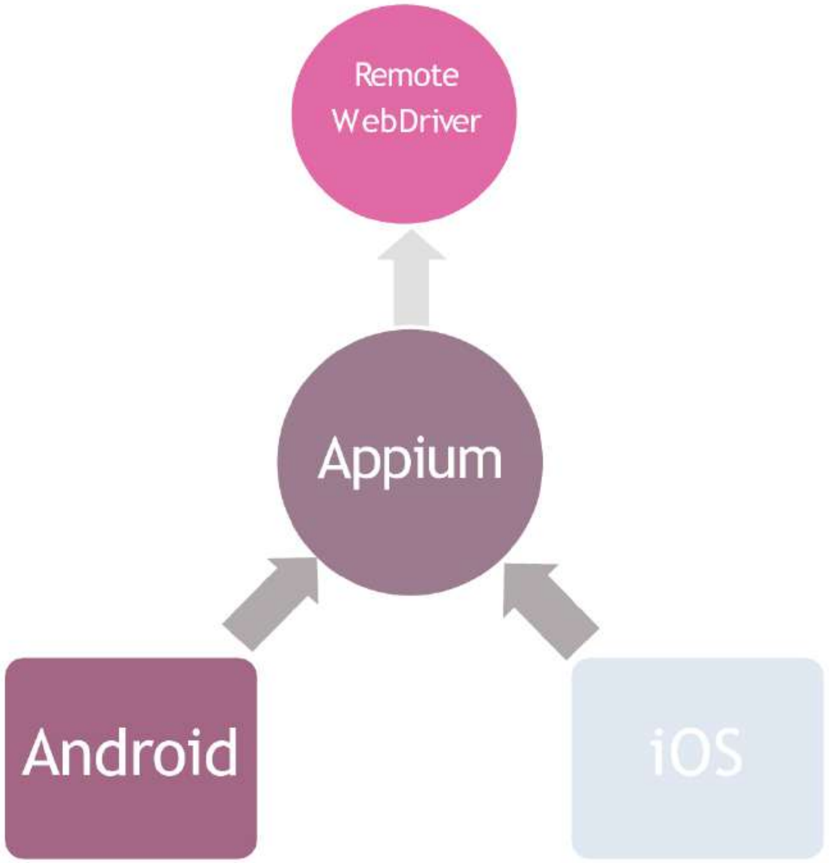
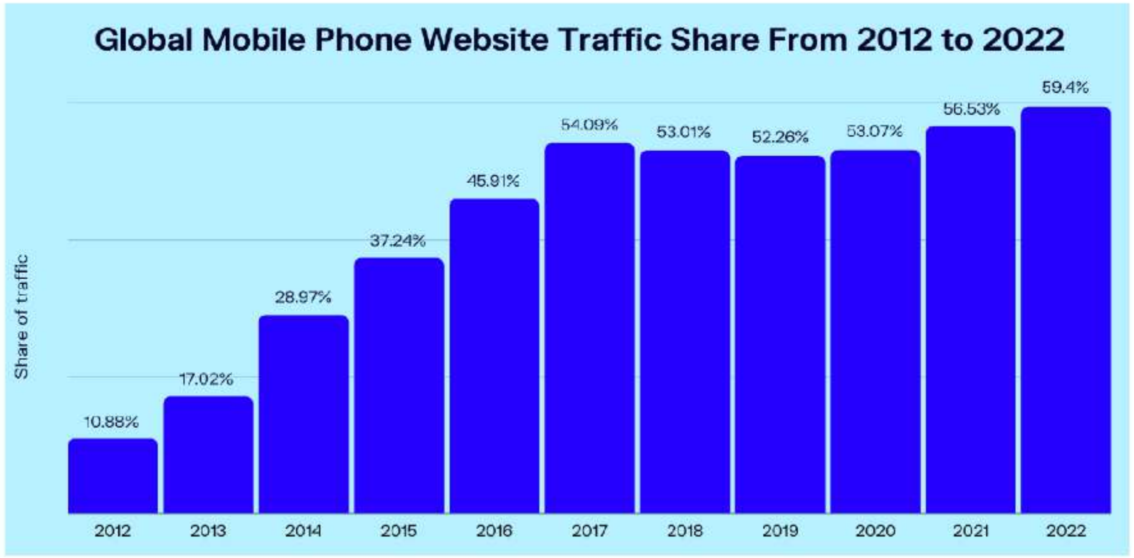
# MOBILE APP AUTOMATION USING APPIUM

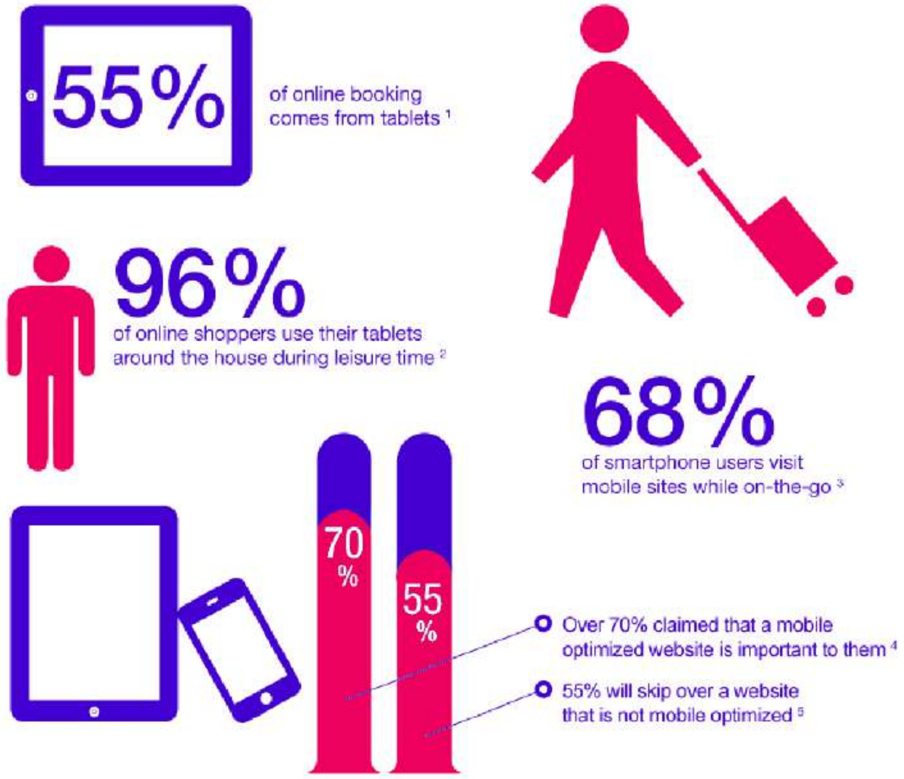
LOKESH KUMAR T

## PRE-REQUISITES FOR LEARNING APPIUM

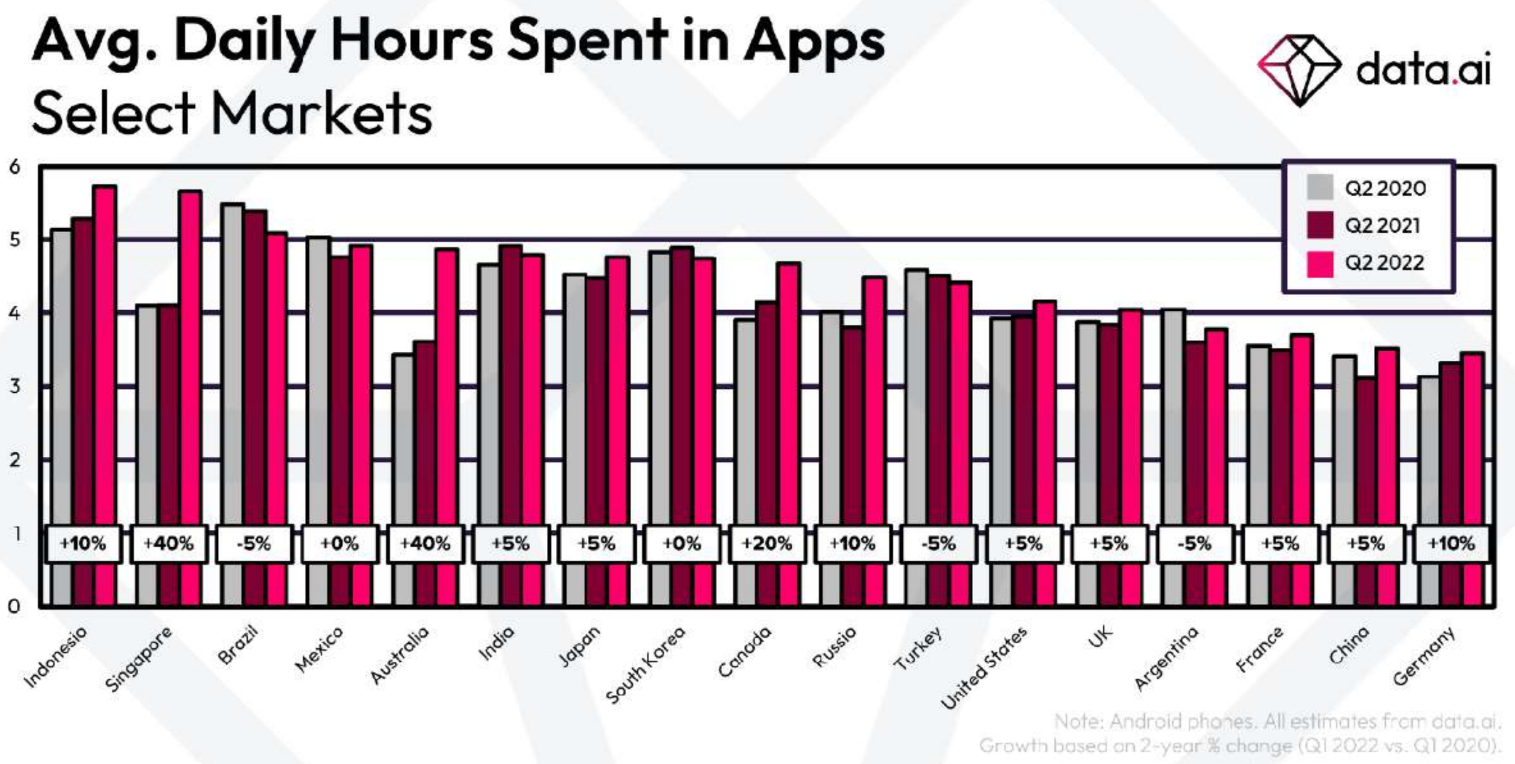


## WHY SHOULD I KNOW MOBILE AUTOMATION?

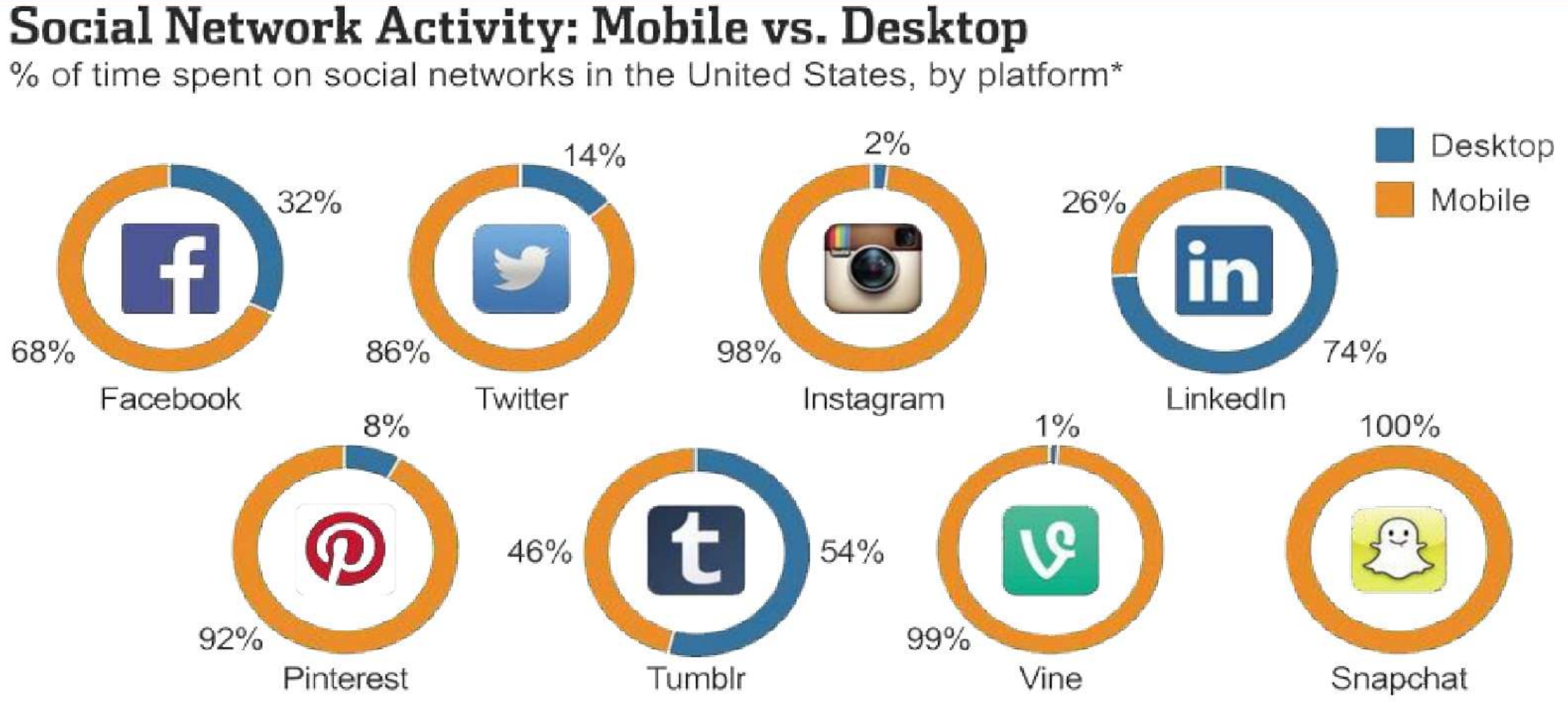



55% of online booking comes from tablets 1
96% of online shoppers use their tablets around the house during leisure time 2
68% of smartphone users visit mobile sites while on-the-go 3
70 %
55 %
Over 70% claimed that a mobile optimized website is important to them 4
55% will skip over a website that is not mobile optimized 5

