WebdriverIO

2020, Oleksandr Khotemskyi
xotabu4.github.io

Hello!

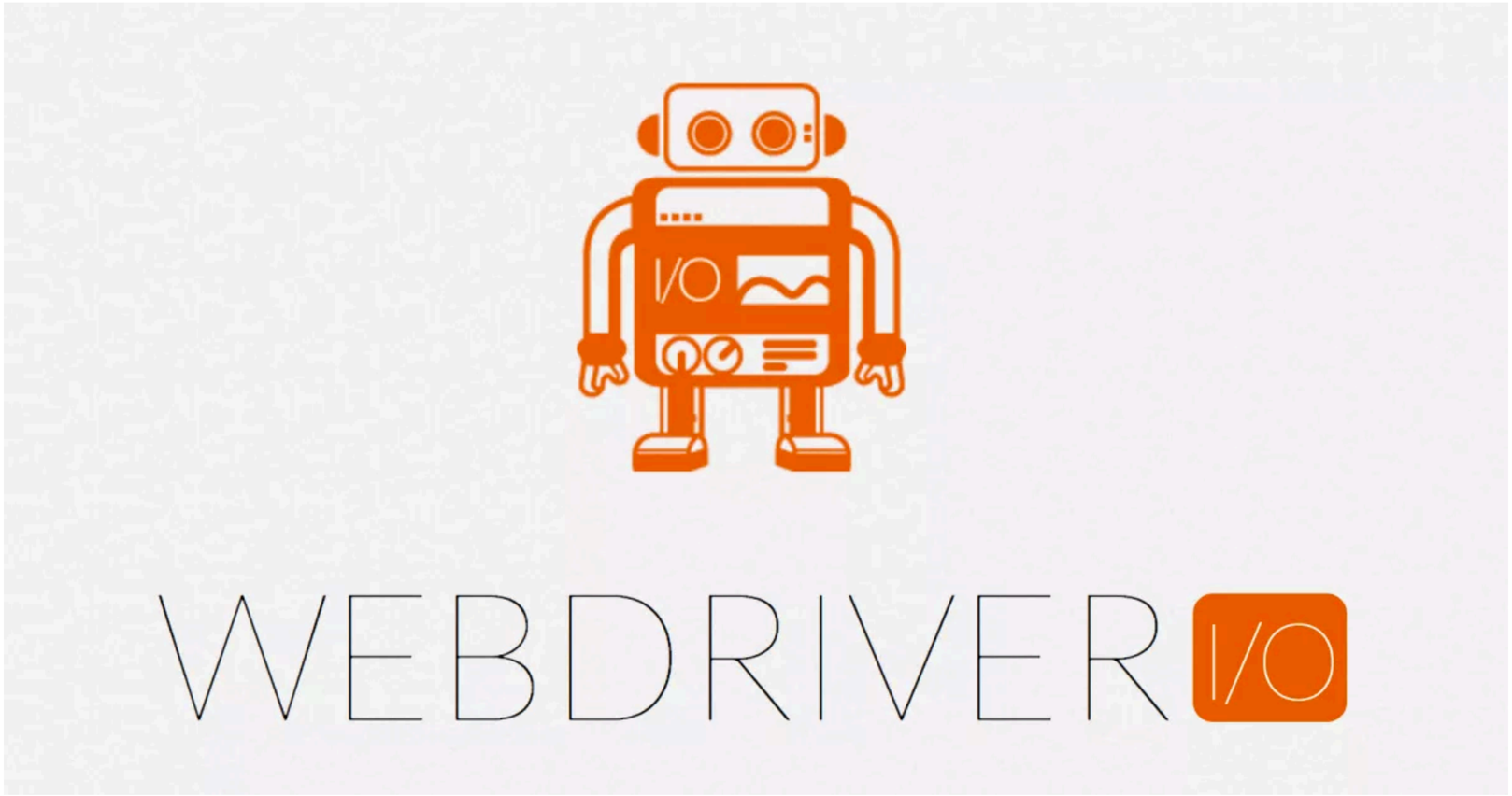
I Am Oleksandr Khotemskyi

Independent Contractor,
Software Developer Engineer in Test

Contacts: xotabu4.github.io

Plan and seed project

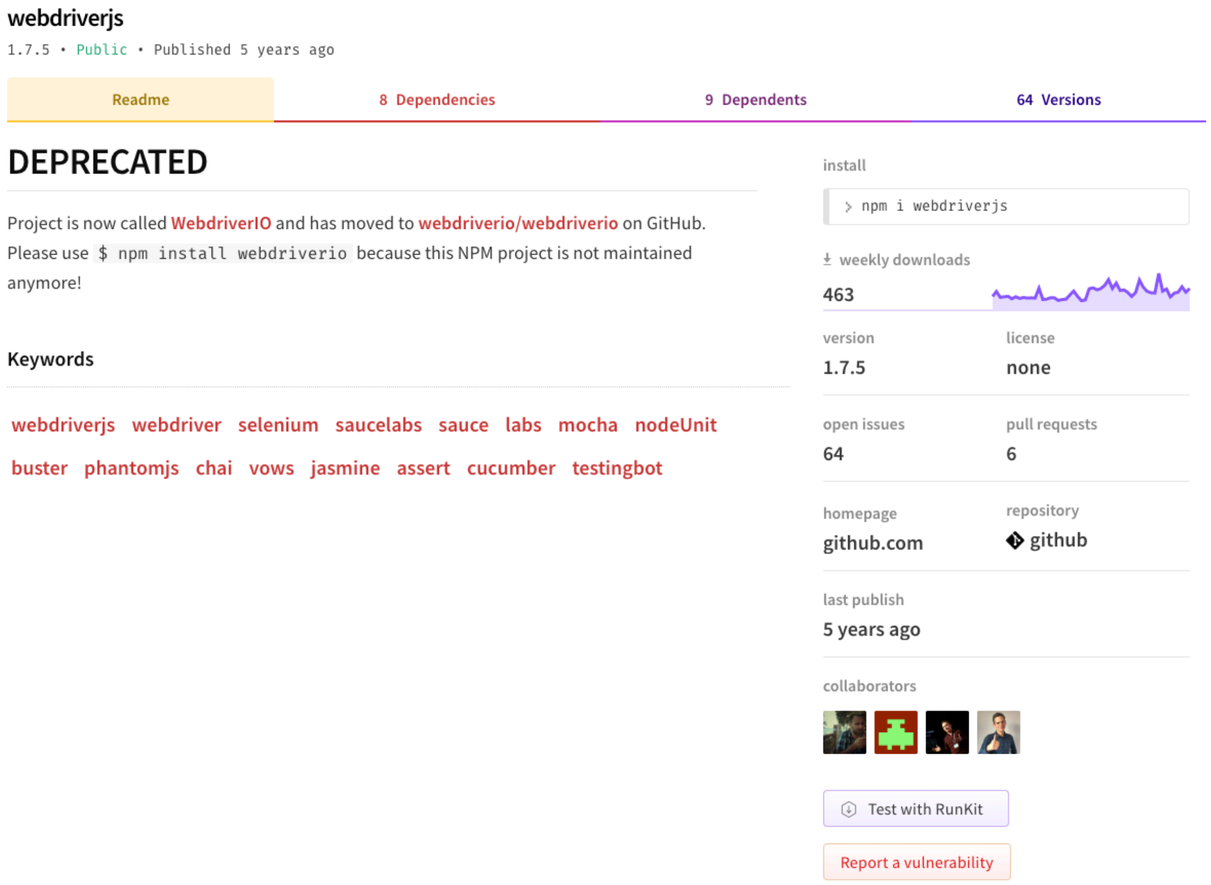
