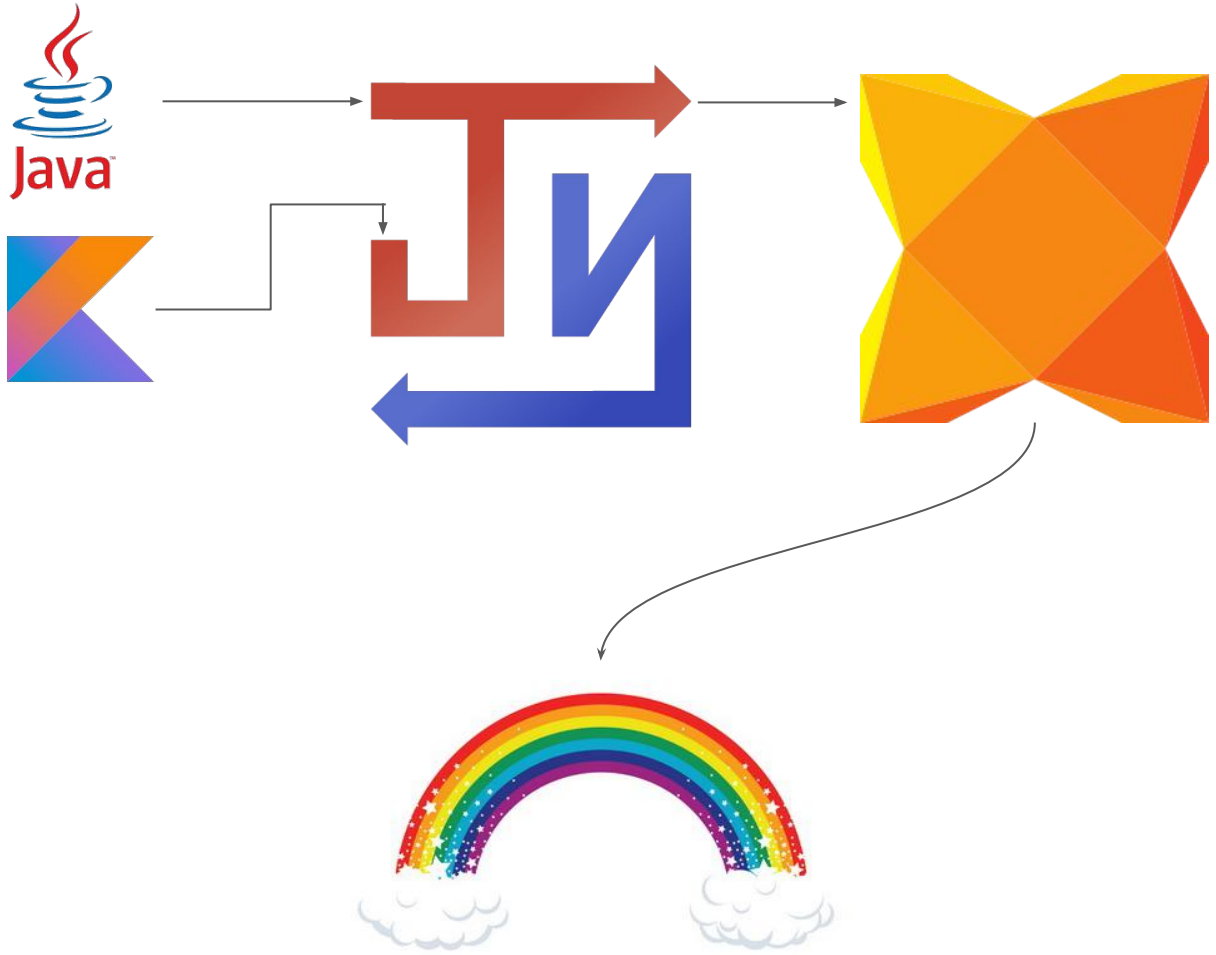
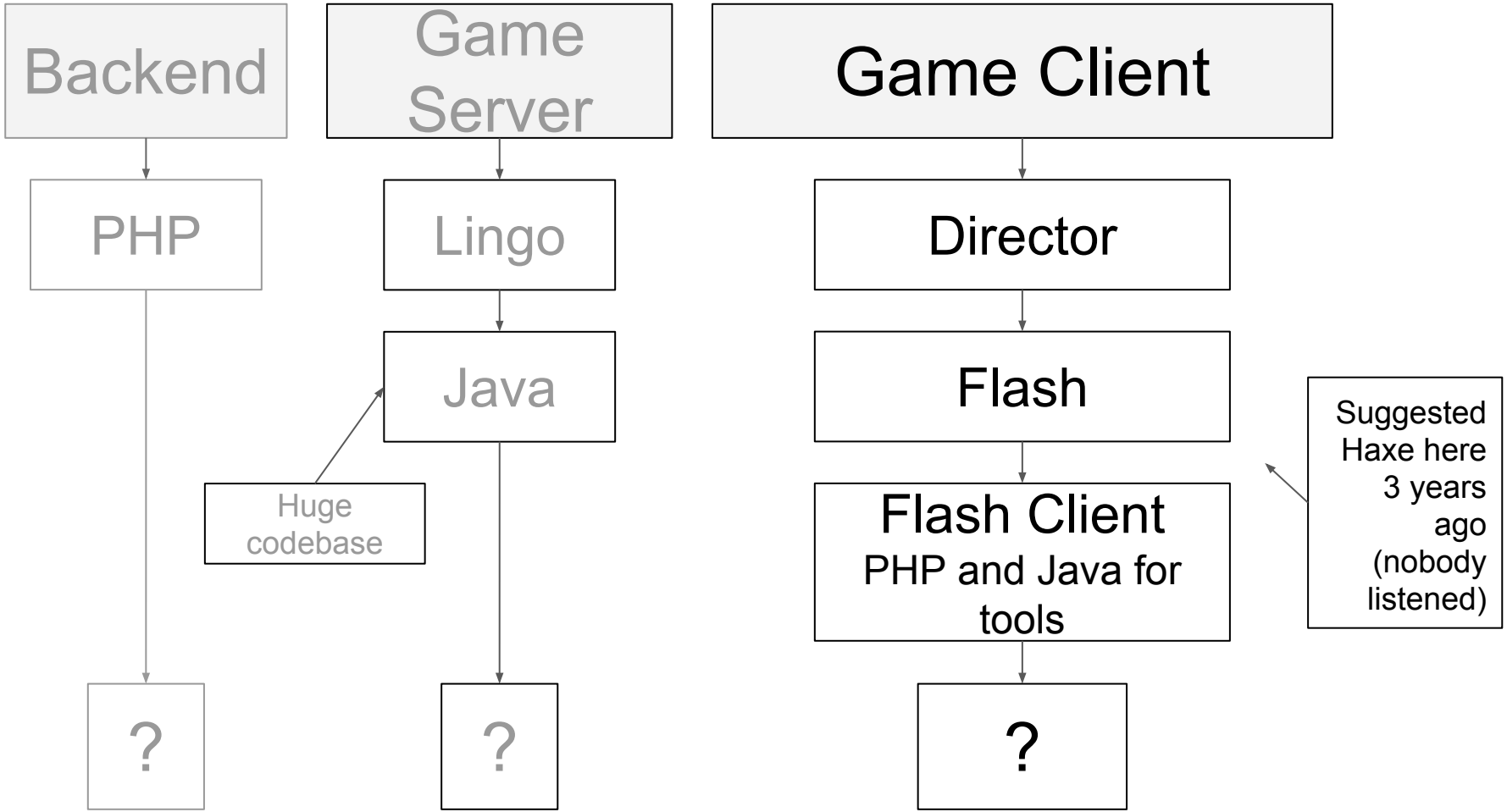

