TruffleでPHPぽい言語を実装したら爆速だった話

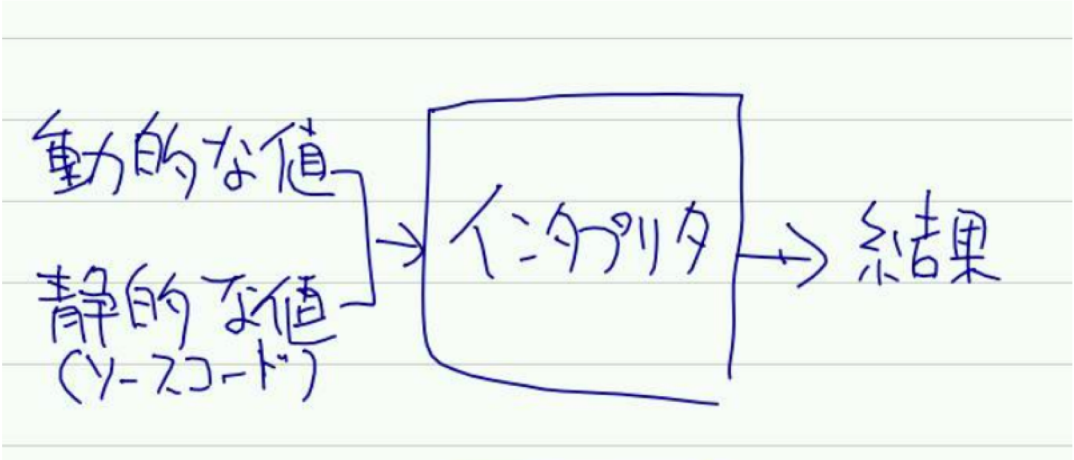
JJUG ナイトセミナー
2019/2/27 LINE Fukuoka きしだ なおき

