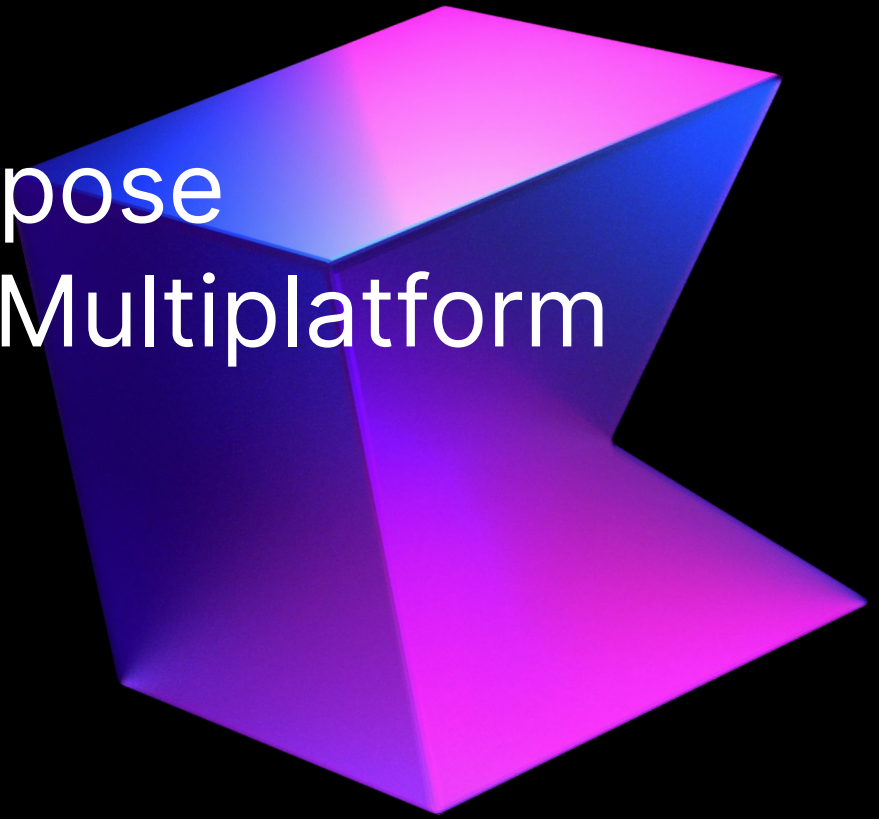
Kotlin

从 Jetpack Compose 到 Compose Multiplatform



王鹏
Android Engineer
ByteDance

Nov 26, 2022

Contents

Jetpack Compose

↓

Compose Runtime

↓

Compose Multiplatform

↓

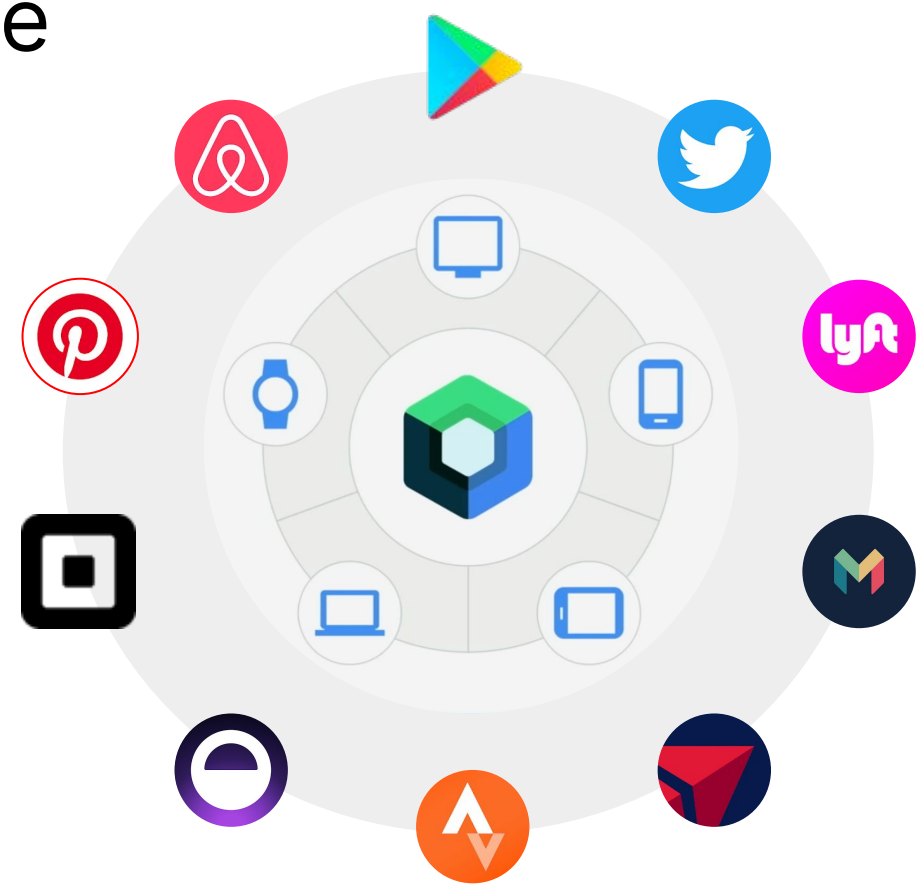
Example Project

Jetpack Compose

Android Native UI

**Phone, Pad,
Watch, TV, etc**

**Play Store
160+ / Top1K**



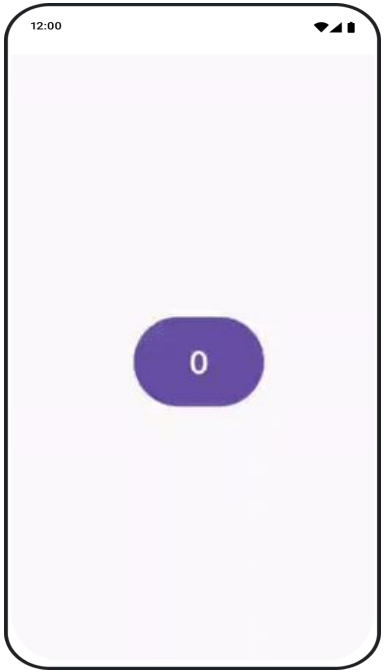
Jetpack Compose

```
@Composable
fun Counter() {

    var count by remember { mutableStateOf(0) }

    Button(
        onClick = { count++ }
    ) {
        Text("$count")
    }

}
```

