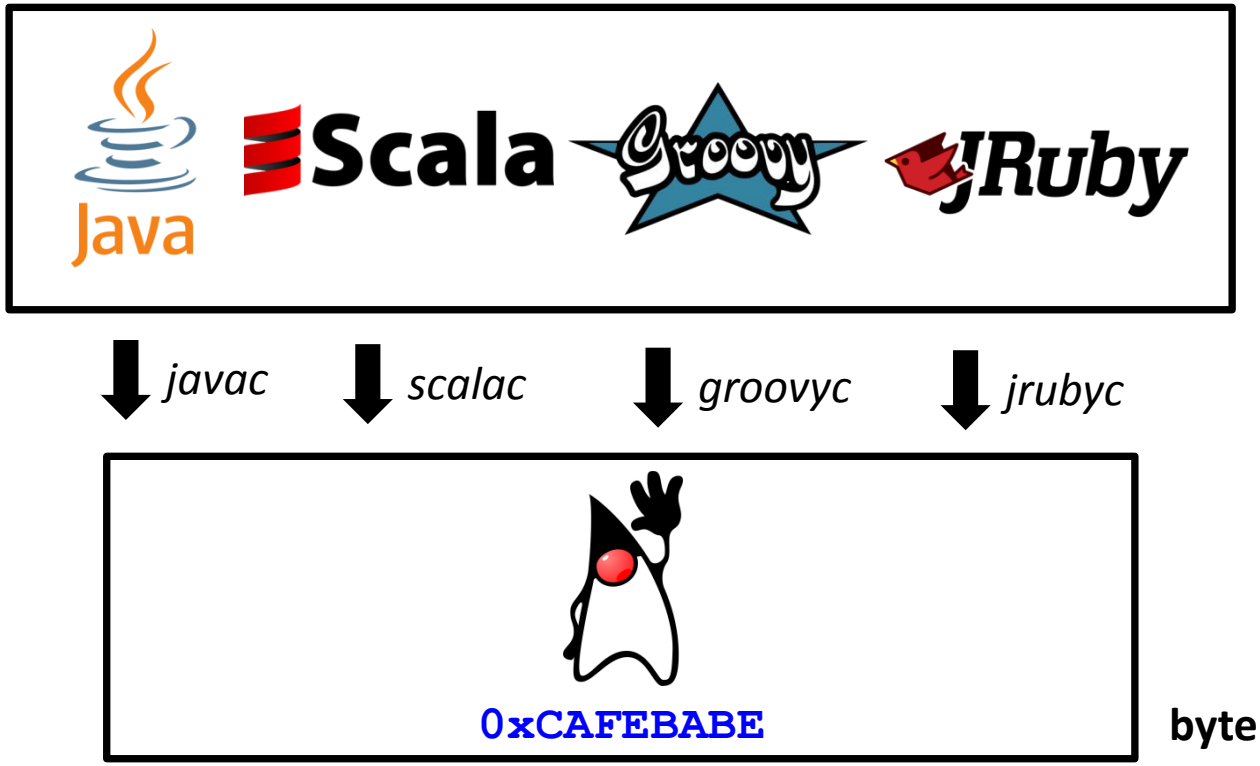
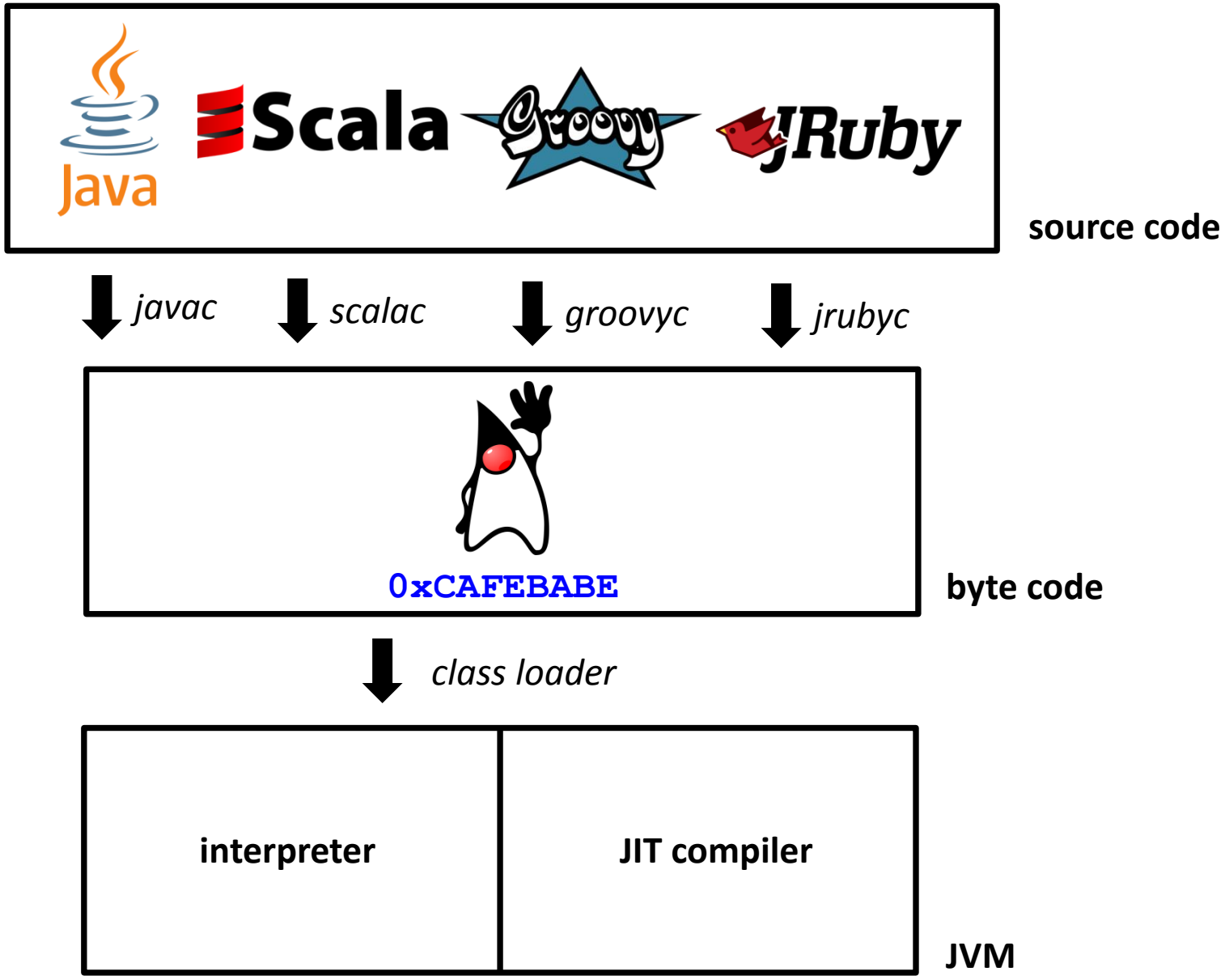
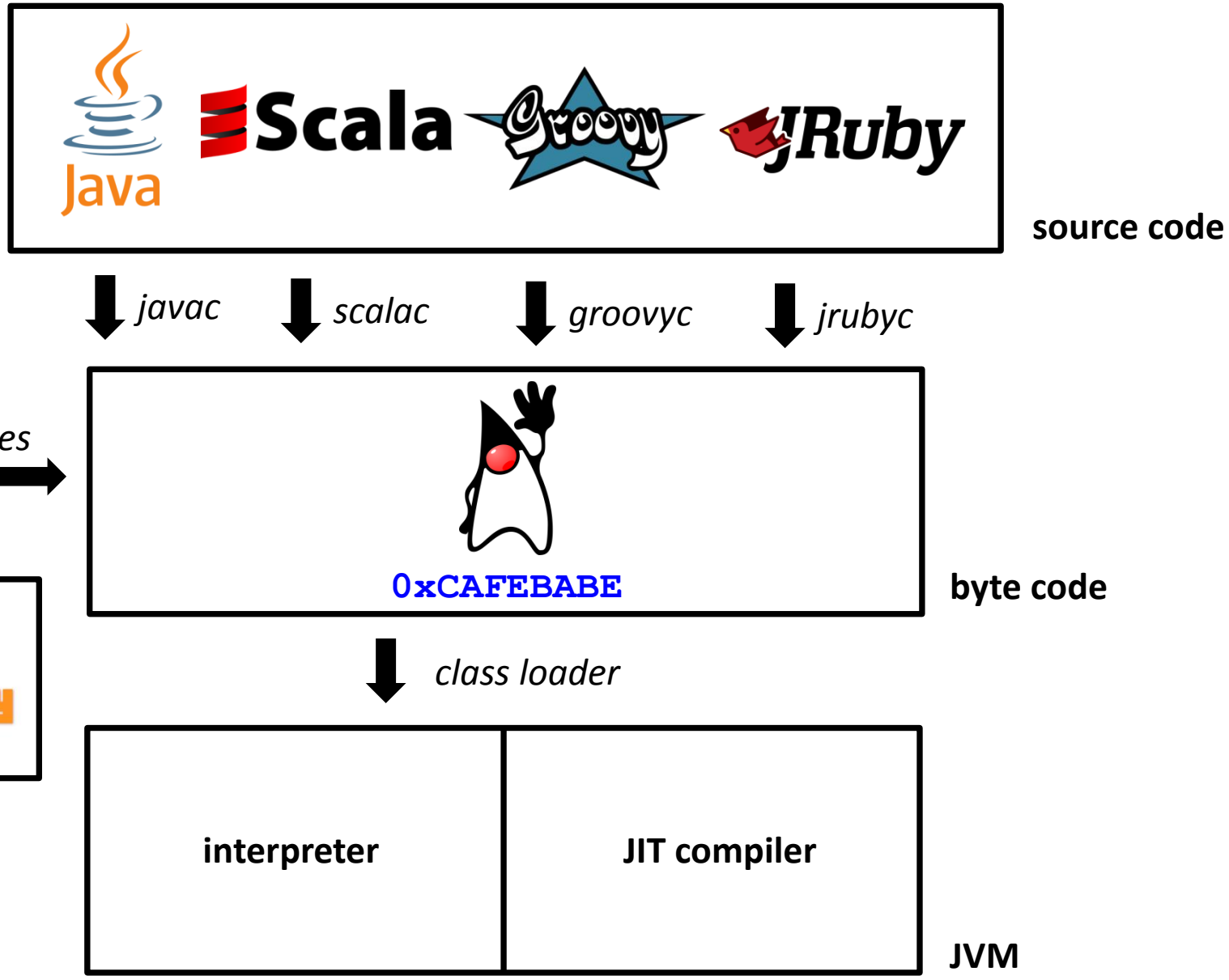
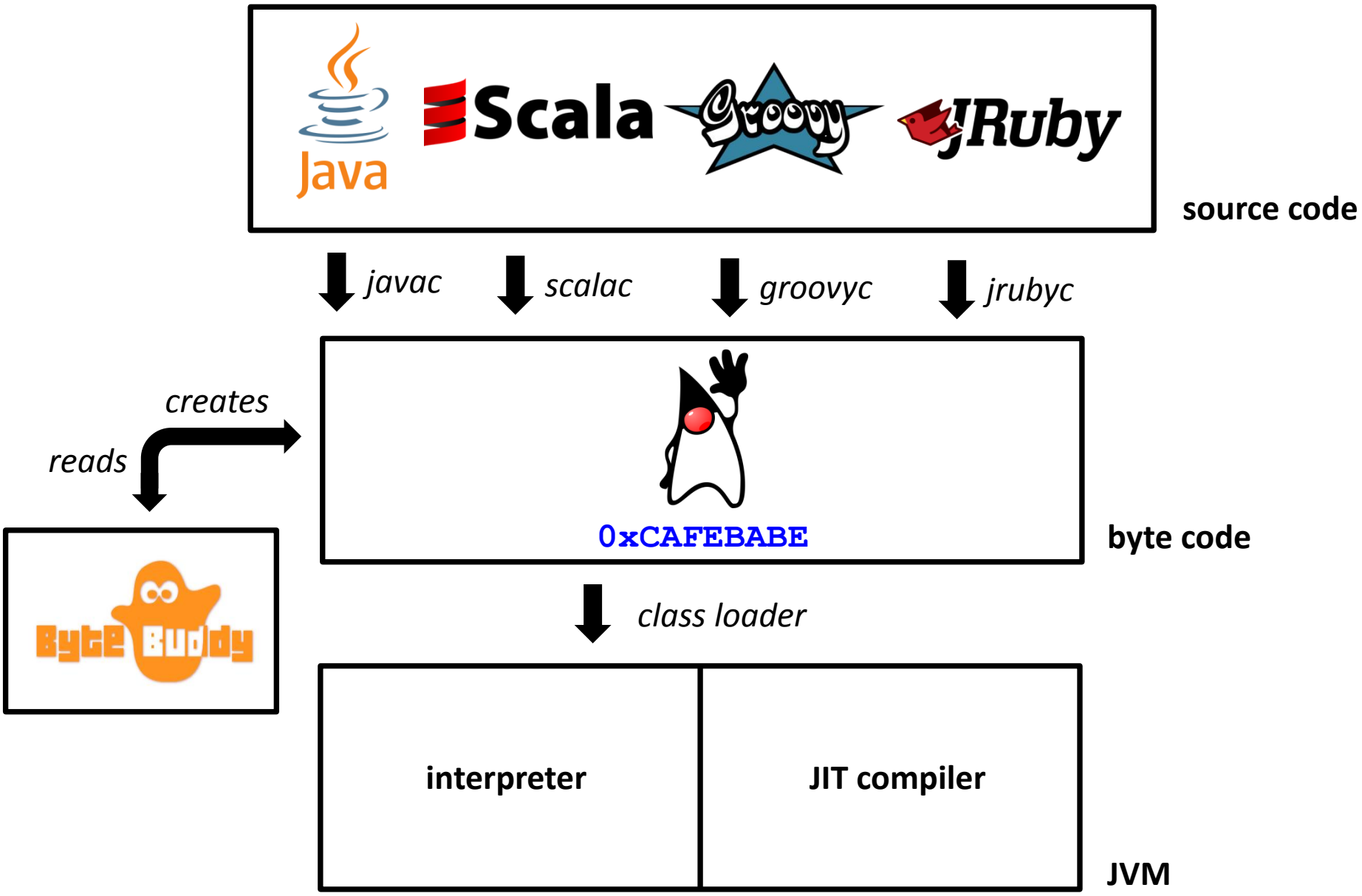
Java byte code in practice

