The following presentation is provided under DHMB license

(Do Not Hurt My Brain).

Lecturer is not responsible for the financial and moral damages resulting from taking too seriously the content of the presentation.

Including permanent damage to neurons, reduction of neurotransmitter activity at the molecular level and group dismissal.

Patterns for JVM languages

Jarosław Pałka

about me

work://chief_architect@lumesse

owner://symentis.pl

twitter://j_palka

blog://geekyprimitives.wordpress.com

scm:bitbucket://kcrimson

scm:github://kcrimson

Pet project

Because having pet projects is like solving crosswords everyday

Or going to a gym

Helps your brain to stay in shape


Xtend


Scala


JRuby


Groovy

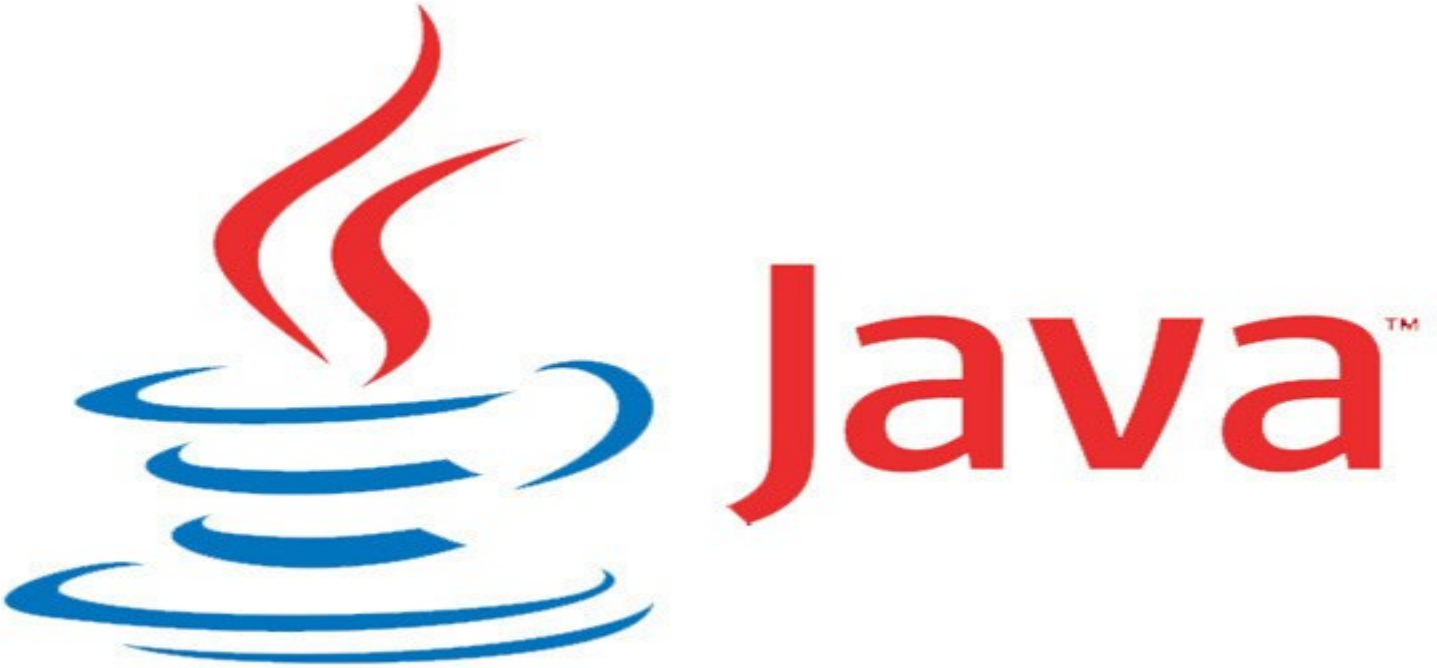

Clojure


Ceylon


JavaScript


PROJECT
Kotlin


Jython


Java™

JVM

is

the

new

assembler

Translator

source code

↓

target language source code

↓

machine code || interpreter

Interpreter

source code

execution results

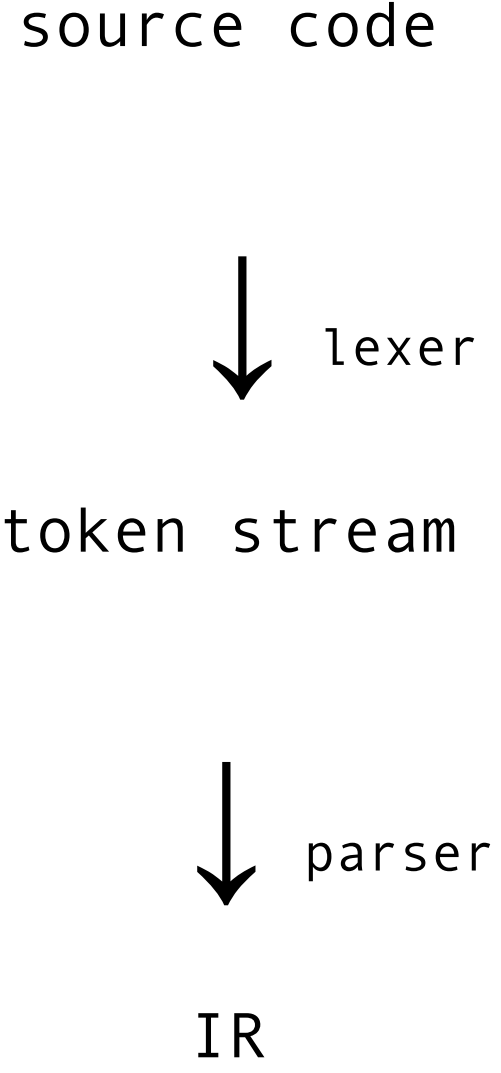
Compiler

source code

machine code

So how does it all work?