自己紹介

- きしだ なおき
- LINE Fukuoka
- 昼間起きれない期間
- @kis

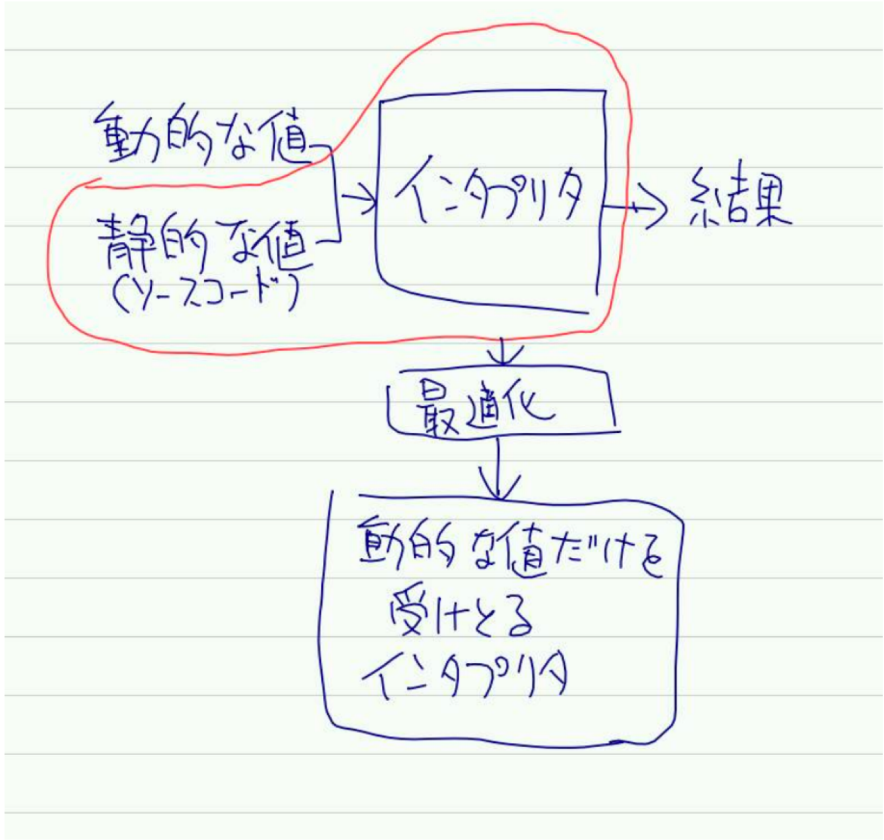
Truffleのコンセプト

- インタプリタがある

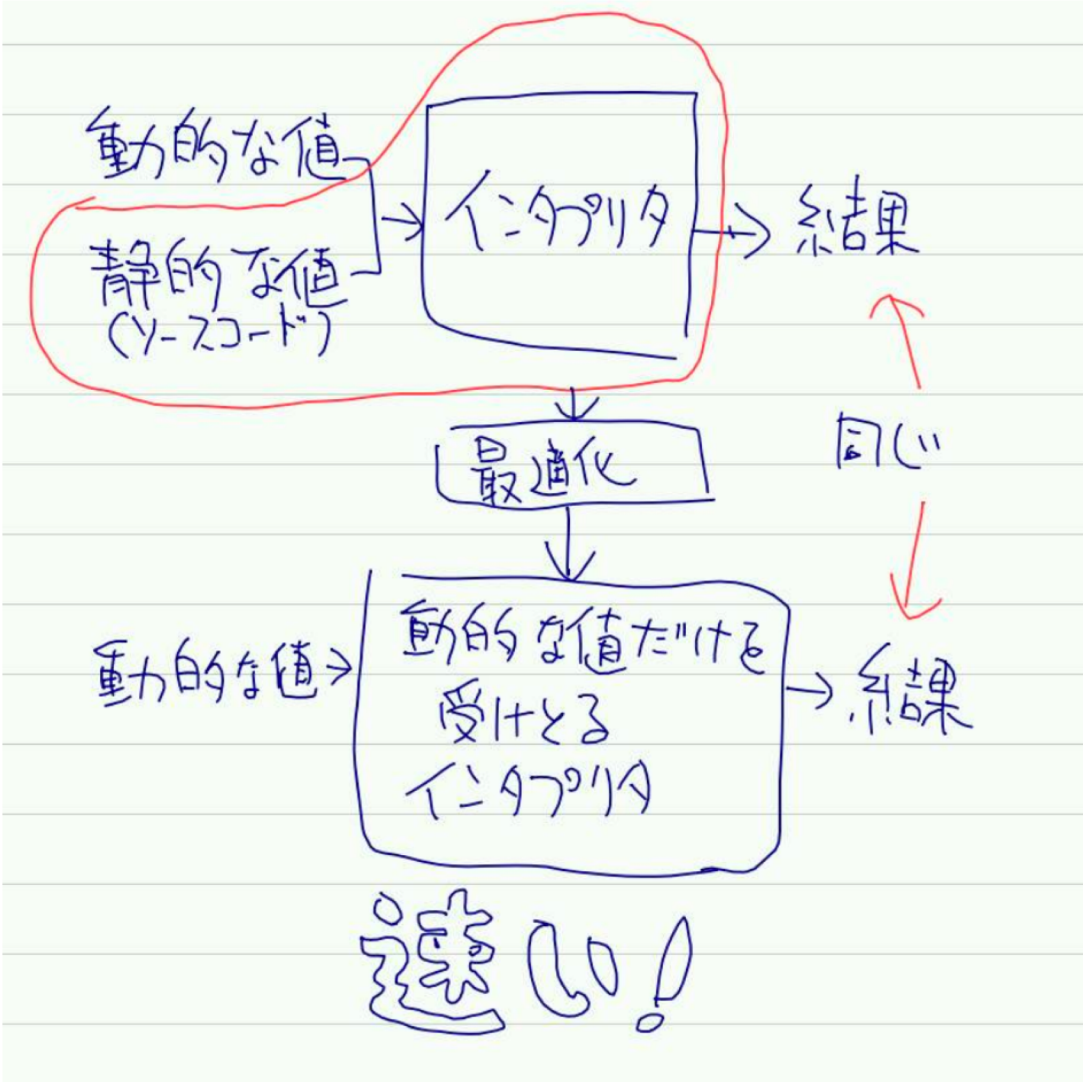


Truffleのコンセプト

- 静的な値(ソース)とインタプリタを受け取って
  動的な値だけを受け取るインタプリタを作る最適化エンジン
  がある



Truffleのコンセプト



PHPっぽい言語を実装してみた

- jparsecを使う(外部の構文定義が不要)
  - https://github.com/jparsec/jparsec/
- 数値と文字列
- +と-と<と>と.(文字列連結)
- 変数
- 関数
- if
- echo

https://github.com/kishida/phpparser

つまりフィボナッチが書ける

```
<?php
function fib($n) {
    if ($n < 2) {
        return $n;
    }
    return fib($n - 1) + fib($n - 2);
}
echo "fib:".fib(31);
echo "¥n";
```

