

Scala + Android

Nick (<http://github.com/stanch>)

ScalaDays 2014

Godspeed from Apple



Swift

@SwiftDevs



Now that #Swift is out, our #Android friends should get a modern alternative to #Java too!

Reply Retweet Favorite More

RETWEETS

78

FAVORITES

36



11:01 AM - 3 Jun 2014

Why is *mobile* hard?

- Variety of devices
- Lots of interaction
- On-off connectivity
- Ancient API design

Android 101

- Very popular
- Obsessed with candy



- *Hard*

Scala as a remedy

- Works on Android
- Solves problems
- Is fun

Elegance vs ancient APIs

- Ubiquitous null
 ⇒ use Option
- Accessing things “too early” clashes with lifecycle
 ⇒ use lazy
- Listener classes everywhere
 ⇒ use SAM (*Scala 2.11 -Xperimental*)

More Elegance: Scaloid

```
val button = new Button(context)
button.setText("Greet")
button.setOnClickListener(new OnClickListener() {
    def onClick(v: View) {
        Toast.makeText(context, "Hello!", Toast.LENGTH_SHORT).show()
    }
})
layout.addView(button)
```



```
SButton("Greet", toast("Hello!"))
```

<https://github.com/pocorall/scaloid>

Concurrency: UI thread

- UI code **only** on the UI thread
- non-UI code **only** outside UI thread

How to do the work on background and show the result to the user?

Concurrency: Futures

```
import macroid.UiThreading._
```

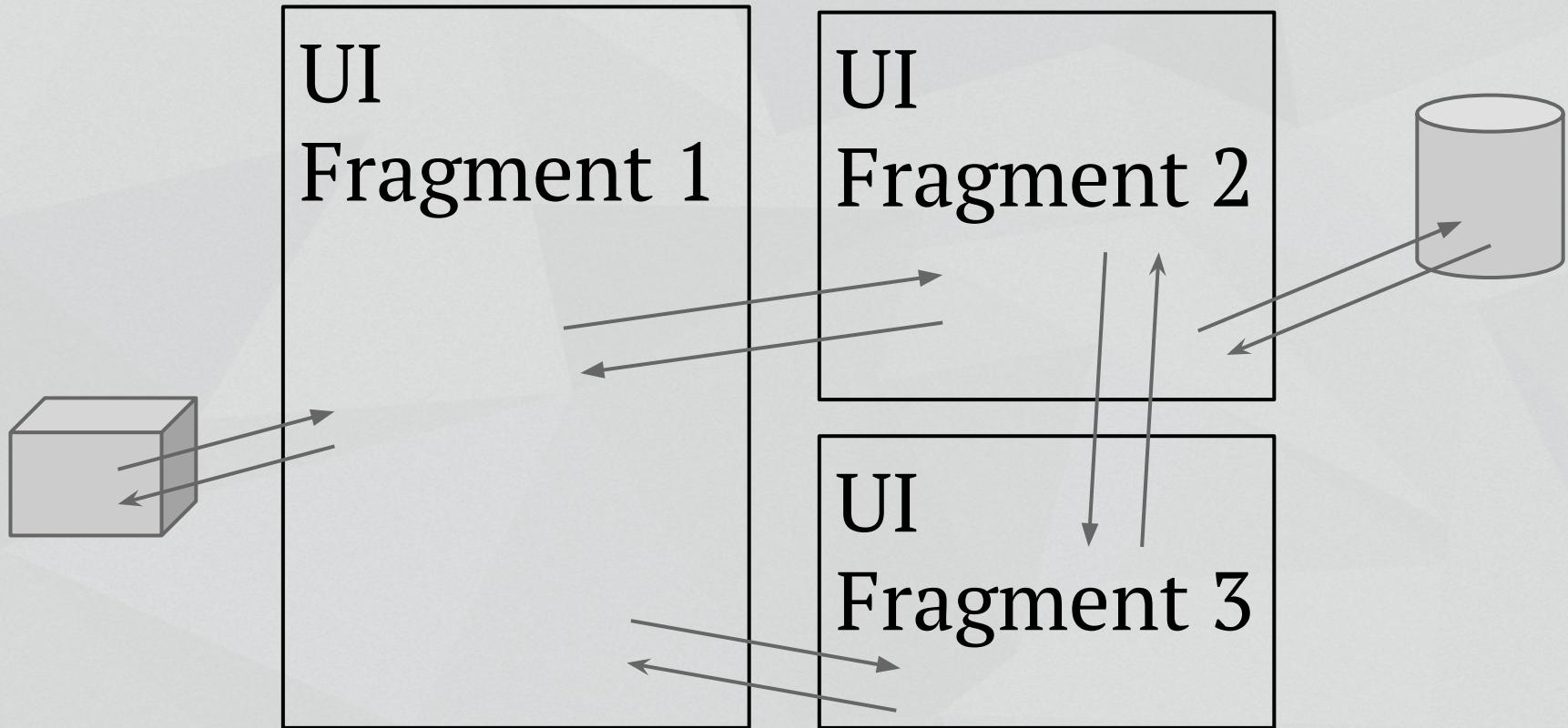
```
Future {  
    // background  
    ...  
} onSuccessUi {  
    case ... ⇒  
        // UI  
        ...  
}
```

