goto;

GOTO CHICAGO 2023

Composing All The Things

#GOTOchgo

Hello There

garth.gilmour@jetbrains.com
@GarthGilmour

Fact Finding

- How many of you:
  - Develop on the JVM or Mobile?
  - Build User Interfaces for a living?

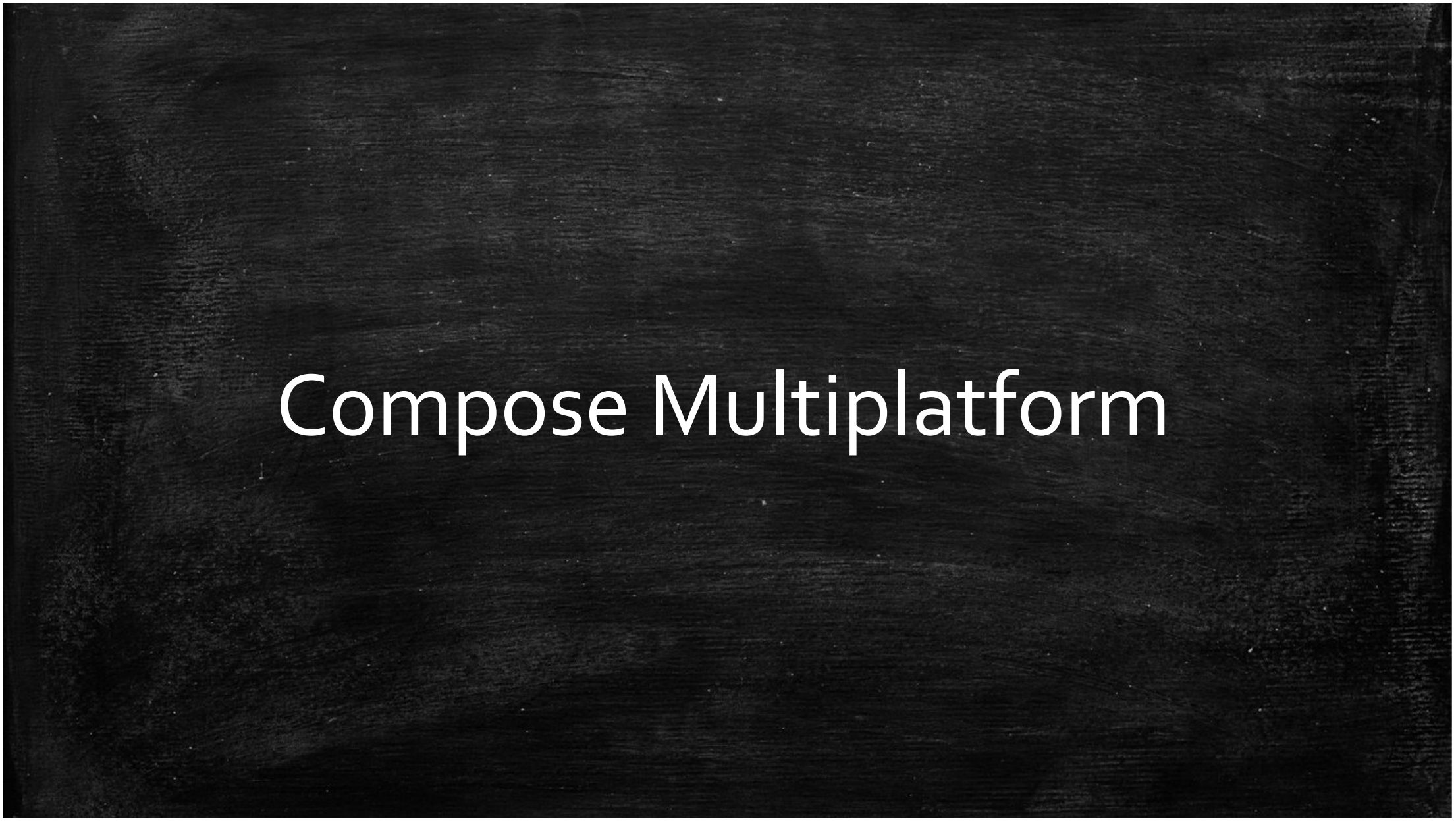
About This Talk

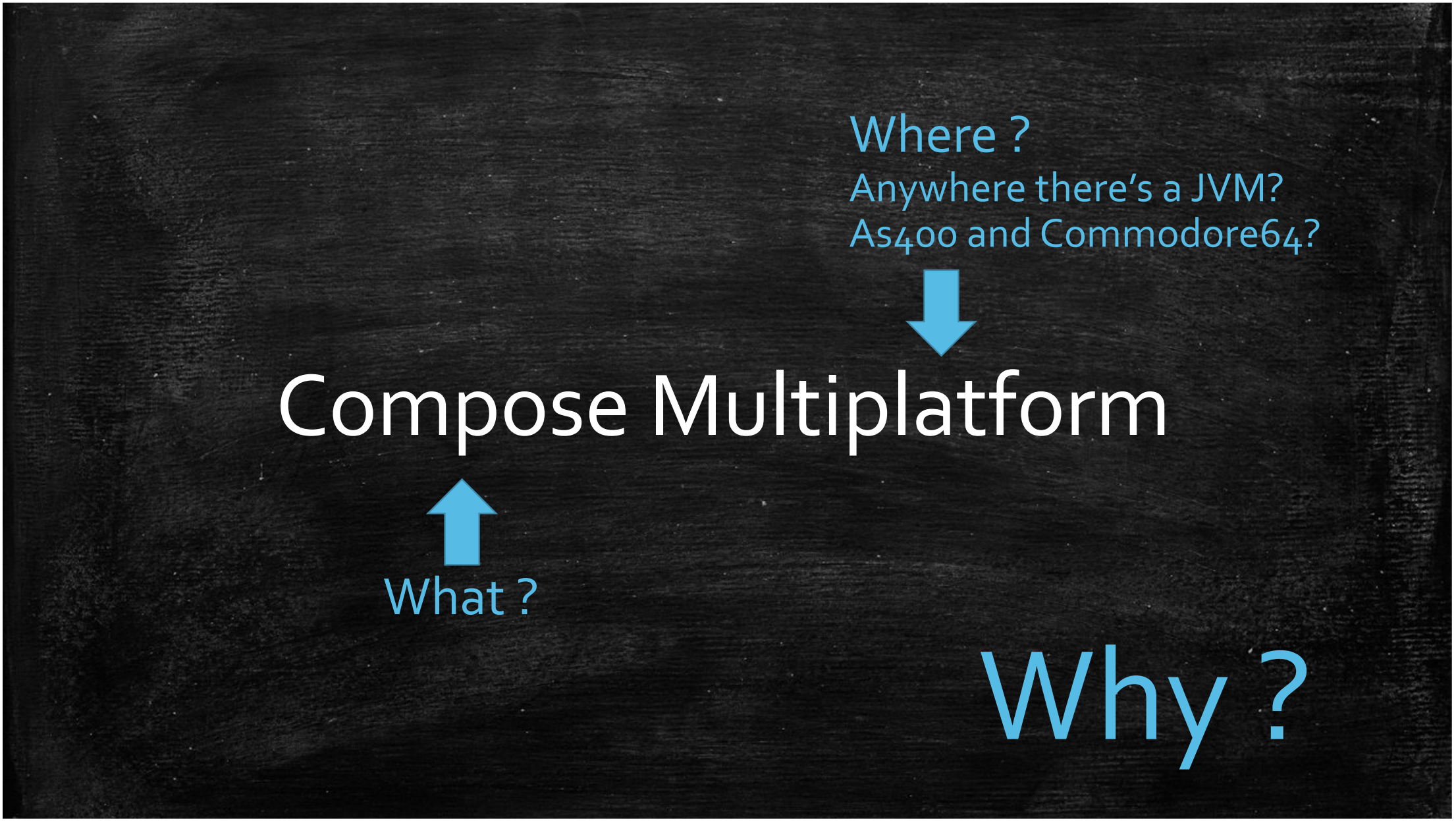
- This talk has 3 parts
  - What, where and why
  - Show me the code!
  - Alternative approaches
- There will be lots of code and links
  - All available at the repo below
  - These slides will be made available

https://github.com/garthgilmourni/composing-all-goto-2023

Part 1

What, Where and Why


Compose Multiplatform


Where ?
Anywhere there's a JVM?
As400 and Commodore64?
Compose Multiplatform
What ?
Why ?

The TLDR On Compose

- Compose lets you create awesome UIs
- Compose Multiplatform is produced by JetBrains
- Built on Kotlin Multiplatform by JetBrains
- Powered by Jetpack Compose from Google

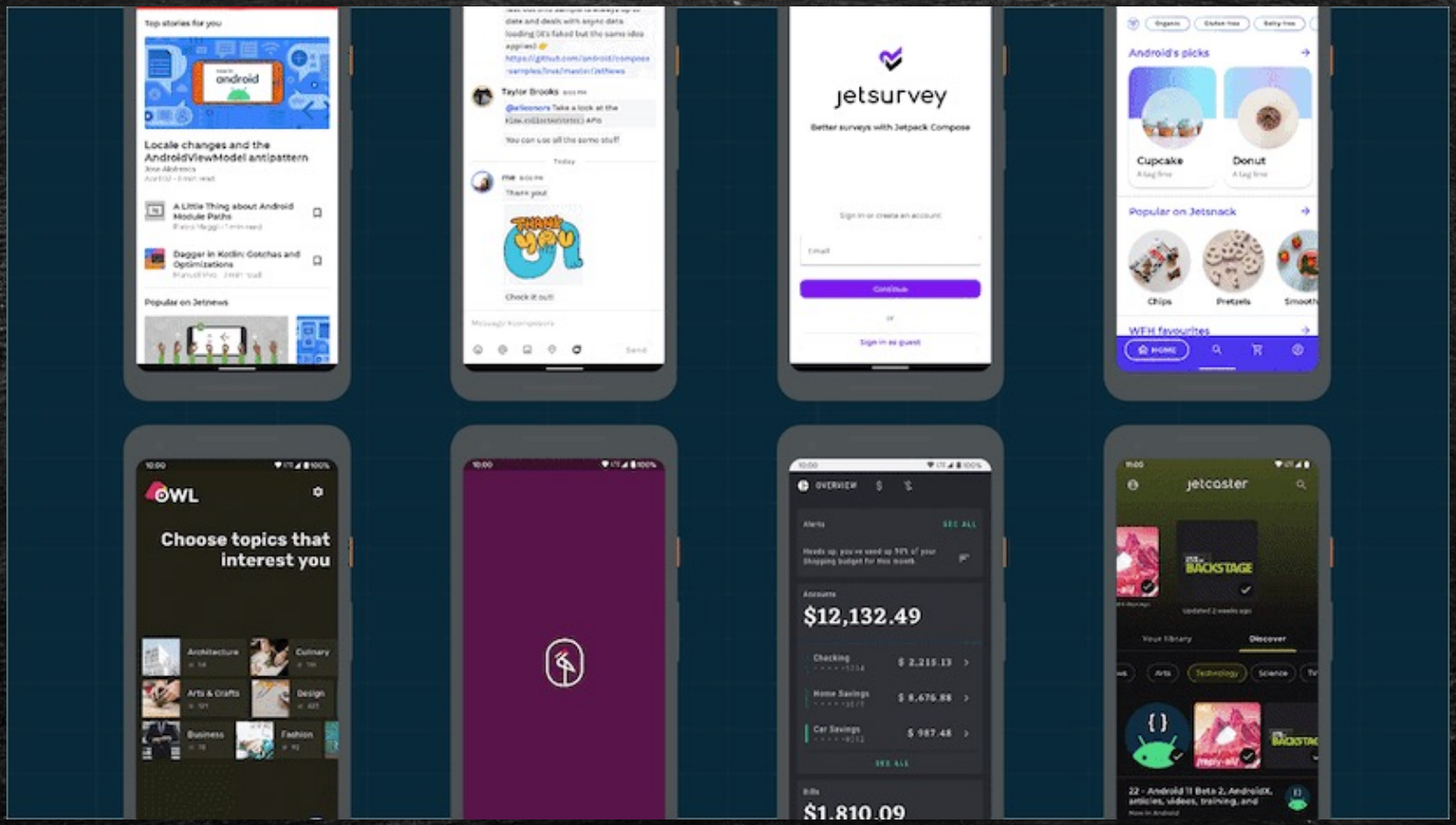

Jordy.app
@Jordy_vD_
I may have finally found an app framework that doesn't completely suck at desktop apps. It's actually quite good so far;
@jetbrains compose Multiplatform.
The code reads nice and it's performant so far.
Just gotta learn an entire new architecture and programming language
3:26 · 06 May 23 · 1,160 Views

Build better apps faster with Jetpack Compose

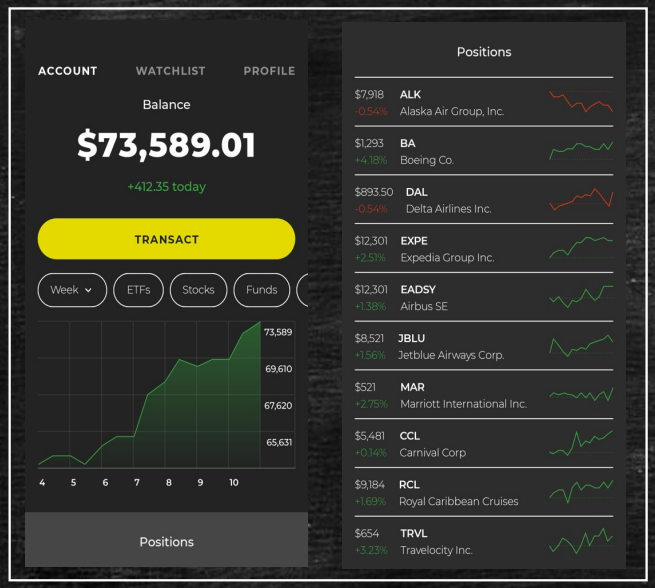
Jetpack Compose is Android's recommended modern toolkit for building native UI. It simplifies and accelerates UI development on Android. Quickly bring your app to life with less code, powerful tools, and intuitive Kotlin APIs.