## WHY SHOULD I KNOW MOBILE AUTOMATION?



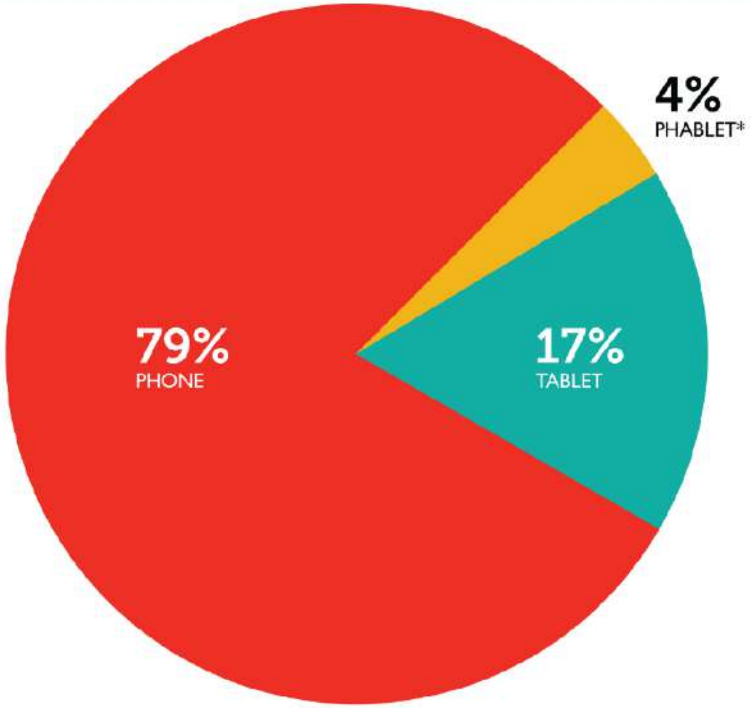
## SOCIAL NETWORKACTIVITY SURVEY



## KEY CHALLENGES

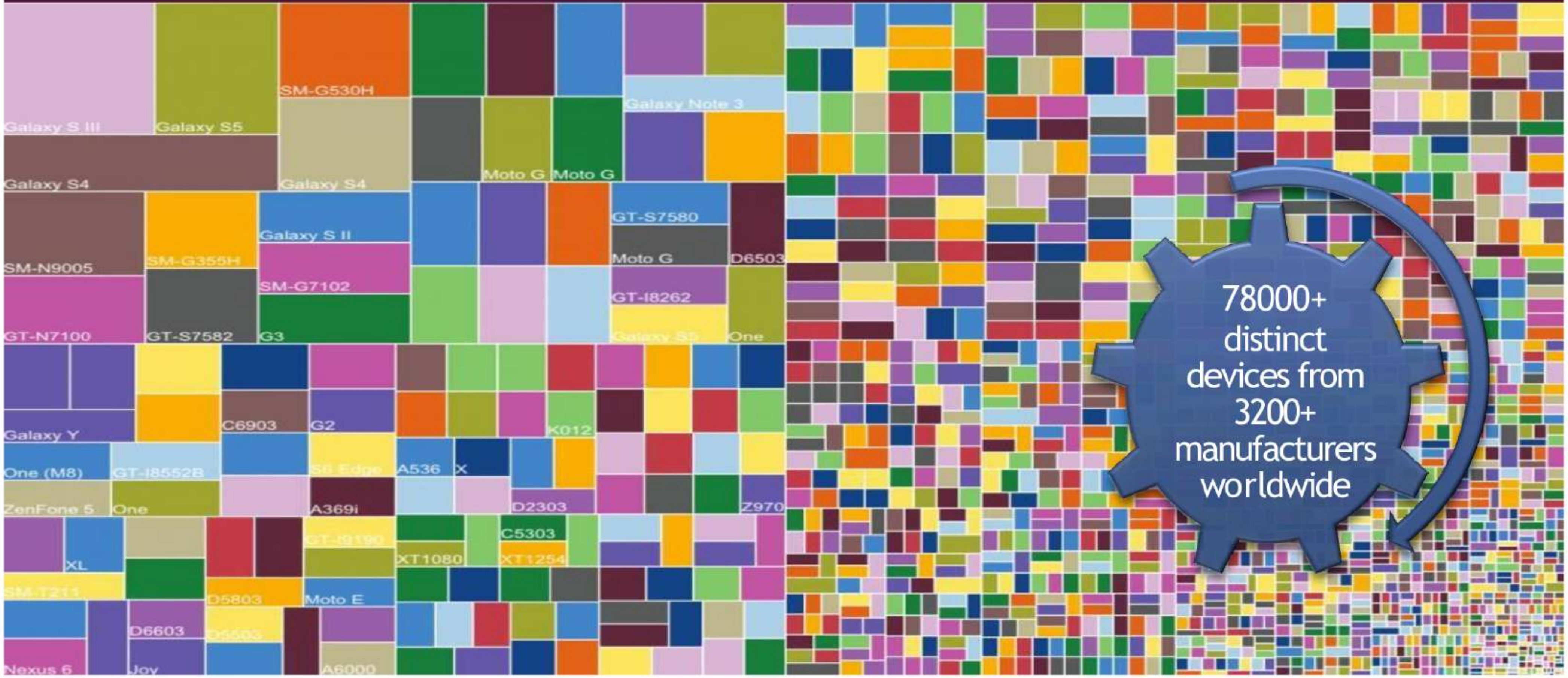
- The number of models and types of smartphones on the market today is already large, with more always entering the market.
- Each device tends to have differences, and deciding which devices to test can be a challenge.
- While everyone wants to support as many devices as possible, adding devices can greatly increase testing time as well as cost.
- In addition to the ever-increasing number of devices on the market, there are also a number of OS platforms, and many companies are developing for an increasing number of them.
- While continuous integration has become standard for most web development projects, the same becomes for mobile development as well.

## DEVICES – MARKET DISTRIBUTION



*Devices with a screen size > 5in but < 7in
(primarily driven by the Samsung Galaxy Note and Galaxy Note 2, which comprise more than 99% of the category)

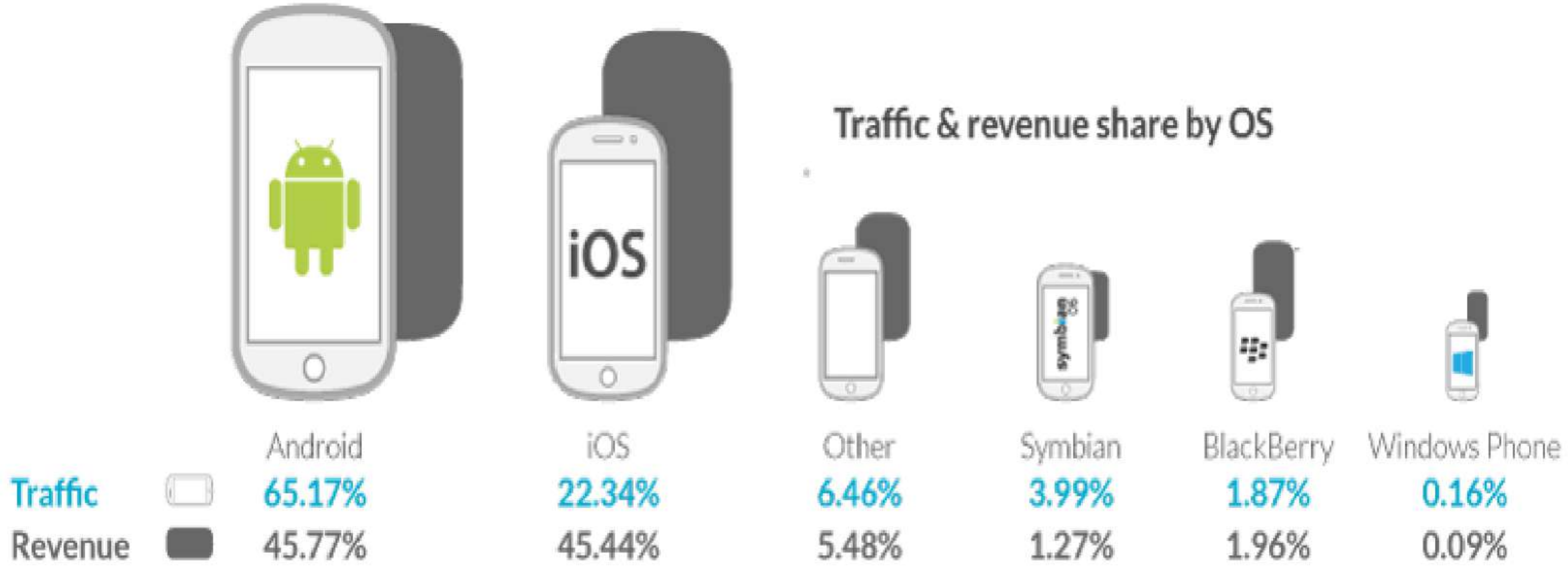
## ANDROID DEVICE FRAGMENTATION



## REAL DEVICES VS. EMULATORS

| # | Parameter | Real Device | Emulator |
|---|---|---|---|
| 1 | Cost Effective | Very expensive considering the type of smart phones and range of testing | Very low or no cost (GenyMotion) |
| 2 | Chipset | Chipsets in particular can also provide vastly different user experiences between high and low end devices | Push notifications, geo-location, orientation, and other functionality are impossible to test adequately without a real device. |
| 3 | Battery | Each real devices have different battery specifications and discharge rates. | Battery cannot be emulated |
| 4 | Screen size, Resolution | Multitude of screen sizes & resolutions | Emulator may still not give the information on brightness, saturation etc. |
| 5 | Phone interruptions + Behaviour | Swipe, Pinch /Expand,Tap /Double Tap, Long Press | An emulator cannot adequately show how network related interruption events will react with your application |
| 6 | Parallel Execution | Not possible /Limited | Relatively easy to build and run |

## OS MARKET SHARE



Traffic & revenue share by OS

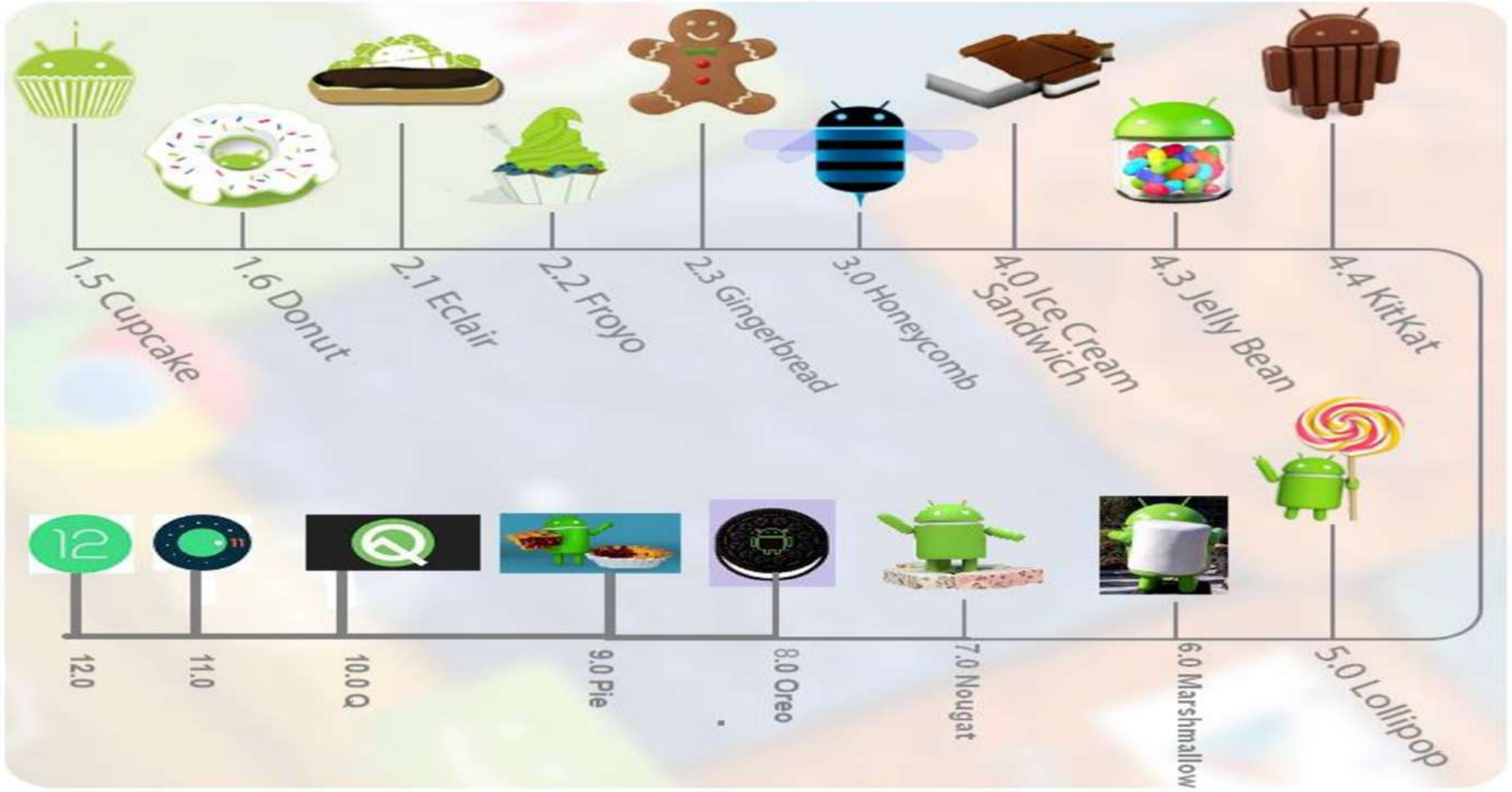
| | | Android | iOS | Other | Symbian | BlackBerry | Windows Phone |
|---|---|---|---|---|---|---|---|
| Traffic | ▭ | 65.17% | 22.34% | 6.46% | 3.99% | 1.87% | 0.16% |
| Revenue | ■ | 45.77% | 45.44% | 5.48% | 1.27% | 1.96% | 0.09% |