Part 1



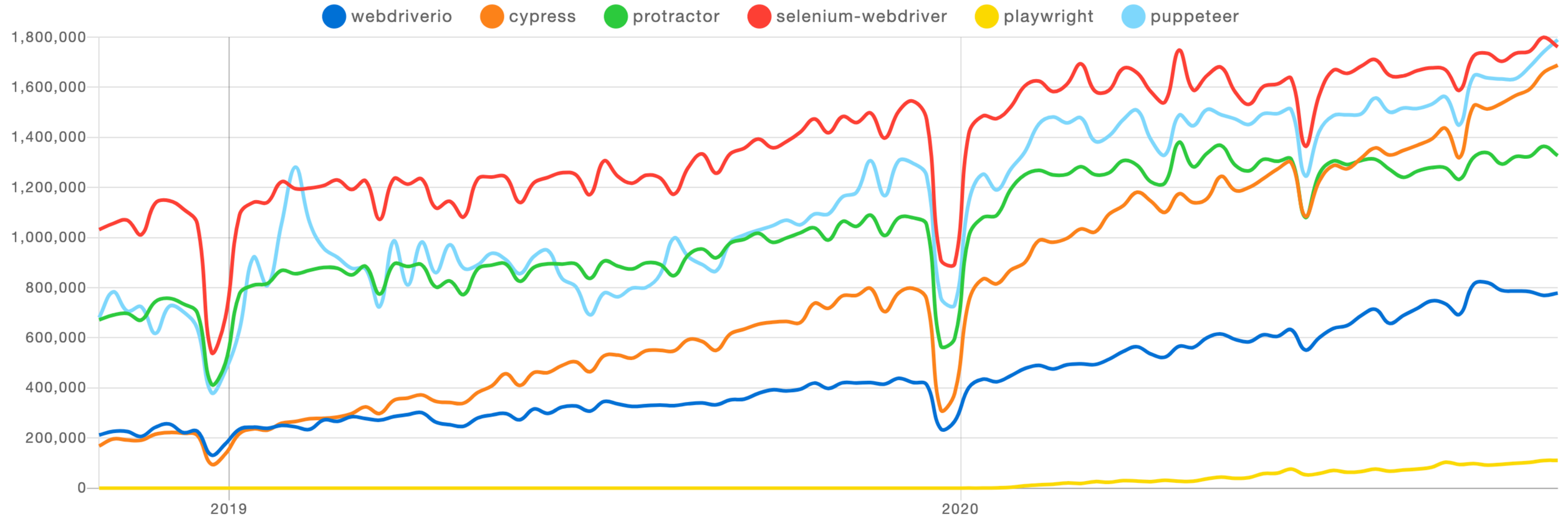
What is WebdriverIO?

History

- WebdriverIO was originally named WebdriverJS (before 2.0.0)
- Renamed to WebdriverIO in 2.0.0
- First version released long time ago and greatly evolved since then

