Mobile Applications
CSCI 448
Lecture 34

Jetpack Compose

Side Effects

Learning Outcomes For Today

- Explain when Compose Side Effects should be used

On Tap For Today

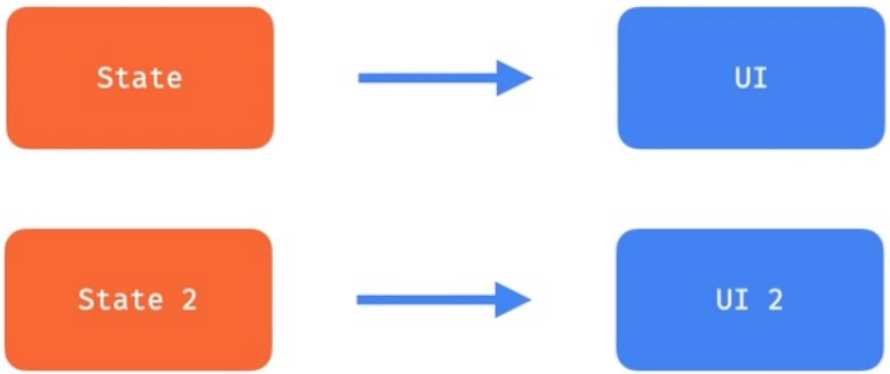
- Jetpack Compose & Side Effects

On Tap For Today

- Jetpack Compose & Side Effects

Compose

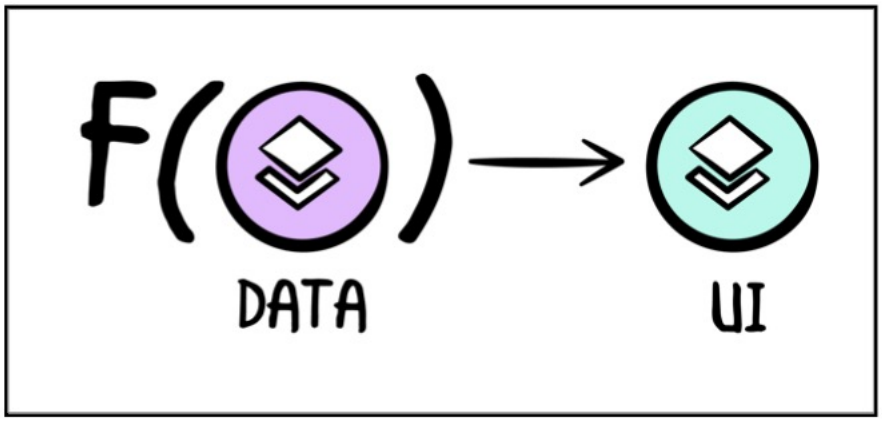
- UI is immutable: there are no objects
  - But UI is dynamic
  - Different inputs → Different UI
- UI is **idempotent**



Composables

- Functions that take data (state) as parameters and emit UI
  - Composable is immutable
  - Function is idempotent without side effects (no global variables)
  - Functions run in parallel – need to be thread safe

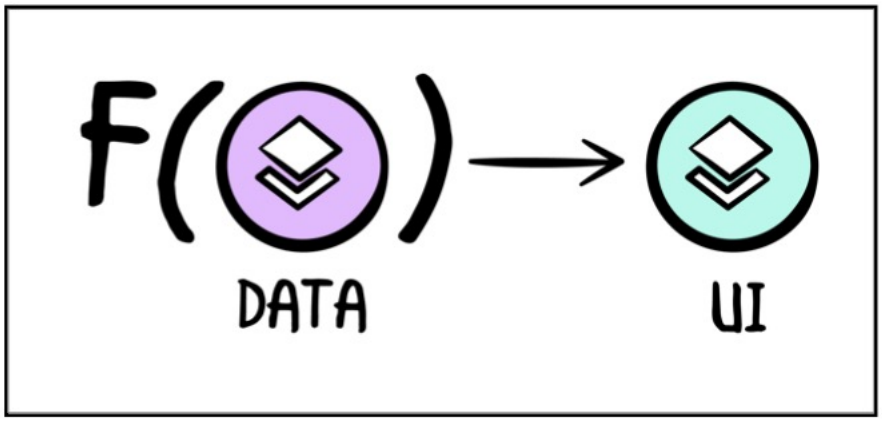
```
@Composable
fun Greeting(name: String) {
  Text(text = "Hello $name!")
}
```



