Android 292

Jérôme Pilliet

Université Paris-Est Marne-la-Vallée

Forewords

Dynamic languages

- Semantic determined at runtime

Smartphone / Tablet

- Constraint memory
- Constraint computing power

Android and Dalvik

- OS most used
- Java technology - Dalvik

invokedynamic
Example

```
class Foo {
    def bar(a, b=5, c=2:float, d=10) {
        ...
    }
}
```

.
.
.
.

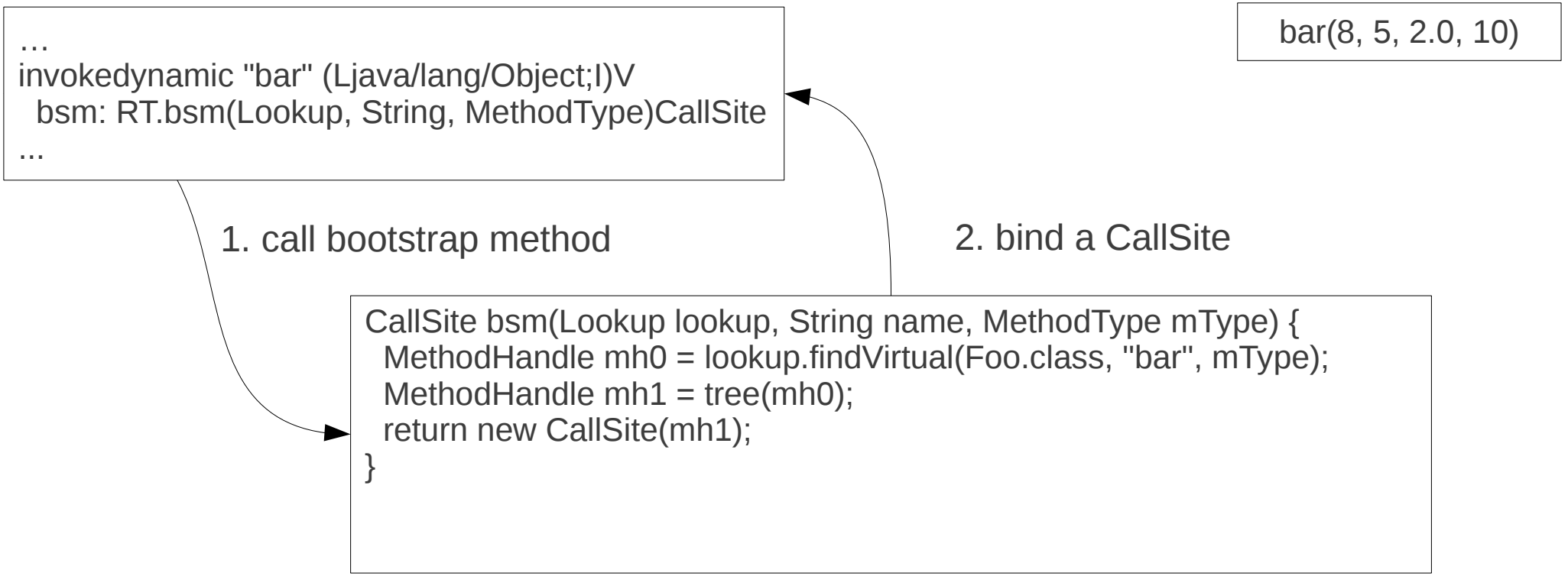
```
...
o.bar(8)
...
```

~~invokevirtual java/lang/Object.bar:(I)V~~

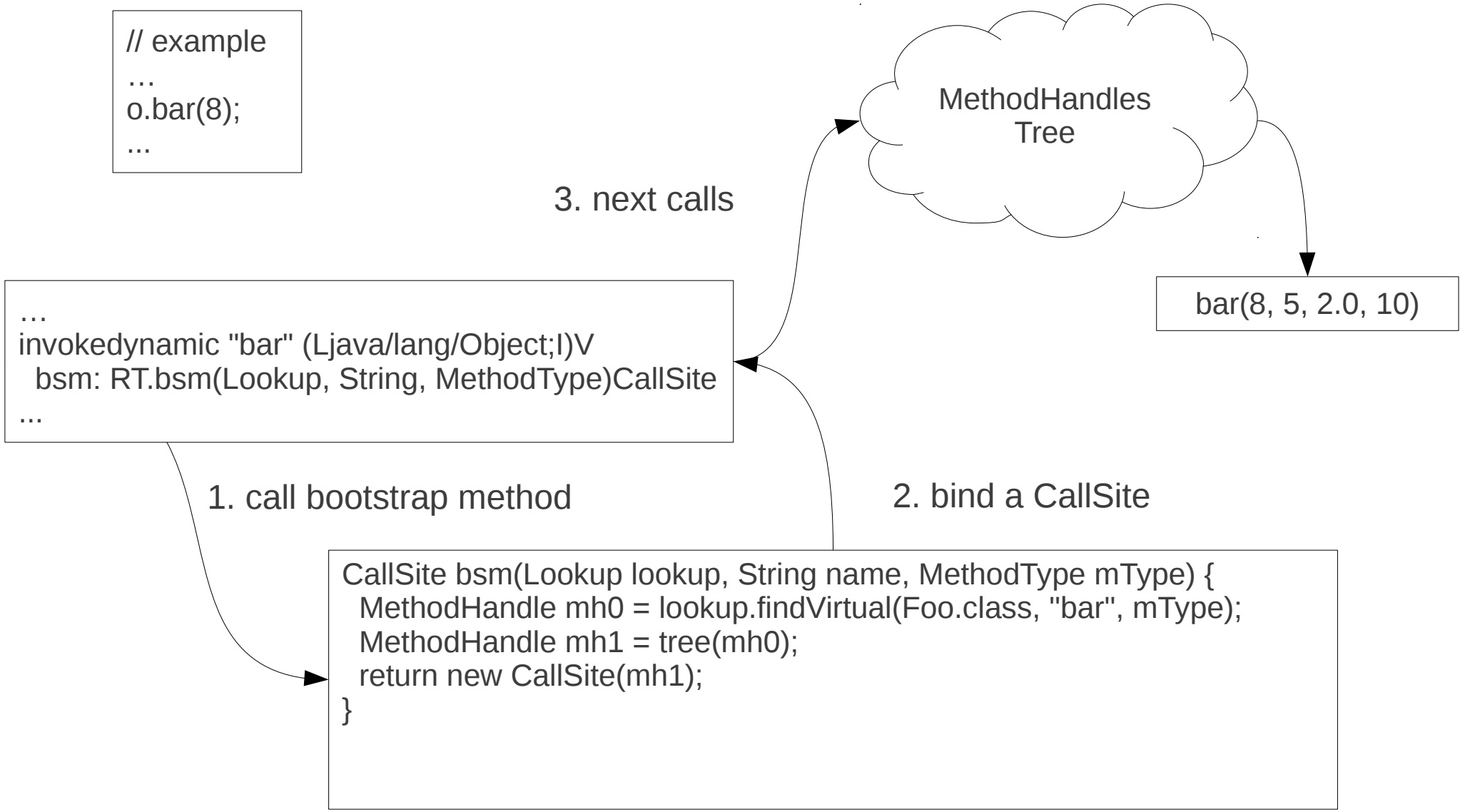
invokedynamic "bar":(Ljava/lang/Object;I)V

