MOBILE APPLICATION DEVELOPMENT

Lecture Two

Two Main Operating Systems “platforms”

- **iOS**
  - Developed by Apple, and it runs **exclusively on Apple products**.
  - Apple provides iOS developers with many **native tools** and **libraries** to develop iOS applications, and, though you do not have to be enforced to use Apple’s development tools to create your apps, you just need to have a mac running OS X to build your application.

- **Android**
  - Thousands of different devices of all shapes and sizes
  - Open Source, available to everyone.
  - Android is based on the Linux kernel, and Google releases the source code for Android as open source.

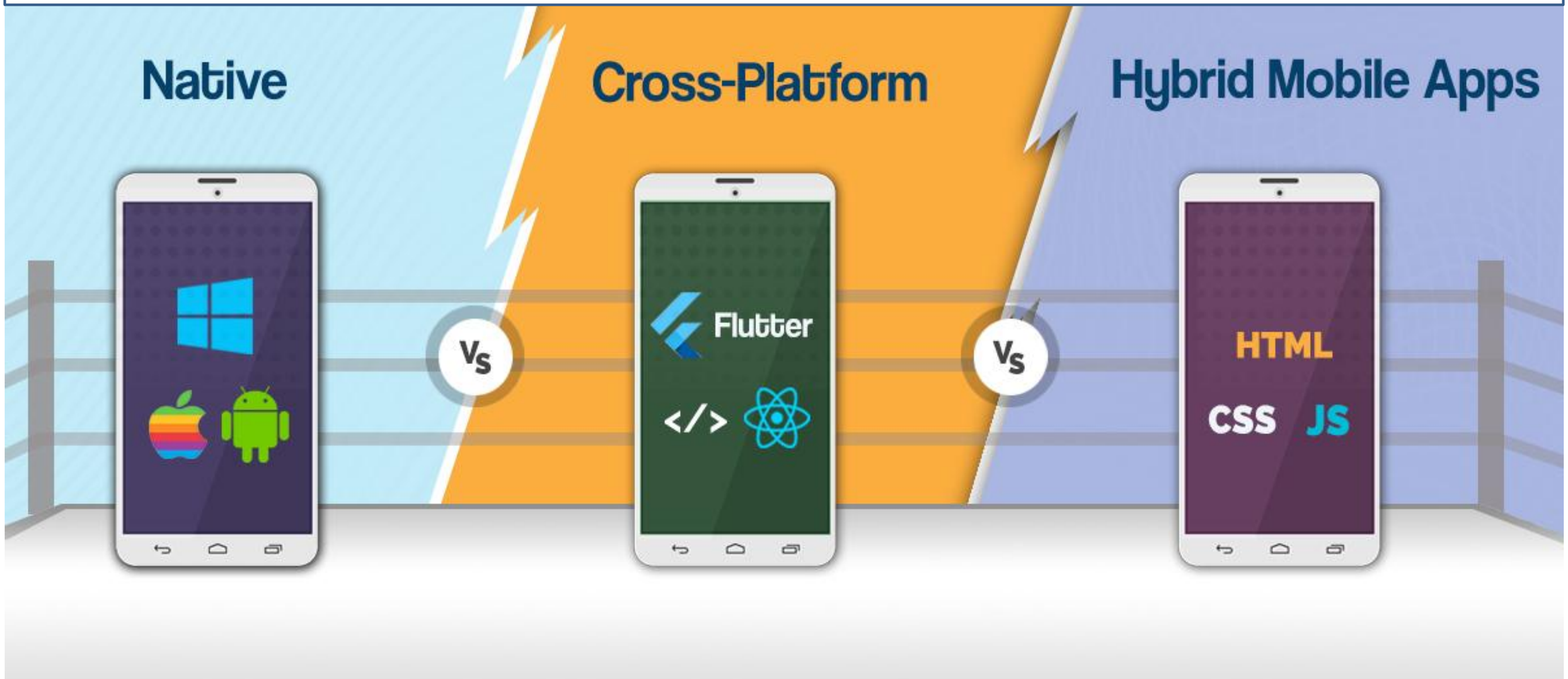
Programming languages and frameworks for mobile application development

- **iOS** apps are developed using the Xcode environment integrated into Mac OS X

“operating system for apple machines” and iOS, in Objective C, Swift, C and C ++.

- **Android** apps are developed using **Android Studio** and **Java, Kotlin** language.

Three main paradigms to develop mobile applications



Native Applications

- This type of application development simply means developing an app specific to each platform.

- Examples of specific tools can be extracted for iOS XCode to Objective-C to Swift or C++, For android-based apps, **Java or Kotlin** are typically used.

- One of the key elements of native applications for Android and iOS is that they’re fully supported by Google and Apple.

- They have access to all of the functionalities of these systems, they have their own SDKs, they have access to all native components.

- **Executed directly by the operating system**

Development approaches for mobile applications

Native applications

Written in the frameworks provided by the platform

1. iOS :

developed using Xcode

Objective C, Swift, C and C ++

2. Android

developed using **Android Studio**

Java and **kotlin**

- **Google Maps**
- **Games**

Cross-platform

In case you need your software to support multiple platforms.

– React Native

– Flutter

– Xamarin

• Hybrid mobile applications

Built with standard web technologies –

such as JavaScript, CSS, and HTML5 –

They are bundled as app installation packages.

