The inside of Selenium

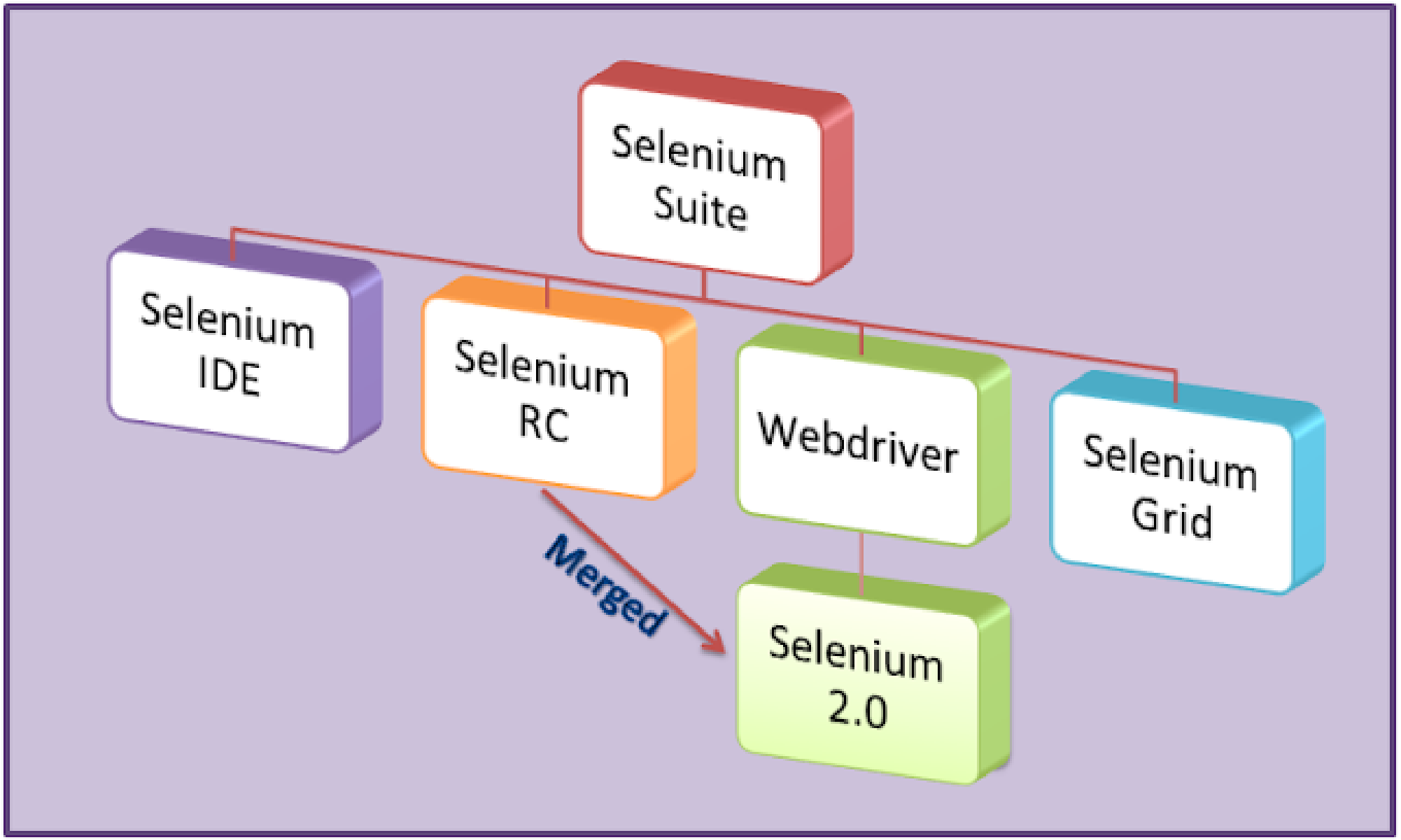
Duy Nguyen
Thach Hoang

NASH TECH
The Power to Innovate

Agenda

- The origins of Selenium
- JSON Wire Protocol
- Selenium - The future
- Selenium WebDriver Exceptions
- Wait Strategies
- Effective Page Objects
- Javascript Executor
- File Downloads
- Super Grid
- AI test automation
- Q&A


