Beyond the UI

Compose as a Foundation for Multiplatform Apps

mDevCamp - June '25 🇨🇿

Ash Davies

Android GDE Berlin

Cat Person 😸

ashdavies.dev

Jetpack Compose UI

github.com/androidx/androidx/tree/androidx-main/compose/ui

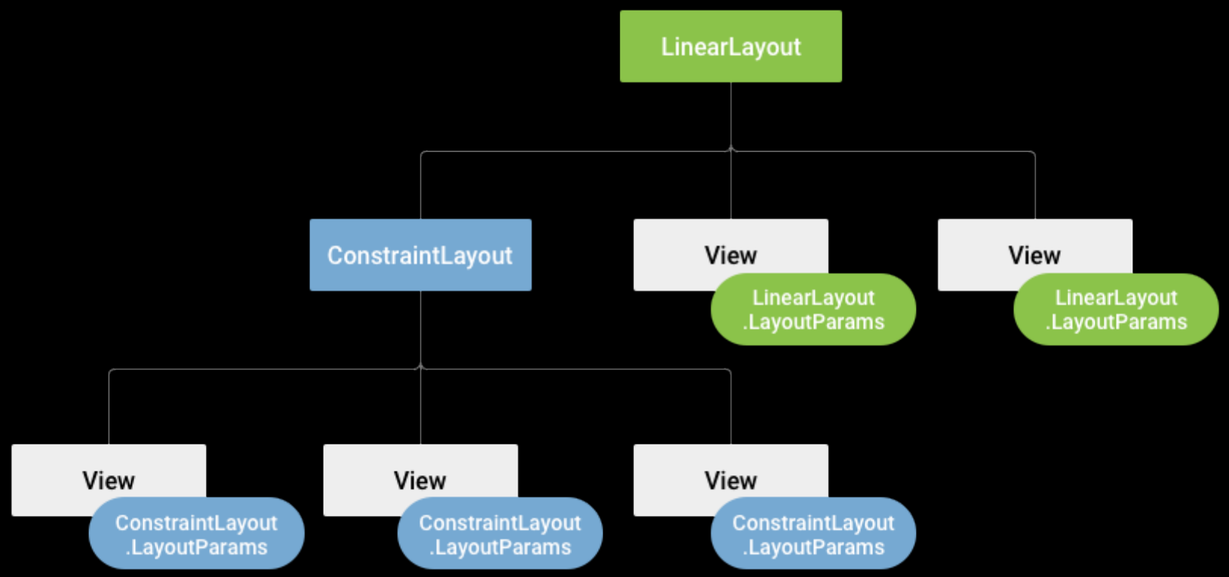
```
@Composable
fun JetpackCompose() {
    Card {
        var expanded by remember { mutableStateOf(false) }
        Column(Modifier.clickable { expanded = !expanded }) {
