CMPS 312

Read Chapter 1

Android Fundamentals

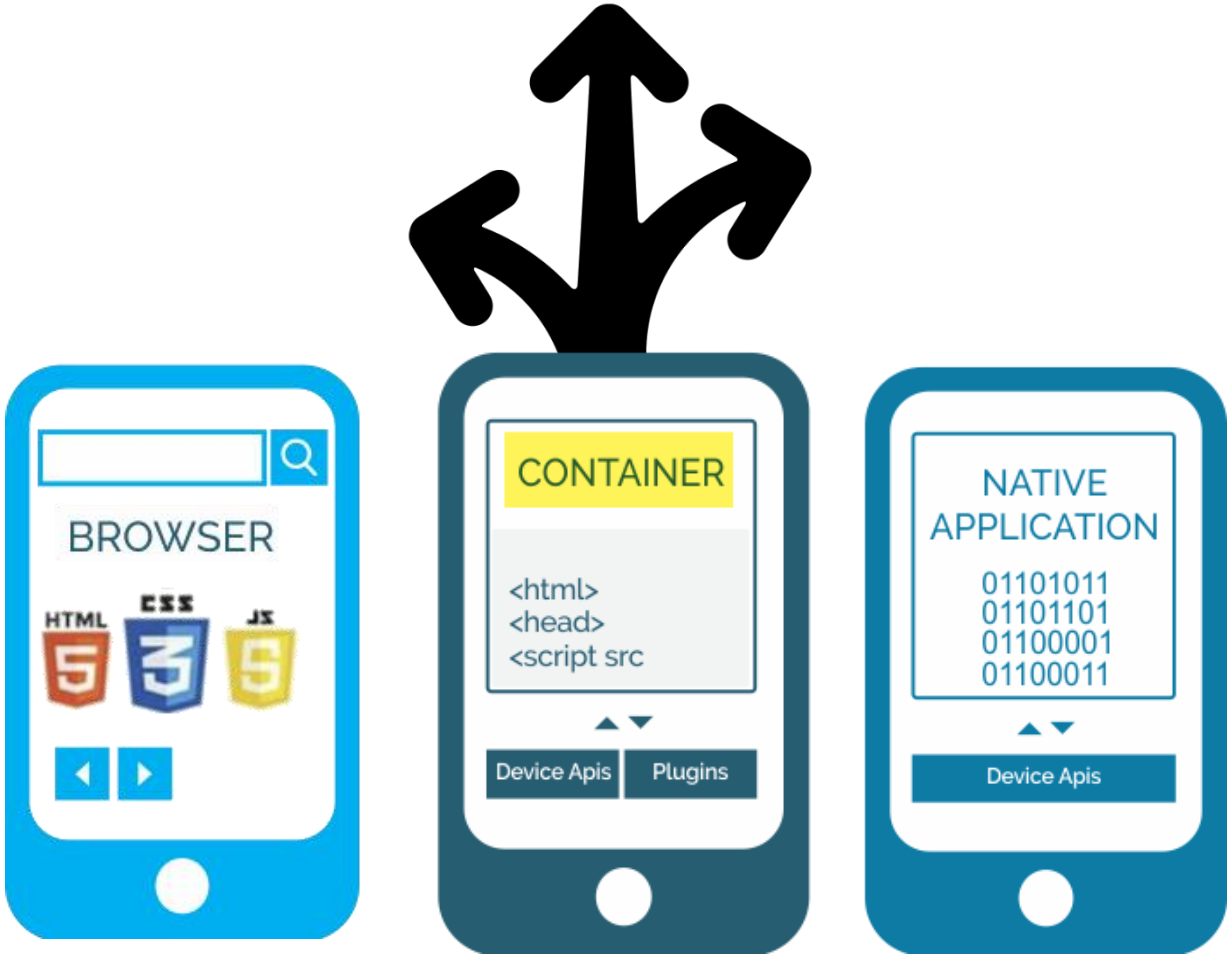
Dr. Abdelkarim Erradi

CSE@QU

Outline

1. Mobile Development Approaches
2. Introduction to Android
3. Imperative UI vs. Declarative UI

Mobile Development Approaches

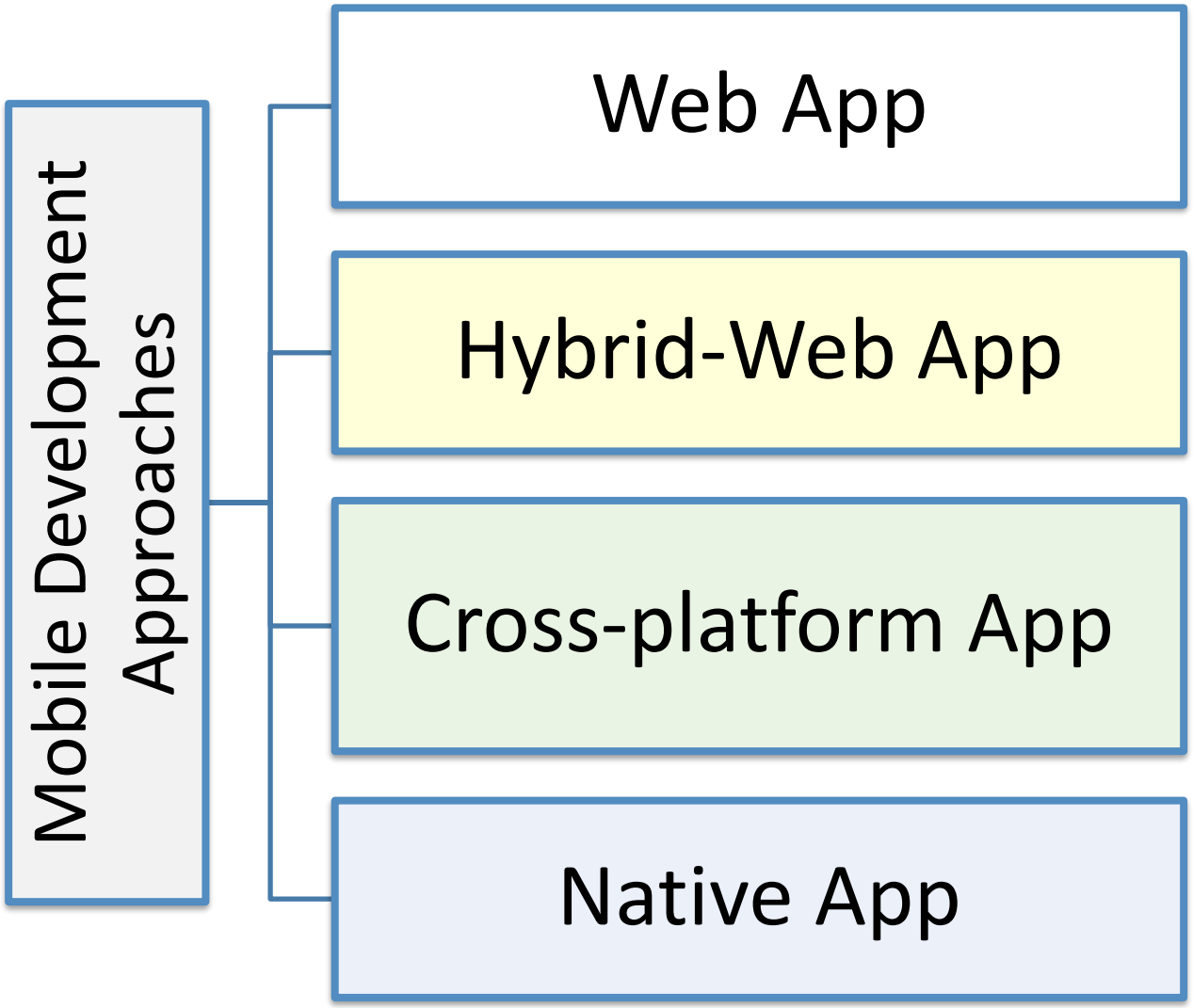


Back

Why learn app development?

- Smart devices are ubiquitous
  - Billions of smartphones, tablets, smart watches, ...
  - Apps interwoven into daily life – work, play, study
  - Mobile = **dominant** end-user device. It represents and intimately "knows" the user: much more than just a PC, it represents the user
  - Connected to the outside world: **sensing**, **location**, **communication**
- Apps less expensive and more portable
- Large market opportunity for businesses and developers