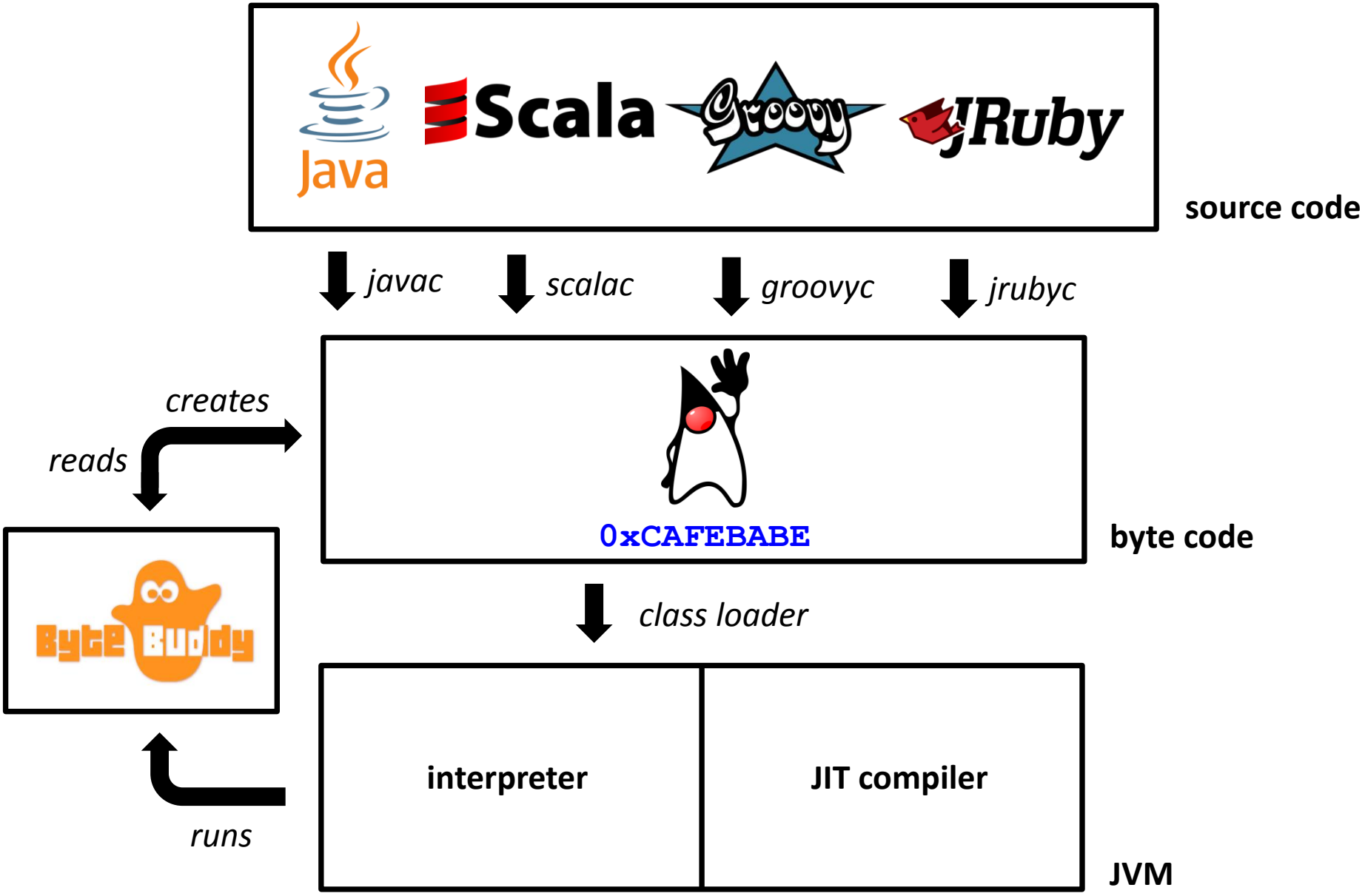
source code


Java
Scala
Groovy
JRuby
source code
javac
scalac
groovyc
jrubyc
0xCAFEBABE
byte code


Java
Scala
Groovy
JRuby
source code
javac
scalac
groovyc
jrubyc
0xCAFEBABE
byte code
class loader
interpreter
JIT compiler
JVM


Java
Scala
Groovy
JRuby
source code
javac
scalac
groovyc
jrubyc
creates
0xCAFEBABE
byte code
Byte Buddy
class loader
interpreter
JIT compiler
JVM


Java
Scala
Groovy
JRuby
source code
javac
scalac
groovyc
jrubyc
creates
reads
0xCAFEBABE
byte code
Byte Buddy
class loader
interpreter
JIT compiler
JVM


Java
Scala
Groovy
JRuby
source code
javac
scalac
groovyc
jrubyc
creates
reads
0xCAFEBABE
byte code
Byte Buddy
class loader
interpreter
JIT compiler
JVM
runs

source code

```
void foo() {

}
```

byte code

source code

```
void foo() {

}
```

byte code

```
RETURN
```

source code

```
void foo() {
  return;
}
```

byte code

```
RETURN
```

source code

```
void foo() {
  return;
}
```

byte code

```
0xB1
```

source code

```
int foo() {
  return 1 + 2;
}
```

byte code

source code

```
int foo() {
  return 1 + 2;
}
```

byte code

```
ICONST_1
ICONST_2
IADD
```

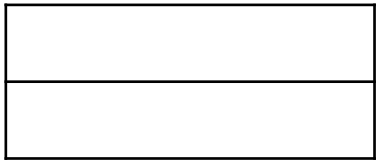
source code

```
int foo() {
  return 1 + 2;
}
```

byte code

```
ICONST_1
ICONST_2
IADD
```

operand stack



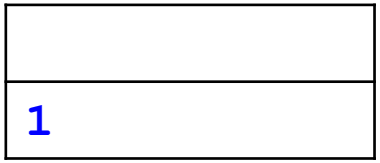
source code

```
int foo() {
  return 1 + 2;
}
```

byte code

```
ICONST_1
ICONST_2
IADD
```

operand stack



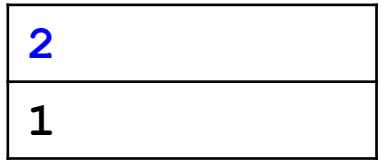
source code

```
int foo() {
  return 1 + 2;
}
```

byte code

```
ICONST_1
ICONST_2
IADD
```

operand stack



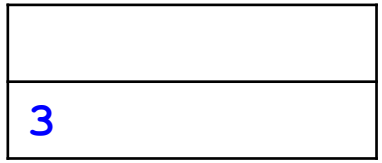
source code

```
int foo() {
  return 1 + 2;
}
```

byte code

```
ICONST_1
ICONST_2
IADD
```

operand stack



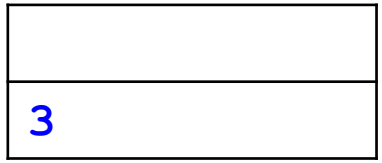
source code

```
int foo() {
  return 1 + 2;
}
```

byte code

```
ICONST_1
ICONST_2
IADD
IRETURN
```

operand stack



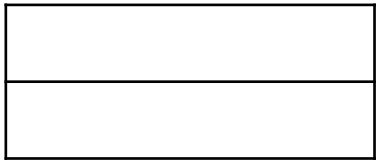
source code

```
int foo() {
  return 1 + 2;
}
```

byte code

```
ICONST_1
ICONST_2
IADD
IRETURN
```

operand stack



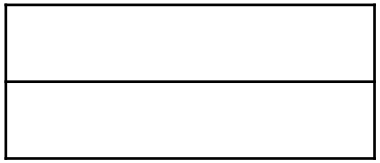
source code

```
int foo() {
  return 1 + 2;
}
```

byte code

```
0x04
0x05
0x60
0xAC
```

operand stack

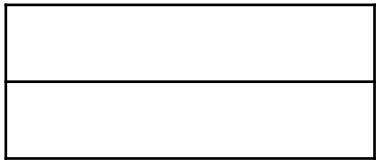


source code

```
int foo() {
  return 11 + 2;
}
```

byte code

operand stack



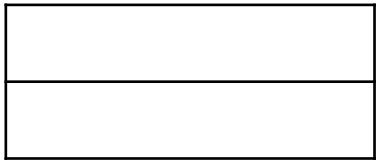
source code

```
int foo() {
  return 11 + 2;
}
```

byte code

```
BIPUSH 11
```

operand stack



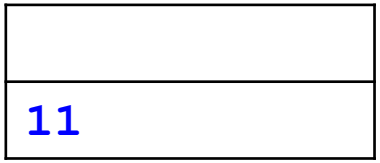
source code

```
int foo() {
  return 11 + 2;
}
```

byte code

```
BIPUSH 11
```

operand stack



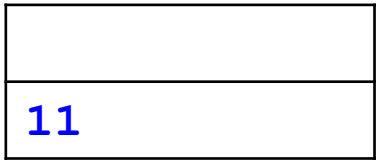
source code

```
int foo() {
  return 11 + 2;
}
```

byte code

```
BIPUSH 11
ICONST_2
IADD
IRETURN
```

operand stack



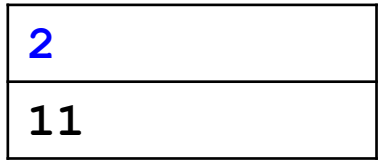
source code

```
int foo() {
  return 11 + 2;
}
```

byte code

```
BIPUSH 11
ICONST_2
IADD
IRETURN
```

operand stack



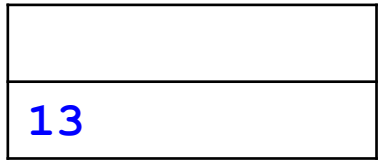
source code

```
int foo() {
  return 11 + 2;
}
```

byte code

```
BIPUSH 11
ICONST_2
IADD
IRETURN
```

operand stack



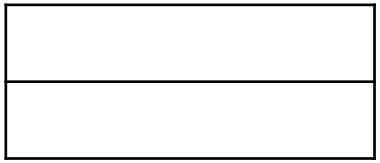
source code

```
int foo() {
  return 11 + 2;
}
```

byte code

```
BIPUSH 11
ICONST_2
IADD
IRETURN
```

operand stack



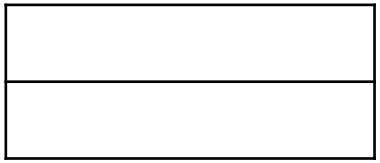
source code

```
int foo() {
  return 11 + 2;
}
```

byte code

```
0x10   11
ICONST_2
IADD
IRETURN
```

operand stack



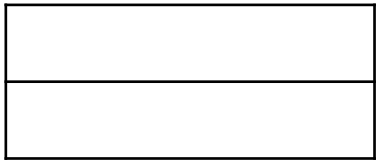
source code

```
int foo() {
  return 11 + 2;
}
```

byte code

```
0x10   0x0B
ICONST_2
IADD
IRETURN
```

operand stack



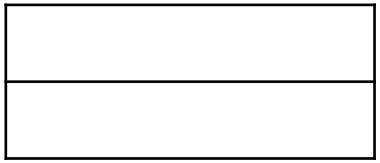
source code

```
int foo() {
  return 11 + 2;
}
```

byte code

```
0x10  0x0B
0x05
0x60
0xAC
```

operand stack

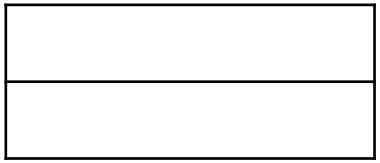


source code

```
int foo(int i) {
  return i + 1;
}
```

byte code

operand stack



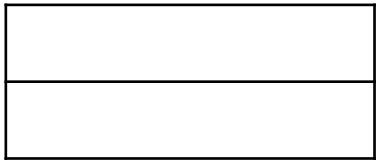
source code

```
int foo(int i) {
  return i + 1;
}
```

byte code

```
ILOAD_1
```

operand stack



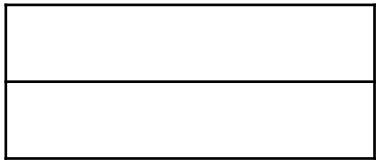
source code

```
int foo(int i) {
  return i + 1;
}
```

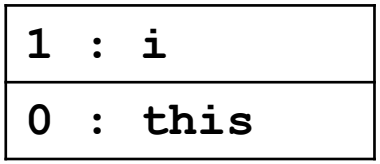
byte code

```
ILOAD_1
```

operand stack



local variable array



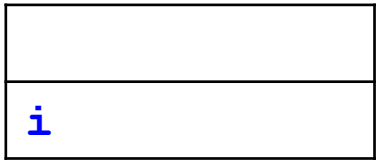
source code

```
int foo(int i) {
  return i + 1;
}
```

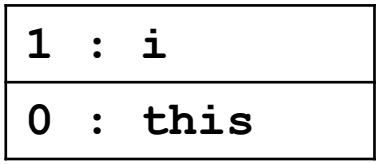
byte code

```
ILOAD_1
```

operand stack



local variable array



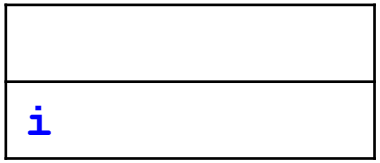
source code

```
int foo(int i) {
  return i + 1;
}
```

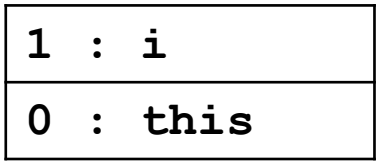
byte code

```
ILOAD_1
ICONST_1
IADD
IRETURN
```

operand stack



local variable array



source code

```
int foo(int i) {
  return i + 1;
}
```

byte code

```
ILOAD_1
ICONST_1
IADD
IRETURN
```

operand stack

1
i

local variable array

1 : i
0 : this

source code

```
int foo(int i) {
  return i + 1;
}
```

byte code

```
ILOAD_1
ICONST_1
IADD
IRETURN
```

operand stack

i + 1

local variable array

1 : i
0 : this

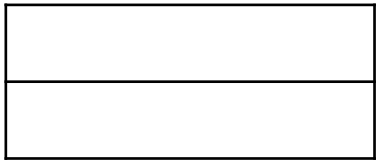
source code

```
int foo(int i) {
  return i + 1;
}
```

