End-to-End Testing

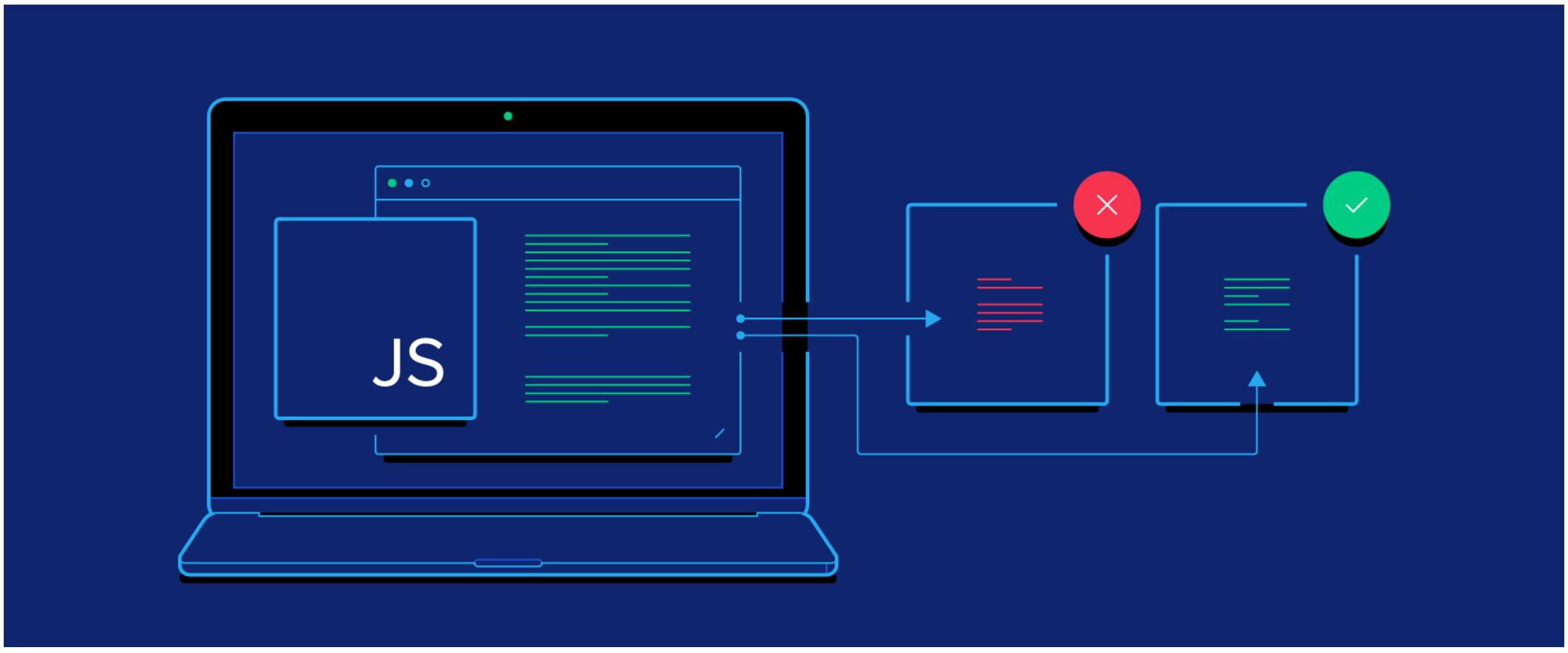
with

Agenda

1. **What is Cypress?**
2. **Why Cypress?**
3. **Setting up Environment and Running Tests**
4. **Core Concepts**
5. **Lite E2E Testing with Cypress**

What Cypress

- **Cypress** is a **next generation front end** testing tool built for **the modern web.**



