Selenium

Selenium is an open-source tool that is used for test automation. It is licensed under Apache License 2.0. Selenium is a suite of tools that helps in automating only web applications. It has capabilities to operate across different browsers and operating systems.

Brief History of The Selenium Project

Selenium first came to life in 2004 when Jason Huggins was testing an internal application at **ThoughtWorks**. He developed a Javascript library that could drive interactions with the page, allowing him to automatically rerun tests against multiple browsers. That library eventually became Selenium Core, which underlies all the functionality of Selenium Remote Control (RC) and Selenium IDE. Selenium RC was ground-breaking because no other product allowed you to control a browser from a language of your choice.

While Selenium was a tremendous tool, it wasn't without its drawbacks. Because of its Javascript based automation engine and **the security limitations browsers apply to Javascript**, different things became impossible to do. To make things worse, webapps became more and more powerful over time, using all sorts of special features new browsers provide and making these restrictions more and more painful.

In 2006 a plucky engineer at Google named Simon Stewart started work on a project he called WebDriver. Google had long been a heavy user of Selenium, but testers had to work around the limitations of the product. Simon wanted a testing tool that spoke directly to the browser using the ‘native’ method for the browser and operating system, thus avoiding the restrictions of a sandboxed Javascript environment. The WebDriver project began with the aim to solve the Selenium’ pain-points.

Selenum Tools

Selenium IDE

Selenium Integrated Development Environment (IDE) is a Firefox plugin that lets testers to record their actions as they follow the workflow that they need to test then exports them as a reusable script in one of many programming languages that can be later executed.

It is not designed to run your test passes nor is it designed to build all the automated tests you will need. Specifically, Selenium IDE doesn’t provide iteration or conditional statements for test scripts.

Selenium RC

Selenium Remote Control (RC) was the flagship testing framework that allowed more than simple browser actions and linear execution. It makes use of the full power of programming languages such as Java, C#, PHP, Python, Ruby and PERL to create more complex tests.

Selenium WebDriver

Selenium WebDriver is the successor to Selenium RC which sends commands directly to the browser and retrieves results.

Selenium Grid

Selenium Grid is a tool used to run parallel tests across different machines and different browsers simultaneously which results in minimized execution time.

