Let's Start Testing

Ynon Perek
ynon@ynonperek.com
http://ynonperek.com

Agenda

- Developing for the web. Why is it hard?
- What can you test?
- Testing architecture

Why is it hard?

Why is it hard?

- Many different browsers
- Many different devices

Automatic testing is your only way out …

Imagine …

- Short feedback loop on changes
- Reduce human mistakes
- TDD

Types of Tests

Unit tests

System tests

http://johnny.github.io/jquery-sortable/

Unit Tests

- During development, for the developer
- Instant feedback
- Future aware
- Easy to maintain

System Tests

- Tests entire system
- For the client
- Easy to write

What Can You Test?

- Everything …
- But you probably shouldn’t

What Should You Test?

- Past bugs
- Intended behaviour
- Edge cases

Testing Architecture

Selenium

js-test-driver

TeamCity

Chutzpa

Jenkins

Mocha

CasperJS

Travis

Karma

Istanbul

Jasmine

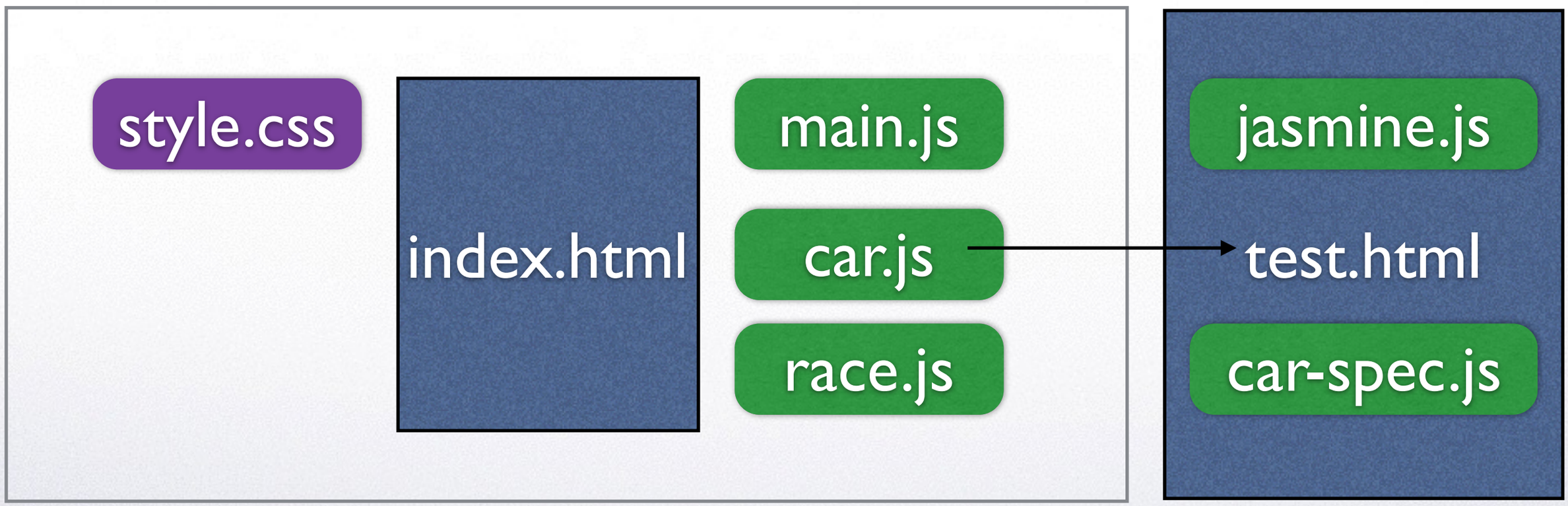
PhantomJS

QUnit

Testing Architecture

Testing Architecture

Testing Architecture



Testing Architecture

Blanket	Chutzpa	Jasmine	TeamCity
Istanbul	Karma	Mocha	Jenkins
JSCoverage	js-test-driver	QUnit	Travis
	Selenium	CasperJS	