invokedynamic
Example

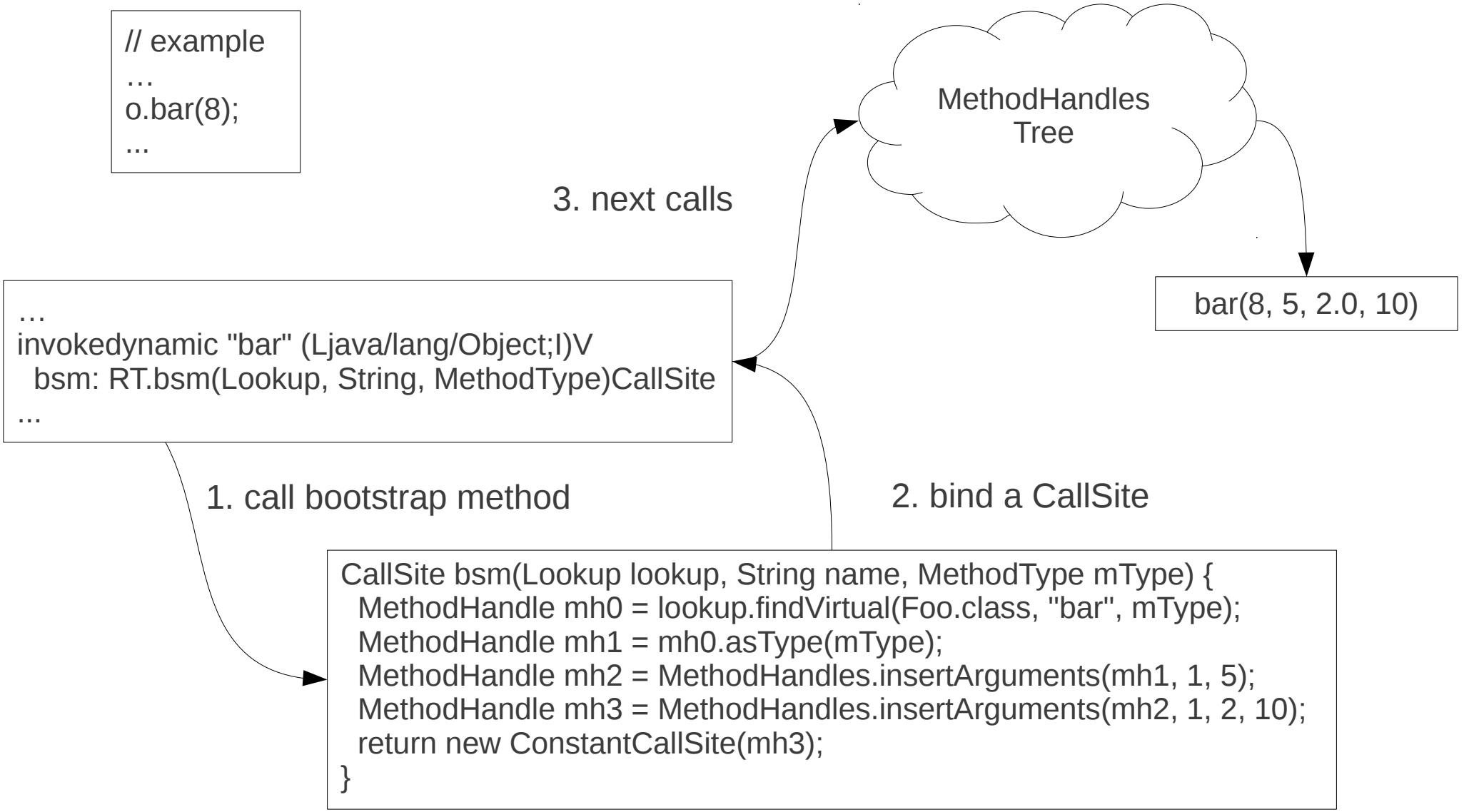
```
// example
…
o.bar(8);
...
```



invokedynamic
Example



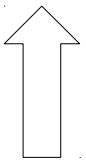
invokedynamic
Example



invokedynamic
Example

```
mh3 = MethodHandles.insertArguments(mh2, 1, 2, 10);
  ↳ mh2 = MethodHandles.insertArguments(mh1, 1, 5);
      ↳ mh1 = mh0.asType(mType);
          ↳ mh0 = lookup.findVirtual(Foo.class, "bar", mType);
```

```
bar(8)          - (I)V
bar(8,2,10)     - (III)V
bar(8,5,2,10)   - (IIII)V
bar(8,5,2.0,10) - (IIFI)V
```

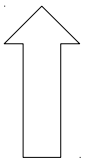


```
CallSite bsm(Lookup lookup, String name, MethodType mType) {
  MethodHandle mh0 = lookup.findVirtual(Foo.class, "bar", mType);
  MethodHandle mh1 = mh0.asType(mType);
  MethodHandle mh2 = MethodHandles.insertArguments(mh1, 1, 5);
  MethodHandle mh3 = MethodHandles.insertArguments(mh2, 1, 2, 10);
  return new ConstantCallSite(mh3);
}
```

invokedynamic
Example

mh3 = MethodHandles.insertArguments(mh2, 1, 2, 10);
↳ mh2 = MethodHandles.insertArguments(mh1, 1, 5);
↳ **mh1 = mh0.asType(mType);**
↳ mh0 = lookup.findVirtual(Foo.class, "bar", mType);

bar(8)	- (I)V
bar(8,2,10)	- (III)V
bar(8,5,2,10)	- (IIII)V
bar(8,5,2.0,10)	- (IIFI)V

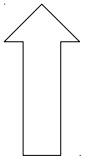


```
CallSite bsm(Lookup lookup, String name, MethodType mType) {
  MethodHandle mh0 = lookup.findVirtual(Foo.class, "bar", mType);
  MethodHandle mh1 = mh0.asType(mType);
  MethodHandle mh2 = MethodHandles.insertArguments(mh1, 1, 5);
  MethodHandle mh3 = MethodHandles.insertArguments(mh2, 1, 2, 10);
  return new ConstantCallSite(mh3);
}
```

invokedynamic
Example

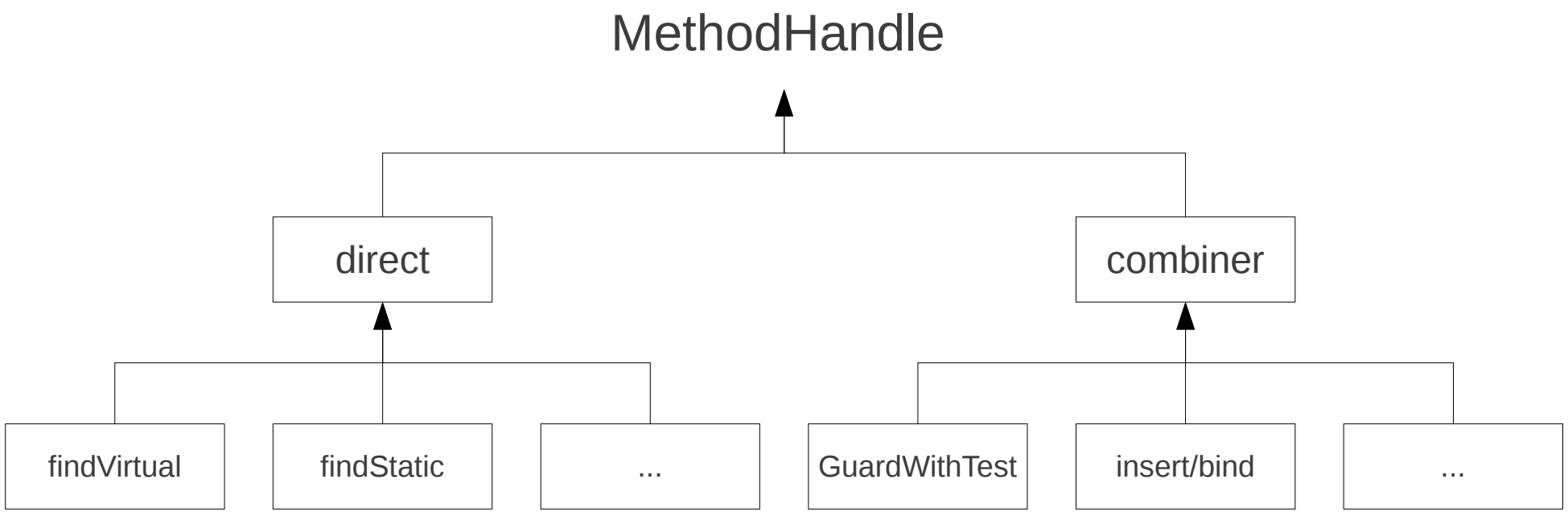
mh3_2 = MethodHandles.insertArguments(mh3_1, 2, 10);
↳ **mh3_1 = MethodHandles.insertArguments(mh2, 1, 2);**
↳ mh2 = MethodHandles.insertArguments(mh1_1, 1, 5);
↳ **mh1_1 = MethodHandles.filterArgument(mh0, 2, i2F);**
↳ mh0 = lookup.findVirtual(Foo.class, "bar", mType);

