

# Rubyのサブセットを動作可能な仮想マシンの 発展と高速化

情報科学専攻 55組 1番 岡本八仁

# 目次

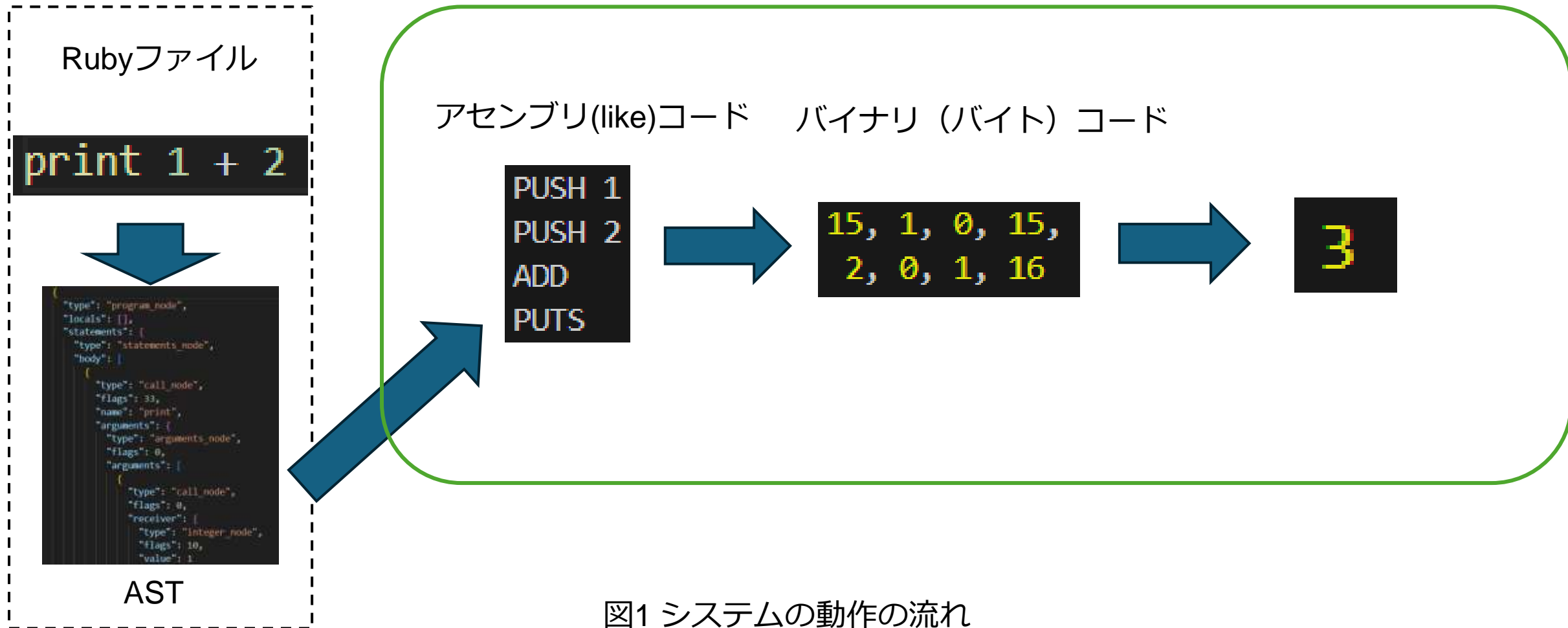
1. 前回の内容
2. 追加設計・実装
3. 高速化
4. おわりに

# 前回の内容

1. 前回の内容
2. 追加設計・実装
3. 高速化
4. おわりに

## 1. 前回の内容

# 動作の流れ



## 1. 前回の内容

# 対応不可能な構文

- クラス
- 一部の演算子
- コメント
- 一部の制御文
- メソッド（宣言/呼び出し）
- 標準入力

参考：Ruby 3.4 リファレンスマニュアル[1]

[1] <https://docs.ruby-lang.org/ja/3.4/doc/index.html>

# 追加設計・実装

1. 前回の内容
2. 追加設計・実装
3. 高速化
4. おわりに

# 新規対応

関数宣言/呼び出しに対応  
(再帰関数を含む)