Unit Testing Framework

- Provides structure to test code
- Easy to report failures
- Written in JavaScript

Unit Testing Framework

```
describe('Array', function(){
  describe('#indexOf()', function(){
    it('should return -1 when not present', function(){
      [1,2,3].indexOf(5).should.equal(-1);
      [1,2,3].indexOf(0).should.equal(-1);
    })
  })
})
```

Unit Test Report

passes: 2 failures: 0 duration: 0.07s 100%

Array

#indexOf

✓ should return -1 if element not found

✓ should return the index if element is found

Available Frameworks

- Mocha
- Jasmine
- QUnit
- http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#JavaScript

System Testing Tools

- Control an automatic browser
- Can use any programming language

System Testing

```
require "selenium-webdriver"

driver = Selenium::WebDriver.for :firefox
driver.navigate.to "http://duckduckgo.com"

element = driver.find_element(:name, 'q')
element.send_keys "Hello WebDriver!"
element.submit

puts driver.title

driver.quit
```

System Testing Tool

- We'll use selenium
- It's stable and flexible
- Really easy to automate

Test Runners

- Run tests, report errors
- Used from command line or IDE
- Available options:
  - Karma, Chutzpa, js-test-driver

Continuous Integration

- Checkout latest from source control
- Run all tests
- Report failures

Continuous Integration

- Travis
- Jenkins
- TeamCity

Code Coverage

- % of the code being tested
- 70-80% is good
- Istanbul is the tool

Coverage Report

File	Statements		Branches		Functions		Lines	
istanbul/	100.00%	(3 / 3)	100.00%	(0 / 0)	100.00%	(0 / 0)	100.00%	(3 / 3)
istanbul/lib/	97.48%	(425 / 436)	89.76%	(184 / 205)	97.98%	(97 / 99)	98.12%	(417 / 425)
istanbul/lib/command/	94.63%	(194 / 205)	87.88%	(58 / 66)	89.58%	(43 / 48)	96.43%	(189 / 196)
istanbul/lib/command/common/	90.91%	(60 / 66)	84.85%	(28 / 33)	100.00%	(6 / 6)	92.31%	(60 / 65)
istanbul/lib/report/	94.78%	(436 / 460)	85.71%	(144 / 168)	100.00%	(94 / 94)	95.32%	(428 / 449)
istanbul/lib/store/	100.00%	(73 / 73)	85.71%	(12 / 14)	100.00%	(28 / 28)	100.00%	(72 / 72)
istanbul/lib/util/	97.54%	(278 / 285)	95.20%	(119 / 125)	100.00%	(68 / 68)	98.16%	(267 / 272)

Balancing Tests

- Few system / scenario tests
- Many unit tests

Resources

- Chuzpa screencast for VS2012: http://www.youtube.com/watch?v=meJ94rAN7P8
- Miško Hevery on Unit Tests (google tech talks): http://www.youtube.com/watch?v=wEhu57pih5w

What Next

- Write unit tests
- Write functional tests
- Deploy a CI
- Set up coverage and test reports

Q & A

Hello Jasmine

Installing and running tests

Agenda

- JS Unit Testing
- A first test
- Running tests with Karma
- IDE integration

Getting Ready To Test

- JS Unit tests (try) make sure our JS code works well

Project Tree

```
index.html

- src
  - main.js
  - buttons.js
  - player.js

- style
  - master.css
  - home.css
```

Project Tree

```
index.html
test.html

- src
  - main.js
  - buttons.js
  - player.js

- style
  - master.css
  - home.css

- spec
  - button-spec.js
  - player-spec.js
```

Testing How

Testing Libraries

- Let's try to write test program for Array
- Verify indexOf(...) actually works

Array#indexof

```
var arr1 = [10, 20, 30, 40];

if ( arr1.indexOf(20) === 1 ) {
  console.log('success!');
} else {
  console.log('error');
}
```

