

FROM REACT TO
JETPACK COMPOSE

STEPHEN WHITE

CADEC 2021.01.27 | CALLISTAENTERPRISE.SE

CALLISTA

AGENDA

- What, Why, When
- React Mindset
- Thinking in Compose
- Thinking in React
- Applying React to Compose
- Conclusions



WHAT

WHAT IS JETPACK COMPOSE

- Jetpack Compose is a modern toolkit for building native Android UI.
- Built on Kotlin
- Declarative programming model
- You declare the layout/ look and feel
- As State Changes your UI automatically updates
- Recomposition - Algorithm
- And then there's Android Studio



WHAT IS JETPACK COMPOSE - GET STARTED



DOWNLOAD THE CANARY!

WHY

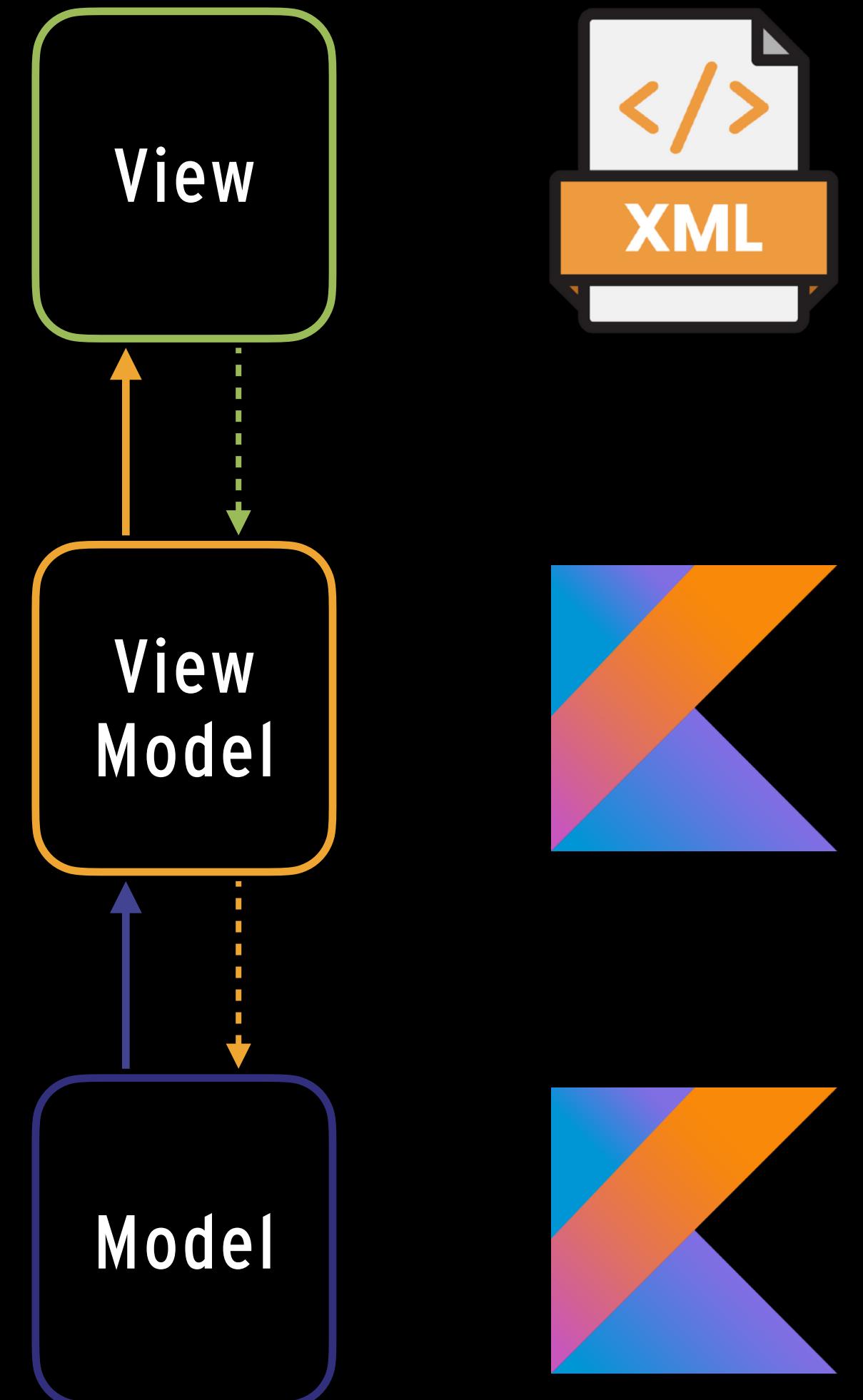
WHY JETPACK COMPOSE

- React!
- Challenge React Native
- Declarative UI
- Faster iterations
- Improved Testability
- Improved Code Reasonability
- Minimise complexity and cost of change
- Draw from a huge community of React developers



WHY JETPACK COMPOSE - MVVM

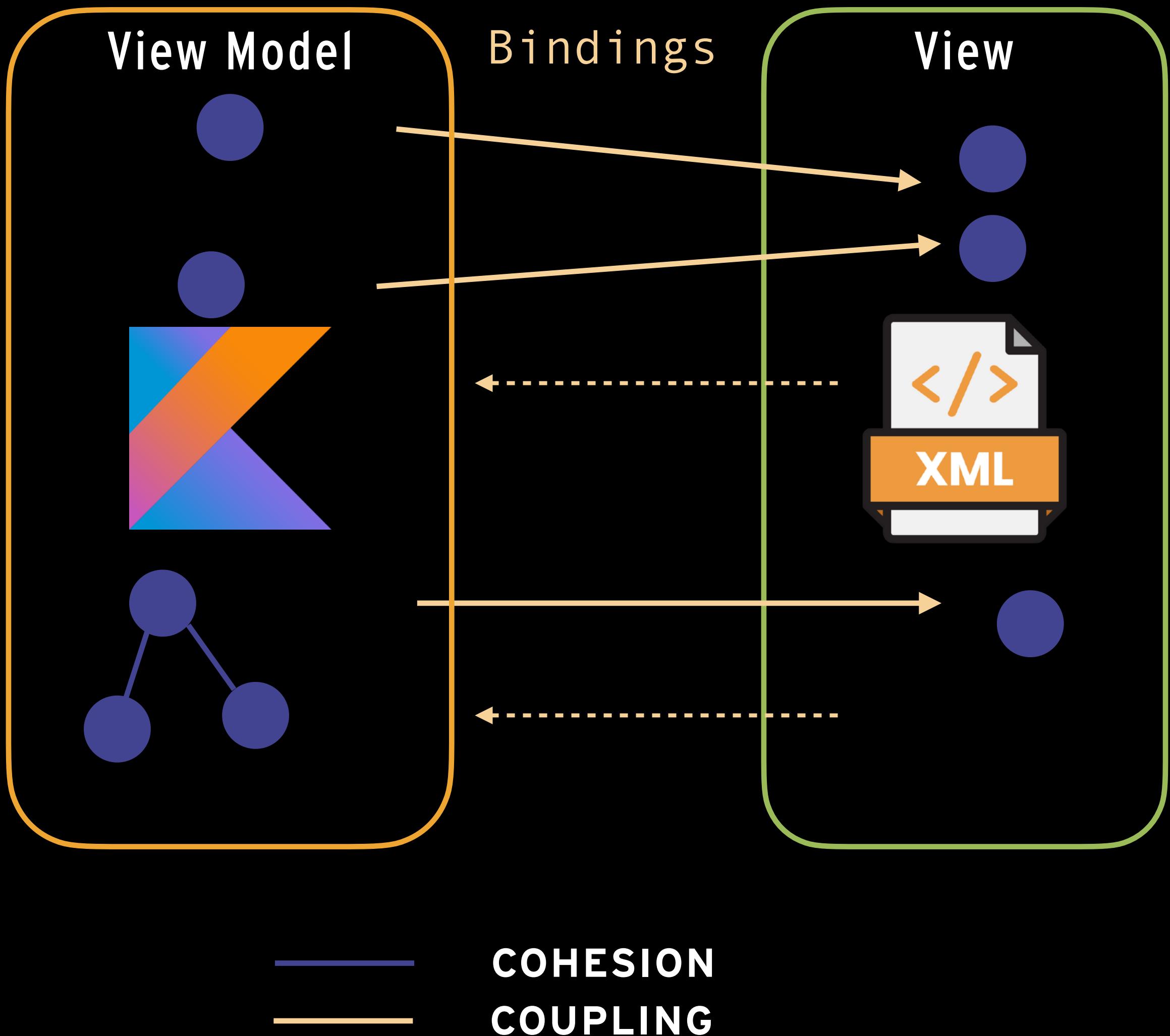
- - View - XML
 - UI of the application, observes data from the View Model
- View Model - Kotlin
 - Link between the View and the model.
- Model - Kotlin
 - Data of the application, can't talk directly to a view