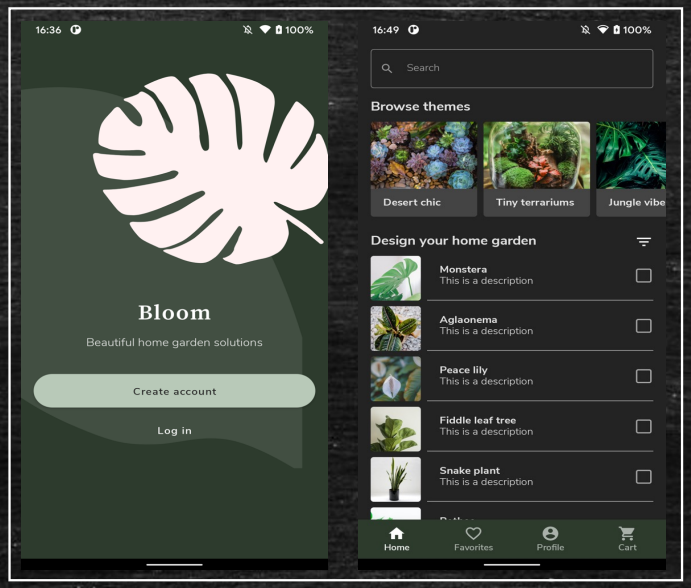
```
@Composable
fun JetpackCompose() {
    Card {
        var expanded by remember { mutableStateOf(false) }
        Column(Modifier.clickable { expanded = !expanded }) {
            Image(painterResource(R.drawable.jetpack_compose))
            AnimatedVisibility(expanded) {
                Text(
                    text = "Jetpack Compose",
                    style = MaterialTheme.typography.bodyLarge,
                )
            }
        }
    }
}
```

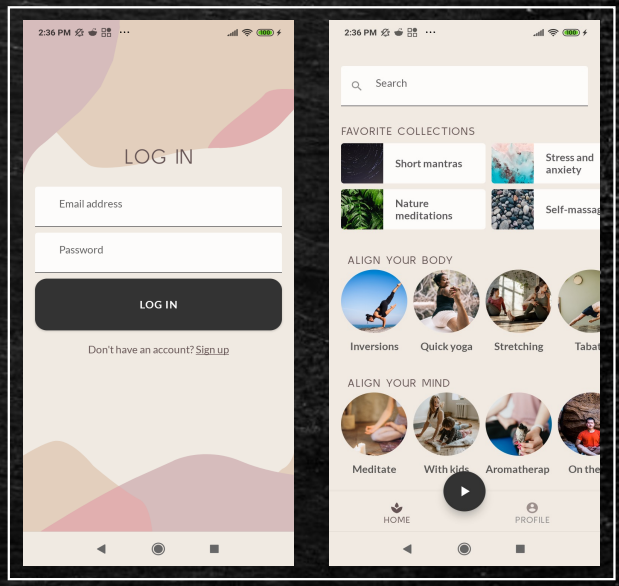
https://developer.android.com/jetpack/compose



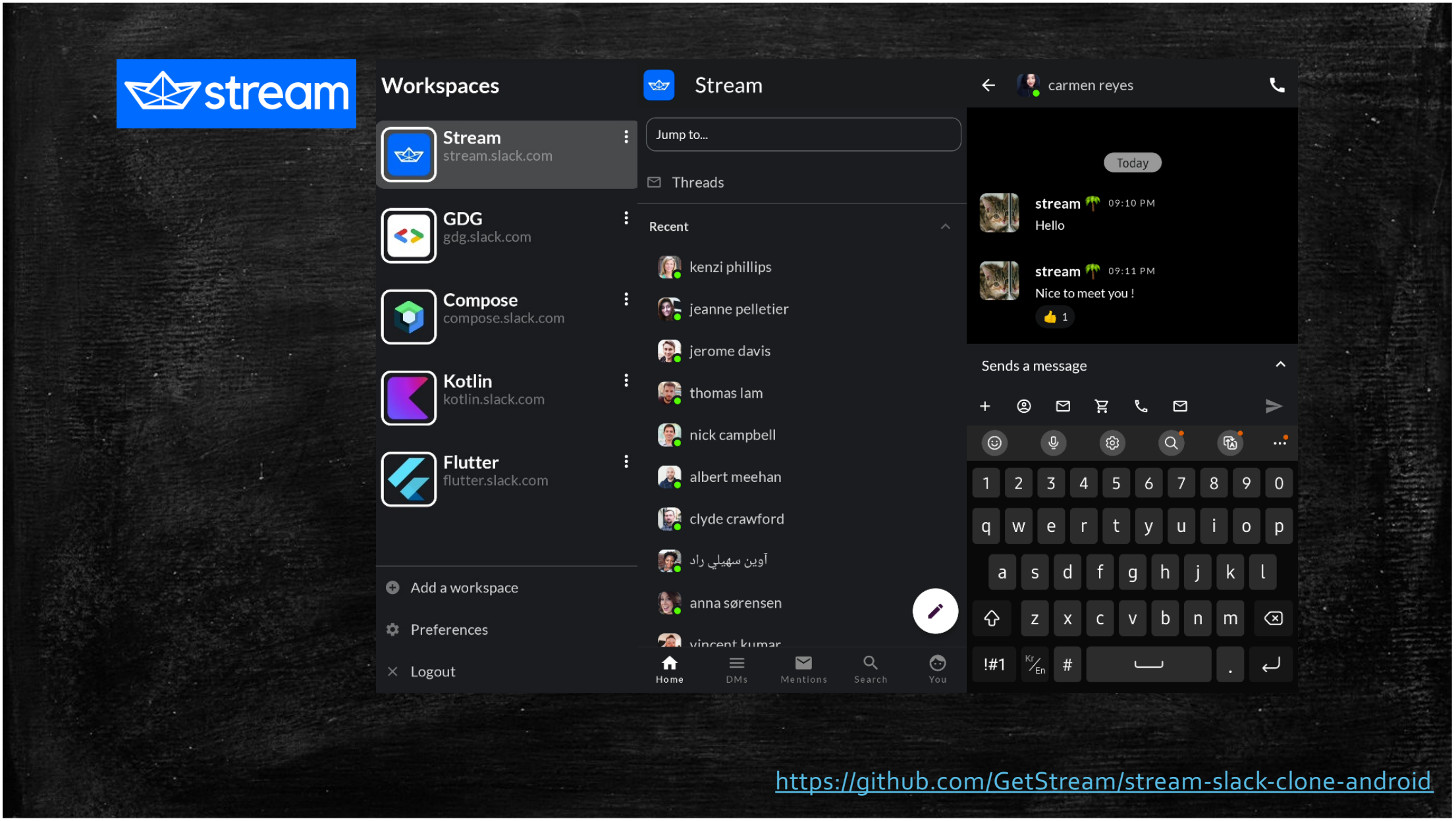
https://github.com/android/compose-samples/





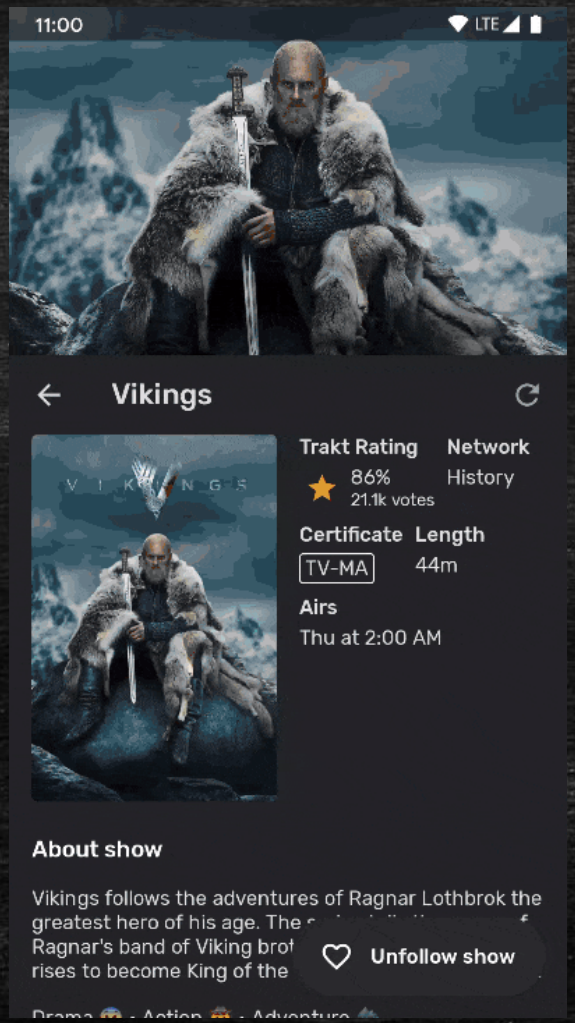


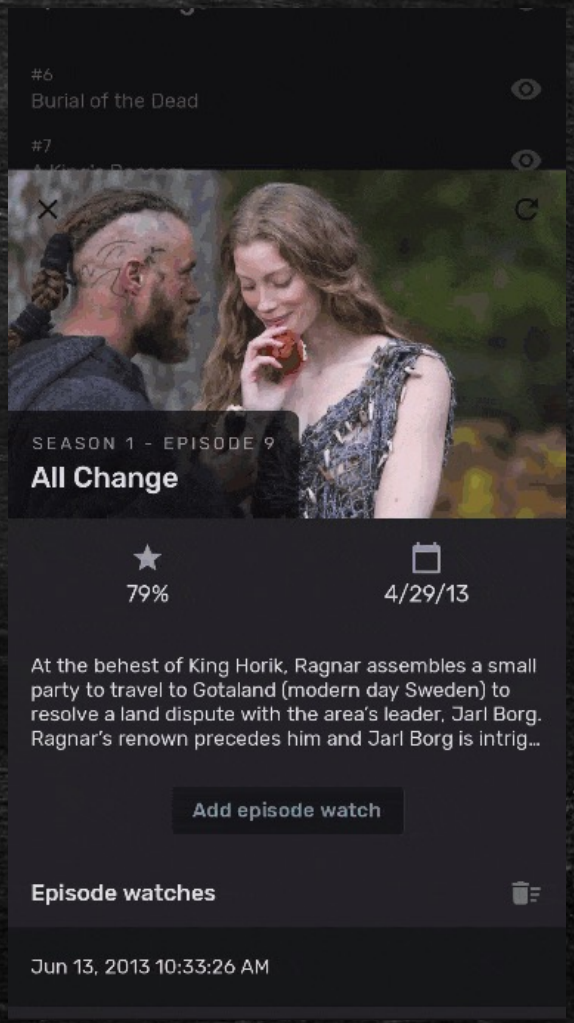
https://android-developers.googleblog.com/2021/05/androiddevchallenge-its-wrap.html