            Image(painterResource(R.drawable.jetpack_compose))
            AnimatedVisibility(expanded) {
                Text(
 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

Compose UI

— Declarative UI Framework
— Open Source Kotlin
— Accelerate UI development
— Intuitive Idiomatic API

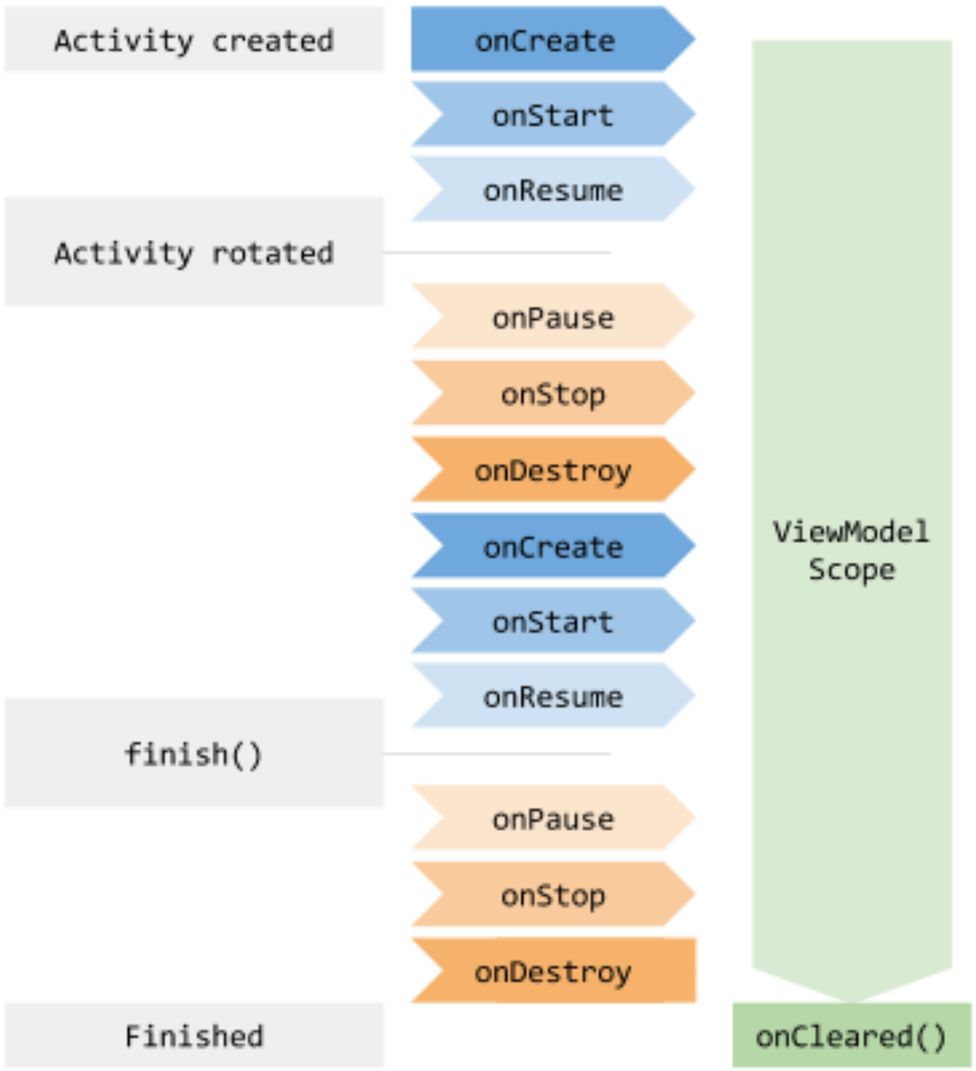
Android Layouts

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_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>
```