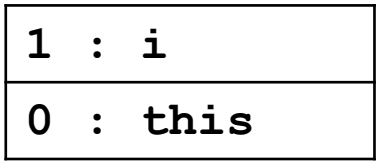
byte code

```
ILOAD_1
ICONST_1
IADD
IRETURN
```

operand stack



local variable array



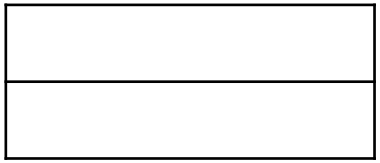
source code

```
int foo(int i) {
  return i + 1;
}
```

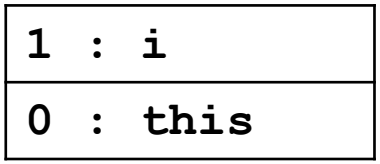
byte code

```
0x1B
0x04
0x60
0xAC
```

operand stack



local variable array



source code

```
long foo(long i) {
  return i + 1L;
}
```

byte code

source code

```
long foo(long i) {
  return i + 1L;
}
```

byte code

```
LLOAD_1
LCONST_1
LADD
LRETURN
```

source code

```
long foo(long i) {
  return i + 1L;
}
```

byte code

```
LLOAD_1
LCONST_1
LADD
LRETURN
```

local variable array

2 : i (cn.)
1 : i
0 : this

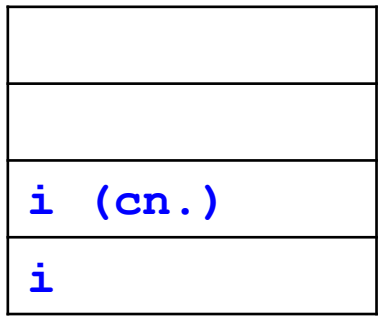
source code

```
long foo(long i) {
  return i + 1L;
}
```

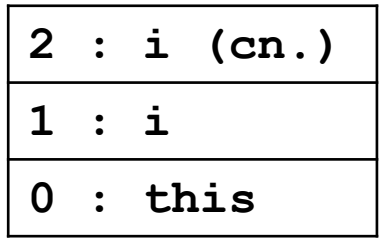
byte code

```
LLOAD_1
LCONST_1
LADD
LRETURN
```

operand stack



local variable array



source code

```
long foo(long i) {
  return i + 1L;
}
```

byte code

```
LLOAD_1
LCONST_1
LADD
LRETURN
```

operand stack

1L (cn.)
1L
i (cn.)
i

local variable array

2 : i (cn.)
1 : i
0 : this

source code

```
long foo(long i) {
  return i + 1L;
}
```

byte code

```
LLOAD_1
LCONST_1
LADD
LRETURN
```

operand stack

i + 1L (cn.)
i + 1L

local variable array

2 : i (cn.)
1 : i
0 : this

source code

```
long foo(long i) {
  return i + 1L;
}
```

byte code

```
LLOAD_1
LCONST_1
LADD
LRETURN
```

operand stack

local variable array

2 : i (cn.)
1 : i
0 : this

source code

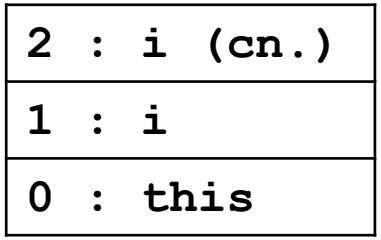
```
long foo(long i) {
  return i + 1L;
}
```

byte code

```
0x1F
0x0A
0x61
0xAD
```

operand stack

local variable array

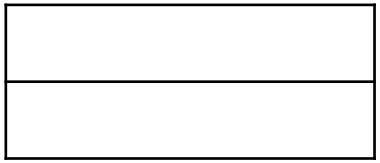


source code

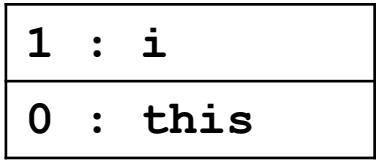
```
short foo(short i) {
  return (short) (i + 1);
}
```

byte code

operand stack



local variable array



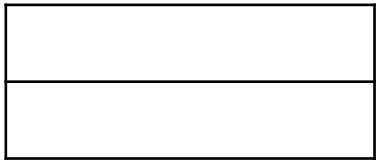
source code

```
short foo(short i) {
  return (short) (i + 1);
}
```

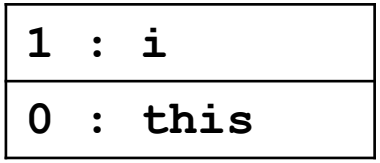
byte code

```
ILOAD_1
ICONST_1
IADD
```

operand stack



local variable array



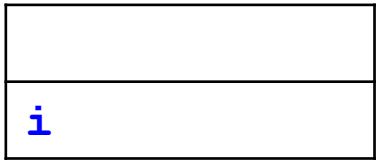
source code

```
short foo(short i) {
  return (short) (i + 1);
}
```

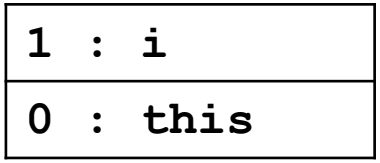
byte code

```
ILOAD_1
ICONST_1
IADD
```

operand stack



local variable array



source code

```
short foo(short i) {
  return (short) (i + 1);
}
```

byte code

```
ILOAD_1
ICONST_1
IADD
```

operand stack

1
i

local variable array

1 : i
0 : this

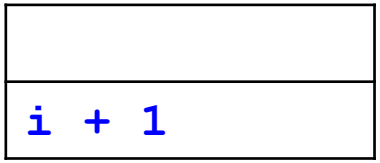
source code

```
short foo(short i) {
  return (short) (i + 1);
}
```

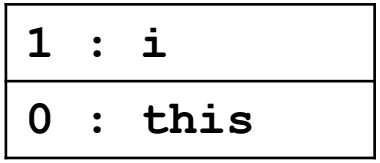
byte code

```
ILOAD_1
ICONST_1
IADD
```

operand stack



local variable array



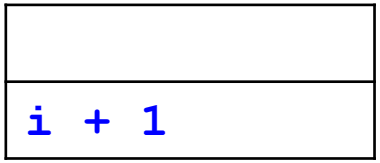
source code

```
short foo(short i) {
  return (short) (i + 1);
}
```

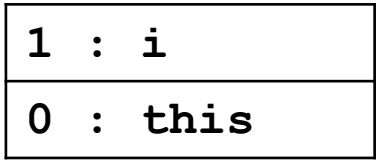
byte code

```
ILOAD_1
ICONST_1
IADD
I2S
```

operand stack



local variable array



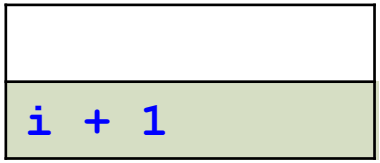
source code

```
short foo(short i) {
  return (short) (i + 1);
}
```

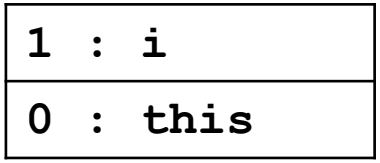
byte code

```
ILOAD_1
ICONST_1
IADD
I2S
```

operand stack



local variable array



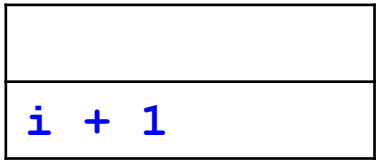
source code

```
short foo(short i) {
  return (short) (i + 1);
}
```

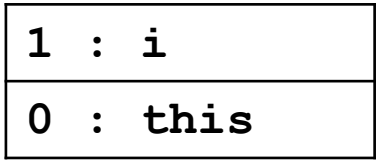
byte code

```
ILOAD_1
ICONST_1
IADD
I2S
IRETURN
```

operand stack



local variable array



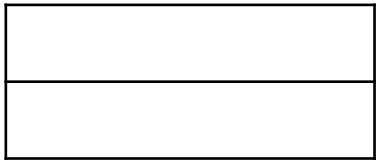
source code

```
short foo(short i) {
  return (short) (i + 1);
}
```

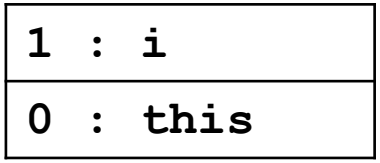
byte code

```
ILOAD_1
ICONST_1
IADD
I2S
IRETURN
```

operand stack



local variable array



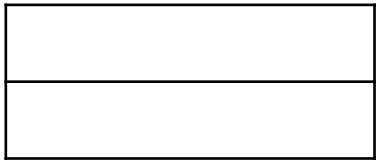
source code

```
short foo(short i) {
  return (short) (i + 1);
}
```

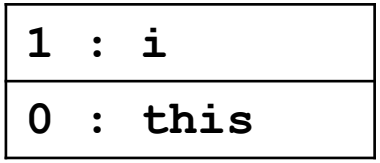
byte code

```
0x1B
0x04
0x60
0x93
0xAC
```

operand stack



local variable array



source code

```
void foo() {
  return;
}
```

byte code

```
RETURN
```

source code

```
void foo() {
  return;
}
```

byte code

```
RETURN
```

source code

```
void foo() {
  return;
}
```

byte code

```
RETURN
```

source code

```
void foo() {
  return;
}
```

byte code

```
0x0000  foo ()V
RETURN
```

source code

```
void foo() {
  return;
}
```

byte code

```
0x0000  foo ()V
RETURN
0  1
```

operand stack ////////////////

local variable array `0 : this`

source code

```
void foo() {
  return;
}
```

byte code

```
0x0000  foo ()V
RETURN
0  1
```

operand stack ////////////////

local variable array `0 : this`

source code

```
package pkg;
class Bar {
  void foo() {
    return;
  }
}
```

byte code

pkg/Bar.class

```
0x0000  foo ()V
RETURN
0  1
```

operand stack ////////////////

local variable array

```
0 : this
```

source code

```
package pkg;
class Bar {
  void foo() {
    return;
  }
}
```

byte code

pkg/Bar.class

```
0x0000  foo ()V
RETURN
0  1
```

constant pool (i.a. UTF-8)

operand stack ////////////////

local variable array `0 : this`

source code

```
package pkg;
class Bar {
  void foo() {
    return;
  }
}
```

byte code

pkg/Bar.class

```
0x0000  foo  ()V
RETURN
0  1
```

0x0000: foo
0x0001: ()V
0x0002: pkg/Bar

constant pool (i.a. UTF-8)