THE ORIGINS

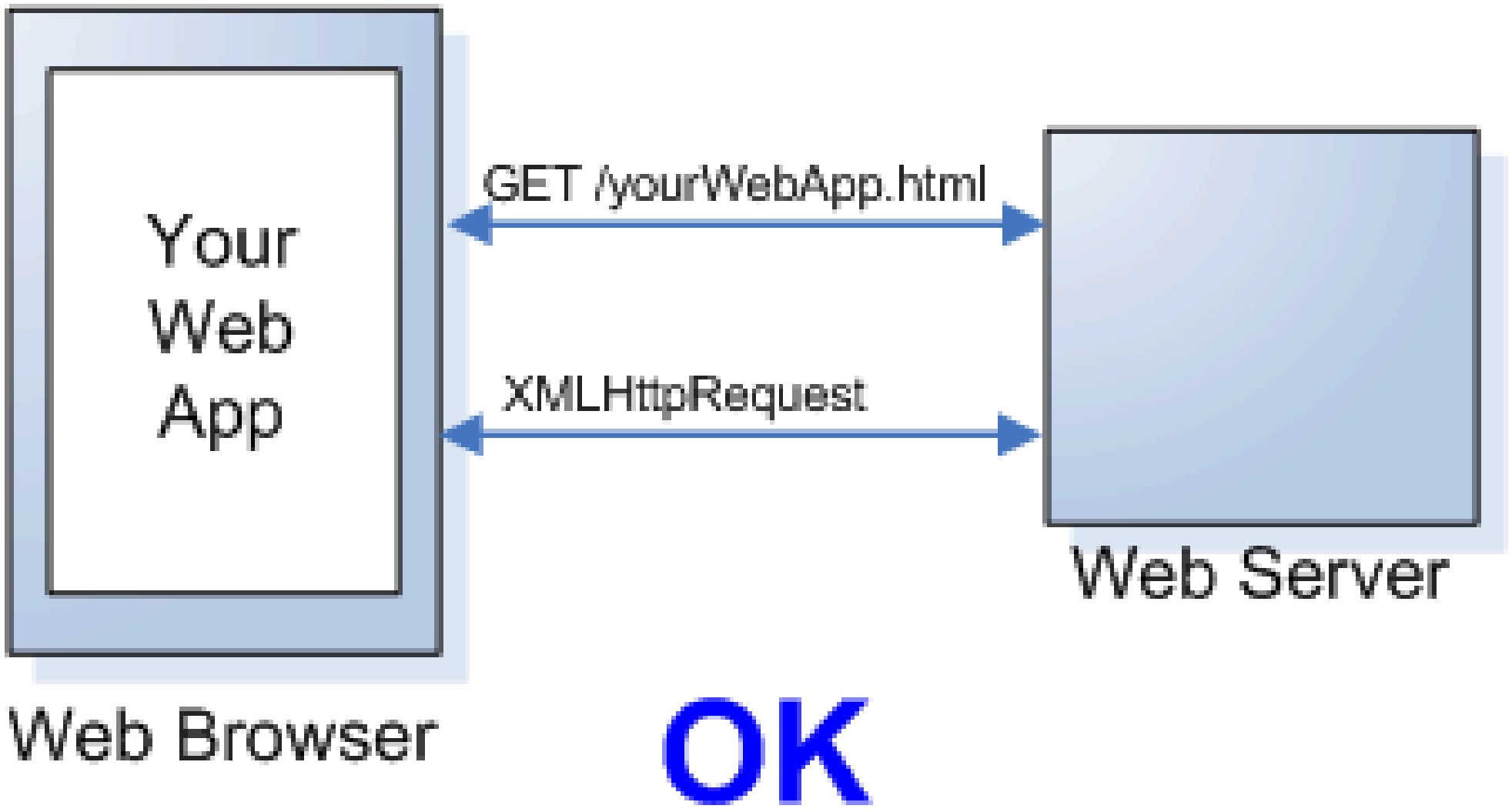


source: http://testingtrooper.blogspot.com/2017/02/what-is-selenium.html

EZ

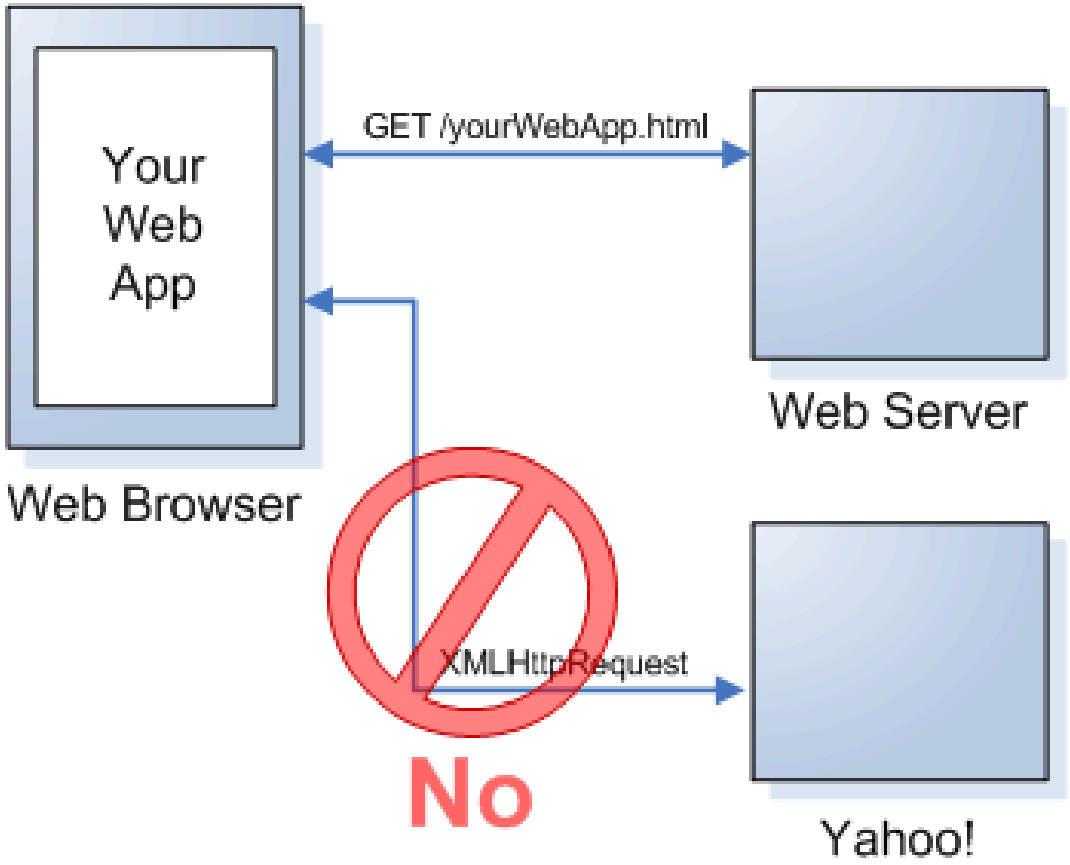
Selenium RC

Same Origin Policy



Source: Selenium-WebDriver-Practical-Guide-Avasarala

Same Origin Policy

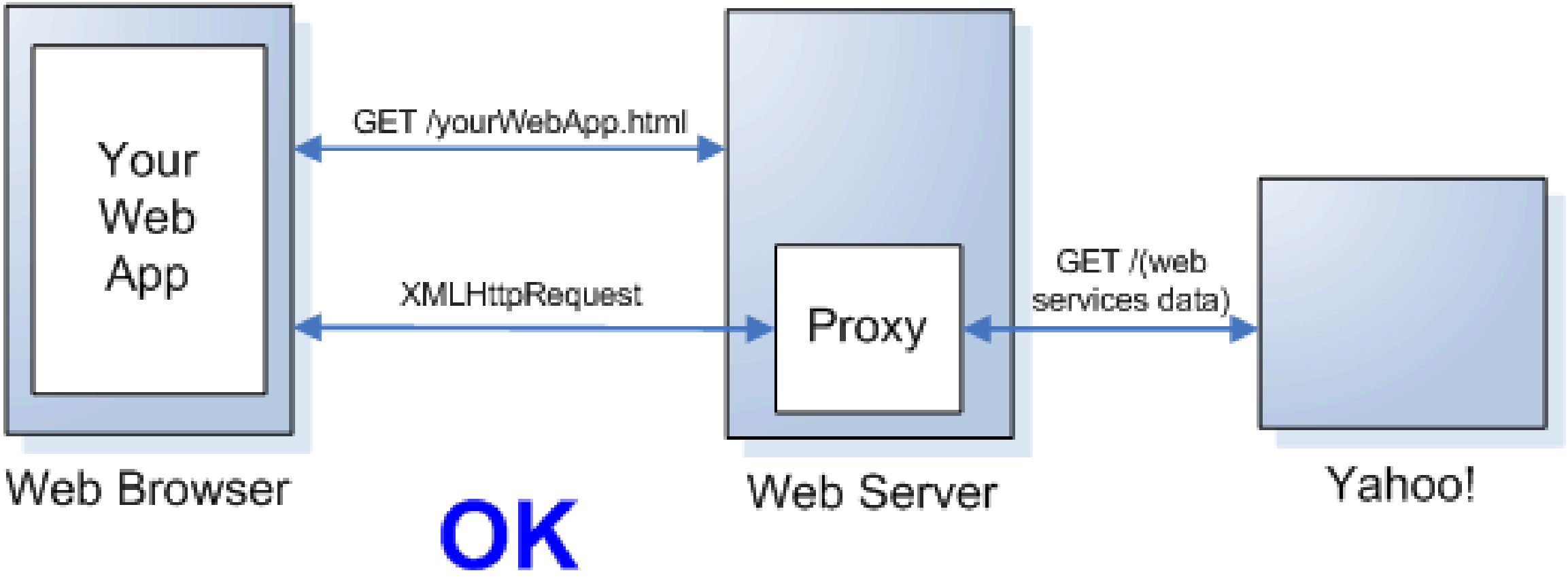


The browser creates a separate sandbox for the website's JavaScript, which restricts the JavaScript to be executed only on it's respective website domain.

Selenium RC, its generic JavaScript is not allowed, by the browser, to execute on a website (WAUT) that is coming from a different domain.

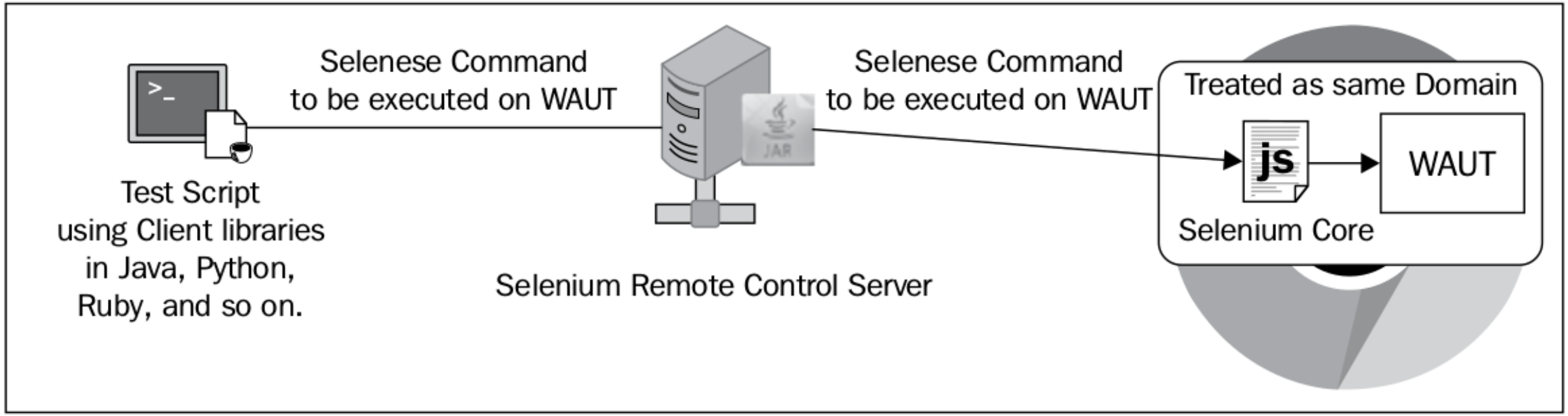
Source: Selenium-WebDriver-Practical-Guide-Avasarala