Mobile Development Approaches



Web App

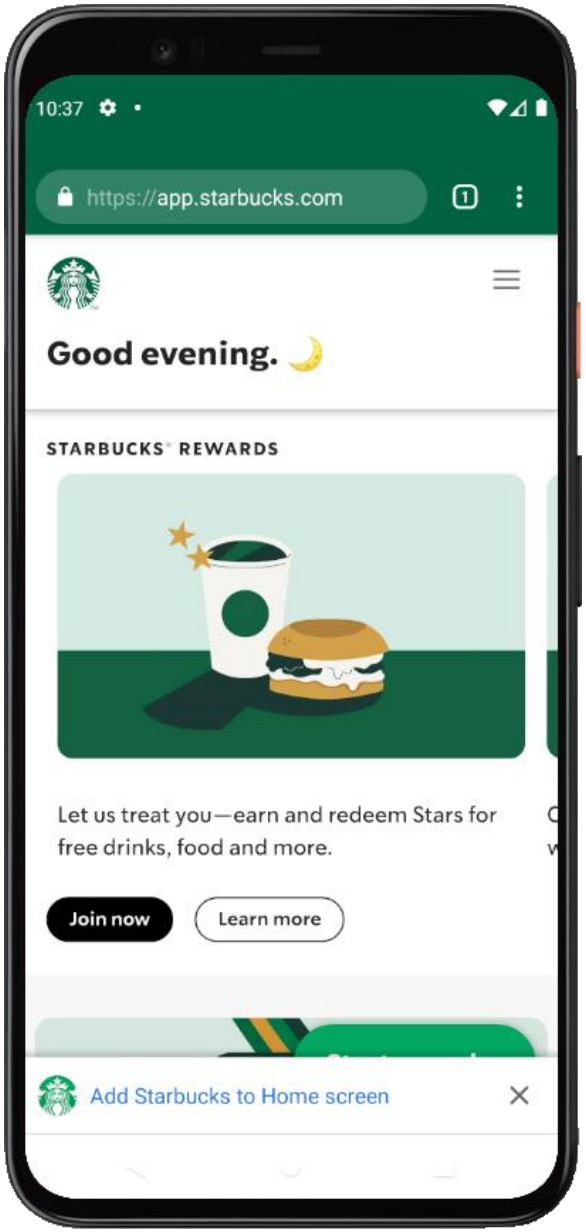
- Responsive Web app adapted to any screen size
- Can be added to Home screen & can work on any platform
- Experience feels like a native app

✓ Can work offline, provide limited access to device's features, such as camera, microphone, location, and notifications

– Slower performance (Run inside a WebView)

– Least access to hardware, sensors, OS

– Not available from the app stores



Hybrid-Web App

- Hybrid-Web Apps: apps blend
  - Mobile-optimized UI components (written using HTML, CSS, and JavaScript) with
  - Native modules or **bridge plugins** for accessing Camera, Geolocation, Bluetooth and other services

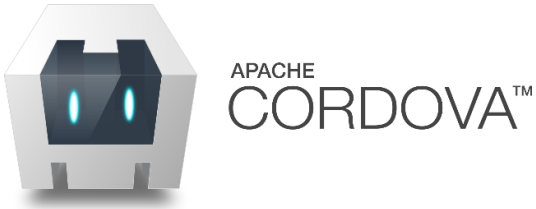
✓ Lower development costs (Single codebase)