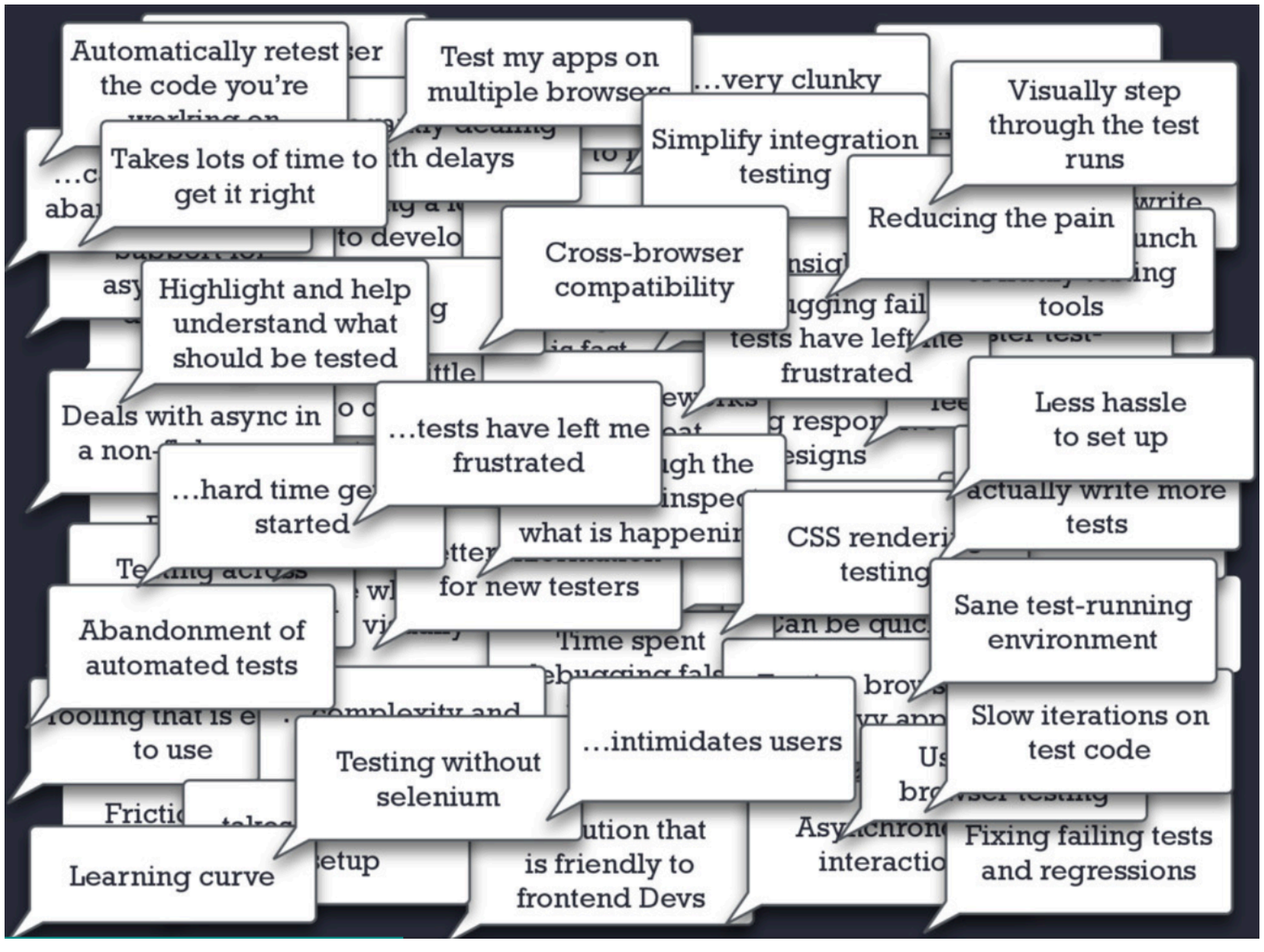
Targets

- Users: developers or QA engineers using modern JavaScript frameworks.

