THE HIDDEN DYNAMIC LIFE OF JAVA

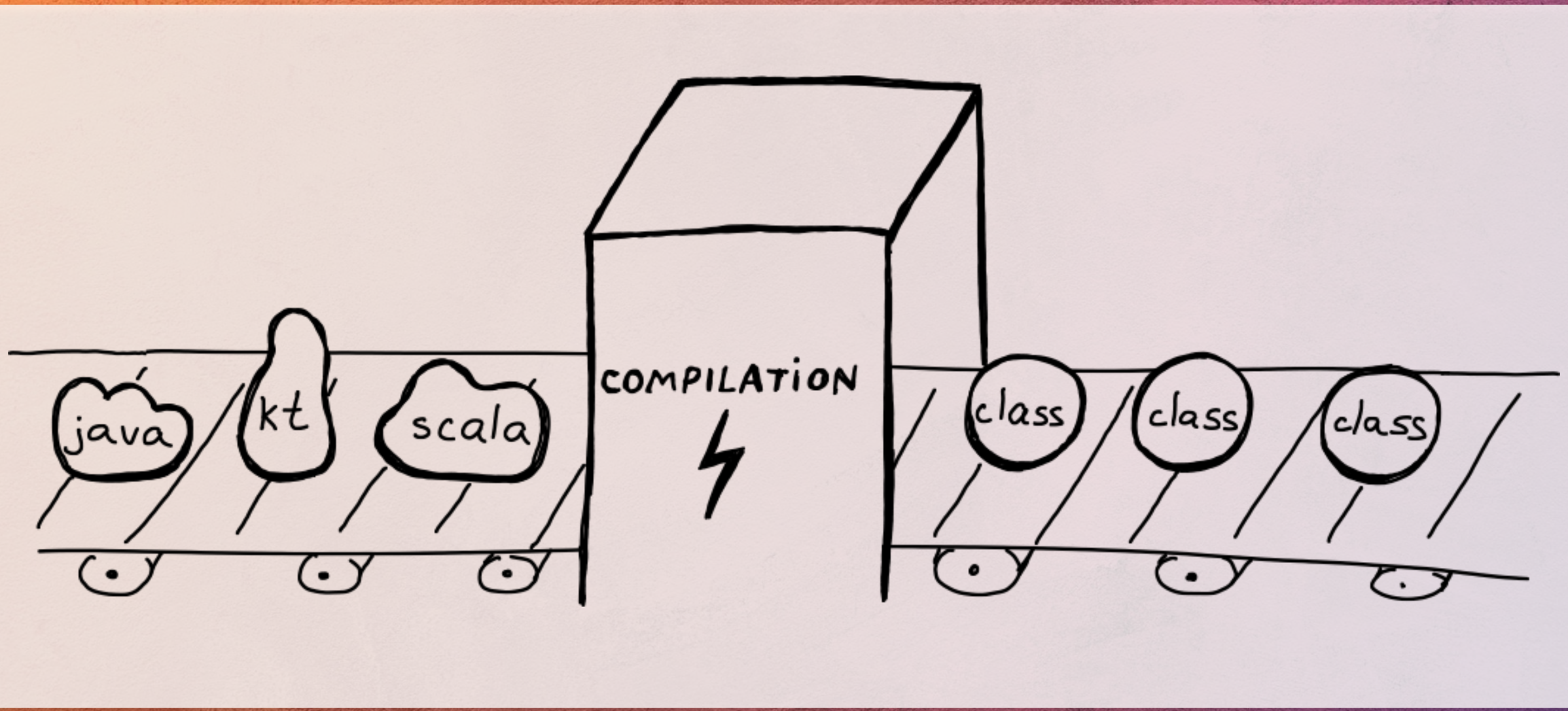
NATALIIA DZIUBENKO