Java

Carlos Ballesteros Velasco (@soywiz)

- I’m a Tech Lead at Akamon.com.
- I’m responsible for the framework and the technology of the frontend / game client in our workplace.
- Also I have created several OSS haxelib libraries: `haxe-ws`, `haxe-crypto`, `haxe-cairo`, `openfl-(opus|ffmpeg|webp|webm)` among others.
- And contributed to the `intellij-haxe` plugin.
- That was in my spare time; lately I don’t have that much free time (sadly).

The story


Backend
Game Server
Game Client
PHP
Lingo
Director
Java
Flash
Huge codebase
Flash Client
PHP and Java for tools
Suggested Haxe here 3 years ago (nobody listened)
?
?
?

- Oh sh**, flash is dying for real.
- Suggested (again) Haxe for the frontend.
- Even migrated the whole frontend framework to Haxe as a spike.
- Backend people didn’t want to switch (because of the tooling).

- We wanted to share code and knowledg, if possible.
- Kotlin appeared.
- Allows to migrate Java to Kotlin in one click.
- Bidirectional interoperability with java.
- Modern language.
- Awesome tooling.

- Tried several available options to use Kotlin and target everywhere as Haxe does.
- No luck.
- Started JTransc as a crazy-holiday spike to solve this.
- We ended using it.

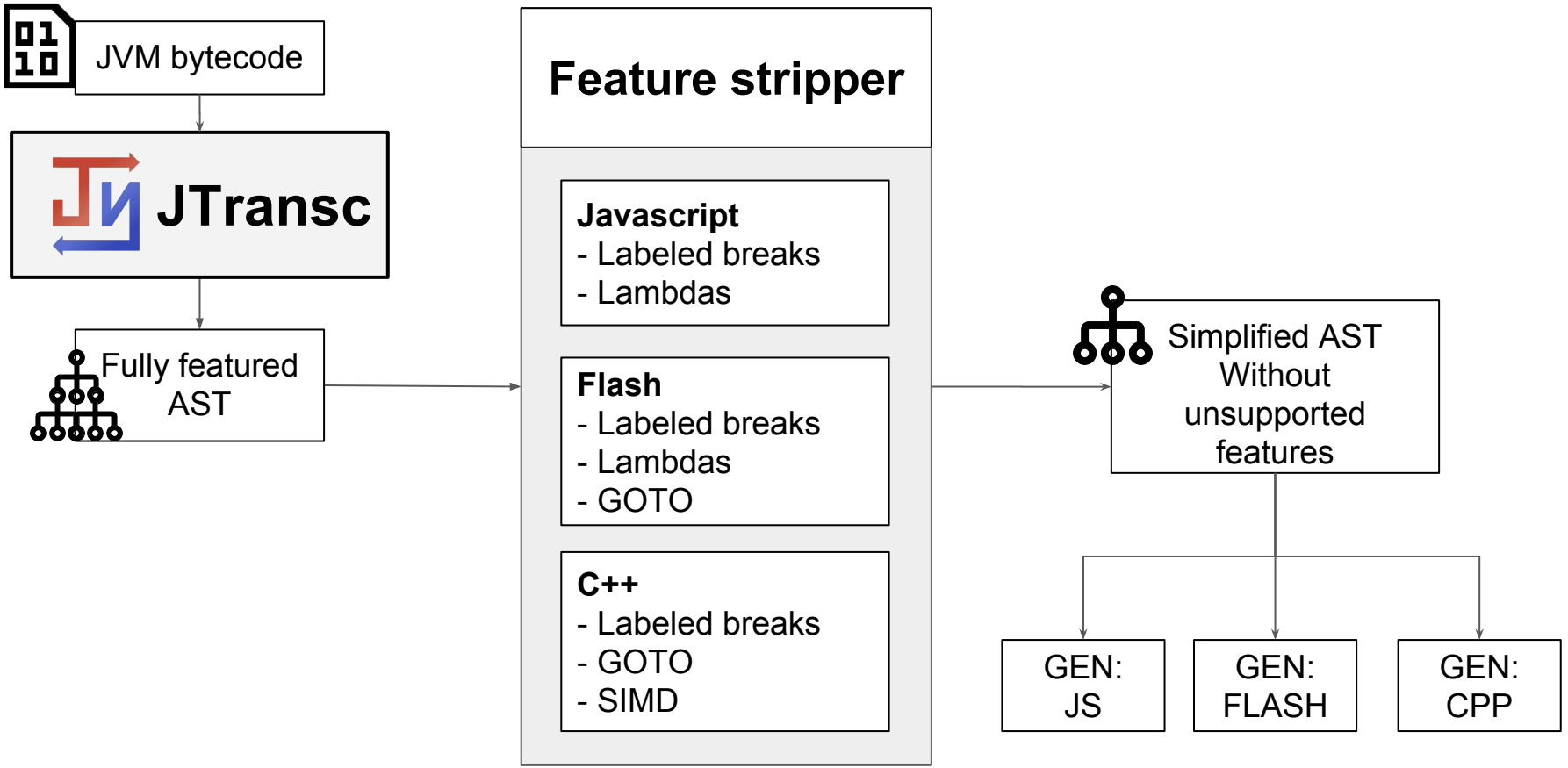
What is JTransc?