✓ Multiplatform - Write once, run anywhere

✓ Downloadable from app stores

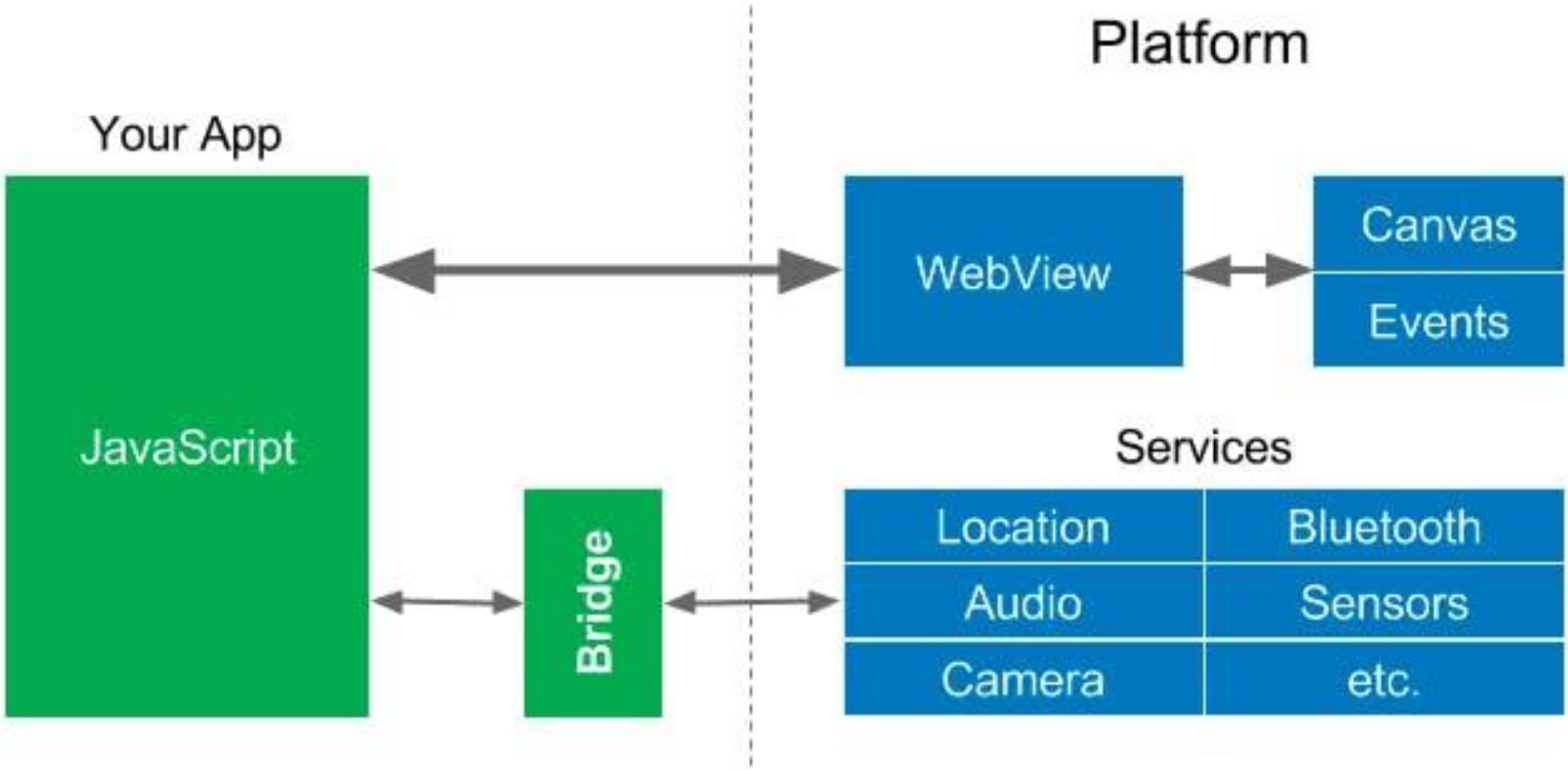
– Slower performance (not suitable for performance-intensive apps such as 3D games)

– Highly dependent on libraries and frameworks



Hybrid-Web App

- App runs inside a **WebView** responsible for UI Rendering
- App access the platform services via a **bridge**



https://wajahatkarim.com/2019/11/how-is-flutter-different-from-native-web-view-and-other-cross-platform-frameworks/

Cross-platform App

- Cross-platform mobile development frameworks can be used to build native-looking apps for multiple platforms, such as Android and iOS, using a single codebase

✓ Lower development costs (Multiplatform utilizing a single codebase)

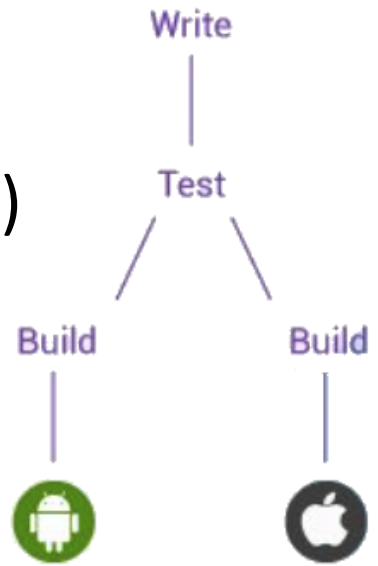
✓ Leverage existing skillset (JavaScript, React, Dart)