Composables

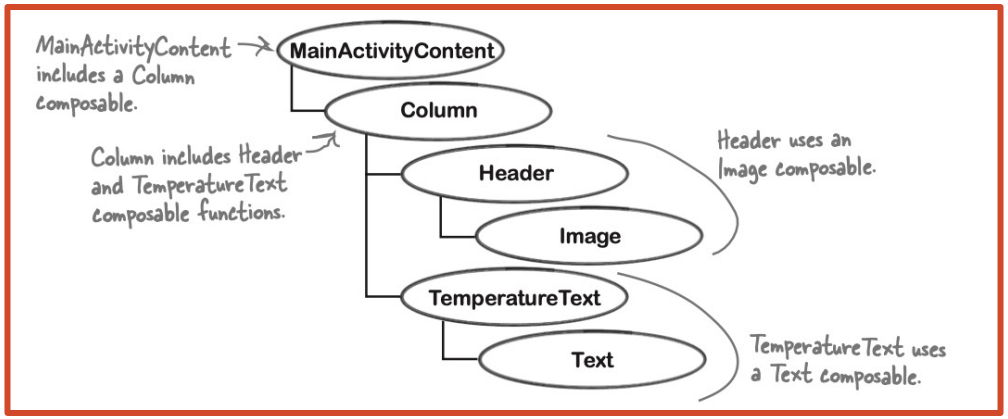
- Functions that take data (state) as parameters and emit UI
  - Composable is immutable
  - Function is idempotent without side effects (no global variables)
  - Functions run in parallel – need to be thread safe

```
@Composable
fun Greeting(name: String) {
  Text(text = "Hello $name!")
}
```



Recomposing

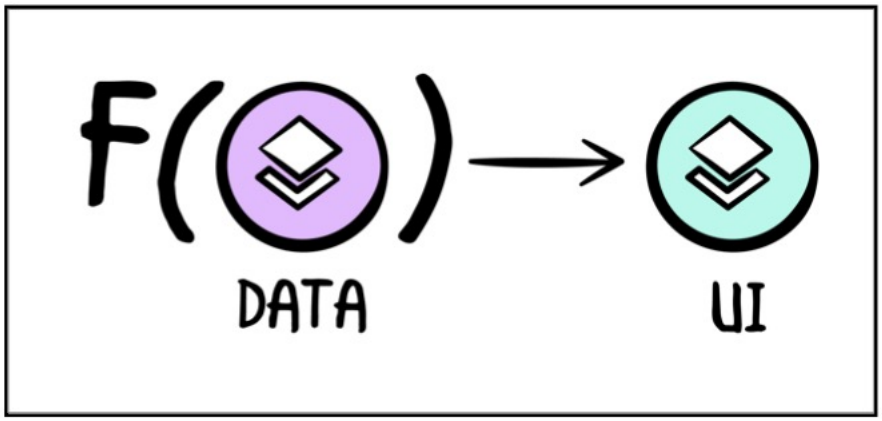
- Functions that take data (state) as parameters and emit UI
- If state changes, function is called again with new parameters and emits new UI
  - Compose compiler observes data and only remakes UI that has changed



Composables

- Functions that take data (state) as parameters and emit UI
  - Composable is immutable
  - Function is idempotent without side effects (no global variables)
  - Functions run in parallel – need to be thread safe

```
@Composable
fun Greeting(name: String) {
  Text(text = "Hello $name!")
}
```



