# Scaling Selenium

Simon Stewart
@shs96c
Selenium Project Lead

# Why Do We Test?

To provide confidence that the software being released to production works as intended.

# Your First Test

```
@Test
public void longAndWrong() {
  WebDriver driver = new FirefoxDriver();
  Wait<WebDriver> wait = new WebDriverWait(driver, timeOutInSeconds: 5);

  driver.get("http://localhost:8080/");
  driver.findElement(By.linkText("Sign in")).click();
  driver.findElement(By.name("email")).sendKeys(EMAIL);
  driver.findElement(By.name("password")).sendKeys(PASSWORD);
  driver.findElement(By.tagName("button")).click();

  wait.until(d -> d.findElement(By.linkText("Create a todo"))).click();

  WebElement title = driver.switchTo().activeElement();
  title.clear();
  title.sendKeys( ...keysToSend: "Write presentation");

  WebElement dueDate = driver.findElement(By.name("dueDate"));
  dueDate.clear();
  dueDate.sendKeys( ...keysToSend: "08/12/2018");
  title.click();

  driver.findElements(By.tagName("button")).stream()
      .filter(e -> e.getText().contains("Create"))
      .findFirst()
      .orElseThrow(() -> new AssertionError( detailMessage: "Cannot find create button"))
      .click();

  driver.quit();
}
```

# Fragile Locators

```
@Test
public void longAndWrong() {
  WebDriver driver = new FirefoxDriver();
  Wait<WebDriver> wait = new WebDriverWait(driver, timeOutInSeconds: 5);

  driver.get("http://localhost:8080/");
  driver.findElement(By.linkText("Sign in")).click();
  driver.findElement(By.name("email")).sendKeys(EMAIL);
  driver.findElement(By.name("password")).sendKeys(PASSWORD);
  driver.findElement(By.tagName("button")).click();

  wait.until(d -> d.findElement(By.linkText("Create a todo"))).click();

  WebElement title = driver.switchTo().activeElement();
  title.clear();
  title.sendKeys( ...keysToSend: "Write presentation");

  WebElement dueDate = driver.findElement(By.name("dueDate"));
  dueDate.clear();
  dueDate.sendKeys( ...keysToSend: "08/12/2018");
  title.click();

  driver.findElements(By.tagName("button")).stream()
      .filter(e -> e.getText().contains("Create"))
      .findFirst()
      .orElseThrow(() -> new AssertionError( detailMessage: "Cannot find create button"))
      .click();

  driver.quit();
}
```

# Grey Box Testing: Locating Elements

- Let the app help
    - Add meaningful identifiers to elements
- Let Selenium help
    - Write your own locators
    - Use JavascriptExecutor to find values.

# Waiting Just Long Enough

```java
/**
 * A generic interface for waiting until a condition is true or not null. The condition may take a
 * single argument of type .
 *
 * @param <F> the argument to pass to any function called
 */
public interface Wait<F> {

  /**
   * Implementations should wait until the condition evaluates to a value that is neither null nor
   * false. Because of this contract, the return type must not be Void.
   *
   * <p>
   * If the condition does not become true within a certain time (as defined by the implementing
   * class), this method will throw a non-specified {@link Throwable}. This is so that an
   * implementor may throw whatever is idiomatic for a given test infrastructure (e.g. JUnit4 would
   * throw {@link AssertionError}).
   *
   * @param <T> the return type of the method, which must not be Void
   * @param isTrue the parameter to pass to the {@link ExpectedCondition}
   * @return truthy value from the isTrue condition
   */
  <T> T until(Function<? super F, T> isTrue);
}
```

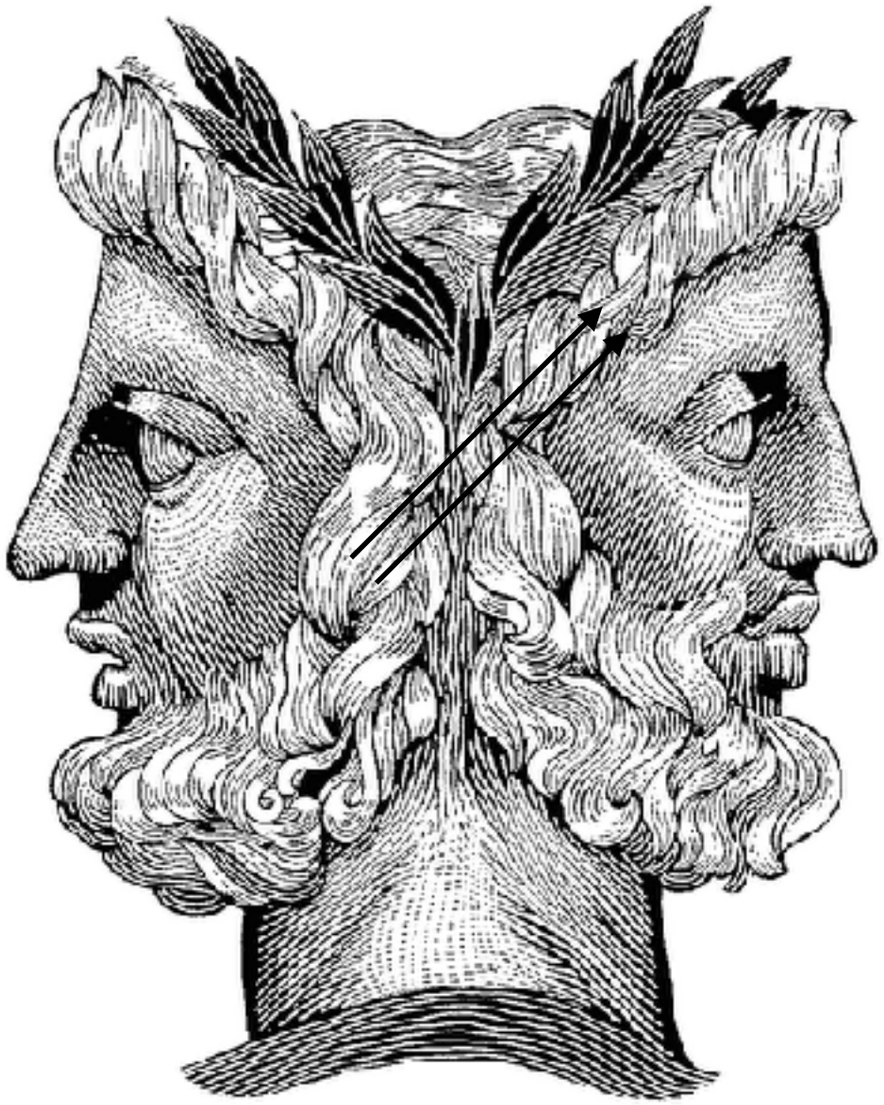
# Grey-Box Testing: Waits

- Let the libraries you use help

```
public boolean isJqueryDone(WebDriver driver) {
  JavascriptExecutor js = (JavascriptExecutor) driver;
  return (Boolean) js.executeScript( script: "return jQuery.active == 0");
}
```

- Let your app help
    - Add a JS attribute somewhere well know to indicate progress
- Let Selenium help
    - Get a reference to something that will go stale, wait until it actually goes stale before continuing to wait.