TWO WAY DATA BINDINGS

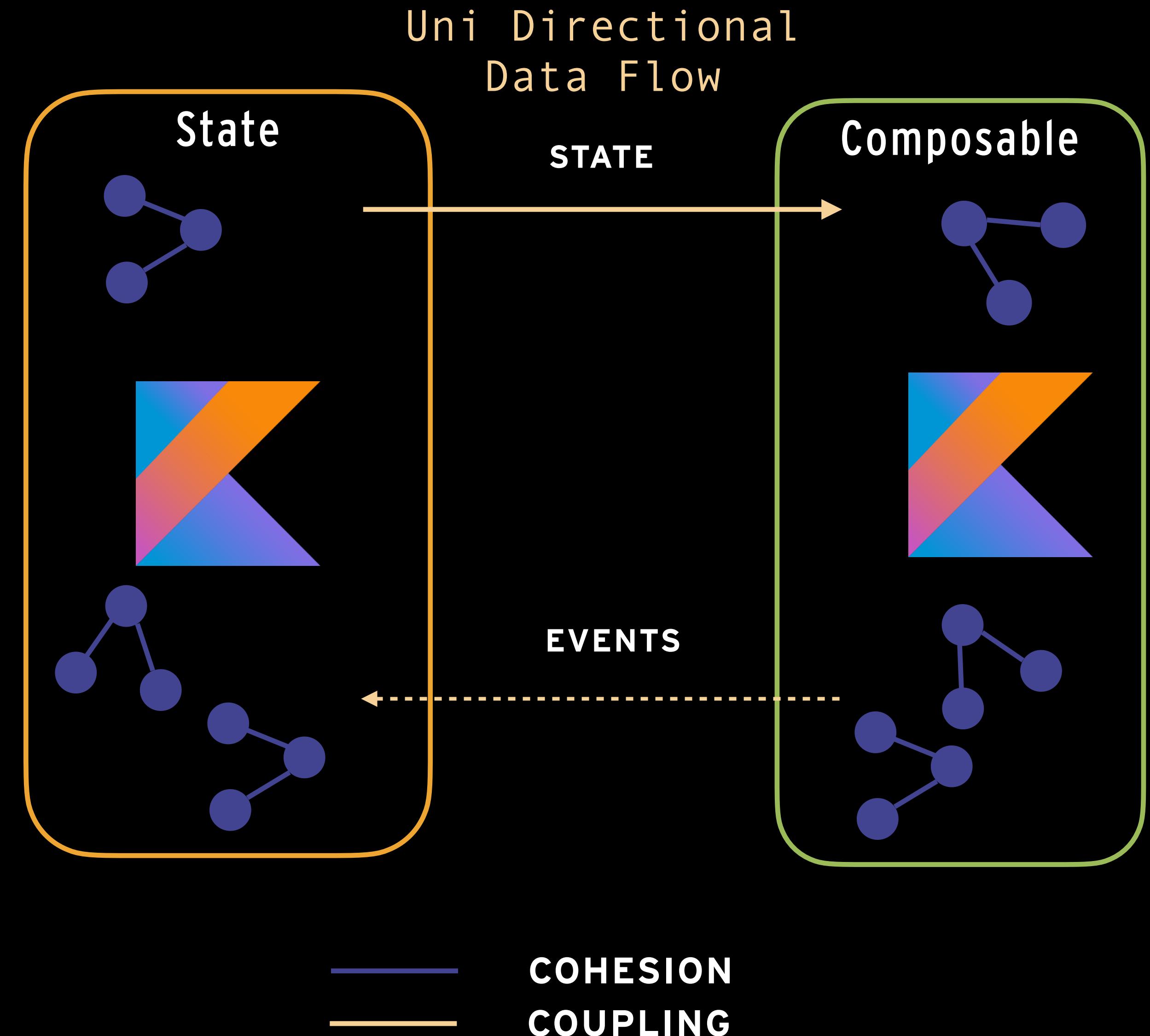
WHY - COUPLING / COHESION

- Coupling
- Cohesion
- Goal is for low *coupling* and high *cohesion*



WHY - LOW COUPLING, HIGH COHESION

- Increased Cohesion
- Logic in the Composable
- Theming in the Composable
- Leads to :
 - Code Clarity
 - Testability
 - Faster iterations
 - Minimise complexity and cost of change



SEPERATION WAS AN ILLUSION ...

WHEN

- Jetpack launched at Googles 2018 I/O Developer conference
- Jetpack Compose launched at Googles 2019 I/O Developer conference
- Alphas appeared in 2020 now on 1.0.0-alpha09
 - Version 1.0.0-alpha09 - December 16, 2020
 - Version 1.0.0-alpha08 - December 2, 2020
 - Version 1.0.0-alpha07 - November 11, 2020
 - Version 1.0.0-alpha01 - August 26, 2020
 - Version 0.1.0-dev15 - July 22, 2020

<https://developer.android.com/jetpack/androidx/releases/compose-animation#1.0.0-alpha09>

WHEN - 2018 - LELAND RICHARDSON

With high comply times in the existing native codebases you always need to write hundreds of lines of code before you can press “built”. React Native enables an entirely different style of development where you get really quick feedback and this is very powerful.

LELAND RICHARDSON, AIRBNB

[mindk - the-best-react-native-apps](#)

- React Native, Air BnB
- Creator of Enzyme
- Hugely popular and influential in the React Community
- Left Air BnB pretty much when they decided to ditch react native and started at Google
- Driving force behind JetPack compose
- Main link between Compose and React



REACT MINDSET

REACT MINDSET



STATE

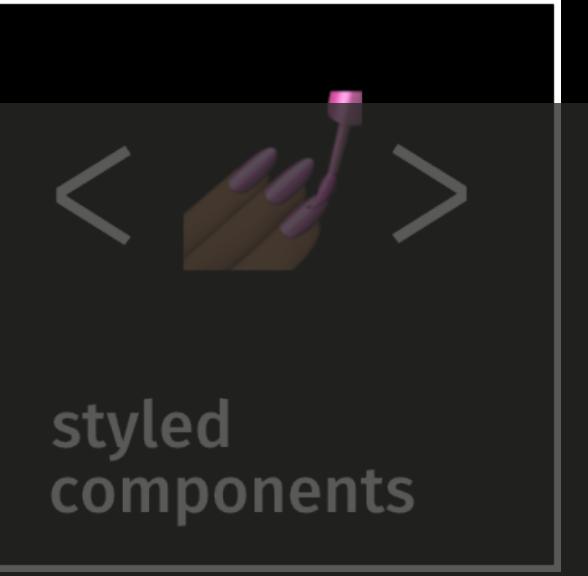


TEST

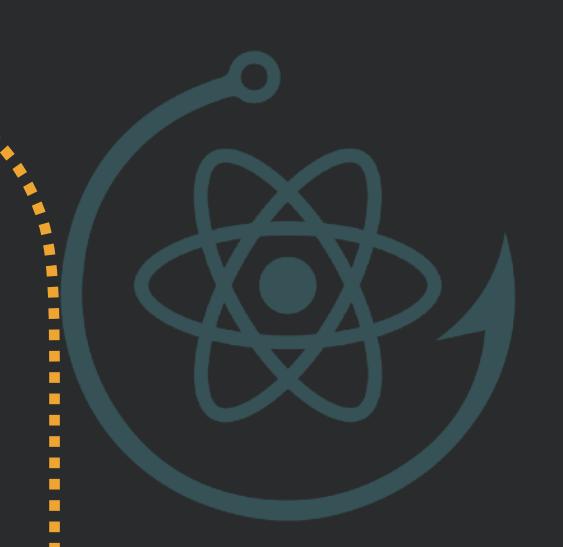
UI IS A VISUAL REPRESENTATION OF STATE



```
const Counter: React.FC<{id}> = ({ id }) => {
  const [count, setCount] = useState(0);
  useEffect(() => {
    if (count > 5) {
      setCount(0);
    }
  }, [count]);
  return (
    <Button
      onPress={() => setCount(count + 1)}
      testID={counter-$id}
      styles={{ backgroundColor: colors[count] }}
    >
      Counter : {count}
    </Button>
  );
};
```



STYLING



EFFECTS

THINKING IN COMPOSE

UI IS A VISUAL REPRESENTATION OF STATE

COMPOSE MINDSET

- Declarative
- Preview
 - Interact
- Composable Functions
- Data Down, Events Up
- Dynamic Content / Logic
- Recomposition

```
@Composable
fun CounterButton(id: String, modifier: Modifier = Modifier) {
    val (count, setCount) = remember { mutableStateOf(0) }
    Row(
        modifier = modifier.padding(4.dp)
    ) {
        OutlinedButton(
            onClick = { setCount(count + 1) },
            border = BorderStroke(1.dp, MaterialTheme.colors.primary),
            shape = RoundedCornerShape(50),
            modifier = modifier.semantics { testTag = "Counter-$id" }
        ) {
            Text(text = "Count : $count")
        }
    }
}
```

COMPOSE MINDSET

- Declarative
- Preview
 - Interact
- Composable Functions
- Data Down, Events Up
- Dynamic Content / Logic
- Recomposition

```
@Composable
fun CounterButton(id: String, modifier: Modifier = Modifier) {
    val (count, setCount) = remember { mutableStateOf(0) }
    Row(
        modifier = modifier.padding(4.dp)
    ) {
        OutlinedButton(
            onClick = { setCount(count + 1) },
            border = BorderStroke(1.dp, MaterialTheme.colors.primary),
            shape = RoundedCornerShape(50),
            modifier = modifier.semantics { testTag = "Counter-$id" }
        ) {
            Text(text = "Count : $count")
        }
    }
}
```

PREVIEW



COMPOSE MINDSET

- Declarative
- Composable Functions
- Data Down, Events Up
- Dynamic Content / Logic
- Recomposition

```
@Composable
fun CounterButtons(modifier: Modifier = Modifier) {
    Row(modifier = modifier
        .background(
            color = MaterialTheme.colors.surface)) {
        CounterButton(id = "1")
        CounterButton(id = "2")
        CounterButton(id = "3")
    }
}

@Preview
@Composable
fun CounterButtonsPreview() {
    CounterButtons()
}
```

CounterButtonsPreview

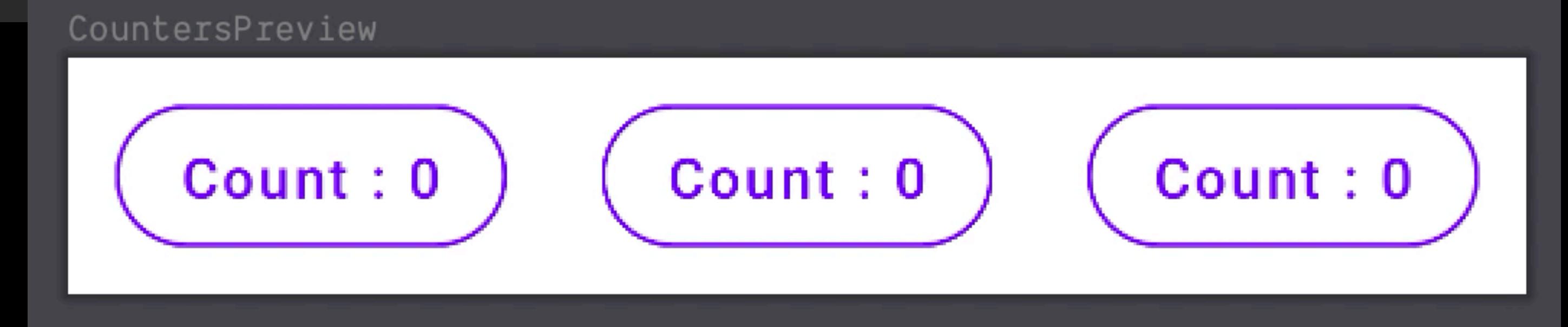


COMPOSE MINDSET

- Declarative
- Composable Functions
- Data Down, Events Up
- Dynamic Content / Logic
- Recomposition