bar(8)	- (I)V
bar(8,10)	- (II)V
bar(8,2,10)	- (III)V
bar(8,5,2,10)	- (IIII)V
bar(8,5,2.0,10)	- (IIFI)V



```
CallSite bsm(Lookup lookup, String name, MethodType mType) {
  MethodHandle mh0 = lookup.findVirtual(Foo.class, "bar", mType);
  MethodHandle mh1 = mh0.asType(mType);
  MethodHandle mh2 = MethodHandles.insertArguments(mh1, 1, 5);
  MethodHandle mh3 = MethodHandles.insertArguments(mh2, 1, 2, 10);
  return new ConstantCallSite(mh3);
}
```

JSR 292
java.lang.invoke



Hotspot
Implementations

Two different implementations

- JDK7:
  - Interpreter : stubs in assembler + ricochet frame
  - JIT only constant method handle (static final + invokedynamic)
    - done by generating java code + classical JIT
- JDK8 :
  - Interpreter : written in Java (lambda form)
  - JIT both invokedynamic and method handle
    - done by generating java code from lambda form + classical JIT

We don't want/can't/think it's a good idea to generate java code

Android 292
Mini-Interpreter

MethodHandle.*invoke*

- mh.asType().invokeExact()
  - mh.asType is a MH tree

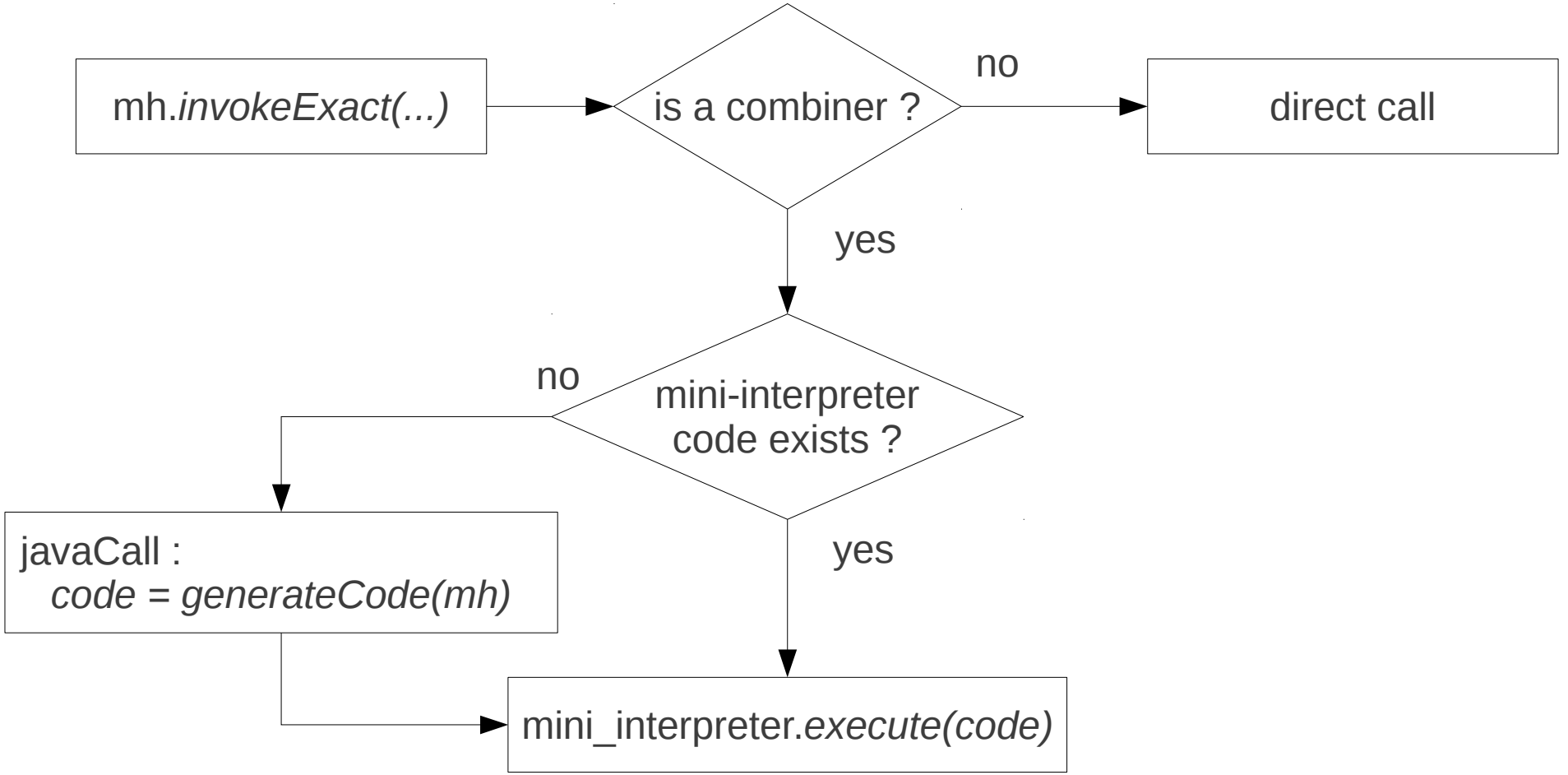
invokedynamic

- CS.getTarget().invokeExact()

MethodHandle.*invokeExact*

- MethodHandle direct
  - direct call, no intermediate stack frame
- MethodHandle Tree
  - mini-interpreter
  - specific calling convention, one specific stack frame

Android 292
Mini-Interpreter



Android 292
Mini-Interpreter

Move/permute/insert/filter/... stack values in **one** stack frame

Use an indirection encoded as integers to avoid to move values

- A stack frame of the mini interpreter is known and typed

Do most of the operations on 3 raw types (Object, 32bits, 64bits)

- Can reuse the same mini-interpreter code for several method handle trees

The mini-interpreter is register based (like Dalvik)

No mini-interpreter

- Hijack Dalvik interpreter to re-use Dalvik opcodes
- Add just one new opcode for the mini-interpreter