- Cypress enables you to write:
  - End-to-end tests
  - Integration tests
  - Unit tests

Why Cypress

- Testing is too hard



Setup

Having **sane configuration**

Less hassle to set up

Being **easy** to setup

Frictionless test setup

No endless **setup hassle**

…having to go through a **pain of setup**

Writing

Simpler to write

Easy to write

Testing **responsive designs**

See what broke visually

Support for **async** testing

Running **fast** and **reliably**

It must be **easy to use**

Management

Flake

Intermittent test failures

Less **brittle** to changes

Easy to maintain

Understand **why** test failed

More **insight** into failing tests

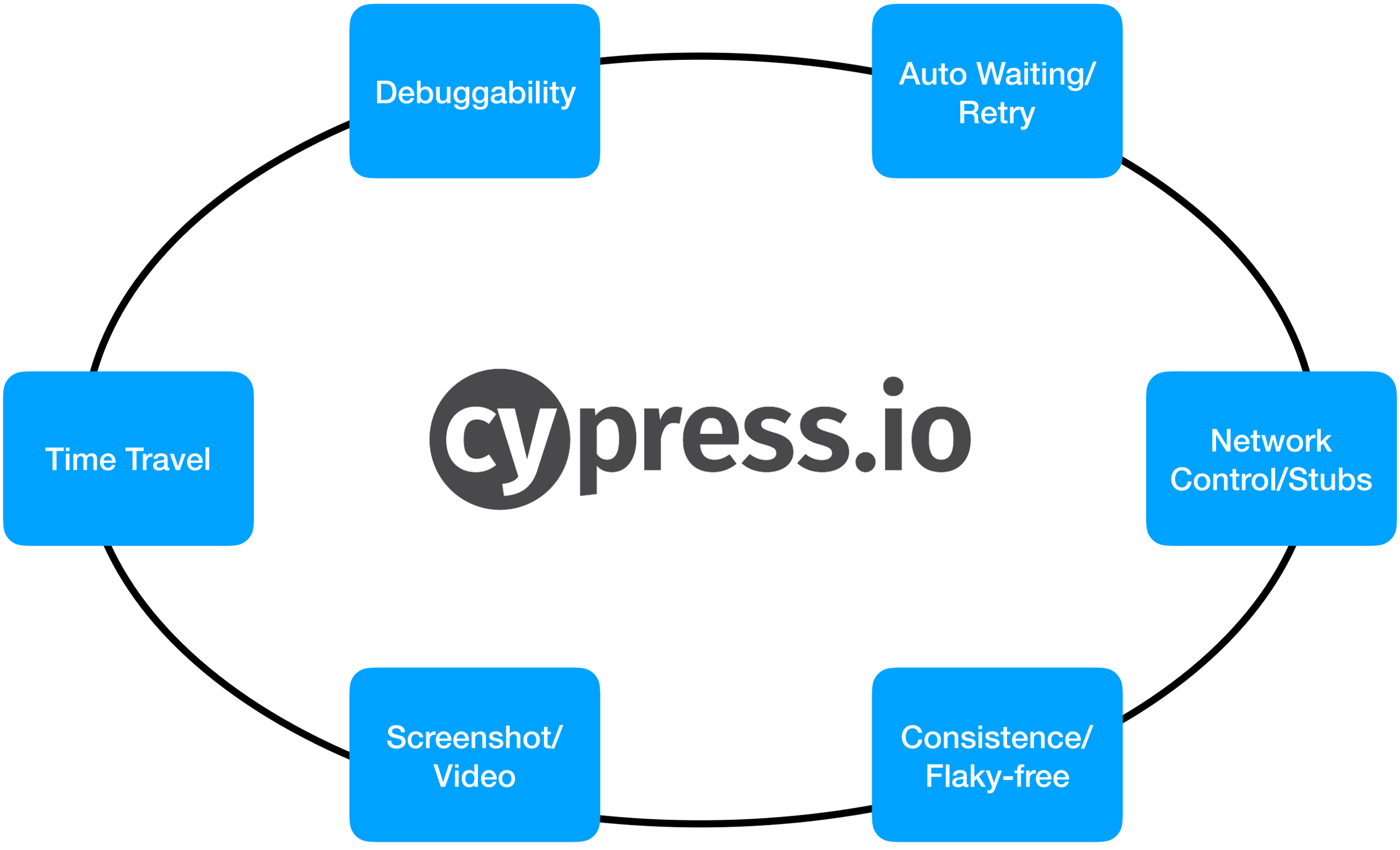
Wasted time spent debugging false negatives

Why Cypress

It makes our life a lot easier with:

- Set up tests
- Write tests
- Run tests
- Debug tests

https://docs.cypress.io/guides/overview/why-cypress.html#In-a-nutshell



https://docs.cypress.io/guides/overview/why-cypress.html#Features

Community

Cypress.io

Fast, easy and reliable testing for anything that runs in a browser

Atlanta, GA https://cypress.io hello@cypress.io Verified

Repositories 134 Packages People 10

Pinned repositories

cypress

Fast, easy and reliable testing for anything that runs in a browser.