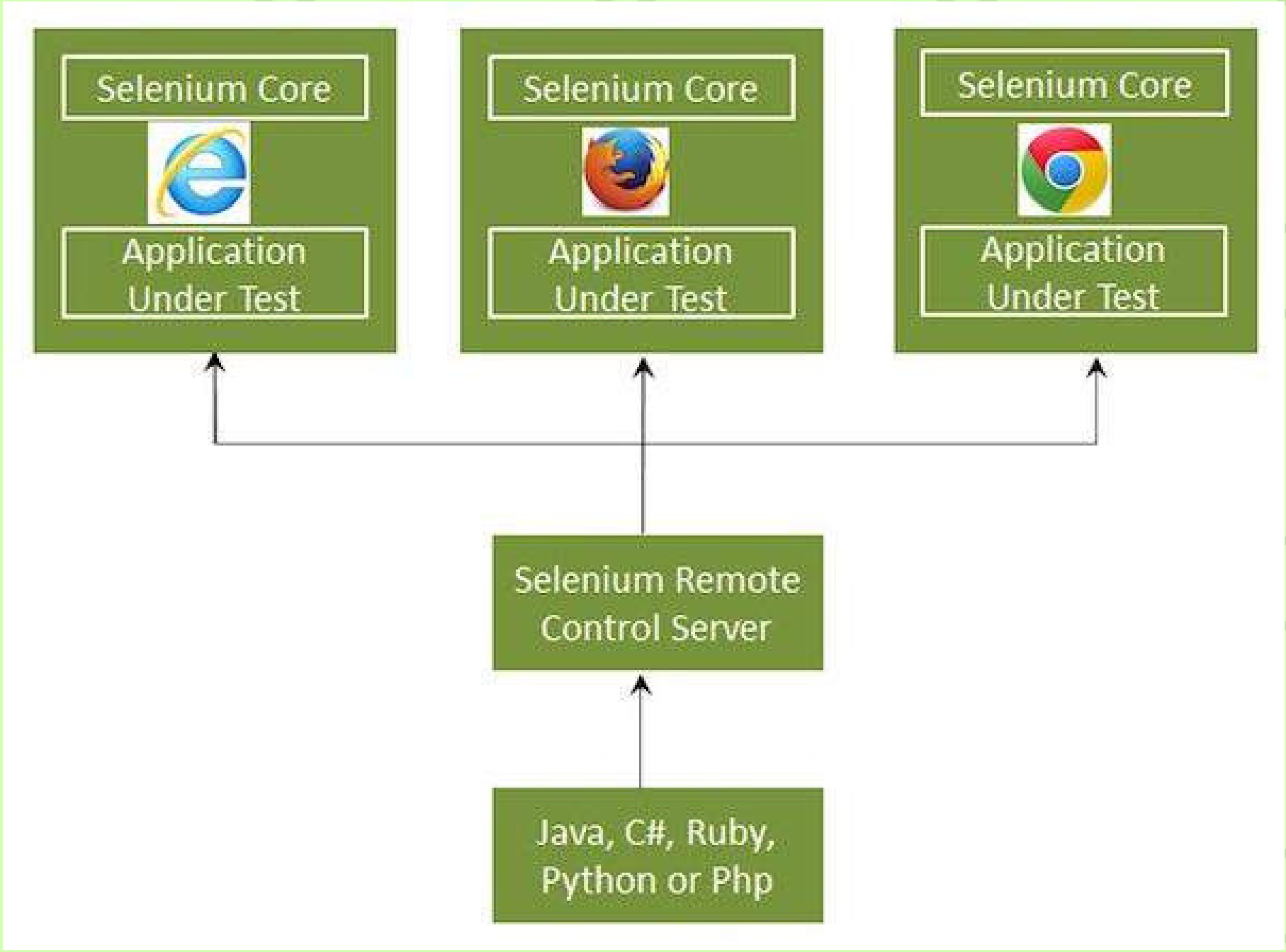
Selenium RC

Selenium RC works in such a way that the client libraries can communicate with the Selenium RC Server passing each Selenium command for execution. Then the server passes the Selenium command to the browser using Selenium-Core JavaScript commands.

Selenium RC comes in two parts.

1. The Selenium Server launches and kills browsers. In addition to that, it interprets and executes the Selenese commands. It also acts as an HTTP proxy by intercepting and verifying HTTP messages passed between the browser and the application under test.