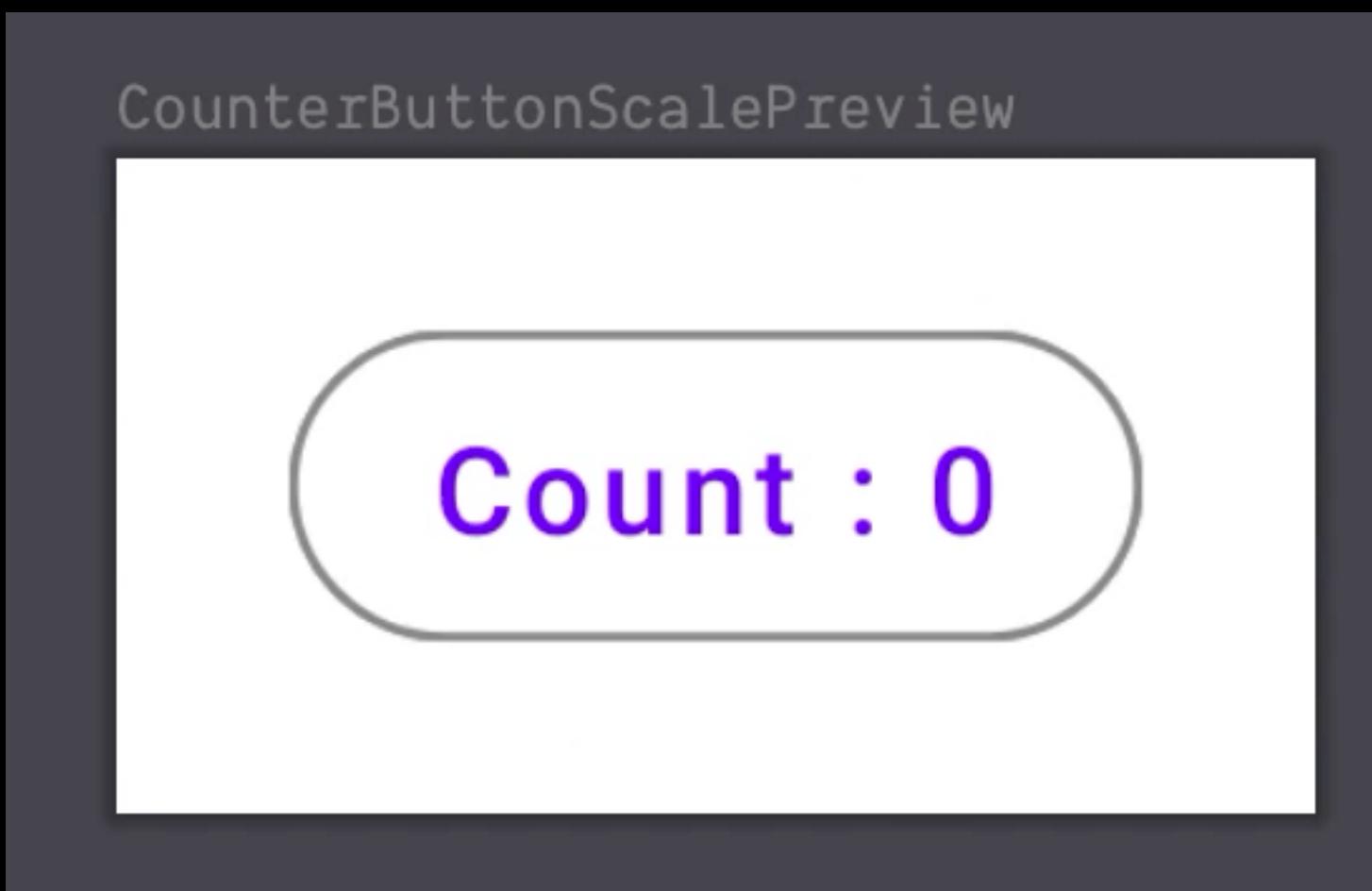
```
@Composable
fun Counters(modifier: Modifier = Modifier) {
    val (count1, setCount1) = remember { mutableStateOf(0) }
    Row() {
        Counter(id = "1", count = count1, setCount = setCount1)
        Counter(id = "2", count = count1, setCount = setCount1)
        Counter(id = "3", count = count1, setCount = setCount1)
    }
}

@Composable
fun Counter(id: String, count: Int, setCount: (count: Int) -> Unit, modifier: Modifier = Modifier) {
    Row(
        modifier =
            modifier.padding(4.dp).background(color = MaterialTheme.colors.surface).padding(8.dp)
    ) {
        OutlinedButton(
            onClick = { setCount(count + 1) },
            border = BorderStroke(1.dp, MaterialTheme.colors.primary),
            shape = RoundedCornerShape(50), //50% percent,
            modifier = modifier.semantics { testTag = "Counter-$id" }
        ) {
            Text(text = "Count : $count")
        }
    }
}
```



COMPOSE MINDSET

- Declarative
- Composable Functions
- Data Down, Events Up
- Dynamic Content / Logic
- Recomposition



```
@Composable
fun CounterButtonScale(id: String, modifier: Modifier = Modifier) {
    val (count, setCount) = remember { mutableStateOf(0) }
    val (scale, setScale) = remember { mutableStateOf(1.0f) }
    val animatedScale = animate(scale, TweenSpec(300))
    onCommit(count, {
        if (count > 2) {
            setScale(1.0f)
            setCount(0)
        } else if (count > 0) {
            setScale(scale - 0.1f)
        }
    })
    Row(
        modifier = modifier.padding(4.dp)
    ) {
        OutlinedButton(
            onClick = { setCount(count + 1) },
            border = BorderStroke(1.dp,
                if (count < 5)
                    colors[count]
                else
                    MaterialTheme.colors.primary),
            shape = RoundedCornerShape(50),
            modifier = modifier.scale(animatedScale)
        ) {
            Text(text = "Count : $count")
        }
    }
}
```

The code is annotated with two callouts:

- A callout labeled "Effect" surrounds the `onCommit` block, which contains logic to handle button presses and update state.
- A callout labeled "Inline logic" surrounds the `BorderStroke` configuration within the `OutlinedButton` modifier, highlighting how logic is embedded directly within the modifier parameters.

COMPOSE MINDSET

- Declarative
- Composable Functions
- Data Down, Events Up
- Dynamic Content / Logic
- Recomposition
- Should we call the functions ourselves?
- Is everything redrawn?
- No!
- Runs in Parallel!

```
@Composable
fun Counters(modifier: Modifier = Modifier) {
    val (count1, setCount1) = remember { mutableStateOf(0) }
    Row() {
        3 Counter(id = "1", count = count1, setCount = setCount1)
        1 Counter(id = "2", count = count1, setCount = setCount1)
        2 Counter(id = "3", count = count1, setCount = setCount1)
    }
}
```

COMPOSE MINDSET

- Declarative
- Composable Functions
- Data Down, Events Up
- Dynamic Content / Logic
- Recomposition
- Test

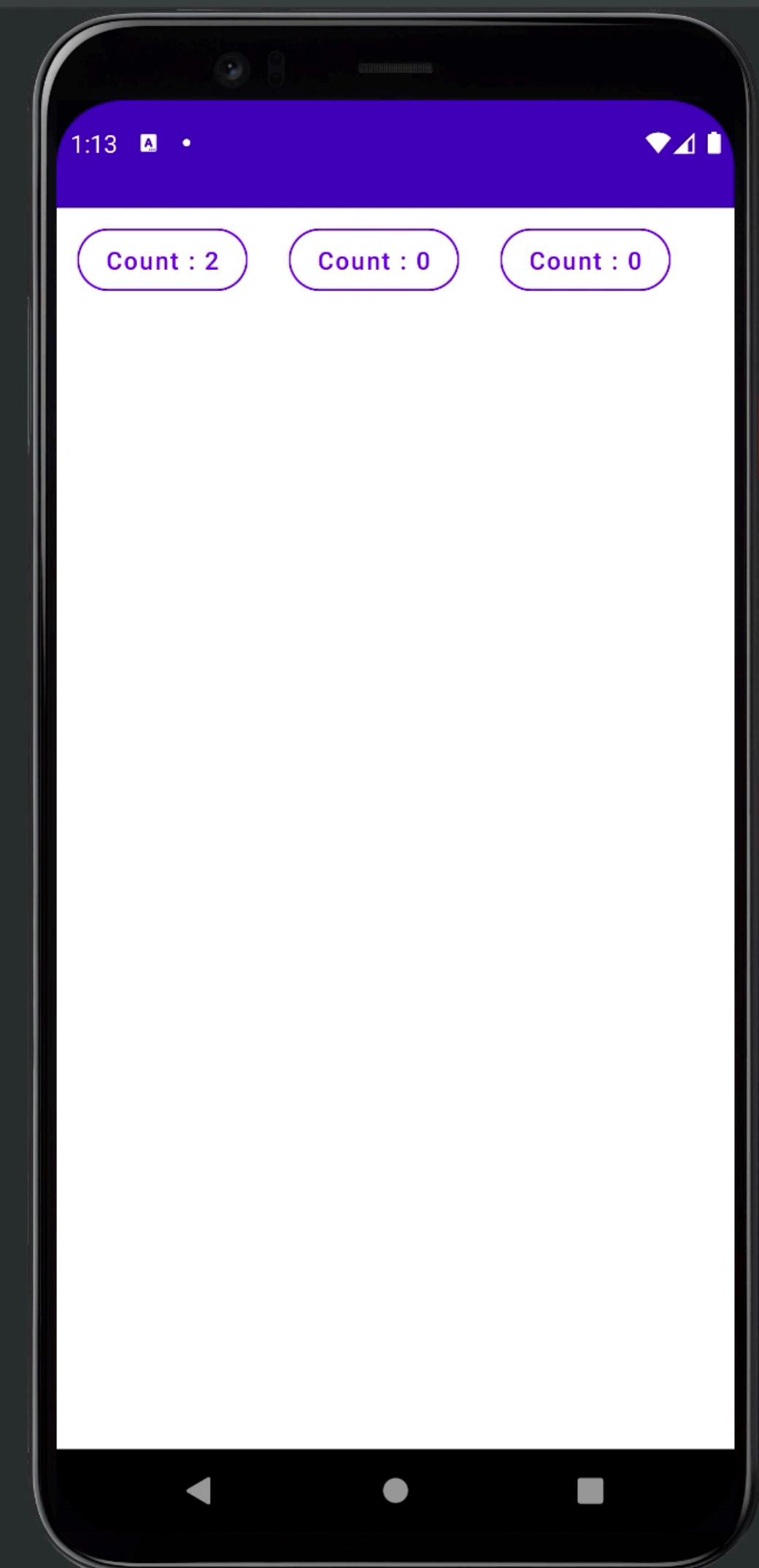
```
@Composable
fun CounterButton(id: String, modifier: Modifier = Modifier) {
    Row(
        modifier = modifier.padding(4.dp)
    ) {
        OutlinedButton(
            ...
            modifier = modifier.semantics { testTag = "Counter-$id" }
        )
    }
}
```

```
@RunWith(JUnit4::class)
class CounterKtTest {

    @get:Rule
    val composeTestRule = createAndroidComposeRule<CounterActivity>()
```

```
@Test
fun testCounter() {
    composeTestRule.setContent {
        Counters()
    }
    composeTestRule.onNodeWithTag("Counter-1").assertTextEquals("Count : 0")
    composeTestRule.onNodeWithTag("Counter-1").performClick();
    composeTestRule.onNodeWithTag("Counter-1").assertTextEquals("Count : 1")
    composeTestRule.onNodeWithTag("Counter-1").performClick();
    composeTestRule.onNodeWithTag("Counter-1").assertTextEquals("Count : 2")
}
```

```
1 package com.sw.mobile.flickrbrowser
2
3 import ...
4
5 @RunWith(JUnit4::class)
6 class CounterKtTest {
7
8     @get:Rule
9     val composeTestRule = createAndroidComposeRule<CounterActivity>()
10
11    @Test
12    fun testCounter() {
13        composeTestRule.setContent {
14            Counters()
15        }
16
17        composeTestRule.onNodeWithTag(testTag: "Counter-1").assertTextEquals("Count : 0")
18        composeTestRule.onNodeWithTag(testTag: "Counter-1").performClick()
19        composeTestRule.onNodeWithTag(testTag: "Counter-1").assertTextEquals("Count : 1")
20        composeTestRule.onNodeWithTag(testTag: "Counter-1").performClick()
21        composeTestRule.onNodeWithTag(testTag: "Counter-1").assertTextEquals("Count : 2")
22    }
23
24 }
```