✓ UI performance is almost as fast as native

✓ Downloadable from app stores

– Highly dependent on libraries and frameworks

– Delayed update to latest native APIs



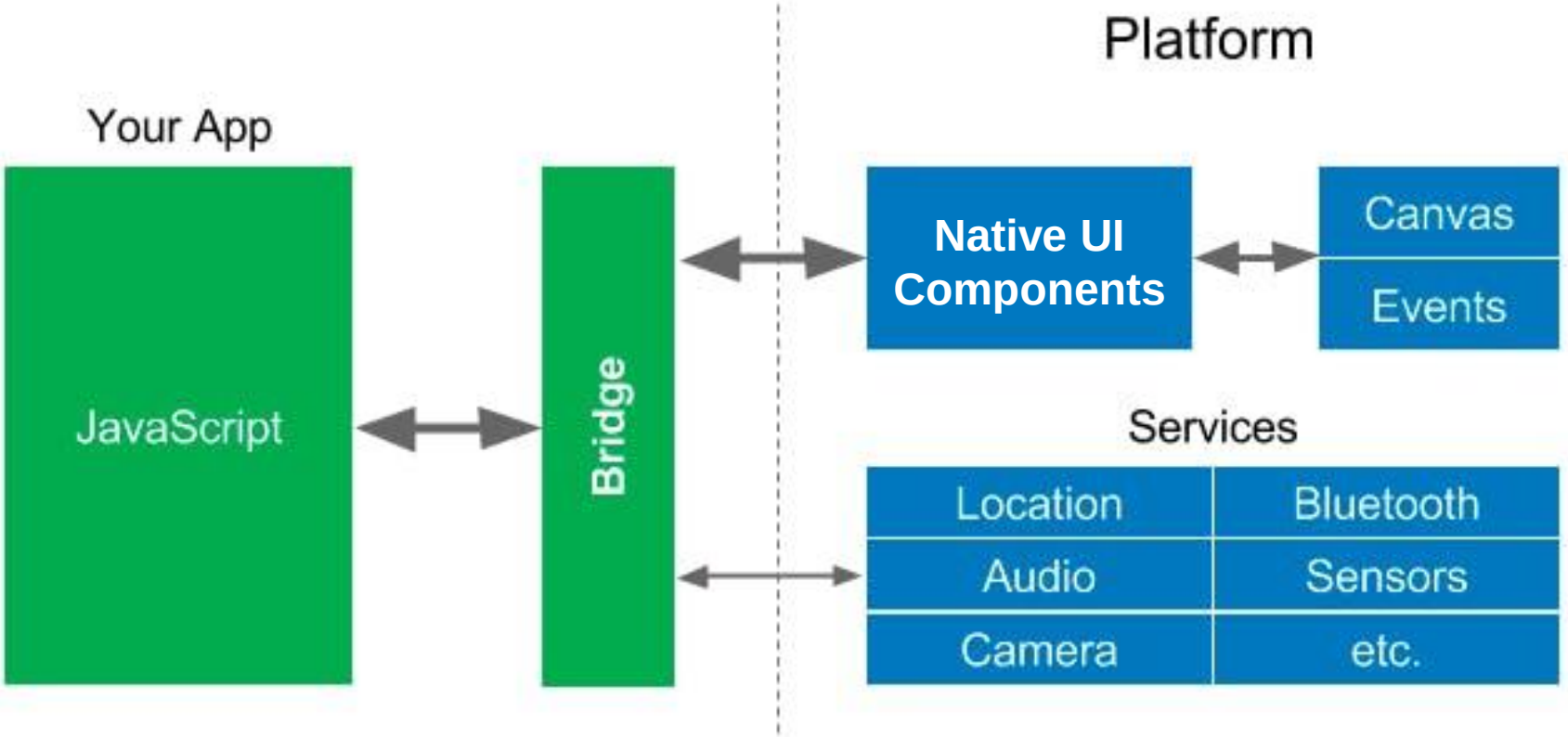
Flutter

React Native

Kotlin Multiplatform

React Native Compiles JavaScript UI components into equivalent **native UI** elements

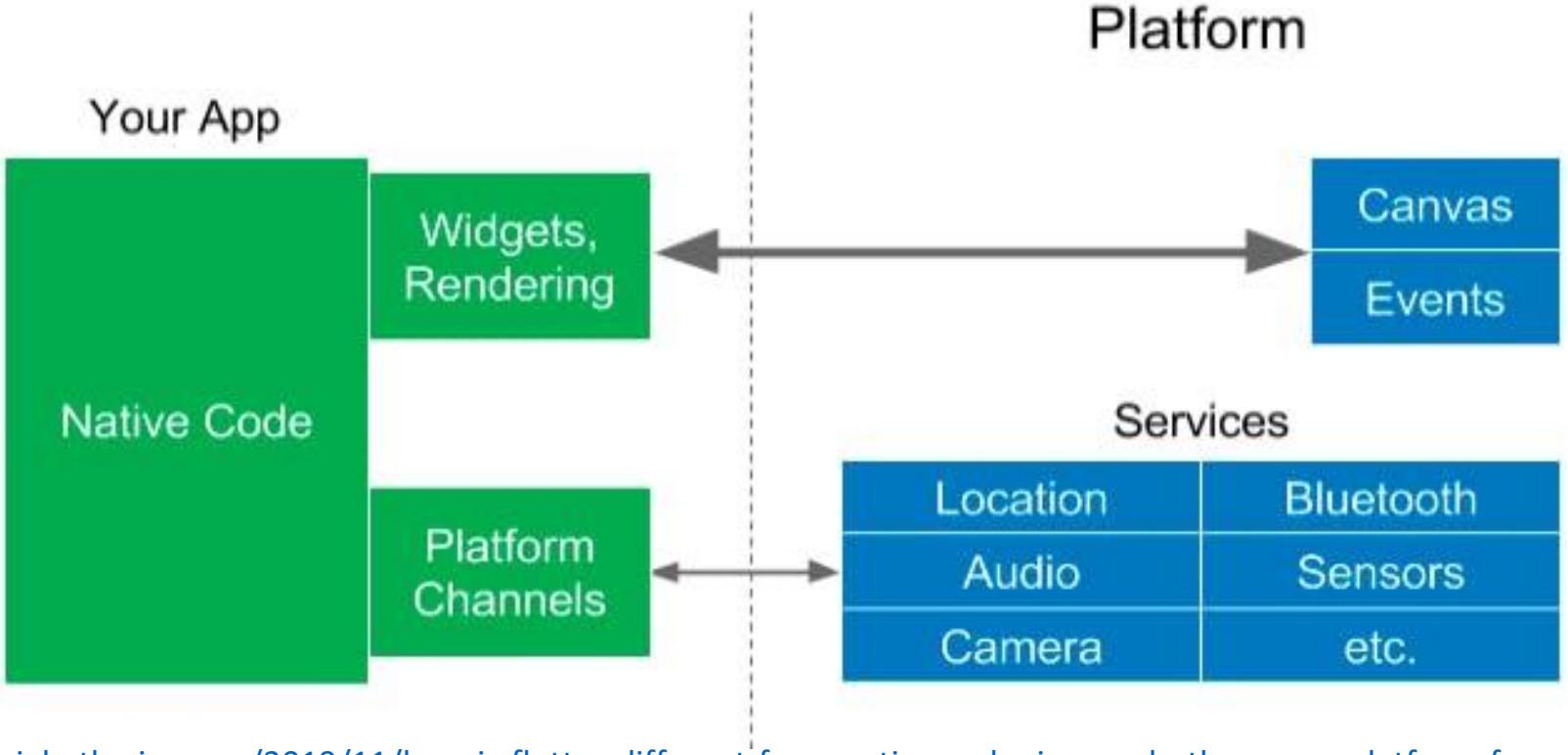
- Remaining code doesn't get compiled, instead runs in a separate JavaScript thread
- App interact with UI and access the platform services via a **bridge**



https://wajahatkarim.com/2019/11/how-is-flutter-different-from-native-web-view-and-other-cross-platform-frameworks/