CoffeeScript ★ 15.5k 859

cypress-docker-images

Docker images with Cypress dependencies and browsers

Dockerfile ★ 256 84

cypress-documentation

Cypress Documentation including Guides, API, Plugins, Examples, & FAQ.

JavaScript ★ 172 373

cypress-example-kitchensink

This is an example app used to showcase Cypress.io testing.

HTML ★ 377 480

cypress-example-todomvc

The official TodoMVC tests written in Cypress.

JavaScript ★ 91 153

cypress-example-recipes

Various recipes for testing common scenarios with Cypress

JavaScript ★ 745 326

Document

- Documents are written extremely well at

  https://docs.cypress.io/

- Cypress provides Guides, Examples, APIs, Plugins and FAQ documentation

Setup

Easy Setup & Run

- **Prerequisite**: Node JS and npm installed

- yarn:

  yarn add —dev cypress

- npm:

  npm init && npm install —save-dev cypress

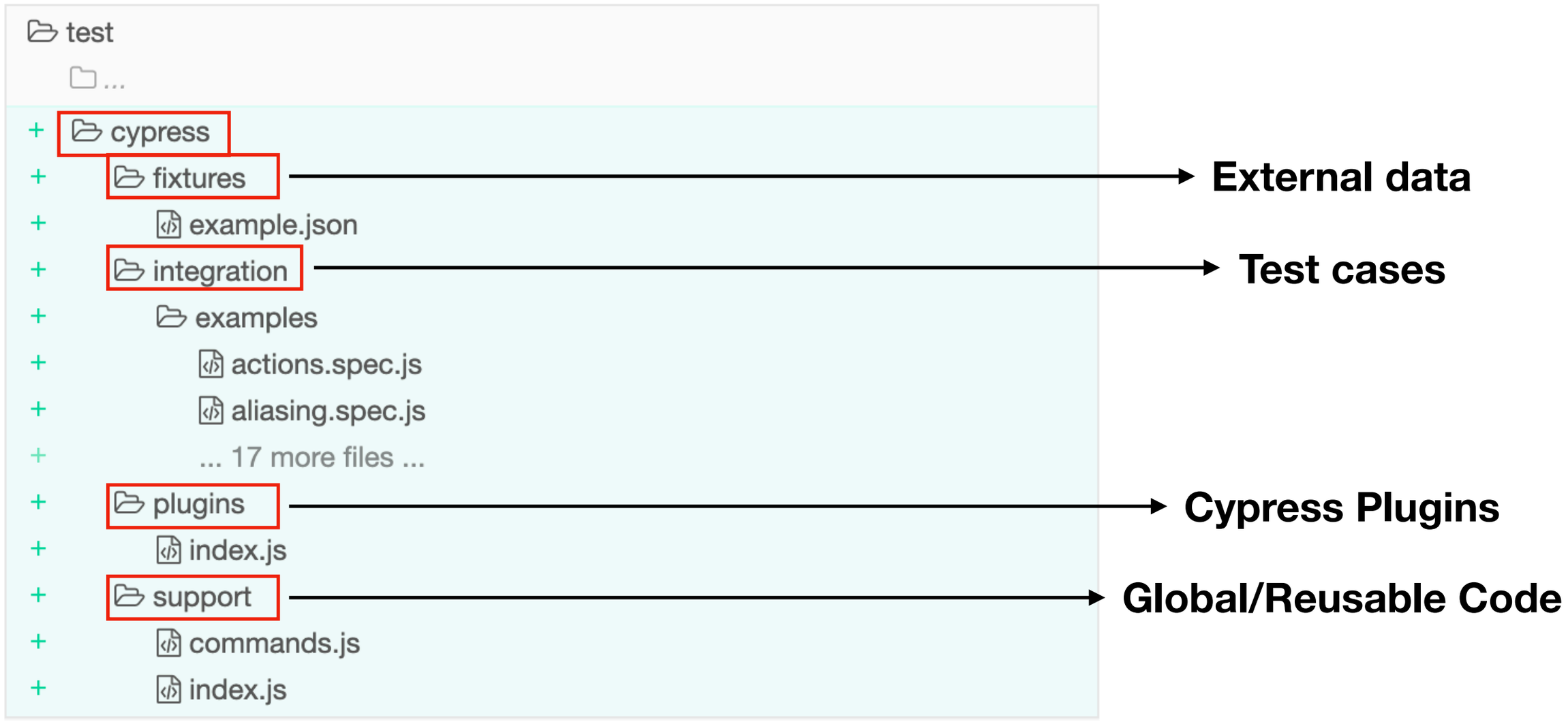
- Run/Open:

  npx cypress open/ npx cypress run

https://docs.cypress.io/guides/getting-started/installing-cypress.html#Installing

Clean Structure

- When running **npx cypress open,** cypress will auto-generate sample test cases along with sample structure



https://docs.cypress.io/guides/core-concepts/writing-and-organizing-tests.html#Folder-Structure

Core Concepts

Command & Assertion

- Simple + Straight forward

```
1
2   describe("TodoMVC - React", function() {
3
4     context("Add to do", function() {
5       it("should allow me to add one todo item", function() {
6         cy.visit("/");
7         cy.get(".new-todo")
8           .type("Create to do note 1{enter}")
9
10        cy.get(".todo-list li")
11          .should("have.length", 1)
12          .should("contain", "Create to do note 1")
13      })
14    })
15
16  })
```

```
it("should allow me to add one todo item", () => {
  cy.visit("/");
  cy.get(".new-todo")
    .type("Create to do note 1{enter}")

  cy.get(".todo-list li")
    .should("have.length", 1)
    .should("contain", "Create to do note 1")
})
```

1. Visit page at /
2. Get element having **class “.new-todo”**
3. Type **“Create to do note 1”** and press **“Enter”**
4. Get elements having **tag li** in **class “.todo-list”**
5. Assert at step 4, we only have one element
6. The element at step 4 have text: **“Create to do note 1”**

```
it("should allow me to add one todo item", () => {
  cy.visit("/");
  cy.get(".new-todo")
    .type("Create to do note 1{enter}")

  cy.get(".todo-list li")
    .should("have.length", 1)
    .should("contain", "Create to do note 1")
})
```

Commands

Assertions

Commands

Commands

- Cypress Commands are asynchronous, but run **serially**
- **Chains of Commands** mechanism**.**
- Commands **are Promises but not Promises**

Asynchronous

- How does Cypress Execute this?

```
it("should allow me to add one todo item", () => {
  cy.visit("/");
  cy.get(".new-todo")
    .type("Create to do note 1{enter}")

  cy.get(".todo-list li")
    .should("have.length", 1)
    .should("contain", "Create to do note 1")
})
```