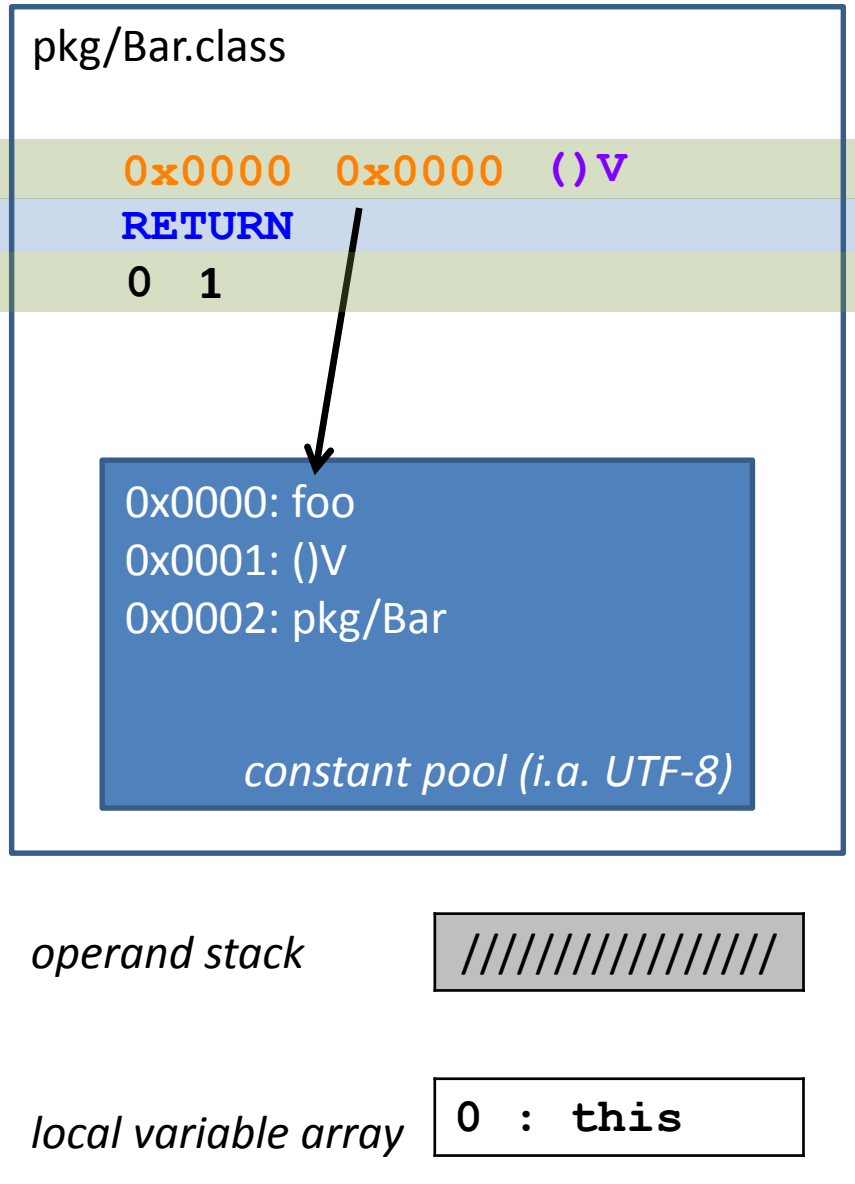
operand stack ////////////////

local variable array `0 : this`

source code

```
package pkg;
class Bar {
  void foo() {
    return;
  }
}
```

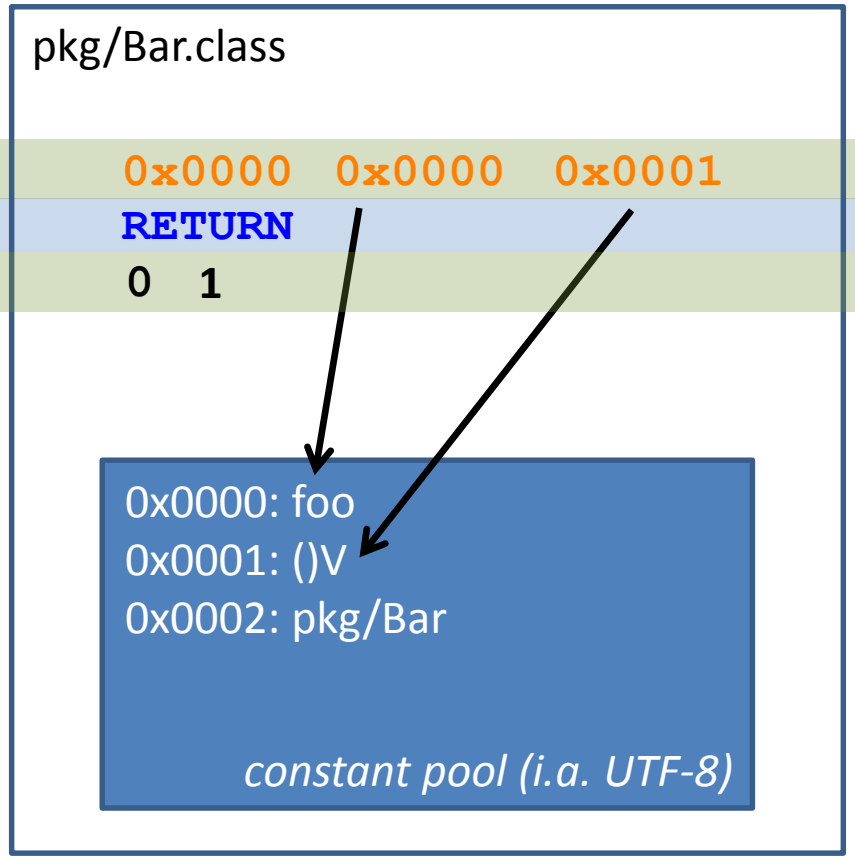
byte code



source code

```
package pkg;
class Bar {
  void foo() {
    return;
  }
}
```

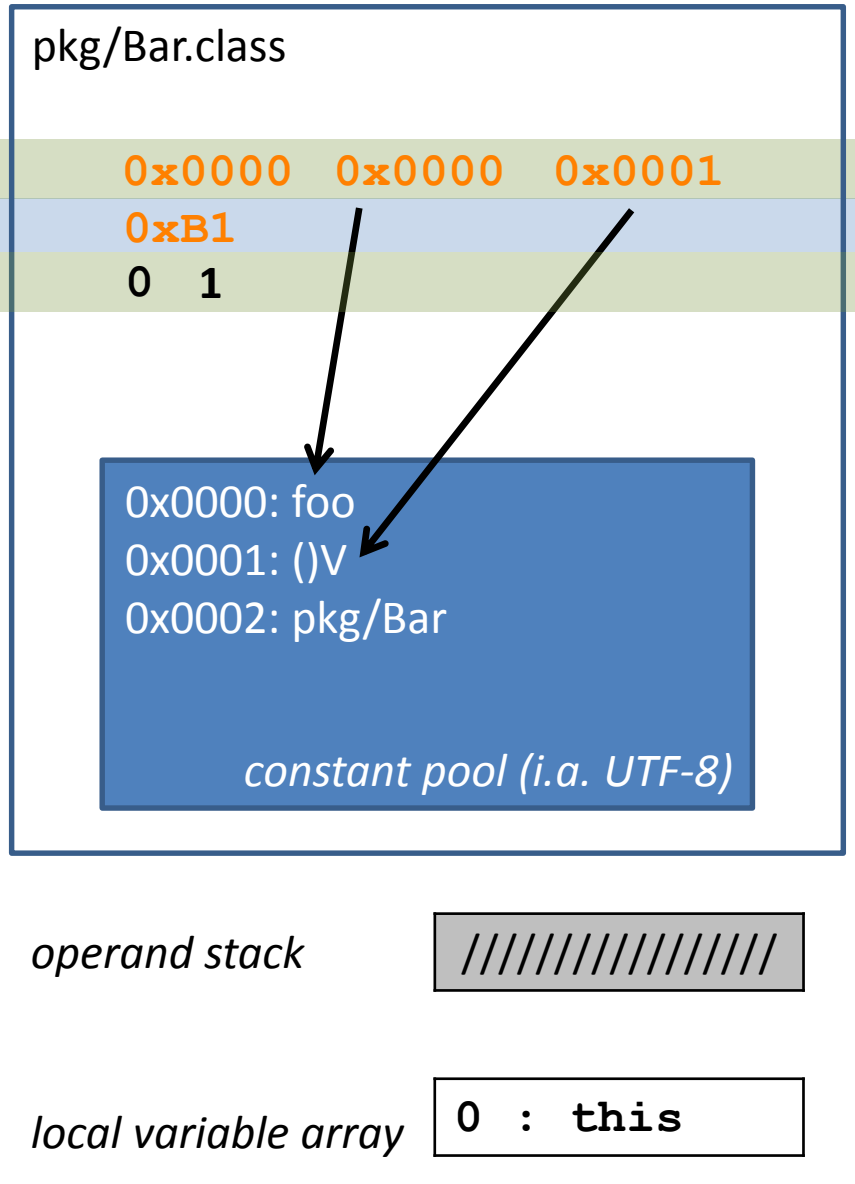
byte code



source code

```
package pkg;
class Bar {
  void foo() {
    return;
  }
}
```

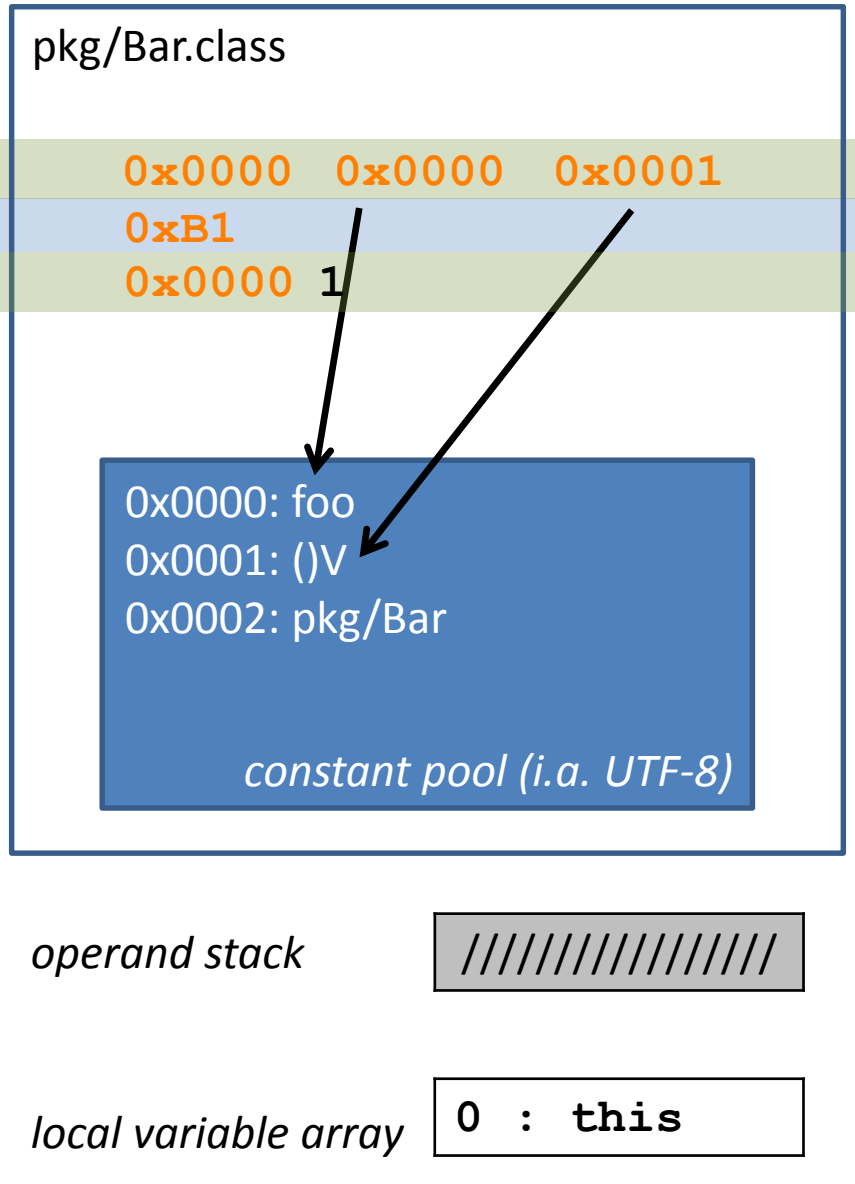
byte code



source code

```
package pkg;
class Bar {
  void foo() {
    return;
  }
}
```

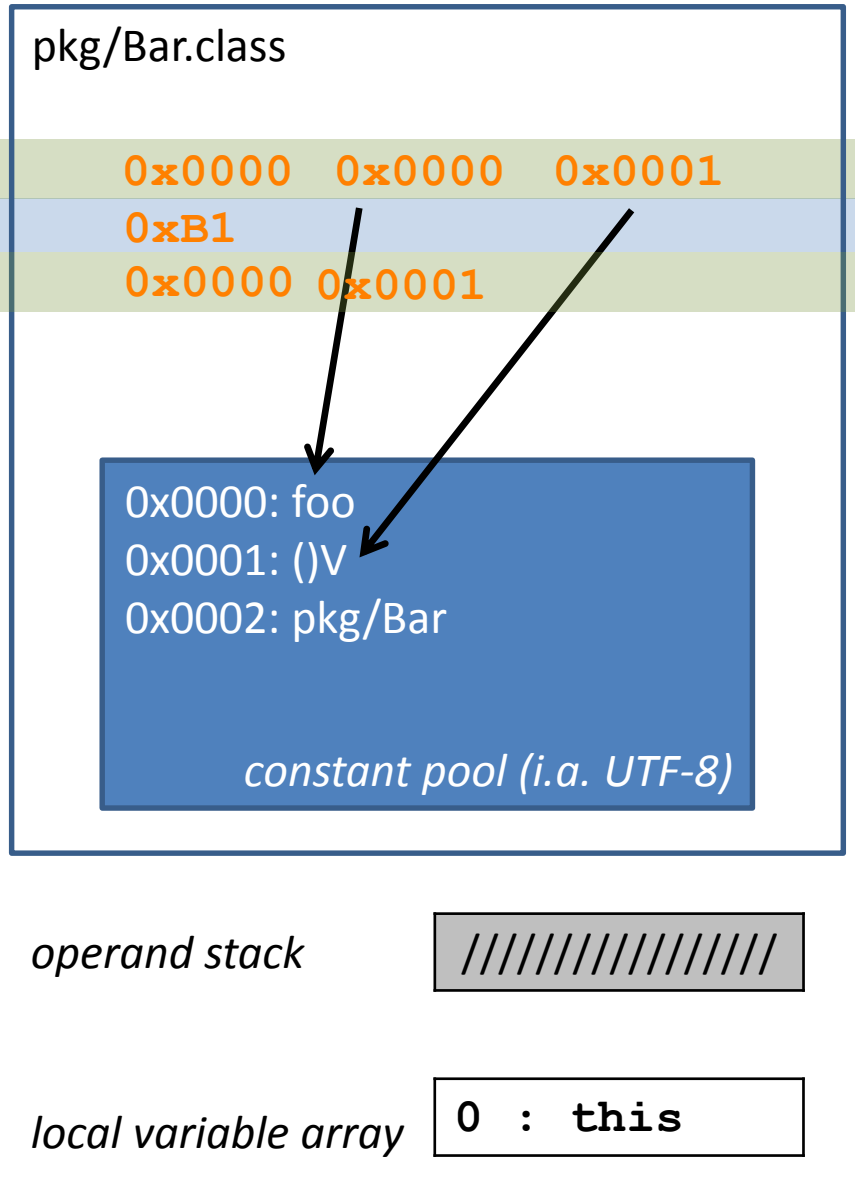
byte code



source code

```
package pkg;
class Bar {
  void foo() {
    return;
  }
}
```

byte code



Java type	JVM type (non-array)	JVM descriptor	stack slots
boolean	I	Z	1
byte		B	1
short		S	1
char		C	1
int		I	1
long	L	J	2
float	F	F	1
double	D	D	2
void	-	V	0
java.lang.Object	A	Ljava/lang/Object;	1

source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
ILOAD_1
ICONST_1
IADD
```

source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
ILOAD_1
ICONST_1
IADD
INVOKEVIRTUAL
```

source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
ILOAD_1
ICONST_1
IADD
INVOKEVIRTUAL  pkg/Bar foo (I)I
```

source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
ILOAD_1
ICONST_1
IADD
INVOKEVIRTUAL  pkg/Bar foo (I)I
```

local variable array

1 : i
0 : this

source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
ILOAD_1
ICONST_1
IADD
INVOKEVIRTUAL  pkg/Bar foo (I)I
```

local variable array

1 : i
0 : this

source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
ALOAD_0
ILOAD_1
ICONST_1
IADD
INVOKEVIRTUAL  pkg/Bar foo (I)I
```

