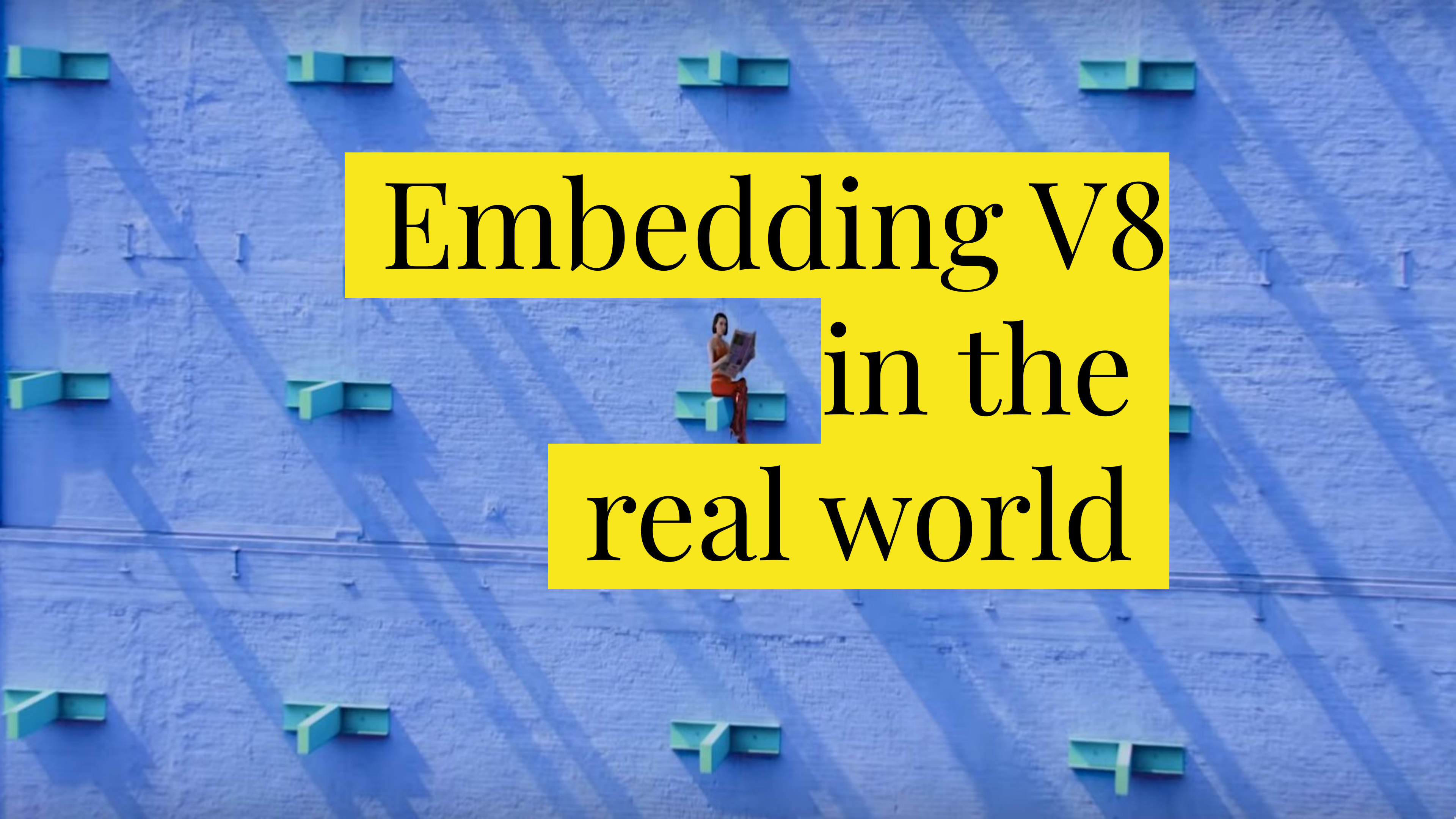

Embedding V8
in the
real world

Vladimir Mutafov & Stanimira Vlaeva
Software Engineers at Progress
NativeScript Core Team


A Story
in 4 Parts

Introduction

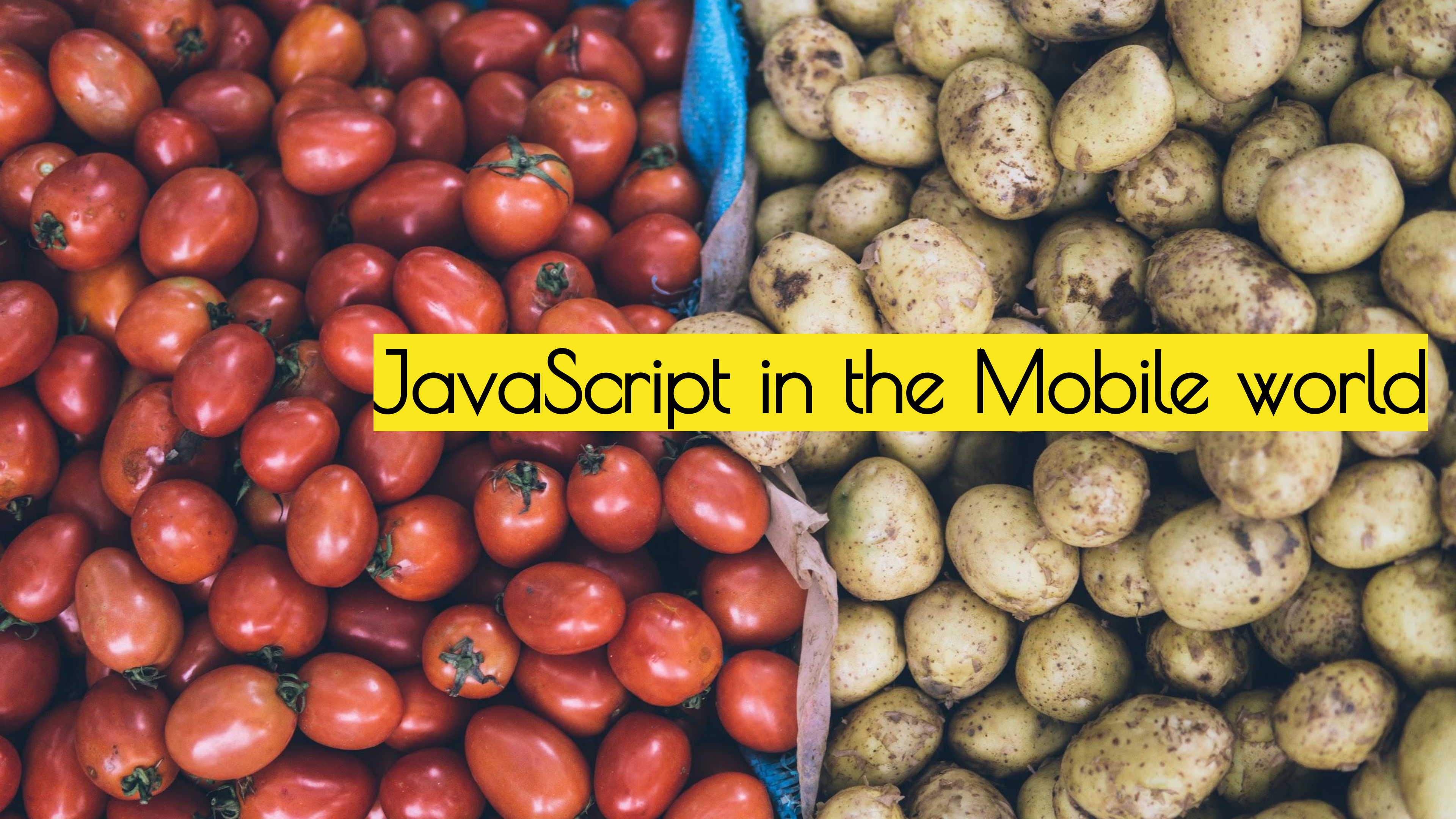
Part 0 / 4

So...

NativeScript

Framework for building native Android and iOS apps with Angular, Vue or plain JS.

