

The definite guide to Java agents

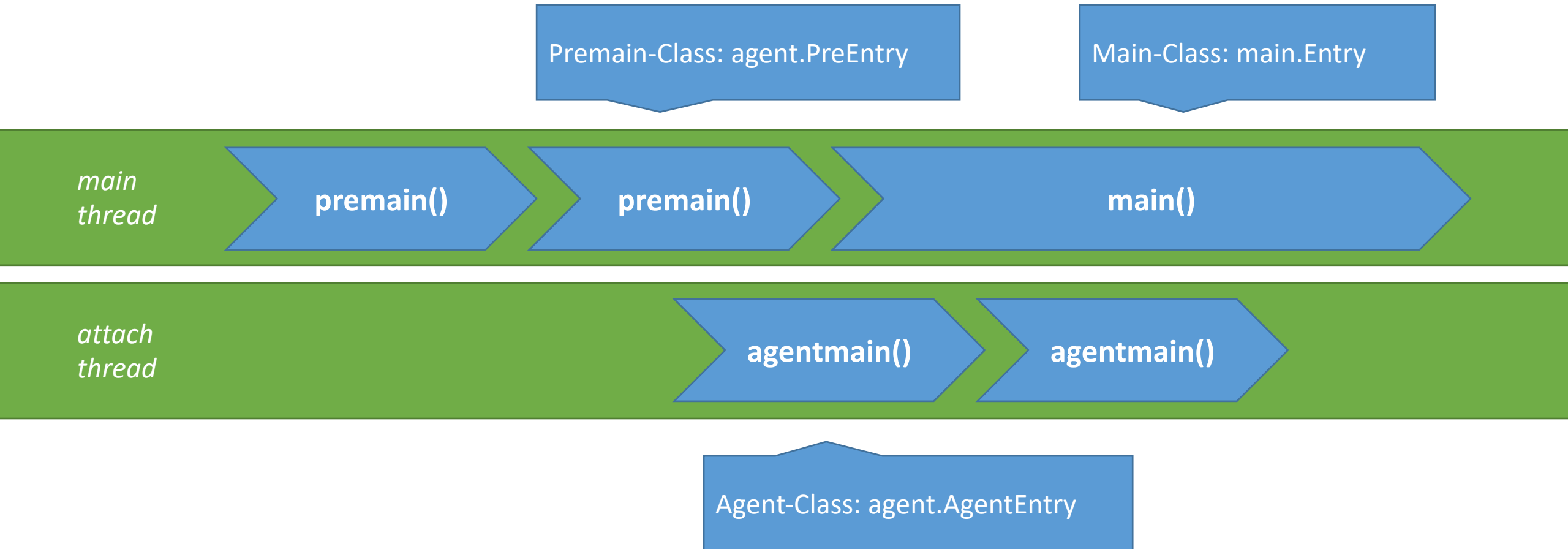
agent fundamentals

code instrumentation

class loading

miscellaneous

What are Java agents?



Running a static Java agent

```
java -cp:. -javaagent:/home/user/myagent.jar=MyAgent main.AppEntry
```

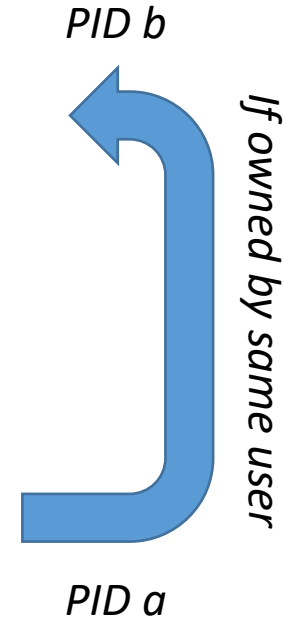
```
class PreEntry {  
    public static void premain(String arg) {  
        System.out.println("Hello from " + arg);  
    }  
}
```

Running a dynamic Java agent

```
java -cp:. main.AppEntry
```

```
class AgentEntry {  
    public static void agentmain(String arg) {  
        System.out.println("Hello from " + arg);  
    }  
}
```

```
VirtualMachine vm = VirtualMachine.attach(pid);  
try {  
    vm.loadAgent("/home/user/myagent.jar", "MyAgent");  
} finally {  
    vm.detach();  
}
```



Transforming a class

```
java -cp:. -javaagent:/home/user/myagent.jar=MyAgent main.AppEntry
```

```
class PreEntry {  
    public static void premain(String arg, Instrumentation inst) {  
        inst.addTransformer(  
            (classLoader, className, classIfLoaded, protectionDomain, classFile) -> {  
                if (shouldTransform(className)) {  
                    return transform(classFile);  
                } else {  
                    return null;  
                }  
            });  
    }  
}
```

Retransforming a class

```
java -cp:. main.AppEntry
```

```
class AgentEntry {  
    public static void agentmain(String arg, Instrumentation inst) {  
        inst.addTransformer(  
            (classLoader, className, classIfLoaded, protectionDomain, classFile) -> {  
                if (shouldTransform(classIfLoaded)) {  
                    return transform(classFile);  
                } else {  
                    return null;  
                }  
            }, true);  
        inst.retransformClasses(getClassesToRetransform());  
    }  
}
```

Retransforming a class: limitations

```
class Sample {  
    String m() {  
        return m2();  
    }  
} private String m2() {  
    return "bar";  
}  
}
```

Possible future extension: JEP 159 or Dynamic code evolution VM

But currently this does not seem to be a goal.

Running a native agent

```
java -agentlib:/home/user/myagent.so=MyAgent main.AppEntry
```

```
JNIEXPORT jint JNICALL Agent_OnLoad(  
    JavaVM *jvm, char *options, void *reserved) {  
    jvmtiEnv *jvmti; jint res; jvmtiError error;  
    jvmtiCapabilities capabilities; jvmtiEventCallbacks callbacks;  
  
    res = (*jvm)->GetEnv(jvm, (void **)&jvmti, JVMTI_VERSION_1);  
  
    (void) memset(&capabilities, 0, sizeof(jvmtiCapabilities));  
    capabilities.can_retransform_classes = 1;  
    error = jvmti->AddCapabilities(&capabilities);  
  
    callbacks.ClassFileLoadHook = &myClassFileLoadHook;  
    error = (*jvmti)->SetEventCallbacks(jvmti, &callbacks, (jint) sizeof(callbacks));  
  
    return JNI_OK;  
}
```

How can classes be manipulated at runtime?