Concurrency: scala-async

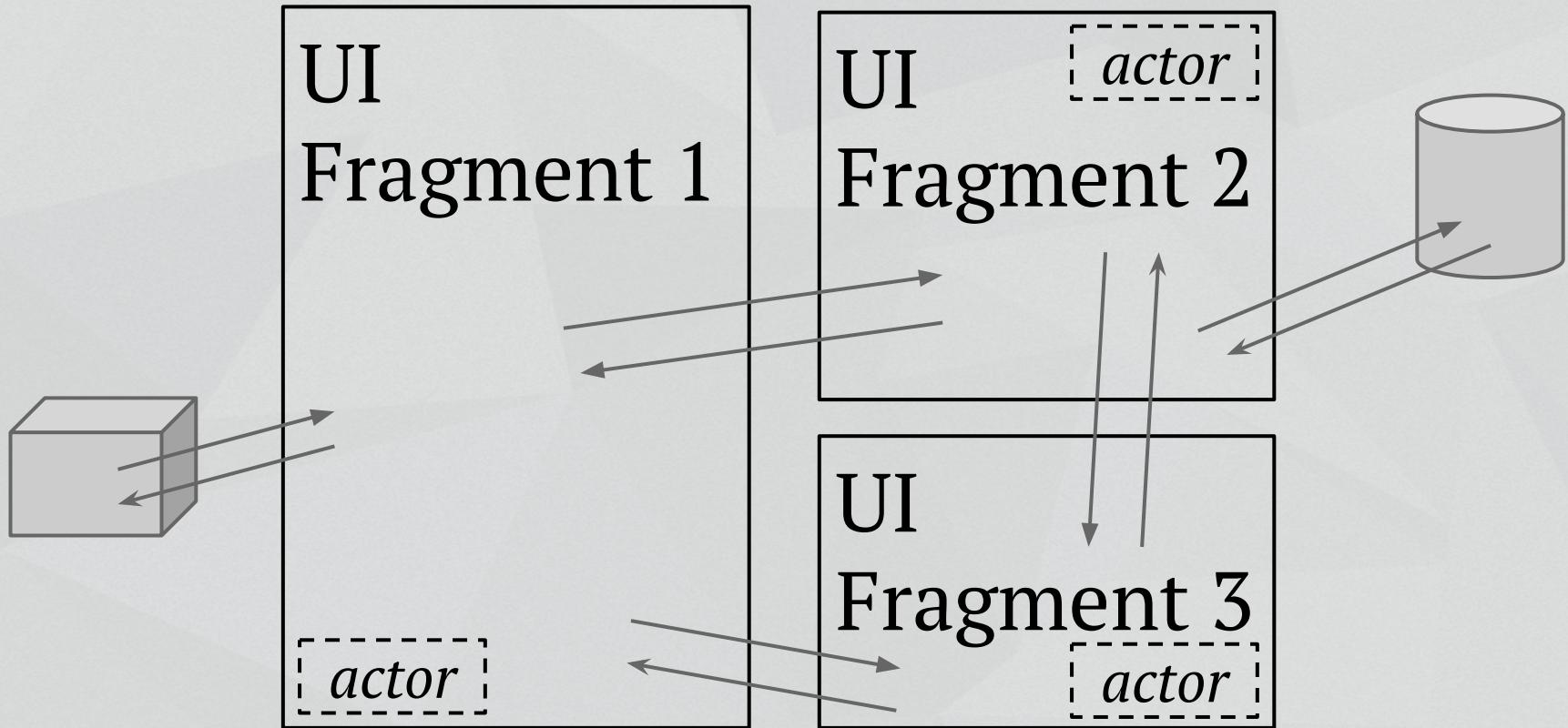
```
import macroid.U.Threading._

async {
    val x = await(future1)
    val y = await(future2)
    x * y
} onSuccessUi {
    case z =>
        showToUser(z)
}
```

Concurrency: Akka



Concurrency: Akka



Concurrency: resolvable

- REST endpoints
- Local database
- Local caches
- On/off connectivity

How to load data from several endpoints in the most flexible and optimal way?

<http://resolvable.github.io>

Macroid

*Experimental modular functional UI
language for Android, written in Scala.*

Macroid: motivation

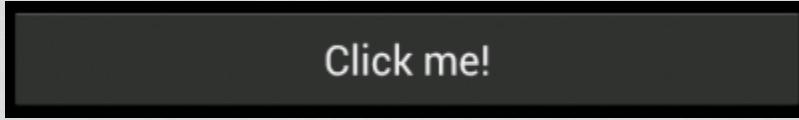
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

- Inflexible
- Poor composability

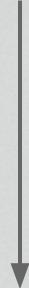
Macroid: bricks

Click me!

Macroid: bricks



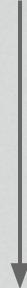
Click me!



```
new Button(ctx)
```

Macroid: bricks

Click me!



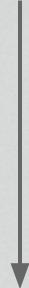
w[Button]

a brick

Macroid: bricks

Howdy

Click me!



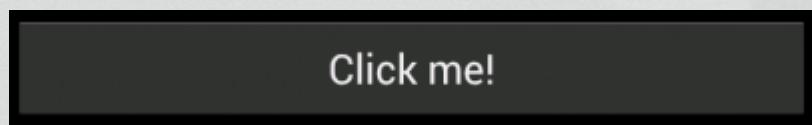
```
l[LinearLayout](  
    w[TextView],  
    w[Button]  
)
```

Macroid: brick *composition*

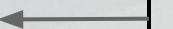
```
val layout1 = ...
val layout2 = ...

l[LinearLayout](
    layout1,
    layout2
)
```