JavaScript in the Mobile world



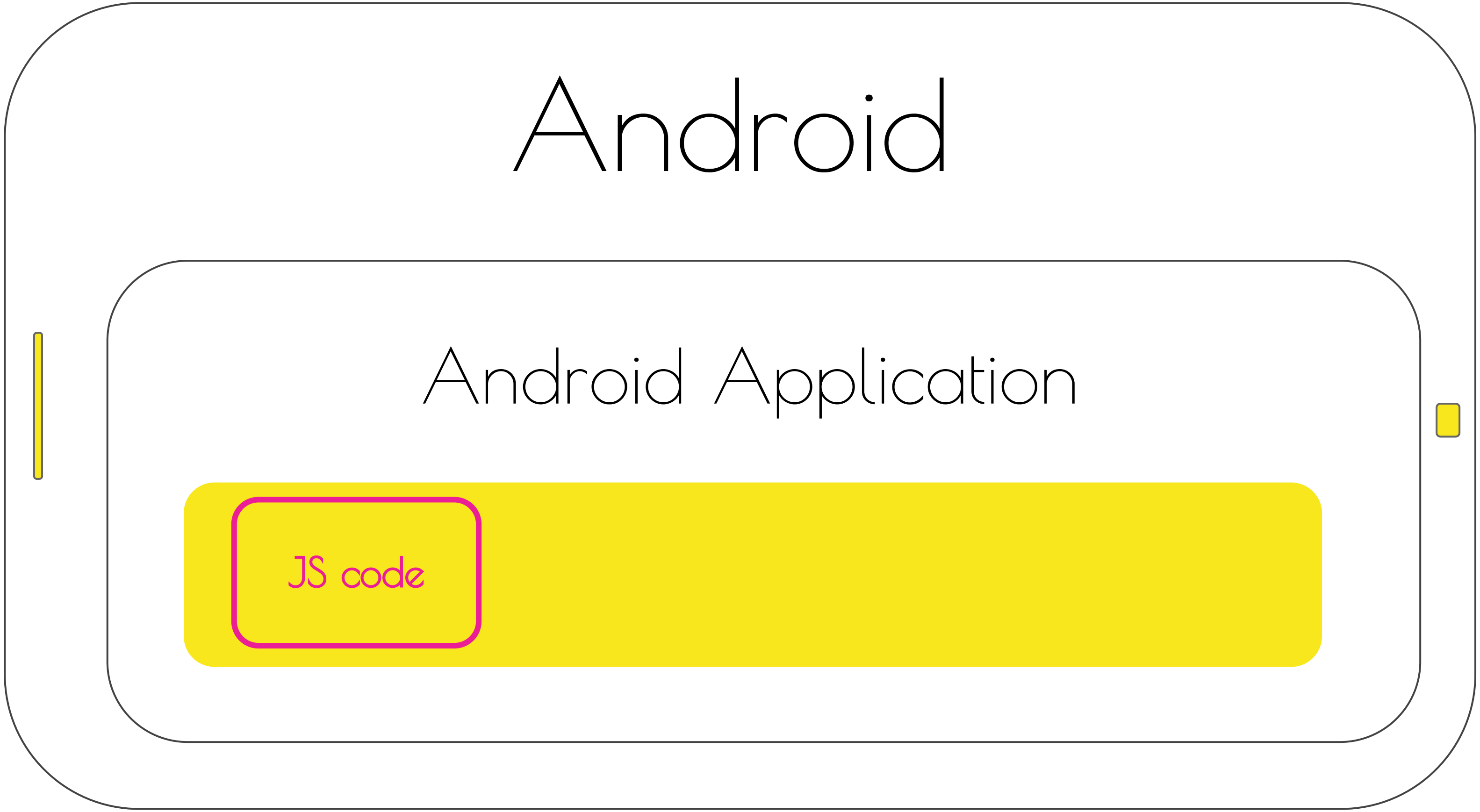

Application Framework
NativeScript 'light'
Data-binding, Navigation, ...
Angular
Vue
Cross Platform Abstraction
Layouts, UI Widgets, CSS, ...
Native API Access
NativeScript Android Runtime
NativeScript iOS Runtime
Native Code
Android
iOS