local variable array

1 : i
0 : this

source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
ALOAD_0
ILOAD_1
ICONST_1
IADD
INVOKEVIRTUAL  pkg/Bar foo (I)I
IRETURN
```

local variable array

1 : i
0 : this

source code

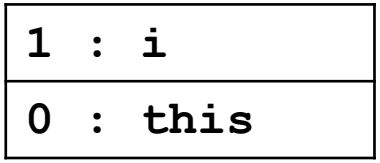
```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
ALOAD_0
ILOAD_1
ICONST_1
IADD
INVOKEVIRTUAL  pkg/Bar foo (I)I
IRETURN
```

operand stack

local variable array

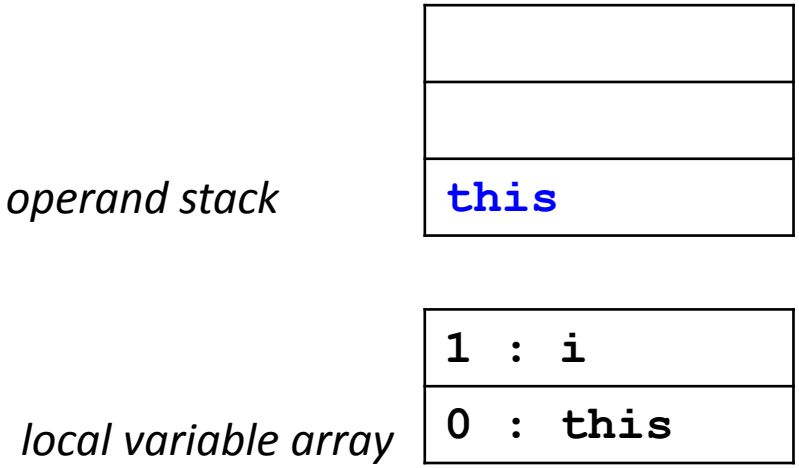


source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
ALOAD_0
ILOAD_1
ICONST_1
IADD
INVOKEVIRTUAL  pkg/Bar foo (I)I
IRETURN
```



source code

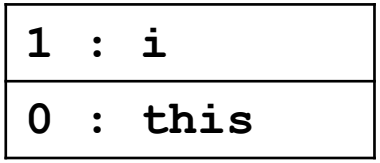
```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
ALOAD_0
ILOAD_1
ICONST_1
IADD
INVOKEVIRTUAL  pkg/Bar foo (I)I
IRETURN
```

operand stack

local variable array



source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
ALOAD_0
ILOAD_1
ICONST_1
IADD
INVOKEVIRTUAL pkg/Bar foo (I)I
IRETURN
```

operand stack

1
i
this

local variable array

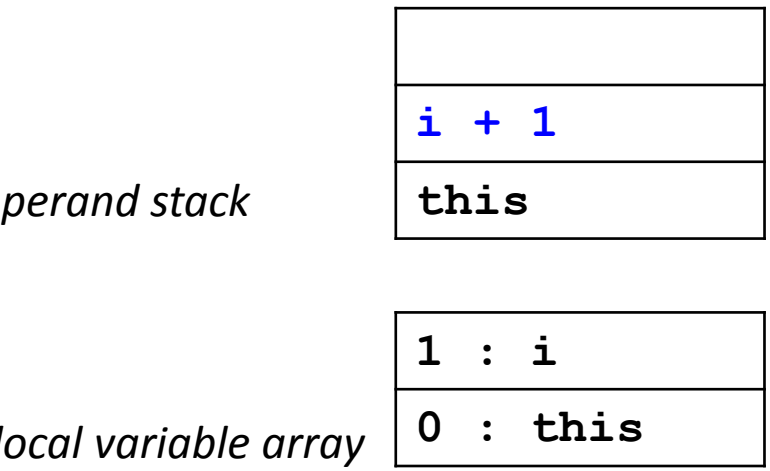
1 : i
0 : this

source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
ALOAD_0
ILOAD_1
ICONST_1
IADD
INVOKEVIRTUAL pkg/Bar foo (I)I
IRETURN
```

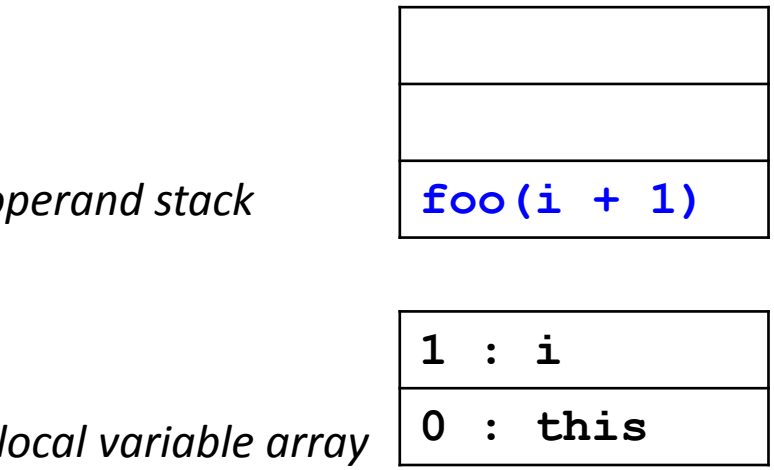


source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
ALOAD_0
ILOAD_1
ICONST_1
IADD
INVOKEVIRTUAL pkg/Bar foo (I)I
IRETURN
```



source code

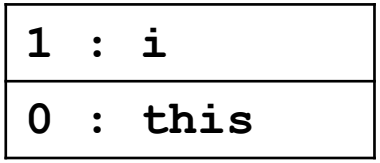
```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
ALOAD_0
ILOAD_1
ICONST_1
IADD
INVOKEVIRTUAL  pkg/Bar foo (I)I
IRETURN
```

operand stack

local variable array

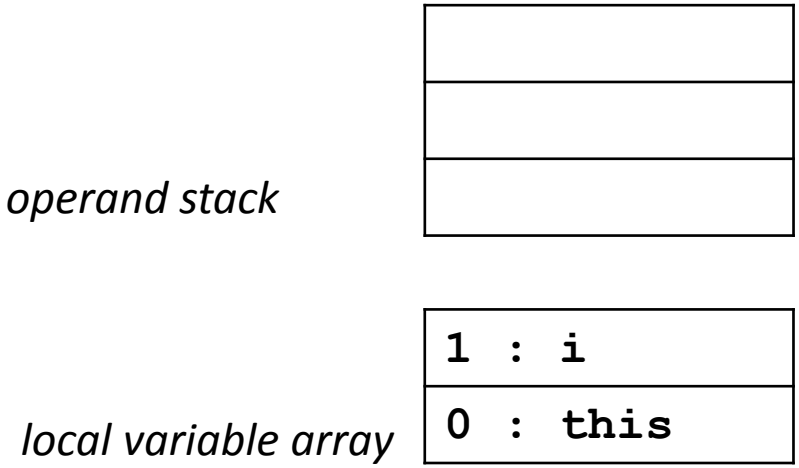


source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
0x2A
0x1B
0x04
0x60
INVOKEVIRTUAL  pkg/Bar foo (I)I
0xAC
```

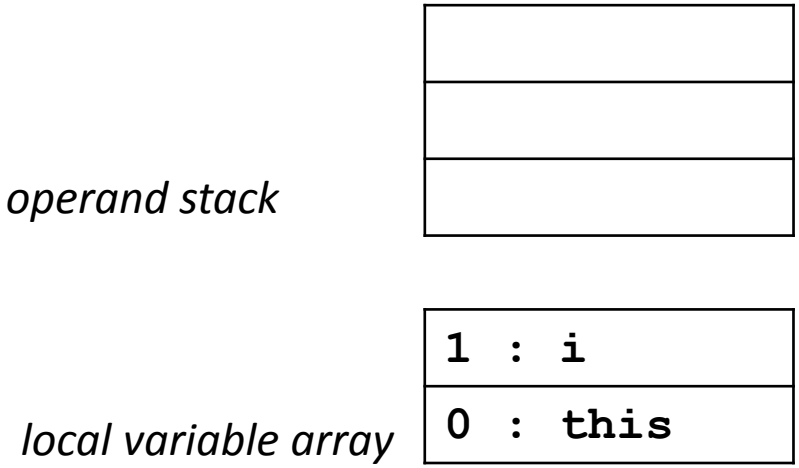


source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

byte code

```
0x2A
0x1B
0x04
0x60
0xB6        pkg/Bar foo (I)I
0xAC
```

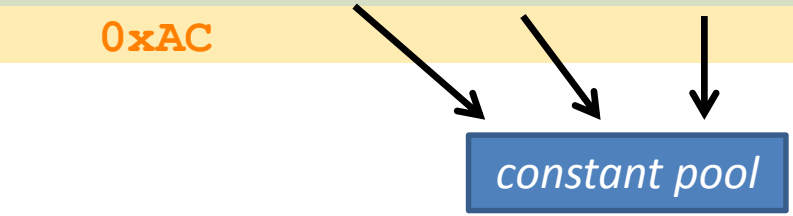


source code

```
package pkg;
class Bar {
  int foo(int i) {
    return foo(i + 1);
  }
}
```

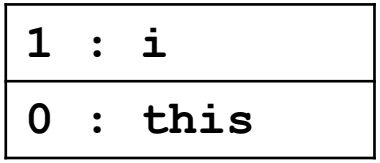
byte code

```
0x2A
0x1B
0x04
0x60
0xB6 0x0002 0x0000 0x0001
0xAC
```



operand stack

local variable array



INVOKESTATIC *pkg/Bar foo ()V*

Invokes a static method.

INVOKESTATIC *pkg/Bar foo ()V*

Invokes a static method.

INVOKEVIRTUAL *pkg/Bar foo ()V*

Invokes the most-specific version of an inherited method on a non-interface class.

INVOKESTATIC *pkg/Bar foo ()V*

Invokes a static method.

INVOKEVIRTUAL *pkg/Bar foo ()V*

Invokes the most-specific version of an inherited method on a non-interface class.

INVOKESPECIAL *pkg/Bar foo ()V*

Invokes a super class‘s version of an inherited method.
Invokes a “constructor method”.
Invokes a private method.
Invokes an interface default method (Java 8).

INVOKESTATIC *pkg/Bar foo ()V*
Invokes a static method.

INVOKEVIRTUAL *pkg/Bar foo ()V*
Invokes the most-specific version of an inherited method on a non-interface class.

INVOKESPECIAL *pkg/Bar foo ()V*
Invokes a super class‘s version of an inherited method.
Invokes a “constructor method”.
Invokes a private method.
Invokes an interface default method (Java 8).

INVOKEINTERFACE *pkg/Bar foo ()V*
Invokes an interface method.
(Similar to INVOKEVIRTUAL but without virtual method table index optimization.)

INVOKESTATIC *pkg/Bar foo ()V*
Invokes a static method.

INVOKEVIRTUAL *pkg/Bar foo ()V*
Invokes the most-specific version of an inherited method on a non-interface class.

INVOKESPECIAL *pkg/Bar foo ()V*
Invokes a super class's version of an inherited method.
Invokes a "constructor method".
Invokes a private method.
Invokes an interface default method (Java 8).

INVOKEINTERFACE *pkg/Bar foo ()V*
Invokes an interface method.
(Similar to INVOKEVIRTUAL but without virtual method table index optimization.)

INVOKEDYNAMIC *foo ()V bootstrap*
Queries the given *bootstrap method* for locating a method implementation at runtime.
(`MethodHandle`: Combines a specific method and an `INVOKE*` instruction.)

INVOKESTATIC *pkg/Bar foo ()V*
Invokes a static method.

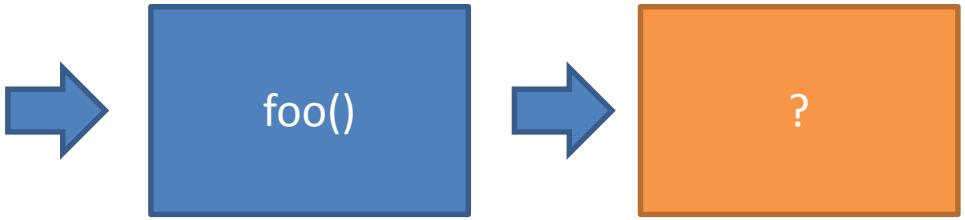
INVOKEVIRTUAL *pkg/Bar foo ()V*
Invokes the most-specific version of an inherited method on a non-interface class.

INVOKESPECIAL *pkg/Bar foo ()V*
Invokes a super class's version of an inherited method.
Invokes a "constructor method".
Invokes a private method.
Invokes an interface default method (Java 8).

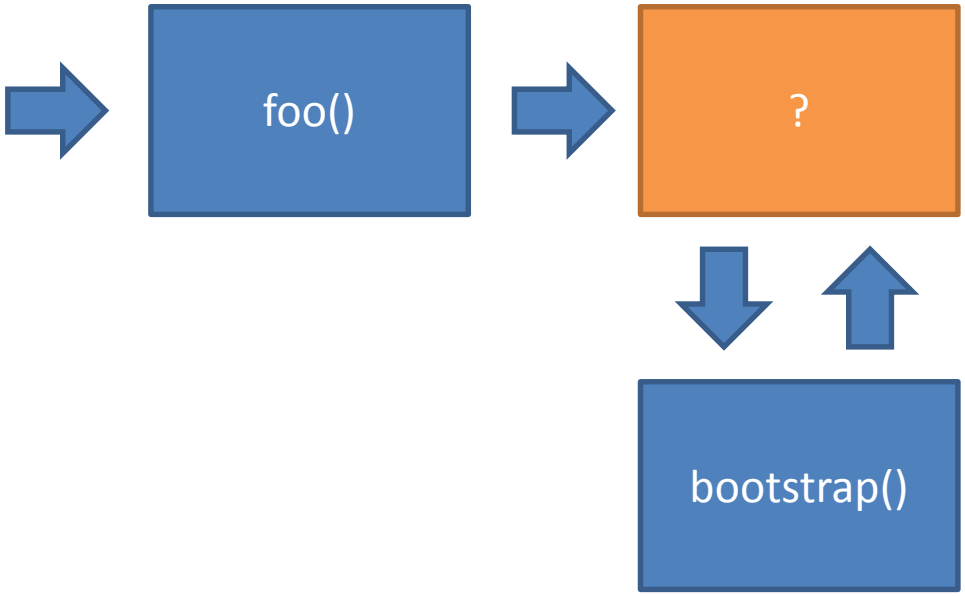
INVOKEINTERFACE *pkg/Bar foo ()V*
Invokes an interface method.
(Similar to INVOKEVIRTUAL but without virtual method table index optimization.)