Android 292
Data Structures

MethodHandle.java

```
int  kind;       // invokestatic, invokevirtual, combiner
Method* m;  // Android raw method pointer
int index;      // field slot/vtable index
                                    …
```

CombinerMethodHandle.java

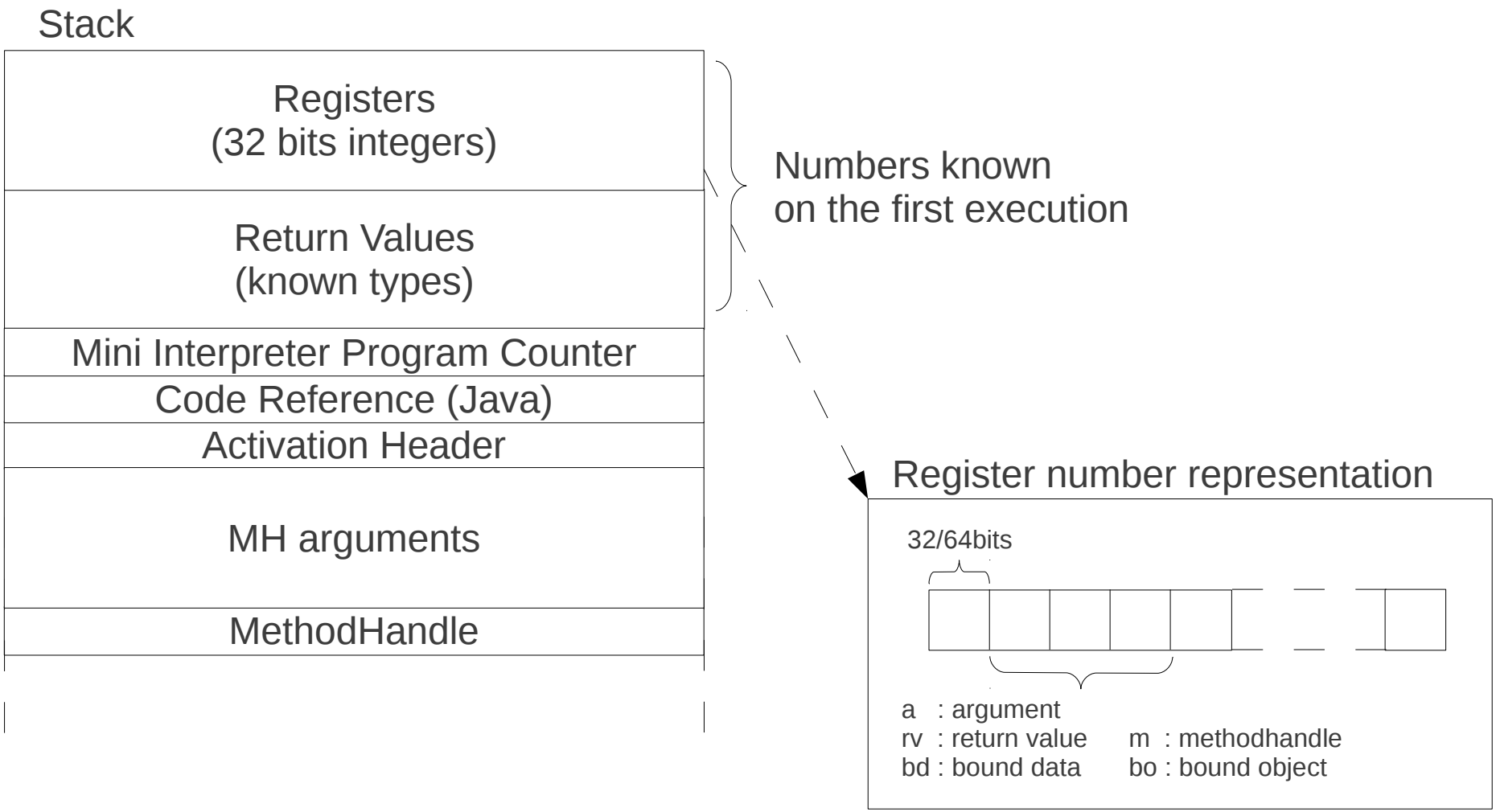
```
int nb_r;   // number of registers
int nb_rv; // number of return values

MethodHandle[] mhs; // list of methodhandles
byte[] code;                // code of methodhandle tree
                                    …
```

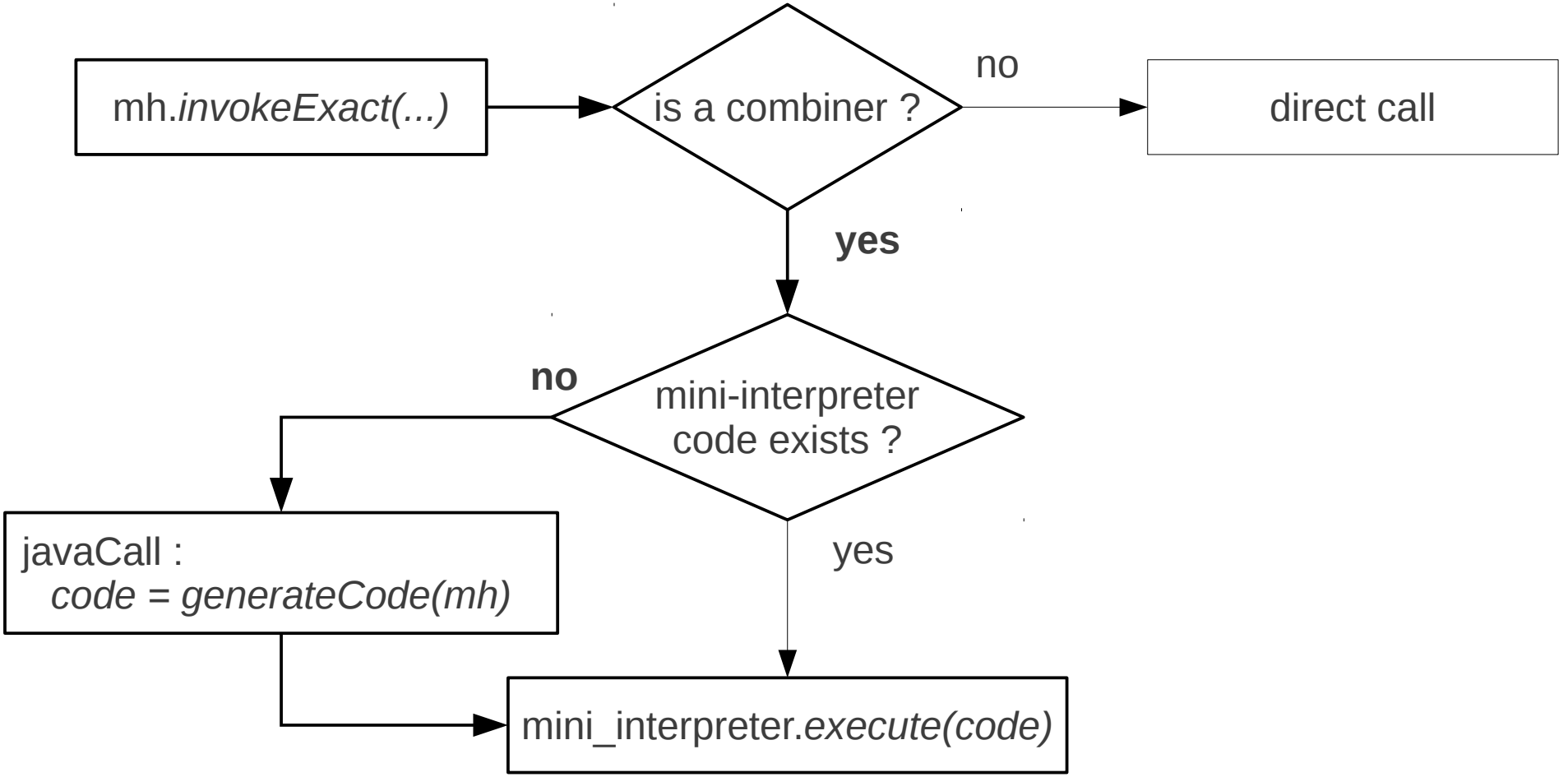
BoundMethodHandle.java

```
int bd32;      // 32bits bound primitive
long bd64;   // 64bits bound primitive
Object bo;   // bound object
                                    …
```