SOFTWARE ENGINEER @XEBIA

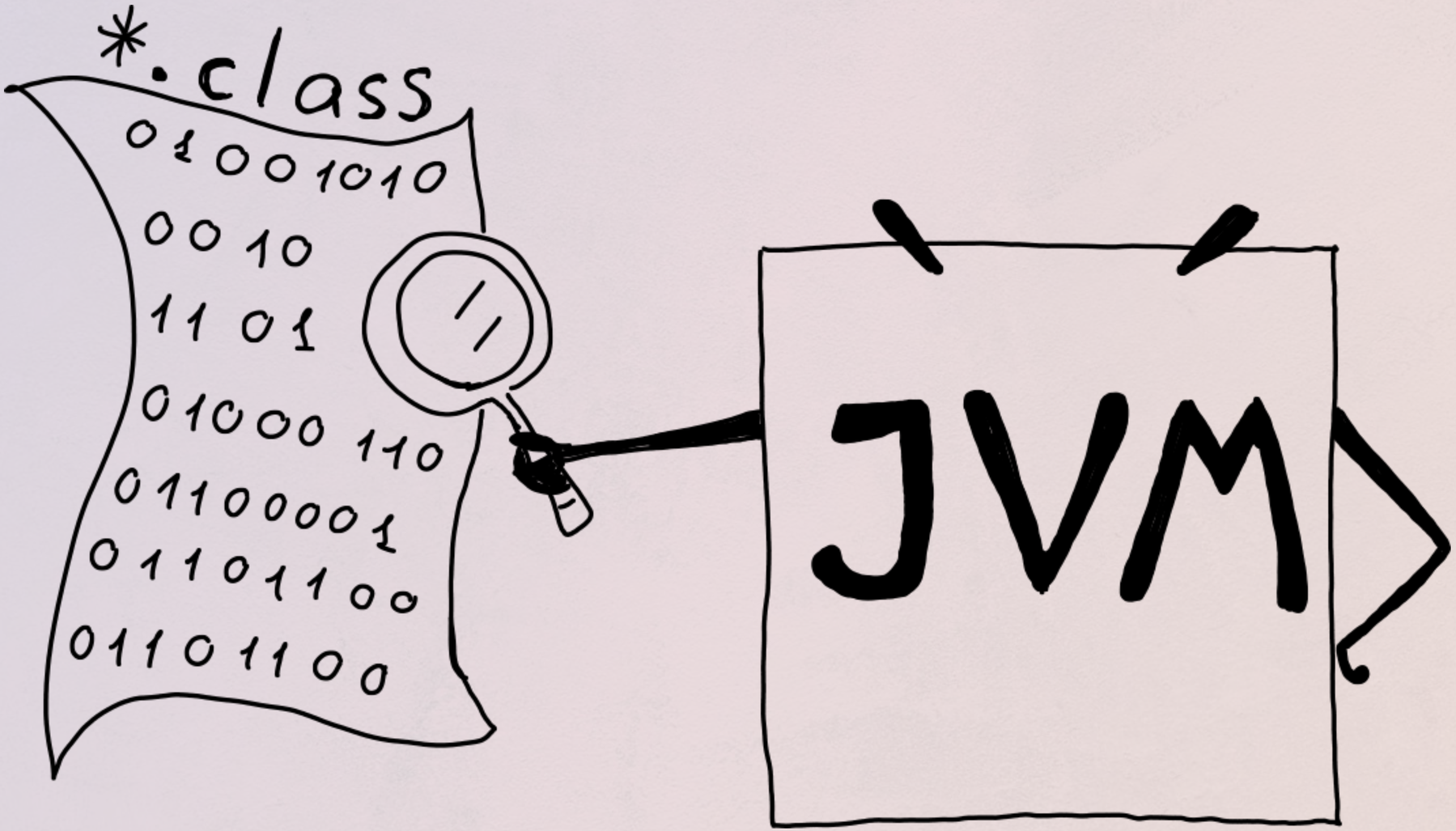
@WORTH_EXPLORING

but first...

