# Mobile Applications
# CSCI 448
# Lecture 06

Stateless Composables

+

View Model

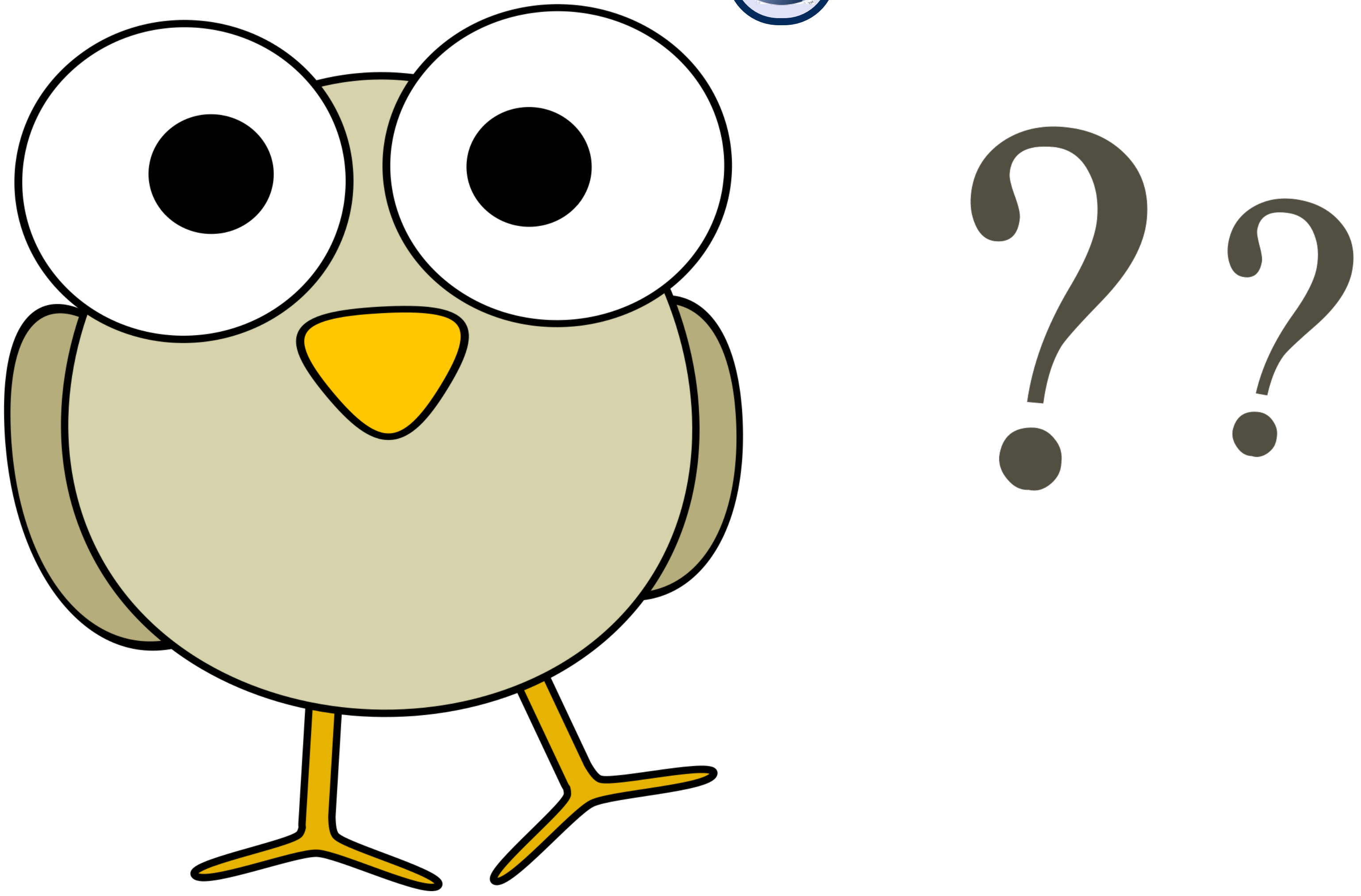
Have TempConverter

Loaded To Continue Implementing

## Previously in CSCI 448

- Stateful Composables
  - Store and track their own state
  - Observe a value
    - When changed → Recompose

# Questions?



# Learning Outcomes For Today

- Explain how a stateless composable stores and modifies state.
- Define the Decorator design pattern and map its application to View Models
- Create an app that uses stateless composables.