Android 292
Mini-Interpreter Stack Frame



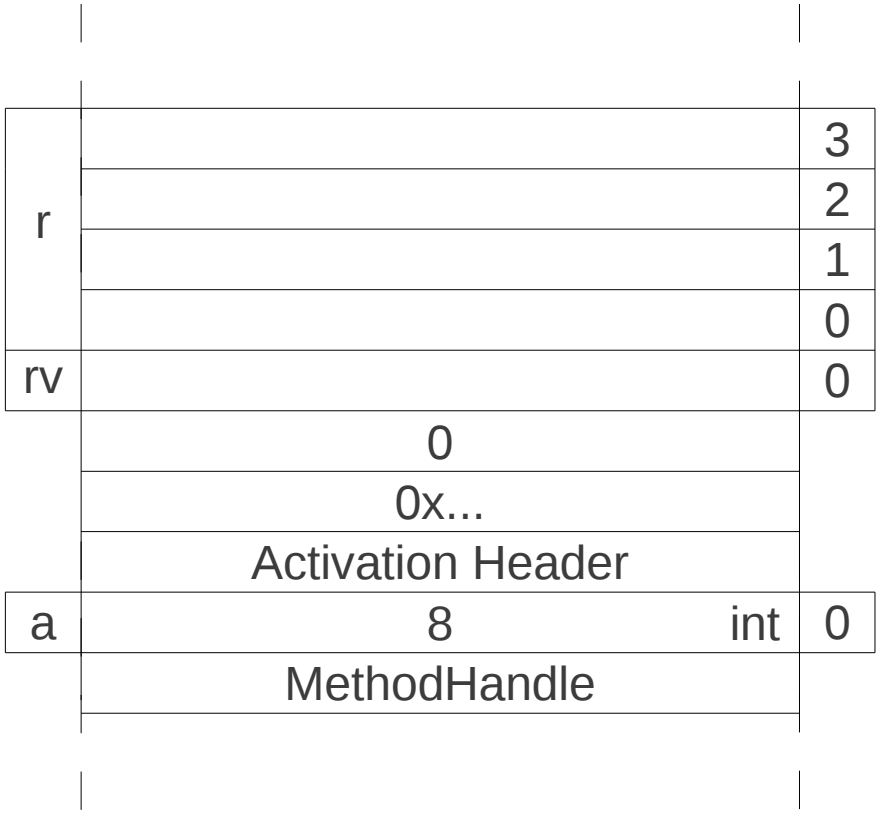
Android 292
Mini-Interpreter



Android 292
Example

```
mh3_2 = MethodHandles.insertArguments(mh3_1, 2, 10);
  ↳ mh3_1 = MethodHandles.insertArguments(mh2, 1, 2);
      ↳ mh2 = MethodHandles.insertArguments(mh1_1, 1, 5);
          ↳ mh1_1 = MethodHandles.filterArgument(mh0, 2, i2F);
              ↳ mh0 = lookup.findVirtual(Foo.class, "bar", mType);
```

```
bar(8)          - (I)V
bar(8,10)       - (II)V
bar(8,2,10)     - (III)V
bar(8,5,2,10)   - (IIII)V
bar(8,5,2.0,10) - (IIFI)V
```



program counter = code + 0

nb_r = 0 nb_rv = 0

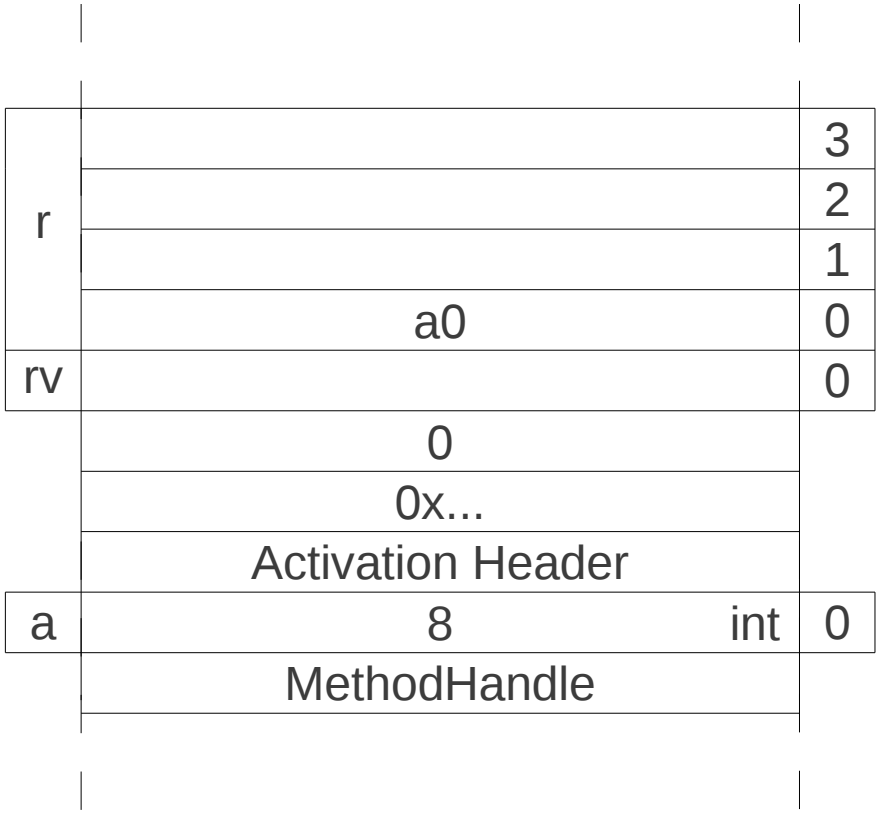
mhs = { mh3_2, mh3_1, mh2, mh1_1, mh0 }

```
code {
  const r0 a0,
  const r2 bd0,
  const r1 bd1,
  move r3 r2,
  move r2 r1,
  const r1 bd2,
  invoke_mh rv0 m3 r2 1,
  const r2 rv0,
  invoke_mh 0 m4 r0 4
}
```