Development approaches for mobile applications

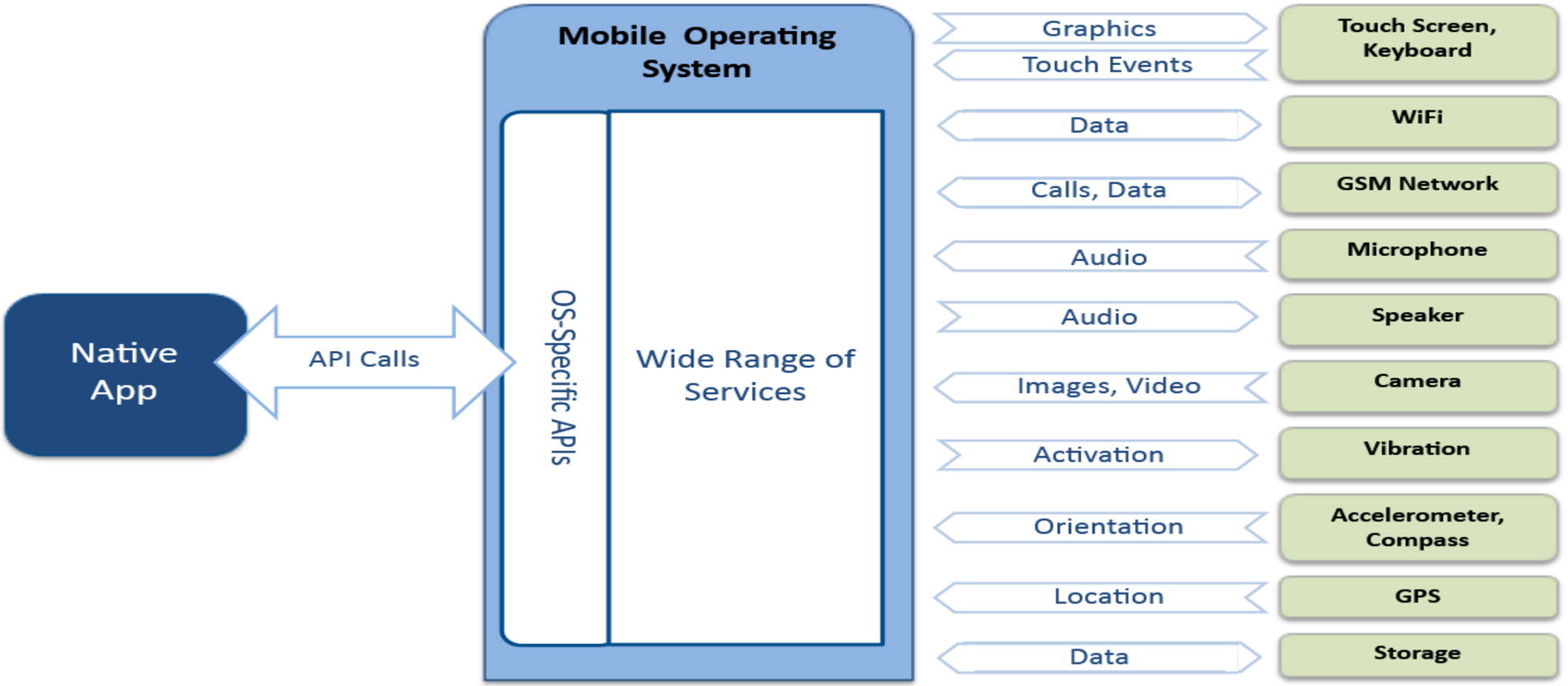
- **Native applications**
  - Native mobile applications are written in the programming language and frameworks provided by the platform
  - **iOS** apps are developed using the Xcode environment integrated into Mac OS X **iOS** “operating system for apple machines” and, in Objective C, Swift, C and C ++.
  - **Android** apps are developed using **Android Studio** using **Java** and **kotlin** language.

- **Cross-platform frameworks**

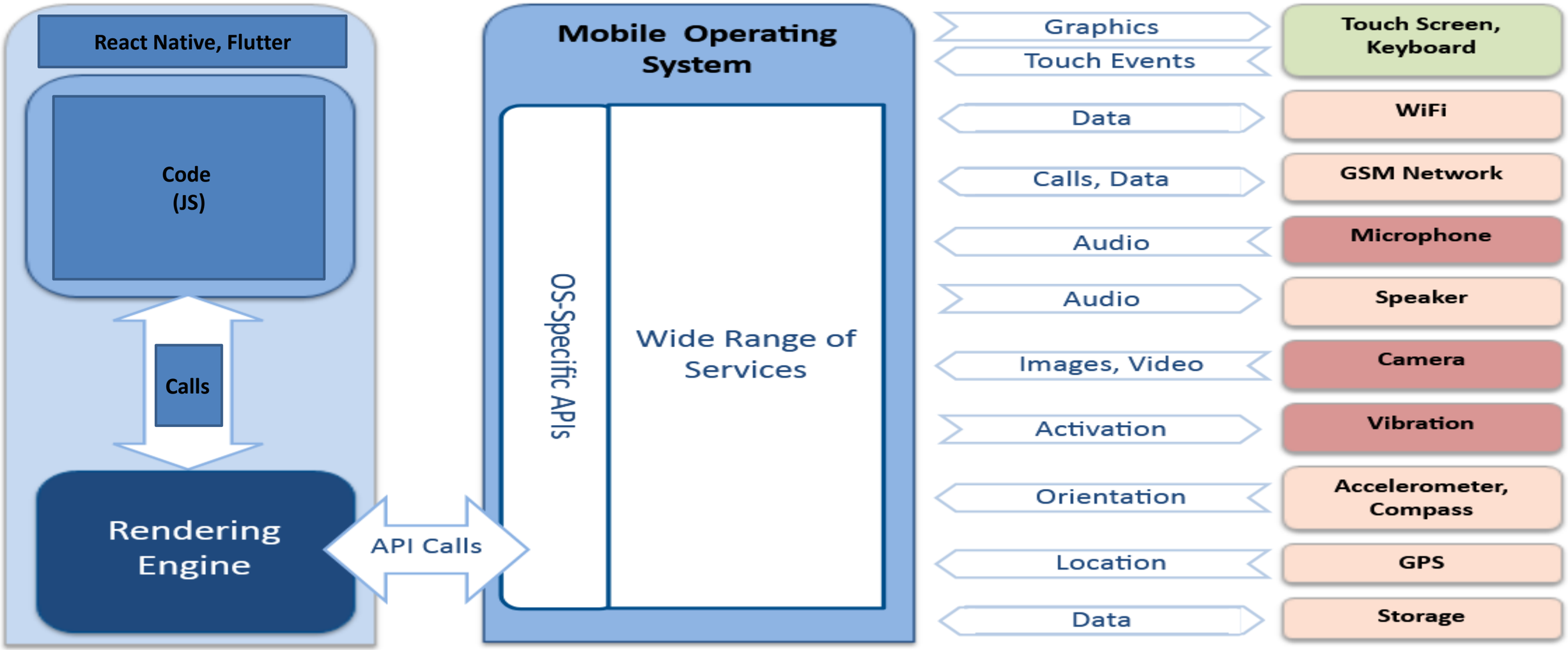
  Allow to reduce costs and accelerate the release of software in case you need your software to support multiple platforms. simultaneously.
  - **React Native**
  - **Flutter**
  - **Xamarin**

- **Hybrid mobile applications** are built with standard web technologies - such as JavaScript, CSS, and HTML5 - and they are bundled as app installation packages.

