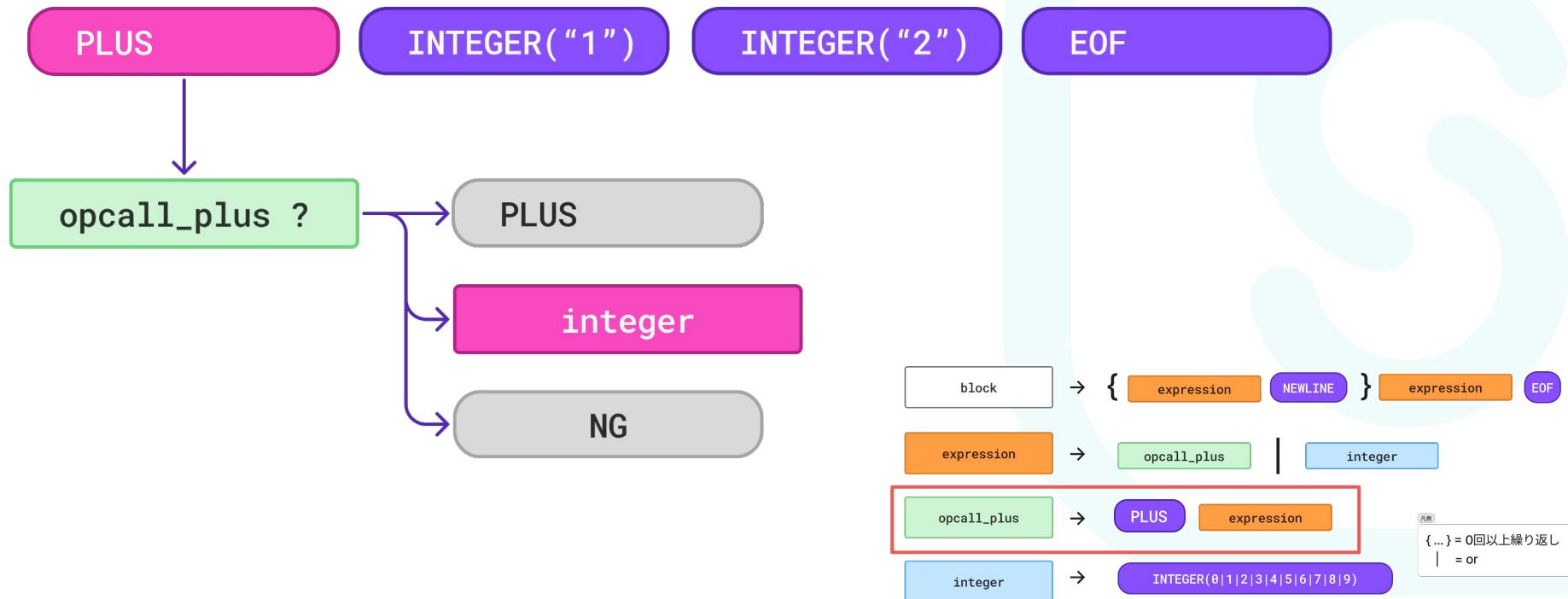
EOF

opcall_plus ?









PLUS

INTEGER("1")

INTEGER("2")

EOF

opcall_plus

integer

PLUS

INTEGER("1")

INTEGER("2")

EOF

opcall_plus

integer

NEWLINE

EOF

NG

block → { expression NEWLINE } expression EOF

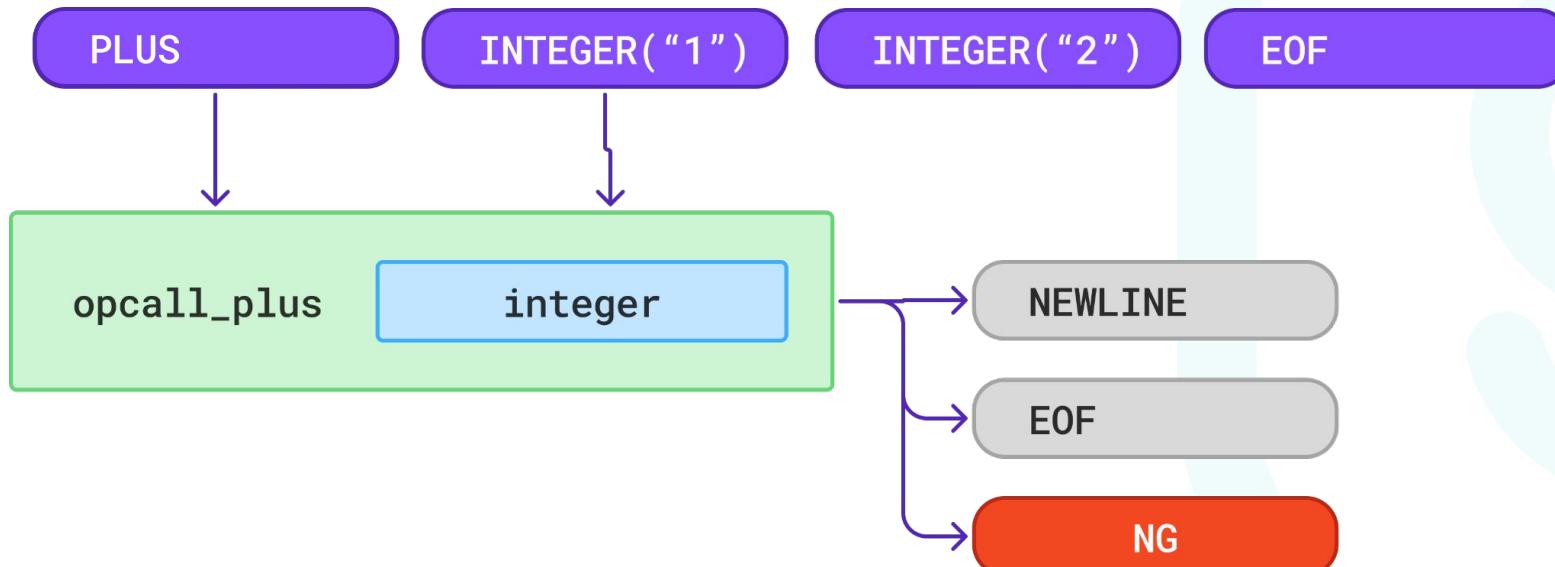
expression → opcall_plus | integer

opcall_plus → PLUS expression

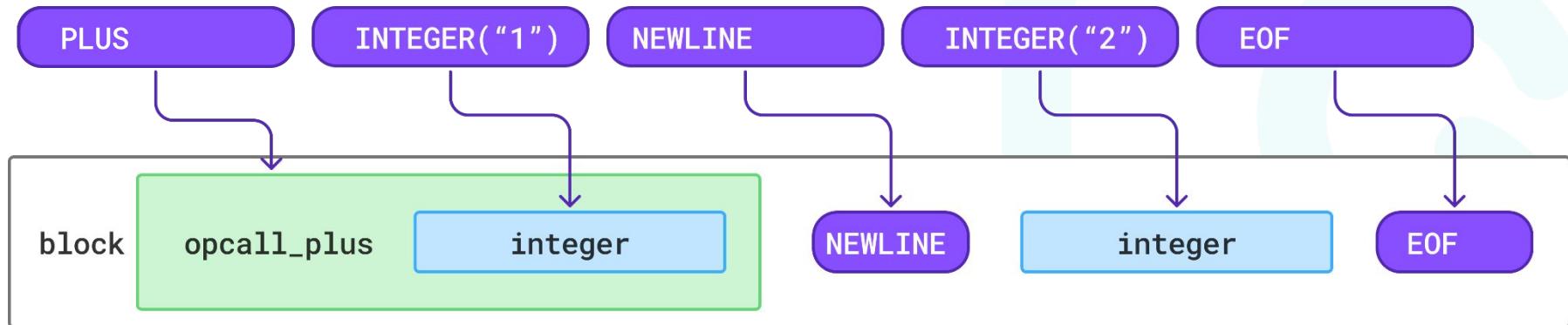
凡例
{ ... } = 0回以上繰り返し
| = or

integer → INTEGER(0|1|2|3|4|5|6|7|8|9)

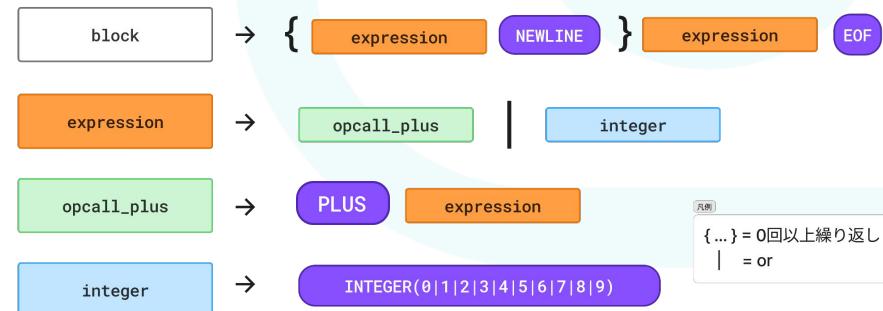
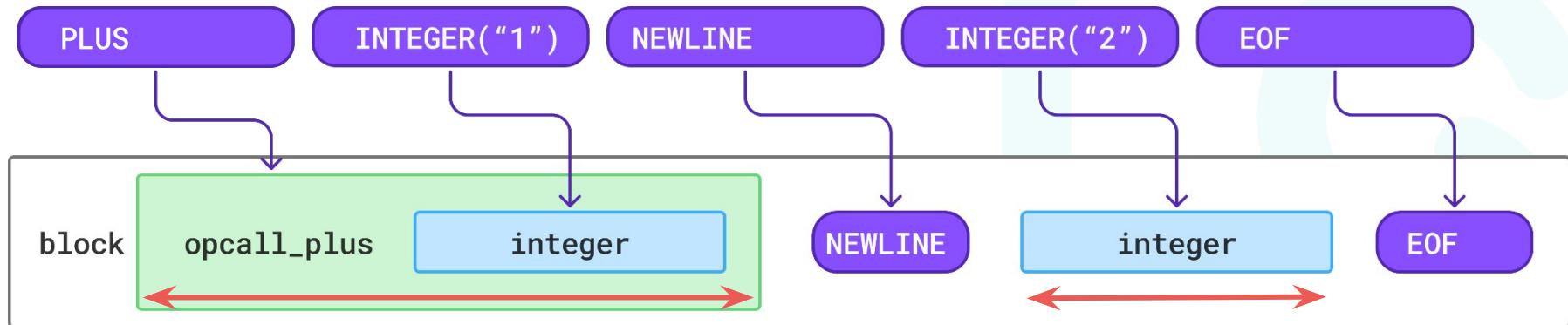
トークン列を受け入れてられない → Error