Convenience of the standard Bytecode format



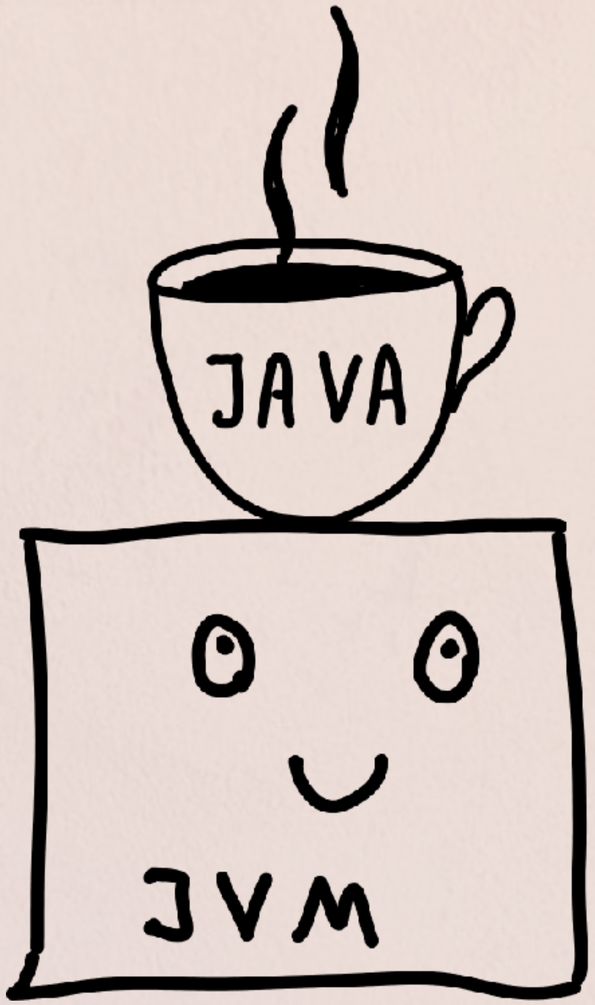
Convenience of the standard Bytecode format



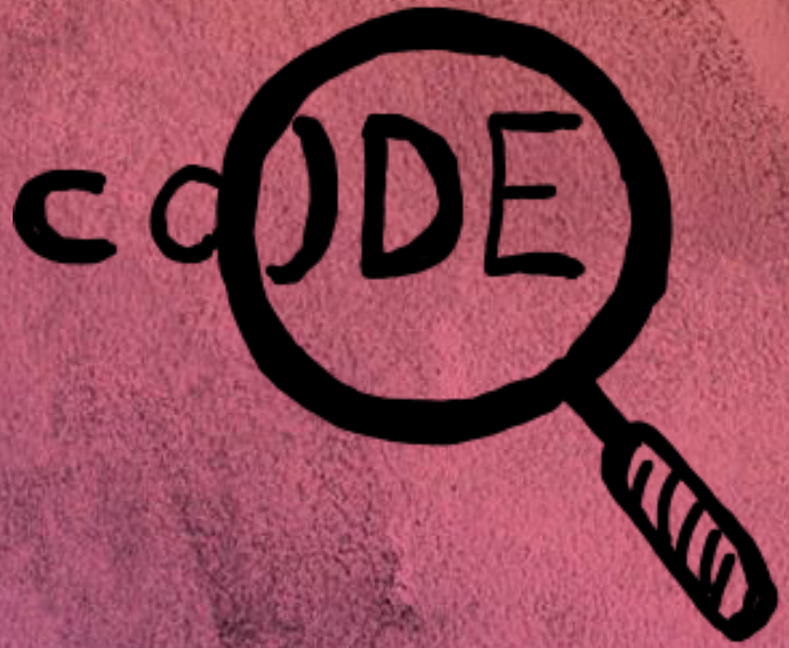
SMART,

right?

IT WAS ALL FOR JAVA !