Flutter

- Flutter App (written in Dart) is **compiled into native code**, UI uses Flutter own custom widgets rendered by the framework's **graphics engine** (https://skia.org/) to work across devices.
- App uses Platform Channels to access the platform services



https://wajahatkarim.com/2019/11/how-is-flutter-different-from-native-web-view-and-other-cross-platform-frameworks/

iOS or

Native App

- Uses platform-specific (Android/iOS) UI components and API

✓ Access to all native APIs, hardware, sensors, & OS

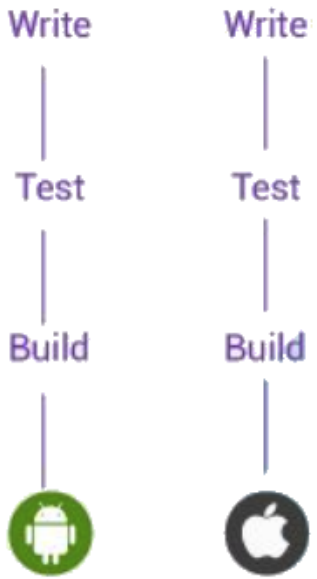
- No third-party dependencies

✓ Fast **performance** as it run directly on OS

✓ High-quality User Experience (UX)

– No codebase reuse

– High dev cost and longer time to market: requires **multiple code bases** and teams