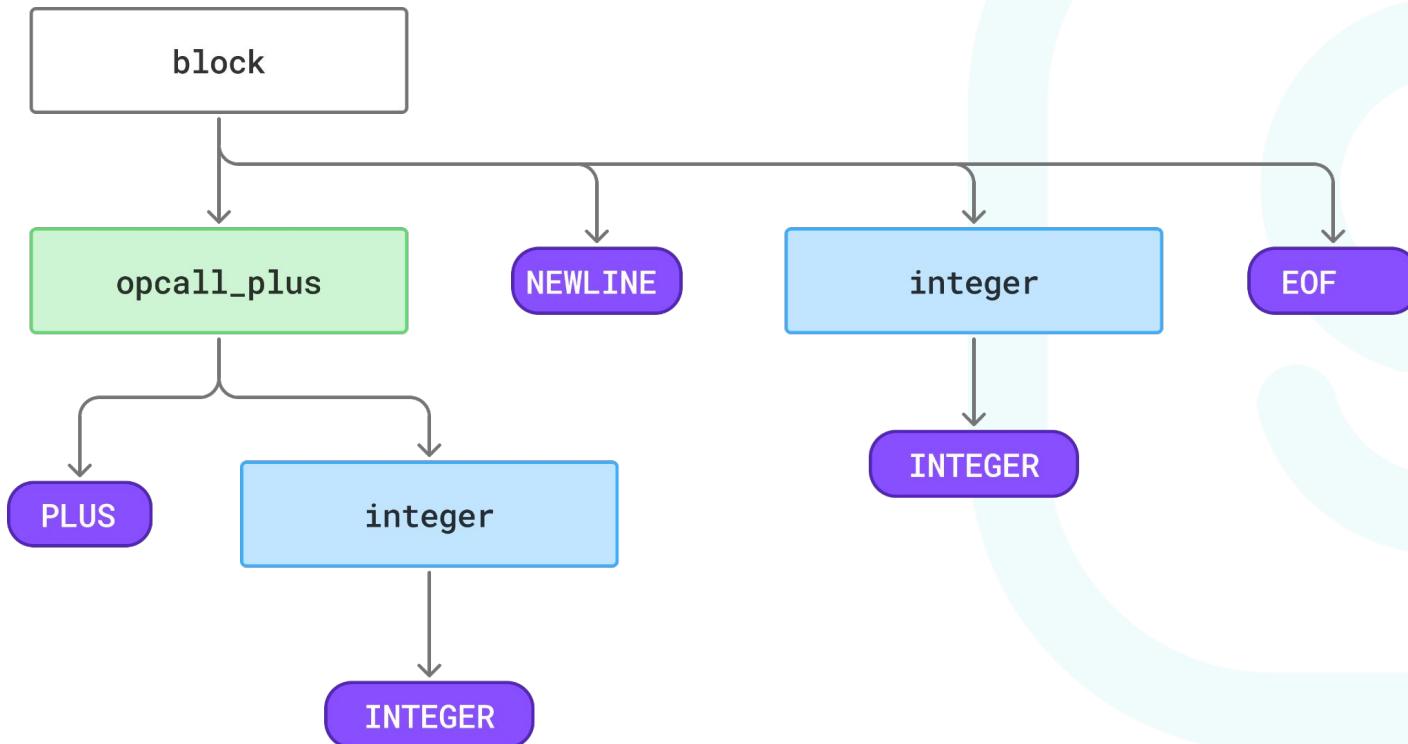
トークン列を受け入れたパターン



トークン列を受け入れたパターン



ちゃんと「木」ができているのを確認



余談

- ・PrismはLLパーサだそうなので、LLを前提とした説明をしました
(と思う)!!
- ・LRパーサの挙動は塩井さんのスライドを読んでください。
- ・LLパーサ/LRパーサとは、についてはjunk0621さんのスライドを…

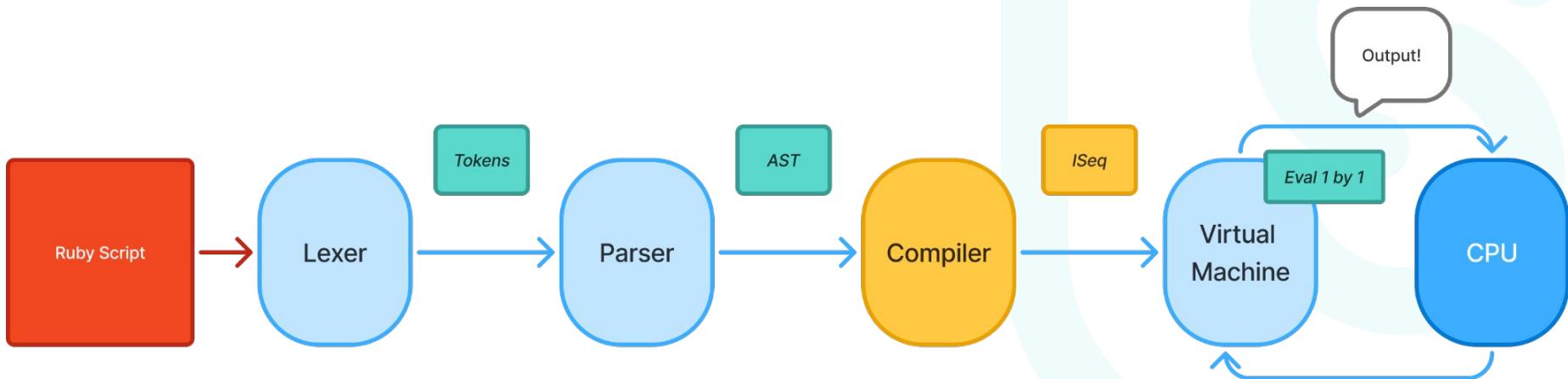
ここまでまとめ

- ・コードの文字列→トークン→ASTにしている
- ・一定の意味があるトークンの並びを受け入れ、構造にするのがパース
 - ・並びのルールがない = 受け入れられない = Syntax Error
- ・構造を持ったデータ(AST)を後ろの工程で使う



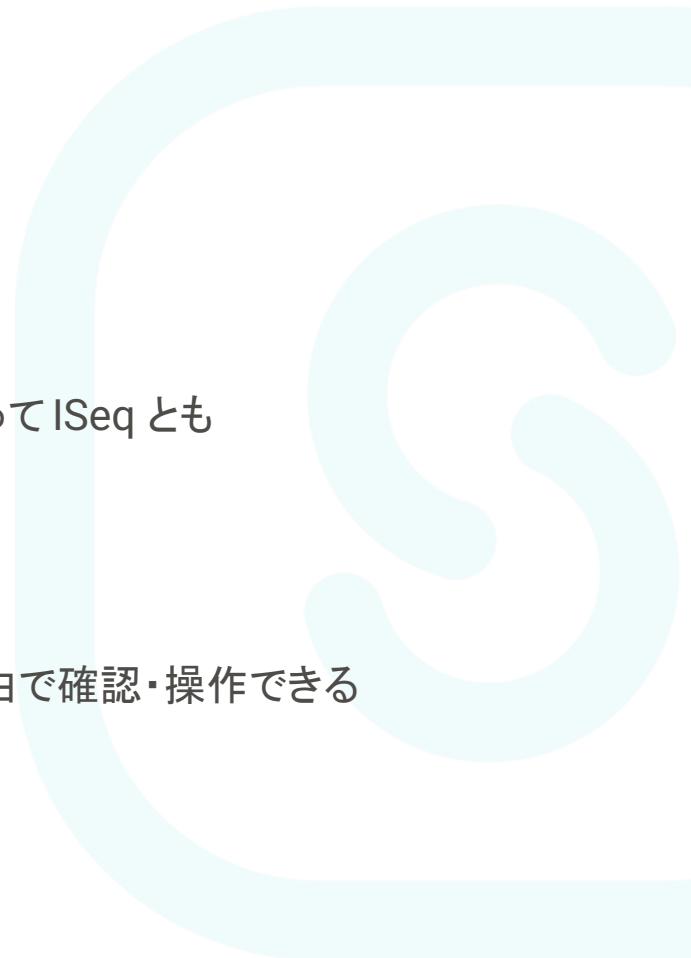
Rubyのバイトコード生成

AST → バイトコードに変換する



バイトコードとは？

- ・Rubyの場合 Instruction Sequence の略称を取って ISeq とも
- ・Instruction Sequence = 「命令の」「列」
- ・命令 = RubyのVMが解釈するための指示
- ・`RubyVM::InstructionSequence` クラス経由で確認・操作できる



具体的なISeqの取り出し方

- 以下のようなRubyコードを書けば、コンパイルされたSeqを
わかりやすく(?) 表示できる

```
iseq = RubyVM::InstructionSequence.compile("1 + 2")
puts iseq.disasm
```