A transcompiler that converts JVM bytecode into plain Haxe

(And let Haxe do its magic)

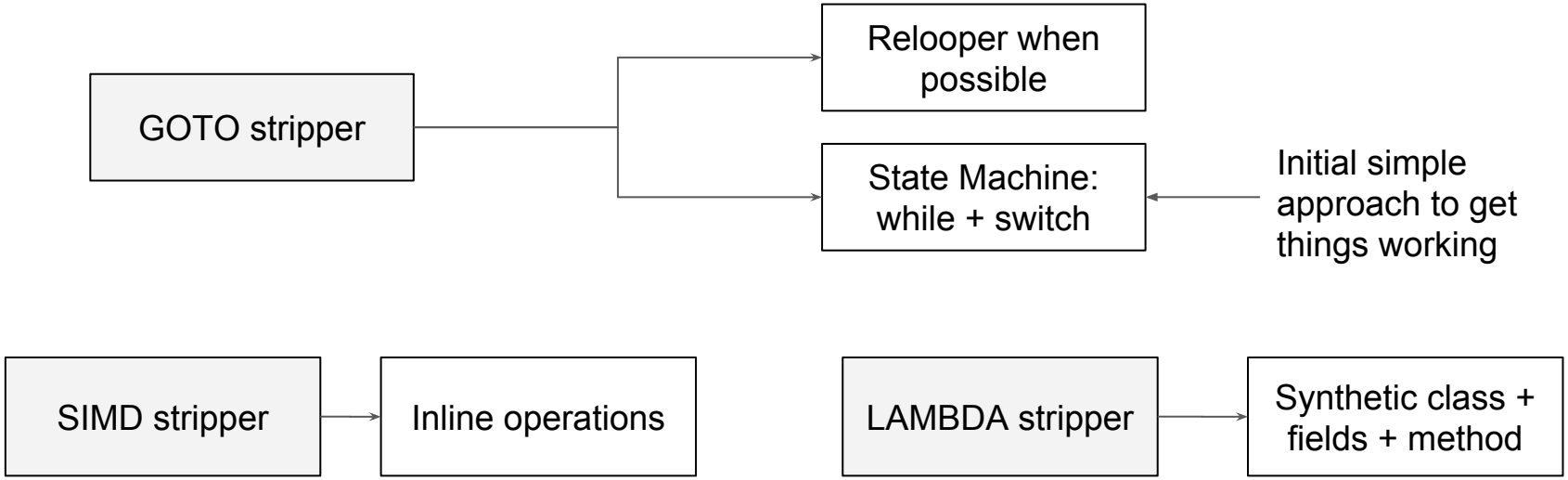
Initially not targeting Haxe

Initial Prototype



Feature-based AST

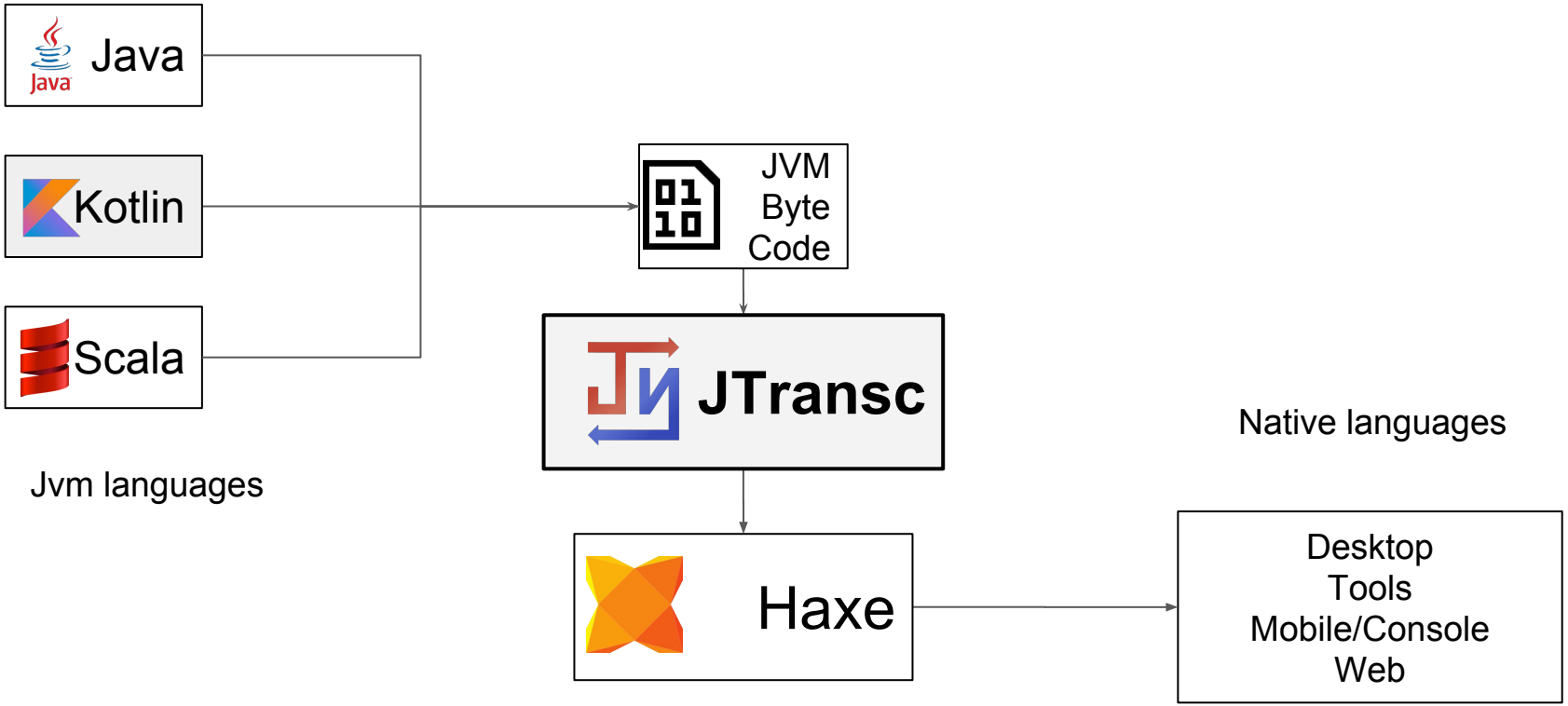
- Targets are not required to know how to handle all features/ast nodes
- New feature just requires a feature stripper. Not mandatory to optimize any target to use that feature.



Cool

- But Haxe already targets everywhere.
- I was investing time where I should not.
- So I started to focus on Haxe.

Current implementation



Still AST and feature based

- Still allows to add targets easily
- But right now, supported, main and only target is Haxe
- Haxe doesn’t support GOTO feature so it is striped (“relooped”)

Why program for JVM?

- Because of Kotlin :)
- Leverage lots and lots of Java libraries
- Very mature tooling. Specially IDE
- Also has maven and gradle: similar to haxelib but with plugins
- We already have a huge codebase in the backend using Java
- Because no preprocessor and macros

Why target to Haxe?

- 11 years of multiple targets
- Very small runtime
- Fast and small overhead
- Very flexible
- Because of preprocessor and macros :)

Alternatives

Why JTransc?

AOT	~~RoboVM~~	GWT	TeaVM	Kotlin (JS)	JTransc
Targets	~~iOS~~	Javascript	Javascript	JS/Android	Haxe (iOS, Android, JS, Flash...)