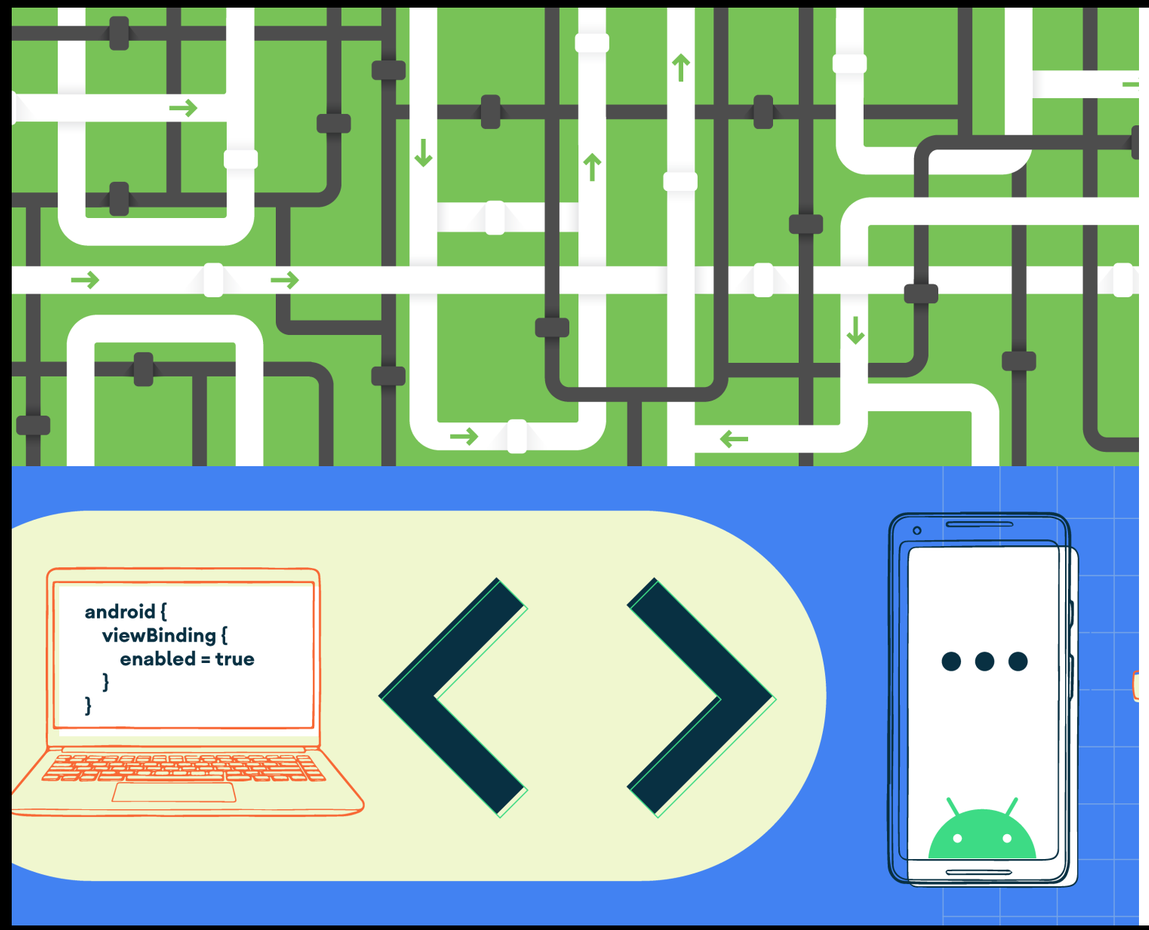


Android Layouts

Lifecycle



Tooling



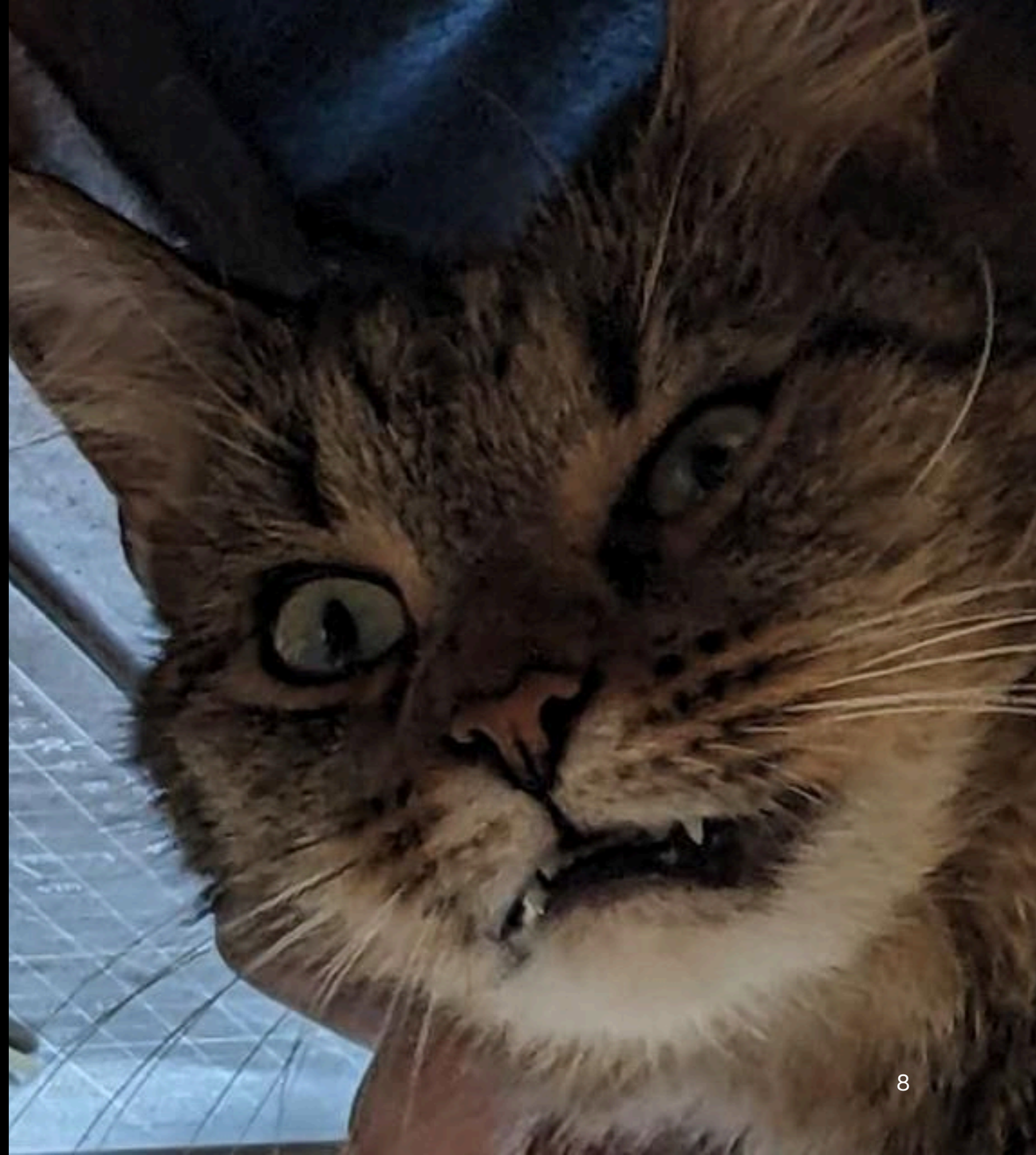
Anko

LouisCAD/**Splitties**

A collection of hand-crafted extensions for your Kotlin projects.

11 Contributors · 51 Issues · 3 Discussions · 3k Stars · 162 Forks

XML?!