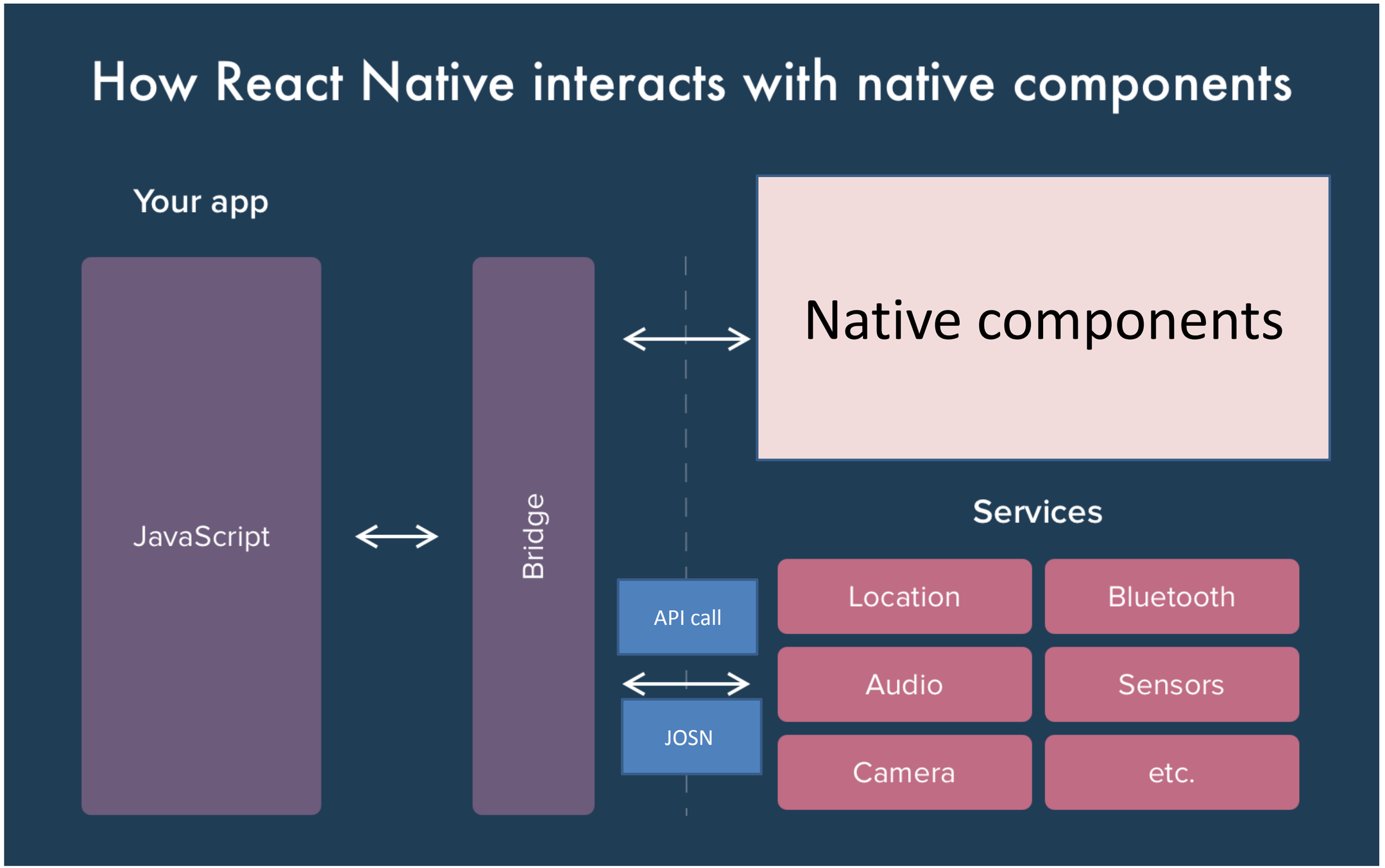
Native App – Interaction with Mobile Device



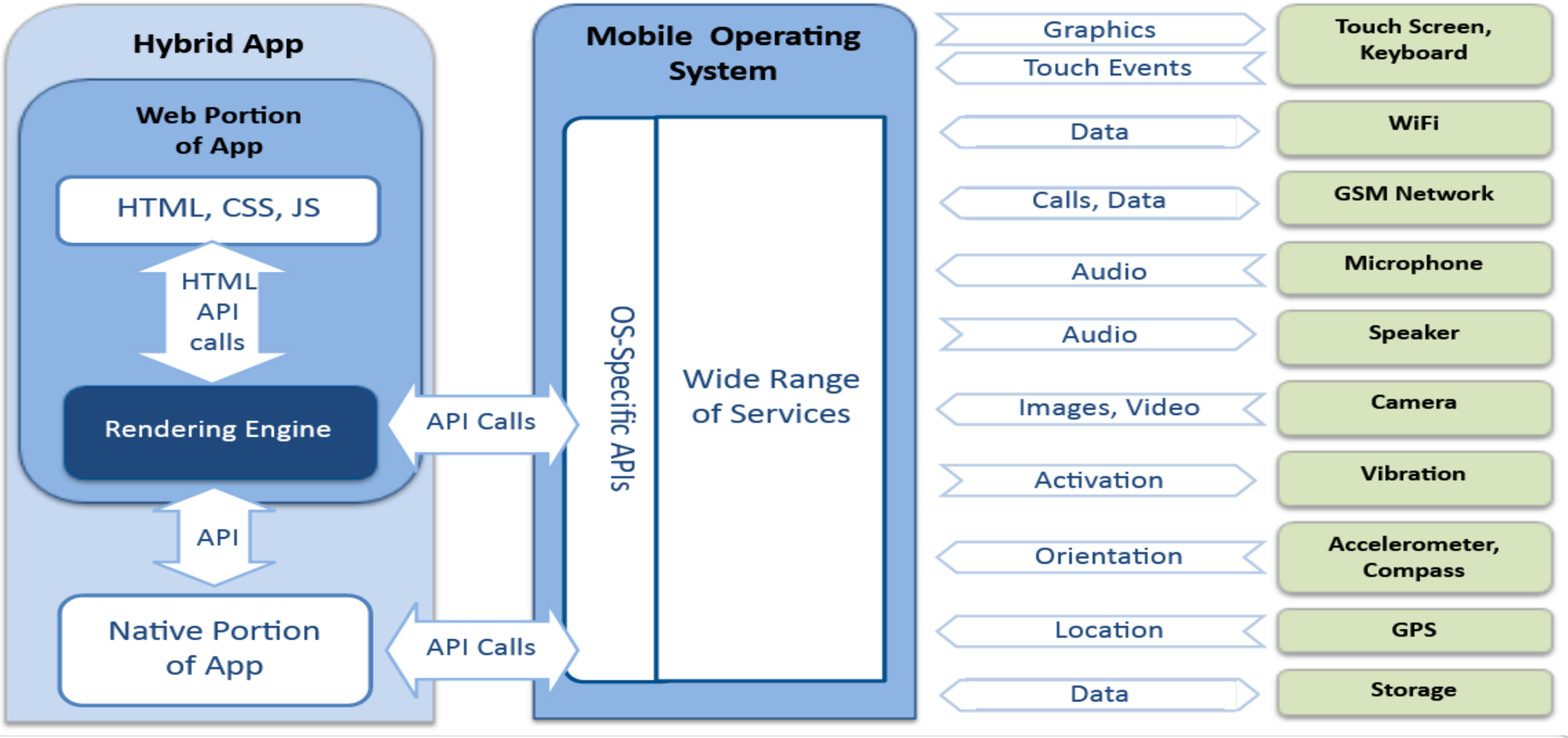
Cross platform applications



How React Native interacts with native components



Hybrid App – Interaction with Mobile Device



Advantages of Native Applications

- Native apps have exclusive operating system API's directly and get benifit of all and application frame work facilities
- High performance and better utilization for device components