Android 292
Example

```
mh3_2 = MethodHandles.insertArguments(mh3_1, 2, 10);
  ↳ mh3_1 = MethodHandles.insertArguments(mh2, 1, 2);
      ↳ mh2 = MethodHandles.insertArguments(mh1_1, 1, 5);
          ↳ mh1_1 = MethodHandles.filterArgument(mh0, 2, i2F);
              ↳ mh0 = lookup.findVirtual(Foo.class, "bar", mType);
```

bar(8)	**- (I)V**
bar(8,10)	- (II)V
bar(8,2,10)	- (III)V
bar(8,5,2,10)	- (IIII)V
bar(8,5,2.0,10)	- (IIFI)V



program counter = code + 1

nb_r = 1 nb_rv = 0

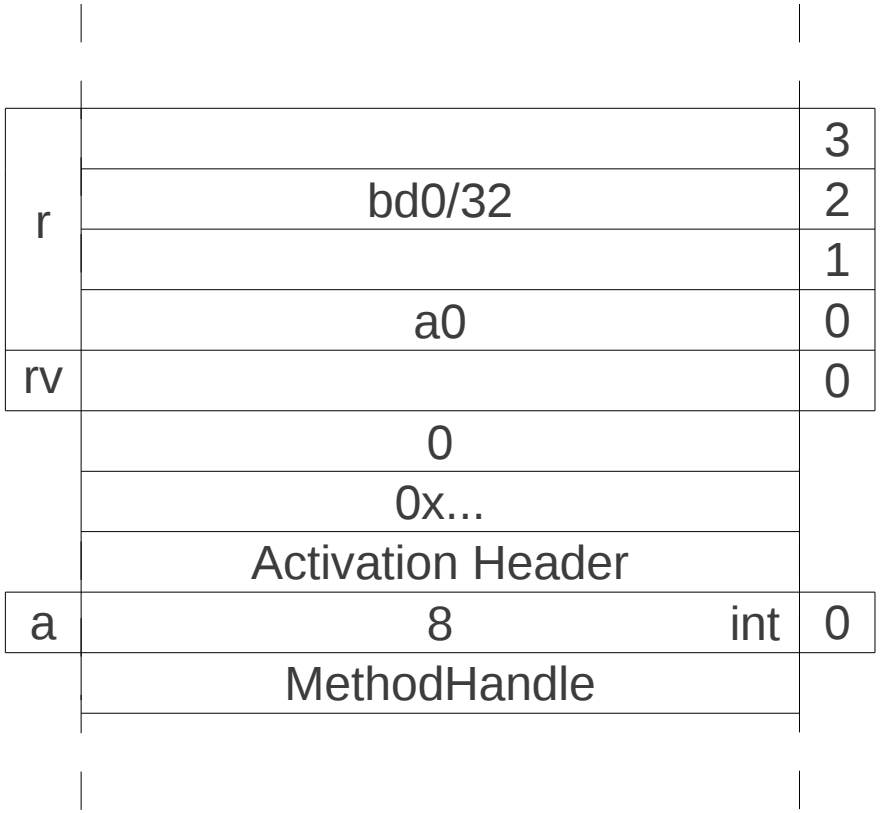
mhs = { mh3_2, mh3_1, mh2, mh1_1, mh0 }

```
code {
  const r0 a0,
  const r2 bd0,
  const r1 bd1,
  move r3 r2,
  move r2 r1,
  const r1 bd2,
  invoke_mh rv0 m3 r2 1,
  const r2 rv0,
  invoke_mh 0 m4 r0 4
}
```

Android 292
Example

mh3_2 = MethodHandles.insertArguments(mh3_1, 2, 10);
↳ mh3_1 = MethodHandles.insertArguments(mh2, 1, 2);
↳ mh2 = MethodHandles.insertArguments(mh1_1, 1, 5);
↳ mh1_1 = MethodHandles.filterArgument(mh0, 2, i2F);
↳ mh0 = lookup.findVirtual(Foo.class, "bar", mType);

	bar(8)		- (I)V
	bar(8,10)		**- (II)V**
	bar(8,2,10)		- (III)V
	bar(8,5,2,10)		- (IIII)V
	bar(8,5,2.0,10)		- (IIFI)V



program counter = code + 2

nb_r = 2 nb_rv = 0

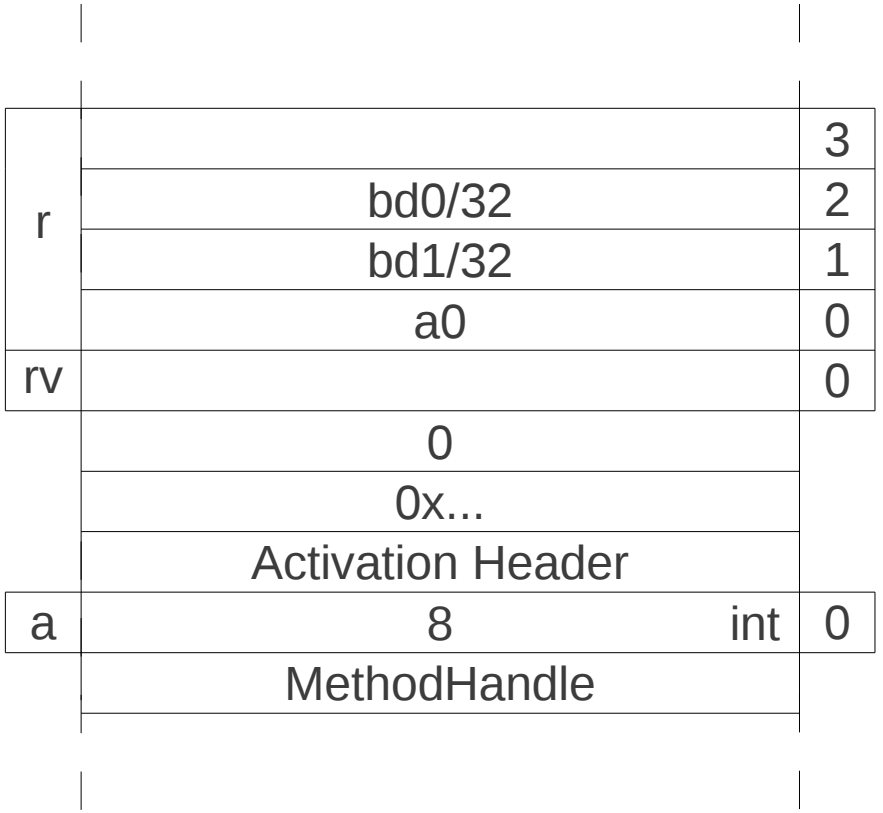
mhs = { **mh3_2**, mh3_1, mh2, mh1_1, mh0 }

```
code {
  const r0 a0,
  const r2 bd0,
  const r1 bd1,
  move r3 r2,
  move r2 r1,
  const r1 bd2,
  invoke_mh rv0 m3 r2 1,
  const r2 rv0,
  invoke_mh 0 m4 r0 4
}
```

Android 292
Example

```
mh3_2 = MethodHandles.insertArguments(mh3_1, 2, 10);
  ↳ mh3_1 = MethodHandles.insertArguments(mh2, 1, 2);
      ↳ mh2 = MethodHandles.insertArguments(mh1_1, 1, 5);
          ↳ mh1_1 = MethodHandles.filterArgument(mh0, 2, i2F);
              ↳ mh0 = lookup.findVirtual(Foo.class, "bar", mType);
```

bar(8)	- (I)V
bar(8,10)	- (II)V
bar(8,2,10)	**- (III)V**
bar(8,5,2,10)	- (IIII)V
bar(8,5,2.0,10)	- (IIFI)V