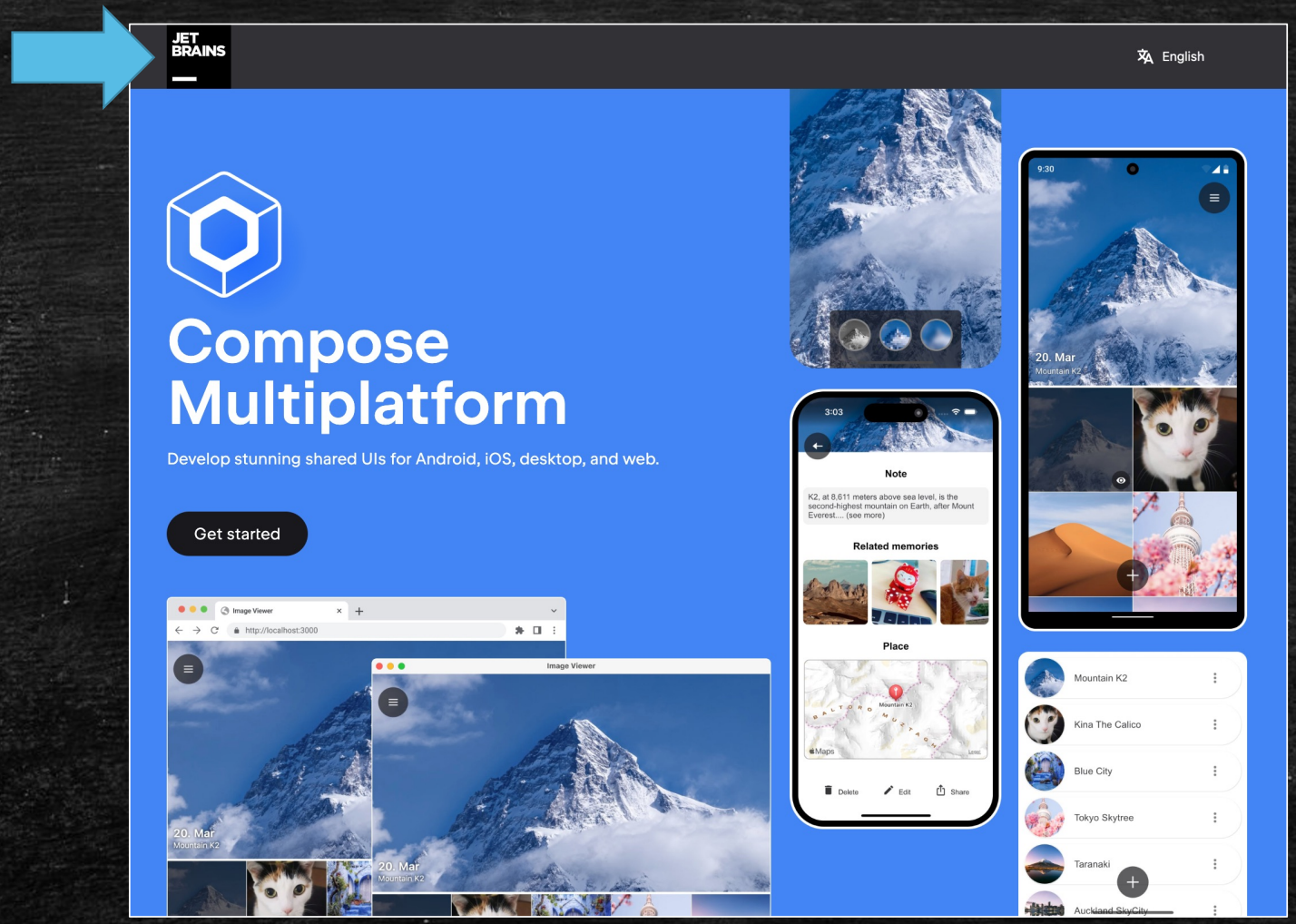
https://github.com/GetStream/stream-slack-clone-android