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

Foo.java

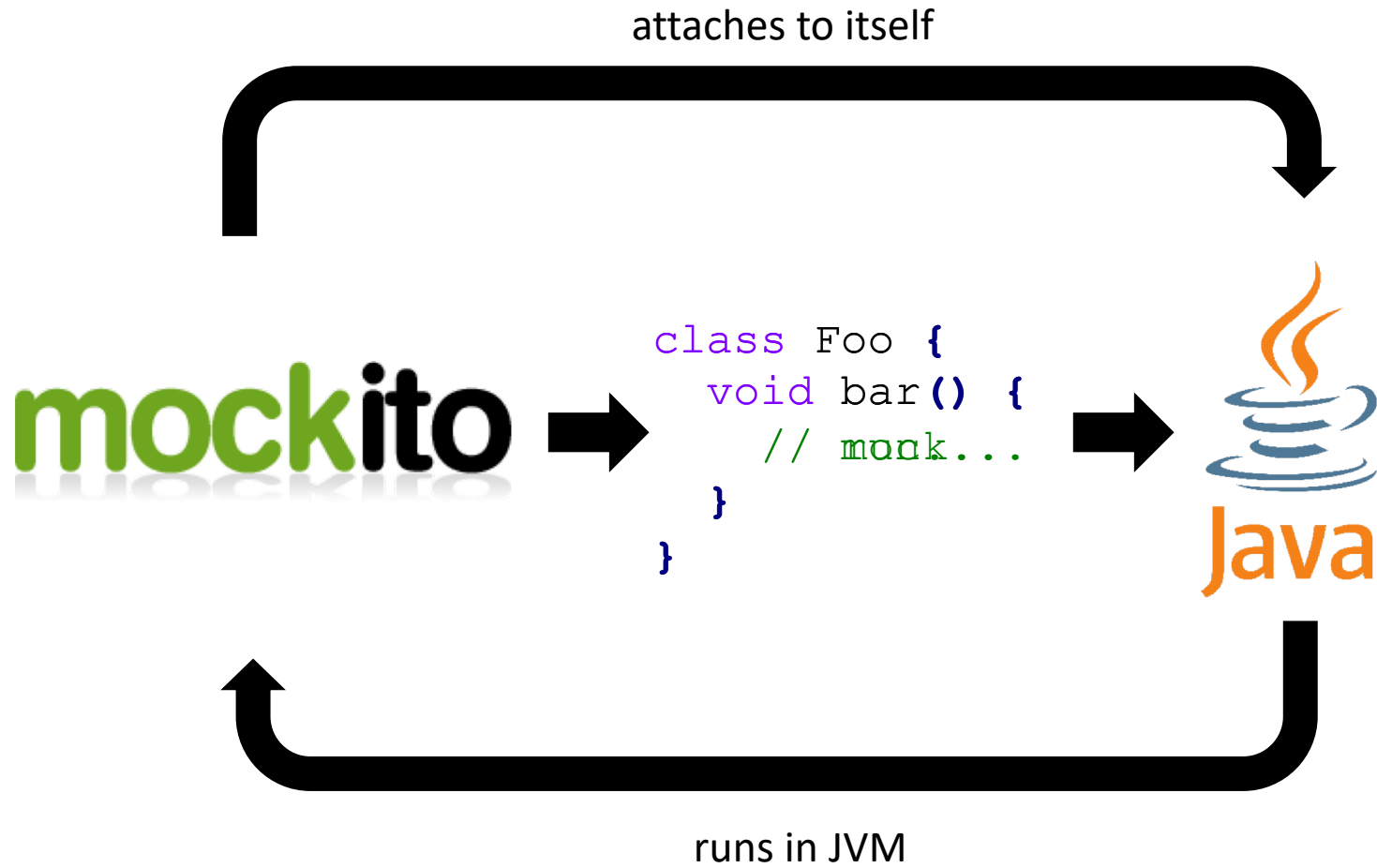
```
class Foo  
Constant pool:  
    #1 = String #4  
    #2 = Class #6  
    #3 = Utf8 Code  
    #4 = Utf8 bar  
    #5 = Utf8 ()Ljava/lang/String;  
    #6 = Utf8 Foo  
    #7 = Utf8 java/lang/Object  
java.lang.String bar();  
    descriptor: ()Ljava/lang/String;  
    Code:  
        0: ldc #2  
        2: areturn
```

Foo.class (javap)

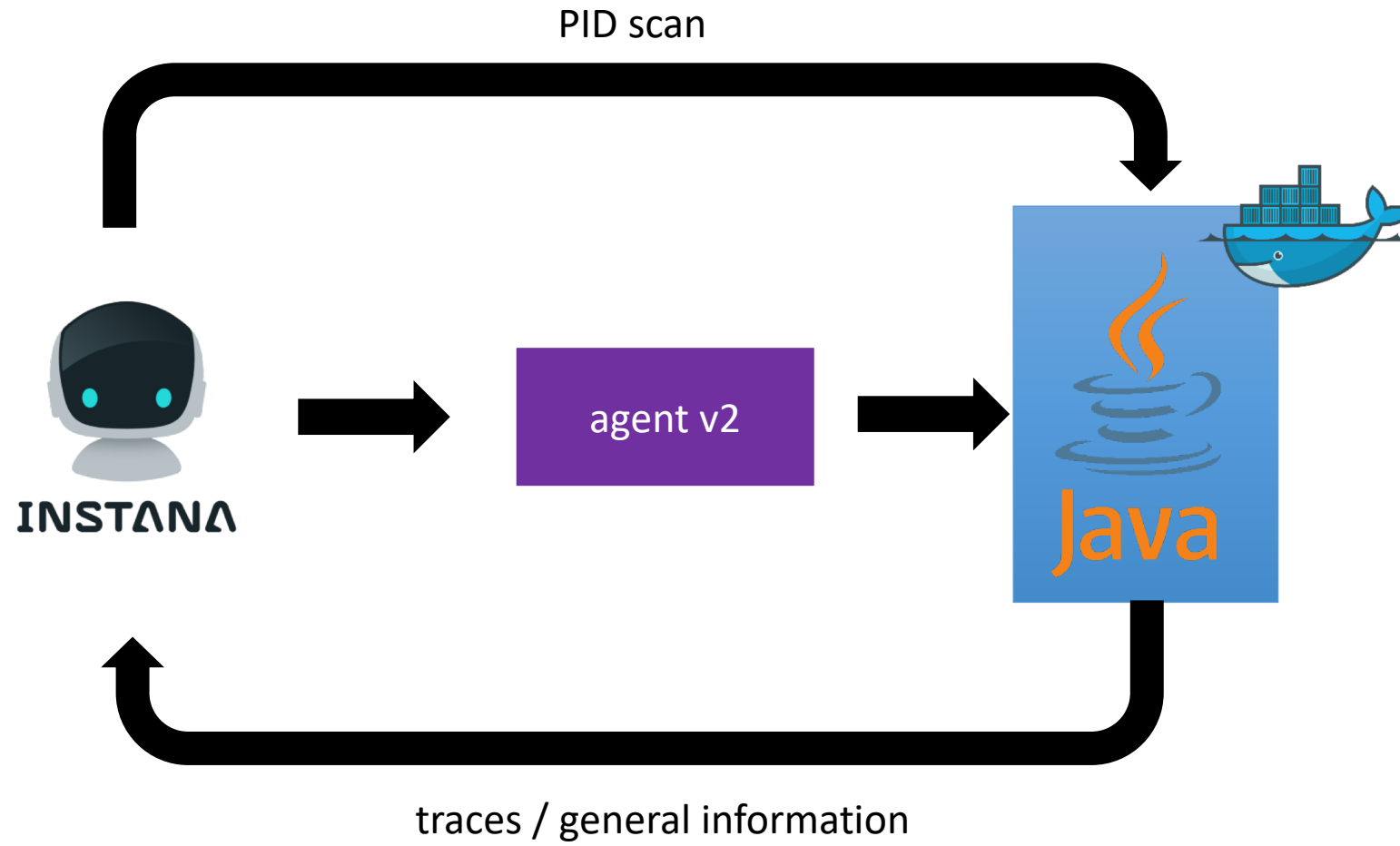
```
ca fe ba be 00 00 00  
34 00 10 0a 00 04 00  
0d 08 00 09 07 00 0e  
07 00 0f 01 00 06 3c  
69 6e 69 74 3e 01 00  
03 28 29 56 01 00 04  
43 6f 64 65 01 00 0f  
4c 69 6e 65 4e 75 6d  
62 65 72 54 61 62 6c  
65 01 00 03 62 61 72  
01 00 14 28 29 4c 6a  
61 76 61 ...
```

byte[]