Disadvantages of Native Applications

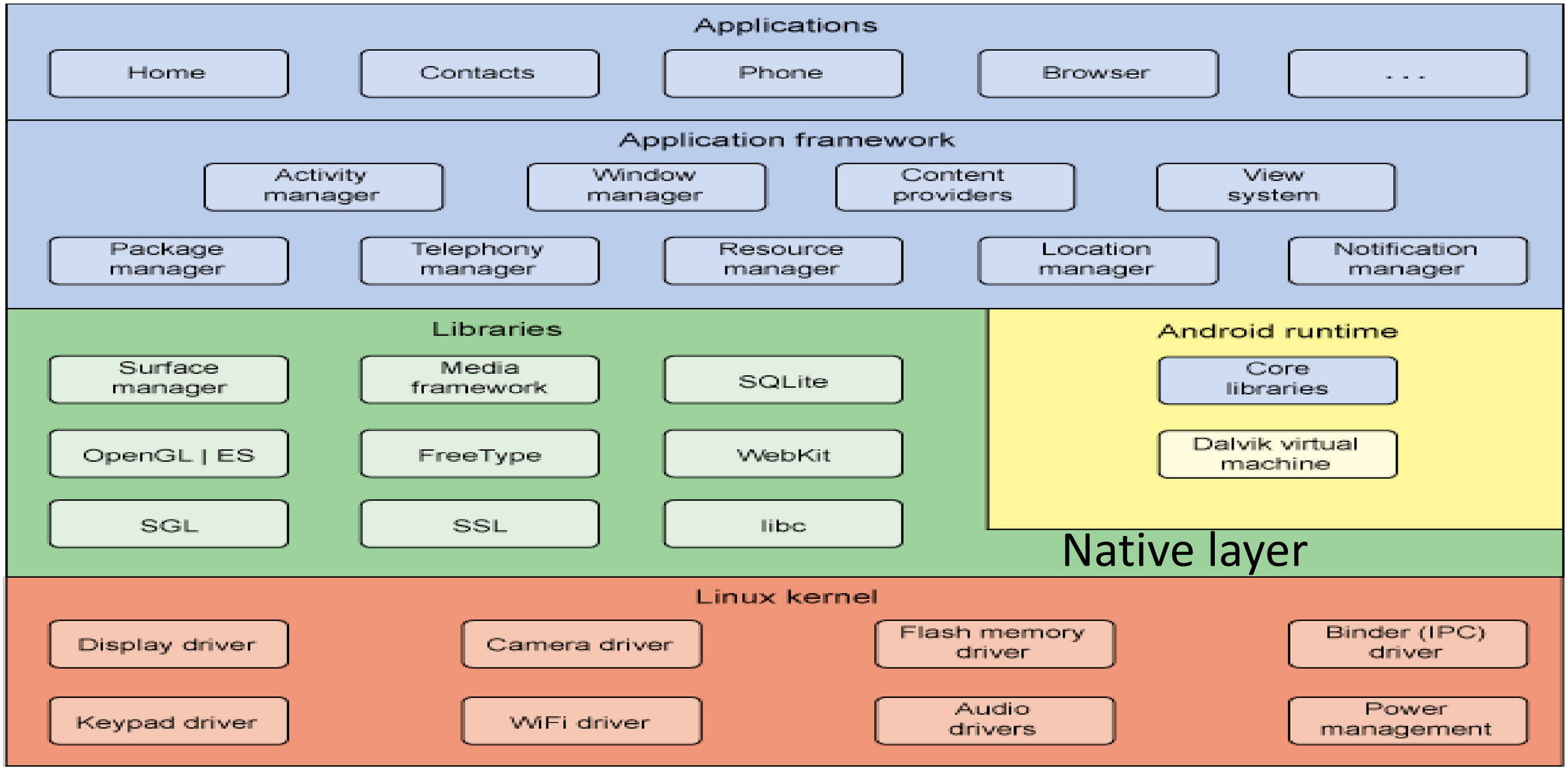
- Native apps can get expensive especially if you want the app to be available cross platform.

- It is very time consuming to create native applications for both Android and iOS.

Cross-Platform Applications

- **Advantages:**
- It is easy to **use code repeatability**, hence offering cost-effective solutions to app development.
- **Disadvantages:**
- Cross-platform applications shows lower performance as each function performs a call to a native function increasing execution time.

Android Architecture



Linux Kernel

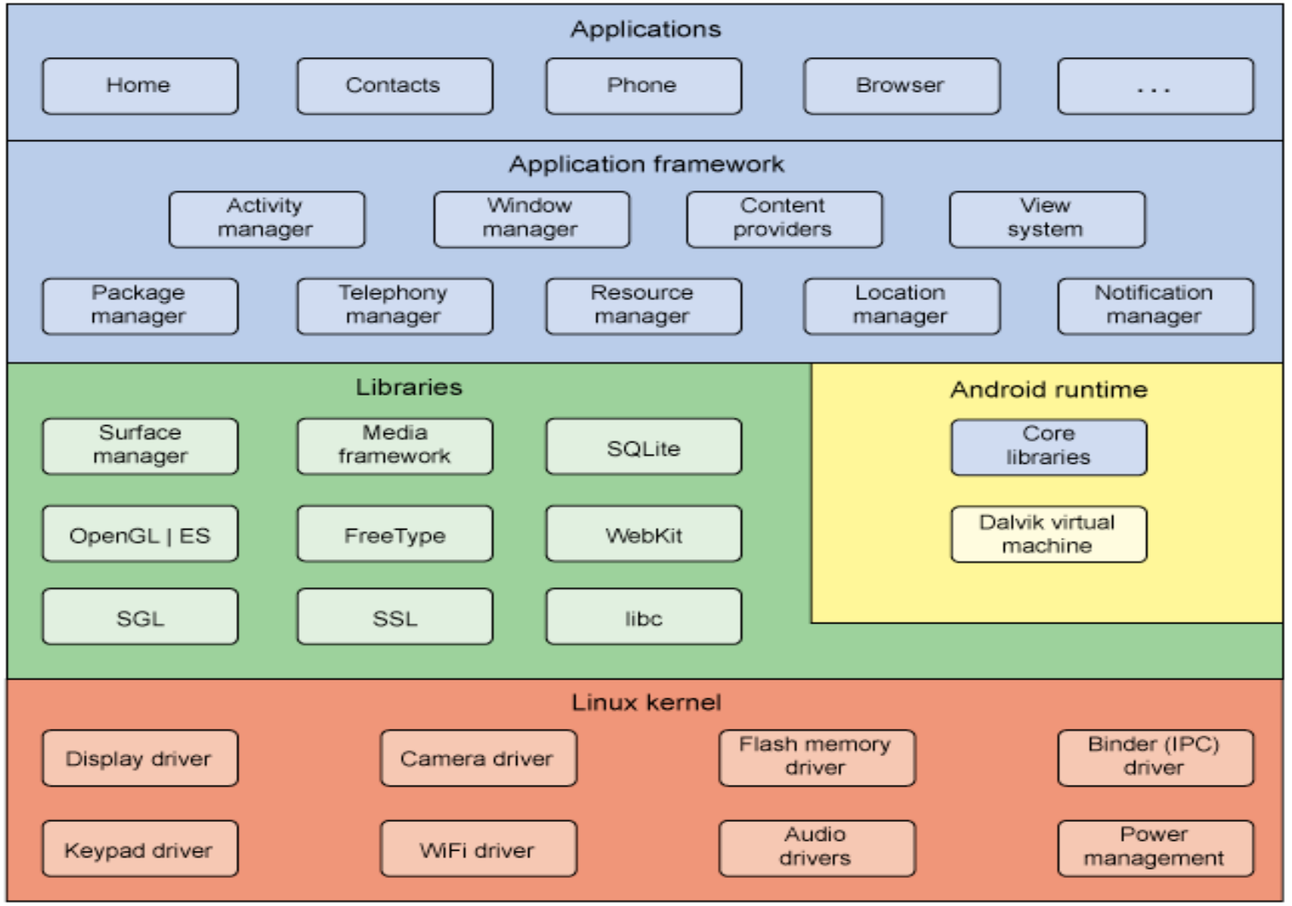
- Android was created on the open source kernel of Linux that provides