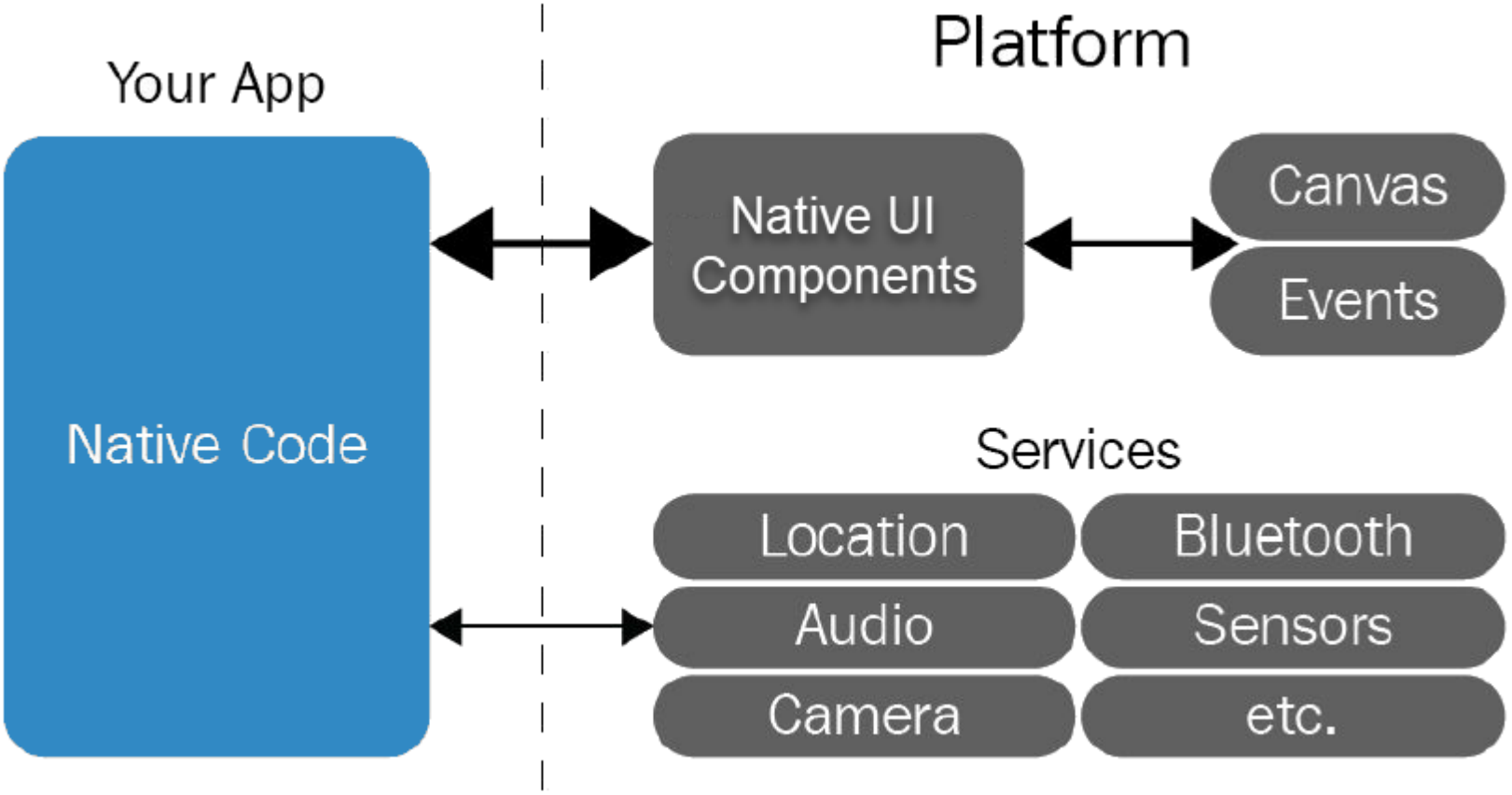


android

iOS

Native Android/iOS Platforms

The **app** has direct access to the platform services



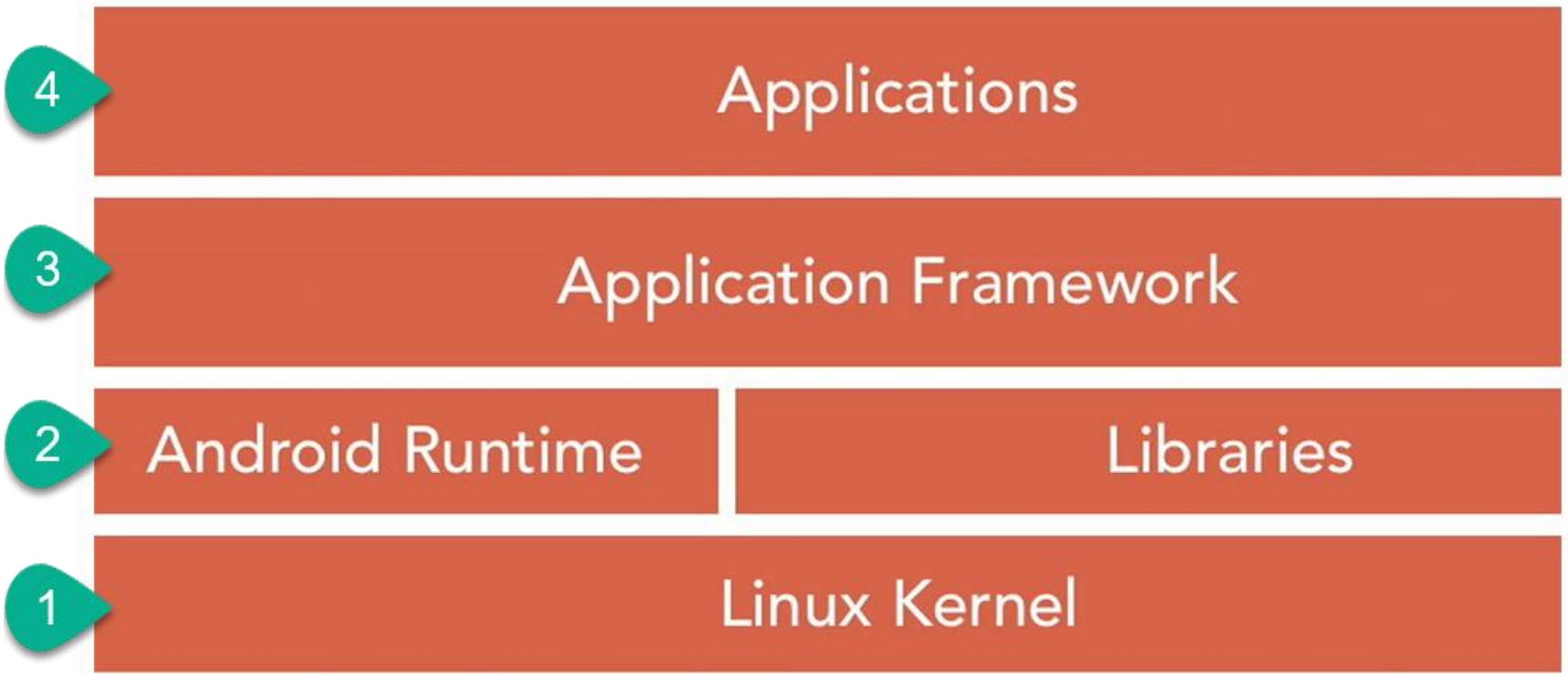
https://wajahatkarim.com/2019/11/how-is-flutter-different-from-native-web-view-and-other-cross-platform-frameworks/

Introduction to Android

What is Android?

- Open-source mobile operating system (OS) based on Linux kernel for phones, tablets, wearable
  - originally purchased by Google from Android, Inc. in 2005
- The #1 OS worldwide
  - Used on over 80% of smartphones
  - As of 2023, over 3.6 billion Android devices worldwide
  - Over 3.5 million Android apps in Google Play store
- Highly customizable for devices by vendors

Android Software Stack



1. Interacts and manages hardware
2. Expose native APIs & run apps
3. Java API exposing Android OS features
4. System and user apps (e.g., contacts, outlook)

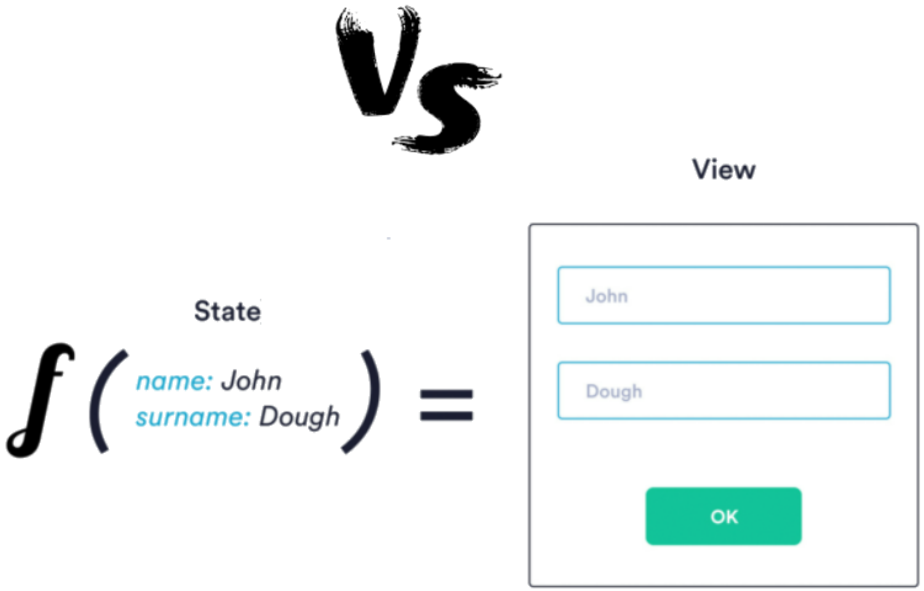
Android Software Stack

1. Optimized **Linux Kernel** for interacting with the device's processor, memory and hardware drivers (e.g., WiFi Driver)
   - Acts as an abstraction layer between the hardware and the rest of the software stack
2. **Android Run Time (ART)** = Virtual Machine to run Apps
   - Each app runs in its own process and with its own instance of the Android Runtime that controls the app execution (e.g., permission checks) in isolation from other apps
   - Expose native APIs and OS Core Libraries including 2D/3D graphics, Audio Manager, SQLite database, encryption …
3. **Application Framework**: Java APIs (Application Programming Interfaces) make Android OS features available to Apps (e.g., Activity Manager that manages the lifecycle of apps)

https://developer.android.com/guide/platform

Imperative UI vs. Declarative UI

```
TextView greetings = (TextView) findViewById(R.id.tv_greeting)
greetings.text = “Hello world.”
```



Imperative UI vs. Declarative UI

👑 In **Imperative UI**, the steps to create the UI are **explicitly and fully** defined and then it is updated using methods / properties of the UI elements

- To change the view the developer, need to specify <u>when</u> to change and <u>how</u> to change the view to display the current data

📢 In **Declarative UI**, Describe what the Ul should look like & the state data to feed to the UI

- The UI runtime has the responsibility to <u>observe</u> the state changes then <u>automatically update</u> the UI to reflect state changes