Declarative

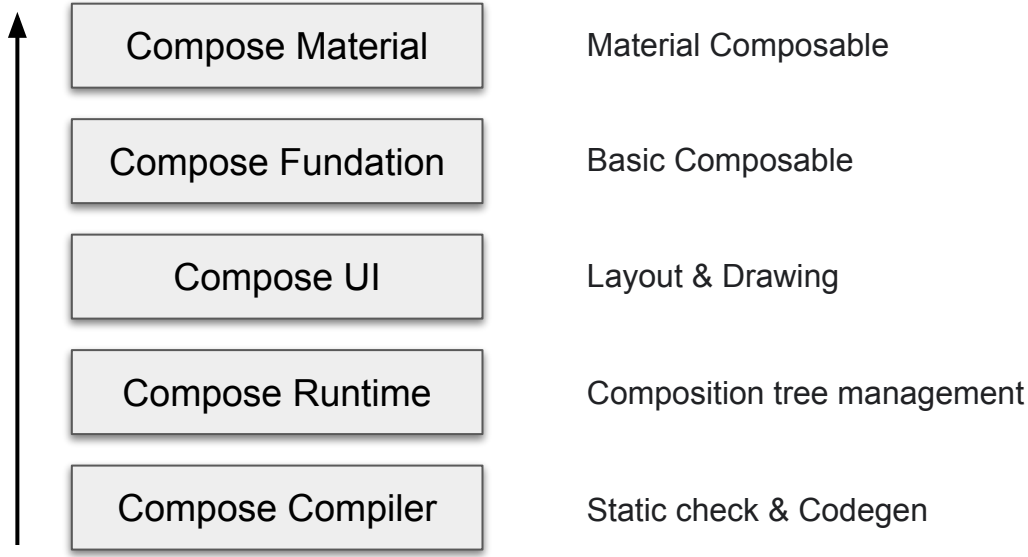
Functional

State driven



Jetpack Compose

Compose Architecture Layers



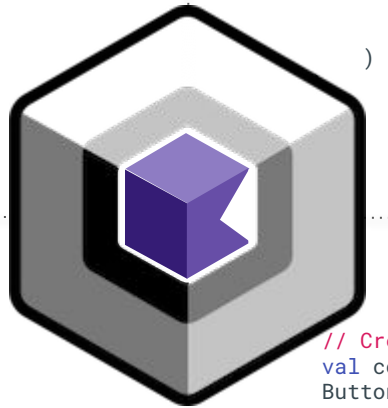
Kotlin for Compose

```
Button(
    // ...
    onClick = {
        // do something
        // do something else
    }
) { /* ... */ }
```

Higher-order functions and lambda expressions

```
Row {//RowScope
    Text(
        text = "Hello world",
        // This Text inside a RowScope can access to
        // Alignment.CenterVertically but not to
        // Alignment.CenterHorizontally
        modifier = Modifier.align(
               Alignment.CenterVertically)
    )
```

Scopes and receivers



Delegated properties

```
var showDialog by
       remember { mutableStateOf(false) }

// Updating the var automatically
// triggers a state change
showDialog = true
```

Kotlin Coroutines

```
// Create a CoroutineScope that follows composable's lifecycle
val composableScope = rememberCoroutineScope()
Button(
    // ...
    onClick = {
        composableScope.launch {
            //call suspend function
        }
    }
) { /* ... */ }
```

Less code

```dart
//Flutter with Dart
class Counter extends StatefulWidget {
  @override
  _Counter createState() => _Counter();
}

class _Counter extends State<Counter> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return TextButton(
      onPressed: _incrementCounter,
      child: Text("$_counter"),
    );
  }
}
```

VS

```kotlin
//Compose with Kotlin
@Composable
fun Counter() {

    var count by remember { mutableStateOf(0) }

    Button(
        onClick = { count++ }
    ) {
        Text("$count")
    }

}
```

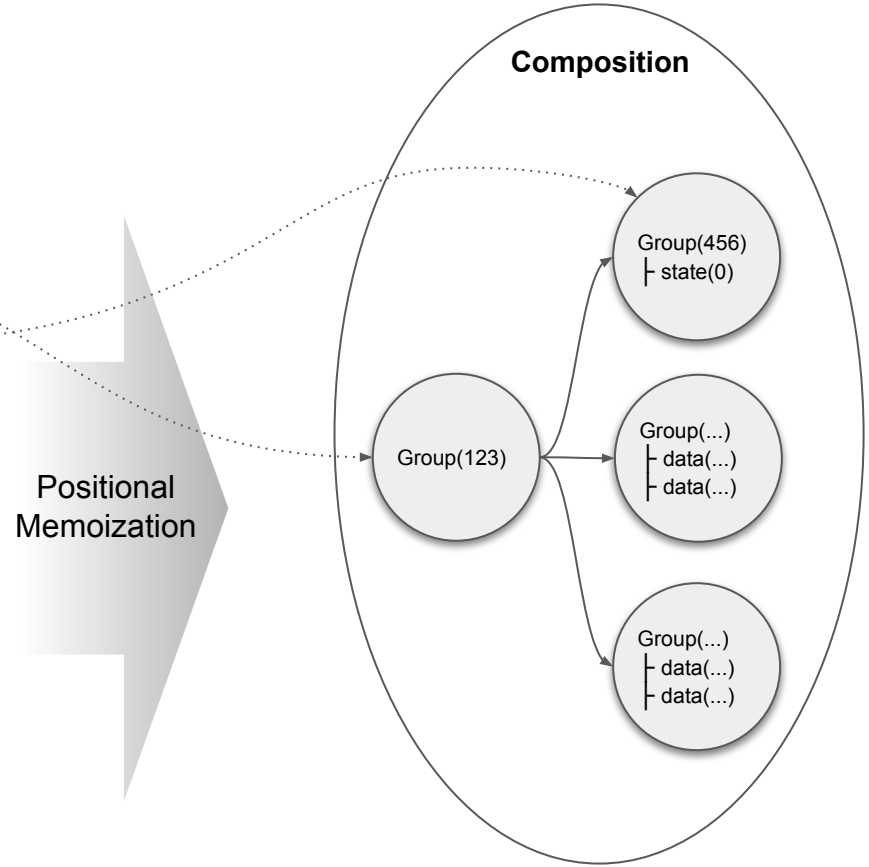
Compose Compiler

```
@Composable
fun Counter() {

    val count by remember { mutableStateOf(0) }

    Button(
        onClick = { count++ }
    ) {
        Text("${count}")
    }

}
```

```
@Composable
fun Counter($composer: Composer,  $changed: Int) {

    $composer.startRestartGroup(123)

    if ($changed != 0) {
        $composer.startReplaceableGroup(456)
        var count by remember { mutableStateOf(0) }
        $composer.endReplaceableGroup()

        Button(
            onClick = { count++ }
        ) {
            Text("$count")
        }
    } else {
        $composer.skipToGroupEnd()
    }

    $composer.endRestartGroup()?.updateScope {
        Counter($composer, 1)
    }

}
```