## MOBILE AUTOMATION – PLATFORM & APPLICATIONS

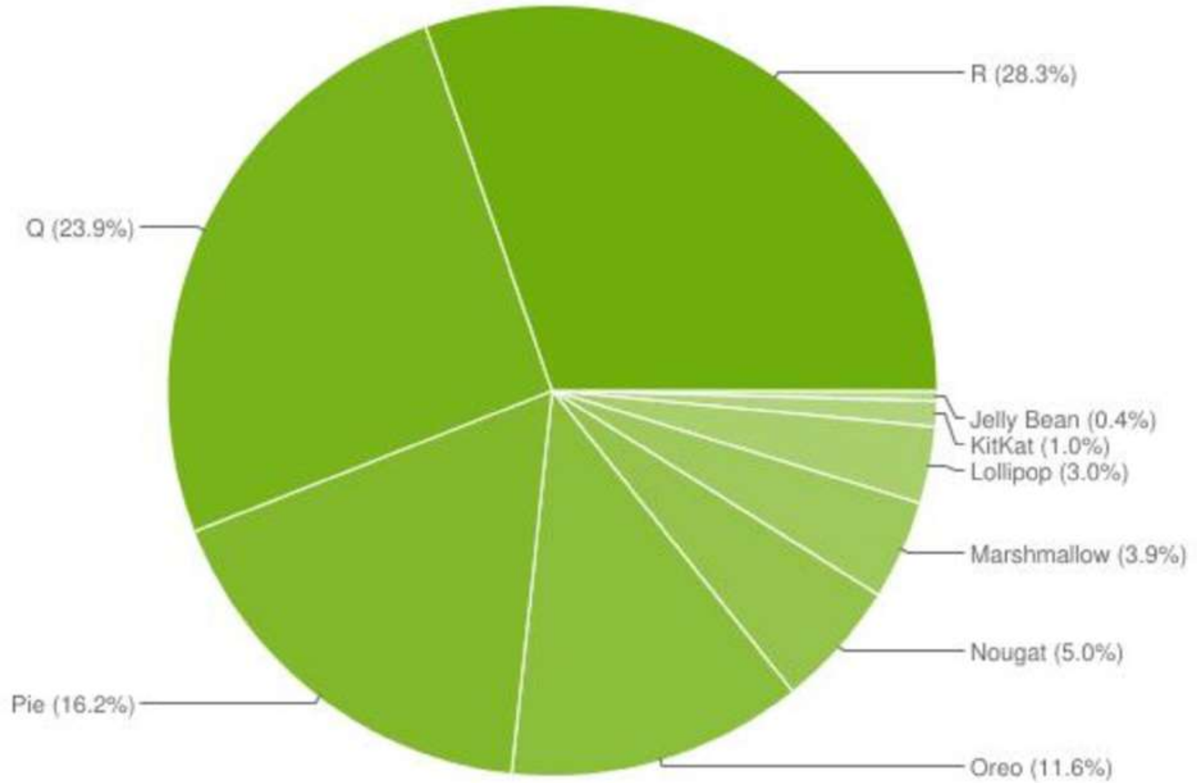
## BENEFITS OF MOBILE AUTOMATION

1. Saves your time
2. Saves your money
3. Higher test coverage
4. Faster time to market
5. Reusability of code
6. Improved accuracy
7. Eliminate Human error

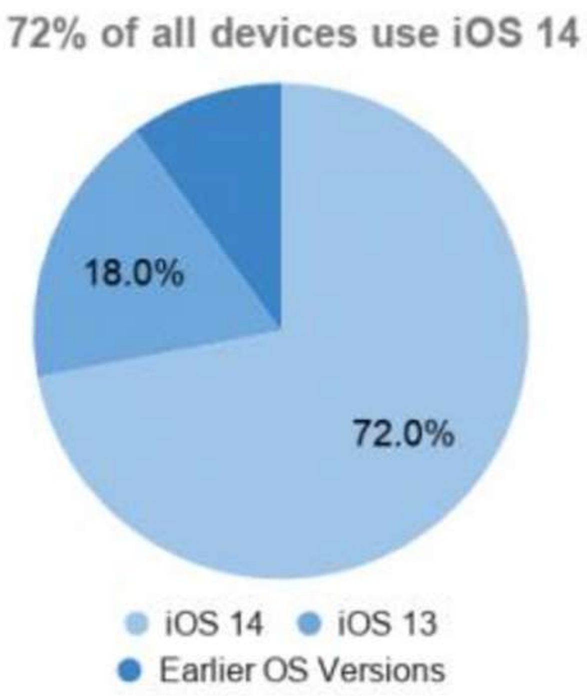
## ANDROID VERSIONS

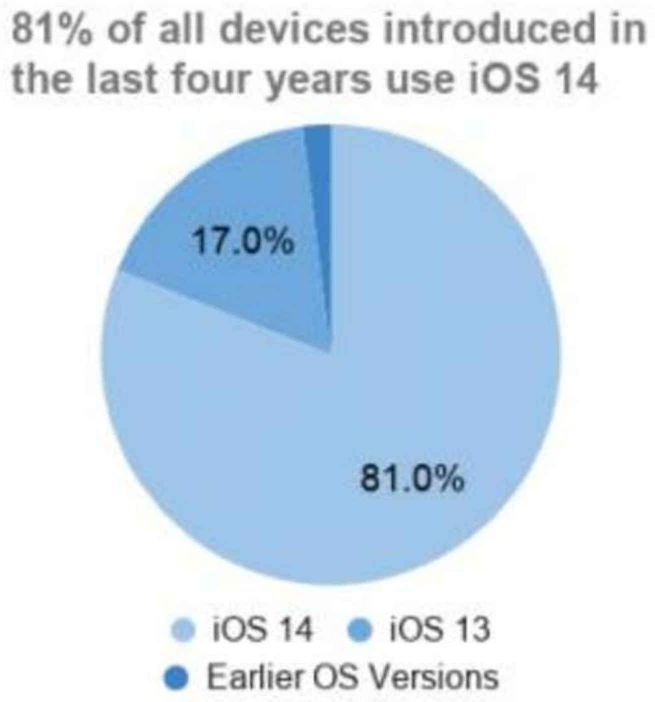


## ANDROID VERSIONS DISTRIBUTION

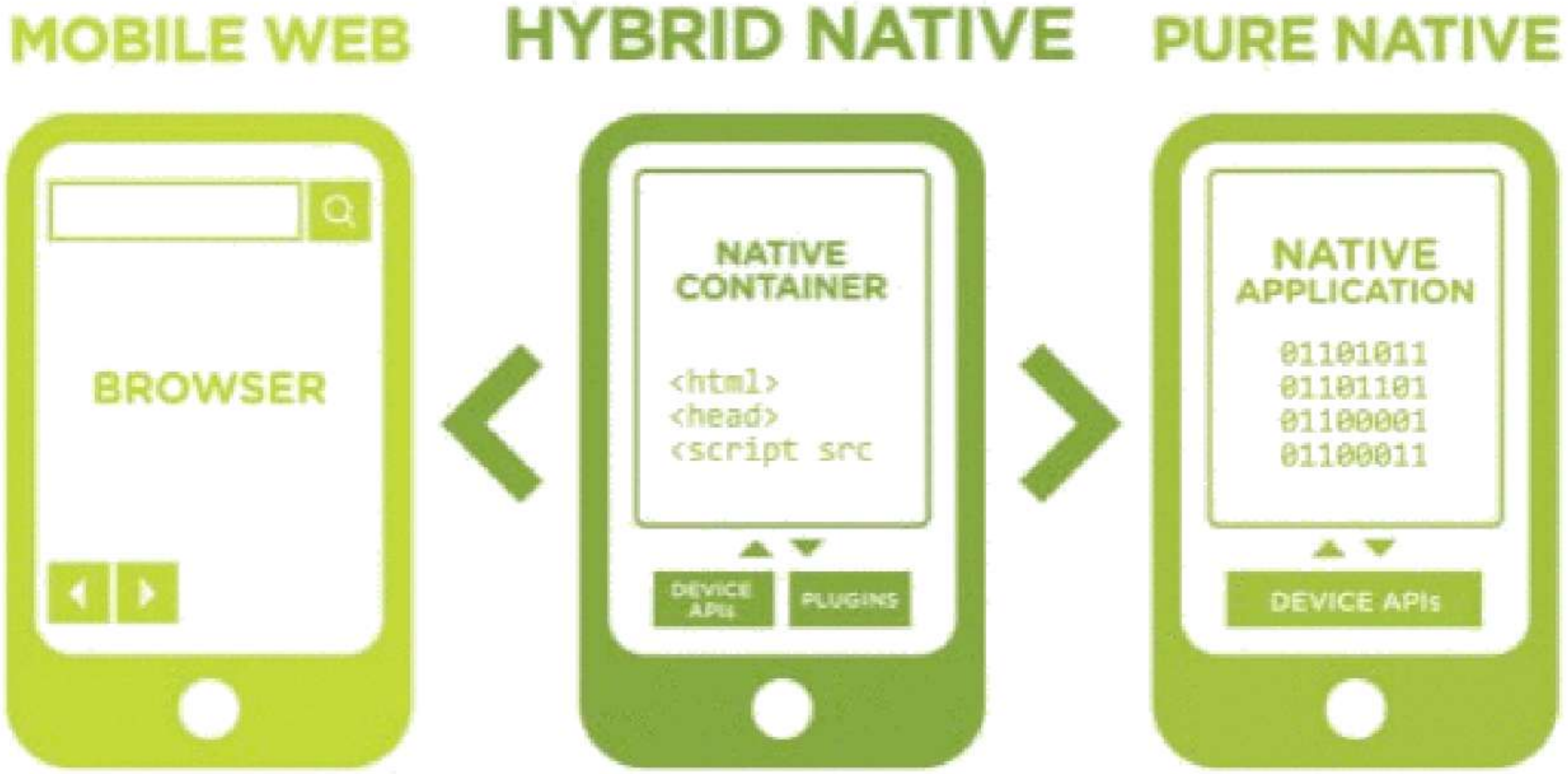


## IOSVERSIONS DISTRIBUTION





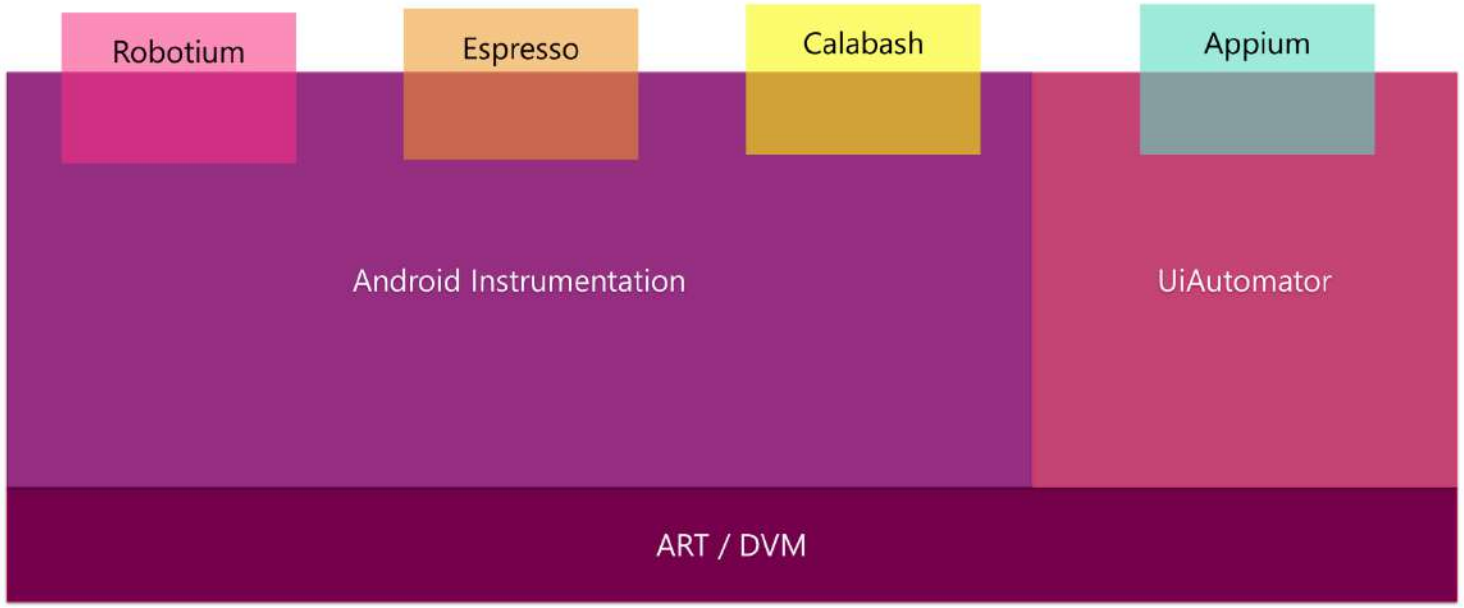
## NATIVE VS. HYBRID VS. MOBILE



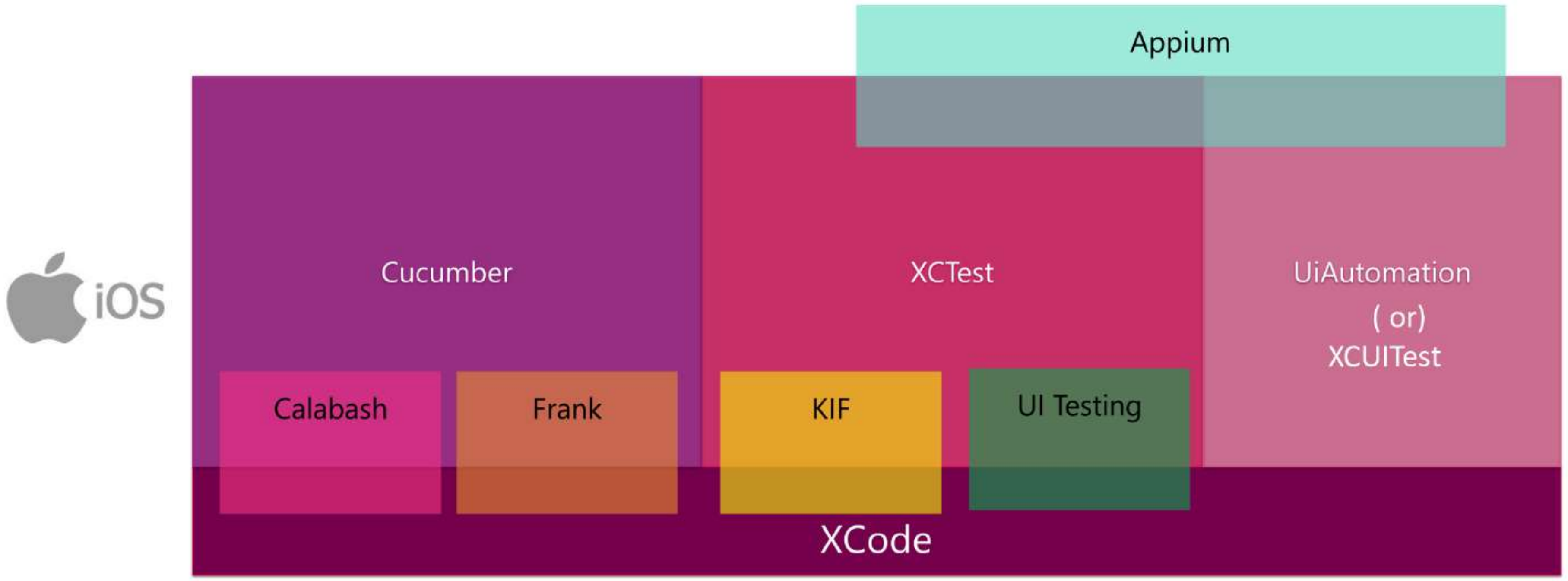
## TOOLS & FEATURES – A COMPARISON STUDY