Example application: Mockito inline-mock maker



Example application: APM



agent fundamentals

code instrumentation

class loading

miscellaneous

Transforming classes using ASM

```
static byte[] transform(byte[] classFile) {
    ClassReader reader = new ClassReader(classFile);
    ClassWriter writer = new ClassWriter(0);
    reader.accept(new ClassVisitor(ASM_7, writer) {
        @Override public MethodVisitor visitMethod(
            int access, String name, String signature, String[] exceptions) {
            MethodVisitor sv = super.visitMethod(access, name, signature, exceptions, 0);
            super.visitMethodInsn(ASM_7, INVOKESTATIC, name, signature, exceptions, true);
            super.visitCode();
        }, 0);
    visitFieldInsn(GET_STATIC, "java/lang/System", "out", "Ljava/lang/String;");
    return writer.toByteArray("Hello World");
    visitLdc("Hello World");
    visitMethodInsn(INVOKEINTERFACE, "Ljava/lang/PrintStream;", "println", "(Ljava/lang/String;)V", true);
};
}
```

Matching classes using ASM

```
static boolean shouldTransform(String className, byte[] classFile, ClassLoader cl) {
    return "my/target/UserType".equals(className) ? return true;
}

String nextType = new String[1];
while (true) {
    ClassReader reader = new ClassReader(new FileInputStream(ASM_7) {
        boolean hasSuperType = new boolean[1];
        public void accept(ClassVisitor cv, int access, String name, String signature,
            @Override String superName, String[] interfaces) {
            public void visit(int version, int access, String name, String signature,
                String superName, String[] interfaces) {
            }, hasSuperType[0] = "my/target/UserType".equals(superName);
            if (nextType[0] == null) return false;
        }, else if ("my/target/UserType".equals(className)) return true;
        return hasSuperType[0];
    });
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    cl.getResourceAsStream(nextType[0] + ".class").writeTo(bos);
    classFile = bos.toByteArray();
}

} throw new IllegalStateException();
}
```

Transforming and matching classes using Byte Buddy

```
class PreEntry {  
    public static void premain(String arg, Instrumentation inst) {  
        new AgentBuilder.Default()  
            .type(hasSuperType(named("my.target.UserType")))  
            .transform((builder, type, classLoader, module) -> builder  
                .method(any())  
                .intercept(MethodCall.invoke(  
                    PrintStream.class.getMethod("println", String.class))  
                .onField(System.class.getField("out"))  
                .andThen(SuperMethodCall.INSTANCE)))  
            .installOn(inst);  
    }  
}
```

How is a transformation applied by Byte Buddy?

```
class UserType {  
  
    String foo() {  
        return "foo"  
    }  
  
}
```

```
class UserType {  
  
    String foo() {  
        System.out.println("Hello World!");  
        return foo$original();  
    }  
  
    String foo$original() {  
        return "foo";  
    }  
  
    public String toString() {  
        System.out.println("Hello World!");  
        return super.toString();  
    }  
  
}
```

Decorating and retransforming classes using Byte Buddy

```
class AgentEntry {
    public static void agentmain(String arg, Instrumentation inst) {
        new AgentBuilder.Default()
            .typeMatcher(new TypeChange("my.target.UserType"))
            .withRenamer(new Renamer("RETRANSFORM"))
            .typeTransformer(new TypeTransformer() {
                @Override public Type transform(Type type, ClassLoader classLoader, Module module) {
                    return type;
                }
            })
            .visit(Advice.to(HelloWorldAdvice.class).on(isMethod()))
            .installOn(inst);
    }
}
```

```
class HelloWorldAdvice {
    @Advice.OnMethodEnter
    static void onEnter() {
        System.out.println("Hello world!");
    }
}
```

```
class UserType {
    String foo() {
        System.out.println("Hello world!");
        return "foo";
    }
}
```



Advice: exit instrumentation

```
class HelloWorldAdvice {  
    @Advice.OnMethodEnter  
    static void onEnter() {  
        System.out.println("Method enter");  
    }  
    @Advice.OnMethodExit  
    static void onExit() {  
        System.out.println("Method exit");  
    }  
}
```

```
class UserType {  
    String foo() {  
        System.out.println("Method enter");  
    }  
    String $return = "foo";  
} } System.out.println("Method exit");  
} return $return;  
}
```

Advice: multiple returns

```
class HelloWorldAdvice {
    @Advice.OnMethodEnter
    static void onEnter() {
        System.out.println("Method enter");
    }
    @Advice.OnMethodExit
    static void onExit() {
        System.out.println("Method exit");
    }
}
```

```
class UserType {
    String foo() {
        if (System.nanoTime() % 2 == 0) {
            System.out.println("Method enter");
            return "bar";
        }
        if (System.nanoTime() % 2 == 0) {
            doSomething("bar");
            return "end";
        }
        doSomething();
        $return = "baz";
        end:
        System.out.println("Method exit");
        return $return;
    }
}
```

Advice: method arguments

```
class HelloWorldAdvice {  
    @Advice.OnMethodEnter  
    static void onEnter(  
        @Advice.Argument(0) String val) {  
        System.out.println("Val: " + val);  
    }  
}
```

```
class UserType {  
    String foo(String val) {  
        System.out.println("Val: " + val);  
    } return "foo" + val;  
} }
```

Advice: changing method arguments

```
class HelloWorldAdvice {  
    @Advice.OnMethodEnter  
    static void onEnter(  
        @Advice.Argument(  
            value = 0  
            readOnly = false) String val) {  
        val = val + "bar";  
    }  
}
```

```
class UserType {  
    String foo(String val) {  
        return val + "bar";  
    }  
}
```

Advice: changing the return value

```
class HelloWorldAdvice {  
    @Advice.OnMethodExit  
    static void onExit(  
        @Advice.Return(  
            readOnly = false) String ret) {  
        ret = "bar";  
    }  
}
```

```
class UserType {  
    String foo() {  
        String $return = "foo";  
    } $return = "bar";  
    return $return;  
}
```

Advice: method arguments under alteration

```
class HelloWorldAdvice {  
    @Advice.OnMethodEnter  
    static void onEnter(  
        @Advice.Argument(0) String val) {  
        System.out.println("Enter: " + val);  
    }  
    @Advice.OnMethodExit  
    static void onExit(  
        @Advice.Argument(0) String val) {  
        System.out.println("Exit: " + val);  
    }  
}
```

```
class UserType {  
    String foo(String val) {  
        System.out.println("Enter: " + val);  
    }  
    String $val = val;  
} } $val = "foo" + $val;  
} String $return = $val;  
System.out.println("Exit: " + val);  
return $return;  
}  
}
```