tivi



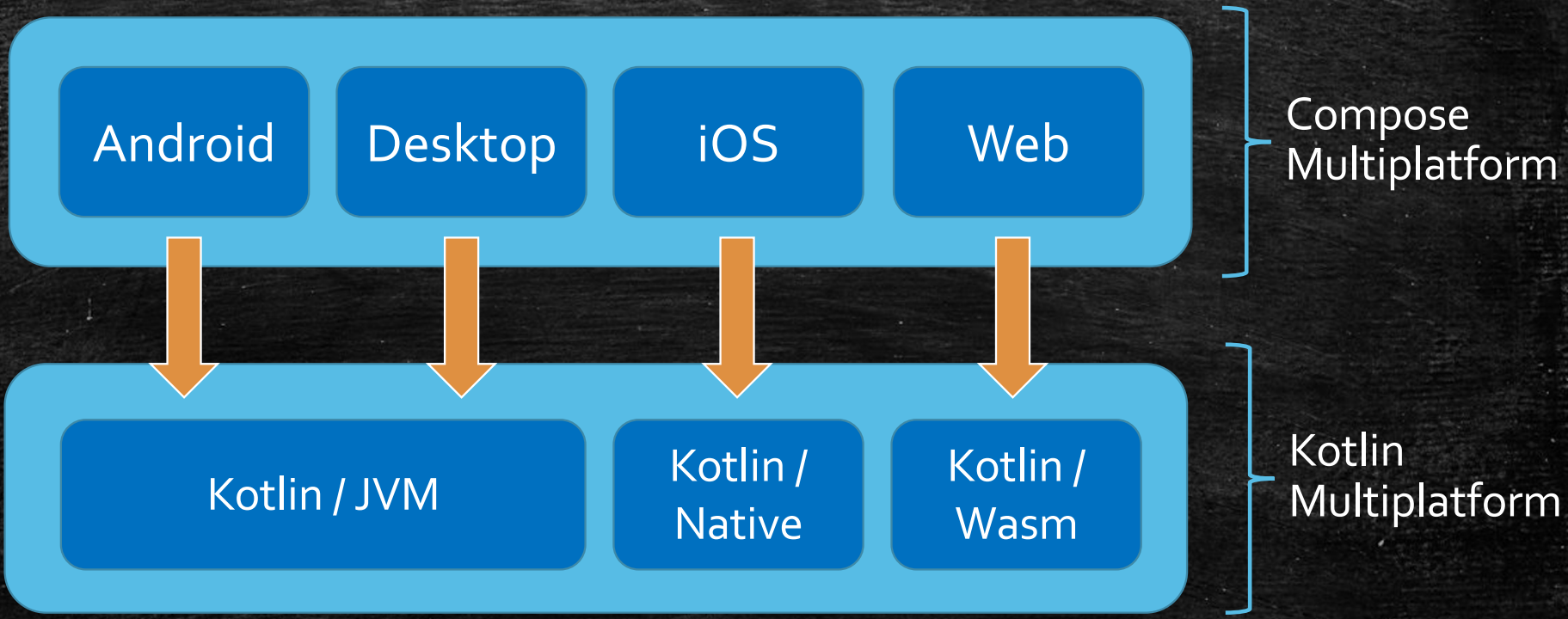


https://github.com/chrisbanes/tivi



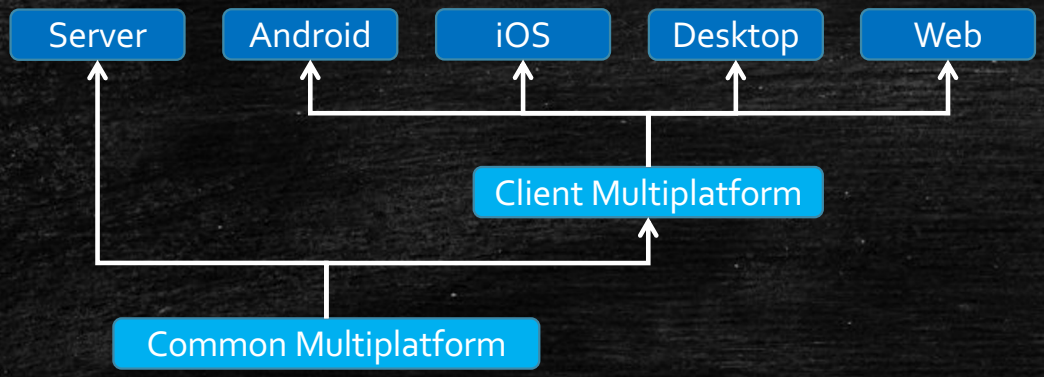
https://jb.gg/compose

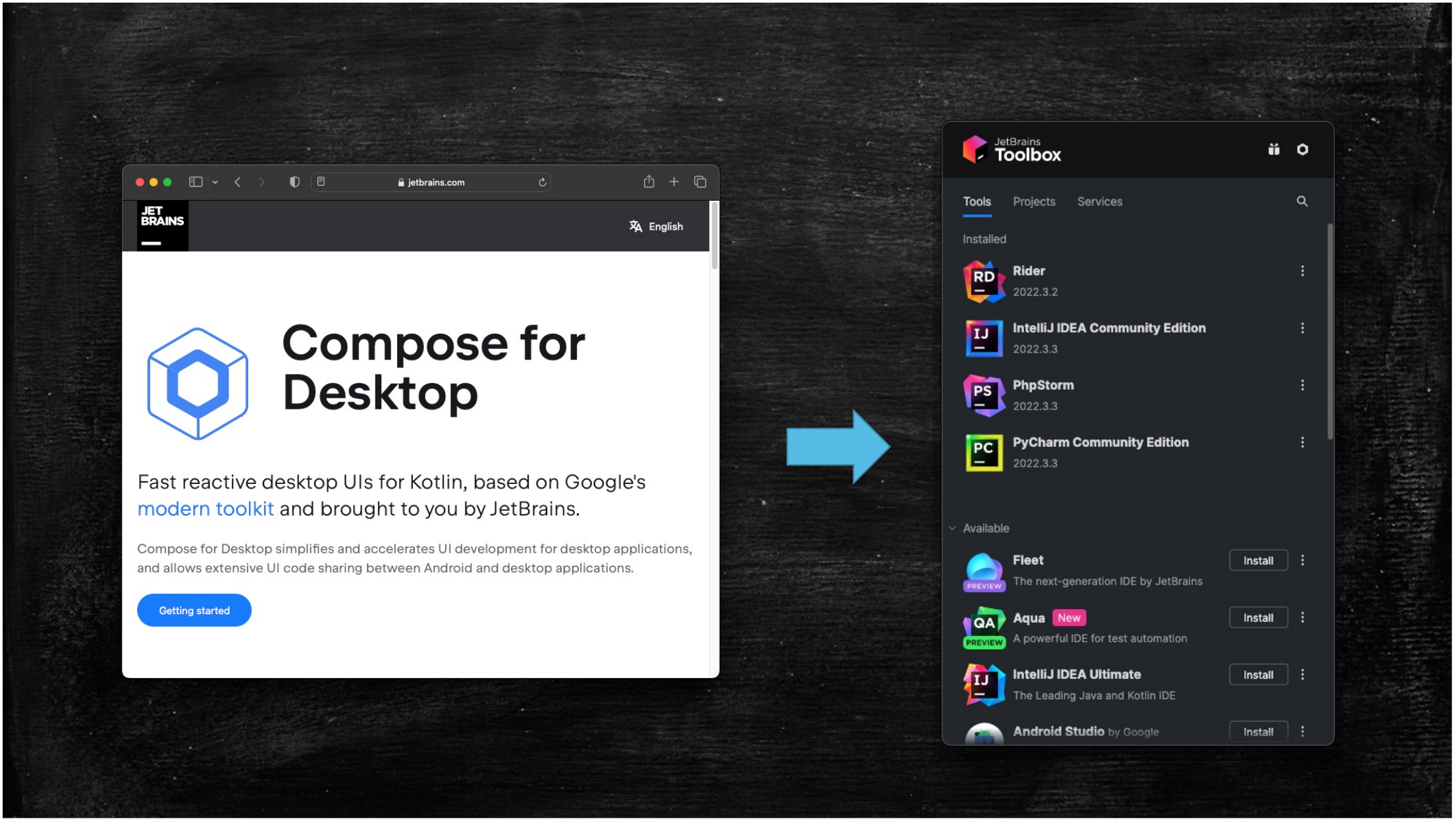
Compose & Kotlin Multiplatform

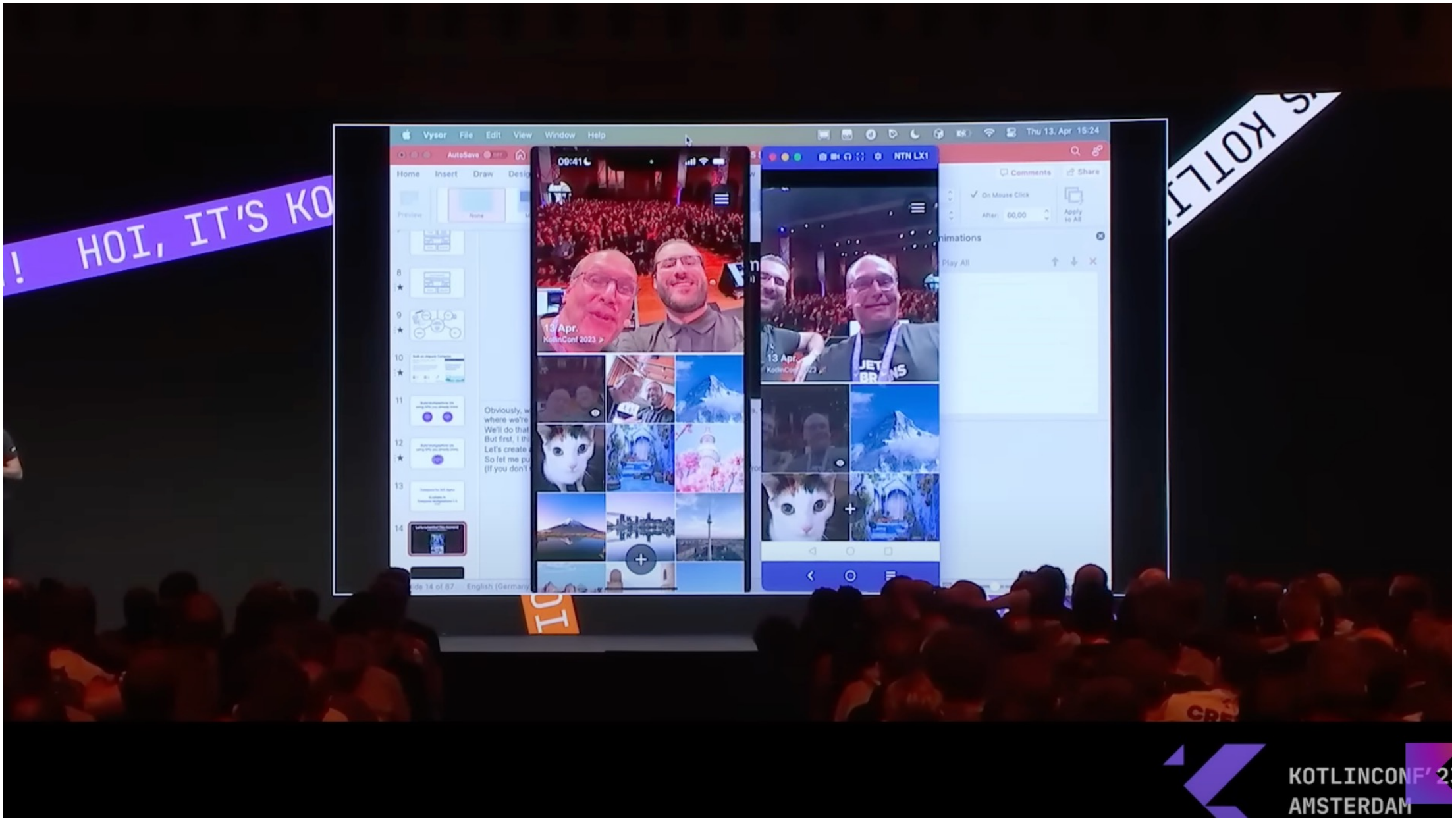


Kotlin Multiplatform == Sharing

- Kotlin Multiplatform simplifies cross platform projects
  - Common networking, data storage & validation, business logic etc.
- Whilst retaining all the benefits of native programming
  - Share code across server and clients where it makes sense
  - What remains is written natively – just as before

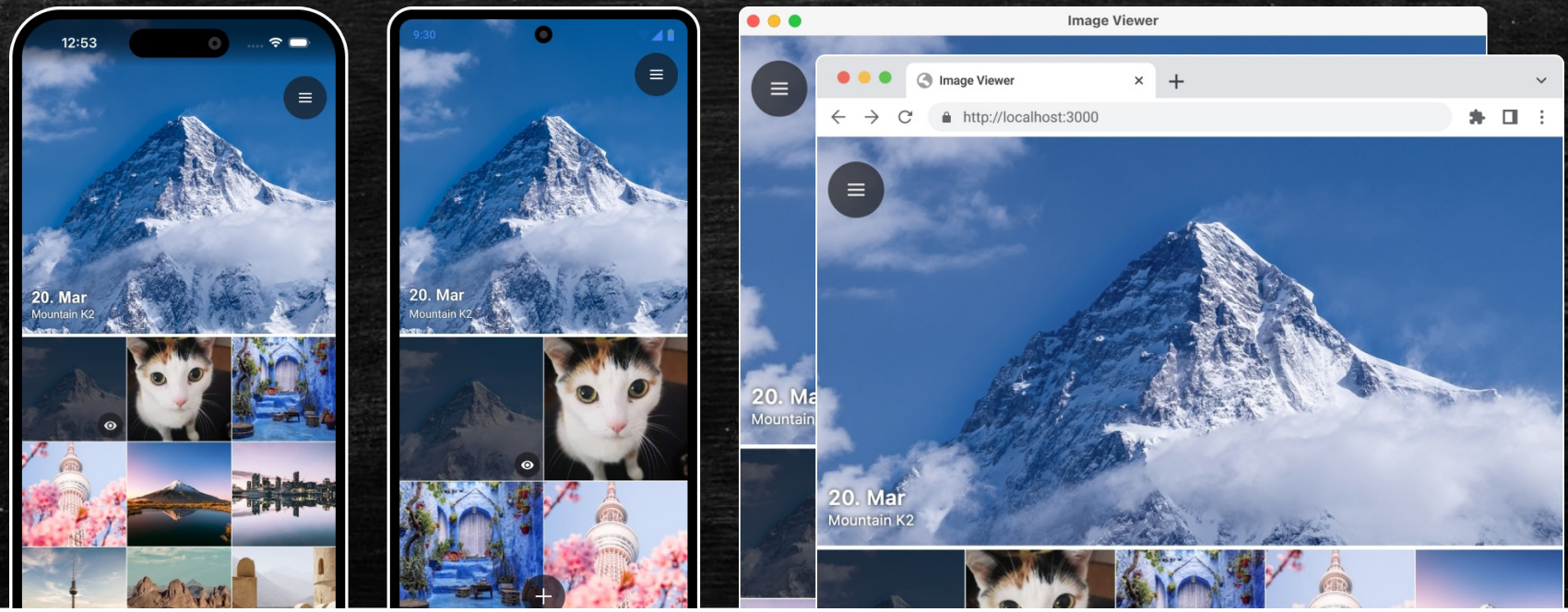


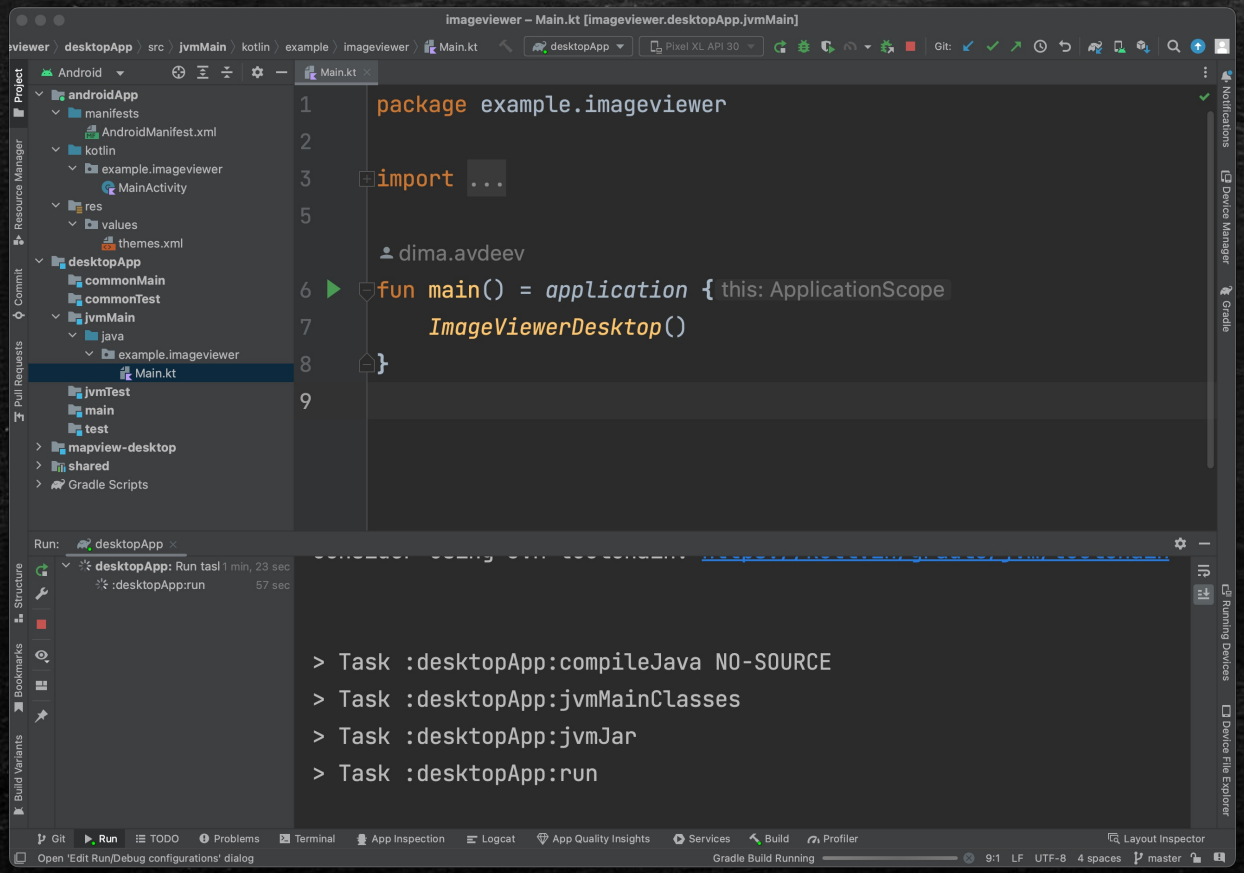

jetbrains.com
JET BRAINS
English
Compose for Desktop
Fast reactive desktop UIs for Kotlin, based on Google's modern toolkit and brought to you by JetBrains.
Compose for Desktop simplifies and accelerates UI development for desktop applications, and allows extensive UI code sharing between Android and desktop applications.
Getting started
JetBrains Toolbox
Tools
Projects
Services
Installed
Rider
2022.3.2
IntelliJ IDEA Community Edition
2022.3.3
PhpStorm
2022.3.3
PyCharm Community Edition
2022.3.3
Available
Fleet
The next-generation IDE by JetBrains
Install
Aqua
New
A powerful IDE for test automation
Install
IntelliJ IDEA Ultimate
The Leading Java and Kotlin IDE
Install
Android Studio by Google
Install


HOI, IT'S KO
KOTLINCONF'2
AMSTERDAM

The ImageViewer Sample Application

- An example image gallery with camera and map support
- Built on Compose Multiplatform (Desktop, Android and iOS)




imageviewer – Main.kt [imageviewer.desktopApp.jvmMain]
Android
androidApp
manifests
AndroidManifest.xml
kotlin
example.imageviewer
MainActivity
res
values
themes.xml
desktopApp
commonMain
commonTest
jvmMain
java
example.imageviewer
Main.kt
jvmTest
main
test
mapview-desktop
shared
Gradle Scripts
Main.kt
1 package example.imageviewer
2
3 import ...
5
dima.avdeev
6 fun main() = application { this: ApplicationScope
7 ImageViewerDesktop()
8 }
9
Run: desktopApp
desktopApp: Run tasl 1 min, 23 sec
:desktopApp:run 57 sec
> Task :desktopApp:compileJava NO-SOURCE
> Task :desktopApp:jvmMainClasses
> Task :desktopApp:jvmJar
> Task :desktopApp:run
Git
Run
TODO
Problems
Terminal
App Inspection
Logcat
App Quality Insights
Services
Build
Profiler
Layout Inspector
Open 'Edit Run/Debug configurations' dialog
Gradle Build Running
9:1 LF UTF-8 4 spaces master