- **Security:**

- **Memory Management:**

- **Network Stack:**

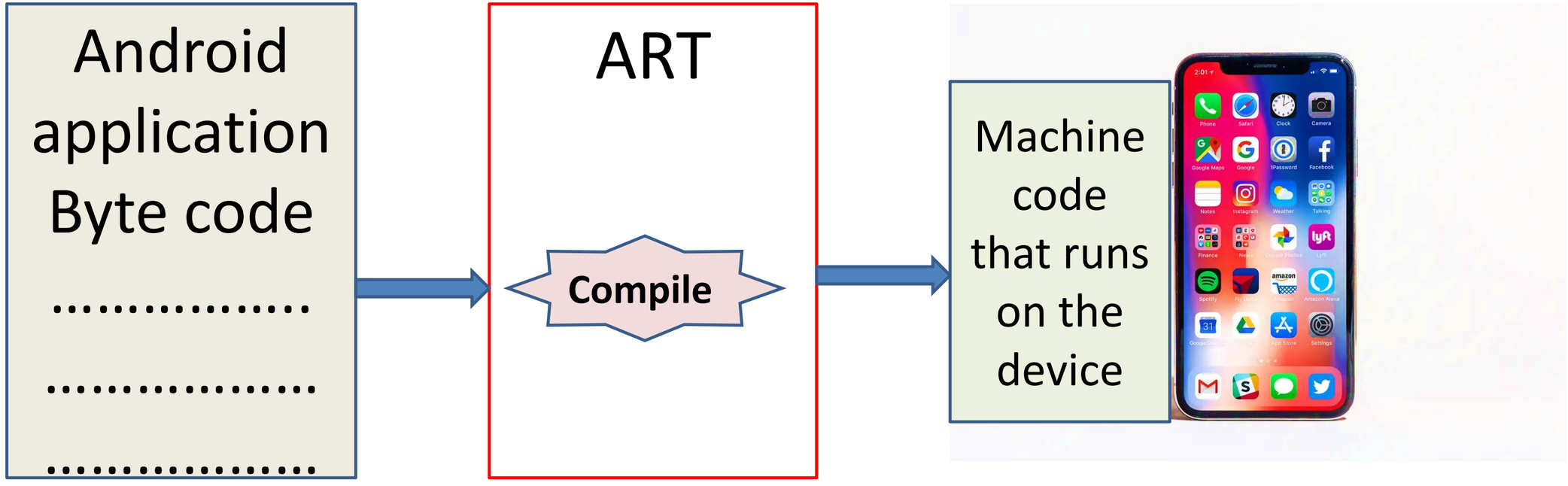
- **Driver Model:**



Linux Kernel

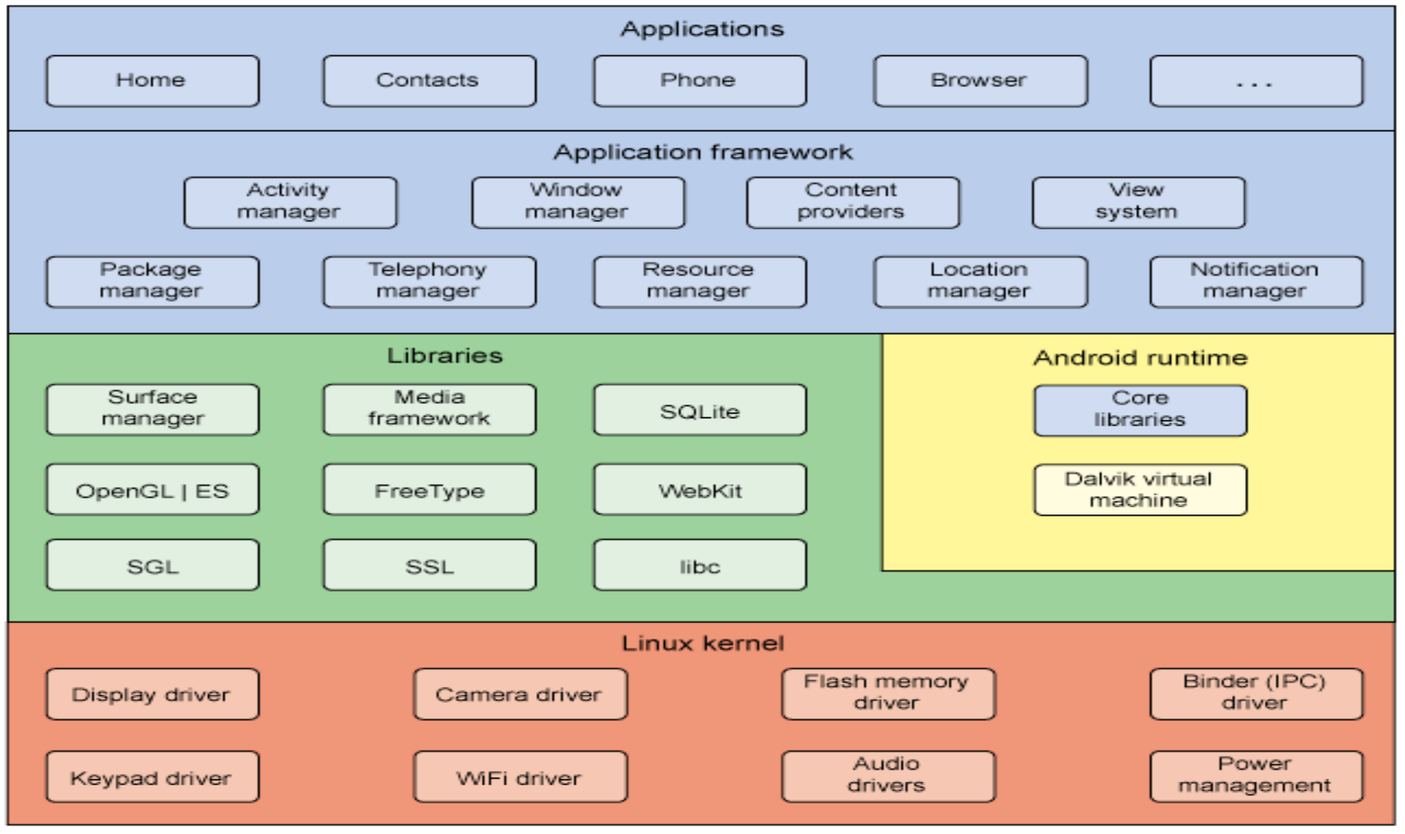
Second layer Libraries and Android runtime layer