コードの特徴(1/6)

- Nodeを継承する

```
@NodeInfo(language = "TrufflePHP", description = "Base class for PHP")
@TypeSystemReference(PHPTypes.class)
static abstract class PHPNode extends Node {
}
```

コードの特徴(2/6)

- 必要な型を返すメソッドが自動的によびだされる
  - 特殊化(Specialization)

```
@AllArgsConstructor
static class DoubleNode extends PHPExpression {
    final double value;

    @Override
    double executeDouble(VirtualFrame vf) throws UnexpectedResultException {
        return value;
    }

    @Override
    Object executeGeneric(VirtualFrame virtualFrame) {
        return value;
    }
```

コードの特徴(3/6)

- メソッド呼び出しもCallNodeを使うと最適化される

```
static class PHPInvokeNode extends PHPExpression {
    FunctionObject function;
    @Children PHPExpression[] argValues;
    @CompilerDirectives.CompilationFinal
    DirectCallNode callNode;

    public PHPInvokeNode(FunctionObject function, PHPExpression[] argValues) {
        this.function = function;
        this.argValues = argValues;
    }

    @Override
    @ExplodeLoop
    Object executeGeneric(VirtualFrame virtualFrame) {

        CompilerAsserts.compilationConstant(argValues.length);
        Object[] args = new Object[argValues.length];
        for (int i = 0; i < argValues.length; ++i) {
```

コードの特徴(4/6)

- ヒントを与えると展開されたりする

```
@NodeInfo(shortName = "block")
@AllArgsConstructor
static class PHPBlock extends PHPStatement {
    @Children PHPStatement[] statements;

    @Override
    @ExplodeLoop
    void executeVoid(VirtualFrame virtualFrame) {
        CompilerAsserts.compilationConstant(statements.length);
        for (PHPStatement st : statements) {
            st.executeVoid(virtualFrame);
        }
    }
```

コードの特徴(5/6)

- NodeとかDirectCallNodeとかCompilerAssertsとか、Graalにもある
  - Javaの最適化と同じ構造

```
30
31 import org.graalvm.compiler.core.common.type.Stamp;
32 import org.graalvm.compiler.core.common.type.StampFactory;
33 import org.graalvm.compiler.core.common.type.StampPair;
34 import org.graalvm.compiler.core.common.type.TypeReference;
35 import org.graalvm.compiler.graph.NodeClass;
36 import org.graalvm.compiler.graph.NodeInputList;
37 import org.graalvm.compiler.nodeinfo.NodeInfo;
38 import org.graalvm.compiler.nodes.spi.LIRLowerable;
39 import org.graalvm.compiler.nodes.spi.NodeLIRBuilderTool;
40
41 import jdk.vm.ci.meta.Assumptions;
42 import jdk.vm.ci.meta.JavaKind;
43 import jdk.vm.ci.meta.JavaType;
44 import jdk.vm.ci.meta.ResolvedJavaMethod;
45 import jdk.vm.ci.meta.ResolvedJavaType;
46
47 @NodeInfo(allowedUsageTypes = Extension, cycles = CYCLES_0, size = SIZE_0)
48 public abstract class CallTargetNode extends ValueNode implements LIRLowerable {
49     public static final NodeClass<CallTargetNode> TYPE = NodeClass.create(CallTargetNode.class);
50
51     public enum InvokeKind {
52         Interface(false),
53         Special(true),
54         Static(true),
```

コードの特徴(6/6)