# Page Objects



**Code & Structure**

**User-facing Services**

# Enough of the Basics!

# You and Your Data

# Prepare for Parallelisation

- The `static` keyword is evil
- `Threadlocals` are also evil
- Code should be immutable and stateless

DependentTest.java - presentation - [~/src/heisenbug/presentation]

entation › javatests › org › infalbibe › heisenbug › DependentTest › DependentTest.usingWaits

SignupPage.java | WaitTest.java | DependentTest.java | MainPage.java | User.java | PageObjectTest.java

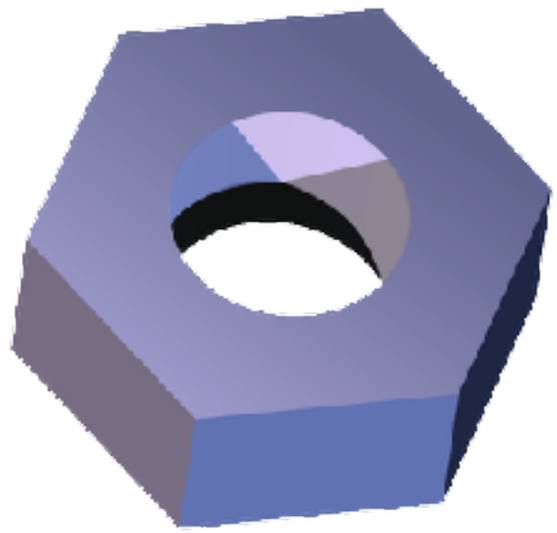
```
1   package org.infalible.heisenbug;
2
3   import ...
16
17  public class DependentTest {
18
19    private WebDriver driver;
20
21    @Test
22    public void usingWaits() {
23      driver = new FirefoxDriver();
24      driver.switchTo().window(driver.getWindowHandle());
25      WebDriver.Window window = driver.manage().window();
26      window.setPosition(new Point( x: 0,  y: 0));
27      window.setSize(new Dimension( width: 1024,  height: 768));
28      Wait<WebDriver> wait = new WebDriverWait(driver,  timeOutInSeconds: 5);
29
30      User user = new User( email: "fred@example.com",  password: "p@ssw0rd");
31      SignupPage signupPage = new SignupPage(driver, new Duration( time: 5, SECONDS)).get();
32      MainPage mainPage = signupPage.signUp(user);
33
34      Todo todo = new Todo().withTitle("Eat cheese");
35
36      Todo existing = mainPage.getTodoNamed(todo.getTitle());
37      assertNull(existing);
38
39      mainPage.create(todo);
40      existing = mainPage.getTodoNamed(todo.getTitle());
41      assertNotNull(existing);
42    }
43
44    @After
45    public void quitDriver() { driver.quit(); }
48  }
49
50
```

DependentTest › usingWaits()

Tests Failed: 0 passed, 1 failed (a minute ago)

25:38 LF UTF-8

# Now Add CI

The same tests in multiple independent locations

# Ossification of Data

## Definition of OSSIFICATION

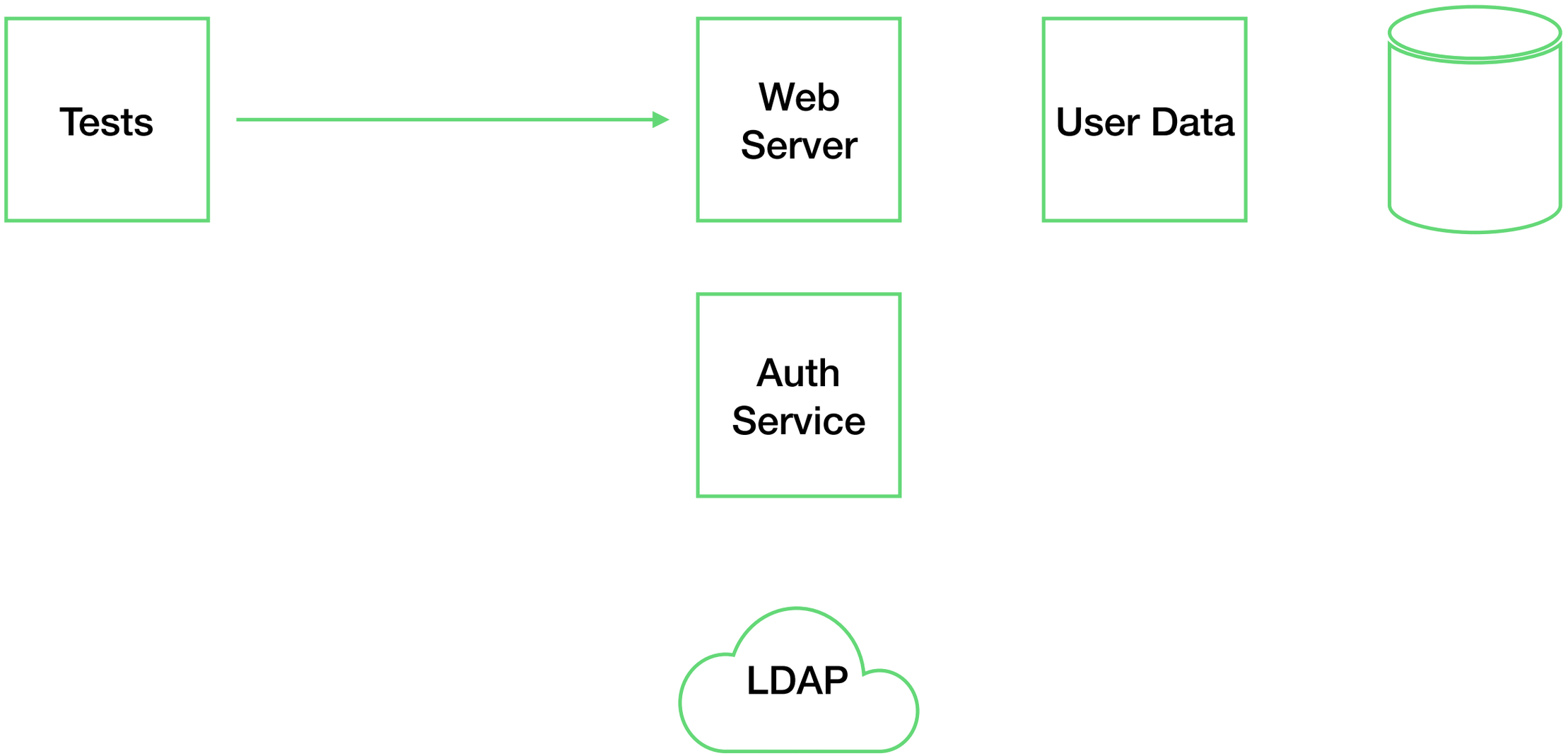
1 **a** : the natural process of bone formation
**b** : the hardening (as of muscular tissue) into a bony substance