- **The Android runtime:**
- The Android runtime (ART) is the default runtime for devices running Android 5.0 and higher.
- This runtime offers a number of features that improve performance and use smaller size of memory.
- ART accepts out put of Android applications as input and generates a compiled executable code for the target device



Second layer Libraries and Android runtime layer

- **Libraries:**
- Android was developed with various features. It consists of various C/C++ core libraries with numerous of open source tools.
- **Open GL(graphics library):**
  - **2Dand 3D graphics**
- **WebKit: (browser engine):**
  - **display web pages**
- **Media frameworks:**
  - **play and record audio and video.**

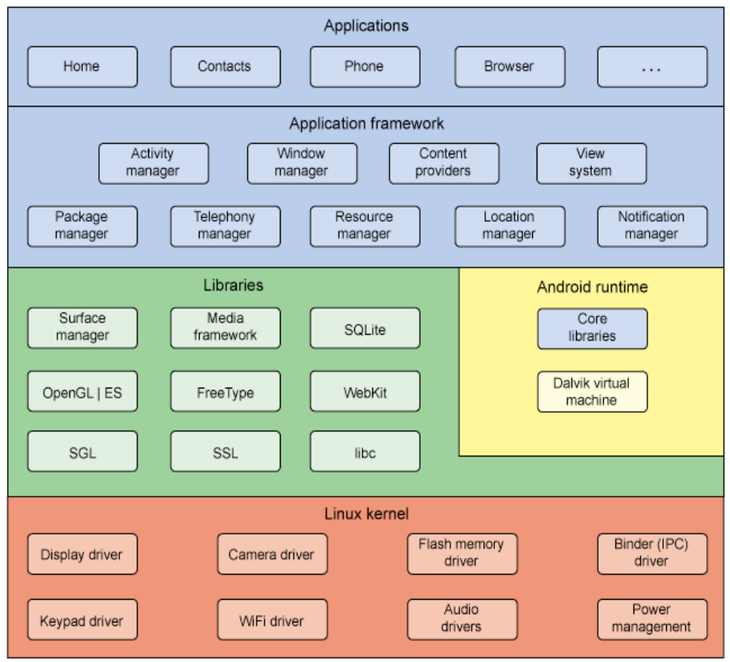


Application Framework Layer

- The Android contains a set of libraries.
- Application framework layer provide developers with different interfaces for all the various libraries.
- Developers don’t have to build any of the functionality provided by the android but just use the predefined classes and interfaces

- **Activity Manager:**
- Manage the main component of Android programs (Activities).
- **Telephony Manager:**
- It provides access to telephony services.
- **View System:**
- It builds the user interface by handling the views and layouts.
- **Location manager:**
- It finds the device’s geographic location.

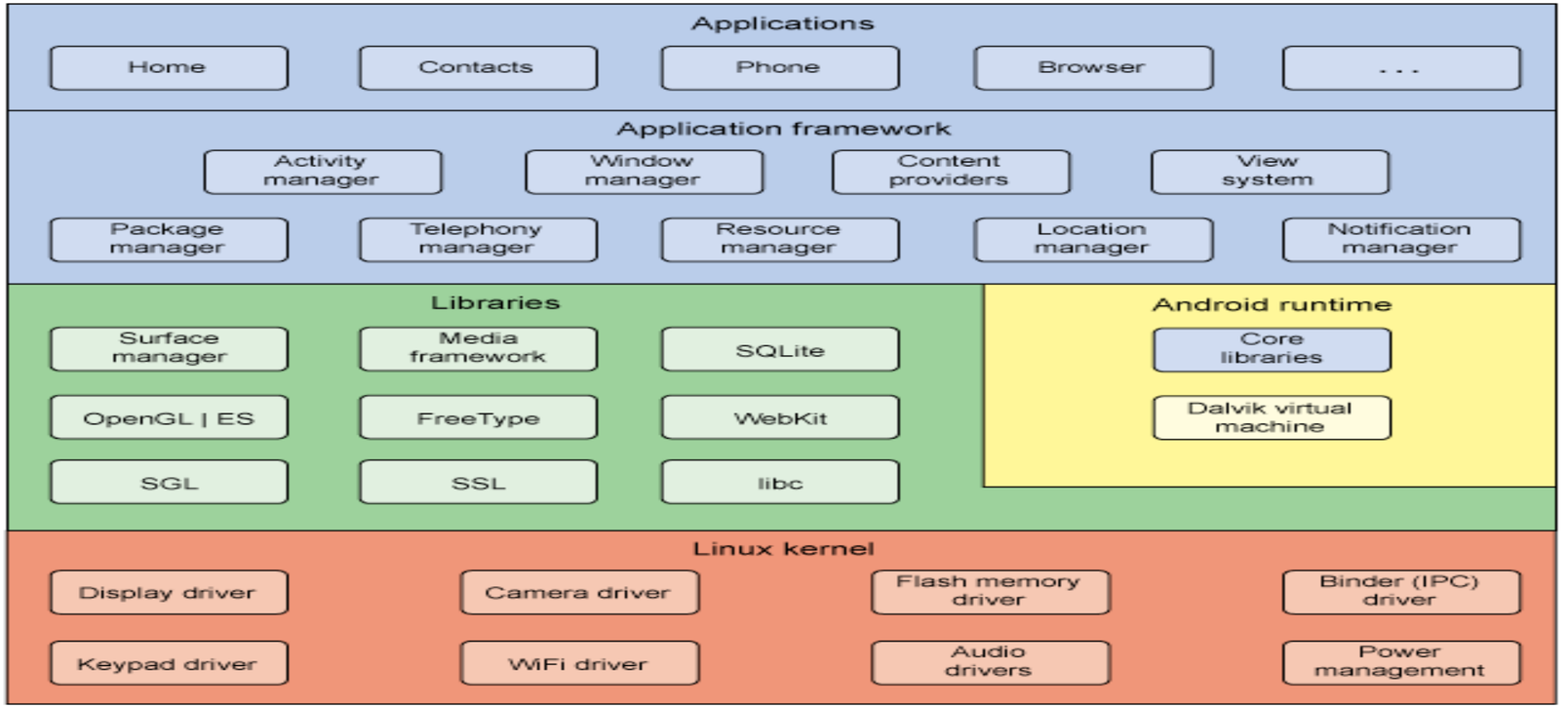
Application Framework Layer



Applications Layer

- Android applications can be found at the topmost layer.
- At application layer we write our application to be installed on this layer only.
- You don’t write code from scratch (pure Java) but you use the defined classes and Interfaces provided by Application Framework Layer.