Same Origin Policy



Source: Selenium-WebDriver-Practical-Guide-Avasarala

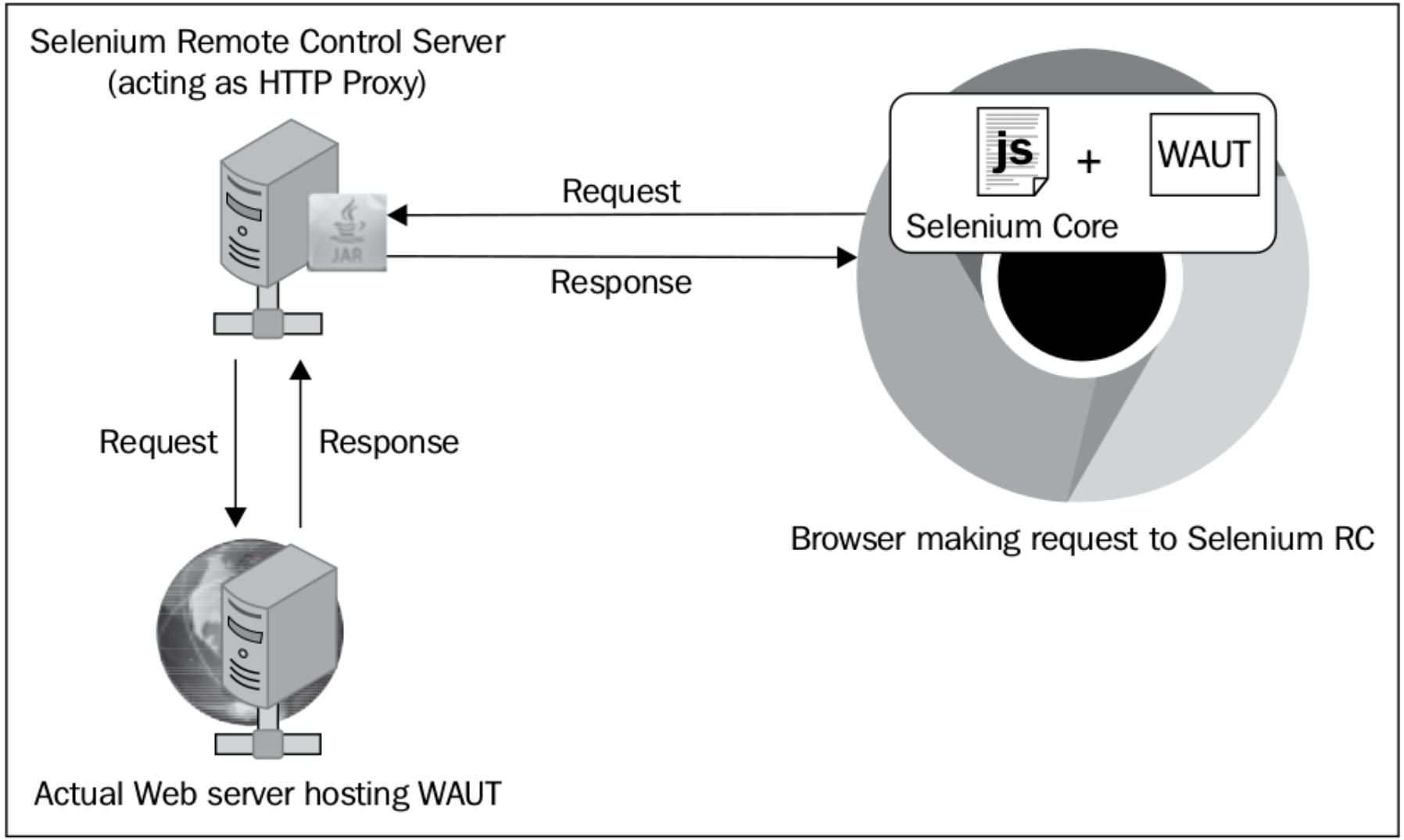
Selenium RC



Selenium RC acts as an HTTP Proxy Server. When the test script asks to launch a browser, Selenium RC server launches the browser and injects its JavaScript (Selenium Core) into the browser

Source: Selenium-WebDriver-Practical-Guide-Avasarala

Selenium RC



The requests and responses of the browser for WAUT go to the actual web server via Selenium RC server

Source: Selenium-WebDriver-Practical-Guide-Avasarala

Why WebDriver?

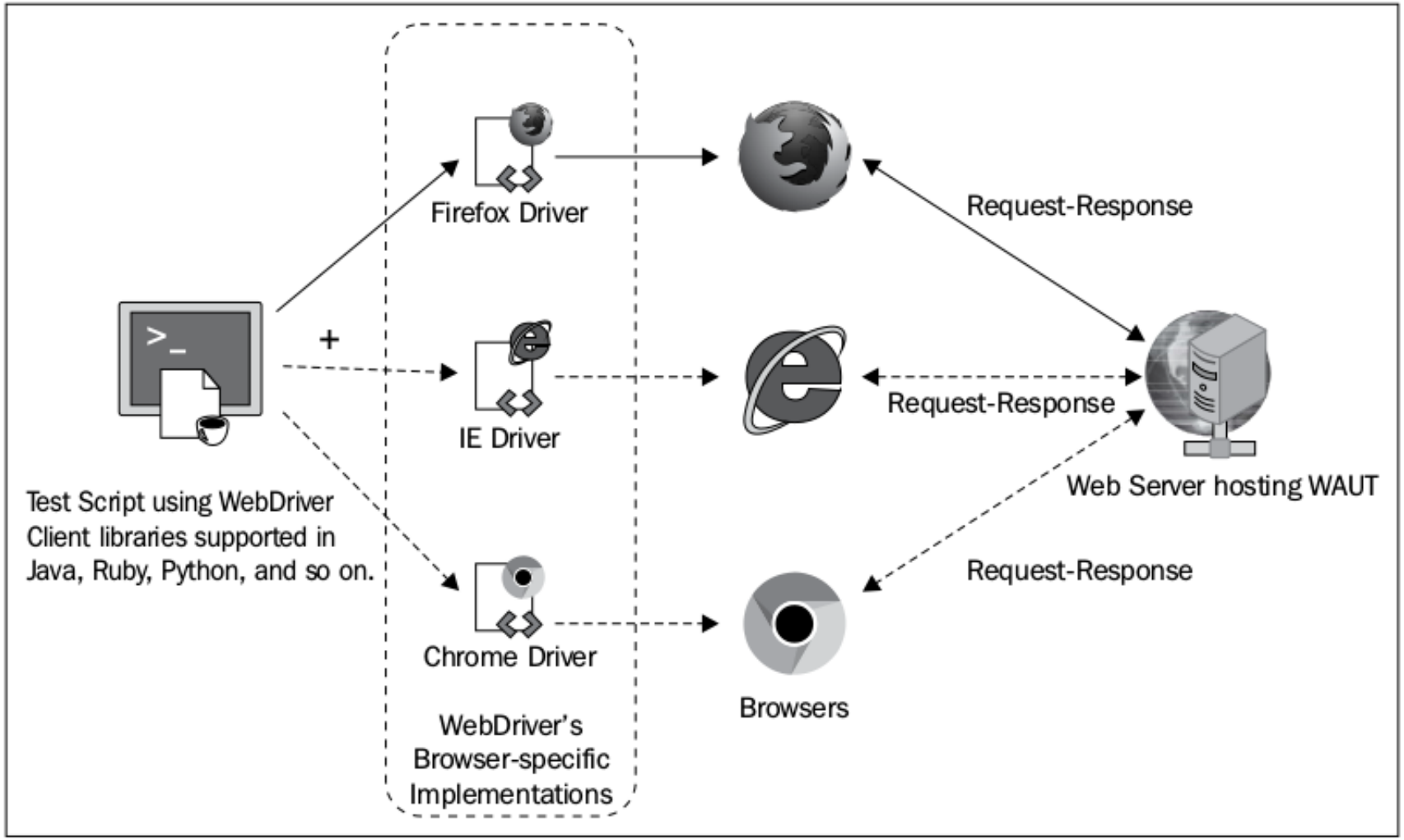
Limitation of Selenium RC:

- Core being limited within the JavaScript sandbox of the browser, as it needs to comply with the Same-Origin policy.
- Because this JavaScript library is generic and not specific to any particular browser, the developers of test scripts sometimes end up with a situation where their test scripts execute very well on some browsers but not on some other.

WebDriver can handle these above problems:

- Giving a better control on the browser by implementing browser-specific implementations.
- Giving a better programming experience to the developer by adhering more closely to the object-oriented programming fundamentals.

Json Wire Protocol



https://github.com/SeleniumHQ/selenium/wiki/JsonWireProtocol

Source: Selenium-WebDriver-Practical-Guide-Avasarala

Start a chrome instance

```
from selenium.webdriver import Chrome
driver = Chrome()
```

Start a chrome instance

Command template
('POST', '/session')

Request url
http://127.0.0.1:54444/session

Request Body

```
{
  'desiredCapabilities': {
      'goog:chromeOptions': {
          'extensions': [],
          'args': []
      },
      'version': '',
      'browserName': 'chrome',
      'platform': 'ANY'
  },
```

Start a chrome instance


```
{
   'sessionId': 'b742d49f729c928422510619e68344c5',
   'status': 0,
   'value': {
      'locationContextEnabled': True,
      'browserName': 'chrome',
      'applicationCacheEnabled': False,
      'nativeEvents': True,
      'webStorageEnabled': True,
      'mobileEmulationEnabled': False,
      'unexpectedAlertBehaviour': '',
      'javascriptEnabled': True,
      'cssSelectorsEnabled': True,
      'handlesAlerts': True,
      'acceptSslCerts': False,
      'acceptInsecureCerts': False,
      'version': '67.0.3396.99',
      'takesScreenshot': True,
      'takesHeapSnapshot': True,
      'rotatable': False,
      'databaseEnabled': False,
      'chrome': {
         'userDataDir': '/tmp/.org.chromium.Chromium.S0UVCW',
         'chromedriverVersion': '2.37.544315
(730aa6a5fdba159ac9f4c1e8cbc59bf1b5ce12b7)'
      },
      'platform': 'Linux',
      'pageLoadStrategy': 'normal',
      'browserConnectionEnabled': False,
      'setWindowRect': True,
      'networkConnectionEnabled': False,
      'hasTouchScreen': False
   }
}
```


return {'success': 0, 'value': None, 'sessionId': 'b742d49f729c928422510619e68344c5'}

Maximize windows

```
from selenium.webdriver import Chrome
driver = Chrome()
driver.maximize_window()
```

Maximize windows

Command template
('**POST**', '**/session/$sessionId/window/$windowHandle/maximize**')

Request URL
http://127.0.0.1:54444/session/f814e98e23e15bcf189169f55913b24f/window/current/maximize

Request Body
{"windowHandle": "current", "sessionId": "f814e98e23e15bcf189169f55913b24f"}

Response
{'status': 0, 'sessionId': 'f814e98e23e15bcf189169f55913b24f', 'value': None}

Find element

```
from selenium.webdriver import Chrome
from selenium.webdriver.common.by import By


driver = Chrome()
driver.maximize_window()
driver.get('https://www.google.com/')
driver.find_element(By.ID, 'lst-ib')
```

Find element

Command template
('**POST**', '**/session/$sessionId/element**')

Request body
{"value": "lst-ib", "using": "id", "sessionId": "f814e98e23e15bcf189169f55913b24f"}

Request URL
http://127.0.0.1:54444/session/f814e98e23e15bcf189169f55913b24f/element

Response
{'status': 0, 'sessionId': 'f814e98e23e15bcf189169f55913b24f', 'value': {'ELEMENT': '0.0195760702552567 6-1'}}


