LANGUAGE ASPECTS

Modularity and API Design

2015/01 Tulach, Würthinger, Stadler

LANGUAGE ASPECTS

- Many languages offer standard module system
  - Java, node.js, Ruby, R, Perl, .Net, …
  - Some have de-facto standards for APIs
- Communities have certain ways of doing things

LANGUAGES

- API concepts can be applied in different ways:
  - for different languages
  - for different language versions
  - for different performance needs
- APIs evolve along with the languages they are written in
- Example: Java, JavaScript, R

LANGUAGES

- Example: result of a complex expression as parameter, only evaluate it if needed

- Basic (non-lazy) Java version

```
void foo(String[] value) {
    // ...
    if (condition) {
        doSomethingWith(value);
    }
    // ...
}

void main(String[] args) {
    foo(complexComputation(args));
}
```

LANGUAGES

- Java version using interface

```
interface LazyValue {
    Object get();
}

void foo(LazyValue value) {
    // ...
    if (condition) {
        Object evaluated = value.get();
        doSomethingWith(evaluated);
    }
    // ...
}

void main(String[] args) {
    LazyValue lazy = new LazyValue() {
        public Object get() {
            return complexComputation(args);
        }
    };
    foo(lazy);
}
```

LANGUAGES

- Java version using generic interface

```
interface LazyValue <T> {
    T get();
}

void foo(LazyValue<String[]> value) {
    // ...
    if (condition) {
        Object evaluated = value.get();
        doSomethingWith(evaluated);
    }
    // ...
}

void main(String[] args) {
    LazyValue<String[]> lazy =
            new LazyValue<String[]>() {
        public String[] get() {
            return complexComputation(args);
        }
    };
    foo(lazy);
}
```

LANGUAGES

- Java version using generic abstract class

```
abstract class LazyValue <T> {
    abstract T get();
}

void foo(LazyValue<String[]> value) {
    // ...
    if (condition) {
        Object evaluated = value.get();
        doSomethingWith(evaluated);
    }
    // ...
}

void main(String[] args) {
    LazyValue<String[]> lazy =
            new LazyValue<String[]>() {
        @Override
        String[] get() {
            return complexComputation(args);
        }
    };
    foo(lazy);
}
```

LANGUAGES

- Java version using generic interface and lambda expression

```
interface LazyValue <T> {
    T get();
}

void foo(LazyValue<String[]> value) {
    // ...
    if (condition) {
        Object evaluated = value.get();
        doSomethingWith(evaluated);
    }
    // ...
}

void main(String[] args) {
    foo(() -> complexComputation(args));
}
```

LANGUAGES

- Java version using predefined interface and lambda expression

```
void foo(Supplier<String[]> value) {
    // ...
    if (condition) {
        Object evaluated = value.get();
        doSomethingWith(evaluated);
    }
    // ...
}

void main(String[] args) {
    foo(() -> complexComputation(args));
}
```

LANGUAGES

- JavaScript version using function

```
function foo(value) {
    // ...
    if (condition) {
        var evaluated = value();
        doSomethingWith(evaluated);
    }
    // ...
}

function main(args) {
    function lazy() {
        return complexCalculation(args)
    }
    foo(lazy);
}
```

LANGUAGES

- JavaScript version using anonymous function

```
function foo(value) {
    // ...
    if (condition) {
        var evaluated = value();
        doSomethingWith(evaluated);
    }
    // ...
}

function main(args) {
    foo(function() {
        return complexCalculation(args)
    });
}
```

LANGUAGES

- JavaScript version using ECMAScript 6 lambdas

```
function foo(value) {
    // ...
    if (condition) {
        var evaluated = value();
        doSomethingWith(evaluated);
    }
    // ...
}

function main(args) {
    foo(() => complexCalculation(args));
}
```

LANGUAGES

- R version using function

```
foo <- function(value) {
    # ...
    if(condition) {
        evaluated <- value()
        doSomethingWith(evaluated)
    }
    # ...
}

main <- function(args) {
    lazy <- function() {
        complexComputation(args)
    }
    foo(lazy)
}
```

LANGUAGES

- R version using lazy argument evaluation

```
foo <- function(value) {
    # ...
    if(condition) {
        doSomethingWith(value)
    }
    # ...
}

main <- function(args) {
    foo(complexComputation(args))
}
```

LANGUAGES

- Some are not as well suited for APIs, e.g. not typing

```
function foo(value) {
```

- Comments:

```
/* void */ function foo(/* string[] function */ value) {
```

- TypeScript, Flow, …:

```
function foo(value: () => string): void {
```

- asm.js (int parameter):

```
function foo(value) {
    value = value|0;
    // ...
```

LANGUAGES

- Function overloading

```
void foo(A a) {
    // ...
}
void foo(B b) {
    // ...
}
```

- Different names:

```
function foo_A(value) {
    // ...
}
function foo_A(value) {
    // ...
}
```

- Dynamic checks:

```
function foo_A(value) {
    if (value instanceof A) {
        // ...
    } else {
        // ...
    }
}
```