What Went Wrong

- Hard to debug
- Hard to run automatically

We Need . . .

```
Jasmine  2.4.1                                   finished in 0.015s

● ● ●

3 specs, 0 failures                              raise exceptions ☐

  Array Spec

    #indexOf
      should return -1 when value is not there
      should return the index when value is there

    #pop
      should remove and return the last element
```

Hello Jasmine

```
describe('Array Spec', function() {
  describe('#indexOf', function() {
    it('should return -1 when value is not there', function() {
      const arr = [10, 20, 30];
      const idx = arr.indexOf(50);

      expect(idx).toEqual(-1);
    });
  });
});
```

Hello Jasmine

- describe() defines a block
- it() defines functionality

Matchers

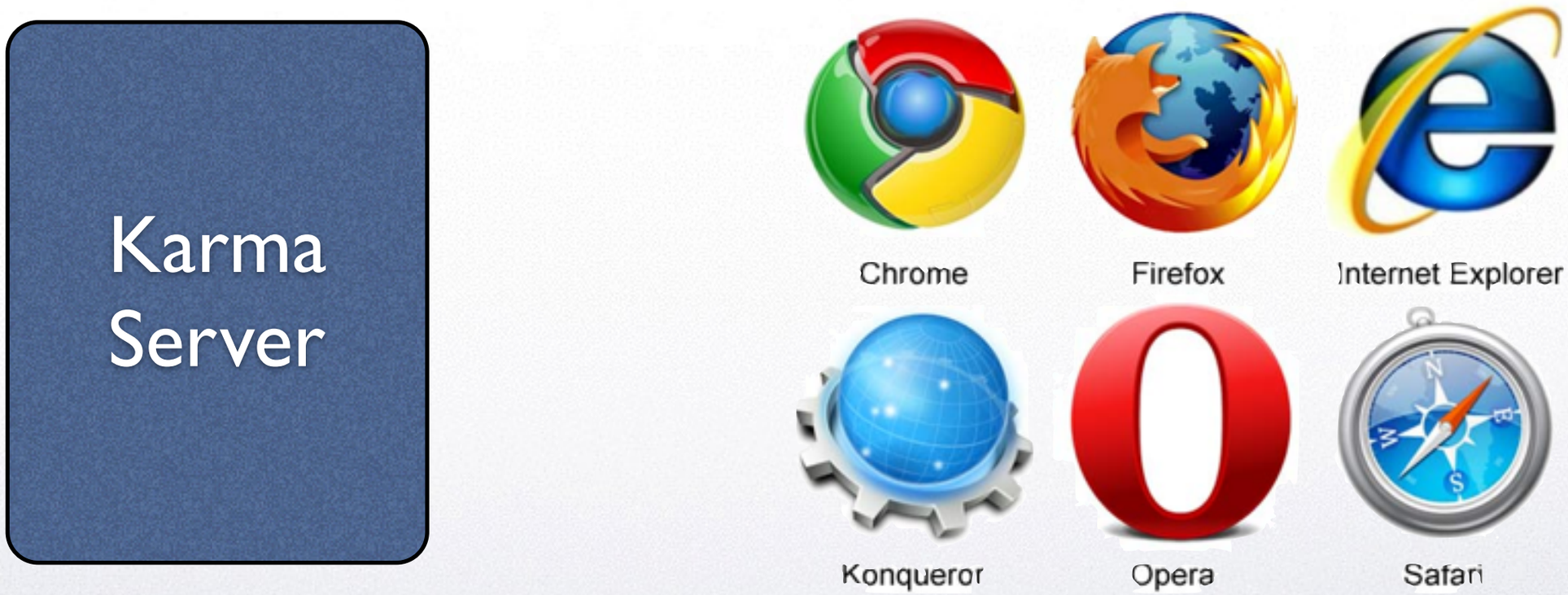
- Each expect(…) returns an object that has expectations
- Expectations are checked using matchers
- Jasmine matchers: https://github.com/jasmine/jasmine/tree/master/src/core/matchers

Running Our Test: Karma

Meet Karma

- A test runner for JS
- Integrates with many IDEs
- Integrates with CI servers

Karma Architecture



Karma Getting Started

```
# run just once to install
npm install karma -g

# create a project directory
mkdir myproject
cd myproject

# create karma configuration file
karma init
```