| Tools | Rule 1 | Rule 2 | Rule 3 | Rule 4 |
|---|---|---|---|---|
| | The app, which is to be automated is of same code as in production /store | The tool scripting supported by any language and can be of any frameworks | The tool using a standard automation specification an a large and thriving open-source community effort d API. | The tool has a large and thriving open-source community effort for support. |
| Calabash-ios | ✗ | ✗ | ✗ | ✓ |
| Frank | ✗ | ✗ | ✗ | ✗ |
| UIAutomation | ✗ | ✗ | ✗ | ✗ |
| ios-driver | ✓ | ✓ | ✓ | ✗ |
| KeepItFunctional | ✓ | ✗ | ✗ | ✗ |
| Calabash-android | ✗ | ✗ | ✗ | ✗ |
| MonkeyTalk | ✗ | ✗ | ✗ | ✗ |
| Robotium | ✗ | ✗ | ✓ | ✗ |
| UiAutomator | ✗ | ✗ | ✗ | ✗ |
| Selendroid | ✗ | ✓ | ✓ | ✗ |
| Appium | ✓ | ✓ | ✓ | ✓ |

## FAMILY OF ANDROID TEST FRAMEWORKS



## FAMILY OF IOS TEST FRAMEWORKS



## WHAT ISAPPIUM?

Appium is an open source, cross-platform test automation tool for native, hybrid and mobile web apps, tested on simulators (iOS), emulators (Android), and real devices (iOS, Android). Appium opens up the possibility of true cross-platform native mobile automation.

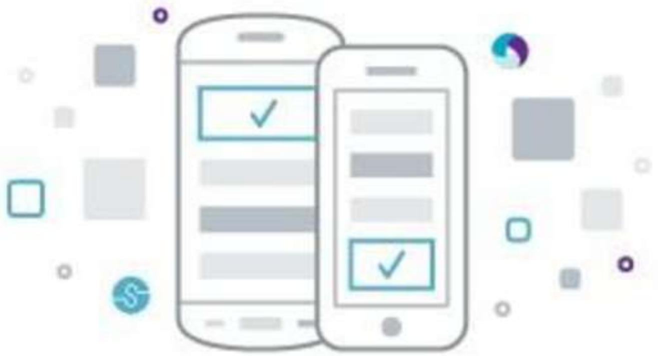
- You don't have to recompile your app or modify it in any way, due to use of standard automation APIs on all platforms.
- You can write tests with your favourite dev tools using any WebDriver-compatible language such as Java, Objective-C, JavaScript with Node.js (in promise, call-back or generator flavors), PHP, Python, Ruby, C#, Clojure, or Perl with the Selenium WebDriver API and language-specific client libraries.
- Investing in the WebDriver protocol means you are betting on a single, free and open protocol for testing that has become a defacto standard. Don't lock yourself into a proprietary stack.

appium

Lokesh Kumar T

appium
sponsored by Sauce Labs

In 2014, Sauce Labs released Appium 1.0 and ended the year with over **135 contributors** and **5,000 commits**.



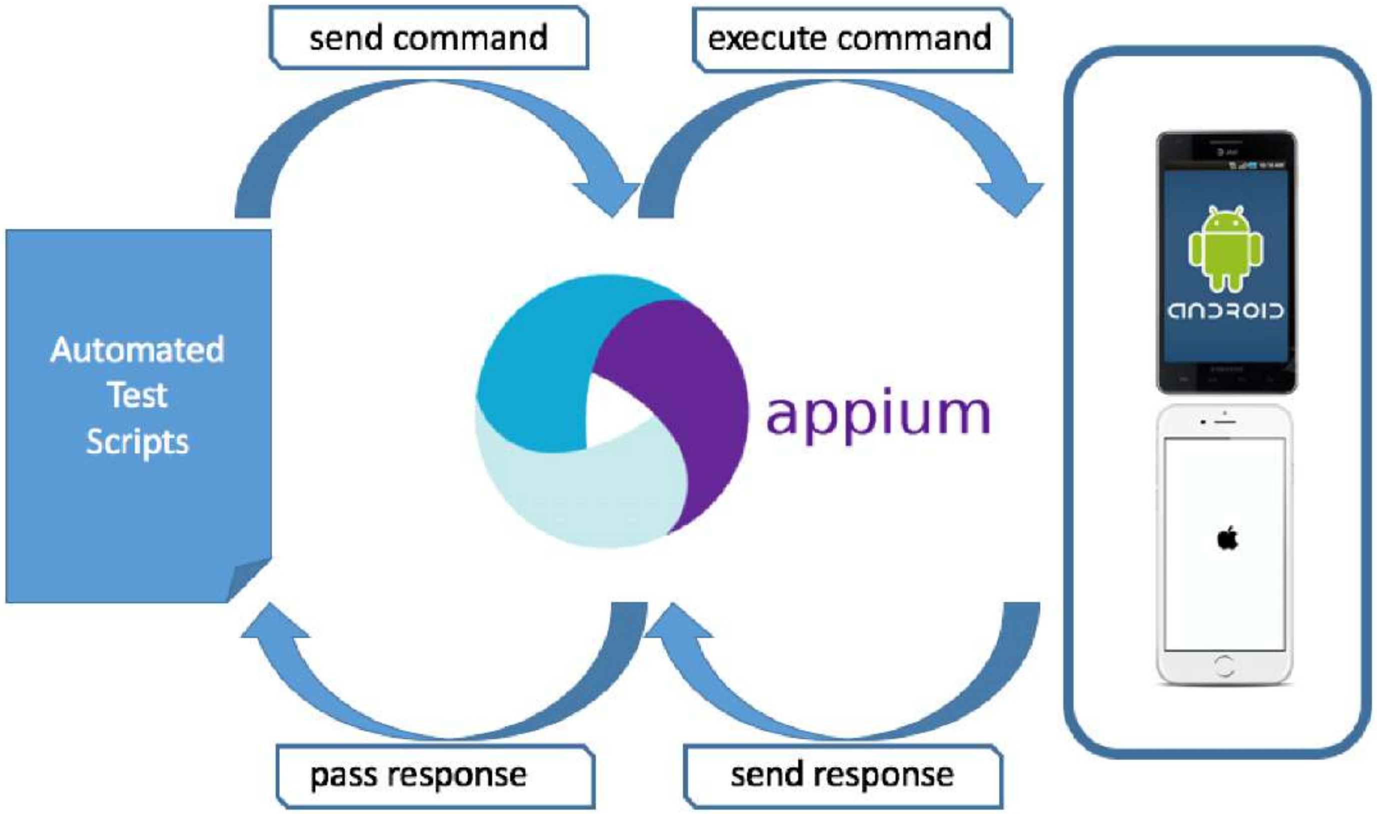
Native and hybrid app testing is trending up on Sauce!

In 2013-2014 versus 2014-2015, we saw an **increase of 1023% of mobile tests run** on our platform year over year.

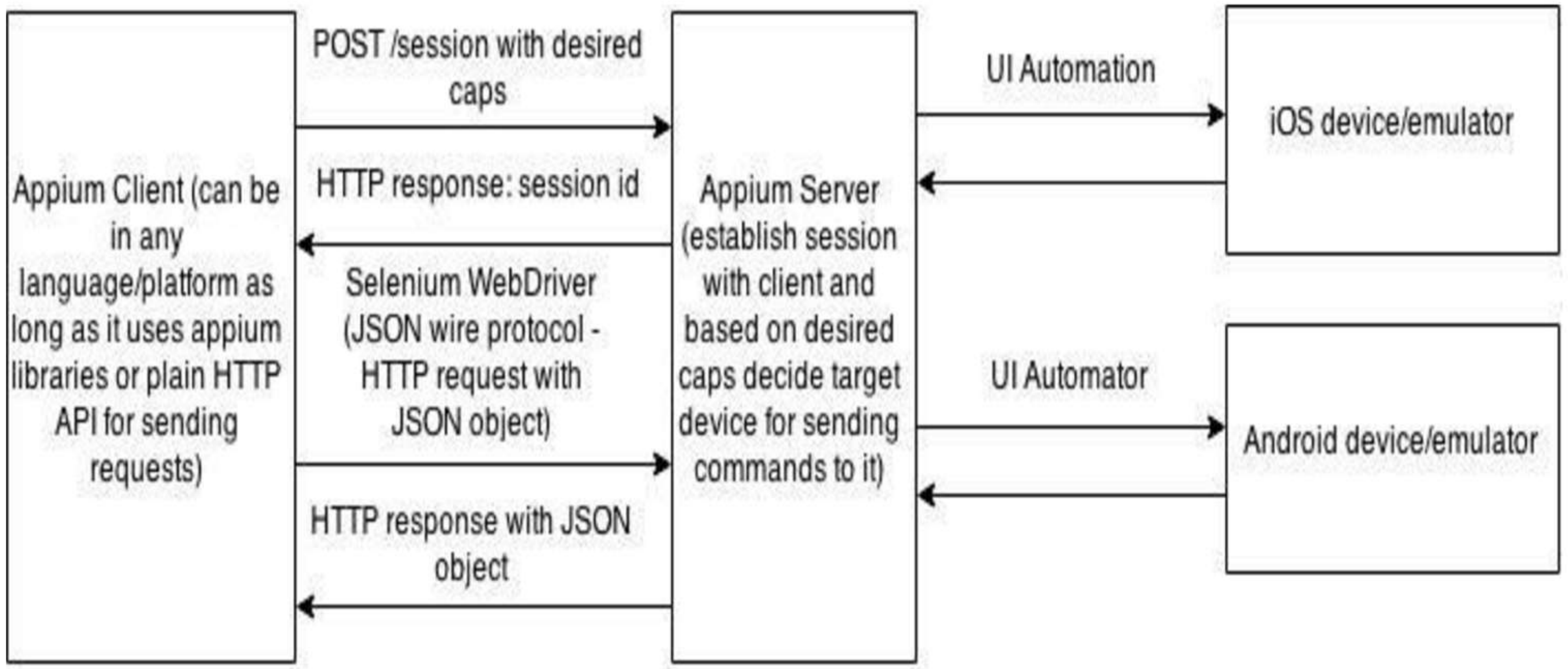
Android native + hybrid mobile tests run on Sauce **increased 993%** year over year (2013-2014 versus 2014-2015)

iOS tests increased a staggering **1760%**!

## HIGH LEVEL - WORKFLOW OFAPPIUM – Native App



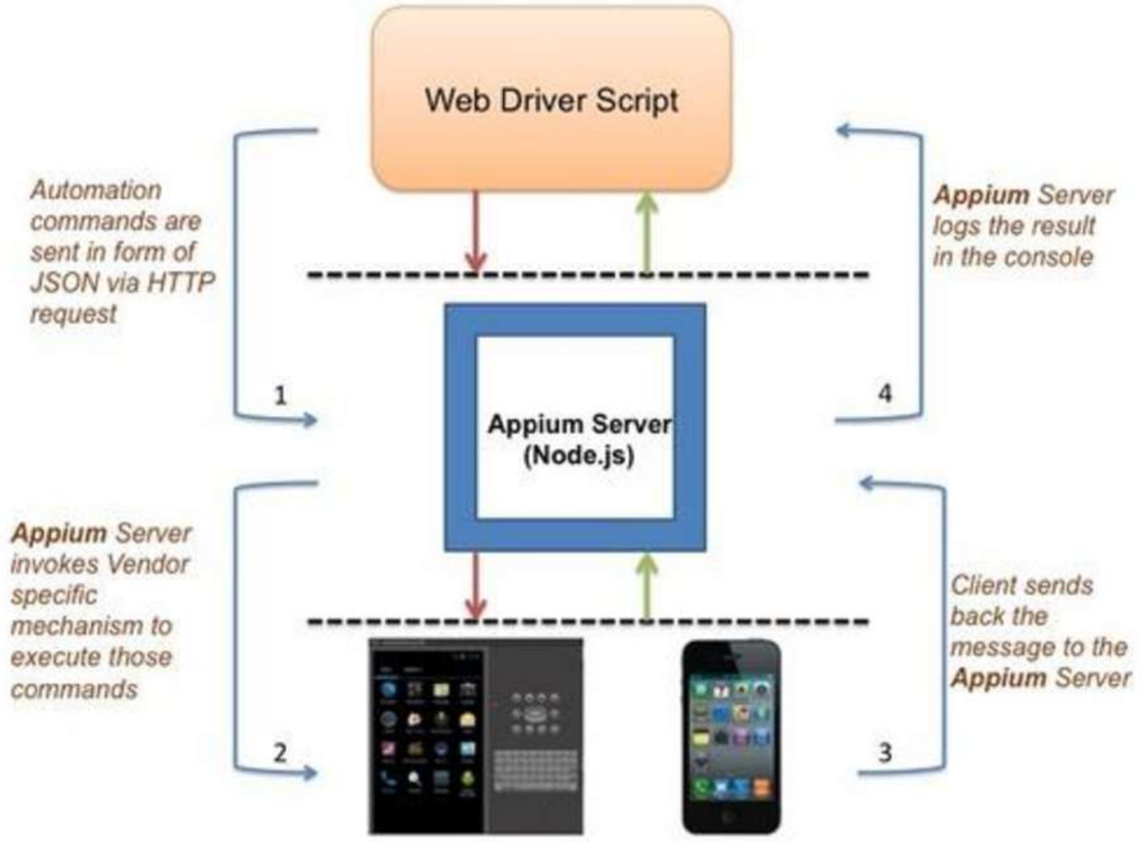
## WORKFLOW OFAPPIUM – Native App


Appium Client (can be in any language/platform as long as it uses appium libraries or plain HTTP API for sending requests)
POST /session with desired caps
HTTP response: session id
Selenium WebDriver (JSON wire protocol - HTTP request with JSON object)
HTTP response with JSON object
Appium Server (establish session with client and based on desired caps decide target device for sending commands to it)
UI Automation
iOS device/emulator
UI Automator
Android device/emulator

## WORKFLOW OFAPPIUM – Native App

1. From Web-driver, Automation Commands are sent in form of JSON via HTTP request to Appium Server.
2. Appium Server invokes Vendor specific mechanism to execute those commands on the Mobile-Device.
3. Mobile Device (Client) sends back the message to the Appium Server.
4. Appium Server logs the result in the console of the Web Driver.