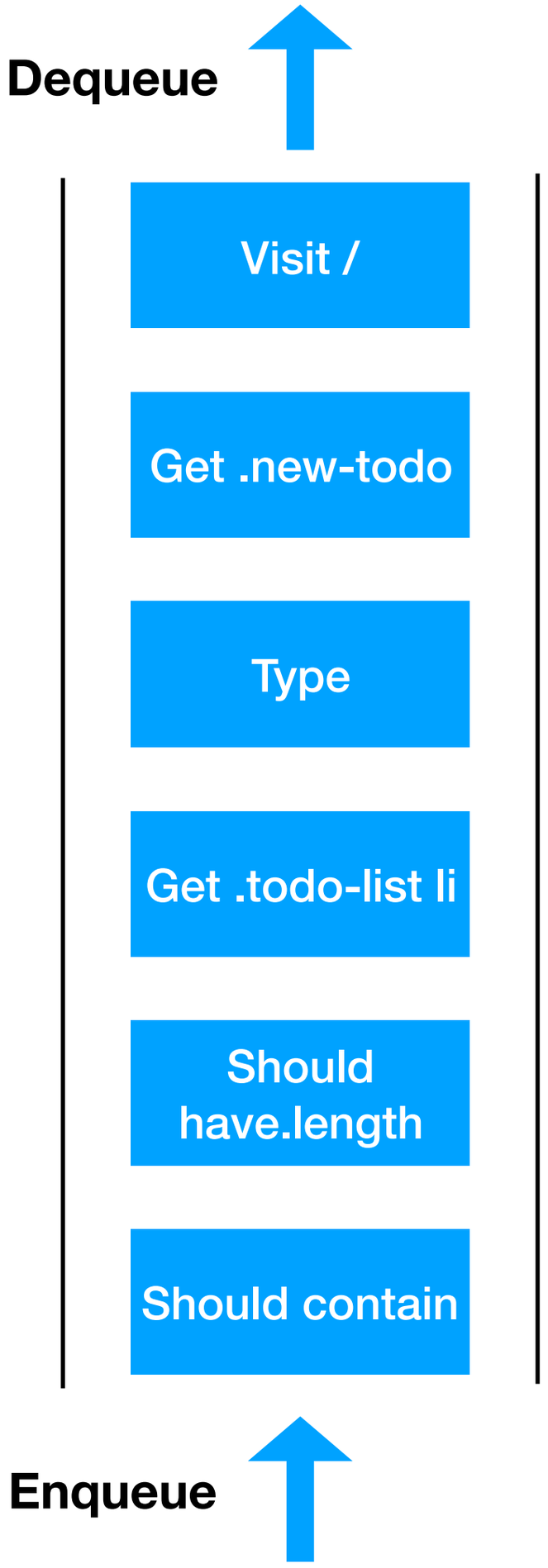
https://docs.cypress.io/guides/core-concepts/introduction-to-cypress.html#Commands-Are-Asynchronous

How does Cypress Execute this?

```
it("should allow me to add one todo item", () => {
  cy.visit("/");
  cy.get(".new-todo")
    .type("Create to do note 1{enter}")

  cy.get(".todo-list li")
    .should("have.length", 1)
    .should("contain", "Create to do note 1")
})
```

- Cypress run through all commands/ assertions
- Enqueue each command for later execution
- Magic happens when reaching the end of function