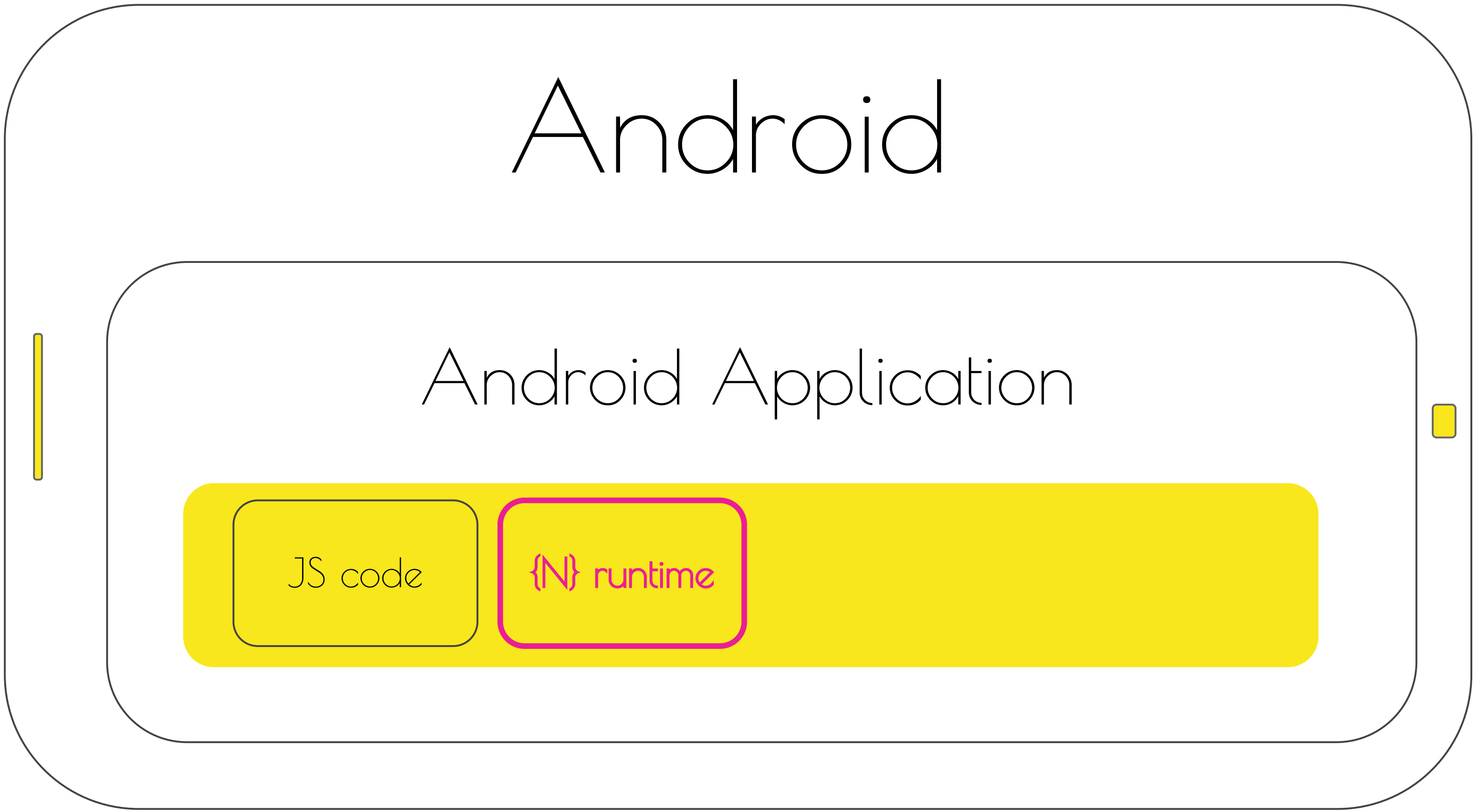
Native API Access

Part 1 / 4

The Application Package


Android
Android Application
NativeScript Magic


Android
Android Application
JS code


Android
Android Application
JS code
{N} runtime


Executing JavaScript

V8

JavaScript Engine

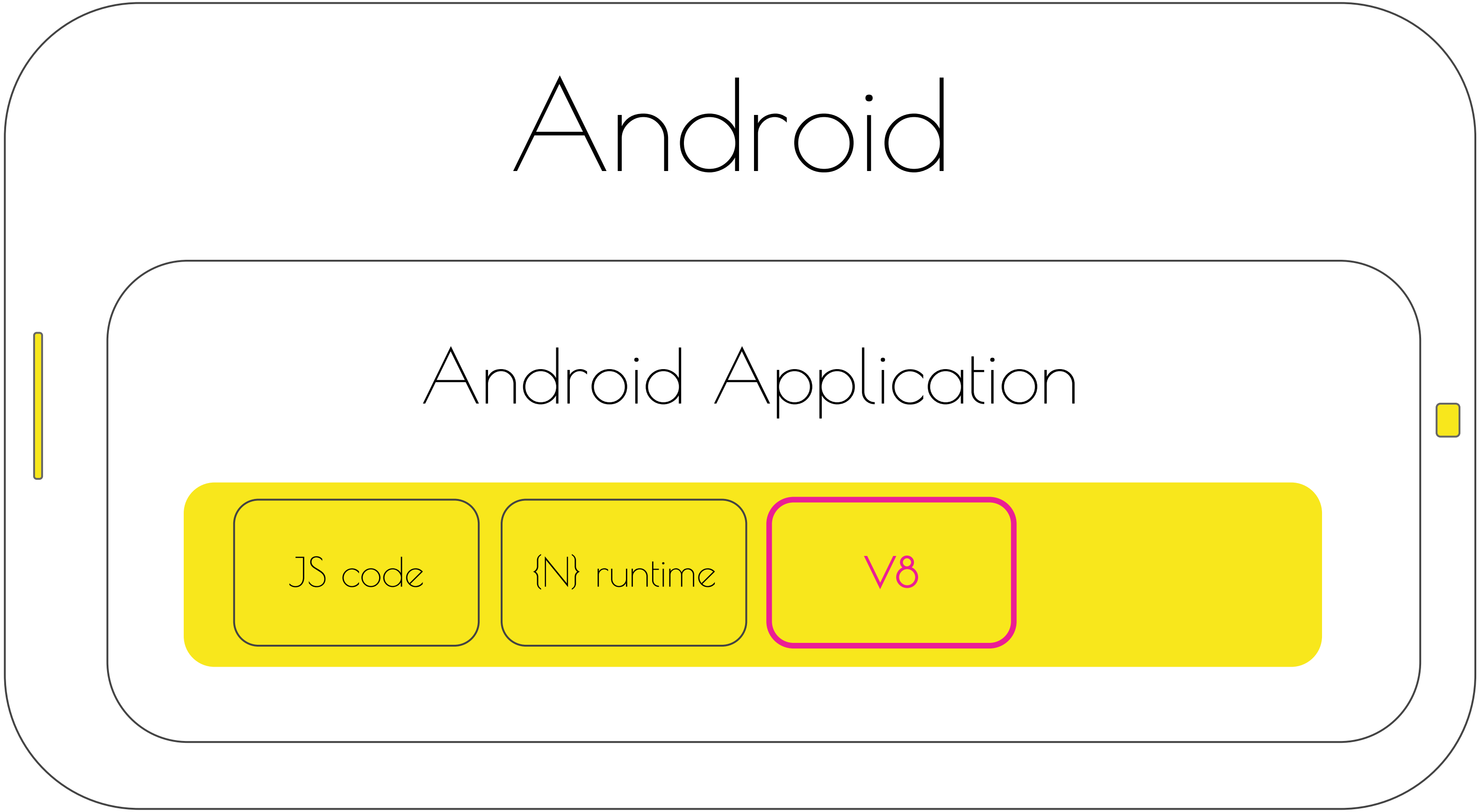
Executes JS

Embedded in Chrome, Node, and NativeScript

Read more

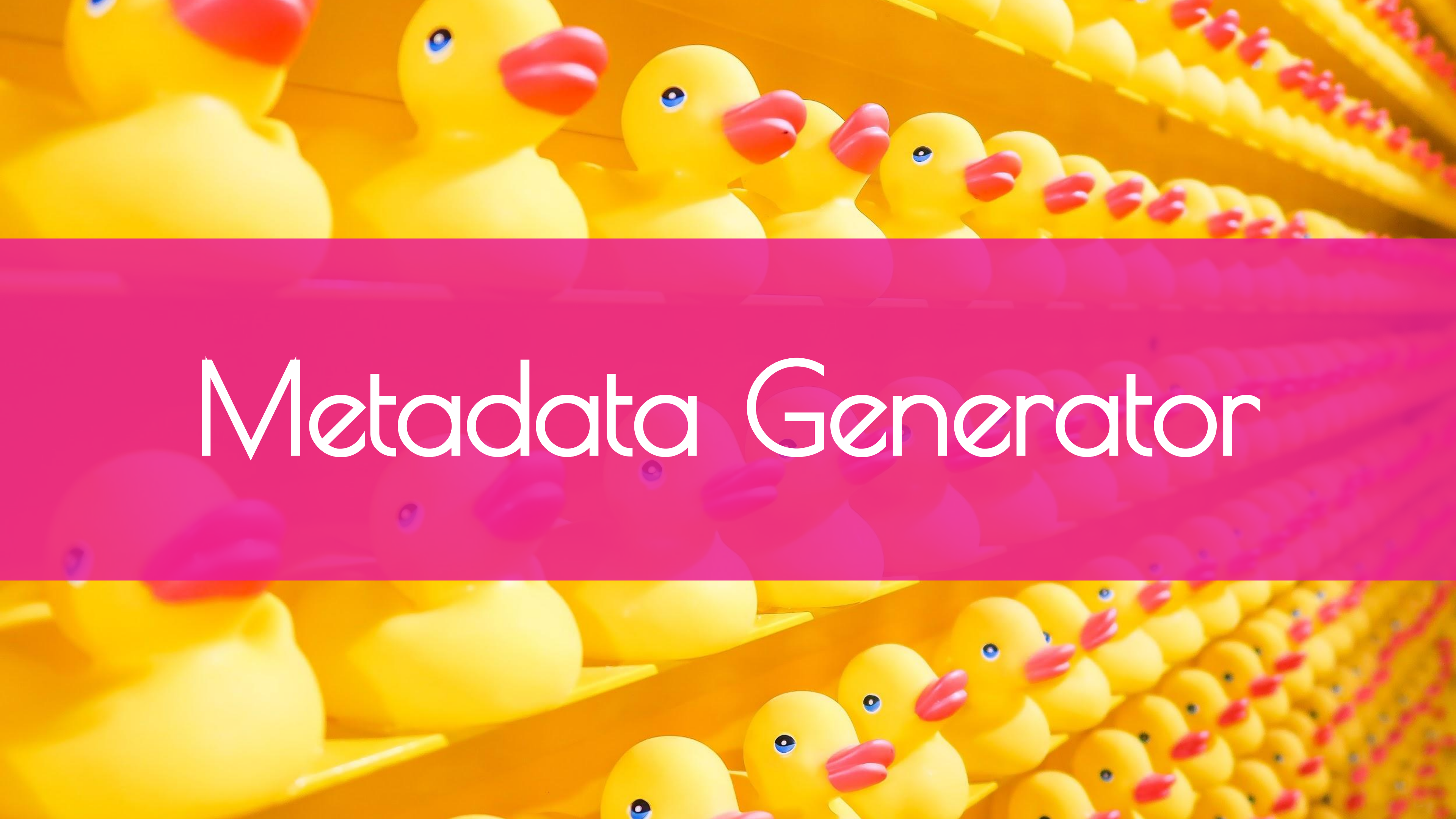
A crash course in JIT compilers by Lin Clark

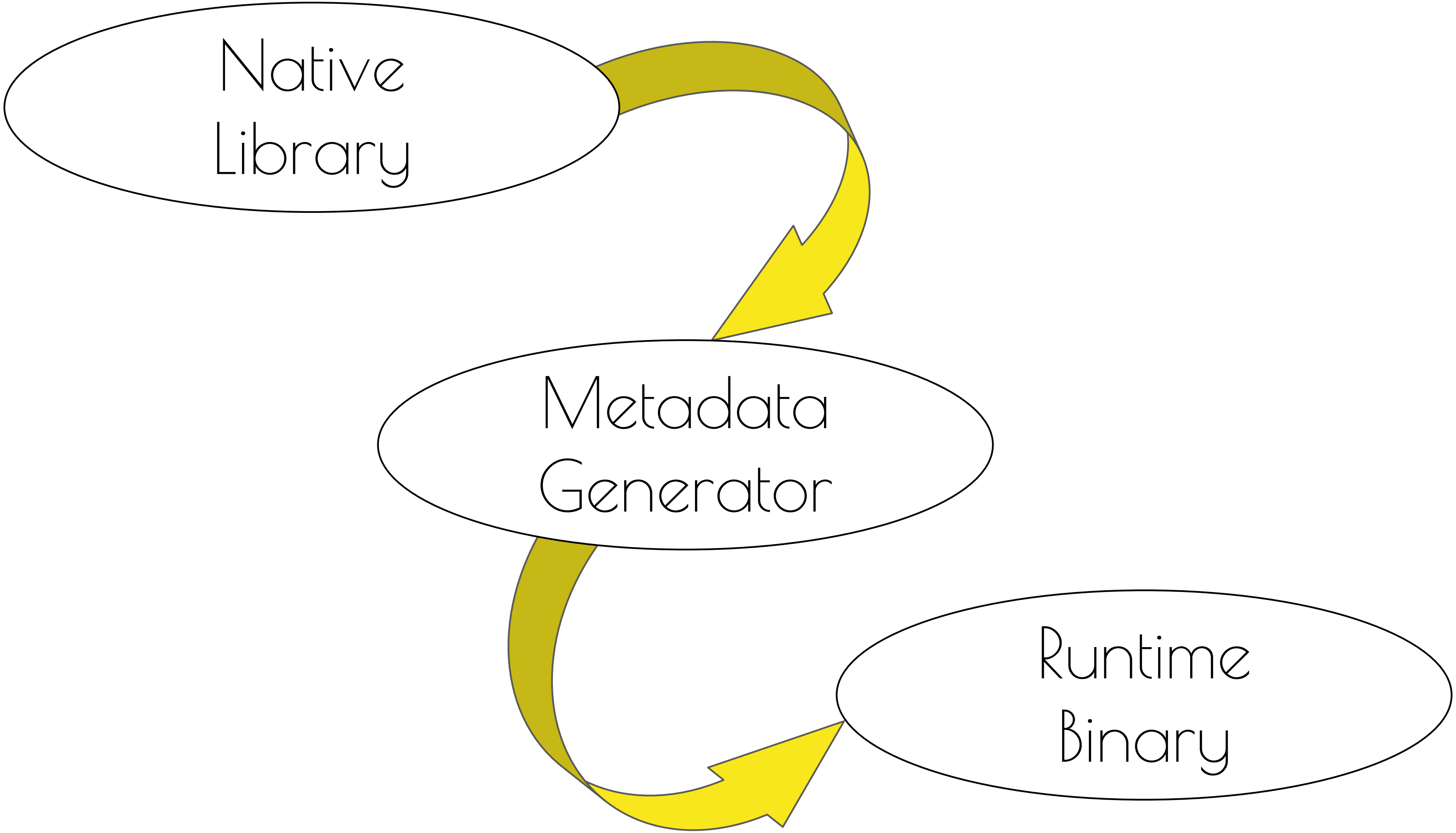
Life of a Script
by Sathya Gunasekaran & Jakob Kummerow


Android
Android Application
JS code
{N} runtime
V8

```
const recorder = new android.media.MediaRecorder();
```

Wut?