FUTURE

W3C Webdriver

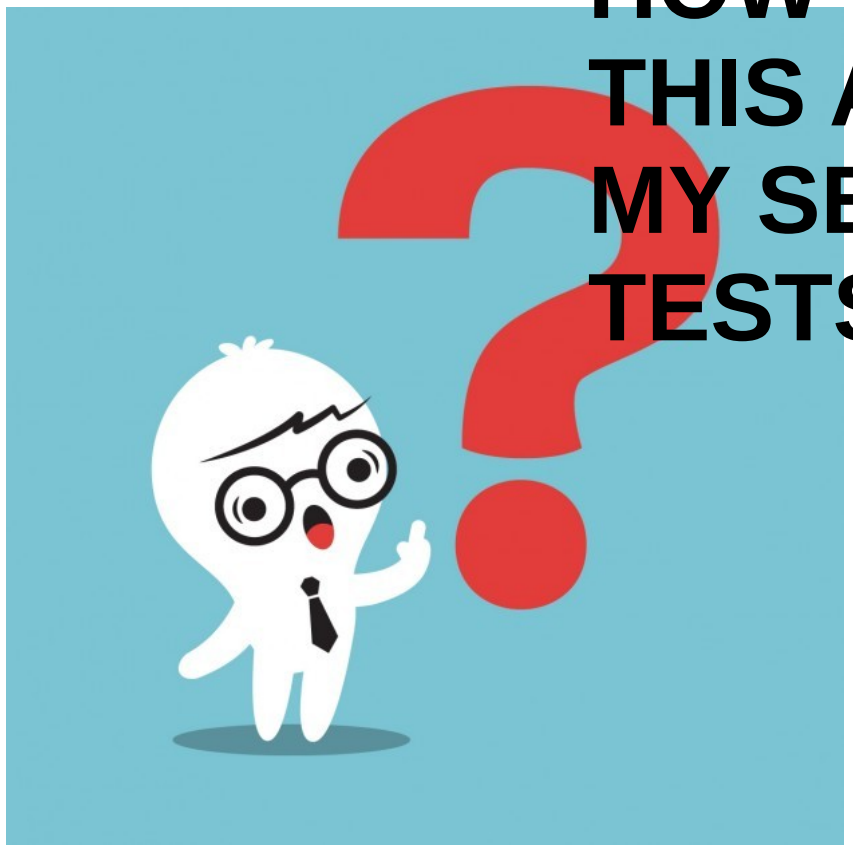
https://www.w3.org/TR/webdriver/

W3C Webdriver

	Core	WebDriver	W3C WebDriver
1.0			
2.0			
3.0			
4.0			

source: https://www.quora.com/What-are-some-new-features-of-Selenium

W3C Webdriver



HOW DOES THIS AFFECT MY SELENIUM TESTS?

W3C Webdriver

HOW TO TRY THIS NEW W3C PROTOCOLS?

1. Ensure you are using a Selenium language **binding version that supports W3C.**

2. Use one of these W3C compatible browser versions:

Safari - v.11+

Firefox - v.53+

Chrome - v.61+

Internet Explorer 11

Microsoft Edge

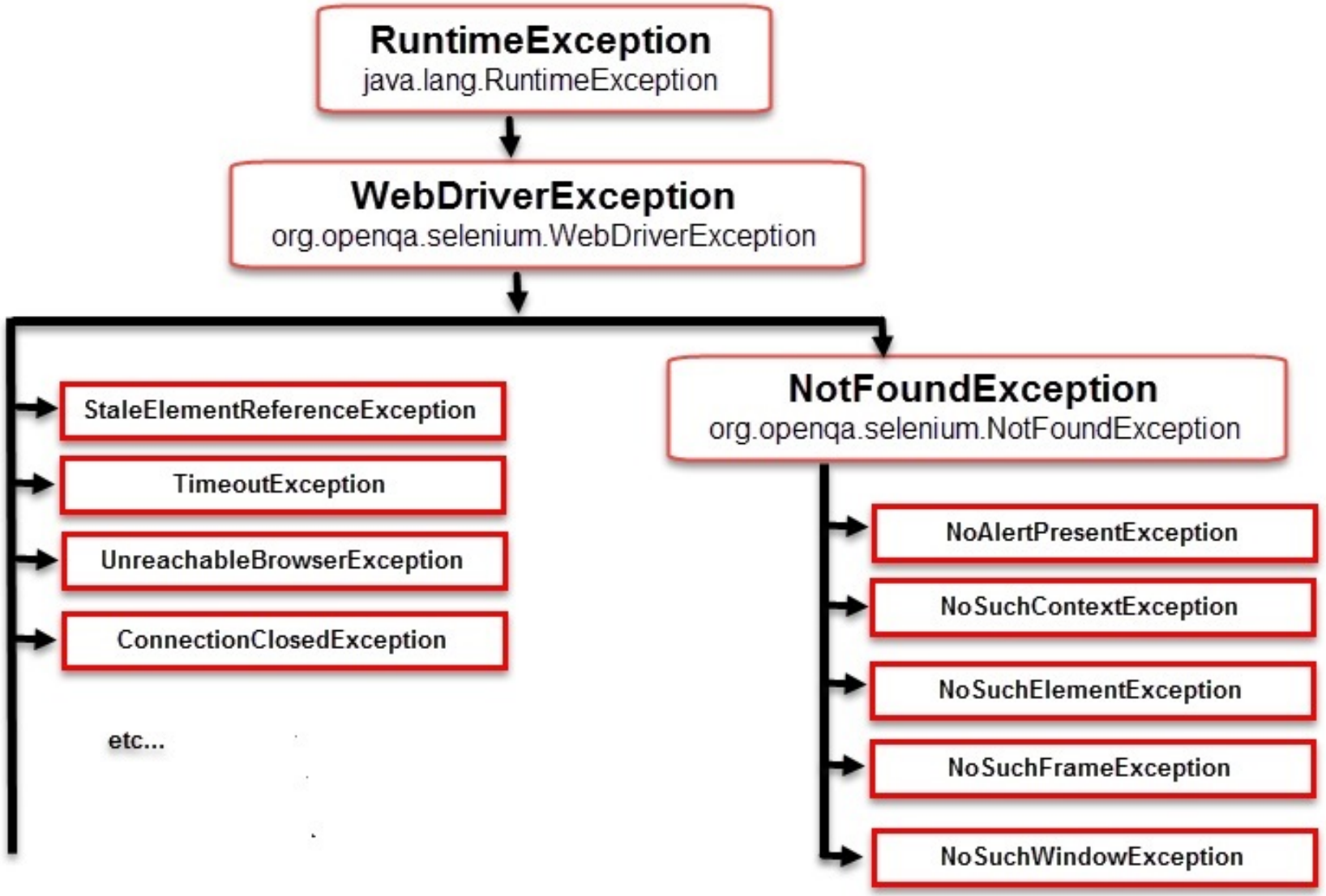
3. chrome_option.add_experimental_option('w3c', True)

Agenda

- History
- JSON Wire Protocol
- Selenium - The future
- **Selenium WebDriver Exceptions**
- **Wait Strategies**
- **Effective Page Objects**
- **Javascript Executor**
- File Downloads
- Super Grid
- AI test automation
- Q&A

WEBDRIVER EXCEPTIONS

Selenium Exceptions



https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/package-tree.html

Common Selenium Exceptions

NoSuchElementException

```
Error
Traceback (most recent call last):
  File "C:\Users\Administrator\PycharmProjects\demoProject\Google_Search_notClickable.py", line 19, in test_google_example
    self.driver.find_element_by_name('qqq').send_keys("Eiffel tower")
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 495, in find_element_by_name
    return self.find_element(by=By.NAME, value=name)
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 966, in find_element
    'value': value})['value']
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 320, in execute
    self.error_handler.check_response(response)
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\errorhandler.py", line 242, in check_response
    raise exception_class(message, screen, stacktrace)
selenium.common.exceptions.NoSuchElementException: Message: Unable to locate element: [name="qqq"]
```

Common Selenium Exceptions

NoSuchElementException

Reason:

- The locator of Element is incorrect
- The element does not present in DOM
- You had tried to find the element before it was presented
- Element may have been inside an iframe

Common Selenium Exceptions

ElementNotVisibleException

```
Error
Traceback (most recent call last):
  File "C:\Users\Administrator\PycharmProjects\demoProject\NotVisible_Exception.py", line 22, in test_not_visible
    elem.click()
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webelement.py", line 80, in click
    self._execute(Command.CLICK_ELEMENT)
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webelement.py", line 628, in _execute
    return self._parent.execute(command, params)
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 320, in execute
    self.error_handler.check_response(response)
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\errorhandler.py", line 242, in check_response
    raise exception_class(message, screen, stacktrace)
selenium.common.exceptions.ElementNotVisibleException: Message: element not visible
  (Session info: chrome=68.0.3440.106)
  (Driver info: chromedriver=2.41.578737 (49da6702b16031c40d63e5618de03a32ff6c197e),platform=Windows NT 10.0.17134 x86_64)
```

Common Selenium Exceptions

ElementNotVisibleException

Reason:

- The element is **present on DOM** but **not visible**
  - Element is hidden
  - Element has not finished rendering

- The element is **visible** but **not in your current view.**

Common Selenium Exceptions

ElementClickInterceptedException

```
Error
Traceback (most recent call last):
  File "C:\Users\Administrator\PycharmProjects\demoProject\Google_Search_notClickable.py", line 18, in test_google_example
    self.driver.find_element_by_name('btnK').click()
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webelement.py", line 80, in click
    self._execute(Command.CLICK_ELEMENT)
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webelement.py", line 628, in _execute
    return self._parent.execute(command, params)
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 320, in execute
    self.error_handler.check_response(response)
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\errorhandler.py", line 242, in check_response
    raise exception_class(message, screen, stacktrace)
selenium.common.exceptions.ElementClickInterceptedException: Message: Element <input name="btnK" type="submit"> is not clickable at point (562,411)
\class="sbqs_c"> obscures it
```