program counter = code + 3

nb_r = 3 nb_rv = 0

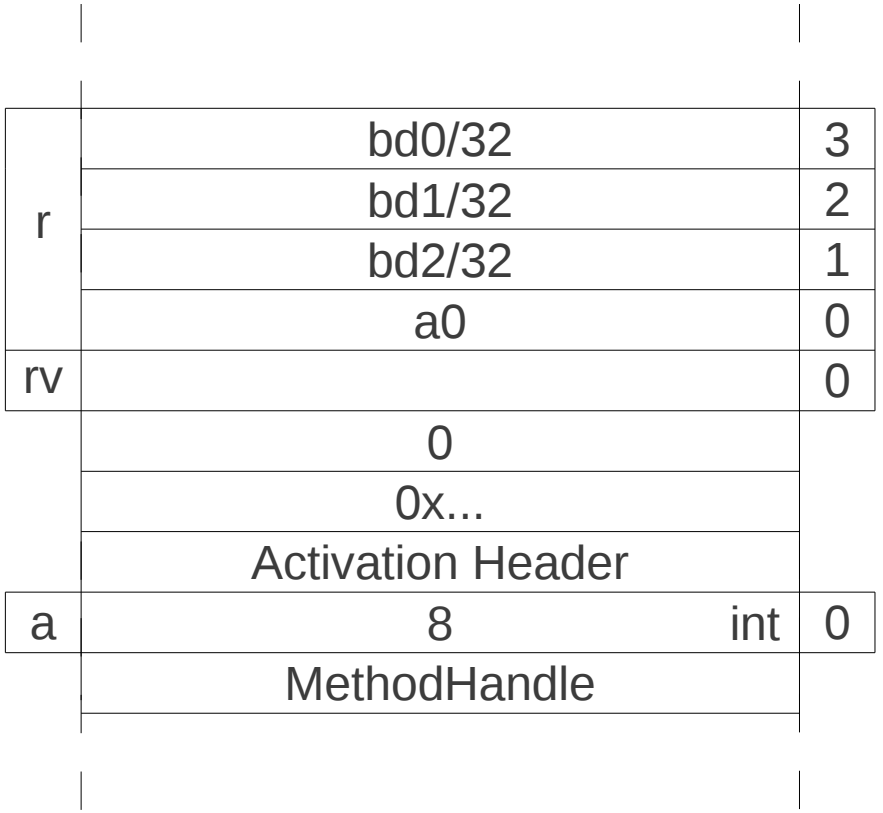
mhs = { mh3_2, **mh3_1**, mh2, mh1_1, mh0 }

```
code {
  const r0 a0,
  const r2 bd0,
  const r1 bd1,
  move r3 r2,
  move r2 r1,
  const r1 bd2,
  invoke_mh rv0 m3 r2 1,
  const r2 rv0,
  invoke_mh 0 m4 r0 4
}
```

Android 292
Example

```
mh3_2 = MethodHandles.insertArguments(mh3_1, 2, 10);
  ↳ mh3_1 = MethodHandles.insertArguments(mh2, 1, 2);
      ↳ mh2 = MethodHandles.insertArguments(mh1_1, 1, 5);
          ↳ mh1_1 = MethodHandles.filterArgument(mh0, 2, i2F);
              ↳ mh0 = lookup.findVirtual(Foo.class, "bar", mType);
```

	bar(8)		- (I)V
	bar(8,10)		- (II)V
	bar(8,2,10)		- (III)V
	bar(8,5,2,10)		**- (IIII)V**
	bar(8,5,2.0,10)		- (IIFI)V



program counter = code + 6

nb_r = 4 nb_rv = 0

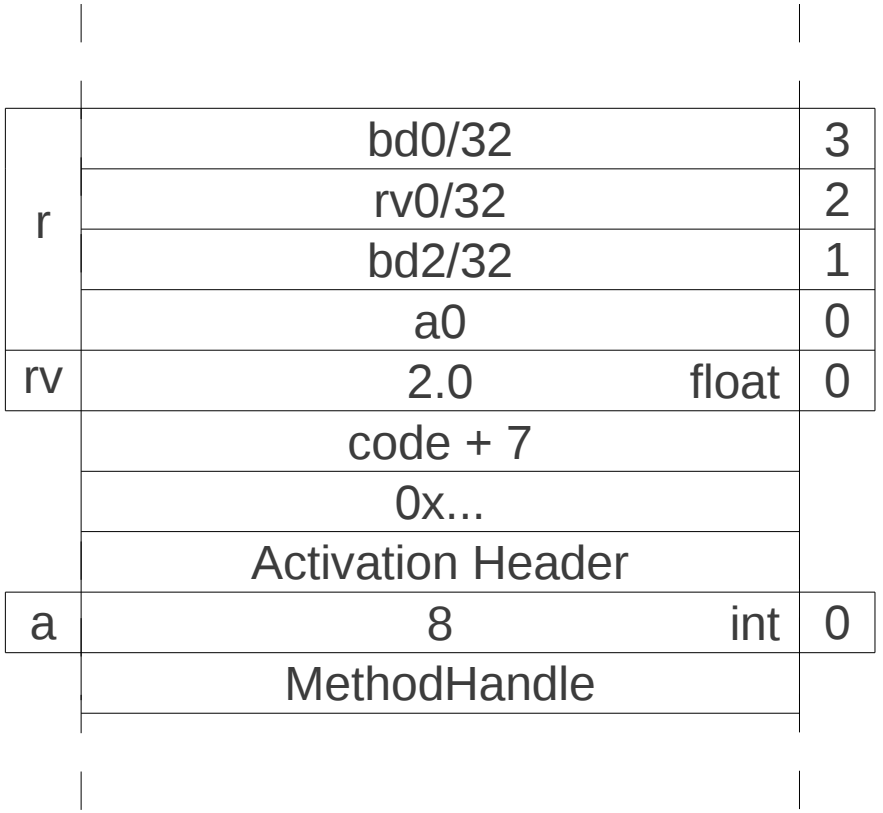
mhs = { mh3_2, mh3_1, **mh2**, mh1_1, mh0 }

```
code {
  const r0 a0,
  const r2 bd0,
  const r1 bd1,
  move r3 r2,
  move r2 r1,
  const r1 bd2,
  invoke_mh rv0 m3 r2 1,
  const r2 rv0,
  invoke_mh 0 m4 r0 4
}
```

Android 292
Example

```
mh3_2 = MethodHandles.insertArguments(mh3_1, 2, 10);
  ↳ mh3_1 = MethodHandles.insertArguments(mh2, 1, 2);
      ↳ mh2 = MethodHandles.insertArguments(mh1_1, 1, 5);
          ↳ mh1_1 = MethodHandles.filterArgument(mh0, 2, i2F);
              ↳ mh0 = lookup.findVirtual(Foo.class, "bar", mType);
```

	bar(8)		- (I)V
	bar(8,10)		- (II)V
	bar(8,2,10)		- (III)V
	bar(8,5,2,10)		- (IIII)V
	bar(8,5,2.0,10)		**- (IIFI)V**