## WORKFLOW OFAPPIUM – Native App

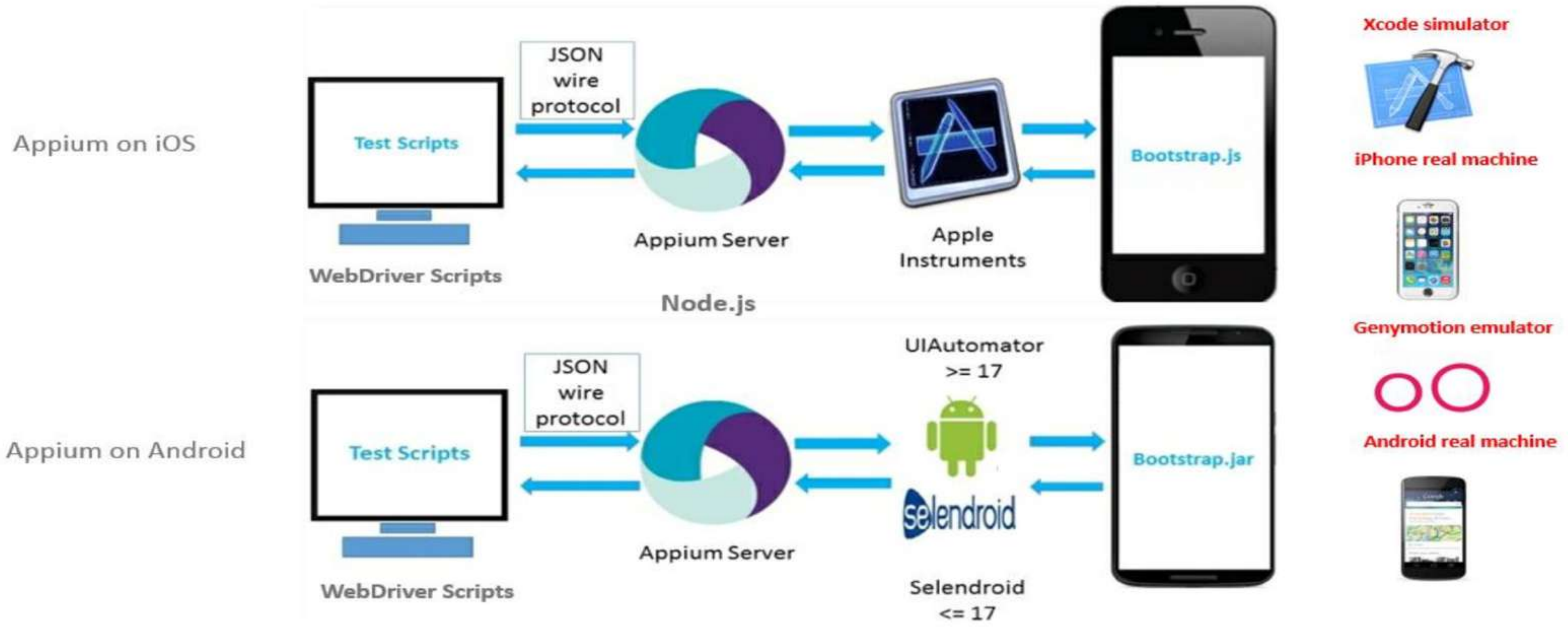

Web Driver Script
Automation commands are sent in form of JSON via HTTP request
1
Appium Server (Node.js)
Appium Server logs the result in the console
4
Appium Server invokes Vendor specific mechanism to execute those commands
2
Client sends back the message to the Appium Server
3

## VENDOR PROVIDED FRAMEWORKS IN APPIUM

The vendor-provided frameworks used are:

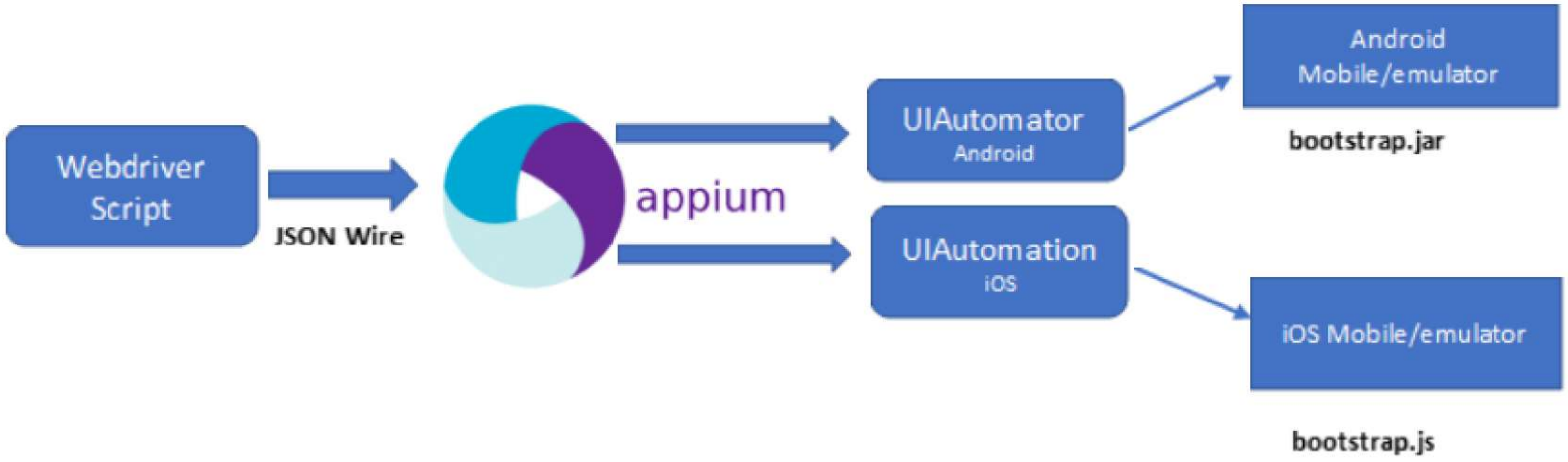
- iOS v9.3 and above: Apple's **XCUITest (UIAutomation)**
- iOS v9.2 and lower: Apple's **XCTest**
- Android v4.2 to v6.0: Google's **UiAutomator1**
- Android v4.2 and above: Google's **UiAutomator2**
- Android v10.0 and above: Google’s **Espresso**
- Android v2.3 to v4.1: Google's Instrumentation, **Selendroid**