GRAAL

Modularity and API Design

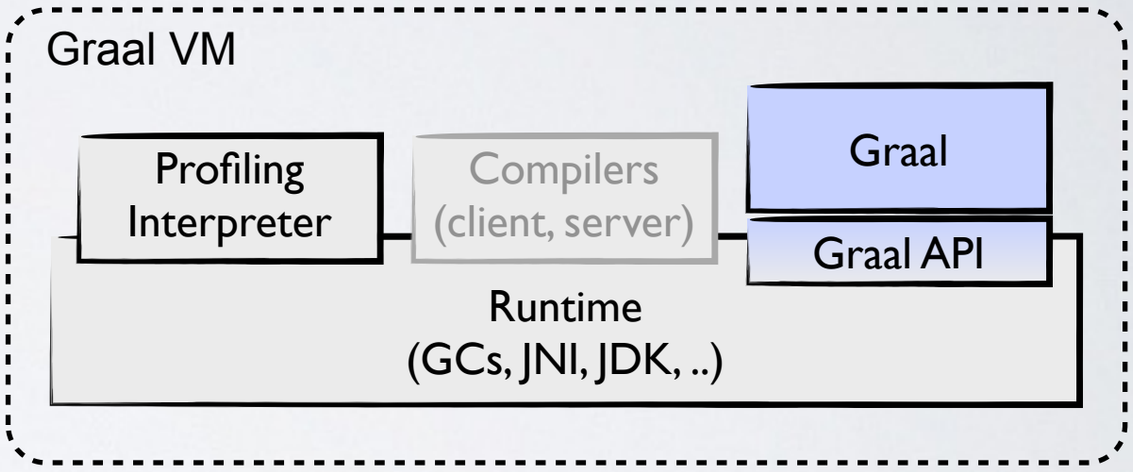
2015/01 Tulach, Würthinger, Stadler

GRAAL COMPILER

- OpenJDK project:
  http://openjdk.java.net/projects/graal/

- Easy to build and use:

```
hg clone http://hg.openjdk.java.net/graal/graal
mx build
mx vm ...
```



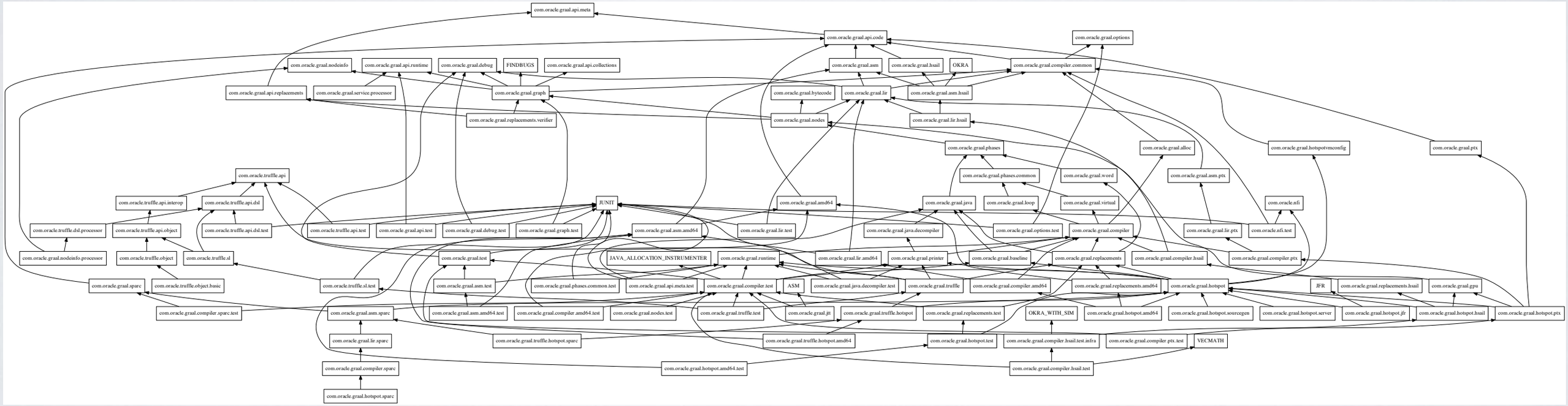
- Java JIT - compiler written in Java

- Productivity (powerful IDEs, easy refactoring, debugging, approachability, ...)