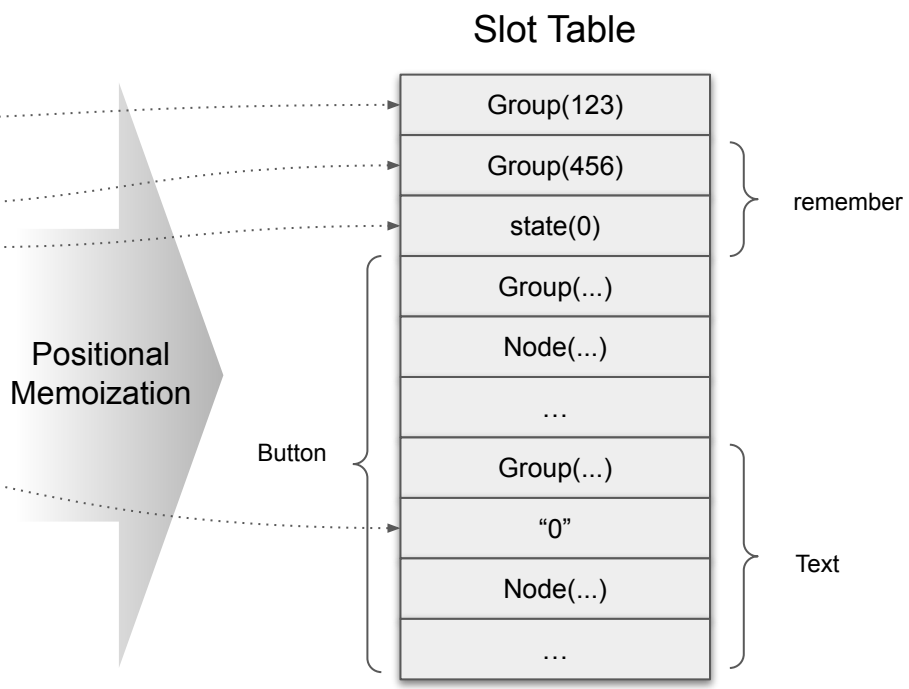
Compose Runtime

```
@Composable
fun Counter($composer: Composer,  $changed: Int) {

    $composer.startRestartGroup(123)

    if ($changed != 0) {
        $composer.startReplaceableGroup(456)
        var count by remember { mutableStateOf(0) }
        $composer.endReplaceableGroup()

        Button(
            onClick = { count++ }
        ) {
            Text("$count")
        }
    } else {
        $composer.skipToGroupEnd()
    }

    $composer.endRestartGroup()?.updateScope {
        Counter($composer, 1)
    }

}
```

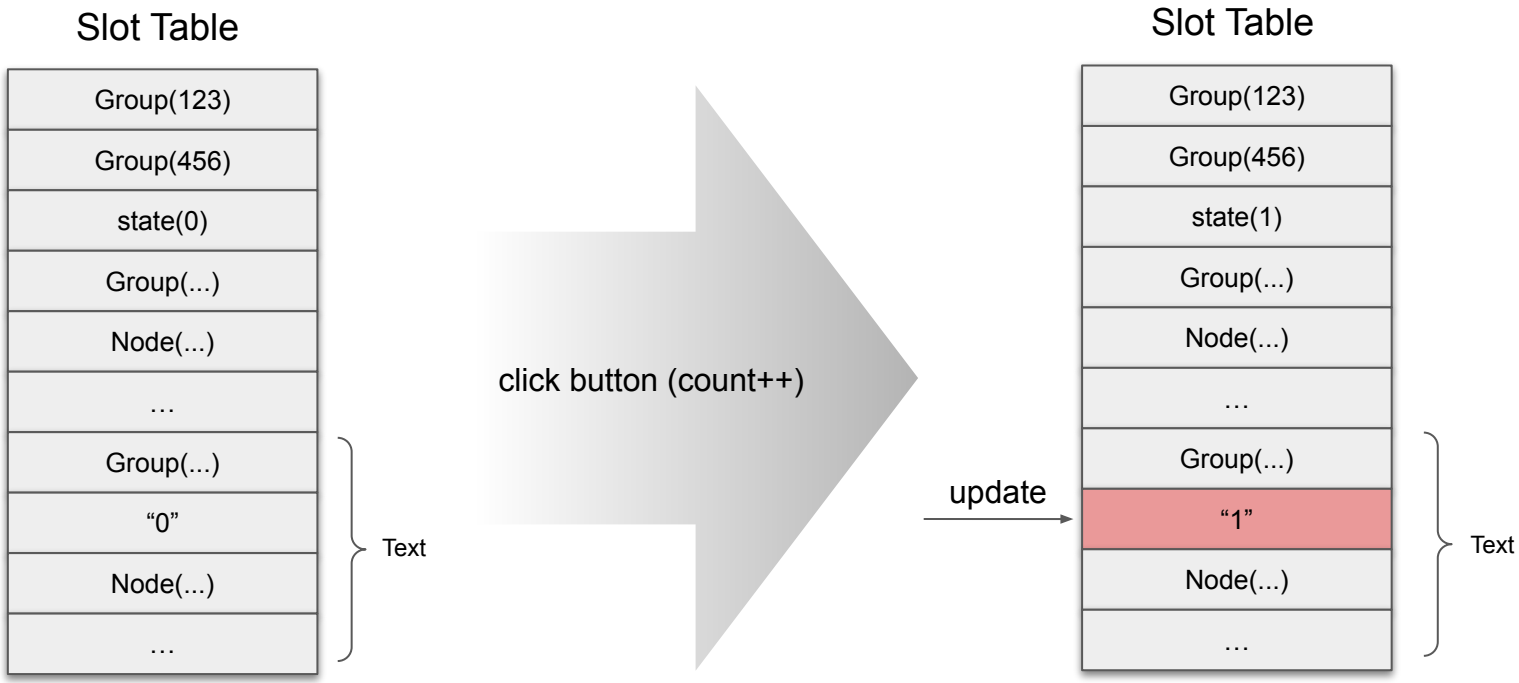


Compose Runtime

```
@Composable
fun Counter($composer: Composer,  $changed: Int) {

    $composer.startRestartGroup(123)

    if ($changed != 0) {
        $composer.startReplaceableGroup(456)
        var count by remember { mutableStateOf(0) }
        $composer.endReplaceableGroup()

        Button(
            onClick = { count++ }
        ) {
            Text("$count")
        }
    } else {
        $composer.skipToGroupEnd()
    }

    $composer.endRestartGroup()?.updateScope {
        Counter($composer, 1)
    }

}
```



Recomposition



Applier & Node Tree

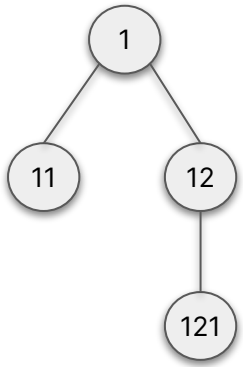
```
interface Applier<Node> {
    fun insertTopDown(index: Int, instance: Node)
    fun insertBottomUp(index: Int, instance: Node)
    fun remove(index: Int, count: Node)
    fun move(from: Int, to: Int, count: Int)
    fun onClear()
}
```