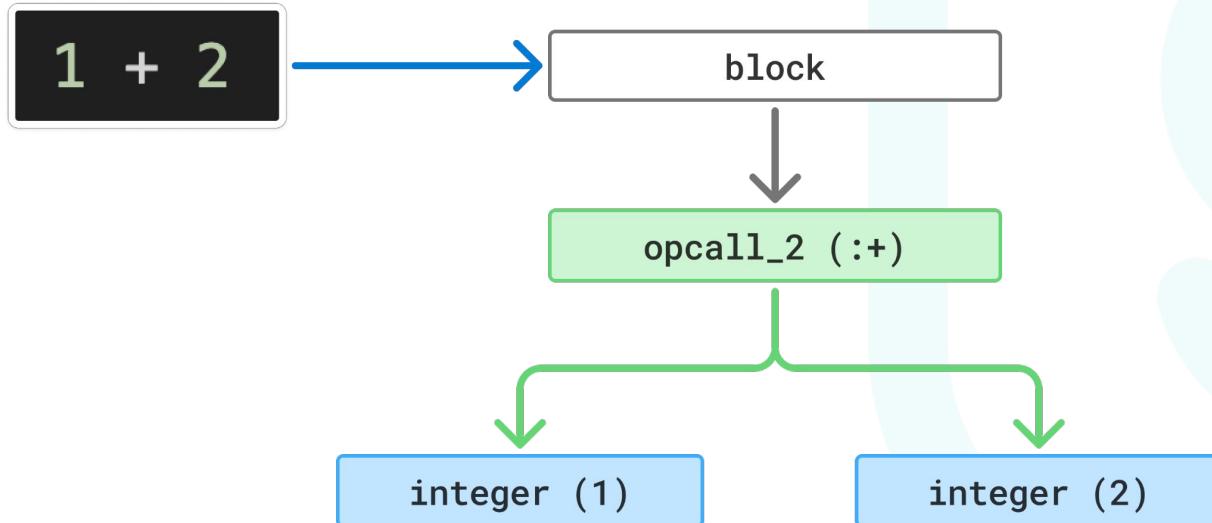
具体的なISeqの例

1 + 2



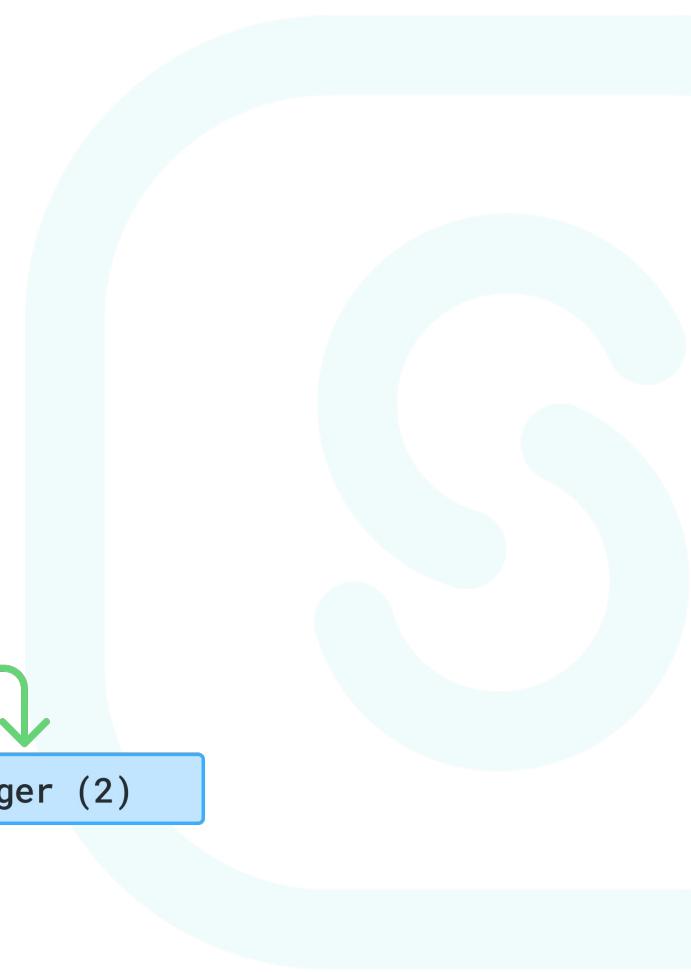
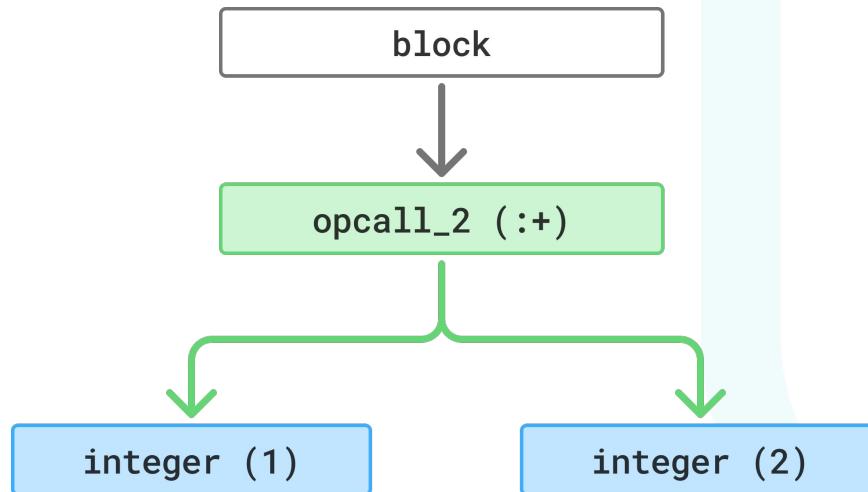
```
== disasm: #<ISeq:<main>@-e:1 (1,0)-(1,9)>
0000 putobject           1
0002 putobject           2
0004 opt_plus
0006 leave
```