Diving into the stream


source code
lexer
token stream
parser
IR

Intermediate Representation

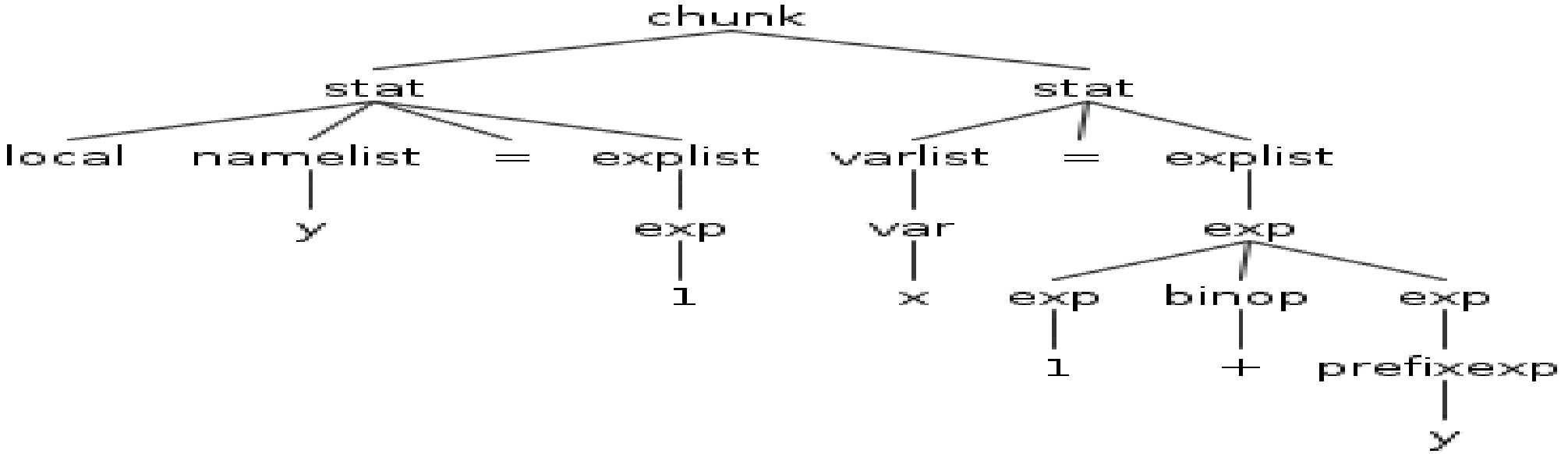
visitor

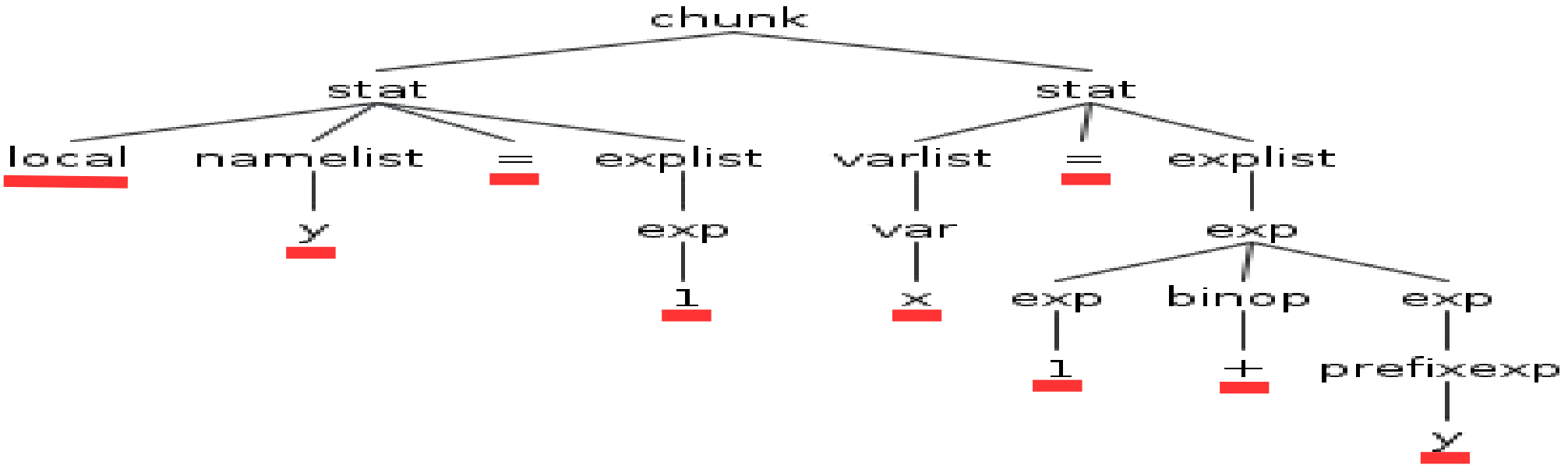
listener

tree

Syntax tree

```
local y = 1
x = 1 + y
```


chunk
stat
stat
local
namelist
=
explist
varlist
=
explist
y
exp
var
exp
1
x
exp
binop
exp
1
+
prefixexp
y


chunk
stat
stat
local
namelist
=
explist
varlist
=
explist
y
exp
var
exp
1
x
exp
binop
exp
1
+
prefixexp
y


ANTLR

```
fragment
DIGIT
:
[0-9]
;

fragment
LETTER
:
[a-zA-Z]
;

String
:
'\'' .*? '\''
| '"' .*? '"'
;

Number
:
DIGIT+
;

Name
:
(
' '
 _
| LETTER
)
(
' '
 _
| LETTER
| DIGIT
)*
;

COMMENT
:
'--' .*? '\n' -> skip
;

NL
:
'\r'? '\n' -> skip
;

WS
:
[ \t\f]+ -> skip
;
```

```
[@0,[0..1)='x',<49>]
[@1,[2..3)='=',<36>]
[@2,[4..5)='1',<48>]
[@3,[6..7)='+',<33>]
[@4,[8..15)='""Hello""',<47>]
[@5,[15..16)=';',<37>]
[@6,[17..17)='',<-1=EOF>]
```

```
varlist returns [Varlist result]
:
var
{$result = createVarlist($var.result);}

(
',' var
{$result = createVarlist($result,$var.result);}

)*
;

//

var returns [Var result]
:
Name
{ $result = createVariableName($Name.text); }

| prefixexp '[' exp ']'
| prefixexp '.' Name
;
```