## APPIUM – WAY OF WORKING – Native App

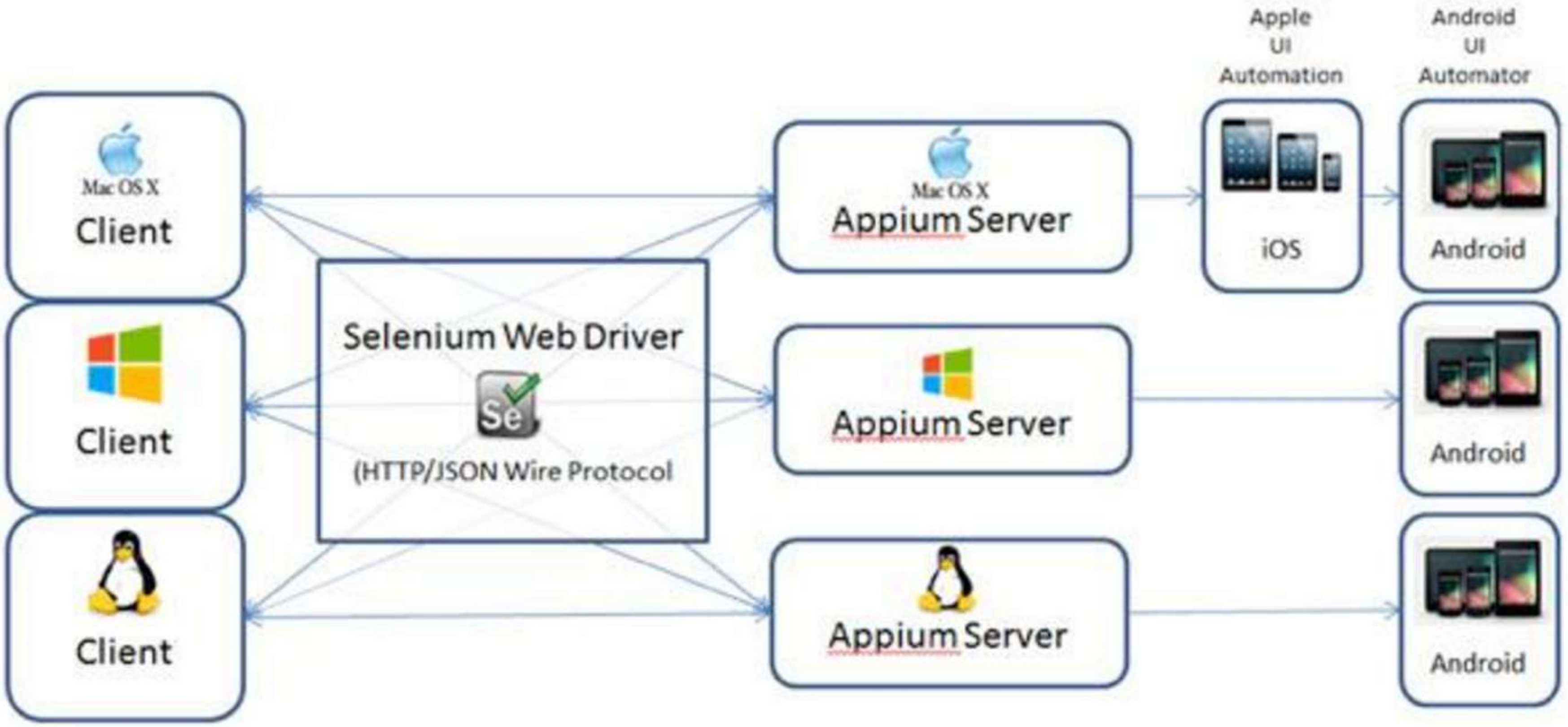


** From version 1.6, Appium added support to XCUITest for iOS10 / Xcode 8, and UiAutomator2 support for Android*

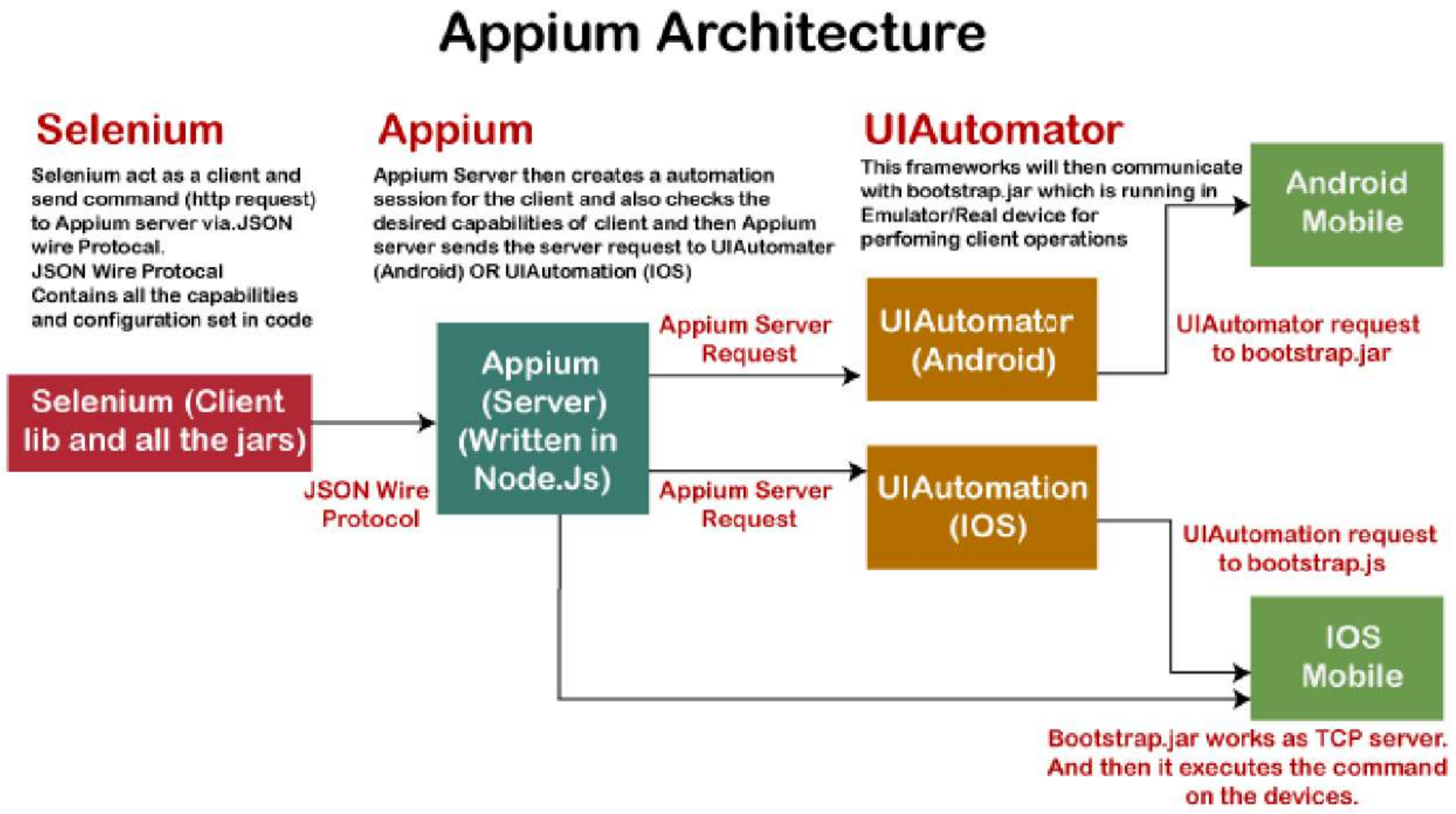
## APPIUM – WAY OF WORKING – Native App



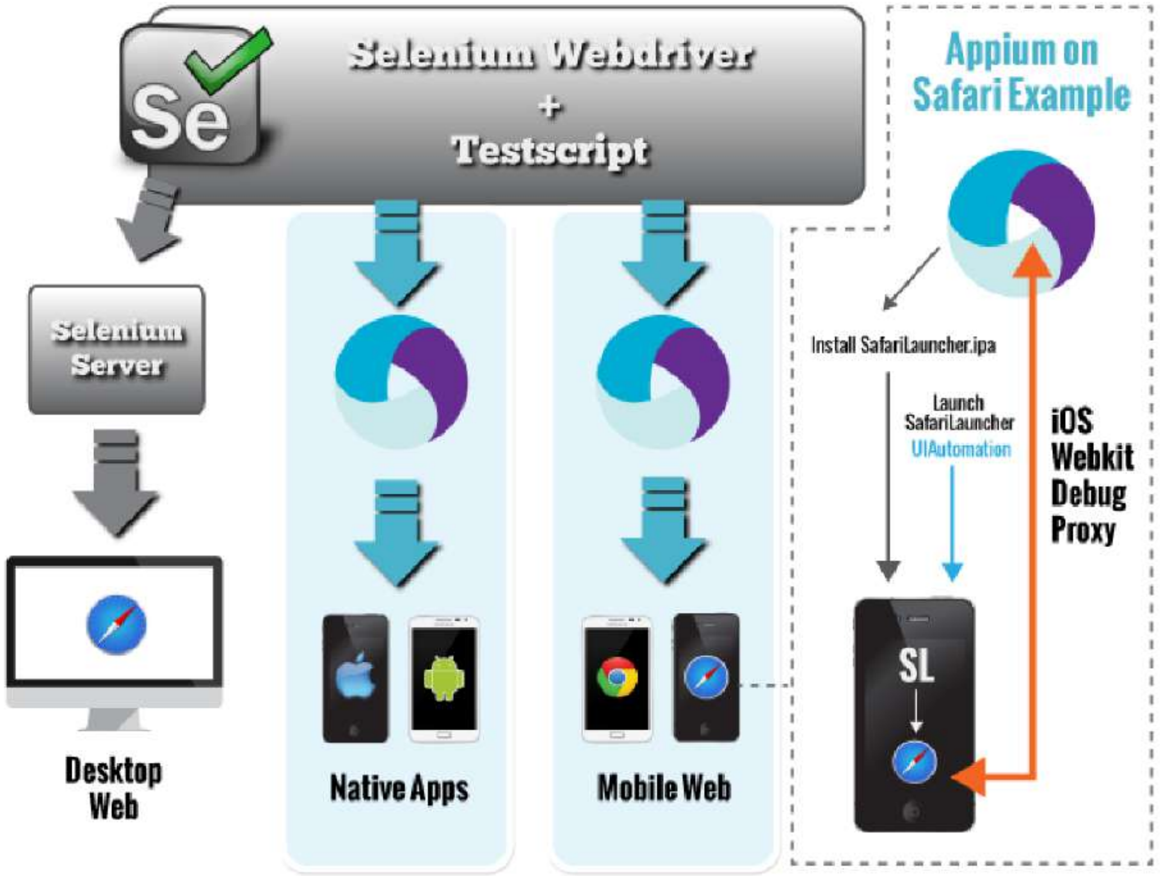
## APPIUM – WAY OF WORKING – Native App



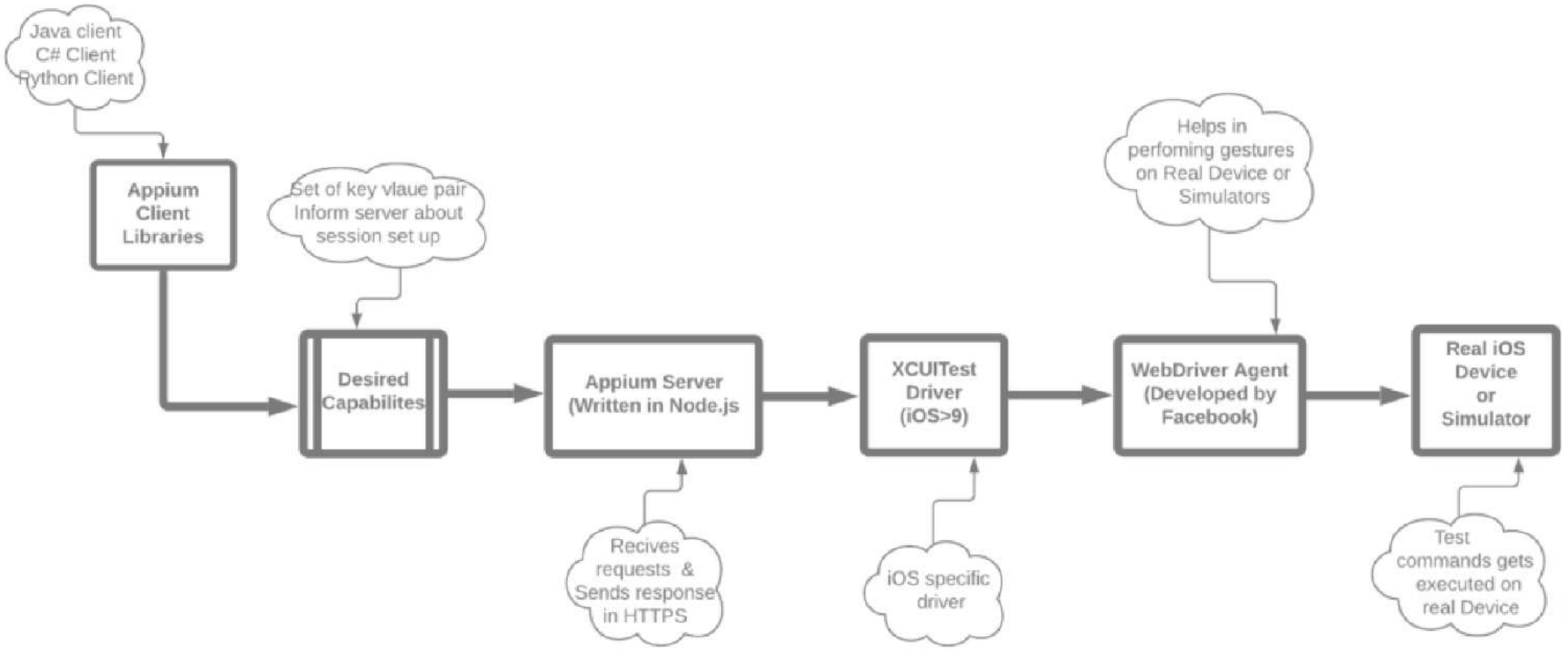
## APPIUM – WAY OF WORKING – Native App


Appium Architecture
Selenium
Selenium act as a client and send command (http request) to Appium server via.JSON wire Protocal.
JSON Wire Protocal Contains all the capabilities and configuration set in code
Appium
Appium Server then creates a automation session for the client and also checks the desired capabilities of client and then Appium server sends the server request to UIAutomater (Android) OR UIAutomation (IOS)
UIAutomator
This frameworks will then communicate with bootstrap.jar which is running in Emulator/Real device for perfoming client operations
Selenium (Client lib and all the jars)
JSON Wire Protocol
Appium (Server) (Written in Node.Js)
Appium Server Request
Appium Server Request
UIAutomator (Android)
UIAutomation (IOS)
UIAutomator request to bootstrap.jar
UIAutomation request to bootstrap.js
Android Mobile
IOS Mobile
Bootstrap.jar works as TCP server. And then it executes the command on the devices.

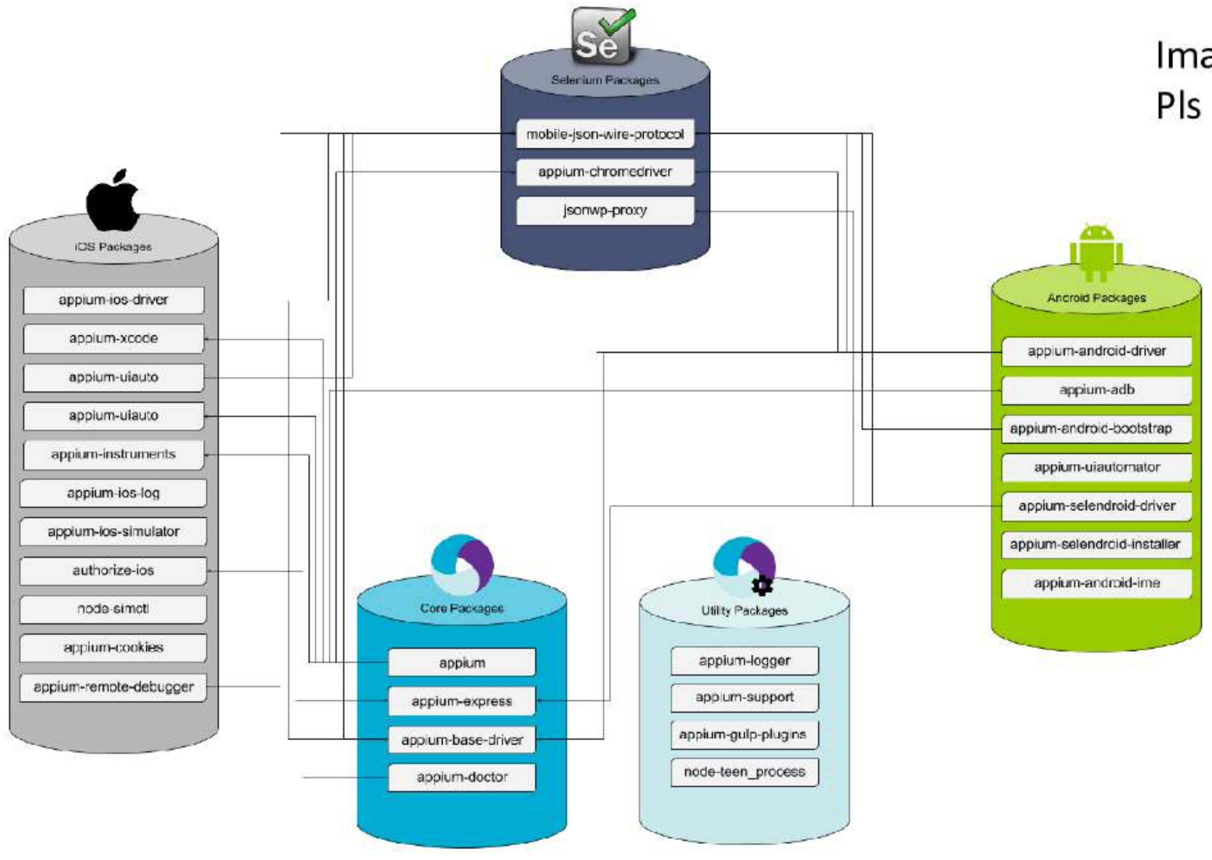
## APPIUM – WAY OF WORKING – Mobile Web



## APPIUM – iOS-WAY OF WORKING – Native App


Java client
C# Client
Python Client
Appium Client Libraries
Set of key vlaue pair Inform server about session set up
Desired Capabilites
Appium Server (Written in Node.js
Recives requests & Sends response in HTTPS
XCUITest Driver (iOS>9)
iOS specific driver
Helps in perfoming gestures on Real Device or Simulators
WebDriver Agent (Developed by Facebook)
Real iOS Device or Simulator
Test commands gets executed on real Device