2 : a mass or particle of ossified tissue

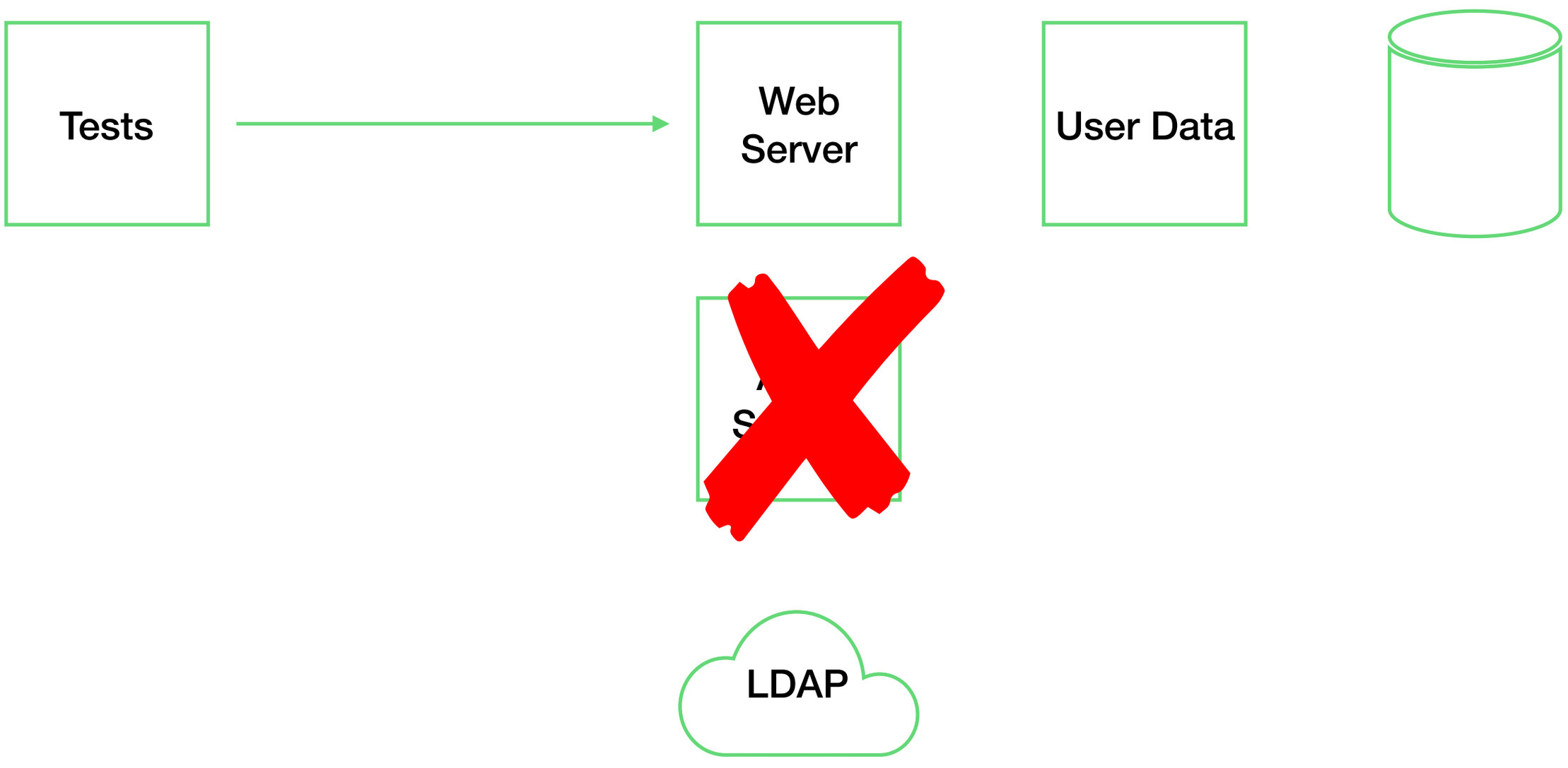
3 : a tendency toward or state of being molded into a rigid, conventional, sterile, or unimaginative condition

- The closer to production you are, the more rigid your test data becomes.
- Can set up anything you like in a dev env
- Can’t inject random data into prod