beware of left recursion

~~translator~~

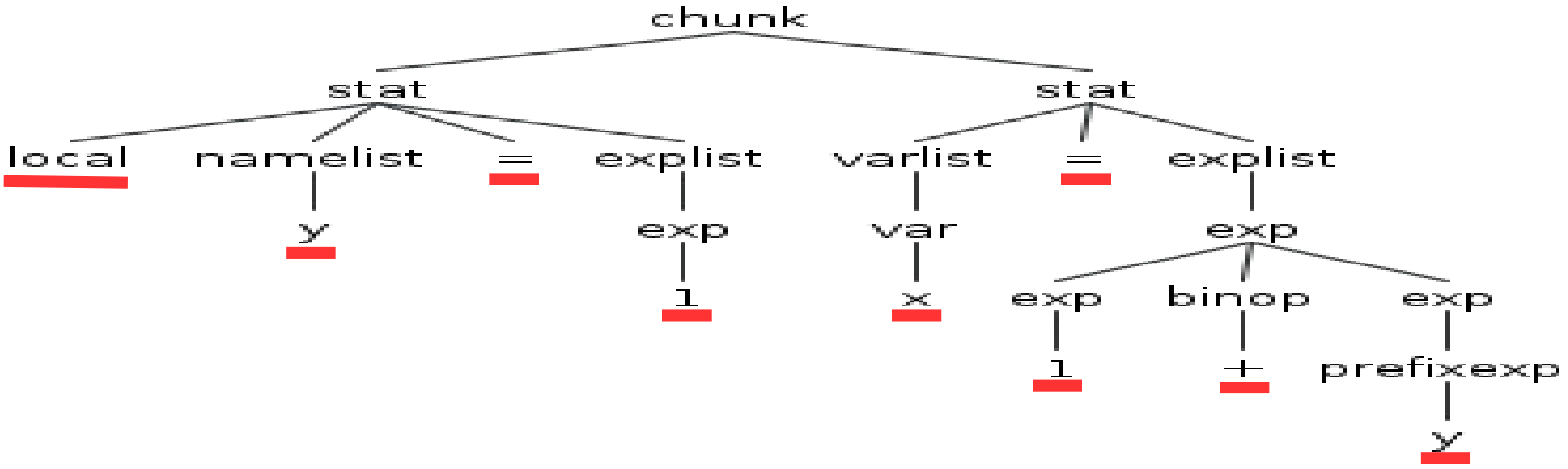
interpreter

compiler

Interpreter pattern

Non terminal nodes

Terminal nodes


chunk
stat
stat
local
namelist
=
explist
varlist
=
explist
y
exp
var
exp
1
x
exp
binop
exp
1
+
prefixexp
y

```
package pl.symentis.lua.grammar;

import pl.symentis.lua.runtime.Context;

public interface Statement {

    void execute(Context context);

}
```

```
package pl.symentis.lua.grammar;

Import pl.symentis.lua.runtime.Context;

public interface Expression<T> {

    T evaluate(Context ctx);

}
```

```java
package pl.symentis.lua.runtime;

public interface Context {

    VariableRef resolveVariable(String name);

    Scope enterScope();

    Scope exitScope();

}
```

```java
package pl.symentis.lua.runtime;

public interface Scope {

    Scope getOuterScope();

    VariableRef resolveVariable(String name);

    void bindVariable(String name, Object value);

}
```