- 書いたとおりに動いてない！
  - 最適化されるので、重そうなコードを書いても実際は最適化された状態で動く。
  - returnは例外で実現するけどたぶんgoto(JVMコード)になり、jmp(ネイティブ)になりReturnExceptionのオブジェクトは生成されない。もちろんresultもObjectではなく特殊型になるはず

```
@AllArgsConstructor @Getter
static class ReturnException extends ControlFlowException {
    Object result;
}

@NodeInfo(shortName = "return")
@AllArgsConstructor
static class PHPReturnNode extends PHPStatement {
    @Child private PHPExpression result;

    @Override
    void executeVoid(VirtualFrame virtualFrame) {
        Object value = result.executeGeneric(virtualFrame);
        throw new ReturnException(value);
    }
}
```

ベンチマークをとってみる

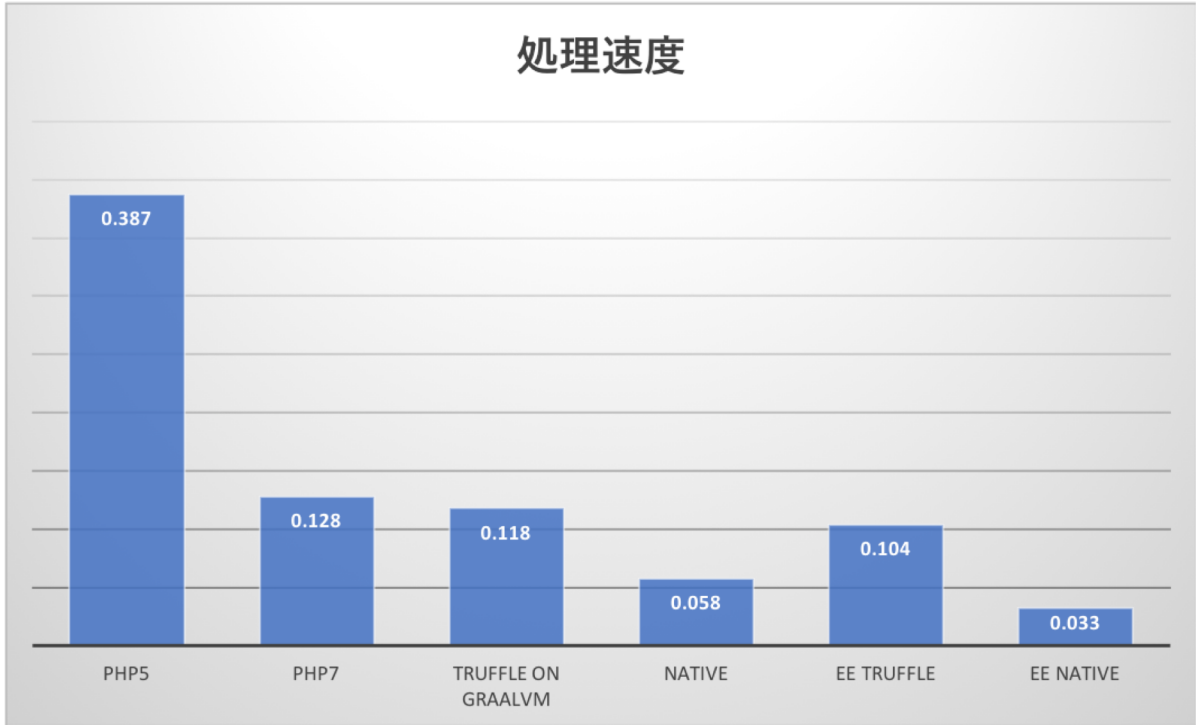
```
<?php function fib($n) {
    if ($n < 2) {
        return $n;
    }
    return fib($n - 1) + fib($n - 2);
}
$start = microtime(1);
echo "fib:".fib(31);
echo "¥n";
echo "time:".(microtime(1) - $start);
echo "¥n";

fib(31);fib(31);fib(31);fib(31);fib(31);
$start = microtime(1);
echo "fib:".fib(31);
echo "¥n";
echo "time:".(microtime(1) - $start);
echo "¥n";
```

動かした環境

- 通常のスクリプト処理(interpreter) on JDK 11
- Truffleで書いたものをJDK 11で動かす
- Truffleで書いたものをGraalVM CEで動かす
- Truffleで書いたものをGraalVM CEでネイティブ化して動かす
- Truffleで書いたものをGraalVM EEで動かす
- Truffleで書いたものをGraalVM EEでネイティブ化して動かす
- PHP 5.6で動かす
- PHP 7.3で動かす