- ・ **return文**が必ず書かれている
- ・ 呼び出す際の**引数の数**は宣言時と必ず等しい    という想定

## 2. 追加設計・実装

# 実装（アセンブリレベル）

追加で

- ・ RETURN命令
- ・ CALL命令（関数呼び出し） を実装

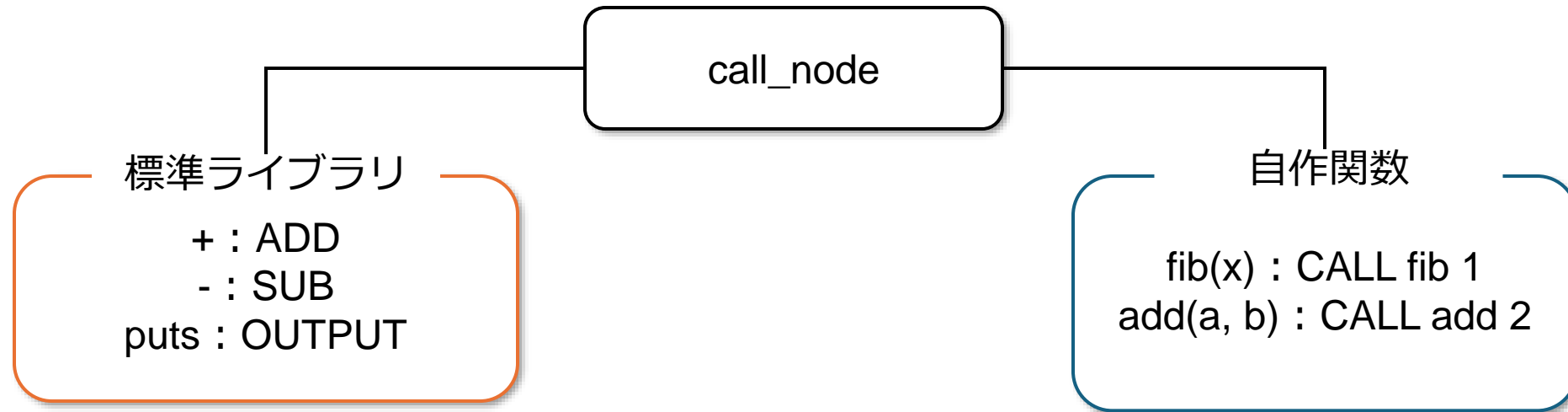


図2 call\_nodeの分類



## 2. 追加設計・実装

# 実装（アセンブリ→バイトコード）

関数の名前：引数の個数をセットで管理

CALL fib 1

➡ CALL命令の際に名前と引数の個数を渡す



1. ジャンプ命令で関数部分に飛ぶ
2. 引数の個数分だけスタックから取り出す

## 2. 追加設計・実装

# 実装（VM）

VMに新たな情報を追加

- callStack
  - ・ 関数呼び出し時のアドレスを保管
  - ・ RETURN命令で取り出しそこに戻る
- envStack
  - ・ 再帰関数用の旧レジスタ
  - ・ 階層ごとに変数の値を管理

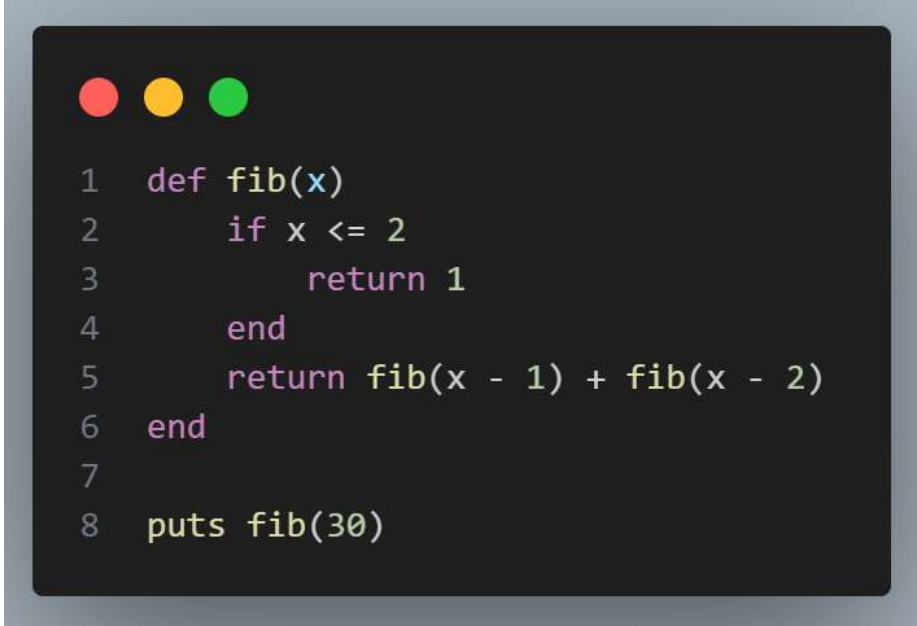
# 高速化

1. 前回の内容
2. 追加設計・実装
3. 高速化
4. おわりに

### 3. 高速化

## 速度計測用のコード

フィボナッチ数列の第30項を求めるRubyコード  
(再帰回数が多くなるため)



```
1  def fib(x)
2    if x <= 2
3      return 1
4    end
5    return fib(x - 1) + fib(x - 2)
6  end
7
8  puts fib(30)
```

図3 サンプルコード (main.rb)

### 3. 高速化

## 計測結果 (1/2)

各命令ごとの実行時間の合計と  
VM実行全体の実行時間を測定 (5回平均)

VM実行 : 4,535(ms)