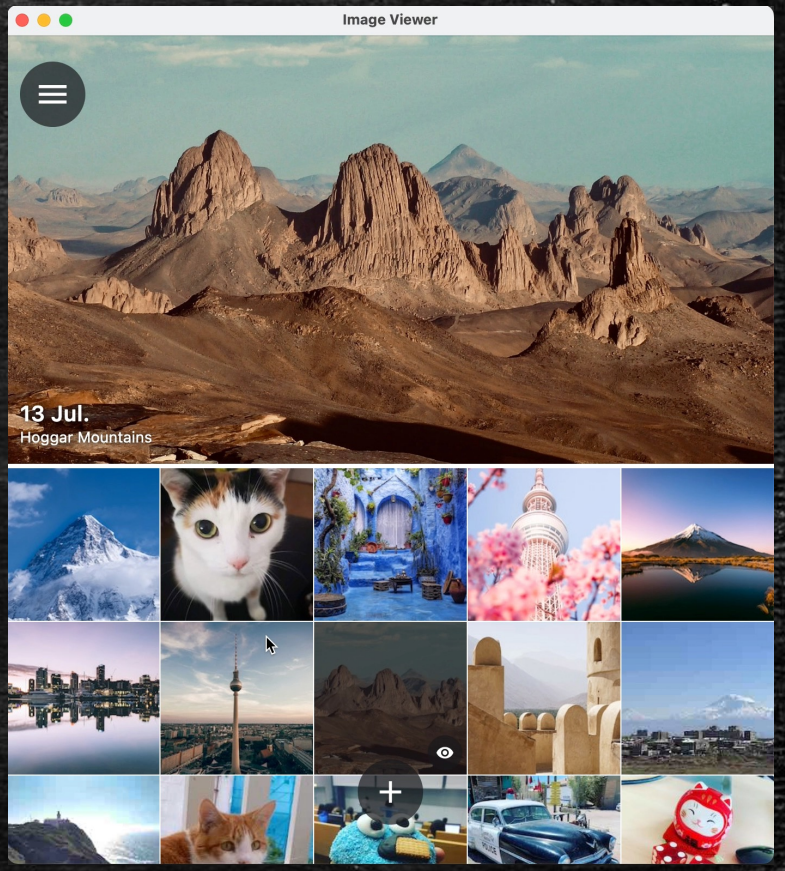
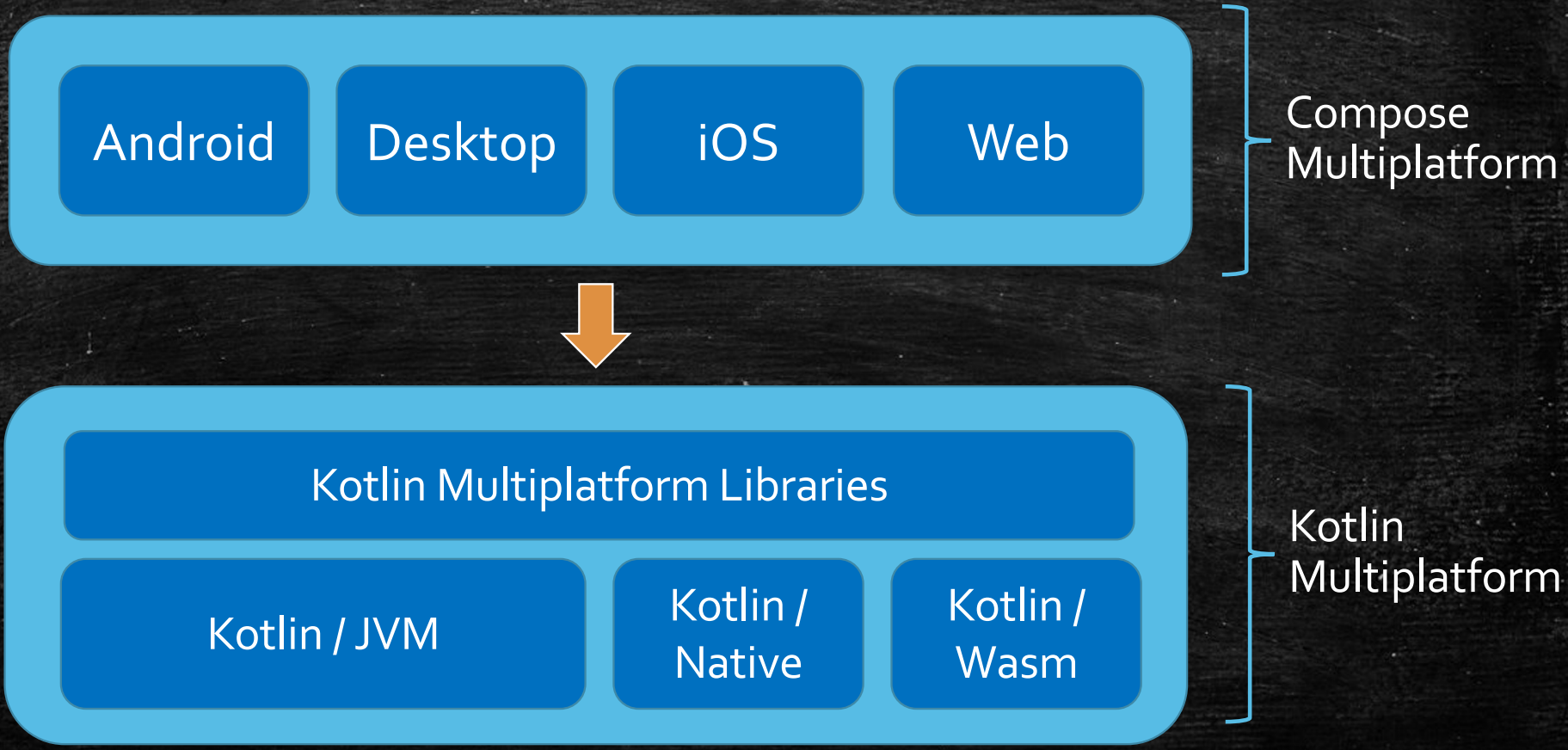


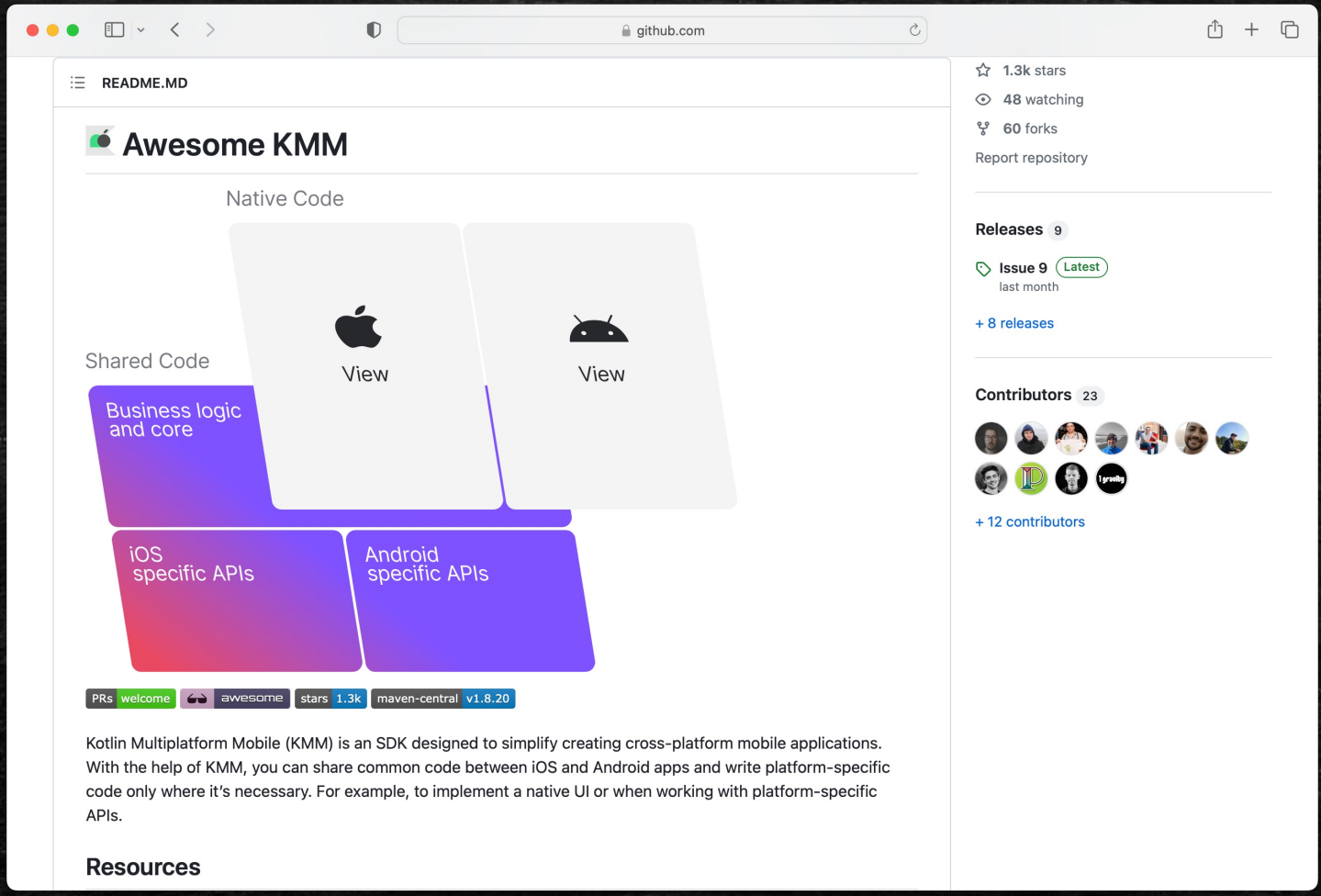
Image Viewer
13 Jul.
Hoggar Mountains

Compose & Kotlin Multiplatform & Libraries



Some Kotlin Multiplatform Libraries

Name	Description
Koin	Dependency Injection framework
Ktorfit	Adds declarative functionality to Ktor, similar to Retrofit
KMM-ViewModel	Allows sharing of ViewModels across Android and iOS
KMP-NativeCoroutines	Allows coroutines to be used / cancelled in Swift code
SQLDelight	Generates type safe Kotlin API's from SQL
Multiplatform Settings	Simplifies persisting key/value pairs in multiplatform code
JetPack DataStore	Stores key/value pairs and objects using coroutines and flows
Circuit	Architectural framework for Compose Multiplatform



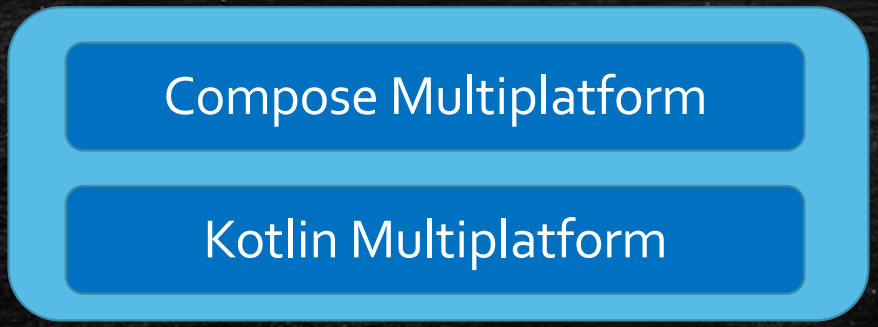
https://github.com/terrakok/kmm-awesome

https://terrakok.github.io/Compose-Multiplatform-Wizard/

Summary Of Part 1

- Compose lets you create awesome UIs on multiple platforms

- Compose Multiplatform lives on Kotlin Multiplatform

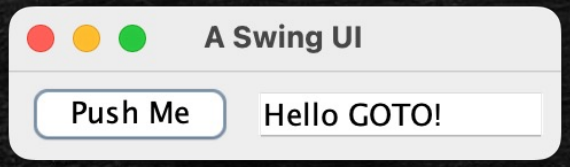


Part 2

Show Me The Code!

Manually Coding The UI

- Early UI libraries required a lot of low-level coding
  - AWT or Swing in Java
  - WinForms in .NET



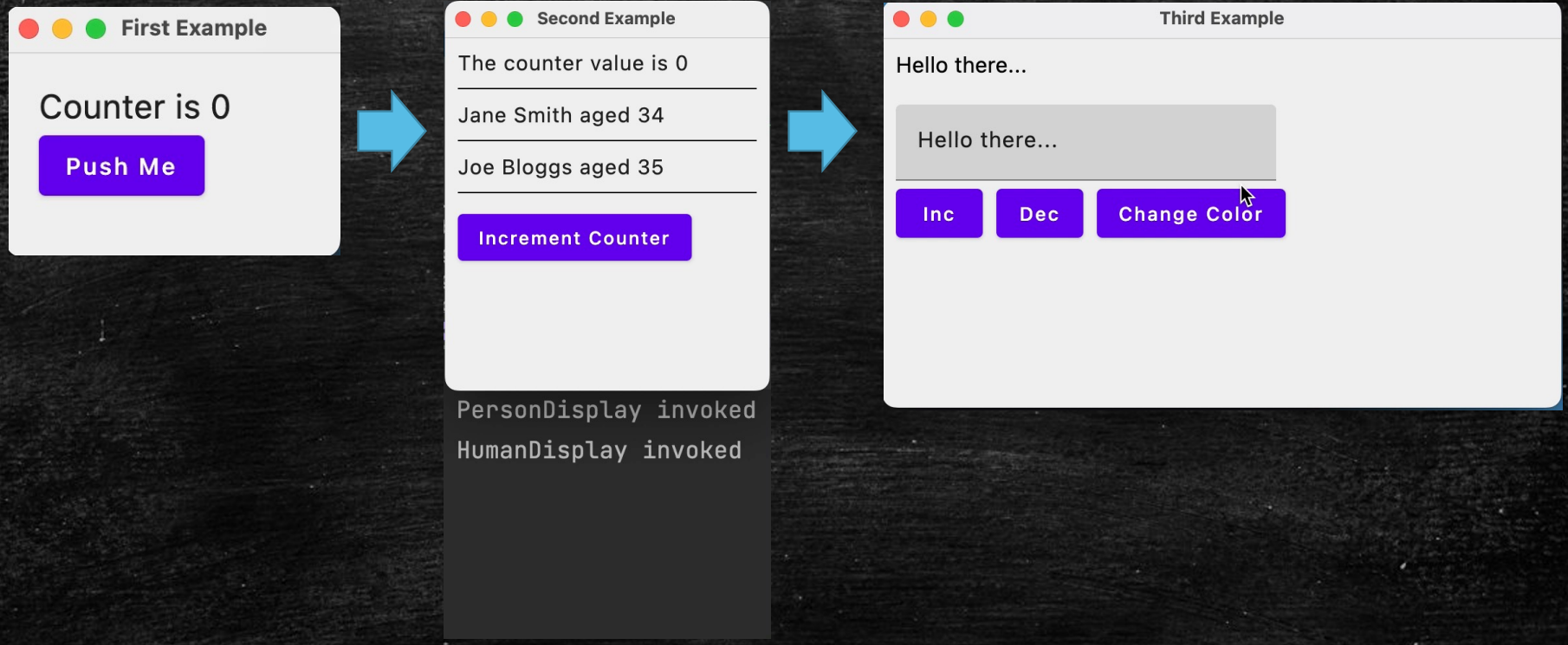

```
fun main() {
    val frame = JFrame("A Swing UI")
    frame.defaultCloseOperation = EXIT_ON_CLOSE

    val button = JButton("Push Me")
    val textField = JTextField(10)

    button.addActionListener {
        textField.text = "Hello GOTO!"
    }

    frame.layout = FlowLayout()
    frame.add(button)
    frame.add(textField)
    frame.pack()
    frame.isVisible = true;
}
```