- Compiler (mostly) decoupled from VM

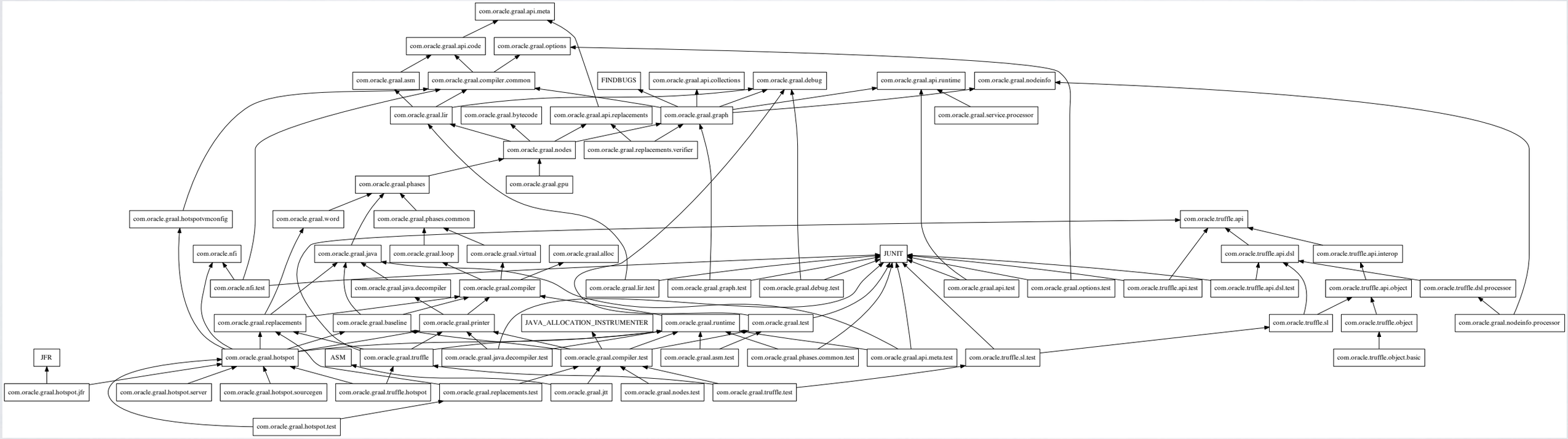
GRAAL

- 100+ modules, 200+ module dependencies, 20+ contributors
- Project graph: script + “graphviz dot tool”