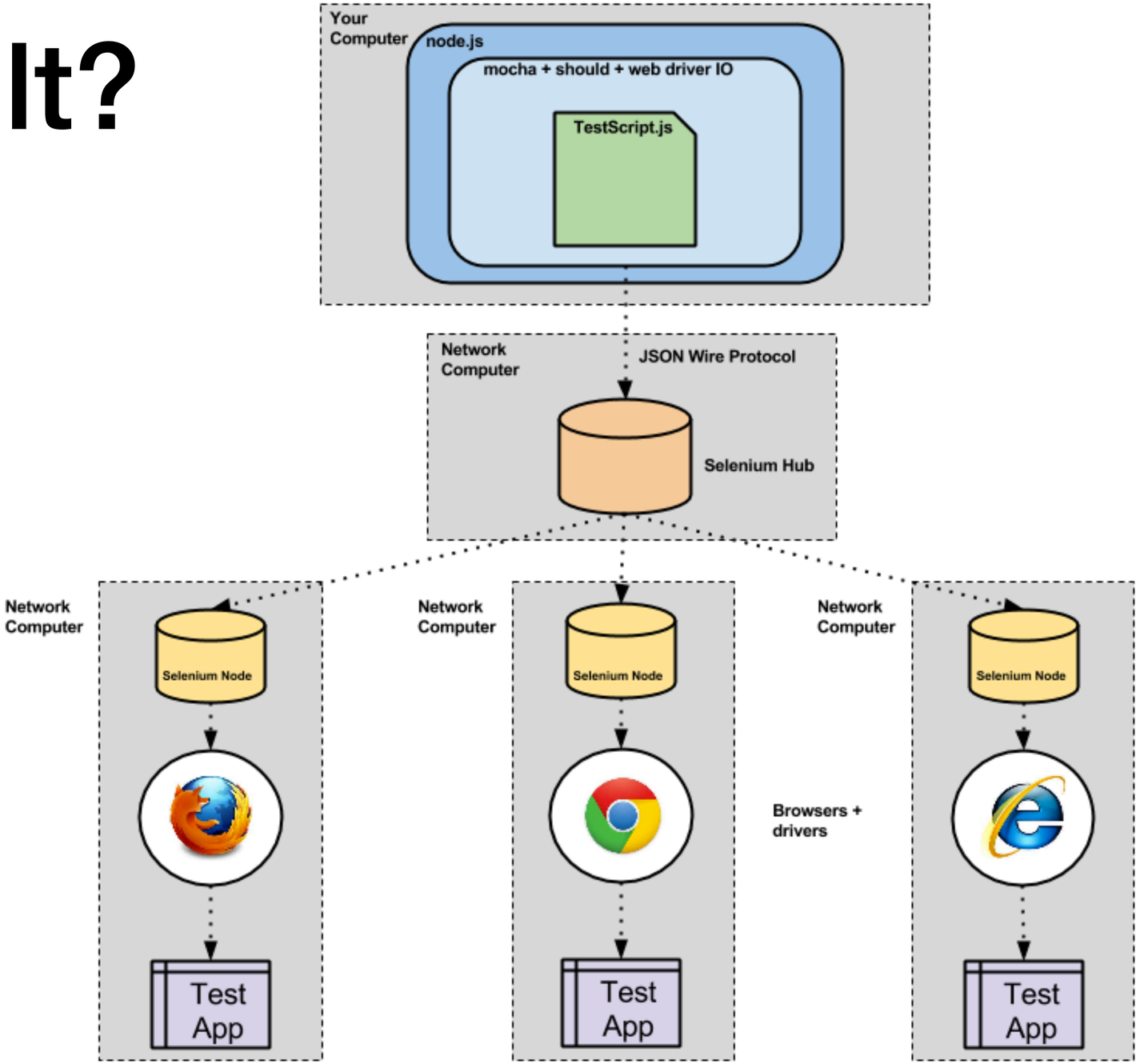


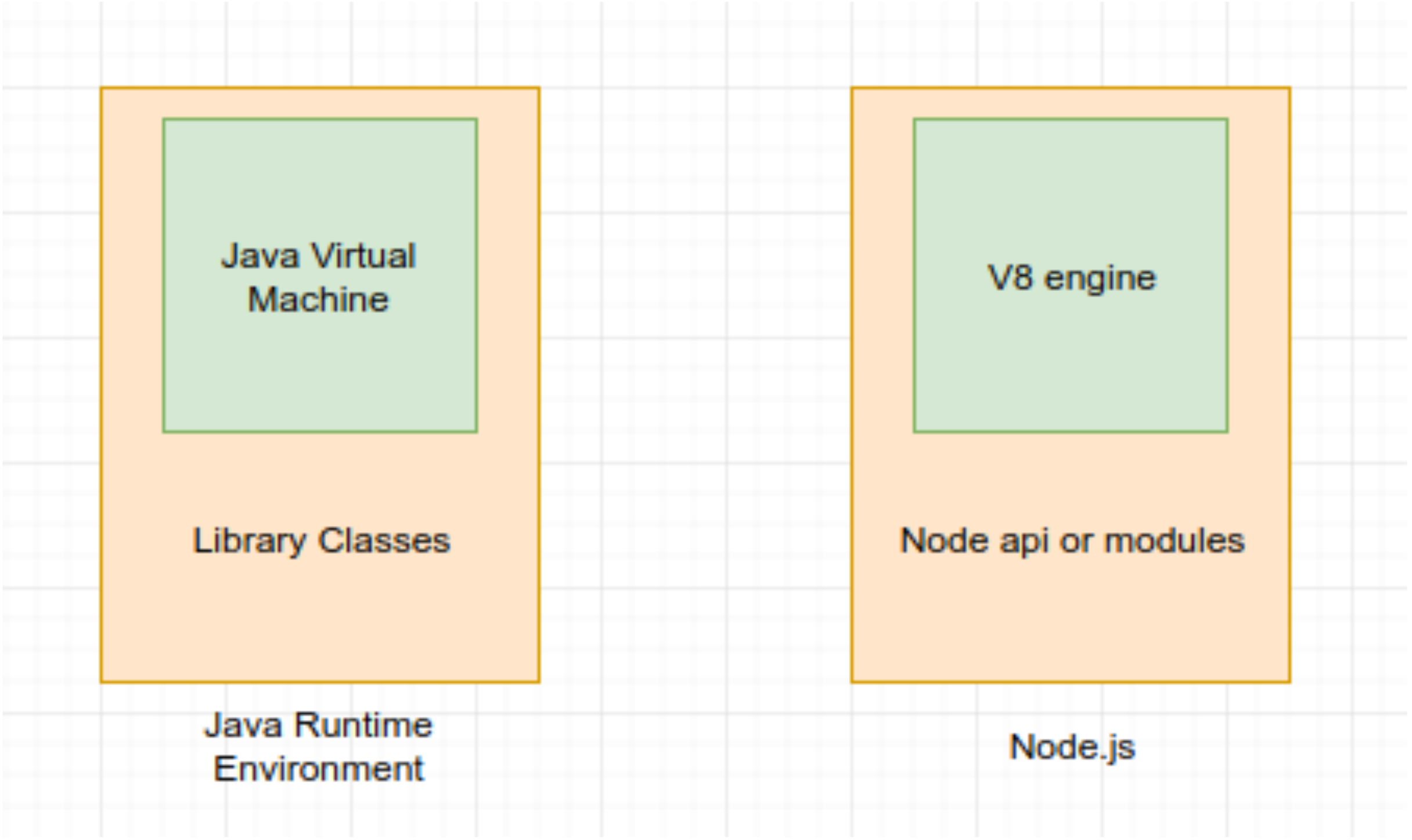
Downloads in past 2 Years



How it is built?