Common Selenium Exceptions

WebDriver exception (Element is not clickable at point)

```
Error
Traceback (most recent call last):
  File "C:\Users\Administrator\PycharmProjects\demoProject\test.py", line 33, in test_example2
    self.driver.find_element(By.CSS_SELECTOR, '[name$="PromoCode"]').click().send_keys('fdsafaf')
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webelement.py", line 80, in click
    self._execute(Command.CLICK_ELEMENT)
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webelement.py", line 628, in _execute
    return self._parent.execute(command, params)
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 320, in execute
    self.error_handler.check_response(response)
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\errorhandler.py", line 242, in check_response
    raise exception_class(message, screen, stacktrace)
selenium.common.exceptions.WebDriverException: Message: unknown error: Element <input name="ctl00$UcRightV31$TxtPromoCode" type="text" maxlength="25"
id="ctl00_UcRightV31_TxtPromoCode" onkeyup="valid(this,'special')" style="width:163px;"> is not clickable at point (870, 397). Other element would receive the click: <li
class="select2-results__option" id="select2-selectOrigin-result-epp4-HAN" role="treeitem" aria-selected="false" data-select2-id="select2-selectOrigin-result-epp4-HAN">
</li>
  (Session info: chrome=69.0.3497.100)
  (Driver info: chromedriver=2.41.578737 (49da6702b16031c40d63e5618de03a32ff6c197e),platform=Windows NT 10.0.17134 x86_64)
```

Common Selenium Exceptions

Reason:

- Target Element was obscured by another Element.

- The element is in semi-rendered state.

Common Selenium Exceptions

StaleElementReferenceException

```
Error
Traceback (most recent call last):
  File "C:\Users\Administrator\PycharmProjects\demoProject\test.py", line 42, in test_example2
    elem.clear()
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webelement.py", line 96, in clear
    self._execute(Command.CLEAR_ELEMENT)
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webelement.py", line 629, in _execute
    return self._parent.execute(command, params)
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\webdriver.py", line 320, in execute
    self.error_handler.check_response(response)
  File "C:\Program Files\Python37\lib\site-packages\selenium\webdriver\remote\errorhandler.py", line 242, in
 check_response
```

```
selenium.common.exceptions.StaleElementReferenceException: Message: The element reference of <input class="form-control
 header-search-input jump-to-field js-jump-to-field js-site-search-focus js-navigation-enable jump-to-field-active
jump-to-dropdown-visible" name="q" type="text"> is stale; either the element is no longer attached to the DOM, it is
not in the current frame context, or the document has been refreshed
```

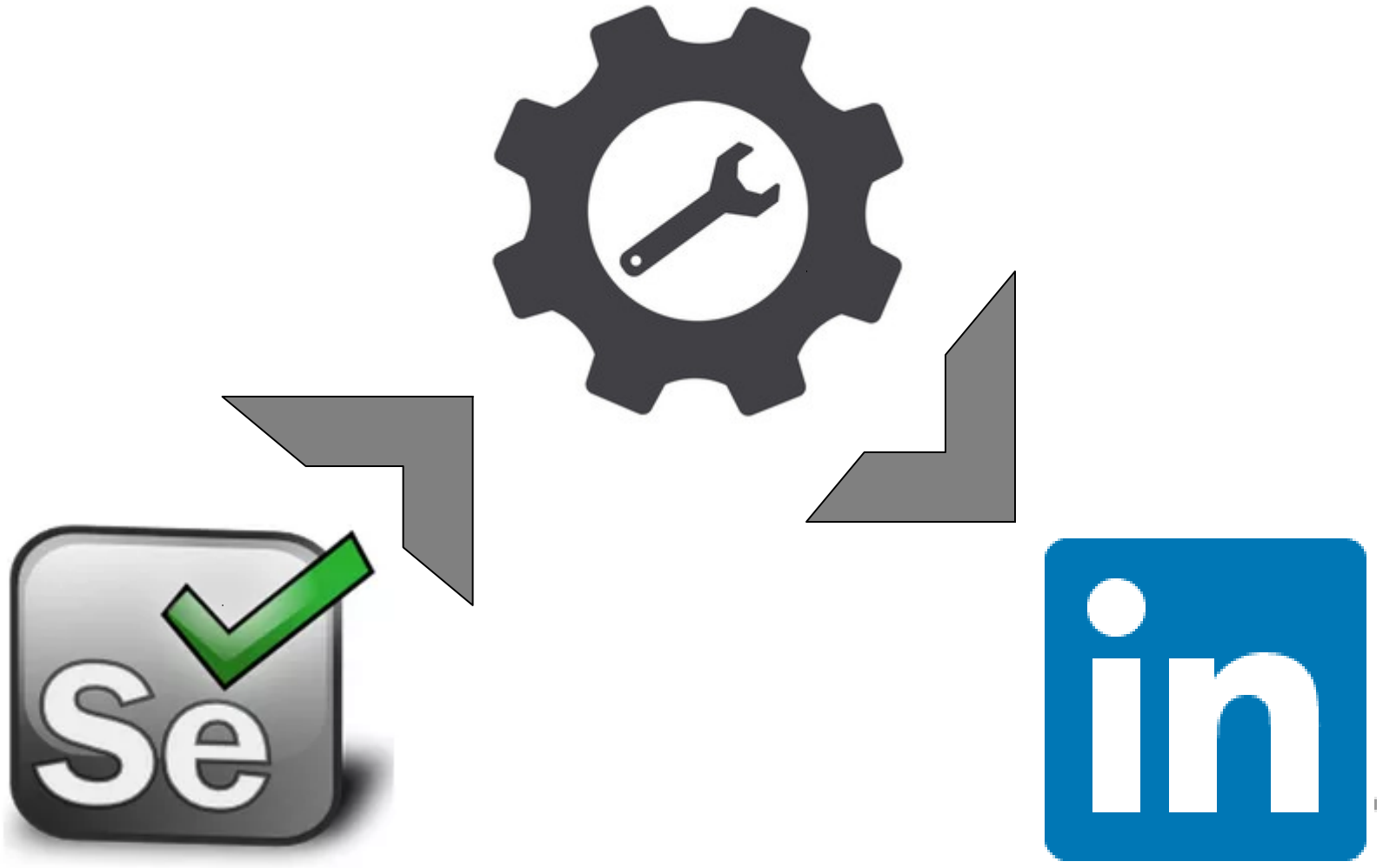
Common Selenium Exceptions

StaleElementReferenceException

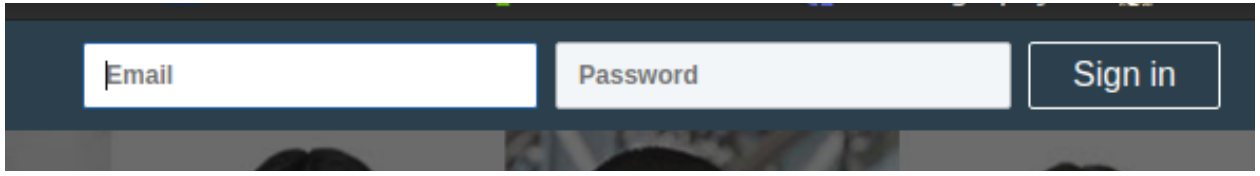
Reason:

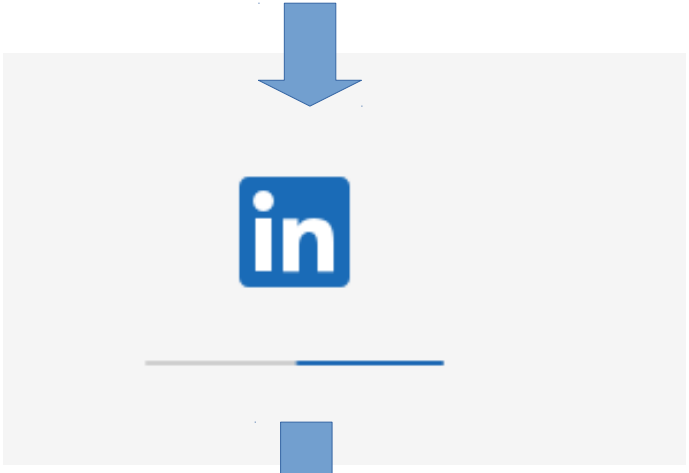
- Element is no longer presented in DOM.
- Element has been re-rendered.
- You are in wrong frame/tab.