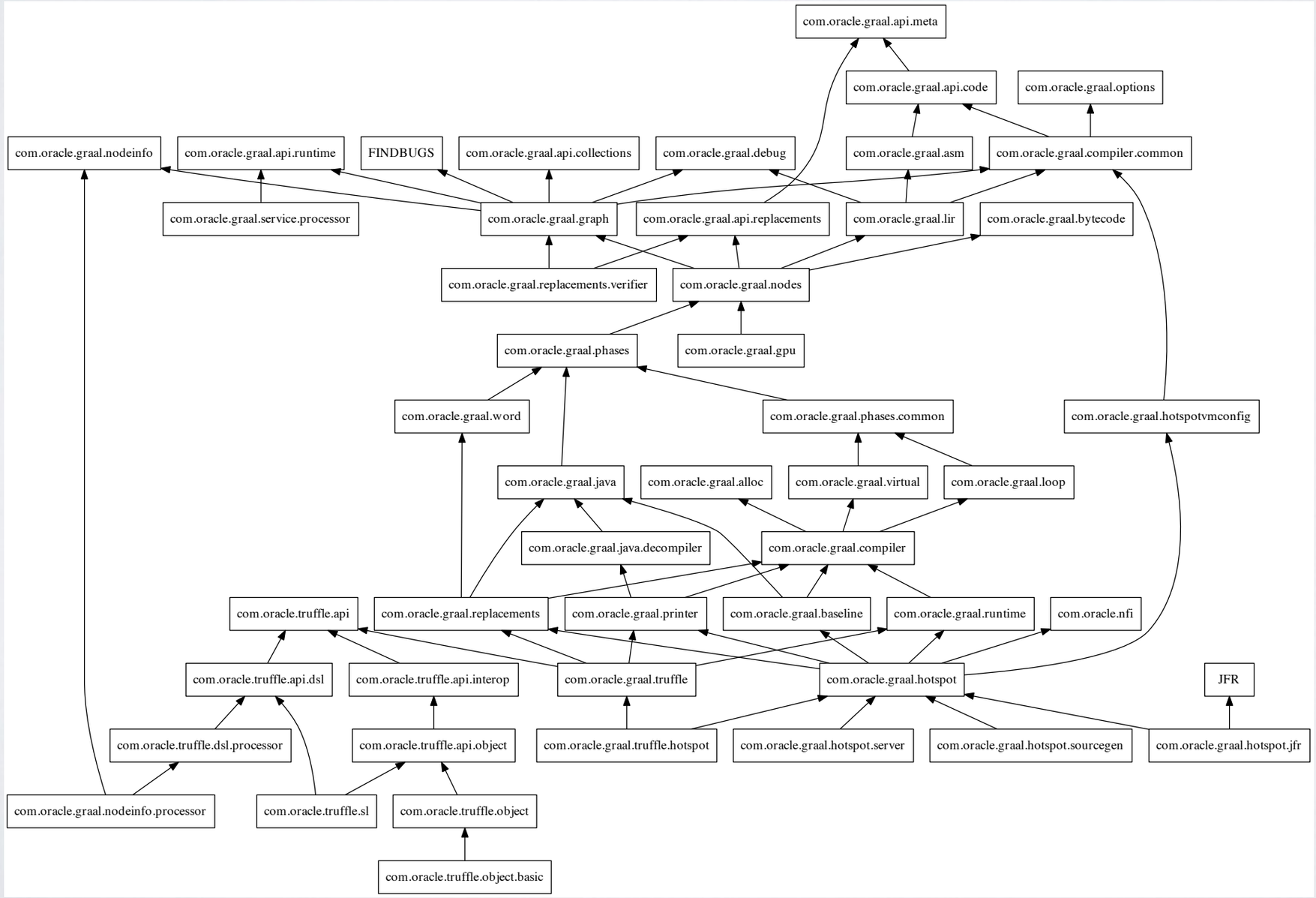
GRAAL

- Without CPU-specific code



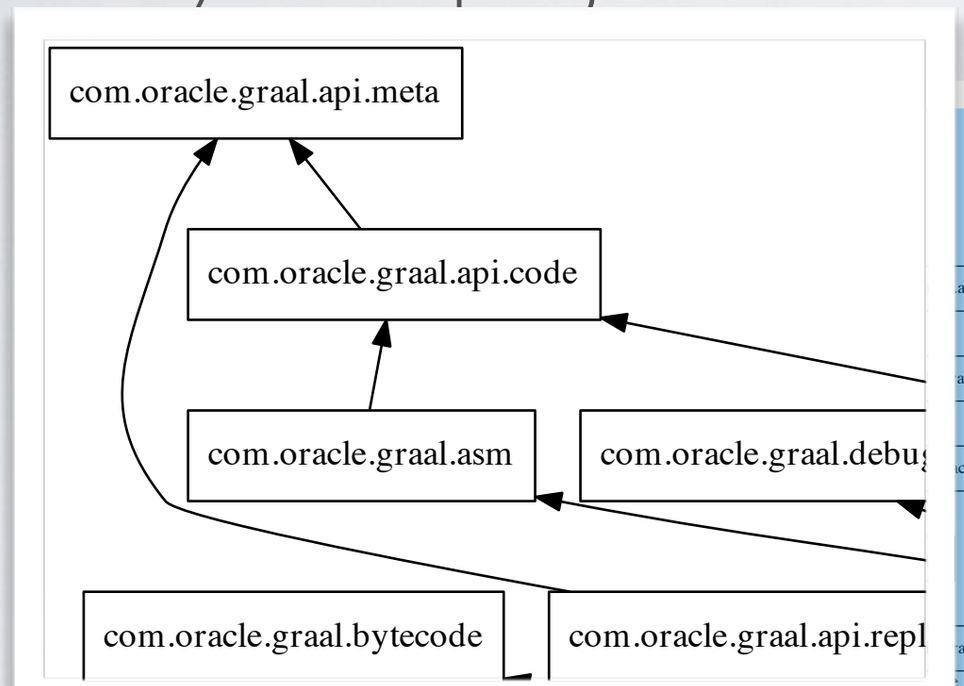
GRAAL

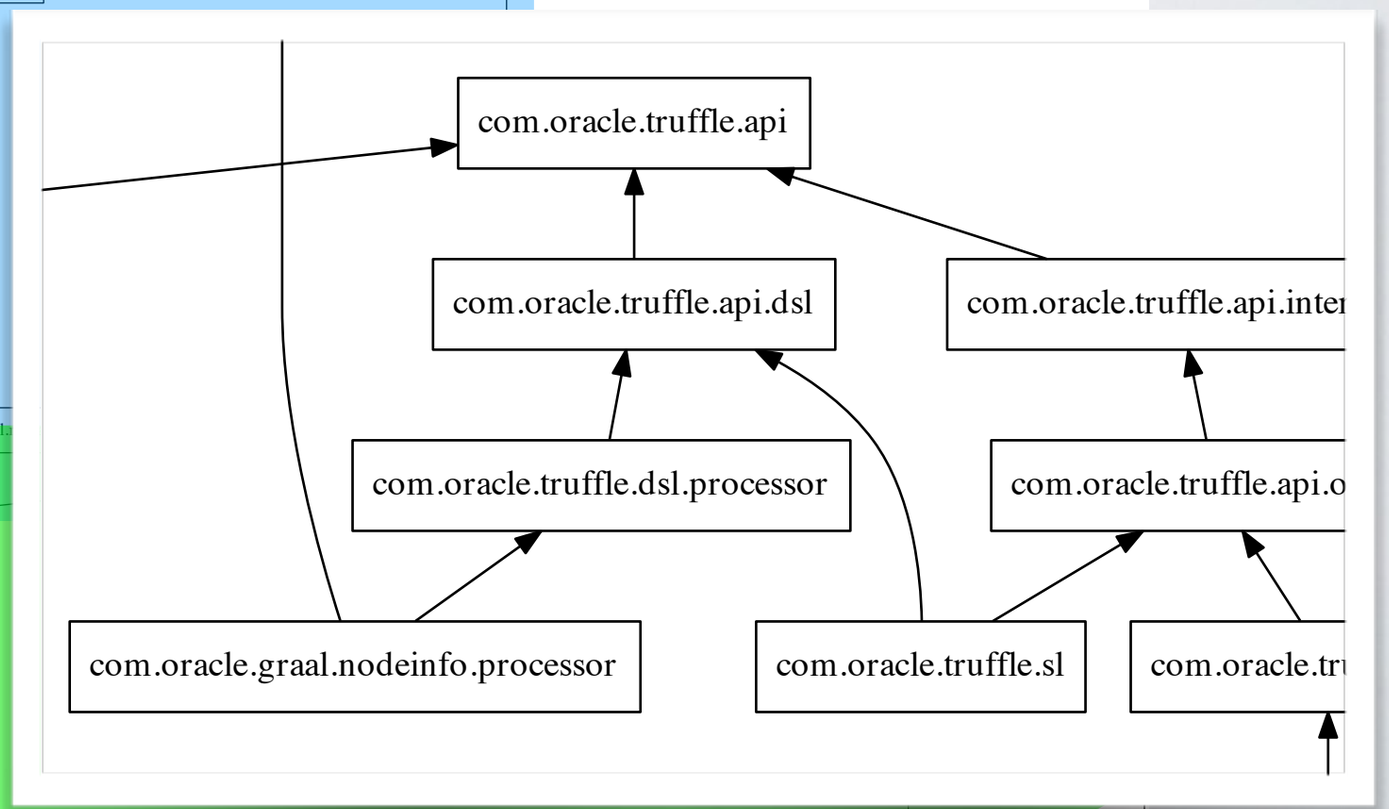
- Without CPU-specific code, without test code