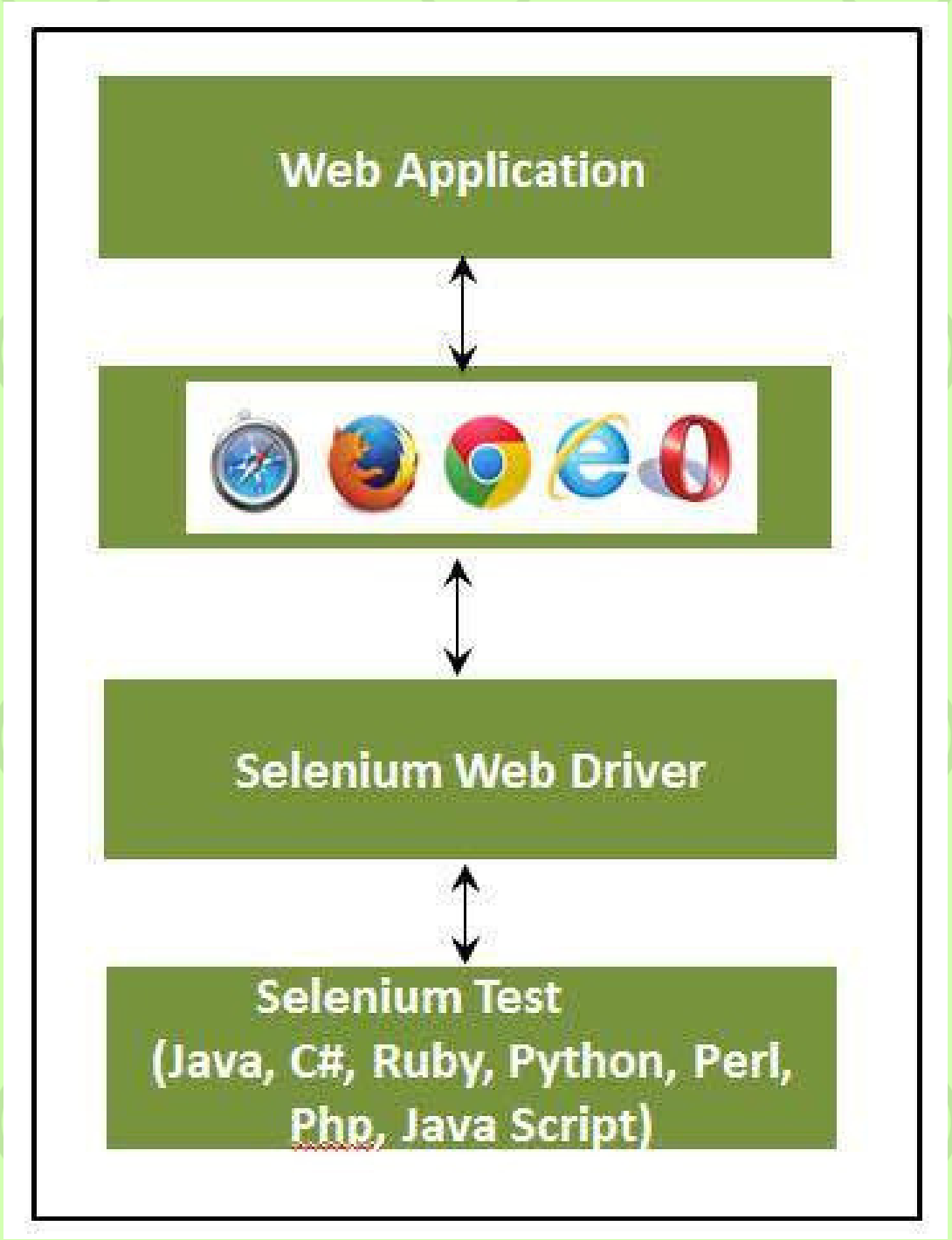
2.Client libraries that provide an interface between each one of the programming languages (Java, C#, Perl, Python and PHP) and the Selenium-RC Server.


Selenium Core
Application Under Test
Selenium Core
Application Under Test
Selenium Core
Application Under Test
Selenium Remote Control Server
Java, C#, Ruby, Python or Php

Selenium Webdriver

WebDriver is a tool for automating testing web applications. It is popularly known as Selenium 2.0.
WebDriver interacts directly with the browser without any intermediary, unlike Selenium RC that depends on a server. It is used in the following context:

1. Multi-browser testing including improved functionality for browsers which is not well-supported by Selenium RC (Selenium 1.0).
2. Handling multiple frames, multiple browser windows, popups, and alerts.
3. Complex page navigation.
4. Advanced user navigation such as drag-and-drop.


Web Application
Selenium Web Driver
Selenium Test
(Java, C#, Ruby, Python, Perl,
Php, Java Script)

Selenium RC VS Selenium WebDriver

1. The architecture of Selenium RC is complicated, as the server needs to be up and running before starting a test.

WebDriver's architecture is simpler than Selenium RC, as it controls the browser from the OS level.

2. Selenium server acts as a middleman between the browser and Selenese commands.

WebDriver interacts directly with the browser and uses the browser's engine to control it.

3. Selenium RC script execution is slower, since it uses a Javascript to interact with RC.

WebDriver is faster, as it interacts directly with the browser.

4. Selenium RC cannot support headless execution as it needs a real browser to work with.

WebDriver can support the headless execution.

5. It's a simple and small API.

Complex and a bit large API as compared to RC.

6. Less object-oriented API.

Purely object oriented API.

7. Cannot test mobile Applications.

Can test iPhone/Android applications.

Most Used Commands

driver.get("URL")	To navigate to an application.
element.sendKeys("inputtext")	Enter some text into an input box.
element.clear()	Clear the contents from the input box.
select.deselectAll()	Deselect all OPTIONs from the first SELECT on the page.
select.selectByVisibleText("some text")	Select the OPTION with the input specified by the user.
driver.switchTo().window("windowName")	Move the focus from one window to another.
driver.switchTo().frame("frameName")	Swing from frame to frame.
driver.switchTo().alert()	Helps in handling alerts.
driver.navigate().to("URL")	Navigate to the URL.
driver.navigate().forward()	To navigate forward.
driver.navigate().back()	To navigate back.
driver.close()	Closes the current browser associated with the driver.
driver.quit()	Quits the driver and closes all the associated window of that driver..