Looking inside

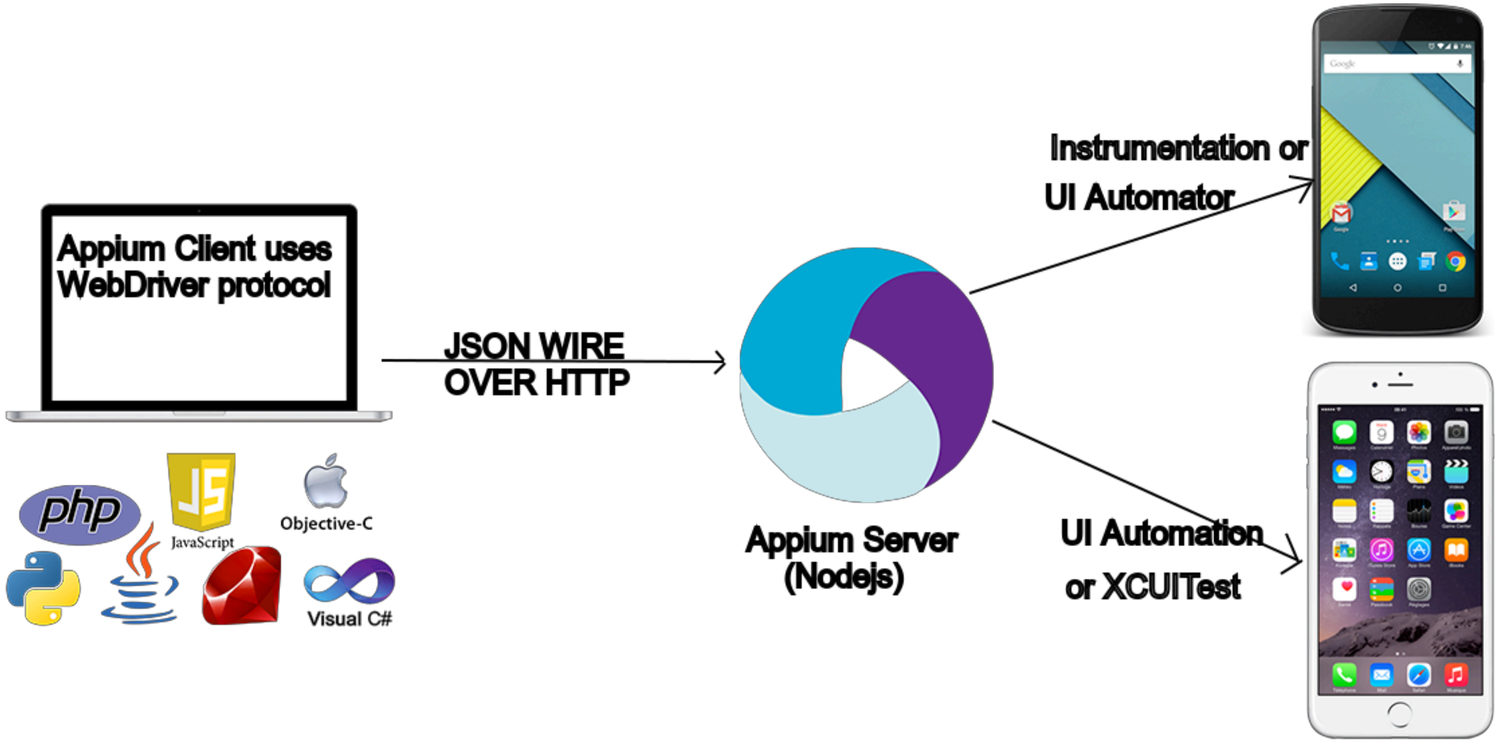



Java Virtual Machine
Library Classes
Java Runtime Environment
V8 engine
Node api or modules
Node.js

```
describe("Guest", function() {
  it("should be able to buy item", function() {
    browser.url("/rubber-ducks-c-1/red-duck-p-3");
    $("button.btn-success").click();
    browser.pause(1000); // yes i know
    browser.url("/checkout");
    // Filling checkout page
    $('input[name="firstname"]').waitForDisplayed();
    $('input[name="firstname"]').setValue("TestFirstName");
    $('input[name="lastname"]').setValue("TestLastName");
    $('input[name="address1"]').setValue("address line 1");
    $('input[name="address2"]').setValue("address line 2");
    $('input[name="postcode"]').setValue(faker.address.zipCode());
    $('input[name="city"]').setValue("CityName");
    $('input[name="email"]').setValue(faker.internet.email());
    $('input[name="phone"]').setValue(faker.phone.phoneNumber());
    $('button[name="save_customer_details"]').waitForDisplayed();
    $('button[name="save_customer_details"]').click();
    browser.waitUntil(
      () => $('button[name="confirm_order"]').getAttribute("disabled") == null,
      undefined,
      "Confirm order button should become enabled to click"
    );
    $('button[name="confirm_order"]').click();
    // Verifying that we are on confirmation page
    $("h1.title").waitForDisplayed();
    const confirmationText = $("h1.title").getText();
    expect(confirmationText).to.match(
      /Your order #.* is successfully completed!/
    );
    // Thank you for your purchase. An order confirmation email has been sent. We
  });
});
```

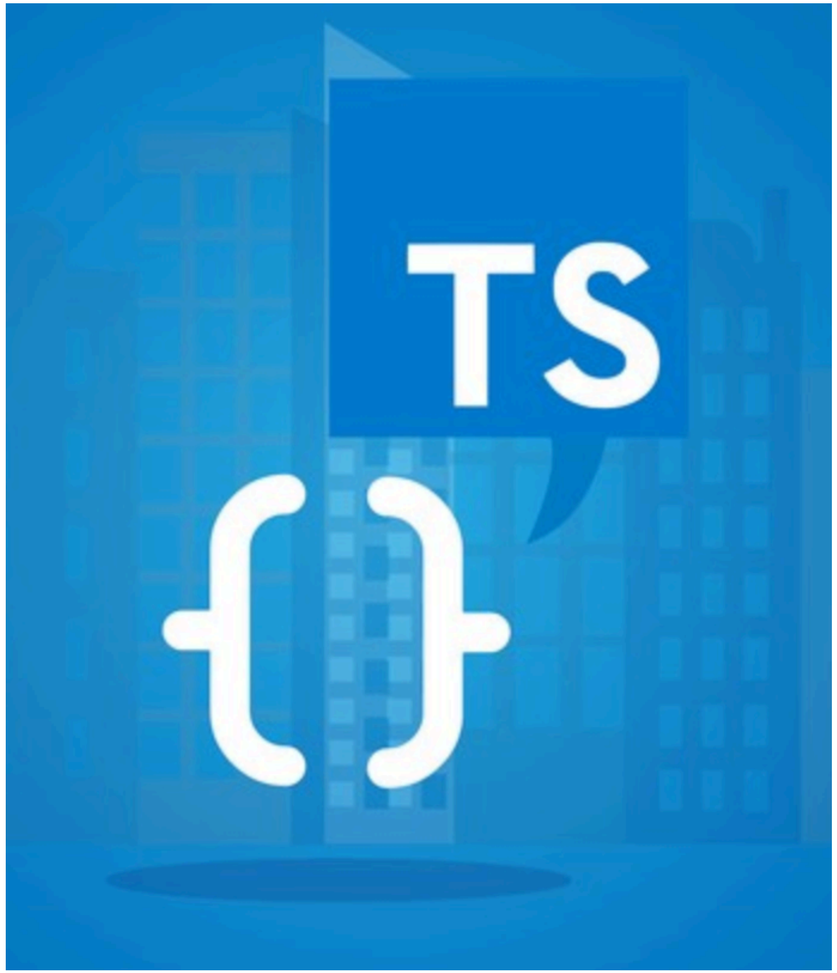
Appium support

- Appium JSON wire protocol is fully supported for both iOS and Android
- WebdriverIO can be used for both mobile web and native applications
- Also different test farms like SauceLabs are supported out of the box