GRAAL API

- Only core projects…





GRAAL API / TOOLS

- Findbugs: findbugs.sourceforge.net
- Checkstyle: checkstyle.sourceforge.net
- JDepend www.clarkware.com/software/JDepend.html
- Used in Continuous Integration

```
/**
 * Represents a resolved Java method. Methods, like fields and types, are resolved through
 * {@link ConstantPool constant pools}.
 */
public interface ResolvedJavaMethod extends JavaMethod, InvokeTarget, ModifiersProvider {

    /**
     * Returns the bytecode of this method, if the method has code. The returned byte array does not
     * contain breakpoints or non-Java bytecodes. This may return null if the
     * {@link #getDeclaringClass() holder} is not {@link ResolvedJavaType#isLinked() linked}.
     *
     * The contained constant pool indices may not be the ones found in the original class file but
     * they can be used with the Graal API (e.g. methods in {@link ConstantPool}).
     *
     * @return the bytecode of the method, or {@code null} if {@code getCodeSize() == 0} or if the
     *         code is not ready.
     */
    byte[] getCode();

    /**
     * Returns the size of the bytecode of this method, if the method has code. This is equivalent
     * to {@link #getCode()}. {@code length} if the method has code.
     *
     * @return the size of the bytecode in bytes, or 0 if no bytecode is available
     */
    int getCodeSize();

    /**
     * Returns the {@link ResolvedJavaType} object representing the class or interface that declares
     * this method.
     */
    ResolvedJavaType getDeclaringClass();

    /**
     * Returns the maximum number of locals used in this method's bytecodes.
     */
    int getMaxLocals();
```