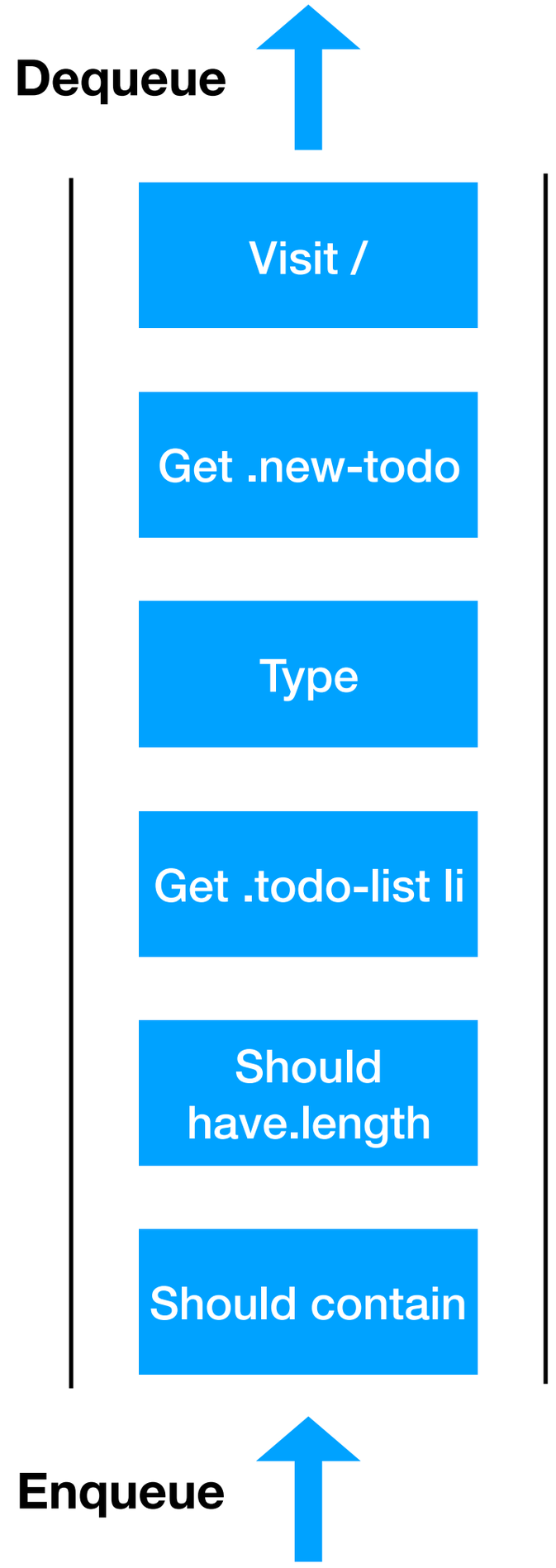


How does Cypress Execute this?

```
it("should allow me to add one todo item", () => {
  cy.visit("/");
  cy.get(".new-todo")
    .type("Create to do note 1{enter}")

  cy.get(".todo-list li")
    .should("have.length", 1)
    .should("contain", "Create to do note 1")
})
```