INVOKEDYNAMIC *foo ()V bootstrap*
Queries the given *bootstrap method* for locating a method implementation at runtime.
(`MethodHandle`: Combines a specific method and an `INVOKE*` instruction.)

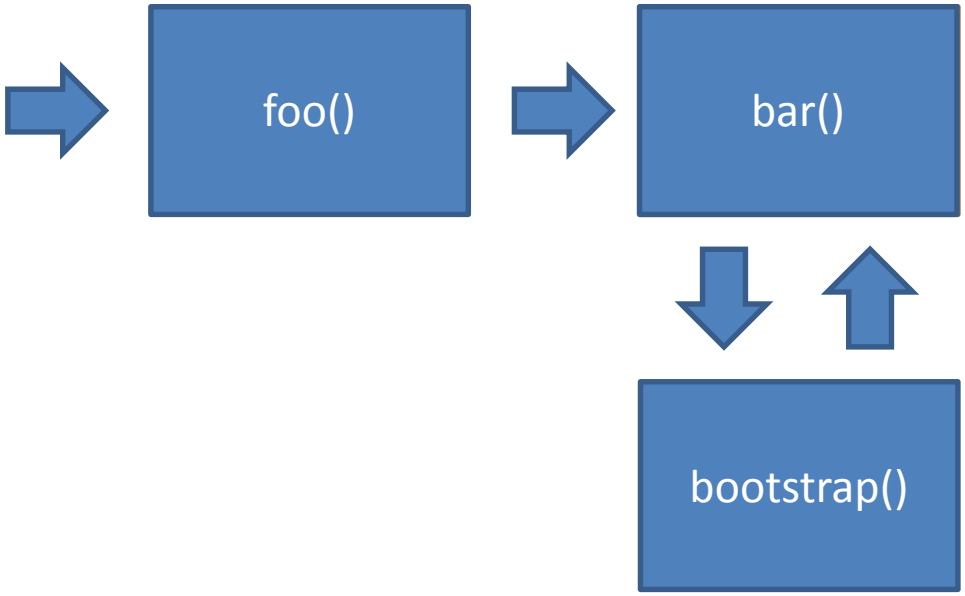
```
void foo() {
  ???(); // invokedynamic
}
```



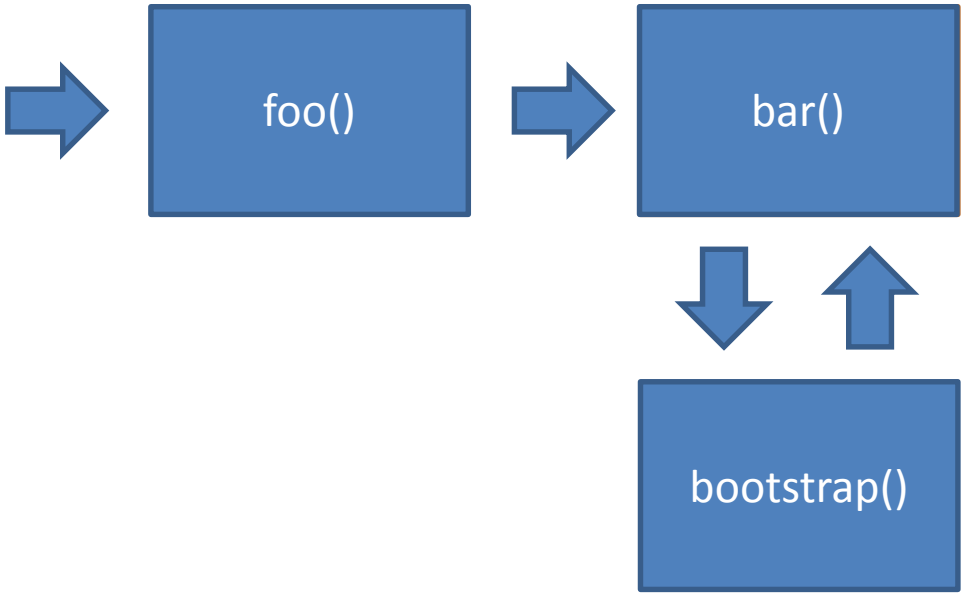
```
void foo() {
  ???(); // invokedynamic
}
```



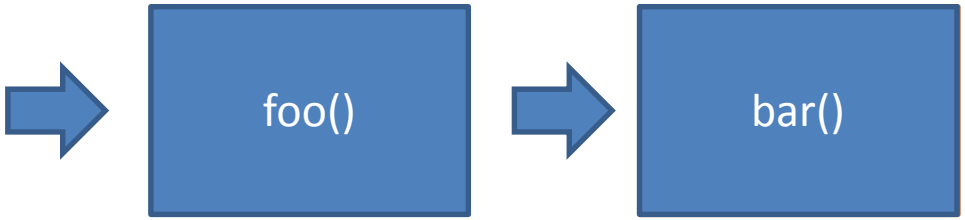
```
void foo() {
  ???(); // invokedynamic
}
```



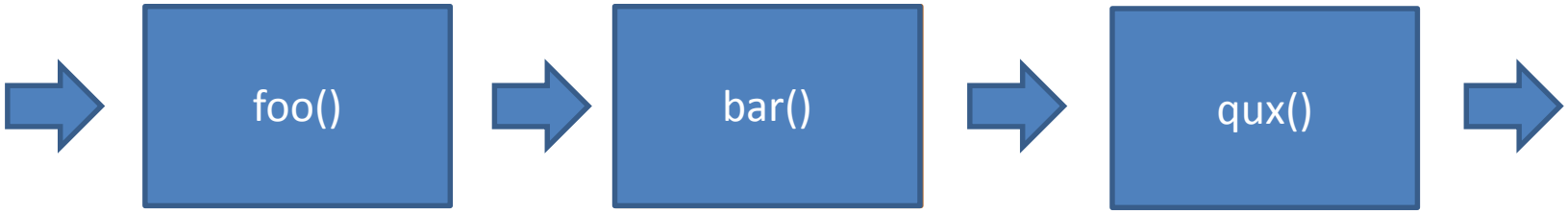
```
void foo() {
  bar();
}
```



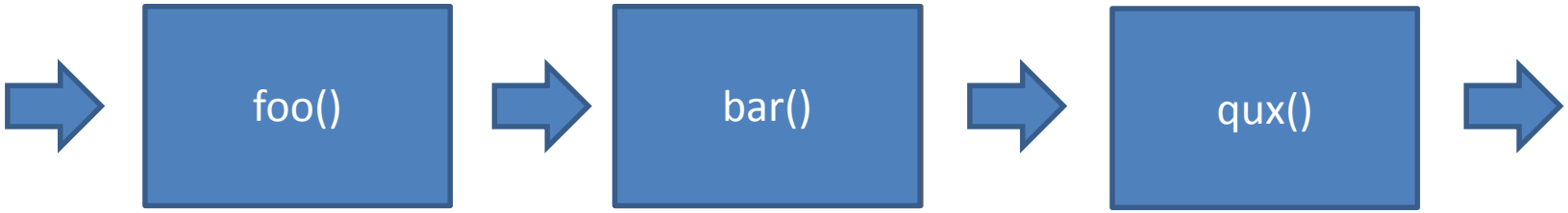
```
void foo() {
  bar();
}
```



```
void foo() {
  bar();
}
```



```
void foo() {
  bar();
}
```



In theory, this could be achieved by using reflection. However, using invokedynamic offers a more native approach. This is especially important when dealing with primitive types (double boxing).
Used to implement lambda expressions, more important for dynamic languages.

```
@interface Secured {
  String user();
}

class SecurityHolder {
  static String user = "ANONYMOUS";
}
```

```
@interface Secured {
  String user();
}

class SecurityHolder {
  static String user = "ANONYMOUS";
}
```

```
class Service {
  @Secured(user = "ADMIN")
  void deleteEverything() {
    // delete everything...
  }
}
```

```
interface Framework {
  <T> Class<? extends T> secure(Class<T> type);
}

@interface Secured {
  String user();
}

class SecurityHolder {
  static String user = "ANONYMOUS";
}
```

```
class Service {
  @Secured(user = "ADMIN")
  void deleteEverything() {
    // delete everything...
  }
}
```

```
interface Framework {
  <T> Class<? extends T> secure(Class<T> type);
}

@interface Secured {
  String user();
}

class SecurityHolder {
  static String user = "ANONYMOUS";
}
```

```
class Service {
  @Secured(user = "ADMIN")
  void deleteEverything() {
    // delete everything...
  }
}
```

```java
interface Framework {
  <T> Class<? extends T> secure(Class<T> type);
}

@interface Secured {
  String user();
}

class SecurityHolder {
  static String user = "ANONYMOUS";
}
```


```java
class Service {
  @Secured(user = "ADMIN")
  void deleteEverything() {
    // delete everything...
  }
}
```

```java
interface Framework {
  <T> Class<? extends T> secure(Class<T> type);
}

@interface Secured {
  String user();
}

class SecurityHolder {
  static String user = "ANONYMOUS";
}
```

depends on

does not know about

```java
class Service {
  @Secured(user = "ADMIN")
  void deleteEverything() {
    // delete everything...
  }
}
```

```
interface Framework {
  <T> Class<? extends T> secure(Class<T> type);
}

@interface Secured {
  String user();
}

class SecurityHolder {
  static String user = "ANONYMOUS";
}
```

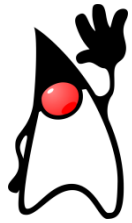
discovers at runtime

depends on

does not know about

```
class Service {
  @Secured(user = "ADMIN")
  void deleteEverything() {
    // delete everything...
  }
}
```

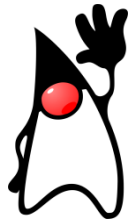
```
class Service {
  @Secured(user = "ADMIN")
  void deleteEverything() {
    if(!"ADMIN".equals(UserHolder.user)) {
      throw new IllegalStateException("Wrong user");
    }
    // delete everything...
  }
}
```



redefine class
(build time, agent)

```
class Service {
  @Secured(user = "ADMIN")
  void deleteEverything() {
    // delete everything...
  }
}
```

```
class SecuredService extends Service {
  @Override
  void deleteEverything() {
    if(!"ADMIN".equals(UserHolder.user)) {
      throw new IllegalStateException("Wrong user");
    }
    super.deleteEverything();
  }
}
```



create subclass
(Liskov substitution)

```
class Service {
  @Secured(user = "ADMIN")
  void deleteEverything() {
    // delete everything...
  }
}
```

The “black magic” prejudice.

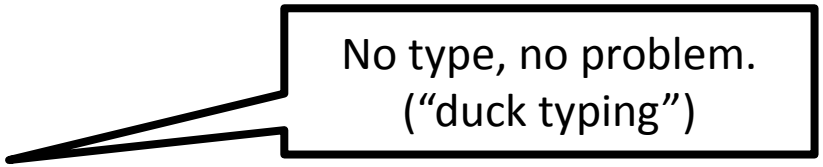
```
var service = {
  /* @Secured(user = "ADMIN") */
  deleteEverything: function () {
    // delete everything ...
  }
}
```

The “black magic” prejudice.

```
var service = {
  /* @Secured(user = "ADMIN") */
  deleteEverything: function () {
    // delete everything ...
  }
}

function run(service) {
  service.deleteEverything();
}
```

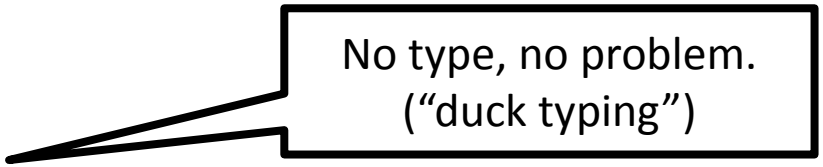
The “black magic” prejudice.

```
var service = {
  /* @Secured(user = "ADMIN") */
  deleteEverything: function () {
    // delete everything ...
  }
}


function run(service) {
  service.deleteEverything();
}
```



The “black magic” prejudice.

```
var service = {
  /* @Secured(user = "ADMIN") */
  deleteEverything: function () {
    // delete everything ...
  }
}

function run(service) {
  service.deleteEverything();
}
```



In dynamic languages (also those running on the JVM) this concept is applied a lot!
For framework implementors, type-safety is conceptually impossible.
But with type information available, we are at least able to **fail fast** when generating code at runtime in case that types do not match.

Isn't reflection meant for this?

```
class Class {
  Method getDeclaredMethod(String name,
                           Class<?>... parameterTypes)
      throws NoSuchMethodException,
             SecurityException;
}

class Method {
  Object invoke(Object obj,
                Object... args)
      throws IllegalAccessException,
             IllegalArgumentException,
             InvocationTargetException;
}
```

Isn't reflection meant for this?

```
class Class {
  Method getDeclaredMethod(String name,
                           Class<?>... parameterTypes)
     throws NoSuchMethodException,
            SecurityException;
}

class Method {
  Object invoke(Object obj,
                Object... args)
     throws IllegalAccessException,
            IllegalArgumentException,
            InvocationTargetException;
}
```

Reflection implies neither type-safety nor a notion of fail-fast.

Note: there are no performance gains when using code generation over reflection!

Thus, runtime code generation only makes sense for *user type enhancement*: While the framework code is less type safe, this type-unsafety does not spoil the user's code.