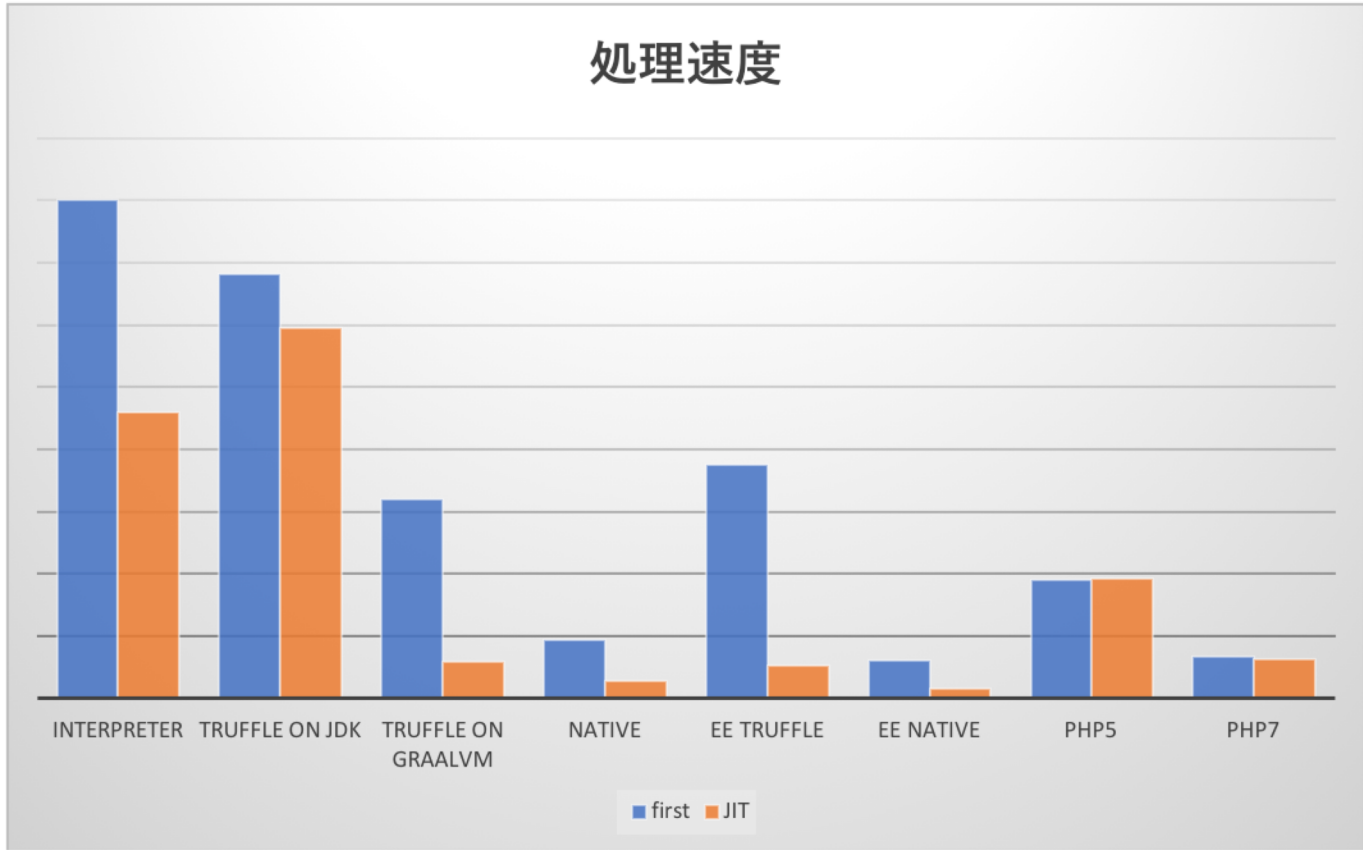
結果(JITが効いた状態)

- PHP 7.3はPHP5.6の3倍速い
- インタプリタはPHP7.3の7倍遅い
- Truffle+GraalVM CEはPHP7.3と同じくらい
- Truffle+GraalVM EEはPHP7.3の2割速い
- GraalVM CEのネイティブイメージはPHP7.3の2倍速い
- GraalVM EEのネイティブイメージはPHP7.3の4倍速い