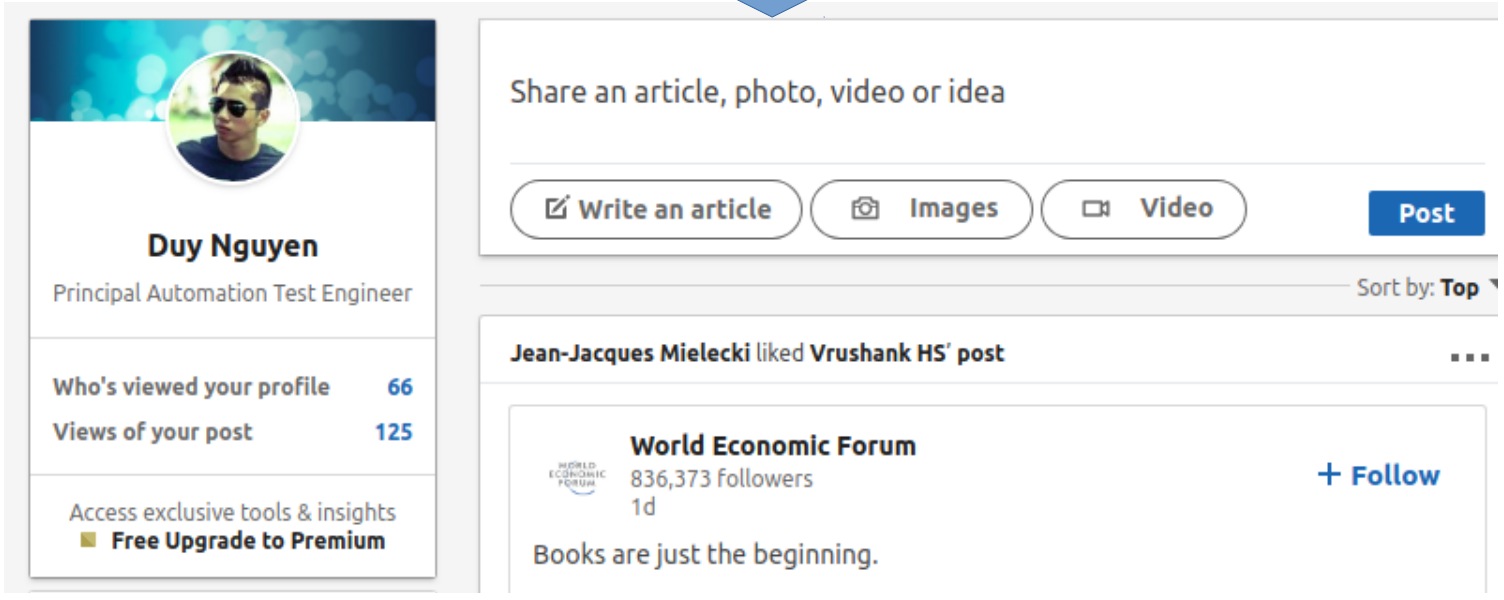
Wait Strategies



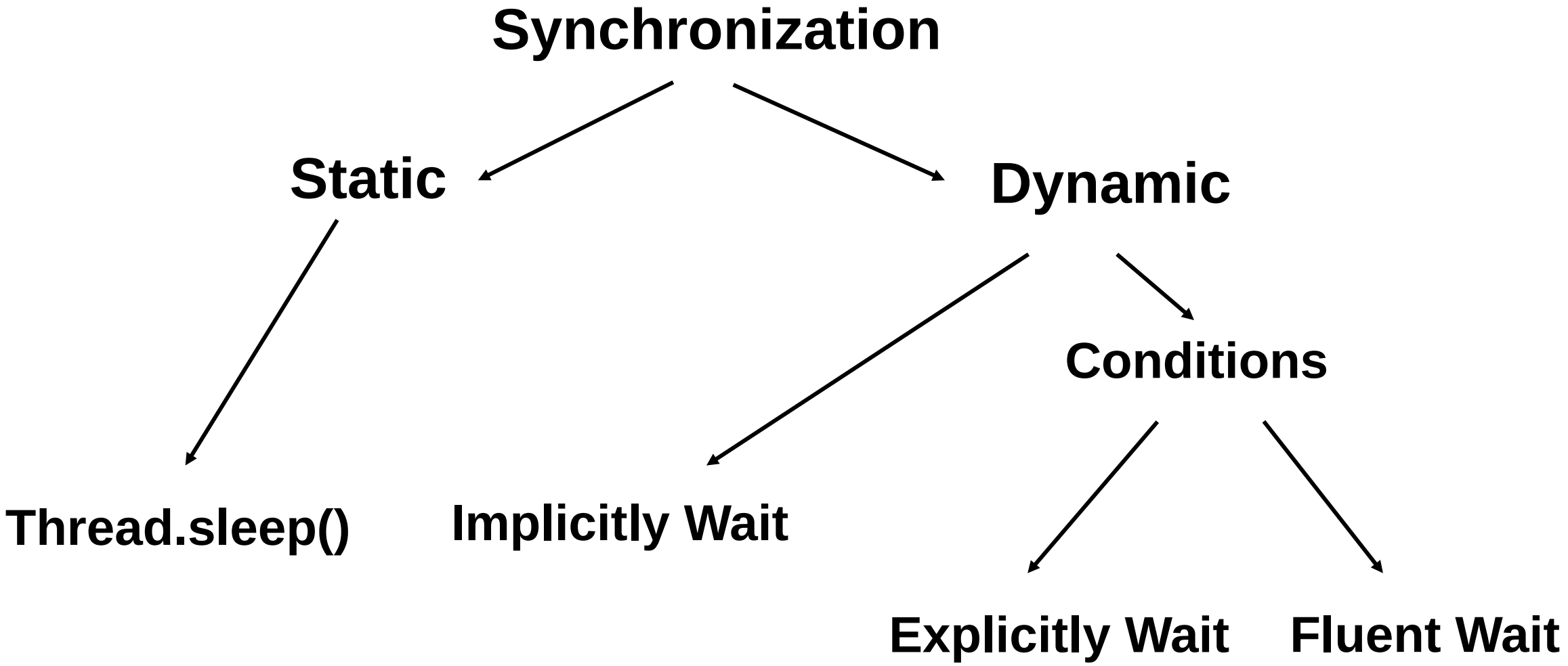
LinkedIn example







Wait Strategies



Wait Strategies

Implicitly Wait

```
# Timeouts
def implicitly_wait(self, time_to_wait):
    """
    Sets a sticky timeout to implicitly wait for an element to be found,
       or a command to complete. This method only needs to be called one
       time per session. To set the timeout for calls to
       execute_async_script, see set_script_timeout.

    :Args:
     - time_to_wait: Amount of time to wait (in seconds)

    :Usage:
        driver.implicitly_wait(30)
    """
    if self.w3c:
        self.execute(Command.SET_TIMEOUTS, {
            'implicit': int(float(time_to_wait) * 1000)})
    else:
        self.execute(Command.IMPLICIT_WAIT, {
            'ms': float(time_to_wait) * 1000})
```

Wait Strategies

Implicitly Wait

Problems:

- Not Reliable
- Poor Performance
- Not Flexible

Wait Strategies

Explicitly Wait

```
1  wait = WebDriverWait(self.driver, 5)
2  self.driver.get("https://www.linkedin.com/")
3  login_email = wait.until(EC.visibility_of_element_located((By.CLASS_NAME, 'login-email')))
4  login_email.send_keys(LINKEDIN_ACCOUNT['Thach Hoang']['USER_NAME'])
5
6  login_password = wait.until(EC.visibility_of_element_located((By.CLASS_NAME, 'login-password')))
7  login_password.send_keys(LINKEDIN_ACCOUNT['Thach Hoang']['PASSWORD'])
8
9  login_submit = wait.until(EC.element_to_be_clickable((By.ID, 'login-submit')))
10 login_submit.click()
11
12 wait = WebDriverWait(self.driver, 10)
13 start_time = time.time()
14 wait.until(EC.invisibility_of_element_located((By.CSS_SELECTOR, '.initial-load-animation:not(.fade-load)')))
15 print("End test")
16 print(time.time() - start_time)
```

Wait Strategies

Explicitly Wait

```
def until(self, method, message=''):
    """Calls the method provided with the driver as an argument until the \
    return value is not False."""
    screen = None
    stacktrace = None

    end_time = time.time() + self._timeout
    while True:
        try:
            value = method(self._driver)
            if value:
                return value
        except self._ignored_exceptions as exc:
            screen = getattr(exc, 'screen', None)
            stacktrace = getattr(exc, 'stacktrace', None)
        time.sleep(self._poll)
        if time.time() > end_time:
            break
    raise TimeoutException(message, screen, stacktrace)
```