TypeScript support

- TypeScript typings are included for both sync and async modes
- Provides autocompletion
- Provides types checks
- Provides code downgrade to be able to run on old NodeJS versions



CDP support

- WDIO has possibility to run commands thru Chrome debug protocol or webdriver protocol
- Also separate devtools service can be added to project

CommonJS: What node.js has been using

```
const fs = require('fs')
const { networkInterfaces } = require('os')
const msg = 'Hello'
module.exports = { msg }
```

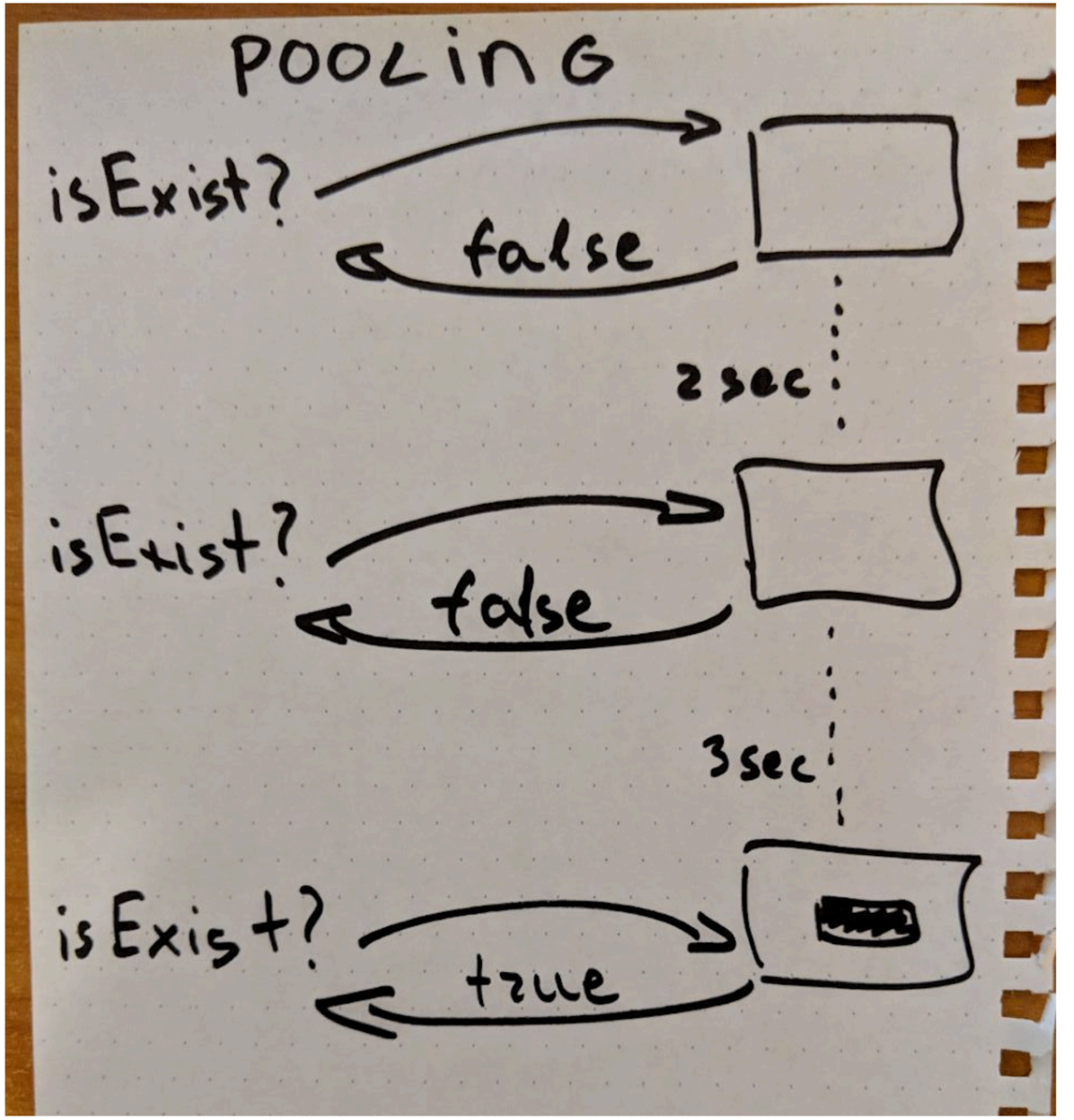
ES Modules: The standard from now on ?

```
import fs from 'fs'
import os, { networkInterfaces } from 'os'
export const msg = 'Hello'
```

Part 2

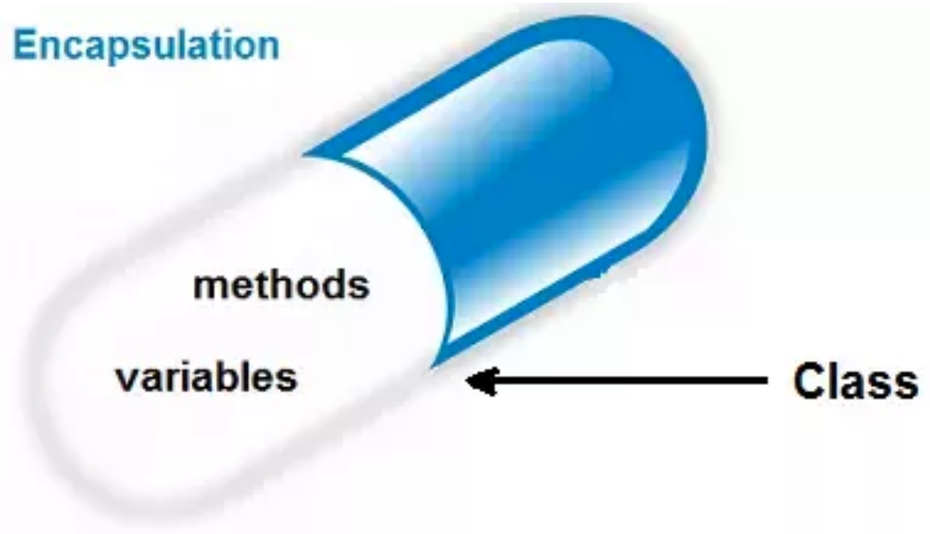
Explicit waits

- There are only 3 prepared conditions to wait:
  - waitForDisplayed
  - waitForEnabled
  - waitForExist
- They should be called on Element object
- And one more, universal - browser.waitUntil, that can take any function to wait for