Into the forest of syntax tree

$$x = 1$$

```java
public class Literal implements ExpressionNode {

    public final LuaType value;

    public Literal(LuaType value) {
        super();
        this.value = value;
    }

    @Override
    public void accept(ExpressionVisitor visitor) {
        visitor.visit(this);
    }

}
```

```
public class VariableName extends Var {

    public final String name;

    public VariableName(String name) {
        this.name = name;
    }

    @Override
    public void accept(ExpressionVisitor visitor) {
        visitor.visit(this);
    }

}
```

```java
public class Assignment implements StatementNode {

    public final Varlist varlist;

    public final Explist explist;

    public Assignment(Varlist varlist, Explist explist) {
        super();
        this.varlist = varlist;
        this.explist = explist;
    }

    @Override
    public void accept(StatementVisitor visitor) {
        visitor.visit(this);
    }

}
```

```
public void visit(Assignment assignment) {
    InterpreterExpressionVisitor varlist = createExpressionVisitor();
    assignment.varlist.accept(varlist);

    InterpreterExpressionVisitor explist = createExpressionVisitor();
    assignment.explist.accept(explist);

    Iterator<ExpressionNode> iterator = ((List<ExpressionNode>) explist.result()).iterator();

    List<VariableRef> vars = (List<VariableRef>) varlist.result();
    for (VariableRef variableRef : vars) {
        if (iterator.hasNext()) {
            ExpressionNode node = iterator.next();

            InterpreterExpressionVisitor exp = createExpressionVisitor();
            node.accept(exp);

            variableRef.set(exp.result());
        } else {
            variableRef.set(LuaNil.Nil);
        }
    }
}
```

```
public void visit(Literal literal) {
    result = literal.value;
}

public void visit(VariableName variableName) {
    result = context.resolveVariable(variableName.name);
}

public void visit(Varlist varlist) {

    List<VariableRef> refs = new ArrayList<>();

    for (Var var : varlist.vars) {
        var.accept(this);
        refs.add((VariableRef) result());
    }

    result = refs;
}
```

~~translator~~

~~interpreter~~

compiler

compiler → byte code

`org.ow2.asm:asm:jar`

`me.qmx.jitescript:jitescript:jar`

```
JiteClass jiteClass = new JiteClass(classname, new String[] { p(Script.class) });

jiteClass.defineDefaultConstructor();

CodeBlock codeBlock = newCodeBlock();

// bind context into current runtime
codeBlock.aload(1);
codeBlock.invokestatic(p(RuntimeContext.class), "enterContext", sig(void.class, Context.class));

// parse file and start to walk through AST visitor
StatementNode chunk = parseScript(readFileToString(script));

// generate byte code
chunk.accept(new CompilerStatementVisitor(codeBlock, new HashSymbolTable()));

codeBlock.voidreturn();

jiteClass.defineMethod("execute", JiteClass.ACC_PUBLIC, sig(void.class, new Class[]
{ Context.class }),
        codeBlock);

return jiteClass.toBytes(JDKVersion.V1_8);
}
```

```
JiteClass jiteClass = new JiteClass(classname, new String[] { p(Script.class) });

jiteClass.defineDefaultConstructor();

CodeBlock codeBlock = newCodeBlock();

// bind context into current runtime
codeBlock.aload(1);
codeBlock.invokestatic(p(RuntimeContext.class), "enterContext", sig(void.class, Context.class));

// parse file and start to walk through AST visitor
StatementNode chunk = parseScript(readFileToString(script));

// generate byte code
chunk.accept(new CompilerStatementVisitor(codeBlock, new HashSymbolTable()));

codeBlock.voidreturn();

jiteClass.defineMethod("execute", JiteClass.ACC_PUBLIC, sig(void.class, new Class[]
{ Context.class }),
        codeBlock);

return jiteClass.toBytes(JDKVersion.V1_8);
}
```

```
JiteClass jiteClass = new JiteClass(classname, new String[] { p(Script.class) });

jiteClass.defineDefaultConstructor();

CodeBlock codeBlock = newCodeBlock();

// bind context into current runtime
codeBlock.aload(1);
codeBlock.invokestatic(p(RuntimeContext.class), "enterContext", sig(void.class, Context.class));

// parse file and start to walk through AST visitor
StatementNode chunk = parseScript(readFileToString(script));

// generate byte code
chunk.accept(new CompilerStatementVisitor(codeBlock, new HashSymbolTable()));

codeBlock.voidreturn();

jiteClass.defineMethod("execute", JiteClass.ACC_PUBLIC, sig(void.class, new Class[]
{ Context.class }),
        codeBlock);

return jiteClass.toBytes(JDKVersion.V1_8);
}
```