Karma Config

- Just a JavaScript file
- keys determine how test should run

Karma Config

- files is a list of JS files to include in the test
- Can use wildcards

Karma Config

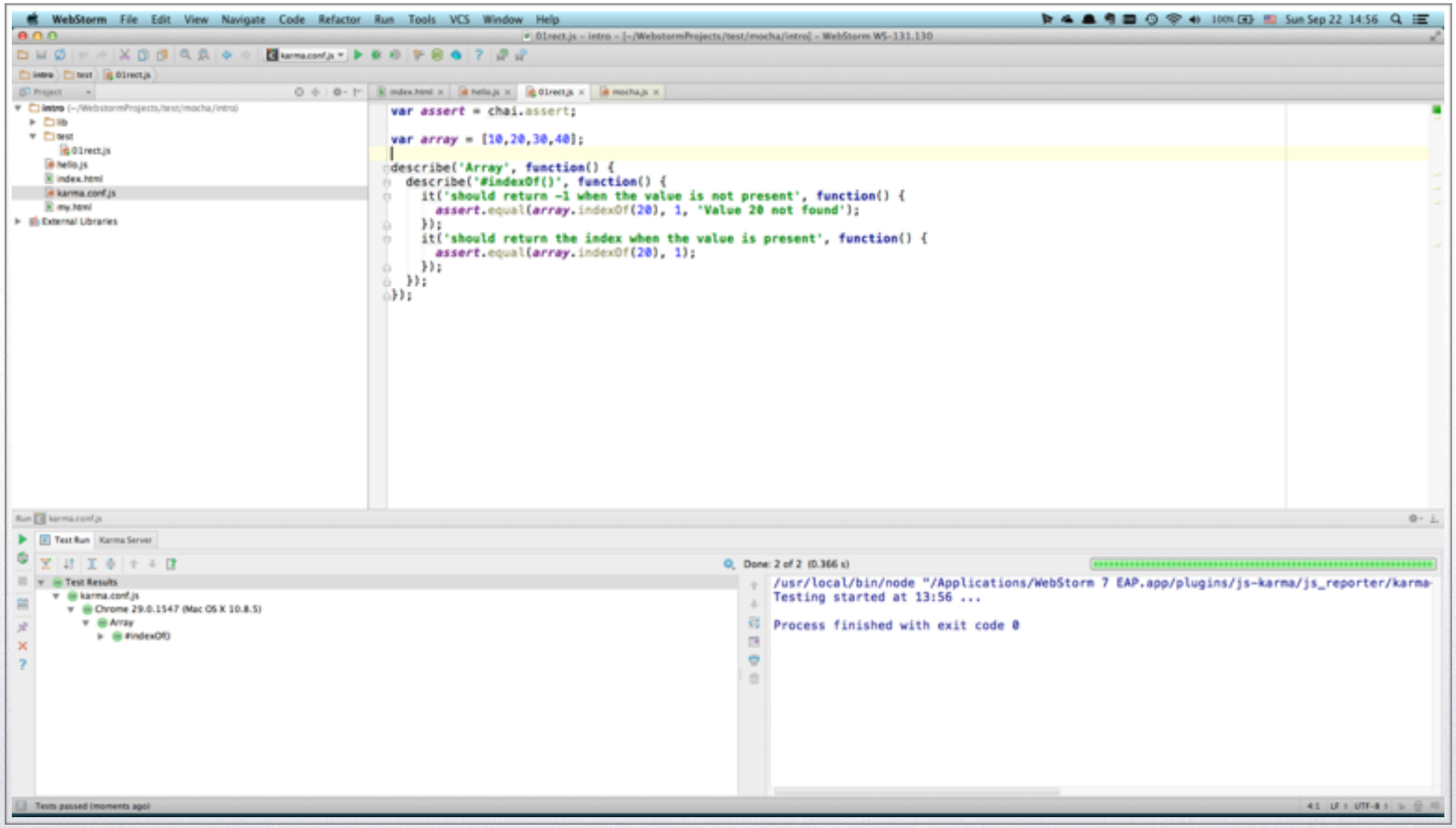
- browsers is a list of supported browsers