Slot Table

Node(1)
Node(11)
Node(12)
Node(121)
{ Gap Buffer }

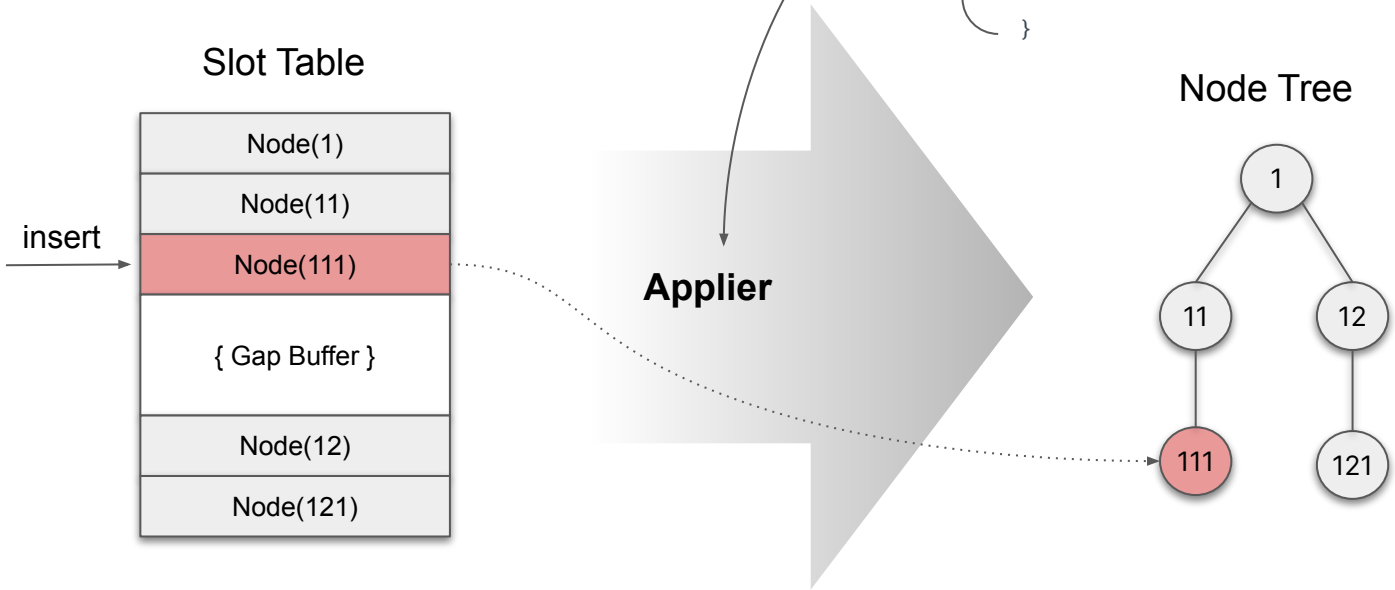
Applier

Node Tree

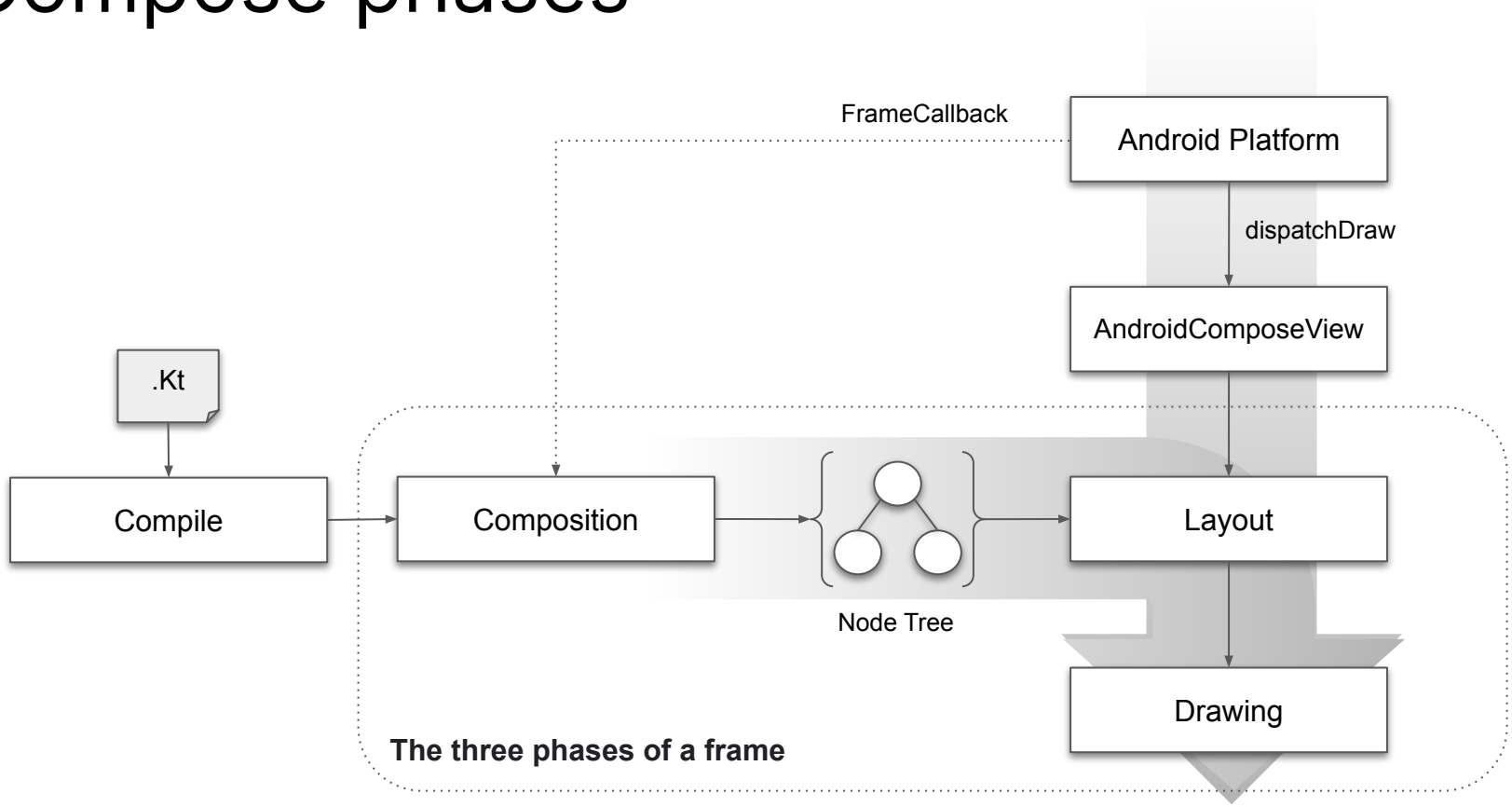


Applier & Node Tree

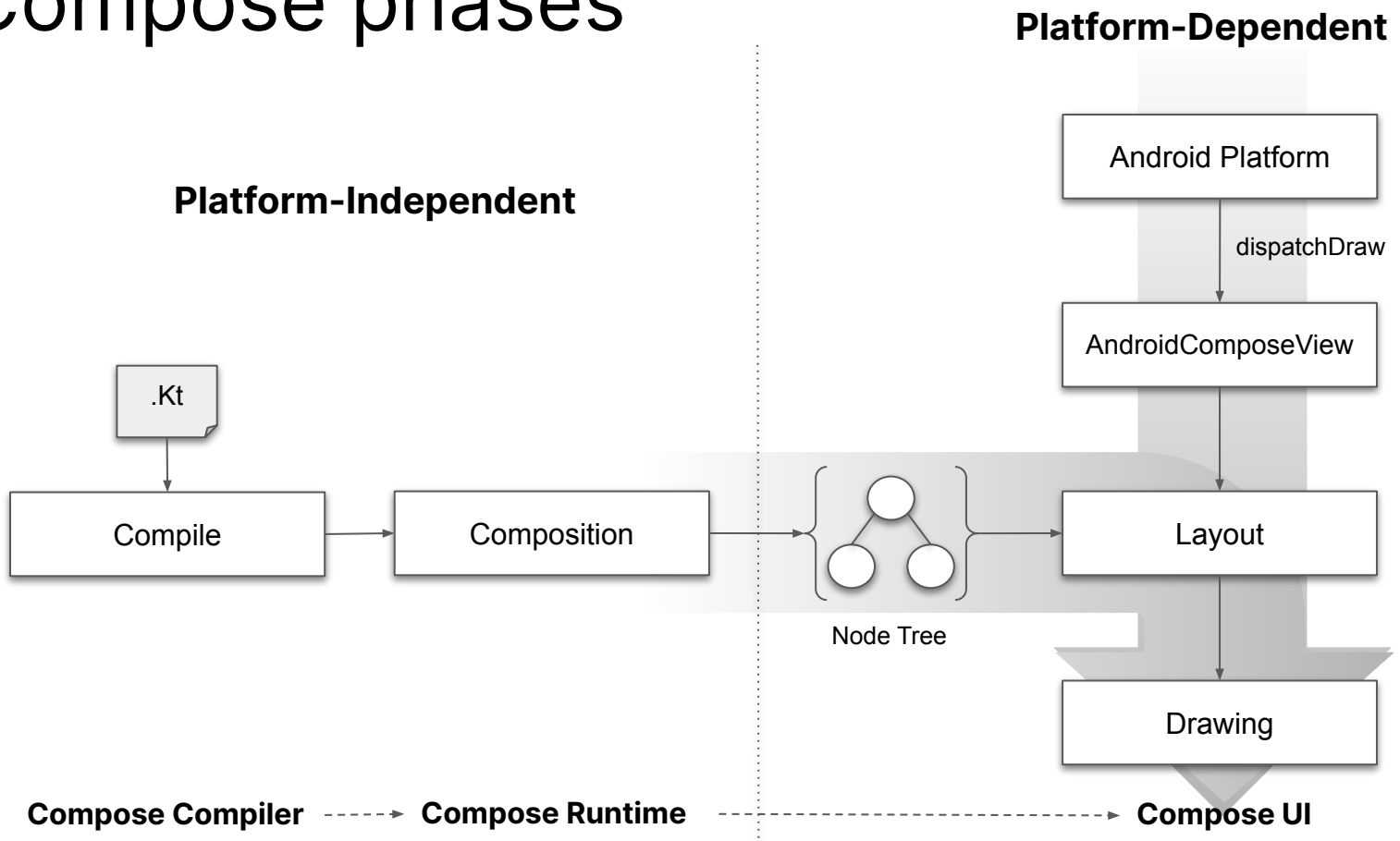
```
interface Applier<Node> {
    fun insertTopDown(index: Int, instance: Node)
    fun insertBottomUp(index: Int, instance: Node)
    fun remove(index: Int, count: Node)
    fun move(from: Int, to: Int, count: Int)
    fun onClear()
}
```



Compose phases



Compose phases



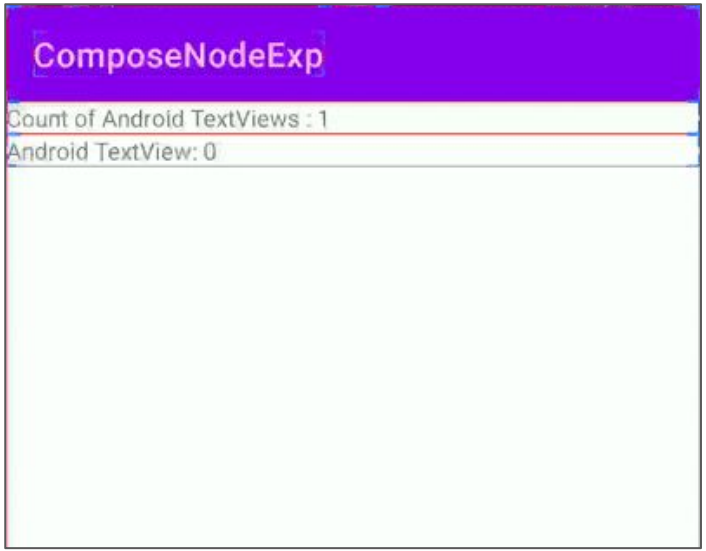
Compose for Android View

```
class ViewApplier(val view: FrameLayout) :
            AbstractApplier<View>(view) {

    override fun onClear() {
        view.removeAllViews()
    }