Page Objects and Page Components

- Elements that you want to use inside PageObjects must be wrapped to getters or function
- This is needed because Page Object is created BEFORE browser is actually started, and all element searches at that point will receive ‘undefined’
- Things are getting even harder when you want to split to components
- Sometimes people just store locators as strings instead storing elements in page objects



```
18  class PageObject {
19    constructor() {
20
21    }
22    get someElement() {
23      return $('div')
24    }
25    doSomething() {
26        this.someElement.click()
27    }
28  }
29  const pageObject = new PageObject()
30  describe("Describe", function() {
31    it("1", function() {
32      browser.url("/dynamic_controls");
33      pageObject.doSomething()
34    });
35  });
```

Lazy elements

- Lazy Elements it is test automation pattern for searching for elements on the page
- It allows to start searching only when element is actually needed in tests - when you try to call some actions on it
- Also allows to automatically re-search element if it was re-created in DOM
- WebdriverIO claims that it uses this pattern
- But there is no any docs how it works

🚀 **New Feature**

- new package `@wdio/applitools-service` for simple visua
- new package `eslint-plugin-wdio` for WebdriverIO specif
- `@wdio/devtools-service` now with frontend performance
- new `region` option to simply run tests on SauceLabs in di
- `debug` command now allows to connect the runner with th debugging
- decoupling of `@wdio/sync` package from framework adapt want to run your commands with async/await
- autofetching of all provides log types
- auto retry mechansim for all command requests
- auto refetch mechanism for stale elements
- simplified reattachment to existing sessions with `attach` f
- integrated and auto maintained TypeScript definitions
- wdio testrunner fails if no spec files were found

Part 3

Sync mode

- No need to worry about promises, callbacks, async/await, control flow
- Much simpler to understand for beginners
- You can use core JavaScript features to handle exceptions, iterate, and others (try/catch, for/of …)
- Cant forget about putting await, or handle rejections
- Reduces cognitive load

WebdriverIO

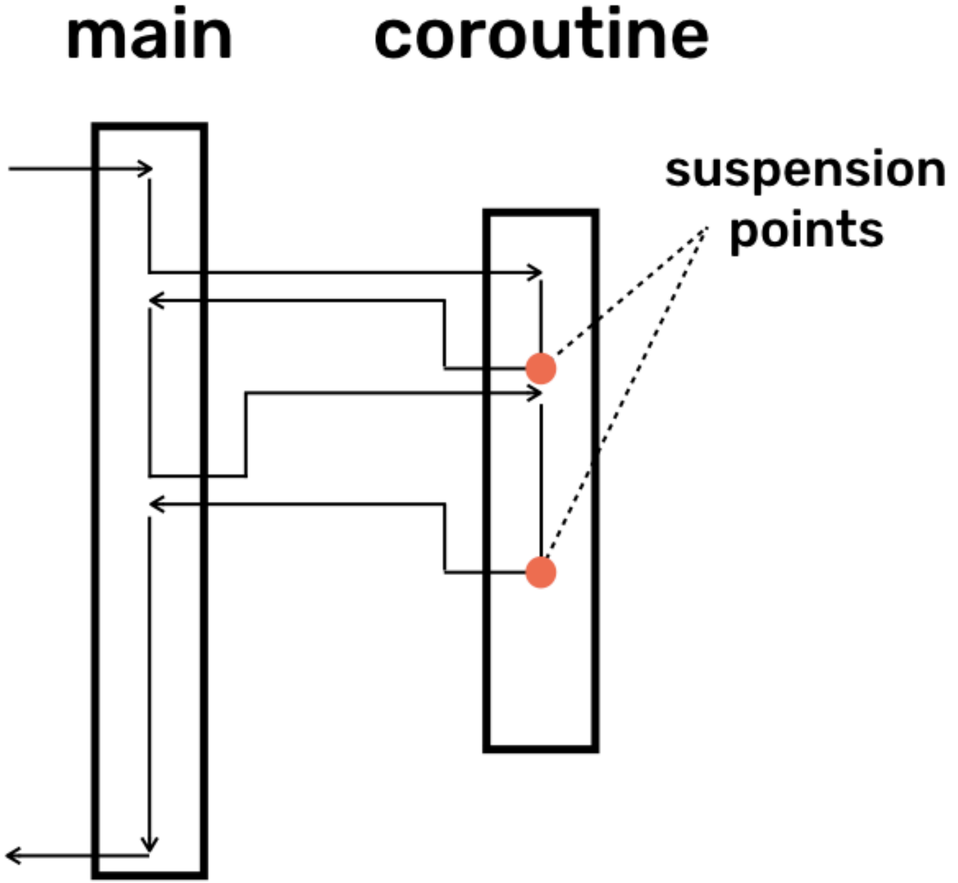
```
const divs = $$('div')
for (let div of divs) {
    console.log(div.getText())
}
```