Byte Buddy: subclass creation

```
Class<?> dynamicType = new ByteBuddy()
  .subclass(Object.class)
  .method(named("toString"))
  .intercept(value("Hello World!"))
  .make()
  .load(getClass().getClassLoader(),
        ClassLoadingStrategy.Default.WRAPPER)
  .getLoaded();


assertThat(dynamicType.newInstance().toString(),
           is("Hello World!"));
```

Byte Buddy: subclass creation

```
Class<?> dynamicType = new ByteBuddy()
  .subclass(Object.class)
  .method(named("toString"))
  .intercept(value("Hello World!"))
  .make()
  .load(getClass().getClassLoader(),
        ClassLoadingStrategy.Default.WRAPPER)
  .getLoaded();


assertThat(dynamicType.newInstance().toString(),
           is("Hello World!"));
```

Byte Buddy: subclass creation

```
Class<?> dynamicType = new ByteBuddy()
  .subclass(Object.class)
  .method(named("toString"))
  .intercept(value("Hello World!"))
  .make()
  .load(getClass().getClassLoader(),
        ClassLoadingStrategy.Default.WRAPPER)
  .getLoaded();



assertThat(dynamicType.newInstance().toString(),
           is("Hello World!"));
```

Byte Buddy: subclass creation

```
Class<?> dynamicType = new ByteBuddy()
  .subclass(Object.class)
  .method(named("toString"))
  .intercept(value("Hello World!"))
  .make()
  .load(getClass().getClassLoader(),
        ClassLoadingStrategy.Default.WRAPPER)
  .getLoaded();



assertThat(dynamicType.newInstance().toString(),
           is("Hello World!"));
```

Byte Buddy: subclass creation

```
Class<?> dynamicType = new ByteBuddy()
  .subclass(Object.class)
  .method(named("toString"))
  .intercept(value("Hello World!"))
  .make()
  .load(getClass().getClassLoader(),
        ClassLoadingStrategy.Default.WRAPPER)
  .getLoaded();



assertThat(dynamicType.newInstance().toString(),
           is("Hello World!"));
```

Byte Buddy: subclass creation

```
Class<?> dynamicType = new ByteBuddy()
  .subclass(Object.class)
  .method(named("toString"))
  .intercept(value("Hello World!"))
  .make()
  .load(getClass().getClassLoader(),
        ClassLoadingStrategy.Default.WRAPPER)
  .getLoaded();


assertThat(dynamicType.newInstance().toString(),
           is("Hello World!"));
```

Byte Buddy: subclass creation

```
Class<?> dynamicType = new ByteBuddy()
  .subclass(Object.class)
  .method(named("toString"))
  .intercept(value("Hello World!"))
  .make()
  .load(getClass().getClassLoader(),
        ClassLoadingStrategy.Default.WRAPPER)
  .getLoaded();


assertThat(dynamicType.newInstance().toString(),
           is("Hello World!"));
```

Byte Buddy: subclass creation

```
Class<?> dynamicType = new ByteBuddy()
  .subclass(Object.class)
  .method(named("toString"))
  .intercept(value("Hello World!"))
  .make()
  .load(getClass().getClassLoader(),
        ClassLoadingStrategy.Default.WRAPPER)
  .getLoaded();
```

```
assertThat(dynamicType.newInstance().toString(),
           is("Hello World!"));
```

Byte Buddy: invocation delegation

```
Class<?> dynamicType = new ByteBuddy()
  .subclass(Object.class)
  .method(named("toString"))
  .intercept(to(MyInterceptor.class))
  .make()
  .load(getClass().getClassLoader(),
        ClassLoadingStrategy.Default.WRAPPER)
  .getLoaded();
```

```
class MyInterceptor {
  static String intercept() {
    return "Hello World";
  }
}
```

Byte Buddy: invocation delegation

```
Class<?> dynamicType = new ByteBuddy()
  .subclass(Object.class)
  .method(named("toString"))
  .intercept(to(MyInterceptor.class))
  .make()
  .load(getClass().getClassLoader(),
        ClassLoadingStrategy.Default.WRAPPER)
  .getLoaded();
```

```
class MyInterceptor {
  static String intercept() {
    return "Hello World";
  }
}
```

Byte Buddy: invocation delegation

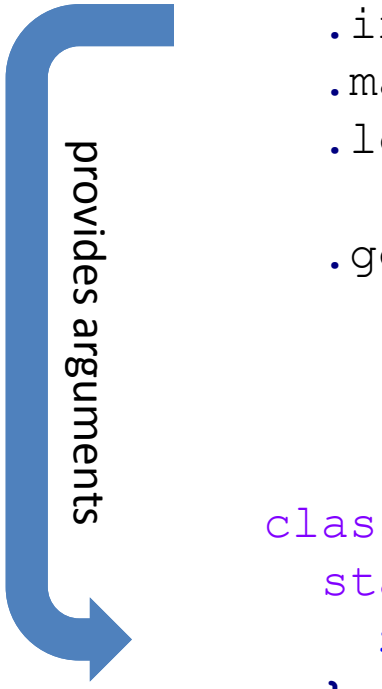
```
Class<?> dynamicType = new ByteBuddy()
  .subclass(Object.class)
  .method(named("toString"))
  .intercept(to(MyInterceptor.class))
  .make()
  .load(getClass().getClassLoader(),
        ClassLoadingStrategy.Default.WRAPPER)
  .getLoaded();
```

identifies best match

```
class MyInterceptor {
  static String intercept() {
    return "Hello World";
  }
}
```

Byte Buddy: invocation delegation (2)

```
Class<?> dynamicType = new ByteBuddy()
  .subclass(Object.class)
  .method(named("toString"))
  .intercept(to(MyInterceptor.class))
  .make()
  .load(getClass().getClassLoader(),
        ClassLoadingStrategy.Default.WRAPPER)
  .getLoaded();
```




```
class MyInterceptor {
  static String intercept(@Origin Method m) {
    return "Hello World from " + m.getName();
  }
}
```

Byte Buddy: invocation delegation (2)

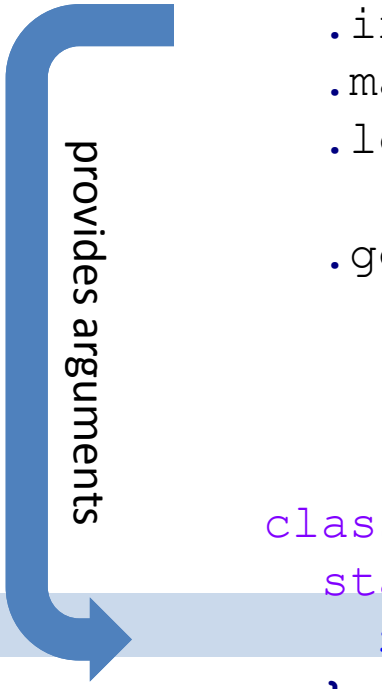
```
Class<?> dynamicType = new ByteBuddy()
  .subclass(Object.class)
  .method(named("toString"))
  .intercept(to(MyInterceptor.class))
  .make()
  .load(getClass().getClassLoader(),
        ClassLoadingStrategy.Default.WRAPPER)
  .getLoaded();
```

provides arguments

```
class MyInterceptor {
  static String intercept(@Origin Method m) {
    return "Hello World from " + m.getName();
  }
}
```

Byte Buddy: invocation delegation (2)

```
Class<?> dynamicType = new ByteBuddy()
  .subclass(Object.class)
  .method(named("toString"))
  .intercept(to(MyInterceptor.class))
  .make()
  .load(getClass().getClassLoader(),
        ClassLoadingStrategy.Default.WRAPPER)
  .getLoaded();
```




```
class MyInterceptor {
  static String intercept(@Origin Method m) {
    return "Hello World from " + m.getName();
  }
}
```

Annotations that are not on the class path are ignored at runtime.

Thus, Byte Buddy's classes can be used without Byte Buddy on the class path.

Byte Buddy: dependency injection

`@Origin Method|Class<?>|String`	Provides caller information
`@SuperCall Runnable|Callable<?>`	Allows super method call
`@DefaultCall Runnable|Callable<?>`	Allows default method call
`@AllArguments T[]`	Provides boxed method arguments
`@Argument(index) T`	Provides argument at the given index
`@This T`	Provides caller instance
`@Super T`	Provides super method proxy

Byte Buddy: runtime Hot Swap

```
class Foo {
  String bar() { return "bar"; }
}

Foo foo = new Foo();

new ByteBuddy()
  .redefine(Foo.class)
  .method(named("bar"))
  .intercept(value("Hello World!"))
  .make()
  .load(Foo.class.getClassLoader(),
        ClassReloadingStrategy.installedAgent());

assertThat(foo.bar(), is("Hello World!"));
```

Byte Buddy: runtime Hot Swap

```
class Foo {
  String bar() { return "bar"; }
}

Foo foo = new Foo();

new ByteBuddy()
  .redefine(Foo.class)
  .method(named("bar"))
  .intercept(value("Hello World!"))
  .make()
  .load(Foo.class.getClassLoader(),
        ClassReloadingStrategy.installedAgent());

assertThat(foo.bar(), is("Hello World!"));
```

Byte Buddy: runtime Hot Swap

```
class Foo {
  String bar() { return "bar"; }
}

Foo foo = new Foo();

new ByteBuddy()
  .redefine(Foo.class)
  .method(named("bar"))
  .intercept(value("Hello World!"))
  .make()
  .load(Foo.class.getClassLoader(),
        ClassReloadingStrategy.installedAgent());

assertThat(foo.bar(), is("Hello World!"));
```

Byte Buddy: runtime Hot Swap

```
class Foo {
  String bar() { return "bar"; }
}

Foo foo = new Foo();

new ByteBuddy()
  .redefine(Foo.class)
  .method(named("bar"))
  .intercept(value("Hello World!"))
  .make()
  .load(Foo.class.getClassLoader(),
        ClassReloadingStrategy.installedAgent());

assertThat(foo.bar(), is("Hello World!"));
```

The instrumentation API does not allow introduction of new methods.

This might change with JEP-159: Enhanced Class Redefiniton.

Byte Buddy: Java agents

```
class Foo {
  String bar() { return "bar"; }
}

assertThat(new Foo().bar(), is("Hello World!"));
```

Byte Buddy: Java agents

```
class Foo {
  String bar() { return "bar"; }
}

assertThat(new Foo().bar(), is("Hello World!"));
```


```
public static void premain(String arguments,
    Instrumentation instrumentation) {
  new AgentBuilder.Default()
    .rebase(named("Foo"))
    .transform( (builder, type) -> builder
          .method(named("bar"))
          .intercept(value("Hello World!"));
     )
    .installOn(instrumentation);
}
```

Byte Buddy: Java agents

```
class Foo {
  String bar() { return "bar"; }
}

assertThat(new Foo().bar(), is("Hello World!"));
```


```
public static void premain(String arguments,
    Instrumentation instrumentation) {
  new AgentBuilder.Default()
    .rebase(named("Foo"))
    .transform( (builder, type) -> builder
          .method(named("bar"))
          .intercept(value("Hello World!"));
     )
    .installOn(instrumentation);
}
```

Byte Buddy: Java agents

```
class Foo {
  String bar() { return "bar"; }
}

assertThat(new Foo().bar(), is("Hello World!"));
```

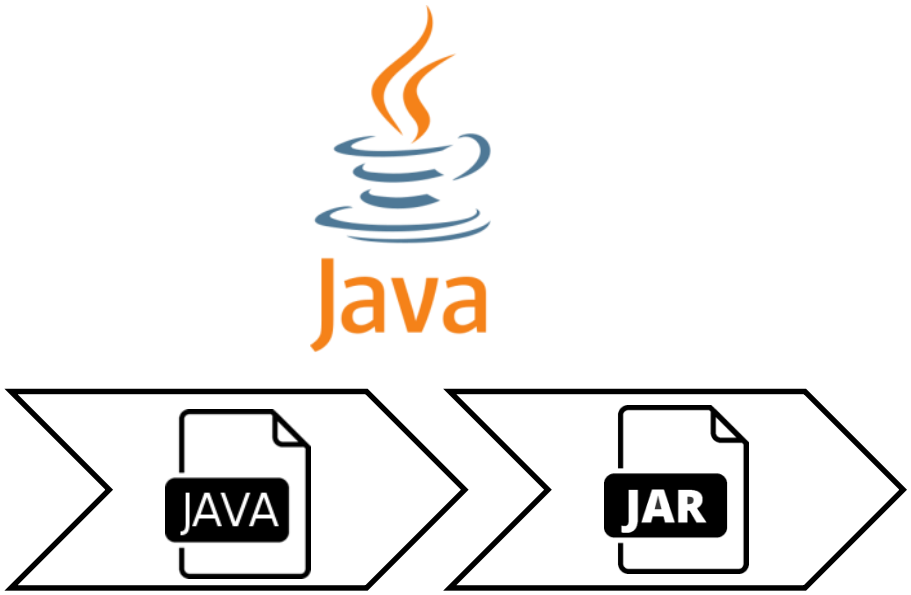

```
public static void premain(String arguments,
    Instrumentation instrumentation) {
  new AgentBuilder.Default()
    .rebase(named("Foo"))
    .transform( (builder, type) -> builder
          .method(named("bar"))
          .intercept(value("Hello World!"));
     )
    .installOn(instrumentation);
}
```