Building The UI Declaratively

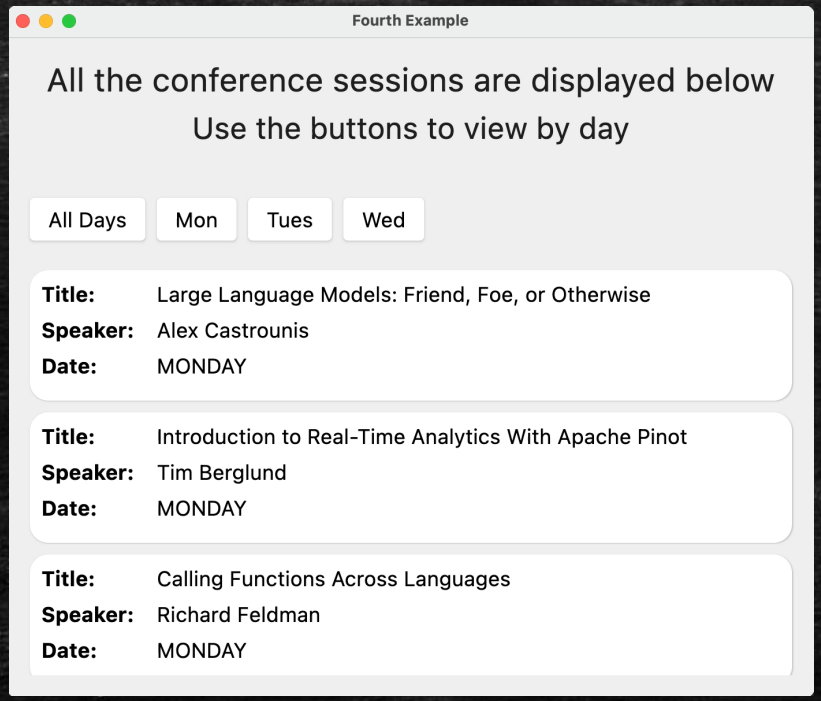
- Later UI libraries were declarative
  - XAML in .NET WPF
- Compose is also declarative
  - But everything is done within Kotlin
  - Via Composable Functions

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:id="@+id/textView1"
        android:fontFamily="@font/dancing_script"
        android:autoSizeTextType="uniform"
        android:layout_width="wrap_content"
        android:layout_height="99dp"
        android:text="@string/android_desserts"

        android:textAppearance="@style/MyTextAppearance" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="99dp"
        android:fontFamily="@font/typo_graphica"
        android:text="@string/android_desserts"
        android:textAppearance="@style/MyTextAppearance" />

</LinearLayout>
```

Our Gameplan



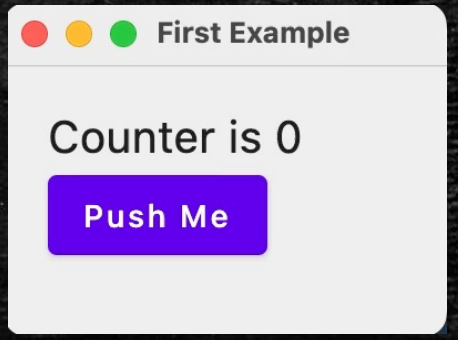
Our Gameplan



Hello Compose Multiplatform

```
val state = WindowState(
    size = DpSize(200.dp, 150.dp),
    position = WindowPosition(300.dp, 300.dp)
)

const val title = "First Example"

fun main() = singleWindowApplication(state = state, title = title) {
    MaterialTheme {
        FirstExample()
    }
}
```

```
@Composable
fun FirstExample() {
    var counter by remember { mutableStateOf(0) }

    Column(modifier = Modifier.padding(20.dp)) {
        Row {
            Text(
                style = TextStyle(fontSize = 20.sp),
                text = "Counter is $counter"
            )
        }
        Row {
            Button(onClick = { counter++ }) {
                Text("Push Me")
            }
        }
    }
}
```



Terms From Our First Example

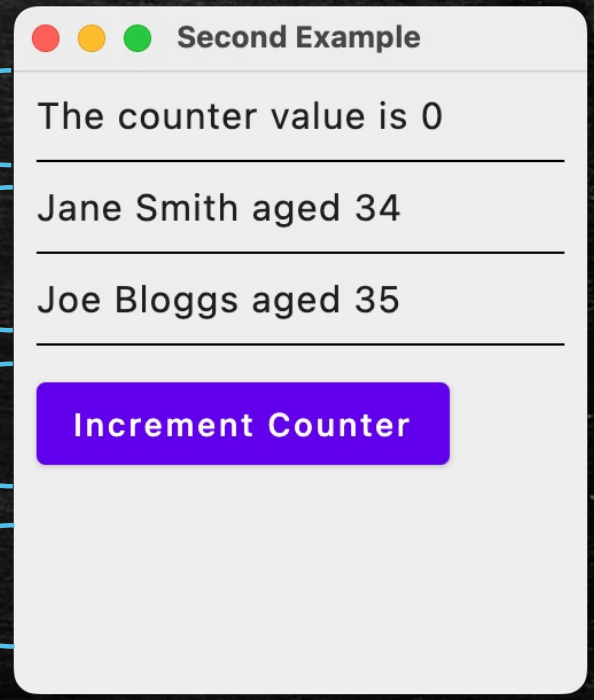
- We are coding with Composable Functions / Composables
- Underneath the hood these functions emit nodes
- The tree-structure that results is the Composition
- When observed data changes there is a recomposition

Maximizing Performance

- Compose is very good at avoiding unnecessary work
- It will skip Composable Function calls where possible
- Let's see a simple example of this…

Maximizing Performance

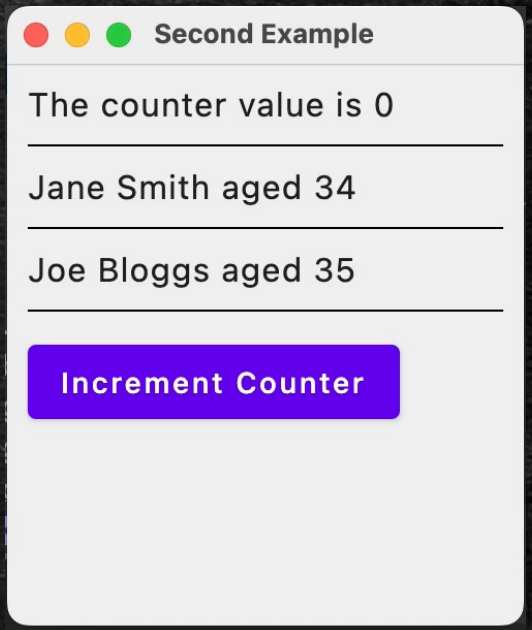
```
Column {
    CounterDisplay(counter.value)
    PersonDisplay(person)
    HumanDisplay(human)
    CounterButton {
        counter.value
            = counter.value + 1
    }
}
```



```kotlin
class Person(val name: String, val age: Int) {
    override fun toString() = "$name aged $age"
}

class Human(var name: String, var age: Int) {
    override fun toString() = "$name aged $age"
}
```

```kotlin
class Person(val name: String, val age: Int) {
    override fun toString() = "$name aged $age"
}

class Human(var name: String, var age: Int) {
    override fun toString() = "$name aged $age"
}


@Composable
fun PersonDisplay(person: Person) {
    println("PersonDisplay invoked")
    Row { Text(person.toString()) }
}


@Composable
fun HumanDisplay(human: Human) {
    println("HumanDisplay invoked")
    Row { Text(human.toString()) }
}
```

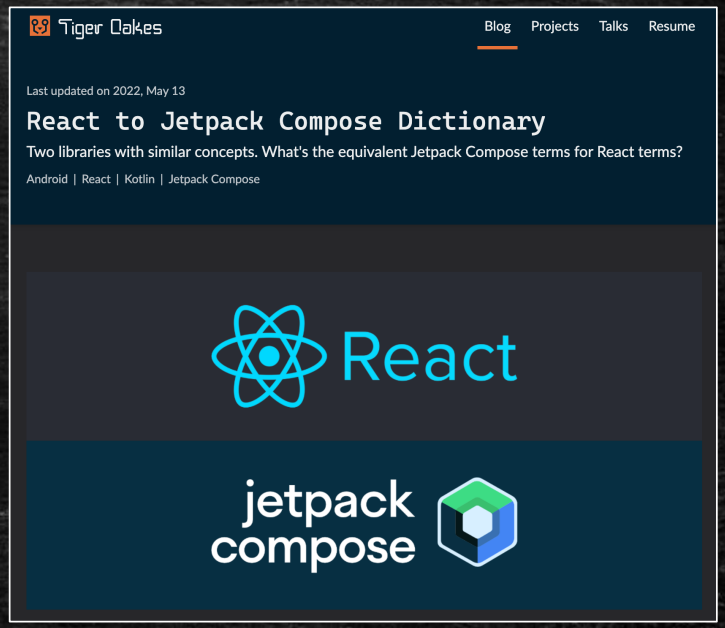
```kotlin
class Person(val name: String, val age: Int) {
    override fun toString() = "$name aged $age"
}

class Human(var name: String, var age: Int) {
    override fun toString() = "$name aged $age"
}


@Composable
fun PersonDisplay(person: Person) {
    println("PersonDisplay invoked")
    Row { Text(person.toString()) }
}


@Composable
fun HumanDisplay(human: Human) {
    println("HumanDisplay invoked")
    Row { Text(human.toString()) }
}
```




```
PersonDisplay invoked
HumanDisplay invoked
```

This Seems Very Familiar…

React	Compose
Component	Composable
Render	Composition
Hook	Effect
useEffect	LaunchedEffect
useMemo	remember



https://tigeroakes.com/posts/react-to-compose-dictionary/

A Composable Function

```
@Composable
fun ScrollingBox(content: @Composable () -> Unit) {
    Box(
        modifier = Modifier
            .padding(10.dp)
            .verticalScroll(rememberScrollState())
            .horizontalScroll(rememberScrollState())
    ) {
        content()
    }
}
```

Modifiers customize and extend existing Composables

Composables are just functions

A Composable Function

```
@Composable
fun ScrollingBox(content: @Composable () -> Unit) {
    Box(
        modifier = Modifier
            .padding(10.dp)
            .verticalScroll(rememberScrollState())
            .horizontalScroll(rememberScrollState())
    ) {
        content()
    }
}
```

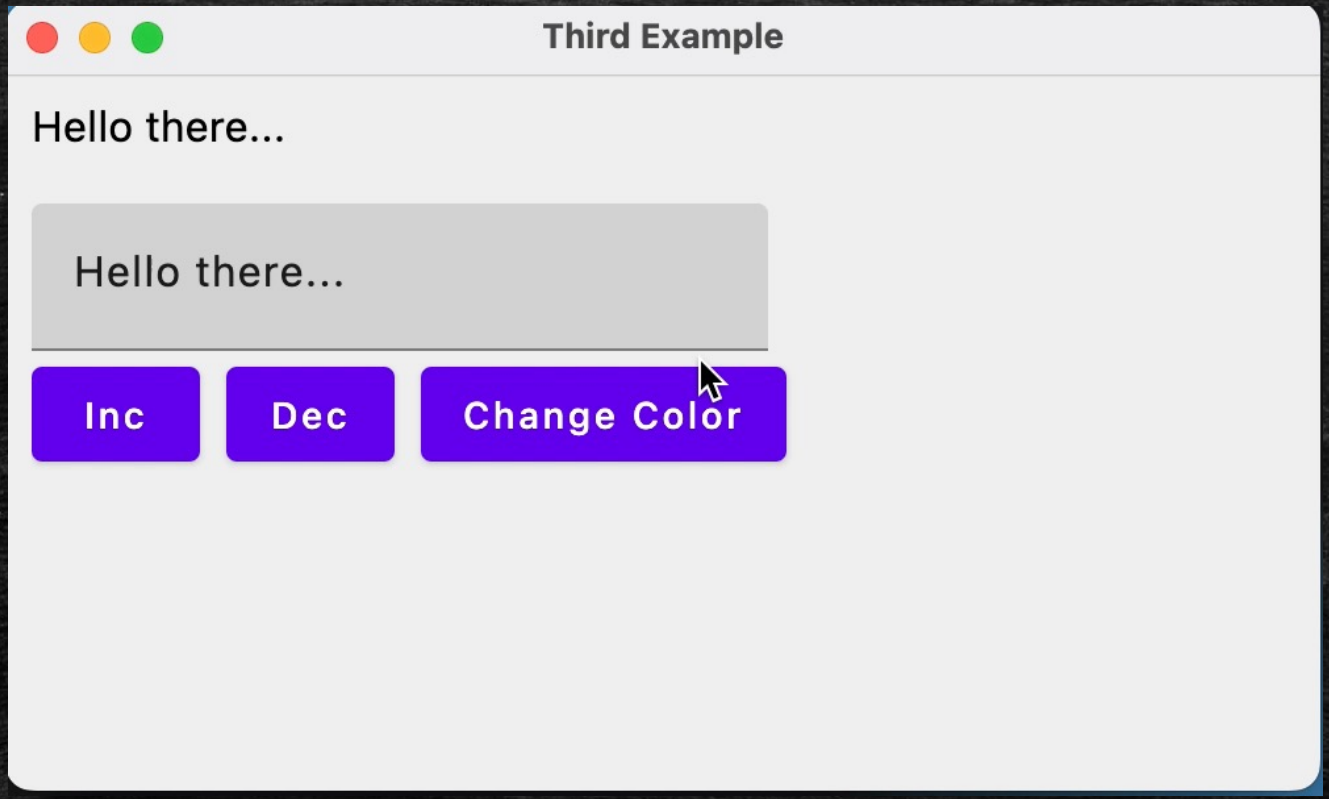
Modifiers customize and extend existing Composables

Composables are just functions

Our Third Example

- Let's write something a little more complex…
- Then port it to Android and iOS

Our Third Example (On Desktop)



Our Third Example