ASTを読み取って ISeqを作る

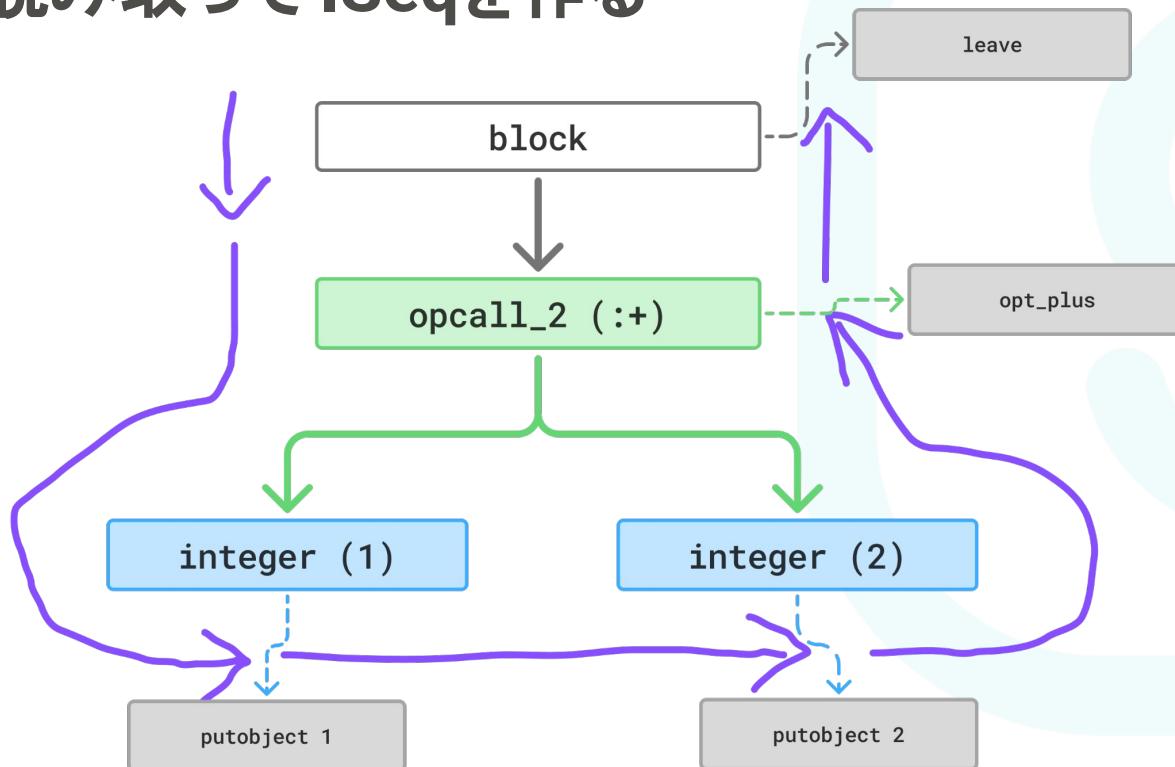


1 + 2 をパースして、こういう ASTが作れたと考えてください

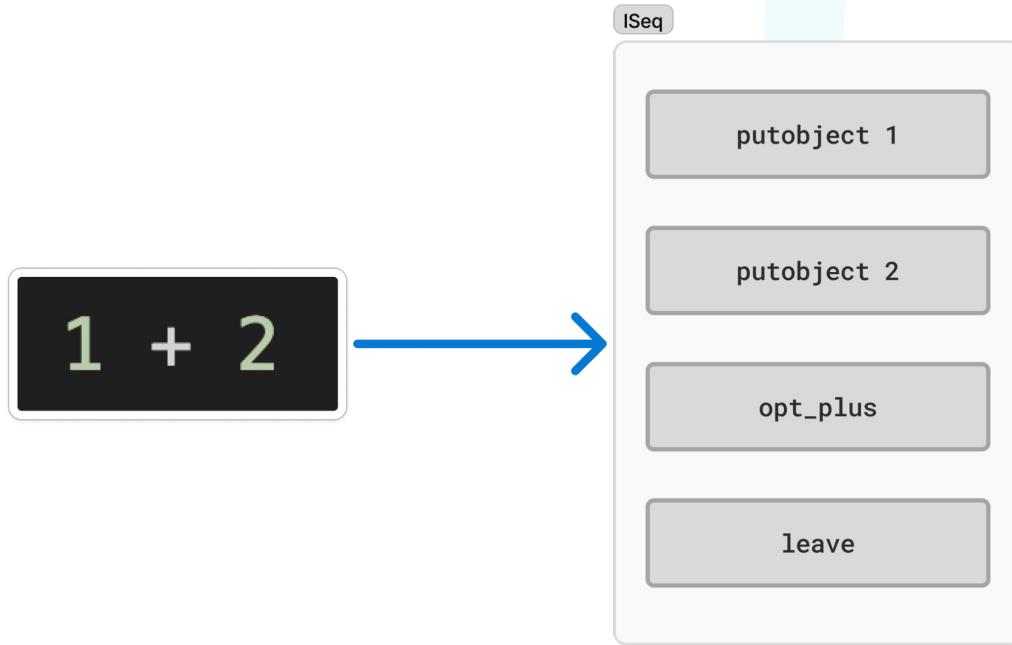
ASTを読み取って ISeqを作る



ASTを読み取って ISeqを作る



ASTを読み取って ISeq が作られた結果



ちなみに: 本物のISeqの様子

000000000	59 41 52 42 03 00 00 00 00 04 00 00 00 00 ac 00 00 00 00	YARB.....
000000100	00 00 00 00 01 00 00 00 00 03 00 00 00 84 00 00 00 00
000000200	a0 00 00 00 6c 08 00 00 db db 9b 8b 03 05 03 03 031.....
000000300	07 01 03 09 c1 03 0d 01 01 03 03 05 00 00 00 00 00 00
000000400	05 21 03 01 01 01 0b 3b 09 01 01 01 01 01 01 01 01 01	.!.....;.....
000000500	01 33 01 03 03 03 03 0d 03 01 03 0b 33 1b 09 0b	.3.....3...
000000600	01 0b 00 ff
000000700	ff ff ff ff ff 0b 03 01 01 01 01 01 01 03 05 01
000000800	03 00 00 00 45 00 00 00 f1 09 00 00 45 05 15 3c	...E.....E..<
000000900	63 6f 6d 70 69 6c 65 64 3e 00 00 00 14 05 03 2b	compiled>.....+
000000a00	88 00 00 00 8c 00 00 00 9c 00 00 00
000000ac0		

- ・実際はバイナリフォーマットに固められ、Ruby内部で利用する
- ・なので「バイト」コードと呼ばれています

VMでバイトコードを実行する

VMとは？

- ・VM = Virtual Machine、日本語で「仮想機械」
 - ・ちなみに、“Virtual”は、日本語のニュアンスだと「実質的に」という解釈をするとしっくりくることがある
- ・「機械のような何か」をソフトウェアで書いたもの



例えば...

- ・PCの中で、別のPCを立ち上げるものも“VM”
 - ・これはCPUという機械をエミュレートしたもの
- ・言語の中で、命令を操作するものも“VM”
 - ・特別な命令を実行し続けるという意味では「機械」



参考: おすすめ記事

YARV Maniacs【第2回】VMってなんだろう

計算機資源を仮想化するということ

そもそもなんで仮想化するんでしょうね。

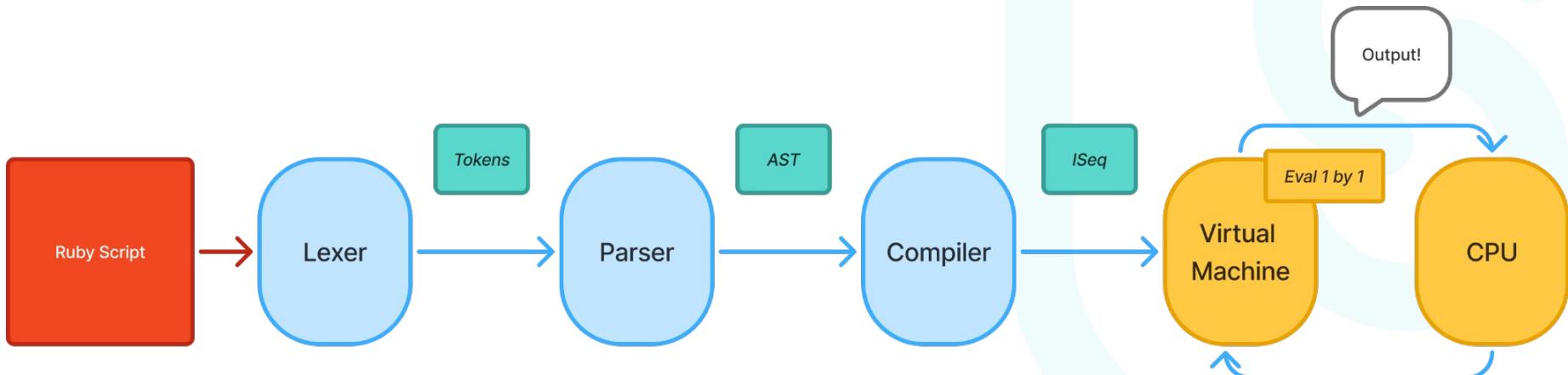
こんなわけで、具体的な何かに依存するよりは、中間層を設けることによって別々のものを扱いやすくしましょう、というのが仮想化です。中間層により、上層で利用することのできるインターフェースを共通化することで利用しやすくなってしまうね、ということです(移植性の向上)。

RubyにもVMがある

- ・そんなVMは実はシンプルな動作原理をしている
- ・最初からたどって、命令に沿った操作を実行していくだけ
- ・Rubyが内部で持っているVMを深掘りしてみよう



バイトコード → VMで一つずつ実行する



先ほど生成した命令を例に

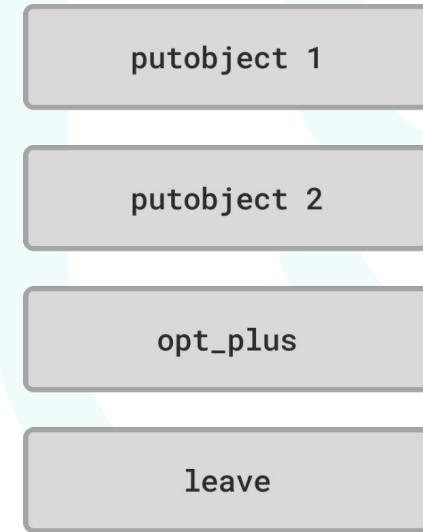
putobject 1

putobject 2

opt_plus

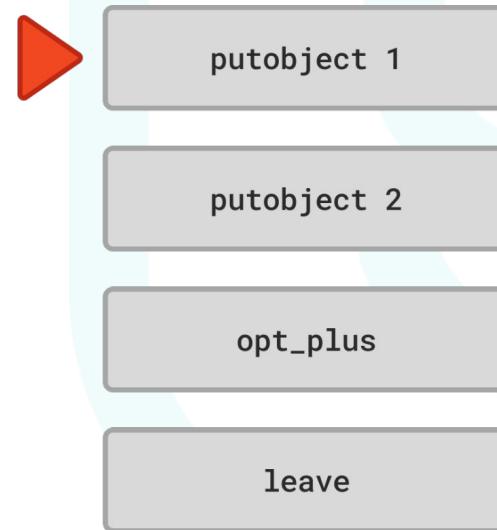
leave

VMのスタックの状態を観測



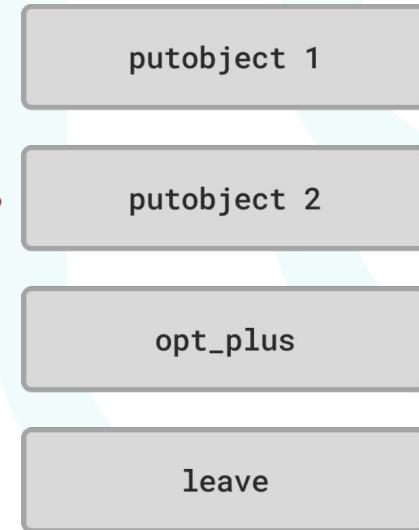
※ Rubyのコード経由で確認することはできないので、あくまで模式図

VMのスタックの状態を観測



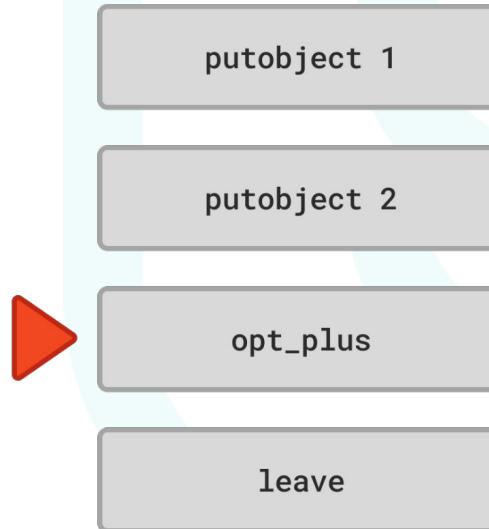
※ Rubyのコード経由で確認することはできないので、あくまで模式図

VMのスタックの状態を観測



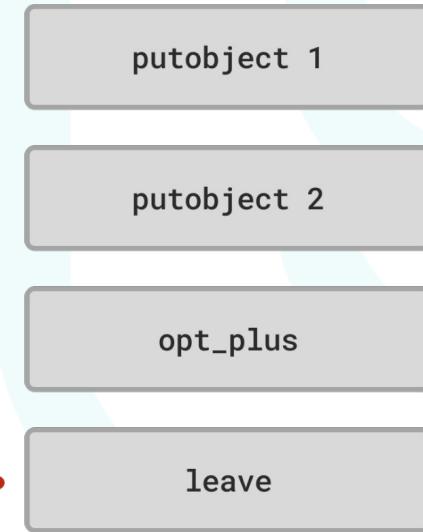
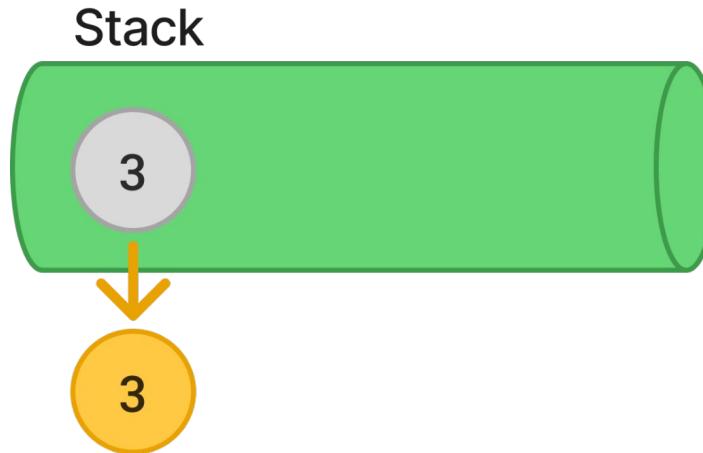
※ Rubyのコード経由で確認することはできないので、あくまで模式図