Reflection (DI, Serialization)	~~YES~~	No (hacks)	Metaprog.	Partial	YES (can opt-out)
Opensource	~~NO~~	YES	YES	YES	YES
Supported languages	~~Any JVM~~	Java7	Any JVM	Kotlin	Any JVM
Maturity	~~Mature~~	Mature	Partial mature	Starting	Starting
Output size	~~Huge~~	Small	Small	Medium	Big (will improve: DCE)
Code quality	~~Good~~	Good	Good	Good	Improving

Intended workflow (for games)

- Several “main” (entrypoints):
  - One main for JVM (injecting dependencies somehow)
  - One main for JTransc/Haxe (injecting jtransc-specific dependencies)
- Have always a default/dummy implementation that does not require JTransc:
  - One interface -> Multiple implementations
  - One class with a default implementation + @HaxeMethodBody* annotations
- Develop and debug using JVM
- Avoid threading: reactive + event-loop to reach single-threaded targets
- Game code totally portable without using JTransc directly: using interfaces.

How to use it? Maven: pom.xml

```
<plugin>
  <groupId>com.jtransc</groupId>
  <artifactId>jtransc-maven-plugin</artifactId>
  <version>${jtransc.version}</version>
  <configuration>
    <mainClass>com.example.MyClass</mainClass>
    <release>true</release>
    <targets><param>haxe:js</param></targets>
    <minimizeNames>false</minimizeNames>
  </configuration>
<executions><execution><goals><goal>jtransc</goal></goals></execution
></executions>
</plugin>
```