Applications Layer



Android Studio as an example of native application

Android Studio Components

- 1. Activities
- 2. Services.
- 3.Layout.
- 4. Broadcast

Structure of Android Studio project

app > manifests > AndroidManifest.xml

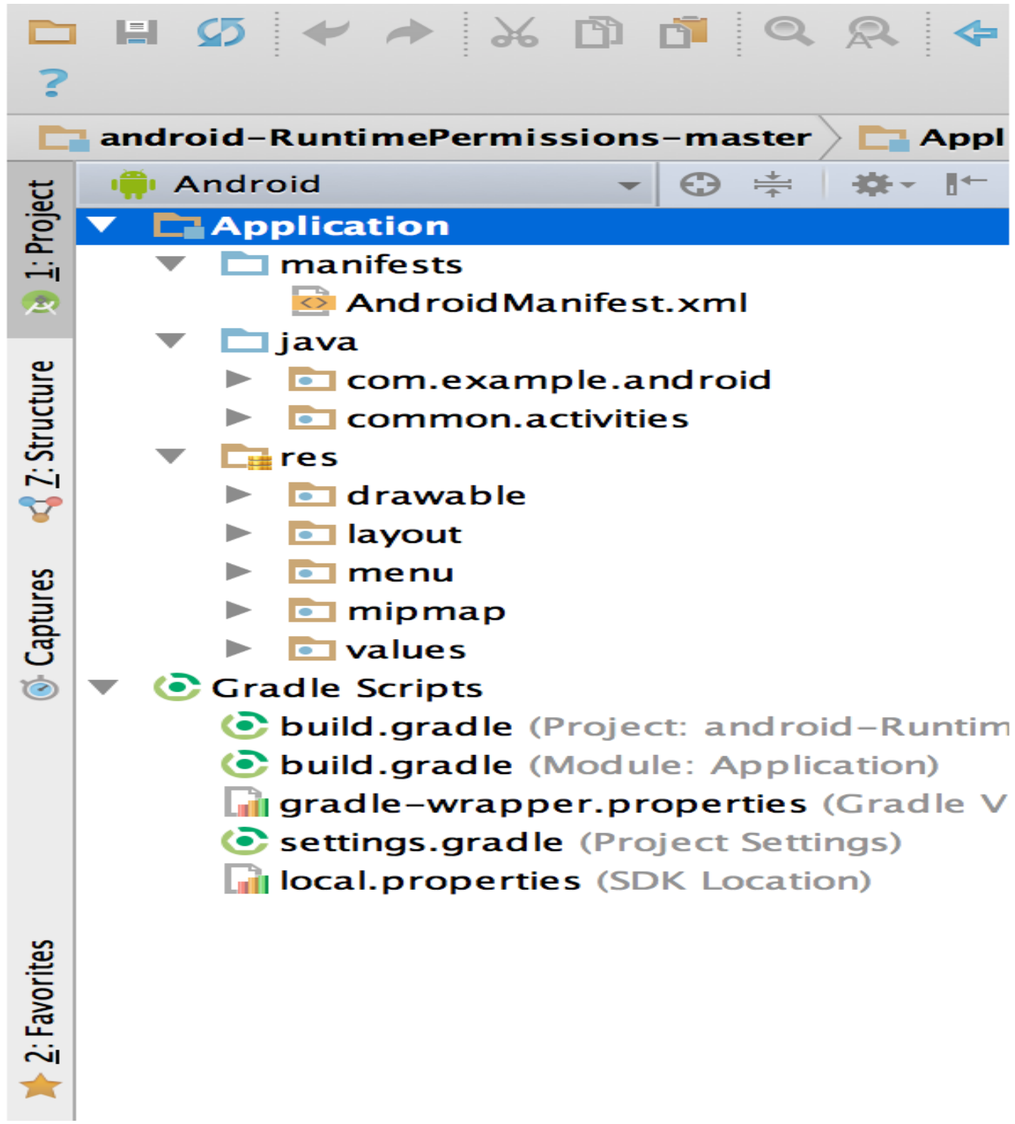
The manifest file describes the fundamental characteristics of the app and defines each of its components.

app > java > com.example.myfirstapp > MainActivity

This is the main activity. It's the entry point for your app. When you build and run your app, the system launches an instance of this Activity and loads its layout.

app > res > layout > activity_main.xml

This XML file defines the layout for the activity's user interface (UI). It contains a TextView element with the text "Hello, World!"



Structure of Android Studio project

- Android Studio displays your project files in the Android project view, organized by modules.
- All the build files are visible at the top level and each app module contains the following folders:

app > manifests > AndroidManifest.xml
The manifest file describes the fundamental characteristics of the app and defines each of its components.

app > java > com.example.myfirstapp > MainActivity This is the main activity. It's the entry point for your app. When you build and run your app, the system launches an instance of this Activity and loads its layout.

app > res > layout > activity_main.xml
This XML file defines the layout for the activity's user interface (UI). It contains a TextView element with the text "Hello, World!"

Project manifests

- The manifest file is a XML that describes essential information about the app to the Android build tools, the Android operating system, and Google Play.

1. The components of the app, which include all activities and services.

Each component must define basic properties such as the name of its Kotlin or Java class.

2. Permissions that the app needs in order to access protected parts of the system or other apps.

3. Device compatibility: The hardware and software features the app requires, which affects which devices can install the app from Google Play

Project Java directory

- This is the main activity.

- It's the entry point for your app.

- When you build and run your app, the system launches an instance of this Activity and loads its layout.

Project Java directory

- **package** com.example.empty1;

  import androidx.appcompat.app.AppCompatActivity;

  import android.os.Bundle;

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void  onCreate  (Bundle savedInstanceState) {
        super. onCreate (savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Activity in android studio

- The Activity class is a crucial component of an Android app, and the way activities are launched and put together is a fundamental part of the platform's application model.

- Unlike programming paradigms in which apps are launched with a main( ) method, the Android system initiates code in an Activity instance.

Activity in android studio

- The Activity class is designed to control the flow of mobile applications.

- When one app invokes another, the calling app invokes an activity in the other app, rather than the app as an atomic whole.

- In this way, the activity serves as the entry point for an app's interaction with the user.

- Each activity is a subclass of the Activity class.

Activity in android studio

- Generally, one activity implements one screen in an app.

- Most apps contain multiple screens, which means they comprise multiple activities.

- One activity in an app is specified as the *main activity*, which is the first screen to appear when the user launches the app.