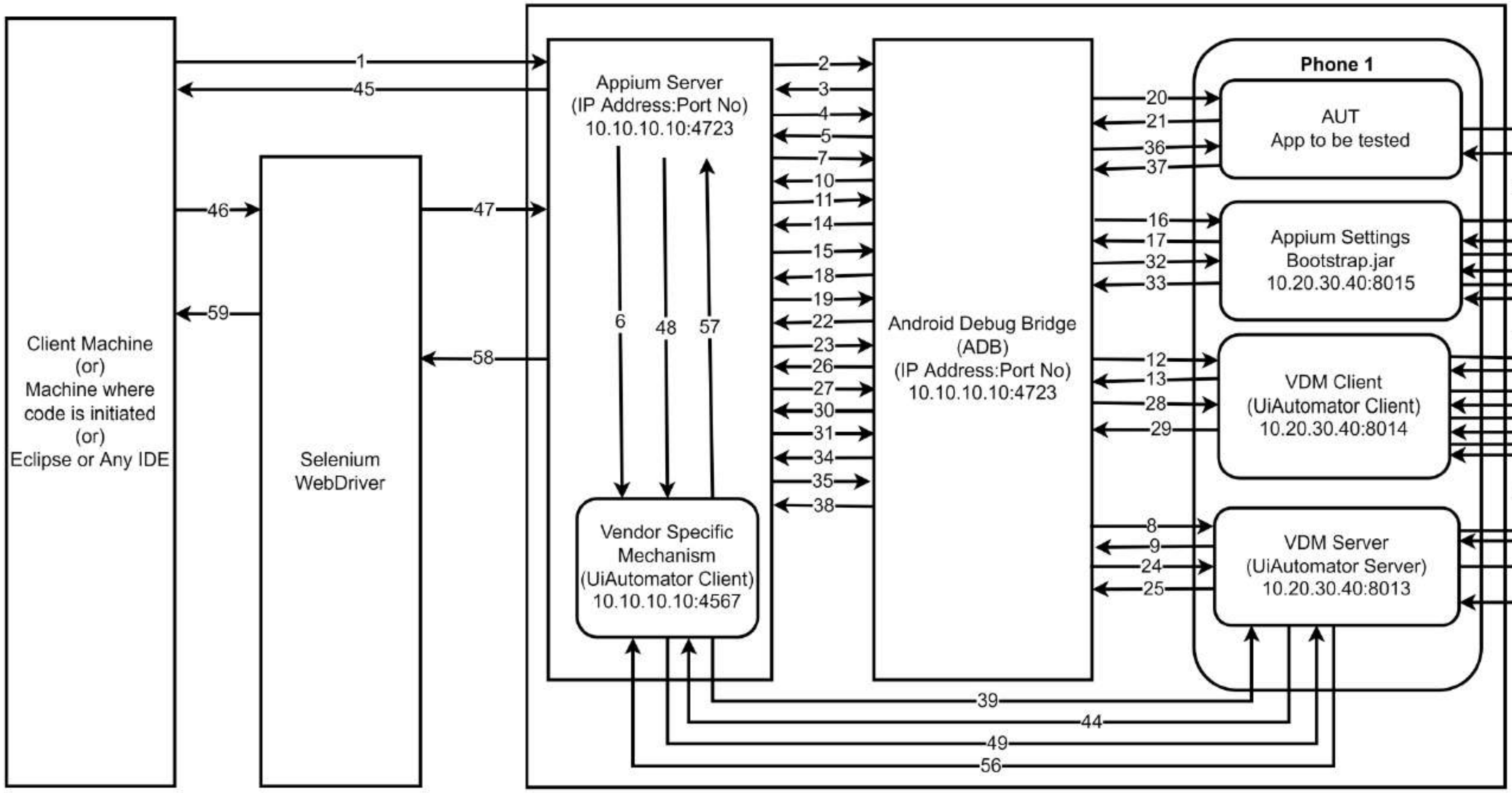
## APPIUM – REFERENCE LIBRARY USED



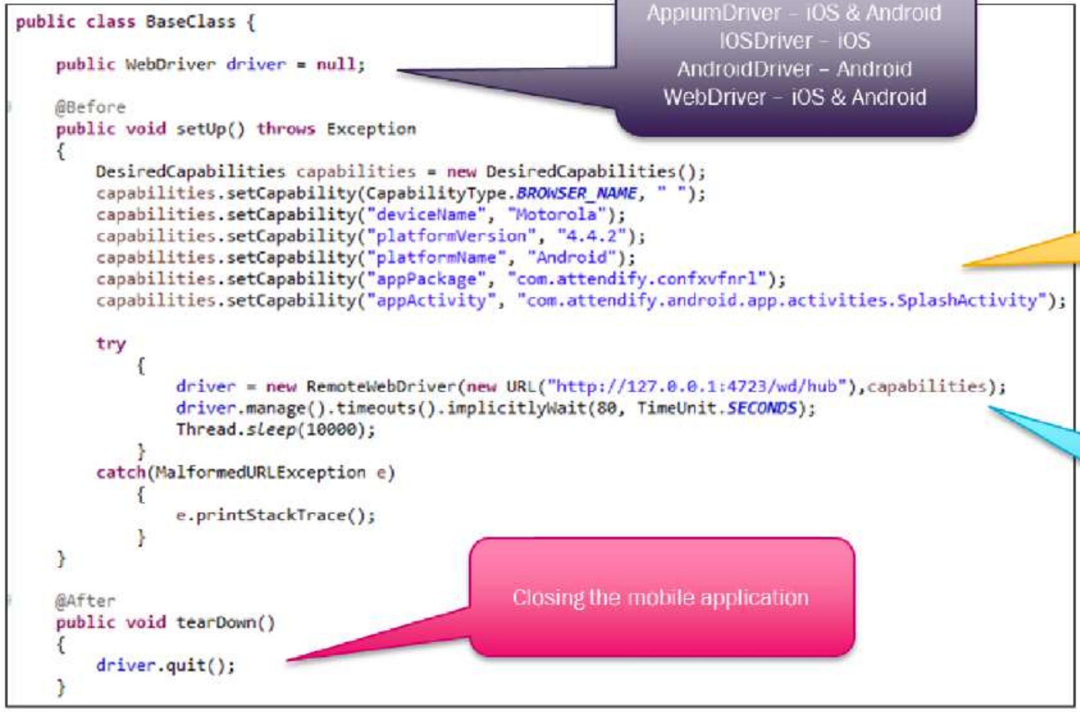
* for the sake of simplicity we pass on dependency relationships of packages within framework container and to commonly used utilities

Image is just for reference.
Pls don't get confused

# ANDROID ARCHITECHTURE - DETAILED – Native App



## CODE STRUCTURE OVERVIEW



```
public class BaseClass {

    public WebDriver driver = null;

    @Before
    public void setUp() throws Exception
    {
        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability(CapabilityType.BROWSER_NAME, " ");
        capabilities.setCapability("deviceName", "Motorola");
        capabilities.setCapability("platformVersion", "4.4.2");
        capabilities.setCapability("platformName", "Android");
        capabilities.setCapability("appPackage", "com.attendify.confxvfnrl");
        capabilities.setCapability("appActivity", "com.attendify.android.app.activities.SplashActivity");

        try
            {
                driver = new RemoteWebDriver(new URL("http://127.0.0.1:4723/wd/hub"),capabilities);
                driver.manage().timeouts().implicitlyWait(80, TimeUnit.SECONDS);
                Thread.sleep(10000);
            }
        catch(MalformedURLException e)
            {
                e.printStackTrace();
            }
    }

    @After
    public void tearDown()
    {
        driver.quit();
    }
```

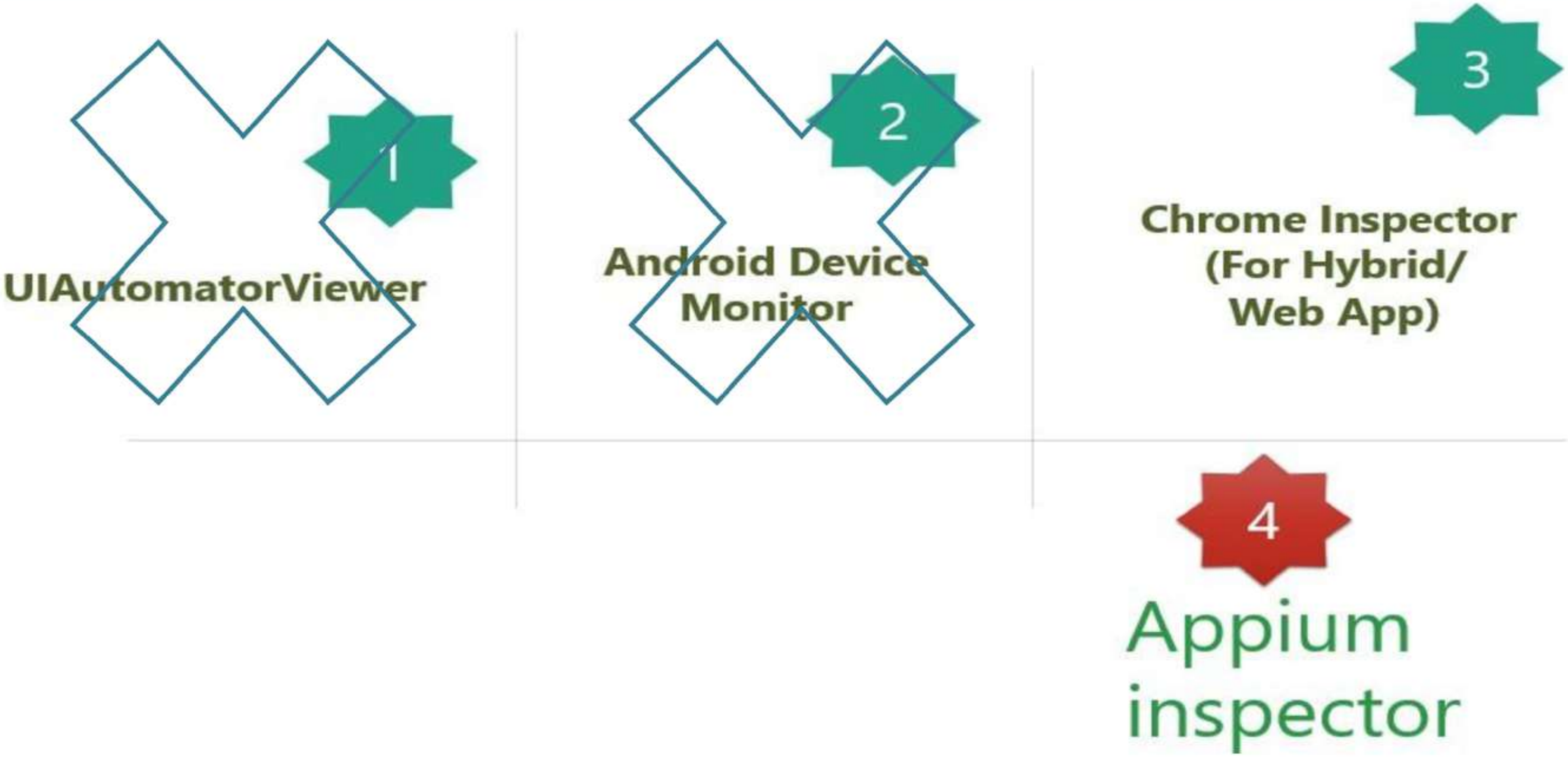
AppiumDriver – iOS & Android
IOSDriver – iOS
AndroidDriver – Android
WebDriver – iOS & Android

Device Capabilities including package name and app activity(screen)

Remote Web Driver instance connect the Appium server

Closing the mobile application

## UI ELEMENT INSPECTORS



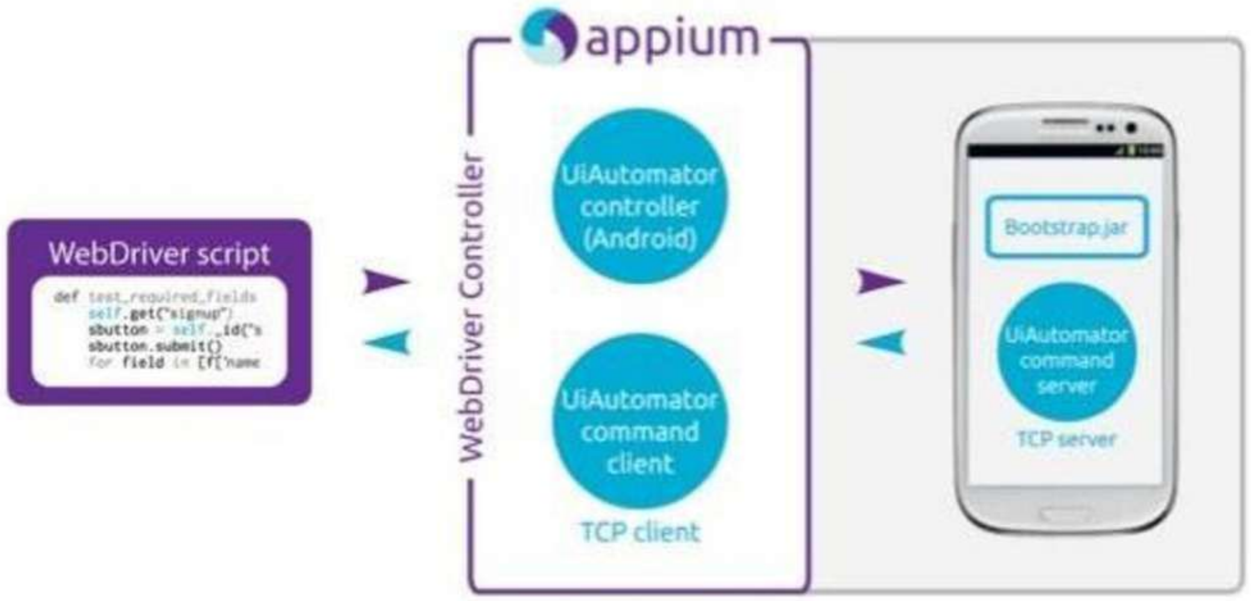
## UI LOCATORS – NATIVE APP/HYBRID APP (NATIVE VIEW)

| | | |
|---|---|---|
| id ✓ | name ✓ | linkText ✗ |
| className ✓ | tagName ✗ | Partial LinkText ✗ |
| xpath ✓ | css ✗ | Accessibility Id ✓ |