# On Tap For Today

- Stateless Composables

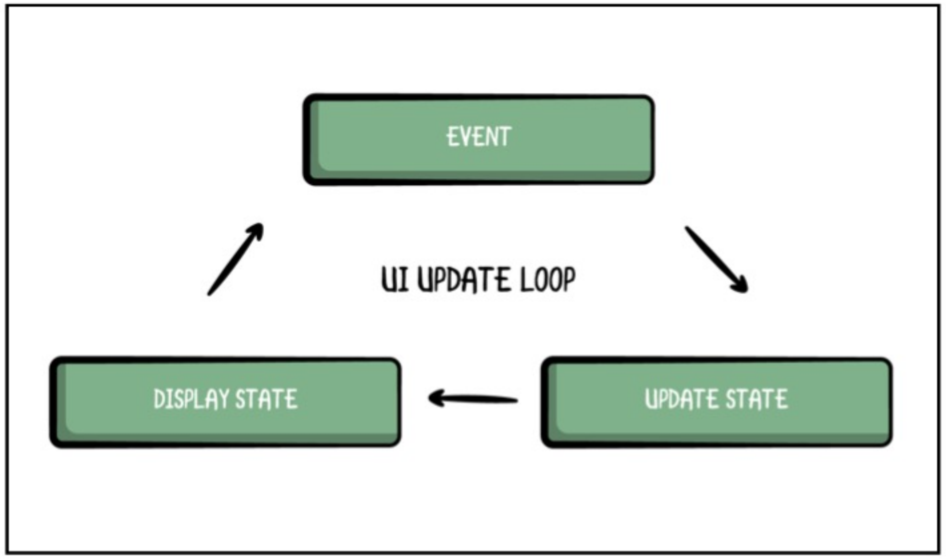
- View Model

# On Tap For Today

- Stateless Composables
- View Model

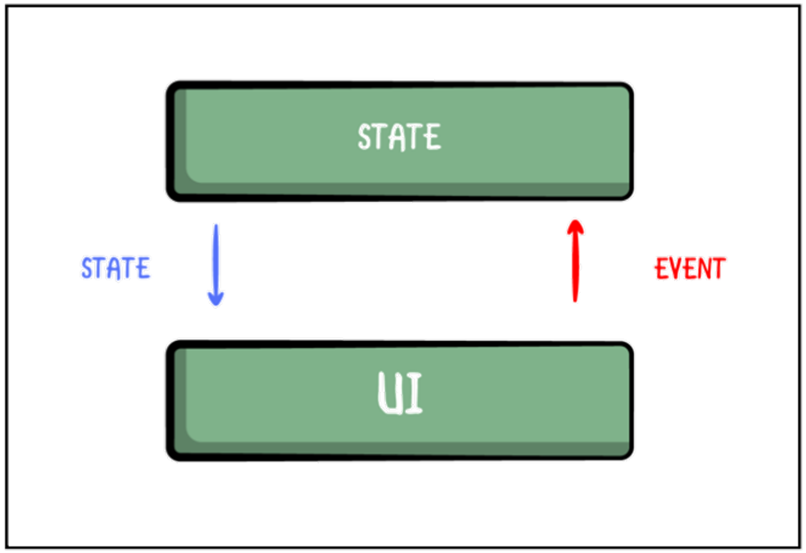
# Unidirectional Data Flow

- External events trigger change in state



# Single Source of Truth

- Keep one state
- State “flows” down
- Events “flow” up
- UI “observes” the state



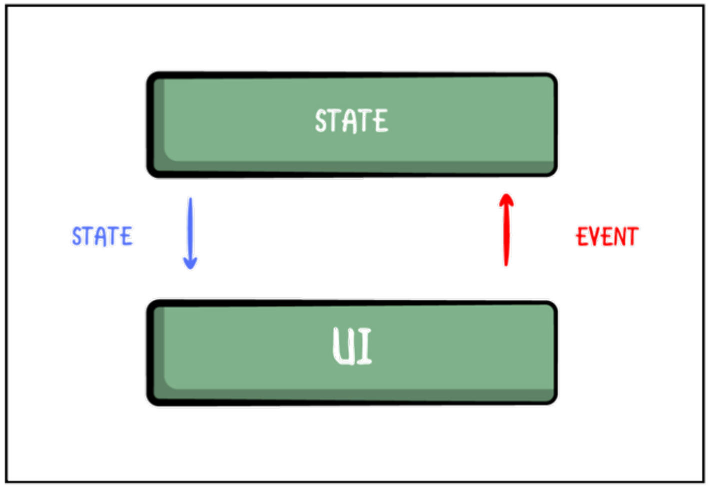
*Unidirectional Data Flow*

# Where to Store State???

- What is the single source of truth?
  1. Composable - In the composable itself
     - A **stateful** composable
     - Can change state itself
  2. ViewModel - “Hoist” the state to the caller of the composable
     - A **stateless** composable
     - Composable requires parameter and event
  3. StateHolder
     - Separate class that stores UI logic & UI element states