VMのスタックの状態を観測



※ Rubyのコード経由で確認することはできないので、あくまで模式図

VMのスタックの状態を観測



※ Rubyのコード経由で確認することはできないので、あくまで模式図

余談： いろいろな VM

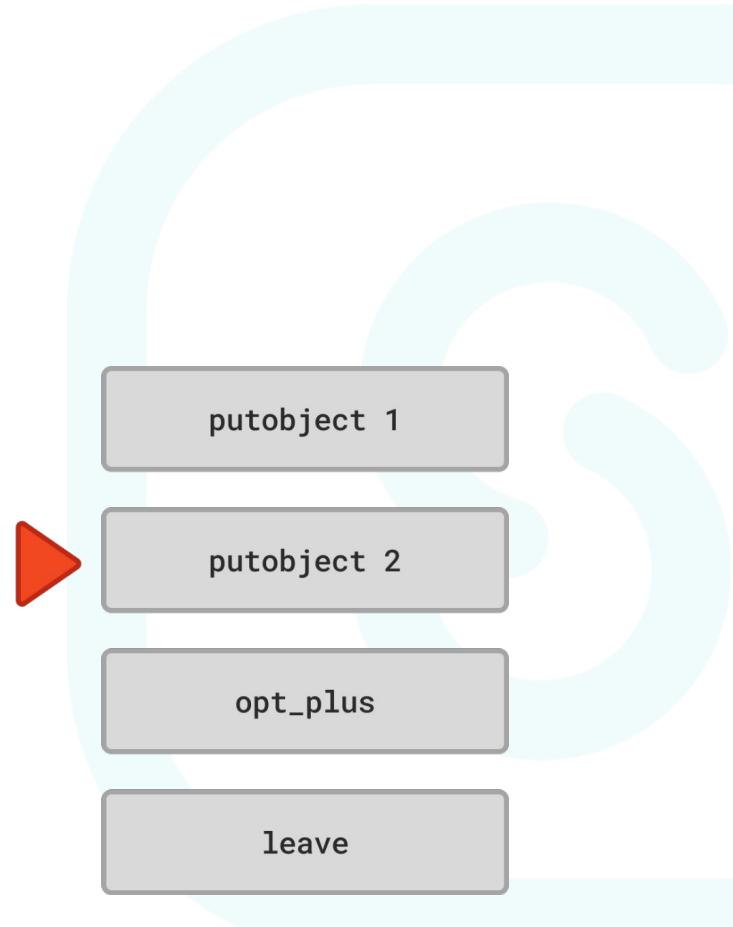


2種類のマシン

- ・スタックマシンとレジスタマシン
- ・CRubyのVMはスタックマシン



スタックマシン



レジスタマシン

R1	1
R2	2
R3	
R4	
R5	



LOADI R1 1

LOADI R2 2

ADD R1 R2

RETURN R1

mruby

- ・もう一つの軽量なRuby実装
- ・バイトコードの仕様はmruby, mruby/c, PicoRuby, mruby/edge で共通
- ・レジスタマシンを使用

```
irep 0x6000003f4000 nregs=4 nlocals=1 pools=2 syms=0 reps=0 ilen=11
file: file1.rb
  1 000 STRING  R1  L[0] ; foo
  1 003 STRING  R2  L[1] ; bar
  1 006 ADD     R1  R2
  1 008 RETURN   R1
  1 010 STOP
```

Lua

- ・組み込みなど各所で使われる軽量言語
- ・レジスタマシンを使用。mrubyも影響を受けた

```
$ luac5.4 -l file1.lua

main <file1.lua:0,0> (7 instructions at 0x600002dc0080)
0+ params, 2 slots, 1 upvalue, 0 locals, 2 constants, 0 functions
 1 [1] VARARGPREP  0
 2 [1] LOADK      0 0 ; "foo"
 3 [1] LOADK      1 1 ; "bar"
 4 [1] ADD        0 0 1
 5 [1] MMBIN     0 1 6 ; __add
 6 [1] RETURN    0 2 1 ; 1 out
 7 [1] RETURN    0 1 1 ; 0 out
```

WebAssembly

- ・Wasmのバイナリを動かすための仕様
- ・色々な言語をWasmのバイナリにコンパイルすれば動く
- ・VMはスタックマシン

```
(module
  (type $t (func (result i32)))
  (export "add" (func $add))
  (func $add (type $t) (result i32)
    i32.const 1
    i32.const 2
    i32.add
    return
  ))
```

ここまでまとめ

- ・ASTをさらに読み込んで、Ruby VMの命令列にする
- ・VMはその命令列を解釈して実際のプログラムを実行する
- ・色々な言語に色々なVMがある



One More Thing...



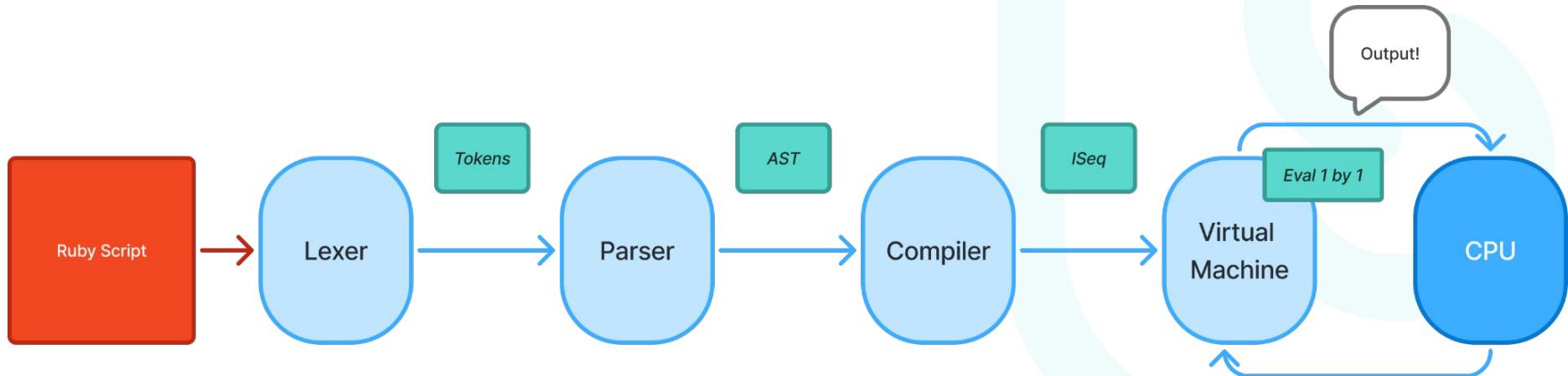
JITコンパイルについて

JITコンパイルとは

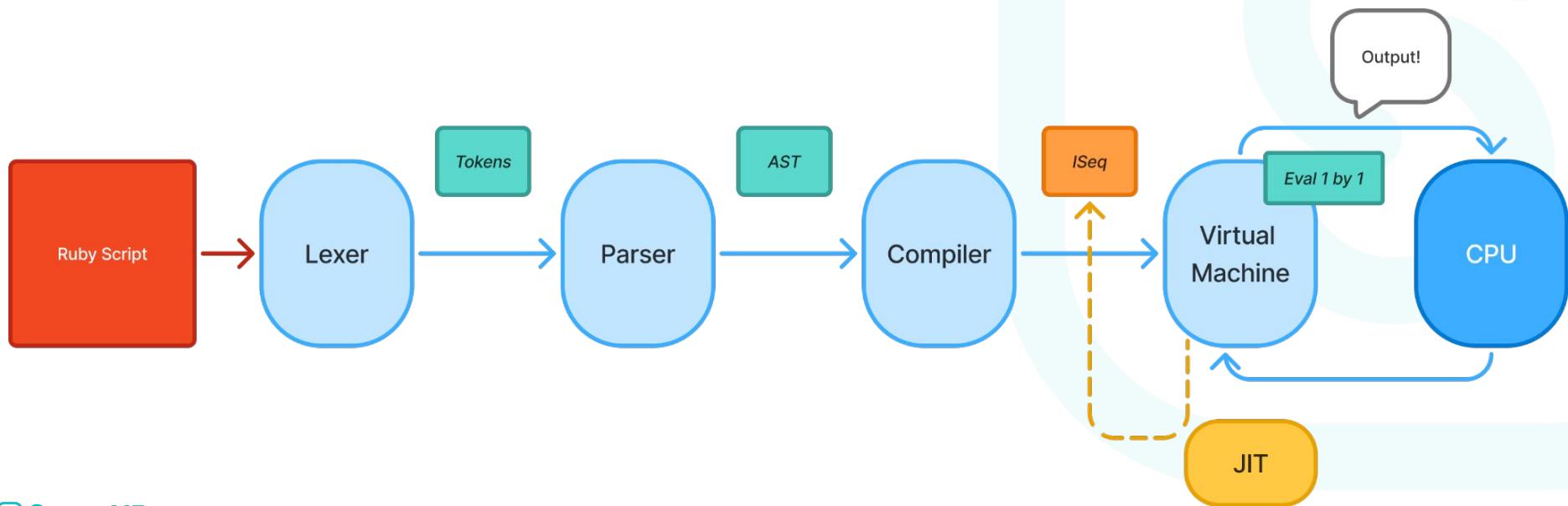
- ・Just In Time (= その場で)コンパイル
- ・VM用の命令を、機械語の命令に、その場でコンパイルする
- ・単に JIT と呼ぶことも多い