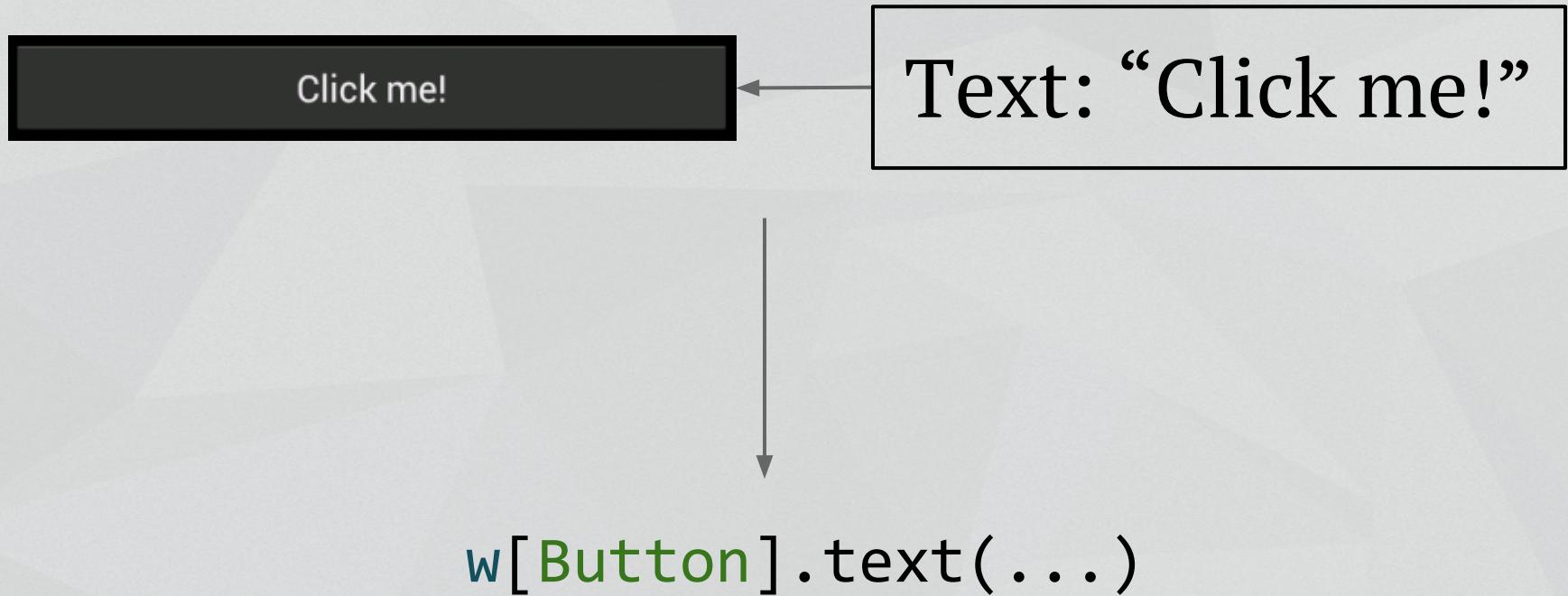
Macroid: tweaks



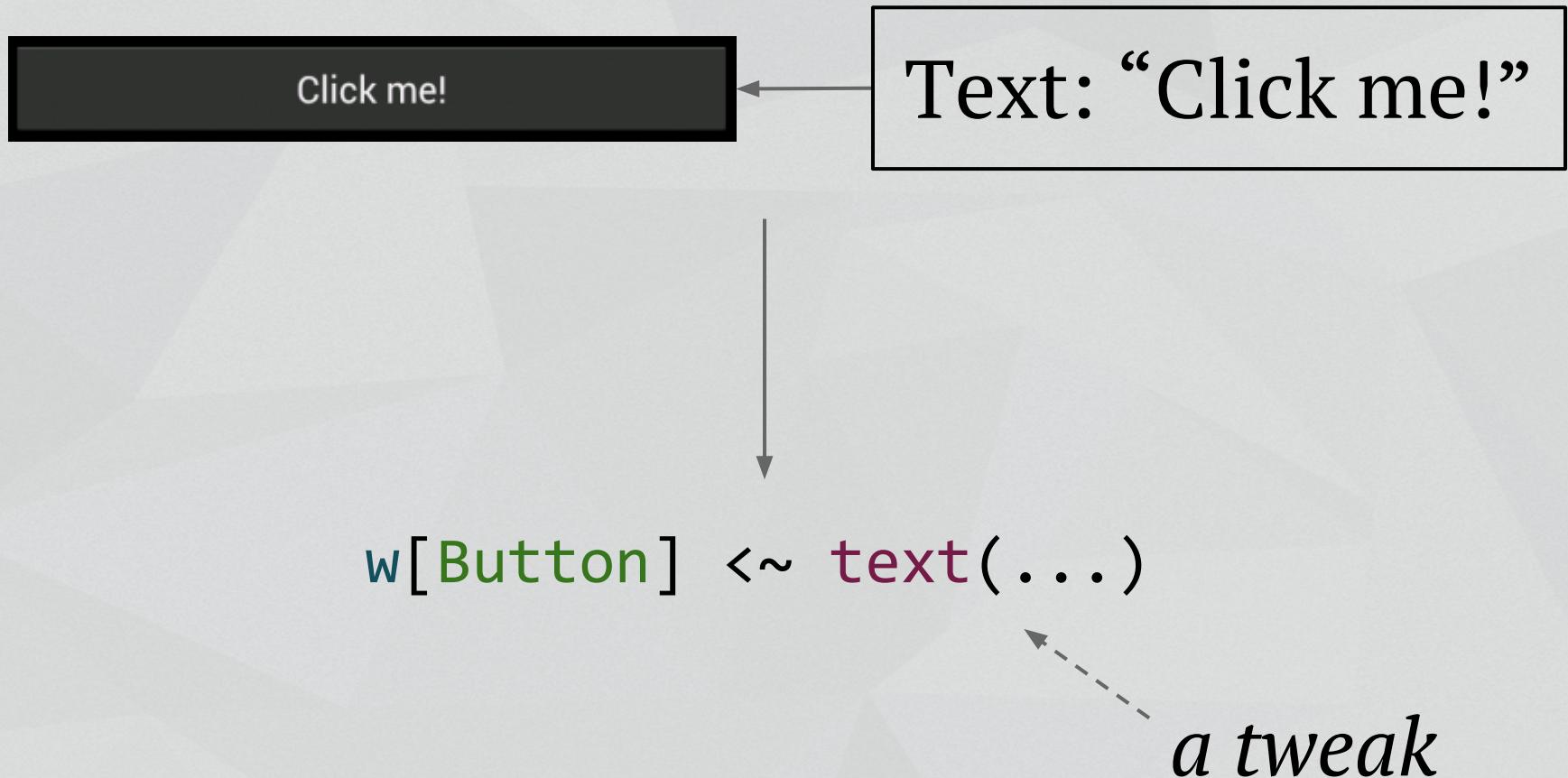
Text: “Click me!”