# Stateful Composables

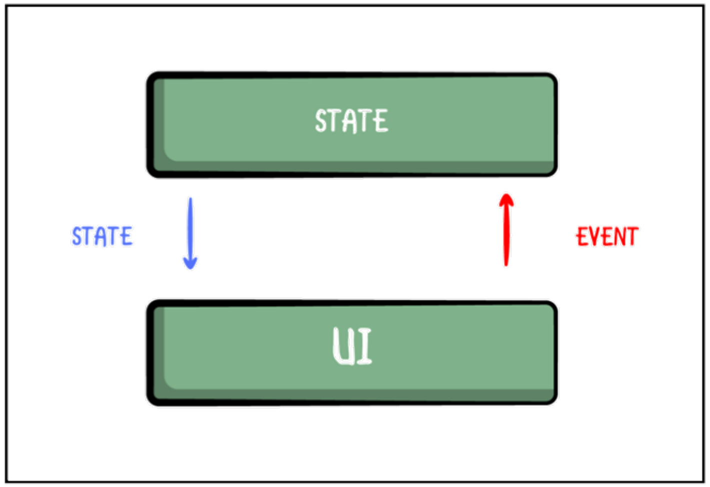
- State stored in the UI
- The UI handles events to update that state
- Compsable does both



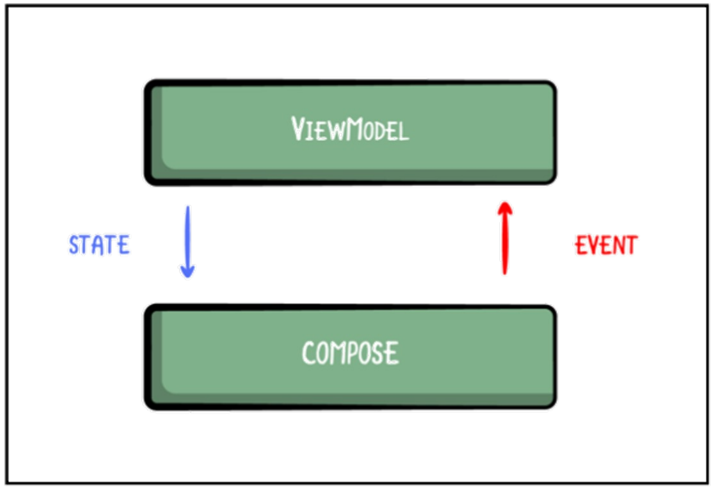
*Unidirectional Data Flow*

# Stateless Composables

- “Hoist” state from the UI
- Define events that the UI can call to update that state