Statically vs Dynamically Typed Languages



COMPILATION

STATICALLY VS DYNAMICALLY TYPED LANGUAGES

COMPILATION

RUNTIME

STATICALLY VS DYNAMICALLY TYPED LANGUAGES

CODE

COMPILATION

STATICALLY VS DYNAMICALLY TYPED LANGUAGES

CODE

COMPILATION

RUNTIME

DYNAMICALLY TYPED LANGUAGES ON JVM

Jython

FOR EXAMPLE

```
def absolute(num):
    return num.__abs__()



print(absolute(-1))
print(absolute("a"))
```

POSSIBLE WORKAROUND

LANGUAGE-SPECIFIC TYPES

```
public static SuperType abs(SuperType num) {
    return num.abs();
}

public static void main(String[] args) {
    SuperType result1 = abs(SuperType.of(-2));
    SuperType result2 = abs(SuperType.of("a"));
}
```

OTHER WORKAROUNDS

- REFLECTION
- EXTRA INTERPRETER

THE NEW WAY

INVOKEDYNAMIC

How is indy different?

invokevirtual

```
0: getstatic     #7                  // Field java/lang/System.out:Ljava/io/PrintStream;
3: ldc           #13                 // String hello
5: invokevirtual #15                 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
```

How is indy different?

invokestatic

```
0: bipush        115
2: invokestatic  #7                  // Method java/lang/String.valueOf:(I)Ljava/lang/String;
```

How is indy different?

invokedynamic

```
3: iload_1
4: invokedynamic #7,  0              // InvokeDynamic #0:makeConcatWithConstants:(I)Ljava/lang/String;
```

How is indy different?

invokedynamic

```
3: iload_1
4: invokedynamic #7,  0              // InvokeDynamic #0:makeConcatWithConstants:(I)Ljava/lang/String;
```

```
BootstrapMethods:
  0: #31 REF_invokeStatic java/lang/invoke/StringConcatFactory.makeConcatWithConstants:
(Ljava/lang/invoke/MethodHandles$Lookup;Ljava/lang/String;Ljava/lang/invoke/MethodType;
Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/invoke/CallSite;
    Method arguments:
      #29 It\'s \u0001 years!
```

How is indy different?

invokedynamic

```
3: iload_1
4: invokedynamic #7,  0              // InvokeDynamic #0:makeConcatWithConstants:(I)Ljava/lang/String;
```

```
BootstrapMethods:
  0: #31 REF_invokeStatic java/lang/invoke/StringConcatFactory.makeConcatWithConstants:
(Ljava/lang/invoke/MethodHandles$Lookup;Ljava/lang/String;Ljava/lang/invoke/MethodType;
Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/invoke/CallSite;
    Method arguments:
      #29 It\'s \u0001 years!
```