Advice: retaining method argument changes

```
class HelloWorldAdvice {
    @Advice.OnMethodEnter
    static void onEnter(
        @Advice.Argument(0) String val) {
        System.out.println("Enter: " + val);
    }
    @Advice.OnMethodExit(
        backupArguments = false)
    static void onExit(
        @Advice.Argument(0) String val) {
        System.out.println("Exit: " + val);
    }
}
```

```
class UserType {
    String foo(String val) {
        System.out.println("Enter: " + val);
        val = "foo" + val;
        String $return = val;
        System.out.println("Exit: " + val);
        return $return;
    }
}
```

Advice: exits on exceptions

```
class HelloWorldAdvice {
    @Advice.OnMethodEnter
    static void onEnter() {
        System.out.println("Method enter");
    }
    @Advice.OnMethodExit(
        onThrowable = IOException.class)
    static void onExit() {
        System.out.println("Method exit");
    }
}
```

```
class UserType {
    String foo() throws Exception {
        if (System.nanoTime() % 2 == 0) {
            throw new IOException();
        } else {
            try {
                return "qux";
            } if (System.nanoTime() % 2 == 0) {
                throw new IOException();
            } else {
                $return = "qux";
            }
        } catch (IOException e) {
            $exception = e;
        }
        System.out.println("Method exit");
        if ($exception == null) {
            return $return;
        } else throw $exception;
    }
}
```

Advice: handling exceptions

```
class HelloWorldAdvice {
    @Advice.OnMethodExit(
        onThrowable = IOException.class)
    static void onExit(
        @Advice.Return(
            readOnly = false) String ret
        @Advice.Thrown(
            readOnly = false) Throwable t) {
        ret = "bar";
        t = null;
    }
}
```

```
class UserType {
    String foo() throws Exception {
        String $return = null;
        IOException $exception = null;
        try {
            if (System.nanoTime() % 2 == 0) {
                throw new IOException();
            } else {
                $return = "qux";
            }
        } catch (IOException e) {
            $exception = e;
        }
        $return = "bar";
        $exception = null;
        if ($exception == null) {
            return $return;
        } else throw $exception;
    }
}
```

Advice: skipping a method

```
class HelloWorldAdvice {
    @Advice.OnMethodEnter(
        skipOn = String.class)
    static String onEnter() {
        if (System.nanoTime() % 2 == 0) {
            return "bar";
        } else {
            return null;
        }
    }
}
```

```
class UserType {
    String foo() {
        doSomething();
        if (System.nanoTime() % 2 == 0) {
            $enter = "bar";
        } else {
            $enter = null;
        }
        String $return;
        if ($enter == null) {
            doSomething();
            $return = "foo";
        } else {
            $return = null;
        }
        return $return;
    }
}
```

Advice: skipping a method and determining its return value

```
class HelloWorldAdvice {
    @Advice.OnMethodEnter(
        skipOn = String.class)
    static String onEnter() {
        if (System.nanoTime() % 2 == 0) {
            return "bar";
        } else {
            return null;
        }
    }
    @Advice.OnMethodExit
    static void onExit(
        @Advice.Enter String enter
        @Advice.Return(
            readOnly = false) String ret) {
        ret = enter;
    }
}
```

```
class UserType {
    String foo() {
        String $enter;
        if (System.nanoTime() % 2 == 0) {
            $enter = "bar";
        } else {
            $enter = null;
        }
        String $return;
        if ($enter == null) {
            doSomething();
            $return = "foo";
        } else {
            $return = null;
        }
        $return = $enter;
        return $return;
    }
}
```

Advice: repeating a method

```
class HelloWorldAdvice {
    @Advice.OnMethodEnter
    static int onEnter(
        @Advice.Exit (
            readOnly = false) int exit) {
        exit = 3
    }
    @Advice.OnMethodExit(
        onThrowable = IOException.class
        repeatOn =
            Advice.OnNonDefaultValue.class)
    static int onExit(
        @Advice.Exit (
            readOnly = false) int exit) {
        exit--;
    }
}
```

```
class UserType {
    String foo() {
        if (System.nanoTime() % 2 == 0) {
            throw new IOException();
        } else {
            return "qux";
        }
    }
    try {
        if (System.nanoTime() % 2 == 0) {
            throw new IOException();
        } else {
            return "qux";
        }
    } catch (IOException e) {
        $exception = e;
    }
    $exit--;
    while ($exit != 0);
    if ($exception == null) {
        return $return;
    } else {
        throw $exception;
    }
}
```

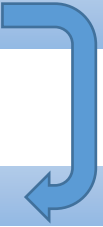
Advice: stack allocation

```
class HelloWorldAdvice {
    @Advice.OnMethodEnter
    static int onEnter(
        @Advice.Local("val") String val) {
        val = "Hello!";
    }
    @Advice.OnMethodExit
    static int onExit(
        @Advice.Local("val") String val) {
        System.out.println(val);
    }
}
```

```
class UserType {
    String foo() {
        String $val = null;
        } $val = "Hello!";
    } String $return = "foo";
    System.out.println($val);
    return $return;
}
```

Stack map frames

```
static void foo(String val) {  
    if (val == null) {  
        return;  
    }  
    System.out.println(val);  
}
```



```
0: ALOAD 0  
1: IFNONNULL 3  
2: RETURN  
3: GETSTATIC java/lang/System.out  
4: ALOAD 0  
5: INVOKEVIRTUAL java/io/PrintStream.println  
6: RETURN
```

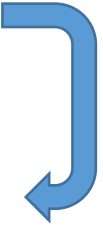


Stack map frames: (since Java 6)

```
3: [String]
```

Computing stack map frames in ASM

```
static void foo(String val) {  
    if (val == null) {  
        return;  
    }  
    System.out.println(val);  
}
```



```
Class<?> loaded1 = ClassWriter.class  
    .getClassLoader()  
    .loadClass(type1.replace("/", "."));
```

```
ClassWriter classWriter = new ClassWriter(COMPUTE_FRAMES);
```

```
MethodVisitor methodVisitor = ...
```

```
Label label = new Label();
```

```
methodVisitor.visitVarInsn(ALOAD, 0);
```

```
methodVisitor.visitJumpInsn(IFNONNULL, label);
```

```
methodVisitor.visitInsn(RETURN);
```

```
methodVisitor.visitLabel(label);
```

```
methodVisitor.visitStackMapFrame(F_SAME, ...);
```

```
methodVisitor.visitFieldInsn(GETSTATIC, "java/lang/System", "out", ...);
```

```
methodVisitor.visitVarInsn(ALOAD, 0);
```

```
methodVisitor.visitMethodInsn(INVOKEVIRTUAL, "java/io/PrintStream", "println", ...);
```

```
methodVisitor.visitInsn(RETURN);
```



Fusing stack map frames in Byte Buddy advice

```
@Advice.OnMethodEnter
static void onEnter(
    @Advice.Argument(0) String val) {
    String copy = val;
    if (false) return;
}

static void foo(String val, Object other) {
    Object copy = other;
    if (false) return;
}
```