    override fun insertBottomUp(
            index: Int,
            instance: View) {
        current.addView(instance, index)
    }

    override fun remove(
            index: Int,
            count: Int) {
        view.removeViews(index, count)
    }

    //…

}
```

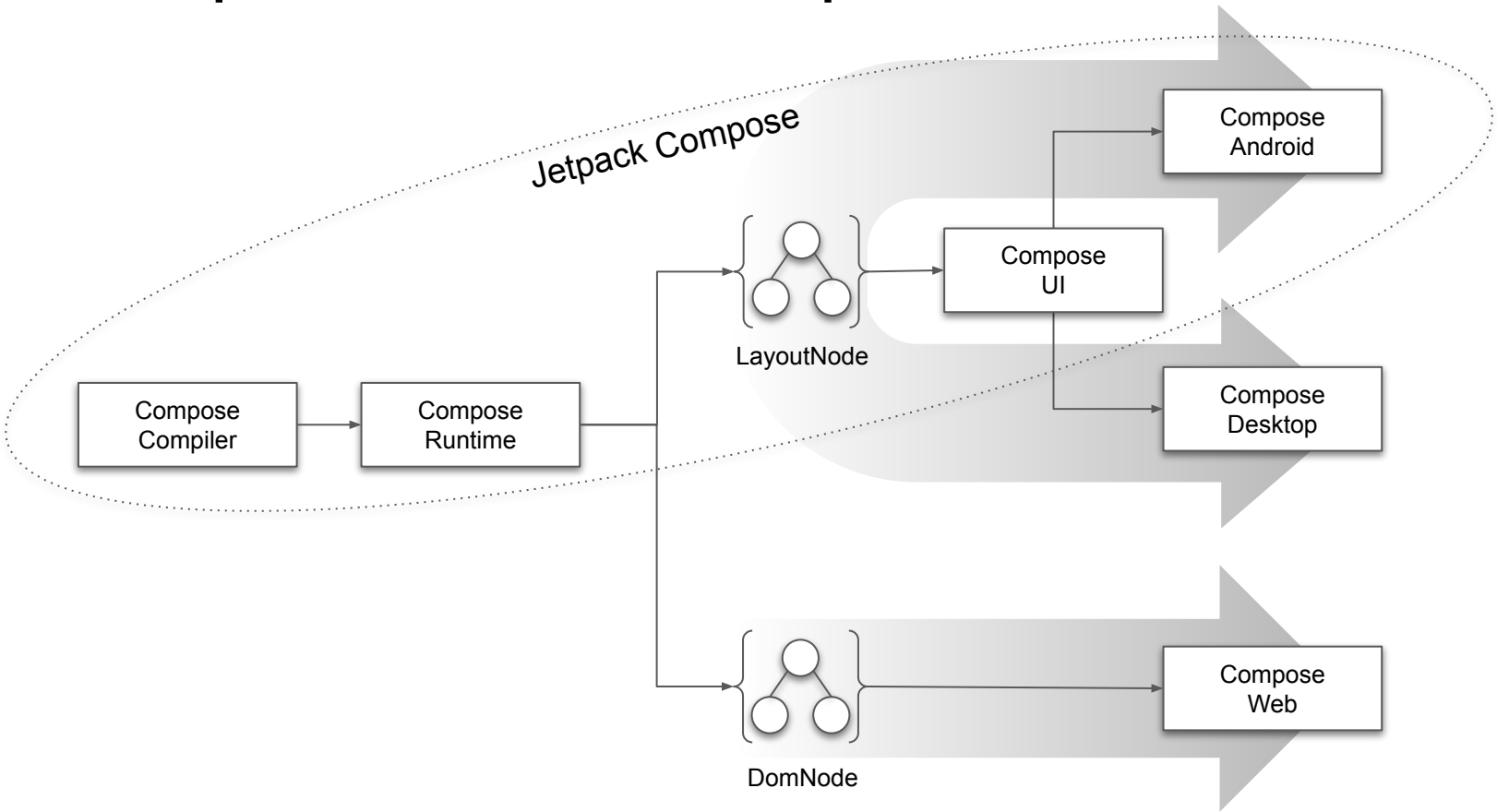
```
@Composable
fun TextView(
    text: String,
    onClick: () -> Unit = {}
) {

    val context = localContext.current

    ComposeNode<TextView, ViewApplier>(
        factory = {
            TextView(context)
        },
        update = {
            set(text) {
                this.text = text
            }
            set(onClick) {
                setOnClickListener { onClick() }
            }
        },
    )

}
```