Byte Buddy: Java agents

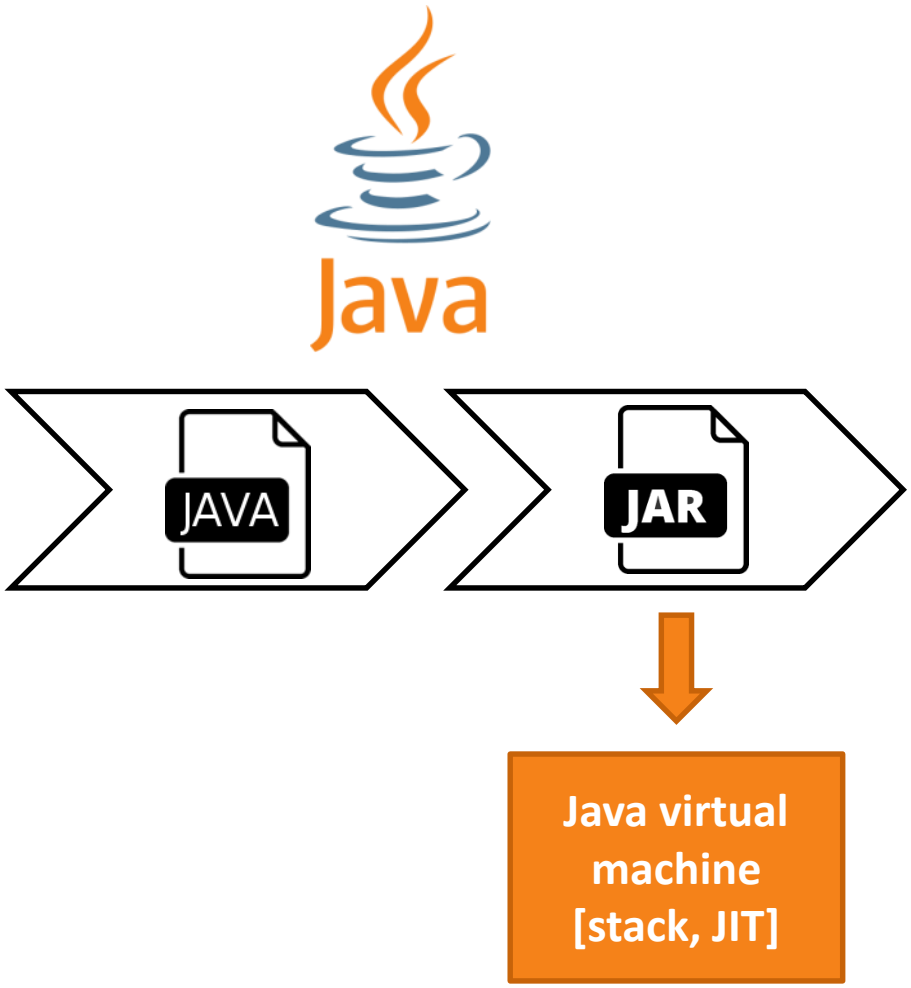
```
class Foo {
  String bar() { return "bar"; }
}

assertThat(new Foo().bar(), is("Hello World!"));
```


```
public static void premain(String arguments,
    Instrumentation instrumentation) {
  new AgentBuilder.Default()
    .rebase(named("Foo"))
    .transform( (builder, type) -> builder
          .method(named("bar"))
          .intercept(value("Hello World!"));
     )
    .installOn(instrumentation);
}
```

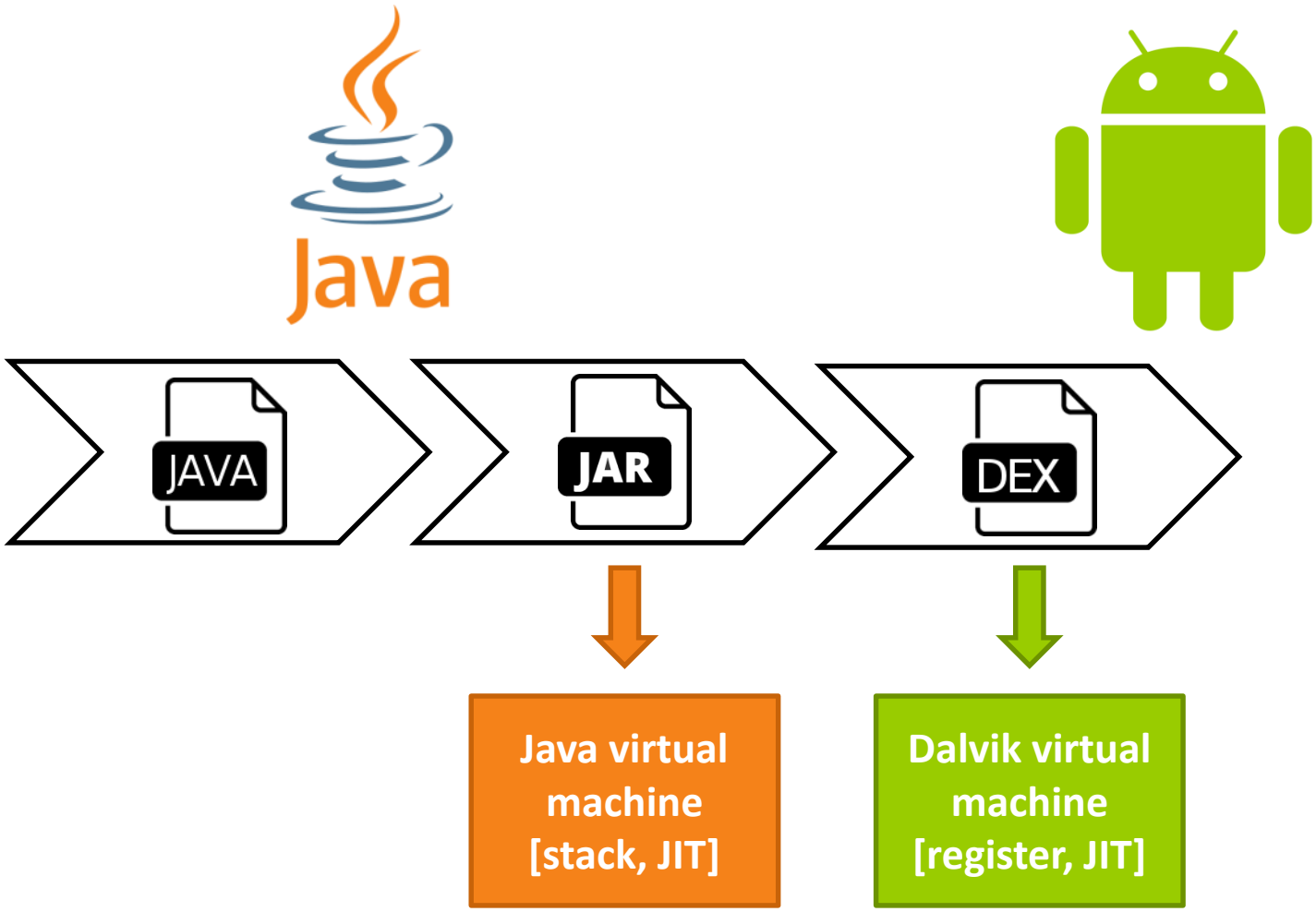
Android makes things more complicated.



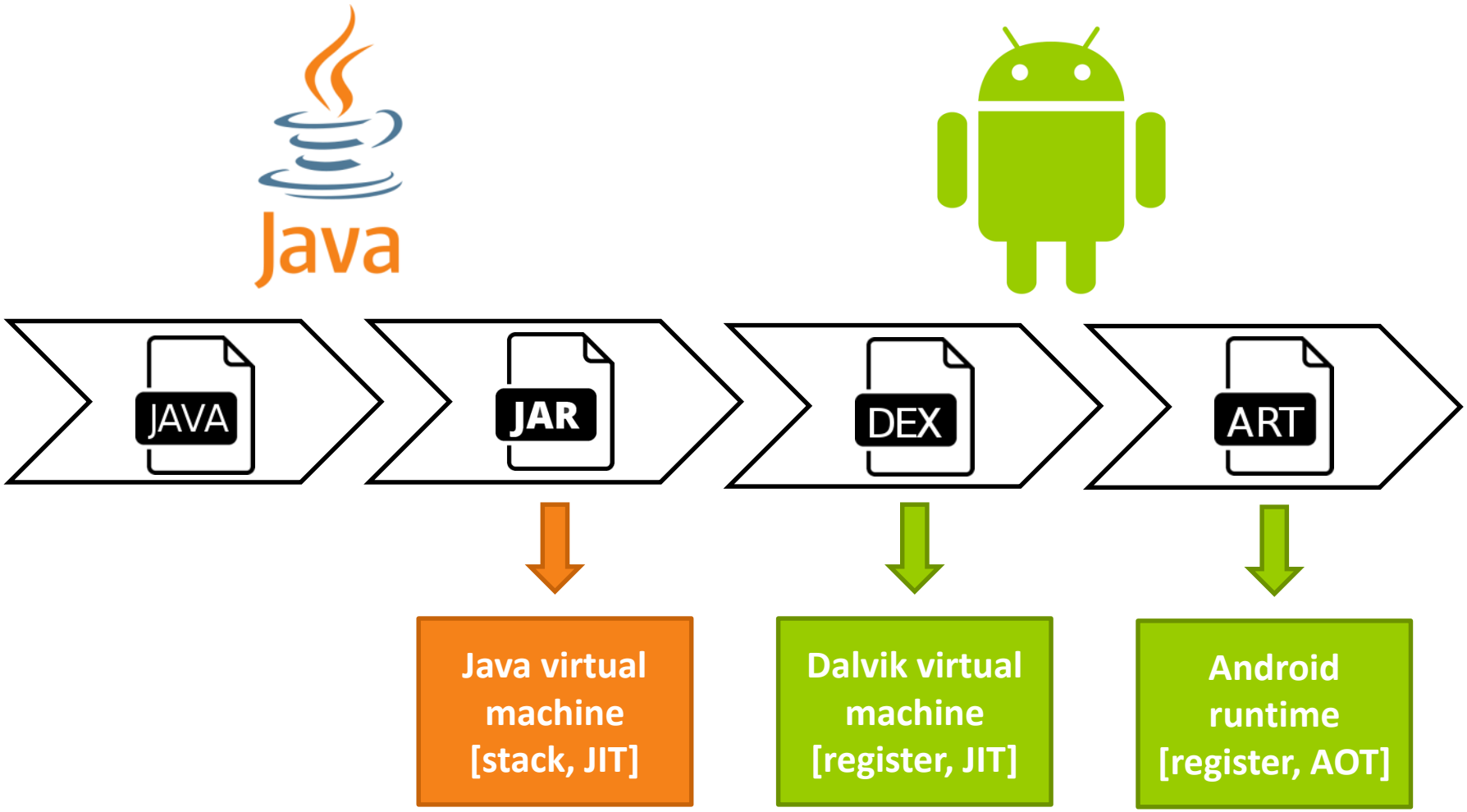
Android makes things more complicated.



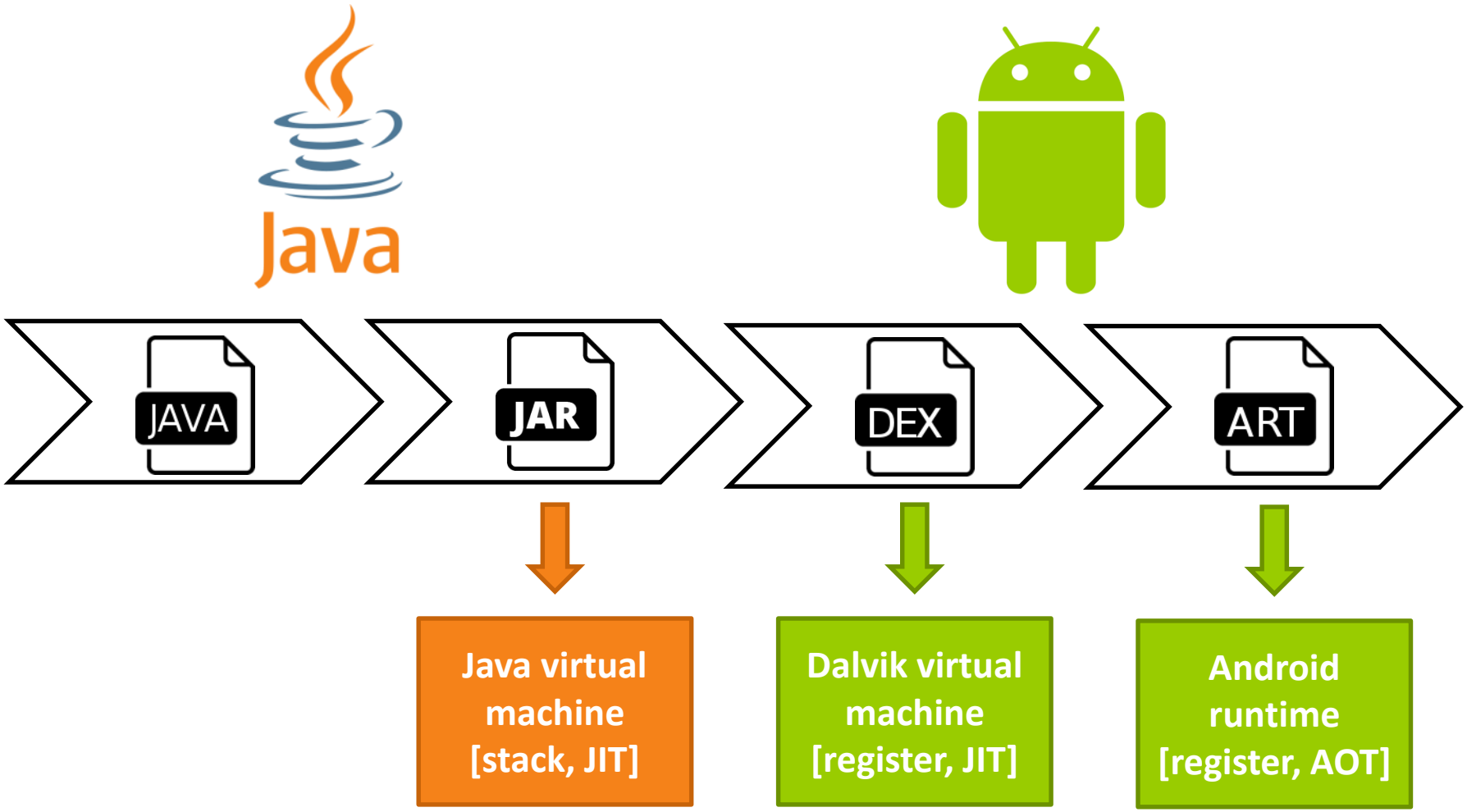
Android makes things more complicated.



Android makes things more complicated.



Android makes things more complicated.



Solution: Embed the Android SDK's dex compiler (Apache 2.0 license).

Unfortunately, no recent version central repository deployments of Android SDK.

	Byte Buddy	cglib	Javassist	Java proxy
(1)	60.995	234.488	145.412	68.706
(2a)	153.800	804.000	706.878	973.650
(2b)	0.001	0.002	0.009	0.005
(3a)	172.126 *1'850.567*	1'480.525	625.778	n/a
(3b)	0.002 *0.003*	0.019	0.027	n/a

All benchmarks run with JMH, source code: https://github.com/raphw/byte-buddy
(1) Extending the Object class without any methods but with a default constructor
(2a) Implementing an interface with 18 methods, method stubs
(2b) Executing a method of this interface
(3a) Extending a class with 18 methods, super method invocation
(3b) Executing a method of this class

http://rafael.codes
@rafaelcodes

http://documents4j.com
https://github.com/documents4j/documents4j

http://bytebuddy.net
https://github.com/raphw/byte-buddy