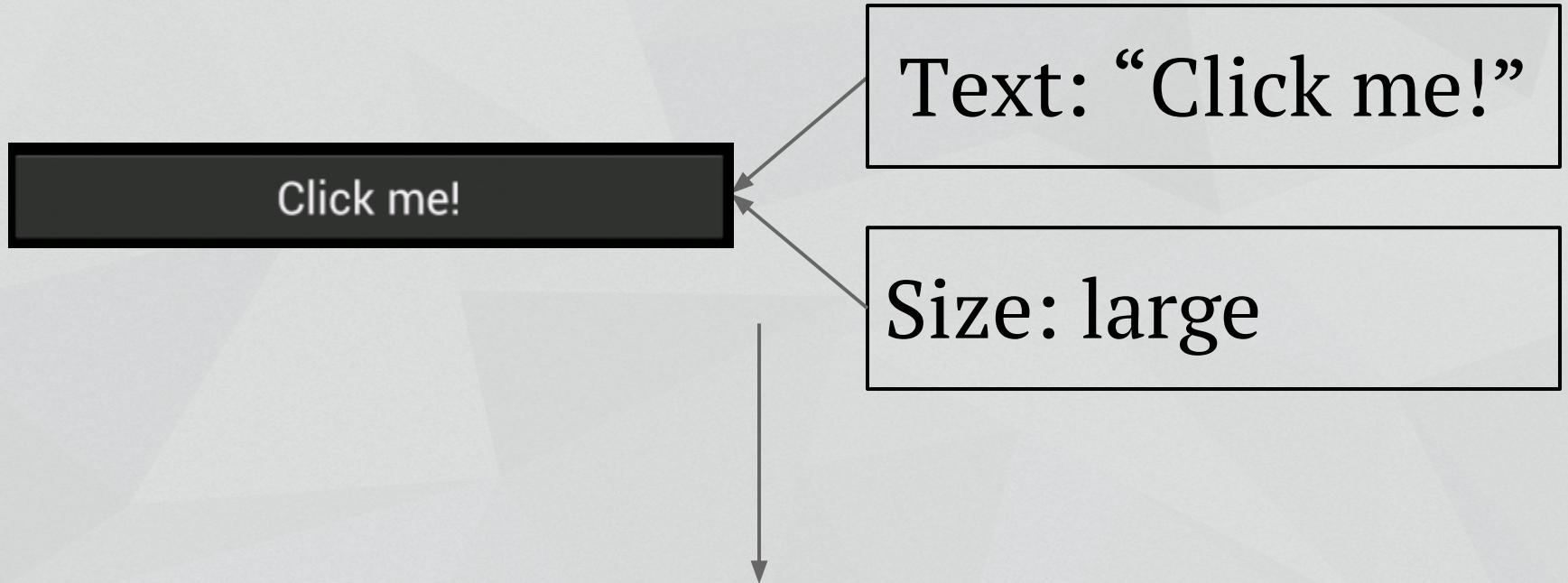
Macroid: tweaks



Macroid: tweaks



Macroid: tweaks



```
w[Button] <~  
text(...) <~  
TextSize.large
```

Macroid: tweak *composition*

```
def largeText(str: String) =  
    text(str) +  
    TextSize.large +  
    padding(left = 8 dp)
```

```
w[Button] <~ largeText(...)
```

Macroid: an example

```
var greeting = slot[TextView]
```

```
l[LinearLayout] (   
    w[Button] <~  
        text("Click me!") <~  
        On.click {  
            greeting <~ show  
        },
```

```
    w[TextView] <~  
        text("Howdy") <~  
        wire(greeting) <~ hide  
)
```

Macroid: Zen tweaking

? <~ ?

Button

Tweak[Button]

Macroid: Zen tweaking

? <~ ?

Button,
List[Button]

Tweak[Button],
List[Tweak[Button]]

Macroid: Zen tweaking

? <~ ?

Button,
List[Button],
Option[Button],
...

Tweak[Button],
List[Tweak[Button]],
Option[Tweak[Button]],
Future[Tweak[Button]],
...

Macroid: Zen tweaking

```
trait CanTweak[W, T, R] {  
    def tweak(w: W, t: T): Ui[R]  
}
```

Macroid: Zen tweaking

```
// will be ready in several minutes
val caption: Future[String] = Future {
    ...
}

// use right away
myTextView <~ caption.map(text)
```