Compose for Android View

```
@Composable
fun AndroidViewApp() {

    var count by remember { mutableStateOf(1) }

    LinearLayout {
        TextView(
            text = "This is the Android TextView!!",
        )
        repeat(count) {
            TextView(
                text = "Android View!!TextView:$it $count",
                onClick = {
                    count++
                }
            )
        }

    }
}
```

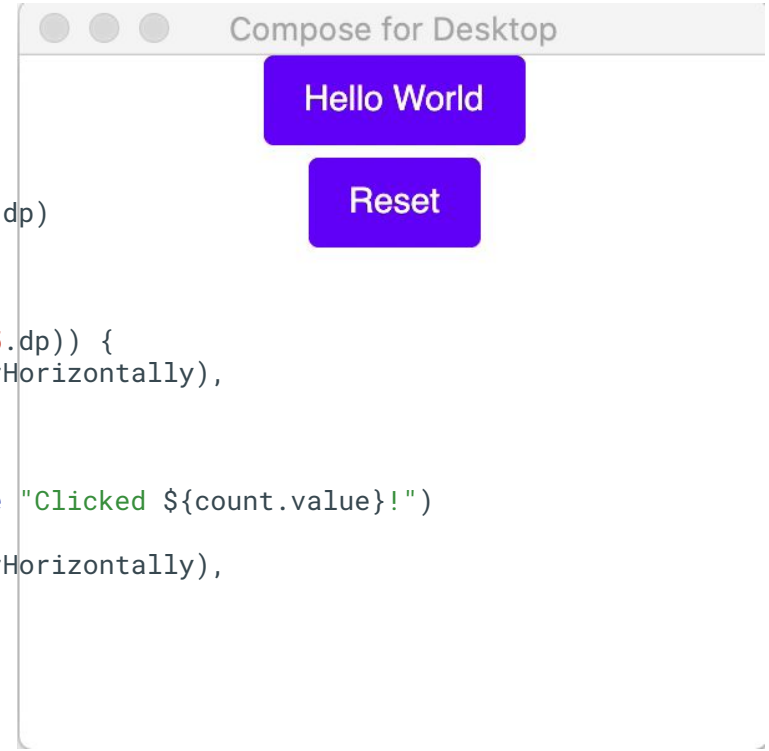


Compose for Desktop & Web



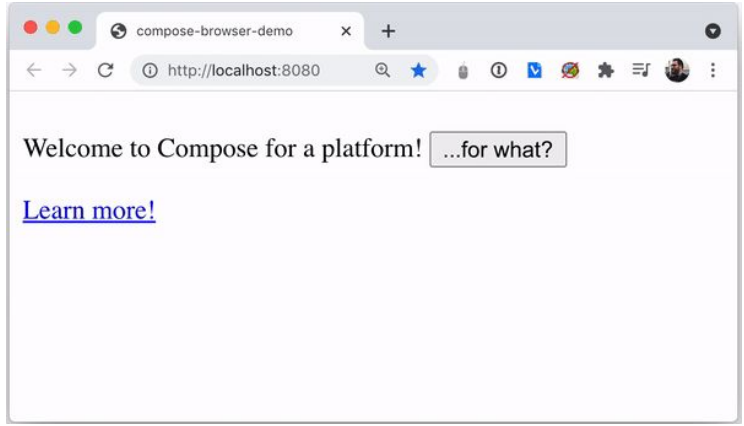
Compose for Desktop

```
fun main() = application {
    Window(
        onCloseRequest = ::exitApplication,
        title = "Compose for Desktop",
        state = rememberWindowState(width = 300.dp, height = 300.dp)
    ) {
        val count = remember { mutableStateOf(0) }
        MaterialTheme {
            Column(Modifier.fillMaxSize(), Arrangement.spacedBy(5.dp)) {
                Button(modifier = Modifier.align(Alignment.CenterHorizontally),
                    onClick = {
                        count.value++
                    }) {
                    Text(if (count.value == 0) "Hello World" else "Clicked ${count.value}!")
                }
                Button(modifier = Modifier.align(Alignment.CenterHorizontally),
                    onClick = {
                        count.value = 0
                    }) {
                    Text("Reset")
                }
            }
        }
    }
}
```