```
val state = WindowState(
    size = DpSize(500.dp, 300.dp),
    position = WindowPosition(100.dp, 100.dp)
)

fun main() = singleWindowApplication(state = state) {
    MaterialTheme {
        val viewModel = remember { AppViewModel() }
        ThirdExample(viewModel)
    }
}
```

Compose for Desktop

Compose Multiplatform

Our Third Example

```
@Composable
fun ThirdExample(viewModel: AppViewModel) {
    ScrollingBox {
        Column {
            Row(modifier = Modifier.padding(bottom = 20.dp)) {
                Text(
                    text = viewModel.content,
                    style = viewModel.style
                )
            }
            Row {
                TextField(value = viewModel.content, onValueChange = {
                    viewModel.content = it
                })
            }
            Row { ButtonPanel(viewModel) }
        }
    }
}
```

The ViewModel holds the state of the UI

Handling the event

ViewModel passed down

Our Third Example

```
@Composable
fun ButtonPanel(viewModel: AppViewModel) {
    PaddedButton("Inc") {
        viewModel.increaseFont()
    }
    PaddedButton("Dec") {
        viewModel.decreaseFont()
    }
    PaddedButton("Change Color") {
        viewModel.switchColor()
    }
}
```

The ViewModel holds the event handlers

Our Third Example

```
class AppViewModel {
    companion object { ... }

    private var color by mutableStateOf(defaultColor)
    private var size by mutableStateOf(defaultFontSize)

    val style by derivedStateOf {
        TextStyle(color = color, fontSize = size.sp)
    }

    var content by mutableStateOf("Hello there...")
```

Constant values stored here

Style is observed by the Text Composable and observes the color and size properties

Content is observed by the Text Composable and modified by the TextField

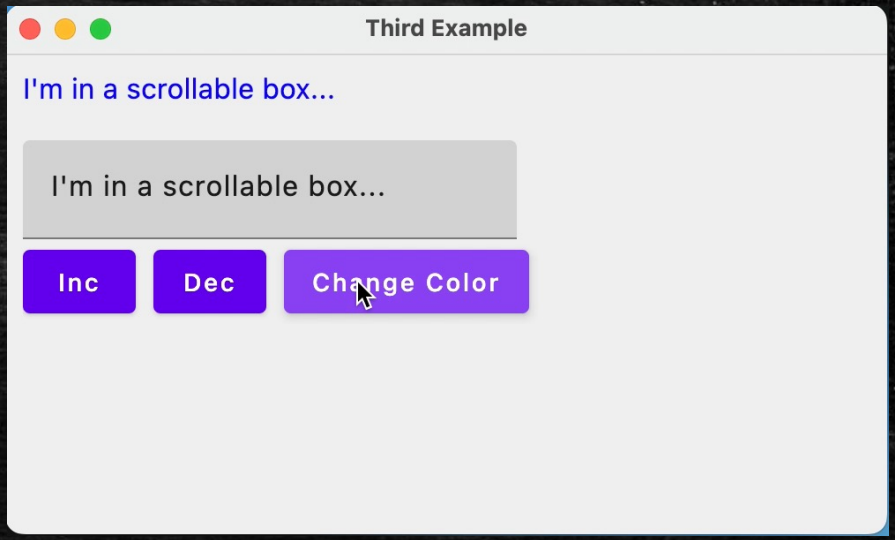
Our Third Example

```
    fun increaseFont() {
        if (size < maximumFontSize)
            size += 2
    }

    fun decreaseFont() {
        if (size > minimumFontSize)
            size -= 2
    }

    fun switchColor() {
        val index = nextInt(0, sampleColors.size)
        color = sampleColors[index]
    }
}
```

The event handlers simply modify the properties

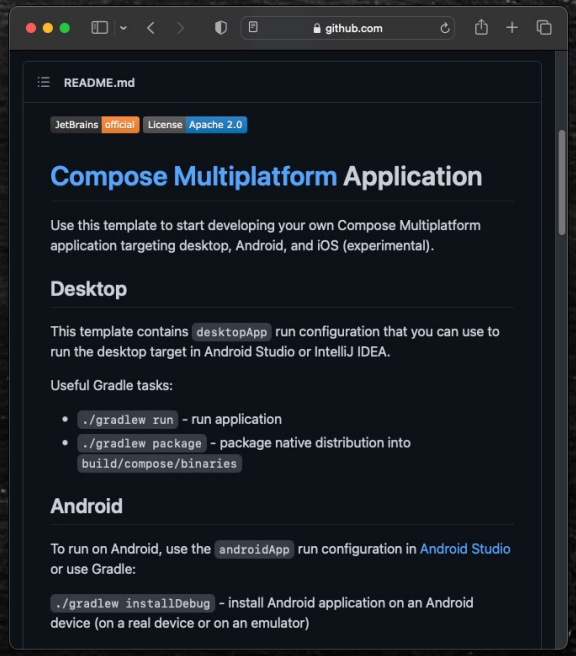
The Scrolling Box Revisited

```
@Composable
fun ThirdExample(viewModel: AppViewModel) {
    ScrollingBox {
        Column {
            Row {
                ...
            }
            Row {
                ...
            }
            Row {
                ...
            }
        }
    }
}
```



Creating An Android / iOS Project

- Create your project using the template provided by JetBrains in GitHub
- This can then be opened in Android Studio
- The README provides advice on setting up your machine for mobile dev



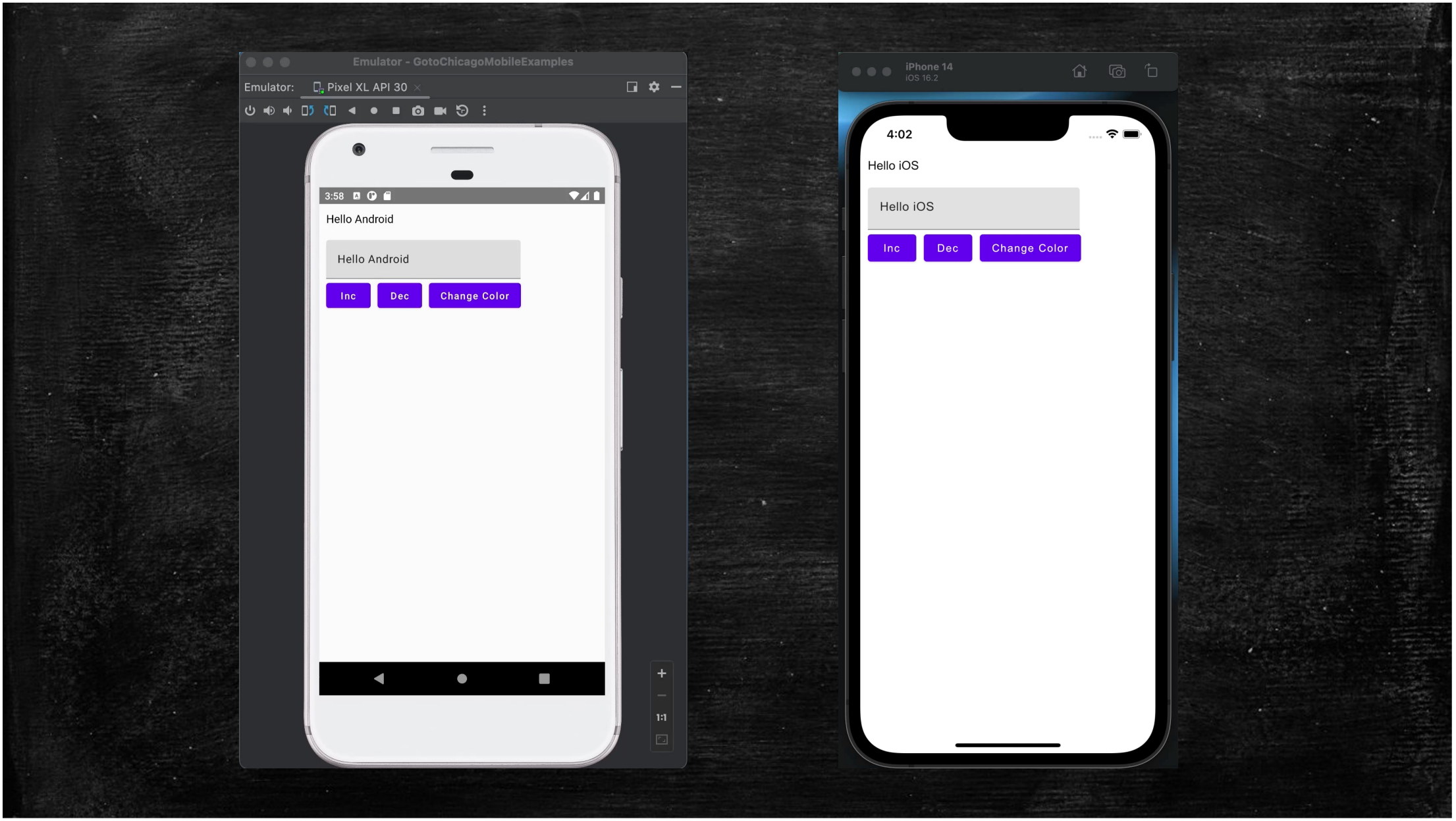
https://github.com/JetBrains/compose-multiplatform-template

Our Third Example On Mobile

```
@Composable
fun App() {
    MaterialTheme {
        val viewModel = remember {
            AppViewModel(getPlatformName())
        }
        FirstExample(viewModel)
    }
}

expect fun getPlatformName(): String
```

It's just the same!!!

With the platform name injected to show the expect function

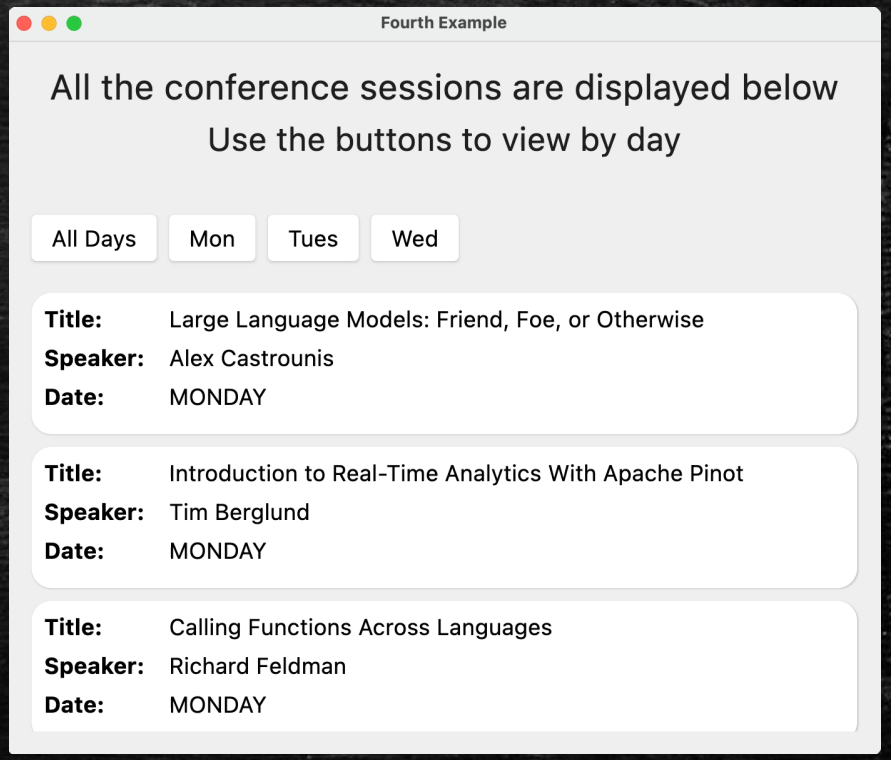

Emulator - GotoChicagoMobileExamples
Emulator: Pixel XL API 30
3:58
Hello Android
Hello Android
Inc
Dec
Change Color
1:1
iPhone 14
iOS 16.2
4:02
Hello iOS
Hello iOS
Inc
Dec
Change Color

More Lessons From Our Third Example

- Making a start on (or off) mobile is easier than you think
- Project templates, utilities, tutorials etc. are all available
- A huge deal for projects trying to reduce expense and duplication

Our Fourth Example

- Let's be more ambitious
- Build a viewer for the GOTO Chicago sessions
- Download talk descriptions in JSON over a WebSocket



The Libraries We Need

- Kotlinx.serialization
- Kotlinx.datetime
- Ktor (Server and Client)
- WebSockets in Ktor
- Kotlin Coroutines

Our Fourth Example