The screenshot shows the Android Studio Test Results window. At the top, it says "Run: CounterKtTest" and "Counters". Below that is a "Status" bar with a progress bar, "0 passed", and "1 tests". Underneath is a "Filter tests:" section with a green checkmark icon, a circular icon with a slash, and other filtering options. The main table has columns for "Tests", "Duration", and "Pixel". It shows one test result: "Test Results" under "CounterKtTest" with a duration of 6 s and 0/1 pixels. A green bar at the bottom right indicates "Launch succeeded".

Test Results

Install successfully finished in 873 ms.

Running tests

```
$ adb shell am instrument -w -m -e debug false -e class 'com.sw.mobile.flickrbrowser.CounterKtTest' com.sw.m
```

Connected to process 12458 device 'emulator-5554'. TV

THINKING IN REACT

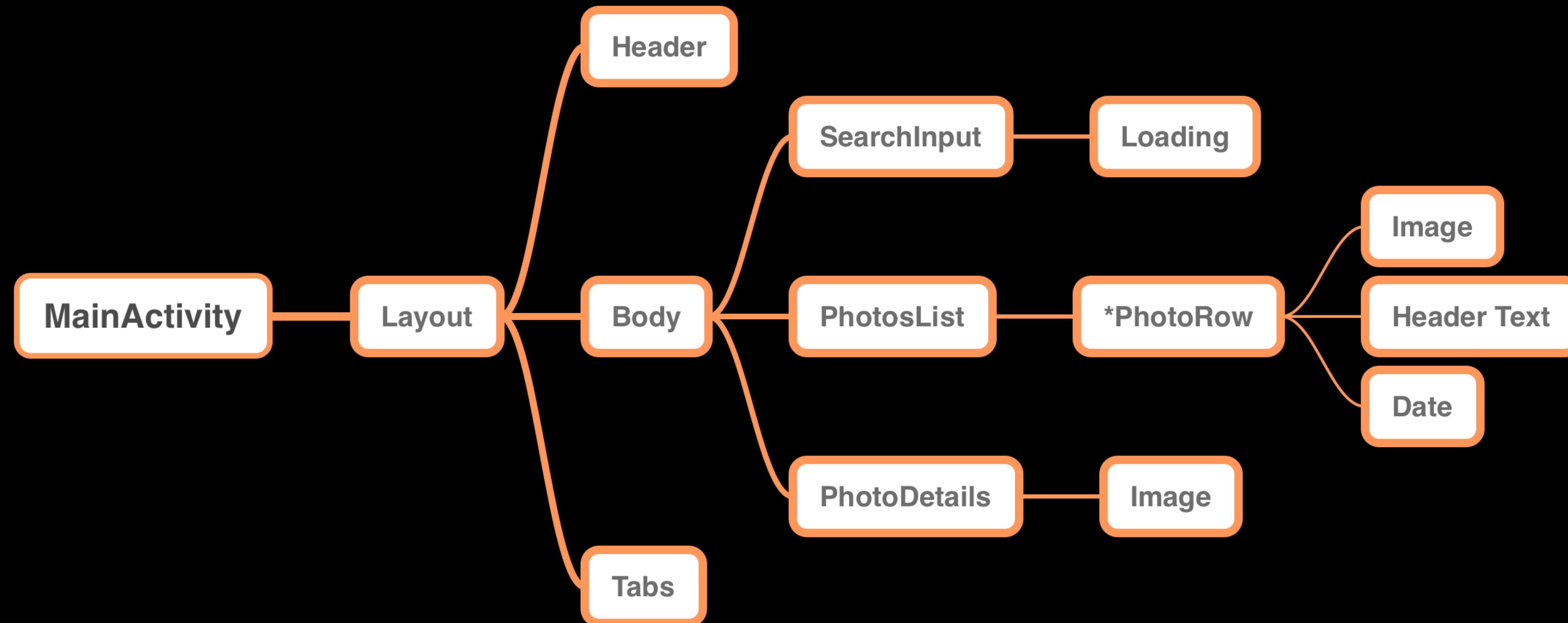
THINKING IN REACT

- Start with a mock, UI and Data
- Break UI in a component Hierarchy
- Build a static version in React
- Identify The minimal Representation Of UI State
- Identify where the state should live (Data down)
- An Inverse Data Flow, top level component passes callbacks to child components to mutate the state.



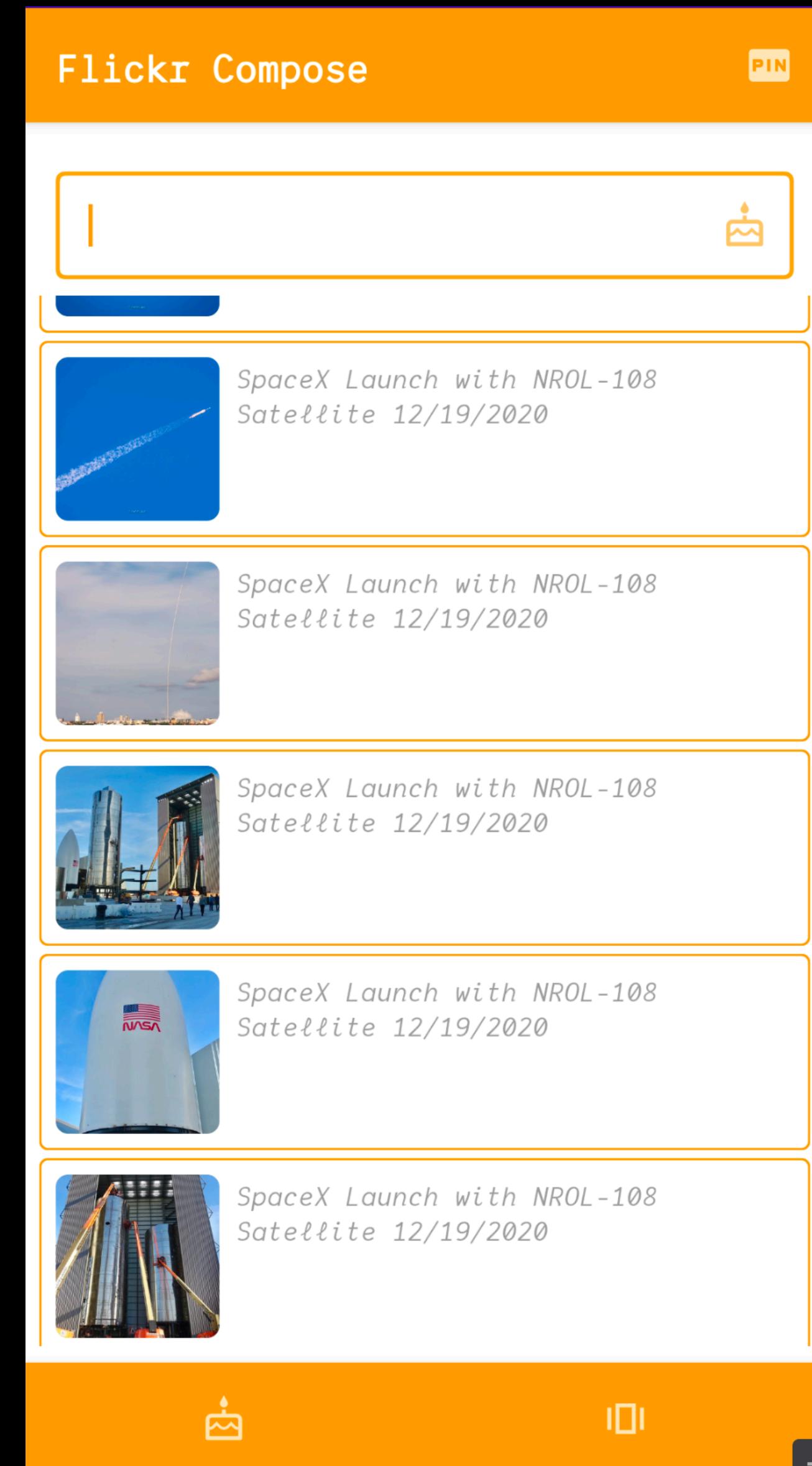
THINKING IN REACT

- Start with a mock, UI and Data
- Break UI in a component Hierarchy
- Build a static version in React