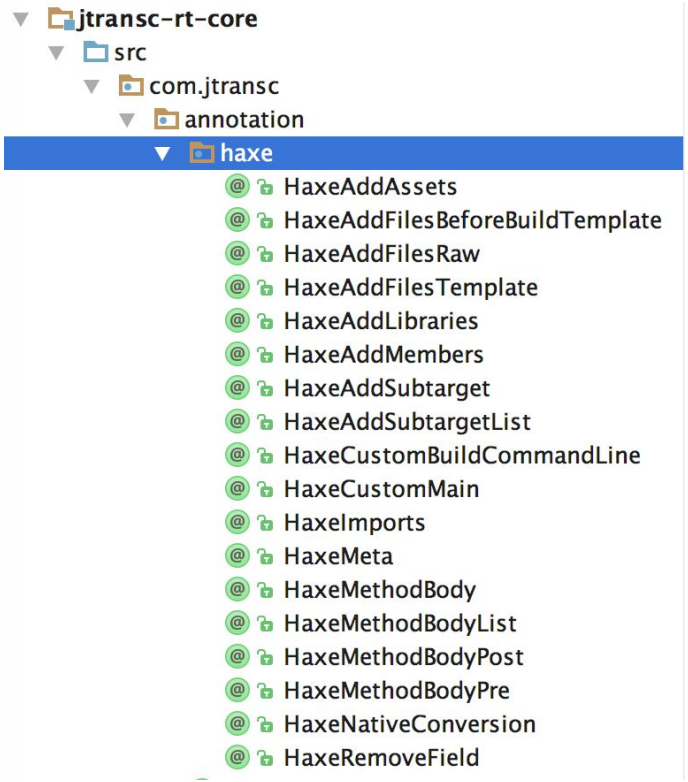
mvn package

Live Example

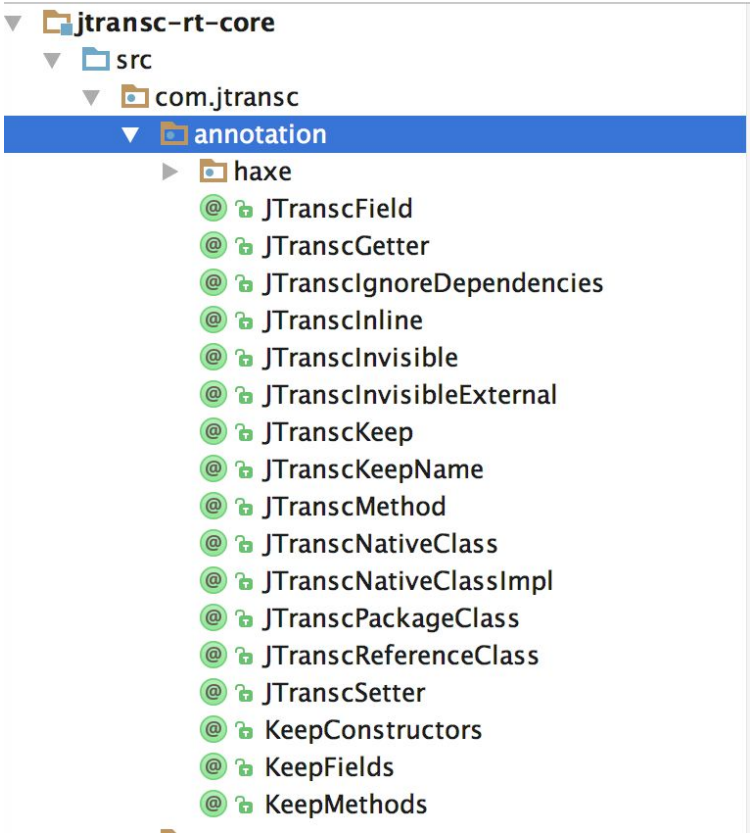
jtransc-rt-core

- Annotations for JTransc
- Optimized utility classes
- Can be used without JTransc

Annotations



Haxe specific



Generic

FastIntMap.java

```java
/.../

package com.jtransc;

import ...

@JTranscInvisible
@HaxeAddMembers({
        "var _map = new Map<Int, Dynamic>();"
})
public class FastIntMap<T> {
    @HaxeRemoveField
    private HashMap<Integer, T> map;

    @HaxeMethodBody("")
    public FastIntMap() { this.map = new HashMap<Integer, T>(); }

    @HaxeMethodBody("return _map.get(p0);")
    public T get(int key) { return this.map.get(key); }

    @HaxeMethodBody("_map.set(p0, p1);")
    public void set(int key, T value) { this.map.put(key, value); }

    @HaxeMethodBody("return _map.exists(p0);")
    public boolean has(int key) { return this.map.containsKey(key); }
}
```

Class works on plain **Java**/Kotlin.

But it is optimized when generating **Haxe**.

```java
final public class Mem {
    @HaxeRemoveField
    static private FastMemory mem;

    @HaxeMethodBody(target = "flash", value = "flash.Memory.select(p0._data.getData());")
    @HaxeMethodBody("" +
        "var mem    = p0._data;\n" +
        "bytes      = mem;\n" +
        "byteMem    = haxe.io.UInt8Array.fromBytes(mem);\n" +
        "shortMem   = haxe.io.UInt16Array.fromBytes(mem);\n" +
        "intMem     = haxe.io.Int32Array.fromBytes(mem);\n" +
        "floatMem   = haxe.io.Float32Array.fromBytes(mem);\n" +
        "doubleMem = haxe.io.Float64Array.fromBytes(mem);\n"
    )
    static public void select(FastMemory mem) { Mem.mem = mem; }

    @JTranscInline
    @HaxeMethodBody(target = "flash", value = "return flash.Memory.getByte(p0 << 0);")
    @HaxeMethodBody("return byteMem.get(p0);")
    static public int li8(int address) { return mem.getAlignedInt8(address); }

    @JTranscInline
    @HaxeMethodBody(target = "flash", value = "return flash.Memory.getUI16(p0 << 1);")
    @HaxeMethodBody("return shortMem.get(p0);")
    static public int li16(int address2) { return mem.getAlignedInt16(address2); }
```

Class works on plain **Java**/Kotlin JVM.

But it is optimized when generating **Haxe**.