Macroid: Zen tweaking

// create a reactive variable

```
val caption = rx.Var("Ola")
```

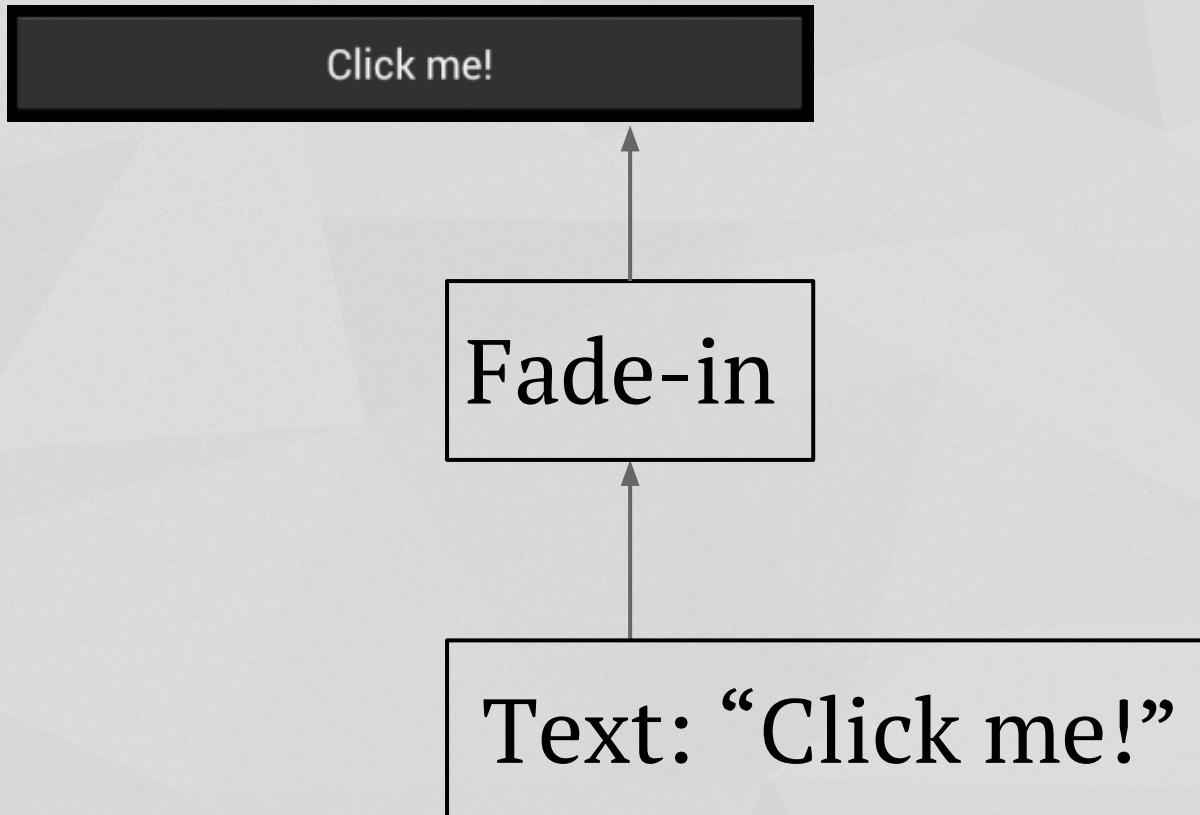
// set text to “Ola”

```
myTextView <~ caption.map(text)
```

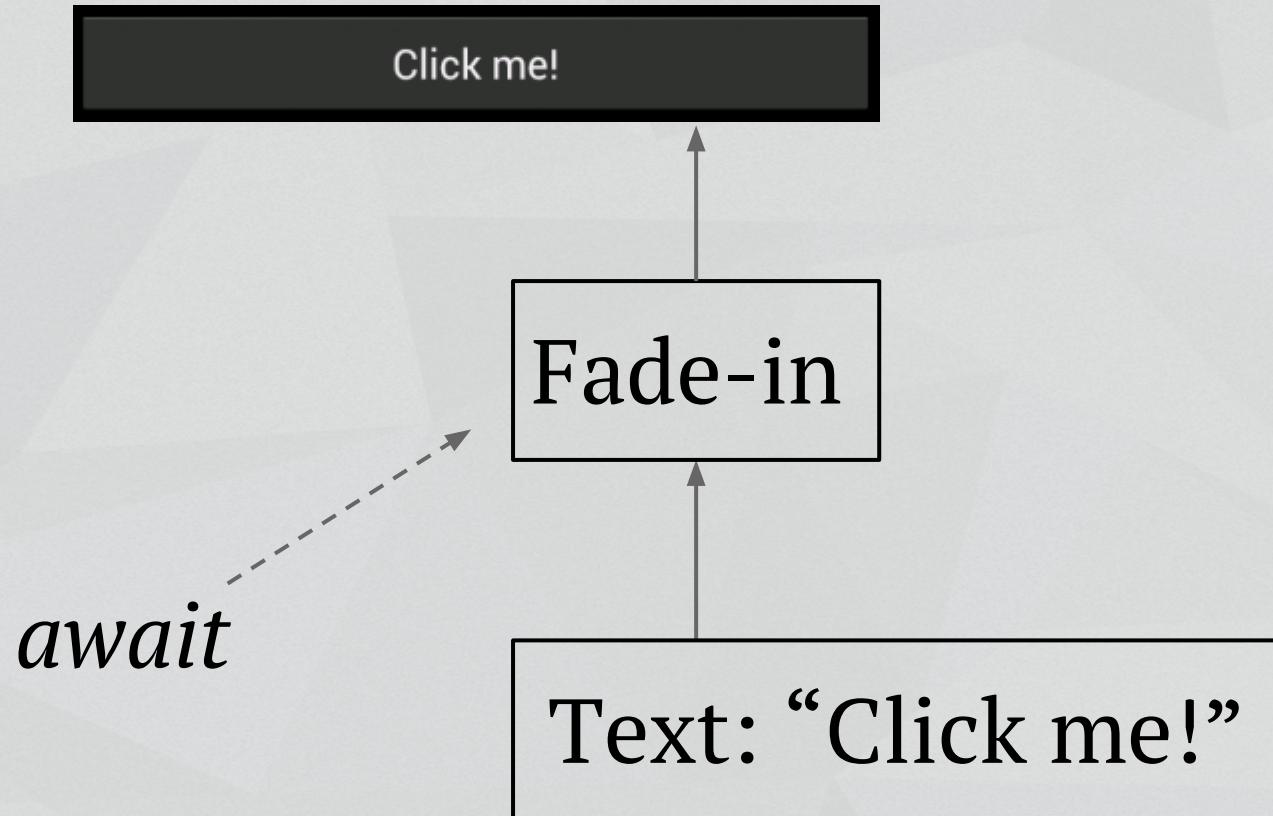
// text automatically updates to “Adeus”

```
caption() = "Adeus"
```