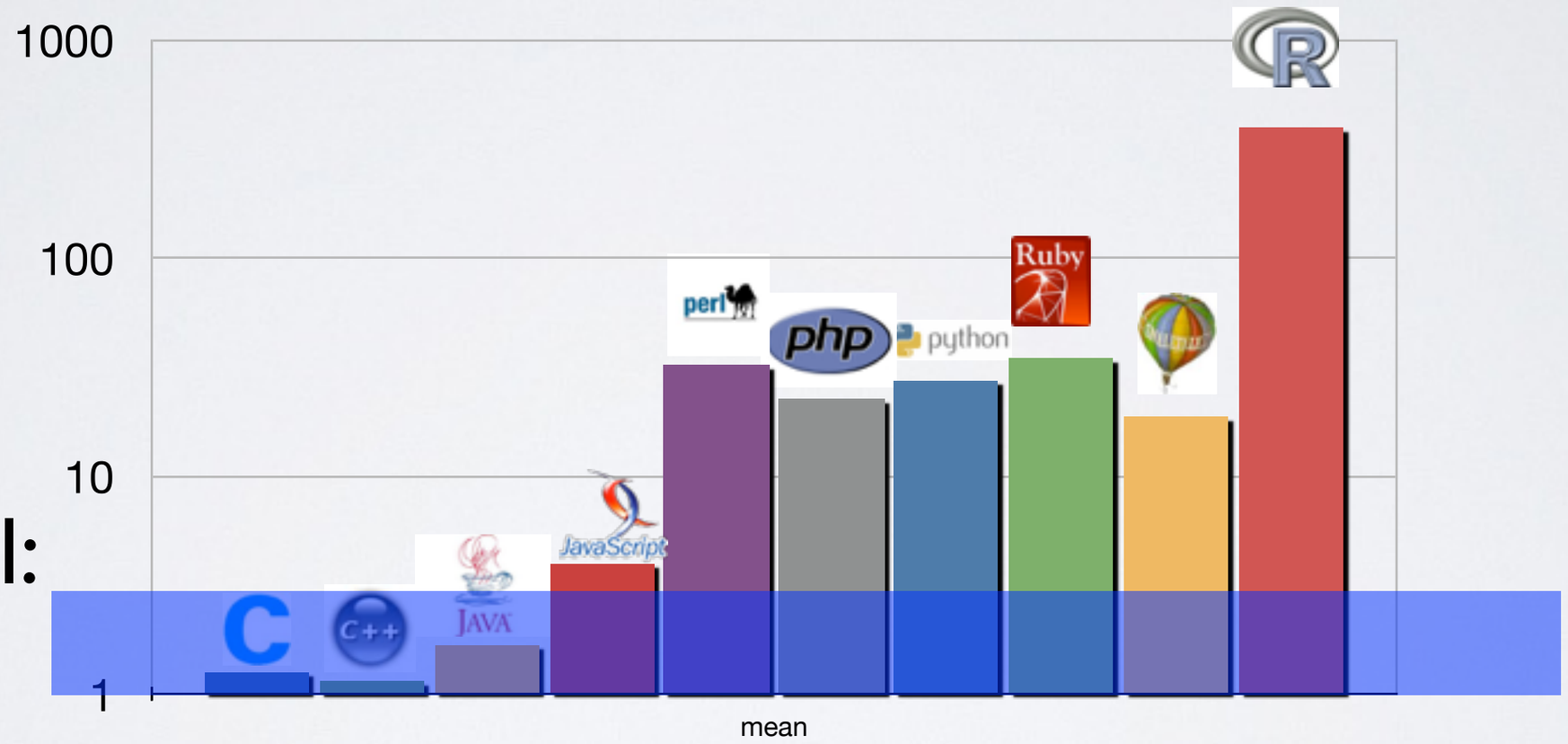
TRUFFLE

Modularity and API Design

2015/01 Tulach, Würthinger, Stadler

TRUFFLE FRAMEWORK

Goal:



TRUFFLE FRAMEWORK

Current situation

Prototype a new language

Parser and language work to build syntax tree (AST), AST Interpreter

Write a “real” VM

In C/C++, still using AST interpreter, spend a lot of time implementing runtime system, GC, …