Selenium Locators

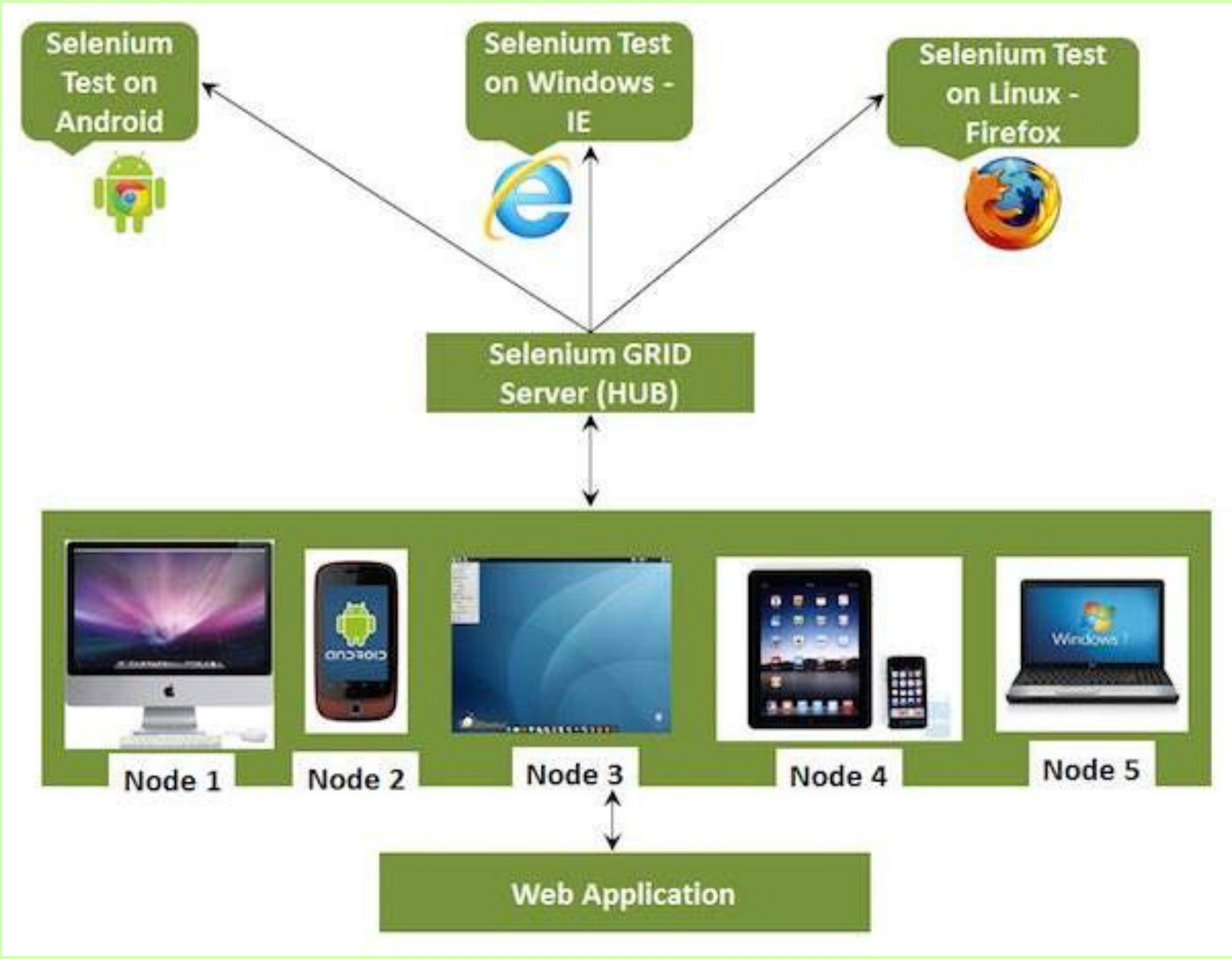
By ID driver.findElement(By.id(<element ID>))
Locates an element using the ID attribute
By name driver.findElement(By.name(<element name>))
Locates an element using the Name attribute
By class name driver.findElement(By.className(<element class>))
Locates an element using the Class attribute
By tag name driver.findElement(By.tagName(<htmltagname>))
Locates an element using the HTML tag
By link text driver.findElement(By.linkText(<linktext>))
Locates a link using link text
By partial link text driver.findElement(By.partialLinkText(<linktext>))
Locates a link using the link's partial text
By CSS driver.findElement(By.cssSelector(<css selector>))
Locates an element using the CSS selector
By XPath driver.findElement(By.xpath(<xpath>))
Locates an element using XPath query