## UI LOCATORS – MOBILE WEB/HYBRID APP (WEB VIEW)

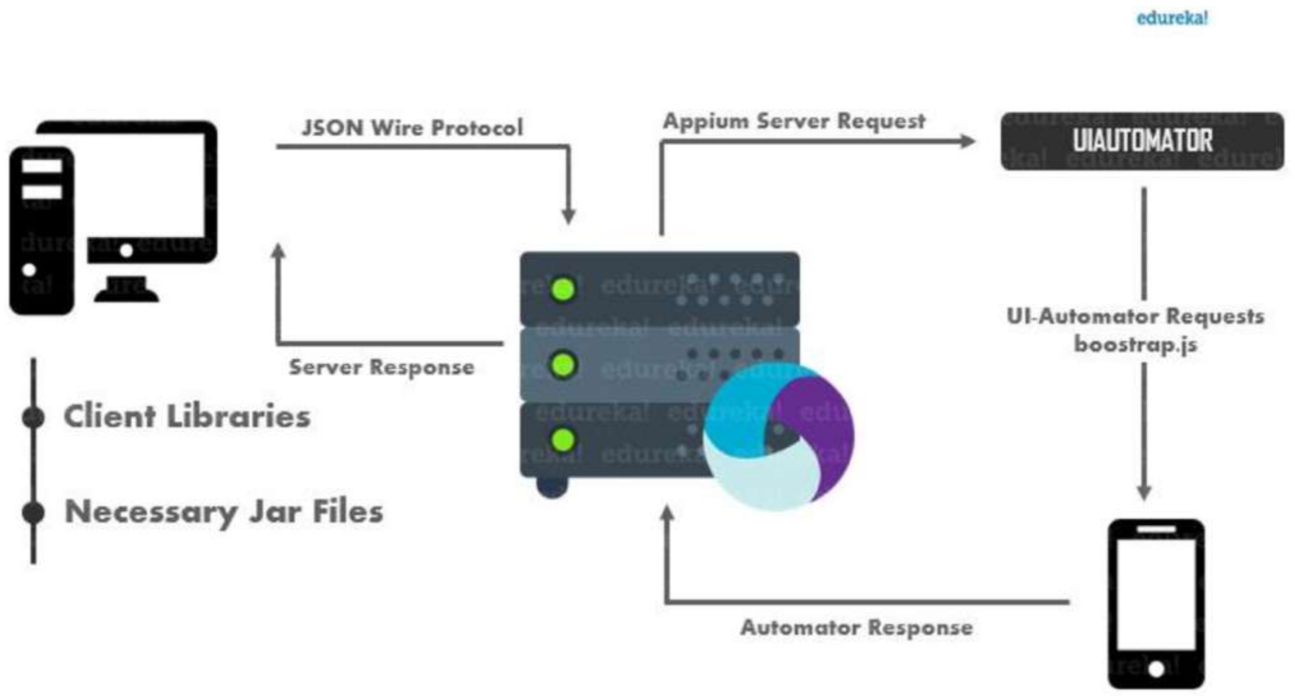
| | | |
|---|---|---|
| id ✓ | name ✓ | linkText ✓ |
| className ✓ | tagName ✓ | Partial LinkText ✓ |
| xpath ✓ | css ✓ | Accessibility Id ✗ |

## APPIUM ARCHITECTURE [ANDROID]

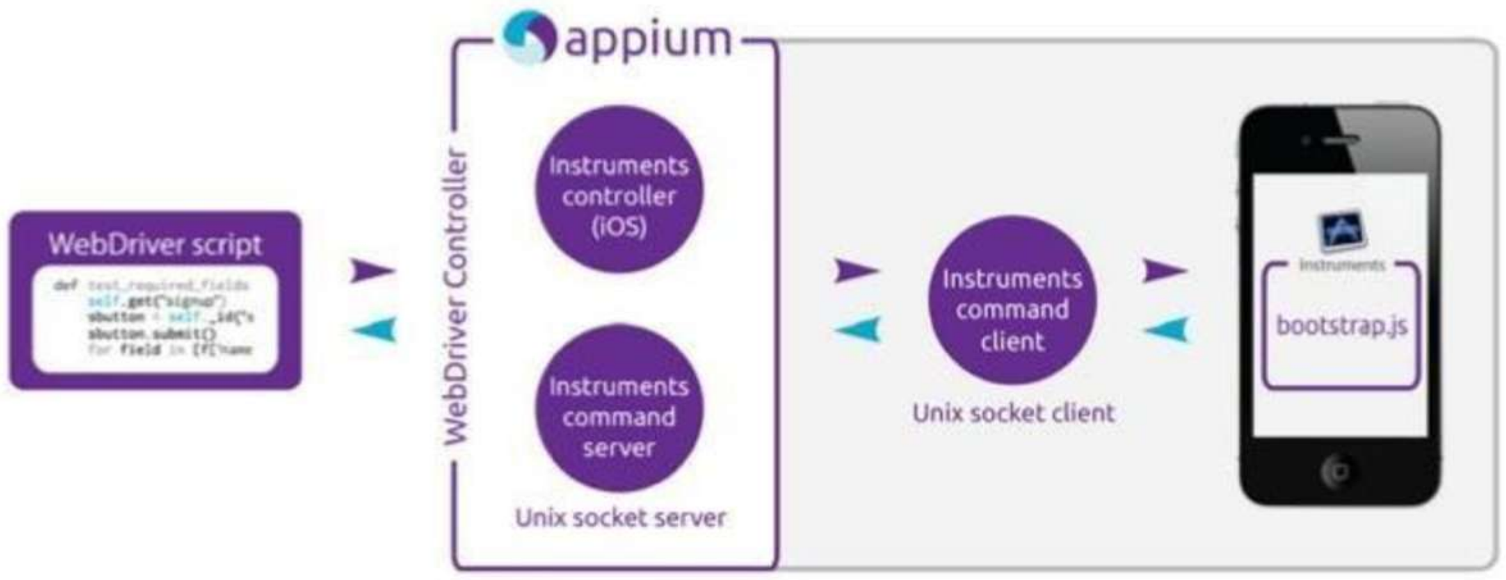

WebDriver script
appium
WebDriver Controller
UiAutomator controller (Android)
UiAutomator command client
TCP client
Bootstrap.jar
UiAutomator command server
TCP server

```
def test_required_fields
    self.get("signup")
    sbutton = self._id("s
    sbutton.submit()
    for field in [f['name
```

## APPIUM ARCHITECTURE [ANDROID]

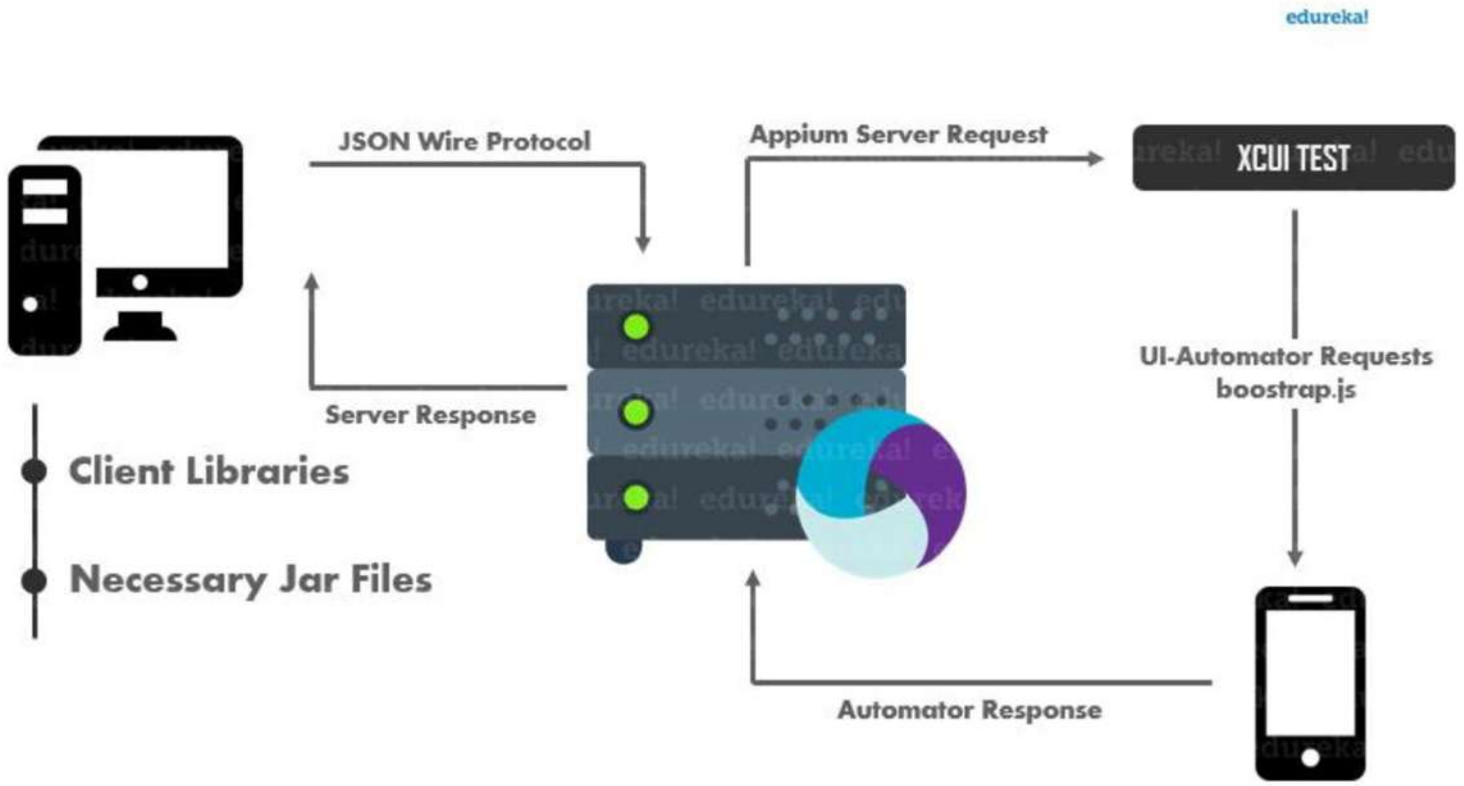


## APPIUM ARCHITECTURE [IOS]



```
def test_required_fields
    self.get("signup")
    $button = self._id("s
    $button.submit()
    for field in [f["name"
```

# APPIUM ARCHITECTURE [IOS]



## ANDROID DEBUG BRIDGE

It is a command line tool that lets you communicate with an emulator instance or connected Android-powered device. It is a client-server program that includes three components:

- A client, which runs on your development machine. You can invoke a client from a shell by issuing an adb command. Other Android tools such as DDMS also create adb clients.
- A server, which runs as a background process on your development machine. The server manages communication between the client and the adb daemon running on an emulator or device.
- A daemon, which runs as a background process on each emulator or device instance.

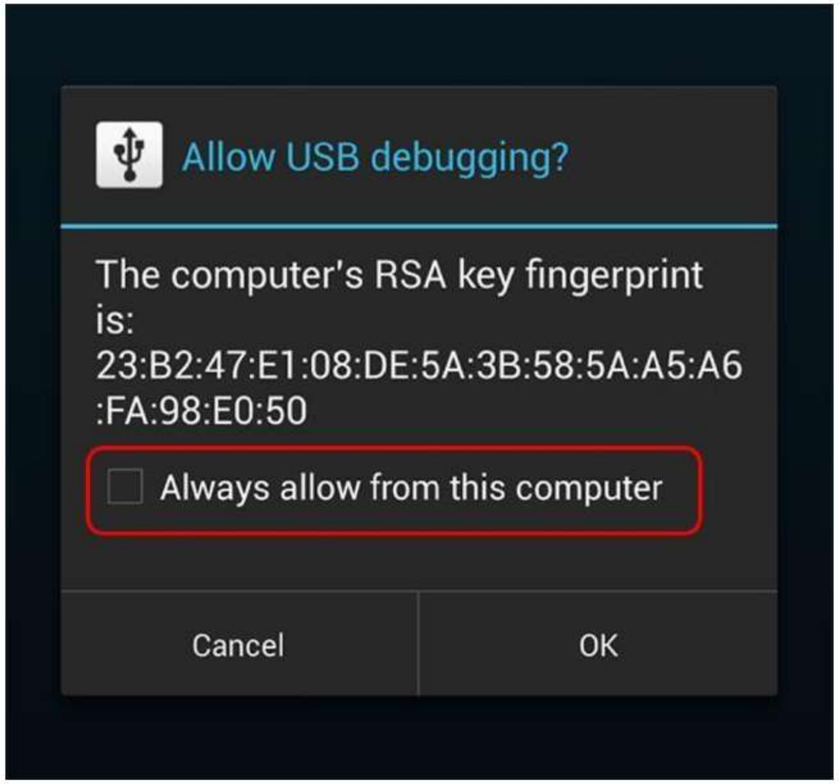
## ENABLING ADB DEBUGGING

On Android 4.2 and higher, the Developer options screen is hidden by default. To make it visible, go to **Settings > About phone** and tap **Build number** seven times.

When you connect a device running Android 4.2.2 or higher to your computer, the system shows a dialog asking whether to accept an RSA key that allows debugging through this computer.

This security mechanism protects user devices because it ensures that USB debugging and other adb commands cannot be executed unless you're able to unlock the device and acknowledge the dialog.

## ENABLING ADB DEBUGGING


Allow USB debugging?

The computer's RSA key fingerprint is:
23:B2:47:E1:08:DE:5A:3B:58:5A:A5:A6:FA:98:E0:50

☐ Always allow from this computer

## ADB COMMANDS

| Category | Command | Description |
|---|---|---|
| Target Device | -d | Direct an adb command to the only attached USB device. |
| | -e | Direct an adb command to the only running emulator instance. |
| General | devices | Prints a list of all attached emulator/device instances. |
| | version | Prints the adb version number. |

## ADB COMMANDS

| Category | Command | Description |
|---|---|---|
| Debug | logcat | Prints log data to the screen. |
| Data | install | Pushes an Android application (specified as a full path to an .apk file) to an emulator/device. |
| | pull | Copies a specified file from an emulator/device instance to your development computer. |
| | push | Copies a specified file from your development computer to an emulator/device instance. |

## ADB COMMANDS

| Category | Command | Description |
|---|---|---|
| server | start-server | Checks whether the adb server process is running and starts it, if not. |
| | kill-server | Terminates the adb server process. |
| Shell | shell | Starts a remote shell in the target emulator/device instance. |

## DEVICE CONNECTION STATE

| State | Description |
|---|---|
| offline | The instance is not connected to adb or is not responding. |
| device | The instance is now connected to the adb server.<br><br>Note that this state does not imply that the Android system is fully booted and operational, since the instance connects to adb while the system is still booting.<br><br>However, after boot-up, this is the normal operational state of an emulator/device instance. |
| no device / blank | There is no emulator/device connected. |
| unauthorized | Debugging permission is not granted |

## CONNECT TO WIRELESS

| Step | Step Description |
|---|---|
| 1 | Connect your Android device and adb host computer to a common Wi-Fi network accessible to both. |
| 2 | Connect the device to the host computer with a USB cable. |
| 3 | Set the target device to listen for a TCP/IP connection on port 5555.<br>`$ adb tcpip 5555` |
| 4 | Disconnect the USB cable from the target device. |
| 5 | Find the IP address of the Android device. For example, on a Nexus device, you can find the IP address at Settings > About tablet (or About phone) > Status > IP address. Or, on an Android Wear device, you can find the IP address at Settings > Wi-Fi Settings > Advanced > IP address. |
| 6 | Connect to the device, identifying it by IP address.<br>`$ adb connect <device-ip-address>` |

## APPIUM SCRIPT EXECUTION IN REAL DEVICE

## THANK YOU!