Beyond the UI

Compose as a Foundation for Multiplatform Apps

mDevCamp - June '25 🇨🇿

Ash Davies

Android GDE Berlin

ashdavies.dev

What is Jetpack Compose UI

When was Compose UI introduced?

- History of XML layouts in Android
- Android developers unhappy with XML

What projects suggested a problem with the status quo?

- DataBinding & ViewBinding
- Community libraries (Anko, Splitties)

What prompted a rethink of Android UI tooling?

— Widespread adoption of Kotlin on Android
— Kotlin language features

What are the principles of Compose?

- Declarative
- Open Source
- Idiomatic

How does it work?

— Kotlin compiler plugin
— Cooperation with Jetbrains
— Manipulates method signatures

What does it look like?

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

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

What does this code remind us of?

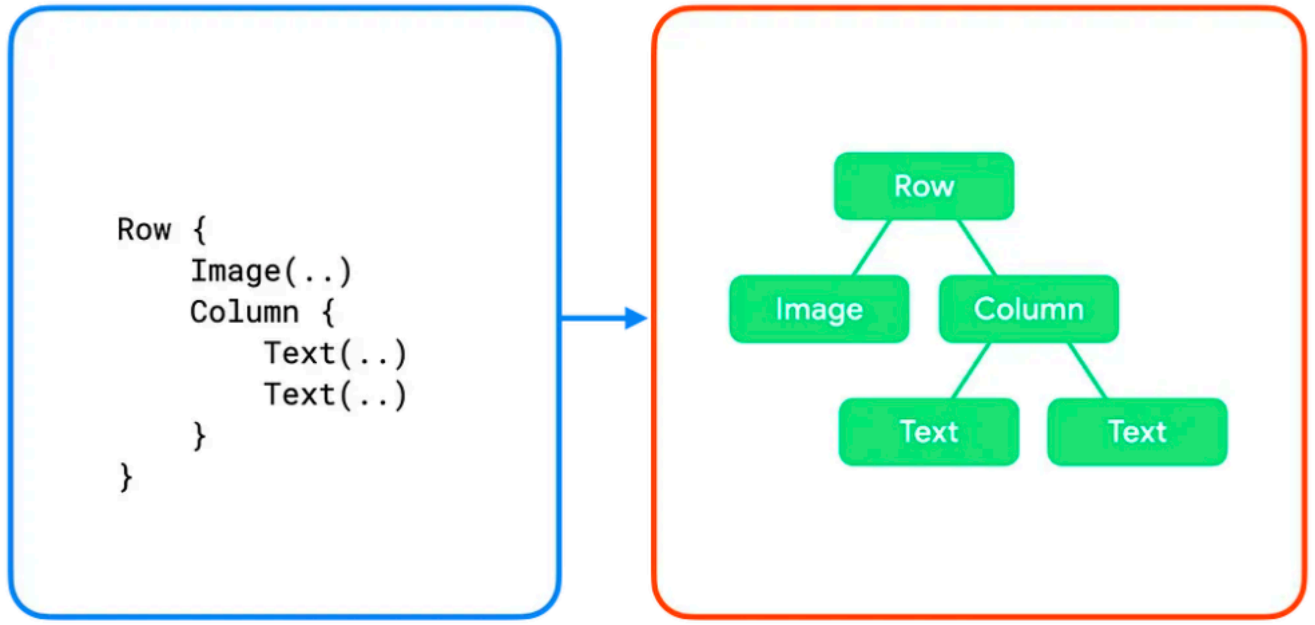
```
fun Counter($composer: Composer) {
    $composer.startRestartGroup(-1913267612)

    /* ... */

    $composer.endRestartGroup()
}
```

How is the KotlinX Coroutine code manipulation similar?

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



```
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

jakewharton.com/a-jetpack-compose-by-any-other-name

⚠️ Compose != Compose UI

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

    /* ... */

    $composer.endRestartGroup()
}
```

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

Remember the problems coroutines were meant to solve?