Android Layouts and Compose

— Widespread adoption of Kotlin on Android
— Idiomatic Kotlin language features

Jetpack Compose UI

Principles

— Composition > Inheritance

— Declarative Syntax

— Immutable State

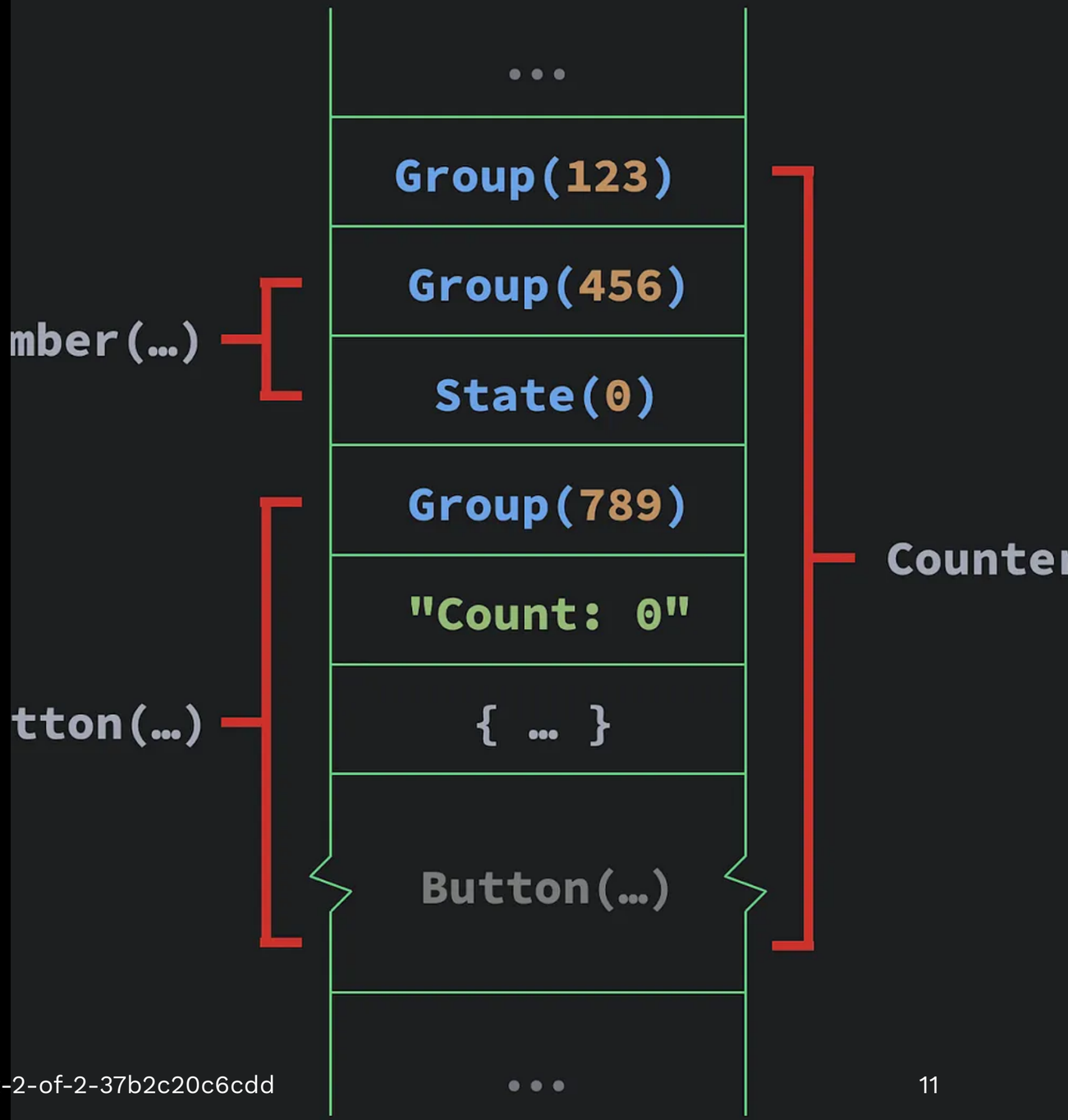
— Recomposition

Jetpack Compose UI

Under-the-hood

— Kotlin compiler plugin

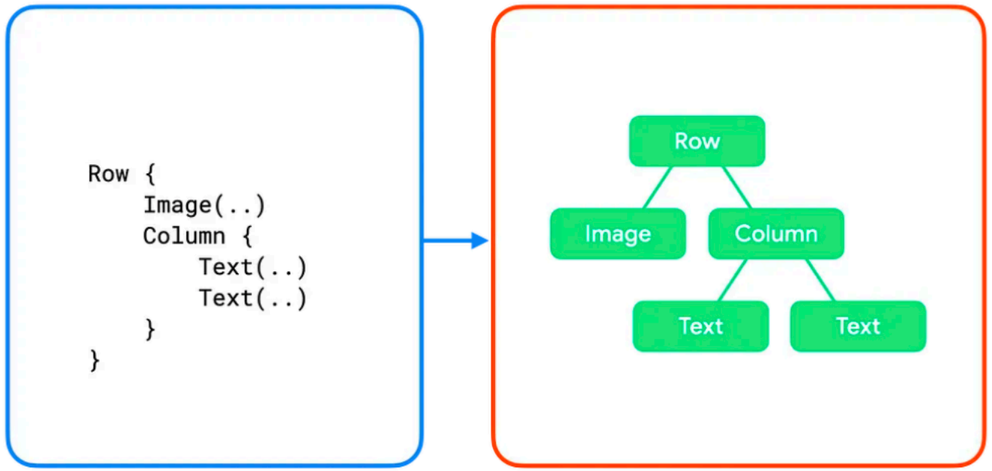
— Gap buffer data structure



Jetpack Compose UI

Talk is Cheap

```
@Composable
fun Counter() {
    var count by remember { mutableStateOf(0) }

    Button(onClick = { count += 1 }) {
        Text("Count: $count")
    }
}
```


Row {
Image(..)
Column {
Text(..)
Text(..)
}
}
Row
Image
Column
Text
Text

Compose is, at its core, a general-purpose tool for managing a tree of nodes of any type … a “tree of nodes” describes just about anything, and as a result Compose can target just about anything.