ProtractorJS

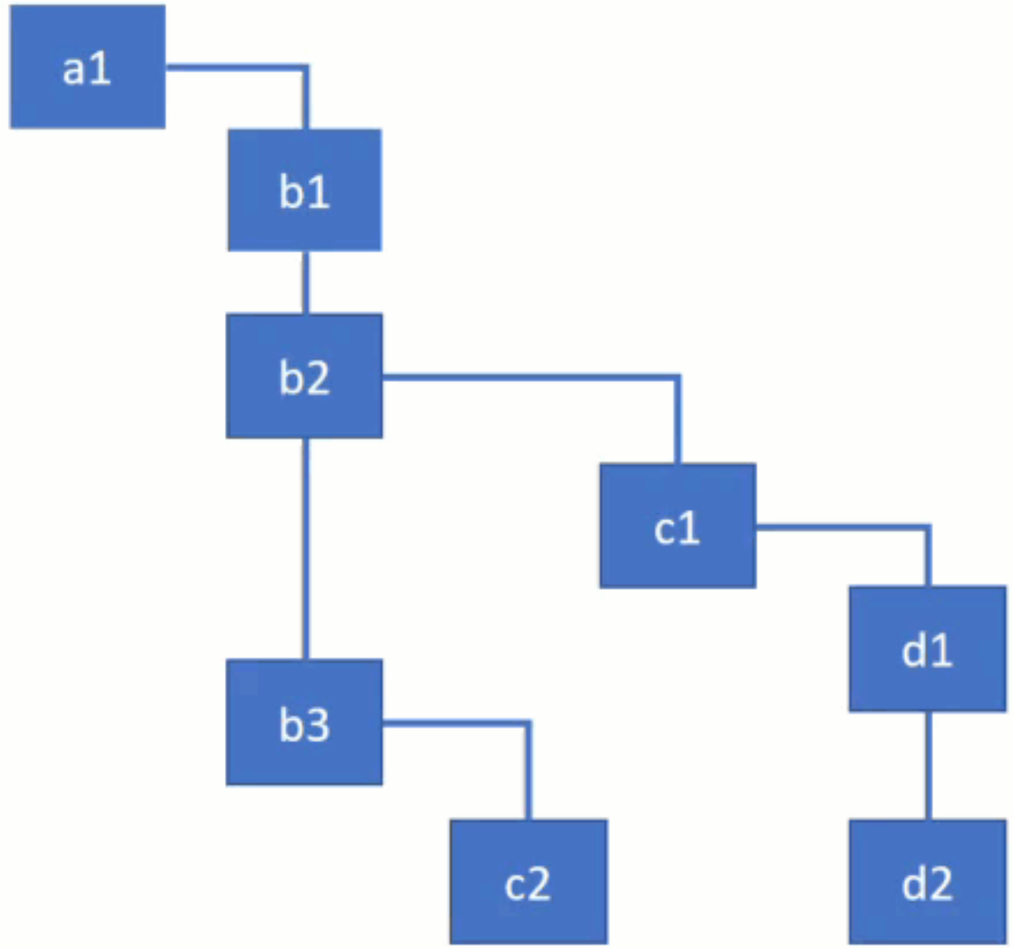
```
const divs = $$('div')
await Promise.all(divs.map(async div => {
    console.log(await div.getText())
}))
```

Fibers?

- Fibers (coroutines, рус. Сопрограммы) - programming approach to control concurrent code execution
- One of the main problems with automated tests that you need all your commands to be synchronized
- For example in official selenium webdriverjs you need to put ‘await’ before every async action
- Fibers allows to suspend code execution and continue it without callbacks, control flows, promises or async/await
- Your code looks synchronous, but actually it is multithreaded on C++ level



Fibers



nextUnitOfWork

performUnitOfWork

beginWork

completeUnitOfWork

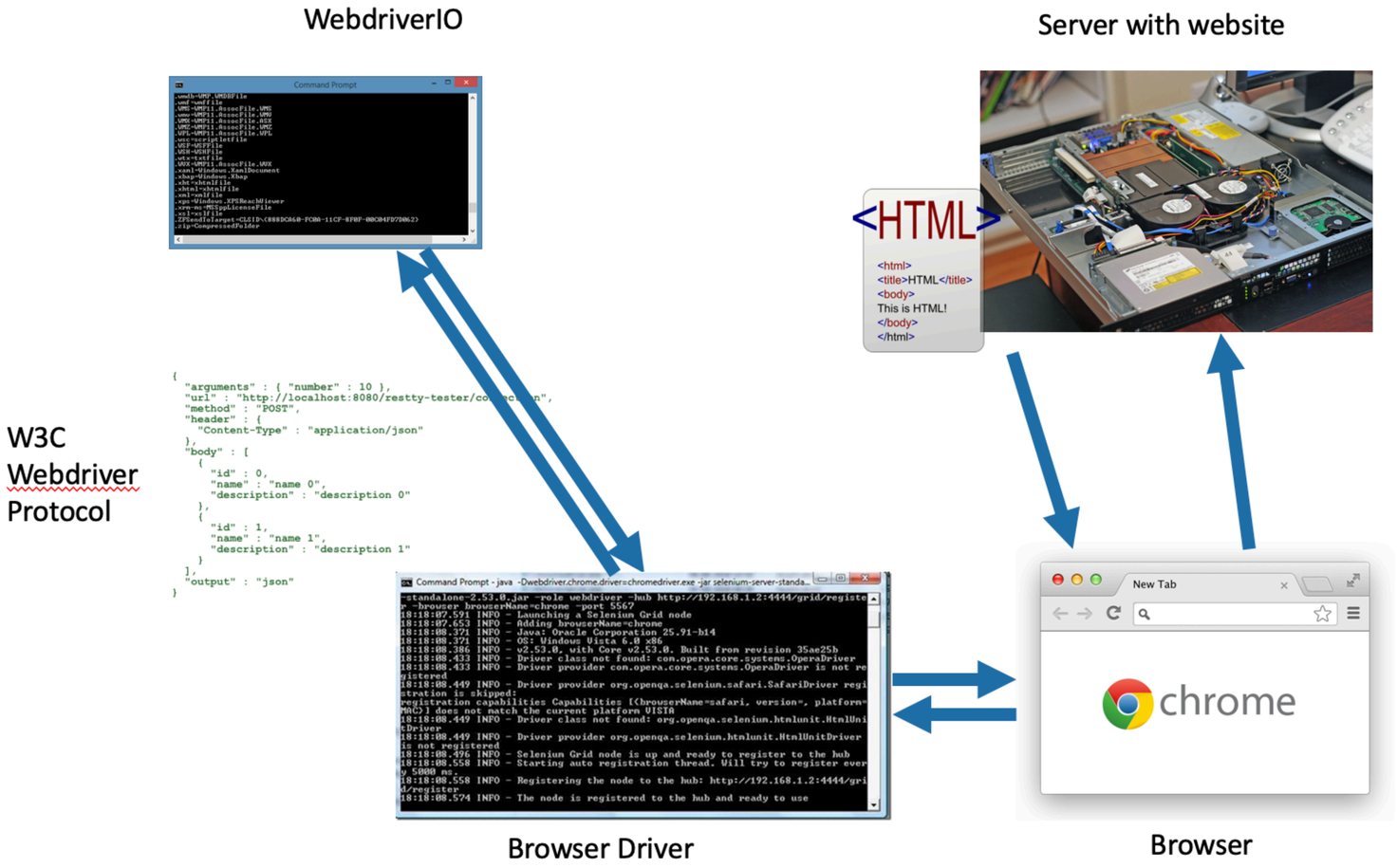
completeWork

WebdriverIO packages

- WebdriverIO is actually a lot of different packages, that work together
- There are different packages to handle different aspects of your tests
- Each package can be used separately, or in combination with others
- Each package might have different versions, and aims to be independent from others