- Reactive pipelines
- Explicit thread handling
- Inline error-handling
- Lifecycle awareness

- Native library
- Imperative syntax
- `suspend fun`

Reactive Architecture

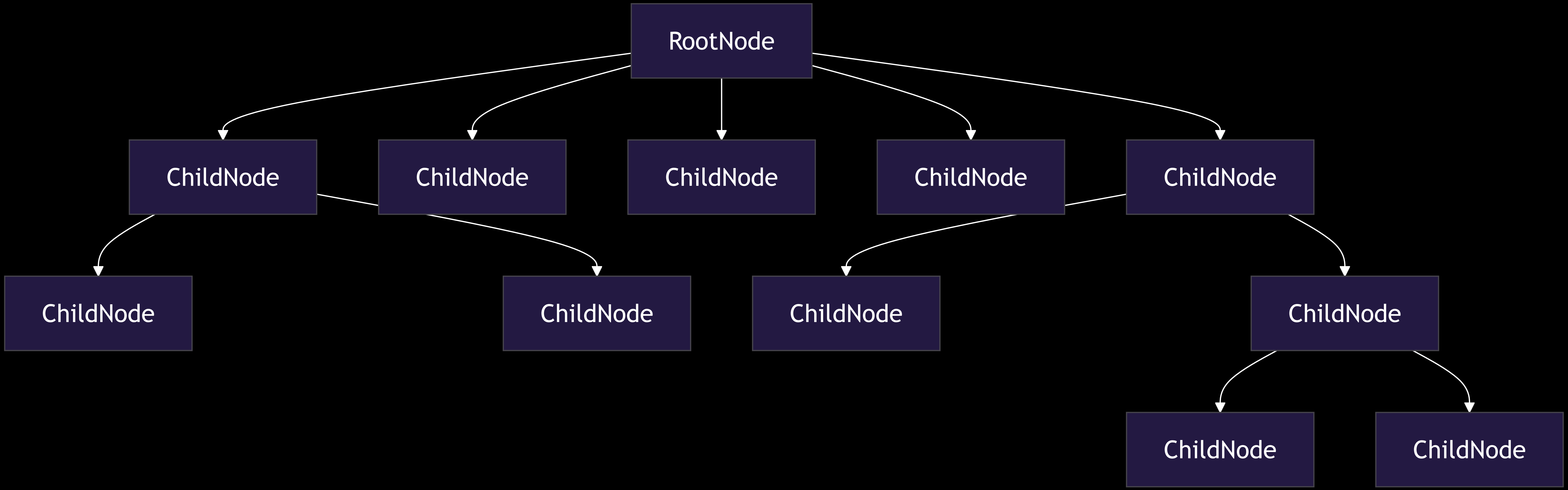
- Push (not pull)
- Unidirectional Data Flow
- Declarative
- Idempotent

```kotlin
downloadManager.downloadFile("https://.../") { result ->
  fileManager.saveFile("storage/file", result) { 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") }
```

```kotlin
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.Loading -> /* ... */
      is State.Saved -> println("Downloaded file successfully")
    }
  }
```


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

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

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

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

Molecule

github.com/cashapp/molecule

Molecule

```kotlin
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

Role of Architecture

Pre-Compose Era

Tooling in Compose MPP

- 🔧 Decompose (Navigation, Lifecycle)
- 🧬 Molecule (State modeling)
- 🪞Voyager / Appyx (Navigation alternatives)
- 🔄 Reaktive / Flow / StateFlow (State Streams)
- 🌈 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

Circuit

github.com/slackhq/circuit

Circuit

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

History of Multiplatform

Why Compose Multiplatform?

- Shared UI logic across Android, Desktop, iOS, Web
- Unified state handling with shared ViewModels or Presenters
- Faster prototyping across form factors
- Composable tooling beyond visual UI (state, business logic)

Compose MPP Enables

- Consistent state handling across platforms
- Shared design system (e.g., Material)
- Deep JetBrains IDE integration
- Integration with Kotlin Multiplatform (KMP) libraries:
  - `Ktor`, `Kotlinx.serialization`, `Decompose`, `Essenty`

Compose Runtime beyond UI

- Composables as reactive functions
- Ideal for:
    - Finite State Machines
    - Orchestration Logic
    - Testing state changes deterministically

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