```
import kotlinx.datetime.LocalDate
import kotlinx.serialization.Serializable

@Serializable
data class GotoSession(
    val speaker: String,
    val talkTitle: String,
    val date: LocalDate
)
```

Using two Kotlinx libraries, plus the data class feature

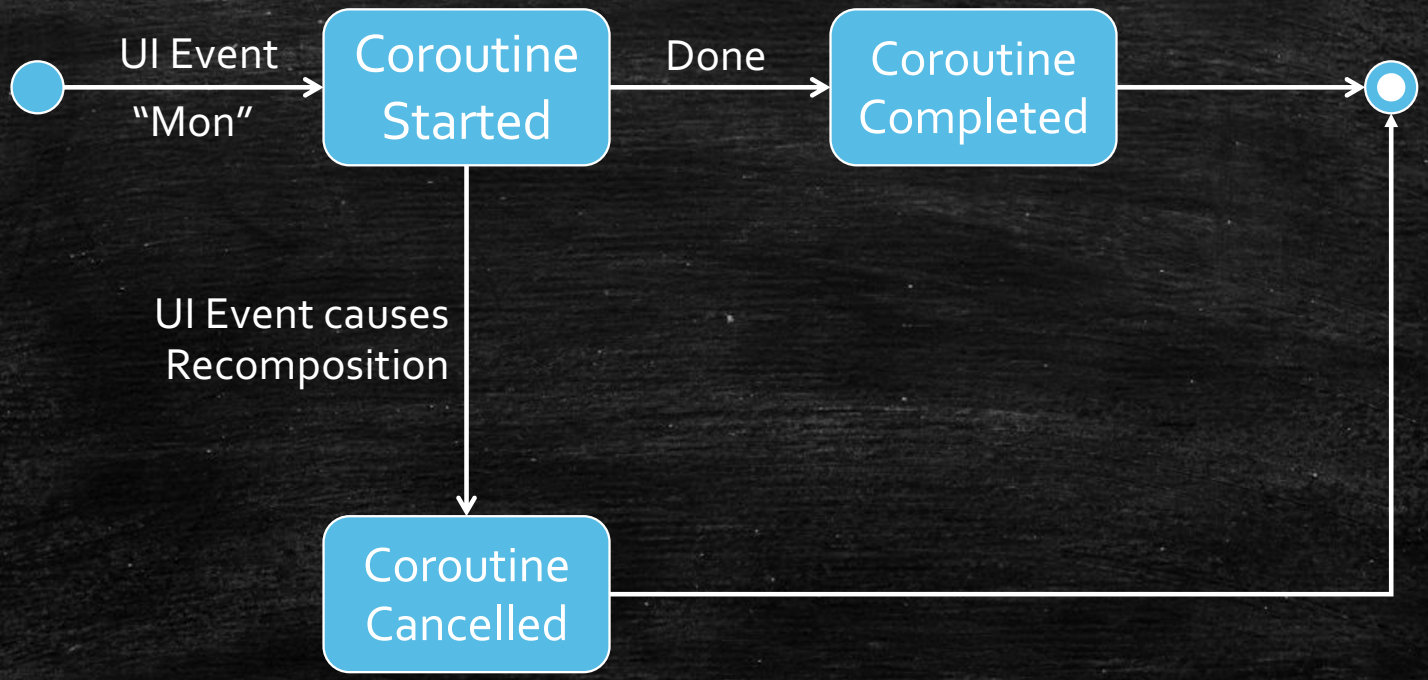
Our Fourth Example

```
class WebSocketClient(
    private val host: String,
    private val port: Int,
    private val basePath: String
) {

    private val client = HttpClient(CIO) {
        install(ContentNegotiation) {...}
        install(WebSockets) { ... }
    }

    suspend fun fetchAllSessions(
        gotoSessions: MutableList<GotoSession>
    ) { ... }

    suspend fun fetchSessionsByDate(
        gotoSessions: MutableList<GotoSession>,
        date: String
    ) { ... }
}
```

Ktor client supports WebSockets with minimal fuss

Suspending functions run within coroutines, which provide concurrency

Managing Coroutines In Composables



Our Fourth Example

ViewModel omitted for the sake of brevity

```
@Composable
fun FourthExample(client: WebSocketClient) {
    val gotoSessions = remember {
        mutableStateOf(emptyList<GotoSession>())
    }

    val scope = rememberCoroutineScope()

    val fetchSessionsByDate: (String) -> Unit = { date ->
        scope.launch {
            if (date.isBlank()) {
                client.fetchAllSessions(gotoSessions)
            } else {
                client.fetchSessionsByDate(gotoSessions, date)
            }
        }
    }
```

We store a list of sessions in the composition

Creating a Coroutine Scope

Creating Coroutines

Our Fourth Example

```
    LaunchedEffect(gotoSessions) {
        client.fetchAllSessions(gotoSessions)
    }

    Box(modifier = Modifier.padding(20.dp)) {
        Column {
            TitleBar()
            FetchByDateButtons(fetchSessionsByDate)
            GotoSessionsDisplay(gotoSessions)
        }
    }
}
```

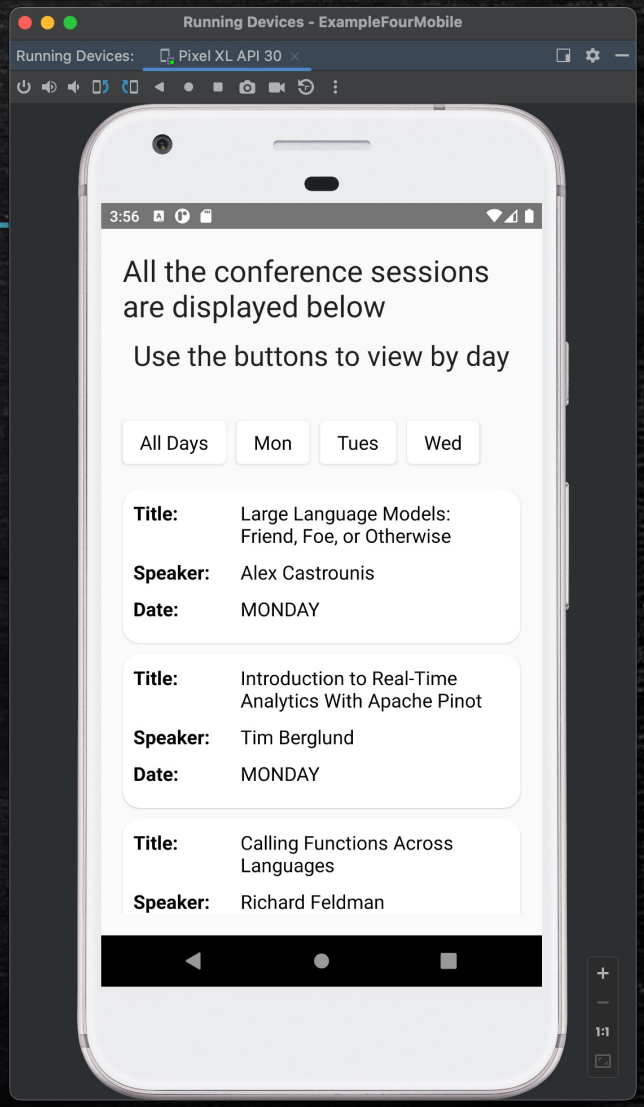
Second use of Coroutines

Lessons From Our Fourth Example

- Concurrency is baked into Compose via Coroutines
- Kotlin Multiplatform provides supporting libraries
- These libraries are essential in real world applications

It Works As Is On Mobile!

```
@Composable
fun App() {
    val client = remember {
        WebSocketClient(
            "...",
            8080,
            "/goto/sessions"
        )
    }
    MaterialTheme {
        FourthExample(client)
    }
}
```



Part 3

Alternative approaches

What about [OTHER] ?

What about [OTHER] ?

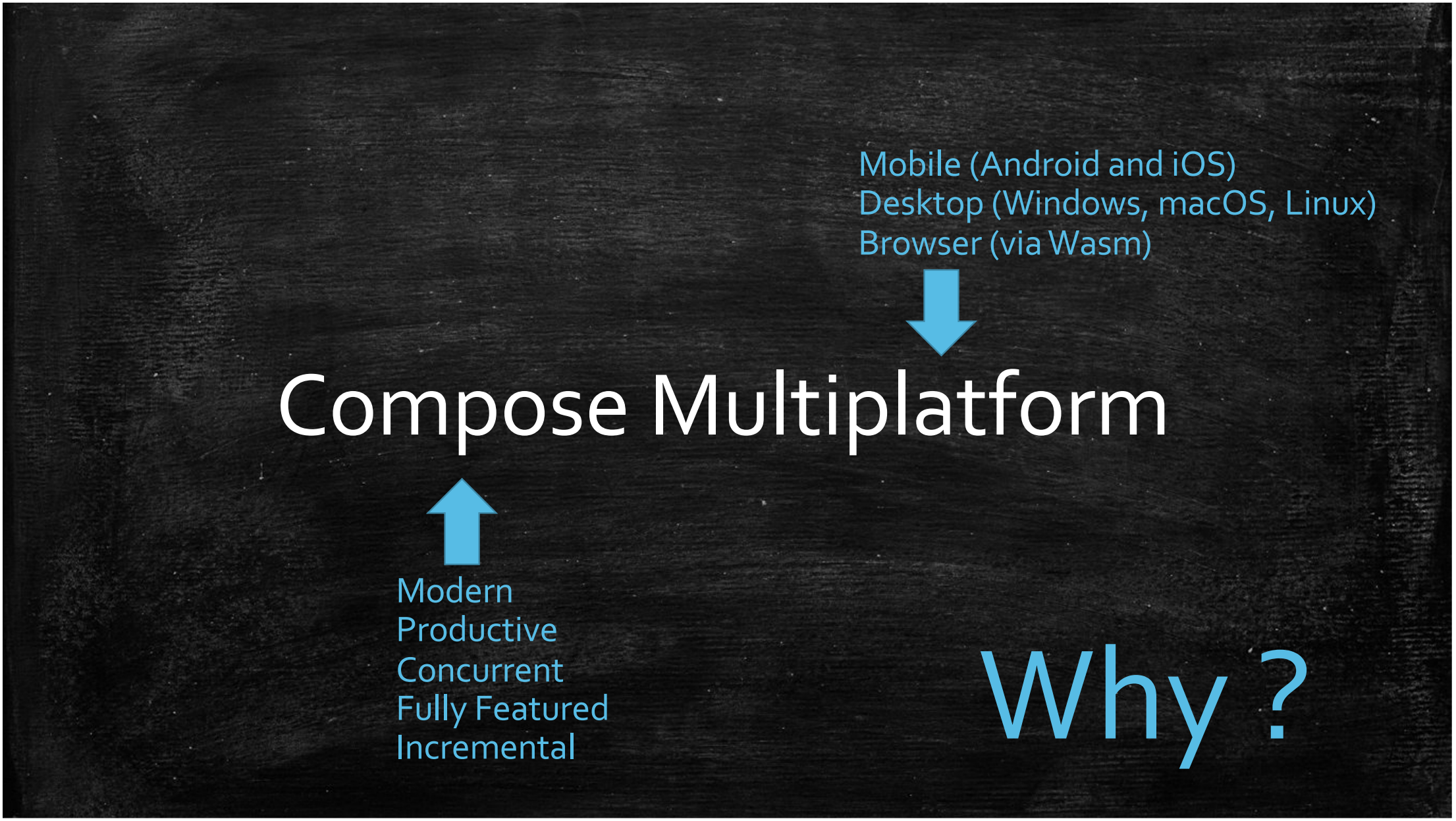
- Kotlin is a highly popular, widely used, multiplatform language
- Which isn't JavaScript ☺
- Plus it compiles natively on iOS
- But most importantly …

What about [OTHER]

- You can use Kotlin Multiplatform with or without Compose
- In some cases, you may only share data types and logic
- In others you may share the entire user interface
- It's not a binary choice / all or nothing

Conclusions

Summing Up & Additional Resources


Mobile (Android and iOS)
Desktop (Windows, macOS, Linux)
Browser (via Wasm)
Compose Multiplatform
Modern
Productive
Concurrent
Fully Featured
Incremental
Why ?

Why ?

- Building on a popular and well supported language
- Both Kotlin Multiplatform and Compose Multiplatform
- Access to the native capabilities of each platform
- Access to a rich ecosystem of libraries and tools
- Leverage work being done by other communities
- Easily support new platforms and reuse components

goto;

Don't forget to
vote for this session
in the **GOTO Guide app**