Imperative vs. Declarative UI

Imperative:

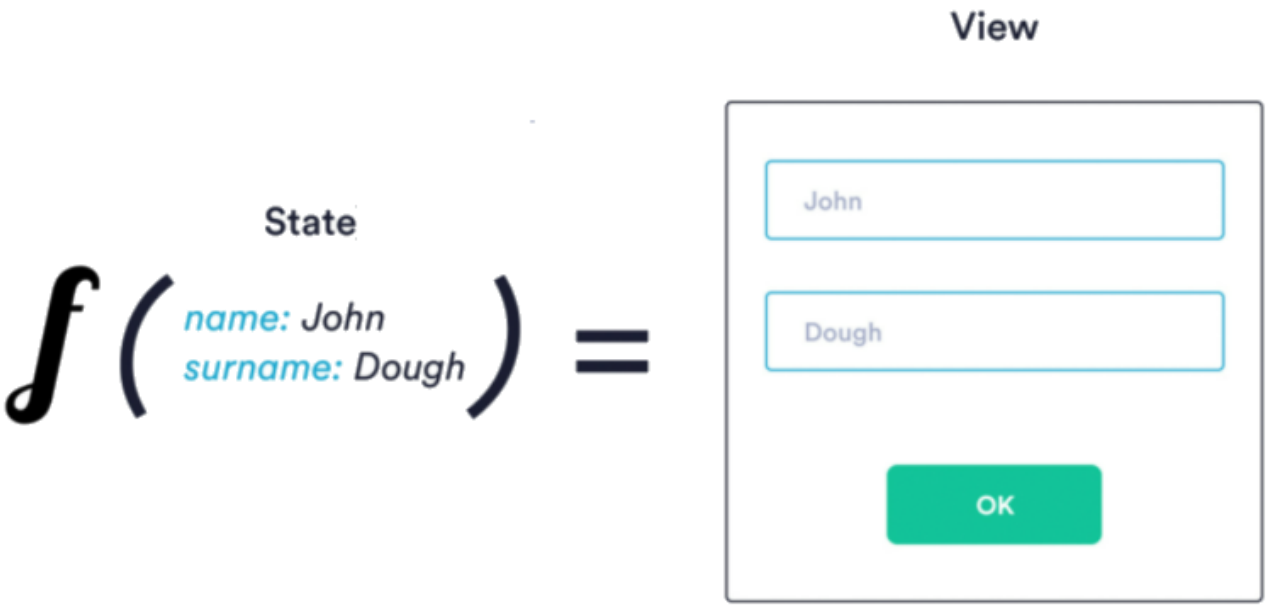
- Lots of boilerplate and boring code
- Errors and bugs prone: e.g., if a piece of data is rendered in multiple places, it's easy to forget to update one of the views that shows it
- Hard to maintain

Declarative: Describe WHAT to see NOT HOW

- ✓ Less code to write → Fewer bugs and more flexible
- ✓ State changes trigger automatic update of the UI to reflect state changes
- ✓ Improves reusability of UI components

Declarative UI

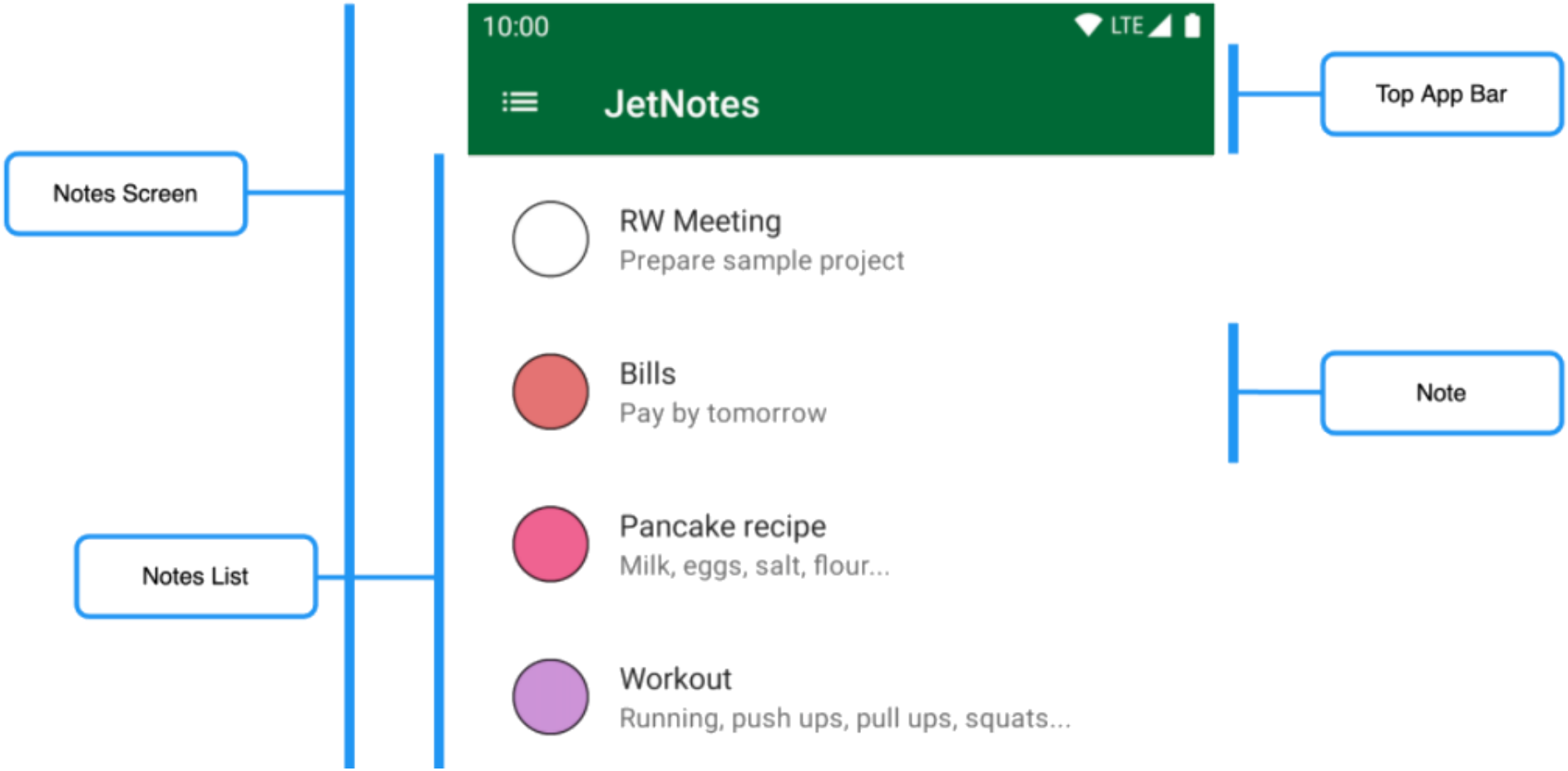
- Describe what elements you need in your UI and to a degree what they should look like
- **UI = f(state)** : UI is a visual representation of state
- State changes trigger automatic update of the UI
  - Eliminates the need to imperatively sync the UI state