$$x = 1$$

```
public void visit(Assignment assignment) {

Explist explist = assignment.explist;
Varlist varlist = assignment.varlist;
Iterator<Var> vars = varlist.vars.iterator();

for (ExpressionNode exp : explist.expressions) {
    exp.accept(new CompilerExpressionVisitor(codeBlock, symbolTable));

    CompilerExpressionVisitor visitor = new CompilerExpressionVisitor(
                                                codeBlock, symbolTable);

    vars.next().accept(visitor);

    String varName = (String) visitor.result();
    codeBlock.astore(symbolTable.index(varName));
}

}
```

```
public void visit(Literal literal) {

Object object = literal.value.asObject();

codeBlock.ldc(object);

// box object into Lua type
if (object instanceof Integer) {
      codeBlock.invokestatic(p(Integer.class), "valueOf", sig(Integer.class, int.class));
} else if (object instanceof Double) {
      codeBlock.invokestatic(p(Double.class), "valueOf", sig(Double.class,
                                                            double.class));
}

codeBlock.invokestatic(p(LuaTypes.class), "valueOf", sig(LuaType.class, Object.class));

}

public void visit(VariableName variableName) {
      result = variableName.name;
}
```

When writing bytecode compiler it helps to think about Java code

But at the end you need to think like stack machine :)

```
.method public execute : (Lpl/symentis/lua4j/runtime/Context;)V
        ; method code size: 21 bytes
        .limit stack 2
        .limit locals 3
        aload_1
        invokestatic pl/symentis/lua4j/compiler/RuntimeContext
enterContext (Lpl/symentis/lua4j/runtime/Context;)V
        ldc2_w 1.0
        invokestatic java/lang/Double valueOf (D)Ljava/lang/Double;
        invokestatic pl/symentis/lua4j/types/LuaTypes valueOf [_28]
        astore_2
        aload_2
        return
.end method
```

Symbol table

maps variable local to its symbolic name

0xBA

invokedynamic

```
print("Welcome","to","Geecon",2014)
```

```
Var var = functionCall.var;

CompilerExpressionVisitor visitor = new CompilerExpressionVisitor(codeBlock,
symbolTable);
var.accept(visitor);
String functionName = (String) visitor.result();

Args args = functionCall.args;

visitor = new CompilerExpressionVisitor(codeBlock, symbolTable);
args.accept(visitor);
Integer argNumber = (Integer) visitor.result();

Class<?>[] paramTypeArr = new Class[argNumber];
fill(paramTypeArr, LuaType.class);

codeBlock.invokedynamic(functionName, sig(void.class, paramTypeArr),
Lua4JDynamicBootstrap.HANDLE,
        new Object[] {});
```

```
static final Handle HANDLE = new Handle(
        CodeBlock.H_INVOKESTATIC,
        p(Lua4JDynamicBootstrap.class),
        "bootstrap",
        sig(CallSite.class, MethodHandles.Lookup.class, String.class,
                                                        MethodType.class));
```

```
public static CallSite bootstrap(
    MethodHandles.Lookup lookup,
    String name,
    MethodType type) throws Throwable {

    Context context = currentContext();

    MethodHandle target=lookup.findStatic(BasicFunctions.class,name,
                                          methodType(
                                                  LuaType.class,
                                                  Context.class,
                                                  LuaType[].class));

    target = MethodHandles.insertArguments(handler, 0, context)
                          .asCollector(LuaType[].class, type.parameterCount());

    return new ConstantCallSite(target.asType(type));
}
```

https://github.com/headius/invokebinder

A Java DSL for binding method handles forward, rather than backward

https://github.com/projectodd/rephract

Another Java DSL for invokedynamic

What's next?