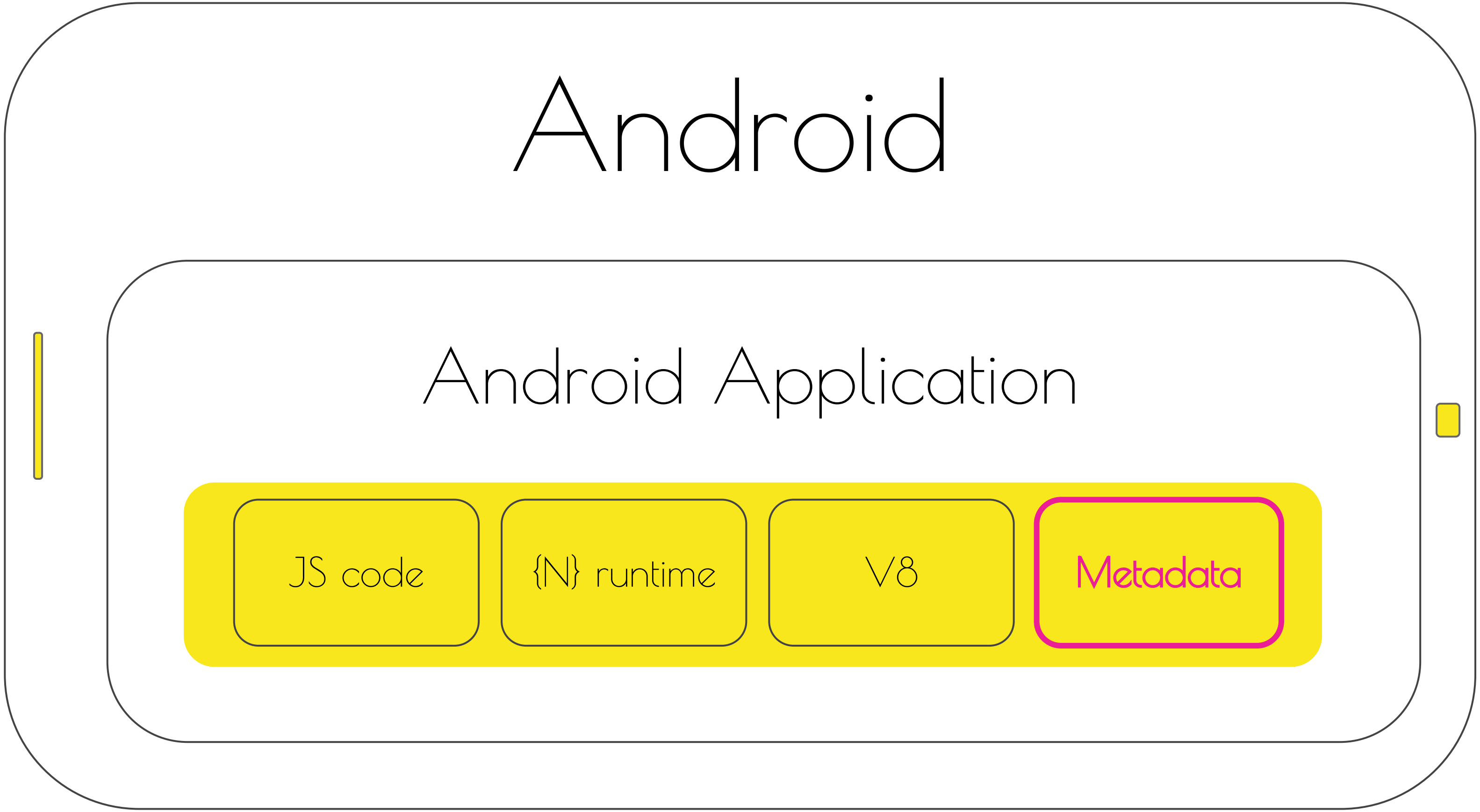

Metadata Generator



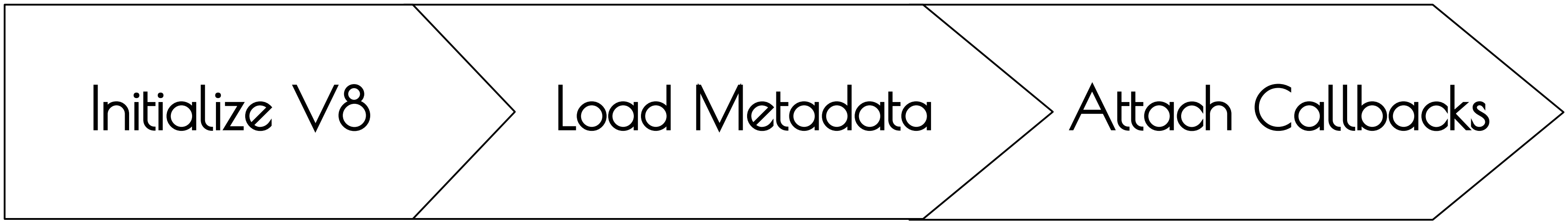
Native Library
Metadata Generator
Runtime Binary

```
const recorder = new android.media.MediaRecorder();
```

METADAT
A


Android
Android Application
JS code
{N} runtime
V8
Metadata

Application launch



Callbacks

`android.media.MediaRecorder`

`android` -> Set as global object in the running V8 instance

`android.media` -> Package getter callback

finds `android.media` in the `android` package metadata

`android.media.MediaRecorder` -> Package getter cb

finds `MediaRecorder` in `android.media`

`MediaRecorder` is a class -> a constructor function is returned

```
new android.media.MediaRecorder()
```

Constructor callback

Instantiates the native object in the Android world

How?

JNI

Java Native Interface

Allows V8 to send instructions to ART and vice versa.

The bridge between the two VMs.

```
new android.media.MediaRecorder()
```

Constructor callback

Instantiates the native object in the Android world

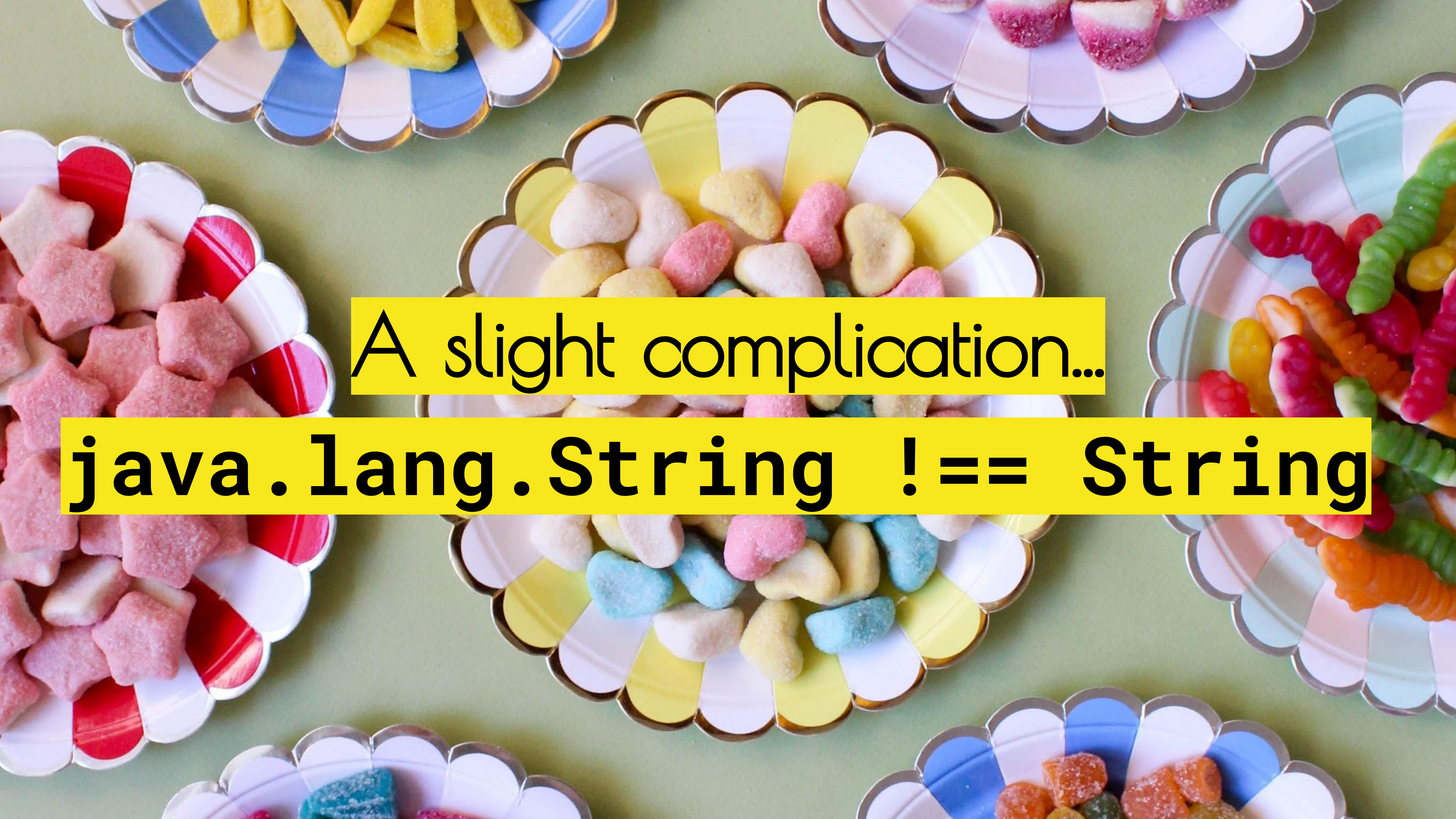
Creates a JS proxy object

Returns it back to the JS world

`recorder.someRandomField`

`Field getter callback`

Queries the original Java object for `someRandomField`


A slight complication...
java.lang.String !== String

Marshalling

Converts data from the Java world to the JS world and vice versa.

Java objects are proxied to special JS objects.