Running Tests

```
# start a karma server
karma start

# execute tests
karma run
```

IDE Integration



What We Learned

- Jasmine is a JS library that helps us write unit tests
- Karma is a JS library that helps us run them

Q & A

Advanced Jasmine

How to write awesome tests

Agenda

- Flow control: before, after, beforeEach, afterEach
- Writing async tests
- Fixtures and DOM testing

Let's Flow

```
describe('Test 1', function() {
  it('should do X', function() {
    var p1 = new Player('bob');
    var p2 = new Player('John');
    var game = new GameEngine(p1, p2);

    // test stuff with game
  });

  it('should do Y', function() {
    var p1 = new Player('bob');
    var p2 = new Player('John');
    var game = new GameEngine(p1, p2);

    // test stuff with game
  });
});
```

Let's Flow

```
describe('Test 1', function() {
  it('should do X', function() {
    var p1 = new Player('bob');
    var p2 = new Player('John');
    var game = new GameEngine(p1, p2);

    // test stuff with game
  });

  it('should do Y', function() {
    var p1 = new Player('bob');
    var p2 = new Player('John');
    var game = new GameEngine(p1, p2);

    // test stuff with game
  });
});
```

Same code...

A Better Scheme

- `beforeEach()` runs before each test
- also has:
  - `afterEach()` for cleanups
  - `before()` and `after()` run once in the suite

```
describe('Test 1', function() {
  var game;

  beforeEach(function() {
    var p1 = new Player('bob');
    var p2 = new Player('John');
    game = new GameEngine(p1, p2);
  });

  it('should do X', function() {
    // test stuff with game
  });


  it('should do Y', function() {
    // test stuff with game
  });
});
```

Async Testing

Async Theory

var x = 10

test x

x has the right value
testing here is OK

Async Theory

$.get(...)

test result

Can’t test now,
result not yet ready

Async Theory

- Async calls take callbacks
- We should tell mocha to wait

Async Code

```
describe('Test 1', function() {

  it('should do wait', function(done) {
    setTimeout(function() {
      // now we can test result
      assert(true);
      done();
    }, 1000);
  });

});
```

Async Code

Taking a function argument tells mocha the test will only end after it's called

```
describe('Test 1', function() {

  it('should do wait', function(done) {
    setTimeout(function() {
      // now we can test result
      assert(true);
      done();
    }, 1000);
  });

});
```

Async Code

```
describe('Test 1', function() {

  it('should do wait', function(done) {
    setTimeout(function() {
      // now we can test result
      assert(true);
      done();
    }, 1000);
  });

});
```