だまされてない？

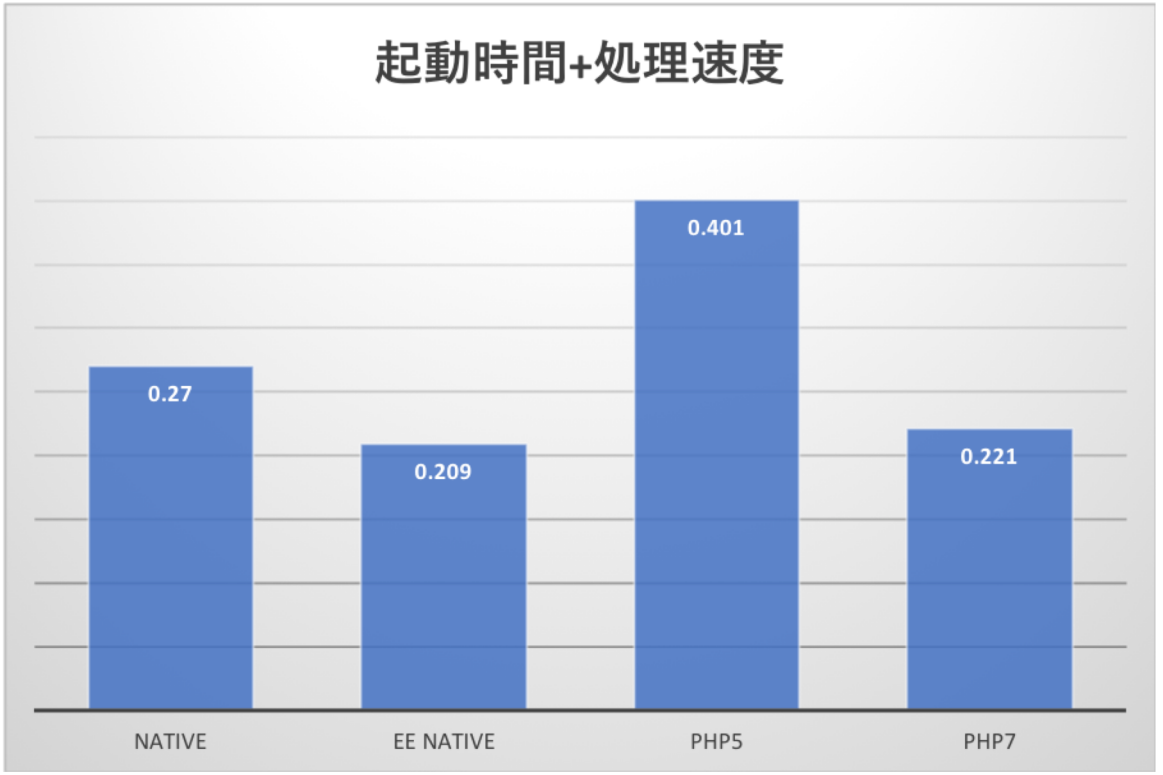
- JITが効いてないときは？
  - ネイティブ以外は
    だいたい遅い
  - ネイティブは
    - EEはPHP7.3よりちょい速
    - CEはPHP7.3よりちょい遅