Rubyスクリプトが実行されるプロセス



JITは、生成後の ISeq を動的に操作する仕組み



JITの仕組み



こういう命令があるとする

- ・あるメソッドをコンパイルした結果と仮定する

putobject 10

putobject 20

opt_plus

leave



この命令は何度も実行されるとわかった

- ・VMの中で統計を取ったりして判断

putobject 10

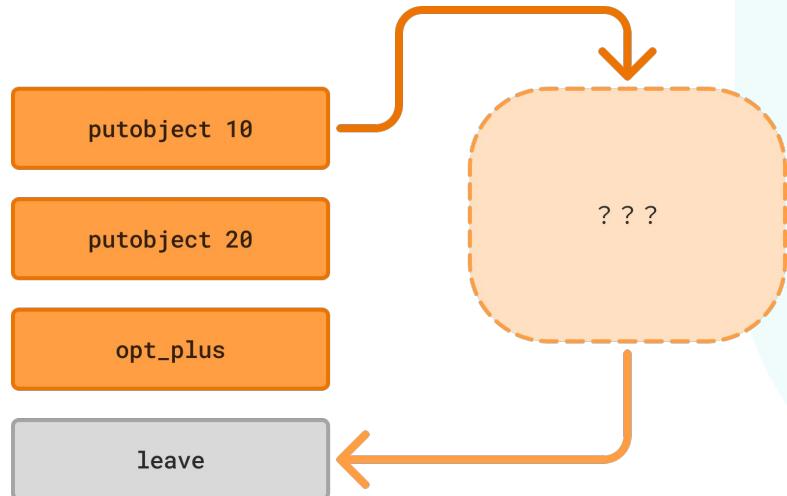
putobject 20

opt_plus

leave

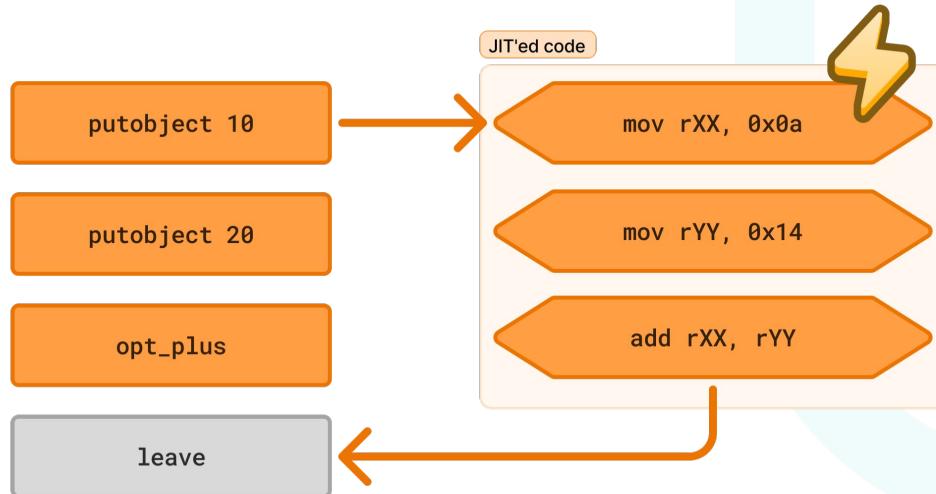
じゃあ、ショートカットできない？

- ・VMを一つ一つ解釈するより、同じ計算を直接機械語(など)で実行できるようになったら高速になるはず



ある命令を動的にコンパイルし、機械語に

- これがJITの基本的な考え方
- もちろん「コンパイルするコスト」もある



RubyのJITの紹介

よく使われているのは YJIT

- ・今日はYJITに絞った説明をします



2025-03-28

SmartHR最大のRailsアプリケーションでYJITを有効化しました

こんにちは、SmartHR プロダクトエンジニアのB6です。

[YJITが本番環境で安定して使える](#)状態になってから、様々な場所でYJITの効果を耳にしてきました。

このたび、「基本機能」と呼ばれるSmartHR最大のRailsアプリケーションにもYJITの導入を完了しました。[1 本記](#)事では、その導入結果と実施した対応について共有いたします。

導入環境

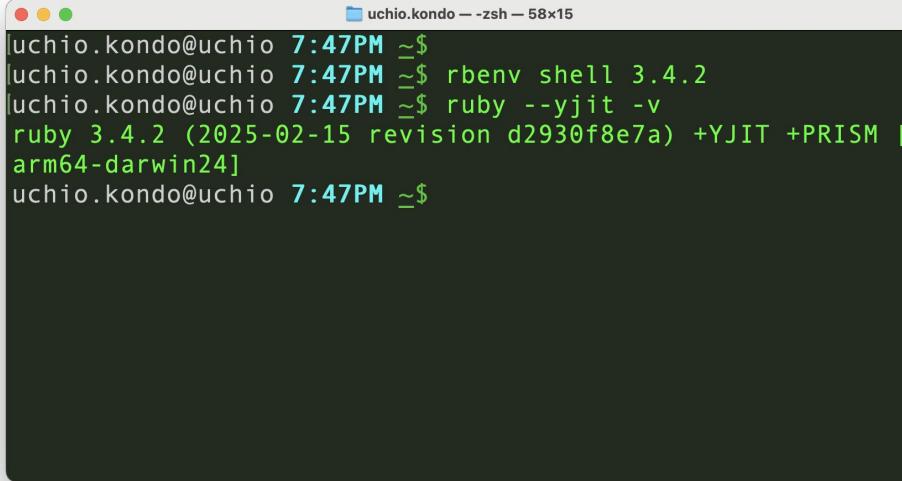
YJIT導入時のRuby、Railsのバージョンは以下の通りです。

- Ruby 3.3
- Rails 7.1

▪ [SmartHRでも有効にしています！](#)

YJITを試してみよう

- ・前提として、YJITをビルドするにはRustのコンパイラが必要になる
- ・`rustc` が使える環境でビルドすれば、自動でYJIT有効のRubyになる
- ・コマンドラインオプション`--yjit` をつけて起動する



```
uchio.kondo@uchio 7:47PM ~$ rbenv shell 3.4.2
uchio.kondo@uchio 7:47PM ~$ ruby --yjit -v
ruby 3.4.2 (2025-02-15 revision d2930f8e7a) +YJIT +PRISM [
  arm64-darwin24]
uchio.kondo@uchio 7:47PM ~$
```

YJITしてみる対象のプログラム

- ・benchmark-ips
- gemを利用
- ・シンプルな分岐
をさせるだけ

```
require 'benchmark/ips'

def bench(a)
  if a + 1 > 3
    true
  else
    false
  end
end

Benchmark.ips do |x|
  x.report('rand') { bench(rand(5)) }
  x.report('always_true') { bench(1) }
  x.report('always_false') { bench(3) }
end
```

YJITなし

```
(* '-') < ruby bench.rb
ruby 3.4.2 (2025-02-15 revision d2930f8e7a) +PRISM [arm64-darwin24]
Warming up -----
      rand      1.186M i/100ms
  always_true      2.579M i/100ms
always_false      3.358M i/100ms
Calculating -----
      rand      11.259M (± 2.6%) i/s   (88.82 ns/i) -      56.916M in  5.058918s
  always_true      25.823M (± 0.8%) i/s   (38.73 ns/i) -     131.510M in  5.093165s
always_false      33.042M (± 1.3%) i/s   (30.26 ns/i) -     167.918M in  5.082869s
```

YJITあり

```
[(*'-') < ruby --yjit bench.rb
ruby 3.4.2 (2025-02-15 revision d2930f8e7a) +YJIT +PRISM [arm64-darwin24]
Warming up -----
      rand      1.562M i/100ms
  always_true      4.719M i/100ms
always_false      4.315M i/100ms
Calculating -----
      rand      17.659M (± 2.4%) i/s    (56.63 ns/i) -      89.032M in  5.044791s
  always_true      61.773M (± 0.1%) i/s    (16.19 ns/i) -     311.473M in  5.042226s
always_false      62.022M (± 0.1%) i/s    (16.12 ns/i) -     310.708M in  5.009650s
```

YJIT の動作を確認する

YJITをdebug buildしてみよう

- ・こういうオプションでビルド

```
export CONFIGURE_OPTS="--enable-yjit=dev"  
rbenv install 3.4.2
```

- ・本番環境ではこのビルドは使わないこと！

YJITがコンパイルした結果を見ることができる

- RubyVM::YJIT.disasm で確認できる

```
def target
| 10 + 10
end

10000.times { target }

isec = RubyVM::InstructionSequence.of method(:target)
puts RubyVM::YJIT.disasm(isec)
```