- Cypress dequeues and run serially each command in order

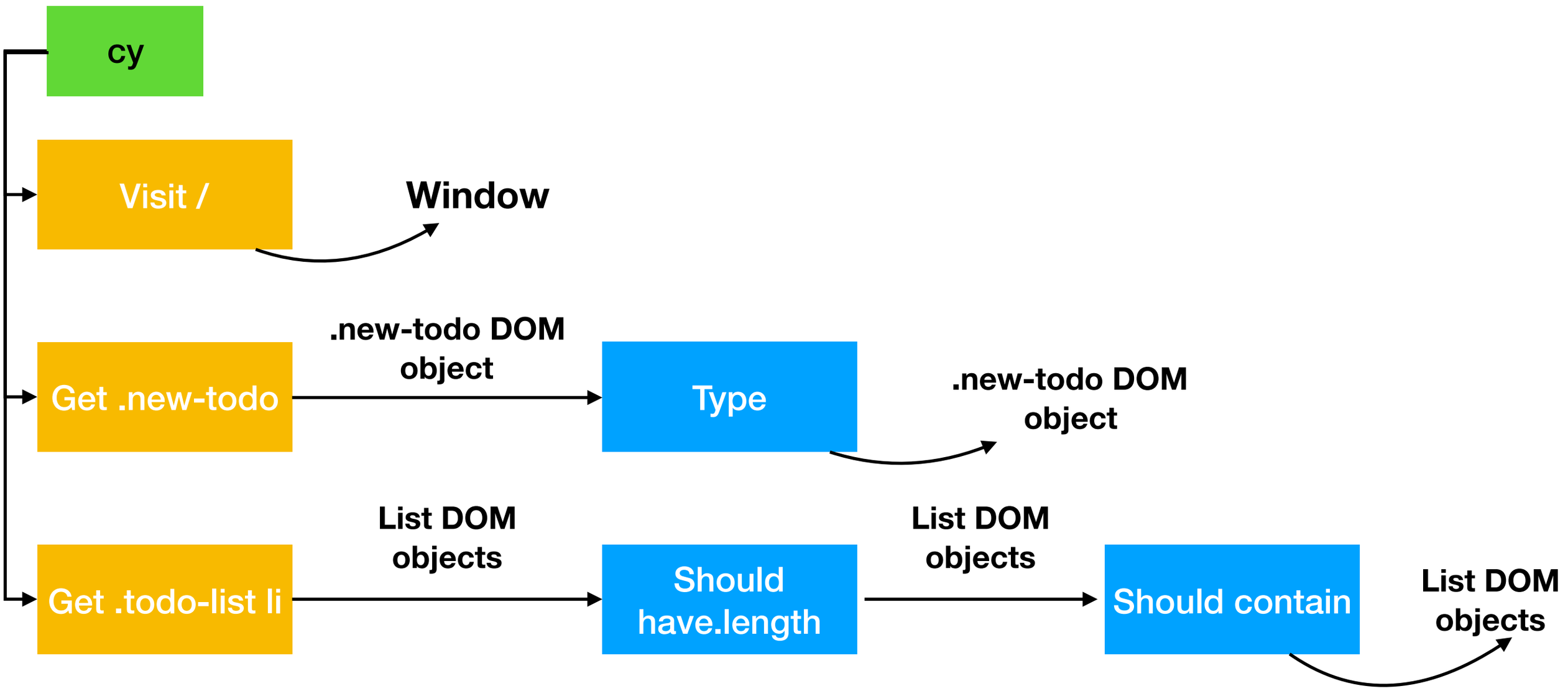


Chains of Commands

- A new Cypress chain always starts with **cy.[command]**, where what is **yielded** by the command establishes what other commands can be called next (chained).

```
it("should allow me to add one todo item", () => {
  cy.visit("/");
  cy.get(".new-todo")
    .type("Create to do note 1{enter}")

  cy.get(".todo-list li")
    .should("have.length", 1)
    .should("contain", "Create to do note 1")
})
```



Promise

The `Promise` object represents the eventual completion (or failure) of an asynchronous operation, and its resulting value

JavaScript Demo: Promise Constructor

```
var promise1 = new Promise(function(resolve, reject) {
  setTimeout(function() {
    resolve('foo');
  }, 1000);
});

promise1.then(function(value) {
  console.log(value);
  // expected output: "foo"
});

console.log("This is is a Promise");

```

Commands are Promises

```
cy.get('form')
.then(($form) => {
  console.log('form is:', $form)
  // undefined is returned here, but $form will be
  // yielded to allow for continued chaining
}).find('input').then(($input) => {
  // we have our $input element here since
  // our form element was yielded and we called
  // .find('input') on it
})
```

!!!!This code is for demonstration

```
it('changes the URL when "awesome" is clicked', function() {
  // THIS IS NOT VALID CODE.
  // THIS IS JUST FOR DEMONSTRATION.
  return cy.visit('/my/resource/path')
  .then(() => {
    return cy.get('.awesome-selector')
  })
  .then(($element) => {
    // not analogous
    return cy.click($element)
  })
  .then(() => {
    return cy.url()
  })
  .then((url) => {
    expect(url).to.eq('/my/resource/path#awesomeness')
  })
})
```

https://docs.cypress.io/guides/core-concepts/introduction-to-cypress.html#Commands-Are-Promises

Commands are not Promises

1. You cannot race or run multiple commands at the same time (in parallel).

2. You cannot ‘accidentally’ forget to return or chain a command.

3. You cannot add a **.catch** error handler to a failed command.

Retry-ability

```
it("should allow me to add one todo item", () => {
  cy.visit("/");
  cy.get(".new-todo")
    .type("Create to do note 1{enter}")

  cy.get(".todo-list li")
    .should("have.length", 1)
    .should("contain", "Create to do note 1")
})
```

Wait for response

Wait for item to render

Wait for processing create note

Cypress commands retry until command is successful or failed

-> Command also has its built-in Assertion

Retry-ability

- Cypress only retries commands that query the DOM
- Commands changing the state of the application under test cannot be retried. Ex: click, type, select, clearCookies, …
- Instead, the actionability is checked before the commands start

https://docs.cypress.io/guides/core-concepts/retry-ability.html#Not-every-command-is-retried

Assertions

Assertions

- Assertions describe the **desired** state of your **elements**, your **objects**, and your **application**.

- Describe what your application should look like, and Cypress will automatically **block, wait, and retry** until it reaches that state.

Retry-ability

- Assertion failed will cause the most recent command to retry along with the assertion

```
it("should allow me to add one todo item", () => {
  cy.visit("/");
  cy.get(".new-todo")
    .type("Create to do note 1{enter}")

  cy.get(".todo-list li")
    .should("have.length", 1)
    .should("contain", "Create to do note 1")
})
```

Retry

Failed

If one of these assertions fails, command will be retried along with it

https://docs.cypress.io/guides/core-concepts/retry-ability.html#Only-the-last-command-is-retried

Cypress in Lite

- Cypress e2e is triggered in 6 countries by Spinnaker every 4 hours

- Cypress e2e is triggered when Lite new version is released

- Demo with Cypress

Thank you !!!