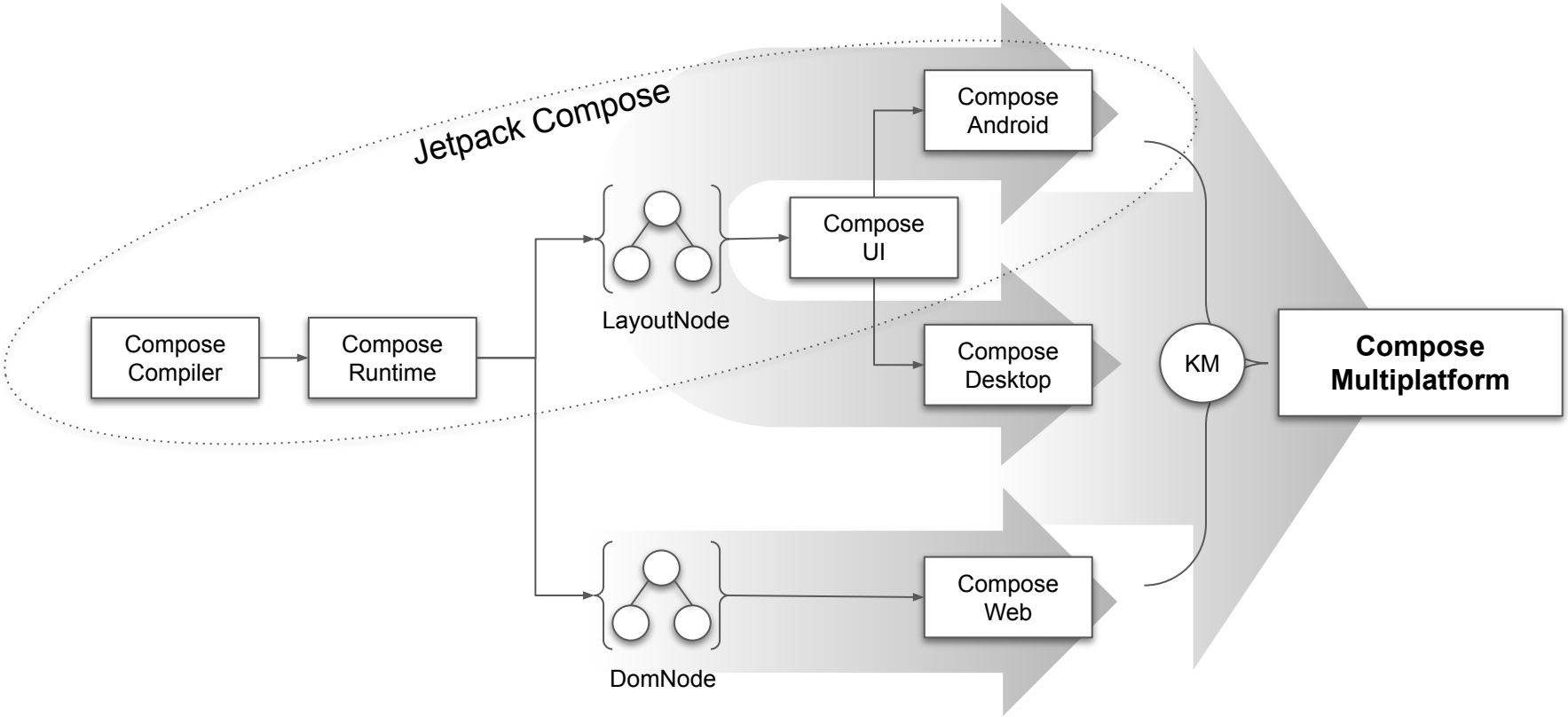


Compose for Web

```
fun main() {
    renderComposable("root") {
        var platform by remember { mutableStateOf("a platform") }

        P {
            Text("Welcome to Compose for $platform! ")
            Button(attrs = { onClick { platform = "Web" } }) {
                Text("...for what?")
            }
        }
        A("https://www.jetbrains.com/lp/compose-web") {
            Text("Learn more!")
        }
    }
}
```

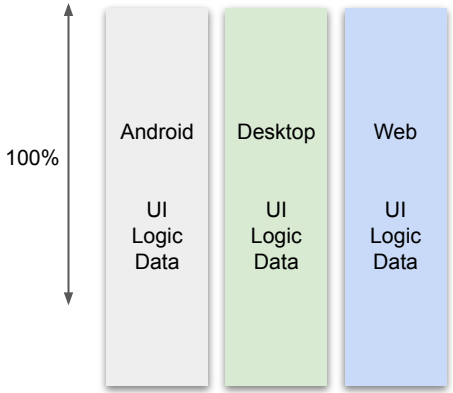


Compose Multiplatform



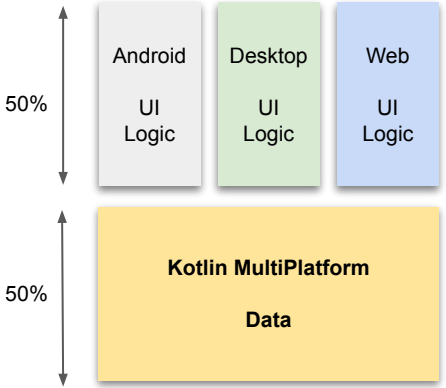
KMP → CMP

Platform-specific only



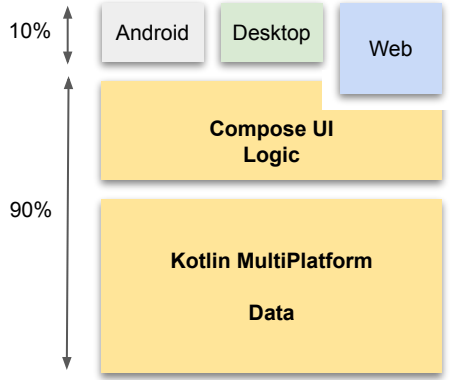
3 x 100% = 300%

Kotlin Multiplatform Project



3 x 50% + 50% = 200%

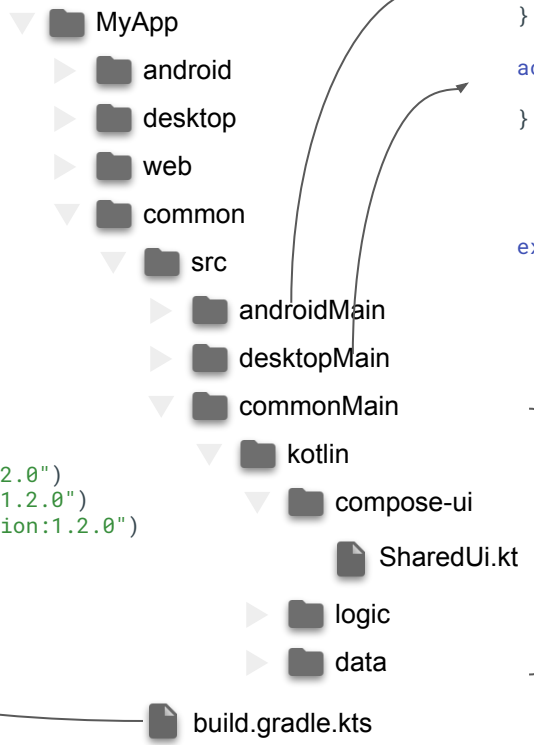
Compose Multiplatform Project



3 x 10% + 90% = 120%

Project Structure

```
sourceSets {

    val androidMain by getting {
        dependencies {
            api("androidx.appcompat:appcompat:1.2.0")
            api("androidx.core:core-ktx:1.3.1")
        }
    }
    val desktopMain by getting {
        dependencies {
            implementation(compose.desktop.common)
        }
    }
     val commonMain by getting {
        dependencies {
            api("org.jetbrains.compose.runtime:runtime:1.2.0")
            api("org.jetbrains.compose.material:material:1.2.0")
            api("org.jetbrains.compose.foundation:foundation:1.2.0")
        }
    }

}
```




```
actual fun getPlatformName(): String {
    return "Android"
}
```

```
actual fun getPlatformName(): String {
    return "Desktop"
}
```

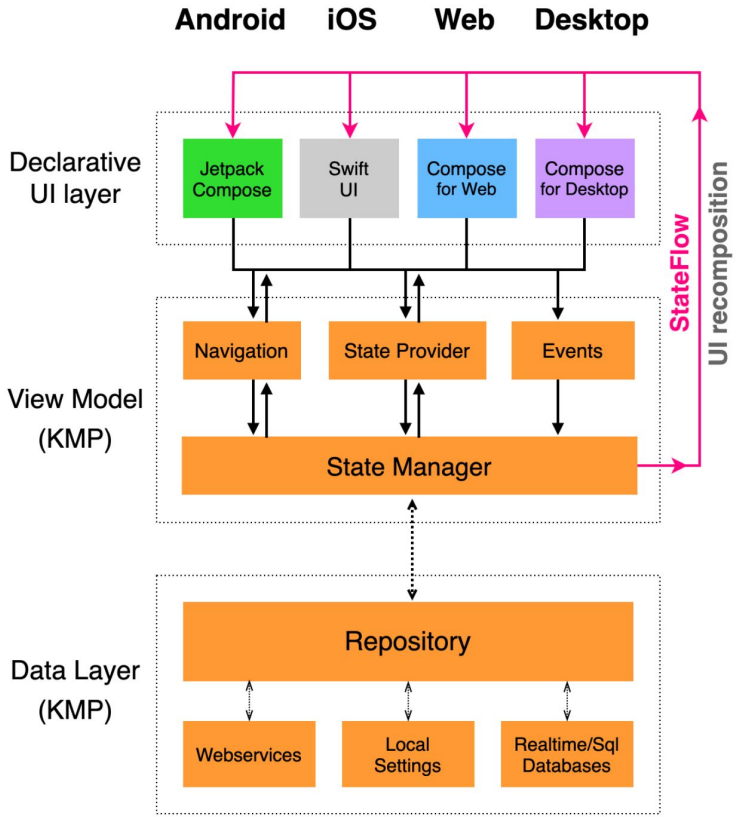
```
expect fun getPlatformName(): String
```