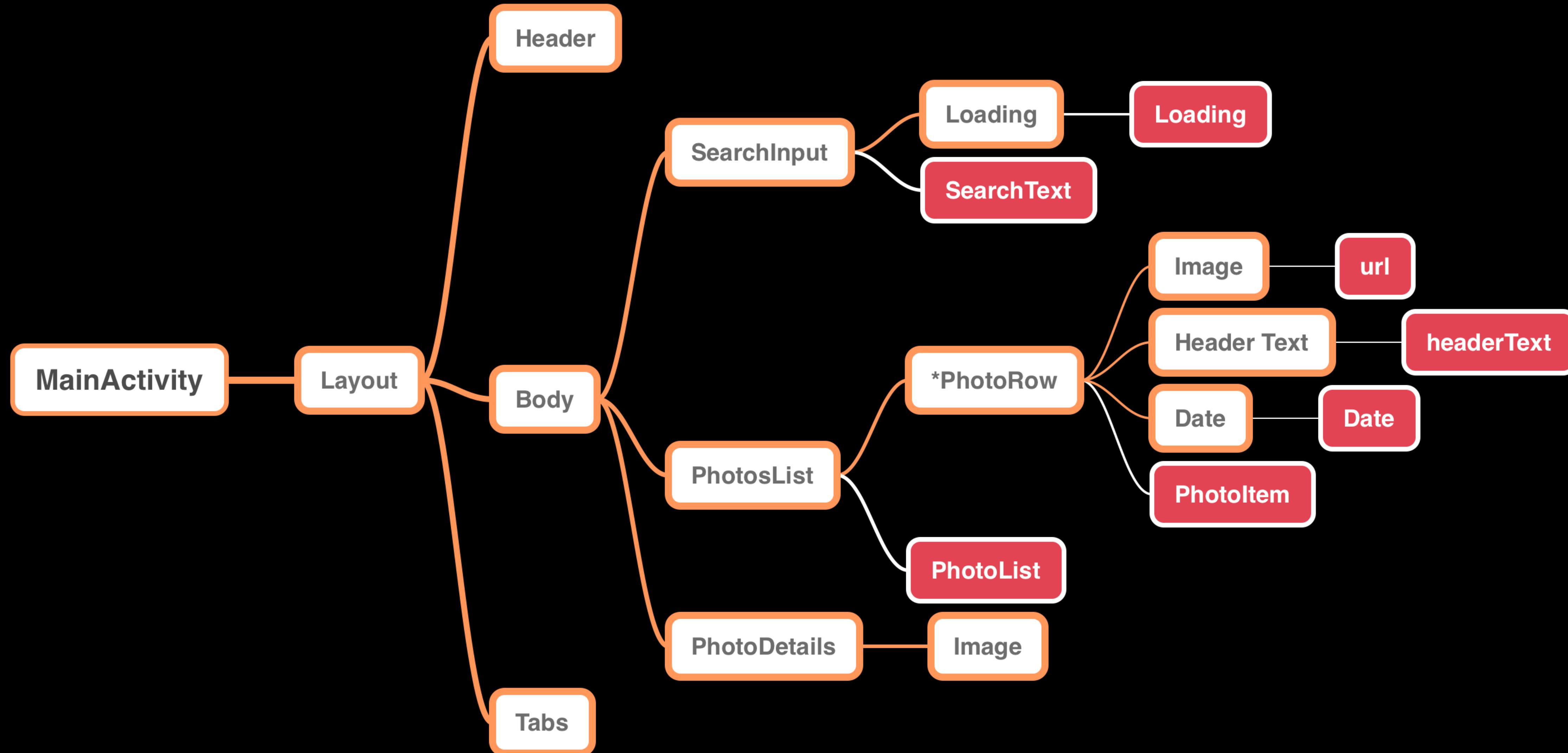


THINKING IN REACT

- Start with a mock, UI and Data
- Break UI in a component Hierarchy
- Build a static version in Compose
- Identify The minimal Representation Of UI State
- Identify where the state should live (Data down)
- An Inverse Data Flow, top level component passes callbacks to child components to mutate the state.

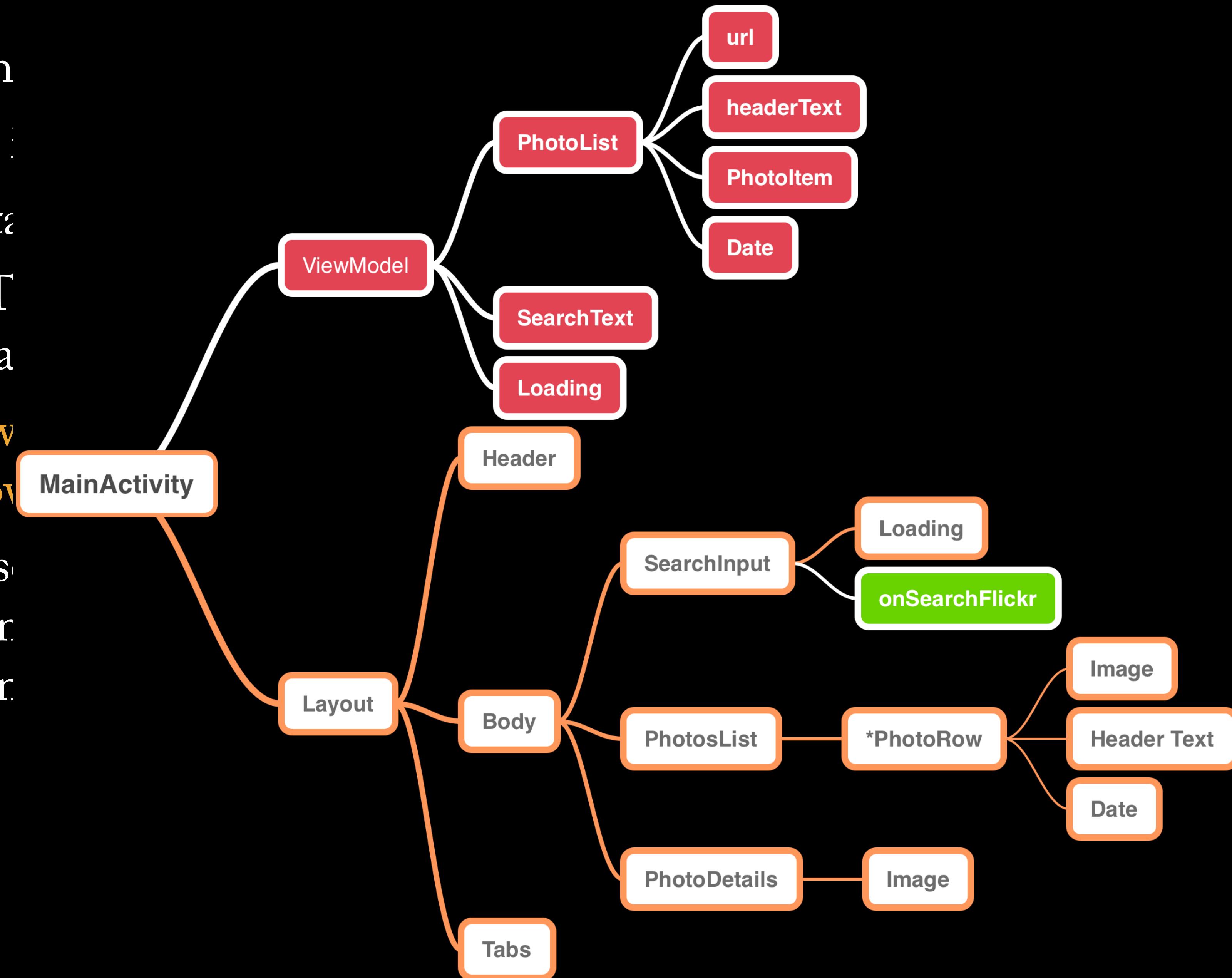


THINKING IN REACT - IDENTIFY THE MINIMAL REPRESENTATION OF UI STATE



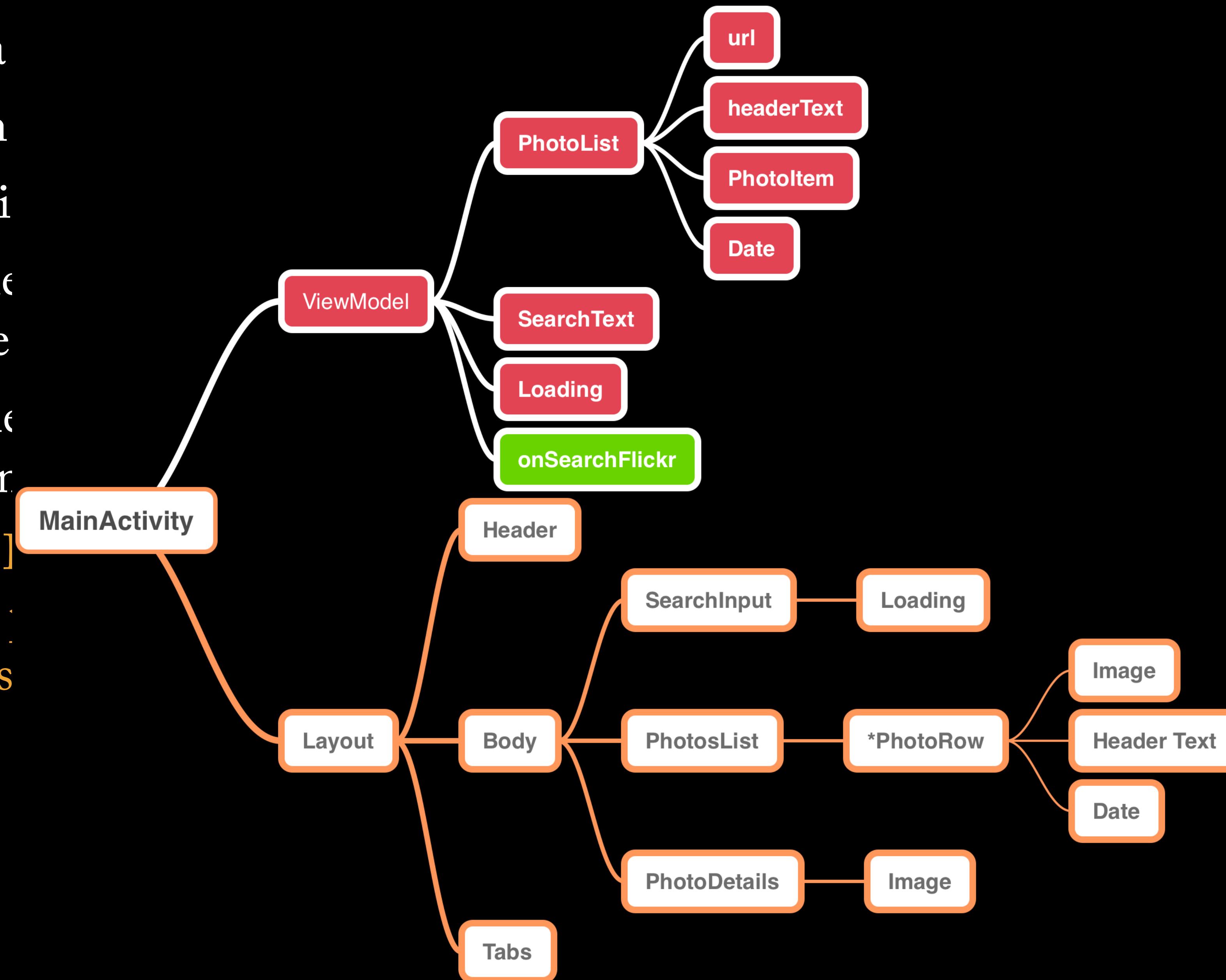
THINKING IN REACT - IDENTIFY WHERE THE STATE SHOULD LIVE (DATA DOWN)