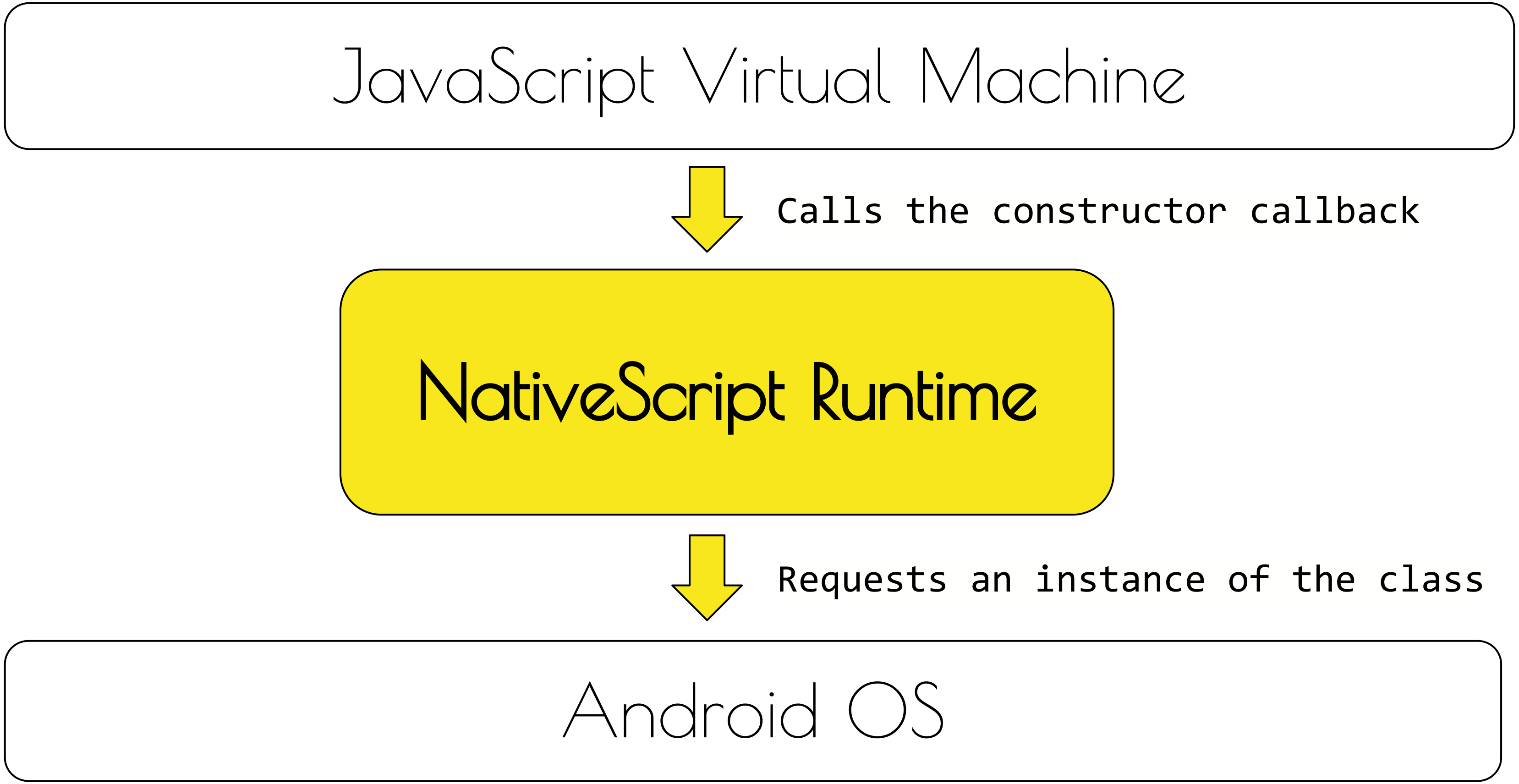
`recorder.doStuff()`

Method callback

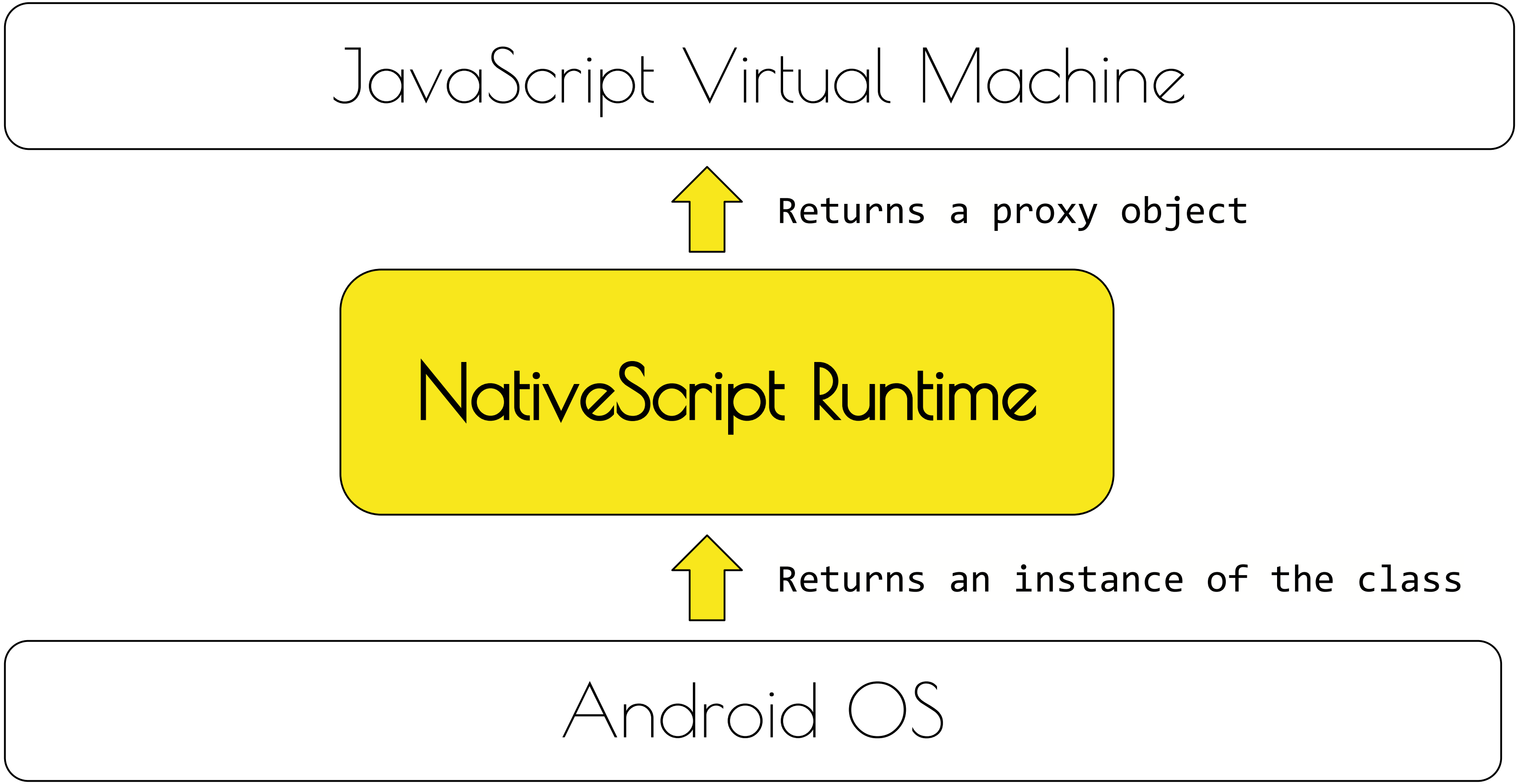
Calls the method on the Java object

The result is marshallized and returned back to the JS world

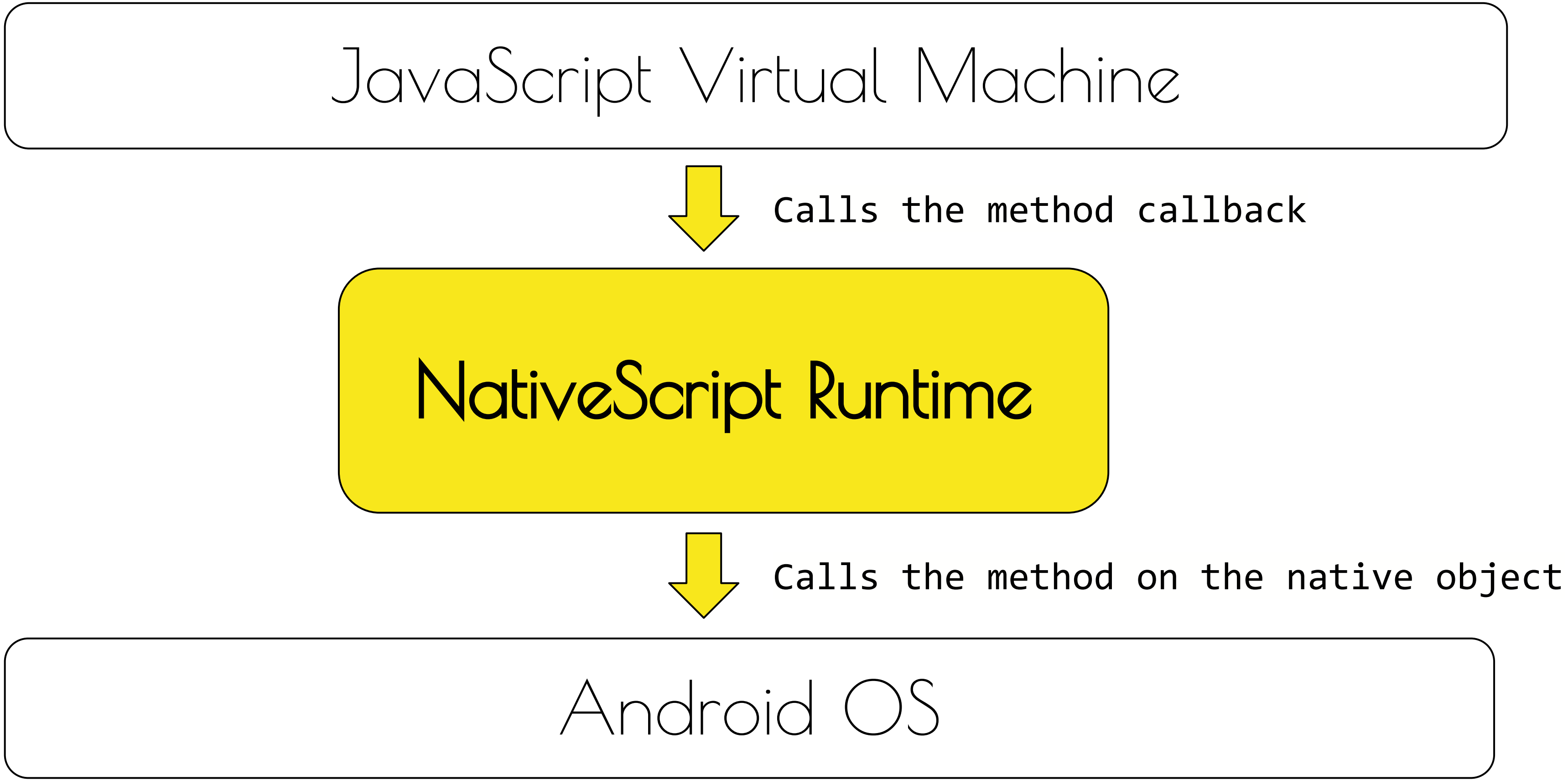
```
const recorder = new android.media.MediaRecorder();
```