And even more optimized for **Haxe-Flash** target (using intrinsic opcodes and inlining)

Generated Haxe

Mem_.hx

```haxe
package com.jtransc;
class Mem_ extends java_.lang.Object_ {
  public function new() {
    super(); SI();
  }
  static private var bytes:haxe.io.Bytes;
  static private var byteMem:haxe.io.UInt8Array;
  // ...

  public   function com_jtransc_Mem_init___V/*<init>*/():com.jtransc.Mem_ {
    // 33
    N.c(this, java_.lang.Object_).java_lang_Object_init___V();
    return this;
  }
  static  public   function select_Lcom_jtransc_FastMemory__V/*select*/(p0:com.jtransc.FastMemory_):Void {
    #if (flash) flash.Memory.select(p0._data.getData());
    #else
      var mem   = p0._data;
      bytes     = mem;
      byteMem   = haxe.io.UInt8Array.fromBytes(mem);
      // ...
    #end
  }
  static  public inline   function li8_I_I/*li8*/(p0:Int):Int {
    #if (flash) return flash.Memory.getByte(p0 << 0);
    #else return byteMem.get(p0); #end
  }
```

@HaxeMethodBody
Is converted into a plain `#if #else` chain (when required).

Names are changed to be compatible with Haxe

- Packages converted to lowercase.
- Class names capitalized.
- Append _ to all class names to avoid conflict with top-level classes (which don’t have a proper fqname) Math class for example.
- Method names are mangled using java’s method descriptor.
- Keywords append _.

Also there are minimized names

- JTransc has a mode that minimizes all identifiers: packages, classes, methods, fields.
- Reflection works even in this mode.
- Intended for javascript release builds to reduce output size and obfuscate.
- **Generated names are not predictable.**

Minitemplates

Since identifiers are not predictable:

To reference JVM ids in embedded Haxe code, there is a template engine (that works on all JTransc's pieces):

```
{% CLASS com.example.MyClass %}
{% METHOD com.exampleMyClass:method:(I)V %}
{% FIELD com.exampleMyClass:field %}
```

Template engine based on Twig / atpl.js
It also supports `{% for %}` `{% if %}` `{{ expr }}`, some `|filters` and other limited but flexible stuff (no inheritance or macros).

Minitemplates (example)

ws_haxe.kt

```
17
18  @HaxeAddMembers("var ws:haxe.net.WebSocket;")
19  @HaxeAddLibraries("haxe-ws:0.0.6")
20  class WebSocketHaxe(url: String, subprotocols: Array<String>?, debug: Boolean) : WebSocket(url) {
21      init {
22          ws_init(url, subprotocols, debug)
23          process()
24      }
25
26      @HaxeMethodBody("""
27          this.ws = haxe.net.WebSocket.create(p0._str, (p1 != null) ? (cast p1.toArray()) : null, null, p2);
28          this.ws.onopen = function() { this.{% METHOD nova.net.ws.WebSocketHaxe:onConnectSend %}(); };
29          this.ws.onclose = function() { this.{% METHOD nova.net.ws.WebSocketHaxe:onDisconnectedSend %}(); };
30          this.ws.onmessageString = function(m:String) { this.{% METHOD nova.net.ws.WebSocketHaxe:onStringMessageSend %
31      """)
32      private fun ws_init(url: String, subprotocols: Array<String>?, debug: Boolean) {
33      }
34
35      @HaxeMethodBody("this.ws.sendString(N.i_str(p0));")
36      external override fun send(message: String)
37
```

```
this.ws.onopen = function() { this.{% METHOD nova.net.ws.WebSocketHaxe:onConnectSend %}(); };
```

Custom Haxe Frameworks and Tools

LIME

We are currently using Lime.

Lime was initially hardcoded in JTransc as it uses a custom command line and requires a `project.xml` file.

Minitemplates now allow potentially to use any custom haxe framework with custom command-line and or project files: kha, nme, flixel, flambe, openfl, snowkit...

JTransc-media https://github.com/jtransc/jtransc-media

JTransc-media provides a very simple interface for graphics, audio and input.

Right now it has two backends: libgdx (JVM) and lime (Haxe).

It is a good example about how to use JTransc with custom Haxe game frameworks.

```
@HaxeAddFilesTemplate({
    "AGALMiniAssembler.hx",
    "HaxeLimeAssets.hx",
    "HaxeLimeAudio.hx",
    "HaxeLimeJTranscApplication.hx",
    "HaxeLimeRender.hx",
    "HaxeLimeRenderFlash.hx",
    "HaxeLimeRenderGL.hx",
    "HaxeLimeRenderImpl.hx",
    "HaxeLimeIO.hx",
    "HaxeLimeLanguage.hx"
})
```

```
@HaxeCustomMain("" +
    "package {{ entryPointPackage }};\n" +
    "class {{ entryPointSimpleName }} extends HaxeLimeJTranscApplication {\n" +
    "    public function new() {\n" +
    "        super();\n" +
    "        {{ inits }}\n" +
    "        {{ mainClass }}.{{ mainMethod }}(HaxeNatives.strArray(HaxeNatives.args()));\n" +
    "    }\n" +
    "}\n"
)
```

```
@HaxeAddSubtargetList({
    @HaxeAddSubtarget(name = "android"),
    @HaxeAddSubtarget(name = "blackberry"),
    @HaxeAddSubtarget(name = "desktop"),
    @HaxeAddSubtarget(name = "emscripten"),
    @HaxeAddSubtarget(name = "flash", alias = { "swf", "as3" }),
    @HaxeAddSubtarget(name = "html5", alias = { "js" }),
    @HaxeAddSubtarget(name = "ios"),
    @HaxeAddSubtarget(name = "linux"),
    @HaxeAddSubtarget(name = "mac"),
    @HaxeAddSubtarget(name = "tizen"),
    @HaxeAddSubtarget(name = "tvos"),
    @HaxeAddSubtarget(name = "webos"),
    @HaxeAddSubtarget(name = "windows"),
    @HaxeAddSubtarget(name = "neko")
```

```
@HaxeCustomBuildCommandLine({
    "@limebuild.cmd"
})
@HaxeAddLibraries({
    "lime:2.9.1"
})

@HaxeAddFilesBeforeBuildTemplate({
    "program.xml"
})

public class JTranscLime {
```