Wait Strategies

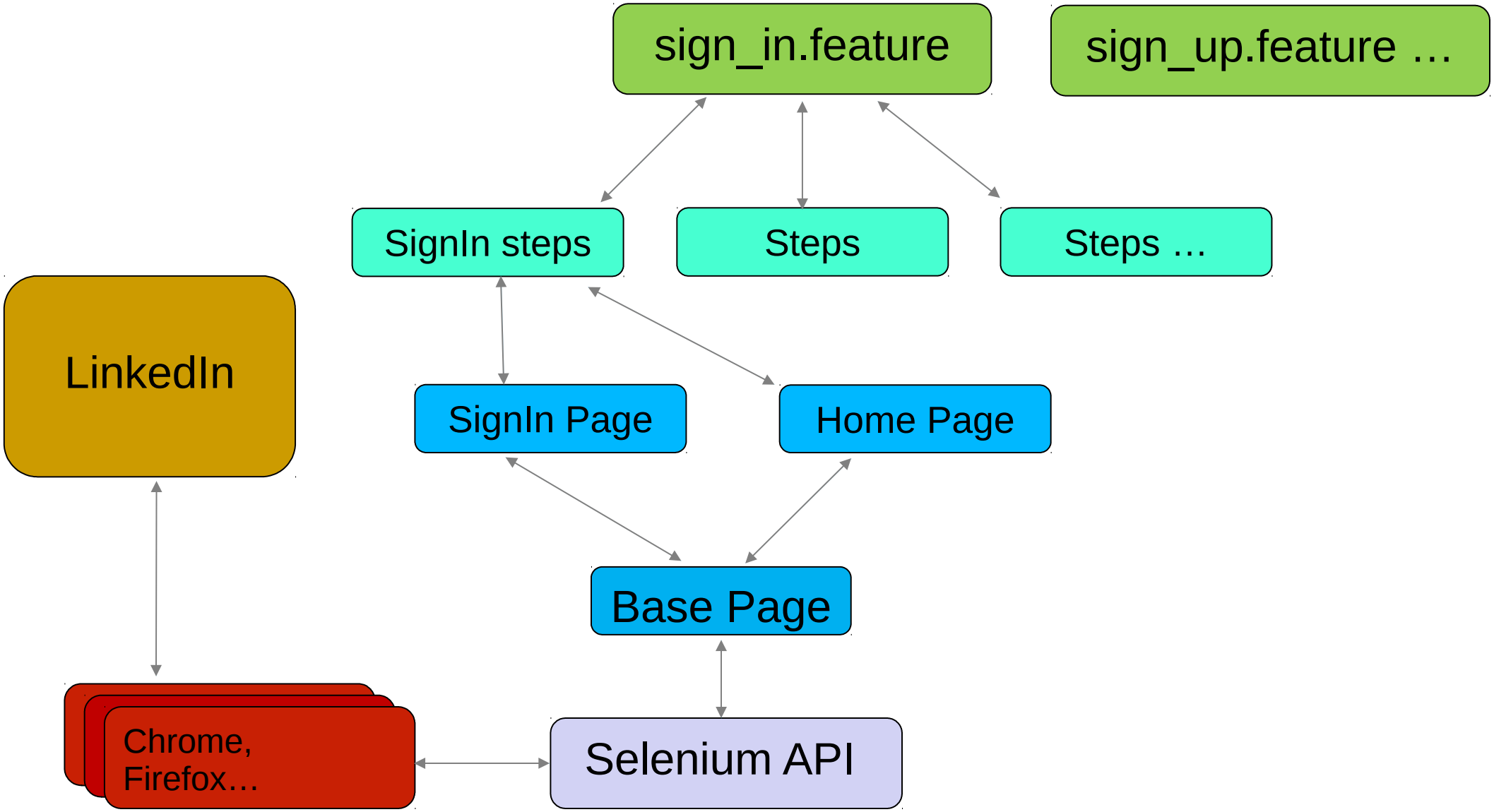
Fluent Wait

```
// Fluent Wait
Wait<WebDriver> fluentWait = new FluentWait<WebDriver>(driver)

  .withTimeout(30, TimeUnit.SECONDS)

  .pollingEvery(1, TimeUnit.SECONDS)

  .ignoring(NoSuchElementException.class);

WebElement content = fluentWait.until(new Function<WebDriver, WebElement>() {
 public WebElement apply(WebDriver driver) {
  return driver.findElement(By.xpath("//h4[text()='Hello World!']"));
 }
});

System.out.println(content.getText());
}
```

Wait Strategies

Using Implicitly and Fluent/Explicitly Wait Together

Effective Page Objects



Effective Page Objects

```
class BasePage:

    def __init__(self, webdriver):
        self.driver = webdriver

    def type_text(self, tuple_selector, value, timeout=constants.SELENIUM_TIMEOUT_SECONDS):
        elem = self.wait_for_visibility_of_element(tuple_selector, timeout)
        elem.send_keys(value)

    def get_element_text(self, tuple_selector, timeout=constants.SELENIUM_TIMEOUT_SECONDS):
        element = self.wait_for_visibility_of_element(tuple_selector, timeout)
        return element.text

    def click_element(self,tuple_selector, timeout=constants.SELENIUM_TIMEOUT_SECONDS):
        elem = self.wait_for_element_to_be_clickable(tuple_selector, timeout)
        elem.click()

    def wait_for_visibility_of_element(self, tuple_selector, timeout=constants.SELENIUM_TIMEOUT_SECONDS):
        wait = WebDriverWait(self.driver, timeout)
        return wait.until(EC.visibility_of_element_located(tuple_selector))

    def wait_for_invisibility_of_element_located(self, tuple_selector, timeout=constants.SELENIUM_TIMEOUT_SECONDS):
        wait = WebDriverWait(self.driver, timeout)
        return wait.until(EC.invisibility_of_element_located(tuple_selector))

    def wait_for_element_to_be_clickable(self, tuple_selector, timeout=constants.SELENIUM_TIMEOUT_SECONDS):
        wait = WebDriverWait(self.driver, timeout)
        return wait.until(EC.element_to_be_clickable(tuple_selector))
```

Effective Page Objects

```
class SignIn(BasePage):

    def enter_user_name(self, value):
        self.type_text(LOCATOR['USER_NAME_BOX'], value)

    def enter_user_password(self, value):
        self.type_text(LOCATOR['PASSWORD_BOX'], value)


    def click_sign_in_button(self):
        self.click_element(LOCATOR['SIGN_IN_BTN'])
```

```
class HomePage(BasePage):

    def get_profile_name(self):
        return self.get_element_text(LOCATOR['PROFILE_NAME'])
```

JavaScript Executor

```
element = self.wait_element_exist(selector)
driver.execute_script("arguments[0].scrollIntoView(true);", element)
driver.execute_script("arguments[0].click();", element)

driver.execute_script("console.log('I logged something to the Javascript
console');");

driver.executeScript("return document.title;");
```

JavaScript Executor

```
token = self.driver.execute_script("return
window.localStorage.getItem(arguments[0]);",
    'adal.access.token.key{client_id}'.format(client_id=client_id))
```

We can also create a function.

//Create an anonymous function that will stored and added into the global window

```
self.driver.execute_script("window.changetitle = function()
{document.title='Title is changed by function!';};"+

"window.changetitle.call();");
self.driver.execute_script("window.changetitle.call();");
```

JavaScript Executor

JavaScript Executor is not a real User behavior

Agenda

- The origins of Selenium
- JSON Wire Protocol
- Selenium - The future
- Selenium WebDriver Exceptions
- Wait Strategies
- Effective Page Objects
- Javascript Executor
- **File Downloads**
- **Super Grid**
- **AI test automation**
- Q&A

File downloads

1. You load the page with the download link
2. You find the <a> element on the page
3. You click on it

Do you really need to download that file?

How many files are you planning to download?
How big are these files?
Do you have enough disk space to hold all of these files?
Do you have network capacity to continually download these files?
What are you planning to do with the file(s) once you have downloaded them?

File downloads

You are not really checking that you can download the file, you are checking for broken links

File downloads

Use AutoIt to click on the download dialog
This is great when you are working with Windows, but our CI server is running on Linux.

Write a Java Robot class to click on the download dialog
is that dialog boxes across operating systems differ, so we will probably have to write code branches for each operating system.

Get our browser to automatically download files when we click on a link
Selenium is unaware of the download process because the browser controls it all. This injects some new problems into the mix.
How do we know when the download has completed?
Is our test going to finish and close the browser before the file download has completed?

Extend our existing code

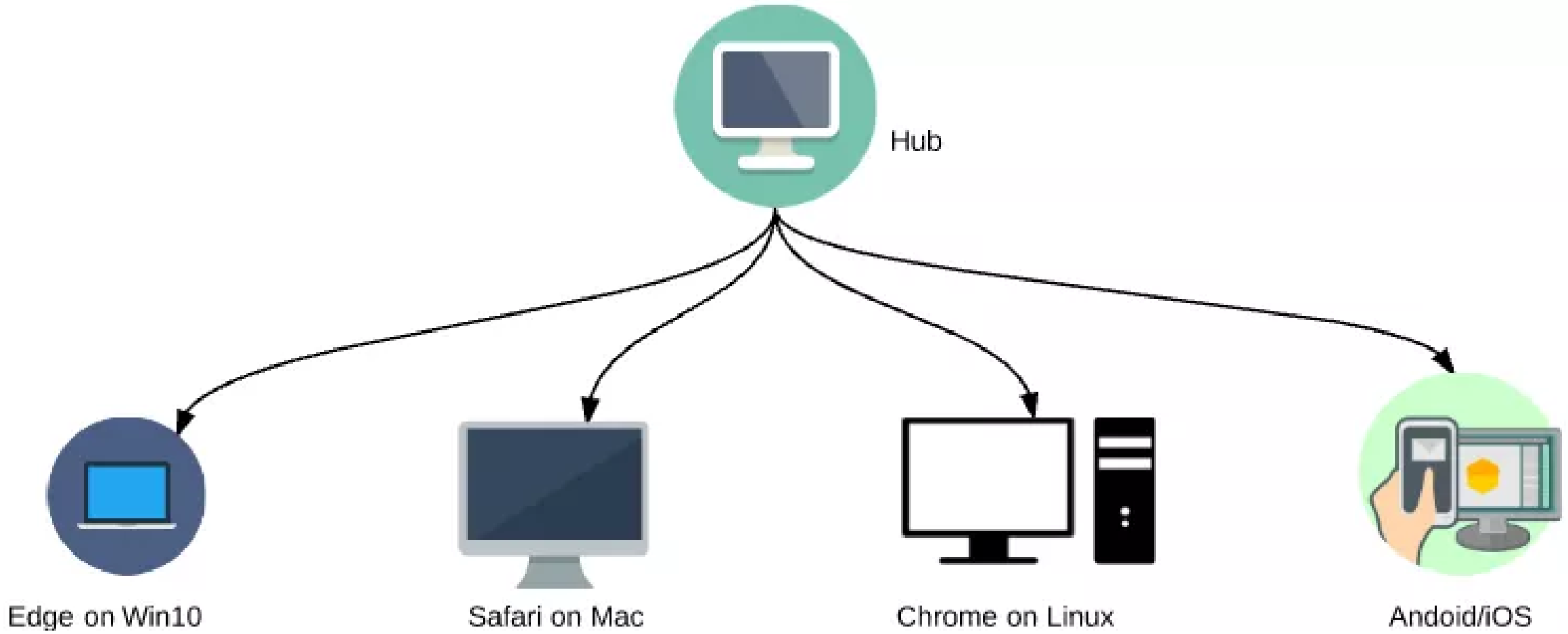
Is it the same as clicking on a link and downloading the file?