Calling the callback ends the test

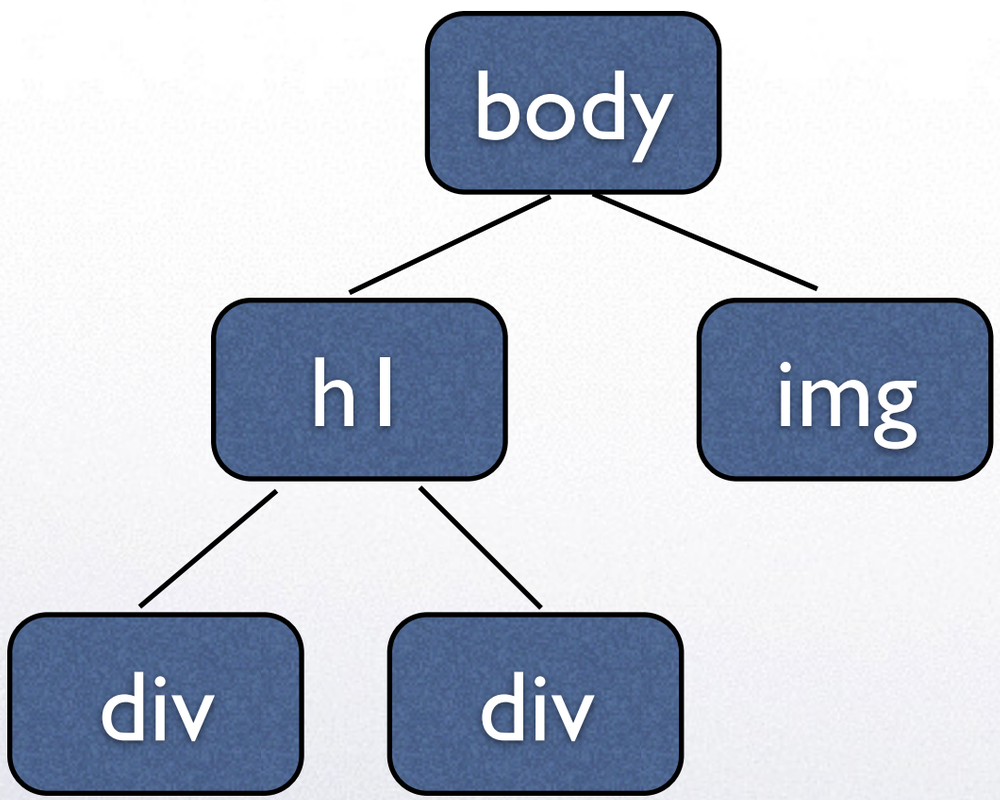
Same goes for Ajax

```
describe('Test 1', function() {
  it('should get user photo', function(done) {
    $.get('profile.png', function(data) {
      // run tests on data
      done();
    });
  });
});
```