Minitemplates in command line and project file

limebuild.cmd

```
haxelib
run
lime
build
{{ actualSubtarget.name }}
{% if debug %}-debug{% else %}-release{% end %}
-Dsource-header=0

{% if actualSubtarget.name == "ios" %}
    ----
    # releasetype = release
    # releasetype = debug

    /usr/bin/xcrun
    -sdk
    iphoneos
    PackageApplication
    -v
    {{ buildFolder }}/export/{{ releasetype }}/ic
    -o
    {{ buildFolder }}/export/{{ releasetype }}/ic
{% end %}
```

program.xml

```
<?xml version="1.0" encoding="utf-8"?>
<project>
    <meta title="{{ title }}" package="{{ package }}" version="{{ versi
    <app main="{{ entryPointClass }}" path="out" file="{{ name }}" />
    <app swf-version="11.8" />

    <window width="{{ initialWidth }}" height="{{ initialHeight }}" bac
    <window fullscreen="false" resizable="true" borderless="false" vsyn
    <window fullscreen="true" if="mobile" />
    <window fps="60" unless="js" />
    <window fps="0" if="js" />
    <window width="0" height="0" if="html5" />

    <set name="BUILD_DIR" value="export/debug/" if="debug" />
    <set name="BUILD_DIR" value="export/release/" if="release" />

    {% for flag in haxeExtraFlags %}
        <haxeflag name="{{ flag.first }}" value="{{ flag.second }}" />
    {% end %}

    {% for define in haxeExtraDefines %}
        <haxedef name="{{ define }}" />
    {% end %}

    <source path="src" />
    <assets path="{{ tempAssetsDir }}" rename="assets" embed="{{ embedR

    {% if hasIcon %}
        <icon path="{{ settings.icon }}" />
    {% end %}

    {% if extra.certificate_path %}
    {% debug "extra.certificate_path: " + extra.certificate_path %}
```

Custom frameworks

In the case the class containing those annotations is referenced in the application, everything is configured.

Everything works seamlessly.

You just use Maven (**`mvn package`**), but it internally installs all the required `haxelib` libraries, generate all the required files and runs the proper haxe or framework commands to build/package your application.

gdx-backend-jtransc

- https://github.com/jtransc/gdx-backend-jtransc
- https://libgdx.badlogicgames.com/

WIP: libGDX backend for JTransc

libGDX is a game library for JVM

Some libGDX games run without modifications.

Direct Interoperability

@JTranscNativeClass

```
static private class HaxeAdler32Tools {
    @HaxeMethodBody("return new haxe.crypto.Adler32();")
    native static public HaxeAdler32 create();
}

@JTranscNativeClass("haxe.crypto.Adler32")
private static class HaxeAdler32 {
    public HaxeAdler32() {
    }

    native public boolean equals(HaxeAdler32 that);

    native public int get();

    native public void update(byte[] b, int pos, int len);

    native static public int make(byte[] b);

    native static public HaxeAdler32 read(InputStream i);
}
```

- Direct haxe calls/definitions
- Class constructor requires a static method like that (that restriction will be lifted in future versions)
- **Use this carefully. Because Haxe objects do not extend java.lang. Object**

Automatic conversions and boxing

- `byte[]` ↹ `Bytes`
- `java.io.InputStream` ↹ `haxe.io.Input`
- `java.lang.String` ↹ `String`
- `...`

Allows custom conversions

Custom conversions: @HaxeNativeConversions

```
@HaxeAddFilesTemplate({ "JavaHaxeInput.hx" })
@HaxeNativeConversion(
    haxeType = "haxe.io.Input",
    toHaxe = "new JavaHaxeInput.Haxe(@self)",
    toJava = "new JavaHaxeInput.Java(@self)"
)
public abstract class InputStream implements Closeable {
```

JavaHaxeInput.hx

```
1  class JavaHaxeInput {
2  }
3
4  class Haxe extends haxe.io.Input {
5      var i: {% CLASS java.io.InputStream %};
6      public function new(i) {
7          this.i = i;
8      }
9      override public function readByte():Int return this.i.{% METHOD java.io.InputStream:read:()I %}();
10 }
11
12 class Java extends {% CLASS java.io.InputStream %} {
13     var i: haxe.io.Input;
14     public function new(i) {
15         super();
16         this.i = i;
17     }
18     override public function {% METHOD java.io.InputStream:read:()I %}() return i.readByte();
19 }
```