表1 各命令ごとの実行時間

命令	時間 (μs)
PUSH_NUM	603,984
CALL	831,736
REFERENCE	565,880
LESS_EQUAL	175,353
JUMP_IF_FALSE	281,269
SUBTRACTION	178,349
RETURN	168,942
ADDITION	75,681

### 3. 高速化

## 高速化の方針

1番目に時間がかかっているCALL命令,  
2番目に時間がかかっているPUSH\_NUM命令を高速化する

CALL命令 ➤ envStackの更新方法改善

PUSH\_NUM命令 ➤ PUSH\_NUMの命令切り出し

### 3. 高速化

## envStackの更新方法改善

CALL命令によってスコープが変化するたびに,

```
const newEnv: { [key: number]: number } = {};
```

という「プレーンオブジェクト」を使用している

※ChatGPT調べ



TypeScript (JavaScript) のプレーンオブジェクトのアクセスは**かなり遅い**



シンプルな配列

```
const newEnv: number[] = [];
```

に変更

### 3. 高速化

## PUSH\_NUMの命令切り出し

今回使用しているサンプルコードでは、  
「1」と「2」が頻繁に使用される

これらを固有の命令に



PUSH\_NUM 1    ➡    NUMBER\_1

```
1 def fib(x)
2   if x <= 2
3     return 1
4   end
5   return fib(x - 1) + fib(x - 2)
6 end
7
8 puts fib(30)
```

図3 サンプルコード (main.rb)

オペランドから値を読む必要がないため、動作の高速化が期待される



### 3. 高速化

## 計測結果 (2/2)

#### 高速化を実行した後の

各命令ごとの実行時間の合計と

VM実行全体の実行時間を測定 (5回平均)

VM実行 : 3,885(ms)

表2 高速化後の各命令ごとの実行時間

命令	時間 (μs)
PUSH_NUM	332,902
CALL	499,553
REFERENCE	578,089
LESS_EQUAL	176,794
JUMP_IF_FALSE	279,603
SUBTRACTION	166,259
RETURN	154,345
ADDITION	840,69

### 3. 高速化 比較

表1 各命令ごとの実行時間

命令	時間 (μs)
PUSH_NUM	603,984
CALL	831,736
REFERENCE	565,880
LESS_EQUAL	175,353
JUMP_IF_FALSE	281,269
SUBTRACTION	178,349
RETURN	168,942
ADDITION	75,681

VM実行 : 4,535(ms)

表2 高速化後の各命令ごとの実行時間

命令	時間 (μs)
PUSH_NUM	332,902
CALL	499,553
REFERENCE	578,089
LESS_EQUAL	176,794
JUMP_IF_FALSE	279,603
SUBTRACTION	166,259
RETURN	154,345
ADDITION	840,69

VM実行 : 3,885(ms)

### 3. 高速化 考察

※VM全体の実行時間

高速化前 : 4,556(ms), 4,471(ms), 4,550(ms), 4,514(ms), 4,535(ms)

高速化後 : 3,868(ms), 3,791(ms), 3,792(ms), 3,848(ms), 3,902(ms)

のデータに対して「対応のあるt検定」（有意水準5%）を行った結果,

p値 =  $8.7e-6 < 0.05$

となり, 実行時間が短縮されていることが有意に示された.

# おわりに

1. 前回の内容
2. 追加設計・実装
3. 高速化
4. おわりに

### 3. おわりに

# まとめ

- ・ VMを関数宣言/呼び出しに対応
- ・ CALL命令, PUSH\_NUM命令の高速化などを行った.

➡ 対応のあるt検定の結果, 実行速度に有意差が認められた.

- ・ レジスタマシンとして実装
  - ・ 呼び出し頻度の高い命令の置き換え
- 等の高速化を追加で行えるとさらに改善が見込める

# 参考文献

[1] Ruby:Ruby3.4 リファレンスマニュアル, <https://docs.ruby-lang.org/ja/3.4/doc/index.html>

# 目次

1. 前回の内容 (pp. 3-5)
2. 追加設計・実装 (pp. 6-10)
3. 高速化 (pp. 11-19)
4. おわりに (pp. 20-21)

以下，補足スライド



### 3. 高速化

## 計測結果：個別

表3 (envStackのみ改善後の) 各命令ごとの実行時間

命令	時間 (μs)
PUSH_NUM	626,177
CALL	470,209
REFERENCE	561,989
LESS_EQUAL	181,425
JUMP_IF_FALSE	269,006
SUBTRACTION	175,247
RETURN	171,543
ADDITION	78,244

VM実行：4,229(ms)

表4 (PUSH\_INTのみ改善後の) の各命令ごとの実行時間

命令	時間 (μs)
PUSH_NUM	326,662
CALL	827,337
REFERENCE	580,073
LESS_EQUAL	180,531
JUMP_IF_FALSE	290,886
SUBTRACTION	161,086
RETURN	150,003
ADDITION	89,002

VM実行：4,245(ms)