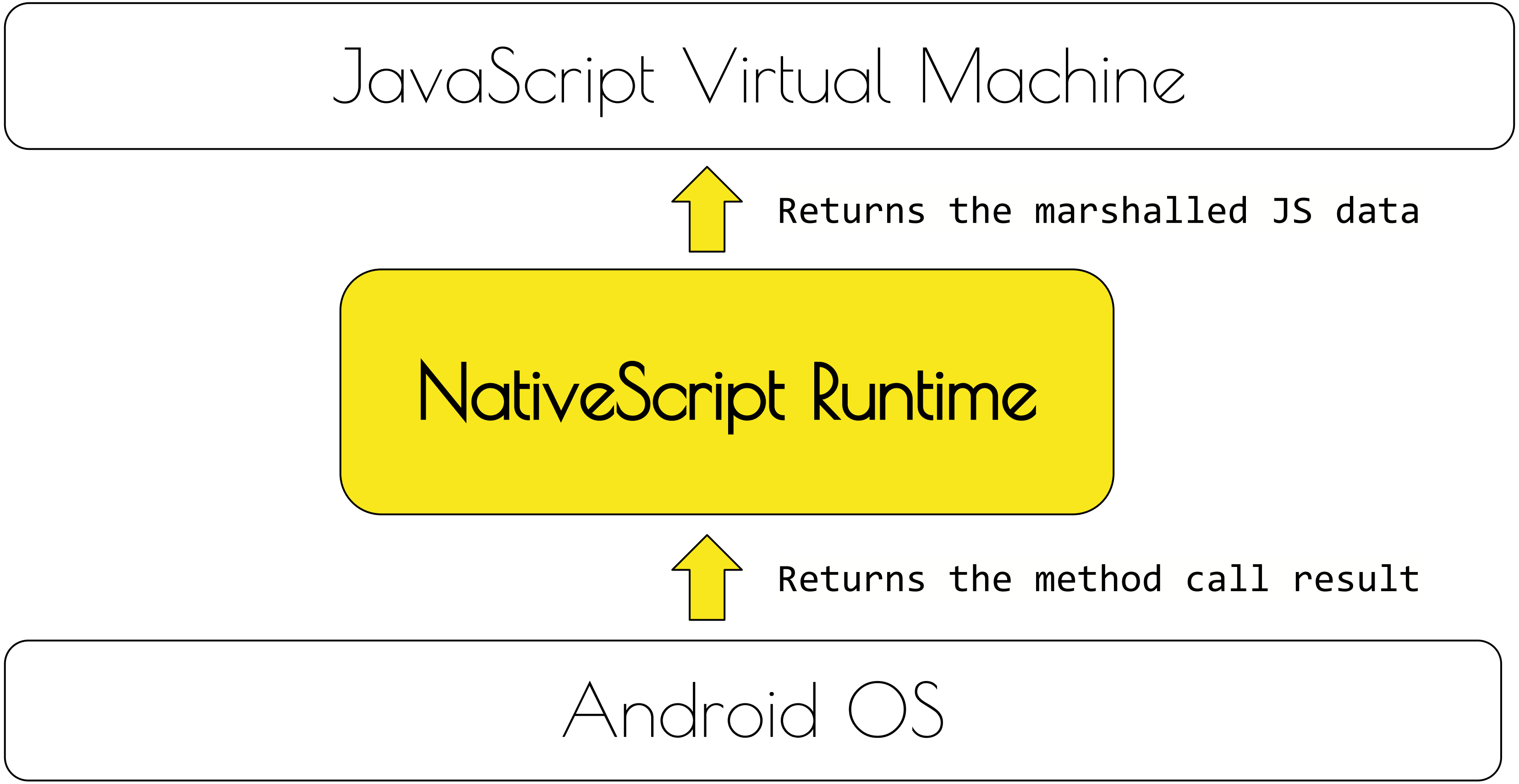
```
const recorder = new android.media.MediaRecorder();
```



```
const result = recorder.doStuff();
```



```
const result = recorder.doStuff();
```



Objects lifecycle

Part 2 / 4

Garbage collection

Retrieves the memory of unused objects

Nondeterministic nature

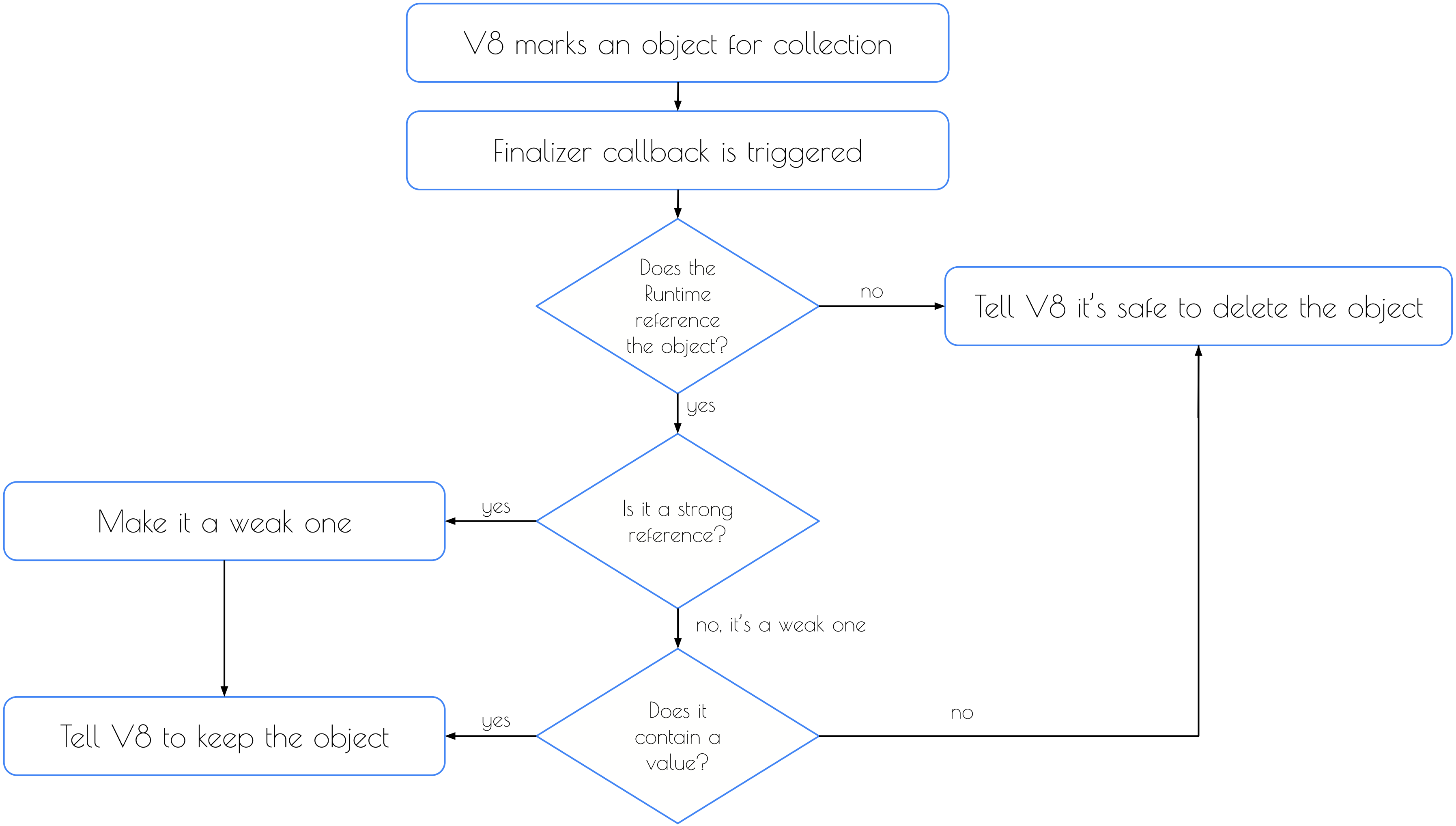
Both the Android Runtime and V8 have GC

Synchronization by the NativeScript Runtime

Ensures no object is prematurely collected

Uses V8 finalizer callbacks

Stores strong/weak references to Java objects created with JS code


V8 marks an object for collection
Finalizer callback is triggered
Does the Runtime reference the object?
no
Tell V8 it's safe to delete the object
yes
Is it a strong reference?
yes
Make it a weak one
no, it's a weak one
Does it contain a value?
yes
Tell V8 to keep the object
no