Inception

Jtransc is able to compile itself

```
 soywiz@MacBook-Pro-de-Carlos  ~/Projects/jtransc/inception/target  master  node jtransc-inception.js ~/Projects/jtransc/inception/example -main Example -o
ut example.js
AllBuild.build(): language=com.jtransc.gen.haxe.HaxeGenDescriptor@97, subtarget=js, entryPoint=Example, output=example.js, targetDirectory=/var/folders/6_/qlw7d
nss3mg1fvwptzpwprt40000gn/T/
ClassPath: /Users/soywiz/.m2/repository/com/jtransc/jtransc-rt/0.1.5/jtransc-rt-0.1.5.jar
ClassPath: /Users/soywiz/.m2/repository/com/jtransc/jtransc-rt-core/0.1.5/jtransc-rt-core-0.1.5.jar
ClassPath: /Users/soywiz/Projects/jtransc/inception/example
Generating AST...
file_separator: / ... PathSeparator: :
Processing classes...
Ok classes=18, time=56474
Ok (62952)
Preparing processor...
Temporal haxe files: /var/folders/6_/qlw7dnss3mg1fvwptzpwprt40000gn/T//jtransc-haxe
Ok (4)
Building source...
Ok (8379)
Compiling...
haxe.build (0.1.5) source path: \var\folders\6_\qlw7dnss3mg1fvwptzpwprt40000gn\T\jtransc-haxe\src
:: REFERENCED LIBS: []
GenTargetInfo.haxeCopyResourcesToAssetsFolder: /Users/soywiz/Projects/jtransc/inception/target
Compiling...
Running: haxe -debug -js /Users/soywiz/Projects/jtransc/inception/target/example.js -cp \var\folders\6_\qlw7dnss3mg1fvwptzpwprt40000gn\T\jtransc-haxe\src -main
Example_EntryPoint_.hx -D no-analyzer
Ok (2984)
 soywiz@MacBook-Pro-de-Carlos  ~/Projects/jtransc/inception/target  master  node example.js
Hello World!
 soywiz@MacBook-Pro-de-Carlos  ~/Projects/jtransc/inception/target  master
```

FFI - JNA

Jtransc is able to load libraries

```java
import ...

public class BeepExample {
    public interface Kernel32 extends Library {
        @StdCall
        boolean Beep(int FREQUENCY, int DURATION);

        @StdCall
        void Sleep(int DURATION);
    }

    public interface User32 extends Library {
        @StdCall
        boolean MessageBoxA(int a, String title, String text, int b);
    }

    public static void main(String[] args) throws InterruptedException {
        if (JTranscSystem.isWindows()) {
            Kernel32 kernel32 = (Kernel32) Native.loadLibrary("kernel32", Kernel32.class);
            User32 user32 = (User32) Native.loadLibrary("user32", User32.class);
            for (int n = 0; n < 6; n++) {
                kernel32.Beep(698 * (n + 1), 300);
                kernel32.Sleep(100);
            }
            user32.MessageBoxA(0, "done!", "done!", 0);
        } else {
            System.out.println("This demo just works on windows!");
        }
    }
}
```

Compatible with plain Java JNA.

It works with:

- Plain JVM
- HXCPP
- Node.js (using ffi)

CPP embed

Jtransc is able to embed c++

```java
package example;

import jtransc.annotation.haxe.HaxeAddFiles;
import jtransc.annotation.haxe.HaxeMeta;
import jtransc.annotation.haxe.HaxeMethodBody;

public class Test {
    static public void main(String[] args) {
        System.out.println(Demo.mysum(7, 3));
    }
}

@HaxeMeta("@:cppInclude('./../test.c')")
@HaxeAddFiles("test.c")
class Demo {
    @HaxeMeta("@:noStack")
    @HaxeMethodBody("return untyped __cpp__('::sum({0}, {1})', p0, p1);")
    static native public int mysum(int a, int b);
}
```

@HaxeMeta does the trick. It allows to put code where haxe metas can be placed.

In the works

What I’m working on right now?

- More size and speed optimizations
- IntelliJ debugger

Future

What’s next?

- Further size and speed optimizations
- SIMD
- Workers/Tasks for safe threading
- Haxe abstracts
- Struct access
- ARC vs GC vs RAII
- Full Java8

Live Example

https://github.com/jtransc/jtransc-examples-binaries

Contributions:

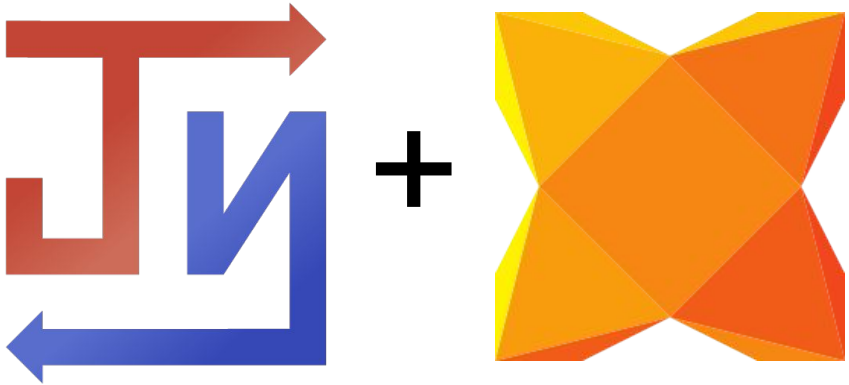
- Project 100% OpenSource
- Contibutions are always welcomed! (and not just code):

- Pull Requests
- Bug Reports
- Suggestions
- Documentation

- Examples
- General feedback
- Spread the Word
- Write about it in blogs

...You name it!

Questions & Answers



Stay in touch:

github.com/**jtransc**

github.com/**soywiz**

@cballesterosvel

blog.jtransc.com

Extra

JTransc-dynarec - https://github.com/jtransc/jtransc-dynarec

Brainfuck Example:

https://github.com/jtransc/jtransc-dynarec/blob/master/src/com/jtransc/dynarec/example/BrainfuckDynarec.java

~100 LoC brainfuck recompiler written in java that transforms at runtime brainfuck code into the executing target (using jtransc-dynarec).

For example on javascript it will generate javascript and will create a function at runtime. The same will go for PHP, JVM, C# and C++ using LibJIT. On unsupported targets it will interpret it.

So maybe I could port my Javascript's PSP emulator to kotlin and run on the browser, desktop and mobile too :)