- Start with
- Break UI
- Build a state tree
- Identify The Tree Of UI State
- Identify where state lives (Data down)
- An Inverse of the component tree

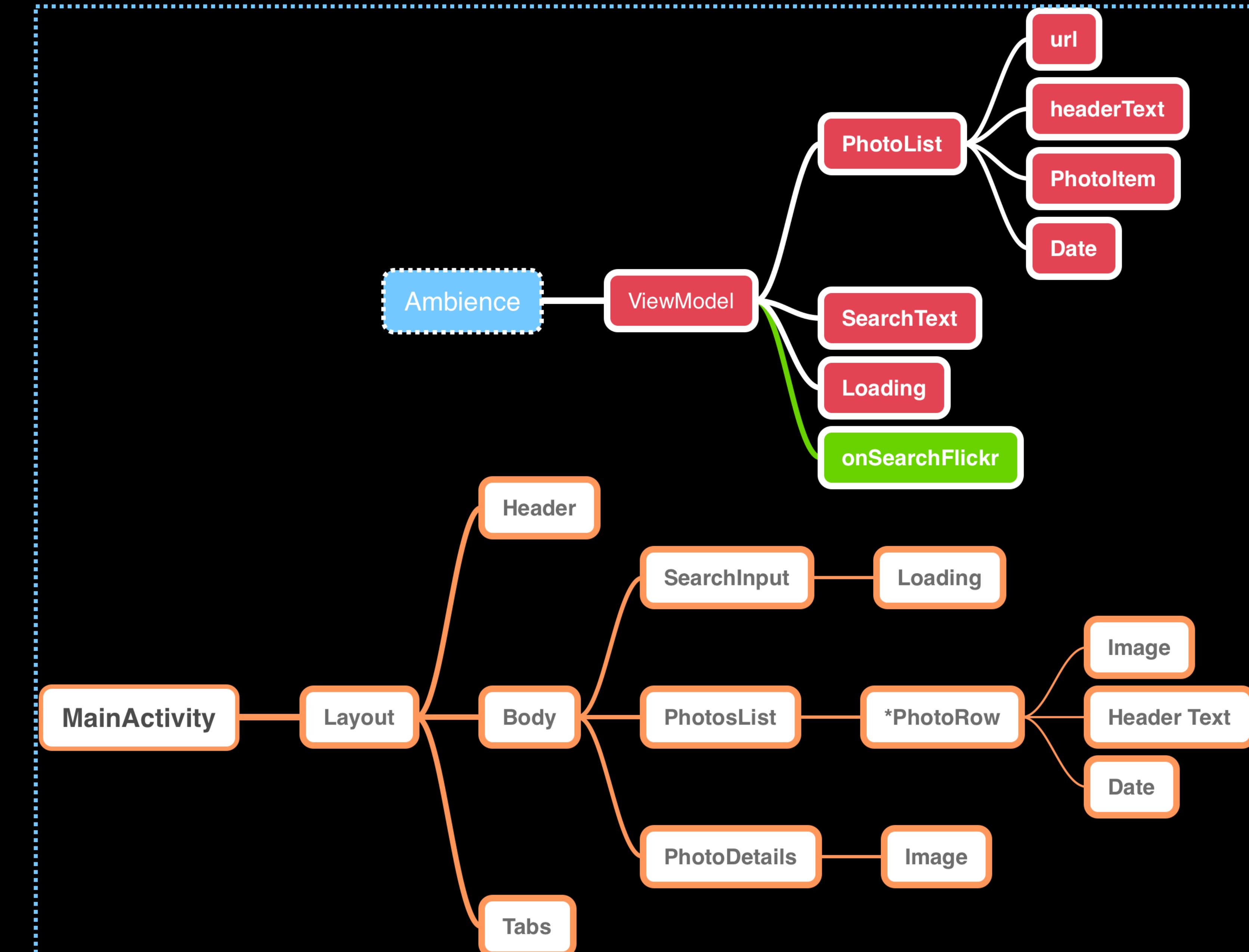


THINKING IN REACT - CALLBACKS

- Start with a
- Break UI in
- Build a static
- Identify The
Of UI State
- Identify what
(Data down)
- An Inverse]
component :
components

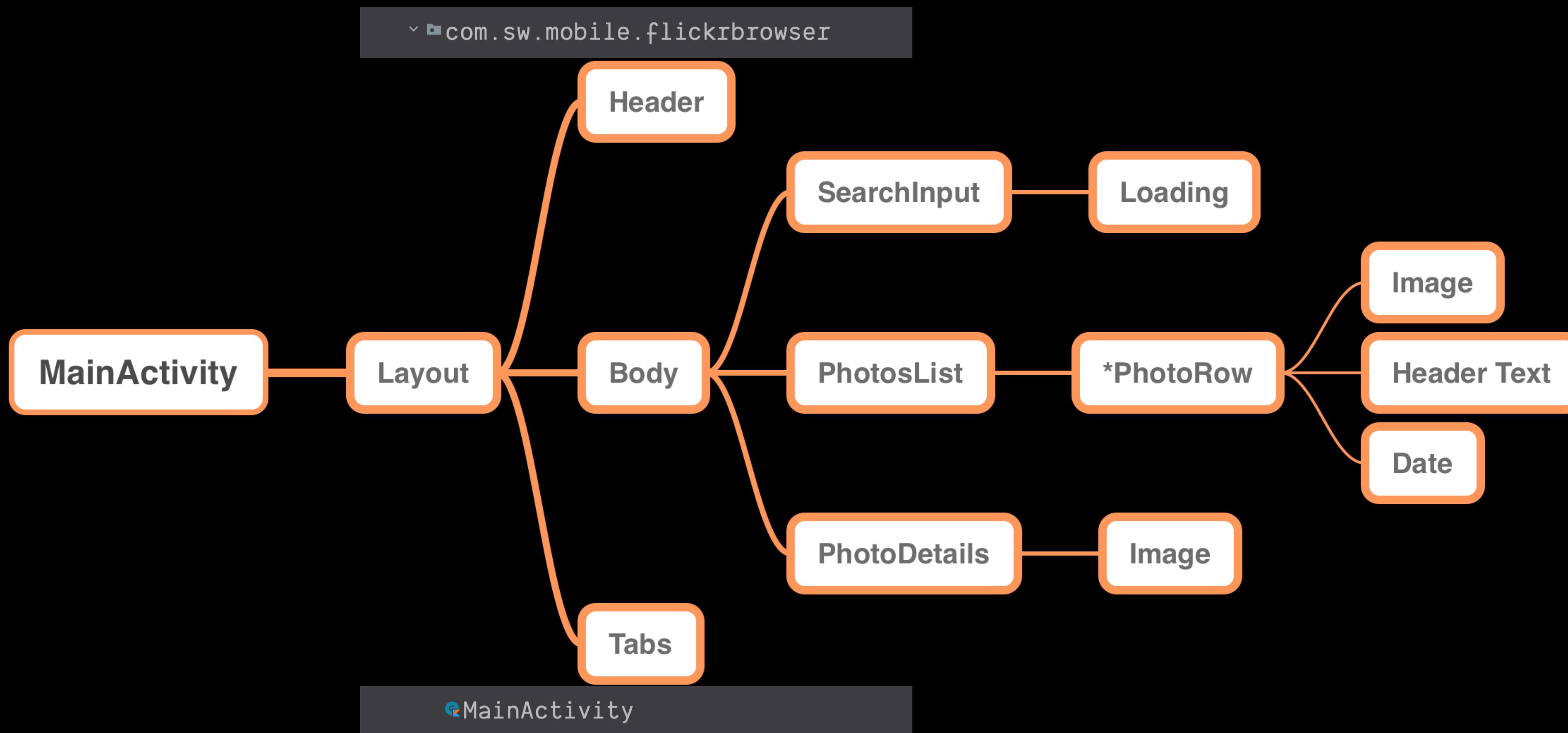


THINKING IN REACT - IDENTIFY WHERE THE STATE SHOULD LIVE (DATA AROUND)



APPLYING REACT TO COMPOSE

APPLYING REACT IN COMPOSE



APPLYING REACT IN COMPOSE - THEME

```
class MainActivity : AppCompatActivity() {  
    val flickrViewModel by viewModels<FlickrViewModel>()  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContent {  
            FlickrBrowserTheme {  
                Providers(AmbientFlickrSearch provides flickrViewModel) {  
                    LayoutFlickr(  
                    )  
                }  
                onActive(callback = { flickrViewModel.onSearchFlickr("rocket") })  
            }  
        }  
    }  
}
```