People start using it

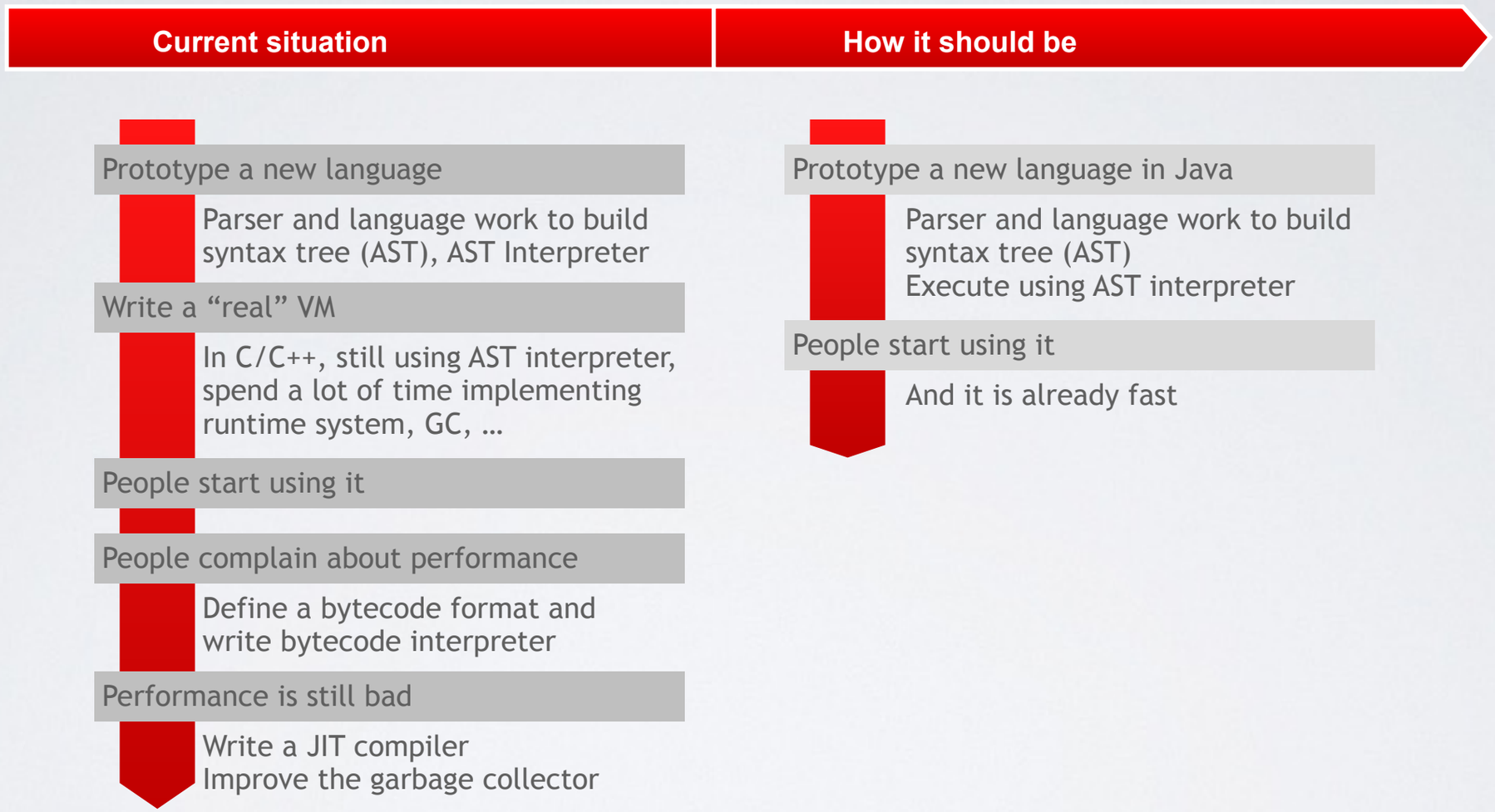
People complain about performance

Define a bytecode format and write bytecode interpreter

Performance is still bad

Write a JIT compiler
Improve the garbage collector

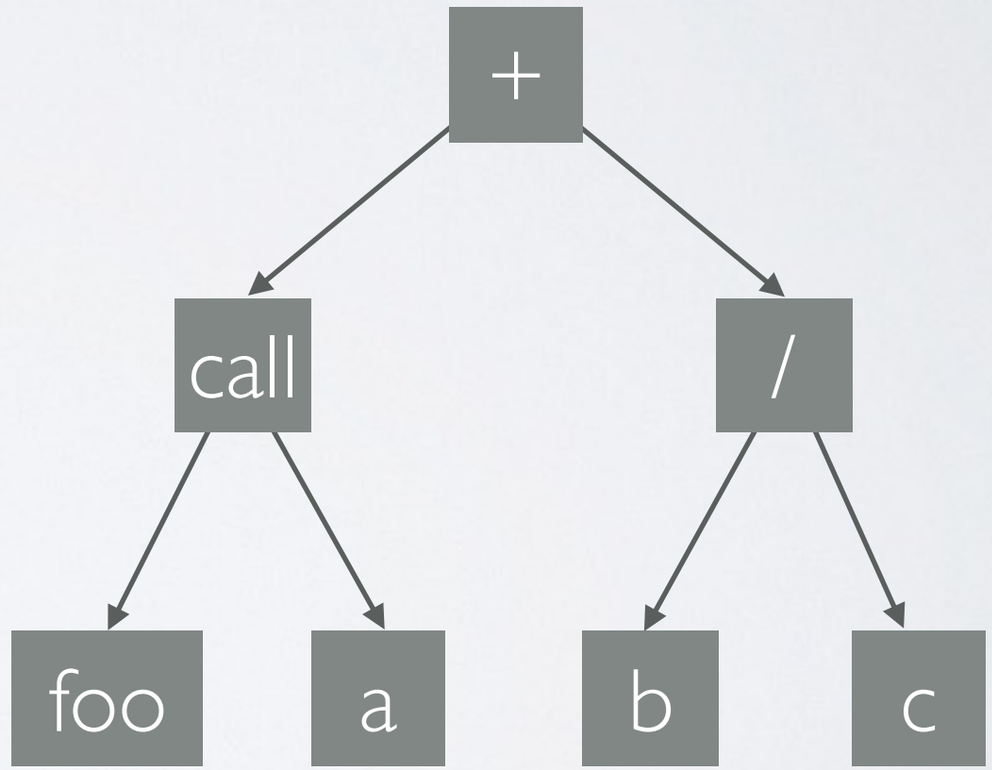
TRUFFLE FRAMEWORK



TRUFFLE FRAMEWORK

- AST interpreter in Java
- Multiple APIs building on each other:
  - Truffle
  - Truffle DSL

```
foo(a)+ b / c
```



TRUFFLE FRAMEWORK

- Part of Graal:
- OpenJDK project:
  http://openjdk.java.net/projects/graal/
- Easy to build and use:

```
hg clone http://hg.openjdk.java.net/graal/graal
mx build
mx vm ...  AST interpreter in Java
```

- “Simple Language” (sl) as an example

TRUFFLE FRAMEWORK

- AST interpreter in Java
- Multiple APIs building on each other:
  - Truffle
  - Truffle DSL

```
@NodeInfo(shortName = "+")
public abstract class AddNode extends BinaryNode {

    public AddNode(SourceSection src) {
        super(src);
    }

    @Specialization(rewriteOn = ArithmeticException.class)
    protected long add(long left, long right) {
        return ExactMath.addExact(left, right);
    }

    @Specialization
    @TruffleBoundary
    protected BigInteger add(BigInteger left, BigInteger right) {
        return left.add(right);
    }

    @Specialization(guards = "isString")
    @TruffleBoundary
    protected String add(Object left, Object right) {
        return left.toString() + right.toString();
    }

    protected boolean isString(Object a, Object b) {
        return a instanceof String || b instanceof String;
    }
}
```

TRUFFLE FRAMEWORK

- AST interpreter in Java
- Multiple APIs building on each other:
  - Truffle
  - Truffle DSL

```
@NodeInfo(shortName = "+")
public abstract class AddNode extends BinaryNode {

    public AddNode(SourceSection src) {
        super(src);
    }

    @Specializa                                              n.class)
    protected lo
        return l
    }

    @Specializa
    @TruffleBoundary
    protected BigInteger add(BigInteger left, BigInteger right) {
        return left.add(right);
    }

    @Speci
    @Truff
    protec
        re
    }

    protec
        re                                                   ng;
    }
}
```

Interpreter gathers runtime feedback

Creates compiled code based on the interpreter (2nd futumura projection, partial evaluation)

TRUFFLE LANGUAGES

- Many languages and language prototypes
  (prototypes - none of these is guaranteed to become a product!)