graal + truffle

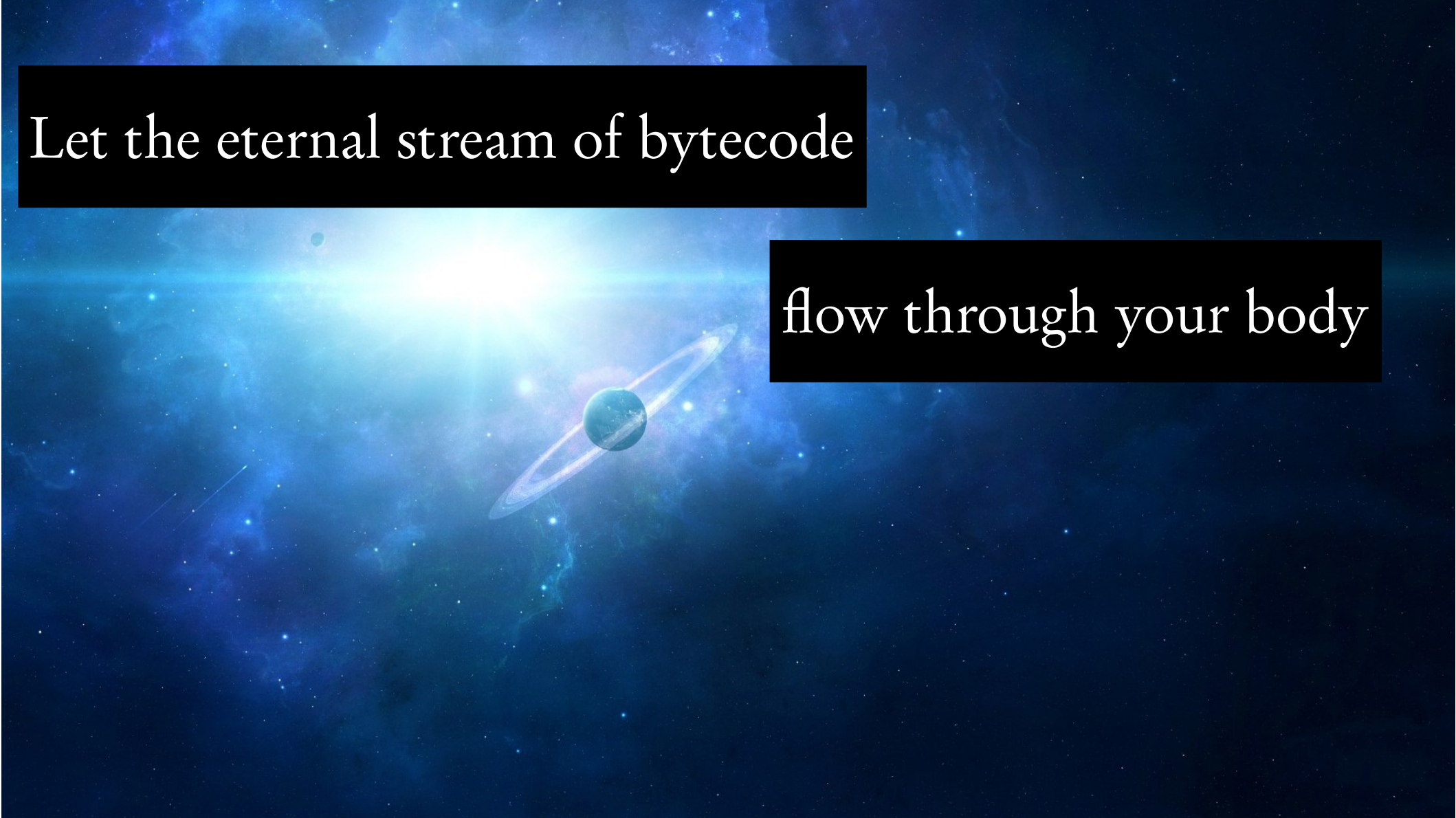
dynamic compiler + AST interpreter

Thanks,

stay cool,

dynamically linked

and...


Let the eternal stream of bytecode
flow through your body

Q&A