[String]Object]

0: ALOAD 0

1: ASTORE 2

2: ICONST_0

3: IFNE 5

4: RETURN

[String, Object]String]

5: RETURN

[String, Object]

[String, Object]

6: ALOAD 1

7: ASTORE 2

8: ICONST_0

9: IFNE 5

10: RETURN

[String, Object, Object]

15: RETURN

Limitations of advice templates

```
class FaultyAdvice {  
  
    static String value;  
  
    @Advice.OnMethodEnter  
    static void onEnter(@MyConstantValue String value) {  
        helper(value);  
    }  
    // #BB-613: since Java 8, a copy would be possible  
    private static void helper(String value) {  
        // ...  
    }  
}
```

In-method code substitution

```
class UserType {  
    void foo() {  
        int random = ThreadLocalRandom.current().nextInt();  
        System.out.println(random);  
    }  
}
```

```
class PreEntry {  
    public static void premain(String arg, Instrumentation inst) {  
        new AgentBuilder.Default()  
            .type(hasSuperType(named("my.target.UserType")))  
            .transform((builder, type, classLoader, module) -> builder  
                .visit(MemberSubstitution.strict()  
                    .method(named("println"))  
                    .replaceWithMethod(named("print").and(takesArguments(String.class))  
                        .on(named("foo"))))  
            .installOn(inst);  
    }  
}
```

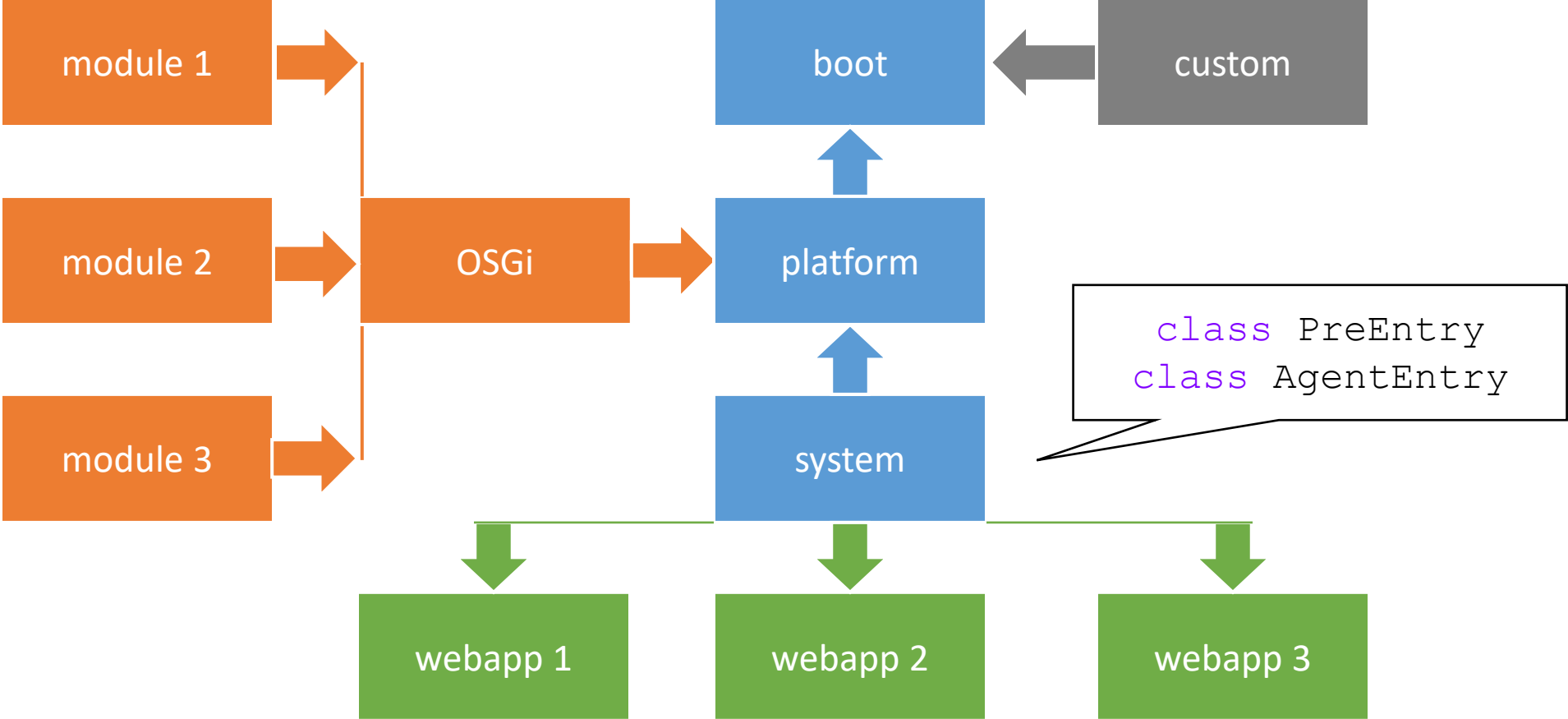
agent fundamentals

code instrumentation

class loading

miscellaneous

Class loader hierarchies



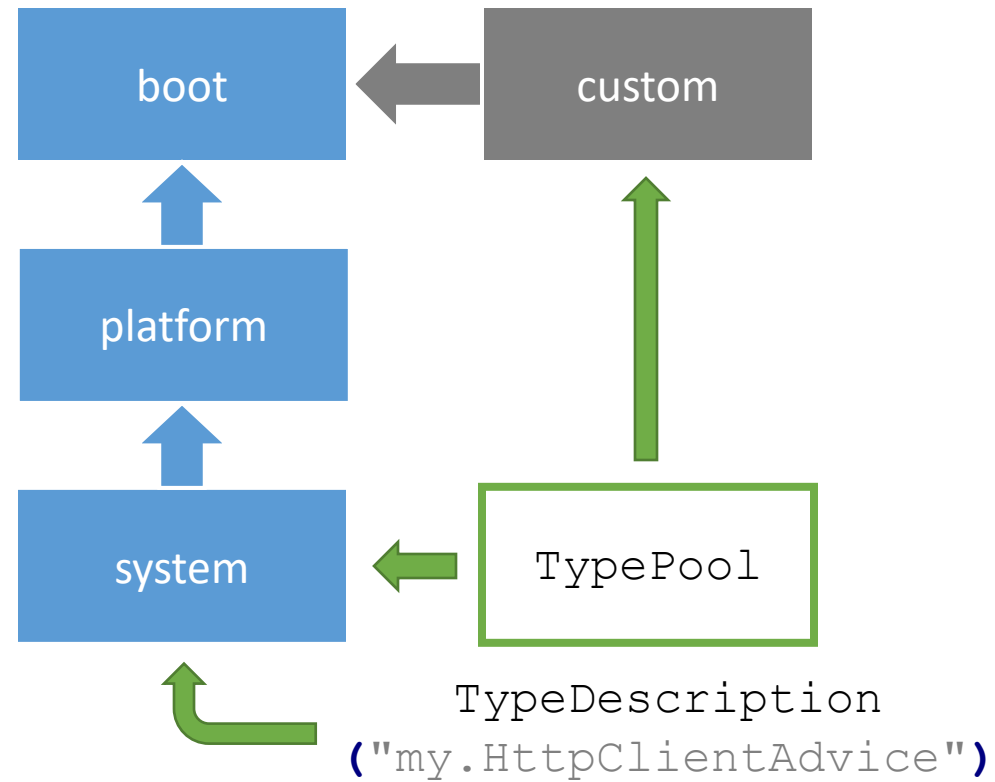
Supporting unknown class loader hierarchies

```
class HttpClientAdvice {  
    @Advice.OnMethodEnter  
    static long onEnter() {  
        return System.currentTimeMillis();  
    }  
    @Advice.OnMethodExit  
    static void onExit(  
        @Advice.Argument(0) HttpRequest request,  
        @Advice.Return HttpResponse response,  
        @Advice.Enter long time) {  
        Tracing.record(request.getUrl(),  
            response.getStatus(),  
            System.currentTimeMillis() - time);  
    }  
}
```