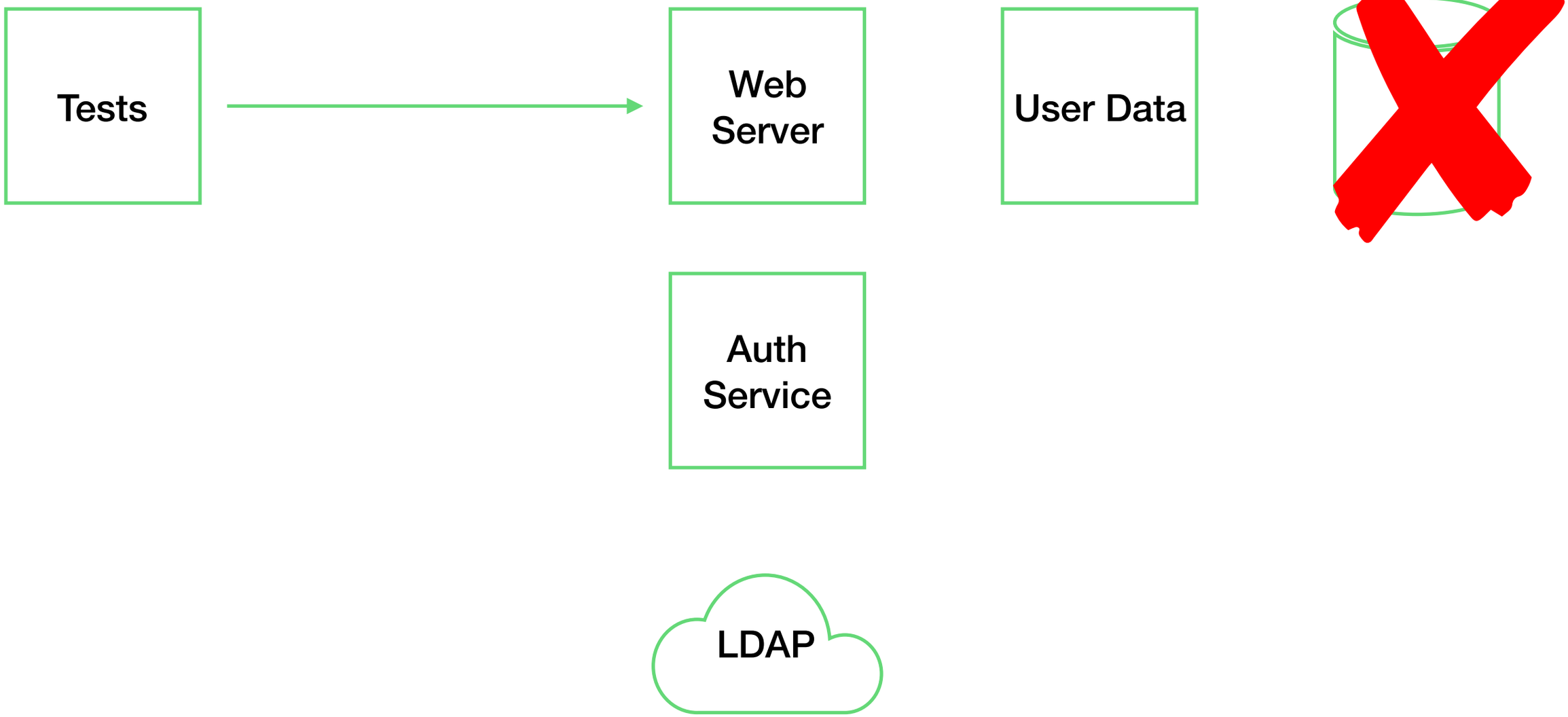
# Asserting Preconditions



# Asserting Preconditions

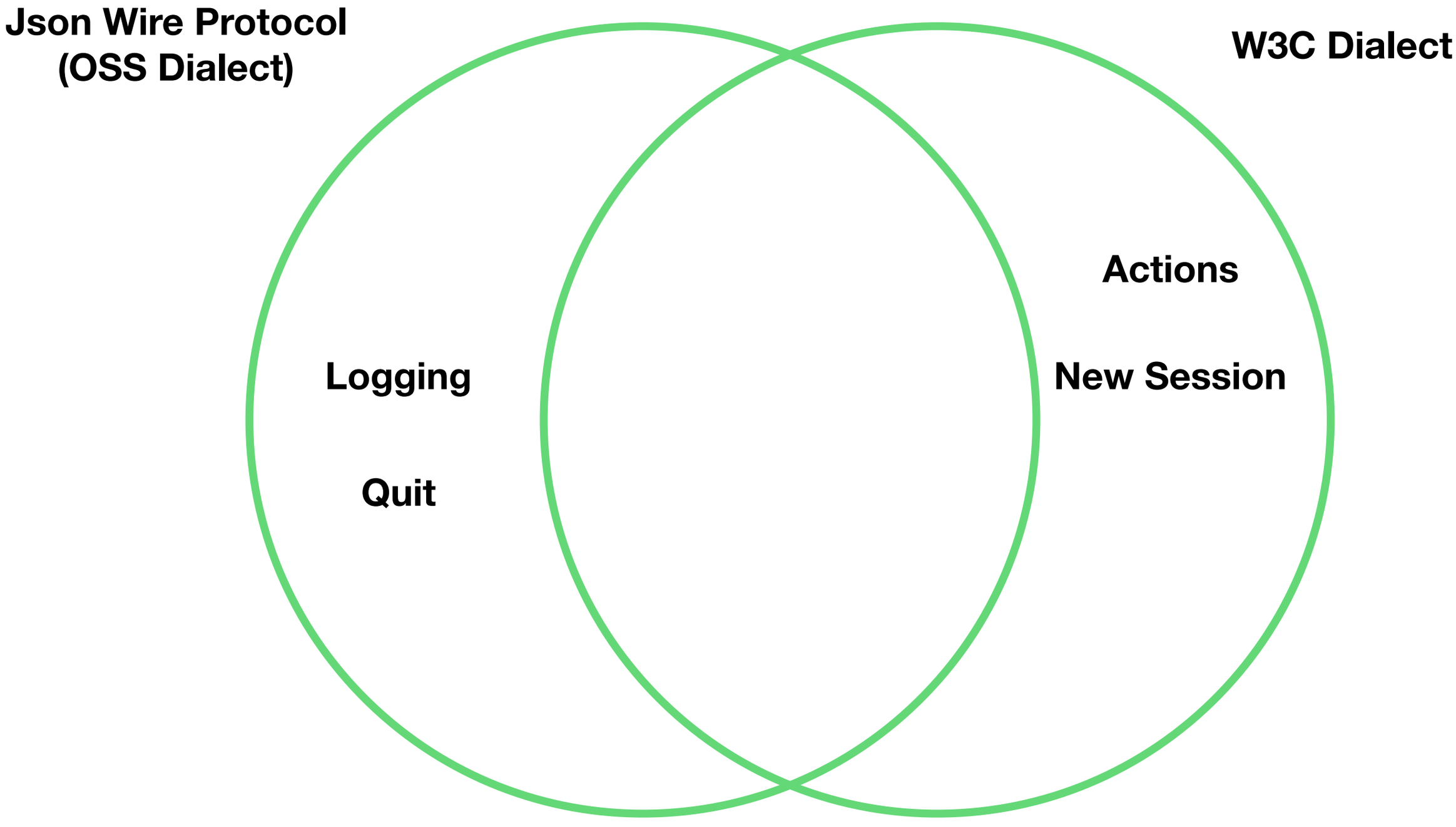


# Asserting Preconditions

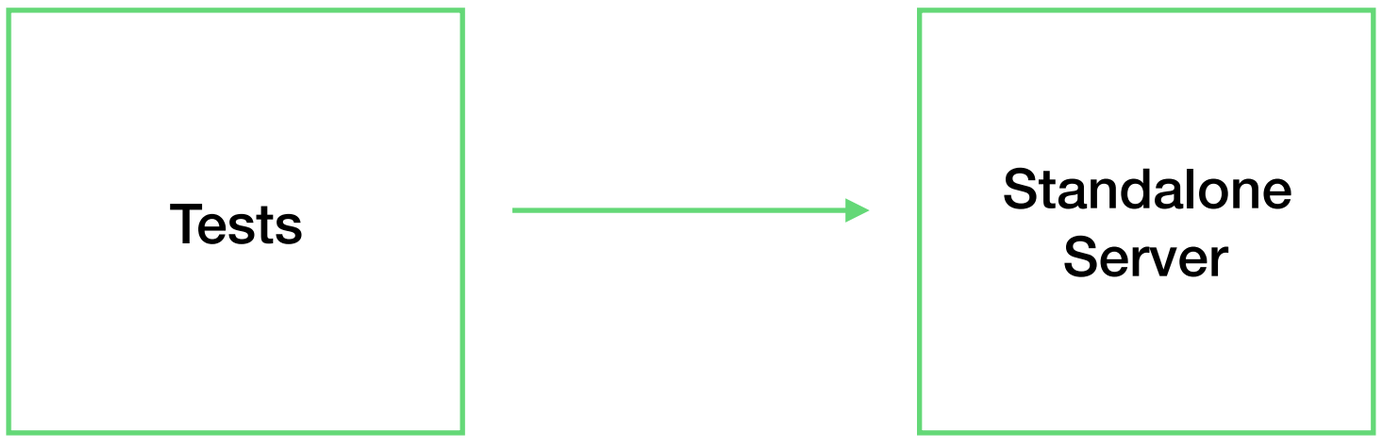


# Go Big or Go Home

# Wire Protocols



# Selenium Architecture



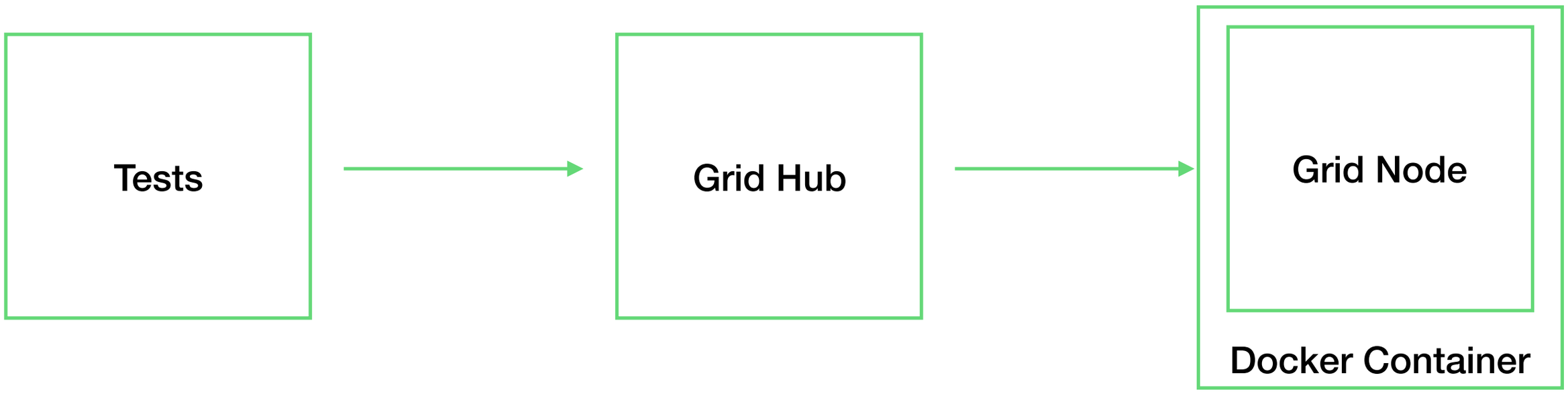