Android GC

If there is a strong reference, object is in use

If there is only a weak reference, object can be collected

Deleting an object depends on V8's GC


Challenges

Possible memory problems

The Java objects require several GC cycles to be collected

Creating big Java objects through JS may lead to *"out of memory"* exceptions

Forcing Garbage Collection

1. V8 GC
2. Android Runtime GC
3. V8 GC


releaseNativeCounterpart: fn

Multithreading

Part 3 / 4

JS in NativeScript -> Single Thread

JS in NativeScript -> Single Thread

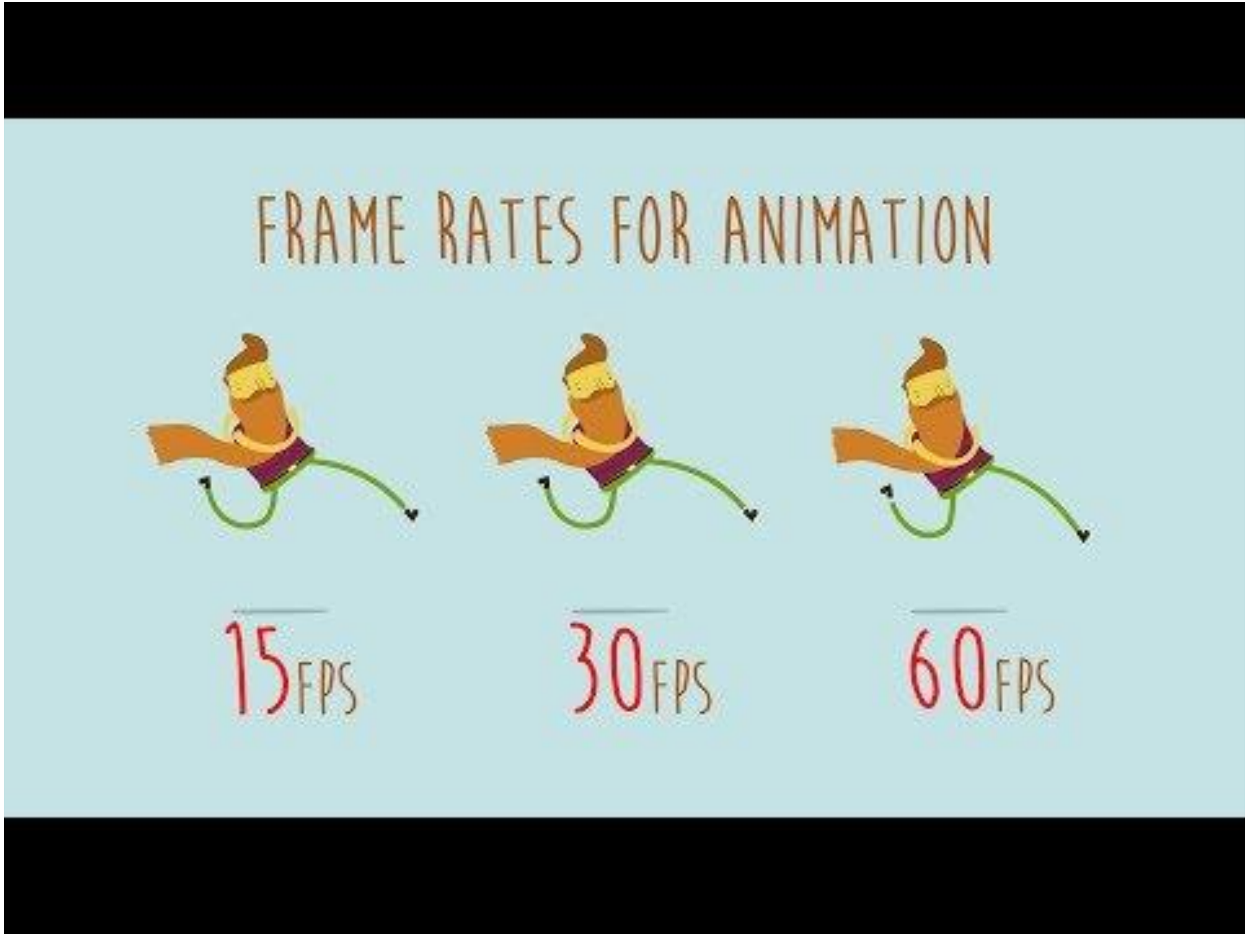
= User Interface Thread

Jank

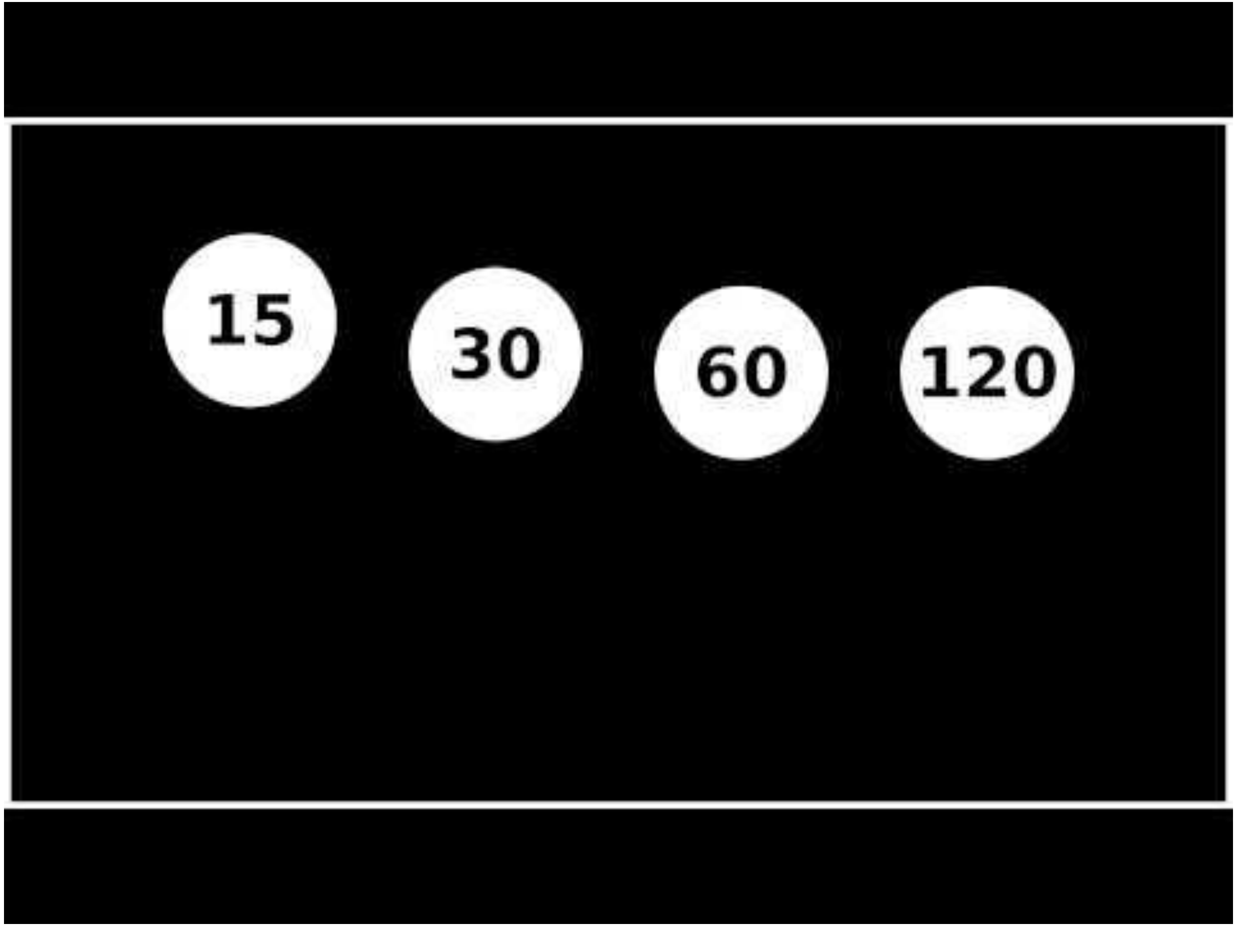
60 frames per second

1 second / 60 frames =
16.66 millisecond budget

Failing to meet the budget
==> frame rate drop



[Frame rate comparison]



[Frame rate comparison 2]

No jank

Building UI

Animations

HTTP/network requests

Jank

Executing CPU-intensive operations.

The same happens in native Android apps.

Worker threads

Background threads in the JavaScript world

Based on the web workers API

No JS memory sharing


Worker thread = ???


Theory time!

Isolate

V8's way to allocate and **isolate** memory for a code that's running.

Isolates can run in parallel.

Context

One isolate = multiple **contexts**.

No memory isolation.

Contexts can't run in parallel.


Worker thread = ???