DOM Testing

Theory

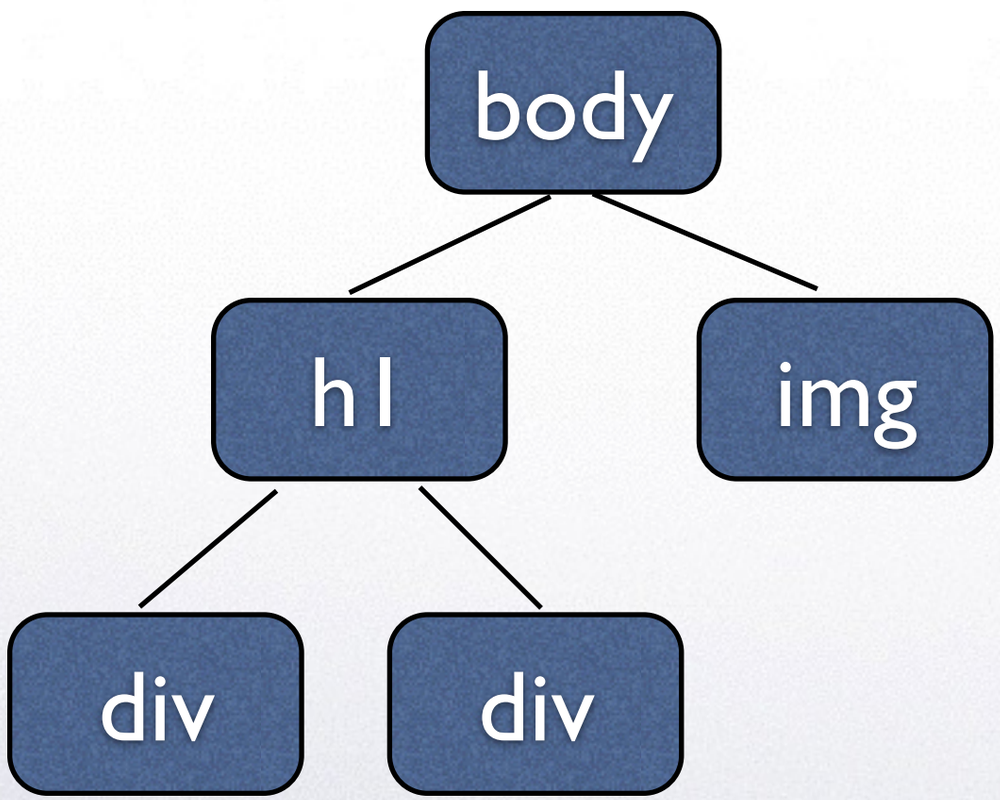
"Real" HTML



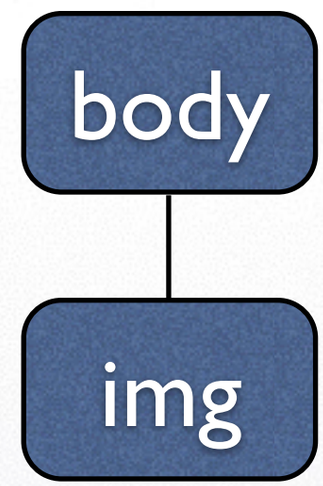
"Test" HTML

Theory

"Real" HTML



"Test" HTML



Theory

images.js

```
$('img.thumbnail').css({
    width: 200,
    height: 200
});
```

fixture.html

```
<img class="thumbmail" src="home.png" />
```

Using Fixtures

```
before(function() {
  fixture_el = document.createElement('div');
  fixture_el.id = "fixture";

  document.body.appendChild(fixture_el);
});

beforeEach(function() {
  fixture_el.innerHTML = window.__html__["fixture.html"];
});
```

Almost Ready

- HTML files are not served by default
- We need to tell karma to serve it

Serving HTMLs

- Modify files section to include the last (HTML) pattern

```
// list of files / patterns to load in the browser
files: [
  'lib/**/*.js',
  'plugins/**/*.js',
  'test/fixtures/*.html',
  'spec/*.js'
],
```

Testing a jQuery Plugin

```
it('should change the header text lowercase', function() {
  $('.truncate').succinct({ size: 100 });

  var result = $('.truncate').text();
  assert.equal( result.length , 100 );
});
```

Fixtures & DOM

- Define DOM fragments in HTML files
- Load from test suite
- Test and clean up

jasmine-jquery

- A nicer library for DOM testing
- https://github.com/velesin/jasmine-jquery

jasmine-jquery fixtures

- setFixtures(html) - to create a fixture

jasmine-jquery matchers

- toBeHidden(), toBeVisible(), toBeEmpty()
- toBeInDOM()
- toBeMatchedBy(selector)
- toContainElement(selector)
- toContainHtml(HTML String)
- toContainText(text)

jasmine-jquery matchers

- toHaveAttr(attr, value)
- toHaveClass(className)
- toHaveCss(css object)
- toHaveLength(len)
- toHandleWith(event, fn)

Demo

- Write a click counter component
- Check everything works with jasmine

Q & A

Spying With Jasmine

Stubs, Spies and Mock Objects explained

Agenda

- Reasons to mock
- Vanilla mocking
- How sinon can help
- Stubs and Spies
- Faking timers
- Faking the server

Reasons To Mock

Reasons To Mock

$.ajax

PersonalData

setTimeout

Reasons To Mock

Let’s Meet Some Spies

Spies

- A spy is a function that provides the test code with info about how it was used

Spies Demo

```
describe('Spies', function() {

  it('should keep count', function() {

    var s = jasmine.createSpy();
    s();
    expect(s).toHaveBeenCalled();

    s();
    expect(s.calls.count()).toEqual(2);

    s();
    expect(s.calls.count()).toEqual(3);
  });
});
```

Spy On Existing Funcs

```
describe('Spies', function() {

  it('spy on existing things', function() {
    const spy = spyOn(localStorage, 'setItem');
    localStorage.setItem('foo', 10);

    expect(spy).toHaveBeenCalledWith('foo', 10);
  });
});
```