Supporting unknown class loader hierarchies

```
class PreEntry {
    public static void premain(String arg, Instrumentation inst) {
        new AgentBuilder.Default()
            .type(named("some.HttpClient"))
            .transform((builder, type, classLoader, module) -> builder
                .visit(new AgentBuilder.Default.ForAdvice().named("send")))
            .installOn(inst)
            .installOn(PreEntry.class.getClassLoader())
            .advice(named("send"), "my.HttpClientAdvice"))
    }
    .installOn(inst);
}
}
```

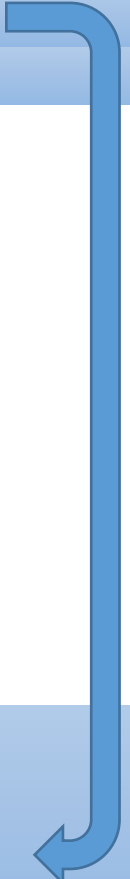
Supporting unknown class loader hierarchies



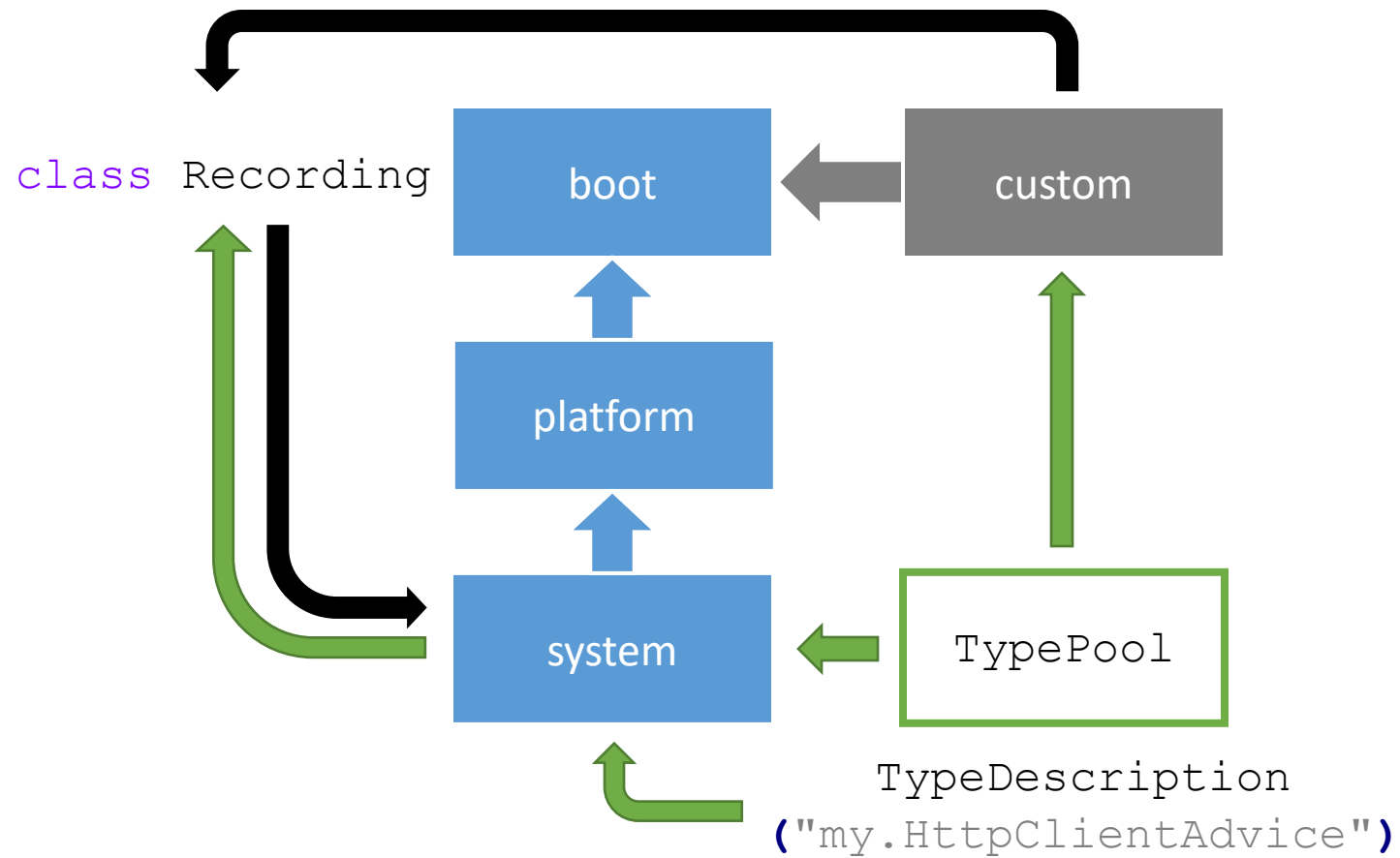
Supporting unknown class loader hierarchies

```
class PreEntry {
    public static void premain(String arg, Instrumentation inst) {
        inst.appendToBootstrapClassLoaderSearch(new JarFile("trace.jar"));
        Recording.dispatcher = initializeRecording();
        new AgentBuilder.Default()
            .type(named("some.HttpClient"))
            .transform((builder, type, classLoader, module) -> builder
                .visit(new AgentBuilder.Transformer.ForAdvice()
                    .include(PreEntry.class.getClassLoader())
                    .advice(named("send"), "my.HttpClientAdvice")))
            .installOn(inst);
    }
}

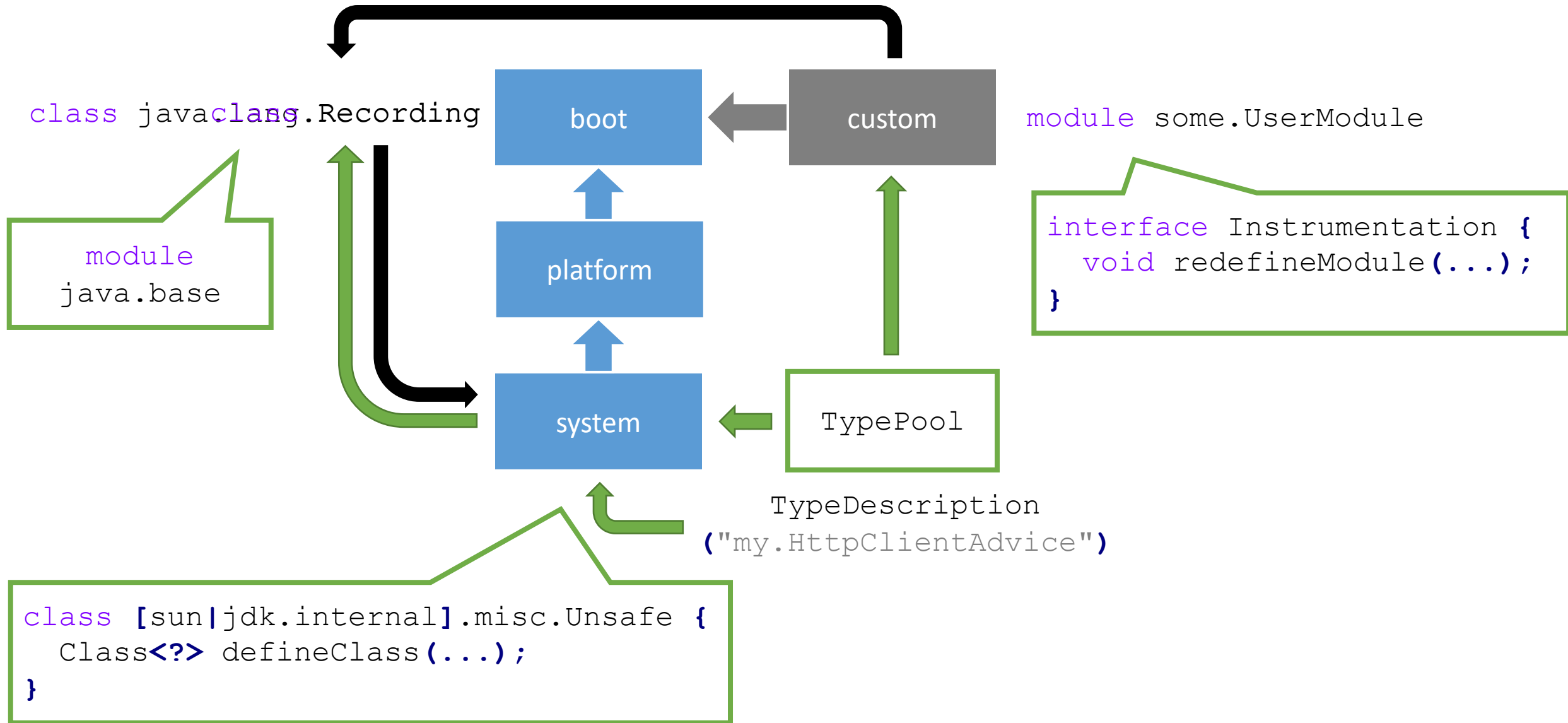
public abstract class Recording {
    public static Recording dispatcher;
    public static void record(String url, int status, long time) {
        tracing.doRecord(url, status, time);
    }
    protected abstract void doRecord(String url, int status, long time);
}
```