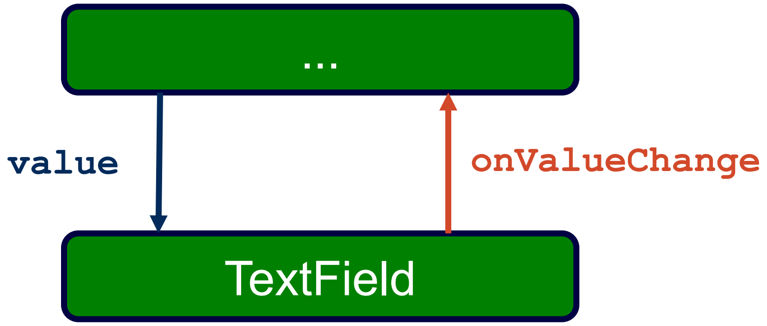
*Unidirectional Data Flow*



*Unidirectional Data Flow With Architecture Components*

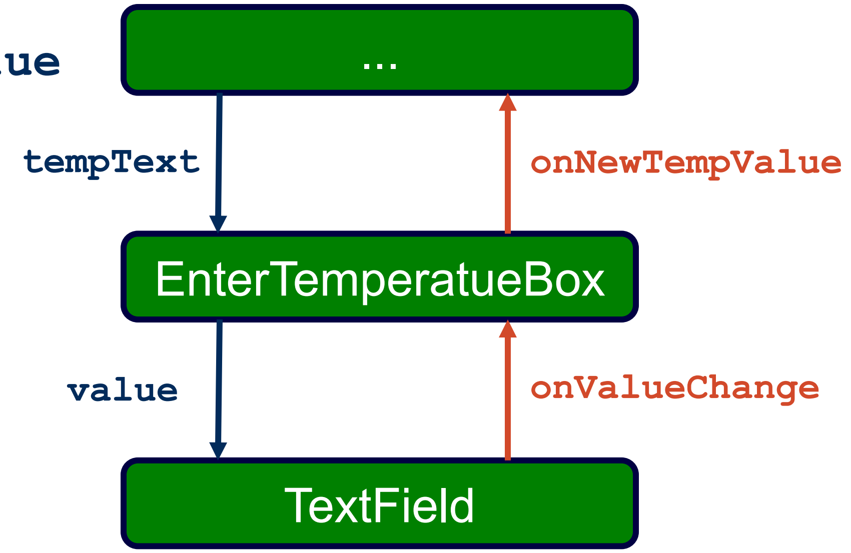
# Create Stateless Composables

```
TextField(
   value =           ,
   onValueChange =
)
```



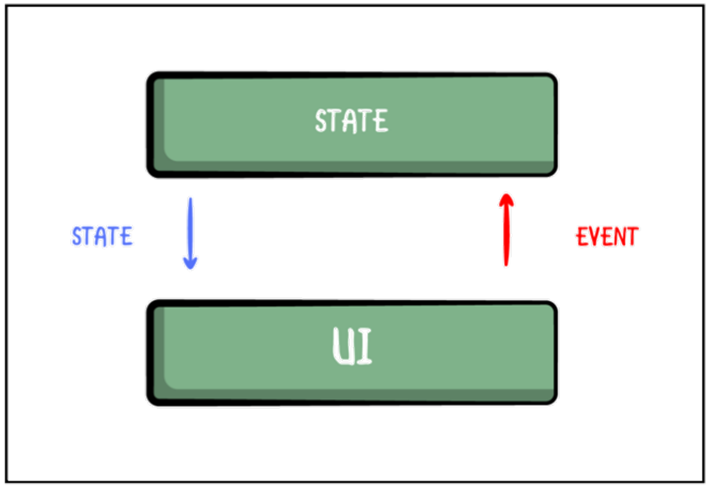
# Create Stateless Composables

```
@Composable
fun EnterTemperatureBox(
    tempText: String,
    onNewTempValue: (String) -> Unit
) {
  TextField(
    value = tempText,
    onValueChange = onNewTempValue
  )
}
```

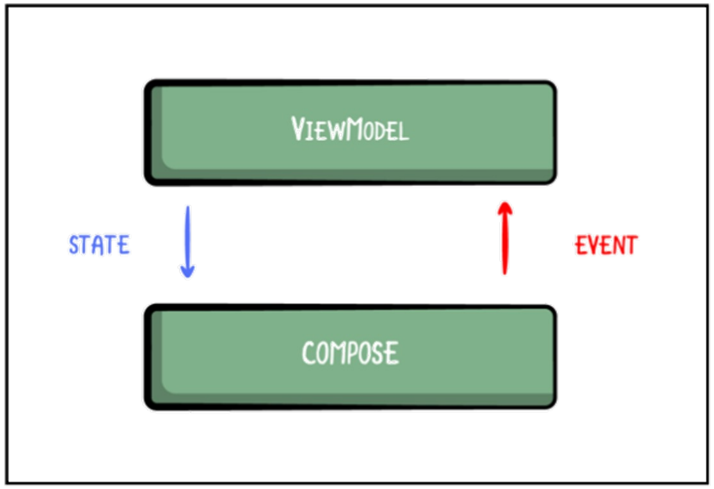


# Stateless Composables

- Extract state from the UI
- Define events that the UI can call to update that state
- View Model does both!