Selenium Grid

Add Selenium Grid is a tool that distributes the tests across multiple physical or virtual machines so that we can execute scripts in parallel (simultaneously). It dramatically accelerates the testing process across browsers and across platforms by giving us quick and accurate feedback. Selenium Grid allows us to execute multiple instances of WebDriver or Selenium Remote Control tests in parallel which uses the same code base, hence the code need NOT be present on the system they execute.

Hub - The hub can also be understood as a server which acts as the central point where the tests would be triggered. A Selenium Grid has only one Hub and it is launched on a single machine once.

Node - Nodes are the Selenium instances that are attached to the Hub which execute the tests. There can be one or more nodes in a grid which can be of any OS and can contain any of the Selenium supported browsers.


Selenium Test on Android
Selenium Test on Windows - IE
Selenium Test on Linux - Firefox
Selenium GRID Server (HUB)
Node 1
Node 2
Node 3
Node 4
Node 5
Web Application

Step 1: Start the hub

The Hub is the central point that will receive all the test request and distribute them the the right .

java -jar selenium-server-standalone-<version>.jar -role hub

The hub will automatically start-up using port 4444 by default. To change the default port, you can add the optional parameter -port when you run the command. You can view the status of the hub by opening a browser window and navigating to: http://localhost:4444/grid/console.

Step 2: Start the nodes

java -jar selenium-server-standalone-<version>.jar -role node -hub http://localhost:4444/grid/register

Step 3 : Using grid to run tests

For WebDriver nodes, you will need to use the RemoteWebDriver and the DesiredCapabilities object to define which browser, version and platform you wish to use. Create the target browser capabilities you want to run the tests against:

```
DesiredCapabilities capability = DesiredCapabilities.firefox();

WebDriver driver = new RemoteWebDriver(new
URL("http://localhost:4444/wd/hub"), capability);
```

The hub will then assign the test to a matching node.

```
capability.setBrowserName();
capability.setPlatform();
capability.setVersion()
capability.setCapability(,);
```

Configuring the nodes

The node can be configured in 2 different ways; one is by specifying command line parameters, the other is by specifying a json file. By default, starting the node allows for concurrent use of 11 browsers... : 5 Firefox, 5 Chrome, 1 Internet Explorer. The maximum number of concurrent tests is set to 5 by default.

```
java -jar selenium-server-standalone.jar -role node -browser
browserName=firefox,version=3.6,maxInstances=5,platform=LINUX
```

with Json

```
java -jar selenium-server-standalone.jar -role node -nodeConfig
nodeconfig.json
```

Configuring the hub by JSON

```
java -jar selenium-server-standalone.jar -role hub -hubConfig hubconfig.json
```

Adding iPhone/iWebDriver node to Selenium Hub

We need to install the iWebDriver application on the iOS device or simulator. In the Grid section, enter the Host and Port details of the Hub. Setting up an iOS device on the simulator on Hub does not require any configuration file or
command-line options. It can be set very easily from the settings UI in iOS.
When we enter the Host and Port details for Hub and start the iWebDriver application,
it automatically registers itself on the Hub providing the capability to run tests on the
iOS Safari browser.

Adding Android node to Selenium Hub

We need to install and set up **WebDriver APK** on an Android device or emulator.
To register the Android node on the Hub, we need to install the flynnid (https://github.
com/davehunt/flynnid) utility in Python.

```
pip install flynnid
flynnid --nodeport=8080 --browsername=android --browserver=2.20
--platform=ANDROID
```

The flynnid utility registers the Android node by using configurations provided as command
arguments. We need to supply the port of the Android node, browser name, platform Android
and version of the Android Server APK installed on the Android device or emulator

THANK YOU!!!!!!!!