# Selenium Architecture

# Selenium Docker

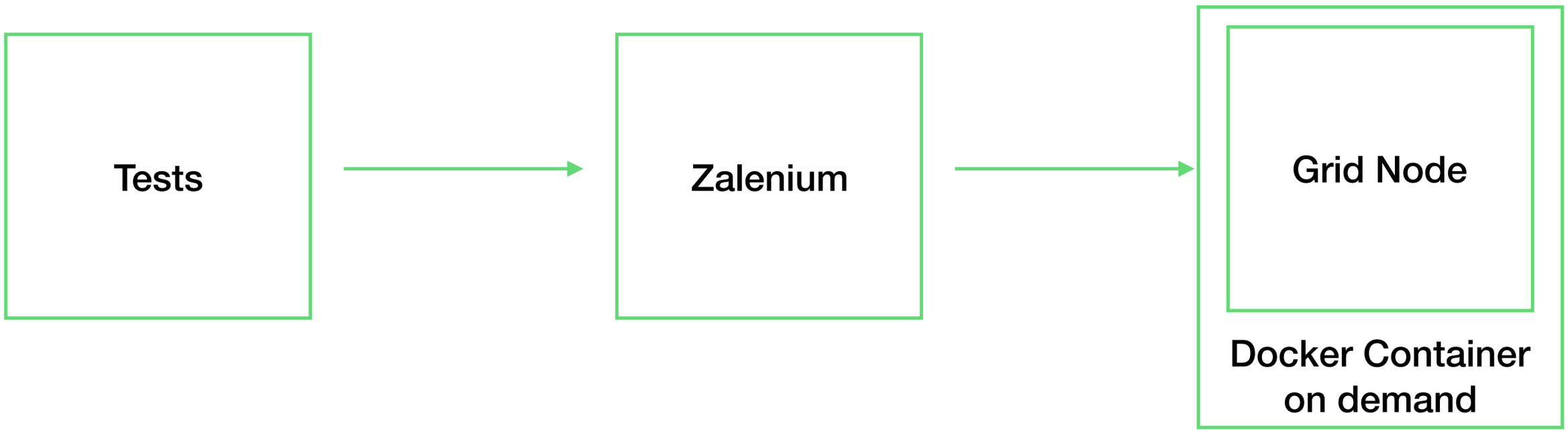
- Consistent browser images for use with testing

- Avoids problems with machines having different versions of browsers and vendor drivers (eg. geckodriver) installed