webdriver package

- Sends commands thru HTTP to webdriver protocol compatible server
- Parses responses into objects
- Provides commands for appium
- Another implementation of webdriver client for javascript (instead official webdriverjs)
- When used as it is - it is called Standalone Mode



webdriverio package

- Special high-level framework that helps organize your code into tests
- Wraps test-runners like jasmine, mocha, cucumber, launches and controls them
- Provides support for configuration files
- Allows connecting additional reporters
- Allows connecting additional services

Other packages

- Reporting packages
  - Allure
  - Different console reporters
  - Junit
- Framework support
  - MochaJS
  - JasmineJS
  - CucumberJS
- CLI - command line support
- Logging package
- And many others

Part 4

CLI (Command line interface)

- WebdriverIO on start can accept some additional options
- This is useful to dynamically pass some arguments
- Also it can generate config file with step-by-step questions

https://webdriver.io/docs/clioptions.html

```
./node_modules/.bin/wdio --help

WebdriverIO CLI runner

Usage: wdio [options] [configFile]
Usage: wdio config
Usage: wdio repl <browserName>
Usage: wdio install <type> <name>

config file defaults to wdio.conf.js
The [options] object will override values from the config file.
An optional list of spec files can be piped to wdio that will override configured specs.
Same applies to the exclude option. It can take a list of specs to exclude for a given run
and it also overrides the exclude key from the config file.

Commands:
  wdio.js repl <browserName>  Run WebDriver session in command line
  wdio.js install <type> <name> Add a `reporter`, `service`, or `framework` to your WebdriverIO project

Options:
  --help                prints WebdriverIO help menu                    [boolean]
  --version             prints WebdriverIO version                      [boolean]
  --hostname, -h        automation driver host address                   [string]
  --port, -p            automation driver port                           [number]
  --user, -u            username if using a cloud service as automation backend
                                                                         [string]
  --key, -k             corresponding access key to the user             [string]
  --watch               watch specs for changes                         [boolean]
  --logLevel, -l        level of logging verbosity
                        [choices: "trace", "debug", "info", "warn", "error", "silent"]
  --bail                stop test runner after specific amount of tests have
                        failed                                           [number]
  --baseUrl             shorten url command calls by setting a base url [string]
  --waitforTimeout, -w  timeout for all waitForXXX commands              [number]
  --framework, -f       defines the framework (Mocha, Jasmine or Cucumber) to
                        run the specs                                    [string]
  --reporters, -r       reporters to print out the results on stdout     [array]
  --suite               overwrites the specs attribute and runs the defined
                        suite                                            [array]
  --spec                run only a certain spec file - overrides specs piped
                        from stdin                                       [array]
  --exclude             exclude spec file(s) from a run - overrides specs piped
                        from stdin                                       [array]
  --mochaOpts           Mocha options
  --jasmineOpts         Jasmine options
  --cucumberOpts        Cucumber options
```

https://webdriver.io/docs/clioptions.html

Configuration file

- Entry-point for your webdriverIO project
- Configuration is written as JS object, and can be dynamic
- Allows to define various parts of how your tests should be executed

```
exports.config = {
    runner: 'local',
    hostname: 'localhost',
    port: 4444,
    path: '/wd/hub',
    specs: [
        'test/spec/**'
    ],
    exclude: [
        'test/spec/multibrowser/**',
        'test/spec/mobile/**'
    ],
    maxInstances: 10,
    capabilities: [
        {
            browserName: 'chrome'
        }
    ],
    logLevel: 'info',
    logLevels: {
        webdriver: 'info',
    },
    outputDir: __dirname,
    bail: 0,
    baseUrl: 'http://localhost:8080',
    waitforTimeout: 1000,
    framework: 'mocha',
    specFileRetries: 1,
    reporters: [
        'dot',
        ['allure', {
            outputDir: './'
        }]
    ]
};
```

https://webdriver.io/docs/configurationfile.html

Services

- In configuration file you can set special hooks-functions
- They allow to inject your code into running tests
- For example - “run this function at the end of everything”
- But if you have too many big hooks, you can migrate them away from configuration file into separate module, called Service

```
export default class CustomService {
  onPrepare (config, capabilities) {
    // TODO: something before the workers launch
  }

  onComplete (exitCode, config, capabilities) {
    // TODO: something after the workers shutdown
  },

  // ...
}
```

https://webdriver.io/docs/customservices.html

Supported service hooks

```
2     onPrepare: function (config, capabilities) {},
3     beforeSession: function (config, capabilities, specs) {},
4     before: function (capabilities, specs) {},
5     beforeSuite: function (suite) {},
6     beforeHook: function () {},
7     afterHook: function () {},
8     beforeTest: function (test) {},
9     beforeCommand: function (commandName, args) {},
10    afterCommand: function (commandName, args, result, error) {},
11    afterTest: function (test) {},
12    afterSuite: function (suite) {},
13    after: function (result, capabilities, specs) {},
14    afterSession: function (config, capabilities, specs) {},
15    onComplete: function (exitCode, config, capabilities, results) {},
16    onReload: function(oldSessionId, newSessionId) {}
```

Reporters

- WebdriverIO tests runner allows to add own test execution flow events
- You can create own Reporter
- Uses event listener approach to react to events happening in your running tests
- Primary idea - allow to collect test results and work with them

https://webdriver.io/docs/customreporter.html

Supported Reporter events

- onRunnerStart () {}
- onBeforeCommand () {}
- onAfterCommand () {}
- onScreenshot () {}
- onSuiteStart () {}
- onHookStart () {}
- onHookEnd () {}
- onTestStart () {}
- onTestPass () {}
- onTestFail () {}
- onTestSkip () {}
- onTestEnd () {}
- onSuiteEnd () {}
- onRunnerEnd () {}

Thanks!

2020, Oleksandr Khotemskyi

xotabu4.github.io