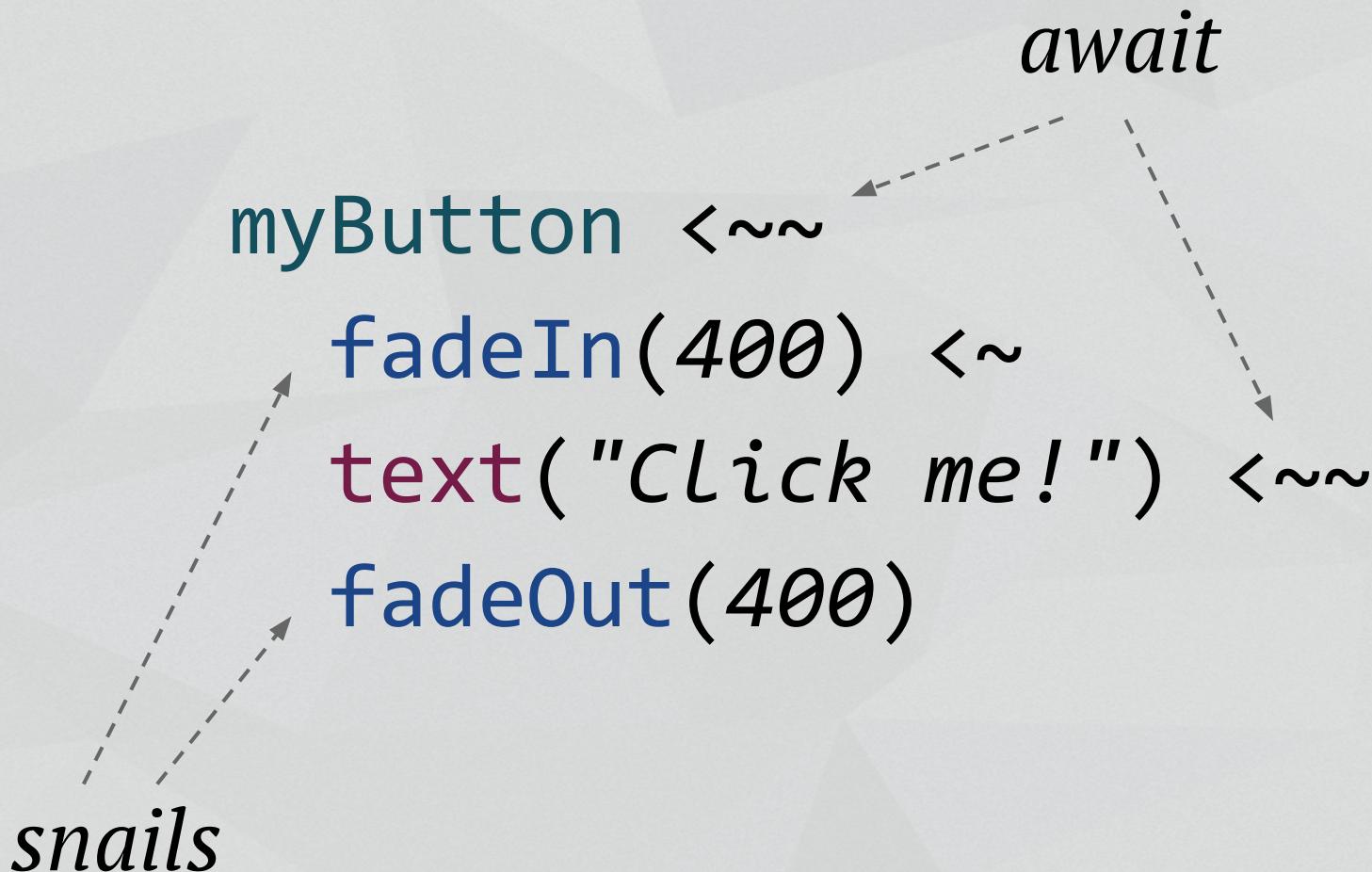
Macroid: snails



Macroid: snails

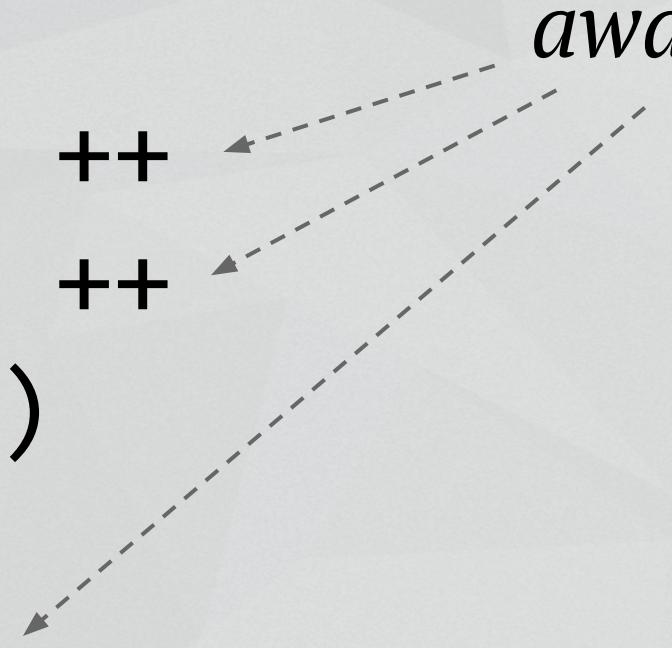


Macroid: snails



Macroid: snail *composition*

```
val blink =  
    fadeIn(400) ++  
    delay(2000) ++  
    fadeOut(400)  
  
myTextView <~~ blink
```



The diagram illustrates the composition of a sequence of events represented by arrows pointing upwards and to the right, forming a snail-like shape. Three dashed arrows originate from the text 'myTextView <~~ blink' at the bottom left. The first arrow points to the 'delay(2000)' event in the code. The second arrow points to the '++' operator after 'delay(2000)'. The third arrow points to the 'await' keyword at the end of the code.