Spy + Action = Stub

Stub Demo

- We'll replace $.ajax with a stub
- That stub always fails

Stub Demo

```
describe('Spies', function() {

  it('spy on existing things', function() {
    const ajaxStub = spyOn($, 'ajax').and.callFake(function(params) {
      params.error();
    });
    const failSpy = jasmine.createSpy();

    $.ajax({
      url: '/getData',
      success: function() {  },
      error: failSpy,
    });

    expect(failSpy).toHaveBeenCalled();
  });
});
```

Other Stubs

```
spyOn(...).and.returnValue(10)
spyOn(...).and.returnValues(10, 20, 30)
spyOn(...).and.callThrough()
```

- Full List: https://github.com/jasmine/jasmine/blob/master/src/core/SpyStrategy.js

Q & A

Fake Timers

- Use `jasmine.clock().install()` to create a fake timer
- Use `jasmine.clock().uninstall` to clear fake timers

Fake Timers

- Use `tick(...)` to advance
- Affected methods:
  - setTimeout, setInterval, clearTimeout, clearInterval
  - Date constructor

Fake Servers

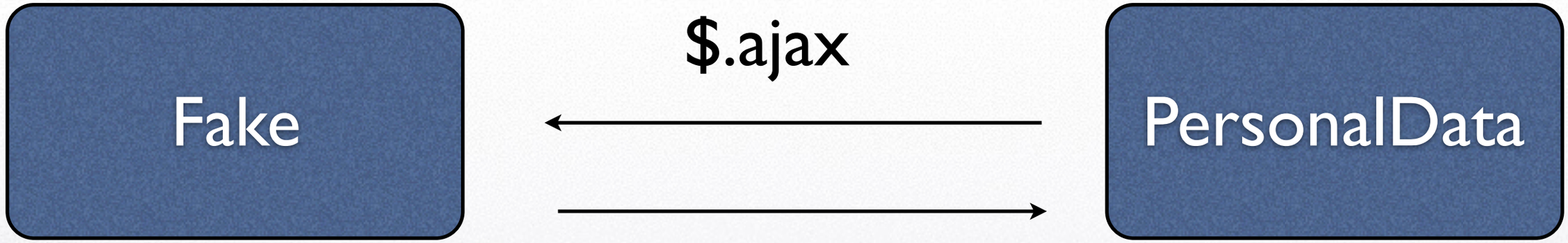
- Testing client/server communication is hard
- Use fake servers to simplify it

Fake Servers

$.ajax

PersonalData

Fake Servers



Let's write a test for the following class

```
import $ from 'jquery';

export default class Person {
  constructor(id) {
    this.id = id;
  }

  load() {
    $.get(`/users/${id}`, ((info) => {
      this.name = info.name;
      this.favoriteColor = info.favorite_color;
    }));
  }
}
```

Testing Plan

- Set-up a fake server
- Create a new Person
- call load()
- verify fields data

Setting Up The Server

```
beforeEach(function() {
  jasmine.Ajax.install();
});

afterEach(function() {
  jasmine.Ajax.uninstall();
});
```

Fake Load

```
it('should call GET with correct URL', function() {
  const p = new Person(1);
  p.load();

  const req = jasmine.Ajax.requests.mostRecent();

  expect(req.url).toEqual('/users/1');
});
```

Fake Response

```
it('should set fields according to given info', function() {
  const p = new Person(1);
  p.load();

  const req = jasmine.Ajax.requests.mostRecent();
  req.respondWith(TestResponse.person.success);

  expect(p.name).toEqual('bob');
  expect(p.favoriteColor).toEqual('blue');
});
```

TestResponse Structure

```
const TestResponse = {
  person: {
    success: {
      status: 200,
      responseText: JSON.stringify(
       { name: 'bob', favorite_color: 'blue' }),
      contentType: 'application/json',
    },
  },
};
```

Wrapping Up

Wrapping Up

- Unit tests work best in isolation
- Sinon will help you isolate units, by faking their dependencies

Wrapping Up

- Write many tests
- Each test verifies a small chunk of code
- Don’t test everything

Thanks For Listening

- Ynon Perek
- http://ynonperek.com
- ynon@ynonperek.com