Mobile App UI Design Process

1. Design the UI wireframe (sketch)
   - Decide what information to present to the user and what input they should supply
   - Decide the UI components and the layout on paper or using a design tool such as Figma
   - Design the app navigation through the screens to achieve the app use cases
2. Breakdown the UI into small reusable UI components (building blocks) that work together to make the whole screen
3. Use a bottom-up approach:
   - Start implementing the smaller UI components and build your way up through the design
   - For each UI component, identify the data needed (app state) and events raised to notify the app logic
   - Manage app state and data exchange between UI components & app logic to respond to the user actions
   - Compose the screens from building block components and arrange them using appropriate layouts

Example - UI decomposition into UI Components



UI Sketch - Example

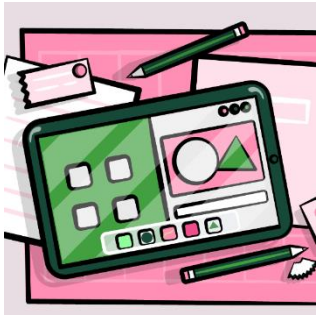


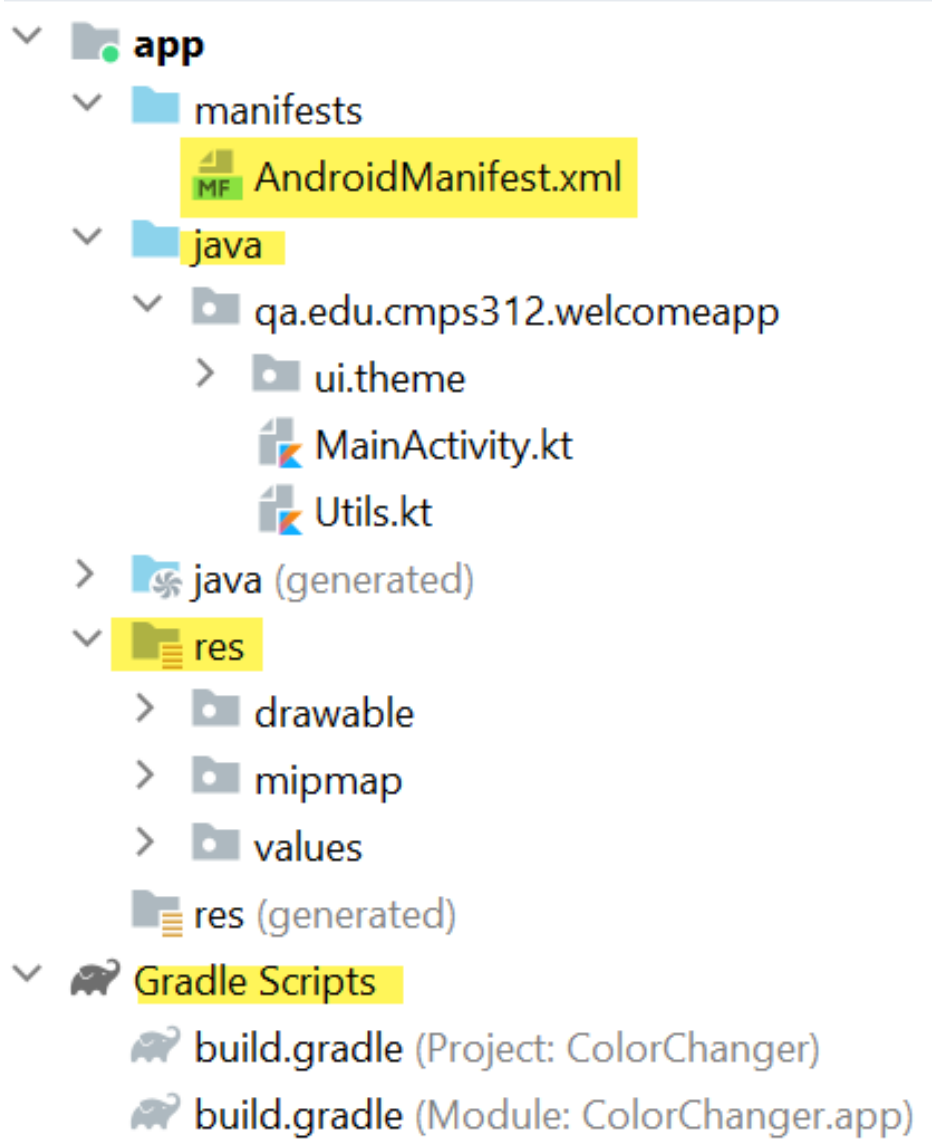
Fig 1. Home screen

Fig 2. Food places

You may design different layouts per screen size

Source: https://balsamiq.com/learn/articles/mobile-app-wireframing-guide/

Android Project structure



- **AndroidManifest.xml**
  - app config and settings (e.g., list app activities and required permissions)
- **java**/...
  - Kotlin source code
- **res**/... = resource files *(many are **XML**)*
  - drawable/ = images
  - Mipmap = app/launcher icons
  - values/ = Externalize constant values
- **Gradle**
  - a build/compile management system
  - **build.gradle** = define config and dependencies (one for entire project & other for app module)

Resources

- Comparing Cross-Platform Frameworks
  - https://ionic.io/resources/articles/ionic-vs-react-native-a-comparison-guide
- Android Kotlin Fundamentals Course
  - https://codelabs.developers.google.com/android-kotlin-fundamentals/
  - https://developer.android.com/courses/android-basics-kotlin/course
- Android Dev Guide
  - https://developer.android.com/guide/