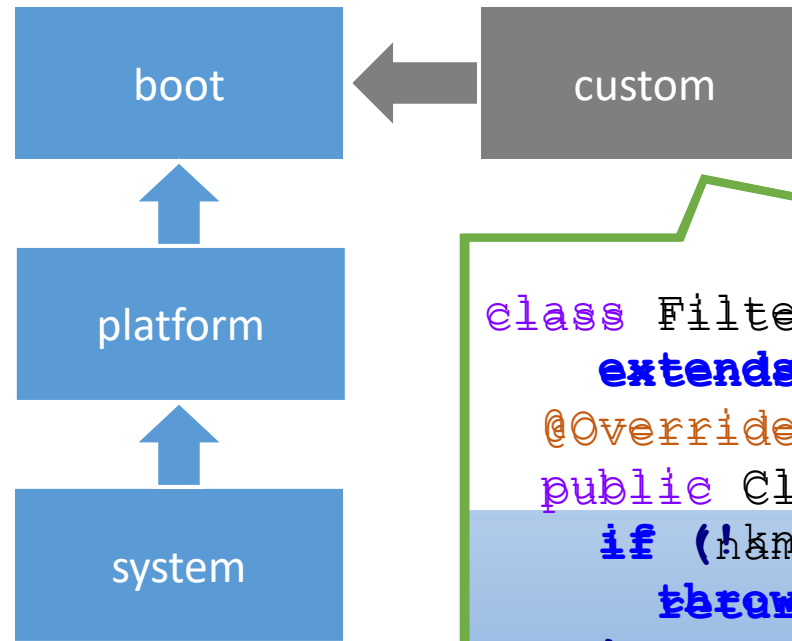
Supporting unknown class loader hierarchies



Supporting unknown module hierarchies



Supporting class loaders with explicit filters

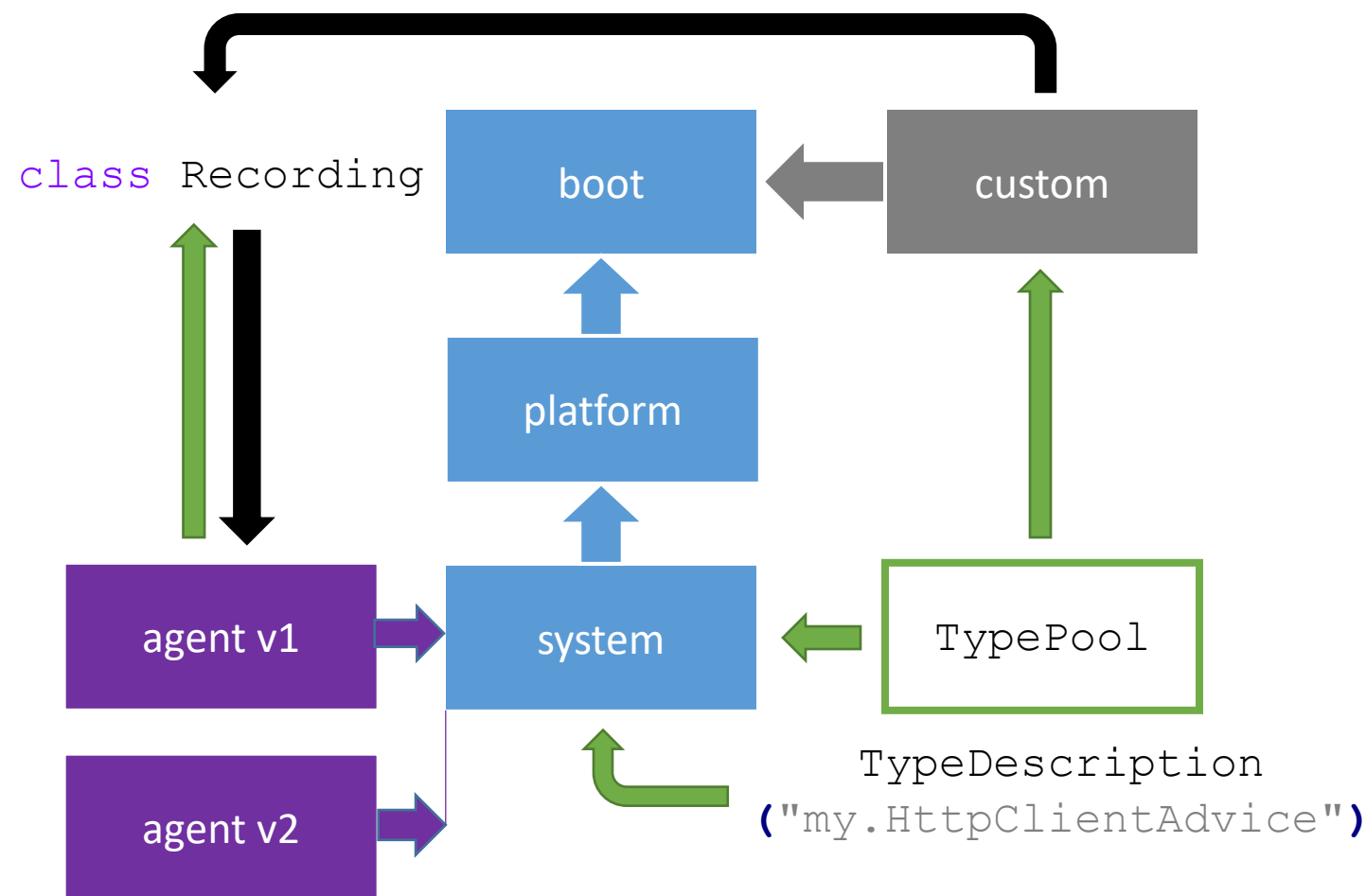


```
class FilteringClassLoader
    extends ClassLoader {
    @Override
    public Class<?> loadClass(String name) {
        if (!knownClasses.containsKey(named)) {
            throw new IllegalStateException(name);
        }
        return false, null);
    }
    if (!knownClasses.containsKey(named)) {
    }
    throw new IllegalStateException();
}
}
```