- Each activity can then start another activity in order to perform different actions.

res layout in android studio

- XML file contains all the object that appears on the screen such as
- Textview, Button,ImageView,

```
<TextView
    android:layout_width ="244dp"
    android:layout_height ="91dp"
android:text =“Good Morning"
android:textSize ="30sp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintHorizontal_bias="0.758"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent" />
```

Inheritance: "is a" relationship

- When creating a class, rather than declaring all variables and methods, they can be inherited from an existing class.
- The existing class is called the superclass, and the new class is the subclass.
- A subclass inherits all methods and attributes from super class and adds its own fields and methods.

```
class A{
int i;
int j;
void m1(){}
}
```

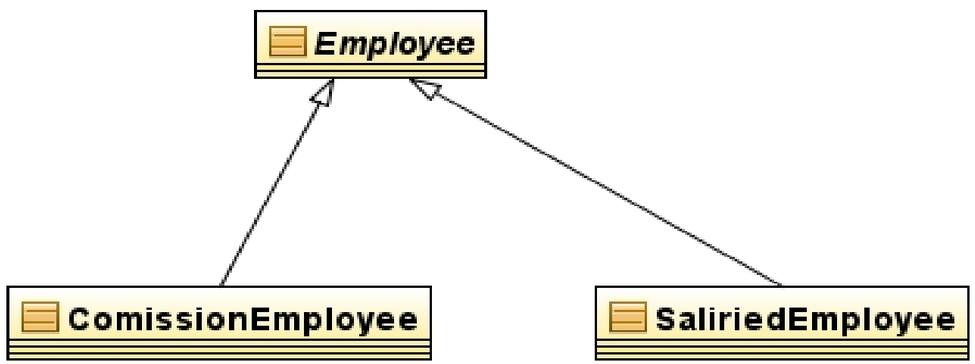
```
class B extends  A{
int k;
void m2(){}
}
```

```
class Test{
public static void main (String [] arg){
B b1=new B();
b1.i=10;
b1.m1();
}}
```

Inheritance examples.

Superclass	Subclasses
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	SaliredEmployee,CommisionEmployee
BankAccount	CheckingAccount, SavingsAccount

UML Employee Hierarchy Diagram



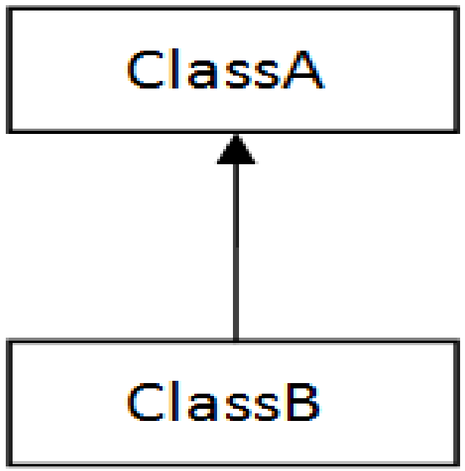

```
class Employee
{
}
```

```
class CommisionEmployee extends employee
{……….
}
```

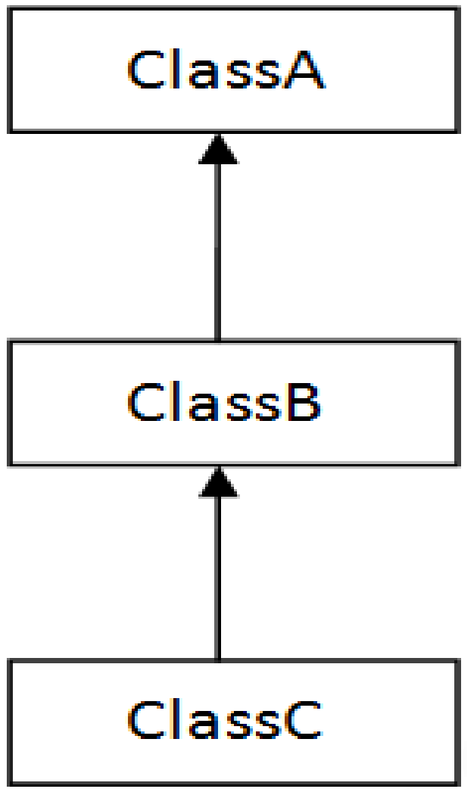
```
class SaliredEmployee extends employee
{……….
}
```

UML defines inheritance as a generalization relation

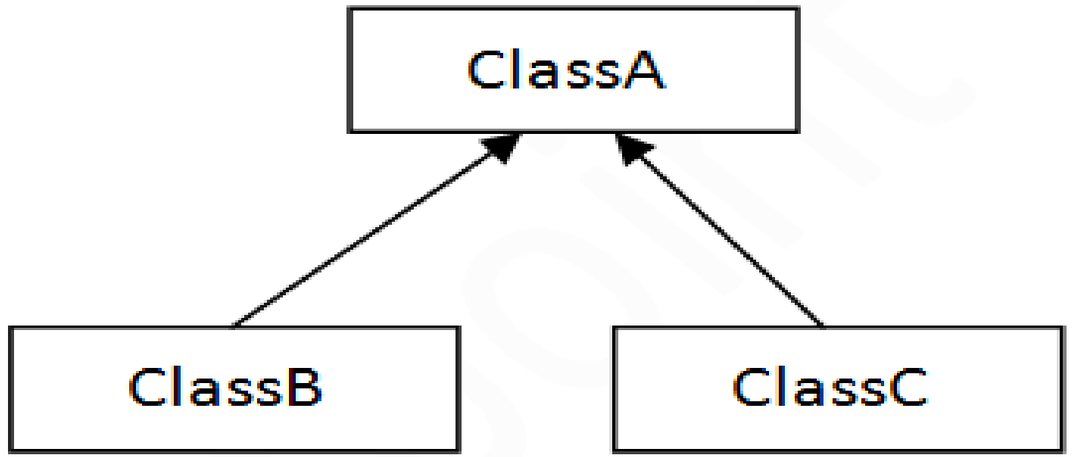
Types of inheritance



1) Single



2) Multilevel



3) Hierarchical

Interface in Object-Oriented

- An *interface is a classlike that contains only constants and abstract methods and no constructors.*

- A class use keyword “implements” to implement all methods in the interface.

➢A class can implements more than one interface.

Java Interface

```
public interface Coloring
{
double getcolor( );
 }
```

```
class Shape implements Coloring{

public double getcolor( )
{
// code to implement method
}
}
```

Java overriding and using super

```
public static void main(String[] args) {
      A a = new A ( ) ;
      B b = new B ( ) ;
      System.out.println (a.x +" "  + a.getx(  )) ;
      System.out.println (b.x +" " + b.getx(  )) ;
   }
class A{
   int x =5 ;
   int getx ( ){
      return x*2;
   }
}
class B extends A {
   int x=10;
   int getx ( ){
      return x*2;
   }
}
```

```
5,10
10,20
```

```
public static void main(String[] args) {
      A a = new A ( ) ;
      B b = new B ( ) ;
      System.out.println (a.x +" "  + a.getx(  )) ;
      System.out.println (b.x +" " + b.getx(  )) ;
   }
class A{
   int x =5 ;
   int getx ( ){
      return x*2;
   }
}
class B extends A {
   int x=10;
   int getx ( ){
      return super.x  * 2;
   }
}
```

```
5,10
10,10
```