# Selenium Architecture

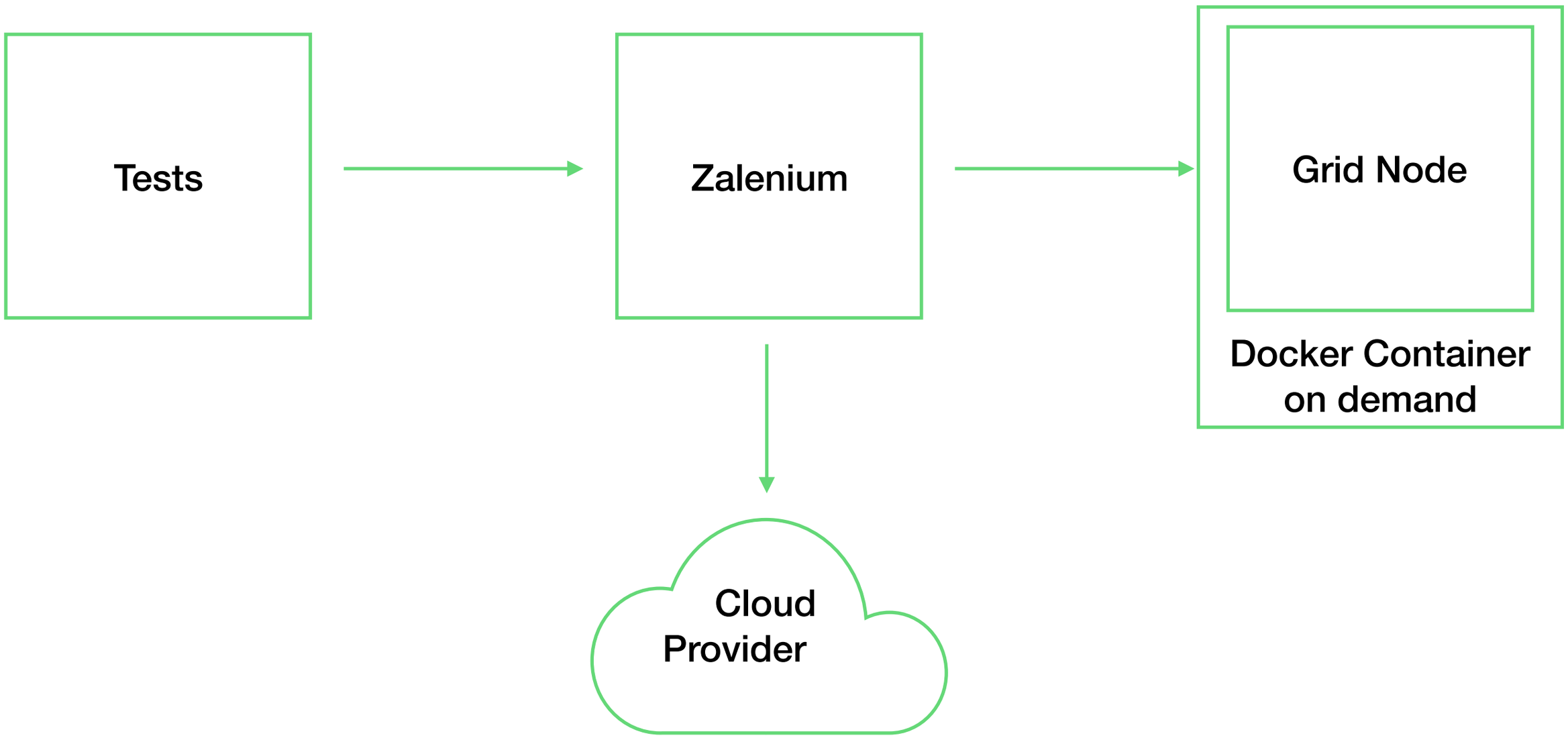


# Selenium Architecture

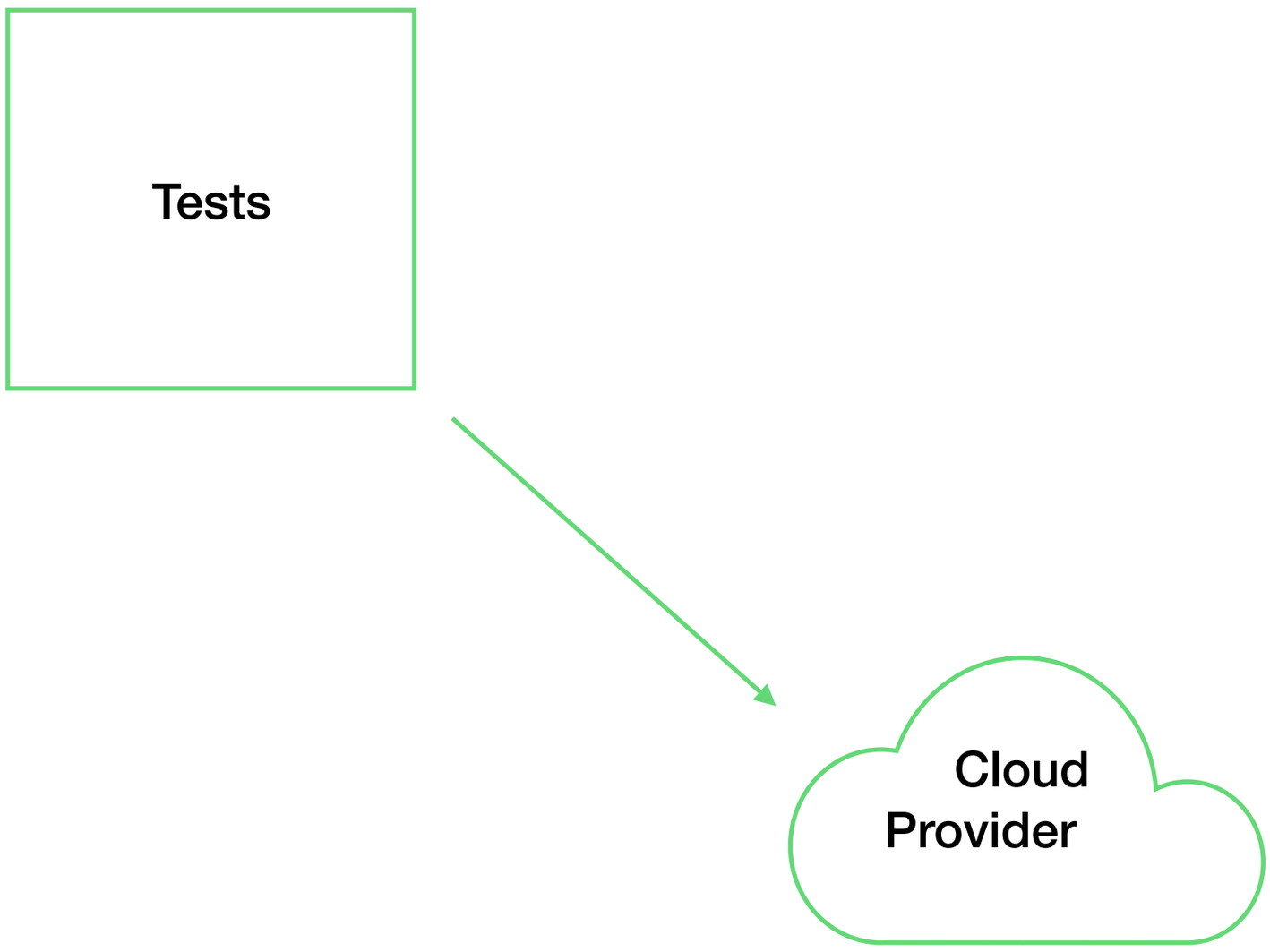



**https://github.com/zalando/zalenium**

# Selenium Architecture

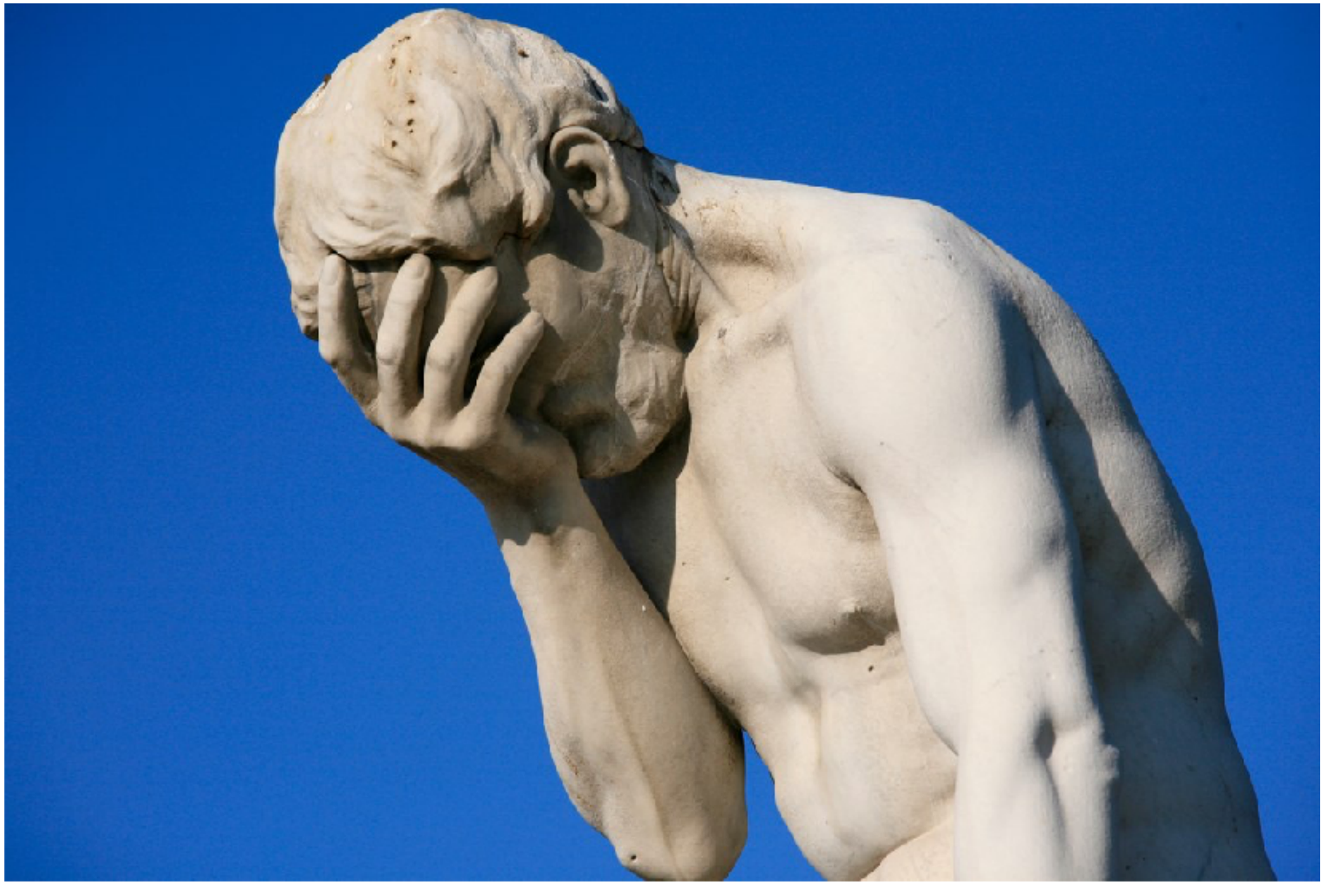


# Selenium Architecture



# Something's Got to Give

# Accidental DDoS

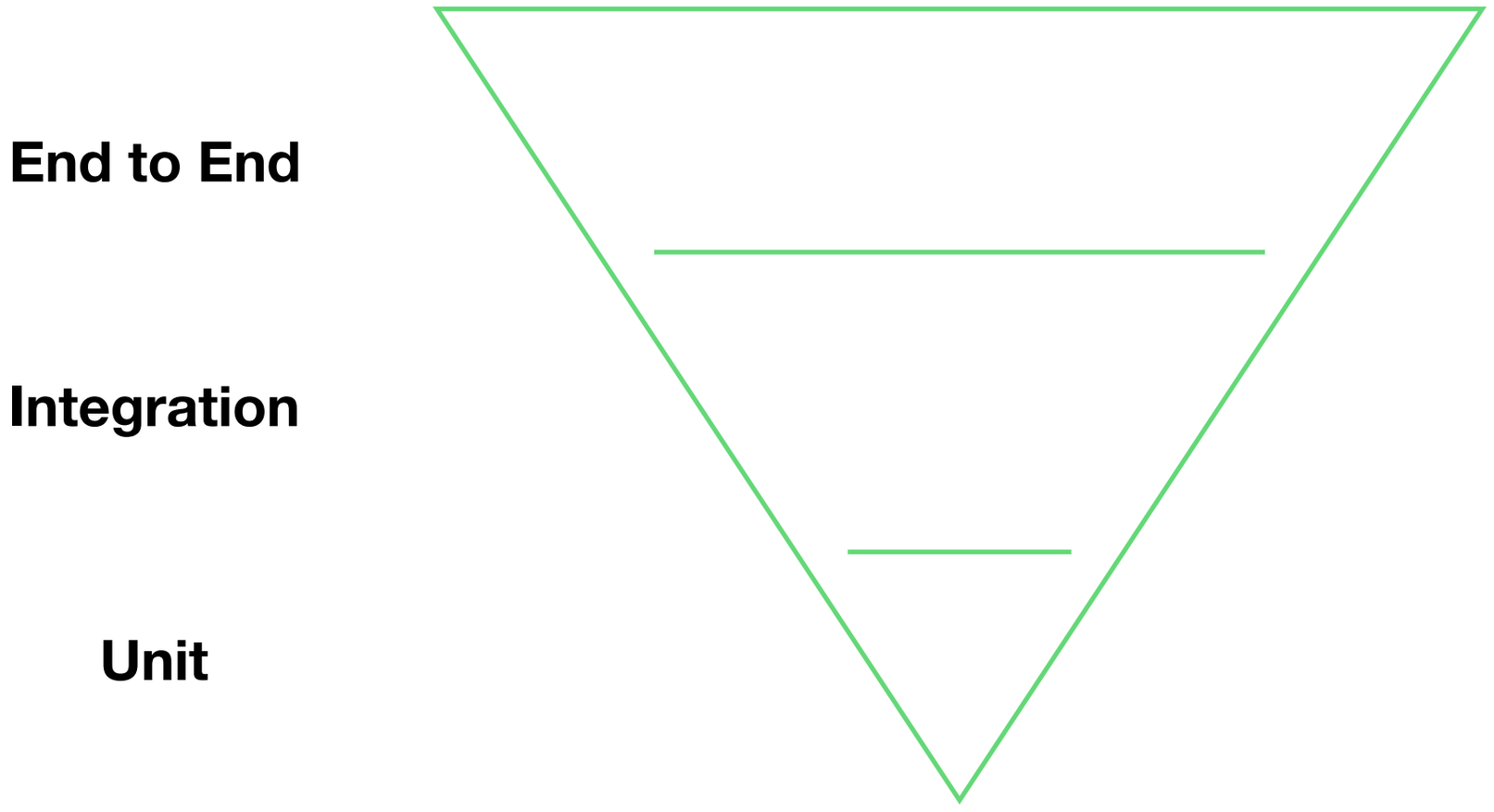


https://flic.kr/p/cBBDNf

# Root Cause Analysis

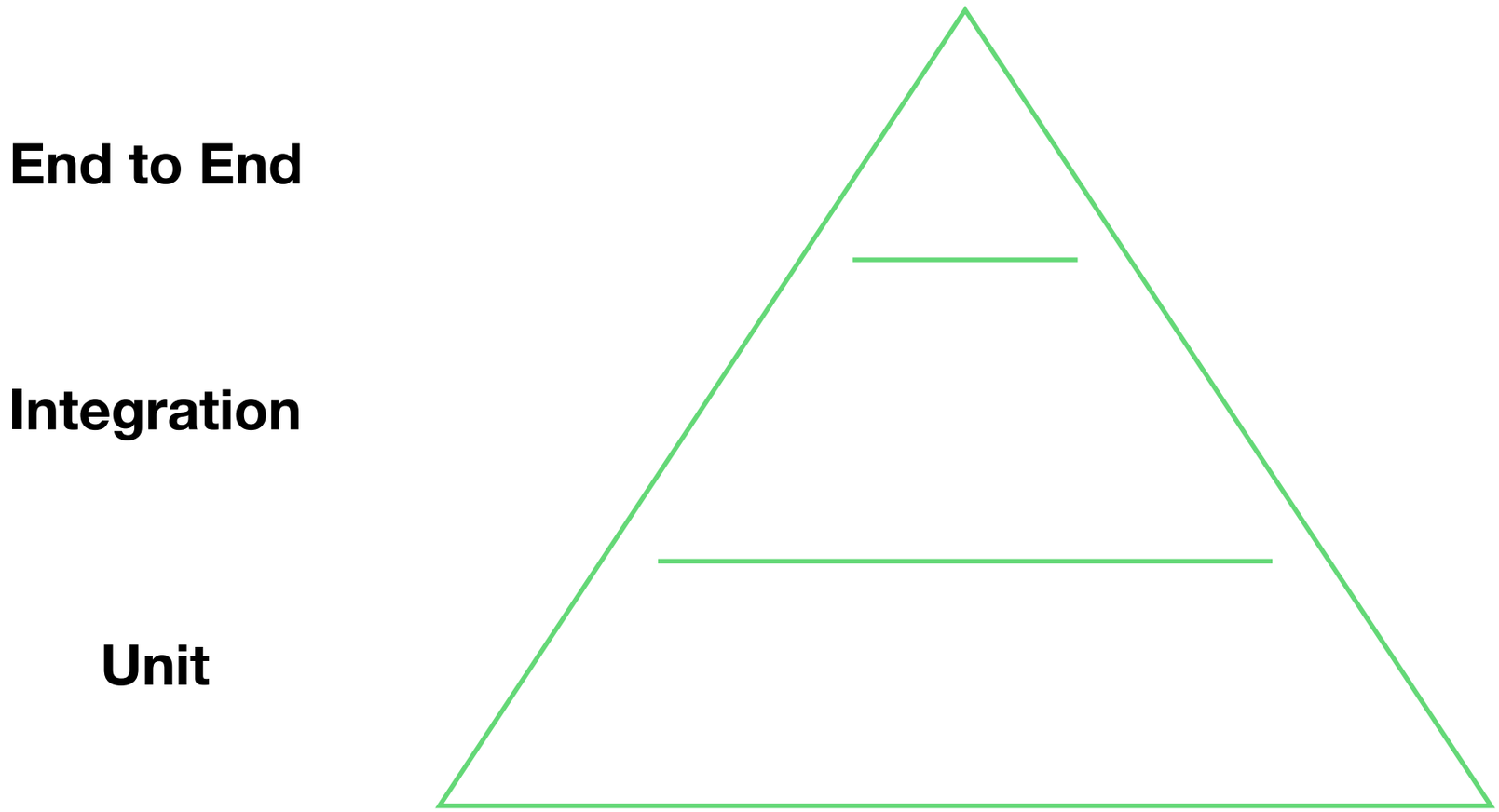
Too often, people “fix” problems in the Selenium tests without understanding why they happen