Run In Any Order

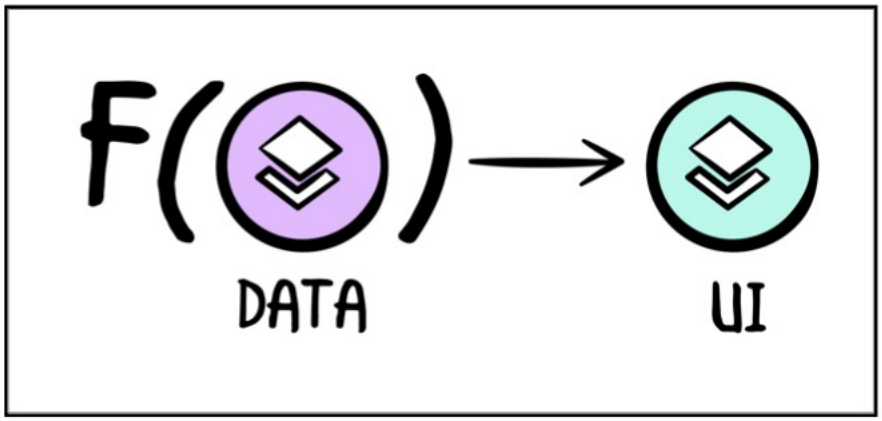
```
@Composable
fun ButtonRow() {
    MyFancyNavigation {
        StartScreen()
        MiddleScreen()
        EndScreen()
    }
}
```

- Can't have `StartScreen()` set some global (a side effect) and have `MiddleScreen()` depend on it
- Each composable function needs to be self contained

Composables

- Functions that take data (state) as parameters and emit UI
  - Composable is immutable
  - Function is idempotent without side effects (no global variables)
  - Functions run in parallel – need to be thread safe

```
@Composable
fun Greeting(name: String) {
  Text(text = "Hello $name!")
}
```



Not Side Effect Free

```
@Composable
@Deprecated("Example with bug")
fun ListWithBug(myList: List<String>) {
    var items = 0

    Row(horizontalArrangement = Arrangement.SpaceBetween) {
        Column {
            for (item in myList) {
                Text("Item: $item")
                items++ // Side-effect of the column recomposing
            }
        }
        Text("Count: $items")
    }
}
```

- If **`Column`** recomposes, **`items`** continues to increment
- **`Text`** remains the same though

Side Effect Free

```
@Composable
fun ListComposable(myList: List<String>) {
    Row(horizontalArrangement = Arrangement.SpaceBetween) {
        Column {
            for (item in myList) {
                Text("Item: $item")
            }
        }
        Text("Count: ${myList.size}")
    }
}
```

- If **`myList`** changes, **`ListComposable`** recomposes which recomposes **`Column`** and **`Text`**

However

- Sometimes, Side Effects can't be avoided
- There can be needs to mutate the state
- Do it from a controlled environment
  - Use the Effect APIs

LaunchedEffect

- Runs suspend functions in scope of a composable as a coroutine
  - When **`LaunchedEffect`** leaves composition, coroutine cancelled
  - If **`LaunchedEffect`** key changes, existing coroutine cancelled and new coroutine launched
- Use case: one-time per composition actions
- Use case: a state change triggers a second required state change

LaunchedEffect

```
LaunchedEffect(null) {
   // will run once
   // can be used with splash screens
}

LaunchedEffect(state.value) {
   // will run whenever state's value changes
   // can trigger additional dependent
   //    state changes
}
```

Splash Screen

```
@Composable
fun ScreenWithSplash() {
  val showSplash = remember { mutableStateOf(true) }
  LaunchedEffect(null) {
    // will run once
    // can be used with splash screens
    delay(3000L) // 3 seconds, set for length of splash
    showSplash.value = false
  }
  if (showSplash.value) {
    // draw splash screen
  } else {
    // draw regular screen
  }
}
```

Demo

- Samodelkin Splash Screen!

State Changes

```
LaunchedEffect(locationState.value) {
   // will run whenever state's value changes
   // can trigger additional dependent
   //    state changes
   locationUtility.getAddress(locationState.value)
}
```

LaunchedEffect

```
LaunchedEffect(state.value, state2.value) {
   // will run whenever state's value or
   //    state2's value changes
}
```

DisposableEffect

- When an effect requires cleanup upon completion/cancellation
  - Must end with **`onDispose`** block

```
DisposableEffect(value) {
    // create observer
    // do some work
    onDispose {
        // remove observer
    }
}
```

SideEffect

- Runs upon successful Composition
  - Can be used to share Compose state with object not managed by Compose

```
SideEffect {
   // do some work after composing
}
```

To Do For Next Time

- Beta Feedback due tomorrow
- Lab11 due Monday
- A3 due Tuesday