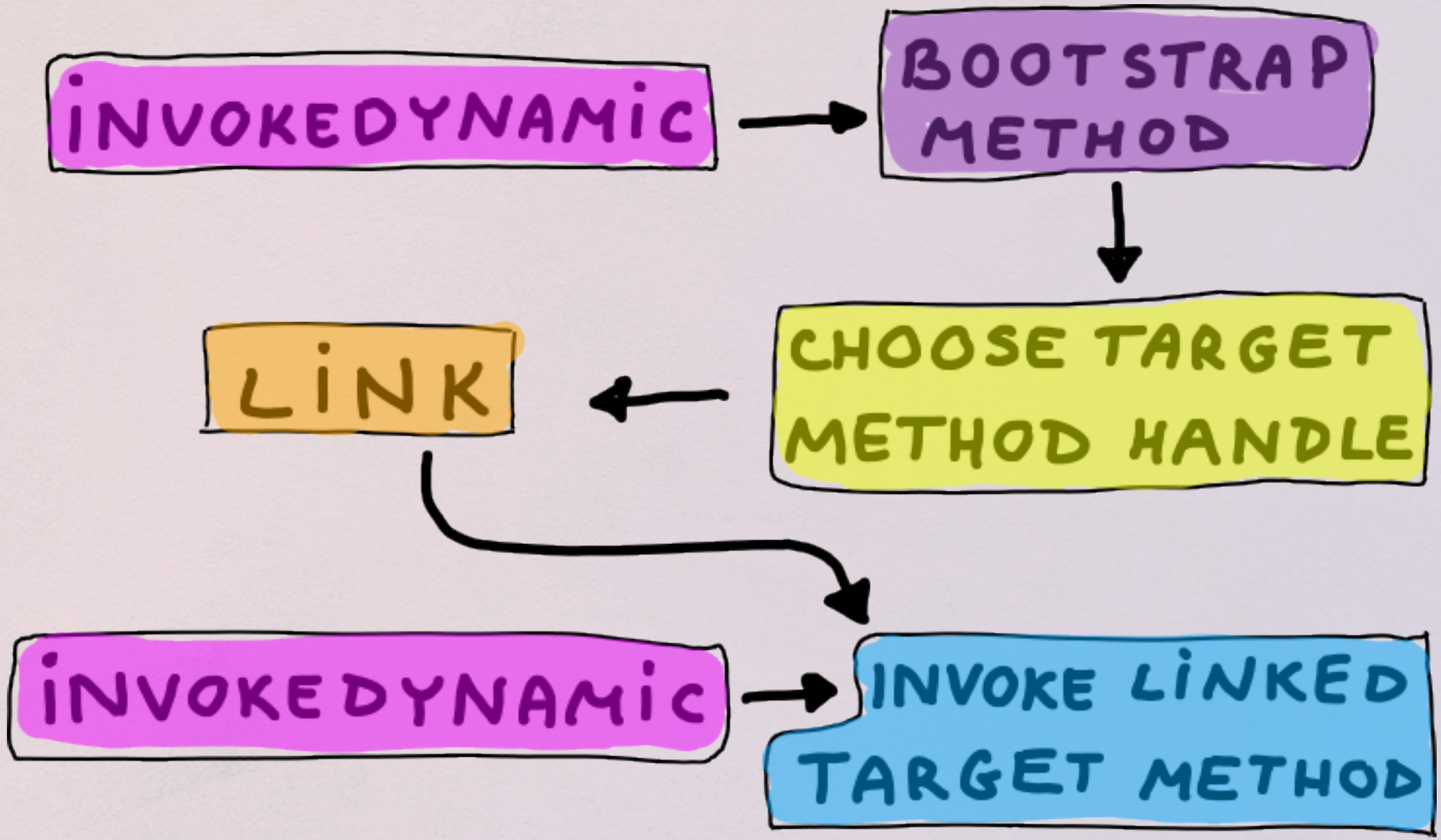
How is indy different?

invokedynamic

```
3: iload_1
4: invokedynamic #7,  0                  // InvokeDynamic #0:makeConcatWithConstants:(I)Ljava/lang/String;
```

```
BootstrapMethods:
  0: #31 REF_invokeStatic java/lang/invoke/StringConcatFactory.makeConcatWithConstants:
(Ljava/lang/invoke/MethodHandles$Lookup;Ljava/lang/String;Ljava/lang/invoke/MethodType;
Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/invoke/CallSite;
    Method arguments:
      #29 It\'s \u0001 years!
```

BOOTSTRAP METHOD



so

WHAT about JAVA?

STRING CONCATENATION

```
String a = "a";
String b = "b";
String test = a + b;
```

STRING CONCATENATION

```
String a = "a";
String b = "b";
String test = a + b;
```

```
        6: aload_1
        7: aload_2
        8: invokedynamic #11,  0              // InvokeDynamic #0:makeConcatWithConstants:
(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
```

STRING CONCATENATION

```
String a = "a";
String b = "b";
String test = a + b;
```

```
         6: aload_1
         7: aload_2
         8: invokedynamic #11,  0              // InvokeDynamic #0:makeConcatWithConstants:
(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
```

STRING CONCATENATION

```
String a = "a";
String b = "b";
String test = a + b;
```

```
         6: aload_1
         7: aload_2
         8: invokedynamic #11,  0              // InvokeDynamic #0:makeConcatWithConstants:
(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
```

```
BootstrapMethods:
  0: #43 REF_invokeStatic java/lang/invoke/StringConcatFactory.makeConcatWithConstants:
(Ljava/lang/invoke/MethodHandles$Lookup;Ljava/lang/String;Ljava/lang/invoke/MethodType;
Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/invoke/CallSite;
    Method arguments:
      #49 \u0001\u0001
```