起動時間も含めたら？

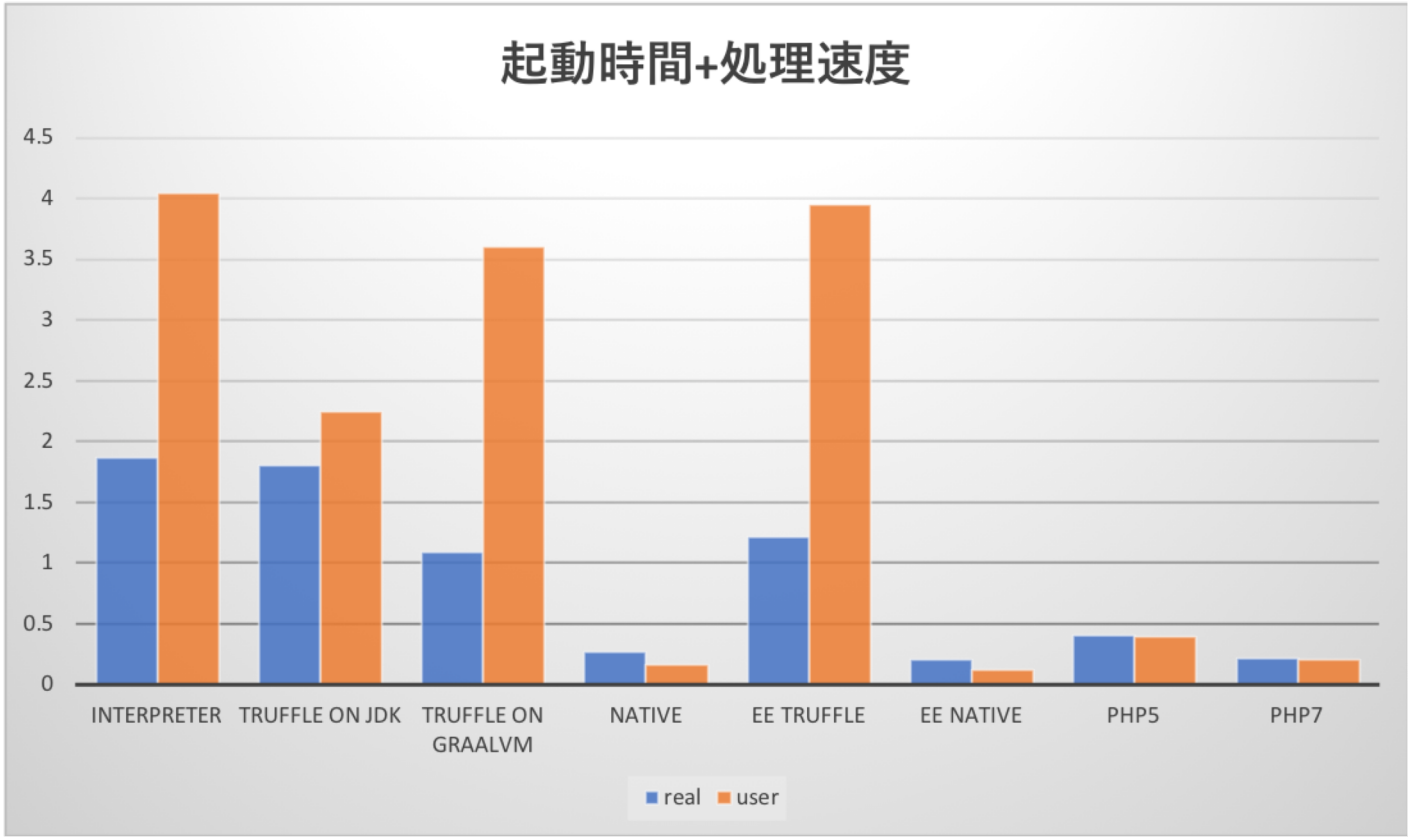
- PHPはリクエストごとに呼び出されるので起動時間も重要
- ネイティブ以外はだいたい遅い
- ネイティブは
  - EEちょい速、CEちょい遅

```
<?php
function fib($n) {
    if ($n < 2) {
        return $n;
    }
    return fib($n - 1) + fib($n - 2);
}
echo "fib:".fib(31);
echo "¥n";
```



もうすこし詳しく

- Realは処理にかかった時間、UserはCPUが使った時間
- JVM系はUser時間がすごく長い
  - マルチスレッドで最適化してるのでCPU時間が長い
- GraalVMはRealとUserの差が大きい
  - Graal自体の最適化にもCPU時間がかかってる？



とはいえ

- PHPはPHP3からPHP4、PHP5からPHP7になるときにすごく最適化をがんばっている。(PHP6は欠番)
- 実装が少ないとはいえ、たどたどしく書いた処理系がそのPHP7に匹敵する速さになるのはすごい
- いろんな型を実装すると遅くなるのでは？
  - 正しく最適化されれば実装が少ないときと同じ状態になるはず
  - 適切な型によって特殊化(Specialization)される

まとめ

- GraalVMたのしい！
- Truffleすごくたのしい！
  - ドキュメントはほぼないので、SimpleLanguageからコピペする
  - Writing a Language in Truffle
  - わかんないことはgitterでチャットで聞く
- Graalも勉強しなくては！