APPLYING REACT IN COMPOSE - THEME

ui/

 └ ui

 └ Color.kt

 └ Shape.kt

 └ Theme.kt

 └ Type.kt

Color.kt

```
val Orange700 = Color(0xFFFFA000)
val Orange800 = Color(0xFFFF8F00)
val Orange900 = Color(0xFFFF6F00)
```

```
val Orange200 = Color(0xFFFFE082)
val Orange300 = Color(0xFFFFE082)
```

```
val LightColors = lightColors(
    primary = Orange700,
    primaryVariant = Orange900,
    onPrimary = Color.White,
    secondary = Orange700,
    secondaryVariant = Orange200,
    onSecondary = Color.White,
    error = Orange800
)
```

```
val DarkColors = darkColors(
    primary = Orange300,
    primaryVariant = Orange700,
    onPrimary = Color.Black,
    secondary = Orange300,
    onSecondary = Color.White,
    error = Orange200
)
```

Shape.kt

```
val shapes = Shapes(
    small = RoundedCornerShape(4.dp),
    medium = RoundedCornerShape(4.dp),
    large = RoundedCornerShape(0.dp)
)
```

Type.kt

```
private val Dank = fontFamily(
    font(R.font.dankmono_italic, style = FontStyle.Italic),
    font(R.font.dankmono_regular),
)

val FlickrTypography = Typography(
    h4 = TextStyle(
        fontFamily = Dank,
        fontWeight = FontWeight.W600,
        fontSize = 30.sp
    ),
    h5 = TextStyle(
        fontFamily = Dank,
        fontWeight = FontWeight.W600,
        fontSize = 24.sp
    ),
    h6 = TextStyle(
        fontFamily = Dank,
        fontWeight = FontWeight.W600,
        fontSize = 20.sp
    ),
    subtitle1 = TextStyle(
        fontFamily = Dank,
        fontWeight = FontWeight.W600,
```

Theme.kt

```
@Composable
fun FlickrBrowserTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    content: @Composable () -> Unit) {
    MaterialTheme(
        colors = if (darkTheme) DarkColors else LightColors,
        typography = FlickrTypography,
        shapes = FlickrShapes,
        content = content
    )
}
```

APPLYING REACT IN COMPOSE - VIEW MODEL - STATE/HANDLERS

```
class FlickrViewModel : ViewModel() {  
    var photos by mutableStateOf(PhotosResponse())  
    var searchText by mutableStateOf("")  
    private set  
    var loading by mutableStateOf(false)  
  
    fun onSearchFlickr(value: String) {  
        loading = true  
        viewModelScope.launch {  
            photos = getApi().search(tags = "spacex", text = value)  
        }  
        loading = false  
    }  
}  
  
val AmbientFlickrSearch = ambientOf<FlickrViewModel>()
```

APPLYING REACT IN COMPOSE - MODEL - IN ACTIVITY

```
class MainActivity : AppCompatActivity() {  
    val flickrViewModel by viewModels<FlickrViewModel>()  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContent {  
            FlickrbrowserTheme {  
                Providers(AmbientFlickrSearch provides flickrViewModel) {  
                    LayoutFlickr(  
                    )  
                }  
                onActive(callback = { flickrViewModel.onSearchFlickr("rocket") })  
            }  
        }  
    }  
}
```

APPLYING REACT IN COMPOSE - ON ACTIVE

```
class MainActivity : AppCompatActivity() {  
    val flickrViewModel by viewModels<FlickrViewModel>()  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContent {  
            FlickrbrowserTheme {  
                Providers(AmbientFlickrSearch provides flickrViewModel) {  
                    LayoutFlickr(  
                        )  
                }  
                onActive(callback = { flickrViewModel.onSearchFlickr("rocket") })  
            }  
        }  
    }  
}
```

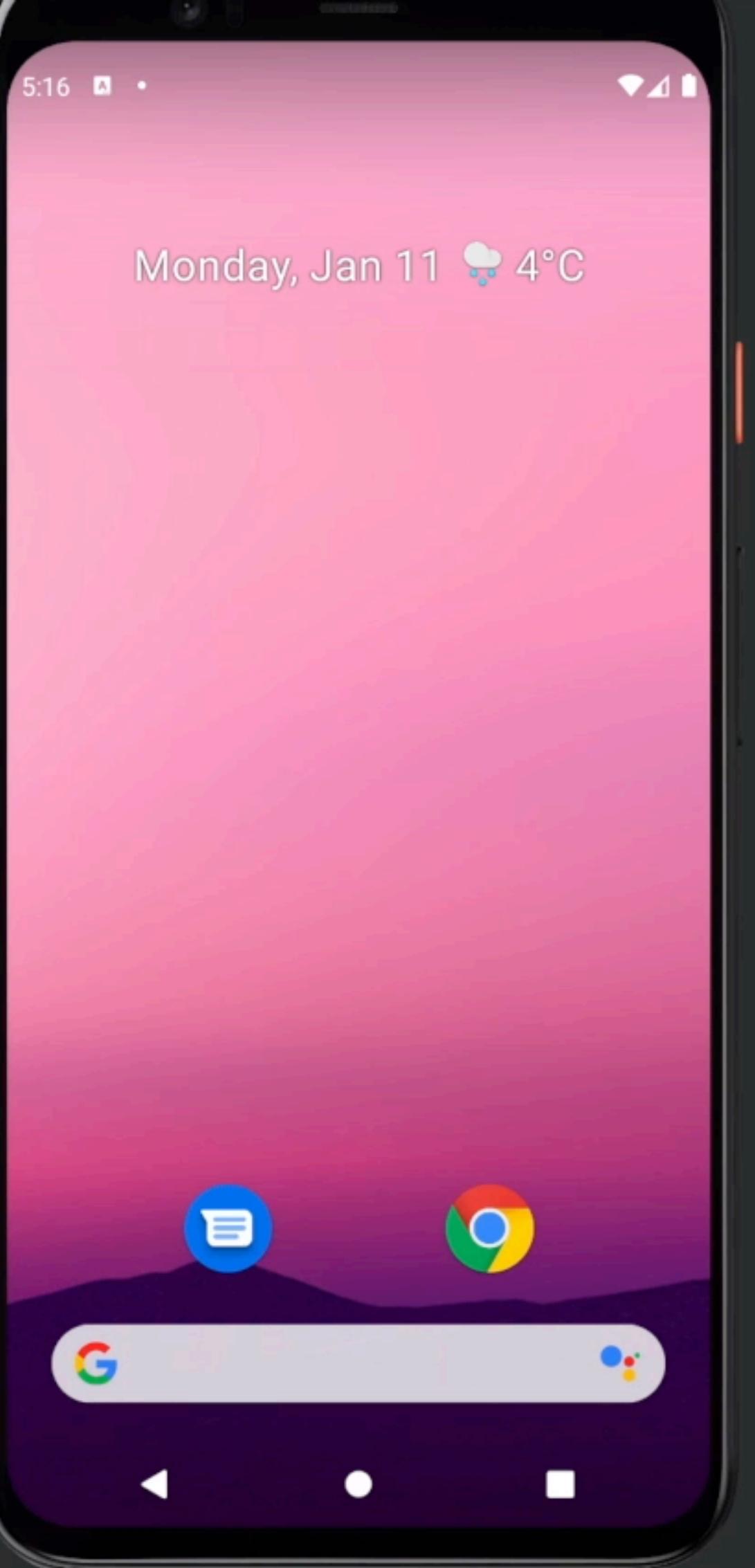
SEARCH INPUT - DATA FLOW

```
@Composable  
fun SearchInputTextContainer() {  
    val flickrViewModel = AmbientFlickrSearch.current  
    SearchInputText(  
        searchText = flickrViewModel.searchText,  
        loading = flickrViewModel.loading,  
        onSearchFlickr = flickrViewModel::onSearchFlickr  
    )  
}
```

```
class FlickrViewModelMock(_searchText: String = "") :  
    FlickrViewModel(_searchText, _photos = ..._photosRes, _photoRes = ..._photoRes, _imageMocks) {  
    override fun onSearchFlickr(value: String) {  
        if (value.length < 3) return  
        loading = true  
        searchJob = viewModelScope.launch {  
            delay(500)  
            photos = getApi().search(tags = "spacex", text = value)  
            loading = false  
        }  
    }  
}
```

```
@Composable  
fun SearchInputText(  
    modifier: Modifier = Modifier,  
    searchText: String = "",  
    loading: Boolean,  
    onSearchFlickr: (text: String) -> Unit,  
    onImeAction: () -> Unit = {},  
) {  
    var textVal by savedInstanceState[searchText]  
    onCommit(textVal, {  
        onSearchFlickr(textVal)  
    })  
    Row(modifier = modifier.fillMaxWidth().background(MaterialTheme.colors.surface)) {  
        OutlinedTextField(  
            value = textVal,  
            onValueChange = {  
                textVal = it  
            },  
            activeColor = MaterialTheme.colors.primary,  
            inactiveColor = MaterialTheme.colors.secondary,  
            keyboardOptions = KeyboardOptions(imeAction = ImeAction.Done),  
            onImeActionPerformed = { action, softKeyboardController ->  
                if (action == ImeAction.Done) {  
                    onImeAction()  
                    softKeyboardController?.hideSoftwareKeyboard()  
                }  
            },  
            trailingIcon = {  
                if (loading)  
                    Icon(  
                        Icons.Filled.Cake,  
                        modifier = modifier.testTag ("icon-loading")  
                    )  
                else  
                    Icon(  
                        Icons.Outlined.Cake,  
                        modifier = modifier.testTag ("icon-not-loading")  
                    )  
            },  
            modifier = modifier.fillMaxWidth().padding(8.dp).testTag("search-input")  
        )  
    }  
}
```

```
class SearchInputTextContainerKtTest {  
  
    @get:Rule  
    val composeTestRule = createComposeRule()  
  
    @Test  
    fun searchInputTextNotLoading() {  
        val flickrViewModel = FlickrViewModelMock(_searchText = "")  
        composeTestRule.setContent {  
            FlickrBrowserTheme {  
                Providers( ...values: AmbientFlickrSearch provides flickrViewModel) {  
                    SearchInputTextContainer()  
                }  
            }  
        }  
  
        composeTestRule.onNodeWithTag(testTag: "search-input").assertTextEquals("")  
        composeTestRule.onNodeWithTag(testTag: "icon-not-loading", useUnmergedTree = true).assertExists()  
        composeTestRule.onNodeWithTag(testTag: "search-input").performTextInput(text: "Space")  
        composeTestRule.onNodeWithTag(testTag: "search-input").assertTextEquals("Space")  
        composeTestRule.onNodeWithTag(testTag: "icon-loading", useUnmergedTree = true).assertExists()  
        // should revert to not loading  
  
        composeTestRule.onNodeWithTag(testTag: "icon-not-loading", useUnmergedTree = true).assertDoesNotExist()  
        Thread.sleep(millis: 2000)  
    }  
}
```



Tests	Duration	Pixel_5
Test Results		
✓ SearchInputTextContainerKtTest	7 s	1/1
✓ searchInputTextNotLoading	7 s	1/1

Tests Passed
1 passed

✓ Test Results

Install successfully finished in 508 ms.