Python

JavaScript

R

J

Truffle

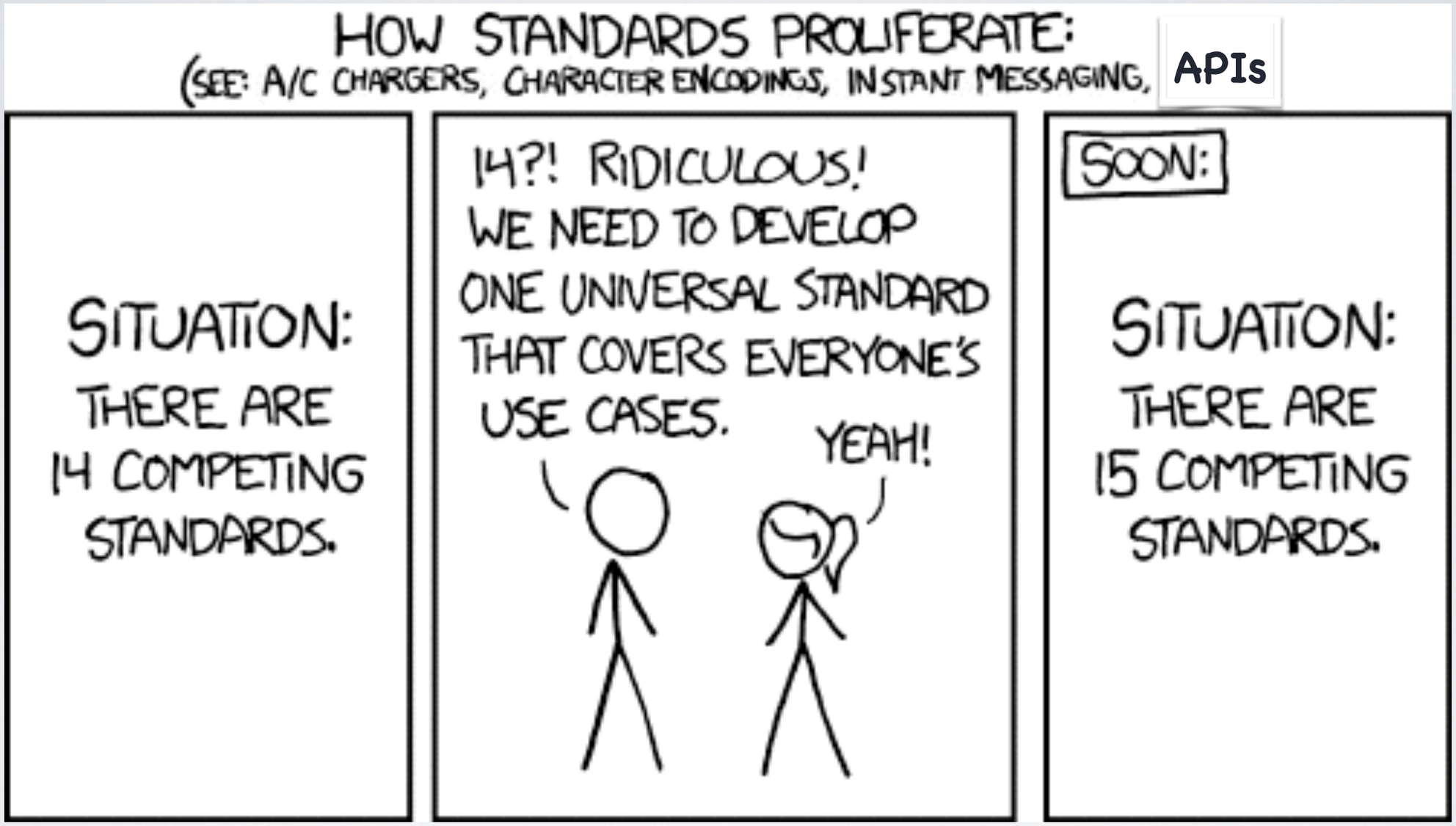
Ruby

```
quicksort=: ((<#[), (=#[), (>#[)) ({~ ?@#)) ^: (1<#)
```

Smalltalk

SL

..?



http://xkcd.com/927/

COURSE PROJECT

- Create a new software artifact, focusing on modularity and extensibility:
  Netbeans plugin, Eclipse plugin, Truffle language interpreter, DukeScript application, JavaScript library wrapper, …
- … or modularize an existing artifact
- Large enough to show modularization aspects and define interfaces (more than just a few classes)

COURSE PROJECT

- Send to thomas.wuerthinger@oracle.com
  - Sourcecode + short description
    Deadline: March 13
- Present the project in person, explaining modularization
  - No fixed date yet, likely March 16-20
- In teams of at least two students!