STRING CONCATENATION

```
String a = "a";
String b = "b";
String test = a + b;
```

```
         6: aload_1
         7: aload_2
         8: invokedynamic #11,  0              // InvokeDynamic #0:makeConcatWithConstants:
(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
```

```
BootstrapMethods:
  0: #43 REF_invokeStatic java/lang/invoke/StringConcatFactory.makeConcatWithConstants:
(Ljava/lang/invoke/MethodHandles$Lookup;Ljava/lang/String;Ljava/lang/invoke/MethodType;
Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/invoke/CallSite;
    Method arguments:
      #49 \u0001\u0001
```

STRING CONCATENATION

```
    // Fast-path binary concatenations
    if (elements.size() == 2) {
        // Two arguments
        String s0 = elements.get(0);
        String s1 = elements.get(1);

        if (mt.parameterCount() == 2 &&
                !mt.parameterType( num: 0).isPrimitive() &&
                !mt.parameterType( num: 1).isPrimitive() &&
                s0 == null &&
                s1 == null) {
            return simpleConcat();
```

STRING CONCATENATION

```
    // Fast-path binary concatenations
    if (elements.size() == 2) {
        // Two arguments
        String s0 = elements.get(0);
        String s1 = elements.get(1);

        if (mt.parameterCount() == 2 &&
                !mt.parameterType( num: 0).isPrimitive() &&
                !mt.parameterType( num: 1).isPrimitive() &&
                s0 == null &&
                s1 == null) {
            return simpleConcat();
```

STRING CONCATENATION

```
    // Fast-path binary concatenations
    if (elements.size() == 2) {
        // Two arguments
        String s0 = elements.get(0);
        String s1 = elements.get(1);

        if (mt.parameterCount() == 2 &&
                !mt.parameterType( num: 0).isPrimitive() &&
                !mt.parameterType( num: 1).isPrimitive() &&
                s0 == null &&
                s1 == null) {
            return simpleConcat();
```

```
MethodHandle simpleConcat = JLA.stringConcatHelper( name: "simpleConcat",
        methodType(String.class, Object.class, Object.class));
```

STRING CONCATENATION

```
    // Fast-path binary concatenations
    if (elements.size() == 2) {
        // Two arguments
        String s0 = elements.get(0);
        String s1 = elements.get(1);

        if (mt.parameterCount() == 2 &&
                !mt.parameterType( num: 0).isPrimitive() &&
                !mt.parameterType( num: 1).isPrimitive() &&
                s0 == null &&
                s1 == null) {
            return simpleConcat();
```

```
MethodHandle simpleConcat = JLA.stringConcatHelper( name: "simpleConcat",
        methodType(String.class, Object.class, Object.class));
```

STRING CONCATENATION

```
// Fast-path binary concatenations
if (elements.size() == 2) {
    // Two arguments
    String s0 = elements.get(0);
    String s1 = elements.get(1);

    if (mt.parameterCount() == 2 &&
            !mt.parameterType( num: 0).isPrimitive() &&
            !mt.parameterType( num: 1).isPrimitive() &&
            s0 == null &&
            s1 == null) {
        return simpleConcat();
```

```
MethodHandle simpleConcat = JLA.stringConcatHelper( name: "simpleConcat",
        methodType(String.class, Object.class, Object.class));
```

```
static String simpleConcat(Object first, Object second)
```

PATTERN MATCHING FOR SWITCH

```java
public static void main(String[] args) {
    test( obj: 115);
    test( obj: "a");
}

2 usages
public static void test(Object obj) {
    switch (obj) {
        case String s                    -> System.out.println("string: " + s);
        case Integer num                 -> System.out.println("int to long: " + num.longValue());
        default                          -> System.out.println("default");
    }
}
```

PATTERN MATCHING FOR SWITCH

```
 9: aload_1
10: iload_2
11: invokedynamic #27,  0              // InvokeDynamic #0:typeSwitch:(Ljava/lang/Object;I)I
16: lookupswitch  { // 2
               0: 44
               1: 64
         default: 89
    }
```