Running tests

```
$ adb shell am instrument -w -m -e debug false -e class 'com.sw.mobile.flickrbrowser.composables.SearchInputTextContainerKtTest' -d Pixel_5
```

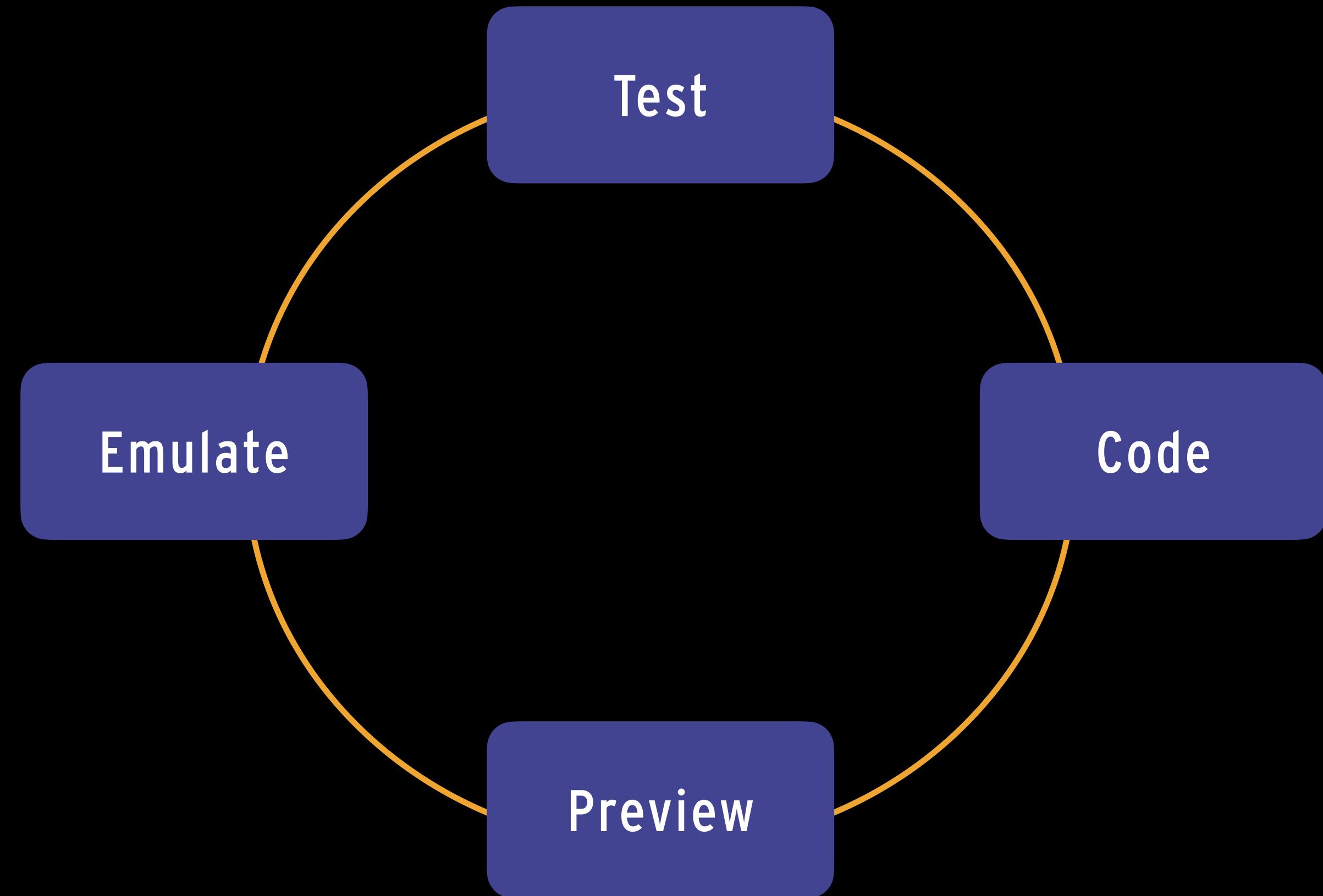
Connected to process 24021 on device 'Pixel_5 [emulator-5554]'.

SEARCH BOX - PREVIEW

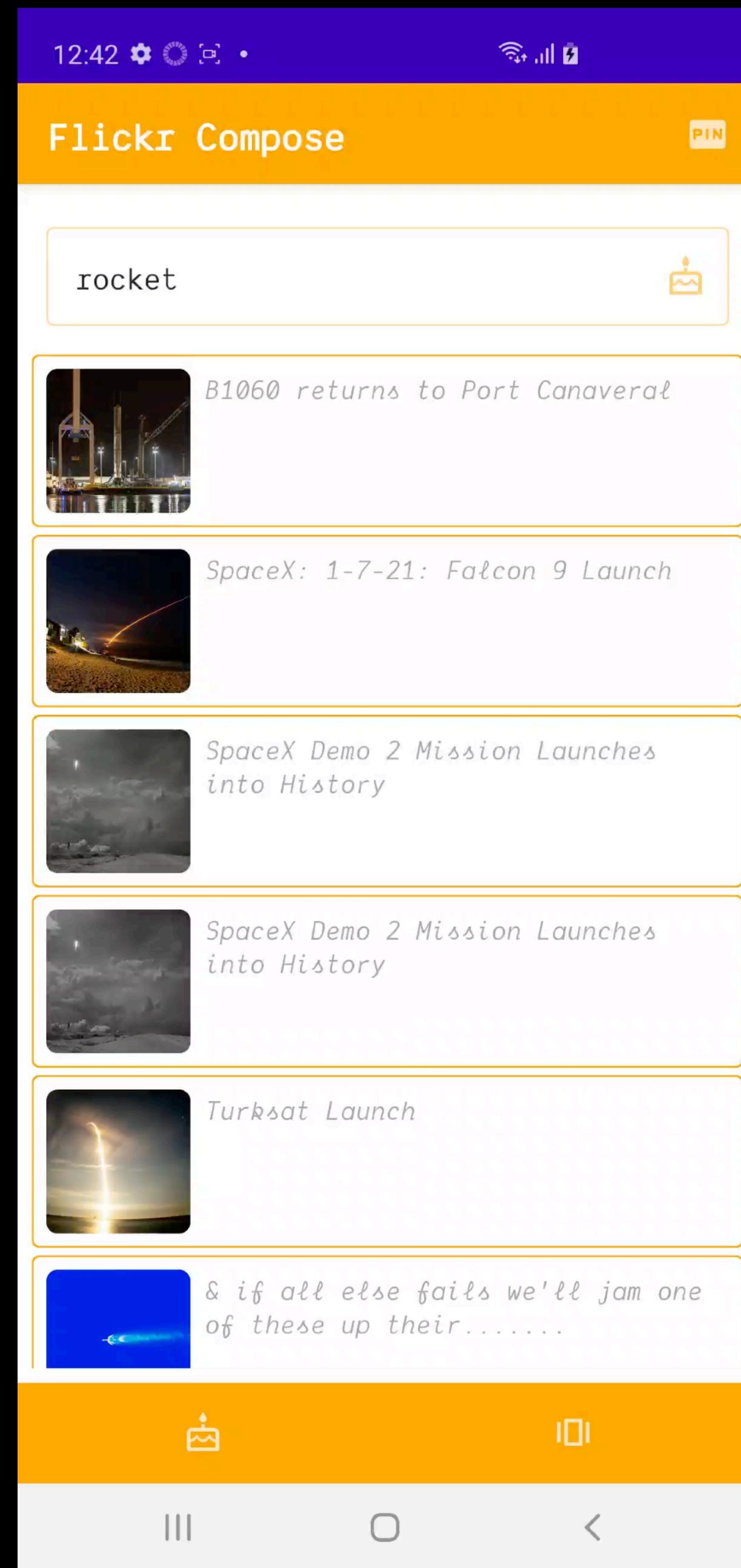
The screenshot shows the Android Studio interface with the following components:

- Left Panel (Code View):** Displays Kotlin code for a `SearchInputText` component. It includes four preview functions: `SearchInputTextPreview`, `SearchInputTextPreviewLoading`, `SearchInputTextPreviewLoadingTheme`, and `SearchInputTextContainerPreviewLoadingTheme`. Each preview function sets up a `SearchInputText` with specific parameters like `searchText` and `loading` status.
- Right Panel (Preview View):** Contains four separate preview windows corresponding to the code above:
 - SearchInputTextPreview:** Shows the text "Rocket has Loaded" in a white box with a light blue border and a small blue candle icon.
 - SearchInputTextPreviewLoading:** Shows the text "Rocket is Loading" in a white box with a light blue border and a small teal candle icon.
 - SearchInputTextPreviewLoadingTheme:** Shows the text "With Theme And Loading" in a white box with an orange border and an orange candle icon.
 - SearchInputTextContainerPreviewLoadingTheme:** Shows the text "With Ambient, it's a Context!!" in a white box with an orange border and an orange candle icon.
- Bottom Navigation Bar:** Includes standard icons for TODO, Problems, Git, Terminal, App Inspection, Run, Build, Logcat, Event Log, and Layout Inspector.
- Right Sidebar:** Features a vertical toolbar with icons for Hand, Plus, Minus, 1:1, and a square icon.

DEVELOPMENT CYCLE



FLICKR BROWSER



CONCLUSIONS

CONCLUSIONS

- Nailed it!
 - React Developer Adoption!
- Kotlin!
- DX
- Animation api
- Theming
- Compilation is slower in comparison to React Natives JS metro bundler
- Complete list of hook to Compose comparison on
 - <https://github.com/callistaenterprise/cadec-2021-compose>

END