program counter = code + 8

nb_r = 4 nb_rv = 1

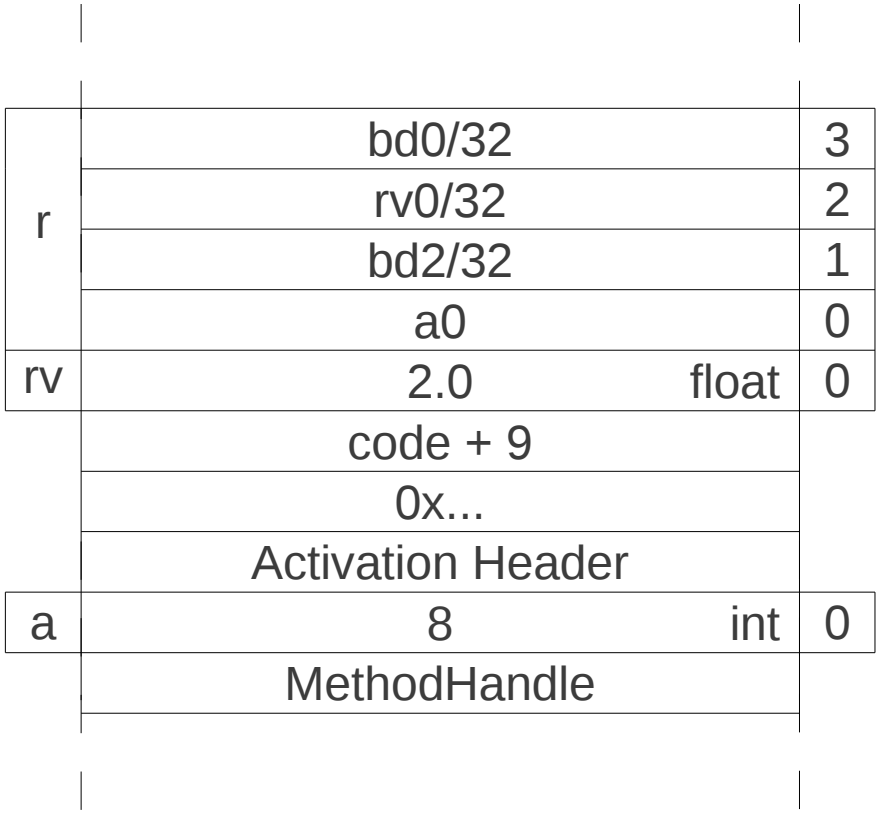
mhs = { mh3_2, mh3_1, mh2, **mh1_1**, mh0 }

```
code {
  const r0 a0,
  const r2 bd0,
  const r1 bd1,
  move r3 r2,
  move r2 r1,
  const r1 bd2,
  invoke_mh rv0 m3 r2 1,
  const r2 rv0,
  invoke_mh 0 m4 r0 4
}
```

Android 292
Example

```
mh3_2 = MethodHandles.insertArguments(mh3_1, 2, 10);
  ↳ mh3_1 = MethodHandles.insertArguments(mh2, 1, 2);
      ↳ mh2 = MethodHandles.insertArguments(mh1_1, 1, 5);
          ↳ mh1_1 = MethodHandles.filterArgument(mh0, 2, i2F);
              ↳ mh0 = lookup.findVirtual(Foo.class, "bar", mType);
```

	bar(8)		- (I)V
	bar(8,10)		- (II)V
	bar(8,2,10)		- (III)V
	bar(8,5,2,10)		- (IIII)V
	bar(8,5,2.0,10)		- (IIFI)V



program counter = code + 9

nb_r = 4 nb_rv = 1

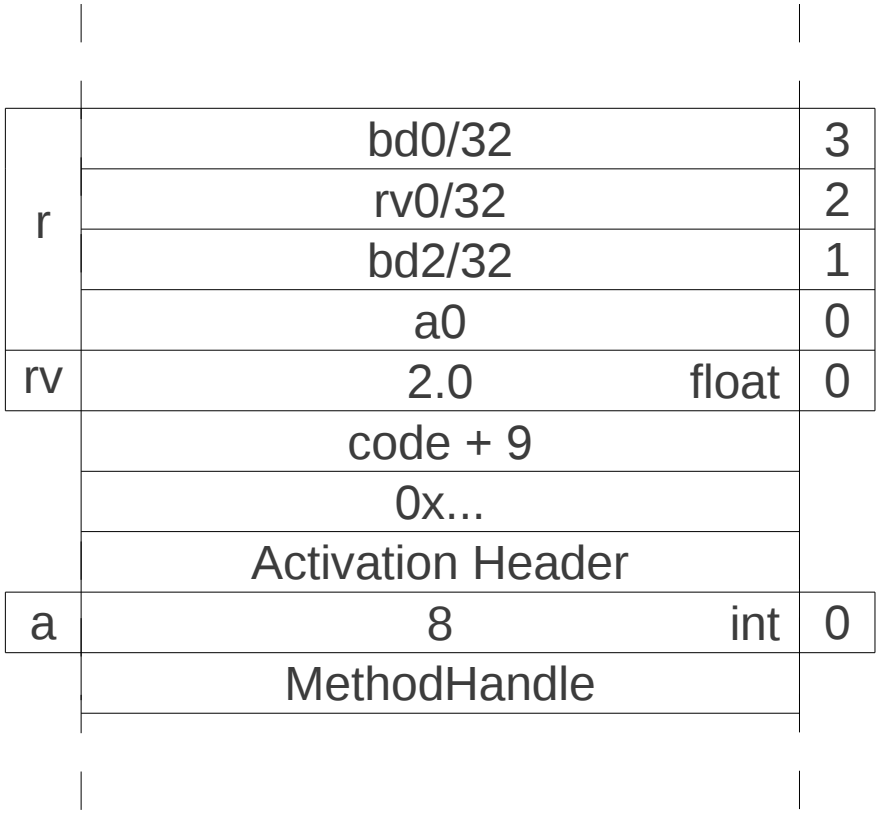
mhs = { mh3_2, mh3_1, mh2, mh1_1, **mh0** }

```
code {
  const r0 a0,
  const r2 bd0,
  const r1 bd1,
  move r3 r2,
  move r2 r1,
  const r1 bd2,
  invoke_mh rv0 m3 r2 1,
  const r2 rv0,
  invoke_mh 0 m4 r0 4
}
```

Android 292
Example

```
mh3_2 = MethodHandles.insertArguments(mh3_1, 2, 10);
  ↳ mh3_1 = MethodHandles.insertArguments(mh2, 1, 2);
      ↳ mh2 = MethodHandles.insertArguments(mh1_1, 1, 5);
          ↳ mh1_1 = MethodHandles.filterArgument(mh0, 2, i2F);
              ↳ mh0 = lookup.findVirtual(Foo.class, "bar", mType);
```

	bar(8)		- (I)V
	bar(8,10)		- (II)V
	bar(8,2,10)		- (III)V
	bar(8,5,2,10)		- (IIII)V
	bar(8,5,2.0,10)		- (IIFI)V



program counter = code + 9

nb_r = 4 nb_rv = 1

mhs = { mh3_2, mh3_1, mh2, mh1_1, mh0 }

```
code {
  const r0 a0,
  const r2 bd0,
  const r1 bd1,
  move r3 r2,
  move r2 r1,
  const r1 bd2,
  invoke_mh rv0 m3 r2 1,
  const r2 rv0,
  invoke_mh 0 m4 r0 4
}
```

Roadmap

Master thesis

- Prototype with Android 4.0
- Generation DEX code

Thesis (start sept 2012)

- Fresh restart with Android 4.2
- Android SDK compiles with java 7
- Add instructions : ldc_method*, invokeExact, invokeGeneric …
- Untainted implementation of java.lang.invoke
  - No SwitchPoint yet !
- Implementation of direct method handle in the interpreter
- Implementation of the mini interpreter

…

- Lambda proxy ? (Java 8)

Conclusion

Goal

Dalvik : a "Java" VM like the other

Open Questions

Perf ?

Can Hotspot Zero reuse the same scheme ?

Not obvious, java bytecode uses a stack

Can we transform the mini interpreter code to JIT instructions ?

Let say yes, all arguments of the opcodes are known

A specific abstract interpretation should work.

Android 292

https://bitbucket.org/jpilliet/android-292

Jérôme Pilliet
<pilliet@univ-mlv.fr>

Université Paris-Est Marne-la-Vallée