When you click on the link, your browser sends a HTTP GET request over to the web server and negotiates a connection. When it has negotiated a connection, it starts to download the file to a temporary location.

It then tells your operating system that it's downloading a file and asks what to do with it

Once you have told the operating system which filename and which download location to use, and decided if you want to overwrite any existing files, the operating system will pass this information back to the browser. The browser then copies the file it has downloaded to a temporary location, the location specified by the operating system.

Super Grid



source: http://www.testautomationguru.com/selenium-grid-setup-using-docker/

AI Test Automation

1. Do visual, automated validation UI testing

"Visual testing is a quality assurance activity that is meant to verify that the UI appears correctly to users"

2. Spidering AI

The most popular AI automation area right now is using machine learning to automatically write tests for your application by spidering.

3. Creating more reliable automated tests

How often do your tests fail due to developers making changes to your application, such as renaming a field ID?

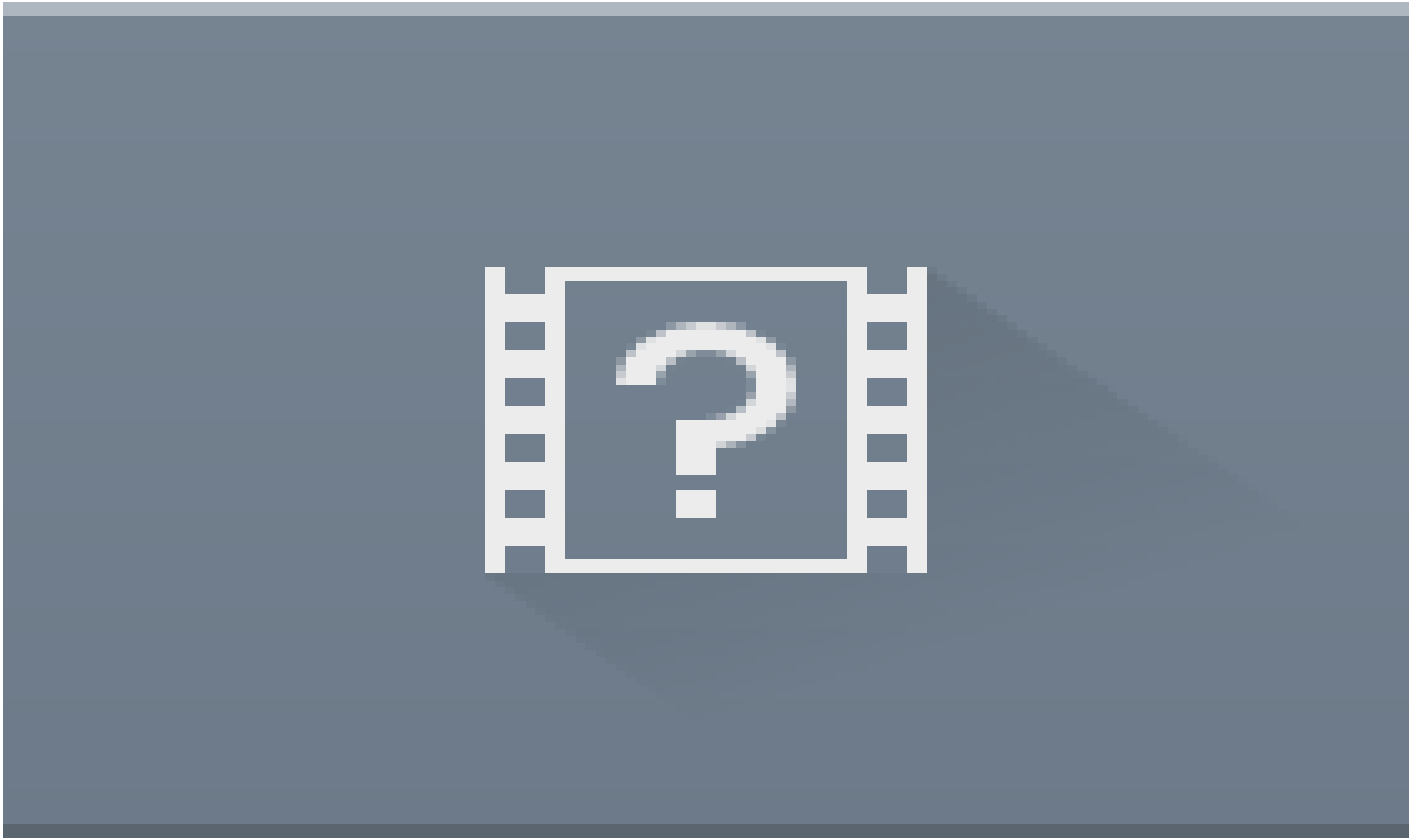
Tools can use machine learning to automatically adjust to these changes. This makes tests more maintainable and reliable.

Leveraging ML/AI-based for automated maintenance (being able to group together similar groups of changes from different pages/browsers/devices)

Modifying their comparison algorithms to be able to discern what changes are meaningful/noticeable

Being able to automatically understand which changes are more likely to be bugs vs. desired changes and prioritize diffs

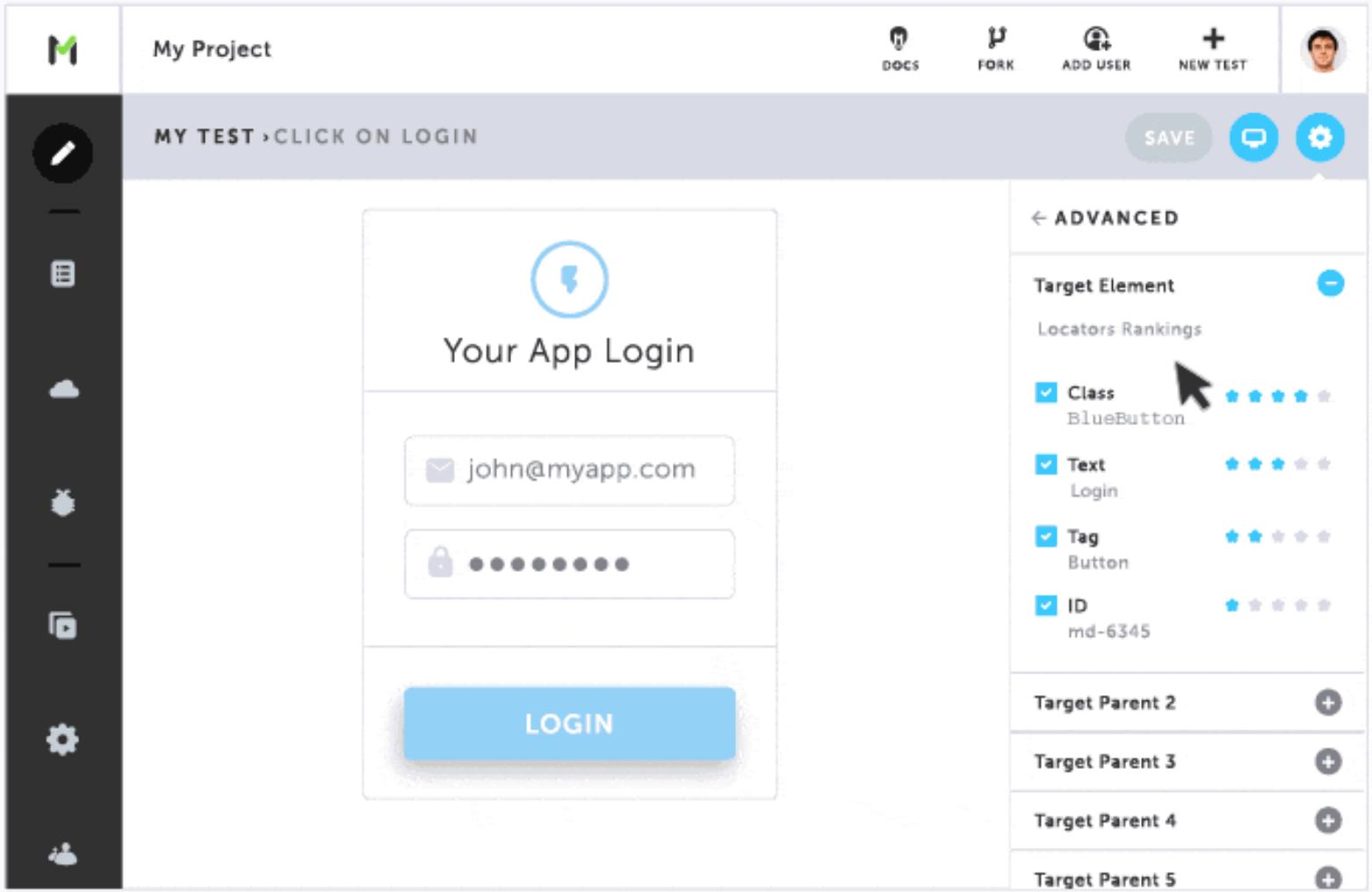
source: https://applitools.com/



source: https://applitools.com/

source: https://applitools.com/

testim



source: https://www.testim.io/

mabl

you “train” your tests to interact with your applications

Eliminates flaky tests–like the other AI-based test automation tools, Mabl can automatically detect whether elements of your application have changed, and dynamically updates the tests to compensate for those changes.

Mabl can continuously compare test results to test history to quickly detect changes and regressions, resulting in more stable releases.

Mabl helps identify and surface problems quickly, alerting you to possible impacts before they impact your customers.

source: https://www.mabl.com/

NASH TECH
The Power to Innovate

Q&A

THANK YOU

www.nashtechglobal.com