— Jake Wharton

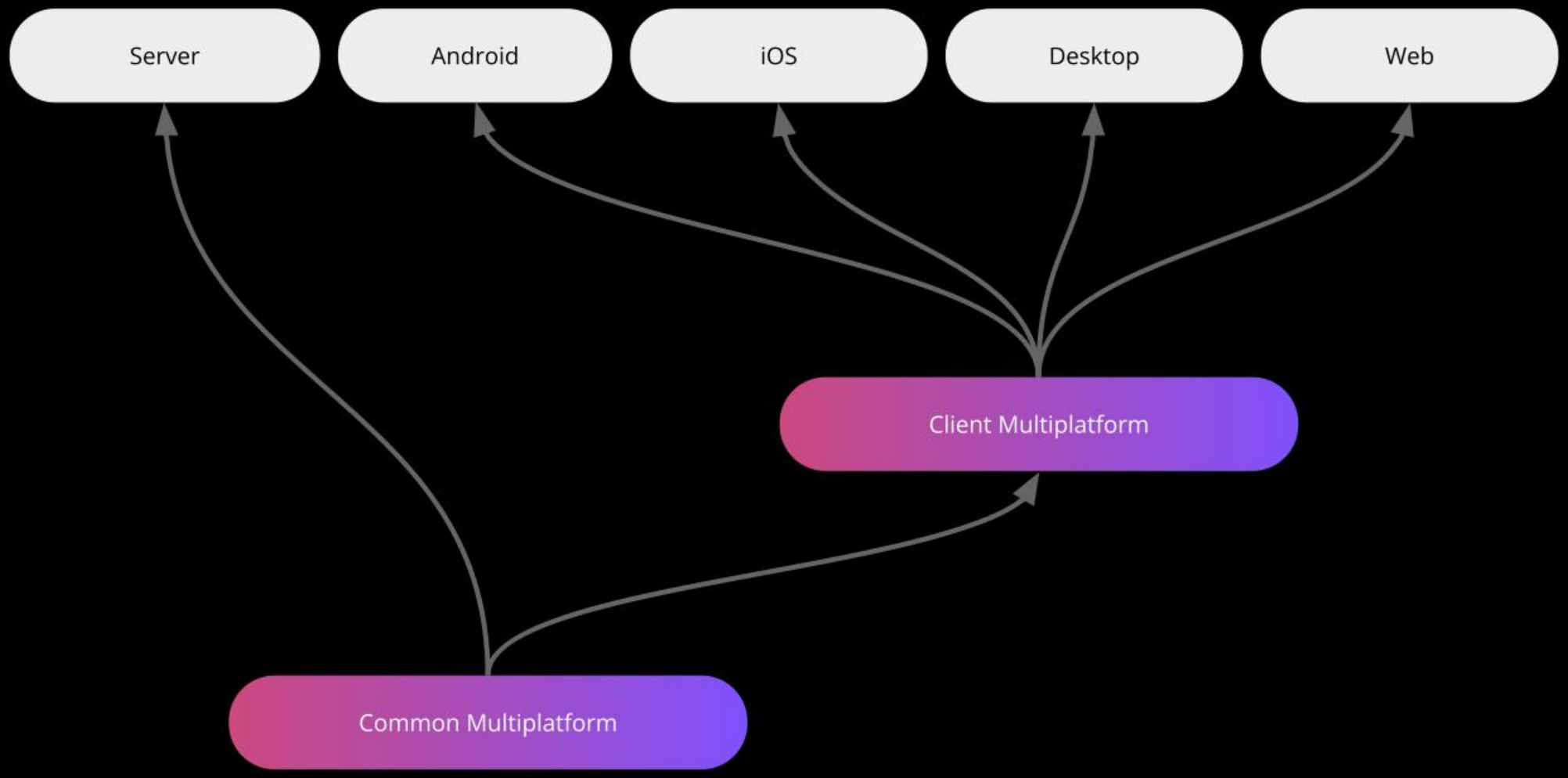
⚠️ Compose != Compose UI

Kotlin Multiplatform

Stable (1.9.20)

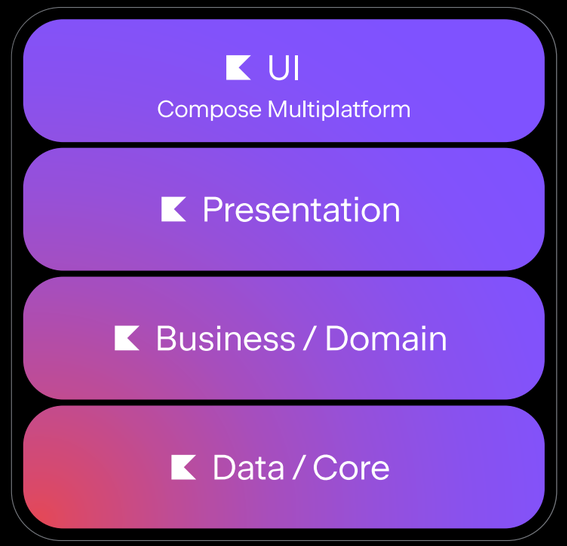
The Before Times

The Before Times


Server
Android
iOS
Desktop
Web
Client Multiplatform
Common Multiplatform

Compose Multiplatform

v1.0 | 2021



JetBrains/**compose-multiplatform-core**

Development environment for Android Jetpack extension libraries under the androidx namespace. Synchronized with Android Jetpack's primary development branch on AOSP.

0	0	541	87
Contributors	Issues	Stars	Forks

Compose Multiplatform Migration

— Change artifact coordinates
— Do nothing
— Profit

```
// Compiled Compose code

fun Counter($composer: Composer) {
    $composer.startRestartGroup(-1913267612)

    /* ... */

    $composer.endRestartGroup()
}

// Compiled Coroutines code

fun counter($completion: Continuation) {
    /* ... */
}
```