*Unidirectional Data Flow*



*Unidirectional Data Flow With Architecture Components*

# On Tap For Today

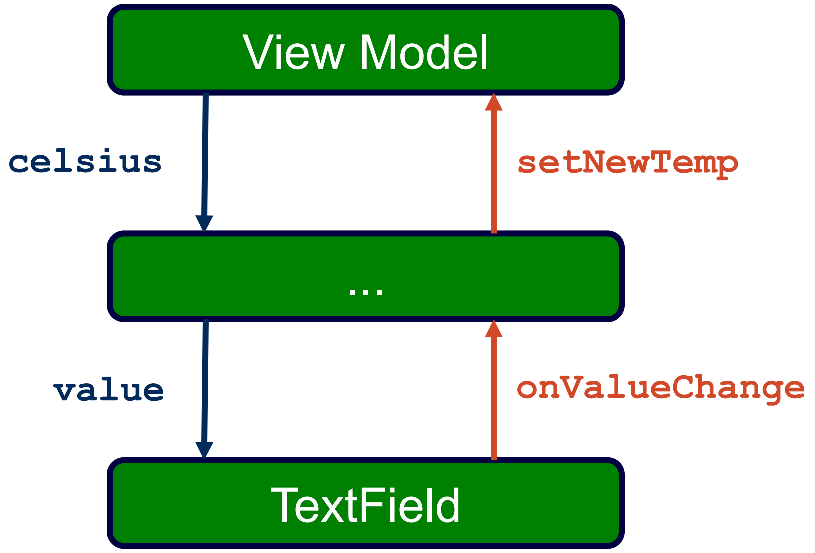
- Stateless Composables
- View Model

## View Model

- Related to a particular View and holds on to a Model object
- Formats Model data to display in View
- Aggregates all data for one screen in one place, formats the data, easy to access end result

# View Model To The Rescue!

- View Model stores Model state and Presentation Logic to manipulate Model for View
  - View Model *decorates* Model for View



# *Design Pattern #3: Decorator*

- Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality
- Participants:
  - `Component`: defines the interface for objects that can have responsibilities added to them
  - `Decorator`: maintains a reference to a `Component` object and defines an interface that conforms to the `Component`'s interface
  - `ConcreteComponent`: defines an object to which additional responsibilities can be attached
  - `ConcreteDecorator`: adds responsibilities to the component

# View Model Decoration

- `Component` →
- `Decorator` →
- `ConcreteComponent` →
- `ConcreteDecorator` →

# Android Design Patterns

- Behavioral Patterns
  1. Command – UI Event Handling
  2. Observer – State
- Structural Patterns
  3. Decorator – View Model

# Making the View Model

- View Model gets data from Model
- View interacts with View Model

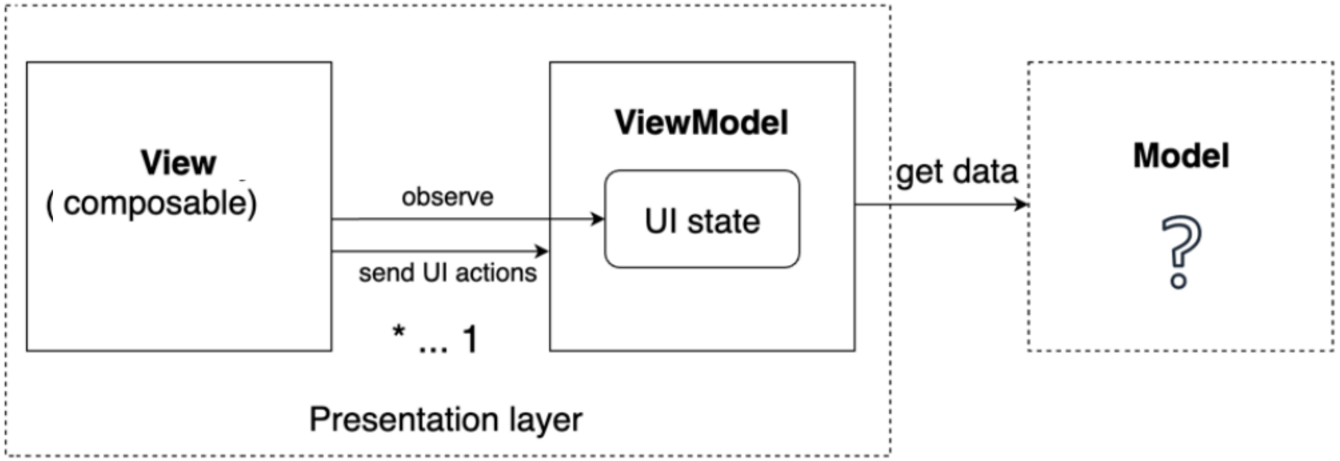


Figure 7.4 – Presentation layer in the MVVM pattern

## On Tap For Today

- Stateless Composables
- View Model

# To Do For Next Time

- Kotlin Collections quiz
  - Due by end of Monday
  - Access code: **`seabees`**