Making agents updatable

```
class AgentEntry {  
    static ClassLoader previous;  
    public static void agentmain(String arg, Instrumentation inst)  
        throws Exception {  
        ClassLoader loader = new URLClassLoader(new URL[] {arg});  
        loader.loadClass("agent.EntryPoint")  
            .getMethod("init", Instrumentation.class, ClassLoader.class)  
            .invoke(inst, previous);  
        previous = loader;  
    }  
}
```

Making agents updatable



agent fundamentals

code instrumentation

class loading

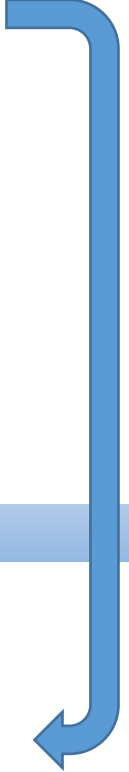
miscellaneous

Self-attaching for testing a Java agent

```
long pid = ProcessHandle.current().pid();
VirtualMachine vm = VirtualMachine.attach(String.valueOf(pid));
try {
    VirtualMachine vm = VirtualMachine.attach(String.valueOf(pid));
    try {
        loadAgent(MyAttachHelper.class.getProtectionDomain()
            .getCodeSource().getLocation().toURI(), null);
    } finally {
        String str = String.valueOf(pid);
        vm.detach();
    }
} finally {
    vm.detach();
}

Instrumentation inst = MyAttachHelper.inst;

class MyAttachHelper {
    Instrumentation inst;
    public static void agentmain(String arg, Instrumentation inst) {
        MyAttachHelper.inst = inst;
    }
}
```



Attachment using Byte Buddy

```
Instrumentation inst = ByteBuddyAgent.install();  
ByteBuddyAgent.attach(new File("myagent.jar"), "123");
```

`interface` AttachmentProvider

1. jdk.attach (JDK only) / IBM-namespace-aware
2. jdk.attach via helper process (JDK only) / IBM-namespace-aware
3. Unix socket emulation (Unix only)
4. Unix socket emulation via helper process (Unix only)

Instance-bound state without adding fields

```
public class StateHandler { // injected into boot loader
    public static final Map<Object, Object> state =
        new WeakConcurrentMap<>() {
            {
                put(new WeakHashMap<>());
            }
        }
}
```

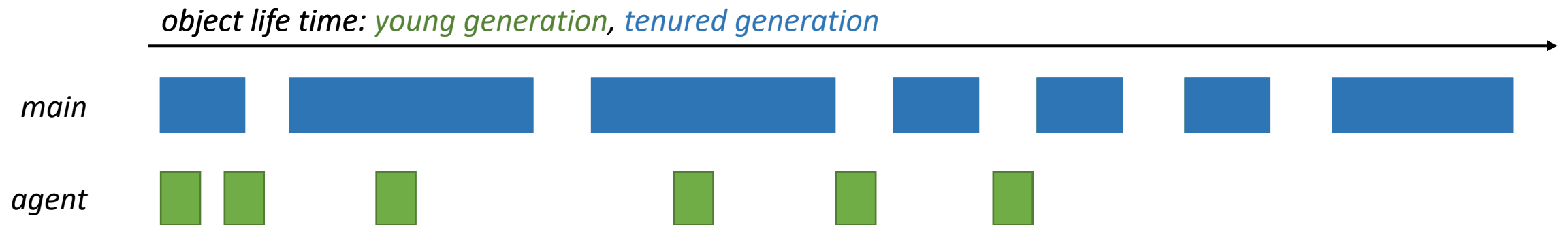
```
class UserClass {
    void foo() { dispatcher dispatcher;
    static {
        } Class.forName(
    } "net.bytebuddy.Nexus", false, ClassLoader.getSystemClassLoader())
        .getMethod("init", Class.class)
        .invoke(null, UserClass.class);
    }
    void foo() {
        dispatcher.run();
        // ...
    }
}
```

Increasing retransformation resilience

```
class AgentEntry {
    public static void agentmain(String arg, Instrumentation inst) {
        new AgentBuilder.Default()
            .disableClassFormatChanges()
            .with(RedefinitionStrategy.RETRANSFORM)
            .with(BatchType.BatchType.Partitioning.of(4))
            .with(DiscoveryStrategy.Repeating.INSTANCE) -> builder
            .with(new SubclassAppender() {
                @Override public void visit(ClassLoader loader, Class type) {
                    visit(loader, type, classLoader, module) -> builder
                }
            }) -> builder
            .installOn(inst, SomeAdvice.class, on(isMethod()))
            .visit(Advice.to(SomeAdvice.class).on(isMethod()))
            .installOn(builder, type, classLoader, module) -> builder
            .visit(Advice.to(SomeAdvice.class).on(isMethod()))
            .installOn(inst);
    }
}
```

Balancing performance: JIT and GC inference

```
-XX:MaxInlineLevel // maximum nested calls
-XX:MaxInlineSize // maximum byte code size
-XX:FreqInlineSize // maximum byte code size (frequently called)
-XX:MaxTrivialSize // maximum byte code size (trivial, e.g. getter/setter)
-XX:MinInliningThreshold // minimum amount of calls before inlining
-XX:LiveNodeCountInliningCutoff // max number of live nodes
```



<http://rafael.codes>
[@rafaelcodes](#)



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



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