KotlinX Coroutines

```
@Suppress("DEPRECATION")
class CallbackLoginPresenter(
  private val service: SessionService,
  private val goTo: (Screen) -> Unit,
) {
  /* ... */

  inner class LoginAsyncTask : AsyncTask<Submit,Void,LoginResult>() {
    private var username: String = ""

    override fun doInBackground(vararg events: Submit?): LoginResult {
      val event = events[0]!!
      username = event.username
      return runBlocking { service.login(event.username, event.password) }
    }

    override fun onPostExecute(result: LoginResult?) {
      when (result) {
        is Success -> goTo(LoggedInScreen(username))
        is Failure -> goTo(ErrorScreen(result.throwable?.message ?: ""))
        else -> {}
      }
    }
  }
}
```

```
Observable.just("Hey")
    .subscribeOn(Schedulers.io())
    .map(String::length)
    .subscribeOn(Schedulers.computation())
    .observeOn(AndroidSchedulers.mainThread())
    .doOnSubscribe { doAction() }
    .flatMap {
        doAction()

        Observable.timer(1, TimeUnit.SECONDS)
            .subscribeOn(Schedulers.single())
            .doOnSubscribe { doAction() }
    }
    .subscribe { doAction() }
```

KotlinX Coroutines

— Lightweight memory usage
— Structured concurrency
— Cancellation propagation
— Lifecycle aware

KotlinX Coroutines

- Native library
- Imperative syntax
- `suspend fun`

Reactive Architecture

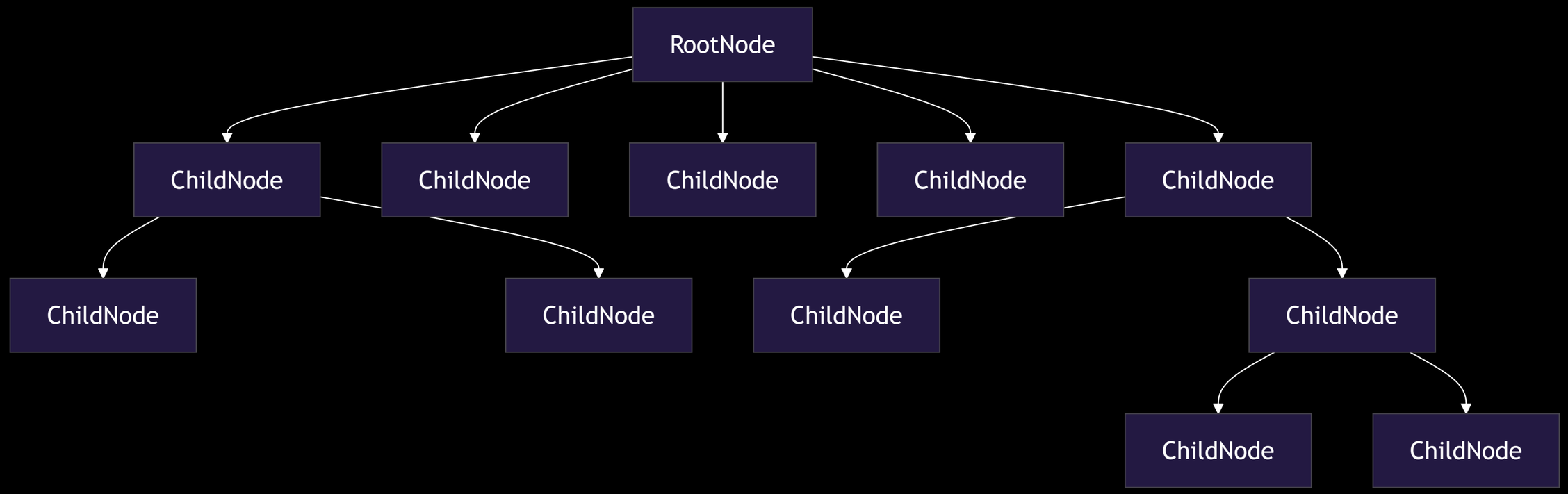
- Push (not pull)
- Unidirectional Data Flow
- Declarative
- Idempotent

```
downloadManager
    .downloadFile("https://.../")
    .addOnCompletionListener { result ->
        fileManager
            .saveFile("storage/file", result)
            .addOnCompletionListener { success ->
                if (success) {
                    println("Downloaded file successfully")
                }
            }
    }
}
```

```
downloadManager
    .downloadFile("https://.../")
    .flatMap { result ->
        fileManager.saveFile("storage/file", result)
    }
    .observe { success ->
        if (success) {
            println("Downloaded file successfully")
        }
    }
```

```
val file = downloadFile("https://.../")
val success = fileManager.saveFile("storage/file", file)

if (success) {
    println("Downloaded file successfully")
}
```

```
downloadManager.
    downloadFile("https://.../")
    .flatMapLatest { state ->
        when (state) {
            is State.Loaded -> stateFileManager.saveFile("storage/file", state.value)
            else -> state
        }
    }
    .collect { state ->
        when (state) {
          is State.Saved -> println("Downloaded file successfully")
          is State.Loading -> /* ... */
        }
    }
```