PATTERN MATCHING FOR SWITCH

```
 9: aload_1
10: iload_2
11: invokedynamic #27,  0              // InvokeDynamic #0:typeSwitch:(Ljava/lang/Object;I)I
16: lookupswitch  { // 2
               0: 44
               1: 64
         default: 89
    }
```

```
BootstrapMethods:
  0: #81 REF_invokeStatic java/lang/runtime/SwitchBootstraps.typeSwitch:(Ljava/lang/invoke/MethodHandles$Lookup;
Ljava/lang/String;Ljava/lang/invoke/MethodType;[Ljava/lang/Object;)Ljava/lang/invoke/CallSite;
    Method arguments:
      #31 java/lang/String
      #8 java/lang/Integer
```

LAMBDAS

```
Stream.of( ...values: "a", "b").forEach(System.out::println);
```

LAMBDAS

```
Stream.of( ...values: "a", "b").forEach(System.out::println);
```

```
        25: invokedynamic #31,  0              // InvokeDynamic #0:accept:(Ljava/io/PrintStream;)
Ljava/util/function/Consumer;
        30: invokeinterface #35,  2            // InterfaceMethod java/util/stream/Stream.forEach
:(Ljava/util/function/Consumer;)V
```

LAMBDAS

```
Stream.of( ...values: "a", "b").forEach(System.out::println);
```

```
       25: invokedynamic #31,  0             // InvokeDynamic #0:accept:(Ljava/io/PrintStream;)
Ljava/util/function/Consumer;
       30: invokeinterface #35,  2           // InterfaceMethod java/util/stream/Stream.forEach
:(Ljava/util/function/Consumer;)V
```

```
BootstrapMethods:
  0: #63 REF_invokeStatic java/lang/invoke/LambdaMetafactory.metafactory:(Ljava/lang/invoke/MethodHandles$Look
up;Ljava/lang/String;Ljava/lang/invoke/MethodType;Ljava/lang/invoke/MethodType;Ljava/lang/invoke/MethodHandle;
Ljava/lang/invoke/MethodType;)Ljava/lang/invoke/CallSite;
    Method arguments:
      #53 (Ljava/lang/Object;)V
      #55 REF_invokeVirtual java/io/PrintStream.println:(Ljava/lang/String;)V
      #62 (Ljava/lang/String;)V
```

CONCLUSIONS

SPACE

COMPLEXITY

ABSTRACT AWAY DECISION MAKING

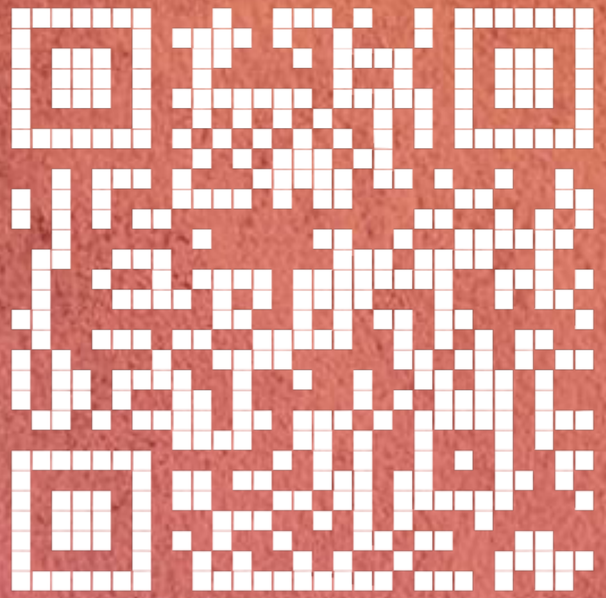
IMPROVED IMPLEMENTATION

NO NEED TO RECOMPILE

PREDICTABILITY

PERFORMANCE

SOME FUN READING



ORACLE

MEDIUM

@WORTH_EXPLORING

THANK YOU!