YJITによるコンパイルの結果

```
[udzura@lima-yjit-test:/Users/udzura/work$ ruby --yjit /tmp/jit.rb
== disasm: #<ISeq:target@/tmp/jit.rb:1 (1,0)-(3,3)>
0000 putobject                      10          ( 2 )[LiCa]
0002 putobject                      10
0004 opt_plus                         <calldata!mid:+, argc:1, ARGS_SIMPLE>[CcCr]
0006 leave                           ( 3 )[Re]

NUM BLOCK VERSIONS: 1
TOTAL INLINE CODE SIZE: 58 bytes
== BLOCK 1/1, ISEQ RANGE [0,7), 58 bytes =====
 0x753ee3e110c6: mov esi, 0x15
 0x753ee3e110cb: mov edi, 0x15
 0x753ee3e110d0: mov rax, rsi
 0x753ee3e110d3: sub rax, 1
 0x753ee3e110d7: add rax, rdi
 0x753ee3e110da: jo 0x753ee3e1310a
 0x753ee3e110e0: mov rsi, rax
 0x753ee3e110e3: mov eax, dword ptr [r12 + 0x20]
 0x753ee3e110e8: test eax, eax
 0x753ee3e110ea: jne 0x753ee3e13132
 0x753ee3e110f0: add r13, 0x38
 0x753ee3e110f4: mov qword ptr [r12 + 0x10], r13
 0x753ee3e110f9: mov rax, rsi
 0x753ee3e110fc: jmp qword ptr [r13 - 8]
```

YJITによるコンパイルの結果

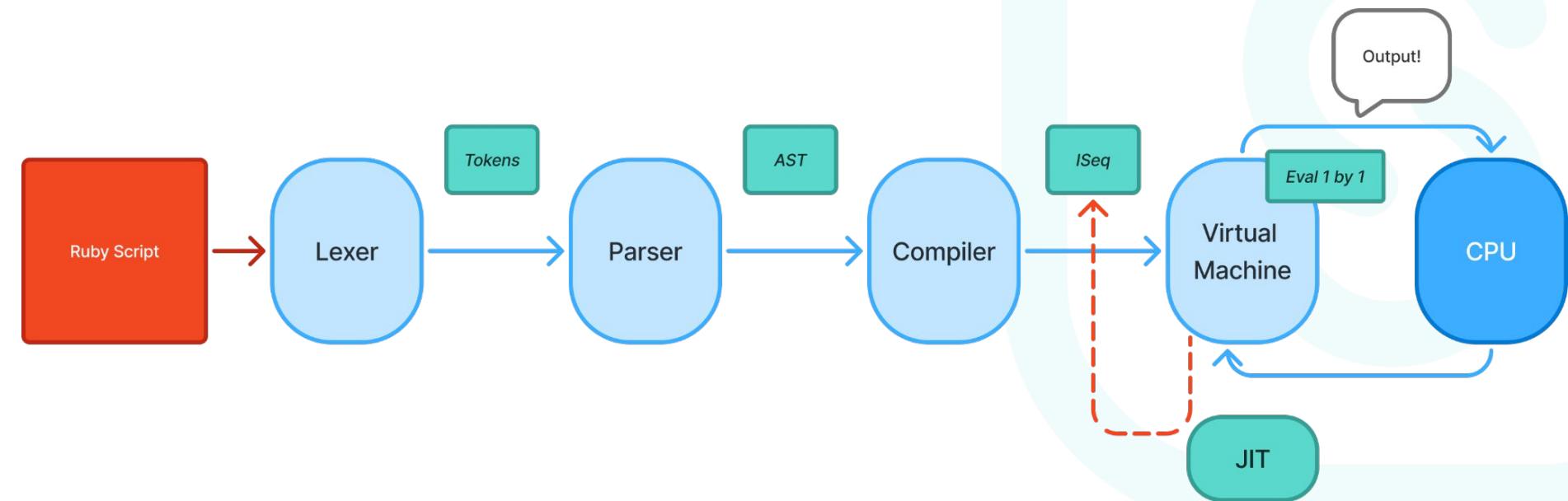
```
udzura@lima-yjit-test:/Users/udzura/work$ ruby --yjit /tmp/jit.rb
== disasm: #<ISeq:target@/tmp/jit.rb:1 (1,0)-(3,3)>
0000 putobject                      10          ( 2 )[LiCa]
0002 putobject                      10
0004 opt_plus                         <calldata!mid:+, argc:1, ARGS_SIMPLE>[CcCr]
0006 leave                           ( 3 )[Re]

NUM BLOCK VERSIONS: 1
TOTAL INLINE CODE SIZE: 58 bytes
== BLOCK 1/1, ISEQ RANGE [0,7], 58 bytes =====
0x753ee3e110c6: mov esi, 0x15
0x753ee3e110cb: mov edi, 0x15
0x753ee3e110d0: mov rax, rsi
0x753ee3e110d3: sub rax, 1
0x753ee3e110d7: add rax, rdi
0x753ee3e110da: jne 0x753ee3e1310a
0x753ee3e110e0: mov rsi, rax
0x753ee3e110e3: mov eax, dword ptr [r12 + 0x20]
0x753ee3e110e8: test eax, eax
0x753ee3e110ea: jne 0x753ee3e13132
0x753ee3e110f0: add r13, 0x38
0x753ee3e110f4: mov qword ptr [r12 + 0x10], r13
0x753ee3e110f9: mov rax, rsi
0x753ee3e110fc: jmp qword ptr [r13 - 8]
```

まとめ



Rubyを通して言語実装の要素技術を確認した



言語実装の一つ一つの操作は比較的小さい

- ・責務を限定して、実装やデバッグをしやすくしている
- ・皆さんのソフトウェア開発と同じ
- ・インプット/アウトプットを押さえれば怖くない！

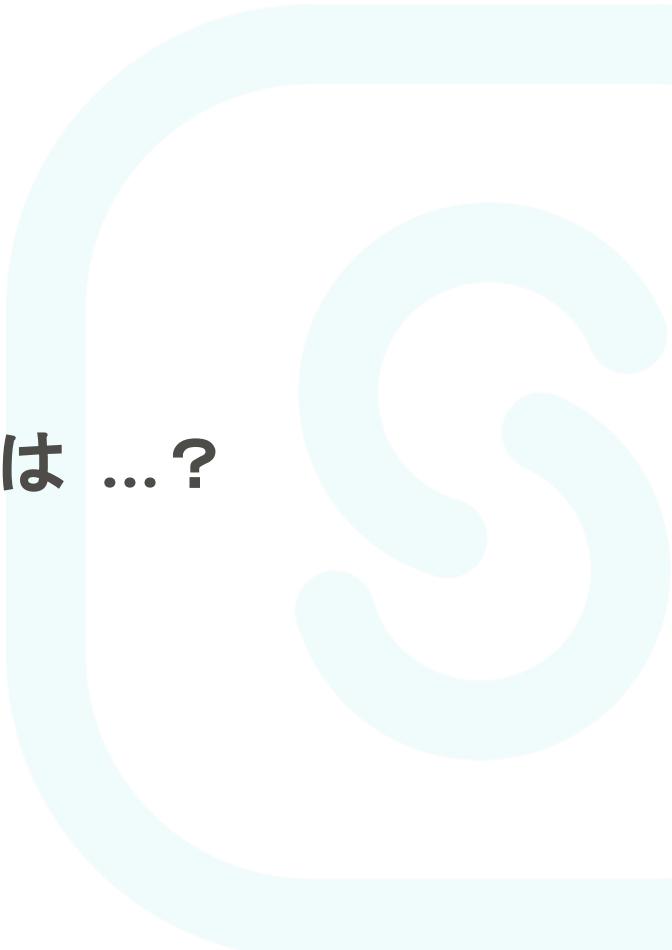
RubyKaigi と 仲良くなるには...?



改めて：言語実装技術は RubyKaigiの重要知識

- ・なんといっても、RubyのKaigiですからね...





言語実装技術を身につけるためには ... ?

言語実装を自分で始めてみては？

- ・色々説明したが、手を動かすのが最高に理解できる
- ・実際に小さなものを作ってみよう！
- ・昔からあるテーマなので参考文献もたくさん
- ・思ったよりも怖くないよ！



今日話していない 言語実装のテーマ

- 型
- その他静的解析
- 最適化
- GC
- AOTコンパイル ...



もうちょっとだけ続くのじゃ (AA略)

参考文献 (今回の個別の内容)

Lexer/Parserの参考スライド

- ・たのしいRubyの構文解析ツアー
- ・たのしいparse.y
- ・他、なぜかたくさん参考になる発表がありますね..



iSeq/VMの参考スライド・本・記事

- [RubyVM を PHP で実装する～Hello World を出力するまで～](#)
- [iSeq探訪 電子版](#)
- [YARV Maniacs【第2回】VMってなんだろう](#)

YJITの参考記事

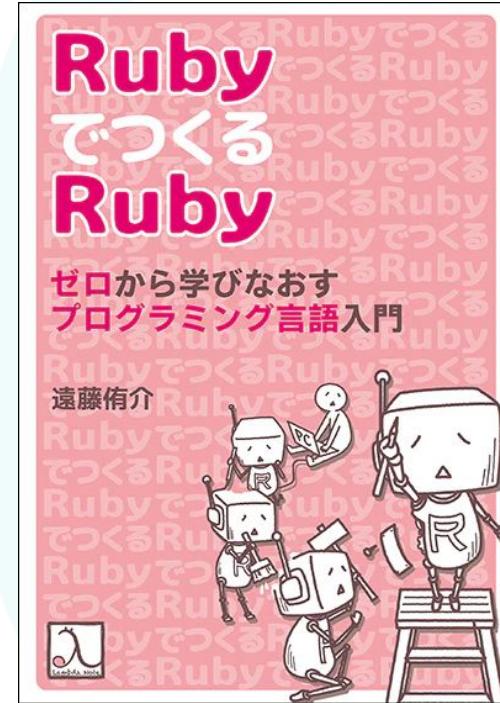
- ・[公式](#)
- ・...
- ・あとJIT一般は、[tenderloveのJIT実装の記事](#)とか？
- ・[mrubyのJITの話](#)とか？



参考文献 (言語実装一般)