Worker thread = Isolate

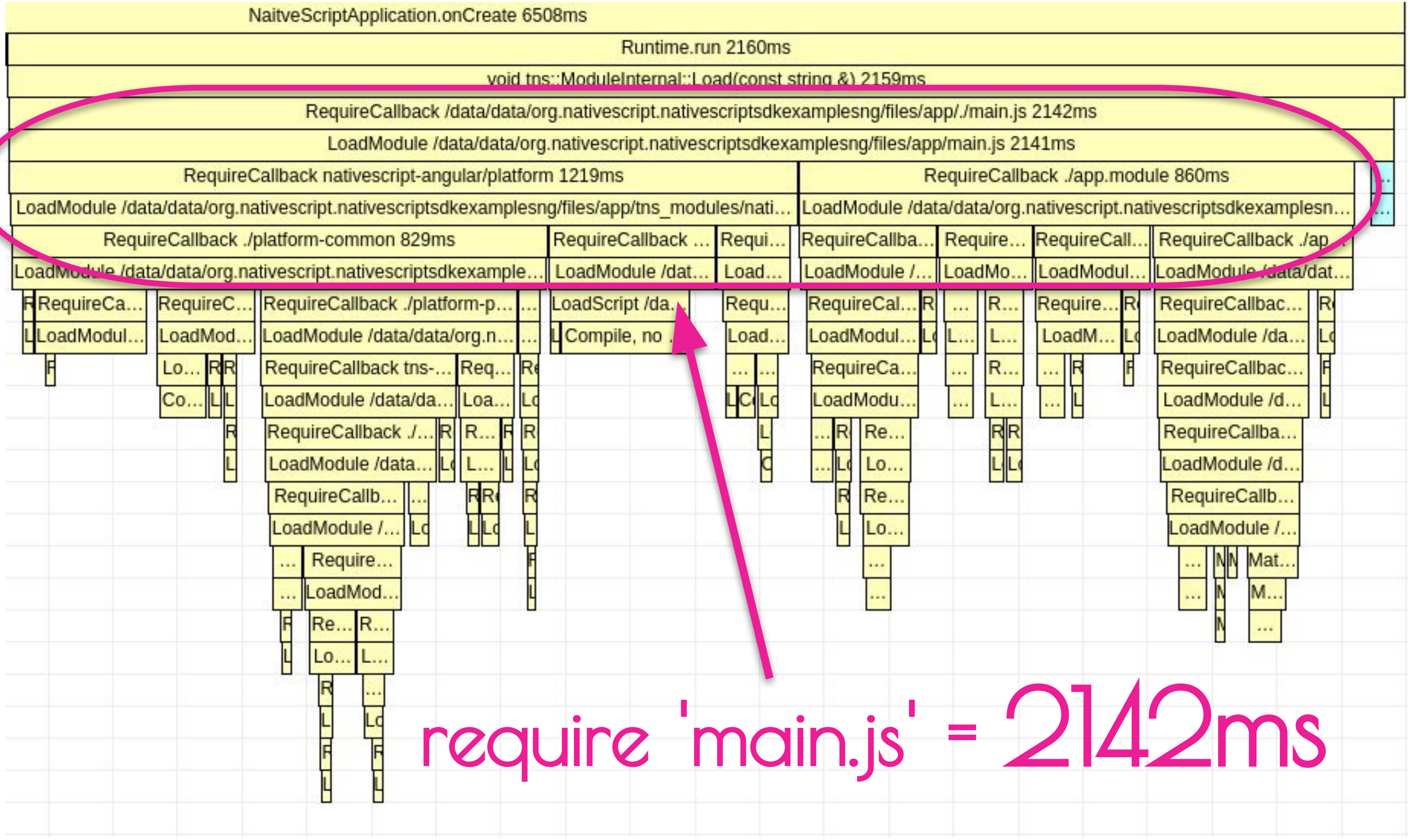
Snapshots

Part 4 / 4

Let's talk about start up time...

File System Requests

Parsing & Compiling JS

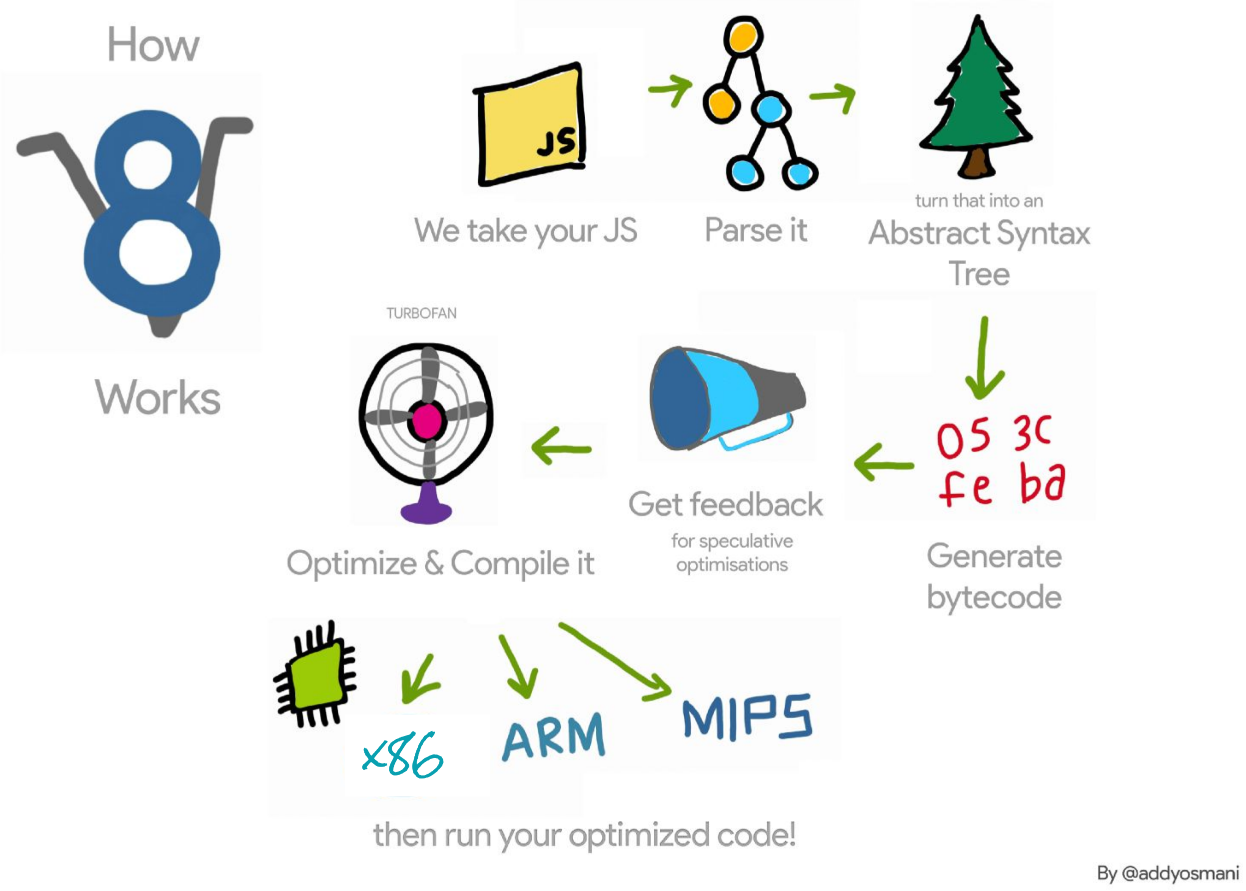

NaitveScriptApplication.onCreate 6508ms
Runtime.run 2160ms
void tns::ModuleInternal::Load(const string &) 2159ms
RequireCallback /data/data/org.nativescript.nativescriptsdkexamplesng/files/app/./main.js 2142ms
LoadModule /data/data/org.nativescript.nativescriptsdkexamplesng/files/app/main.js 2141ms
RequireCallback nativescript-angular/platform 1219ms
RequireCallback ./app.module 860ms
LoadModule /data/data/org.nativescript.nativescriptsdkexamplesng/files/app/tns_modules/nati…
LoadModule /data/data/org.nativescript.nativescriptsdkexamplesn…
RequireCallback ./platform-common 829ms
LoadModule /data/data/org.nativescript.nativescriptsdkexample…
require 'main.js' = 2142ms

Bundled app =

fewer FS requests =

Faster launch time

What about 'Parse & Compile'?



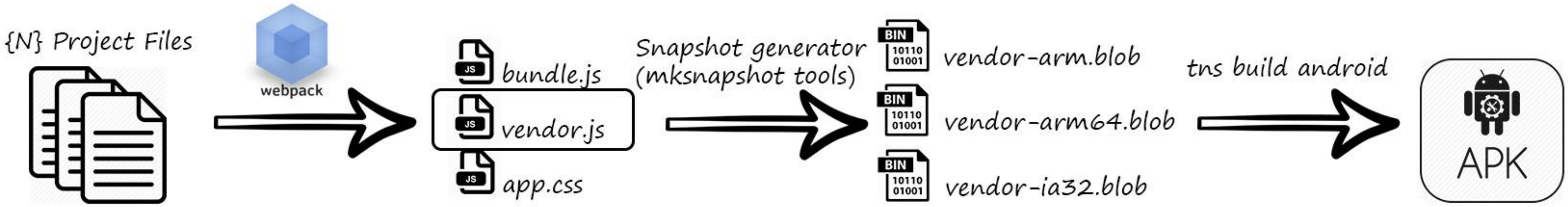
JavaScript Start-up Performance by Addy Osmani

We must load the JS

at some point...


Custom startup snapshots!

Creating custom snapshots



[Snapshots in NativeScript]

[Snapshots in Atom]

Loading snapshots

1. Load the snapshot binary
2. Set up the parameters for the new isolate
3. Create the new isolate

--> The context in the isolate will be a copy of the context in the snapshot.

Limitations

Bare context

-> no native APIs

-> no *require*

3rd party-code

Wrapping native API access

```
// Creating a snapshot throws an error.
// ReferenceError: android is not defined

const version =
      android.os.Build.VERSION.SDK_INT;

function doStuff() {
   console.log(version);
   ...
}
```

```
// Creating a snapshot works.
// The native getter is not evaluated immediately.

const getVersion = () =>
      android.os.Build.VERSION.SDK_INT;

function doStuff() {
   const version = getVersion();
   console.log(version);
   ...
}
```


Be lazy.


@StanimiraVlaeva
@VladimirMutafov