Macroid: UI actions

```
val action =  
    myTextView <~  
        text("Howdy") <~ show  
  
runUi(action)
```

Macroid: UI *composition*

```
val action1 =  
    myTextView <~  
        text("Howdy") <~ show
```

```
val action2 =  
    myProgressBar <~ hide
```

```
runUi(action1 ~ action2)
```

Macroid: UI *composition*

```
runUi {  
    (myProgressBar <~~ fadeIn(400)) ~~  
    (myTextView <~~ blink) ~~  
    (myOtherTextView <~ text("’SUP?"))  
}
```

Macroid: UI *composition*

```
runUi {  
    (myProgressBar <~~ fadeIn(400)) ~~  
    (myTextView <~~ blink) ~~  
    (myOtherTextView <~ text("'SUP?"))  
}
```

await

Macroid: media queries

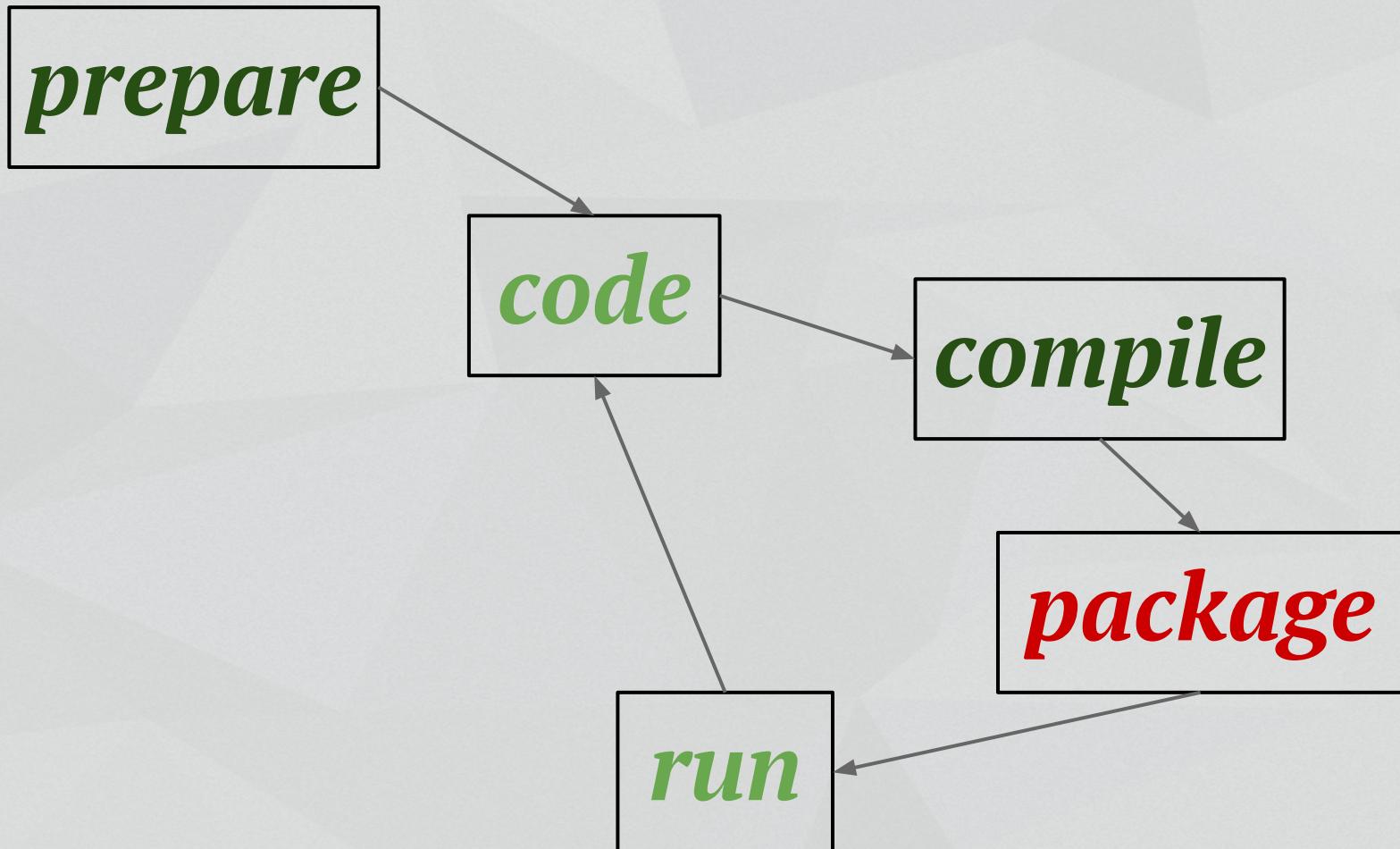
```
def adaptiveSize =  
    maxWidth(300 dp) ?  
        TextSize.large |  
        TextSize.medium
```

*a media
query*

```
myTextView <~ adaptiveSize
```

But how do we actually...