RubyでつくるRuby

- ・よく話題に挙がる気がする
- ・レキサ、パーサの分量が多め



インタプリタの作り方

- ・今一冊ちゃんと読むならこれかなあ
- ・別言語(Rustとか)で実装し直すのも
おすすめ

CRAFTING INTERPRETERS | インタプリタの作り方

—言語設計／開発の基本と2つの方式による実装—

Robert Nystrom 著／吉川邦夫訳



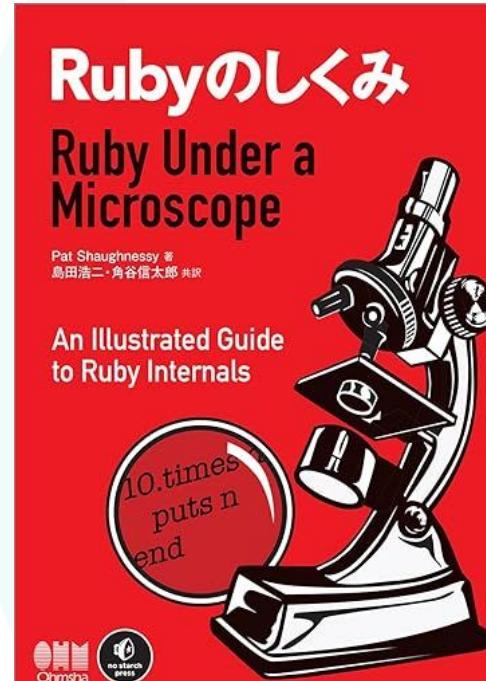
エンジニアに支持される一冊、待望の日本語版

動的型付けやクロージャを備えたモダンな言語をどのように実装するのか?!「字句解析」から「コンパイラコンパイラ」の使い方、「最適化」まで、JavaとCによる2つの実装で開発のエッセンスを解説

インプレス

Rubyのしくみ

- ・有名なやつ
- ・実装する、というよりRubyを通して
解説するという観点



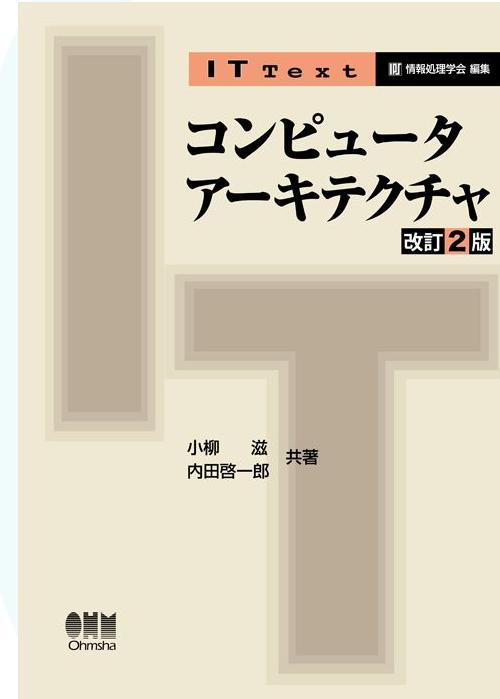
IT Textコンパイラとバーチャルマシン

・教科書なんだけど、普通におすすめ



IT Text コンピュータアーキテクチャ(改訂 2版)

- ・こっちも読んだ方がいいと思う
- ・CPUの気持ちになろう！



ふつうのコンパイラをつくろう

- ・網羅的で良い本なのですが
- ・いかんせん、全てが古い...

ふつうの コンパイラを つくろう

言語処理系をつくりながら学ぶ
コンパイルと実行環境の仕組み

青木峰郎 著



SoftBank Creative

低レイヤを知りたい人のための Cコンパイラ作成入門

- ・有名な自作コンパイラ記事
- ・C言語の勉強にもなるね！

動作環境など

利用した Ruby version

- ruby 3.4.2 (2025-02-15 revision d2930f8e7a) +YJIT +PRISM [arm64-darwin24]
- JITの結果検証環境のみ:
 - ruby 3.4.2 (2025-02-15 revision d2930f8e7a) +YJIT dev +PRISM [x86_64-linux]

言語実装の知識が 役立ちそうな発表

※ 僕GPTによる個人の予想です！

Main Hall #rubykaigiA	Sub Hall #rubykaigiB	Pearls Room #rubykaigiC
09:00 - 10:00 Door Open		
10:00 - 11:00 Between Character and Character Encoding  EKA Keynote Mari Imaiizumi @mariimizumi		
11:10 - 11:40 Make Parsers Compatible Using Automata Learning  JA Keynote Hiroto Fujinami @takorekouzai	Bringing Linux pfdif to Ruby Maciej Mensfeld @maciejmensfeld	Introducing Type Guard to Steep Takeshi KOMIYA @kominya
11:50 - 12:20 The Evolution of the CRuby Build System  Yuto Saito @kateinoigakuen	A side gig for RuboCop, the Bookworm code crawler  David T. Crosby @davetcrosby	Continuation is to be continued Masayuki Mizuno @burnerum
12:20 - 14:00 Lunch Break		
14:00 - 14:30 Deoptimization: How YJIT Speeds Up Ruby by Slowing Down  Tokashi Kokubun @kokubun	Empowering Developers with HTML-Aware ERB Tooling  Marco Roth @marcoroth	Goodbye fat gem 2025  Sutou Kouhei @khou
14:40 - 15:10 Ruby's Line Breaks  Yuchiyo Koneko @spikeleof	SDB: Efficient Ruby Stack Scanning Without the GVL  Mike Yang @yfractl	Automatically generating types by running tests  Tokumi Shotoroku @tsinsoku, lisy
15:10 - 15:40 Afternoon Break		
15:40 - 16:10 State of Namespace  Satoru Tagomori @tagomori	Embracing Ruby magic: Staticaly analyzing DSLs  Vittorio Strick @vitvits	50.000 processed records per second: a CRuby & JRuby story  Christian Planas @christian_planas
16:20 - 16:50 mruby/c and data-flow programming for small devices  Kazuaki Tanaka @katz055	Parsing and generating SQL's SQL dialect with Ruby  Stephan Mergheim @froctolemid	drRuby on Browser Again!  Yoh Oishi @youchan
17:00 - 18:00 TRICK 2025: Episode I  momo & the judges @tric		

Main Hall #rubykaigiA	Sub Hall #rubykaigiB	Pearls Room #rubykaigiC
10:00 - 11:00 Performance Bugs and Low-level Ruby Observability APIs  EKA Keynote Ivo Anja @neukx		
11:10 - 11:40 Dissecting and Reconstructing Ruby Syntactic Structures  JA Keynote Yuuki Tokuda @yukitoku	Benchmark and profile every single change  Daisuke Antono @osoyuu	Running JavaScript within Ruby  Kengo Hamasaki @fhemk
11:50 - 12:20 ZJIT: Building a Next Generation Ruby JITT  Maxim Chevalier-Blois @maximecb	Keeping Secrets: GitHub's ENV Security Learnings  Dennis Pocock @dennispocock	Improvement of REXML and speed up using StringScanner  NAITOH Jun @naitoh
12:20 - 14:00 Lunch Break		
14:00 - 14:30 Writing Ruby Scripts with TypeProf  Yusuke Endoh @mamehatter	Demystifying Ruby Debuggers: A Deep Dive into Internals  Dmitry Pogrebny @DmitryPogrebny	How to make the Groovebox  Ryuuji Fujiwara @sonoson
14:40 - 15:10 MicroRuby: True Microcontroller Ruby  Hitoishi HASUMI @hosumikin	Bazel for Ruby  Alex Rodionov @p0deje	RuboCop: Modularity and AST Insights  Koichi ITO @koic
15:10 - 15:40 Afternoon Break		
15:40 - 16:10 Speeding up Class#new  Aaron Patterson @henderlove	You Can Save Lives With End-to-end Encryption in Ruby  Ryo Koijima @yo1	Write you a Barrier - Automatic Insertion of Write Barriers  Martin J. Dürst @duerst
16:20 - 16:50 Making TCPSocket.new "Happy!"  Miaki Shiota @coe401	From C extension to pure C: Migrating RBS  Alexander Monchilov @amonomchilov	The Implementations of Advanced LR Parser Algorithm  Junichi Kobayashi @junko-kobayashi
17:00 - 18:00 Lightning Talks  RubyKaigi 2025 EN/JA		

Main Hall #rubykaigiA	Sub Hall #rubykaigiB	Pearls Room #rubykaigiC
09:50 - 11:00 Ruby Committers and the World  EN/JA CRuby Committers @rubylongorg		
11:10 - 11:40 API for docs  Soutaro Matsumoto @soutaro	Improving my own Ruby  monochrome @s_iishiki1969	Running ruby.wasm on Pure Ruby Wasm Runtime  uzuura @uzuura
11:50 - 12:20 Eliminating Unnecessary Implicit Allocations  Jeremy Evans @jeremyevans0	A taxonomy of Ruby calls  Alan Wu @alanwusx	The Ruby One-Binary Tool, Enhanced with Kompo  ohgappa @ohgappa
12:20 - 14:00 Lunch Break		
14:00 - 14:30 Toward Ractor local GC  Koichi Sakada @ko1	Inline RBS comments for seamless type checking with Sorbet  Alexandre Terra @Moriar	Road to Go gem  Go Sueyoshi @sue445
14:40 - 15:10 Analyzing Ruby Code in IRB  tomoya ishida @tomping	Optimizing JRuby 10  Charles Nutter @headius	Porting PicoRuby to Another Microcontroller: ESP32  @Y_uuu
15:10 - 15:40 Afternoon Break		
15:40 - 16:10 Modular Garbage Collectors in Ruby  Peter Zhu @pfeierzu2118	The Challenges of Building sigstore-ruby  Samuel Giddins @segiddins	On-the-fly Suggestions of Rewriting Method Deprecations  Masato Ohba @ohbarye
16:20 - 17:20 Matz Keynote  JA Keynote Yukihiko "Matz" Matsumoto @yukihiko_matz		