RootNode
ChildNode
ChildNode
ChildNode
ChildNode
ChildNode
ChildNode
ChildNode
ChildNode
ChildNode
ChildNode
ChildNode

```
val downloadState = downloadManager
    .downloadFile("https://.../")
    .collectAsState(State.Downloading)

val fileState = when(downloadState) {
    is State.Loaded -> stateFileManager.saveFile("storage/file", downloadState.value)
    else -> downloadState
}

when (fileState) {
    is State.Loading -> /* ... */

    is State.Saved -> LaunchedEffect(fileState) {
        println("Downloaded file successfully")
    }
}
```

cashapp/**molecule**

Build a StateFlow stream using Jetpack Compose

25 Contributors

24 Issues

18 Discussions

2k Stars

91 Forks

Molecule

```
fun CoroutineScope.launchCounter(): StateFlow<Int> {
  return launchMolecule(mode = ContextClock) {
    var count by remember { mutableStateOf(0) }

    LaunchedEffect(Unit) {
      while (true) {
        delay(1_000)
        count++
      }
    }

    count
  }
}
```

Testing

```
@Test
fun counter() = runTest {
  moleculeFlow(RecompositionMode.Immediate) {
    Counter()
  }.test {
    assertEquals(0, awaitItem())
    assertEquals(1, awaitItem())
    assertEquals(2, awaitItem())
    cancel()
  }
}
```

Turbine

```
app.cash.turbine:turbine:1.2.0
```

Turbine

```
flowOf("one", "two").test {
  assertEquals("one", awaitItem())
  assertEquals("two", awaitItem())
  awaitComplete()
}
```

Role of Architecture

Pre-Compose Era

Tooling in Compose MPP

— 🔧 Decompose (Navigation, Lifecycle)

— 🧬 Molecule (State modeling)

— ⚡ Circuit (Navigation, State management)

— 🪞 Voyager / Appyx (Navigation alternatives)

— 🌈 Kamel (Image loading)

— 🧪 Paparazzi / Snapshot testing (UI validation)

Navigation with Decompose

- Declarative component hierarchy
- State hoisting via ViewModels (multiplatform-friendly)
- Back stack management without fragments
- Integration with Compose UI and Compose for Web/Desktop

slackhq/**circuit**

⚡ A Compose-driven architecture for Kotlin and Android applications.

33 Contributors | 14 Issues | 86 Discussions | 2k Stars | 91 Forks

Circuit

- Supports most supported KMP platforms
- Compose first architecture
- Presenter & UI separation
- Unidirectional Data Flow

Why Compose Multiplatform?

Shared UI Logic

— Write once, run on Android, Desktop, iOS, Web
— Avoid duplicating presentation logic

Unified State Handling

— Share ViewModels or Presenters across platforms
— Keep logic and state in sync across UIs

Fast Prototyping

— Quickly ship UI to multiple form factors
— Desktop becomes a testbed for mobile UIs

Compose Beyond Visual UI

— Not just visual layout
— Great for business logic and reactive workflows

What Compose MPP Enables

Consistent State Patterns

— Hoisting, unidirectional data flow
— Shared reactive state handling

Shared Design System

— Material components

— Typography, spacing, theming — once

IDE-First Experience

— JetBrains tools tightly integrated

— Live previews and navigation supported

Kotlin Multiplatform Ecosystem

- Compose integrates easily with:
  - `Ktor` for networking
  - `Kotlinx.serialization` for models
  - `Decompose`, `Essenty` for navigation/state

Compose Runtime Beyond UI

Composables Are Reactive Functions

— Input = State

— Output = UI + Side-effects

Ideal Use Cases

— Finite State Machines
— Flow Orchestration
— Deterministic State Testing

Wrap-Up: Why This Matters

✅ Compose is more than a UI toolkit
✅ Enables scalable, shared architecture
✅ Designed for Kotlin-first developers
✅ Multiplatform is no longer just business logic

→ Start rethinking how you architect apps, not just how you render them.

Thank You!

Ash Davies

Android GDE Berlin