# The Test Ice Cream Cone




They'll only tell you that *something* in the stack is failing, but won't pinpoint where.

# The Test Pyramid

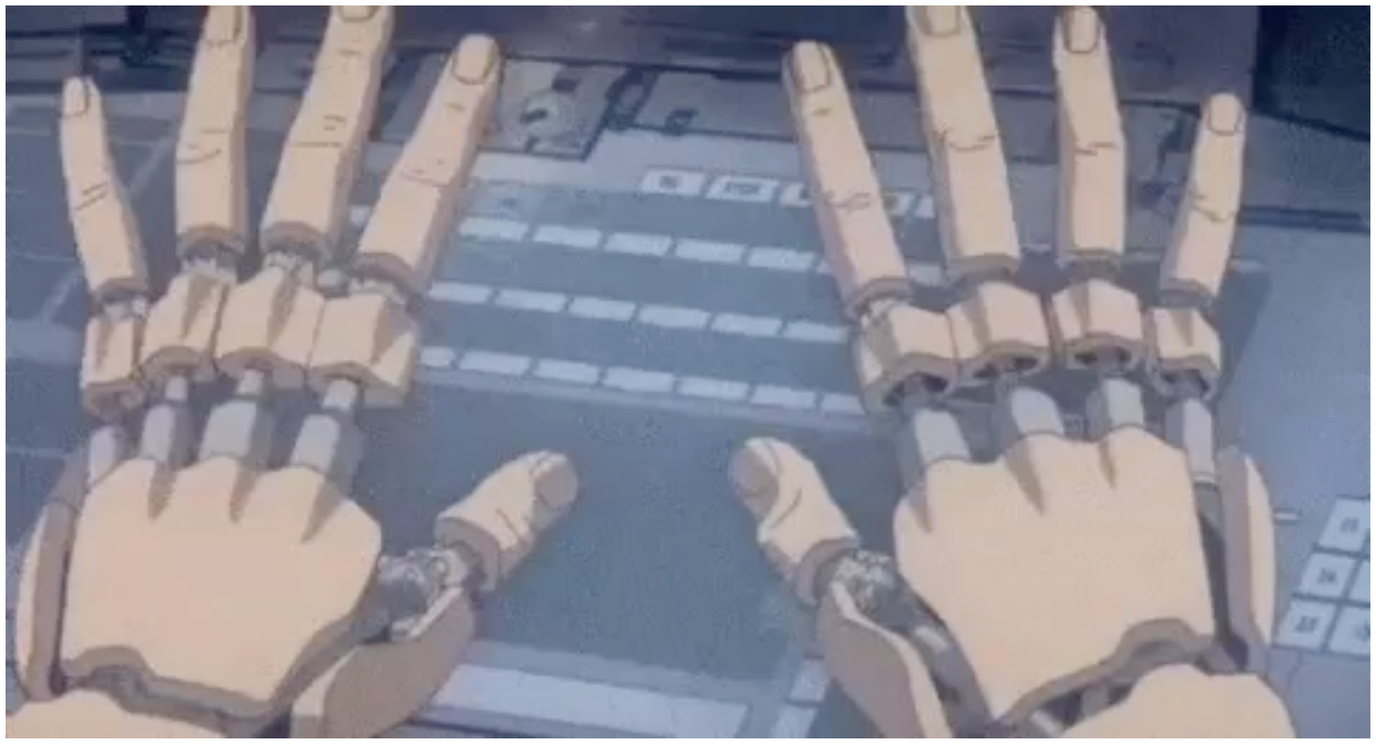



Stop writing Selenium tests. Start writing unit tests

# Test Identification

- Not all changes need all tests to run. Reduce the test time by only running those tests that need to be run.
- Graph analysis
    - Build tools like Buck or Bazel help
- Test labeling

# Data Driven Testing



- Consider running sanity checks in a wider range of browsers than you use for all the detailed tests

- Base browser choices on current user data + future trends

# Q & A

- Selenium Docker: https://github.com/SeleniumHQ/docker-selenium
- Zalenium: https://github.com/zalando/zalenium
- Cloud Providers:
    - Sauce Labs: https://saucelabs.com
    - Browser Stack: https://www.browserstack.com
    - TestingBot: https://testingbot.com
- Build Tools
    - Buck: https://buckbuild.com
    - Bazel: http://bazel.io