D-KMP Architecture

Declarative UI:
Compose or SwiftUI

ViewModel:
Unidirectional data flow

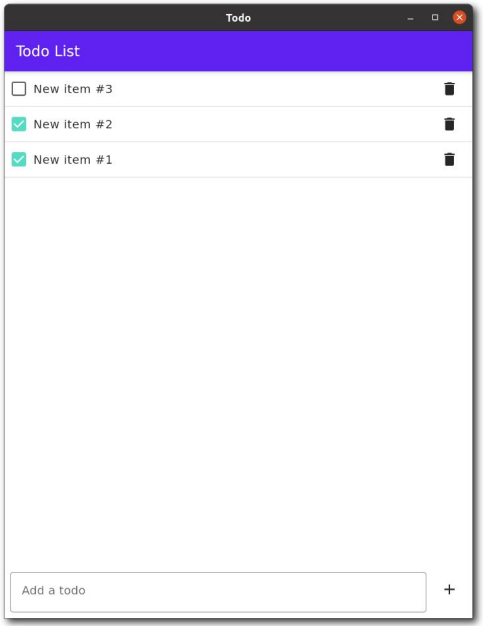
Data Layer:
Repository Pattern



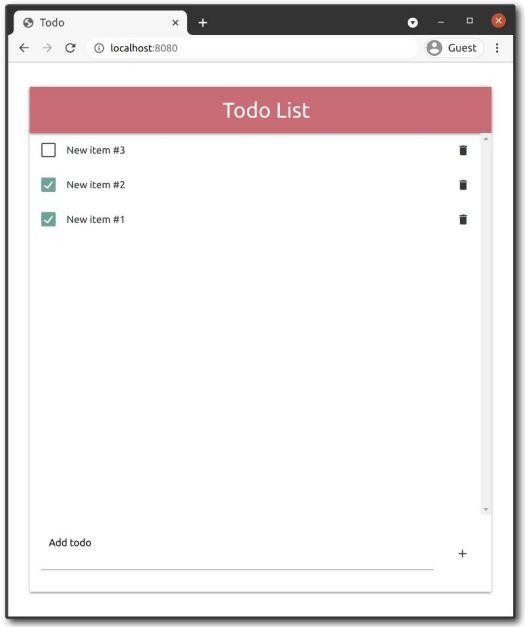
https://github.com/dbaroncelli/D-KMP-sample

Example Project

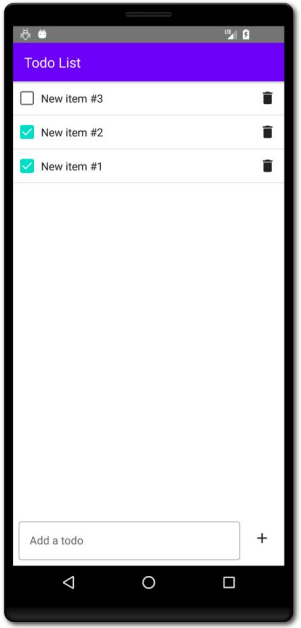
Desktop



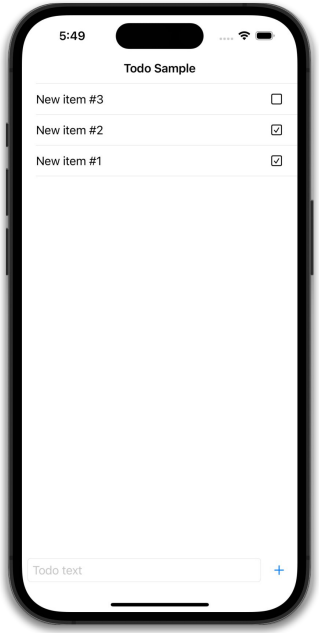
Web



Android



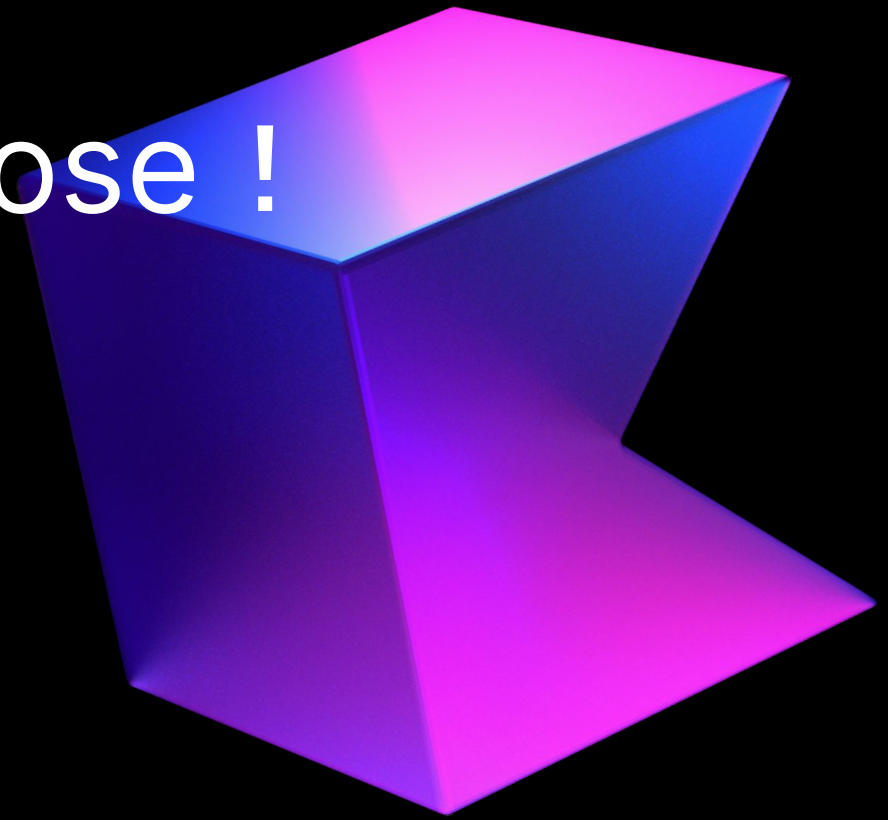
iOS



https://github.com/JetBrains/compose-jb/tree/master/examples/todoapp


Compose
for Multiplatform
Compose
Multiplatform

Thanks!
Happy Compose！



王鹏
Android Engineer
ByteDance