Android 201

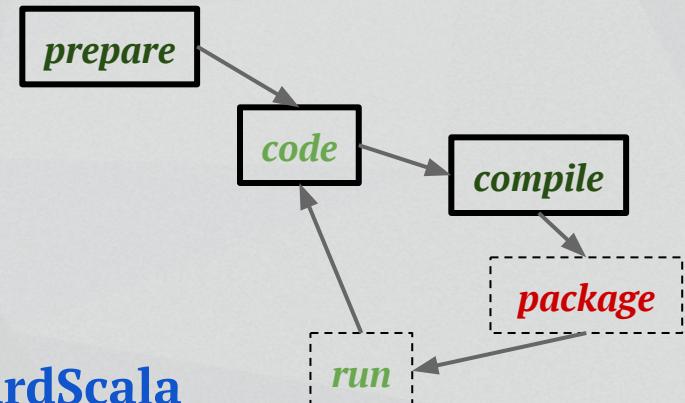


The build tool: none

AKA “*Just use the IDE*”

IntelliJ IDEA, ScalaIDE*

- + Works (almost) out of the box
- No dependency management
- Tied to a specific IDE

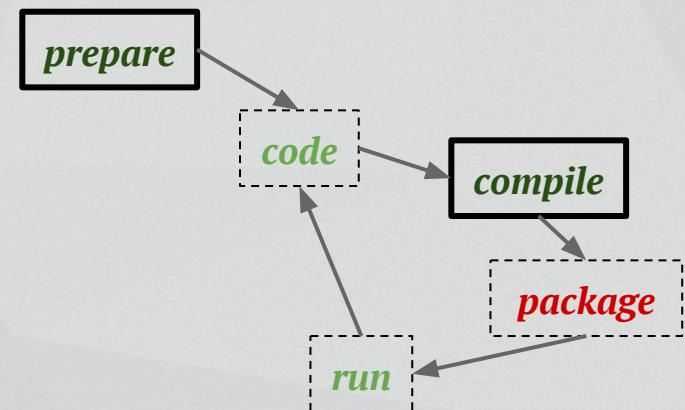


*<https://github.com/banshee/AndroidProguardScala>

The build tool: SBT

<https://github.com/pfn/android-sdk-plugin>

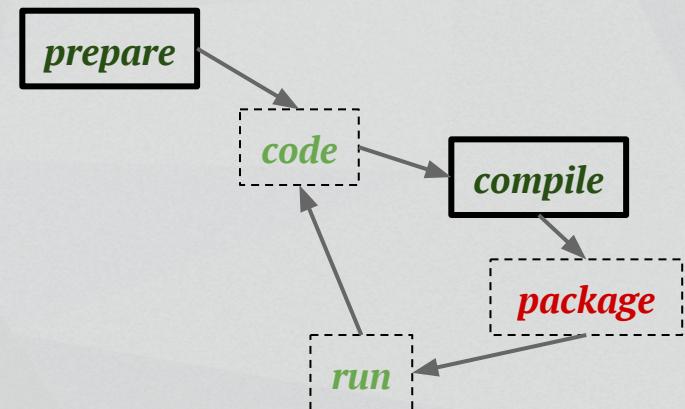
- + Generate project files for IDEs
- + Dependency management
- + Build automation
- (?) Need to learn SBT



The build tool: Gradle

<https://github.com/saturday06/gradle-android-scala-plugin>

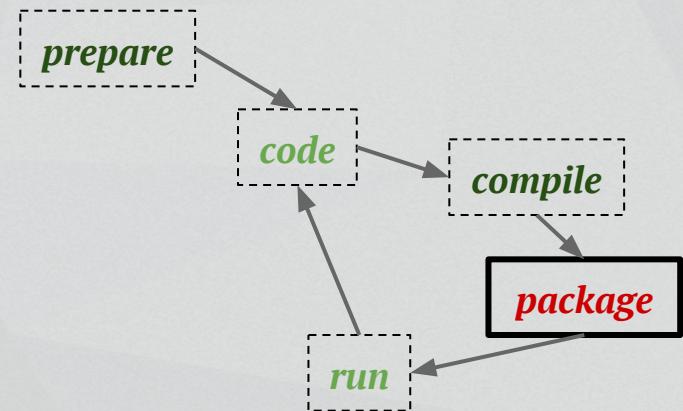
- + Generate project files for IDEs
- + Dependency management
- + Build automation
- (?) Need to learn Gradle



The pain

AKA *Dalvik VM*

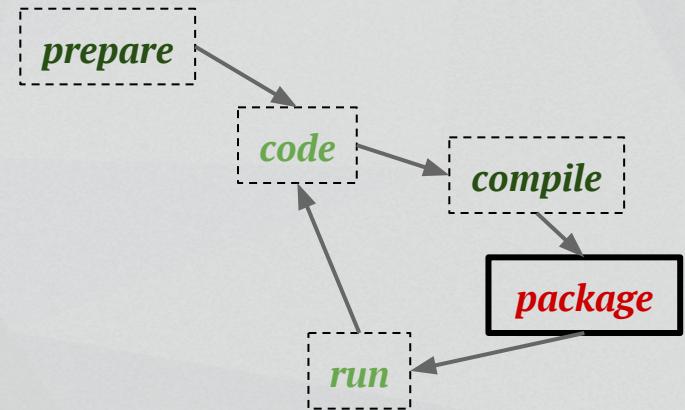
- 65K global method limit
- *slow* conversion from JARs



The pain



from 20s to 2min



Mailing list:

[https://groups.google.com/forum/
#!forum/scala-on-android](https://groups.google.com/forum/#!forum/scala-on-android)

This presentation:

<http://macroid.github.io/Talks.html>